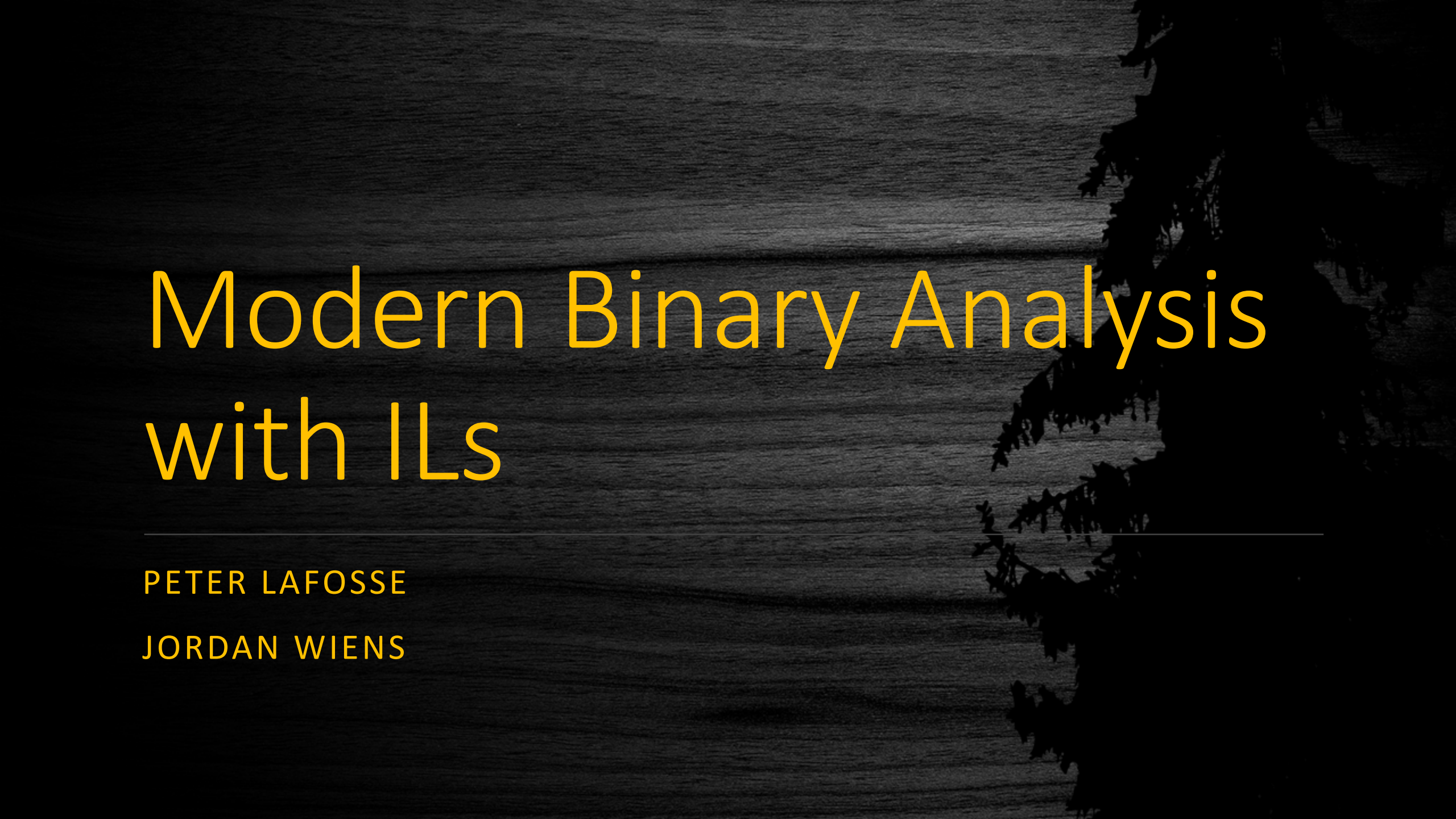

BLUEHAT
SEATTLE 2019



Modern Binary Analysis with ILs

PETER LAFOSSE

JORDAN WIENS

Us

PETER LAFOSSE

- Founder, Vector 35
- Former: Head of Vulnerability Research at Raytheon SIGovs
- Current: Project Manager and developer of Binary Ninja and reverse engineer

JORDAN WIENS

- Founder, Vector 35
- Former: network security engineer, incident responder, reverse engineer, vulnerability researcher, CTF player
- Current: a hacker learning to dev

VECTOR 35

BINARY NINJA

You?

