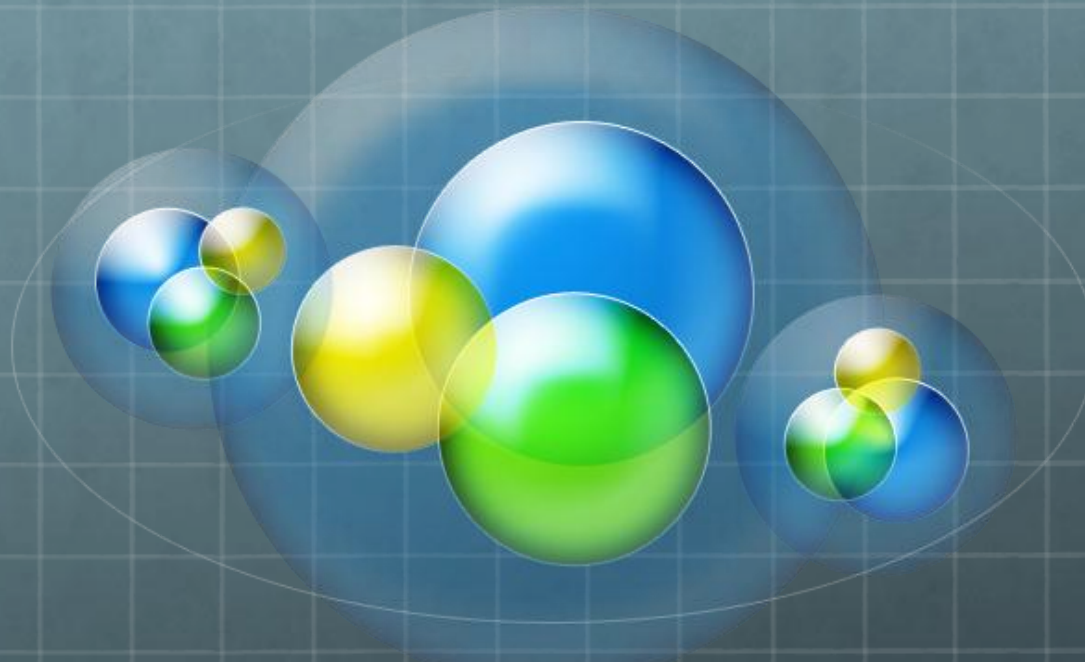


# HEP Computing

**CHEP2012: New York 21 May 2012**  
**René Brun /CERN**



# Software Evolution in HEP

**CHEP2012: New York 21 May 2012**  
**René Brun /CERN**



# Software Evolution in HEP

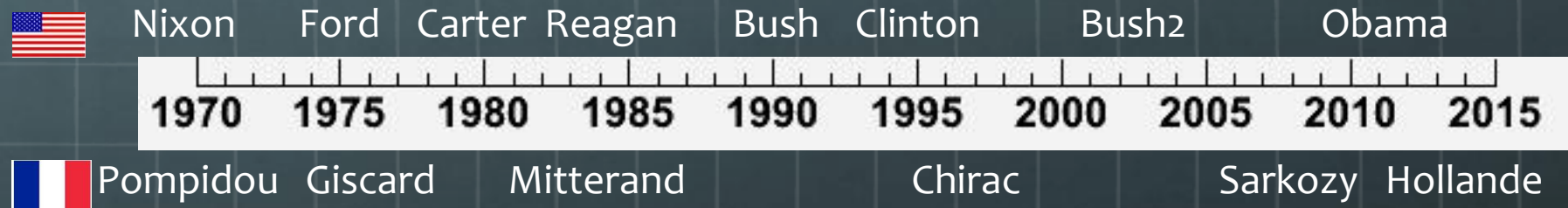
## A **Dynasaur** perspective

CHEP2012: New York 21 May 2012  
René Brun /CERN



# Software Evolution in HEP

## Frameworks and Libraries



**CHEP2012: New York 21 May 2012**  
**René Brun /CERN**



# Detailed Info here



CERN-2006-004  
8 May 2006

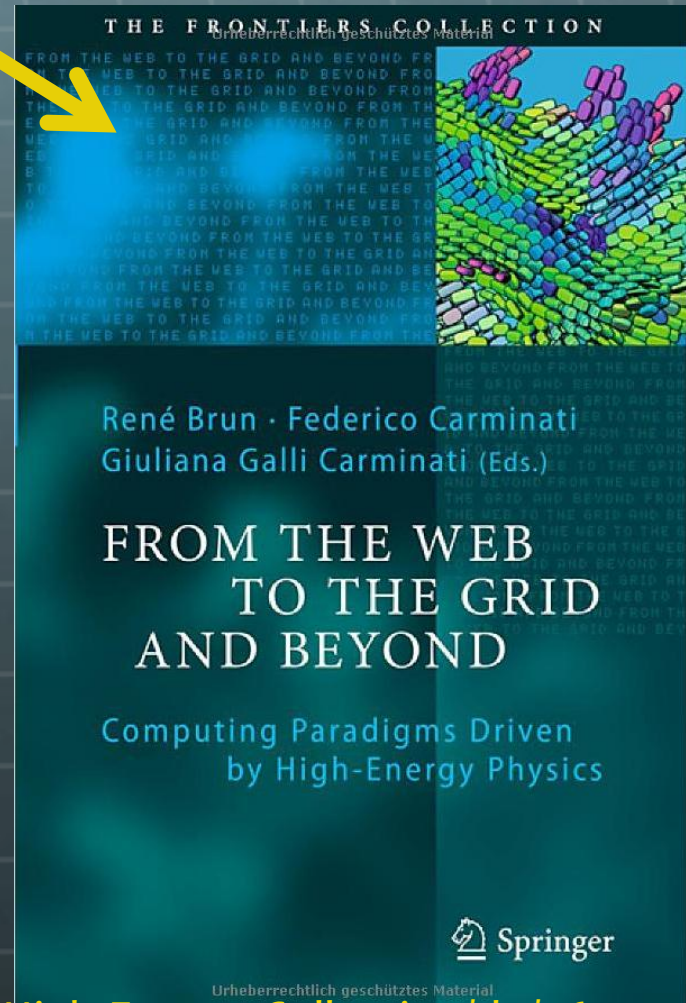
ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE  
**CERN** EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

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

[www.scribd.com/doc/61425123/Cern-History](http://www.scribd.com/doc/61425123/Cern-History)

Academic Training Lectures  
13-16 September 2004, CERN

**D.O. Williams**

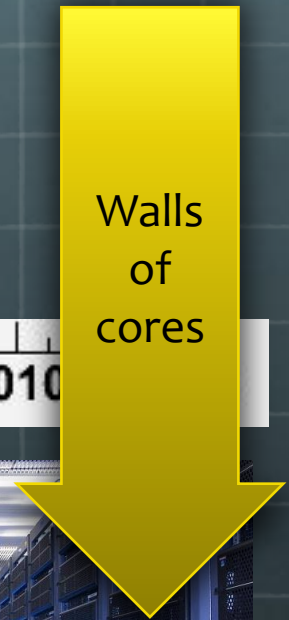


<http://www.amazon.de/Web-Grid-Beyond-High-Energy-Collection/dp/364223156X>



# Machines

From Mainframes =====> Clusters





# Machine Units (bits)

16

pdp 11

32

many

36

univac

48

nord50

56

besm6

60

cdc

64

many

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

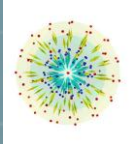
A strong push to develop portable machine independent I/O systems



