# DIANA-HEP Final Presentation
# Runtime C++ modules

Yuka Takahashi - Princeton University, CERN

# Two summary slides

# Goal of this project

Optimize the performance of experiments by using C++ Modules technology in ROOT

# Achievement of this year

- Release of C++ Modules in ROOT 6.16
- CMSSW infrastructure working with ROOT Runtime C++ Modules

# Agenda

1. Motivation of C++ Modules
2. Implementation Details
3. 2018 Roadmap
4. Current Status
5. Performance Results
6. Future Roadmap

# Motivation of C++ Modules

# Motivation of C++ Modules

C++ Modules technology:

- Cache parsed header file information and avoid runtime header parsing

# Motivation of C++ Modules

```
#include <vector>
```

# Motivation of C++ Modules

```
#include <vector>
```

| Textual Include | PCH | Modules |
| --- | --- | --- |
| ✗ Expensive<br>✗ Fragile | ✗ Inseparable | ✓ |

# Motivation of C++ Modules

```
#include "TVirtualPad.h"
#include <vector>
#include <set>

int main() {
…
```

original code

Preprocess

Textual Include

```
                                        ……
                                        ……
# 286 "/usr/include/c++/v1/vector" 
namespace std { inline namespace __
template <bool> class __vector_base
{
   __attribute__
((__visibility__("hidden"),
__always_inline__)) __vector_base_c
{}
                                        ……
# 394 "/usr/include/c++/v1/set" 3
namespace std {inline namespace __1
template <…> class set {
public:
    typedef _Key key_type;
                                        ……

int main {
                                        ……
```

TVirtualPad.h

mmon

vector

on()

set

one big file!

.o

Compile

Parse

# Motivation of C++ Modules

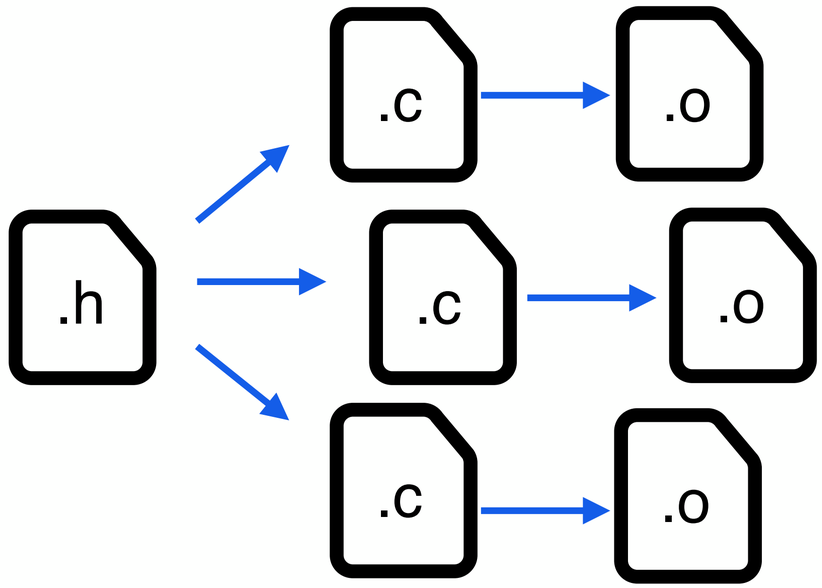
## Textual Include

1. Expensive

   Reparse the same header

2. Fragile

   Name collisions




Rcpp library

```
#define PI 3.14
…
```

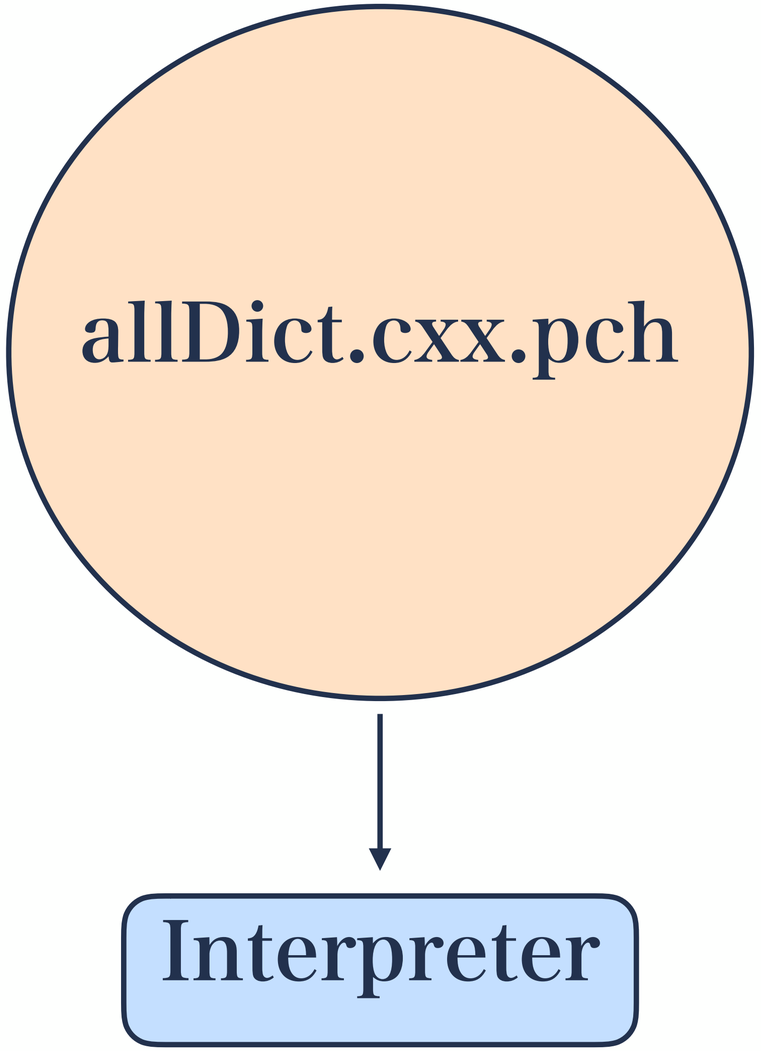
Users' code

```
#include <header.h>
                …
double PI = 3.14;
// => double 3.14 = 3.14;
```

# Motivation of C++ Modules

PCH (Pre Compiled Header)

1. Storing pre compiled header information (same as modules)
2. Stored in one big file

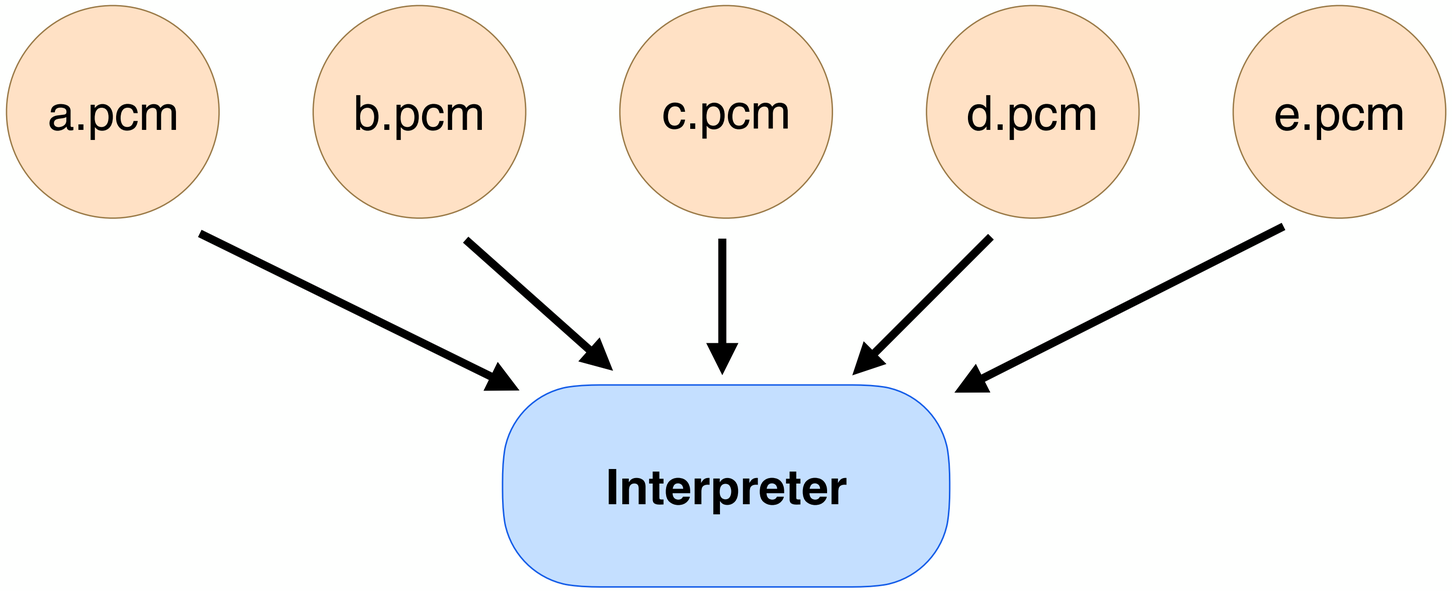


# Motivation of C++ Modules

## Modules

- Pre compiled **PCM files** contain header information
- PCMs are **separated**

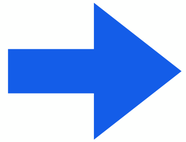
Each PCM file (a.pcm) corresponds to a library (liba.so)



