Precise and Accurate Patch Presence Test for Binaries

Usenix Security'18

Hang Zhang, Zhiyun Qian
University of California, Riverside

1

What’s the problem?

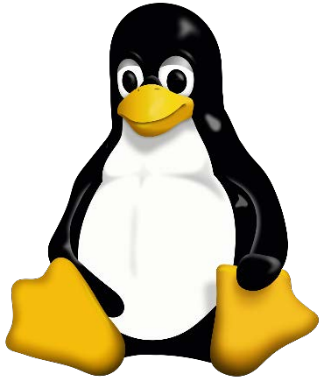
1

What’s the problem?

Short Answer: Given an Android image (or other binary), how do we decide whether a CVE has been patched?

A real-world example

A real-world example



A real-world example

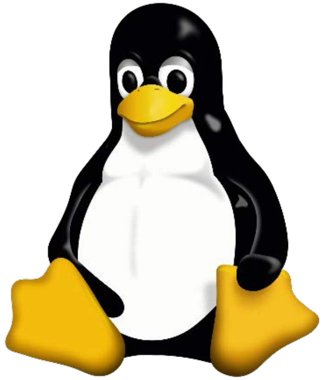
A real-world example

Open Source

A real-world example

Open Source

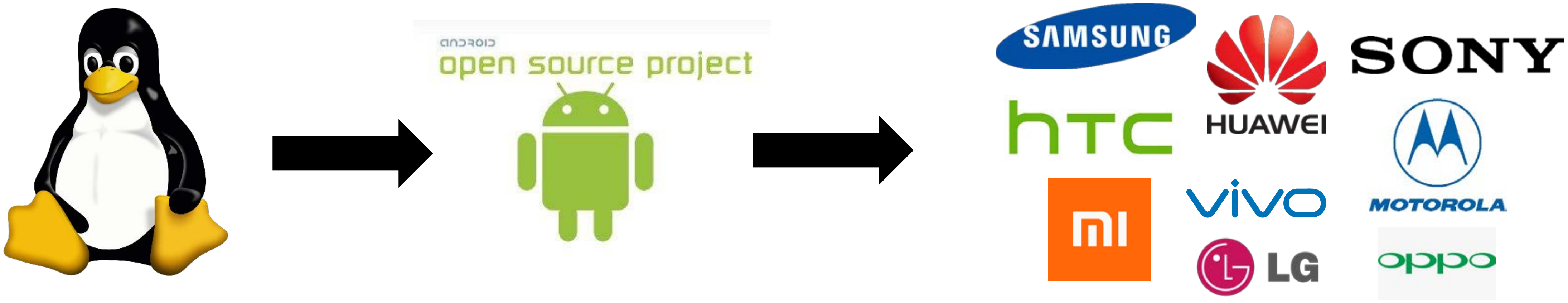
???



A real-world example

Open Source

???

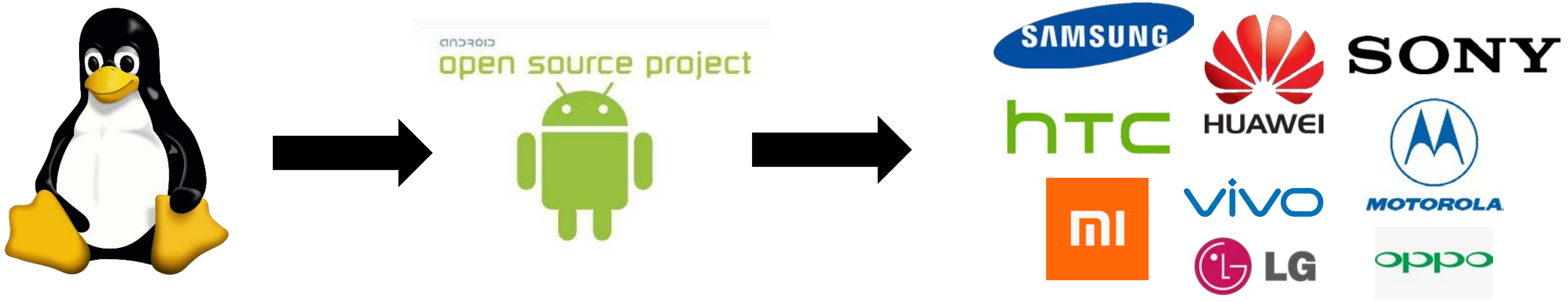


Few source “snapshots” w/o commit history.

A real-world example

Open Source

???



Few source “snapshots” w/o commit history.

- Are the mainstream linux/AOSP patches propagated?

Open vs. Closed

- Open-source is the trend.

- Code reuse in closed-source software.

Open vs. Closed

- Open-source is the trend.

- Code reuse in closed-source software.

- Is the open-source security patch applied in the binary?

Why challenging?

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```
-       if (a > 0)
+       if (a >= 0)
```

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```
-       if (a > 0)
+       if (a >= 0)
```

```
…
+       a = 0;
…
```

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```diff
- if (a > 0)
+ if (a >= 0)
```

```
...
+ a = 0;
...
```

- Patched function as a changing haystack.

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```
-        if (a > 0)
+        if (a >= 0)
```

```
…
+        a = 0;
…
```

- Patched function as a changing haystack.

```
Func():
.......
AAAAAA
+ the line
AAAAAA
.......
```

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```
-     if (a > 0)
+     if (a >= 0)
```

```
...
+     a = 0;
...
```

- Patched function as a changing haystack.

```
Func():
......
BBBBBB
+ the line
BBBBBB
......
```

backport

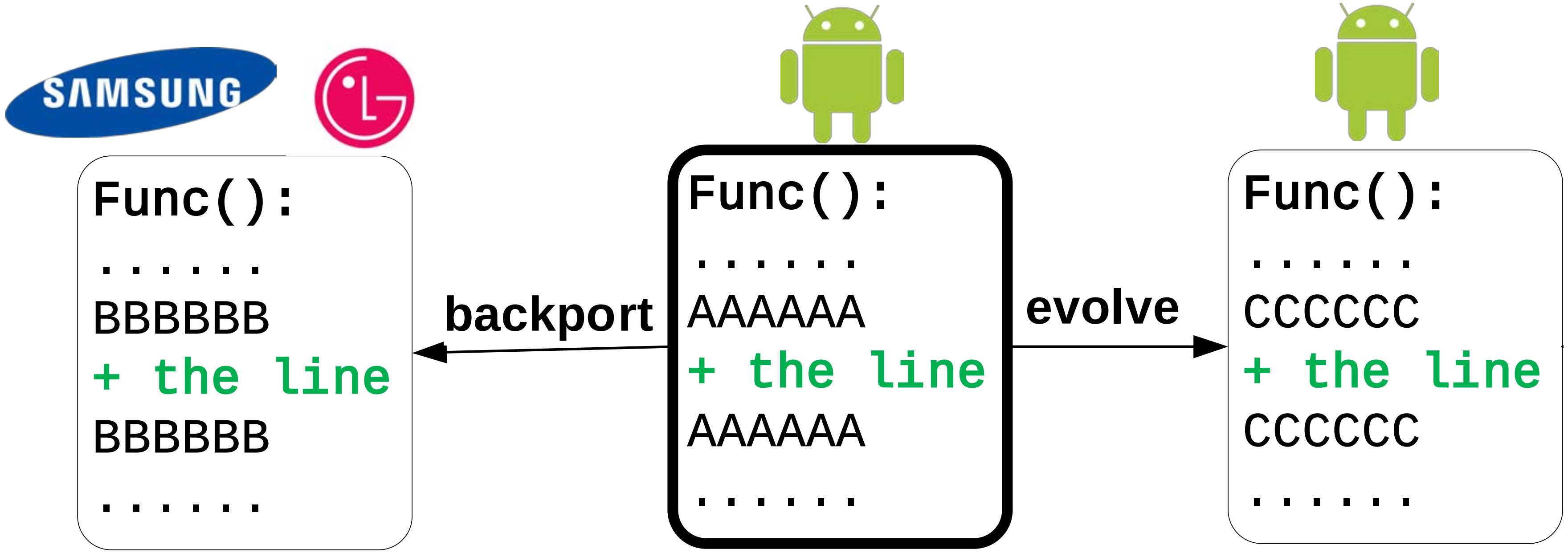
```
Func():
......
AAAAAA
+ the line
AAAAAA
......
```

#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

