C++ Modules in ROOT and Beyond

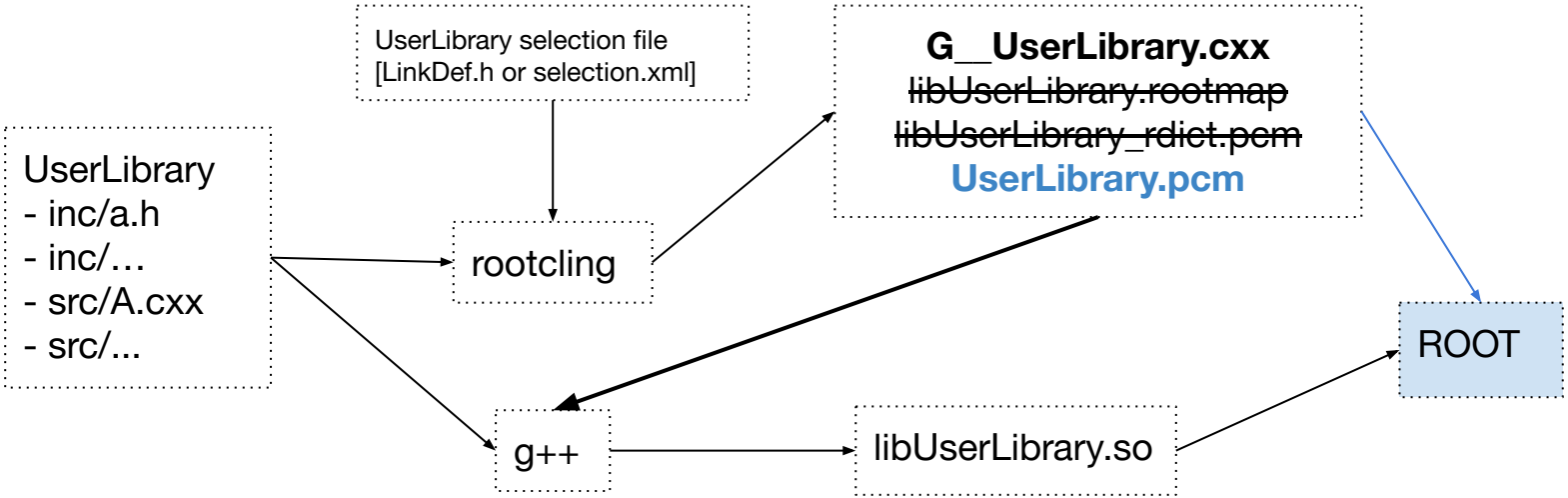
Vasil Georgiev Vasilev (Princeton University (US))
Yuka Takahashi (Princeton University (US))
David Lange (Princeton University (US))
Oksana Shadura (University of Nebraska Lincoln (US))

Motivation

- ROOT interacts with user libraries which require implicit header inclusion. This can be triggered by reading or writing data on disk, or user actions at the prompt.
- **C++ Modules are designed to minimize the reparsing of the same header content** via an efficient on-disk representation of C++ Code
- In practice, the pre-compiled header (PCH) is such representation but per process (for a predefined set of libraries)
  - **The C++ Modules are a “PCH” per library**

Dictionaries in ROOT

- ROOT provides a set of benefits to users automatically. For example, **automatic generation of serialization information (I/O) and automatic header and dependency resolution**.
- In order to do so, an extra file C++ needs to be synthesized and compiled into the defining library. This file is called a dictionary file.



Dictionaries in ROOT

G__UserLibrary.cxx has several sections which require reprocessing immutable headers in runtime

They are replaced by the much more efficient C++ Modules

```
-#include "TEveTrack.h"
-#include "TEveTrackEditor.h"
-#include "TEveTrackGL.h"
-#include "TEveTrackProjected.h"
-#include "TEveTrackProjectedGL.h"
-#include "TEveTrackPropagator.h"
-#include "TEveTrackPropagatorEditor.h"
-#include "TEveTriangleSet.h"
-#include "TEveTriangleSetEditor.h"
-#include "TEveTriangleSetGL.h"
...
-"TEveGedNameTextButton", payloadCode, "@",
-"TEveGeoManagerHolder", payloadCode, "@",
-"TEveGeoNode", payloadCode, "@",
-"TEveGeoNodeEditor", payloadCode, "@",
...
-class __attribute__((annotate(R"ATTRDUMP(An arbitrary polyline with fixed line and marker attributes.)ATTRDUMP"))) ..
```

