C++ MODULES

HSAE - Wuhan · 张轶 · github.com/mutouyun

2012

N3347

Modules in C++ (v6)

2014

N4047

A Module System for C++

2016

P0142

A Module System for C++ (v4)

2017

N4637 - N4681

Working Draft, Extensions to C++ for Modules

ISO/IEC 14882:2017

Programming languages -- C++

2018

N4720

Module TS

P0947/P0986

Another take on Modules (Atom)

P1103/P1156/P1180

Merging Modules

WHY MODULE

HEADERS

- 脆弱的文本展开
- 内部细节的意外导出
- 大量的重复处理（N x M）

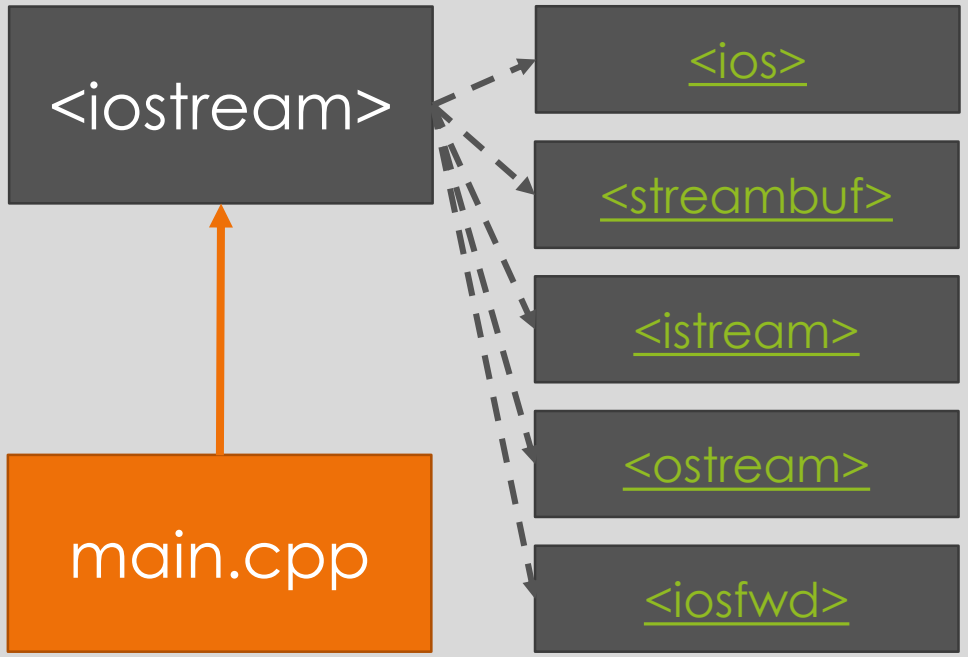
……

HELLO WORLD

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

头文件展开

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```



HELLO WORLD

```
#define __s
#include <iostream>
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

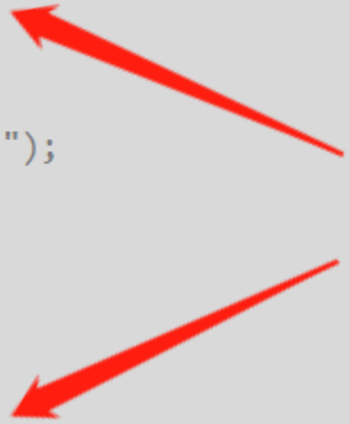
<WINDOWS.H>

```cpp
#include <iostream>
#include <limits>

#include <Windows.h>

int main() {
    std::cout << std::numeric_limits<int>::max() << std::endl;
    /*
       warning C4003: not enough actual parameters for macro 'max'
       error C2589: '(' : illegal token on right side of '::'
       error C2143: syntax error : missing ')' before '::'
       error C2059: syntax error : ')'
    */
    return 0;
}
```

ODR (ONE DEFINITION RULE)

main.cpp :

```cpp
#include "say.h"

int main() {
    say("hello world!");
    return 0;
}
```



Same?

func.cpp :

```cpp
#include "say.h"

void func() {
    say("hello func!");
}
```

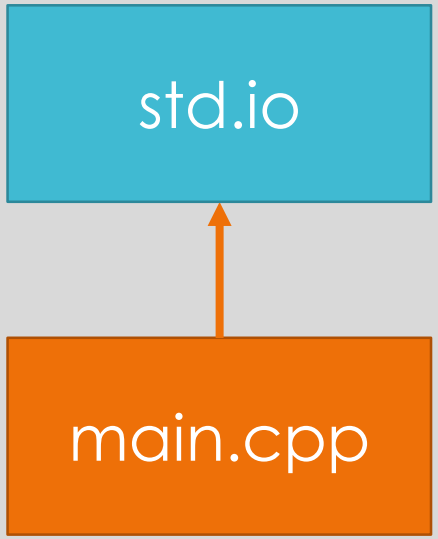
HEADER => MODULE

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

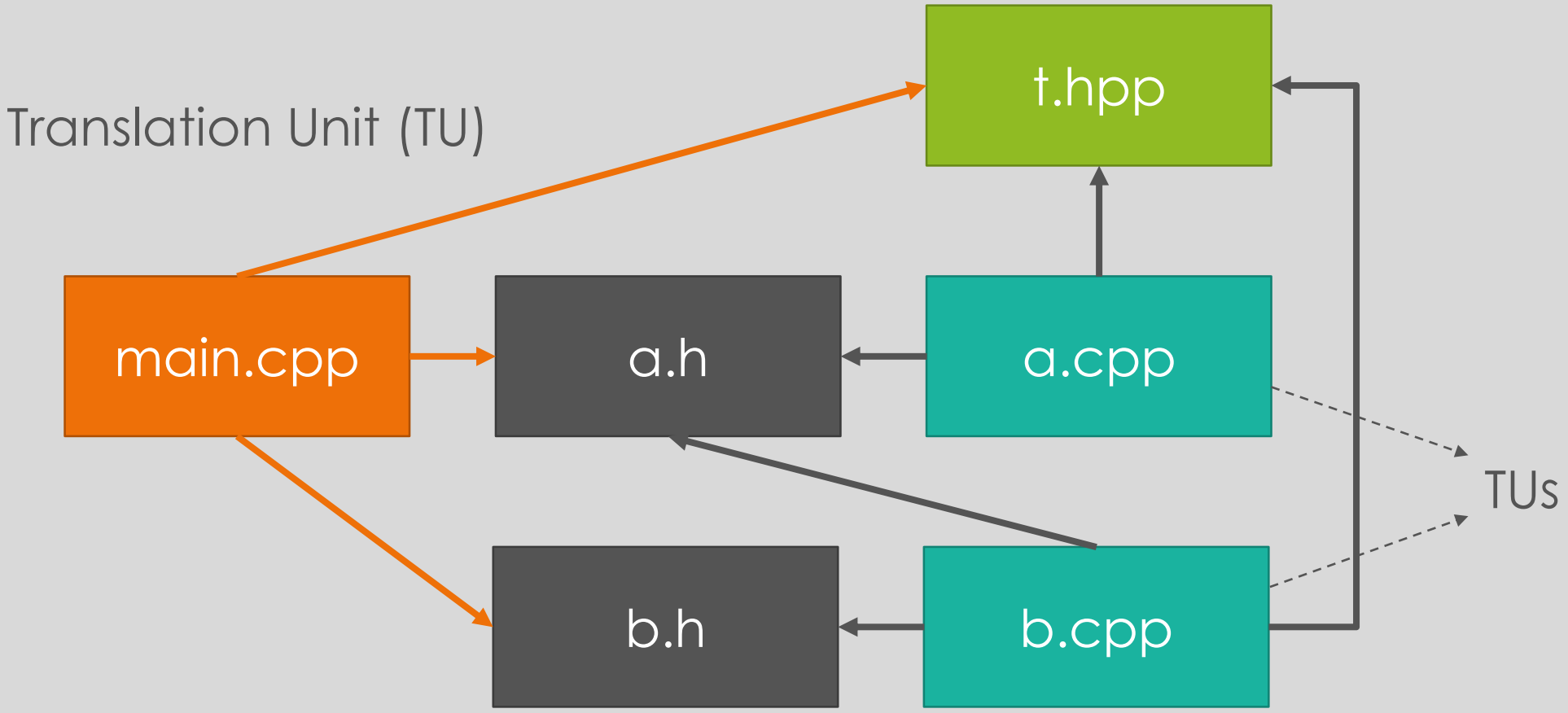
```cpp
import std.io;
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

