

# A Peek Into Static Binary Analysis

Sun Aug 18 03:15:00 PM CEST 2024  
FrOSCon 2024, Hochschule Bonn-Rhein-Sieg

# \$ whoami



## CA&D | Cyber Analysis and Defense



\$ whoami



## CA&D | Cyber Analysis and Defense

Intrusion Detection and Analysis

Maritime Cyber Security

Reaction

Malware Analysis and Intelligence

Digital Evidence and Forensics

Botnet Intelligence and Mitigation

Detection

Secure Production and Energy Networks

Secure Mobile Communications

Software and Firmware Security

aka. SAFE

Prevention

# Open Source at CA&D & SAFE



FKIE-CAD

## Popular repositories

### [awesome-embedded-and-iot-security](#)

Public

A curated list of awesome embedded and IoT security resources.

☆ 1.7k 🍴 232

### [FACT\\_core](#)

Public

Firmware Analysis and Comparison Tool

Python ☆ 1.2k 🍴 224

### [cwe\\_checker](#)

Public

cwe\_checker finds vulnerable patterns in binary executables

Rust ☆ 1.1k 🍴 116

### [friTap](#)

Public

The goal of this project is to help researchers to analyze traffic encapsulated in SSL or TLS.

JavaScript ☆ 255 🍴 27

### [dewolf](#)

Public

A research decompiler implemented as a Binary Ninja plugin.

Python ☆ 164 🍴 9

### [libdesock](#)

Public

A de-socketing library for fuzzing.

C ☆ 126 🍴 13





# Open Source at CA&D & SAFE



FKIE-CAD

## Popular repositories

### [awesome-embedded-and-iot-security](#)

Public

A curated list of awesome embedded and IoT security resources.

☆ 1.7k 🍴 232

### [FACT\\_core](#)

Public

Firmware Analysis and Comparison Tool

Python ☆ 1.2k 🍴 224

### [cwe\\_checker](#)

cwe\_checker finds vulnerable patterns in binary executables.

Rust ☆ 1.1k 🍴 116



### [friTap](#)

Public

The goal of this project is to help researchers to analyze traffic encapsulated in SSL or TLS.

JavaScript ☆ 255 🍴 27

### [dewolf](#)

Public

A research decompiler implemented as a Binary Ninja plugin.

Python ☆ 164 🍴 9

### [libdesock](#)

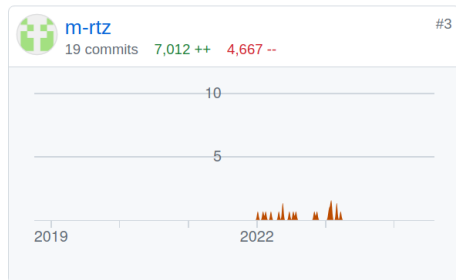
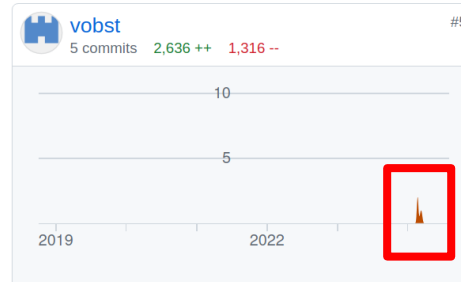
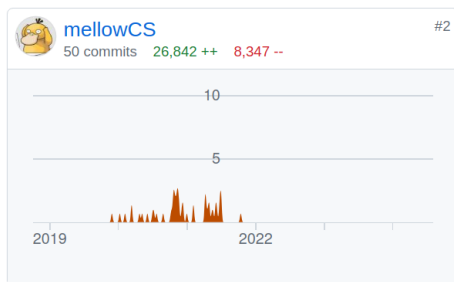
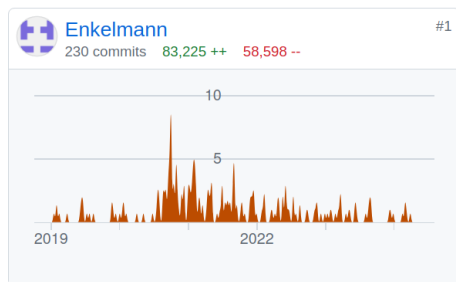
Public

A de-socketing library for fuzzing.

C ☆ 126 🍴 13



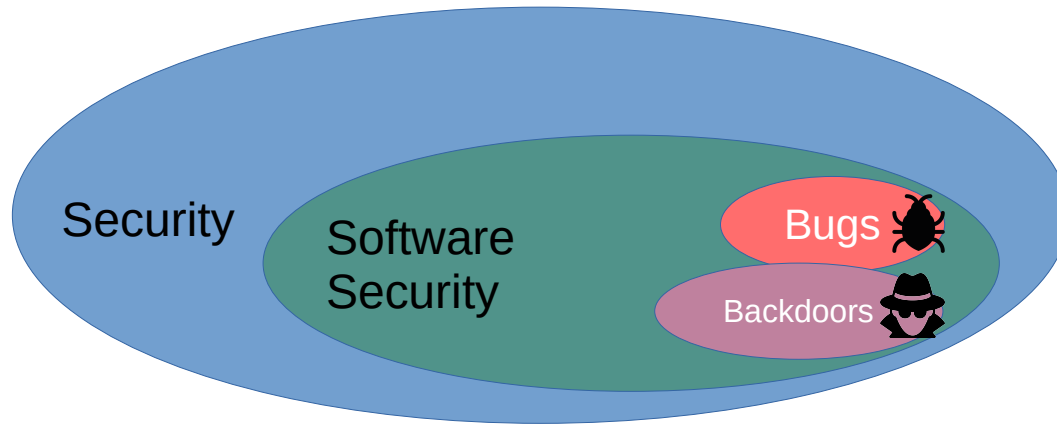
# \$ whoami



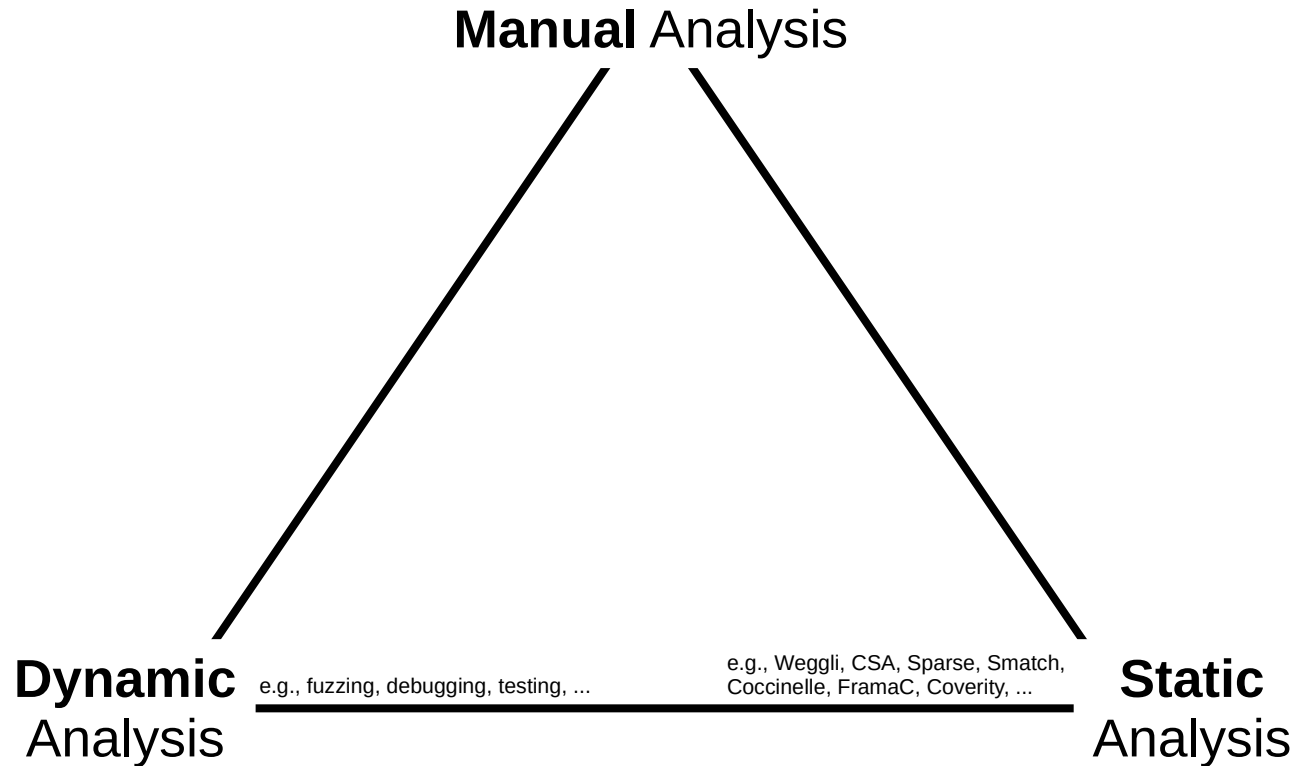
## cwe\_checker contributors

- [Thomas Barabosch](#)
  - Original author, maintainer 2018-2019
- [Nils-Edvin Enkelmann](#)
  - Maintainer 2020-2024
- [Valentin Obst](#)
  - Maintainer since 2024
- [Jörg Stucke](#)
- [Melvin Klimke](#)
- [Mauritz van den Bosch](#)

# Context: Auditing

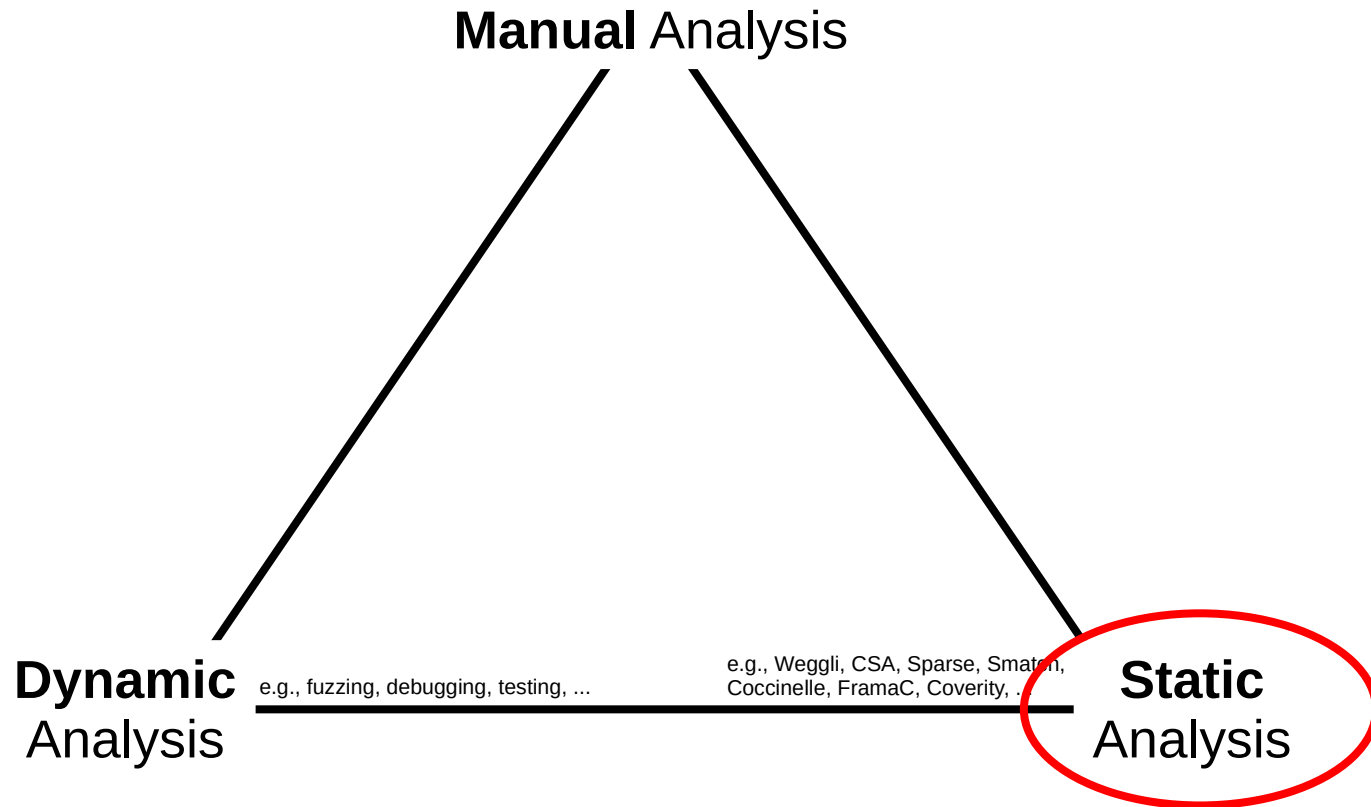


# The Software Auditor's Toolbox





# The Software Auditor's Toolbox




# Why Auditing Matters

Not much I can add to Amir's talk! :) ... however, note everything is open source (yet)

