THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY:
CEDA IEEE Council on Electronic Design Automation
SIGDA ACM

Thrend

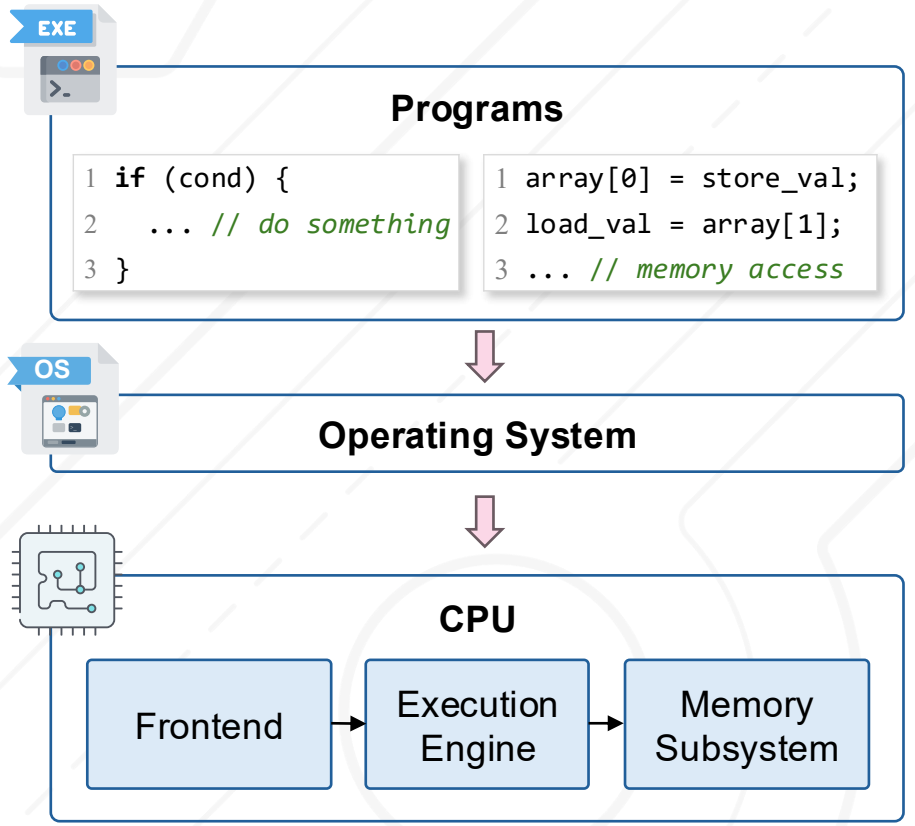
Mitigating Counting Thread-based Fine-grained Timing on Arm and Apple CPUs

Chang Liu*, Shuaihu Feng*, Jiaqi Cui, Hongpei Zheng, Aodong Cui,
Jian Dong, Yuan Li, Trevor E. Carlson, Dongsheng Wang

National University of Singapore

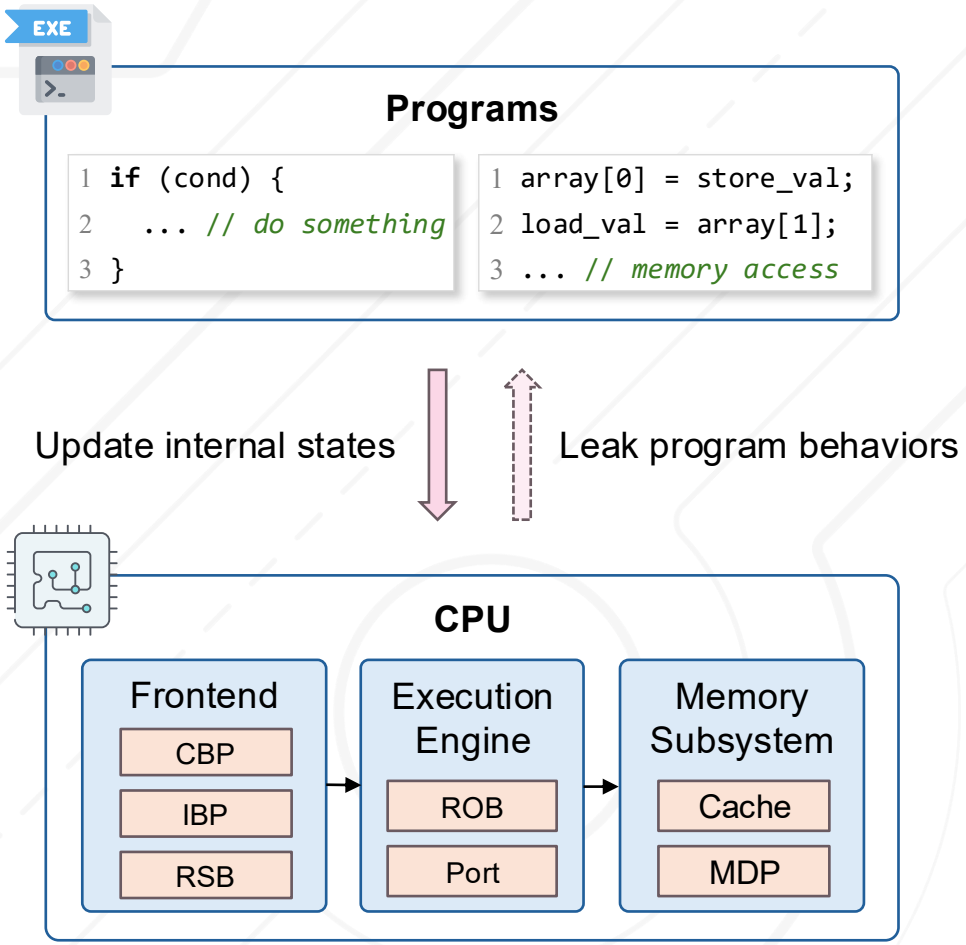
1. **Introduction**
2. Design & Implementation
3. Evaluation
4. Conclusion

