Clang and The Future Of C++ Tooling

Overview

- Why some tools are terrible.
- Introduction to Clang/LLVM and modern compiler architecture.
  - Stages and source code representations in the clang pipeline.
  - Clang APIs with a focus on the C++ API.
- Tales of developing a clang-tidy plugin.

How To Disable Unit Tests?

```cpp
#include <cxxtest/TestSuite.h>

class MyTests : public CxxTest::TestSuite {
public:

  void testSubtraction() {
    TS_ASSERT_EQUALS(1 - 1, 0);
  }

  void testAddition() {
    TS_ASSERT_EQUALS(1 + 1, 2);
  }
  // more test cases below...
};
```

How To Disable Unit Tests?

Attempt 1

```
#include <cxxtest/TestSuite.h>

class MyTests : public CxxTest::TestSuite {
public:
  /*
  void testSubtraction() {
    TS_ASSERT_EQUALS(1 - 1, 0);
  }

  void testAddition() {
    TS_ASSERT_EQUALS(1 + 1, 2);
  }
  */
  // more test cases below...
};
```

How To Disable Unit Tests?

Attempt 2

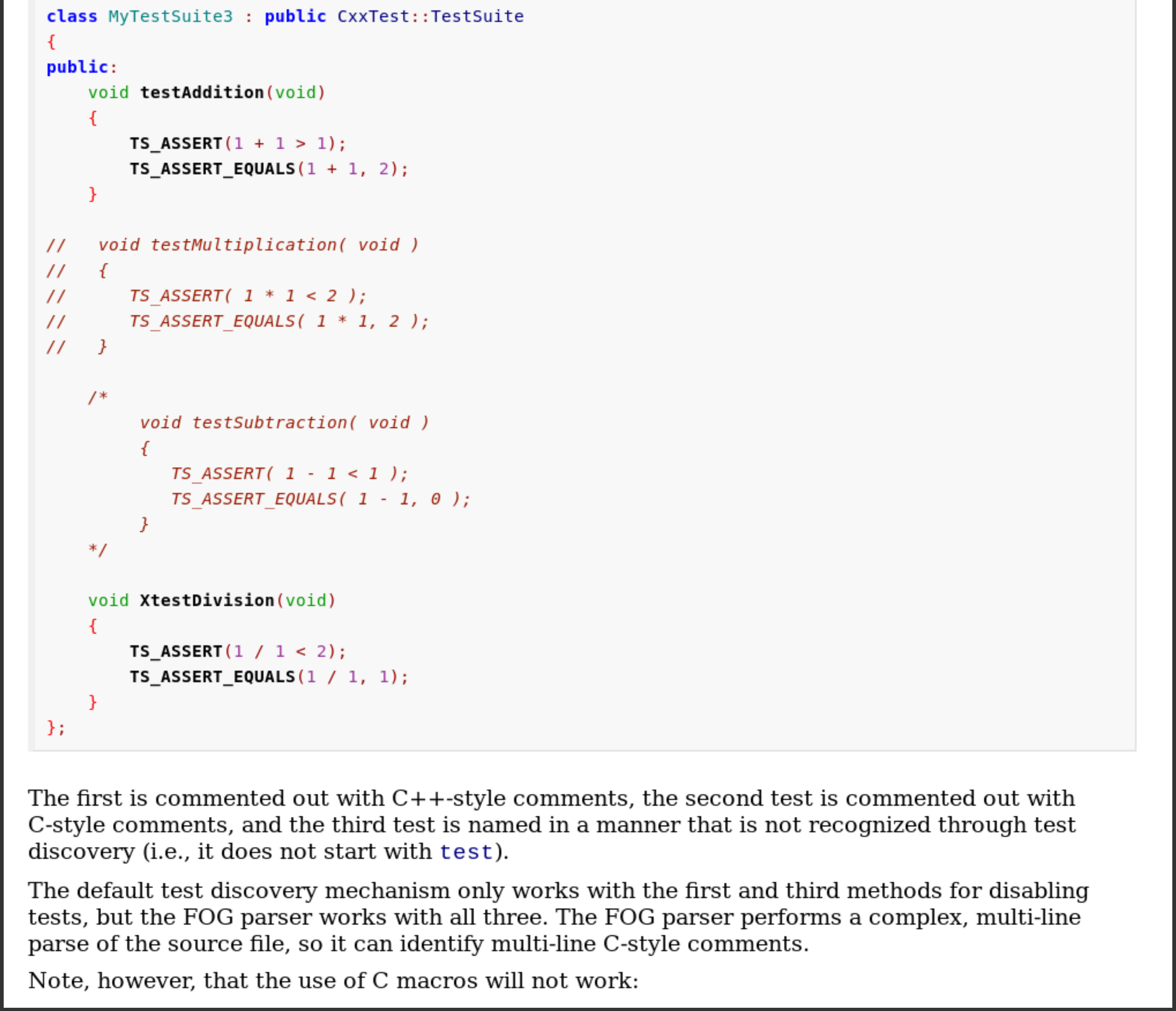
```
#include <cxxtest/TestSuite.h>

class MyTests : public CxxTest::TestSuite {
public:
  #ifdef 0
    void testSubtraction() {
      TS_ASSERT_EQUALS(1 - 1, 0);
    }

    void testAddition() {
      TS_ASSERT_EQUALS(1 + 1, 2);
    }
  #endif
  // more test cases below...
};
```

How To Disable Unit Tests?

Attempt 3

```
#include <cxxtest/TestSuite.h>

class MyTests : public CxxTest::TestSuite {
public:
  //
  //  void testSubtraction() {
  //    TS_ASSERT_EQUALS(1 - 1, 0);
  //  }
  //
  //  void testAddition() {
  //    TS_ASSERT_EQUALS(1 + 1, 2);
  //  }
  //
  // more test cases below...
};
```

More Detail Here...



```
class MyTestSuite3 : public CxxTest::TestSuite
{
public:
    void testAddition(void)
    {
        TS_ASSERT(1 + 1 > 1);
        TS_ASSERT_EQUALS(1 + 1, 2);
    }

//   void testMultiplication( void )
//   {
//      TS_ASSERT( 1 * 1 < 2 );
//      TS_ASSERT_EQUALS( 1 * 1, 2 );
//   }

    /*
         void testSubtraction( void )
         {
            TS_ASSERT( 1 - 1 < 1 );
            TS_ASSERT_EQUALS( 1 - 1, 0 );
         }
    */

    void XtestDivision(void)
    {
        TS_ASSERT(1 / 1 < 2);
        TS_ASSERT_EQUALS(1 / 1, 1);
    }
};
```

The first is commented out with C++-style comments, the second test is commented out with C-style comments, and the third test is named in a manner that is not recognized through test discovery (i.e., it does not start with `test`).

The default test discovery mechanism only works with the first and third methods for disabling tests, but the FOG parser works with all three. The FOG parser performs a complex, multi-line parse of the source file, so it can identify multi-line C-style comments.

Note, however, that the use of C macros will not work:

http://cxxtest.com/guide.html#_test_discovery_options

How Did We Get Here?

- Lack of open source C++ parsers – GCC contains an implementation but it doesn’t have a clean separation from the rest of the compiler.
- GPL may be prohibitive for companies wishing to sell their tools.
- NIH Syndrome perhaps more common amongst C++ developers who are likely the ones developing the tools for C++.

=> Everyone writes their own parser.

But Parsing C++ is Hard

```
#ifdef MAGIC
  template<int N>
  struct X {
    X(int i) {}
  };
#else
  int X = 100;
#endif
  void test() {
    int constexpr N = 200;
    auto a = X<N>(0);     // decltype(a) == X<200> or
    auto a = X < N > (0); // decltype(a) == bool
  }
```

Really, Really Hard

```
template <int n> struct confusing { static int q; };

template <> struct confusing<1> {
  template <int n> struct q {
    q(int x) {
      printf("Separated syntax and semantics.\n");
    }
    operator int() { return 0; }
  };
};

char ch;
int main() {
  int x = confusing<sizeof(ch)>::q<3>(2);
  return 0;
}
```

Introducing Clang/LLVM

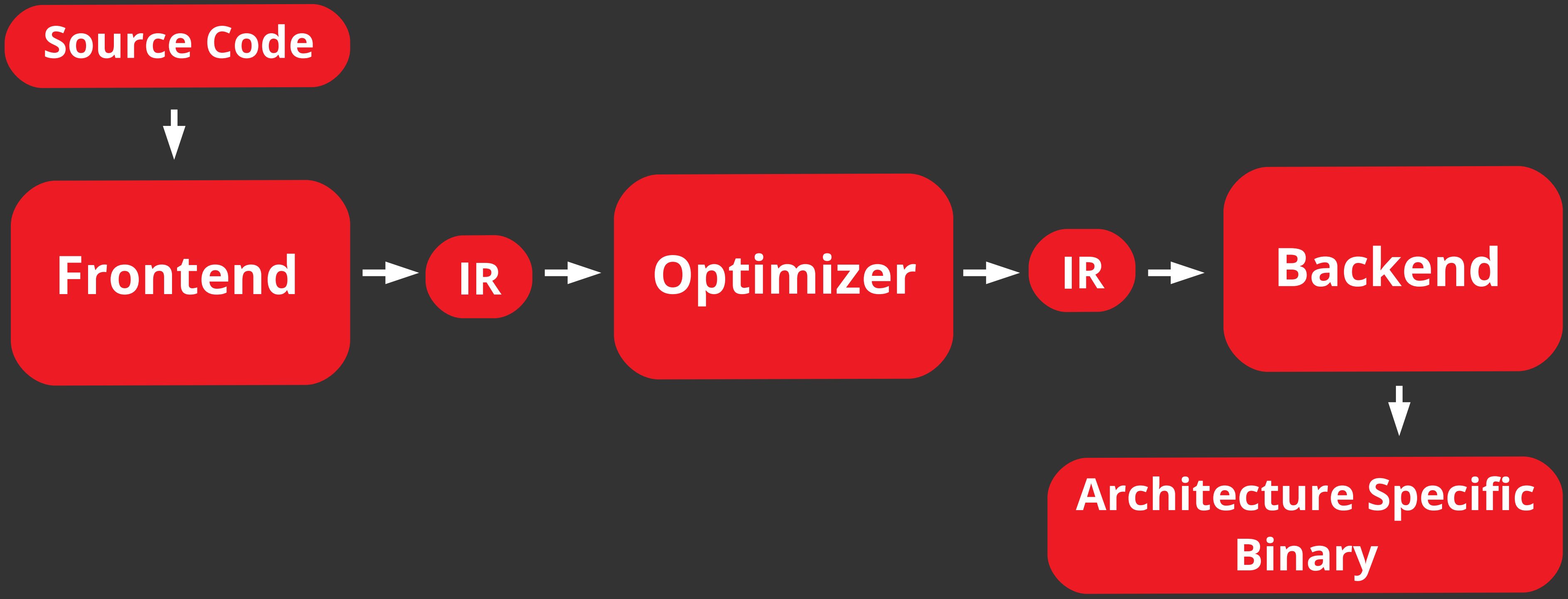
What Is LLVM?

- A collection of open source (BSD) tools and libraries written in C++ which can be used to build programming language implementations.
- Can perform compile, link and run time optimization of programs written in arbitrary programming languages.

What Is Clang?

- A sub-project of LLVM.
- Library-based compiler front end for C, C++ and Objective-C.
- GCC Compatible.

