SSBench: Automated Characterization of Memory Dependence Predictors on Modern CPUs

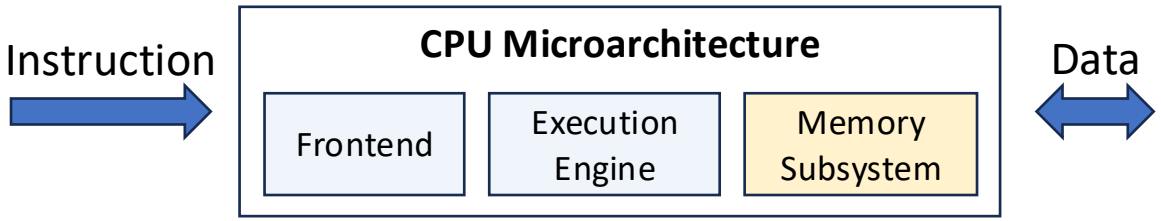
Chang Liu, **Yu Jin**, Yuchen Fan, Tianrui Xiao, Lingfeng Yin, Trevor E. Carlson, Shuwen Deng, Dongsheng Wang

*Presenter

ISCA 2026

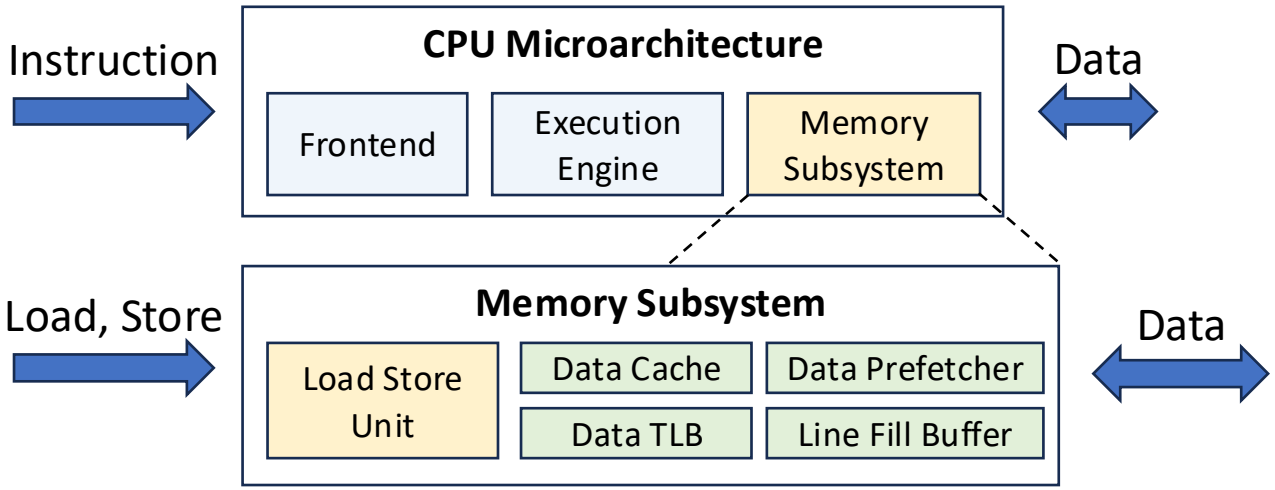
Introduction: Load Store Unit

- **Memory Subsystem**
  - Communication between CPU and memory



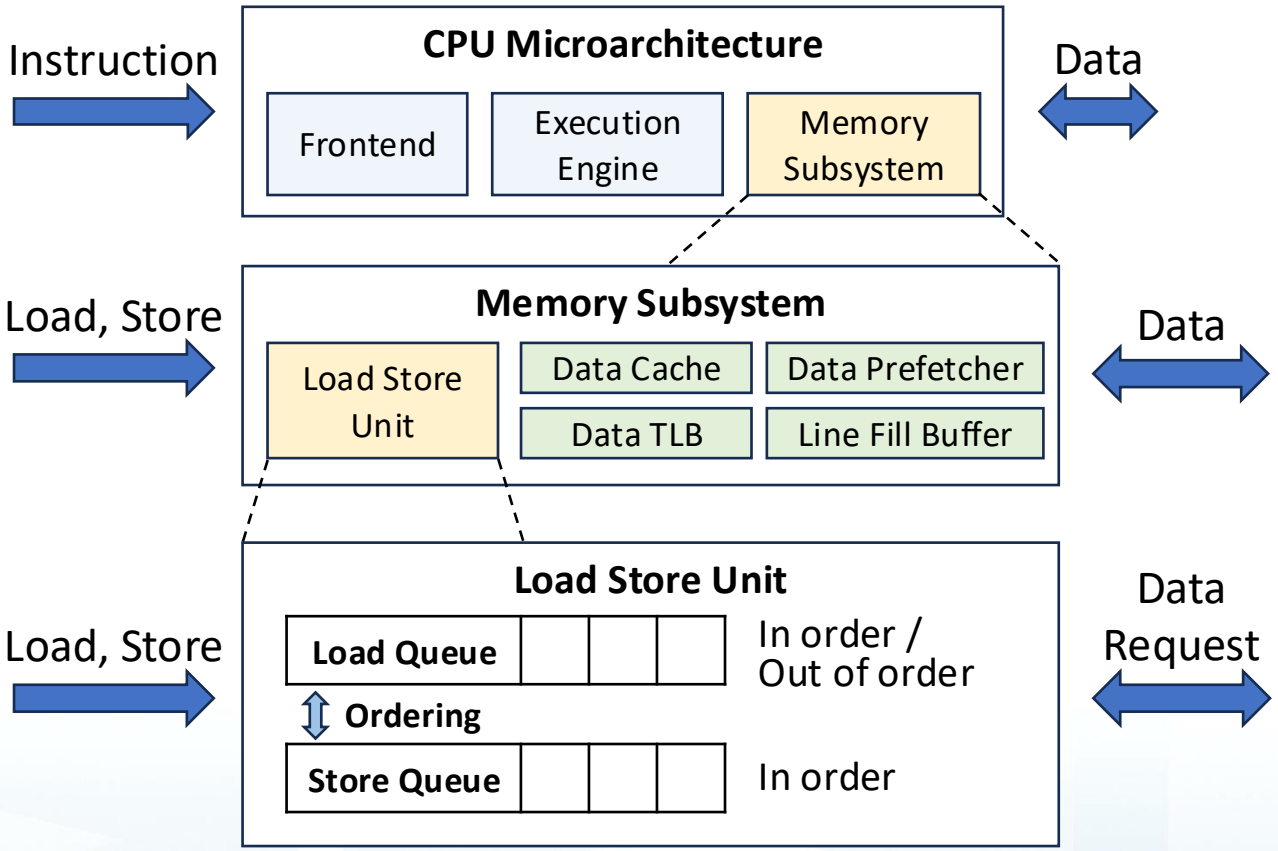
Introduction: Load Store Unit

- **Memory Subsystem**
  - Communication between CPU and memory
  - Memory Hierarchy: Speed up memory access
  - Load Store Unit: Execute loads and stores



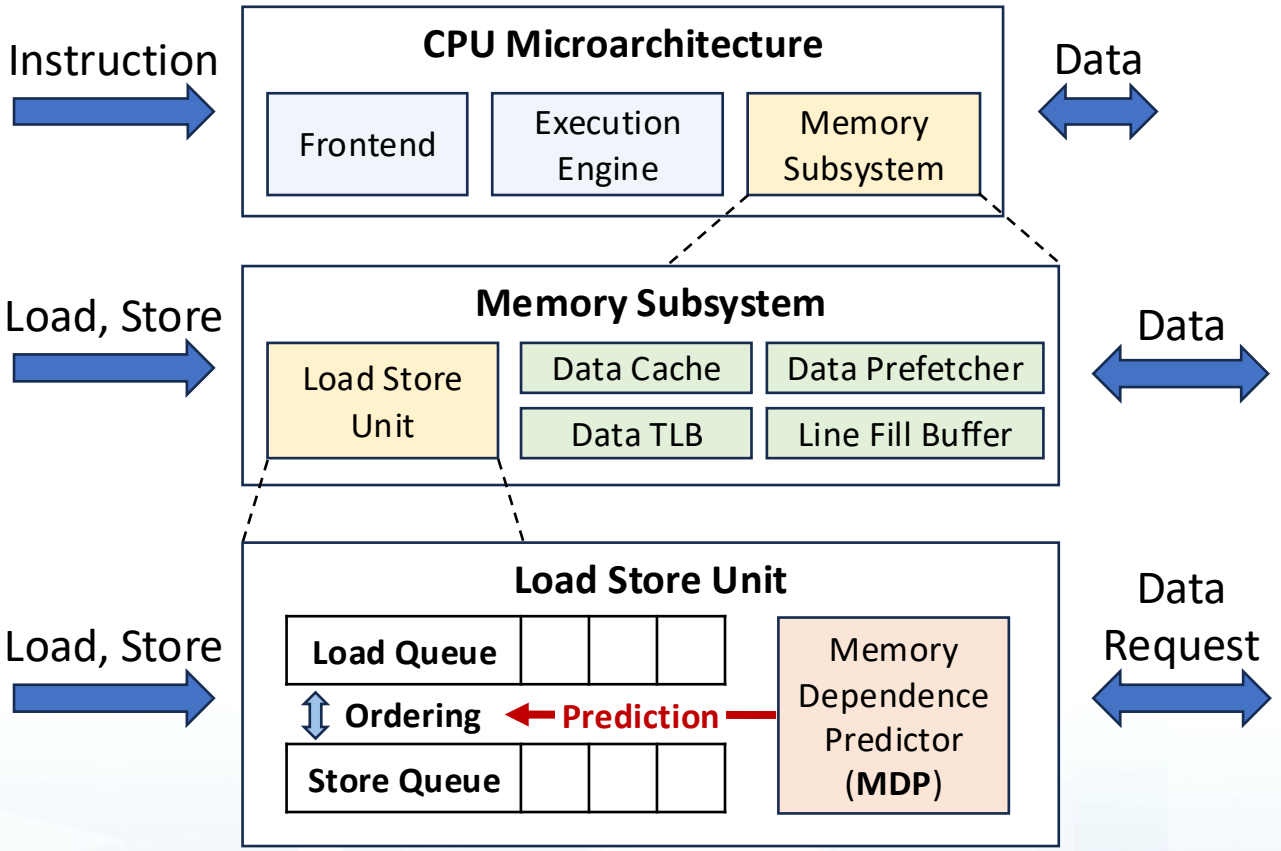
Introduction: Load Store Unit

- **Memory Subsystem**
  - Communication between CPU and memory
  - Memory Hierarchy: Speed up memory access
  - Load Store Unit: Execute loads and stores
- **Load Store Unit**
  - Address generation and translation
  - Memory access
  - **Memory ordering**
    - Load Queue, Store Queue



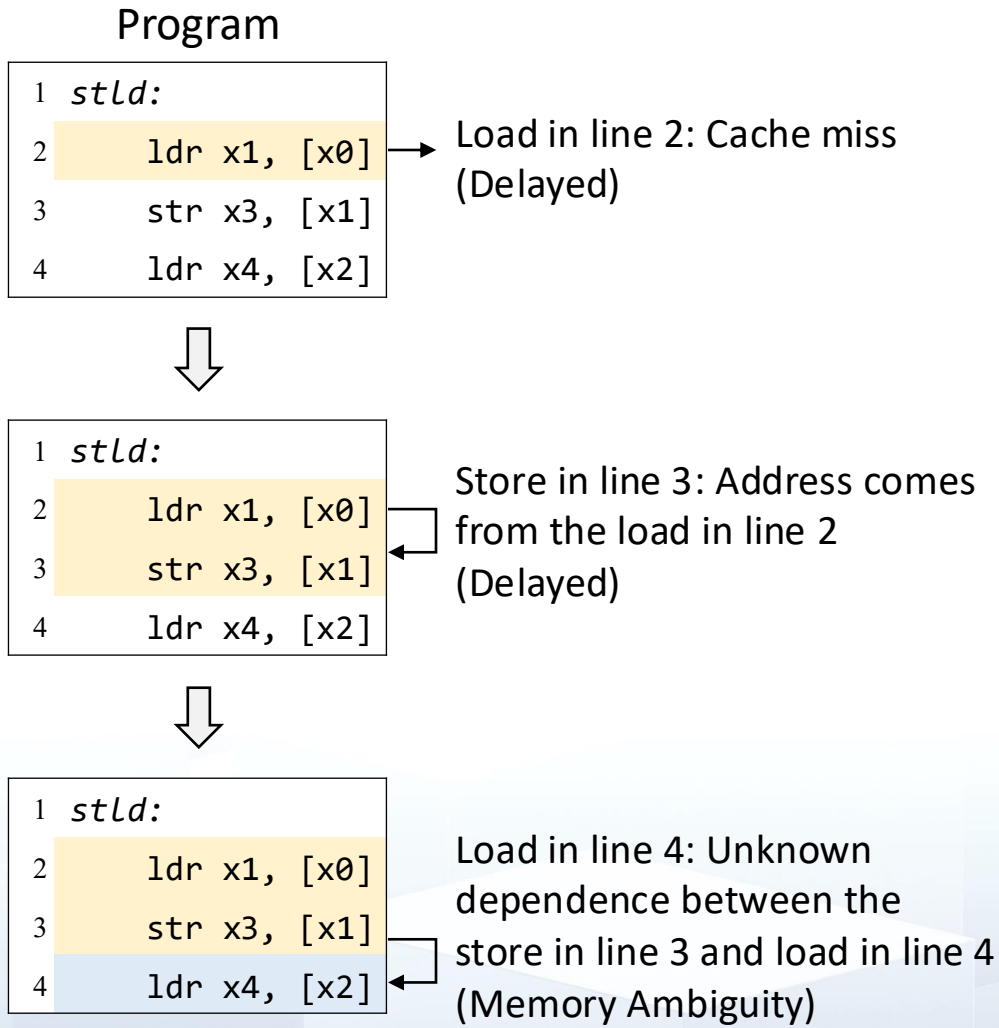
Introduction: Load Store Unit

- **Memory Subsystem**
  - Communication between CPU and memory
  - Memory Hierarchy: Speed up memory access
  - Load Store Unit: Execute loads and stores
- **Load Store Unit**
  - Address generation and translation
  - Memory access
  - **Memory ordering**
    - Load Queue, Store Queue
  - **Memory Dependence Predictor (MDP)**
    - Increase instruction-level parallelism
    - Support memory ordering speculation