CPU Side Channels



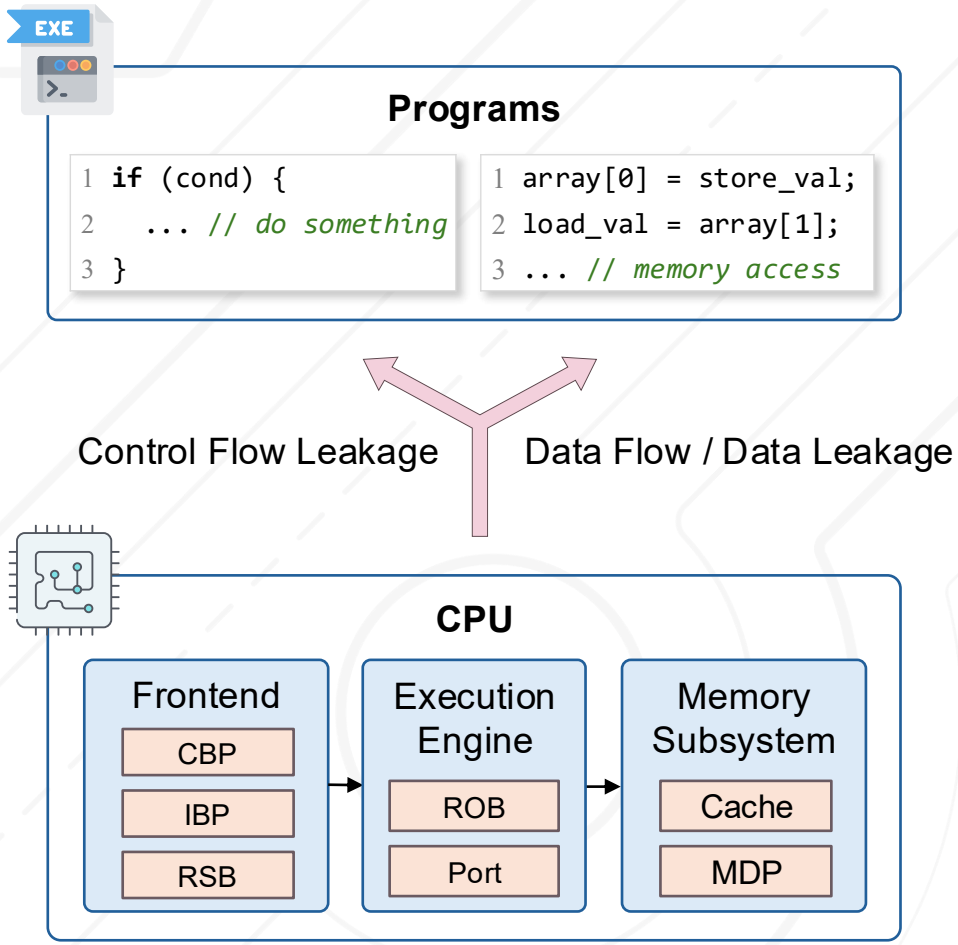
CPU Side Channels

- CPU side channels
  - Leak program behavior from CPU microarchitecture
  - Leakage sources: Cache, TLB, branch predictor, …



