ing Down the Compilation Rabbit Hole

GOLAB 2023

Richard Rowland

whoami

Richard Rowland

`Place:` Tokyo, Japan

`Job:` Staff at OKCoinJapan, a global cryptocurrency exchange

`Hobby:` Photo Nerd! 📸
(Follow my Insta / Share photo spots in Florence)

`Projects:` Created a method to semi-permanently store data in blockchain and P2P network (on GitHub)

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Previous Talks on Python Compiler

- Presented on Python Compiler at Python conferences
- Became curious about other languages's compilers

PyCascades 2023
https://www.youtube.com/watch?v=RcGshw0tzoc

PyCon Ireland 2022
https://www.youtube.com/watch?v=u6XDRzLX6Eo

What is Compiling?

Compiling

Turning source code into executable format

Compilers

Programs used to compile programs

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Famous Compilers

gcc

llvm

Compiling in Golang

```
> go build main.go
```

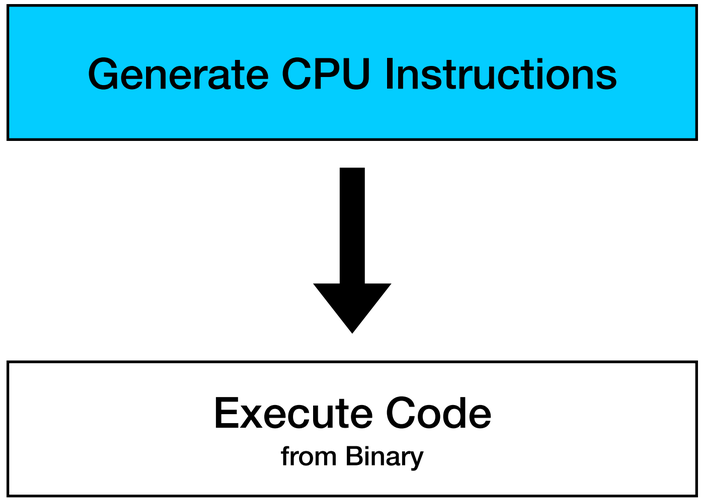
About the Golang Compiler

- Golang project develops its own compiler (doesn't use GCC or LLVM)
- Source Code: https://go.googlesource.com/go
- Compiler code is stored under: `src/cmd/compile`

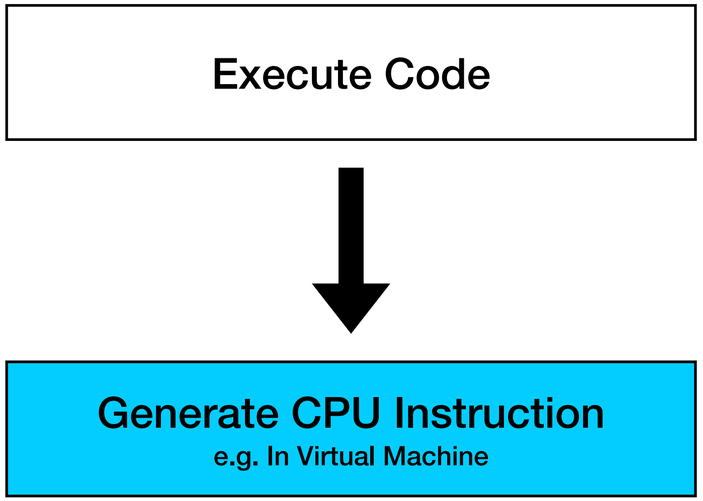
Programming Language Execution Models

Different models of program execution

Compiled Languages



Interpreted Langauges



Compiled vs Interpreted

Examples of Programming Languages

Compiled Languages

- **Golang**
- C, C++
- Rust
- Swift

Interpreted Langauges

- Javascript
- Python
- Ruby

Compiled vs Interpreted

Pros and Cons

	Compiled	Interpreted
Pros	• Fast execution	• No need to create binary for each platform (“Write Once, Run Everywhere”)
Cons	• Takes time to compile • Need to compile for each platform	• Execution tends to be slower • CPU/Memory Resource Overhead of Virtual Machine

Comparing C and Golang Binary

Binary Size - C vs Go

C

```c
#include <stdio.h>

int main(){
  printf("Hello, World!\n");
  return 0;
}
```

Binary Size: 33 KB

Go

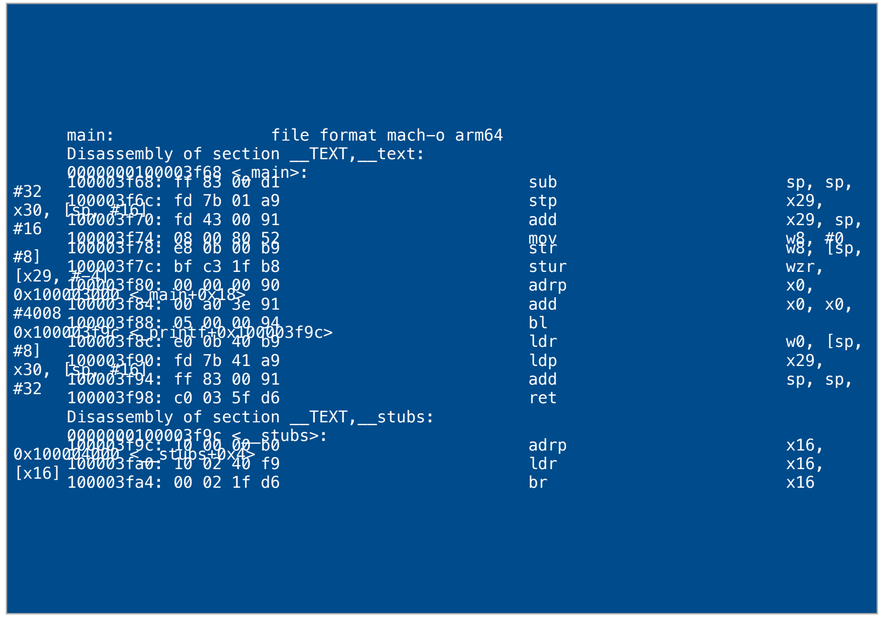
```go
package main
import "fmt"

func main() {
  fmt.Println("Hello, World!")
}
```

