GopherChina2018

Go toolchain internals and implementation based on arm64

Wei Xiao (肖玮)
Arm Staff Software Engineer
Wei.Xiao@arm.com

Go toolchain overview

A toolchain is a package composed of the compiler and ancillary tools, libraries and runtime for a language which together allow you to build and run code written in that language.

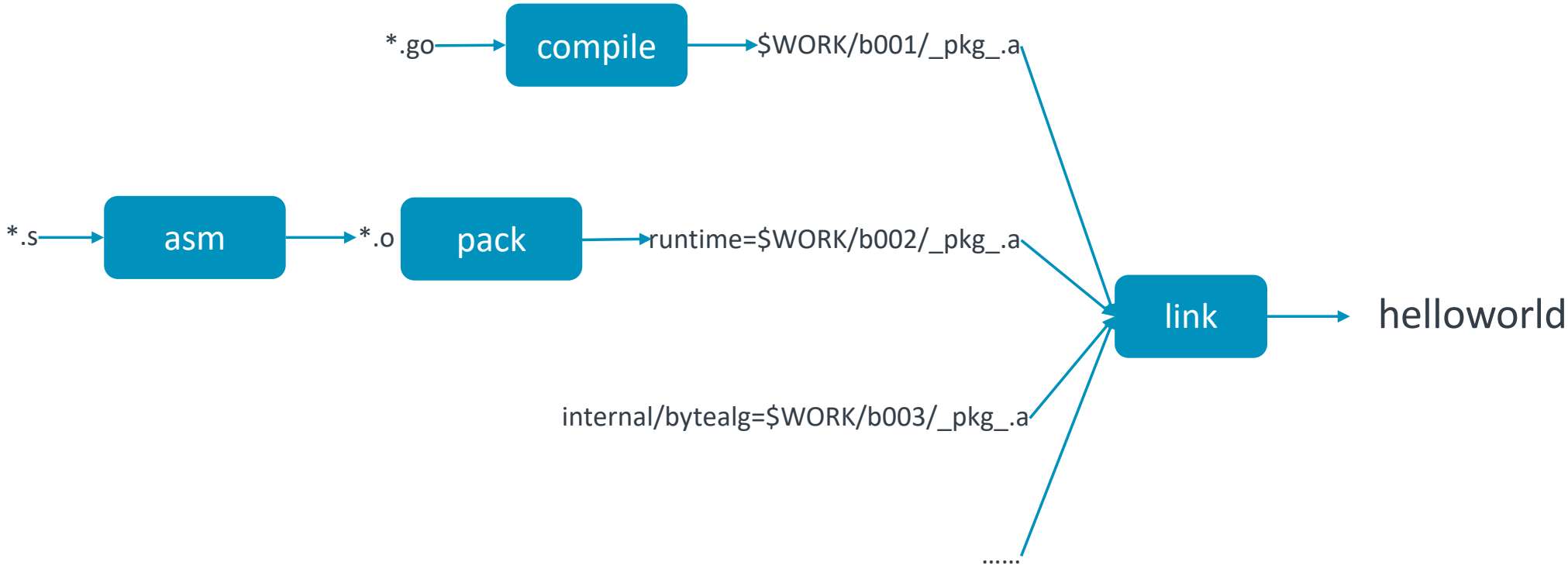
- gc: evolved from the Plan 9 toolchain and includes its own compiler, assembler, linker and tools, as well as the Go runtime and standard library.
- gccgo: extends the gcc project to support Go.
- llgo: built on top of the LLVM compiler infrastructure.

Go toolchain example

```
$go build -x helloworld.go
……
/golang/pkg/tool/linux_arm64/compile -o $WORK/b001/_pkg_.a -trimpath $WORK/b001 -p main -complete -buildid
Lz0Z4IaaV-BMteKblcuy/Lz0Z4IaaV-BMteKblcuy -D _/golang/test -importcfg $WORK/b001/importcfg -pack -c=4
./helloworld.go
/golang/pkg/tool/linux_arm64/buildid -w $WORK/b001/_pkg_.a # internal
……
/golang/pkg/tool/linux_arm64/link -o $WORK/b001/exe/a.out -importcfg $WORK/b001/importcfg.link -
buildmode=exe -buildid=C4PaIvbSKZNsCh5tSi2r/Lz0Z4IaaV-BMteKblcuy/fOp-
Rk_DpGJ5cJq23wER/C4PaIvbSKZNsCh5tSi2r -extld=gcc $WORK/b001/_pkg_.a
/golang/pkg/tool/linux_arm64/buildid -w $WORK/b001/exe/a.out # internal
mv $WORK/b001/exe/a.out helloworld
```