HEADER => MODULE

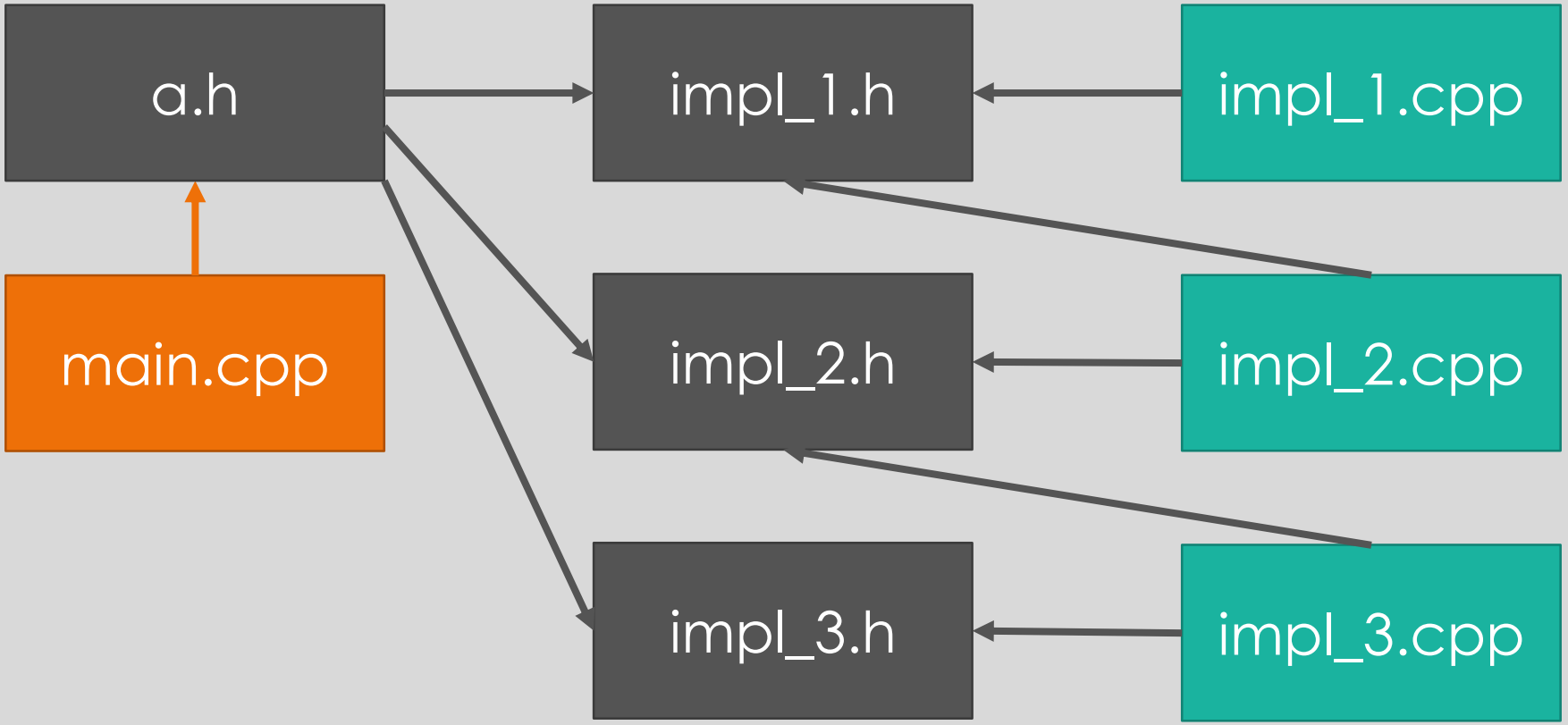
```
import std.io;
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```



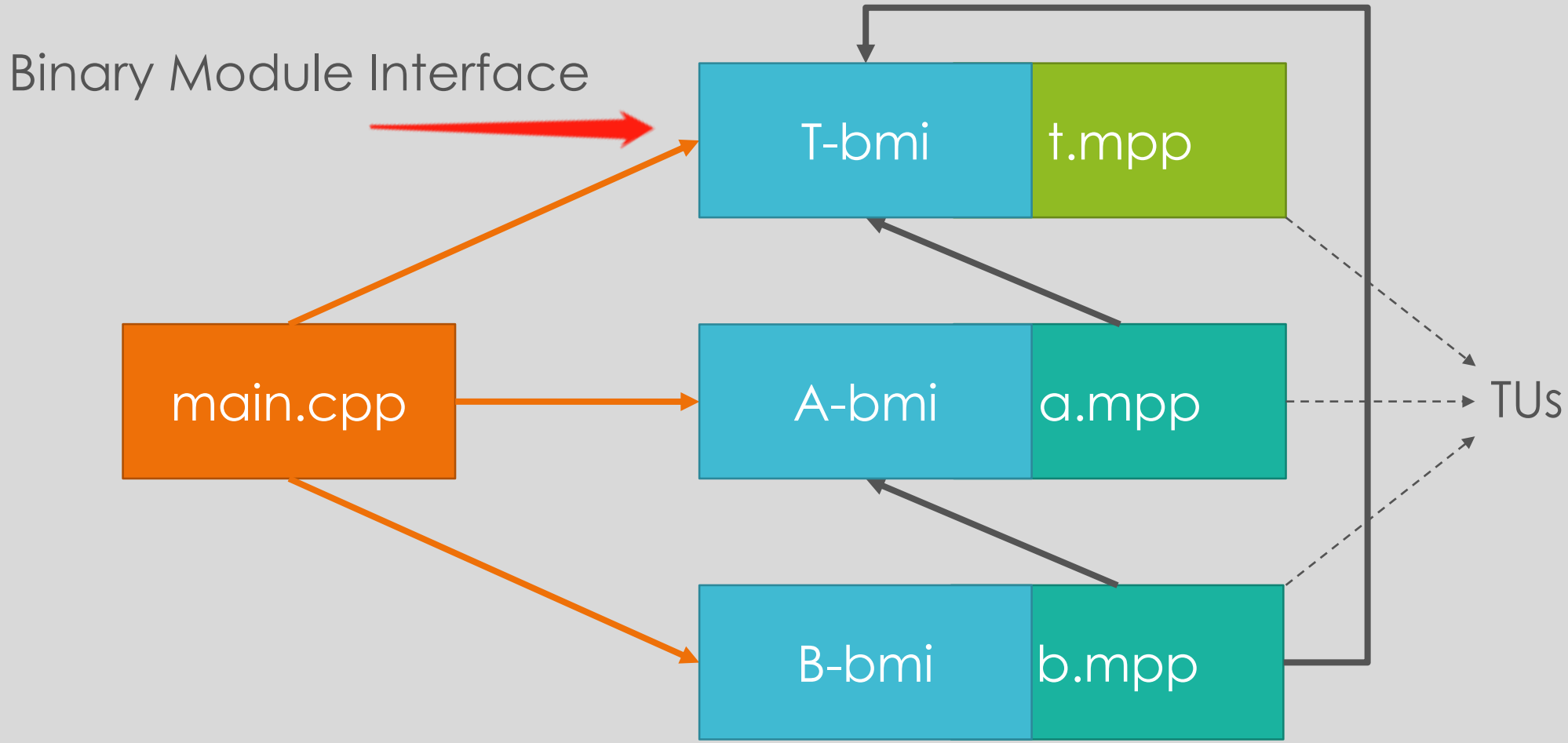
WORLD WITH HEADER



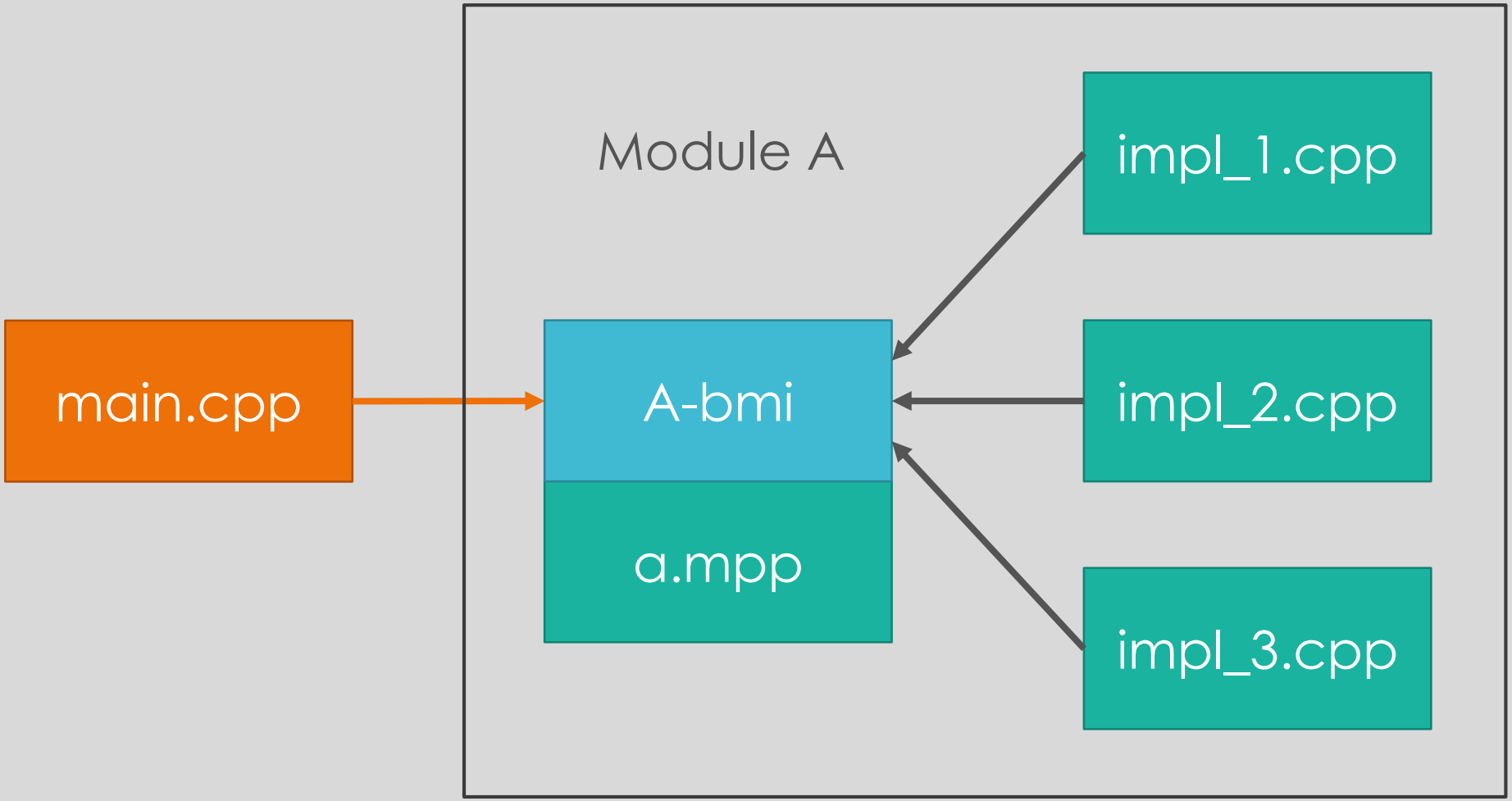
WORLD WITH HEADER



WORLD WITH MODULE



WORLD WITH MODULE

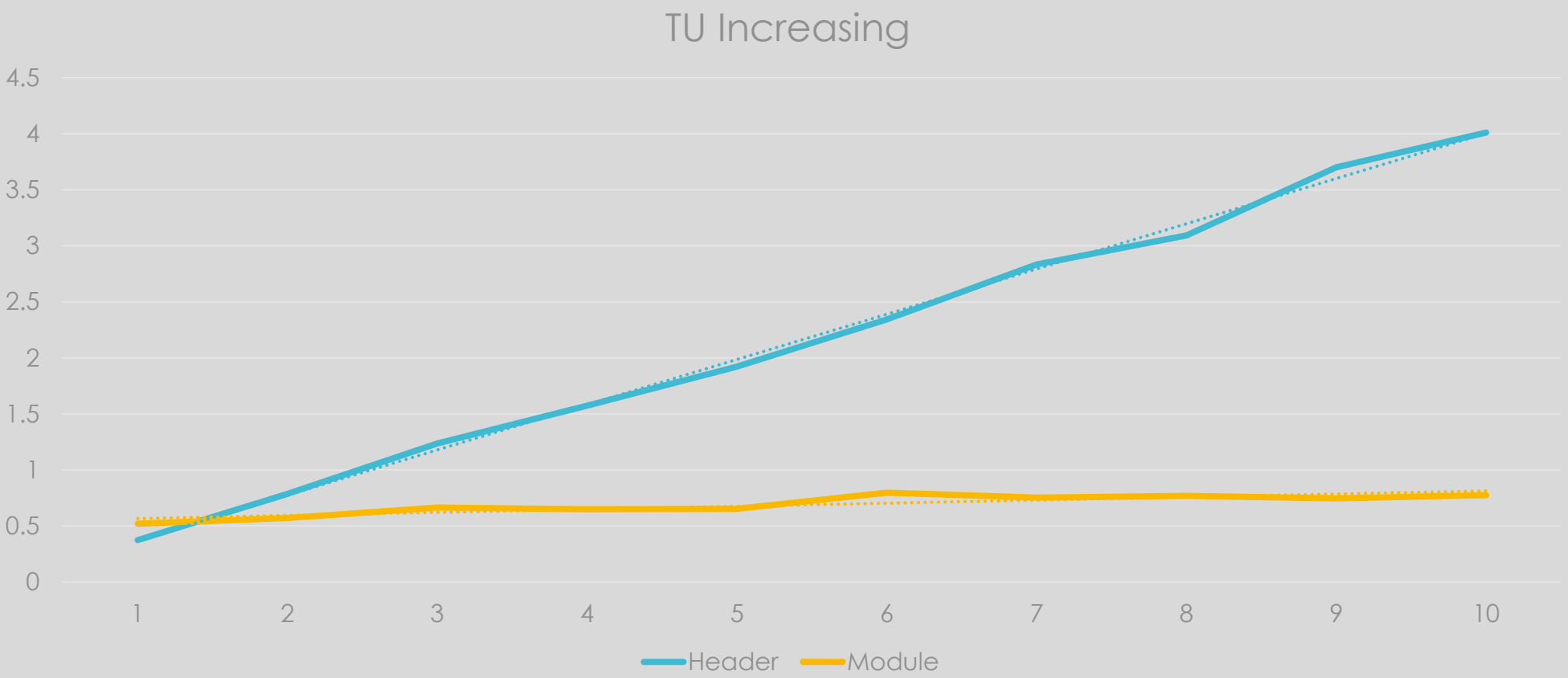


