Real-time Dynamic Function Call Detection

A heuristic-based approach

Franck de Goër
UGA, VU Amsterdam
ANSSI

Sanjay Rawat
VU Amsterdam

Dennis Andriesse
VU Amsterdam

Herbert Bos
VU Amsterdam

Roland Groz
UGA

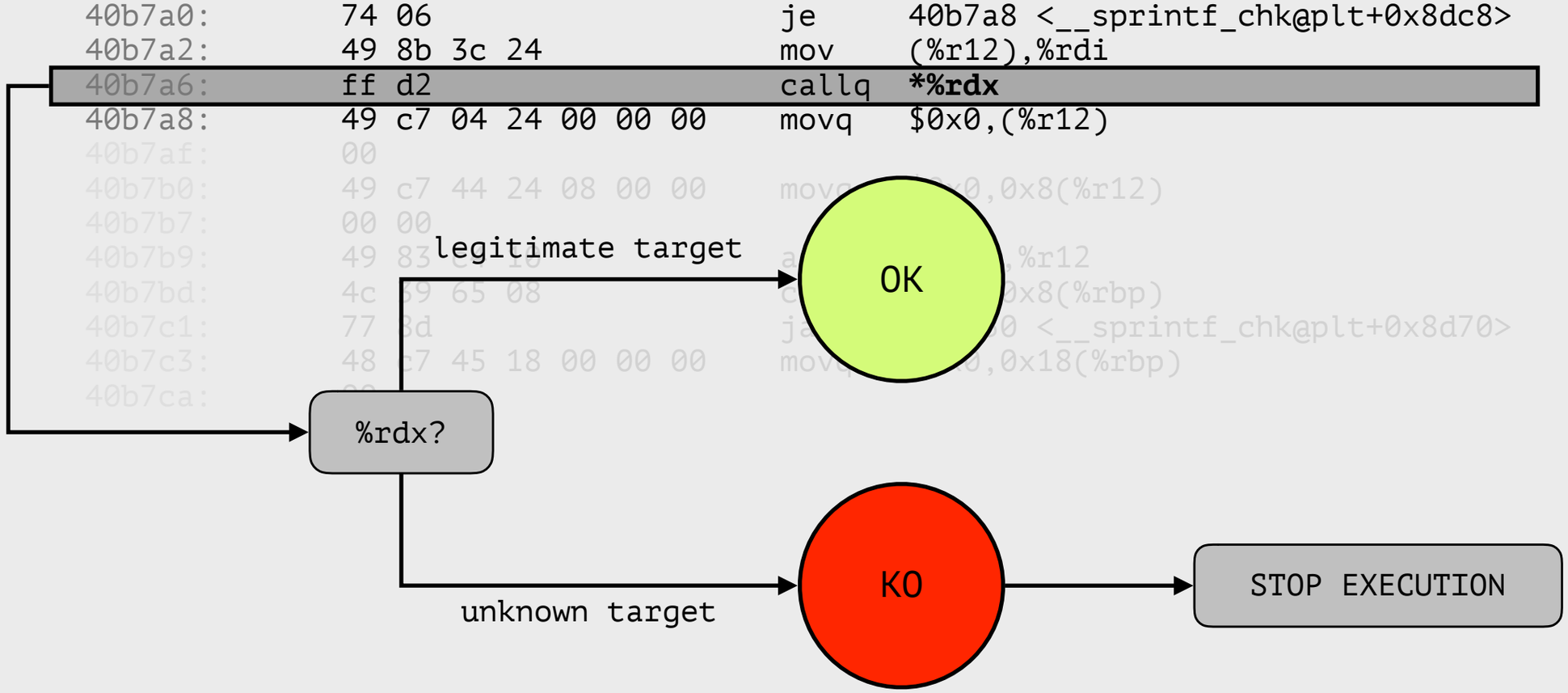
INTRODUCTION

let’s try to keep everyone awake
for two minutes

AN EXAMPLE TO ILLUSTRATE THE PROBLEM – CFI

```
40b77d:   48 8b 55 40              mov    0x40(%rbp),%rdx
40b781:   48 8b 43 08              mov    0x8(%rbx),%rax
40b785:   48 8b 4d 48              mov    0x48(%rbp),%rcx
40b789:   48 c7 03 00 00 00 00     movq   $0x0,(%rbx)
40b790:   48 85 c0                 test   %rax,%rax
40b793:   48 89 4b 08              mov    %rcx,0x8(%rbx)
40b797:   48 89 5d 48              mov    %rbx,0x48(%rbp)
40b79b:   75 d3                    jne    40b770 <__sprintf_chk@plt+0x8d90>
40b79d:   48 85 d2                 test   %rdx,%rdx
40b7a0:   74 06                    je     40b7a8 <__sprintf_chk@plt+0x8dc8>
40b7a2:   49 8b 3c 24              mov    (%r12),%rdi
40b7a6:   ff d2                    callq  *%rdx
40b7a8:   49 c7 04 24 00 00 00     movq   $0x0,(%r12)
40b7af:   00
40b7b0:   49 c7 44 24 08 00 00     movq   $0x0,0x8(%r12)
40b7b7:   00 00
40b7b9:   49 83 c4 10              add    $0x10,%r12
40b7bd:   4c 39 65 08              cmp    %r12,0x8(%rbp)
40b7c1:   77 3d                    ja     40b800 <__sprintf_chk@plt+0x8d70>
40b7c3:   48 c7 45 18 00 00 00     movq   $0x0,0x18(%rbp)
40b7ca:   00
```

execution



AN EXAMPLE TO ILLUSTRATE THE PROBLEM – CFI

execution

```
