Speculator

A Tool to Analyze Speculative Execution Attacks and Mitigations

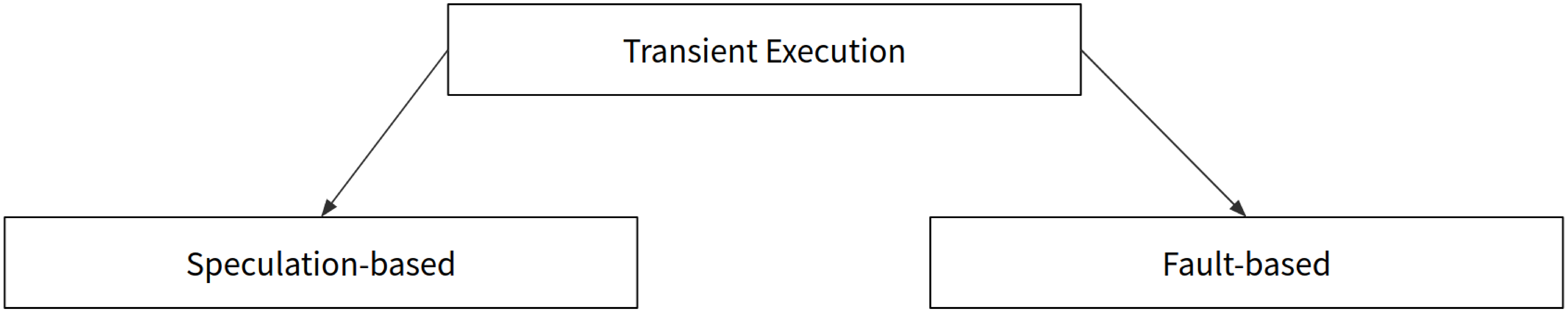
Andrea Mambretti
(mbr@ccs.neu.edu)

Matthias Neugschwandtner, Alessandro Sorniotti and Anil Kurmus - IBM Research
William Robertson and Engin Kirda - Northeastern University

Northeastern University

IBM Research

Transient Execution Attacks



For instance:

- Spectre v1
- Spectre v2

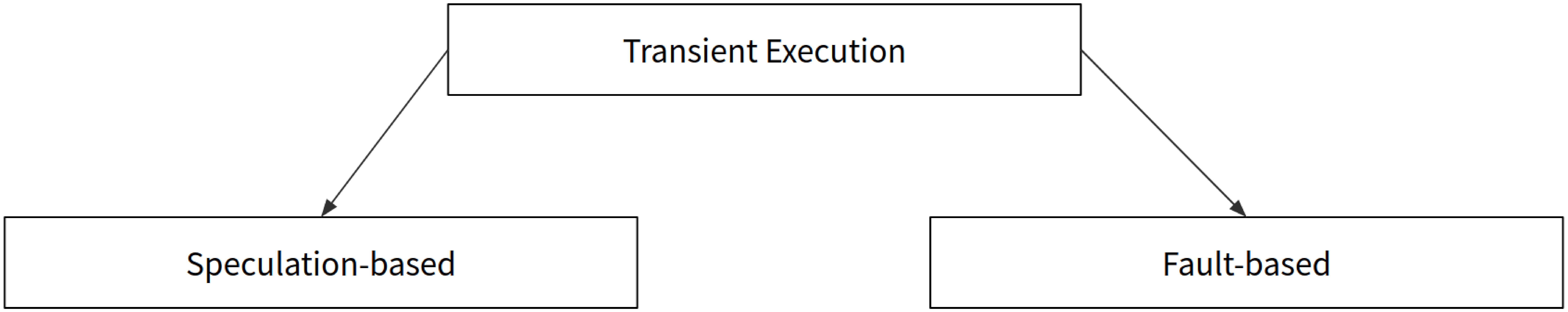
Long term problem…
meant to stay

For instance:

- Meltdown
- Foreshadow

Implementation problem
“Easily” fixable in new CPUs!

Transient Execution Attacks



For instance:

- Spectre v1
- Spectre v2

Long term problem…
meant to stay

For instance:

- Meltdown
- Foreshadow

Implementation problem
“Easily” fixable in new CPUs!

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Cached

Not Cached

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Cached

Not Cached

Example:

- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

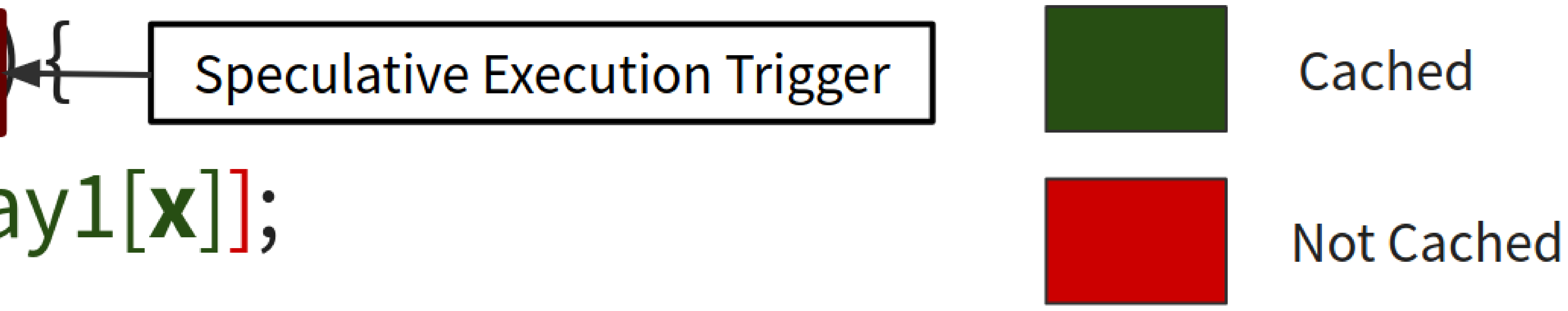
Cached

Not Cached

Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```



Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Cached

Not Cached

Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

Spectre v1 - Bounds Check Bypass

```
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- **x** = 15 (attacker controlled)

How can we study this type of attacks?

In memory corruption?

GDB

In speculative execution attacks (SEA)?

???

To understand a SEA, we should be able to **observe** it

Observe Speculation - Practice

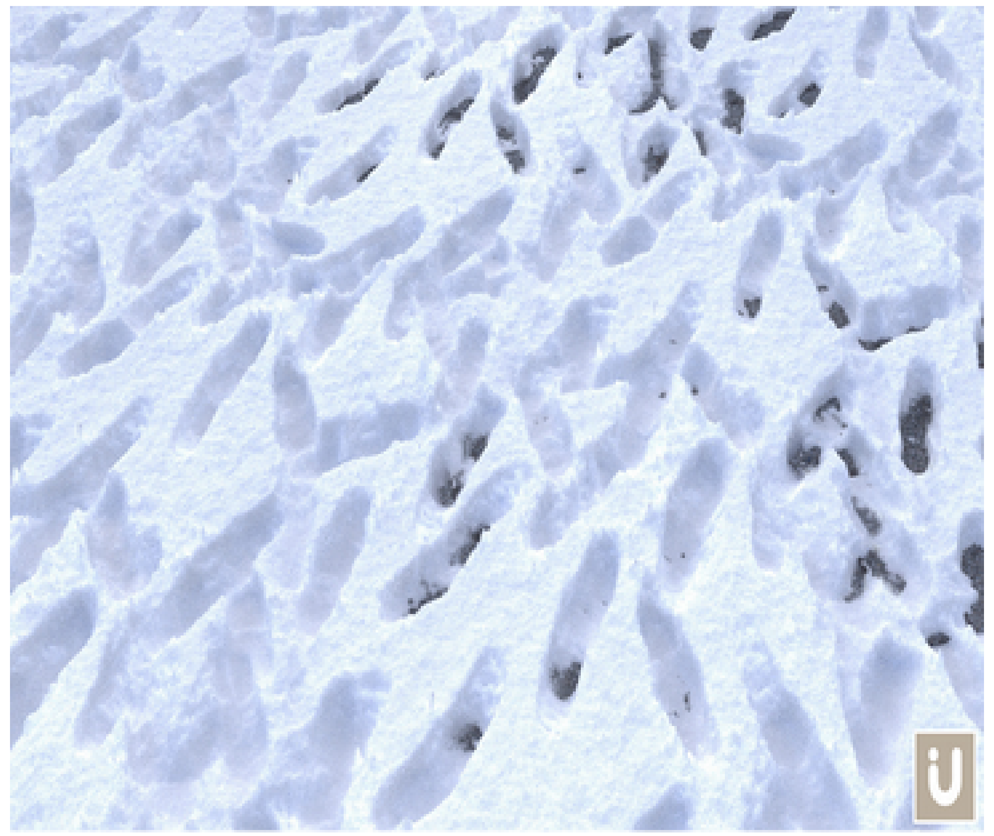
Side channels

Problem:

- Costly to setup
- Noisy to read
- Long execution time for each run

Performance counters (NEW)

- Model-specific
- Architecture-specific
- Plenty of counters available
- Implemented in all modern CPU



Speculative Execution Markers

Special **instructions** or sequences **detectable** by performance counters **even** when they **do not retire**

State of the art

Perf_events (Tool or Syscall)

Sampling mode: impossible to get quantitate info counting

Counting mode: high overhead due to in-kernel design

Likwid

Lack of flexibility, only system-wide measures

Others (e.g. Oprofile, Perfmon2, Perfctl, PAPI)

Outdated, inaccurate, not flexible or unmaintained