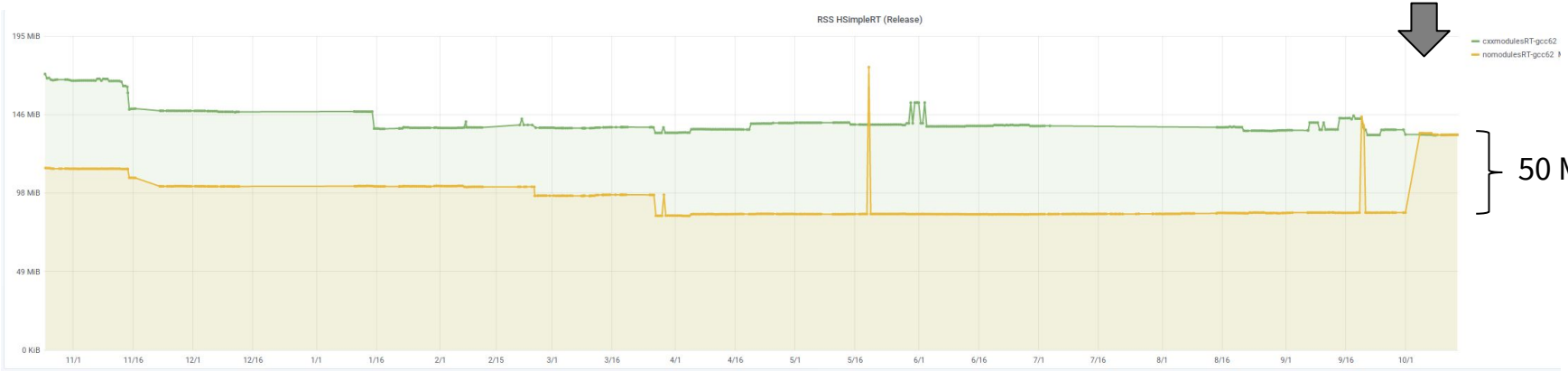
C++ Modules in ROOT

- Technology Preview released in ROOT 6.16
- **Default on UNIX in ROOT 6.20**

C++ Modules in ROOT. Loading strategies

- *Eager loading* of *pcm files at startup in ROOT 6.20
  - Small linear performance overhead depending on the number of modules
- *Delayed loading based on global module indexing (GMI)* planned for ROOT 6.22
  - **No overhead, pay only for what you use!**