Done binary reverse engineering

Used a decompiler

Written code to automate RE

Used an IL or IR for RE

Used an IL or IR for compilation or other task

Published research leveraging ILs

Outline

What is Binary Analysis

Why ILs

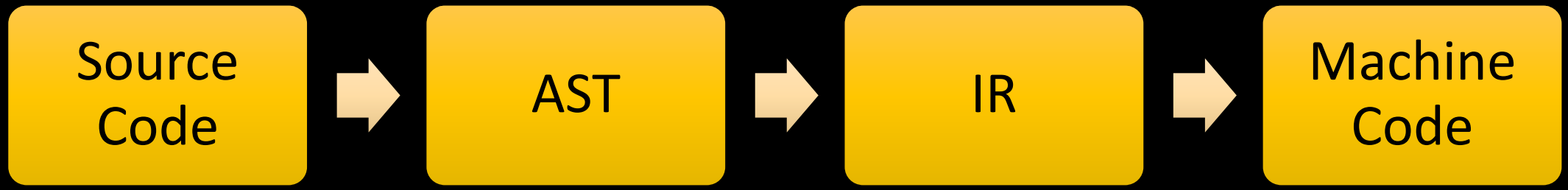
IL overview

DEMOs

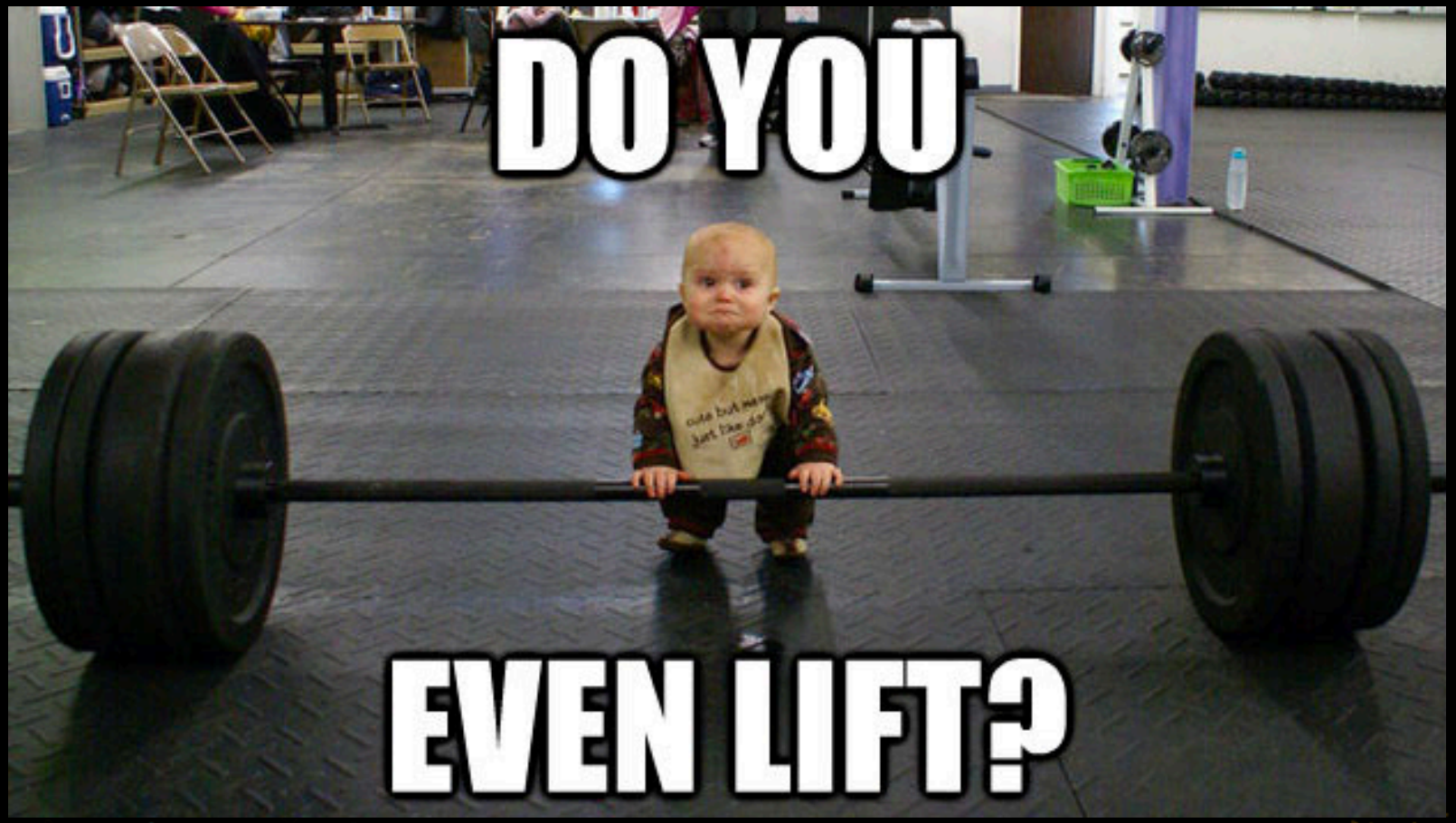
What?

WHAT IS BINARY ANALYSIS?

Compilation



Decompilation/Lifting



Static vs Dynamic

Many tradeoffs

Focus on Static

Binary Analysis != Source Analysis

All we have to do is decompile the binary… Right?

Binary Analysis [illegible] Analysis

All we hav[illegible] do is [illegible]mpile the [illegible]ary… Right?

Compilers mess everything up

Register Allocation

Function Calling Conventions

Variable and Function Names

Types

Compilers mess everything up

Register Allocation

Function Calling Conventions

~~Variable and Function Names~~

~~Types~~

Undecidable Problems

Where are all the…

functions?

strings?

pointers?

Unique Failure conditions

Stack variable resolution fails

Parameter resolution fails

Switch resolution fails

Misidentification of functions

Why?

WHY ILS?

Before we begin

IL vs **IR**

Intermediate Language (IL)

Intermediate Representation (IR)

Bitcode

Virtual Machine Opcodes

P-Code

Premise

Reverse Engineering is fundamental to understanding how software works.

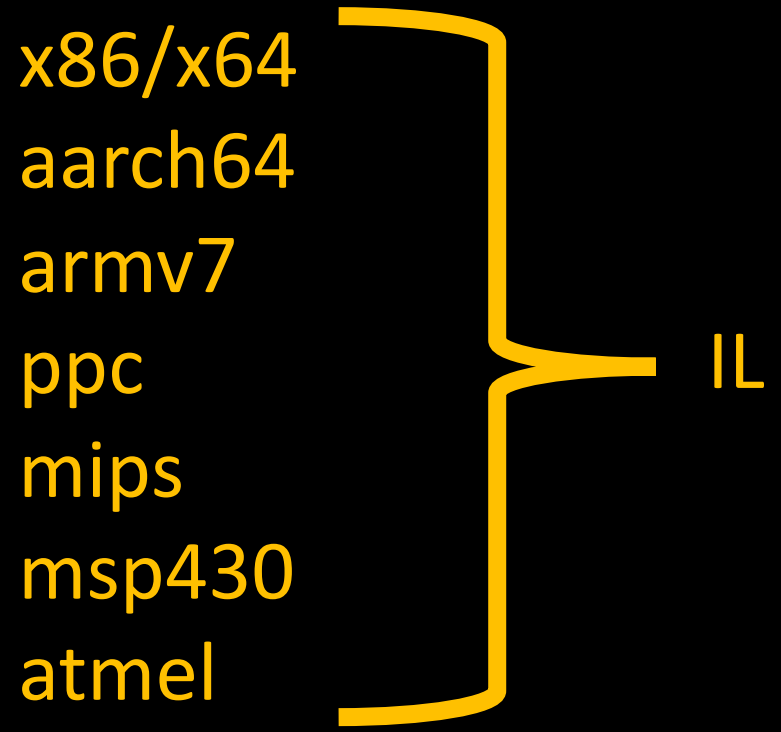
Intermediate Languages are fundamental to modern compiler design.

Intermediate Languages should, therefore, be fundamental to how reverse engineering works.

Smaller Instruction Set

Instruction Set	Number of instructions
P-Code (Ghidra)	62
Microcode (IDA)	72
RISC-V	72
LLIL (Binary Ninja)	106
MIPS	166
ARMv7	424
X86/x64	>1000

Architecture Agnostic



More robust, faster, easier

THE DISASSEMBLY WAY

```
for index, item in enumerate(ins):

  count = 0

  if 'svc' in ''.join(map(str,
ins[index])):

    for iter in ins[index-1]:

      if count == 5:

        print "syscall: %s @ func:
%s " % (iter, func)

        count += 1
```

THE IL WAY

```
for i in mlil_instructions:

  if i.operation == MLIL_SYSCALL:

    syscallNum = i.params[0].value
```

More robust, faster, easier

THE DISASSEMBLY WAY

```
for index, item in enumerate(ins):

  count = 0

  if 'svc' in ''.join(map(str,
ins[index])):

    for iter in ins[index-1]:

      if count == 5:

        print "syscall: %s @ func:
%s " % (iter, func)

        count += 1
```

THE IL WAY

```
for i in mlil_instructions:

  if i.operation == MLIL_SYSCALL:

    syscallNum = i.params[0].value
```

More robust, faster, easier

THE DISASSEMBLY WAY

```
for index, item in enumerate(ins):

  count = 0

  if 'svc' in ''.join(map(str,
ins[index])):

    for iter in ins[index-1]:

      if count == 5:

        print "syscall: %s @ func:
%s " % (iter, func)

        count += 1
```

THE IL WAY

```
for i in mlil_instructions:

  if i.operation == MLIL_SYSCALL:

    syscallNum = i.params[0].value
```

Why not a decompiler?

Missing compound types thwarts analysis

Abstractions increase errors in translations

Why not C?

