# Modifying C++ Compilers for Better Static Analysis

REDPOINT

# Welcome!

- I'm June (She/Her)
- I make plugins for Unreal Engine
- I don't like debugging issues that the computer *could* have told me about at compilation time
- Existing static analysis solutions didn't really find the issues I wanted to detect
- So I modified the Clang C++ compiler...

# Agenda

- Examples of incorrect C++ code we want to catch
- Why existing static analysis solutions aren't good enough
- "Clang for Unreal Engine"
- Deep dive into modifying Clang
  - *So you can customize the compiler for your own project-specific needs...*

# Examples of incorrect C++ code

i.e. code that we want the compiler to complain about

REDPOINT

# Incorrect C++ example: Not using const&

```cpp
static void ThisCausesAnUnnecessaryCopy(TArray<FString> CopiedArray)
{
    for (const auto &Entry : CopiedArray)
    {
        UE_LOG(LogTemp, Verbose, TEXT("%s"), *Entry);
    }
}

void Caller()
{
    ThisCausesAnUnnecessaryCopy(TArray<FString>{
        TEXT("A"),
        TEXT("B"),
        TEXT("C"),
    });
}
```

```cpp
static void ThisCausesAnUnnecessaryCopy(const TArray<FString> &CopiedArray)
{
    for (const auto &Entry : CopiedArray)
    {
        UE_LOG(LogTemp, Verbose, TEXT("%s"), *Entry);
    }
}
```

# Incorrect C++ example: Missing field initialization

```cpp
class FMyClass
{
private:
    int Value1;
    int Value2; // new field we forgot to add to initializer list; not initialized in optimized builds.

public:
    FMyClass()
        : Value1()
    {
    }
};
```

**Note:** You might really want to leave this uninitialized - Clang for Unreal Engine supports doing this with pragmas; the same way you would silence any warning inline

# Incorrect C++ example: Runtime assert due to API usage

```cpp
static void ThisCausesARuntimeAssert()
{
    TArray<FString> Array;
    Array.Add(TEXT("Test"));
    Array.Add(TEXT("Test2"));
    Array.Add(TEXT("Test3"));

    for (const auto &Entry : Array)
    {
        if (Entry == TEXT("Test2"))
        {
            // asserts at runtime, even though we're not continuing iteration after modification
            Array.Remove(Entry);
            break;
        }
    }
}
```

Unreal Engine's TArray template forbids calling 'Remove' with a reference that points inside the array's own memory

We should use indexed iteration here.

We could also avoid this by doing 'for (auto Entry ...)' but then we're copying each element..

# Existing static analysis solutions for C++

REDPOINT

## **Inline static analysis:** Clang --analyze flag, MSVC /analyze:plugin (ESPX)

- ✅ Runs during compilation
- ⛔ Only finds generic C++ mistakes - can't find project or domain specific issues
- ⛔ Runs over all headers included in the translation unit
- 🟡 Need to modify your build toolchain to turn these on

## **Out-of-band static analysis:** clang-tidy, etc.

- ⛔ Runs in addition to compilation
  *(so you can't run this on every build in VS without ~doubling compilation time)*
- 🟡 More customizable and catches a wider range of C++ mistakes, but still probably can't find project or domain specific issues
- ⛔ Runs over more code than you want
  - Some options for excluding headers, but at least for clang-tidy, it's not very performant
  - Also - still has to re-parse all your code! So skipping headers only saves so much..

# What did I need from a solution?

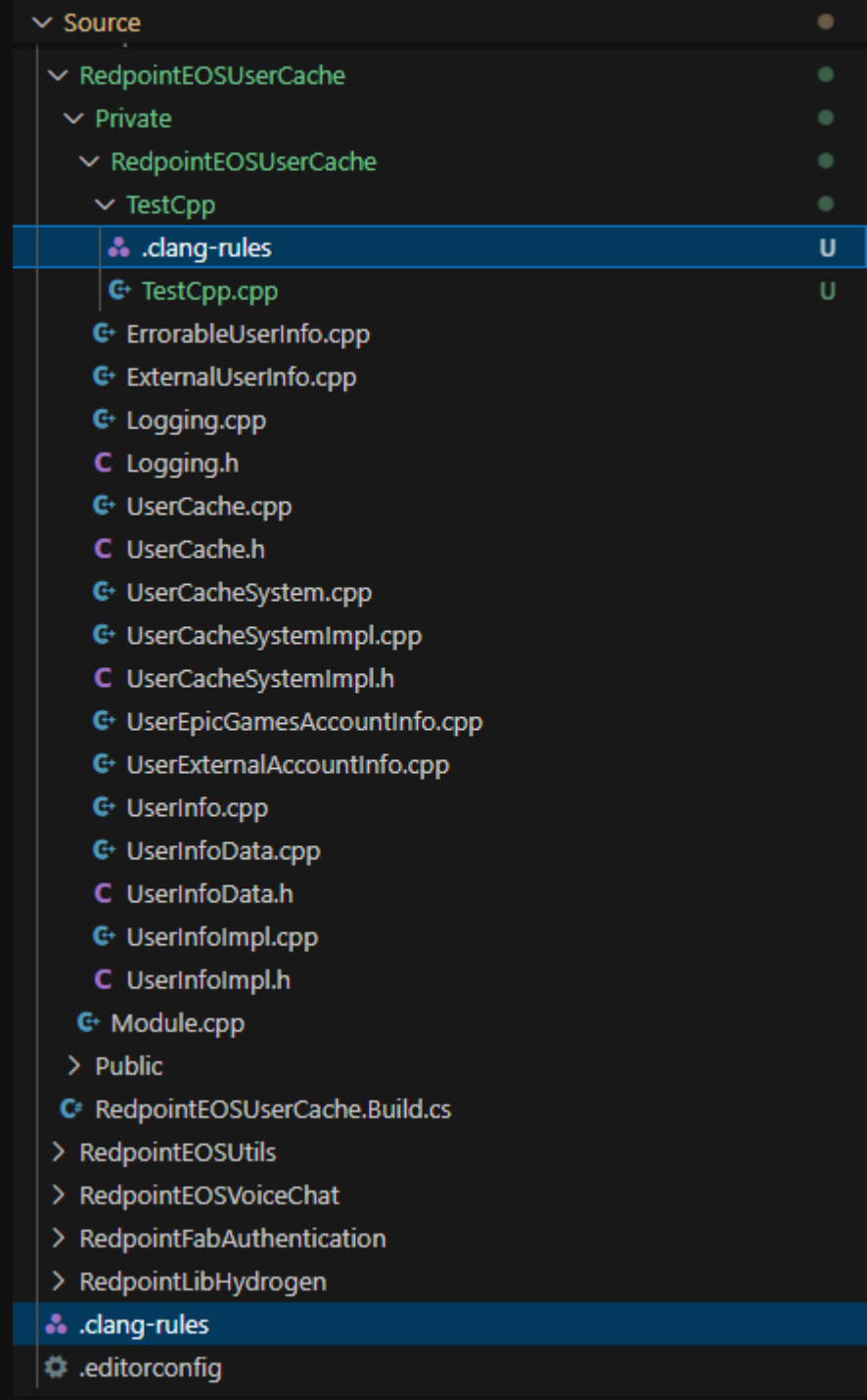
- Run on every build in VS, must be an inline solution
  - *Re-use all the compiler's parsing work*
- Target arbitrary parts of the C++ AST for our own rules
- Customizable *without* recompiling the compiler
- Aware of Unreal Engine specific macros such as UCLASS()
- Run only on the code that I care about in the translation unit
  - *Skip quickly over system and engine headers*