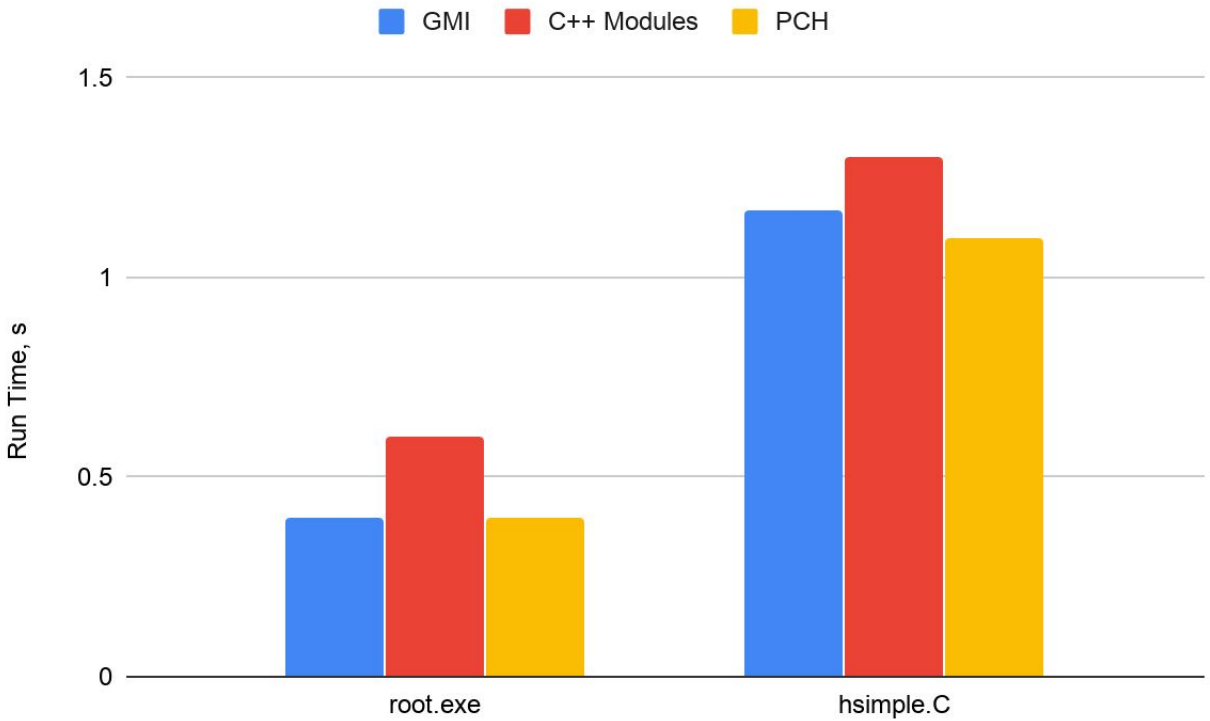
C++ Modules become default on UNIX



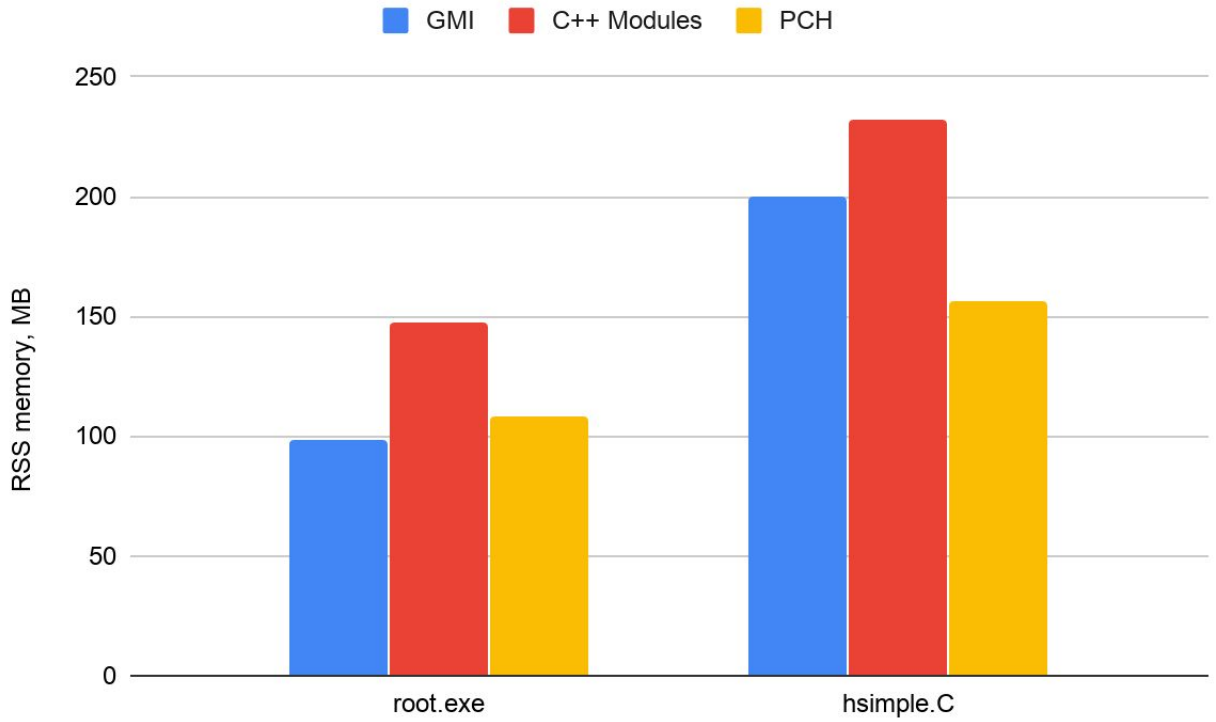
Global Module Index (GMI)

- The Global Module Index (**`clang::GlobalModuleIndex`**) is **an aid for name lookup into modules**
- It allows the client *to restrict the search to only those module files known* to have a information about that identifier, **improving performance**!
- **Global Module Index returns a set of modules where a given identifier is present**
  - This allows us to avoid eager loading of all modules and load only necessary ones on demand

GMI index measurements



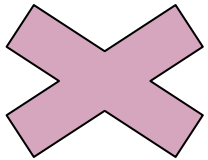
GMI index measurements

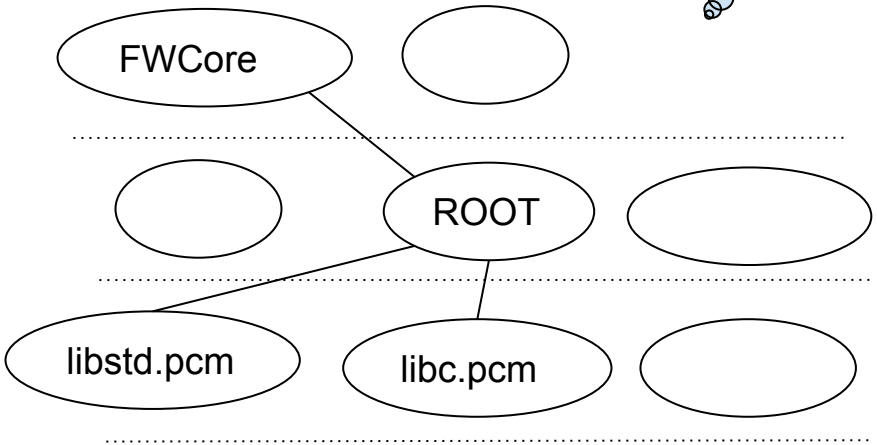


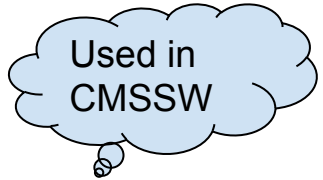
C++ Modules in CMSSW

- Available in CMS_CXXMODULE_IB

C++ Modularization in CMSSW







Top-down approach

Library without modulemap

- Header file pollute the rest of the library dependencies
  - **The header duplication is problematic for performance but also triggers a lot of bugs when resolving the duplicate content**

Bottom-up approach

Library with modulemap

- Headers are persisted in the PCM
  - **Header duplication is reduced by referencing the dependent module**

Dictionaries Using C++ Modules in CMSSW

[Automatic generation of modulemap]

- CMSSW has “interface” headers
  - Exposed to libraries outside
- Automatically generate the modulemap by adding interface headers
  - Modulemap needs to be generated before the execution of genreflex
- *Design features*
  - *Headers can be easily vetoed from generation of modulemap via “env” settings*

C++ Modules-aware dictionary generation in CMSSW

[module.modulemap]

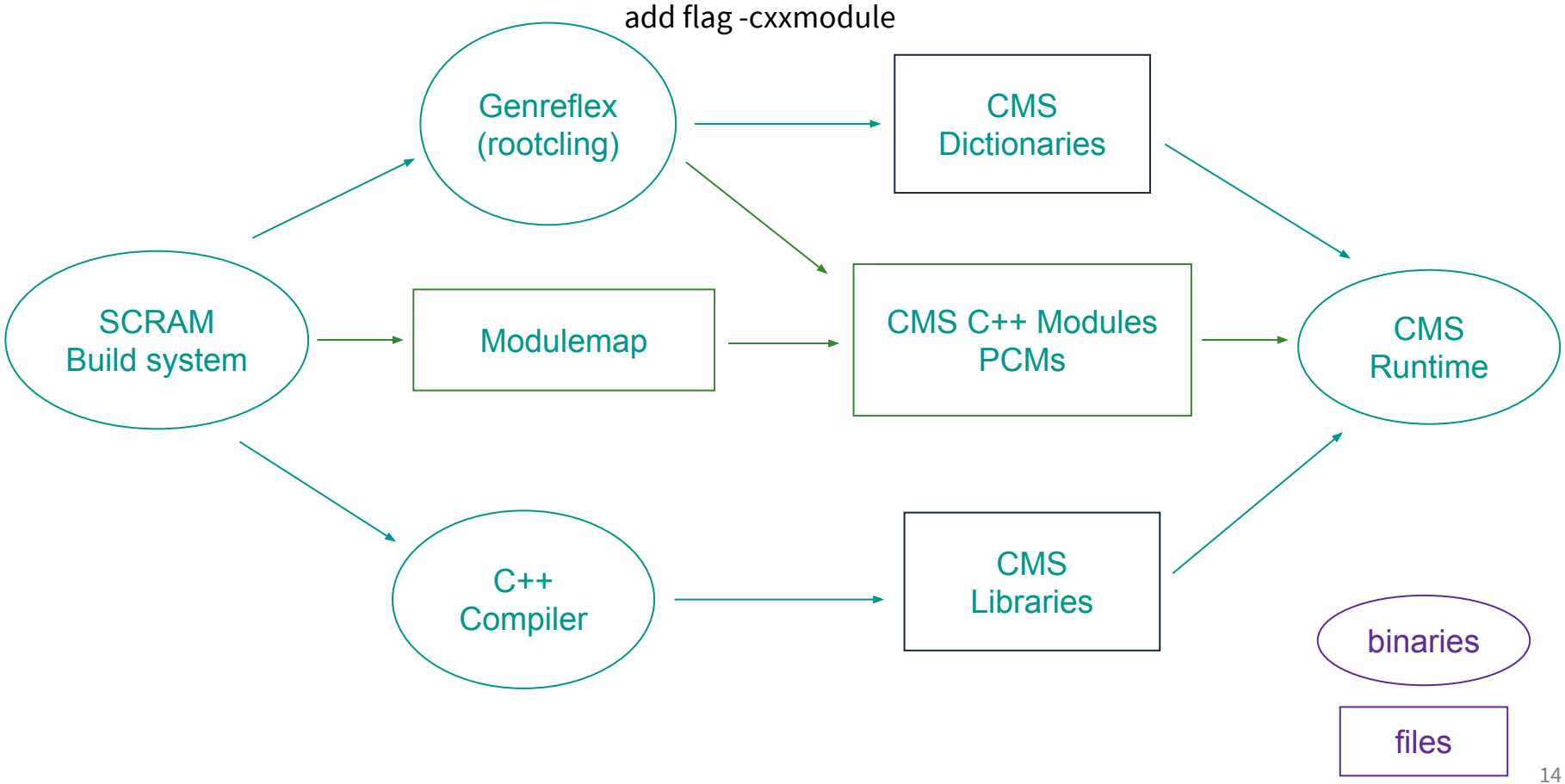
- Definition file of headers to build a PCM in Clang
- Contain all “interface” headers, which are used by libraries

```
module "MathCore" {
  module “TComplex name” { header "TComplex.h" export * }
  module <name of the file> {
      header <relative path to the header file location> }
  export libMathCore.so
}
```

modulemap will contain all interface header files

Automatic generation of modulemap

C++ Modules-aware dictionary generation in CMSSW



C++ Modules in CMSSW. Preliminary measurements

C++ Modules in ROOT. Roadmap

C++ Modules in ROOT. Roadmap

- Modular ROOT and non-modular software stacks should work seamlessly but at no performance benefits (and costs)
- **ROOT 6.20:** C++ Modules will be default for all UNIX platforms (implementation based on eager module loading)
- **ROOT 6.22:** C++ Modules by default for OS X (implementation based on global module indexing (GMI))

We are here to help to migrate your stack!

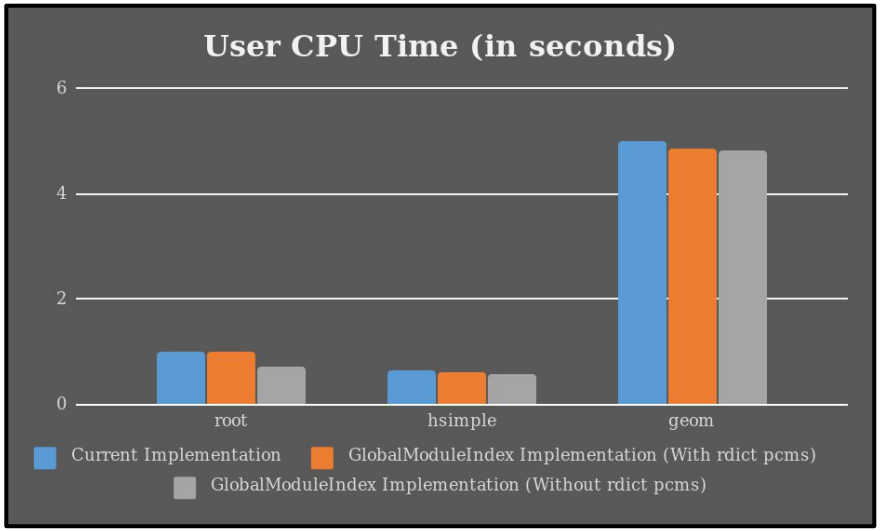
Thank you for your attention!

