ROOT C++ modules

Vassil Vassilev and Raphael Isemann

What Are C++ Modules (PCMs)?

- Modules are a system to more efficiently **handle used libraries in C++**.
  - Replaced the old method of `#include "Header.h"` and textual inclusion.
- Work like **precompiled headers** (PCHs).
  - With less restrictions on how they can be used inside programs.
  - We only load those parts of a module that we need for the program (lazy loading).
- **Developed by companies** like Google, Apple in the clang parser.
  - Motivation for them is reducing their compilation times.
  - Collaborate and have regular meetings with us.
  - Code is open source.
- We want to use **modules in ROOT** to optimize the loading of our libraries.

Why Optimizing Library Loading?

User/Experiments' code has a lot of semantical equivalents to this.

```
// ROOT prompt (no C++ Modules):
gSystem->Load("MyLib");
// => dlopen("MyLib.so");
//    => cling->parse("1000s_of_fwd_decls.h");
MyLibClass<float> c; c.do();
// => cling->parse("#include <MyClass.h>");
```

Forces ROOT's interpreter to parse headers related to MyLib (even when we intend to use only tiny fraction of them).

This results in increased memory use and slowdown.

```
// ROOT prompt (no C++ Modules):
gSystem->Load("MyLib");
// => dlopen("MyLib.so");
//    => cling->mmap("MyLib.so.pcm");
MyLibClass<float> c; c.do();
```

C++ Modules-aware ROOT runtime will lazily allocate memory only for what you use and at the point of use!

Everything unused is mmaped.

Slide from Vassil's CHEP 2016 talk

Profiling example 1 - Using ROOT headers

```
#include "THtml.h"
#include "TTree.h"
#include "TLorentzVector.h"

// Definitions to actually require the #includes

THtml h;
TTree t;
TLorentzVector l;
```

Profiling example 2 - Using EVE library

Without modules

```
class __attribute__… TEveShape;
class __attribute__… TEveShapeEditor;
class __attribute__… TEveLine;
// ...

#include "TEvePlot3DGL.h"
TEvePlot3DGL a;
```

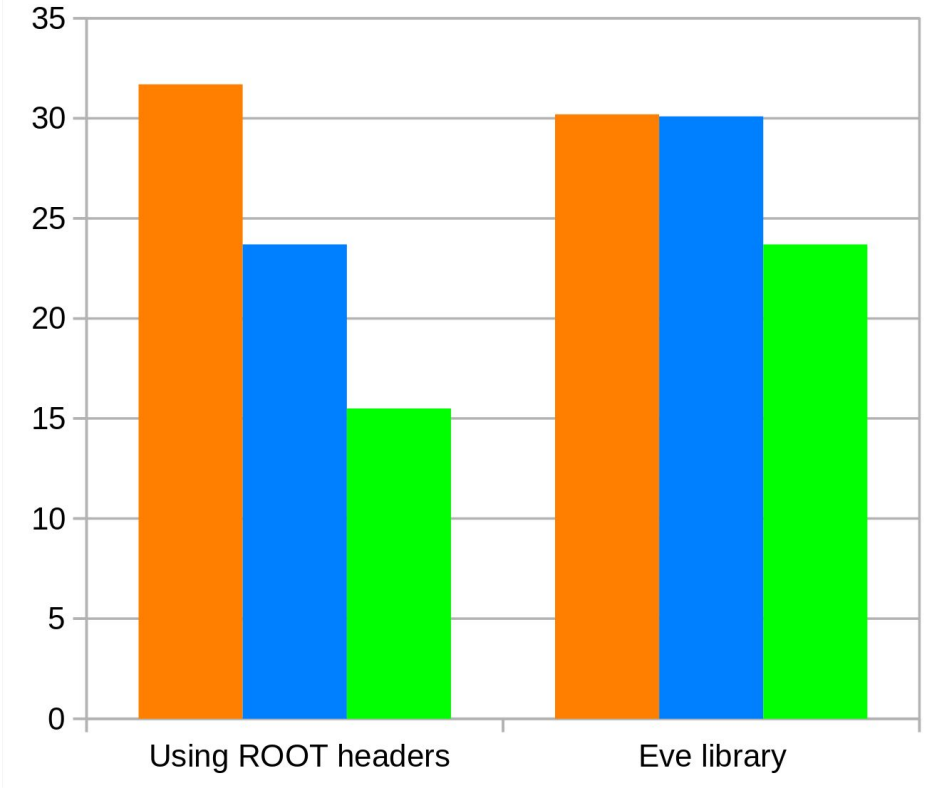
With modules

```
#include "TEveShape.h"
#include "TEveShapeEditor.h"
#include "TEveLine.h"
// ...

#include "TEvePlot3DGL.h"
TEvePlot3DGL a;
```