Speculator

Speculator

Based on CPU program counters

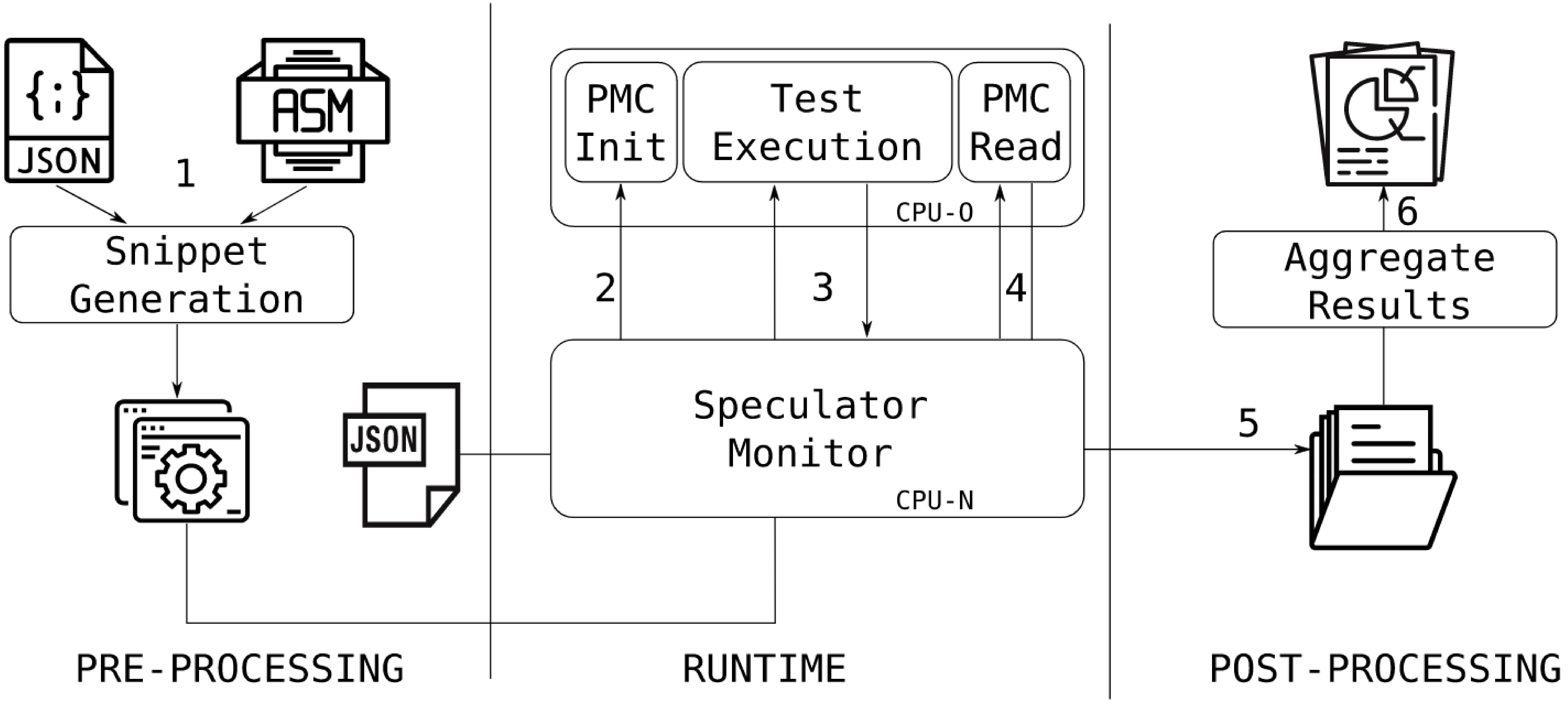
Direct configuration through MSR register

Creation of incremental snippets

Two modes of execution

- Test mode
- Attacker/Victim mode

Speculator



JSON
ASM
1
Snippet Generation
JSON
PMC Init
Test Execution
PMC Read
CPU-0
2
3
4
Speculator Monitor
CPU-N
5
6
Aggregate Results
PRE-PROCESSING
RUNTIME
POST-PROCESSING

Speculative Execution Markers (Intel)

UOPS_EXECUTED.CORE/THREAD

Count μ-ops executed by a CPU

UOPS_ISSUED.SINGLE_MUL

(e.g. **mulps xmm2, xmm1**)

Count single-precision floating-point instructions that operates on xmm register is issued

UOPS_ISSUED.SLOW_LEA

(e.g. **lea rax, [array+rax*2]**)

Count lea instruction with 3 operands

Drawback: clflush are counted as slow lea

LD_BLOCK.STORE_FORWARD

Count failed store forward

Example:

mov DWORD[array], eax

mov DWORD[array+4], edx

movq xmm0, QWORD[array]

Findings

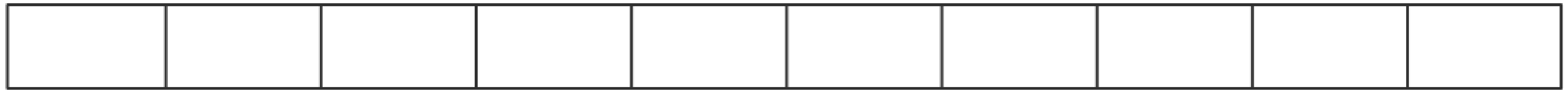
Findings - RSB Size

Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

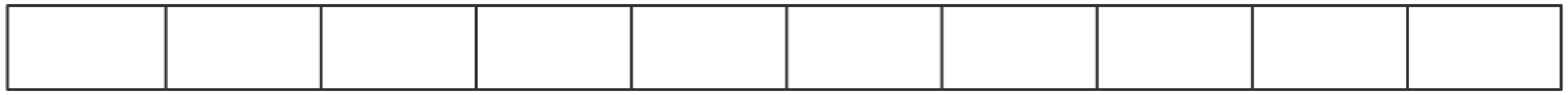
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Findings - RSB Size

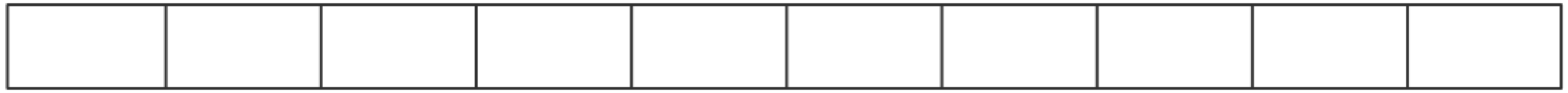
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

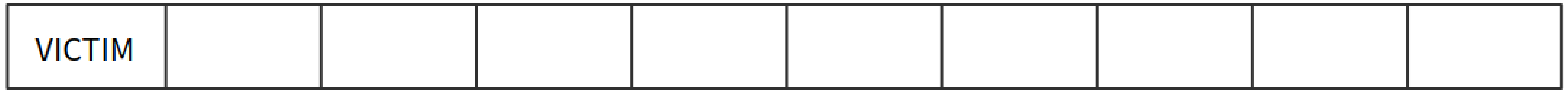
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