40b77d:       48 8b 55 40             mov    0x40(%rbp),%rdx
40b781:       48 8b 43 08             mov    0x8(%rbx),%rax
40b785:       48 8b 4d 48             mov    0x48(%rbp),%rcx
40b789:       48 c7 03 00 00 00 00    movq   $0x0,(%rbx)
40b790:       48 85 c0                test   %rax,%rax
40b793:       48 89 4b 08             mov    %rcx,0x8(%rbx)
40b797:       48 89 5d 48             mov    %rbx,0x48(%rbp)
40b79b:       75 d3                   jne    40b770 <__sprintf_chk@plt+0x8d90>
40b79d:       48 85 d2                test   %rdx,%rdx
40b7a0:       74 06                   je     40b7a8 <__sprintf_chk@plt+0x8dc8>
40b7a2:       49 8b 3c 24             mov    (%r12),%rdi
                                      push   40b7a8
                                      jmp    *%rdx
40b7a6:       ff d2                   callq  *%rdx
40b7a8:       49 c7 04 24 00 00 00    movq   $0x0,(%r12)
40b7af:       00
40b7b0:       49 c7 44 24 08 00 00    movq   $0x0,0x8(%r12)
40b7b7:       00 00
40b7b9:       49 83 c4 10             add    $0x10,%r12
40b7bd:       4c 39 65 08             cmp    %r12,0x8(%rbp)
40b7c1:       77 8d                   ja     40b750 <__sprintf_chk@plt+0x8d70>
40b7c3:       48 c7 45 18 00 00 00    movq   $0x0,0x18(%rbp)
40b7ca:       00
```

CFI **SHOULD** APPLY, BUT WILL IT?

THE PROBLEM

HOW TO INSTRUMENT **EVERY CALL**

DURING EXECUTION?

THE REAL PROBLEM

HOW TO DISTINGUISH **INTRA-PROCEDURAL**

JUMPS FROM **JUMP-BASED CALLS**?

THE REAL REAL PROBLEM

HOW TO DISTINGUISH **INTRA-PROCEDURAL JUMPS** FROM **TAIL CALLS**?

