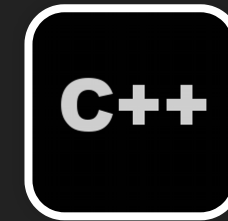# TEACHING OLD COMPILERS NEW TRICKS

Transpiling C++17 to C++11

Tony Wasserka

@fail_cluez

Meeting C++

17 November 2018

WHO AM I?

- Berlin-based consultant: Workflow optimization, Code Modernization
- Focus: Low-level & Type-safety
- Side projects: Game console emulators

PPSSPP

Dolphin

Citra

- Twitter: @fail_cluez

- GitHub: neobrain

- neobrain.github.io

C++14/17: WHY?

```cpp
auto [it, inserted] = my_map.insert(...);
```

```cpp
uint32_t bitmask = 0b1000'0010;
```

```cpp
if (QVariant v = getAnswer(); v.isValid())
    use(var);
```

```cpp
optional<Token> parse(string&);
```

```cpp
using calc_expr = variant<sum, prod>;
calc_expr parsed = calc_parse("5+3*8");
std::visit(eval_expr, parsed);
```

```cpp
template<typename... Bars>
auto total_foos(Bars... bars) {
    return (bars.foo() + ...);
}
```

```cpp
if constexpr (is_array_v<T>) {
  return t[5];
} else {
  return t;
}
```

```cpp
path dir = temp_directory_path() / "test";
create_directory(dir);
```

Check out the C++17 Tony tables

THE PROBLEM

C++17

```cpp
int get_mask() {
    return 0b1000'0000;
}
```

gcc 4.9 →

Assembly

```asm
get_mask:
    mov eax, 128
    ret
```

C++17

```cpp
int get_mask() {
    return 0b1000'0000;
}
```

gcc 4.8 →

Assembly

```asm
get_mask:
    mov eax, 128
    ret
```

SOLUTIONS

- Upgrade the compiler…
  - … if you can
- Stick with old C++…
  - …and put up with the consequences
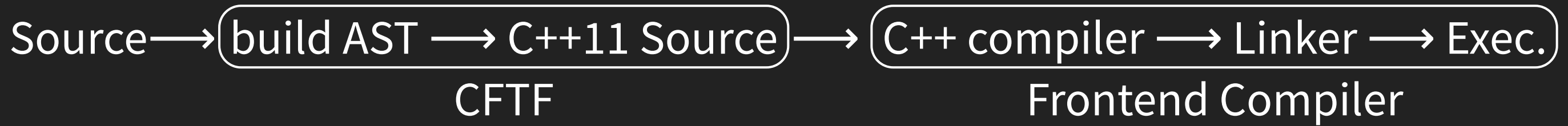- Teach your compiler some C++17

Enter Clang-from-the-Future

CLANG-FROM-THE-FUTURE

- libclang-based tool
- Preprocessor before the compiler runs

Source ⟶ build AST ⟶ C++11 source ⟶ C++ compiler ⟶ Linker ⟶ Exec.

- Automatic conversion of C++17 to C++11

CFTF-ENHANCED PIPELINE

Source ⟶ (build AST ⟶ C++11 Source) ⟶ (C++ compiler ⟶ Linker ⟶ Exec.)

CFTF

Frontend Compiler

CFTF = Black box precompilation step

```
int get_mask() {
  return 0b1000'0000;
}
```

