(Artifact Reusable)

Probabilistic Disassembly

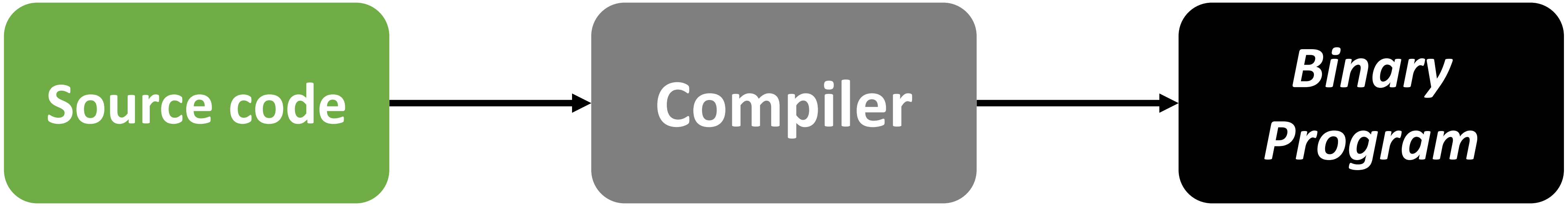
Kenneth Adam Miller*, Yonghwi Kwon†, Yi Sun*, Zhuo Zhang*, Xiangyu Zhang*, Zhiqiang Lin‡

* Department of Computer Science, ***Purdue University***, *West Lafayette, USA*

†Department of Computer Science, ***University of Virginia***, *Charlottesville, USA*

‡Department of Computer Science and Engineering, ***Ohio State University***, *Columbus, USA*

What is Disassembler?



Source code

Compiler

Binary Program

Various *semantics lost*:

(1) Variable names

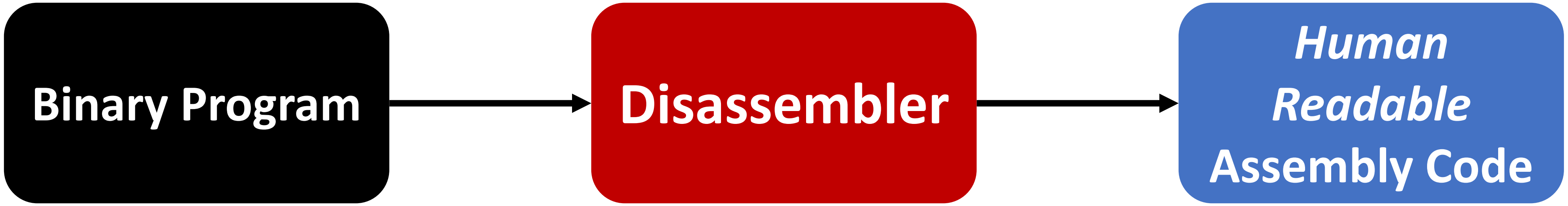
(2) Data structures (e.g., variable boundaries)

(3) Function names (i.e., indirect call targets)

(4) …

irreversible transformation

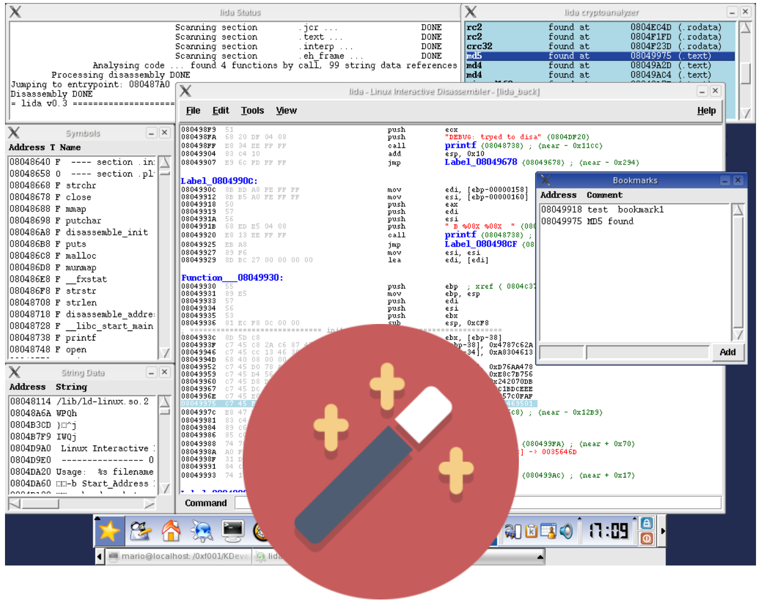
What is Disassembler?



Binary Program

Disassembler

Human Readable Assembly Code



```
                    Scanning section      .jcr ...              DONE
                    Scanning section      .text ...             DONE
                    Scanning section      .interp ...           DONE
                    Scanning section      .eh_frame ...         DONE
          Analysing code ... found 4 functions by call, 99 string data references
      Processing disassembly DONE
Jumping to entrypoint: 080487A0
Disassembly DONE
= lida v0.3 ====================
```

```
rc2      found at   0804EC4D (.rodata)
rc2      found at   0804F1FD (.rodata)
crc32    found at   0804F23D (.rodata)
md5      found at   08049975 (.text)
md4      found at   08049A2D (.text)
md4      found at   08049AC4 (.text)
```

```
Address T Name
08048640 F  ---- section .in
08048658 O  ---- section .pl
08048668 F strchr
08048678 F close
08048688 F mmap
08048698 F putchar
080486A8 F disassemble_init
080486B8 F puts
080486C8 F malloc
080486D8 F munmap
080486E8 F __fxstat
080486F8 F strstr
08048708 F strlen
08048718 F disassemble_addre
08048728 F __libc_start_main
08048738 F printf
08048748 F open
```

```
Address  String
08048114 /lib/ld-linux.so.2
08048A6A WPQh
0804B3CD }□^j
0804B7F9 IWQj
0804D9A0  Linux Interactive
0804D9E0  ---------------- 0
0804DA20 Usage:  %s filename
0804DA60 □□-b Start_Address
```

```
080498F9  51                    push   ecx
080498FA  68 20 DF 04 08        push   "DEBUG: tryed to disa" (0804DF20)
080498FF  E8 34 EE FF FF        call   printf (08048738) ; (near - 0x11cc)
08049904  83 C4 10              add    esp, 0x10
08049907  E9 6C FD FF FF        jmp    Label_08049678 (08049678) ; (near - 0x294)

Label_0804990C:
0804990c  8B BD A8 FE FF FF     mov    edi, [ebp-00000158]
08049912  8B B5 A0 FE FF FF     mov    esi, [ebp-00000160]
08049918  50                    push   eax
08049919  57                    push   edi
0804991A  56                    push   esi
0804991B  68 ED E5 04 08        push   " B %08X %08X  " (08
08049920  E8 13 EE FF FF        call   printf (08048738) ;
08049925  EB A8                 jmp    Label_080498CF (08
08049927  89 F6                 mov    esi, esi
08049929  8D BC 27 00 00 00 00  lea    edi, [edi]

Function___08049930:
08049930  55                    push   ebp  ; xref ( 0804c37
08049931  89 E5                 mov    ebp, esp
08049933  57                    push   edi
08049934  56                    push   esi
08049935  53                    push   ebx
08049936  81 EC F8 0C 00 00     sub    esp, 0xCF8
```

```
Address  Comment
08049918 test  bookmark1
08049975 MD5 found
```

```
  .globl  "_*add_forty_two<Int32>:Int32"
  .align  4, 0x90
"_*add_forty_two<Int32>:Int32":
  .cfi_startproc
  pushq %rbp
Ltmp1992:
  .cfi_def_cfa_offset 16
Ltmp1993:
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
Ltmp1994:
  .cfi_def_cfa_register %rbp
  addl  $42, %edi
  movl  %edi, %eax
  popq  %rbp
  retq
  .cfi_endproc
```

Recover lost semantics with ...

