Computing in High Energy Physics
An Introductory Overview

Frank Gaede
DESY IT Physics Computing
Summer Student Lecture
DESY, August 18, 2009
Introduction

• The aim of this lecture is to provide an overview and some understanding of the basic concepts of Computing in High Energy Physics.

• The (randomly :) ) chosen topics are of course a subset of possible topics.

• We will just scratch the surface of this wide field.

• For an overview of topics that are currently discussed and under development see the programme of the CHEP 2006 Conference in Mumbai (next two slides):
Thank you for your attendance. We hope to see you all again for CHEP'07 at Victoria BC Canada.

To view some of the photographs taken during CHEP'06, please click here.

Tata Institute of Fundamental Research
Welcomes you to the international conference on
Computing in High Energy and Nuclear Physics
13-17 February 2006, T.I.F.R. Mumbai, India

CHEP conferences provide an international forum to exchange information on computing experience and needs for the High Energy Physics and Nuclear Physics communities, and to review recent, ongoing and future activities. CHEP conferences are held every 18 months.
CHEP 06 Programme I

- Online Computing
  - CPU farms for high-level triggering; Farm configuration and run control; Describing and managing configuration data and conditions databases; Online software frameworks and tools

- Event processing applications
  - Event simulation and reconstruction; Physics analysis; Event visualisation and data presentation; Toolkits for simulation and analysis; Event data models; Detector geometry models; Specialised algorithms for event processing

- Software Components and Libraries
  - Persistency; Interactivity; Foundation and utility libraries; Mathematical libraries; Component models; Object dictionaries; Scripting; Graphics; Use of 3rd party software components (open source and commercial)

- Software Tools and Information Systems
  - Programming techniques and tools; Software testing; Configuration management; Software build, release and distribution tools; Quality assurance; Documentation
Computing Facilities and Networking

- Global network status and outlook; Advanced technologies and their use in applications; HENP networks and their relation to future grid systems; The digital divide and issues of access, readiness and cost; Collaborative systems, progress in technologies and applications

Grid middleware and e-Infrastructure operation

- Integral systems (cpu/storage) and their operation and management; Functionality and operation of regional centres; Global usage and management of resources; Grid infrastructure and its exploitation in distributed computing models.

Distributed Event production and processing

- Development of the distributed computing models of experiments; Real experience in prototypes and production systems; Emphasis on the early days of LHC running.

Distributed Data Analysis

- Large distributed data-base over wide area network; Low-latency
Selected Topics

- **Online Computing - DAQ (data acquisition)**
  - Readout software
  - Monitoring
  - Trigger

- **Offline Computing**
  - Monte Carlo Simulation
  - Reconstruction

- **Computing infrastructure (hardware)**
  - large PC farms

- **GRID Computing**
HEP Computing overview

Monte Carlo Production

Event Generation

Detector Simulation

Digitization

Online Data Taking

Trigger & DAQ

Data Processing

Event Reconstruction

Data Analysis

Publication

CON GEO

A. Gellrich
Online - DAQ

- The Online/DAQ computing makes sure that the interesting physics data is read out from the detector and written to tape/disk (mass storage)

- it is typically divided in three main tasks:
  - **Online Monitoring (slow control)**
    - temperature readings, high voltage, gas supplies...
    - manage the running of the detector
  - **Trigger (software/hardware)**
    - give signal that data needs to be read out 'coz sth. interesting happened in the detector
  - **readout (data flow)**
    - actual readout is tightly coupled to hardware (front end electronics)
Online detector/run control

Modern particle physics detectors are run using online software tools, example: BaBar ODC

- Online Monitoring – Slow Control systems typically provide a GUI that allows the physicist to run and monitor the detector, by;
  - configuring the detector / online software / trigger
  - start & stop data taking runs
  - monitor temperature readings, high voltage, gas supplies...
Readout software

Specific features (the FEE model)...

- the readout software is very tightly coupled to the hardware. i.e. front end electronics and readout boards it typically involves tasks as:
  - buffering of data read out from the detector
  - feature extraction (zero suppression, integrating electronics signals, fitting of peak positions,...)

example: front end readout software (Data Flow) of the BaBar experiment
Multilevel Trigger System I

- **Trigger**
- typically collider
- experiments have far more
- activity in sensitive parts
- than can be read out,
- stored or analyzed
- due to:
  - background from beam-gas interactions
  - high cross sections of (soft) relatively uninteresting physics, e.g. photoproduction
- multilevel trigger system reduce the rate through
  - successive application of more advanced algorithms
- buffering pipelines help to reduce the dead time
Multilevel Trigger System II

Other examples:
HERA-B and ATLAS trigger systems

Exception: planned ILC (Linear Collider) due to comparatively low rates and high extrapolated bandwidth (~2015?) no trigger system foreseen but continuous read out planned -> no dead time!
Atlas trigger and physics rates
Trigger rates and event sizes
Monte Carlo Simulation
Why Monte Carlo Simulations?

