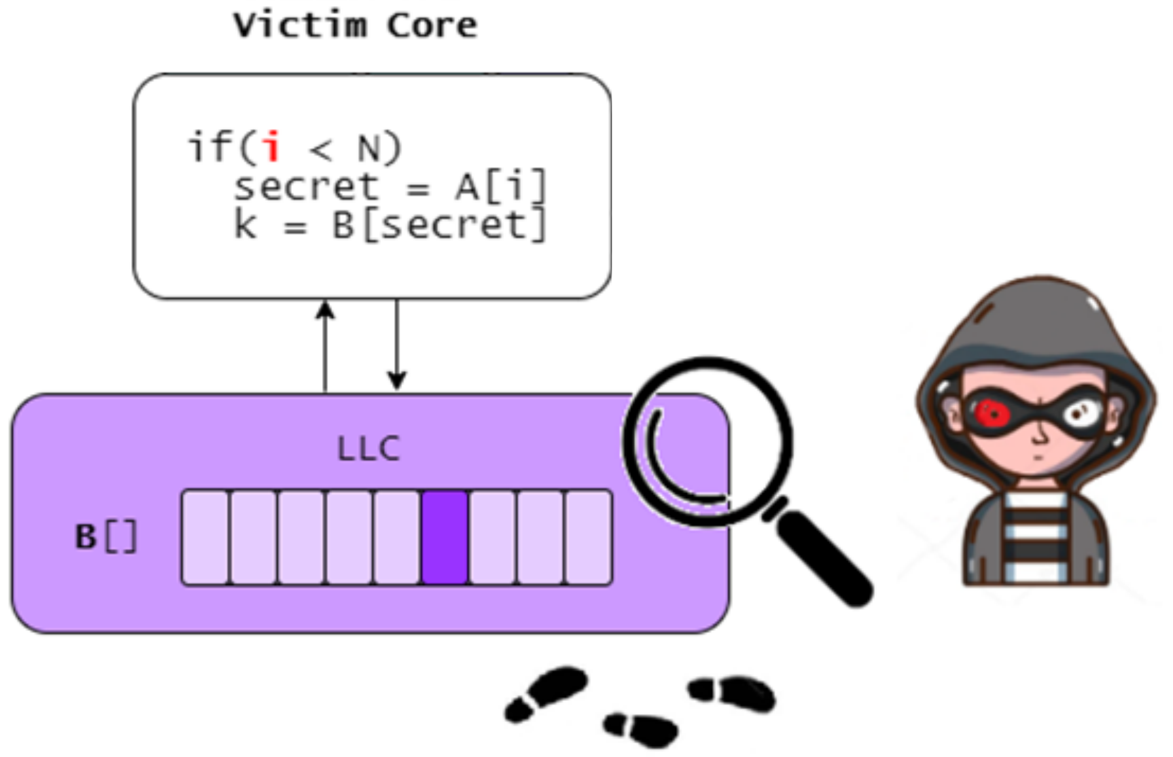
Speculative Interference Attacks: Breaking Invisible Speculation Schemes

Mohammad Behnia, [1] Prateek Sahu, Riccardo Paccagnella, Jiyong Yu, Zirui Zhao, [2] Xiang Zou, [2] Thomas Unterluggauer, Josep Torrellas, [2] Carlos Rozas, [3] Adam Morrison, [2] Frank Mckeen, [2] Fangfei Liu, [4] Ron Gabor, Christopher W. Fletcher, [2] Abhishek Basak, [5] Alaa Alameldeen

University of Illinois at Urbana-Champaign, [1] University of Texas at Austin, [2] Intel Corporation, [3] Tel Aviv University, [4] Toga Networks, [5] Simon Fraser University

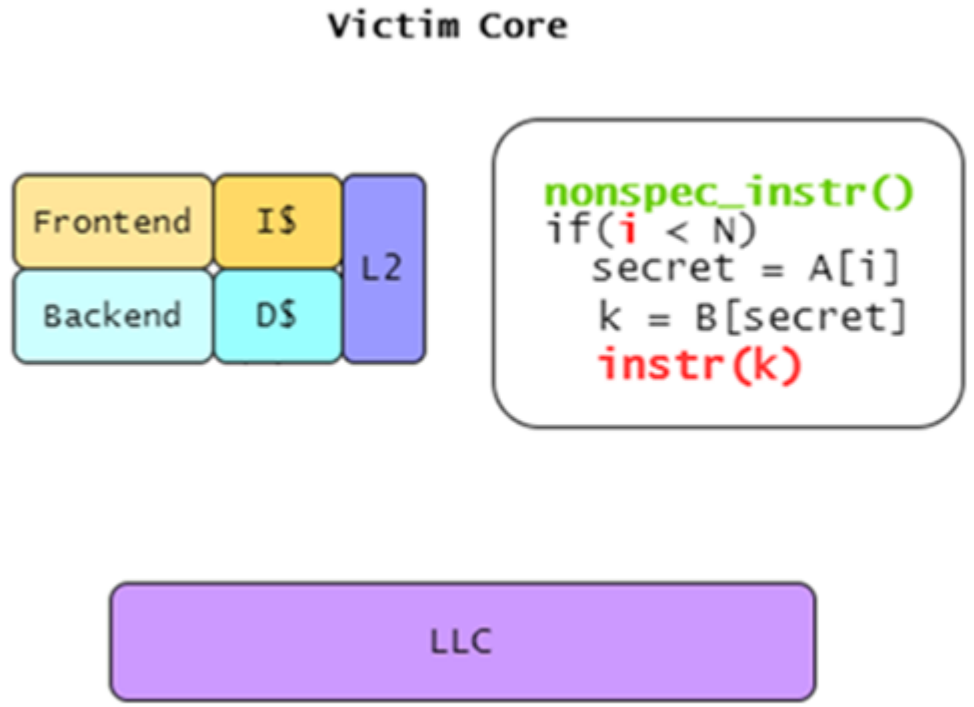
Introduction

- Microarchitectural Side Channels
  - Cache-based
- Spectre Attack
  - Variant 1
- Advantages to Attacker
  - Persistent State Change
  - Shared Cache Hierarchy



