ROOT6: The Quest For Performance

Danilo Piparo – CERN, PH-SFT
For the ROOT Team

Goal of this Contribution

- Illustrate strategies adopted to increase ROOT performance
- Review design choices and lessons learned

Successful collaboration of the ROOT team and the LHC experiments. **Without their contribution, ROOT6 would not be as good as it is now.**

ROOT6: A Big Change

Problem: ROOT5 interpreter Cint

- C parser, with some C++ capabilities
- Reflection, I/O: no support for new C++ standards, e.g. C++11
- Cracks in the infrastructure: e.g. support for gccxml on OSX

Solution: Replace Cint with Cling

- Cling: a C++ interpreter based on Clang/LLVM technology

A production quality compiler toolkit!

Side effect: a lot of work!

- **We believe the benefits outweigh the costs**

Investments are needed for future sustainability

Challenges Involved

Push forward software technology

- Cling: first of its kind (JIT of C++!)
- Re-write of entire ROOT Core components
- Including layer between ROOT and its interpreter

Compared with CINT, optimised during 20y!

Existing features to support, rich set of new ones

- Many users: $O(10^4)$ – Backward compatibility guarantees
- Experiment setups: multi-MLOC software systems

A quest but an opportunity

- Such a radical change rarely happens in core software

Improve strategies to evolve our sw, e.g. with agile techniques

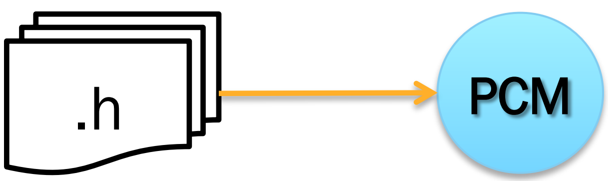
Interlude: Clang, the AST and ROOT

C++ entities in Clang: Abstract Syntax Tree (AST)

- Classes, functions, templates, statements …
- Exists in memory and can be persisted on disk in two forms

1) **P**re-**C**ompiled **H**eader: can load only one, file granularity
2) **P**re-**C**ompiled **M**odules: can load many, AST node granularity

- Both queried lazily by the compiler
- Dictionaries: a thin layer around portions of AST