Binary Size: 2 MB

Objdump Assembly - C vs Go

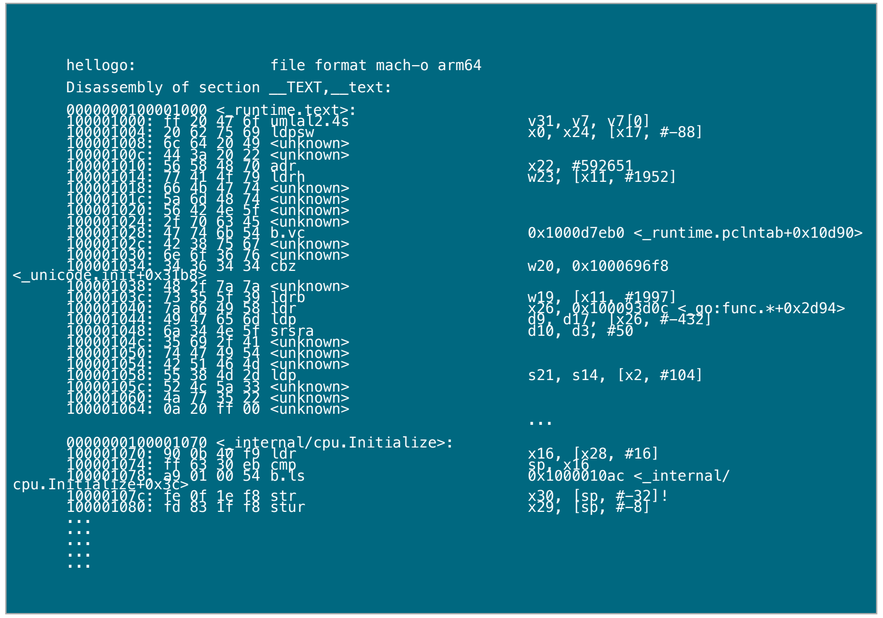
C



```
main:               file format mach-o arm64
Disassembly of section __TEXT,__text:
0000000100003f68 <_main>:
100003f68: ff 83 00 d1                          sub     sp, sp,
#32
100003f6c: fd 7b 01 a9                          stp     x29,
x30, [sp, #16]
100003f70: fd 43 00 91                          add     x29, sp,
#16
100003f74: 08 00 80 52                          mov     w8, #0
100003f78: e8 0b 00 b9                          str     w8, [sp,
#8]
100003f7c: bf c3 1f b8                          stur    wzr,
[x29, #-4]
100003f80: 00 00 00 90                          adrp    x0,
0x100003000 <_main+0x18>
100003f84: 00 a0 3e 91                          add     x0, x0,
#4008
100003f88: 05 00 00 94                          bl
0x100003f9c <_printf+0x100003f9c>
100003f8c: e0 0b 40 b9                          ldr     w0, [sp,
#8]
100003f90: fd 7b 41 a9                          ldp     x29,
x30, [sp, #16]
100003f94: ff 83 00 91                          add     sp, sp,
#32
100003f98: c0 03 5f d6                          ret
Disassembly of section __TEXT,__stubs:
0000000100003f9c <_stubs>:
100003f9c: 10 00 00 b0                          adrp    x16,
0x100004000 <_stubs+0x4>
100003fa0: 10 02 40 f9                          ldr     x16,
[x16]
100003fa4: 00 02 1f d6                          br      x16
```

Objdump: 22 Lines

Go



```
hellogo:              file format mach-o arm64
Disassembly of section __TEXT,__text:
0000000100001000 <_runtime.text>:
100001000: ff 20 47 6f umlal2.4s                v31, v7, v7[0]
100001004: 20 62 75 69 ldpsw                    x0, x24, [x17, #-88]
100001008: 6c 64 20 20 <unknown>
10000100c: 44 3a 20 22 <unknown>
100001010: 56 58 48 20 adr                      x22, #592651
100001014: 77 41 4f 79 ldrh                     w23, [x11, #1952]
100001018: 66 4b 47 74 <unknown>
10000101c: 53 6d 28 74 <unknown>
100001020: 59 42 4e 5f <unknown>
100001024: 2f 70 63 45 <unknown>
100001028: 47 74 6b 54 b.vc                     0x1000d7eb0 <_runtime.pclntab+0x10d90>
10000102c: 42 38 75 67 <unknown>
100001030: 6e 6f 36 76 <unknown>
100001034: 60 6f 34 34 cbz                      w20, 0x1000696f8
<_unicode.init+0x3fb8>
100001038: 48 2f 7a 7a <unknown>
10000103c: 73 55 3f 30 ldrb                     w19, [x11, #1997]
100001040: 7a 6b 4b 58 ldr                      x26, 0x100003dac <_go:func.*+0x2d94>
100001044: 49 47 65 6d ldp                      d9, d17, [x26, #-432]
100001048: 6a 34 4e 5f srsra                    d10, d3, #50
10000104c: 37 69 2f 41 <unknown>
100001050: 74 47 49 54 <unknown>
100001054: 42 51 46 4d <unknown>
100001058: 55 38 4d 2d ldp                      s21, s14, [x2, #104]
10000105c: 52 4c 5a 33 <unknown>
100001060: 4a 77 35 22 <unknown>
100001064: 0a 20 ff 00 <unknown>
                                                ...
0000000100001070 <_internal/cpu.Initialize>:
100001070: 90 0b 40 f9 ldr                      x16, [x28, #16]
100001074: f1 53 30 eb cmp                      sp, x16
100001078: a9 01 00 54 b.ls
0x1000010ac <_internal/
cpu.Initialize+0x3c>
10000107c: fe 0f 1e f8 str                      x30, [sp, #-32]!
100001080: fd 83 1f f8 stur                     x29, [sp, #-8]
      ...
      ...
      ...
      ...
      ...
```

Objdump: 140k Lines

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Syscalls - C vs Go

C

```
execve("./main", ["./main"], 0xffffcd34e2c0 /* 4 vars */) = 0
set_tid_address(0xffff9cc74230)         = 8
brk(NULL)                               = 0xaaaad0305000
brk(0xaaaad0307000)                     = 0xaaaad0307000
mmap(0xaaaad0305000, 4096, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0xaaaad0305000
mprotect(0xaaaac4cff000, 4096, PROT_READ) = 0
ioctl(1, TIOCGWINSZ, 0xfffff28fad48)    = -1 ENOTTY (Not a tty)
writev(1, [{iov_base="Hello, World!", iov_len=13}, {iov_base="\n", iov_len=1}], 2)
= 14
exit_group(0)                           = ?
+++ exited with 0 +++
```

