ZAPCC, CACHING COMPILATION SERVER

Yaron Keren

Ceemple Software Ltd.

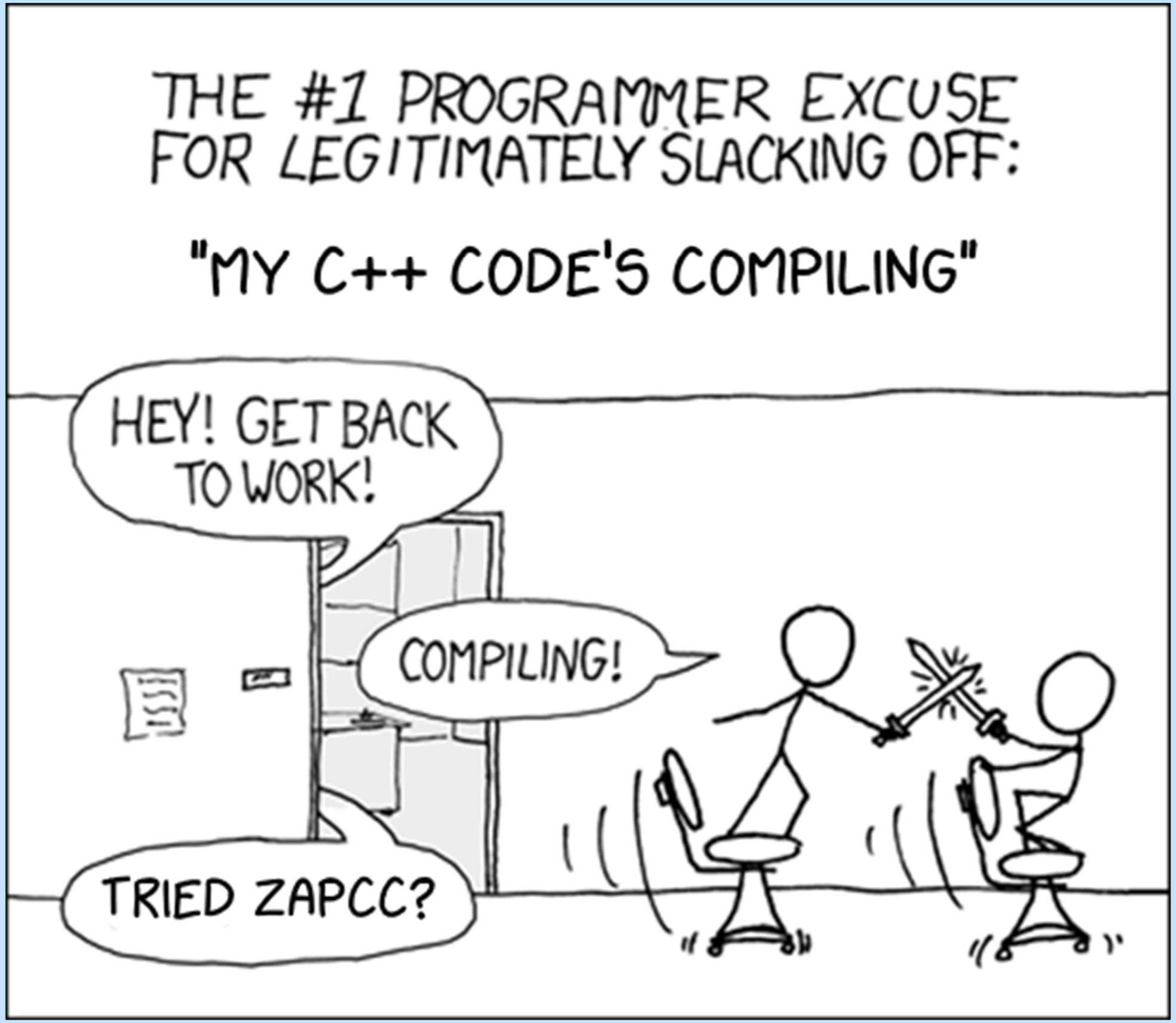
www.zapcc.com

Opt. level	Standard [s]	Precomp [s]
-O0	28	25
-O1	55	52
-O2	85	82

File name	File size	Speedup
database.cc	193864	X1.32
mutation_partition.cc	78913	X2.4
main.cc	31967	X2.5
query.cc	9908	X9.7
clocks-impl.cc	780	X43

AGENDA

- Introduction
- C++ Compilation is long
- Precompiled headers
- C++ Modules
- Test case
- Multicore & other options
- Zapcc approach
- Zapcc testing
- Zapcc results
- Q&A


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY C++ CODE'S COMPILING"
HEY! GET BACK TO WORK!
COMPILING!
TRIED ZAPCC?

Credit: https://xkcd.com/303

C++ COMPILATION IS LONG

- Classic answer
- Walter Bright, Dr. Dobb's (2010) https://goo.gl/mfN2VJ
- Textual includes
- Monolithic compilation
- Just include everything
- Repeated templates instantiations
- Walter created the D language

Precompiled Headers

```
g++ test.h
g++ test.cpp
```

```
cl /Yc test.h
cl /Yu test.h test.cpp
```

```
clang++ test.h -o test.h.pch
clang++ –include test.h.pch test.cpp
```

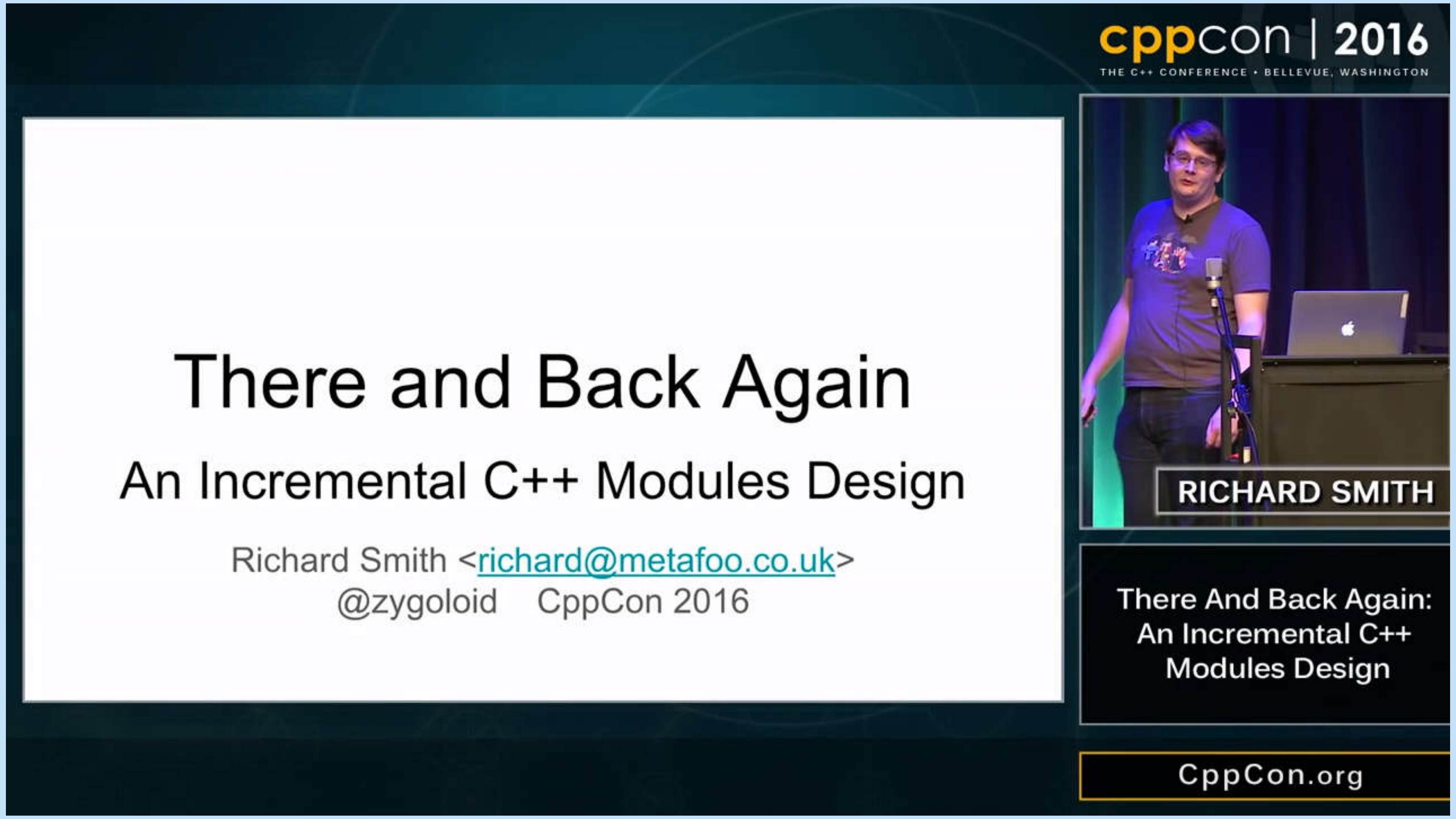
- Precompile (pre-parse) common headers used in multiple files
- Not standard, limitations
- Refactor include structure
- Resulting in dependency hell: everything depend on precomp header dependent on all included headers
- Useful for rarely-changing source files, such as OS X system headers

https://gcc.gnu.org/onlinedocs/gcc/Precompiled-Headers.html

- Only one precompiled header…
- Cannot include a precompiled header from inside another header
- Macros defined before the precompiled header must be same

