Similarity of Binaries through re-Optimization

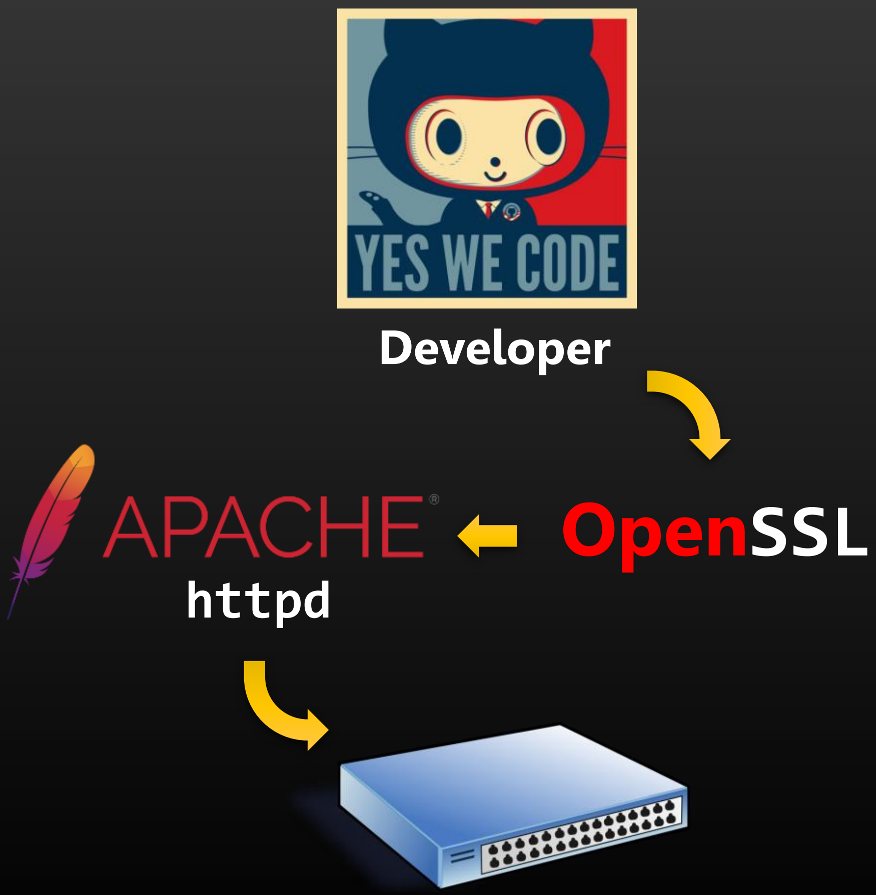
By

Yaniv David, **Nimrod Partush** & Eran Yahav

Technion, Israel

The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688– ERC- COG-PRIME.

Motivation

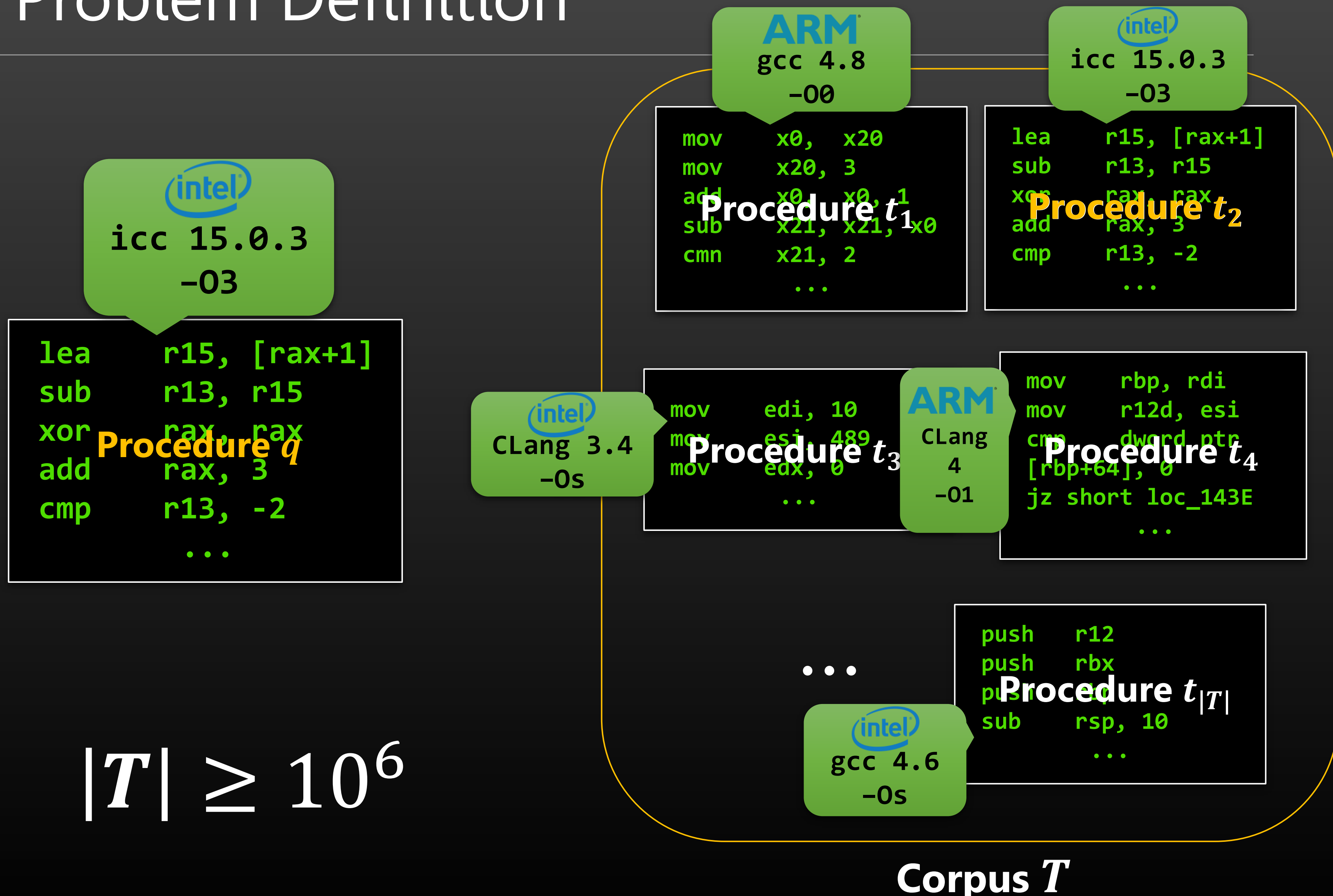


Motivation

OpenSSL

Security Researcher

Problem Definition



$$|T| \geq 10^6$$

Challenge

OpenSSL's

`dtls1_buffer_message()`

```
mov     x0,   x20
mov     x20,  3
add     x0,   x0,  1
sub     x21,  x21, x0
cmn     x21,  2
```

