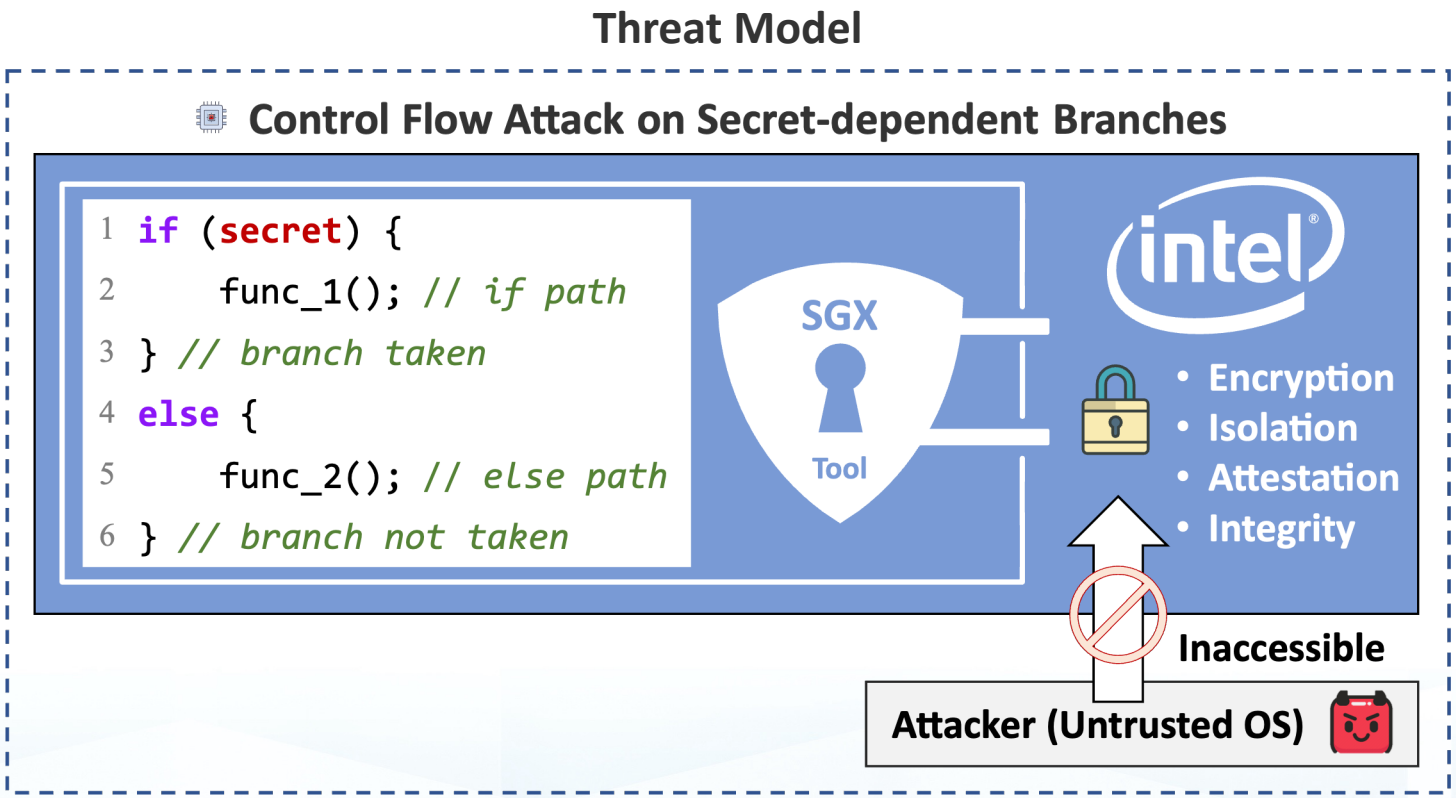
MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels

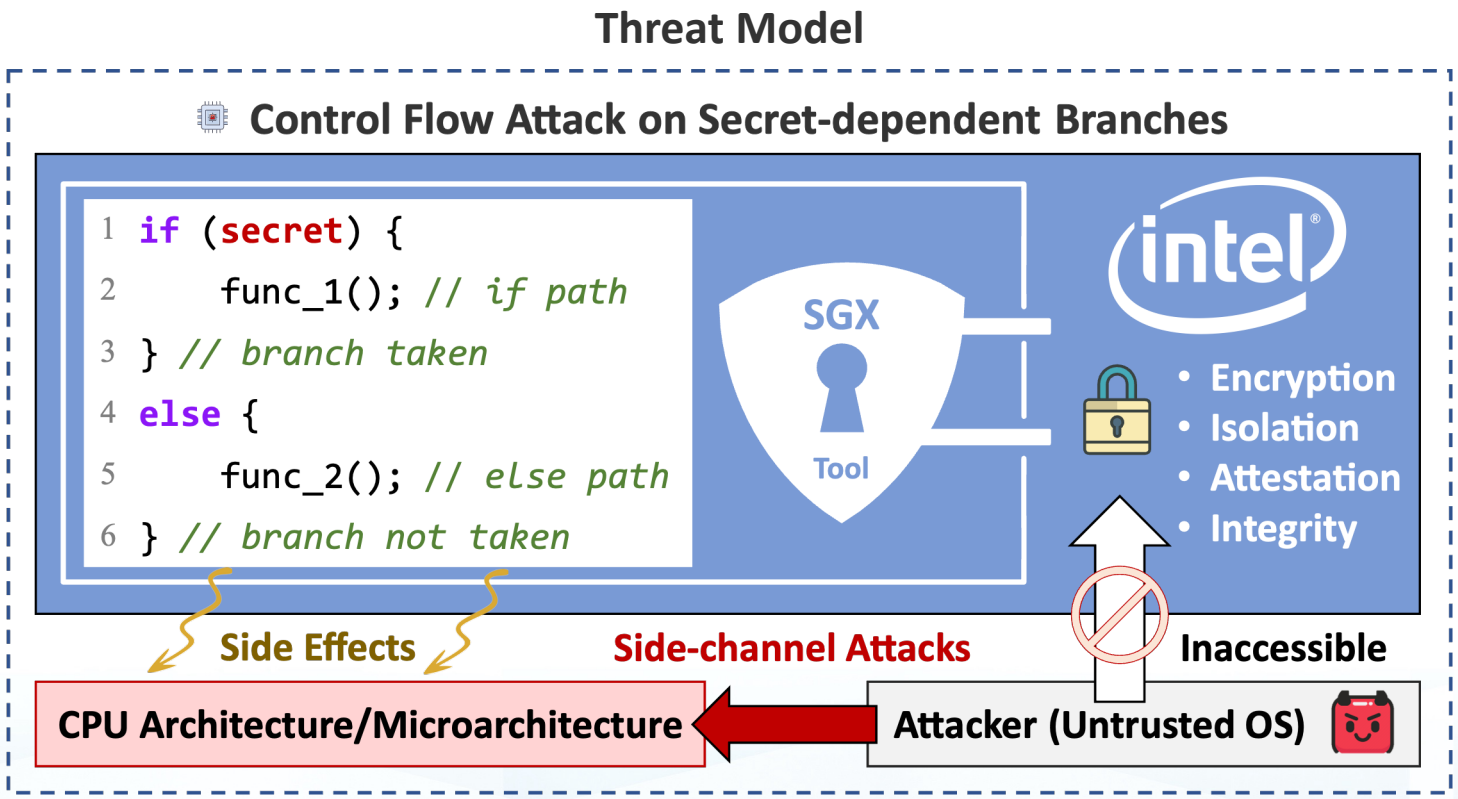
Chang Liu, Shuaihu Feng, Yuan Li, Dongsheng Wang, Wenjian He, Yongqiang Lyu and **Trevor E. Carlson**

ASPLOS 2025

Introduction: Control Flow Attack against SGX



Introduction: Control Flow Attack against SGX



Introduction: Control Flow Attack against SGX

Threat Model

Control Flow Attack on Secret-dependent Branches

```
1 if (secret) {
2     func_1(); // if path
3 } // branch taken
4 else {
5     func_2(); // else path
6 } // branch not taken
```

SGX Tool

intel

- Encryption
- Isolation
- Attestation
- Integrity

Side Effects

Side-channel Attacks

Inaccessible

CPU Architecture/Microarchitecture

Attacker (Untrusted OS)

Recent Control Flow Attacks

Effects on Branch Predictor

- Branch Shadowing
- NightVision
- PathFinder

Instruction Count or Type

- Copycat
- Nemesis
- PortSmash

Effects on other HW Units

- CacheZoom
- Frontal Attack
- AfterImage

Introduction: Control Flow Attack against SGX

[1] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In USENIX Security Symposium (USENIX Security), pages 557–574, 2017.

[2] Jiyong Yu, Trent Jaeger, and Christopher W. Fletcher. All Your PC Are Belong to Us: Exploiting Non-control-Transfer Instruction BTB Updates for Dynamic PC Extraction. In Proceedings of the Annual International Symposium on Computer Architecture (ISCA), pages 1–14, 2023.

[3] Hosein Yavarzadeh, Archit Agarwal, Max Christman, Christina Garman, Daniel Genkin, Andrew Kwong, Daniel Moghimi, Deian Stefan, Kazem Taram, and Dean Tullsen. Pathfinder: High-Resolution Control-Flow Attacks Exploiting the Conditional Branch Predictor. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 770–784, 2024.

[4] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. CopyCat: Controlled Instruction-Level Attacks on Enclaves. In USENIX Security Symposium (USENIX Security), pages 469– 486, 2020.

[5] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In Proceedings of the Conference on Computer and Communications Security (CCS), pages 178–195, 2018.

[6] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port Contention for Fun and Profit. In Symposium on Security and Privacy (SP), pages 870–887, 2019.

[7] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In International Conference on Cryptographic Hardware and Embedded Systems (CHES), pages 69–90, 2017.

[8] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Čapkun. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In USENIX Security Symposium (USENIX Security), pages 663–680, 2021.

[9] Yun Chen, Lingfeng Pei, and Trevor E. Carlson. AfterImage: Leaking Control Flow Data and Tracking Load Operations via the Hardware Prefetcher. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 16–32, 2023.

Recent Control Flow Attacks

Effects on Branch Predictor

- Branch Shadowing
- NightVision
- PathFinder

Instruction Count or Type

- Copycat
- Nemesis
- PortSmash

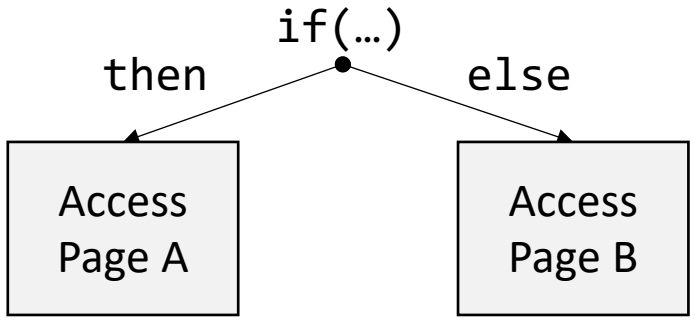
Effects on other HW Units

- CacheZoom
- Frontal Attack
- AfterImage

Motivation: Existing Defenses Bypassing

Attacks:
Interrupt-based

Controlled Channel



Page Fault Side Channel

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. ControlledChannel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In Symposium on Security and Privacy (SP), pages 640–656, 2015.

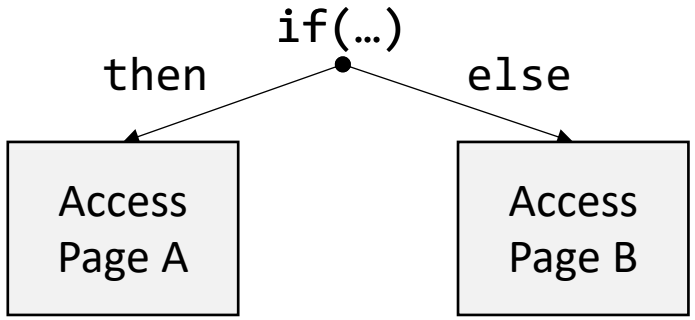
Motivation: Existing Defenses Bypassing

Attacks:
Interrupt-based

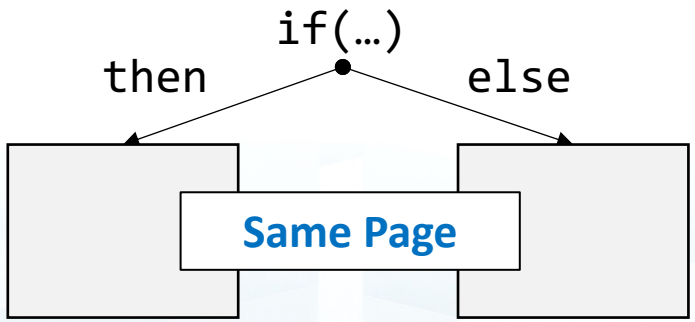
Controlled Channel

Defense:
Branch Balancing

Page Level



Page Fault Side Channel



Page Level Balancing

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. ControlledChannel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In Symposium on Security and Privacy (SP), pages 640–656, 2015.

Motivation: Existing Defenses Bypassing

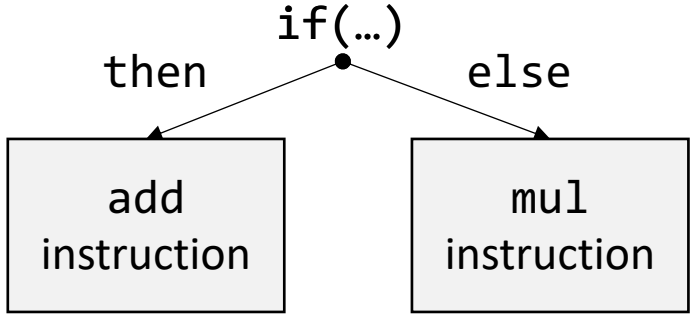
Attacks:
Interrupt-based

Controlled Channel

Nemesis

Defense:
Branch Balancing

Page Level



Single-step Execution Timing Side Channel

Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In Proceedings of the Conference on Computer and Communications Security (CCS), pages 178–195, 2018.

Motivation: Existing Defenses Bypassing

Attacks:
Interrupt-based

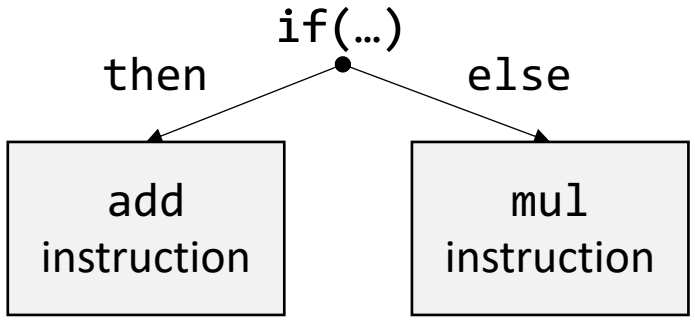
Controlled Channel

Nemesis

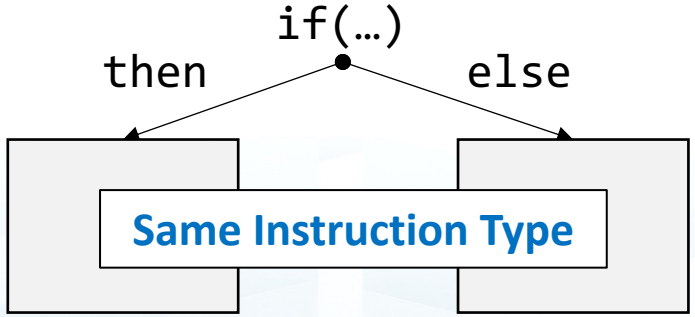
Defense:
Branch Balancing

Page Level

Instruction Type



Single-step Execution Timing Side Channel

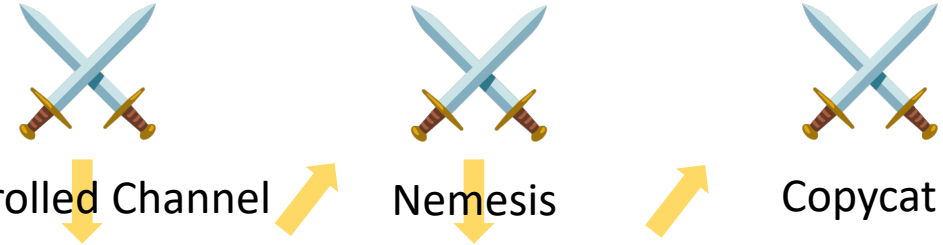


Instruction Type Balancing

Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In Proceedings of the Conference on Computer and Communications Security (CCS), pages 178–195, 2018.

