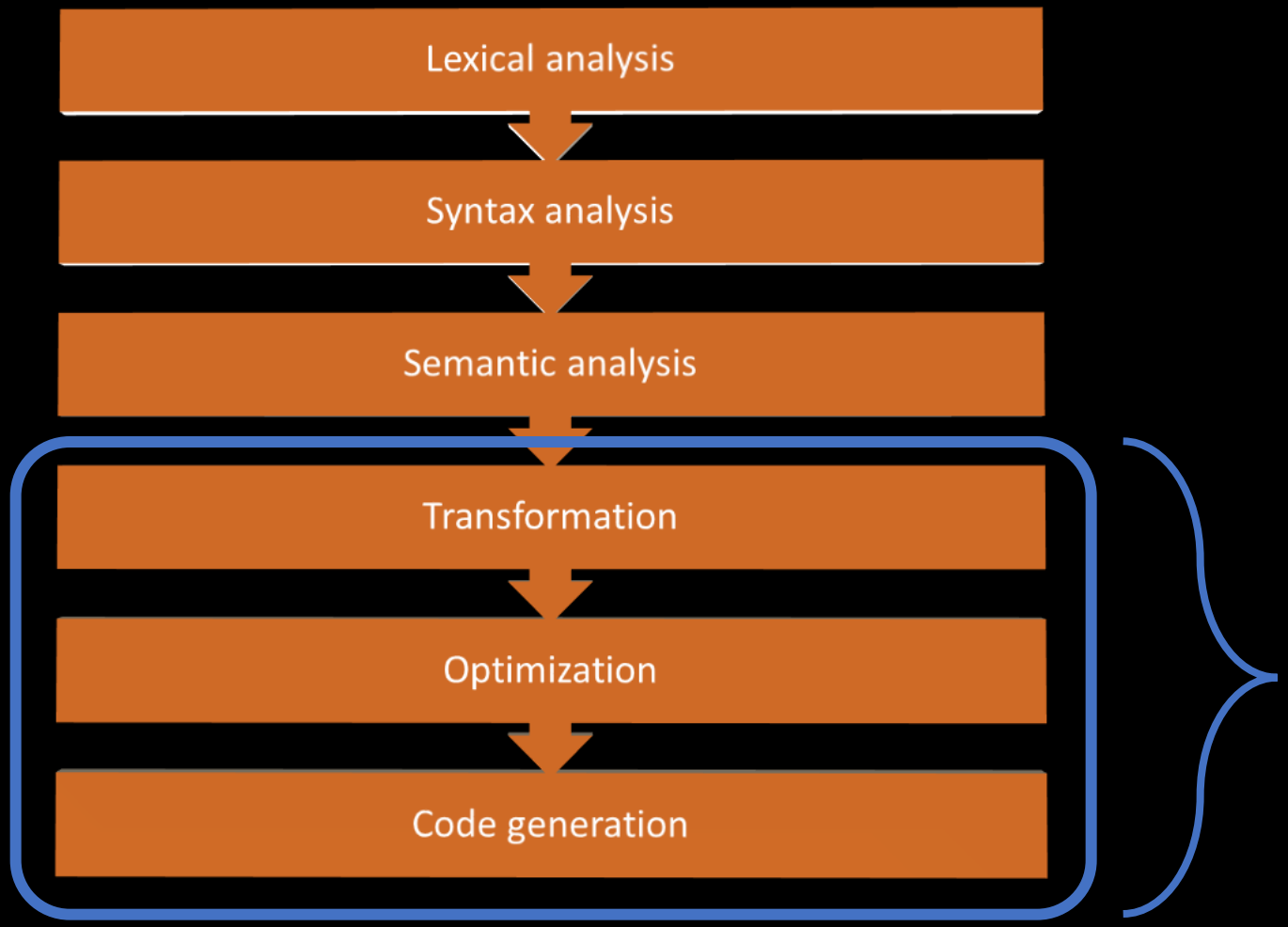
# LLVM

Gembela Gergely

# What is LLVM?

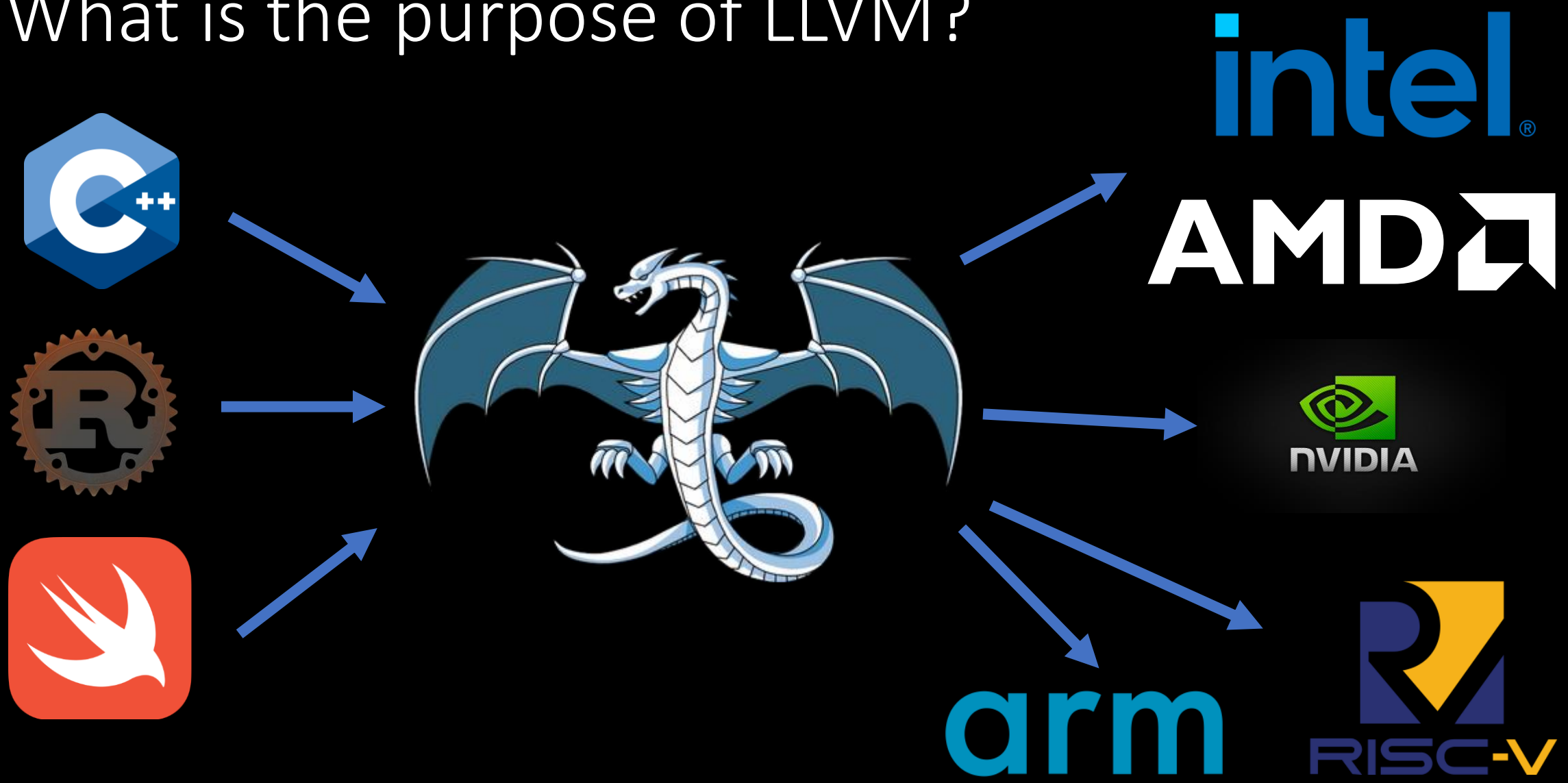
- An open-source compiler infrastructure project, that provides tooling for developing and testing compilers
- LLVM based compilers are available for most popular languages
- There are languages that are developed along with their LLVM based compilers
- LLVM based languages/compilers
  - Rust
  - Zig
  - Nim
  - Clang – C++, Objective C, C
  - Swift

# LLVM in the compilation pipeline




LLVM provides solutions mainly for the middle-end and the backend

# What is the purpose of LLVM?



# The power of the dragon

- IR language
  - Frontend-independent*
  - Platform-independent*
- Optimizations
  - Extensible, customizable passes
  - Applied to the common IR
- Platform-specific backends
- Linker
  - Capable of link-time optimizations

