The Rough Road Towards Upgrading to C++ *Modules*

Richárd Szalay @Whisperity,
Zoltán Porkoláb

Eötvös Loránd University
Faculty of Informatics
Dept. of Prog. Lang. & Compilers

2019. 05. 10.

Research effort supported by the ÚNKP-18-2 “New National Excellence Program” of the Ministry of Human Capacities.
Publication supported by the European Union, co-financed by the European Social Fund in project EFOP-3.6.3-VEKOP-16-2017-00002.

Emberi Erőforrások Minisztériuma

SZÉCHENYI 2020

Magyarország Kormánya

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE

Using third-party solutions in our code

After some download & build configuration…

- Java: `import org.apache.hadoop.*;`
- Python: `import networkx as nx`
- Fortran (≥ 90): `use opengl_gl`

Using third-party solutions in our code

After some download & build configuration...

- Java: `import org.apache.hadoop.*;`
- Python: `import networkx as nx`
- Fortran (≥ 90): `use opengl_gl`
- C++:

```
#include <boost/fiber/bounded_channel.hpp>
#include <boost/filesystem/path.h>
#include <complex>
```

Using third-party solutions in our code

After some download & build configuration…

- Java: `import org.apache.hadoop.*;`
- Python: `import networkx as nx`
- Fortran (≥ 90): `use opengl_gl`

- C++:

```
#include <boost/fiber/bounded_channel.hpp>
#include <boost/filesystem/path.h>
#include <complex>
```

  - and also:

```
g++ a.o -lboost_fileystem@1.58.0 -o a.out
```

Using third-party solutions in our code

After some download & build configuration…

- Java: `import org.apache.hadoop.*;`
- Python: `import networkx as nx`
- Fortran (≥ 90): `use opengl_gl`

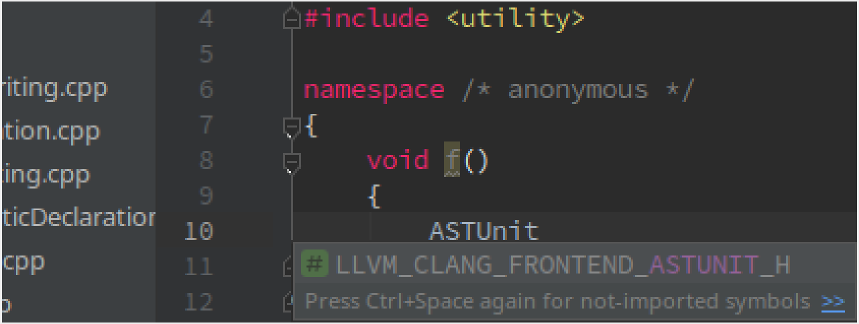
- C++:

```
#include <boost/fiber/bounded_channel.hpp>
#include <boost/filesystem/path.h>
#include <complex>
```

  - ~~and also:~~
    ~~`g++ a.o -lboost_fileystem@1.58.0 -o a.out`~~

Using third-party solutions in our code

Assuming a set environment, good build configuration and IDE…



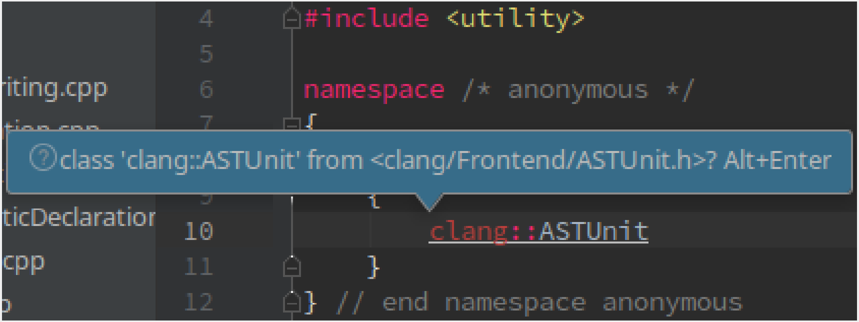
```
#include <utility>

namespace /* anonymous */
{
    void f()
    {
        ASTUnit
```

# LLVM_CLANG_FRONTEND_ASTUNIT_H

Press Ctrl+Space again for not-imported symbols >>

Using third-party solutions in our code

Assuming a set environment, good build configuration and IDE...

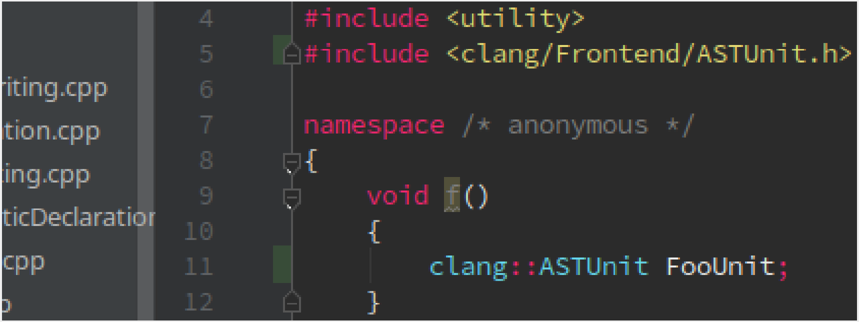


```
#include <utility>

namespace /* anonymous */
{
    {
        clang::ASTUnit
    }
} // end namespace anonymous
```

ⓘ class 'clang::ASTUnit' from <clang/Frontend/ASTUnit.h>? Alt+Enter

Using third-party solutions in our code

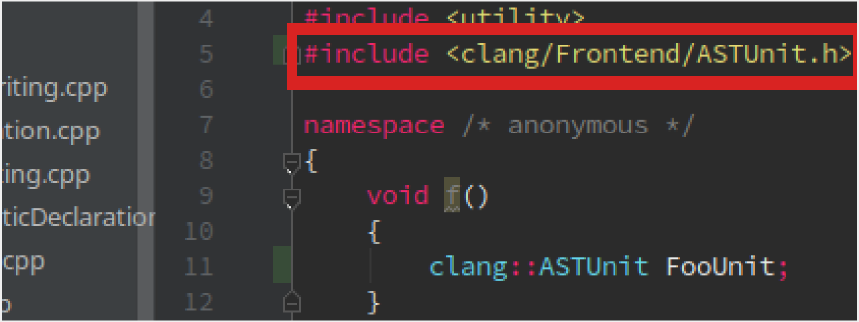
Assuming a set environment, good build configuration and IDE…



```cpp
#include <utility>
#include <clang/Frontend/ASTUnit.h>

namespace /* anonymous */
{
    void f()
    {
        clang::ASTUnit FooUnit;
    }
```

Using third-party solutions in our code

Did I subtly break something?



```cpp
#include <utility>
#include <clang/Frontend/ASTUnit.h>

namespace /* anonymous */
{
    void f()
    {
        clang::ASTUnit FooUnit;
    }
```

Outline of This Talk

1. C++'s Compilation Model
   - Software technology issues
   - Performance drawbacks and possible solutions

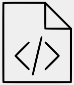
2. C++ Modules
   - "How it should work?"
   - Traps and pitfalls (even) with Modules

3. The Wish for Automatic Modularisation
   - Formal overview
   - Case study — Apache Xerces
   - Evaluation of findings
   - Requirements for upgrading to Modules

4. Summary

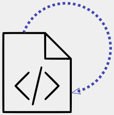
Compilation Model

1. Compiler invoked with a **single** source file.



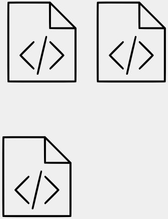
Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed



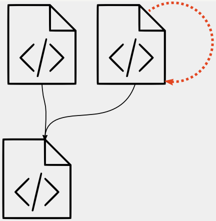
Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed
3. Included headers located



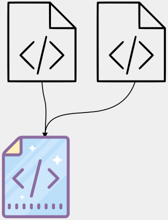
Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed
3. Included headers located
4. Included headers transitively preprocessed



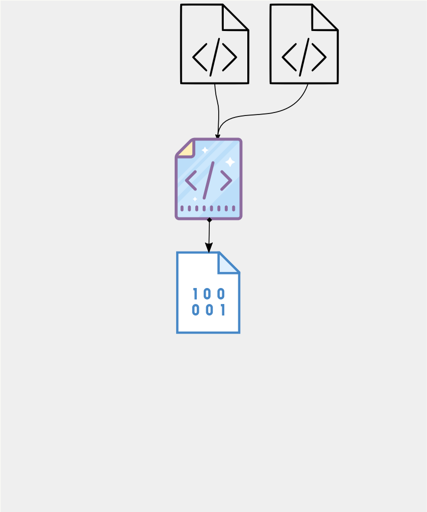
Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed
3. Included headers located
4. Included headers transitively preprocessed
5. The input buffer is complete.



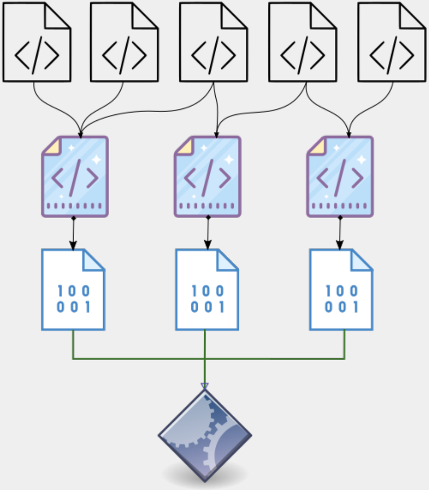
Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed
3. Included headers located
4. Included headers transitively preprocessed
5. The input buffer is complete.
6. Template instantiation, code generation



Compilation Model

1. Compiler invoked with a **single** source file.
2. Source file preprocessed, directives such as `#define`, `#include` executed
3. Included headers located
4. Included headers transitively preprocessed
5. The input buffer is complete.
6. Template instantiation, code generation
7. Generated objects linked into binary



“Faces”, “Metrics” and “Goals” around Development

- Good (?) code
- Often easily written
- Almost always well readable
- Good run-time performance
- Stable behaviour, tested, …

- Good tooling:
  - Build (and package?) management
  - Static analysis, coverage
  - Code comprehension
- Solid releases, nightlies
- Easy, fast incremental development

“Faces”, “Metrics” and “Goals” around Development

- Good (?) code
- Often easily written
- Almost always well readable
- Good run-time performance
- Stable behaviour, tested, …

- Good tooling:
  - Build (and package?) management
  - Static analysis, coverage
  - Code comprehension
- ~~Solid releases, nightlies~~
- ~~Easy, fast incremental development~~

Issues with token leak

header.hpp

```
#define APP_DATE __DATE__         /*   Build   date */
```

main.cpp

```
int main() { std::cout << APP_DATE << std::endl; }
```

Issues with token leak

header.hpp

```
#define APP_DATE __DATE__         /*   Build   date */
```

main.cpp

```
int main() { std::cout << APP_DATE << std::endl; }
```

lib.hpp

```
#define APP_DATE "2017. Oct. 20." /* Licensing date */
```

lib.cpp

```
const char* LicenseStartDate() { return APP_DATE; }
```

Issues with token leak/single input buffer

header.hpp

```
#define APP_DATE __DATE__          /*   Build   date */
```

lib.hpp

```
#define APP_DATE "2017. Oct. 20." /* Licensing date */
```

main.cpp

```
#include "header.hpp"
#include "lib.hpp"
int main() {
    std::cout << LicenseStartDate() << '/'
              << APP_DATE << std::endl;     }
```

Issues with name leak

header1.hpp

```
namespace A {
namespace   {
    inline int detail() { return 1; }
}
    class X { /* ... */ };
}
```

header2.hpp

```
namespace A {
namespace   {
    inline int detail() { return 2; }
}
    class Y { /* ... */ };
}
```

Issues with subtle breaking changes

client.cpp

```
struct B, D;

int f(const void* vp) { return 1; }
int f(const B*    bp) { return 0; }

int test(D* dp) [[ensures t: t == 1]]
{
    return f(dp); // resolved as 'f(const void*)'
}
```

Source: google.github.io/styleguide/cppguide.html#Forward_Declarations

Issues with subtle breaking changes

types.hpp

```
struct B     { /* ... */ };
struct D : B { /* ... */ };
```

client.cpp

```
#include "types.hpp"

int f(const void* vp) { return 1; }
int f(const B*    bp) { return 0; }

int test(D* dp) [[ensures t: t == 1]]
{ return f(dp); }
```

Source: google.github.io/styleguide/cppguide.html#Forward_Declarations

Issues with subtle breaking changes

types.hpp

```
struct B     { /* ... */ };
struct D : B { /* ... */ };
```

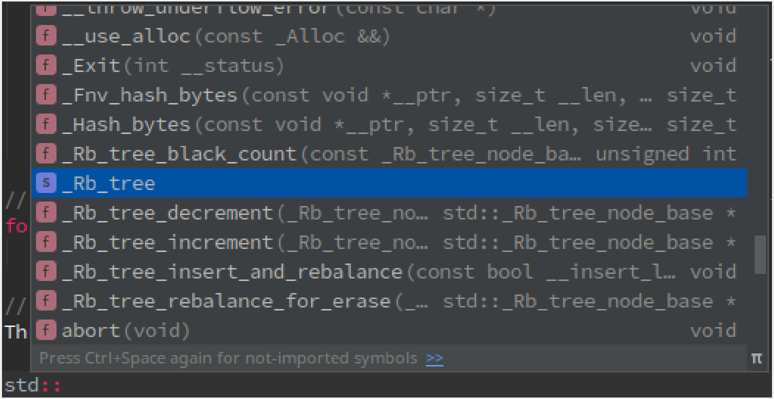
client.cpp

```
#include "types.hpp"

int f(const void* vp) { return 1; }
int f(const B*    bp) { return 0; }

int test(D* dp) [[ensures t: t == 1]]
{ return f(dp); }  // resolved as 'f(const B*)'
```

