Филипп Хандельянц

Лекция 10/12

Сборка C/C++ проектов и ее оптимизация

Докладчик

Хандельянц

Филипп Александрович

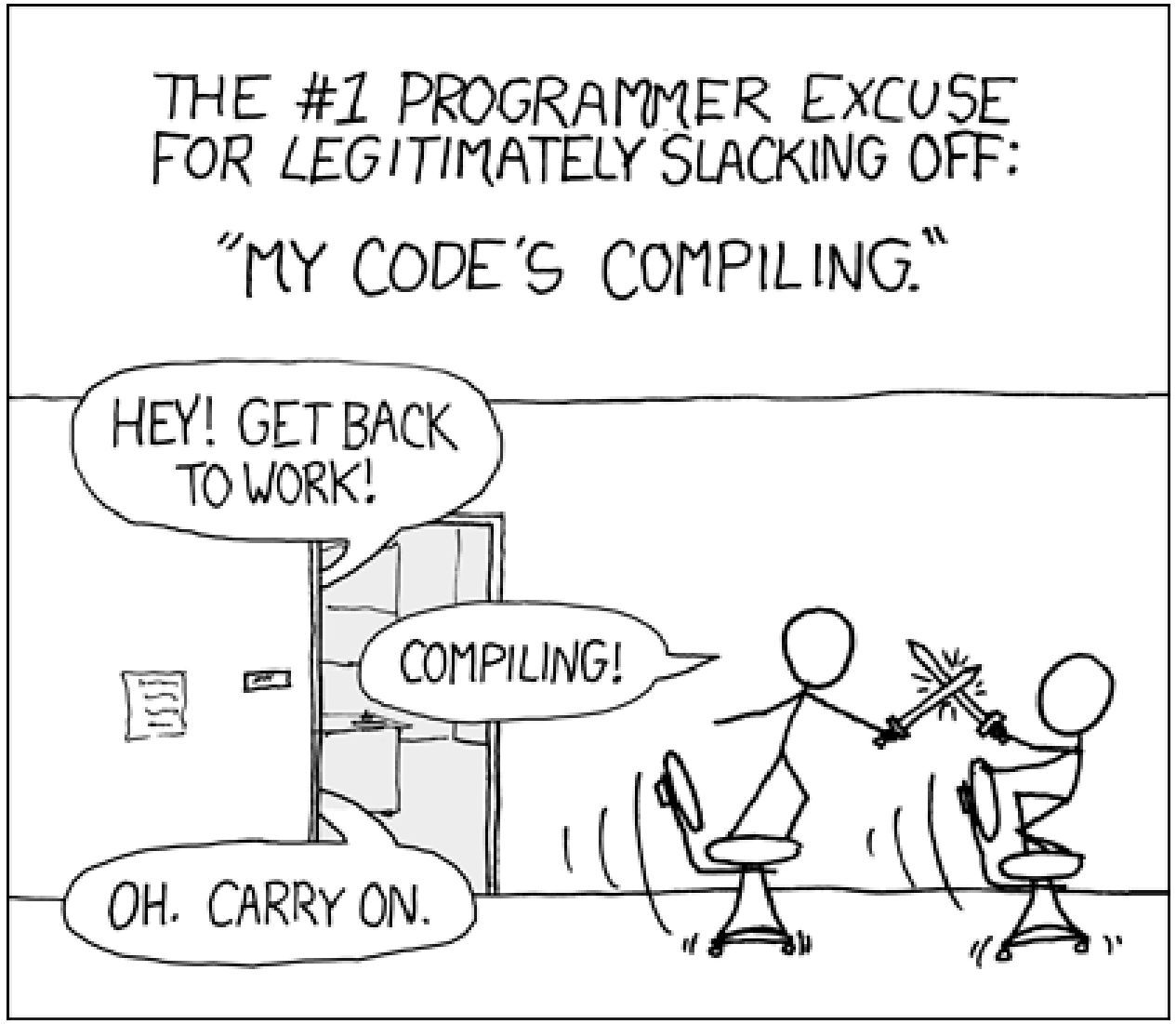
- Ведущий разработчик в команде PVS-Studio (C++/C#)
- 3 года участвую в разработке ядра C++ анализатора
- Автор статей о проверке open source-проектов

Why should we care?

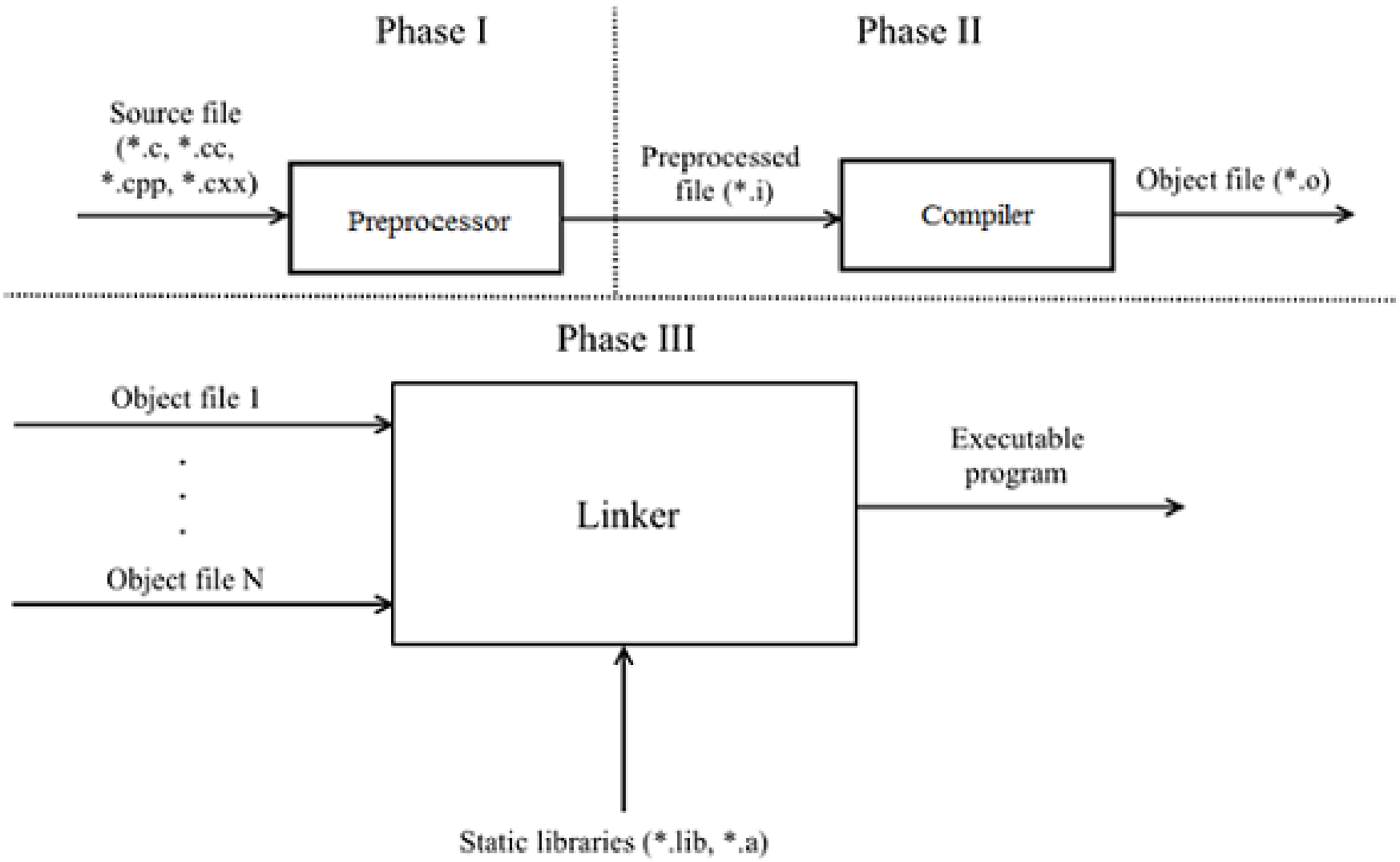
- Размеры проектов неуклонно растут:
  - Ядро Linux v1.0.0 : 177 KLOC
  - Ядро Linux v5.1-rc2: ~21 MLOC
  - Windows XP: ~40 MLOC
  - Windows 10: ~50-60 MLOC

Why should we care?