MODULES

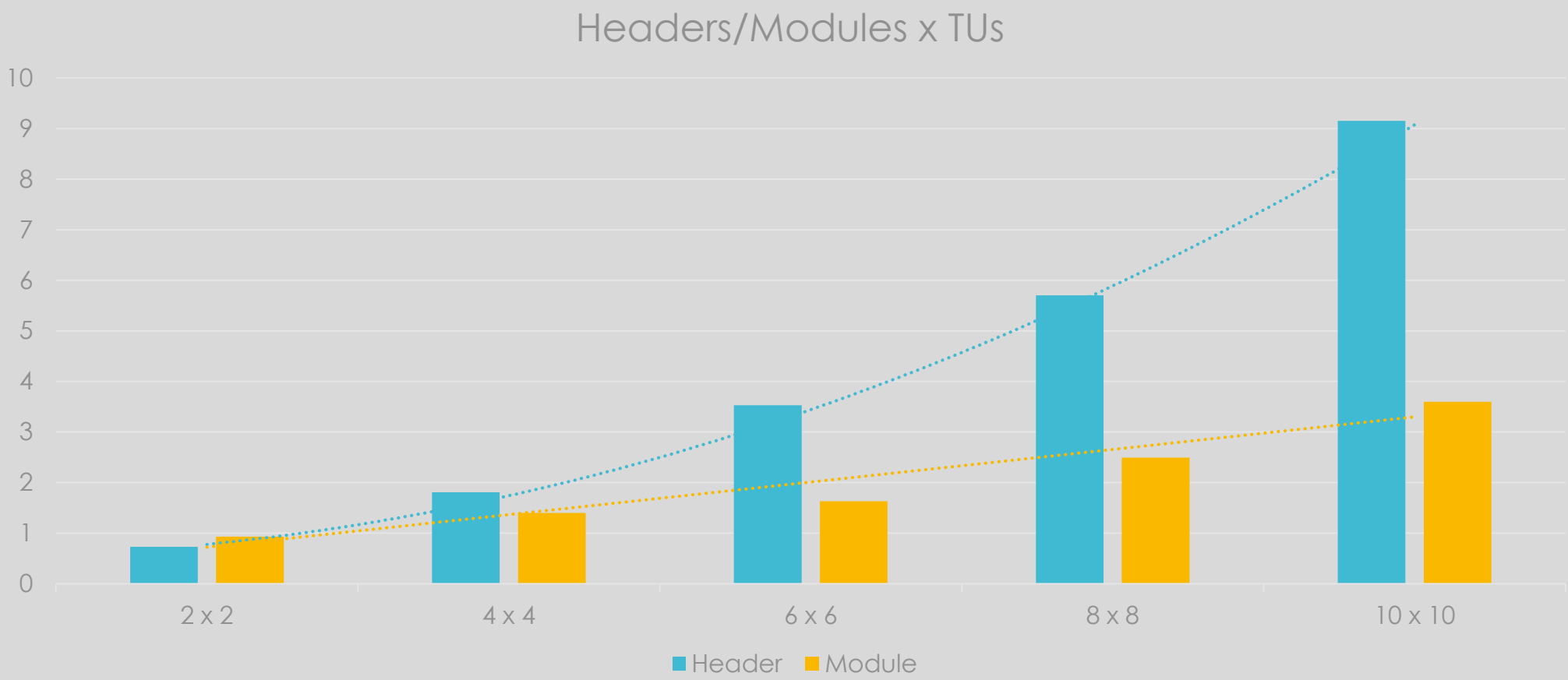
- 为编译单元（Translation Unit）提供二进制符号信息
- ODR（One Definition Rule）
- 基于依赖关系（Dependency Graph）的构建
- 避免大量的重复解析-编译过程

……

PERFORMANCE



PERFORMANCE




GELIVABLE

HOW TO USE

SAY HELLO

main.cpp :

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

SAY HELLO

main.cpp :

