C++ modules in ROOT

Vassil Vassilev and Raphael Isemann

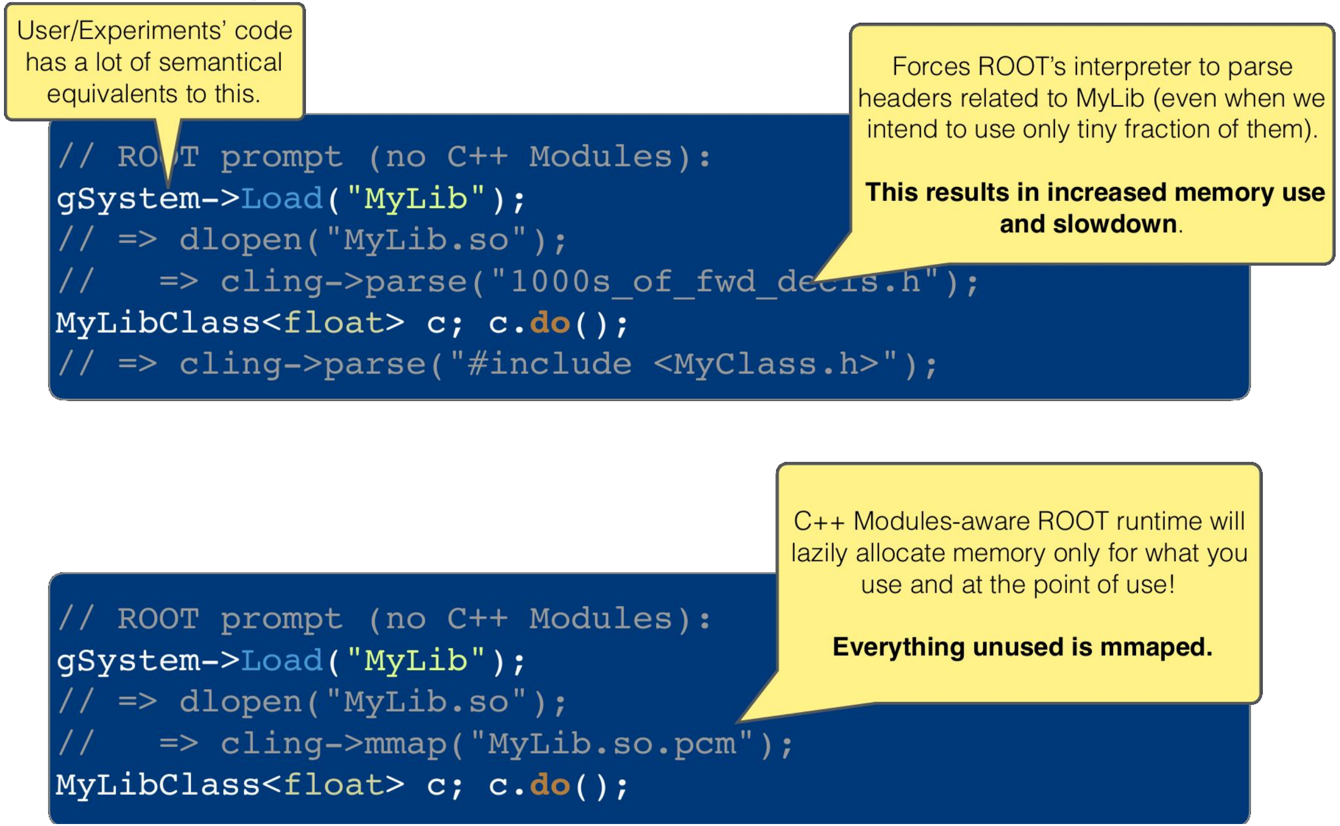
About me: Raphael Isemann

- Technical student working for CMS in the SFT group.
- Currently studying for Master of Computer Science @ Chalmers University.
- Previously Google Summer of Code student at LLVM/Apple.
- Here at CERN since February.
- Working on moving CMS/ROOT to C++ modules.

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!