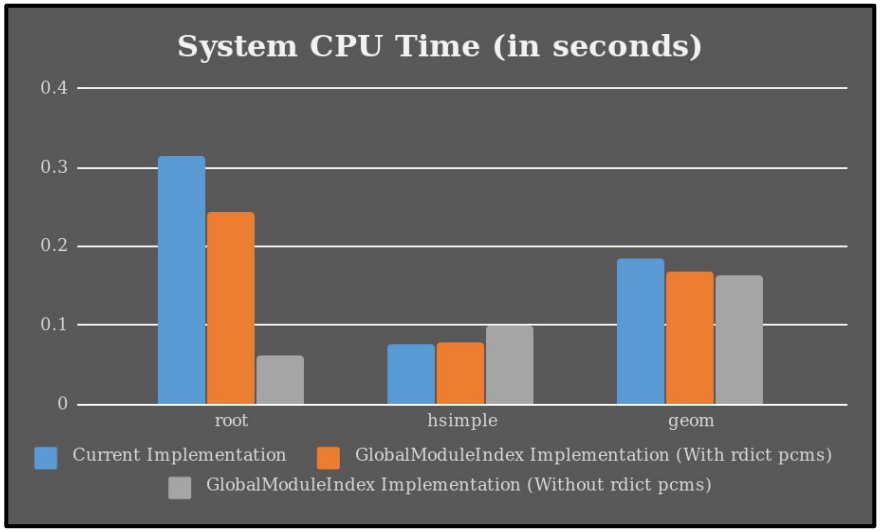
For more questions please contact: vvasilev@cern.ch
root-cxxmodules@cern.ch

We would like to thank a CMSSW dev team: **Malik Shahzad Muzaffar** and **Mircho Rodozov** for the help in integration of C++ modules in CMSSW !

Backup

CPU Time Performance Evaluation





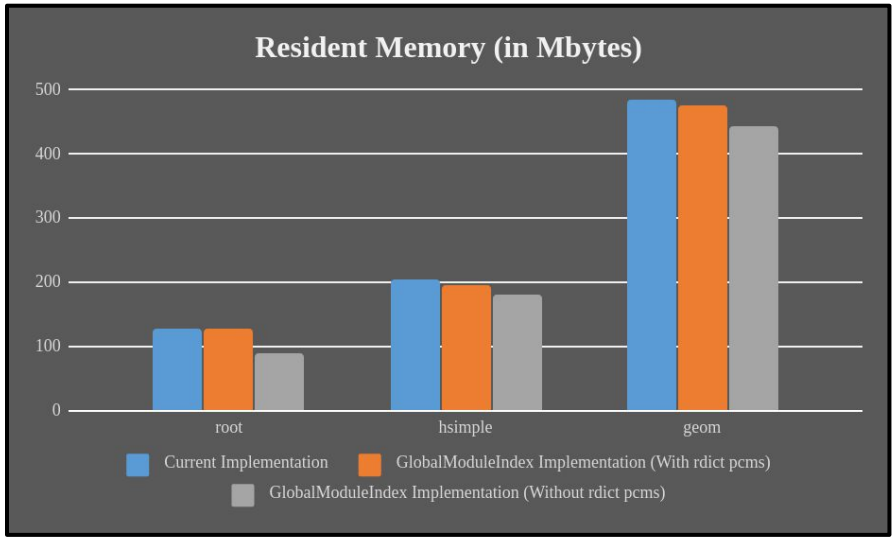
Root :	28% decrease
hsimple.C :	8% decrease
geometry.C :	4% decrease

There is an increase in System CPU time in the case of hsimple because GMI loads 30 redundant modules, which we know how to fix.

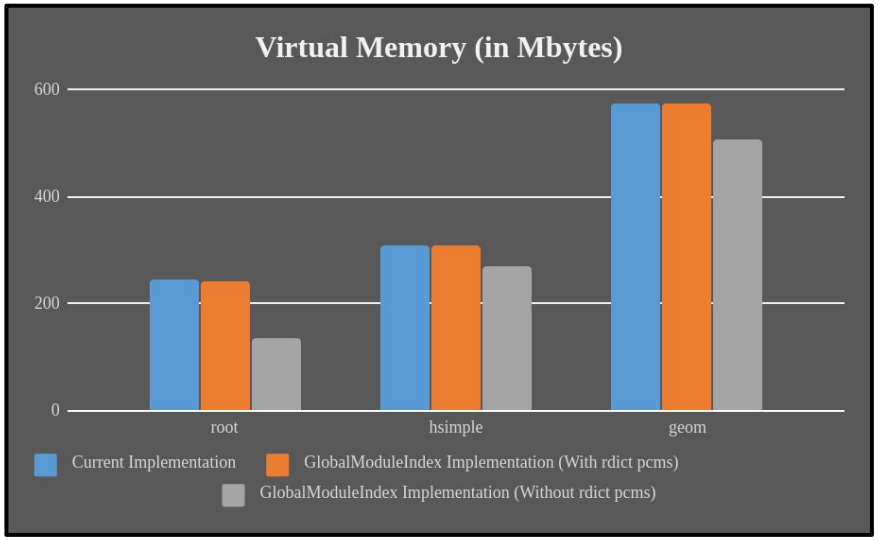
Root :	80% decrease
hsimple.C :	32% increase
geometry.C :	11% decrease