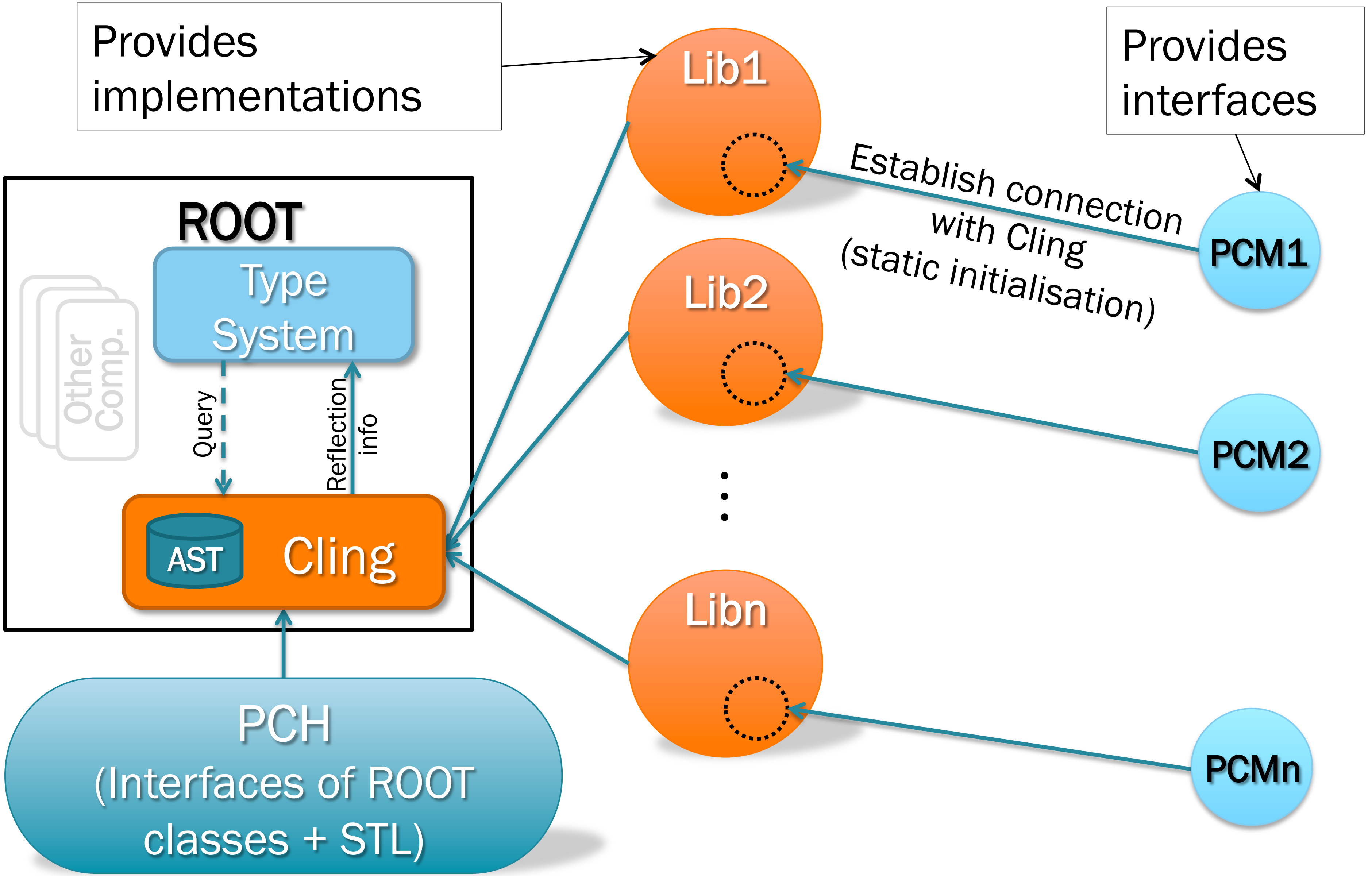


PCM: Bleeding edge technology during LS1

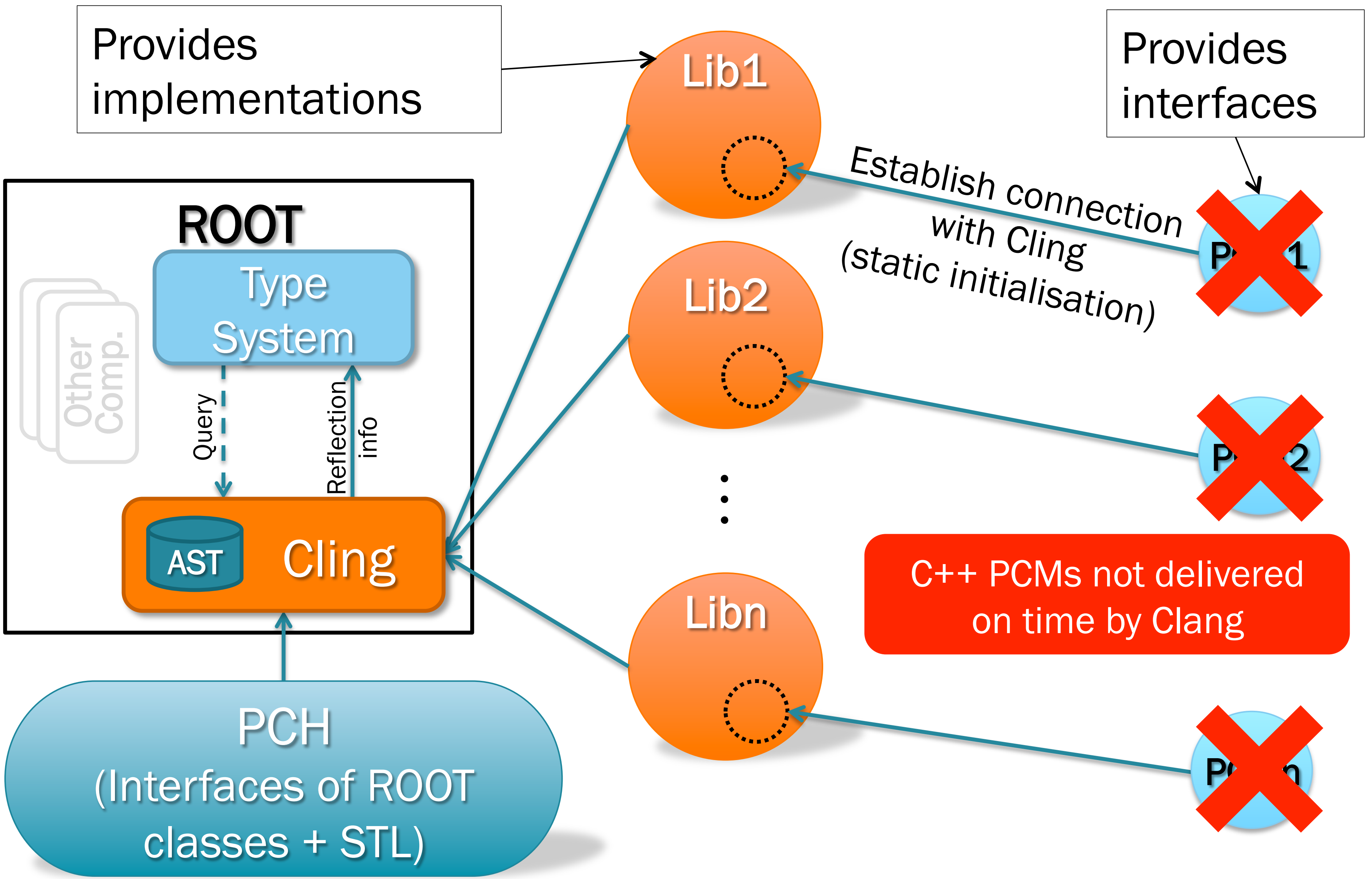
Original ROOT6 design: AST source of information for