⟶

```
int get_mask() {
  return 128;
}
```

A SOLUTION

C++17

```
int get_mask() {
    return 0b1000'0000;
}
```

gcc 4.9 →

Assembly

```
get_mask:
    mov eax, 128
    ret
```

C++17

```
int get_mask() {
    return 0b1000'0000;
}
```

gcc 4.8 →

Assembly

```
get_mask:
    mov eax, 128
    ret
```

C++17

```
int get_mask() {
    return 0b1000'0000;
}
```

CFTF →

C++11

```
int get_mask() {
    return 128;
}
```

gcc 4.8 →

Assembly

```
get_mask:
    mov eax, 128
    ret
```
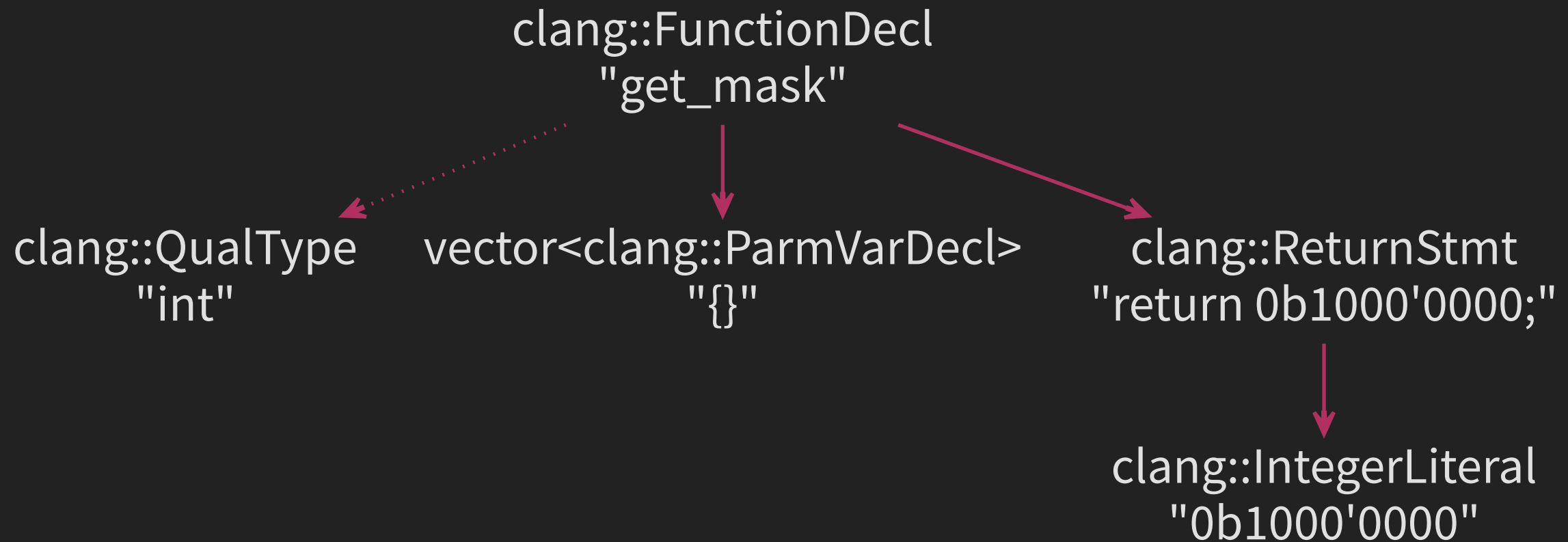
HOW IT WORKS

THE CLANG TOOLING UNIVERSE

- clang-format
- clang-tidy
- CPP2C
- C++ Insights
- Reflection frameworks
- Static analysis
- Tons of custom tools

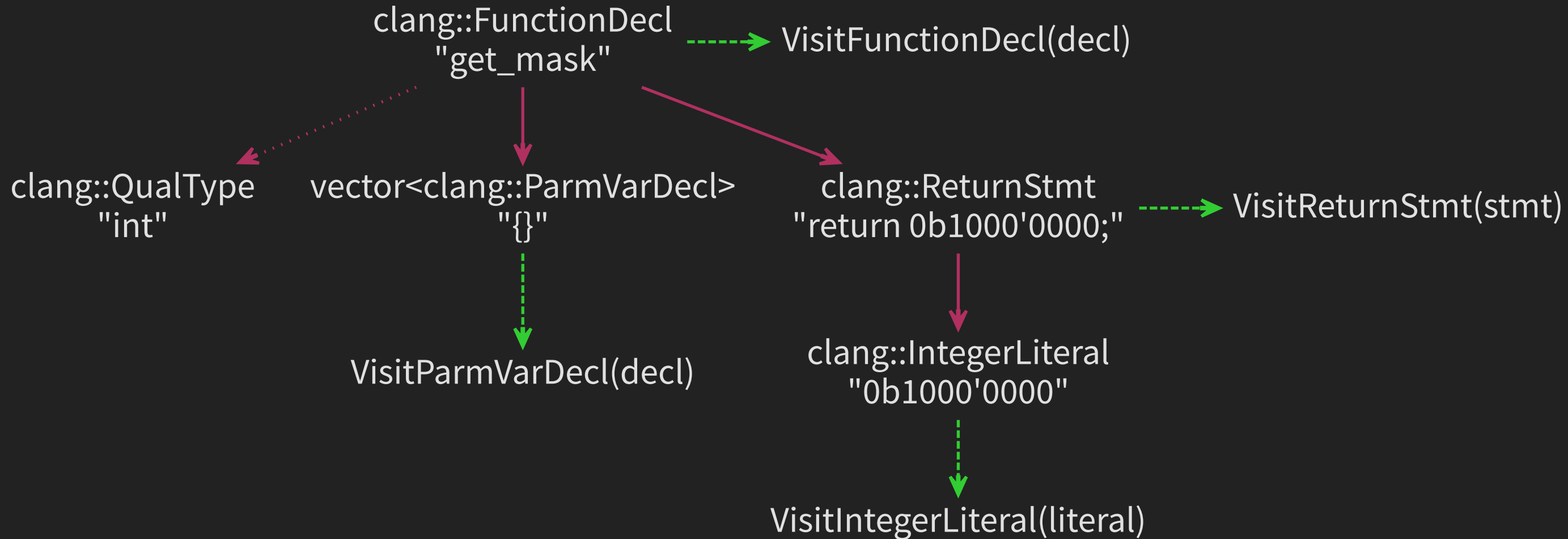# ASTs: Abstract Syntax Trees

```
int get_mask() {
    return 0b1000'0000;
}
```