Invisible Speculation Schemes

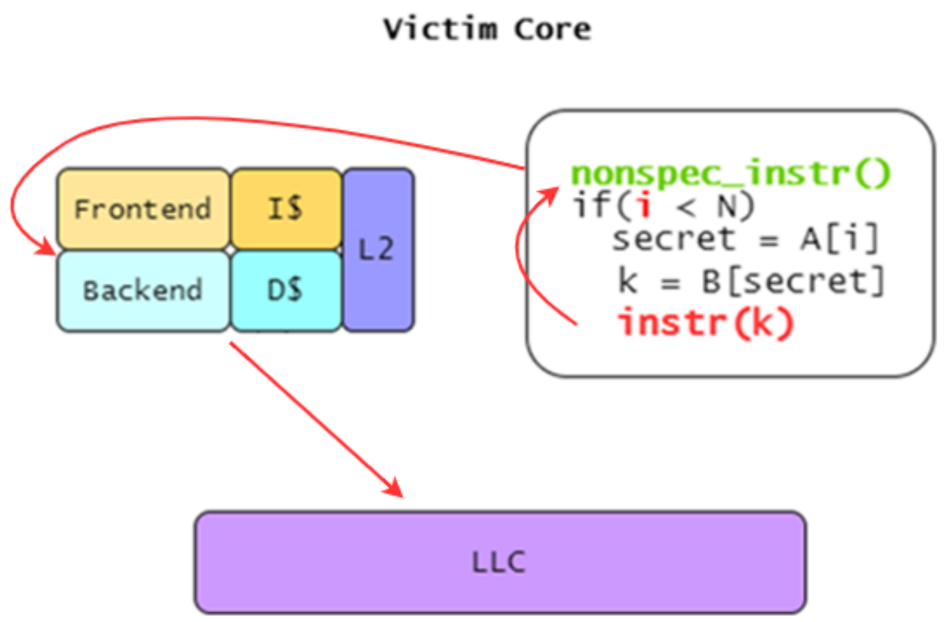
- Invisible Speculation Schemes
  - Mechanisms to thwart speculative, persistent cache state changes
- Example: Delay-On-Miss
  - Any cache state change is deferred until load becomes non-speculative
  - Loads that hit in the L1 forward results to dependent instructions



```
nonspec_instr()
if(i < N)
  secret = A[i]
  k = B[secret]
  instr(k)
```

Speculative Interference Attacks

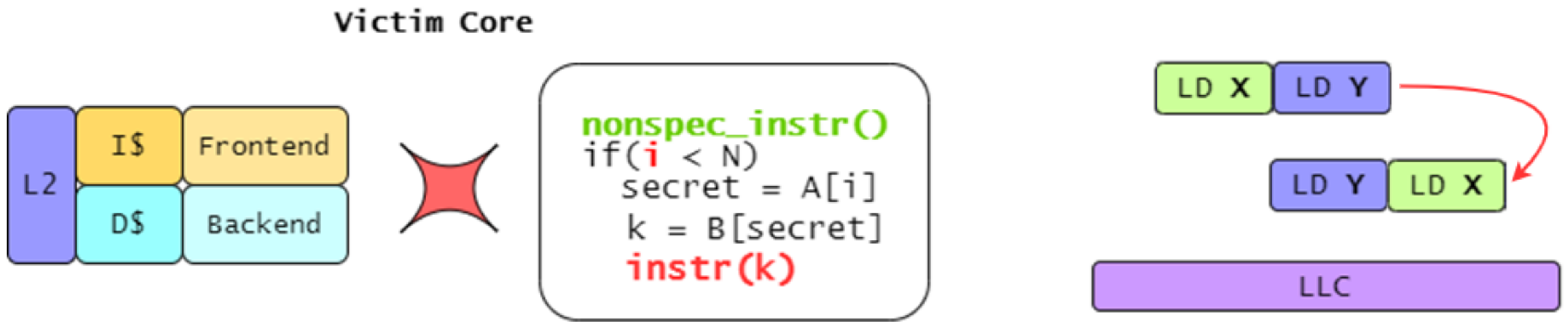
- **Observation:** Secret-Dependent timing effects can be monitored indirectly by how they interact with **older non-speculative instructions**
- **Idea:** By creating a "ripple effect" we can transform transient interactions into persistent state changes in the cache even with invisible speculation enabled



```
nonspec_instr()
if(i < N)
  secret = A[i]
  k = B[secret]
  instr(k)
```

