

Performance Improvements in ROOT and Status of C++ Modules

Yuka Takahashi (University of Cincinnati)
Vasil Georgiev Vasilev (Princeton University)

ROOT

Data Analysis Framework

<https://root.cern>



About myself

- ▶ Yuka Takahashi
- ▶ Affiliated with University of Cincinnati, funded by SFT until August
- ▶ Started in March and staying for a year
- ▶ Working on C++ interpreter in ROOT



- ▶ Our goal
 - Performance improvement in memory
- ▶ Today we will briefly talk about
 - Recent performance improvement in ROOT
 - Demonstrate the improvement and give a tip
 - Status of C++ modules

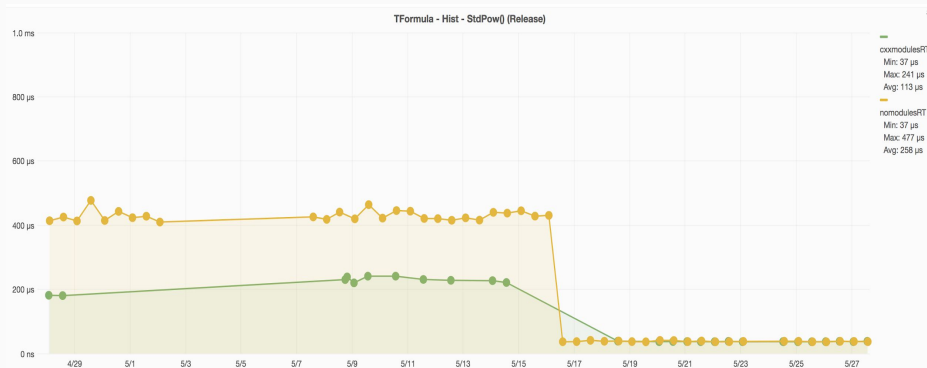
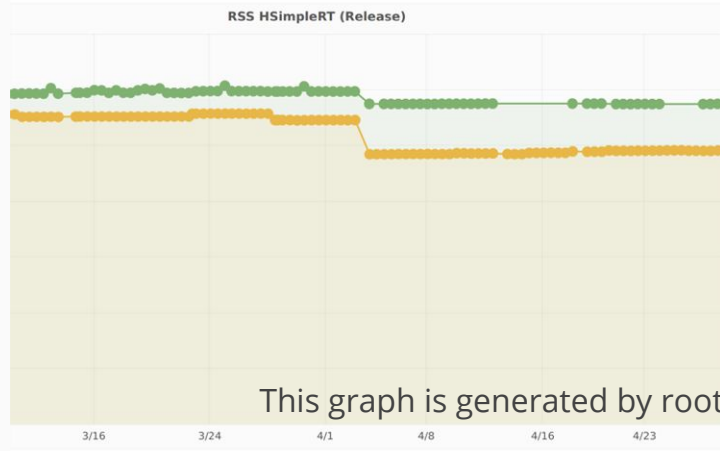


Recent performance
improvement



Recent performance improvement

- ▶ Hsimple benchmark
 - 9.2% of cpu time and 8.8% of memory improvement
- ▶ TFormula hist benchmark
 - 14% of memory improvement
- ▶ It is also visible in experiments
 - In CMS, they reported that memory dropped by 20MB on average





- ▶ Some code/headers needs to be interpreted at startup time

```
#include "cling/Interpreter/RuntimeUniverse.h"  
#include "cling/Interpreter/DynamicLookupRuntimeUniverse.h"  
namespace cling { class Interpreter; namespace runtime { Interpreter* gCling }}}
```

These headers include meta information of interpreter such as builtins, must be run before users' code

- ▶ Who was causing overhead in startup?
 - Global variables
Functions, variables, STL classes and all the functions derives from them. Some were changed to constexpr, so that it's processed at compile time
 - Eagerly Deserialized decls
These are decls which have to be deserialized from external AST source. Which cost a lot