- Reflection and I/O
- Interactive function calls

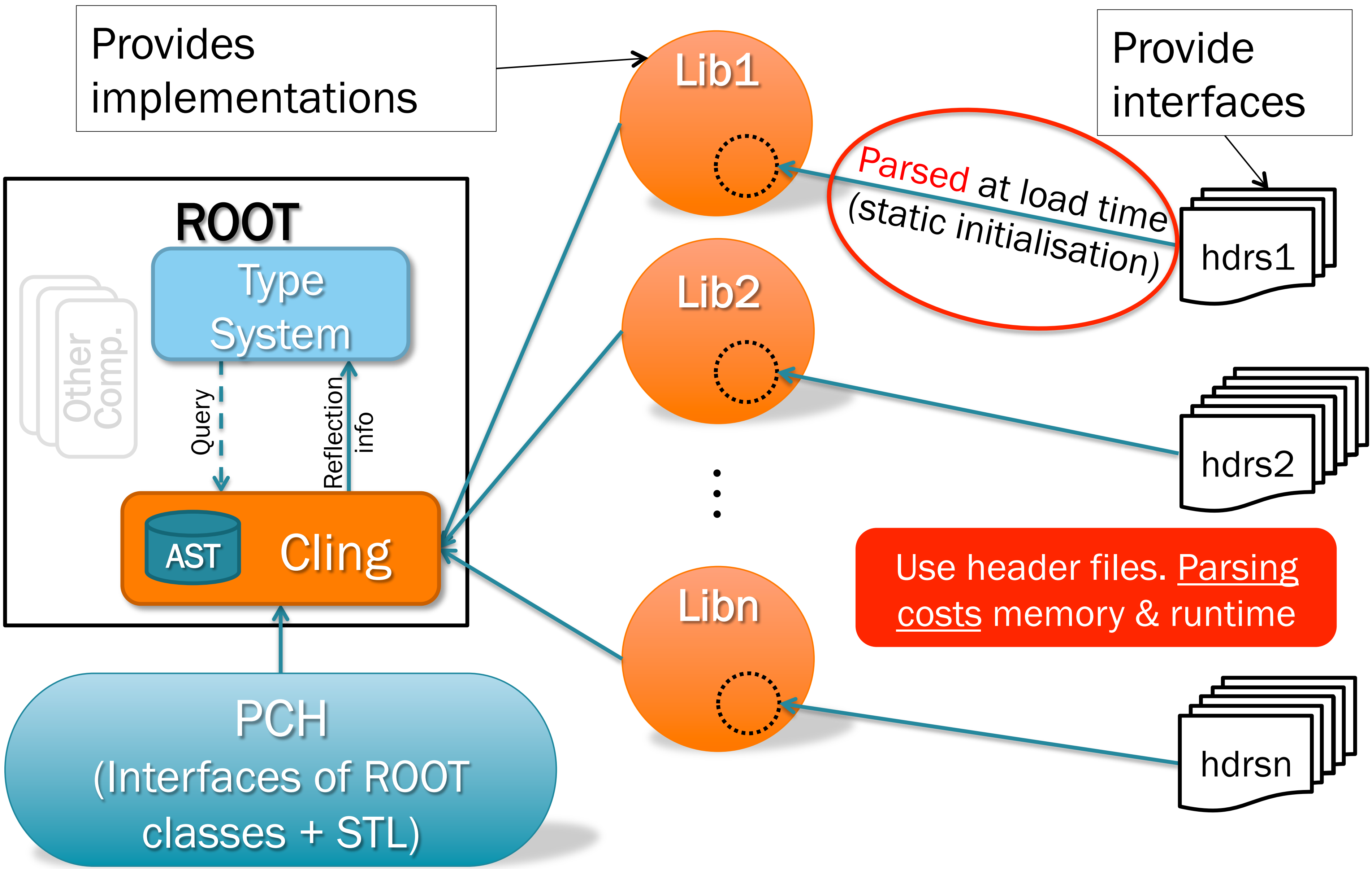
Interlude: ROOT, Clang and the AST



Interlude: ROOT, Clang and the AST

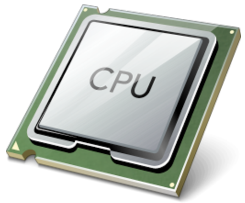


The Memory Excess



Performance: Change of Plans Demanded

- Issue solved already in Autumn 2014
  - 6.02, 6.04 series not affected!



- Consequences of absent PCMs at the time:
  - Good for analysis and single users
  - Too much memory when integrated with LHC experiments' software stacks: ~1 GB RSS extra ☹