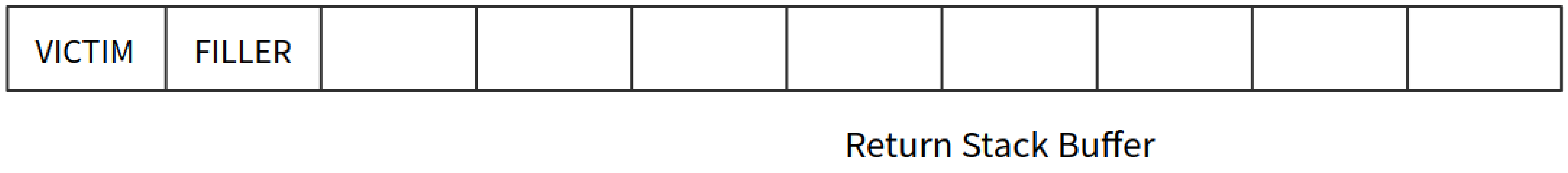
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

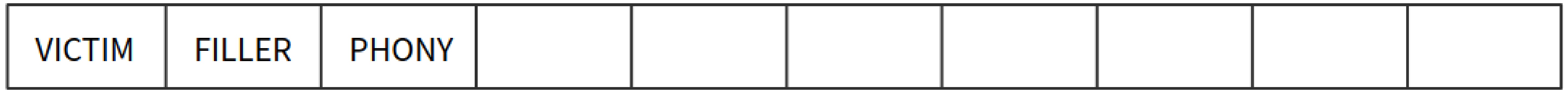
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Findings - RSB Size

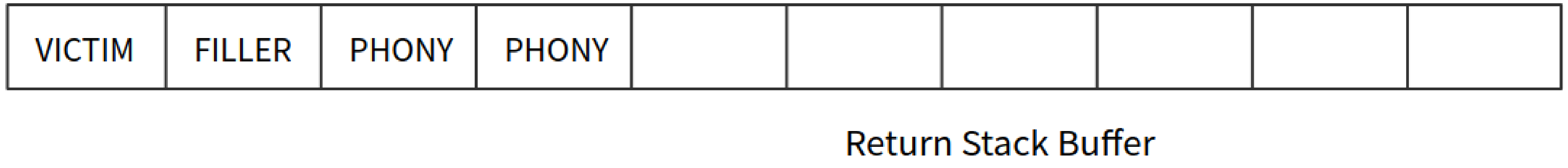
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

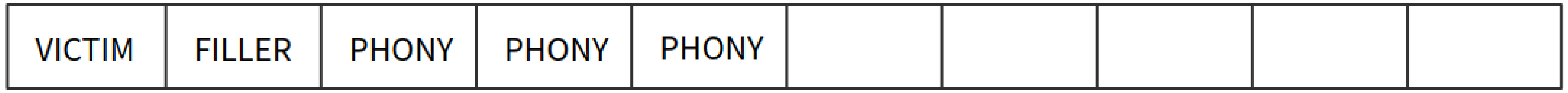
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Findings - RSB Size

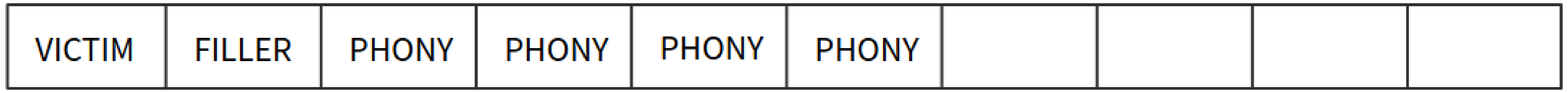
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

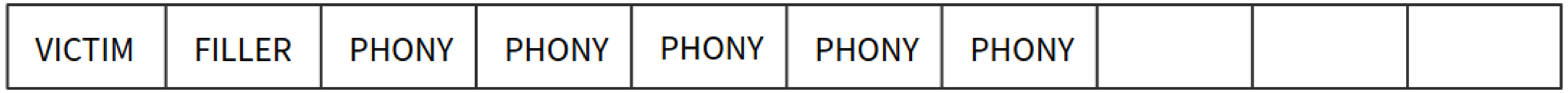
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

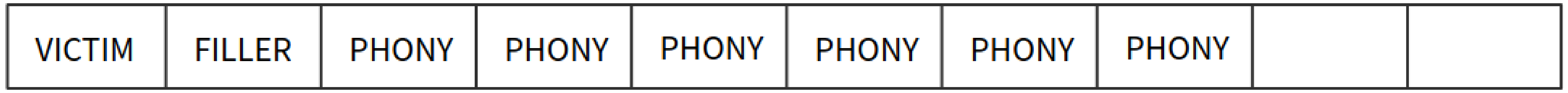
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

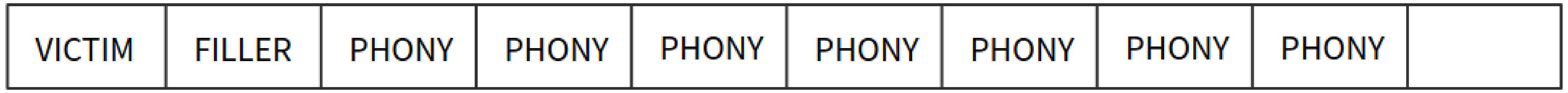
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

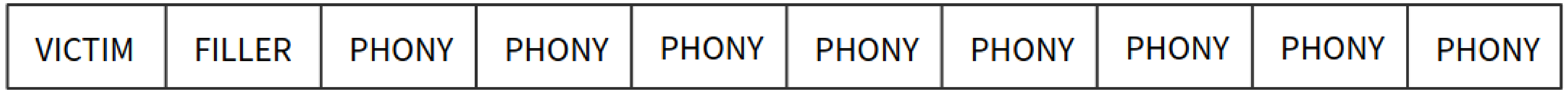
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

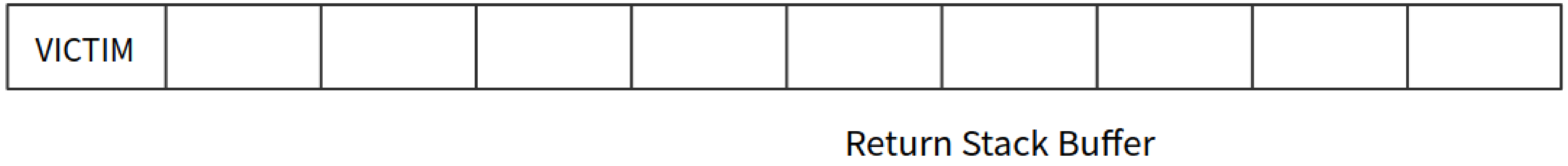
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

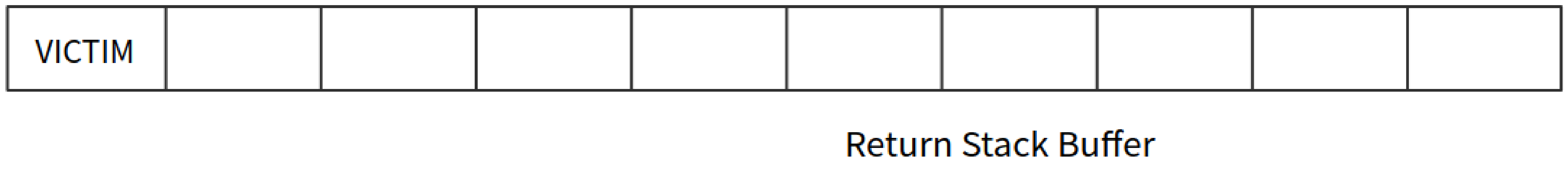
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



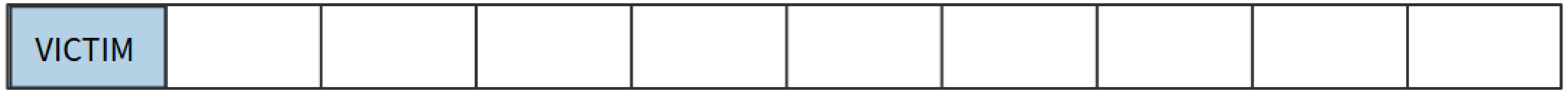
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Findings - RSB Size

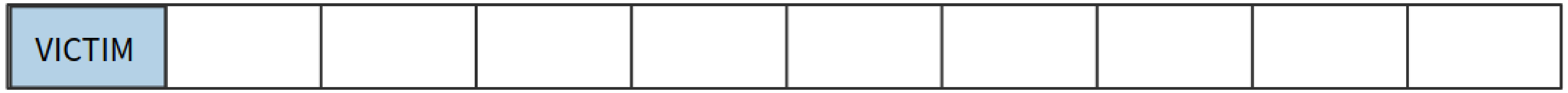
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

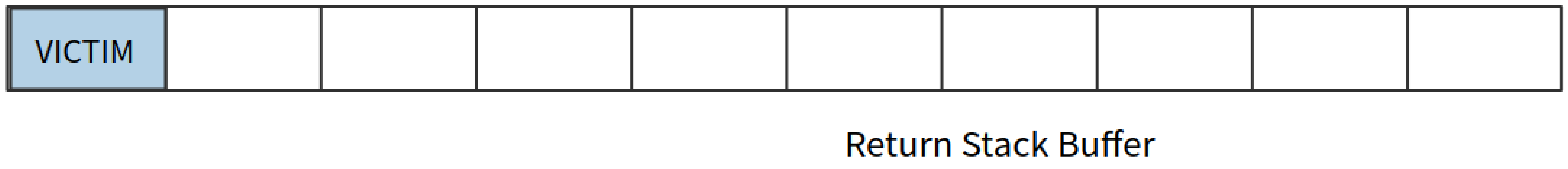


