30th International Symposium on High-Performance Computer Architecture (HPCA 2024)

GadgetSpinner: A New Transient Execution Primitive using the Loop Stream Detector

Yun Chen* Ali Hajiabadi* Trevor E. Carlson

National University of Singapore

NUS National University of Singapore

* Equal contribution first authors

GadgetSpinner: Overview

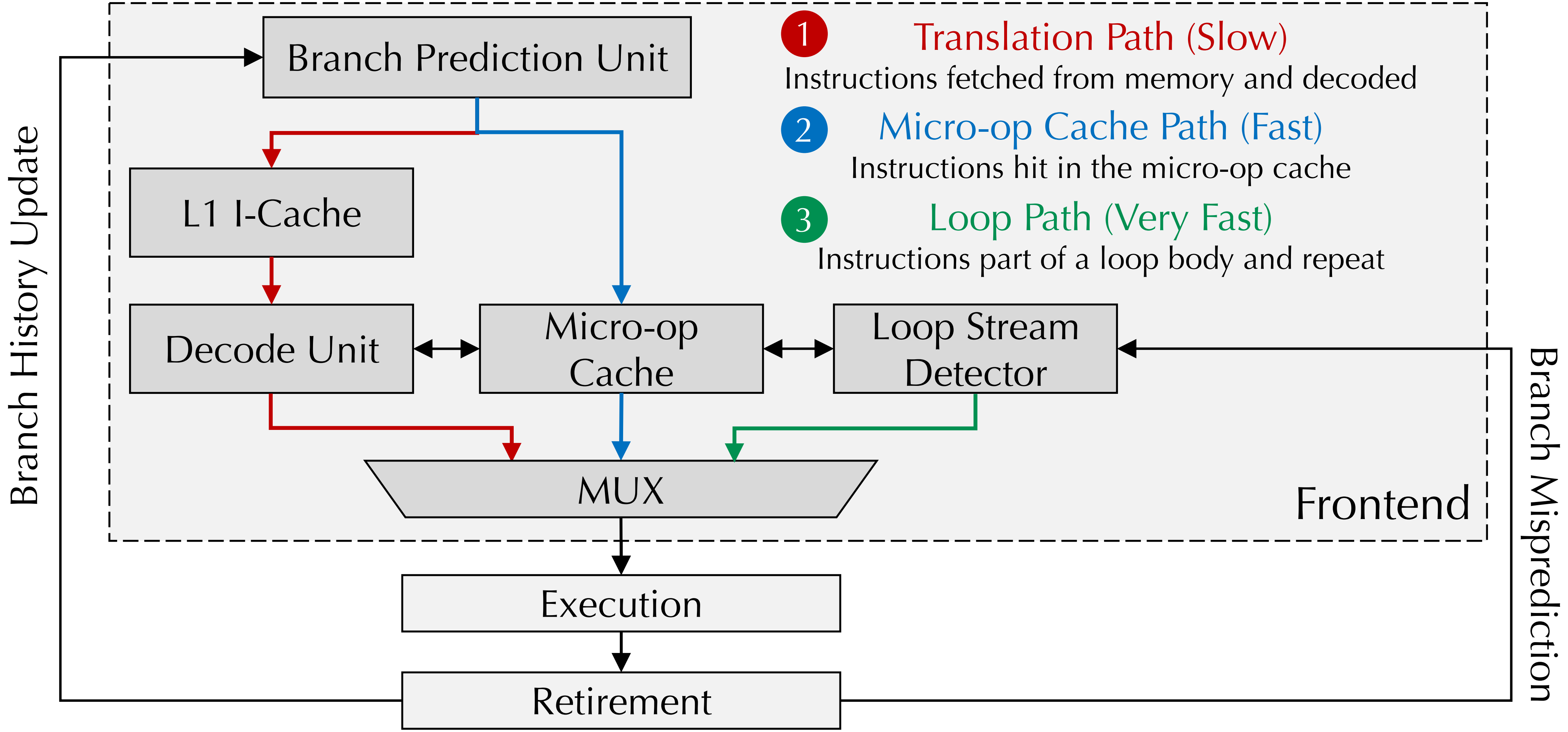
- Introducing a new transient execution primitive: **Loop Stream Detector**