- Размеры проектов неуклонно растут:
  - Ядро Linux v1.0.0 : 177 KLOC
  - Ядро Linux v5.1-rc2: ~21 MLOC
  - Windows XP: ~40 MLOC
  - Windows 10: ~50-60 MLOC
- А компилировать это все надо ☺


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

General principles


Phase I
Phase II
Source file
(*.c, *.cc,
*.cpp, *.cxx)
Preprocessor
Preprocessed
file (*.i)
Compiler
Object file (*.o)
Phase III
Object file 1
Object file N
Linker
Executable
program
Static libraries (*.lib, *.a)

```cpp
// TU1.cpp

#include <cstdint>

int64_t abs(int64_t num)
{
  return num >= 0 ? num : -num;
}
```

```cpp
// TU2.cpp

#include <cstdint>

extern int64_t abs(int64_t num);

int main()
{
  return abs(0);
}
```

```cpp
// TU1.cpp

#include <cstdint>

int32_t abs(int32_t num)
{
  return num >= 0 ? num : -num;
}
```

```cpp
// TU2.cpp

#include <cstdint>

extern int64_t abs(int64_t num);

int main()
{
  return abs(0);
}
```

```
// utilities.h

#include <cstdint>

int64_t abs(int64_t num);
```

```
// TU1.cpp

#include "utilities.h"

int64_t abs(int64_t num)
{
  return num >= 0 ? num : -num;
}
```

```
// TU2.cpp

#include "utilities.h"

int main()
{
  return abs(0);
}
```

```
// utilities.h

#pragma once

#include <cstdint>

int64_t abs(int64_t num);
extern int counter;
```

```
// TU1.cpp

#include "utilities.h"

int counter = 0;

int64_t abs(int64_t num)
{
  ++counter
  return num >= 0 ? num : -num;
}
```

```
// TU2.cpp

#include "utilities.h"

int main()
{
  return abs(0);
}
```

Dependencies

```
// header or source file

#include <vector>
#include <list>
#include <deque>

template <class T>
void foo(const std::vector<T> &v)
{
  ....
}

template <class T>
void foo(const std::list<T> &l)
{

}

template <class T>
void foo(const std::deque<T> &d)
{
  ....
}
```

```
// header or source file

#include <vector>
#include <list>
#include <deque>

template <class T>
void foo(const std::vector<T> &v)
{
  ....
}

template <class T>
void foo(const std::list<T> &l)
{

}

template <class T>
void foo(const std::deque<T> &d)
{
  ....
}
```

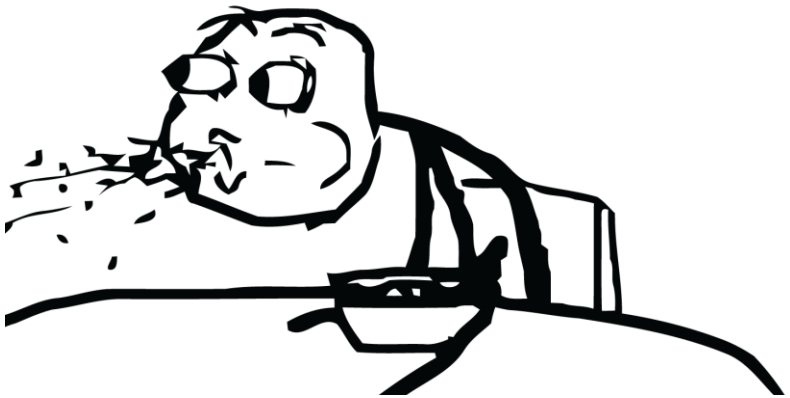
```
// header or source file

#include <vector>
#include <list>
#include <deque>

template <typename FwdIt>
void foo(FwdIt first, FwdIt last)
{
  ....
}

template <typename FwdIt, typename UnaryOp>
void foo(FwdIt first, FwdIt last, UnaryOp op)
{
  ....
}
```

```
// header or source file

#include <vector>
#include <list>
#include <deque>

template <typename FwdIt>
void foo(FwdIt first, FwdIt last)
{
  ....
}

template <typename FwdIt, typename UnaryOp>
void foo(FwdIt first, FwdIt last, UnaryOp op)
{
  ....
}
```

```
cloc.exe "%PROJECT_PATH%\TestExamples\Cpp\Cpp.i"
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 1.76  T=0.50 s (2.0 files/s, 81080.0 lines/s)

-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              1          19855              0          20685
-------------------------------------------------------------------------------

Type: Preprocessed C/C++ Source
Size: 1.19 MB
```

```
cloc.exe "%PROJECT_PATH%\TestExamples\Cpp\Cpp.i"
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 1.76  T=0.50 s (2.0 files/s, 81080.0 lines/s)

-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              1          19855              0          20685
-------------------------------------------------------------------------------
```

Type: Preprocessed C/C++ Source
Size: **1.19 MB**

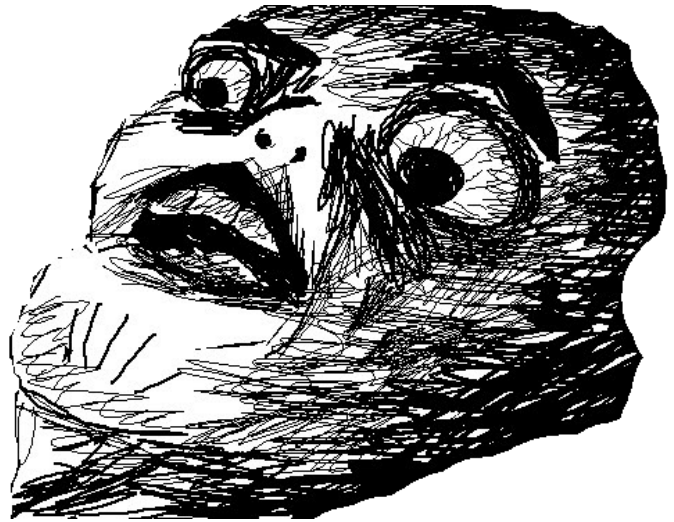


```
// header file

#include <iostream>

class SomeClass
{
  ....
};

std::ostream& operator<<(std::ostream &out, const SomeClass &obj);
```

```
cloc.exe "%PROJECT_PATH%\TestExamples\Cpp\Cpp.PVS-Studio.i"
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 1.76  T=0.50 s (2.0 files/s, 106508.0 lines/s)

-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              1          24016              0          29238
-------------------------------------------------------------------------------

Type: Preprocessed C/C++ Source
Size: 1.57 MB
```

```
cloc.exe "%PROJECT_PATH%\TestExamples\Cpp\Cpp.i"
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 1.76  T=0.50 s (2.0 files/s, 106508.0 lines/s)

-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              1          24016              0          29238
-------------------------------------------------------------------------------
```

Type: Preprocessed C/C++ Source
Size: **1.57 MB**



```
// Foo.h

#pragma once

class Foo
{
  ....
};

// Bar.h

#pragma once

#include "Foo.h"

class Bar
{
  void foo(Foo obj);  // <= Pass by value
  ....
};
```

```
// Foo.h

#pragma once

class Foo
{
  ....
};

// Bar.h

#pragma once

class Foo;

class Bar
{
  void foo(const Foo &obj);  // <= Pass by reference to const Foo
  void foo(const Foo *obj);  // <= Pass by pointer to const Foo
  ....
};
```

```
// Foo.h

#pragma once

struct Foo
{
  ....
};
```

```
// Bar.h

#pragma once

struct Foo;

struct Bar
{
  void f(Foo obj);  // <= Pass by value
  ....
};
```

```
// Bar.cpp

#include "Foo.h" // <= definition of Foo
#include "Bar.h" // <= definition of Bar

void Bar::f(Foo obj) // <= definition of Foo exists at function definition
{
  ....
}
```

```cpp
// Foobar.cpp

#include "Foo.h" // <= definition of Foo
#include "Bar.h" // <= definition of Bar

void foo()
{
  Foo foo;
  Bar bar;
  bar.f(foo); // <= definition of Foo exists at function call
}
```

```
// Foofwd.h

#pragma once

class Foo;


// Foo.h

#include <iosfwd>

#pragma once

class Foo
{
  ....
};

std::ostream& operator<<(std::ostream &, const Foo &);
```