Modern Compiler Architecture



Source Code

Frontend

IR

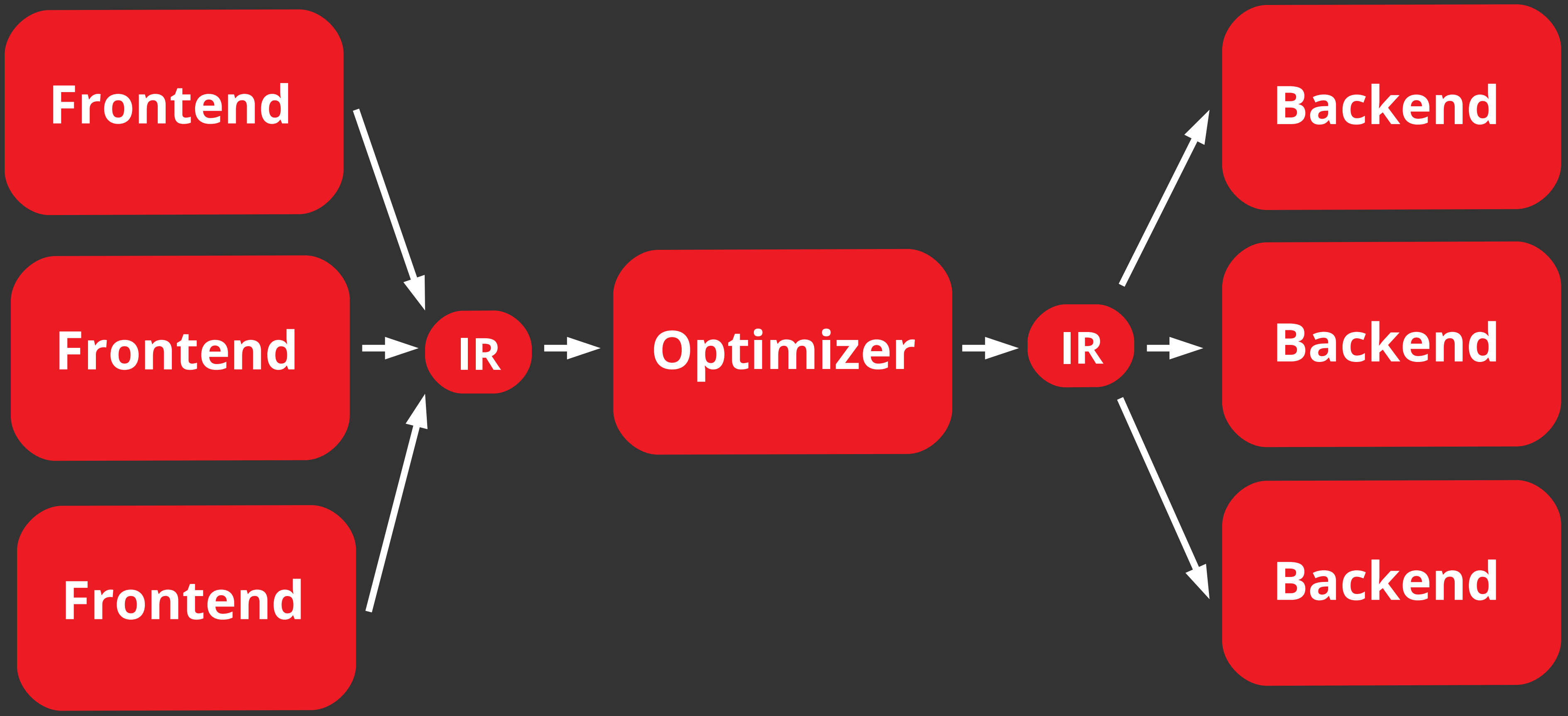
Optimizer

IR

Backend

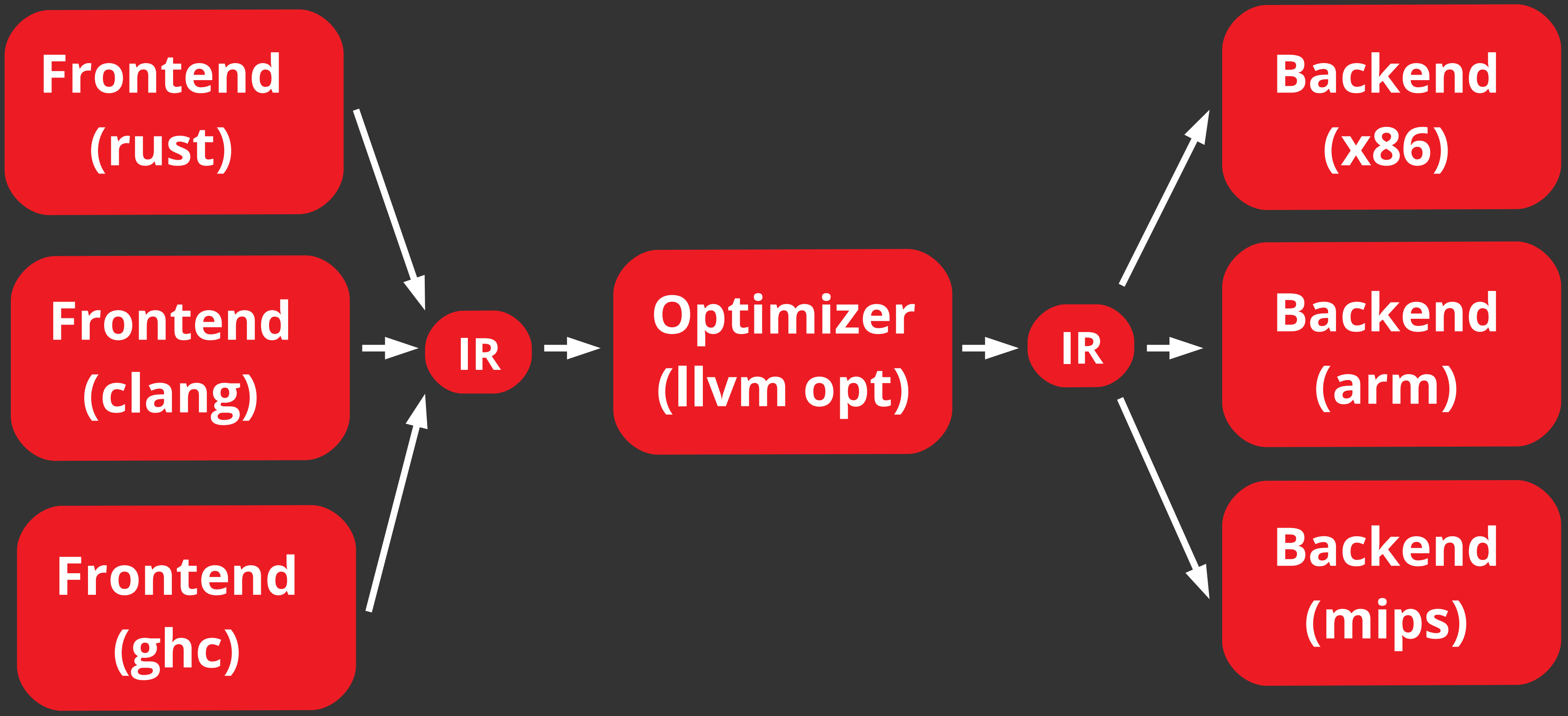
Architecture Specific Binary

Modern Compiler Architecture



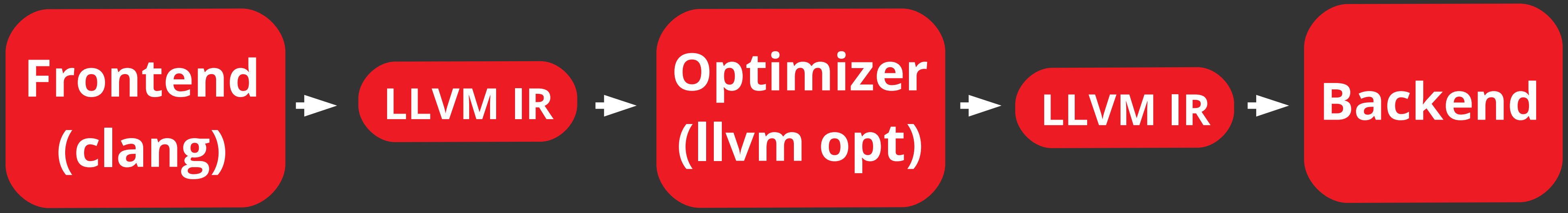
This scales to an arbitrary number of **front ends** and **architectures**.

Modern Compiler Architecture



Allowing us to write *N* front ends and *M* back ends rather than *N* * *M* individual compilers.

Back To One Pipeline

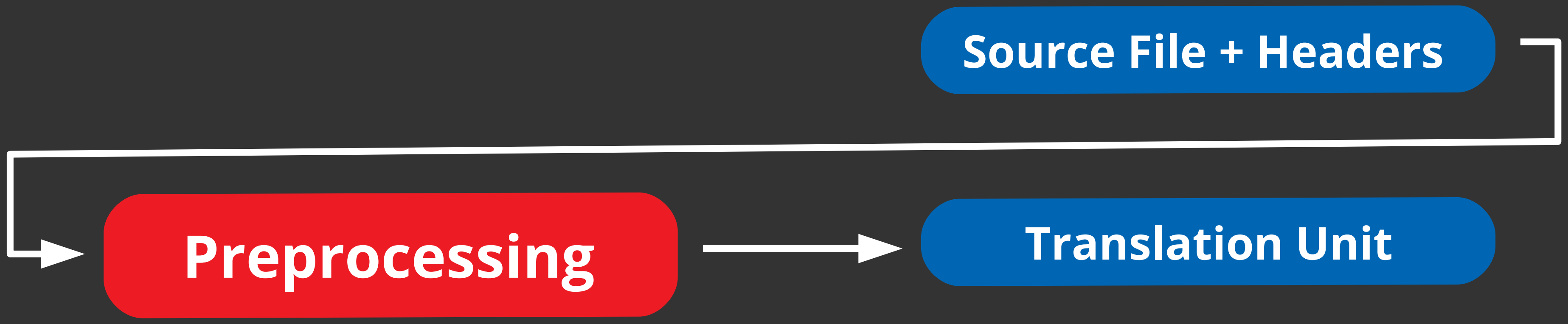

Frontend (clang) → LLVM IR → Optimizer (llvm opt) → LLVM IR → Backend

Back To One Pipeline

Frontend (clang)

Preprocessing

```
$ clang -E main.cpp -o preprocessed-main.cpp \
    -std=c++11
```

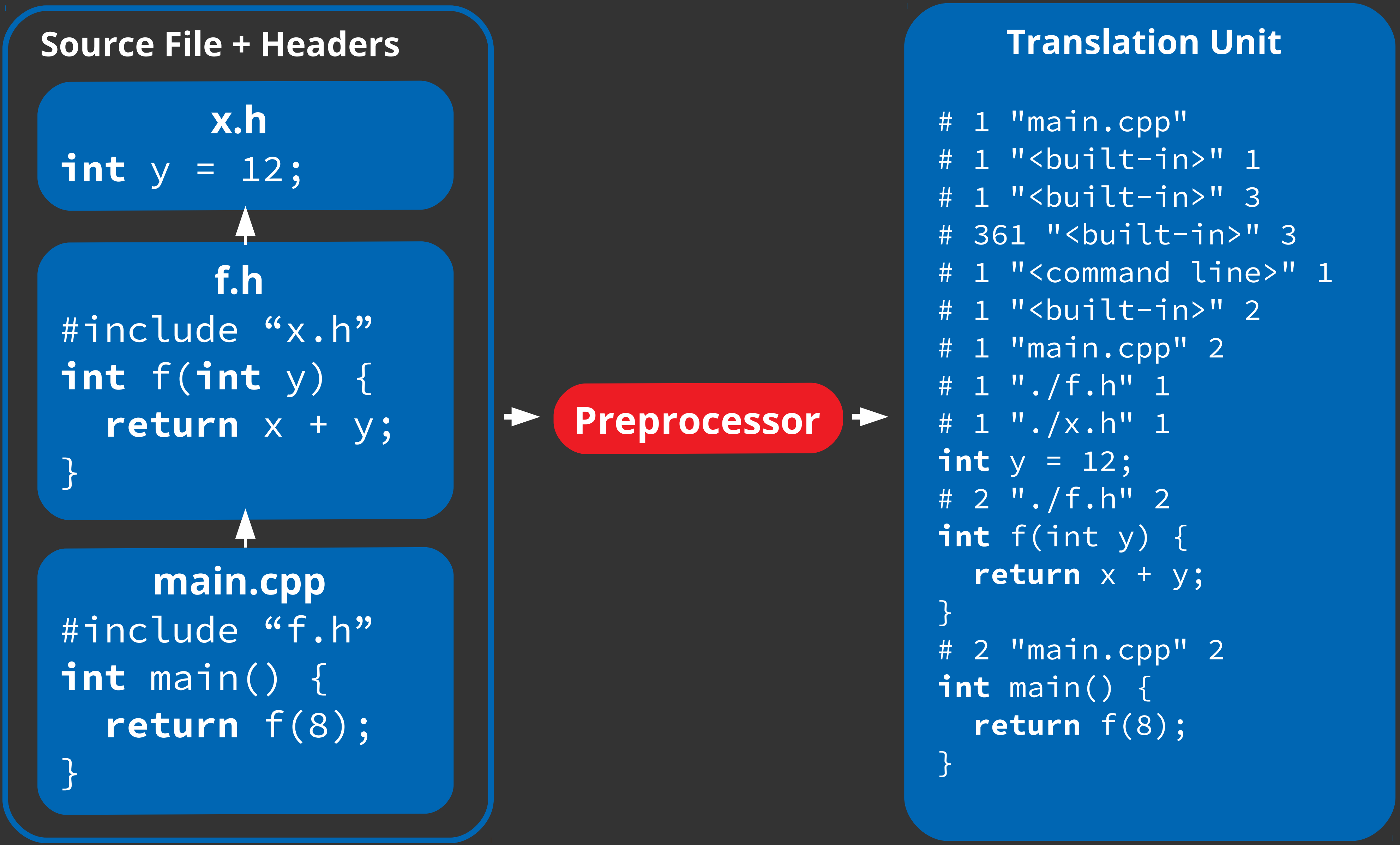


Source File + Headers

Preprocessing

Translation Unit

Preprocessing



Source File + Headers

x.h

```
int y = 12;
```

f.h

```
#include “x.h”
int f(int y) {
  return x + y;
}
```

main.cpp

```
#include “f.h”
int main() {
  return f(8);
}
```

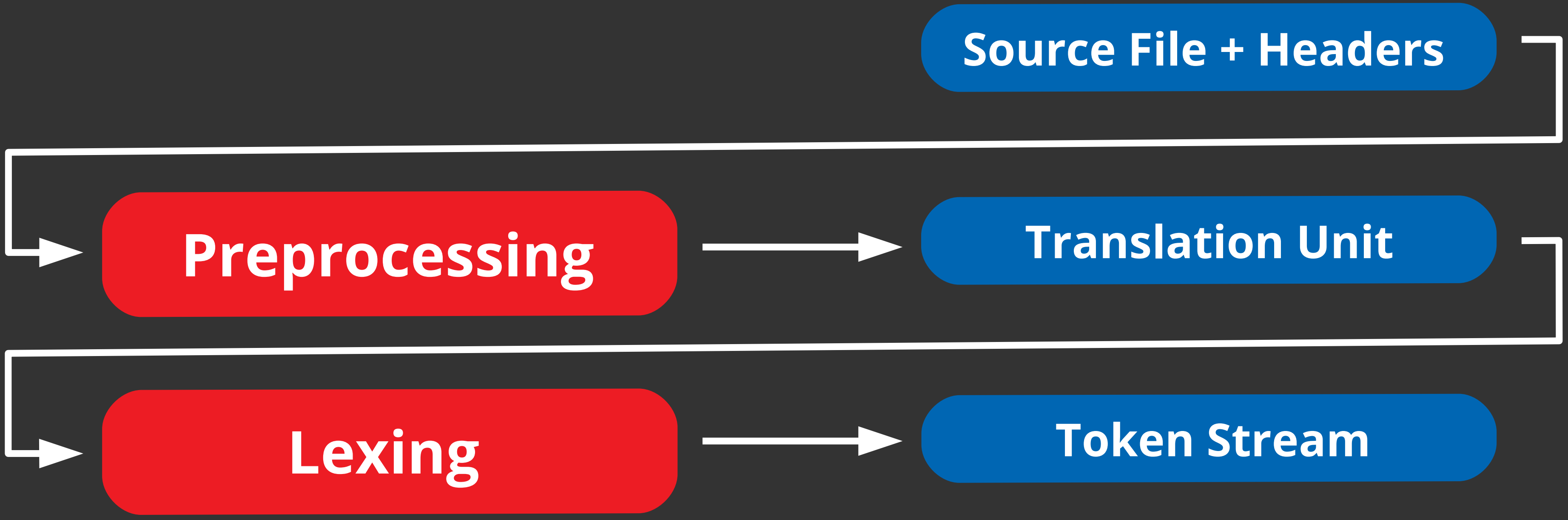
Preprocessor

Translation Unit

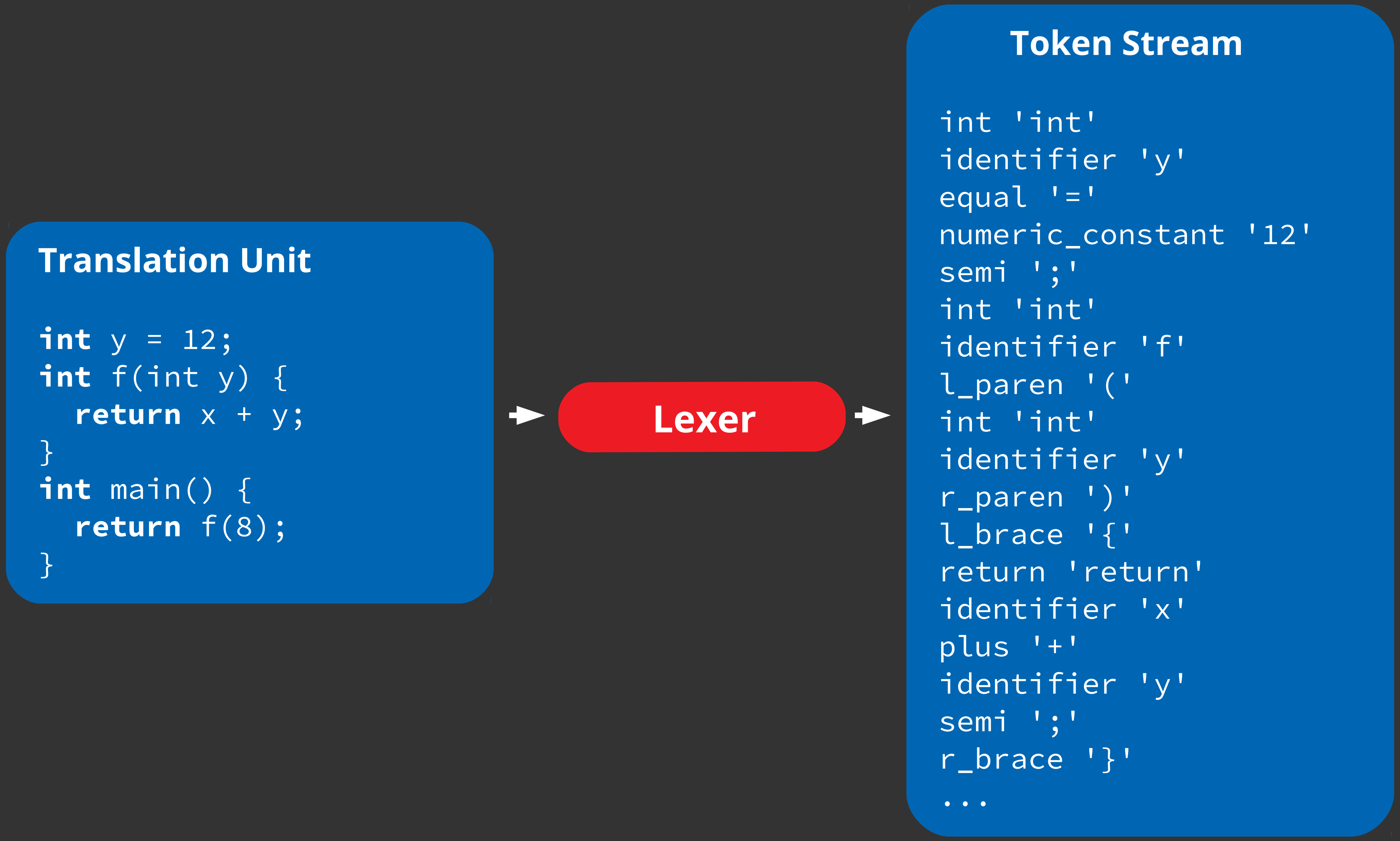
```
# 1 "main.cpp"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 361 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "main.cpp" 2
# 1 "./f.h" 1
# 1 "./x.h" 1
int y = 12;
# 2 "./f.h" 2
int f(int y) {
  return x + y;
}
# 2 "main.cpp" 2
int main() {
  return f(8);
}
```