- Proposing **GadgetSpinner**; an attack methodology based on the LSD

  - Bypassing secure Branch Prediction Unit (BPU) designs

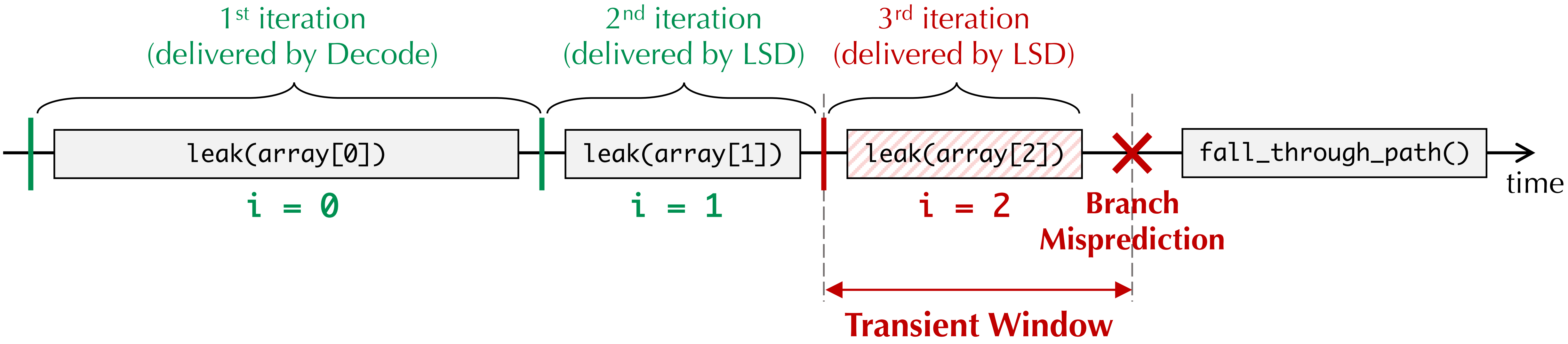
  - Providing more practical cross-core transient execution attacks

Frontend in Intel x86 CPUs



LSD: Operation and Transient Window

```
int array[2];
for (i = 0; i < 2; i++)
    leak(array[i]);
fall_through_path();
```



Out-of-loop-bounds Access: leaking array[2]

Spectre-v1 Leaks via PHT Speculation Primitive

Attacker (Core-0)

1

```
    void mistrain() {
        for (j=0; j<1000; j++) {
BR1         if (true) //do sth
        }
    }
```

Victim (Core-0)

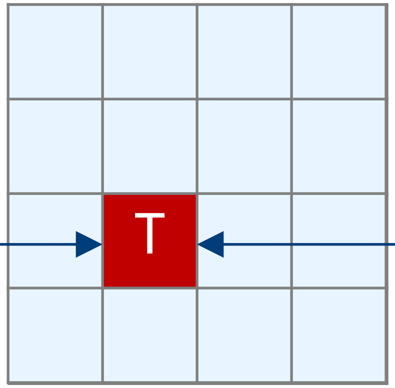
2

```
    int array[2];
    void victim (int index) {
BR2     if (index < 2)
            leak(array[index]);
    }
```

Out-of-bounds access

index≥2

Pattern History Table
(Core-0)



Key Requirement: Sharing the PHT (i.e., co-location on the same core)

Examples of BPU-based mitigations:

BPU Partitioning	BPU Flushing/Randomization	BPU Encryption
e.g., Half&Half [S&P'23]	e.g., HyBP [HPCA'22]	e.g., STBPU [DSN'22]

GadgetSpinner Leaks via LSD Speculation Primitive

Attacker (Core-0)