# Static analysis with Clang for Unreal Engine

REDPOINT

# Overview of Clang for Unreal Engine

- Modified version of Clang that can be used as a drop-in replacement
- Has 'clang-cl' to replace MSVC
- Reads .clang-rules files for custom static analysis rules
- Applies rules only to code in that directory tree
- Provides extra AST metadata from UCLASS(), etc macros
- *Extremely* fast




REDPOINT

# How can we can catch those incorrect C++ examples?

- This will be a brief intro to setting Clang for Unreal Engine rulesets for static analysis
- There's installation instructions and plenty more examples on the official wiki:
  - https://github.com/RedpointGames/llvm-project/wiki
- *(We want to get to the deep dive of Clang internals!)*

# Detected: Not using const&

```cpp
static void ThisCausesAnUnnecessaryCopy(TArray<FString> CopiedArray)
{
    for (const auto &Entry : CopiedArray)
    {
        UE_LOG(LogTemp, Verbose, TEXT("%s"), *Entry);
    }
}

void Caller()
{
    ThisCausesAnUnnecessaryCopy(TArray<FString>{
        TEXT("A"),
        TEXT("B"),
        TEXT("C"),
    });
}
```

```
- # Find function parameters that could be 'const &'.
  # @note: We exclude functions starting with 'On' because we assume they might
  # have delegate captures that must not be passed by reference.
  Name: performance-unnecessary-value-param
  Matcher: |
    functionDecl(
      hasBody(stmt()),
      isDefinition(),
      unless(isImplicit()),
      unless(matchesName("::On.*")),
      unless(cxxMethodDecl(anyOf(isOverride(), isFinal()))),
      has(typeLoc(forEach(
        parmVarDecl(
          hasType(qualType(
            hasCanonicalType(isExpensiveToCopy()),
            unless(hasCanonicalType(referenceType())))),
          decl().bind("param")
        )
      ))),
      unless(isInstantiated()), decl().bind("functionDecl")
    )
  ErrorMessage: |
    parameter should be made 'const &' to avoid unnecessary copy
  Callsite: param
```

REDPOINT

# Detected: Not using const&

```cpp
static void ThisCausesAnUnnecessaryCopy(TArray<FString> CopiedArray)
{
    for (const auto &Entry : CopiedArray)
    {
        UE_LOG(LogTemp, Verbose, TEXT("%s"), *Entry);
    }
}
```

```
…\TestCpp.cpp(18,41): error : parameter should be made 'const &' to avoid unnecessary copy
 [-Wredpoint.games/performance-unnecessary-value-param]
   18 | static void ThisCausesAnUnnecessaryCopy(TArray<FString> CopiedArray)
      |                                         ^
```

REDPOINT

# Detected: Missing field initialization

```
class FMyClass
{
private:
    int Value1;
    int Value2; // new field we forgot to add to initializer list; not initialized in optimized builds.

public:
    FMyClass()
        : Value1()
    {
    }
};
```

```
- # Detects if a field in a class or struct is not initialized in the
  # constructor's initialization list when at least one member is initialized
  # via the initializer list.
  Name: field-not-initialized
  Matcher: |
    cxxConstructorDecl(
      unless(isImplicit()),
      unless(isDelegatingConstructor()),
      unless(isDeleted()),
      unless(isDefaulted()),
      hasBody(stmt()),
      unless(ofClass(cxxRecordDecl(isUClass()))),
      unless(ofClass(cxxRecordDecl(isUInterface()))),
      ofClass(cxxRecordDecl(forEach(fieldDecl().bind("declared_field")))),
      forNone(cxxCtorInitializer(forField(fieldDecl(equalsBoundNode("declared_field")).bind("referenced_field"))))
    ).bind("bad_constructor")
  ErrorMessage: |
    one or more fields will be uninitialized when class or struct is constructed; please add the field to the initializer list.
  Callsite: bad_constructor
  Hints:
    declared_field: this field must be initialized
```

REDPOINT

# Detected: Missing field initialization

```
class FMyClass
{
private:
    int Value1;
    int Value2; // new field we forgot to add to initializer list; not initialized in optimized builds.

public:
    FMyClass()
        : Value1()
    {
    }
};
```

```
…\TestCpp.cpp(12,5): error : one or more fields will be uninitialized when class or struct is constructed;
please add the field to the initializer list.
 [-Wredpoint.games/field-not-initialized]
   12 |     FMyClass()
      |     ^
…\TestCpp.cpp(9,5): note: this field must be initialized
    9 |     int Value2; // new field we forgot to add to initializer list; not initialized in optimized
builds.
      |     ^
```