# Motivation of C++ Modules

## Modules

- Pre compiled **PCM files** contain header information
- PCMs are **separated**



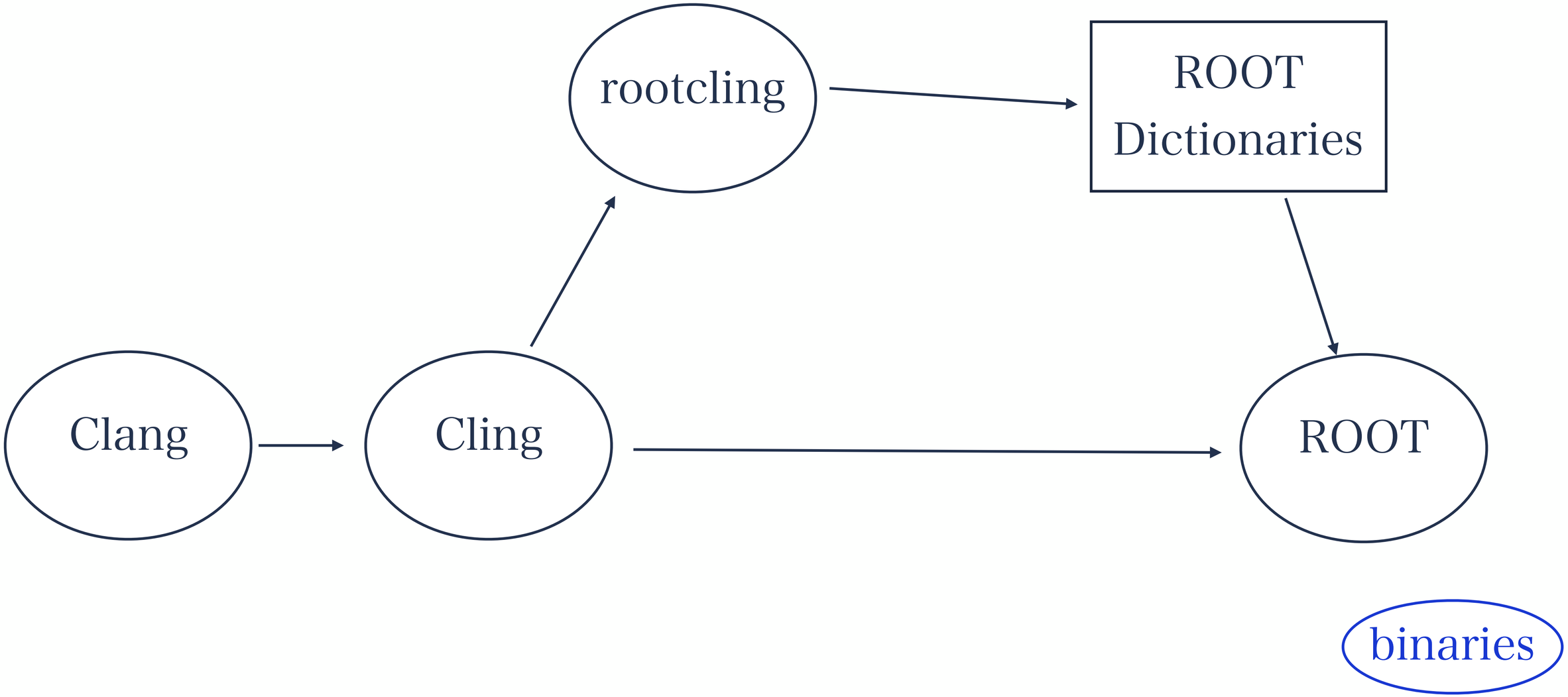
✓ Compile-time scalability

✓ Fragility

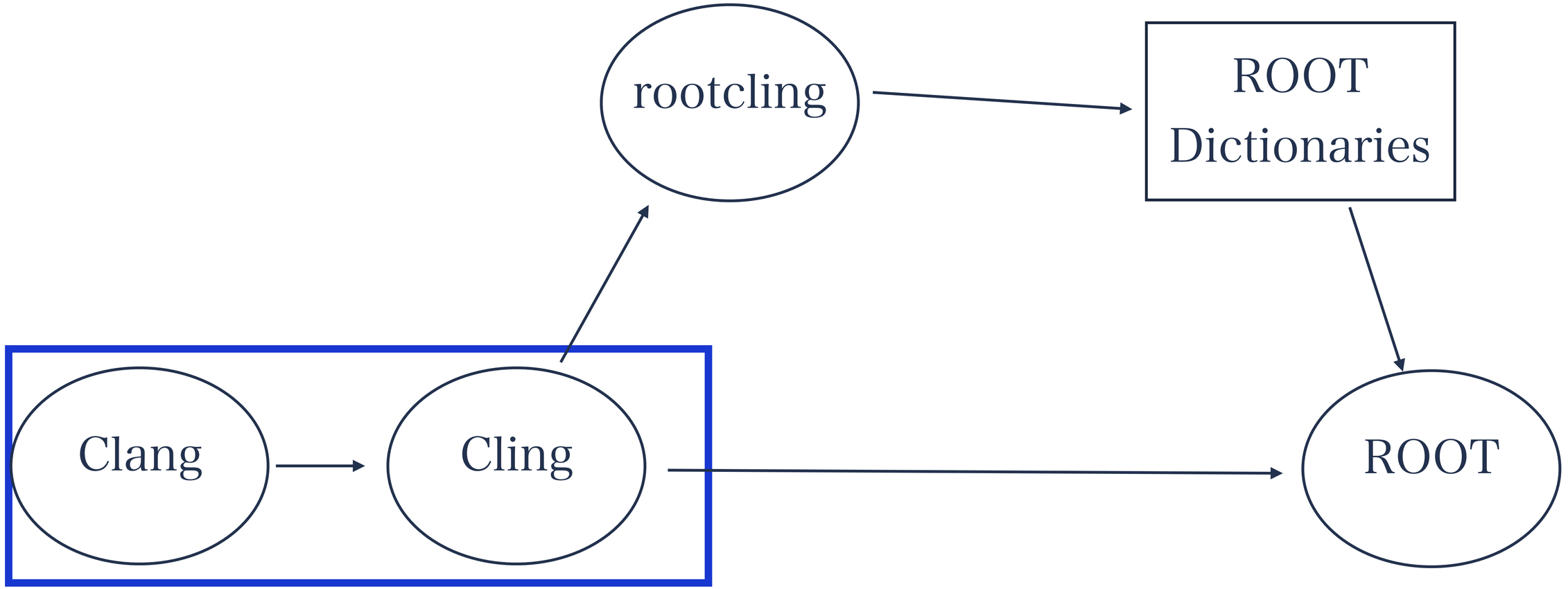
✓ Separable

# Implementation Details

# Implementation Details in ROOT



# Implementation