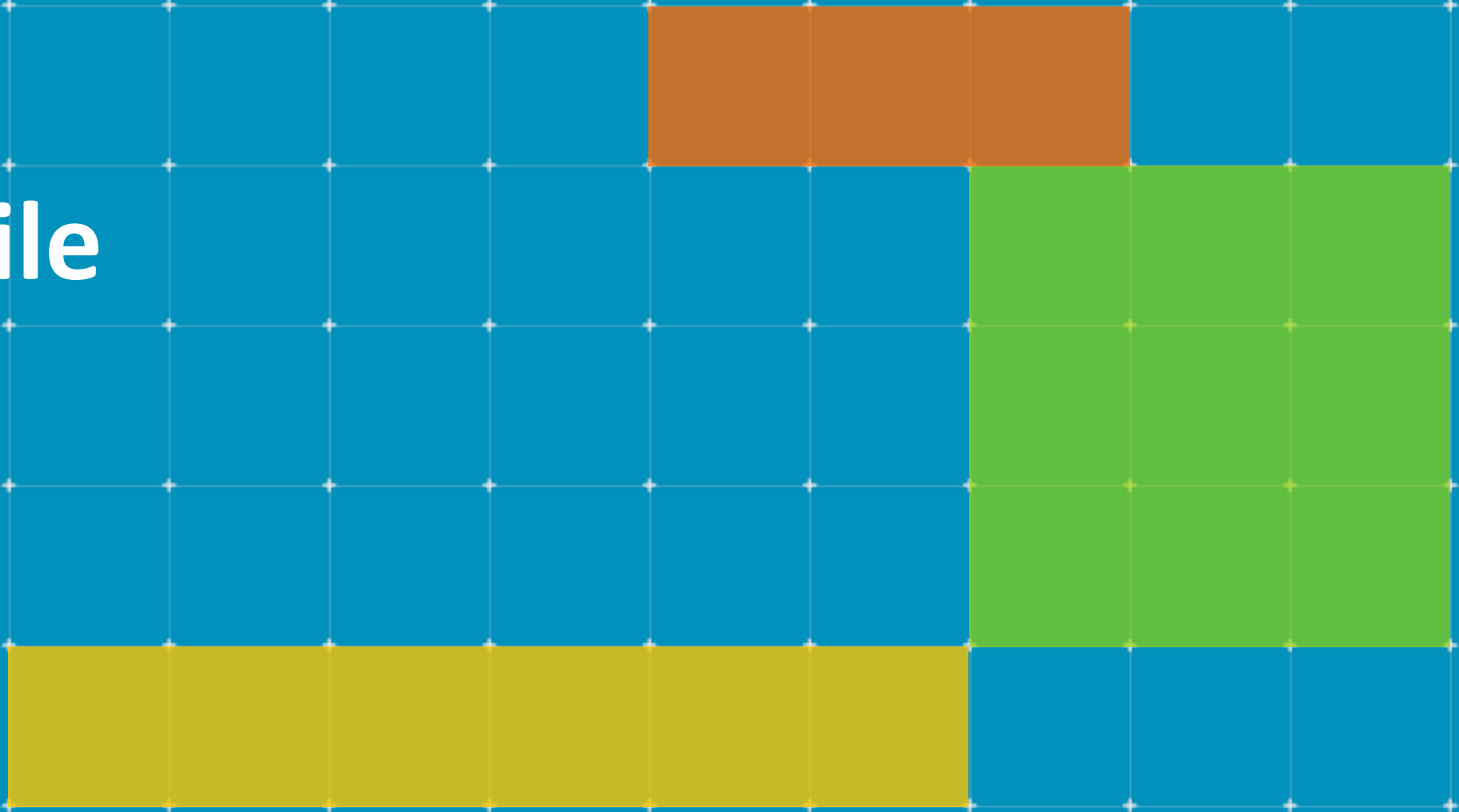
Go toolchain workflow

```
$ go tool
addr2line
api
asm
buildid
cgo
compile
cover
dist
doc
fix
link
nm
objdump
pack
pprof
test2json
trace
vet
```