```
-       if (a > 0)                    ...
+       if (a >= 0)                   +       a = 0;
                                      ...
```

- Patched function as a changing haystack.

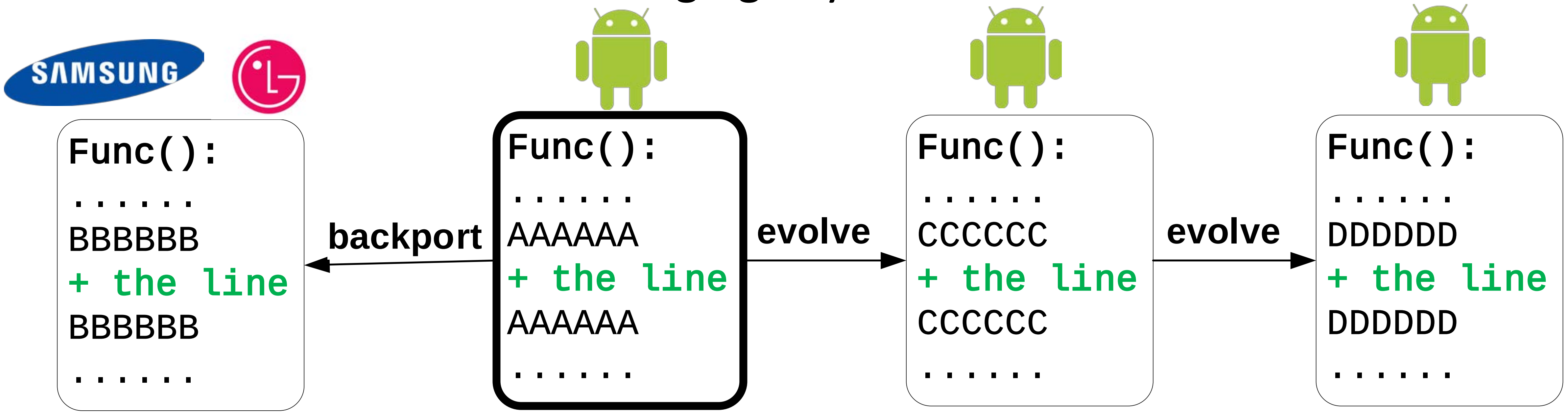


#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

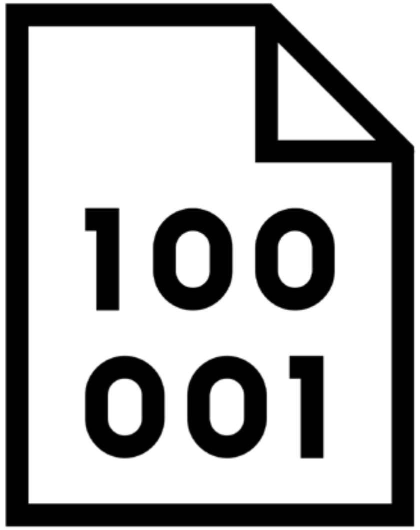
```
-    if (a > 0)          ...
+    if (a >= 0)         +    a = 0;
                         ...
```

- Patched function as a changing haystack.



#2: Haystack is a binary…

- Find the needle in a binary.



Related work

Related work

Category 1: Source-source matching.

Related work

Category 1: Source-source matching.

Cannot deal with binary haystack.

Related work

Category 1: Source-source matching.

Cannot deal with binary haystack.

Category 2: Binary-binary matching.

Related work

Category 1: Source-source matching. ✗

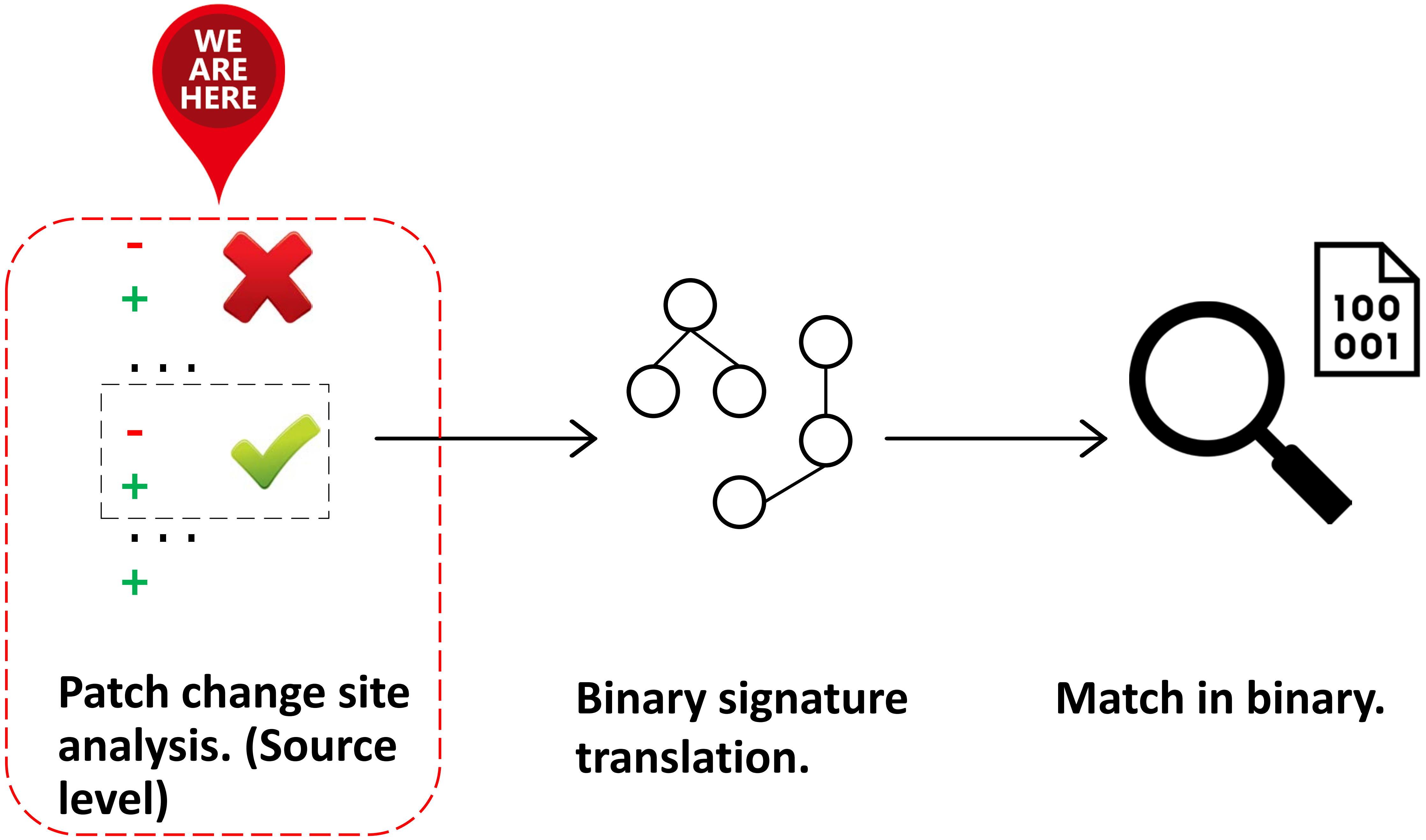
Cannot deal with binary haystack.

Category 2: Binary-binary matching. ✗

Lack of knowledge about the needle (i.e. the patch).

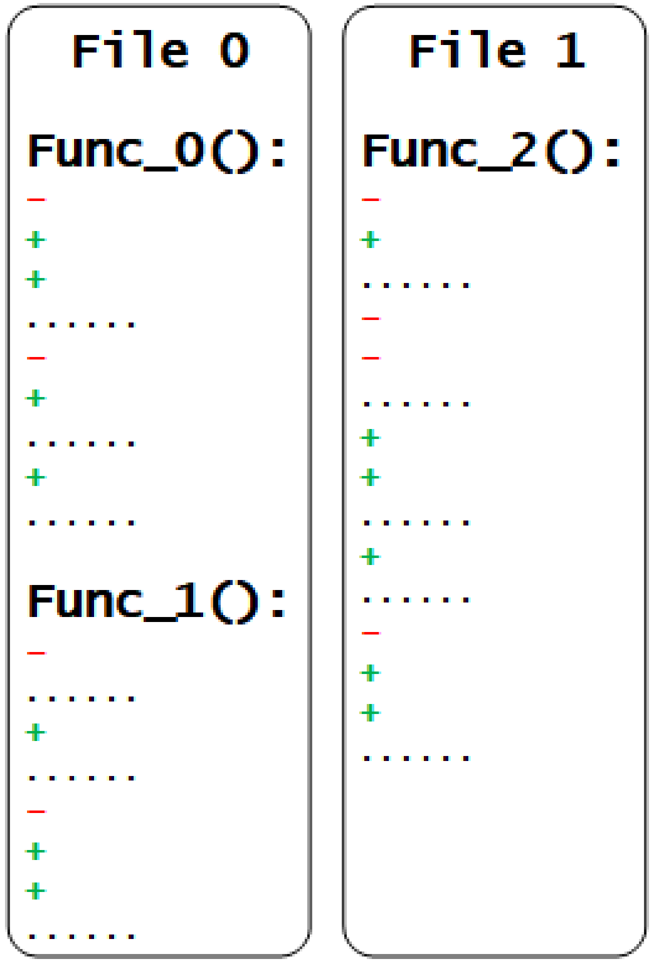
How does FIBER work?

How does a human expert work?


WE ARE HERE
Patch change site analysis. (Source level)
Binary signature translation.
Match in binary.
100
001

Change Site Analysis: What will human do?

- Given an open-source security patch, you need to locate it in a binary.
- What will you do at first?



Change Site Analysis: What will human do?

- Given an open-source security patch, you need to locate it in a binary.
- What will you do at first?

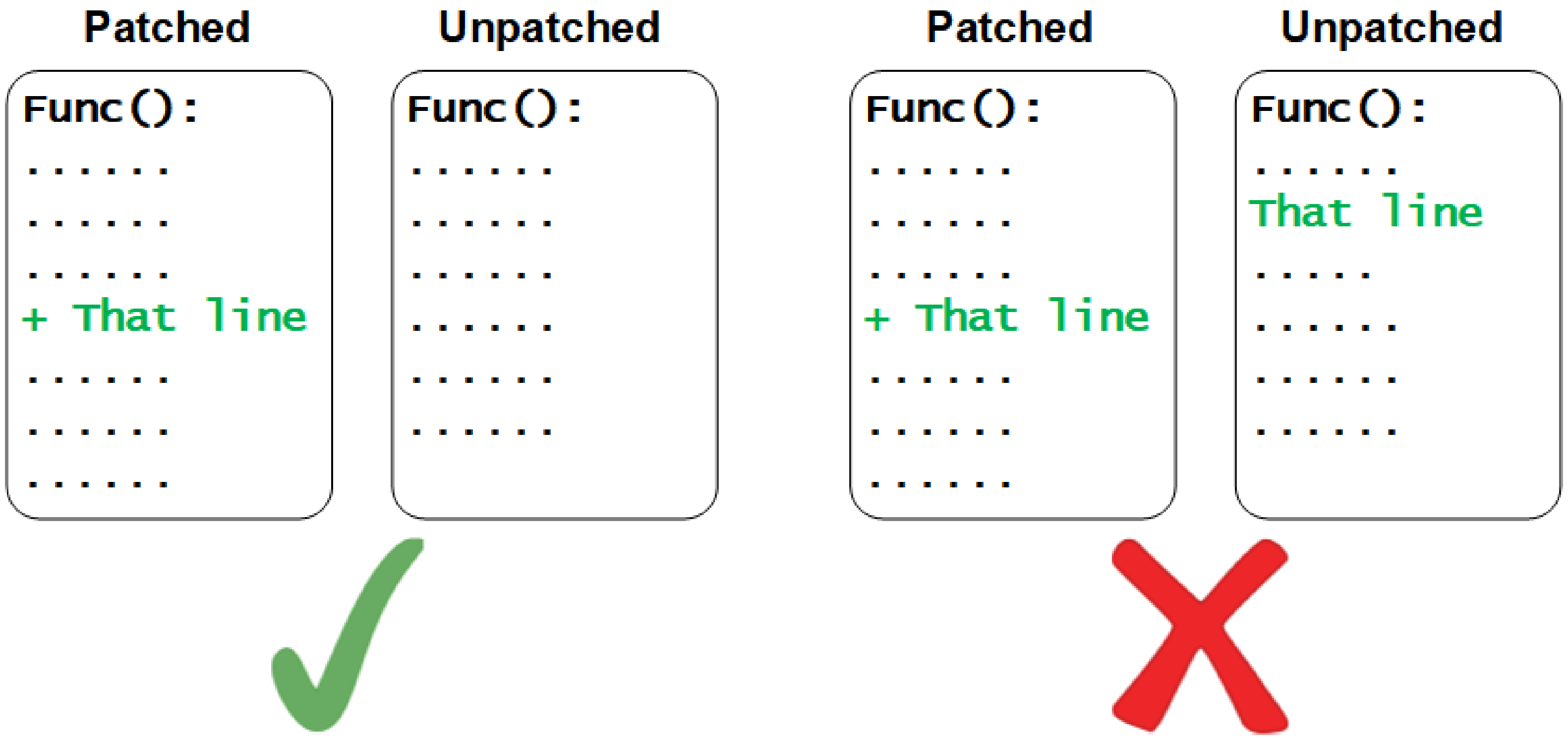
Pick those most obvious, unique and representative change sites.

```
File 0

Func_0():
-
+
+
......
-
+
......
+
......

Func_1():
-
......
+
......
-
+
+
......
```

```
File 1

Func_2():
-
+
......
-
-
......
+
+
......
+
......
-
+
+
......
```

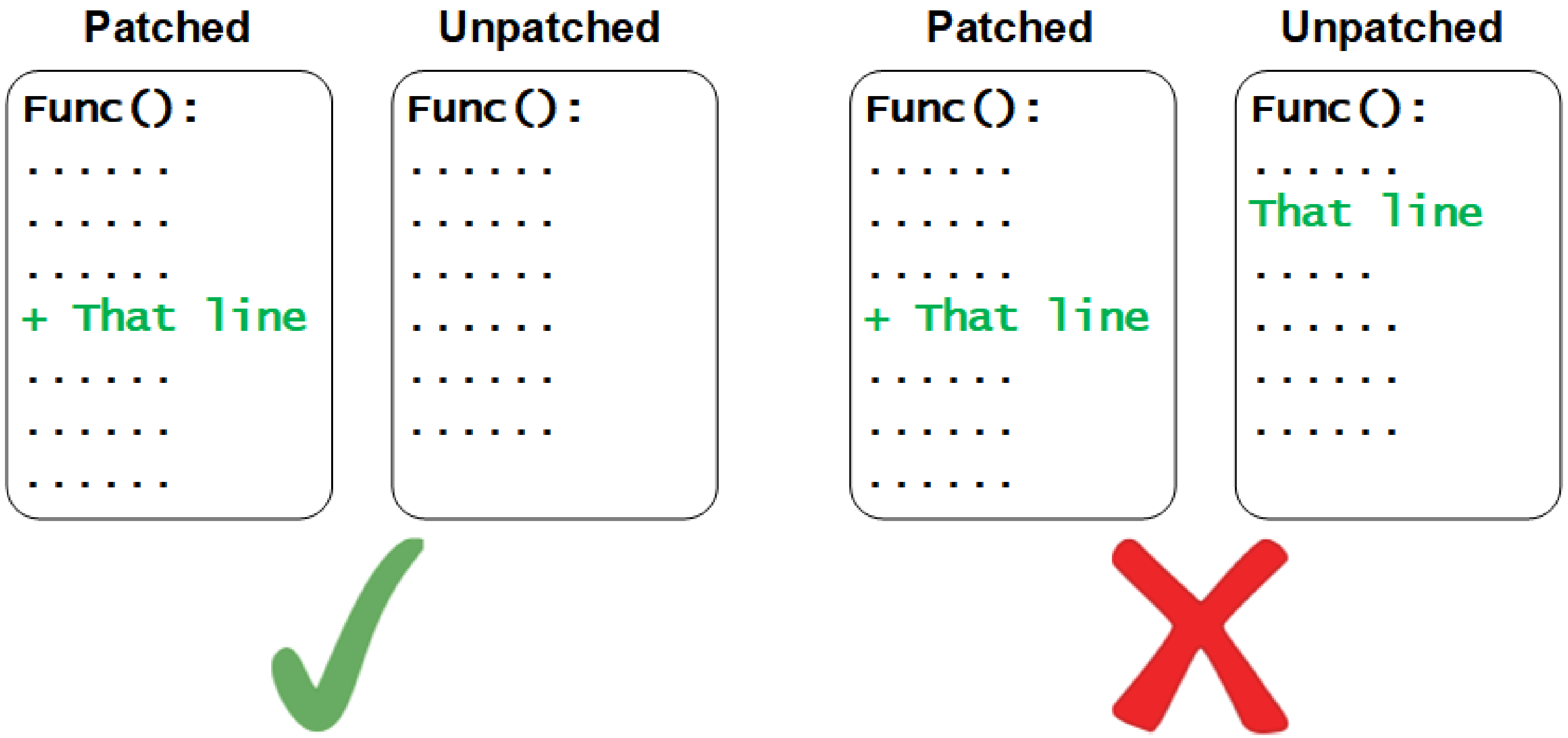
Change Site Analysis

- **Unique** – Exists only in the patched version.



Change Site Analysis

- **Unique** – Exists only in the patched version.



Solution: token-based string search to test uniqueness, add contexts if not unique.

Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.

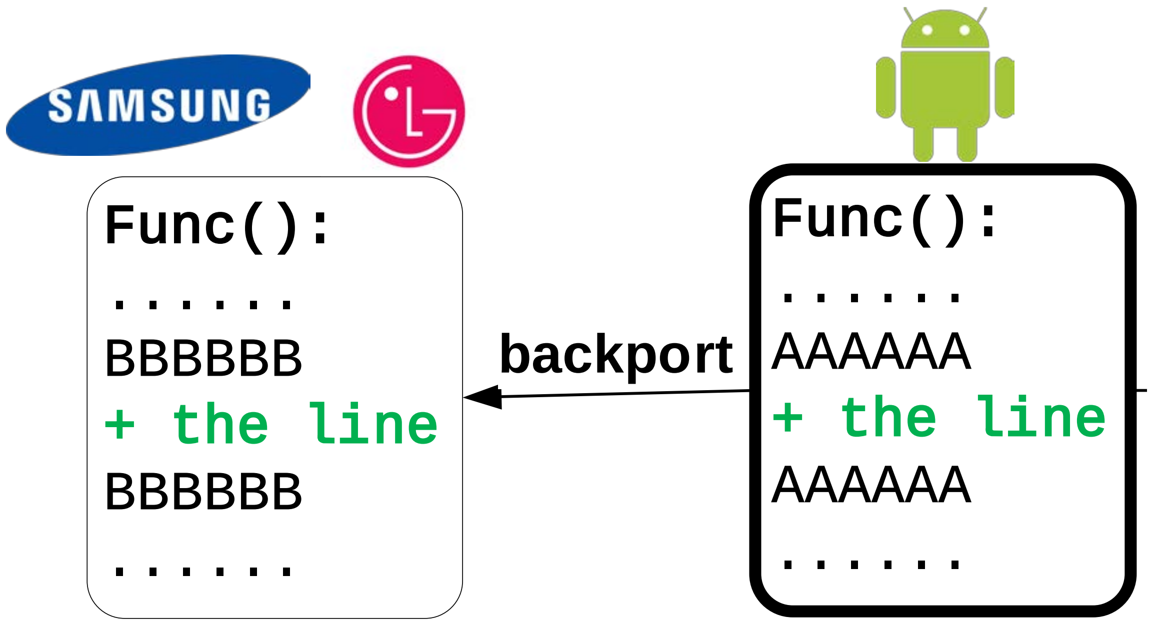
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.

