



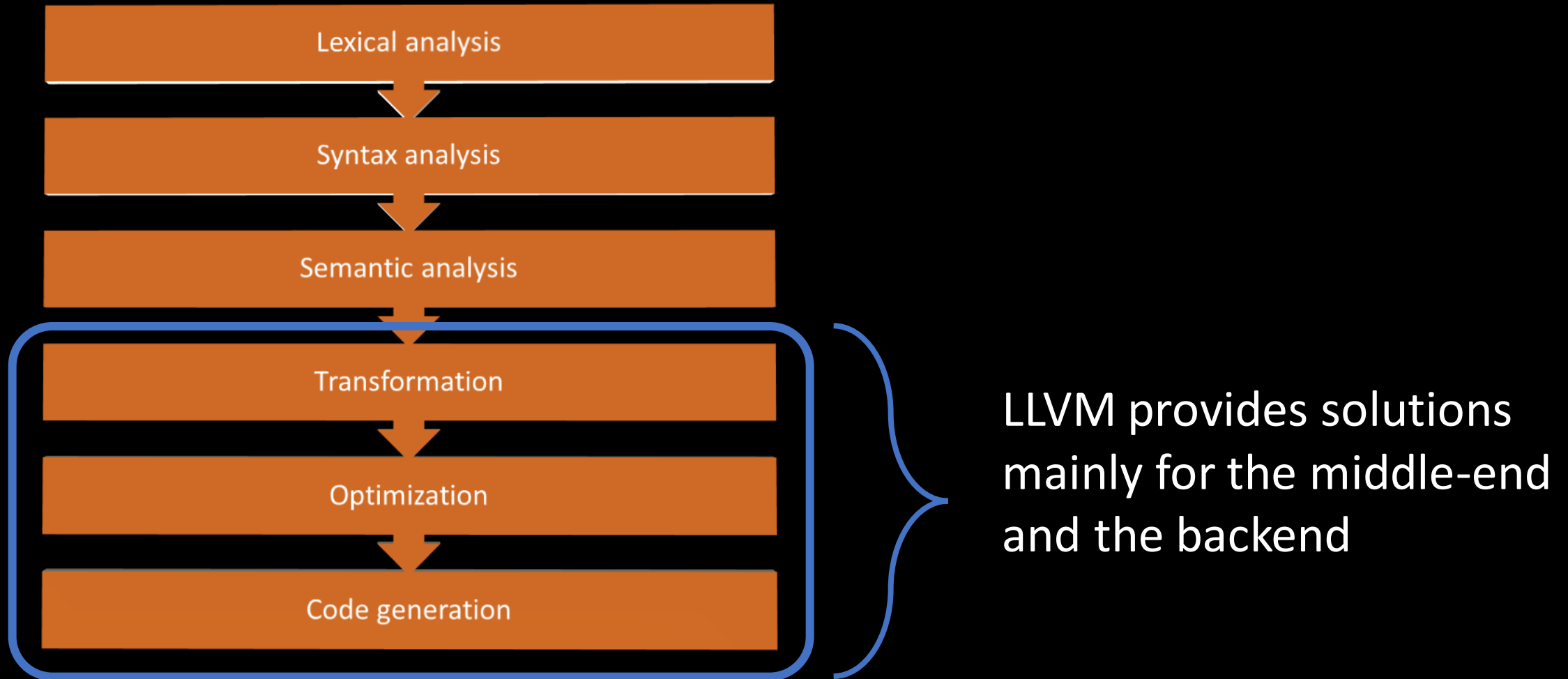
LLVM

Gembela Gergely