Details in ROOT

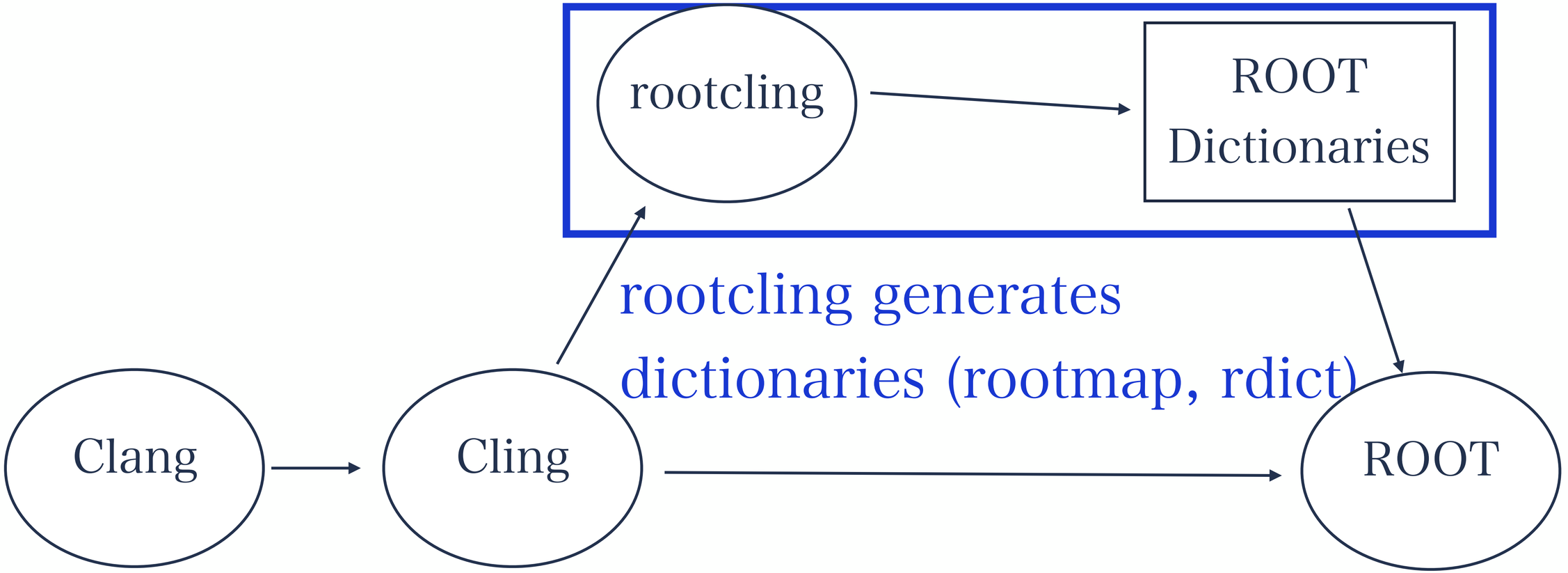



Cling calls Clang API

binaries

files

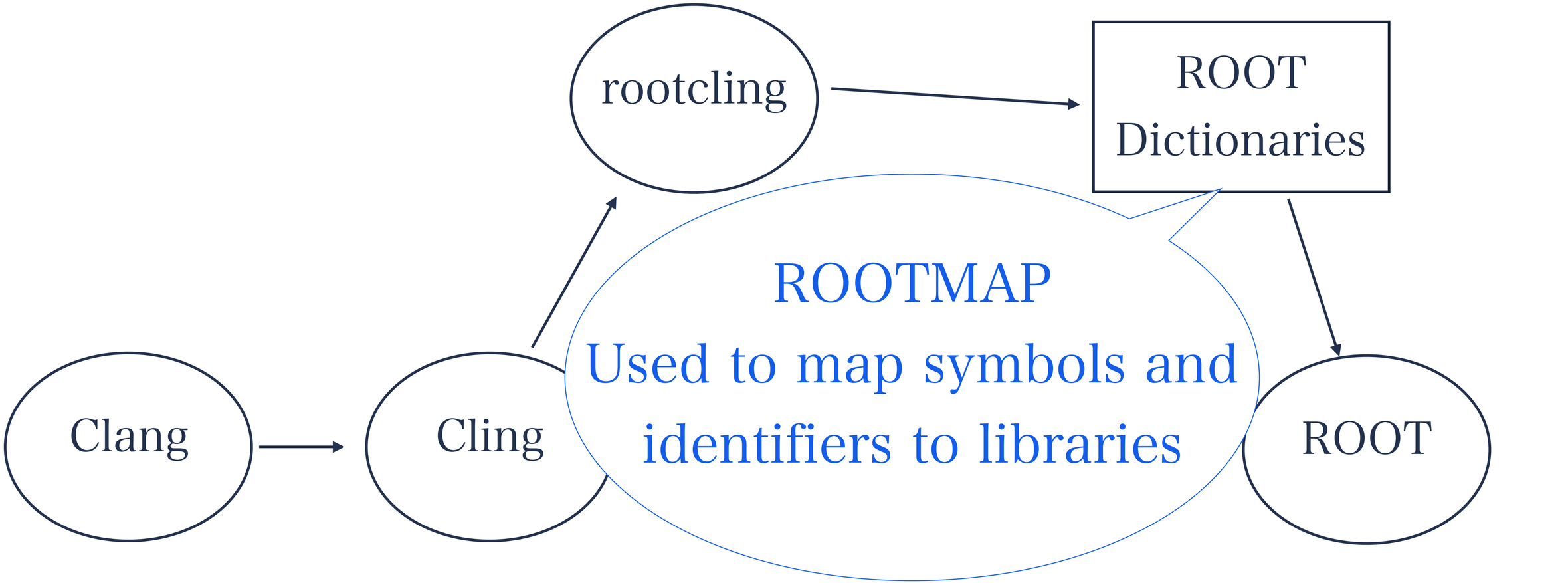
# Implementation Details in ROOT



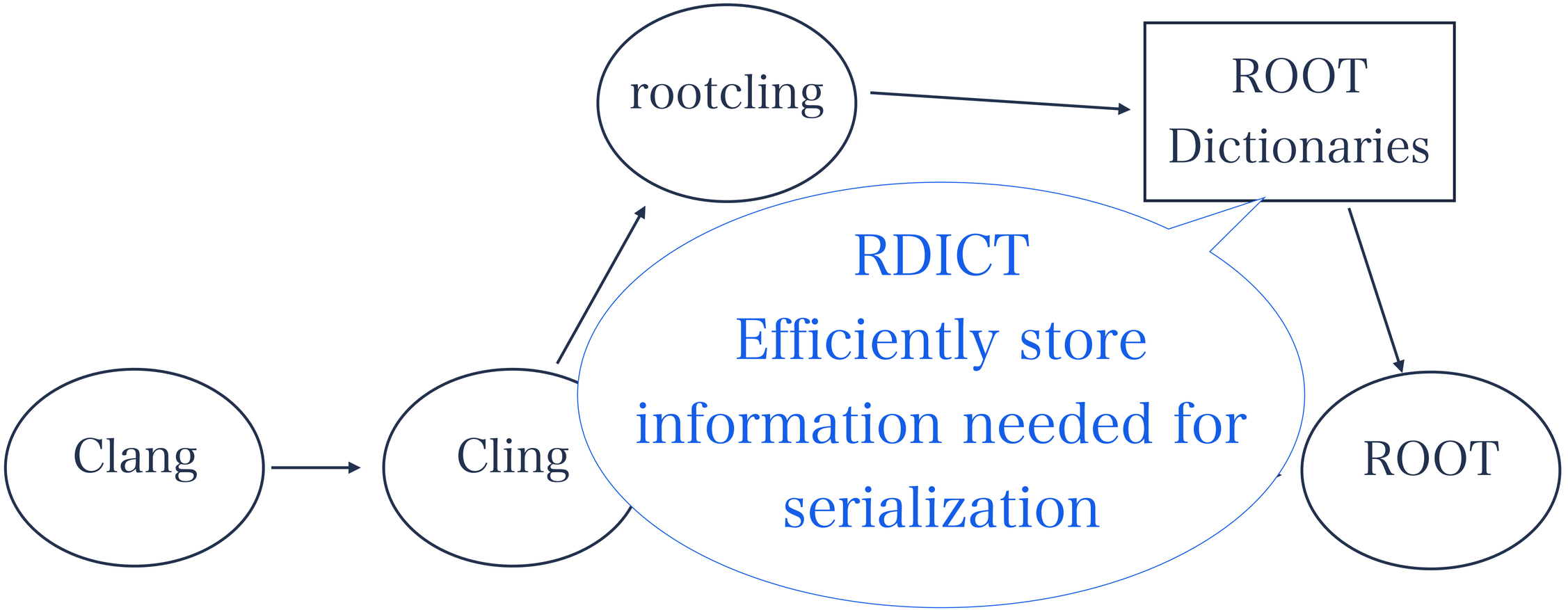

binaries

files

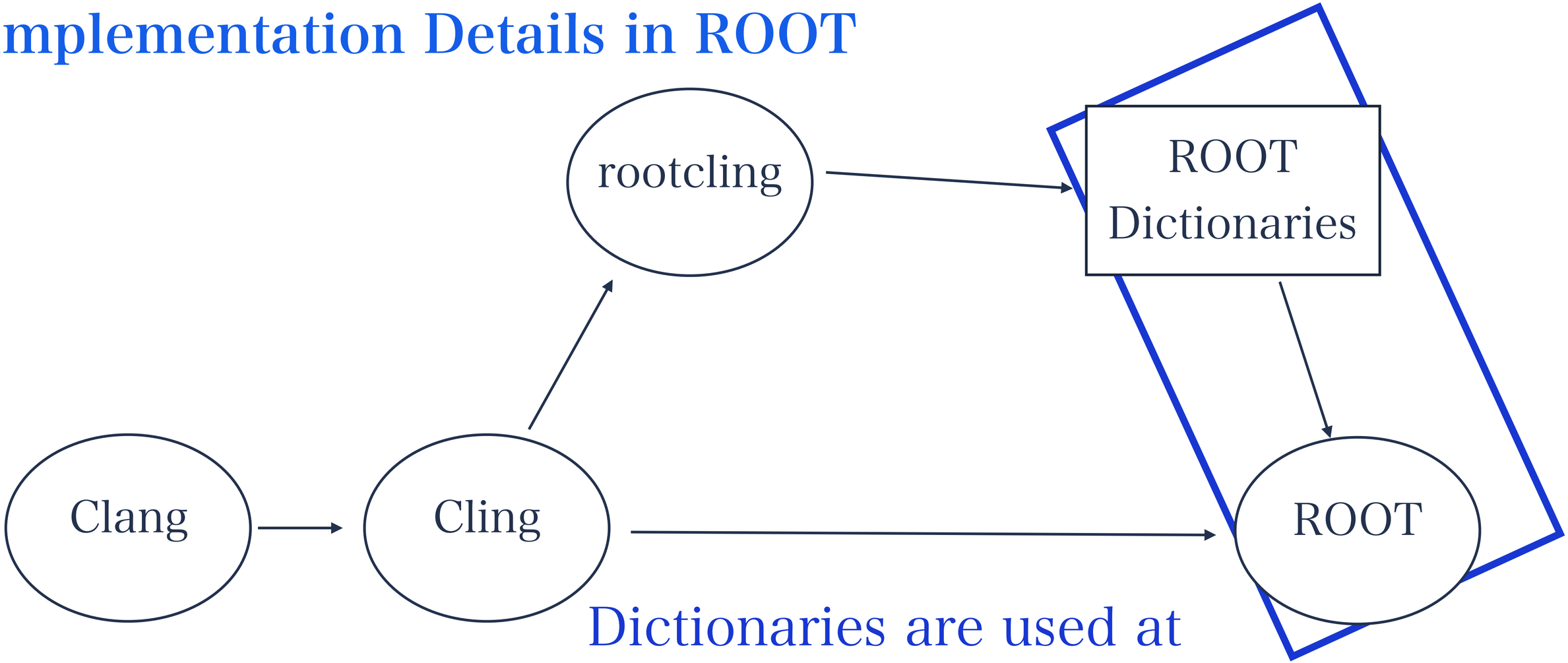
# Implementation Details in ROOT



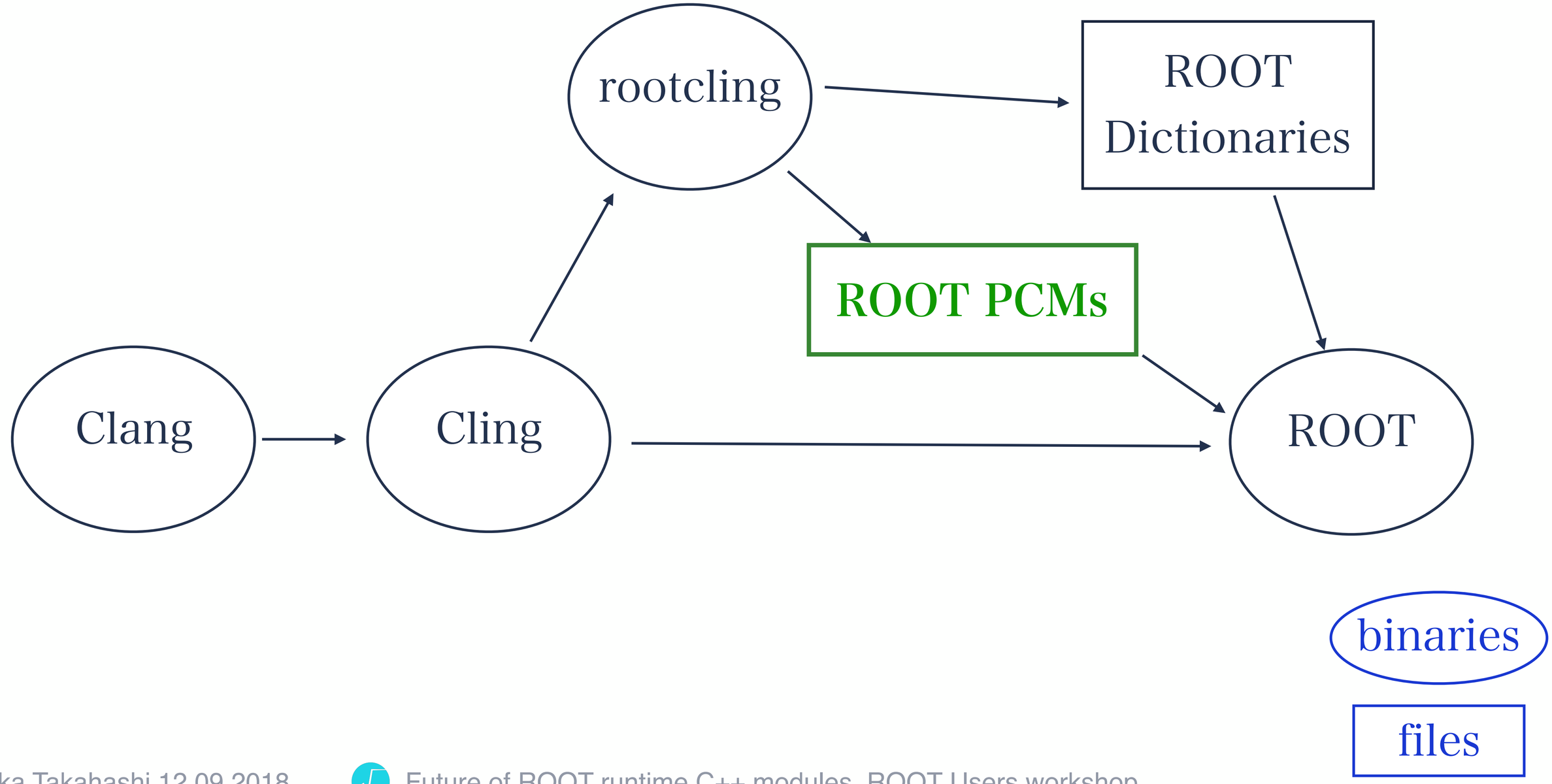