Motivation: Existing Defenses Bypassing

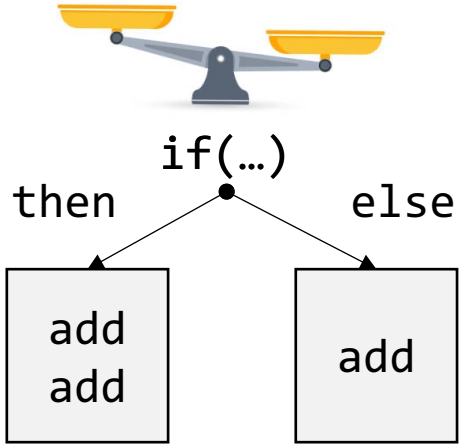
Attacks:
Interrupt-based



Controlled Channel → Nemesis → Copycat

Defense:
Branch Balancing

Page Level → Instruction Type



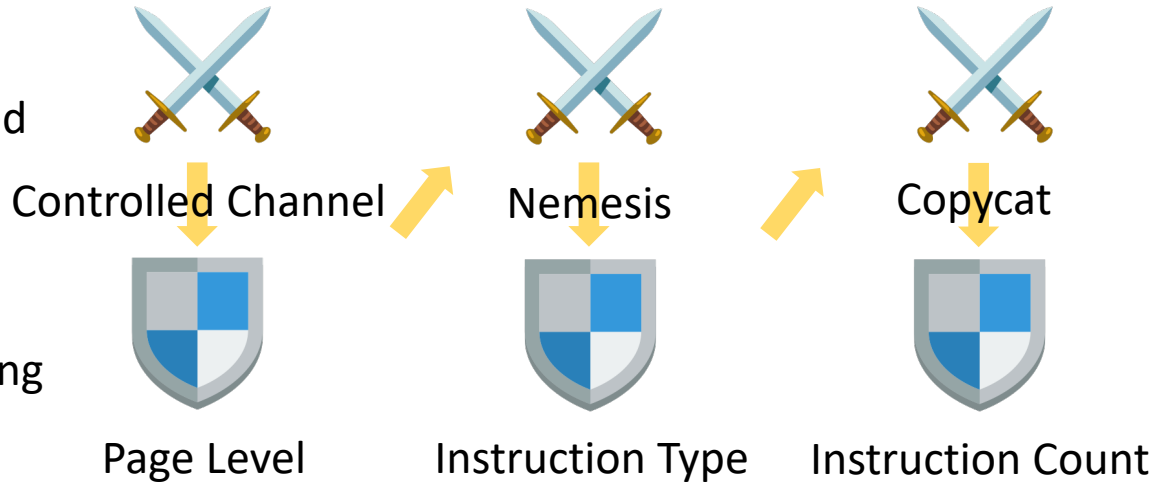
Single-step Counting Side Channel

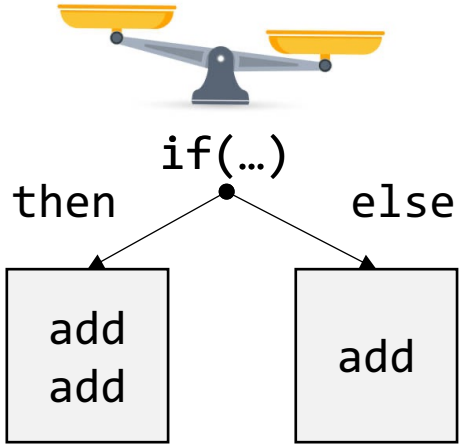
Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. CopyCat: Controlled Instruction-Level Attacks on Enclaves. In USENIX Security Symposium (USENIX Security), pages 469– 486, 2020.

Motivation: Existing Defenses Bypassing

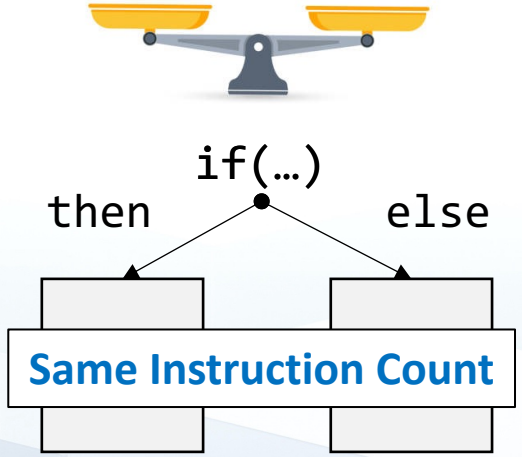
Attacks:
Interrupt-based

Defense:
Branch Balancing





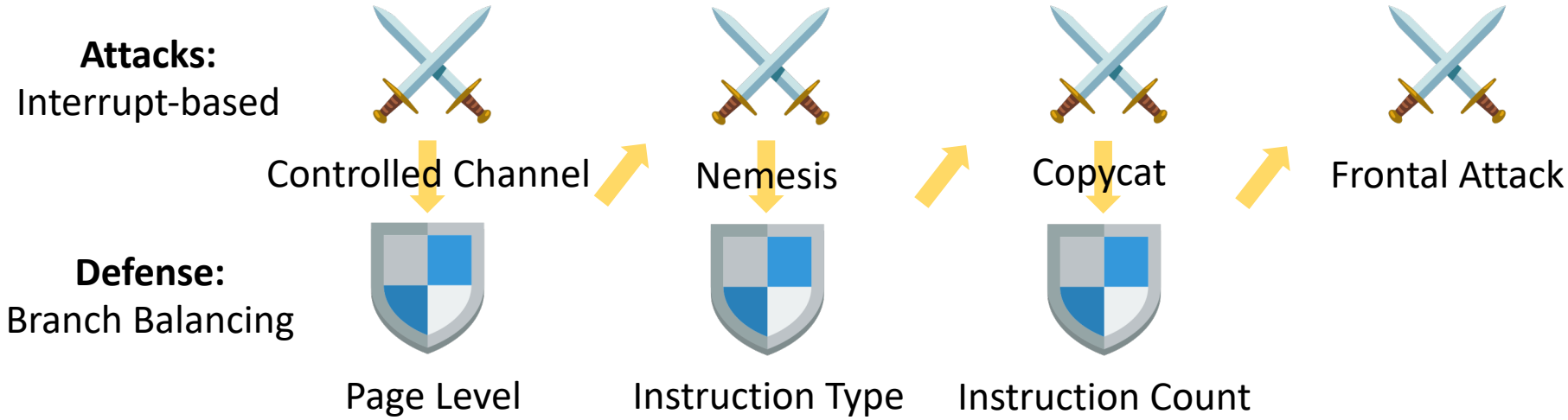
Single-step Counting Side Channel



Instruction Count Balancing

Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. CopyCat: Controlled Instruction-Level Attacks on Enclaves. In USENIX Security Symposium (USENIX Security), pages 469– 486, 2020.

Motivation: Existing Defenses Bypassing



Frontal Attack

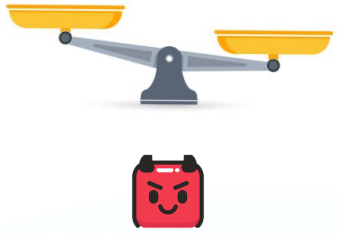
```
a1:  jne  0x0c
a2:  movq $0, (%rdi)
a3:  movq $0, (%rdi)
```

Single-step Execution

T_1

T_2

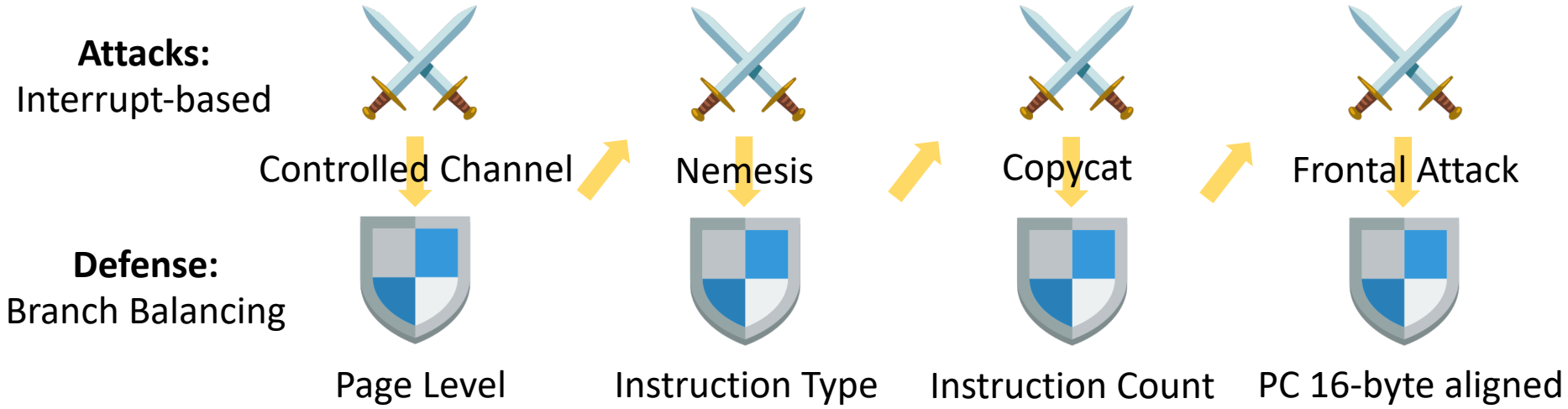
$T_1 \neq T_2$ iif $(a_2 - a_1) \neq (a_3 - a_1) \bmod 16$



Store PC Leakage through Frontal Side Channel

Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Čapkun. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In USENIX Security Symposium (USENIX Security), pages 663–680, 2021.

Motivation: Existing Defenses Bypassing



Branch Balancing against Frontal Attack

```
a1:   jne   0x0c
a2:   movq $0, (%rdi)
      nop nop ...
a'3:  movq $0, (%rdi)
```