OUR PROPOSED SOLUTION: iCi

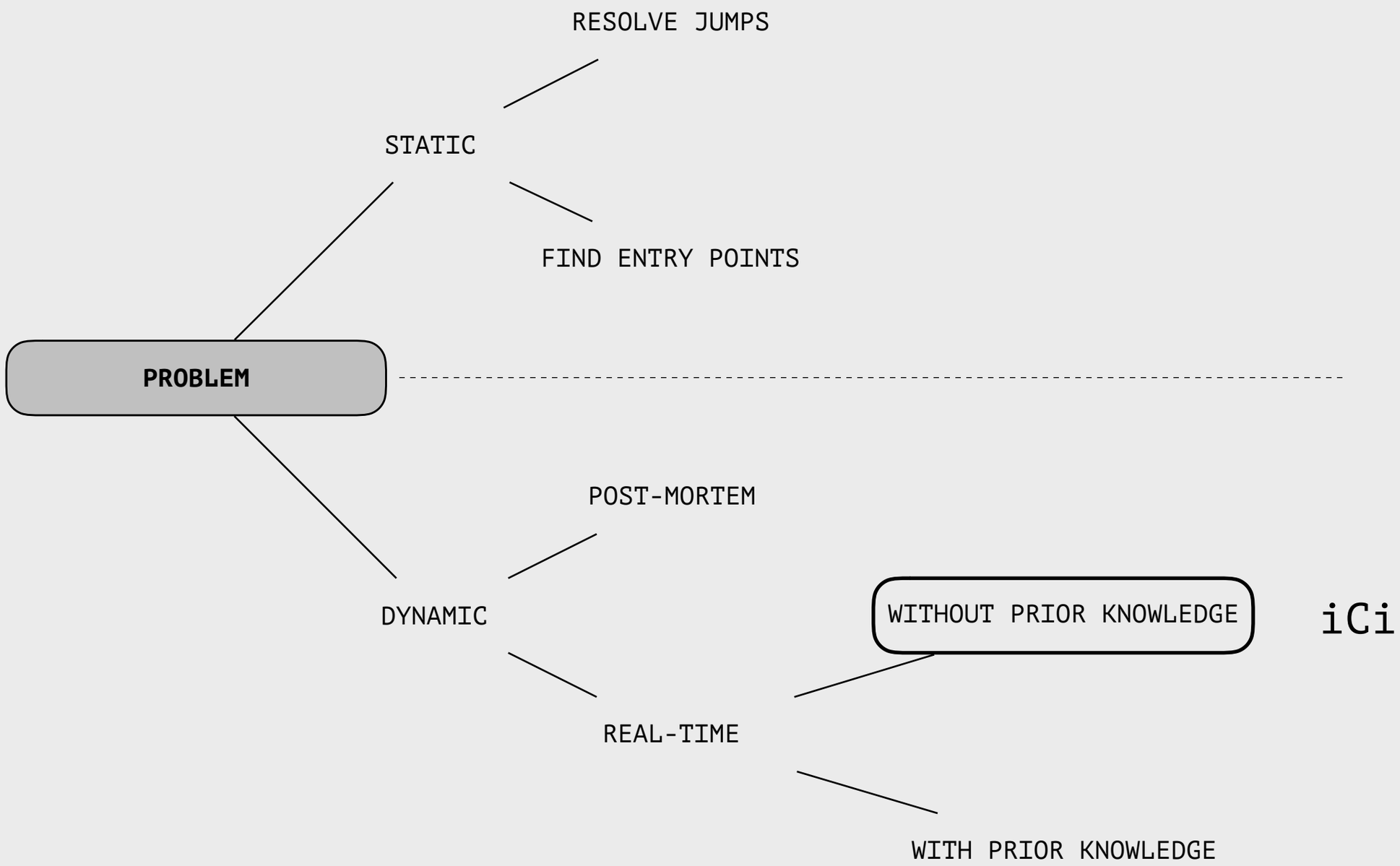
	-O0	-O1	-O2	-O3
	iCi	iCi	iCi	iCi
binutils	1,000	1,000	1,000	1,000
evince	0,991	0,985	0,985	0,986
coreutils	0,999	0,998	0,998	0,998
ffmpeg	0,997	0,997	0,997	0,997
SPEC2006 jcall worst	nc	nc	1,000	nc
SPEC2006 iCi worst	nc	nc	0,947	nc