ARM

gcc 4.8

-O0

```
lea     r15, [rax+1]
sub     r13, r15
xor     rax, rax
add     rax, 3
cmp     r13, -2
```

intel

icc 15.0.3

-O3

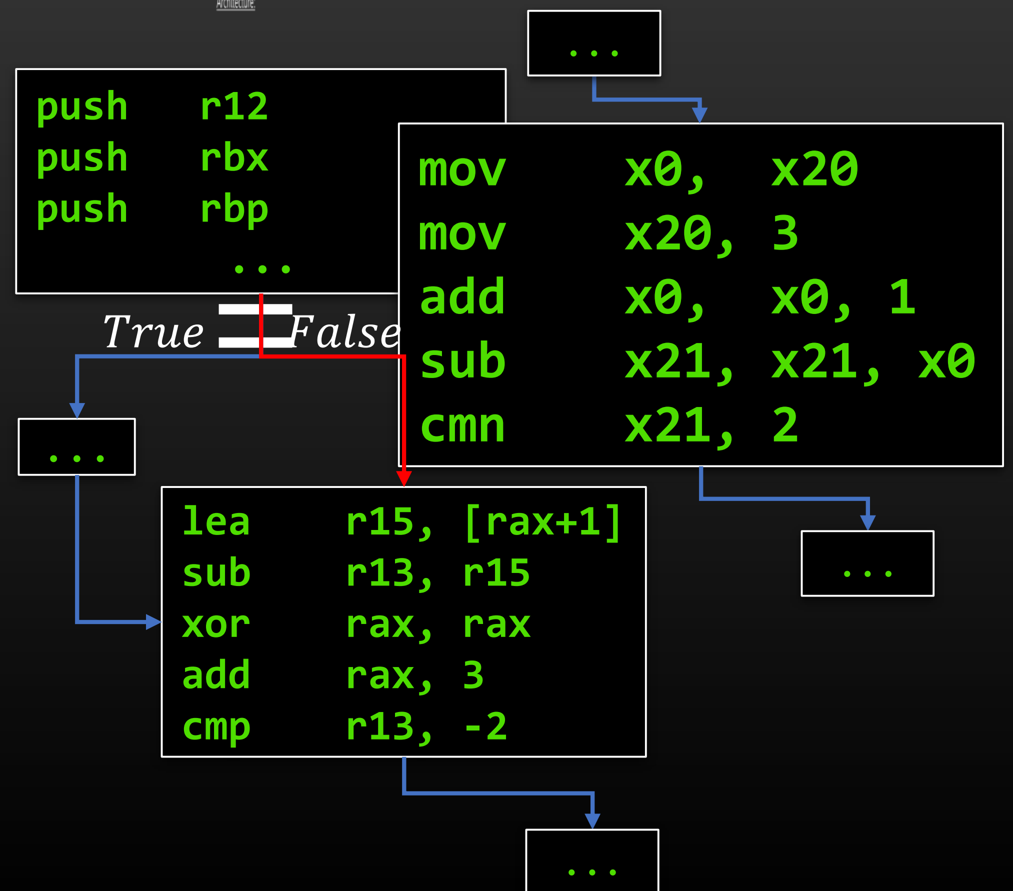
Our Approach

For finding Similarity of Binaries

Our Approach: **What**

Query *q*: `dtls1_buffer_message()`
Compiler: **icc 15.0.3 -O3**
Architecture:

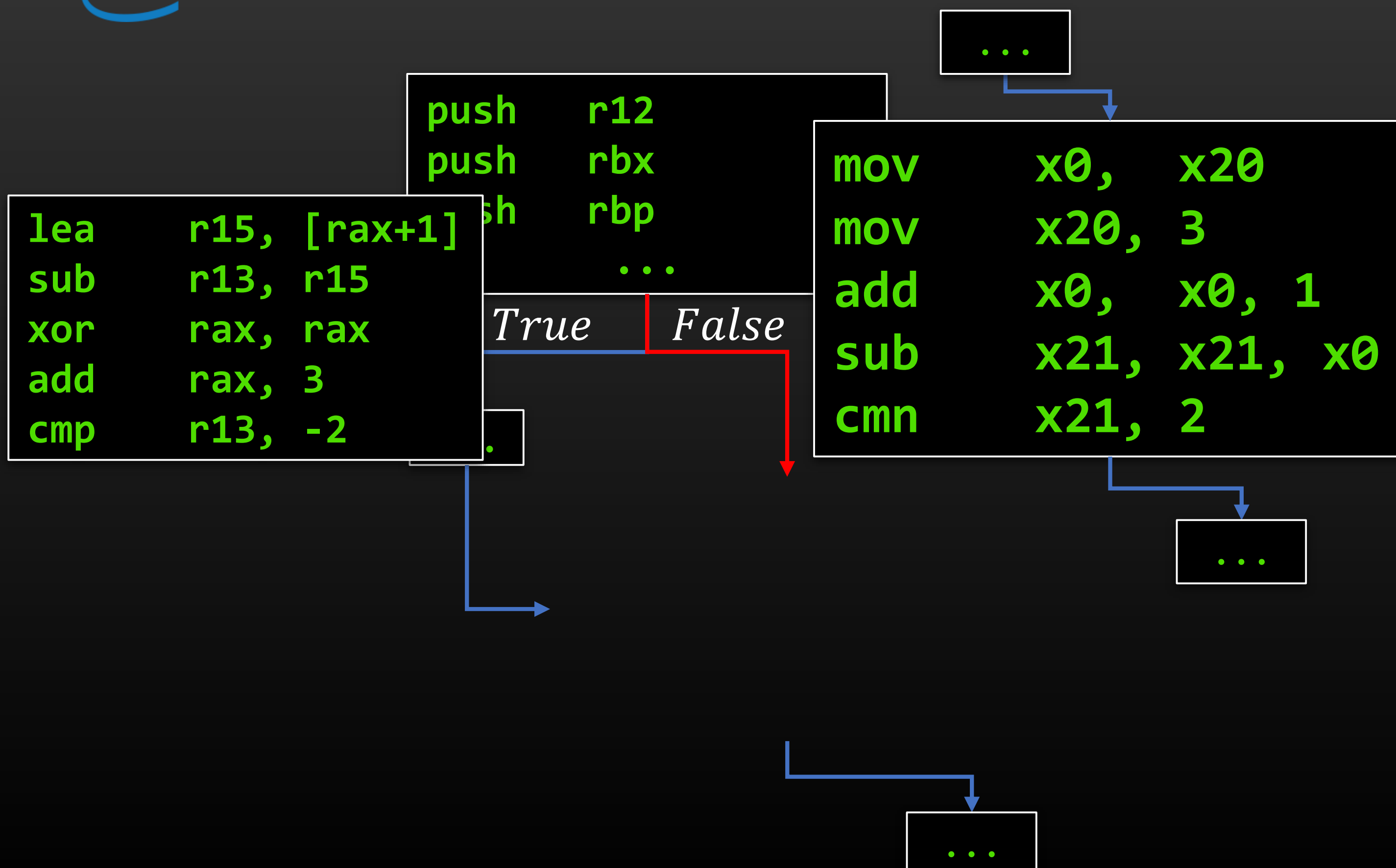
Query *q*: `dtls1_buffer_message()`
Compiler: **gcc 4.8 -O0**
Architecture:



Our Approach: **What**

Query *q*: `dtls1_buffer_message()`
Compiler: **`icc 15.0.3 -O3`**
Architecture: intel