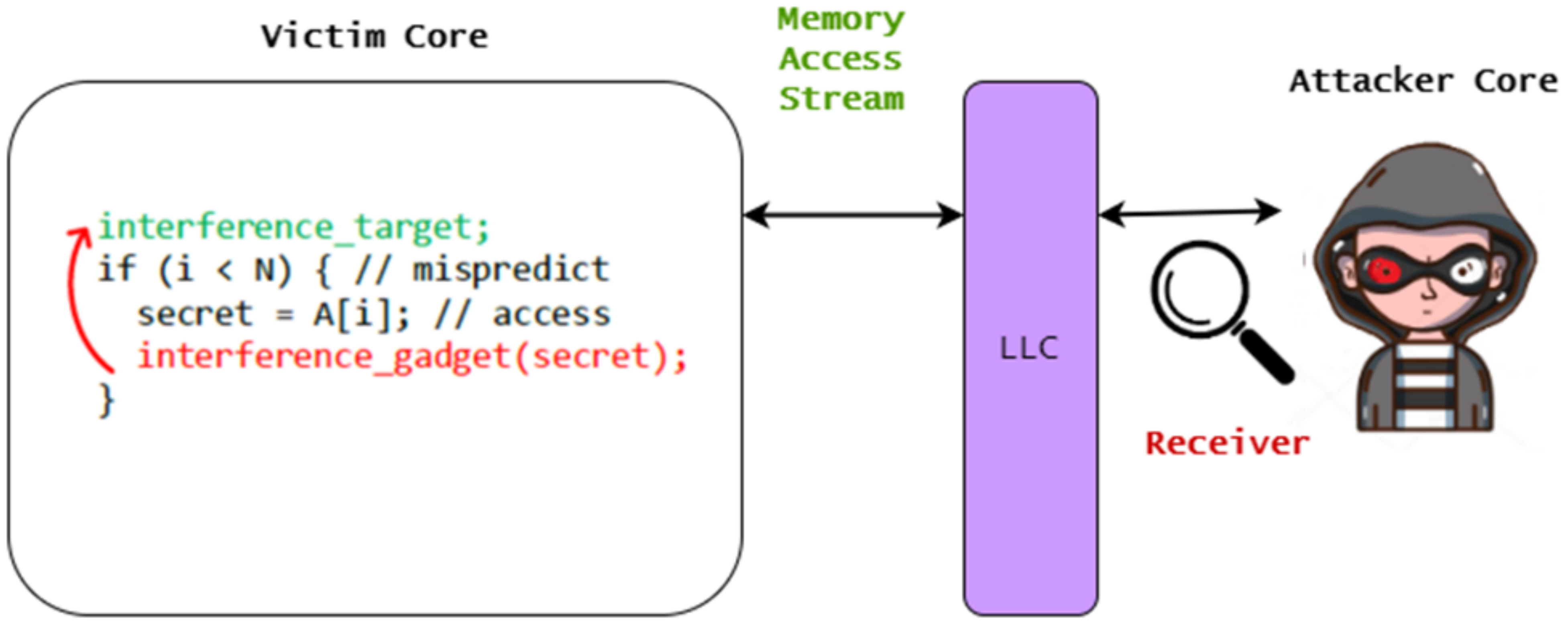
Speculative Interference Attacks

- Can induce contention on a large number of microarchitectural resources using different instructions
- If this “ripple effect” targets non-speculative **memory accesses** it can affect their ordering



```
nonspec_instr()
if(i < N)
  secret = A[i]
  k = B[secret]
  instr(k)
```

Attack Framework



```
interference_target;
if (i < N) { // mispredict
  secret = A[i]; // access
  interference_gadget(secret);
}
```

Story of this Paper

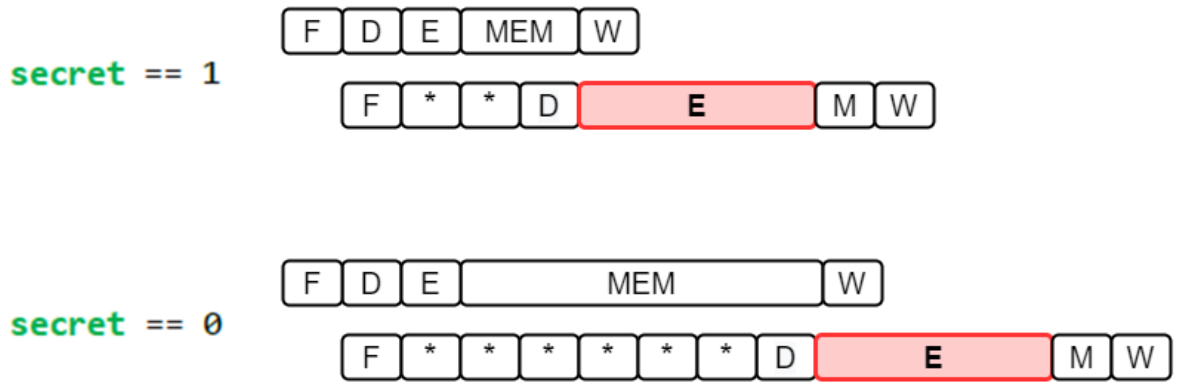
- Speculative Interference Attacks undermine the security of a prominent family of Hardware Spectre Defenses
- **1.** Mis-speculated younger instructions can affect the timing of older bound-to-retire instructions including memory operations
- **2.** Altering timing of memory operations can change the order of one memory operation relative to others and expose secrets via persistent changes to cache state

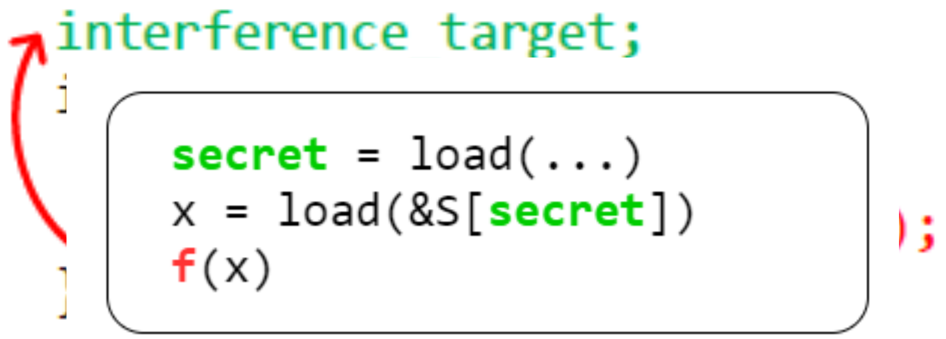
Outline

- Attack Variants
- D-Cache PoC
- Defenses

Interference Gadgets

- **Type 2**: Secret-dependent interference time





```
secret = load(...)
x = load(&S[secret])
f(x)
```

(a) Attack framework

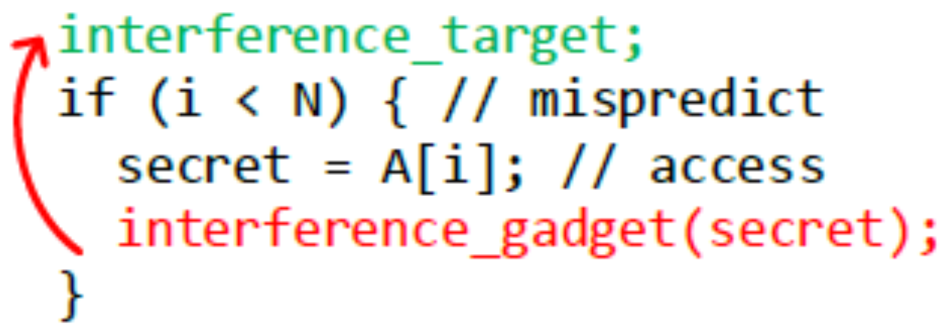
Interference Gadgets

- **Type 1**: Operand-dependent resource usage patterns

```
secret = load(...)
f(secret)
```

- **Type 3**: Interference existence is secret-dependent

```
secret = load(...)
if(secret)
   f()
```