Syscalls: 9 times

Go

```
execve("./main", ["./main"], 0xffffcd5741c0 /* 8 vars */) = 0
sched_getaffinity(0, 8192, [0, 1])      = 8
openat(AT_FDCWD, "/sys/kernel/mm/transparent_hugepage/hpage_pmd_size", O_RDONLY) = 3
read(3, "2097152\n", 20)                = 8
close(3)                                = 0
mmap(NULL, 262144, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff953ed000
mmap(NULL, 131072, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff953cd000
mmap(NULL, 1048576, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff952cd000
mmap(NULL, 8388608, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff94acd000
mmap(NULL, 67108864, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff90acd000
mmap(NULL, 536870912, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff70acd000
mmap(NULL, 8388608, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff702cd000
mmap(0x4000000000, 67108864, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4000000000
mmap(NULL, 33554432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff6e2cd000
mmap(NULL, 1133584, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff6e1b8000
mmap(0x4000000000, 4194304, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4000000000
mmap(0xffff953cd000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff953cd000
mmap(0xffff952cd000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff952cd000
mmap(0xffff94acf000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff94acf000
mmap(0xffff90add000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff90add000
mmap(0xffff70b4d000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff70b4d000
mmap(0xffff702cd000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff702cd000
mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff6e0b8000
mmap(NULL, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff6e0a8000
mmap(NULL, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff6e098000
rt_sigprocmask(SIG_SETMASK, NULL, [], 8) = 0
sigaltstack(NULL, {ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=0}) = 0
sigaltstack({ss_sp=0x4000004000, ss_flags=0, ss_size=32768}, NULL) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
gettid()                                = 453
rt_sigaction(SIGHUP, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGHUP, {sa_handler=0x6c5f0, sa_mask=~[], sa_flags=SA_ONSTACK|SA_RESTART|SA_SIGINFO}, NULL, 8) = 0
rt_sigaction(SIGINT, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGINT, {sa_handler=0x6c5f0, sa_mask=~[], sa_flags=SA_ONSTACK|SA_RESTART|SA_SIGINFO}, NULL, 8) = 0
rt_sigaction(SIGQUIT, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
...
...
...
```

Syscalls: 209 times

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Why does Go make many syscalls?

- Overhead of Golang runtime
- Go functions enabled by runtime
  - Garbage Collection
  - Goroutine scheduler
  - Channel operations

Golang Compiler(s)

What is Golang?

A programming language specification

Golang program execution environment

What is Golang?

A programming language specification

Programmer's Concern

Golang program execution environment

Golang's Concern

Golang Specification

https://go.dev/ref/spec

Golang Specification

Rules to Follow

Notation

Source code representation

Constants

Variables

Properties of types and values

Blocks

Declarations and scope

Expressions

Statements

Built-in functions

Packages

Program initialization and execution

Errors

Run-time panics

System considerations

Type unification rules

If you follow the specs, you can build your own compiler!

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

If you follow the specs, you can build your own compiler!

(You have to maintain it too.)

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Different Golang Compilers

gc (go compiler)

gcc (gccgo)

llvm (llgo)

Different Golang Compilers

Caveats

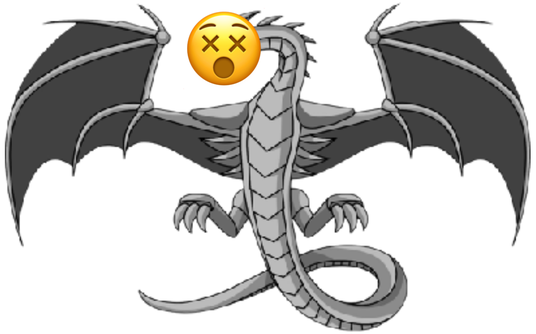
gc (go compiler)

Latest Version

gcc (gccgo)

Up to Go ver. 1.18

No generics support [1]



llvm (llgo)

