Building LLVM-IR

Johannes Doerfert and Christoph Mallon

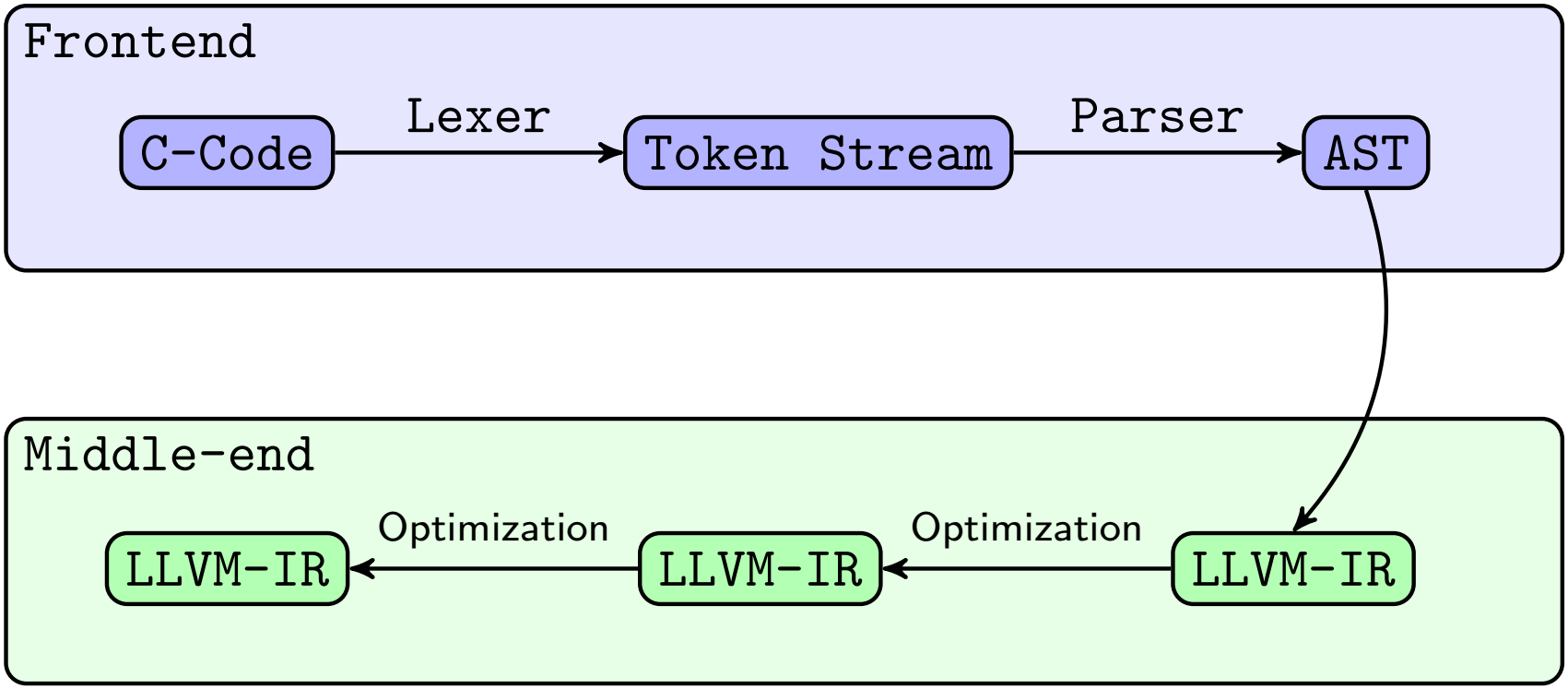
Saarbrücken Graduate School of Computer Science
Saarland University
Saarbrücken, Germany