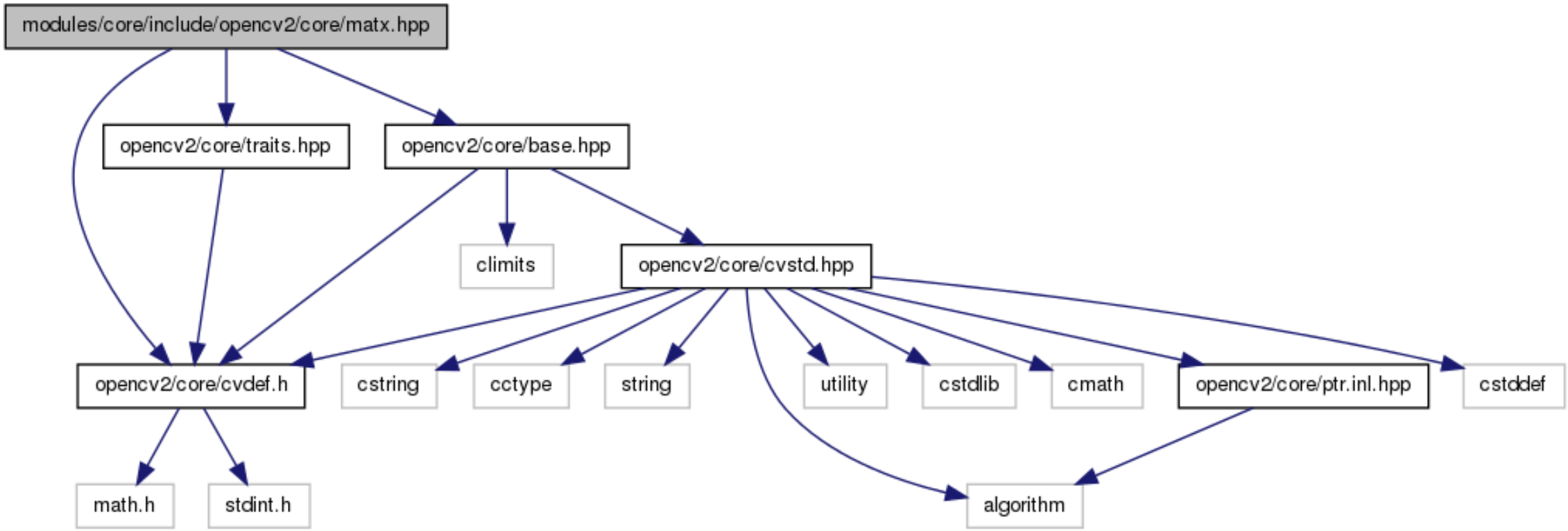
```cpp
// Foo.cpp

#include <iostream>
#include "Foo.h"

std::ostream& operator<<(std::ostream &out, const Foo &obj)
{
  out << ... << '\n';
  return out;
}
```

```
// Widget.h

#include <memory>

class Widget
{
  struct Impl;
  std::unique_ptr<Impl> m_impl;

public:
  void foo() const noexcept;
  ....
};

// Widget.cpp

struct Widget::Impl
{
  ....
};

Widget::Widget() : m_impl(std::make_unique<Impl>()) { }
void Widget::foo() const noexcept { m_impl->.... }
```

Tools

- IWYU (include-what-you-use)
- CppClean
- ReSharper C++
- Clion

Doxygen + GraphViz



Extern templates

```
// SomeHeader.h

template <typename T>
void foo(T arg) { .... }

template <typename T>
class SomeClass { .... };
```

```
// A.cpp

#include "SomeHeader.h"

void bar1()
{
  foo(0);
  SomeClass<int> obj1;
}
```

```
// B.cpp

#include “SomeHeader.h”

void bar2()
{
  foo(255);
  SomeClass<int> obj2;
}
```

```
A.o
  void bar1()
  void foo<int>(int)
  SomeClass<int>

B.o
  void bar2()
  void foo<int>(int)
  SomeClass<int>
```

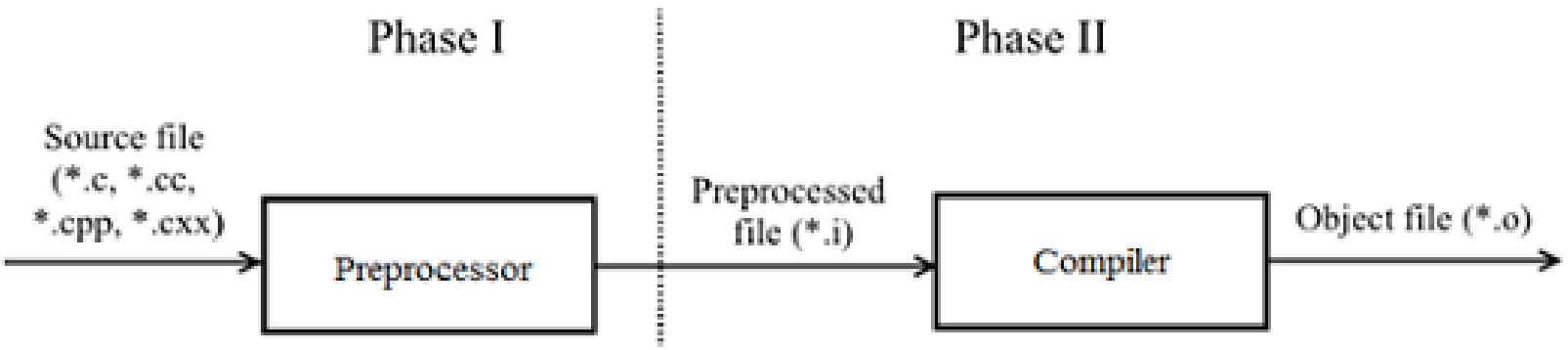
```
// SomeHeader.h

template <typename T>
void foo(T arg) { .... }

template <typename T>
class SomeClass { .... };
```

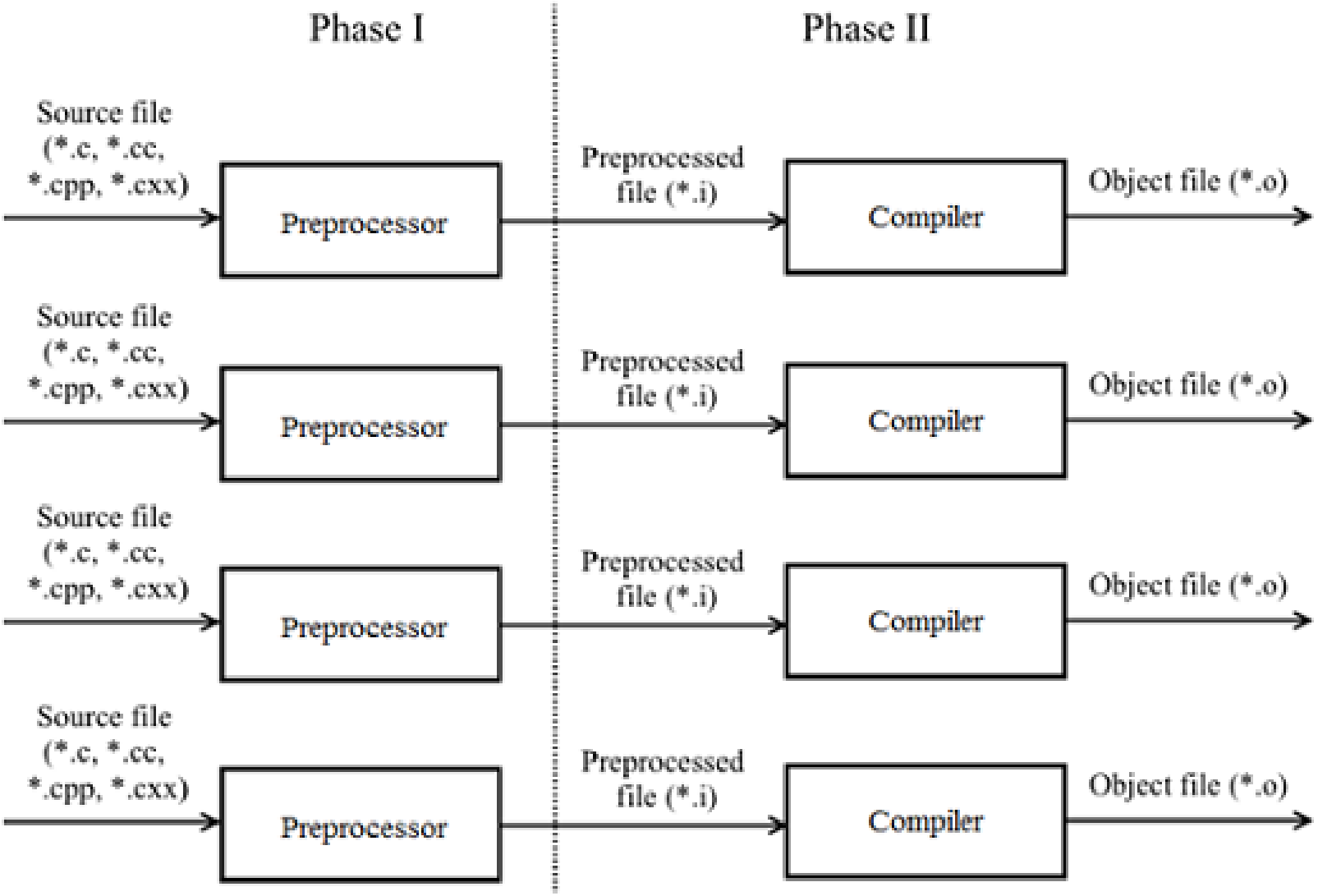
```
// A.cpp

#include “SomeHeader.h”

void bar1()
{
  foo(0);
  SomeClass<int> obj1;
}
```

```
// B.cpp

#include “SomeHeader.h”

extern template void foo<int>(int);
extern template class SomeClass<int>;

void bar2()
{
  foo(255);
  SomeClass<int> obj2;
}
```