# A modern LLVM-based compiler

Lexer+Parser

AST

# A modern LLVM-based compiler

Lexer+Parser

AST

Semantic analysis

High-level IR

Language specific optimiaztions

Transformations

LLVM IR Generation

# A modern LLVM-based compiler

Lexer+Parser

AST

Semantic analysis

High-level IR

Language specific optimiaztions

Transformations

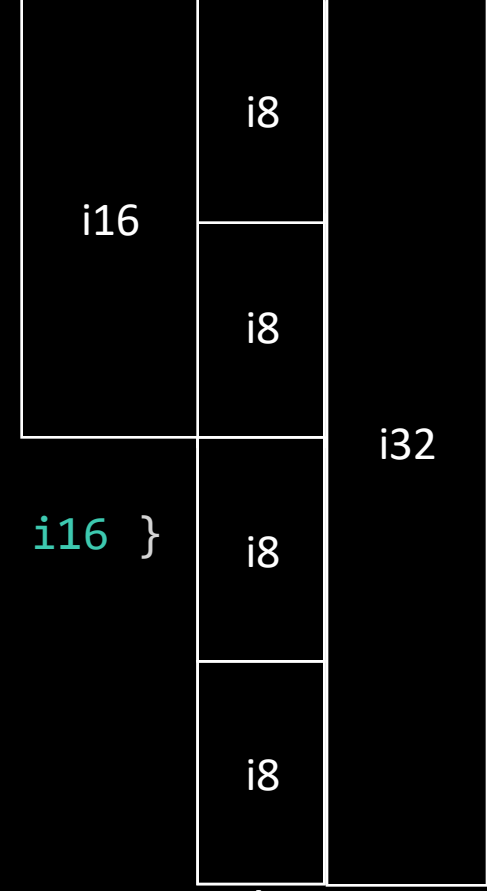
LLVM IR Generation

Optimizations

Target-specific backend

Link-time optimizations (lld)

# LLVM IR




- 3 forms are used – bitcode, in-memory IR, text
- LLVM IR is statically typed

```
%struct.vec4 = type { i16, i16, i16, i16 }
```

  - Even the pointers had types in previous versions
- Custom types can be defined
  - They are similar to C structs, but don't forget alignment!
- 2 kinds of IDs – they identify elements (variables, functions, modules)
  - @global
  - %local
  - LLVM IR organizes code into modules (a module is a compilation unit, e.g. a .cpp file when using C++)
  - In this presentation, we focus on module scope optimizations

# @square – C++

```
int square(int num) {
    return num * num;
} //C/C++
```

```
define i32 @square(int)(i32 noundef
%0) {
  %2 = mul nsw i32 %0, %0
  ret i32 %2
} ;LLVM IR
```

Compiler explorer

# @square – rs

```rust
pub fn square(num: i32) -> i32 {
    num * num
} //RS
```

```
define i32 @square(int)(i32 noundef
%0) {
  %2 = mul nsw i32 %0, %0
  ret i32 %2
} ;LLVM IR
```

Compiler explorer

# @square – Swift

```swift
func square(n: Int) -> Int {
    return n * n
} //Swift
```

```
define i32 @square(int)(i32 noundef
%0) {
  %2 = mul nsw i32 %0, %0
  ret i32 %2
} ;LLVM IR
```

Compiler explorer

# @square – llc (debug/release/ssa/...)

```
define dso_local noundef i32
@square(int)(i32 noundef %0) #0 !dbg !10 {
  %2 = alloca i32, align 4
  store i32 %0, ptr %2, align 4
  tail call void @llvm.dbg.declare(metadata
ptr %2, metadata !16, metadata
!DIExpression()), !dbg !17
  %3 = load i32, ptr %2, align 4, !dbg !18
  %4 = load i32, ptr %2, align 4, !dbg !19
  %5 = mul nsw i32 %3, %4, !dbg !20
  ret i32 %5, !dbg !21
}
```

```
define i32 @square(int)(i32 noundef
%0) {
  %2 = mul nsw i32 %0, %0
  ret i32 %2
} ;LLVM IR
```