December 18, 2013

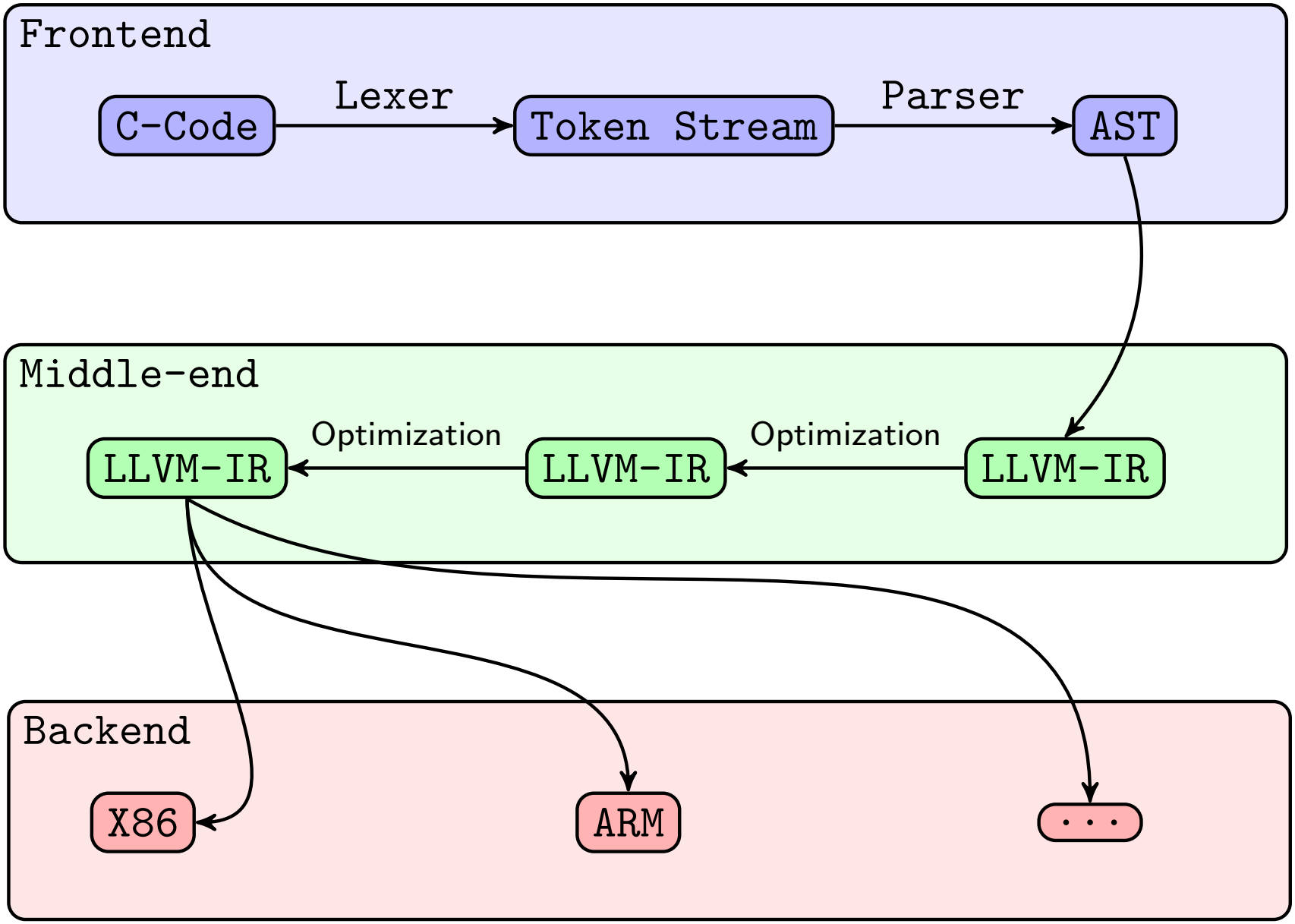
C4

C4

C4


Frontend
C-Code
Lexer
Token Stream
Parser
AST
Middle-end
LLVM-IR
Optimization
LLVM-IR
Optimization
LLVM-IR

C4


Frontend
C-Code
Lexer
Token Stream
Parser
AST
Middle-end
LLVM-IR
Optimization
LLVM-IR
Optimization
LLVM-IR
Backend
X86
ARM
...