Stack layout

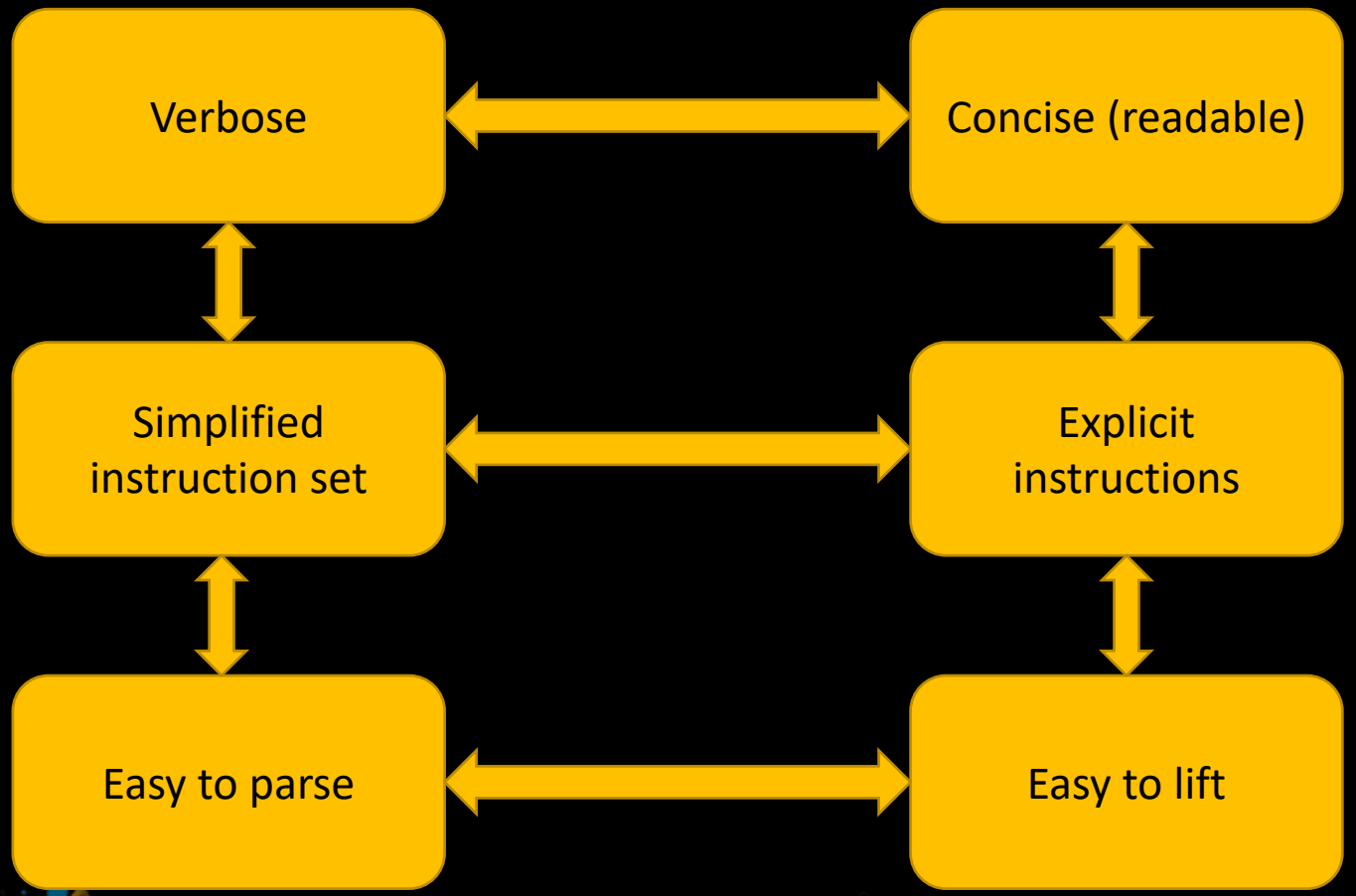
Variable aliasing

Semantic bindings between variables

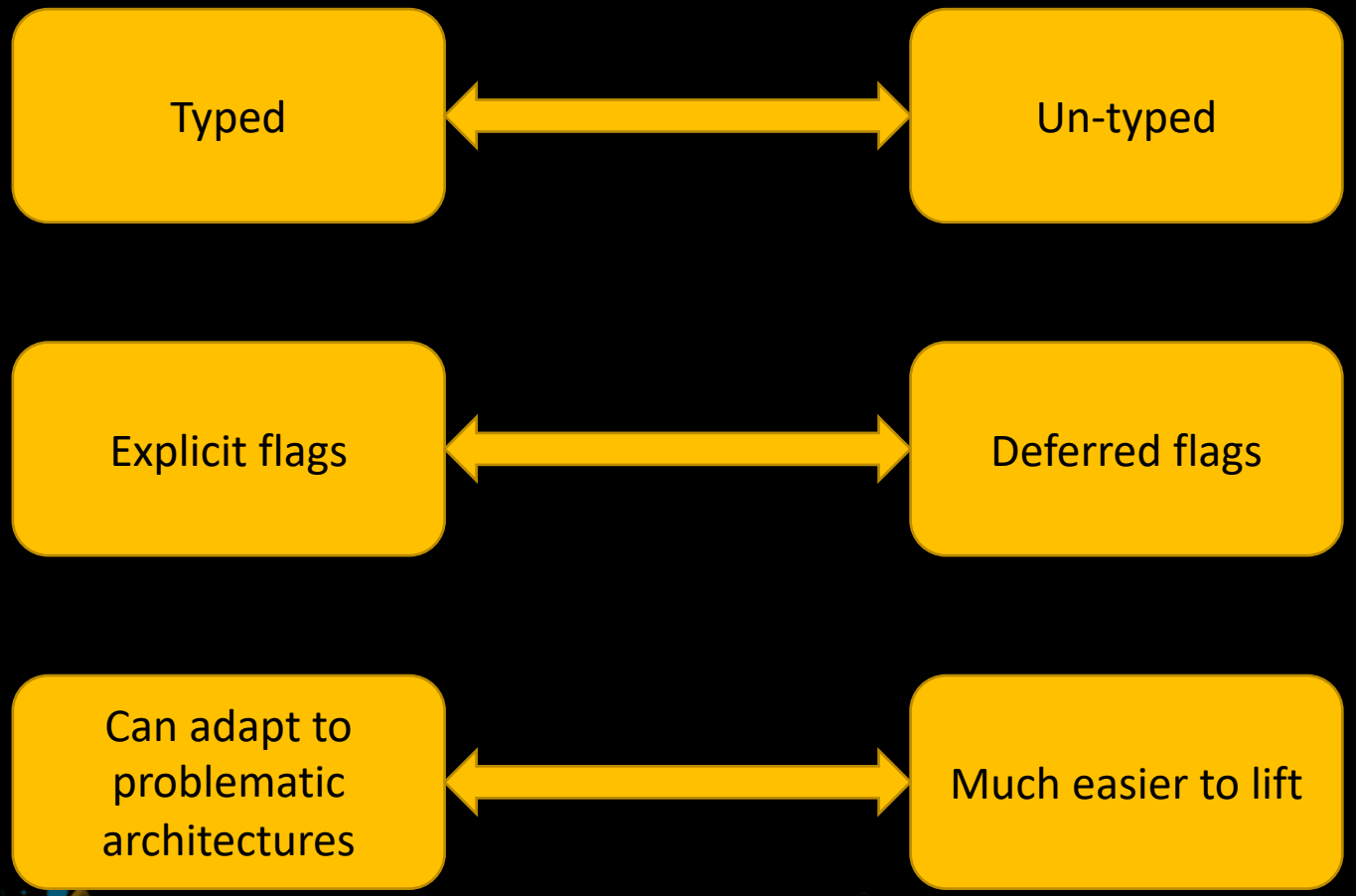
IL Overview

OR: TOO MANY ~~SECRETS~~ INTERMEDIATE LANGUAGES

Tradeoffs


Verbose
Concise (readable)
Simplified instruction set
Explicit instructions
Easy to parse
Easy to lift

Tradeoffs Pt 2.