C++ MODULES (A NEW HOPE)

https://youtu.be/h1E-XyxqJRE



There and Back Again

An Incremental C++ Modules Design

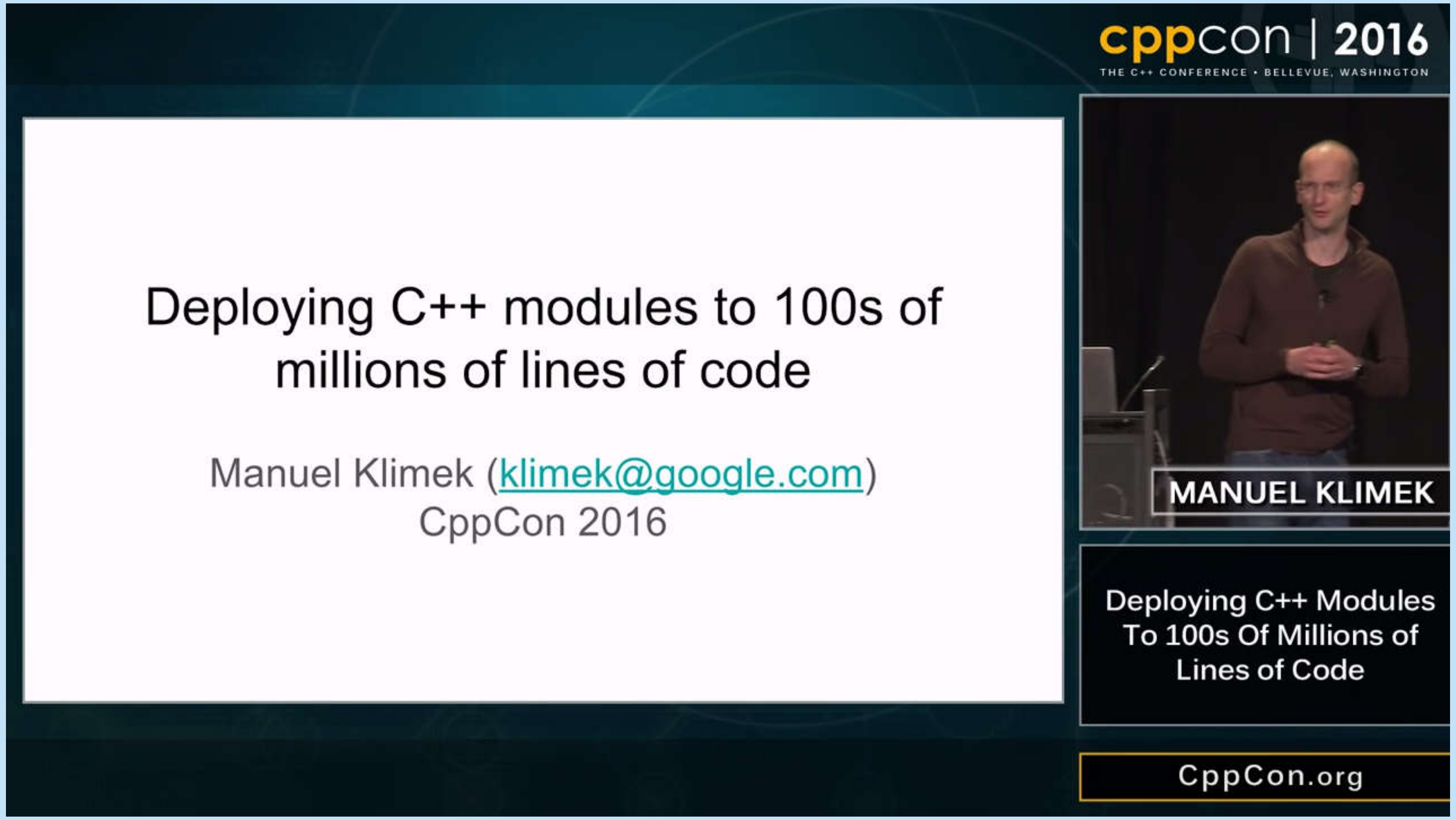
Richard Smith <richard@metafoo.co.uk>
@zygoloid CppCon 2016

- Clean design
- Independent parsing
- Compile-time scalability
- Already implemented in, clang, gcc, Visual C++
- All is well?

In depth: https://clang.llvm.org/docs/Modules.html

C++ MODULES (LEGACY STRIKES BACK)

https://youtu.be/dHFNpBfemDI



Deploying C++ modules to 100s of millions of lines of code

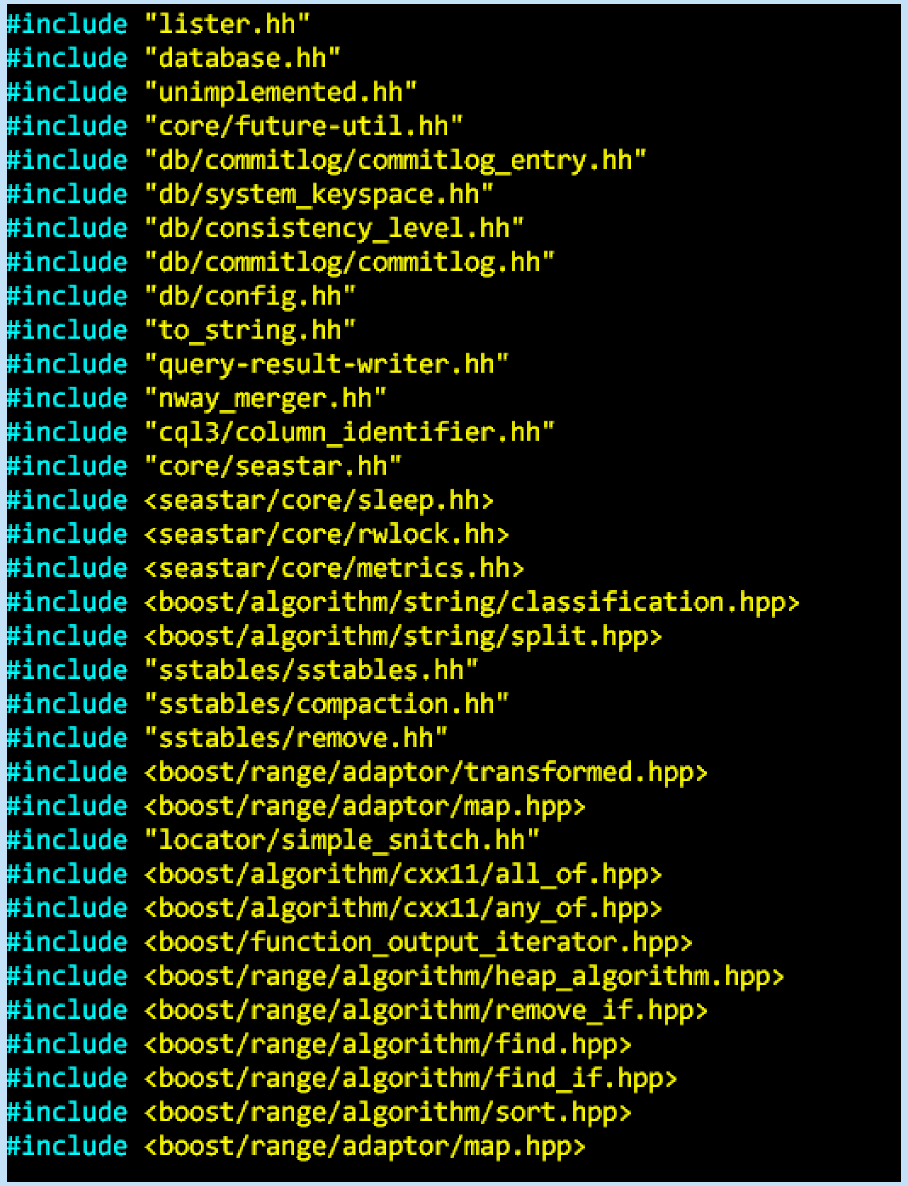
Manuel Klimek (klimek@google.com)
CppCon 2016

cppcon | 2016
THE C++ CONFERENCE • BELLEVUE, WASHINGTON
MANUEL KLIMEK
Deploying C++ Modules To 100s Of Millions of Lines of Code
CppCon.org

- Still experimental, maybe C++ 20
- Google, Apple, Microsoft already using modules
- They are C++ compiler **developers**
- Rewrite the world's code?
- Tools can help:
  Raphael Isemann, A CMake toolkit for migrating C++ projects to clang's module system, 2017 US LLVM Developers' Meeting

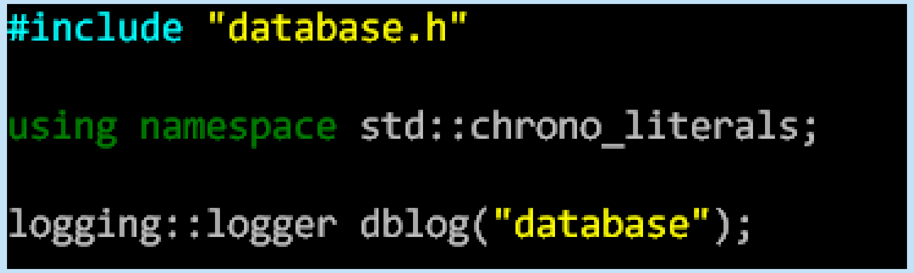
http://scylladb.com

database.cc
194K, about 4000 LOC
#includes the world



```
#include "lister.hh"
#include "database.hh"
#include "unimplemented.hh"
#include "core/future-util.hh"
#include "db/commitlog/commitlog_entry.hh"
#include "db/system_keyspace.hh"
#include "db/consistency_level.hh"
#include "db/commitlog/commitlog.hh"
#include "db/config.hh"
#include "to_string.hh"
#include "query-result-writer.hh"
#include "nway_merger.hh"
#include "cql3/column_identifier.hh"
#include "core/seastar.hh"
#include <seastar/core/sleep.hh>
#include <seastar/core/rwlock.hh>
#include <seastar/core/metrics.hh>
#include <boost/algorithm/string/classification.hpp>
#include <boost/algorithm/string/split.hpp>
#include "sstables/sstables.hh"
#include "sstables/compaction.hh"
#include "sstables/remove.hh"
#include <boost/range/adaptor/transformed.hpp>
#include <boost/range/adaptor/map.hpp>
#include "locator/simple_snitch.hh"
#include <boost/algorithm/cxx11/all_of.hpp>
#include <boost/algorithm/cxx11/any_of.hpp>
#include <boost/function_output_iterator.hpp>
#include <boost/range/algorithm/heap_algorithm.hpp>
#include <boost/range/algorithm/remove_if.hpp>
#include <boost/range/algorithm/find.hpp>
#include <boost/range/algorithm/find_if.hpp>
#include <boost/range/algorithm/sort.hpp>
#include <boost/range/adaptor/map.hpp>
```