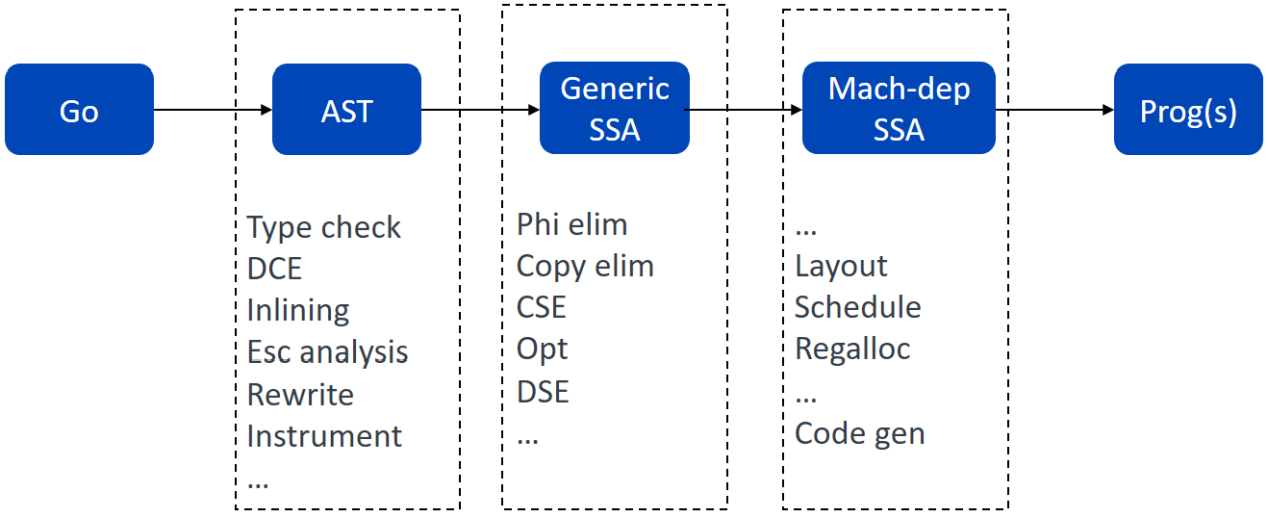
arm

Compile

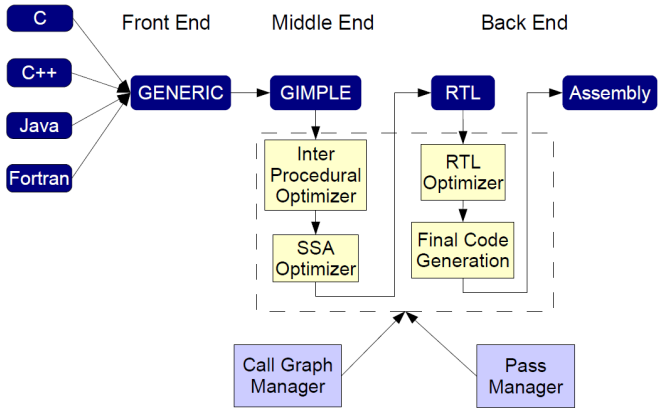


arm

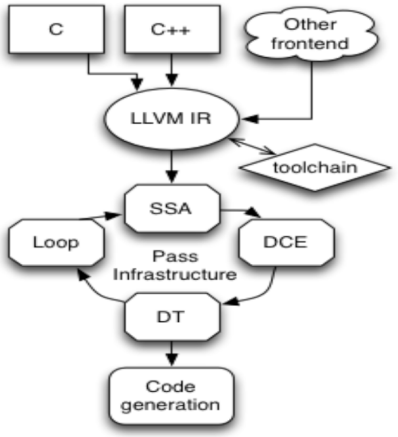
Go compiler overview



GCC



LLVM



Front end

- Syntax check
- Type check
- Dead code elimination
- Inline
- Escape analysis
- Declared and not used check
- Rewrite
- Instrument

Middle end

Machine-independent Passes

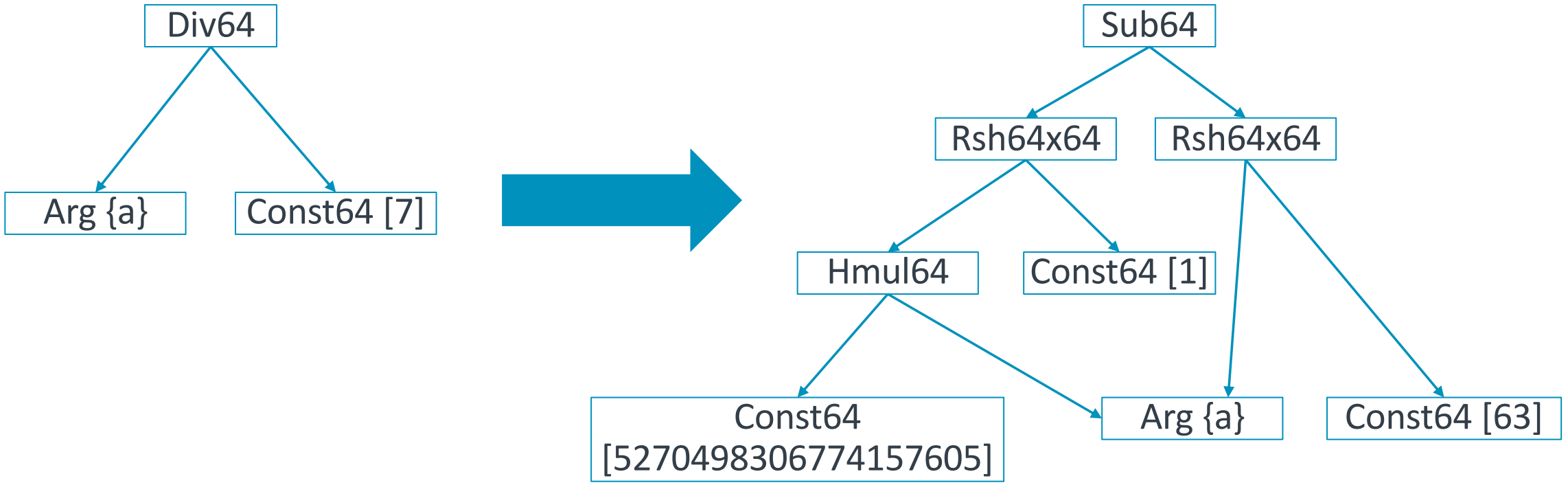
```
330 // list of passes for the compiler
331 var passes = [...]pass{
332     // TODO: combine phielim and copyelim into a single pass?
333     {name: "early phielim", fn: phielim},
334     {name: "early copyelim", fn: copyelim},
335     {name: "early deadcode", fn: deadcode}, // remove generated dead code to avoid doing pointless work during opt
336     {name: "short circuit", fn: shortcircuit},
337     {name: "decompose user", fn: decomposeUser, required: true},
338     {name: "opt", fn: opt, required: true},                // TODO: split required rules and optimizing rules
339     {name: "zero arg cse", fn: zcse, required: true},      // required to merge OpSB values
340     {name: "opt deadcode", fn: deadcode, required: true}, // remove any blocks orphaned during opt
341     {name: "generic cse", fn: cse},
342     {name: "phiopt", fn: phiopt},
343     {name: "nilcheckelim", fn: nilcheckelim},
344     {name: "prove", fn: prove},
345     {name: "loopbce", fn: loopbce},
346     {name: "decompose builtin", fn: decomposeBuiltIn, required: true},
347     {name: "softfloat", fn: softfloat, required: true},
348     {name: "late opt", fn: opt, required: true}, // TODO: split required rules and optimizing rules
349     {name: "generic deadcode", fn: deadcode},
350     {name: "check bce", fn: checkbce},
351     {name: "fuse", fn: fuse},
352     {name: "dse", fn: dse},
353     {name: "writebarrier", fn: writebarrier, required: true}, // expand write barrier ops
354     {name: "insert resched checks", fn: insertLoopReschedChecks,
355         disabled: objabi.Preemptibleloops_enabled == 0}, // insert resched checks in loops.
356     {name: "tighten", fn: tighten}, // move values closer to their uses
```