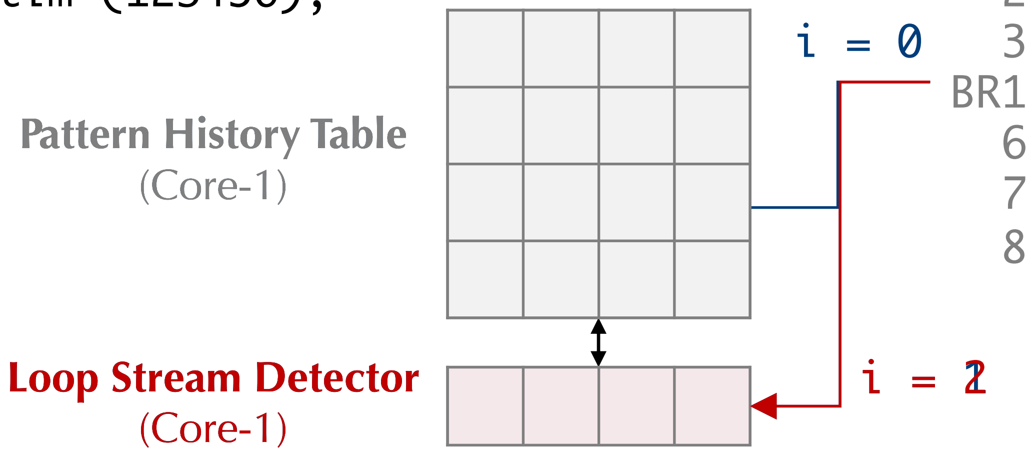
```
//no mistraining
victim (123456);
```

Victim (Core-1)

```
int array[2];
void victim (int offset) {
    for (i = 0; i < 2; i++){
        index = ((2 ^ i) - 1) & offset;
        leak(array[index]);
    }
}
```

Out-of-bounds access

$$\text{index} = \begin{cases} 0 & i<2 \\ \text{offset} & i \geq 2 \end{cases}$$



BPU-based mitigations are not effective because BPU is disabled during LSD operations

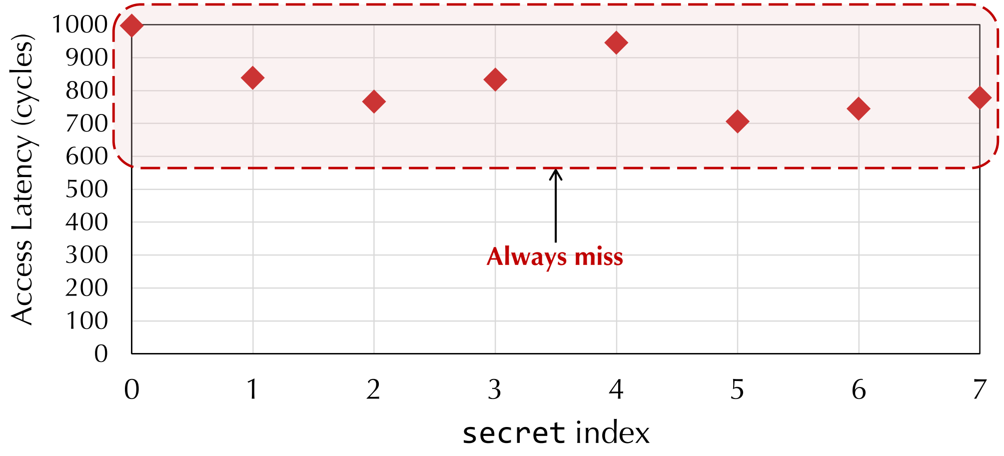
BPU Partitioning	BPU Flushing/Randomization	BPU Encryption
e.g., Half&Half [S&P'23]	e.g., HyBP [HPCA'22]	e.g., STBPU [DSN'22]