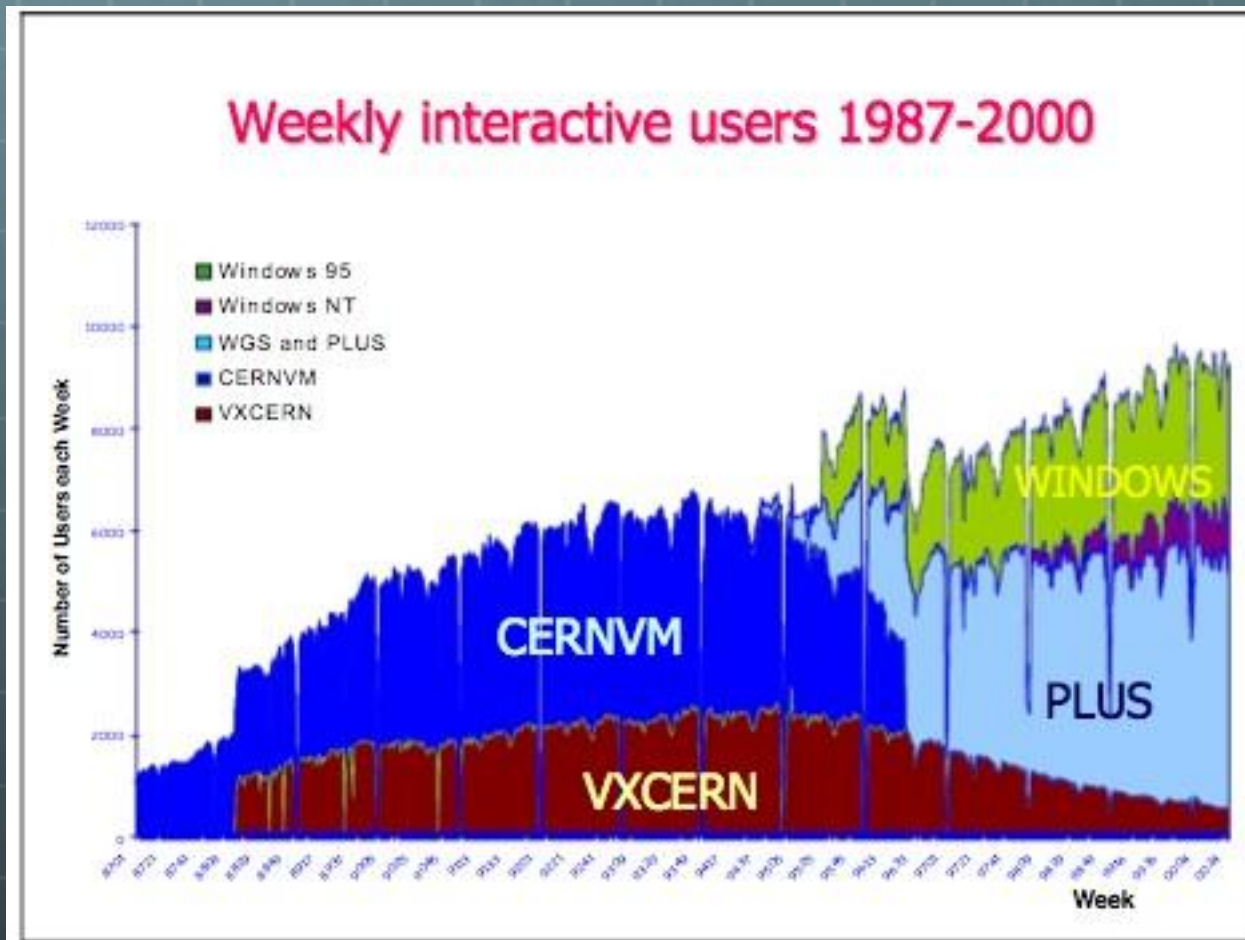
# User machine interface



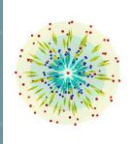




# Interactive Systems



Linux or MAC  
laptops  
and desktops



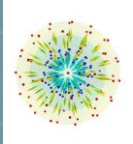
# Man Machine Interface

D.O.W

## 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?

Questions  
for a  
selection  
board?



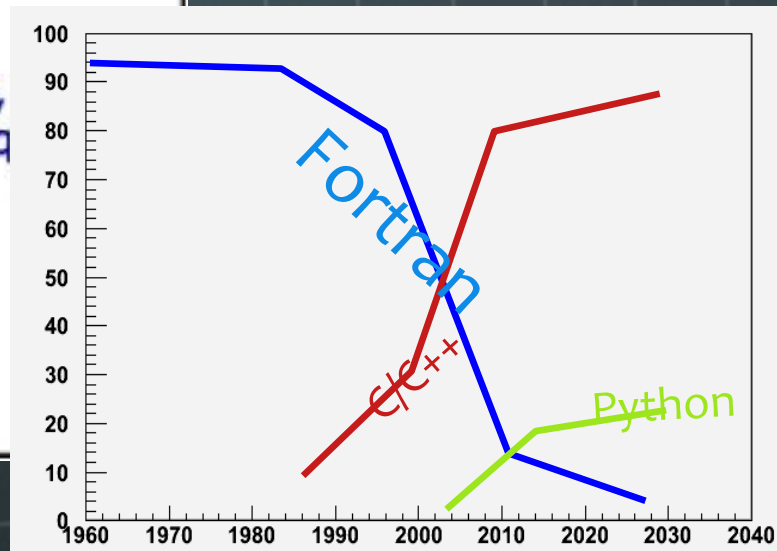
# Languages and OS

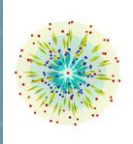
## Languages and Operating Systems

- 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 q 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





# HEP Software

D.O.W

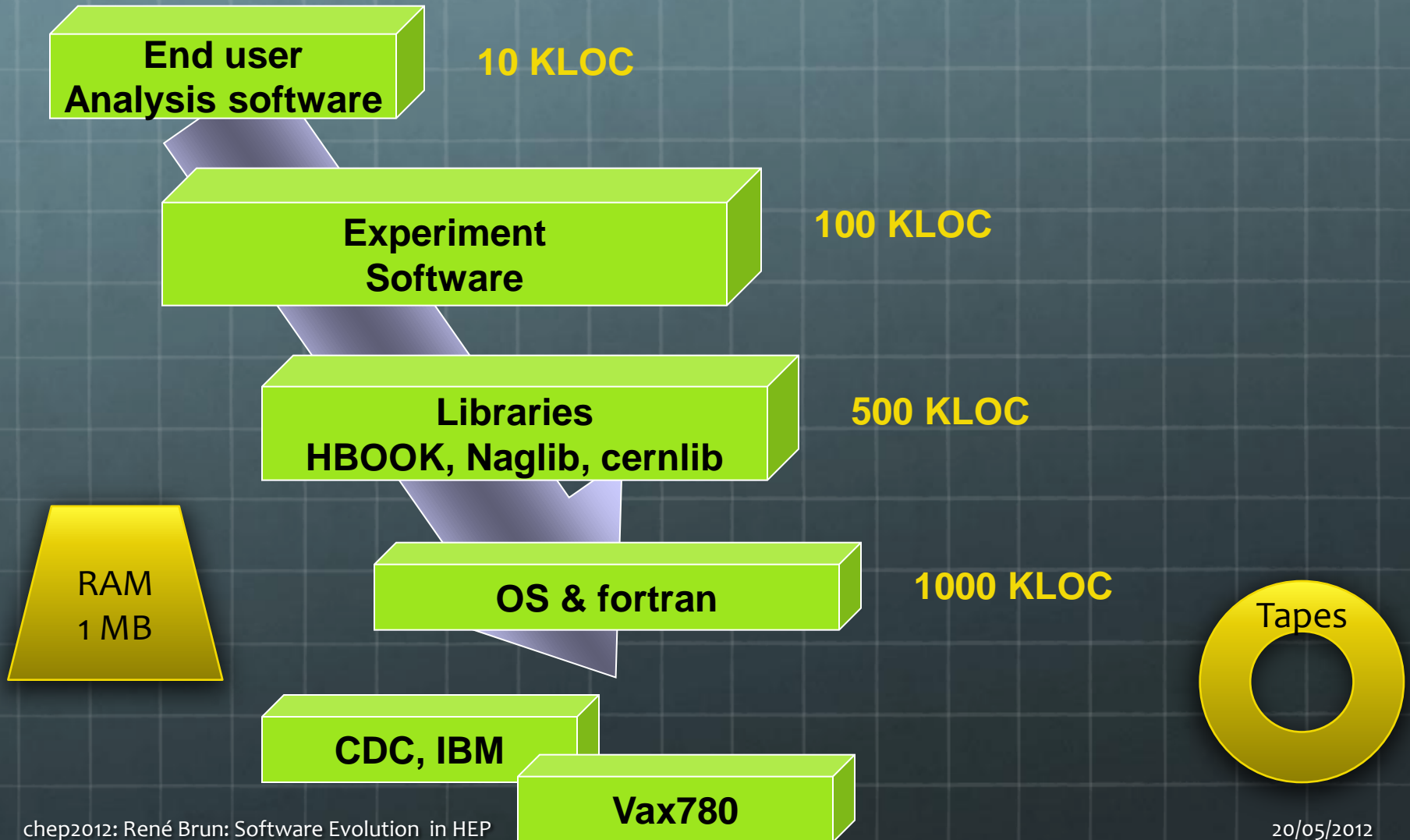
## Did we make any progress with software?

- Answer has to be YES
- In 1960 an experiment involved (I guess) several hundreds, up to a few thousands, of lines of code, running on one single computer, and which had been written by no more than  $\sim 3$  programmers
- In 2000-2005 an LHC experiment involves several tens of millions of l.o.c., running in 1000-10'000 processors, and written by at least  $\sim 300$  programmers
- So, over 40 years,  $10^4$ - $10^5$ - $10^6$  more l.o.c, involving  $O(100)$ x more authors

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.