Uses of Disassembly

- Reverse-Engineering (e.g., of legacy applications)
- Malware Analysis
- Binary Rewriting for
  - Optimization
  - Security – Software Hardening, Hot Patches, De-bloating (Attack Surface reduction)
  - Instrumentation and Debugging
- …

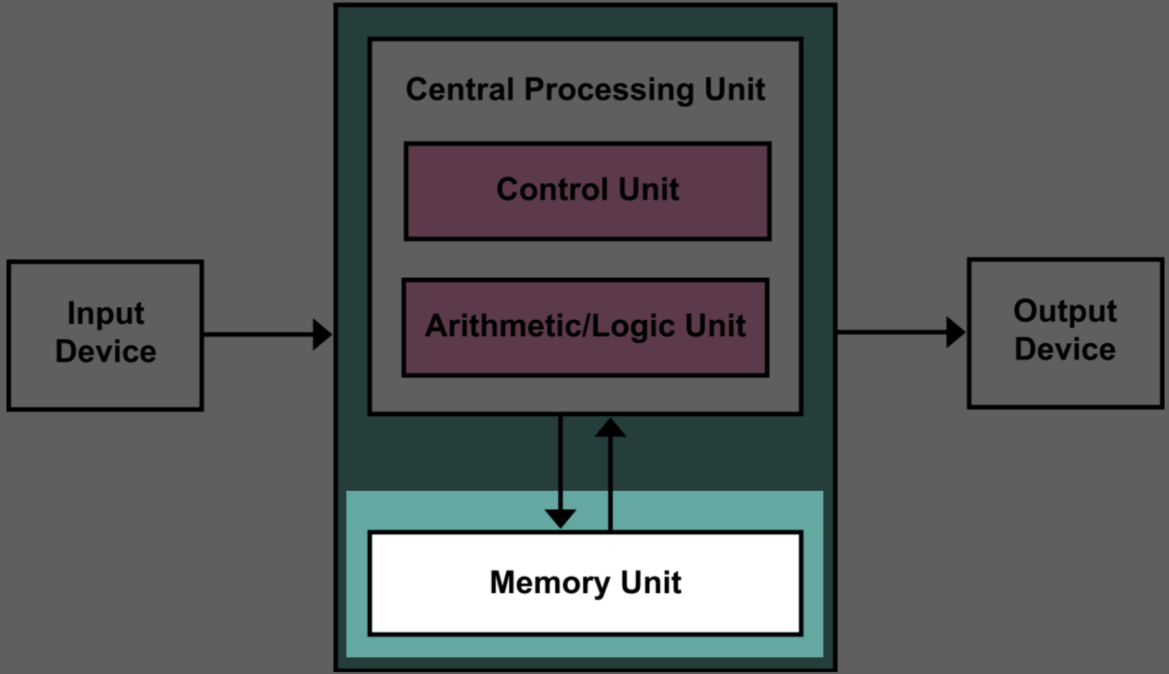
Fundamental Problem in Disassembly

When we recover the lost semantics,
***uncertainty** is inevitable.*

Especially problematic in recovering
indirect control flow by
function pointers, virtual tables, switch-case, etc.,
*and making it difficult to know **where the code begins***

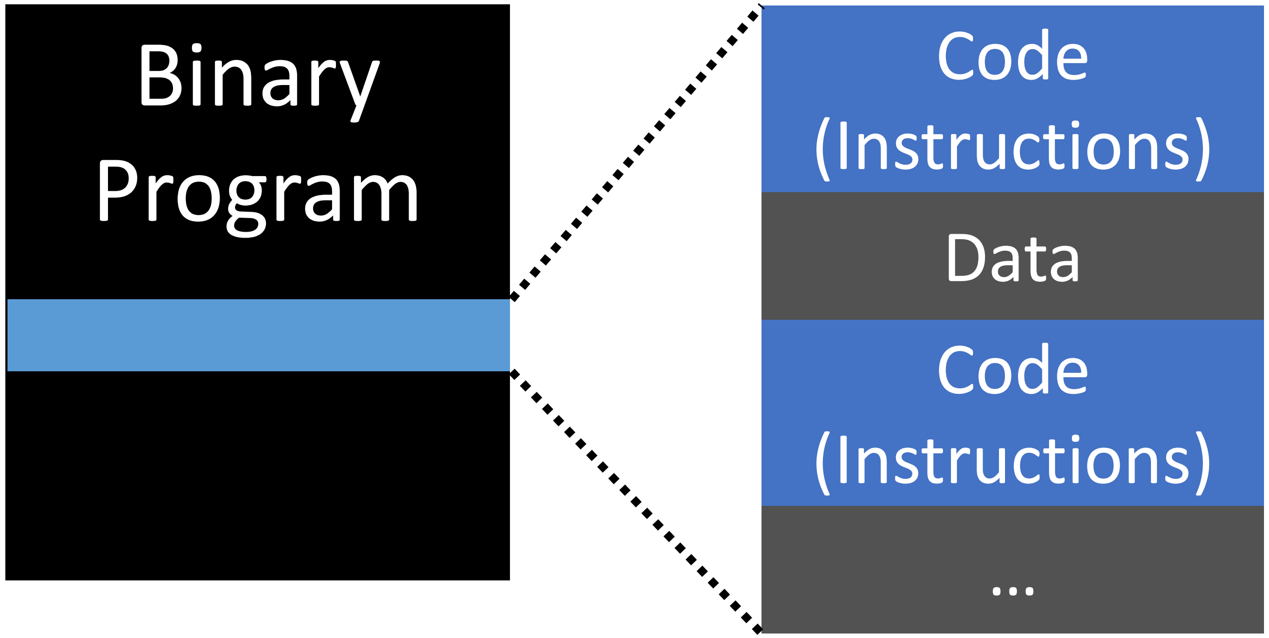
Fundamental Problem in Disassembly

In Von-Neumann Architecture,
code and data coexist within the same memory space,
and they could be interleaving



Fundamental Problem in Disassembly

In Von-Neumann Architecture,
code and data coexist within the same memory space,
and they could be interleaving