# @square – IR to asm

```
square:
        # x86-64
        mov     eax, edi
        imul    eax, eax
        ret
```

```
square(int): # WASM
        local.get       0
        local.get       0
        i32.mul
        end_function
```

```
define i32 @square(int)(i32 noundef %0)
{
  %2 = mul nsw i32 %0, %0
  ret i32 %2
} ;LLVM IR
```

```
square:   //arm64
mul     w0, w0, w0
ret
```

```
square: # RISC-V
mul     a0, a0, a0
ret
```

# nvcc

```
.visible .entry square(int*, int)(
        .param .u64 square(int*, int)_param_0,
        .param .u32 square(int*, int)_param_1
)
{

        ld.param.u64    %rd1, [square(int*, int)_param_0];
        ld.param.u32    %r1, [square(int*, int)_param_1];
        cvta.to.global.u64      %rd2, %rd1;
        mul.lo.s32      %r2, %r1, %r1;
        st.global.u32   [%rd2], %r2;
        ret;

}
```

# LLVM Pass (IR->IR)

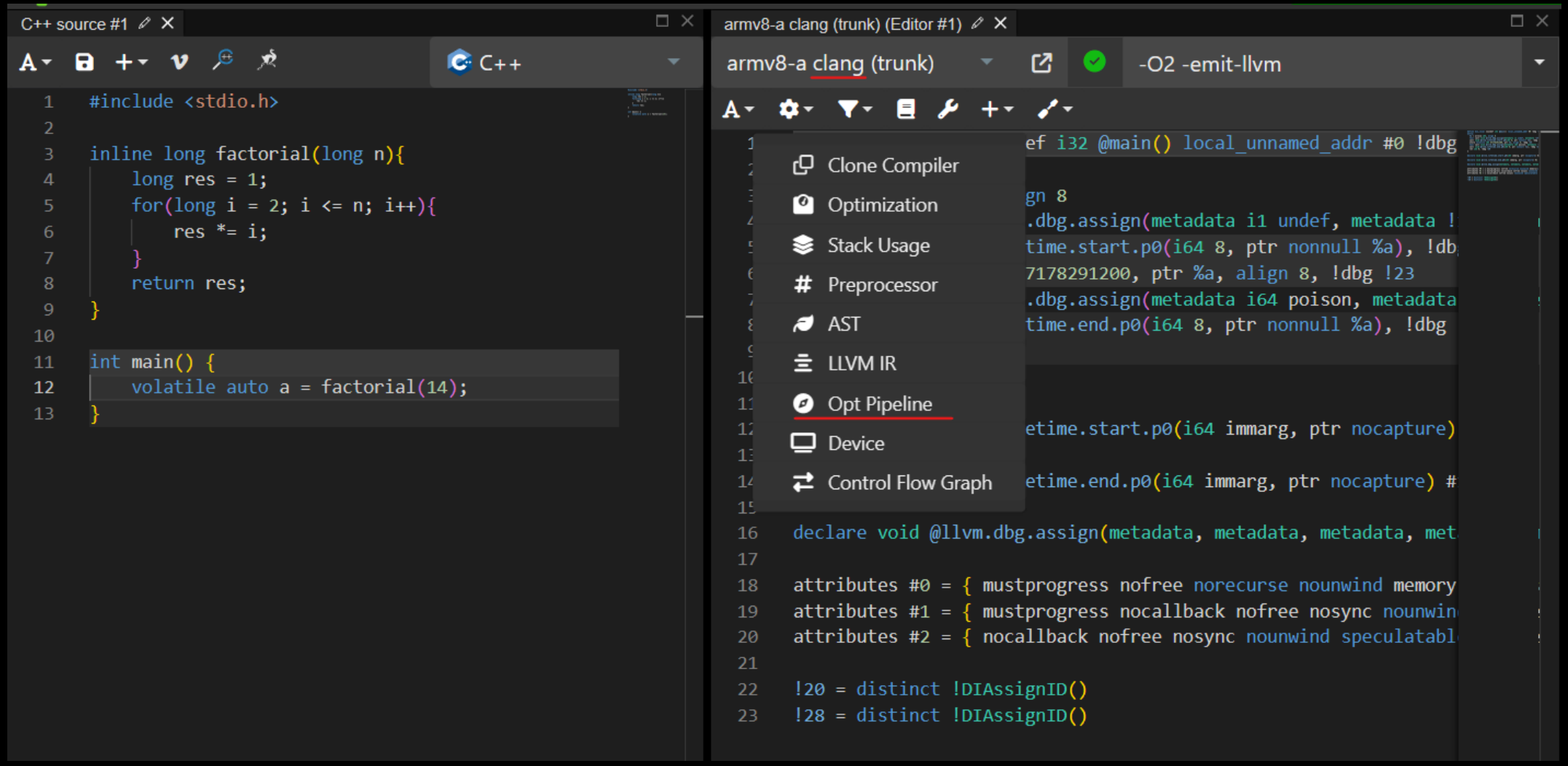
- IR to IR transformation
- Everyone is free to contribute passes
- A *few* passes are already available, ready to use even with custom frontends
  - A custom pass:

```
#include "llvm/IR/PassManager.h"

namespace llvm {

class HelloWorldPass : public PassInfoMixin<HelloWorldPass> {
public:
  PreservedAnalyses run(Function &F, FunctionAnalysisManager &AM);
};

} // namespace llvm
```

