CERN School of Computing

inverted CSC-2006

"Where students turn into teachers"

6-8 March 2006, CERN*

- ▶ Computational Intelligence for HEP Data Analysis
- ▶ The Art of Designing Parallel Applications
- ▶ Software Testing: Fundamentals and Best Practices

Lecturers - all former CSC2005 students

Marek Biskup	Warsaw University
Jaroslaw Przybyszewski	Warsaw University of Technology
Vijayalakshmi Sundararajan	NewCastle
Liliana Teodorescu	Brunel University
Anselm Vossen	Albert-Ludwigs-Universität
Yushu Yao	University of Alberta

- a novel idea already prototyped in 2005
- a series of lectures proposed and delivered by selected students
- advanced topics, rarely taught at CERN before

http://cern.ch/csc

* Room **40-SS-D01**, building 40
Free attendance but registration recommended

CERN School of Computing

Welcome to **iCSC2006**, the second edition of the Inverted School, "*Where Students turn into Teachers*".

iCSC is a novel idea t hat was experimented for the first time in 2005.

The idea comes from the observation that at regular CSCs, the sum of the students' knowledge often exceeds that of the lecturer, and that it is common to find someone in the room who knows more on a particular topic than the lecturer. So why not to try and exploit this?

CSC2005 students made proposals via an electronic discussion forum. The best proposals were selected and their authors appointed as theme coordinators. From this point on, they were on their own to design the content.

As for the first edition, this year's programme contains several novel, sometimes challenging topics, and rarely taught a CERN: *Computational Intelligence for HEP Data Analysis, The Art of Designing Parallel Applications , Software Testing: Fundamentals and Best Practices.*

I would like to thank to all those who developed proposals and to those actually lecturing. This is their school and I am confident all will go very well. As this is only the second edition, do not hesitate to comment and advise us on how to improve it.

Enjoy the school.

François Fluckiger
Director of the CERN School of Computing

Programme overview

The programme is formed of three themes, selected from proposals developed by students via an electronic forum.

	Computational Intelligence for HEP Data Analysis	The Art of Designing Parallel Applications	Software Testing: Fundamentals and Best Practices
Theme Coordinator	**Liliana Teodorescu** Brunel University **Anselm Vossen** Albert-Ludwigs-Universität	**Marek Biskup** Warsaw University	**Vijayalakshmi Sundararajan** NewCastle
Key words	• Statistical Learning • Attribute Selection • Optimal Classification: Bayes theorem • Classification and Grouping Algorithms • Structural Risk Minimization • Kernel Methods • Support Vector Machines • Neutral Networks	• Parallel Architectures • Parallel Applications • Distributed Programming • Synchronization of Parallel Processes • Design Patterns in Concurrent Programming • Portable Programming • The ROOT Framework • PROOF - Parallel Analysis with ROOT	• Principles of Testing • Testing throughout the Life-cycle • Dynamic Testing Techniques • Static Testing Techniques • Test Management • Tool Support for Testing
Lecturers	**Liliana Teodorescu** Brunel University **Anselm Vossen** Albert-Ludwigs-Universität **Jaroslaw Przybyszewski** Warsaw University of Technology	**Marek Biskup** Warsaw University **Yushu Yao** University of Alberta	**Vijayalakshmi Sundararajan** NewCastle
When	**Monday** 6 March 2006 09:00 - 17:30 40-SS-D01	**Tuesday** 7 March 2006 09:00 - 16:30 40-SS-D01	**Wednesday** 8 March 2006 09:00 - 11:00 40-SS-D01

iCSC2006 Schedule

	Monday 6 March 2006 40-SS-D01	Tuesday 7 March 2006 40-SS-D01	Wednesday 8 March 2006 40-SS-D01
Theme	**Computational Intelligence for HEP Data Analysis: Status and Recent Advances**	**The Art of Designing Parallel Applications**	**Software Testing: Fundamentals and Best Practices**
Theme Coord.	Liliana Teodorescu Brunel University **Anselm Vossen** Albert-Ludwigs-Universiteit	**Marek Biskup** Warsaw University	**Vijayalakshmi Sundararajan** **Newcastle**
09:00 - 09:55	09:00 **Registration** 09:30 School opening Theme presentations	**Parallel Computing** Marek Biskup	**Testing Process and Management** Vijayalakshmi Sundararajanl
10:00- 11:00	**Introduction** Liliana Teodorescu **Feature Selection and Statistical Learning Basics** Anselm Vossen	**Synchronizing Threads and Processes** Marek Biskup	**Testing Techniques and Tools** Vijayalakshmi Sundararajanl
11:00 - 11:30	Coffee Break	Coffee Break	Coffee and Adjourn
11:30 - 12:25	**Basic Machine Learning Algorithms** Jaroslaw Prybyszewski	**Design Patterns for Concurrent Objects** Marek Biskup	
12:30 - 14:00	Lunch	Lunch	
14:00 - 14:55	**Neural Networks** Liliana Teodorescu	**Portable Programming** Yushu Yao	
15:05 - 16:00	**Support Vector Machines** Anselm Vossen	**Parallel computing with ROOT and PROOF** Marek Biskup Yushu Yao	
16:00 - 16:30	Coffee Break	Coffee and Adjourn	
16:30 - 17:25	**Evolutionary Computation** Liliana Teodorescu		
17:30 - 18:30			
18:00 - 19:30		Cocktail (all participants invited) Restaurant 1 (*)	

(*): To be confirmed

iCSC 2006 Lecturer Biographies

Marek Biskup **Warsaw University, Warsaw Poland** iCSC

Marek is a PhD student at Institute of Informatics, Warsaw University. Currently he is involved in the "Pi of the Sky" astronomical experiment, whose goal is online monitoring of the sky, looking for optical flashes. He stayed at CERN as a technical student working with the ROOT team on improving transparency of the subsystem for parallel data analysis - PROOF. He received Master's degree in Computer Science from Warsaw University, Warsaw and Vrije Universiteit, Amsterdam. The research for his Master's project included development of a parallel grid application for spectral data analysis (biophysics). Apart from his Computer Science studies Marek is finishing his Master's Degree in Experimental High Energy Physics. His research interests include parallel programming and distributed systems, computing for physics as well as theoretical computer science.

Jaroslaw Przybyszewski Warsaw University of Technology, Warsaw Poland iCSC

I am a MSc. student at Warsaw University of Technology, Faculty of Electronics and Information Technology. I stayed at CERN from October 2004 until September 2005 as a technical student. I was a member of DES-DIS group and worked with Oracle databases backups. My thesis is related to the area of my work at CERN - I am developing an application that will support development of backup strategies for Oracle databases. Apart from Oracle I am interested in machine learning, evolutionary, heuristic and data mining algorithms.

Liliana Teodorescu **Universty of Brunel United Kingdom** iCSC

Liliana Teodorescu is a Lecturer at Brunel University, UK. She holds a Ph.D. in Particle Physics from Bucharest University, Romania. Since her graduation she worked on particle physics experiments at different laboratories around the world: Thomas Jefferson National Accelerator Facility (TJNAF), USA, Instituto Nazionale di Fisica Nucleare (INFN) – Pisa, Italy and Stanford Linear Accelerator Centre (SLAC), USA. She is currently working on CMS and BaBar experiments, being particularly interested in development of new algorithms for data analysis.

Vijayalakshmi Sundararajan — **Newcastle, United Kingdom** — **iCSC**

Vijayalakshmi Sundararajan is currently a BCS-ISEB certified Test Analyst based in NewCastle, UK. She has been actively working in the IT industry since 1999. She worked for one of India's largest software company and gained expertise in software testing while serving clients like Morgan Stanley and Lucent Technologies. In 2005, she collaborated through Université Libre de Bruxelles, Belgium with the CMS computing team at CERN to develop the infrastructure for automatic software validation.

Anselm Vossen — **Albert-Ludwigs Universität , Freiburg Germany** — **iCSC**

I am currently working on my doctoral degree in physics at the Albert-Ludwigs Universität in Freiburg, Germany after receiving my diploma in computer science at the same University. Our group participates in the COMPASS experiment at CERN. The experiments main goal is to investigate the structure of the nucleon. There I am involved in the physics analysis and the development of intelligent tracking algorithms. My interests include statistical learning and its application in HEP.

Yushu Yao — **University of Alberta Canada** — **iCSC**

I am a PhD student at University of Alberta, Canada. Working on Atlas, the "computing part" is on the simulation of LUCID detector and the Monte Carlo study of some physics processes. Although I'm still a newcomer to HEP, I have been playing with computers since I was 8 years old. Starting with Basic and DOS, I came across quite a few OS and several languages, and started programming with C in junior high school. Programming is one of my most favorite exercises (both mental and physical). I've get a lot of stories to share with everyone. However, the more I learn, the more I realize my weaknesses. I'm looking forward for the chance to learn from all of you.

Computational Intelligence for HEP Data Analysis

iCSC2006 Computational Intelligence for HEP Data Analysis: Status and Recent Advances

Coordinators:

Liliana Teodorescu - Brunel University

Anselm Vossen - Albert-Ludwigs-Universiteit

This lecture series will provide an introduction to advanced data analysis techniques developed in computer science and their current or potential applications in High Energy Physics.

The lectures will cover basic topics of Machine Learning Techniques and detailed discussions of how to prepare data, how to extract meaningful attributes, and how to evaluate algorithms performance.

Artificial Neural Networks and their current status in High Energy Physics will be presented as an example of a technique becoming more and more common in this field. Other state of the art algorithms such as Support Vector Machines and new developments in Evolutionary Computation will also be introduced and their prospects in High Energy Physics will be discussed.

A few questions

- *Are there new prospects for **Machine Learning** Techniques in High Energy Physics?*
- *Have you ever heard of **Support Vector Machines** or **Gene Expression Programming**? Are they used in High Energy Physics?*
- *Do you know the current status of the **Artificial Neural Networks** applications in High Energy Physics?*
- *Are you aware of the state of the art algorithms for **classification**, **regression** or **optimisation** problems?*
- *Do you know how to evaluate the **performance** of such algorithms?*

All the answers in the Computational Intelligence at **iCSC**

Overview

Slot	Lecture	Description	Lecturer
Monday 6 March 2006			
10:00 - 10:30		**Introduction**	Liliana Teodorescu
10:30 - 11:00	Lecture 1	**Feature Selection and Statistical Learning Basics**	Anselm Vossen
11:30 - 12:25	Lecture 2	**Basic Machine Learning Algorithms**	Jaroslaw Prybyszewski
12:30 - 14:00		Lunch	
14:00 - 14:55	Lecture 3	**Neural Networks**	Liliana Teodorescu
15:05 - 16:00	Lecture 4	**Support Vector Machines**	Anselm Vossen
16:30 - 17:25	Lecture 5	**Evolutionary Computation**	Liliana Teodorescu
17:30		Adjourn	

LECTURE 1

Monday 6 March 2006			
10:00 - 10:30		**Introduction** This introduction will present a brief overview of the main challenges of the Data Analysis in HEP and how Computational Intelligence methods can help in addressing these challenges. It will provide the general background that will allow the audience to put in the context the specific information presented in the lectures of the series. The introduction together with the whole series of lectures target both particle physicists and computer scientists. They are meant as an inspiration and encouragement for particle physicists to explore new algorithms and as an invitation for computer scientists to propose other powerful algorithms developed in their field.	Liliana Teodorescu

Feature Selection and Statistical Learning Basics

10:30 - 11:00	Lecture 1	**Feature Selection and Statistical Learning Basics** Selecting meaningful features is as important for the success of a classification or regression task as the intelligent system used for the actual work. This lecture shows you important aspects in feature selection and introduces the basics of classification with the Bayesian theorem. This is done while walking through the data analysis process, from the initial features to the classification decision. The lecture targets physicists and computer scientists alike and lays some of the groundwork for the following lectures. **Motivation** - The Data Analysis Process - Why Attribute Selection and Data Preparation is important **Strategies for Feature Selection** - Reducing the Dimensionality of the feature space with the principal Component Analysis (Karhunen-Loeve Transformation) - Measuring the Importance of a Feature with Information Gain and Significance Measures1 **Statistical Learning Basics** - Optimal classification: Bayes theorem - Bayes Classifier - Evaluating the performance of a classifier	Anselm Vossen

Feature Selection and Statistical Learning Basics

Anselm Vossen

Universität Freiburg

Goal of this lecture

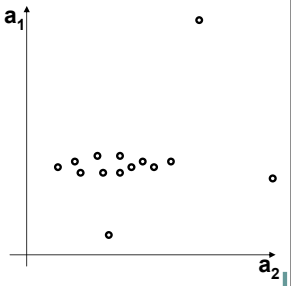
- **Show how to go through a classification task**
- **Common pitfalls in data preparation and selection of appropriate classifiers**
- **Emphasis on feature selection**
- **Introducing feature selection by**
  - Principal component analysis
  - Information gain and significance measures
- **Introduction to simple classification schemes**
  - Maximum a posteriori classification
  - Maximum likelihood classifier

Example

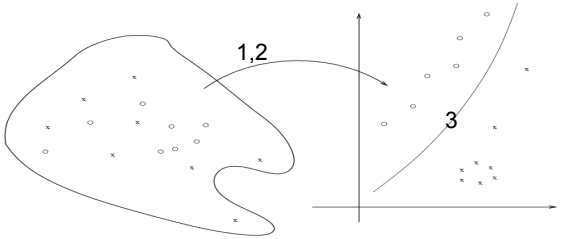
- **Particle identification: Suppose we want to estimate, if a given event is signal or background. The information that you have include**
  - Missing energy
  - Leading particle momentum
  - Track length
  - Output of another algorithm
  - …
- Naïve: look for a function that maps our features to the correct answer
- How do accomplish this task?

Data Preparation

- **Each Feature is a random variable a_i**
- **E.g. a_1: Missing energy…**
- **Feature vector: $v=[a_1,a_2,\ldots,a_n]$**
- **First step: data preparation**
  - Remove outliers
  - Scale data for numerical stability and to give each feature same weight
    - E.g. to [-1,1] or same mean and variance
    - Do not forget to scale your test and training data with the same factors



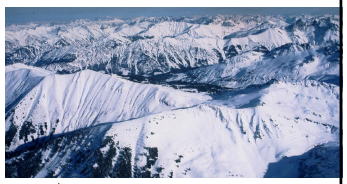
Data Analysis Process

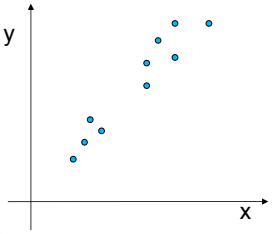


- **1: Preprocessing: Data preparation,**
- **2 Feature selection**
- **3: Classification (regression, clustering, mining)**

Motivation Feature Selection

- **Dimension of the feature space is determined by the number of features**
- **Data analysis consists of finding structures in the feature space or finding borders between classes in it**
- **Preprocessing can facilitate this task a lot!**
  - Finding the deepest valley in the alps is hard
  - Dimensional reduction
  - Decorrelation

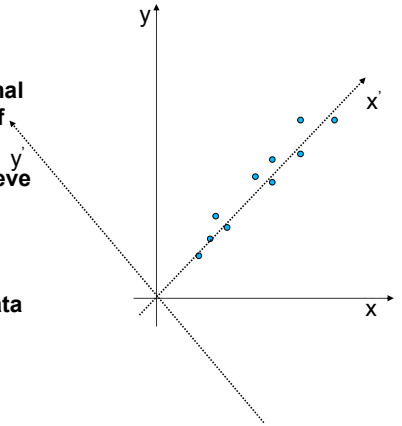




Principal Component Analysis

Remove dimension with minimal error

- **Take directions of maximal variance, eigenvectors of correlation matrix (CM)**
- **Algorithm: Karhunen Loeve Transformation (KLT)**
- **Diagonalizes CM**
- **Precondition: Enough data to estimate CM**



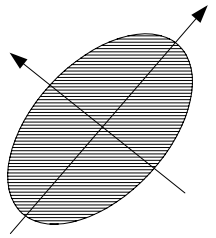
Diagonalizing the Correlation Matrix

- **Feature vector** $\vec{v} = [a_1, a_2, ..., a_n]$
- **The correlation matrix (CM):**

$$R_{vv} = E\{vv^T\}$$

- **The KLT is the unitary transform *A* with**

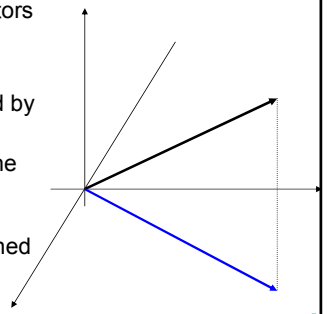
$$\vec{w} = A^T \vec{v}$$



$$R_{ww} = E\{\vec{w}\vec{w}^T\} = E\{A^T \vec{x}\vec{x}^T A\} = A^T R_{xx} A = \Lambda = \mathrm{diag}(\lambda_1, ..., \lambda_n)$$

KLT, Continued

- **Properties:**
  - The matrix *A* consists of eigenvectors of $R_{vv} = E\{\vec{v}\vec{v}^T\}$
  - Expected error: $E\{\vec{v} - \vec{w}\} = \sum_{i=m+1}^{n} \lambda_i$
  - Projection onto subspace spanned by the first m vectors is optimal
  - Sum of eigenvalues (variance in the direction of the corresponding Eigenvector can be used as a measure of the information contained in the feature
  - Data dependent
  - No fast algorithm
  - Not optimized for class separability
    - Fishers linear discriminant



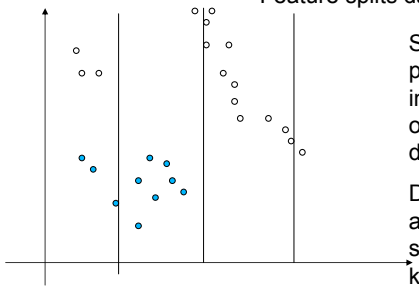
Other useful Transformations

- **For different tasks, different data representations might be useful, e.g.**
  - Fourier Transformation
  - Haar Transformation
    - Integration over superfluos degrees of freedom
  - …

Feature Selection for Classification

- **How much information does a feature provide for classification?**

Feature splits data in different sets



Select features that provide much information about the outcome of a random draw

Does the probability of a class change significantly if we know in which subset we are?

Feature Selection by Information Gain

- **Select features that contain as much information about the class as possible. Information of an Answer with possible outcomes v_i with Probabilities $P(v_i)$ is defined as**

$$\mathrm{Inf}(P(v_1),\ldots,P(v_n)) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$$

- **Reject features that are random (Noise)**
- **Possible measure: Information gain for Attribute B:**

$$\mathrm{Inf}(P(A)) - \mathrm{Inf}(P(A|B))$$

  - Difference in information that we need to predict the outcome
  - If the attribute is good, then the difference in information is big, because we do not need further information
  - Information is measured in bit, 1bit=information needed to answer a yes/no question

Information Gain Example

- **Consider toss of a fair coin**
- **Information from knowing the outcome:**

$$I\left(\frac{1}{2},\frac{1}{2}\right) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1\text{bit}$$

- **If we have the information that the outcome is 99% heads:**

$$I\left(\frac{1}{100},\frac{99}{100}\right) = 0.08\text{bit}$$

Information Gain, Cont

- **Generally, if a training set has p positive and n negative examples:**

$$I_{\text{Init}}\left(\frac{p}{p+n},\frac{n}{p+n}\right) = -\frac{-p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- **After we know attribute A that splits training set into v subsets:**

$$\text{Remainder}\quad (A) = -\frac{\sum_{i=1}^{v} p_i + n_i}{p+n} I\left(\frac{p_i}{p+n},\frac{n_i}{p+n}\right)$$

- **The Information gain is thus I_{Init} – Remainder(A)**
- **Information gain for feature selection is popular with decision trees, see lecture by Jarek**
- **Add hoc strategies are needed to avoid high branching attributes (like an ID field)**

Determining Significance by χ^2

- **Question: Is a given feature significant for a given classification?**
- **Null Hypothesis: The feature is irrelevant, compute χ^2 for this hypothesis (no significant derivation of PDF given feature from expected PDF)**
  - The feature splits the set of examples into I subsets
  - Expectation value for number of positive and negative examples: $\hat{p}_i = \frac{p\cdot(p_i+n_i)}{p+n}, \hat{n}_i = \frac{n_i\cdot(p_i+n_i)}{p+n}$
  - If the attribute is irrelevant, the deviation D should follow a χ^2 distribution:

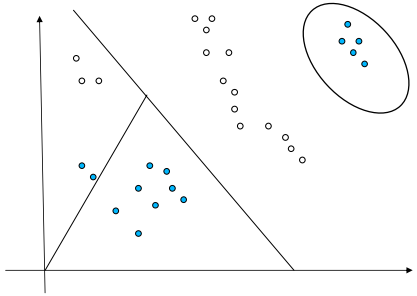
$$D = \frac{\sum_{i=1}^{v}(p_i - \hat{p}_i)}{\hat{p}_i} + \frac{\sum_{i=1}^{v}(n_i - \hat{n}_i)}{\hat{n}_i}$$

Feature Selection

- **For complex classification algorithms it is harder to determine optimal features, but greedy strategies often work well**
- **Popular strategy:**
  - Consider subsets of features, train, evaluate performance
- **Optimal set of features is algorithm dependent, exponential computing time (NP hard)**
- **Even when using complex algorithms, do**
  - Renormalization
  - Outlier Removal

Classification

- Given Evidence $\vec{v}$ classification is the mapping to a class ω_k (e.g. signal or background)
- Mapping divides feature space



Classification with Bayes-Classifier

Maximum a Posteriori (MAP)

- If we know the probability distribution *P*, then the answer is obtained by

$$\max_k \{ P(\omega_k|\vec{v}) \}$$ **(MAP Classifier)**

- Example: for given symptoms, what are the probabilities for certain illnesses

Bayes Theorem

A Posteriori Prob

A Priori Prob.

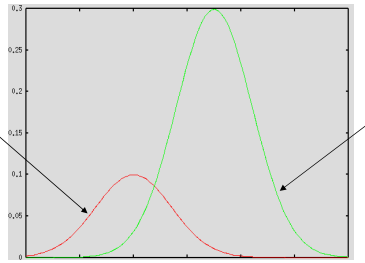
$$P(\omega_k|\vec{v}) = \frac{P(\vec{v}|\omega_k)P(\omega_k)}{P(\vec{v})}$$

- Example: Suppose that a test for an illness is positiv. For an ill person, the test is correct 98% of the time, but in 2% of all cases the test gives a positiv result. It is a rare illness, only 1 in 10 000 people have it. Should you be worried?
- Answer: The probablility that you have the illness is 98% * (1/10000)/2% = 0.5 %, so no reason to be worried

Bayes, Cont

$$\frac{P(\vec{v}|\omega_j)P(\omega_j)}{P(\vec{v})} \overset{?}{\geq} \frac{P(\vec{v}|\omega_k)P(\omega_k)}{P(\vec{v})}$$

$P(\vec{v}|\omega_j)P(\omega_j)$ $P(\vec{v}|\omega_k)P(\omega_k)$



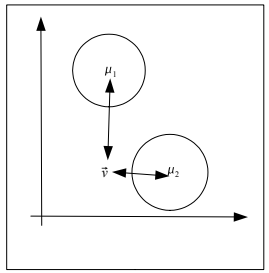
- **Assume Gaussians and compute probabilities**
- **Estimation of PDFs necessary!**

Bayes, Continued

- **Assuming**
  - uncorrelated features
  - Equal variance and $P(\omega_k) = P(\omega_j)$
- **Maximum Likelihood Estimator (MLE)**

$$P(\omega_k) \propto e^{\frac{(v_1-\mu_k^{\,1})^2}{\sigma}} \cdot \ldots \cdot e^{\frac{(v_1-\mu_k^{\,2})^2}{\sigma}} = e^{\frac{\|\vec{v}-\mu_k\|^2}{\sigma}}$$

$$P(\omega_k|\vec{v}) \overset{?}{\geq} P(\omega_j|\vec{v}) \Leftrightarrow \|\vec{v}-\mu_k\| \overset{?}{\geq} \|\vec{v}-\mu_j\|$$



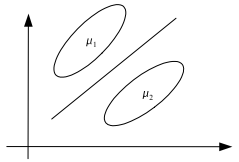
Probability can be measured by Euclidian distance

MAP Classificator

- **If we know the covariance matrix we can use:**

$$P(\omega_k) \propto e^{-(\vec{v}-\mu_k)K^{-1}(\vec{v}-\mu_k)^T}$$

$$\left(K = E\left\{(\vec{v}-\bar{v})(\vec{v}-\bar{v})^T\right\}\right)$$



- **Mahalanobis distance D_M:**

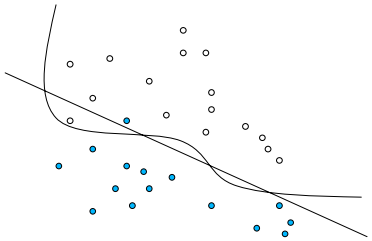
$$D_M^2(\vec{v},\vec{w}) = (\vec{v}-\vec{w})K^{-1}(\vec{v}-\vec{w})^T$$

- **and**

$$P(\omega_k|\vec{v}) \overset{?}{\geq} P(\omega_j|\vec{v}) \Leftrightarrow D_M(\vec{v},\mu_k) \overset{?}{\geq} D_M(\vec{v},\mu_j)$$

Overfitting

- **How complex should the model be?**



Overfitting

If the decision function is too complex and there is not enough evidence, statistical fluctuations are learned

good performance on training set, bad on independent set

- **Occams Razor: Always use the simplest theory that describes the data**
- **Bayes: a complex theory is more unlikely, so we need more evidence to support it**
- **Especially important when training and real data differ (e.g. in physics: MC)**

Overfitting, Continued

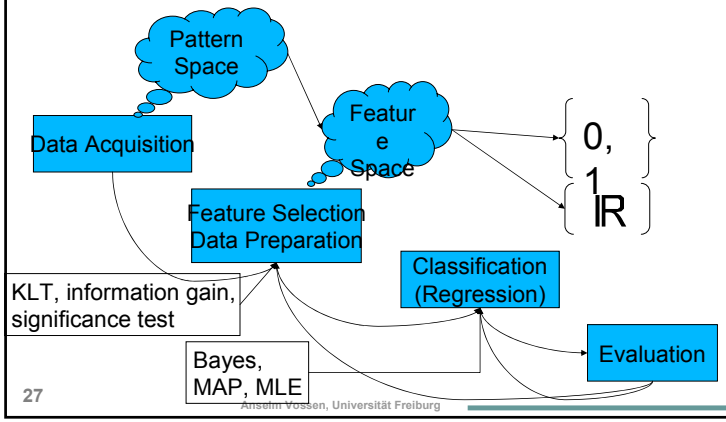
- **There exists an elaborate theory that connects the predictive power of a function and the training error to the expected error (by Vapnik and Chervonenkis)**
- **Complex Algorithms have to take precautions to avoid overtraining, e.g. pruning of decision trees, weight decay with NN**
- **Need a way to compute/control the „complexity“ of a predicting function**
  - Decision Trees use pruning
  - SVM algorithms have a parameter that controls the complexity
  - Neural Networks use ad-hoc techniques (e.g. weight decay)

Evaluating Performance

- **How to make predictions about the performance of the model that we selected (Expected Error)**
- **Straightforward: Split Data in test- and training set. This overestimates the error**
- **Additional parameter estimation 3 sets**
- **Better: n-fold Cross-Validation: Split data in n subsets, train on n-1 and test on the remaining data, do this n times.**
- **Extreme but best: leave-one-out (loo) error: use subsets of size 1**

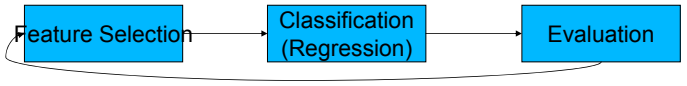
Conclusion

- **Data Analysis Process Not necessary a waterfall model…**



Advise

- **Get a feeling for the discrimating power of the attributes and the algorithms**
  - **e.g. use the Weka workbench to play around** (Weka Tool: Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.)



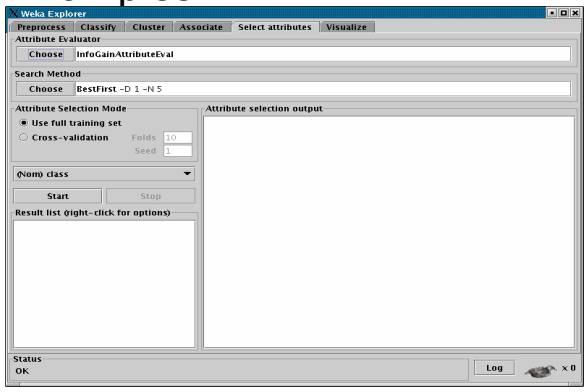
Further Reading

- [1] B. Schölkopf and A.J. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002
- S. Theodoridis and K. Koutroumbas, Pattern Recognition, Academic Press, 1999
- S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2003
- I. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations, Morgan Kaufmann, 1999

Questions?