Data and Code Coexistence: Linear Sweep

ADDRESS	RAW BYTE	ASSEMBLY (Ground-truth)
0xBBF72	66 66 66 66	DB 66 66 66 66
0xBBF76	66 2E 0F 1F	DB 66 2E 0F 1F
0xBBF7A	84 00 00 00	DB 84 00 00 00
0xBBF7E	00 00 90 0F	DB 00 00 90 0F
0xBBF82	1F 84 00 00	DB 1F 84 00 00
0xBBF84	00 00 00 00	DB 00 00 00 00
0xBBF8A	00 00 00 00	DB 00 00 00 00
0xBBF8E	00 00	DB 00 00
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF97	41 56	push r14
...	...	...
0xBBFA2	48 8D 50 10	lea 16(rax), rdx
0xBBFA6	48 05 90 00 00 00	add 144, rax
0xBBFAC	48 89 47 10	mov rax, 16(rdi)
0xBBFB0	48 89 17	mov rdx, (rdi)

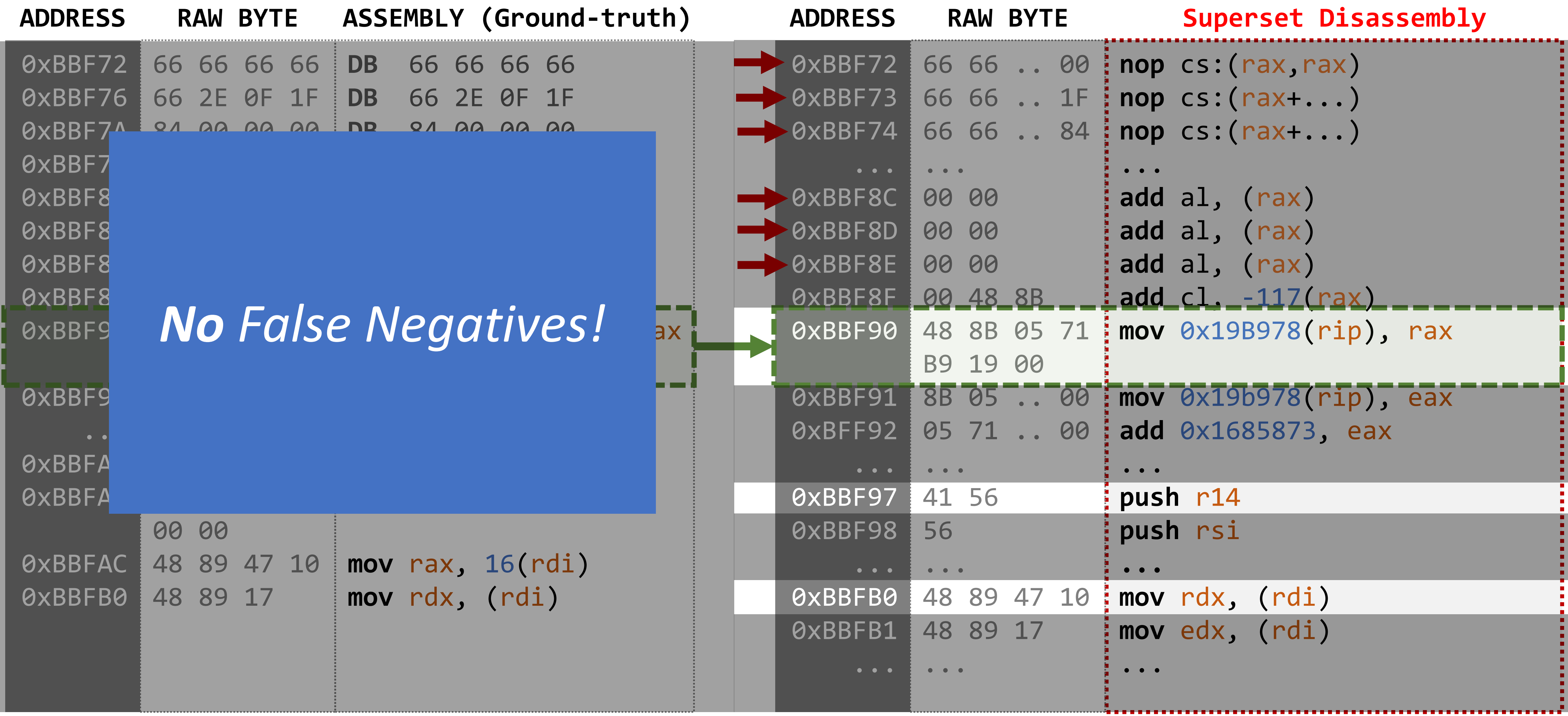
ADDRESS	RAW BYTE	Linear Sweep
0xBBF72	66 66 66 66	nop cs:(rax,rax)
0xBBF76	66 2E 0F 1F	...
0xBBF7A	84 00 00 00	...
0xBBF7C	00 00 90 0F	...
0xBBF82	...	...
0xBBF8A	00	...
0xBBF8B	00 00	add al, (rax)
0xBBF8D	00 00	add al, (rax)
0xBBF8F	00 48 8B	add cl, -117(rax)
0xBBF92	05 71 B9 19 00	add 1685873, eax
0xBBF97	41 56	push r14
...	...	...
0xBBFA2	48 8D 50 10	lea 16(rax), rdx
0xBBFA6	48 05 90 00 00 00	add 144, rax
0xBBFAC	48 89 47 10	mov rax, 16(rdi)
0xBBFB0	48 89 17	mov rdx, (rdi)

Data and Code Coexistence: Superset Disassembly

ADDRESS	RAW BYTE	ASSEMBLY (Ground-truth)
0xBBF72	66 66 66 66	DB 66 66 66 66
0xBBF76	66 2E 0F 1F	DB 66 2E 0F 1F
0xBBF7A	84 00 00 00	DB 84 00 00 00
0xBBF7E	00 00 90 0F	DB 00 00 90 0F
0xBBF82	1F 84 00 00	DB 1F 84 00 00
0xBBF84	00 00 00 00	DB 00 00 00 00
0xBBF8A	00 00 00 00	DB 00 00 00 00
0xBBF8E	00 00	DB 00 00
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF97	41 56	push r14
...	...	...
0xBBFA2	48 8D 50 10	lea 16(rax), rdx
0xBBFA6	48 05 90 00 00 00	add 144, rax
0xBBFAC	48 89 47 10	mov rax, 16(rdi)
0xBBFB0	48 89 17	mov rdx, (rdi)

ADDRESS	RAW BYTE	Superset Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)
0xBBF73	66 66 .. 1F	nop cs:(rax+...)
0xBBF74	66 66 .. 84	nop cs:(rax+...)
...	...	...
0xBBF8C	00 00	add al, (rax)
0xBBF8D	00 00	add al, (rax)
0xBBF8E	00 00	add al, (rax)
0xBBF8F	00 48 8B	add cl, -117(rax)
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax
0xBFF92	05 71 .. 00	add 0x1685873, eax
...	...	...
0xBBF97	41 56	push r14
0xBBF98	56	push rsi
...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)
0xBBFB1	48 89 17	mov edx, (rdi)
...	...	...