Source: google.github.io/styleguide/cppguide.html#Forward_Declarations

Issues with lack of hiding details

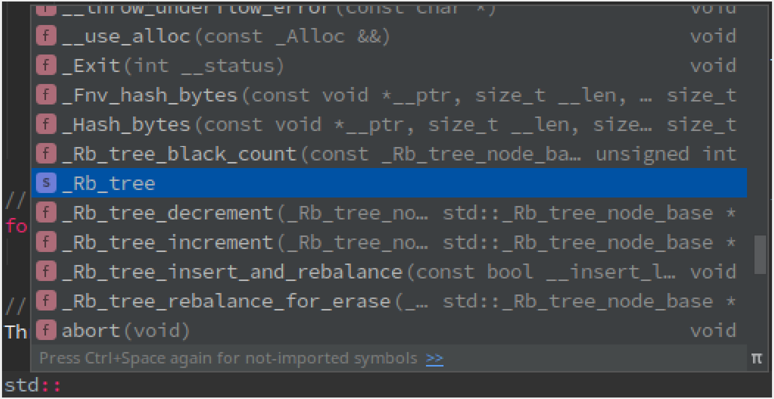
Symbol in input buffer ⟶ it's there forever



```
   f __throw_underflow_error(const char *)                   void
   f __use_alloc(const _Alloc &&)                            void
   f _Exit(int __status)                                     void
   f _Fnv_hash_bytes(const void *__ptr, size_t __len, … size_t
   f _Hash_bytes(const void *__ptr, size_t __len, size… size_t
   f _Rb_tree_black_count(const _Rb_tree_node_ba… unsigned int
   s _Rb_tree
// f _Rb_tree_decrement(_Rb_tree_no… std::_Rb_tree_node_base *
fo f _Rb_tree_increment(_Rb_tree_no… std::_Rb_tree_node_base *
   f _Rb_tree_insert_and_rebalance(const bool __insert_l… void
// f _Rb_tree_rebalance_for_erase(_… std::_Rb_tree_node_base *
Th f abort(void)                                             void
   Press Ctrl+Space again for not-imported symbols >>           π
std::
```

Issues with lack of hiding details

Symbol in input buffer ⟶ it's there forever



```
f __throw_underflow_error(const char *)                         void
f __use_alloc(const _Alloc &&)                                   void
f _Exit(int __status)                                            void
f _Fnv_hash_bytes(const void *__ptr, size_t __len, …           size_t
f _Hash_bytes(const void *__ptr, size_t __len, size…           size_t
f _Rb_tree_black_count(const _Rb_tree_node_ba…           unsigned int
s _Rb_tree
f _Rb_tree_decrement(_Rb_tree_no…    std::_Rb_tree_node_base *
f _Rb_tree_increment(_Rb_tree_no…    std::_Rb_tree_node_base *
f _Rb_tree_insert_and_rebalance(const bool __insert_l…          void
f _Rb_tree_rebalance_for_erase(_…    std::_Rb_tree_node_base *
f abort(void)                                                    void
Press Ctrl+Space again for not-imported symbols >>              π
```

```
//
fo
//
Th
std::
```

Hide with **`static`** (or anonymous **`namespace`**)?

Issues with lack of hiding details

For templates, *internal linkage* isn’t a solution…

```
template <class T>
class X
{
    private:
        T foo(T t);
};
```

Issues with lack of hiding details

For templates, *internal linkage* isn’t a solution…

```
template <class T>
class X
{
    private:
        T foo(T t);
};
```

- Is `X<int>{}.foo` a valid naming?
- Can I actually call the function?

Issues with lack of hiding details

For templates, *internal linkage* isn't a solution…

```
template <class T>
class X
{
    private:
        T foo(T t);
};
```

- Is `X<int>{}.foo` a valid naming? ✓ ***Yes!***
- Can I actually call the function?

Issues with lack of hiding details

For templates, *internal linkage* isn't a solution…

```
template <class T>
class X
{
    private:
        T foo(T t);
};
```

- Is `X<int>{}.foo` a valid naming? ✓ ***Yes!***
- Can I actually call the function? ✗ ***No!***

Issues with lack of hiding details

For templates, *internal linkage* isn't a solution...

```
template <class T>
class X
{
    private:
        T foo(T t);
};
```

- Is `X<int>{}.foo` a valid naming? ✓ ***Yes!***
- Can I actually call the function? ? ***Maybe...***[1]

[1] Márton and Porkoláb, "Unit Testing in C++ with Compiler Instrumentation and `friends`".

Issues with lack of hiding details

`detail` namespace

```
static boost::regex rCppFiles("\\.cpp\b");

/* ... */

std::string sFilenameArgs("/tmp/foo.cpp");
boost::re_detail::matcher mBuf(
    sFilename.begin(), sFilename.end(),
    /* ... */,
    &rCppFiles);

// Inner detail through mBuf instead of documented API
```

Issues with lack of hiding details

```
boost::re_detail::matcher mBuf(/* ... */);
```

Issues with lack of hiding details

```
boost::re_detail::matcher mBuf(/* ... */);
```

Compiler errors only go so long...

Issues with lack of hiding details

```
boost::re_detail::matcher mBuf(/* ... */);
```

Compiler errors only go so long...
Linters and other analysis tools only go so long...

Issues with lack of hiding details

```
boost::re_detail::matcher mBuf(/* ... */);
```

Compiler errors only go so long…
Linters and other analysis tools only go so long…

Users go further.

“Faces”, “Metrics” and “Goals” around Development

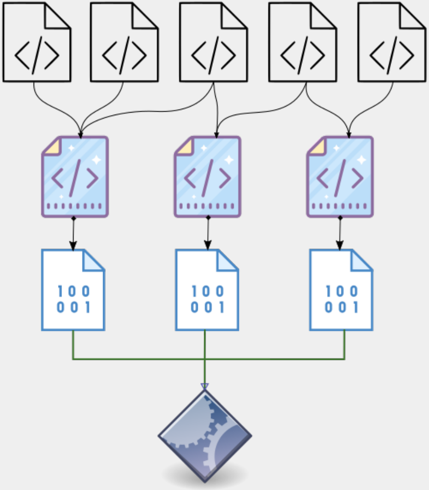
- Good (?) code
- Often easily written
- Almost always well readable
- Good run-time performance
- Stable behaviour, tested, …

- Good tooling:
  - Build (and package?) management
  - Static analysis, coverage
  - Code comprehension
- **Solid releases, nightlies**
- **Easy, fast incremental development**

Not just compilation model

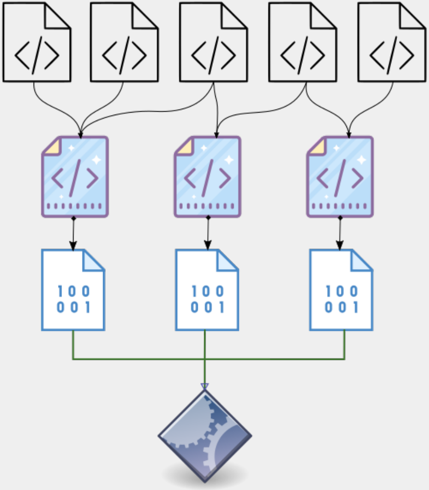
Remember this?

We only do this for all translation units **once** per (re-)build, right?



Not just compilation model

Remember this?

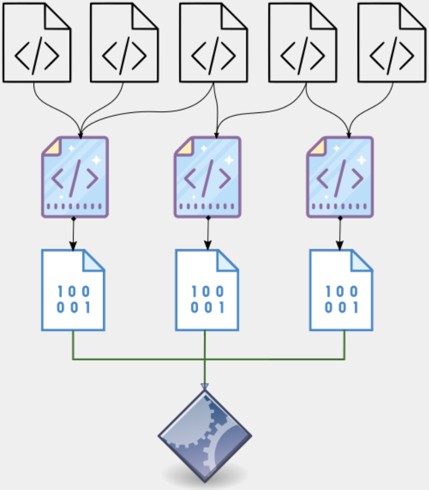


We only do this for all translation units **once** per (re-)build, right?

❌ ***Wrong!***

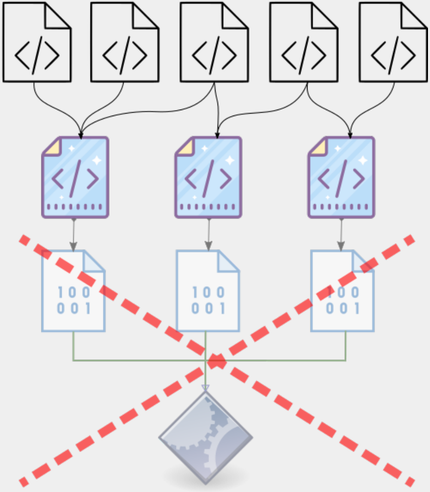
NOT JUST COMPILATION MODEL

- *Compiler*



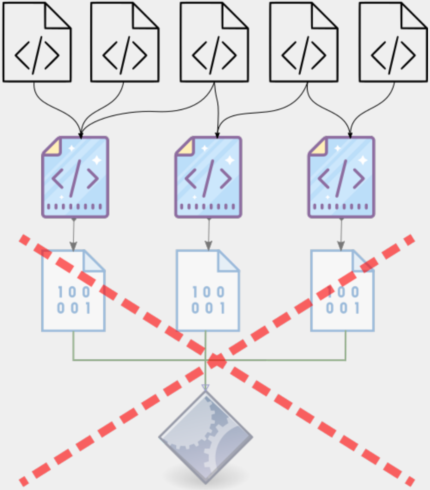
Not just compilation model

- *Compiler*
- Static analysis



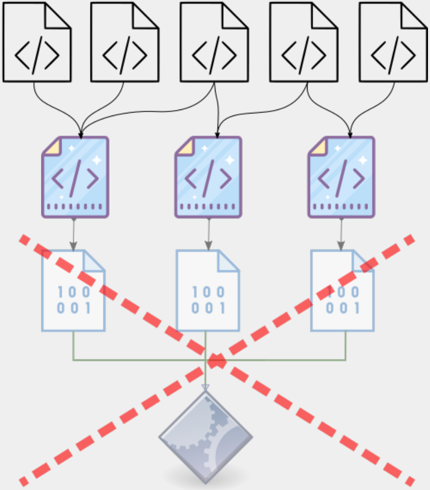
Not just compilation model

- *Compiler*
- Static analysis
- Code comprehension, indexing



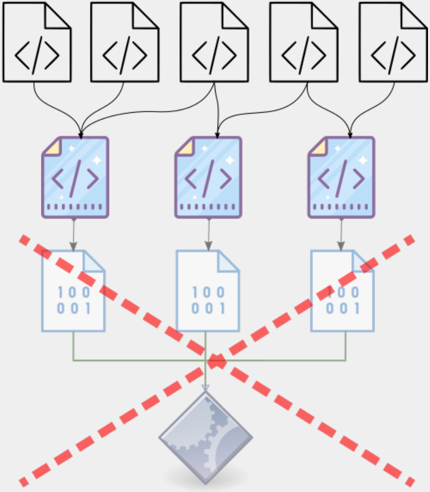
Not just compilation model

- *Compiler*
- Static analysis
- Code comprehension, indexing
- Formatting



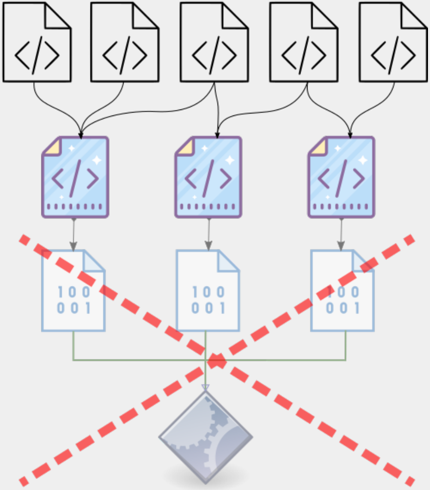
Not just compilation model

- *Compiler*
- Static analysis
- Code comprehension, indexing
- Formatting
- Automatic refactoring



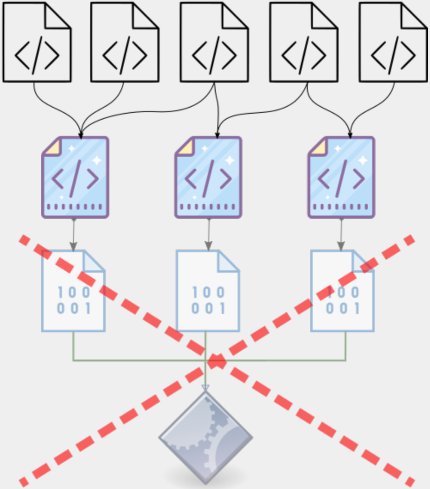
Not just compilation model

- *Compiler*
- Static analysis
- Code comprehension, indexing
- Formatting
- Automatic refactoring
- Metrics



NOT JUST COMPILATION MODEL

- *Compiler*
- Static analysis
- Code comprehension, indexing
- Formatting
- Automatic refactoring
- Metrics
- ...



Performance concerns

Personal story…

Performance concerns

Personal story…

Working on an algorithm in *CodeCompass*[2].

- TU depends on: STL, Boost, LLVM, ODB (database), …
- 24-core system, plenty of RAM
- Incremental rebuild 1 TU: $\sim$ 6 minutes

[2] `http://github.com/Ericsson/CodeCompass`

Performance concerns

Personal story…

Working on an algorithm in *CodeCompass*[2].

- TU depends on: STL, Boost, LLVM, ODB (database), …
- 24-core system, plenty of RAM
- Incremental rebuild 1 TU: $\sim$ 6 minutes

1. Move sources and build to memory. $\sim 2.5 - 3$ minutes

[2] http://github.com/Ericsson/CodeCompass

Performance concerns

Personal story…

Working on an algorithm in *CodeCompass*[2].

- TU depends on: STL, Boost, LLVM, ODB (database), …
- 24-core system, plenty of RAM
- Incremental rebuild 1 TU: $\sim$ 6 minutes

1. Move sources and build to memory. $\sim 2.5 - 3$ minutes
2. Cut down on includes. $\sim$ 1 minute 40 seconds

[2] `http://github.com/Ericsson/CodeCompass`

Performance concerns

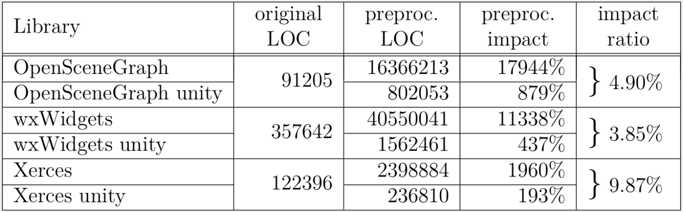
Because of *separate compilation*, many times:

- `#include` files read from “storage”

Performance concerns

Because of *separate compilation*, many times:

- `#include` files read from "storage"
- Big chunk of the input buffer is copy-paste:



Library	original LOC	preproc. LOC	preproc. impact	impact ratio
OpenSceneGraph	91205	16366213	17944%	} 4.90%
OpenSceneGraph unity		802053	879%	
wxWidgets	357642	40550041	11338%	} 3.85%
wxWidgets unity		1562461	437%	
Xerces	122396	2398884	1960%	} 9.87%
Xerces unity		236810	193%	

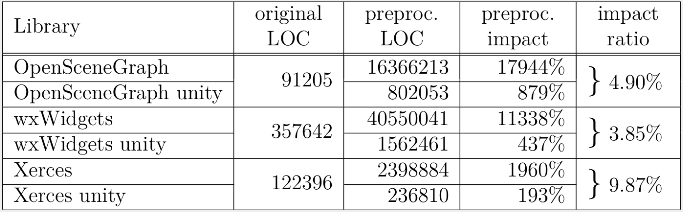
Figure: Impact on input buffer *LoC* by preprocessing.[3]

[3] Mihalicza, "How `#includes` Affect Build Time in Large Systems".