Lexing

```
$ clang -fsyntax-only -Xclang -dump-tokens file.cpp \
    -std=c++11
```



Lexing



Translation Unit

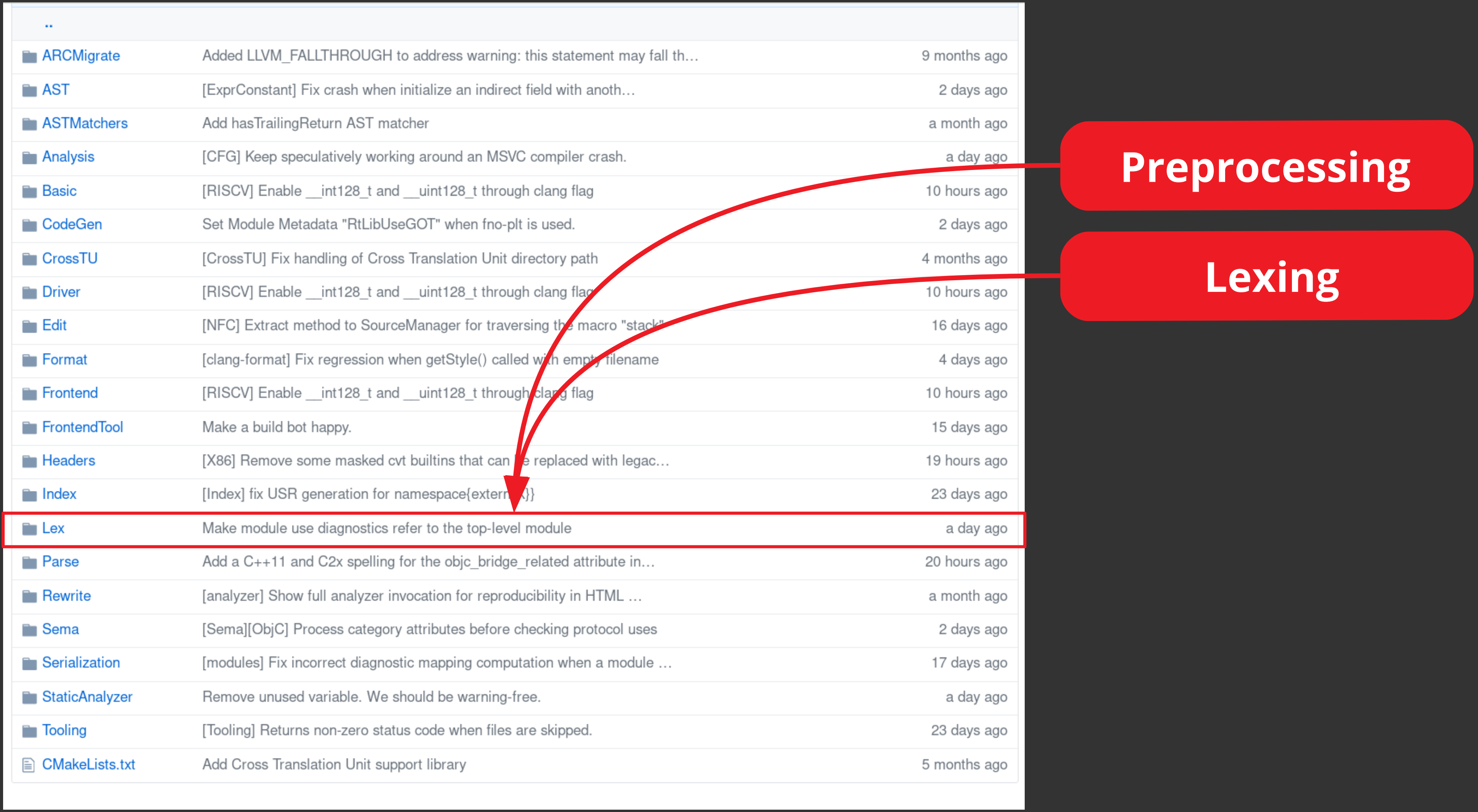
```
int y = 12;
int f(int y) {
  return x + y;
}
int main() {
  return f(8);
}
```

Lexer

Token Stream

```
int 'int'
identifier 'y'
equal '='
numeric_constant '12'
semi ';'
int 'int'
identifier 'f'
l_paren '('
int 'int'
identifier 'y'
r_paren ')'
l_brace '{'
return 'return'
identifier 'x'
plus '+'
identifier 'y'
semi ';'
r_brace '}'
...
```

Lex Library



..		
ARCMigrate	Added LLVM_FALLTHROUGH to address warning: this statement may fall th…	9 months ago
AST	[ExprConstant] Fix crash when initialize an indirect field with anoth…	2 days ago
ASTMatchers	Add hasTrailingReturn AST matcher	a month ago
Analysis	[CFG] Keep speculatively working around an MSVC compiler crash.	a day ago
Basic	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
CodeGen	Set Module Metadata "RtLibUseGOT" when fno-plt is used.	2 days ago
CrossTU	[CrossTU] Fix handling of Cross Translation Unit directory path	4 months ago
Driver	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
Edit	[NFC] Extract method to SourceManager for traversing the macro "stack"	16 days ago
Format	[clang-format] Fix regression when getStyle() called with empty filename	4 days ago
Frontend	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
FrontendTool	Make a build bot happy.	15 days ago
Headers	[X86] Remove some masked cvt builtins that can be replaced with legac…	19 hours ago
Index	[Index] fix USR generation for namespace{extern{X}}	23 days ago
Lex	Make module use diagnostics refer to the top-level module	a day ago
Parse	Add a C++11 and C2x spelling for the objc_bridge_related attribute in…	20 hours ago
Rewrite	[analyzer] Show full analyzer invocation for reproducibility in HTML …	a month ago
Sema	[Sema][ObjC] Process category attributes before checking protocol uses	2 days ago
Serialization	[modules] Fix incorrect diagnostic mapping computation when a module …	17 days ago
StaticAnalyzer	Remove unused variable. We should be warning-free.	a day ago
Tooling	[Tooling] Returns non-zero status code when files are skipped.	23 days ago
CMakeLists.txt	Add Cross Translation Unit support library	5 months ago

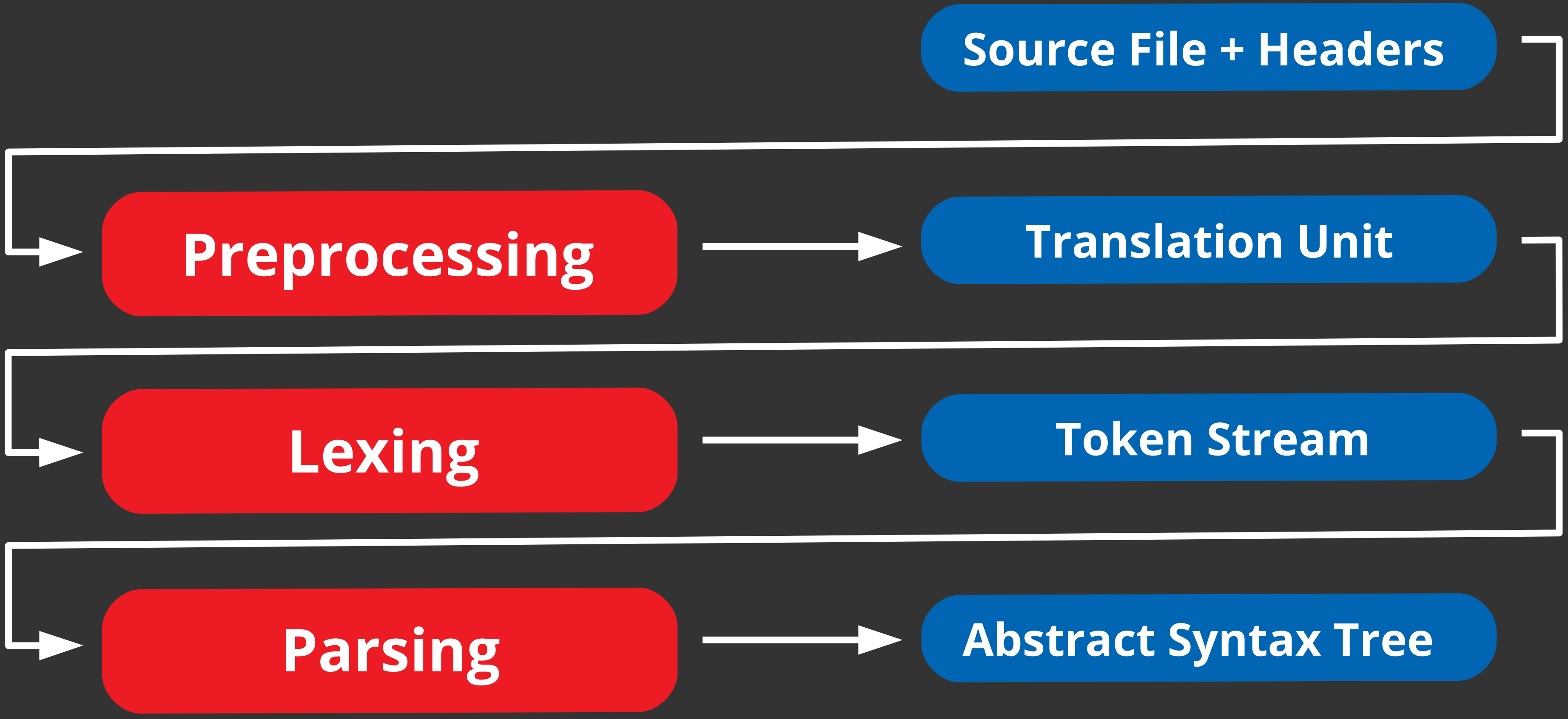
Preprocessing

Lexing

clang/lib

Parsing

```
$ clang -fsyntax-only -Xclang -ast-dump file.cpp \
    -std=c++11
```



Source File + Headers

Preprocessing

Translation Unit

Lexing

Token Stream

Parsing

Abstract Syntax Tree

What Is The Clang AST?

- A structured tree-like representation of source code constructs in a single translation unit.
- Most nodes are objects of a type which is derived from either Stmt/Expr, Decl or Type but there is no common ASTNode base class.
  - To traverse the AST requires visitation.
- Fully type resolved.
- Contains some mappings back to the source code positions.

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return 15;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

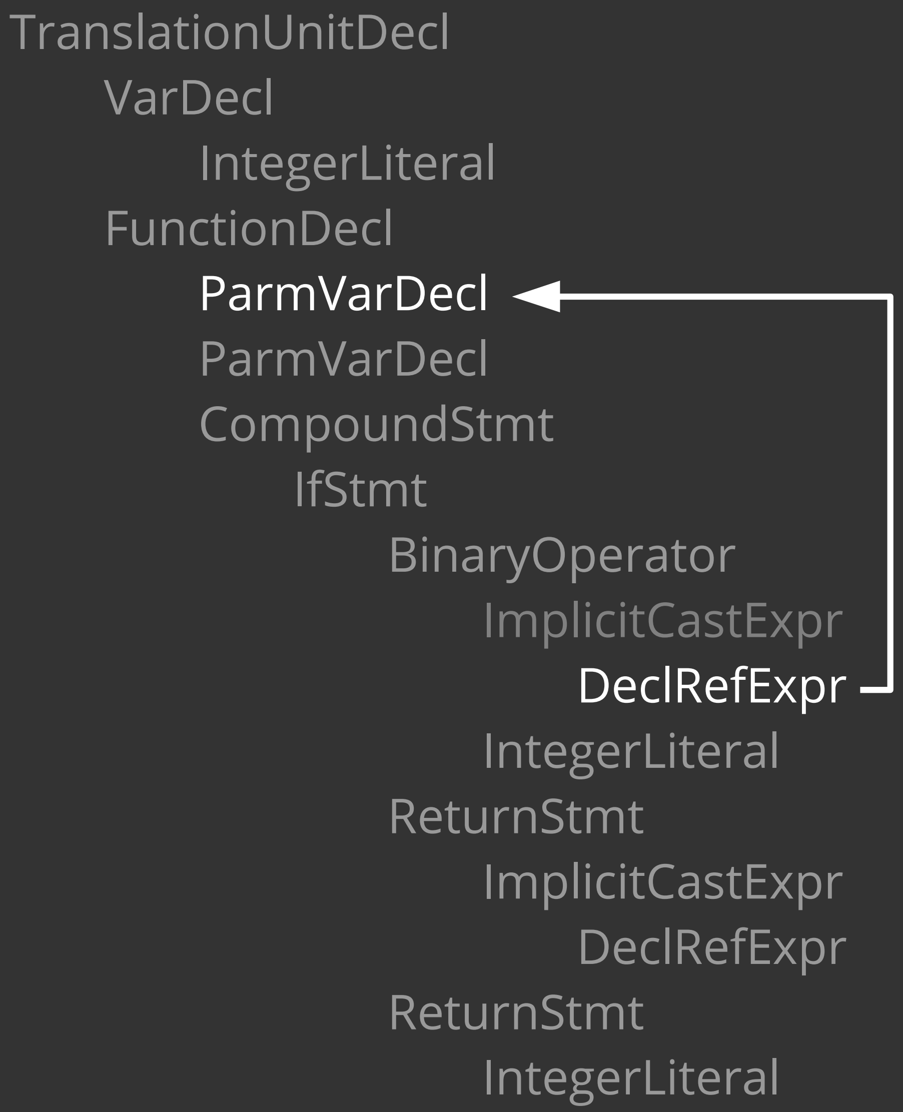
```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

What Is The Clang AST?