REDPOINT

# Detected: Runtime assert due to API usage

```
static void ThisCausesARuntimeAssert()
{
    TArray<FString> Array;
    Array.Add(TEXT("Test"));
    Array.Add(TEXT("Test2"));
    Array.Add(TEXT("Test3"));

    for (const auto &Entry : Array)
    {
        if (Entry == TEXT("Test2"))
        {
            // asserts at runtime, even though we're not c
            Array.Remove(Entry);
            break;
        }
    }
}
```

```
- Name: broken-array-call
  Matcher: |
    cxxMemberCallExpr(
      on(
        declRefExpr(
          hasType(cxxRecordDecl(hasName("TArray"))),
          to(decl().bind("array_declared_here"))
        ).bind("array_callsite")
      ),
      callee(
        cxxMethodDecl(
          matchesName("(Insert|Insert_GetRef|Add|Add_GetRef|Remove|RemoveSwap)")
        )
      ),
      hasArgument(
        0,
        declRefExpr(
          to(
            varDecl(
              hasType(referenceType()),
              hasAncestor(
                cxxForRangeStmt(
                  hasRangeInit(
                    declRefExpr(
                      to(decl(equalsBoundNode("array_declared_here")))
                    )
                  )
                ).bind("array_for_range")
              )
            ).bind("dangerous_ref_declaration")
          )
        ).bind("dangerous_ref_usage")
      )
    ).bind("bad_callsite")
  ErrorMessage: |
    incorrect usage of mutating array call with non-copy will lead to crash at runtime
  Callsite: bad_callsite
  Hints:
    array_for_range: "make this a copy instead of a ref (i.e. not const&), or switch to an index-based for loop"
```

REDPOINT

# Detected: Runtime assert due to API usage

```cpp
static void ThisCausesARuntimeAssert()
{
    TArray<FString> Array;
    Array.Add(TEXT("Test"));
    Array.Add(TEXT("Test2"));
    Array.Add(TEXT("Test3"));

    for (const auto &Entry : Array)
    {
        if (Entry == TEXT("Test2"))
        {
            // asserts at runtime, even though we're not continuing iteration after modification
            Array.Remove(Entry);
            break;
        }
    }
}
```

```
…\TestCpp.cpp(38,13): error : incorrect usage of mutating array call with non-copy will lead to crash at
runtime
 [-Wunreal.redpoint.games/broken-array-call]
   38 |             Array.Remove(Entry);
      |             ^
…\TestCpp.cpp(33,5): note: make this a copy instead of a ref (i.e. not const&), or switch to an index-based
for loop
   33 |     for (const auto &Entry : Array)
      |     ^
```

REDPOINT

# How do I figure out what the rule expression should be?

- Godbolt: https://godbolt.org/
- Select Clang as the compiler
- Turn on the "clang-query" tool with the AST matcher reference open
  - https://clang.llvm.org/docs/LibASTMatchersReference.html
- Write sample code you want to match against
- Use Godbolt to look at the AST, and the clang-query input to test your matcher with:
  - match <expr>

# Deep dive into modifying Clang

REDPOINT

## Get the source code

- github.com/llvm/llvm-project/
- Clang 18.x - use 'release/18.x' branch

- *Tip:* Modifying the compiler means you can also backport fixes from newer versions!

## Generate projects

- CMake
- Recommend to keep Debug/Release build directories completely separate
- Otherwise when you change configurations in VS, it can delete all your intermediate build artifacts
- Clang takes a *long* time to build from scratch
- Debug is not that useful - simple 30 line C++ files are OK, but not much else

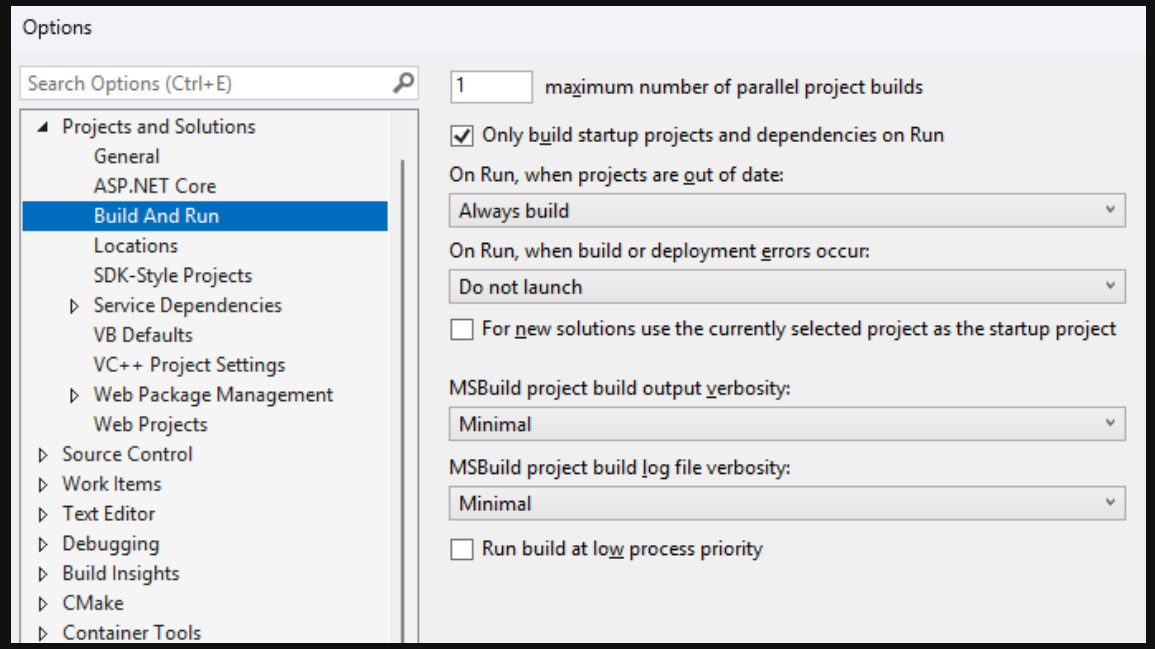
# Generate projects - Recommended settings