Evaluation done by GSOC student Arpitha Ragnunandan

Memory Performance Evaluation

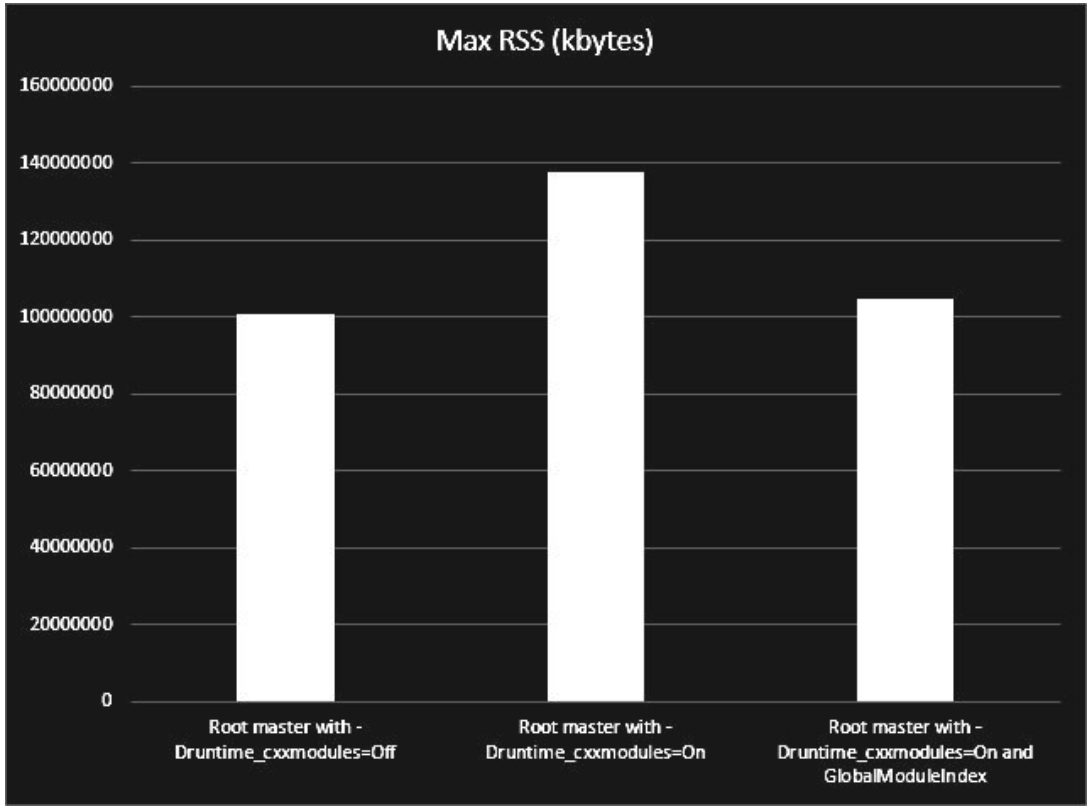


Root :	31% decrease
hsimple.C :	12% decrease
geometry.C :	9% decrease



Root :	44% decrease
hsimple.C :	13% decrease
geometry.C :	12% decrease

Memory Performance Evaluation



ROOT master with modules and GlobalModuleIndex use almost the same memory as ROOT master with PCH

The memory usage can be further decreased as currently a superset of the required modules is getting loaded

Still loading too much modules! (needs to be optimized further)

Motivation of C++ Modules

```
#include <vector>
```

Textual Include	Precompiled Headers (PCH)	Modules
✗ Expensive ✗ 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" 2 3
namespace std { inline namespace __1 {
template <bool> class __vector_base_commo
  __attribute__ ((__visibility__("hidden"),
__always_inline__)) __vector_base_common
                                        ……
# 394 "/usr/include/c++/v1/set" 3
namespace std {inline namespace __1 {
template <…> class set {
public:
   typedef _Key key_type;
                                        ……