## Lecture: Improving Security Posture of Critical FOSS Projects with Security Audits

Version 1.0\_first\_flush




*The Open Source Technology Improvement Fund, Inc has organized and managed close to 100 security audits for critical open source projects since 2015. This session will go over the top types of vulnerabilities found and fixed in open source security audits, the top 5 lessons learned, and highlight common auditing mistakes and how to avoid them.*

Security Audits are a proven and effective method for improving the security posture of Open Source Projects. The Open Source Technology Improvement Fund, Inc (ostif.org) has been a trusted partner for facilitating and managing security audits for critical open source projects since 2020, helping critical FOSS projects mature and improve. With recent funding from Sovereign Tech Fund, OSTIF wishes to share case studies of successful engagements.

### Info

Day: 2024-08-18  
Start time: 10:00  
Duration: 01:00  
Room: HS7  
Track: **Security**  
Language: en

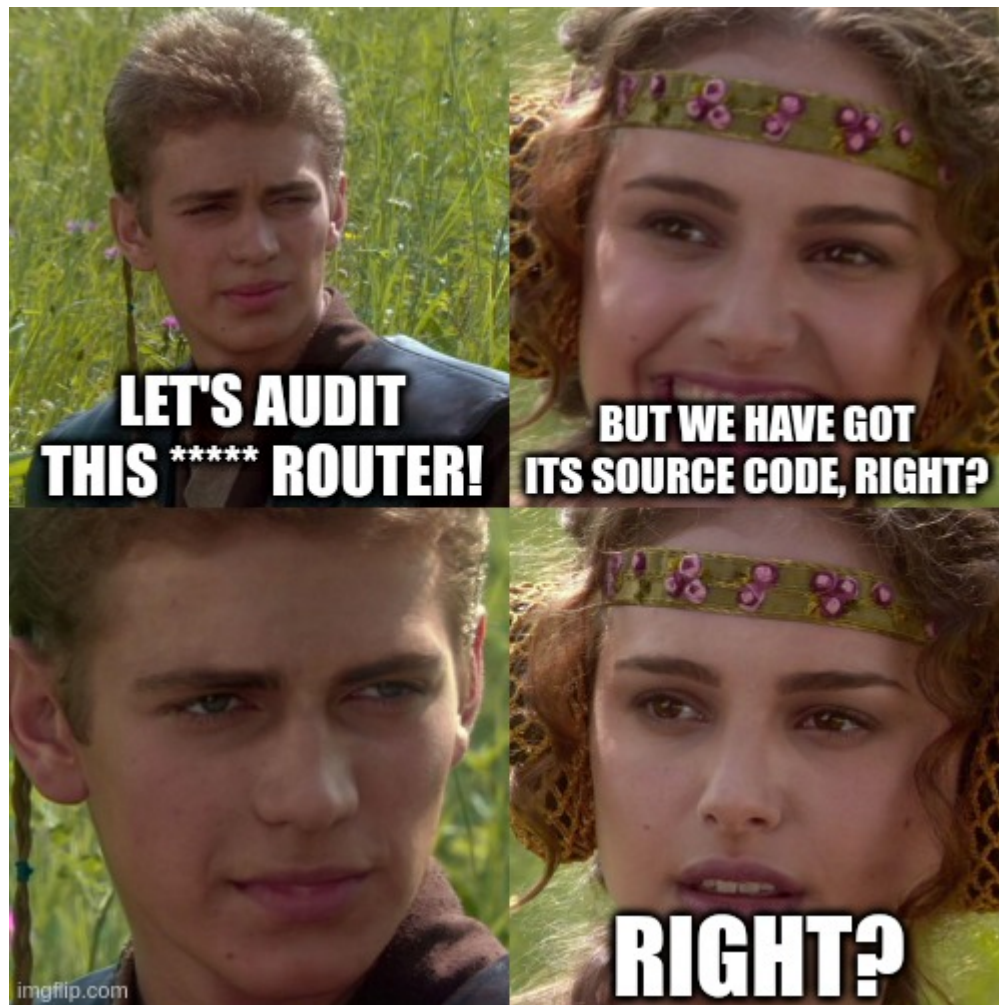
### Speakers

[Amir Montazery](#)

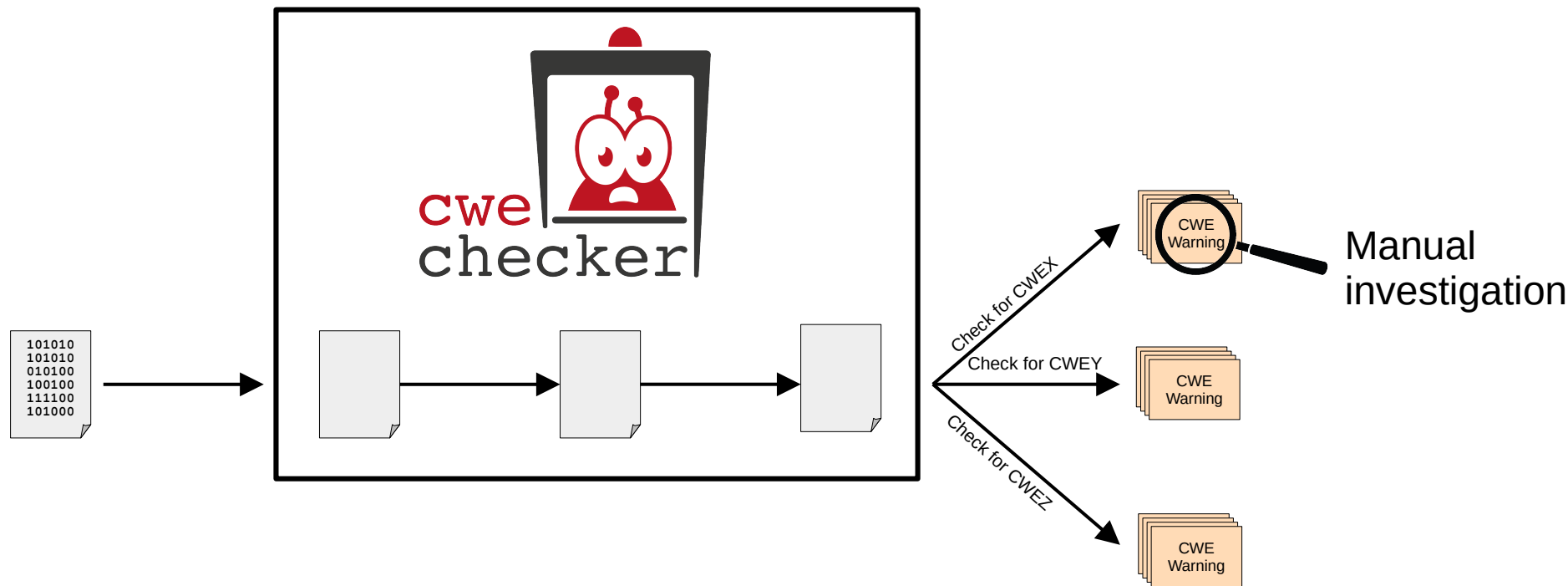
### Links:

- [iCalendar](#)
- [OSTIF Collection of Security Audit Reports](#)

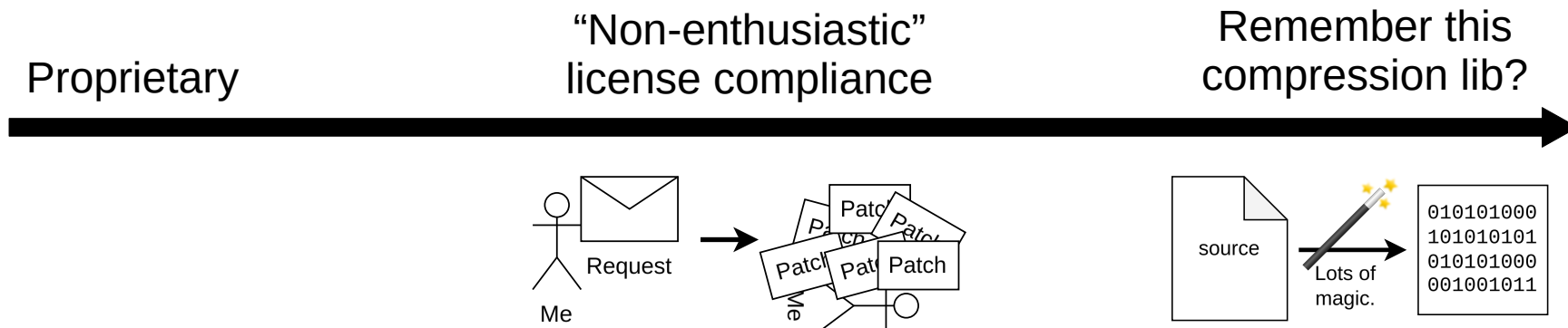
### Concurrent Events



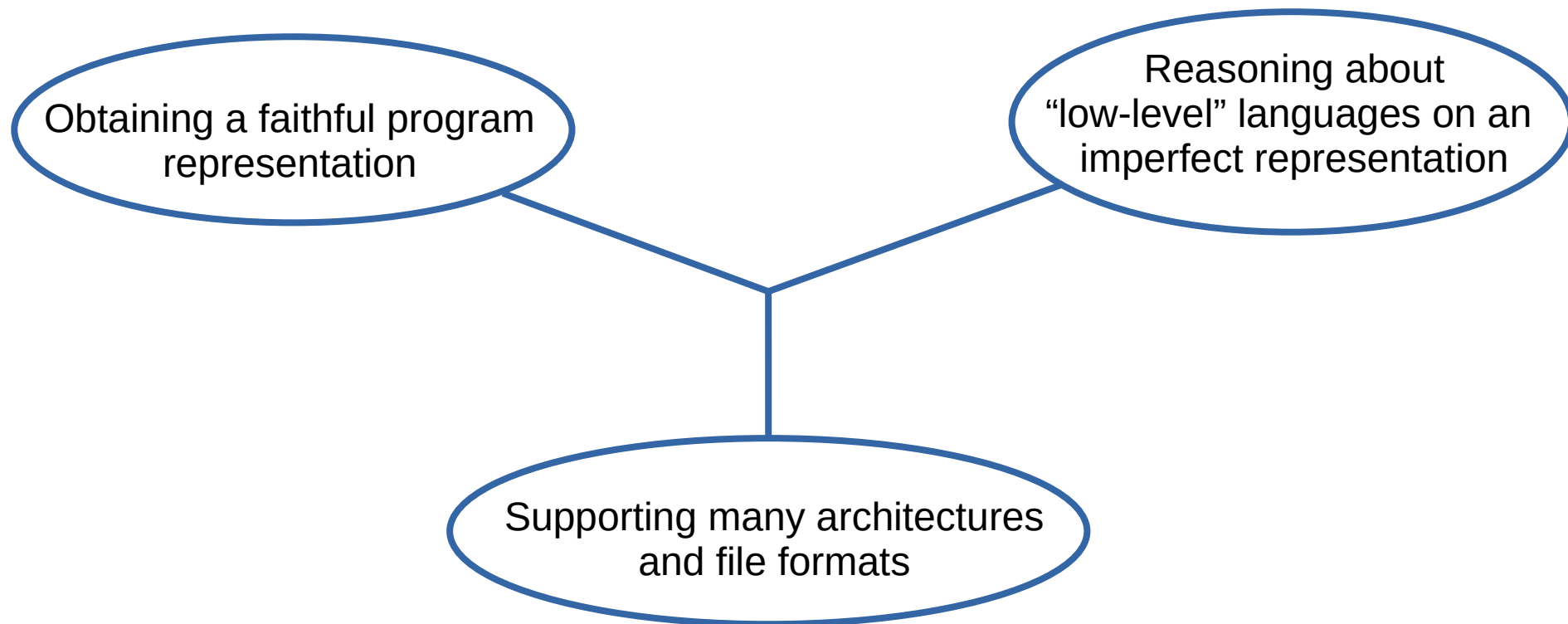
# Our Goal: Practical Static Analysis for Binary Code