Performance concerns

Because of *separate compilation*, many times:

- #include files read from “storage”
- Big chunk of the input buffer is copy-paste:



Library	original LOC	preproc. LOC	preproc. impact	impact ratio
OpenSceneGraph	91205	16366213	17944%	4.90%
OpenSceneGraph unity		802053	879%	
wxWidgets	357642	40550041	11338%	3.85%
wxWidgets unity		1562461	437%	
Xerces	122396	2398884	1960%	9.87%
Xerces unity		236810	193%	

Figure: Impact on input buffer *LoC* by preprocessing.[3]

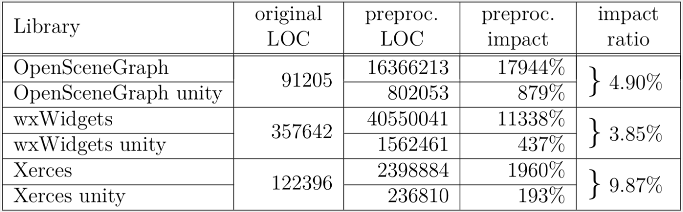
- Template generation – even worse than preprocessor

[3] Mihalicza, “How #includes Affect Build Time in Large Systems”.

Performance concerns

Because of *separate compilation*, many times:

- `#include` files read from “storage”
- Big chunk of the input buffer is copy-paste:



Library	original LOC	preproc. LOC	preproc. impact	impact ratio
OpenSceneGraph	91205	16366213	17944%	} 4.90%
OpenSceneGraph unity		802053	879%	
wxWidgets	357642	40550041	11338%	} 3.85%
wxWidgets unity		1562461	437%	
Xerces	122396	2398884	1960%	} 9.87%
Xerces unity		236810	193%	

Figure: Impact on input buffer *LoC* by preprocessing.[3]

- Template generation – even worse than preprocessor
- Weak references thrown away at linking

[3] Mihalicza, “How `#includes` Affect Build Time in Large Systems”.

Possible solutions to performance

- Boosting build process

Possible ~~*solutions*~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …

Possible ~~*solutions*~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …
- **P**re**c**ompiled **h**eaders (PCH)
  - + Save and reuse compiler representation (e.g. AST) into a file

Possible ~~*solutions*~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …
- **P**re**c**ompiled **h**eaders (PCH)
  - + Save and reuse compiler representation (e.g. AST) into a file
  - - *PCH*s require build system support

Possible ~~*solutions*~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …
- **P**re**c**ompiled **h**eaders (PCH)
  - + Save and reuse compiler representation (e.g. AST) into a file
  - - *PCH*s require build system support
- HP’s aCC compiler[4]

[4] Krishnaswamy, “Automatic Precompiled Headers: Speeding up C++ Application Build Times”.

Possible ~~*solutions*~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …
- **P**re**c**ompiled **h**eaders (PCH)
  - + Save and reuse compiler representation (e.g. AST) into a file
  - - *PCHs* require build system support
- HP's aCC compiler[4]
  - PCH for TU's "preamble"

[4] Krishnaswamy, "Automatic Precompiled Headers: Speeding up C++ Application Build Times".

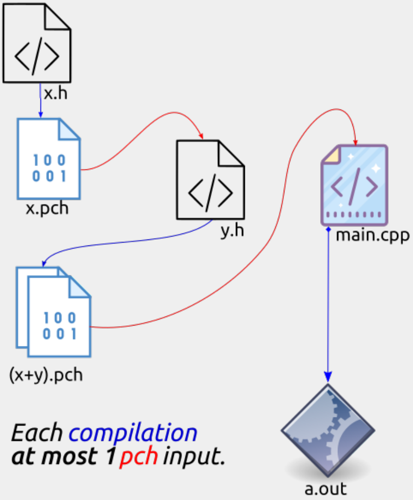
Possible ~~solutions~~ **workarounds**

- Boosting build process: `ccache`, `distcc`, *Bazel*, …
- **P**re**c**ompiled **h**eaders (PCH)
  - + Save and reuse compiler representation (e.g. AST) into a file
  - - *PCHs* require build system support
- HP's aCC compiler[4]
  - ► PCH for TU's "preamble"
- C++ Modules?

[4] Krishnaswamy, "Automatic Precompiled Headers: Speeding up C++ Application Build Times".

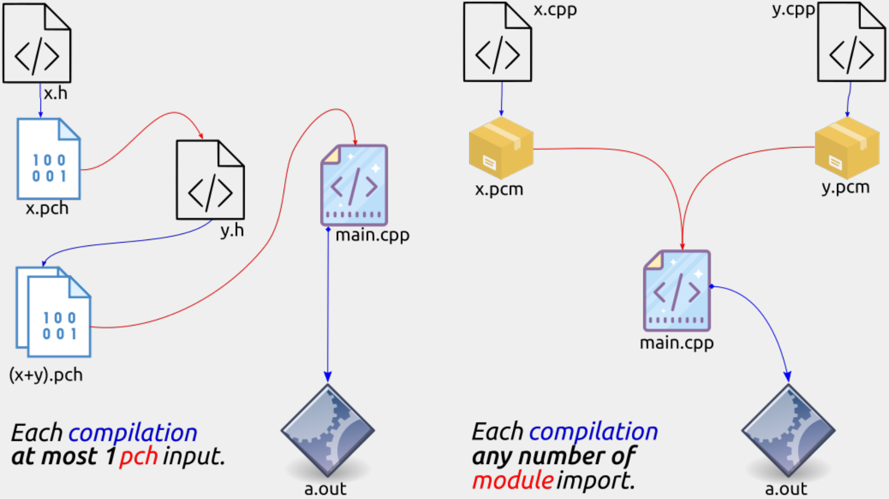
1. C++’s Compilation Model
   - Software technology issues
   - Performance drawbacks and possible solutions
2. C++ Modules
   - “How it should work?”
   - Traps and pitfalls (even) with Modules
3. The Wish for Automatic Modularisation
   - Formal overview
   - Case study — Apache Xerces
   - Evaluation of findings
   - Requirements for upgrading to Modules
4. Summary

PCHs vs. C++ Modules



Each compilation at most 1 pch input.

PCHs vs. C++ Modules



Each compilation at most 1 pch input.

Each compilation any number of module import.

PCM extension for something like the “binary module interface” is Modules for Clang, is a beefed up PCH under the hood.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**

[5] Smith, P1103R3: *Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set

[5] Smith, P1103R3*: Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set

+ Header mechanics ⟶ logical packaging

[5] Smith, P1103R3*: Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set
- + Header mechanics ⟶ logical packaging
- One module is still a (bunch of) translation units

[5] Smith, P1103R3*: Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set
- + Header mechanics ⟶ logical packaging
- One module is still a (bunch of) translation units
- How to use? `import M;`

[5] Smith, P1103R3: *Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set
- + Header mechanics ⟶ logical packaging
- One module is still a (bunch of) translation units
- How to use? `import M;`
- + No preprocessor effect between TUs

[5] Smith, P1103R3: *Merging Modules*.

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set
- + Header mechanics ⟶ logical packaging
- One module is still a (bunch of) translation units
- How to use? `import M;`
- + No preprocessor effect between TUs
- Name not part of FQN: ~~`M::std::vector`~~

[5] Smith, P1103R3: *Merging Modules.*

C++ Modules (as of P1103R3[5])

- Original form first proposed in **2004**
- Officially part of *C++20* feature set
- + Header mechanics ⟶ logical packaging
- One module is still a (bunch of) translation units
- How to use? `import M;`
- + No preprocessor effect between TUs
- Name not part of FQN: ~~`M::std::vector`~~
- ? Wording of standard proposal is rather flexible to allow compiler optimisations?

[5] Smith, P1103R3: *Merging Modules.*

“Hello Modules!”

“Hello Modules!”

Module file

```
export module MyModule;

       int four() { return 4; }
       int  six() { return four() + 2; }
```

“Hello Modules!”

Module file

```
export module MyModule;

       int four() { return 4; }
       int  six() { return four() + 2; }
```

Client code

```
import MyModule;
int main() {
    int s;
    int f;

}
```

“Hello Modules!”

Module file

```
export module MyModule;

       int four() { return 4; }
export int  six() { return four() + 2; }
```

Client code

```
import MyModule;
int main() {
    int s =  six();
    int f = four();

}
```

“Hello Modules!”

Module file

```
export module MyModule;

       int four() { return 4; }
export int  six() { return four() + 2; }
```

Client code

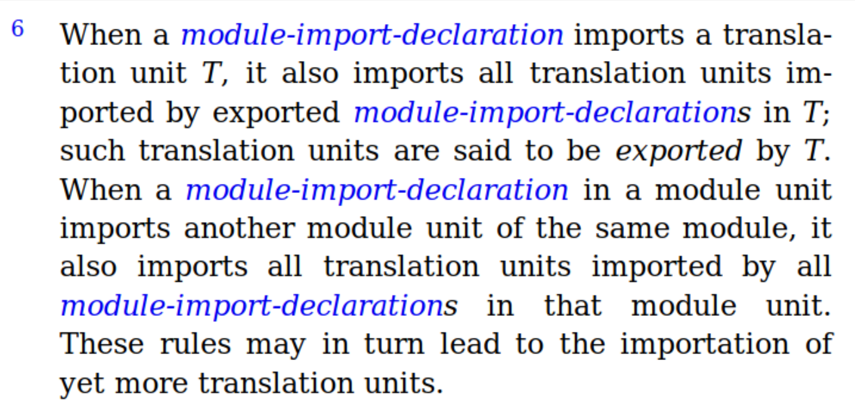
```
import MyModule;
int main() {
    int s =  six();
    int f = four(); // <- Compile error:
                    // no function named four().
}
```

```cpp
export module M;

int const export foo() noexcept { /* ... */ }
```

```
export module M;

int const export foo() noexcept { /* ... */ }
```

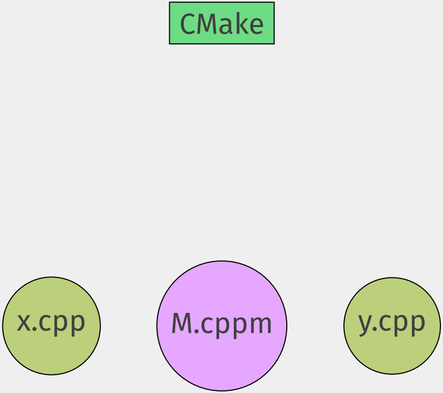
```
east-export.cpp:3:11:
        error: expected unqualified-id
int const export foo() { return 2; }
          ^
1 error generated.
```

How it should work?

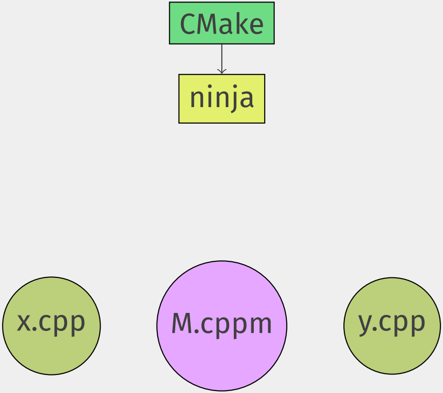


6 When a *module-import-declaration* imports a translation unit *T*, it also imports all translation units imported by exported *module-import-declarations* in *T*; such translation units are said to be *exported* by *T*. When a *module-import-declaration* in a module unit imports another module unit of the same module, it also imports all translation units imported by all *module-import-declarations* in that module unit. These rules may in turn lead to the importation of yet more translation units.

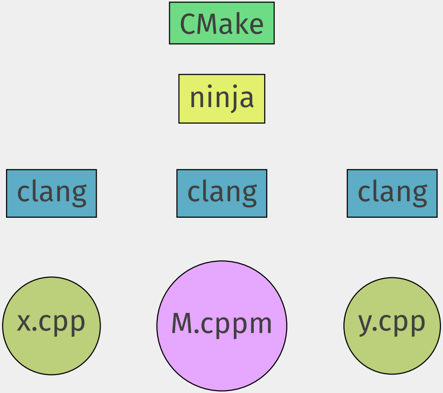
How it should work? (version 1, “preventive”)



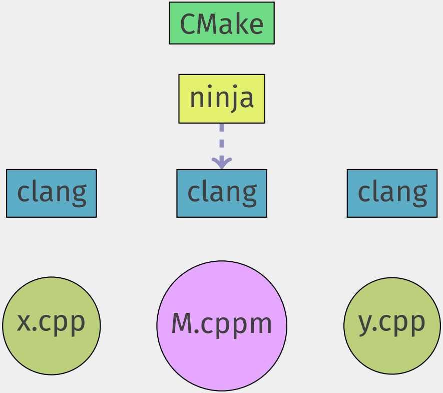
How it should work? (version 1, "preventive")



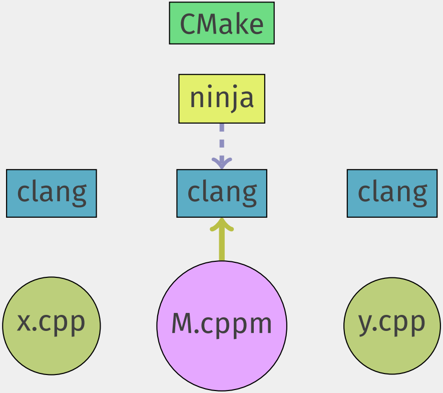
How it should work? (version 1, "preventive")



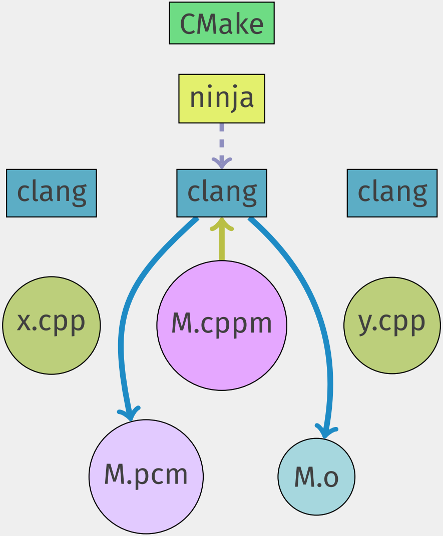
How it should work? (version 1, “preventive”)