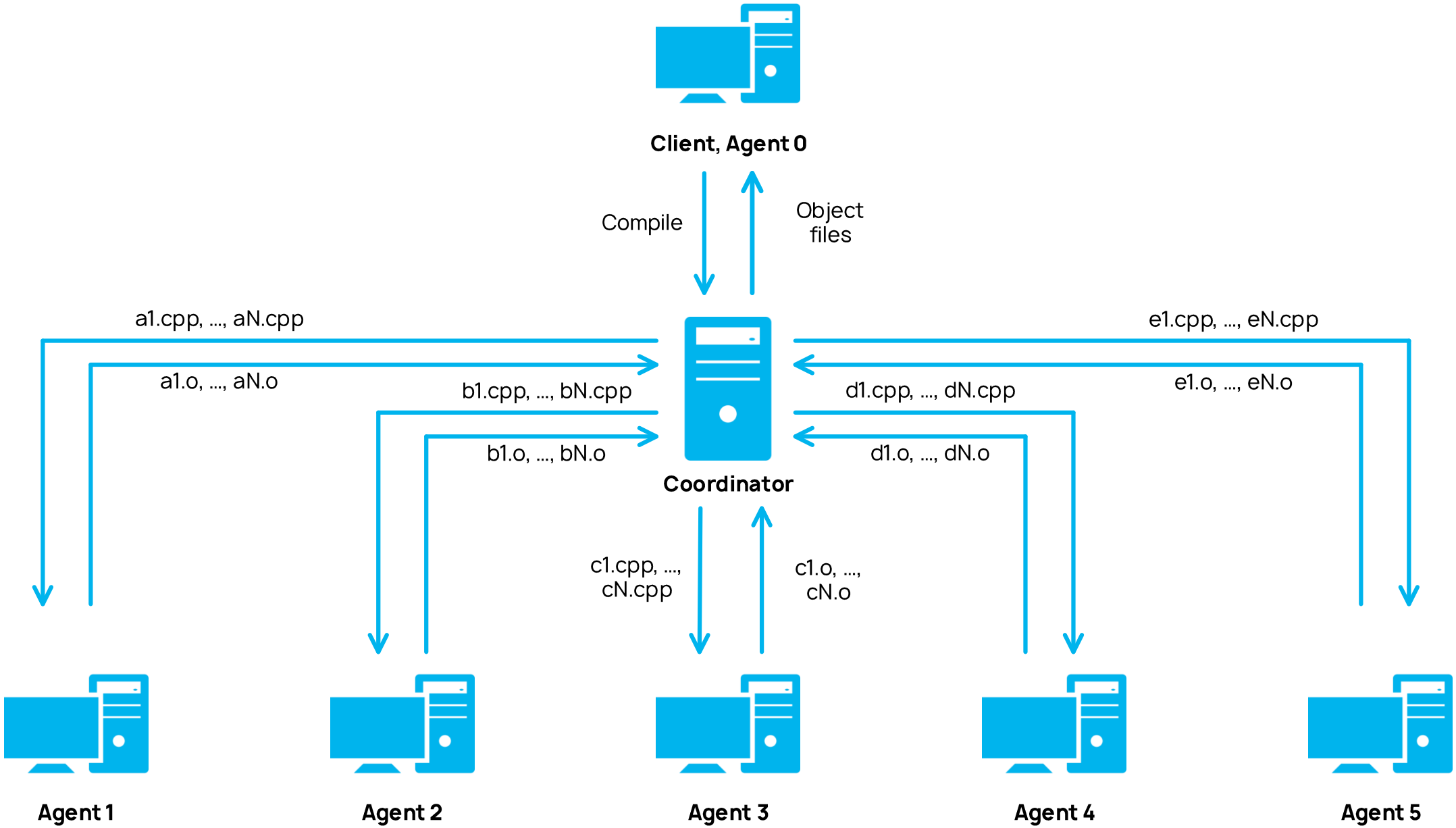
```
A.o
  void bar1()
  void foo<int>(int)
  SomeClass<int>

B.o
  void bar2()
  void foo<int>(int)
  SomeClass<int>
```

Parallel compilation


Phase I
Phase II
Source file
(*.c, *.cc,
*.cpp, *.cxx)
Preprocessor
Preprocessed
file (*.i)
Compiler
Object file (*.o)


Phase I
Phase II
Source file (*.c, *.cc, *.cpp, *.cxx)
Preprocessor
Preprocessed file (*.i)
Compiler
Object file (*.o)
Source file (*.c, *.cc, *.cpp, *.cxx)
Preprocessor
Preprocessed file (*.i)
Compiler
Object file (*.o)
Source file (*.c, *.cc, *.cpp, *.cxx)
Preprocessor
Preprocessed file (*.i)
Compiler
Object file (*.o)
Source file (*.c, *.cc, *.cpp, *.cxx)
Preprocessor
Preprocessed file (*.i)
Compiler
Object file (*.o)