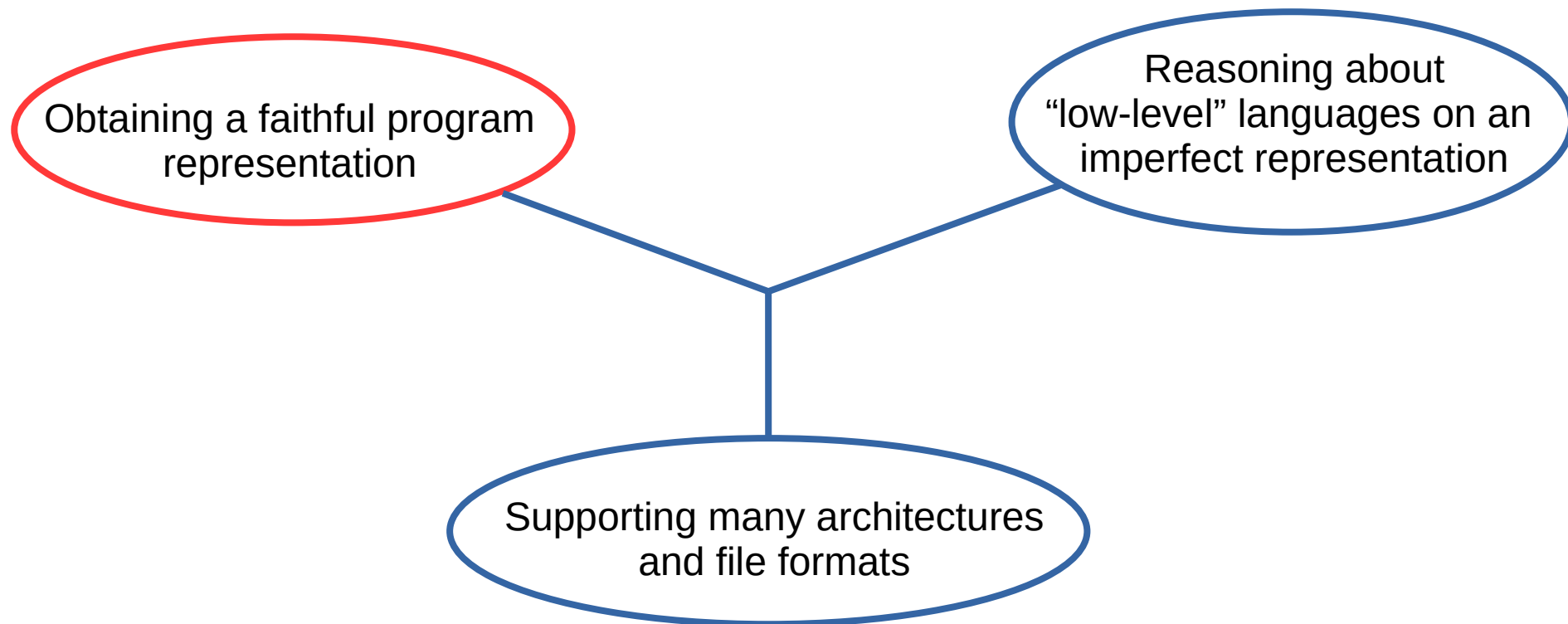
# Why Audit w/o Source Code?



# So ... what makes this hard(er)?

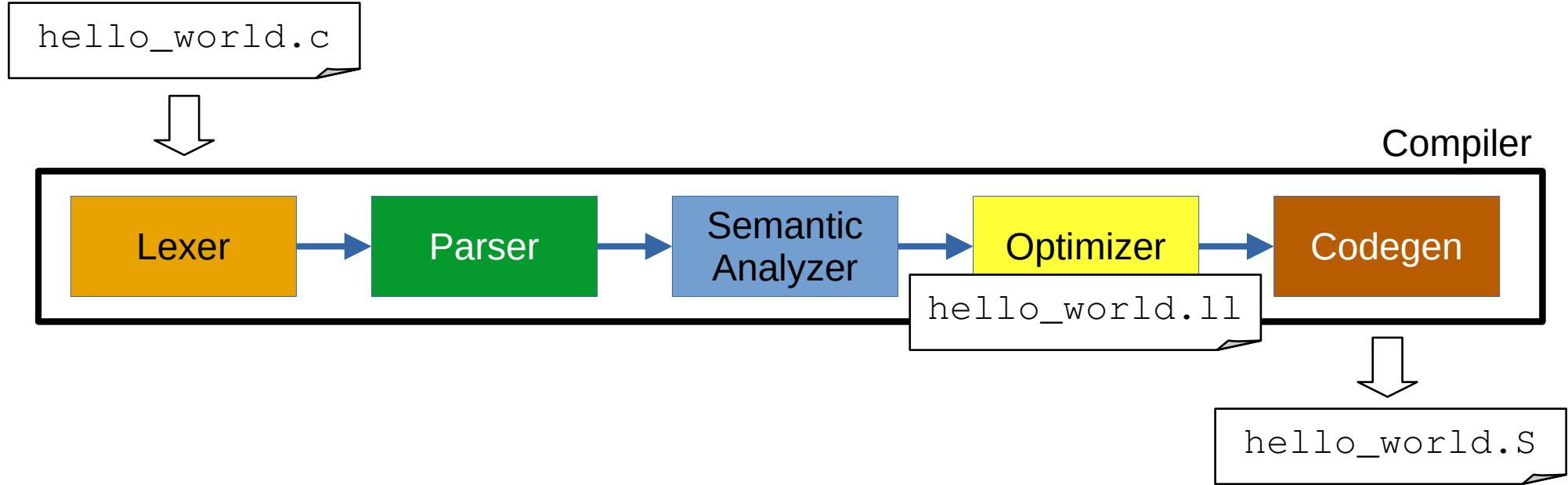


# So ... what makes this hard(er)?





# Reminder: Build Process



hello\_world.c

```
#include <stdio.h>

void main() {
    puts("Hello, world.");
}
```

Compiler

hello\_world.ll

```
[...]
@.str = private unnamed_addr constant [14 x i8]
c"Hello, world.\00", align 1

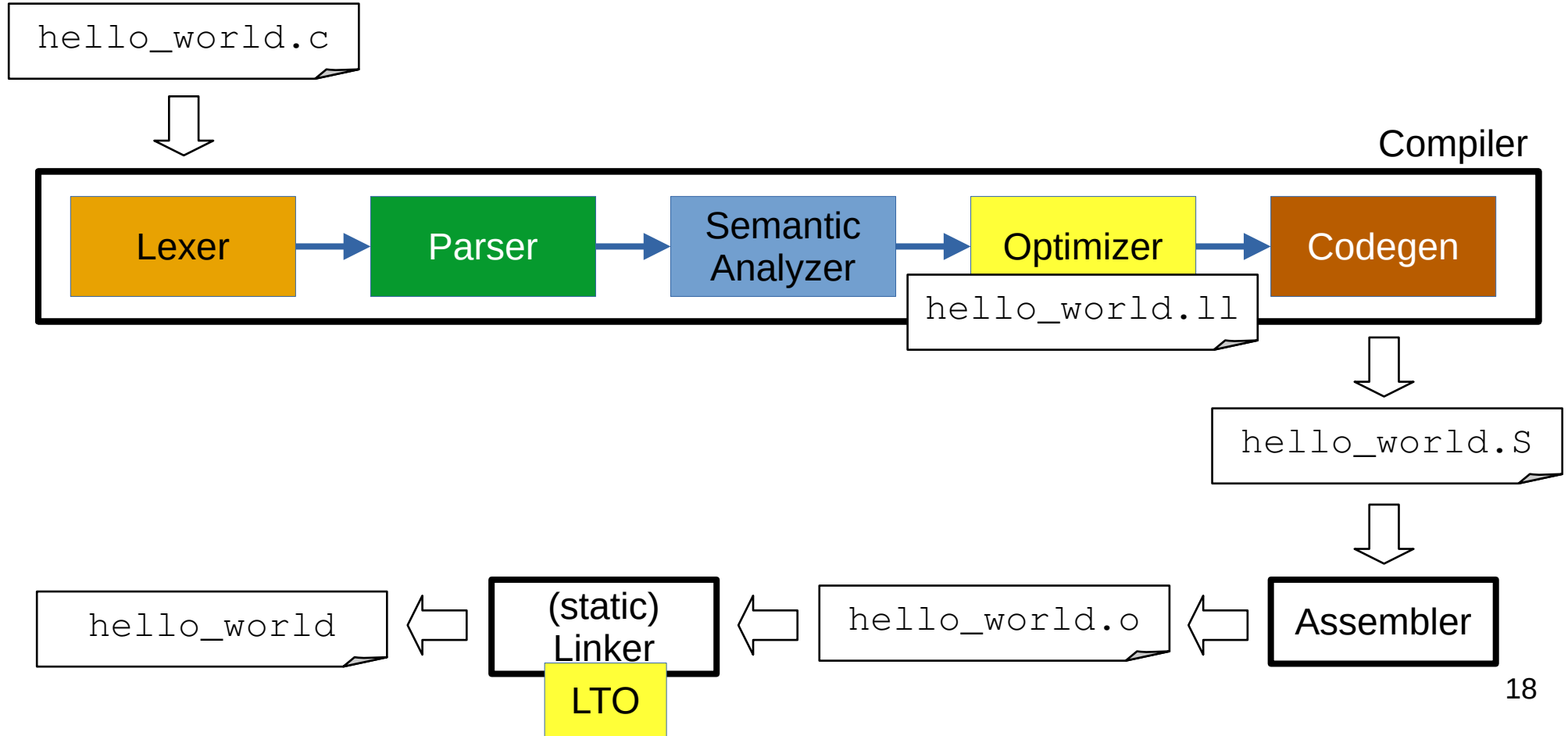
; Function Attrs: noinline nounwind optnone sspstrong
uwtable
define dso_local void @main() #0 {
    %1 = call i32 @puts(ptr noundef @.str)
    ret void
}
[...]
```

hello\_world.S

```
[...]
.LC0:
    .string "Hello, world."
    .text
    .align 2
    .global main
    .type main, %function
main:
.LFB0:
    stp x29, x30, [sp, -16]!
    mov x29, sp
    adrp x0, .LC0
    add x0, x0, :lo12:.LC0
    bl puts
    nop
    ldp x29, x30, [sp], 16
    ret
[...]
```