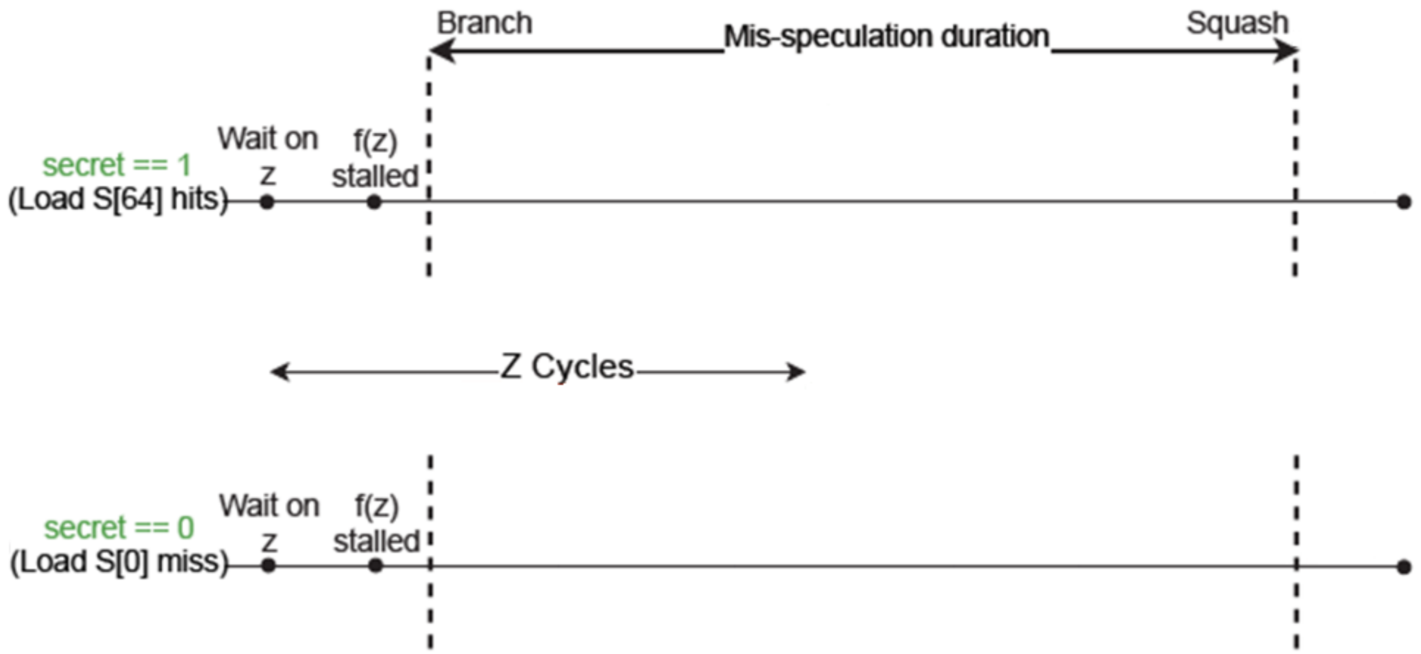


```
interference_target;
if (i < N) { // mispredict
  secret = A[i]; // access
  interference_gadget(secret);
}
```

(a) Attack framework

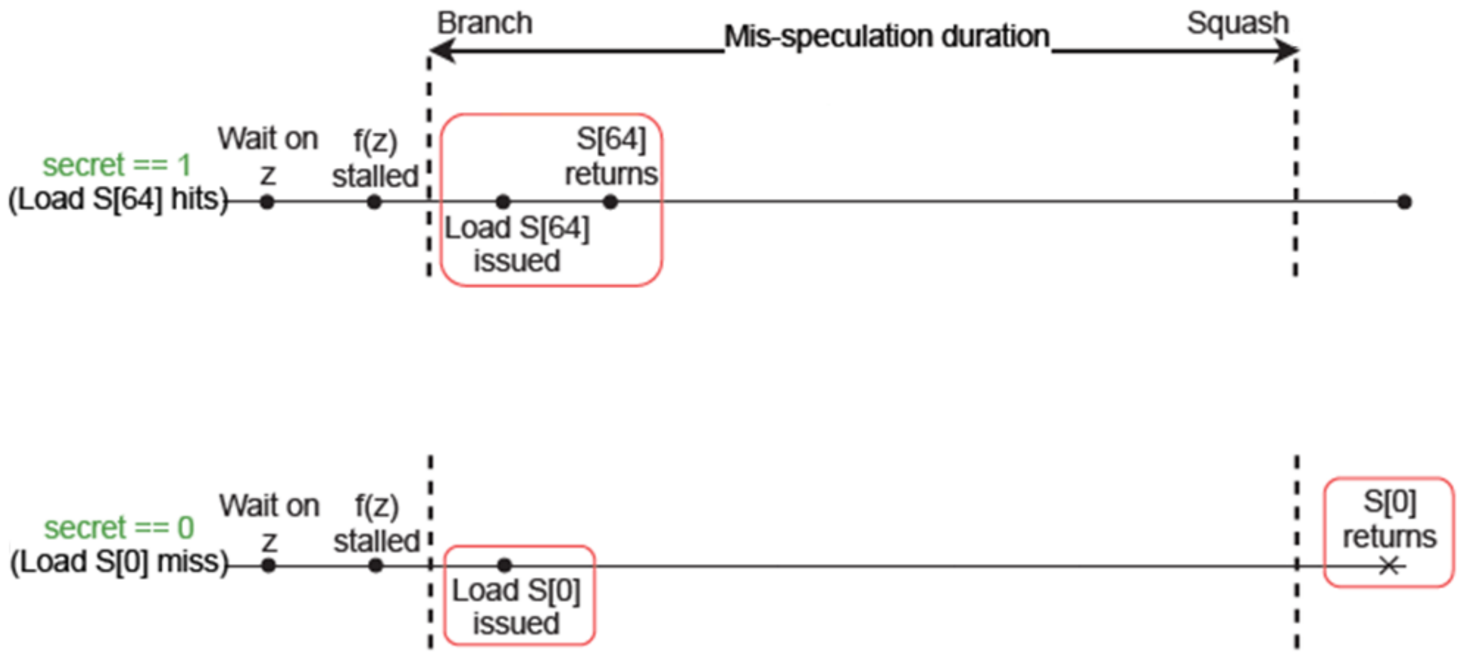
Gadget + Target

```
z = ...
A = f(z)
y = load(A)
if (i<N):
    secret = load(&TArray[i])
    x = load(&S[secret*64])
    f'(x)
```