Dropped from repo in 2020

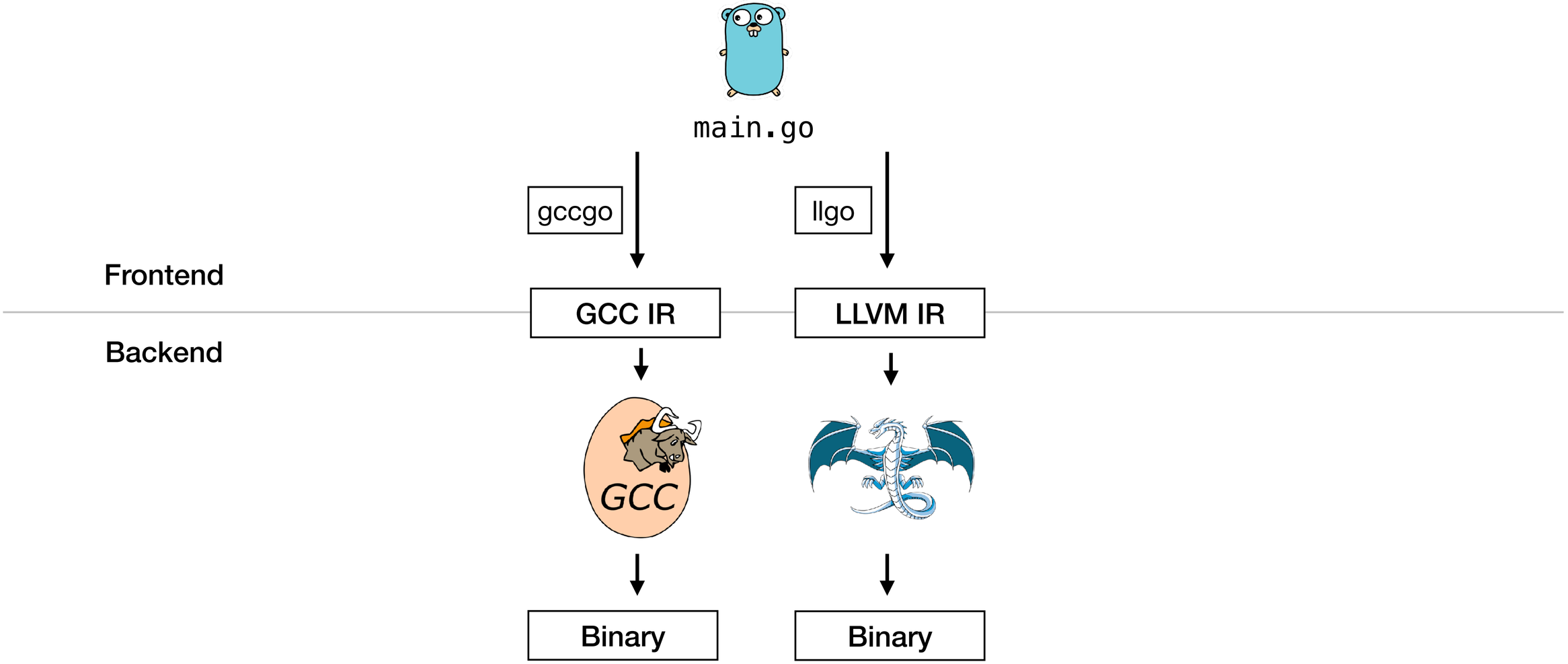
Demonstrates difficulty of maintaining

Mentioned for illustrative purposes only

[1] https://go.dev/doc/install/gccgo

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Using Different Compiler Backends



Why Use Other Compiler Backends?

When to Use

- Using Optimizations not implemented in gc (standard impl.)
- Targeting platforms not supported by gc
- Dynamic Linking of Libraries
- Making sure Implementation doesn’t become Specification

How Golang gets compiled

Different Golang Compilers

We'll talk about the official compiler

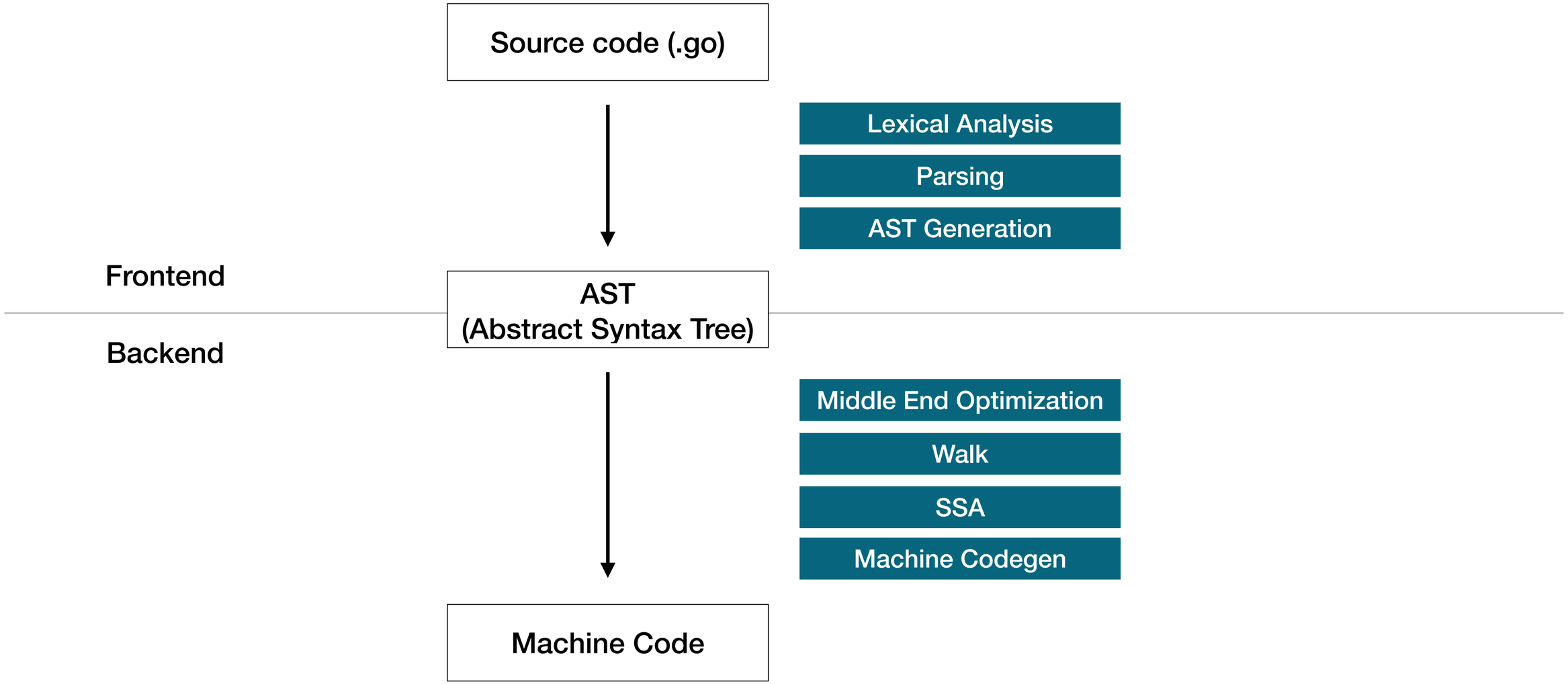
gc (go compiler)

~~gcc (gccgo)~~

~~llvm (llgo)~~

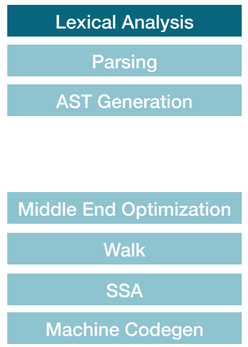
Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Golang Compiler Overview



Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Tokenization (Lexical Analysis)