Compiler

# Reminder: Build Process



hello\_world.S

```
.LC0:
.string "Hello, world."
.text
.align 2
.global main
.type main, %function
main:
.LFB0:
    stp x29, x30, [sp, -16]!
    mov x29, sp
    adrp x0, .LC0
    add x0, x0, :lo12:.LC0
    bl puts
    nop
    ldp x29, x30, [sp], 16
    ret
```

Assembler

hello\_world.o

```
0000000000000000 <main>:
    0:    a9bf7bfd    stp    x29, x30, [sp, #-16]!
    4:    910003fd    mov    x29, sp
    8:    90000000    adrp   x0, 0 <main>
    c:    91000000    add    x0, x0, #0x0
   10:    94000000    bl     0 <puts>
   14:    d503201f    nop
   18:    a8c17bfd    ldp    x29, x30, [sp], #16
   1c:    d65f03c0    ret
```

hello\_world.o

(static) linker

hello\_world

0000000000000000 <main>:

```
0: a9bf7bfd stp x29, x30, [sp, #-16]!
4: 910003fd mov x29, sp
8: 90000000 adrp x0, 0 <main>
c: 91000000 add x0, x0, #0x0
10: 94000000 bl 0 <puts>
14: d503201f nop
18: a8c17bfd ldp x29, x30, [sp], #16
1c: d65f03c0 ret
```

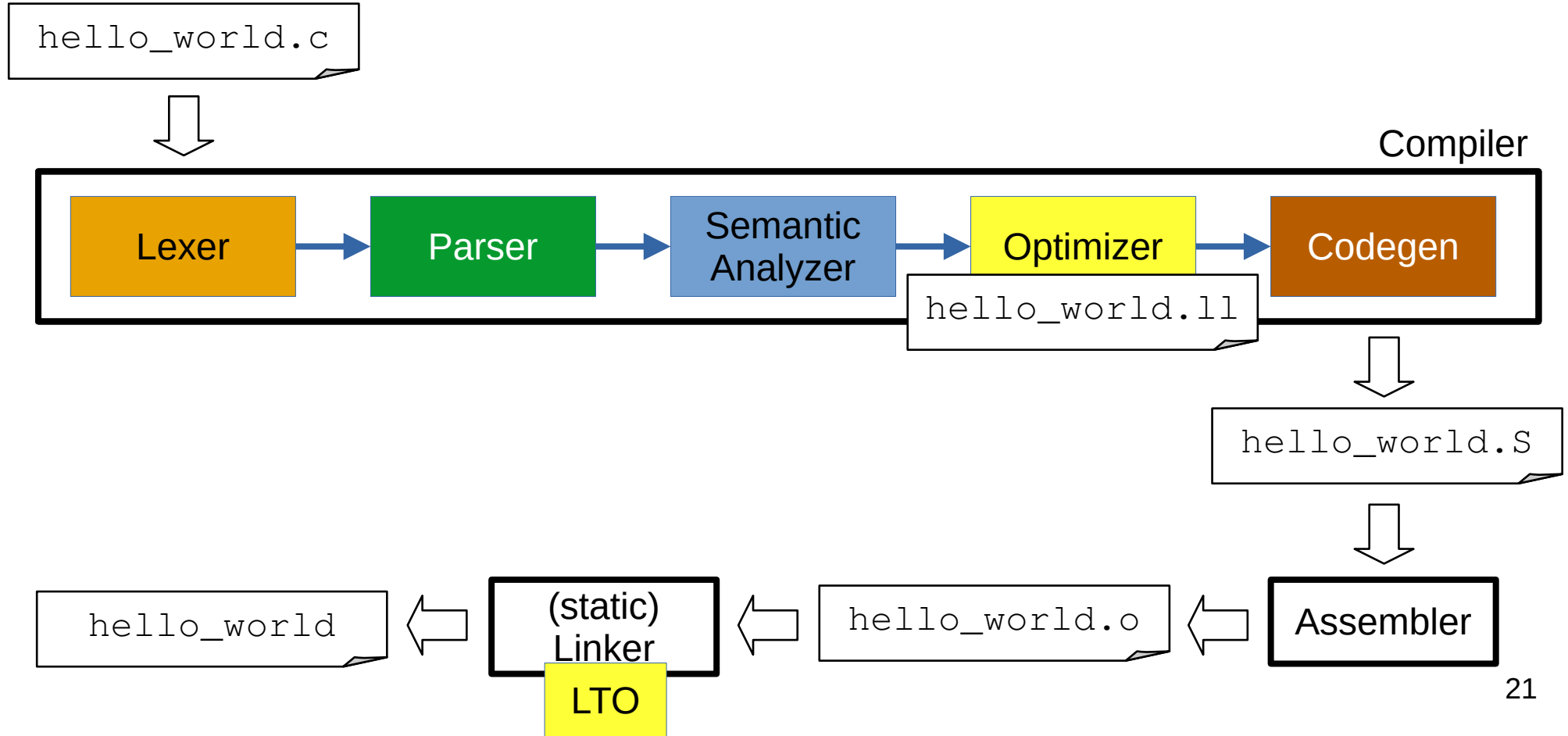
0000000000000764 <main>:

```
764: a9bf7bfd stp x29, x30, [sp, #-0x10]!
768: 910003fd mov x29, sp
76c: 90000000 adrp x0, 0x0 <puts@GLIBC_2.17>
770: 911e8000 add x0, x0, #0x7a0
774: 97ffffaf bl 0x630 <puts@plt>
778: d503201f nop
77c: a8c17bfd ldp x29, x30, [sp], #0x10
780: d65f03c0 ret
```

0000000000000640 <\_start>:

```
640: d503201f nop
644: d280001d mov x29, #0x0
648: d280001e mov x30, #0x0
64c: aa0003e5 mov x5, x0
650: f94003e1 ldr x1, [sp]
654: 910023e2 add x2, sp, #0x8
658: 910003e6 mov x6, sp
65c: f00000e0 adrp x0, 0x1f000
660: f947ec00 ldr x0, [x0, #0xfd8]
664: d2800003 mov x3, #0x0
668: d2800004 mov x4, #0x0
66c: 97ffffe1 bl 0x5f0 <__libc_start_main@plt>
670: 97ffffec bl 0x620 <abort@plt>
```

# Reminder: Build Process



hello\_world.o

Assembler

hello\_world

0000000000000000 <main>:

```
0: a9bf7bfd stp x29, x30, [sp, #-16]!
4: 910003fd mov x29, sp
8: 90000000 adrp x0, 0 <main>
c: 91000000 add x0, x0, #0x0
10: 94000000 bl 0 <puts>
14: d503201f nop
18: a8c17bfd ldp x29, x30, [sp], #16
1c: d65f03c0 ret
```

0000000000000764 <main>:

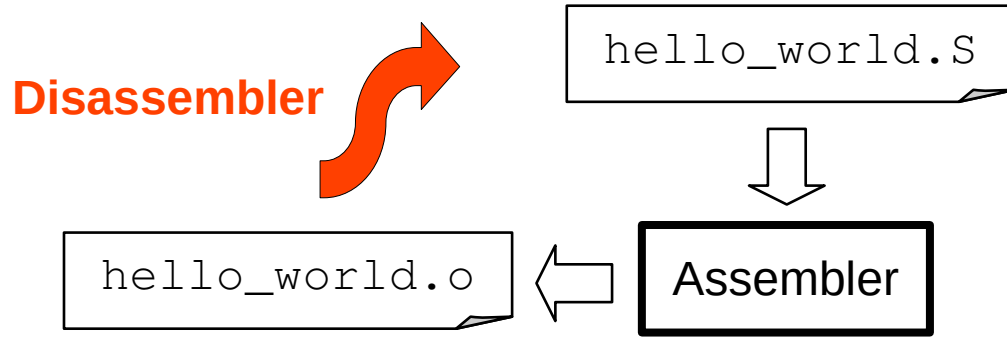
```
764: a9bf7bfd stp x29, x30, [sp, #-0x10]!
768: 910003fd mov x29, sp
76c: 90000000 adrp x0, 0x0 <puts@GLIBC_2.17>
770: 911e8000 add x0, x0, #0x7a0
774: 97ffffaf bl 0x630 <puts@plt>
778: d503201f nop
77c: a8c17bfd ldp x29, x30, [sp], #0x10
780: d65f03c0 ret
```

0000000000000640 <\_start>:

```
640: d503201f nop
644: d280001d mov x29, #0x0
648: d280001e mov x30, #0x0
64c: aa0003e5 mov x5, x0
650: f94003e1 ldr x1, [sp]
654: 910023e2 add x2, sp, #0x8
658: 910003e6 mov x6, sp
65c: f00000e0 adrp x0, 0x1f000
660: f947ec00 ldr x0, [x0, #0xfd8]
664: d2800003 mov x3, #0x0
668: d2800004 mov x4, #0x0
66c: 97ffffe1 bl 0x5f0 <__libc_start_main@plt>
670: 97ffffec bl 0x620 <abort@plt>
```



# Program Representation: **Disassembly**

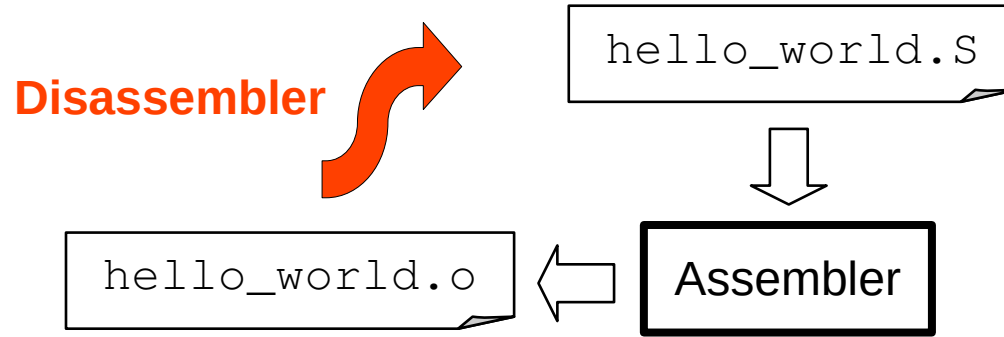


## Challenges:

- variable-length instructions
- padding bytes
- overlapping instructions
- self-modifying code
- inline data
- thumb code
- code discovery (code vs. data)
- unknown base address

=> *meaningless* disassembly

# Program Representation: **Disassembly**



Challenges:

- variable-length instructions
- padding bytes

- inline data
- thumb code
- code discovery (code vs. data)
- unknown base address

out of scope

~~overlapping instructions~~  
~~self-modifying code~~

=> *meaningless* disassembly

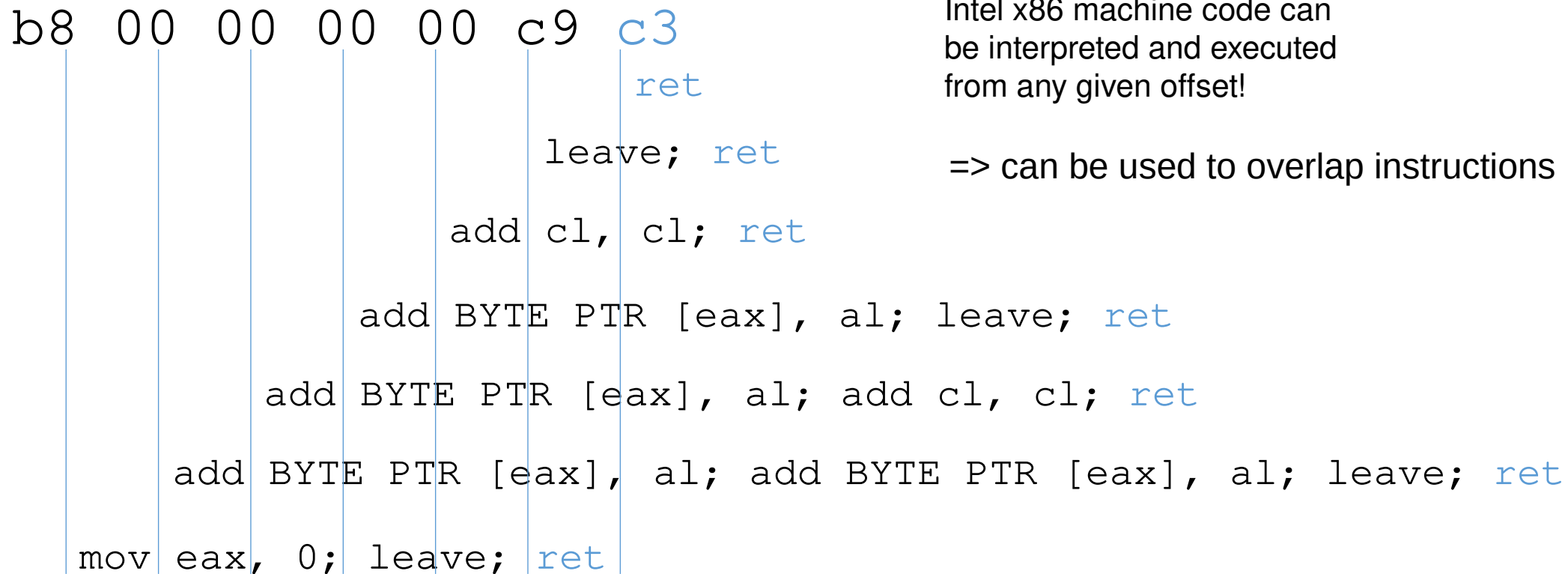
# Inline Data

Example: inline jump tables  
-> what is its size?

```
0000315e      2101      movs    r1, #1
00003160      e8dff000     tbb [pc, r0]
00003164      030e      lsls     r6, r1, #12
00003166      0907      lsrs     r7, r0, #4
00003168      050b      lsls     r3, r1, #20
0000316a      2106      movs     r1, #6
0000316c      e008      b.n 0x3180
0000316e      2102      movs     r1, #2
00003170      e006      b.n 0x3180
00003172      2103      movs     r1, #3
00003174      e004      b.n 0x3180
00003176      2104      movs     r1, #4
00003178      e002      b.n 0x3180
0000317a      2105      movs     r1, #5
0000317c      e000      b.n 0x3180
0000317e      2100      movs     r1, #0
00003180      4608      mov r0, r1
00003182      4770      bx  lr
```

```
0000315e      2101      movs    r1, #1      ; ret = default value
00003160      e8dff000     tbb [pc, r0]      ; switch (r0)
; jump table, byte-sized offsets
00003164      03 0e 09 07 05 0b
; case 1: (0x3164 + 0x3 * 2)
0000316a      2106      movs     r1, #6      ; ret = 6
0000316c      e008      b.n 0x3180      ; break
; case 5: (0x3164 + 0x5 * 2)
0000316e      2102      movs     r1, #2      ; ret = 2
00003170      e006      b.n 0x3180      ; break
; case 2: (0x3164 + 0x7 * 2)
00003172      2103      movs     r1, #3
00003174      e004      b.n 0x3180
; case 3: (0x3164 + 0x9 * 2)
00003176      2104      movs     r1, #4
00003178      e002      b.n 0x3180
; case 4: (0x3164 + 0xb * 2)
0000317a      2105      movs     r1, #5
0000317c      e000      b.n 0x3180
; default:
0000317e      2100      movs     r1, #0
; case 0: (0x3164 + 0xe * 2)
; end switch
00003180      4608      mov r0, r1      ; mov ret to r0 (return value)
00003182      4770      bx  lr      ; return
```

# Variable-length Instructions



# Linear-sweep

**Idea:** disassemble a stream of bytes into successive instructions

...

48 8b 44 24

04 53 eb 01

12 e8 00 00

00 00 48 83

04 24 42 c3

...



• 0 • 48 8b 44 24 04

• 5 • 53

• 6 • eb 01

• 8 • 12 e8

• 9 • 00 00

• mov

• push

• jmp

• adc

• add

• rax, [rsp+4]

• rbx

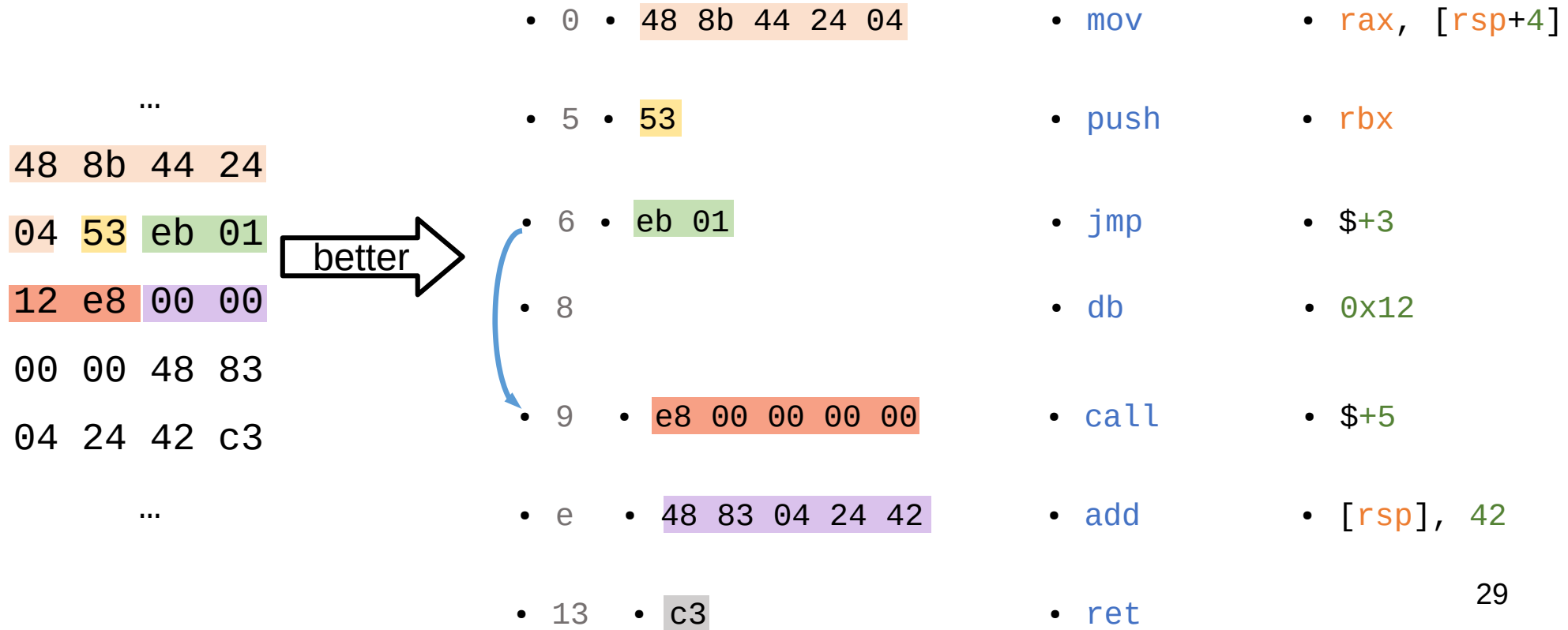
• \$+3

• ch, al

• [rax], al

# Recursive Traversal

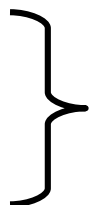
Idea: follow the control flow



# Overlapping Instructions

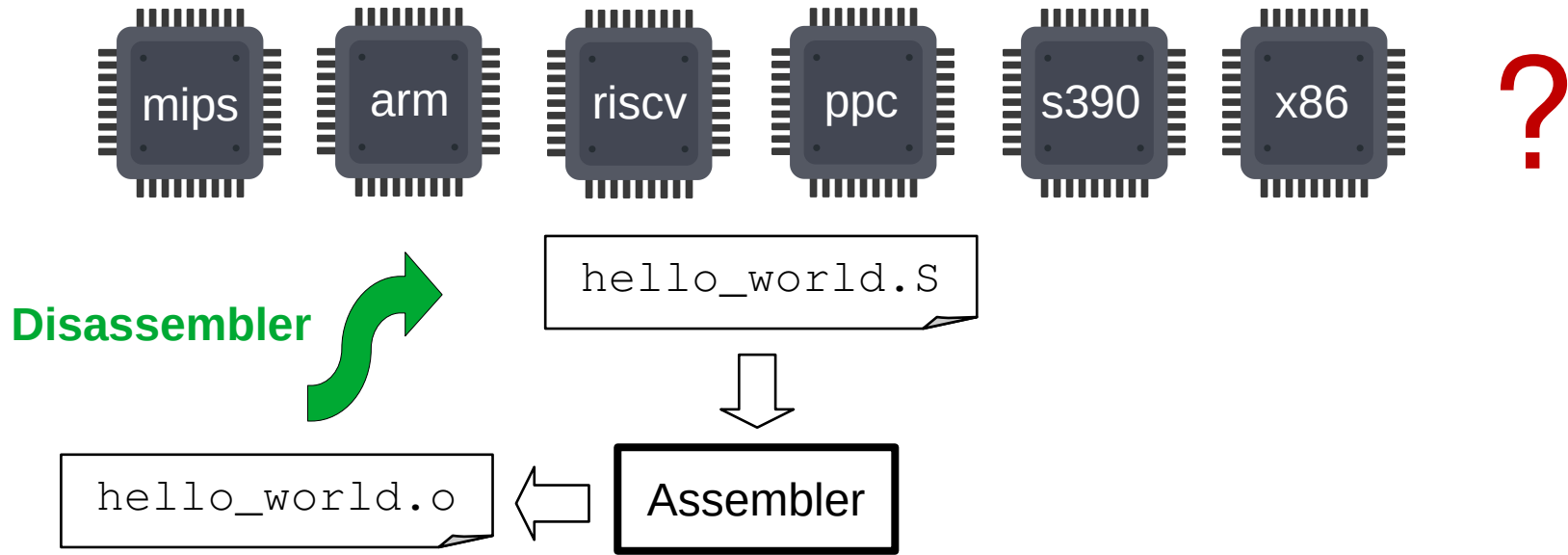
Recursive disassembler might follow zero-branch first and produce meaningless disassembly.

```
MOV EAX, 0x1  
TEST EAX, EAX  
JZ foo+1  
foo:  
CALL something
```