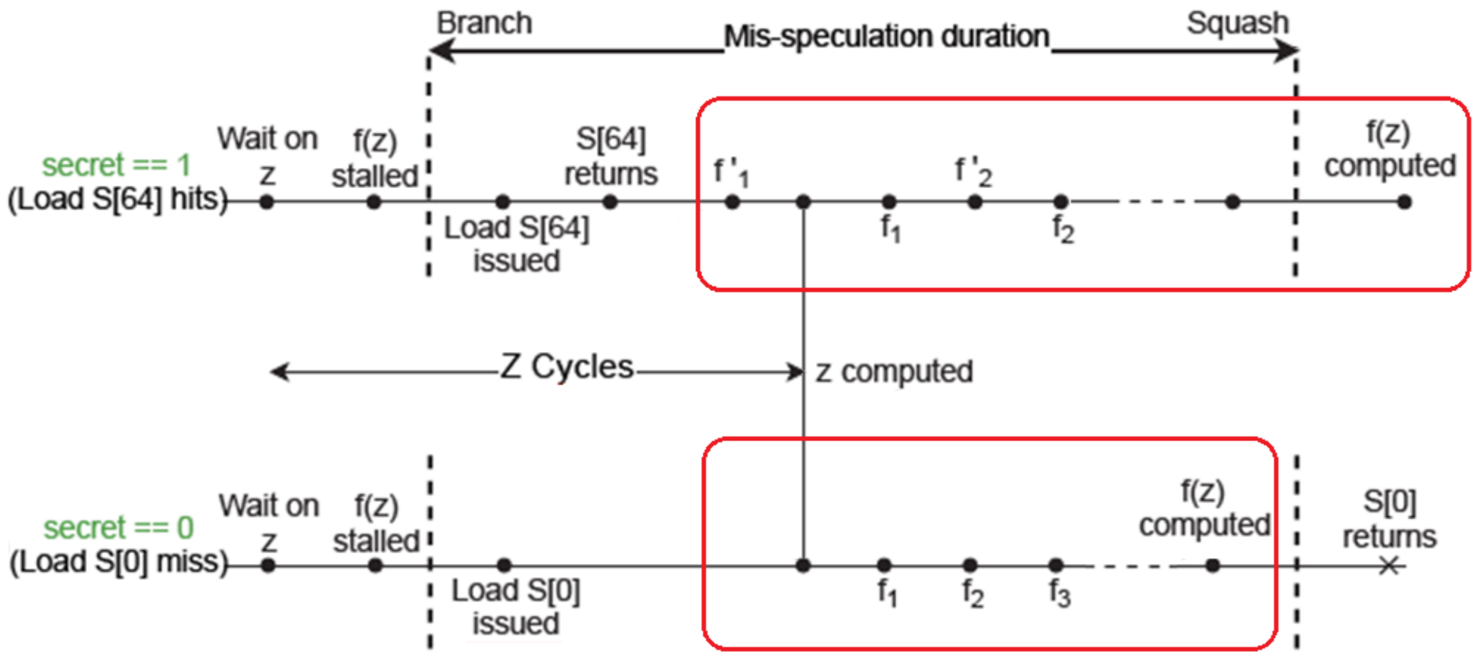
Gadget + Target

```
z = ...
A = f(z)
y = load(A)
if (i<N):
    secret = load(&TArray[i])
    x = load(&S[secret*64])
    f'(x)
```



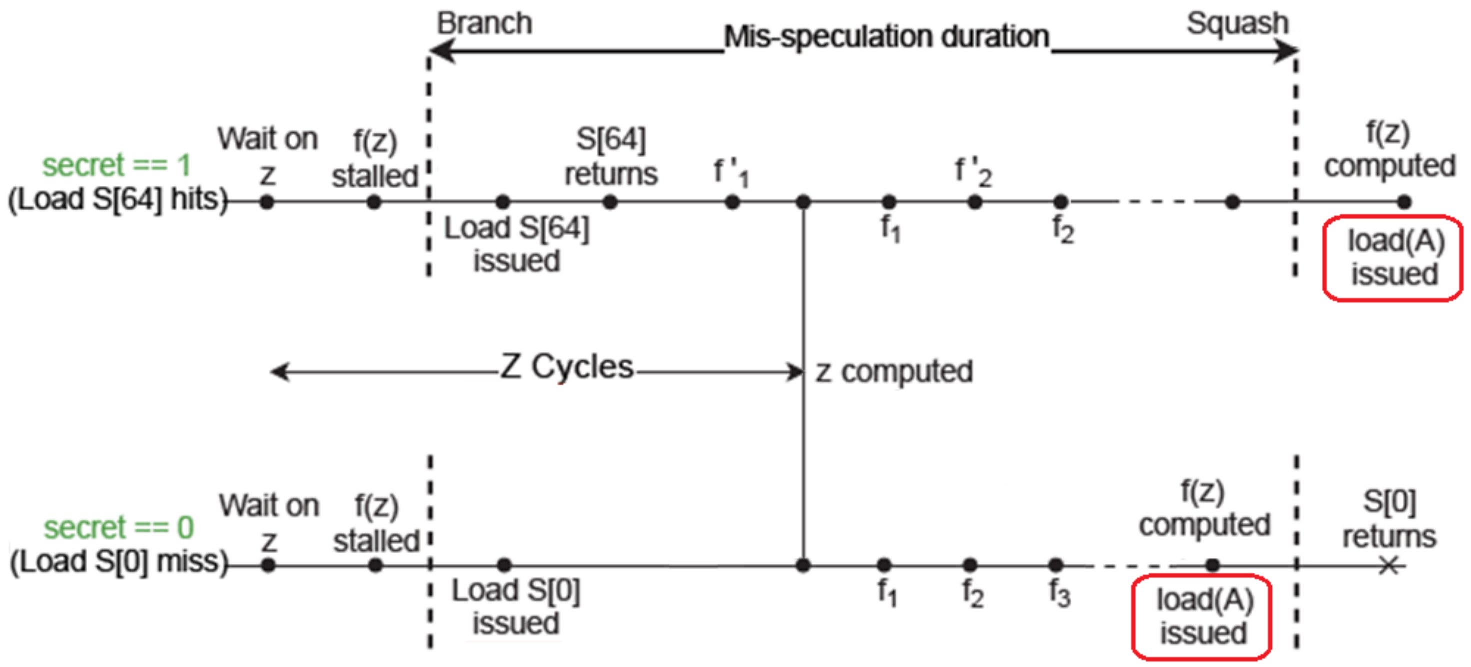
Gadget + Target

```
z = ...
A = **f(z)**
y = load(A)
if (i<N):
    secret = load(&TArray[i])
    x = load(&S[secret*64])
    **f'(x)**
```