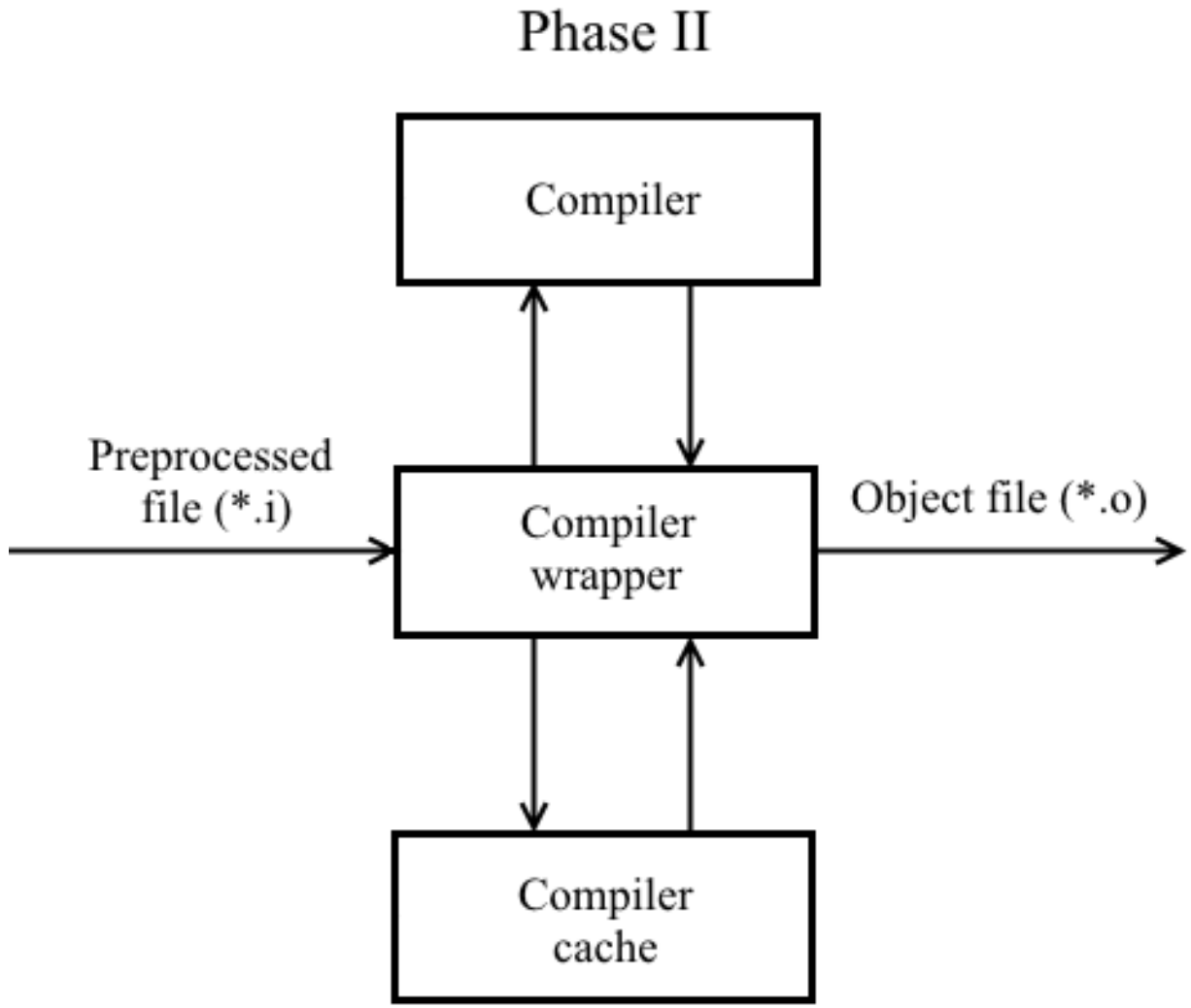
Distributed compilation


Client, Agent 0
Compile
Object files
a1.cpp, ..., aN.cpp
a1.o, ..., aN.o
b1.cpp, ..., bN.cpp
b1.o, ..., bN.o
Coordinator
c1.cpp, ..., cN.cpp
c1.o, ..., cN.o
d1.cpp, ..., dN.cpp
d1.o, ..., dN.o
e1.cpp, ..., eN.cpp
e1.o, ..., eN.o
Agent 1
Agent 2
Agent 3
Agent 4
Agent 5

Tools

- IncrediBuild
- distcc
- icecc (Icecream)

Compiler cache


Phase II
Compiler
Preprocessed
file (*.i)
Compiler
wrapper
Object file (*.o)
Compiler
cache

Tools

- ccache (Linux)
- cachecc1 (Linux)
- clcache (Windows, MSVC)
- cclash (Windows, MSVC)

Precompiled header files

```cpp
// TU1.cpp

#include <functional>
#include <vector>
#include <iostream>
#include <algorithm>

void foo()
{
  ....
}
```

```cpp
// TU2.cpp

#include <list>
#include <iostream>
#include <algorithm>
#include <functional>

void bar()
{
  ....
}
```

```cpp
// TU1.cpp

#include <functional>
#include <vector>
#include <iostream>
#include <algorithm>

void foo()
{
  ....
}
```

```cpp
// TU2.cpp

#include <list>
#include <iostream>
#include <algorithm>
#include <functional>

void bar()
{
  ....
}
```

```
// stdafx.h

#include <funcitonal>
#include <vector>
#include <iostream>
#include <list>
#include <algorithm>
```

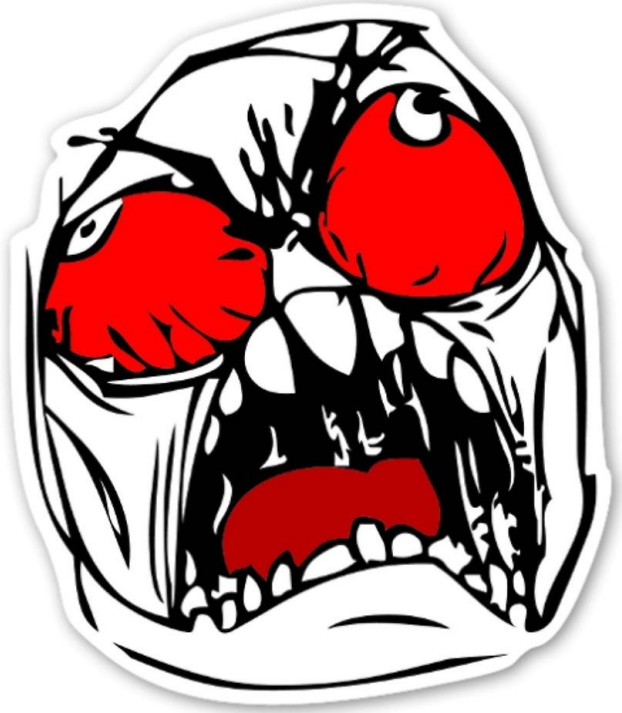
```
// TU1.cpp

#include "stdafx.h"

void foo()
{
  ....
}
```

```
// TU2.cpp

#include "stdafx.h"

void bar()
{
  ....
}
```

Single compilation unit

```
// TU1.cpp
....

// TU2.cpp
....

// TU_N.cpp
....

// SCU.cpp

#include "TU1.cpp"
#include "TU2.cpp“
...
#include "TU_N.cpp"
```

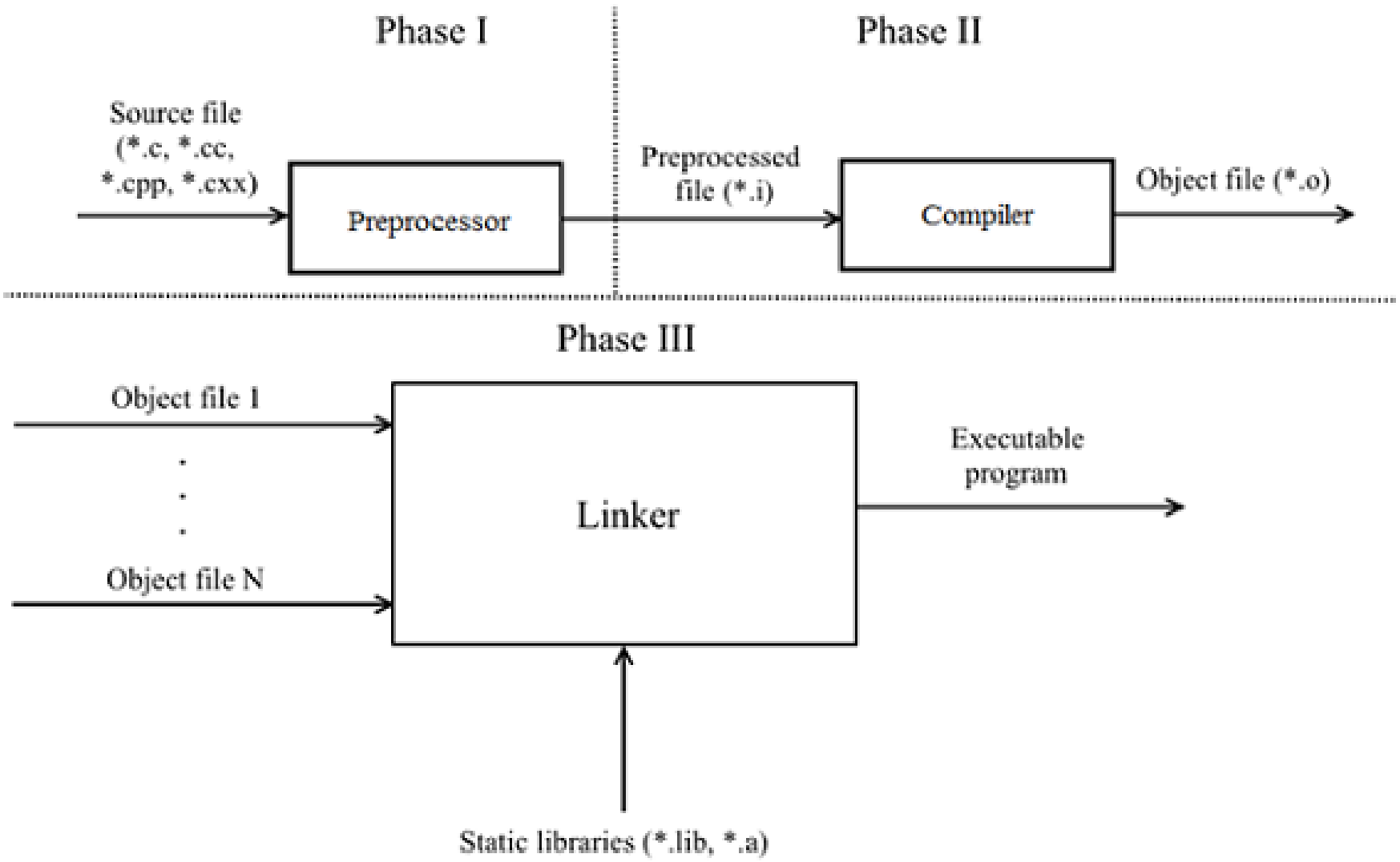
```
// TU1.cpp
....

// TU2.cpp
....

// TU_N.cpp
....

// SCU.cpp

#include "TU1.cpp"
#include "TU2.cpp“
...
#include "TU_N.cpp"
```