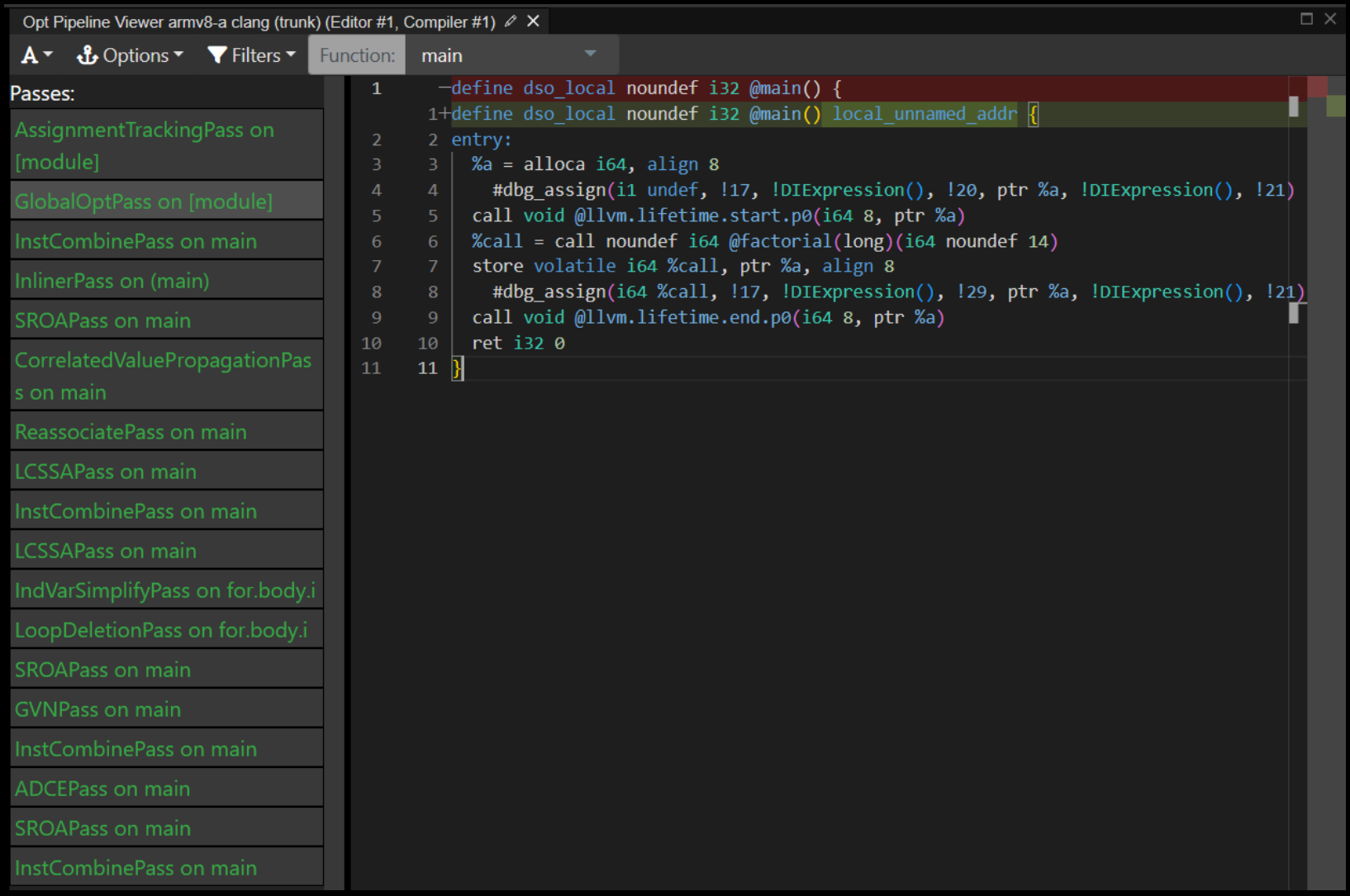
# Optimization with LLVM

- All optimizations presented in the LLVM lecture are available
- There are passes that are not part of LLVM optimization levels (these levels are O[0-3], Ofast, Os, Oz, etc.)
- Not all optimizations are platform independent!
  - Pl. SIMD instructions (SSE/AVX, Arm Neon)
- A pass may revert the transformation of a previous pass if it determines that it can produce a better result
- Specific passes may be applied using the opt tool, e.g.
  - `opt -passes='loop-unroll'`
  - `opt -passes='dce'`

# Inspecting the opt pipeline



# Inspecting the opt pipeline



# Example: dead code elimination

```
long dce(long n){
    long res = 1;
    auto c = res - 3;
    return res;
}
```

Compiler explorer

# Example: Mem2Reg (SROA+Mem2Reg)

```
long sroa(long n){
    long res = 1;
    return res + n;
}
```

Compiler explorer

# Example: loop átalakítása (InvVarSimplify, deletion)

```
int whatever42(int a){

    int x = 3;
    for(int i = 0; i < a; i++){
        int c = a*a;
        x += c;
    }
    return x;
}
```

Compiler explorer

# Example: constant inlining

```
inline long factorial(long n){
    long res = 1;
    for(long i = 2; i <= n; i++){
        res *= i;
    }
    return res;
}

int main() {
    volatile auto a = factorial(14);
}
```

Compiler explorer

# Example: loop unroll

```
void vecadd(float a[4], float b[4]){
    for(int i = 0; i < 4; i++){
        a[i] = b[i];
    }
}
```