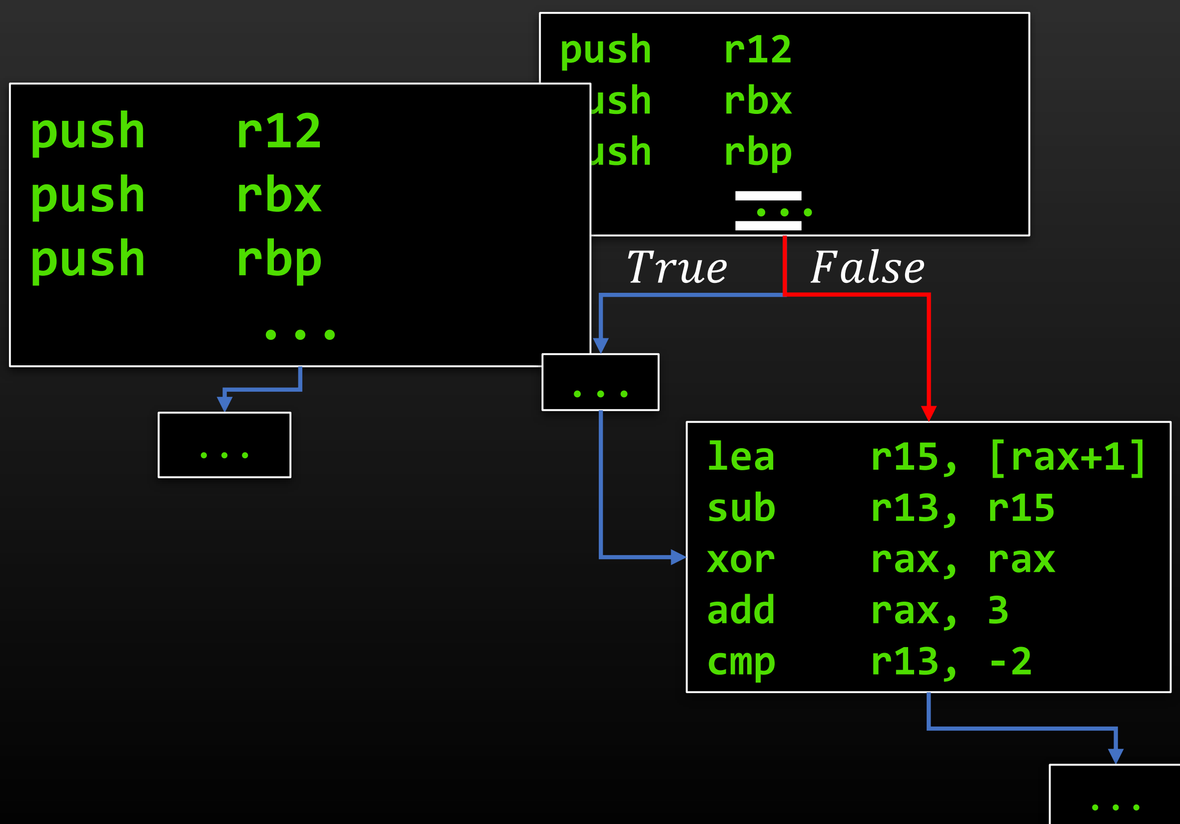
Query *q*: `dtls1_buffer_message()`
Compiler: **`gcc 4.8 -O0`**
Architecture: ARM



Our Approach: **What**

Procedure t_2: unrelated()
Compiler: **icc 15.0.3 -O3**
Architecture:

Query q: dtls1_buffer_message()
Compiler: **icc 15.0.3 -O3**
Architecture:



Our Approach: **How**

- **Decompose** procedure to fragments
- **Transform** fragments to canonical form

```
mov     x0,  x20
add     x0,  x0, 1
sub     x21, x21, x0
cmn     x21, 2
```

```
lea     r15, [rax+1]
sub     r13, r15
cmp     r13, -2
```

```
tmp0 ष़ register0 + झ़
register1 = tmp0
tmp1 ष़ registerञ़ – tmp0
registerञ़ = tmp1
tmp2 ष़ tmp1 - ञ़
```

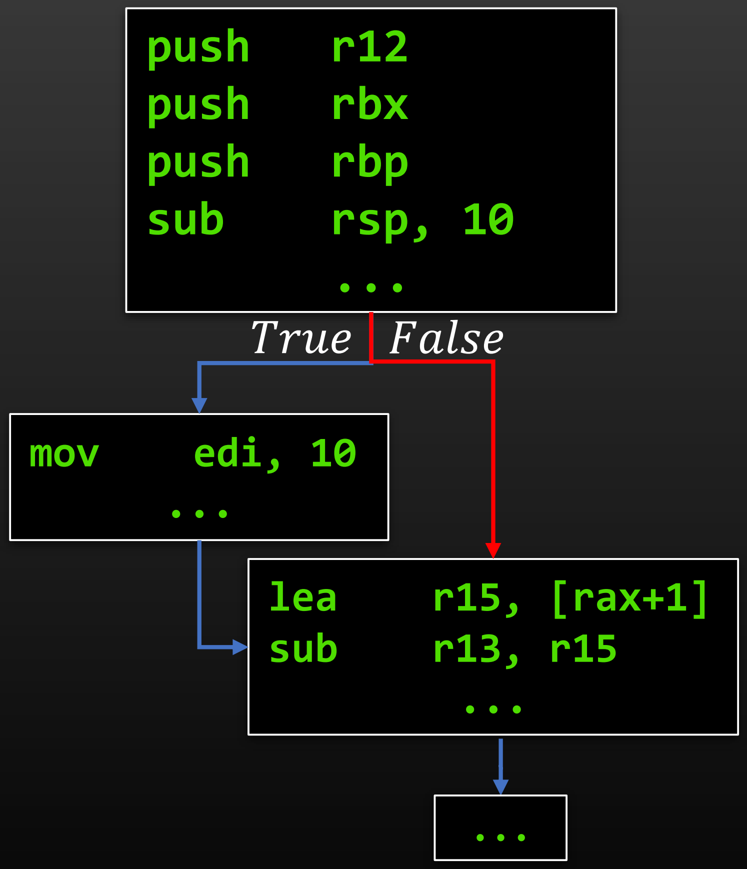
=

```
tmp0 ष़ register0 + झ़
register1 = tmp0
tmp1 ष़ registerञ़ – tmp0
registerञ़ = tmp1
tmp2 ष़ tmp1 - ञ़
```

- Count shared fragments while weighing in their **statistical significance**

Decomposing Assembly Procedures

- The procedure is broken at **basic block** level
  - Block ordering is ignored



Slicing Basic Blocks

- We use slicing to break basic blocks into separate data-independent computations:

```
mov     x0,   x20
mov     x20,  3
add     x0,   x0, 1
sub     x21,  x21, x0
cmn     x21,  2
```

Slice

slice 1

```
mov     x0,   x20
add     x0,   x0, 1
sub     x21,  x21, x0
cmn     x21,  2
```

slice 2

```
mov     x20,  3
```

Moving to Canonical Form (re-Optimizing)

- “Out-of-context” re-optimization

```
mov     x0,   x20
add     x0,   x0, 1
sub     x21,  x21, x0
cmn     x21,  2
```

Lift

LLVM

```
t0   = load x20
store t0, x0
t1 = load x0
t2 = 1
t3 = add t1, t2
store t3, x0
      ...
```

Optimize

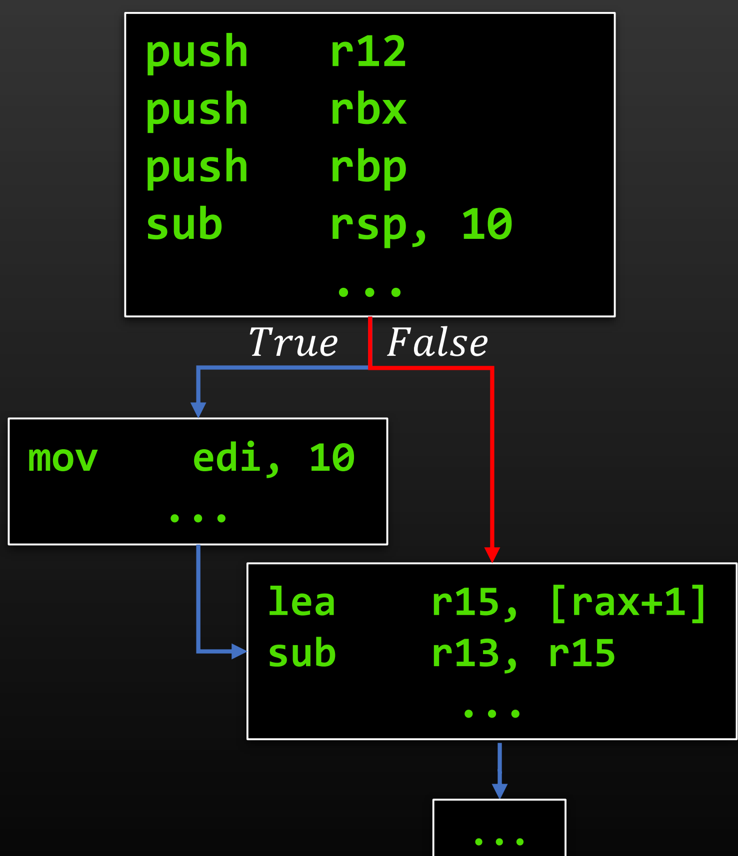
```
t0   = load x20
t1 = add t0, 1
store t1, x0
      ...
```