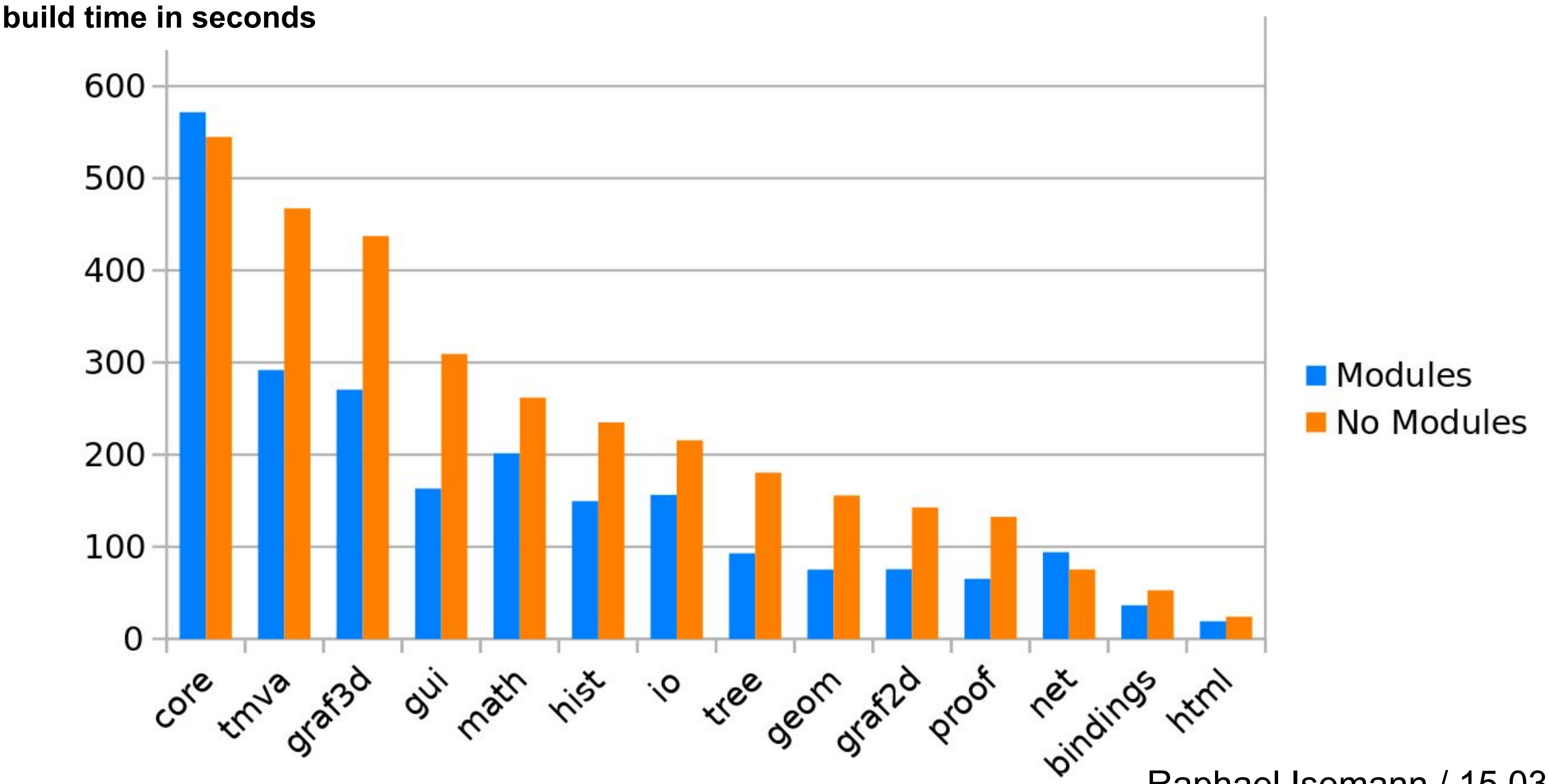
Everything unused is mmaped.

Slide from Vassil's CHEP 2016 talk

Adoption plan for C++ modules in ROOT

1. **Use modules to compile ROOT.** ✓
    - Compiling ROOT with modules to test if they can handle the codebase.
    - Works in the ROOT nightly builds.
    - Impact so far:
        - Improved code quality in parts of ROOT’s code base.
        - Reduced ROOT compilation times.

ROOT compilation time with modules



Raphael Isemann / 15.03.2017

Adoption plan for C++ modules in ROOT

1. Use modules to compile ROOT. ✓
2. **Use modules to optimize ROOT’s runtime.** 🕒
   - This is scheduled next.
   - Provide support for rootcling (genreflex) to build PCMs and load them in ROOT.
   - We made a few tests to estimate the possible performance gains...