Loop Stream Detector Out-of-Loop-Bounds Access

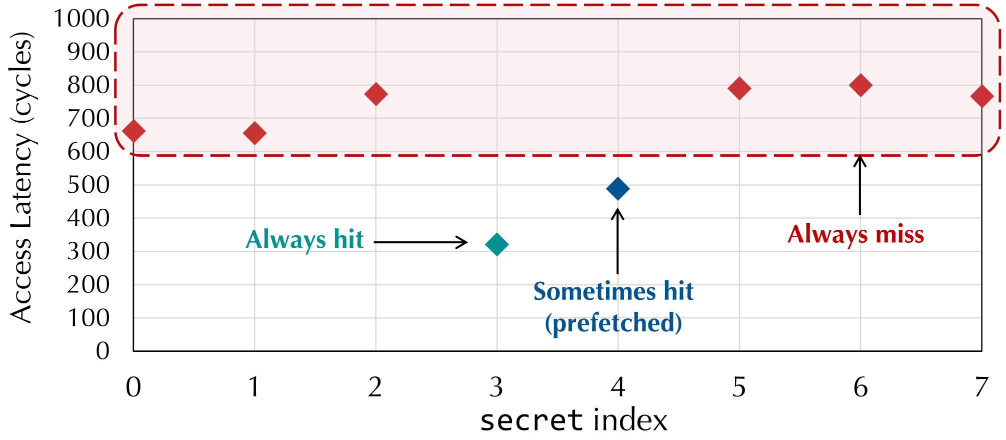
```c
uint8_t A[8];

void victim (uint64_t offset){
    uint64_t index = 0;
    for (int i = 0; i < 8; i++) {
        temp = A[index];
        index = ((8 ^ i) - 1) & offset;
    }
}
```

$$\text{index} = \begin{cases} 0 & i<8 \\ \text{offset} & i\geq 8 \end{cases}$$



(1) Expected behavior (LSD disabled)



(2) Illegal out-of-loop-bounds access (LSD enabled)

How to trigger LSD speculation primitive?

1. Loop body smaller than 64 micro-ops (size of the LSD)
2. All micro-ops in loop body align with micro-op cache lines
3. Absence of hard-to-predict branches in loop body

LSD Qualified Loops

(a) Aligned loop body

PC	Instruction	Assembly	
1220	74 35	je	L1
1222	ba 00 00 00 00	mov	0x0,edx
1227	b9 00 00 00 00	mov	0x0,ecx
122c	48 01 f1	add	rsi,rcx
122f	48 89 01	mov	rax,(rcx)
1232	83 c2 01	add	0x1,edx
1235	89 d1	mov	edx,ecx
1237	83 f1 08	xor	0x8,ecx
123a	0f b6 c9	movzbl	cl,ecx
123d	83 e9 01	sub	0x1,ecx
1240	48 63 c9	movslq	ecx,rcx
1243	4c 21 c1	and	r8,rcx
1246	48 89 d0	mov	rdx,rax
1249	48 c1 e0 0c	shl	0xc,rax
124d	25 00 f0 0f 00	and	0xff000,eax
1252	38 14 07	cmp	dl,(rdi,rax,1)
1255	77 d5	ja	L2

(b) Misaligned loop body

PC	Instruction	Assembly	
127c	74 35	je	L1
127e	ba 00 00 00 00	mov	0x0,edx
1283	b9 00 00 00 00	mov	0x0,ecx
1288	48 01 e9	add	rbp,rcx
128b	48 89 01	mov	rax,(rcx)
128e	83 c2 01	add	0x1,edx
1291	89 d1	mov	edx,ecx
1293	83 f1 08	xor	0x8,ecx
1296	0f b6 c9	movzbl	cl,ecx
1299	83 e9 01	sub	0x1,ecx
129c	48 63 c9	movslq	ecx,rcx
129f	4c 21 e1	and	r12,rcx
12a2	48 89 d0	mov	rdx,rax
12a5	48 c1 e0 00	shl	0xc,rax
12a9	25 00 f0 0f 00	and	0xff000,eax
12ae	38 14 03	cmp	dl,(rbx,rax,1)
12b1	77 d5	ja	L2

```
int value = 0, B[1] = {2};
for (i = 0; i < 2; i++){
    index = ((2 ^ i) - 1) & offset;
    leak(array[index]);
    if (value < B[0]) x+=2; else
x+=1;
    flush(&B);
}
```