Memory Consumption with/without C++ Modules

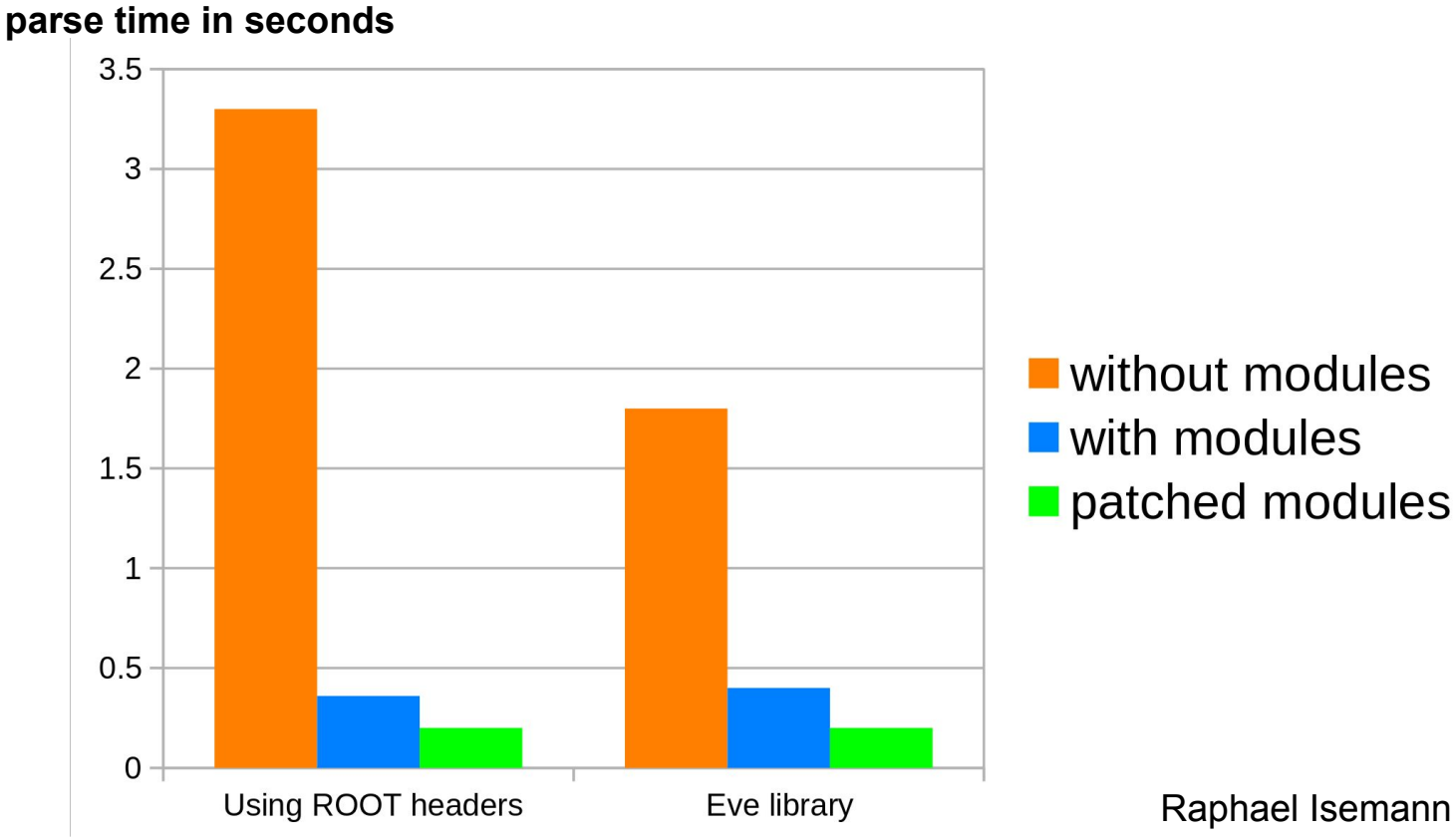
memory of parsed AST in MiB



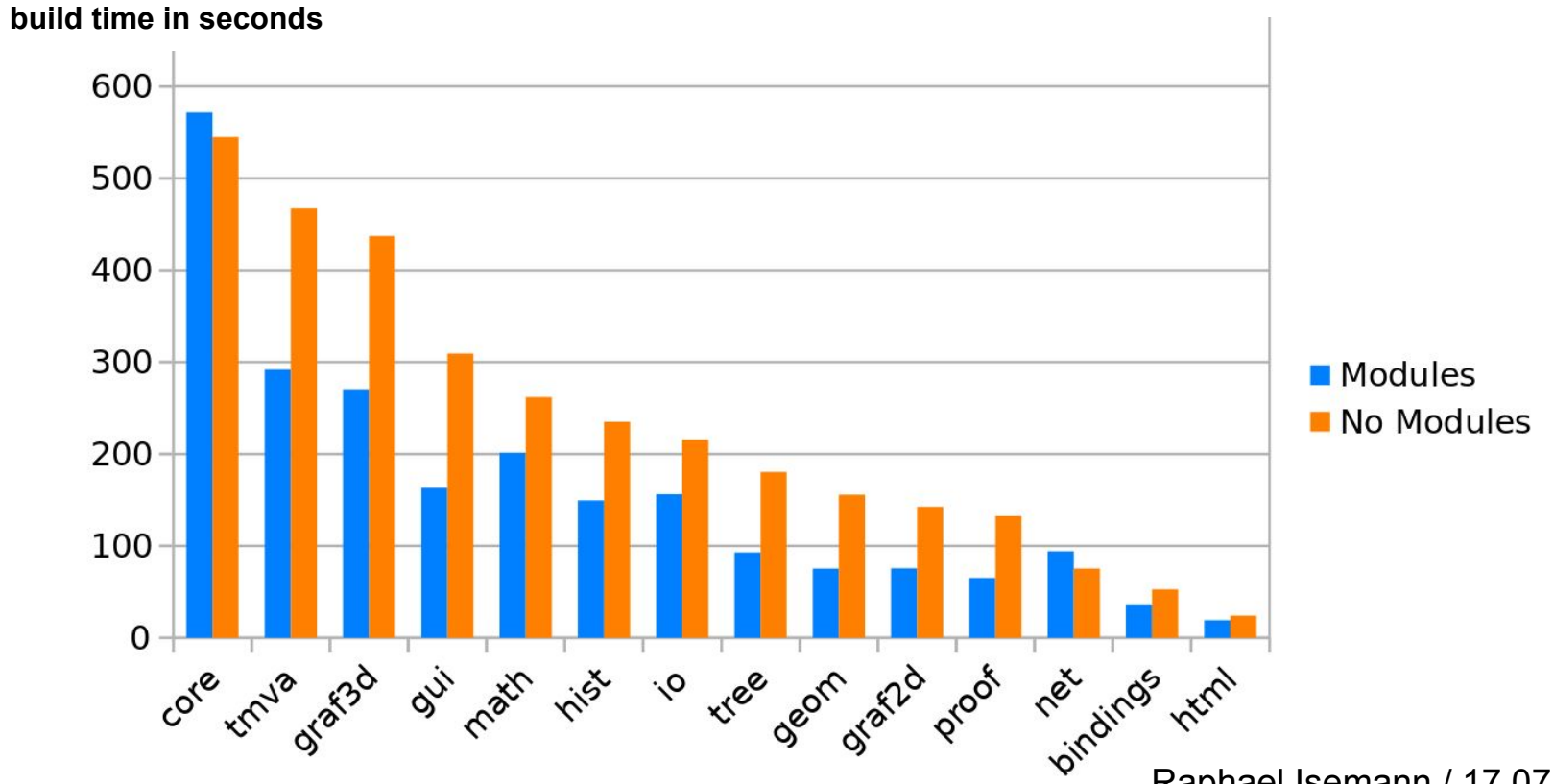
- without modules
- with modules
- patched modules

(patches modules =>
We have a upcoming patch for optimizing memory consumption when using templates in modules)

Parse Time with/without C++ Modules



ROOT Compilation Time with Modules



Status: Moving ROOT and CMSSW to Modules

1. Compile ROOT headers into C++ modules - **completed**
2. Compile CMSSW headers into C++ modules - **60% completed**
3. Teach rootcling to produce C++ modules - **mostly done**
4. Teach ROOT interpreter to load modules - **prototype works**
5. Compile C++ modules for CMSSW with rootcling - **blocked by no. 3**
6. Use C++ modules for CMSSW runtime - **blocked by no. 5**