Estimating performance of modules in ROOT

- ROOT uses the same parsing/AST as clang.
    - And modules already when compiling with clang.
- So we profiled clang's parsing code to estimate ROOT's performance when parsing.
- We currently miss an memory optimization in clang.
    - Because clang loads redundant template specializations.
    - There is a short patch to fix this (at least for the examples we profile).
    - We refer to the parsing with this optimization as "patched modules".

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

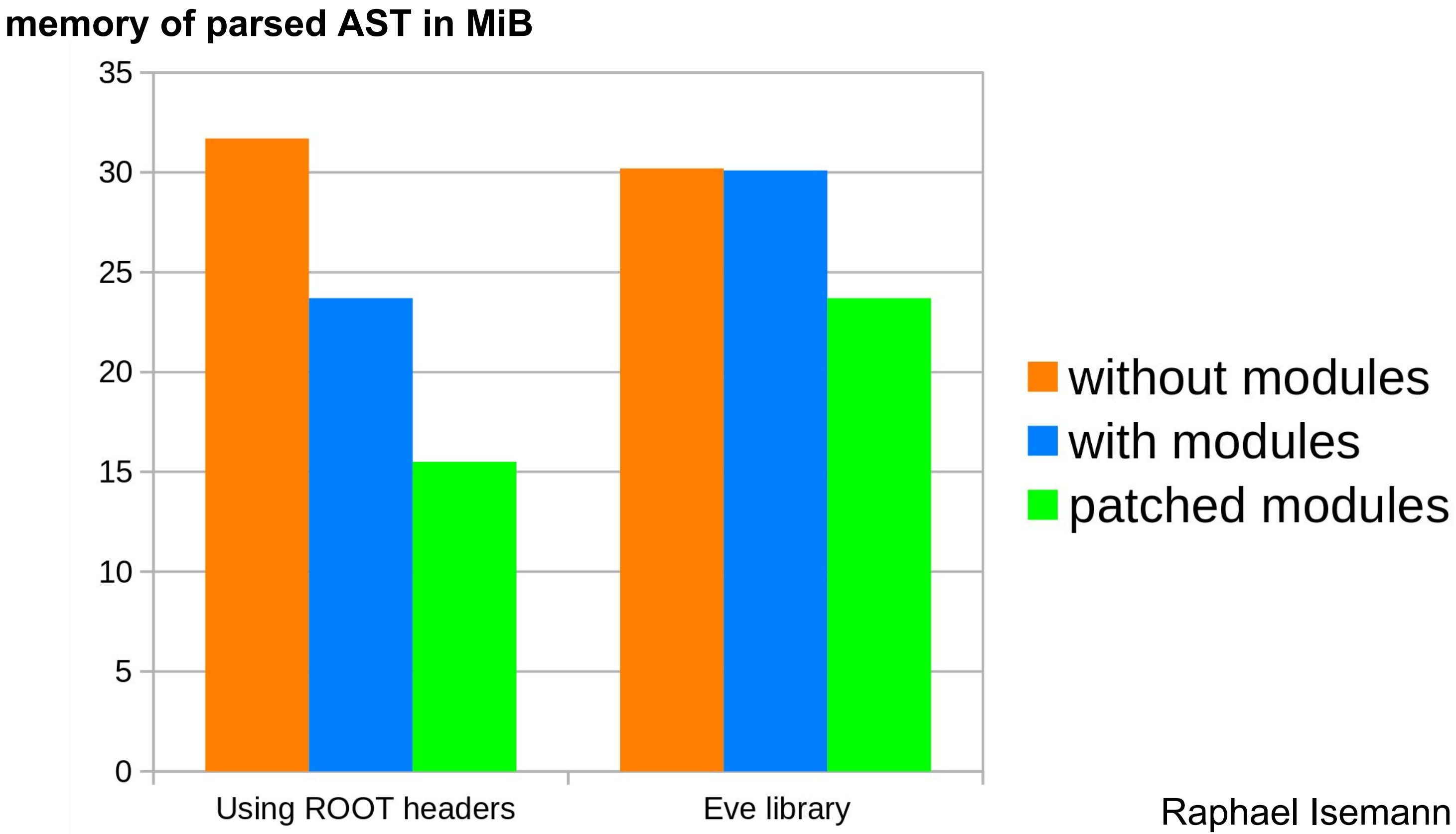
Without modules

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

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

With modules

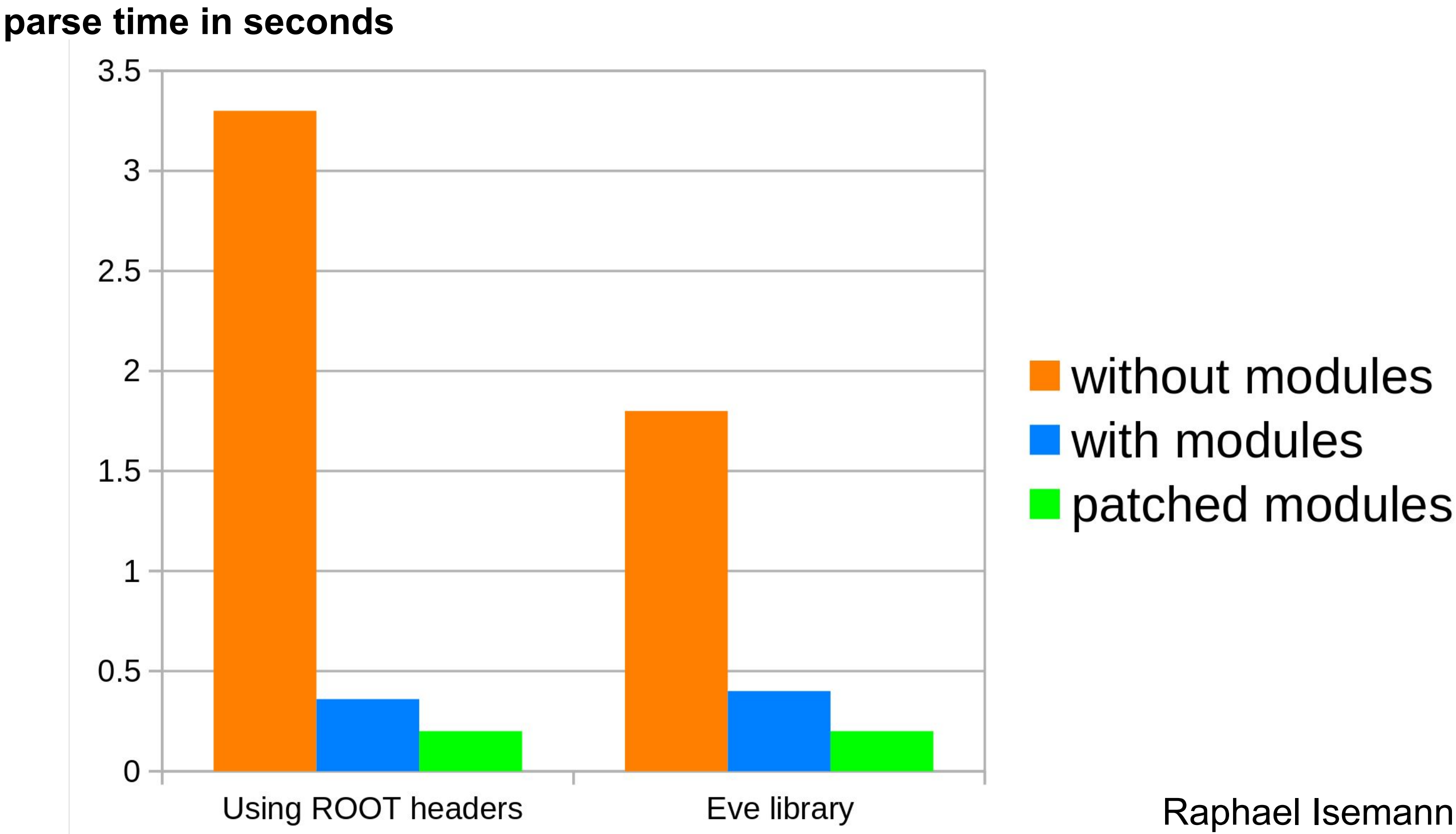
```
#include "TEveShape.h"
#include "TEveShapeEditor.h"
#include "TEveLine.h"
// ...

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

Memory consumption with/without modules



memory of parsed AST in MiB

