SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

Jacob Fustos, Farzad Farshchi, Heechul Yun

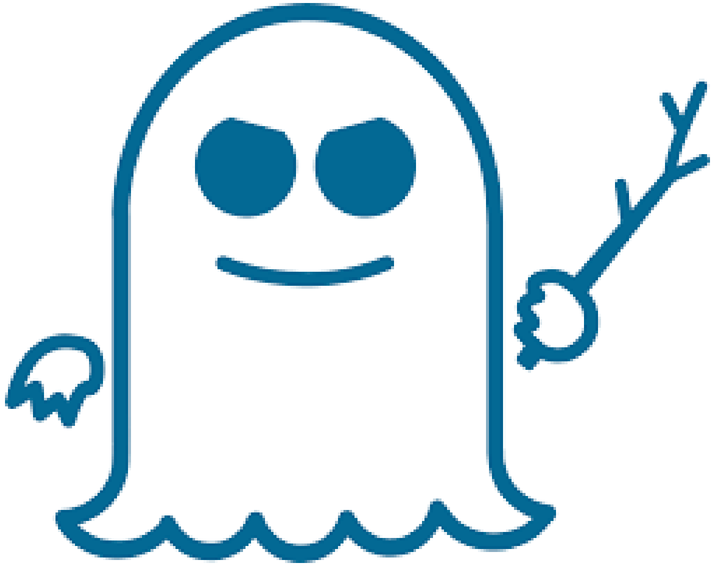
University of Kansas

Speculative Execution Attacks

- Attacks exploiting microarchitectural side-effects of executing speculative (transient) instructions
- Many variants

Attack	Description
Variant 1 (Spectre) [16]	Bounds Check Bypass
Variant 1.1 [15]	Bounds Check Bypass Store
Variant 1.2 [15]	Read-only Protection Bypass
Variant 2 (Spectre) [16]	Branch Target Injection
Variant 3 (Meltdown) [18]	Supervisor Protection Bypass
Variant 3a [12]	System Register Bypass
Lazy FP [24]	FPU Register Bypass
Variant 4 [9]	Speculative Store Bypass
ret2spec [20]	Return Stack Buffer
L1 Terminal Fault [11, 26]	Virtual Translation Bypass

No hardware support planned in near future



Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

- Assume `x` is under the attacker’s control
- Attacker trains the branch predictor to predict the branch is in-bound

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

1. [ACCESS]

- Speculative execution of the first line **accesses** the secret (array1[x])

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

2. [TRANSMIT]

- Speculative execution of the second, secret dependent load **transmits** the secret to a *microarchitectural state (e.g., cache)*

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

3. [RECEIVE]

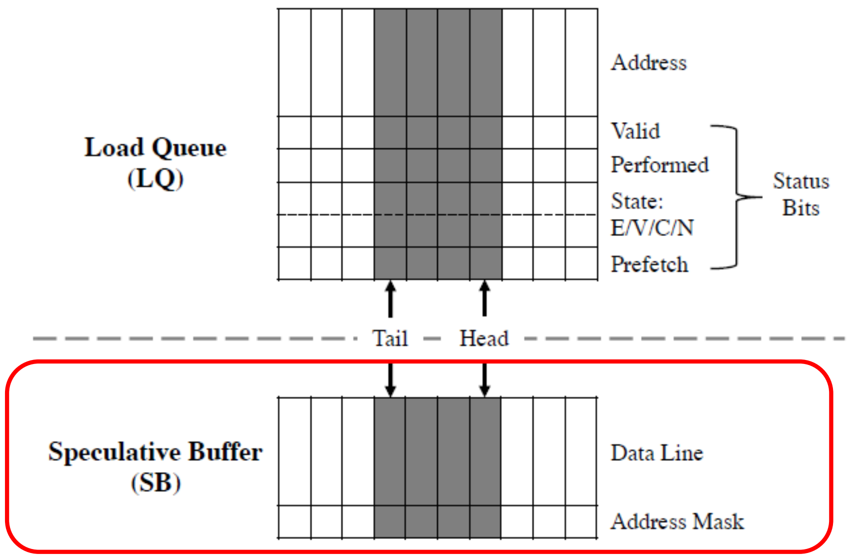
- Attacker **receives** the secret by timing access latency differences (cache hit vs. miss) among the elements in the probe array
  - Flush+reload, prime+probe, …