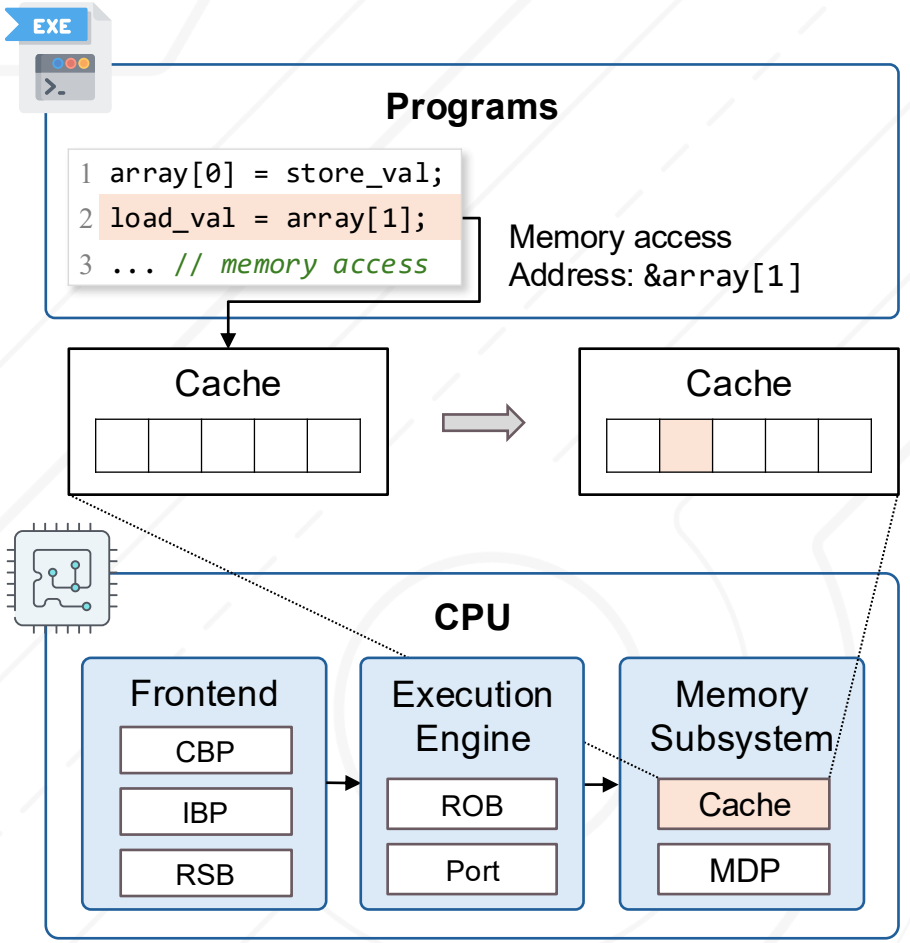
CPU Side Channels

- CPU side channels
  - Leak program behavior from CPU microarchitecture
  - Leakage sources: Cache, TLB, branch predictor, …
- What can be leaked?
  - Control Flow (e.g., instruction PC)
  - Data Flow (e.g., data address)
  - Data



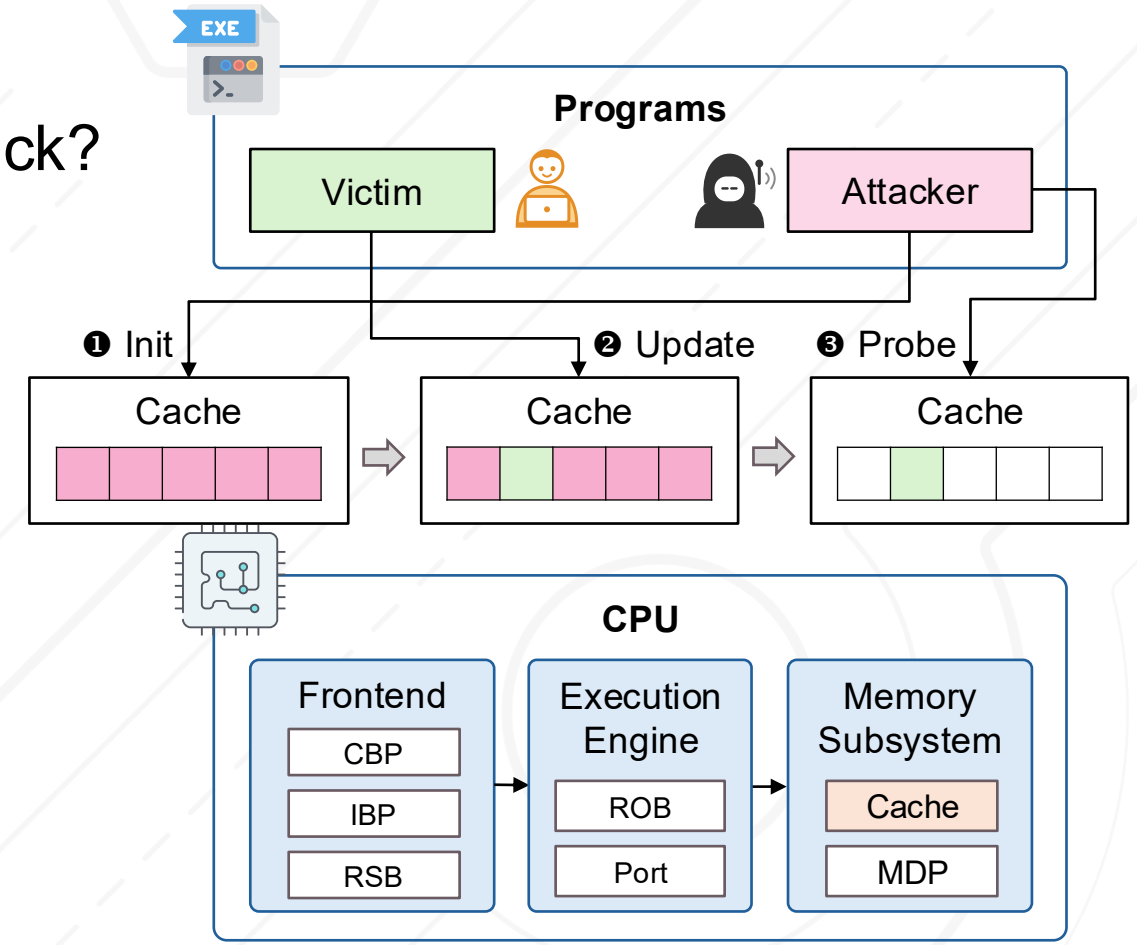
CPU Side Channels

- CPU side channels
  - Leak program behavior from CPU microarchitecture
  - Leakage sources: Cache, TLB, branch predictor, …
- What can be leaked?
  - Control Flow (e.g., instruction PC)
  - Data Flow (e.g., data address)
  - Data