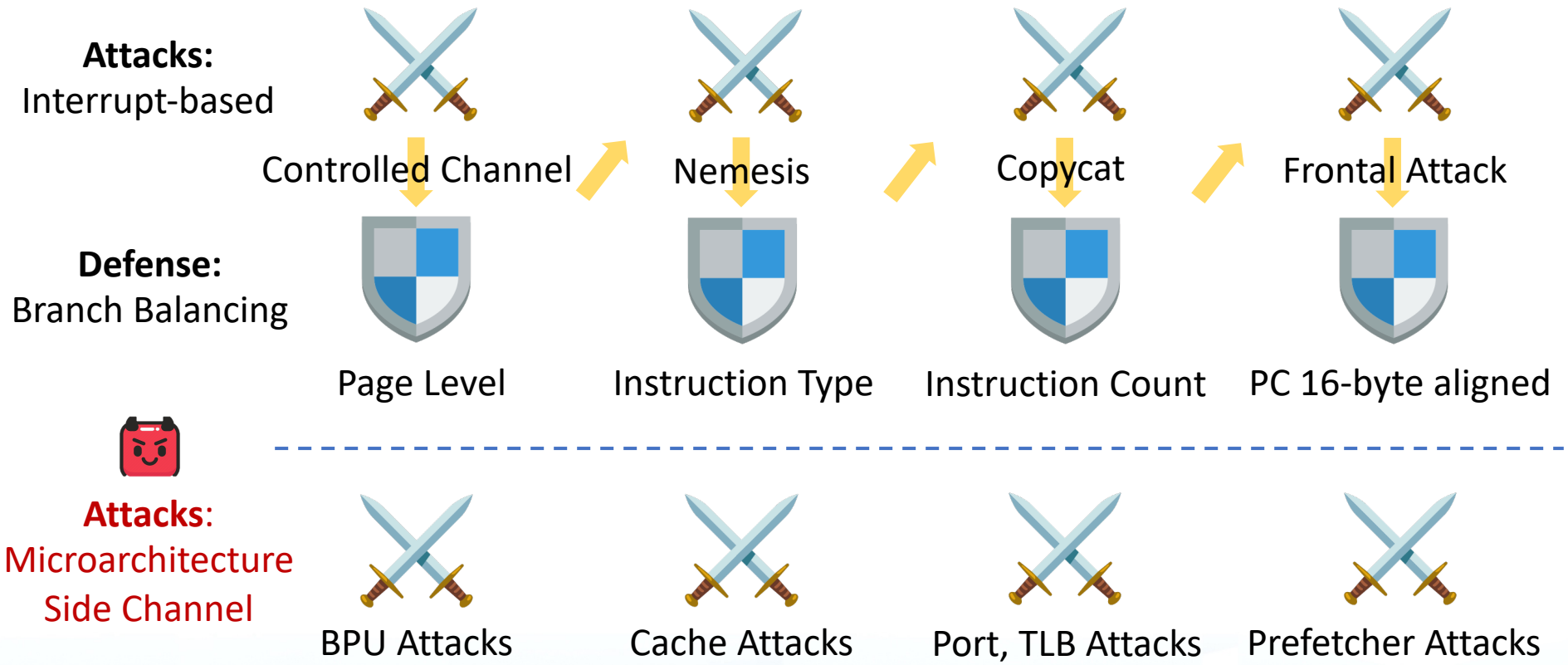
Store PC Aligned

$$(a_2 - a_1) = (a'_3 - a_1) \bmod 16$$

Store PC 16-byte Aligned from Branch

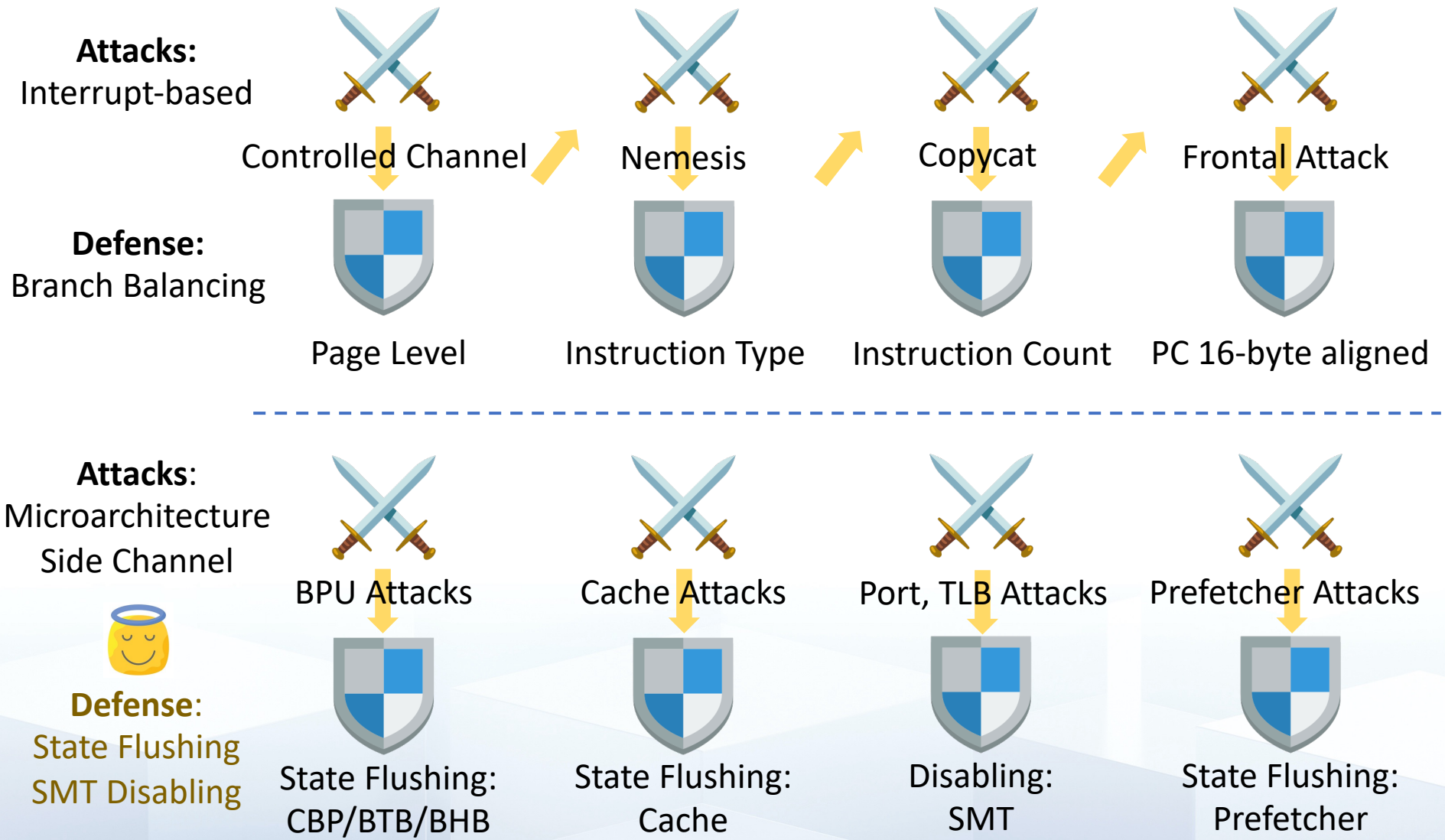
Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Čapkun. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In USENIX Security Symposium (USENIX Security), pages 663–680, 2021.

Motivation: Existing Defenses Bypassing



Hosein Yavarzadeh, Archit Agarwal, Max Christman, Christina Garman, Daniel Genkin, Andrew Kwong, Daniel Moghimi, Deian Stefan, Kazem Taram, and Dean Tullsen. Pathfinder: High-Resolution Control-Flow Attacks Exploiting the Conditional Branch Predictor. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 770–784, 2024.

Motivation: Existing Defenses Bypassing

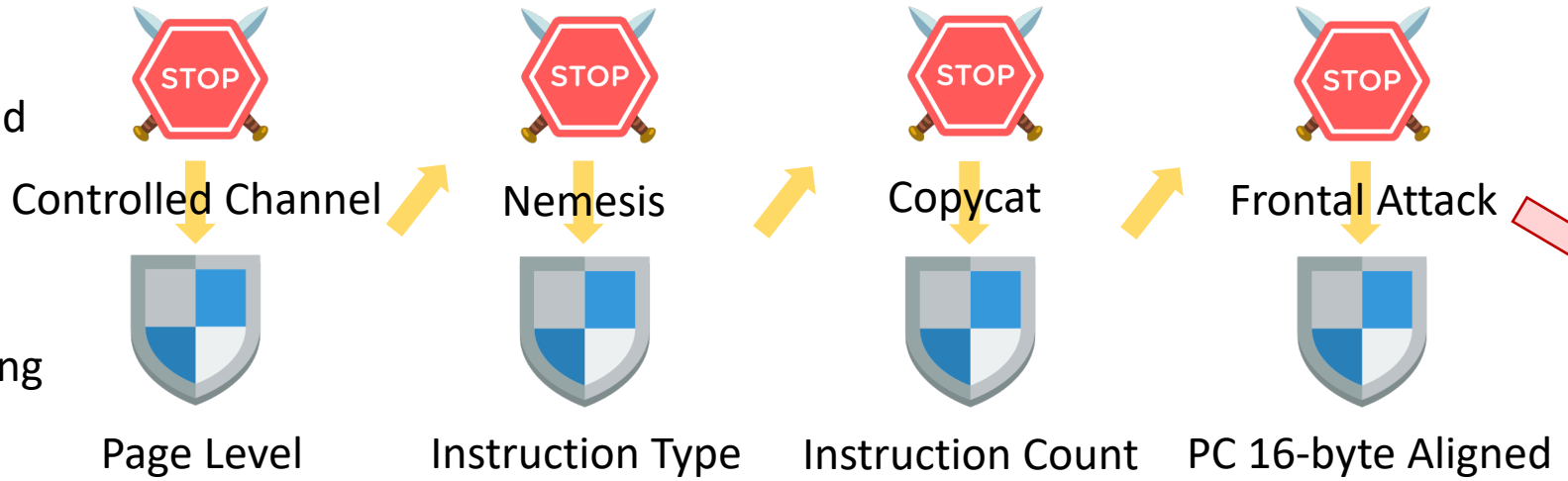


Hosein Yavarzadeh, Archit Agarwal, Max Christman, Christina Garman, Daniel Genkin, Andrew Kwong, Daniel Moghimi, Deian Stefan, Kazem Taram, and Dean Tullsen. Pathfinder: High-Resolution Control-Flow Attacks Exploiting the Conditional Branch Predictor. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 770–784, 2024.

Motivation: Existing Defenses Bypassing

Attacks:
Interrupt-based
Side Channel

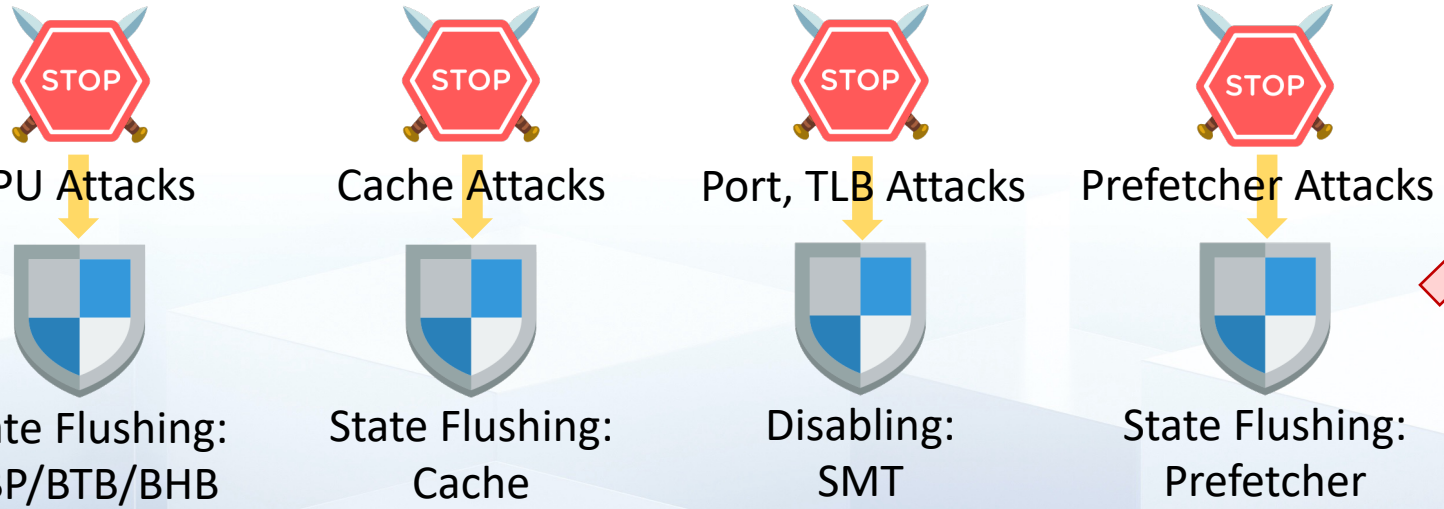
Defense:
Branch Balancing



NEW ATTACKS that can bypass all these deployable defenses?

Attacks:
Microarchitecture
Side Channel

Defense:
State Flushing
SMT Disabling



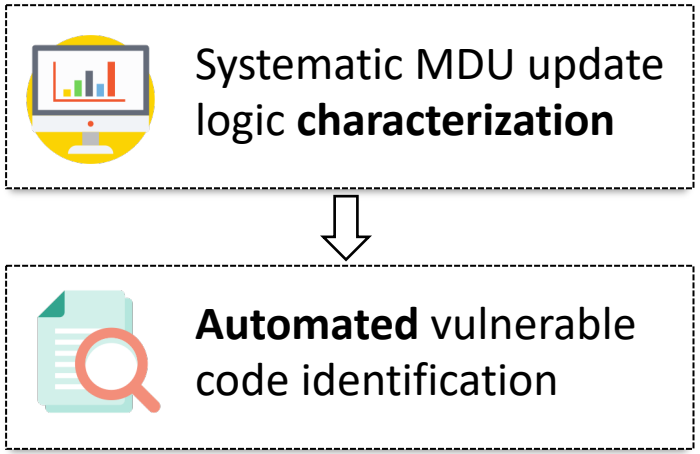
Our Contribution

- **Systematic MDU Characterization**
  - Update condition
  - Interaction with cache, TLB and ROB
  - Multiple stores and loads
  - **NOT isolated between the normal and secure world**

Systematic MDU update logic **characterization**

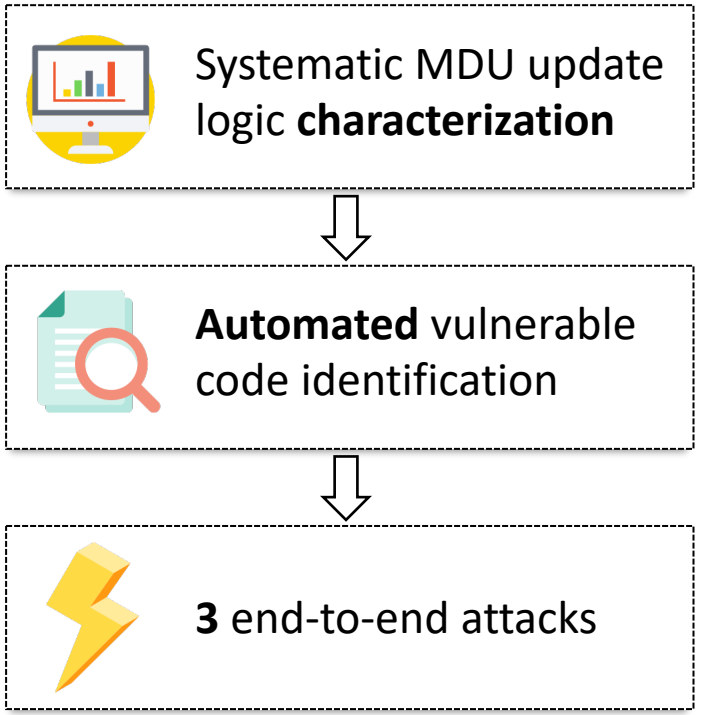
Our Contribution

- **Systematic MDU Characterization**
  - Update condition
  - Interaction with cache, TLB and ROB
  - Multiple stores and loads
  - **NOT isolated between the normal and secure world**
- **Vulnerable Loads Identification in Real-world Applications**
  - Modeling address generation delay
  - Measuring Delay Capacity



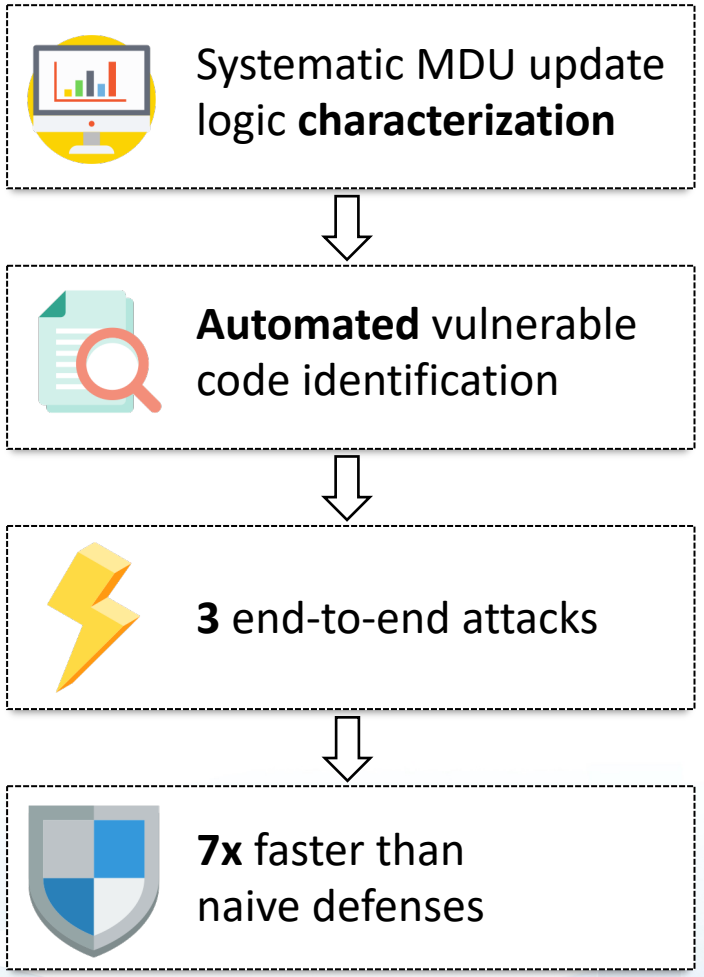
Our Contribution

- **Systematic MDU Characterization**
  - Update condition
  - Interaction with cache, TLB and ROB
  - Multiple stores and loads
  - **NOT isolated between the normal and secure world**
- **Vulnerable Loads Identification in Real-world Applications**
  - Modeling address generation delay
  - Measuring Delay Capacity
- **MDPeek: End-to-end Attacks with MDU Side Channel**
  - Attacking Libjpeg
  - Attacking RSA Generation (MbedTLS and WolfSSL)



Our Contribution

- **Systematic MDU Characterization**
  - Update condition
  - Interaction with cache, TLB and ROB
  - Multiple stores and loads
  - **NOT isolated between the normal and secure world**
- **Vulnerable Loads Identification in Real-world Applications**
  - Modeling address generation delay
  - Measuring Delay Capacity
- **MDPeek: End-to-end Attacks with MDU Side Channel**
  - Attacking Libjpeg
  - Attacking RSA Generation (MbedTLS and WolfSSL)
- **Defenses against MDPeek**
  - Naive defenses
  - Store-to-load Coupling



Background: Memory Disambiguation Unit

Unresolved Data Dependence

```
        movq (%rdx), %rdi
0x0e:   movq $0, (%rdi)
0x15:   movl (%rsi), %eax
        addl %eax, %ebx
        addl %eax, %edx
        addl %ebx, %ecx
```

(1) Fetch address from [%rdx]

(2) **Store** 0 to this address

(3) **Load** a value from [%rsi]