Disables LSD

GadgetSpinner: Threat Model and Setup

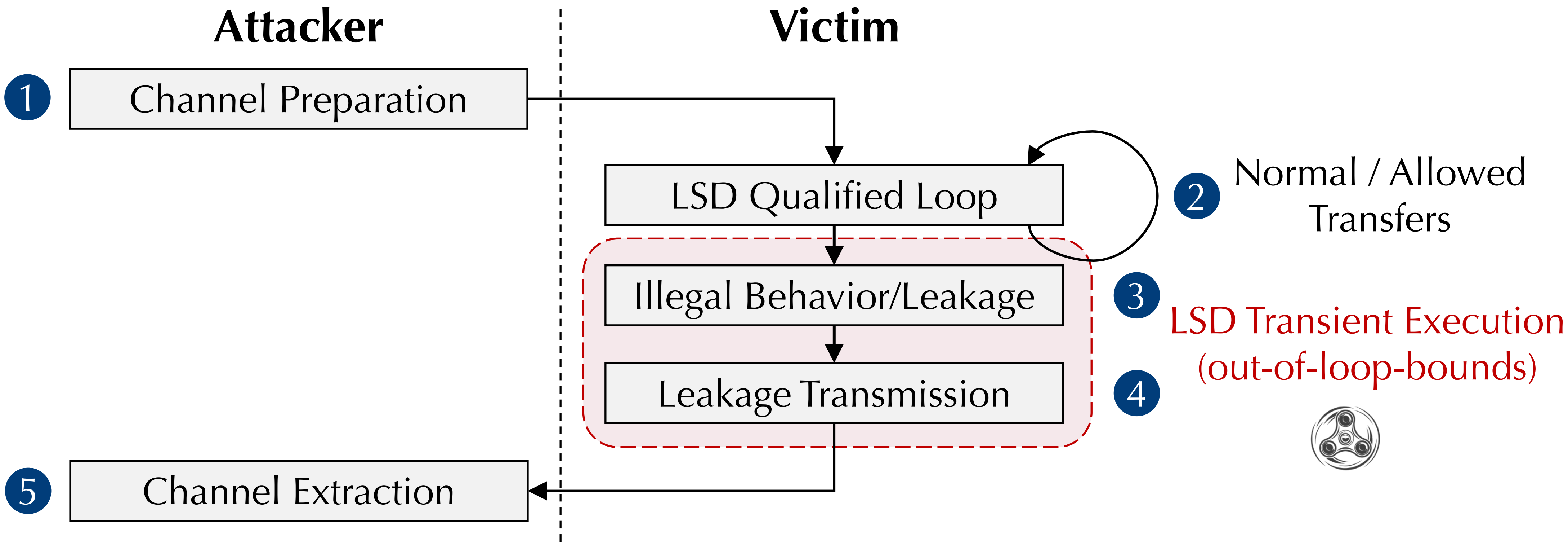
Threat Model

- Gadget existence in the victim (i.e., LSD qualified loops)
- Co-location with the victim not needed (enabling cross-core attacks)
- Triggering the victim with LSD qualified loops
- A transmission/extraction channel (e.g., cache primitives)

Specification	System 1	System 2
Cloud Provider	AWS EC2	Microsoft Azure
Processor	Xeon Platinum 8375C	Xeon Platinum 8370C
Architecture	Ice Lake (Sunny Cove)	Cascade Lake
Operating System	Ubuntu 20.04	
SGX	Not supported	SDK:2.19.100.3

NUS National University of Singapore

GADGETSPINNER: Attack Methodology and PoCs



Our Proof-of-Concept Attacks:

1. **Illegal Read Attack**
2. **Cross-Core Attack**
3. **Extracting CNN weights**
4. **Breaking KASLR**