Data and Code Coexistence: Superset Disassembly

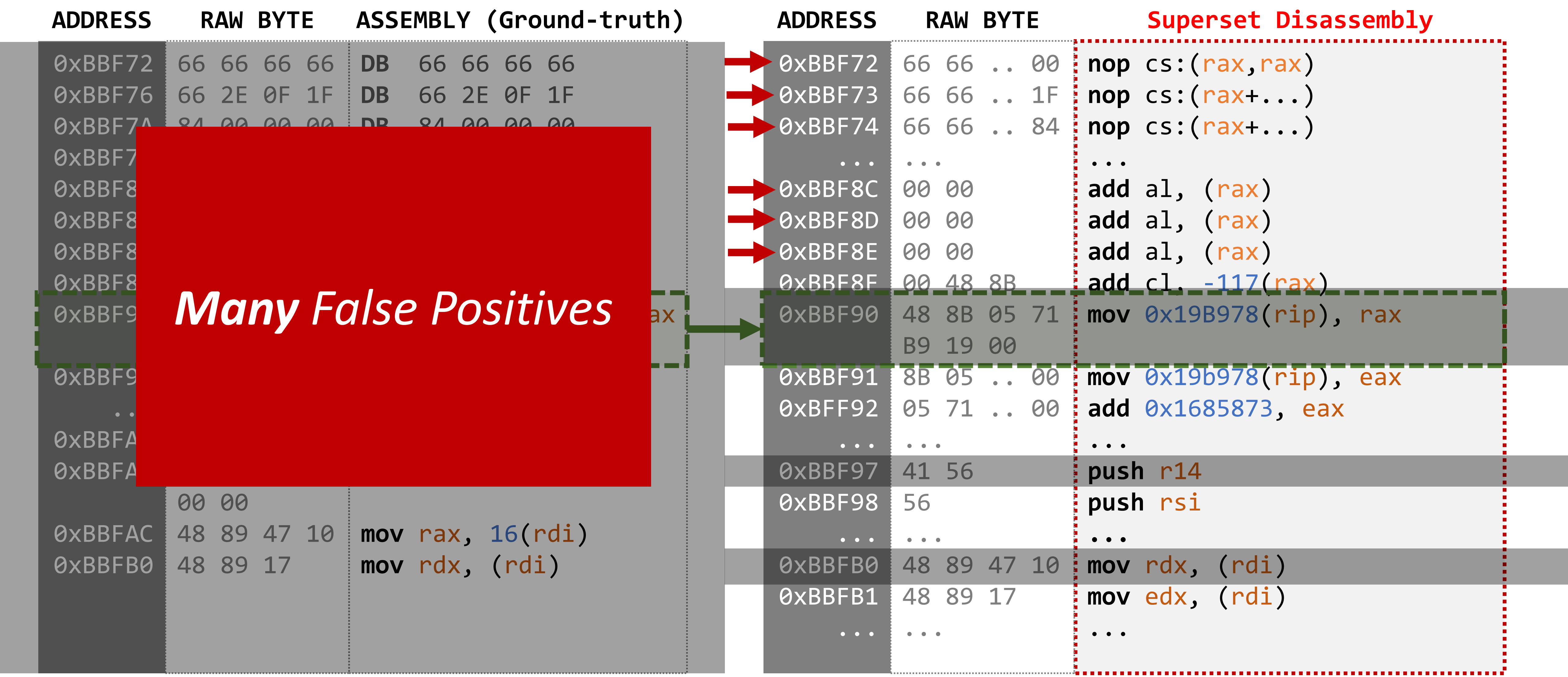


```
ADDRESS    RAW BYTE       ASSEMBLY (Ground-truth)
0xBBF72    66 66 66 66    DB  66 66 66 66
0xBBF76    66 2E 0F 1F    DB  66 2E 0F 1F
0xBBF7A    84 00 00 00    DB  84 00 00 00
0xBBF7
0xBBF8
0xBBF8
0xBBF8
0xBBF8
0xBBF9                                         ax
0xBBF9
      ..
0xBBFA
0xBBFA
           00 00
0xBBFAC    48 89 47 10    mov rax, 16(rdi)
0xBBFB0    48 89 17       mov rdx, (rdi)
```

***No** False Negatives!*

```
ADDRESS    RAW BYTE       Superset Disassembly
0xBBF72    66 66 .. 00    nop cs:(rax,rax)
0xBBF73    66 66 .. 1F    nop cs:(rax+...)
0xBBF74    66 66 .. 84    nop cs:(rax+...)
    ...    ...            ...
0xBBF8C    00 00          add al, (rax)
0xBBF8D    00 00          add al, (rax)
0xBBF8E    00 00          add al, (rax)
0xBBF8F    00 48 8B       add cl, -117(rax)
0xBBF90    48 8B 05 71    mov 0x19B978(rip), rax
           B9 19 00
0xBBF91    8B 05 .. 00    mov 0x19b978(rip), eax
0xBFF92    05 71 .. 00    add 0x1685873, eax
    ...    ...            ...
0xBBF97    41 56          push r14
0xBBF98    56             push rsi
    ...    ...            ...
0xBBFB0    48 89 47 10    mov rdx, (rdi)
0xBBFB1    48 89 17       mov edx, (rdi)
    ...    ...            ...
```