```
import std.io;
using namespace std;

int main() {
    cout << "hello world!" << endl;
    return 0;
}
```

自定义MODULE

module interface unit

hello.mpp：

```
export module hello;
import std.io;

export namespace hello {
    void say_hello() {
        std::cout << "hello world!" << std::endl;
    }
}
```

main.cpp：

```
import hello;

int main() {
    hello::say_hello();
    return 0;
}
```

自定义MODULE

hello.mpp：

```
export module hello;
import std.io;

export namespace hello {
    void say_hello() {
        std::cout << "hello world!" << std::endl;
    }
}
```

main.cpp：

```
import hello; // no import std.io

int main() {
    hello::say_hello();
    std::cout << "hello!" << std::endl; // error
    return 0;
}
```

自定义MODULE

hello.mpp：

```
export module hello;
export import std.io;

export namespace hello {
    void say_hello() {
        std::cout << "hello world!" << std::endl;
    }
}
```

main.cpp：

```
import hello; // import std.io

int main() {
    hello::say_hello();
    std::cout << "hello!" << std::endl; // ok
    return 0;
}
```

MODULE LINKAGE

hello.mpp :

```
export module hello;
export import std.io;

/* module linkage */
namespace hello {
    void say_hi   () { std::cout << "hello hi!" << std::endl; }
}

/* external linkage */
export namespace hello {
    void say_hello() { std::cout << "hello world!" << std::endl; }
    void say_xz   () { std::cout << "hello xz!"    << std::endl; }
}
```

MODULE LINKAGE

hello.mpp :

```
export module hello;
export import std.io;

/* module linkage */
namespace hello {
    void say_hi();
}

/* external linkage */
export namespace hello {
    void say_hello();
    void say_xz   ();
}
```

MODULE LINKAGE

module implementation unit

hello_impl.cpp :

```
module hello;

namespace hello {
    void say_hi   () { std::cout << "hello hi!"    << std::endl; }
    void say_hello() { std::cout << "hello world!" << std::endl; }
    void say_xz   () { std::cout << "hello xz!"    << std::endl; }
}
```

main.cpp :

```
import hello;

int main() {
    hello::say_hello(); // ok
    hello::say_xz();    // ok
    hello::say_hi();    // error
    return 0;
}
```

MODULE LINKAGE

hello.mpp：

修改这里会……？

```
export module hello;
export import std.io;

/* module linkage */
namespace hello {
    void say_hi   () { std::cout << "hello hi!" << std::endl; }
}

/* external linkage */
export namespace hello {
    void say_hello() { std::cout << "hello world!" << std::endl; }
    void say_xz   () { std::cout << "hello xz!"    << std::endl; }
}
```

MODULE PARTITIONS

hello.mpp：

```
export module hello;
export import std.io;

/* module linkage */
namespace hello {
    void say_hi();
}

/* external linkage */
export namespace hello {
    void say_hello();
    void say_xz   ();
}
```

.....如果有很多的实体

MODULE PARTITIONS

module implementation partition

hello_hi.cpp :

```
module hello:hi;

namespace hello {
    void say_hi() { std::cout << "hello hi!" << std::endl; }
}
```

hello_impl.cpp :

```
module hello;
import std.io;
import :hi; // say_hi

namespace hello {
    void say_hello() {
        std::cout << "hello world!" << std::endl;
        say_hi();
    }
}
```

MODULE PARTITIONS

module interface partition

hello_xz.mpp :

```
export module hello:xz;

export namespace hello {
    void say_xz() { std::cout << "hello xz!"    << std::endl; }
}
```

re-export is necessary

hello.mpp :

```
export module hello;
export import :xz; // say_xz

export namespace hello {
    void say_hello();
}
```

MODULES的循环依赖

hello => mod => hello

MODULES的循环依赖

```
module hello:impl;
import std.io;
import mod;

void inner() {
    std::cout << "hello world! data: "
              << mod::foo() << std::endl;
}
```

```
export module mod;
import hello;

export namespace mod {
    int foo {
        hello::data__ = 123;
        return hello::data__;
    }
}
```

hello:impl => mod => hello

GLOBAL MODULE FRAGMENT

hello_impl.cpp :

```
module hello;
#include <iostream> // error

void inner();

namespace hello {
    int data__;
    void say_hello() { ::inner(); }
}

void inner() {
    int data = hello::data__;
    std::cout << "hello world! data: " << data << std::endl;
}
```

GLOBAL MODULE FRAGMENT

hello_impl.cpp :

```
module;
#include <iostream>
module hello;

void inner();

namespace hello {
    int data__;
    void say_hello() { ::inner(); }
}

void inner() {
    int data = hello::data__;
    std::cout << "hello world! data: " << data << std::endl;
}
```

GLOBAL MODULE FRAGMENT

hello.mpp :

```
module;
#include <string>
export module hello;

export namespace hello {
    extern int data__;
    void say_hello();
}
```

main.cpp :

```
import hello;
#include <iostream>

int main() {
    hello::data__ = 123;
    hello::say_hello();
    std::cout << "hello world!" << std::endl;
    return 0;
}
```

Different view of global module

LEGACY HEADER UNITS

hello_impl.cpp :

```
module hello;
import <iostream>; // import everything, including macros

void inner();

namespace hello {
    int data__;
    void say_hello() { ::inner(); }
}

void inner() {
    int data = hello::data__;
    std::cout << "hello world! data: " << data << std::endl;
}
```

LEGACY HEADER UNITS

legacy.hpp :

```
#pragma once

inline void legacy() {
}

#define LEGACY
```

hello_impl.cpp :

```
module hello;
import "legacy.hpp";