Verbose, Simple Instructions

```
test eax, eax
```

```
00000000.00 STR R_EAX:32, , V_00:32
00000000.01 STR 0:1, , R_CF:1
00000000.02 AND V_00:32, ff:8, V_01:8
00000000.03 SHR V_01:8, 7:8, V_02:8
00000000.04 SHR V_01:8, 6:8, V_03:8
00000000.05 XOR V_02:8, V_03:8, V_04:8
00000000.06 SHR V_01:8, 5:8, V_05:8
00000000.07 SHR V_01:8, 4:8, V_06:8
00000000.08 XOR V_05:8, V_06:8, V_07:8
00000000.09 XOR V_04:8, V_07:8, V_08:8
00000000.0a SHR V_01:8, 3:8, V_09:8
00000000.0b SHR V_01:8, 2:8, V_10:8
00000000.0c XOR V_09:8, V_10:8, V_11:8
00000000.0d SHR V_01:8, 1:8, V_12:8
00000000.0e XOR V_12:8, V_01:8, V_13:8
00000000.0f XOR V_11:8, V_13:8, V_14:8
00000000.10 XOR V_08:8, V_14:8, V_15:8
00000000.11 AND V_15:8, 1:1, V_16:1
00000000.12 NOT V_16:1, , R_PF:1
00000000.13 STR 0:1, , R_AF:1
00000000.14 EQ V_00:32, 0:32, R_ZF:1
00000000.15 SHR V_00:32, 1f:32, V_17:32
00000000.16 AND 1:32, V_17:32, V_18:32
00000000.17 EQ 1:32, V_18:32, R_SF:1
00000000.18 STR 0:1, , R_OF:1
```

Concise, Many Instructions

```
fld1
```

```
x87.push{x87c1z}(float.t(1))
```

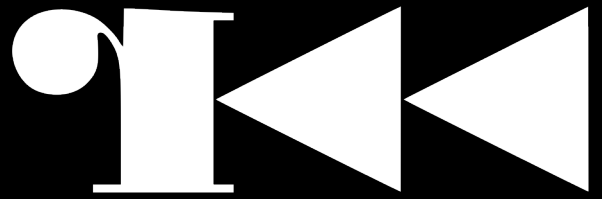
Landscape of ILs

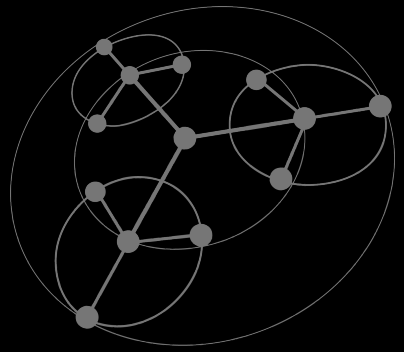
Name	Project	URL
BIL	BAP	https://github.com/BinaryAnalysisPlatform/bap
BNIL	Binary Ninja	http://docs.binary.ninja/dev/bnil-llil.html
Boogie	Boogie	https://www.microsoft.com/en-us/research/project/boogie-an-intermediate-verification-language/
Cas	Amoco	https://github.com/bdcht/amoco/blob/release/amoco/cas/expressions.py
DBA	BINSEC	https://link.springer.com/chapter/10.1007%2F978-3-662-46681-0_17
ESIL	Radare	https://github.com/radare/radare2/wiki/ESIL
Falcon IL	Falcon	https://github.com/falconre/falcon
FalkerIL	Falker*	https://gamozolabs.github.io
GDSL	GDSL	https://github.com/gdslang/gdsl-toolkit
JEB IR	JEB	https://www.pnfsoftware.com/blog/jeb-native-pipeline-intermediate-representation/
LowUIR	B2R2	https://github.com/B2R2-org/B2R2
Miasm IR	Miasm	https://github.com/cea-sec/miasm
Microcode	Hex-Rays	https://hex-rays.com/products/ida/support/ppt/recon2018.ppt
Microcode	Insight	https://github.com/hotelzululima/insight
P-Code	GHIDRA	http://ghidra.re/courses/languages/html/pcoderef.html
REIL	BinNavi	https://www.zynamics.com/binnavi/manual/html/reil_language.htm
RREIL	Bindead	https://bitbucket.org/mihaila/bindead/wiki/Introduction%20to%20RREIL
SSL	Jakstab	http://www.jakstab.org/
TSL	CodeSonar and others	http://pages.cs.wisc.edu/~reps/past-research.html#TSL_overview
Unnamed	EiNSTeiN-	https://github.com/EiNSTeiN-/decompiler/tree/master/src/ir
VEX	Valgrind	https://github.com/smparkes/valgrind-vex/blob/master/pub/libvex_ir.h
Vine	BitBlaze	http://bitblaze.cs.berkeley.edu/vine.html

LLVM IR

Name	Project	URL
LLVM IR	LLVM	http://llvm.org/docs/LangRef.html
allin	allin	http://sdasgup3.web.engr.illinois.edu/Document/allin_poster.pdf
bin2llvm	S2E	https://github.com/cojocar/bin2llvm
Dagger	Dagger	https://github.com/repzret/dagger
fcd	fcd	https://github.com/zneak/fcd
Fracture™	Fracture™	https://github.com/draperlaboratory/fracture
libbeauty	libbeauty	https://github.com/jcdutton/reference
mctoll	mctoll	https://github.com/microsoft/llvm-mctoll
remill	McSema	https://github.com/trailofbits/mcsema
reopt	reopt	https://github.com/GaloisInc/reopt
RetDec	RetDec	https://github.com/avast/retdec
revng	revng	https://github.com/revng/revng

Landscape





BINARY NINJA

Landscape: LLVM IR

PROS

Leverages existing compiler infrastructure

Many analysis passes

Existing community

Trivial to re-emit to native

CONS

Difficult to single-shot lift from binary

Each architecture must implement SSA, stack tracking, other generic solutions

Not designed for translation from binaries

Landscape: Microcode

```
004014FB    mov     eax, [ebx+4]
004014FE    mov     dl, [eax+1]
00401501    sub     dl, 61h ; 'a'
00401504    jz      short loc_401517
```

```
2. 0 mov    ebx.4, eoff.4           ; 4014FB u=ebx.4      d=eoff.4
2. 1 mov    ds.2, seg.2             ; 4014FB u=ds.2       d=seg.2
2. 2 add    eoff.4, #4.4, eoff.4    ; 4014FB u=eoff.4     d=eoff.4
2. 3 ldx    seg.2, eoff.4, et1.4    ; 4014FB u=eoff.4,seg.2,
                                              ; (STACK,GLBMEM)
d=et1.4
2. 4 mov    et1.4, eax.4            ; 4014FB u=et1.4      d=eax.4
2. 5 mov    eax.4, eoff.4           ; 4014FE u=eax.4      d=eoff.4
2. 6 mov    ds.2, seg.2             ; 4014FE u=ds.2       d=seg.2
2. 7 add    eoff.4, #1.4, eoff.4    ; 4014FE u=eoff.4     d=eoff.4
2. 8 ldx    seg.2, eoff.4, t1.1     ; 4014FE u=eoff.4,seg.2,
                                              ; (STACK,GLBMEM)
d=t1.1
2. 9 mov    t1.1, dl.1              ; 4014FE u=t1.1       d=dl.1
2.10 mov    #0x61.1, t1.1           ; 401501 u=           d=t1.1
2.11 setb   dl.1, t1.1, cf.1        ; 401501 u=dl.1,t1.1  d=cf.1
2.12 seto   dl.1, t1.1, of.1        ; 401501 u=dl.1,t1.1  d=of.1
2.13 sub    dl.1, t1.1, dl.1        ; 401501 u=dl.1,t1.1  d=dl.1
2.14 setz   dl.1, #0.1, zf.1        ; 401501 u=dl.1       d=zf.1
2.15 setp   dl.1, #0.1, pf.1        ; 401501 u=dl.1       d=pf.1
2.16 sets   dl.1, sf.1              ; 401501 u=dl.1       d=sf.1
2.17 mov    cs.2, seg.2             ; 401504 u=cs.2       d=seg.2
2.18 mov    #0x401517.4, eoff.4     ; 401504 u=           d=eoff.4
2.19 jcnd   zf.1, $loc_401517       ; 401504 u=zf.1
```