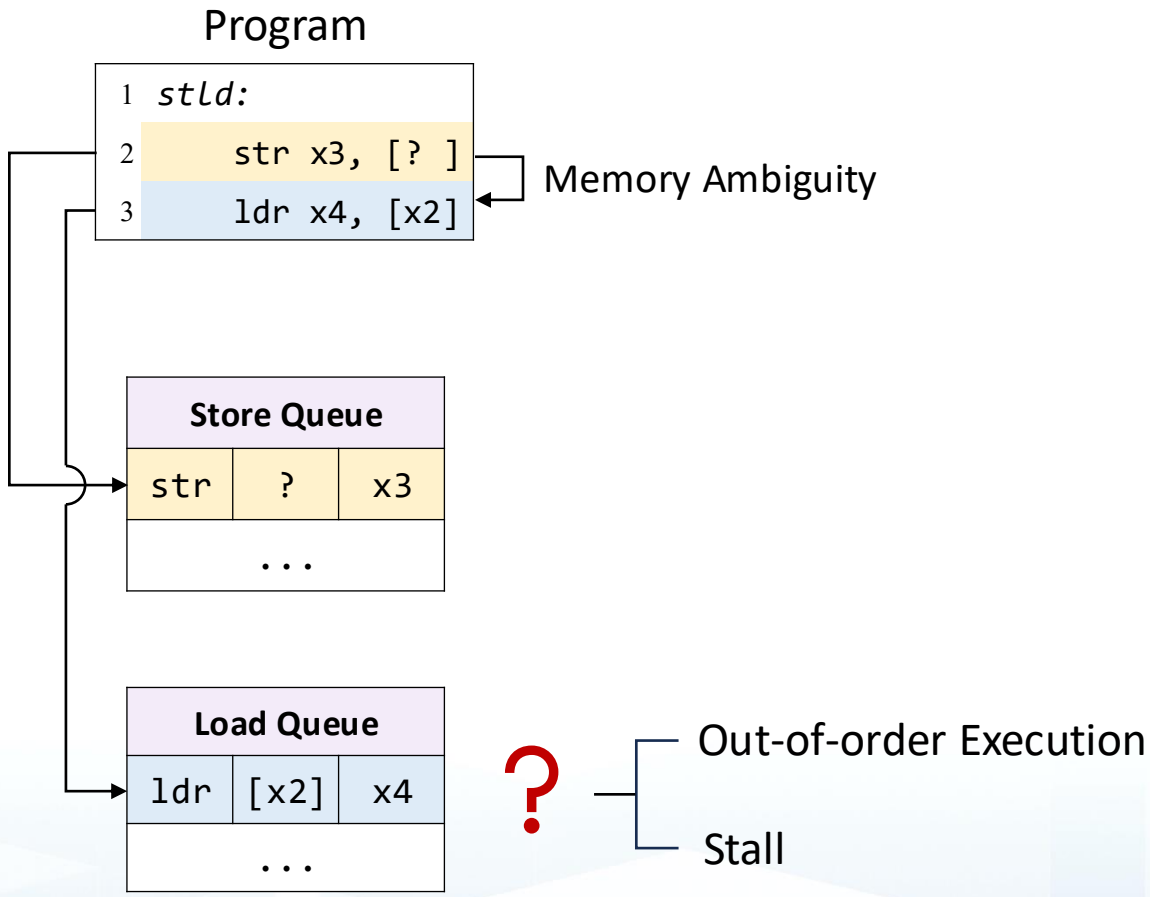
Introduction: Memory Dependence Predictor

- **Memory Ambiguity**
  - Unresolved data dependence
  - Read-after-write (RAW)



Introduction: Memory Dependence Predictor

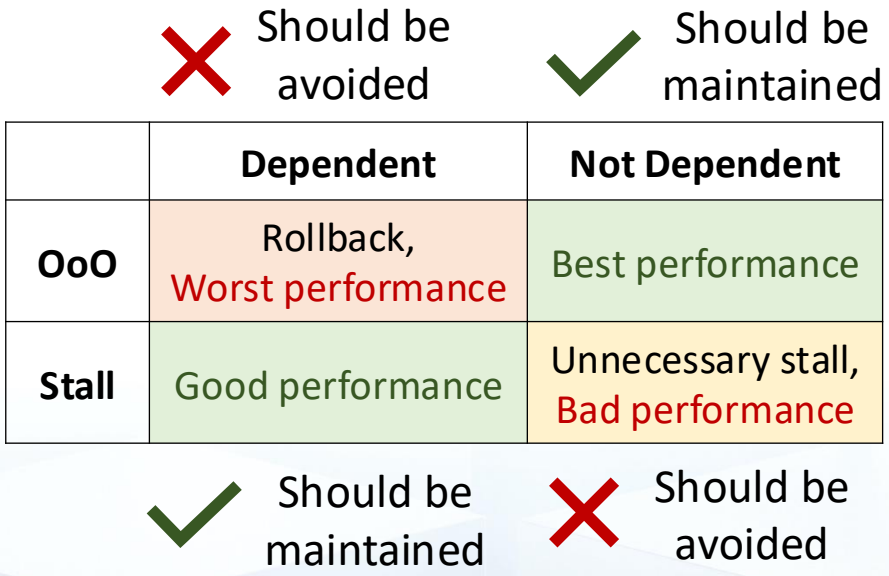
- **Memory Ambiguity**
  - Unresolved data dependence
  - Read-after-write (RAW)
  - Reduction of ILP due to stalled loads

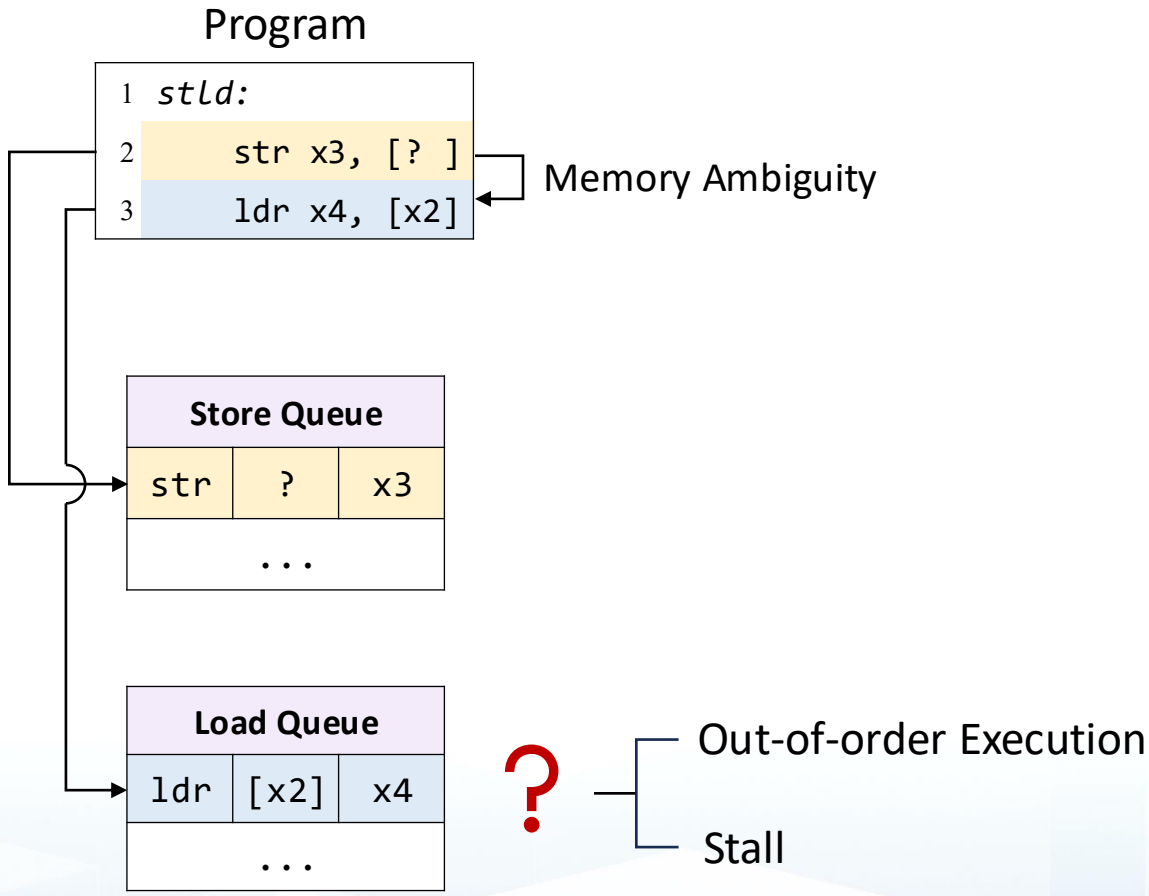


Introduction: Memory Dependence Predictor

Memory Ambiguity

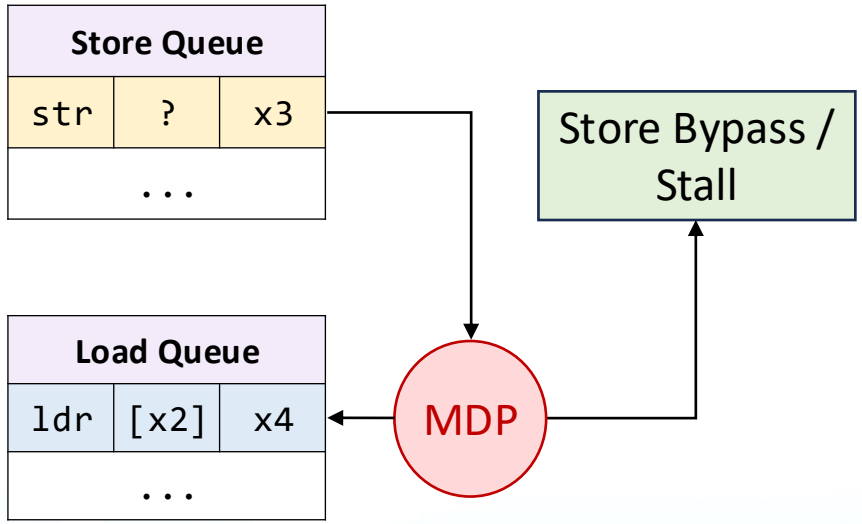
- Unresolved data dependence
- Read-after-write (RAW)
- Reduction of ILP due to stalled loads





Introduction: Memory Dependence Predictor

- **Memory Ambiguity**
  - Unresolved data dependence
  - Read-after-write (RAW)
  - Affect ILP by stalling loads
- **MDP**
  - Target: Data dependence of a store-load pair
  - Trigger:
    - Store address is not ready
    - Load address is ready and translated
  - Prediction:
    - Dependent: Load should be stalled
    - Not dependent: Load can be executed out of order before preceding stores