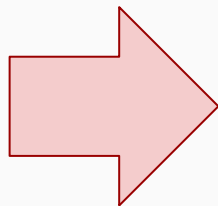


Interesting example

- ▶ Moving the first virtual function definition to cpp file improved performance significantly

```
// Foo.h  
  
class foo {  
public:  
    foo() {}  
  
    virtual ~foo() {}  
  
    virtual char* bar () { return "bar"; }  
.... }
```

```
// Foo.cpp  
  
#include "Foo.h"  
  
// using class foo below
```



```
// Foo.h  
  
class foo {  
public:  
    foo() {}  
  
    virtual ~foo();  
  
    virtual char* bar () { return "bar"; }  
.... }
```

```
// Foo.cpp  
  
#include "Foo.h"  
  
Foo::~foo() { }  
  
// using class foo below
```



Interesting example - Why?

Vtable: A table of information used to dispatch virtual functions

Key function: The first non-pure virtual function that is not inline at the point of class definition

- ▶ Itanium C++ abi “ If there is no key function, it(vtable) is emitted everywhere used”
 - In Clang, it was implemented as eagerly deserializing the decl and Vtable was emitted in every object file where the class was used
- ▶ **Don't write a function which has only inline virtual function**
 - **I recommend to pin the definition to cpp file**



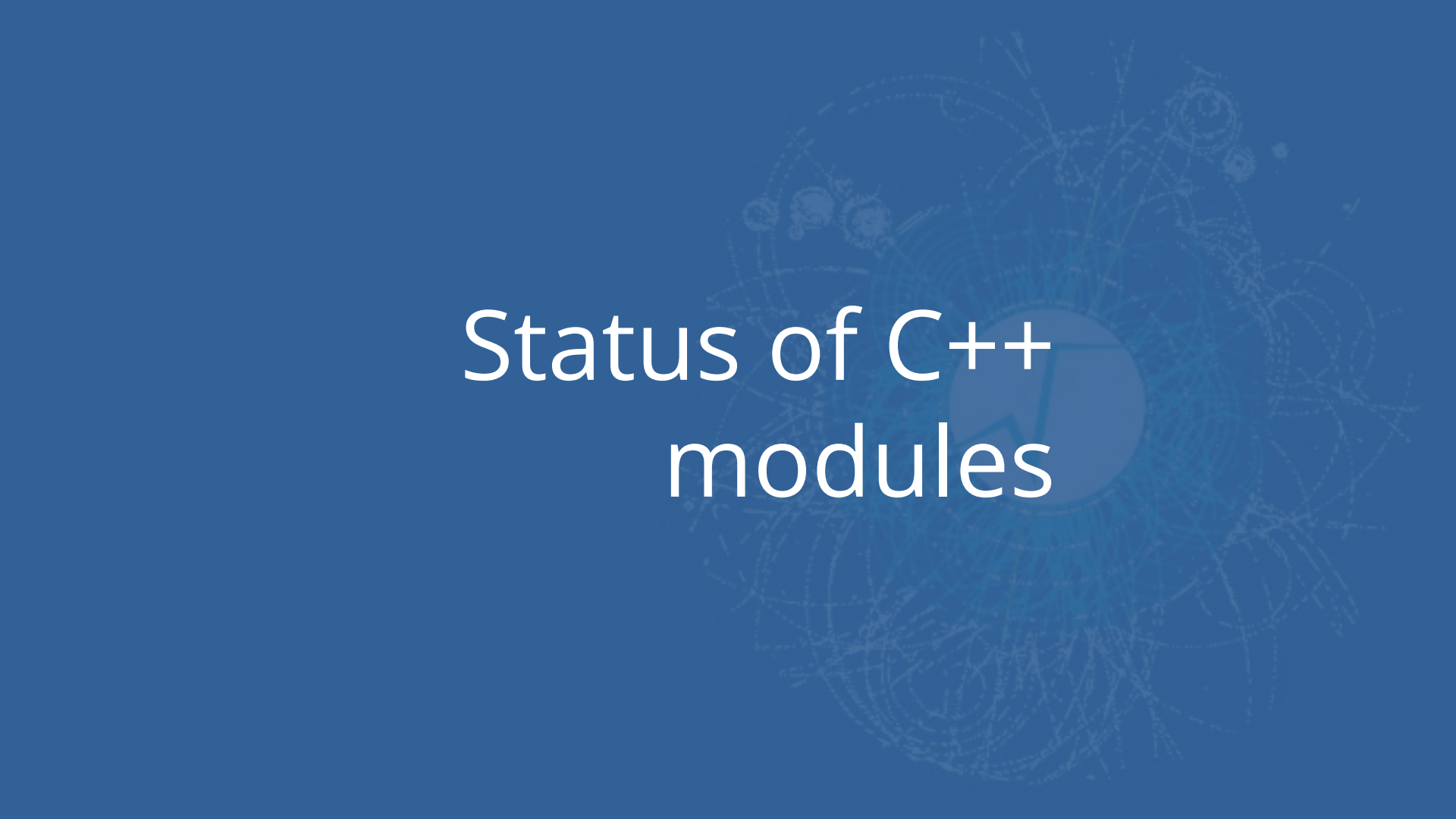
```
// Foo.h
class foo {
public:
    foo() {}
    virtual ~foo() {}
    virtual char* bar () { return "bar"; }
    .... }

```



```
// Foo.h
class foo {
public:
    foo() {}
    virtual ~foo();
    virtual char* bar () { return "bar"; }
    .... }

```

Status of C++ modules



C++ Modules - What is it?

- ▶ Generalized precompiled headers (pch)
 - NO header parsing at runtime
 - Header information is stored in pcm files
 - C++ modules give us more flexibility, and enable us to modularize experiments
 - Currently experiments are still using textual headers, because pch doesn't work for experiments
- ▶ Developed by Google, Apple in Clang
 - Open source
 - They want to make compilation time faster
- ▶ C++ modules is a mechanism to boost compilation time
 - In ROOT, compilation time turns into runtime as we're using C++ interpreter behind



Roadmap

1. Compile ROOT with C++ modules
2. Compile CMSSW with C++ modules
3. Use runtime C++ modules in ROOT
4. Use runtime C++ modules in experiments



Roadmap

1. Compile ROOT with C++ modules **Completed**
2. Compile CMSSW with C++ modules **60%**
3. Use runtime C++ modules in ROOT **95%**
4. Use runtime C++ modules in experiments **Not started**



Compile CMSSW with C++ modules

- ▶ Working with CMSSW developers
 - Their goal is to have better performance and diagnostics
 - Debugging CMSSW with developers
- ▶ Having a meeting once in two weeks

Status: 60%



Runtime C++ modules in ROOT

Overview: Correctness status 95%, Performance status 60%

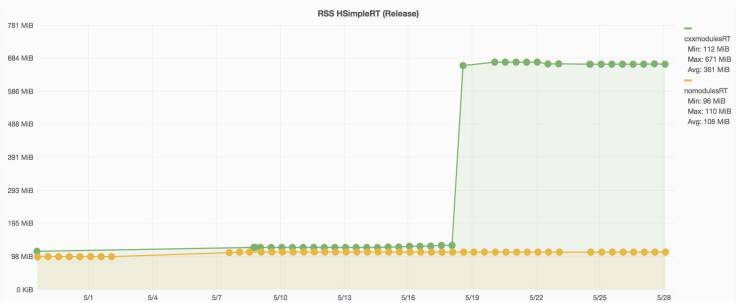
- ▶ Working, but it's not performant yet
 - We realize that it needs to be better than pch in order to get users
- ▶ Tests
 - Fixed 20+ tests, I would say runtime modules are working but tests are fragile to master changes and usually failing
- ▶ Performance
 - Needs improvement