Normalize

```
t0   = load r0
t1 = add t0, 1
store t1, r1
      ...
```

Procedure Representation

dtls1_buffer_message()



R(dtls1_buffer_message)=

{

```
t0  = load r0
t1 = add t0▯ झ
store tझ▯r1
t2 = load rञ
t3 = sub t2, tझ
store t3▯rञ
```

,

```
t0  = load r0
t1 ष sub t0▯ 10
t2 = load r1
t3 ष add t2, 3
t3 ष mul t3, tझ
store t3▯rञ
```

, … }

Computing Similarity

$$Similarity(q, t) = |R(q) \cap R(t)|$$

- Reminder:

`unrelated()`
intel, icc 15.0.3, -O3

```
push    r12
push    rbx
push    rbp
sub     rsp, 10
```

=

`dtls1_buffer_message()`
intel, icc 15.0.3, -O3

```
push    r12
push    rbx
push    rbp
sub     rsp, 10
```

Statistical Significance

- Given a canonical fragment s, we need to determine its significance.

$$\Pr_{P}(s) = \frac{|\{\, p \in P \mid s \in R(p) \,\}|}{|P|}$$

Set of all procedures, in existence

- We estimate W with a bound, random sample of procedures P – a "**Global Context**"

Computing Similarity

$$Similarity(q,t) = \sum_{\substack{s\,\in \\ R(q)\cap R(t)}} \frac{1}{Pr_P(s)}$$

A distinctive fragment with $\underset{P}{Pr}(f) = 0.001$ will contribute 1000

A sum ranging over the *shared canonical fragments*

A common fragment with $\underset{P}{Pr}(f) = \frac{1}{5}$ will contribute 5

Evaluation

Prototype Implementation: **`GitZ`**

`GitZ` Output

Procedure q: **OpenSSL**'s `dtls1_buffer_message()`

Security Researcher

1. **Procedure t_{42}**
Similarity: 170.34

2. **Procedure t_{13}**
Similarity: 168.91

3. **Procedure t_{900}**
Similarity: 130.41

4. **Procedure $t_{218,777}$**
Similarity: 101.11

5. **Procedure $t_{43,081}$**
Similarity: 13.19

…

Evaluation Corpus

- Real-world code packages
  - OpenSSL, BASH , , git, , QEMU, wget, ffmpeg, Coreutils
  - Containing ~11,000 procedures

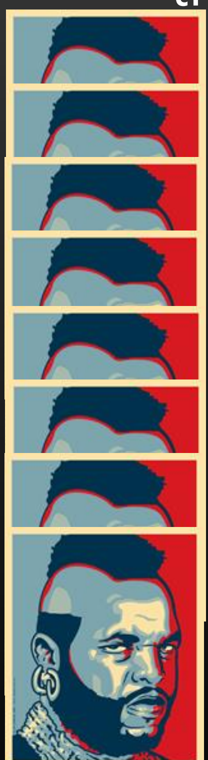
- Compiled to:
  - x86_64 with CLang 3.{4,5}, gcc 4.{6,8,9} and icc {14,15} x 9
  - ARM-64 with aarch64-gcc 4.8 and aarch64-Clang 4
- Optimization levels -O{0,1,2,3,s} x 5

- Corpus size: $|T| = 45 * \sim 11{,}000 = \sim 500{,}000$

- Queries: 9 procedures from notable CVEs

GitZ Accuracy

- Here, we report accuracy as the number of FPs ranked above the lowest TP



1. **Procedure t_{42}** — **Positive**
   Similarity: 170.34
2. **Procedure t_{13}** — **Positive**
   Similarity: 168.91
3. **Procedure t_{900}** — **Negative**
   Similarity: 130.41
4. **Procedure $t_{218,777}$** — **Positive**
   Similarity: 101.11
5. **Procedure $t_{43,081}$** — **Negative**
   Similarity: 13.19

... — Negative Negative Negative

500,000. Procedure t_{81} — **Negative**
Similarity: 0.0

$$\#FP = 1$$

$$FPr = \frac{1}{500,000}$$

GitZ Scalability

GitZ Usefulness

- How well does GitZ scale?
- How useful is GitZ in the vulnerability search scenario?

0.12s for query-target pair, single core

#	Alias	CVE	#FPs	FP Rate	Time
1	Heartbleed	2014-0160	52	0.000104	15m
2	Shellshock	2014-6271	0	0	17m
3	Venom	2015-3456	0	0	16m
4	Clobberin'	2014-9295	0	0	16m
5	Shellshock #2	2014-7169	0	0	12m
6	WS-snmp	2011-0444	0	0	14m
7	wget	2014-4877	0	0	10m
8	ffmpeg	2015-6826	0	0	17m
9	WS-statx	2014-8710	0	0	18m