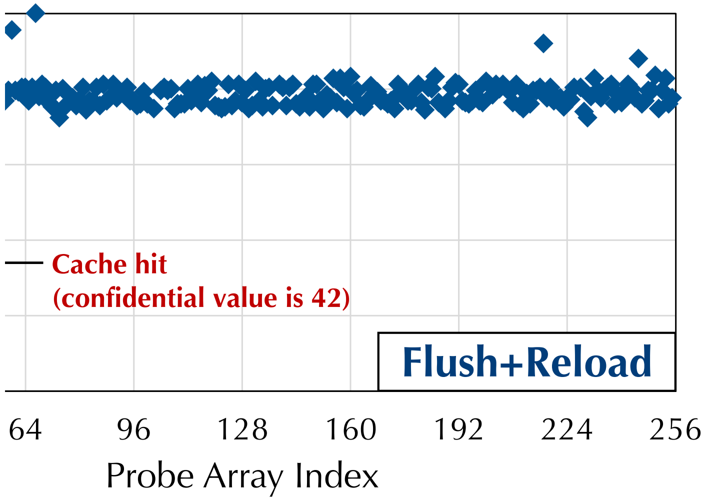
PoC #1: Illegal Read from Protected Memory

```
uint8_t A [8 * CACHE_LINE] = {0};
uint8_t B [8 * CACHE_LINE] = {42};

uint8_t *page_B = B & 0xfffffffffffff000;
mprotect(page_B, PAGE_SIZE, PORT_WRITE | PORT_EXEC);

```

B is protected (not readable)



PoC #2: Cross-Core Illegal Arbitrary Read

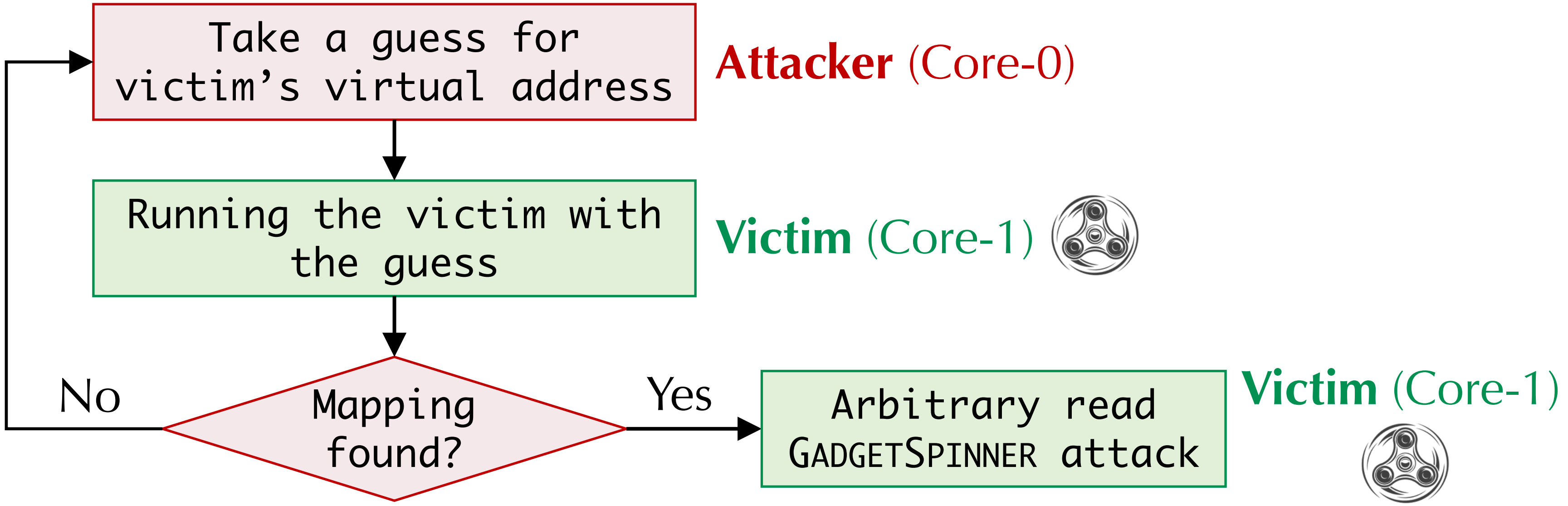
Cross-core and cross-process arbitrary reads are feasible

1. Only requires LSD, GADGETSPINNER qualified loops
2. No need for branch Mistraining (i.e., no need to share BPU)

Challenge: How to determine the virtual mapping of the victim?

16GiB search space in Linux[1]

~30 minutes runtime with SoTA search algorithms[2]



1. N. Vella, "Breaking 64 bit ASLR on linux x86-64,"https://github.com/nick0ve/how-to-bypass-aslr-on-linux-x86_64
2. Gruss, Daniel, et al. "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR." CCS 2016.