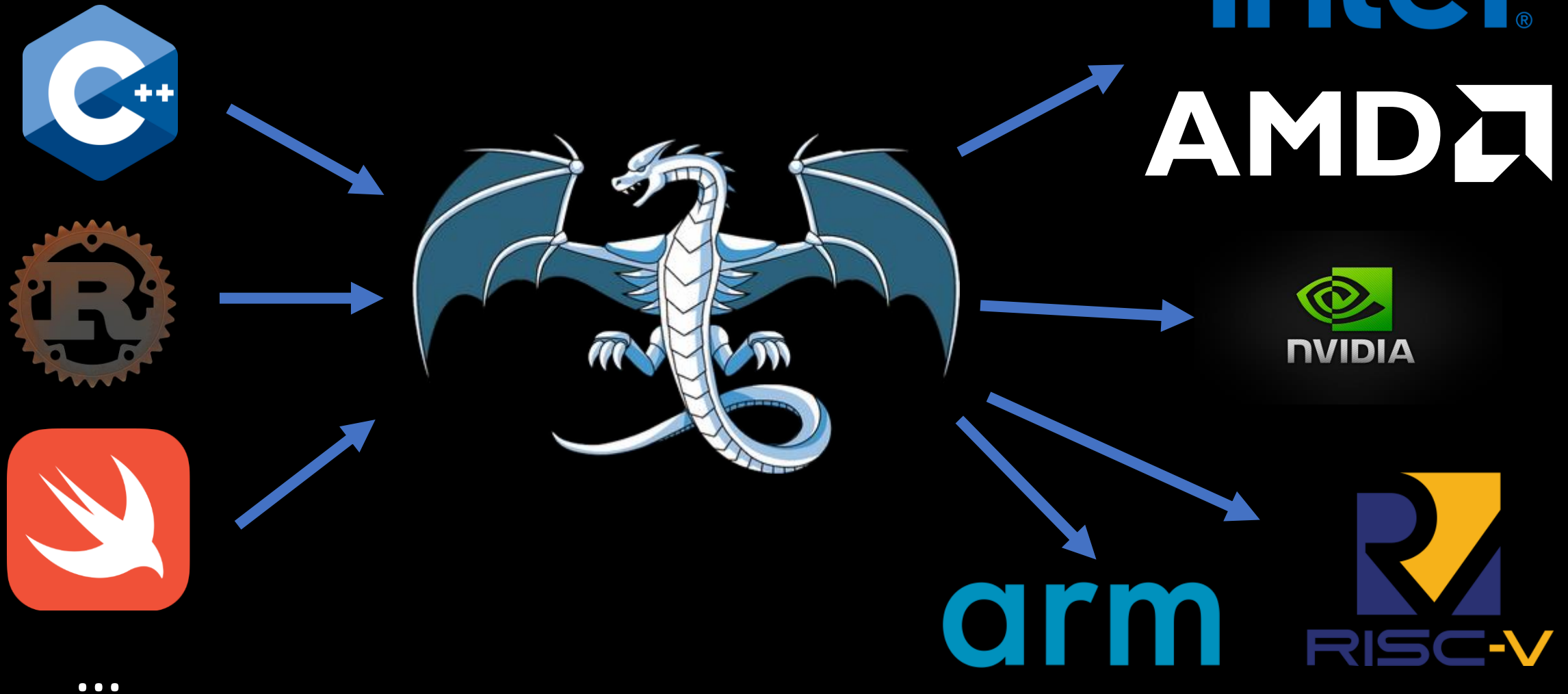
What is LLVM?