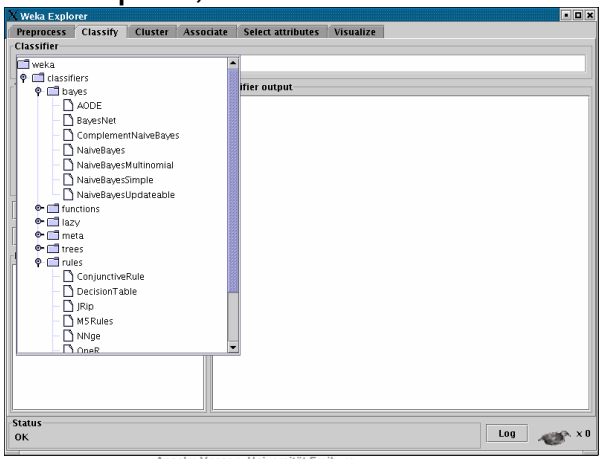
- **Thank you for your attention!**

Examples



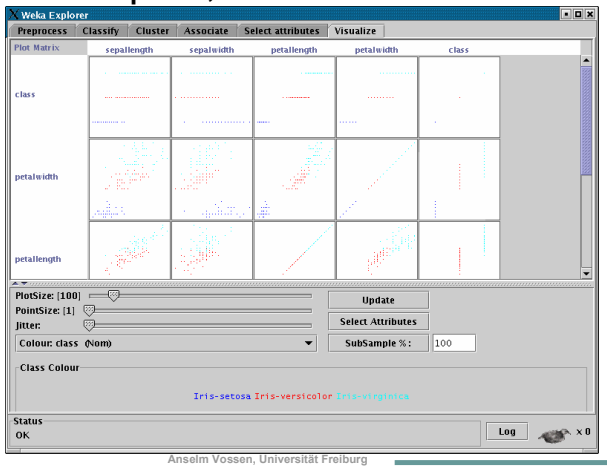
- **Weka Tool: Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.**

Examples, Cont



CERN School of Computing

Examples, Cont



LECTURE 2

Basic Machine Learning Algorithms

Monday 6 March 2006			
11:30 12:25	Lecture 2	**Basic Machine Learning Algorithms**	Jaroslaw Prybyszewski
		This lecture will present a set of fundamental Machine Learning algorithms and software tools for their easy application to data analysis. The lecture targets both physicists and computer scientists. It will rely on some of the background information presented in the Future Selection and Statistical Learning Basics lecture of this series. The lecture will address the following issues:	
		What are and how to build decision trees? Short description of the idea, method of choosing correct test, advantages and disadvantages of decision trees. **Random forest as a variation of decision tree algorithm** Why and when random forest are better than a single tree. What do we loose in comparison to decision trees? **Main ideas of "lazy learning"** Nearest neighbour algorithm and its generalization - kNN as examples of lazy algorithms. They provide very good results in classification with minimum effort. **Grouping algorithms - when to use them** Presentation of cobweb algorithm. **R-language - open source environment for development and testing** Some of the algorithms described in the presentation will be presented inR-environment.	

Machine Learning

Jarosław Przybyszewski

Warsaw University of Technology

Inverted CERN School of Computing, 6-8 March 2006

Game plan

- **Introduction**
- **Classification algorithms**
  - Decision trees
  - Random forest
  - "Lazy learning" – kNN algorithm
- **Grouping algorithms**
  - COBWEB example
- **Association algorithms**
- **R-language environment**

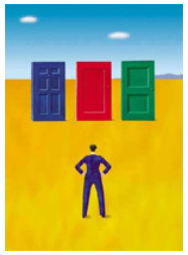
Introduction 1/2

- **Why do we use machine learning algorithms?**
  - Impossible to develop deterministic algorithms
  - Unstable environment
  - Large amount of data
  - Autonomous systems

Introduction 2/2

- **Group of algoritms**
  - Classification – assigning event to a class
  - Approximation – e.g. function approximation
  - Grouping – dividing something into groups
  - Association – finding sets that coexist together

Classification algorithms



Decision trees 1/5

- **Very popular**
- **Simple**
- **Effective**

Decision trees 2/5

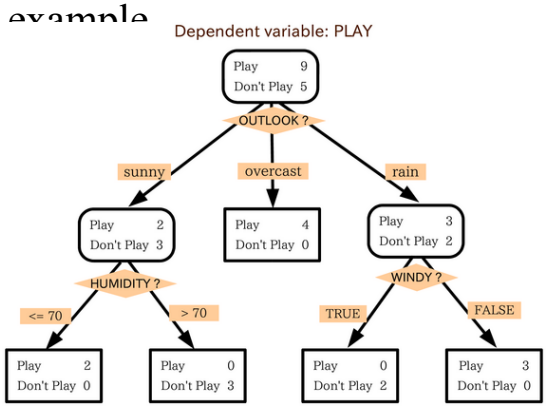
- **Every node represents a test**
- **Subtrees represent results of a test**
- **How to choose the right test?**
  - Entropy
  - Information gain
  - Gini index

Decision trees 3/5 – classic

Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

Decision trees 4/5 – classic example

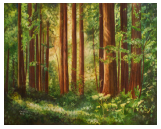


Decision trees 5/5

- **Advantages**
  - Easy to implement
  - Easy to interpret output
  - Handles nominal and categorical data
  - Perform well even with large data
- **Problems**
  - Overfitting
  - Limited tests' number
  - Stop criteria
  - Missing attribute values

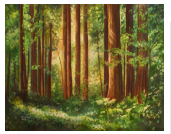
Random Forests 1/2

- **How to create them?**
  1. Choose subset of all tests $(\sqrt{number\_of\_tests})$
  2. Create tree -> untill the end
  3. Repeat steps 1-2 several hundred times
- **How to use them?**
  1. Take a single example
  2. Check the category assigned by every tree
  3. Vote for the final category



Random Forests 2/2

- **Advantages**
  - Free from overfitting
  - Typically better than a single tree (best possible?)
  - Handles large number of attributes
  - Possible to create attributes' rating
- **Disadvantages**
  - Slower than a single tree
  - Impossible to interpret the result



Better to be lazy?

- **Why should we create any model?**
- **Better to wait until somebody asks…**
- **When asked give the best possible answer**
  - What is "best possible answer"?
  - How to measure similarity?
  - …

Ridiculously easy algorithm

- **k-Nearest Neighbors (kNN)**
  - Gets the training set
  - Creates search structure (k-d tree)
  - Waits until asked…
- **When asked:**
  - Searches in the k-d tree appropriate k-dimensional rectangle
  - Founded neighbors vote
  - Answer is returned

Lazy learning c.d.

- **Very often the best possible algorithm**
- **Creating k-d tree is the most difficult step**
- **How to measure distance?**
  - Euclidean distance
  - Minkowski distance
  - Chebyshev distance

Measuring distance

- **General formula – Minkowski distance**

$$\sqrt[p]{\left(\sum_{i=1}^{n}[a_i(x_1)-a_i(x_2)]^p\right)}$$

- **For p = 2 => Euclidean distance**
- **For p = *infinity* => Chebyshev distance**
- **The higher p the bigger influence of single attribute**

kNN algorithm summary

- **Advantages**
  - Very good performance
  - Excellent results in many tests
  - Easy implementation
- **Disadvantages**
  - Overfitting (for k = 1) – training set errors' duplication
  - Problem with measuring similarity

Classification algorithms

SPAM7 (6 attributes)

Algorithm	Settings	Error
rpart	cp = 0.01, minsplit = 20, minbucket = 7, maxdepth = 30 (default)	0.1491739
rpart	cp = 0.1, minsplit = 20, minbucket = 7, maxdepth = 30	0.2138043
rpart	Default	0,144666660
kNN	k = 1	0,160333340
kNN	k = 5	0,133666660
kNN	k = 10	0,122333340
randomForest	mtry = 1, ntree = 500	0.12
randomForest	mtry = 2, ntree = 500 (default)	0.1192609
randomForest	mtry = 2, ntree = 1000	0.1184783
NaiveBayes	-	0,238999980

Is accuracy only important thing?

- **Attributes' cost**
  - Some attributes are more easily measured than the others
- **Errors' cost**
  - Preferring one category than others
  - E.g. diagnostic applications, searching for new particles

Grouping algorithms

When grouping can be useful?

- **Knowledge itself**
  - Market segmentation
- **Anomalies discovering**
  - Diagnostics
  - Frauds
- **Pre-processing when one model is to complicated**

Grouping characteristics

- **Maximal similarity inside group**
- **Maximal diffrence between groups**
- **Unsupervised learning – no categories given**

Cobweb – hierarchical algorithm

- **Builds hierarchical groups**
- **Tree representation**
  - Level 0 – root, covers whole training set
  - Level 1 – just below the root, final division
  - Leaves – nodes with only 1 example

COBWEB – short overview

1. **Starts with one example and creates tree with one node**
2. **Adds next examples and creates new nodes, leaves, merges nodes...**
3. **Repeats step 2 until all examples are covered**

Association algorithms

- **Detect associations**
- **Do not classify examples**
- **Equal to finding frequent item-sets**

Frequent item-sets

- **Bit of math:**
  - Support factor

$$S_D(A => B) = \frac{|D_{A \cup B}|}{|D|}$$

  - Fidelity factor

$$f_D(A => B) = \frac{|D_{A \cup B}|}{|D_A|}$$

Finding frequent item-sets

- **Apriori Algorithm assumptions**
  - Every subset of frequent set is frequent
  - All subsets of frequent set must be frequent

Apriori algorithm steps

- **Calculate support factor for S_1**
- **For i = 2, 3, ... do**
  - Merge frequent subsets that differ on 1 element
  - Remove sets that have not frequent subsets
  - Calculate support for new sets

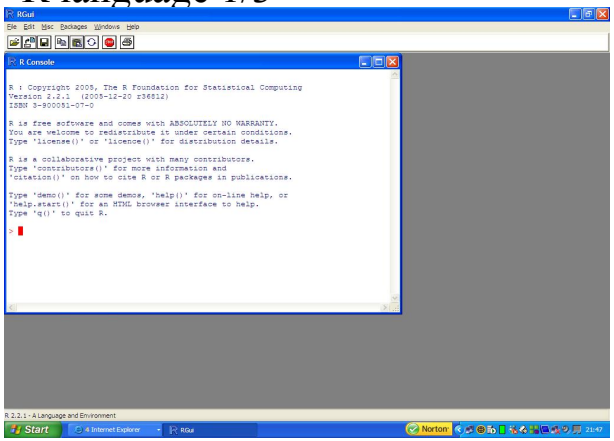
Where it can be applied?

- **Shoping basket analysis**
- **Courses scheduling**
- **Repairs planning**
- **Text analysis**
- **Other ...**

Particle physics and ML

- **Classifing particles – classification algorithms**
- **Grouping events – grouping algorithms**
- **Detecting events that coexist together – association algorithms**
- **Other...**

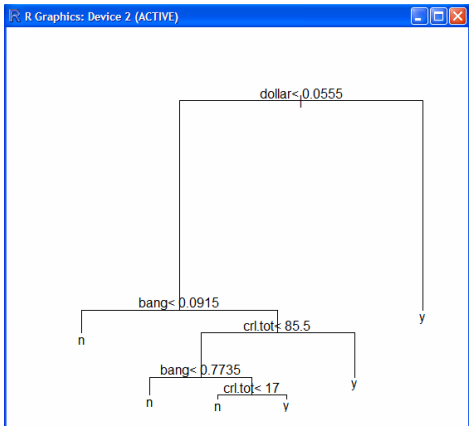
R language 1/3



R language 2/3

```
R Console

>
>
>
> library(rpart)
> tree <- rpart(yesno ~.,spam7,method="class");
> tree
n= 4601

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 4601 1813 n (0.6059552 0.3940448)
   2) dollar< 0.0555 3471  816 n (0.7649092 0.2350908)
     4) bang< 0.0915 2420  246 n (0.8983471 0.1016529) *
     5) bang>=0.0915 1051  481 y (0.4576594 0.5423406)
      10) crl.tot< 85.5 535  175 n (0.6728972 0.3271028)
        20) bang< 0.7735 418  106 n (0.7464115 0.2535885) *
        21) bang>=0.7735 117   48 y (0.4102564 0.5897436)
          42) crl.tot< 17 43   12 n (0.7209302 0.2790698) *
          43) crl.tot>=17 74   17 y (0.2297297 0.7702703) *
      11) crl.tot>=85.5 516  121 y (0.2344961 0.7655039) *
   3) dollar>=0.0555 1130  133 y (0.1176991 0.8823009) *
> plot(tree)
> text(tree)
```


R language 3/3



R-environment

- **Easy to test algorithms**
- **Online packages installation**
- **Powerfull language**
- **Easy data manipulation**
- **Statistical operators**
- **Extensions in another languages (C/C++)**

Summary 1/2

- **Classification algorithms**
  - Decision trees, rules induction, kNN...
- **Grouping algorithms**
  - COBWEB
- **Association algorithms**
  - Apriori algorithm

Summary 2/2

- **R language environment**
  - Flexible, still under developement
  - Easy data manipulation
  - Statistical analysis
  - Easy to test and compare algorithms

Questions?

LECTURE 3

Neural Networks

Monday 6 March 2006			
14:00 - 14:55	Lecture 3	**Neural Networks**	Liliana Teodorescu
		This lecture will present the fundamentals of the Artificial Neural Networks and examples of their applications in HEP data analysis. The examples will show the development cycle of these algorithms in HEP: a slow and late start followed by a gradually increasing presence and acceptance in the physicists' community. The lecture targets both physicists and computer scientists interested in algorithms for data analysis. A minimal general background in particle physics data analysis techniques is sufficient for understanding the topic. No a priory knowledge on Artificial Neural Networks is required.	
		Introduction - Biological Neural Networks - Artificial Neural Networks (NN) **Basics of NN** - Artificial neuron - Perception - Classification of NN **Operation of NN** - Learning types and rules - Learning and testing **Examples of NN** - Feed-forward NN - Recurrent NN - Functional NN **Performance Issues** - Performance factors and measures - Analysis of performance **Examples of NN in HEP** - NN triggers - NN for offline data analysis applications **Pro's and Con's NN in HEP**	

Neural Networks

Liliana Teodorescu

Brunel University

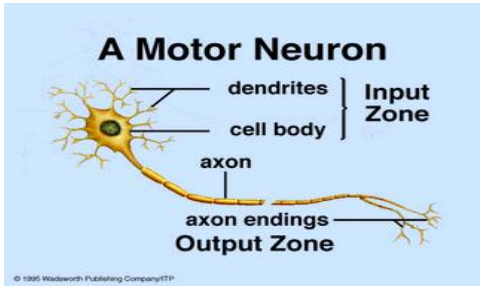
Inverted CERN School of Computing, 6-8 March 2006

Outline

- biological and artificial neurons
- artificial neural network architectures
- learning rules – supervised learning
- performance measures
- applications in HEP

Biological Neural Networks

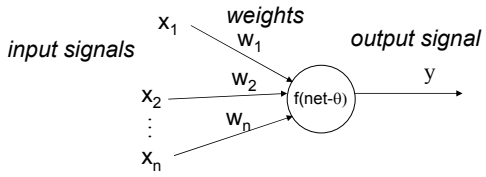
Biological neural system – network of neural cells called *neurons*



Signal transmission - from dendrites to the axon and connected dendrites when the cell "fire" => a neuron can inhibit or excite a signal

Artificial Neural Networks

Artificial neural networks (NN) – layered network of ***artificial neurons (AN)***



AN - receives signal from environment or other AN

weights – inhibit (negative values) or excite (positive values) an input signal

AN - collects input signals, calculates a net signal and transmits an output signal

$$net = \sum_{i=1}^{n} w_i x_i$$

AN – summation unit

$$net = \prod_{i=1}^{n} w_i x_i$$

AN – product unit (allow higher order combinations of inputs => increased information capacity)

Activation function

AN – calculates the output signal using the *activation function*

depends on *net* and θ (threshold, bias)

❖ introduce additional input unit (*bias unit*) $x_{n+1} = -1$ with weight $w_{n+1} = \theta$

$$net = \sum_{i=1}^{n+1} w_i x_i \longrightarrow \text{activation function } f(net)$$

Types of activation functions

Linear function

$$f(net) = \beta \cdot net$$

Sigmoid function

$$f(net) = \frac{1}{1 + e^{-\lambda \cdot net}}$$

usually λ=1

Gaussian function

$$f(net) = e^{-\frac{net^2}{\sigma^2}}$$

Step function

$$f(net) = 1 \quad \text{if} \quad net \geq 0$$
$$f(net) = 0 \quad \text{if} \quad net < 0$$

Ramp function

$$f(net) = \beta \quad \text{if} \quad net \geq \beta$$
$$f(net) = net \quad \text{if} \quad |net| < \beta$$
$$f(net) = -\beta \quad \text{if} \quad net \leq -\beta$$

Neural Networks Types

Multilayer NN

Feedforward NN – **receive external signals and propagate them through all the layers producing the output signal (no feedback connection to previous layers)**

- ❖ **standard multilayer NN**
- ❖ **functional link NN**
- ❖ **product unit NN**

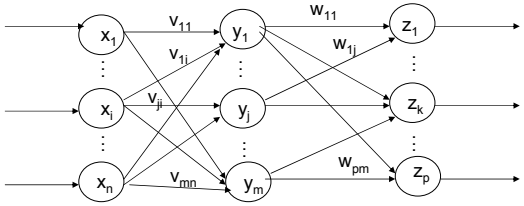
Recurrent NN – **have feedback connection to previous layers**

Time-delay NN – **memorise a window of the previously observed patterns**

Feedforward NN

❖one input layer, one output layer, one or more hidden layers

❖can have direct (linear) connections between the input and output layers



$$z_k = f_{z_k}(net_{z_k}) = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(net_{y_j})) = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(\sum_{i=1}^{n+1} v_{ji} x_i))$$

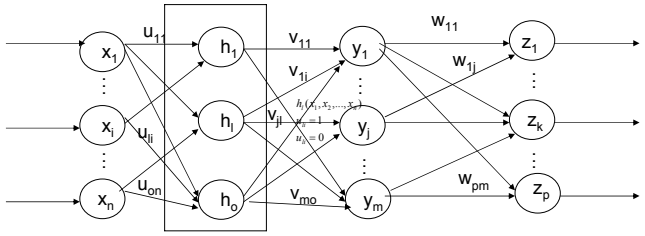
❖each activation functions can be different

❖Input unit can implement activation functions (usually a linear function)

Functional Link NN

❖input layer expended into a layer of functional units $h_l(x_1,\ldots x_n)$

$u_{li}=1$ if h_l depends of x_i, $u_{li}=0$ otherwise



$$z_k = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(\sum_{l=1}^{o+1} v_{jl} h_l(x_1,\ldots x_n)))$$

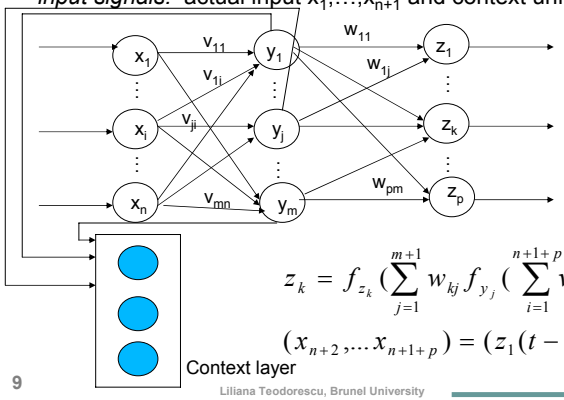
❖faster training and improved accuracy

J. Ghosh, Y. Shin, International Journal of Neural Systems, Vol. 3. No4, pp 323-350

Recurrent NN

❖Copy the hidden layer (context layer) that store the previous state of the hidden layer

input signals: actual input $x_1,\ldots,x_{n+1}$ and context units $x_{n+2},\ldots x_{n+1+m}$



$$z_k = f_{z_k}\left(\sum_{j=1}^{m+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{n+1+p} v_{ji} x_i\right)\right)$$

$$(x_{n+2},\ldots x_{n+1+p}) = (z_1(t-1),\ldots z_p(t-1))$$

NN Learning

Process through which the weight values are determined, adjusting them until certain criteria are satisfied.

Types of learning

❖ ***Supervised learning***
✓ training data - data set containing the input vector and a target (desired output) associated with each input vector
✓ aim – to adjust the weight values such that the difference between the real output and the target output is minimised

❖ ***Unsupervised learning***
✓aim – discover patterns in the input data (perform a clustering of the training patterns)

❖ ***Reinforcement learning***
✓ aim – reward the parts of NN responsible for good performance and penalise those responsible for bad performance

Supervised Learning Rules -AN

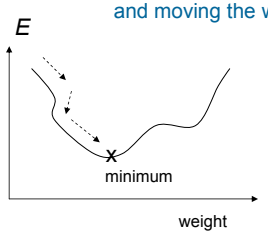
Gradient descent (GD)

Objective function – error function that measure the AN's error in approximating the output target

$$E = \sum_{p=1}^{P} (t_p - f_p)^2$$

t_p- target output for pattern *p*
f_p- actual output for pattern *p*
P **– total number of patterns**
Pattern = input-target vector pairs

Aim of GD - to find the weight that minimize *E*
- achieved by calculating the gradient of *E* in weight space and moving the weight vector along the negative gradient



For a single training pattern

$$w_i(t) = w_i(t-1) + \Delta w_i(t)$$

$$\Delta w_i(t) = \eta\left(-\frac{\partial E}{\partial w_i}\right)$$

$$\frac{\partial E}{\partial w_i} = -2(t_p - f_p)\frac{\partial f}{\partial net_p} x_{i,p}$$

η - learning rate (size of steps taken in the negative direction of the gradient)

Supervised Learning Rules -NN

Gradient Descent Optimisation => ***backpropagation*** **learning mechanism**

Learning iteration - ***epoch***

Phases

1. ***Feedforward pass*** – calculates the output value of NN
2. ***Backward propagation*** – propagates the error signal back from the output layer toward the input layer and adjust the weights as functions of the backpropagated error signal

Objective function

$$E_p = \frac{1}{2}\frac{\sum_{k=1}^{K}(t_{k,p} - z_{k,p})^2}{K}$$

Weights update

$$w_{kj}(t) = w_{kj}(t) + \Delta w_{kj}(t) + \alpha\Delta w_{kj}(t-1)$$
$$v_{ji}(t) = v_{ji}(t) + \Delta v_{ji}(t) + \alpha\Delta v_{ji}(t-1)$$

α- momentum
η-learning rate

$$\Delta w_{kj} = \eta\left(-\frac{\partial E}{\partial w_{kj}}\right) = -\eta\delta_{z_k} y_j$$
$$\Delta v_{ji} = \eta\left(-\frac{\partial E}{\partial w_{ji}}\right) = -\eta\delta_{y_j} z_i$$

where

$$\delta_{z_k} = \frac{\partial E}{\partial net_{z_k}}$$ output error

$$\delta_{y_j} = \frac{\partial E}{\partial net_{y_j}}$$ hidden layer error

Supervised Learning

Algorithm

1. Initialise wieghts, η, α, number of epochs $\xi=0$
2. Initialise $E_T=0$
3. For each training pattern p
   a. calculate $y_{j,p}$ and $z_{k,p}$ (feedforward)
   b. calculate output error $\delta_{z_{k,p}}$ and hidden layer error $\delta_{y_{j,p}}$
   c. adjust weights w_{kj} and v_{ji} (backpropagation of erors)
   d. $E_T = E_T + E_p$
4. $\xi=\xi+1$
5. Test stopping criteria; if not met then go to step 2

Stopping criteria

- maximum number of epochs
- E_T on the training set is small enough
- overfitting point

Supervised Learning

Weight initialisation

- small random weights centred around 0 => net input signal close to 0 => activation function generates midrange values
- random weights in the range $\left[\frac{-1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right]$

fan_in – number of connections leading to a unit

Learning rate (η)

- controls the size of each step toward the minimum of the objective function
- start with small values around 0 (e.g. 0.1)
- increase the value if the convergence is low
- decrease the value if the error does not decrease fast enough

Momentum (α)

- momentum term averages the weight changes ensuring the search path is in the average downhill direction
- usually value 0.9

NN Performance Measures

Accuracy

- ***Mean squared error***

$$E = \frac{\sum_{p=1}^{P}\sum_{k=1}^{K}(t_{k,p} - z_{k,p})^2}{PK}$$

P – number of patterns
K – number of outputs

E_T – training error (error on the training set)
E_G – generalisation error (error on the generalisation set – independent on training set)
NN objective – low generalisation error

- ***Classification accuracy*** = the percentage of the patterns correctly classified (specific to classification problems)
- ***Correlation coefficient*** between the output and target values

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\sum_{i=1}^{n}(y_i - \bar{y})}{\sigma_x \sigma_y}$$

$x = z_{k,p}$
$y = t_{k,p}$

NN Performance Measures

Overfitting

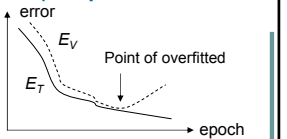
Overfitting a training set - NN memorises the training patterns => low generalisation power
- occurs when the NN architecture is too large (many hidden units => many weights)

Remedies
- optimising NN architecture
- using enough training patterns

Detection of the overfitting point – estimation of the generalisation error during training on a validation set (independent of the training set)

$E_V > E_{V\,average} + \sigma_{E_V}$

Validation error | average validation error over the previous epochs | standard deviation in the validation error



NN Performance Measures

Complexity – ***measure of the computational time***

Influenced by
- ❖ network architecture
- ❖ training set size
- ❖ complexity of the learning rule

Measures
- ❖ number of epochs to reach a certain error
- ❖ number of patterns presentations
- ❖ number of weights updates
- ❖ total number of calculations made during training

Convergence
- ❖ ability of the NN to converge to a specific error level
- ❖ expressed as number of times, out of a fixed number of simulations, that the NN succeeds to reach that error

NN in HEP

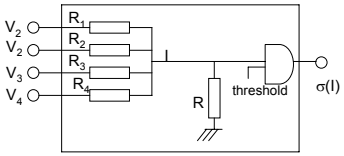
Applications in
- ❖ Real time triggering
- ❖ Offline reconstructions
  - ✓ track reconstruction
  - ✓ vertex reconstruction
  - ✓ particle identification or discrimination
- ❖ Physics processes reconstruction

Types of NN used – supervised learning NN
- ❖ feedforward NN
- ❖ recurrent NN

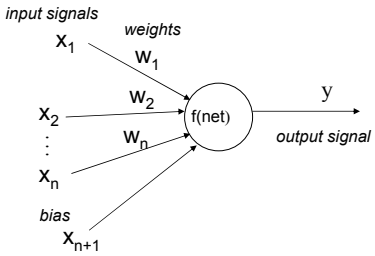
NN in HEP

Trigger

Coincidence circuit



Artificial neuron



Similarity

$V_i \longleftrightarrow x_i$
$R_i \longleftrightarrow w_i$
Threshold $\longleftrightarrow$ Bias
$\sigma(I) \longleftrightarrow$ f(net)

NN in HEP

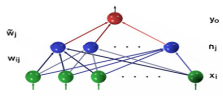
L2 trigger for H1 experiment

„L2NN“ Trigger Scheme

10 MHz

L1 2.3 µs

500 Hz



L2 20 µs → 3 layers feedforward NN
- ❖ one NN for each process
- ❖ discriminate physics rrom bkg.