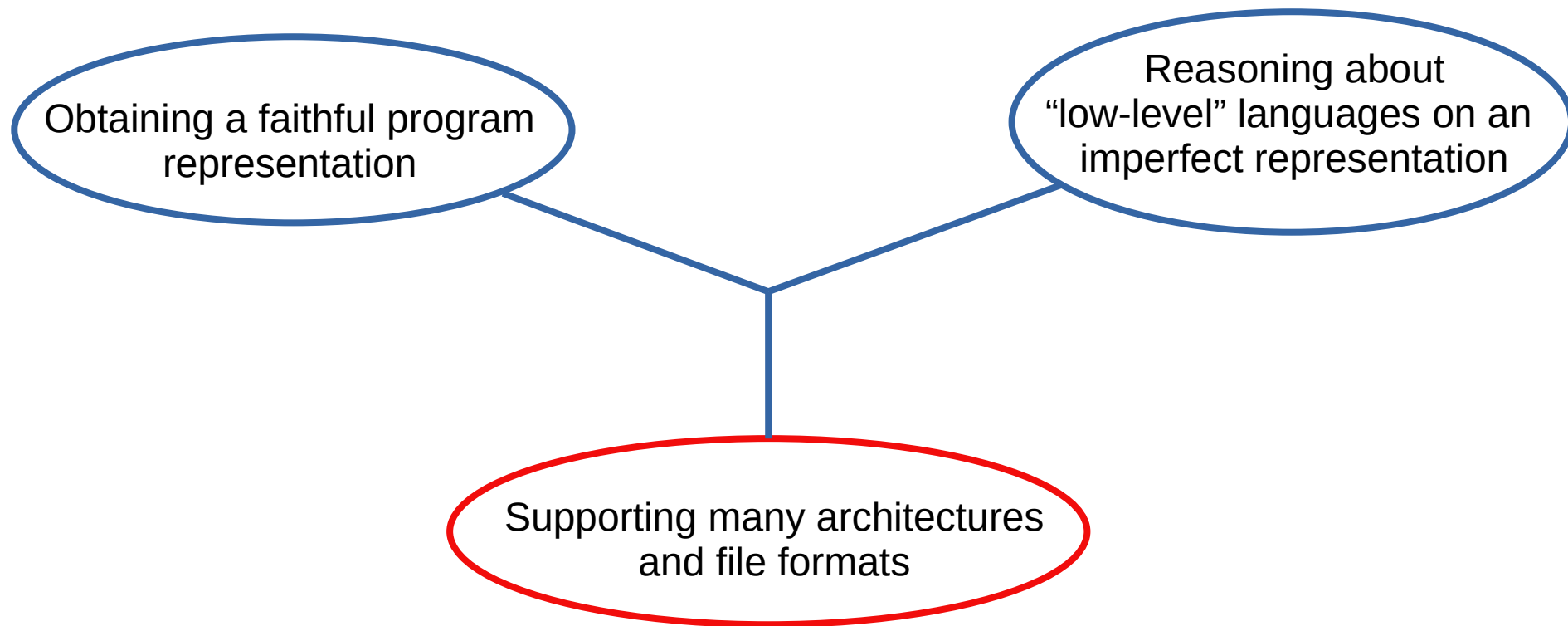


Example of “opaque predicate” -> not in scope.

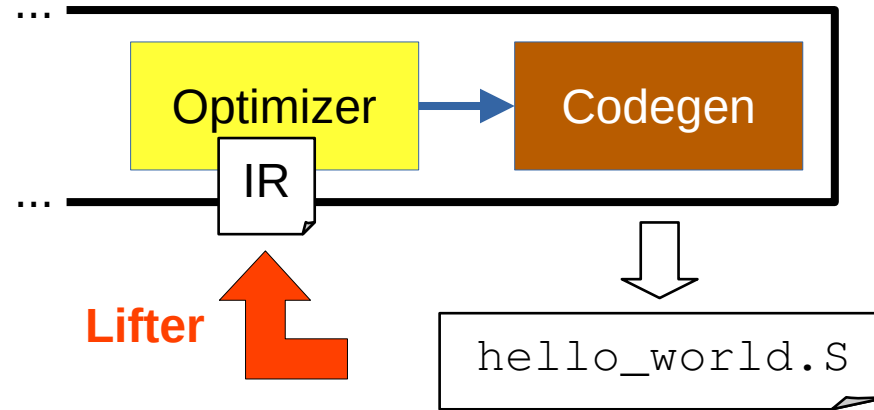




# So ... what makes this hard(er)?



# Program Representation: **Lifting**



Challenges:

- complex instructions
- “many instructions x many architectures = too much work”
- choosing appropriate abstract machine model

# Intermediate Representations

HOW STANDARDS PROLIFERATE:  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)



esil

VEX IR

reil

BN{LLIL,MLIL  
,HLIL}

{high,low}pcode

llvm IR

esil

VEX IR

reil

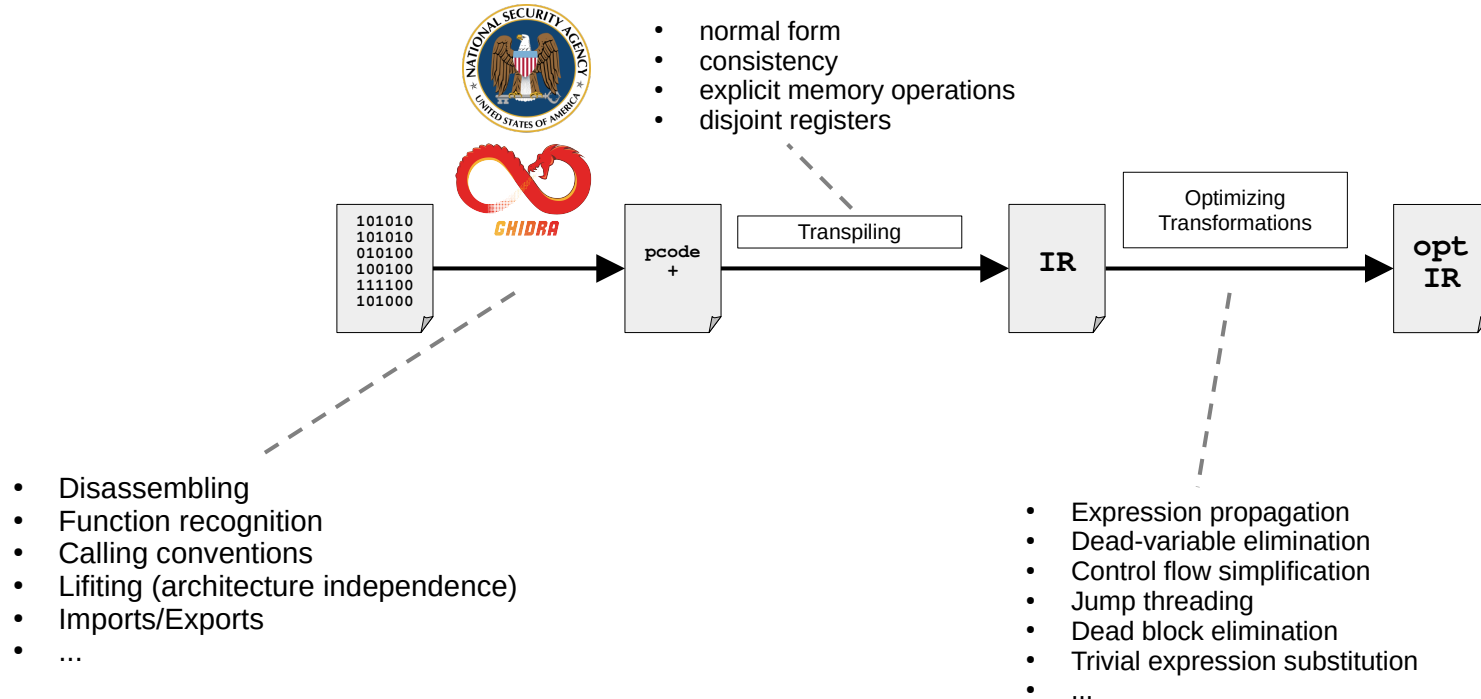
BN{LLIL,MLIL  
,HLIL}

{high,low}pcode

llvm IR

cwe\_checker IR

# cwe\_checker's Approach



# Ghidra: Pcode

Free & Open Source  
Software Reverse  
Engineering framework  
by the NSA.



```
00100764 fd 7b      stp      x29,x30,[sp, #local_10]!
           bf a9

           $U3a200:8 = COPY x29
           $U3a280:8 = COPY x30
           sp = INT_ADD sp, -16:8
           STORE ram(sp), $U3a200:8
           $U3a300:8 = INT_ADD sp, 8:8
           STORE ram($U3a300:8), $U3a280:8

00100768 fd 03      mov      x29,sp
           00 91

           x29 = COPY sp

0010076c 00 00      adrp     x0,0x100000
           00 90

           x0 = COPY 0x100000:8

00100770 00 80      add      x0=>s_Hello,_world._001007a0,... = "Hello, world."
           1e 91

           $U11e80:8 = COPY 0x7a0:8
           tmpCY = INT_CARRY x0, $U11e80:8
           tmpOV = INT_SCARRY x0, $U11e80:8
           $U11f80:8 = INT_ADD x0, $U11e80:8
           tmpNG = INT_SLESS $U11f80:8, 0:8
           tmpZR = INT_EQUAL $U11f80:8, 0:8
           x0 = COPY $U11f80:8
```

# Pcode to IR Translation

## Pcode

```
FUNCTION: main @ 0x00100764
BLOCK @ 00100764
0x00100764:>0:> (Register[0x000040e8]:8) Expr(COPY) -> Unique[0x00039800]:8
0x00100764:>1:> (Register[0x000040f0]:8) Expr(COPY) -> Unique[0x00039880]:8
0x00100764:>2:> (Register[0x00000008]:8, Const[0xfffffffffffff0]:8) Expr(INT_ADD) -> Register[0x00000008]:8
0x00100764:>3:> (Const[0x000001b1]:8, Register[0x00000008]:8, Unique[0x00039800]:8) Expr(STORE)
0x00100764:>4:> (Register[0x00000008]:8, Const[0x00000008]:8) Expr(INT_ADD) -> Unique[0x00039900]:8
0x00100764:>5:> (Const[0x000001b1]:8, Unique[0x00039900]:8, Unique[0x00039880]:8) Expr(STORE)
0x00100768:>0:> (Register[0x00000008]:8) Expr(COPY) -> Register[0x000040e8]:8
0x0010076c:>0:> (Const[0x00100000]:8) Expr(COPY) -> Register[0x00004000]:8
0x00100770:>0:> (Const[0x000007a0]:8) Expr(COPY) -> Unique[0x00011e80]:8
0x00100770:>1:> (Register[0x00004000]:8, Unique[0x00011e80]:8) Expr(INT_CARRY) -> Register[0x00000105]:1
0x00100770:>2:> (Register[0x00004000]:8, Unique[0x00011e80]:8) Expr(INT_SCARRY) -> Register[0x00000106]:1
0x00100770:>3:> (Register[0x00004000]:8, Unique[0x00011e80]:8) Expr(INT_ADD) -> Unique[0x00011f80]:8
0x00100770:>4:> (Unique[0x00011f80]:8, Const[0x00000000]:8) Expr(INT_SLESS) -> Register[0x00000107]:1
0x00100770:>5:> (Unique[0x00011f80]:8, Const[0x00000000]:8) Expr(INT_EQUAL) -> Register[0x00000108]:1
0x00100770:>6:> (Unique[0x00011f80]:8) Expr(COPY) -> Register[0x00004000]:8
0x00100774:>0:> (Const[0x00100774]:8, Const[0x00000004]:8) Expr(INT_ADD) -> Register[0x000040f0]:8
0x00100774:>1:> (Ram[0x00100630]:8) Jmp(CALL)
BLOCK @ 00100778
0x0010077c:>0:> (Register[0x00000008]:8) Expr(COPY) -> Unique[0x00007c80]:8
0x0010077c:>1:> (Register[0x00000008]:8, Const[0x0000010]:8) Expr(INT_ADD) -> Register[0x00000008]:8
0x0010077c:>2:> (Const[0x000001b1]:8, Unique[0x00007c80]:8) Expr(LOAD) -> Register[0x000040e8]:8
0x0010077c:>3:> (Unique[0x00007c80]:8, Const[0x00000008]:8) Expr(INT_ADD) -> Unique[0x00023e80]:8
0x0010077c:>4:> (Const[0x000001b1]:8, Unique[0x00023e80]:8) Expr(LOAD) -> Register[0x000040f0]:8
0x00100780:>0:> (Register[0x000040f0]:8) Expr(COPY) -> Register[0x00000000]:8
0x00100780:>1:> (Register[0x00000000]:8) Jmp(RETURN)
```