```
Func():
.......
AAAAAA
+ the line
AAAAAA
.......
```

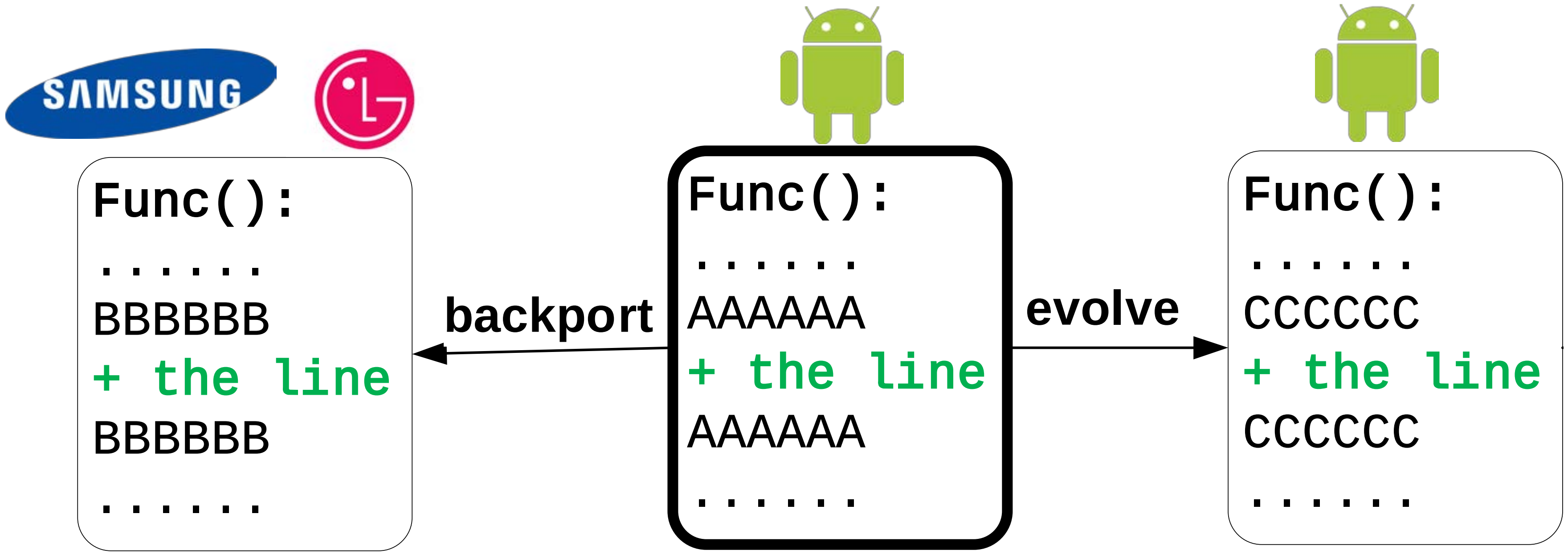
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.



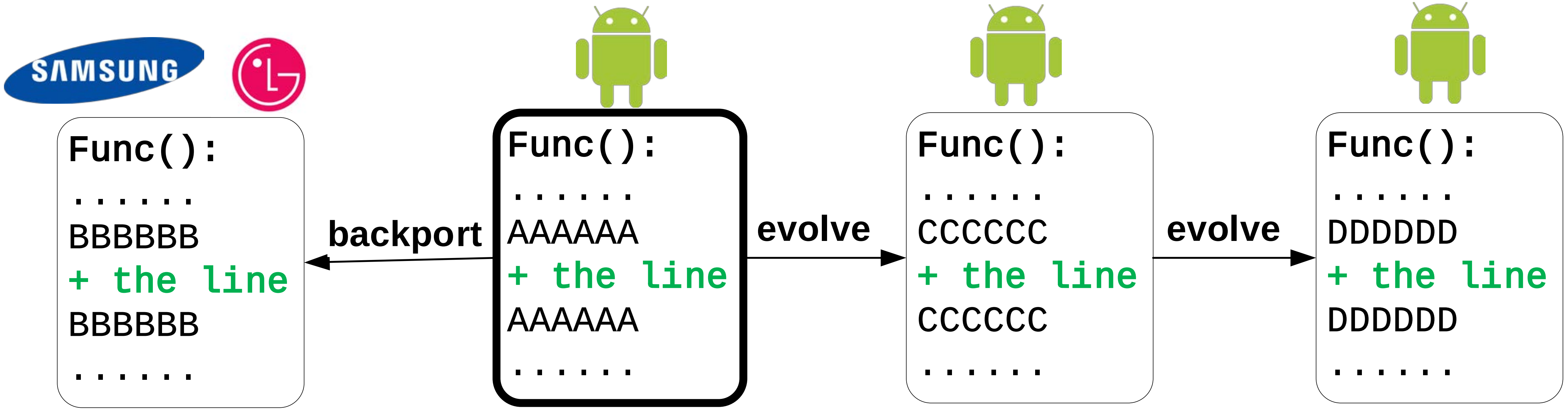
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.



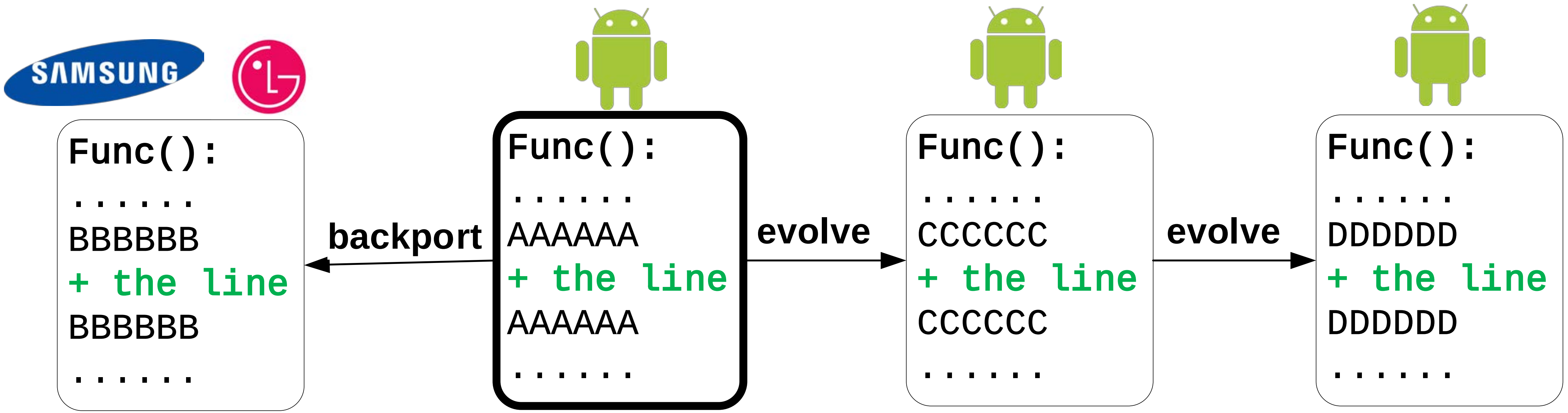
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.



Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.



Solution: keep the change site as small as possible (always start from a single line), add contexts only when necessary.

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

```
+ if(cond)
```

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

```
+ if(cond)
```

Good: both syntax structure and semantic change.

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

```
+ if(cond)
```

Good: both syntax structure and semantic change.

```
+ a = b * c
```

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

```
+ if(cond)
```

Good: both syntax structure and semantic change.

```
+ a = b * c
```

Meh: only semantic change without syntax change.

Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

```
+ func_noinline()
```

Perfect: easily located by call instruction and function name (Android images have symbol table).

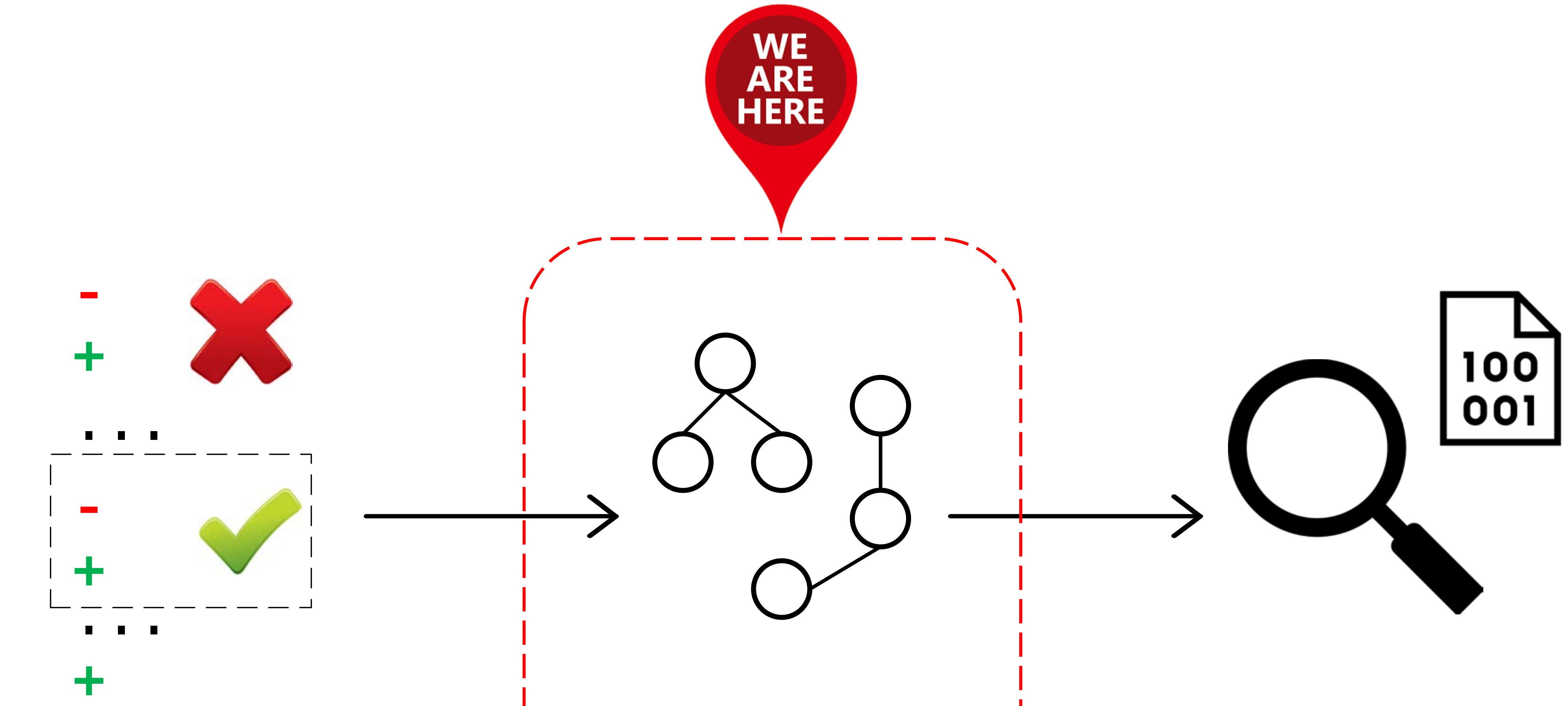
```
+ if(cond)
```

Good: both syntax structure and semantic change.

```
+ a = b * c
```

Meh: only semantic change without syntax change.

Solution: we rank the change sites based on statement types involved, according to our domain knowledge.



Patch change site analysis. (Source level)

Binary signature translation.

Match in binary.

What if we do it manually?

- How to connect the source change with binary code?

What if we do it manually?

- How to connect the source change with binary code?

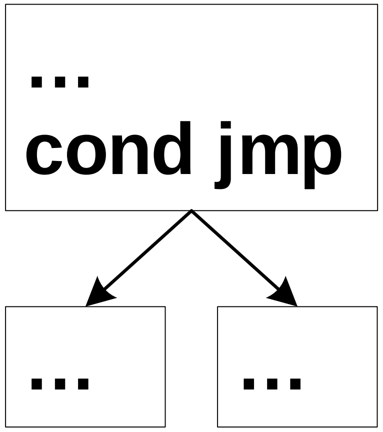
```
if (a > 1)
    do A;
else
    do B;
```

What if we do it manually?

- How to connect the source change with binary code?

```
if (a > 1)
    do A;
else
    do B;
```

Syntax

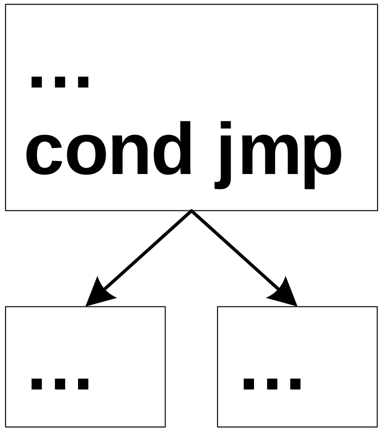


What if we do it manually?

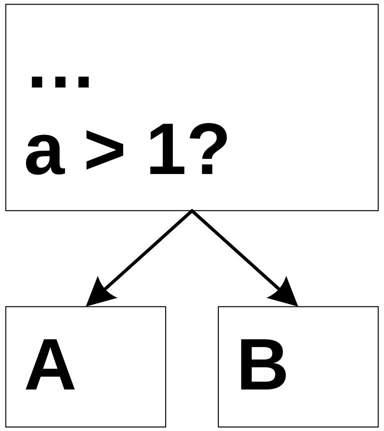
- How to connect the source change with binary code?

```
if (a > 1)
    do A;
else
    do B;
```

Syntax



Semantics

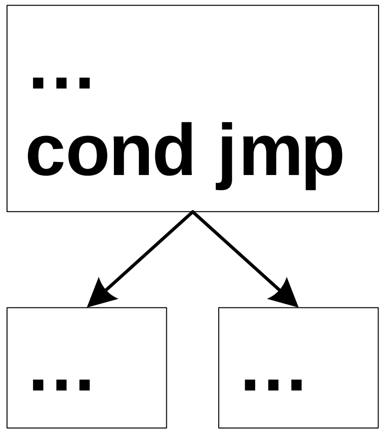


What if we do it manually?

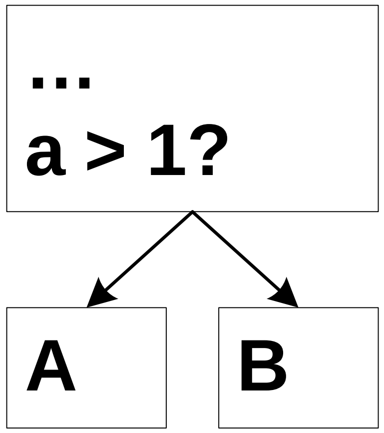
- How to connect the source change with binary code?