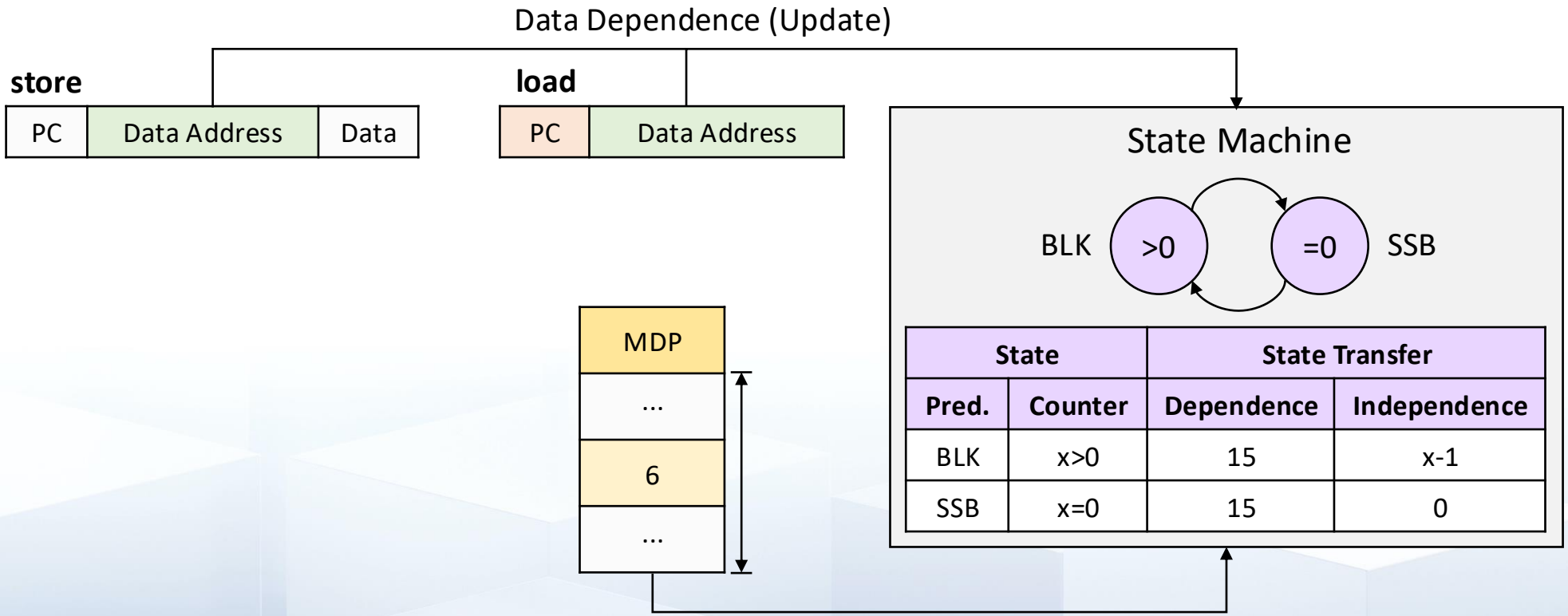


Background: MDP on Intel Skylake CPUs

- **MDP on Intel Skylake[1]**
  - State machine: 4-bit counter

Reference

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

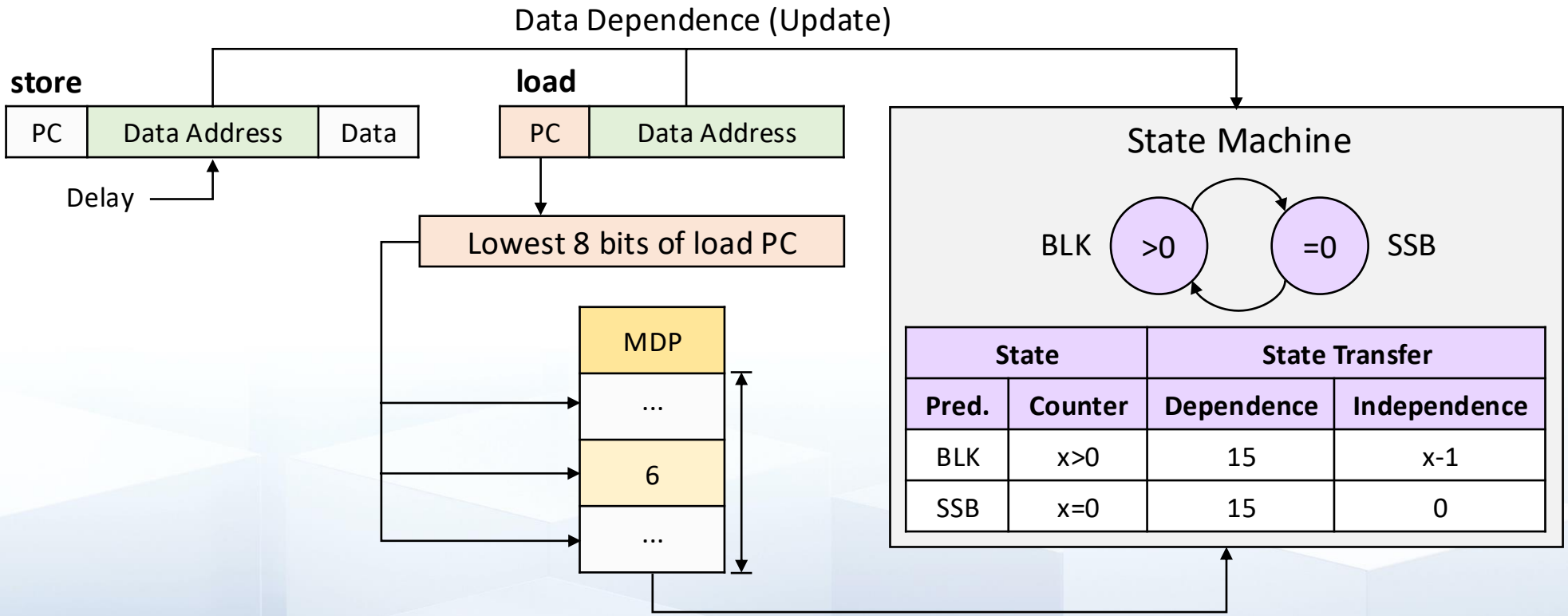


Background: MDP on Intel Skylake CPUs

- **MDP on Intel Skylake[1]**
  - State machine: 4-bit counter
  - Hash function: Lowest 8 bits of load PC

Reference

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

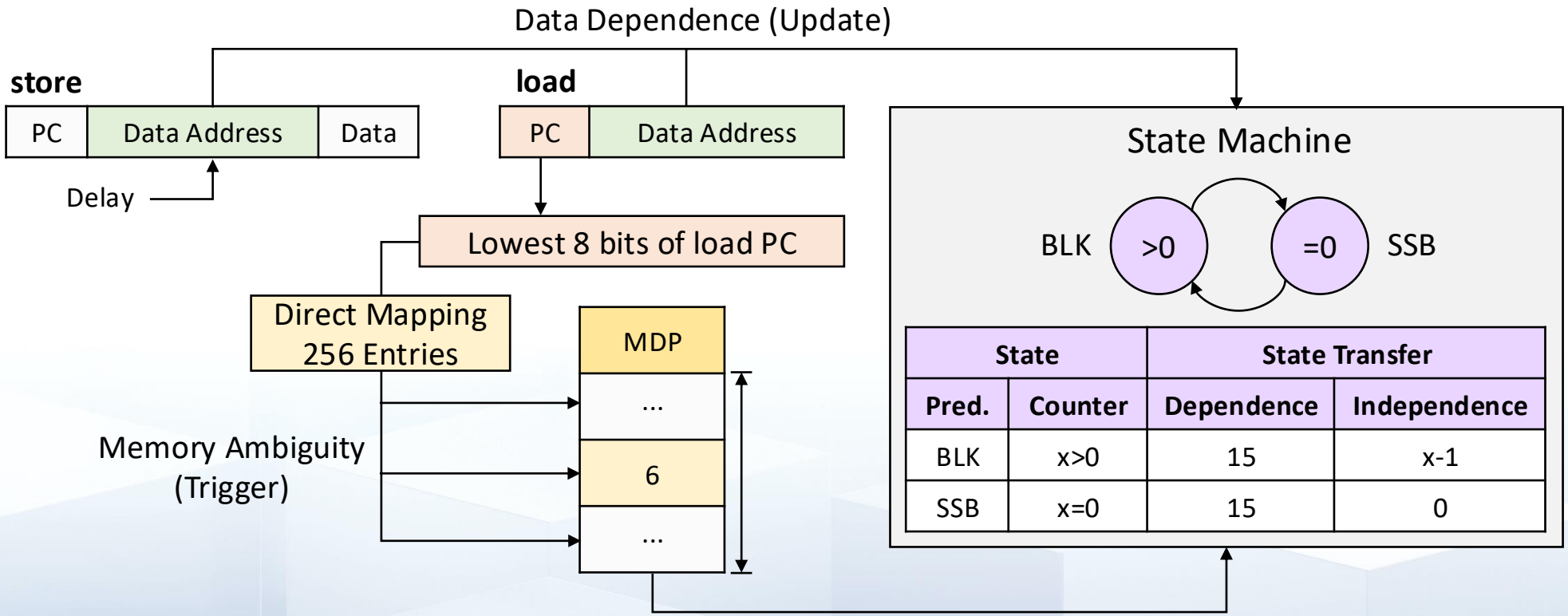


Background: MDP on Intel Skylake CPUs

- **MDP on Intel Skylake[1]**
  - State machine: 4-bit counter
  - Hash function: Lowest 8 bits of load PC
  - Organization: 256 entries, fully-associative

Reference

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.



Motivation: MDP Security

- **MDP Security**
  - **Spectre Attacks**
    - Misspeculation of a load
    - Intel: Spectre-v4 attacks in browser sandboxes[1]
    - AMD: Cross-process Spectre-v4 attacks[2]
  - **Control-flow leakage of loads**
    - Intel: Leak byte-level control flow information within Intel SGX[3]
    - Arm: Website fingerprinting[4]

References

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

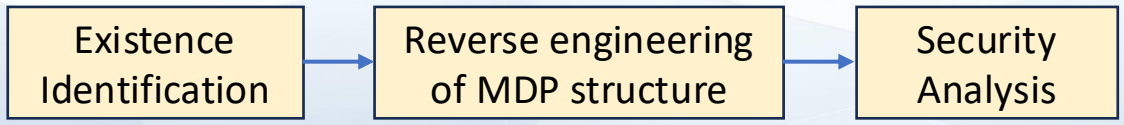
[2] C. Liu, D. Wang, Y. Lyu, P. Qiu, Y. Jin, Z. Lu, Y. Zhang, and G. Qu, "Uncovering and Exploiting AMD Speculative Memory Access Predictors for Fun and Profit," in International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 31–45.

[3] C. Liu, S. Feng, Y. Li, D. Wang, W. He, Y. Lyu, and T. E. Carlson, "MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025, pp. 622–638.

[4] C. Liu, Z. Li, H. Wang, P. Qiu, G. Qu, and D. Wang, "Exploiting ARMeD Channels by Reverse Engineering ARM Memory Disambiguation Unit," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 45, no. 2, pp. 1075–1088, 2026.

Motivation: MDP Security

- **MDP Security**
  - **Spectre Attacks**
    - Misspeculation of a load
    - Intel: Spectre-v4 attacks in browser sandboxes[1]
    - AMD: Cross-process Spectre-v4 attacks[2]
  - **Control-flow leakage of loads**
    - Intel: Leak byte-level control flow information within Intel SGX[3]
    - Arm: Website fingerprinting[4]
- **How to study MDP security?**
  - **Manual** identification of MDP
  - **Manual** reverse engineering of MDP structure
  - **Manual** analysis of MDP security



References

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

[2] C. Liu, D. Wang, Y. Lyu, P. Qiu, Y. Jin, Z. Lu, Y. Zhang, and G. Qu, "Uncovering and Exploiting AMD Speculative Memory Access Predictors for Fun and Profit," in International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 31–45.

[3] C. Liu, S. Feng, Y. Li, D. Wang, W. He, Y. Lyu, and T. E. Carlson, "MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025, pp. 622–638.

[4] C. Liu, Z. Li, H. Wang, P. Qiu, G. Qu, and D. Wang, "Exploiting ARMeD Channels by Reverse Engineering ARM Memory Disambiguation Unit," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 45, no. 2, pp. 1075–1088, 2026.

Motivation: MDP Security

- **MDP Security**
  - Spectre Attacks
    - Misspeculation of a load
    - Intel: Spectre-v4 attacks in browser sandboxes[1]
    - AMD: Cross-process Spectre-v4 attacks[2]
  - Control-flow leakage of loads
    - Intel: Leak byte-level control flow information within Intel SGX[3]
    - Arm: Website fingerprinting[4]
- **How to study MDP security?**
  - **Manual** identification of MDP
  - **Manual** reverse engineering of MDP structure
  - **Manual** analysis of MDP security
  - **Several limitations**

References

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

[2] C. Liu, D. Wang, Y. Lyu, P. Qiu, Y. Jin, Z. Lu, Y. Zhang, and G. Qu, "Uncovering and Exploiting AMD Speculative Memory Access Predictors for Fun and Profit," in International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 31–45.

[3] C. Liu, S. Feng, Y. Li, D. Wang, W. He, Y. Lyu, and T. E. Carlson, "MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025, pp. 622–638.

[4] C. Liu, Z. Li, H. Wang, P. Qiu, G. Qu, and D. Wang, "Exploiting ARMeD Channels by Reverse Engineering ARM Memory Disambiguation Unit," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 45, no. 2, pp. 1075–1088, 2026.

Incredibly time-consuming

Lack of generalization

Unable to test in batches

Motivation: MDP Security

- **MDP Security**
  - Spectre Attacks
    - Misspeculation of a load
    - Intel: Spectre-v4 attacks in browser sandboxes[1]
    - AMD: Cross-process Spectre-v4 attacks[2]
  - Control-flow leakage of loads
    - Intel: Leak byte-level control flow information within Intel SGX[3]
    - Arm: Website fingerprinting[4]
- **How to study MDP security?**
  - **Manual** identification of MDP
  - **Manual** reverse engineering of MDP structure
  - **Manual** analysis of MDP security
  - **Several limitations**
  - **Question**: How to further improve MDP research?