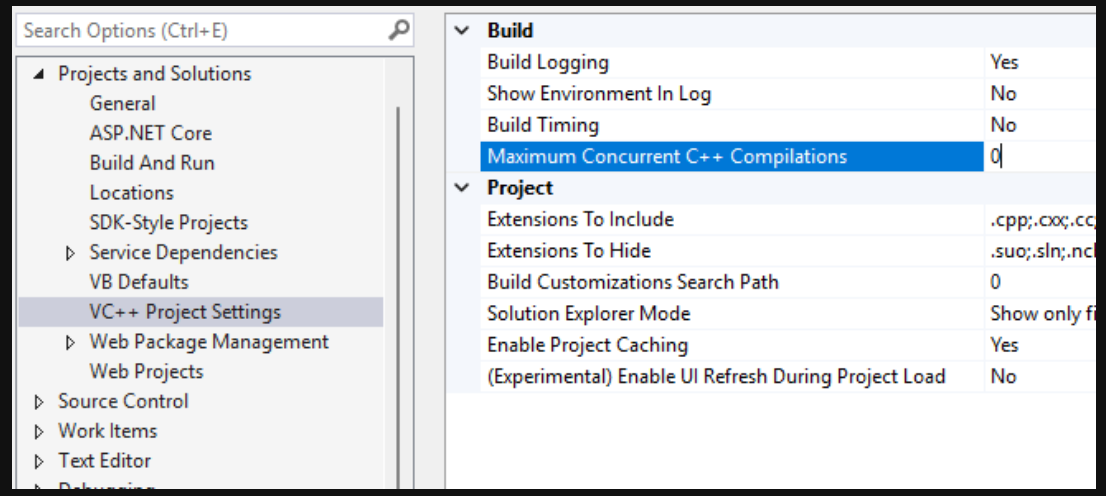
```
& "C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\cmake.exe" `
    "-G" `
    "Visual Studio 17 2022" `
    "-DLLVM_ENABLE_PROJECTS:STRING=clang;lld" `
    -DLLVM_INSTALL_TOOLCHAIN_ONLY:BOOL=TRUE `
    -DLLVM_INCLUDE_BENCHMARKS:BOOL=FALSE `
    -DLLVM_INCLUDE_DOCS:BOOL=FALSE `
    -DLLVM_INCLUDE_EXAMPLES:BOOL=FALSE `
    -DLLVM_INCLUDE_TESTS:BOOL=FALSE `
    -DCLANG_INCLUDE_TESTS:BOOL=FALSE `
    -DCLANG_INCLUDE_DOCS:BOOL=FALSE `
    -DLLVM_ENABLE_DIA_SDK:BOOL=FALSE `
    -DCMAKE_BUILD_TYPE=Release `
    -DCMAKE_CFG_INTDIR=Release `
    -DCMAKE_CONFIGURATION_TYPES=Release `
    "-DCMAKE_INSTALL_PREFIX=C:\Program Files\LLVM" `
    -Hllvm `
    "-Bbuild\win64\release"
if ($LastExitCode -ne 0) {
    exit $LastExitCode
}
```

- Generate.ps1 in the example repository

# Visual Studio optimal settings for clean build

- Clean build or lots of files need to recompile:

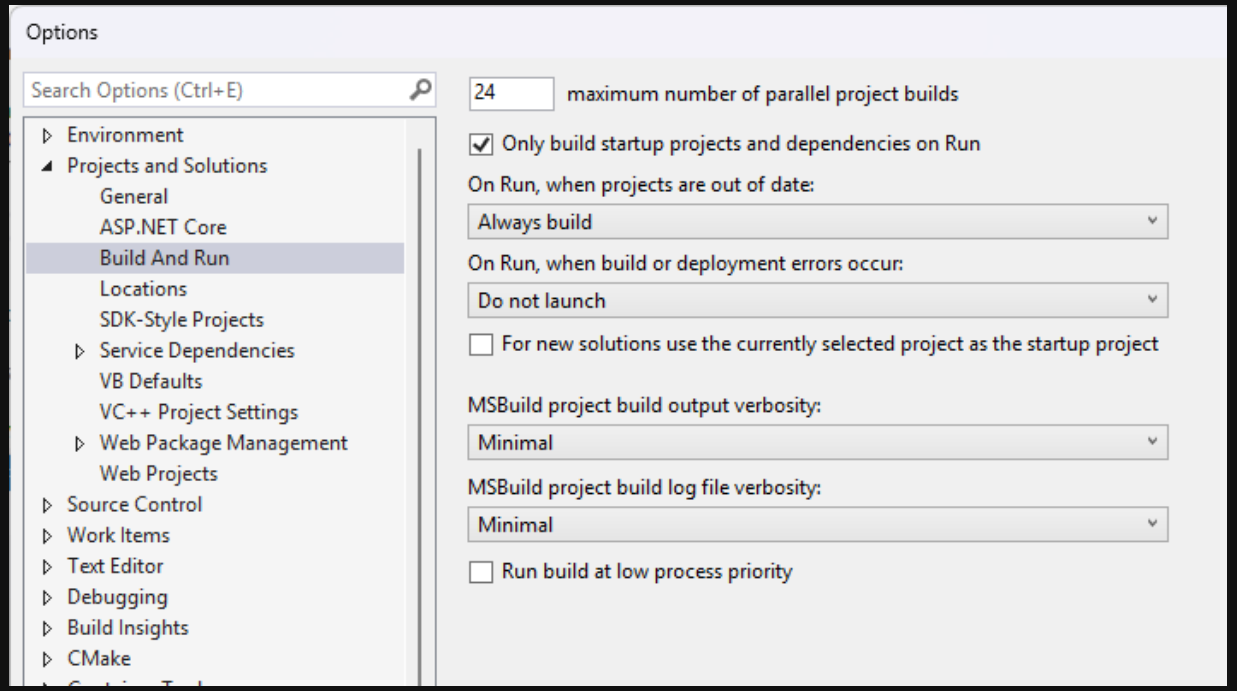



REDPOINT

# Visual Studio optimal settings for incremental build

- Not that many to rebuild? Increase maximum number of parallel project builds
- Otherwise you will be bottlenecked on VS checking if each project is "up to date"