```
if (a > 1)
    do A;
else
    do B;
```

Syntax



Semantics



Correlate both its syntax and semantics to the binary code.

Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.

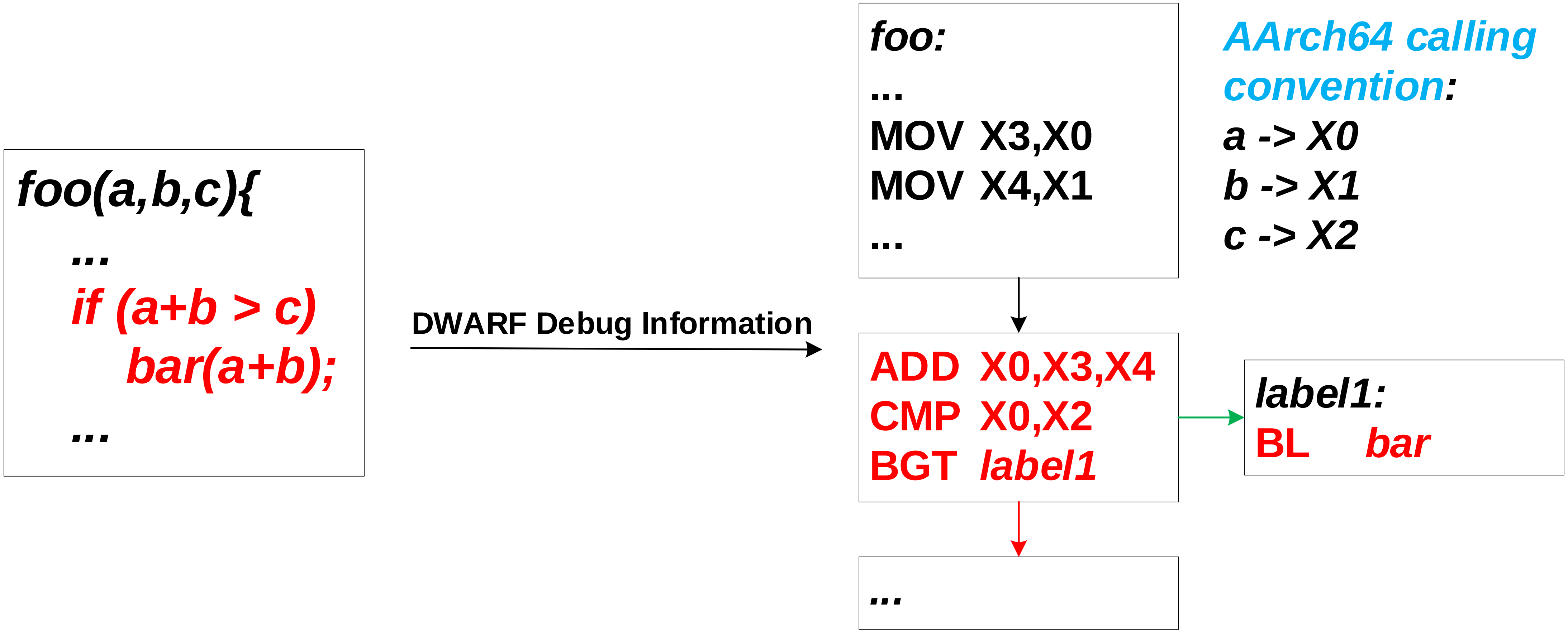
Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.

```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```

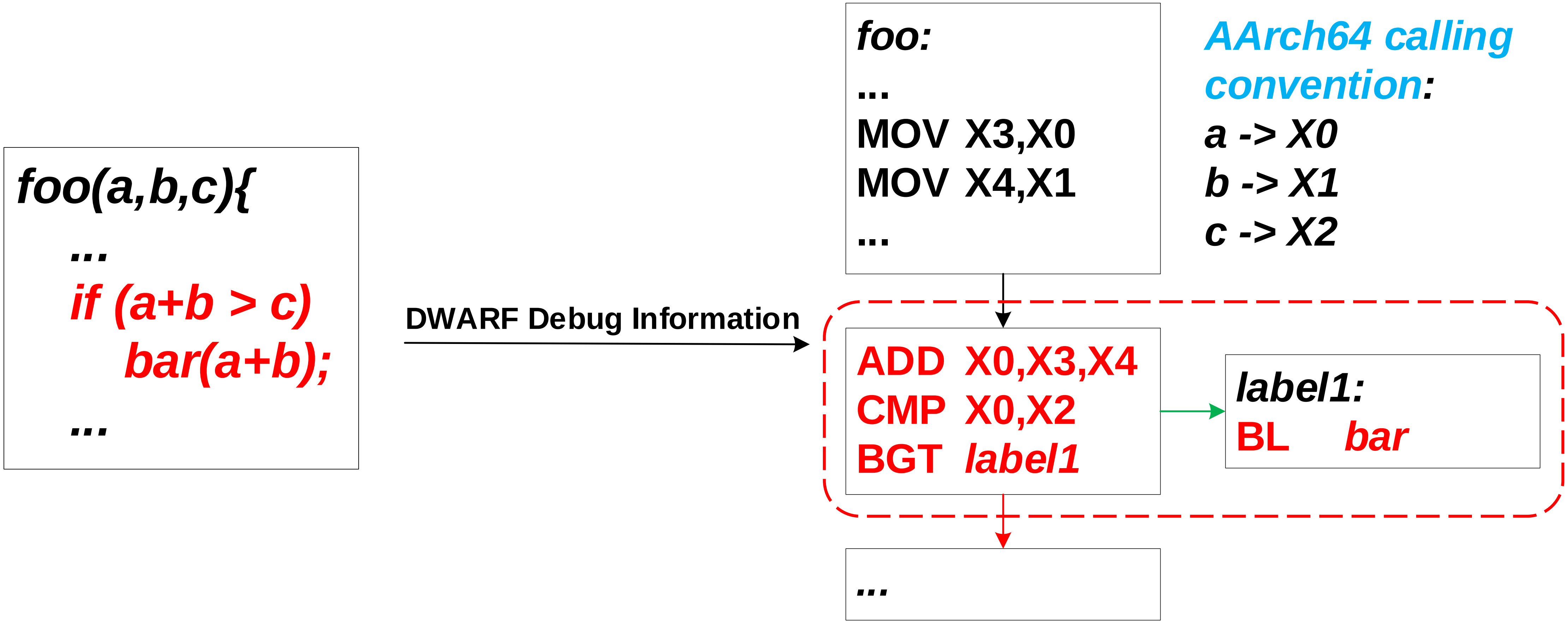
Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.



Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.



Binary Signature Translation

- Find the “root” instructions.

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL     bar
```

Binary Signature Translation

- Find the "root" instructions.

X0 = X3 + X4

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL    bar
```

Binary Signature Translation

- Find the "root" instructions.

X0 = X3 + X4

(X3 + X4) > X2

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL    bar
```

Binary Signature Translation

- Find the “root” instructions.

$X0 = X3 + X4$

$(X3 + X4) > X2$

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL     bar
```

Binary Signature Translation

- Find the “root” instructions.

$X0 = X3 + X4$

$(X3 + X4) > X2$

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL     bar
```

Binary Signature Translation

- Find the “root” instructions.

X0 = X3 + X4

(X3 + X4) > X2

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

```
label1:
BL    bar
```

Root instructions: whose outputs will no longer be consumed by other instructions.

Binary Signature Translation

- Find the “root” instructions.

$X0 = X3 + X4$

$(X3 + X4) > X2$