Pros and cons

- Меньше компилируемых файлов
- Link time optimization на этапе компиляции
- Страдает инкрементальная сборка
- Делать один SCU невыгодно, лучше разбить на N частей
- Возможное нарушение ODR
- Коллизии имен из-за '`using namespace`'

Replacing translation components


Phase I
Phase II
Source file
(*.c, *.cc,
*.cpp, *.cxx)
Preprocessor
Preprocessed
file (*.i)
Compiler
Object file (*.o)
Phase III
Object file 1
Object file N
Linker
Executable
program
Static libraries (*.lib, *.a)

Tools

- Preprocessor: Warp (Facebook), Clang, MSVC

Tools

- Preprocessor: Warp (Facebook), Clang, MSVC
- Compiler: Zapcc

Tools

- Preprocessor: Warp (Facebook), Clang, MSVC
- Compiler: Zapcc
- Linker: GNU gold (for ELF)

Modules

```
// A.cpp

#include <vector>
#include <list>
....

// all macros and symbols are available
```

```
// B.cpp

#define _ITERATOR_DEBUG_LEVEL 0

#include <vector>
#include <list>
....

// all macros and symbols are available

// "A.cpp" and "B.cpp" are different

// includes are sensitive to previous macro definitions
```

```
// B.cpp

#define _ITERATOR_DEBUG_LEVEL 0

// modules for legacy code, export macros and symbols
import <vector>;
import <list>;

// new cool modules, export symbols that you want, don't export macros
import std.vector;
import std.list;
....

// "A.cpp" and "B.cpp" are the same

// imports aren't sensitive to previous macro definitions!!!
```

```
// Interface part

// Starts with preamble

export module M;

export import N; // imports
                 // module N
                 // and exports

void foo(); // doesn't export

export void bar(); // exports
```

```
// Implementation part

// Starts with preamble

module M;

void foo() { .... }

void bar() { foo(); .... }
```

Implementations

- MSVC
- GCC
- Clang

Speed!!!

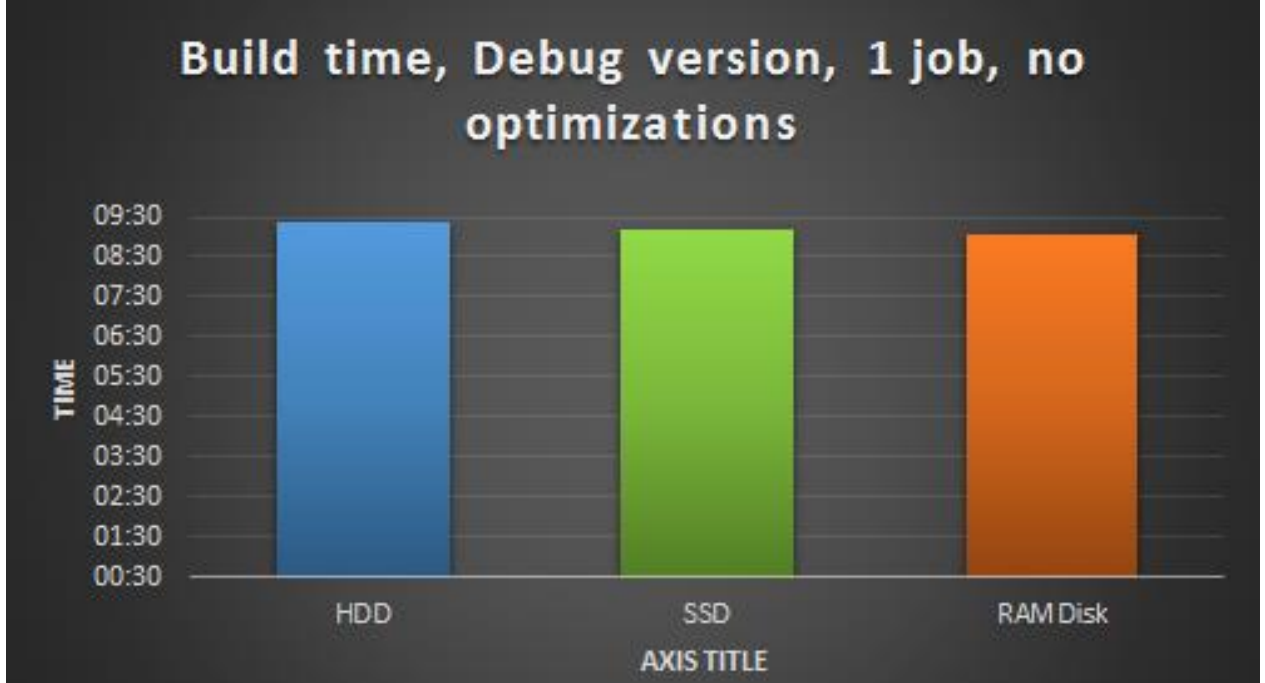
Photo credit: bhmpics

It's time for benchmarks!