Compiler explorer

# Extreme example: dead code elimination #2

```
fn square(num: i32) -> i32 {
    num * num
} //RS
```

# Extreme example: dead code elimination #2

```
fn square(num: i32) -> i32 {
    num * num
}
```

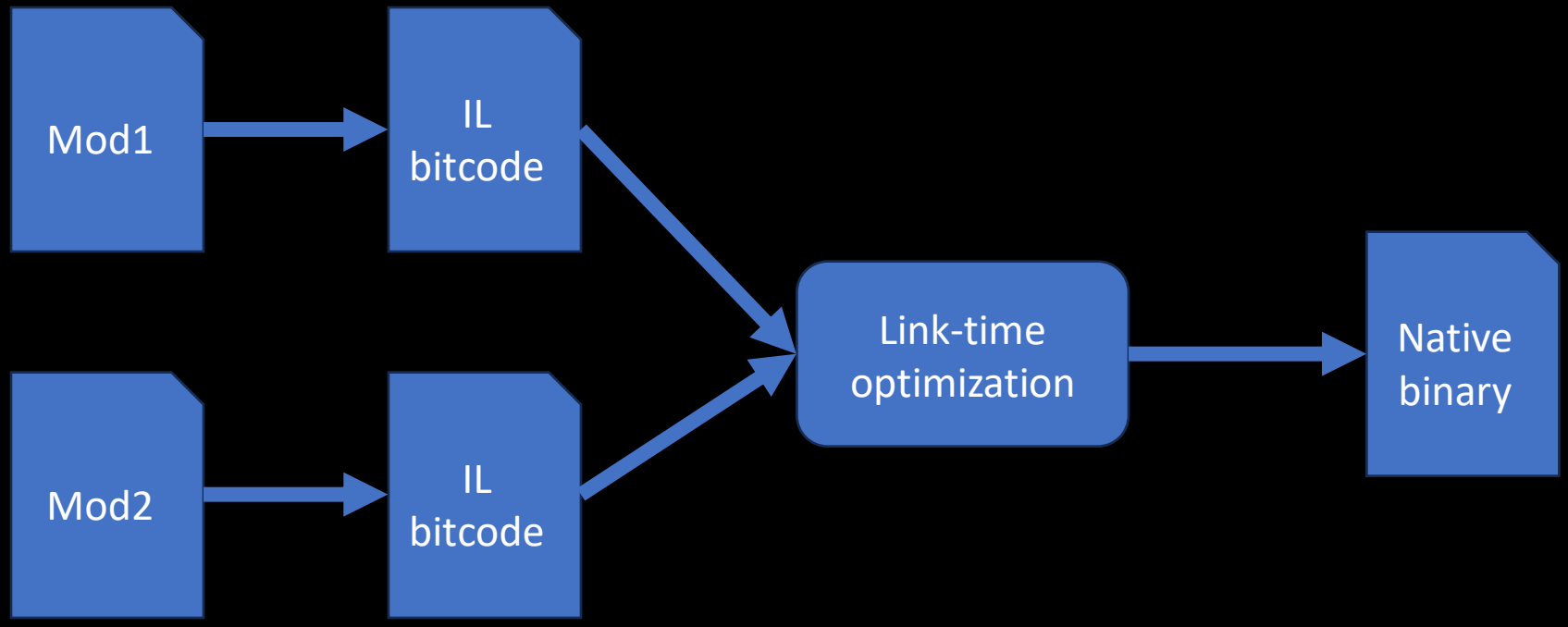
```
;just an empty textbox :D
```

The function above is not public, and not used by any other functions, thus the compiler removes it.

Compiler explorer

# Link-time optimization

- An inter-module level optimization, capable of optimizing the whole program
- Increases compilation time, as parallelization is problematic in this stage