IDA PRO

Lifting is verbose

Later optimizations

Not designed for reading

Landscape: Microcode

```
004014FB    mov     eax, [ebx+4]
004014FE    mov     dl, [eax+1]
00401501    sub     dl, 61h ; ‘a’
00401504    jz      short loc_401517
```

```
2. 0 ldx    ds.2, (ebx.4+#4.4), eax.4 ; 4014FB u=ebx.4,ds.2,
                                      ;(STACK,GLBMEM) d=eax.4
2. 1 ldx    ds.2, (eax.4+#1.4), dl.1 ; 4014FE u=eax.4,ds.2,
                                      ;(STACK,GLBMEM) d=dl.1
2. 2 setb   dl.1, #0x61.1, cf.1     ; 401501 u=dl.1       d=cf.1
2. 3 seto   dl.1, #0x61.1, of.1     ; 401501 u=dl.1       d=of.1
2. 4 sub    dl.1, #0x61.1, dl.1     ; 401501 u=dl.1       d=dl.1
2. 5 setz   dl.1, #0.1, zf.1        ; 401501 u=dl.1       d=zf.1
2. 6 setp   dl.1, #0.1, pf.1        ; 401501 u=dl.1       d=pf.1
2. 7 sets   dl.1, sf.1              ; 401501 u=dl.1       d=sf.1
2. 8 jcnd   zf.1, $loc_401517       ; 401504 u=zf.1
```

IDA PRO

Lifting is verbose

Later optimizations

Not designed for reading

Landscape: Microcode

```
004014FB    mov     eax, [ebx+4]
004014FE    mov     dl, [eax+1]
00401501    sub     dl, 61h ; ‘a’
00401504    jz      short loc_401517
```

```
2. 1 ldx    ds.2{3}, ([ds.2{3}:(ebx.4+#4.4)].4+#1.4), dl.1{5} ; 4014FE
                        ; u=ebx.4,ds.2,(GLBLOW,sp+20..,GLBHIGH) d=dl.1
2. 2 sub    dl.1{5}, #0x61.1, dl.1{6} ; 401501 u=dl.1       d=dl.1
2. 3 jz     dl.1{6}, #0.1, @7       ; 401504 u=dl.1
```

IDA PRO

Lifting is verbose

Later optimizations

Not designed for reading

Landscape: Microcode

```
004014FB    mov     eax, [ebx+4]
004014FE    mov     dl, [eax+1]
00401501    sub     dl, 61h ; ‘a’
00401504    jz      short loc_401517
```

```
2. 0 jz     [ds.2{4}:([ds.2{4}:(ebx.4{8}+#4.4){7}].4{6}+#1.4){5}].1{3},
            #0x61.1,
            @7
                        ; 401504 u=ebx.4,ds.2,(GLBLOW,GLBHIGH)
```

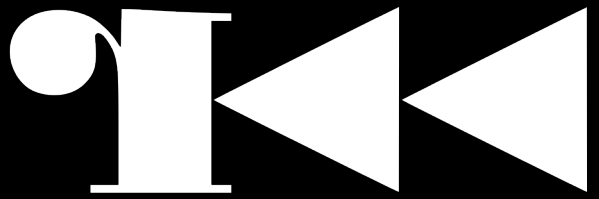
IDA PRO

Lifting is verbose

Later optimizations

Not designed for reading

Landscape: ESIL



- Radare
- String based
- Post-fix notation
- Concise

```
|  ; ebp=0xfffffffc -> 0xffffff00
|          0x004033d4      81ec2c020000    556,esp,-=,$o,of,=,$s,sf,=,$z,zf,=,$p,pf,=,$b4,cf,= ;
|          0x004033da      53              ebx,4,esp,-=,esp,=[4]       ; esp=0xffffdcc -> 0xffffff00
|          0x004033db      56              esi,4,esp,-=,esp,=[4]       ; esp=0xffffdc8 -> 0xffffff00
|          0x004033dc      57              edi,4,esp,-=,esp,=[4]       ; esp=0xffffdc4 -> 0xffffff00
|          0x004033dd      68dd344000      4207837,4,esp,-=,esp,=[4]   ; esp=0xffffdc0 -> 0xffffff00
|          0x004033e2      58              esp,[4],eax,=,4,esp,+=      ; eax=0xffffffff -> 0xffffff00  ; esp=0xffffdc4 -> 0xffffff00
|          0x004033e3      8945e0          eax,0x20,ebp,-,=[4]
|          0x004033e6      68fd414000      4211197,4,esp,-=,esp,=[4]   ; esp=0xffffdc0 -> 0xffffff00
```

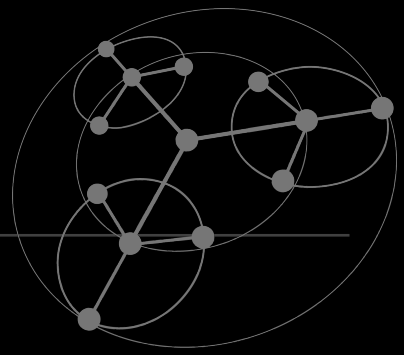
Landscape: P-Code

- Ghidra
- Sleigh definitions

- More human readable
- Many architectures

```
SUB         RSP,0x618
                        (register, 0x200, 1) = INT_LESS (register, 0x20, 8), (const, 0x618, 8)
                        (register, 0x20b, 1) = INT_SBORROW (register, 0x20, 8), (const, 0x618, 8)
                        (register, 0x20, 8) = INT_SUB (register, 0x20, 8), (const, 0x618, 8)
                        (register, 0x207, 1) = INT_SLESS (register, 0x20, 8), (const, 0x0, 8)
                        (register, 0x206, 1) = INT_EQUAL (register, 0x20, 8), (const, 0x0, 8)
MOV         R15,RSI
                        (register, 0xb8, 8) = COPY (register, 0x30, 8)
MOV         R14D,EDI
                        (register, 0xb0, 4) = COPY (register, 0x38, 4)
                        (register, 0xb0, 8) = INT_ZEXT (register, 0xb0, 4)
```