(4) **Use** the loaded value

(5) **Use** the loaded value

(6) **Use** the loaded value after calculation

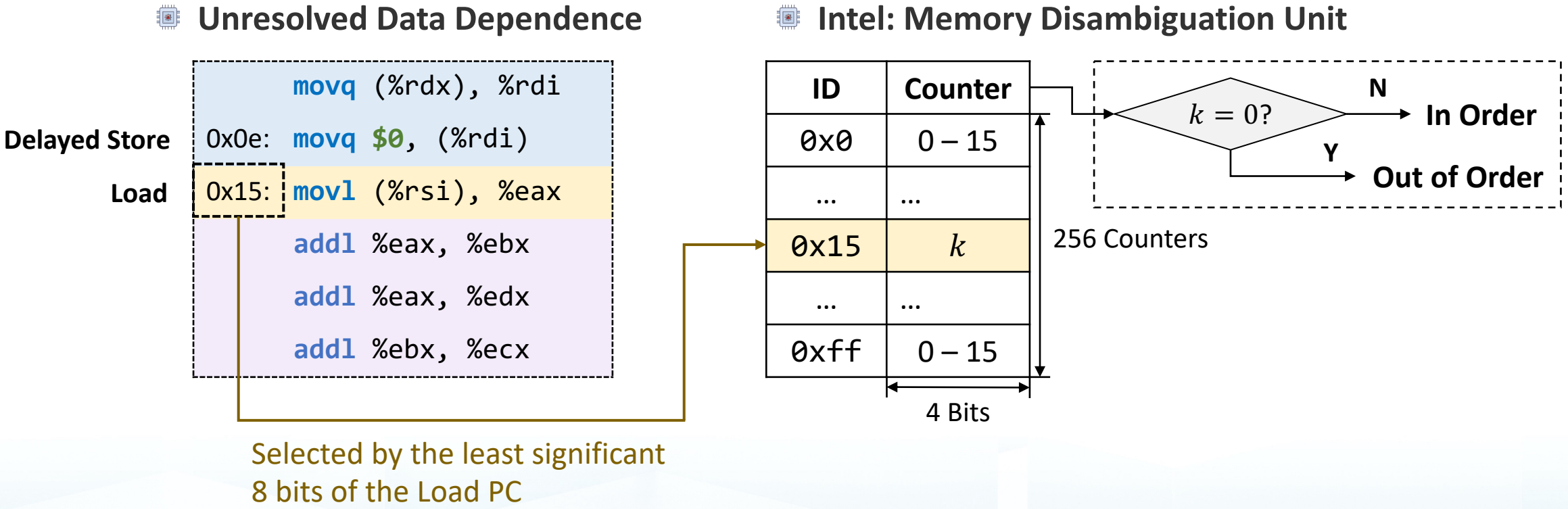
Before %rdi is ready

Delayed Store

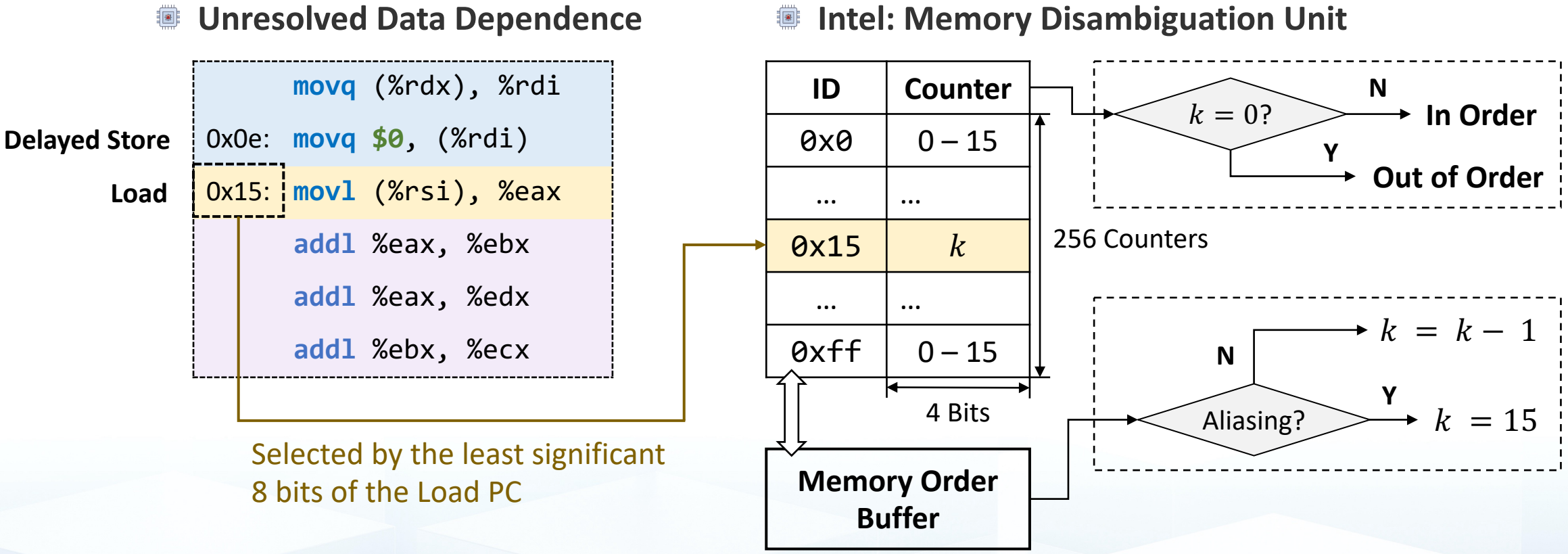
Ready Load

Out of order ?
In Order ?

Background: Memory Disambiguation Unit



Background: Memory Disambiguation Unit



Main Idea: MDU Side Channel

Microbenchmark

```
        .rep 10
        imul $1, %rdi, %rdi
        .endr
0x0e:   movq $0, (%rdi)      Delayed Store
0x15:   movl (%rsi), %eax    Load
```

Real-world Applications ?

Intel: Memory Disambiguation Unit

ID	Counter
0x0	0 – 15
…	…
0x15	k
…	…
0xff	0 – 15

256 Counters

4 Bits

Memory Order Buffer

$k = 0$? — N: **In Order**; Y: **Out of Order**

Aliasing? — N: $k = k - 1$; Y: $k = 15$

Change of k leaks which load is executed (**Control Flow Information**)

Which code can update the MDU ? (**Update Condition**)

1 **Reverse-engineering of MDU Update Logic**

2 Vulnerable Load Identification

3 MDPeek: End-to-end Attacks

4 Defenses against MDPeek

Characterization: Method

- **Method**
  - Microbenchmark
  - Transient Execution
  - Performance Monitor Counter

Microbenchmark

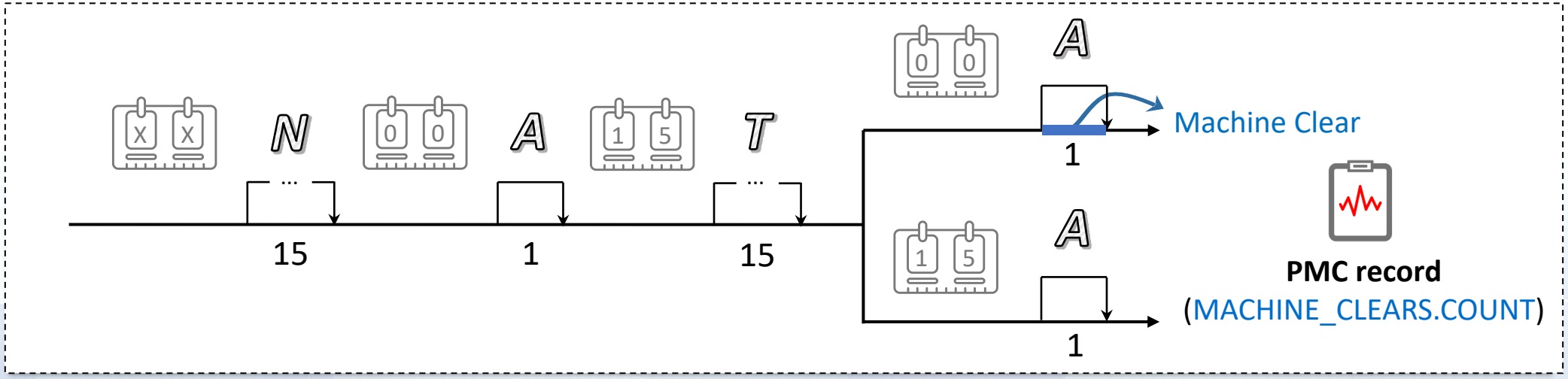
```
.rep 10
imul $1, %rdi, %rdi
.endr
0x0e: movq $0, (%rdi)
0x15: movl (%rsi), %eax
```

Testcase 

Notation

N	%rdi≠%rsi (non-aliased store and load)
A	%rdi=%rsi (aliased store and load)
T	Testcase with non-aliased store and load

Workflow of Characterization



Characterization: Method

- **Method**
  - Microbenchmark
  - Transient Execution
  - Performance Monitor Counter
- **An Example of Test Case**
  - Effects of ROB on MDU

Testcase

```
mdu_update_dispatch:
    mov $1, %rcx
    mov %rdi, %rax
    clflush (%rdx)
    mfence
    lfence
    movq (%rdx), %rdx
    .rep NUM_NOP
    nop
    .endr
    mov $0, %rdx
    div %rcx
    mov %rax, %rdi
    movq %rdi, (%rdi)
    .rep 60
    nop
    .endr
    movq (%rsi), %rsi
    lfence
    ret
```

Delay the instruction in the ROB head

Adjust the number of nop to control the layout of ROB

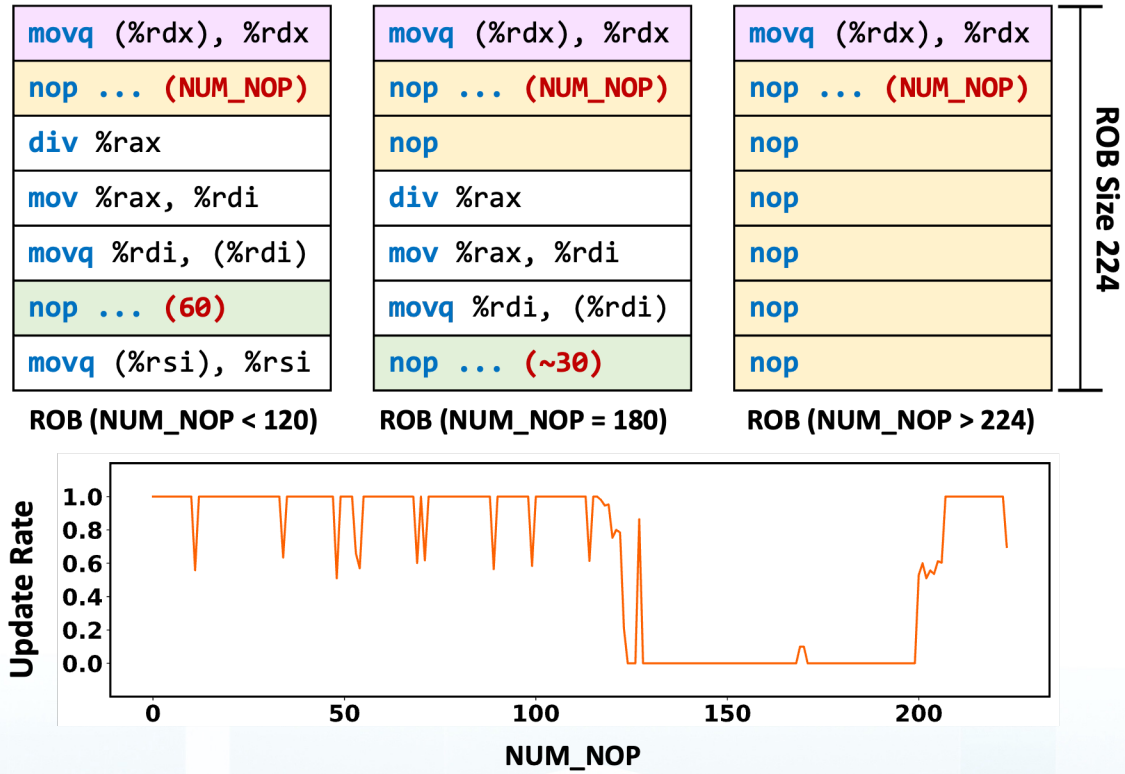
Store with delayed address generation

Load to be tested

Characterization: Method

- **Method**
  - Microbenchmark
  - Transient Execution
  - Performance Monitor Counter
- **An Example of Test Case**
  - Effects of ROB on MDU
  - Adjust NUM_NOP

```
mdu_update_dispatch:
    mov $1, %rcx
    mov %rdi, %rax
    clflush (%rdx)
    mfence
    lfence
    movq (%rdx), %rdx
    .rep NUM_NOP
    nop
    .endr
    mov $0, %rdx
    div %rcx
    mov %rax, %rdi
    movq %rdi, (%rdi)
    .rep 60
    nop
    .endr
    movq (%rsi), %rsi
    lfence
    ret
```

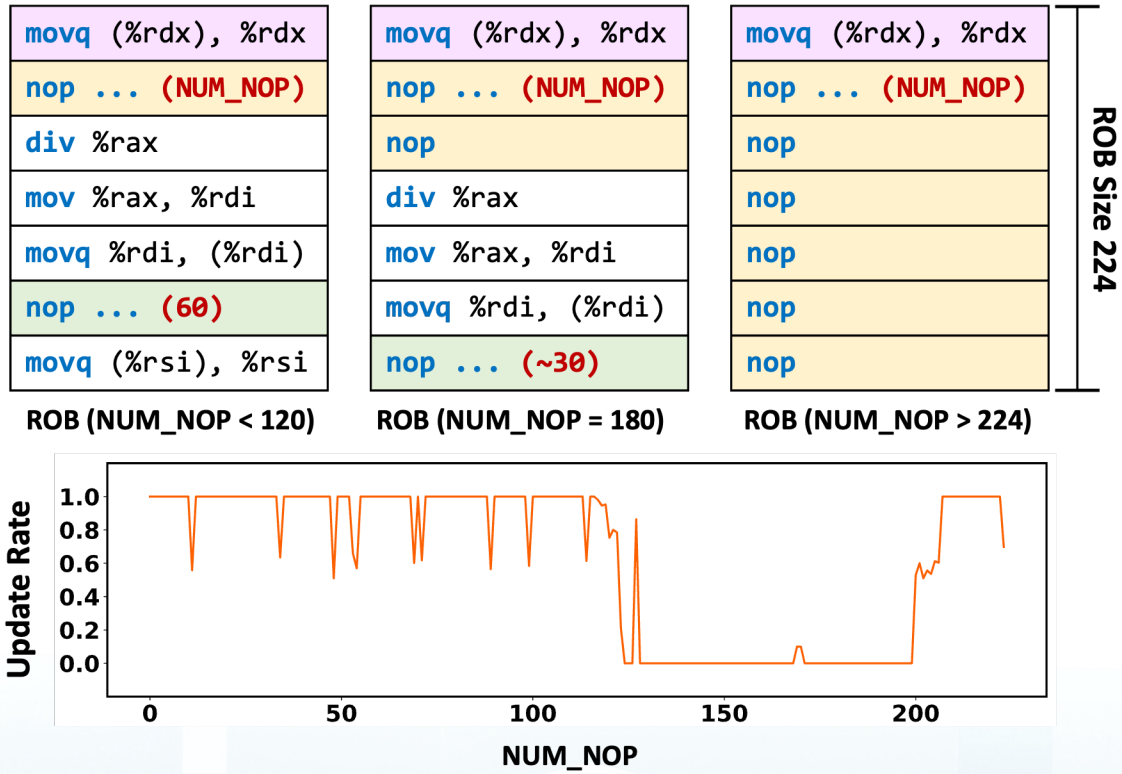


Characterization: Method

- **Method**
  - Microbenchmark
  - Transient Execution
  - Performance Monitor Counter
- **An Example of Test Case**
  - Effects of ROB on MDU
  - Adjust NUM_NOP

Insight
MDU can update only when both the delayed store and load are in the ROB.