Return Stack Buffer

Findings - RSB Size

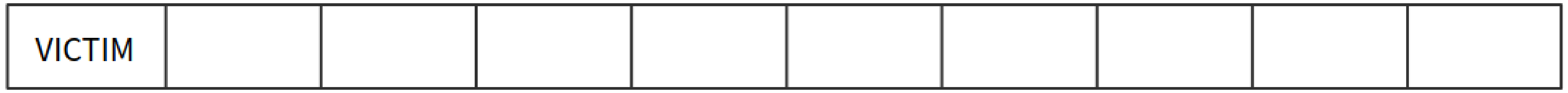
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

Marker Hit



Findings - RSB Size

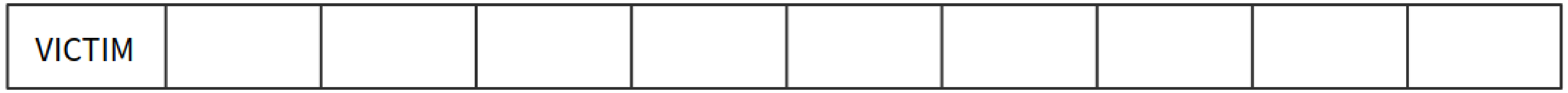
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

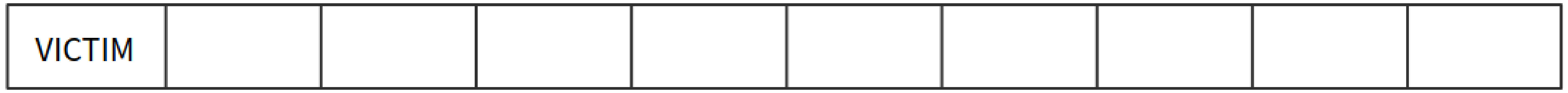
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

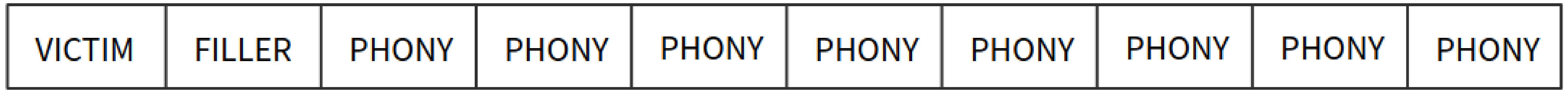
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

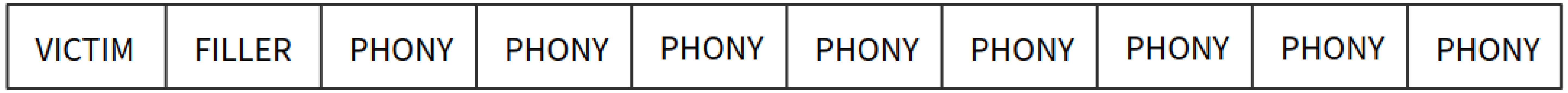
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

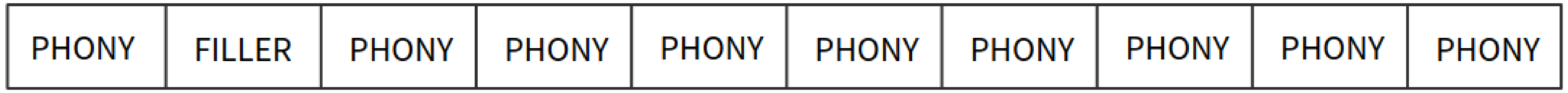
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

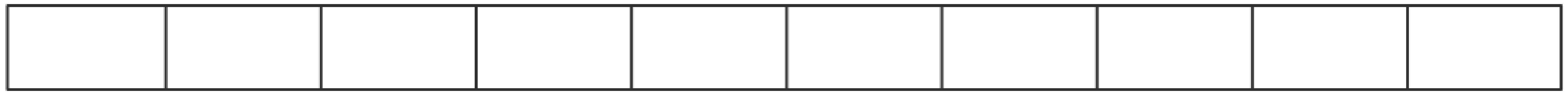
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

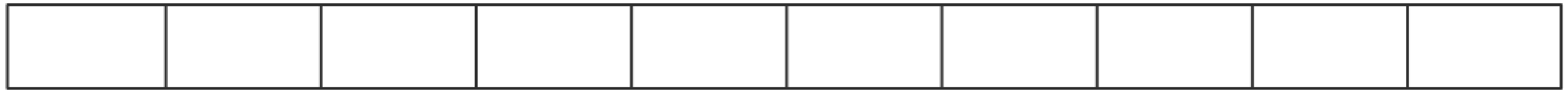
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

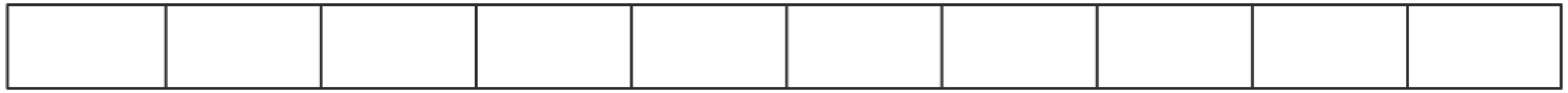
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

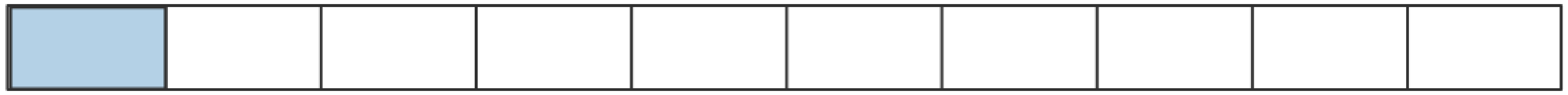
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

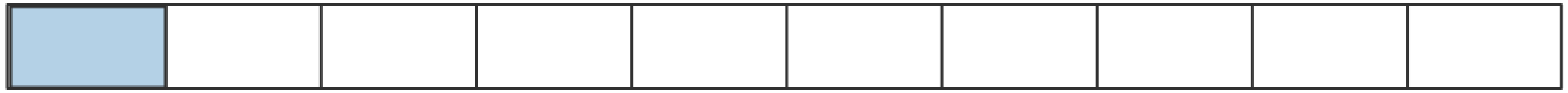


Return Stack Buffer

Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

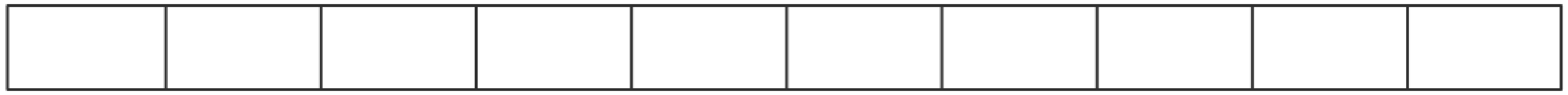
NO Marker Hit



Return Stack Buffer

Findings - RSB Size

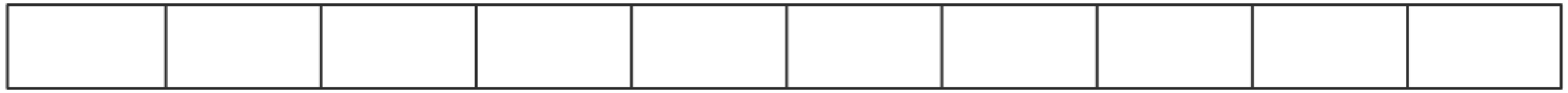
```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