Compile headers in CMSSW into C++ modules

- ~800/1300 modules compile with normal clang
  - The remaining 500 either don't compile because of blocking problems in their headers or because one of their dependencies doesn't compile (e.g. because of missing includes).
- 132 merged commits to CMSSW so far, 2000 lines diff
  - Mostly fixing non-standalone headers, but also repairing broken code that the compiler didn't see without modules and resolving layering violations.
  - All problems that would have also affected normal builds at some point.
  - PRs tracked in this issue: https://github.com/cms-sw/cmssw/issues/15248
- 3 patches to the module/serialization system of clang regarding CMSSW
  - see D33366, D32499, D30793
- Started to generate PCMs/provide patches for used libraries.
  - HepMC, HepMC3, boost, xerces-c, CLHEP, etc. ...

Teach rootcling to produce C++ modules

- rootcling performs transformations on the code that we need in the PCMs.
  - E.g. transforming comments into annotations on the respective field/class.
  - Without this parts like IO don't work, so we have to teach rootcling to produce PCMs.
- rootcling now requires that all referenced PCMs that require reflection information are built beforehand.
  - rootcling references other modules when generating a module, so we need to built every PCM that is referenced beforehand.
  - This also meant that we had to start fixing the dependencies/race conditions in ROOT's build system (You can now do `make -j200`)!
- Can build external all PCMs that don't require reflection information as a side product when running rootcling.
- We also need a clang modulemap that contains information about all PCMs.
  - Simple text file that maps between headers and the module they belong to.

Example modulemap:

```
module CMS_DataFormats_Version {
  module “Version.h” {
    header "Version.h"
    export *
  }
  module “OtherHeader.h” {
    header "OtherHeader.h"
    export *
  }

  // ...more headers follow...
}
```

Teach ROOT interpreter to load modules

- Looks for PCMs in the usual ROOT paths (LD_LIBRARY_PATH, …)
  - `LD_LIBRARY_PATH, DYLD_LIBRARY_PATH,` etc.
- Needs the same modulemap as used during the rootcling invocation.
  - To map between an `#include <header>` to the specific module for this header.
  - And to know which modules are available.
- Also links the appropriate shared library when a module is used.
- Right now just eagerly loads/links all available modules.
  - Will be optimized once we can build proper PCMs with reflection information.

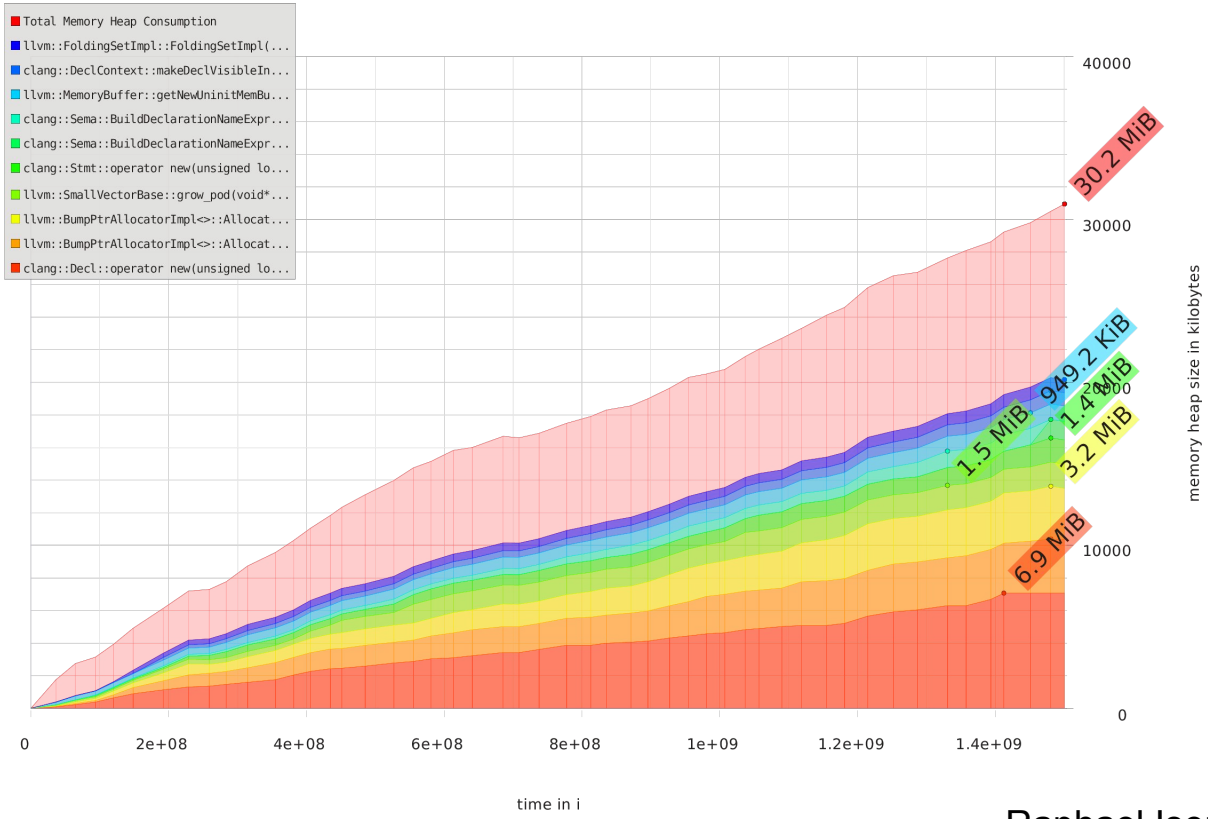
Compile C++ modules for CMSSW with rootcling