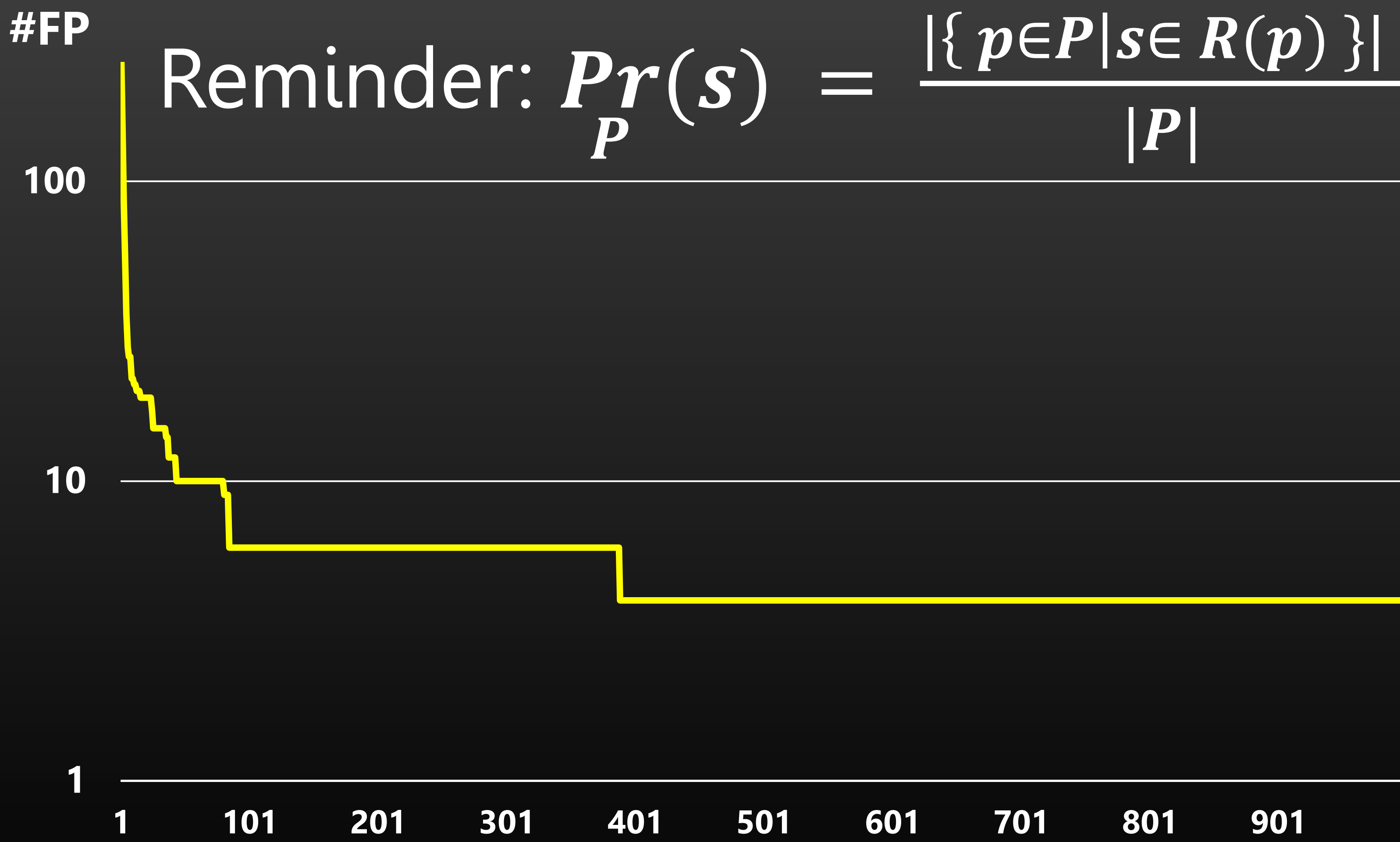
Evaluating Solution Components

- How does each component of our solution affect the accuracy of **`GitZ`**?

Query	**Corpus Size $\|T\|$**	**#Positives**
Heartbleed	10000	45

Evaluating the Global Context

Reminder: $\Pr_{P}(s) = \dfrac{|\{\, p \in P \,|\, s \in R(p) \,\}|}{|P|}$

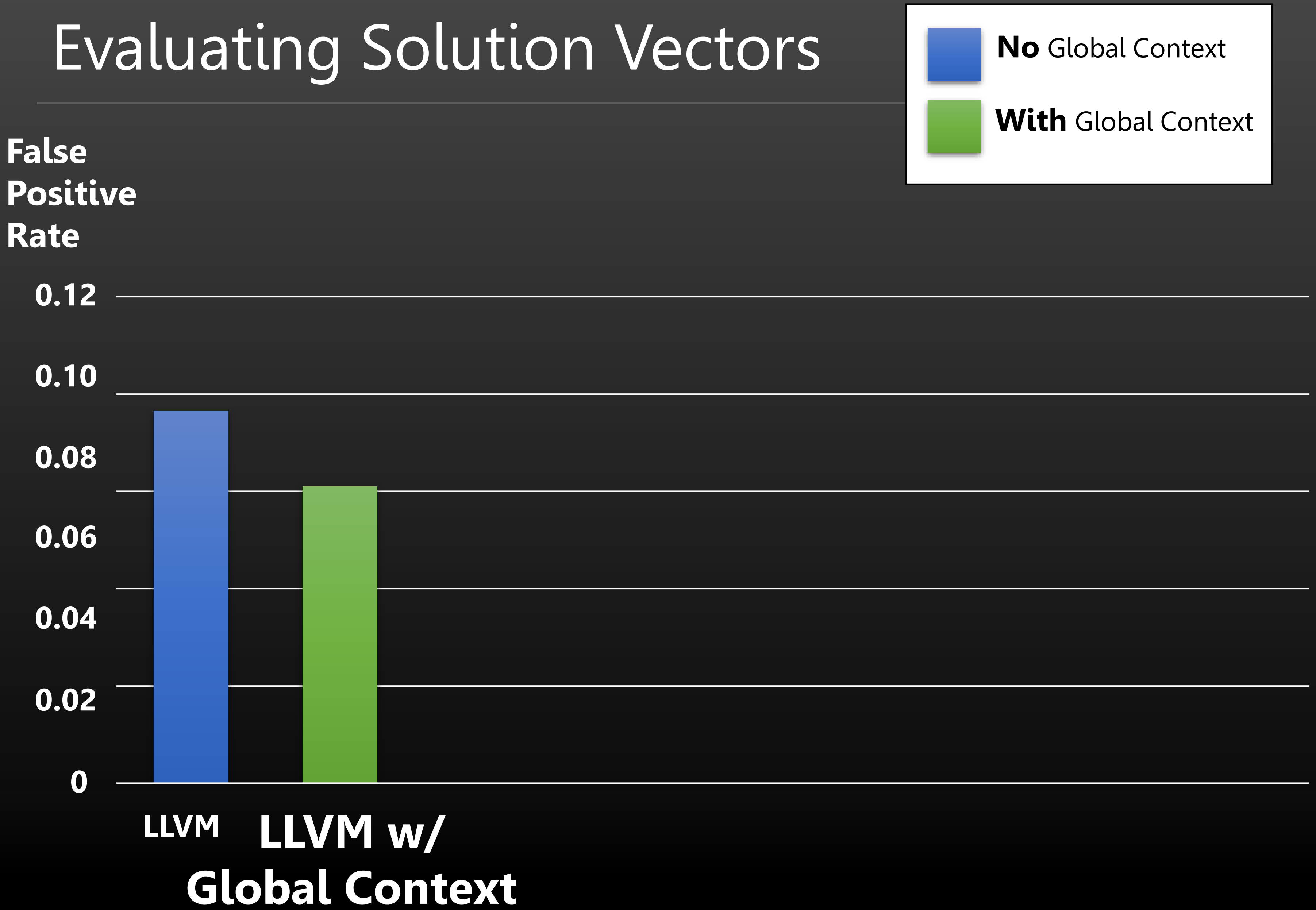


Size of P (in # Procedures)