LLVM-IR

LLVM-IR

- Low-level, “typed”, SSA-based, assembly language
- Types are sign agnostic

LLVM-IR

- Low-level, “typed”, SSA-based, assembly language
- Types are sign agnostic
- Hierarchical structured

LLVM-IR

- Low-level, “typed”, SSA-based, assembly language
- Types are sign agnostic
- Hierarchical structured
  - Module
    - Global Variables
    - Composite Types (“structs”)
    - Function Declarations
    - Function Definitions

LLVM-IR

- Low-level, “typed”, SSA-based, assembly language
- Types are sign agnostic
- Hierarchical structured
  - Module
    - Global Variables
    - Composite Types (“structs”)
    - Function Declarations
    - Function Definitions
  - Function Definition
    - Basic Blocks

LLVM-IR

- Low-level, “typed”, SSA-based, assembly language
- Types are sign agnostic
- Hierarchical structured
  - Module
    - Global Variables
    - Composite Types (“structs”)
    - Function Declarations
    - Function Definitions
  - Function Definition
    - Basic Blocks
  - Basic Blocks
    - Instructions

LLVM-IR

Instructions

LLVM-IR

Instructions

```
%sum = add i32 4, %var              ; Binary operations
%cmp = icmp sge i32 %a, %b
```

LLVM-IR

Instructions

```
%sum = add i32 4, %var              ; Binary operations
%cmp = icmp sge i32 %a, %b

%value = load i32* %location        ; Memory operations
store i32 %value, i32* %location
```

LLVM-IR

Instructions

```
%sum = add i32 4, %var                ; Binary operations
%cmp = icmp sge i32 %a, %b

%value = load i32* %location          ; Memory operations
store i32 %value, i32* %location

br label %next-block                  ; Terminator Instructions
br i1 %cmp, label %then-block, label %else-block
ret i32 %a
```

LLVM-IR

Instructions

```
%sum = add i32 4, %var                ; Binary operations
%cmp = icmp sge i32 %a, %b

%value = load i32* %location          ; Memory operations
store i32 %value, i32* %location

br label %next-block                  ; Terminator Instructions
br i1 %cmp, label %then-block, label %else-block
ret i32 %a

%X = trunc i32 257 to i8              ; Cast Instructions
%Y = sext i32 %V to i64
%Z = bitcast i8* %x to i32*
```

LLVM-IR

Instructions

```
%sum = add i32 4, %var                ; Binary operations
%cmp = icmp sge i32 %a, %b

%value = load i32* %location          ; Memory operations
store i32 %value, i32* %location

br label %next-block                  ; Terminator Instructions
br i1 %cmp, label %then-block, label %else-block
ret i32 %a

%X = trunc i32 257 to i8              ; Cast Instructions
%Y = sext i32 %V to i64
%Z = bitcast i8* %x to i32*

%ptr = alloca i32                     ; Other Instructions
%ret = call i32 @foo(i8* %fmt, i32 %val)
%phi = phi i32 [ %value-a, %block-a ], [ %value-b, %block-b ]
%I-th-element-addr = getelementptr i8** %Array, i64 %I
```

LLVM-IR

Basic Blocks

LLVM-IR

Basic Blocks

```
int max(int a, int b) {
  if (b < a) {
    return a;
  } else {
    return b;
  }
}
```

LLVM-IR

Basic Blocks

```
if-header:
  %A = load i32* %Aptr
  %B = load i32* %Bptr
  %if-condition = icmp slt i32 %B, %A
  br i1 %if-condition, label %if-cons, label %if-alt

if-cons:                                          ; preds = %if-header
  %A2 = load i32* %Aptr
  ret i32 %A2

if-alt:                                           ; preds = %if-header
  %B2 = load i32* %Bptr
  ret i32 %B2
```

LLVM-IR

Functions

LLVM-IR

Functions

```
define i32 @max(i32 %a, i32 %b) {
entry:
  %0 = alloca i32
  %1 = alloca i32
  store i32 %a, i32* %1
  store i32 %b, i32* %0
  br label %if-header

if-header:                                        ; preds = %entry
  %2 = load i32* %0
  %3 = load i32* %1
  %if-condition = icmp slt i32 %2, %3
  br i1 %if-condition, label %if-cons, label %if-alt

if-con:                                           ; preds = %if-header
  %4 = load i32* %1
  ret i32 %4

if-alt:                                           ; preds = %if-header
  %5 = load i32* %0
  ret i32 %5
}
```