clang::FunctionDecl
"get_mask"

clang::QualType
"int"

vector<clang::ParmVarDecl>
"{}"

clang::ReturnStmt
"return 0b1000'0000;"

clang::IntegerLiteral
"0b1000'0000"

AST VISITORS

💡 Call C++ function for each node type

clang::FunctionDecl "get_mask" - - → VisitFunctionDecl(decl)

clang::QualType "int"

vector<clang::ParmVarDecl> "{}" - - → VisitParmVarDecl(decl)

clang::ReturnStmt "return 0b1000'0000;" - - → VisitReturnStmt(stmt)

clang::IntegerLiteral "0b1000'0000" - - → VisitIntegerLiteral(literal)

Handled via clang::RecursiveASTVisitor

AST VISITORS

```cpp
void MyASTVisitor::VisitIntegerLiteral(clang::IntegerLiteral* literal) {
  llvm::APInt value = literal->getValue();
  rewriter->ReplaceText(literal->getSourceRange(), value.toString());
}
```
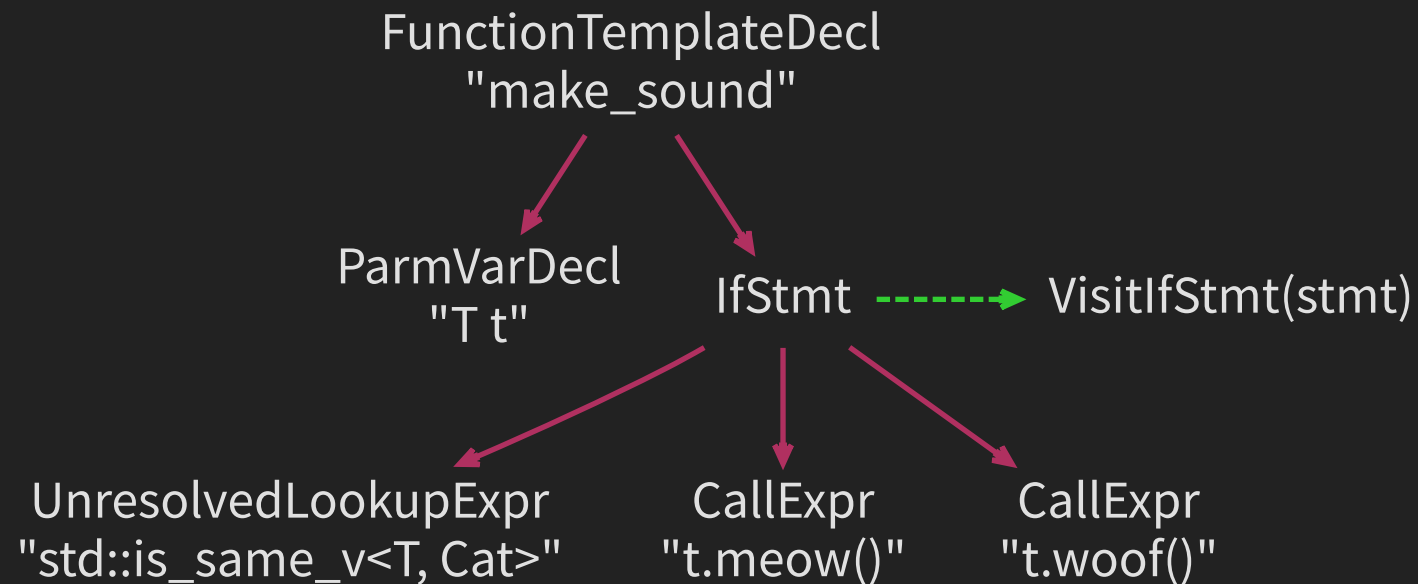
```cpp
int get_mask() {
    return 0b1000'0000;
}
```