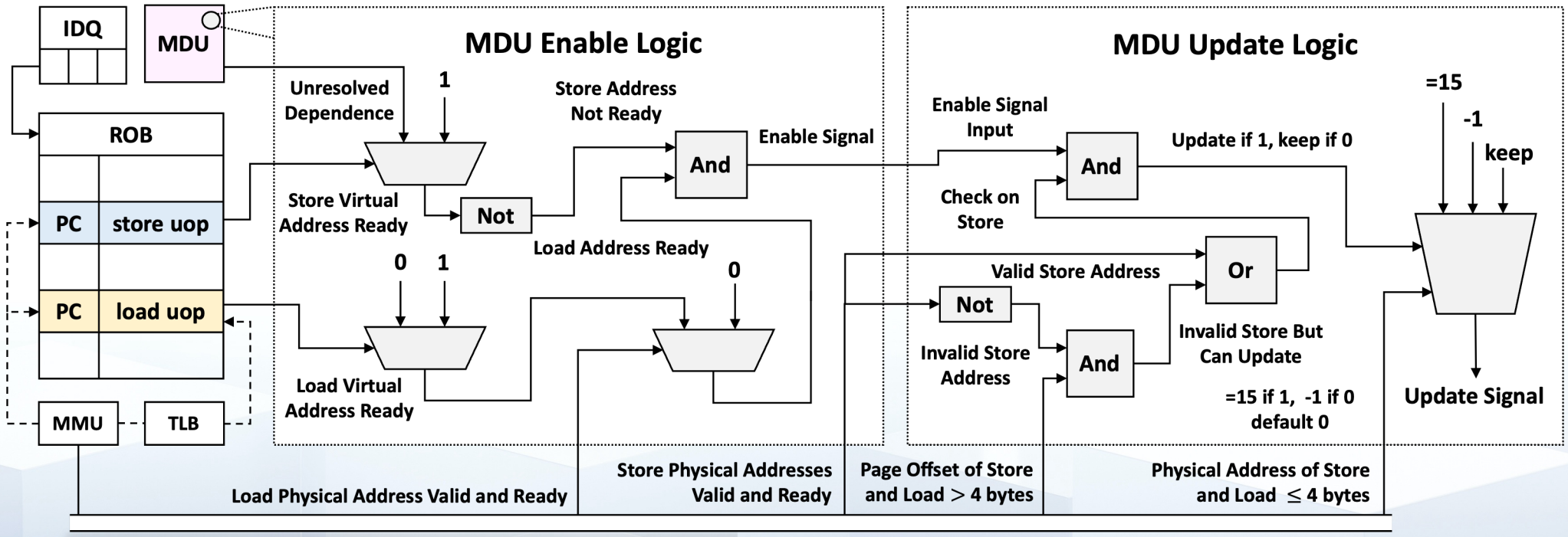
```
mdu_update_dispatch:
    mov $1, %rcx
    mov %rdi, %rax
    clflush (%rdx)
    mfence
    lfence
    movq (%rdx), %rdx
    .rep NUM_NOP
    nop
    .endr
    mov $0, %rdx
    div %rcx
    mov %rax, %rdi
    movq %rdi, (%rdi)
    .rep 60
    nop
    .endr
    movq (%rsi), %rsi
    lfence
    ret
```



Characterization: Results and Insights

Results

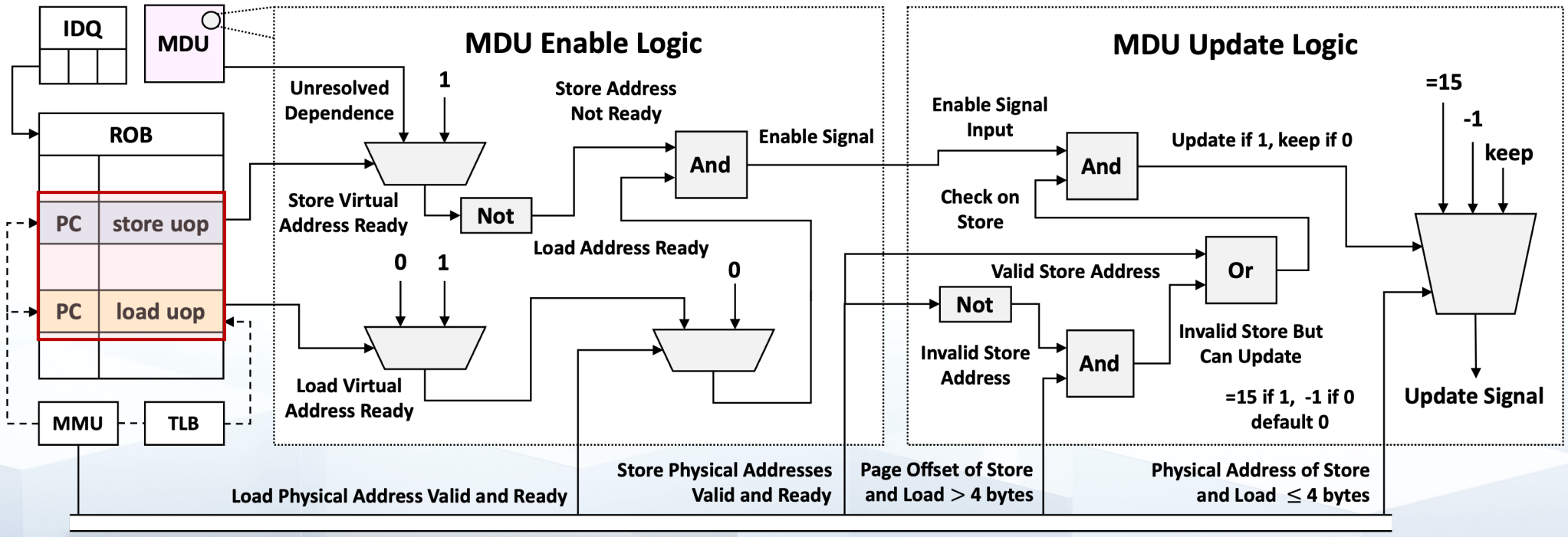
- Shown as follows



Characterization: Results and Insights

Insights on Update Condition

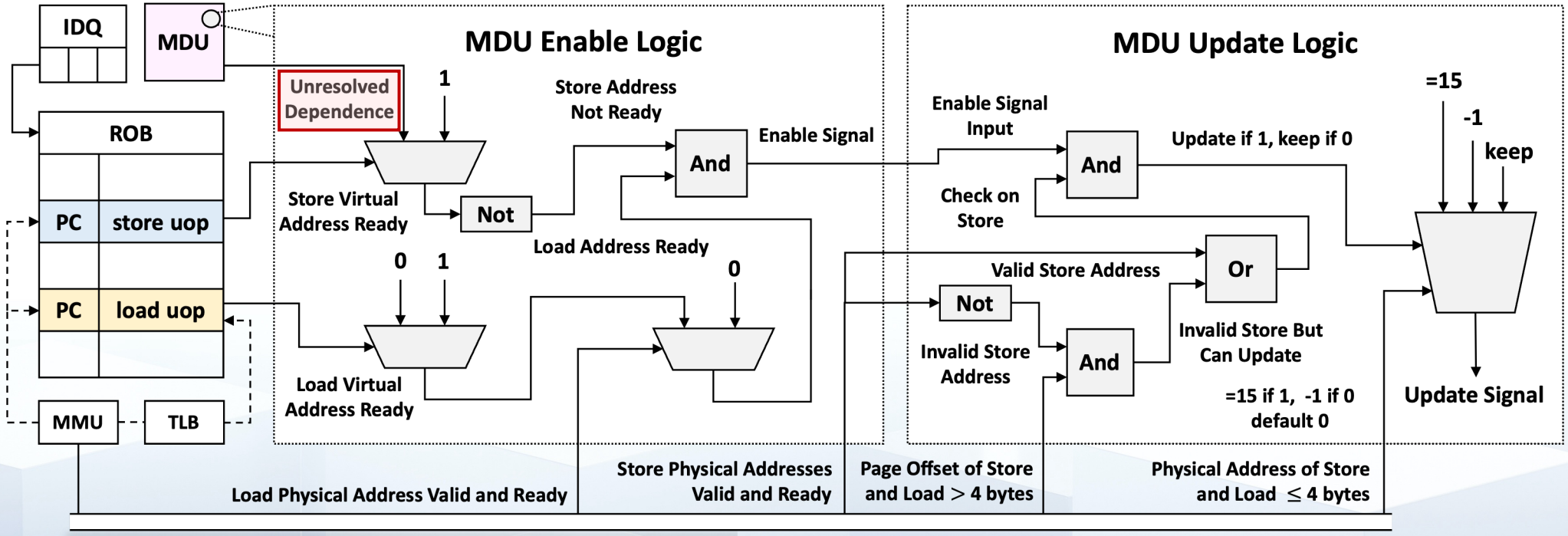
- Store is allocated in the ROB earlier than load



Characterization: Results and Insights

Insights on Update Condition

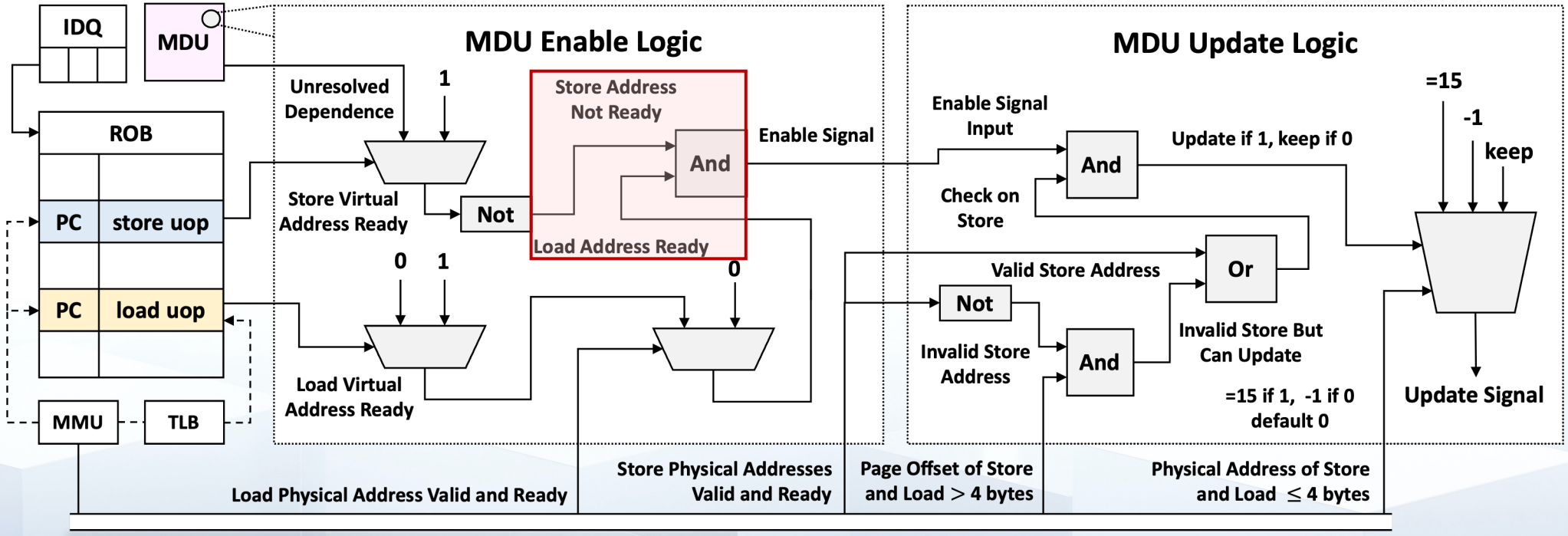
- Store is allocated in the ROB earlier than load
- Unresolved dependence is necessary



Characterization: Results and Insights

Insights on Update Condition

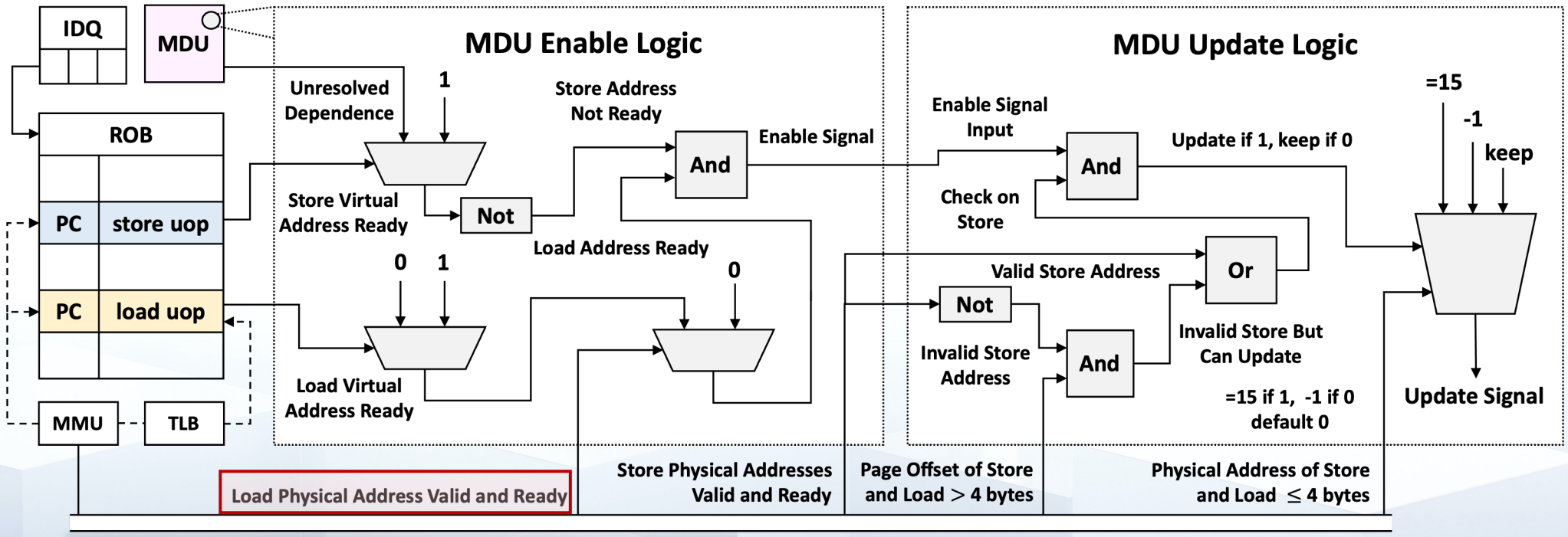
- Store is allocated in the ROB earlier than load
- Unresolved dependence is necessary
- Address of the store is generated slower than load