- 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



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



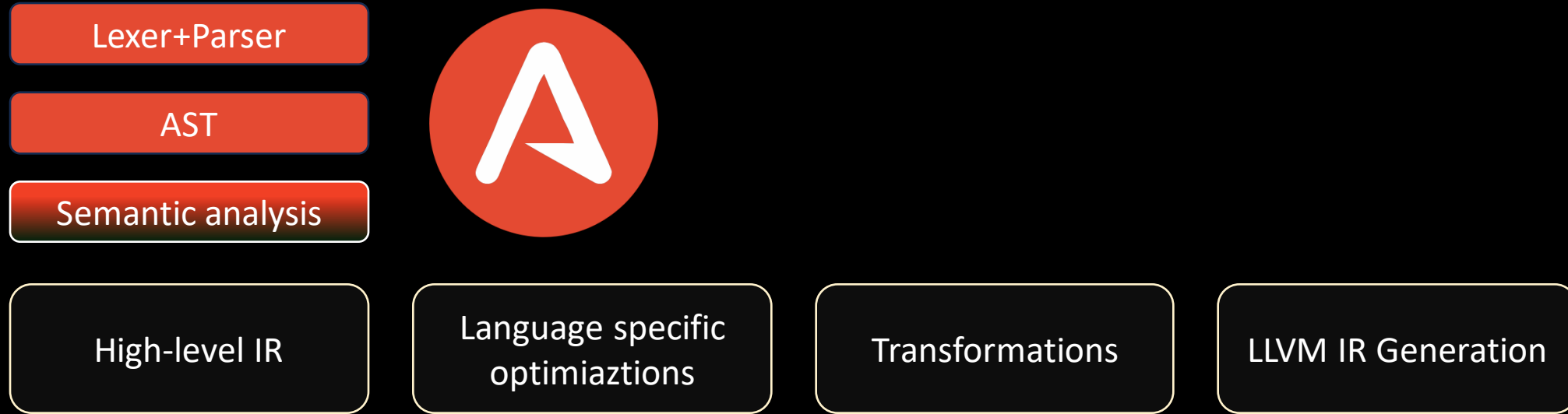
A modern LLVM-based compiler

Lexer+Parser

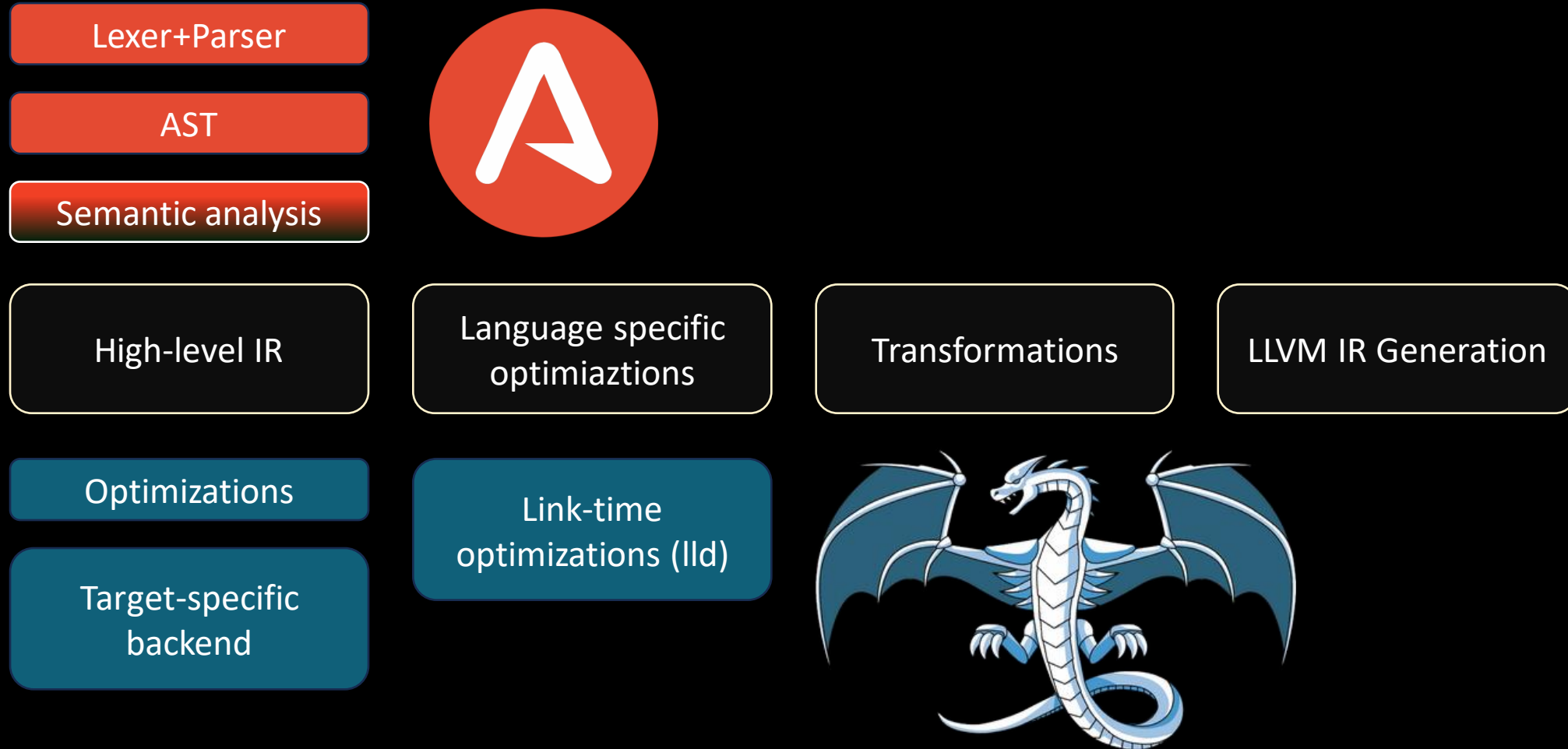
AST



A modern LLVM-based compiler

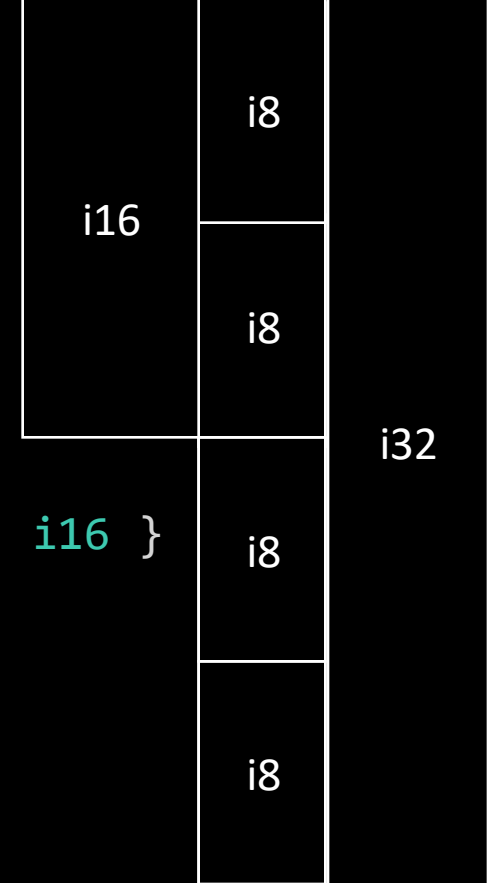


A modern LLVM-based compiler



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
```

@square – rs

```
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
```

@square – 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
```

@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

The image shows a code editor with two panes. The left pane displays C++ source code, and the right pane displays the corresponding LLVM IR. A menu is open in the right pane, highlighting the 'Opt Pipeline' option.

C++ source #1

```
1 #include <stdio.h>
2
3 inline long factorial(long n){
4     long res = 1;
5     for(long i = 2; i <= n; i++){
6         res *= i;
7     }
8     return res;
9 }
10
11 int main() {
12     volatile auto a = factorial(14);
13 }
```

armv8-a clang (trunk) (Editor #1)

armv8-a clang (trunk) -O2 -emit-llvm

- Clone Compiler
- Optimization
- Stack Usage
- Preprocessor
- AST
- LLVM IR
- Opt Pipeline**
- Device
- Control Flow Graph

ef i32 @main() local_unnamed_addr #0 !dbg

gn 8

.dbg.assign(metadata i1 undef, metadata !

time.start.p0(i64 8, ptr nonnull %a), !db

7178291200, ptr %a, align 8, !dbg !23

.dbg.assign(metadata i64 poison, metadata

time.end.p0(i64 8, ptr nonnull %a), !dbg

etime.start.p0(i64 immarg, ptr nocapture)

etime.end.p0(i64 immarg, ptr nocapture) #

declare void @llvm.dbg.assign(metadata, metadata, metadata, met