Landscape: REIL



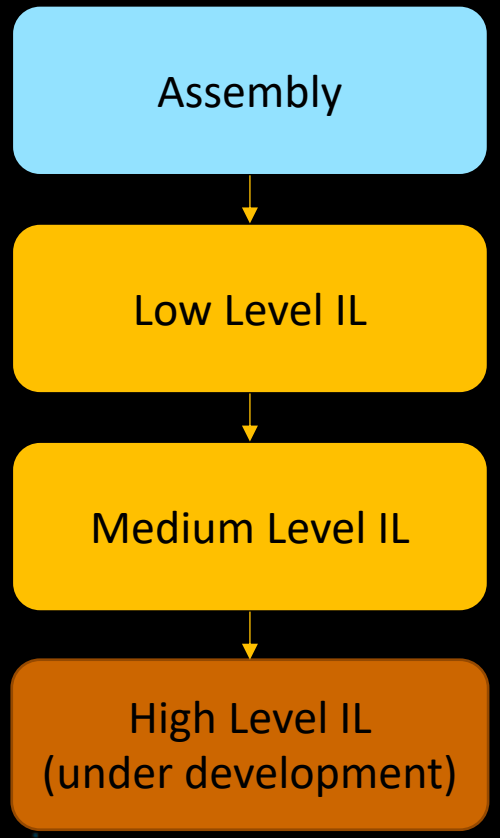
```
00000000.00 STR R_EAX:32, , V_00:32
00000000.01 STR 0:1, , R_CF:1
00000000.02 AND V_00:32, ff:8, V_01:8
00000000.03 SHR V_01:8, 7:8, V_02:8
00000000.04 SHR V_01:8, 6:8, V_03:8
00000000.05 XOR V_02:8, V_03:8, V_04:8
00000000.06 SHR V_01:8, 5:8, V_05:8
00000000.07 SHR V_01:8, 4:8, V_06:8
00000000.08 XOR V_05:8, V_06:8, V_07:8
00000000.09 XOR V_04:8, V_07:8, V_08:8
00000000.0a SHR V_01:8, 3:8, V_09:8
00000000.0b SHR V_01:8, 2:8, V_10:8
00000000.0c XOR V_09:8, V_10:8, V_11:8
00000000.0d SHR V_01:8, 1:8, V_12:8
00000000.0e XOR V_12:8, V_01:8, V_13:8
00000000.0f XOR V_11:8, V_13:8, V_14:8
00000000.10 XOR V_08:8, V_14:8, V_15:8
00000000.11 AND V_15:8, 1:1, V_16:1
00000000.12 NOT V_16:1, , R_PF:1
00000000.13 STR 0:1, , R_AF:1
00000000.14 EQ V_00:32, 0:32, R_ZF:1
00000000.15 SHR V_00:32, 1f:32, V_17:32
00000000.16 AND 1:32, V_17:32, V_18:32
00000000.17 EQ 1:32, V_18:32, R_SF:1
00000000.18 STR 0:1, , R_OF:1
```

- BinDiff/BinNavi
- 17 instructions
- Extremely verbose

Landscape: BNIL

- Binary Ninja
- Tiered family of ILs
- Tree-based
- Deferred flags

Landscape: BNIL

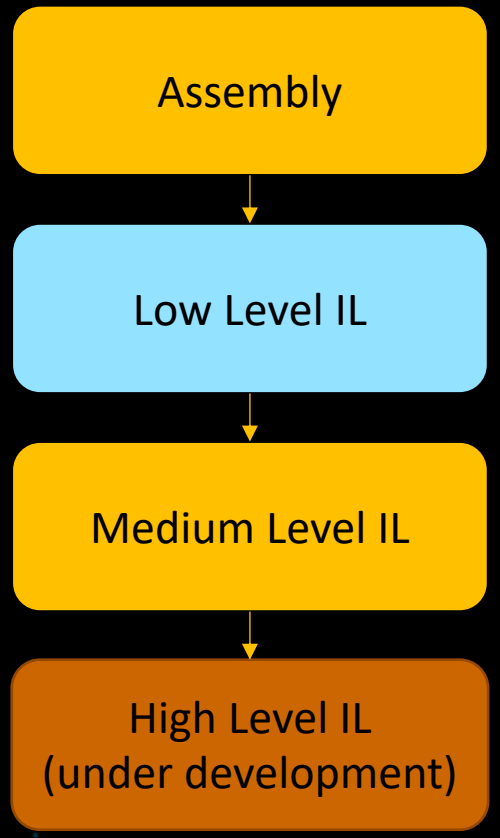



```
main:
00400746  55                push    rbp {__saved_rbp}
00400747  4889e5            mov     rbp, rsp {__saved_rbp}
0040074a  4883c480          add     rsp, 0xffffffffffffff80
0040074e  64488b0425280000… mov     rax, qword [fs:0x28]
00400757  488945f8          mov     qword [rbp-0x8 {var_10}], rax
0040075b  31c0              xor     eax, eax  {0x0}
0040075d  c7458000000000    mov     dword [rbp-0x80 {var_88}], 0x0
00400764  c7458400000000    mov     dword [rbp-0x7c {var_84}], 0x0
0040076b  c7458c41414141    mov     dword [rbp-0x74 {var_7c}], 0x41414141
00400772  81458c00332221    add     dword [rbp-0x74 {var_7c}], 0x21223300  {0x62637441}
00400779  eb44              jmp     0x4007bf
```

```
004007bf  837d8400          cmp     dword [rbp-0x7c {var_84}], 0x0
004007c3  74b6              je      0x40077b
```

```
004007c5  c17d8c02          sar     dword [rbp-0x74 {var_7c}], 0x2  {0x1898dd10}
004007c9  8b4580            mov     eax, dword [rbp-0x80 {var_88}]
004007cc  89c6              mov     esi, eax
004007ce  bf08094000        mov     edi, 0x400908  {"\nYou tried: %d\nLet's see if th…"}
004007d3  b800000000        mov     eax, 0x0
004007d8  e803feffff        call    printf
004007dd  c7458800000000    mov     dword [rbp-0x78 {var_80}], 0x0
004007e4  eb27              jmp     0x40080d
```

Landscape: BNIL

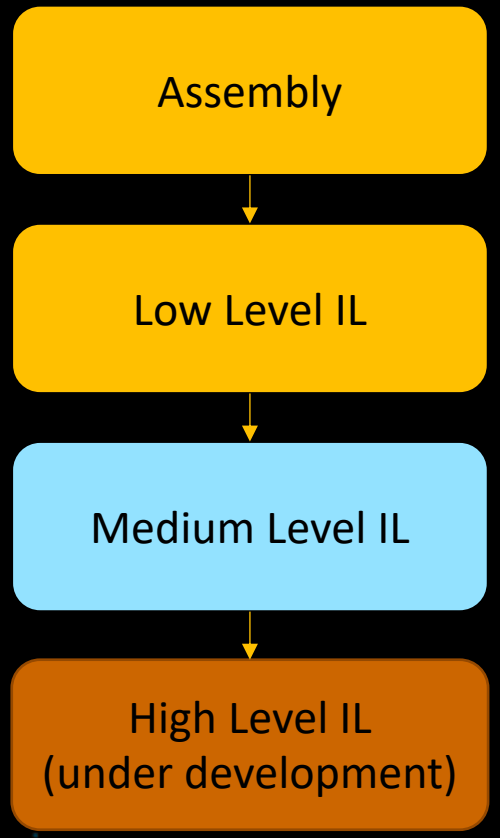



```
main:
   0 @ 00400746  push(rbp)
   1 @ 00400747  rbp = rsp {__saved_rbp}
   2 @ 0040074a  rsp = rsp - 0x80
   3 @ 0040074e  rax = [fs + 0x28].q
   4 @ 00400757  [rbp - 8 {var_10}].q = rax
   5 @ 0040075b  eax = 0
   6 @ 0040075d  [rbp - 0x80 {var_88}].d = 0
   7 @ 00400764  [rbp - 0x7c {var_84}].d = 0
   8 @ 0040076b  [rbp - 0x74 {var_7c}].d = 0x41414141
   9 @ 00400772  [rbp - 0x74 {var_7c}].d = [rbp - 0x74 {var_7c}].d + 0x21223300
  10 @ 00400779  goto 11 @ 0x4007c3
```

```
11 @ 004007c3  if ([rbp - 0x7c {var_84}].d == 0) then 12 @ 0x40077b else 28 @ 0x4007c5
```

```
28 @ 004007c5  [rbp - 0x74 {var_7c}].d = [rbp - 0x74 {var_7c}].d s>> 2
29 @ 004007c9  eax = [rbp - 0x80 {var_88}].d
30 @ 004007cc  esi = eax
31 @ 004007ce  edi = 0x400908  {"\nYou tried: %d\nLet's see if th…"}
32 @ 004007d3  eax = 0
33 @ 004007d8  call(printf)
34 @ 004007dd  [rbp - 0x78 {var_80}].d = 0
35 @ 004007e4  goto 36 @ 0x400811
```