int main {
                                        ……
```

TVirtualPad.h

vector

set

one big file!

Compile

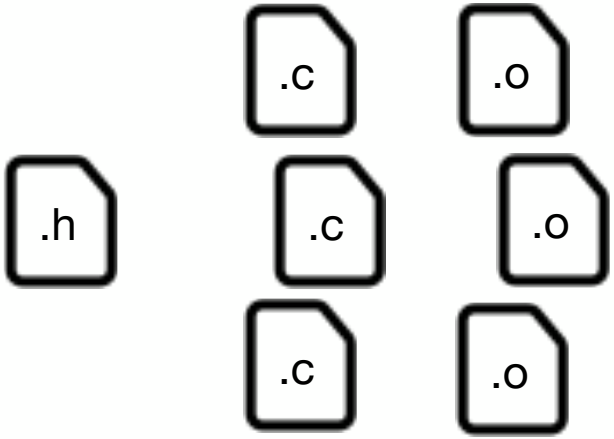
Parse

.o

Motivation of C++ Modules

Textual Include

1. Expensive
   Reparse the same header
2. Fragile
   Name collisions



Rcpp library

```
#define PI 3.14
…
```

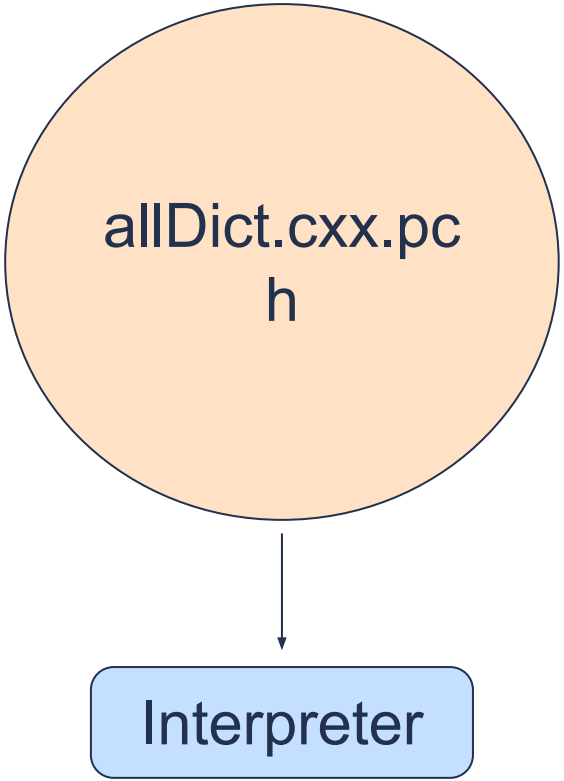
Users' code

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

Motivation of C++ Modules

PCH (Precompiled Headers)

1. Storing precompiled header information (same as modules)
2. Stored in one big file
   - Inseparable

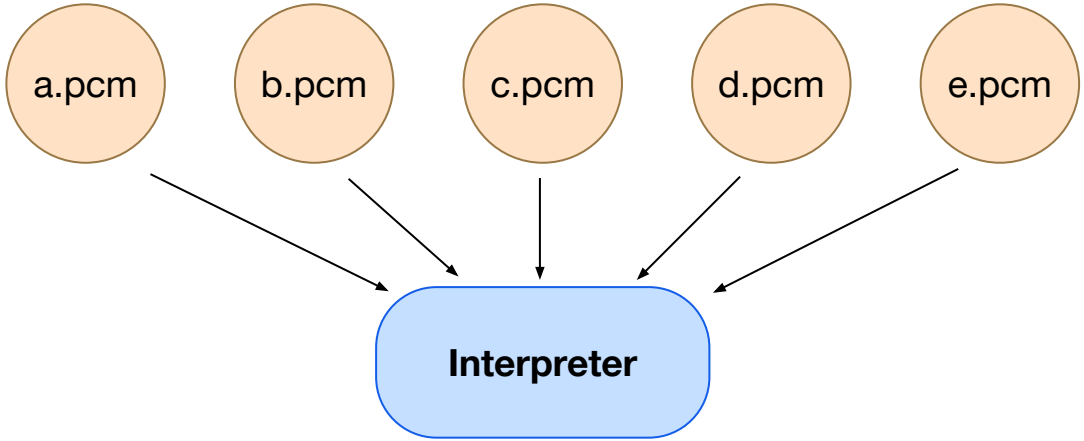


Motivation of C++ Modules

Modules

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

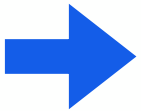
Each PCM file (a.pcm) corresponds to a library (liba.so)



Motivation of C++ Modules

Modules

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



✓ Compile-time scalability

✓ Fragility

✓ Separable

C++ Modules in CMSSW. Mechanism of the modulemap

[modulemap, modulemap overlay file, virtual modulemap overlay]