## Unoptimized IR

```
FN [fun_0x00100764] name:main entry:yes non_returning:no
BLK [blk_0x00100764]
DEF [instr_0x00100764_0] $U_0x00039800:8(temp) = x29:8
DEF [instr_0x00100764_1] $U_0x00039880:8(temp) = x30:8
DEF [instr_0x00100764_2] sp:8 = (sp:8 + 0xfffffffffffff0:8)
DEF [instr_0x00100764_3] Store at sp:8 := $U_0x00039800:8(temp)
DEF [instr_0x00100764_4] $U_0x00039900:8(temp) = (sp:8 + 0x8:8)
DEF [instr_0x00100764_5] Store at $U_0x00039900:8(temp) := $U_0x00039880:8(temp)
DEF [instr_0x00100768_0] x29:8 = sp:8
DEF [instr_0x0010076c_0] x0:8 = 0x100000:8
DEF [instr_0x00100770_0] $U_0x00011e80:8(temp) = 0x7a0:8
DEF [instr_0x00100770_1] tmpCY:1 = (x0:8 IntCarry $U_0x00011e80:8(temp))
DEF [instr_0x00100770_2] tmpOV:1 = (x0:8 IntSCarry $U_0x00011e80:8(temp))
DEF [instr_0x00100770_3] $U_0x00011f80:8(temp) = (x0:8 + $U_0x00011e80:8(temp))
DEF [instr_0x00100770_4] tmpNG:1 = ($U_0x00011f80:8(temp) < 0x0:8)
DEF [instr_0x00100770_5] tmpZR:1 = ($U_0x00011f80:8(temp) == 0x0:8)
DEF [instr_0x00100770_6] x0:8 = $U_0x00011f80:8(temp)
DEF [instr_0x00100774_0] x30:8 = (0x100774:8 + 0x4:8)
JMP [instr_0x00100774_1] call ext_fun_puts ret blk_0x00100778
BLK [blk_0x00100778]
DEF [instr_0x0010077c_0] $U_0x00007c80:8(temp) = sp:8
DEF [instr_0x0010077c_1] sp:8 = (sp:8 + 0x10:8)
DEF [instr_0x0010077c_2] x29:8 := Load from $U_0x00007c80:8(temp)
DEF [instr_0x0010077c_3] $U_0x00023e80:8(temp) = ($U_0x00007c80:8(temp) + 0x8:8)
DEF [instr_0x0010077c_4] x30:8 := Load from $U_0x00023e80:8(temp)
DEF [instr_0x00100780_0] pc:8 = x30:8
JMP [instr_0x00100780_1] ret pc:8
```



# IR Optimization

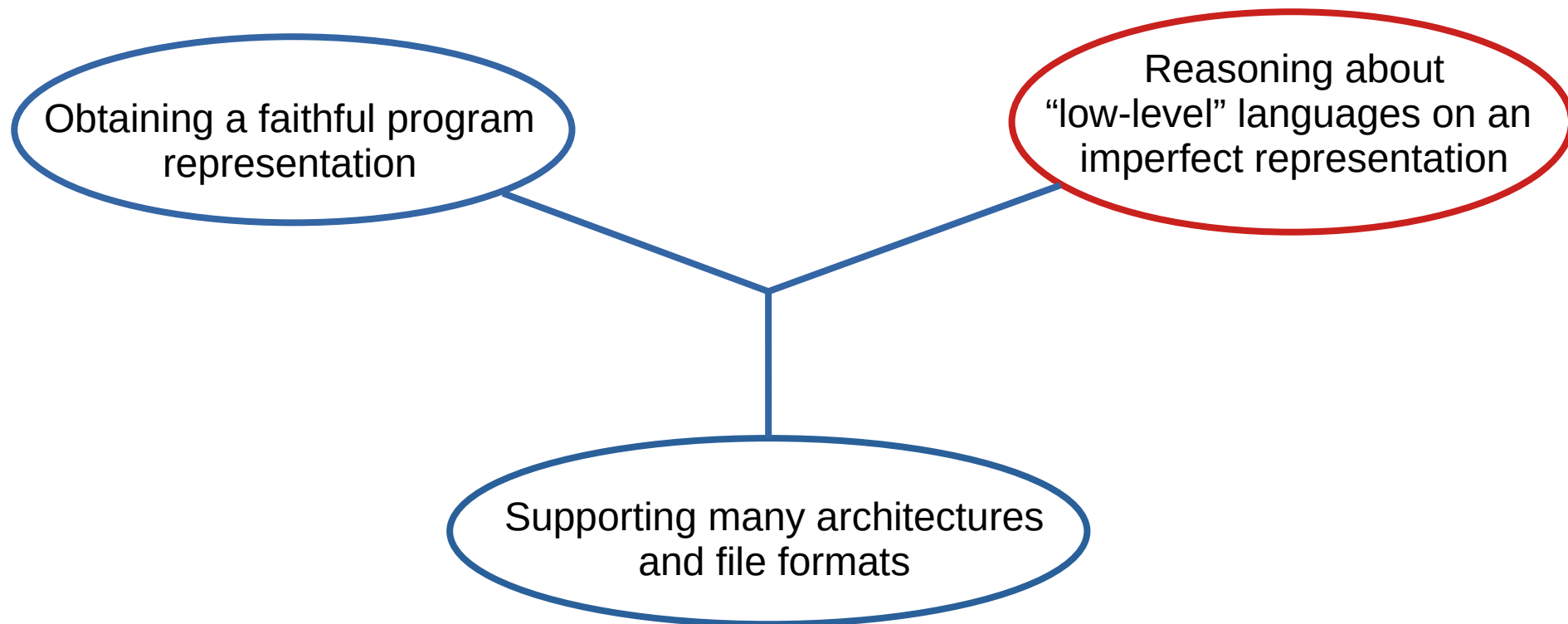
## Unoptimized IR

```
FN [fun_0x00100764] name:main entry:yes non_returning:no
BLK [blk_0x00100764]
  DEF [instr_0x00100764_0] $U_0x00039800:8(temp) = x29:8
  DEF [instr_0x00100764_1] $U_0x00039880:8(temp) = x30:8
  DEF [instr_0x00100764_2] sp:8 = (sp:8 + 0xffffffffffffff0:8)
  DEF [instr_0x00100764_3] Store at sp:8 := $U_0x00039800:8(temp)
  DEF [instr_0x00100764_4] $U_0x00039900:8(temp) = (sp:8 + 0x8:8)
  DEF [instr_0x00100764_5] Store at $U_0x00039900:8(temp) := $U_0x00039880:8(temp)
  DEF [instr_0x00100768_0] x29:8 = sp:8
  DEF [instr_0x0010076c_0] x0:8 = 0x100000:8
  DEF [instr_0x00100770_0] $U_0x00011e80:8(temp) = 0x7a0:8
  DEF [instr_0x00100770_1] tmpCY:1 = (x0:8 IntCarry $U_0x00011e80:8(temp))
  DEF [instr_0x00100770_2] tmpOV:1 = (x0:8 IntSCarry $U_0x00011e80:8(temp))
  DEF [instr_0x00100770_3] $U_0x00011f80:8(temp) = (x0:8 + $U_0x00011e80:8(temp))
  DEF [instr_0x00100770_4] tmpNG:1 = ($U_0x00011f80:8(temp) < 0x0:8)
  DEF [instr_0x00100770_5] tmpZR:1 = ($U_0x00011f80:8(temp) == 0x0:8)
  DEF [instr_0x00100770_6] x0:8 = $U_0x00011f80:8(temp)
  DEF [instr_0x00100774_0] x30:8 = (0x100774:8 + 0x4:8)
  JMP [instr_0x00100774_1] call ext_fun_puts ret blk_0x00100778
BLK [blk_0x00100778]
  DEF [instr_0x0010077c_0] $U_0x00007c80:8(temp) = sp:8
  DEF [instr_0x0010077c_1] sp:8 = (sp:8 + 0x10:8)
  DEF [instr_0x0010077c_2] x29:8 := Load from $U_0x00007c80:8(temp)
  DEF [instr_0x0010077c_3] $U_0x00023e80:8(temp) = ($U_0x00007c80:8(temp) + 0x8:8)
  DEF [instr_0x0010077c_4] x30:8 := Load from $U_0x00023e80:8(temp)
  DEF [instr_0x00100780_0] pc:8 = x30:8
  JMP [instr_0x00100780_1] ret pc:8
```

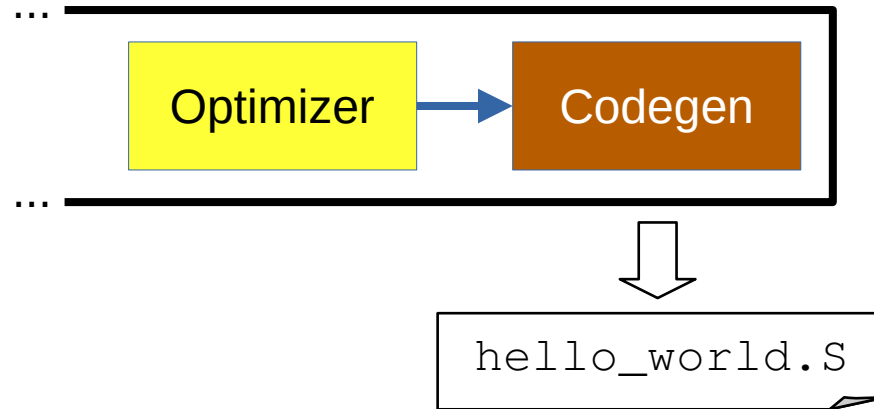
## Optimized IR

```
FN [fun_0x00100764] name:main entry:yes non_returning:no
BLK [blk_0x00100764]
  DEF [instr_0x00100764_2] sp:8 = (sp:8 + 0xffffffffffffff0:8)
  DEF [instr_0x00100764_3] Store at sp:8 := x29:8
  DEF [instr_0x00100764_5] Store at (sp:8 + 0x8:8) := x30:8
  DEF [instr_0x00100768_0] x29:8 = sp:8
  DEF [instr_0x00100770_1] tmpCY:1 = (0x100000:8 IntCarry 0x7a0:8)
  DEF [instr_0x00100770_2] tmpOV:1 = (0x100000:8 IntSCarry 0x7a0:8)
  DEF [instr_0x00100770_4] tmpNG:1 = (0x1007a0:8 < 0x0:8)
  DEF [instr_0x00100770_5] tmpZR:1 = (0x1007a0:8 == 0x0:8)
  DEF [instr_0x00100770_6] x0:8 = 0x1007a0:8
  DEF [instr_0x00100774_0] x30:8 = 0x100778:8
  JMP [instr_0x00100774_1] call ext_fun_puts ret blk_0x00100778
BLK [blk_0x00100778]
  DEF [instr_0x0010077c_0] $U_0x00007c80:8(temp) = sp:8
  DEF [instr_0x0010077c_1] sp:8 = (sp:8 + 0x10:8)
  DEF [instr_0x0010077c_2] x29:8 := Load from $U_0x00007c80:8(temp)
  DEF [instr_0x0010077c_4] x30:8 := Load from ($U_0x00007c80:8(temp) + 0x8:8)
  DEF [instr_0x00100780_0] pc:8 = x30:8
  JMP [instr_0x00100780_1] ret x30:8
```

# So ... what makes this hard(er)?



# Loss of Information/Structure



Many concepts that help with analyzing source languages are “lost” during code generation:

- type annotations, type definitions
- local/global variables, function parameters, function return values
- functions, loops, structured control flow

# Static Program Analysis

*“The art of reasoning about the possible runtime behaviors of programs without executing them.”*

Comprehension

e.g. all call sites of a function

Verification

e.g. borrowck

Bug Detection

e.g. reading uninit. memory

Optimization

e.g. loop invariant expressions

Testing

e.g. can this assertion fail?