Characterization: Results and Insights

Insights on Update Condition

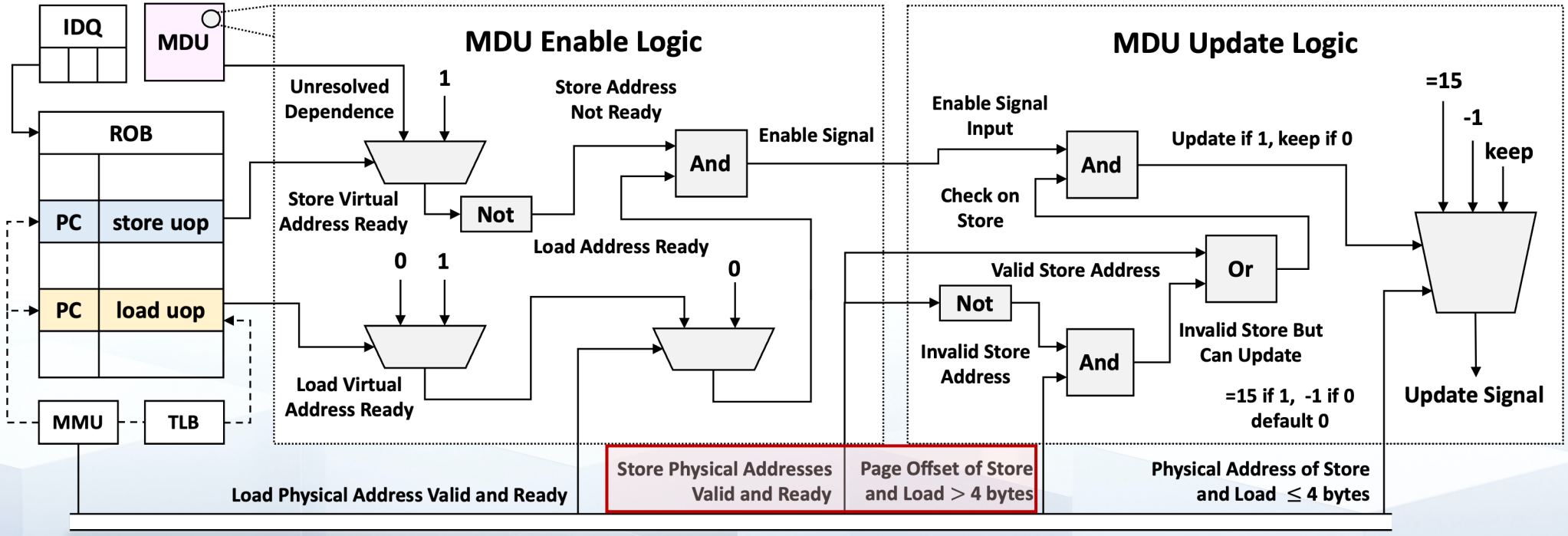
- Store is allocated in the ROB earlier than load
- Unresolved dependence is necessary
- Address of the store is generated slower than load
- Physical address of the load is ready



Characterization: Results and Insights

Insights on Update Condition

- Store is allocated in the ROB earlier than load
- Unresolved dependence is necessary
- Address of the store is generated slower than load
- Physical address of the load is ready
- Both the addresses of the store and load are valid, or the page offset of store and load ≥ 4 bytes



1 Reverse-engineering of MDU Update Logic

2 **Vulnerable Load Identification**

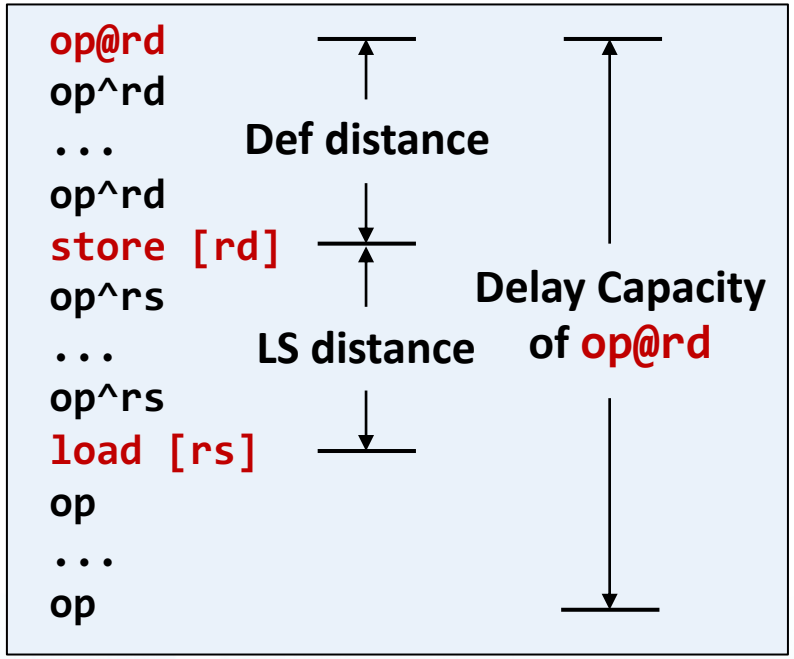
3 MDPeek: End-to-end Attacks

4 Defenses against MDPeek

Modeling Vulnerable Codes

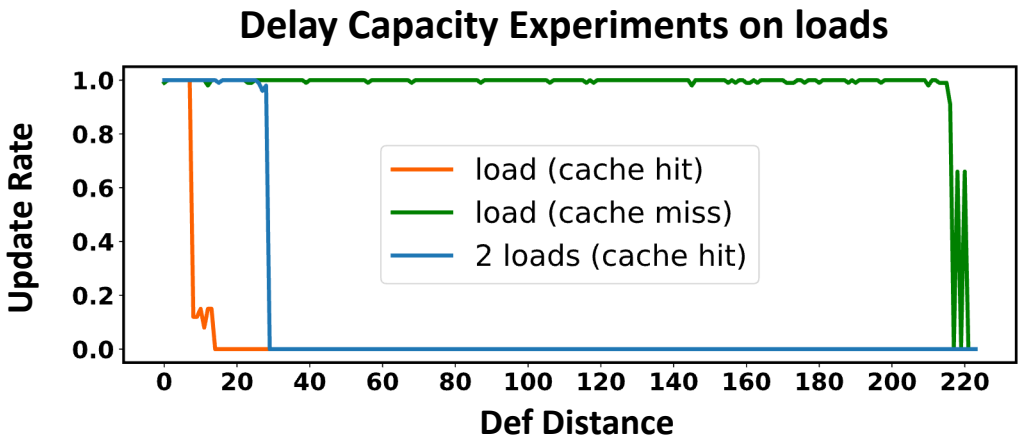
- **Instruction Model**
  - `store [rd]`, `load [rs]`, op@rd, op^rd, op^rs
- **Distance Model**
  - Distance: number of instructions
  - Delay capacity
- **Update Condition**
  - Def distance + LS distance < Delay capacity

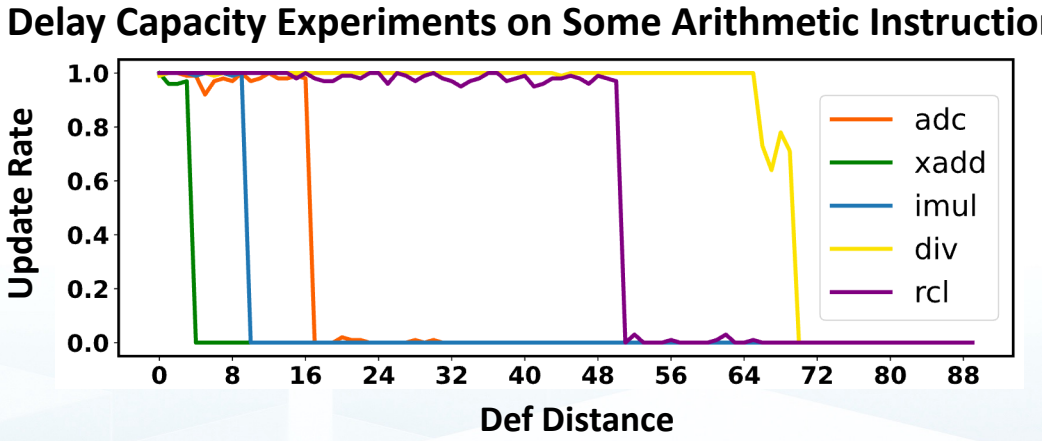
Code Pattern for MDU Update



Modeling Vulnerable Codes

- **Instruction Model**
  - `store [rd]`, `load [rs]`, op@rd, op^rd, op^rs
- **Distance Model**
  - Distance: number of instructions
  - Delay capacity
- **Update Condition**
  - Def distance + LS distance < Delay capacity
- **Precomputed Delay Capacity**
  - Input: `uops.info`
  - Distance computing: using `nop` instructions
  - Cache state of the load
  - Instruction chains





Andreas Abel and Jan Reineke. uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 673–686, 2019.

Modeling Vulnerable Codes

- **Instruction Model**
  - `store [rd]`, `load [rs]`, `op@rd`, `op^rd`, `op^rs`
- **Distance Model**
  - Distance: number of instructions
  - Delay capacity
- **Update Condition**
  - Def distance + LS distance < Delay capacity
- **Precomputed Delay Capacity**
  - Input: `uops.info`
  - Distance computing: using `nop` instructions
  - Cache state of the load
  - Instruction chains
- **Implementation**
  - LLVM v11.0.0

Compute **shortest path** between basic blocks

⇩

Identify **stores** and **loads** inside each basic block, and a **define instruction chain** for each store

⇩

Identify preceding stores for each load, calculate **Def distance** and **LS distance**

⇩

Identify loads that is **potential** to update the MDU

An Example

- WolfSSL (v5.7.2)

mp_invmod_slow

```
1  // if u >= v
2  if (mp_cmp(&u, &v) != MP_LT) {
3      // u = u - v
4      mp_sub(&u, &v, &u);
5  }
6  else {
7      // v = v - u
8      mp_sub(&v, &u, &v);
9  }
```

Secret Dependent Branch

An Example

- WolfSSL (v5.7.2)

mp_invmod_slow

```
1  // if u >= v
2  if (mp_cmp(&u, &v) != MP_LT) {
3      // u = u - v
4      mp_sub(&u, &v, &u);
5  }
6  else {
7      // v = v - u
8      mp_sub(&v, &u, &v);
9  }
```

Secret Dependent Branch

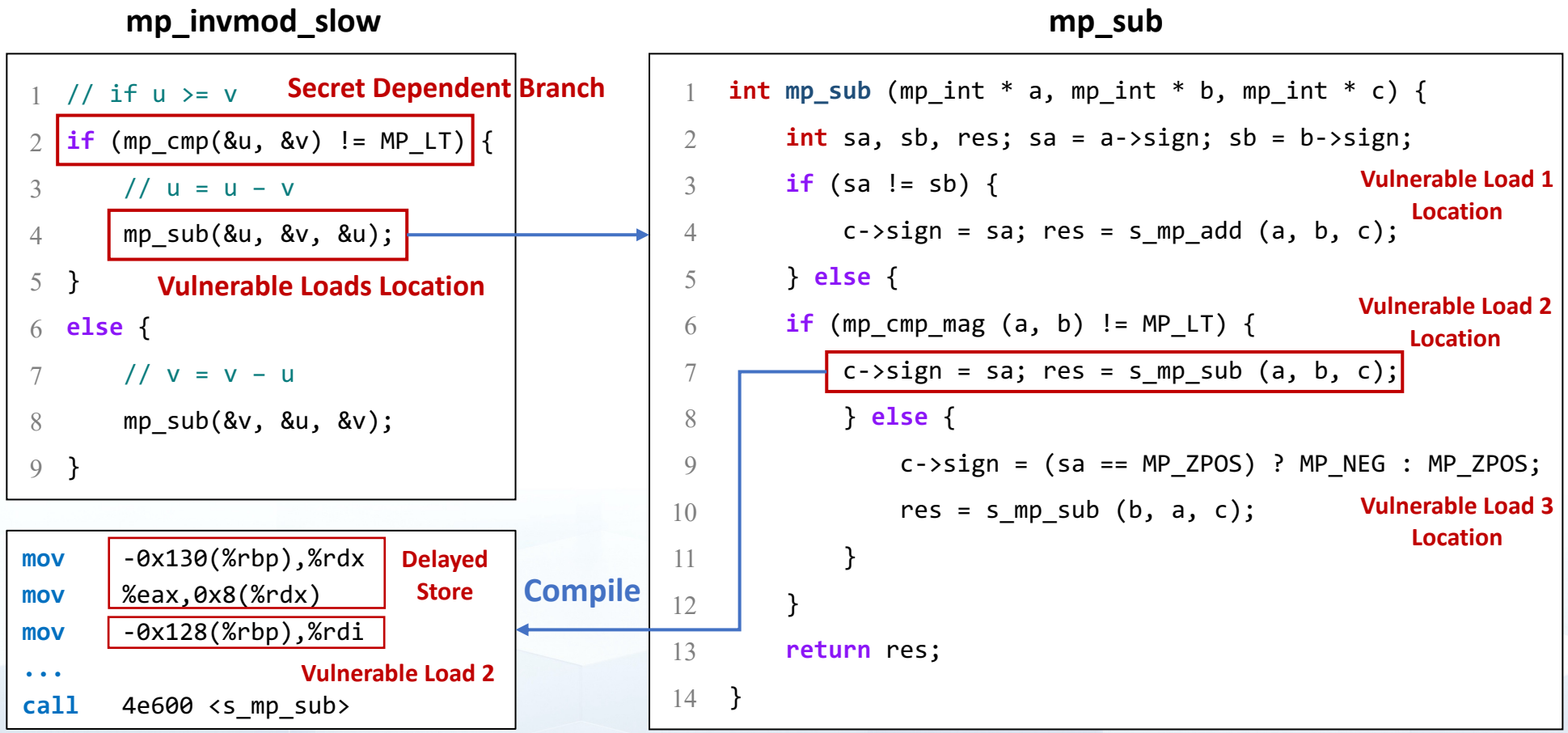
Inline (O3)

mp_sub

```
1  int mp_sub (mp_int * a, mp_int * b, mp_int * c) {
2      int sa, sb, res; sa = a->sign; sb = b->sign;
3      if (sa != sb) {
4          c->sign = sa; res = s_mp_add (a, b, c);
5      } else {
6      if (mp_cmp_mag (a, b) != MP_LT) {
7          c->sign = sa; res = s_mp_sub (a, b, c);
8          } else {
9              c->sign = (sa == MP_ZPOS) ? MP_NEG : MP_ZPOS;
10             res = s_mp_sub (b, a, c);
11         }
12     }
13     return res;
14 }
```

An Example

- WolfSSL (v5.7.2)



1 Reverse-engineering of MDU Update Logic

2 Vulnerable Load Identification

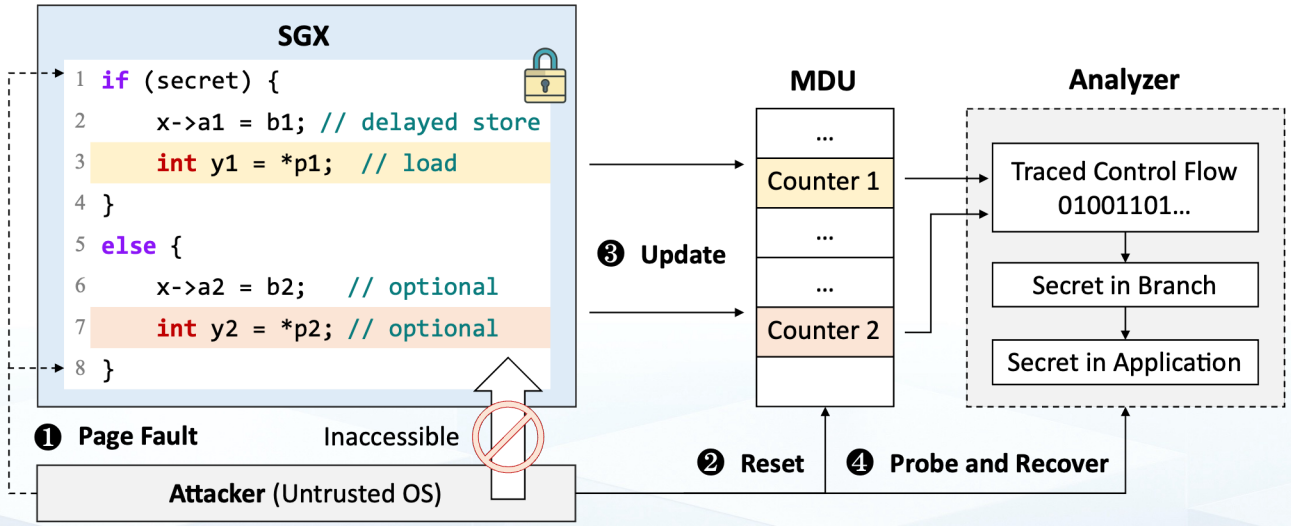
3 **MDPeek: End-to-end Attacks**

4 Defenses against MDPeek

Attack Framework and Primitives

Attack Framework

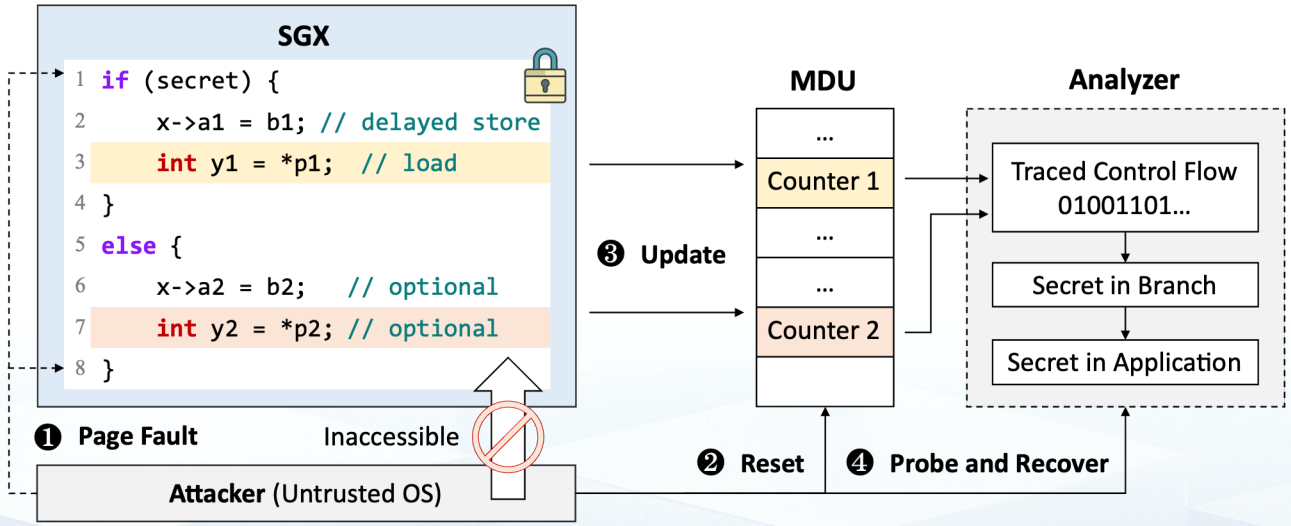
- Synchronization: page fault (with SGX-Step, **page level**)
- Control flow Leakage: MDU counter update (**byte level**)
- Reset: aliased store-load pairs
- Prime: non-aliased store-load pairs

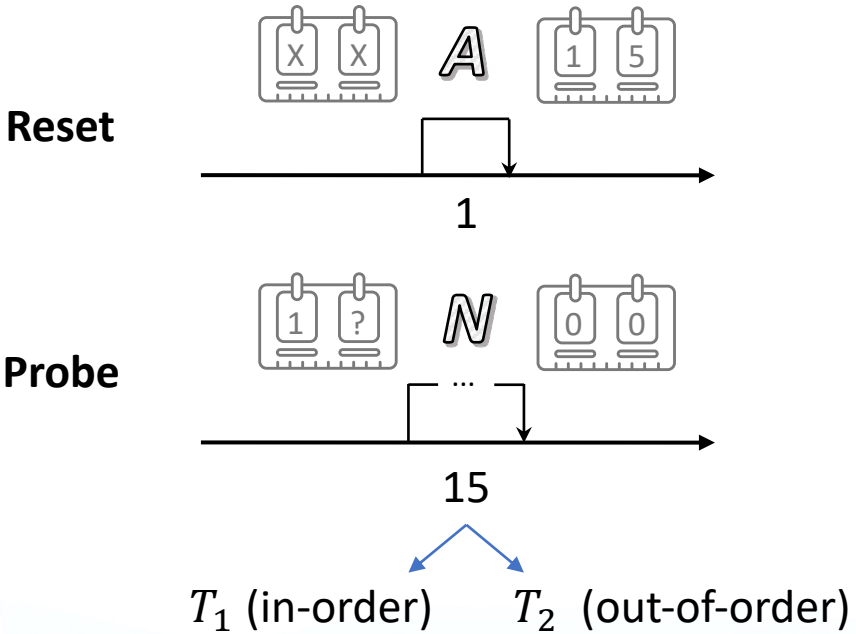


Attack Framework and Primitives

Attack Framework

- Synchronization: page fault (with SGX-Step, **page level**)
- Control flow Leakage: MDU counter update (**byte level**)
- Reset: aliased store-load pairs
- Prime: non-aliased store-load pairs





$|T_1 - T_2| < 150$ cycles

Attack Framework and Primitives

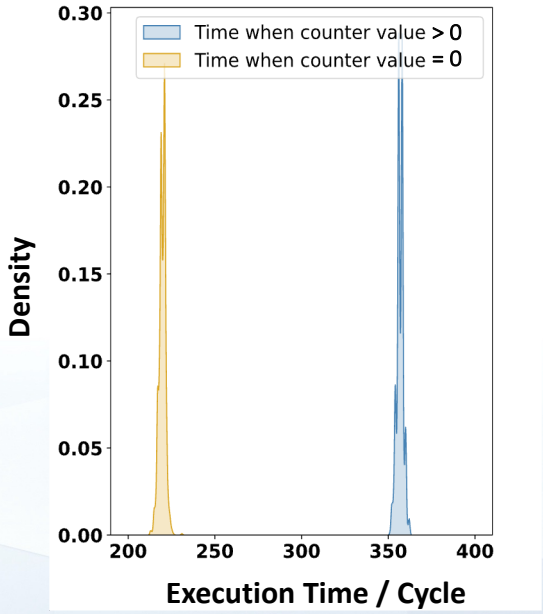
Attack Framework

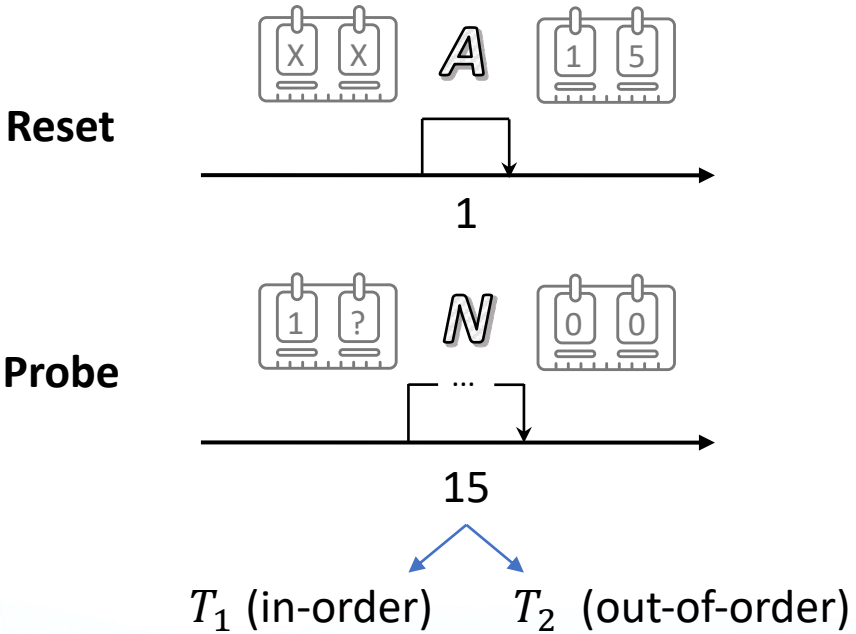
- Synchronization: page fault (with SGX-Step, **page level**)
- Control flow Leakage: MDU counter update (**byte level**)
- Reset: aliased store-load pairs
- Prime: non-aliased store-load pairs

Attack Primitive

```
stld_probe:
  mov     $-8, %r9
  .rep 50
  lea     0x40(%rdi,%r9,8), %rdi
  .endr
  mov     %rdi, (%rdi)
  mov     (%rsi), %rax
  .rep 50
  lea     0x40(%rax,%r9,8), %rax
  .endr
  lfence
  ret
```

Timing Difference



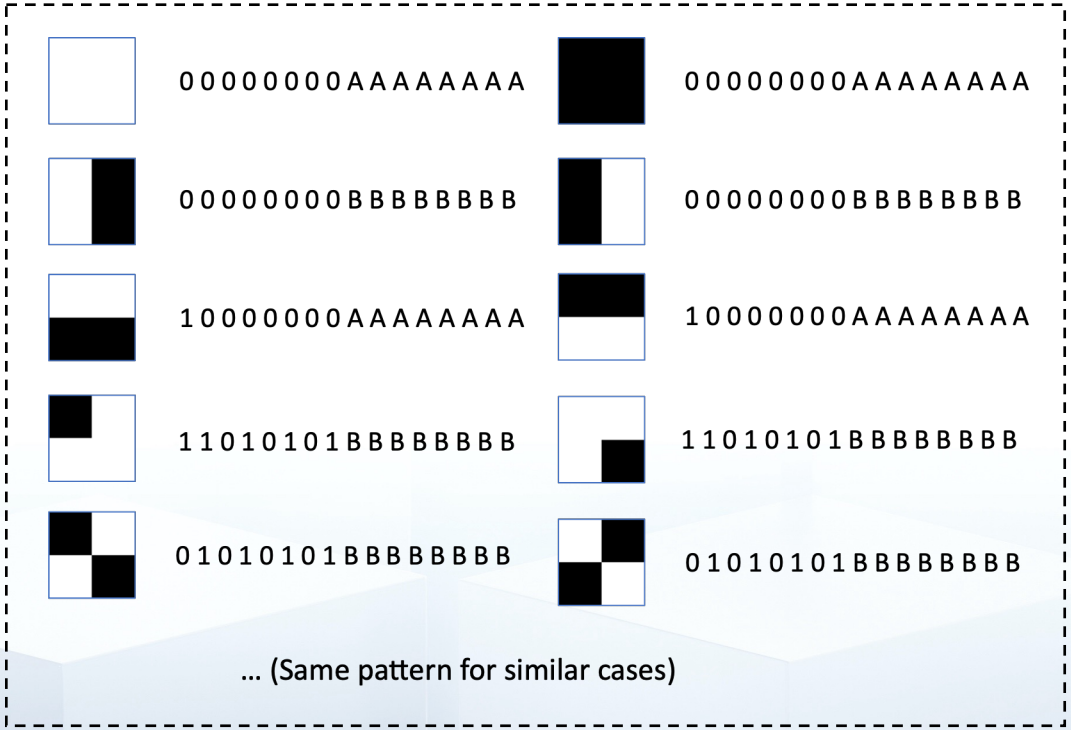


$|T_1 - T_2| < 150$ cycles

Attacking Libjpeg

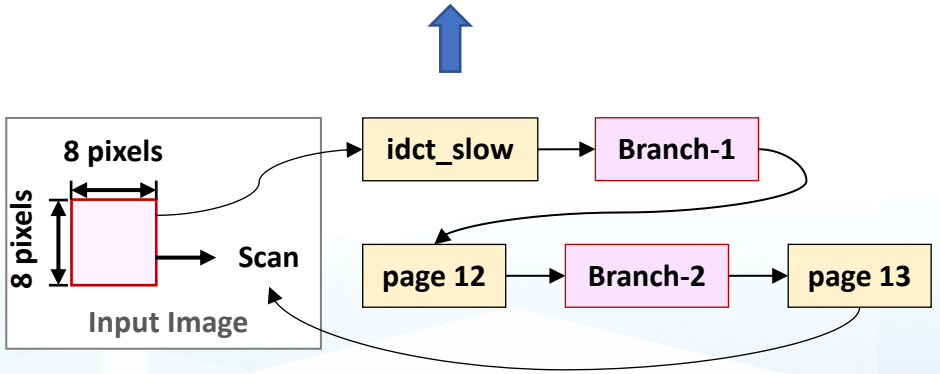
Observation

- IDCT function iterates 8 times for each 8×8 pixel block
- Different pixel layout results in different control flow
- After scaling the image 12×, only 16 layouts are possible



jidcint.c: **jpeg_idct_islow**

```
/* Pass 1: process columns from input, store into work array. */
/* Note results are scaled up by sqrt(8) compared to a true IDCT; */
/* furthermore, we scale the results by 2**PASS1_BITS. */
for (ctr = DCTSIZE; ctr > 0; ctr--) {
  ...
}
/* Pass 2: process rows from work array, store into output array. */
/* Note that we must descale the results by a factor of 8 == 2**3, */
/* and also undo the PASS1_BITS scaling. */
for (ctr = 0; ctr < DCTSIZE; ctr++) {
  ...
}
```



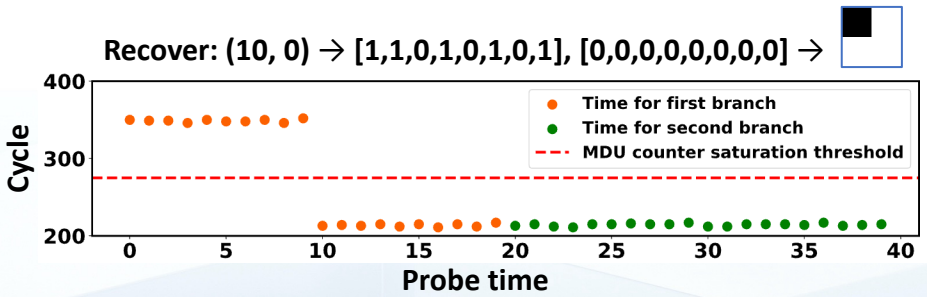
Attacking Libjpeg

Observation

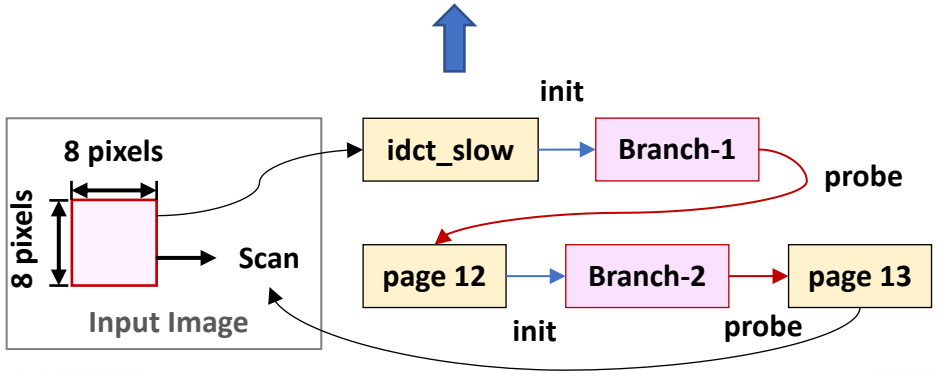
- IDCT function iterates 8 times for each 8×8 pixel block
- Different pixel layout results in different control flow
- After scaling the image 12×, only 16 layouts are possible