- Set/export environment variable ROOT_MODULES to “1”
  - Can also be set to “DEBUG” and rootcling/ROOT will print debug information.
- Generate a modulemap for CMSSW
  - scram2cmake already generates one that can be used.
  - We can also make a small script that generates the modulemap...
- Generate PCMs with rootcling
  - Depending PCMs needs to be build first, so SCRAM needs to handle those dependencies.
  - All PCMs that don’t need reflection information (boost, HepMC, etc.) will be built as a side-product when used for the first time from a rootcling invocation.
  - Clang handles the dependencies to those other PCMs.

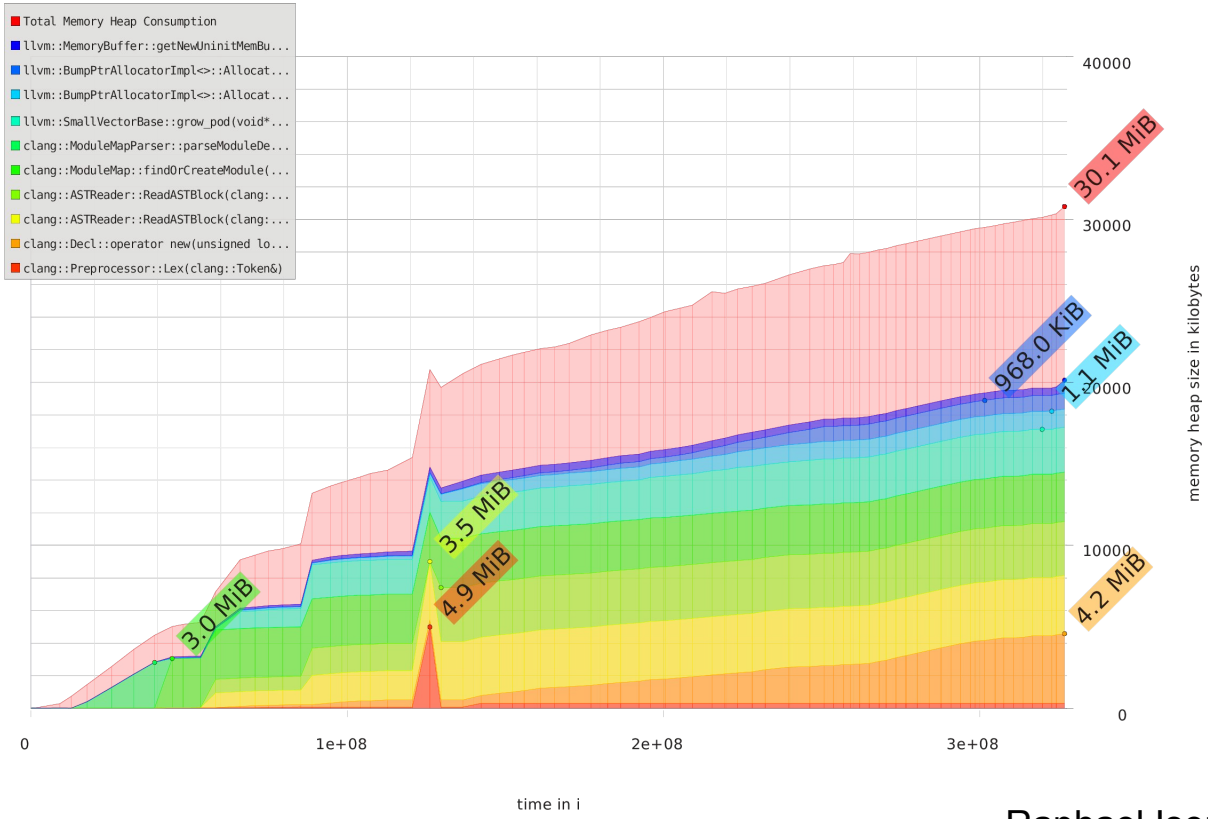
Thanks!