Options providing insight into compiler decisions:

- -d=ssa/<phase>/stats
- -m

arm

Machine-independent optimization example



a/7

(((a* 52704983067741576O5)>>64)>>1)-(a>>63)

arm

Back end

Machine-dependent Passes

```
357     {name: "lower", fn: lower, required: true},
358     {name: "lowered cse", fn: cse},
359     {name: "elim unread autos", fn: elimUnreadAutos},
360     {name: "lowered deadcode", fn: deadcode, required: true},
361     {name: "checkLower", fn: checkLower, required: true},
362     {name: "late phielim", fn: phielim},
363     {name: "late copyelim", fn: copyelim},
364     {name: "phi tighten", fn: phiTighten},
365     {name: "late deadcode", fn: deadcode},
366     {name: "critical", fn: critical, required: true}, // remove critical edges
367     {name: "likelyadjust", fn: likelyadjust},
368     {name: "layout", fn: layout, required: true},     // schedule blocks
369     {name: "schedule", fn: schedule, required: true}, // schedule values
370     {name: "late nilcheck", fn: nilcheckelim2},
371     {name: "flagalloc", fn: flagalloc, required: true}, // allocate flags register
372     {name: "regalloc", fn: regalloc, required: true},   // allocate int & float registers + stack slots
373     {name: "loop rotate", fn: loopRotate},
374     {name: "stackframe", fn: stackframe, required: true},
375     {name: "trim", fn: trim}, // remove empty blocks
```

Convert to machine-dependent ops

Example for arm64

Before

```
410 Div <T>
411   b1:
412     v1 = InitMem <mem>
413     v2 = SP <uintptr>
414     v5 = Addr <*int64> {~r1} v2
415     v6 = Arg <int64> {a}
416     v10 = VarDef <mem> {~r1} v1
417     v3 = Const64 <uint64> [5270498306774157605]
418     v12 = Const64 <uint64> [1]
419     v14 = Const64 <uint64> [63]
420     v4 = Hmul64 <int64> v3 v6
421     v13 = Rsh64x64 <int64> v6 v14
422     v7 = Rsh64x64 <int64> v4 v12
423     v9 = Sub64 <int64> v7 v13
424     v11 = Store <mem> {int64} v5 v9 v10
425     Ret v11
```

After

```
428 Div <T>
429   b1:
430     v1 = InitMem <mem>
431     v2 = SP <uintptr>
433     v6 = Arg <int64> {a}
434     v10 = VarDef <mem> {~r1} v1
435     v3 = MOVDconst <uint64> [5270498306774157605]
443     v4 = MULH <int64> v3 v6
444     v7 = SRAconst <int64> [1] v4
445     v9 = SUBshiftRA <int64> [63] v7 v6
446     v11 = MOVDstore <mem> {~r1} v2 v9 v10
447     Ret v11
```