```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

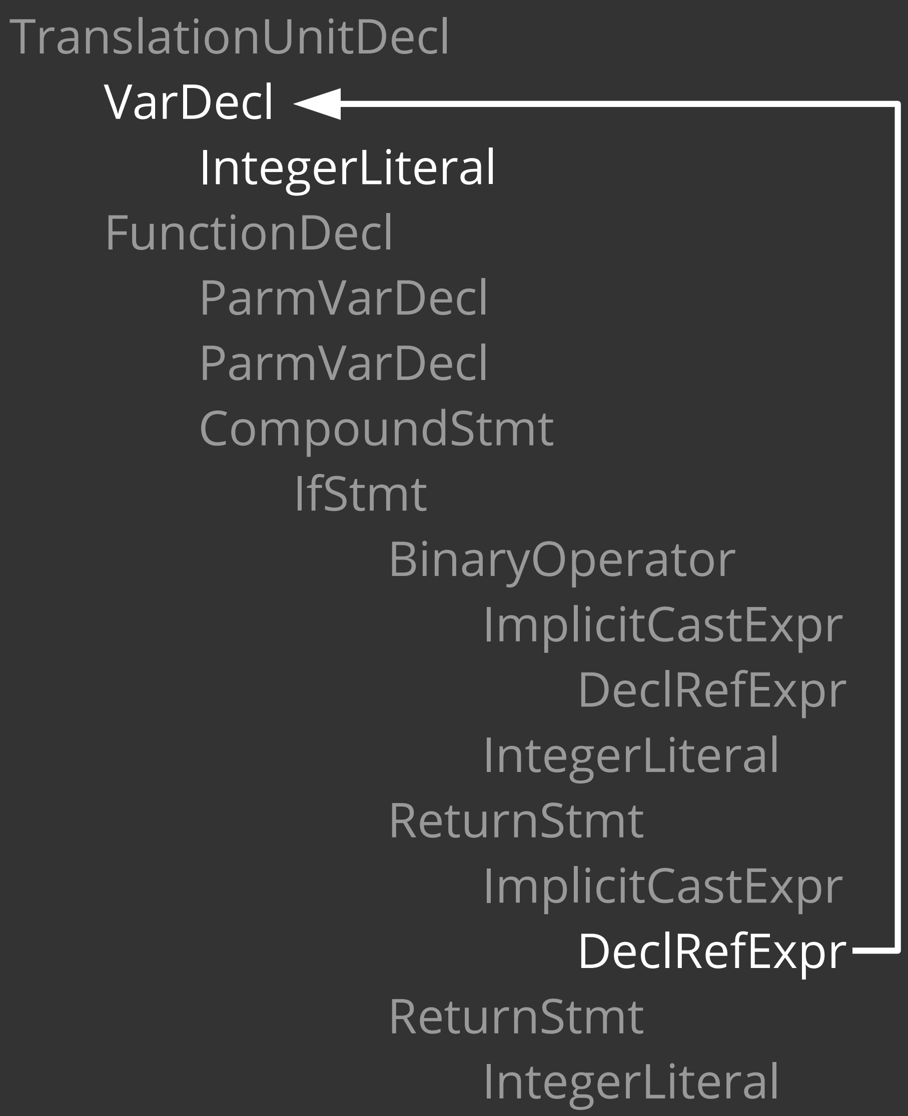
```
int x = 10;
int main(int argc,
             char const* argv[]) {
      if (argc < 5)
            return x;
      else
            return 0;
}
```

What Is The Clang AST?

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

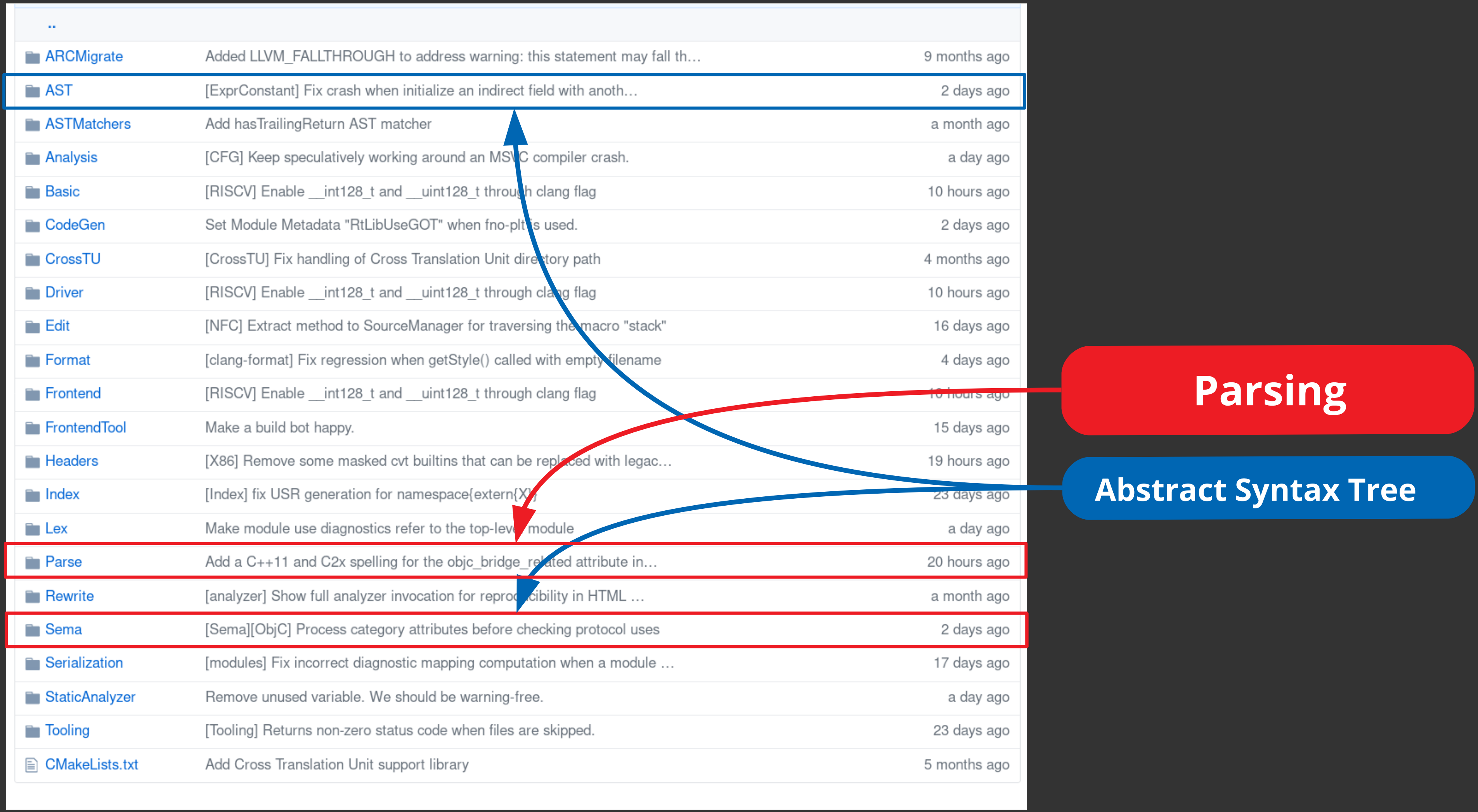
What Is The Clang AST?



```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
int x = 10;
int main(int argc,
         char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

AST, Parse and Sema Libraries



..		
ARCMigrate	Added LLVM_FALLTHROUGH to address warning: this statement may fall th…	9 months ago
AST	[ExprConstant] Fix crash when initialize an indirect field with anoth…	2 days ago
ASTMatchers	Add hasTrailingReturn AST matcher	a month ago
Analysis	[CFG] Keep speculatively working around an MSVC compiler crash.	a day ago
Basic	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
CodeGen	Set Module Metadata "RtLibUseGOT" when fno-plt is used.	2 days ago
CrossTU	[CrossTU] Fix handling of Cross Translation Unit directory path	4 months ago
Driver	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
Edit	[NFC] Extract method to SourceManager for traversing the macro "stack"	16 days ago
Format	[clang-format] Fix regression when getStyle() called with empty filename	4 days ago
Frontend	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
FrontendTool	Make a build bot happy.	15 days ago
Headers	[X86] Remove some masked cvt builtins that can be replaced with legac…	19 hours ago
Index	[Index] fix USR generation for namespace{extern{X}}	23 days ago
Lex	Make module use diagnostics refer to the top-level module	a day ago
Parse	Add a C++11 and C2x spelling for the objc_bridge_related attribute in…	20 hours ago
Rewrite	[analyzer] Show full analyzer invocation for reproducibility in HTML …	a month ago
Sema	[Sema][ObjC] Process category attributes before checking protocol uses	2 days ago
Serialization	[modules] Fix incorrect diagnostic mapping computation when a module …	17 days ago
StaticAnalyzer	Remove unused variable. We should be warning-free.	a day ago
Tooling	[Tooling] Returns non-zero status code when files are skipped.	23 days ago
CMakeLists.txt	Add Cross Translation Unit support library	5 months ago

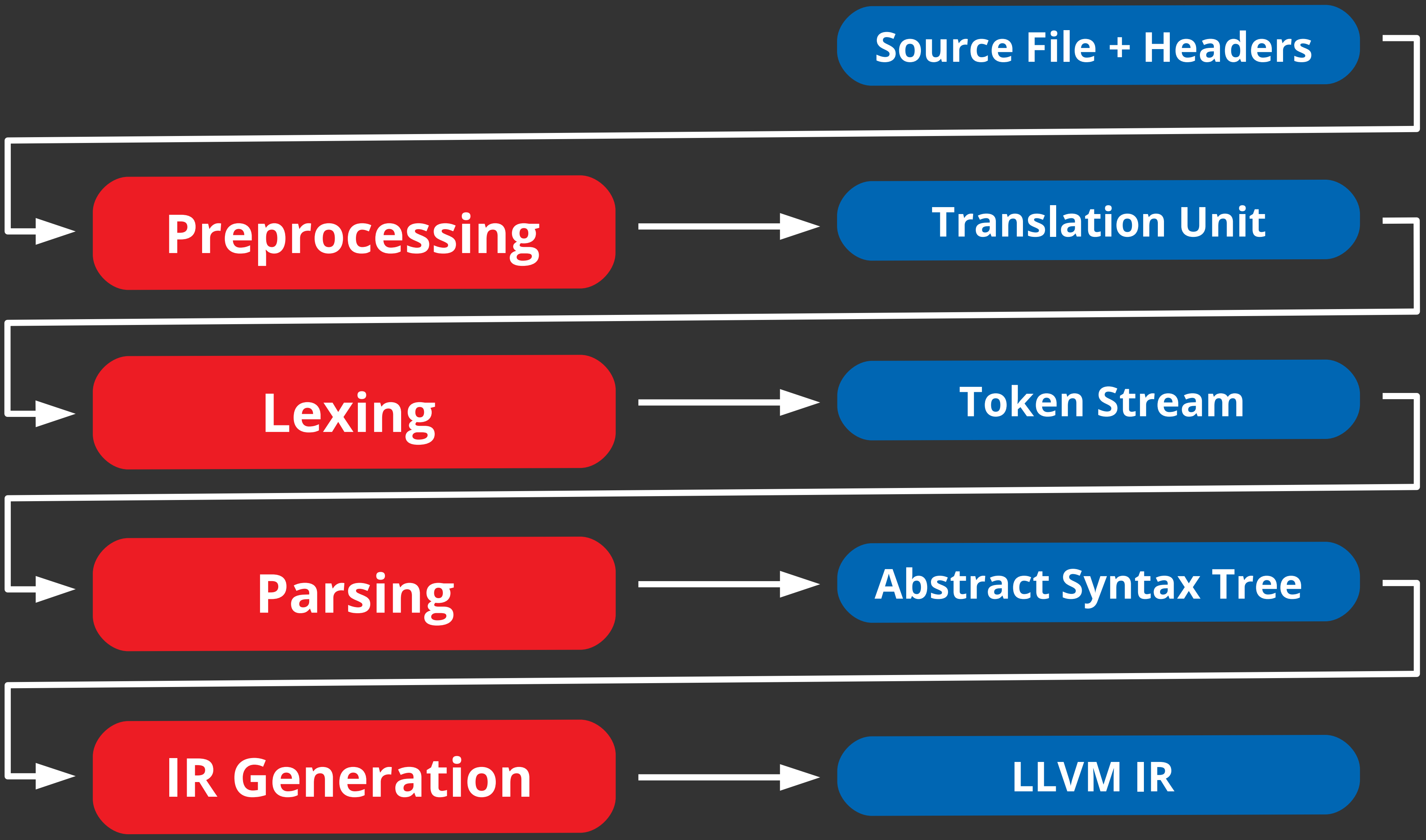
Parsing

Abstract Syntax Tree

clang/lib

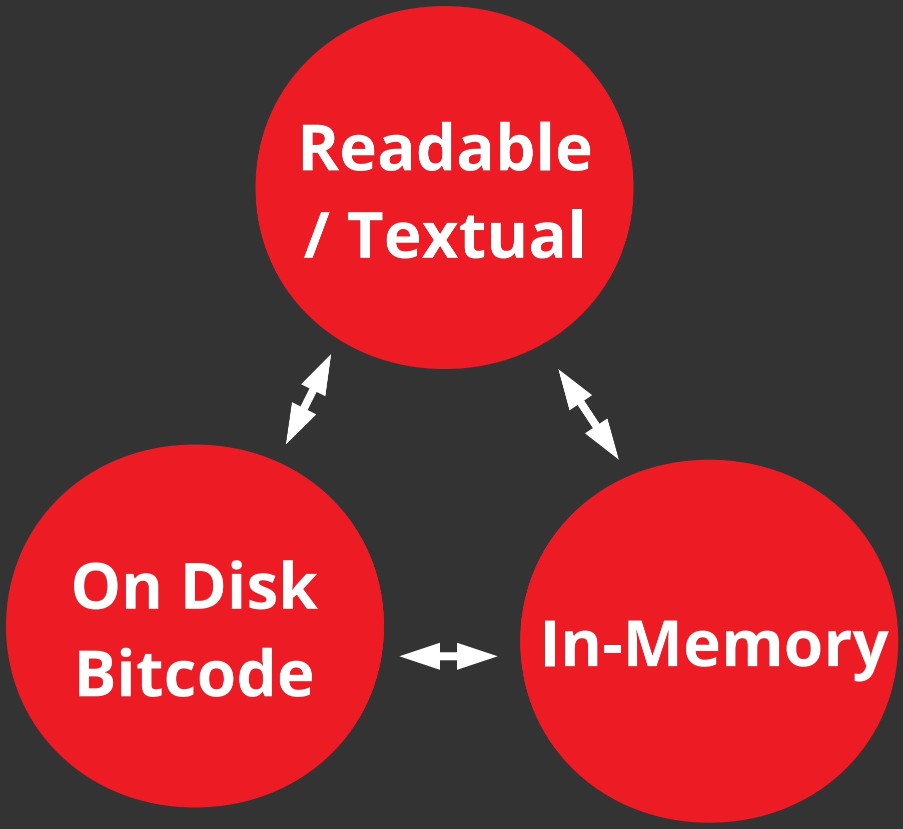
IR Generation

```
$ clang -S -emit-llvm file.cpp -std=c++11
```



What Is LLVM IR?

- Strongly Typed Language
- Infinite immutable registers
- Syntax half way between C and assembly
- Three equivalent representations



What Is LLVM IR?

- Strongly Typed Language
- Infinite immutable registers
- Syntax half way between C and assembly
- Three equivalent representations

C++ Code