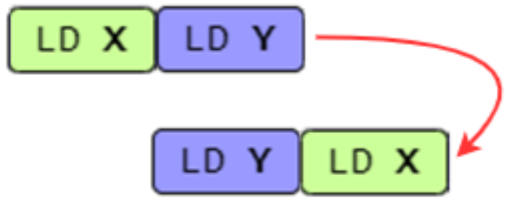
Gadget + Target

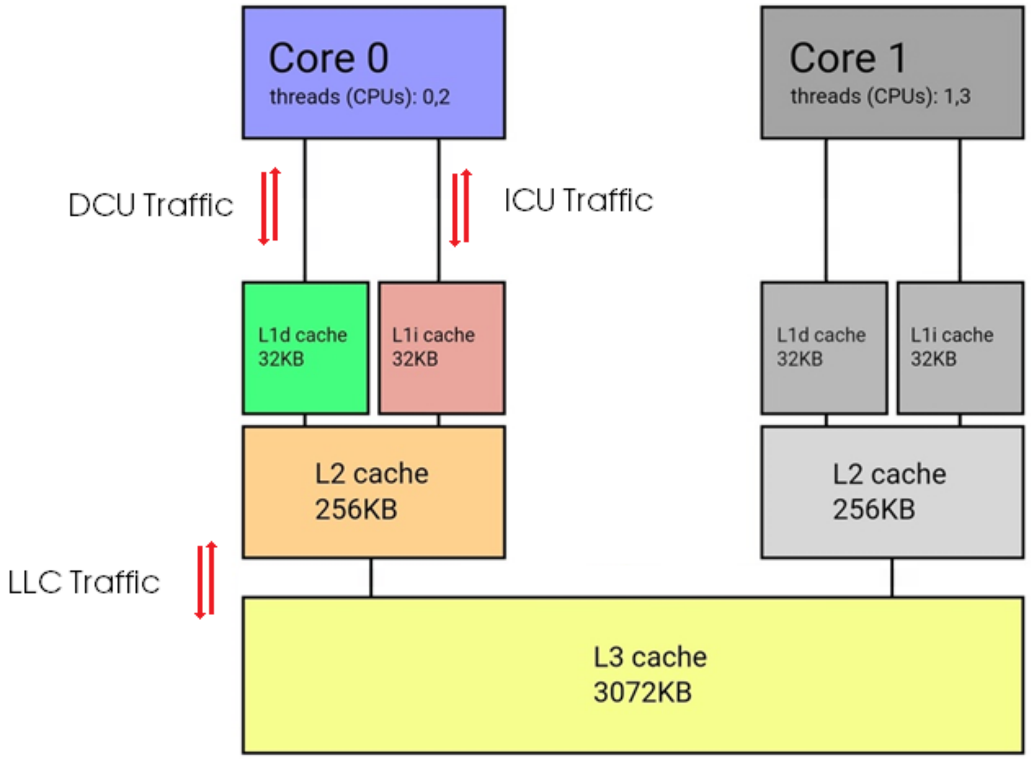
```
z = ...
A = f(z)
y = load(A)
if (i<N):
    secret = load(&TArray[i])
    x = load(&S[secret*64])
    f'(x)
```



Interference Targets

- Victim L1 D-cache and L1 I-cache access streams
- Can also manifest in permutations of D-cache and I-cache memory access patterns





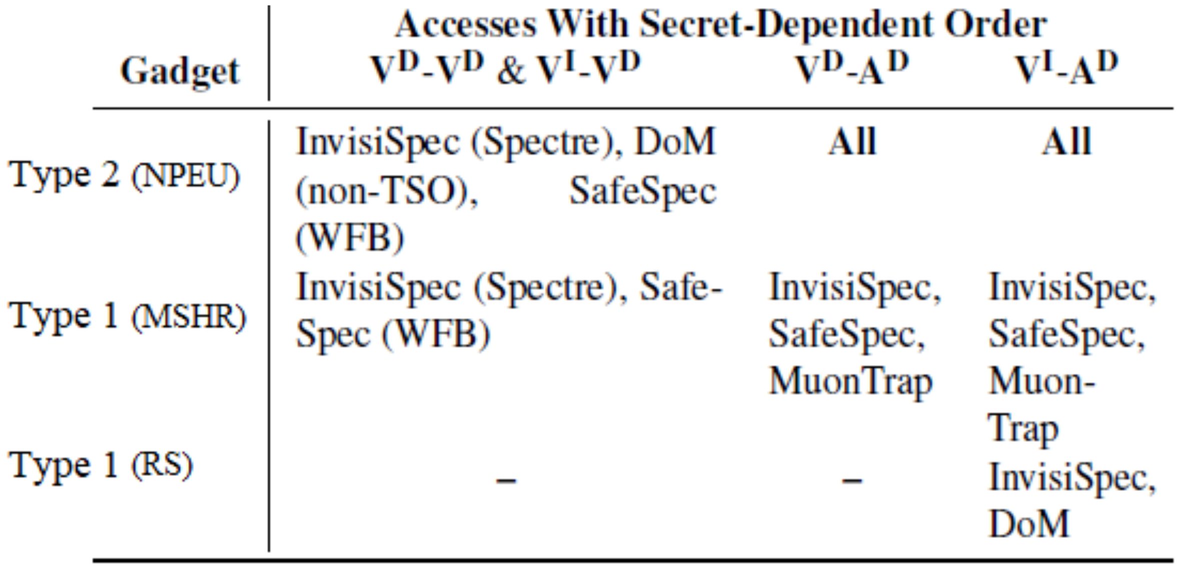
Vulnerability Matrix

Target Variant	Reference Load
$V \uparrow D - V \uparrow D$	$V \uparrow D$
$V \uparrow D - V \uparrow I$	$V \uparrow D$
$V \uparrow D - A \uparrow D$	$A \uparrow D$
$V \uparrow I - A \uparrow D$	$A \uparrow D$

$V \uparrow D$: Victim Data Access

$V \uparrow I$: Victim Instruction Fetch

$A \uparrow D$: Attacker Data Access



Gadget	Target		
	Accesses With Secret-Dependent Order		
Gadget	V^D-V^D & V^I-V^D	V^D-A^D	V^I-A^D
Type 2 (NPEU)	InvisiSpec (Spectre), DoM (non-TSO), SafeSpec (WFB)	All	All
Type 1 (MSHR)	InvisiSpec (Spectre), SafeSpec (WFB)	InvisiSpec, SafeSpec, MuonTrap	InvisiSpec, SafeSpec, Muon-Trap
Type 1 (RS)	–	–	InvisiSpec, DoM