arm

Generate Prog

Prog describes a single “machine” instruction

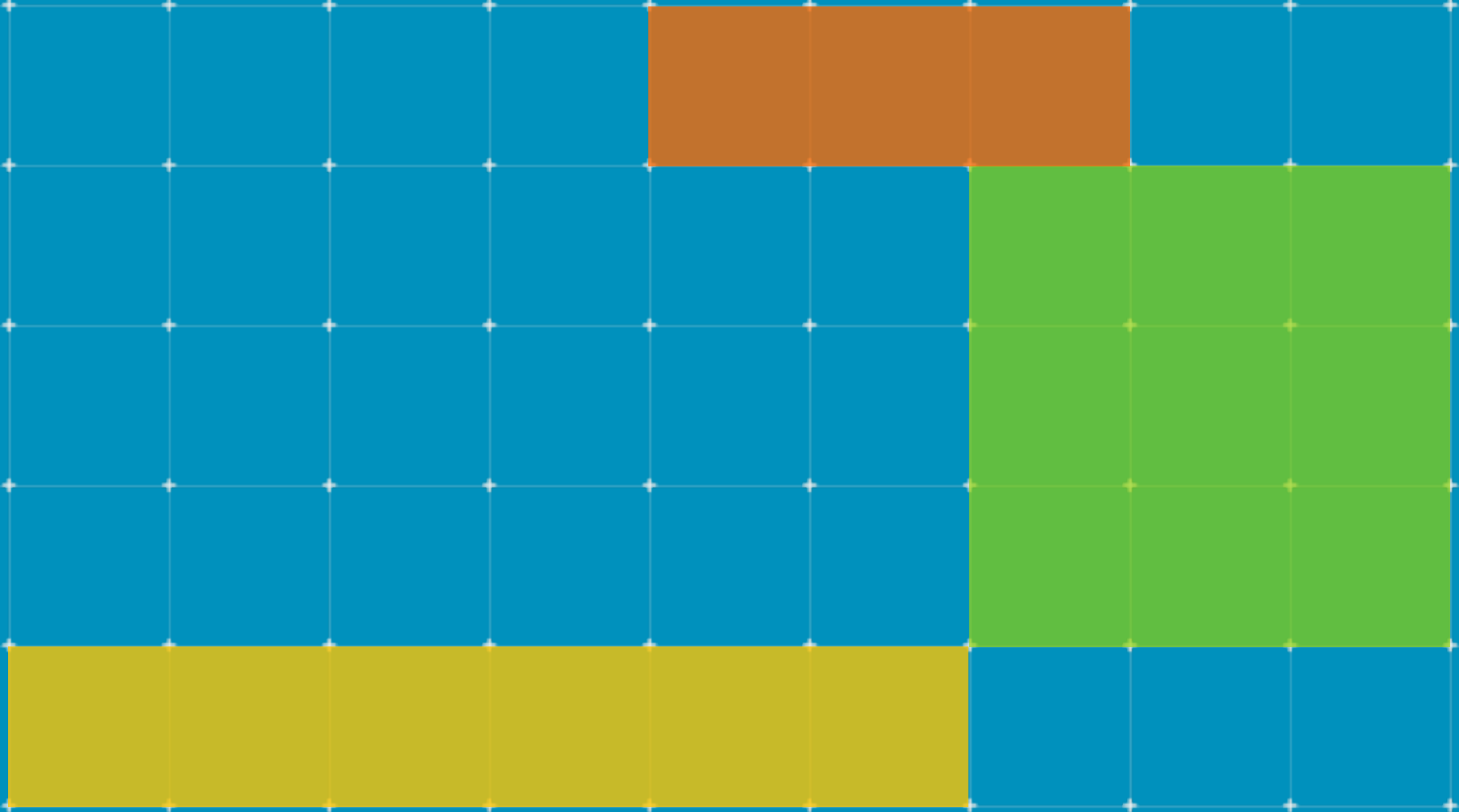
```
type Prog struct {
    Ctxt     *Link    // linker context
    Link     *Prog    // next Prog in linked list
    From     Addr     // first source operand
    RestArgs []Addr   // can pack any operands that not fit into {Prog.From, Prog.To}
    To       Addr     // destination operand (second is RegTo2 below)
    Pcond    *Prog    // target of conditional jump
    Forwd    *Prog    // for x86 back end
    Rel      *Prog    // for x86, arm back ends
    Pc       int64    // for back ends or assembler: virtual or actual program counter, depending on phase
    Pos      src.XPos // source position of this instruction
    Spadj    int32    // effect of instruction on stack pointer (increment or decrement amount)
    As       As       // assembler opcode
    Reg      int16    // 2nd source operand
    RegTo2   int16    // 2nd destination operand
    Mark     uint16   // bitmask of arch-specific items
    Optab    uint16   // arch-specific opcode index
    Scond    uint8    // condition bits for conditional instruction (e.g., on ARM)
    Back     uint8    // for x86 back end: backwards branch state
    Ft       uint8    // for x86 back end: type index of Prog.From
    Tt       uint8    // for x86 back end: type index of Prog.To
    Isize    uint8    // for x86 back end: size of the instruction in bytes
}
```

```
722       00000 (3)     TEXT    "".Div(SB)
……

725  v19    00003 (4)     MOVD    $5270498306774157605, R0
726  v18    00004 (4)     MOVD    "".a(RSP), R1
727  v4     00005 (4)     SMULH   R1, R0, R0
728  v7     00006 (4)     ASR     $1, R0, R0
729  v9     00007 (4)     SUB     R1->63, R0, R0
730  v11    00008 (4)     MOVD    R0, "".~r1+8(RSP)
731  b1     00009 (4)     RET
732       00010 (?)     END
```