# Companies that use LLVM





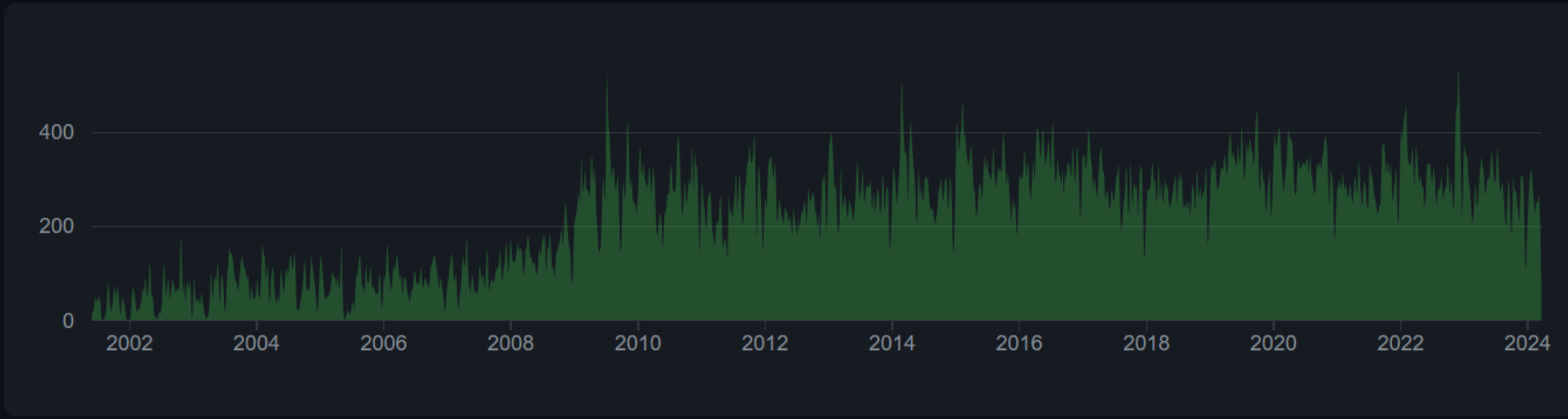


# Stats

493,156 Commits

Jun 3, 2001 – Mar 26, 2024

Contributions to main, line counts have been omitted because commit count exceeds 10,000.



Thank You!

# Sources, additional material

The Architecture of Open Source Applications (Volume 1)LLVM (aosabook.org)
LLVM IR and Go | Gopher Academy Blog
Using the New Pass Manager — LLVM 19.0.0git documentation
Compiling With Clang Optimization Flags – Incredibuild

LLVM IR:
LLVM Assembly Language Reference Manual (apple.com)

Videos:
LLVM in 100 Seconds (youtube.com)
(5) 2023 EuroLLVM - Tutorial: A whirlwind tour of the LLVM optimizer – YouTube