database.cc

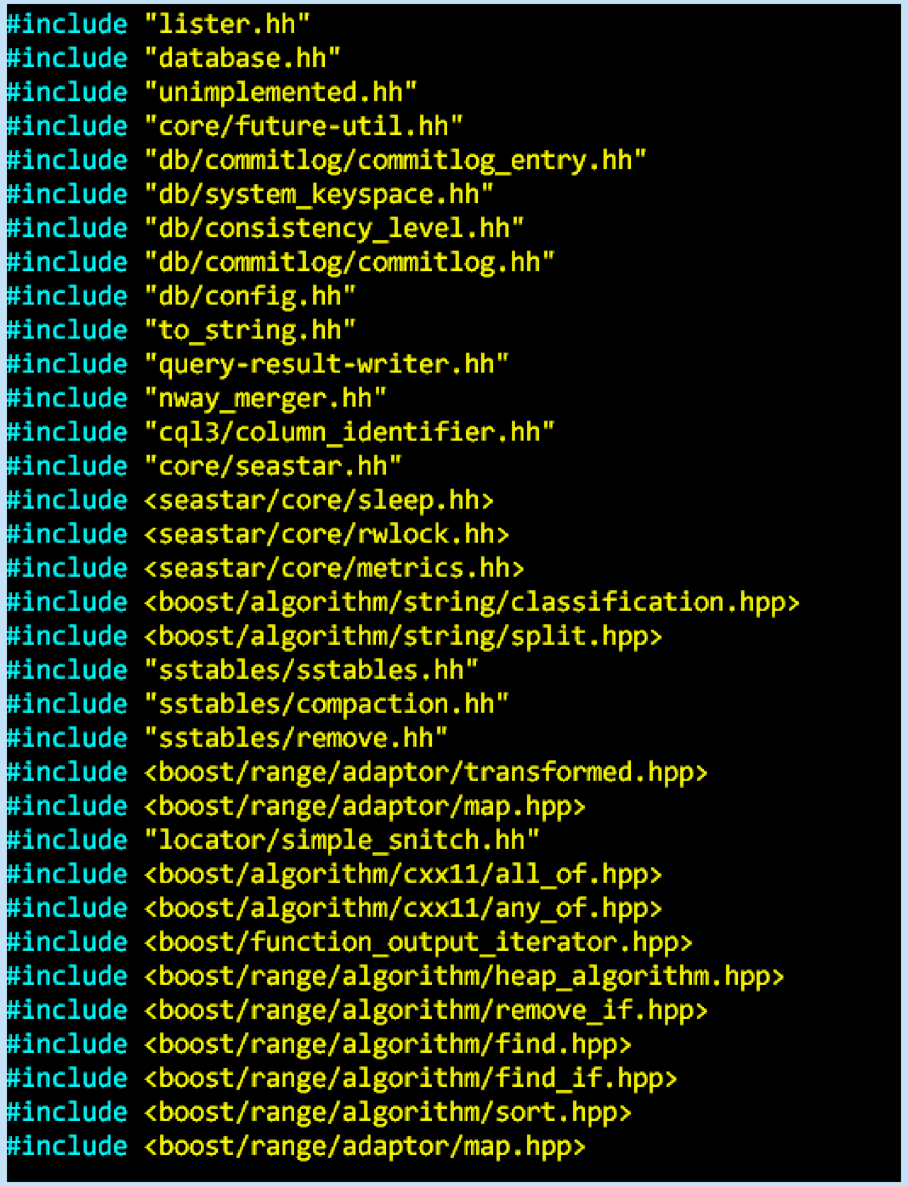


```
#include "database.h"

using namespace std::chrono_literals;

logging::logger dblog("database");
```

database.h



```
#include "lister.hh"
#include "database.hh"
#include "unimplemented.hh"
#include "core/future-util.hh"
#include "db/commitlog/commitlog_entry.hh"
#include "db/system_keyspace.hh"
#include "db/consistency_level.hh"
#include "db/commitlog/commitlog.hh"
#include "db/config.hh"
#include "to_string.hh"
#include "query-result-writer.hh"
#include "nway_merger.hh"
#include "cql3/column_identifier.hh"
#include "core/seastar.hh"
#include <seastar/core/sleep.hh>
#include <seastar/core/rwlock.hh>
#include <seastar/core/metrics.hh>
#include <boost/algorithm/string/classification.hpp>
#include <boost/algorithm/string/split.hpp>
#include "sstables/sstables.hh"
#include "sstables/compaction.hh"
#include "sstables/remove.hh"
#include <boost/range/adaptor/transformed.hpp>
#include <boost/range/adaptor/map.hpp>
#include "locator/simple_snitch.hh"
#include <boost/algorithm/cxx11/all_of.hpp>
#include <boost/algorithm/cxx11/any_of.hpp>
#include <boost/function_output_iterator.hpp>
#include <boost/range/algorithm/heap_algorithm.hpp>
#include <boost/range/algorithm/remove_if.hpp>
#include <boost/range/algorithm/find.hpp>
#include <boost/range/algorithm/find_if.hpp>
#include <boost/range/algorithm/sort.hpp>
#include <boost/range/adaptor/map.hpp>
```

- Having modularized the world’s code, where will we be?
- Modules generalize precompiled headers, offers similar performance without the restrictions

  clang modules & precompiled headers in depth:
  http://clang.llvm.org/docs/PCHInternals.html
- Split one source file into source and precompiled header as proxy performance for modules
- Best-case performance as real world code would not pre-include everything

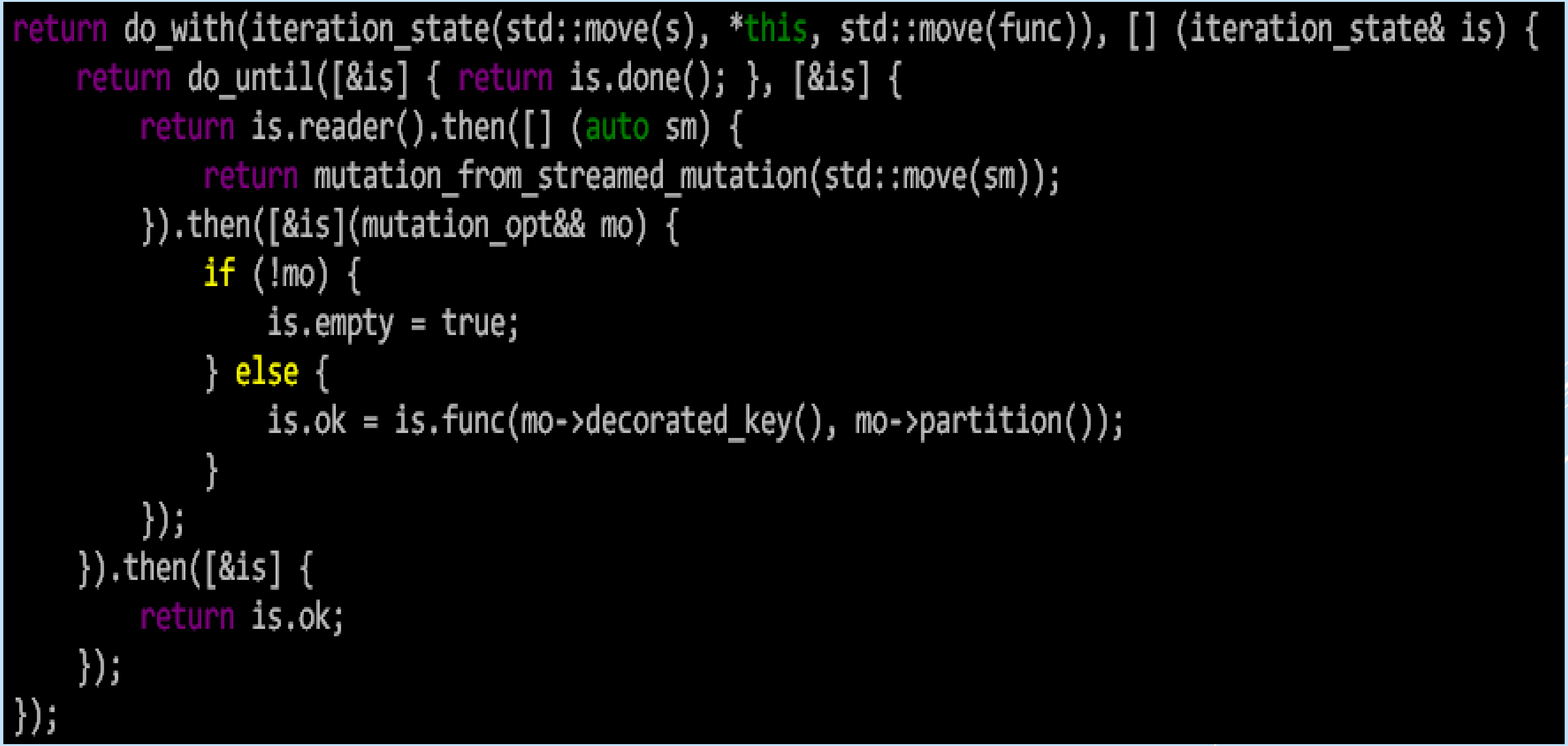
Opt. level	Standard [s]	Precomp [s]
-O0	28	25
-O1	55	52
-O2	85	82

- Precompiled headers compile time was 6s
- Clang 4.0.0 trunk (June 2017)