Data and Code Coexistence: Superset Disassembly

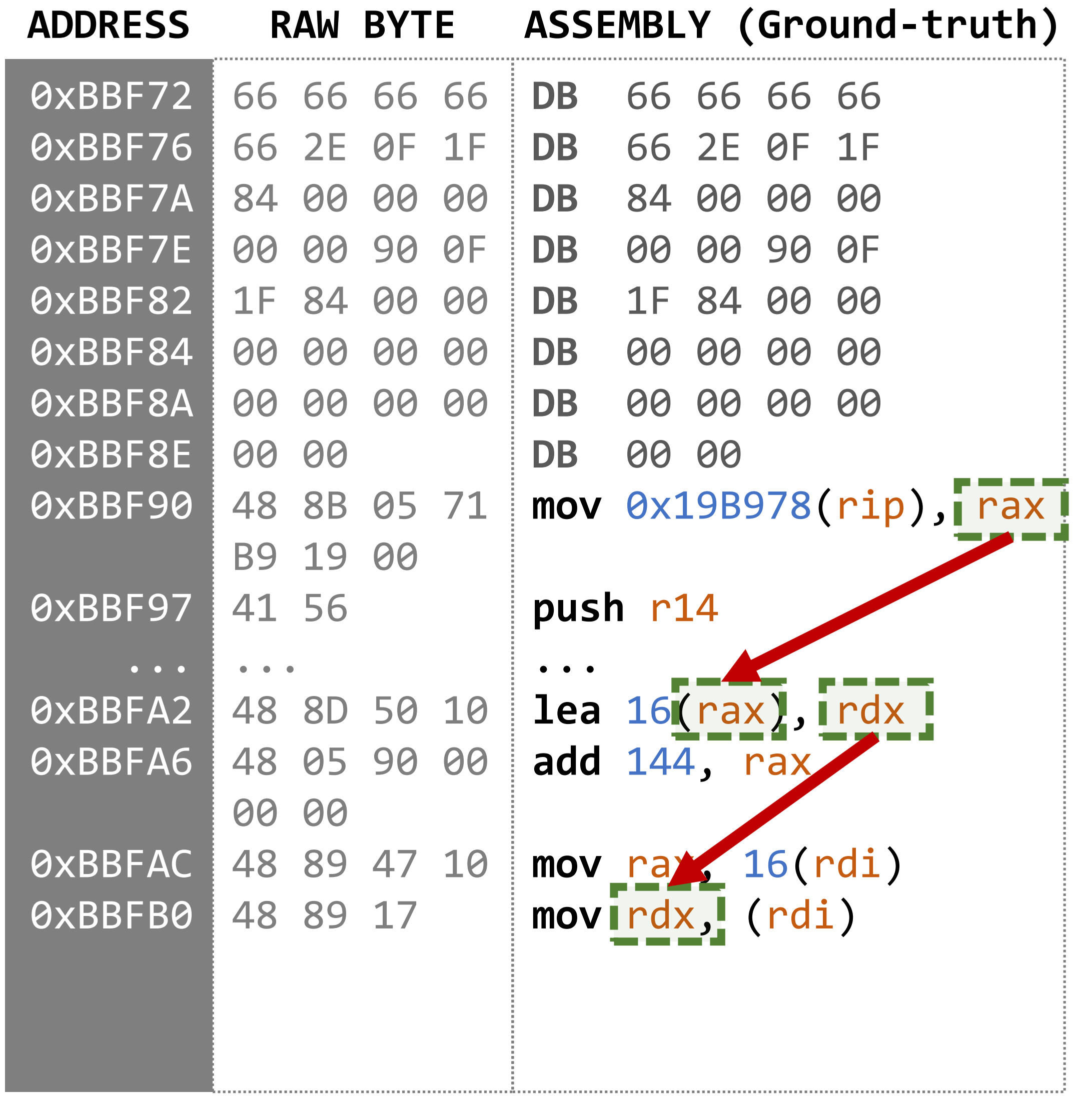


ADDRESS	RAW BYTE	ASSEMBLY (Ground-truth)
0xBBF72	66 66 66 66	DB 66 66 66 66
0xBBF76	66 2E 0F 1F	DB 66 2E 0F 1F
0xBBF7A	84 00 00 00	DB 84 00 00 00
0xBBF7		
0xBBF8		
0xBBF8		
0xBBF8		
0xBBF8		
0xBBF9		ax
0xBBF9		
..		
0xBBFA		
0xBBFA		
	00 00	
0xBBFAC	48 89 47 10	mov rax, 16(rdi)
0xBBFB0	48 89 17	mov rdx, (rdi)

Many *False Positives*

ADDRESS	RAW BYTE	Superset Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)
0xBBF73	66 66 .. 1F	nop cs:(rax+...)
0xBBF74	66 66 .. 84	nop cs:(rax+...)
...	...	...
0xBBF8C	00 00	add al, (rax)
0xBBF8D	00 00	add al, (rax)
0xBBF8E	00 00	add al, (rax)
0xBBF8F	00 48 8B	add cl, -117(rax)
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax
0xBFF92	05 71 .. 00	add 0x1685873, eax
...	...	...
0xBBF97	41 56	push r14
0xBBF98	56	push rsi
...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)
0xBBFB1	48 89 17	mov edx, (rdi)
...	...	...

Data and Code Coexistence: Probabilistic Disassembly



ADDRESS	RAW BYTE	ASSEMBLY (Ground-truth)
0xBBF72	66 66 66 66	DB 66 66 66 66
0xBBF76	66 2E 0F 1F	DB 66 2E 0F 1F
0xBBF7A	84 00 00 00	DB 84 00 00 00
0xBBF7E	00 00 90 0F	DB 00 00 90 0F
0xBBF82	1F 84 00 00	DB 1F 84 00 00
0xBBF84	00 00 00 00	DB 00 00 00 00
0xBBF8A	00 00 00 00	DB 00 00 00 00
0xBBF8E	00 00	DB 00 00
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF97	41 56	push r14
...	...	...
0xBBFA2	48 8D 50 10	lea 16(rax), rdx
0xBBFA6	48 05 90 00 00 00	add 144, rax
0xBBFAC	48 89 47 10	mov rax, 16(rdi)
0xBBFB0	48 89 17	mov rdx, (rdi)

ADDRESS	RAW BYTE	Superset Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)
0xBBF73	66 66 .. 1F	nop cs:(rax+...)
0xBBF74	66 66 .. 84	nop cs:(rax+...)
...	...	...
0xBBF8C	00 00	add al, (rax)
0xBBF8D	00 00	add al, (rax)
0xBBF8E	00 00	add al, (rax)
0xBBF8F	00 48 8B	add cl, -117(rax)
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax
0xBFF92	05 71 .. 00	add 0x1685873, eax
...	...	...
0xBBF97	41 56	push r14
0xBBF98	56	push rsi
...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)
0xBBFB1	48 89 17	mov edx, (rdi)
...	...	...

Data and Code Coexistence: Probabilistic Disassembly

ADDRESS	RAW BYTE	Superset Disassembly	Probabilistic Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)	0.04
0xBBF73	66 66 .. 1F	nop cs:(rax+...)	0.04
0xBBF74	66 66 .. 84	nop cs:(rax+...)	0.04
...	...	...	...
0xBBF8C	00 00	add al, (rax)	**0.695**
0xBBF8D	00 00	add al, (rax)	0.04
0xBBF8E	00 00	add al, (rax)	**0.695**
0xBBF8F	00 48 8B	add cl, -117(rax)	0.04
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax	**0.695**
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax	0.04
0xBFF92	05 71 .. 00	add 0x1685873, eax	0.04
...	...	...	...
0xBBF97	41 56	push r14	**0.94**
0xBBF98	56	push rsi	0.06
...	...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)	**1.0 (approx.)**
0xBBFB1	48 89 17	mov edx, (rdi)	0.0 (approx.)
...	...	...	...

Data and Code Coexistence: Probabilistic Disassembly

ADDRESS	RAW BYTE	Superset Disassembly	Probabilistic Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)	0.04
0xBBF73	66 66 .. 1F	nop cs:(rax+...)	0.04
0xBBF74	66 66 .. 84	nop cs:(rax+...)	0.04
...	...	...	...
0xBBF8C	00 00	add al, (rax)	**0.695**
0xBBF8D	00 00	add al, (rax)	0.04
0xBBF8E	00 00	add al, (rax)	**0.695**
0xBBF8F	00 48 8B	add cl, -117(rax)	0.04
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax	**0.695**
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax	0.04
0xBFF92	05 71 .. 00	add 0x1685873, eax	0.04
...	...	...	...
0xBBF97	41 56	push r14	**0.94**
0xBBF98	56	push rsi	0.06
...	...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)	**1.0 (approx.)**
0xBBFB1	48 89 17	mov edx, (rdi)	0.0 (approx.)
...	...	...	...