- **R&D phase:** (planning phase, e.g. ILC)
  - determine the best geometry of the detector
  - study the (needed) performance of subdetectors
  - compare different designs (competition)
  - evaluate feasibility (bg-rates/radiation)

- **Data taking (running experiments)**
  - study **efficiency** of the detector for all the different physics channels (cross sections)
  - determine the **fundamental parameters** of underlying physics
Monte Carlo Simulation Programs

Generator: generates 4-vectors of all particles in the event e.g. PYTHIA, Herwig

Simulation: detector response to longlived particles -> HITS e.g. Geant4

"Physics"

"Measurement"
Simulating the detector response

- example: *geant4* – a C++ toolkit that simulates passage of particles through matter using “known physics”:
  - particle decay (lifetime/branching ratios)
  - photoelectric effect
  - Compton scattering
  - pair creation (EM-cascade)
  - energy loss due to ionization (exaltation), multiple scattering
  - Cherenkov radiation
  - Positron – Electron Annihilation
  - Bremsstrahlung
  - ~hadronic interactions
  - cross section tables
  - parameterizations
  - ... many more ...

*Geant4*
A toolkit to simulate the interaction of particles with matter

**Concept**
- General structure: the passage of particles through matter
- Provides a complete set of tools for all domains of radiation transport:
  - Detector response
  - Physics processes and models
  - Tracking and scoring
  - Graphics and user interfaces
- Proven in practice

**Geant physics processes**
- Provide electromagnetic and hadronic physics processes

**Applications**
- High energy and nuclear physics detectors
- EPICS, DATAMAT, DREEDS and DELPHI at CERN and Hall A at SLAC
- Accelerator and shielding
- Linear accelerators and ring accelerators
- Medical
  - Radiography
  - Radiological physics
  - Brachytherapy
- Medicine
  - Cancer treatments
  - Radiation therapy
  - Radiation effects
  - Radiation dose estimation
  - Radiation therapy
- Space environments
  - spacecraft design
  - cosmic ray studies
- Cross sections
  - QED
  - QCD

**Advantages**
- Simulates the passage of complex energy efficiently
- Provides comprehensive physics processes for application areas
- Enables users to tailor simulation components and address accuracy needs
- Robust and adaptable
- Easy to integrated into graph applications

*The European Organization for Nuclear Research (CERN) is the world’s largest and most powerful particle physics laboratory. Its mission is to provide a unique environment for particle physicists to explore the fundamental nature of matter, energy and space-time. The CERN Accelerating Science and Technology (CERN)ⵍ להביןות, a strategic part of CERN’s principal mission of fundamental research.*
passage of particles through matter

- simulating the detector response only meaningful if the underlying physics is known well enough

- in general true for all electromagnetic interactions
  - ionization in tracking detectors
  - electromagnetic showers in calorimeters
    - EM-cascade due to repeating Bremsstrahlung/pair-creation
    - QED has a non divergent perturbation series

- in general not so true for hadronic interactions
  - QCD has divergent perturbation series in
    - low energy (soft) hadron interactions
Ionization energy loss

\[ -\frac{dE}{dx} = \kappa z^2 \cdot \frac{Z}{A} \cdot \frac{1}{\beta^2} \left[ \frac{1}{2} \ln \frac{2m_e c^2 \gamma^2 \beta^2}{I^2} E_{\text{kin}}^\text{max} - \beta^2 - \frac{\delta}{2} \right] \]

Bethe-Bloch Formula

Mu on Fe

dE/dx in a TPC
hadronic shower parameterization

Models

- electromagnetic:
  • standard:
  • low energy
  • Penelope:
    • hadronic:
      • Elastic: generic elastic, medium and high energy elastic, coherent elastic pp, pn, coh
      • Precompound
      • Leading particle bias
    • Cascade: Bertini cascade, Binary cascade, Binary light ion
  • Low energy parameterized: pi+ inelastic, pi- inelastic, K+ inelastic, K- inelastic, K0 inelastic, K0S inelastic, proton inelastic, neutron inelastic, lambda inelastic, sigma+ inelastic, sigma- inelastic, xi- inelastic, xi0 inelastic, omega- inelastic, anti-proton inelastic, anti-neutron inelastic, anti-lambda inelastic, anti-sigma+ inelastic, anti-sigma- inelastic, anti-xi- inelastic, anti-xi0 inelastic, anti-omega- inelastic, deuteron inelastic, triton inelastic, alpha inelastic
  • High energy parameterized: pi+ inelastic, pi- inelastic, K+ inelastic, K- inelastic, K0 inelastic, K0S inelastic, proton inelastic, neutron inelastic, lambda inelastic, sigma+ inelastic, sigma- inelastic, xi- inelastic, xi0 inelastic, omega- inelastic, anti-proton inelastic, anti-neutron inelastic, anti-lambda inelastic, anti-sigma+ inelastic, anti-sigma- inelastic, anti-xi- inelastic, anti-xi0 inelastic, anti-omega- inelastic
  • High energy: Fritiof-CHIPS (FTF), Fritiof-precompound (FTFP), Quark-gluon string CHIPS (QGSC), Quark-gluon string precompound (QGSP)
  • Nucleus-nucleus: electromagnetic dissociation, abrasion/ablation, Binary light ion
  • Gamma- and Lepto-Nuclear: electro-nuclear, gamma-nuclear, muon-nuclear
  • Neutrons: generic capture, generic fission, high precision elastic, high precision inelastic, high precision capture, high precision fission, high precision parameterized elastic, high precision parameterized inelastic, high precision parameterized capture, high precision parameterized fission

Figure 1: Current GEANT4 LEP physics list setting against data (Ouyang, Peterson 1992)

Figure 2: Bertini cascade model physics list setting against data (Ouyang, Peterson 1992)
hadronic showers in ILC-HCal prototype

GEANT4.6.1

- P. Melchior, summer student programme 2004
- need verification with testbeam
  - still ongoing!

=> only two classes of physics lists in given energy domain:
  - LEP like parameterization
  - Bertini cascade
physics processes in geant4

- each particle has its own list of applicable processes
- at each step, all processes listed are invoked to get random physical interaction lengths (Monte Carlo Method!)
- the process with the shortest interaction length limits the step
- each process can have any of the following actions:
  - AtRest (e.g. muon decay at rest)
  - AlongStep - continuous process (e.g. ionization)
  - PostStep - discrete process (e.g. decay in flight)
- every action that is applied to a particle is defined as a process:
  - transportation (E,B fields)
  - decay
  - interactions with Material (ionization, delta-electrons,......)
  - step length cut off
dominating processes in simulation

and in a real detector...

- ionization
- multiple scattering
- B-Field

- electromagnetic cascade (shower)

• hadron shower parameterization
simulation output: hits

- **simulation** output:
  - calorimeter hits
    - cell position
    - amplitude (energy)
  - tracker hits
    - amplitude
    - dE/dx
- **digitization**
  - smear hits
  - apply noise
    - electronics
    - physics
Reconstruction

- now we have simulated the detector response (hits) to the generated event
- ideally this is indistinguishable from real data
  - (not true in practice as of course MC-Truth is conserved for cross checks !)
- next step: Reconstruction – combining hits to reconstructed particles in order to perform the actual physics analysis
reconstruction - tracking

- **tracking** (pattern recognition):
  - **track finding**
    - combine hits that most likely belong to one particle
  - **track fitting**
    - apply fit to all hits taking B-field into account
    - 'Helix approximation'
  - Kalman Filter typically perform both steps in one, taking fit to previous points as estimate to next point
a charged particle in a homogenous field describes a helix
(except for energy loss and multiple scattering)
a helix is described by five parameters, e.g
\( d_0, \phi_0, \omega, \tan \lambda, z_0 \)
after identifying the hits from one particle fitting the
above parameters determines the 3-momentum (and charge):

\[
p_t = a \left| \frac{B}{\Omega} \right|, \quad p_x = p_t \sin \phi_0, \quad p_y = p_t \cos \phi_0, \quad p_z = p_t \tan \lambda
\]
reconstruction - clustering

- clustering
- combine hits that most likely belong to a particle shower
- compute energy of shower

- typically based on some metric that links nearby hits “Nearest-Neighbor”
- could additionally use
  - tracks as seeds
  - hit energy amplitude
example NNClustering

```c++
template <class In, class Out, class Pred>
void cluster( In first, In last, Out result, Pred* pred ) {

typedef typename In::value_type GenericHitPtr;
typedef typename Pred::hit_type HitType;

typedef std::vector< GenericCluster<HitType> >* ClusterList;

ClusterList tmp;
tmp.reserve( 256 );

while( first != last ) [
    for( In other = first+1 ; other != last ; other ++ ) {
        if( pred->mergeHits( (*first) , (*other) ) ) [
            if( (*first)->second == 0 && (*other)->second == 0 ) // no cluster exists
                GenericCluster<HitType>* cl = new GenericCluster<HitType>( (*first) );
                cl->addHit( (*other) );
                tmp.push_back( cl );
        ] else if( (*first)->second != 0 && (*other)->second != 0 ) // two clusters
            (*first)->second->mergeClusters( (*other)->second );
        } else { // one cluster exists
            if( (*first)->second != 0 ) [
                (*first)->second->addHit( (*other) );
            ] else [
                (*other)->second->addHit( (*first) );
            ]
        } // ocut
    } ++first;

} // remove empty clusters
```

**simplest algorithm: nearest neighbor clustering:**
- loop over all hit pairs
- merge hits into one cluster if $d( h_1, h_2) < \text{cut}$
- $d()$ could be 3D-distance – typically more complicated

- in real life the NNClustering does not provide the necessary accuracy, e.g. in dense jets where showers overlap
- -> more advanced algorithms needed and under development/study, e.g.
  - tracking like clustering
  - genetic algorithms
  - unsupervised learning, ....
advanced example: photon shower ID

Choose 
N threshold levels 
(N=10 at the moment) 
and get N sets of hits 

For each set do a 
NN clustering 
Only in particular set!! 

- sophisticated photon ID 
- based on N clusterings with 
different thresholds
reconstruction - PFA

- track cluster merging
  *(particle flow)*
- extrapolate the tracks into the calorimeter and merge with clusters that are consistent with the momentum/direction and energy of the track
- the unmerged cluster are then the neutral particles
- ideally one would like reconstruct every single particle (PFA)
example: reconstruction @ the ILC

- general ILC detector features:
  - precision tracking
  - precision vertexing
  - high granularity in calorimeters
    - (Ecal ~1cm, Hcal ~1-5cm)
- important: very high jet-mass resolution
  - ~30%/sqrt(E/GeV)

Particle Flow

- reconstruct all single particles
- use **tracker** for **charged particles**
- use Ecal for **photons**
- use Hcal for **neutral hadrons**

- dominant contribution (E<50 GeV):
  - Hcal resolution
  - confusion term

\[
\sigma_{E_{\text{jet}}}^2 = \varepsilon_{\text{trk}}^2 \sum_i E_{\text{trk},i}^4 + \varepsilon_{\text{Ecal}}^2 E_{\text{Ecal}}^4 + \varepsilon_{\text{Hcal}}^2 E_{\text{Hcal}}^4 + \sigma_{\text{confusion}}^2
\]

\[
\varepsilon_{\text{trk}} = \delta (1/p) \approx 5 \cdot 10^{-5}, \quad \varepsilon_{\text{Ecal}} = \frac{\delta E}{\sqrt{E}} \approx 0.1, \quad \varepsilon_{\text{Hcal}} \approx 0.5
\]
Particle flow calorimetry @ ILC

Hardware:

- Need to be able to resolve energy deposits from different particles
- Highly granular detectors (as studied in CALICE)

Software:

- Need to be able to identify energy deposits from each individual particle!
- Sophisticated reconstruction software

★ Particle Flow Calorimetry = HARDWARE + SOFTWARE
example PandoraPFA clustering

i. Preparation (MIP hit ID, isolation, tracking)
ii. Loose clustering in ECAL and HCAL
iii. Topological linking of clearly associated clusters
iv. Courser grouping of clusters
v. Iterative reclustering
vi. Photon Recovery (NEW)
vii. Fragment Removal (NEW)
viii. Formation of final Particle Flow Objects
(reconstructed particles) – not very sophisticated

Order inter-changable

If track momentum and cluster energy inconsistent: RECLUSTER

18 GeV
30 GeV
10 GeV

12 GeV

Pandora is the most sophisticated and best performing PFA to date
example: ILC - Detector Concept Study

recently* three international detector concepts in R&D

Concepts currently studies differ mainly in **SIZE** and aspect ratio

Relevant: inner radius of ECAL: defines the overall scale

- **SiD**: Silicon based concept
- **GLD**: even larger detector concept
- **LDC**: large detector concept

need of sophisticated **Monte Carlo Simulation** programs as well as full **reconstruction** tools to improve and compare the different detector concepts

* now ILD, SID, 4th
Monte Carlo for detector optimization

- vary detector parameters, e.g.:
  - Hcal thickness
  - Tracking radius
  - B-field strength

- use Monte Carlo for (cost conscious) optimization of detector
HEP Software Frameworks
From generated 4-vectors and/or data
to published histograms
ILC Monte Carlo software chain

Persistency Framework

Generator

Simulation

Reconstruction

Analysis

geometry/conditions

Generator

Simulation

Reconstruction

Analysis

Java, C++, Fortran

Geant3, Geant4
ILC Monte Carlo software chain

Generator

Simulation

Reconstruction

Analysis

LCIO

Geometry/Conditions

Mokka

Simdet

Marlin

MarlinReco

Brahms

Brahms

Brahms

LCIO

Java, C++, Fortran

Geant3, Geant4

Java, C++

Fortran

Java, C++, Fortran

Java, C++, Fortran

LCIO

Mokka

Simdet

Marlin

MarlinReco

Brahms

Brahms

Brahms

Simulation

Reconstruction

Analysis

GEAR

LCCD
LCIO overview

- DESY and SLAC joined project:
  - provide common basis for ILC software
- Features:
  - Java, C++ and f77 (!) API
  - extensible data model for current and future simulation and testbeam studies
  - user code separated from concrete data format
  - no dependency on other frameworks

**simple & lightweight**

- current release: v01-11
event data model

LCIO DataModel Overview

Monte Carlo

- SimTrackerHit
- MCParticle
- SimCalorimeterHit

RawData

- TrackerRawData
- TrackerData
- RawCalorimeterHit

Digitization

- CalorimeterHit

Reconstruction & Analysis

- Cluster
- Reconstructed Particle
- Track
- Particle
example MCParticle data class

virtual ~MCParticle ()
    Destructor.

virtual double getEnergy () const=0
    Returns the energy of the particle (at the vertex) in [GeV] computed from
    particle's momentum and mass - only float used in files.

virtual const MCParticleVec & getParents () const=0
    Returns the parents of this particle.

virtual const MCParticleVec & getDaughters () const=0
    Returns the daughters of this particle.

virtual int getNumberOfParents () const=0
    Returns the number of parents of this particle - 0 if mother.

virtual MCParticle * getParent (int i) const=0
    Returns the i-th parent of this particle.

virtual int getPDG () const=0
    Returns the PDG code of the particle.

virtual int getGeneratorStatus () const=0
    Returns the status for particles as defined by the generator, typically:
    0 empty line
    1 undecayed particle, stable in the generator
    2 particle decayed in the generator
    3 documentation line.

virtual int getSimulatorStatus () const=0
    Returns the status for particles from the simulation, e.g.

virtual bool isCreatedInSimulation () const=0
    True if the particle has been created by the simulation program (rather
    than the generator).

virtual bool isBackscatter () const=0
    True if the particle was created by the simulator as a result of an interac-
    tion decay in non-tracking region, e.g.

virtual bool vertexIsNotEndpointOfParent () const=0
    True if the particle was created as a result of a continuous process where
    the parent particle continues, i.e.

virtual bool isDecayedInTracker () const=0
    True if the particle decayed or interacted in a tracking region.

virtual bool isDecayedInCalorimeter () const=0
    True if the particle decayed or interacted (non-continuous interaction, particle
    terminated) in non-tracking region.

virtual bool hasLeftDetector () const=0
    True if the particle left the world volume undecayed.

virtual bool isStopped () const=0
    True if the particle lost all kinetic energy inside the world volume and did not
    decay.

virtual const double * getVertex () const=0
    Returns the production vertex of the particle in [mm].

virtual float getTime () const=0
    The creation time of the particle in [ns] wrt.

virtual const double * getEndpoint () const=0
    Returns the endpoint of the particle in [mm] if the endpoint has been set
    explicitly.

virtual const double * getMomentum () const=0
    Returns the particle's 3-momentum at the production vertex in [GeV] only float
    used in files.

virtual double getMass () const=0
    Returns the mass of the particle in [GeV] - only float used in files.

virtual float getCharge () const=0
    Returns the particle's charge.

virtual int getNumberOfDaughters () const=0
    Returns the number of daughters of this particle.

virtual MCParticle * getDaughter (int i) const=0
    Returns the i-th daughter of this particle.
Persistency – file formats I

• most OO-languages such as C++ don't have a built in persistency mechanism

• typically experiments defines their own binary format, due to convenience and efficiency reasons

• -> need tools/code to 'persist' class contents to files

• preferences and requirements change
LCIO class design

- **Abstract Event**
  - `EVENT::LCEvent`
  - abstract methods: `getRunNumber()`, `getEventNumber()`, `getEventHeader()`, `compareTo()`, `compareTo(event)`

- **Abstract IO**
  - `IO::LCIOWriter`
  - abstract methods: `open()`, `writeRunHeader()`, `writeEvent()`, `close()`
  - `IO::LCIOReader`
  - abstract methods: `open()`, `readRunHeader()`, `readEvent()`, `close()`

- **Concrete Classes**
  - `LCEventImpl`
    - private fields: `runNumber`, `eventNumber`, `eventHeader`, `eventTimestamp`, `collectionNames`, `collection`
    - constructors: `LCEventImpl()`, `LCEventImpl(runNumber, eventNumber, eventHeader)`
  - `LCIOWriterImpl`
    - private fields: `eventRecord`, `runRecord`, `stream`, `eventHeader`, `runHeader`, `collectionMap`, `collectionVec`
    - constructors: `LCIOWriterImpl()`, `LCIOWriterImpl(eventRecord)`
  - `LCIOReaderImpl`
    - private fields: `eventRecord`, `runRecord`, `stream`, `eventHeader`, `runHeader`, `collectionMap`, `collectionVec`
    - constructors: `LCIOReaderImpl()`, `LCIOReaderImpl(eventRecord)`

- **Persistency Implementation**
  - `SIEventWriter`
    - private fields: `eventRecord`, `runRecord`, `stream`, `eventHeader`, `runHeader`, `collectionMap`, `collectionVec`
    - constructors: `SIEventWriter()`, `SIEventWriter(eventRecord)`
  - `SIEventReader`
    - private fields: `eventRecord`, `runRecord`, `stream`, `eventHeader`, `runHeader`, `collectionMap`, `collectionVec`
    - constructors: `SIEventReader()`, `SIEventReader(eventRecord)`

- **Miscellaneous**
  - `friend` declaration: `friend SIEventWriter SIEventReader`
Persistency – file formats II

- file formats need to be flexible:
- not always same information stored
- different tasks need different level of detail, ie.
different amount of data, eg.
  - calibration needs all detector signals (Hits)
  - physics analyses need only output of reconstruction, ie. high
    level objects such as reconstructed particles or even jets
- typically several levels of data files exist
  with increased condensation of information and
decreased file sizes
### ATLAS multilevel persistency scheme

<table>
<thead>
<tr>
<th>EDM Level</th>
<th>Contents</th>
<th>Primary Intent</th>
<th>Size/Event (KB)</th>
<th>Max Ideal Input rate (Hz)</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Raw Data Objects</strong></td>
<td>Raw Channels</td>
<td>Reconstruction (calibration)</td>
<td>1600</td>
<td>N/A</td>
<td>Central Reco/Reprocessing: Tier 0/1</td>
</tr>
<tr>
<td><strong>Event Summary Data</strong></td>
<td>Cells, Hits, Clusters, Tracks, MET, Electron, Jet, Muon, Tau, Truth</td>
<td>Derive calibrations, Re-reconstruction, Recalibration</td>
<td>500</td>
<td></td>
<td>CERN CAF (access limited), Tier 1 (on tape)</td>
</tr>
<tr>
<td><strong>Analysis Object Data</strong></td>
<td>Lepton Cells, Hits, Clusters, Tracks, MET, Electron, Jet, Muon, Tau, Slimmed Truth</td>
<td>Limited Re-reconstruction (eg Jets, b-tag), limited recalibration, Analysis</td>
<td>100</td>
<td>1000</td>
<td>Full: Tier 1,2 (disk) Subset: Tier 3</td>
</tr>
<tr>
<td><strong>Derived Physics Data</strong></td>
<td>Any of the above + composites (eg top) + derived quantities (sphericity)</td>
<td>Interactive Analysis: Making plots, performing studies</td>
<td>Typically ~10</td>
<td>106</td>
<td>Tier 3: eg your laptop</td>
</tr>
<tr>
<td><strong>TAG</strong></td>
<td>Summary. Ex: $p_T, \eta$ of 4 best $e, \gamma, \mu, \tau, \text{jet}$</td>
<td>Selection Events for analysis</td>
<td>1</td>
<td>108</td>
<td>Everywhere</td>
</tr>
</tbody>
</table>
Conditions Data:
- all data that is needed for analysis/reconstruction besides the actual event data
- typically has lifetime (validity range) longer than one event
  - can change on various timescales, e.g. seconds to years
  - need for tagging mechanism, e.g. for calibration constants
- example: trigger configuration, temperature readings, gas pressures,
  calibration constants, electronic channels mapping,...
example analysis/reconstruction framework: Marlin

**Modular Analysis & Reconstruction for the Linear Collider**

- modular C++ application framework for the analysis and reconstruction of LCIO data
- uses LCIO as transient data model
- software modules called Processors
- provides main program
- provides simple user steering:
  - program flow (active processors)
  - user defined variables
    - per processor and global
  - input/output files
- **Plug&Play** of processors

```
marlin::main

Digitization

Tracking

Clustering

PFlow

OutputProcessor

LCEvent

MyInput0.slcio
MyInput1.slcio
MyInput2.slcio

MyInput.slcio

read and add collections
```

```
read and add collections
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```

```
... collection0
```
Marlin Processor

- provides main **user callbacks**
- has **own set of input parameters**
  - int, float, string (single and arrays)
  - parameter description
- naturally modularizes the application
- order of processors is defined via steering file:
  - easy to exchange one or several modules w/o recompiling
  - can run the same processor with different parameter set in one job
- processor task can be as simple as creating one histogram or as complex as track finding and fitting in the central tracker

```cpp
marlin::Processor
init()
processRunHeader(LCRunHeader* run)
processEvent( LCEvent* evt)
check( LCEvent* evt)
end()
```

```cpp
UserProcessor
processEvent( LCEvent* evt){
  // your code goes here...
}
```
Analysis Tools – example root

- analysis tools in HEP
- provide core features:
  - ntuples
  - histograms (1D/2D)
  - fitting
  - plotting /publishing

note: the root C++ framework provides much more functionality not discussed here
Analysis Tools – example JAS3


- analysis tools in HEP
- typically also provide:
  - file I/O, file browsers
  - event displays
  - scripting environments
  - integration with experiment software
example: ILD event display

- summer student work 2008: extend event display to view DSTs (no hits available for visualization):
  - show Tracks as helices
  - Clusters as lines & Cylinders (scaled w/ E)
  - jets as cones (E, p_t) + particles coloured
  - 3 momentum of all particles in the event

S. Daraszevic
tools/languages used in HEP

- **OS Machines**
  - CDC
  - NORSK
  - UNIVAC
  - APOLLO
  - VAX
  - SUN
  - HP
  - SGI
  - LINUX
  - MAC
  - WINDOWS

- **Compiled Languages**
  - FORTRAN IV
  - F77
  - OBJC
  - JAVA
  - C++

- **Code Management**
  - UPDATE
  - PATCHY
  - CMZ
  - CVS
  - SVN
  - CMCT

- **Data Structures I/O**
  - BOS
  - ZBOOK
  - HYDRA
  - ZEBRA
  - FATMEN
  - MySQL/Oracle
  - ROOT

- **GUI Graphics**
  - GD3
  - X11
  - CORE
  - GKS
  - HIGZ
  - MOTIF
  - JAVA
  - GL
  - GT
  - ROOT
  - AIDA
  - ROOT

- **Histograms Statistics**
  - SUMX
  - HBOOK
  - AIDA
  - ROOT

- **Scripting Interpreters**
  - SIGMA
  - TkTcl
  - COMIS
  - KUIP
  - Perl
  - Python
  - SIGMA
  - TkTcl
  - COMIS
  - KUIP
  - Perl
  - Python

- **Interactive Analysis**
  - GEP
  - PAW
  - JAS
  - ROOT

- **Detector Simulation**
  - EGS
  - GEANT1.2
  - MCNP
  - GHEISHA
  - GEANT3
  - MARS/MCNP
  - ROOT/VNC
  - GEANT4
multi language frameworks

- different languages have different advantages in different application domains
- programmers have different skills (and preferences!)
  - Atlas and CMS: more than 2000 physicists
- multi language frameworks are a possible way to 'keep everyone happy'

- example:
  - Atlas sw framework:
    - core sw in C++
    - python binding for modules
    - main application incl. configuration and simple analyses in python

![Diagram showing interactions between Python and C++ for configurable Electron Reconstruction and EMShower with automatically generated interfaces.](image)
Computing Infrastructure - Hardware
Mass Storage

- Mass storage of HEP data and Monte Carlo is typically done on tapes.
- E.g. @ DESY we have (8/2006):
  - 22,000 tapes
  - 46 tape drives
  - 4 robots
  - ~ 1.6 PetaByte data (mostly HERA experiments and Monte Carlo)

Access to data on tape fairly slow
-> need smart disk caching system
**dCache**

- **SRM** for storage management
- **nfs2/3** for name space
- **dCap, xRoot** for random LAN
- **gsiFtp, http(g)** for random WAN

- **dCache developed @ DESY&FNAL**
- **transparent access to files on tape via pnfs file system**
- **various protocols for worldwide access and file transfer**

**SRM provides interface to grid**

Osm, Enstore, Tsm, Hpss, ...

Computing platforms I

- the main working horse for HEP computing today are **large PC clusters or farms**, which are mostly operated with **linux**

- typically **the high level trigger** operates on a **dedicated experiment specific farms** in order to guarantee the throughput needed

- **Monte Carlo production, reconstruction and analysis** run on often on **shared farms**
  - shared between tasks
  - shared between experiments (→ institute batch system)
  - shared between institutes (→ grid)
Computing platforms II
### Computing requirements

<table>
<thead>
<tr>
<th>Application</th>
<th>Input</th>
<th>Output</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC generator</td>
<td>none</td>
<td>small</td>
<td>little</td>
</tr>
<tr>
<td>MC simulation</td>
<td>small</td>
<td>large</td>
<td>huge</td>
</tr>
<tr>
<td>MC digitization</td>
<td>large/huge</td>
<td>large</td>
<td>little</td>
</tr>
<tr>
<td>Reconstruction</td>
<td>large</td>
<td>large</td>
<td>large</td>
</tr>
<tr>
<td>Analysis</td>
<td>large</td>
<td>small</td>
<td>small</td>
</tr>
</tbody>
</table>

**example:**
- simulating (geant4) an ILC event takes ~200s on a standard PC (2007) (an Atlas event takes ~600s !)
- $O(10^6)$ events will take 10 CPU years !
- need hundreds (thousands) of CPUs to get events in reasonable time
increase of computing power

- **Moore's law:**
  - number of transistors per chip doubles every ~2y
  - ~true for >40 years!
- however:
  - computing power does not grow at the same speed
  - -> cache, memory, architecture
- requirements grow even faster:
  - more and larger data sets
  - complexity of algorithms
  - 'sloppiness' of programmers (why bother writing efficient code – the next generation of CPUs will run the slower/larger algorithm as fast ...)

- the ever increasing need for more
- computing power leads to larger and larger farms (thousands of CPUs)
- problem (cost):
  - power and cooling
  - need for new facilities
replace old 'junk'

- hardware older than 2–4 years is too expensive to operate → replace w./ new hardware
- use water cooled racks (more efficient)
- increase density of CPUs per rack
  -> multi core CPUs

Electricity bills for computing will become a large fraction of labs' total budgets!
**multi core CPUs**

- HEP computing is 'embarrassingly parallel':
  - process one event at a time on one computer
  - → total processing power scales w/ number of CPUs
  - HEP computing paradigm of 90s and 0s

- current trend: multi core CPUs
  - memory (RAM and L2 cache) does not scale w. # cores
  - processing one event per core is inefficient
  - → need new computing paradigms and tools
    - multithreading (non trivial)
    - dedicated compilers that do multithreading for you
    - smaller memory footprint of programs
Grid Computing

“Sharing resources within Virtual Organizations in a global world.”
LHC Computing model

- One bunch crossing per 25 ns
- 100 triggers per second
- Each event is ~1 Mbyte

Tier 0
- CERN Computer Centre >20 TIPS
- 1 TIPS = 25,000 SpecInt95
- PC (1999) = ~15 SpecInt95
- ~100 MBytes/sec

Tier 1
- US Regional Centre
- Italian Regional Centre
- French Regional Centre
- RAL Regional Centre
- ~1 TIPS

Tier 2
- ScotGRID++ ~1 TIPS
- Tier 2 Centre ~1 TIPS
- Centre TIPS
- ~Gbits/sec

Tier 3
- Institute ~0.25TIPS
- Institute ~0.25TIPS
- Institute ~0.25TIPS
- Physics data cache
- 100 - 1000 Mbits/sec
- Workstations

Physicists work on analysis "channels"
Each institute has ~10 physicists working on one or more channels
Data for these channels should be cached by the institute server
ATLAS grid computing model

- Resources Spread Around the GRID
  - Reprocessing of full data with improved calibrations 2 months after data taking.
  - Managed Tape Access: RAW, ESD
  - Disk Access: AOD, fraction of ESD

- Derive 1st pass calibrations within 24 hours.
- Reconstruct rest of the data keeping up with data taking.

Tier 0
- CERN Analysis Facility
  - Primary purpose: calibrations
  - Small subset of collaboration will have access to full ESD.
  - Limited Access to RAW Data.

Tier 1
- RAW/AOD/ESD

Tier 2
- RAW

Tier 3
- Interactive Analysis
- Plots, Fits, Toy MC, Studies, ...

30 Sites Worldwide

DPD
Grid Definition


“A Grid is a system that:

- coordinates resources which are not subject to centralized controls …
  - integration and coordination of resources and users of different domains vs. local management systems (batch systems)
  - … using standard, open, general-purpose protocols and interfaces …
    - standard and open multi-purpose protocols vs. application specific system
- … to deliver nontrivial qualities of services.”
The Grid dream

- Mobile Access
- Desktop
- Visualizing
- Supercomputer, PC-Cluster
- Data Storage, Sensors, Experiments
- Internet, Networks

Visualizing Supercomputer, PC-Cluster Data Storage, Sensors, Experiments Internet, Networks
Grid Types

• **Data Grids:**
  - Provisioning of transparent access to data which can be physically distributed within Virtual Organizations (VO)

• **Computational Grids:**
  - allow for large-scale compute resource sharing within Virtual Organizations (VO)

• **Information Grids:**
  - Provisioning of information and data exchange, using well defined standards and web services
Grid Ingredients

• **Authorization:**
  • Users must be registered in a **Virtual Organization (VO)**

• **Information Service:**
  • Provide a system which keeps track of the available resources

• **Resource Management:**
  • Manage and exploit the available computing resources

• **Data Management:**
  • Manage and exploit the data
Grid: Authentication & Authorization

• a user is uniquely identified through a certificate
  - an encrypted electronic document, digitally signed by a Certification Authority (CA)
  - a certificate is your passport to enter the grid world
  - example: /O=GermanGrid/OU=DESY/CN=Frank Gaede

• access to resources is provided (controlled) via membership in a Virtual Organization
  - a dynamic collection of individuals, institutions, and resources which is defined by certain sharing rules
  - the VO a user belongs to is not part of the certificate.
  - a VO is defined in a central list, e.g. a LDAP tree.
Grid Middleware

Globus:
• Toolkit
• Argonne, U Chicago

EDG (EU DataGrid):
• Project to develop Grid middleware
• Uses parts of Globus
• Funded for 3 years (01.04. 2001 - 31.03.2004)

LCG (LHC Computing Grid):
• Grid infrastructure for LHC production
• Based on stable EDG versions plus VDT etc.
• LCG-2 for Data Challenges

EGEE (Enabling Grids for E-Science in Europe)
• Started 01.04.2004 for 2 + 2 years
• developed gLite as successor of LCG middleware
Job submission to the Grid

requirements:
- grid certificate
- VO membership
- all files (input, binary, libraries,...) on SE

jobscript (JDL) that:
- retrieves all needed files from SE onto WN
- sets custom environment
- executes binary/script
- stores output on SE

submission:
- start your proxy (secured interface)
- put all job depended files on SE
- write your JDL script specifying:
  - name of jobscript
  - arguments
  - input/output Sandbox
  - VO
- edg-job-submit *your-jdl-file*
- check status via job-ID
thanks to A.Gellrich for Grid material