# Implementation Details in ROOT



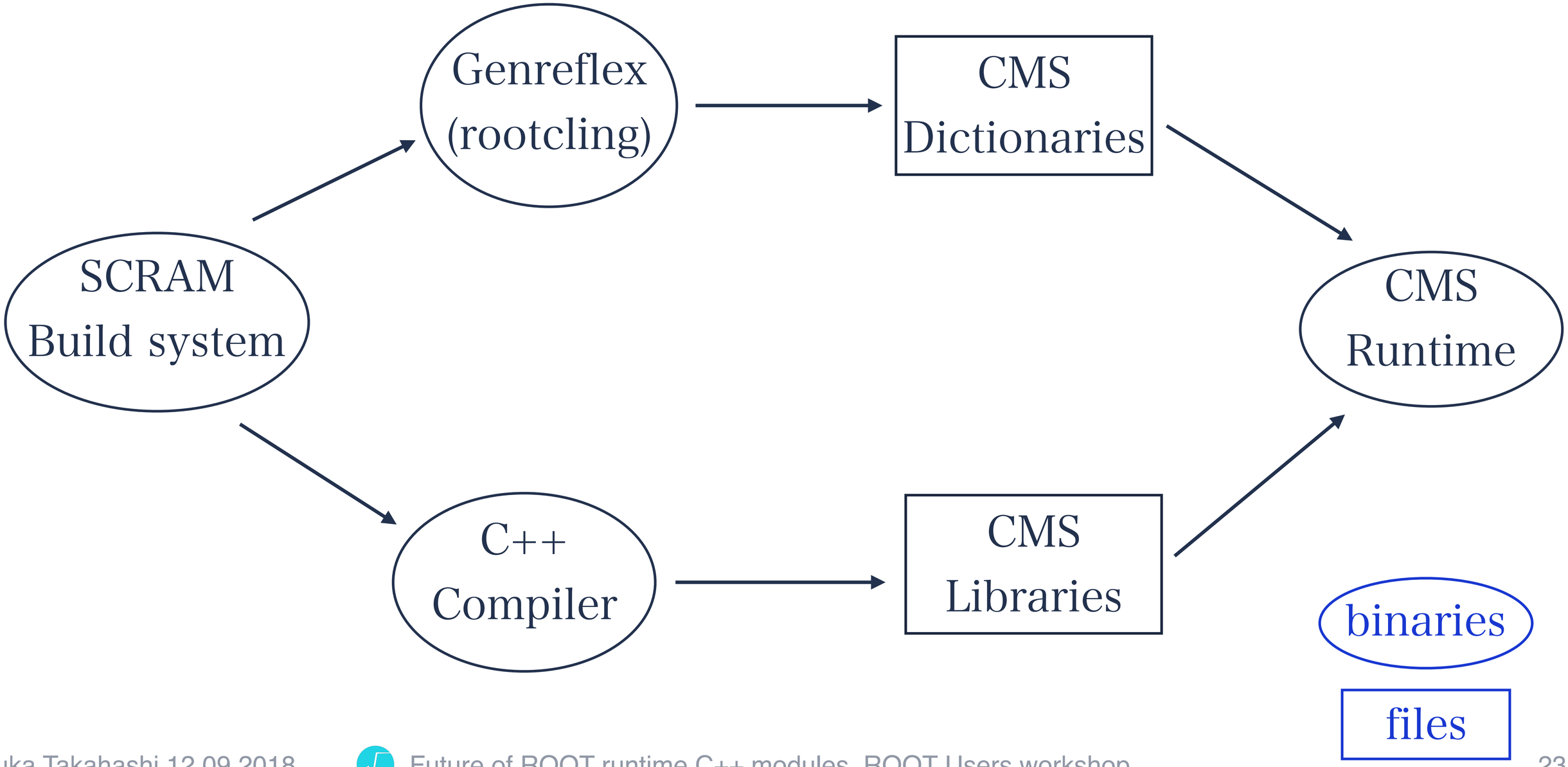
# Implementation Details in ROOT



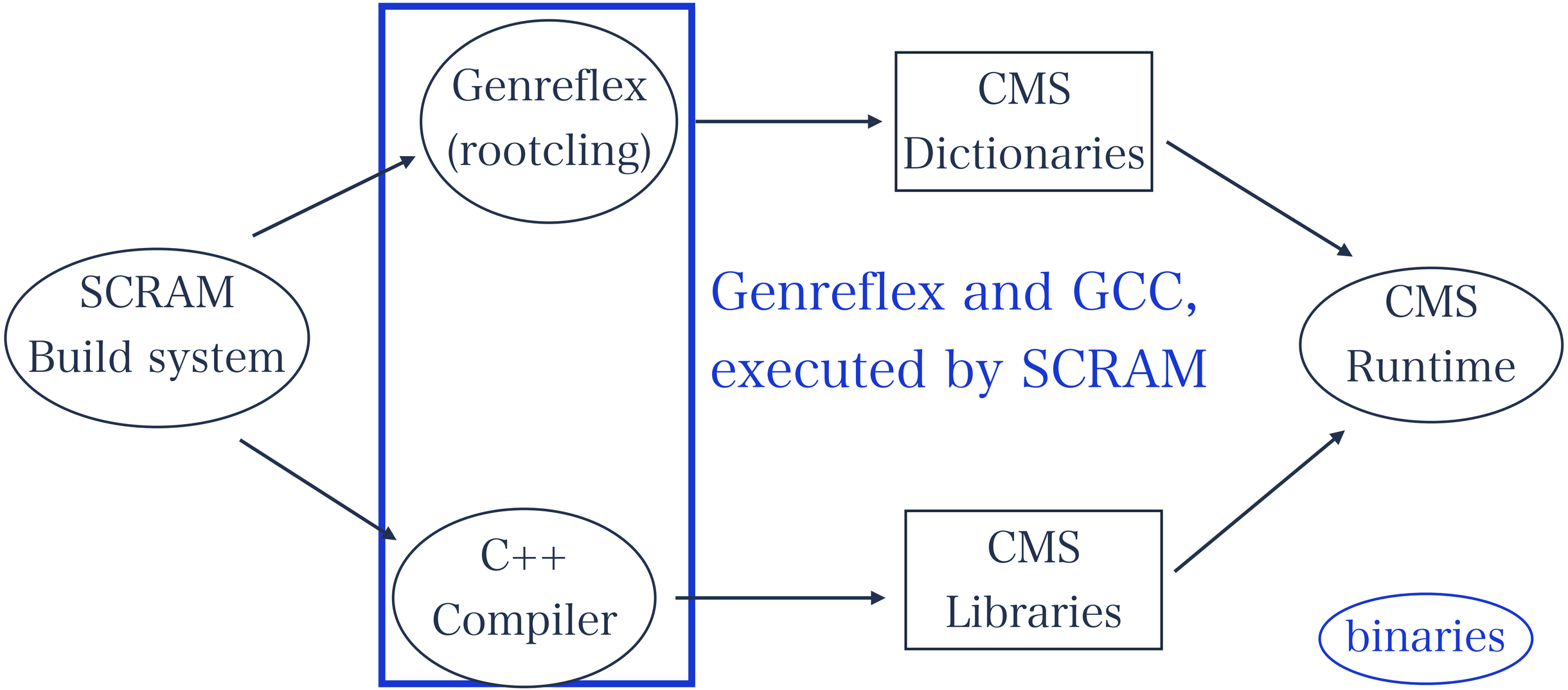
# Implementation Details in ROOT



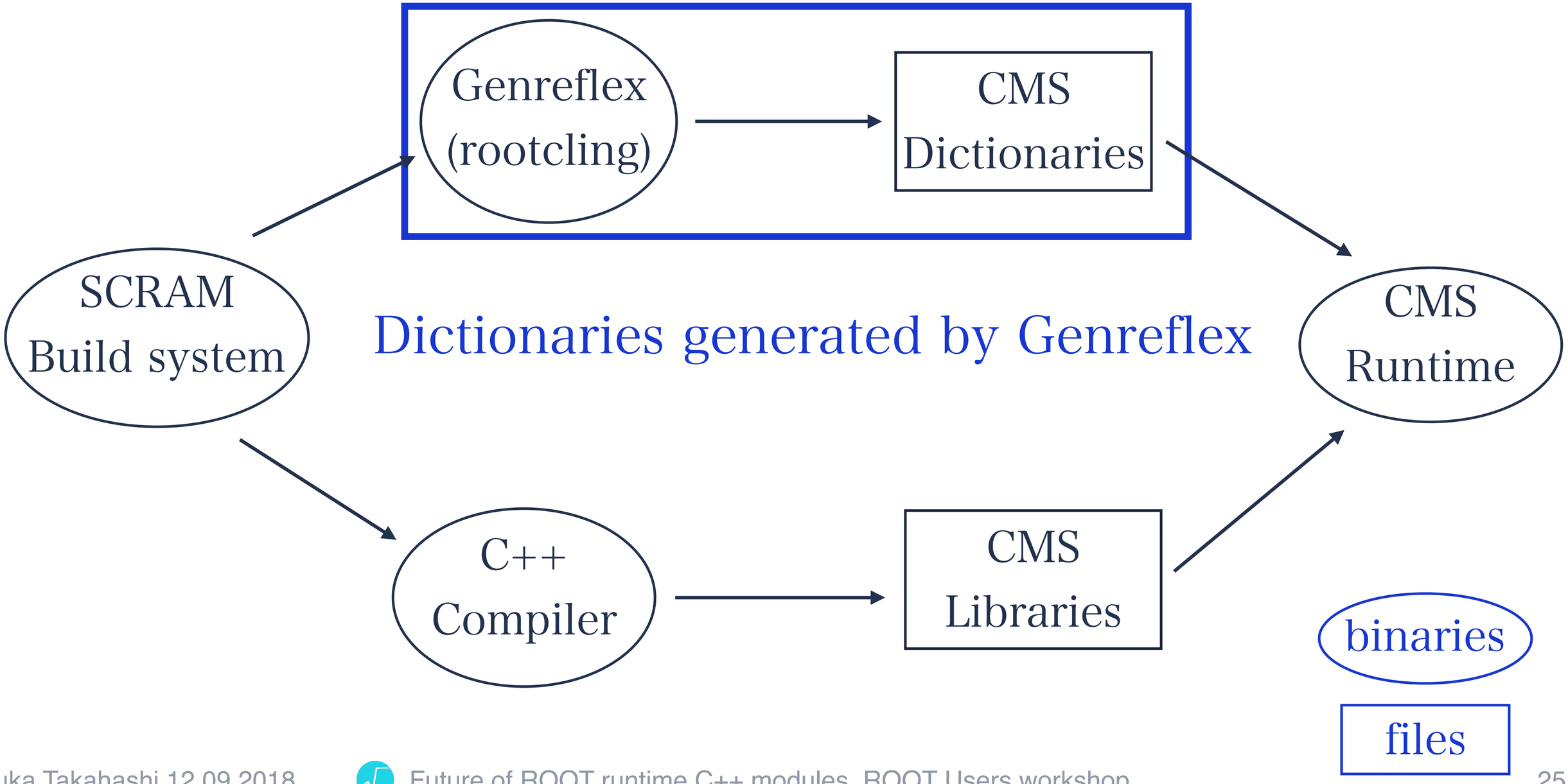
# Implementation Details in CMSSW



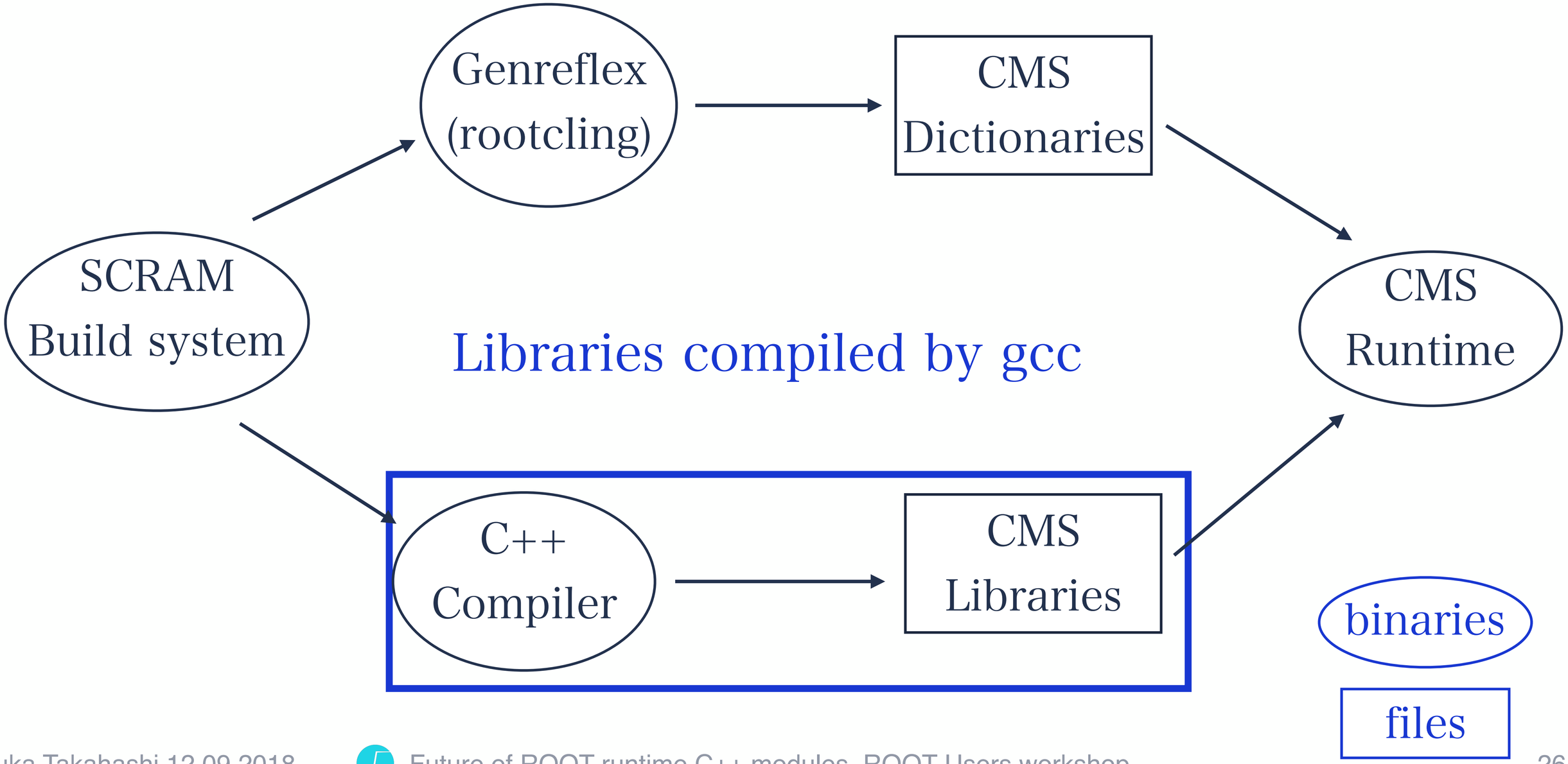