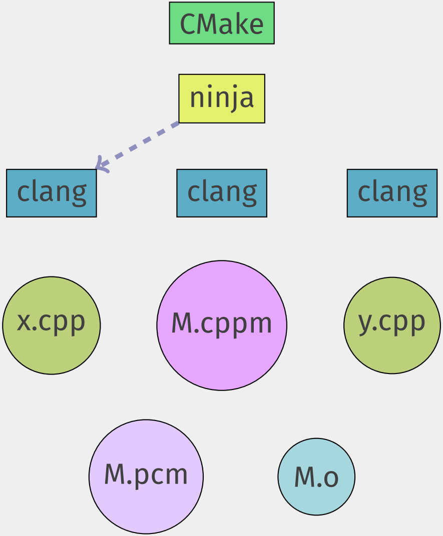
How it should work? (version 1, “preventive”)



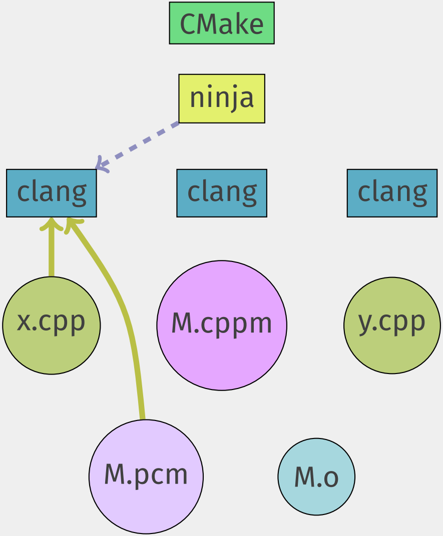
How it should work? (version 1, “preventive”)



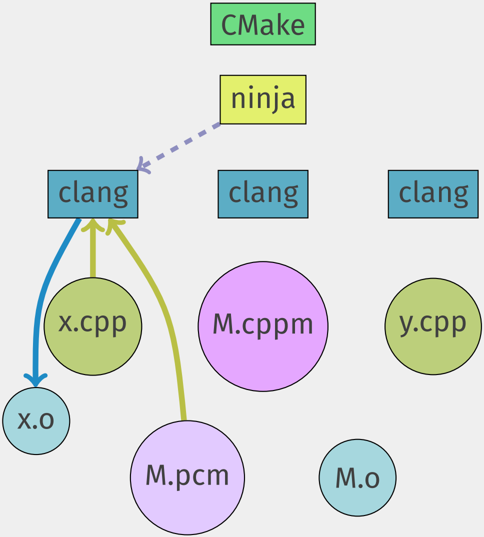
How it should work? (version 1, “preventive”)



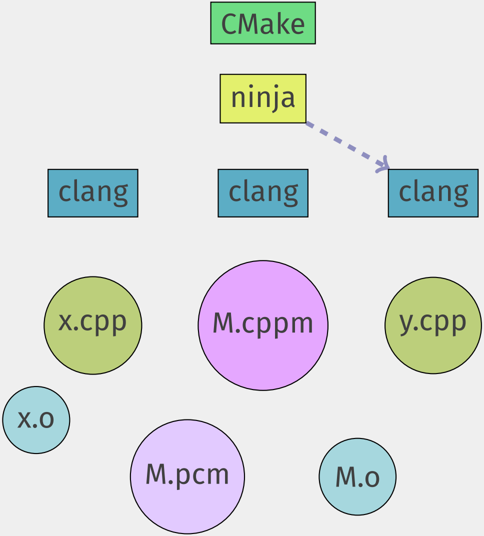
How it should work? (version 1, "preventive")



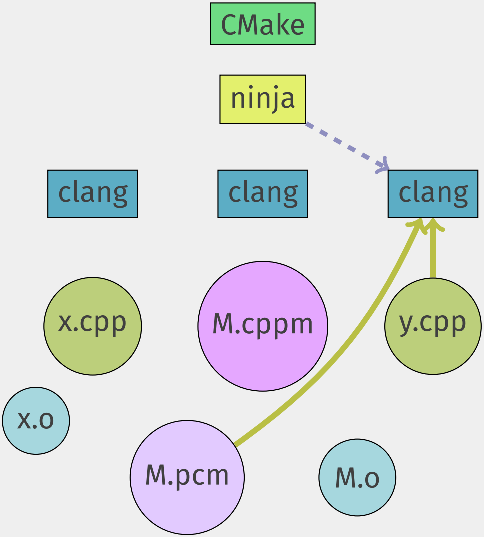
How it should work? (version 1, “preventive”)



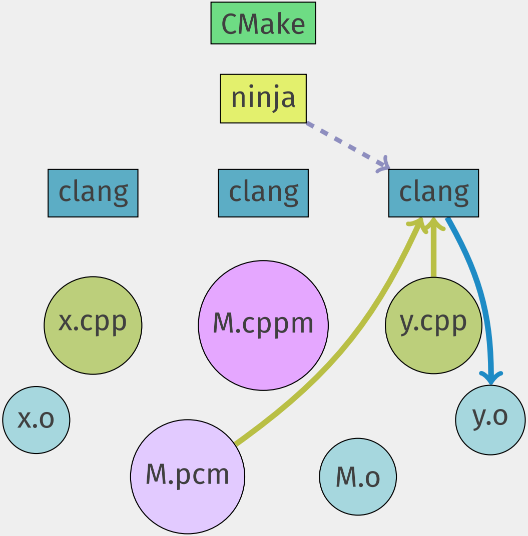
How it should work? (version 1, "preventive")



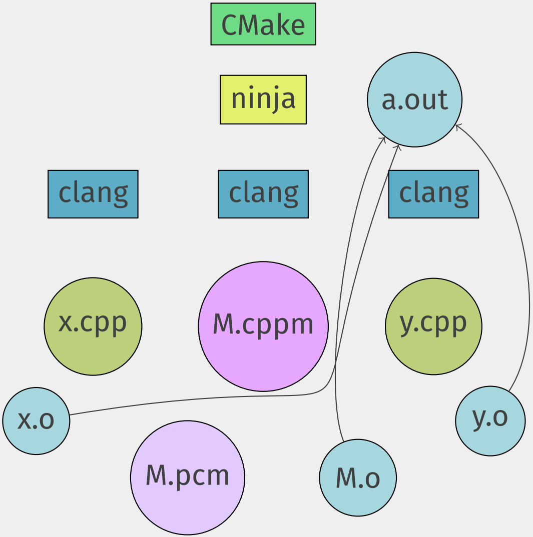
How it should work? (version 1, “preventive”)



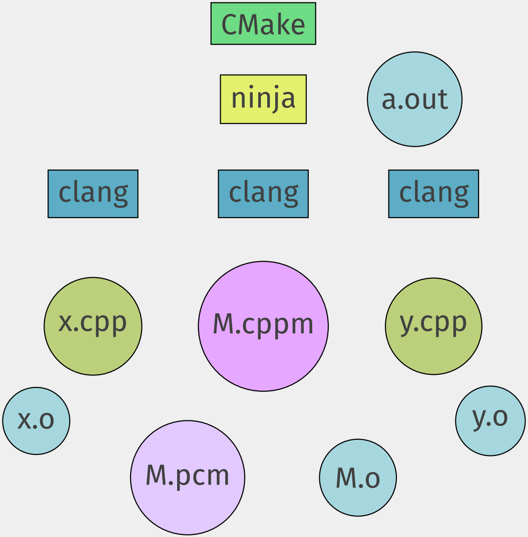
How it should work? (version 1, “preventive”)



How it should work? (version 1, "preventive")



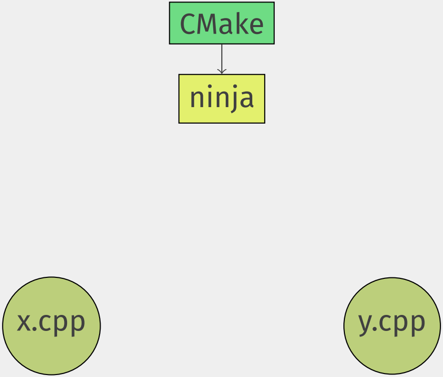
How it should work? (version 1, "preventive")



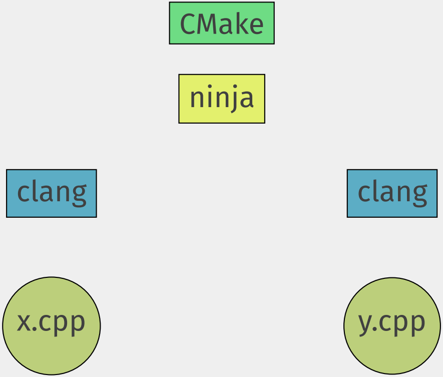
How it should work? (version 2, "on-demand")

CMake
x.cpp
y.cpp

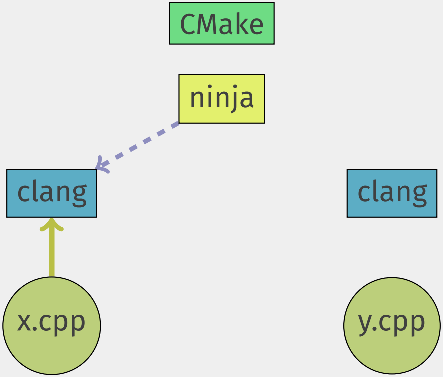
How it should work? (version 2, "on-demand")



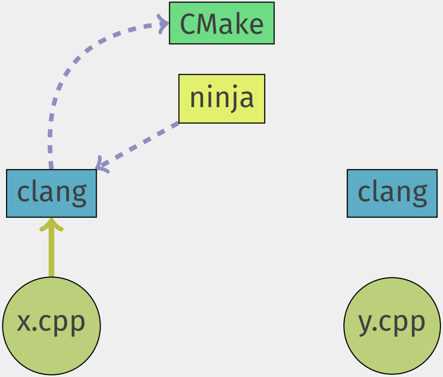
How it should work? (version 2, “on-demand”)



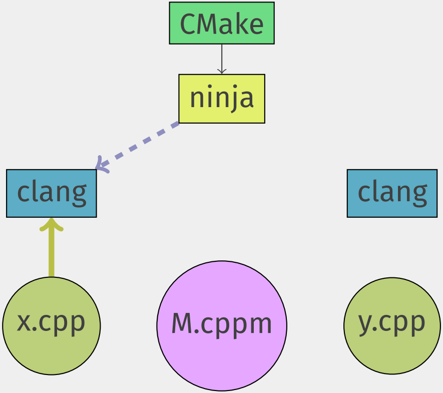
How it should work? (version 2, "on-demand")



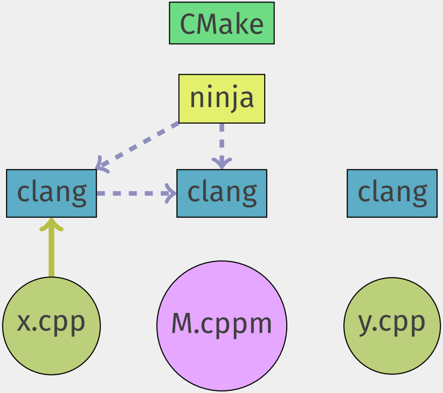
How it should work? (version 2, "on-demand")



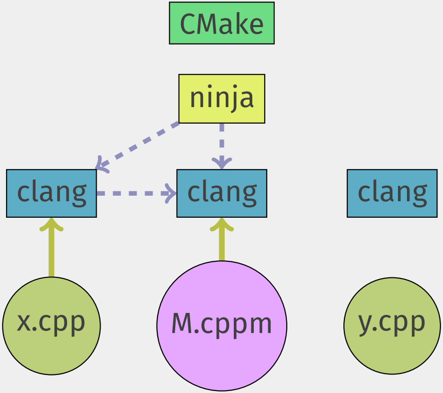
How it should work? (version 2, "on-demand")



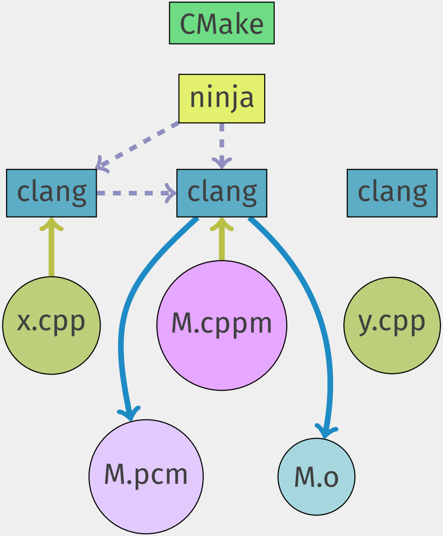
How it should work? (version 2, "on-demand")



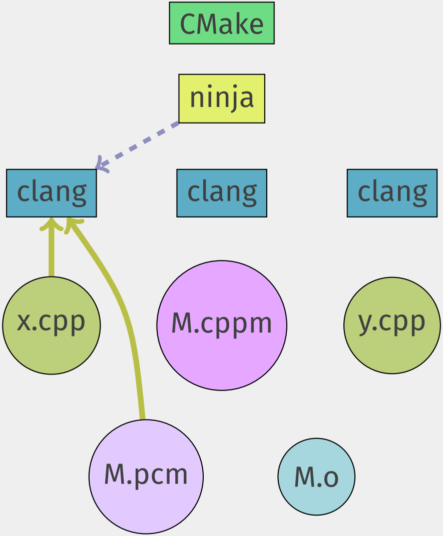
How it should work? (version 2, "on-demand")



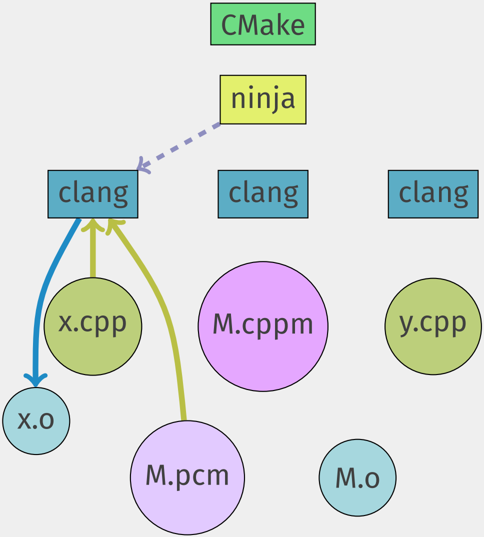
How it should work? (version 2, “on-demand”)