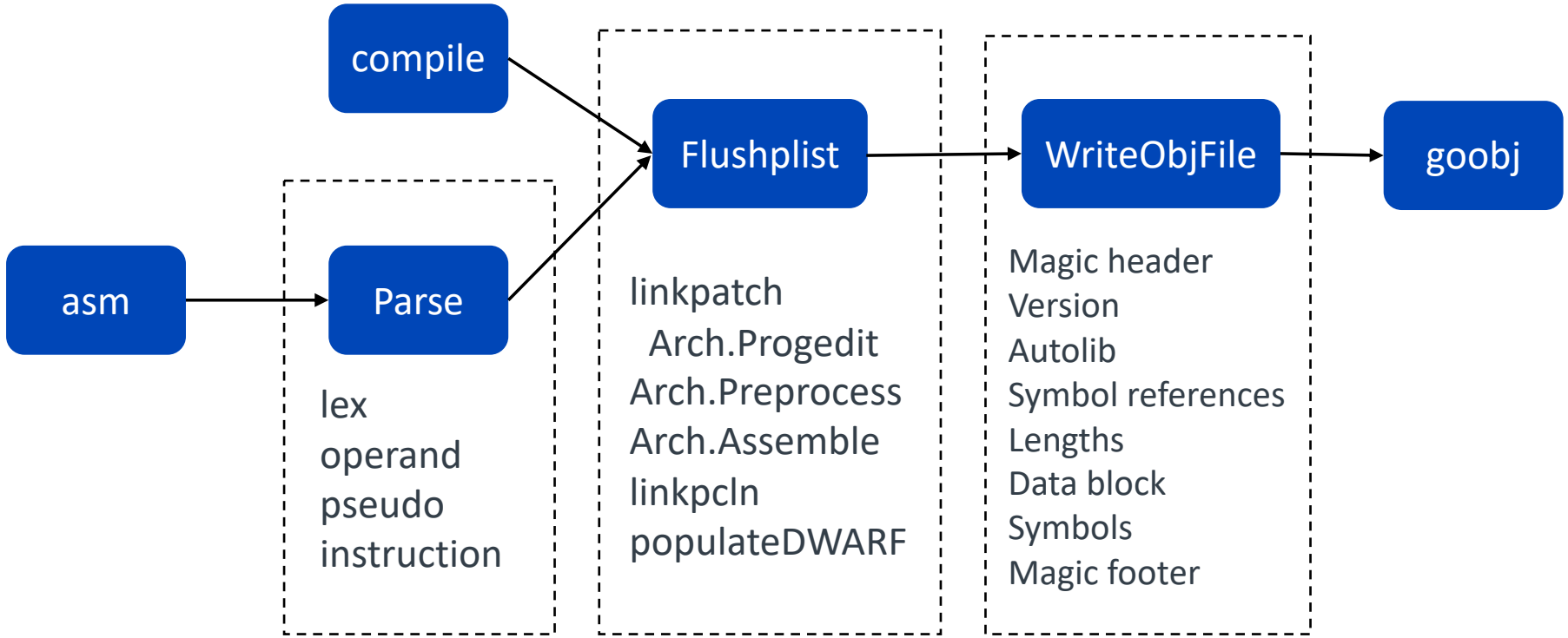

arm

Asm



arm

Go assembler overview



arm

Go arm64 assembly example

```
1 // func Add(a, b int) int
2 TEXT ·Add(SB),$0-24
3       MOVD    arg1+0(FP), R0
4       MOVD    arg2+8(FP), R1
5       ADD     R1, R0, R0
6       MOVD    R0, ret+16(FP)
7       RET
```

```
<asm.Add>:
f9400b81        ldr     x1, [x28,#16]
910003e2        mov     x2, sp
eb01005f        cmp     x2, x1
540000c9        b.ls    8b674 <asm.Add+0x24>
f94007e0        ldr     x0, [sp,#8]
f9400be1        ldr     x1, [sp,#16]
8b010000        add     x0, x0, x1
f9000fe0        str     x0, [sp,#24]
d65f03c0        ret
aa1e03e3        mov     x3, x30
97ff2e72        bl      57040 <runtime.morestack_noctxt>
17fffff5        b       8b650 <asm.Add>
```

Semi-abstract instruction set

Move instructions

	386	amd64	arm	arm64	mips64	ppc64	s390x
1-byte	MOVB		MOVB	-	-	-	-
1-byte sign extend	MOVBLSX	MOVBQSX	MOVBS	MOVB	MOVB	MOVB	MOVB
1-byte zero extend	MOVBLZX	MOVBQZX	MOVBU	MOVBU	MOVBU	MOVBZ	MOVBZ
2-byte	MOVW		MOVH	-	-	-	-
2-byte sign extend	MOVWLSX	MOVWQSX	MOVHS	MOVH	MOVH	MOVH	MOVH
2-byte zero extend	MOVWLZX	MOVWQZX	MOVHU	MOVHU	MOVHU	MOVHZ	MOVHZ
4-byte	MOVL		MOVW	-	-	-	-
4-byte sign extend	-	MOVLQSX	-	MOVW	MOVW	MOVW	MOVW
4-byte zero extend	-	MOVLQZX	-	MOVWU	MOVWU	MOVWZ	MOVWZ
8-byte	-	MOVQ	-	MOVD	MOVV	MOVD	MOVD

arm

Goobj

```
00000000: 676f 206f 626a 6563 7420 6c69 6e75 7820  go object linux
00000010: 6172 6d36 3420 6465 7665 6c20 2b64 3835  arm64 devel +d85
00000020: 6233 3561 2054 6875 204e 6f76 2039 2030  b35a Thu Nov 9 0
00000030: 323a 3134 3a30 3120 3230 3137 202b 3030  2:14:01 2017 +00
00000040: 3030 0a21 0a00 0067 6f31 396c 6401 00fe  00.!...go19ld...
00000050: 0661 6464 00fe 4267 6f66 696c 652e 2e2f  .add..Bgofile../
00000060: 686f 6d65 2f66 616e 6661 6e30 312f 7465  home/fanfan01/te
00000070: 7374 2f61 6464 2e73 00fe 1667 6f2e 696e  st/add.s...go.in
00000080: 666f 2e61 6464 00fe 1867 6f2e 7261 6e67  fo.add...go.rang
00000090: 652e 6164 6400 ffa2 0108 0000 0002 e107  e.add...........
000000a0: 40f9 e20b 40f9 4100 018b e10f 00f9 c003  @...@.A.........
000000b0: 5fd6 0000 0000 0000 0000 0000 0000 0208  _...............
000000c0: 0002 0800 0a01 0201 0201 0201 0204 0000  ................
000000d0: 0800 0261 6464 0000 0000 0000 0000 0000  ...add..........
000000e0: 0000 0000 0000 0001 9c00 0000 0001 00fe  ................
000000f0: 0102 0040 0040 0220 0016 0000 3000 0202  ...@.@. ....0...
00000100: 0006 0616 0600 0002 0400 fe08 0600 3a00  ..............:.
00000110: 3a06 0a10 0200 021a 1002 4002 2e08 3a00  :.........@...:.
00000120: 04fe 0908 0000 0000 00ff ff67 6f31 396c  ...........go19l
00000130: 64                                       d
```