#if defined(LEGACY)
void func() {
    legacy();
}
#endif
```

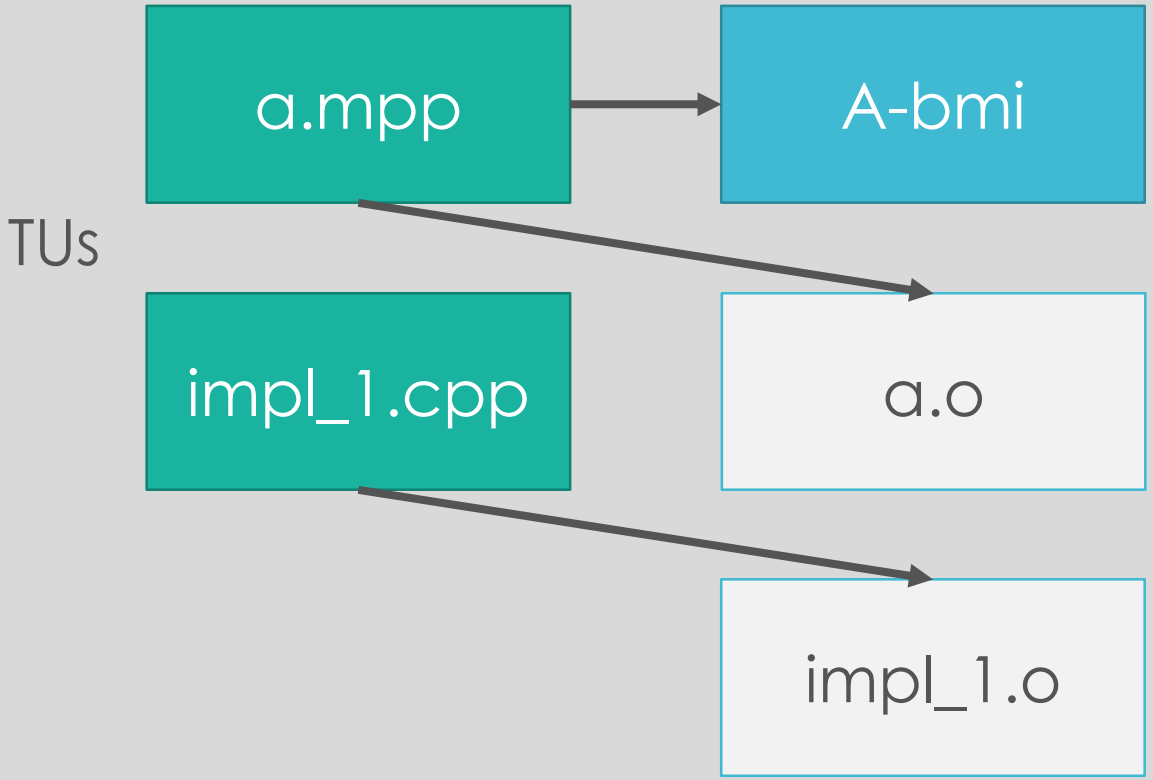
LEGACY HEADER UNITS

hello.mpp :

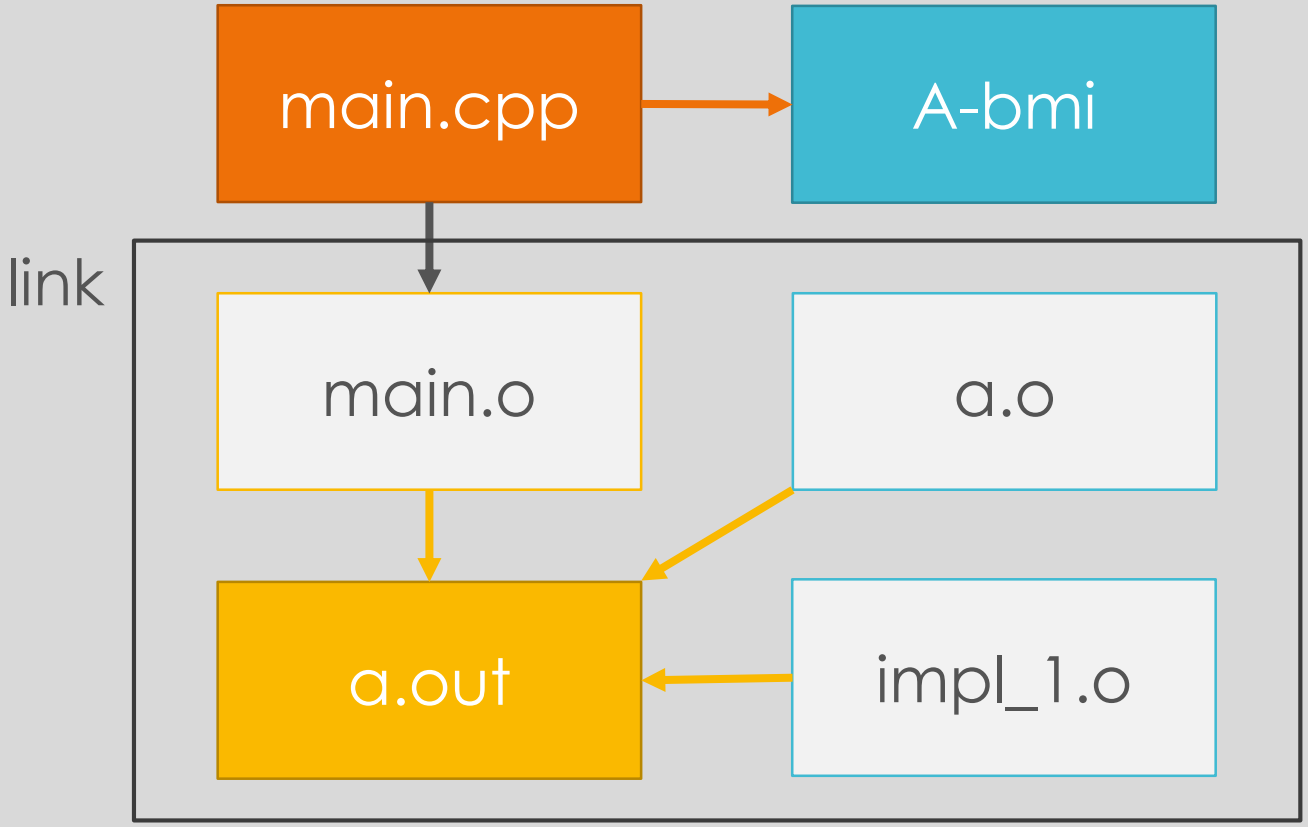
```
export module hello;
export import <iostream>; // export everything, except macros