References

[1] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks," in USENIX Security, 2021, pp. 1451–1468.

[2] C. Liu, D. Wang, Y. Lyu, P. Qiu, Y. Jin, Z. Lu, Y. Zhang, and G. Qu, "Uncovering and Exploiting AMD Speculative Memory Access Predictors for Fun and Profit," in International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 31–45.

[3] C. Liu, S. Feng, Y. Li, D. Wang, W. He, Y. Lyu, and T. E. Carlson, "MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025, pp. 622–638.

[4] C. Liu, Z. Li, H. Wang, P. Qiu, G. Qu, and D. Wang, "Exploiting ARMeD Channels by Reverse Engineering ARM Memory Disambiguation Unit," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 45, no. 2, pp. 1075–1088, 2026.

Incredibly time-consuming

Lack of generalization

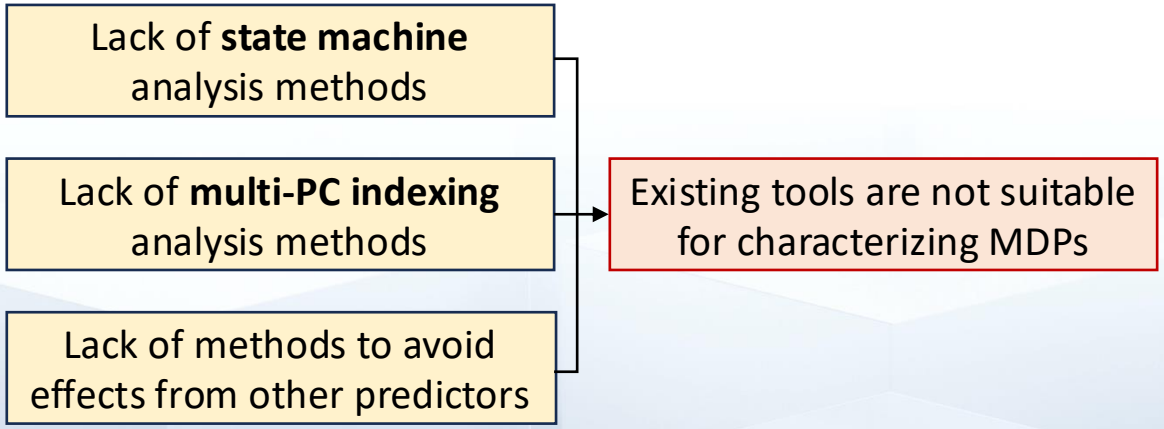
Unable to test in batches

MDP Benchmark

Automated Characterization

Motivation: Related Works

- **Existing Tools**
  - **Characterization of µarch structures**
    - X-ray[1]: TLB / Cache
    - FetchBench[2]: Data prefetcher
    - Stride RE[3]: Stride prefetcher
    - Plumber[4]: Cache / CBP
  - **Characterization of µarch parameters**
    - Hash Function[5,6]: Cache, TLB, DRAM
    - Replacement Policy[7]: Cache



References

[1] K. Yotov, K. Pingali, and P. Stodghill, "Automatic measurement of memory hierarchy parameters," in Proceedings of the SIGMETRICS international conference on Measurement and modeling of computer systems, 2005, pp. 181–192.

[2] T. Schlüter, A. Choudhari, L. Hetterich, L. Trampert, H. Nemati, A. Ibrahim, M. Schwarz, C. Rossow, and N. O. Tippenhauer, "Fetch-Bench: Systematic Identification and Characterization of Proprietary Prefetchers," in Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS), 2023, pp. 975–989.

[3] L. Hetterich, F. Thomas, L. Gerlach, R. Zhang, N. Bernsdorf, E. Ebert, and M. Schwarz, "ShadowLoad: Injecting State into Hardware Prefetchers," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025, pp. 1060–1075.

[4] A. Ibrahim, H. Nemati, T. Schluter, N. O. Tippenhauer, and C. Rossow, "Microarchitectural Leakage Templates and Their Application to Cache-Based Side Channels," in Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS), 2022, pp. 1489–1503.

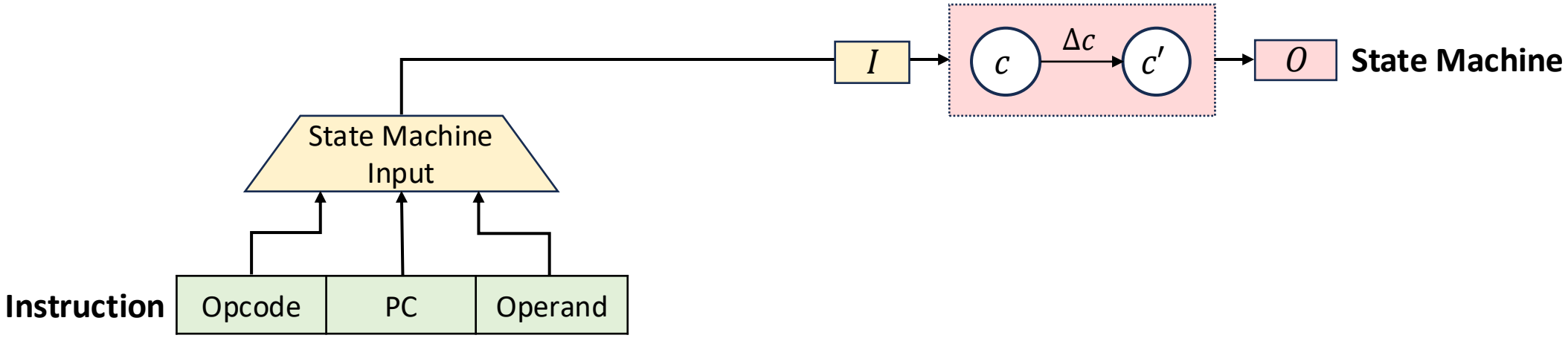
[5] L. Gerlach, S. Schwarz, N. Faroß, and M. Schwarz, "Efficient and Generic Microarchitectural Hash-Function Recovery," in Symposium on Security and Privacy (S&P), 2024.

[6] M. Rainer, L. Hetterich, F. Thomas, T. Hornetz, L. Trampert, L. Gerlach, and M. Schwarz, "Rapid Reversing of Non-Linear CPU Cache Slice Functions: Unlocking Physical Address Leakage," in Symposium on Security and Privacy (S&P), 2025, pp. 3497–3515.

[7] A. Abel and J. Reineke, "Measurement-based Modeling of the Cache Replacement Policy," in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 65–74.