50 Hz

L4 100 ms

10 Hz

$y_0 \approx 0$ bkg.
$y_0 \approx 1$ physics

Input variables
- ❖energy sums in subsets of the calorimeter
- ❖Information on vertex position
- ❖charge tracks multiplicity

…

http://wwwh1.mppmu.mpg.de/projects/neuro/neuro.html

TE	L1ST	Physics
*00	78	Charged Current old
01	68	Phi K+K-
02	52,54	J/Psi ee
03	83	DiJet
04	54	J/Psi µµ
05	32	D* untagged
06	40	Spacal back2back
07	78	Charged Current
08	33	J/Psi ee TC (1999)
09	41	DVCS
10	83	D* tagged
*11	33	J/Psi ee TC (2004)
12	15	J/Psi µµ inelastic

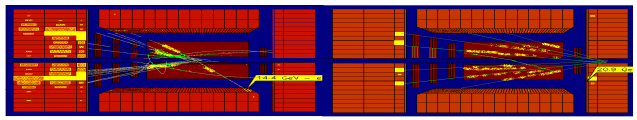
NN in HEP – offline applications

Examples – ACAT2000

- Selecting WW (DELPHI)
- Searching for single top (DØ)
- Measuring the top mass
- Searching for the Higgs
- Calorimeter energy estimation (ATLAS)
- Electron/jet discrimination (ATLAS)
- Particle identification (DØ)
- Tracking in vertex detector (HERA-B)

+ Job schedule!

e.g. Vertex finding (ZEUS) -E. Etzion ACAT2000



- **NN Layer 1 (large retina with ~ 100,000 neurons)**
  - receptive fields of neurons give (x,y) position of hits
- **NN Layer 2 (segment finder)** Gives segment orientation.
- **NN Layer 3 (arc finder)** Gives arc (k,q,ring)
- **NN Final layer (track finder)**

Pro's and Con's of NN in HEP

Pron's

- highly non-linear cuts => extends beyond the conventional cut based approach
- tolerance to noise
- speed due to intrinsic parallelism -> easy implementation as electronic VLSI circuits
- programmability => quick adaptability to new running conditions

Con's

- less evident interpretation of cuts
- rely on MC for learning
  - OK for low level pattern reconstraction applications
  - unusable for unknown/poor known processes
    -> what about unsupervised NN? – not used yet

Bibliography

Books

- Phil Picton, Introduction to Neural Networks
- Kevin Gurney, An Introduction to Neural Networks
- Christopher Bishop, Neural Networks for Pattern Recognition
- Xiang-Sun Zhang, Neural Networks in Optimisation
- Joey Rogers, Object-Oriented Neural Networks in C++

Journals

- IEEE Transactions on Neural Networks

NN in HEP

- B. Denby, Neural Networks in HEP – a Ten Years Perspective, Computer Physics Communications 119 (1999) 219
- B. Denby, Tutorial on Neural Networks, FERMILAB-CONF-92/121-E
- ACAT conference proceedings

Software

- ROOT implementation
- STATISTICA – http://www.statsoft.com
- many others …

LECTURE 4

Support Vector Machines

Monday 6 March 2006			
15:05 16:00	Lecture 4	**Support Vector Machines**	Anselm Vossen
		Support Vector Machines (SVMs) are advanced algorithms for classification and regression that are conceptually easy to understand. Recently SVMs gained increased popularity due to state-of-the-art performance paired with a good mathematical understanding which enables users to choose arbitrary complex classification or regression functions without over-fitting the data. A technique known as structural risk minimization. The lecture targets computer scientists interested in state of the art pattern recognition algorithms. It will also be interesting for physicists interested in making these algorithms working for them. For some of the intricacies a basic knowledge of linear algebra and statistics will be helpful. Additionally, this lecture will use the vocabulary introduced in the preceding ones, especially "Feature Selection and Classification Basics".	
		The Linear Classifier - Toy Example: Separating points on a plane - Optimal Margin and Support Vectors **Structural Risk Minimization** - Short (and incomplete) Introduction to Vapnik-Chervonenkis (VC) Theory - Finding a balance between fitting and overfitting the data - How to incorporate this into the linear classifier **Kernel Methods** -The "Kernel Trick": Mapping the data into a convenient higher dimensional space with little computing overhead - Using the "Kernel Trick" to extend linear algorithms to nonlinear ones **Support Vector Machines** - Putting everything together to build powerful classification and regression algorithms - SVM Libraries: how to use SVMs in your code	

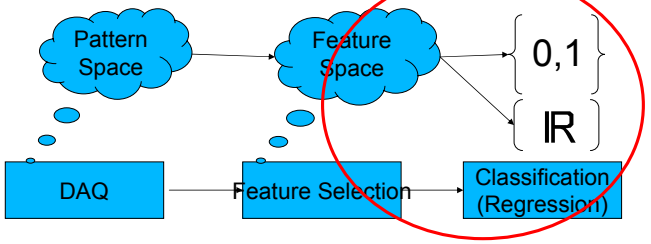
Support Vector Machines

Anselm Vossen

Universität Freiburg

The Big Picture

- **Generalities**
  - Support Vector Machines (SVMs) can be used for classification
  - Support Vector Regression (SVR) for regression
  - State of the art performance, easy to use, well understood
- **Complex algorithms, perform well with little training data and little preprocessing, well grounded in statistical learning theory**

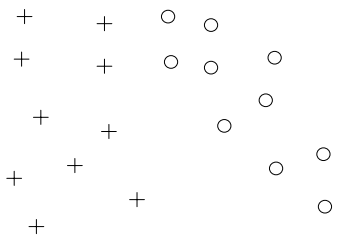


Outline

- **Introducing Linear Separation for Classification**
  - Optimal separating hyperplane
  - Support vectors
- **Nonlineare Separation: The Kernel Trick**
  - Kernel functions
- **A Glimpse on Vapnik Chervonenkis Theory**
  - Complexity of a function
  - Bounds on the expected error
  - Structural risk minimization
- **Application of SVMs**
  - Why use SVMs
  - SVMs in HEP
  - Regression
  - Libsvm, libsvmtl: free C (C++) Libraries that you can use

Classification Problem

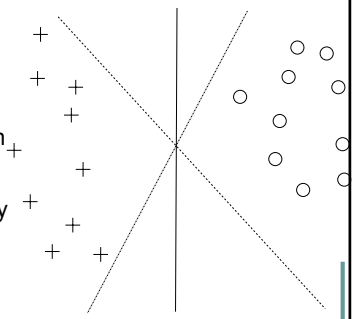
Separate crosses and circles (in an optimal way)



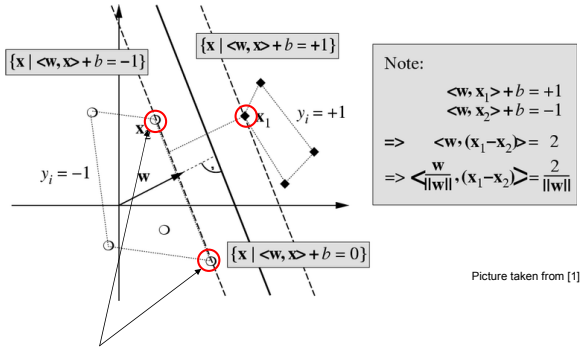
- Supervised learning
- Given $\vec{x} \in \mathrm{H}$
- Construct $f : \mathrm{H} \to \{\pm 1\}$

Linear Classifier

- Linear separation is the easiest way to separate two classes
- Nevertheless: there are many possibilities to draw the line
- Here: large margin classifier: take the one that has the biggest margin to the closest points
- Note: this line is only determined by these points (the so called support vectors)



Construction of a Linear Classifier



Picture taken from [1]

Support Vector

Mathematical Details

- Classification function: $f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b$
- Maximize margin ↔ Minimize target function:

$$\tau(\vec{w}) = \frac{1}{2}\|\vec{w}\|^2$$

- Lagrangian:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\| - \sum_{i=1}^{m} \alpha_i \left(y_i(<\vec{x}_i, \vec{w}> + b) - 1\right)$$

- Minimize with respect to primary variables $\vec{w}$,b
- Maximize with respect to α_i
- Convex optimization problem

Support Vector Expansion

- Solution can be written as
  - $f(\vec{x}) = \mathrm{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i < \vec{x}, \vec{x}_i > + b\right)$
  - α_i :Lagrange multipliers
  - $\alpha_i = 0$ for $\vec{x}_i$ not on the border $\left(y_i(<\vec{x}_i, \vec{w}> + b) - 1 \neq 0\right)$
  - The solution is called “Support Vector Expansion”
  - Only SVs are needed for the computation

Kernel Functions

- **Up to now we used the scalar product as a measure for similarity**
- **But we can also use other functions that are more appropriate for the data, an example would be a Gaussian kernel:**

$$< \vec{x}, \vec{x}' > \rightarrow k(\vec{x}, \vec{x}') = e^{-\gamma |\vec{x} - \vec{x}'|^2}$$

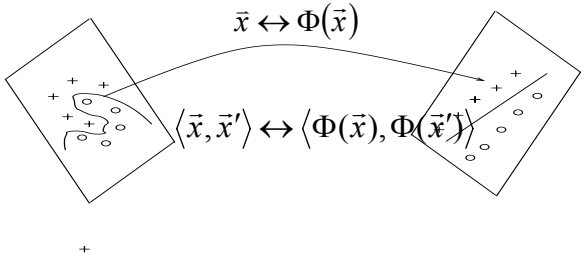
Question: Does our classification procedure still work, if we substitute the scalar product with a kernel function?

Mercers Theorem and Kernel Trick

- Answer with Mercers Theorem: Yes, as long as the kernel matrix is positive definite
- $\mathbf{K_{ij}} = k(\vec{x}_i, \vec{x}'_j);\ \vec{x}_i, \vec{x}_j \in H$
- This is called the Kernel Trick
- If the kernel matrix is pos. def. there exists a function $\Phi: H \rightarrow H'$ where H' is another space (possibly with infinite dimension).
- If the kernel function is chosen well, the data can be separated much better in H'

Kernel Functions for SVMs

- Kernel Trick + SVM= nonlinear classifier!



(Some) Advantages of SVMs

- **Global optimum (in contrast to NN)**
- **Sparse representation (SV-Expansion)**
- **(Some) Interpretation in feature space possible**
- **Kernel-Methods allow the choice of an appropriate similarity measure**
- **Firmly grounded in statistical learning theory**

Examples of Kernel Functions

- Some ordinary kernels, linear, gaussian, polynomial

$$k(\vec{x},\vec{x}') = -\left|\vec{x}-\vec{x}'\right|$$

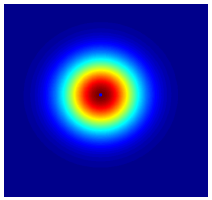
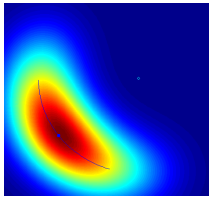
$$k(\vec{x},\vec{x}') = e^{-\gamma\left|\vec{x}-\vec{x}'\right|^2}$$

$$k(\vec{x},\vec{x}') = \left(\left\langle \vec{x},\vec{x}'\right\rangle + a\right)^d$$

Examples for Kernel Functions, Cont.

- Example for a complex kernel function: Haar kernel k_H computed from a base kernel k_0

$$k_H(\vec{x},\vec{x}') = \int\int_{GG} k_0(g\vec{x}, g'\vec{x})\,dg\,dg'$$

k_0 :  k_H : 

Other Kernel Methods

- **Novelty detection**
- **Kernel principal component analysis**
- **…**

Begin Detour to Structural Risk Minimization

- **Introduction to SRM**
- **A glimpse on Vapnik Chervonenkis theory**
- **Soft margin SVMs**

SVM and Structural Risk Minimization

y_i : Real value, $p(x,y)$: PDF

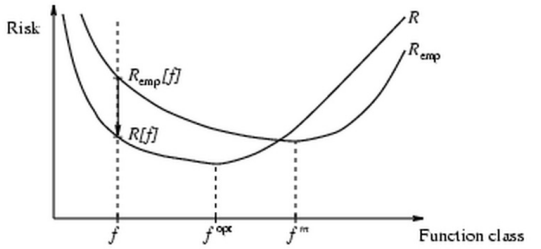
Empirical risk : $R_{emp}[f] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}|f(x_i) - y_i|$

Expected risk : $R[f] = \int |f(x) - y| dp(x,y)$

Vapnik Chervonenkis (VC) theory relates expected risk, empirical risk and the capacity of a function

- Soft margin SVM allows for capacity control

Structural Risk Minimization

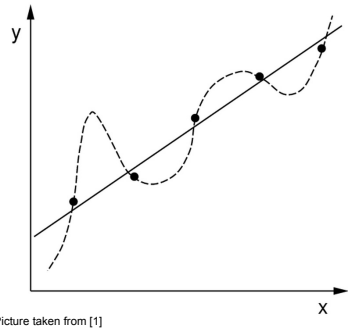


- **Find optimal function**

Picture taken from [1]

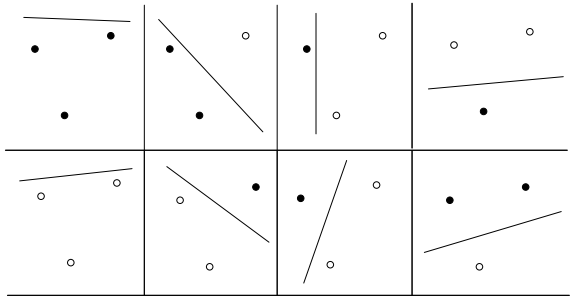
Empirical <-> Expected Risk

- Expected risk has to be estimated from empirical risk
- Empirical risk=0 not necessary good (overfitting)
- But we can give bounds on the expected risk, given the empirical risk and the capacity of a function



Picture taken from [1]

The Capacity of a function



- Vapnik Chervonenkis Dimension of a function=number of points that can be separated, independent from labeling
- E.g. Class of hyperplanes in $\mathbb{R}^2$

Risk Estimation

- Result from VC theory:

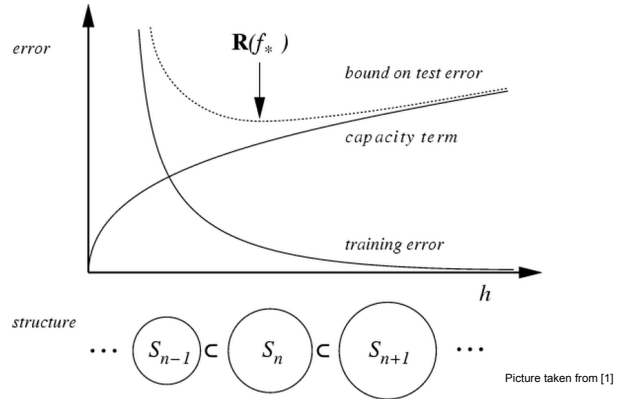
$$R[f] \leq R_{emp}[f] + \Phi(h, m, \delta)$$

With probability 1-δ
m: Number of training examples
h: VC dimension

Confidence term:

$$\Phi(h, m, \delta) = \sqrt{\frac{1}{m}\left(h(\ln\frac{2m}{h}+1) + \ln\frac{4}{\delta}\right)}$$

Structural Risk Minimization



Picture taken from [1]

Capacity of a SVM->Soft Margin SVM

- It can be shown, that the capacity of a SVM is directly related to the width of the margin
- Wide margin ->smooth function in feature space
- For SRM: Optimize margin, accept wrong classification in favor of a wider margin: Soft Margin SVM
- Parameter C controls trade off margin width – training error

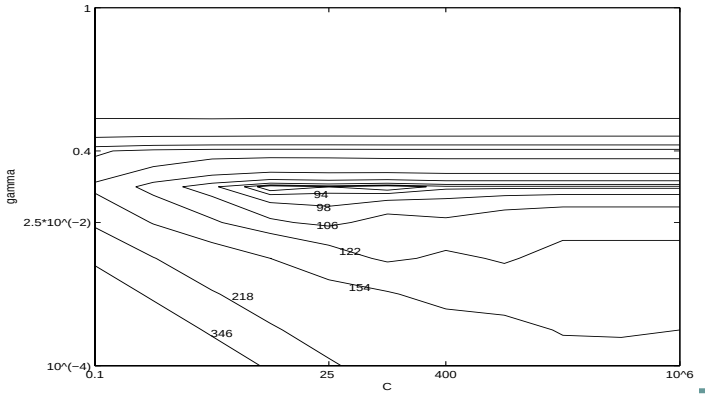
New cost function :

$$\tau(\vec{w}, \vec{\xi}) = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{u=1}^{m}\xi_i$$

ξ_i : Slack Variables

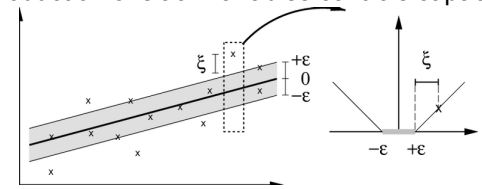
Practical: How hard is it to find good Parameters?

- **Example, gauss kernel, parameter space C, γ**



Support Vector Regression

- Regression : Minimize $\frac{1}{2}\|\vec{w}\| + C\sum_{i=1}^{m} \underbrace{\xi}_{|y_i - f(\vec{x}_i)|_\varepsilon}$
- Problem analogous to SVM, ε-insensitive loss function leads to sparse representation
- Introduction of slack variables controls capacity



Picture taken from [1]

How to use SVMs in your Code

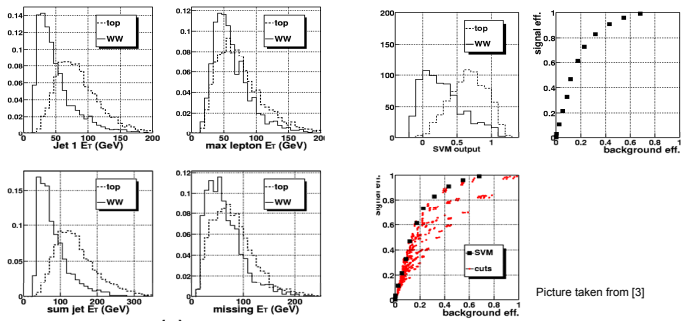
- **Standard Library for SVM: libsvm by Chih-Chung Chang and Chih-Jen Lin**
  - C Library and command line tools
  - Good documentation and tutorial introduction
  - Easy to use
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- **Template based libsvmtl by Olaf Ronneberger**
  - Based on libsvm code
  - more modern coding
  - http://lmb.informatik.uni-freiburg.de/lmbsoft/libsvmtl/index.en.html?1000000000011101
- **Weka (see prev. lecture) can use SVMs**

Applications in HEP

- **In HEP SVMs were used for**
  - Particle Identification for DELPHI data
  - flavour tagging, muon Identification in OPAL
  - Signal/background discrimination in tau production @Tevatron
- **Performance out of the box comparable to Neural Networks**
  - Fewer optimization parameters
  - Mathematically better understood, results can be interpreted

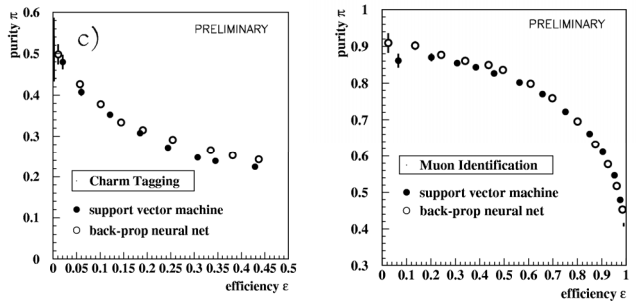
Example: Top Quark Analysis

- Fermilab proton-antiproton Run
- WW Background



Picture taken from [3]

Example: Flavor Tagging and Muon Identification @ OPAL



Picture taken from [2]

When and why should you use SVMs?

- **Neural Networks still perform slightly better**
  - Do not replace a running NN system
- **SVMs are a new ⇒ potential for development in HEP**
  - Best performance on many datasets by SVMs
- **Easy to use, even for a non expert**
- **Interpretation in feature space**
- **There are many more kernel methods that can be used**
  - E.g. kernelised PCA

Further Reading

- **[1] B. Schölkopf and A.J. Smola, Learning with kernels, MIT Press, Cambridge, MA, 2002**
- **J. Shawe-Taylor and N. Cristiani, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004**
- **Various Documentation on www.kernel-machine.org**
- **SVMs in HEP:**
  - [2] P. Vannerem et. Al. , Classifying LEP Data with Support Vector Algorithms (1999)
  - [3] A. Vaiciulis, Support Vector Machines in Analysis of Top Quark Production (2002)
  - [4] N. Barabino et. Al. Support Vector Machines vs. Multi-Layer Perceptrons in Particle Identification (1999)
- **Libsvm: Chih-Chung Chang and Chih-Jen Lin, LIBSVM: a library for support vector machines (2001)**

Questions?

- **Thank you for your attention!**

LECTURE 5

Evolutionary Computation

Monday 6 March 2006			
16:30 - 17:25	Lecture 5	**Evolutionary Computation**	Liliana Teodorescu
		This lecture will present the fundamentals of the Evolutionary Computation and of the main types of evolutionary algorithms. A survey of the applications of these algorithms in HEP data analysis will also be presented, illustrating the early development phase of an emerging technique in HEP data analysis. A new evolutionary algorithm, Gene Expression Programming, and its first application to HEP data analysis will also be presented. The lecture targets both physicists and computer scientists interested in algorithms for data analysis. A minimal general background in particle physics data analysis techniques is sufficient for understanding the topic. No a priory knowledge on Evolutionary Computation is required.	
		Introduction - Natural evolution - Simulation of the natural evolution on a computer - Specific terminology **Structure of an evolutionary algorithm** - Problem representation (encoding solutions) - Fitness functions - Genetic operators - Termination conditions **Types of evolutionary algorithms: Genetic Algorithms, Genetic Programming** - Problem representation for each type of algorithm - Genetic variation in each type of algorithm - Comparison of the different types of algorithms - Applications in HEP data analysis **New development in Evolutionary Computation: Gene Expression Programming** - Problem representation - Genetic variation - First application of Gene Expression Programming to HEP data analysis **Recommendations on when to use Evolutionary Algorithms**	

Evolutionary Computation

Liliana Teodorescu

Brunel University

Inverted CERN School of Computing, 6-8 March 2006

Outline

- Introduction to evolutionary computation
- Evolutionary algorithms
  - solution representation
  - fitness function
  - initial population generation
  - genetic and selection operators
- Types of evolutionary algorithms
  - Genetic Algorithms
  - Evolutionary Strategies
  - Genetic Programming
  - Gene Expression Programming
- Applications in HEP

Natural Evolution

Natural evolution - *optimisation* process aiming to increase *the ability* of individuals to survive and reproduce in a specific environment

- quantitatively measured by (*evolutionary*) *fitness*
- influenced by individual characteristics represented in its *chromosomes*

Sexual reproduction => offsprings with a combination of chromosome information from both parents

Natural selection => the more fit individuals will mate more often leading to offsprings with a similar or better fitness.

Goal of natural evolution - to generate a *population* of *individuals* with increasing fitness

Evolutionary Computation

Evolutionary computation – simulation of the *natural evolution* on a computer

Goal of natural evolution - to generate a *population* of *individuals* with increasing fitness

Goal of evolutionary computation - to generate *a set* of *solutions* (to a problem) of increasing *quality*

Terminology

- Individual – candidate solution to a problem

decoding ⇅ *encoding*

- Chromosome – representation of the candidate solution
- Gene – constituent entity of the chromosome
- Population – set of individuals/chromosomes
- Fitness function – representation of how good a candidate solution is
- Genetic operators – operators applied on chromosomes in order to create ***genetic variation*** (other chromosomes)

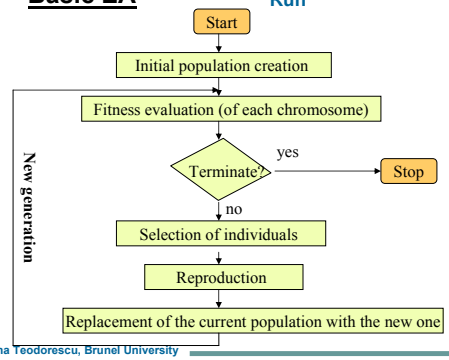
Evolutionary Algorithms

Natural evolution simulation - core of the ***evolutionary algorithms*** (EA)

EA - optimisation algorithms, iteratively improve the quality of the solutions until an ***optimal / feasible solution*** is found

- Problem definition
- Encoding of the candidate solution
- Fitness definition
- Run
- Decoding the best fitted chromosome = solution

Basic EA Run



Representation of Solutions

Chromosome – representation of the candidate solution

Each chromosome represents a point in search space

Appropriate chromosome representation
- very important for the success of EA
- influence the efficiency and complexity of the search algorithm

Representation schemes
- ***Binary strings*** – each bit is a boolean value, an integer or a discretized real number
- ***Real-valued variables***
- ***Trees***

Fitness Function

The most important component of EA !

Fitness function - representation of how good (close to the optimal solution) a candidate solution is

- maps a chromosome representation into a scalar value

$F : C^l \rightarrow \Re$ l – chromosome dimension

Fitness function needs to model accurately the optimisation problem

Used:
- in the selection process
- to define the probability of the genetic operators

Includes:
- all criteria to be optimised
- reflects the constraints of the problem penalising the individuals that violates the constraints

Initial Population

Generation of the initial population:

- random generation of gene values from the allowed set of values (*standard method*)
  Advantage - ensure the initial population is a uniform representation of the search space
- biased generation toward potentially good solutions if prior knowledge about the search space exists.
  Disadvantage – possible premature convergence to a local optimum

Size of the initial population:

- *small population* – represents a small part of the search space
  - ✓ time complexity per generation is low
  - ✓ needs more generations
- *large population* – covers a large area of the search space
  - ✓ time complexity per generation is higher
  - ✓ needs less generations to converge

Reproduction Operators

Purpose - to produce offsprings from selected individuals
Application - to create individuals for the next generation
- to replace parents with better fit offsprings

Typical operators

- *cross-over* – creates new individuals through the combination of the genetic material of the parents
- *mutation* - randomly changes the values of genes in the chromosome (introduces new genetic material into an existing individual, enlarging the search-space)
  - has low probability – in order not to distorts the genetic structure of the chromosome and to generate loss of good genetic material
- elitism/cloning – copies the best individuals in the next generation

The exact structure of the operators – dependent on the type of EA

Selection Operators

Used to select individuals for applying genetic operators or for composing the next generation

- Random selection – individuals are selected randomly, without any reference to fitness
- Proportional selection – the probability to select an individual is proportional with the fitness value

$$P(C_n) = \frac{F(C_n)}{\sum_{n=1}^{N} F(C_n)}$$

$P(C_n)$ –selection probability of the chromosome C_n
$F(C_n)$ – fitness value

✓ Normalised distribution by dividing to the maximum fitness - accentuate small differences in fitness values (***roulette wheel method)***

- Rank-based selection – uses the rank order of the fitness value to determine the selection probability (not the fitness value itself)

e.g. non-deterministic linear sampling – individual sorted in decreasing order of the fitness value are randomly selected

- Elitism – k best individuals are selected for the next generation, without any modification
  k – generation gap

Evolutionary Algorithms vs Classical Optimisation

	EA	CO
Transition from one point to another in the search space	Probabilistic rules Parallel search	Deterministic rules Sequential search
Starting of the search process	Set of point	One point
Search surface information that guides to the optimal solution	Derivative information (first or second order)	No derivative information (only fitness value)

Types of EA

- **Genetic Algorithms (GA)**
  - ***J. H. Holland,*** *Adaptation in Natural and Artificial Systems, Ann Arbor, MI, University of Michigan Press, 1075*
- **Evolutionary Strategies (ES)**
  - ***I. Rechenberg,*** *Evolutionstrategie:Optimierung technischer Systeme mavh Prinzipien der Biologischen Evolution, Frammann-Helzboog Verlag, Stuttgart, 1973*
  - ***H-P. Schwefel,*** *Evolutionsstrategie und numerische Optimierung, PhD Thesis, Technical Univerity Berlin, 1975*
- **Genetic Programming (GP)**
  - *J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press,1992*
- **Gene Expression Programming (GEP)**
  - *C. Ferreira, Gene Expression Programming: Mathematica Modelling by an Artificial Intelligence, Angra do Heroisma, Protugal,2001*

Main differences

- **Encoding method**
- **Reproduction method**

Genetic Algorithms

Chromosome representation

Classical method - binary string of fixed length

Each l-dimensional chromosome consists of l variables, each variable encoded as a bit string

- ✓**binary variables – each encoded as bit**
- ✓**nominal-valued discrete variables – each nominal value encoded as a bit string**
- ✓**continuous-valued variables – each variable is mapped to a bit string**

e.g. $x \in [x_{\min}, x_{\max}]$ converted to a 10-bit representation

$$(2^{10} - 1)\frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Other methods

- **integer or real-valued representations**
- **order-based representations**
- **chromosomes of variable length**

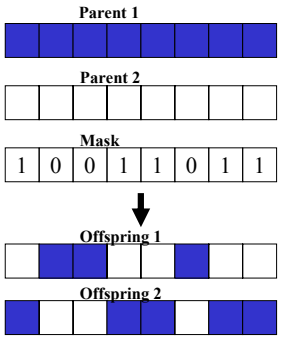
The Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991

Genetic Algorithms

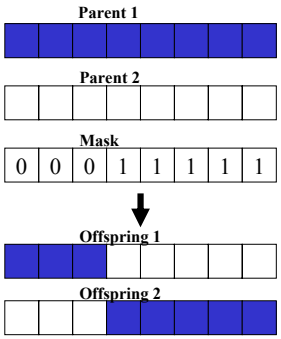
Reproduction

Cross-over operator **- takes place with a probability p_c** (cross-over rate

Uniform cross-over



One-point cross-over

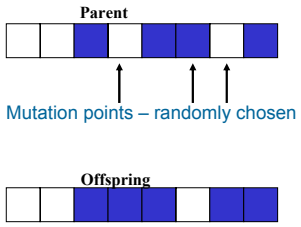


Genetic Algorithms

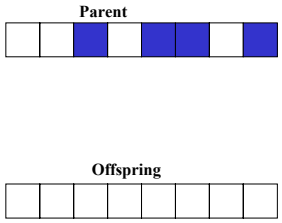
Reproduction

Mutation **- takes place with a probability p_m (mutation rate)- usually small**

Random mutation



Inorder mutation



Genetic Algorithms in HEP

Mainly for large-scale optimisation and fitting problems

Experimental HEP

- trigger optimisation (L1 and L2 CMS SUSY trigger)
  NIM A502 (2003) 693)
- neural-netwok optimisation for Higgs search
  F. Hakl et.al., talk at STAT2002

Theoretical/phenomenological HEP

- fitting isobar models to data for $p(\gamma,K^+)\Lambda$
  NP A 740 (2004)147
- discrimination of SUSY models
  hep-ph/0406277
- lattice calculations
  NP B (Pric. Suppl.) 73 (1999) 847; 83-84 (2000)837

Evolutionary Strategies

Based on the concept of evolution of evolution:
the evolution optimises itself

Individual – represented by its genetic characteristics and a strategy parameter that models the behaviour of the model in environment
Evolution – evolve both genetic characteristics and strategy parameter

Chromosome representation

$$C_n = (G_n, S_n)$$

G_n – genetic material (floating-point values)
S_n – strategy parameter

Strategy parameter
- ✓standard deviation associate with each individual
- ✓Standard deviation associated with each variable of an individual

Evolutionary Strategies

Genetic operators

- ***Cross-over operator***
  - ✓ local cross-over – one offspring is generated from material randomly selected from two parents
  - ✓ global cross-over – one offspring is generated from material randomly selected from the entire generation

  Recombination of the selected material
  - ✓ discrete – offsprings' gene value is the gene value of the parents
  - ✓ intermediate recombination – offsprings' gene value is the midpoint between the gene value of the parents

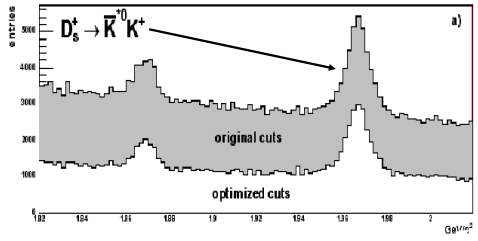
- ***Mutation operator***
  - ✓mutation of the standard deviation $\sigma_{g+1,n} = \sigma_{g,n} e^{\tau \xi_\tau}$ $\tau = \sqrt{I}$ $\xi_\tau \propto N(0,1)$
  - ✓mutation of the genetic material $G_{g+1,n} = G_{g,n} + \sigma_{g+1,n}\xi$ $\xi \propto N(0,1)$

Mutation accepted only if the offspring is better fit

Evolutionary Strategies in HEP

- **event selection optimisation, *NIM A534 (2004) 147***

Chromosome: cut values
cos(θ_H), p_{Ds} , mass constraint, vertex fit probability
Fitness function: $sig^2=S^2/(S+2B)$



45.4% improvement in sig^2

Genetic programming

GP searches for the *computer program* to solve the problem, not for the solution to the problem.