attributes #0 = { mustprogress norecurse nounwind memory

attributes #1 = { mustprogress nocallback norecurse nounwind

attributes #2 = { nocallback norecurse nounwind speculatabl

!20 = distinct !DIAssignID()

!28 = distinct !DIAssignID()

Inspecting the opt pipeline

Opt Pipeline Viewer armv8-a clang (trunk) (Editor #1, Compiler #1)

Function: main

Passes:

- AssignmentTrackingPass on [module]
- GlobalOptPass on [module]
- InstCombinePass on main
- InlinerPass on (main)
- SROAPass on main
- CorrelatedValuePropagationPass on main
- ReassociatePass on main
- LCSSAPass on main
- InstCombinePass on main
- LCSSAPass on main
- IndVarSimplifyPass on for.body.i
- LoopDeletionPass on for.body.i
- SROAPass on main
- GVNPass on main
- InstCombinePass on main
- ADCEPass on main
- SROAPass on main
- InstCombinePass on main

```
1  -define dso_local noundef i32 @main() {  
1+define dso_local noundef i32 @main() local_unnamed_addr {  
2  2 entry:  
3  3   %a = alloca i64, align 8  
4  4   #dbg_assign(i1 undef, !17, !DIExpression(), !20, ptr %a, !DIExpression(), !21)  
5  5   call void @llvm.lifetime.start.p0(i64 8, ptr %a)  
6  6   %call = call noundef i64 @factorial(long)(i64 noundef 14)  
7  7   store volatile i64 %call, ptr %a, align 8  
8  8   #dbg_assign(i64 %call, !17, !DIExpression(), !29, ptr %a, !DIExpression(), !21)  
9  9   call void @llvm.lifetime.end.p0(i64 8, ptr %a)  
10 10  ret i32 0  
11 11 }
```

Example: dead code elimination

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

Example: Mem2Reg (SROA+Mem2Reg)

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

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;  
}
```

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);  
}
```