REDPOINT

# Clang has lots of moving parts

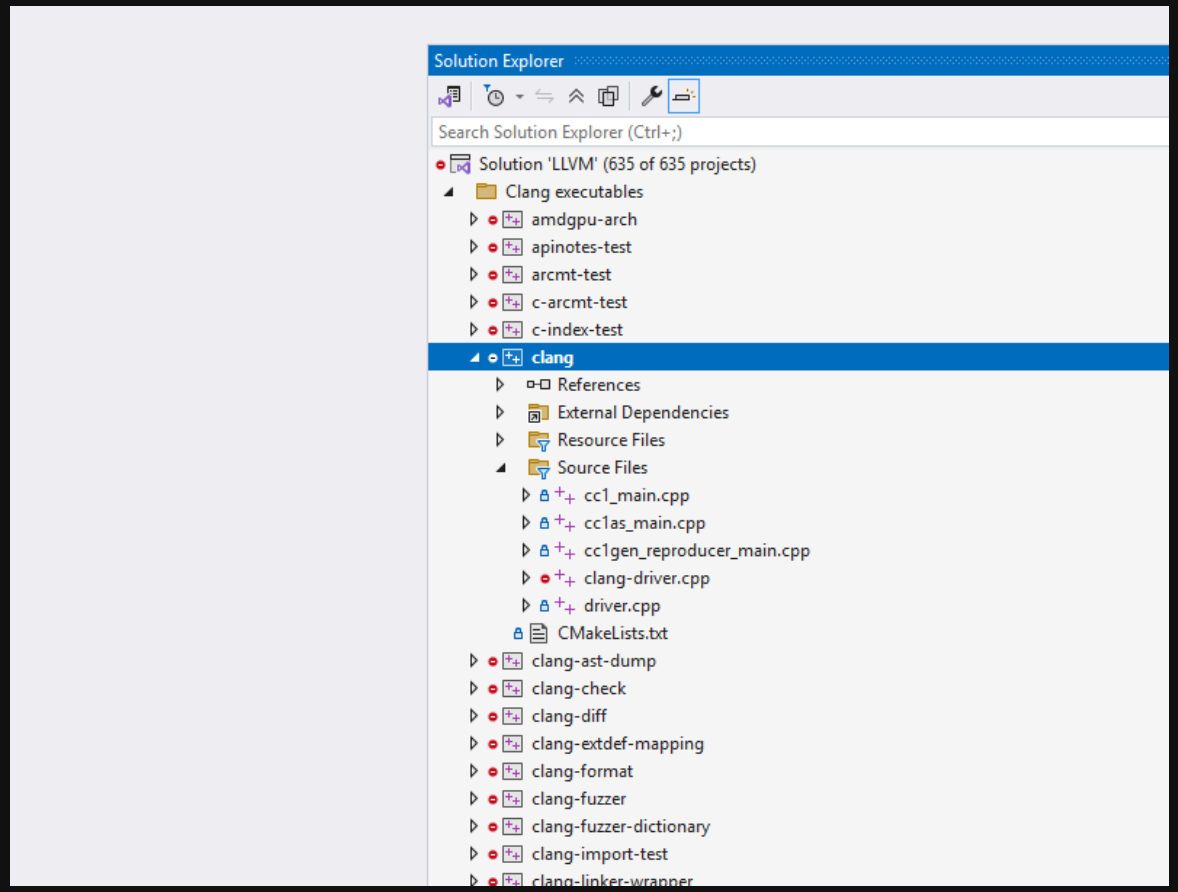
- Starting with simple injection via ASTConsumer
  - Parser -> ASTConsumer (typically codegen or multiplex consumer)

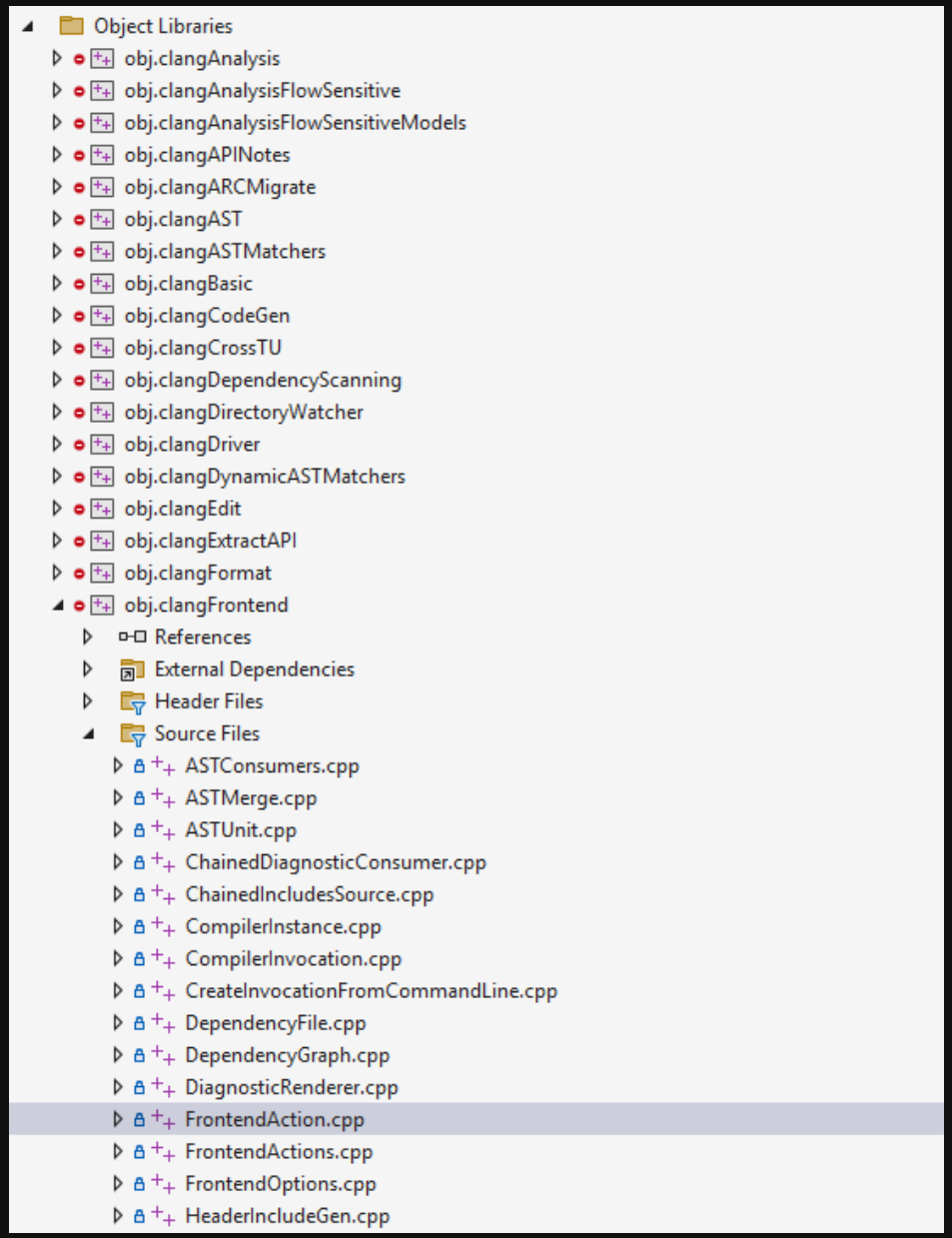
- `clang/lib/Frontend/FrontendAction.cpp`
- `FrontendAction::CreateWrappedASTConsumer`
  - This is where the compiler allows dynamic plugins on Linux to register AST consumers, and where we'll force our AST consumer to get added to the pipeline.