- Groups characters into tokens (unit of meaning)
- Takes place in:
  - `src/cmd/compile/internal/syntax/tokens.go (List of tokens)`
  - `src/cmd/compile/internal/syntax/tokenizer.go (Turns code in tokens)`
- List of Tokens

```
_EOF
_Name
_Literal
_Operator
_AssignOp
_IncOp
_Assign
_Define
_Arrow
_Star
_Lparen
_Lbrack
_Lbrace
_Rparen
_Rbrack
_Rbrace
_Comma
_Semi
_Colon
_Dot
_DotDotDot
_Break
_Case
_Chan
_Const
_Continue
_Default
_Defer
_Else
_Fallthrough
_For
_Func
_Go
_Goto
_If
_Import
_Interface
_Map
_Package
_Range
_Return
_Select
_Struct
_Switch
_Type
_Var
tokenCount
Break
Continue
Fallthrough
Goto
Go
Defer
Def
Not
Recv
Tilde
OrOr
AndAnd
Eql
Neq
Lss
Leq
Gtr
Geq
Add
Sub
Or
Xor
Mul
Div
Rem
And
AndNot
Shl
Shr
```

main.go

```
package main
import "fmt"

func main() {
  fmt.Println("Hello, World!")
}
```

main.go tokenized

```
Token: PACKAGE, Literal: package
Token: IDENT, Literal: main
Token: IMPORT, Literal: import
Token: STRING, Literal: "fmt"
Token: FUNC, Literal: func
Token: IDENT, Literal: main
Token: LPAREN, Literal: (
Token: RPAREN, Literal: )
Token: LBRACE, Literal: {
Token: IDENT, Literal: fmt
Token: PERIOD, Literal: .
Token: IDENT, Literal: Println
Token: LPAREN, Literal: (
Token: STRING, Literal: "Hello, World!"
Token: RPAREN, Literal: )
Token: RBRACE, Literal: }
Token: EOF, Literal:
```

Tokenized using “`scanner`” pkg

Parsing (AST Generation)

- Takes place in:
  - `src/cmd/compile/internal/syntax/parser.go (Parsing)`
  - `src/cmd/compile/internal/syntax/nodes.go (Noder)`
- Takes tokens and builds AST (Abstract Syntax Tree)

Parsing (AST Generation)

What is an AST?

- AST is representation of source file in tree form
- Nodes correspond to elements
  - expressions
  - declarations
  - statements

Parsing (AST Generation)

Example of an AST

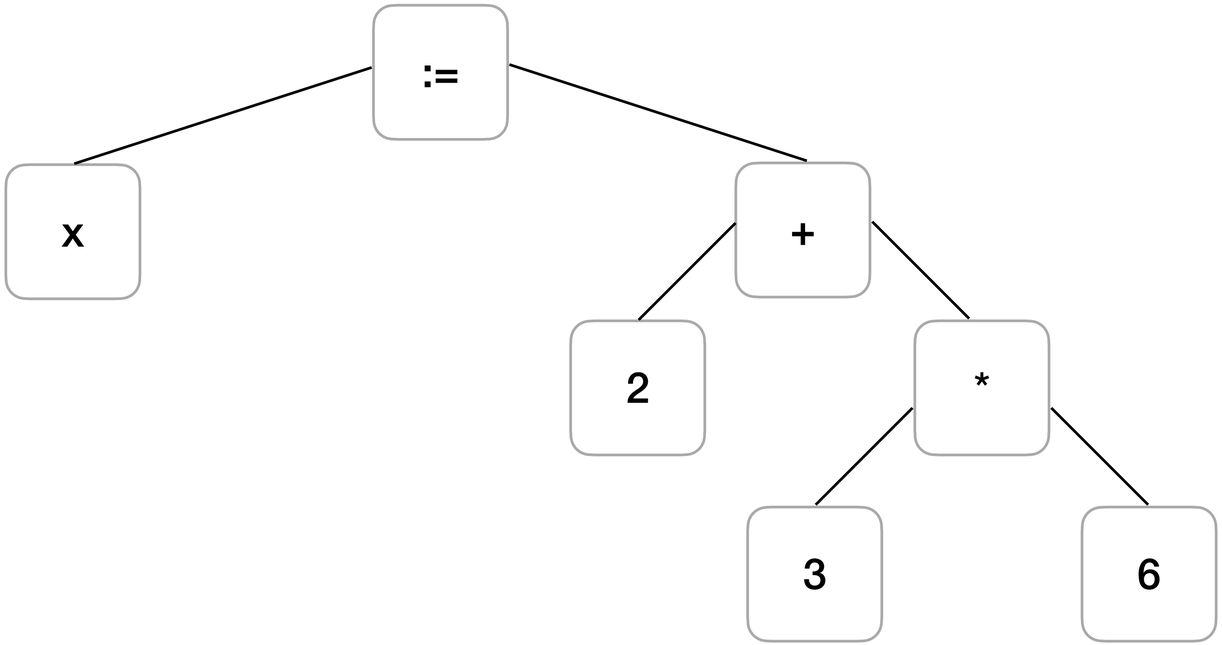
```
x := 2 + 3 * 6
```

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Parsing (AST Generation)

Example of an AST

```
x := 2 + 3 * 6
```



Parsing (AST Generation)

“Hello World” in AST

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

main.go

```
fmt.Println("Hello, World!")
```

main.go AST

```
Body: *ast.BlockStmt {
    Lbrace: 4:6
    List: []ast.Stmt {
        *ast.ExprStmt {
            X: *ast.CallExpr {
                Fun: *ast.SelectorExpr {
                    X: *ast.Ident {
                        NamePos: 5:5
                        Name: "fmt"
                    }
                    Sel: *ast.Ident {
                        NamePos: 5:9
                        Name: "Println"
                    }
                }
                Lparen: 5:15
                Args: []ast.Expr {
                    *ast.BasicLit {
                        ValuePos: 5:16
                        Kind: STRING
                        Value: "\"Hello, World!\""
                    }
                }
                Ellipsis: 0:0
                Rparen: 5:30
            }
        }
    }
    Rbrace: 6:1
}
```

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Middle End Optimization

- Perform optimization “passes”
- Takes place in:
  - `src/cmd/compile/internal/deadcode (Dead code elimination)`
  - `src/cmd/compile/internal/inline (Function call inlining)`
  - `src/cmd/compile/internal/devirtualize (Devirtualize known inferface method call)`
  - `src/cmd/compile/internal/escape (Escape analysis)`

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

Middle End Optimization

Examples of Optimizations

1. Inlining
2. Escape Analysis

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

Middle End Optimization

1. Inlining - Highlighting Optimizations

- Embed called functions into caller functions
- Saves overhead of starting new stack with new function call
- Budget of 80 nodes (If called function exceeds 80 nodes, not inlined)

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

Middle End Optimization

2. Escape Analysis - Highlighting Optimizations

- “Escape” variables from stack territory to heap territory
  - Some variables “outlive” current stack frame
- Makes variables accessible even after function disappear

Walk

- Takes place in: `cmd/compile/internal/walk`
- Decompose complex statements
- Desugar high-level Go construct
  - Examples
    - switch statement ➡️ binary search or jump tables
    - map and channel ➡️ runtime calls

SSA Conversion

- SSA (Single-Static Assignment)
- Takes place in:
  - `src/cmd/compile/internal/ssagen (Convert IR to SSA)`
  - `src/cmd/compile/internal/ssa (SSA passes and rules)`

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

SSA Conversion

What is SSA (Single Static Assignment) ?