Example: loop unroll

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


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  
}
```

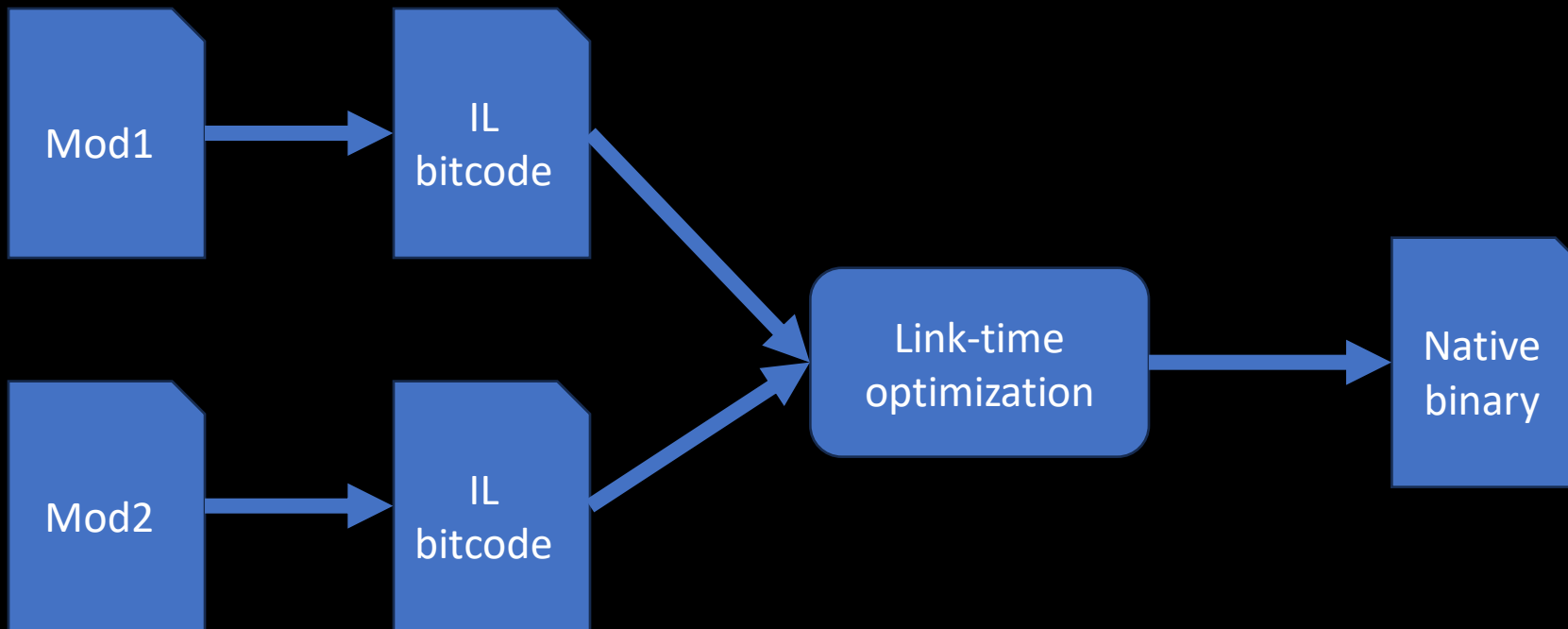


;just an empty textbox :D

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

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



493,156 Commits

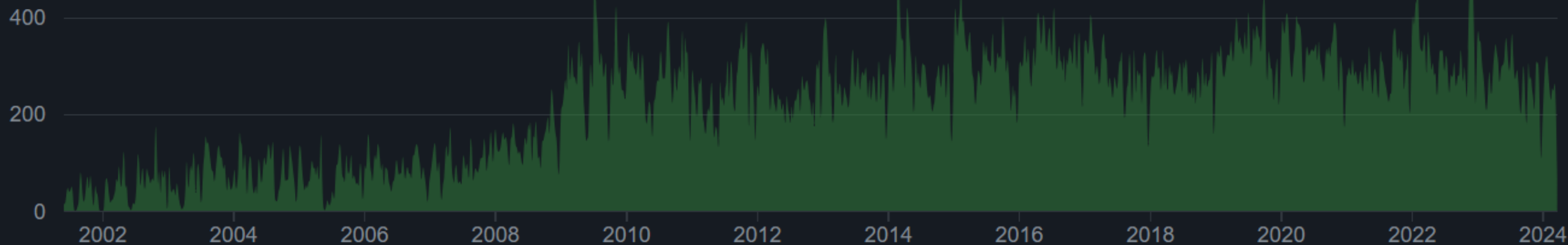
Stats



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](#)