LLVM-IR

The human readable form

- Globals start with an “@”
- Local values start with an “%”
- Basic block names start with an “%” when used
- Basic block names end with an “:” when definied
- There is always “`label`” written before the branch target
- Indention is not important (but nice)

LLVM-IR

Ill-formed Examples

LLVM-IR

Ill-formed Examples

```
; Declaration needs to strictly dominate use
%x = add i32 1, %x
```

LLVM-IR

Ill-formed Examples

```
; Declaration needs to strictly dominate use
%x = add i32 1, %x


; Binary operations need equal types
%valI32 = load i32 *P
%valI64 = load i64 *Q
%val = add i32 %valI32, %valI64
```

LLVM-IR

Ill-formed Examples

```
; Declaration needs to strictly dominate use
%x = add i32 1, %x


; Binary operations need equal types
%valI32 = load i32 *P
%valI64 = load i64 *Q
%val = add i32 %valI32, %valI64


; A basic block *needs* a terminator (branch or return)
block:
  %VAL = add i32 %A, %B
```

LLVM-IR

Ill-formed Examples

```
; Declaration needs to strictly dominate use
%x = add i32 1, %x
```

```
; Binary operations need equal types
%valI32 = load i32 *P
%valI64 = load i64 *Q
%val = add i32 %valI32, %valI64
```

```
; A basic block *needs* a terminator (branch or return)
block:
  %VAL = add i32 %A, %B
```

```
; A terminator *ends* a basic block
block:
  ret i32 %A
  %VAL = add i32 %A, %B
```

LLVM-IR

Command line

Generate (human readable) LLVM-IR from C/C++ input:

```
clang -emit-llvm -c -S -o OUT.ll IN.c
```

Apply an optimization on LLVM-IR:

```
opt -S -o OUT.ll IN.ll
```

Execute (via JIT) an LLVM-IR module:

```
lli IN.ll <argv arguments>
```

Create a binary from an LLVM-IR module:

```
clang -o OUT IN.ll
```

Create architecture specific assembly:

```
llc -o OUT.s IN.ll
```

Create a binary from architecture specific assembly:

```
cc -o OUT IN.s
```

Create API calls for an LLVM-IR module:

```
llc -march=cpp -o OUT.cpp IN.ll
```

Get more help:

```
<TOOL> --help
```

LLVM-IR

Documentation

General language reference manual:

`http://llvm.org/docs/LangRef.html`

Doxygen code documentation (auto generated):

`http://llvm.org/docs/doxygen/html/index.html`

Stable binaries and source code (llvm + clang):