D-Cache PoC

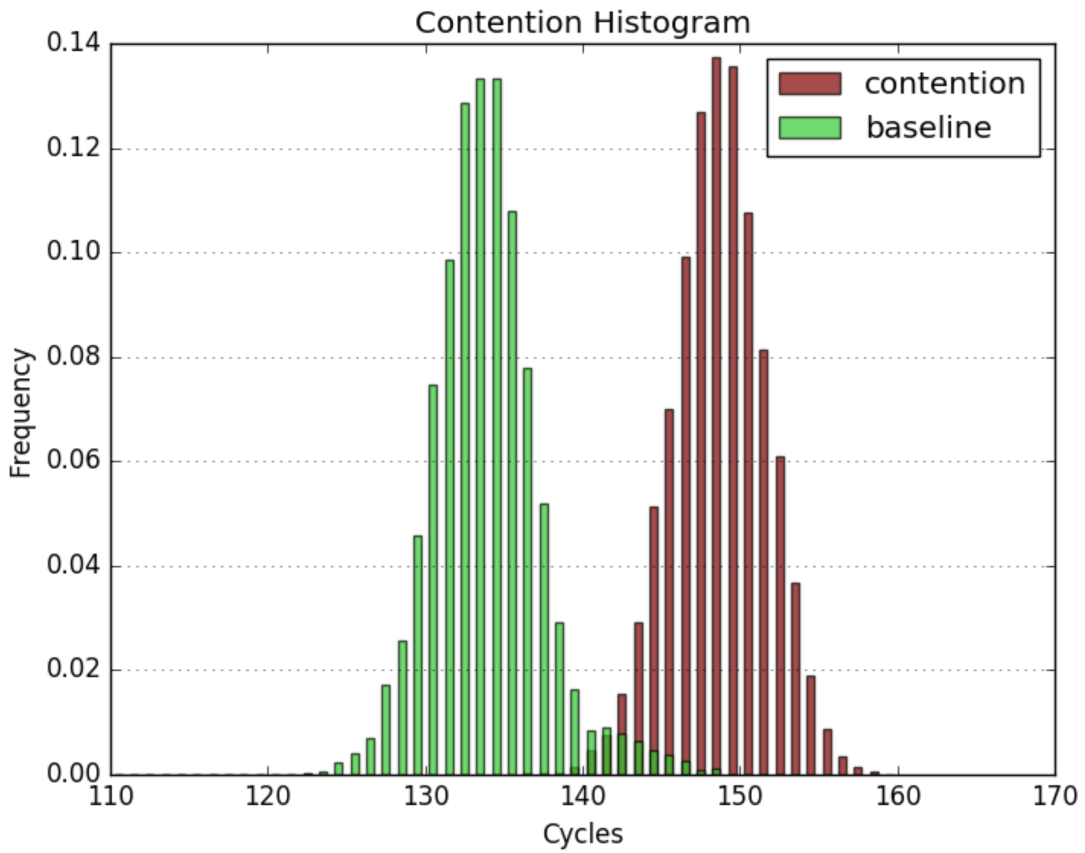
- Victim and Attacker Threads on Separate Cores
- Shared memory addresses A and B that map to same LLC set and slice
- Victim issues A-B or B-A using secret dependent load ordering
- Attacker primes and probes replacement policy state of LLC set to identify issue order

```
z = ...
A = f(z)
y = load(A)
if (i<N):
    secret = load(&TArray[i])
    x = load(&S[secret*64])
    f'(x)
```

D-Cache PoC Interference Gadget

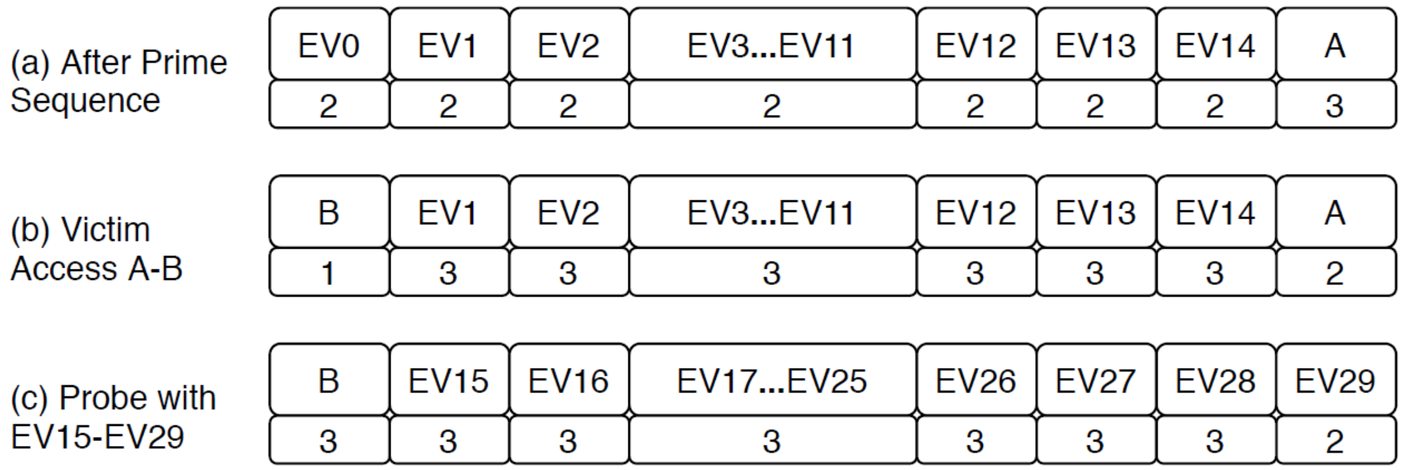
```
+-> A = load(interference_target())
|             // VSQRTPD dependency chain
|   if (...)  //miss-speculation
|       secret = load(...);
|       x = load(&S[secret]); //hit-miss
+---    interference_gadget(x);
        // VSQRTPD ready to execute
```

VSQRTPD
1 micro-op execution port 0
Latency of 15–16 cycles
Reciprocal throughput of 9–12 cycles