```
ADD  X0,X3,X4
CMP  X0,X2
BGT  label1
```

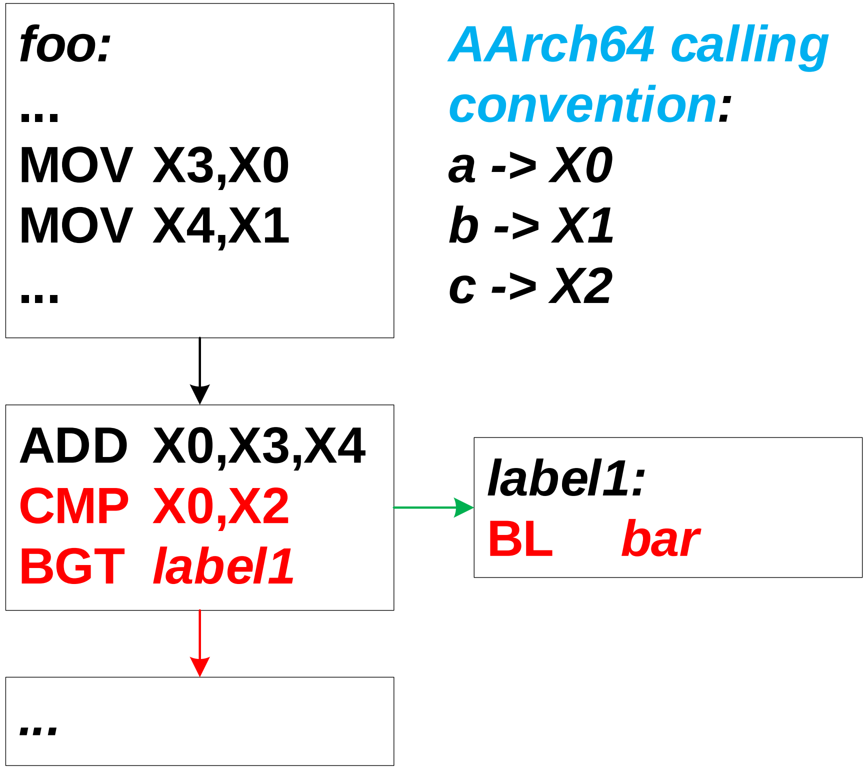
```
label1:
BL    bar
```

Root instructions: whose outputs will no longer be consumed by other instructions.

Solution: we perform a basic-block level data-flow analysis to identify root instructions.

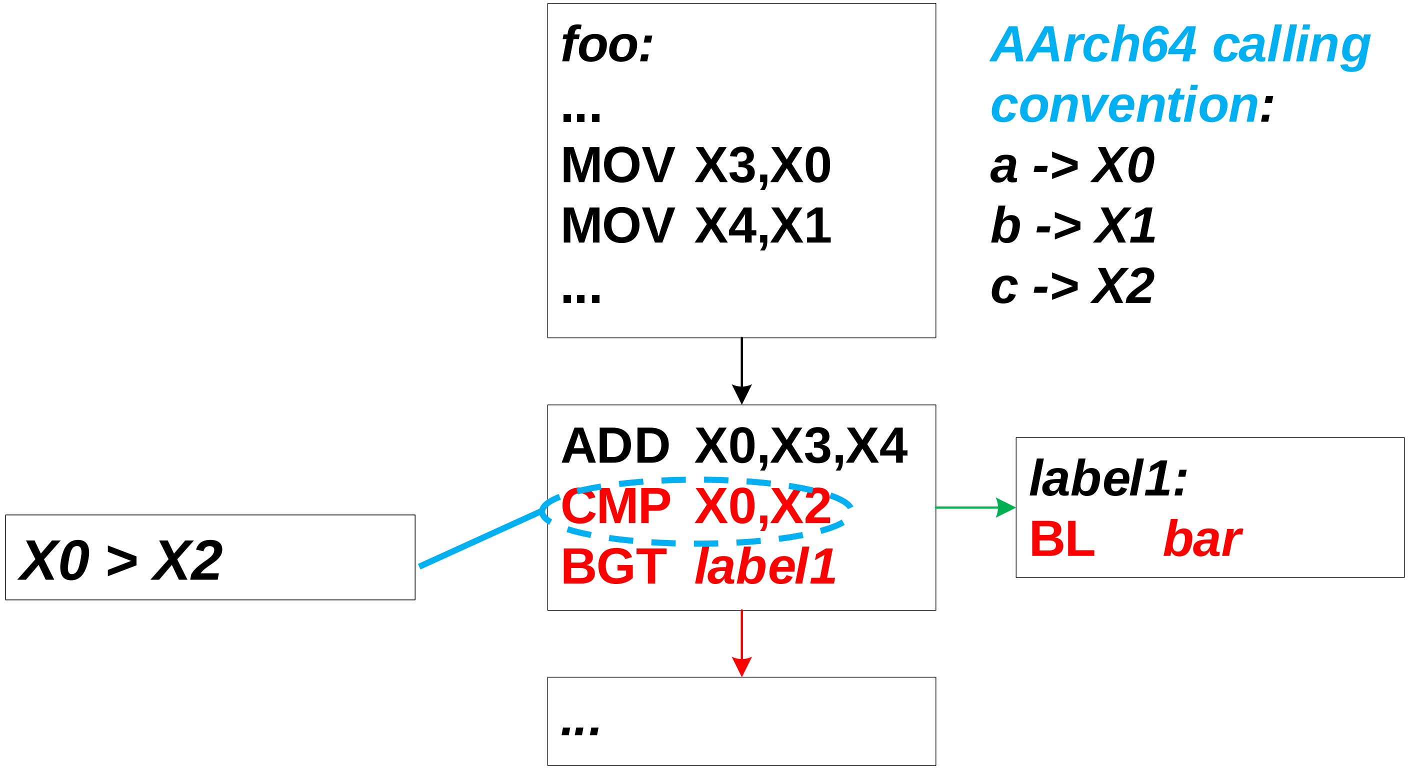
Binary Signature Translation

- Extract semantic formulas for root instructions.



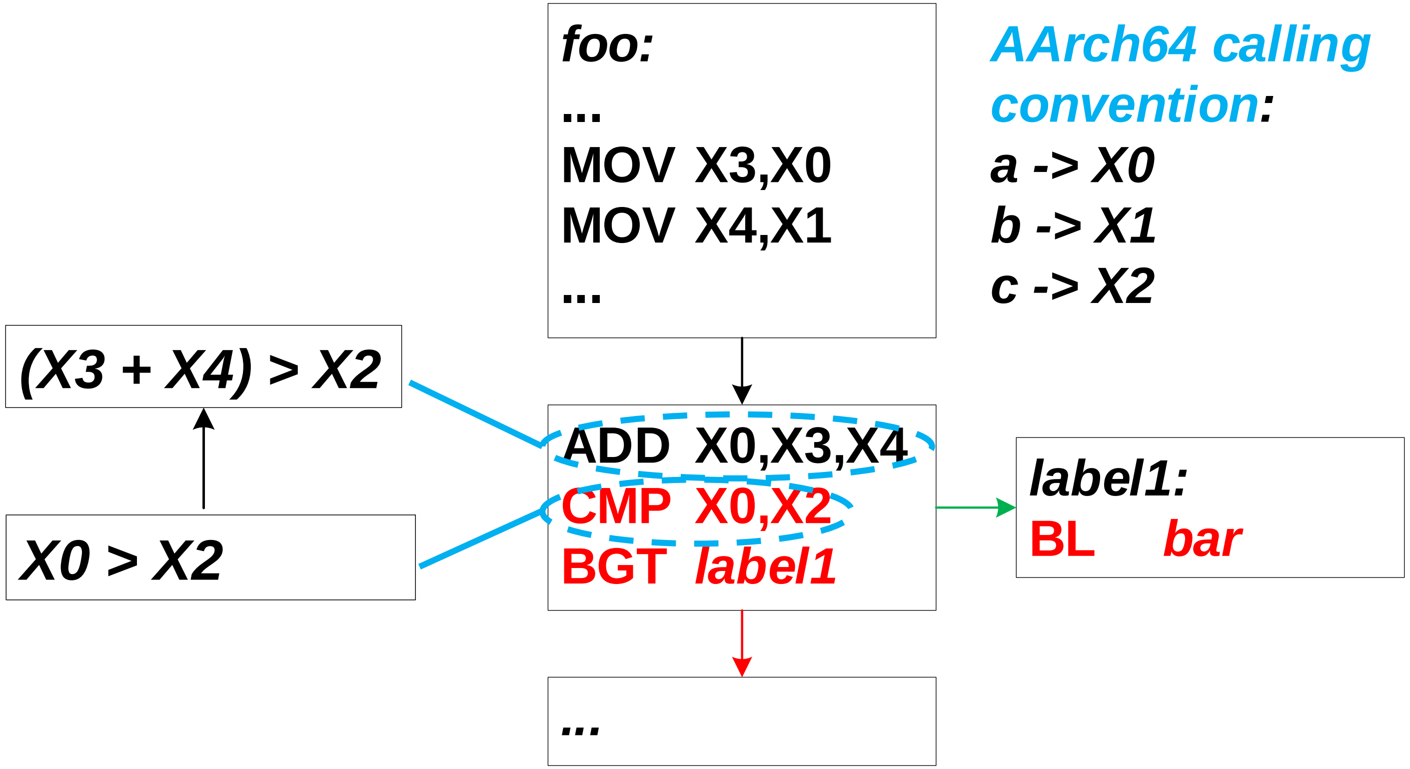
Binary Signature Translation

- Extract semantic formulas for root instructions.



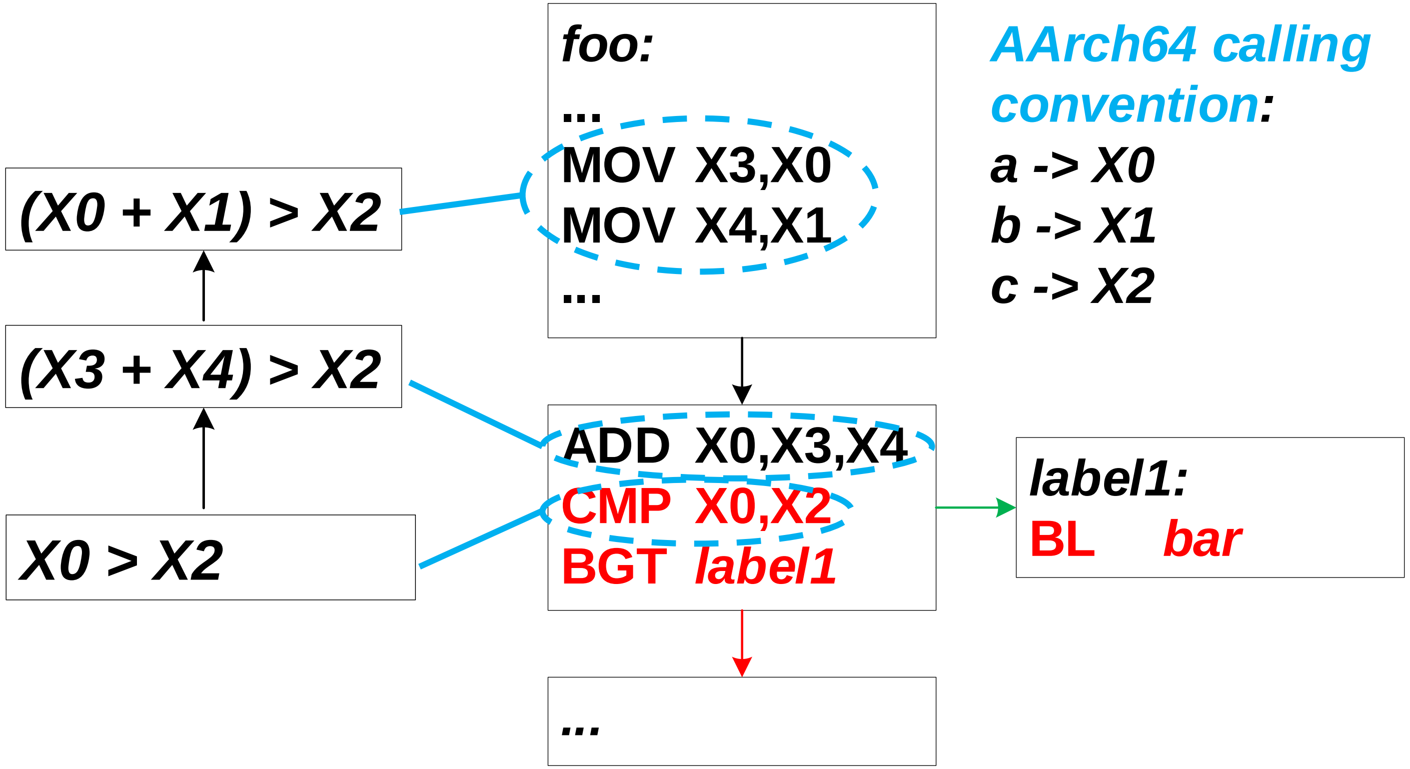
Binary Signature Translation

- Extract semantic formulas for root instructions.



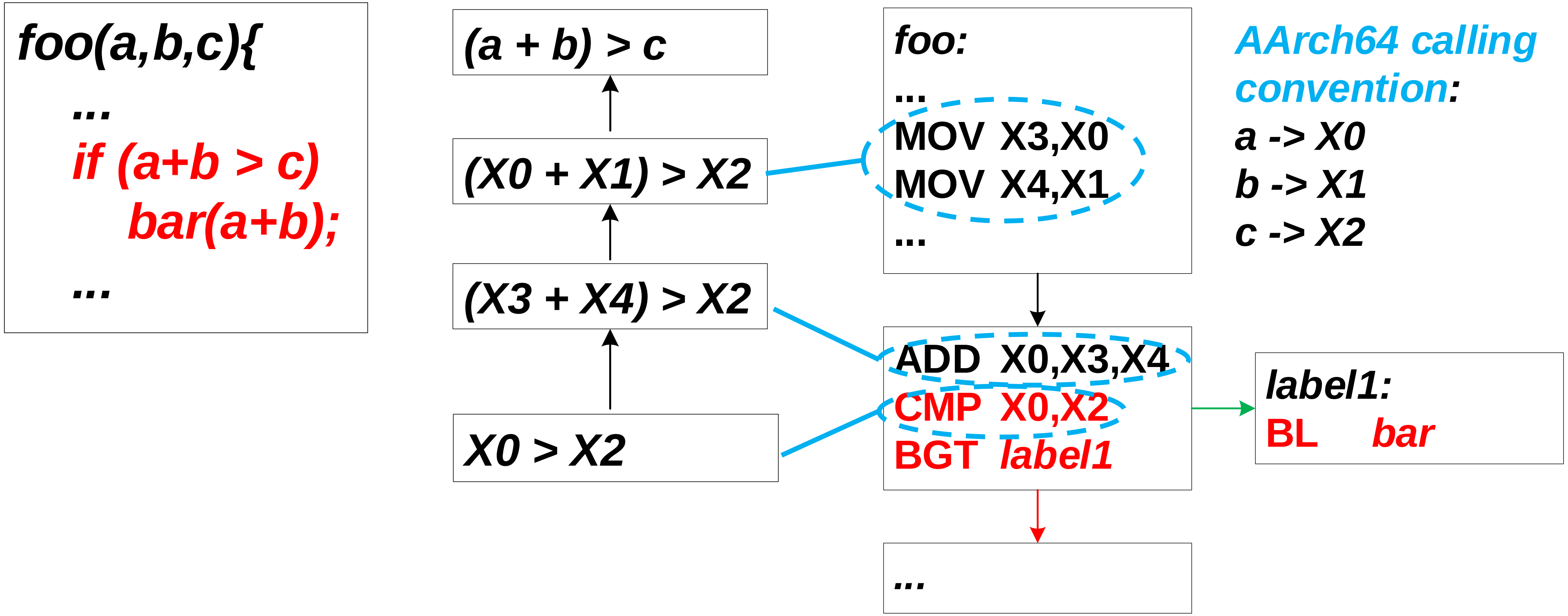
Binary Signature Translation

- Extract semantic formulas for root instructions.



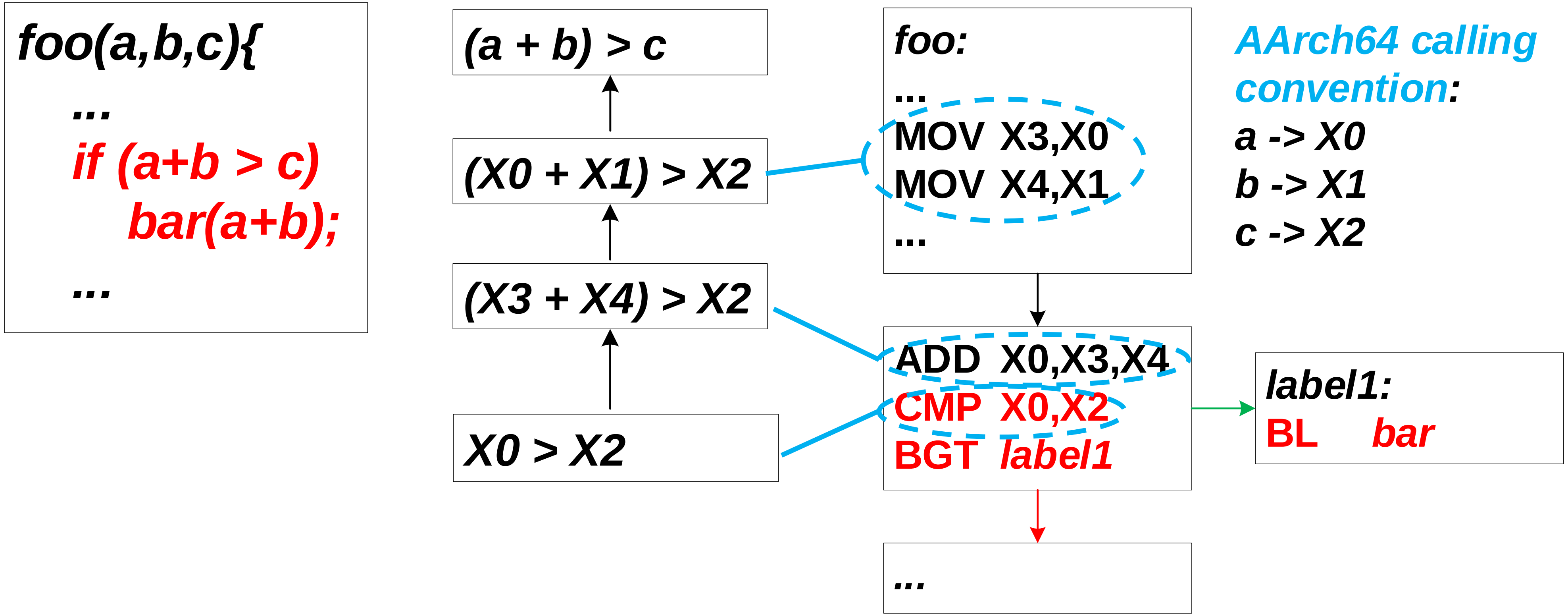
Binary Signature Translation

- Extract semantic formulas for root instructions.

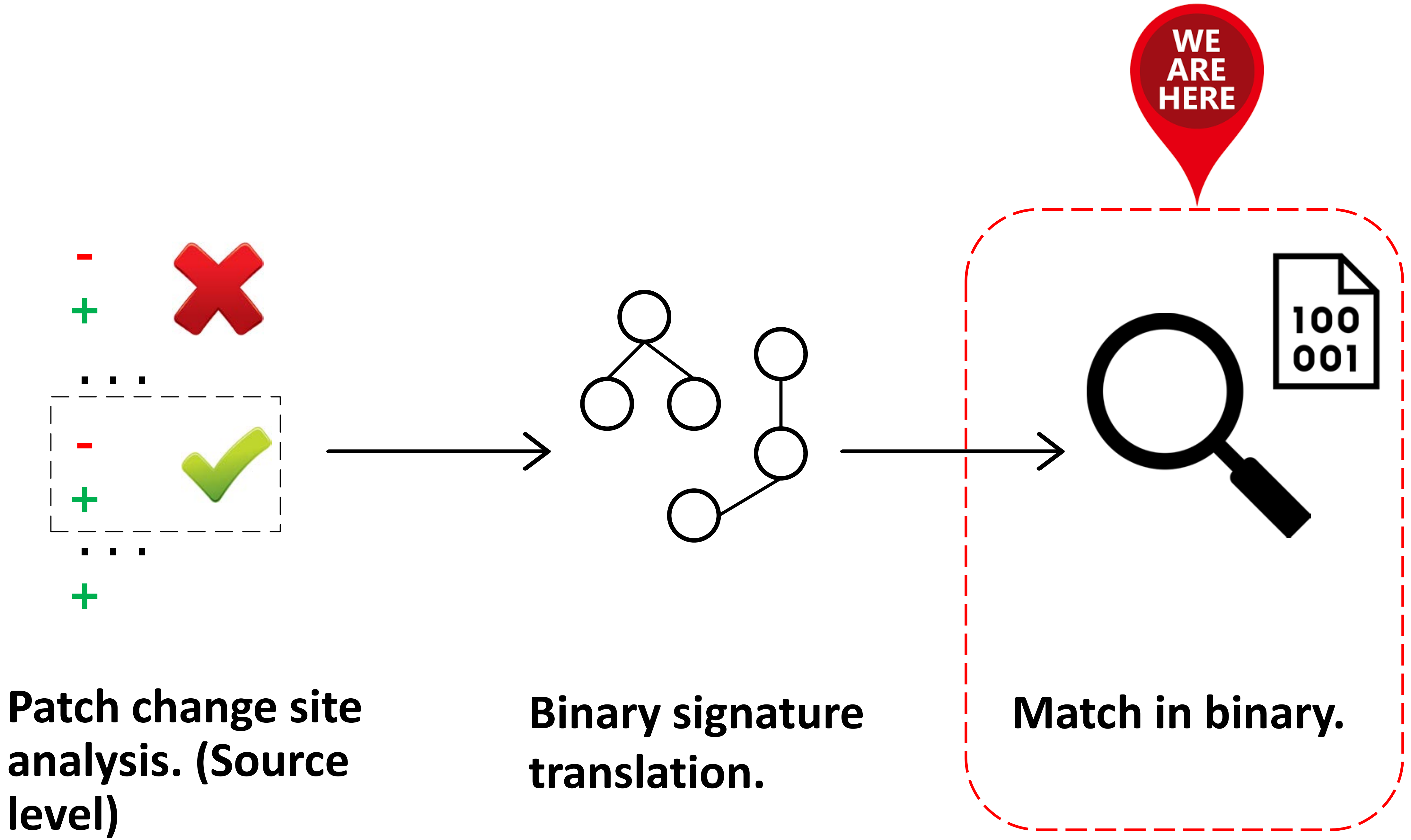


Binary Signature Translation

- Extract semantic formulas for root instructions.



Solution: we use function-level, intra-procedure and under-constrained symbolic execution to obtain formulas.


WE ARE HERE
-
+
...
-
+
...
+
100
001
Patch change site analysis. (Source level)
Binary signature translation.
Match in binary.

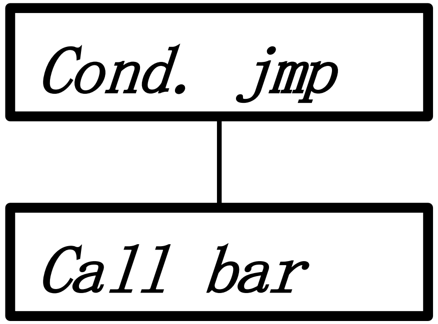
Matching

- Quick Pass.

Matching

- Quick Pass.

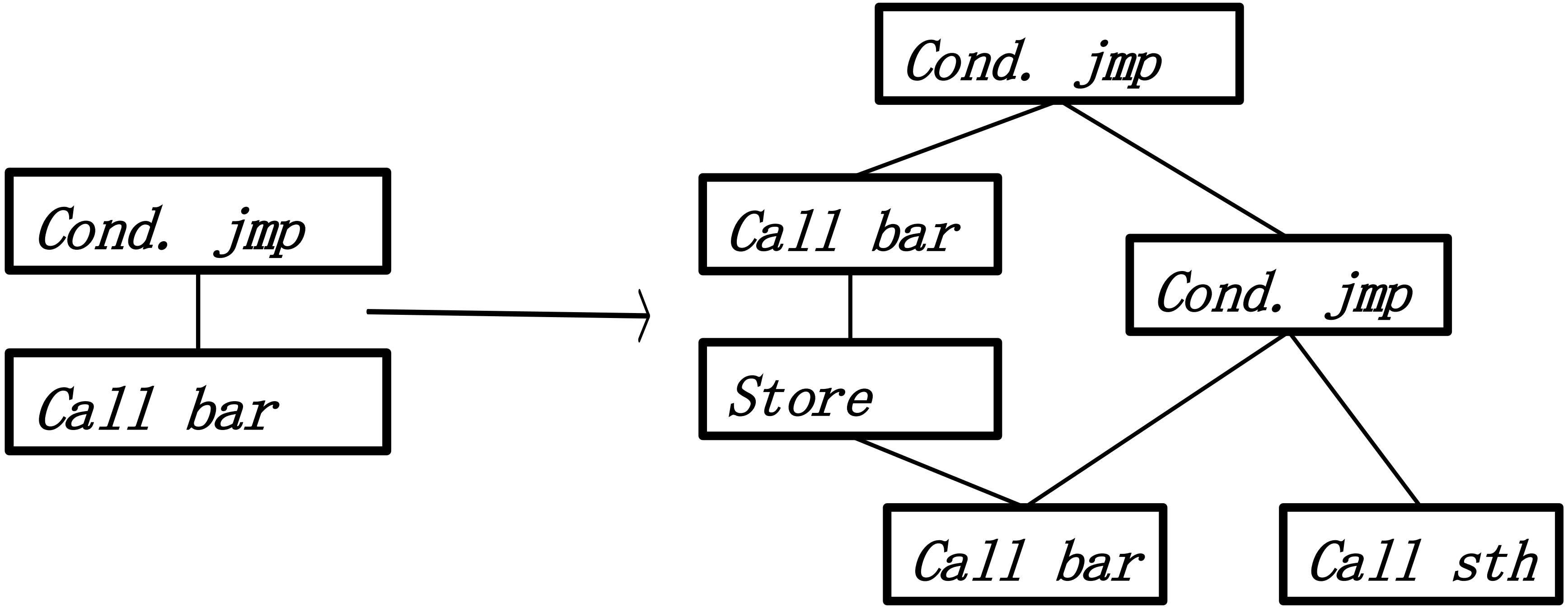
```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```