# Implementation Details in CMSSW



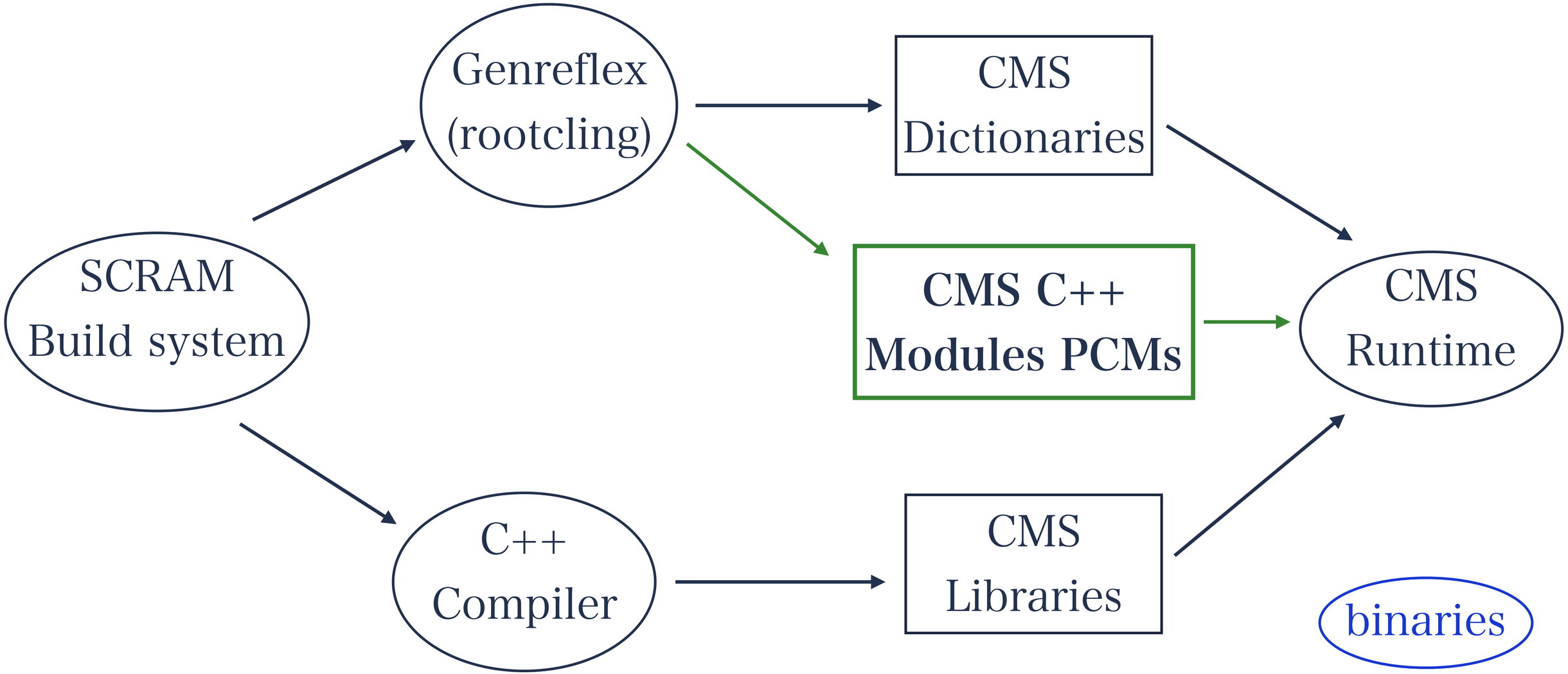
# Implementation Details in CMSSW



# Implementation Details in CMSSW



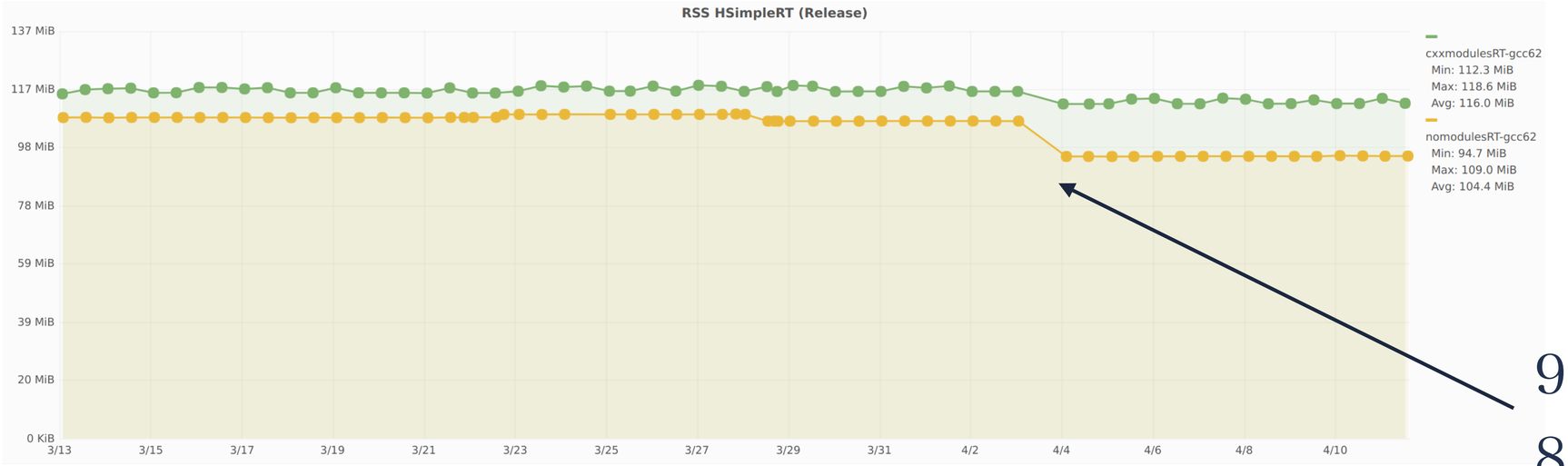
# Implementation Details in CMSSW



# 2018 Roadmap

# Roadmap


- Start working on ROOT
- Optimize reflection layer, reduce eagerly deserialized decls