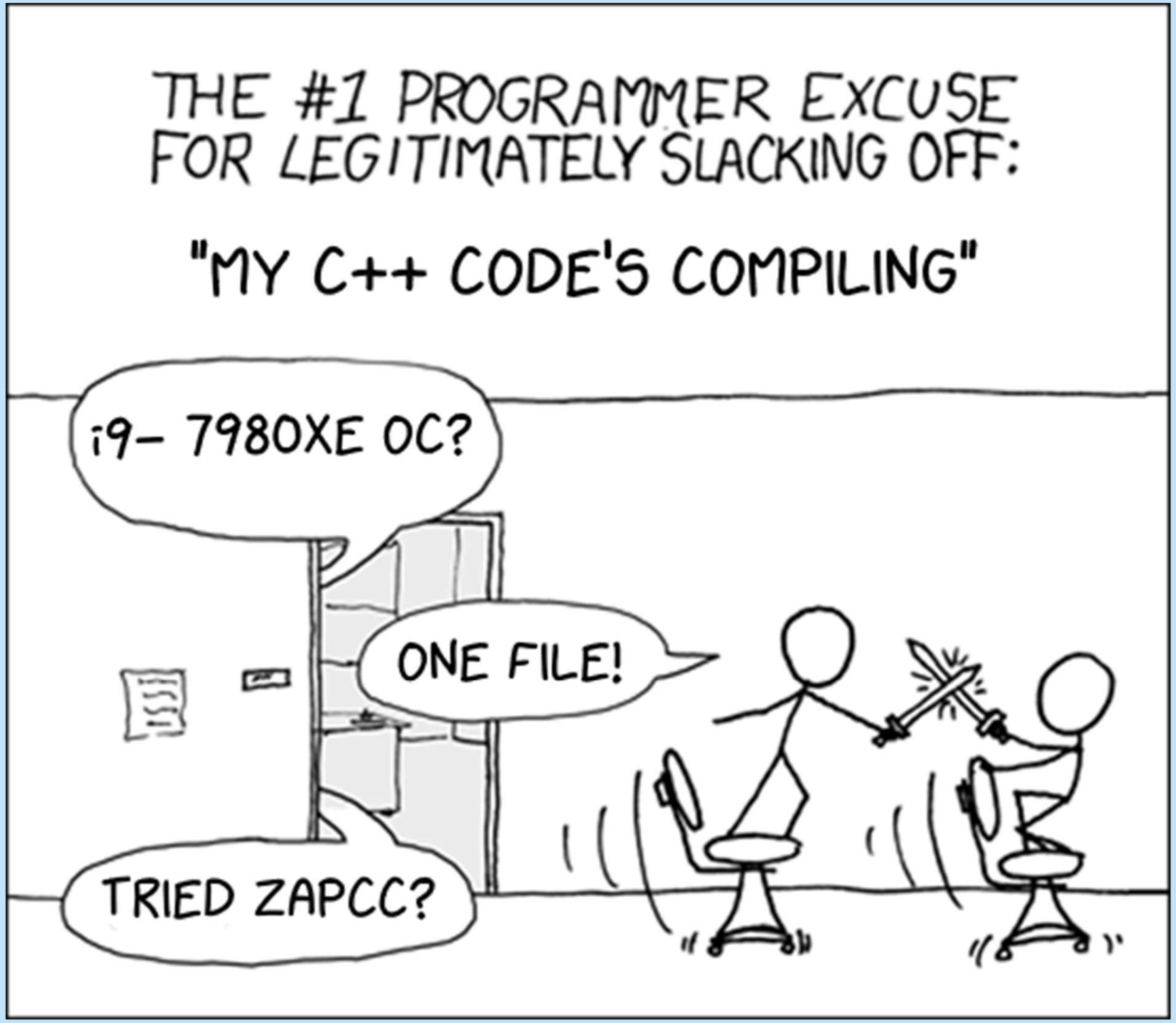
WHAT HAPPENED?

Dorothy: Toto, I've a feeling we're not in Kansas anymore (The Wizard of Oz)

- For modern C++ code, parsing not always the main reason for long compilation
- Template instantiation
- Lambda functions
- Optimization, Debug mode as well:
- ScyllaDB devs use –O2 since –O0 executables are painfully slow
- Some LLVM/clang devs do the same
- Having waited that long, modules may fall short of expectations



```
return do_with(iteration_state(std::move(s), *this, std::move(func)), [] (iteration_state& is) {
    return do_until([&is] { return is.done(); }, [&is] {
        return is.reader().then([] (auto sm) {
            return mutation_from_streamed_mutation(std::move(sm));
        }).then([&is](mutation_opt&& mo) {
            if (!mo) {
                is.empty = true;
            } else {
                is.ok = is.func(mo->decorated_key(), mo->partition());
            }
        });
    }).then([&is] {
        return is.ok;
    });
});
```


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY C++ CODE'S COMPILING"
i9- 7980XE OC?
ONE FILE!
TRIED ZAPCC?

Credit: https://xkcd.com/303

- **Most popular: manycore `–jMANY`**
- Just use the Intel Core i9-7980XE EXTREME EDITION or the AMD Ryzen Threadripper
- Works for “build all”/CI scenario
- Can’t parallelize single file compilation, developer mode
- Explicit instantiation
- Unity compilation
- ccache https://ccache.samba.org or clcache https://github.com/frerich/clcache

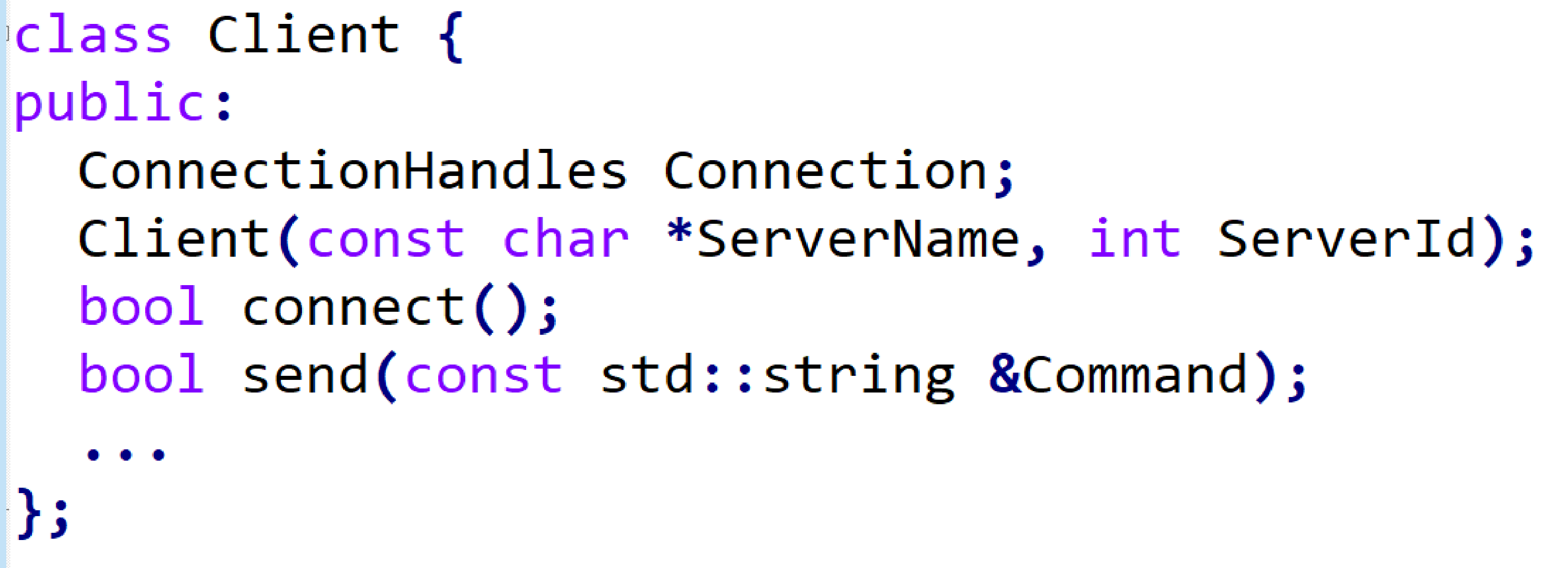
OTHER OPTIONS

ZAPCC APPROACH

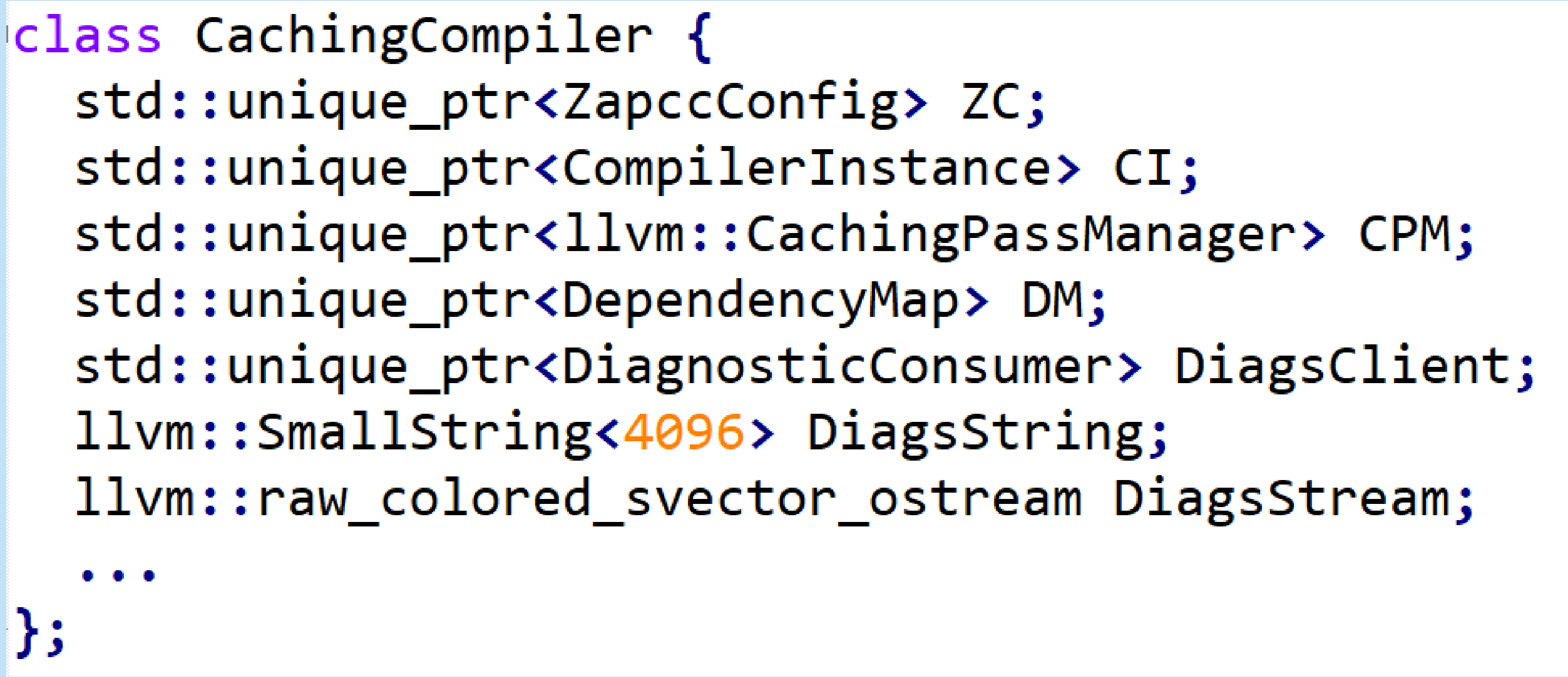
- Heavily-modified clang (300K diff) + out of tree code
- Drop-in replacement
- Avoid duplicate compilation work
- Reuse Source files, AST, IR, Debug info, Machine code
- Too much data to serialize efficiently
- Compilation server, awaits client requests
- Next compilation, unload last main file, modified headers and all dependencies
- Try hard to accomodate existing code without changes (macros)
- Manual mechanisms to hint zapcc in problematic cases