```
int main() {
  auto x = 10;
  x += 10;
  return x;
}
```

LLVM IR

```
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 10, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = add nsw i32 %3, 10
  store i32 %4, i32* %2, align 4
  %5 = load i32, i32* %2, align 4
  ret i32 %5
}

attributes #0 = { noinline norecurse
nounwind optnone uwtable … }
```

What Is LLVM IR?

- Strongly Typed Language
- Infinite immutable registers
- Syntax half way between C and assembly
- Three equivalent representations

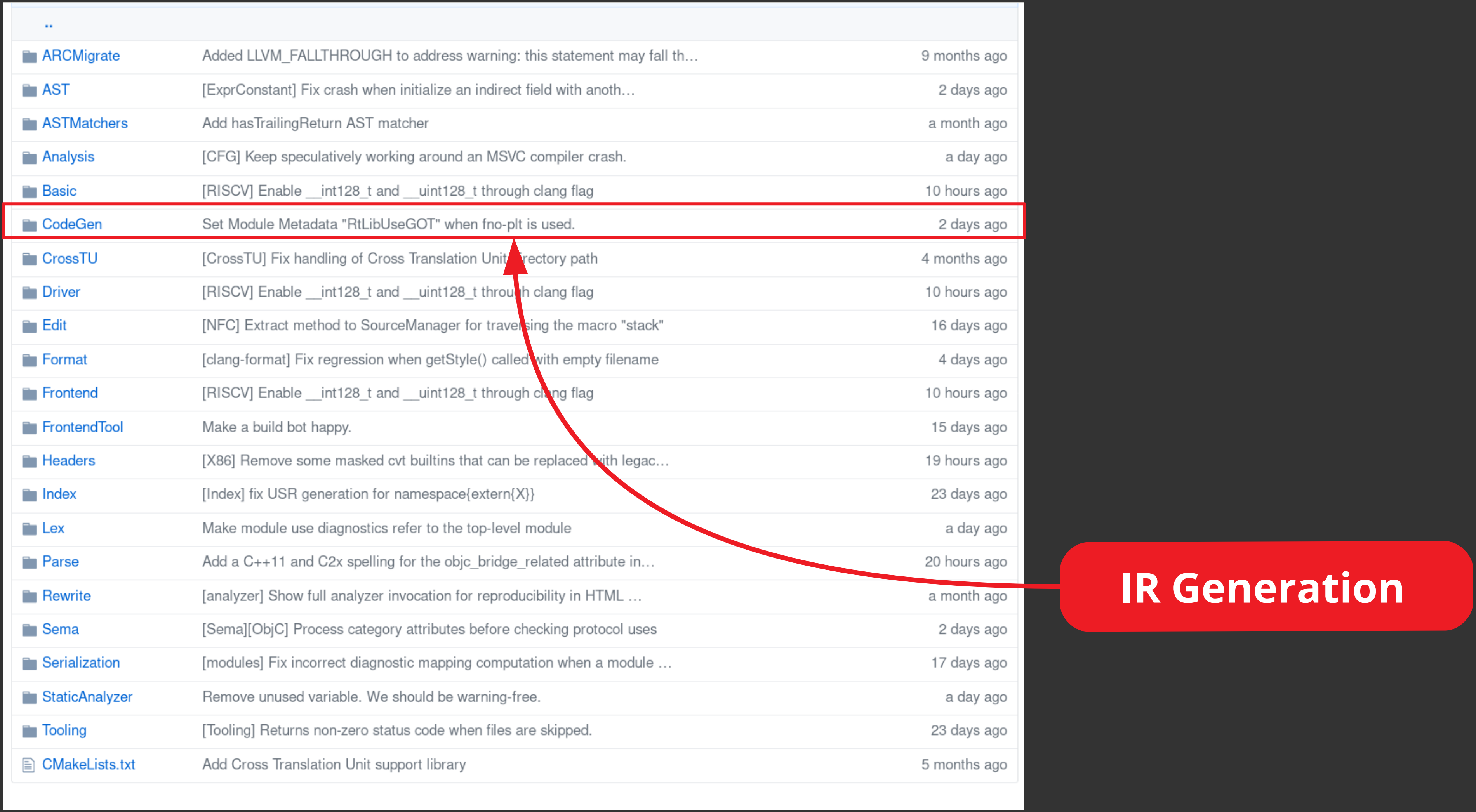
C++ Code

```
int main() {
  auto x = 10;
  x += 10;
  return x;
}
```

Optimized LLVM IR (-O3)

```
define i32 @main()
local_unnamed_addr #0 {
  ret i32 20
}

attributes #0 = { norecurse nounwind
readnone uwtable … }
```

CodeGen Library



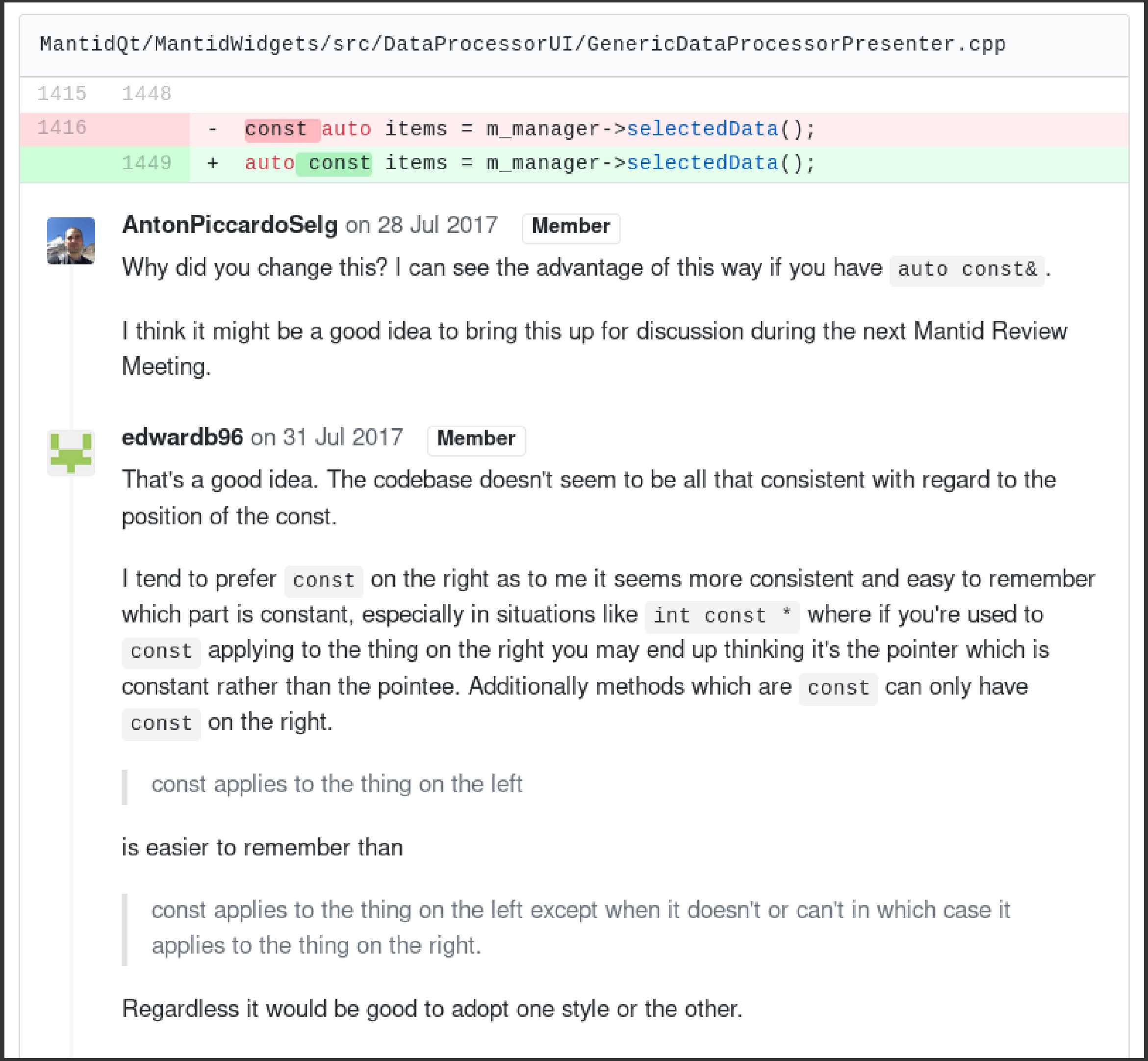
..		
ARCMigrate	Added LLVM_FALLTHROUGH to address warning: this statement may fall th…	9 months ago
AST	[ExprConstant] Fix crash when initialize an indirect field with anoth…	2 days ago
ASTMatchers	Add hasTrailingReturn AST matcher	a month ago
Analysis	[CFG] Keep speculatively working around an MSVC compiler crash.	a day ago
Basic	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
CodeGen	Set Module Metadata "RtLibUseGOT" when fno-plt is used.	2 days ago
CrossTU	[CrossTU] Fix handling of Cross Translation Unit directory path	4 months ago
Driver	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
Edit	[NFC] Extract method to SourceManager for traversing the macro "stack"	16 days ago
Format	[clang-format] Fix regression when getStyle() called with empty filename	4 days ago
Frontend	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
FrontendTool	Make a build bot happy.	15 days ago
Headers	[X86] Remove some masked cvt builtins that can be replaced with legac…	19 hours ago
Index	[Index] fix USR generation for namespace{extern{X}}	23 days ago
Lex	Make module use diagnostics refer to the top-level module	a day ago
Parse	Add a C++11 and C2x spelling for the objc_bridge_related attribute in…	20 hours ago
Rewrite	[analyzer] Show full analyzer invocation for reproducibility in HTML …	a month ago
Sema	[Sema][ObjC] Process category attributes before checking protocol uses	2 days ago
Serialization	[modules] Fix incorrect diagnostic mapping computation when a module …	17 days ago
StaticAnalyzer	Remove unused variable. We should be warning-free.	a day ago
Tooling	[Tooling] Returns non-zero status code when files are skipped.	23 days ago
CMakeLists.txt	Add Cross Translation Unit support library	5 months ago

IR Generation

clang/lib

Building A Tool

The Problem To Solve



MantidQt/MantidWidgets/src/DataProcessorUI/GenericDataProcessorPresenter.cpp

```
- const auto items = m_manager->selectedData();
+ auto const items = m_manager->selectedData();
```

AntonPiccardoSelg on 28 Jul 2017 **Member**

Why did you change this? I can see the advantage of this way if you have `auto const&`.

I think it might be a good idea to bring this up for discussion during the next Mantid Review Meeting.

edwardb96 on 31 Jul 2017 **Member**

That's a good idea. The codebase doesn't seem to be all that consistent with regard to the position of the const.

I tend to prefer `const` on the right as to me it seems more consistent and easy to remember which part is constant, especially in situations like `int const *` where if you're used to `const` applying to the thing on the right you may end up thinking it's the pointer which is constant rather than the pointee. Additionally methods which are `const` can only have `const` on the right.

> const applies to the thing on the left

is easier to remember than

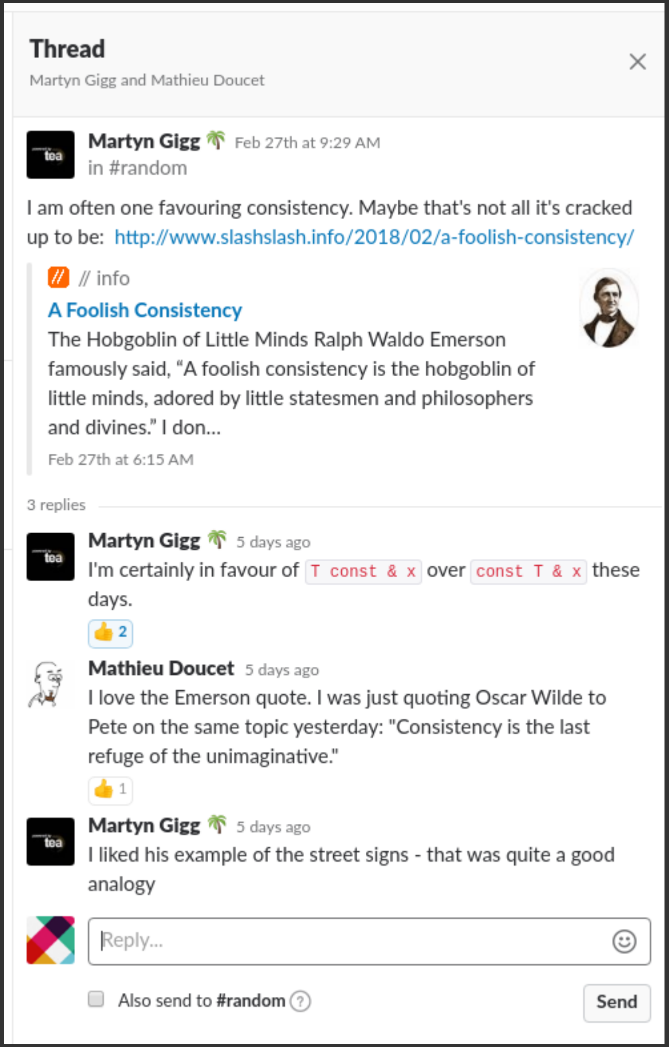
> const applies to the thing on the left except when it doesn't or can't in which case it applies to the thing on the right.

Regardless it would be good to adopt one style or the other.

https://github.com/mantidproject/mantid/pull/20082

And Then More Recently

(by sheer coincidence)



Thread

Martyn Gigg and Mathieu Doucet

Martyn Gigg 🌴 Feb 27th at 9:29 AM
in #random

I am often one favouring consistency. Maybe that's not all it's cracked up to be: http://www.slashslash.info/2018/02/a-foolish-consistency/

// info

A Foolish Consistency

The Hobgoblin of Little Minds Ralph Waldo Emerson famously said, "A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines." I don...

Feb 27th at 6:15 AM

3 replies

Martyn Gigg 🌴 5 days ago
I'm certainly in favour of `T const & x` over `const T & x` these days.

👍 2

Mathieu Doucet 5 days ago
I love the Emerson quote. I was just quoting Oscar Wilde to Pete on the same topic yesterday: "Consistency is the last refuge of the unimaginative."

👍 1

Martyn Gigg 🌴 5 days ago
I liked his example of the street signs - that was quite a good analogy

https://mantid.slack.com/archives/C02J4MMGJ/p1519723759000316

Motivation

Personal

- Conceptually simple but non-trivial transformation with no semantic changes. Therefore it would serve as a good introduction to some of the Clang tooling facilities.
- When you work using one style all day and then go home and work on projects which use the other, you end up with a mix of both.

Why would you use it?

- Having `const` on the right simplifies the rule for what the `const` applies to in cases like `const int* const` where it is not always obvious.
- Whichever style is preferred, it is more confusing if neither is applied or enforced consistently.

LibClang

- High Level Library with a Stable API
- Sparsely documented Python bindings
- Popular choice for editors.
- Used to implement many features of Xcode.
- C API :(

Clang C++ Library APIs

- Less stable than LibClang but more powerful.
- LibTooling can be used as a framework for building command line tools
  - Handles loading of the compilation database.
  - Handles parsing command line arguments for both the compiler and your tool.
  - Handles setting up and invoking the parser.

Clang Tidy Plugin

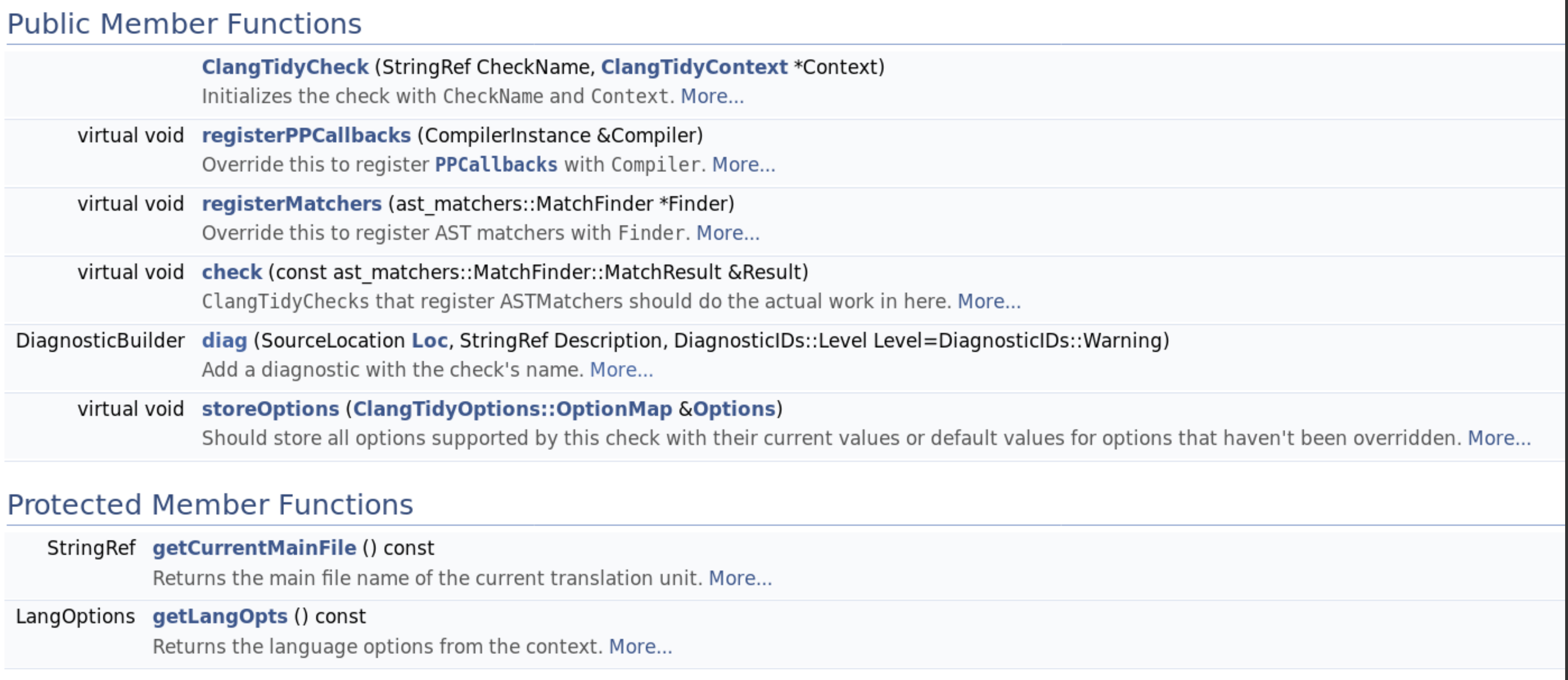
- Clang Tidy provides a uniform interface to many different static-analysis based linter-style checks.
- Designed to be combined with the C++ Library APIs.
- Eliminates boilerplate code associated with writing each check as it’s own tool with the C++ Library APIs.
- Provides a framework for extension by adding new checks.

Clang Tidy Readability - Demo