Matching

- Quick Pass.

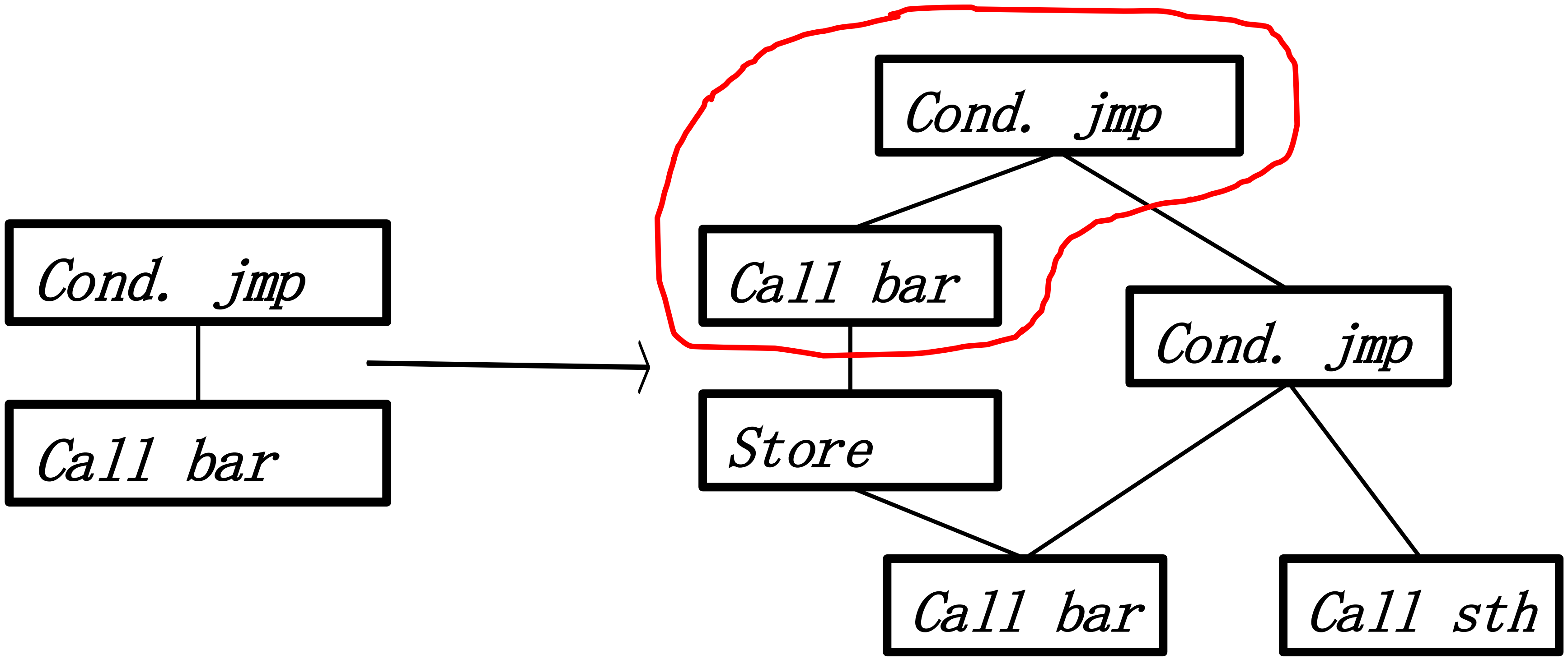
```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```



Matching

- Quick Pass.

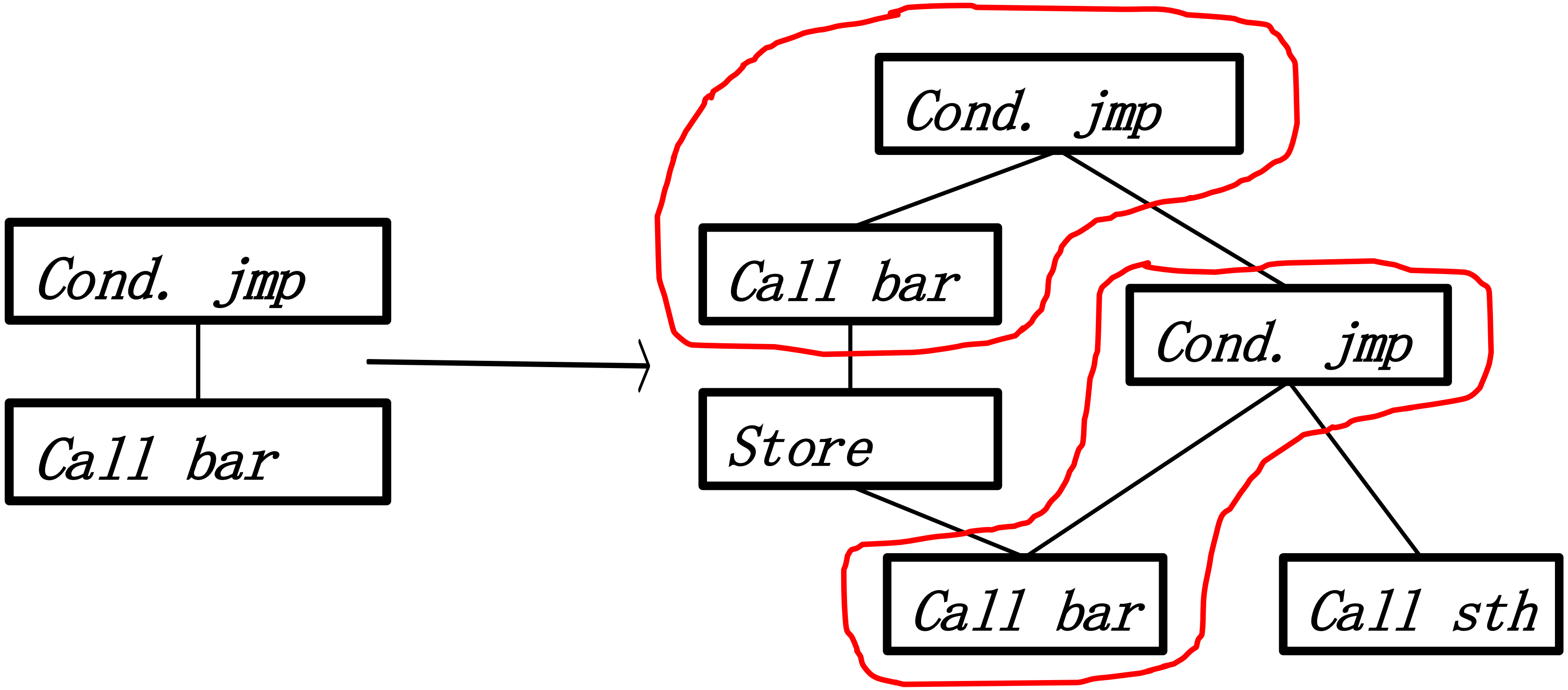
foo(a,b,c){
...
if (a+b > c)
bar(a+b);
...



Matching

- Quick Pass.

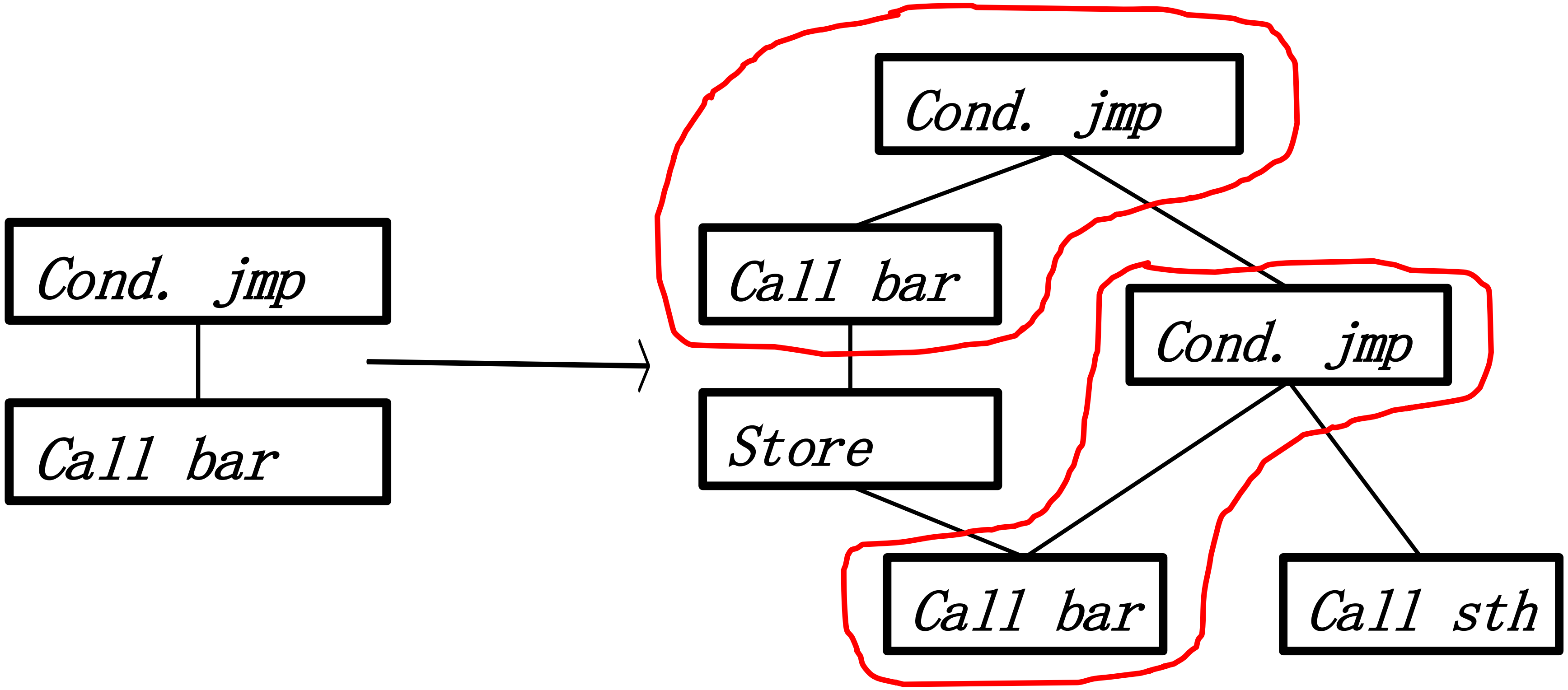
```
foo(a,b,c){
   ...
   if (a+b > c)
      bar(a+b);
   ...
```



Matching

- Quick Pass.