Computer program - any computing language (in principle)
- LISP (List Processor) (in practice)

LISP - highly symbol-oriented

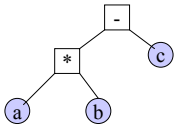
Mathematical expression

a*b-c

S-expression

(-(*ab)c)

Graphical representation of S-expression



functions (+,*) and terminals (a,b,c)

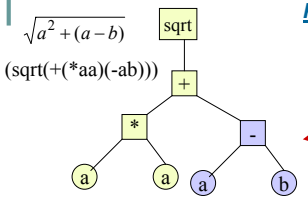
❖ ***Chromosome representation***

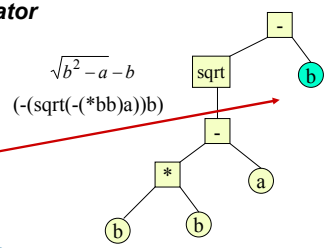
S-expression - variable length => more flexibile
- sintax constraints => invalid expressions produced in the evolution process must be eliminated => waste of CPU

Genetic programming

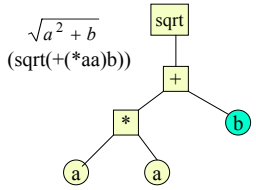
❖ ***Reproduction: cross-over operator***

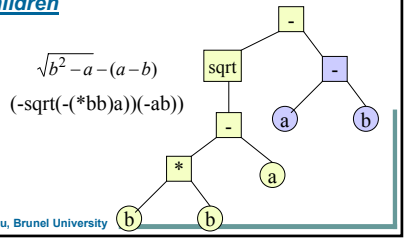
Parents





Children

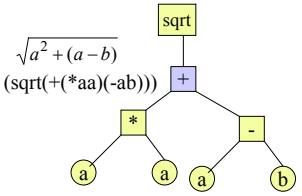


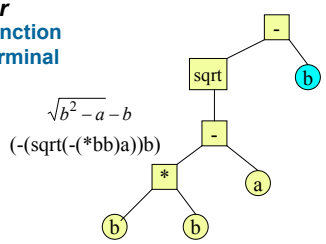


Genetic programming

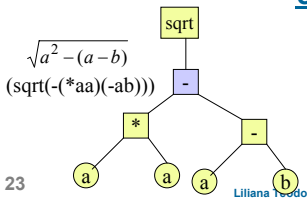
❖ ***Reproduction: mutation operator***
- ✓ function replaced by another function
- ✓ terminal replaced by another terminal

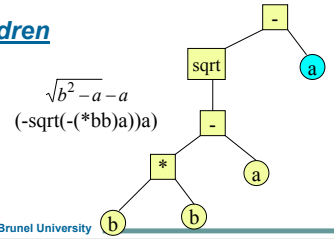
Parents





Children





Genetic programming in HEP

Experimental HEP - event selection

❖ Higgs search in ATLAS (physics/0402030)

❖ D, D_s and Λ_c decays in FOCUS (hep-ex/0503007, hep-ex/0507103)

e.g. Search for $D^+ \to K^+\pi^+\pi^-$ *(hep-ex/0503007)*

Chromosome: **candidate cuts - tree of:**
- ✓**functions: mathematical functions and operators, boolean operators**
- ✓**variables: vertexing variables, kinematical variables, PID variables**
- ✓**constants: reals (-2,2), integers (-10,+10)**

In total: 55

Fitness function ***(will be minimised)***

$$\frac{S+B}{S^2}\times 10000(1+0.005\times n)$$

n- number of tree nodes

penalty based on the size of the tree

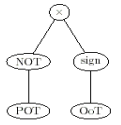
(big trees must make significant contribution to bkg reduction or signal increase)

Genetic programming in HEP

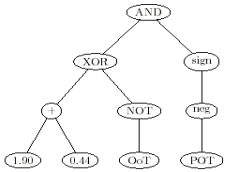
Basic procedure:

1. *Generates (almost randomly) a population of chromosomes*
2. *Loop over events and calculate the fitness for each chromosome*
   - *loop over each event and keep events where the tree evaluates to > 0*
   - *for survival events, fit signal (S) and bkg. (B)*
   - *calculate fitness of each chromosome*
3. *Select chromosomes, apply genetic operators and create the next generation*
4. *Repeat for the desired number of generations (40)*

Best fitted chromosomes from generation 0



(a) Fitness 0.458

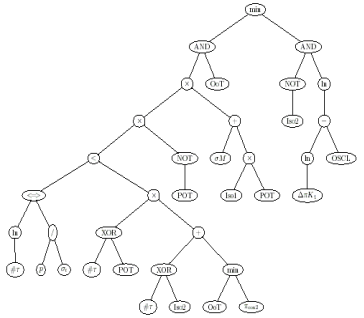


(b) Fitness 0.469

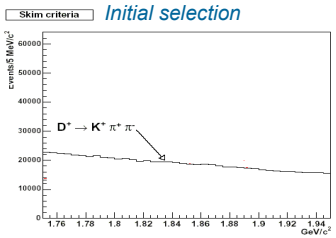
Inter point in target (POT<0) and Decay vertex out of target (OoT>0)

Genetic programming in HEP

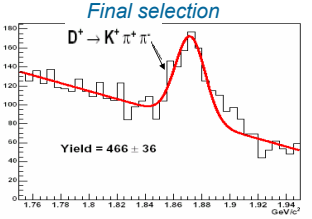
Best candidate, after 40 generations = final selection criteria


(a) Most fit tree: fitness 0.1234

Initial selection

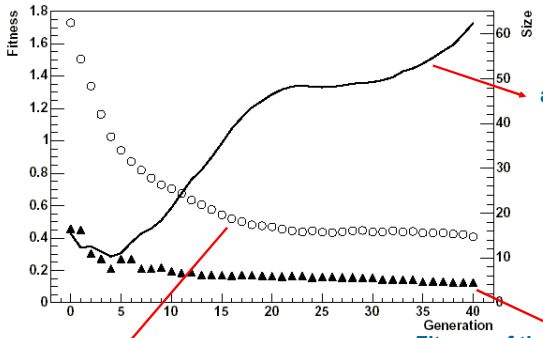


Final selection



Genetic programming in HEP

Evolution graph



average size of the individuals

Fitness of the best individual

Average fitness of the population

Gene Expression Programming

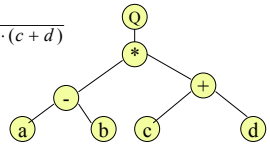
- **search for the computer program that solve the problem (as GP)**
- **works with two entities: chromosomes and expression trees**

Chromosome representation

Candidate solution represented by an expression tree (ET) (similar with GP tree)

ET encoded in a chromosome: read ET from left to right and from up to down

$\sqrt{(a-b)\cdot(c+d)}$



Q*-+abcd
Q means sqrt

Decoding the chromosome (translates the chromosome in an ET)

- **first line of ET (root) – first element of the chromosome**
- **next line of ET – as many arguments needed by the element in the previous line**

Gene Expression Programming

Chromosome representation (cont.)

Chromosome – has one or more genes of equal length

Gene – ***head***: contains both functions and terminals (length h)
- ***tail***: contains only terminals (length t)

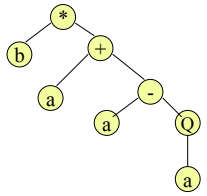
$t=h(n-1)+1$

n – number of arguments of the function with the highest number of arguments

e.g. set of functions: Q,*,/,-,+
set of terminals: a,b

n=2; h=15 (choosen) =>t =16 =>
length of gene=15+16=31

*b+a-aQab+//+b+babbabbbababbaaa



ET ends before the end of the gene !

Gene Expression Programming

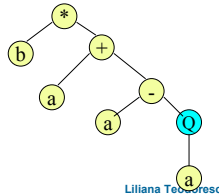
Reproduction

Genetic operators applied on chromosoms not on ET =>
always produce sintactically correct structures!

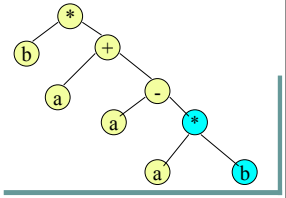
- Cross-over
- Mutation
- Transposition – a part of the chromosome moved to another part of the same chromosome

e.g. Mutation: Q replaced with *

*b+a-aQab+//+b+babbabbbababbaaa



*b+a-a*ab+//+b+babbabbbababbaaa



Gene Expression Programming (GEP) in HEP

L. Teodorescu, CHEP06

GEP for event selection
- cuts/selection criteria finding
- classification problem (signal/background classification)
- statistical learning approach

Data samples:
- *Monte-Carlo simulation from BaBar experiment*
- *Ks production in e^+e^- (~10 GeV)*

Functions and constants **to be used in the classification rules (cut type rule)**

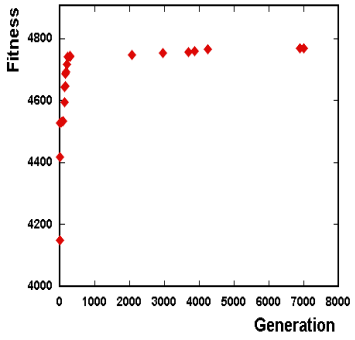
10 functions
- AND1 (x<0 and y<0 => 1 else 0), AND2 (x≥0 and y≥0 => 1 else 0)
- OR1 (x<0 or y<0 => 1 else 0), OR2 (x≥0 or y≥0 => 1 else 0)
- <, >, <=, >=, =, !=

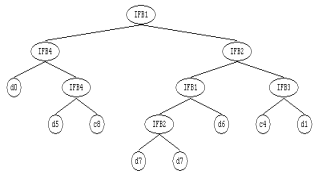
constants
- floating point constants (-10,10)

GEP in HEP

Data sample S/N=0.25

No. of genes = 1, Head length =10





Fsig ≥ 5.26,
Rxy < 0.19,
doca <1,
Pchi > 0

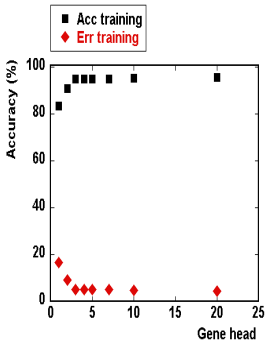
Classification Accuracy = 95.36%

GEP in HEP

Standard analysis

Fsig ≥ 4.0, Rxy ≤ 0.2cm, SFL ≥ 0cm, Pchi > 0.001
doca ≤ 0.4cm, |Rz| ≤ 2.8cm

Data sample S/N=0.25



GEP

Head	Acc. (%)	Selection criteria
1	83.34	Fsig ≥ 9.93
2	90.76	Fsig≥ 8.80, doca <1
3	94.88	Fsig > 3.67, Rxy ≤ Pchi
4	94.88	Fsig > 3.67, Rxy ≤ Pchi
5	95.04	Fsig ≥ 3.63, |Rz| ≤ 2.65, Rxy < Pchi
7	95.04	Fsig ≥ 3.64, Rxy < Pchi, Pchi > 0
10	95.360	Fsig ≥ 5.26, Rxy < 0.19, doca <1, Pchi > 0
20	95.50	Fsig > 4.1, Rxy ≤ 0.2, SFL > 0.2, Pchi > 0, doca > 0, Rxy ≤ Mass

When to use EA

- when the interrelations among the relevant variables are poorly understood
- when finding the size and shape of the ultimate solution is a major part of the problem
- when conventional mathematical analysis does not provide analytical solutions
- when an approximate solution is acceptable or is the only result that is ever likely to be obtained
- when small improvements in the performance a easily measurable and highly prized
- when there is large amount of data to be examined, classified and integrate

W. Banzhal et. al. Genetic Programming – an Introduction, Morgan Kaufmann, 1998

Bibliography

Books

D. Dumitrescu et. al., Evolutionary Computation
M. Mitchell, An Introduction to Genetic Algorithms
W. Banzhaf et.al., Genetic Programming: An Introduction
C. Ferreira, Gene Expression Programming: Mathematica Modelling by an Artificial Intelligence,

Journals

IEEE Transactions on Evolutionary Computation

The Art of Designing Parallel Applications

iCSC2006 The Art of Designing Parallel Applications

Coordinator:

Marek Biskup – Warsaw University

The theme focuses on parallel programming, either local on a multiprocessor system, or distributed, with computers communicating over a network.

The topics covered by the theme are hardware for parallel programs, scheme of designing parallel applications, means of synchronizing threads and processes (mutexes, semaphores, monitors) with examples in C and Java, the underlying principle of RMI and RPC and their usage for distributed programs.

The theme also covers writing cross-platform programs, which is especially important for distributed systems. We will conclude with a presentation of a portable, parallel data analysis platform - the ROOT framework.

A few questions

- *Have you realized the growing importance of **parallel programming**?*
- *Do you know how to **distribute data** and computations among working nodes?*
- *Do you know how to **synchronize threads** in your application?*
- *Can you efficiently use **mutexes**, **semaphores**, **monitor objects**, **condition variables**?*
- *Can you write **concurrent software** in an object-oriented way?*
- *Are you aware of tools which can help you write **portable software**?*

All the answers in the Art of Designing Parallel Theme at **iCSC**

Overview

Slot	Lecture	Description	Lecturer
Tuesday 7 March 2006			
09-00-09-55	Lecture 1	**Parallel Computing**	Marek Biskup
10:05 - 11:00	Lecture 2	**Synchronizing Threads and Processes**	Marek Biskup
11:30 - 12:25	Lecture 3	**Design Patterns for Concurrent Objects**	Marek Biskup
12:30 - 14:00		Lunch	
14:00 - 14:55	Lecture 4	**Portable Programming**	Yushu Yao
15:05 - 16:00	Lecture 5	**Parallel Computing with ROOT and PROOF**	Marek Biskup Yushu Yao
16:30		Adjourn	

LECTURE 1

Parallel Computing

Tuesday 7 March 2006			
09:00 - 09:55	Lecture 1	**Parallel Computing**	Marek Biskup
		The objective of this lecture is to present the principles of Parallel Computing, and the key underlying techniques, with a special highlight on the relationships between hardware architecture and the corresponding software techniques. The lecture targets computer scientists interested in an overview of Parallel Computing, as well as more expert programmers. No specific knowledge of parallel computing is required but familiarity with parallel programming will help understanding some of the techniques presented in the second part of the lecture.	
		Supercomputers - Motivation - Users - Architectures - Example: BlueGene/L - Parallel Desktop Computers: multicore CPUs **Parallel Application Examples** - HEP Data Analysis **Designing a Parallel Application** - Methodology - Example - Successive Overrelaxation - Performance Metrics **Communication within a Parallel Application** - Shared Memory - Message passing - Synchronous and Asynchronous Communication - Transposition Table-driven Scheduling **Sources of Errors** - Deadlock - Race Condition - Message Ordering **Higher Level Communication** - MPI - RPC - Java RMI **Parallel Application Example - 15 puzzle solver** - Static Job Allocation - Dynamic Load Balancing	

Parallel Computing

Marek Biskup

Warsaw University

Outline

- **Supercomputers**
- **Hardware architectures**
- **Motivation for parallel computing**
- **Designing a parallel application**
- **Message passing**
- **Higher-level methods**
- **Examples**

High Performance Computing

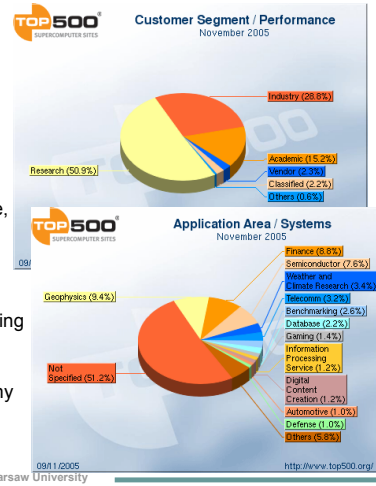
Human	**0.1 OPS**
First computer	**400 FLOPS**
Average PC ('05)	**2 GFLOPS**
The best CPU ('05)	**6 GFLOPS**

Weather forecast: 100 GFLOPS
LHC: a lot!

Cluster of 1000 PCs	**6 TFLOPS**
IBM Blue Gene/L (131k CPUs)	**280 TFLOPS**

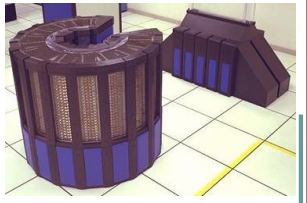
HPC users

- **Science**
  - Climate, astrophysics, biology (genetics), chemistry, material sciences, HEP
- **Engineering**
  - Crash simulations, earthquake, fluid dynamics, combustion
- **Business**
  - financial and economic modeling, transaction processing
- **Defense**
  - Nuclear weapons, cryptography

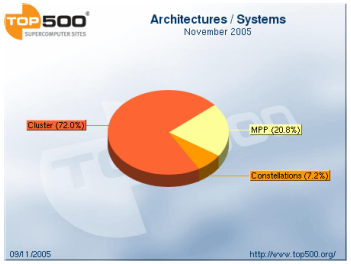


Hardware architectures

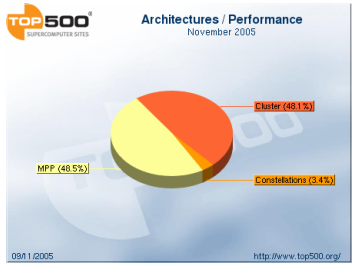
- **Common parallel architectures**
  - Commercially available for a reasonable price
  - Desktops (2-4 processors)
  - Multiprocessor servers (4-64 CPUs)
  - Clusters of workstations
- **Supercomputer architectures**
  - Vector processors
  - Distributed memory multiprocessors (MPP – massively parallel processing)
  - Constellations – clusters of supercomputers



Top500 architectures



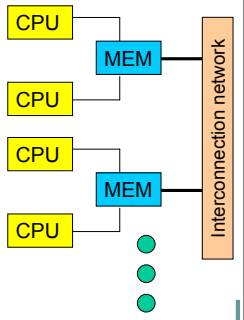
Most supercomputers are clusters



But MPPs are more powerful

Architectures in details

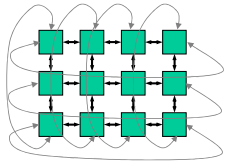
- **SMP – symmetric multiprocessing (up to 64CPUs)**
  - All processors access common memory on the same rights
  - Used in desktops
- **NUMA – nonuniform memory access**
  - Global address space (as in SMP)
  - Faster access to local memory
  - Slower to remote
- **Distributed memory multicomputers**
  - Communication via messages
- **Vector computers**
  - Multiple functional units performing the same operation on vector registers (very long ones)
    - E.g. vector addition, dot product
  - Almost disappeared



NUMA

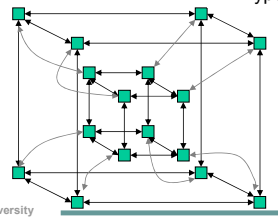
Network Topologies

- **Goal**
  - Limited number of connections per node
  - Small width
  - Scalability
- **Topologies**
  - Mesh
  - Ring
  - Torus
  - Hypercube
  - …



Torus

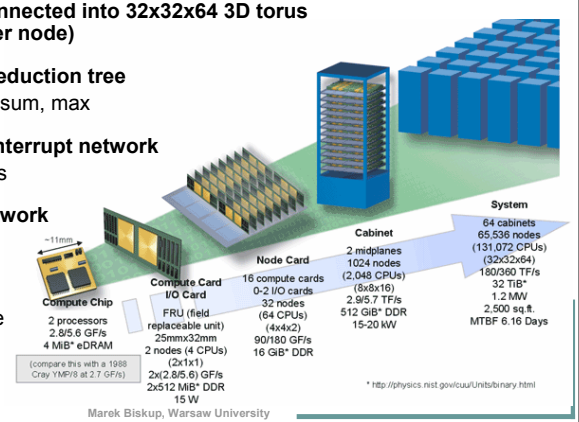
4D hypercube



IBM Blue Gene/L

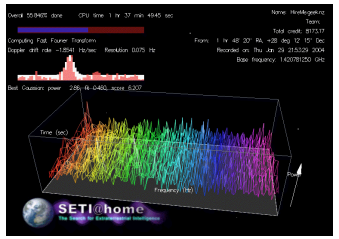
- **Nodes connected into 32x32x64 3D torus (6 links per node)**
- **+ global reduction tree**
  - Global sum, max
- **+ global interrupt network**
  - Barriers
- **+ data network**

The fastest computer in the world! … at the moment



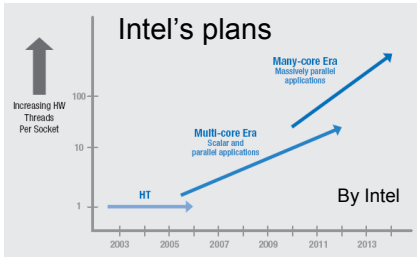
Seti@home-like computing

- **Idle computers can be used**
  - Application running as a screen saver
- **Only computational-intensive application**
  - no communication while computing
- **Very successful – people are willing to share their computing power**
- **LHC@home – started recently**
  - Testing stability of the beam (60 particles 100k loops)



Parallel systems at home?

- **Moore's law:**
  - transistors density will double every 18 months
- **Increasing the frequency is not possible forever**



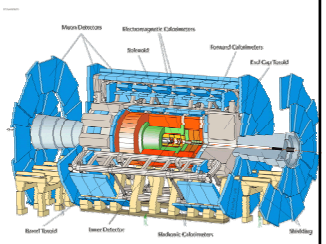
AMD: More than 2 CPU cores in 2007

• desktop computers with eight CPUs in five years time.
• efficient usage only with parallel applications.

Parallel application example – HEP

- **Detector Simulation**
  - Simulate 10000 events with Geant4
  - One event – (e.g.) 1 minute
    - One week of computations!

Trivial parallelization – give parts of all the events to different CPUs



- **Event analysis**
  - Plot a quick histogram of 10000 events
  - One event – (e.g.) 0.1s
    - 20 minutes – This is not a quick histogram!

Trivial parallelization – each CPU analyzes a part of all the events. At the end histograms are added.
But making it interactive and transparent is a challenge.

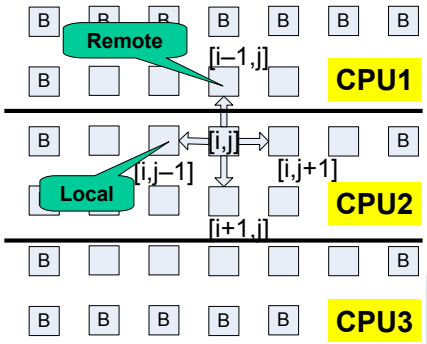
Nontrivial example – SOR

- **SOR – Successive overrelaxation**
  - Solution to the Laplace equation

$$A_{new}[i,j] = \frac{\alpha}{4}(A[i-1,j] + A[i+1,j] + A[i,j-1] + A[i,j+1]) + (1-\alpha)A[i,j]$$

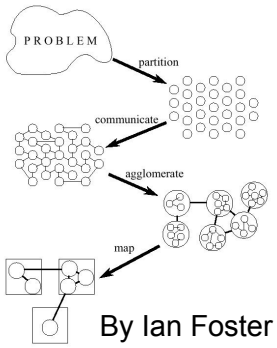
Boundary rows have to be transferred between CPU's

Nontrivial but easy



Designing a parallel application

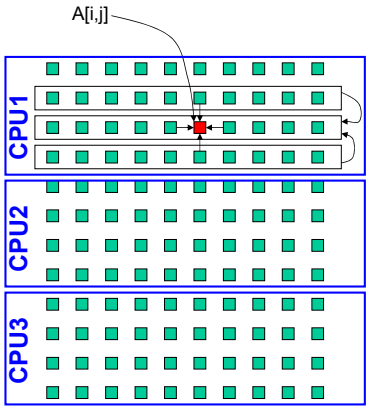
- **Partitioning**
  - decompose data and computations into small tasks
- **Communication**
  - Analyze communication required to coordinate tasks
- **Agglomeration – reduce communication**
  - Increase granularity, improve locality
- **Mapping – map processors to tasks**
  - Concurrent tasks on different CPUs
  - Frequently communicating tasks on the same CPU



By Ian Foster

Example – SOR

- **Partitioning**
  - A task is to compute A[i,j] for each step
- **Communication**
  - Transfer neighboring cells from other tasks
- **Agglomeration**
  - Take a whole row as a task
  - 2 times less communication per matrix element
- **Mapping**
  - Assign several rows to a single CPU



Designing, part 2

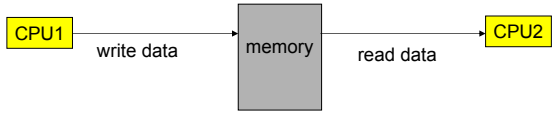
- **Minimize communication overhead**
  - Data locality is important
- **Load balancing**
  - Make sure processors are never idle
    - Dynamic load balancing: divide work at runtime
- **Take system architecture into account!**
  - Fast local and slow remote memory for NUMA machines
  - Hardware for broadcasting, reduction
  - Faster access to neighboring nodes

Performance metrics

- **Execution time:**
  - Time when the last processor finishes its work
- **Speedup:**
  - (time on 1CPU) / (time on P CPUs)
- **Efficiency**
  - Speedup / P
- **Note:**
  - always compare to the best sequential algorithm
    - 100% efficiency with parallel BubbleSort is not an achievement
- **We'd like to have 100% efficiency ☺**
  - But it is hardly ever possible ☹

Communication

- **Depends on systems architecture**
  - On SMP and NUMA machines memory is common to all processors
    - Use the memory to exchange data

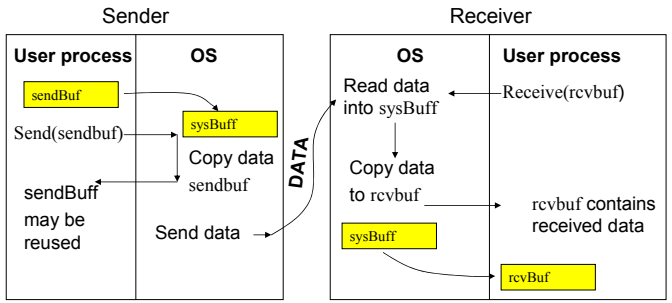


Access to common memory must be synchronized to prevent data corruption.

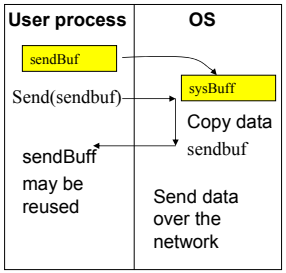
- Without global address space – exchange messages through the network

Communication – messages

- **General scheme – Message Passing**
  - Sender chooses the receiver and sends data

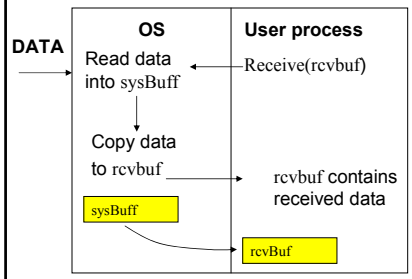


Sender



- **Large messages will not fit into sysBuf**
  - The sender will be blocked until most data has been sent

Receiver



- **Receiver is blocked until all data has been written into rcvbuf**

Synchronous communication

CPU1 - sender

```
int outBuffer[SIZE];
doComputations1();
send(CPU2, outBuffer, SIZE);
continueComputations1();
```

CPU2 - receiver

```
int inBuffer[SIZE];
doComputations2();
receive(CPU1, inBuffer, SIZE);
continueComputations2();
```

- **send blocks CPU1 until the message is sent**
- **receive blocks CPU2 until the message is received**

We'd better do something when waiting for messages!

Asynchronous communication

CPU1 - sender

```
int outBuffer[SIZE];
doComputations1();
id = sendAsync(CPU2, outBuffer, SIZE);
continueComputations1();
wait(id);
reuseOutBuffer();
```

outBuffer musn't be overwritten yet!

CPU2 - receiver

```
int inBuffer[SIZE];
doComputations2();
id = receiveAsync(CPU1, inBuffer, SIZE);
continueComputations2();
wait(id);
useInBuffer();
```

inBuffer not filled yet!

- **Both processes can continue immediately**
  - We have to make sure send/receive have finished before (re) using buffers

Sources of errors

- **Deadlocks**
  - Two (or more) processes are waiting for each other
- **Message ordering**
  - Asynchronous messages may arrive in different order then in which they have been sent
- **Race conditions**
  - When using shared memory CPUs may use partially written data

Deadlocks

PROCESS 1
send(CPU2, buf);
receive(CPU2, buf2);

PROCESS 2
send(CPU1, buf);
receive(CPU1, buf2);

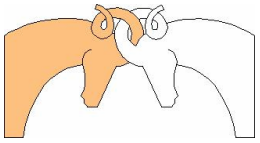
Large messages (bigger then the buffers' size)

Waits for PROCESS 2 to receive the message

This is a deadlock!

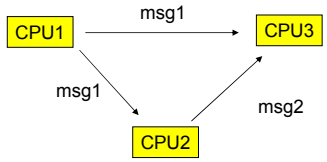
Waits for PROCESS 1 to receive the message

- **Solution**
  - Use asynchronous IO (more complicated)
  - Always be careful ☺



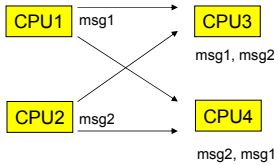
Message ordering

- **CPU1:**
  - sendToAll(msg1)
- **CPU2:**
  - rcvFromAny(msg1)
  - sendToAll(msg2)
- **CPU3:**
  - rcvFromAny(msg1)
  - rcvFromAny(msg2)



Wrong! msg2 may arrive before msg1!

- **Solutions**
  - Number messages
  - Specify CPU when communicating



Race condition

- **Traveling salesman problem:**
  - Given distances between cities
  - Find shortest path through all of them

Pseudo-code

```
global int min = +infinity;
int cpu = getMyCpuNumber()
while (path = getNextPath(cpu, min)) {
  length = getLength(path);
  if (length < min)
    min = length;
}
```

Current best path

Generates new path for this CPU.

cuts of partial paths longer than min

Execution:

minimum == 100;

CPU1: (length == 50)		CPU2: (length == 40)	
1. if (length < min)	// true		
2.		if length < min)	// true
3.		min = length;	// min = 40
4. min = length;	// min = 50		

WRONG!