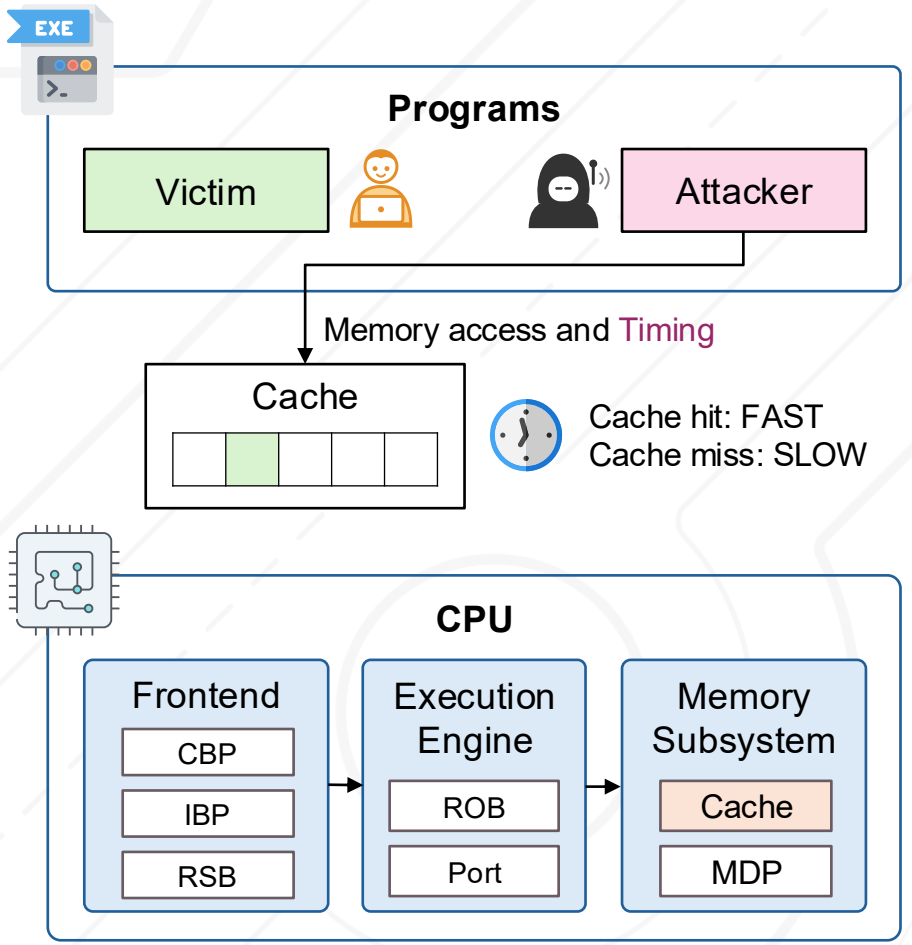
CPU Side Channels

- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state



CPU Side Channels

- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state
- How to probe microarchitecture information?
  - **Timing** the microbenchmark
  - Requirement: Cycle-level, nonprivileged



CPU Side Channels

- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state
- How to probe microarchitecture information?
  - **Timing** the microbenchmark
  - Requirement: Cycle-level, nonprivileged
  - Intel / AMD: `rdtsc` / `rdtscp` / `rdpru` (AMD only)

Architecture	Timer	Cycle-level	Non-privilege
x86	`rdtsc`	✓	✓
x86	`rdtscp`	✓	✓
x86 (AMD)	`rdpru`	✓	✓

CPU Side Channels

- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state
- How to probe microarchitecture information?
  - **Timing** the microbenchmark
  - Requirement: Cycle-level, nonprivileged
  - Intel / AMD: `rdtsc` / `rdtscp` / `rdpru` (AMD only)
  - Arm / Apple: `pmccntr` (root)

Architecture	Timer	Cycle-level	Non-privilege
x86	`rdtsc`	✓	✓
x86	`rdtscp`	✓	✓
x86 (AMD)	`rdpru`	✓	✓
Arm	`pmccntr`	✓	⊗

CPU Side Channels

- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state
- How to probe microarchitecture information?
  - **Timing** the microbenchmark
  - Requirement: Cycle-level, nonprivileged
  - Intel / AMD: `rdtsc` / `rdtscp` / `rdpru` (AMD only)
  - Arm / Apple: ~~`pmccntr`~~ / `cntvct_el0` (low granularity)

Architecture	Timer	Cycle-level	Non-privilege
x86	`rdtsc`	✓	✓
x86	`rdtscp`	✓	✓
x86 (AMD)	`rdpru`	✓	✓
Arm	`pmccntr`	✓	⊗
Arm	`cntvct`	⊗	✓