  - Runtime penalty associated to these allocations

Adapt quickly to changing reality

The Change

Improve memory consumption: Reduce parsing

1) I/O operations

- I/O info for selected classes in "ROOT-PCMs" (ROOT files)
- Optimise file format for those
- Information forwarded directly to ROOT type system

2) Interactive usage

- Parse "on demand" (or "Autoparsing")

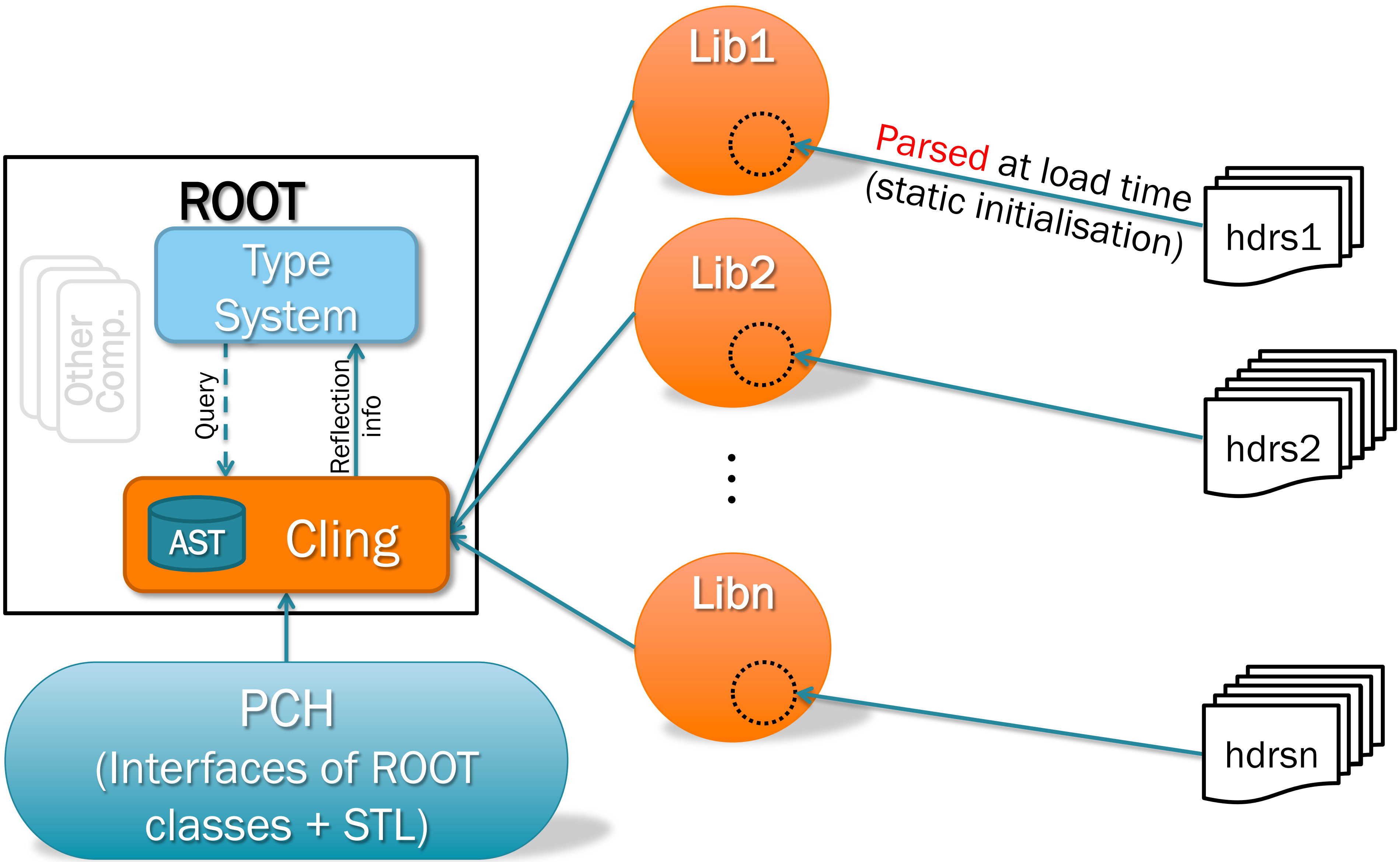
Trigger parsing of headers related to library only when needed