fscore

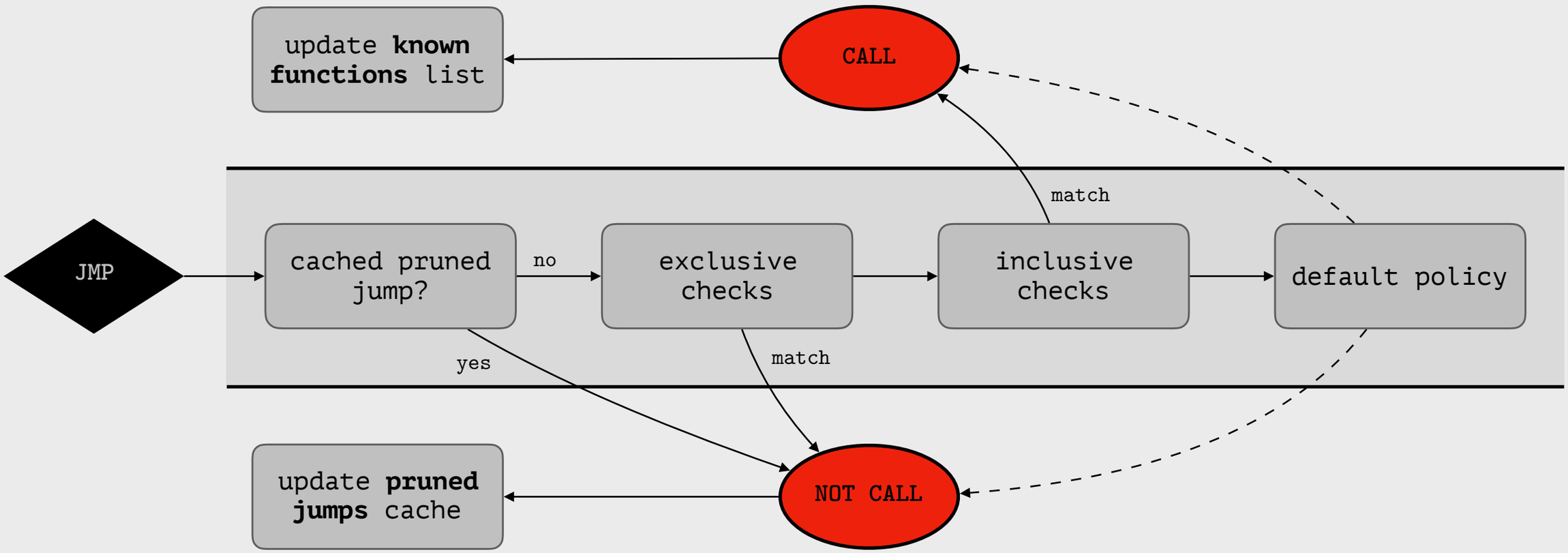
APPROACH

OVERVIEW

POSSIBLE APPROACHES



JMP SELECTION IN iCi



BACKGROUND INFORMATION

LIST OF ENTRY POINTS

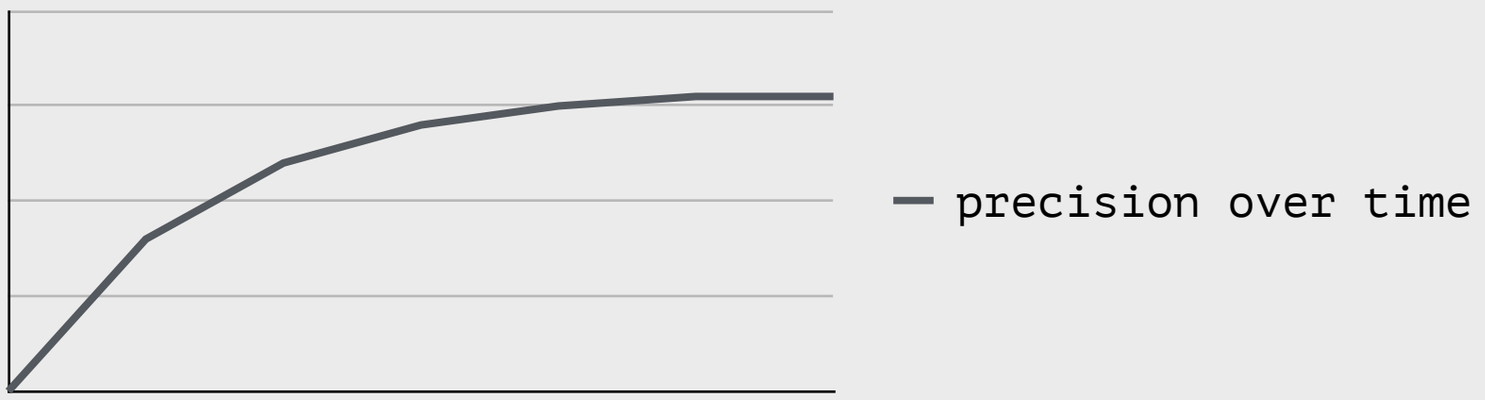
Each time a function call is detected, its **target** is marked as an **entry point**.

LIST OF RETURN LOCATIONS

Each time a function returns, the location of the last instruction is marked as a **return point**.

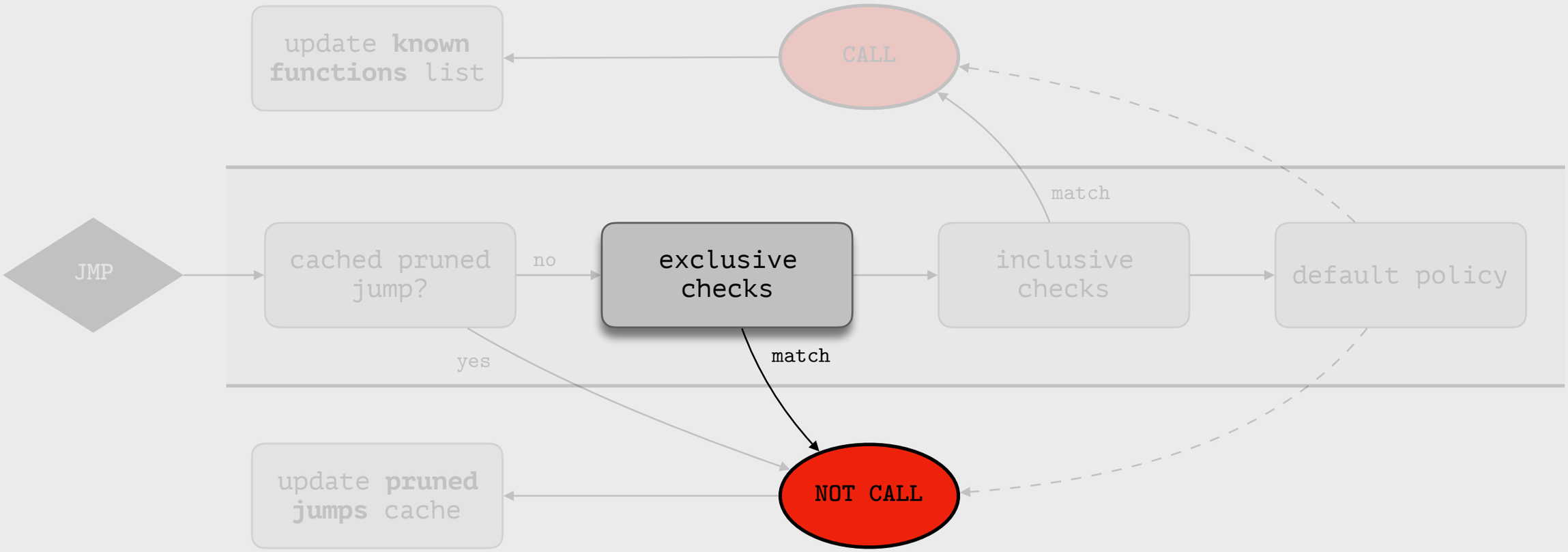
AVAILABLE INFO FOR EACH JMP

information	accessible?
target of the JMP	ok (dynamic)
current %rip, %rsp	ok (dynamic)
current entry point	**~ok (depend on the known entry points)**
state of stack when last call occurred	**~ok (idem)**
return locations for the current function	**not at first call; then ~ok**