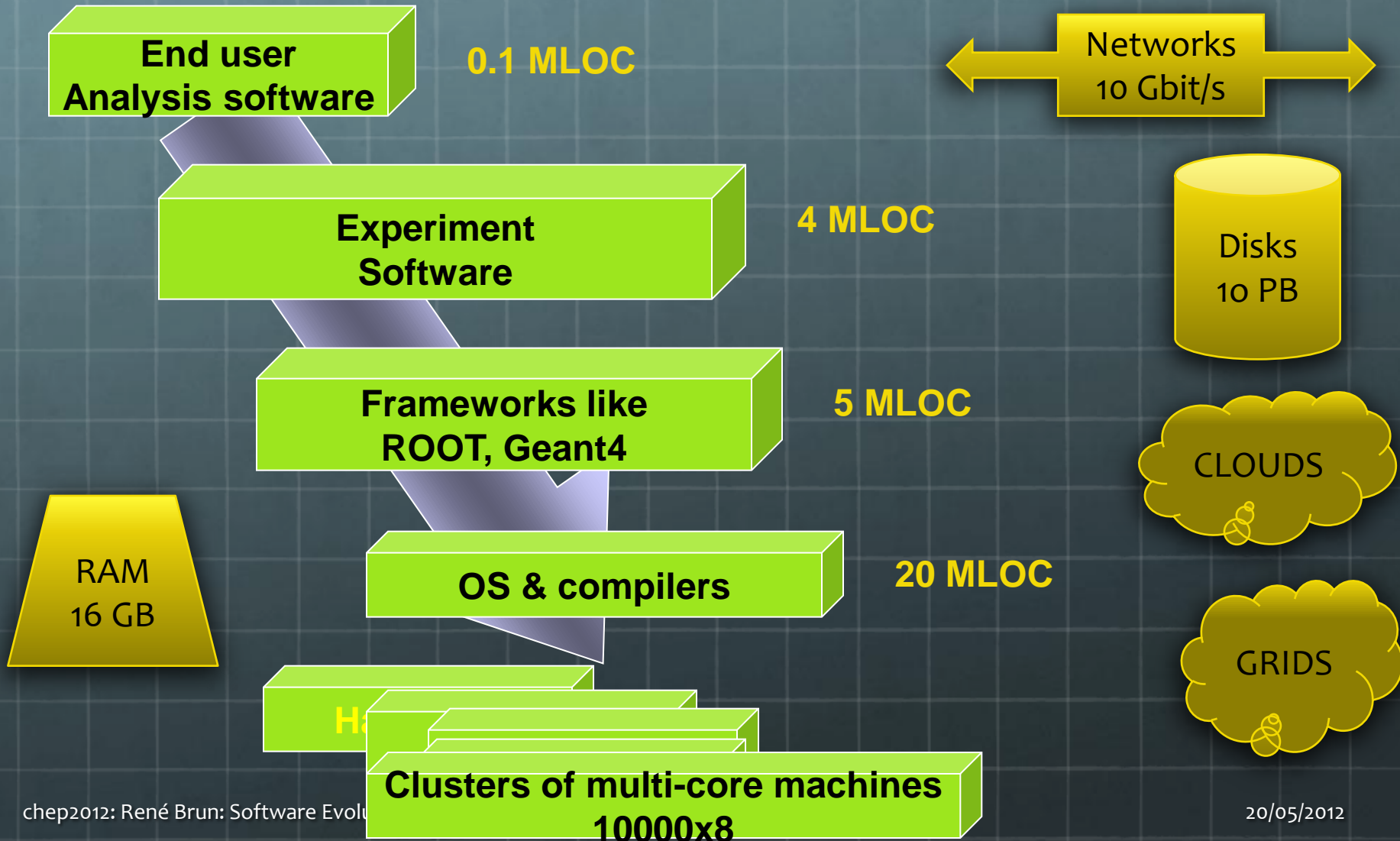


# Systems in 1980





# Systems today



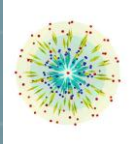


# What went wrong ?

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

## The "offline programmers"

- (Macleod, YGC†), Bock, Zollt, Kellner, Pagiola, Burmeister, Bruyant, Norton
- Brun, Palazzi, Grote, Metcalf, Onions
- Giani, Apostolakis



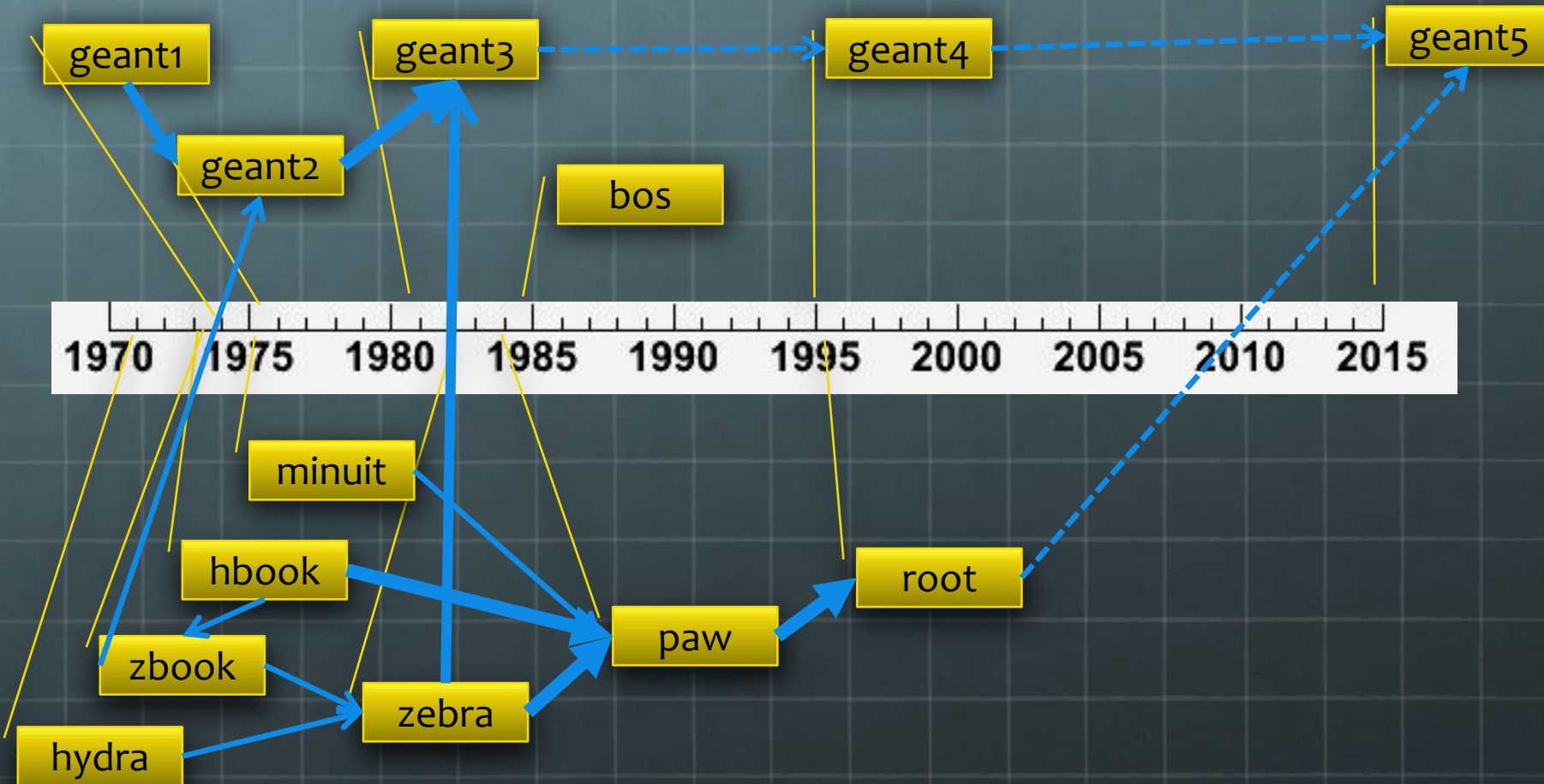
A strongly biased list by D.O.W In 2004

Let's see now some facts





# Tools & Libs





# Exp/Soft life time

	BaBar	H1	ZEUS	HERMES	Belle	BESIII	CDF	DØ
<b>End of data taking</b>	07.04.08	30.06.07	30.06.07	30.06.07	30.06.10	2017	30.09.11	30.09.11
<b>Type of data to be preserved</b>	RAW data Sim/rec level Data skims in ROOT	RAW data Sim/rec level Analysis level ROOT data	Flat ROOT based ntuples	RAW data Sim/rec level Analysis level ROOT data	RAW data Sim/rec level	RAW data Sim/rec level ROOT data	RAW data Rec. level ROOT files (data+MC)	Raw data Rec. level ROOT files (data+MC)
<b>Data Volume</b>	2 PB	0.5 PB	0.2 PB	0.5 PB	4 PB	6 PB	9 PB	8.5 PB
<b>Desired longevity of long term analysis</b>	Unlimited	At least 10 years	At least 20 years	5-10 years	5 years	15 years	Unlimited	10 years
<b>Current operating system</b>	SL/RHEL3 SL/RHEL 5	SL5	SL5	SL3 SL5	SL5/RHEL5	SL5	SL5 SL6	SL5
<b>Languages</b>	C++ Java Python	C C++ Fortran Python	C++	C C++ Fortran Python	C C++ Fortran	C++	C C++ Python	C++
<b>Simulation</b>	GEANT 4	GEANT 3	GEANT 3	GEANT 3	GEANT 3	GEANT 4	GEANT 3	GEANT 3
<b>External dependencies</b>	ACE CERNLIB CLHEP CMLOG Flex GNU Bison MySQL Oracle ROOT TCL XRootD	CERNLIB FastJet NeuroBayes Oracle ROOT	ROOT	ADAMO CERNLIB ROOT	Boost CERNLIB NeuroBayes PostgresQL ROOT	CASTPR CERNLIB CLHEP HepMC ROOT	CERNLIB NeuroBayes Oracle ROOT	Oracle ROOT

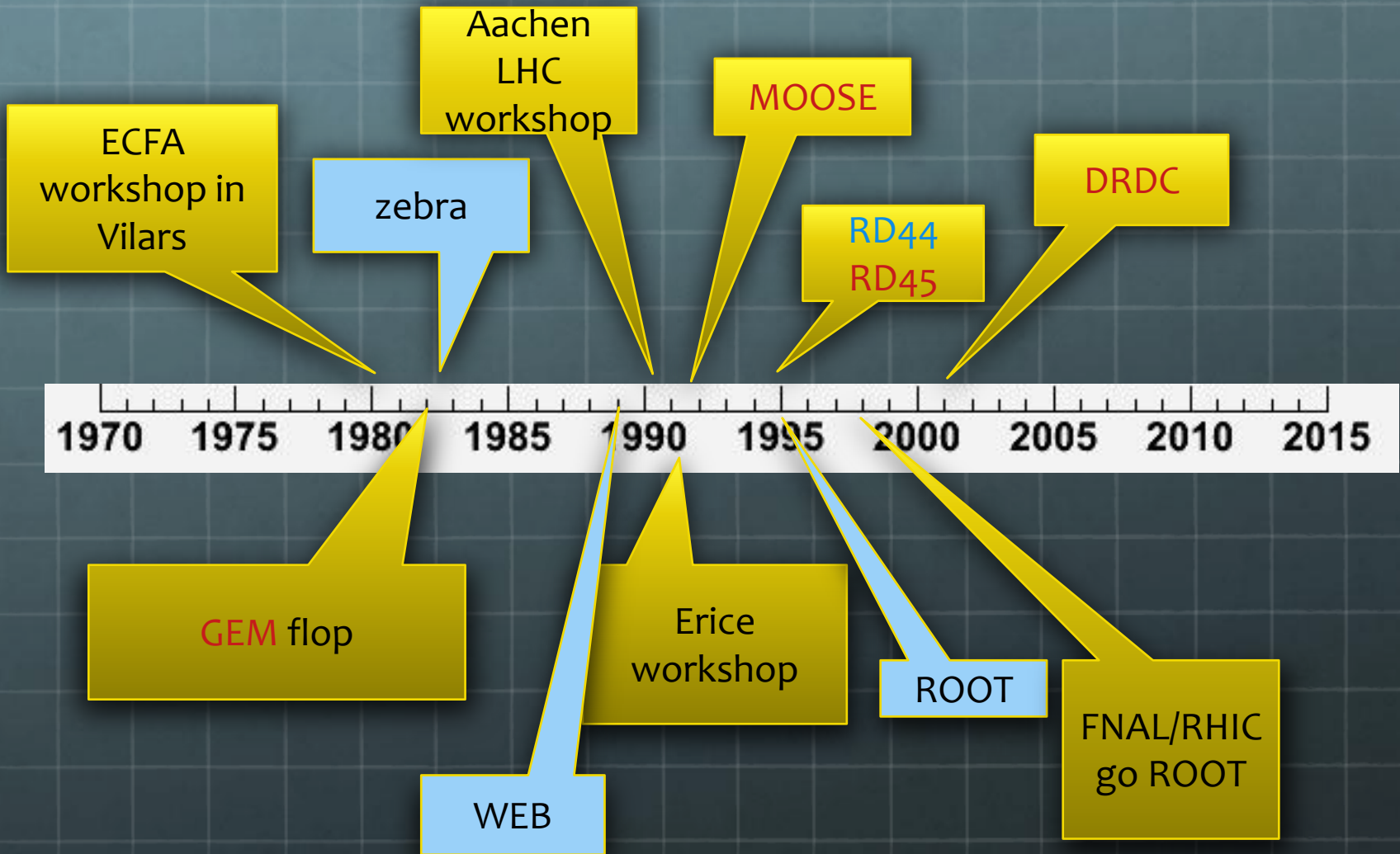
DPHEP DRAFT 2, May 2012

from a recent DPHEP report

See presentation by D.Mouth on Thursday at 9h30



# Some Important Events

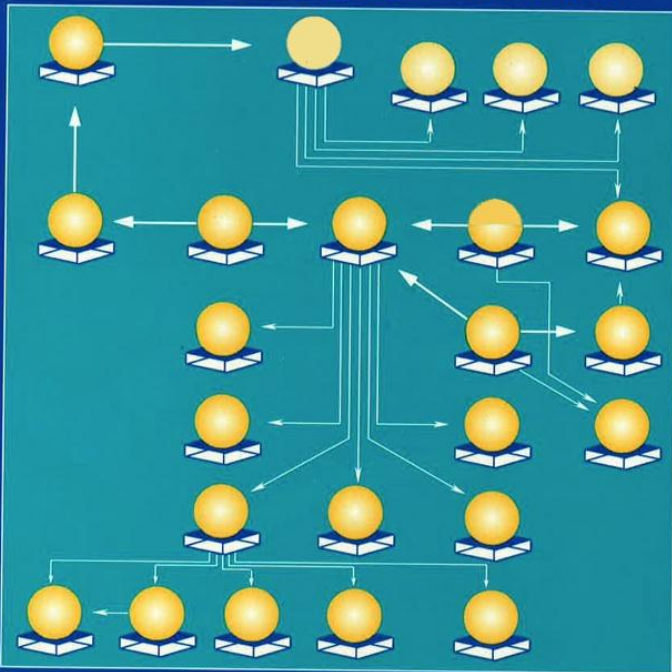




# Erice workshop 1991

Proceedings of the 14th Workshop of the INFN Eloisatron Project

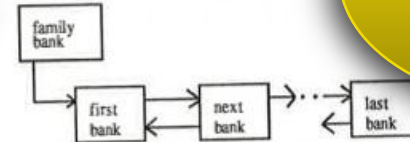
## Data Structures for Particle Physics Experiments: Evolution or Revolution?



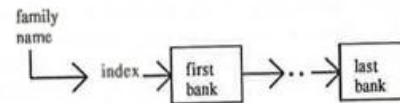
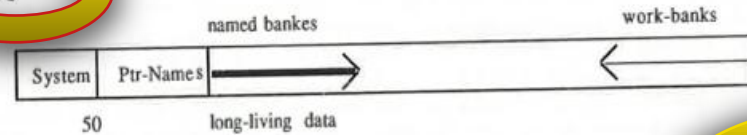
Editors: Rene Brun, Paul Kunz & Paolo Palazzi

World Scientific

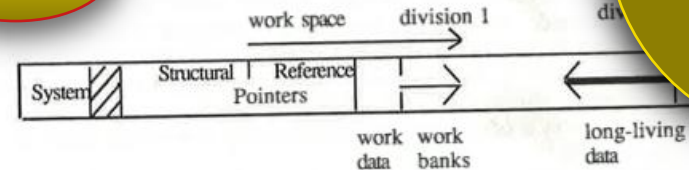
JAZELLE



BOS



ZEBRA



Where to go with DS tools & languages ?

Powerful tools but programming with them was odd



# ZEBRA wonders & oddities

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/

Q(JMA + 6) = A  
Q(JMA + 7) = Z  
Q(JMA + 8) = DENS  
Q(JMA + 9) = RADL  
Q(JMA + 10) = ABSL  
Q(JMA + 11) = 1.

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

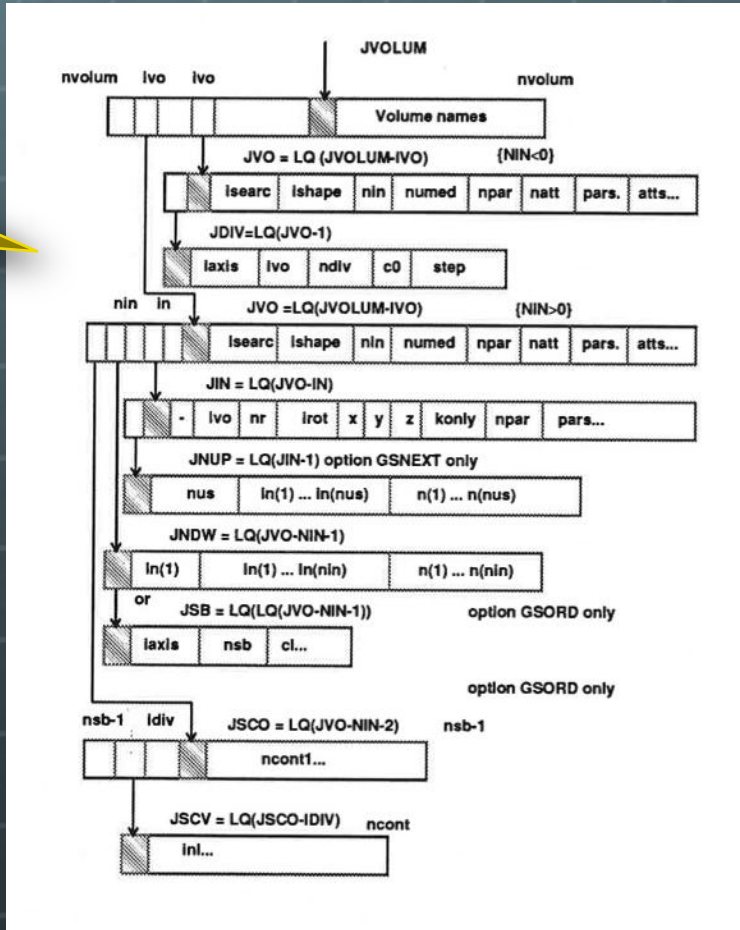
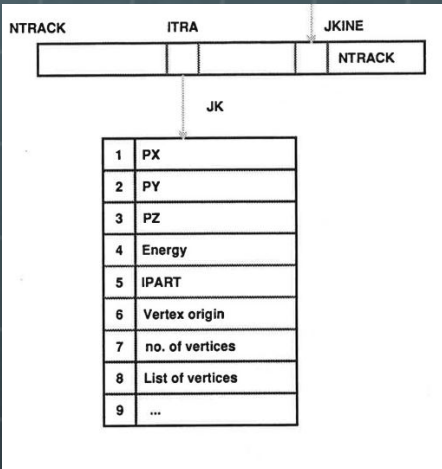
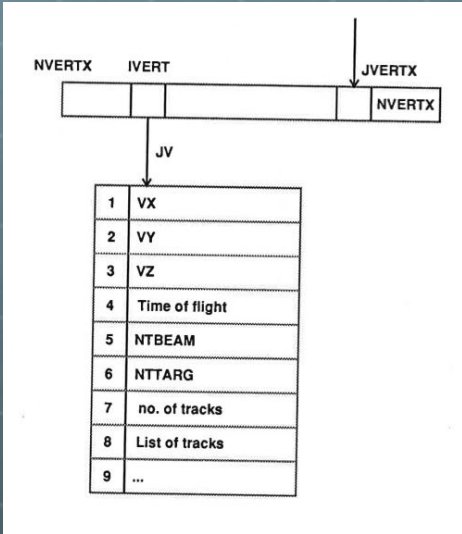
In 1991 we did not know  
that he will be very hard  
to implement a reflexion  
system with C/C++



# An essential tool viewing data structures

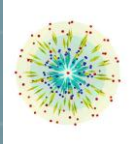
Thanks to structural (owners) pointers and references

More important than UML class diagrams!!





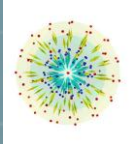
# Some messages from the past



# 1994: Move to C++

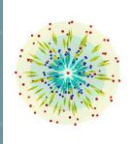
- 🌐 Painful but successful , even if it took 10 years !!
- 🌐 Too many false hopes with **Objectivity**
- 🌐 Missing support for **Reflexion**
- 🌐 Abuse of inheritance
- 🌐 Long learning phase to make modular shared libs
- 🌐 Abuse of « **new** » generating scattered structures
- 🌐 Missing concept of **ownership**





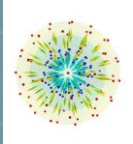
# No ownership in C++

- 🌐 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.



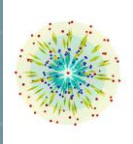
# Development Process

- **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**.



# Complexity

[Bock and Zoll in the 1972 CERN Courier – Central Computers in Bubble Chamber analysis]

*"The computer spends most of its time processing un-problematic events. The programmer, on the other hand, spends most of their time foreseeing – or discovering – possible difficulties and programming the computer to deal with them. The computer programs for bubble chamber experiments start with elegant and simple ideas, and end up complex and sophisticated."*

D.O.W  
2004

Automated  
meaningful test  
suites with static &  
dynamic test  
coverages are  
essential

- Computing 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!!

See Axel Naumann's poster about Electric Commander



# 17 years with ROOT

after  
PAW  
GEANT  
3

Objectivity  
era

LCG projects  
start  
PI, SEAL,  
POOL

Large systems evolve  
with time thanks to  
users and lessons  
learned

PAW-like  
ROOT

FNAL/RHIC  
Go ROOT

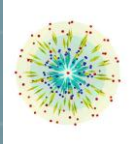
ROOT  
mainstream

1995

2000

2005

2010



# Input/Output: Major Steps

User written  
streamers  
filling TBuffer

member-wise streaming  
for STL collections<T\*>

streamers generated  
by rootcint

automatic streamers  
from dictionary  
with StreamerInfos  
in self-describing files

member-wise streaming  
for TClonesArray

TreeCache

parallel  
merge

1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010

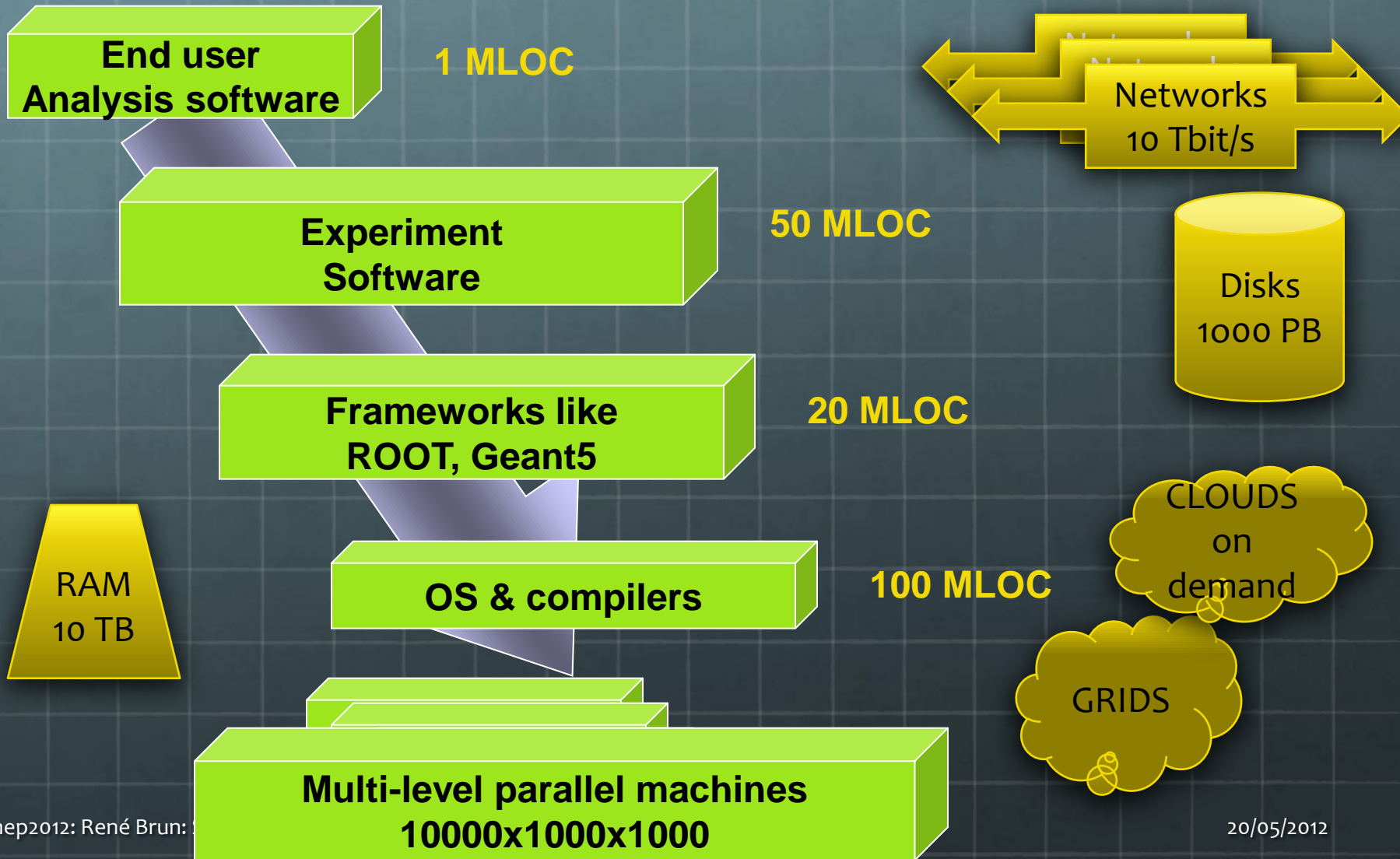


# Some messages for the future

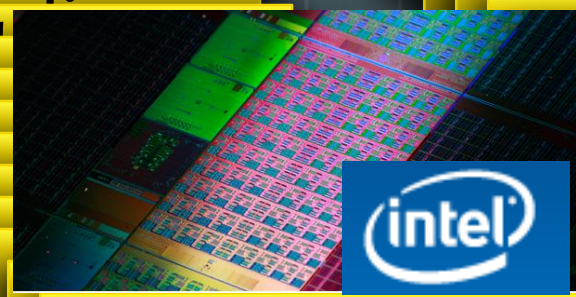
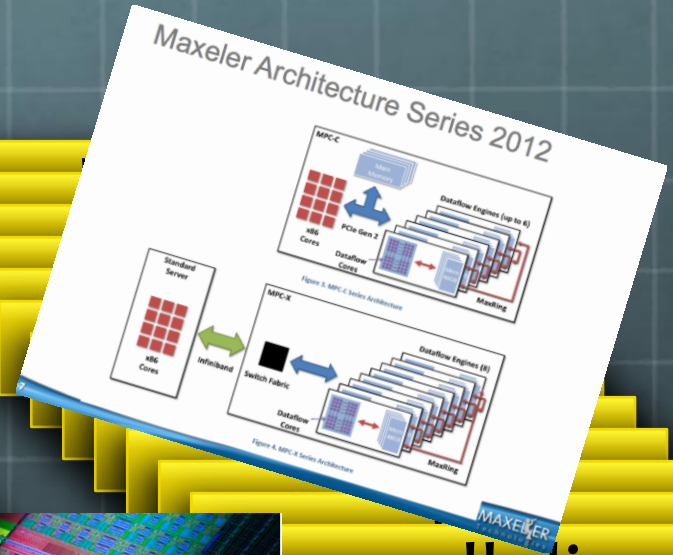
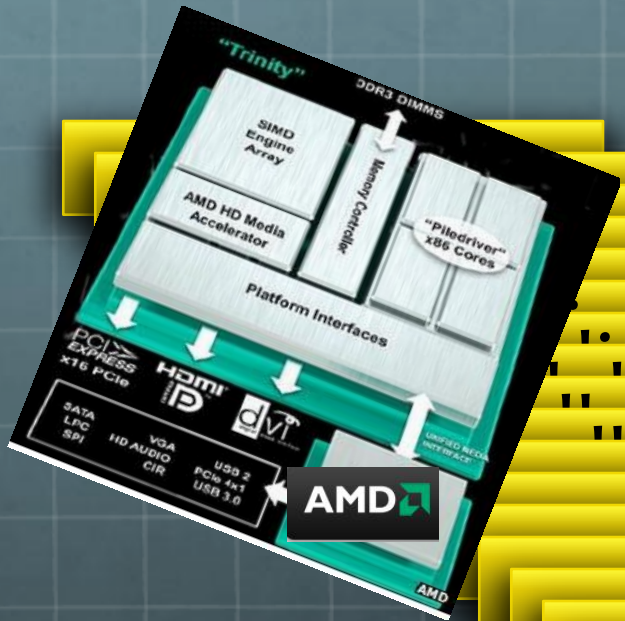




# Systems in 2030 ?



# Keyword: parallelism



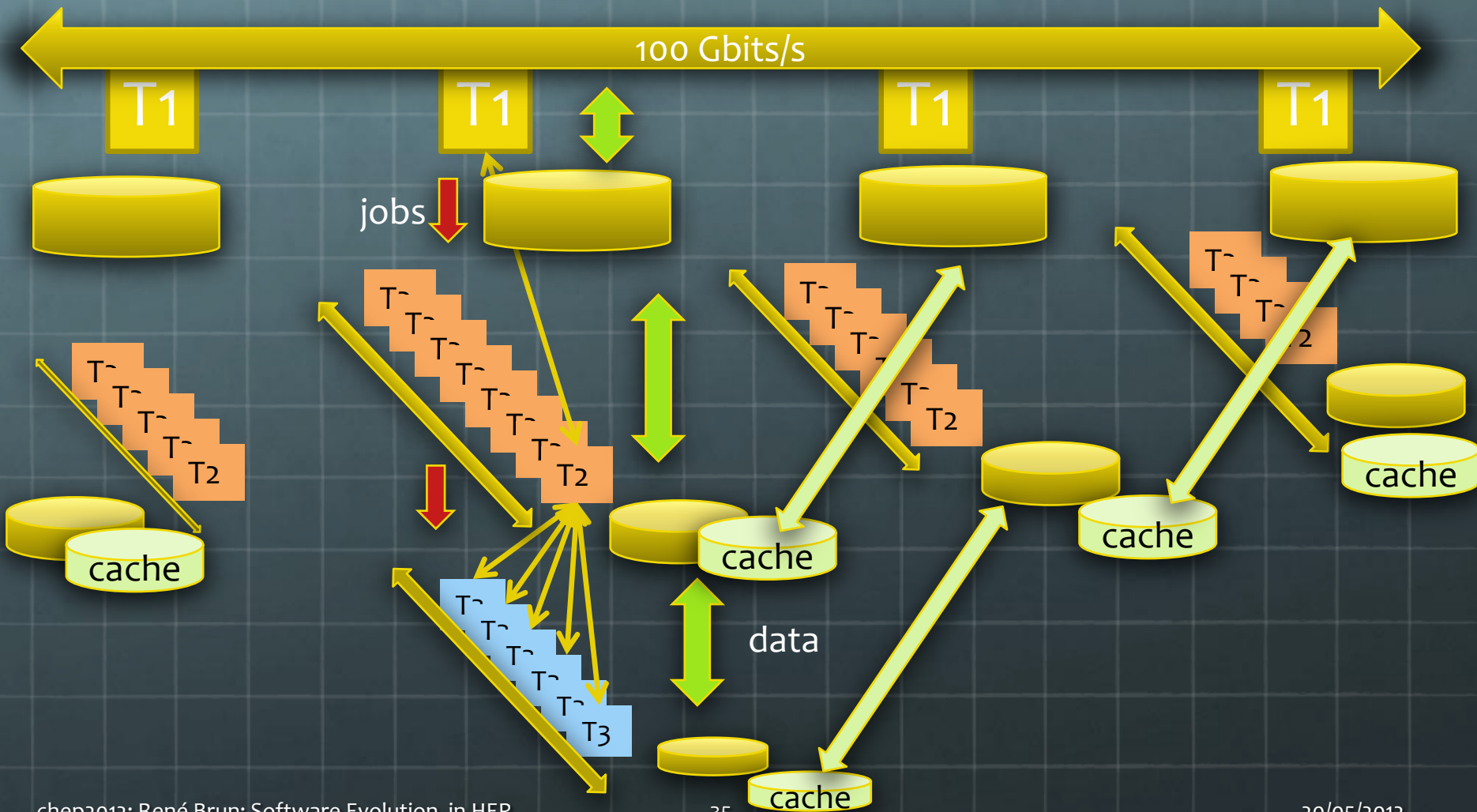
parallelism



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**



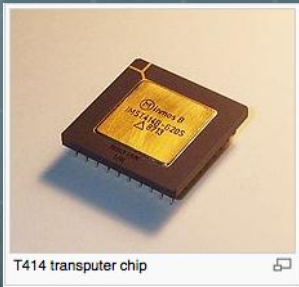
# Parallelism: **many failures**

cray



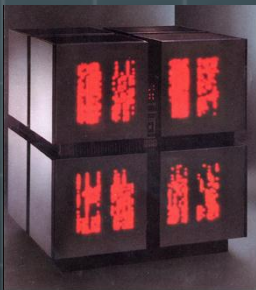
We failed in vectorizing codes like GEANT3 in 1985-1987 on CRAY, Cyber205, ETA10, IBM3090 because **our approach was wrong**