Questions?

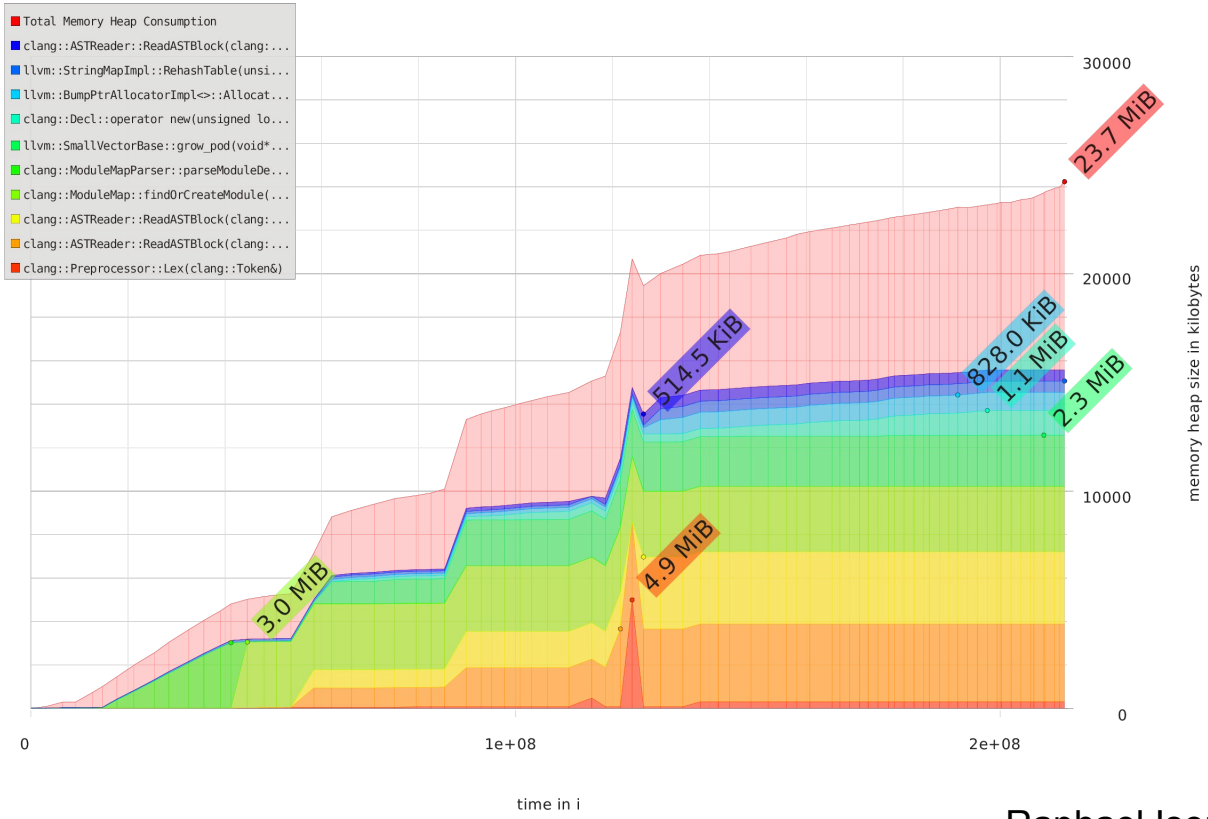
Memory usage - No modules - Eve library



Memory usage - Modules - Eve library



Memory usage - Patched modules - Eve library



Textual inclusion in C++

```
/* foo.h */
int foo(int a);
```

```
/* main.cpp */
#include "foo.h"

int main(int argc, char **argv) {
  return foo(3);
}
```

Textual inclusion in C++

```
/* main.cpp.m after preprocessing*/
int foo(int a);


int main(int argc, char **argv) {
  return foo(3);
}
```

Textual inclusion in C++

```
/* preprocessed main.cpp.m */
int foo(int a); // <- will be parsed for every compilation!


int main(int argc, char **argv) {
  return foo(3);
}
```