Runtime C++ modules in ROOT

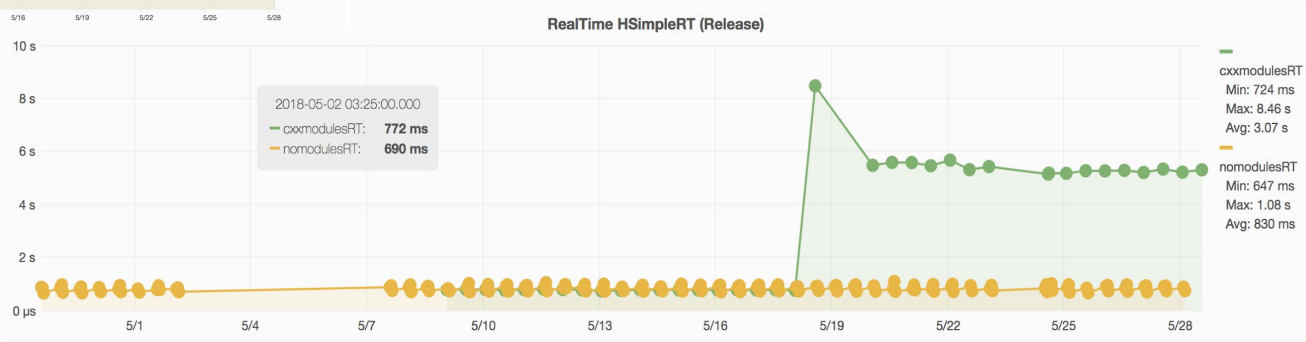
► Slow!

- Recently we had severe memory & time increase due to our new functionality



This graph is generated by rootbench

<https://rootbnch-grafana-test.cern.ch/>





Runtime C++ modules in ROOT

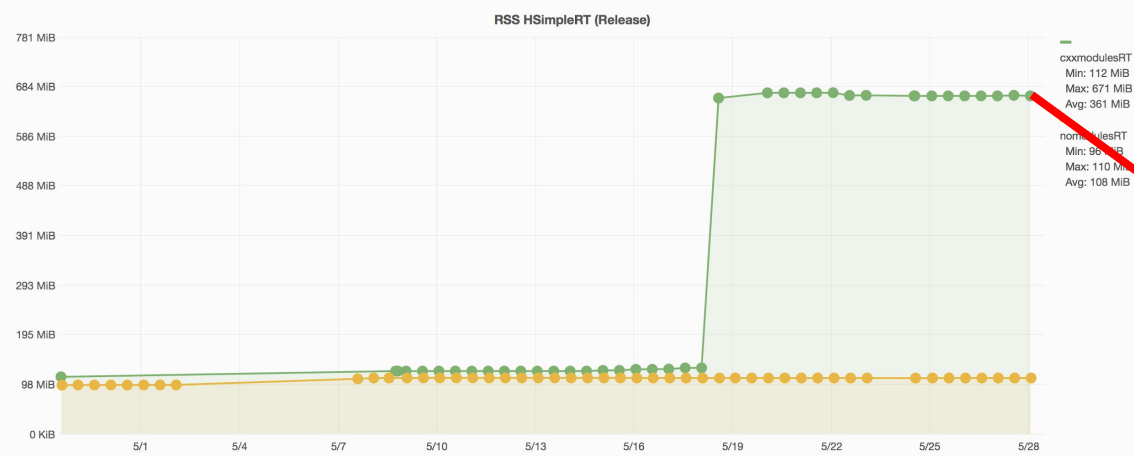
- ▶ New functionality: Preloading all modules and autoloading libraries
 - By preloading all modules, we don't need to rely on rootmap files
 - ROOT can dynamically import declarations rather than using rootmap files maintained manually by hand
 - Which gives us more correctness and fix 20+ tests out of 1650 tests
- ▶ Slow, but we already have PR which makes it 2x faster



Use runtime C++ modules in ROOT

► Summary & Future plan

- We need to focus on performance optimization
- Runtime C++ modules' correctness is already better than pch's. Performance is the work left to do



????



Thanks for your attention!