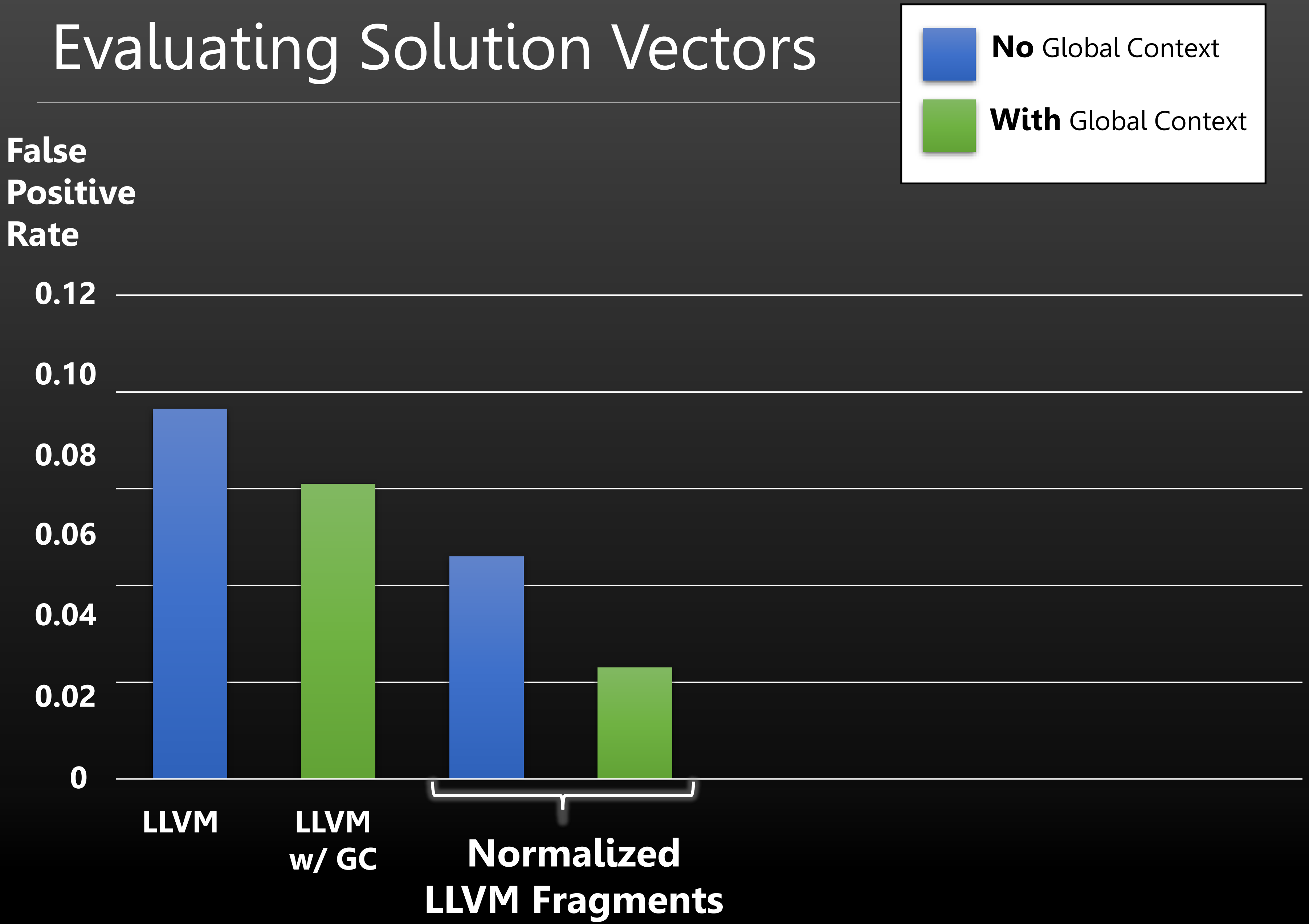
Evaluating Solution Vectors

Evaluating Solution Vectors

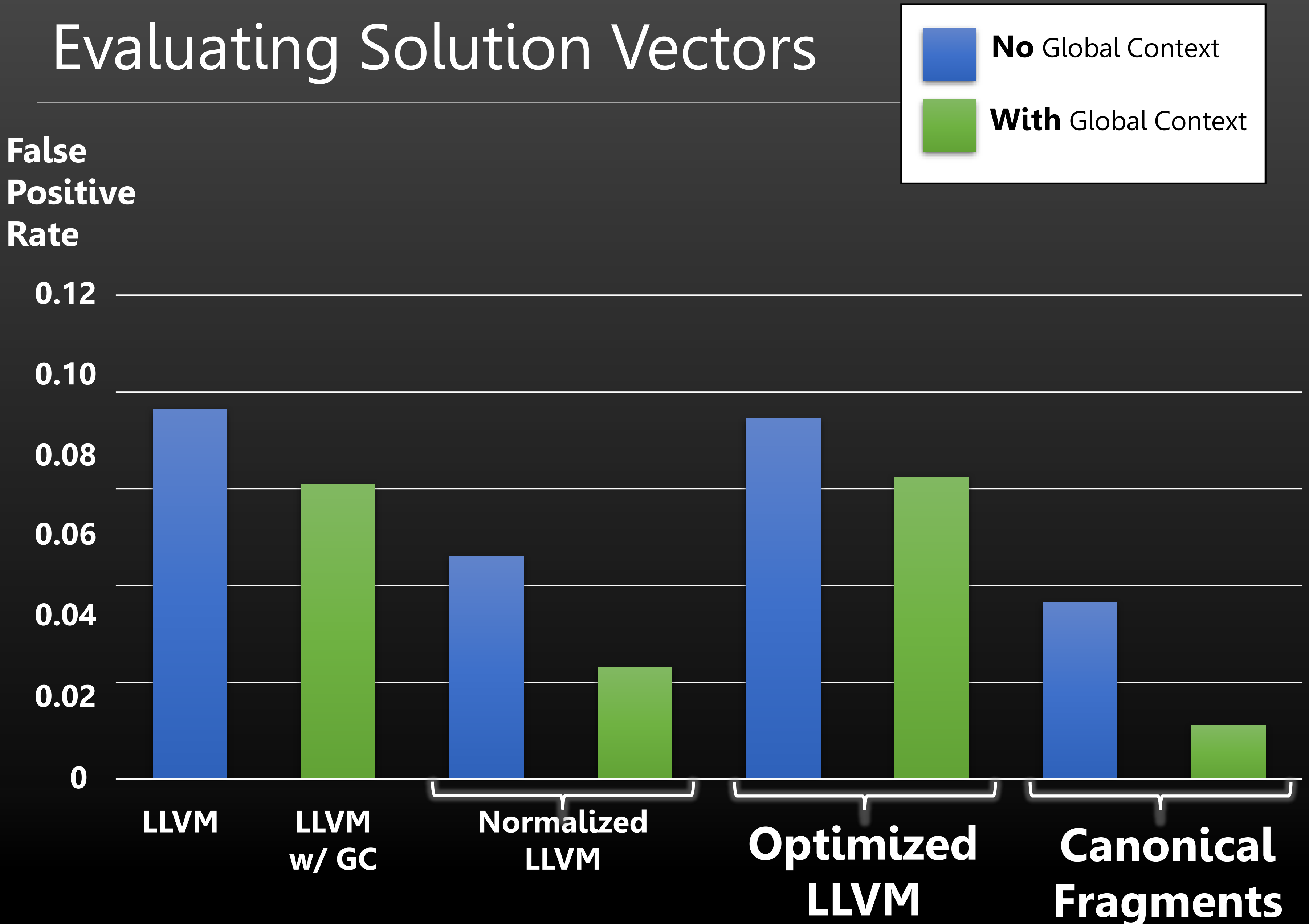
Evaluating Solution Vectors



Evaluating Solution Vectors



Evaluating Solution Vectors



Take Aways

- A procedure can be **identified** using a set of **statistically significant fragments**
  - The statistical data can be collected over a **relatively small set**

- Applying an optimizer "**out-of-context**" is useful at transforming fragments to canonical form
  - A form that allows finding similarity

Questions

- Canonical Form: The Good
- Canonical Form: The Bad
- Previous Work
- BinDiff
- More Experiments!
- You’re over-thinking this
- Out-of-Context?
- Limitations
- Evaluating Binary Classifiers
- All v. All
- Is Pr(s) a probability even?
- The Global Context

Canonical Form: The Good

```
mov     rax,  -2
sub     rbx,  rax
```

Canonical →

```
t0 = load r0
t1 = add 2, t0
store t1, r0
```

=

```
add     r12,  2
```

Canonical →

```
t0 = load r0
t1 = add 2, t0
store t1, r0
```

The Bad:

```
add     rax,  1
add     rbx,  2
add     rbx,  rax
```

Canonical →

```
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r1
```

=

```
add     rbx,  2
add     rax,  1
add     rax,  rbx
```

Canonical →

```
t0 = load r0
t1 = add 2, t0
store t1, r0
```

Canonical Form: The Bad

```
add     rax,  1
add     rbx,  2
add     rbx,  rax
```