Data and Code Coexistence: Probabilistic Disassembly

ADDRESS	RAW BYTE	Superset Disassembly	Probabilistic Disassembly
0xBBF72	66 66 .. 00	nop cs:(rax,rax)	0.04
0xBBF73	66 66 .. 1F	nop cs:(rax+...)	0.04
0xBBF74	66 66 .. 84	nop cs:(rax+...)	0.04
...	...	...	...
0xBBF8C	00 00	add al, (rax)	**0.695**
0xBBF8D	00 00	add al, (rax)	0.04
0xBBF8E	00 00	add al, (rax)	**0.695**
0xBBF8F	00 48 8B	add cl, -117(rax)	0.04
0xBBF90	48 8B 05 71 B9 19 00	mov 0x19B978(rip), rax	**0.695**
0xBBF91	8B 05 .. 00	mov 0x19b978(rip), eax	0.04
0xBFF92	05 71 .. 00	add 0x1685873, eax	0.04
...	...	...	...
0xBBF97	41 56	push r14	**0.94**
0xBBF98	56	push rsi	0.06
...	...	...	...
0xBBFB0	48 89 47 10	mov rdx, (rdi)	**1.0 (approx.)**
0xBBFB1	48 89 17	mov edx, (rdi)	0.0 (approx.)
...	...	...	...

State-of-the-art: Where we are

Existing disassemblers have *various limitations*

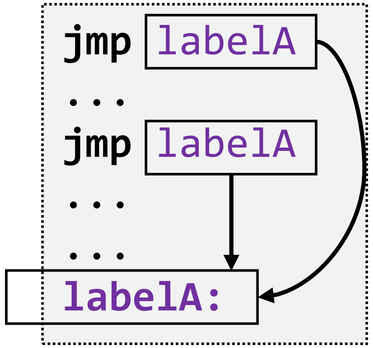
Disassembler	False Negatives	False Positives
Linear sweep	Some	Substantial
Traversal	Substantial	None
Superset	None	Bloated
Shingled Graph	Some	Some
BAP ByteWeight	Some	Some
Probabilistic Disassembly	***None****	***Some****

Desirable
Undesirable

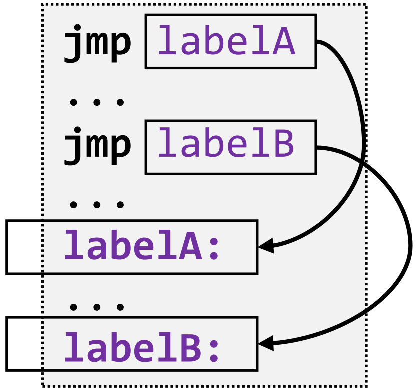
***With probabilistic guarantees**

Key Ideas: *Hints* of true instructions

Hint 1: Control flow convergence



Hint 2: Control flow crossing

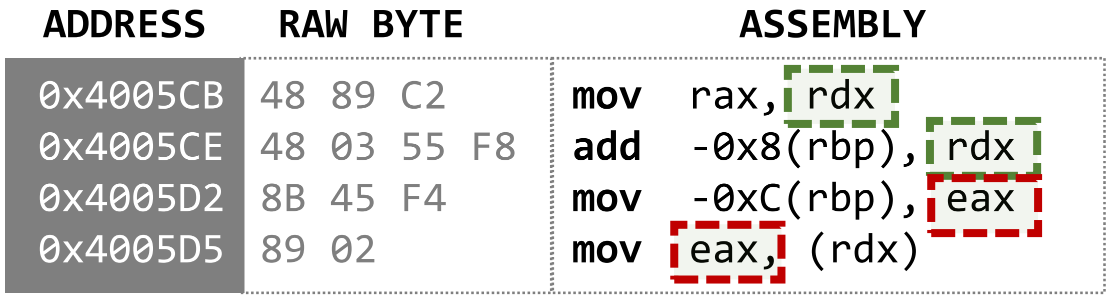


Unlikely *happen in disassembled* ***code from data (e.g., random) bytes***

Key Ideas: *Hints* of true instructions

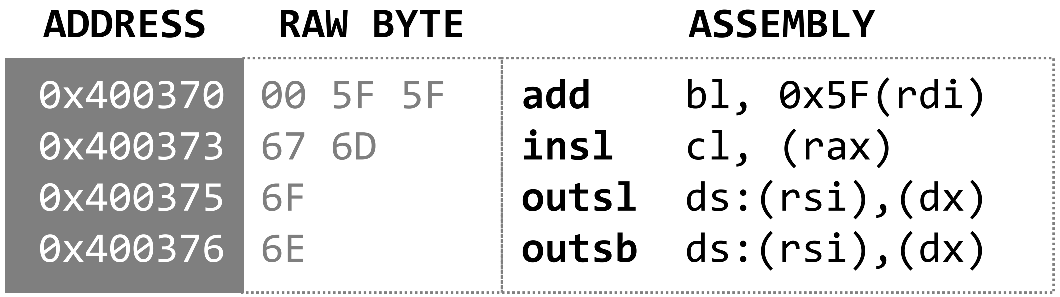
Hint 3: Register Def-use relation

(a) True Instructions (Reg. def-use)



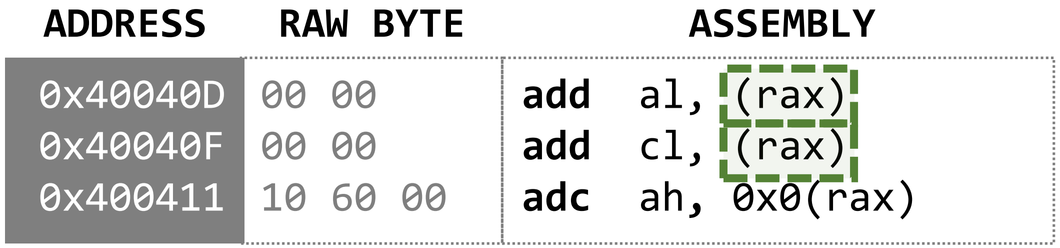
ADDRESS	RAW BYTE	ASSEMBLY
0x4005CB	48 89 C2	**mov** rax, rdx
0x4005CE	48 03 55 F8	**add** -0x8(rbp), rdx
0x4005D2	8B 45 F4	**mov** -0xC(rbp), eax
0x4005D5	89 02	**mov** eax, (rdx)

(b) String (No def-use)



ADDRESS	RAW BYTE	ASSEMBLY
0x400370	00 5F 5F	**add** bl, 0x5F(rdi)
0x400373	67 6D	**insl** cl, (rax)
0x400375	6F	**outsl** ds:(rsi),(dx)
0x400376	6E	**outsb** ds:(rsi),(dx)