```
foo(a,b,c){
   ...
   if (a+b > c)
      bar(a+b);
   ...
```

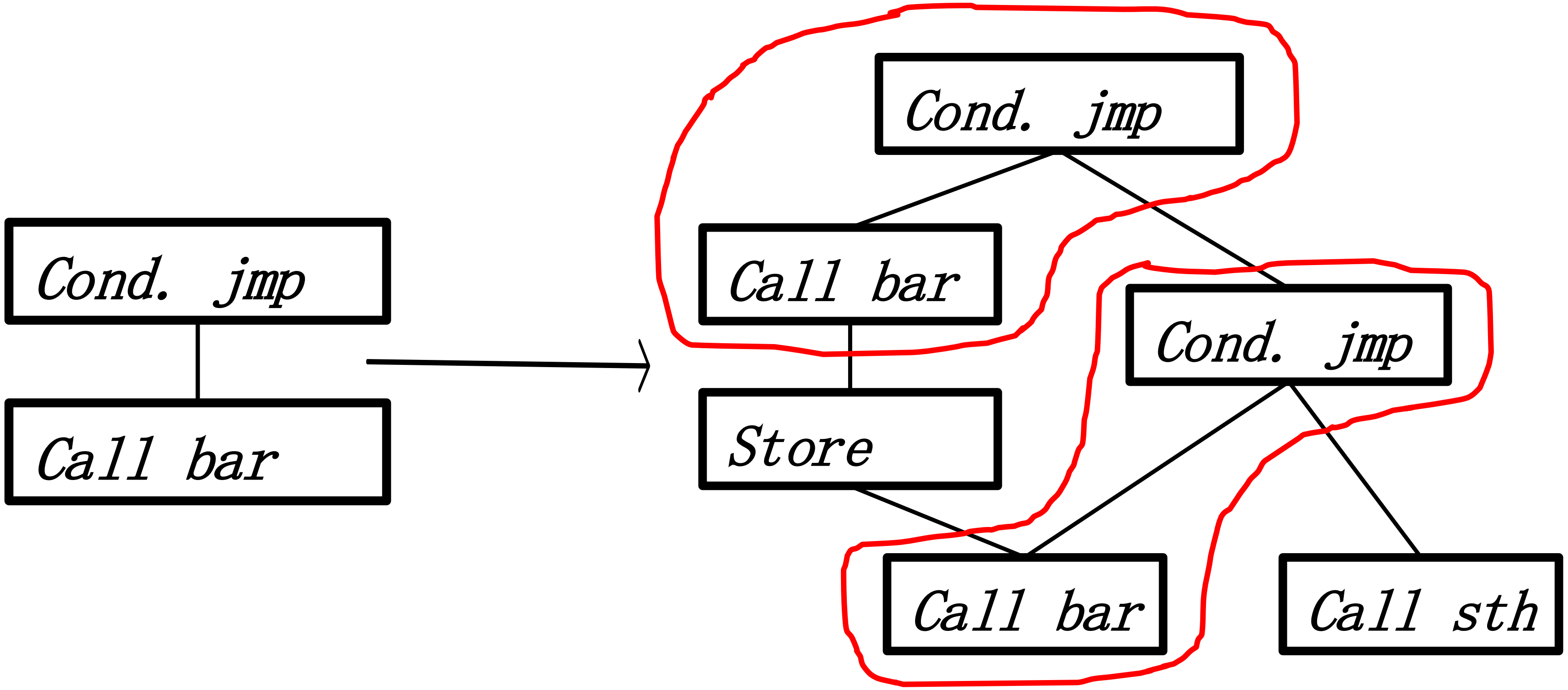


Solution: look at easy-to-match attributes, e.g. topology, root instruction type, etc.

Matching

- Slow Pass.

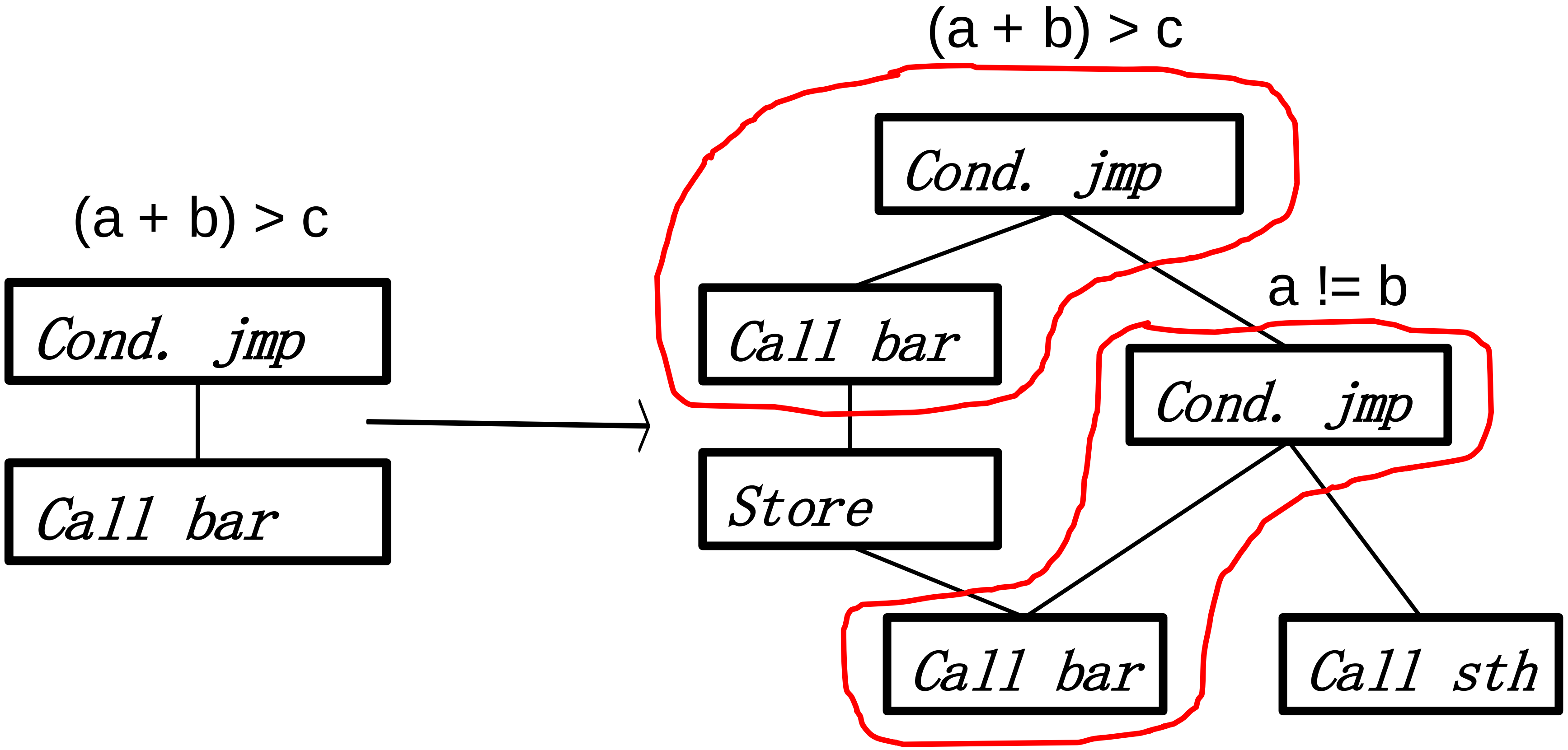
```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```



Matching

- Slow Pass.

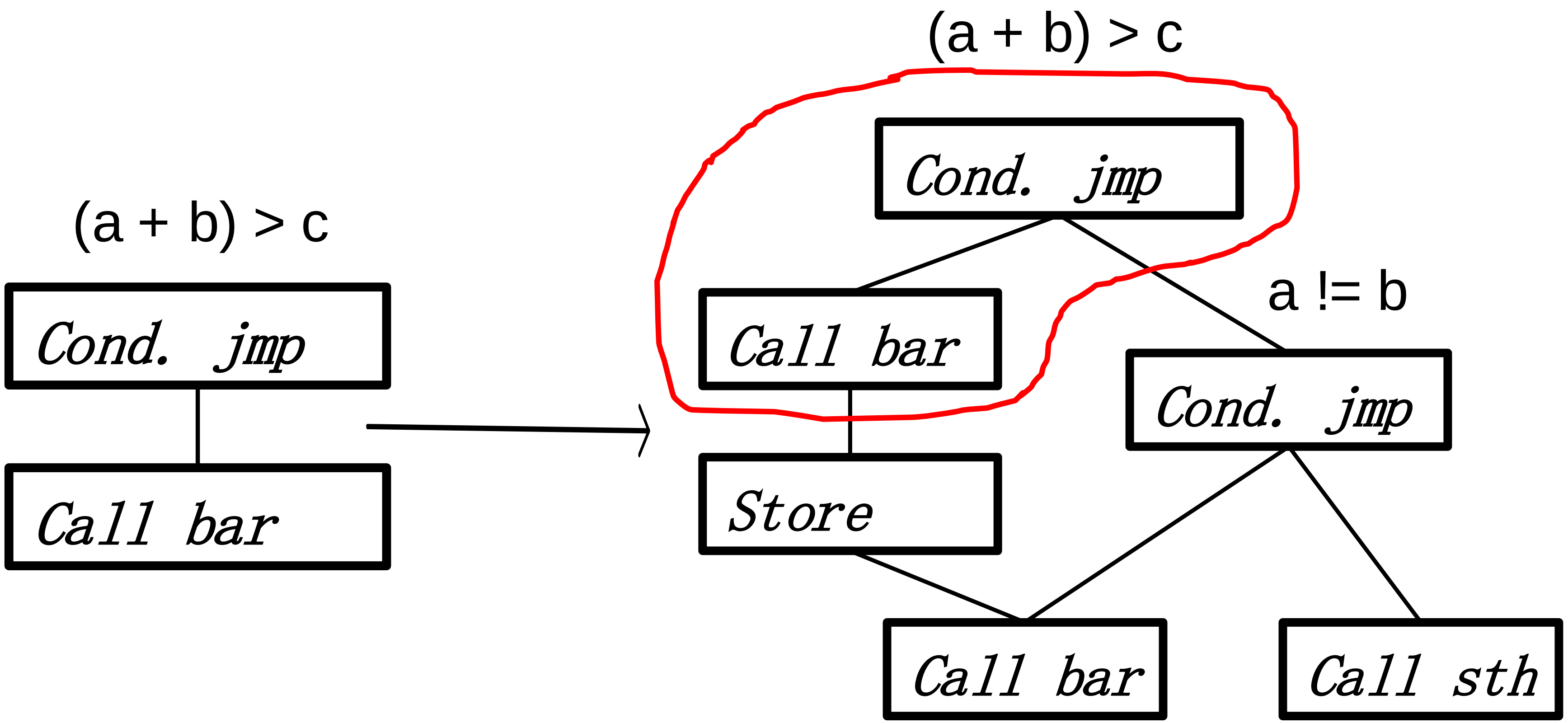
```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```



Matching

- Slow Pass.

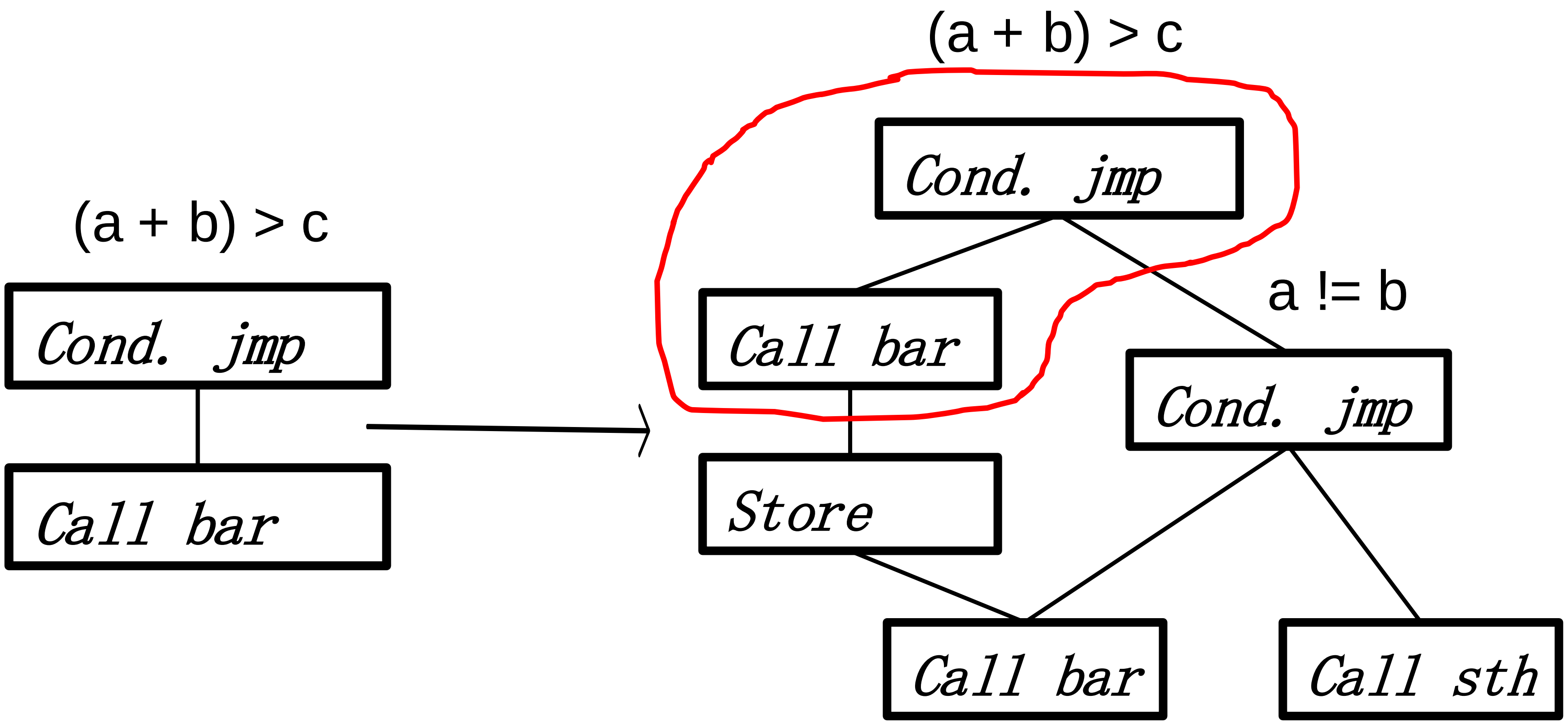
```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
```



Matching

- Slow Pass.

foo(a,b,c){
...
if (a+b > c)
bar(a+b);
...



Solution: basically we strictly compare two formulas simplified by Z3 solver , with necessary relaxations. (e.g. commutative operators)

Special (and Interesting) Cases