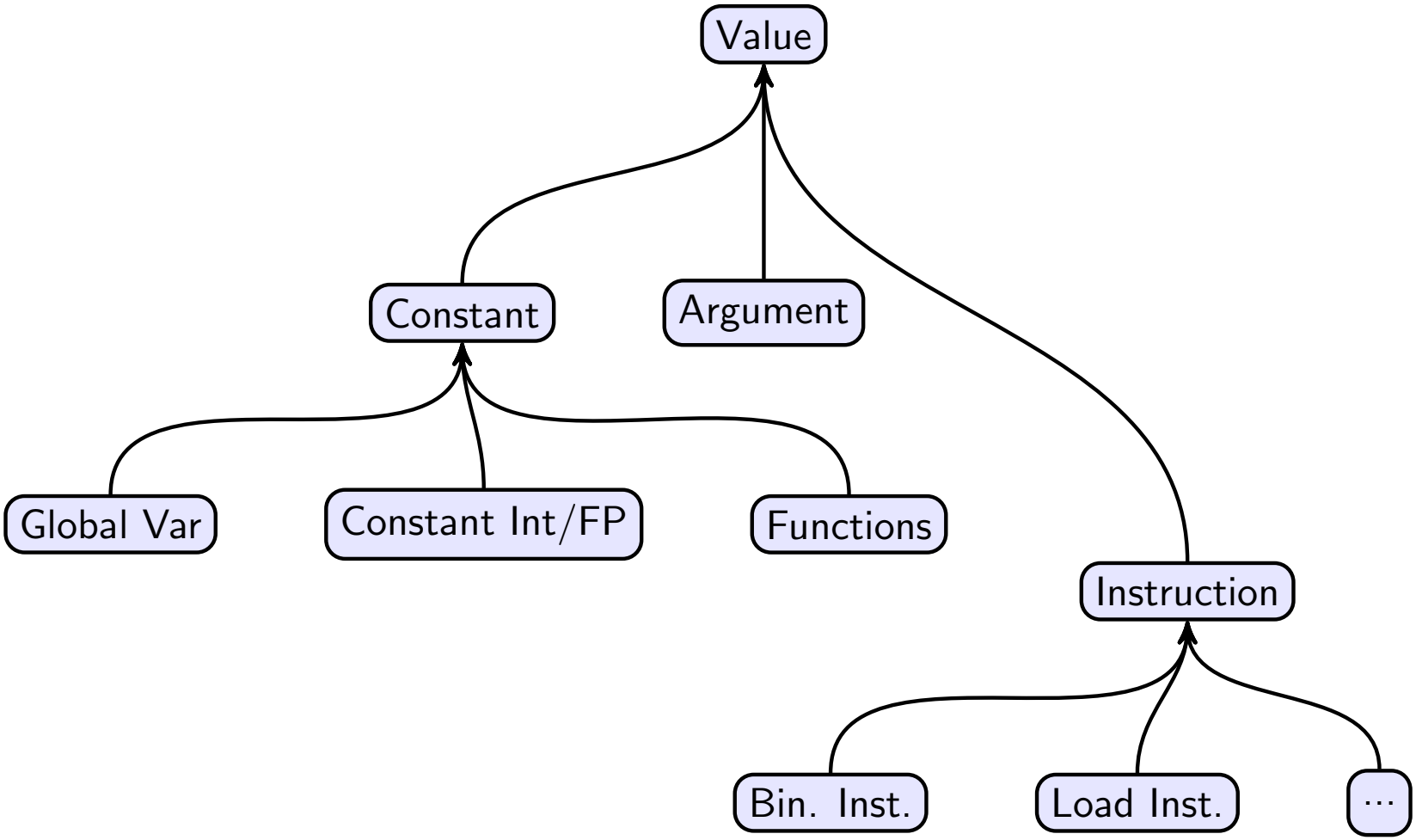
`http://llvm.org/releases/download.html#3.3`

Full command line tools guide:

`http://llvm.org/docs/CommandGuide/`

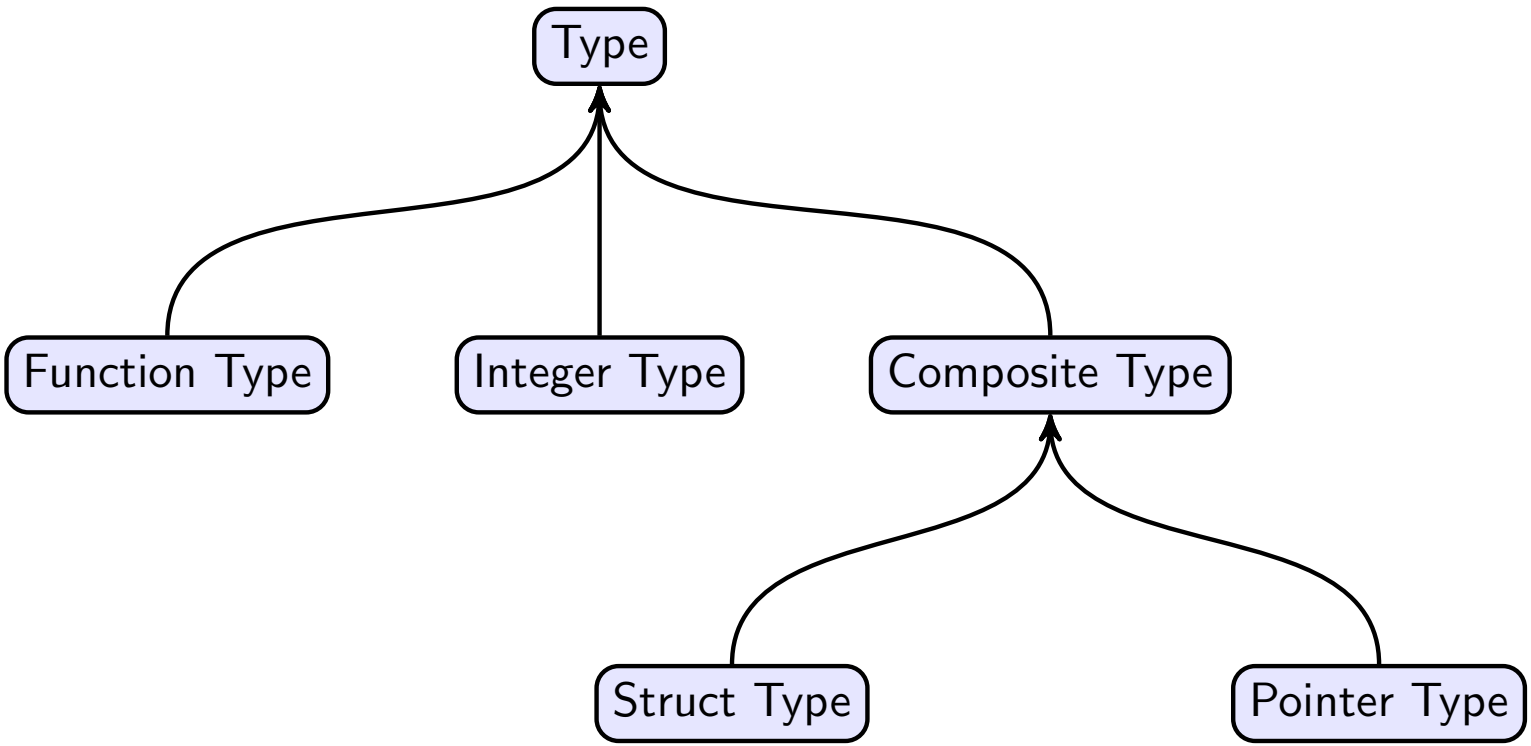
LLVM-IR

API classes



LLVM-IR

API classes



LLVM

Final Notes

LLVM

Final Notes

- LLVM is HUGE
- LLVM is grown, the interface is **not** consitent

LLVM

Final Notes

- LLVM is HUGE
- LLVM is grown, the interface is **not** consitent

Keep it Simple and Straightforward

Doxygen + Examples

```
int max(int a, int b) {
  if (b < a) {
    return a;
  } else {
    return b;
  }
}
```

```
; ModuleID = 'max.c'
```

```
; ModuleID = 'max.c'
@X = global i32 0
```

```
; ModuleID = 'max.c'
@X = global i32 0
define i32 @max(i32 %a, i32 %b) #0 {



}
```

```
; ModuleID = 'max.c'
@X = global i32 0
define i32 @max(i32 %a, i32 %b) #0 {



}
define i32 @main(i32 %argc, i8** %argv) #0 {



}
```

```
; ModuleID = 'max.c'
@X = global i32 0
define i32 @max(i32 %a, i32 %b) #0 {


}
define i32 @main(i32 %argc, i8** %argv) #0 {


}
attributes #0 = { nounwind uwtable ...
```

```
; ModuleID = 'max.ll'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64"
target triple = "x86_64-unknown-linux-gnu"

@X = global i32 0

; Function Attrs: nounwind uwtable
define i32 @max(i32 %a, i32 %b) #0 {
  %1 = icmp slt i32 %b, %a
  br i1 %1, label %2, label %3

; <label>:2                                       ; preds = %0
  ret i32 %a

; <label>:3                                       ; preds = %0
  ret i32 %b
}

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc, i8** %argv) #0 {
  %1 = load i32* @X, align 4
  %2 = call i32 @max(i32 %argc, i32 %1)
  ret i32 %2
}

attributes #0 = { nounwind uwtable "no-frame-pointer-elim"="true" }
```