Efficiency is intelligent laziness

CLANG PERSISTENCY



```
class Client {
public:
  ConnectionHandles Connection;
  Client(const char *ServerName, int ServerId);
  bool connect();
  bool send(const std::string &Command);
  ...
};
```



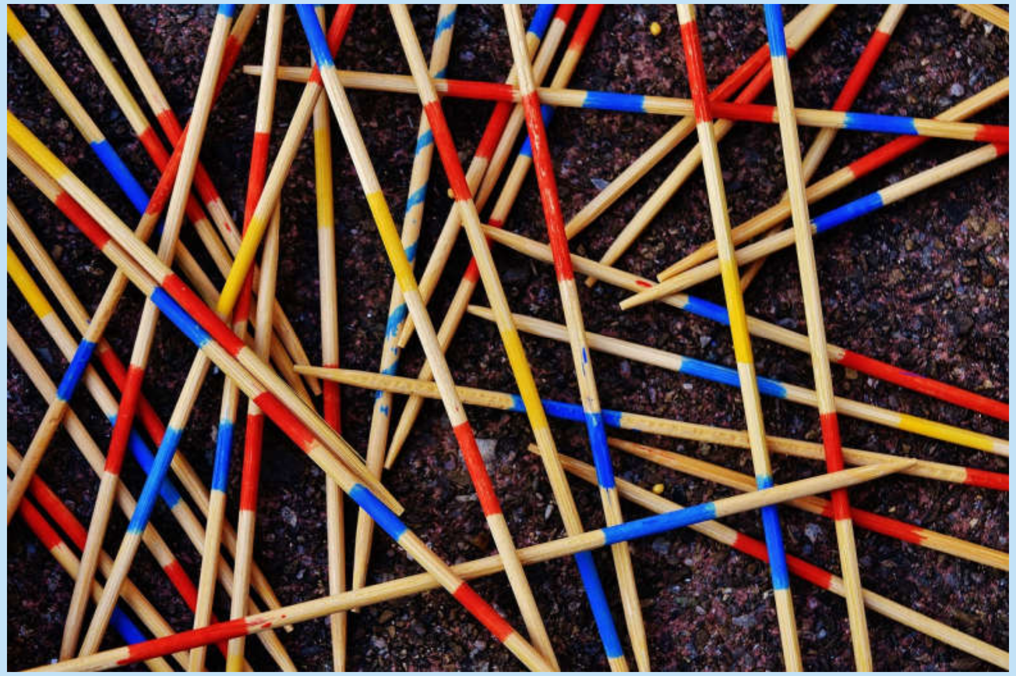
```
class CachingCompiler {
  std::unique_ptr<ZapccConfig> ZC;
  std::unique_ptr<CompilerInstance> CI;
  std::unique_ptr<llvm::CachingPassManager> CPM;
  std::unique_ptr<DependencyMap> DM;
  std::unique_ptr<DiagnosticConsumer> DiagsClient;
  llvm::SmallString<4096> DiagsString;
  llvm::raw_colored_svector_ostream DiagsStream;
  ...
};
```

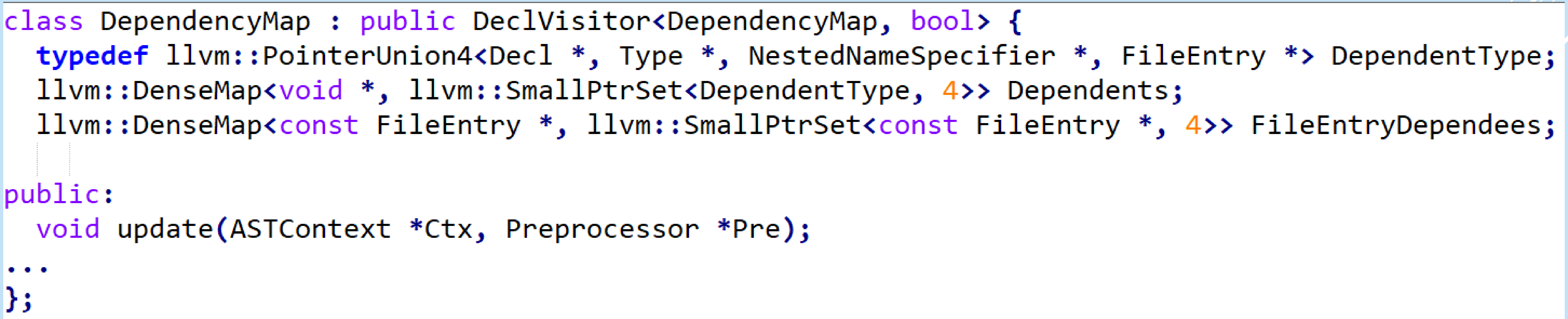
- Client/Server code on Linux & Windows with redirected I/O streams
- *Most* of clang & LLVM state in objects
- Zapcc CachingCompiler class keeping CompilerInstance & helpers
- More patches in CompilerInstance, CompilerInvocation, FrontendAction, InitHeaderSearch
- Logic changes & callbacks to zapcc
- `MaxMemory` server setting
- Cache lost upon memory limit

DEPENDENCIES

- Full dependency map of relevant clang entities
- Also, auxiliary mapping to LLVM entities
- After source file is modified, Zapcc knows which entities are invalidated by walking the map
- Enable updating the compiler state
- Map update overhead ~7% of compilation time

Mikado Is called “Dukim” in Hebrew





```
class DependencyMap : public DeclVisitor<DependencyMap, bool> {
  typedef llvm::PointerUnion4<Decl *, Type *, NestedNameSpecifier *, FileEntry *> DependentType;
  llvm::DenseMap<void *, llvm::SmallPtrSet<DependentType, 4>> Dependents;
  llvm::DenseMap<const FileEntry *, llvm::SmallPtrSet<const FileEntry *, 4>> FileEntryDependees;

public:
  void update(ASTContext *Ctx, Preprocessor *Pre);
...
};
```

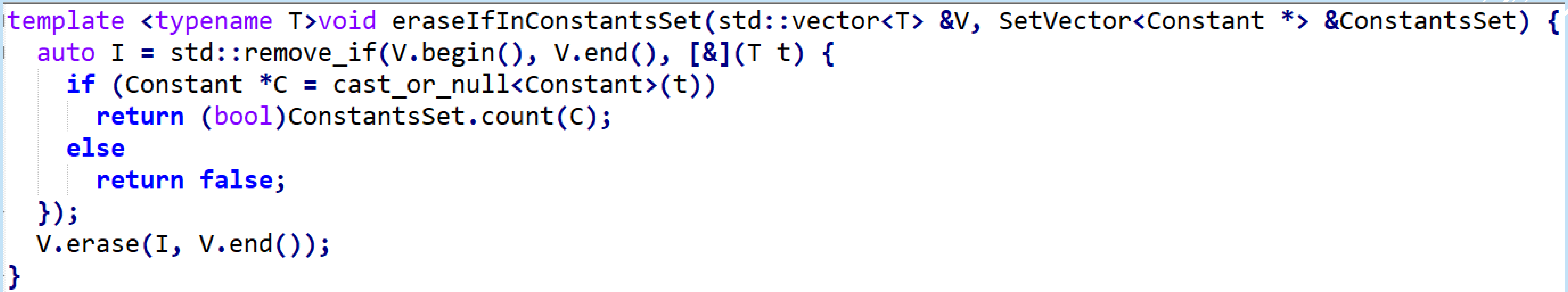
* Code reformatted for presentation only We do follow the 80 columns rule.

UNLOAD

- Cleanly remove set of clang and LLVM entities while keeping hundreds of data structures coherent
- Not really designed to be used like this
- Lots of details and special cases must be exactly right
- While keeping it efficient, ~2% of compilation time

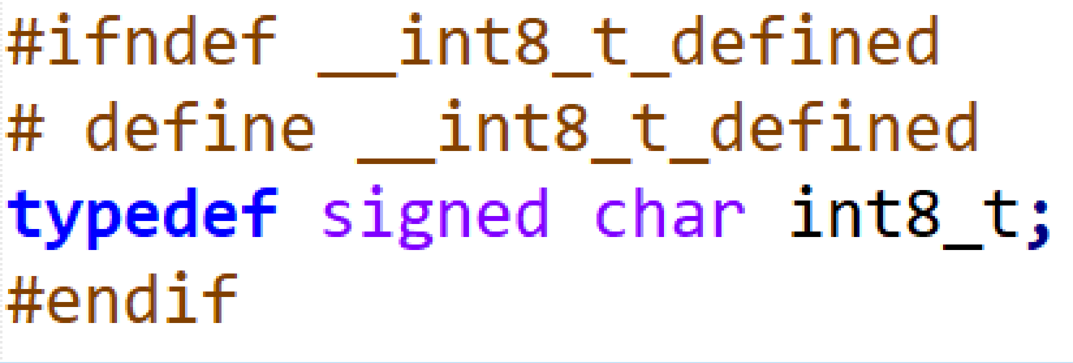
Find the N

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMNMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM



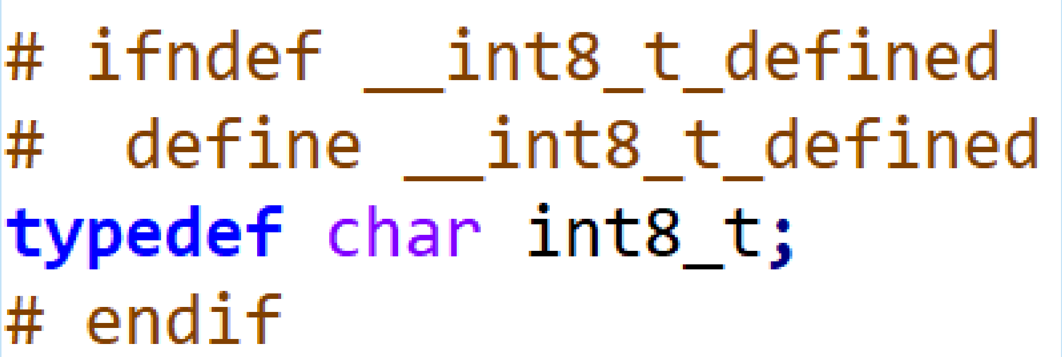
```
template <typename T>void eraseIfInConstantsSet(std::vector<T> &V, SetVector<Constant *> &ConstantsSet) {
  auto I = std::remove_if(V.begin(), V.end(), [&](T t) {
    if (Constant *C = cast_or_null<Constant>(t))
      return (bool)ConstantsSet.count(C);
    else
      return false;
  });
  V.erase(I, V.end());
}
```

/usr/include/stdint.h



```
#ifndef __int8_t_defined
# define __int8_t_defined
typedef signed char int8_t;
#endif
```

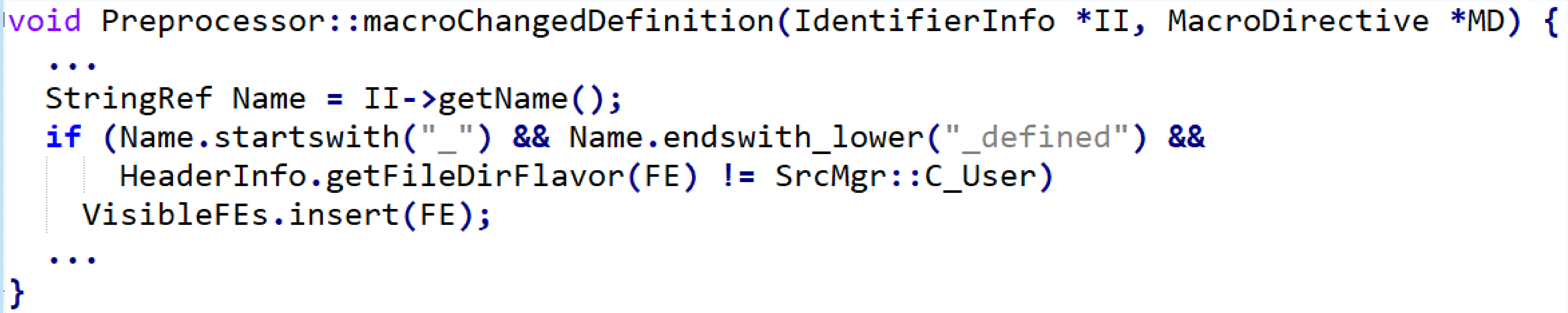
/usr/include/x86_64-linux-gnu/sys/types.h



```
# ifndef __int8_t_defined
#  define __int8_t_defined
typedef char int8_t;
# endif
```

COMPATIBILITY: SYSTEM MACROS

- Common C&P pattern in system headers
- Once this pattern is identified, the header becomes “sticky” and will stay visible later
- Special handling of **__need_** macros
- With these rules zapcc caches system includes without source changes or specific hints



```
void Preprocessor::macroChangedDefinition(IdentifierInfo *II, MacroDirective *MD) {
  ...
  StringRef Name = II->getName();
  if (Name.startswith("_") && Name.endswith_lower("_defined") &&
      HeaderInfo.getFileDirFlavor(FE) != SrcMgr::C_User)
    VisibleFEs.insert(FE);
  ...
}
```

COMPATIBILITY: USER MACROS

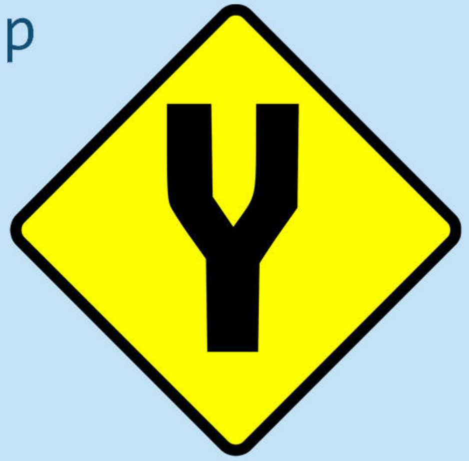
```
[WatchMacro]
# Eigen
EIGEN_TEST_FUNC
# libcxx
_LIBCPP_DEBUG
# LLVM
GET_INSTRINFO_CTOR_DTOR
GET_INSTRINFO_NAMED_OPS
GET_INSTRINFO_MC_DESC
GET_INSTRMAP_INFO
GET_LLVM_INTRINSIC_FOR_MS_BUILTIN
GET_REGINFO_TARGET_DESC
GET_SUBTARGETINFO_MC_DESC
GET_SUBTARGETINFO_TARGET_DESC
DONT_GET_PLUGIN_LOADER_OPTION
# MongoDB
MONGO_LOG_DEFAULT_COMPONENT
```

- Whereas modules are isolated from macros, real world code is not
  A Module System for C++ (Revision 4)
- Zapcc automatically detects most macro-dependent headers
- Based upon macro usage pattern, to avoid too many false positives
- If the macro changes value, zapcc invalidates the cache
- Optional list of manually-added macros in config file
- Typically only few macros per project
- The combination works well in practice

COMPATIBILITY: NON CACHED

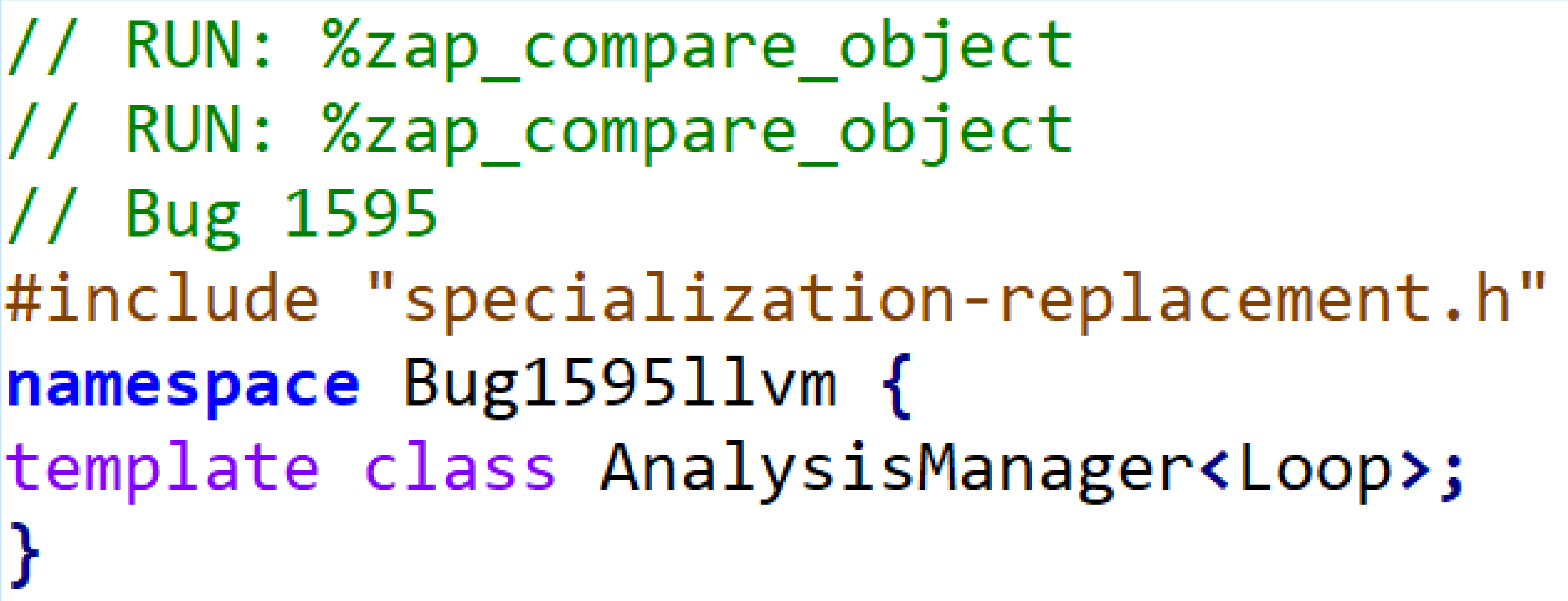
```
[DoNotZap]
# Boost
*/libs/python/test/result.cpp
# LibreOffice (aBibliographyDataFieldMap)
*/xmloff/source/text/XMLSectionExport.cxx
# Qt
*/qtdeclarative/src/qml/jsruntime/qv4object.cpp
*/qtwebengine/src/core/content_client_qt.cpp
# XROOTD (#define _FILE_OFFSET_BITS)
*/src/XrdPosix/XrdPosixPreload32.cc
# webkit
*/WebCore/Modules/indexeddb/client/IDBOpenDBRequestImpl.cpp
```

- zapcc will not cache or use cache for these files == compile with clang
- Rarely used, to the left is the **full** non cached list for over 40 open source projects
- Required in exceptional cases
- Such cases are better fixed in the source code with trivial code patches



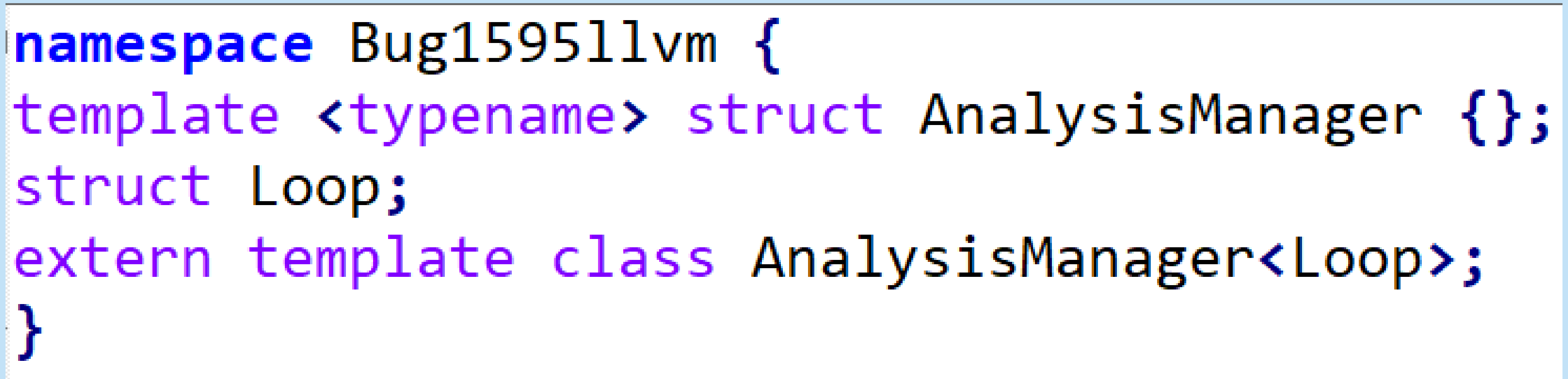
VALIDATION

specialization-replacement.h



```
// RUN: %zap_compare_object
// RUN: %zap_compare_object
// Bug 1595
#include "specialization-replacement.h"
namespace Bug1595llvm {
template class AnalysisManager<Loop>;
}
```

specialization-replacement.cpp



```
namespace Bug1595llvm {
template <typename> struct AnalysisManager {};
struct Loop;
extern template class AnalysisManager<Loop>;
}
```