http://root-bench.cern.ch

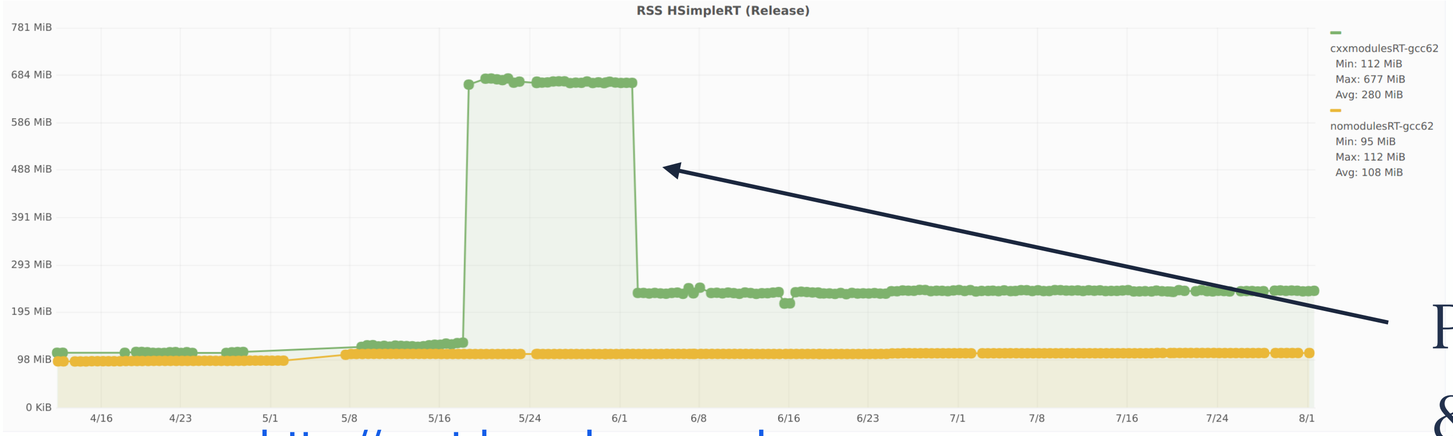
9.2% cpu time

8.8% memory improvement

# Roadmap


Reduce Eagerly

- Start working on ROOT Modules
- Fixed 142 / 153 tests in ROOT test suite




http://root-bench.cern.ch

Preloading of all modules & Bloom filter

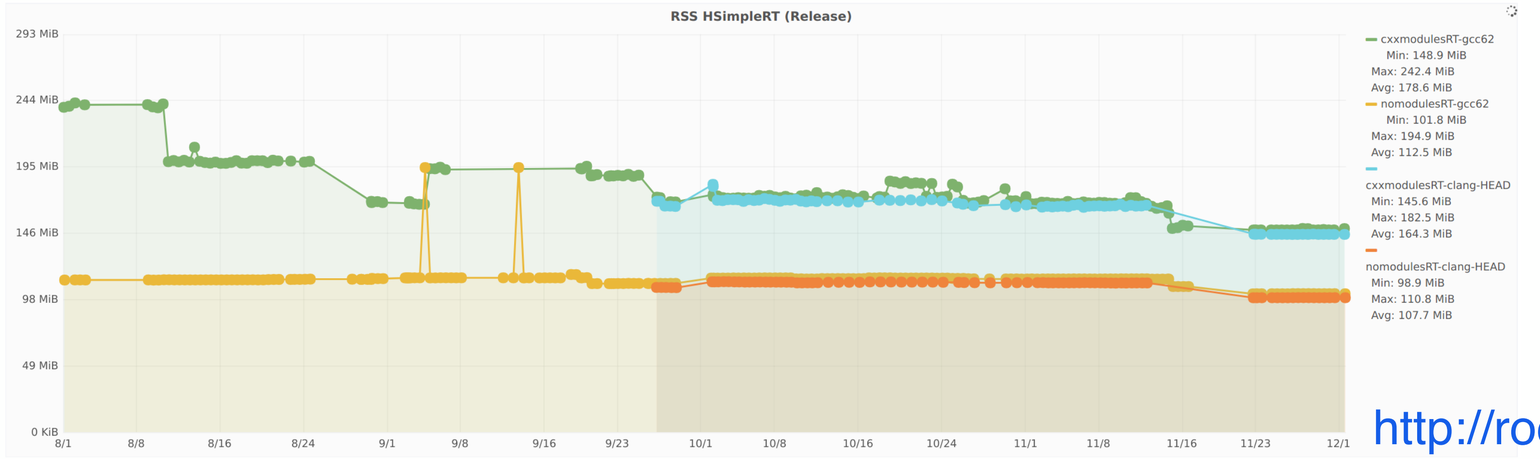
# Roadmap

03/18 04/18 07/18 12/18 02/19

Reduce Eagerly

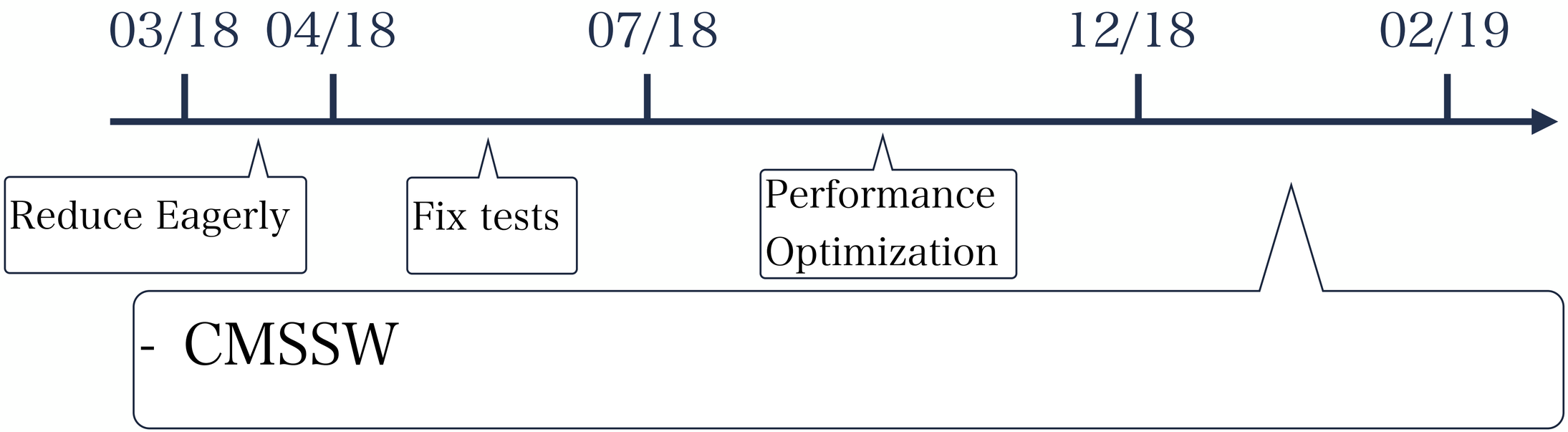
Fix tests

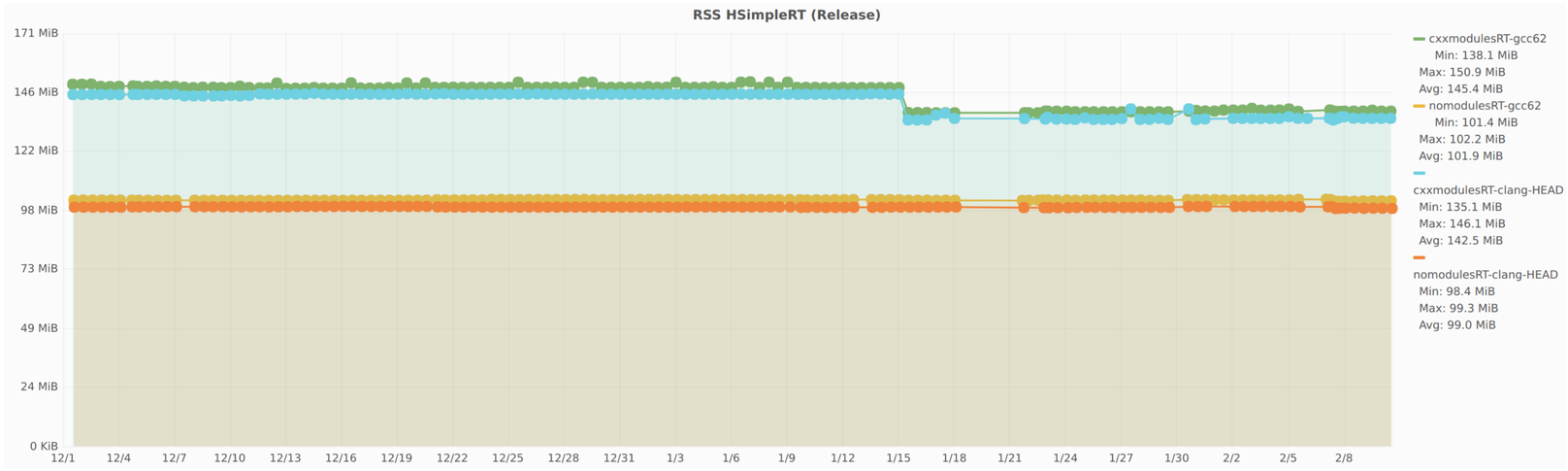
- Performance optimization on ROOT side
- ROOT 6.16 Release




http://root-bench.cern.ch

# Roadmap




http://root-bench.cern.ch

# Current Status
# ROOT and CMSSW

# Current Status



1. Technology Preview released in ROOT 6.16
   - Full support on Linux OS
   - Mac OS support enabled by V.Vassilev
2. Green nightly builds

# Current Status

CMSSW


1. Enable ROOT runtime modules in CMSSW Done
2. Add C++ Modules IB to CMSSW Done
3. Generate implicit pcms for CMSSW Done
4. Generate explicit pcms for CMSSW WIP
5. Generate explicit pcms for external libraries WIP

# Current Status

CMSSW

1 - End of December

Enable ROOT runtime C++ modules in CMSSW Done

- ROOT with -Druntime_cxxmodules=On
- All CMSSW and external libraries are compiled
- All tests are passing

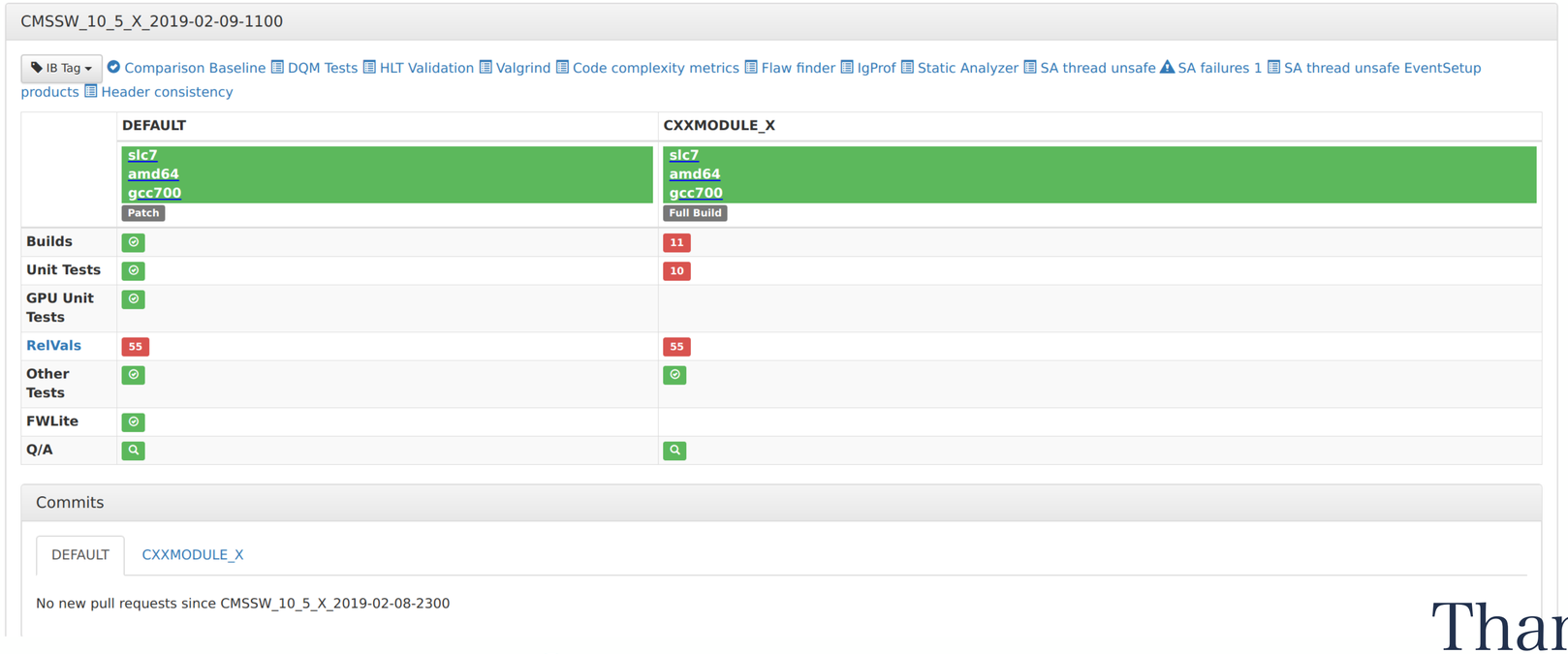
# Current Status

CMSSW

1 2 - End of December

Add C++ Modules IB to CMSSW Done

- Regularly run full CMS integration builds




Thanks Shahzad!!

## Implicit pcms and explicit pcms

“Implicit pcms” is implicitly generated without modulemaps

- Puts all possible header files needed to generate a dictionary
  - **Huge** header duplication

## module map and explicit pcms

“Explicit pcms” can be generated by introducing “module maps”

```
module "MathCore" {
  requires cplusplus
  module "TComplex.h" { header "TComplex.h" export * }
  module "TMath.h" { header "TMath.h" export * }
  module "TRandom.h" { header "TRandom.h" export * }
  module "TRandom1.h" { header "TRandom1.h" export * }
… }
```

Module map is a definition file of headers for pcms

- Reduces header duplication between modules

# Current Status

CMSSW


Generate implicit pcms for CMSSW Done

- genreflex with - - `cxxmodule` works
- 258 CMSSW pcms were generated (3.2GBytes)
- 10% of all CMSSW tests are failing
  - These'll be fixed with modulemaps (next slide)

# Current Status

CMSSW


Generate explicit pcms for CMSSW WIP

- 25 out of 107 DataFormats libraries
- 2 weeks to modularize first one library
  - After having the infrastructure ready, other libraries were modularized in one day

# Current Status

CMSSW


Generate explicit pcms for external libraries WIP

- Raphael created compilation-time modulemap for boost and other external libraries (boost, libxml)
- Needs to be refreshed, and to be integrated to runtime system

# Performance Results

# CMSSW Performance

CMSSW with ROOT master

↓

CMSSW with ROOT pcms (-Druntime_cxxmodules=On)

Core.pcm, RIO.pcm, etc.

↓

CMSSW with ROOT pcms + genreflex CMS pcms (25 pcms)

DataFormatsCommon_xr.pcm, DataFormatsMath_xr.pcm..

# CMSSW Performance

CMSSW with ROOT master

CMSSW with ROOT pcms (-Druntime_cxxmodules=On)

Core.pcm, RIO.pcm, etc.

CMSSW with ROOT pcms + genreflex CMS pcms (25 pcms)

DataFormatsCommon_xr.pcm, DataFormatsMath_xr.pcm..

# CMSSW with ROOT master

2018 CMS detector Digitization with pile-up (250199.18)

Real Time

- Total loop: Mean=334.3s, S=11.0
- Total init: Mean=66.50s, S=6.1
- Total job: Mean=401.6s, S=16.3
- EventSetup Lock: Mean=16.3s, S=4.9
- EventSetup Get: Mean=25.2s, S=13.5

CPU Time

- Total loop: Mean=676.7s, S=25.3
- Total init: Mean=12.6s, S=1.53

RSS

- Mean (Average): 4119.29 Mbytes (3548.29 - 4672.56)

10 events
Average of 5
times execution

# CMSSW with ROOT pcms

2018 CMS detector Digitization with pile-up (250199.18)

Real Time

- Total loop: Mean=314.4s, S=13.17
- Total init: Mean=69.9s, S=16.2
- Total job: Mean=385.3s, S= 26.2
- EventSetup Lock: Mean=13.9s, S=1.8
- EventSetup Get: Mean=16.6s, S=2.6

CPU Time

- Total loop: Mean=678.7s, S=36.8
- Total init: Mean=12.6s, S=1.5

RSS

- Mean (Average): 4523.152 Mbytes (3795.09 - 4927.45)

10 events
Average of 5
times execution

# CMSSW genreflex CMS pcms

2018 CMS detector Digitization with pile-up (250199.18)

Real Time

- Total loop: Mean=322.73s, S=9.83
- Total init: Mean=66.23s, S=4.6
- Total job: Mean=389.9s, S=13.7
- EventSetup Lock: Mean=15.05s, S=4.4
- EventSetup Get: Mean=15.58s, S=2.54

11.6s faster than ROOT master

11.7s faster than ROOT master

9.62s faster than ROOT master

CPU Time

- Total loop: Mean=684.7s, S=23.7
- Total init: Mean=12.36s, S= 2.0

RSS

- Mean (Average): 4841.198 Mbytes (4556.75 - 5006.84)