```
Func():
.......
.......
+ uniq_func_noinline()
.......
.......
```

```
Func():
.......
.......
+ uniq_func_noinline()
.......
.......
```

Simply test the function call presence, no semantic formulas needed.

```
Func():
.......
.......
- f(a,b)
+ f(a,c)
.......
.......
```

```
Func():
......
......
- f(a,b)
+ f(a,c)
......
......
```

That ***line*** matters? No, that ***parameter*** matters!

How well does FIBER work?

In evaluation:

107 security patches crawled from Android Bulletin (*Jun 2016 – May 2017*)

8 Android kernel images from **3** mainstream vendors.

Accuracy

Device	No.	Patch Cnt*	Build Date (mm/dd/yy)	Kernel Version	Accuracy				Online Matching Time (s)			
					TP	TN	FP	FN	Total	Avg	~70%	Max.
Samsung S7	0	102	06/24/16	3.18.20	42	56	0	4(3.92%)	1690.43	16.57	8.47	306.47
	1	102	09/09/16	3.18.20	43	55	0	4(3.92%)	1888.06	18.51	8.24	438.76
	2	102	01/03/17	3.18.31	85	11	0	6(5.88%)	2421.44	23.74	5.49	1047.10
	3	102	05/18/17	3.18.31	92	4	0	6(5.88%)	1770.66	17.36	5.33	386.94
LG G5	4	103	05/27/16	3.18.20	32	65	0	6(5.88%)	2122.37	20.61	8.90	648.93
	5	103	10/26/17	3.18.31	95	0	0	8(7.77%)	1384.47	13.44	4.76	229.46
Huawei P9	6	31	02/22/16	3.10.90	10	20	0	1(3.23%)	390.35	12.59	8.47	89.35
	7	30	05/22/17	4.1.18	25	2	0	3(10.00%)	515.64	17.19	7.4	279.49

* Some patches we collected are not applicable for certain test subject kernels.

Accuracy

Device	No.	Patch Cnt*	Build Date (mm/dd/yy)	Kernel Version	Accuracy				Online Matching Time (s)			
					TP	TN	FP	FN	Total	Avg	~70%	Max.
Samsung S7	0	102	06/24/16	3.18.20	42	56	0	4(3.92%)	1690.43	16.57	8.47	306.47
	1	102	09/09/16	3.18.20	43	55	0	4(3.92%)	1888.06	18.51	8.24	438.76
	2	102	01/03/17	3.18.31	85	11	0	6(5.88%)	2421.44	23.74	5.49	1047.10
	3	102	05/18/17	3.18.31	92	4	0	6(5.88%)	1770.66	17.36	5.33	386.94
LG G5	4	103	05/27/16	3.18.20	32	65	0	6(5.88%)	2122.37	20.61	8.90	648.93
	5	103	10/26/17	3.18.31	95	0	0	8(7.77%)	1384.47	13.44	4.76	229.46
Huawei P9	6	31	02/22/16	3.10.90	10	20	0	1(3.23%)	390.35	12.59	8.47	89.35
	7	30	05/22/17	4.1.18	25	2	0	3(10.00%)	515.64	17.19	7.4	279.49

* Some patches we collected are not applicable for certain test subject kernels.

Accuracy: excellent, on average 94% accuracy w/o FP.

Accuracy

Device	No.	Patch Cnt*	Build Date (mm/dd/yy)	Kernel Version	Accuracy				Online Matching Time (s)			
					TP	TN	FP	FN	Total	Avg	~70%	Max.
Samsung S7	0	102	06/24/16	3.18.20	42	56	0	4(3.92%)	1690.43	16.57	8.47	306.47
	1	102	09/09/16	3.18.20	43	55	0	4(3.92%)	1888.06	18.51	8.24	438.76
	2	102	01/03/17	3.18.31	85	11	0	6(5.88%)	2421.44	23.74	5.49	1047.10
	3	102	05/18/17	3.18.31	92	4	0	6(5.88%)	1770.66	17.36	5.33	386.94
LG G5	4	103	05/27/16	3.18.20	32	65	0	6(5.88%)	2122.37	20.61	8.90	648.93
	5	103	10/26/17	3.18.31	95	0	0	8(7.77%)	1384.47	13.44	4.76	229.46
Huawei P9	6	31	02/22/16	3.10.90	10	20	0	1(3.23%)	390.35	12.59	8.47	89.35
	7	30	05/22/17	4.1.18	25	2	0	3(10.00%)	515.64	17.19	7.4	279.49

* Some patches we collected are not applicable for certain test subject kernels.

FP: we wrongly believe the patch is present. *Dangerous!*

FN: we wrongly believe the patch is not there. *Extra time to confirm.*

Accuracy: excellent, on average 94% accuracy w/o FP.

Why FN?

- Function inline.
- Function prototype change.
- Code customization.
- Patch adaptation.
- Other engineering issues.

Refer to section 6.2 in the paper for more details.

Why FN?

Function inline:

Added new callee function in the change site is inlined in different ways across reference and target binaries.

Why FN?

Patch adaptation:

The change site itself has been customized during patch porting.

Performance

Device	No.	Patch Cnt*	Build Date (mm/dd/yy)	Kernel Version	Accuracy				Online Matching Time (s)			
					TP	TN	FP	FN	Total	Avg	~70%	Max.
Samsung S7	0	102	06/24/16	3.18.20	42	56	0	4(3.92%)	1690.43	16.57	8.47	306.47
	1	102	09/09/16	3.18.20	43	55	0	4(3.92%)	1888.06	18.51	8.24	438.76
	2	102	01/03/17	3.18.31	85	11	0	6(5.88%)	2421.44	23.74	5.49	1047.10
	3	102	05/18/17	3.18.31	92	4	0	6(5.88%)	1770.66	17.36	5.33	386.94
LG G5	4	103	05/27/16	3.18.20	32	65	0	6(5.88%)	2122.37	20.61	8.90	648.93
	5	103	10/26/17	3.18.31	95	0	0	8(7.77%)	1384.47	13.44	4.76	229.46
Huawei P9	6	31	02/22/16	3.10.90	10	20	0	1(3.23%)	390.35	12.59	8.47	89.35
	7	30	05/22/17	4.1.18	25	2	0	3(10.00%)	515.64	17.19	7.4	279.49

* Some patches we collected are not applicable for certain test subject kernels.

Performance

Device	No.	Patch Cnt*	Build Date (mm/dd/yy)	Kernel Version	Accuracy				Online Matching Time (s)			
					TP	TN	FP	FN	Total	Avg	~70%	Max.
Samsung S7	0	102	06/24/16	3.18.20	42	56	0	4(3.92%)	1690.43	16.57	8.47	306.47
	1	102	09/09/16	3.18.20	43	55	0	4(3.92%)	1888.06	18.51	8.24	438.76
	2	102	01/03/17	3.18.31	85	11	0	6(5.88%)	2421.44	23.74	5.49	1047.10
	3	102	05/18/17	3.18.31	92	4	0	6(5.88%)	1770.66	17.36	5.33	386.94
LG G5	4	103	05/27/16	3.18.20	32	65	0	6(5.88%)	2122.37	20.61	8.90	648.93
	5	103	10/26/17	3.18.31	95	0	0	8(7.77%)	1384.47	13.44	4.76	229.46
Huawei P9	6	31	02/22/16	3.10.90	10	20	0	1(3.23%)	390.35	12.59	8.47	89.35
	7	30	05/22/17	4.1.18	25	2	0	3(10.00%)	515.64	17.19	7.4	279.49

* Some patches we collected are not applicable for certain test subject kernels.

Performance: acceptable, some cases may take long time to match, overall still much more efficient than manual work. Parallelization is also easily possible.

Un-ported patches

CVE	Type**	Severity*
CVE-2014-9781	P	High
CVE-2016-2502	P	High
CVE-2016-3813	I	Moderate
CVE-2016-4578	I	Moderate
CVE-2016-2184	P	Critical
CVE-2016-7910	P	Critical
CVE-2016-8413	I	Moderate
CVE-2016-10200	P	Critical
CVE-2016-10229	E	Critical

* Obtained from Android security bulletin.

** **P:** Privilege Elevation **E:** Remote Code Execution

** **I:** Information Disclosure

Un-ported patches

CVE	Type**	Severity*
CVE-2014-9781	P	High
CVE-2016-2502	P	High
CVE-2016-3813	I	Moderate
CVE-2016-4578	I	Moderate
CVE-2016-2184	P	Critical
CVE-2016-7910	P	Critical
CVE-2016-8413	I	Moderate
CVE-2016-10200	P	Critical
CVE-2016-10229	E	Critical

* Obtained from Android security bulletin.

** **P:** Privilege Elevation **E:** Remote Code Execution

** **I:** Information Disclosure

Lag (month)	Cnt.
1	2
2	5
6	2

Un-ported patches

CVE	Type**	Severity*
CVE-2014-9781	P	High
CVE-2016-2502	P	High
CVE-2016-3813	I	Moderate
CVE-2016-4578	I	Moderate
CVE-2016-2184	P	Critical
CVE-2016-7910	P	Critical
CVE-2016-8413	I	Moderate
CVE-2016-10200	P	Critical
CVE-2016-10229	E	Critical

* Obtained from Android security bulletin.

** **P:** Privilege Elevation **E:** Remote Code Execution

** **I:** Information Disclosure

Lag (month)	Cnt.
1	2
2	5
6	2

Some critical patches were not propagated even after 6 months (confirmed)!

CVE-2016-7910


```
diff --git a/block/genhd.c b/block/genhd.c
index 3c9dede..0ad8796 100644
--- a/block/genhd.c
+++ b/block/genhd.c
@@ -856,6 +856,7 @@ static void disk_seqf_stop(                    , void
*v)                                    struct seq_file *seqf
   if (iter) {
       class_dev_iter_exit(iter);
       kfree(iter);
+       seqf->private = NULL;
   }
 }
```

0x0 → [X0+ offset]

CVE-2016-7910


```
diff --git a/block/genhd.c b/block/genhd.c
index 3c9dede..0ad8796 100644
--- a/block/genhd.c
+++ b/block/genhd.c
@@ -856,6 +856,7 @@ static void disk_seqf_stop(struct seq_file *seqf, void
*v)
   if (iter) {
       class_dev_iter_exit(iter);
       kfree(iter);
+      seqf->private = NULL;
   }
}
```

0x0 → [X0 + offset]

How much code do you write?

How much code do you write?

We use Angr as our symbolic execution engine. (w/ modifications)

How much code do you write?

We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.

How much code do you write?

We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.

Still under improvement.

How much code do you write?

We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.

Still under improvement.

Now fully open-sourced on Github!

https://fiberx.github.io

Thanks!

Q & A

https://fiberx.github.io

UNIVERSITY OF CALIFORNIA
UC RIVERSIDE