- ASTConsumers get passed the translation unit (root of AST) after all parsing is complete

- *Tip:* CodeGenAction (the thing that produces object files) is an ASTConsumer!

# Where is FrontendAction?




REDPOINT

# Example ASTConsumer

```
// GCAP

class GCAPCustomASTConsumer : public ASTConsumer {
public:
  void HandleTranslationUnit(ASTContext &AST) override {
    // @todo: do something with the AST.
    llvm::errs() << "Hello from GCAP demo!";
  }

  static std::unique_ptr<ASTConsumer> CreateASTConsumer(clang::CompilerInstance &CI) {
    return std::make_unique<GCAPCustomASTConsumer>();
  }
}
```

```
FrontendAction::CreateWrappedASTConsumer(CompilerInstance &CI,
    }

    // Add custom consumer prior to code generation.
    Consumers.push_back(
      GCAPCustomASTConsumer::CreateASTConsumer(CI));

    // Add to Consumers the main consumer, then all the plugins that go after it
    Consumers.push_back(std::move(Consumer));
    if (!AfterConsumers.empty()) {
```

- In our demo, we're just putting GCAPCustomASTConsumer directly inside FrontendAction.cpp
- You probably want to split this out for better maintainability
- Add it to pipeline in CreateWrappedASTConsumser

REDPOINT

# Make sure you delete this check!

```
// If there are no registered plugins we don't need to wrap the consumer
if (FrontendPluginRegistry::begin() == FrontendPluginRegistry::end())
  return Consumer;
```

If you don't, your manually added consumer later won't ever run!

# Example ASTConsumer - Demo

- Live demo time (hopefully)

```cpp
#include <string>
#include <iostream>

class SomeClass {
private:
    int MyField;
};

int main() {
    std::cout << "Hello world!\n" << std::endl;
    return 0;
}
```

```
PS F:\gcap-llvm-project\__gcap_examples> .\001-simple-ast-consumer\bin\clang-cl.exe
 -D_ALLOW_COMPILER_AND_STL_VERSION_MISMATCH .\Example.cpp
Hello from GCAP demo!
```

# Simple example of matching on the AST

- Iterate through nodes in the translation unit received
- Using AST matchers

# What are AST matchers?

- Expressions that allow easier traversal of the AST

- Explore what the AST looks like, and use "clang-query" to test your matchers
  - Online: https://godbolt.org/
  - AST matcher reference: https://clang.llvm.org/docs/LibASTMatchersReference.html

- Dynamic AST matchers -> construct an AST matcher expression from a string
- Static AST matchers -> expression in your modified Clang C++ code

- "Clang for Unreal Engine" fork uses dynamic AST matchers for .clang-rules
- Static AST matchers are more powerful since you have the full Clang C++ API

# First, we need to patch Clang a little bit

- AST matcher API is a little weird - no way to test a match against a decl *and* recurse inside it
  - Only recursive match API is against the whole translation unit
  - Which you will want to avoid if you want to change what matchers run for different files

# First, we need to patch Clang a little bit

- `clang/include/clang/ASTMatchers/ASTMatchFinder.h`
- Add "matchDecl" after "matchAST"

```
/// Finds all matches in the given AST.
void matchAST(ASTContext &Context);

void matchDecl(clang::Decl *Decl, ASTContext &Context);
```

# First, we need to patch Clang a little bit

- `clang/lib/ASTMatchers/ASTMatchFinder.cpp`
- Add "matchDecl" after "matchAST"

```
void MatchFinder::matchAST(ASTContext &Context) {
  internal::MatchASTVisitor Visitor(&Matchers, Options);
  internal::MatchASTVisitor::TraceReporter StackTrace(Visitor);
  Visitor.set_active_ast_context(&Context);
  Visitor.onStartOfTranslationUnit();
  Visitor.TraverseAST(Context);
  Visitor.onEndOfTranslationUnit();
}

void MatchFinder::matchDecl(clang::Decl *Decl, ASTContext &Context) {
  internal::MatchASTVisitor Visitor(&Matchers, Options);
  Visitor.set_active_ast_context(&Context);
  Visitor.getDerived().TraverseDecl(Decl);
}
```

# Simple example of matching on the AST

```
// GCAP

class GCAPCustomASTConsumer : public ASTConsumer,
    public clang::ast_matchers::MatchFinder::MatchCallback
{
public:
  void HandleTranslationUnit(ASTContext &AST) override {
    using namespace clang::ast_matchers;

    // Get the translation unit node.
    const auto *UnitDeclEntry = AST.getTranslationUnitDecl();
    if (UnitDeclEntry == nullptr) {
      // no AST for some reason; ignore.
      return;
    }

    // Create our matcher.
    std::unique_ptr<ast_matchers::MatchFinder> Finder =
        std::make_unique<ast_matchers::MatchFinder>();

    // Add our AST matcher to find a field called 'A'.
    Finder->addMatcher(fieldDecl(hasName("A")).bind("target"), this);

    // Iterate through top-level declarations.
    for (const auto &DeclEntry : UnitDeclEntry->decls()) {
      Finder->matchDecl(DeclEntry, AST);
    }
  }
```

- Get the translation unit decl
- Create an AST match finder
- Add the matcher with ourselves as the callback (note the additional parent class on our ASTConsumer)
- Iterate through top-level decls and call matchDecl

# Simple example of matching on the AST

```
  virtual void
    run(const clang::ast_matchers::MatchFinder::MatchResult& Result) override
  {
    const Decl* FoundTarget = Result.Nodes.getNodeAs<Decl>("target");
    if (!FoundTarget) {
      // 'target' not found in matcher expression, or node wasn't a Decl.
      return;
    }

    // @todo: we're just dumping the AST node to output.
    FoundTarget->dump();
  }

  static std::unique_ptr<ASTConsumer>
  CreateASTConsumer(clang::CompilerInstance &CI) {
    return std::make_unique<GCAPCustomASTConsumer>();
  }
};
```

- In our callback...
- Grab the 'target' node of the match
- For now, just dump the AST of that node to the console.