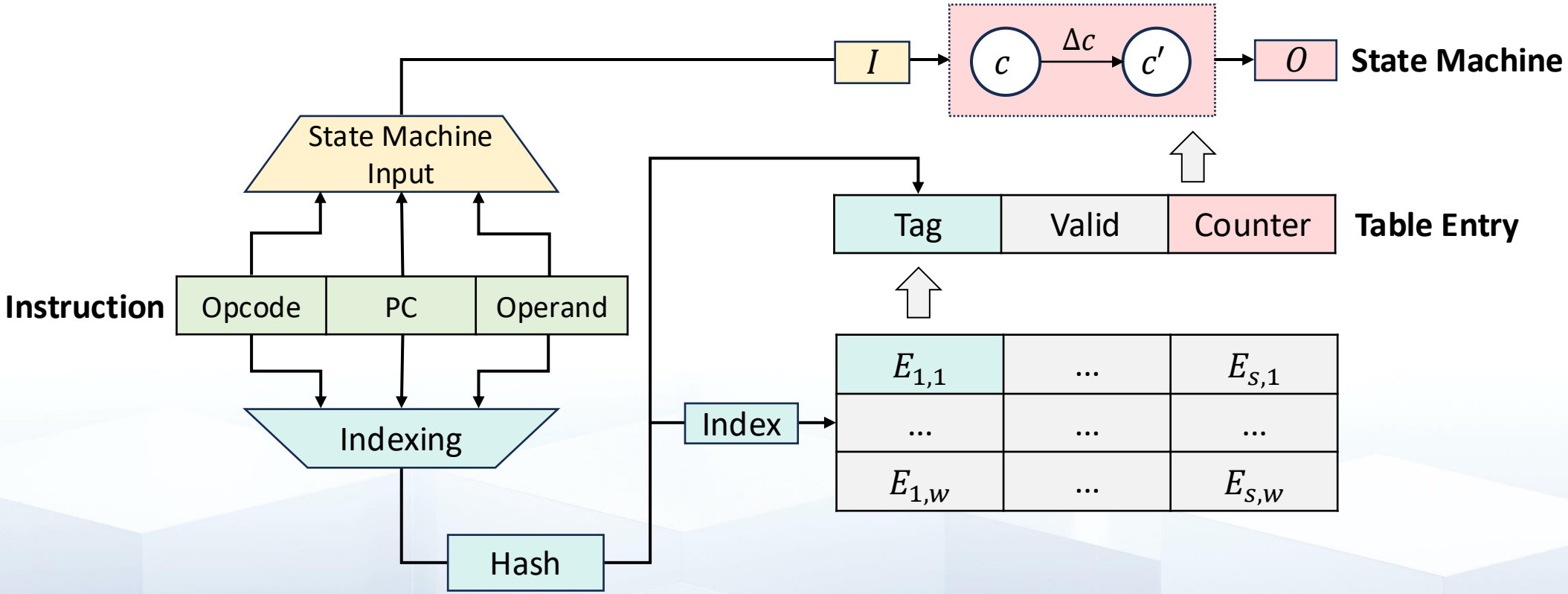
Method: Modeling a Predictor

- **Key parameters**
  - State machine



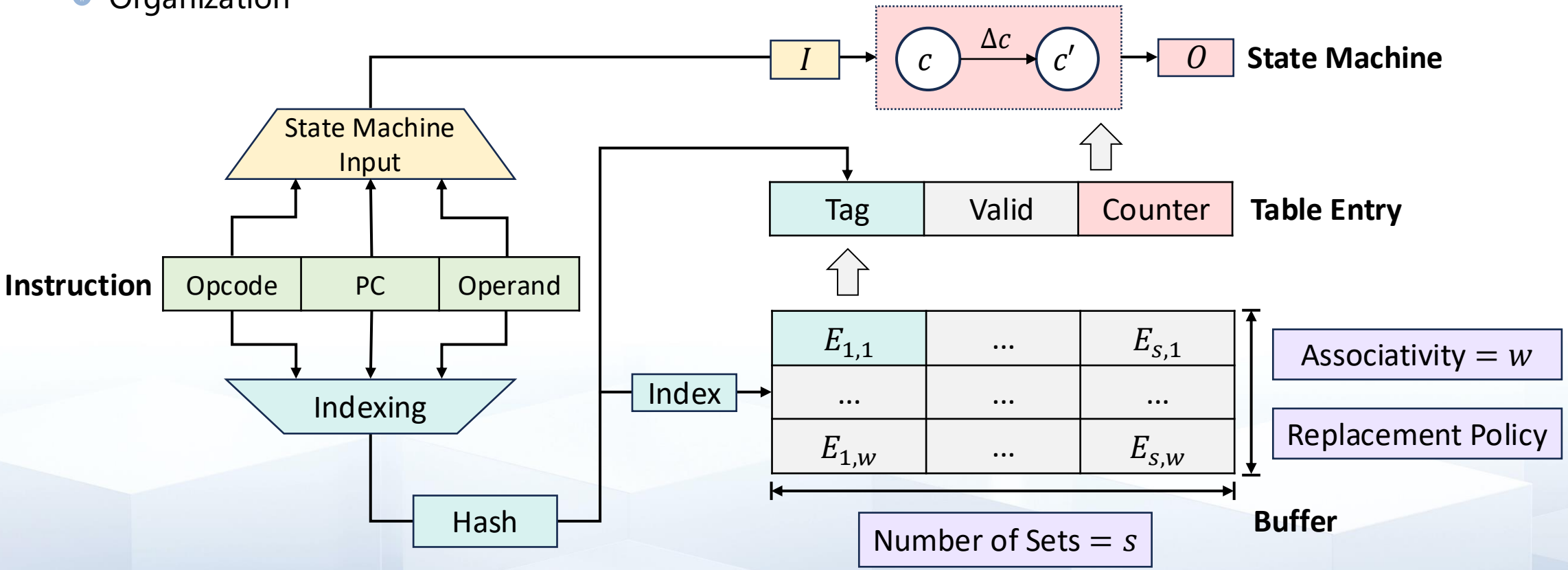
Method: Modeling a Predictor

- **Key parameters**
  - State machine
  - Indexing



Method: Modeling a Predictor

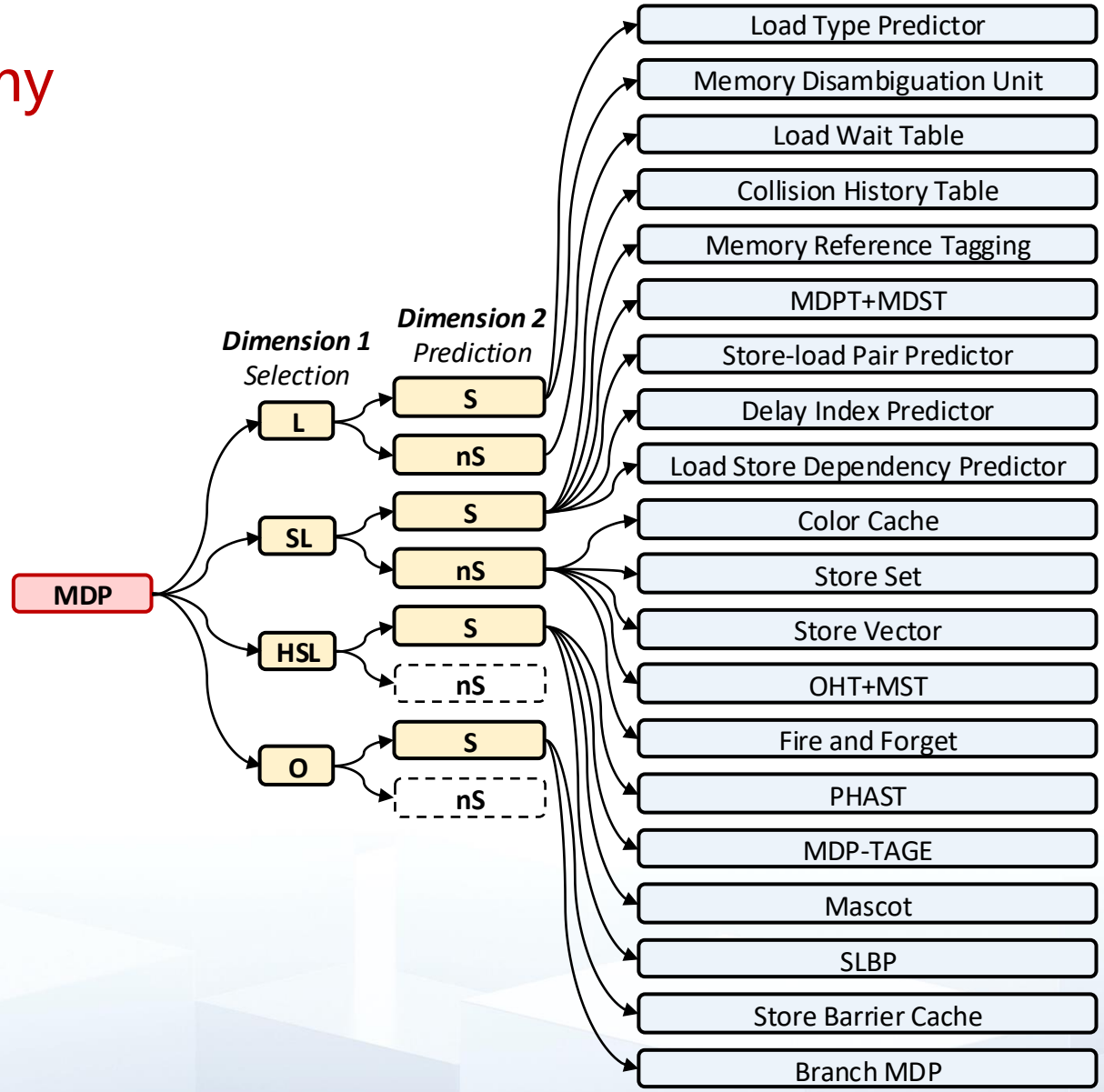
- **Key parameters**
  - State machine
  - Indexing
  - Organization



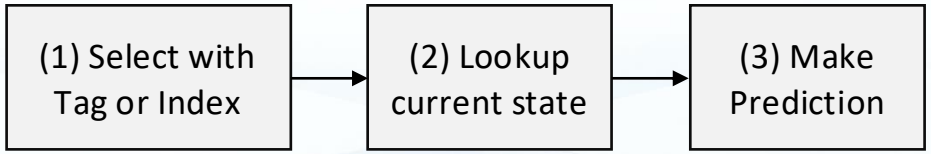
Method: Workflow-based Taxonomy

Two-Dimensional Taxonomy

- Indexing / Selection
  - L: Load PC
  - SL: Store and load PCs
  - HSL: PC and branch history
  - O: Others
- Prediction / State Machine
  - S: Stateful
  - nS: Stateless



Workflow of MDP



Method: Microbenchmark

Microbenchmark in x86-64

- Delayed store
  - Line 3: 10 multiply instructions
  - Line 5: Store 0 to [%rdi]
- Fast load
  - Line 7: Load from [%rsi]
  - Line 10: Consume load data

Code for MDP Probing (Pair)

```
1  stld:
2  .rep 10
3      imul $1, %rdi
4  .endr
5  movq $0, (%rdi)
6  ld_start:
7  movq (%rsi), %rax
8  ld_end:
9  .rep 10
10     imul $1, %rax
11 .endr
```

Method: Microbenchmark

- **Microbenchmark in x86-64**
  - Delayed store
    - Line 3: 10 multiply instructions
    - Line 5: Store 0 to [%rdi]
  - Fast load
    - Line 7: Load from [%rsi]
    - Line 10: Consume load data
- **Function Caller**
  - **Timing** the microbenchmark
  - Avoid address and value prediction
    - Random addresses
    - Random load value

Code for MDP Probing (Pair)

```
1  stld:
2  .rep 10
3      imul $1, %rdi
4  .endr
5  movq $0, (%rdi)
6  ld_start:
7  movq (%rsi), %rax
8  ld_end:
9  .rep 10
10     imul $1, %rax
11 .endr
```

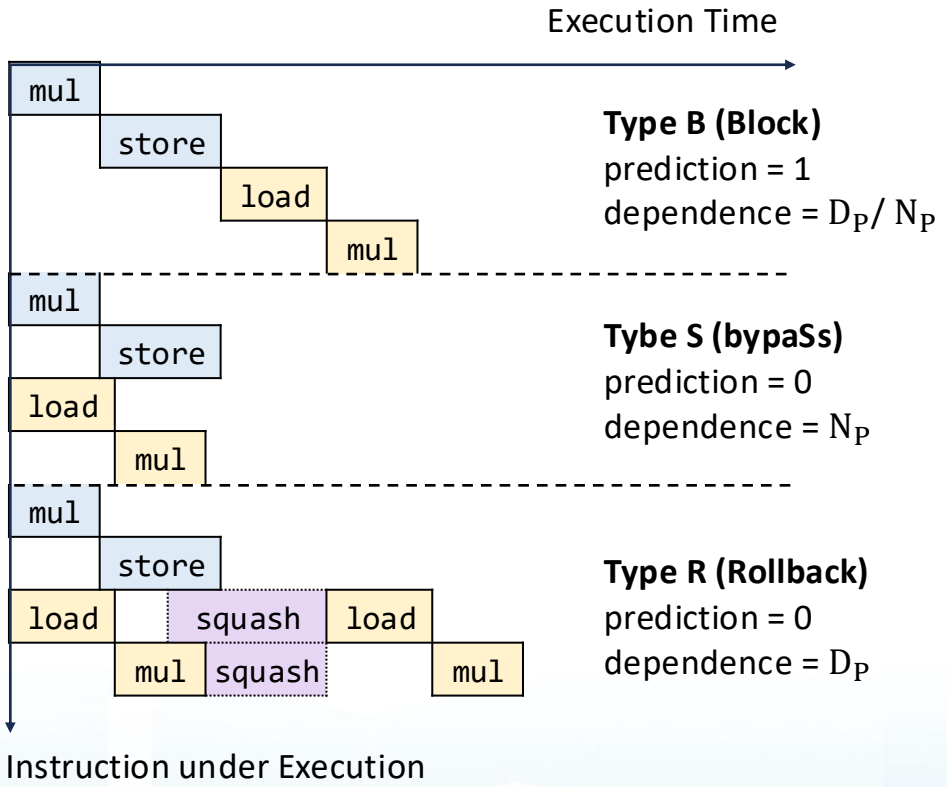
Function Caller in C Code

```
1  size_t stld_call(int dep) {
2    int* st_addr = rand_addr();
3    int* ld_addr = st_addr;
4    if (!dep) ld_addr += 2;
5    int v = rand_val();
6    *ld_addr = *st_addr = v;
7    size_t t1 = time();
8    stld(st_addr, ld_addr);
9    size_t t2 = time();
10   return t2 - t1;
11 }
```

Timing

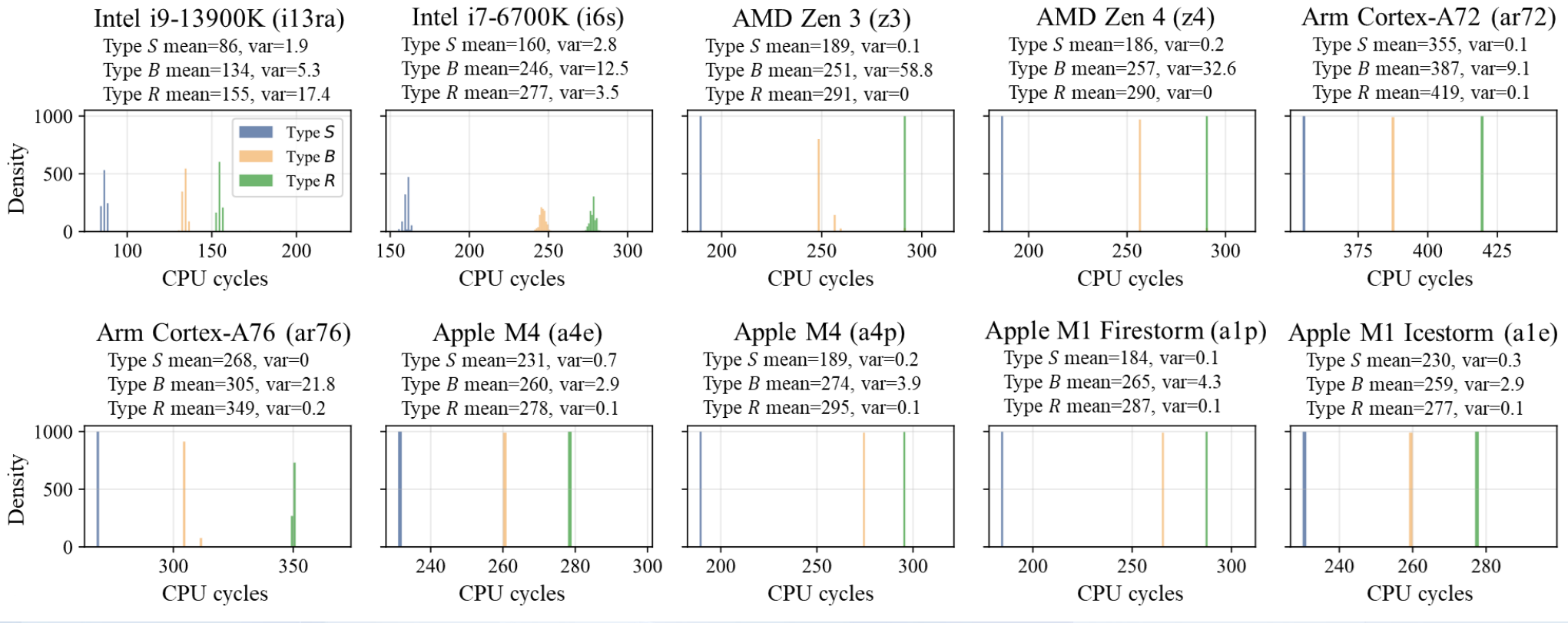
Method: Existence Analysis

- **Timing the Sequence**
  - N_P: Non-dependent store-load pairs
  - D_P: Dependent store-load pairs
  - Seq1: $100N_P$ $100N_P$ (Collect the last 100 samples)
  - Seq2: $100N_P$ $100D_P$ (Collect the last 100 samples)
- **Analysis Method**
  - Clustering 200 time samples with DBSCAN
  - If MDP exists:
    - ≥3 clusters should be identified
    - The clusters should be distinguishable
    - Execution time of D_P should be divided into ≥2 clusters



Result: Existence Analysis

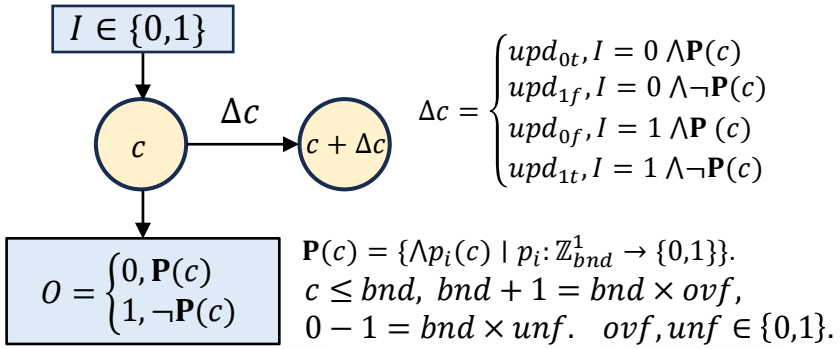
- **Experiments on ten CPUs**
  - Vendors: Intel, AMD, Arm, Apple
  - Conclusion: MDP exists on these CPUs