Existing Software Mitigation

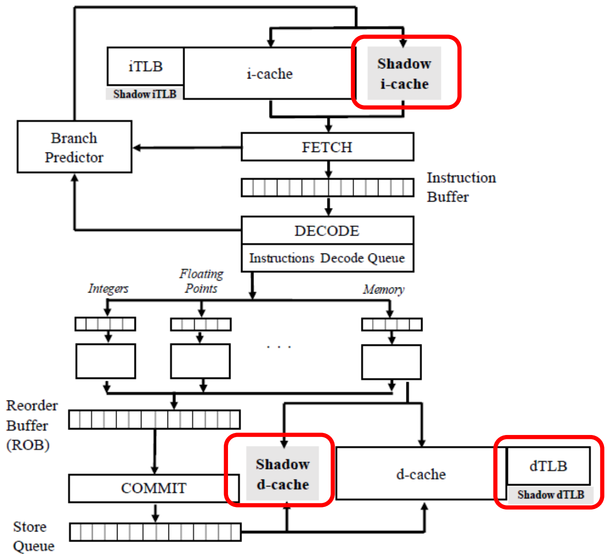
```
if(x < array1_length){
    _mm_lfence();
    val = array1[x];
    tmp = array2[val*512];
}
```

- Manually stop speculation
  - By inserting '`lfence`' instructions [Intel, 2018]
  - Or by introducing additional data dependencies [Carruth, 2018]
  - Error prone, high programming complexity, performance overhead

Existing Hardware Mitigation



InvisiSpec [Yan et al., MICRO'18]



SafeSpec [Khasawneh et al., DAC'19]

- Hide speculative execution
  - By buffering speculative results into additional "*shadow*" hardware structures
  - High complexity, high overhead (performance, space)

SpectreGuard

- Data-centric software/hardware collaborative approach
  - Software tells hardware what **data** (not code) needs protection
  - Hardware selectively protects the identified **data** from Spectre attacks

- Key observations
  - Not all data is secret
  - Not all speculative loads in a vulnerable code leak secret

Obs. 1: Not All Data Is Secret

- Non-sensitive data
  - Most program code, data
  - **Optimize for performance**
- Sensitive (secret) data
  - Cryptographic keys, passwords, …
  - **Optimize for security**

Memory
Attacker’s controlled data
AES encryption table
Other public information
RSA private key
Bank account information
Other secret data