10 events
Average of 5 times execution

# CMSSW with ROOT master

2015 detector CMS digitization workflow (500199.0)

Real Time

- Total loop: Mean=153.9s, S=30.067
- Total init: Mean=71.7s, S=3.628
- Total job: Mean=228.079s, S=30.178
- EventSetup Lock: Mean=9.98s, S=3.64
- EventSetup Get: Mean=23.48s, S=16.134

CPU Time

- Total loop: Mean=278.84s, S=37.857
- Total init: Mean=11.39s, S=2.28

RSS

- Mean (Average): 2544.224 Mbytes (2491.2 - 2625.57)

10 events
Average of 5
times execution

# CMSSW with ROOT pcms

2015 detector CMS digitization workflow (500199.0)

Real Time

- Total loop: Mean=149.16s, S=14.57
- Total init: Mean=72.12s, S=s5.84
- Total job: Mean=221.94s, S=19.90
- EventSetup Lock: Mean=7.856s, S=1.175
- EventSetup Get: Mean=1.57s, S=0.820

CPU Time

- Total loop: Mean=269.86s, S=13.84
- Total init: Mean=11.24s, S=0.399

RSS

- Mean (Average): 2618.754 Mbytes (2540.24 - 2717.27)

10 events
Average of 5
times execution

# CMSSW genreflex CMS pcms

2015 detector CMS digitization workflow (500199.0)

Real Time

- Total loop: Mean=146.03s, S=7.069
- Total init: Mean=70.56s, S=6.887
- Total job: Mean=217.11s, S=3.49
- EventSetup Lock: Mean=8.169s, S=0.81
- EventSetup Get: Mean=11.77s, S=1.588

7.87s faster than ROOT master

10.96s faster than ROOT master

11.71s faster than ROOT master

CPU Time

- Total loop: Mean=269.51s, S=10.762
- Total init: Mean=11.67s, S=0.691

RSS

- Mean (Average): 2656.314 Mbytes (2366.51 - 2941.77)

10 events
Average of 5 times execution

# Performance Results

For ACAT 2019

Make more progress towards CMSSW

- Extend the tests results
- Increase the number of events

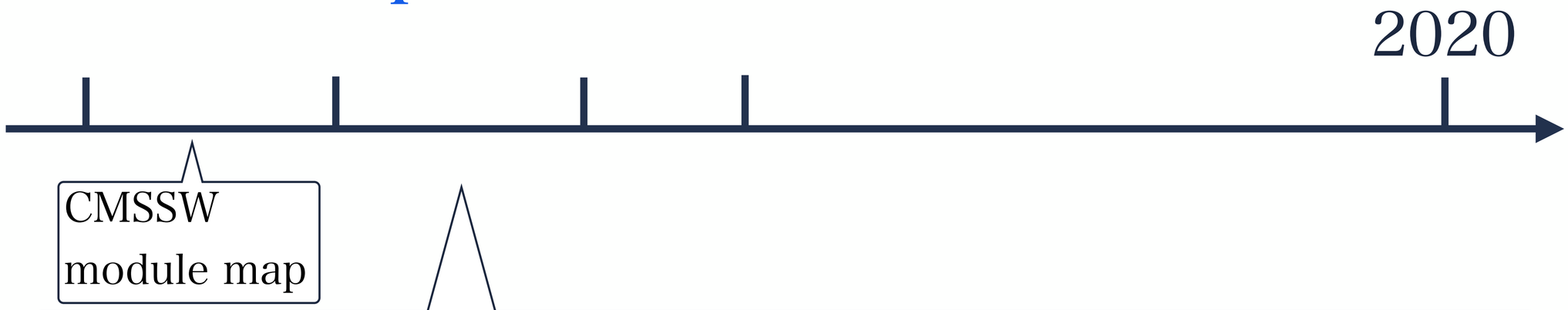
Thanks a lot for David and Shahzad for advises & explanations!!

# Future Roadmap

# Future Roadmap


- 2 months “full time student” (FTS): Generate remaining explicit pcms for CMSSW

# Future Roadmap



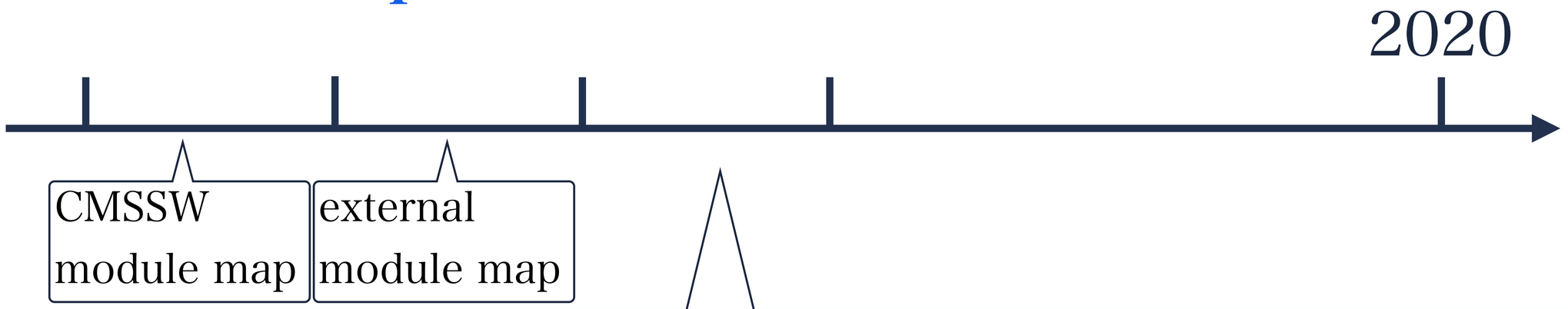

- 3 months FTS: Global Modules Index
- 3 months FTS: Generate explicit pcms for external libraries

# Future Roadmap

## Global Modules Index

- Remove further overhead in ROOT, introduced by preloading
- Mechanism to create the table of symbols and PCM names
  - ROOT will be able to load corresponding library when a symbol lookup failed
  - V.Vassilev already has a prototype, which shows a promising results

# Future Roadmap




- 3 months FTS: Reduce header duplication for optimization

# Remaining Two weeks

- Write documentation
- Prepare for ACAT 2019
- Upstream patches to CMSSW

# Acknowledgment

Huge thanks to everyone involved, especially

Vassil, Oksana, Shahzad, Axel, David, Mircho, Raphael, and the ROOT team in general!!

# Thank you for your attention!

# Backup slides

# Implementation in ROOT

## Terminology

ROOTMAP

- Used to map symbols and identifiers to libraries

RDICT

- Efficiently store information needed for serialization

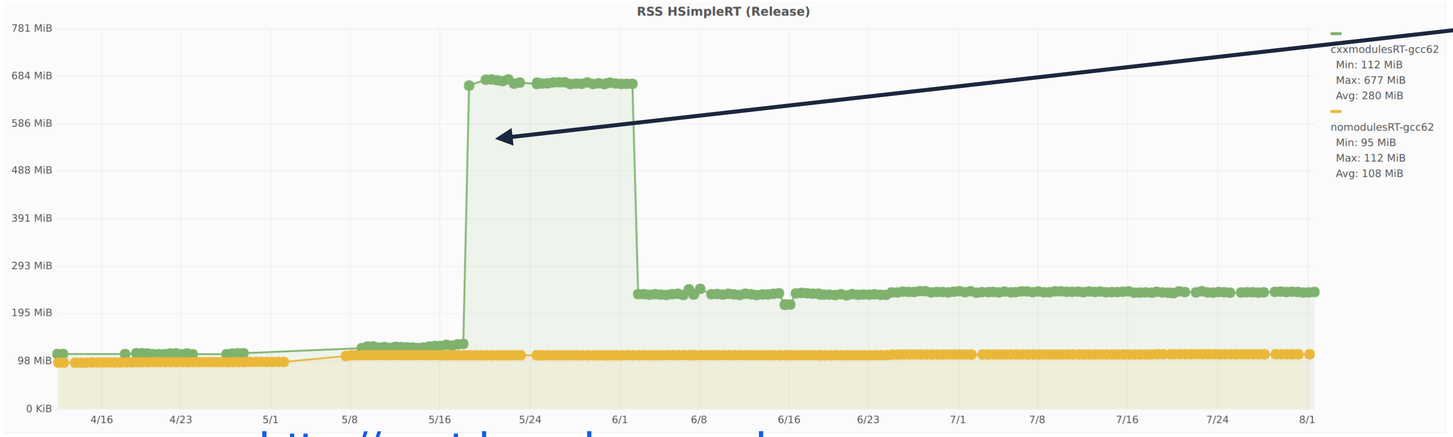
Preloading of all modules

Preloading C++ modules offers a stable implementation

- Partly replace their performance benefits

# Implementation in ROOT

Terminology




http://root-bench.cern.ch

Kind of expensive :)