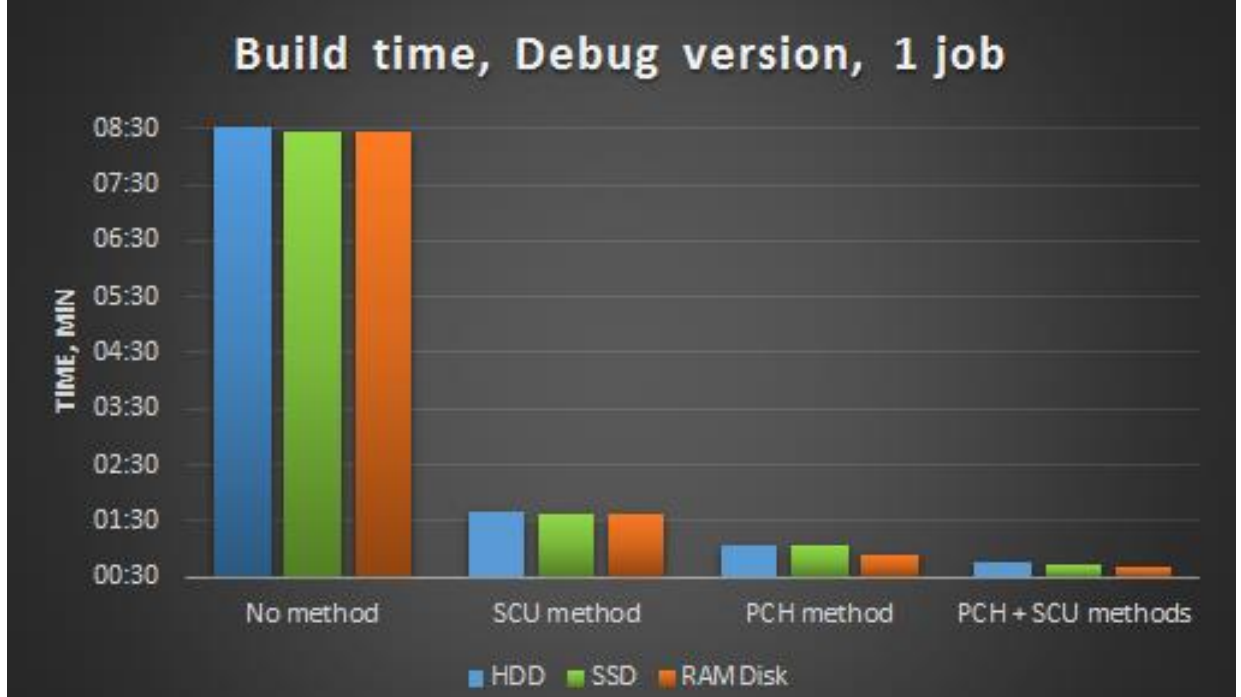
Test machine

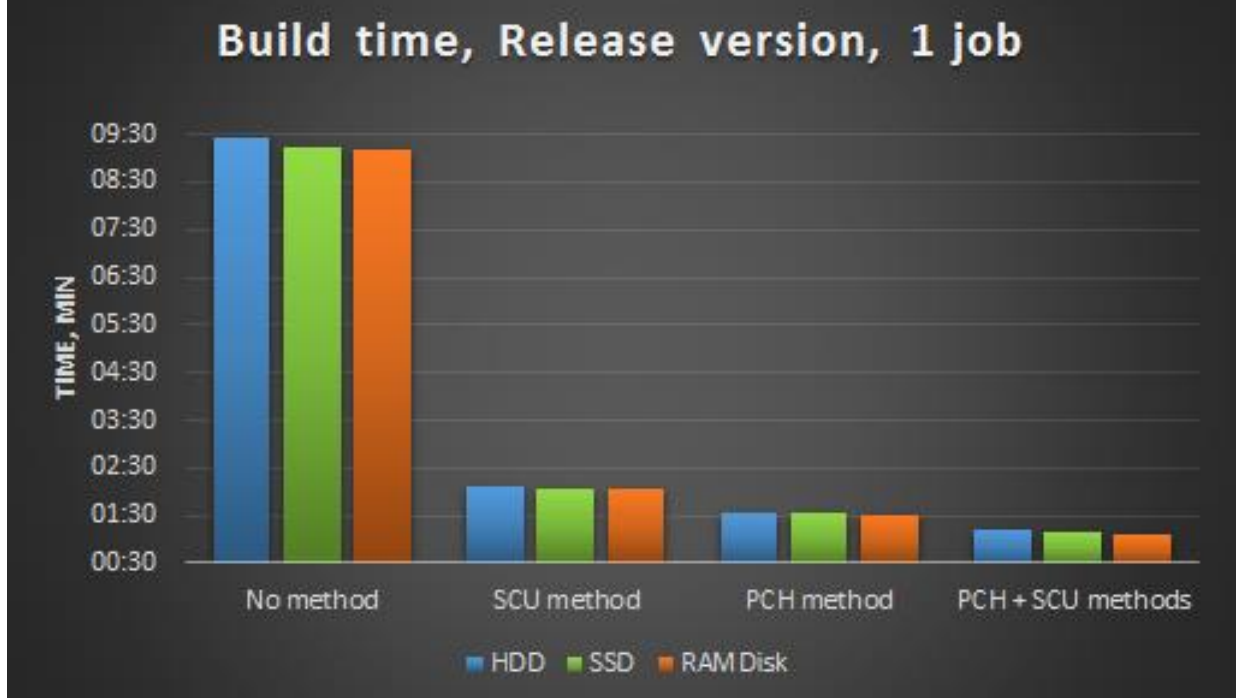
- Intel Core i7-4770 3.4 GHz (8 CPU)
- 16 Gb RAM DDR3-1333 MHz
- SSD Samsung SSD 840 EVO 250 Gb
- HDD WDC WD20EZRX-00D8PB0 2 Tb

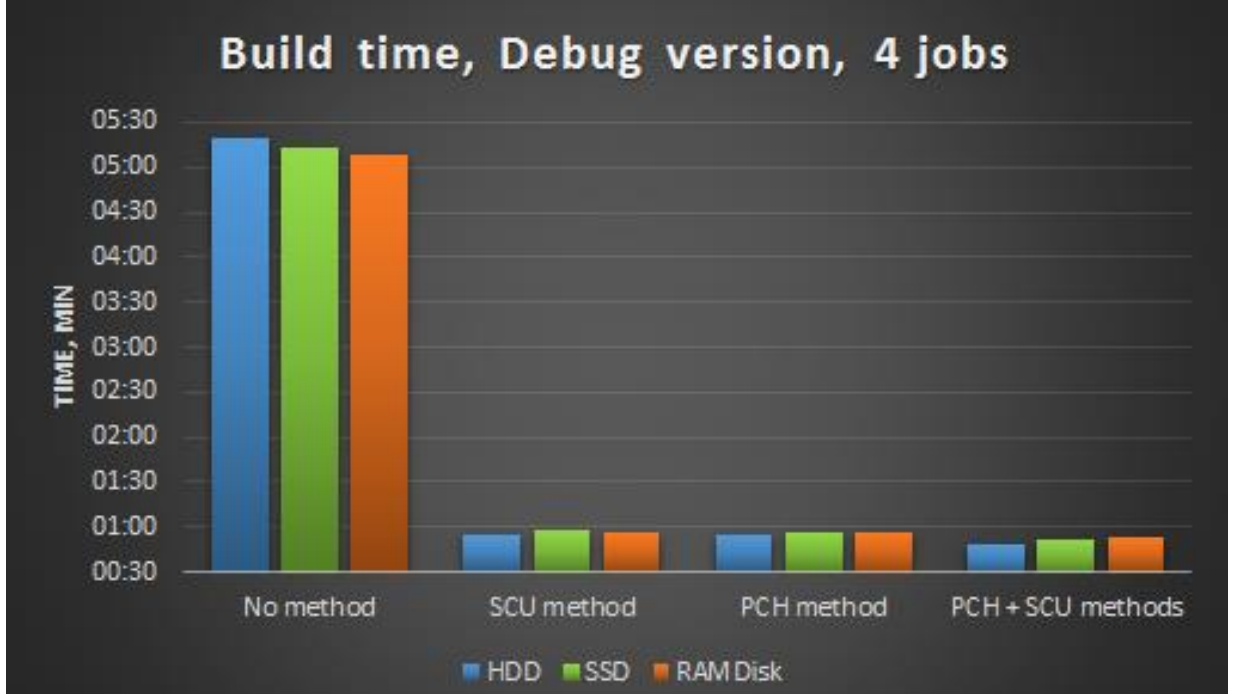
Language	files	blank	comment	code
C++	380	28556	17574	150222
C/C++ Header	221	9049	9847	46360
Assembly	1	13	22	298
SUM:	602	37618	27443	196880