- More time spent on testing than development
- CI runs builds 44 open source packages including (partial list)
- bitshares, cmake, codeblocks, ETL, folly, ITK, LLVM, mongodb, root, scylladb, vexcl
- Build & run regression tests (where available) that clang passes
- About 600 more custom tests, mostly generated using creduce & manual reduce

Expression Templates Library (ETL), Baptiste Wicht

https://github.com/wichtounet/etl/

ZAPCC BETA VS CLANG 3.9 VS GCC 5.4.0 BUILD TIME

	Debug			Release
Compiler	-j1	-j2	-j4	-j1
g++-5.4.0	469s	230s	130s	782s
clang++-3.9	710s	371s	218s	960s
zapcc++ (beta)	214s	112s	66s	640s
Speedup VS Clang	X3.31	X3.31	X3.3	X1.5
Speedup VS GCC	X2.19	X2.05	X1.96	X1.22

https://baptiste-wicht.com/posts/html.39-clang-and-54-gcc-against-speed-compilation-cpp-zapcc/2016/12

Even better results for ZAPCC 1.0 (–j4 only)

https://baptiste-wicht.com/posts/2017/03/release-zapcc-10-fast-cpp-compiler.html

RECOMPILE

File name	File size	Speedup
database.cc	193864	X1.32
mutation_partition.cc	78913	X2.4
main.cc	31967	X2.5
query.cc	9908	X9.7
clocks-impl.cc	780	X43

FULL BUILD

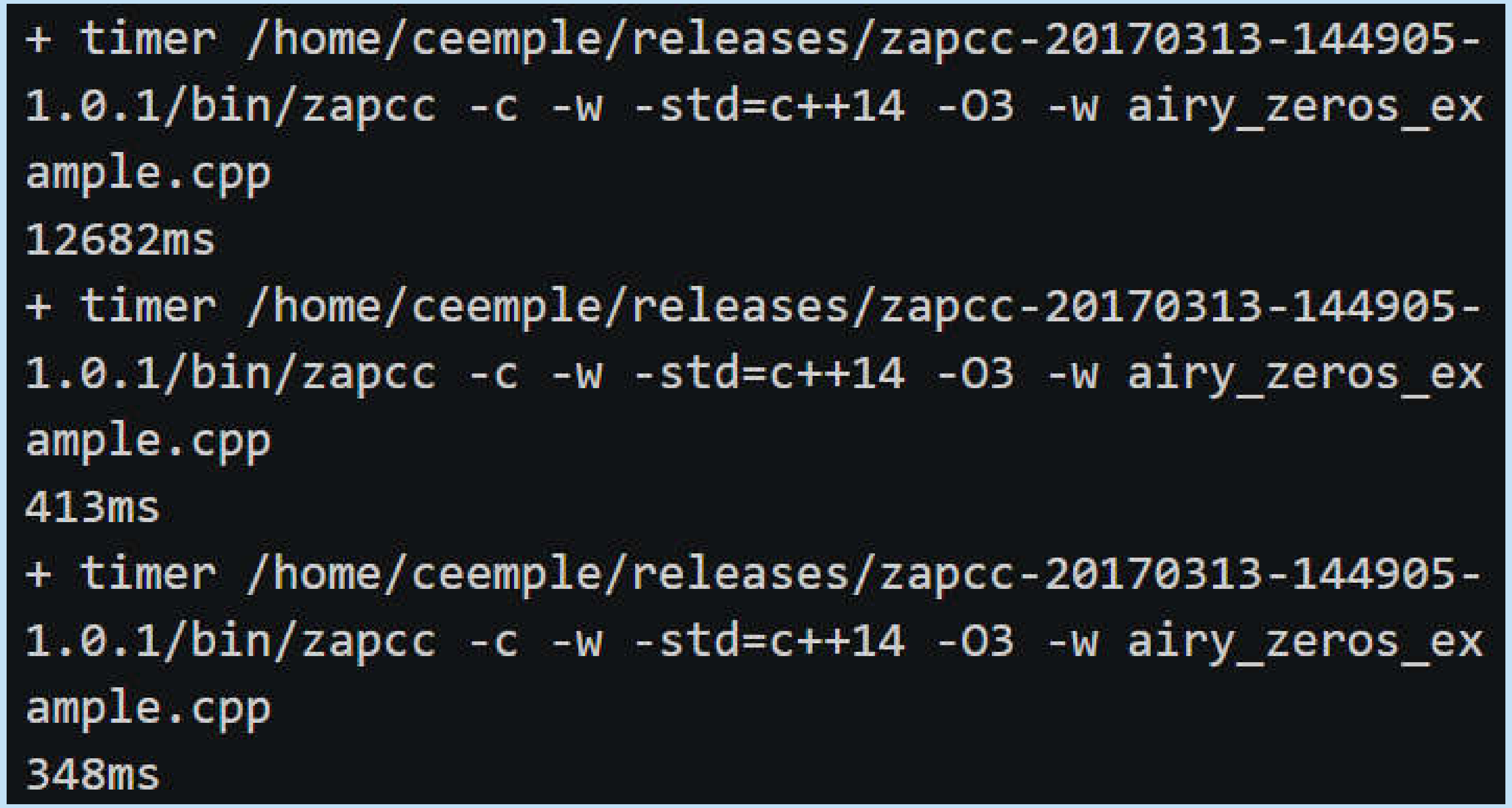
# Cores	4	5	6	7	8	9
Speedup	X1.51	X1.55	X1.56	X1.55	X1.55	X1.5

```
/.configure.py --compiler clang --cflags=-w
ninja clean
ninja -j N build/release/scylla
```

SCYALLADB

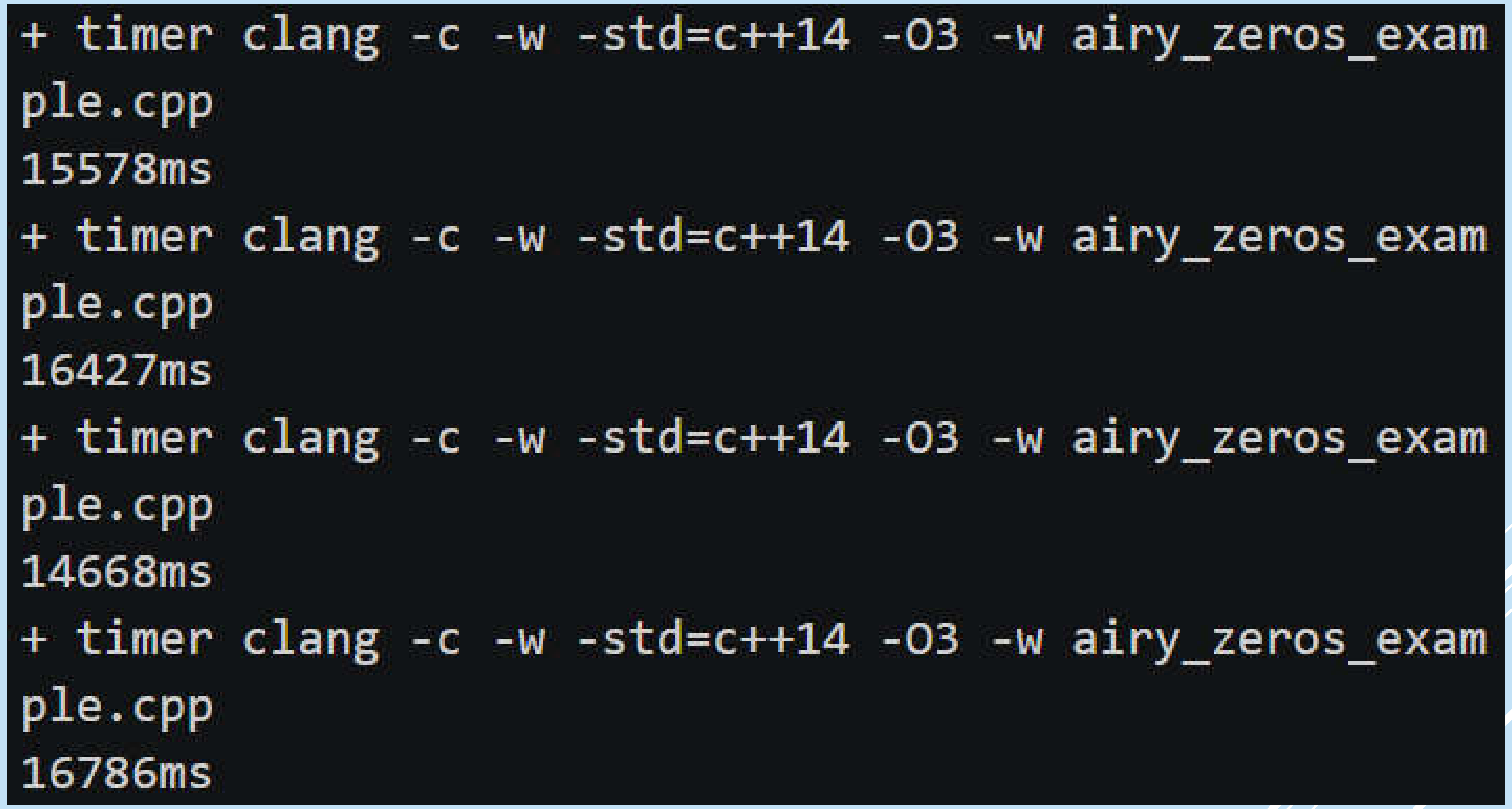
- Clang relative to zapcc
- Trunk, July 2017
- Measure wall time
- `-O2`
- `MaxMemory=4000`