CPU Side Channels

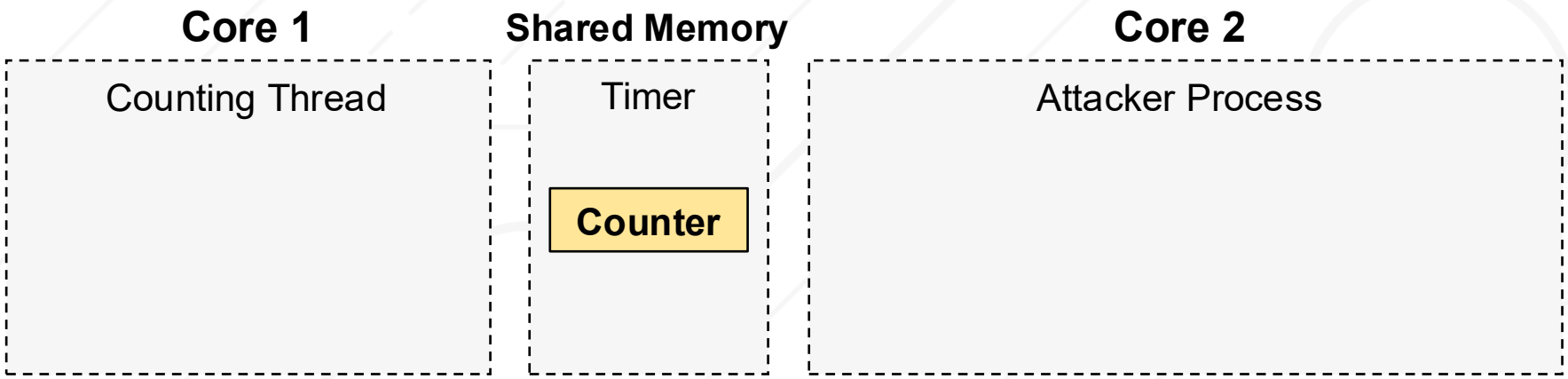
- How to perform a cache side-channel attack?
  - Step 1: Attacker initialize the cache state
  - Step 2: Victim update several cache lines
  - Step 3: Attacker probe the cache state
- How to probe microarchitecture information?
  - **Timing** the microbenchmark
  - Requirement: Cycle-level, nonprivileged
  - Intel / AMD: `rdtsc` / `rdtscp` / `rdpru` (AMD only)
  - Arm / Apple: ~~`pmccntr`~~ / ~~`cntvct_el0`~~

    counting thread

Architecture	Timer	Cycle-level	Non-privilege
x86	`rdtsc`	✓	✓
x86	`rdtscp`	✓	✓
x86 (AMD)	`rdpru`	✓	✓
Arm	`pmccntr`	✓	⊗
Arm	`cntvct`	⊗	✓
Arm	counting thread	✓	✓

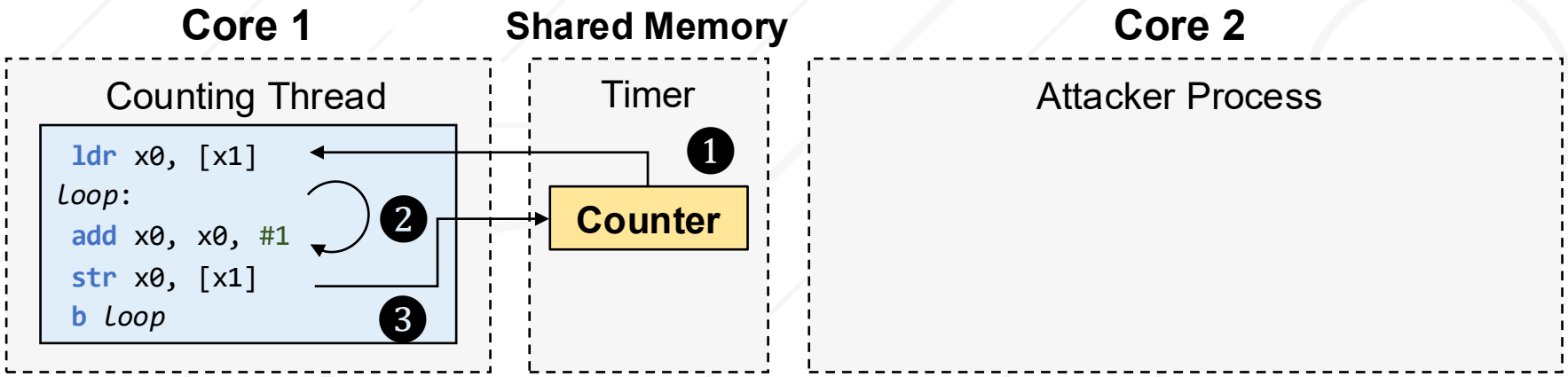
Counting Thread on Arm CPUs

- Counting thread
  - Software-based, nonprivileged, cycle-level
  - Counter: A variable in the shared memory