(c) Jump Table (*Memory* def-use)



ADDRESS	RAW BYTE	ASSEMBLY
0x40040D	00 00	**add** al, (rax)
0x40040F	00 00	**add** cl, (rax)
0x400411	10 60 00	**adc** ah, 0x0(rax)

Computing Probabilistic Constraints

Reasoning Rules: probabilistic inference rules

1. If an instruction starting at an address is ***valid***, its *successor along control flow* ***must be a valid*** instruction.

$$C1: \quad x \quad \overset{1.0}{\longrightarrow} x_{succ}$$

2. If an instruction starting at an address is ***valid***, its *preceding instruction* ***may*** *be a valid instruction.*

$$C2: \quad x \quad \overset{p}{\rightarrow} x_{pre}$$

3. If an instruction starting at an address is ***valid***, *all the other addresses starting inside its body* are ***invalid***

…

Hints and Reasoning rules are encoded as *probabilistic constraints.*

Each instruction's probability is propagated to *its control flow successor(s)*

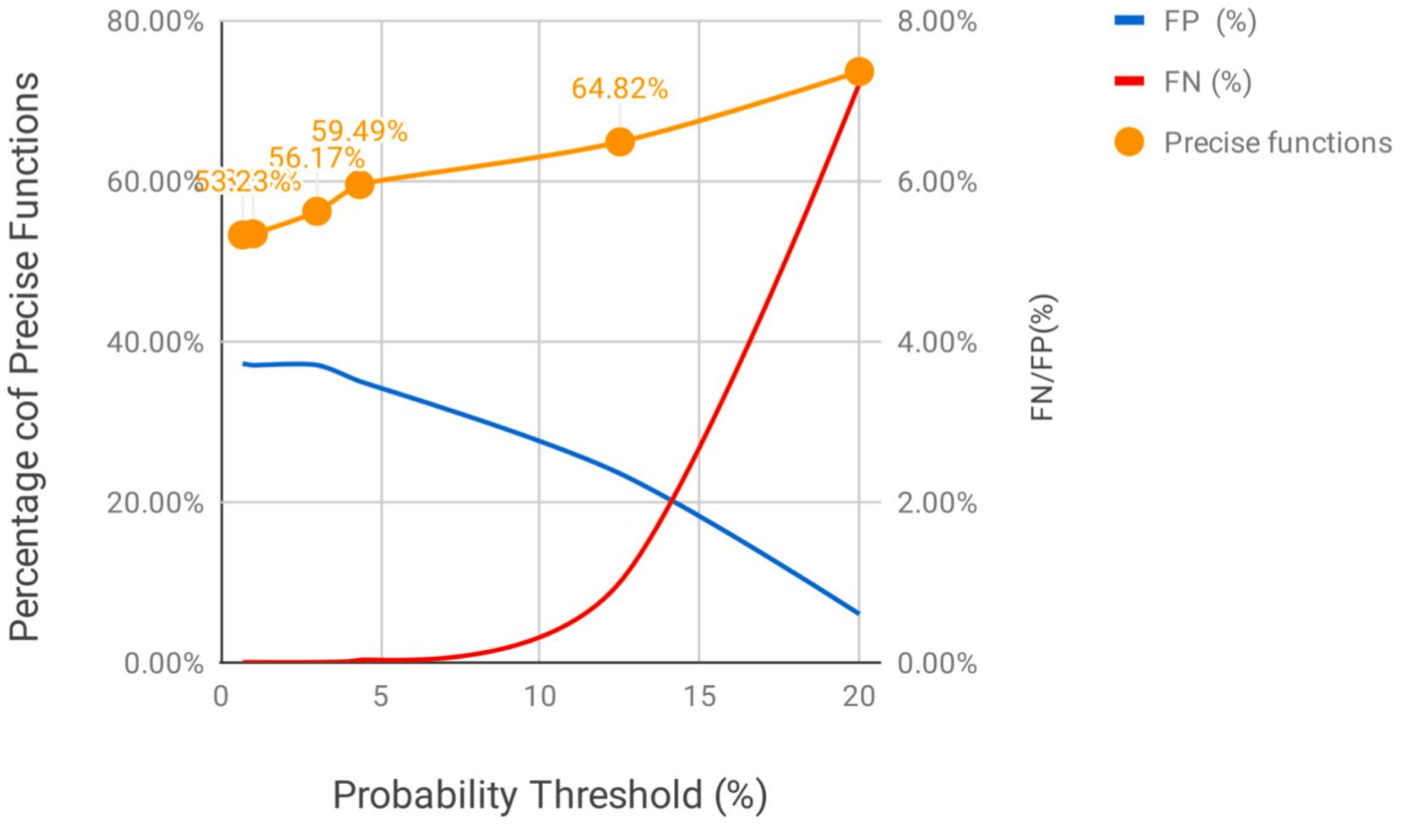
Implementation and Evaluation

- **Implementation**
  - ELF binaries: based on **BAP** (by CMU)
  - PE binaries: based on **Capstone**
  - Released at