$\longrightarrow$

```cpp
int get_mask() {
    return 128;
}
```

"Cool, now do constexpr if!"

CONSTEXPR IF

```cpp
template<typename T>
void make_sound(T t) {
  if constexpr (std::is_same_v<T, Cat>) {
    t.meow();
  } else {
    t.woof();
  }
}
```

FunctionTemplateDecl
"make_sound"

ParmVarDecl
"T t"

IfStmt ------> VisitIfStmt(stmt)

UnresolvedLookupExpr
"std::is_same_v<T, Cat>"

CallExpr
"t.meow()"

CallExpr
"t.woof()"

🤔 So which branch is it?

DEPENDENT CONTEXTS

```cpp
template<typename T>
void make_sound(T t) {
    if constexpr (std::is_same_v<T, Cat>) {
        t.meow();
    } else {
        t.woof();
    }
}
```

↓   ↓

```cpp
template<>
void make_sound(Cat t) {
    using T = Cat;
    if (std::is_same_v<T, Cat>) {
        t.meow();
    } else {
        // Nothing
    }
}
```

```cpp
template<>
void make_sound(Dog t) {
    using T = Dog;
    if (std::is_same_v<T, Cat>) {
        // Nothing
    } else {
        t.woof();
    }
}
```

🔔 Template specializer: Implicit ⇒ Explicit instantiations

CONSTEXPR IF

```cpp
void MyASTVisitor::VisitIfStmt(clang::IfStmt* stmt) {
  if (!stmt->isConstexpr())
    return;

  clang::Expr* cond = stmt->getCond(); // e.g. "std::is_same_v<T, Cat>"
  bool result;
  cond->EvaluateAsBooleanCondition(result, context);
  clang::Stmt* branch_taken = result ? stmt->getThen() : stmt->getElse();

  // Remove "constexpr"
  rewriter->ReplaceText(stmt->getLocStart(), cond->getLocStart(), "if (");

  // Remove inactive branch body
  if (!result) {
    rewriter->ReplaceText(cond->getLocEnd(), branch_taken->getLocStart(), ") {} else");
  } else {
    rewriter->ReplaceText(branch_taken->getLocEnd(), stmt->getLocEnd(), "");
  }
}
```

IN PRACTICE

```
cftf  -frontend-compiler=g++  input.cpp
```

Easy integration into your build pipeline:

- Make:

  ```
  CXX=cftf CXX_FLAGS="-frontend-compiler=g++" make
  ```

- CMake:

  ```
  CXX=cftf cmake -DCMAKE_CXX_FLAGS="-frontend-compiler=g++" .
  ```

- Other setups may need some creativity

SHOWCASE

CURRENT STATUS

- Usable drop-in for gcc/clang on Linux
  Windows/macOS support planned!
- Small initial set of supported C++14/17 features
  (correctness first, then features)

Available for licensing now

Prototype on GitHub

(Full version is proprietary)

FUTURE

- More rewriting rules
- C++20: Contracts, concepts, …
- C++xy to C++03?
- Better test coverage
- Robustness against macros

Your wishes?

FUNDING

Tailor CFTF to your needs:

- Platform integration
- Language features
- Extra tooling

Flexible licensing model

Let's talk!

USE CASES

- Early adoption of new standards
- Use of C++17 libraries in C++11 setups
- Ports to legacy platforms
- Cherry-pick specific features: Concepts, Contracts

SUMMARY

- Compile C++14/17 on an old compiler
- Functional drop-in preprocessor
- Easy integration into existing toolchains
- No source code changes needed

github.com/neobrain/cftf

@fail_cluez   tony.wasserka@gmx.de   neobrain