Return Stack Buffer

Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```



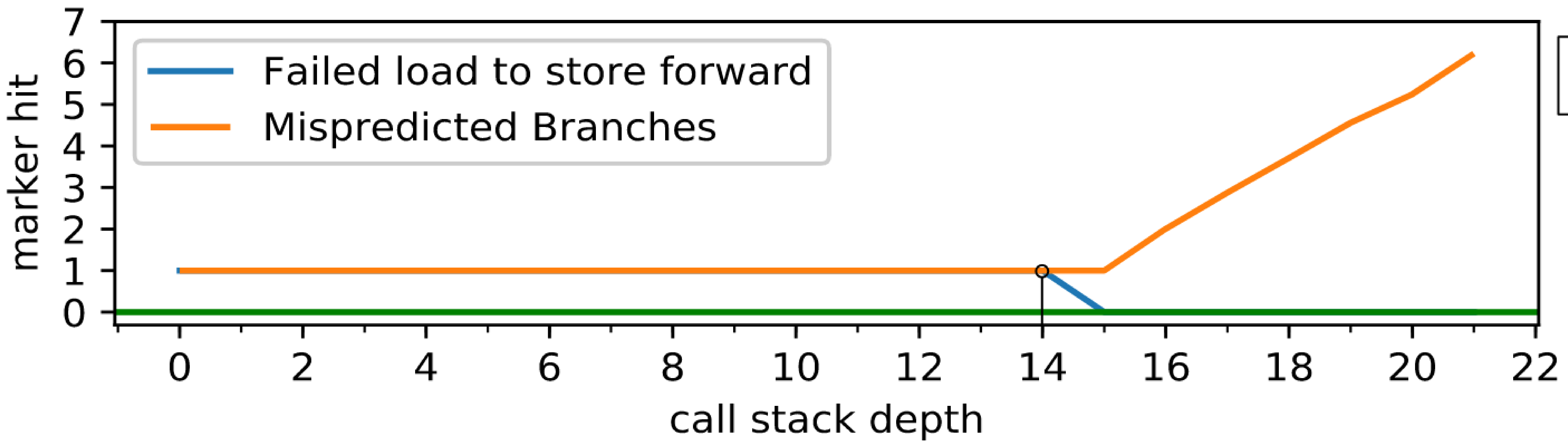
Return Stack Buffer

Therefore:

RSB size = VICTIM + FILLER + X nested call/ret

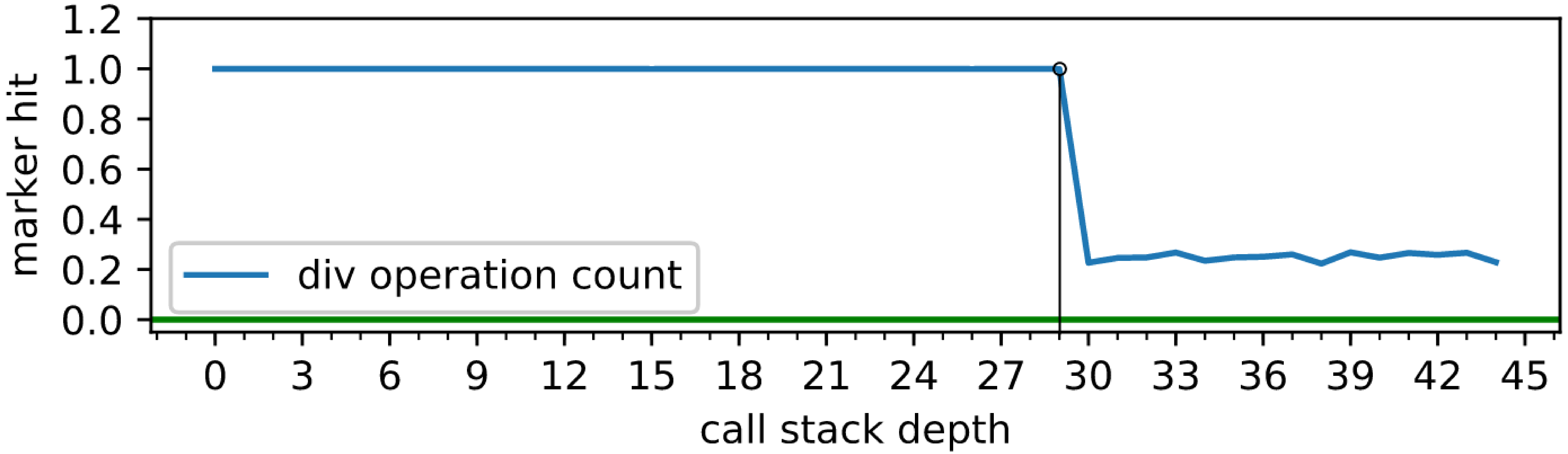
When X+1 nested calls cause no marker hit

Findings - RSB Size Results



RSB$_{size}$ = 1+ 1+ 14 Nested call/ret = 16

Intel



RSB$_{size}$ = 1+ 1+ 29 Nested call/ret = 31

AMD

Conclusions

New methodology to observe speculative execution **based on markers**

New low-overhead tool, **Speculator**, tailored to study new attacks and mitigations

Several **new insights** on speculative execution behavior on different CPU (e.g. Broadwell, Skylake, AMD Ryzen)

Speculator and the markers **easier** the **study** of old and new **attacks techniques**

Speculator is open source:

https://github.com/ibm-research/speculator

Questions?