Landscape: BNIL

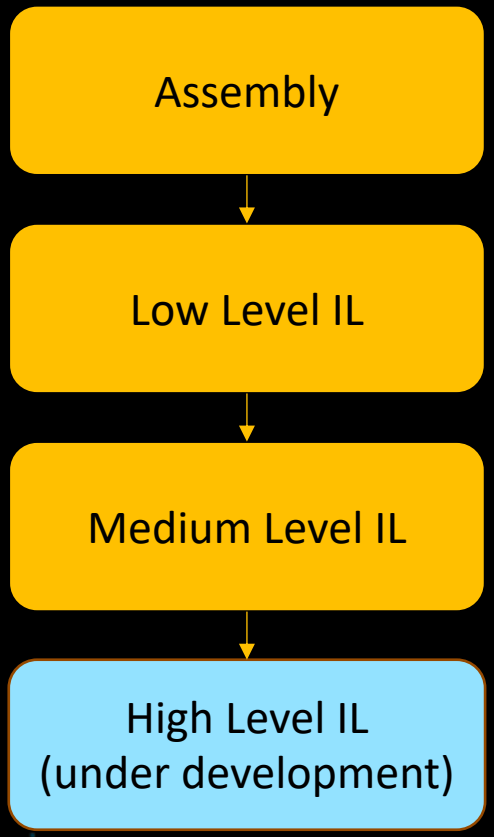



```
main:
   0 @ 0040074e  int64_t rax = [fs + 0x28].q
   1 @ 00400757  int64_t var_10 = rax
   2 @ 0040075b  int64_t rax_1 = 0
   3 @ 0040075d  int32_t var_88 = 0
   4 @ 00400764  int32_t var_84 = 0
   5 @ 00400779  goto 6 @ 0x4007c3

   6 @ 004007c3  if (var_84 == 0) then 7 @ 0x400780 else 18 @ 0x4007c9

  18 @ 004007c9  uint64_t rax_5 = zx.q(var_88)
  19 @ 004007cc  uint64_t rsi = zx.q(rax_5.eax)
  20 @ 004007d3  int64_t rax_6 = 0
  21 @ 004007d8  printf(0x400908, rsi)  {"\nYou tried: %d\nLet's see if th…"}
  22 @ 004007dd  int32_t var_80 = 0
  23 @ 004007e4  goto 24 @ 0x400811
```

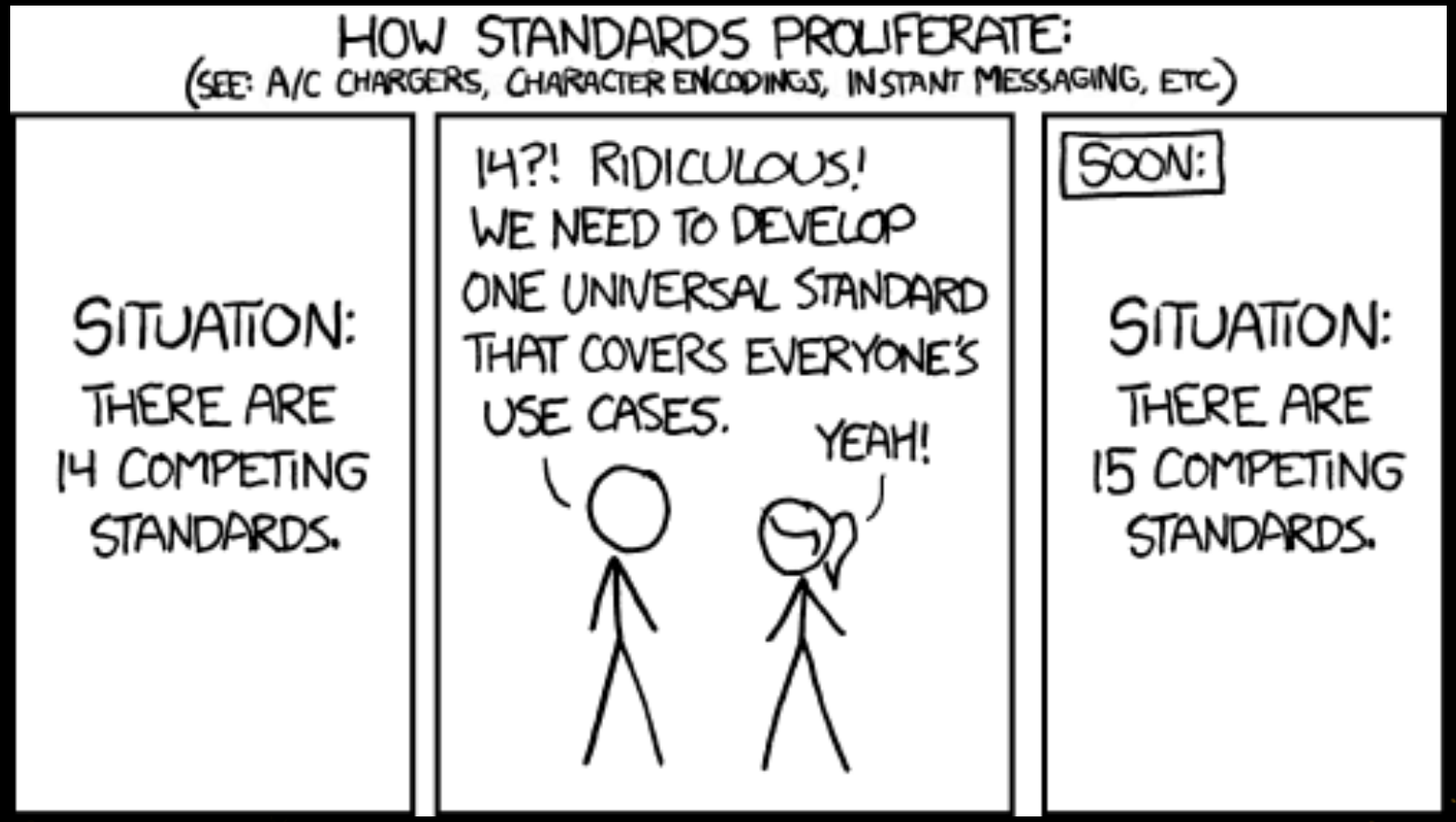
Landscape: BNIL




```
int32_t main(int32_t argc, char** argv, char** envp)

rax = *(fs + 0x28)
var_88 = 0
for (var_84 = 0; var_84 == 0; var_84 = sscanf(&var_78, 0x400905, &var_88):0.d)
    printf(0x4008f8)  {"Password 2: "}
    fgets(&var_78, 0x64, stdin)
rax_5 = zx.q(var_88)
printf(0x400908, zx.q(rax_5:0.d))  {"\nYou tried: %d\nLet's see if th…"}
for (var_80 = 0; var_80 s<= 5; var_80 = var_80 + 1)
    putchar(0x2e)
    fflush(stdout)
    sleep(1)
putchar(0xa)
rax_7 = zx.q(var_88)
if (rax_7:0.d != 0x1898d542)
    printf(0x40094f)  {"I'm sorry, you have failed."}
else
    printf(0x400935)  {"Great job! You succeeded."}
if ((rax ^ *(fs + 0x28)) == 0)
    return 0
__stack_chk_fail()
noreturn
```

Why so many?


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Why so many?

Good reasons:

- Requirements
  - IL Abstractions
  - IL API Language support
  - Source Architecture
  - Source Language
- Landscape full of unmaintained ILs
- Licensing

Why so many?