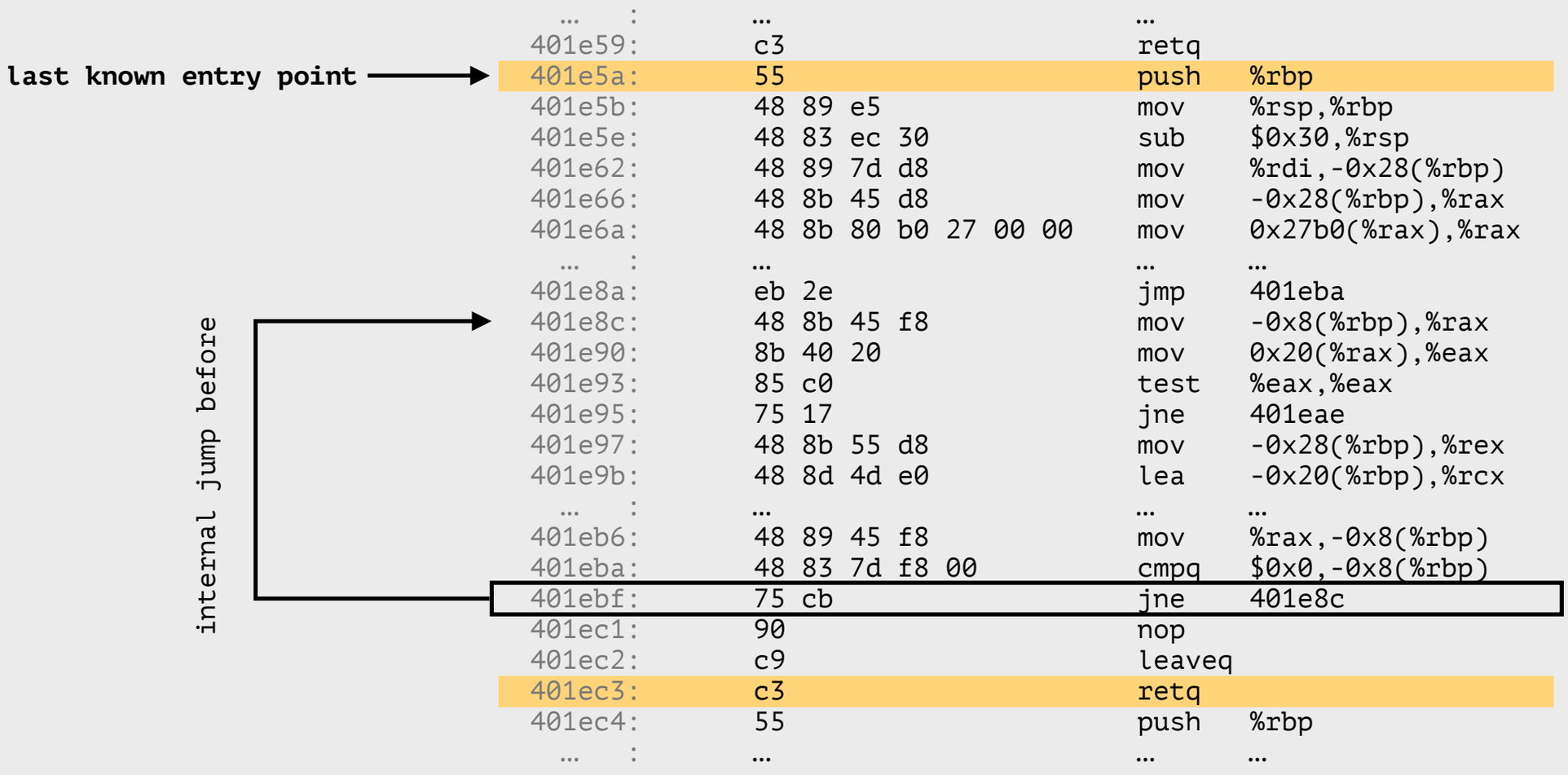
HEURISTICS

exclusive checks



INTERNAL JUMP BEFORE

A jump to a location that is between the current entry point and the current %rip **is not a call**.



ERROR EXAMPLE

last known entry point →

```
000000000040ed24 <uninit >:
    …

000000000040ed47 <init >:
    …

000000000040ee41 <init_alphaextract >:   ← real entry point
    40ee41: mov 0x48(%rdi),%rax
    40ee45: movl $0x8,0x8(%rax)
    40ee4c: jmpq 40ed47 <init>
```

INTERNAL JUMP AFTER

A jump to a location that is between the current %rip and a known ret of the current function **is not a call**.

```
  …   :        …                          …
401e59:        c3                         retq
401e5a:        55                         push   %rbp
401e5b:        48 89 e5                   mov    %rsp,%rbp
401e5e:        48 83 ec 30                sub    $0x30,%rsp
401e62:        48 89 7d d8                mov    %rdi,-0x28(%rbp)
401e66:        48 8b 45 d8                mov    -0x28(%rbp),%rax
401e6a:        48 8b 80 b0 27 00 00       mov    0x27b0(%rax),%rax
  …   :        …                          …      …
401e8a:        eb 2e                      jmp    401eba
401e8c:        48 8b 45 f8                mov    -0x8(%rbp),%rax
401e90:        8b 40 20                   mov    0x20(%rax),%eax
401e93:        85 c0                      test   %eax,%eax
401e95:        75 17                      jne    401eae
401e97:        48 8b 55 d8                mov    -0x28(%rbp),%rex
401e9b:        48 8d 4d e0                lea    -0x20(%rbp),%rcx
  …   :        …                          …      …
401eb6:        48 89 45 f8                mov    %rax,-0x8(%rbp)
401eba:        48 83 7d f8 00             cmpq   $0x0,-0x8(%rbp)
401ebf:        75 cb                      jne    401e8c
401ec1:        90                         nop
401ec2:        c9                         leaveq
401ec3:        c3                         retq
401ec4:        55                         push   %rbp
  …   :        …                          …      …
```

internal jump after

known ret

STACK INCONSISTENCY (IDEA)

A jump occurring when %ebp ≠ %esp **is not a (tail) call**.