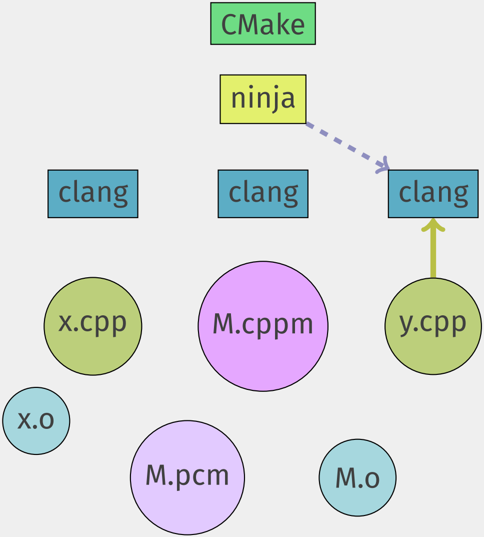
How it should work? (version 2, “on-demand”)



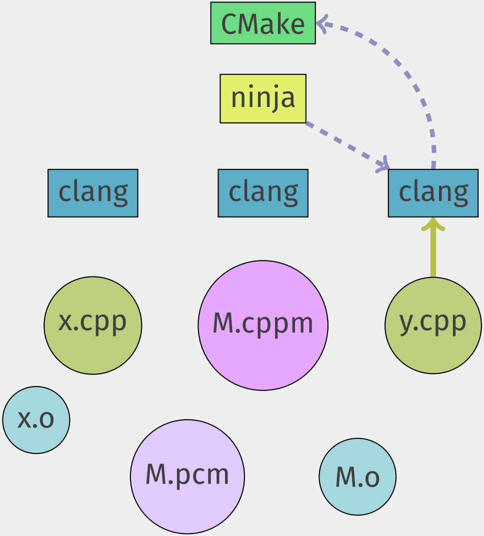
How it should work? (version 2, "on-demand")



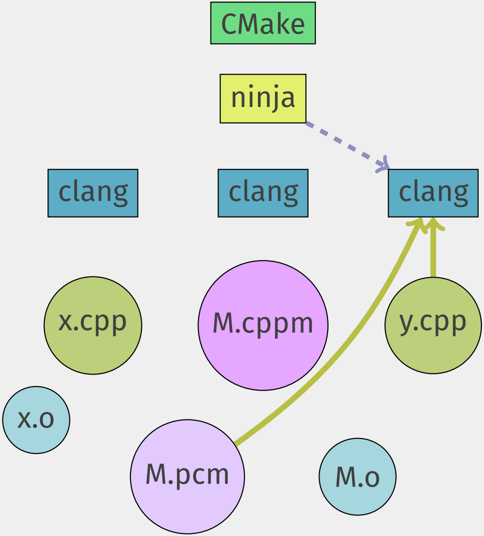
How it should work? (version 2, "on-demand")



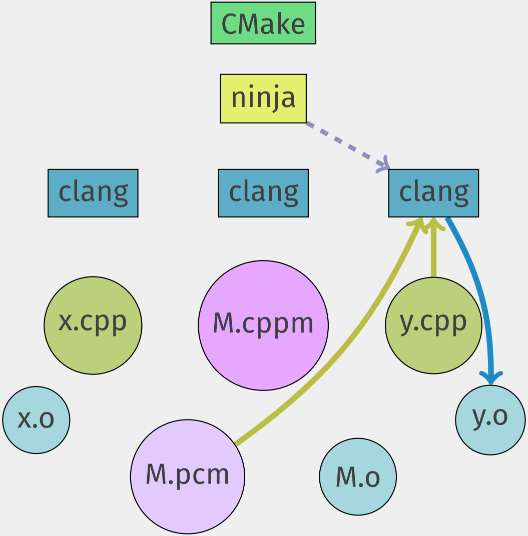
How it should work? (version 2, "on-demand")



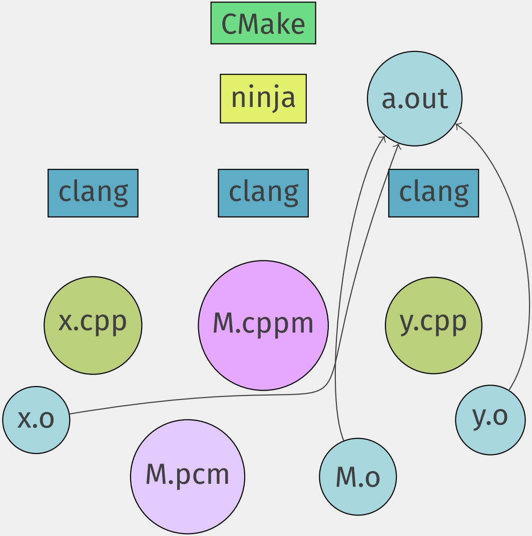
How it should work? (version 2, “on-demand”)



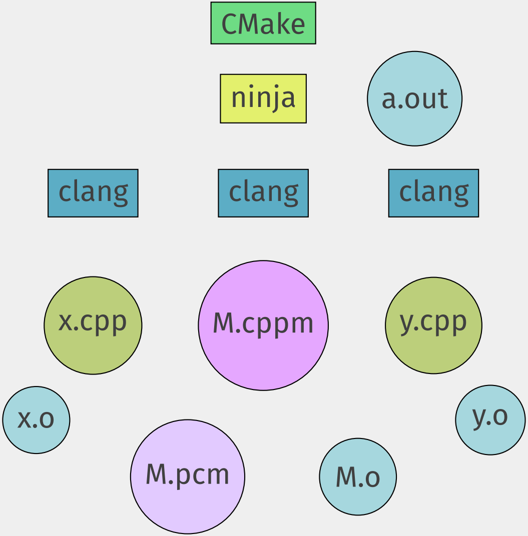
How it should work? (version 2, "on-demand")



How it should work? (version 2, “on-demand”)



How it should work? (version 2, "on-demand")



What has Modules ever done for us?

- Token leak solved — PP shouldn’t affect client
- Hiding true detail!
- Better explained interfaces

Modules + Contracts ⊂ Static Analysis 💙

What is in the library?

library.hpp

```
/* Precondition: a greater or equal to b! */
int fun(int a, int b);
```

secret_library_code.cpp

```
int fun(int a, int b)
{
    if (a < b)
        throw std::domain_error{
            "first argument must be bigger!"};
    /* ... */
}
```

Modules + Contracts ⊂ Static Analysis 💙

What do the user (and tools) see?

library.hpp

```
/* Precondition: a greater or equal to b! */
int fun(int a, int b);
```

secret_library_code.cpp

Modules + Contracts ⊂ Static Analysis 💙

library.cppm

```
export int fun(int a, int b)
        [[expects P: a >= b]];
```

library_secret.cppm

What has Modules ever done for us?

- Token leak solved — PP shouldn't affect client
- Hiding true detail!
- Better explained interfaces

What has Modules ever done for us?

- Token leak solved — PP shouldn’t affect client
- Hiding true detail!
- Better explained interfaces
- Fix some burden on developers: which header to use?!

What has Modules ever done for us?

- Token leak solved — PP shouldn't affect client
- Hiding true detail!
- Better explained interfaces
- Fix some burden on developers: which header to use?!

So all problems discussed before?

So then... what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot

New ways too mess it up

module.cppm

```
export module M;
struct S
{
    S(int i)     : m(i) {}
    S(const S&) = delete;
    S(S&&)      = default;

    int m;
};
export S make_s() { return S{0}; }
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```
export module M;
struct S                              // Not exported!
{                                     // But reachable.
    S(int i)     : m(i) {}
    S(const S&) = delete;
    S(S&&)      = default;

    int m;
};
export S make_s() { return S{0}; }
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```
export module M;
/* not exported */ struct S   { /* ... */ };
       export      S make_s() { return S{0}; }
```

main.cpp

```
import M;
int main() {
    S s{1};

}
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```
export module M;
/* not exported */ struct S   { /* ... */ };
       export      S make_s() { return S{0}; }
```

main.cpp

```
import M;
int main() {
    S s{1};
//! ^ error: no type name 'S' in current scope.
}
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```cpp
export module M;
/* not exported */ struct S   { /* ... */ };
       export      S make_s() { return S{0}; }
```

main.cpp

```cpp
import M;
int main() {
    auto s = make_s();

}
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```
export module M;
/* not exported */ struct S   { /* ... */ };
      export       S make_s() { return S{0}; }
```

main.cpp

```
import M;
int main() {
    auto s = make_s();

}
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

New ways too mess it up

module.cppm

```cpp
export module M;
/* not exported */ struct S   { /* ... */ };
       export      S make_s() { return S{0}; }
```

main.cpp

```cpp
import M;
int main() {
    auto s = make_s();
    s.m = 1;
}
```

Source: https://vector-of-bool.github.io/2019/03/31/modules-2.html

So then… what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot

So then... what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot
- Some new implementation-defined behaviour

So then… what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot
- Some new implementation-defined behaviour
- **Performance concerns!**

Performance issues

main.cpp

```cpp
import containers;
int main()
{
    std::vector V {1, 2, 3};
    return V.size() - 3;
}
```

```
$ objdump -t main.o | grep vector | grep size
```

```
0000000000000000  w    F .text._ZNKSt6vectorIiSaIiEE4sizeEv
// std::vector<int, std::allocator<int> >::size() const
```

Hannah McC 2018
"LONG LIVE THE KING..."
@MIGRAINEARTS

Source: https://migrainearts.deviantart.com/art/Long-Live-the-King-The-Lion-King-740054413

~~Preprocessor~~ ⟶ ASTReader/ASTImporter

Almost personal story…

Σ TU size for LLVM/Clang: ∼ 65 GiB

~~Preprocessor~~ ⟶ ASTReader/ASTImporter

Almost personal story…

Σ TU size for LLVM/Clang: $\sim$ 65 GiB
For Cross-TU analysis[6]:

- High disk I/O
- Often 40–50 TU loaded
- up to 10 GiB RAM usage **per thread**

[6] Horváth et al., “Implementation and Evaluation of Cross Translation Unit Symbolic Execution for C Family Languages”.

~~Preprocessor~~ ⟶ ASTReader/ASTImporter

Almost personal story…

Σ TU size for LLVM/Clang: ∼ 65 GiB
For Cross-TU analysis[6]:

- High disk I/O
- Often 40–50 TU loaded
- up to 10 GiB RAM usage **per thread**
- Arbitrary 8 TU cap, meaning *10-thread* analysis stays just under 30 GiB

[6] Horváth et al., "Implementation and Evaluation of Cross Translation Unit Symbolic Execution for C Family Languages".

So then… what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot
- Some new implementation-defined behaviour
- **Performance concerns!**

So then... what Modules **does not** solve?

- Increase burden on build system
- Library vendors might need an extra step
- Some new ways to shoot yourself in the foot
- Some new implementation-defined behaviour
- **Performance concerns!**

But wait, there's more!...

The big problem?

100 000 000 000

The big problem?

$$10^9 - 10^{11}$$

The big problem?

$$10^9 - 10^{11}\ LoC$$

1. C++’s Compilation Model
   - Software technology issues
   - Performance drawbacks and possible solutions
2. C++ Modules
   - “How it should work?”
   - Traps and pitfalls (even) with Modules
3. The Wish for Automatic Modularisation
   - Formal overview
   - Case study — Apache Xerces
   - Evaluation of findings
   - Requirements for upgrading to Modules
4. Summary

Automatic Modularisation: Goals

Goal

Break existing project up into modules.

Automatic Modularisation: Goals

Goal

Break existing project up into modules.

Input

- Initial mapping, usually from pure physical layout
- The source code, and conventional (“translation unit–based”) configuration metadata
- **The input contains no domain knowledge.**

Automatic Modularisation: Goals

Goal

Break existing project up into modules.

Input

- Initial mapping, usually from pure physical layout
- The source code, and conventional (“translation unit–based”) configuration metadata
- **The input contains no domain knowledge.**

Output

Refined mapping which is sensible.

Automatic Modularisation: Goals

Goal

Break existing project up into modules.

Input

- Initial mapping, usually from pure physical layout
- The source code, and conventional (“translation unit–based”) configuration metadata
- **The input contains no domain knowledge.**

Output

Refined mapping which ~~is sensible~~ compiles.

Automatic Modularisation: Overview

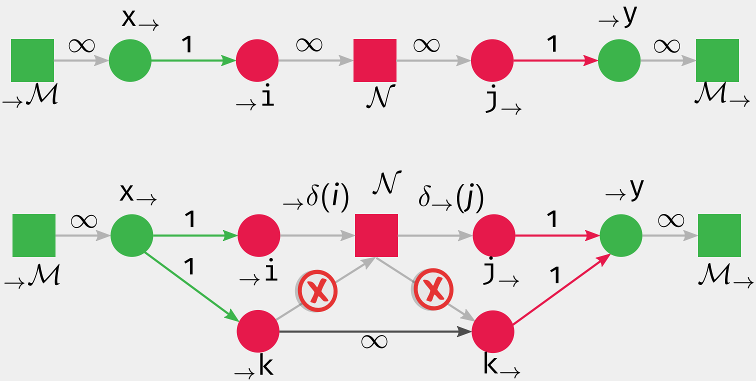
1. Deep fact extraction from the source code

Automatic Modularisation: Overview

1. Deep fact extraction from the source code
2. Build a dependency graph purpose-tailored

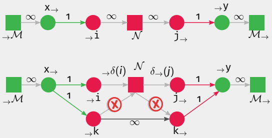
Automatic Modularisation: Overview

1. Deep fact extraction from the source code
2. Build a dependency graph purpose-tailored
3. Cuts isolate acyclic *interface dependencies* — using commodity flow algorithms



Automatic Modularisation: Overview

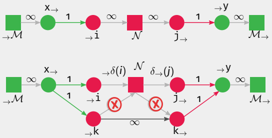
1. Deep fact extraction from the source code
2. Build a dependency graph purpose-tailored
3. Cuts isolate acyclic *interface dependencies* — using commodity flow algorithms



4. Move implementation to interface due to ownership

Automatic Modularisation: Overview

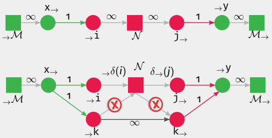
1. Deep fact extraction from the source code
2. Build a dependency graph purpose-tailored
3. Cuts isolate acyclic *interface dependencies* — using commodity flow algorithms



4. Move implementation to interface due to ownership
5. Fix graph so modules compile without (re-)introducing cycles — using path search

Automatic Modularisation: Overview

1. Deep fact extraction from the source code
2. Build a dependency graph purpose-tailored
3. Cuts isolate acyclic *interface dependencies* — using commodity flow algorithms



4. Move implementation to interface due to ownership
5. Fix graph so modules compile without (re-)introducing cycles — using path search

Result: modules $\sim$ *“unity build”* for synthesised components.

Step 1 — Fact extraction

Using LLVM/Clang AST matchers, visitors.
Driver infrastructure and graph algorithms in Python.[7]

Find, emit, organise:

- `#includes`[8]
- usage dependencies (true symbol usage)

[7] `github.com/whisperity/buildtooling/tree/module-making`

[8] Eventually could turn this over to `clang-scan-deps`: `lists.llvm.org/pipermail/cfe-dev/2018-October/059831.html`

Step 1 — Fact extraction

Using LLVM/Clang AST matchers, visitors.
Driver infrastructure and graph algorithms in Python.[7]

Find, emit, organise:

- #includes[8]
- usage dependencies (true symbol usage)
- "problematic" names, unification breakers[9]:
  - TU-internal names
  - (defined macros that bleed out)

[7] github.com/whisperity/buildtooling/tree/module-making

[8] Eventually could turn this over to clang-scan-deps: lists.llvm.org/pipermail/cfe-dev/2018-October/059831.html

[9] Mihalicza, "How #includes Affect Build Time in Large Systems".

Step 1 — Fact extraction

Using LLVM/Clang AST matchers, visitors.
Driver infrastructure and graph algorithms in Python.[7]

Find, emit, organise:

- `#includes`[8]
- usage dependencies (true symbol usage)
- “problematic” names, unification breakers[9]:
  - TU-internal names
  - (defined macros that bleed out)
- forward declarations
  - herald true usage
  - (might cause name collisions)

[7] `github.com/whisperity/buildtooling/tree/module-making`

[8] Eventually could turn this over to `clang-scan-deps`: `lists.llvm.org/pipermail/cfe-dev/2018-October/059831.html`

[9] Mihalicza, “How `#includes` Affect Build Time in Large Systems”.

Forward Declarations

M1.cppm