Head: go object + os + arch + version

Magic header

Magic footer

Version

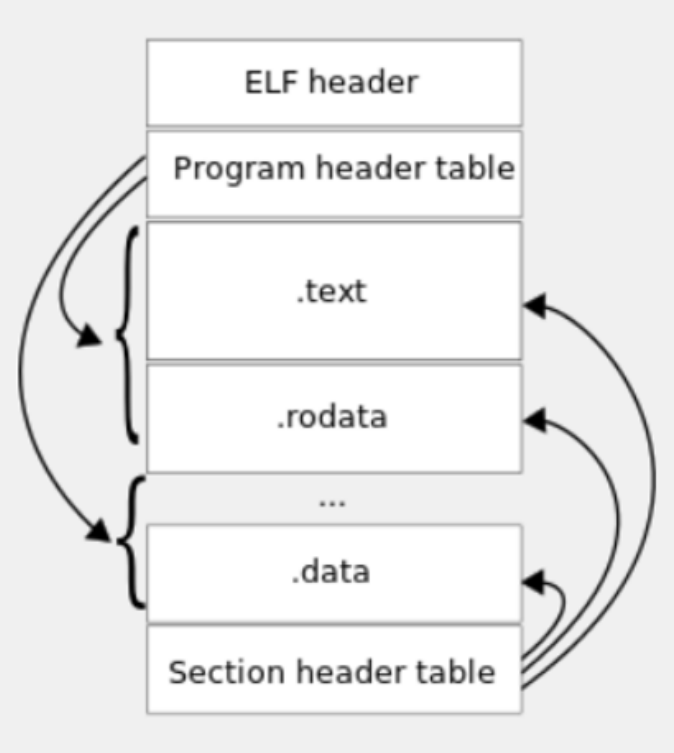
Symbol references

Length

Symbol

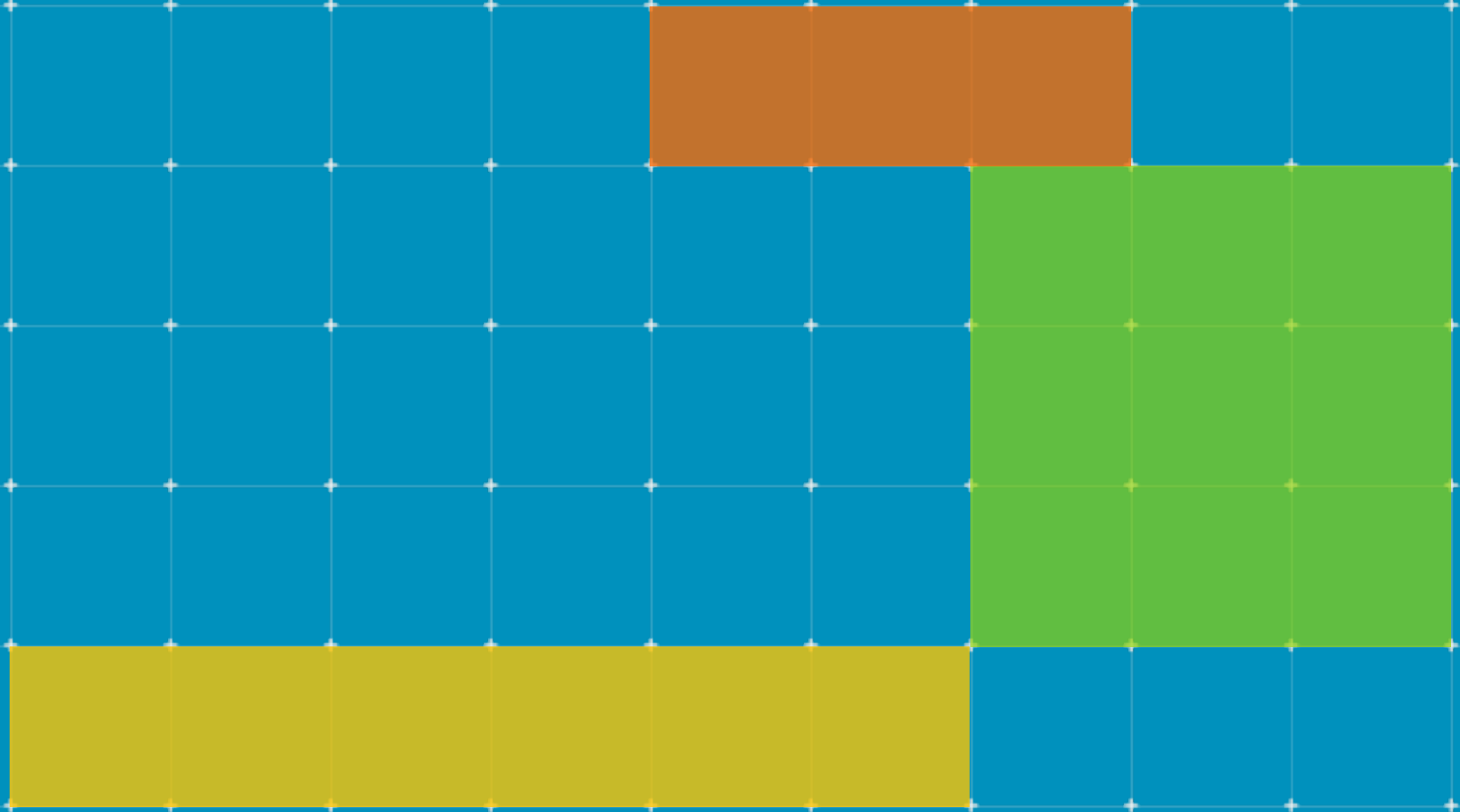
Data block: machine code + pc-related data array