Higher level methods

- **MPI – Message Passing Interface**
  - Interface for C and other languages
- **RPC – Remote Procedure Call**
- **Java RMI**
  - Object-oriented RPC

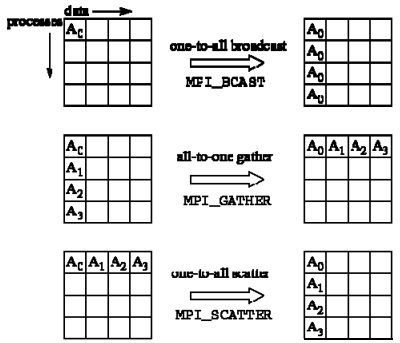
MPI – basics

- **Interface specification for message passing**
  - Many implementations (C, Fortran)
- **Basic operations:**
  - MPI_INIT: Initiate an MPI computation
  - MPI_FINALIZE: Terminate a computation
  - MPI_COMM_SIZE: number of processes
  - MPI_COMM_RANK: my process identifier
- **Communication**
  - MPI_SEND – send (sync. or async.)
  - MPI_RECV – receive (as above)
- **Global communication – broadcasting, reduction**
- **Subclusters within a cluster**
- **Tags used to distinguish various kind of messages**

```
MPI_INIT();

MPI_COMM_SIZE(
        MPI_COMM_WORLD, count);
MPI_COMM_RANK(
        MPI_COMM_WORLD, myid);
print("I am", myid, "of", count);

if( myid == 1)
  MPI_SEND(arr, 100, MPI_INT, CPU2, itag,
        MPI_COMM_WORLD);
else
  MPI_RECV(arr, 100, MPI_INT,
        MPI_ANY_SOURCE, itag,
        MPI_COMM_WORLD &status);

MPI_FINALIZE();
```


MPI – global operations

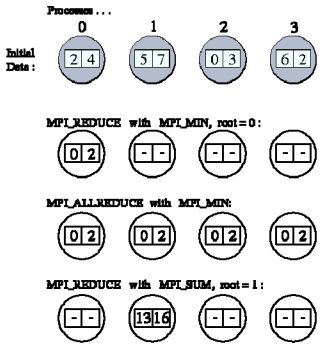
- **Data can be send to/received from many processes in one operation**
  - If the network doesn't support broadcasts implementing it on our own may be tedious
- **MPI_BARRIER**
  - Wait for all processes to reach this function



From Ian Foster's book

MPI – Reduction

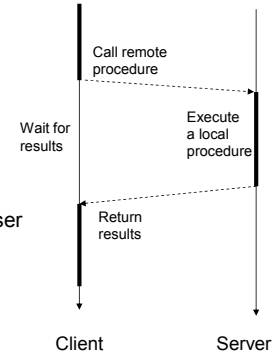
- **MPI_REDUCE**
  - Compute a simple function (min, max, sum, product, logical operations) on values on all the nodes
  - The result may be sent to a single process or to all the processes (ALLREDUCE)
- **Such reduction may be tedius to implement on ones own**
  - Naive algorithm: O(P) operations
  - Tree structure: O(logP)
  - Sometimes be implemented in hardware (BlueGene/L)



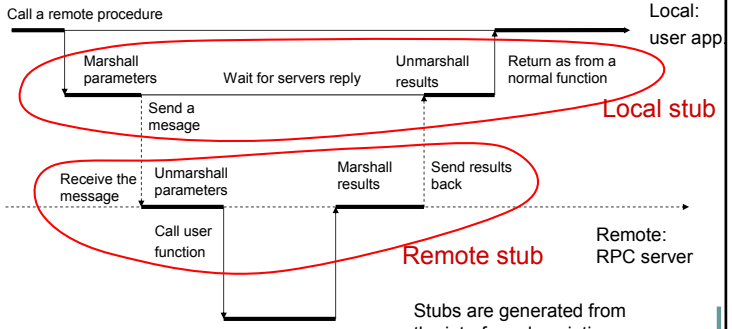
From Ian Foster's book

RPC

- **Like normal procedure call**
- **But executed on a remote host**
- **We need to:**
  - Specify the function interface
    - Special language (IDL)
  - Write its code
  - Generate client's and server's stub
  - Configure RPC server
  - Connect to the RPC server from the user application
- **Then we use remote procedures in the same way as local ones**



RPC - internals



Stubs are generated from the interface description.

Java RMI

- **Similar to RPC**
- **Object oriented**
- **Stubs are generated from java interfaces**
  - No additional language to learn
- **Methods for locating a remote object**

Example – 15-puzzle solver

2	3	10	4
1		7	8
5	9	12	15
13	6	14	11

Scrambled

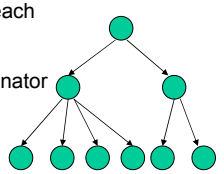
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Solved

- **Goal – find minimal number sequence solving the puzzle**
- **IDA* – iterative deepening depth-first search**
  - Try all the possible sequences with N moves by depth-first search
  - If not found increase the depth to N+1
  - Until found
- **DFS(board, N):**
  - if board is the solution return true
  - If N is 0 return false
  - For all possible moves
    - Apply a move to a copy of the board
    - DFS(newBoard, N - 1)

15-puzzle – Idea 1

- **Execute IDA* steps, one by one, in parallel**
  - Wait for the whole step to finish before examining a new depth
- **One coordinator**
  - Creates P jobs, one for each processor
- **P processes**
  - Get a job from the coordinator
  - Perform DFS
  - Return the result



Static load balancing will not work!

Some processor will finish their jobs while others are still computing.

	3	10	4
2	1	7	8
5	9	12	15
13	6	14	11

2 possible moves

2	3	10	4
1	9	7	8
5		12	15
13	6	14	11

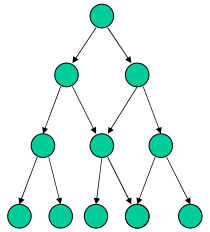
4 possible moves – much more positions to analyze

Dynamic load balancing

- **As before, but the coordinator generates much smaller tasks**
  - After a processor has finish its job it gets another one from the coordinator
- **Load imbalance becomes much smaller**
  - But it is still significant
- **Work stealing – when no more jobs:**
  - An idle process choses randomly another one
  - And asks it for a part of its job
- **Even better performance**

Transposition table

- **When searching huge number of positions some of them will reappear**
- **Information about positions searched should be kept in a transposition table**
  - Local:
    - Doesn't know about positions searched on other processors
  - Global:
    - Replicated on all nodes
    - Split over the nodes
- **Much more difficult**

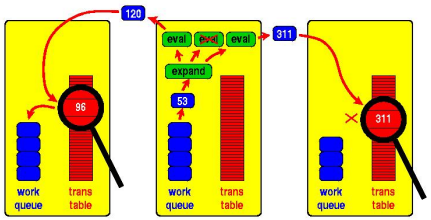


Double blank puzzle has much more transpositions

	3	10	4
2	1	7	8
5	9	12	
13	6	14	11

Transposition table scheduling

- **Each processor has its own transposition table**
- **Send the work to the transposition table (asynchronously) instead of evaluating it locally**
- **Asynchronous messages can be combined**



By Henri Bal

Much better scalability then other approaches to transposition tables

Summary

- **Parallel programs are inevitable when solving time consuming problems in reasonable time**
- **To write an efficient parallel program one has to be aware of the systems architecture**
- **Programming a parallel application requires much more knowledge and is much more error-prone**
  - Luckily there are standards and libraries that simplify this task
- **Multi-CPU desktops are appearing on the market**
  - One has to be prepared for writing parallel applications for a PC to make use of all its CPU power

CERN School of Computing

Further reading and materials used

- Hanri Bal – „Parallel Programming" course at the Vrije Universiteit, Amsterdam: http://www.cs.vu.nl/~bal/college05.html
- Ian Foster – „Desiging and Building Parallel software": http://www-unix.mcs.anl.gov/dbpp/
- IBM: RS/6000 SP: Practical MPI Programming
- Top500 site: http://www.top500.org/
- Wikipedia
- Nan's Parallel Computing Page: http://www.cs.rit.edu/~ncs/parallel.html
- Contact: mbiskup@mimuw.edu.pl

LECTURE 2

Synchronizing Threads and Processes

Tuesday 7 March			
10:05 - 11:00	Lecture 2	**Synchronizing Threads and Processes**	Marek Biskup
		The lecture will give an overview of ways and means of synchronizing threads and processes of a parallel application running on a multi-CPU system. It is meant for programmers designing multithreaded applications and willing to learn more about general techniques. It requires basic understanding of multiprogramming computer systems (threads, processes). Additional knowledge of Java will help in better understanding of the examples.	
		Introduction and Motivation - Concurrent Programs - Example: a Concurrent Web Server - Parallel Execution - Processes and Threads **Synchronizing Access to Memory** - Example of Memory Corruption - Critical Section - Hardware Support - the test-and-set Instruction **Means of Synchronization** - Mutexes - Bounded Buffer Example - Semaphores - Monitor Objects - Java Monitors - Condition Variables - Bounded Buffer with Monitor Object and Condition **Using Threads** - Java - Complete Bounded-Buffer example - C-Language Threads - the pthreads **The new Concurrent Java API** **Interprocess Communication in UNIX**	

Synchronizing Threads and Processes

Marek Biskup

Warsaw University

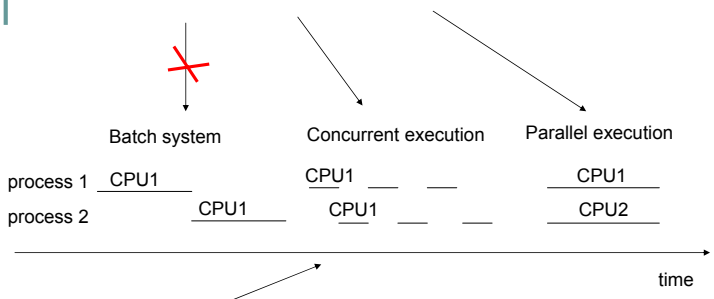
Outline

- **Why to synchronize processes and threads**
- **Means of synchronization**
  - Mutexes
  - Semaphores
  - Monitors
  - Condition variables
- **Threads in Java and C**

Motivation

- **Multiprogramming operating systems**
  - Single CPU may execute several programs (time sharing)
- **Some problems are intrinsically concurrent**
  - e.g. a web server
- **Hyper-threading technology**
  - Single processor executing a couple of threads at the same time
- **Multi-core CPUs emerging**
  - To achieve enough speed all processors have to be used

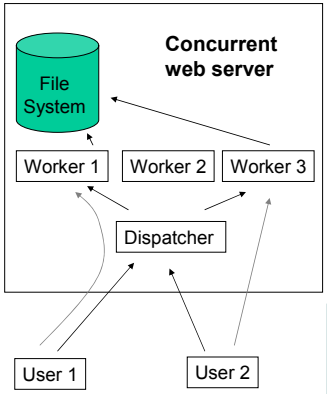
Concurrent programs



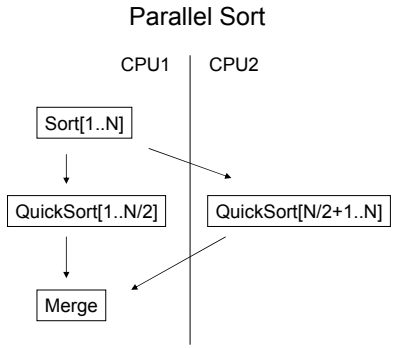
- **Time sharing**
  - several programs executed concurrently on a single CPU
  - a processor executes one program for certain time
  - and switches to another one

Example – concurrent web server

- **Before sending a web page the server has to read it from disk**
  - Synchronous IO with a single process would block the whole server
  - Solution:
    - Use asynchronous IO (a bit complicated)
    - Use multiple processes/threads
- **Each process serves one client:**
  - Reads the request from a socket
  - Reads a local file to be send
  - Writes results to the socket



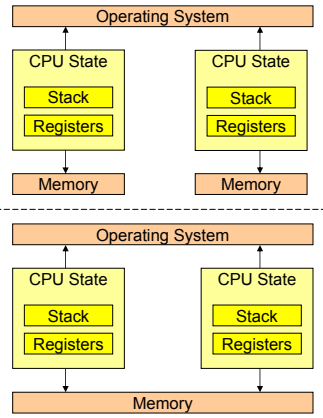
A parallel program



- **Problems**
  - Start a program on the second CPU
  - Exchange data between CPUs
  - Wait for results
- **With two processors: ~ 2x faster**

Multiprogramming systems

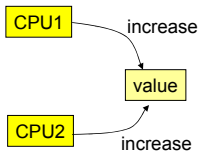
- **Processes**
  - Different CPU state
  - Different memory space
  - Communication via operating system mechanisms
    - Messages, shared memory (allocated using O.S.), Sockets
- **Threads**
  - Exist within a single process
  - Same (global) memory space
  - Different CPU state
  - Communication via memory
  - Faster switching



Accessing a memory location

- **Example: Two CPUs execute the same function: add one to a global variable**

```
int value; // global
void increase() {
   int x = value
   x++;
   value = x;
}
```



value == 1

CPU1:		CPU2:	
1. **int** x = value;	// 1		
2.		**int** x = value;	// 1
3.		x++;	// 2
4. x++;	// 2		
5.		value = x;	// 2
6. value = x;	// 2		

value is 2 instead of 3!

Accessing a memory location

- **The same example. Reading/writing an int (4 bytes) is byte-by-byte**
  - CPU1 reads the value (255: |0|0|0|255|)
  - CPU1 writes the new result: 256 : |0|0|1|0|, first the 3 upper bytes
  - CPU2 reads the value (511: |0|0|1|255|)
  - CPU1 writes the last byte (256: |0|0|1|0|)
  - CPU2 writes the result (512: |0|0|2|0)
- **We get 512 instead of 257!**
- **Luckily most architectures read whole ints at once**

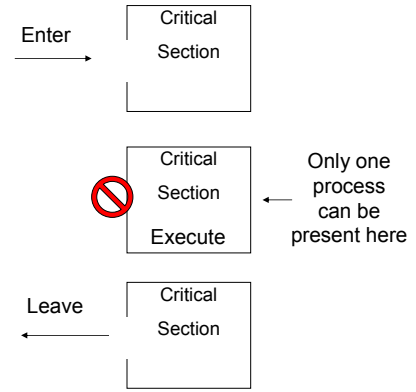
The access to a common variable must be blocked when it is accessed by a processor

Critical section

- **A region of code that can only be executed by one thread or process at a time**
- **Atomic execution (synonym) – execution of code that cannot be interfered with by another process**

```
int value; // global
void increase() {
  int x = value
  x++;
  value = x;
}
```

Critical section

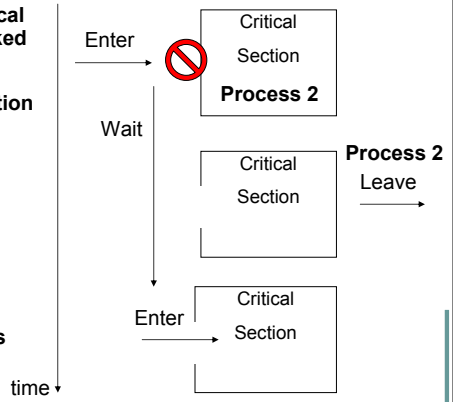


Critical Section 2

- **Process entering a critical section should be blocked until the section is free**
- **How to check if the section is free?**

```
bool lock = false;
while (lock == true)
  ; // empty instruction
lock = true;
  // critical section
lock = false;
```

- **Wrong! – two processes may check lock at the same time and enter**



Hardware support

- **The test-and-set instruction**
  - Executed atomically

This is sufficient to synchronize processes

```
bool testAndSet(int* lock) {
  // atomic instruction !
  if (*lock == false) {
    *lock = true;
    return false;
  }
  else
    return true;
}
```

Often called SpinLock

```
int value;
int lock = 0;
void increase() {
  while (testAndSet(&lock))
    ;              // just wait
  int x = value; // now lock == 1
  x++;
  value = x;
  lock = 0;      // release
}
```

Active waiting! – the CPU keeps on testing the variable.
On multiprogramming systems, the CPU should be assigned to another task.
O.S. support is needed.

Mutexes

Operations:

- **lock()**
  - If the mutex is locked
    - Release the CPU (sleep)
    - Wake up when the mutex is unlocked
  - Lock the mutex and pass over
- **unlock()**
  - Unlock the mutex
  - Wake up one process waiting for the lock (if any)
  - May be executed only by the process that owns the mutex (executed lock())

Atomic once the process is woken up

```
int value;
mutex m;
void increase() {
   m.lock()
   int x = value
   x++;
   value = x;
   m.unlock()
}
```

Critical section

Bounded buffer

- **Buffer of a fixed size**
- **Processes can read from it and write to it**
  - Readers should be blocked if nothing to read
  - Writers should be blocked if no space to write
- **put/get must be exclusive (critical section)**

```
mutex m;
int buffer[N];
int items = 0;
```

```
void put(int value) {
   m.lock();
   if (items == N)
      wait();    // ????
   buffer[items++] = value;
   m.unlock();
}
```

```
int get() {
   m.lock();
   if (items == 0)
      wait();    // ????
   int rv = buffer[--items];
   m.unlock();
   return rv;
}
```

How can we wait?
- use another mutex (complicated)
- introduce more advanced means of synchronization

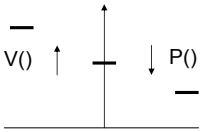
Semaphores

- **An object which holds a value *n* ≥ 0**
- **Operations**
  - Init(int v)
    - n := v;
  - V() – atomic
    - n++;
  - P() – atomic once ***n***>0 is detected
    - await n > 0;
    - n--;
- **Semaphores are considered nonstructural, obsolete, error-prone**
  - But they are useful

Bounded Buffer

```
semaphore empty(N);
semaphore full(0);
mutex m;
int buffer[N];
int items = 0;
```

```
void put(int value) {
   empty.P();
   m.lock();
   buffer[items++] = value;
   m.unlock();
   full.V();
}
```

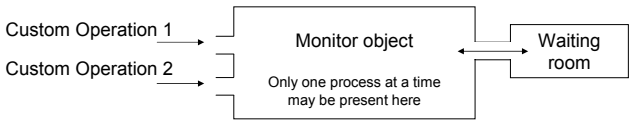



```
int get(int value) {
   full.P();
   m.lock();
   int rv = buffer[--items];
   m.unlock();
   empty.V();
   return rv;
}
```

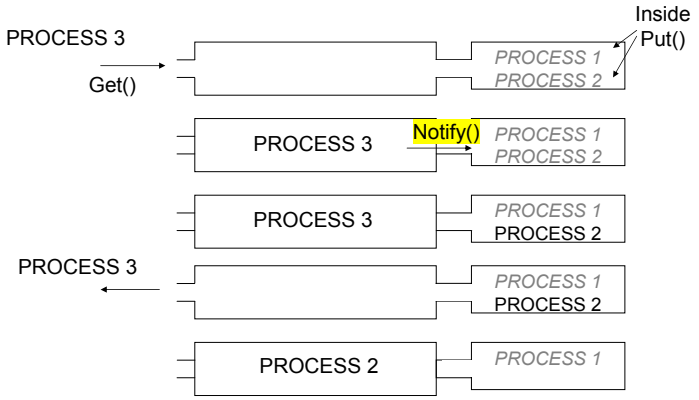

Monitor Objects

- **Previous example was rather complicated**
  - We'd prefer to use the lock-wait strategy as in the first trial with mutexes
- **Monitors – higher level synchronization**
- **Operations:**
  - Custom operations (fully synchronized)
  - Wait() – release the monitor and wait
  - Notify() – wake up one of the waiting processes
  - NotifyAll() – wake up all the waiting processes

```
void put(int value) {
   m.lock();
   if (items == N)
      wait();    // ????
   buffer[items++] = value;
   m.unlock();
}
```



Monitor Objects – Notify()



Java monitors

- **Exactly this kind of monitors has been implemented in Java**

```
public class BoundedBuffer {
  int A[], N; // buffer and its size
  int  k = 0;  // filled elements

  public BoundedBuffer(int size) {
    N = size;
    A = new int[N];
  }
  // custom operations
  public synchronized void put(int r) {…}
  public synchronized int get() { … }
}
```

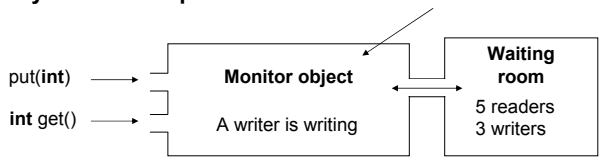
```
public synchronized void put(int r)
      throws InterruptedException {
  while (k == N)
    wait();            // full buffer
  A[k++] = r;
  notifyAll();
}
```

```
public synchronized int get()
      throws InterruptedException {
  while (k == 0)
    wait();            // empty buffer
  notifyAll();
  return A[--k];
}
```


Java monitors cont.

- **Why notifyAll() instead of notify()?**
- **Why this is not optimal?**

The buffer will be full after this write



Notify() would wake up just one process. Writer => deadlock!

NotifyAll() wakes up everybody in the waiting room

But only readers can continue!

We would prefer to have separate waiting rooms for readers and writers.

Condition Variables

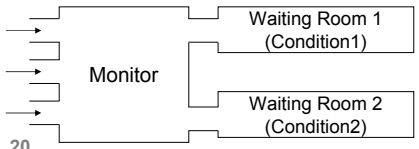
- **A monitor objects holds several *condition variables* – waiting rooms**
- **When waiting, a condition should be specified**
- **When notifying, a condition should be specified**
  - Wakes up only processes waiting on that condition

Pseudo–java code:

```
Condition readers, writers;
```

```
put(int r) {
  while (k == N)
    writers.wait();   // full buffer
  A[k++] = r;
  readers.notify();
}
```

```
int get() {
  while (k == 0)
    readers.wait();   // empty buffer
  writers.notify();
  return A[--k];
}
```



Bounded buffer with conditions

- **Real Java code:**
  - Monitor's lock must be managed explicitly (synchronized keyword doesn't work)
  - Condition variables are associated with that lock. In the wait method:
    - Lock is released when waiting
    - Lock is reacquired when waking up
  - Operation names: await(), signal() and signalAll()

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

public class BoundedBuffer {
  int N, A[], k = 0;
  private ReentrantLock lock = new ReentrantLock();
  Condition readers, writers;   // ...
}
```

The constructor:

```
public BoundedBuffer(int size) {
  N = size;
  A = new int[N];
  readers = lock.newCondition();
  writers = lock.newCondition();
}
```


Bounded buffer with conditions 2

- **Methods code**
  - Note: there is no **synchronized** statement (explicit locking)

```
public int get()
      throws InterruptedException {
  lock.lock();
  try {
    while (k == 0)
      readers.await();
    writers.signal();
    return A[--k];
  } finally {
    lock.unlock();
  }
}
```

lock must be locked when calling signal!

```
public void put(int r)
      throws InterruptedException {
  lock.lock();
  try {
    while (k == N)
      writers.await();
    A[k++] = r;
    readers.signal();
  } finally {
    lock.unlock();
  }
}
```


Java: how to create a thread

- **Create your thread's class:**
  - Inherit from the Thead class
  - Implement the run() method
- **Create such an object and call start()**
  - Calling start() will create a new thread which starts in the run() method
- **Use the constructor to pass initial data to the thread**
  - E.g. in the BoundedBuffer example all readers and writers have to know the buffer object

```
public class MyThread extends Thread {
  // local thread's data
  public MyThread(/* Parameters */) {
    // use constructor parameters to pass
    // initial data to the thread
    // e.g. global data structures
  }
  public void run() {
    // thread's own code
  }
}
```

```
myThread = new MyThread(globalData);
myThread.start();
```


Java: Complete Example

```
public class Producer extends Thread {
  BoundedBuffer buff;
  public Producer(BoundedBuffer b) {
    buff = b;
  }
  public void run() {
    try {
      for (int i = 0; i < 100; i++)
        buff.put(i);
    } catch (InterruptedException e) {}
  }
}
```

```
public class Consumer extends Thread {
  BoundedBuffer buff;
  public Consumer(BoundedBuffer b) {
    buff = b;
  }
  public void run() {
    try {
      for (int i = 0; i < 100; i++)
        System.out.println( buff.get() );
    } catch (InterruptedException e) {}
  }
}
```

```
public class TestBounderBuffer {
  public static void main(String[] args) {
    BoundedBuffer b = new BoundedBuffer(10);
    Producer p = new Producer(b);
    p.start();
    Consumer c = new Consumer(b);
    c.start();
  }
}
```

C language threads

- **Pthreads – POSIX threads standard**
  - Just API specification
  - Implementations available for various systems (windows, linux, other unixes)
- **Synchronization:**
  - mutexes
  - condition variables
- **Same semantics as in Java (almost)**

Datatypes:
```
pthread_t
pthread_cond_t
pthread_mutex_t
```

```
void* start_routine(void*);
```

Thread functions:
```
pthread_create(&thread,
        attr, start_routine, arg)
pthread_exit(ret_val)
pthread_join(thread_id);
```

Mutex functions:
```
pthread_mutex_init(mutex, attr);
pthread_mutex_destroy(mutex);
pthread_mutex_lock(mutex);
pthread_mutex_unlock(mutex);
```

Condition functions:
```
pthread_cond_init(condition, 0);
pthread_cond_destroy(condition);
pthread_cond_wait(condition, mutex);
pthread_cond_signal(condition);
pthread_cond_broadcast(condition);
```


New concurrent Java API

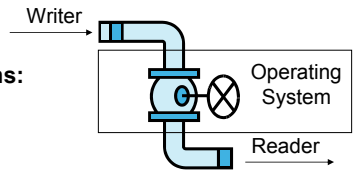
- **From Java 1.5 – new concurrent API: java.util.concurrent**
- **Atomic datatypes: java.util.concurrent.atomic**
  - AtomicBoolean
  - AtomicInteger
  - AtomicReference<E>
  - AtomicIntegerArray
  - Operations: get, set, getAndSet, compareAndSet, incrementAndGet
- **New means of synchronization:**
  - Semaphores
  - Condition, Lock
  - ReadWriteLock (allow many readers but writing is exclusive)
- **Synchronized datastructures**
  - BlockingQueue<E> – more or less our BoundedBuffer)
  - ConcurrentMap<K, V> – additional synchronized methods: putIfAbsent(), remove(), replace()

Interprocess communication in UNIX

- **Pipes**
  - Two file descriptors connected to each other
  - Created before forking the second process
- **Fifos**
  - Special file in a file system.
  - One process opens it for reading and the other for writing
- **Message queues**
  - Structured messages
- **Shared memory**
  - Processes can locate shared memory segments using *keys*
- **Signals**
  - One process may send a signal to another one
  - Breaks the normal execution of a sequential program to execute the signal handler
  - Very primitive way of communication

Pipes, FIFOs

- **„Virtual" file in memory**
  - One process writes data
  - The other reads data
- **Standard file access functions:**
  - read(), write()
  - fprintf(), fscanf()



- **Pipes**
  - Only for related processes (parent-child)
- **FIFOs**
  - For any processes
  - Can be found via the file system (special file)

Message queues, shared memory

- **Both reside within the kernel**
  - Can be found using O.S. calls
  - ID of the queue/shmem has to be known to the processes
- **Message queues:**
  - msgsnd()
  - msgrcv()

- **Shared memory**
  - Just read/write data as with threads
  - Don't forget about synchronization (semaphores, mutexes, etc.)!

Difficulties

- **Testing is not enough**
  - Problems with synchronization may occur very rarely. E.g. the famous AT&T crash ('90)
- **Deadlocks**
  - Two processes waiting for each other
- **Starvation**
  - Some processes may never be allowed to enter a critical section

Summary

- **Concurrent programs are inevitable for efficient usage of a multiprocessor system**
- **Threads and processes within a concurrent application have to be synchronized**
- **There are typical means of synchronization: mutexes, semaphores, monitors**
- **Other interprocess communication mechanisms are based on messages**
- **Writing and testing a concurrent application is much more difficult than a sequential one**

Further reading and materials used

- ***Concurrent Programming*** **course at Warsaw University**
- **Stevens –** ***UNIX Network Programming, Volume 2: Interprocess Communication***
- **Wikipedia**
- **Contact: mbiskup@mimuw.edu.pl**

CERN School of Computing

LECTURE 3

Design Patterns for Concurrent Objects

Tuesday 7 March 2006			
11:30 - 12:25	Lecture 3	**Design Patterns for Concurrent Objects**	Marek Biskup
		Design Patterns are recipes for writing maintainable and reusable object-oriented software. This lecture presents several patterns used for distributed and multithreaded programming. The lecture targets programmers interested in making their software more generic and reusable, especially those working on a multithreaded application. Also programmers interested in Event-Handling Techniques may benefit from it. Basic knowledge of threads, object oriented programming (classes, objects, methods, inheritance) is required. Additional knowledge of the classical Design Patterns might help but is not required.	
		Design Patterns - Overview - Example: Proxy pattern - Example: Abstract Factory pattern **Synchronization Patterns** - Motivation for Synchronization - Scoped Locking pattern - Locking in Java - finally statement - Tread-Safe Interface pattern - Synchronizing Proxy pattern - Read-Write Lock pattern - Double-Checked Locking pattern - Monitor Object pattern **Concurrency Patterns** - Thread Pool pattern - Tread Pools in Java - Leader-Followers pattern - Thread-Specific Storage pattern **Event-Handling Patterns** - Future pattern - Futures in Java - Events - Event-driven Programming - Event Loop - Reactor pattern - Proactor pattern - Java AWT Event handling	

Design Patterns for Concurrent Objects

Marek Biskup

Warsaw University

Outline

- **Design patterns**
- **Synchronization patterns**
- **Concurrency patterns**
- **Event-driven programming**
- **Example**

Design Patterns

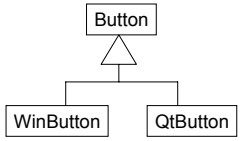
- **Some software design problems reappear in many application developments**
- **Successful solutions to such problems may be reused**
- **Design Patterns – documented reusable solution**
  - Not a finished design that can be transferred into code
  - Idea of how to solve a problem in a particular context
- **Mostly related to Object-Oriented design**
- **Give us common vocabulary to discuss O-O design**

Example – Proxy Pattern

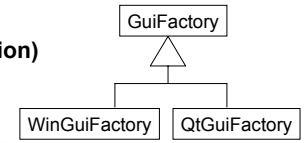
- **Proxy: An object acting as an interface to another object**
  - All methods of the original object but doing something ”in between”
- **CacheProxy**
  - Results of expensive operations are cached so that the next call for the same data will return immediately
- **RemoteProxy (e.g. JavaRMI stubs)**
  - Client calls a method of a local stub
  - The stubs sends method parameters to the server
  - The server executes the method
  - Results are returned from the stub as if the call were local

Example – Abstract Factory

- **Decouples creation of objects from the implementation**
- **Example: Bad GUI desing – code dependent on the platform**
  - Windows code:
    - Button b = new WinButton();
  - Linux code:
    - Button b = new QtButton();



- **AbstractFactory – platform independent code**
  - Button b = guiFactory.getButton();
- **Initialization (once in the application)**
  - guiFactory = new WinGuiFactory();



Synchronization

- **Access to the same memory by several threads has to be synchronized**
- **Example: increasing a value**