```
export module M1;

class Fwd;
export void  open(Fwd* fptr) { /* ... */ };
```

M2.cppm

```
export module M2;

export class Fwd  { /* ... */ };
export void  close(Fwd  f)   { /* ... */ }
```

Forward Declarations

M1.cppm

```
export module M1;

class Fwd;
export void  open(Fwd* fptr) { /* ... */ };
```

M2.cppm

```
export module M2;
import M1;

export class Fwd  { /* ... */ };
export void  close(Fwd  f)   { /* ... */ }
```

Forward Declarations

M1 ∪ M2.cppm

```
export module M;

export class Fwd  { /* ... */ };
export void  open (Fwd* fptr) { /* ... */ };
export void  close(Fwd  f)    { /* ... */ };
```

Forward Declarations

M1 ∪ M2.cppm

```
export module M;

export class Fwd  { /* ... */ };
export void  open (Fwd* fptr) { /* ... */ };
export void  close(Fwd  f)    { /* ... */ };
```

How about **Module Partitions**?

Forward Declarations

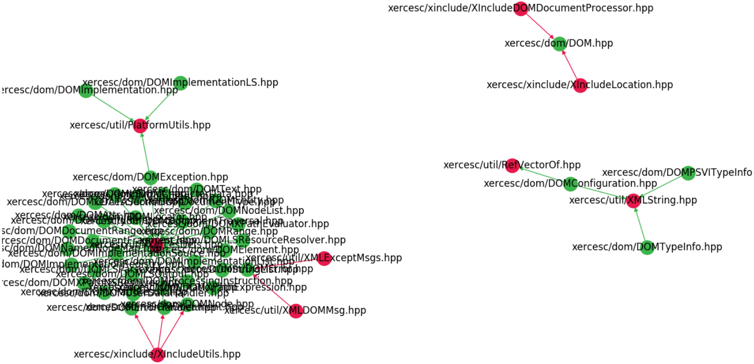
M1 ∪ M2.cppm

```
export module M;

export class Fwd  { /* ... */ };
export void  open (Fwd* fptr) { /* ... */ };
export void  close(Fwd  f)    { /* ... */ };
```

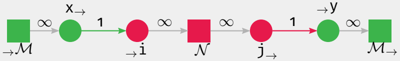
How about **Module Partitions**?

- Partitions are “just” module-internal fluff
- Don’t change the client’s view

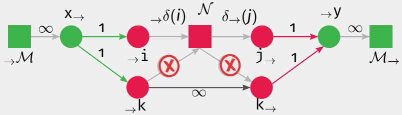
Step 2 — Build dependency graph


xercesc/xinclude/XIncludeDOMDocumentProcessor.hpp
xercesc/dom/DOM.hpp
xercesc/xinclude/XIncludeLocation.hpp
xercesc/dom/DOMImplementationLS.hpp
xercesc/dom/DOMImplementation.hpp
xercesc/util/PlatformUtils.hpp
xercesc/dom/DOMException.hpp
xercesc/dom/DOMText.hpp
xercesc/dom/DOMNodeList.hpp
xercesc/dom/DOMXPathEvaluator.hpp
xercesc/dom/DOMRange.hpp
xercesc/dom/DOMLSResourceResolver.hpp
xercesc/dom/DOMElement.hpp
xercesc/dom/DOMImplementationSource.hpp
xercesc/dom/DOMImplementationList.hpp
xercesc/dom/DOMLSOutput.hpp
xercesc/dom/DOMProcessingInstruction.hpp
xercesc/dom/DOMUserDataHandler.hpp
xercesc/dom/DOMNode.hpp
xercesc/dom/DOMDocumentRange.hpp
xercesc/util/XMLExceptMsgs.hpp
xercesc/util/XMLDOMMsg.hpp
xercesc/xinclude/XIncludeUtils.hpp
xercesc/util/RefVectorOf.hpp
xercesc/dom/DOMPSVITypeInfo
xercesc/dom/DOMConfiguration.hpp
xercesc/util/XMLString.hpp
xercesc/dom/DOMTypeInfo.hpp

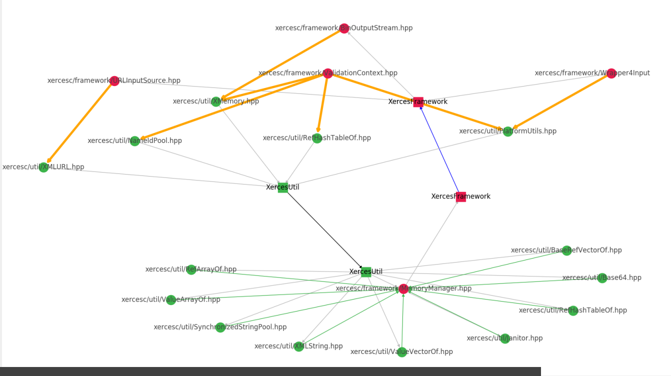
Step 3 — Perform cuts on cycles



$\mathrm{x}_{\rightarrow}$
∞
1
∞
∞
1
$_{\rightarrow}\mathrm{y}$
∞
$_{\rightarrow}\mathcal{M}$
$_{\rightarrow}\mathrm{i}$
$\mathcal{N}$
$\mathrm{j}_{\rightarrow}$
$\mathcal{M}_{\rightarrow}$

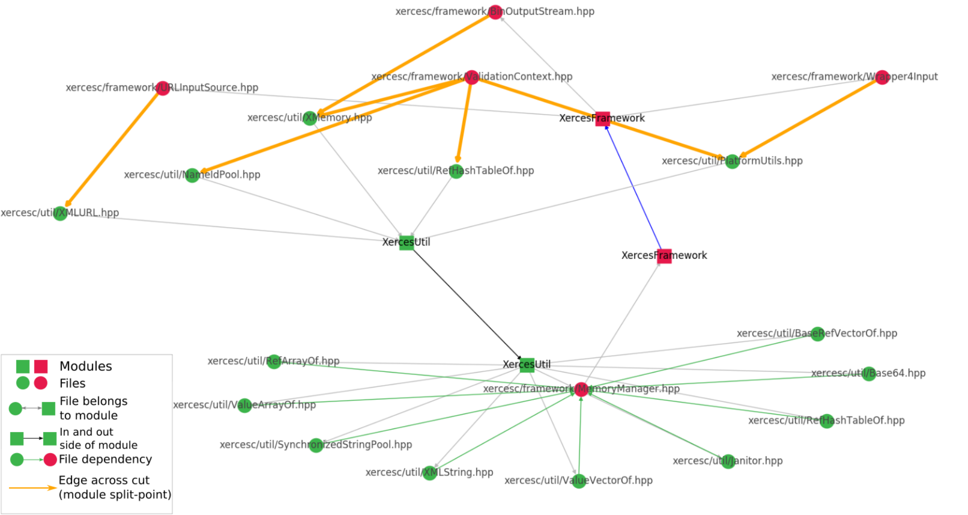


$\mathrm{x}_{\rightarrow}$
$_{\rightarrow}\delta(i)$
$\mathcal{N}$
$\delta_{\rightarrow}(j)$
$_{\rightarrow}\mathrm{y}$
∞
1
1
1
1
∞
$_{\rightarrow}\mathcal{M}$
$_{\rightarrow}\mathrm{i}$
$\mathrm{j}_{\rightarrow}$
$\mathcal{M}_{\rightarrow}$
∞
$_{\rightarrow}\mathrm{k}$
$\mathrm{k}_{\rightarrow}$



xercesc/framework/BinOutputStream.hpp
xercesc/framework/URLInputSource.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

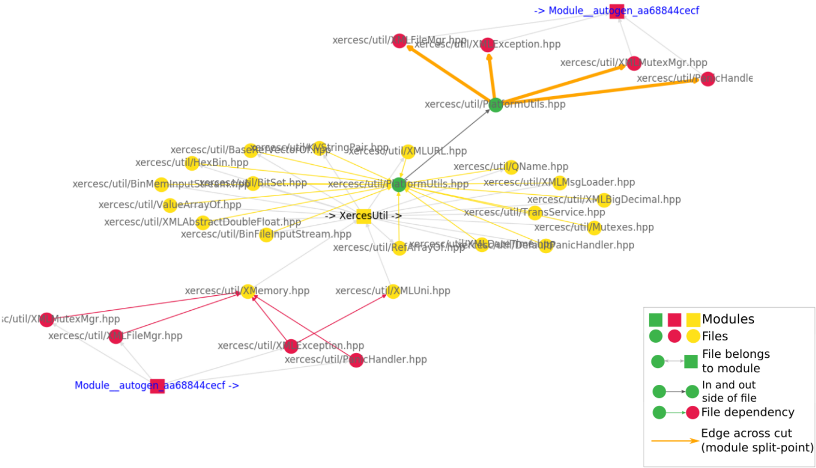
Step 3 — Perform cuts on cycles



xercesc/framework/BinOutputStream.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/URLInputSource.hpp
xercesc/framework/Wrapper4Input
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/PlatformUtils.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp
Modules
Files
File belongs to module
In and out side of module
File dependency
Edge across cut (module split-point)

Step 3 — Perform cuts on cycles

Module__autogen_aa68844cecf -> XercesUtil -> Module__autogen_f02a9dad0 -> Module__autogen_aa68844cecf


-> Module__autogen_aa68844cecf
xercesc/util/XMLFileMgr.hpp
xercesc/util/XMLException.hpp
xercesc/util/XMLMutexMgr.hpp
xercesc/util/PanicHandle
xercesc/util/PlatformUtils.hpp
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/KVStringPair.hpp
xercesc/util/XMLURL.hpp
xercesc/util/HexBin.hpp
xercesc/util/QName.hpp
xercesc/util/BinMemInputStream.hpp
xercesc/util/BitSet.hpp
xercesc/util/PlatformUtils.hpp
xercesc/util/XMLMsgLoader.hpp
xercesc/util/XMLBigDecimal.hpp
xercesc/util/ValueArrayOf.hpp
-> XercesUtil ->
xercesc/util/TransService.hpp
xercesc/util/XMLAbstractDoubleFloat.hpp
xercesc/util/Mutexes.hpp
xercesc/util/BinFileInputStream.hpp
xercesc/util/RefArrayOf.hpp
xercesc/util/XMLDateTime.hpp
xercesc/util/DefaultPanicHandler.hpp
xercesc/util/XMemory.hpp
xercesc/util/XMLUni.hpp
c/util/XMLMutexMgr.hpp
xercesc/util/XMLFileMgr.hpp
xercesc/util/XMLException.hpp
xercesc/util/PanicHandler.hpp
Module__autogen_aa68844cecf ->
Modules
Files
File belongs to module
In and out side of file
File dependency
Edge across cut (module split-point)

Step 4 — Move implementation to interface

```
import MODULE_NAME_Module_07899b6_XMemory;
/* ... */
export module FULL_NAME_Module_e77754b;

#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
#include "xercesc/xinclude/XIncludeUtils.hpp"
#include "xercesc/util/TransService.hpp"

#include "xercesc/util/TransService.cpp"
#include "xercesc/util/XMLUri.cpp"
#include "xercesc/xinclude/XIncludeUtils.cpp"
```

Step 4 — Move implementation to interface

```
import MODULE_NAME_Module_07899b6_XMemory;
/* ... */
export module FULL_NAME_Module_e77754b;

#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
#include "xercesc/xinclude/XIncludeUtils.hpp"
#include "xercesc/util/TransService.hpp"

#include "xercesc/util/TransService.cpp"
#include "xercesc/util/XMLUri.cpp"
#include "xercesc/xinclude/XIncludeUtils.cpp"
```

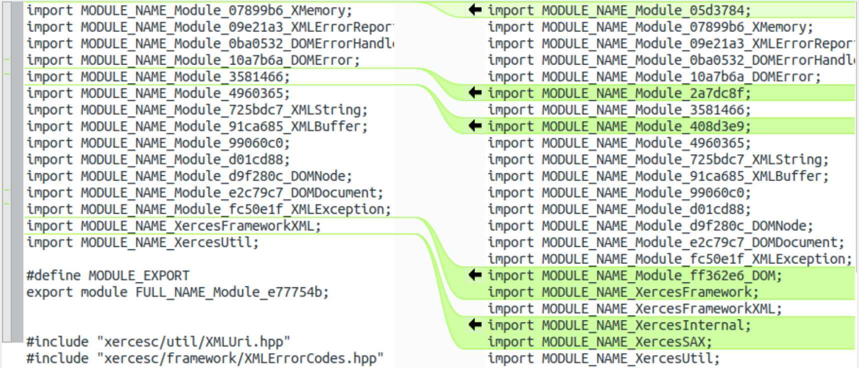
What about *module partitions*?

Step 4 — Move implementation to interface

```
import MODULE_NAME_Module_07899b6_XMemory;
/* ... */
export module FULL_NAME_Module_e77754b;

#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
#include "xercesc/xinclude/XIncludeUtils.hpp"
#include "xercesc/util/TransService.hpp"