# Static Program Analysis

*“The art of reasoning about the possible runtime behaviors of programs without executing them.”*

but

Thm. (Rice): *“Any nontrivial semantic property of a program is undecidable.”*



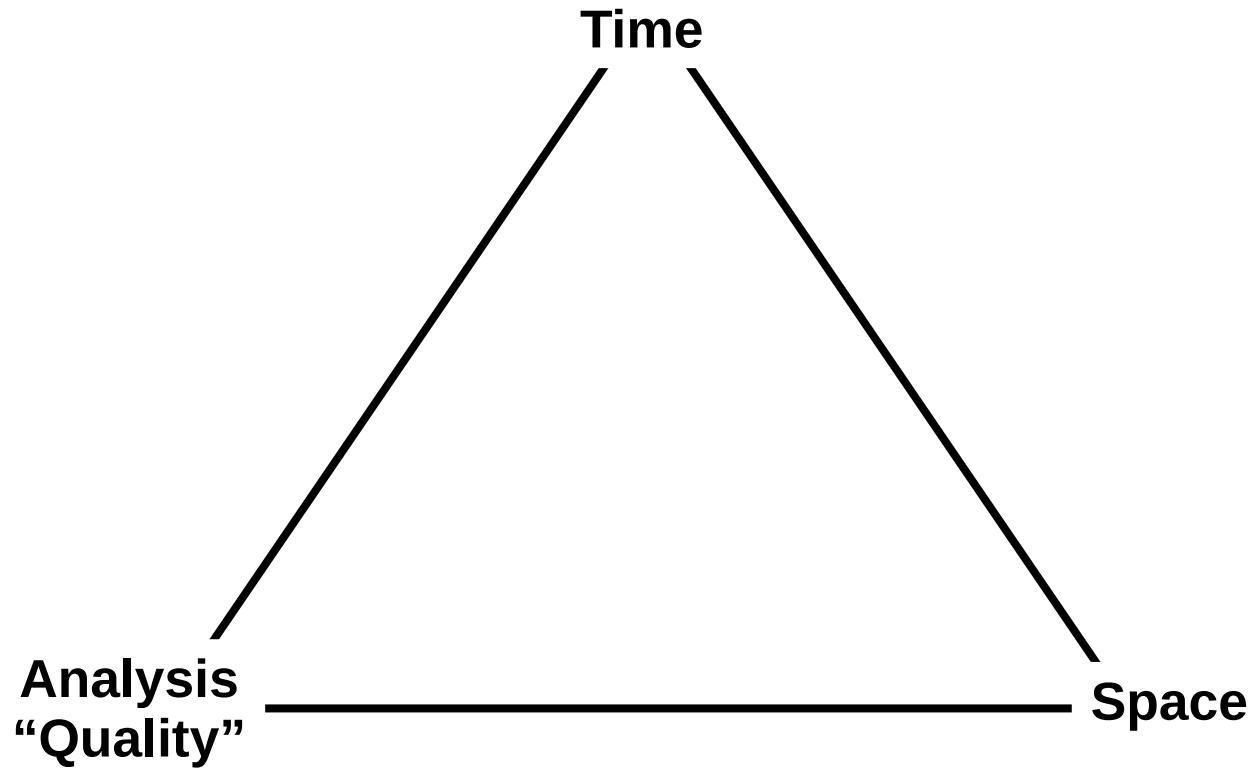
There cannot be a “perfect” static analysis (Could solve Halting Problem...).



Analysis results will in general always be an approximation.

Thm. (Full employment theorem for static analysis designers.) There is no universally ideal approximation.

# Static Program Analysis



# Analysis Properties

Complete

*“The analysis may **miss** some **facts**.”*

Sound

*“The analysis may report **additional** facts.”*

“Soundy” aka.  
Incomplete & Unsound



Testing

- assertion may still fail at runtime
- + no phony test failures

Bug Detection

- > some bugs are missed
- > some false warnings are generated

Verification

- some well-behaved programs are rejected
- + no ill-behaved programs are accepted

Optimization

- some optimizations are missed
- + no misscompilations

Sound & Complete  
Comprehension

- > refactorings don't break code (mostly *syntactic* properties)

# Formulation Approaches

Dataflow Analysis

Type and Effect Systems

Symbolic Execution

Abstract Interpretation

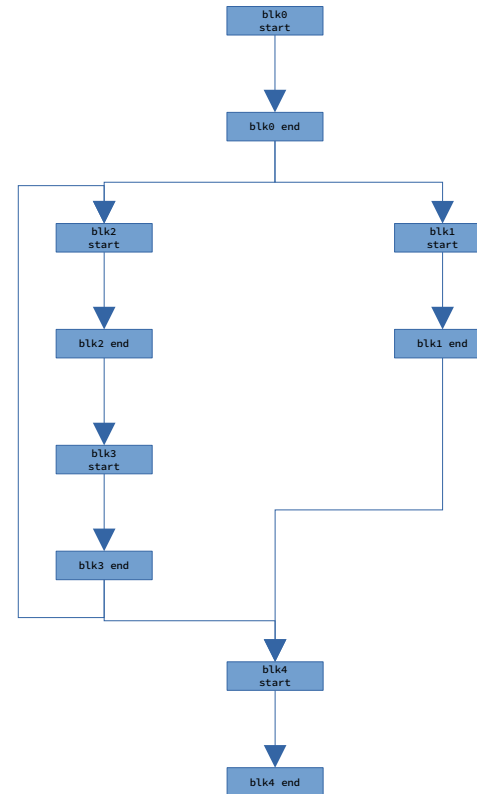
Constraint Based Analysis



# Recipe: Dataflow Analysis

Control Flow Graph (CFG)

Direction ▼

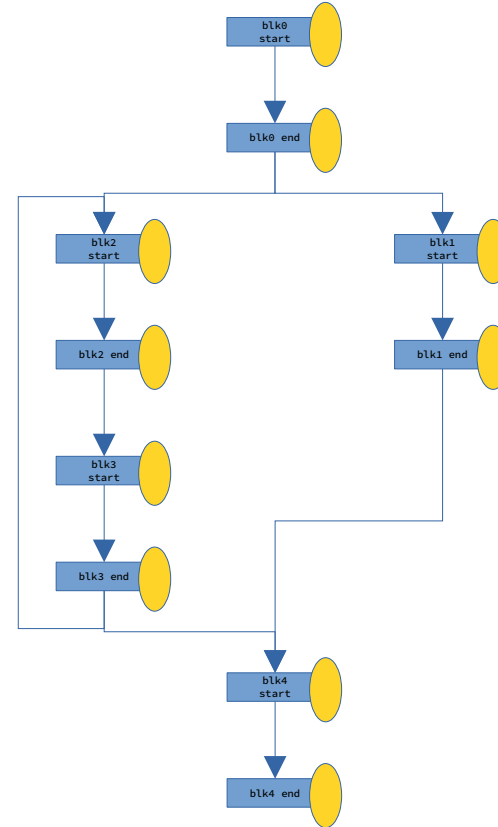


# Recipe: Dataflow Analysis

Control Flow Graph (CFG)

Direction ▼

Property Space ○



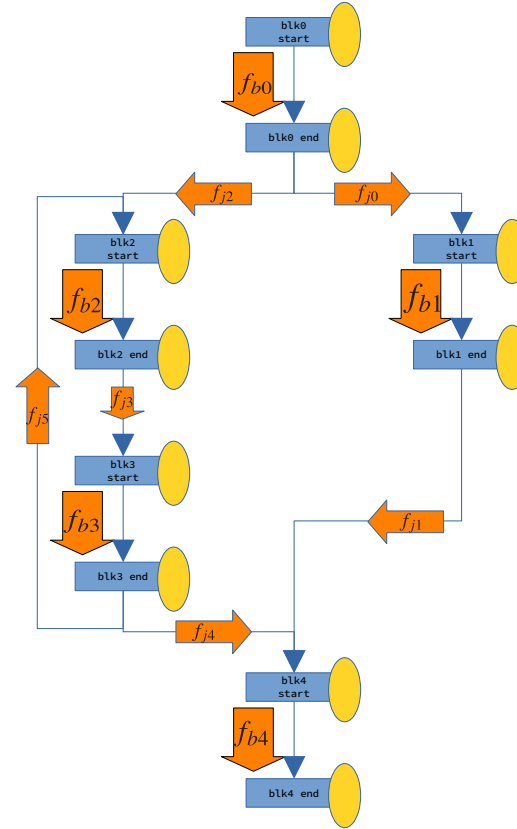
# Recipe: Dataflow Analysis

Control Flow Graph (CFG)

Direction 

Property Space 

Transfer Functions 



# Recipe: Dataflow Analysis

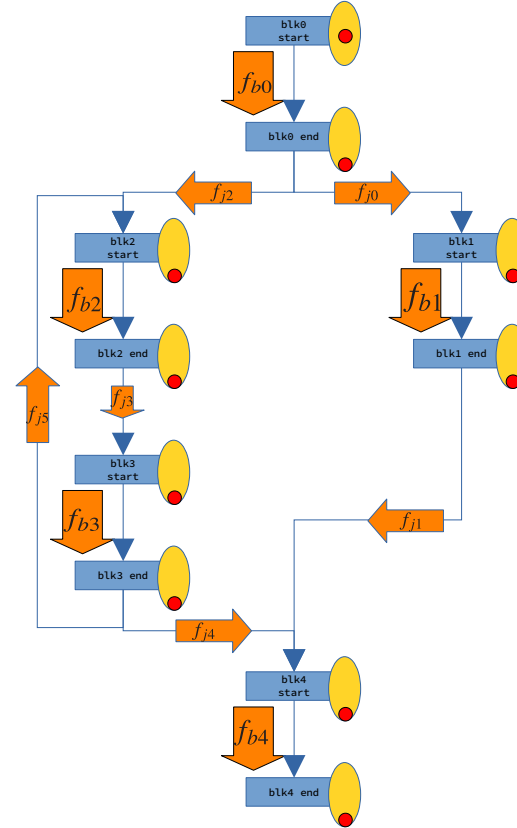
Control Flow Graph (CFG)

Direction ▼

Property Space ○

Transfer Functions →

Initial State •



# Defining a new Dataflow Analysis

1. Define a type that represents your analysis, e.g., `struct MyAnalysis`.
2. Implement a trait to define the direction, property space & transfer functions of your analysis:

```
trait analysis::fixpoint::Context
```



blanket impl

```
trait analysis::forward_interprocedural_fixpoint::Context
```

(`abstract_domain` has Types and traits that help with defining property spaces & transfer functions.)

# Solving a Dataflow Problem

1. Instantiate `MyAnalysis` on a concrete program's CFG.
2. Create an `analysis::fixpoint::Computation` by adding an initial state.
3. Run the computation to obtain a minimal fixed point solution to the dataflow problem.

# Try it out Yourself

Docs



Code