PoC #3: Extracting CNN Weights in SGX DNNL

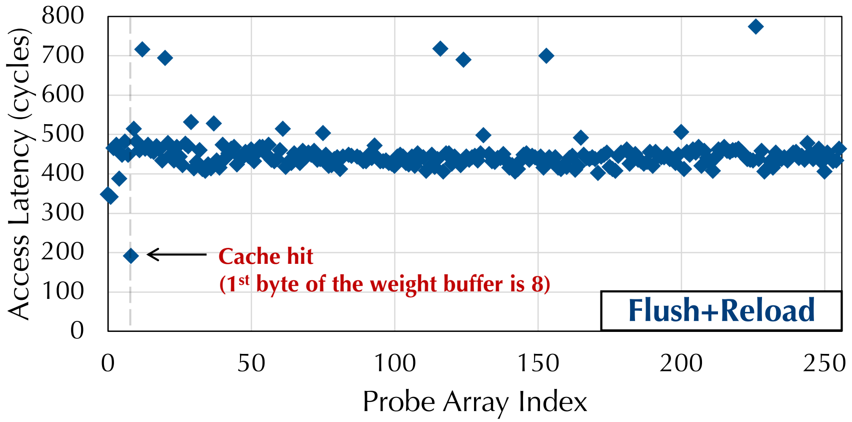
Untrusted Zone (Attacker)

```
//carefully crafting data and params
cpu_cnn_train_f32 (void *data, void *params);
```

Trusted Zone (Victim)

```
int cpu_cnn_train_f32 (void *data, void *params) {
    //training cnn
    init_net_data(data, dim, params);//clean unused data
}
```

Attacker-chosen inputs



PoC #4: Breaking Kernel ASLR (KASLR)

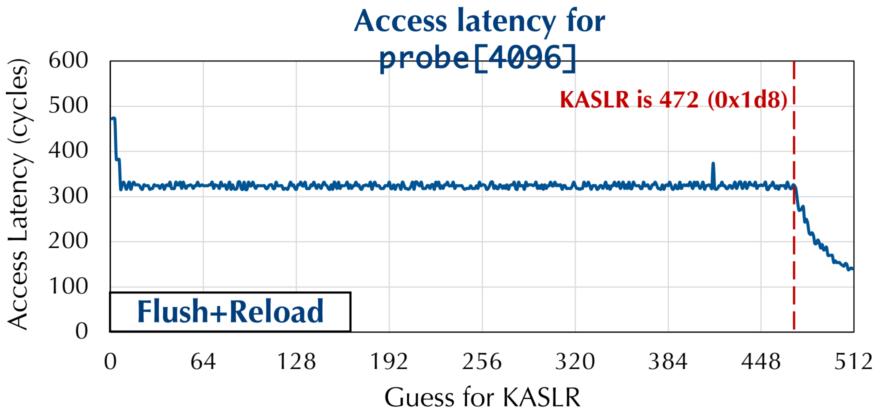
- KASLR: randomizes kernel memory placement at each system boot

512 possible kernel placements[1]

```
uint8_t probe[2 * 4096];

int loop_function (int offset) {
    uint64_t idx = 0;
    for (int i = 0; i < 8; i++)
    {
        uint8_t p = *(uint8_t *)(&array[0] + idx);
        uint8_t value = probe[p + (idx / offset) * 4096];
        idx = ((ARRAY_SIZE ^ i) - 1) & offset;
    }
}
```

value = { probe[0] i<8 (within loop bounds); probe[4096] i≥8 (out of loop bounds) }