- Lower-level IR
- All Variables changed into Immutable Variables (hence “Single Static”)
- Easier to optimize code in SSA form
- Introduced in Go 1.7 (`amd64`) and 1.8 (`all other archs`)
- Technique also used by GCC and LLVM

SSA Conversion

Example of SSA (Single Static Assignment) Conversion


```
//before
a := 1
a = 2
b = a + 1
```

rename vars

```
//after
a1 := 1
a2 := 2
b2 = a2 + 1
```

a is assigned value *twice*

a1 is assigned value *once*

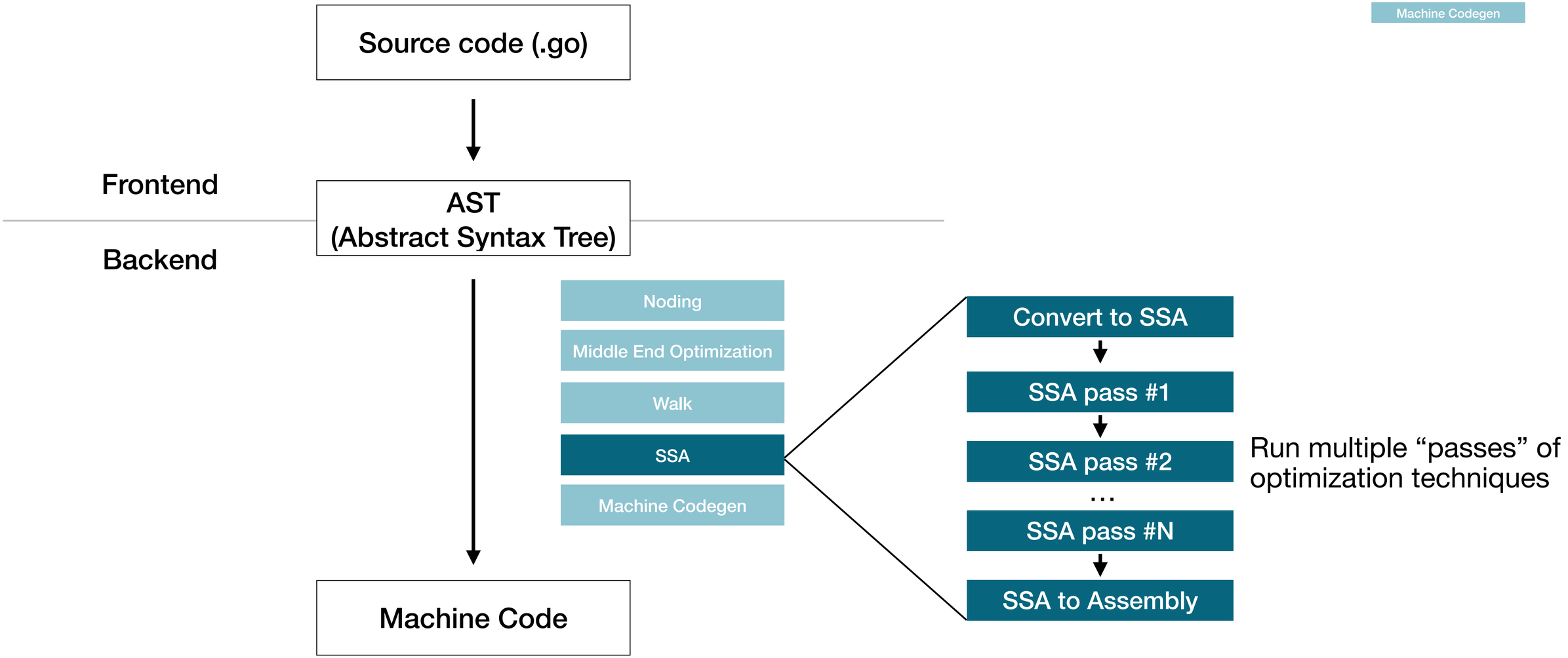
a2 is assigned value *once*

No variable gets value assigned > 1 times

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

SSA Optimization

When does it happen

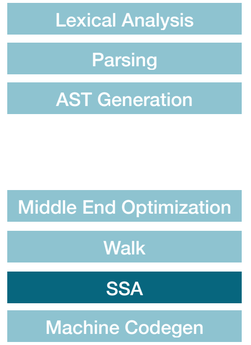


Lexical Analysis
Parsing
AST Generation
Middle End Optimization
Walk
SSA
Machine Codegen

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

SSA Optimization

Kinds of Optimization Passes




```
PhiElim
CopyElim
ShortCircuit
Decompose
CSE
PhilOpt
```

```
NilCheckElim
Prove
BCE
Loop
Fuse
DSE
```

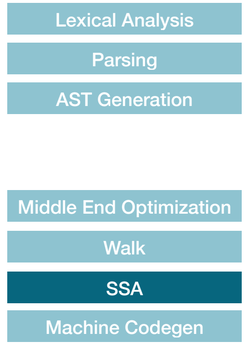
```
WriteBarrier
insertLoopReschedChecks
Critical
LikelyAdjust
Layout
```

```
FlagAlloc
RegAlloc
LoopRotate
```

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

SSA Optimization

Kinds of Optimization Passes




```
PhiElim
CopyElim
ShortCircuit
Decompose
CSE
PhilOpt
```

```
NilCheckElim
Prove
BCE
Loop
Fuse
DSE
```

```
WriteBarrier
insertLoopReschedChecks
Critical
LikelyAdjust
Layout
```

```
FlagAlloc
RegAlloc
LoopRotate
```

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

SSA Optimization

Ex. Common Subexpression Elimination

(Reduce redundant expressions)