export namespace hello {
    extern int data__;
    void say_hello();
}
```

HOW IT WORKS

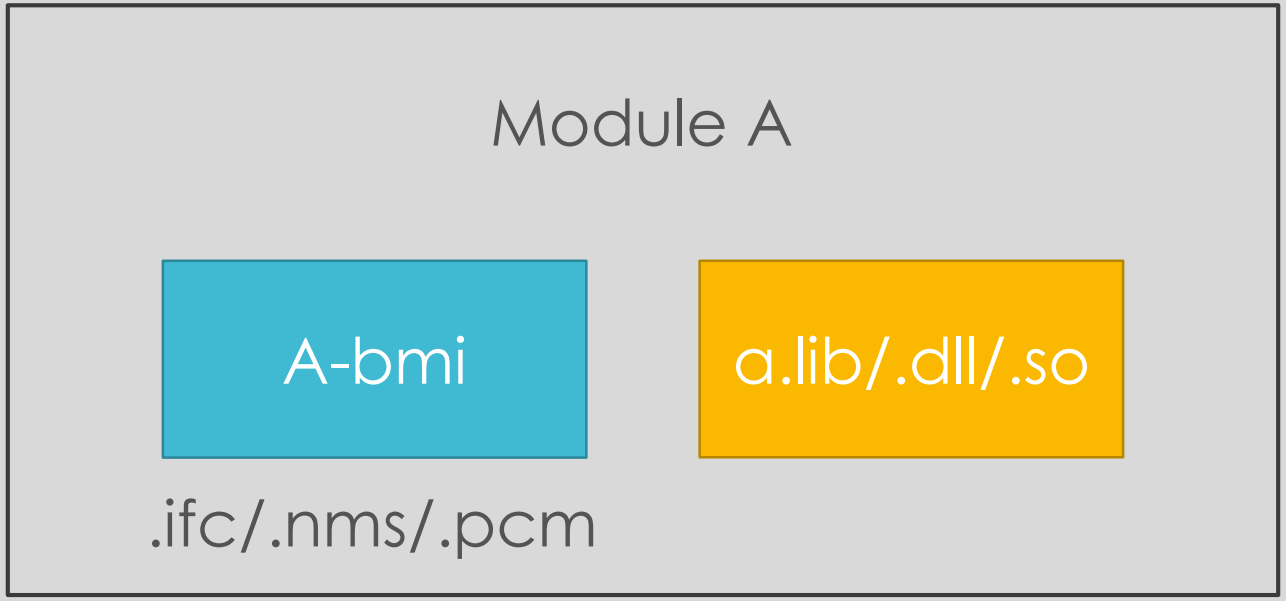
COMPILE



LINK



RELEASE



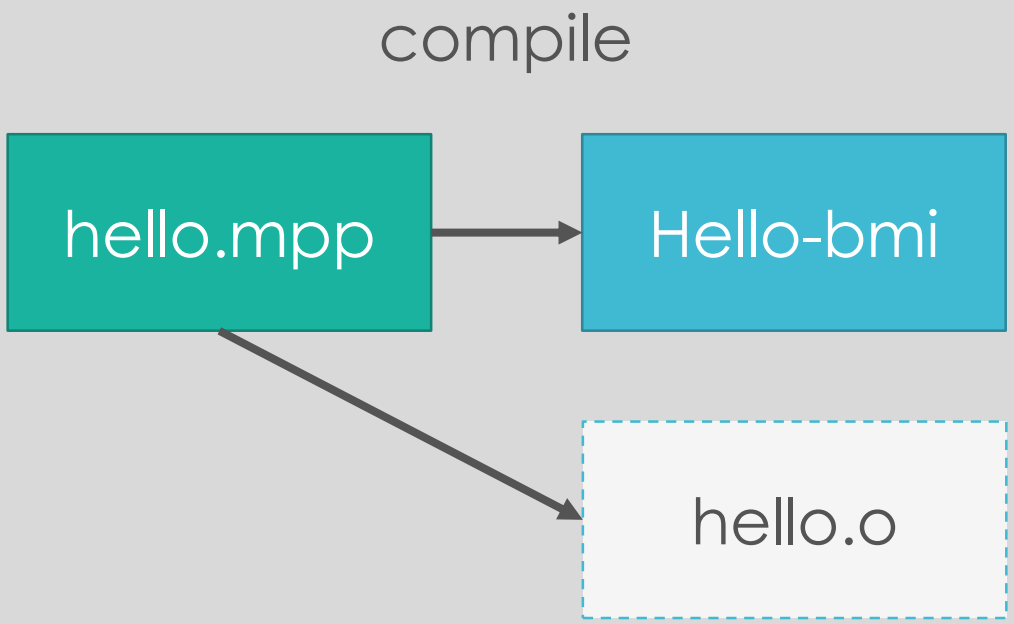
No Header

BMI & .O

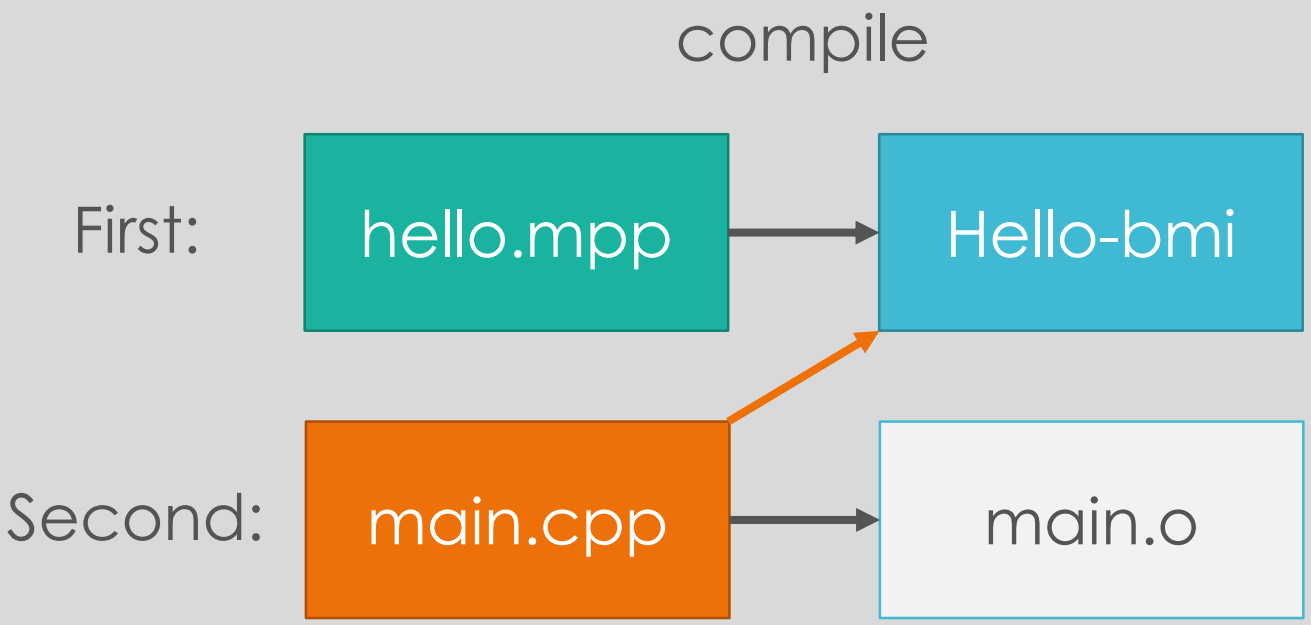
hello.mpp :

```
export module hello;

export namespace hello {
    include void func {
        // ...
    }

    template <int N>
    struct square { constexpr static int value = N * N; };

    class foo {
        int a_ = 123;

    public:
        int bar() const { return a_; }
    };
}
```

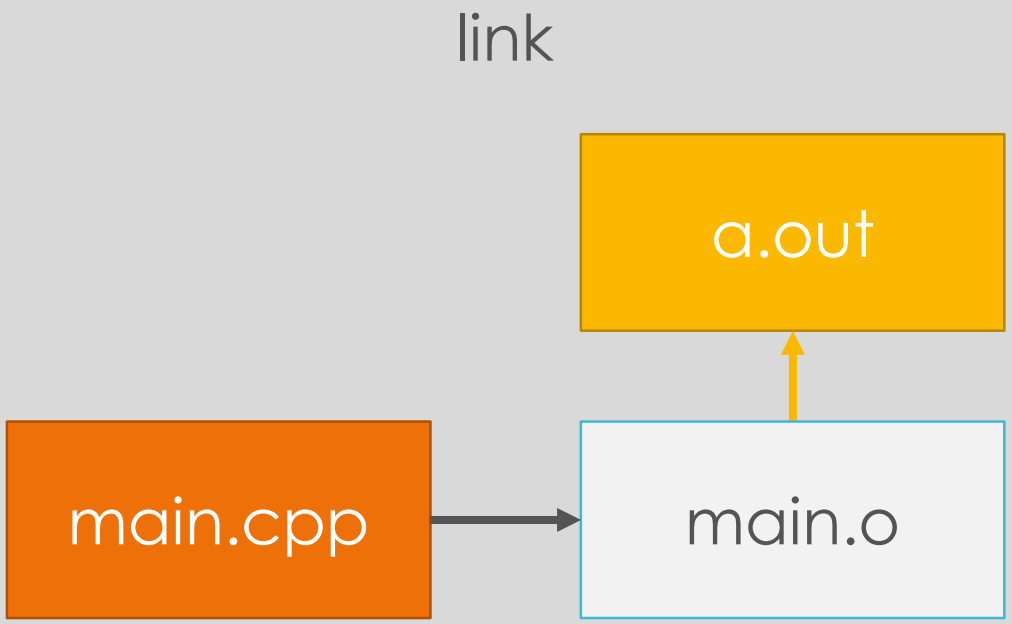
BMI & .O



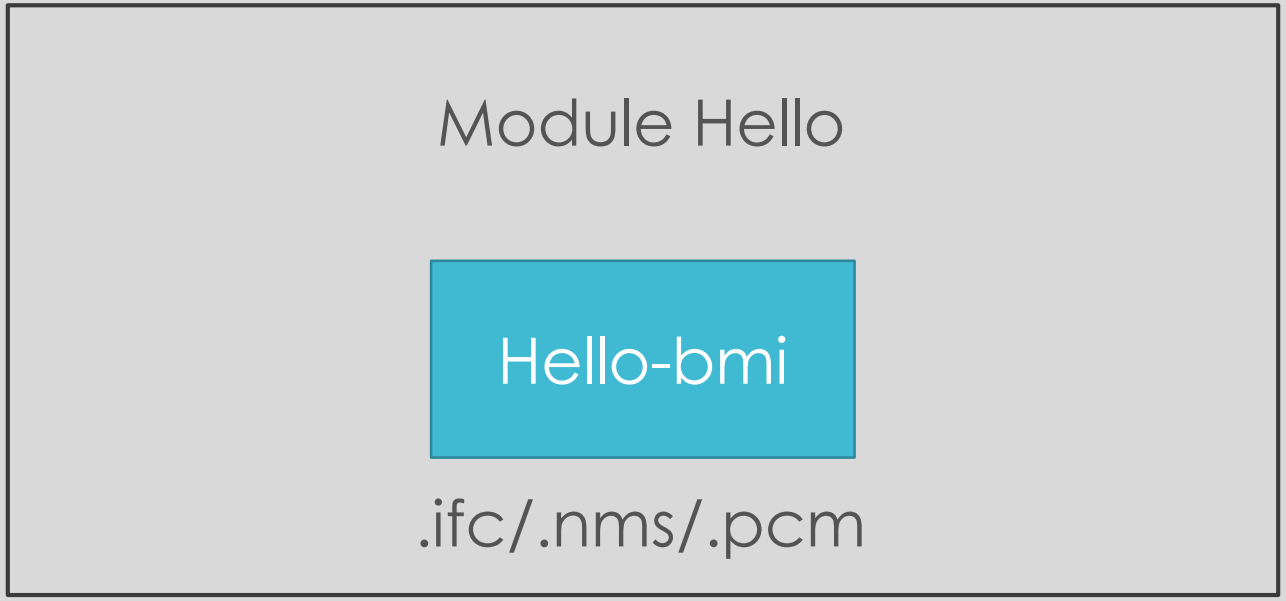
BMI & .O



BMI & .O



RELEASE HELLO



NOTES

‘SUBMODULE’

main.cpp :