```
$ clang-tidy -checks=”*” -list-checks
```

```
$ clang-tidy -checks=”-*,readability-identifier-naming” \
              file.cpp -- -std=c++11
```

Clang Tidy Plugin



Public Member Functions

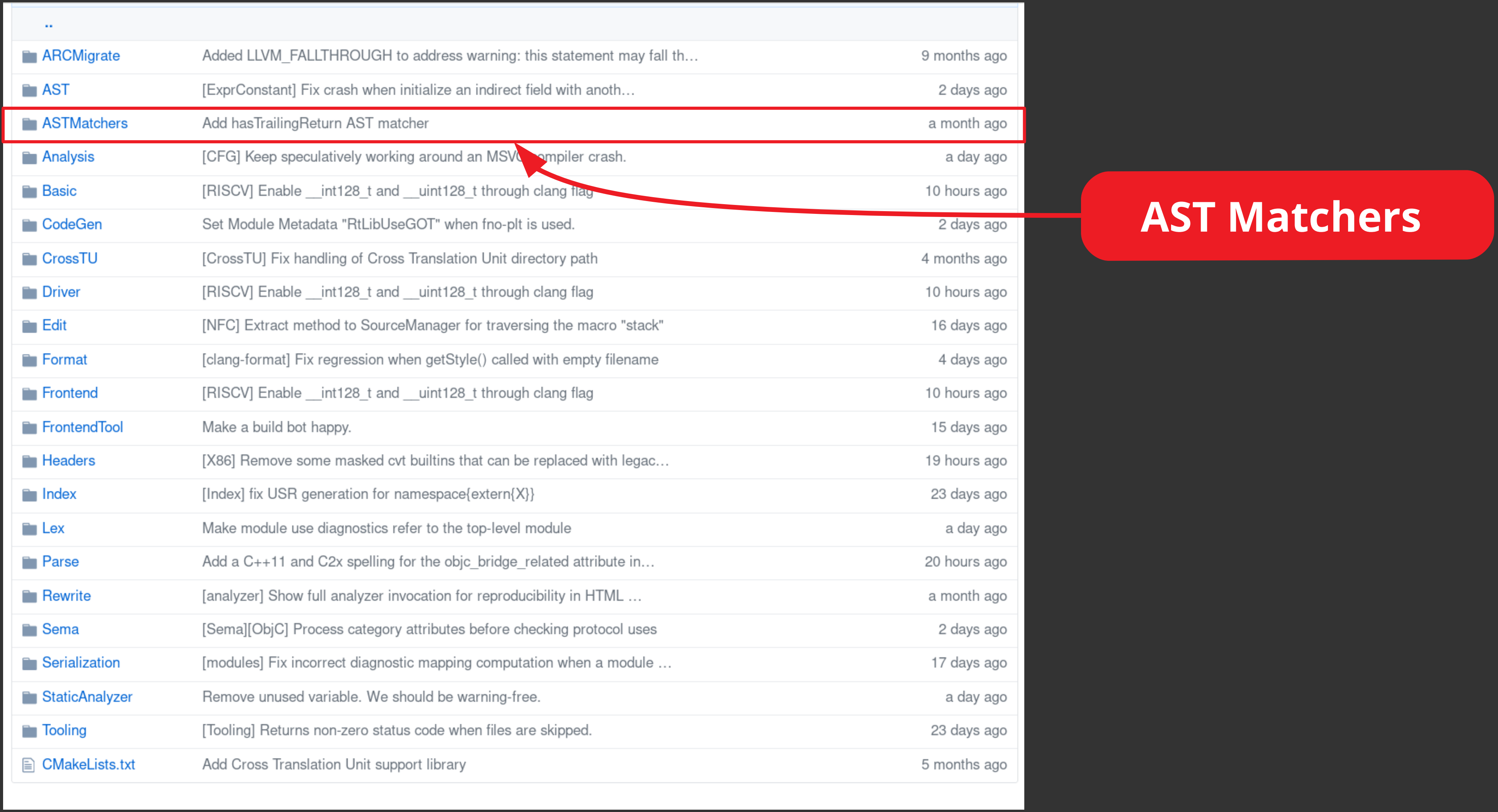
	ClangTidyCheck (StringRef CheckName, **ClangTidyContext** *Context) Initializes the check with `CheckName` and `Context`. More...
virtual void	**registerPPCallbacks** (CompilerInstance &Compiler) Override this to register **`PPCallbacks`** with `Compiler`. More...
virtual void	**registerMatchers** (ast_matchers::MatchFinder *Finder) Override this to register AST matchers with `Finder`. More...
virtual void	**check** (const ast_matchers::MatchFinder::MatchResult &Result) `ClangTidyChecks` that register ASTMatchers should do the actual work in here. More...
DiagnosticBuilder	**diag** (SourceLocation **Loc**, StringRef Description, DiagnosticIDs::Level Level=DiagnosticIDs::Warning) Add a diagnostic with the check's name. More...
virtual void	**storeOptions** (**ClangTidyOptions::OptionMap** &**Options**) Should store all options supported by this check with their current values or default values for options that haven't been overridden. More...

Protected Member Functions

StringRef	**getCurrentMainFile** () const Returns the main file name of the current translation unit. More...
LangOptions	**getLangOpts** () const Returns the language options from the context. More...

https://clang.llvm.org/extra/doxygen/classclang_1_1tidy_1_1ClangTidyCheck.html

AST Matchers Library



..		
ARCMigrate	Added LLVM_FALLTHROUGH to address warning: this statement may fall th…	9 months ago
AST	[ExprConstant] Fix crash when initialize an indirect field with anoth…	2 days ago
ASTMatchers	Add hasTrailingReturn AST matcher	a month ago
Analysis	[CFG] Keep speculatively working around an MSVC compiler crash.	a day ago
Basic	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
CodeGen	Set Module Metadata "RtLibUseGOT" when fno-plt is used.	2 days ago
CrossTU	[CrossTU] Fix handling of Cross Translation Unit directory path	4 months ago
Driver	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
Edit	[NFC] Extract method to SourceManager for traversing the macro "stack"	16 days ago
Format	[clang-format] Fix regression when getStyle() called with empty filename	4 days ago
Frontend	[RISCV] Enable __int128_t and __uint128_t through clang flag	10 hours ago
FrontendTool	Make a build bot happy.	15 days ago
Headers	[X86] Remove some masked cvt builtins that can be replaced with legac…	19 hours ago
Index	[Index] fix USR generation for namespace{extern{X}}	23 days ago
Lex	Make module use diagnostics refer to the top-level module	a day ago
Parse	Add a C++11 and C2x spelling for the objc_bridge_related attribute in…	20 hours ago
Rewrite	[analyzer] Show full analyzer invocation for reproducibility in HTML …	a month ago
Sema	[Sema][ObjC] Process category attributes before checking protocol uses	2 days ago
Serialization	[modules] Fix incorrect diagnostic mapping computation when a module …	17 days ago
StaticAnalyzer	Remove unused variable. We should be warning-free.	a day ago
Tooling	[Tooling] Returns non-zero status code when files are skipped.	23 days ago
CMakeLists.txt	Add Cross Translation Unit support library	5 months ago

clang/lib

AST Matchers

- Embeded DSL for describing AST Queries.
- Queries are formed by composition of 3 types of predicates over an AST Node and its descendants
  - Node Matchers – Filter based on the type.
  - Narrowing Matchers – Filter based on the value.
  - Traversal Matchers – Filter based on the existence of descendant/connected nodes.
- The final query will be a predicate over some top level AST node type. e.g. Stmt, Decl, Type.

AST Matchers - Example

- We want to find all return statements returning an integer literal zero and their parent function.

```
auto x = 10;
int main(int argc, char const* argv[]) {
    if (argc < 5)
        return x;
    else
        return 0;
}
```

AST Matchers - Example

```
$ clang-query simple-if-then-else.cpp -- -std=c++11
```

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
functionDecl(anything())
.bind(“root”)
```

AST Matchers - Example

```
$ clang-query simple-if-then-else.cpp -- -std=c++11
```

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
functionDecl(
    forEachDescendant(
        stmt(anything())
        .bind(“stmt”)))
.bind(“root”)
```

AST Matchers - Example

```
$ clang-query simple-if-then-else.cpp -- -std=c++11
```

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
functionDecl(
    forEachDescendant(
        returnStmt(anything())
        .bind(“stmt”)))
.bind(“root”)
```

AST Matchers - Example

```
$ clang-query simple-if-then-else.cpp -- -std=c++11
```

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
functionDecl(
    forEachDescendant(
        returnStmt(
            hasReturnValue(
                integerLiteral(
                    anything())))
        .bind(“stmt”)))
.bind(“root”)
```

AST Matchers - Example

```
$ clang-query simple-if-then-else.cpp -- -std=c++11
```

```
TranslationUnitDecl
    VarDecl
        IntegerLiteral
    FunctionDecl
        ParmVarDecl
        ParmVarDecl
        CompoundStmt
            IfStmt
                BinaryOperator
                    ImplicitCastExpr
                        DeclRefExpr
                    IntegerLiteral
                ReturnStmt
                    ImplicitCastExpr
                        DeclRefExpr
                ReturnStmt
                    IntegerLiteral
```

```
functionDecl(
    forEachDescendant(
        returnStmt(
            hasReturnValue(
                integerLiteral(
                    equals(0))))
        .bind(“stmt”)))
.bind(“root”)
```

A Simple Clang Tidy Check - Example

```
#include “../ClangTidy.h”
#include “llvm/ADT/StringRef.h”

namespace clang { namespace tidy { namespace readability {

class GenericVariableNamesCheck : public ClangTidyCheck {
public:
  GenericVariableNamesCheck(StringRef Name, ClangTidyContext *Context)
      : ClangTidyCheck(Name, Context) {}
  void registerMatchers(ast_matchers::MatchFinder *Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};

void GenericVariableNamesCheck::registerMatchers(MatchFinder *Finder) {
  Finder->addMatcher(varDecl().bind("var"), this);
}

void GenericVariableNamesCheck::check(const MatchFinder::MatchResult &Result) {
  if (auto* MatchedVar = Result.Nodes.getNodeAs<VarDecl>("var")) {
    auto name = MatchedVar->getName();
    if (name == "data")
      diag(MatchedVar->getLocation(),
           "Data too generic as a variable name, all variables hold data.")
        << MatchedVar;
  }
}

}}}
```

Our First Integration Test

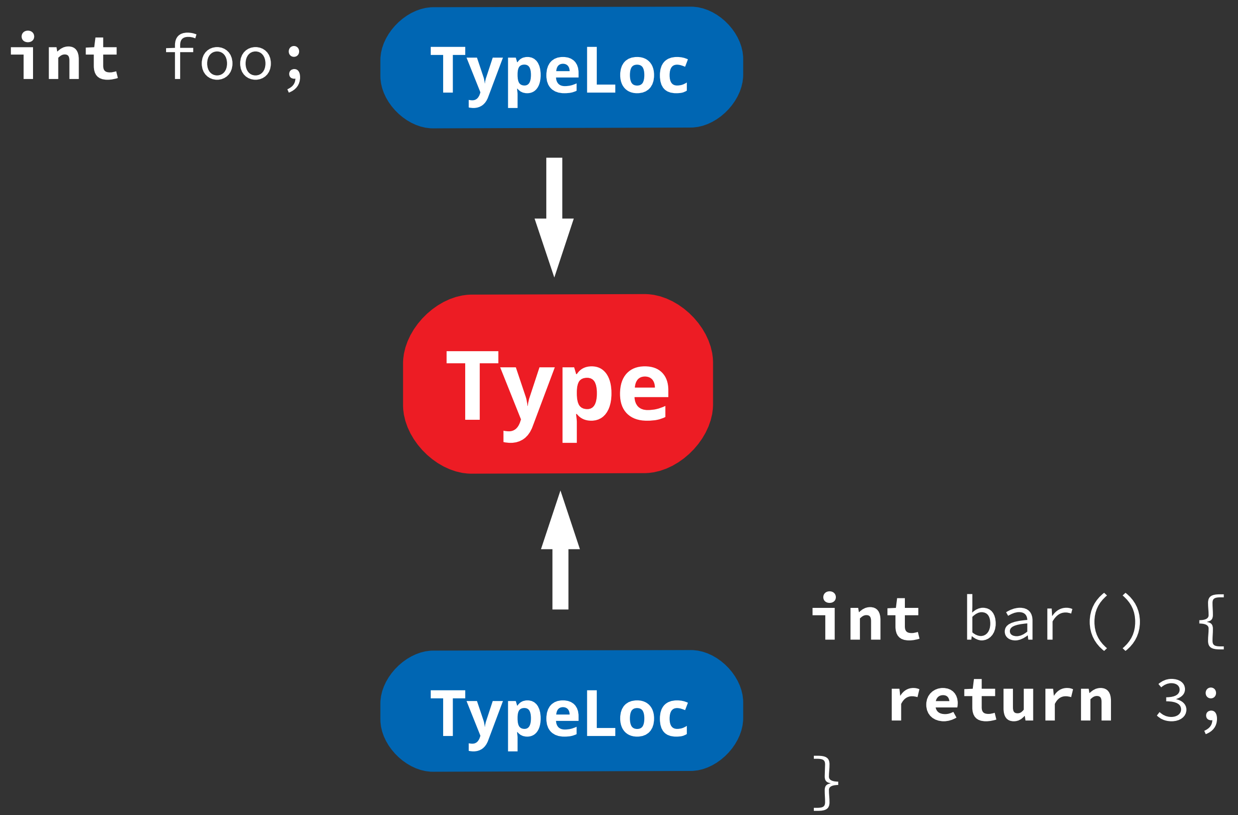
```
// RUN: %check_clang_tidy %s voxel-const-position-check %t \
// RUN:   -config="{
    CheckOptions: [{
      key: "voxel-const-position-check.ConstPosition",
      value: "Left"}]}" -- -std=c++11

int main() {
  // CHECK-MESSAGES: :[[@LINE+1]]:7: warning: misplaced const.
      [voxel-const-position-check]
  int const a = 5;
  // CHECK-FIXES: {{^}}  const int a = 5;{{$}}

  // CHECK-MESSAGES: :[[@LINE+1]]:8: warning: misplaced const.
      [voxel-const-position-check]
  auto const b = 14.5f;
  // CHECK-FIXES: {{^}}  const auto b = 14.5f;{{$}}
  return 0;
}
```

Writing the Matcher – Attempt 1

```
typeLoc(
  loc(isConstQualified())).bind(“loc”)
```

- Matches all occurrences of a type which is **const** qualified.

What is a TypeLoc?