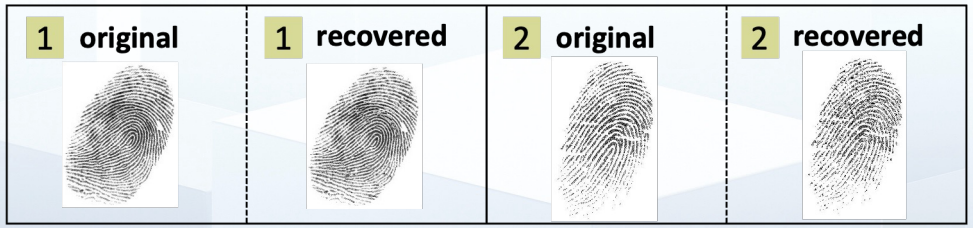
Method

- Synchronize with page faults
- Measure the branch taken through MDU counters
- Recover leaked pixels with pre-computed patterns




```
// Trigger Page Fault Here
/* Pass 1: process columns from input, store into work array. */
/* Note results are scaled up by sqrt(8) compared to a true IDCT; */
/* furthermore, we scale the results by 2**PASS1_BITS. */
for (ctr = DCTSIZE; ctr > 0; ctr--) {
  ...
}
/* Pass 2: process rows from work array, store into output array. */
/* Note that we must descale the results by a factor of 8 == 2**3, */
/* and also undo the PASS1_BITS scaling. */
for (ctr = 0; ctr < DCTSIZE; ctr++) {
  ...
}
// Trigger Page Fault Here
```





Attacking RSA Key Generation

Observation

- Inverse modular (invmod) is used during RSA key generation
- Secrets (p, q or `lcm(p-1,q-1)`) serve as parameters of invmod
- Secret-dependent branches exist in invmod function

WolfSSL v5.7.2

```
int wc_MakeRsaKey(RsaKey* key, int size, long e, WC_RNG* rng)
{   ...
    if (err == MP_OKAY)  /* key->d = 1/e mod lcm(p-1, q-1) */
        err = mp_invmod(&key->e, tmp3, &key->d);
}
```

MbedTLS v3.6.1

```
int mbedtls_rsa_deduce_crt(const mbedtls_mpi *P,
    const mbedtls_mpi *Q, const mbedtls_mpi *D,
    mbedtls_mpi *DP, mbedtls_mpi *DQ, mbedtls_mpi *QP)
{   ...
    if (QP != NULL) {  /* QP = Q^{-1} mod P */
        MBEDTLS_MPI_CHK(mbedtls_mpi_inv_mod(QP, Q, P));
    }
}
```

invmod function

```
1  // if u >= v
2  if (mp_cmp(&u, &v) != MP_LT) {
3      // u = u - v
4      mp_sub(&u, &v, &u);
5  }
6  else {
7      // v = v - u
8      mp_sub(&v, &u, &v);
9  }
```

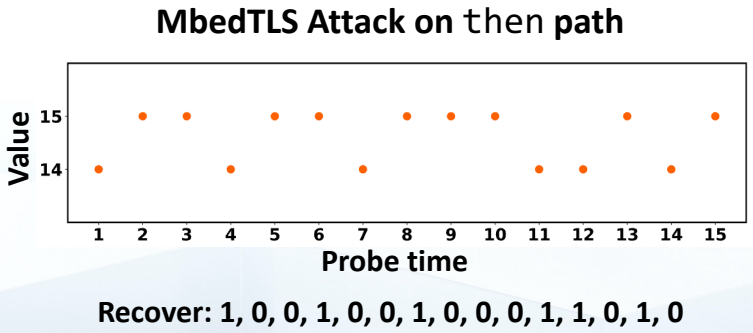
Attacking RSA Key Generation

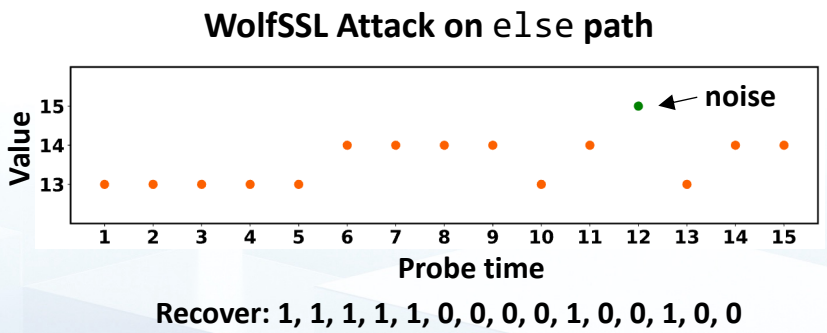
Observation

- Inverse modular (invmod) is used during RSA key generation
- Secrets (p, q or `lcm(p-1,q-1)`) serve as parameters of invmod
- Secret-dependent branches exist in invmod function

Evaluation

- 1000 attacks on 2048-bit key
- MbedTLS: 830 ms for a single trace, with success rate exceeding 97%
- WolfSSL: 880 ms for a single trace, with success rate exceeding 95%





1 Reverse-engineering of MDU Update Logic

2 Vulnerable Load Identification

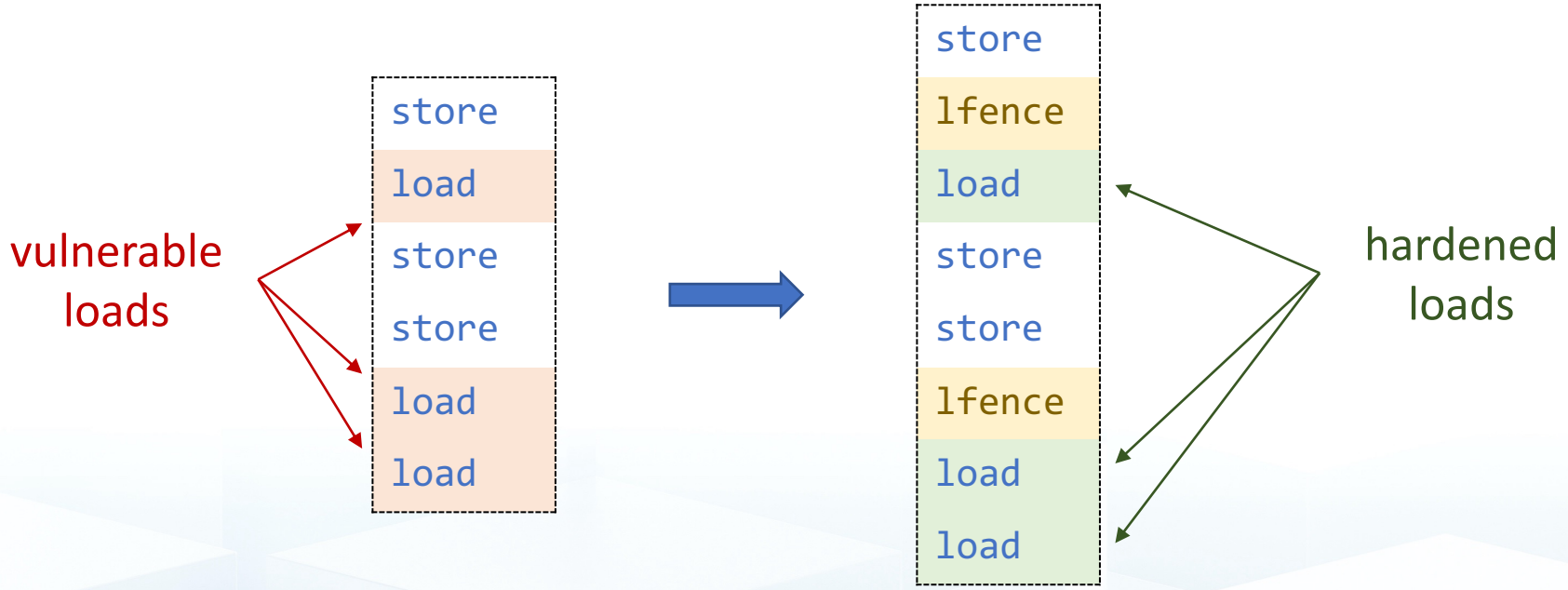
3 MDPeek: End-to-end Attacks

4 **Defenses against MDPeek**

Defenses: Serialization and Alignment

Serialization

- Insight: MDU is enabled only when both a delayed store and load are allocated in the **ROB**
- Method: Insert an `lfence` instruction between a potential delayed store and following loads
- Performance Overhead: **~140%**



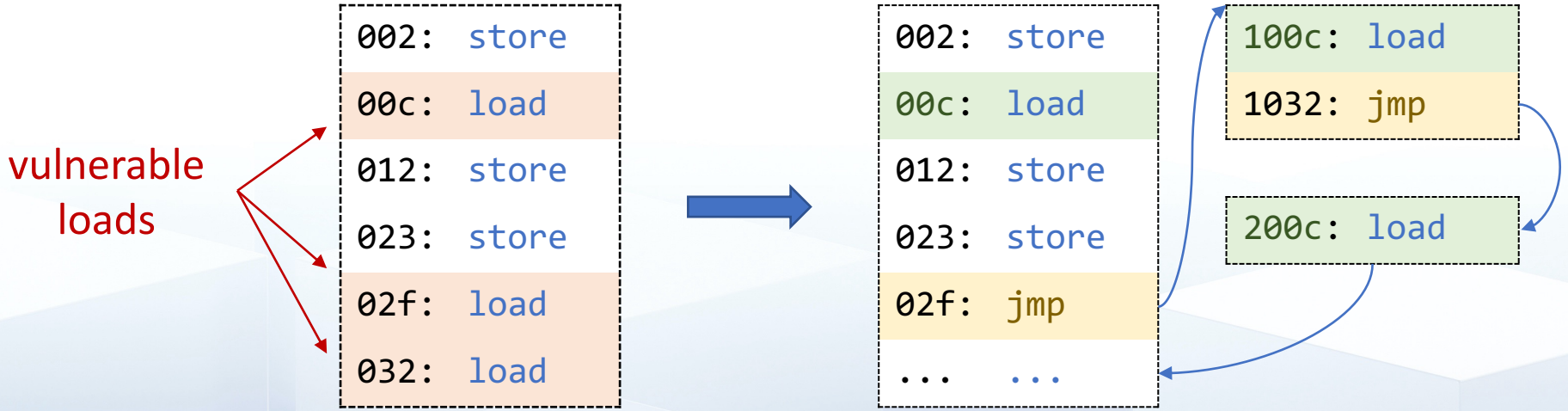
Defenses: Serialization and Alignment

Serialization

- Insight: MDU is enabled only when both a delayed store and load are allocated in the **ROB**
- Method: Insert an **`lfence`** instruction between a potential delayed store and following loads
- Performance Overhead: **~140%**

Alignment

- Insight: MDU is selected by the **lowest 8 bits** of the load PC
- Method: Align the load PC to **256 bytes** by inserting **`nop`** instructions
- Performance Overhead: **~160%**



Defenses: Store-to-load Coupling

- **Insight**
  - Unresolved data dependence is necessary to update the MDU
  - Making the dependence explicit to the CPU
  - Adding deterministic dependence between the store and load addresses

vulnerable load

```
mdu_update_dependence_1:
    movq (%rdi), %rdi
    movq $0, (%rdi)
    nop
    nop
    nop
    movq (%rsi), %rsi
    lfence
    ret
```

Update rate: 100%

Explicit Dependence:

`%rsi = %rdi | 0`

```
mdu_update_dependence_2:
    movq (%rdi), %rdi
    movq $0, (%rdi)
    mov %rdi, %rax
    and $0, %rax
    or %rax, %rsi
    movq (%rsi), %rsi
    lfence
    ret
```

Update rate: 0

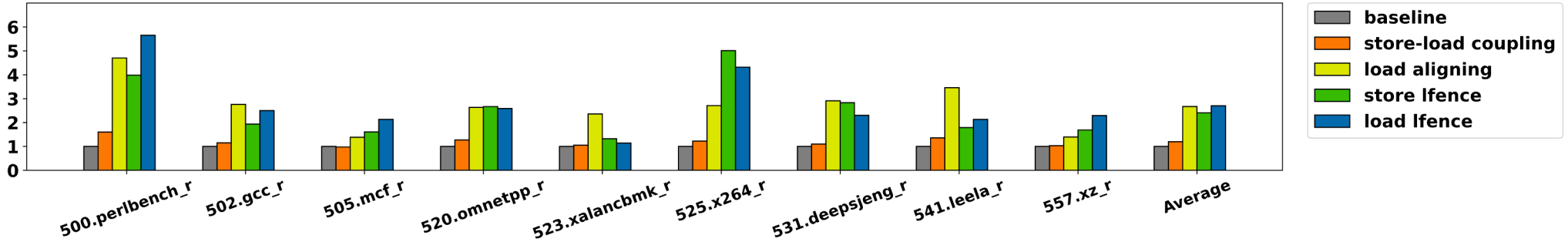
Defenses: Store-to-load Coupling

Insight

- Unresolved data dependence is necessary to update the MDU
- Making the dependence explicit to the CPU
- Adding deterministic dependence between the store and load addresses

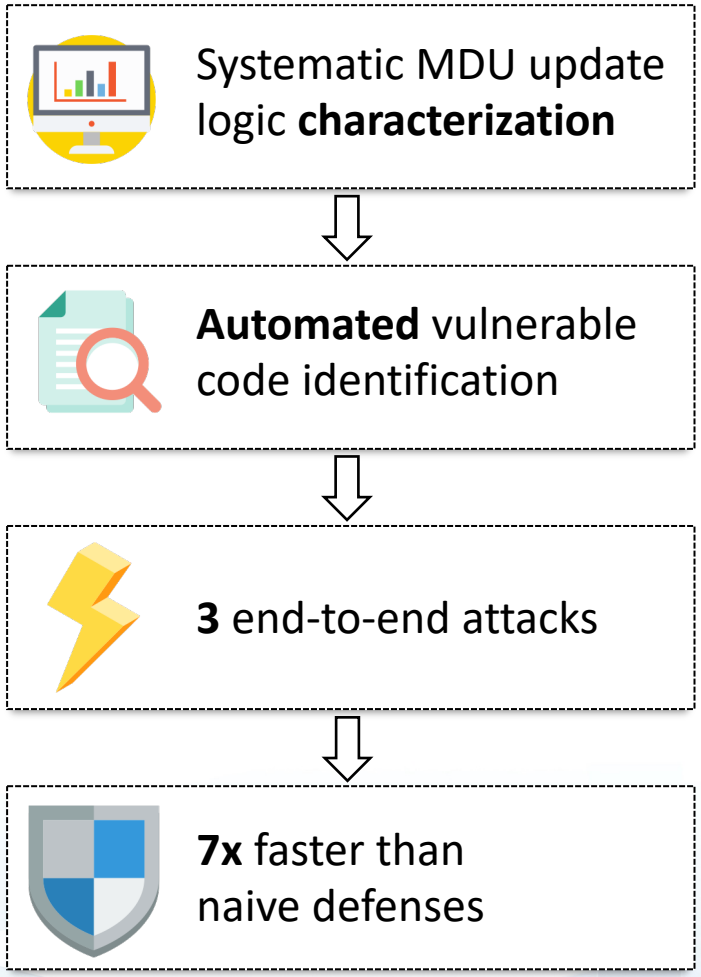
Evaluation

- Serialization: **~140%**
- Alignment: **~160%**
- Store-to-load Coupling: **~20%**



Conclusion

- **Systematic MDU Characterization**
  - Update condition
  - Interaction with cache, TLB and ROB
  - Multiple stores and loads
  - NOT isolated between the normal and secure world
- **Vulnerable Loads Identification in Real-world Applications**
  - Modeling address generation delay
  - Measuring Delay Capacity
- **MDPeek: End-to-end Attacks with MDU Side Channel**
  - Attacking Libjpeg
  - Attacking RSA Generation (MbedTLS and WolfSSL)
- **Defenses against MDPeek**
  - Naive defenses
  - Store-to-load Coupling



Tsinghua University

NUS National University of Singapore

Thanks

MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels

Chang Liu, Shuaihu Feng, Yuan Li, Dongsheng Wang, Wenjian He, Yongqiang Lyu and **Trevor E. Carlson**

Questions?