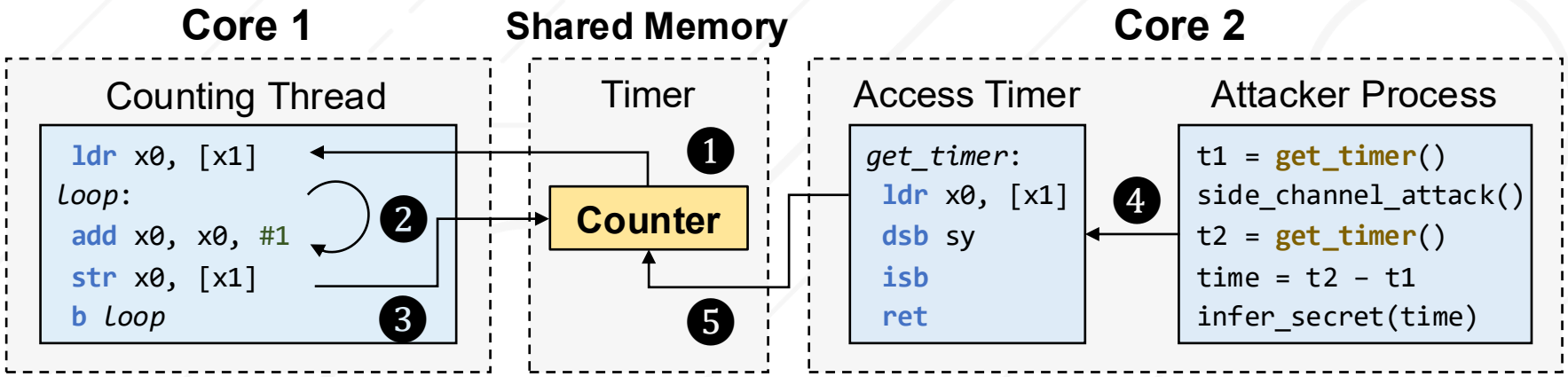
Counting Thread on Arm CPUs

- Counting thread
  - Software-based, nonprivileged, cycle-level
  - Counter: A variable in the shared memory
  - Counter update: Loop, increase and store



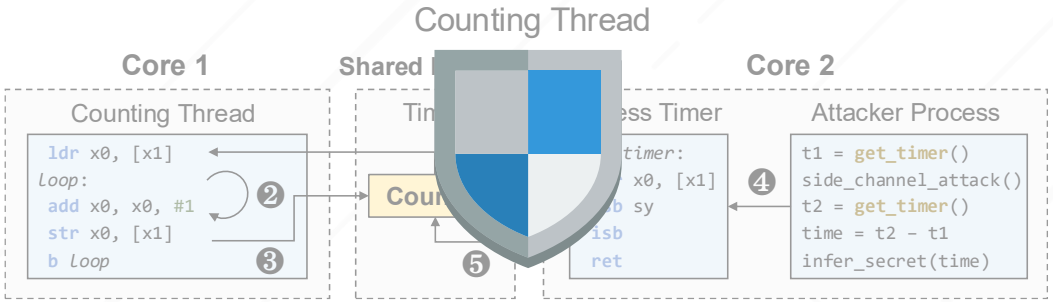
Counting Thread on Arm CPUs

- Counting thread
  - Software-based, nonprivileged, cycle-level
  - Counter: A variable in the shared memory
  - Counter update: Loop, increase and store
  - Counter access: Load
  - Timing simulation: Difference between two consecutive readings



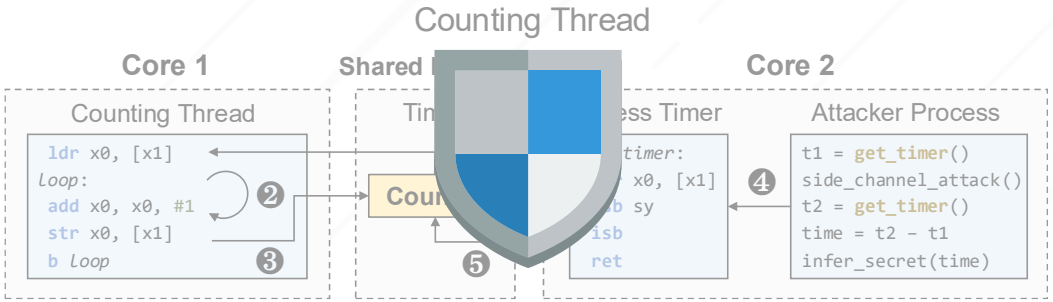
Motivation

- Motivation
  - The first defense against counting threads
  - Readily deployable
  - High accuracy and low performance overhead



Motivation

- Motivation
  - The first defense against counting threads
  - Readily deployable
  - High accuracy and low performance overhead
- Threat model
  - Traditional threat model (TT)
  - Advanced persistent threat model (APT)