```
int value; // global
void increase() {
   int x = value
   x++;
   value = x;
}
```

- **Executed on two processors simultaneously may increase the value by one instead of two**
- **There are standard means of synchronization: mutexes (lock and unlock), monitors (as in java), semaphores**

Locking

- **Critical section has to be guarded by a lock**
- **When leaving the critical section the lock must be released, but:**
  - There are different execution paths
  - Exceptions may be thrown
- **Failing to release a lock results in a deadlock**
- **Such an error is difficult to spot**
- **And the code is difficult to maintain**

```
class Counter {
  bool increment(int) {
    lock.lock();
    if ( … ) {
      // ...
      lock.unlock();
      return false;
    }
    // ...
    lock.unlock();
    return true;
  }
};
```


Scoped Locking Pattern

- **Solution: Use a class that:**
  - Acquires the lock in the constructor
  - Releases the lock in the destructor
- **Then the lock will always be released**
- **Use that pattern even if you have a critical section inside a block:**

```
class MutexGuard {
  Mutex* lock;
public:
  MutexGuard(Mutex& aLock)
          : lock(aLock) {
    lock->lock();
  }
  ~MutexGuard() {
    lock->unlock();
  }
};
```

```
mutex lock;
void f() {
  // do whatever
  {
    MutexGuard(lock);
    // critical section
    // ...
  }
}
```

```
class counter {
  Mutex lock;
  bool increment() {
    MutexGuard(lock);
    // critical section
  } // guard will be released in ~counter()
};
```

Locking in Java

- **Use *synchronized* statement**
- **With an explicit lock: no local objects, so no destructors will be called**
  - Cannot use the Scoped Locking Pattern
- **Solution: the *finally* statement**
- **Finally block is always executed when leaving a try block:**
  - Normal leaving
  - After the return statement
  - When an exception is thrown

```
synchronized int aFunction() {
  // synchronized code
}
```

```
public int aFunction()
        throws MyException {
  lock.lock();
  try {
    if (A < 0)
      throw new MyException(A);
    if (A == 1)
      return 1;
    A++;
    return 0;
  } finally {
    lock.unlock();
  }
}
```


Intra-Component Call

- **Often there is need to call another method from the same component (class)**
- **If component methods are synchronized the same lock will be acquired twice**
  - Deadlock if the lock is not reentrant
  - Unnecessary overhead with reentrant locks

```
class buffer {
Mutex lock;
public:
  void put(int a) {
    MutexGuard(lock);
    A[num++] = a;
    if (num == N)
      flush(); // deadlock!
  }
  void flush() {
    MutexGuard(lock);
    // write the contents to a file
  }
};
```


Thread-Safe Interface

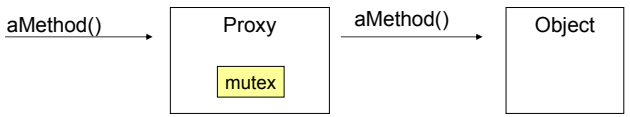
- **Structure the component into two layers**
  - Interface methods – acquire a lock and call an implementation method
  - Implementation methods – assume that the lock has already been acquired and do the job
- **Implementation methods may only call other implementation methods (and not interface ones)**
- **But:**
  - Still be careful when calling other component methods!
  - It adds a bit of overhead (more methods, twice more function calls)

```
class buffer {
Mutex lock;
public:
  void put(int a) {
    MutexGuard(lock);    putImp();
  }
  void flush() {
    MutexGuard(lock);    flushImp();
  }
protected:
  void putImp() {
    A[num++] = a;
    if (num == N)
      flushImp(); // implementation
  }
  void flushImp() {
    // write the contents to a file
  }
};
```


Synchronizing Proxy

- **Synchronize access to an object without changing its code**
- **Solution: Synchronizing Proxy Pattern**
  - The proxy contains:
    - all the methods from our object
    - a mutex
  - Call the proxy instead of the object itself
  - Proxy synchronizes access to the object

```
Proxy::aMethod() {
  mutex.lock();
  object.aMethod();
  mutex.unlock();
}
```



Read-Write Lock Pattern

- **Consider a concurrent dictionary:**
  - Many threads can read at the same time
  - Exclusive access when writing
- **Better locking strategy – a lock with two operations:**
  - readLock(); – blocks when the write-lock is taken
  - writeLock(); – blocks when the read or write-lock is taken

  ReadWriteLock in Java
- **Starvation problem:**
  - When readers keep on coming, writers will wait forever
- **Solution:**
  - e.g.: don't allow readers when a writer is waiting

Initializing a value

- **Think of a value that should be initialized when accessed for the first time**

```
// no synchronization

class Foo {
  private Helper = null;
  public Helper getHelper() {
    if (helper == null)
      helper = new Helper();
    return helper;
  }
  // other functions and members...
}
```

Will not work with multiple threads

```
// too much synchronization (?)

class Foo {
  private Helper = null;
  public synchronized Helper getHelper() {
    if (helper == null)
      helper = new Helper();
    return helper;
  }
  // other functions and members...
}
```

Every access is synchronized, even after the object has been created

Double Checked Locking

- **Can we do it better?**

```
// double checked locking optimization

class Foo {
  private Helper = null;
  public Helper getHelper() {
    if (helper == null) {
      synchronized(this) {
        if (helper == null)
          helper = new Helper();
      }
    return helper;
  } // ...
}
```

Double check: several processes may enter here

Scenario:

Thread A:

```
helper == null;
helper = new Helper();
```

compilers may assign the pointer value before **A** has finished executing the constructor!

Thread B:

```
helper != null; //!
useHelper  // not initialized!
CRASH!
```

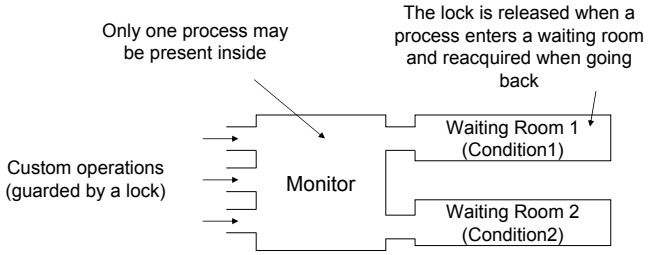

D.C.L. is an AntiPattern

- **In most programming languages (java, c++) using D.C.L optimization may result in a crash**
- **Correct code is the previous one:**

```
// correct code

class Foo {
  private Helper = null;
  public synchronized Helper getHelper() {
    if (helper == null)
      helper = new Helper();
    return helper;
  }
  // other functions and members...
}
```

Monitor object

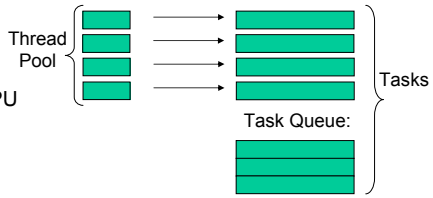
- **As presented in the previous lecture:**
  - a lock that synchronizes access to the object
  - Condition variables for waiting for specific events
  - Notify() wakes up waiting processes



Thread Pool Pattern

- **Steady stream of tasks to be performed, e.g.**
  - A web server handling many clients
  - A database server
- **Single thread:**
  - Not optimal on multiCPU systems
  - Asynchronous IO is difficult
- **Thread per request**
  - Simplifies programming
  - Creating a thread is an expensive operation

Avoid the expense of creating a new thread by reusing existing ones.



•A thread that finished a task takes another one from the queue.
•If there are no more tasks the thread may terminate or sleep.

Thread Pool in Java

- **ThreadExecutor interface**
  - execute(Runnable command)
- **Decouples submitting a task and running it**
- **Several executors Possible:**
  - Direct – just starts the task (synchronously)
  - Thread per task – creates a new thread for each task
  - ThreadPoolExecutor
    - corePoolSize – minimum # of threads
    - maximumPoolSize – maximum # of threads
    - keepAliveTime – threads wait at most this amount of time

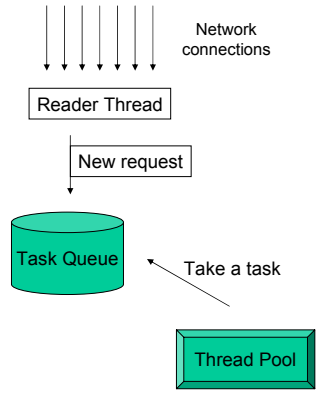
```
Executor executor = anExecutor;
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2());
```

```
class DirectExecutor
        implements Executor {
  public void execute(Runnable r) {
    r.run();
  }
}
```

```
class ThreadPerTaskExecutor
              implements Executor {
  public void execute(Runnable r) {
    new Thread(r).start();
  }
}
```


Multithreaded server

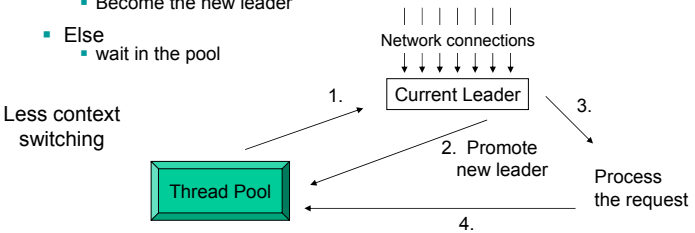
- **Pool of threads**
- **Each new request is taken by an available thread**
- **A possible solution:**
  - One thread reads request from the network (UNIX *select* call)
  - Creates a request and passes it to a working thread
- **Problems:**
  - Passing request between threads requires frequent context switching and synchronization
- **Can we do it better?**



Leader-Followers pattern

- **Threads take turns in reading request**
- **After reading a request:**
  - Wake up another thread to read the next request
  - Process the request
  - If there is no leader
    - Become the new leader
  - Else
    - wait in the pool

```
loop {
  ReadARequest();
  ThreadsPool.Notify();
  ProcessTheRequest();
  if (there is another leader)
    ThreadsPool.Wait();
}
```



Global values

- **The C errno value**
  - Global for the whole application
  - Is set after a system function's call has failed
- **When there are threads the value might get overwritten in another thread**
- **Possible solution: protect the value with a lock**
  - System function will acquire a lock when an error appears
  - User application will read the value and release the lock
- **Very bad a solution**
  - It is difficult to change system functions
  - A programmer may forget to release the lock – deadlock

Such a problem appears in other legacy systems as well.

Thread-specific storage

- **Introduce a global access point for such a global variable**
- **But keep separate value that variable per thread**
- **Values can be associated with threads**
  - Java: Thread class:
    - public static Thread currentThread()
    - When subclassed values can be kept in the Thread object
    - ThreadLocal object – a value, different for each thread
  - C (pthread):
    - pthread_t pthread_self(void);
    - Gives the thread's ID. ID's can be mapped to objects.
    - pthread also has functions for associating values with keys for a thread

```
#define errno (*(__errno()))
```

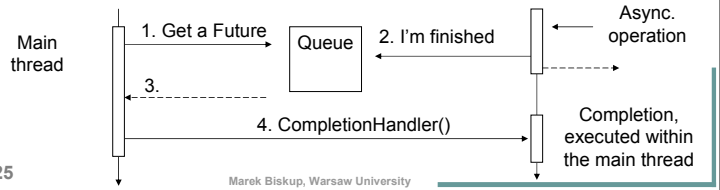
```
int *__errno() {
  // find the pointer p to thread-specific errno
  return p;
}
```


Future Object

- **Some operations have to be processes asynchronously**
  - Using a Thread Pool
  - Via the operating system (async. IO)
- **Future Object encapsulates**
  - The asynchronous operation's code
  - Completion (called after the operation has been finished)
- **All code at the same place**
- **Example: web server serving a client**
  - Asynchronous operation: reading file from disk
  - Completion: Invoke another async. operation – send the file.
    - Another completion: e.g. close the connection

Future Object cont.

- **At some point one has to use the result from the asynchronous IO.**
- **How to check if the task is finished?**
  - call isDone() method of the Future
  - call get() method (waits for the result)
  - Use a queue of finished Futures
- **Queue**
  - The future at the end of its asynchronous operation puts its identifier into a queue
  - The main task waits on the queue for finished tasks

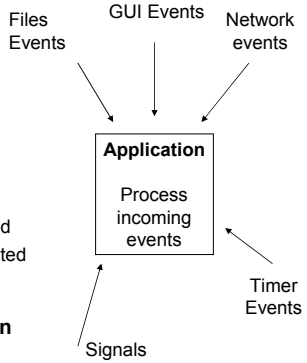


Futures in Java

- **Interface Future<V>:**
  - **boolean** cancel()
  - V get()
  - **boolean** isDone()
  - **boolean** isCancelled()
  - Additional methods may implement the completion
- **Class FutureTask<V>; additional methods:**
  - run() – the async. operation itself
  - done() – called when the state turns to *done*. May be overridden e.g. to notify the main thread
  - Additional methods for completion may be added when subclassed
- **FutureTask may be executed using Executors (e.g. ThreadPoolExecutor)**

Events

- **External „message"**
  - GUI event
    - The user clicked on a button
    - The user moved the mouse pointer
  - Network event
    - New network connection is coming
    - New data is arriving
    - More data can be sent right now
  - Operating system's events
    - Application is going to be terminated
    - Asynchronous IO has been completed
- **Or internal message, from another thread/process within the application**



Event Handles

- **Each event has an associated Event Handle**
  - An identifier of the event source (network connection, open file, GUI widget, etc.)
  - Usually provided by the O.S
- **Examples:**
  - For a network connection, files: descriptor number
  - For GUI: widgetID
  - Timers: timerID
- **There may be several kinds of events for one handle**
  - GUI widget:
    - MouseClicked
    - MouseMoved
  - For a file descriptor
    - Read event – more data to read
    - Write event – more data to write

Event-driven programming

- **Normal model of programming:**

```
Initialize();
DoComputations();
Terminate();
```

- **Event-driven programming**
  - Events are processed in a loop (so called „Event Loop")
  - When no events no computations are performed

```
Initialize();
while ((e = GetEvent()) != 0) {
  HandleEvent(e);
}
Terminate();
```


Event Loop

- **The Event Loop is often provided by the O.S.**
- **Event handler – user defined function that handles an event**
  - Has to be registered for a specific event type
  - The O.S. will call back the event handler when a specific event appears
- **Inversed program's logic**
  - No main loop
  - Just event handlers should be written (+ the main initialization function)

Pseudo code:

```
void SocketHandler() {
  // read the incomming data
}
void ExitHandler() {
  OS.StopEventLoop();
}
void main() {
  socket s = OpenConnection();
  OS.registerEventHandler(
    EXIT_EVENT, ExitHandler);

  OS.registerEventHander(
    SOCKET_EVENT, s,
    SocketHandler);

  OS.EventLoop();
}
```

callback functions (event handlers)

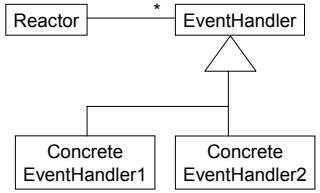
EventType

EventHandle

Reactor pattern

Object-oriented way of implementing the event loop

- **EventHandler – abstract class**
  - Owns a handle
  - HandleEvent() method
- **Concrete Event Handlers inherit from EventHandler class**
- **Reactor**
  - EventLoop()
  - RegisterHandler(handler)
  - RemoveHandler(handler)



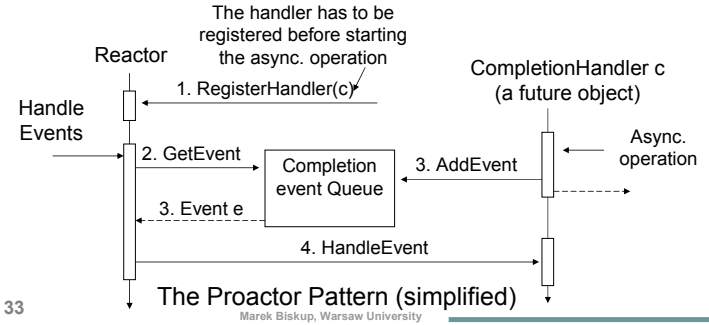
Loop:
- Get handles from all registered handlers
- Wait for an event from one of the handles (using the O.S)
- Call EventHandler->HandleEvent() for the handle returned by the demultiplexer

Java AWT – Event Handling

- **The Event Loop runs in a separate thread**
- **EventHandlers: EventListener interface**
  - Separate interfaces for different events
    - *MouseListener* for mouse events
    - *KeyListener* for keyboard events
- **Registering handlers to widgets:**
  - aWidget.addMouseListener(aMouseListener);
  - aWidget.addKeyListener(aKeyListener);
- **Handling events:**
  - Appropriate method of a listener is called:
    - e.g. mousePressed(), mouseEntered()
  - Each method takes a parameter: EventObject
    - MouseEvent for mouse events (mouse position, button state, etc.)

Handling asynchronous operations

- **Reactor Pattern can be used for handling asynchronous operations**
- **Remember the Future pattern?**



The Proactor Pattern (simplified)

Summary

- **Design Patterns are successful solutions to common design problems**
- **There are useful patterns for concurrent programs**
- **Synchronization patterns: Scoped Locking, Thread-Safe Interface, Synchronizing Proxy, Read-Write Lock, Monitor**
- **Concurrency Patterns: Thread Pool, Leaders-Followers, Thread-Specific Storage**
- **Event-Handling Patterns: Future, Reactor, Proactor**

Further Reading and materials used

- **Schmidt, Stal, Rohnert, Buchmann – *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2***
- **Gamma, Helm, Johnson, Vlissides – *Design Patterns: Elements of Reusable Object-Oriented Software***
- **Java standard library**
- **Wikipedia**
- **Contact: mbiskup@mimuw.edu.pl**

LECTURE 4

Portable Programming

Tuesday 7 March 2006			
14:00 - 14:55	Lecture 4	**Portable Programming**	Yushu Yao
		As the variety of computer hardware and software systems grows quickly, it becomes more and more important for a developer to design and develop portable applications, so that with the minimum cost, the maximum number of user could benefit from it. This Lecture begins with an introduction to the portable application design / development process in the software engineering point of view, followed by some practical guidelines of portable C programming. The lecture targets the programmers who need his applications to run on different platforms while don't have many experiences doing it. Basic knowledge of C is needed for the second part of the lecture.	
		Basics of Portable Programming -Definition of Portability -Measuring Portability -How to Achieve Portability in the Software Process **Portable Programming With C** **Portable Graphics User Interface** **An Example to Both Portable and Parallel Computing Project**	

Portable Programming

Yushu Yao
Univ. of Alberta
Canada

Inverted CERN School of Computing, 6-8 March 2006

Index

Part 1

Basics of Portable Programming

What is Portability?

- **Portability:**
  - Property of an application which permits it to be mapped from one platform to a different platform.
  - Portable means the **cost** to transport and adapt it to a new platform is less than the cost of redevelopment.
  - A platform (environment) can be a combination of hardware and software system.
  - The application must maintain the uniform operating characteristics and the same designed function on different platforms to be portable.

Why Portable?

- **For the Developer:**
  - One tantalizing feature. **Gain more users**.
  - **Save the cost** (time and money) of redeveloping the application for a new platform.
  - Survive the frequent software and hardware update.
  - **Extend the area of testing the software.** Some bugs may be more obvious on some platforms than other.
  - **Benefit the development process:** it have to be cleaner, more systematic, more disciplined, and more readable. Hence the reliability, adaptability, maintainability, and overall quality will be improved

Why Portable?

- **For the User:**
  - More Convenient for the user. Choose the favoriate system to run the application
  - Not restricted to use on one computer or one kind of System.
  - A More Reliable System for the Users

What Can you port?

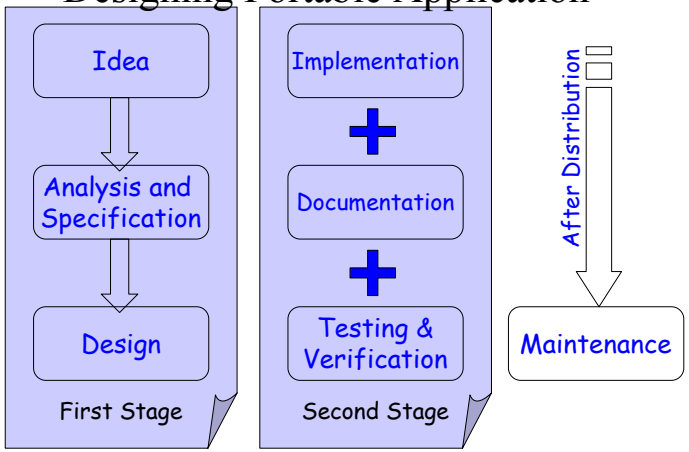
- **Binary Portability**
  - Porting the Executables
  - Least Modification
  - Must be very similar platform
- **Source Portability**
  - Need to reinterpret or recompile.
- **Intermediate**

Evaluating Portability

- **Degree of Portability**

  DP = 1- (Cost to port / Cost to redevelop)
- **If DP>0, we'd better redevelop it.**
- **If DP=1, perfect portability.**

Designing Portable Application



Analysis and Specification

- **Requirement Analysis of the Application**
  - What kind of function does the application provide. How does it interact with the users, etc.
- **Write down the requirement specification**
  - E.g. "User can view a text file in a window and can scroll up and down with a keyboard or other similar device"

Portable Analysis and Specification.1

- **"Use Ctrl+ right mouse click to open a new window of 240*480 pixel in size"**
  - Mac Doesn't have right mouse button
  - Window size might be larger than the Screen?
- **A More Portable Specification:**

  "Use a keyboard and pointing device event combination to Open a new window of the size of half the screen"

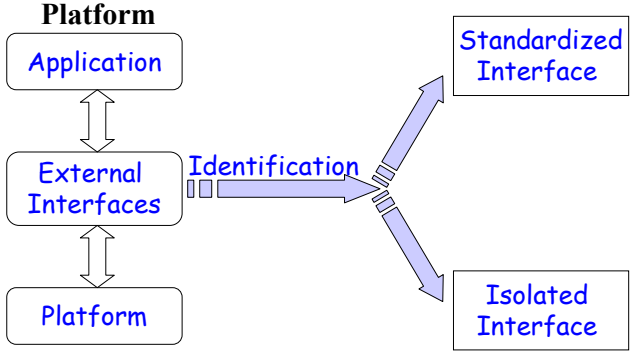
Portable Analysis and Specification.2

- **"The application shall be easily ported to new platforms"**
  - This specification means no meaning.
- **A More Portable Specification:**

  "This application shall easily be ported to any computer environment supporting at least 16-bit color, 128M memory, and with certain program library installed"

Portable Design

Interfacing between Application and Platform



Portable Design

Interfaces

- **What resources and services does this application access?**
- **Which of these services is not provided by the target platform?**
- **For the interfaces,**
  - Use Standards, or
  - Isolation / Encapsulation

Portable Interfacing 1

Use of Standards

- **The Standards:**
  - Provide a complete and unambiguous specification for a significant interface or subset, in a form suitable for the software to be developed
  - Have implementations which are widely available or may be easily developed for likely target environments
  - E.g.
    - The Character Set, use Unicode?
    - Communication protocols?
    - POSIX when writing an operation system
- **If No Standards available …**

Portable Interfacing 2

Isolation

- **Isolate all the access to this interface to a very small number of modules.**
- **And other modules access the interface by calling this module.**
- **Adaptation work needed only for this module when porting**

Portable Implementation

- **Implementation transform a design into a working application.**
- **The Principal Guidelines:**
  - Choose a portable language
  - Follow a portability discipline
  - Apply Standards Carefully

Portable Implementation 1
Choosing Language

- **Choose a Well-Standardized, Widely Available language.**
- **Does the language work well with the identified interfaces? For all the target platforms?**
- **Use the portable subset of the language.**
  - E.g. blurring data types in C is allowed, but highly non-portable. Will behave very different in different system.

Portable Implementation 2
Applying the Standard

- **Try to stick to the standard, follow the structure of the standard when accessing the interface.**

Portable Implementation 3
The Remedy

- **The Remedy:**

  Always keep "Portable" in mind when writing codes.

Portable Testing and Documentation

- **Testing**
  - Develop a reusable test plan
  - Learn from Errors
  - Test the Portability
- **Documentation**
  - Identify and separate system-dependent and independent portion into distinct sections
  - Only the system-dependent part need to be redeveloped

Part 2

Portable Programming With C

Portable Programming

- **Write Portable Code First, Worry about the platform-specified stuffs when porting.**
- **Separate platform-dependent and platform-independent source codes.**
- **Don't Depend on Non-Portable Libraries**

Portable C Programming

- **Points to Be Stressed:**
  - Follow the a Standard Coding Style
  - Use of Version Control System
  - Use of Compiling Tools, (Make/libtool…)
  - Use Tests
  - Handle Errors
  - Remember Portability

Indian Hill C Style

- **The Style for:**
  - Files
    - Naming Files, File Extensions
    - The layout of a Program or Header File
  - Declarations and Naming Conventions
    - Function
    - Variable
  - How to Add Comment to File
  - How to format a File
    - Indents, Whitespaces, whereto put "{", etc.

Indian Hill C Style

- **The Style Helps to:**
  - Make to code more Readable to others
  - Help yourself remember what you did
  - Keep everything clean and organized
  - Minimize Ambiguities

Version Control Programs

- **Enforce self-discipline**
- **Allow group cooperation**
- **Keep track of the history**
- **Easier Porting**

- **Example, CVS**

Some Practical Problems

- **File Names**
- **Data Types**
- **Data Files**
- **Floating Point**
- **Pointers**

File Names

- **No white space in a file name**
- **No special characters except "_"**
- **Length of the File Name can't be too long**
- **Format of a Pathname is different for different systems:**
  - C:\Windows\win.ini
  - /etc/passwd

Data Types

- **Length of the same data type maybe different in different system**
  - *E.g. int* maybe 16-bit or 32-bit

Solution: You can use your own data types (int, long, string, etc), and *typedef* them in the platform-dependent part.

- **Exact Data Types are Defined in <stdint.h>, or <sys/types.h>, depending the system**
- **Look into <limits.h> for the Max Values of the System.**

Pointers

- **Size of different kind of pointers may be different.**
- **Even if they are the same, the format maybe different.**
- **Casting Pointers is dangerous.**
  And works differently for different system.
- **Null Pointers**

Data Files

- **Binary File Endianness:**
  - How the number 1245391901 is stored?
  - 4A 3B 2C 1D - Big-Endian
  - 1D 2C 3B 4A - Little-Endian
  - Use wrapper to Read/Write binary files
- **File Per program/process environment variables.**
- **Support of renaming or unlinking open files ⇒ Close files before doing suspicious things to them.**

Floating Numbers

- **The Floating Number is "Floating"**
- **Comparing Float Numbers**
  - Never Use ***(A==B),*** because many systems don't do what you expected.
  - Instead, Use ***abs(A-B)<1e-6***
    If 1e-6 is not accurate enough, use a smaller number, e.g. 1e-30
    But Don't be smaller than the min-value of the float type.
- **Look into <float.h> for those values.**

Graphics User Interface
Choice 1

- **Find a Portable GUI Library**
  - wxWidget, QT, Tcl/Tk …
  - Reduce the Redevelopment Cost for Different Systems.
  - But the portability rely on the portability of the GUI Library. And when the library crashes on a specific system, have to wait until it is fixed. If choose to go around it for this system, then the platform-independent discipline is broken.

Graphics User Interface
Choice 2

- **Redevelop a UI for each target platform**
  - Carbon for MacOS
  - X-Windows for Unix Like Systems
  - MFC for Windows
- Certainly more work to do. However, could be minimized by a good system design.
- Much higher performance because of the use of the best fit UI for each target system.

About Java

- Java is **not** a portable programming language.
- Java is a platform itself.
- However, Java still need to run on a platform. So the platform specific features are not easy to achieve.

Other Ideas?

- **Scripting**
- **Perl / Python**
- **Emulate Unix, Cygwin**
- **Web-Based Application**

SETI@HOME, An Example

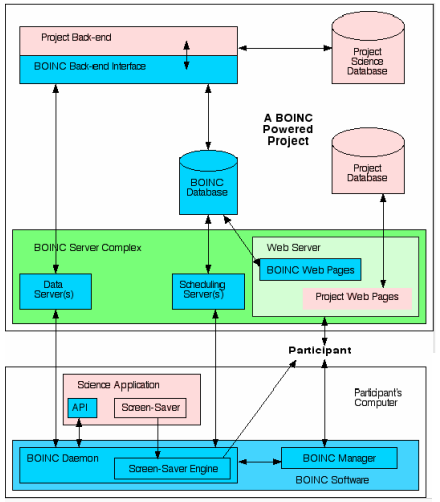
BOINC

Berkeley Open Infrastructure for Network Computing (BOINC)

- **Open Source Platform**
- **Support Distributed Computing Projects**
- **Using Volunteered Computer Resource**
- **Utilize idle resources**

Distributed Computing

- The tracking of the Work Units, make sure all of them are processed.
- Result Validation to ensure that the processing did produce a Valid Result.
- Result integration
- Communication connections so that the work can be distributed and the results collected.



BOI
NC
Syst

BOINC Components

Desi

Project Specific

BOINC System Design

- **Mostly Platform-Independent**
- **The Client UI is developed separately for different platforms, using the interface from Core to UI**
- **The Project Programmer don't need to worry about the Client**

References

Portability Basics

- Mooney, J.D. **Bringing Portability to the Software Process**
- Mooney J.D. **Strategies for Supporting Application Portability**
- Sommerville I. **Software Engineering** (5th ed.)

References

Portable Programming

- Porting C/C++
  - http://www.mozilla.org/hacking/portable-cpp.html
  - http://sources.redhat.com/autobook/autobook/autobook_toc.html
- Recommended C Style and Coding Standards
- A. Dolenc, Notes on Writing Portable Program in C

CERN School of Computing

Thanks To:

- CSC Management Team and all CSC2005 Lecturers
- The EU Funding
- The ***Mary Louise Imrie Graduate Student Award***, University of Alberta
- ***François*** for the valuable suggestions
- ***Wingyin Leung***
  -My Beautiful and Supportive Fiancée

LECTURE 5

Parallel Computing with ROOT and PROOF

Tuesday 7 March 2006			
15:05 - 16:00	Lecture 5	**Parallel computing with ROOT and PROOF**	Marek Biskup Yushu Yao
		The lecture consists of two parts. In the first one Yushu Yao presents the ROOT framework as a platform for portable programming. In the second part Marek Biskup describes the PROOF subsystem, which allows parallel data analysis with ROOT. The lecture is intended for physicists interested in speeding up their analysis by spreading calculations on a cluster of computers. It will also be interesting for computer scientists willing to learn the internals of a parallel application. No knowledge of the ROOT framework is required but the vocabulary used in the first four lectures may be helpful for understanding all details.	
		Introduction to ROOT - ROOT Concepts - ROOT Data Structures, Trees and Files **A Step-by-Step Example for ROOT beginners** - Create and Fill Trees - Histogram - Curve Fitting **PROOF** - Motivation - Architectures - Computations **Parallel Analysis with Proof** - Selectors - Selectors Example **Other PROOF Modes** - Sequential Mode - GRID Mode	

Introduction to Root

Yushu Yao
Univ. of Alberta
Canada

Inverted CERN School of Computing, 6-8 March 2006

Index

- **What is Root**
- **From Paw to Root**
- **Root Concepts**
  - Root and CINT
  - Root Framework
  - Root Data Structure
- **Tasting Root - A Step by Step Guide**

What is Root

- **Framework for data analysis – mainly for HEP**
- **Root is Free**
- **Portable platform**
  - Runs on Windows, Linux, Mac, other Unix systems
- **ROOT is intended to be next generation software for use in High Energy Physics**
- **Then How about PAW?**

Different from PAW

- **Regular grammar (C++) on command line**
- **Single language (compiled and interpreted)**
- **Object Oriented (use your class in the interpreter)**
- **Advanced Interactive User Interface**
- **Well Documented code. HTML class descriptions for every class.**

CINT

- **C++ INTerpreter**
- **Features**
  - Covers about 95% of ANSI C, 85% of C++
  - CINT is solid enough to interpret itself(80K lines of ANSI C)
  - CINT Scripts can call compiled classes/functions
  - Compiled code can make callbacks to CINT user defined functions
  - Source files and shared objects can be dynamically loaded/unloaded
  - Portability: Linux, Mac, FreeBSD, HP-UX, Sun, Windows, Dos, VMS,...

Object Oriented Concepts

- **Class - Description of a "Thing" Encapsulated Members and Methods**
  - Members - Parts of the "Thing"
  - Methods - How Things work
- **Object - An Instance of Class**
- **Members: a "has a" relationship to the class.**
- **Inheritance: an "is a" relationship to the class.**

The Root Framework

- **Root is a Framework, which**
  - Provide Services and Utilities to the User

So the user doesn't have to know about what below Root. All he need is to tell Root do that.

Root Services and Utilities

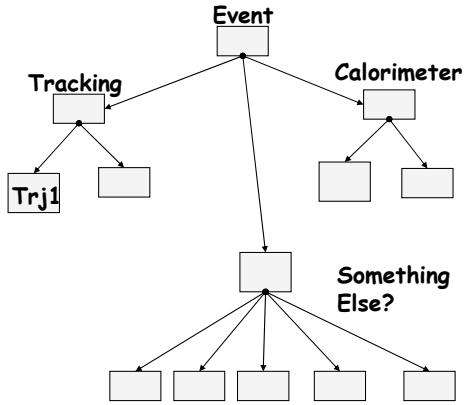
- **2D/3D Graphics**
- **Container Class**
- **Data Storage**
- **Histogram and Fitting**
- **Networking**
- **User Interface**
  - Graphic User Interface
  - Command Line interface: C++ Commands
  - Script Processor

Portable Platform

- **Root is an application with high portability.**
  - Root Files are OS independent
  - Root Wraps the underlying OS
- **It provide a platform to application development. One can write a Root application without having any OS dependence.**

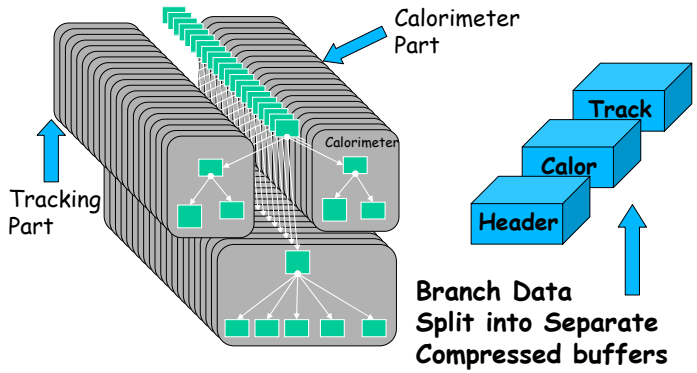
Yushu Yao, University of Alberta, Canada

Root Data Structure, Trees



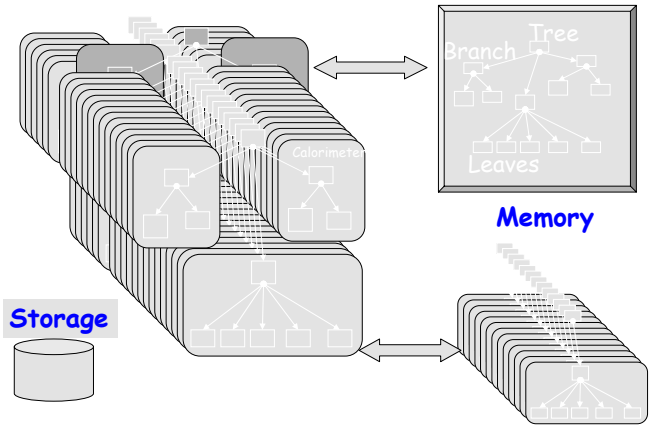
Yushu Yao, University of Alberta, Canada

Storage of A Tree



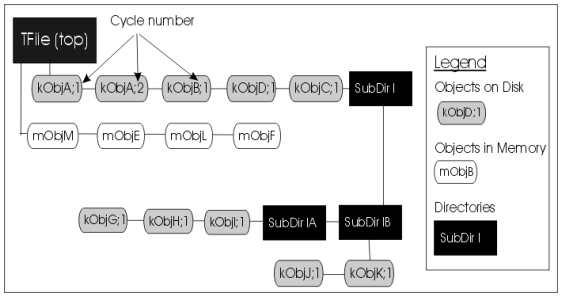
Yushu Yao, University of Alberta, Canada

Storage of A Tree



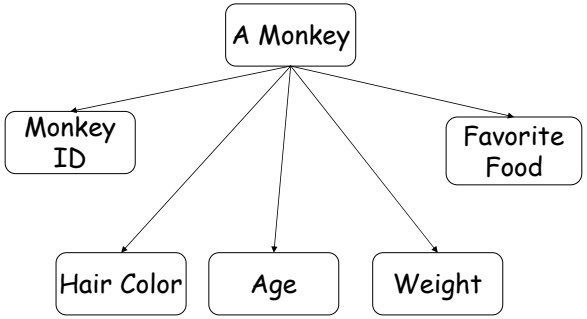
Yushu Yao, University of Alberta, Canada

Root Files



- **A root file is like a UNIX file Directory**
- **System Independent**

A Step by Step Example
The Monkey Survey



Create the Tree

- **Open File**
  TFile f("Monkey.root","recreate");
- **Create Tree**
  TTree t1("MonkeyTree","All About the Monkeys");
- **Data Members**
  Int_t ID, HairColor, Age;
  Float_t Weight;
  Int_t FavoriteFood;
  "I" Means Integer
- **Add Branches**
  t1.Branch("ID",&ID,"ID/I");
  t1.Branch("HairColor",&HairColor,"HairColor/I");
  t1.Branch("Age",&Age,"Age/I");
  t1.Branch("Weight",&Weight,"Weight/F");

Filling A Tree

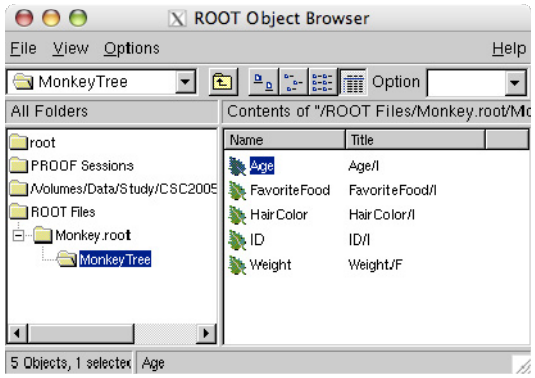
- **Fill Tree**