```
import std.io;
import std.thread;
import std.string;

int main() {
    // ...
}
```

‘SUBMODULE’

它们之间没什么关系

main.cpp：

```
import std.io;
import std.thread;
import std.string;

int main() {
    // ...
}
```

‘submodule’是另一个不同的module

‘SUBMODULE’

hello.mpp：

```
export module hello;
// export import :xz;
export import hello.xz // say_xz

export namespace hello {
    void say_hello();
}
```

这里是另一个独立的module

NAMESPACE

hello.mpp :

```
export module hello;
export void say_hello();
```

main.cpp :

```
import hello;

int main() {
    hello::say_hello(); // error
    say_hello();        // ok
    return 0;
}
```

NAMESPACE

hello.mpp：

```
export module hello;
export void say_hello();
```

main.cpp：

```
import hello;

int main() {
    hello::say_hello(); // error
    say_hello();        // ok
    return 0;
}
```

module并不引入namespace

NAMESPACE

hello.mpp：

```
export module hello;
export import std.io;

export namespace hello {
    void say_hello() {
        std::cout << "hello world!" << std::endl;
    }
}

export namespace {
    void anonymous() {

    }
}
```

能导出么？

类的PRIVATE成员 & PIMPL

hello.mpp：

```
export module hello;

export namespace hello {
    class say {
    public:
        say(int data);
        void hello();

    private:
        int data_;
    };
}
```

看得到么？

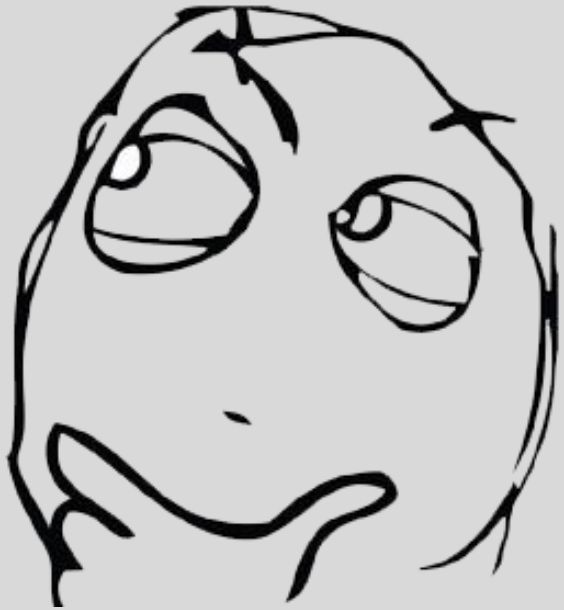
COMPILE-TIME

```
export module mod;

export template <int N, int M, int Loops = 1000000>
void test_prod_cons() {…
}
```

```
export template <int N>
struct foo {};

export inline void test_performance(foo<1>, foo<1>) {
    test_prod_cons<1, 1>();
}

export template <int N>
void test_performance(foo<N>, foo<1>) {
    test_performance(foo<N - 1>{}, foo<1>{});
    test_prod_cons<N, 1>();
};

export template <int N, int M>
void test_performance(foo<N>, foo<M>) {
    test_performance(foo<N>{}, foo<M - 1>{});
    test_prod_cons<N, M>();
};
```

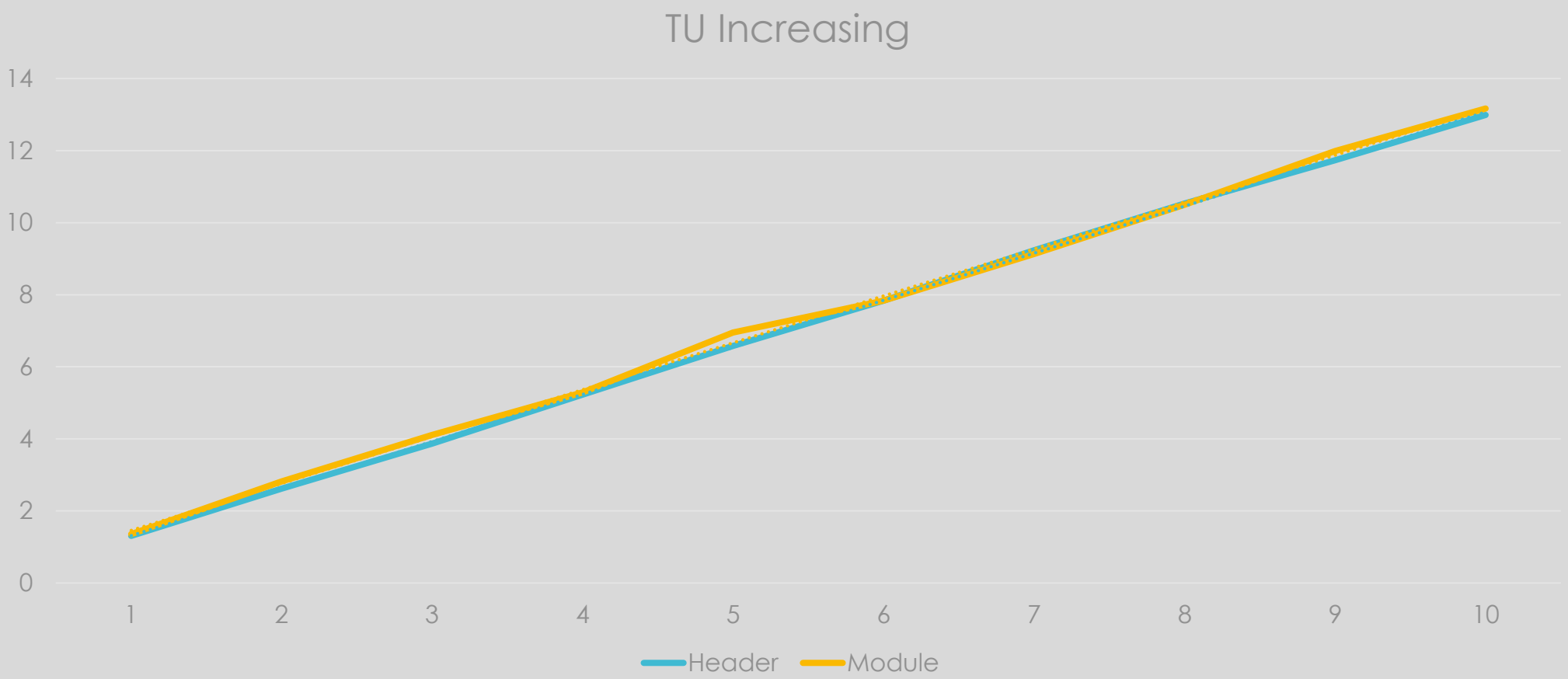
COMPILE-TIME