Preloading C++ modules offers a stable implementation

- Partly replace their performance benefits

# Implementation in ROOT

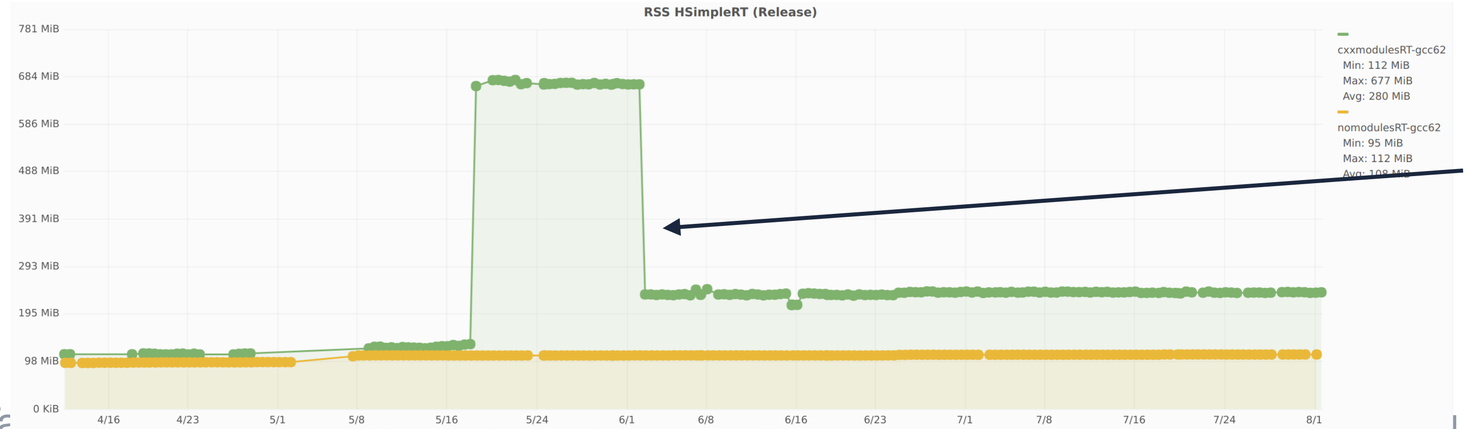
Terminology

Bloom filter

Optimization for the library autoloading

Shared object files contains .gnu.hash section

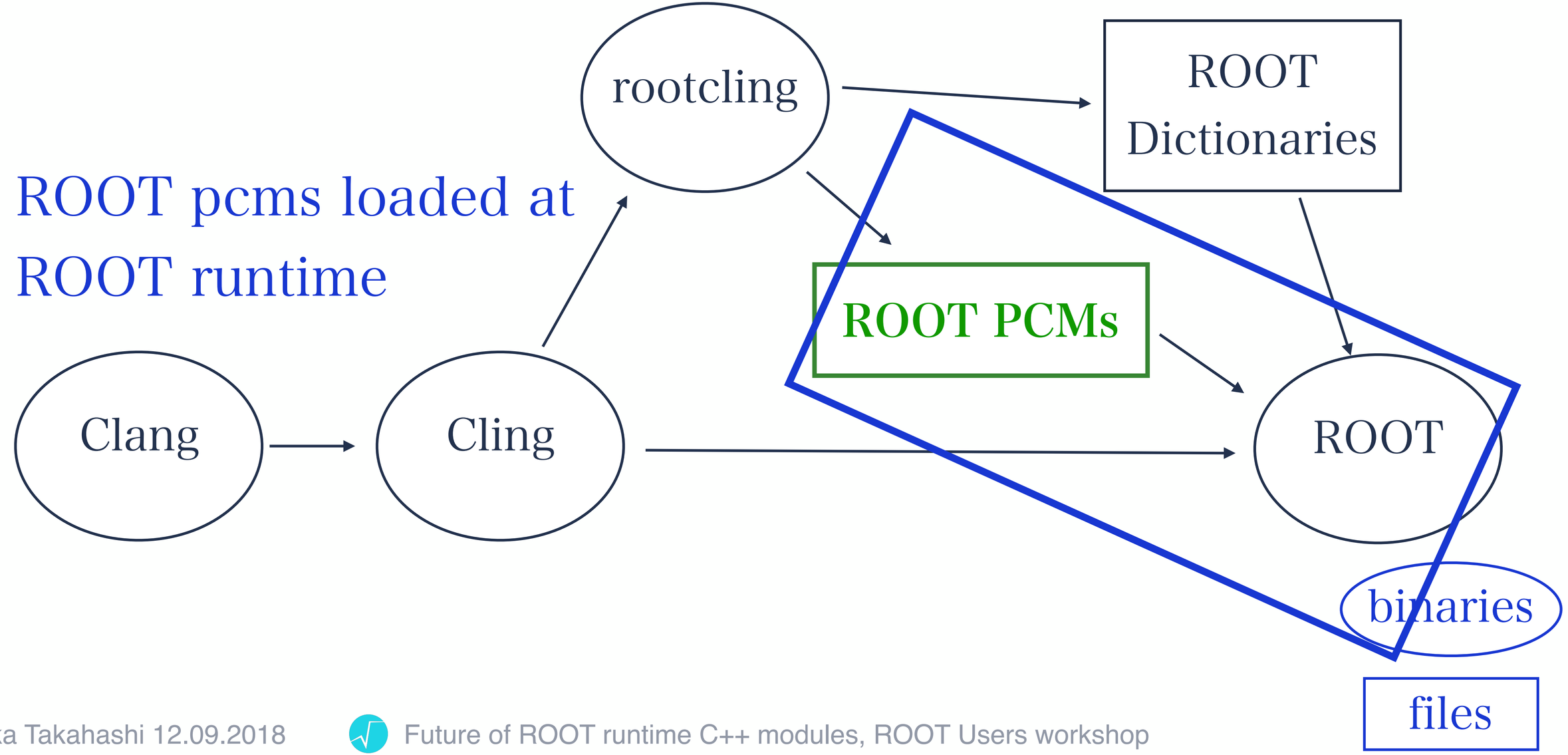
- Bloom filter hash is a false positive probability data structure
- Skip libraries which clearly doesn't contain mangled name



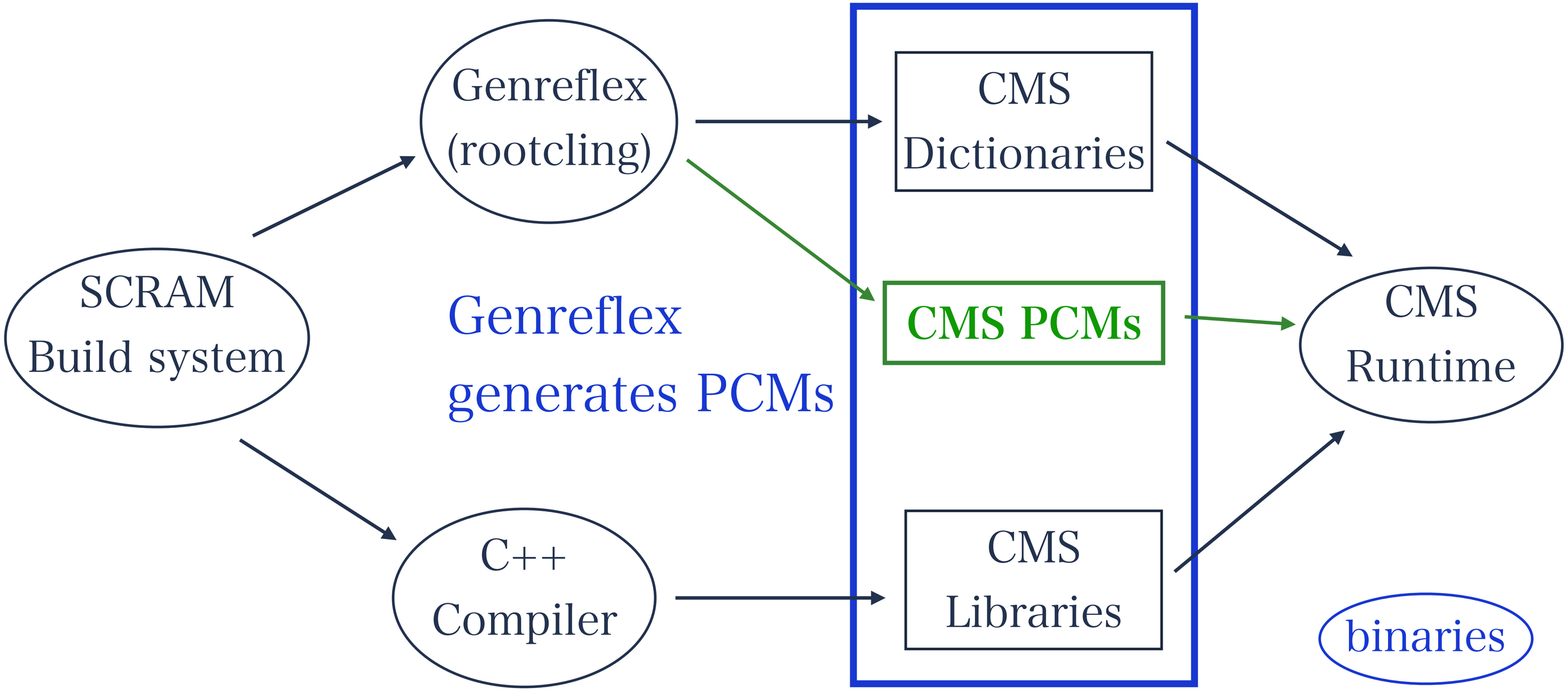

Reduce the overhead

http://root-bench.cern.ch

# System overview of ROOT



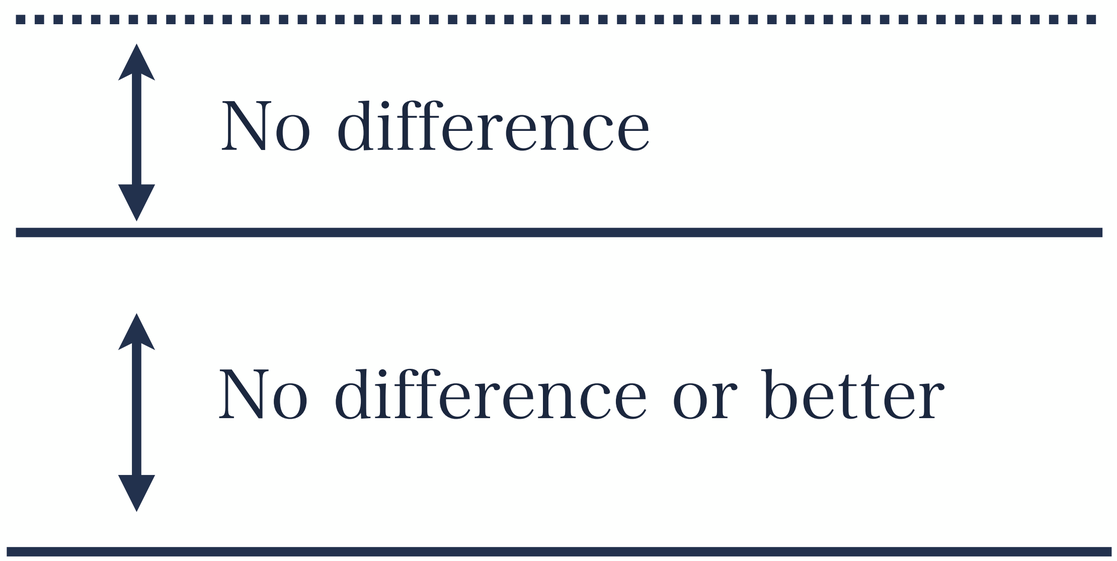
# System overview of CMSSW



# Performance Results

## CMSSW Performance

Runtime



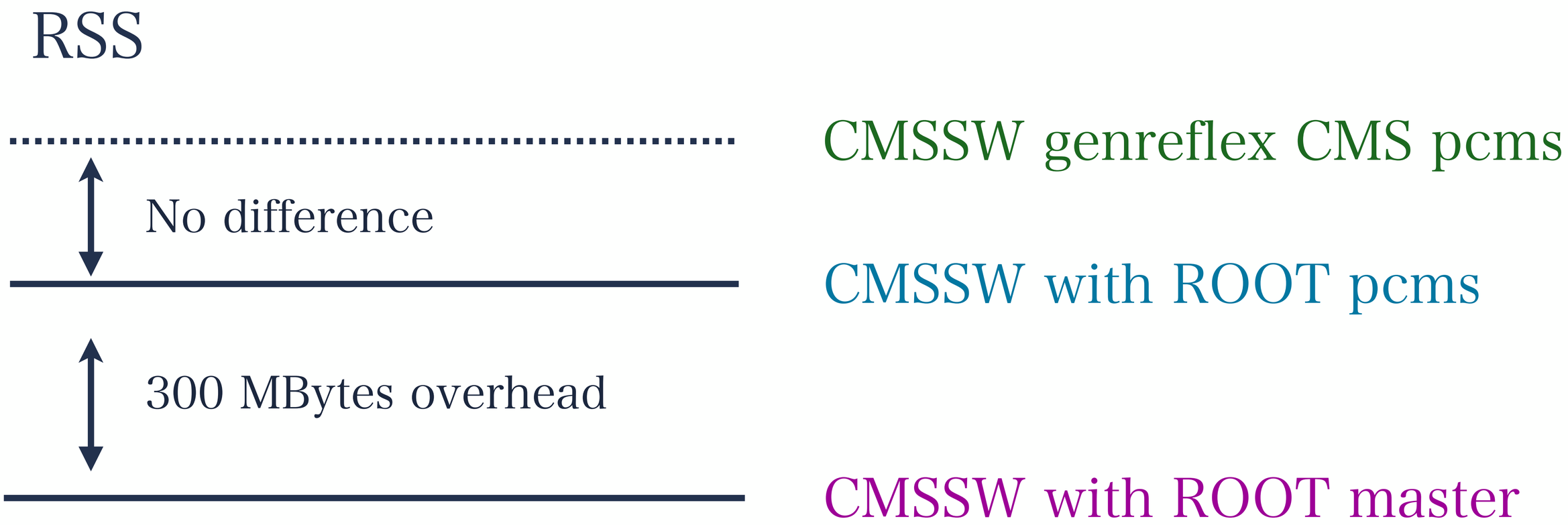

CMSSW genreflex CMS pcms

CMSSW with ROOT pcms

CMSSW with ROOT master

# Performance Results

# CMSSW Performance



# Performance Results

For ACAT 2019

Measure those tests with more events

- 250202.118
- 250399.17
- 10824

Thanks a lot for David for advise & explanation!!