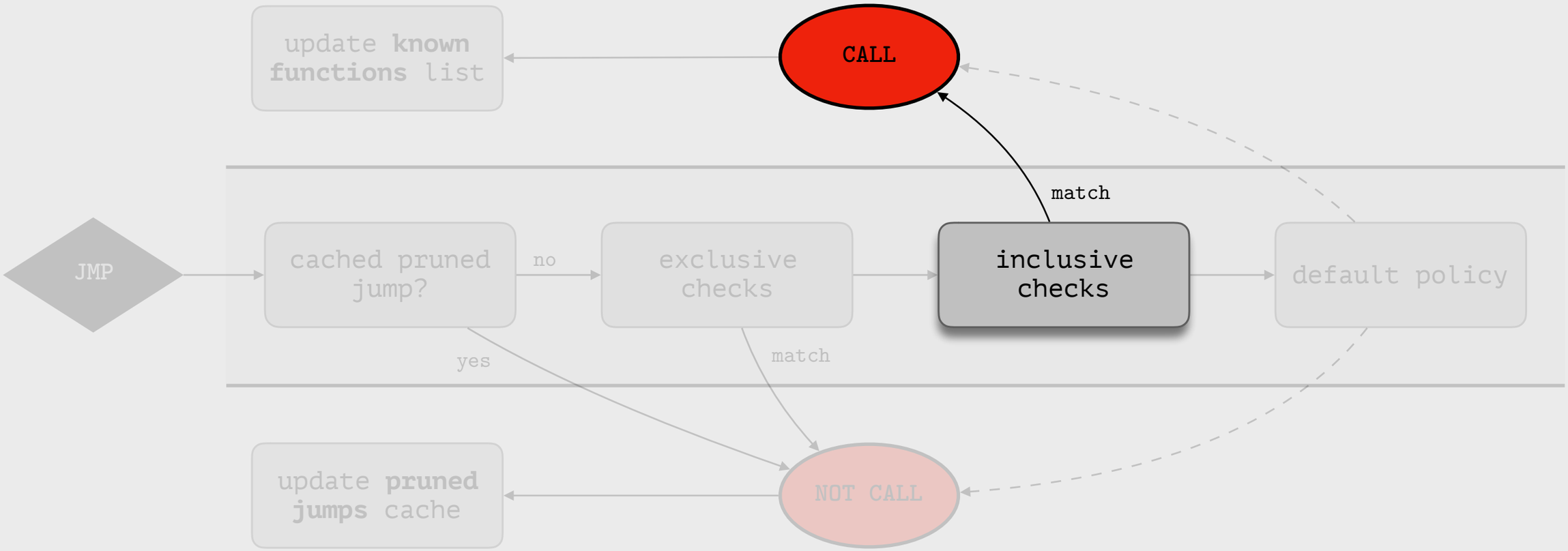
```
push    %rbp
mov     %rsp,%rbp
sub     $0x30,%rsp
push    …
pop     …
push    …
pop     …
push    …
jmp     XXX
```

4 push / 2 pop => NOT A CALL

```
push    %rbp
mov     %rsp,%rbp
sub     $0x30,%rsp
push    …
push    …
pop     …
pop     …
pop     …
jmp     XXX
```

3 push / 3 pop => MAYBE A CALL

inclusive checks



KNOWN ENTRY POINT

A jump to a known entry point **is a call**.

EXTERNAL JUMP BEFORE

A jump to a location that is before the current entry point **is a call**.

CROSS ENTRY POINT

A jump crossing a known entry point **is a call**.

EXPERIMENTS

introducing iCi

METHODOLOGY

ORACLE

INPUT: entry points of functions

ALGORITHM: for each non-sequential update of %rip, check its new value. If entry point => call.

REMARKS: existing corner cases, but best oracle we have

JCALL

ALGORITHM: CALL instructions + JMP to .plt => calls

REMARKS: low overhead, no false positive

JMP

ALGORITHM: CALL + JMP instructions => calls

REMARKS: high overhead, no false negative

BENCHMARK

BINARIES

- coreutils: 98 programs
- binutils: 13 programs
- evince: 1 program
- ffmpeg: 1 program
- SPEC CPU2006 (C/C++): 19 programs