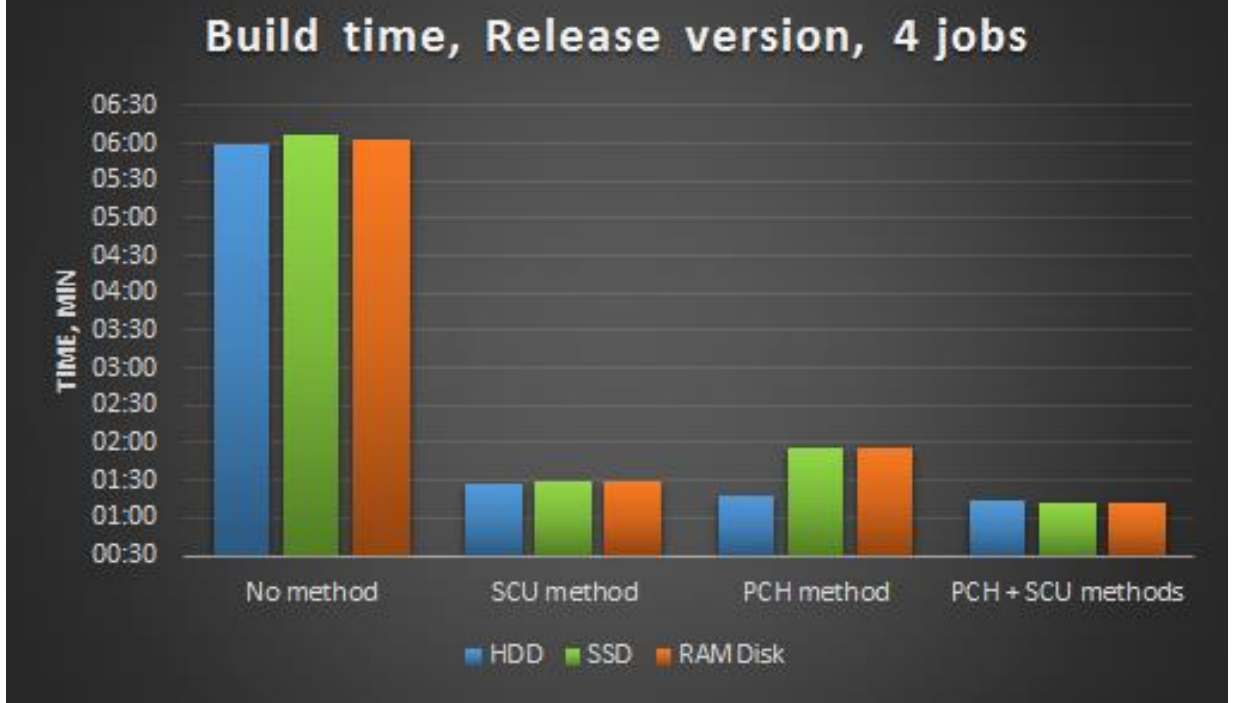

Build time, Debug version, 1 job, no optimizations
09:30
08:30
07:30
06:30
05:30
04:30
03:30
02:30
01:30
00:30
TIME
HDD
SSD
RAM Disk
AXIS TITLE

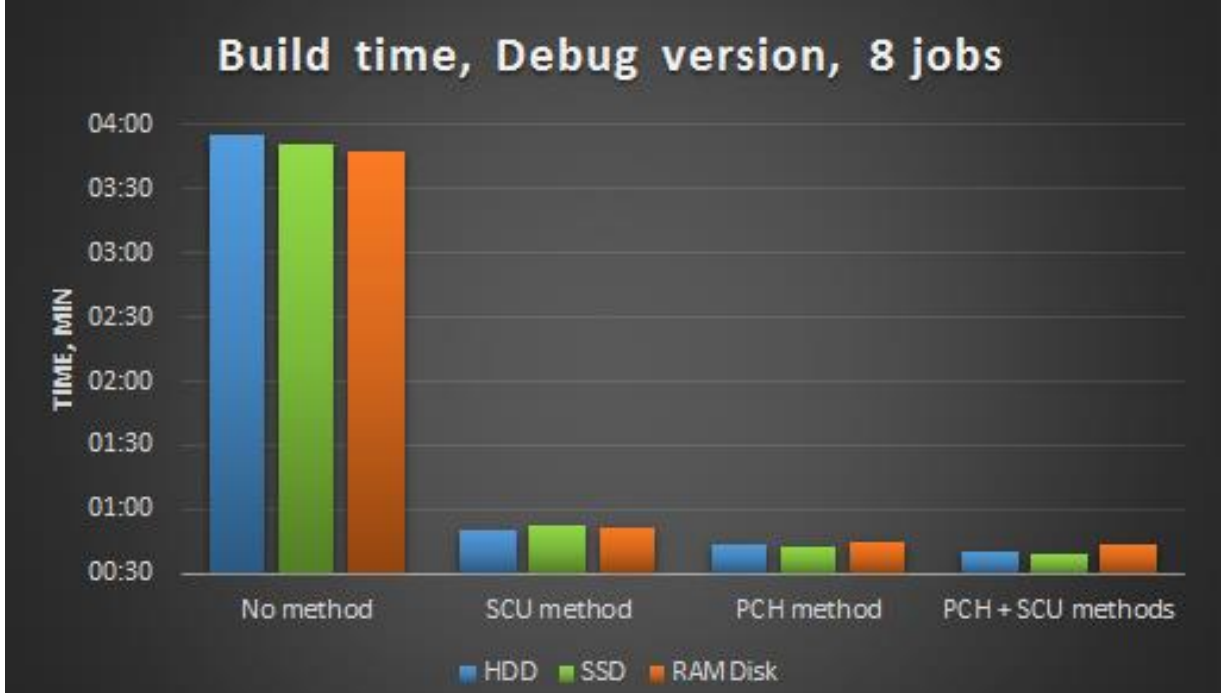

Build time, Release version, 1 job, no optimizations
10:30
09:30
08:30
07:30
06:30
05:30
04:30
03:30
02:30
01:30
00:30
TIME
HDD
SSD
RAM Disk

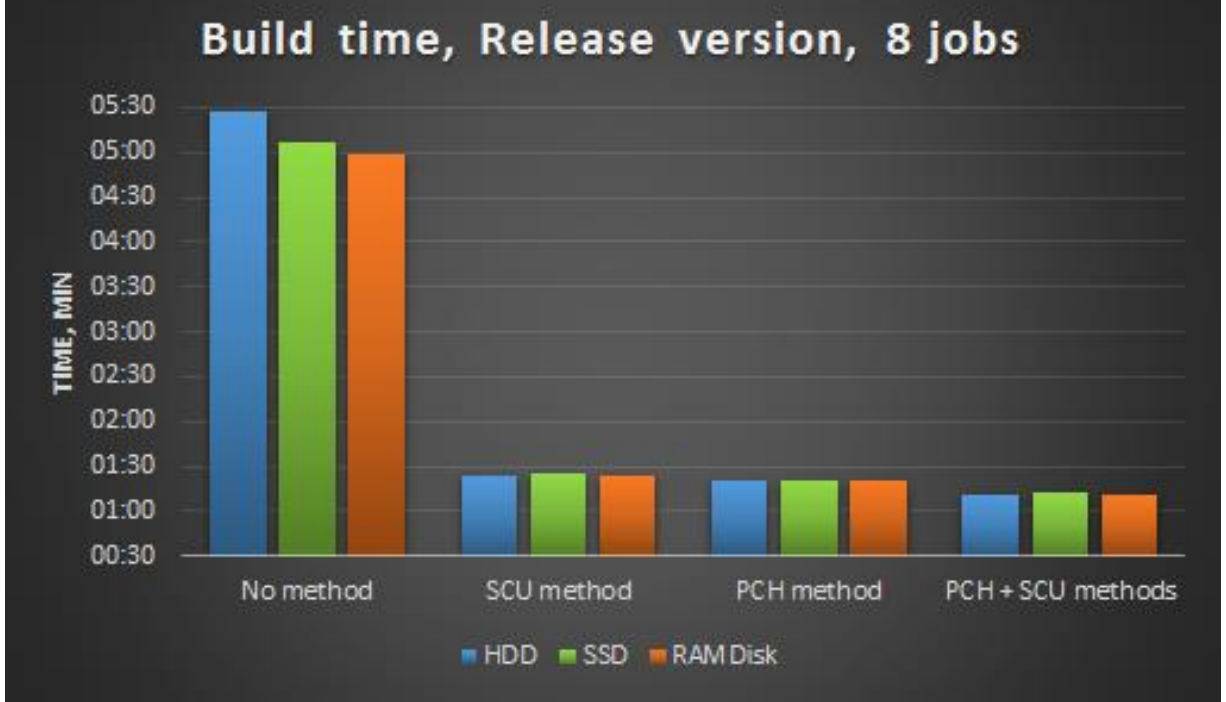

Build time, Debug version, 1 job
TIME, MIN
08:30
07:30
06:30
05:30
04:30
03:30
02:30
01:30
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk


Build time, Release version, 1 job
TIME, MIN
09:30
08:30
07:30
06:30
05:30
04:30
03:30
02:30
01:30
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk


Build time, Debug version, 4 jobs
TIME, MIN
05:30
05:00
04:30
04:00
03:30
03:00
02:30
02:00
01:30
01:00
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk


Build time, Release version, 4 jobs
TIME, MIN
06:30
06:00
05:30
05:00
04:30
04:00
03:30
03:00
02:30
02:00
01:30
01:00
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk


Build time, Debug version, 8 jobs
TIME, MIN
04:00
03:30
03:00
02:30
02:00
01:30
01:00
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk


Build time, Release version, 8 jobs
TIME, MIN
05:30
05:00
04:30
04:00
03:30
03:00
02:30
02:00
01:30
01:00
00:30
No method
SCU method
PCH method
PCH + SCU methods
HDD
SSD
RAM Disk

END

Q&A