Without SSA

```
y = x + 2
...
z = x + 2
```

Need to traverse code between `y = ...` and `z = ...`

With SSA

```
y = x + 2
...
z = y
```

No need to traverse code since variable is assigned value once

Lexical Analysis
Parsing
AST Generation

Middle End Optimization
Walk
SSA
Machine Codegen

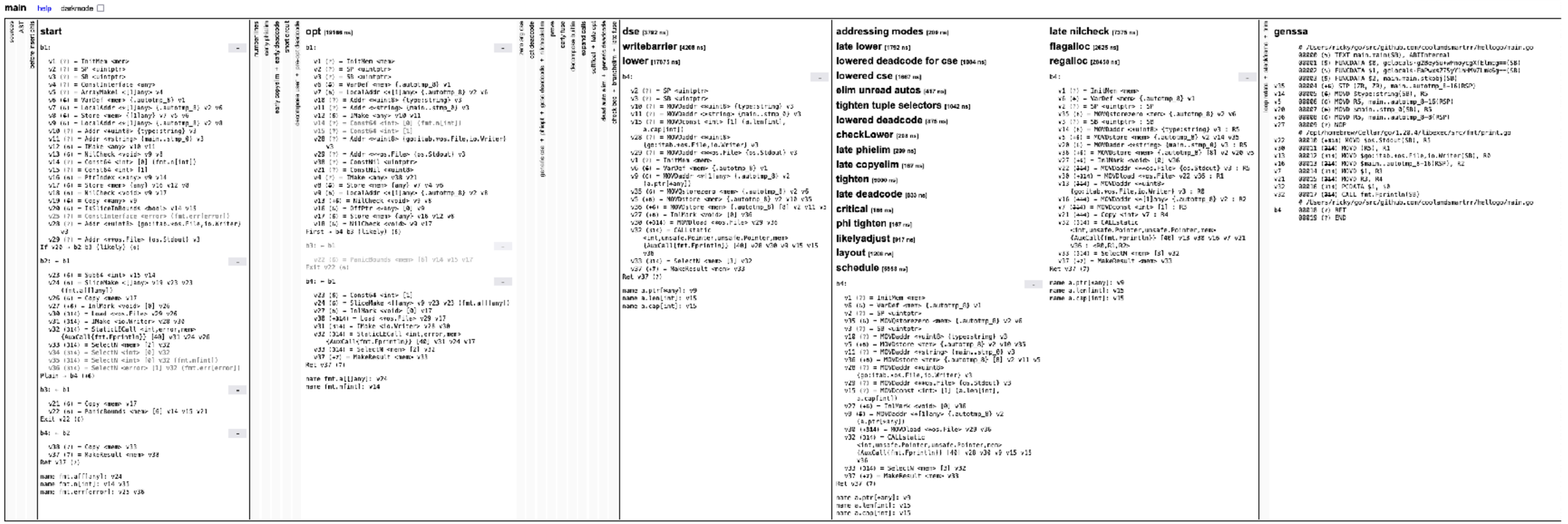
SSA Conversion

How to inspect Golang SSA

- `GOSSAFUNC`
  - Build flag to output SSA compilation in HTML
  - Shows code through each step of compilation
  - invoke by `GOSSAFUNC=main go build main.go`

SSA Conversion

See your intermediate code in GOSSAFUNC



Lexical Analysis

Parsing

AST Generation

Middle End Optimization

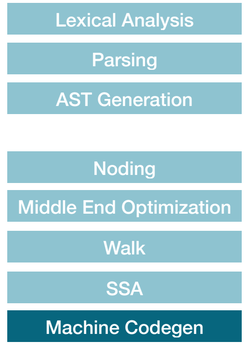
Walk

SSA

Machine Codegen

Twitter/GitHub/Instagram: @coolandsmartrr | Email: coolandsmartrr@gmail.com

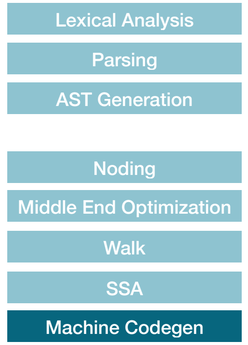
Machine Code Generation



- Turn the SSA representation into CPU Instruction
- Takes place in:
  - `src/cmd/compile/internal/ssa (SSA lowering + arch-specific passes)`
  - `src/cmd/compile/internal/obj (Machine code generation)`

Machine Code Generation

Lowering Pass



- “lowers” the abstraction of SSA code from machine-independent into machine-specific
- ex. ARM64: memory operands are possible, so maybe use load-store ops

Machine Code Generation

Examples of Final Code Passes

- Dead code elimination
- Move value closer to point of usage
- Remove local variables that are never read
- Register allocation
- Stack frame layout: assign stack offsets to local variables
- Pointer liveness analysis
  - compute which on-stack pointers are alive at each safepoint in Garbage Collector

