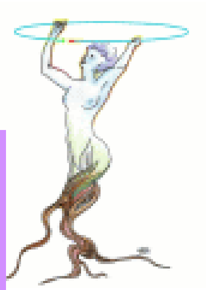


C++ memory management

Discussion with Bjarne Stroustrup

René Brun/CERN

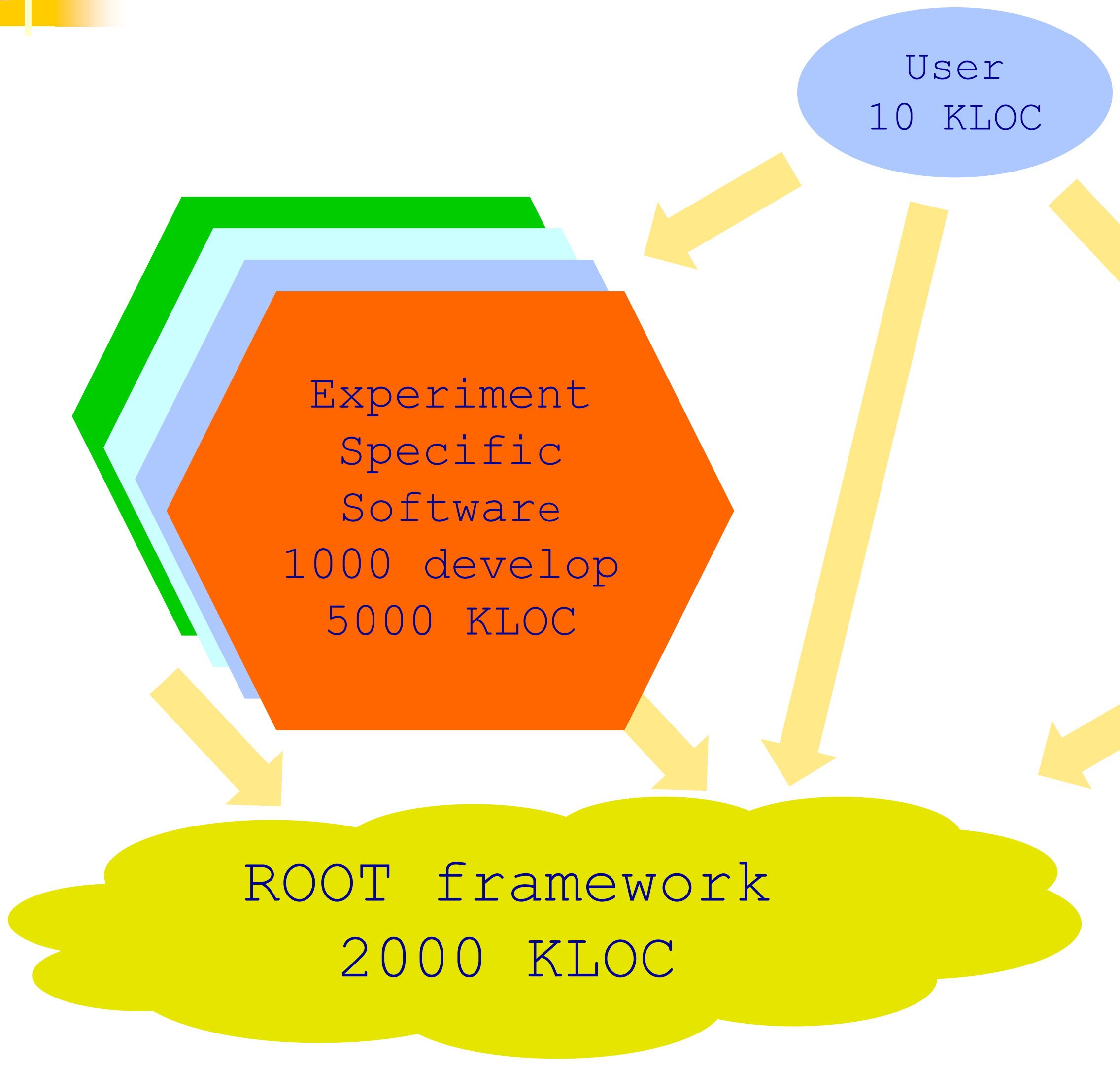
Some preliminary remarks

- In general everything is possible within C++ via templates or libraries like boost.
- This may be, however, at the cost of unreadable or not portable code.
- Memory management is and becoming more and more a KEY issue in very large applications. Better tools should be implemented in the basics of the language.

C++ in High Energy Physics

- About 10,000 physicists in HEP, 6,000 in 4 LHC experiments (Alice, Atlas, CMS, LHCb)
- We moved from F77 to C++ in 1995
- Atlas alone:
  - 2,000 physicists in 150 labs in the world
  - about 20,000 classes, 30,000 typedefs
  - about 400 shared libs
- ROOT alone:
  - 2500 classes, 100 shared libs, 10 developers
  - More than 20,000 users

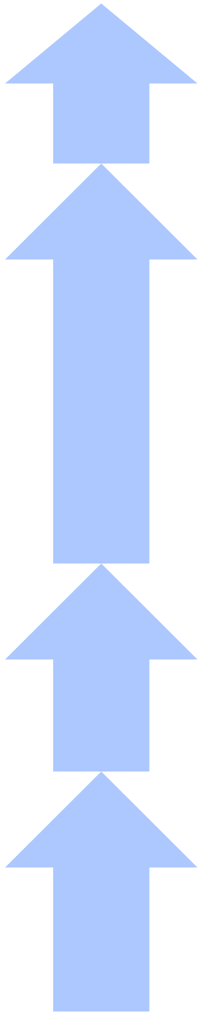
Layers in HEP software



Memory pools

- Because memory pools are not available in the language, most applications suffer from memory fragmentation or memory leaks.
- We implemented memory pools in 1980 in fortran packages like zbook or zebra.
- All “old” programmers found C++ very restrictive because pools and automatic garbage collection were not implemented.

Memory pools



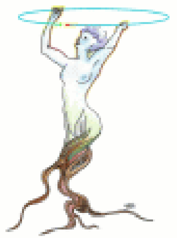
temp, eg I/O buffers

main event

geometry

histograms

Memory pools

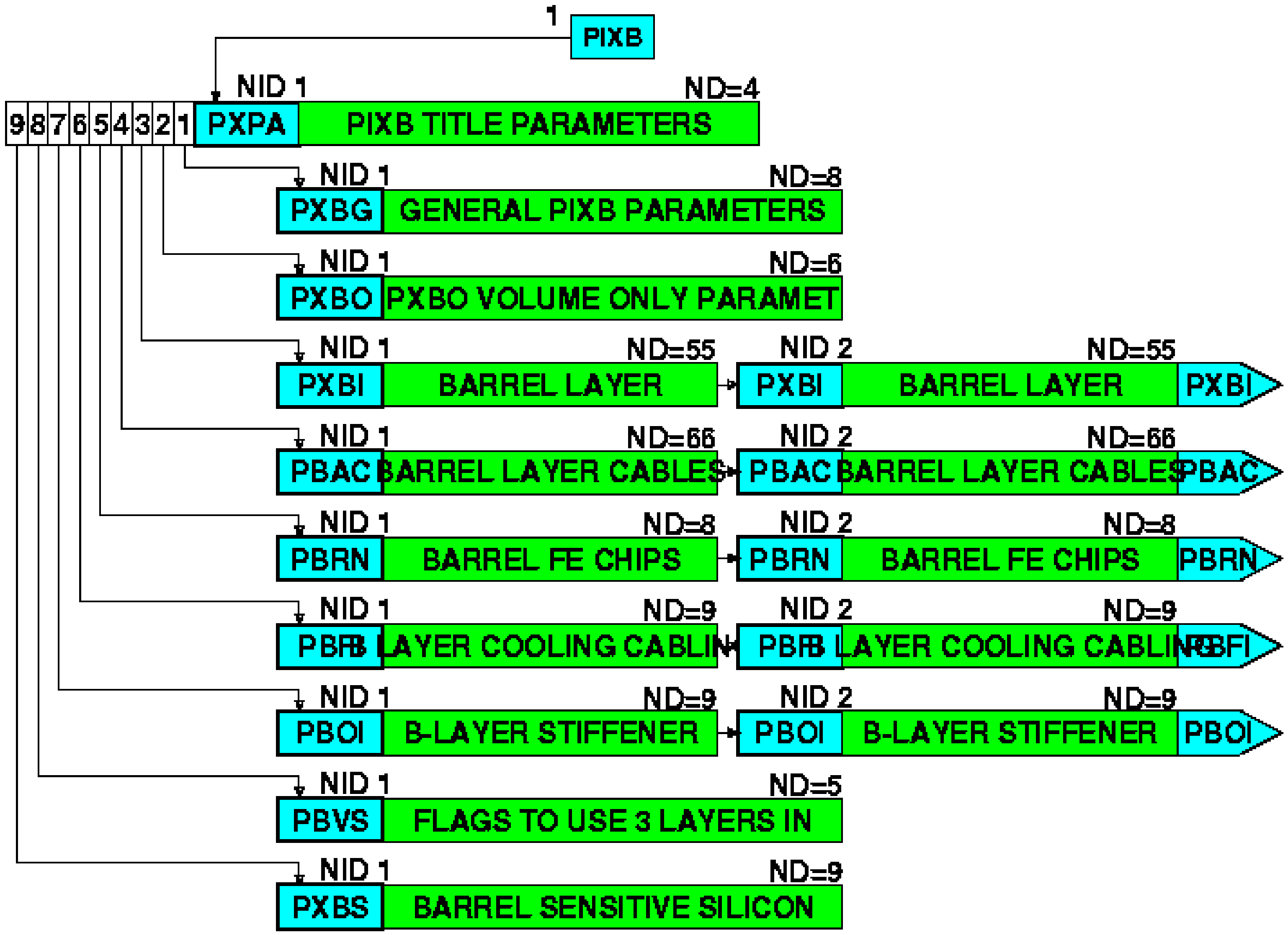


- pool = new pool(size);
- myObject = new(pool) myObject
- …
- pool->clear(); //objects in pool are deleted without calling their destructor
- pool->delete(); //object destructors called
- Ideally memory pools should be re-locatable, but this would imply "handles" like in MS C++

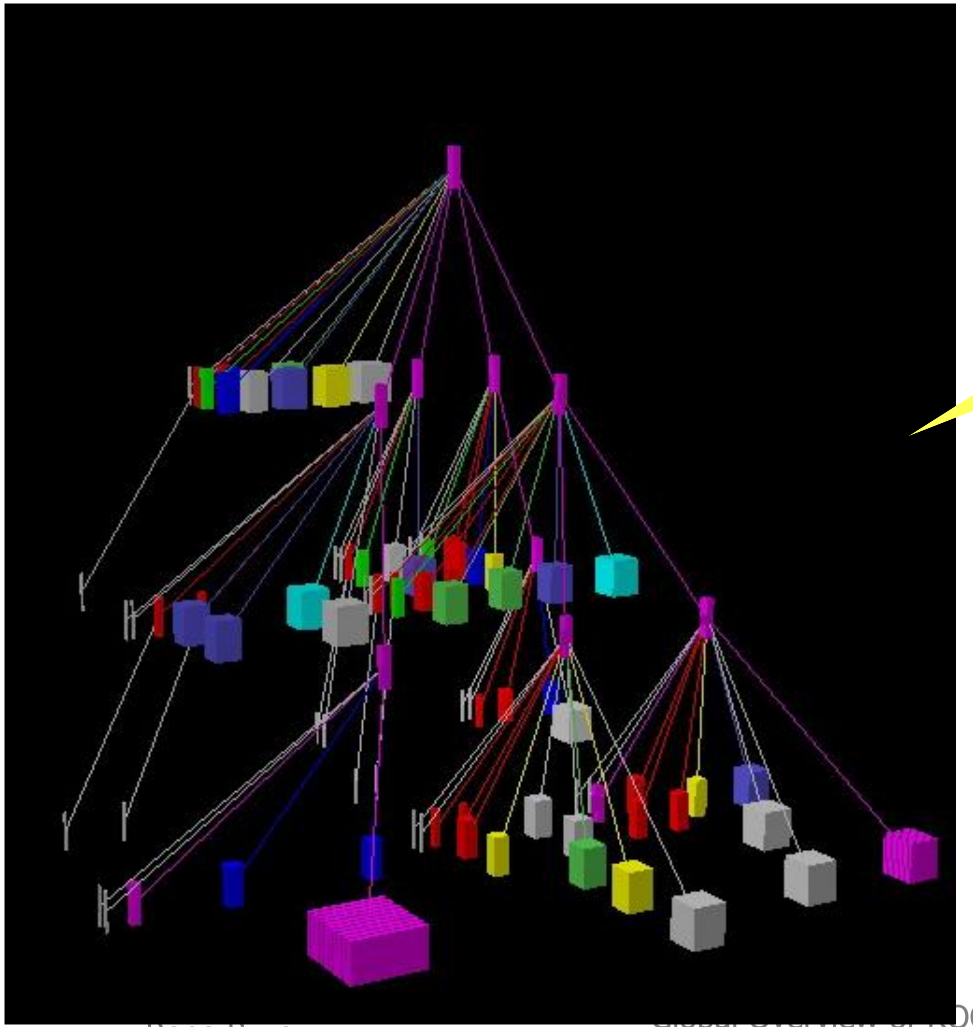
Object ownership

- C++ pointers are like C pointers
- Impossible to know if a class owns an object
- Most applications have implemented a smart reference pointer to resolve circularity issues, ownership problems when copying/deleting objects or when doing I/O.
- In all large applications we have implemented persistent reference pointers, a tricky issue for an efficient implementation.

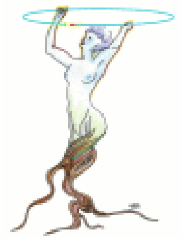
Atlas DZDOC


1
PIXB
NID 1
ND=4
9 8 7 6 5 4 3 2 1
PXPA
PIXB TITLE PARAMETERS
NID 1
ND=8
PXBG
GENERAL PIXB PARAMETERS
NID 1
ND=6
PXBO
PXBO VOLUME ONLY PARAMET
NID 1
ND=55
PXBI
BARREL LAYER
NID 2
ND=55
PXBI
BARREL LAYER
PXBI
NID 1
ND=66
PBAC
BARREL LAYER CABLES
NID 2
ND=66
PBAC
BARREL LAYER CABLES
PBAC
NID 1
ND=8
PBRN
BARREL FE CHIPS
NID 2
ND=8
PBRN
BARREL FE CHIPS
PBRN
NID 1
ND=9
PBFI
LAYER COOLING CABLIN
NID 2
ND=9
PBFI
LAYER COOLING CABLIN
PBFI
NID 1
ND=9
PBOI
B-LAYER STIFFENER
NID 2
ND=9
PBOI
B-LAYER STIFFENER
PBOI
NID 1
ND=5
PBVS
FLAGS TO USE 3 LAYERS IN
NID 1
ND=9
PXBS
BARREL SENSITIVE SILICON

Object viewers



Inspecting objects
requires RTTI
and the discovery
of objects ownership



Member-wise storage

- C++ objects are created object-wise. sizeof(object) is the consecutive number of bytes for the object.
- In case of STL collections (std::vector<T>) storing objects member-wise will take advantage of pipelines and parallelism.
- When doing I/O member-wise storage has proven to be more efficient.