inmos

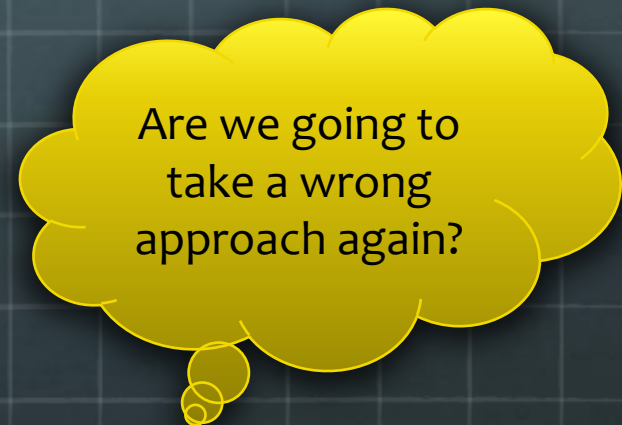


Some successful attempts in online systems in 1983

cm2



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





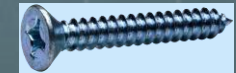
# Parallelism: **key points**



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.



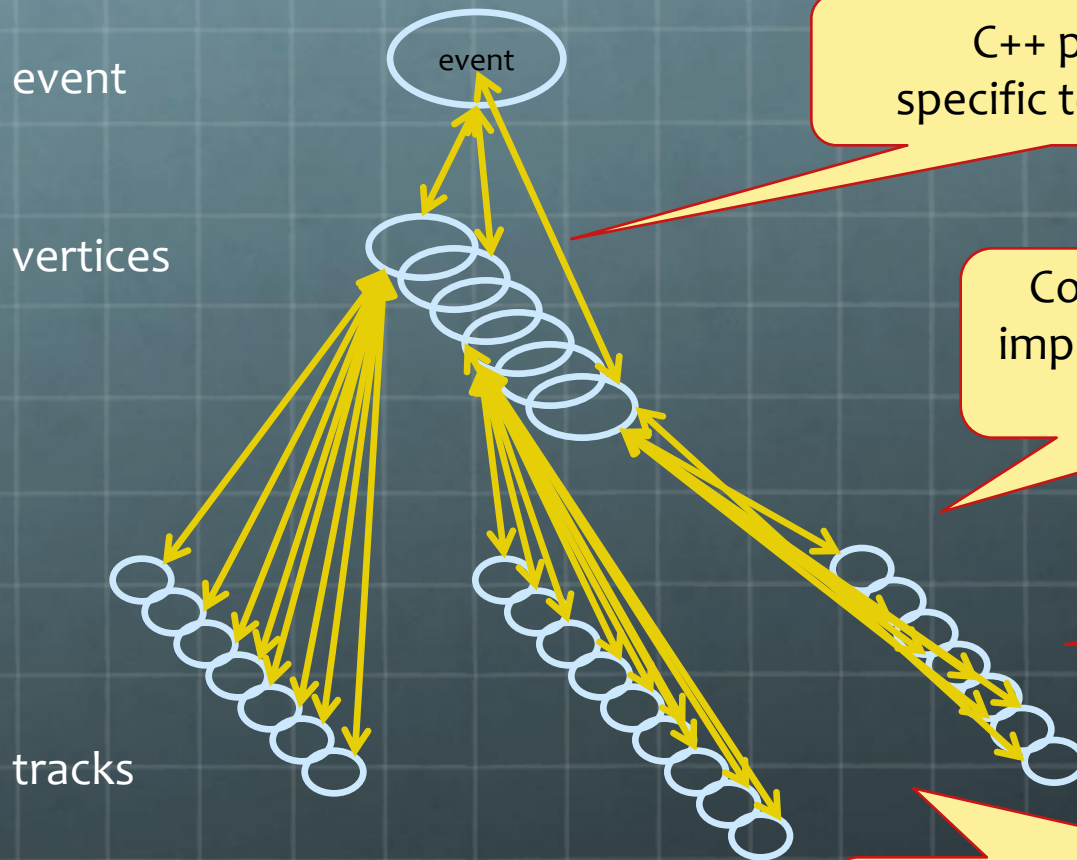
Exploit the **vector capabilities** but be careful with the new/delete/gather/scatter problem



Reorganize your code to **reduce tails**



# Data Structures & parallelism

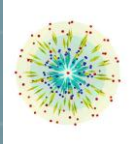


C++ pointers  
specific to a process

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



# Data Structures & Locality

sparse data structures defeat the system memory caches



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

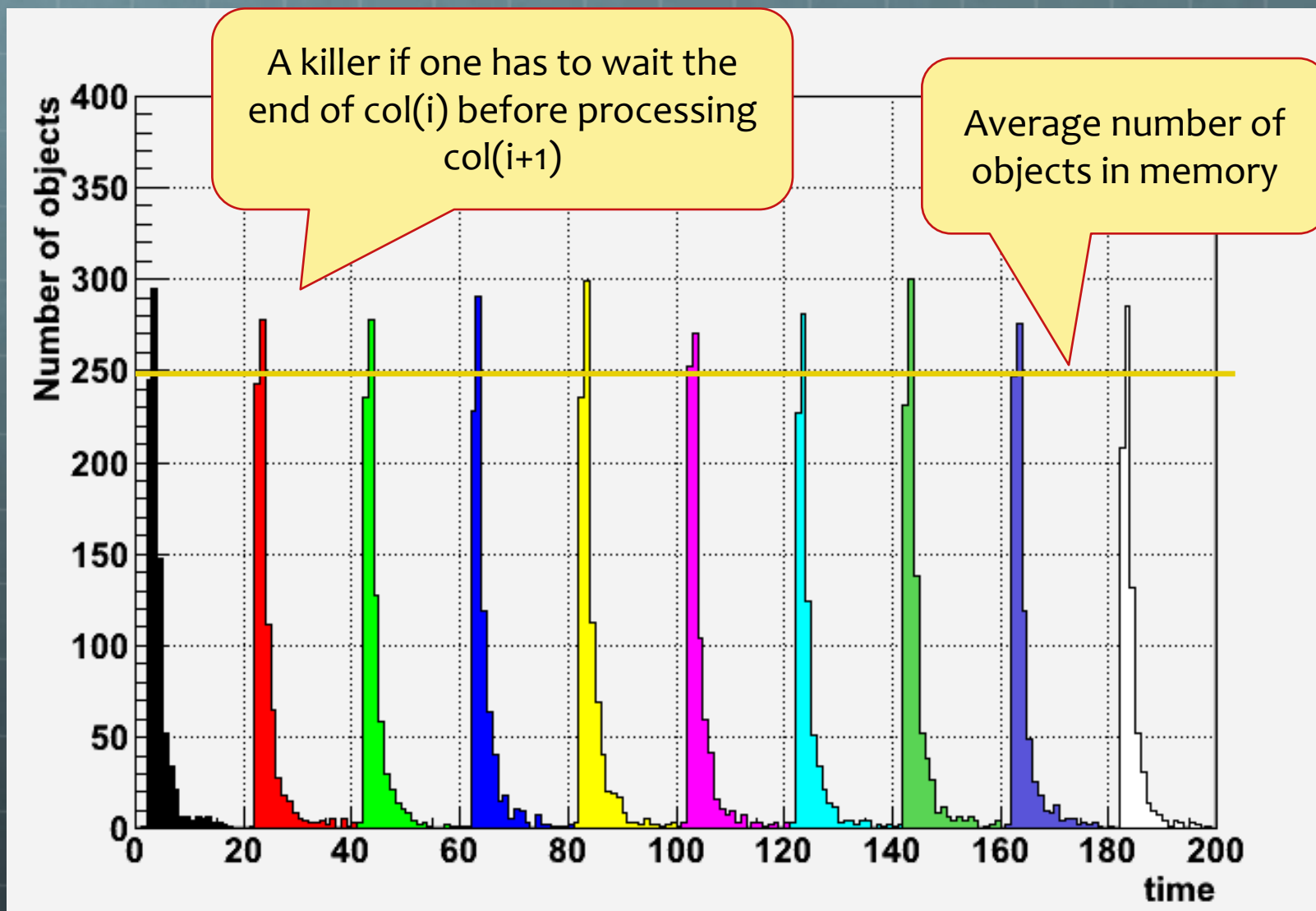
For example: group the cross-sections for all processes per material instead of all materials per process





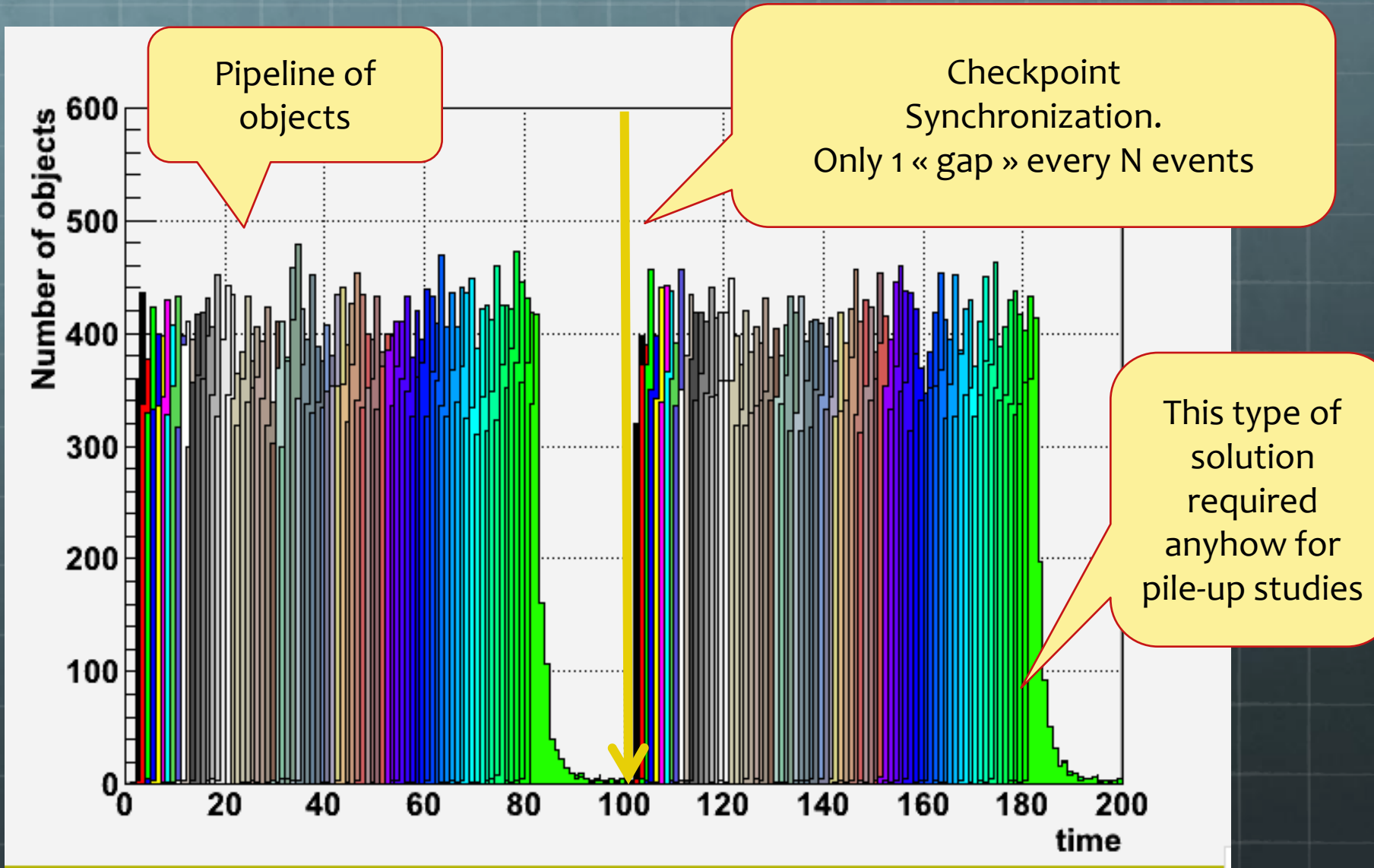


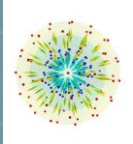
# Tails again





# A better solution

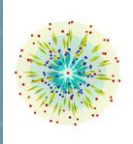




# Other requirements

- 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.

This was not obvious 15 years ago

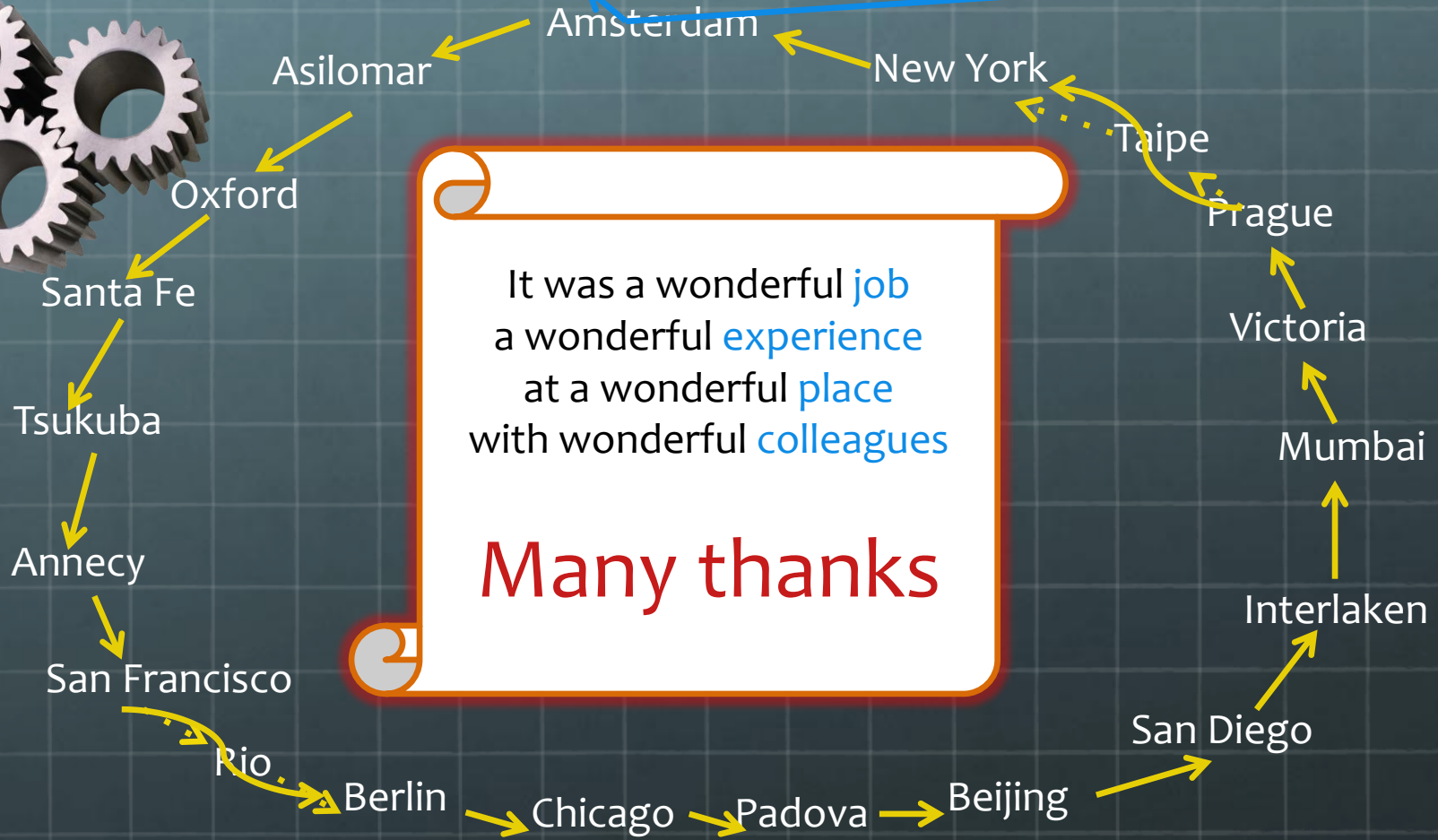
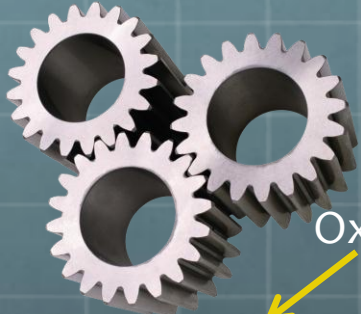
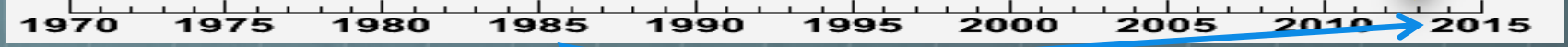
What is the best way to pass the message?

Fantastic opportunities for creative people



# Wheels are turning (for me and for CHEP)

New job  
New life



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

**Many thanks**