Bad reasons:

- Not-Invented-Here
- Lack of awareness
- Publish or Perish

Questions to ask your IL before committing

1. What architectures are supported?
2. What languages are supported?
3. How complete is the lifting?
4. How are stack variables handled?
5. How are functions discovered?
6. How are function parameters determined?

Questions to ask your IL before committing

7. Are types recovered?
8. What APIs exist for manipulating the IL?
9. What dataflow APIs exist?
10. How good is the documentation/examples?
11. How verbose is the IL?
12. What support options exist?

DEMOS

~~Questions?~~

NOT NOW, FIND US IN THE SPEAKER SPOT!

Addendum: Bonus Slides

Additional Resources

https://blog.quarkslab.com/an-experimental-study-of-different-binary-exporters.html

https://adalogics.com/blog/binary-to-llvm-comparison

Allin Poster: http://sdasgup3.web.engr.illinois.edu/Document/allin_poster.pdf

Working with ILs

GENERAL TECHNIQUES AND TIPS

Tree-Based

Simplifies lifting

Concise representations

Analysis code requires visitor or recursive search

Parallels native forms (`mov eax, [ecx + eax*4]`)

Tree-Based

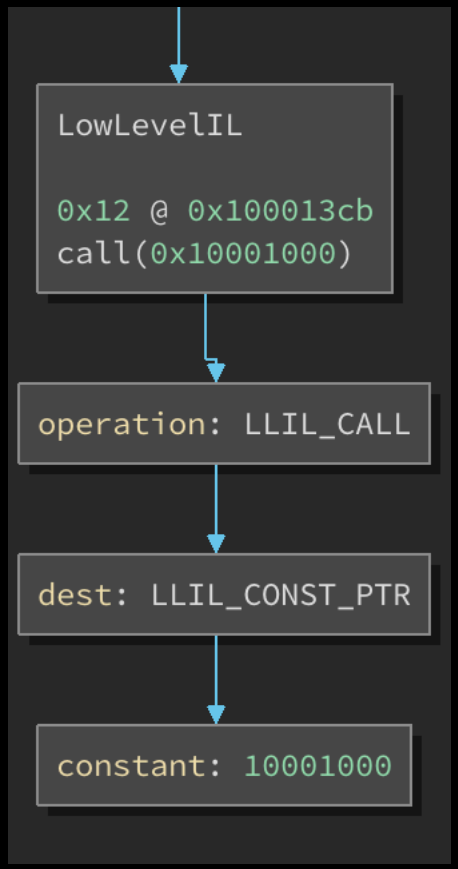
Simplifies lifting

Concise representations

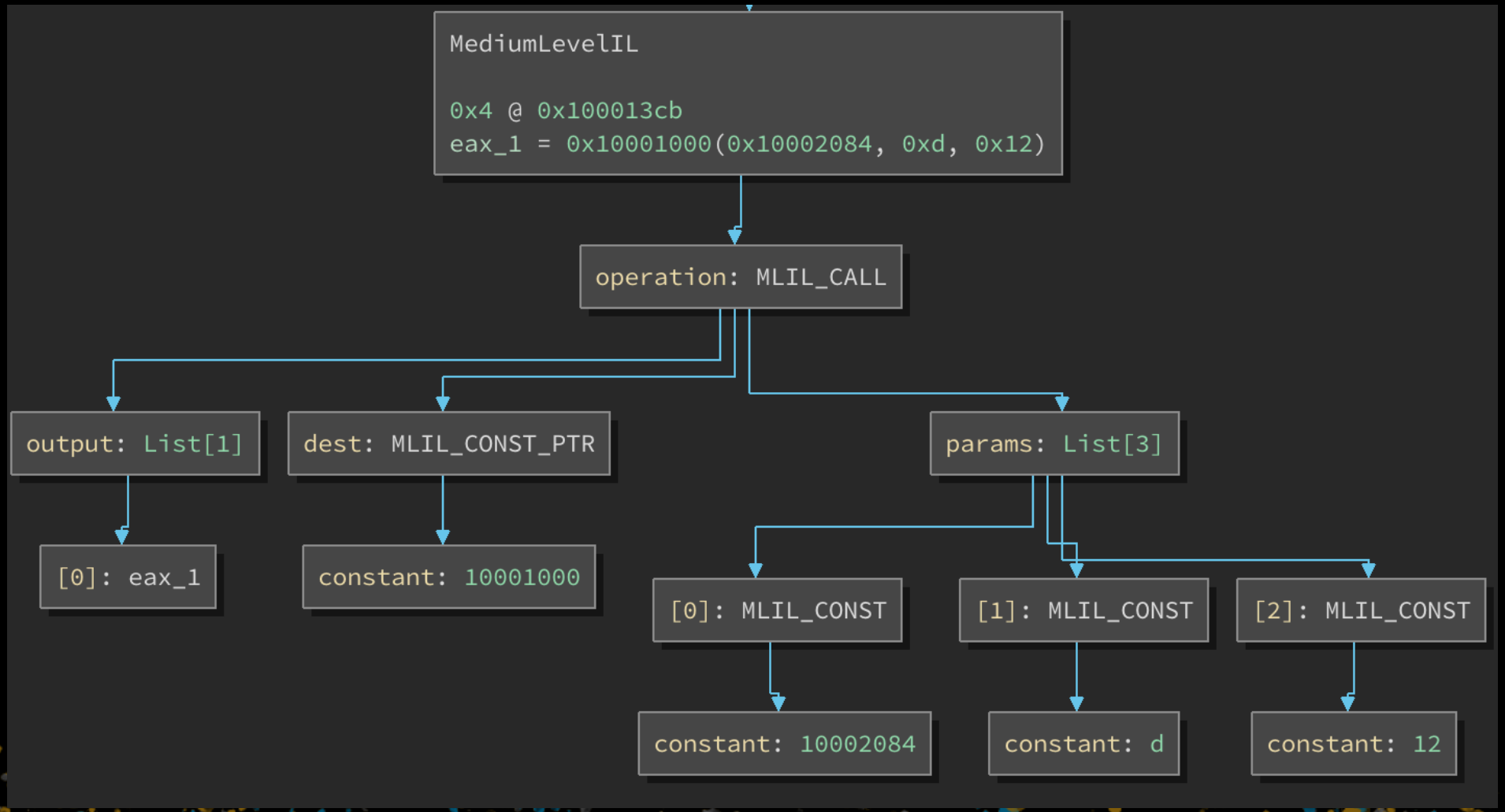
Analysis code requires visitor or recursive search

Parallels native forms (`mov eax, [ecx + eax*4]`)

Tree-Based


LowLevelIL
0x12 @ 0x100013cb
call(0x10001000)
operation: LLIL_CALL
dest: LLIL_CONST_PTR
constant: 10001000

Tree-Based


MediumLevelIL
0x4 @ 0x100013cb
eax_1 = 0x10001000(0x10002084, 0xd, 0x12)
operation: MLIL_CALL
output: List[1]
dest: MLIL_CONST_PTR
params: List[3]
[0]: eax_1
constant: 10001000
[0]: MLIL_CONST
[1]: MLIL_CONST
[2]: MLIL_CONST
constant: 10002084
constant: d
constant: 12

Three-Address Code

One operation, three arguments (sometimes two in, one out)

Used internally in optimizing compilers

Lots of temporaries

Simplifies some analysis

```
xor(var1, var1, var1)
```

SSA Forms

Single-Static-Assignment

All variables read-only

ϕ used to merge paths

Quickly backtrack expressions that make up a variable