a. To call functions and methods

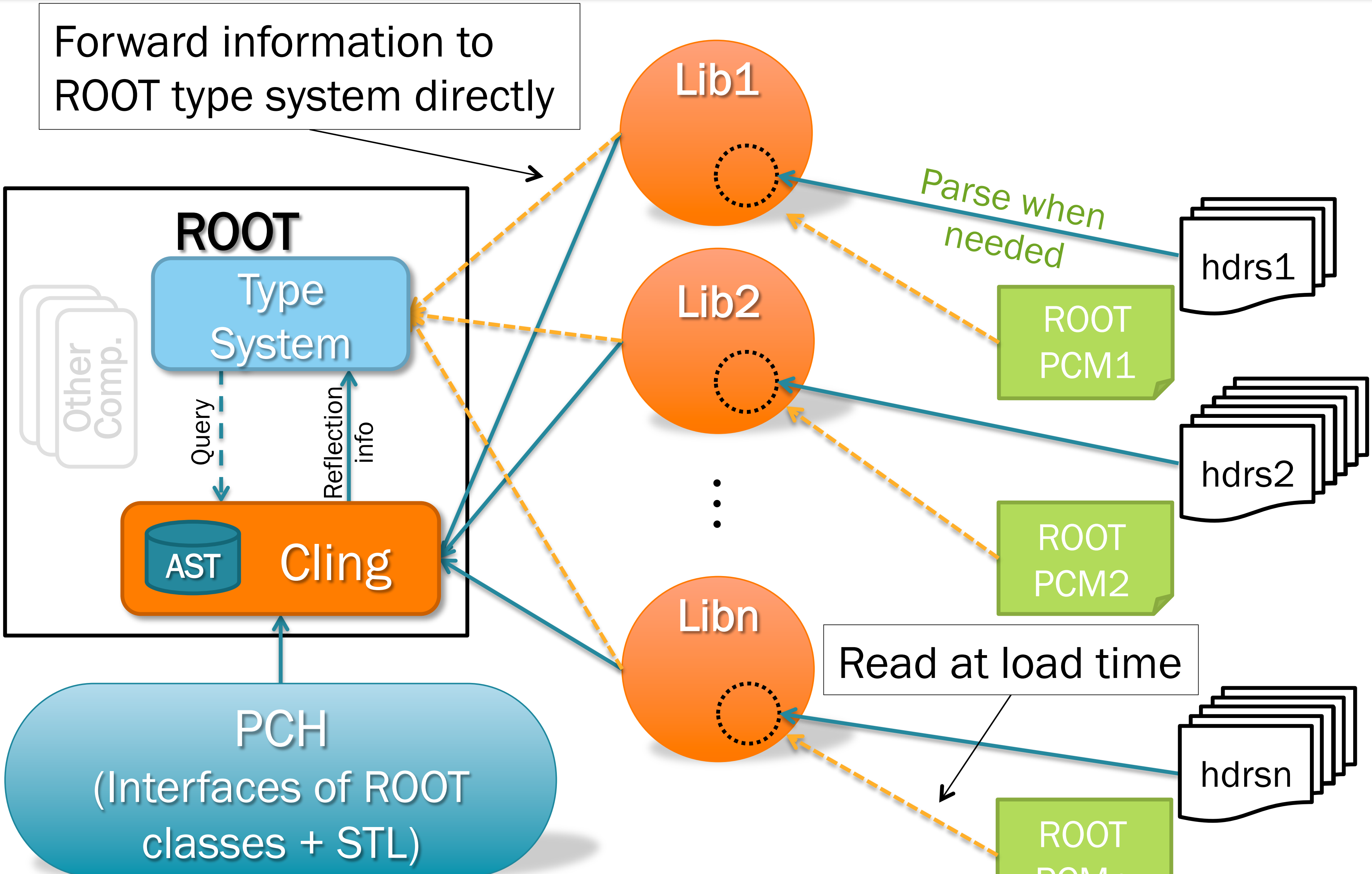
b. To get I/O info when not provided by ROOT-PCMS