Canonical

```
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r1
```

≠

```
add     rax,  1
add     rbx,  2
add     rax,  rbx
```

Canonical

```
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r0
```

Previous Work

	GitZ-1500: Cross-{Comp, Arch, Opt}			Esh-1500: Cross-Comp		
	#FPs	CROC	⏱	#FPs	CROC	⏱
Heartbleed	0	1	1s	0	1	19h
Shellshock	0	1	3s	3	.996	15h
Venom	0	1	1s	0	1	16h
Clobberin' Time	0	1	2s	19	.956	16h
Shellshock #2	0	1	2s	0	1	11h
WS-snmp	0	1	1s	1	.997	10h
wget	0	1	2s	0	1	15h
ffmpeg	0	1	1s	0	1	20h
WS-statx	0	1	2s	-	-	-

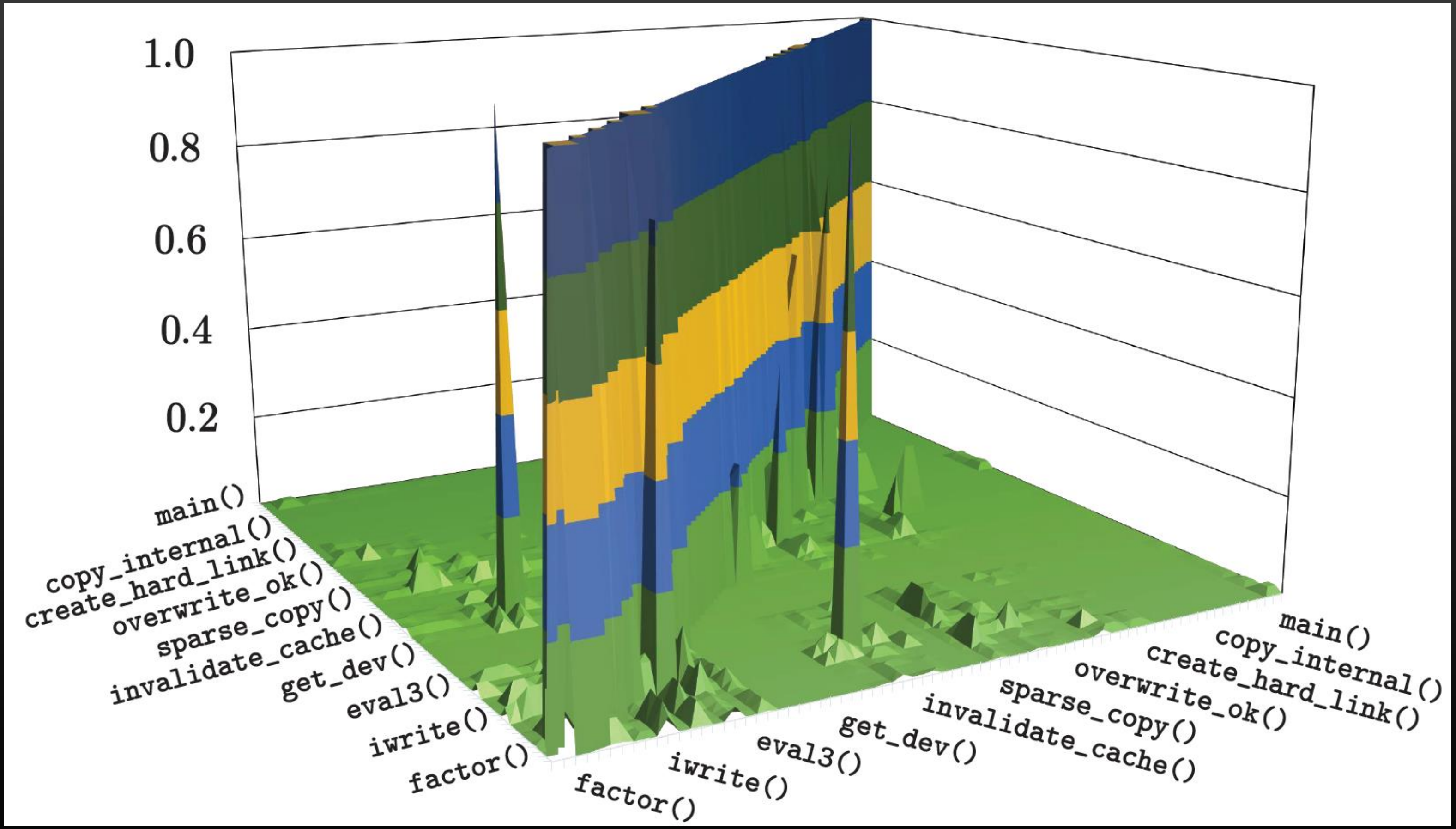
BinDiff

Alias	Matched?	Similarity	Confidence
Heartbleed	✗	-	-
Shellshock	✗	-	-
Venom	✗	-	-
Clobberin' Time	✗	-	-
Shellshock #2	✗	-	-
ws-snmp	✓	0.89	0.91
wget	✗	-	-
ffmpeg	✓	0.72	0.79