Zapcc 1.0.1



```
+ timer /home/ceemple/releases/zapcc-20170313-144905-
1.0.1/bin/zapcc -c -w -std=c++14 -O3 -w airy_zeros_ex
ample.cpp
12682ms
+ timer /home/ceemple/releases/zapcc-20170313-144905-
1.0.1/bin/zapcc -c -w -std=c++14 -O3 -w airy_zeros_ex
ample.cpp
413ms
+ timer /home/ceemple/releases/zapcc-20170313-144905-
1.0.1/bin/zapcc -c -w -std=c++14 -O3 -w airy_zeros_ex
ample.cpp
348ms
```

Clang 5.0.0 (r298211)



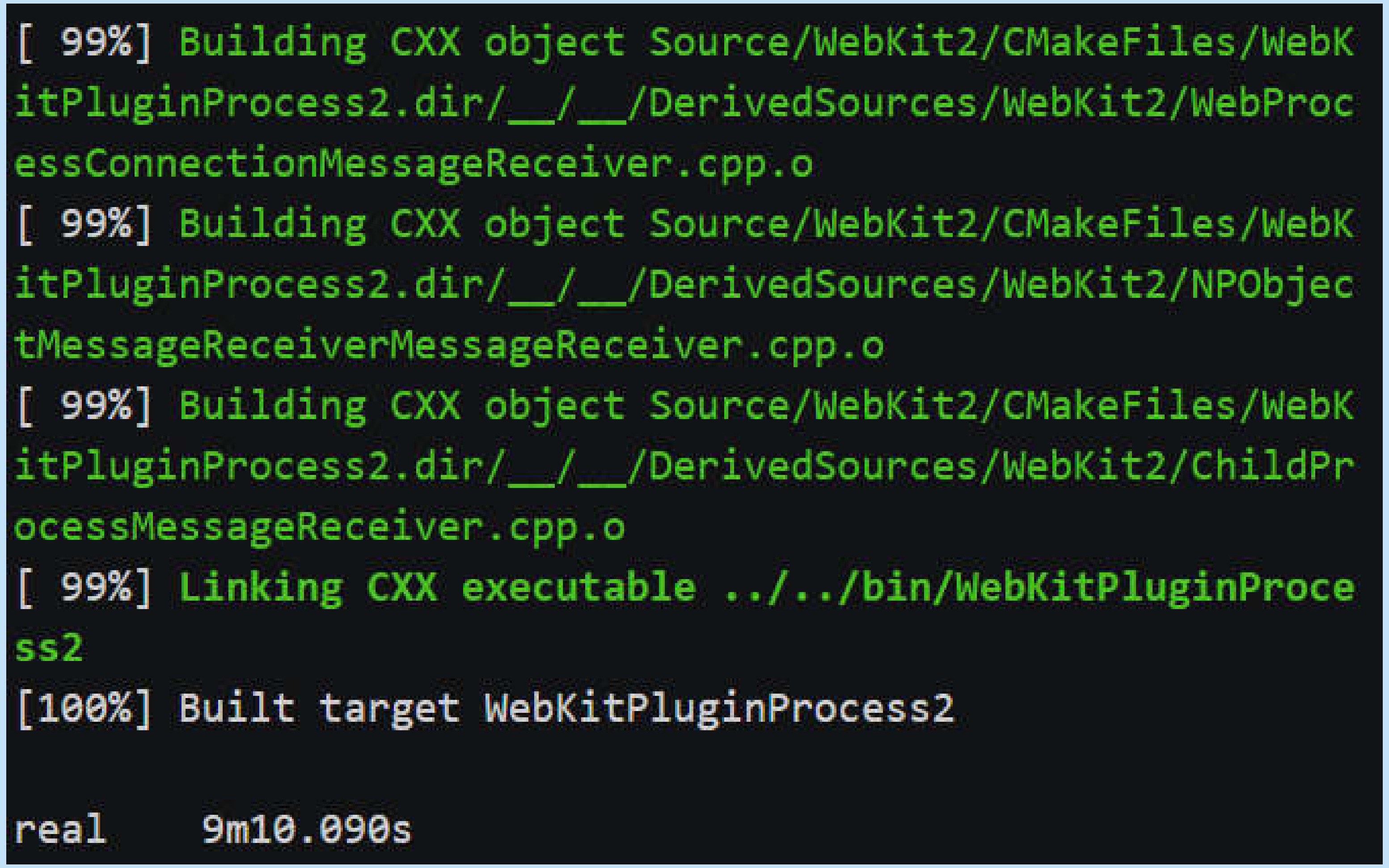
```
+ timer clang -c -w -std=c++14 -O3 -w airy_zeros_exam
ple.cpp
15578ms
+ timer clang -c -w -std=c++14 -O3 -w airy_zeros_exam
ple.cpp
16427ms
+ timer clang -c -w -std=c++14 -O3 -w airy_zeros_exam
ple.cpp
14668ms
+ timer clang -c -w -std=c++14 -O3 -w airy_zeros_exam
ple.cpp
16786ms
```

BOOST::MATH EXAMPLE RECOMPILATION

40X FASTER USING ZAPCC

INTEL(R) CORE(TM) I7-4790, 6 CPUS, 16GB, UBUNTU 16.04.2 LTS

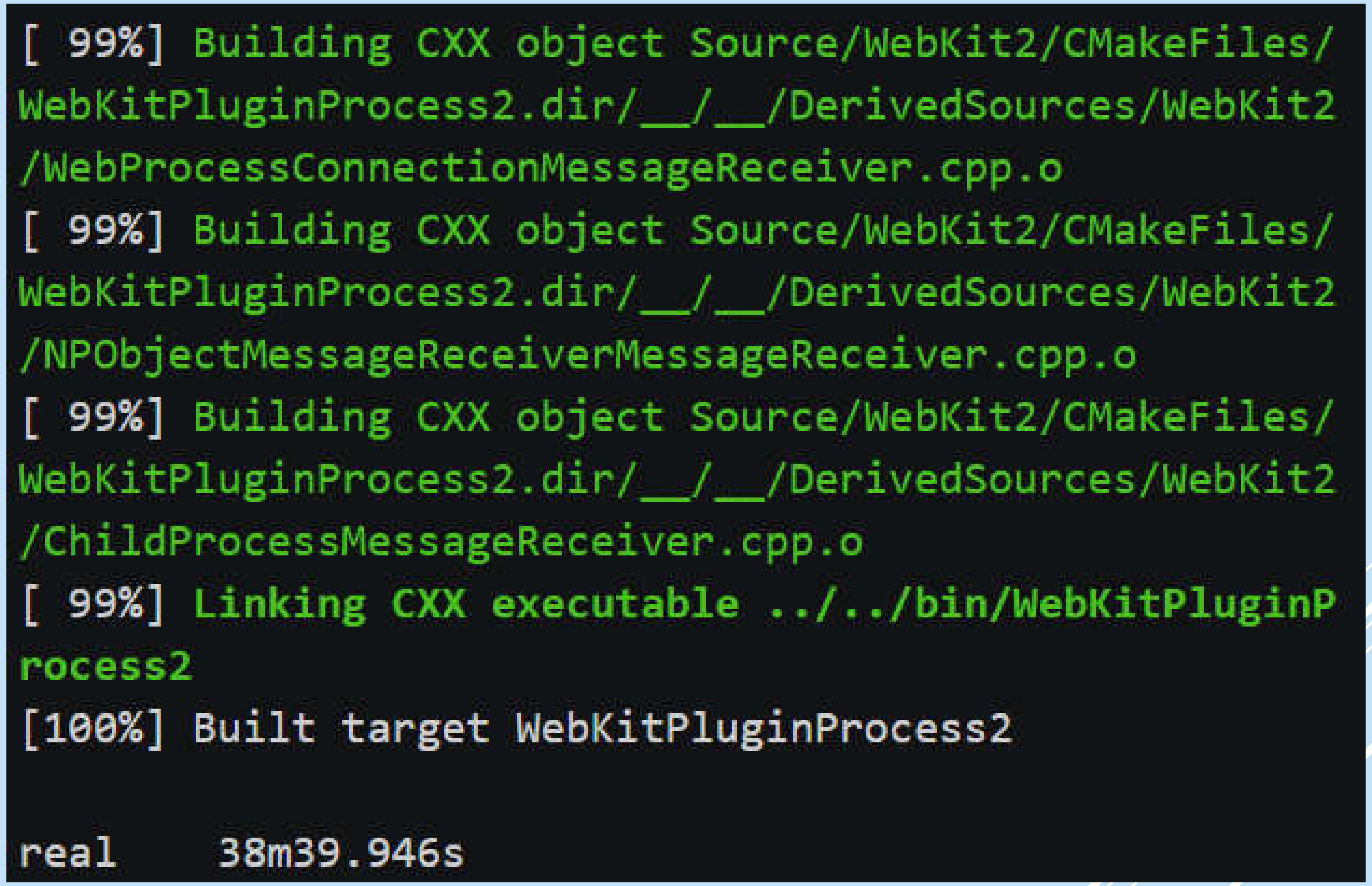
Zapcc 1.0



```
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/WebK
itPluginProcess2.dir/__/__/DerivedSources/WebKit2/WebProc
essConnectionMessageReceiver.cpp.o
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/WebK
itPluginProcess2.dir/__/__/DerivedSources/WebKit2/NPObjec
tMessageReceiverMessageReceiver.cpp.o
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/WebK
itPluginProcess2.dir/__/__/DerivedSources/WebKit2/ChildPr
ocessMessageReceiver.cpp.o
[ 99%] Linking CXX executable ../../bin/WebKitPluginProce
ss2
[100%] Built target WebKitPluginProcess2

real    9m10.090s
```

Clang 4.0.0 (r291267)



```
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/
WebKitPluginProcess2.dir/__/__/DerivedSources/WebKit2
/WebProcessConnectionMessageReceiver.cpp.o
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/
WebKitPluginProcess2.dir/__/__/DerivedSources/WebKit2
/NPObjectMessageReceiverMessageReceiver.cpp.o
[ 99%] Building CXX object Source/WebKit2/CMakeFiles/
WebKitPluginProcess2.dir/__/__/DerivedSources/WebKit2
/ChildProcessMessageReceiver.cpp.o
[ 99%] Linking CXX executable ../../bin/WebKitPluginP
rocess2
[100%] Built target WebKitPluginProcess2

real    38m39.946s
```

WEBKIT FULL BUILD

4X FASTER USING ZAPCC
INTEL(R) CORE(TM) I7-4790, 6 CPUS, 16GB, UBUNTU 16.04.2 LTS

Q & A