```
int foo;
```

TypeLoc

Type

TypeLoc

```
int bar() {
  return 3;
}
```

The representation of a type’s occurrence location is kept separate from the type itself. The class which stores a type’s occurrence location information is **TypeLoc**.

`SourceLocations` from `TypeLoc`

- `.getLocEnd()` and `getLocBegin()` return `SourceLocations`.
- But `SourceLocations` returned by these methods are only accurate to the nearest token. So for types consisting of only one token such as `int`…

  ```
  t.getLocBegin() == t.getLocEnd()
  ```
- So `EndCharLoc` was born…

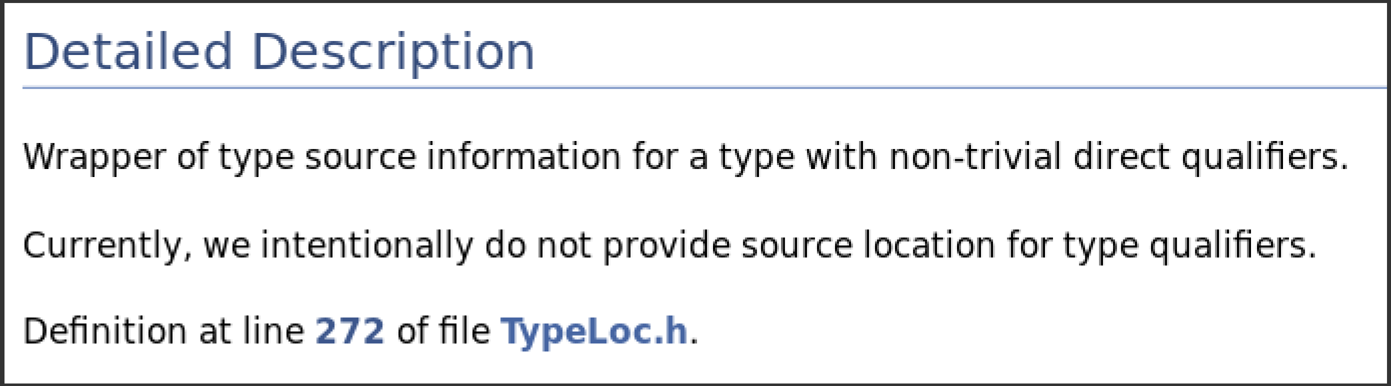
The First Setback

```
const auto x = 0;
```

and

```
auto const x = 0;
```

Have equivalent AST representations



Detailed Description

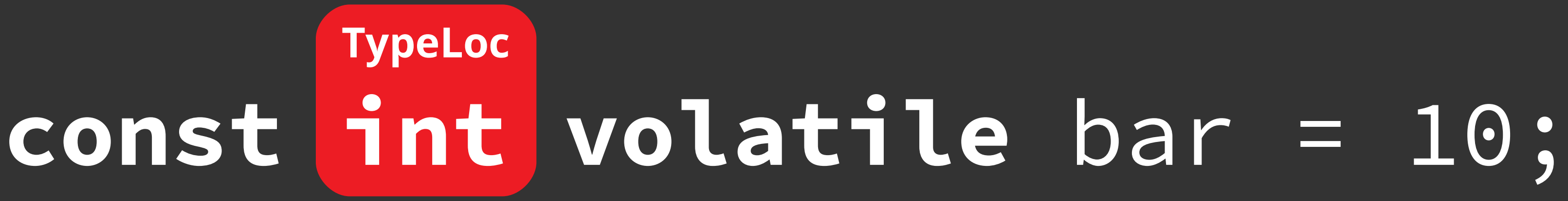
Wrapper of type source information for a type with non-trivial direct qualifiers.

Currently, we intentionally do not provide source location for type qualifiers.

Definition at line **272** of file **TypeLoc.h**.

https://clang.llvm.org/doxygen/classclang_1_1QualifiedTypeLoc.html

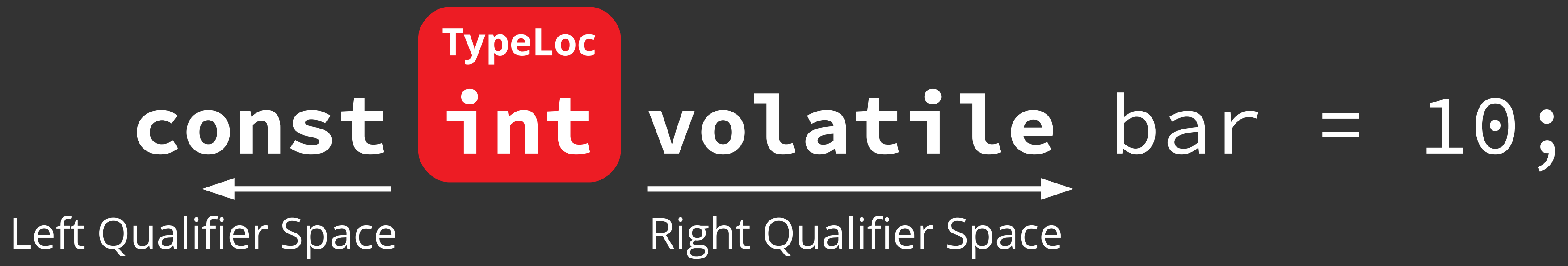
Furthermore...



```
const int volatile bar = 10;
```

The locations given by TypeLoc exclude the surrounding qualifiers.

The New Plan



TypeLoc

```
const int volatile bar = 10;
```

Left Qualifier Space

Right Qualifier Space

Starting at the beginning and the end of the **TypeLoc** we walk away from the type until we find **const** or reach the edge of where **const** could be.

Creating a `clang::Lexer` Object

```
std::pair<FileID, unsigned> DecomposedLoc =
  SourceManager.getDecomposedLoc(Start);
auto File = DecomposedLoc.first;
auto OffsetInFile = DecomposedLoc.second;
StringRef FileContent = SourceManager.getBufferData(File);
const char *StartPoint =
  FileContent.data() + OffsetInFile;

Lexer RawLexer(Sources.getLocForStartOfFile(File),
               LangOpts,
               FileContent.begin(),
               StartPoint,
               FileContent.end());
```

Using a clang::Lexer

```
template <typename Action>
void ConstPositionCheck::walkRight(Action shouldContinue) {
  auto Tok = Token();
  while (RawLexer.LexFromRawLexer(Tok)) {
    if (Tok.is(tok::raw_identifier))
      withIdentifierInfo(Tok);
    if (!shouldContinue(Tok))
      break;
  }
}

Token ConstPositionCheck::withIdentifierInfo(Token& Tok) {
  auto Identifier = StringRef(
    getSourceManager().getCharacterData(Tok.getLocation()),
               Tok.getLength());
  auto& IdentifierInfo = getASTContext().Idents.get(Identifier);
  Tok.setIdentifierInfo(&IdentifierInfo);
  Tok.setKind(IdentifierInfo.getTokenID());
  return Tok;
}
```

The Lexer API

- The clang Lexer allows us to start at some position and then walk over the tokens until we reach the end of the file.
- The API for this is somewhat more complex than it needs to be for our purposes.
- Lexing over the tokens in reverse until the beginning of the file isn't directly supported.
    - To do this simply we are required to supply some upper bound on the distance we will lex back..
    - If we guess too high then we allocate more memory and lex more tokens than necessary
    - If we guess too low then the behaviour of our program is incorrect.
- Introducing `SimpleLexer`.

Knowing When To Stop

- There are a number of different tokens which can legally appear between the edge of a type identifier and a legitimate **const** qualifier **constexpr**, **volatile**, **restrict** (C-only), **static** and vendor specific extensions or attributes. e.g.

```
int volatile const y = 0;
```

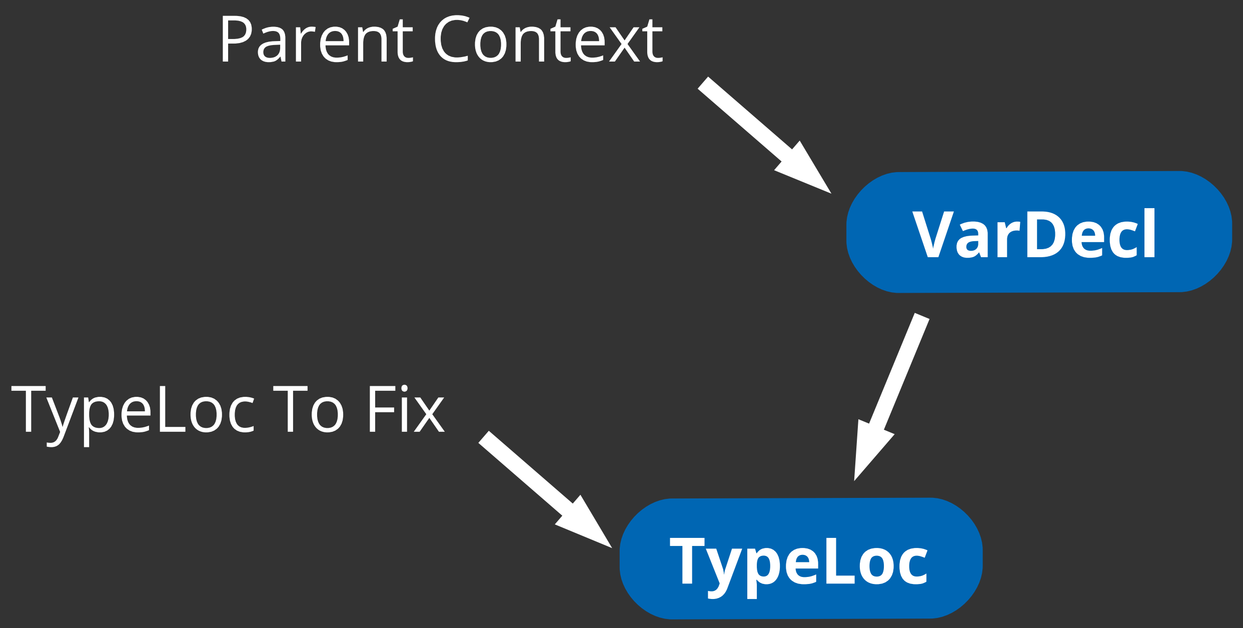
- Equally there are also a number of tokens which signal that the search should be terminated, continuing beyond them may cause the program to ‘steal’ the **const** qualifier. e.g.

```
int* const x = nullptr; → const int* x = nullptr;
```

↑ Should have stopped here!

=> We need to know more about the context of the type we’re finding qualifiers for.

Variable Context



Parent Context

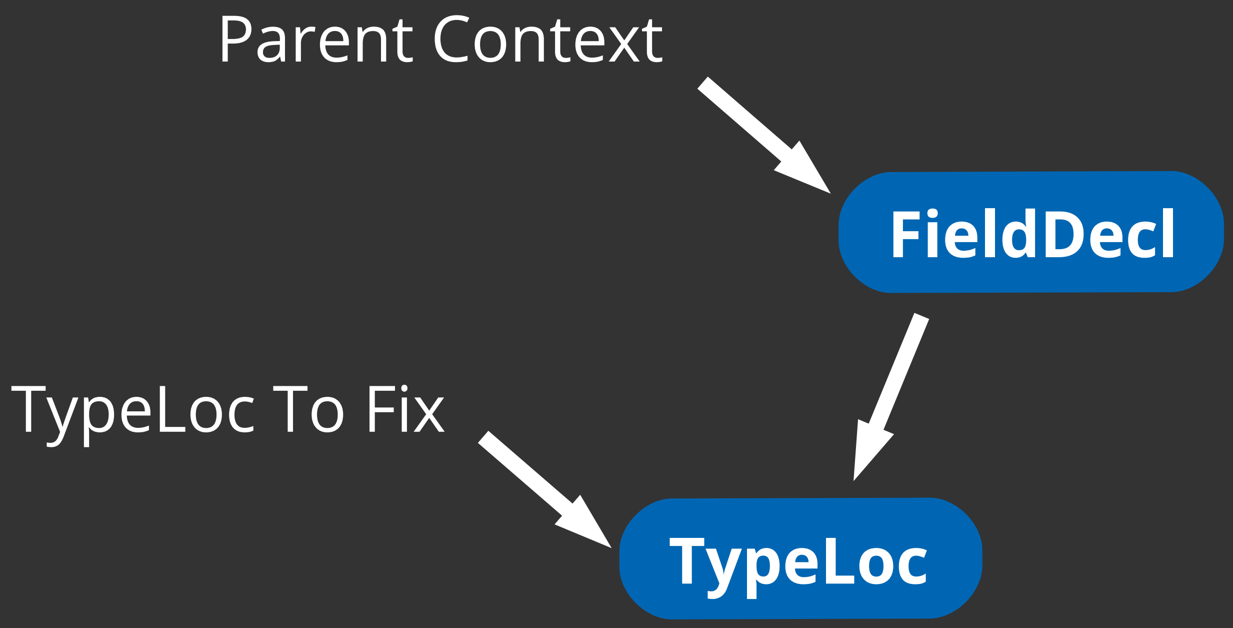
VarDecl

TypeLoc To Fix

TypeLoc