#include "xercesc/util/TransService.cpp"
#include "xercesc/util/XMLUri.cpp"
#include "xercesc/xinclude/XIncludeUtils.cpp"
```

What about *module partitions*? 🤔

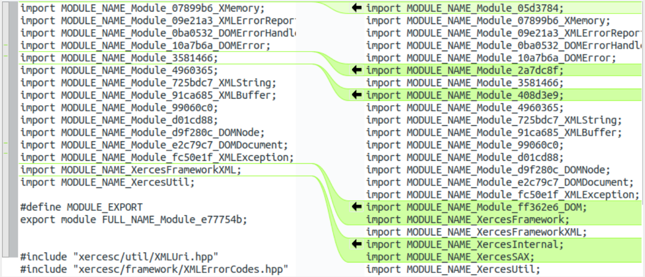
Step 5 — Join cycles



```
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesUtil;

#define MODULE_EXPORT
export module FULL_NAME_Module_e77754b;


#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
```

```
import MODULE_NAME_Module_05d3784;
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_2a7dc8f;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_408d3e9;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_Module_ff362e6_DOM;
import MODULE_NAME_XercesFramework;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesInternal;
import MODULE_NAME_XercesSAX;
import MODULE_NAME_XercesUtil;
```

Step 5 — Join cycles

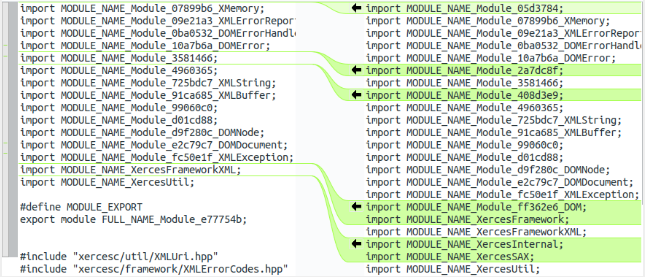


```
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesUtil;

#define MODULE_EXPORT
export module FULL_NAME_Module_e77754b;


#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
```

```
import MODULE_NAME_Module_05d3784;
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_2a7dc8f;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_408d3e9;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_Module_ff362e6_DOM;
import MODULE_NAME_XercesFramework;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesInternal;
import MODULE_NAME_XercesSAX;
import MODULE_NAME_XercesUtil;
```

New dependencies ⟶ potential new cycles.

Step 5 — Join cycles



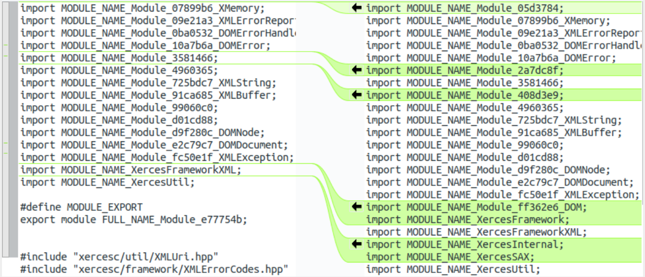
```
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesUtil;

#define MODULE_EXPORT
export module FULL_NAME_Module_e77754b;


#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
```

```
import MODULE_NAME_Module_05d3784;
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_2a7dc8f;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_408d3e9;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_Module_ff362e6_DOM;
import MODULE_NAME_XercesFramework;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesInternal;
import MODULE_NAME_XercesSAX;
import MODULE_NAME_XercesUtil;
```

New dependencies ⟶ potential new cycles.

- Analyse the paths

Step 5 — Join cycles



```
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesUtil;

#define MODULE_EXPORT
export module FULL_NAME_Module_e77754b;


#include "xercesc/util/XMLUri.hpp"
#include "xercesc/framework/XMLErrorCodes.hpp"
```

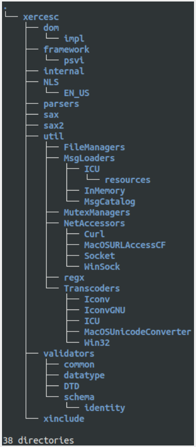
```
import MODULE_NAME_Module_05d3784;
import MODULE_NAME_Module_07899b6_XMemory;
import MODULE_NAME_Module_09e21a3_XMLErrorRepor
import MODULE_NAME_Module_0ba0532_DOMErrorHandl
import MODULE_NAME_Module_10a7b6a_DOMError;
import MODULE_NAME_Module_2a7dc8f;
import MODULE_NAME_Module_3581466;
import MODULE_NAME_Module_408d3e9;
import MODULE_NAME_Module_4960365;
import MODULE_NAME_Module_725bdc7_XMLString;
import MODULE_NAME_Module_91ca685_XMLBuffer;
import MODULE_NAME_Module_99060c0;
import MODULE_NAME_Module_d01cd88;
import MODULE_NAME_Module_d9f280c_DOMNode;
import MODULE_NAME_Module_e2c79c7_DOMDocument;
import MODULE_NAME_Module_fc50e1f_XMLException;
import MODULE_NAME_Module_ff362e6_DOM;
import MODULE_NAME_XercesFramework;
import MODULE_NAME_XercesFrameworkXML;
import MODULE_NAME_XercesInternal;
import MODULE_NAME_XercesSAX;
import MODULE_NAME_XercesUtil;
```

New dependencies ⟶ potential new cycles.

- Analyse the paths
- Merge the cycle into a bigger module

Initial input

Test bed: Apache Xerces[10].
$\sim$ 800 source files, roughly 40% : 60% (cpp : hpp) ratio.



```
.
└── xercesc
    ├── dom
    │   └── impl
    ├── framework
    │   └── psvi
    ├── internal
    ├── NLS
    │   └── EN_US
    ├── parsers
    ├── sax
    ├── sax2
    ├── util
    │   ├── FileManagers
    │   ├── MsgLoaders
    │   │   ├── ICU
    │   │   │   └── resources
    │   │   ├── InMemory
    │   │   └── MsgCatalog
    │   ├── MutexManagers
    │   ├── NetAccessors
    │   │   ├── Curl
    │   │   ├── MacOSURLAccessCF
    │   │   ├── Socket
    │   │   └── WinSock
    │   ├── regx
    │   └── Transcoders
    │       ├── Iconv
    │       ├── IconvGNU
    │       ├── ICU
    │       ├── MacOSUnicodeConverter
    │       └── Win32
    ├── validators
    │   ├── common
    │   ├── datatype
    │   ├── DTD
    │   └── schema
    │       └── identity
    └── xinclude

38 directories
```

[10] http://github.com/whisperity/xerces-c-modules

Broken code?

- Headers implemented in multiple files, across initial modules

Broken code?

- Headers implemented in multiple files, across initial modules
- Header used macro from config but didn't include config file

Broken code?

- Headers implemented in multiple files, across initial modules
- Header used macro from config but didn't include config file
- Missing forward declaration in header using the symbol

Broken code?

- Headers implemented in multiple files, across initial modules
- Header used macro from config but didn’t include config file
- Missing forward declaration in header using the symbol
- Missing header guard

The progress…

14 initial “modules”.

The progress...

14 initial “modules”.

After step 3 *cycle-breaking*, 67 “modules”.

The progress...

14 initial “modules”.

After step 3 *cycle-breaking*, 67 “modules”.

Without handling forward declarations, after step 4:
$692 + 11 + 2 + 1 + 1 + 1 + 1 + 1 + 1$.

The progress…

14 initial “modules”.

After step 3 *cycle-breaking*, 67 “modules”.

Without handling forward declarations, after step 4:
$692 + 11 + 2 + 1 + 1 + 1 + 1 + 1 + 1.$

With handling…

“End result”

```
Step merging modules on fwddecl codependencies:
    Module_09e21a3_XMLErrorReporter,
    Module_0ba0532_DOMErrorHandler,
    Module_10a7b6a_DOMError,
    Module_1a62989_MemoryManager,
    Module_3b94cf2,
    Module_9544d86,
    Module_ba6ca70,
    Module_d9f280c_DOMNode.
- Final file # after forwards -
     Module Module_23efd64: 710
     Module Module_55b5fc8_PSVIDefs: 1
```

“End result”

```
- Final file # after forwards -
     Module Module_23efd64: 710
     Module Module_55b5fc8_PSVIDefs: 1
```

“End result”

```
- Final file # after forwards -
     Module Module_23efd64: 710
     Module Module_55b5fc8_PSVIDefs: 1
```

Original number of input files: 711.

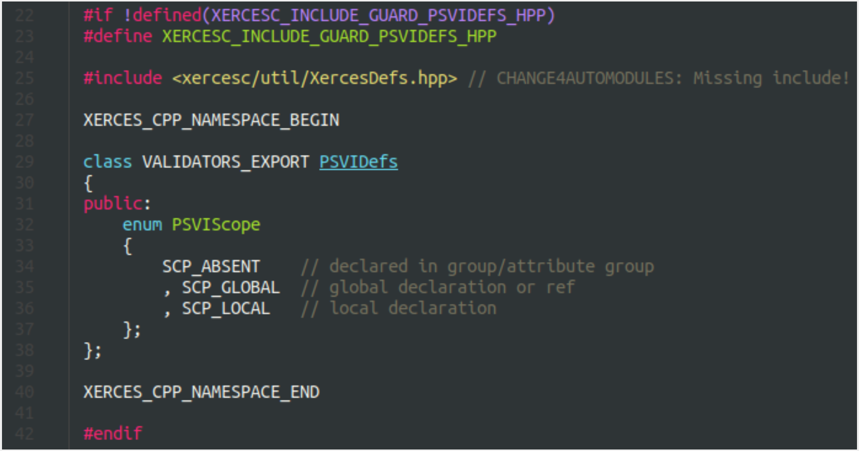
“End result”

```
- Final file # after forwards -
     Module Module_23efd64: 710
     Module Module_55b5fc8_PSVIDefs: 1
```

Original number of input files: 711.

1 *almost* true unity build + a lone **enum**.

PSVIDefs



```
#if !defined(XERCESC_INCLUDE_GUARD_PSVIDEFS_HPP)
#define XERCESC_INCLUDE_GUARD_PSVIDEFS_HPP

#include <xercesc/util/XercesDefs.hpp> // CHANGE4AUTOMODULES: Missing include!

XERCES_CPP_NAMESPACE_BEGIN

class VALIDATORS_EXPORT PSVIDefs
{
public:
    enum PSVIScope
    {
        SCP_ABSENT    // declared in group/attribute group
        , SCP_GLOBAL  // global declaration or ref
        , SCP_LOCAL   // local declaration
    };
};

XERCES_CPP_NAMESPACE_END

#endif
```

Performance difference with *Modules*

Conventional style: `-Ilib/xerces` …fix`.cpp` …`-lxerces`

```
#include <string>
#include <xerces/util/XMLString.hpp>

char* fix(const char* S)
{
    char* R = new char[std::strlen(S) * 2];
    XMLString::fixURI(S, R);
    return R;
}
```

Performance difference with *Modules*

Conventional style: `-Ilib/xerces` … `fix.cpp` … `-lxerces`

```
#include <string>
#include <xerces/util/XMLString.hpp>

char* fix(const char* S)
{
    char* R = new char[std::strlen(S) * 2];
    XMLString::fixURI(S, R);
    return R;
}
```

```
import Xerces;