probe[4096] will hit in TLB if p (guessed kernel address) is mapped

1. Canella, Claudio, et al. "Fallout: Leaking data on meltdown-resistant cpus." CCS 2019.

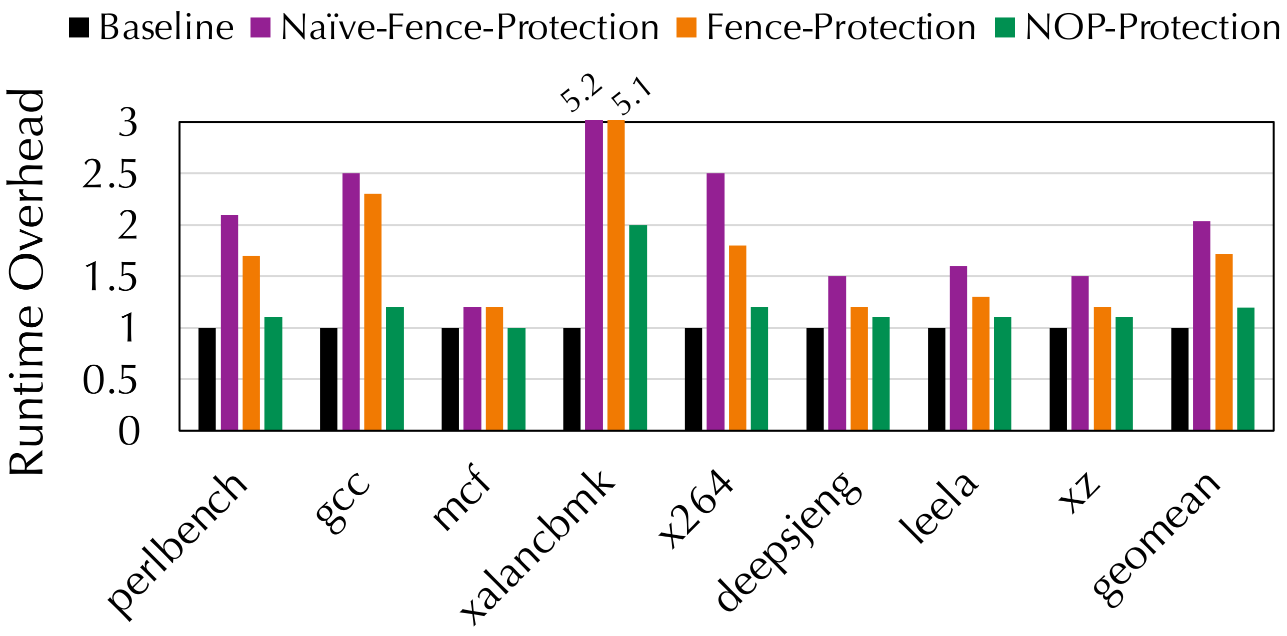
Mitigating GadgetSpinner for Existing Hardware

- We implemented three LLVM compiler passes and tested SPEC2017:

1. **Naive-Fence-Protection**
   Inserting a fence before all loop branches
   +103% overhead

2. **Fence-Protection**
   Inserting a fence before qualified loop branches
   +72% overhead

3. **NOP-Protection**
   Inserting NOPs to qualified loop bodies to disable the LSD
   +19% overhead



Mitigating GADGETSPINNER for Future Hardware

	Protection against Spectre-v1	Protection against GADGETSPINNER
1 Secure BPU Design and Usage Partitioning, randomizing, flushing the BPU Examples: HyBP [HPCA'22], Half&Half [S&P'23]	✓	X
2 Securing Specific Channels E.g., protecting caches against Spectre Examples: InvisiSpec [MICRO'18], CleanupSpec [MICRO'19]	◐✓	◐✓
3 Securing All Potential Channels E.g., restricting execution of speculative instructions Examples: STT [MICRO'19], DOLMA [USENIX Sec'21]	✓	✓

Conclusions

- Loop Stream Detector (LSD) creates a transient window, unrelated to BPU decisions
- We propose GADGETSPINNER, a new transient attack primitive exploiting the LSD transient window:
  - It bypasses Secure BPU protections
  - It makes cross-core transient execution attacks more practical
- We demonstrate threats of GADGETSPINNER via four different PoC attacks
- We investigate different compiler mitigations for GADGETSPINNER in existing hardware

30th International Symposium on High-Performance Computer Architecture (HPCA 2024)

Thanks for your Attention
Questions?

GadgetSpinner: A New Transient Execution Primitive using the Loop Stream Detector

Yun Chen* Ali Hajiabadi* Trevor E. Carlson

National University of Singapore

NUS
National University of Singapore

* Equal Contribution