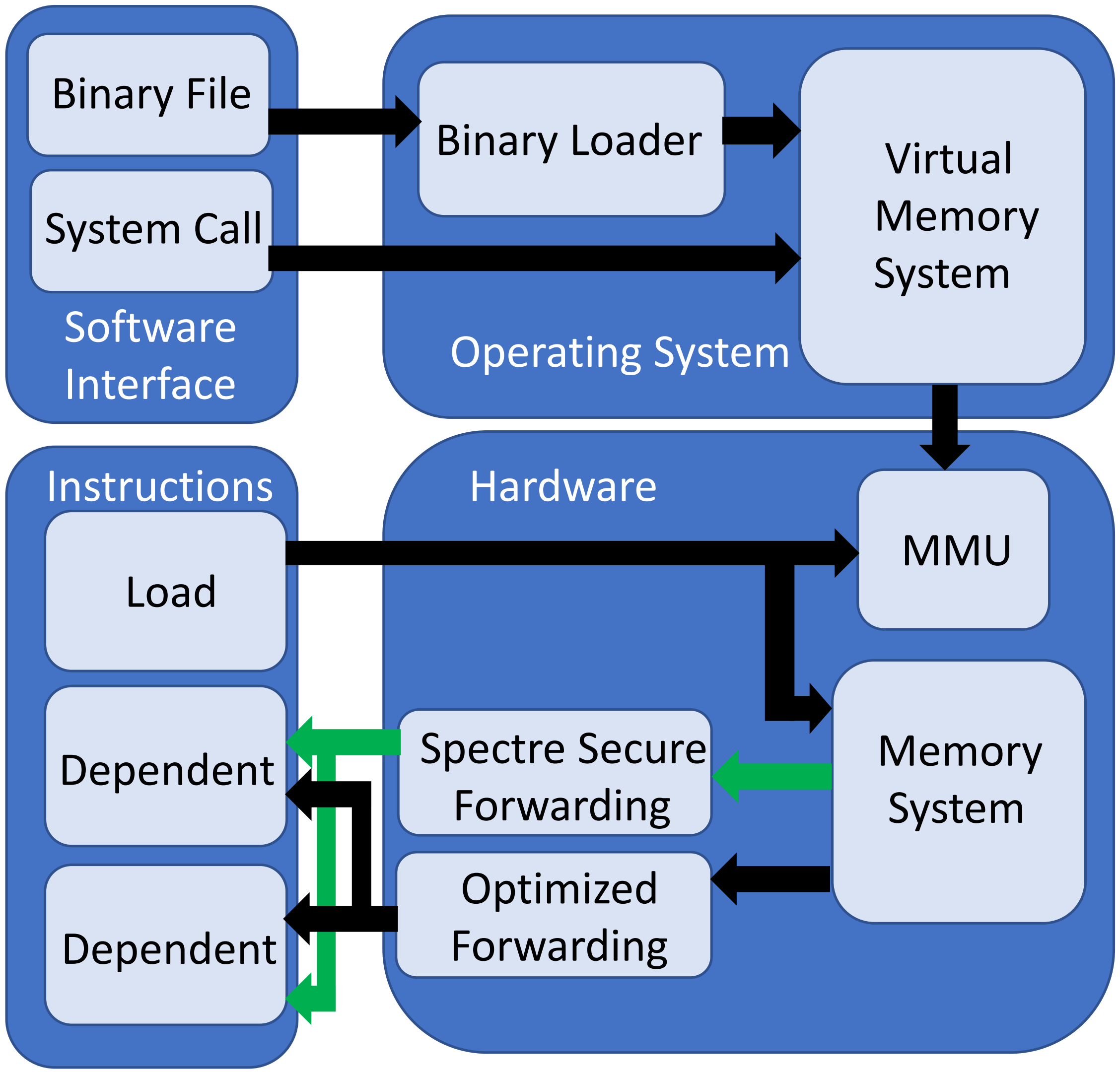
Obs. 2: Not All Speculative Loads Leak Secret

```
if(x < array1_length){
    val = array1[x];           1. [ACCESS]
    tmp = array2[val*512];     2. [TRANSMIT]
}
.........
```

- The first load does **NOT** leak secret
- The second, **secret dependent load** leaks the secret
- Delay the secret dependent load until *after* the branch is resolved