char* fix(const char* S)
{
    /* ... same as above ... */
}
```

Evaluation

- Theoretical time complexity: $\mathcal{O}(\text{build time} + |\text{files}|^9)$.
  - Empirical: for *Apache Xerces*, 4 min. build, 25 sec. algorithm execution.

Evaluation

- Theoretical time complexity: $\mathcal{O}(\text{build time} + |\text{files}|^{9})$.
  - Empirical: for *Apache Xerces*, 4 min. build, 25 sec. algorithm execution.
- Modules made from existing code without touching it

Evaluation

- Theoretical time complexity: $\mathcal{O}(\text{build time} + |\text{files}|^9)$.
  - Empirical: for *Apache Xerces*, 4 min. build, 25 sec. algorithm execution.
- Modules made from existing code without touching it
- Not really feasible, it seems…

Evaluation

- Theoretical time complexity: $\mathcal{O}(\text{build time} + |\text{files}|^9)$.
  - Empirical: for *Apache Xerces*, 4 min. build, 25 sec. algorithm execution.
- Modules made from existing code without touching it
- Not really feasible, it seems…
- “TU semantics” approach $\longrightarrow$ coupling seems to break it
  - Collapse to **very few** modules per binary – almost *unity build*

Evaluation

- Theoretical time complexity: $\mathcal{O}(\text{build time} + |\text{files}|^9)$.
  - Empirical: for *Apache Xerces*, 4 min. build, 25 sec. algorithm execution.
- Modules made from existing code without touching it
- Not really feasible, it seems…
- "TU semantics" approach $\longrightarrow$ coupling seems to break it
  - Collapse to **very few** modules per binary – almost *unity build*
  - …or cross-binary modules

Requirements for “True Modularisation”

“The only solution here we can think of is actually to make people split up libraries.” (Manuel Klimek[11]*)*

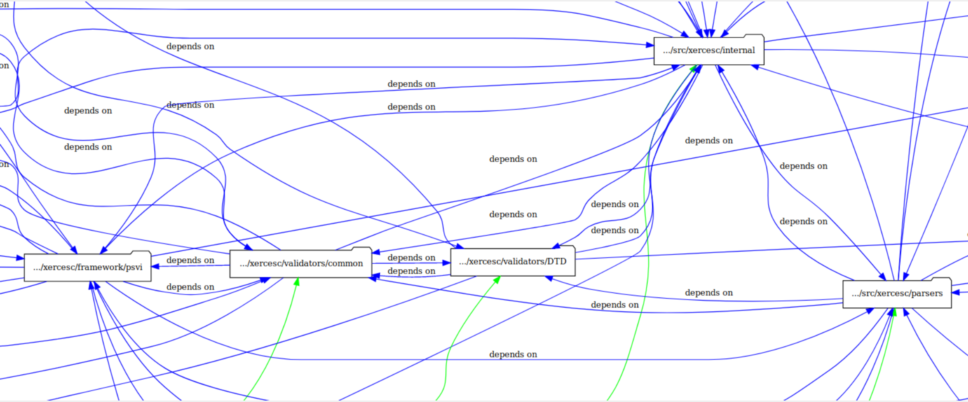
[11] CppCon 2016 talk, `youtube.com/watch?v=dHFNpBfemDI&t=38m48s`

Requirements for "True Modularisation"

"The only solution here we can think of is actually to make people split up libraries." (Manuel Klimek[11])

? ***What if the results ARE right?...***

[11] CppCon 2016 talk, `youtube.com/watch?v=dHFNpBfemDI&t=38m48s`



on
depends on
.../src/xercesc/internal
depends on
on
depends on
depends on
depends on
depends on
depends on
depends on
on
depends on
depends on
depends on
depends on
depends on
depends on
depends on
depends on
.../xercesc/framework/psvi
.../xercesc/validators/common
depends on
depends on
.../xercesc/validators/DTD
depends on
depends on
depends on
.../src/xercesc/parsers
depends on

Requirements for upgrading to Modules

In any case if upgrading to modules happens:

Requirements for upgrading to Modules

In any case if upgrading to modules happens:

- Either libs/modules end up staying same “size”/composition
  - Cue performance concerns…

Requirements for upgrading to Modules

In any case if upgrading to modules happens:

- Either libs/modules end up staying same “size”/composition
    - Cue performance concerns…
- Or we’ll have to break API…

Requirements for upgrading to Modules

In any case if upgrading to modules happens:

- Either libs/modules end up staying same “size”/composition
    - Cue performance concerns…
- Or we’ll have to break API…
- …and put in actual work

1. C++’s Compilation Model
   - Software technology issues
   - Performance drawbacks and possible solutions

2. C++ Modules
   - “How it should work?”
   - Traps and pitfalls (even) with Modules

3. The Wish for Automatic Modularisation
   - Formal overview
   - Case study — Apache Xerces
   - Evaluation of findings
   - Requirements for upgrading to Modules

4. **Summary**

References I

Gábor Horváth et al. “Implementation and Evaluation of Cross Translation Unit Symbolic Execution for C Family Languages”. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ICSE ’18. Gothenburg, Sweden: ACM, May 2018, pp. 428–428. ISBN: 978-1-4503-5663-3. DOI: `10.1145/3183440.3195041`.

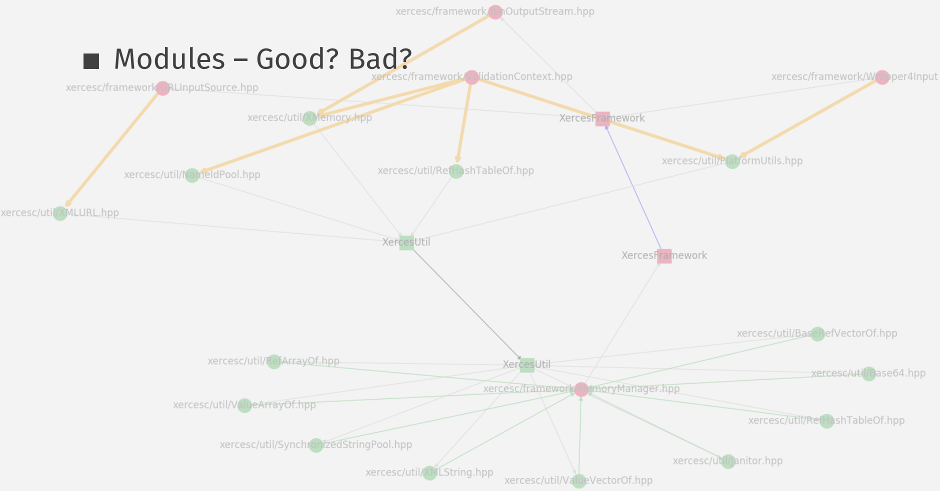
Tara Krishnaswamy. “Automatic Precompiled Headers: Speeding up C++ Application Build Times”. In: *Proceedings of the First Workshop on Industrial Experiences with Systems Software, WIESS 2000, October 22, 2000, San Diego, CA, USA*. Ed. by Dejan S. Milojicic. USENIX, Jan. 2000, pp. 57–66. ISBN: 1-880446-15-4.

References II

- Gábor Márton and Zoltán Porkoláb. “Unit Testing in C++ with Compiler Instrumentation and `friends`”. In: *Acta Cybernetica* 23.2 (Jan. 2017), pp. 659–686. DOI: `10.14232/actacyb.23.2.2017.14`.
- József Mihalicza. “How `#includes` Affect Build Time in Large Systems”. In: *Proceedings of the 8th International Conference on Applied Informatics (ICAI)*. Ed. by Attila Egri-Nagy et al. Vol. 2. 8. Eszterházy Károly College. BVB Nyomda és Kiadó Kft., Eger, Hungary, Jan. 2010, pp. 343–350. ISBN: 978-963-9894-72-3.
- Richard Smith. `P1103R3`: *Merging Modules*. Tech. rep. 3. JTC1/SC22/WG21 - Papers Mailing List, Feb. 2019.
- Zachary Turner. *Host is now dependency free*. `lldb-dev` mailing list. (accessed 20 Mar. 2019). Mar. 2019.

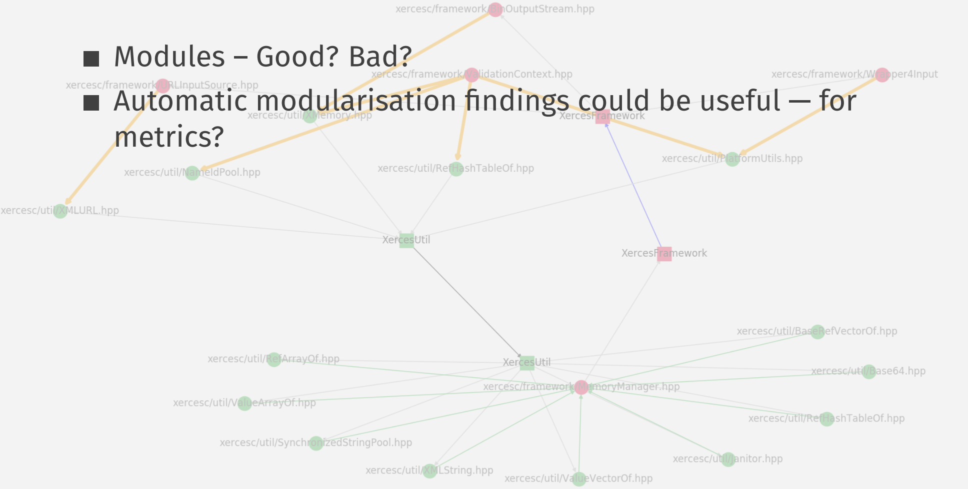
Conclusion

- Modules – Good? Bad?


xercesc/framework/BinOutputStream.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/framework/URLInputSource.hpp
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

Conclusion

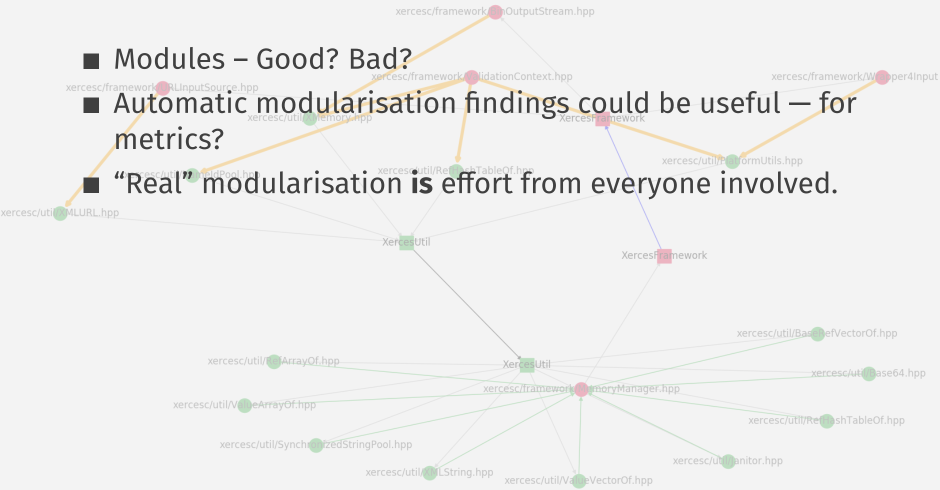
- Modules – Good? Bad?
- Automatic modularisation findings could be useful — for metrics?



xercesc/framework/BinOutputStream.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/framework/URLInputSource.hpp
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

Conclusion

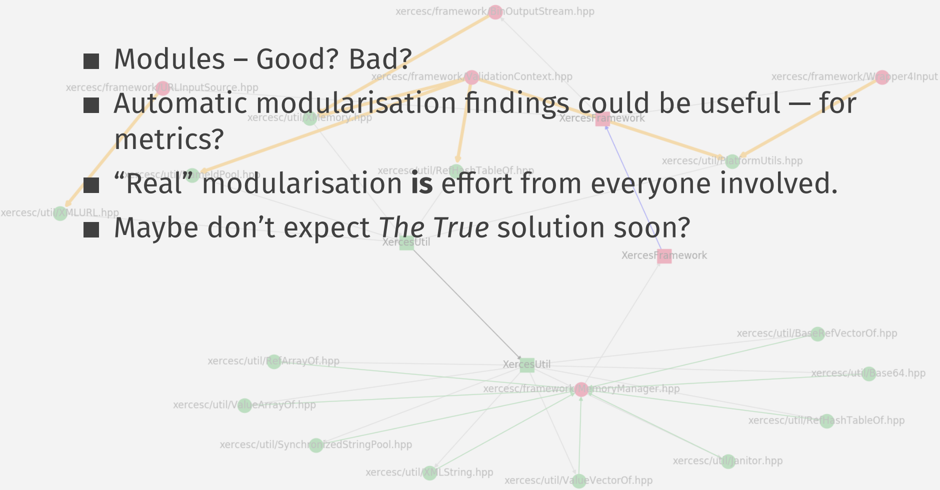
- Modules – Good? Bad?
- Automatic modularisation findings could be useful — for metrics?
- “Real” modularisation **is** effort from everyone involved.



xercesc/framework/BinOutputStream.hpp
xercesc/framework/URLInputSource.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

Conclusion

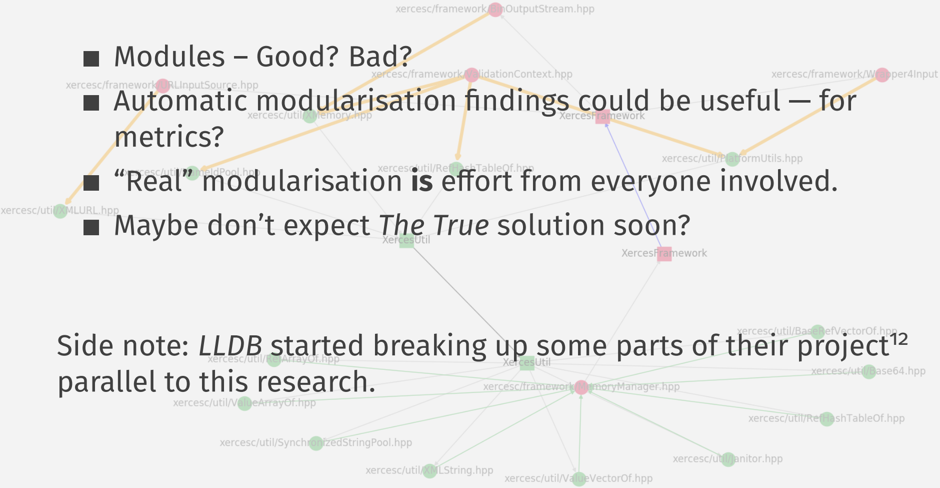
- Modules – Good? Bad?
- Automatic modularisation findings could be useful — for metrics?
- “Real” modularisation **is** effort from everyone involved.
- Maybe don’t expect *The True* solution soon?


xercesc/framework/BinOutputStream.hpp
xercesc/framework/URLInputSource.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

Conclusion

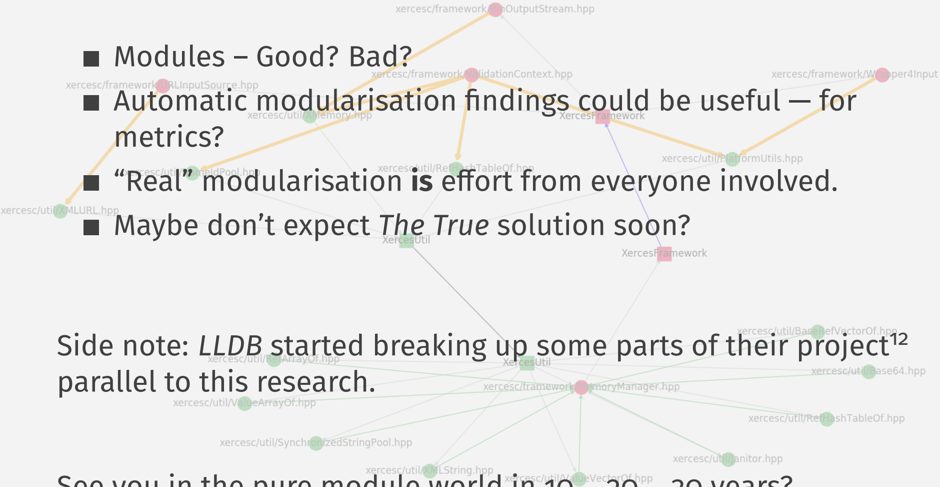
- Modules – Good? Bad?
- Automatic modularisation findings could be useful — for metrics?
- “Real” modularisation **is** effort from everyone involved.
- Maybe don’t expect *The True* solution soon?

Side note: *LLDB* started breaking up some parts of their project[12] parallel to this research.


xercesc/framework/BinOutputStream.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/framework/URLInputSource.hpp
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

[12] Turner, *Host is now dependency free.*

Conclusion


xercesc/framework/BinOutputStream.hpp
xercesc/framework/ValidationContext.hpp
xercesc/framework/Wrapper4Input
xercesc/framework/URLInputSource.hpp
xercesc/util/XMemory.hpp
XercesFramework
xercesc/util/PlatformUtils.hpp
xercesc/util/NameIdPool.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/XMLURL.hpp
XercesUtil
XercesFramework
xercesc/util/BaseRefVectorOf.hpp
xercesc/util/RefArrayOf.hpp
XercesUtil
xercesc/util/Base64.hpp
xercesc/framework/MemoryManager.hpp
xercesc/util/ValueArrayOf.hpp
xercesc/util/RefHashTableOf.hpp
xercesc/util/SynchronizedStringPool.hpp
xercesc/util/Janitor.hpp
xercesc/util/XMLString.hpp
xercesc/util/ValueVectorOf.hpp

- Modules – Good? Bad?
- Automatic modularisation findings could be useful — for metrics?
- “Real” modularisation **is** effort from everyone involved.
- Maybe don’t expect *The True* solution soon?

Side note: *LLDB* started breaking up some parts of their project[12] parallel to this research.

See you in the pure module world in 10. . . 20. . . 30 years?

[12] Turner, *Host is now dependency free.*