Extra

Findings - Nesting Speculative Execution

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

TRAINING PHASE

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
	// marker
	if (slow condition) {
		// marker
		if (fast condition) {
			//marker
		}
		else  { //marker}
	}
	else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
        // marker
        if (slow condition) {
                // marker
                if (fast condition) {
                        //marker
                }
                else  { //marker}
        }
        else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Nesting Speculative Execution

```
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else  { //marker}
    }
    else { //marker}
}
else {//marker}
```

SPECULATIVE EXECUTION

Findings - Speculation Window Size

Findings - Speculation Window Size

Conditional Branches

Conditional branch	Broadwell	Skylake	Zen
Register access	14	16	7
Access to cached memory	19	17	9
Access to uncached memory	144	280	321
Mul with register	19	19	2
Mul with cached memory	33	33	8
Mul with uncached memory	154	290	362
Div with register	35	41	17
Div with cached memory	34	39	30
Div with uncached memory	164	306	353

Findings - Speculation Window Size

Conditional Branches

Conditional branch	Broadwell	Skylake	Zen
Register access	14	16	7
Access to cached memory	19	17	9
Access to uncached memory	144	280	321
Mul with register	19	19	2
Mul with cached memory	33	33	8
Mul with uncached memory	154	290	362
Div with register	35	41	17
Div with cached memory	34	39	30
Div with uncached memory	164	306	353

Findings - Speculation Window Size

Conditional Branches

Conditional branch	Broadwell	Skylake	Zen
Register access	14	16	7
Access to cached memory	19	17	9
Access to uncached memory	144	280	321
Mul with register	19	19	2
Mul with cached memory	33	33	8
Mul with uncached memory	154	290	362
Div with register	35	41	17
Div with cached memory	34	39	30
Div with uncached memory	164	306	353

Findings - Speculation Window Size

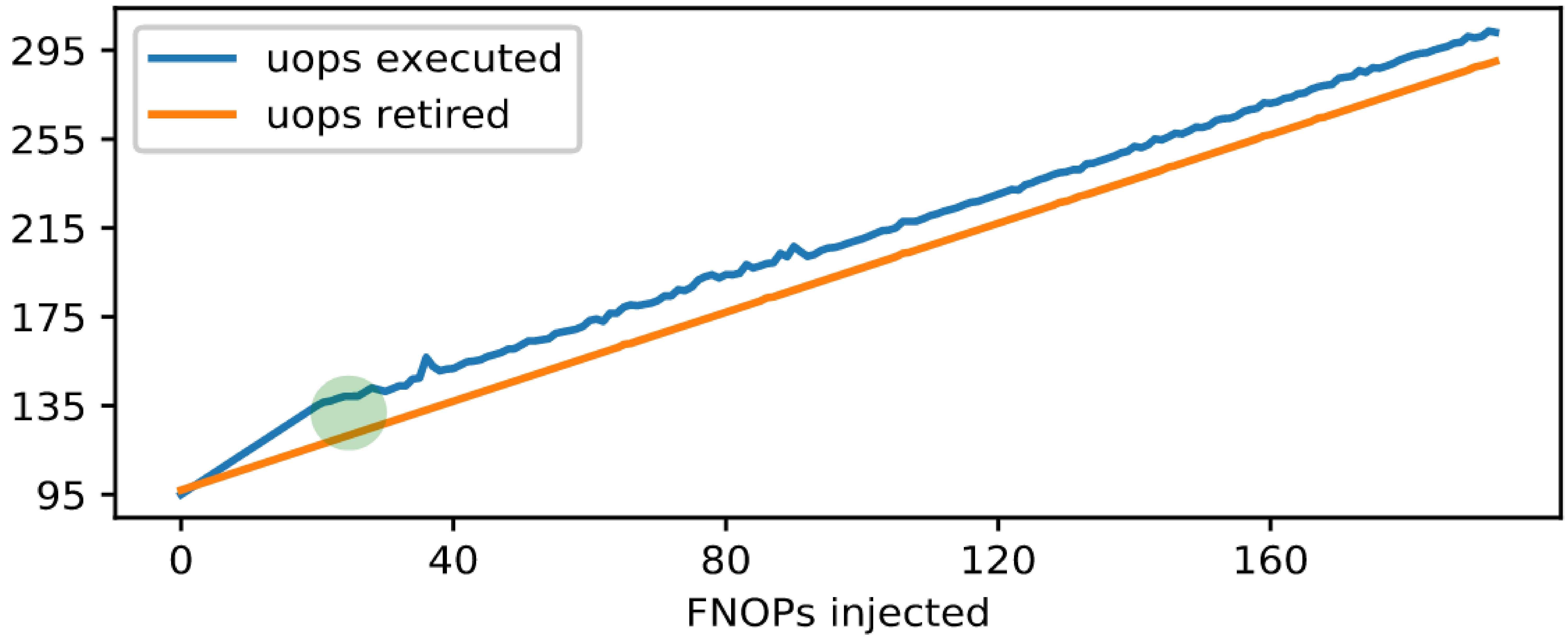
Indirect Control Flow Transfer

Indirect branch target location	Broadwell	Skylake	Zen
Register	28	22	24
Cached memory	41	34	35
Uncached memory	154	303	301

Findings - Speculation Window Size

Store to Load Forward

Avg: 15 μ-ops
Max: 23 μ-ops
55 CPU cycles



Findings - MPX

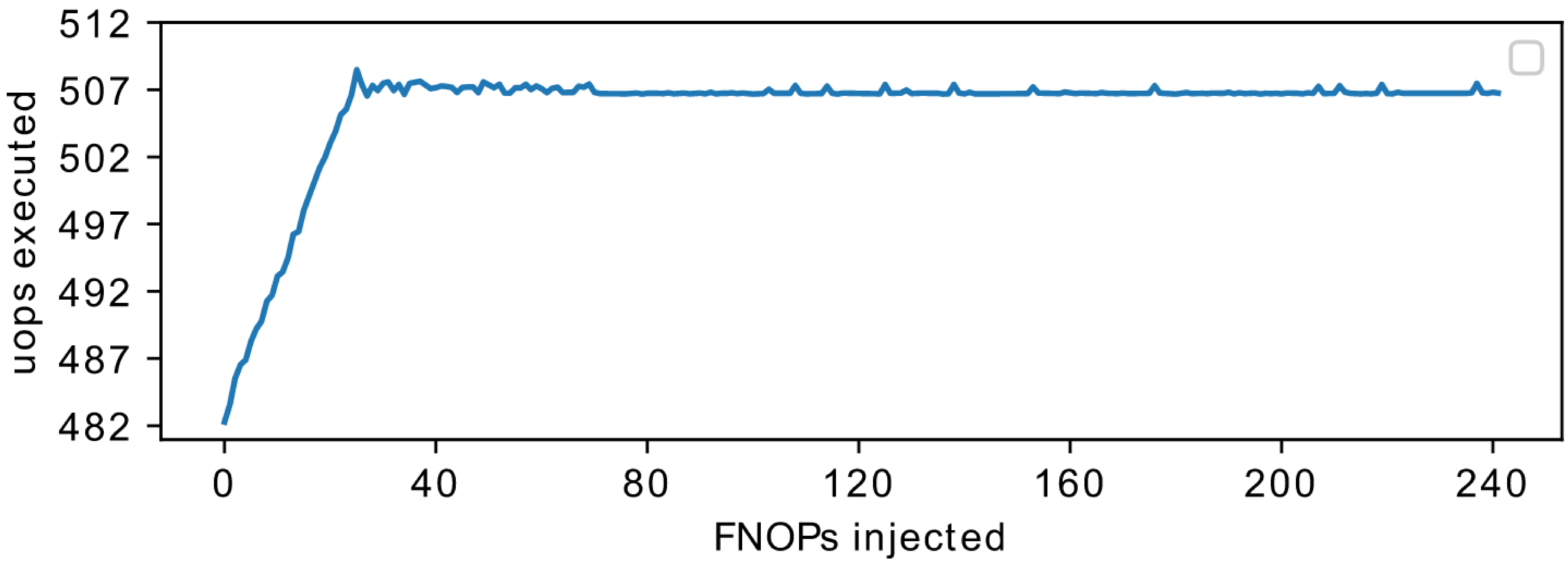
Setup:

10 iterations with correct bound check

Then fail on *bndcu* instruction

Using NOP sled we can speculative execute 122 instructions after bound check violation

Findings - MPX



Result: 22 FNOP instructions speculation window

Findings - Executable Page Permission

Is the NX bit lazily evaluated as access permission in Meltdown?

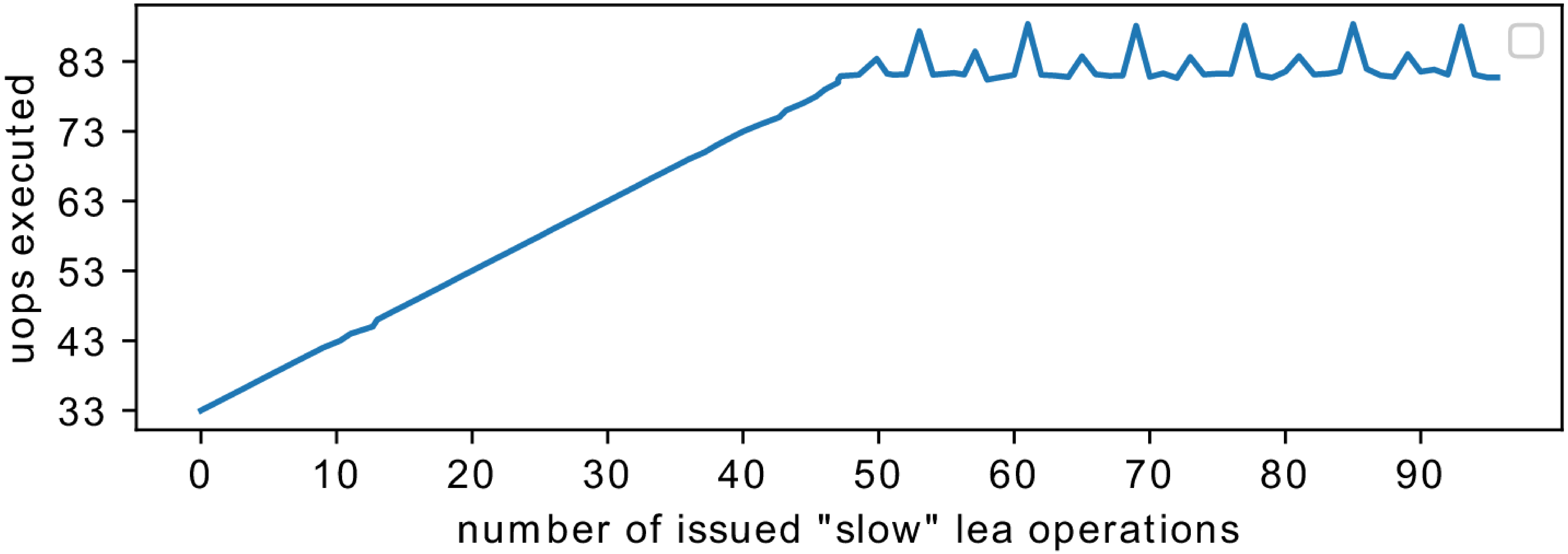
We load a memory area we control inside TLB and cache

We speculatively execute a control transfer to such area

Result: The execute page table permission bit is honored

Findings - Issued vs Executed

Are issued µ-ops measured by the markers really executed?



Findings - Speculation across system calls

Traced very small syscall (sys_getppid, ~47 instructions)

User-mode μ-ops count does not variate with more instructions after call

Kernel-mode μ-ops count does not variate between speculated and non speculated execution

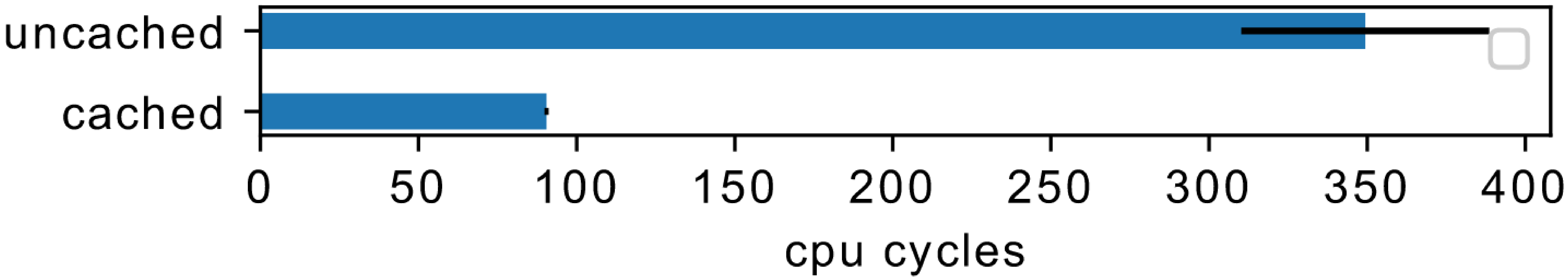
Conclusion: system calls stop speculation

Findings - Flushing the cache

Findings - Flushing the cache

```
    setup
.loop:
    clflush[counter]
    clflush[var]
    lfence

    mov eax, DWORD[var]     ;cached version
    lfence                  ;only

    start_counter

    cmp 12, DWORD[counter]
    je .else

    clflush[var]
    lfence

.else:
    mov eax, DWORD[var]     ;final load
    lfence

    stop_counter

    inc DWORD[counter]
    cmp DWORD[counter], 13
    jl loop
```

Findings - Flushing the cache



Conclusions:

CLFLUSH does not affect the cache until it retires

CLFLUSH must be paired with speculation blocker (e.g LFENCE) to be sure it has the intended effect

Performance counters (INTEL)

3 fixed counters

4 programmable counters **with SMT**

8 programmable counters **without SMT**

Plenty of different counters available for front-end or back-end of the CPU

Performance counters (INTEL)

3 fixed counters

4 programmable counters **with SMT**

8 programmable counters **without SMT**

Plenty of different counters available for front-end or back-end of the CPU