Parse time with/without modules



Possible performance gains with modules

- Estimates from the tests we just seen:
    - 5-10 times faster loading of libraries.
    - 5-25% less memory consumption from loaded libraries **now.**
    - 20-40% less memory consumption from loaded libraries **planned.**
    - Further optimizations in the future...

Future optimizations in modules

- Google has 100 Million lines of code already compiling with modules.
  - Likely that they will continue investing into this feature.
- Google, Apple mostly want to **optimize time, not memory.**
  - But we observe that **memory consumption is proportional** to runtime.
  - => Future speed optimizations will probably also reduce memory usage.
- Once we moved to modules, we get future optimizations for free!
  - Optimizations happen behind the scenes in the module implementation.
  - We probably get them all without doing any changes to ROOT/experiments.

Adoption plan for C++ modules in ROOT

1. Use modules to compile ROOT. ✓
2. Use modules to optimize ROOT’s runtime. 🕒
3. **Use this ROOT feature in CMS/other experiments.** 🕒
    - We already started preparing for this.
    - Making patches for modules implementation to handle the codebase.
        - We fixed two issues in the modules implementation so far in collaboration with the developers from Google and Apple.
        - Bug 32186 and Review D30496.
    - Making CMS codebase compatible with modules.

Making code compatible with modules

1. Changes are in general fixing minor implementation bugs:
   - Having all **headers self-contained**.
   - **No circular dependencies** between libraries (between headers in the same library is OK).
2. Only requires minimal code changes for the experiments:
   - Modules often require no further changes in modern C++ code.
   - For CMS we have so far a 10 line diff to compile FWCore with modules: PR17943.
   - Changes are all adding missing includes/removing unnecessary includes.