# Simple example of matching on the AST - Demo

- Live demo time (hopefully)

```
#include <string>
#include <iostream>

#define HELLO()

HELLO()
class SomeClass {
private:
    int MyField;
    int A;
    int B;
    int C;
};

int main() {
    std::cout << "Hello world!\n" << std::endl;
    return 0;
}
```

```
PS F:\gcap-llvm-project\__gcap_examples> .\002-ast-matcher\bin\clang-cl.exe  -D_ALLO
W_COMPILER_AND_STL_VERSION_MISMATCH .\Example.cpp
FieldDecl 0x210e1defc90 <.\Example.cpp:10:5, col:9> col:9 A 'int'
```

REDPOINT

# Write a custom AST matcher

- `clang/include/clang/ASTMatchers/ASTMatchers.h`
  - Matches are templated, so all defined in the header file here

- `clang/lib/ASTMatchers/Dynamic/Registry.cpp`
  - Registers matches with the dynamic AST matcher registry
  - Only needed if you're doing stuff with dynamic matcher expressions (e.g. clang-query)

# Write a custom AST matcher

```
/// Matches if the matched type is a Plain Old Data (POD) type.
///
/// Given
/// \code
///   class Y
///   {
///   public:
///       int a;
///       std::string b;
///   };
/// \endcode
/// fieldDecl(hasType(qualType(isPODType())))
///   matches Y::a
AST_MATCHER(QualType, isPODType) {
  return Node.isPODType(Finder->getASTContext());
}
```

- We needed this in Clang for Unreal Engine to detect when fields omitted from the constructor initializer list would not be zeroed out.

# Use our custom matcher - example

- Just changing our matcher expression to use isPODType:

```
// Add our AST matcher.
Finder->addMatcher(fieldDecl(hasType(qualType(isPODType()))).bind("target"), this);
```

# Use our custom matcher - demo

- Live demo time (hopefully)

```
PS F:\gcap-llvm-project\__gcap_examples> .\003-ispodtype\bin\clang-cl.exe -D
_ALLOW_COMPILER_AND_STL_VERSION_MISMATCH .\Example.cpp

ity\VC\Tools\MSVC\14.44.35207\include\istream:709:5, col:25> col:16 _Chcount
'streamsize':'long long'
`-InitListExpr 0x22d30b467d0 <col:24, col:25> 'streamsize':'long long'
FieldDecl 0x22d30b4f2b0 <.\Example.cpp:10:5, col:9> col:9 A 'int'
FieldDecl 0x22d30b4f3c8 <.\Example.cpp:12:5, col:9> col:9 C 'int'
```

```cpp
#include <string>
#include <iostream>

#define HELLO()

HELLO()
class SomeClass {
private:
    std::string MyField;
    int A;
    std::string B;
    int C;
};

int main() {
    std::cout << "Hello world!\n" << std::endl;
    return 0;
}
```

# Create a new diagnostic & emit

- You probably want to emit diagnostics properly
- `clang/include/clang/Basic/Diagnostic*.td` files
  - Special syntax Clang has for defining diagnostics and other tables
- `clang/include/clang/Basic/DiagnosticCommonKinds.td`
- `clang/include/clang/Basic/DiagnosticGroups.td`

```
56
57  def err_gcap_custom_error : Error<"this is our custom error">;
58  def err_gcap_custom_warning : Warning<"this is our custom warning, for field '%0'">, InGroup<GCAP>;
59
60  }
```

```
5  def GCAP : DiagGroup<"gcap">;
```

REDPOINT

# Create a new diagnostic & emit - example

```
// GCAP

class GCAPCustomASTConsumer : public ASTConsumer,
    public clang::ast_matchers::MatchFinder::MatchCallback
{
public:
  ASTContext *OurAST;

  void HandleTranslationUnit(ASTContext &AST) override {
    using namespace clang::ast_matchers;

    // Get the translation unit node.
    const auto *UnitDeclEntry = AST.getTranslationUnitDecl();
    if (UnitDeclEntry == nullptr) {
      // no AST for some reason; ignore.
      return;
    }

    // Create our matcher.
    std::unique_ptr<ast_matchers::MatchFinder> Finder =
        std::make_unique<ast_matchers::MatchFinder>();

    // Store a pointer to the AST so we can use it in the callback.
    //
    // @note: It's much safer to create a nested class that implements MatchCallback
    //        and instantiate that on the stack, passing the AST by reference so that
    //        we don't have an ASTContext pointer that can last beyond the lifetime
    //        of this HandleTranslationUnit call.... but this is example code.
    this->OurAST = &AST;

    // Add our AST matcher.
    Finder->addMatcher(fieldDecl(matchesName("GCAP.+")).bind("target"), this);

    // Iterate through top-level declarations.
    for (const auto &DeclEntry : UnitDeclEntry->decls()) {
      Finder->matchDecl(DeclEntry, AST);
    }
  }
```

- Added ASTContext* field so we can share AST between HandleTranslationUnit and matcher callback
- Change our matcher to match on fields that start with 'GCAP'

# Create a new diagnostic & emit - example

```
virtual void
  run(const clang::ast_matchers::MatchFinder::MatchResult& Result) override
{
  const NamedDecl *FoundTarget = Result.Nodes.getNodeAs<NamedDecl>("target");
  if (!FoundTarget) {
    // 'target' not found in matcher expression, or node wasn't a NamedDecl.
    return;
  }

  // Get the location in the source code of our match.
  clang::SourceLocation Loc = FoundTarget->getSourceRange().getBegin();

  // Emit our diagnostic.
  if (FoundTarget->getNameAsString() == "GCAPErrorField") {
    this->OurAST->getDiagnostics().Report(Loc, diag::err_gcap_custom_error);
  } else {
    this->OurAST->getDiagnostics().Report(Loc, diag::err_gcap_custom_warning)
        << FoundTarget->getNameAsString();
  }
}
```

- Decl -> NamedDecl since we want to check the name of the field declaration in the callback now and do something different based on it
- getSourceRange to get the source location of the field
- getDiagnostics().Report to emit
- << operator to pass arguments to %0 ... %N in diagnostic message format