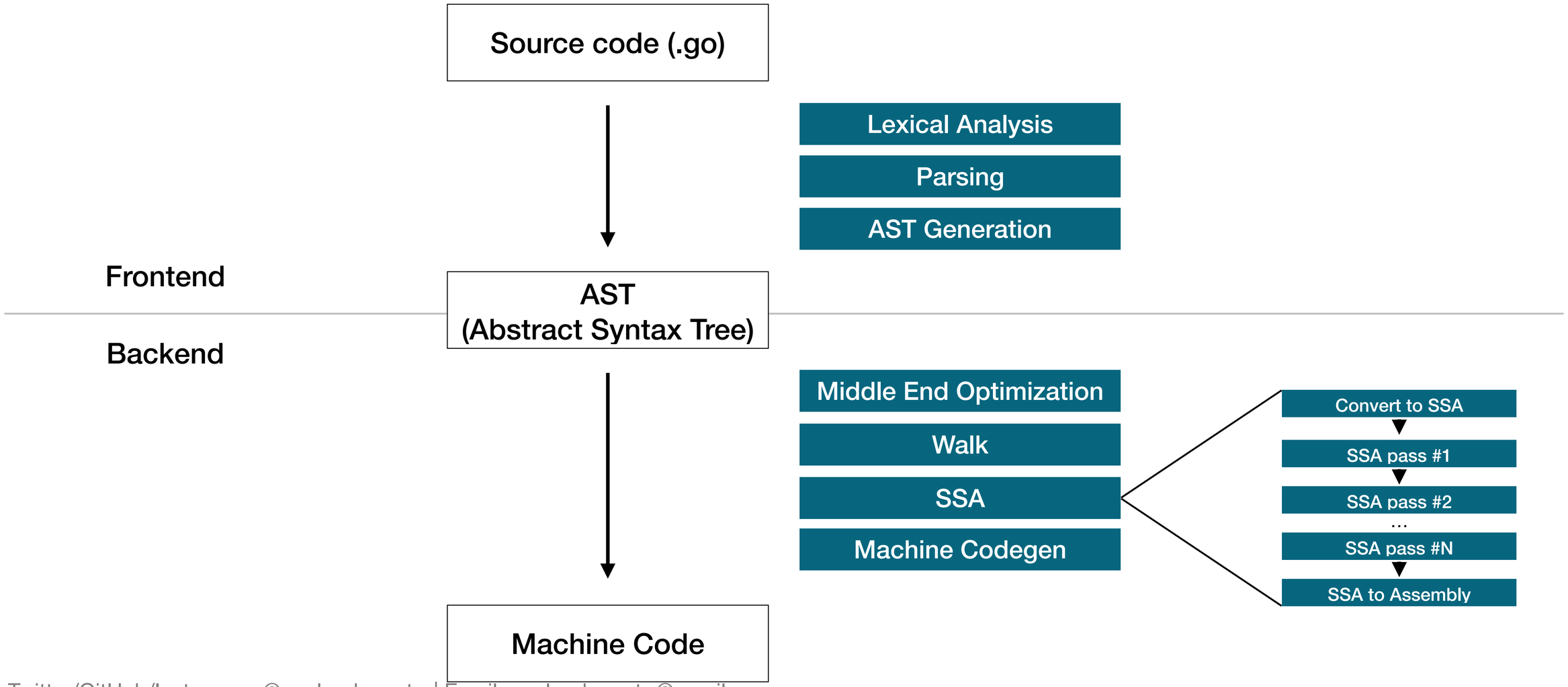
Machine Code Generation

Object File Writing

- Transform SSA into obj.Prog instructions
- Write out final object file
  - contains reflect data, export data, and debugging info

Conclusion

What We Covered



Resources

Compiler Internals

Introduction to the Go compiler (https://go.dev/src/cmd/compile/README)

A Journey With GO (https://medium.com/a-journey-with-go)

Eli Bendersky (https://eli.thegreenplace.net/tag/go)

Hacking Go Compiler Internals (https://speakerdeck.com/moriyoshi/hacking-go-compiler-internals-2nd-season)

SSA

Introduction to the Go compiler's SSA backend (https://go.dev/src/cmd/compile/internal/ssa/README)

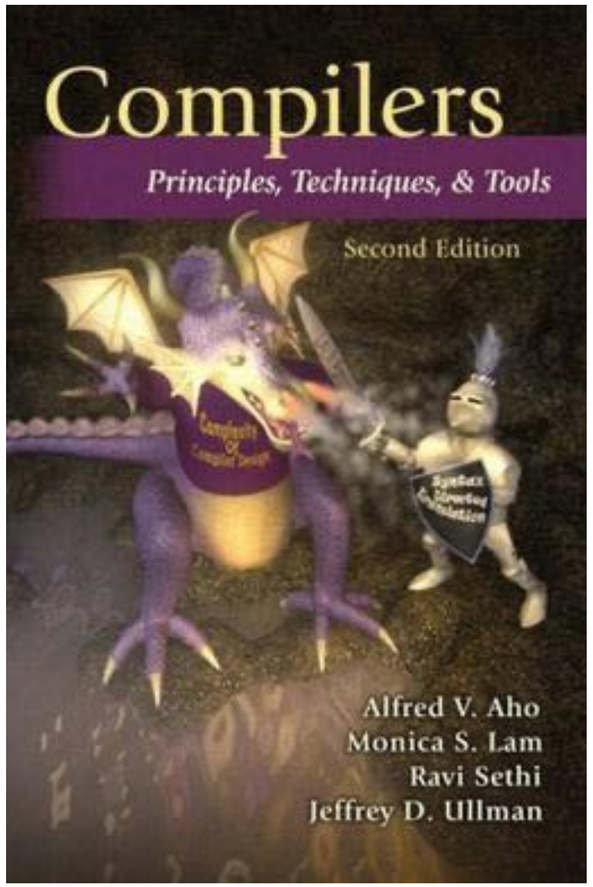
Keith Randall - Generating Better Machine Code with SSA (https://youtu.be/uTMvKVma5ms)

Inlining optimisations in Go (https://dave.cheney.net/2020/04/25/inlining-optimisations-in-go)

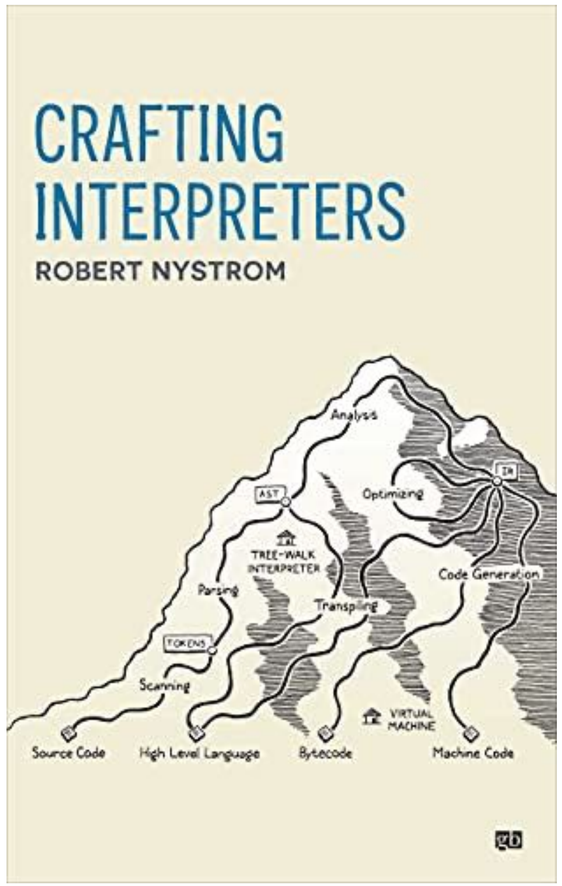
GCCGO

Gccgo in GCC 4.7.1 (https://go.dev/blog/gccgo-in-gcc-471)

Learn More on Compilers



Aho, A., Sethi, R. and Ullman, J. (2006). Compilers: Principles, Techniques, and Tools



Nystrom, R. (2021). Crafting interpreters

`exit()`

➡ Thank you for listening!