   - The configuration is done in an external modulemap file.
3. Available tools help with finding/fixing those issues:
   - Clang itself: Can point out what headers to include (or directly include them for you).
   - “Modularize”: Checks for violations of the One-Definition-Rule, generates modulemap files.

Future work for next months

1. Making all of CMS compile with modules.
2. Fixing few remaining bugs in the module implementation and bring them upstream.
3. Working on the template specialization patch.

Thanks!

Questions?

Raphael Isemann / 15.03.2017

FAQ

- Q: Will modules force us to a certain compiler/vendor?
  - A: No, in a production ready environment PCMs will be provided by rootcling.

- Q: What are the mechanics of modules? They work the same way as #include? E.g. making all globals available.
  - A: In the current implementation they do. But that depends on the modulemap.

- Q: How do we handle autoloading? Do we still need forward declarations?
  - A: The modules implementation should do this for us.

Some open questions:

- Q: How do we handle autoloading? Do we still need forward declarations?
  - A: The modules implementation should do this for us. But we will see what is more efficient. Suggestions welcome.

- Q: How do we handle modules across different systems? E.g. different OS versions of SLC6.X?

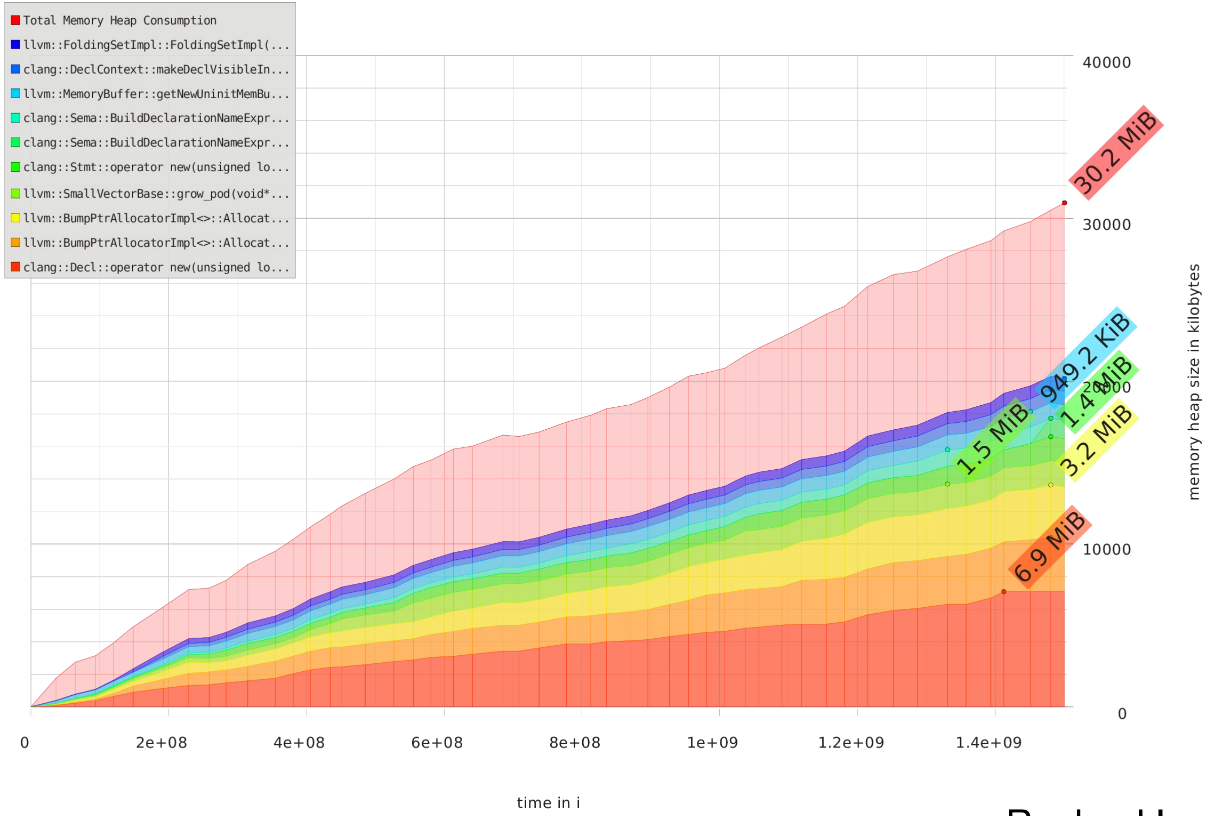
- Q: Should we replace all PCH with PCMs?

Backup slides

Next slides:

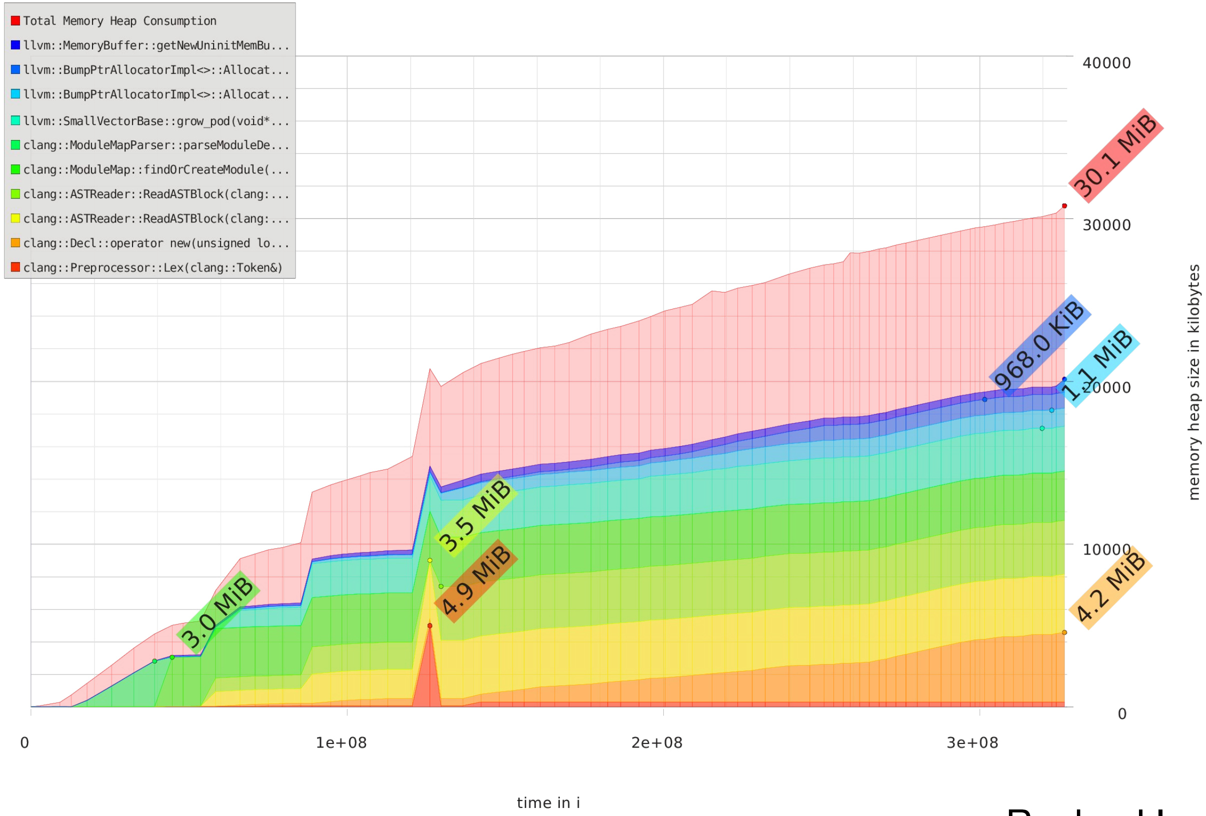
Memory usage when running clang over Eve library

Memory usage - No modules - Eve library

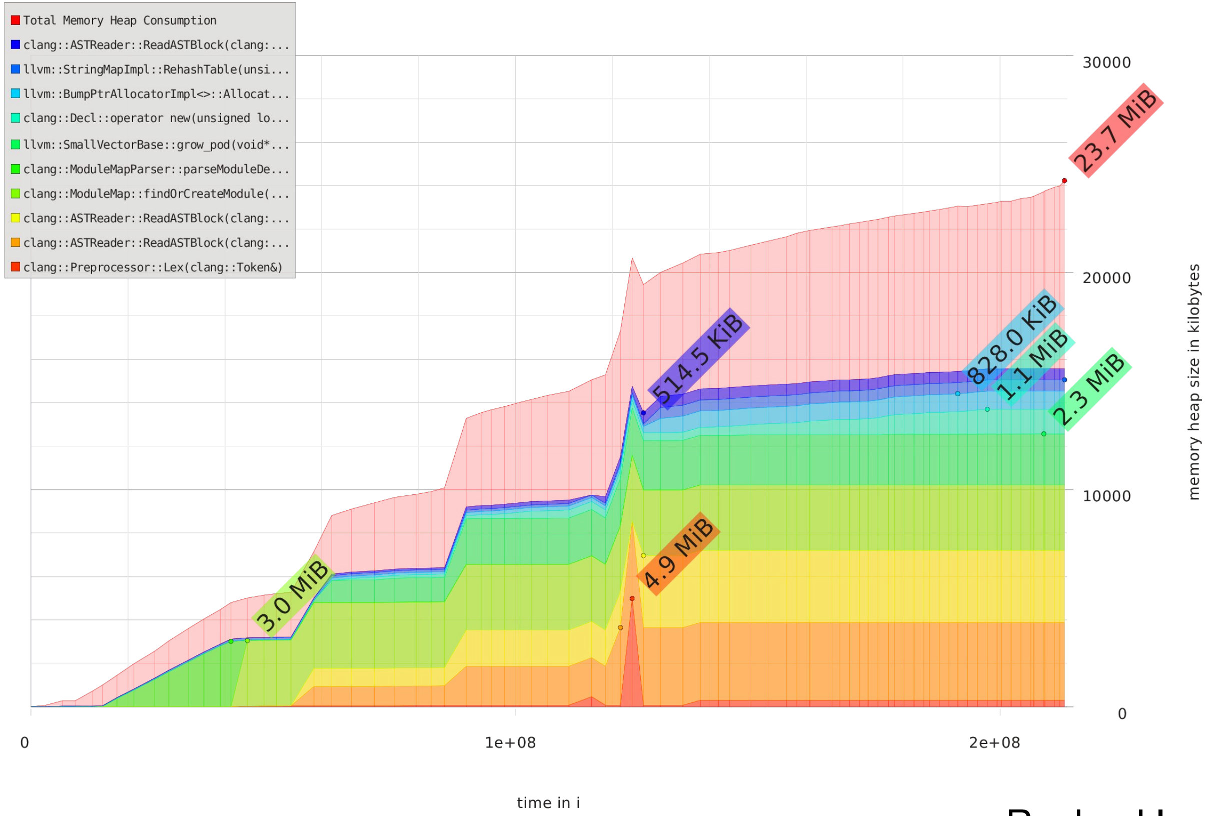


Raphael Isemann / 15.03.2017

Memory usage - Modules - Eve library



Memory usage - Patched modules - Eve library



Raphael Isemann / 15.03.2017

Textual inclusion in C++

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

```cpp
/* 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);
}
```