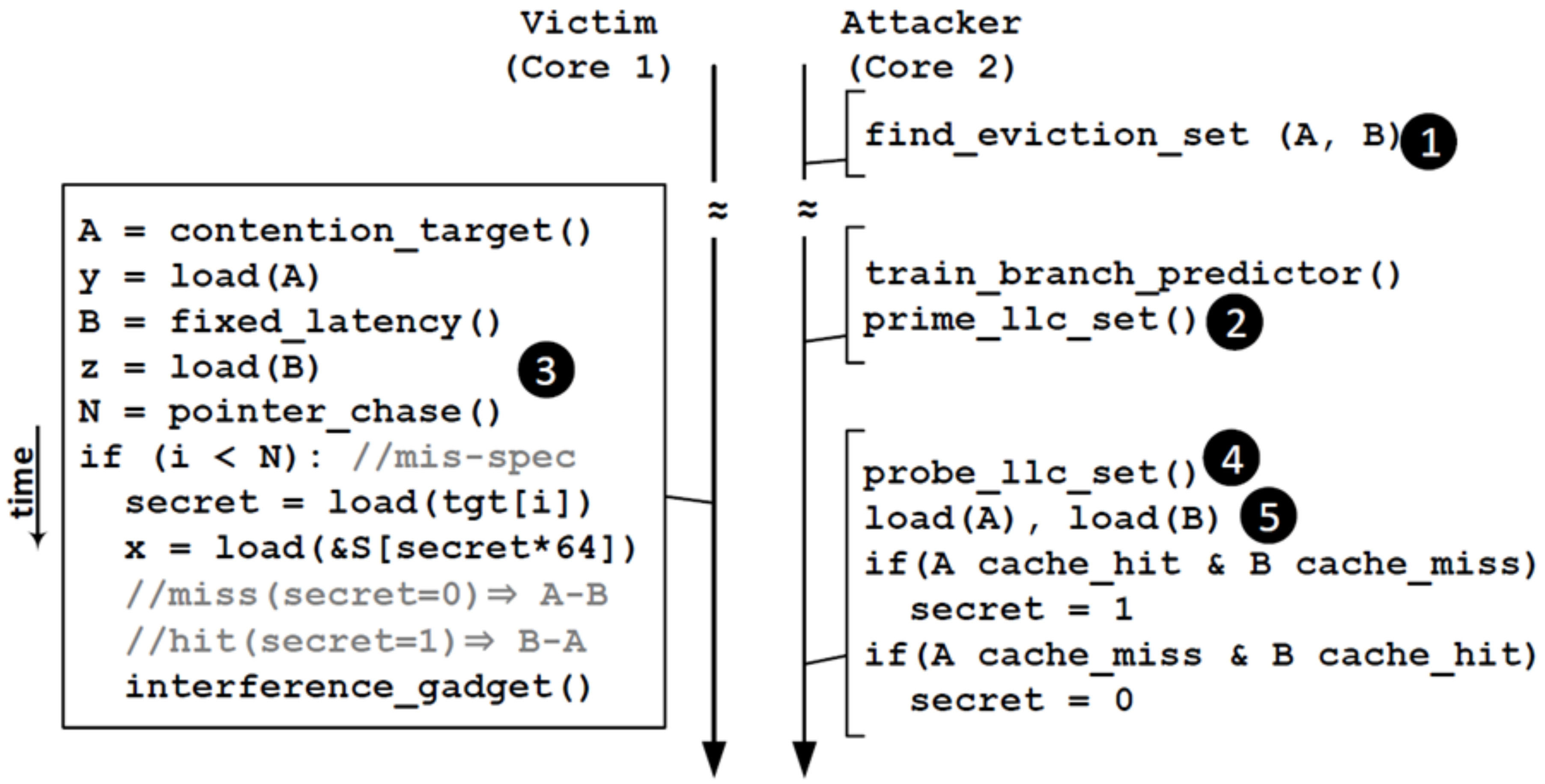
D-Cache PoC Receiver Protocol

- Quad Age LRU Replacement Policy
  - QLRU_H11_M1_R0_U0



(a) After Prime Sequence	EV0	EV1	EV2	EV3...EV11	EV12	EV13	EV14	A
	2	2	2	2	2	2	2	3
(b) Victim Access A-B	B	EV1	EV2	EV3...EV11	EV12	EV13	EV14	A
	1	3	3	3	3	3	3	2
(c) Probe with EV15-EV29	B	EV15	EV16	EV17...EV25	EV26	EV27	EV28	EV29
	3	3	3	3	3	3	3	2

D-Cache PoC End-to-End



```
find_eviction_set (A, B)

train_branch_predictor()
prime_llc_set()

A = contention_target()
y = load(A)
B = fixed_latency()
z = load(B)
N = pointer_chase()
if (i < N): //mis-spec
  secret = load(tgt[i])
  x = load(&S[secret*64])
  //miss(secret=0)⇒ A-B
  //hit(secret=1)⇒ B-A
  interference_gadget()

probe_llc_set()
load(A), load(B)
if(A cache_hit & B cache_miss)
  secret = 1
if(A cache_miss & B cache_hit)
  secret = 0
```

D-Cache PoC Bitrate

- Intel Core i7-7700 Kaby Lake CPU with 4 physical cores @ 3.6GHz
- Unified Reservation Station, 8 execution ports
- POC Attacker and Victim Threads run in multi-core configuration

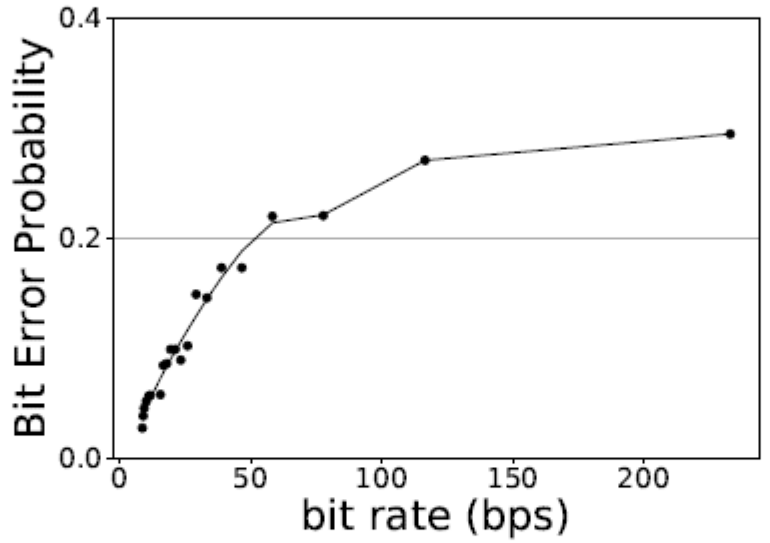


Figure 10: D-Cache PoC channel error vs. bit rate.

Discussion of Defenses

- Ideal Invisible Speculation: LLC access pattern being invariant of speculation
- Basic Defense: Fences to prevent issue of ROB instructions until window becomes non-speculative
- More advanced Defense:
  - Not Delaying Older Instructions:
    - Priority Tagging based on speculative window in RS
    - Scheduler to predict speculative interference
  - Not Releasing Resources Early:
    - Operand independent executions times

Thank You