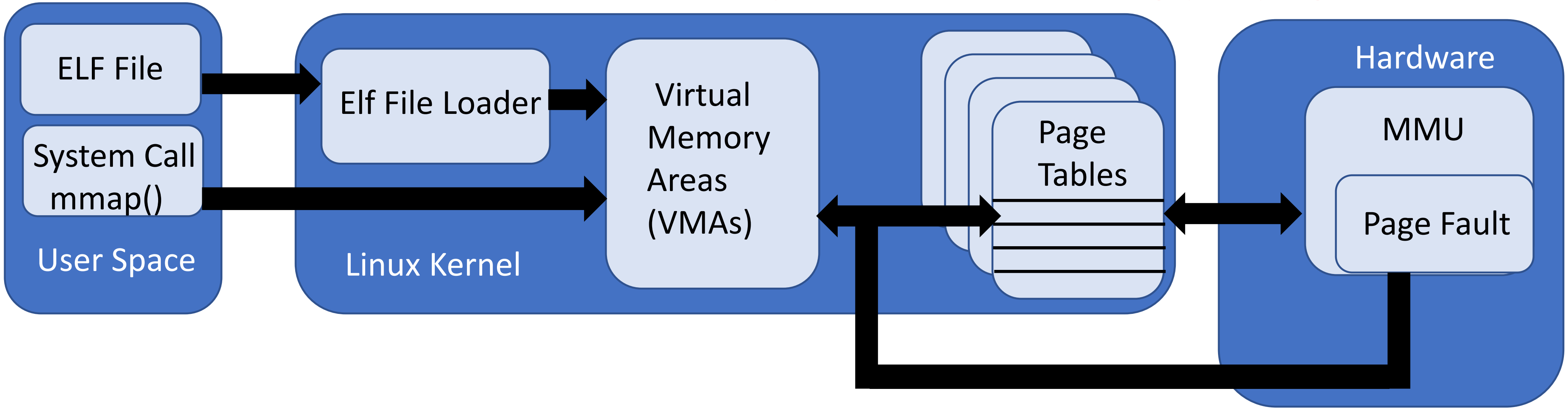
Approach

- Step 1: Software tells OS what data is secret
- Step 2: OS updates the *page table* entries
- Step 3: Load of the secret data is identified by MMU
- Step 4: Non-speculative data forwarding is **delayed** until safe