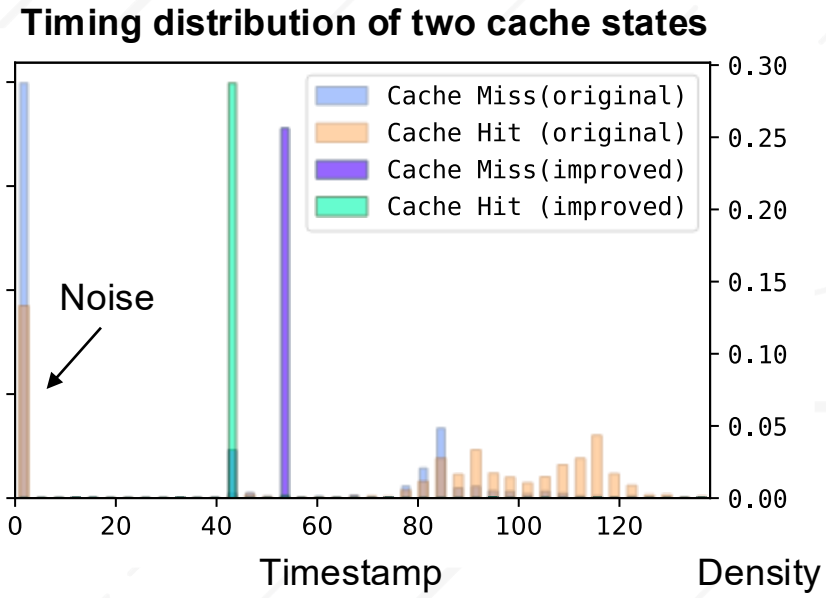
- **Traditional Thread Model**
  - Maximize accuracy and bandwidth
  - Keep counting thread stable and reliable
  - Do not consider the possibility to be detected or mitigated by any defenses

- **Advanced Persistent Threat Model**
  - Prioritize stealth over performance
  - Avoid frequent usage of counting threads
  - Hide the thread's behavior and bypass potential real-time detection

1. Introduction
2. **Design & Implementation**
3. Evaluation
4. Conclusion

Insights

- **Insight 1**
  - Counting threads can be further improved



Original Counting Thread

```
1 cnt_deadloop:
2   add x10, x10, #1
3   str x10, [x0]
4   b cnt_deadloop
```

```
1 gettime:
2   ldr x0, [x0]
3   ret
```

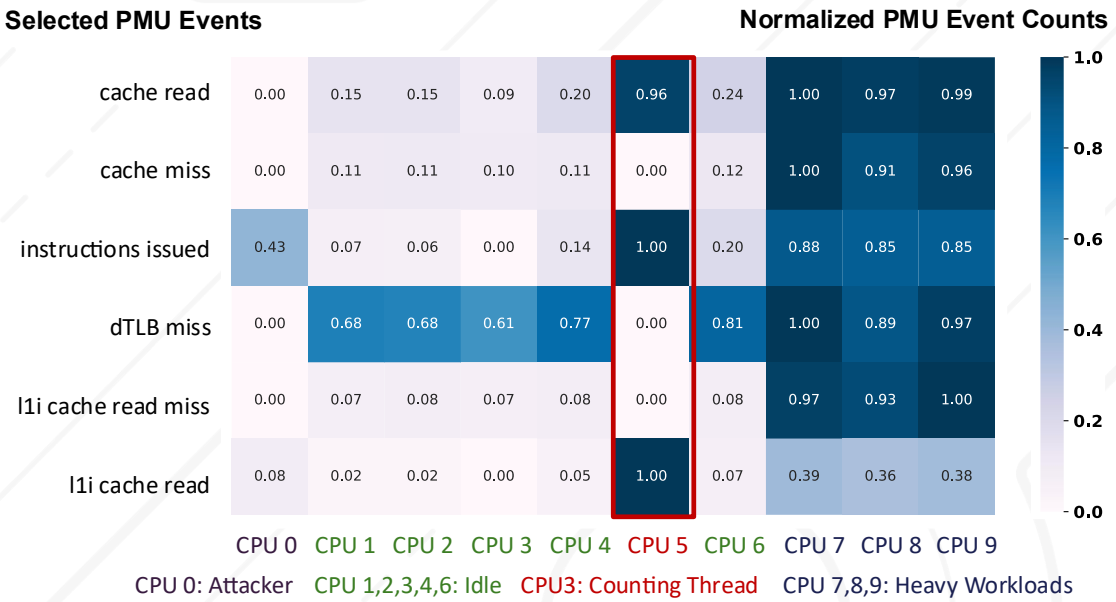
Improved Counting Thread

```
1 cnt_deadloop:
2   subs x0, x0, -1
3   subs x0, x0, -1
4   dsb ish
5   b cnt_deadloop
```

```
1 gettime:
2   dsb osh
3   isb
4   ldnp q0, q1, [x0]
5   mov x0, v0.d[0]
6   ret
```

Insights

- **Insight 1**
  - Counting threads can be further improved
- **Insight 2**
  - Counting threads exhibit significant microarchitectural signatures