```
import mod;

int main(void) {
    test_performance(foo<1000>{}, foo<1000>{});
    return 0;
}
```



COMPILE-TIME

COMPILE-TIME



COMPILE-TIME

module mod interface unit：

```
export extern template void test_performance(foo<1000>, foo<1000>);
```

BUILDING

MSVC 2017 VERSION 15.9

```
cl /experimental:module /std:c++latest /c hello.ixx
cl /experimental:module /std:c++latest /c hello_impl.cxx
cl /experimental:module /std:c++latest /c mod.ixx
cl /experimental:module /std:c++latest main.cxx *.obj
```

GCC

```
svn://gcc.gnu.org/svn/gcc/branches/c++-modules
```

GCC

```
g++ -c -fmodule-legacy legacy.hpp
g++ -c -fmodules-ts -x c++ hello.mxx
g++ -c -fmodules-ts hello_impl.cxx
g++ -c -fmodules-ts -x c++ mod.mxx

g++ -fmodules-ts -o hello \
    hello.o hello_impl.o mod.o main.cxx
```

CLANG 8

```
svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
cd llvm/tools
svn co http://llvm.org/svn/llvm-project/cfe/trunk clang
```

CLANG 8

```
clang++ -fmodules-ts --precompile hello.cppm
clang++ -fmodules-ts -c hello.pcm
clang++ -fmodules-ts -fmodule-file=hello.pcm -c hello_impl.cxx
clang++ -fmodules-ts --precompile -x c++-module \
        -fprebuilt-module-path=. mod.mxx
clang++ -fmodules-ts -c mod.pcm

clang++ -fmodules-ts -fprebuilt-module-path=. \
        -o hello hello.o hello_impl.o mod.o main.cxx
```

BUILD2

`build2` | C++ Build Toolchain

https://build2.org/

BUILD2

To build a `hello` executable from these files we can write the following `buildfile`:

```
exe{hello}: cxx{driver} {mxx cxx}{hello}
```

Or, if you prefer to use wildcard patterns:

```
exe{hello}: {mxx cxx}{*}
```

REFERENCE

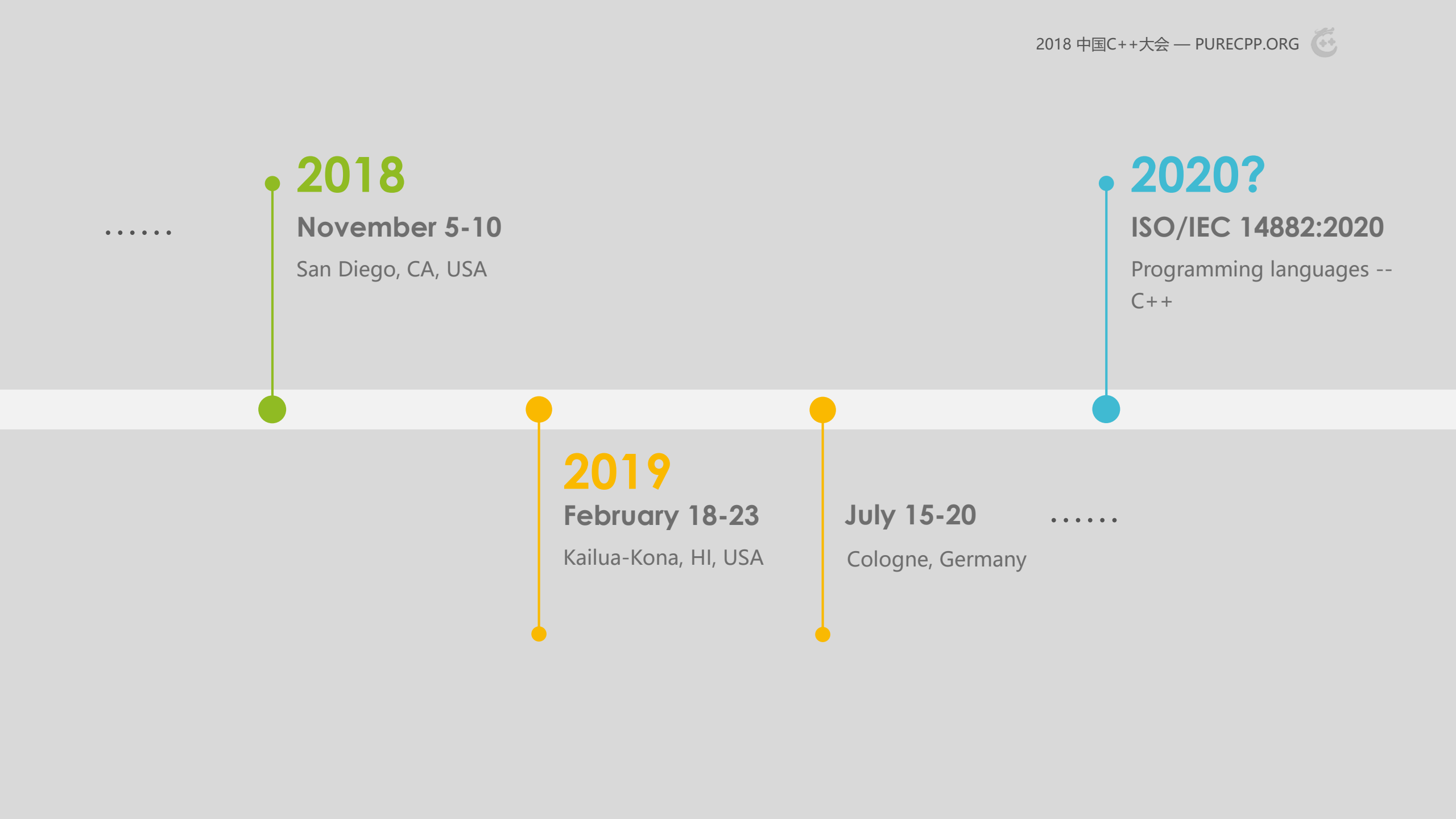
- MSVC
  https://blogs.msdn.microsoft.com/vcblog/2018/11/27/better-template-support-and-error-detection-in-c-modules-with-msvc-2017-version-15-9/
- GCC
  https://gcc.gnu.org/wiki/cxx-modules
- Clang
  https://clang.llvm.org/docs/Modules.html
- build2
  https://build2.org/build2/doc/build2-build-system-manual.xhtml

UNDERWAY

- P1213R0：Global Module Fragment is Unnecessary
- P1203R0：Modular main()
- P1303R0：Inline Module Partitions
- P1300R0：Remember the FORTRAN

……

Feature	Status	Depends On	Current Target (Conservative Estimate)	Current Target (Optimistic Estimate)
Concepts	Concepts TS v1 published and merged into C++20		C++20	C++20
Ranges	**Ranges TS v1 published and merged into C++20**	Concepts	**C++20**	C++20
Contracts	Merged into C++20		C++20	C++20
Modules	**Merged design approved for C++20**		**C++23**	C++20
Coroutines	Coroutines TS v1 published		C++23	C++20
Executors	**Proposed v1 design approved for C++20**		TS in C++20 timeframe and IS in C++23	C++20
Networking	Networking TS v1 published	Executors	**C++26**	**C++23**
Futures	Proposal	Executors	TS in C++23 timeframe and IS in C++26	TS in C++20 timeframe and IS in C++23
Reflection	Draft Reflection TS v1 is out for ballot		TS in C++20 timeframe and IS in C++26	TS in C++20 timeframe and IS in C++23


2018
November 5-10
San Diego, CA, USA
2019
February 18-23
Kailua-Kona, HI, USA
July 15-20
Cologne, Germany
2020?
ISO/IEC 14882:2020
Programming languages -- C++

THANKS！