Method: State Machine Analysis

Counter Model

- State is defined by a hardware counter (c)
- The counter updates differently in four cases (upd_{0t}, upd_{1f}, upd_{0f}, upd_{1t})
- The counter has an upper bound (bnd)



$$\Delta c = \begin{cases} upd_{0t}, I = 0 \wedge \mathbf{P}(c) \\ upd_{1f}, I = 0 \wedge \neg\mathbf{P}(c) \\ upd_{0f}, I = 1 \wedge \mathbf{P}(c) \\ upd_{1t}, I = 1 \wedge \neg\mathbf{P}(c) \end{cases}$$

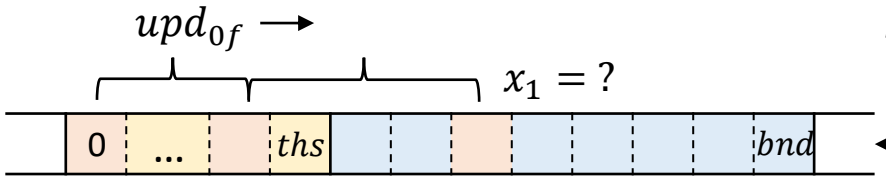
$$\mathbf{P}(c) = \{\wedge p_i(c) \mid p_i : \mathbb{Z}^1_{bnd} \to \{0,1\}\}.$$

$$c \leq bnd,\ bnd + 1 = bnd \times ovf,$$

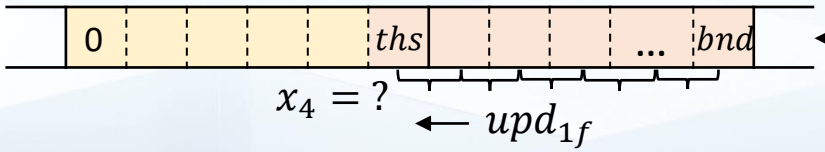
$$0 - 1 = bnd \times unf.\quad ovf, unf \in \{0,1\}.$$

Method: State Machine Analysis

- **Counter Model**
  - State is defined by a hardware counter (c)
  - The counter updates differently in four cases (upd_{0t}, upd_{1f}, upd_{0f}, upd_{1t})
  - The counter has an upper bound (bnd)
- **Solve counter parameters**
  - Define constraints by optimization problem (x_1, x_2, …, x_6)
  - Solve parameters by Linear Programming (EQ_1, EQ_2, …, EQ_7)



x_1 defines the relationship between upd_{0f} and ths



x_4 defines the relationship between bnd and ths

Algorithm 1 Parameter Solving for the Single-Counter Model

Input: Store-load pair a and n, and timing differences S, B and R
Output: upd_{0t}, upd_{1f}, upd_{0f}, upd_{1t}, bnd, ovf, unf, ths

$upd_{1f} \leftarrow -1$ *// Assumption: counter decreases on independent cases*
$unf \leftarrow 0$ *// Assumption: on underflow*
$x_1 \leftarrow \min_x T((x+1)D_P) = xR\ B$
$x_2 \leftarrow \min_x T(x_1 D_P\ (x+1)N_P) = x_1 R\ xB\ S$
$x_3 \leftarrow \min_x T(x_1 D_P\ (x+1)D_P) = x_1 R\ xB\ R$
$EQ_1 \leftarrow (x_1 - 1) \cdot upd_{0f} \leqslant ths < x_1 \cdot upd_{0f}$
$EQ_2 \leftarrow ths = x_1 \cdot upd_{0f} - x_2$
if $x_3 = \infty$ **then** *// no overflow, no decrease on dependent cases*
  $x_4 \leftarrow \max_x T((x_1+y)D_P\ (x+1)N_P) = x_1 R\ (y+x)B\ S,\ \forall\ y$
  $x'_4 \leftarrow \min_x T(x_1 D_P\ (x_2-1)N_P\ D_P\ (x+1)N_P)$
    $= x_1 R\ (x_2+x)B\ S$
  $EQ_3 \leftarrow upd_{1t} = x'_4 - 1$
  $EQ_4 \leftarrow bnd = x_4 + ths$
  $EQ_5 \leftarrow ovf = 1$
else *// overflow or counter decreases on dependent cases*
  $x_4 \leftarrow \min_x T((x_1+1)D_P\ (x+1)N_P) = x_1 R\ (x+1)B\ S$
  if $x_4 > x_2$ **then** *// overflow when the counter reaches bnd and reset*
    $x'_4 \leftarrow \min_x T(x_1 D_P\ (x_2-1)N_P\ (x+1)D_P)$
      $= x_1 R\ (x_2-1+x)B\ R$
    $x''_4 \leftarrow \min_x T(x_1 D_P\ (x_2-1)N_P\ (x'_4-1)D_P\ xN_P\ 2D_P)$
      $= x_1 R\ (x_2+x'_4+x)B$
    $EQ_3 \leftarrow upd_{1t} > 0$
    $EQ_4 \leftarrow bnd - ths = x'_4 \cdot upd_{1t} - x''_4$
    $EQ_5 \leftarrow ovf = 0$
  else *// counter decreases on dependent cases*
    $x'_4 \leftarrow \min_x T(x_1 D_P\ (x_2-1)N_P\ (x+2)D_P)$
      $= x_1 R\ (x_2-1)B\ S\ xB$
    $x''_4 \leftarrow \min_x T(x_1 D_P\ (x_2-1)N_P\ (x'_4+1)D_P\ (x+1)N_P)$
      $= x_1 R\ (x_2-1)B\ S\ x'_4 R\ xB\ S$
    $EQ_3 \leftarrow upd_{1f} = x''_4 - 1 - x'_4 \cdot upd_{0f}$
    $EQ_4 \leftarrow bnd = x_1 \cdot upd_{0f}$
    $EQ_5 \leftarrow ovf = 1$
  end if
end if
$x_5 \leftarrow \min_x T((x_1-1)D_P\ N_P\ (x+1)D_P) = (x_1-1)R\ S\ xR\ B$
$x_6 \leftarrow \min_x T((x_1-1)D_P\ N_P\ x_5 D_P\ (x+1)N_P)$
  $= (x_1-1)R\ S\ x_5 R\ xB\ S$
$EQ_6 \leftarrow x_1 \cdot upd_{0f} + upd_{0t} = x_6 + ths$
$EQ_7 \leftarrow upd_{0t} \leqslant ths$
Solve the system of inequalities $\{EQ_1, ..., EQ_7\}$ and get parameters

Method: Hash Function Analysis

- **Store-load Bounce**
  - Replace store/load with two branches
  - Each branch targets to a store/load
- **Workflow**
  - **Step 1**: Generate differential matrix $\boldsymbol{R}$, where each row is the XOR of two collided addresses
  - **Step 2**: Solve the null space $\mathcal{N}(\boldsymbol{R})$ of $\boldsymbol{R}$, where $\mathcal{N}(\boldsymbol{R}) = \{x \mid \boldsymbol{R}x = \boldsymbol{0}\}$
  - **Step 3**: Given a basis of $\mathcal{N}(\boldsymbol{R})$, denoted as n_1, n_2, ..., n_r, the hash function $\boldsymbol{M} = \begin{bmatrix} n_1^T \\ n_2^T \\ \dots \\ n_r^T \end{bmatrix}$

Code for MDP Probing (Pair)

```
1  stld:
2  .rep 10
3      imul $1, %rdi
4  .endr
5  jmp *%rdx // jump to store
6  ld_start:
7  jmp *%rcx // jump to load
8  ld_end:
9  .rep 10
10     imul $1, %rax
11 .endr
```

Store-load Bounce

Store Bounce

```
movq $0, (%rdi) // store
jmp ld_end  // jump back
```

Instruction Pages

```
movq 0(%rsi), %rax // load
jmp ld_end  // jump back
```

Load Bounce

Step 0 (with Store-load Bounce)

Collect collided addresses that select the same MDP table entry

Step 1

Generate differential matrix $\boldsymbol{R}$

Step 2

Solve $\mathcal{N}(\boldsymbol{R})$

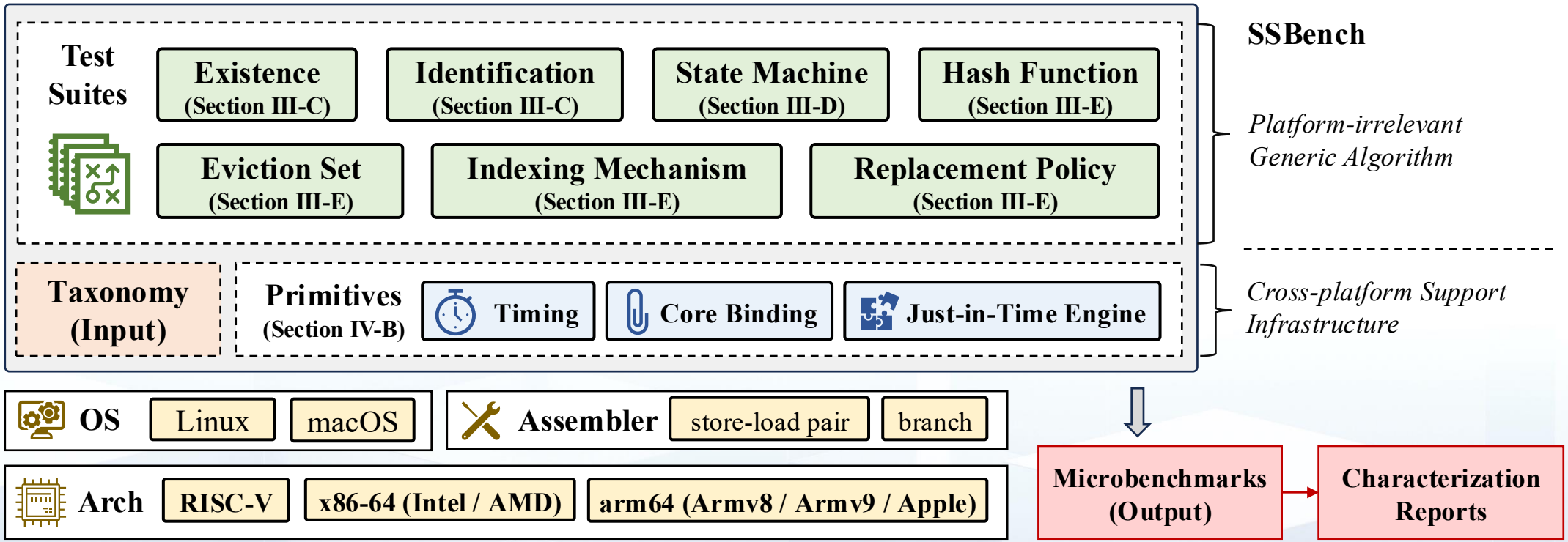
Step 3

Solve hash function $\boldsymbol{M}$

SSBench: Implementation

Loose Coupling Design

- Platform-irrelevant Generic Algorithm: Python
- Cross-platform Support Infrastructure: C & Assembly



SSBench: Evaluation

Evaluation across 30 CPUs

- Identify 3 different MDP designs
- Identify 14 different MDP configurations
- Execution time ≤ 2 hours

TABLE IV
EXECUTION TIME (IN SECONDS) OF SSBENCH ON SOME CPUS.

CPU ID	Test Time	State Machine	Hash Function*	Organization
i13ra	2531.59 s	59.92 (2.4%)	2417.47 (95.5%)	54.21 (2.1%)
z3	4047.87 s	77.99 (1.9%)	2467.47 (61.0%)	1502.47 (37.1%)
ar76	1474.41 s	92.29 (6.3%)	920.92 (62.5%)	461.20 (31.2%)
a4e	3500.45 s	125.66 (3.6%)	1895.27 (54.1%)	1479.52 (42.3%)
a4p	2102.49 s	41.541 (2.0%)	359.27 (17.1%)	1701.68 (80.9%)

* We separate the time of hash function test from organization test.