```
for (Int_t i=0; i<10000; i++) {
  ID          =???;
  Age         =???;
  Weight      =???;
  HairColor   =???;
  FavoriteFood=???;

  t1.Fill();
}
```

- **Save To Disk**
  t1.Write();

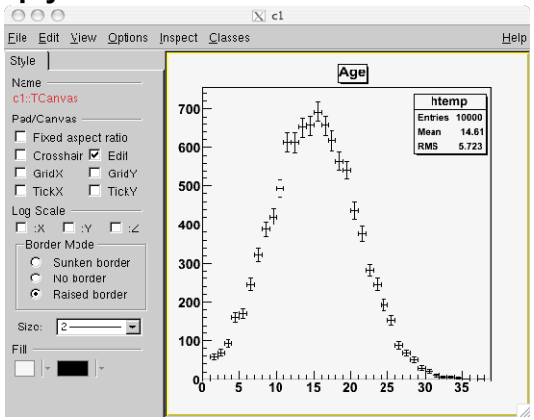
Browsing Trees

- **Open File and Browser**
  Tfile f("Monkey.root"); new Tbrowser;



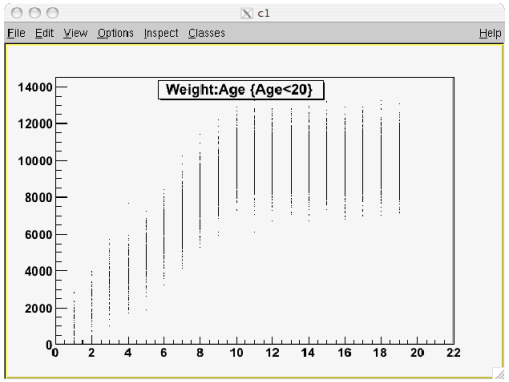
Histograms 1

- **Simply Double-Click on a Leaf**



Histograms 2

- **By Command: Scatter plot of two Fields**
  MonkeyTree->Draw("Weight:Age","Age<20");



Filling Histogram in Script

Weight Distribution

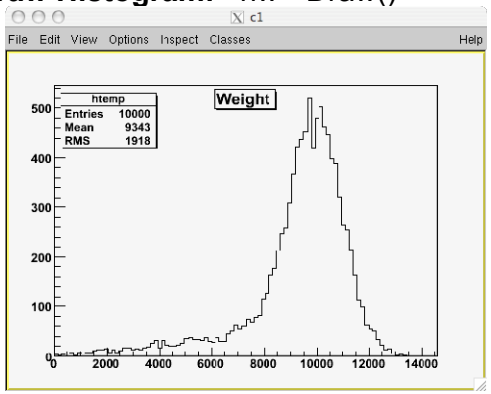
- **Create 1D Histogram**
  TH1F *hw = new TH1F("weightdist","Weight Distribution",100,0.,14000.);
- **Open File**
  TFile f1("Monkey.root");
  TTree *t1=(TTree*)f1.Get("MonkeyTree");
- **Associate Branch with Local Variable**
  Float_t weight;
  t1->SetBranchAddress("Weight",&weight);
- **Loop Over all entries and Fill**
  Int_t nEntries=(Int_t)t1->GetEntries();
  for(Int_t i=0;i<nEntries;i++) {
  t1->GetEntry(i);
  hw->Fill(weight); }

Only Read One Field, Save Time And Memory

Filling Histogram in Script

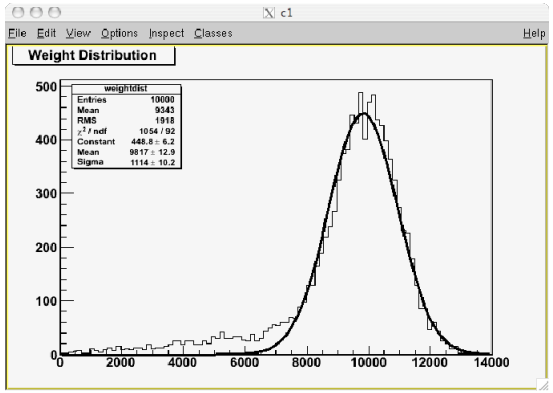
Weight Distribution

- **Draw Histogram:** hw->Draw()



Fitting

- **Fitting Weight with Gaussian**
  hw->Fit("gaus")



End of the Monkey Example

Summary

- **Root and PAW**
- **Root Concepts:**
  - Data Structure
  - Framework
- **Tasting Root**
  - The Monkey Example

References

- ROOT Tutorial FermLab: http://www-root.fnal.gov/root/
- The ROOT Tutorials: http://root.cern.ch/root/Tutorials.html
- The ROOT How To's: http://root.cern.ch/root/Howto.html
- For on-line help for a particular topic it's very useful to use their facility to search the ROOT site. http://root.cern.ch/root

Lots More …

- Math Libraries
- Linear Algebra
- Networking
- Graphics User Interface
- Parallel
- …

Parallel ?

…

Parallel Computing with ROOT and PROOF

Parallel Data Analysis with PROOF

Marek Biskup

Warsaw University

Outline

- **ROOT data structures**
- **Data analysis model of ROOT**
- **Overview of PROOF**
- **PROOF's architecture**
- **Analysis with PROOF**

ROOT

- **Framework for data analysis – mainly for HEP**
- **Many advanced features**
  - Object Streaming
  - C++ interpreter and compiler
  - File format optimized for analysis of HEP events
  - Web scripting (php-like)
- **Portable platform**
  - Runs on Windows, Linux, Mac, other Unix systems
  - Wrappers for O.S. functions
    - Custom GUI widgets
    - Wrappers for threads, semaphores, etc.
- **Free, even for commercial use**

ROOT Data Structures – Trees

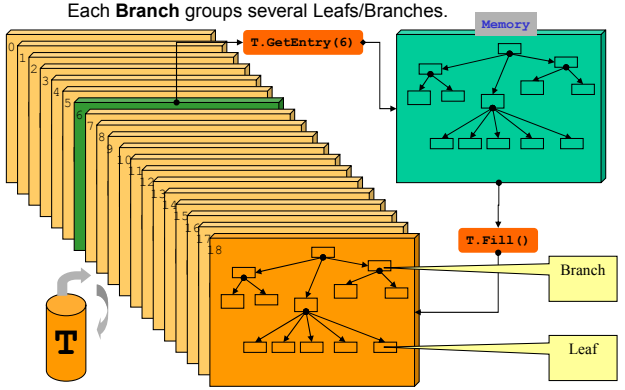
- **Set of records (entries) – often holding events from a detector**
- **Record may contain**
  - basic C types (int, double, arrays)
  - C++ object, polymorphic object,
  - collection, stl collection

```
// tracks
   stl::list<TrackClass>        tracks;
// Electrons
   Int_t                        NoElectrons;
   Double_t                     Momentum[NoElectrons][4];
   Float_t                      Vertex[NoElectrons][4];
// Muons
   Int_t                        NoMuons;
// ...
```

- **Efficient access to partial entry data**
- **Typical size < 2GB**

Trees in memory and in files

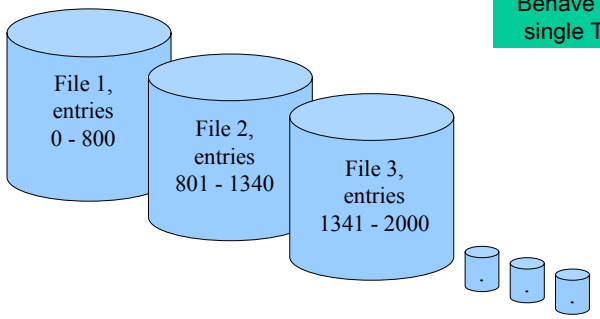
Each **Leaf** is an object (c++ object, array, basic type).
Each **Branch** groups several Leafs/Branches.



ROOT Chains

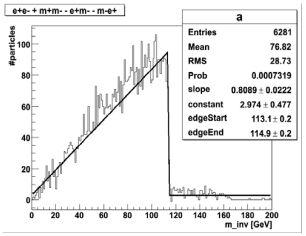
- **A typical Tree: < 2GB (to avoid problems with some operating systems)**
- **Chain – list of trees**
  - e.g. 1000 files – the processing takes long time!

Behave as a single Tree

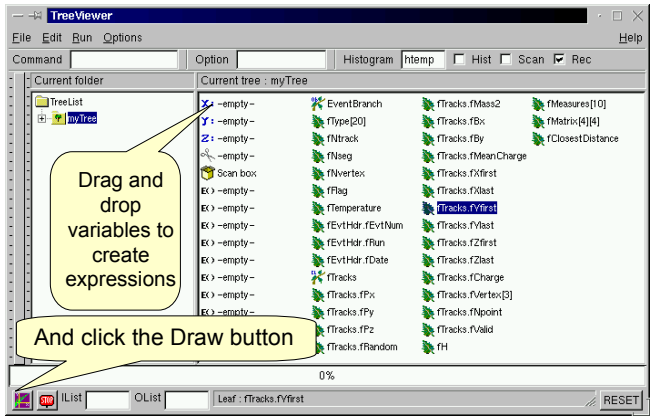


Example Analysis

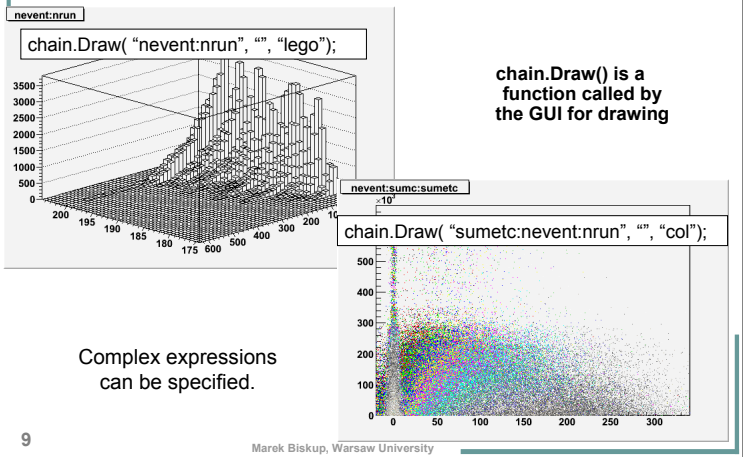
- **Apply a cut on your data**
  - Find all Events with an electron or muon with P_t > 10GeV
  - And a jet with 40 GeV
- **Plot a histogram of the energy of the most energetic jets of each event**
- **A 2D histogram: P_t of electrons missing energy**
- **Histogram of the energy of all electrons with P_t > 10GeV**
- **Fit a curve to your histogram**



Tree Viewer



Chain.Draw()



chain.Draw() is a function called by the GUI for drawing

Complex expressions can be specified.

Selectors

Selectors encapsulate functions important for data analysis. Those functions will be called by the ROOT system:

- **Preprocessing and initialization**
- **Processing each entry (called in a loop over all files and entries in each file)**
- **Post processing and clean-up**

Skeleton can be generated from a Tree file and filled by a programmer

Terminal

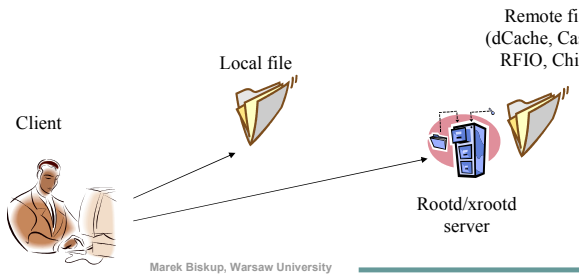
```
void MySelector::Begin(TTree *tree)
{  // function called before starting the event loop
   fPtBranch = tree->GetBranch("Pt");
   fPtBranch->SetAddress(&fPt);
   fMyHist = new TH1("Pt", "Pt");
}
Bool_t MySelector::Process(Long64_t entry)
{   // entry is the entry number in the current Tree
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::Terminate()
{   // function called at the end of the event loop
   fMyHist->Draw();
}
```

Arbitrary order when processing entries

Read only one branch

Root Analysis Model

- Files analyzed on a local computer
- Single thread (no use for multi-core CPUs / hyper threading)
- Remote data accessed via a remote fileserver (rootd/xrootd)



PROOF

Data transfer takes time.

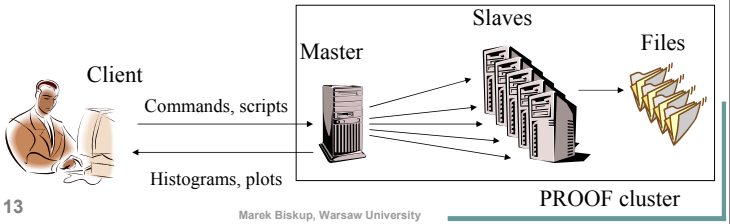
Processing power of a laptop/desktop is limited

Bring the KiloBytes to the PetaBytes and not the PetaBytes to the KiloBytes

- **Parallel analysis of ROOT Data**
- **Using the same ROOT Selectors (transparency)**
- **Execution on clusters of heterogeneous computers (scalability)**

PROOF Architecture

- Queries executed on a cluster of PCs
- The Master divides the work among the Slaves
- After the processing finishes, merges the results (histograms, scatter plots)
- And returns the result to the Client



Installing PROOF

- **ROOT installed on each node**
- **rootd and proofd services (daemons) running on each node**
- **Configuration file on the client**

```
master   node1.cern.ch              image=nfs
worker   dual1.cern.ch   perf=100   image=dual1
worker   dual1.cern.ch   perf=100   image=dual1
worker   fastPC.cern.ch  perf=200   image=nfs
worker   slowPC.cern.ch  perf=30    image=nfs
```

- **Security setup – authentication method**
  - Password, Globus/GSI, Krb5

Using PROOF

- **Normal ROOT analysis:**

```
TChain chain("h42");
{// Define the data set
   chain.Add("root://oplapro62.cern.ch//tmp/dstarmb.root");
   chain.Add("root://oplapro62.cern.ch//tmp/dstarp1a.root");
   chain.Add("root://oplapro62.cern.ch//tmp/dstarp1b.root");
   chain.Add("root://oplapro62.cern.ch//tmp/dstarp2.root");
   // Process the selector
   chain.Process("h1analysis.C");
}
```

Local processing

- **Analysis with PROOF:**

```
{// Open PROOF
   TProof *proof = new TProof("master");
   // Process the selector
   chain.Process("h1analysis.C");
}
```

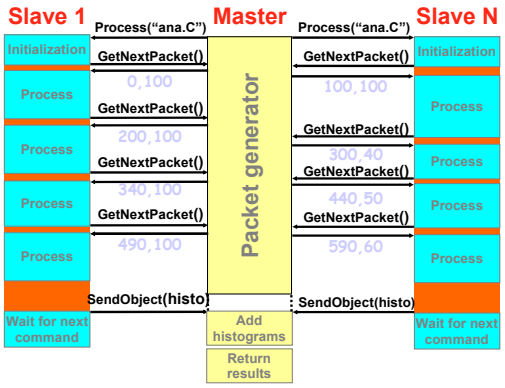
The same chain

Remote processing

PROOF internals

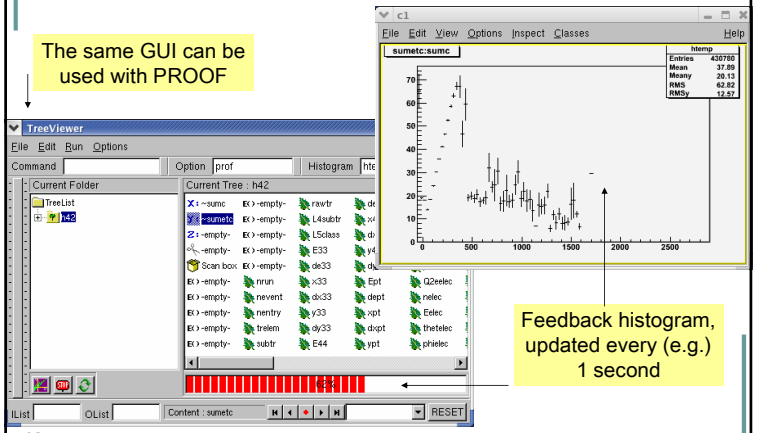
- **Starting Proof:** `TProof *proof = new TProof("master");`
  - The Client connects to the PROOF Master giving a configuration file
  - The PROOF Master:
    - reads names of PROOF Slaves from the configuration file
    - Connects to each Slave
- **Processing a query:** `chain.Process("h1analysis.C");`
  - The Client sends the selector (or selector's name) to the Master
    - Additional libraries and other binaries can be included
      - PAR archives with BUILD.sh and SETUP.C scripts for setup
  - The Master forwards it to the Slaves
  - Slaves start processing

Dynamic load balancing



Real time feedback

The same GUI can be used with PROOF

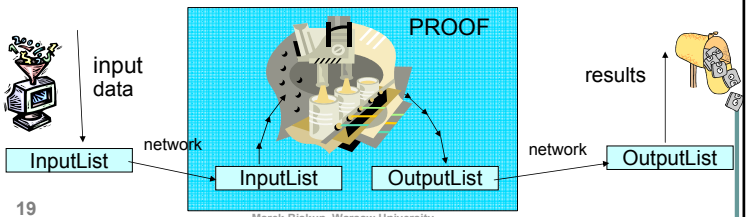


Selectors with PROOF

With parallel analysis things get complicated:

- Many computers to initialize and clean-up
- Many trees in a chain
- Input and output results should be transparently sent over network

```
TSelector::SlaveBegin();
TSelector::SlaveTerminate();
TSelector::Init(TTree*);
TList* fInput, fOutput;
```



Selector – complete example

```
void MySelector::Begin(TTree *tree)
{// called on the client before processing
}
void MySelector::SlaveBegin(TTree *tree)
{// called on each slave before processing
   fMyHist = new TH1F("Pt", "Pt");
   fOutput->Add(fMyHist);
}
void MySelector::Init(TTree* tree)
{// called  each time a tree is changed
   fPtBranch = tree->GetBranch("Pt")
   fPtBranch->SetAddress(&fPt);
}
Bool_t MySelector::Process(Long64_t entry)
{// called on each slave for their entries
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::SlaveTerminate()
{// called on each slave after processing
}
void MySelector::Terminate()
{// called on the client after processing
   fMyHist = (TH1F*)fOutput->FindObject("Pt");
   fMyHist->Draw();
}
```

Client — Initialize each slave

Slaves — Many trees are being processed

Slaves

Slaves — The same code works also without PROOF

Slaves — No user's control on the order

Client

Selectors - summary

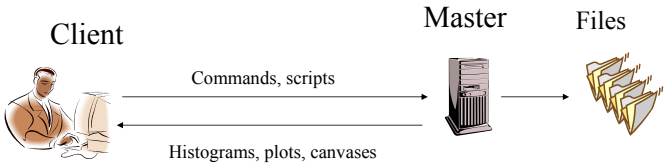
- **Skeletons generated from a tree**
- **Only methods need to be filled**
- **Simplify programs' structure**
- **Can be used for parallel processing as well as for local analysis**

Terminal