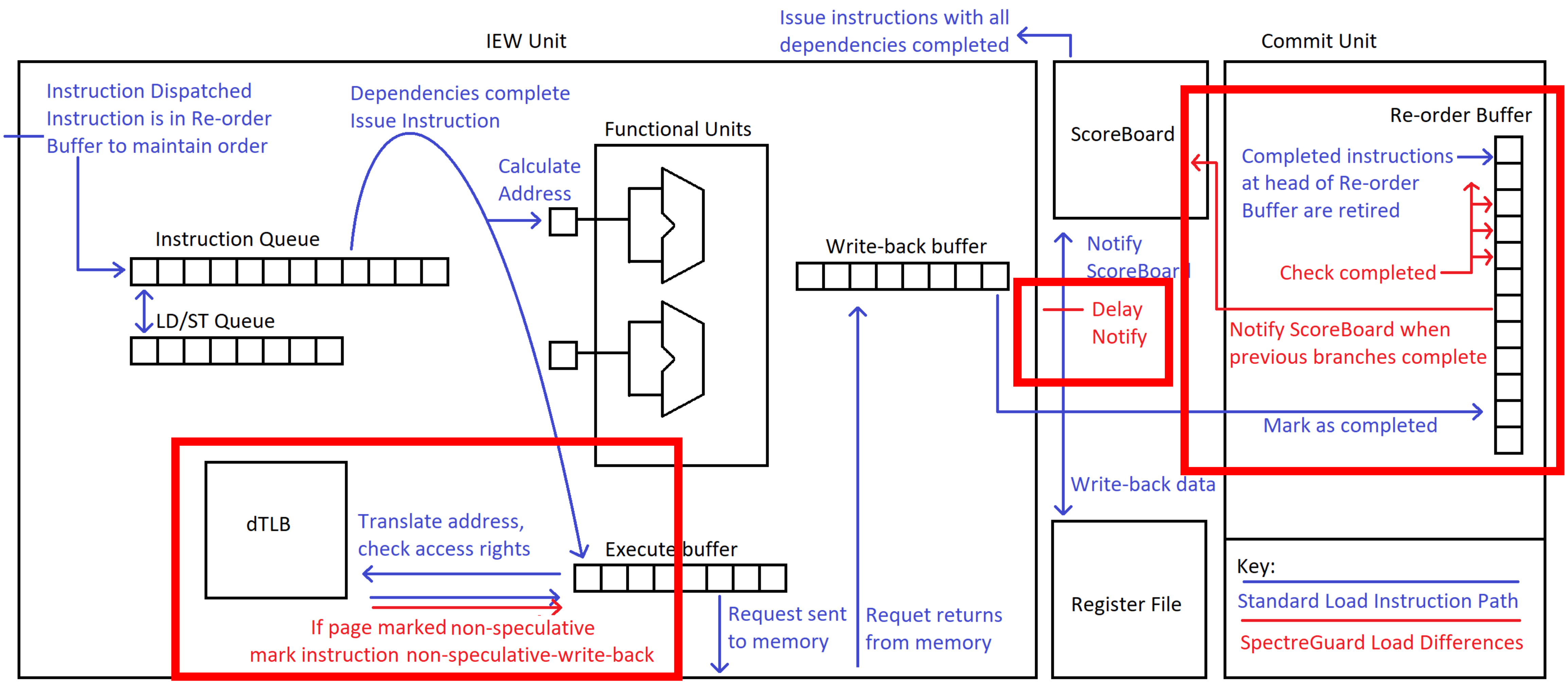
Linux Kernel Support

Non-speculative (NS) flag propagation



- Non-Speculative (NS) memory regions
  - Memory regions that may contain secret
  - Declared by software through a system call (`mmap`) or ELF header
  - Updated by OS in the page table (a single bit *NS* flag per page)

Gem5 Implementation



Evaluation Setup

- Full system simulation using Gem5 (O3CPU model) and Linux kernel (4.18)

Core	Single-core (x86 ISA), 8 issue, out-of-order, 2 GHz IQ: 64, ROB: 192, LSQ: 32/32
Cache	Private L1-I/D: 16/64 KiB (4/8-way), 1 cycle latency Shared L2: 256 KiB (16-way), 8 cycle latency
DRAM	Read/write buffers: 32/64, open-adaptive policy DDR3@800MHz, 1 rank, 8 banks

- Comparison
  - *Native*: unmodified baseline system
  - *InvisiSpec*: a fully hardware solution [Yan et al., Micro'18]
  - *Fence*: a fully software solution (insert `lfence` after all branches)
  - *SG*: SpectreGuard

Synthetic Workloads

```
char *secret_key // secret data
```

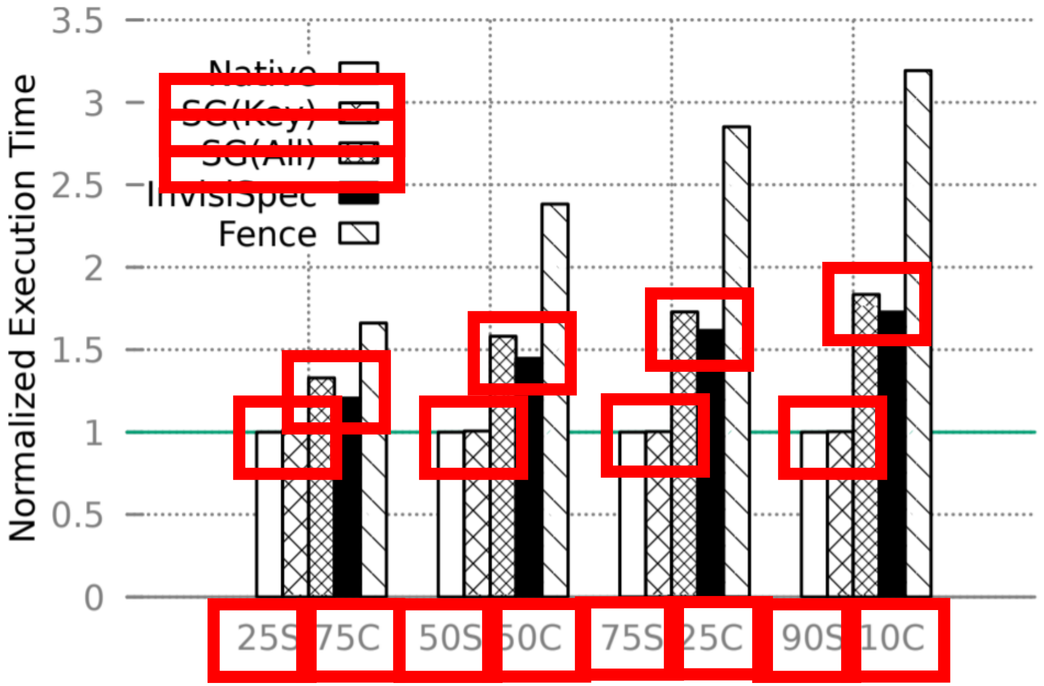
Secret data

```
void benchmark(int S, int C)
{
    // (S)pectre gadget, unrelated to the secret
    for (i = 0; i < S; i++)
        do_work();

    // En(C)ryption task accessing the secret
    for (i = 0; i < C; i++)
        encrypt();
}
```