```
const int volatile bar = 10;
```

Field Context



Parent Context

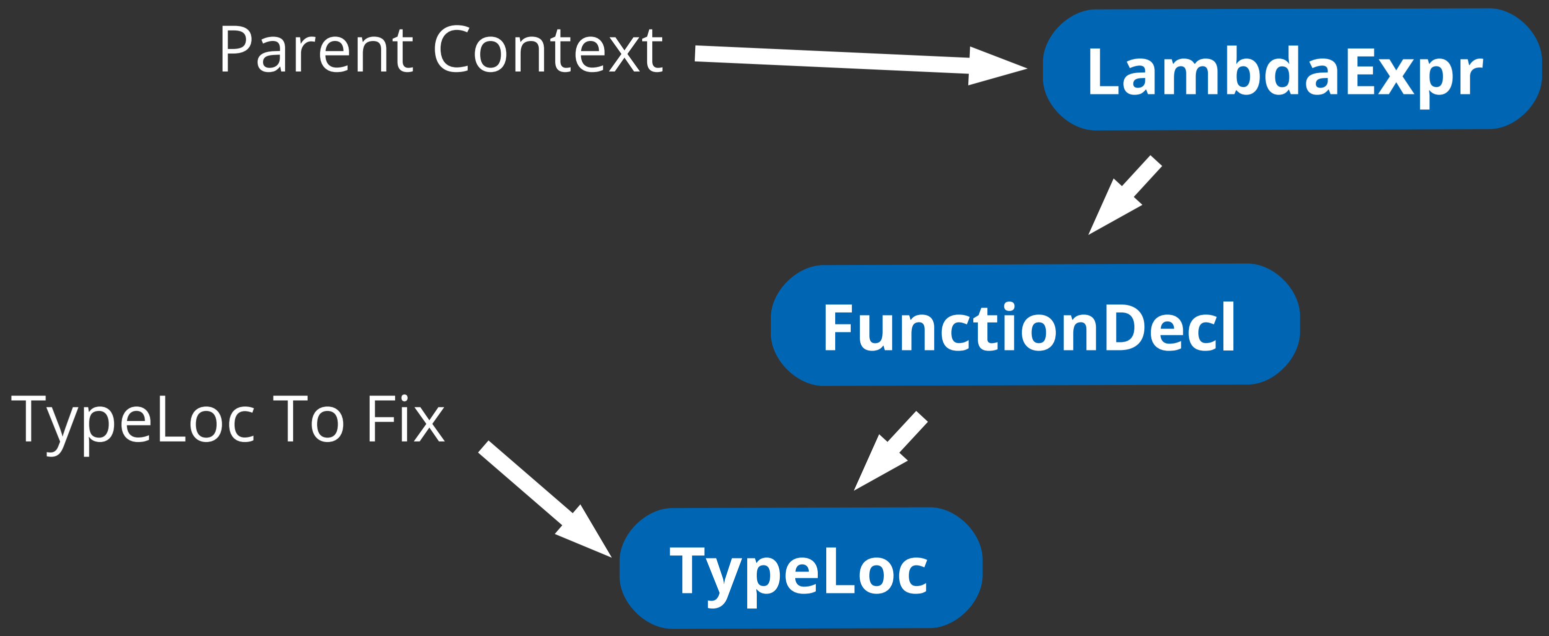
FieldDecl

TypeLoc To Fix

TypeLoc

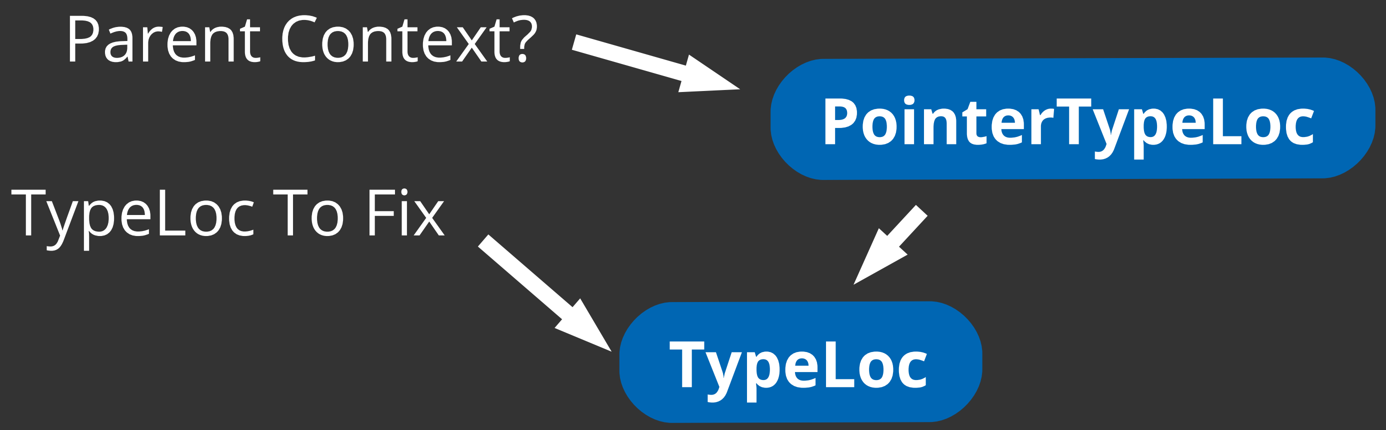
```
class X {
  const int bar;
};
```

Lambda Return Type Context



```
[](int x) -> const int {
  return x + 2;
};
```

How About Nested Types?



Parent Context?

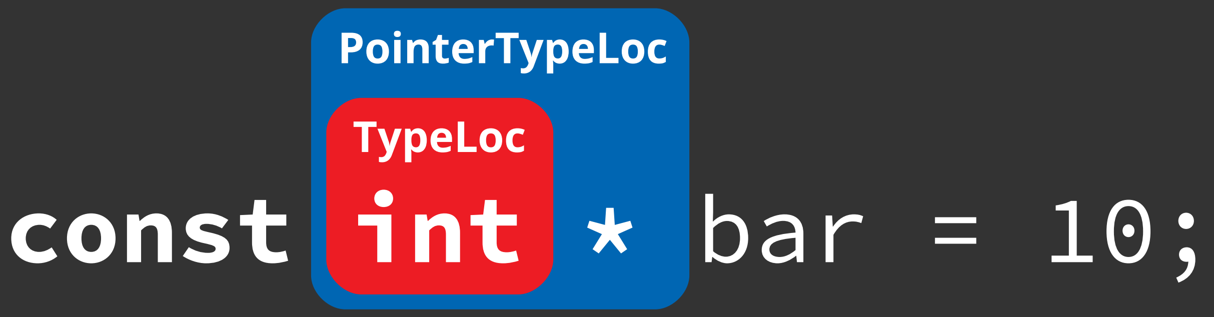
PointerTypeLoc

TypeLoc To Fix

TypeLoc

```
const int* bar = 10;
```

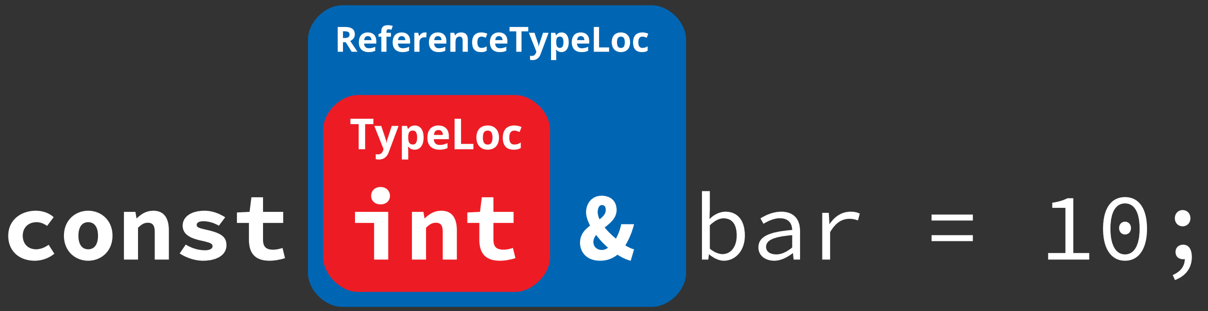
What About Nested Types?



```
const int * bar = 10;
```

The locations given by `PointerTypeLoc` exclude the left qualifiers of the pointee type.

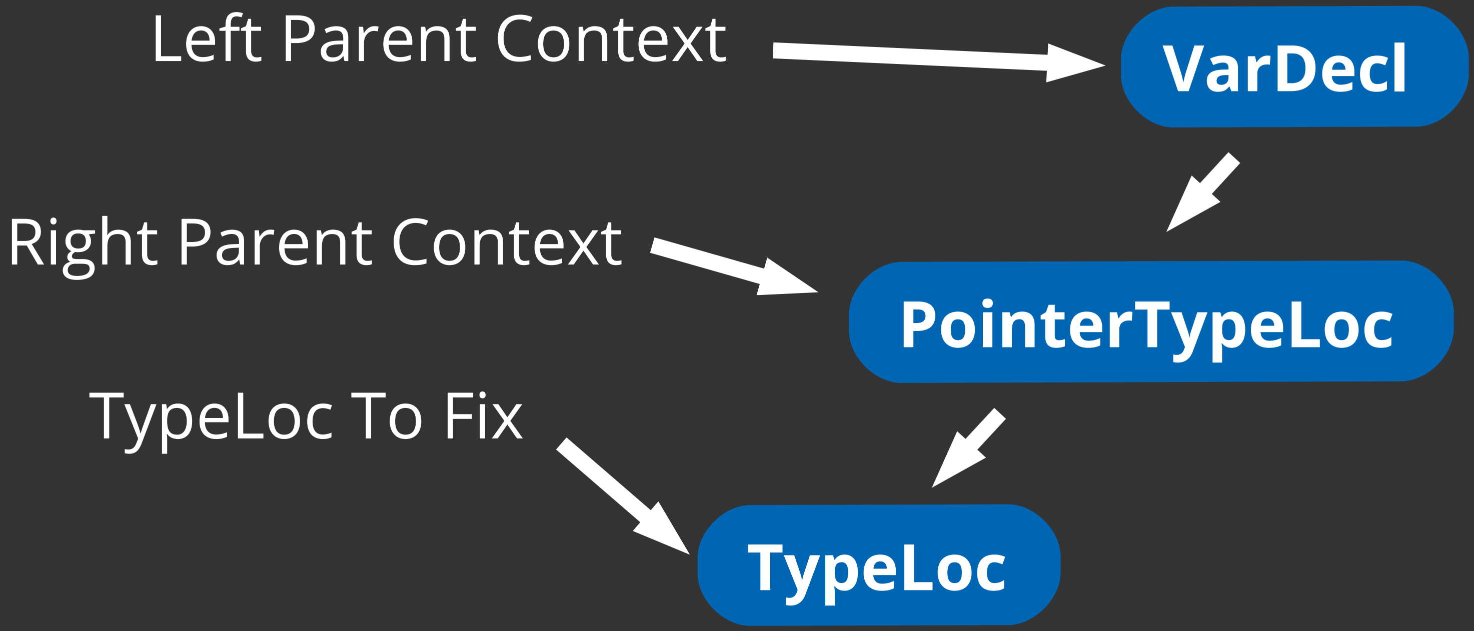
What About Nested Types?



```
const int & bar = 10;
```

The same is true for `ReferenceTypeLoc`, `ArrayTypeLoc` and others.

What About Nested Types?



```
const int* y = 10;
```

The `PointerTypeLoc` is still useful as it gives the location of the * which we can use as a right bound.

The Higher Level Logic

```
auto MaybeNodes = ExtractUsefulNodes(BoundNodes);
if (MaybeNodes.hasValue()) {
  auto const &Nodes = MaybeNodes.getValue();
  auto BadConst = FindBadConst(Nodes);
  if (BadConst.hasValue()) {
    auto const &TypeLocation = TypeLocToFix(Nodes);
    auto GoodConstLocation = CorrectConstPosition(TypeLocation);
    ApplyDiagnostic(BadConst.getValue(),
                    GoodConstLocation,
                    TypeLocation);
  }
  return true;
} else {
  return false;
}
```

The Higher Level Logic

- For each ‘context’ we need:
  - An Extractor to convert a `MatchResult` into our ‘context’ object.
  - A `FindLeftConst()` and `FindRightConst()` function which uses source locations obtained from the ‘context’ object and the lexer to find the ‘bad’ **`const`** token if it exists.
- We also need a correction function which finds the correct location for **`const`** given the Left/Rightness, and the `TypeLoc` to fix.
- Finally we need to be able to use this information to generate a clang-tidy diagnostic.

Contexts

A ‘context’ class is a class containing a `TypeLoc` and one or more other nodes extracted from the AST which together are sufficient for finding the `TypeLoc`’s corresponding **`const`** qualifier.

Variable/Parameter
Field
Using
Typedef
Function Return Type
Lambda Return Type
Function Template
Class Template
Non-Type Template Parameter
C Style Cast
C++ Named Casts
New Expression

×

Primative
Pointee
Referee
Array

=

48

classes

Templatized Sub-contexts

- We use templates to make composable ‘subcontexts’ which contain useful information but are not sufficient to determine both the left and right bounds of the search.
- This significantly reduces the amount of code we have to write and when combined with the searcher, it allows us to express more general rules.

```
template <typename PointeeContext, typename LinearSearcher>
llvm::Optional<Token>
FindRightConst(LinearSearcher const &LinearSearchForConst,
               PointerNodes<PointeeContext> const &Nodes) {
  return LinearSearchForConst(
      RightQualifierSpace(TypeLocToFix(Nodes.Pointee),
                          Nodes.PointerLoc));
}

LeftToRightSourceTraversal<RightOfNode<TypeLoc>, SourceLocation>
RightQualifierSpace(TypeLoc PointeeLoc,
                    PointerTypeLoc const &Ancestor) {
  return LeftToRight(RightOf(PointeeLoc), Ancestor.getStarLoc());
}
```

Currently Implemented...

	Primative	Pointee*	Referee	Array
Variable/Parameter	✓	✓	✓	✓
Field	✓	✓	✓	✓
Using	✓	✓	✓	
Typedef	✓	✓	✓	
Function Return Type	✓	✓	✓	
Lambda Return Type	✓	✓	✓	
Function Template	✓	✓	✓	
Class Template	✓	✓	✓	
Non-Type Template Parameter				
C Style Cast				
C++ Named Cast				
new Expression				

*Excludes function pointer return types.

Does it work?

- Limited number of unit tests.
- 51 FileCheck tests passing for contexts completed so far.
- Ran on a mantid source file the other day generates a 181kB patch.
  - Looks correct
  - Compiles and passes unit tests.
- Looking to upstream to the official clang project when complete.

What Else Can We Do With These Tools?

- Google have used this to perform large scale refactoring and API migrations on their monolithic C++ codebase.
- Clang-Refactor is an ongoing project to do to refactoring tools what Clang-Tidy does for linter checks.
- Clang-D is an ongoing project to build a C++ ‘Language Server’.
  - This is what powers the VS Code C++ support.

Limitations

- Unlike Clang-Format the source code has to be in a compilable state for the AST to be built and the tool to run.
- The AST is built after the preprocessor is run so if large sections of the code are conditionally compiled then the tool needs to be run for each possible combination of macro values.
- The tool has to be run for each Translation Unit individually and so cross translation unit refactorings are currently ~~not possible~~/~~slow~~/difficult.
- At the moment the check only works for a set of contexts determined at compile time and will need to be updated to fix any new contexts.
  - On the plus side, this guarantees that any contexts which were not considered at compile time are likely to be left in-tact.

Conclusions

- Building tools for C++ is still difficult because of the complexity of the language.
- It is however easier than you might expect and you no longer need to write you own parser (so please don’t).

Any Questions?

Source code available under BSD-3 on GitLab here:
https://gitlab.com/edwardb96/voxel-clang-tidy

`master` currently not guaranteed to be stable.

To install (build clang/llvm first)…

```
$ cd llvm/tools/clang/tools/extra/clang-tidy
$ git clone git@gitlab.com:edwardb96/voxel-clang-tidy.git voxel/
$ git apply voxel/enable_voxel_module.patch
$ cd {{clang-build-dir}}
$ ninja check-voxel-tidy
```