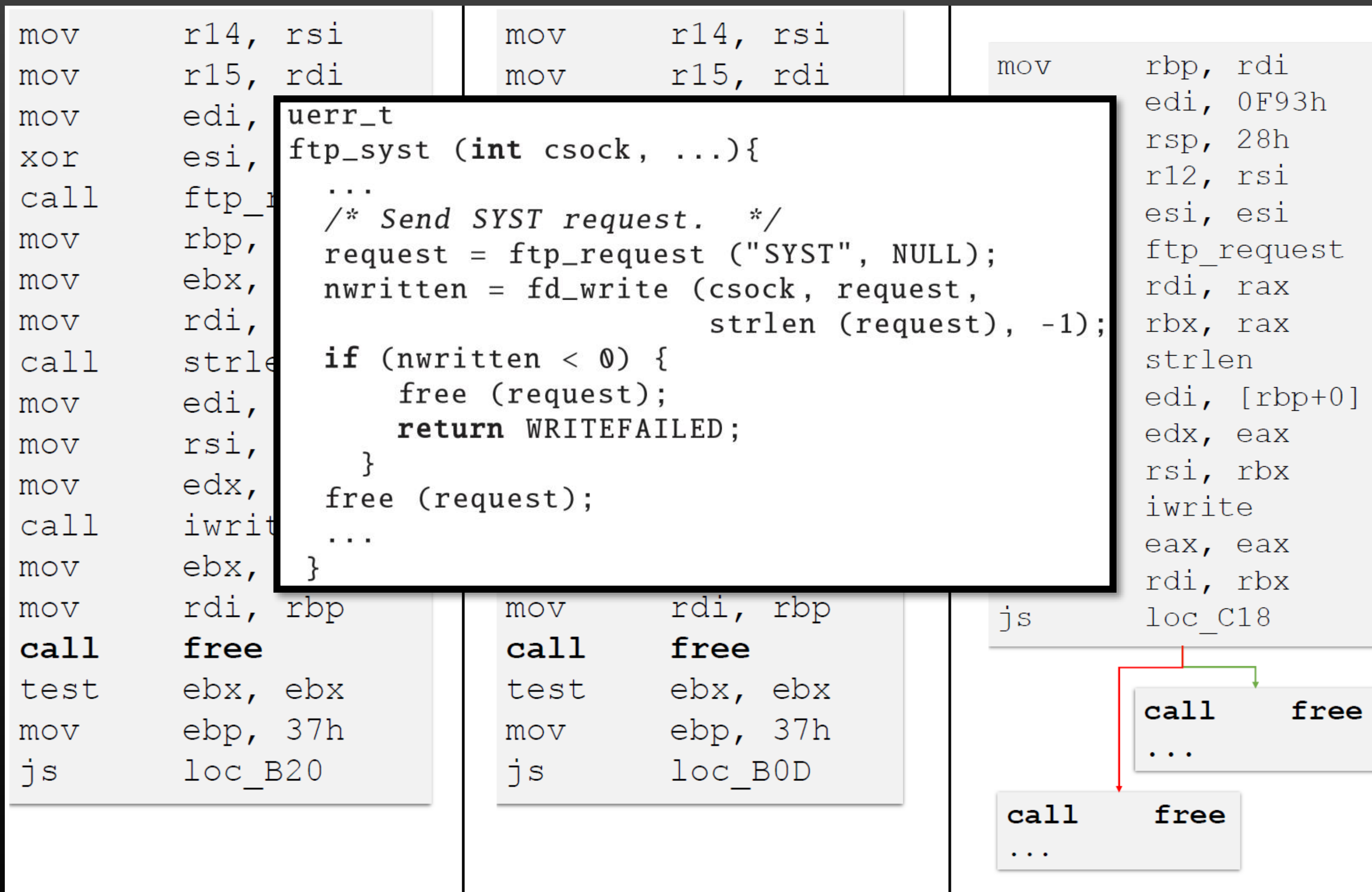
The Global Context

- Sampled over **canonical fragments**, from procedures of all archs\compilers\optimizations
- A new arch\compiler\optimization?
  - We are somewhat future proof due to optimization
    - Even if a compiler decides to do things a bit different, it should arrive at the same canonical form
  - Entirely new behaviors will require a (partial) resampling
    - For instance: using the stack in offsets of 13 O_0

All v. All


1.0
0.8
0.6
0.4
0.2
main()
copy_internal()
create_hard_link()
overwrite_ok()
sparse_copy()
invalidate_cache()
get_dev()
eval3()
iwrite()
factor()
main()
copy_internal()
create_hard_link()
overwrite_ok()
sparse_copy()
invalidate_cache()
get_dev()
eval3()
iwrite()
factor()

Limitations



(b) CLang 3.5 -O2

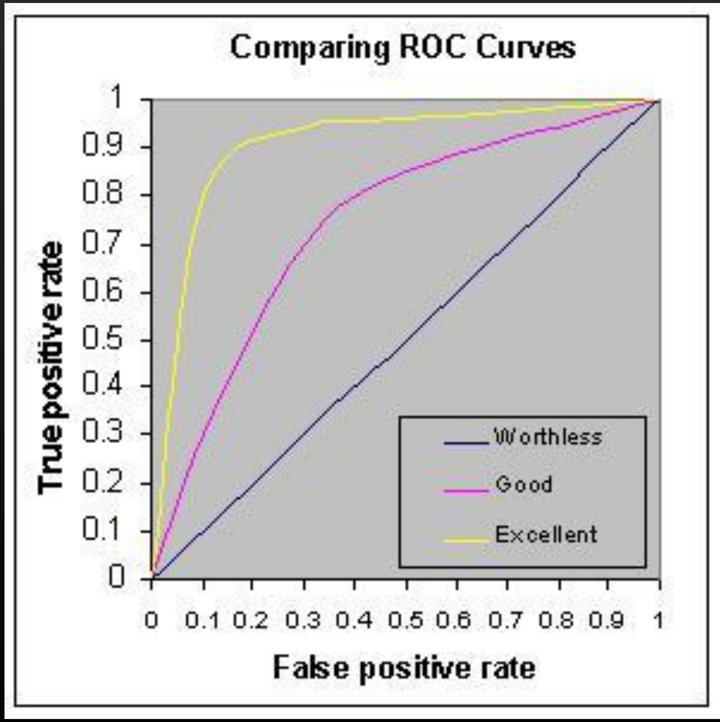
(c) CLang 3.4 -O2

(d) gcc 4.6 -O2

Evaluating Binary Classifiers

- The Receiver Operating Characteristic (ROC) is a widespread method for evaluating a binary classifier, by plotting the ratio of TPs to FPs, for all the different thresholds
- The Area Under Curve is then summed and value between 0-1 is produced. Our results were > .96.

- ROC means "how well did we cover all true positives, before we encounter false positives"
- We used Concentrated ROC, an adaptation for huge corpora, which further "punishes" highly ranked FPs



Jaccard Index?

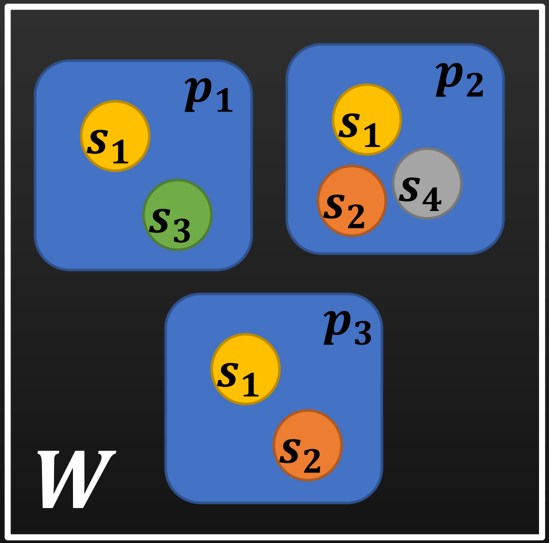
- Major difference: does not take statistical significance into account, at all.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Is $\Pr(s)$ a probability even?

$$\Pr_W(s) = \frac{|\{\, p \in W \mid s \in R(p) \,\}|}{|W|}$$

- $\Pr_W(s)$ is a probability over the sample space of W
  - W is a "multiset" of all canonical fragments in existence
  - $|\{\, p \in W \mid s \in R(p) \,\}|$ counts the occurrences of s in W
  - $\Pr(s_1) + \Pr(s_2) + \Pr(s_3) + \Pr(s_4) = \frac{3}{7} + \frac{2}{7} + \frac{1}{7} + \frac{1}{7} = 1$



- $\Pr_p(s)$ is an estimation of W, which betters as P grows
  - As we evaluated

You're over-thinking this

- Why not just run the binary and get a version string??
  - Sometimes a lib is embedded
  - You can't always easily run (different arch, dependencies, etc.)
  - Running can put you in unnecessary risk
  - Purely static
  - We've seen cases where the version string is not maintained correctly!

Out-of-Context?

- In context: The slice is surrounded with:
  - Other instructions from the block
  - Other blocks
  - Other procedures

The optimizer here must account for the surroundings, and cannot easily “cut-through” unrelated operations

- Out-of-context: Just the slice. The optimizer can easily extract a concise canonical fragment, that can be matched with semantically equivalent fragments from other procedures.

More Experiments!

Scenario	Queries	Targets	FP Rate
Cross-Optimization **ARM-64**	`aarch64-gcc 4.8 -O*`	`aarch64-gcc 4.8 -O*`	**0**

Scenario	Queries	Targets	FP Rate
Cross-(Optimization ∨ Version) **x86_64**	`gcc 4.{6,8,9} -O*`	`gcc 4.{6,8,9} -O*`	**0.001**

Scenario	Queries	Targets	FP Rate
Cross-Compiler **x86_64**	$Compilers_{x86}$ -O1	$Compilers_{x86}$ -O1	**0.002**

`GitZ` Accuracy: Cross-Compiler/Optimization/Architecture

Scenario	Queries	Targets	FP Rate
Cross-Architecture Low Optimization	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O1**	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O1**	**0.006**

Scenario	Queries	Targets	FP Rate
Cross-Architecture Standard Optimization	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O2**	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O2**	**0.005**

Scenario	Queries	Targets	FP Rate
Cross-Architecture Heavy Optimization	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O3**	$(Compilers_{x86} \vee Compilers_{ARM})$ **−O3**	**0.004**