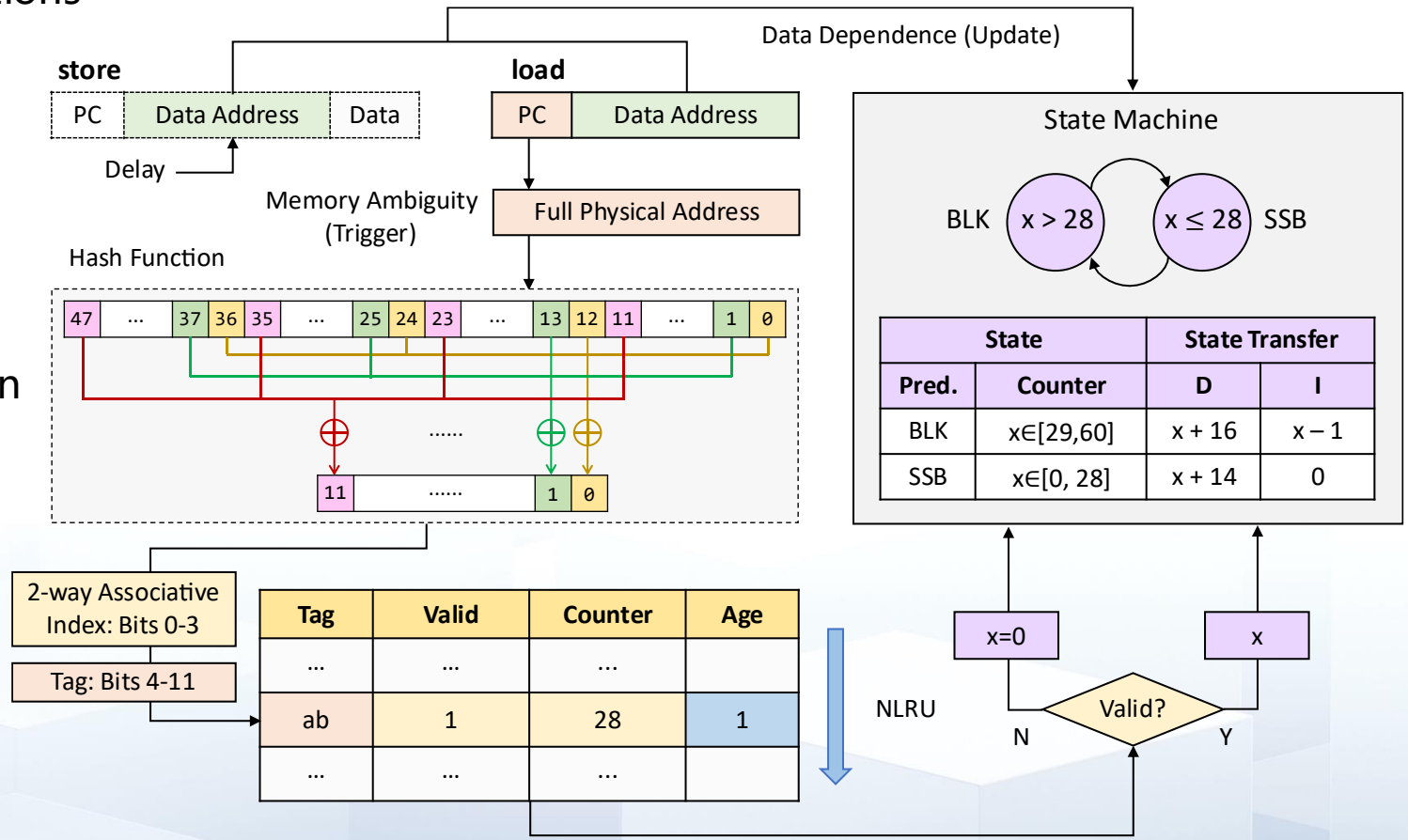
TABLE II
EXPERIMENTAL RESULTS OF SSBENCH ON 30 CPUS. V-IP MEANS SELECTION BY VIRTUAL IP. REP MEANS REPLACEMENT POLICY.

x86-64*	Design	State Machine	V-IP	Hash Function	Size	Way	Tag	Index	Rep
i6s, i7k, i8c, i9cf, i10cm, i11ro	L-S	0, -1, 15, 14, 15, 1, 0, 0	✔	[0], [1], [2], [3], [4], [5], [6], [7]	256	1	–	0-7	–
i12a, i13ra, i14ra, ixg6s	L-S	0, -1, 15, 14, 15, 1, 0, 0	✔	[0], [1], [2], [3], [4], [5], [6], [7], [8]	512	1	–	0-8	–
z3-mdp1	SL-S	0, -1, 4, 3, 4, -1, 0, 0	✗	[0,12,24,36], [1,13,25,37], [8,20,32,44], [2,14,26,38], [3,15,27,39], [9,21,33,45], [4,16,28,40], [5,17,29,41], [10,22,34,46], [6, 18,30,42], [7,19,31,43], [11,23,35,47]			–**		
z3-mdp2	L-S	0, -1, 15, 16, 62, 1, 0, 30			32	2	4-11	0-3	LRU
z4	L-S	0, -1, 14, 0, 42, 1, 0, 28			32	2	4-11	0-3	FIFO
z5	L-S	0, -1, 14, 16, 60, 1, 0, 28			64	4	4-11	0-3	NLRU
Arm64 / RISC-V	**Design**	**State Machine**	**V-IP**	**Hash Function**	**Size**	**Way**	**Tag**	**Index**	**Rep**
ar72	L-nS	–	✔	[2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]	16	16	6-15 ***	–	PLRU
ar73	L-nS	–	✔	[2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]	16	16	2-16	–	FIFO
ar76	L-S	0, -1, 14, 0, 14, 1, 0, 0	✔	[2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]	32	32	2-14	–	FIFO
a1p	SL-S	0, -1, 3, 1, 7, 1, 0, 0	✔	[2], [3], [4], [5], [6], [7, 14], [8, 15], [9, 16], [10, 17], [11, 18], [12, 19], [13]	90	90	0-11	–	LRU
a1e, a2e, a3e	SL-S	0, -1, 3, 1, 7, 1, 0, 0			21	21	0-11	–	LRU
a2p, a3p	SL-S	0, -1, 3, 1, 7, 1, 0, 0			65	65	0-11	–	LRU
a4p	SL-S	0, -1, 7, 1, 7, 1, 0, 0			69	69	0-11	–	LRU
a4e	SL-S	0, -1, 3, 1, 7, 1, 0, 0			21	21	0-11	–	FIFO
ar53, ar55, ar510, rvoc910, rvx60, rvboom			MDP is not employed						

* i6s: Intel i7-6700K (Skylake). i7k: Intel i7-7700K (Kaby Lake). i8c: Intel i5-8400 (Coffee Lake). i9cf: Intel i7-9700 (Coffee Lake). i10cm: Intel i7-10700 (Comet Lake). i11ro: Intel i7-11700 (Rocket Lake). i12a: Intel i9-12900K (Alder Lake). i13ra: Intel i9-13900K (Raptor Lake). i14ra: Intel i9-14900K (Raptor Lake). ixg6s: Intel Xeon Gold 6438Y+ (Sapphire Rapids). z3: AMD EPYC 7543 (zen 3). z4: AMD Ryzen 7950X (zen 4). z5: AMD Ryzen 9950X (zen 5). ar53: Arm Cortex-A53. ar55: Arm Cortex-A55. ar510: Arm Cortex-A510. ar72: Arm Cortex-A72. ar73: Arm Cortex-A73. ar76: Arm Cortex-A76. a1e: Apple M1 efficiency core. a1p: Apple M1 performance core. a2p: Apple M2 performance core. a2e: Apple M2 efficiency core. a3p: Apple M3 performance core. a3e: Apple M3 efficiency core. a4p: Apple M4 performance core. a4e: Apple M4 efficiency core. rvoc910: OpenC910 Core. rvx60: SpacemiT X60. rvboom: SonicBOOM [81].
** z3-mdp1 overlaps with AMD's PSFP and cannot be characterized directly through SSBench.
*** Results of bits 2-5 on ar72 are used for offset.

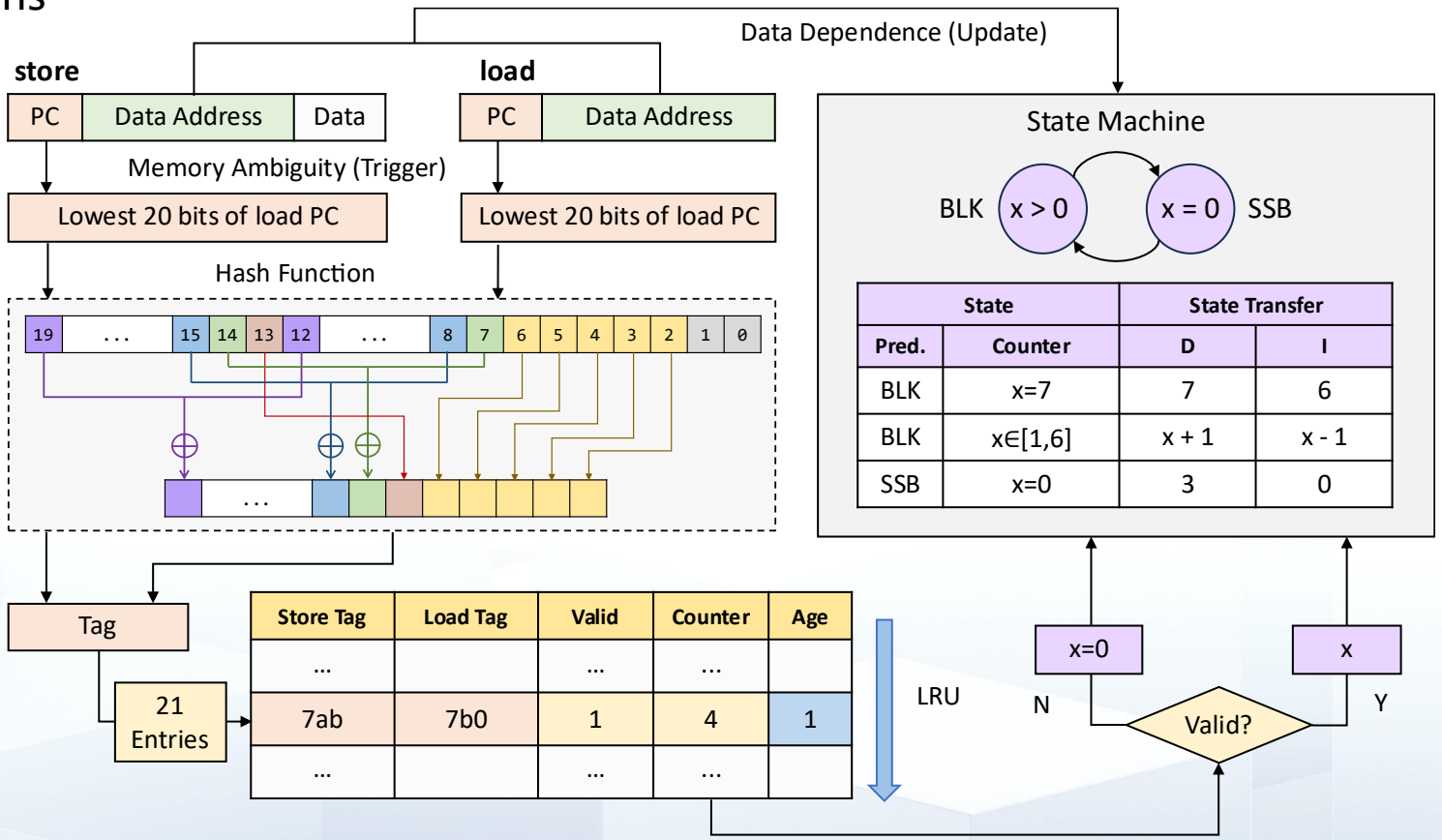
SSBench: Evaluation

- **Evaluation across 30 CPUs**
  - Identify 3 different MDP designs
  - Identify 14 different MDP configurations
  - Execution time: ≤ 2 hours
- **Example**
  - **AMD Zen 5 MDP**
    - State machine: 6-bit counter
    - 2-way associative table
    - Physical PC-based hash function
    - Triggerred by a single load



SSBench: Evaluation

- **Evaluation across 30 CPUs**
  - Identify 3 different MDP designs
  - Identify 14 different MDP configurations
  - Execution time: ≤ 2 hours
- **Example**
  - **AMD Zen 5 MDP**
    - State machine: 6-bit counter
    - 2-way associative table
    - Physical PC-based hash function
    - Trigger by a single load
  - **Apple M2 efficiency core**
    - State machine: 3-bit counter
    - Fully-associative table
    - Indexed by both store and load
    - 21 entries
    - Virtual PC-based hash function



Case Study: MDP-Gate

- **Microarchitectural Weird Machine (µWM)**
  - Programming at the microarchitecture level
  - Register: µarch buffers
  - Memory: Architectural registers/memory
  - Calculation: Speculative execution

Cache-based NOT Gate

```
1  if(in[0] == 0)
2    out[delay()] = 0;
```

Cache-based NAND Gate

```
1  if(in1[in2[0]] == 0)
2    out[delay()] = 0;
```

Case Study: MDP-Gate

- **Microarchitectural Weird Machine (µWM)**
  - Programming at the microarchitecture level
  - Register: µarch buffers
  - Memory: Architectural registers/memory
  - Calculation: Speculative execution

Insights from SSBench

- State of Intel MDP can be propagated
- Intel MDP has up to 512 directly mapped entries
- Intel MDP uses 9 LSBs of load PC for entry selection

Cache-based NOT Gate

```
1  if(in[0] == 0)
2    out[delay()] = 0;
```

Cache-based NAND Gate

```
1  if(in1[in2[0]] == 0)
2    out[delay()] = 0;
```

Advantages of MDP-Gates

- Branch misprediction is not required
- Cache misses are not required
- Entry eviction does not occur

Case Study: MDP-Gate

- **Microarchitectural Weird Machine (µWM)**
  - Programming at the microarchitecture level
  - Register: µarch buffers
  - Memory: Architectural registers/memory
  - Calculation: Speculative execution

Insights from SSBench

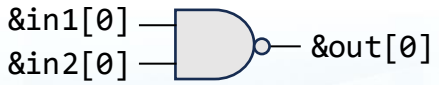
- State of Intel MDP can be propagated
- Intel MDP has up to 512 directly mapped entries
- Intel MDP uses 9 LSBs of load PC for entry selection