# Create a new diagnostic & emit - demo

- Live demo time (hopefully)

```
PS F:\gcap-llvm-project\__gcap_examples> .\004-diagnostic-emit\bin\clang-cl.exe
 -D_ALLOW_COMPILER_AND_STL_VERSION_MISMATCH .\Example.cpp
.\Example.cpp(13,5): warning: this is our custom warning, for field
      'GCAPMyField' [-Wgcap]
   13 |     int GCAPMyField;
      |     ^
.\Example.cpp(14,5): error: this is our custom error
   14 |     int GCAPErrorField;
      |     ^
1 warning and 1 error generated.
```

```cpp
#include <string>
#include <iostream>

#define HELLO()

HELLO()
class SomeClass {
private:
    std::string MyField;
    int A;
    std::string B;
    int C;
    int GCAPMyField;
    int GCAPErrorField;
};

int main() {
    std::cout << "Hello world!\n" << std::endl;
    return 0;
}
```

REDPOINT

## Source manager

- Every unique file on disk has a file ID
- We can look at the file ID of declarations when iterating through them
- Skip over the decls that aren't in files we care about
- This is how the "Clang for Unreal Engine" fork knows what matchers to evaluate for what files - it can lookup file IDs, file paths and figure out what .clang-rules files apply

# Inspect file IDs of decls - example

```
this->OurAST = &AST;

// Add our AST matcher.
Finder->addMatcher(fieldDecl(hasType(qualType(isPODType()))).bind("target"),
                   this);

// Get the source manager.
const SourceManager &SrcMgr = AST.getSourceManager();

// Track what file we're currently in, and whether we're matching.
FileID CurrentFileID;
bool ShouldRunMatcher = false;

// Iterate through top-level declarations.
for (const auto &DeclEntry : UnitDeclEntry->decls()) {
```

- Switch our matcher back to just isPODType for this example
- Get the source manager
- Declare some local variables to track what file we're in and whether we'll run our matcher

# Inspect file IDs of decls - example

```
// Iterate through top-level declarations.
for (const auto &DeclEntry : UnitDeclEntry->decls()) {

  // Try to figure out if the file this declaration is in has changed.
  bool FileChanged = false;
  {
    FileID NewFileID =
        SrcMgr.getFileID(SrcMgr.getFileLoc(DeclEntry->getLocation()));
    if (NewFileID.isInvalid()) {
      // Not in a real file on disk. Skip matching.
      continue;
    }

    FileChanged = (NewFileID != CurrentFileID);
    if (FileChanged) {
      // Always set the CurrentFileID, even if the next section fails to
      // find file data. This allows us to continue quickly skipping over
      // decls while ever the current file is invalid.
      CurrentFileID = NewFileID;
      ShouldRunMatcher = false;
    }
  }
```

- Get the file ID that the declaration is in
- Set FileChanged to true if the file ID is different
- Turn off 'ShouldRunMatcher' when the file ID changes

# Inspect file IDs of decls - example

```
// If the file has changed, get information about the file.
if (FileChanged) {
  auto FileEntry = SrcMgr.getFileEntryRefForID(CurrentFileID);
  if (!FileEntry) {
    // No file entry for current file ID. Skip matching.
    continue;
  }

  // Contrived example: Just checking the filename.
  ShouldRunMatcher = FileEntry->getName().contains("Example.cpp");
}

if (ShouldRunMatcher) {
  Finder->matchDecl(DeclEntry, AST);
}
```

- If the file has changed...
- Then get the full file entry data in memory.
- For this example, just check that it contains "Example.cpp" - that should be enough to exclude system headers.
- Condition matchDecl on ShouldRunMatcher

# Inspect file IDs of decls - example

```
virtual void
  run(const clang::ast_matchers::MatchFinder::MatchResult& Result) override
{
  const NamedDecl *FoundTarget = Result.Nodes.getNodeAs<NamedDecl>("target");
  if (!FoundTarget) {
    // 'target' not found in matcher expression, or node wasn't a Decl.
    return;
  }

  // Get the location in the source code of our match.
  clang::SourceLocation Loc = FoundTarget->getSourceRange().getBegin();

  // Emit our diagnostic.
  this->OurAST->getDiagnostics().Report(Loc, diag::err_gcap_custom_warning)
      << FoundTarget->getNameAsString();
}
```

- Slight change to our callback
- No longer checking field name.
- We'll emit a warning on all plain old data type fields declared in Example.cpp

# Inspect file IDs of decls - demo

- Live demo time (hopefully)

```
PS F:\gcap-llvm-project\__gcap_examples> .\005-source-manager-fileids\bin\clang-cl.exe  -D_ALLOW_COMPILER_AND_STL_VERSION_MISMATCH .\Example.cpp
.\Example.cpp(9,5): warning: this is our custom warning, for field 'A' [-Wgcap]
    9 |     int A;
      |     ^
.\Example.cpp(10,5): warning: this is our custom warning, for field 'B' [-Wgcap]
   10 |     int B;
      |     ^
2 warnings generated.
```

```cpp
#include <string>
#include <iostream>

#define HELLO()

HELLO()
class SomeClass {
private:
    int A;
    int B;
};

int main() {
    std::cout << "Hello world!\n" << std::endl;
    return 0;
}
```

REDPOINT

# Things I don't have time to cover in depth

- Attaching preprocessor metadata to the AST
  1. Define new annotation tokens in `clang/lib/Basic/TokenKinds.def`
  2. Create a derived class of PPCallbacks & always register from 'Preprocessor' constructor code
  3. When PPCallbacks see macros you care about, inject annotation tokens into the token stream
  4. Replace all calls to PP.Lex(Token) in Parser with your own function
  5. Have this function check for your annotation tokens and call into the Parser/Sema to use them
  6. Otherwise it skips over them until the next normal token
  7. So that you don't need to modify any of the existing Clang parsing code to deal with them

## Things I don't have time to cover in depth

- Threading APIs in Clang
- YAML APIs / reading / writing in Clang
- Dynamic AST matchers
- Runtime creation of diagnostics
- Modifying #pragma to support changing diagnostic levels of runtime diagnostics
- Timing/performance APIs in Clang

# Questions?

Important Links

- Slides: junerhodes.au/history
- Clang for Unreal Engine: github.com/RedpointGames/llvm-project/wiki

- LLVM upstream: github.com/llvm/llvm-project
- Godbolt: godbolt.org
- AST matcher reference: clang.llvm.org/docs/LibASTMatchersReference.html

- Examples/demo: github.com/hach-que/gcap-llvm-project

Social

- Mastodon: mastodon.social/@hq