OPTIMIZATION LEVEL

for *\{SPEC CPU2006}: -O0 to -O3 with gcc-6.0

COMPILATION

for coreutils: -O0 to -O3 with gcc-6.0 and clang-3.8

RESULTS

F-SCORE

	-O0			-O1			-O2			-O3		
	jcall	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**
binutils	1,000	0,223	1,000	1,000	0,222	1,000	0,999	0,222	1,000	1,000	0,221	1,000
evince	0,988	0,831	0,991	0,980	0,796	0,985	0,936	0,803	0,985	0,947	0,818	0,986
coreutils	0,998	0,518	0,999	0,998	0,352	0,998	0,946	0,405	0,998	0,997	0,351	0,998
ffmpeg	**0,948**	**0,227**	**0,997**	**0,919**	**0,183**	**0,997**	**0,874**	**0,182**	**0,997**	**0,884**	**0,181**	**0,997**
SPEC2006 jcall worst	nc	nc	nc	nc	nc	nc	0,848	0,432	1,000	nc	nc	nc
SPEC2006 iCi worst	nc	nc	nc	nc	nc	nc	0,906	0,260	0,947	nc	nc	nc

OVERHEAD

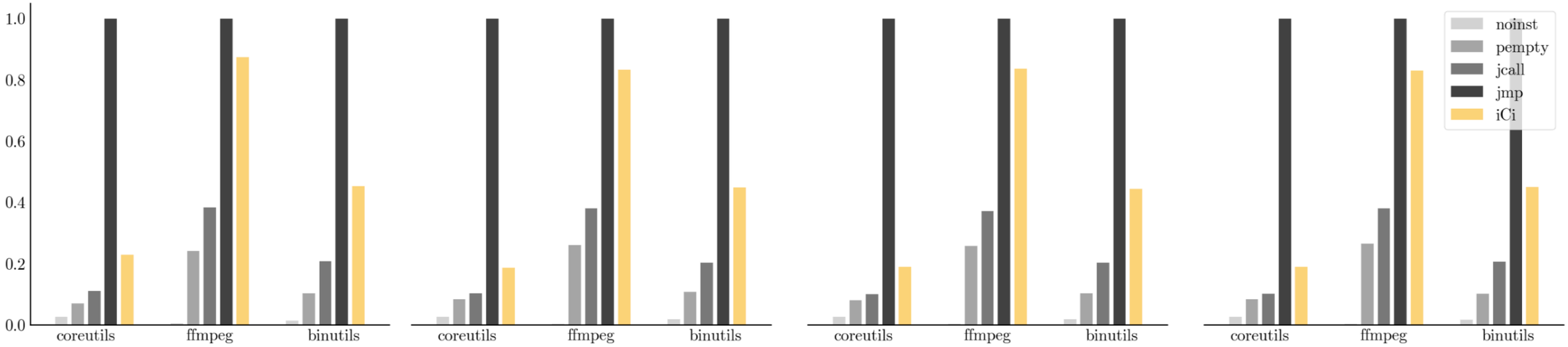


Figure 2: Overhead of each analysis in comparison with `jmp` for `-O2`

COMPILATION

	-O0			-O1			-O2			-O3		
	jcall	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**	**jcall**	**jmp**	**iCi**
gcc-6.3	0,998	0,518	0,999	0,998	0,352	0,998	0,946	0,405	0,998	0,997	0,351	0,998
clang-3.8	0,997	0,360	0,998	0,997	0,359	0,998	0,997	0,359	0,998	0,997	0,359	0,998

FINAL WORDS

time to wake everyone up

PROBLEM NOT ADDRESSED BEFORE (TO OUR KNOWLEDGE)

- **TOOLS**: No high-level API to instrument every call (PIN, DynamoRIO, etc.)
- **RESEARCH**: A lot of papers about tail calls… but none about detecting them dynamically

RESULTS INTERPRETATION

iCi is cool! It achieves great f-scores on C/C++ programs, and works with optimization levels and different compilers.

BUT . . .

RESULTS INTERPRETATION

Is it really worth it?

A simpler implementation (jcall) performs almost as good, with a lower overhead.

YES if it is critical to catch all calls

NO otherwise

OPENING

approach | heuristics | experiments

background information

AVAILABLE INFO FOR EACH JMP

information	accessible?
target of the JMP	ok (dynamic)
current %rip, %rsp	ok (dynamic)
current entry point	~ok (depend on the known entry points)
state of stack when last call occurred	~ok (idem)
return locations for the current function	not at first call; then ~ok

+ overhead decreases over execution

OPENING

BETTER OVER TIME: keep learnt info from an execution to another > should reduce overhead significantly

MIXED ANALYSIS: first static analysis (e.g. Nucleus)

https://github.com/Frky/iCi

IMPLEMENTATION

BENCHMARK (EXCEPT SPEC)

EVERYTHING NEEDED TO REPLAY