Cache-based NOT Gate

```
1  if(in[0] == 0)
2    out[delay()] = 0;
```

Cache-based NAND Gate

```
1  if(in1[in2[0]] == 0)
2    out[delay()] = 0;
```



NOT MDP-Gate

```
1  movq 0(%rdi), %rdi
2  movq %rdi, 0(%rdi)
3  movq (%rsi), %rax
4  .rep 30
5  imul $1, %rsi
6  .endr
7  movq 0(%rsi, %rax), %rax
```

NAND MDP-Gate

```
1  movq 0(%rdi), %rdi
2  movq %rdi, 0(%rdi)
3  movq (%rsi), %rax
4  movq (%rsi), %rdx
5  movq (%rsi, %rax), %rax
6  jmp nand_o2 // 512 bytes
7  movq (%rsi, %rdx), %rax
```

Case Study: MDP-Gate

- **Microarchitectural Weird Machine (µWM)**
  - Programming at the microarchitecture level
  - Register: µarch buffers
  - Memory: Architectural registers/memory
  - Calculation: Speculative execution
- **Evaluation of MDP-Gate**
  - Gate-level Performance: 3-15× faster
  - Circuit-level Performance: 100× faster than cache gates

Insights from SSBench

- State of Intel MDP can be propagated
- Intel MDP has up to 512 directly mapped entries
- Intel MDP uses 9 LSBs of load PC for entry selection

TABLE V
EVALUATION OF MDP-GATES

MDP-Gates	NOT	Assign	OR	NOR	NAND	XOR
Accuracy	99.96%	99.89%	99.19%	99.83%	99.45%	99.60%
Time / 10^7 gates	2.10 s	2.05 s	3.22 s	3.08 s	3.20 s	3.14 s
Cache-Gates	NOT	Assign	OR	NOR	NAND	XOR
Accuracy	82.77%	69.58%	82.91%	74.23%	82.83%	75.12%
Time / 10^7 gates	9.82 s	8.98 s	8.82 s	25.74 s	22.25 s	47.05 s

TABLE VI
COMPARISON ON CIRCUITS OF MDP-GATES AND DATA CACHE-GATES

Weird Circuit		MDP-Gates		Data Cache-Gates	
	Strategy	**Naive**	**Best-of-5**	**Naive**	**Best-of-5**
Adder	**Accuracy**	92.63%	97.12%	67.15%	75.37%
	Time	0.238 ms	1.191 ms	41.9 ms	47.1 ms
	Strategy	**Naive**	**Best-of-5**	**Gates of Time [26]**	
ALU	**Accuracy**	87.53%	99.38%	43.7% to 84.1%	
	Time	0.882 ms	4.423 ms	106 ms	

Case Study: MDP-CF

Control-flow Side-channel Attacks

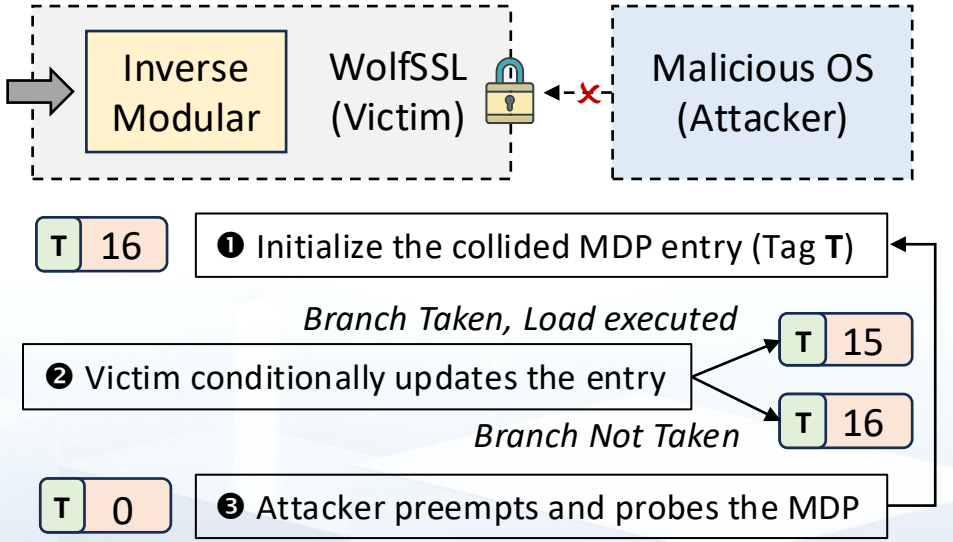
- Leak load PCs
- Infer which load is executed
- Break secret-dependent branch

Insights from SSBench

- AMD MDP is not isolated between processes
- A single load can update AMD MDP
- Different loads within one page will select different entries

Attack `_sp_invmod_bin()` in wolfSSL v5.8.4 (latest version at paper submission)

```
1  while (!sp_isone(v) && !sp_iszero(u)) {
2    if ((u->dp[0] & 1) == 0) { ...
3    } // if u % 2 == 0, u /= 2
4    else if ((v->dp[0] & 1) == 0) { ...
5    } // else if u % 2 == 0, v /= 2
6    else if (_sp_cmp_abs(u, v) != MP_LT) { ...
7    } // else if u >= v, u -= v
8    else { ...
9    } // else, u < v, v -= u
10 } // Loop until u = 0 and v = 1
```

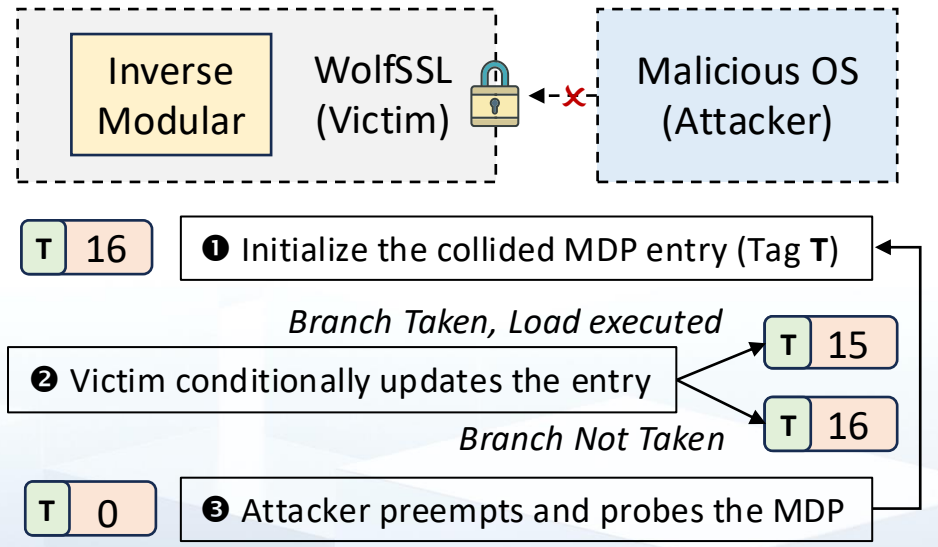


Case Study: MDP-CF

- **Control-flow Side-channel Attacks**
  - Leak load PCs
  - Infer which load is executed
  - Break secret-dependent branches
- **Evaluation of MDP-CF**
  - 98% accuracy in a single trace
  - 0.275 seconds to perform a single-trace attack
  - 75% success rate to recover 4096-bit inputs

Insights from SSBench

- AMD MDP is not isolated between processes
- A single load can update AMD MDP
- Different loads within one page will select different entries



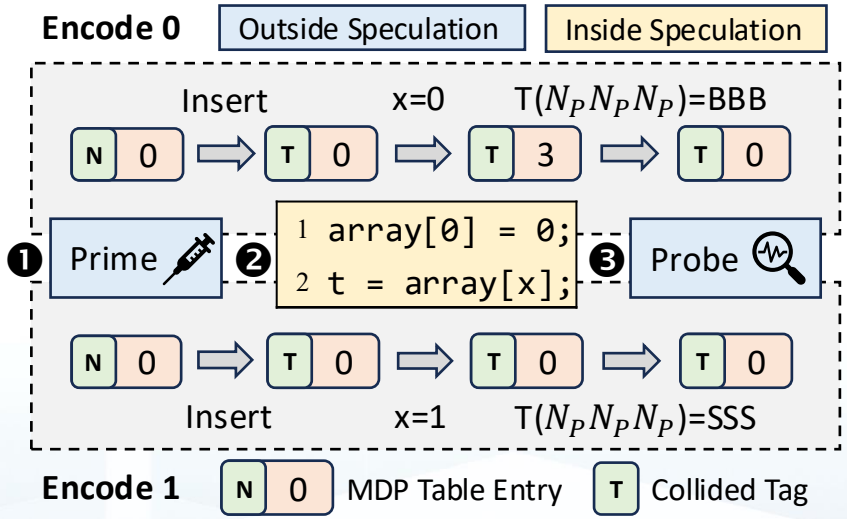
Case Study: MDP-CC

Covert Channel in Spectre Attacks

- Replace cache- and TLB-based covert channels
- Does not require eviction set construction
- Does not trigger cache/TLB misses

Insights from SSBench

- Apple MDP can be updated by speculative loads
- Apple MDP is selected by both store and load PC
- Apple MDP uses a virtual address-based hash function



Case Study: MDP-CC

Covert Channel in Spectre Attacks

- Replace cache- and TLB-based covert channels
- Does not require eviction set construction
- Does not trigger cache/TLB misses

Evaluation of MDP-CC

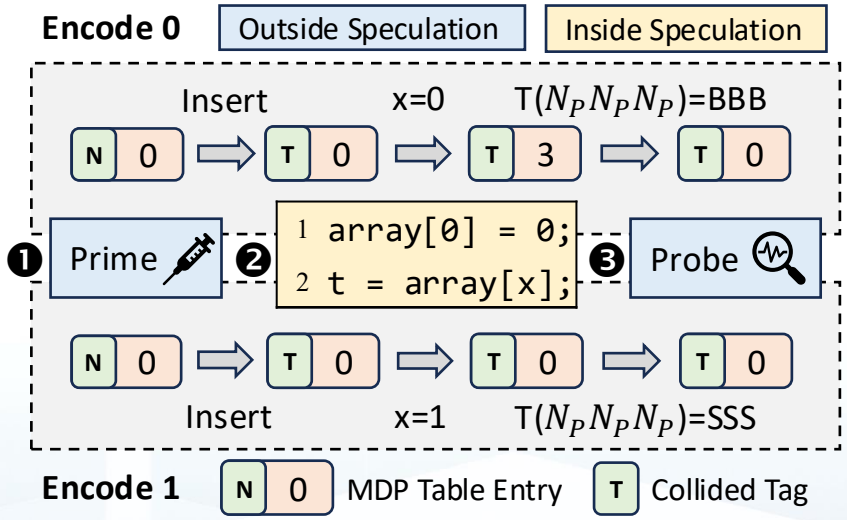
- Comparable performance
- Better stealthiness

TABLE VII
COMPARISON OF MDP-CC, CACHE, AND TLB COVERT CHANNELS

Source	Evaluation	M1	M2	M3	M4
MDP	True Capacity / bps	41 129.04	59 887.23	109 428.51	152 144.41
	Bit Error Rate	0.01	0.01	0.00	0.06
	Cache Miss / # of Inst	0.01	0.01	0.01	0.01
	TLB Miss / Cache Miss	0.01	0.00	0.00	0.00
Cache	True Capacity / bps	36 934.17	33 112.37	58 766.14	139 319.73
	Bit Error Rate	0.15	0.23	0.33	0.14
	Cache Miss / # of Inst	0.25	0.26	0.24	0.27
	TLB Miss / Cache Miss	0.01	0.00	0.28	0.00
TLB	True Capacity / bps	1085.09	1145.48	0.49	76.13
	Bit Error Rate	0.00	0.00	0.48	0.45
	Cache Miss / # of Inst	0.01	0.00	0.00	0.01
	TLB Miss / Cache Miss	0.66	0.69	0.99	0.69

Insights from SSBench

- Apple MDP can be updated by speculative loads
- Apple MDP is selected by both store and load PC
- Apple MDP uses a virtual address-based hash function



Conclusion

Summary

- **SSBench**: Automated reverse engineering of MDPs across 30 CPUs, identifying 14 different MDP configurations
- **MDP-Gate**: 100× faster than cache-based implementation
- **MDP-CF**: Instruction-level control-flow side-channel attack
- **MDP-CC**: Covert channel for transient execution attacks with better performance and stealthiness

Source Code

- https://github.com/CPU-Security/SSBench

① **Automated reverse engineering** of MDP across 30 CPUs

② **MDP-Gate**: Fastest µWM gates and circuits on Intel CPUs

③ **MDP-CF**: Breaking secret-dependent branches on AMD CPUs

④ **MDP-CC**: Faster and stealthier covert channel on Apple CPUs

Tsinghua University

National University of Singapore

Thanks

SSBench: Automated Characterization of Memory Dependence Predictors on Modern CPUs

Chang Liu, Yu Jin, Yuchen Fan, Tianrui Xiao, Lingfeng Yin, Trevor E. Carlson, Shuwen Deng, Dongsheng Wang

Questions?