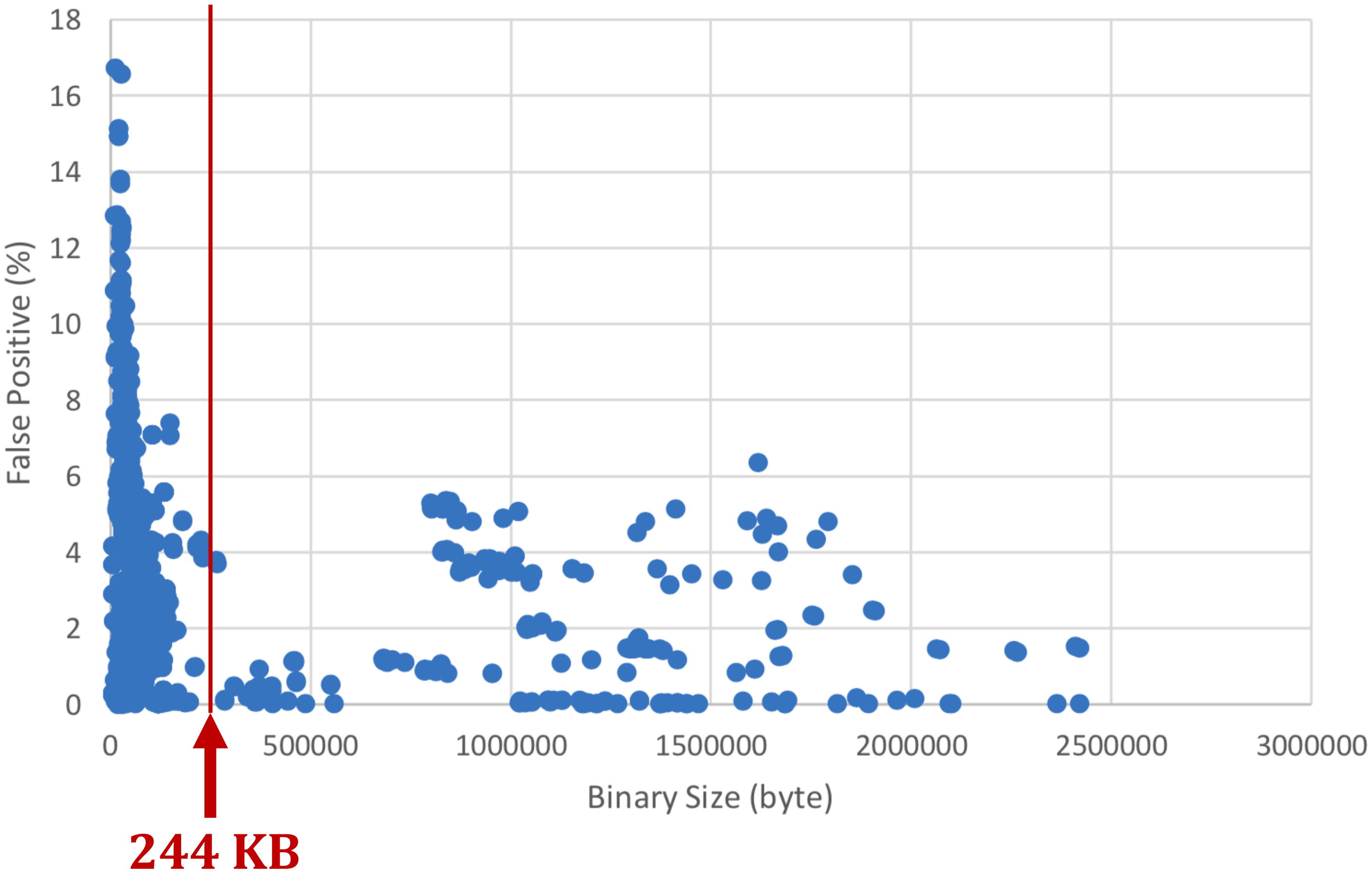
    https://github.com/KennethAdamMiller/superset_disassembler

- **Evaluation**
  - 2048 ELF binaries
  - SPEC2000 PE binaries

ELF Results: Tradeoffs of Threshold



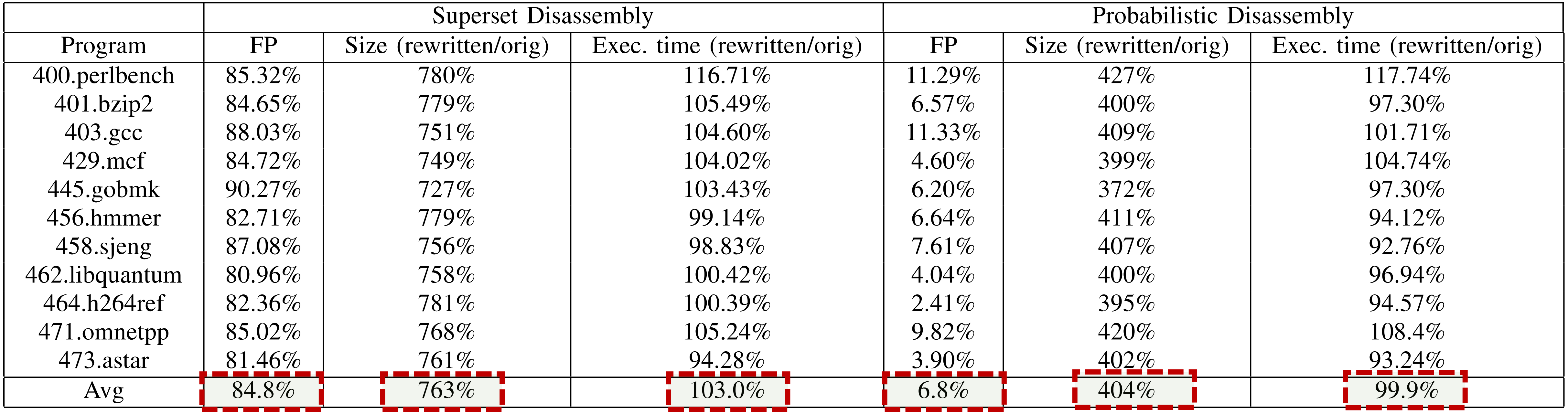
False Positive wrt Binary Size



Superset Disassembly vs. Probabilistic Disassembly

Superset Disassembly + Binary Rewriter* vs.
Probabilistic Disassembly + Binary Rewriter*

** The same binary rewriter. No added instructions*



	Superset Disassembly			Probabilistic Disassembly		
Program	FP	Size (rewritten/orig)	Exec. time (rewritten/orig)	FP	Size (rewritten/orig)	Exec. time (rewritten/orig)
400.perlbench	85.32%	780%	116.71%	11.29%	427%	117.74%
401.bzip2	84.65%	779%	105.49%	6.57%	400%	97.30%
403.gcc	88.03%	751%	104.60%	11.33%	409%	101.71%
429.mcf	84.72%	749%	104.02%	4.60%	399%	104.74%
445.gobmk	90.27%	727%	103.43%	6.20%	372%	97.30%
456.hmmer	82.71%	779%	99.14%	6.64%	411%	94.12%
458.sjeng	87.08%	756%	98.83%	7.61%	407%	92.76%
462.libquantum	80.96%	758%	100.42%	4.04%	400%	96.94%
464.h264ref	82.36%	781%	100.39%	2.41%	395%	94.57%
471.omnetpp	85.02%	768%	105.24%	9.82%	420%	108.4%
473.astar	81.46%	761%	94.28%	3.90%	402%	93.24%
Avg	84.8%	763%	103.0%	6.8%	404%	99.9%

TABLE II: Superset Disassembly vs Probabilistic Disassembly

Related Work

- Linear Sweep Disassembly
  - SecondWrite (WCRE’13), D. Andriesse et al. (Security’15), …
- Traversal based Disassembly
  - IDA, BAP, N. E. Rosenblum et al. (AAAI’08), BYTEWEIGHT (Security’14), E. C. R. Shin et al. (Usenix’15), D. Andriesse et al. (Security’15), R. Qiao et al (DSN’17), …
- Superset Disassembly (NDSS’18)
- Probabilistic Inference in Program Analysis
  - G. K. Baah et al. (ISSTA’08), Merlin (PLDI’09), Dimsum (NDSS’12), …
- Machine Learning for Binary Analysis
  - R. Wartell (PKDD’11), Shingled graph disassembly (PAKDD’14), …
- Dynamic Disassembly
  - D. Bruening et al. (CGO’03), H. Patil et al. (MICRO’04), BIRD, …

Discussions and Questions

- Limitations:
  - Our benchmarks may not represent all possible real-world binaries
  - We focus on compiler generated binaries. Hand-crafted binaries/obfuscated binaries are not tested.
    - However, we believe semantic hints still exist in such code
- Source code available!
  https://github.com/KennethAdamMiller/superset_disassembler