Iterative, incremental, evolutionary

A Change In the Design



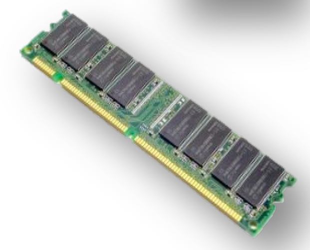
A Change In the Design

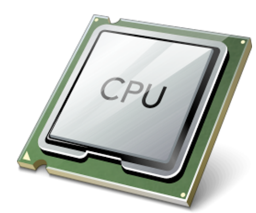


Where Are We Now: CMS Example

Memory

- pp→ttbar events @ 13 TeV (event loop):
  - Generation & Simulation: **-6% RSS** wrt ROOT5
    - → Yes, better than ROOT5 ☺
  - Reconstruction: **+4% MB RSS** wrt ROOT5
- RSS variations: depend on amount of interpreted functions
  - E.g. cuts specified in job configuration





Runtime: ~Identical in the event loop

Also thanks to experiments' flexibility and willingness to make this happen – thank you!

Ensure Correctness: Testing

Why? Profiling & improvements: meaningless w/o correctness!

- Significant expansion of ROOT test suite
  - Target test-driven development
  - All plugins and externals tested (e.g. Davix, xRootd)
- In addition: increase of test platforms (~8 -> ~17)
  - And planning to add more, also non-x86_64*

*

See:

493: Future Computing Platforms for Science in a Power Constrained Era

500: Building a Tier-3 Based on ARMv8 64-bit Server-on-Chip for the WLCG

Ensure Correctness: Testing

Nightly

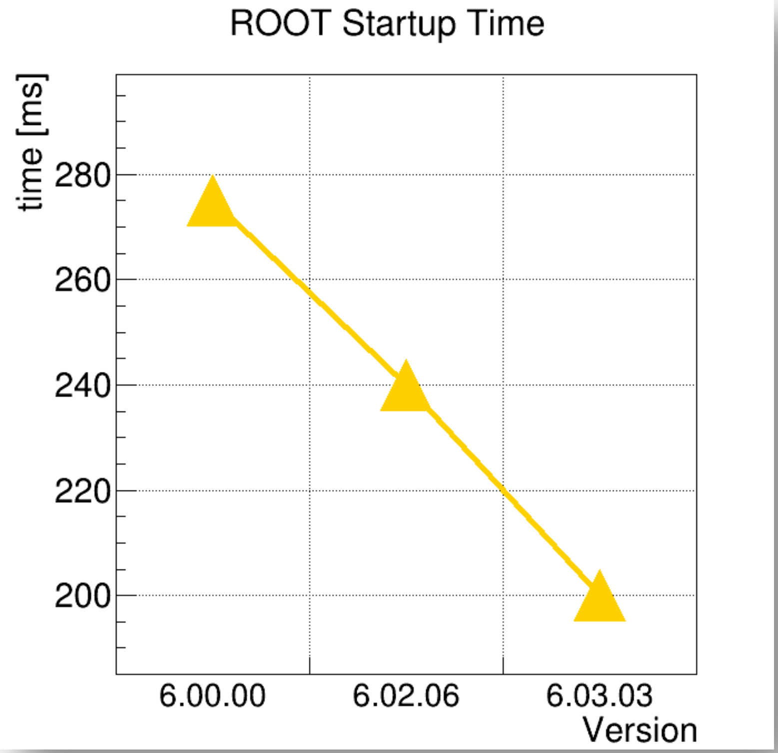
Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
macitois13.cern.ch	v6-02-00-patches-x86_64-mac1010-clang35-opt	0	0	0	0	1	0	0	1035	10 hours ago
lcgapp-slc6-x86-64-22.cern.ch	v6-02-00-patches-x86_64-slc6-gcc49-opt	0	0	0	0	6	0	0	1049	10 hours ago
lcgapp-slc6-x86-64-9.cern.ch	v6-02-00-patches-x86_64-slc6-gcc48-opt	0	0	0	0	6	0	0	1049	10 hours ago
macitois13.cern.ch	master-x86_64-mac1010-clang35-opt	7	0	0	0	9^{+1}	0	0	1088	9 hours ago
macitois17.cern.ch	master-x86_64-mac108-clang34-opt-classic	7	0	0	0	2	0	0	224	9 hours ago
ec-ubuntu-14-04-x86-64-1	master-x86_64-ubuntu14-gcc48-opt	7	0	0	0	4^{+3}	0	2^{+1}	1062_{-1}	9 hours ago
macitois18.cern.ch	master-x86_64-mac108-clang34-opt	7	0	0	0	9^{+1}	0	0	1088	9 hours ago
ec-fedora20-x86-64-3	master-x86_64-fedora20-gcc48-opt	7	0	0	0	4^{+3}	0	2	1062	9 hours ago
macitois14.cern.ch	master-x86_64-mac1010-clang35-opt-classic	7	0	0	0	2	0	0	224	9 hours ago
lcgapp-slc6-x86-64-22.cern.ch	master-x86_64-slc6-clang35-dbg	7	0	0	0	1	0	2	1053	9 hours ago
lcgapp-slc6-x86-64-9.cern.ch	master-x86_64-slc6-clang35-opt	7	0	0	0	1^{+1}_{-6}	0	1^{+1}_{-4}	1054^{+16}_{-2}	9 hours ago
macphsft14.cern.ch	master-x86_64-mac109-clang35-opt	7	0	0	0	9^{+1}	0	1^{+1}	1087_{-1}	9 hours ago
macphsft15.cern.ch	master-x86_64-mac109-clang35-opt-classic	7	0	0	0	2	0	0	224	9 hours ago
lcgapp-slc6-x86-64-6.cern.ch	master-x86_64-slc6-gcc48-dbg	7	0	0	0	9^{+3}	0	4^{+1}_{-1}	1103^{+1}_{-1}	9 hours ago
lcgapp-cc7-x86-64-10.cern.ch	master-x86_64-cc7-gcc48-opt	7	0	0	0	4^{+3}	0	6^{+1}	1058_{-1}	9 hours ago
lcgapp-cc7-x86-64-6.cern.ch	master-x86_64-cc7-gcc48-opt-classic	7	0	0	0	2	0	0	224	9 hours ago
macitois17.cern.ch	v5-34-00-patches-x86_64-mac108-clang34-opt-classic	0	0	0	0	0	0	0	220	9 hours ago
macitois11.cern.ch	v5-34-00-patches-x86_64-mac109-clang35-opt	0	0	0	0	0	0	0	527	9 hours ago
macitois14.cern.ch	v5-34-00-patches-x86_64-mac1010-clang35-opt	0	0	0	0	0	0	0	527	9 hours ago
pcphsft70vm.cern.ch	v5-34-00-patches-x86-winxp-vc10-opt	0	0	0	0	0	0	0	440	9 hours ago
macphsft15.cern.ch	v5-34-00-patches-x86_64-mac109-clang35-opt-classic	0	0	0	0	0	0	0	220	9 hours ago
ec-ubuntu-12-04-x86-64-1	v5-34-00-patches-x86_64-ubuntu12-gcc46-opt	0	0	0	0	0	0	0	503	9 hours ago
ec-win7-x64-01.cern.ch	v5-34-00-patches-x86-winxp-vc12-opt	0	0	0	0	50^{+13}_{-13}	0	0	440	8 hours ago
macitois17.cern.ch	v5-34-00-patches-x86_64-mac108-clang34-opt	0	0	0	0	0	0	0	527	8 hours ago
ec-ubuntu-14-04-x86-64-1	master-x86_64-ubuntu14-gcc48-opt-classic	7	0	0	0	2	0	0	224	8 hours ago
macitois13.cern.ch	v5-34-00-patches-x86_64-mac1010-clang35-opt-classic	0	0	0	0	0	0	0	220	8 hours ago
lcgapp-slc6-x86-64-22.cern.ch	master-x86_64-slc6-clang36-dbg	7	0	0	0	1^{+1}	0	2	1053	8 hours ago
ec-ubuntu-12-04-x86-64-1	v5-34-00-patches-x86_64-ubuntu12-gcc46-opt-classic	0	0	0	0	0	0	0	220	8 hours ago
ec-ubuntu-14-04-x86-64-1	v5-34-00-patches-x86_64-ubuntu14-gcc48-opt-classic	0	0	0	0	0	0	0	220	8 hours ago

Public link to status of tests:
cdash.cern.ch/index.php?project=ROOT

Impossible to optimise w/o continuous & automatised correctness checks

Start up Time

- Very first feature seen by the user
  - Baseline: ROOT5, ~100 ms (Python 2.7 ~20 ms)



Solution:

- Leverage PCH to store I/O information of ROOT most used classes (Hist, RooFit, …)
- Optimise data structures and algorithms holding/manipulating autoloading info: e.g. use STL!
- Optimise reading of ROOTmap files

Strive for technical excellence in all corners

The Profiling Toolbox

IgProf

- Both for memory and runtime studies
- "Make diffs" of counters' snapshots, e.g.:
  - Given a symbol: Event-by-event differences in memory

Valgrind family

- *Callgrind*: very short runs (e.g. startup times)
- *Massif*: complement IgProf information and information display

Kernel Data Structures

- "Poor Man's Solution" `TSystem::GetProcInfo`
  - e.g.: memory before/after method invocation
  - Use as "tracing bullet"

poormansprofiler.org

Crucial to choose the right ones

Conclusion

ROOT6: many new features, backward compatible

- We now look towards an exciting future!

Lessons learned:

- Agile principles: asset when betting on cutting edge sw technologies
- Close collaboration with "clients": clear benefit for big sw projects
- Ruthless, automated & ubiquitous testing: requirement of ambitious performance improvement campaigns
- "Right" mix of profiling tools can make the difference

- LHC, SuperKEKB, Intensity Frontier: Challenging scenarios!
  → The quest will continue
  – Leverage even more STL in the ROOT Core
  – More vectorisation (and het. platforms): Tree analysis and Math
  – Better integration with profilers
  – Exploit many cores architectures even more

Backup

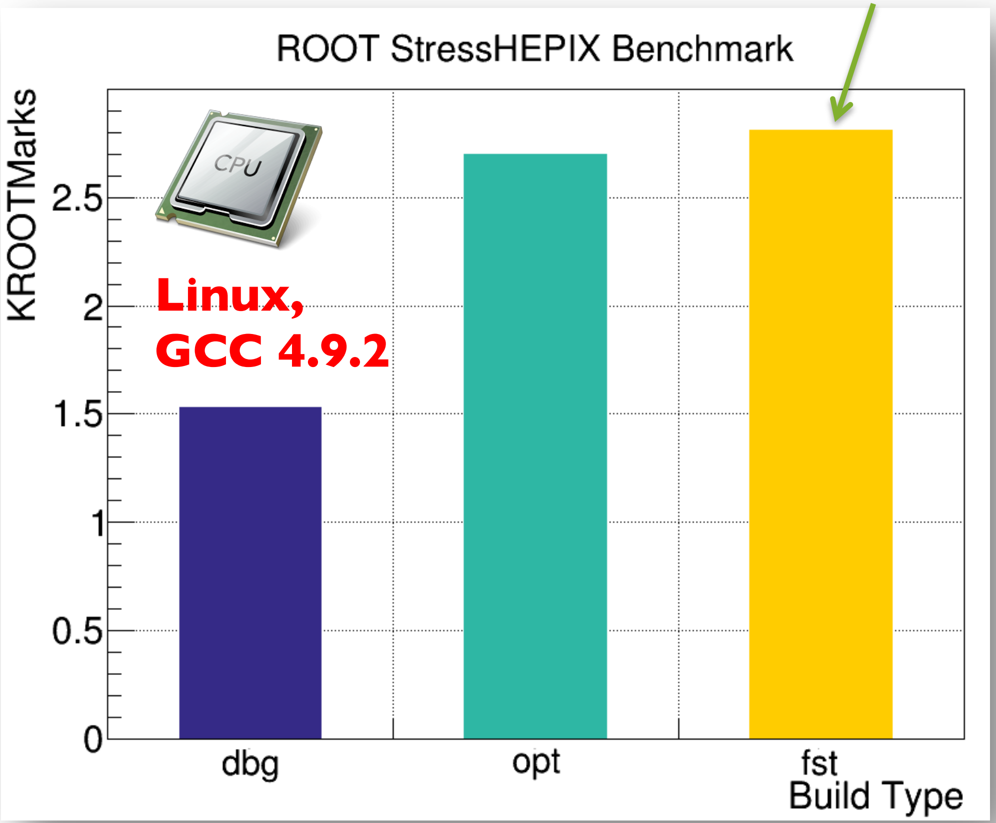
Leveraging Modern Compilers

ROOT6: written in C++11. Need recent compiler (e.g. GCC ≥ 4.8)

- Idea: leverage compilers' optimisations ("-Ofast")
  - Optimise FP treatment (e.g. operands re-ordering)
  - More inlining
- Improve routines fragile wrt changes in FP behaviour
- Optional, not the default

"Technical" (non algorithmic) optimisations **do** matter.

+4% wrt opt



Enabled with -D CMAKE_BUILD_TYPE=Optimized