- *(S)pectre*: contains Spectre gadget; does not access the secret key
- *En(C)ryption*: background communication, access the secret key

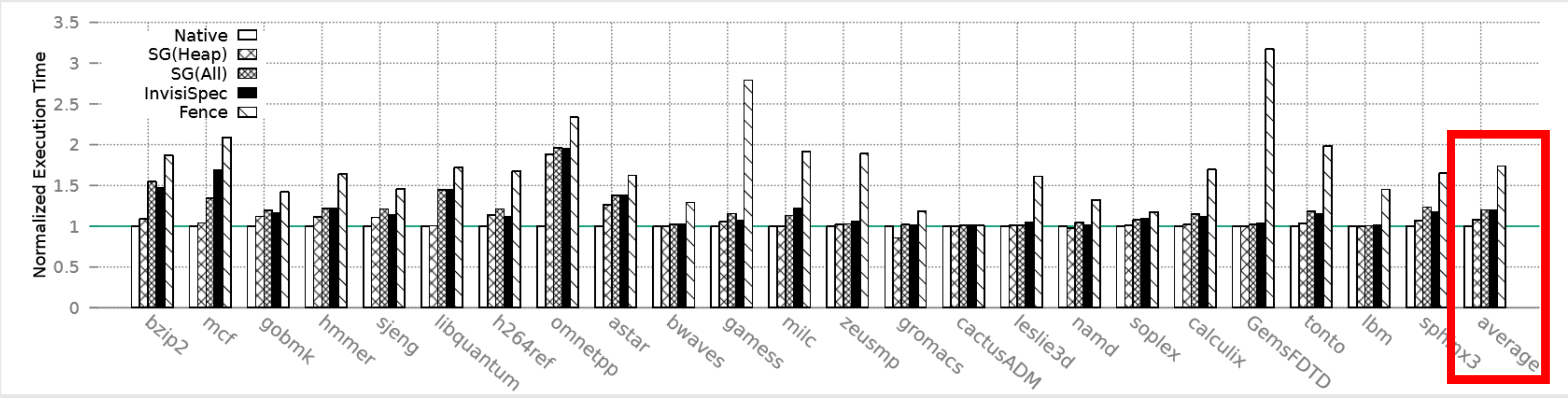
Results of Synthetic Workloads

```
char *secret_key; // secret data

void benchmark(int S, int C)
{
    // (S)pectre gadget, unrelated to the secret
    for (i = 0; i < S; i++)
        do_work();

    // En(C)ryption task accessing the secret
    for (i = 0; i < C; i++)
        encrypt();
}
```



- Varies percent time spent in S and C
- *SG(Key)* achieves native performance
    - Only secret key is marked non-speculative
- *SG(All) a*chieves comparable performance with *InvisiSpec*
    - <u>All memory</u> (code, data, heap, stack) is marked non-speculative (NS)

Results of SPEC2006 Benchmarks



- *SG(All)* achieves comparable performance with *InvisiSpec*
- *SG(Heap)* achieves better performance than *InvisiSpec*
  - Only heap is marked as non-speculative (NS) pages
- SpectreGuard enables targeted security and performance trade-offs

Conclusion

- Speculative execution attacks
  - Affect all high-performance out-of-order processors
  - Existing software mitigation suffers high programming complexity/overhead
  - Hardware only mitigation is costly
- SpectreGuard
  - A data-centric software/hardware collaborative defense mechanism
  - Low programming effort (identifying secret **data**, not vulnerable code)
  - Low hardware cost (no additional "shadow" structure)
  - Effective, targeted defense against Spectre attacks

https://github.com/CSL-KU/SpectreGuard

Future Work

- FPGA implementation extending an open-source RISC-V SoC
- Additional compiler/library support to aid programmers
- Apply our data-centric approach to address other speculative execution attacks

Thank You!

Disclaimer:

This research is supported by NSF CNS 1718880 and NSA Science of Security initiative contract #H98230-18-D-0009.