```
void MySelector::Begin(TTree *tree)
{// called on the client before processing
}
void MySelector::SlaveBegin(TTree *tree)
{// called on each slave before processing
   fMyHist = new TH1F("Pt", "Pt");
   fOutput->Add(fMyHist);
}
void MySelector::Init(TTree* tree)
{// called  each time a tree is changed
   fPtBranch = tree->GetBranch("Pt")
   fPtBranch->SetAddress(&fPt);
}
Bool_t MySelector::Process(Long64_t entry)
{// called on each slave for their entries
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::SlaveTerminate()
{// called on each slave after processing
}
void MySelector::Terminate()
{// called on the client after processing
   fMyHist = (TH1F*)fOutput >FindObject("Pt");
   fMyHist->Draw();
}
```


PROOF Sequential mode

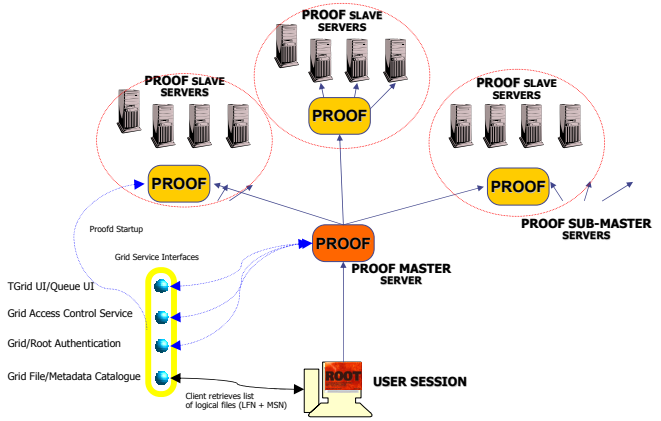
- The Master executes scripts (**Selectors**) and returns results to the Client
- Canvases will be fetched from the Master automatically
- Pseudo-remote desktop (better than XWindow for WAN)



The canvas is automatically displayed after the processing has finished

Executes the selector and creates an off-screen canvas

PROOF on the Grid



Proof as a batch system

- **Several queries (analysis programs) may be submitted to different (or the same) PROOF clusters at the same time**
- **Partial results from the analysis may be viewed**
- **Results are kept on the server and can be retrieved from a different machine**
- **There is a GUI for managing servers and queries**
- **A query can be turned into interactive mode (preview of growing histograms)**

PROOF ideas

- **Local parallel processing**
  - Configure PROOF server on your PC
  - Use efficiently all your processors
- **Office-area cluster**
  - Install PROOF server on your office mate's PC (with low priority)
  - Get your histograms twice faster
- **Dedicated PROOF clusters**
  - Computations performed on the same place where the data is
  - No need to transfer data – just results

Summary

- **ROOT is a framework for HEP data analysis**
- **ROOT data consist of separate events**
- **Histograms are filled by analyzing each event independently**
- **Parallelization of such a task is rather trivial**
- **Making it interactive, transparent, scalable is more difficult**
- **PROOF a part of ROOT dedicated transparent parallel data analysis with ROOT**
- **To be able to use PROOF all custom code must be encapsulated into Selectors**
- **Executing selectors in parallel is transparent for the user**

Further Reading

- **ROOT web site:** http://root.cern.ch
- **Contact: mbiskup@mimuw.edu.pl**

Questions

Software Testing:

Fundamentals and Best Practices

iCSC2006 Software Testing: Fundamentals and Best Practices

Coordinator:

Vijayalakshmi Sundararajan - Newcastle

Software testing plays a vital role in the software development lifecycle. Testing improves the quality of the system and minimizes the resources. In this lecture we will see how to make the Software Testing as an Integral part of the Software System.

The lecture covers topics like testing process and describes the 'V' model and its relevance to testing throughout the project life cycle. Also includes dynamic testing (black/white box) and static testing (reviews etc); describes test management; will also describe the types of CAST tool for automated testing and effective test plans.

A few questions

- *Why is a **V-model** usually preferred over other models (spiral, waterfall, etc)?*
- *Are you aware of the differences between **Walkthrough** and **Inspection** in software testing?*
- *Do you know how to write a **Test plan** based on **IEEE** standards?*
- *Why can't we test our own work - **Psychology** of a **Developer** and of a **Tester**!*
- *How to know when to **stop testing**?*

All the answers in the Software Testing Theme at **iCSC**

Overview

Slot	Lecture	Description	Lecturer
Wednesday 8 March 2006			
09:00 - 09:55	Lecture 1	**Testing Process and Management**	Vijayalakshmi Sundararajan
10:05 - 11:00	Lecture 2	**Testing Techniques and Tools**	Vijayalakshmi Sundararajan
11:30		Adjourn	

LECTURE 1

Testing Process and Management

Wednesday 8 march 2006			
09:00 - 09:55	Lecture 1	**Testing Process and Management**	Vijayalakshmi Sundararajan
		This lecture will cover the testing processes and methods which will help to improve the quality of software. "**Quality Software doesn't happen – it has to be planned**". In the scientific environment, good testing and test planning will result in more accurate and reliable results. This lecture targets a broader audience whoever is involved in software development. It will help you to take your software testing into the next level.	
		Principles of Testing - What is Testing - Why testing is necessary - Cost of Errors - Fundamental Test Process - Prioritization of Testing **Testing in the Software Development Lifecycle** - The V-Model - Economics of testing - High level Test Planning - Acceptance Testing - Functional and Non-Functional System Testing - Integration Testing in the Small **Test Management** - Organization - Configuration Management - Test estimation, Monitoring and Control - Incident Management	

Software Testing: Process and Management

Vijayalakshmi Sundararajan
Test Analyst, UK

Inverted CERN School of Computing, March 8th 2006

Overview

- **Principles of Testing**
- Testing Throughout the Lifecycle
- Test Management

What is Testing ?

is the **process of executing a program with the intent of finding errors**. [Glen Myers]

- Any activity aimed at **evaluating the capability of a program or system** and determining whether **it meets the specifications and gives the desired results** [Hetzel]
- It is **not** the objective of Software Testing to **PROVE** that the software works.

How Failures Occur ?

ERROR ⇒ **FAULT** ⇒ **FAILURE**

- An error is a **human action** that produces an **incorrect result**.
- A fault is a **manifestation of an error** in software.
- A **fault**, **when encountered** is called Failure.

What causes Bugs?

- **Imperfection**
- **Poorly defined requirements and specifications**
- **Undocumented changes**
- **Incorrect assumptions**
- **Miscommunication**

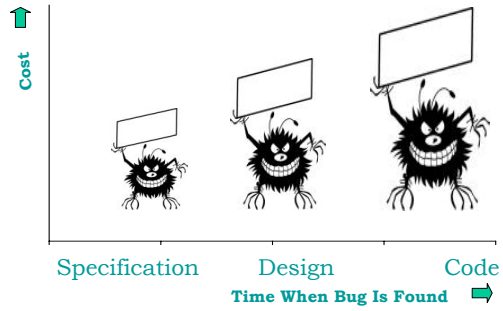
What causes Bugs?

- **Poor training**
- **Lack of testing**

- **Mistakes made under pressure**
- **Poorly documented code**

How much does it cost ?

The cost to fix bugs **increases dramatically over time**



Cost of Errors - Examples

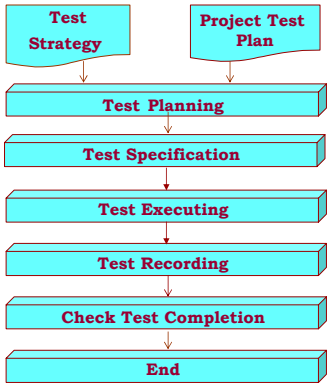
- **A single failure can cost very little or nothing**
  The wrong font in a letter may just look strange or may not be noticed at all and may only cost a few pennies to correct.
- **A single failure can cost lives**
  US Patriot Missile Defense System, 1991 -A Software bug was found that caused a small timing error in the Patriot's tracking system. 28 US soldiers were killed when the tracking system (which had been accumulating errors for 100 hours) caused a Patriot to miss an incoming missile.

Cost of Errors - Examples

A single failure can cost millions

- NASA Mars Polar Lander, Lost 3rd December 1999
- The NASA Mars Polar Lander used a parachute to slow its descent to Mars. After deploying the parachute, three legs would open in preparation of touchdown. At 1800 feet above the surface, it was to release the parachute and ignite landing thrusters. A switch was incorporated in one of the feet of the craft to switch off the fuel ton touchdown by setting a data bit in a computer. It was found that on opening the legs, vibration caused the switch to operate, turning off the fuel and causing the craft to plummet 1800 feet to the surface.
- **The Lander was tested by multiple teams** – one for the leg opening process and one for the rest of the landing process. The first team did not check to see if the landing bit was set (not their area) and the second team always cleared the bit before testing.

Test process stages



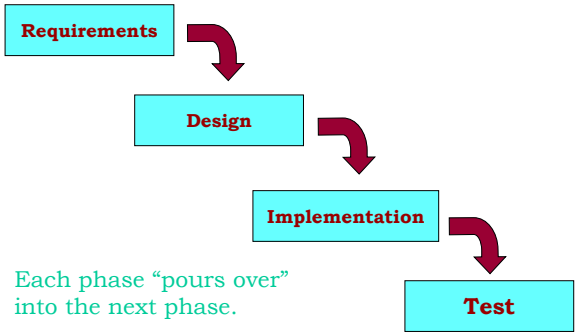
How much Testing is Enough ?

- Complete testing is neither theoretically nor practically possible – Exhaustive Testing
- How much testing would you be willing to perform if the risk of failure were negligible?
  or
  if a single defect could cost you your life's savings, or,
  even more significantly, your life? [Hetzel 88].
- Depends on the **business criticality**.
- Severity, Visibility , Complexity, Resources, Technical Criticality and Probability of Failure

Overview

- Principles of Testing
- **Testing Throughout the Lifecycle**
- Test Management

SDLC - Water Fall model

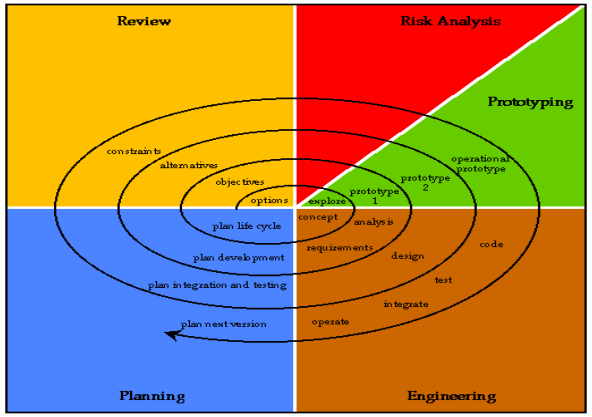


Each phase "pours over" into the next phase.

Limitations of Waterfall model

- Development **assumes** that **requirements** are set, **stable**, and fully evolved before analysis begins.
- The **problems** with **one phase** are **never solved** completely during that phase.

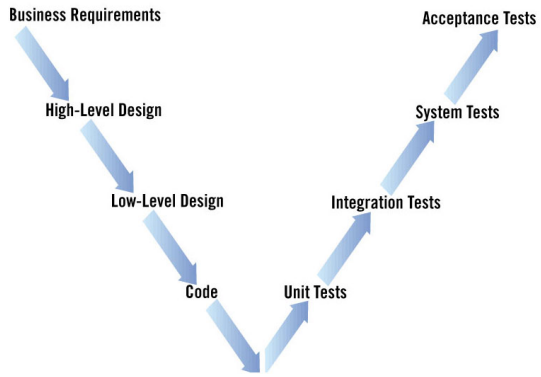
SDLC - Spiral model



Limitations of Spiral model

- Can be a **costly** model.
- Risk analysis **requires** highly specific **expertise**.
- Doesn't work well for **smaller projects**.

SDLC - V model



Advantages of "V Model"

- The V Model, **gives equal weight to testing** rather than treating it as an afterthought
- V Model is the process model to provide quality product by **combining SDLC and STLC**
- STLC – Software Testing Lifecycle
- SDLC – Software Development Lifecycle

Test Plan

- IEEE 829 : Purpose of master test plan is to prescribe scope, approach, resources and schedule of testing activities.

Test Plan Identifier	Approach	Responsibilities
Introduction	Item pass/fail criteria	Staffing/training needs
Test Items	Suspension/resumption	Schedule
Featured to be tested	Testing deliverables	Risks/contingencies
Features not to be tested	Testing Tasks	Approvals
	Environmental needs	

Acceptance Testing

- Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component. [IEEE]
- **Types**
  - **User Acceptance Testing**
  - **Contract Acceptance Testing**
  - **Operations Acceptance Testing**
  - **Alpha Testing**
    in-house site not involved with the software developers
  - **Beta Testing**

Integration Testing in Large

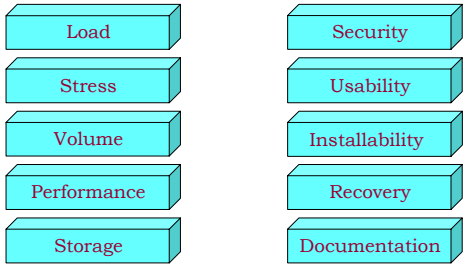
Integration Testing

Is performed to expose faults in the interfaces and in the interaction between integrated components.

- **Incremental**
  - Top-Down
  - Bottom-Up
- **Non-Incremental**
  - Big-Bang

System Testing

- **Functional Testing**
- **Non-Functional Testing**



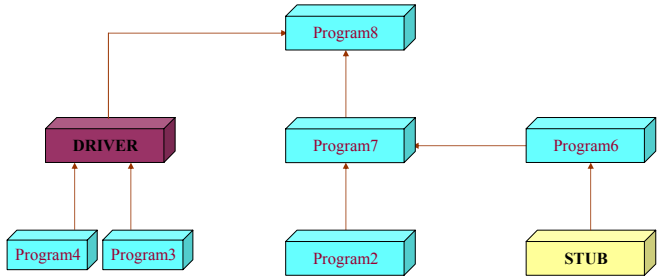
Integration Testing in Small

- Components are assembled into sub-systems and sub-systems are linked together to form complete systems.
- **Stubs**

  A stub is a skeletal or special purpose implementation of a software module, used to develop or test a component that calls or is otherwise dependent on it.
- **Drivers**

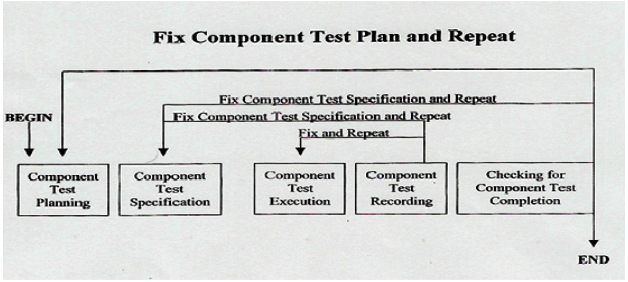
  Supporting code and data providing an environment for testing part of a system in isolation

Drivers & Stubs - Illustration



Component Testing

- **Testing of individual components.**
- **A component is**
  minimal software item which has separate specification.



Overview

- Principles of Testing
- Testing Throughout the Lifecycle
- **Test Management**

Testing Independence

- If **Test cases**
  - are designed by the software developer (Low)
  - are designed by another member in the team.
  - are designed by someone from a different team.
  - are designed by a different organization
  - **Test Cases are not chosen by a person (HIGH)**

Can't we test our own work ?

- Same **Assumptions** are carried
- We see what we want to see
- **Emotional** Binding
- We concentrate on **proving**
- **Familiarity**

Different Testing Structures

- **Self** Testing
- **Buddy** System
- Test Team
- Independent Test Consultant
- Different Organization

Configuration Management

Configuration Management :

- Method to prevent any **unauthorized changes** to any asset of the system.
- Provides version control, traceability and enables reconstruction.

- **Testing**, not just development must also **be under effective Configuration Management** - Test Configuration Items

Test Estimation

Test Estimation Output :

- Overall **time** required for Testing – write and execute Test cases
- **Resources** needed to run Tests – Environment, skills
- Allowance for **rework** – Recording, Investing and Retesting faults.

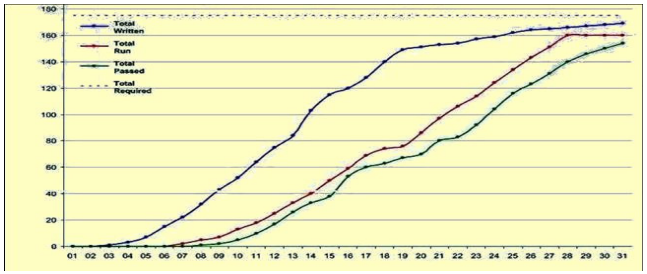
Test Estimation

Factors to Consider When Estimating:

- Risks
- Complexity of Code and hardware
- Stability of Code
- Coverage Required

Test Monitoring

- **Test efforts can fail despite wonderful plans**
- **Use Metrics**
- **Track the quantity and also the time taken.**



Test Control

- The **better** the testing is **monitored** the **easier** it is for the management to exercise **control** over the testing process.
  - Resources
    Assign more, Relocate and Train
  - Schedules
    Postpone, Bring Forward
  - Environment
    Reconfigure, Add new
  - Completion Criteria
    May change if testing falls behind

Incident Management

What is an Incident ?

Any condition that departs from the expectations -

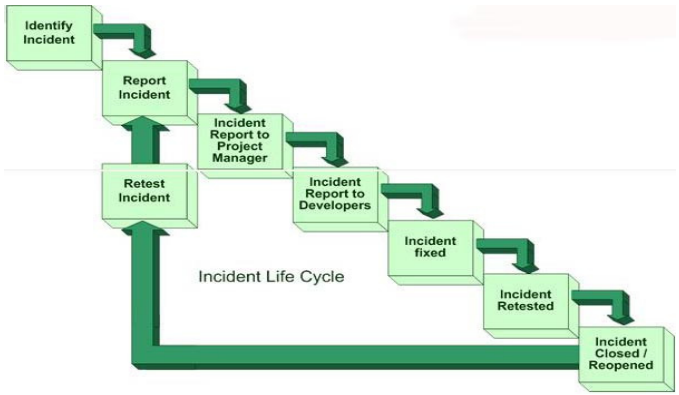
Purpose of Incident Tracking :

To identify and track potential defects

To Ensure defects are addressed in a timely manner

To discover areas of improvement

Incident Life Cycle



Testing Standards

Three Types of standards:

- **QA standards**
  - Testing SHOULD be performed
  - IEEE 1012:1998 – Software Verification and Validation
- **Industry standards**
  - Testing at WHAT LEVEL?
  - Railway signaling Standard
- **Testing standards**
  - HOW TO perform Testing?
  - BS 7925-1, BS 7925-2 and IEEE 829:1998

Thank You ☺

LECTURE 2

Testing Techniques and Tools

Wednesday 8 march 2006			
10:05 - 11:00	Lecture 2	**Testing Techniques and Tools**	Vijayalakshmi Sundararajan
		The aim of this lecture is to describe various software testing techniques. It will describe structural testing techniques as well as error guessing. This lecture is also intended to give you a flavor of the many different types of computer aided testing tools available. This lecture targets an audience with a software background and experience.	
		Dynamic Testing Techniques - Black Box Test Techniques - White Box Test Techniques - Error guessing **Static Testing** - Reviews and Types of reviews - Fagan Inspection method - Static Analysis **Tool Support for Testing** - Types of CAST tool : Requirements testing; Static analysis; Test design; Data preparation; Character-based test running; GUI test running; Test harnesses, drivers and simulators; Performance testing; Dynamic analysis; Debugging; Comparison; Test management; Coverage measurement. - Tool selection and implementation	

Software Testing: Techniques and Tools

Viji Sundararajan
Test Analyst, UK

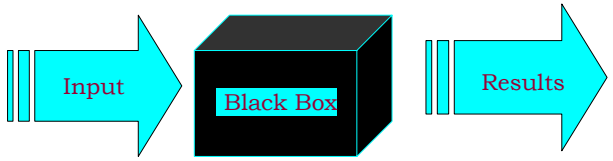
Inverted CERN School of Computing, 6-8 March 2006

Overview

- **Dynamic Testing**
- Static Testing
- Tool support

Black Box Testing (Functional Testing)

- Test case selection is based on the **analysis of the specifications** of the component **without referring** to internal working details.

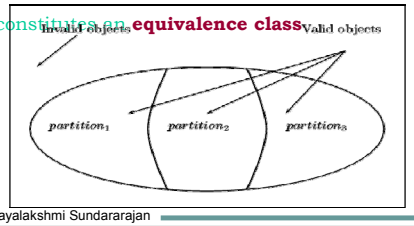


- Black box testing is relevant throughout the lifecycle.

Black Box Techniques Contd.

- **Equivalent Partitioning** - What is it ?
- Why -> **Good Coverage** with less number of Test cases.
- **Example :**
  - The specifications for a DBMS state that the product must handle any number of records between **1 and 16,383**
  - Range (1..16,383) constitutes an **equivalence class**

p1: < 1 record
p2: <> 1 and 16,383
p3: > 16,383 records



Black Box Techniques Contd.

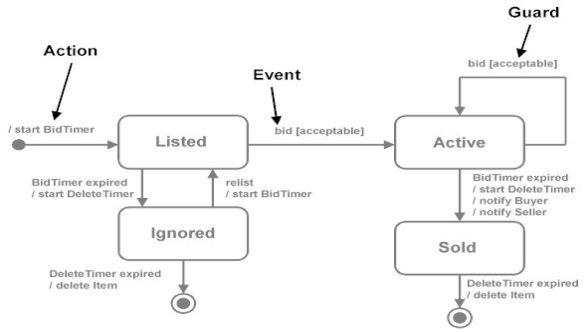
- **Boundary Value Analysis** - The aim of boundary value analysis is to **examine the boundaries** of equivalence classes.
- Boundary Value Analysis is both a **refinement** of equivalence partitioning and an **extension** of equivalence partitioning.

Test case 1: 0 records
Test case 2: 1 record
Test case 3: 2 records
Test case 4: 723 records
Test case 5: 16,382 records
Test case 6: 16,383 records
Test case 7: 16,384 records

Black Box Techniques Contd.

State Transition Testing

Test cases are designed to execute state transitions.



Black Box Techniques Contd.

Cause and Effect Graphs

- Examine the requirements.
- **Restate** them as logical relation between inputs and outputs.
- The result is a Boolean graph representing the relationships called a **cause-effect graph.**
- **Convert** the graph to a decision table.
- Each column of the **decision table** corresponds to a test case for functional testing.

Syntax Testing

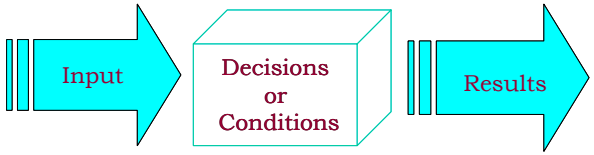
Choose Test Cases that violates format rules for the input.

Random Testing

White Box Testing

Logic Driven

- Test case selection that is based on an **analysis** of the specifications of the **internal structure** of the component.



- White Box testing is less useful towards system and acceptance testing.

White Box Techniques

Statement testing

Test cases are designed to execute statements.

- Consider the following extract of a code:
  1. **A= Prompt ('Enter a number')**
  2. **If A> 1000 then**
  3. **Print Message 'Large Number'**
  4. **End If**

$$Statement\ \ Coverage = \frac{Lines\ of\ code\ executed}{Total\ lines\ of\ code}$$

- Test case where line 2 equates to TRUE will give100% statement coverage.

White Box Techniques Contd.

- **Branch Decision Testing**
- **Branch Condition Testing**
- **Branch Condition Combination Testing**
- **Modified Condition Decision Testing**
- **Linear Code Sequence And Jump Testing**
- **Data Flow Testing**

White Box Techniques Contd.

Error Guessing :

- **Experience of the tester** is used to postulate what faults exist, and To design tests - BS7925-1
- Although not a systematic approach
  - Should be planned
  - Should be documented
- Based on knowledge of the product, experience and intuition.
- This complements other techniques does not replace.

Overview

- Dynamic Testing
- **Static Testing**
- Tool support

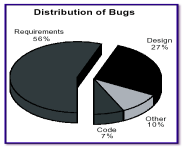
Reviews

- Testing an **object without execution** on a computer is called **Static Testing**
- **Reviews are** part of **Static Testing**
- Review is evaluation of technical matter by a group of people working together
- Reviews are the **only option** in the early stages of testing

Why Review ?

Early discovery of errors.

Majority of bugs found are due to unclear requirements.



What to Review?

Anything can be reviewed

- **Specifications**
- **Code and Program Designs**
- **Testing Documentation**
  - It is estimated that only 25% of Test cases are written correctly in first draft.
- **Cost Of Reviews:**
  - Reviews cost 15% of the total development budget.
  - 70% of the project issues can be uncovered during Review process.

Types of Reviews

Informal Review
- Buddy Review/Sanity Check involves one or two reviewers and no formal documentation required.

Walk-through
- Educating the users regarding the software product.
- Involves walkthrough leader, author and recorder.
- Outcomes are formally recorded.

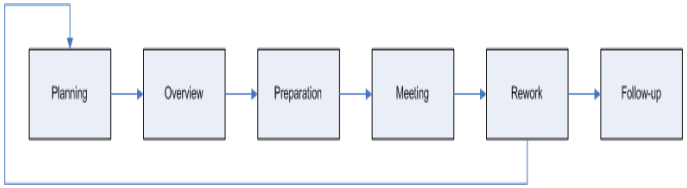
Technical reviews
- To evaluate a software product by a team of qualified personnel to determine its suitability for intended use.
- Involves Review Leader, Decision maker, Technical Staff and Recorder

Types of Reviews Contd.

Inspection

Is to detect and identify software **product anomalies**.

Inspection method defined here is known **Fagan Inspection**.



Fagan Inspection Roles

- **Moderator**
- **Author**

Fagan Inspection Roles

- **Reader**

- **Inspector**
- **Recorder**

Overview of Reviews

Type	Approach	Led By	Documented	Usual Activities / By Whom
Informal Review	Informal	No formal leader	No	Look at product Provide comments on it
Walkthrough	Less Formal	Author	Yes	Product presented Peers find errors
Technical Review	Formal	Presenter	Yes	Find faults in code Peers and technical experts
Inspection	Most Formal	Moderator (not Author)	Yes	Product reviewed sequentially Peers find errors

Review Pit falls

- All members must be trained - **particularly moderator**
- There must be **sufficient document** to review
- **Management** must fully **support** review process
- Reviewers must have sufficient expertise
- Team should be of reasonable size

Static Analysis

- **What is it ?**
- **What can Static Analysis find ?**
  - Unreachable code
  - Undeclared variables
  - Parameter Type mismatches
  - Uncalled Functions
  - Infinite loops
  - Possible array bound violations

Static Analysis Pitfalls

- Not possible to check declared variables values
- It can identify errors which would not necessarily cause failures
- Code is not executed

Types of Static Analysis

Compiler Generated information

Data Flow Analysis

Control Flow Graphing

Complexity Metrics

Provide a way of quantifying the likelihood of errors and the basic amount of testing required

Examples

Mccabe's Cyclomatic Complexity
Lines of Code

McCabe's complexity metric

- The metric can be calculated in **three** different ways.
- **M=L-N+2P** , where: **L** = links in graph, **N** = nodes in the graph, **P** = disconnected parts of the graph.

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
> 50	very high risk

Weakness of the McCabe metric

- McCabe metric **assumes that faults are proportional to decision complexity**
- Equally it **does not distinguish between different kinds of decisions**
- **CASE statements** are treated the **same as nested IF statements** which is again counter intuitive

Overview

- Dynamic Testing
- Static Testing
- **Tool support**

What are CAST Tools ?

- **Computer Aided Software Testing** Tools are pieces of Software automate and support the test process.
- **Time Saving**
  As long the tools are setup and run correctly.
- **Impossible Tasks**
  Can be performed by CAST. such as examine locations in computer memory.

- **Reduce Boredom**
- **Available for the entire life cycle.**

Types of CAST tools

Requirements testing tools

Requirements Analysis Tool
- Example is Rational Requisite Pro

Requirement Management Tools, etc...

Static analysis tools

Provide information about the quality of the software by **examining the code**, rather than running test cases.

Types of CAST tools (Cont.)

Dynamic analysis tools

Monitor allocation, use and de-allocation of **memory**, flag
memory leaks, unassigned **pointers**, pointer arithmetic and other errors difficult to find ' *Statically* '

Test design tools

Generate test cases from a specification that must normally be held in a CASE tool repository. An example is
McCabe Test

Types of CAST tools (Cont.)

Character-based test running tools

Capture and replay facilities for dumb-terminal. Often used in regression testing.

Example of CAPBAK User Interface

Types of CAST tools (Cont.)

GUI test running tools

Capture and Replay for GUI interface. To automate regression testing. **Win Runner** is an example

Test harnesses and drivers

To execute software under test which may not have a user interface. Mostly **custom-written**

Performance test tools

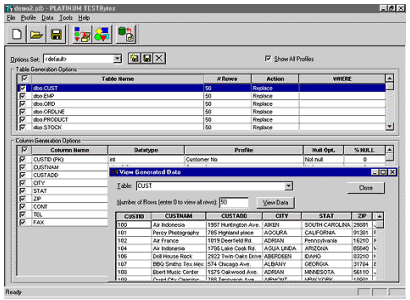
- Load generation and Transaction measurement.
- Recently **Microsoft was sued** as their **Xbox product** burnt a house while using because of the charger heats up to such an extent.
  http://www.theinquirer.net/?article=27873

Types of CAST tools (Cont.)

Test data preparation tools

Data created, generated, manipulated and edited for use in tests.

TestBytes is an example.



Types of CAST tools (Cont.)

Debugging tools

mainly used by programmers to reproduce bugs and investigate state of programs. **Visual Basic Editor** has an inbuilt debugger.

Comparison tools

- Standalone comparison tools
- Build-in Comparison tools

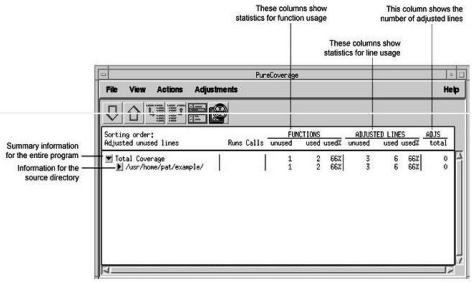
Test management tools

Manages and Stores test assets - test plans, scripts, specifications, results and so on. **Test Director** is an example.

Types of CAST tools (Cont.)

Coverage measurement (or analysis) tools

- Provide structural test coverage when tests are executed.
- Report of Rational **Pure Coverage** for the **"Hello world"** program.

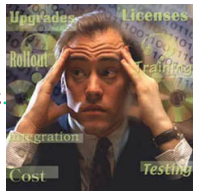


To Automate or Not?

- Not possible to automate everything
- Multiple tools may be needed
- Test effort does not decrease immediately after using tools
- Test schedules do not shorten immediately
- Use of tools may require new skills
- Tools will not provide 100% coverage

Tool Selection

- Once automation requirements are agreed, selection process has **four** stages:
  - Creation of a candidate tool **shortlist**.
  - Arrange **demos**.
  - **Evaluation** of selected tool.
  - **Review** and select tool.



- **Pilot Project**
- **Roll out**

Thank You ☺