ELF



arm

Link



arm

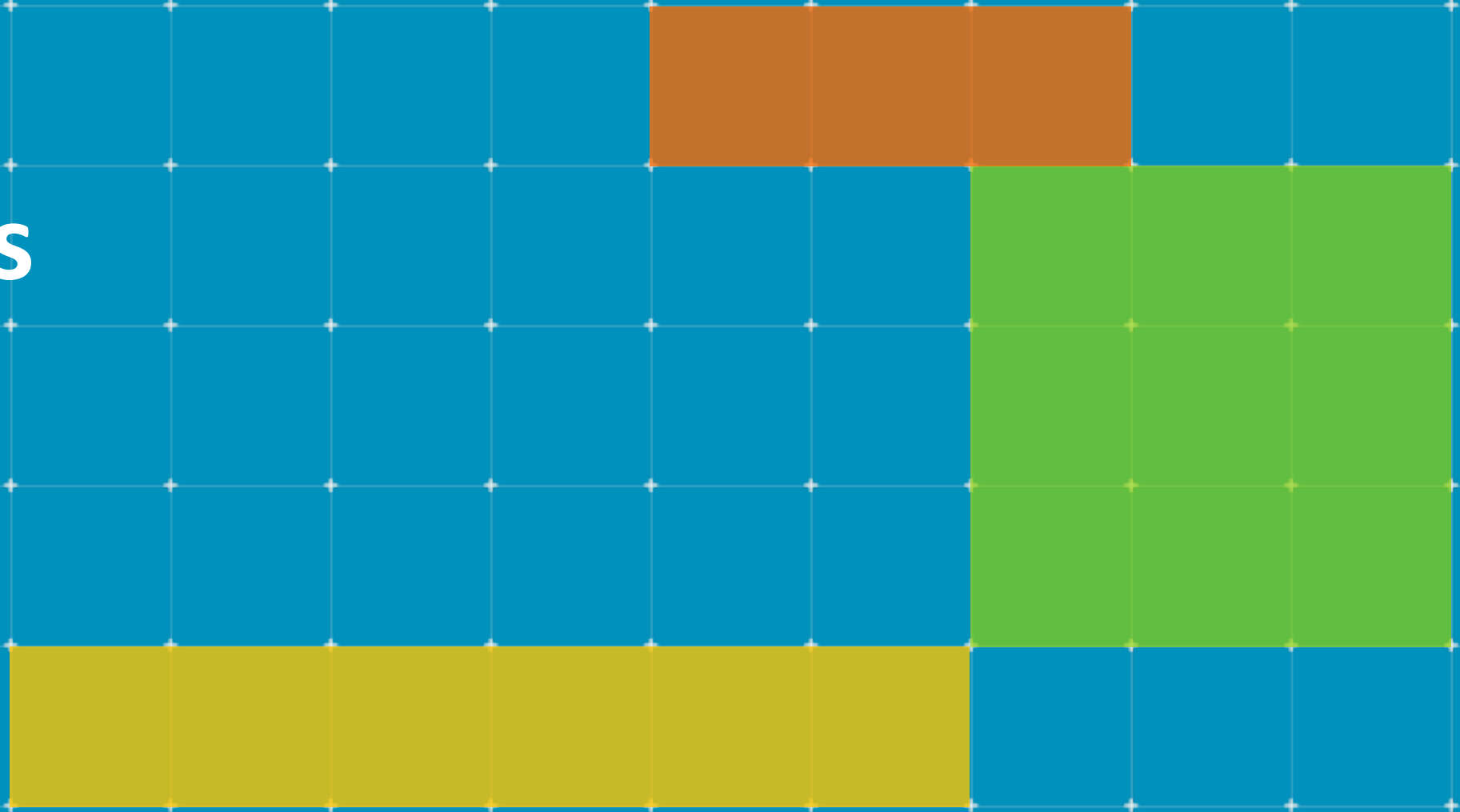
Go link overview

arm

Go link workflow

1. Load package libraries and objects (including host objects)
2. Determine link mode
3. Mark all reachable symbols
4. Prepare ELF symbols (e.g set up PLT)
5. Assign addresses to text
6. Reorganize Go meta data (pcln, functab, type and …)
7. Handle relocations of host objects (if internal link)
8. Assign addresses to data
9. Generate DWARF (set up the per-compilation unit part of the DIE tree)
10. Resolve relocations (target dependent)
11. Write out ELF (code, data, table, header and …)
12. Emit relocations for host link (if external link)
13. Check undefined symbols
14. Invoke host link to generate output (if external link)
15. Invoke host ar to generate output (if C Archive)

Others



arm

Some interesting topics

- VGo: Go += Package Versioning
- WebAssembly architecture
- Safe-points everywhere
- Register-based calling convention
- Mid-stack inlining
- ……

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद