HEP Computing

René Brun /CERN
Software Evolution in HEP

René Brun /CERN
Software Evolution in HEP

A Dynasaur perspective

René Brun /CERN
Software Evolution in HEP
Frameworks and Libraries

Nixon       Ford    Carter  Reagan      Bush    Clinton           Bush2               Obama
Pompidou   Giscard       Mitterand                Chirac              Sarkozy   Hollande


René Brun /CERN
**Detailed Info here**

---

**CERN**

*European Organization for Nuclear Research*

---

**Fifty years of research at CERN, from past to future**

---

www.scribd.com/doc/61425123/Cern-History

---

D.O. Williams

---

Machines

From Mainframes ➞ Clusters

Walls of cores
Machine Units (bits)

- 16 bits: pdp 11, many
- 32 bits: many, univac
- 36 bits: nord50
- 48 bits: besm6
- 56 bits: cdc, many
- 60 bits: many
- 64 bits: many

With even more combinations of exponent/mantissa size or byte ordering

A strong push to develop portable machine independent I/O systems
User machine interface
Interactive Systems

Weekly interactive users 1987-2000

Linux or MAC laptops and desktops
Man Machine Interface

Questions for a selection board?

Programmers

- To be a good programmer some part of your mind must sympathise with a machine – which has to do what it is told, so must be told – in all gory detail – exactly what it must do.
- If you can communicate well with a machine can you also communicate well with people?
- And can you communicate without your machine?
Languages and Operating Systems

- Many software engineers thought that physicists were renegades and would only ever write in FORTRAN? But things moved.
- F-77, and also C (portable assembler) and C++ (object orientation)
- Basic → Java for interactivity
- Not to forget scripting languages – primarily to interact with OSs
- SCOPE, MVS+Wylbur, VM/CMS, VMS, Windows, Unix,
- Digital and Norsk Data died for not recognising Unix quite enough
- And in the future ...?
- What role for Open Source?

As seen by D.O.W in 2004

I do not know what the next language will do, but I know that it will be called Fortran
P. Zanella 1991
The Tevatron/RHIC/ LHC (and many others) software is a big success. Unprecedented data volumes have been analyzed world-wide by teams nicely organized. The solutions in the experiments are different, but all working.
Systems in 1980

- End user Analysis software: 10 KLOC
- Experiment Software: 100 KLOC
- Libraries: HBOOK, Naglib, cernlib: 500 KLOC
- OS & fortran: 1000 KLOC
- CDC, IBM: Vax780

RAM: 1 MB

Tapes
Systems today

- End user Analysis software: 0.1 MLOC
- Experiment Software: 4 MLOC
- Frameworks like ROOT, Geant4: 5 MLOC
- OS & compilers: 20 MLOC
- Clusters of multi-core machines 10000x8

Networks: 10 Gbit/s
Disks: 10 PB

RAM: 16 GB

CHP2012: René Brun: Software Evolution in HEP
What went wrong?

How was it possible in 1981 to make a full detailed simulation of OPAL on a machine with 1 MB RAM?

Where a full detailed simulation of CMS requires more than 1 GB of RAM.

Is CMS more than 1000 times more complex than OPAL?
Terrorists wanted

A strongly biased list by D.O.W
In 2004

Let’s see now some facts
## Exp/Soft life time

<table>
<thead>
<tr>
<th></th>
<th>BaBar</th>
<th>H1</th>
<th>ZEUS</th>
<th>HERMES</th>
<th>Belle</th>
<th>BESIII</th>
<th>CDF</th>
<th>DØ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>End of data taking</strong></td>
<td>07.04.08</td>
<td>30.06.07</td>
<td>30.06.07</td>
<td>30.06.07</td>
<td>30.06.10</td>
<td>2017</td>
<td>30.09.11</td>
<td>30.09.11</td>
</tr>
<tr>
<td><strong>Type of data to be preserved</strong></td>
<td>RAW data Sim/rec level Data skims in ROOT</td>
<td>RAW data Sim/rec level Analysis level ROOT data</td>
<td>Flat ROOT based ntuples RAW data Sim/rec level Analysis level ROOT data</td>
<td>RAW data Sim/rec level</td>
<td>RAW data Sim/rec level</td>
<td>RAW data Rec. level</td>
<td>Raw data Rec. level</td>
<td>Raw data ROOT files (data+MC) ROOT files (data+MC)</td>
</tr>
<tr>
<td><strong>Data Volume</strong></td>
<td>2 PB</td>
<td>0.5 PB</td>
<td>0.2 PB</td>
<td>0.5 PB</td>
<td>4 PB</td>
<td>6 PB</td>
<td>9 PB</td>
<td>8.5 PB</td>
</tr>
<tr>
<td><strong>Desired longevity of long term analysis</strong></td>
<td>Unlimited</td>
<td>At least 10 years</td>
<td>At least 20 years</td>
<td>5-10 years</td>
<td>5 years</td>
<td>15 years</td>
<td>Unlimited</td>
<td>10 years</td>
</tr>
<tr>
<td><strong>Current operating system</strong></td>
<td>SL/RHEL5 3 SL/RHEL 5</td>
<td>SL5</td>
<td>SL5</td>
<td>SL3/SL5</td>
<td>SL5/RHEL5</td>
<td>SL5</td>
<td>SL5 SL6</td>
<td>SL5</td>
</tr>
<tr>
<td><strong>Languages</strong></td>
<td>C++ Java Python</td>
<td>C++ C++ Fortran Python</td>
<td>C++ C++ Fortran Python</td>
<td>C++ C++ Fortran</td>
<td>C++ C++ Fortran Python</td>
<td>C++ C++ Fortran Python</td>
<td>C++ C++ Fortran Python</td>
<td></td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td>GEANT 4</td>
<td>GEANT 3</td>
<td>GEANT 3</td>
<td>GEANT 3</td>
<td>GEANT 3</td>
<td>GEANT 4</td>
<td>GEANT 3</td>
<td>GEANT 3</td>
</tr>
<tr>
<td><strong>External dependencies</strong></td>
<td>ACE CERNLIB CLHEP CMLOG Flex GNU Bison MySQL Oracle ROOT</td>
<td>CERNLIB FastJet NeuroBayes Oracle ROOT</td>
<td>ROOT</td>
<td>ADAMO CERNLIB ROOT</td>
<td>Boost CERNLIB NeuroBayes PostgresQL ROOT</td>
<td>CASTPR CERNLIB CLHEP HepMC ROOT</td>
<td>CERNLIB NeuroBayes Oracle ROOT</td>
<td>Oracle ROOT</td>
</tr>
</tbody>
</table>

*See presentation by D.Mouth on Thursday at 9h30*
Some Important Events

- ECFA workshop in Vilars
- Aachen LHC workshop
- MOOSE
- RD44 RD45
- DRDC
- GEM flop
- Erice workshop
- ROOT
- WEB
- FNAL/RHIC go ROOT

Where to go with DS tools & languages?

Powerful tools but programming with them was odd.
Pools of banks (divisions)
Keep together DS
Structural & Reference links
No memory leaks
Self-describing banks
Machine independent I/O
Banks documentation tools

Odd style programming
One single store
Too many globals in /common blocks/

Attempts to implement zebra in Fortran90 failed in 1991.
Very hard to implement a reflexion system

In 1991 we did not know that he will be very hard to implement a reflexion system with C/C++

Q(JMA + 6) = A
Q(JMA + 7) = Z
Q(JMA + 8) = DENS
Q(JMA + 9) = RADL
Q(JMA + 10) = ABSL
Q(JMA + 11) = 1.
An essential tool viewing data structures

Thanks to structural (owners) pointers and references

More important than UML class diagrams!!
Some messages from the past
1994: Move to C++

- Painful but successful, even if it took 10 years!!
- Too many false hopes with Objectivity
- Missing support for Reflexion
- Abuse of inheritance
- Long learning phase to make modular shared libs
- Abuse of « new » generating scattered structures
- Missing concept of ownership
No ownership in C++

Unlike ZEBRA with *structural* and *reference* links, there is no concept of ownership in C++. This has many unwanted consequences:

- Memory leaks, double delete
- Complex algorithms during I/O or deep copy operations to avoid *circular dependencies*.
- No « *deep_sizeof*(pointer) »
- No way to *visualize* a tree/graph structure.
ROOT Trees vs Objectivity

Compared to the best OODBMS candidate in 1995 (Objectivity) ROOT supports a persistent class that may be a subset of the transient class.

ROOT supports compression (typical factors 3 to 6), file portability and access in heterogeneous networks.

ROOT supports branch splitting that increases drastically the performance when reading.

It is based on classical system files and does not require the nightmare of a central data base.
Development Process

- **Software committees** generate more bureaucracy than practical results.

- Launching a new major system requires **agility**, a lot of prototyping between a few people (2,3,4), facing realities soon enough. It is a rewarding challenge!!

- Developing a good **automatized test suite** may be as much work as the development of the system itself.

- Software metrics, dynamic & static code analyzers (eg **Coverity**) are essential.

- **User support with instantaneous response time** is a must.
World-Wide effort

Of course, the development of the experiment specific software has always been (and is more and more) an international effort.

This is also true for the development of tools and libraries. In the case of ROOT essential contributions from RHIC, FNAL and FAIR. It is good to understand and inject the requirements from experiments at the time when they design their software.
Automated meaningful test suites with static & dynamic test coverages are essential.

D.O.W 2004

Computation will never be easy because of complexity
- We probably don’t explain the source of the complexity enough
- Take a serious application from 1970 - say 10,000 lines of code (l.o.c.)
- Assume that every 10 l.o.c. we encounter a 2-way logical decision, where the route taken depends on the input data, or on the prior state of calculations. That is a very conservative estimate.
- There are, therefore, $2^{1000}$ possible routes through our code
- $2^{1000} = (2^{10})^{100} = 10^{300}$ routes! A googol cubed.
- LHC experiments will use a total of some $10^7$ lines of code!!
17 years with ROOT

- After PAW
- GEANT 3
- PAW-like ROOT
- Objectivity era
- FNAL/RHIC Go ROOT
- LCG projects start PI, SEAL, POOL
- ROOT mainstream

Large systems evolve with time thanks to users and lessons learned.

1995  2000  2005  2010
User written streamers filling TBuffer

streamers generated by rootcint

automatic streamers from dictionary with StreamerInfos in self-describing files

member-wise streaming for STL collections<T*>

member-wise streaming for TClonesArray

TreeCache

parallel merge
Some messages for the future
Systems in 2030?

OS & compilers

Frameworks like ROOT, Geant5

Experiment Software

End user Analysis software

1 MLOC

50 MLOC

20 MLOC

100 MLOC

Multi-level parallel machines 10000x1000x1000

Hardware

Networks 10 Tbit/s

Disks 1000 PB

Networks on demand

GRIDS

100 Gbit/s

1 Tbit/s

10 TB

1000 PEB

Keyword: parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism

parallelism
Grid(s) evolution (maybe)
Towards Parallel Software

A long way to go!!

There is no point in just making your code thread-safe. Use of parallel architectures requires a deep rethinking of the algorithms and dataflow.

No committees please!! But well focused projects with well defined milestones and reference benchmarks.

One such project is GEANT → GEANT5 launched 18 months ago. See presentation by Federico
Parallelism: many failures

We failed in vectorizing codes like GEANT3 in 1985-1987 on CRAY, Cyber205, ETA10, IBM3090 because our approach was wrong.

Some successful attempts in online systems in 1983.

We failed too on MPP systems like the Thinking Machines, Elxsi in 1991-1993 because our approach was wrong.

Are we going to take a wrong approach again?
Parallelism: key points

Minimize the sequential/synchronization parts (Amdhal law): Very difficult

Run the same code (processes) on all cores to optimize the memory use (code and read-only data sharing)

Job-level is better than event-level parallelism for offline systems.

Use the good-old principle of data locality to minimize the cache misses.

Exploit the vector capabilities but be careful with the new/delete/gather/scatter problem

Reorganize your code to reduce tails
Data Structures & parallelism

C++ pointers specific to a process

Copying the structure implies a relocation of all pointers

I/O is a nightmare

Update of the structure from a different thread implies a lock/mutex

event

vertices

tracks
Data Structures & Locality

Sparse data structures defeat the system memory caches.

Group object elements/collections such that the storage matches the traversal processes.

For example: group the cross-sections for all processes per material instead of all materials per process.
A killer if one has to wait the end of col(i) before processing col(i+1)

Average number of objects in memory
A better solution

Pipeline of objects

Checkpoint Synchronization.
Only 1 « gap » every N events

This type of solution required anyhow for pile-up studies
Other requirements

Eliminate the sequential part (like merging files) when running jobs/threads/processes in parallel. Use parallel buffer merges instead.

Use efficient tools to monitor bottlenecks like memory allocation, cache misses, too many locks, etc.

Compare the results with the most efficient sequential version and not just the version using one single thread.

Prove that you use a 8 cores-node with one job more efficiently than running 8 independent jobs (memory, cpu, I/O).

This still requires more effort. Urgent!!
Summary

The HEP offline software has evolved with small and large steps from the Fortran libraries with independent subroutines to dynamic linking of C++ classes in several hundred shared libs.

Software development by large collaborations is working.

The convergence in the large HEP community towards general common tools and libraries is now well established.

It would be nice if our current language C++ could include some features that were successful in our previous systems (ownership concept in particular).

We are just starting the effort to understand and use efficiently the emerging parallel architectures.
Wheels are turning
(for me and for CHEP)

It was a wonderful job
a wonderful experience
at a wonderful place
with wonderful colleagues

Many thanks