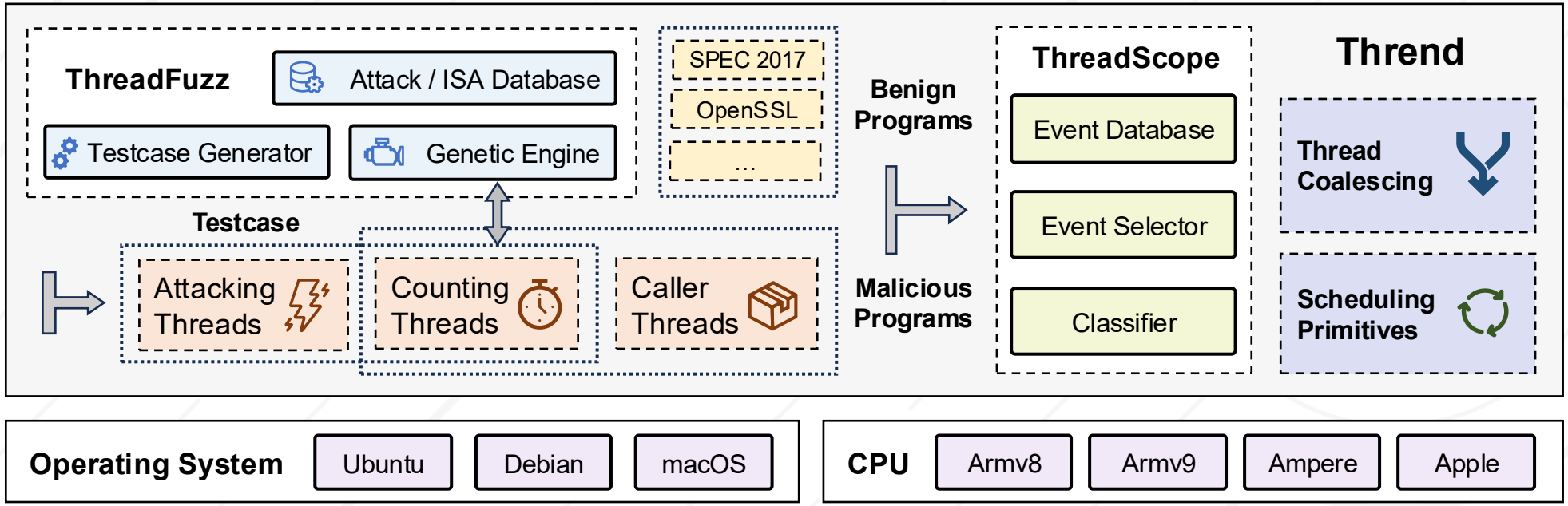
Insights

- **Insight 1**
  - Counting threads can be further improved
- **Insight 2**
  - Counting threads exhibit significant microarchitectural signatures
- **Insight 3**
  - Counting threads fundamentally rely on cross-core execution

Counting Thread Core	Attacker Process Core	Accuracy
0	1	✓
0	2	✓
0	3	✓
0	0	⊗
1	1	⊗
2	2	⊗

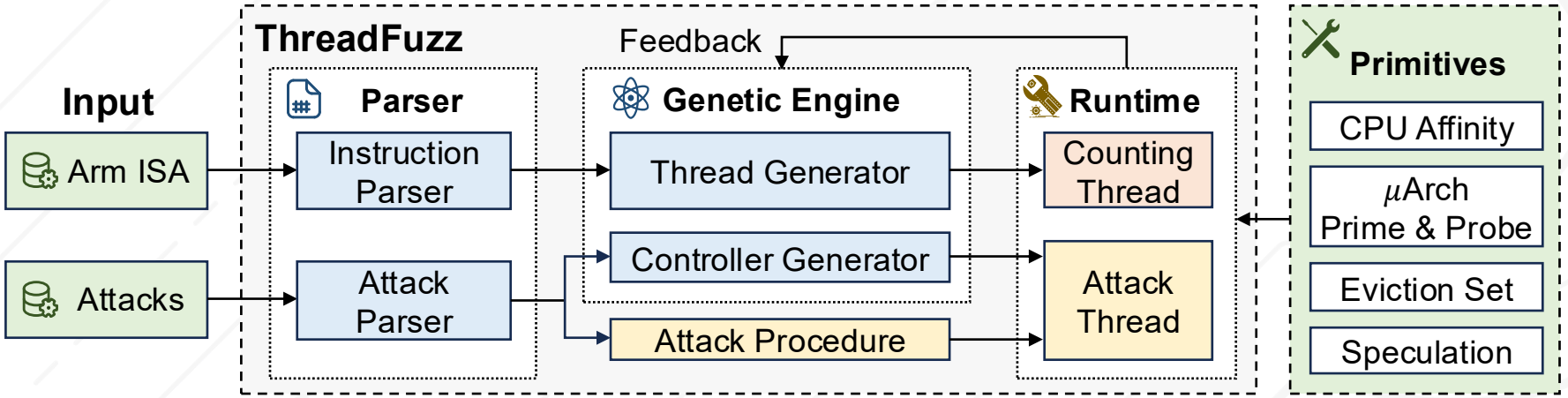
Design Overview

- Overview of Thrend
  - **ThreadFuzz**: Automated generation of malicious counting threads
  - **ThreadScope**: Real-time detection of counting threads using Performance Monitoring Unit (PMU)
  - **Thread coalescing**: Mitigate suspicious counting threads through process/thread scheduling



ThreadFuzz: Automated generation of counting threads

- ThreadFuzz
  - **Attack parser**: Provide templates for different side-channel attacks
  - **Instruction parser**: Provide instructions for generating counting threads
  - **Genetic engine**: Generate counting threads with varying quality
  - **Runtime configurations**: Run counting threads within attacks and collect feedback
  - **Fitness function**: Different metrics for TT and APT models

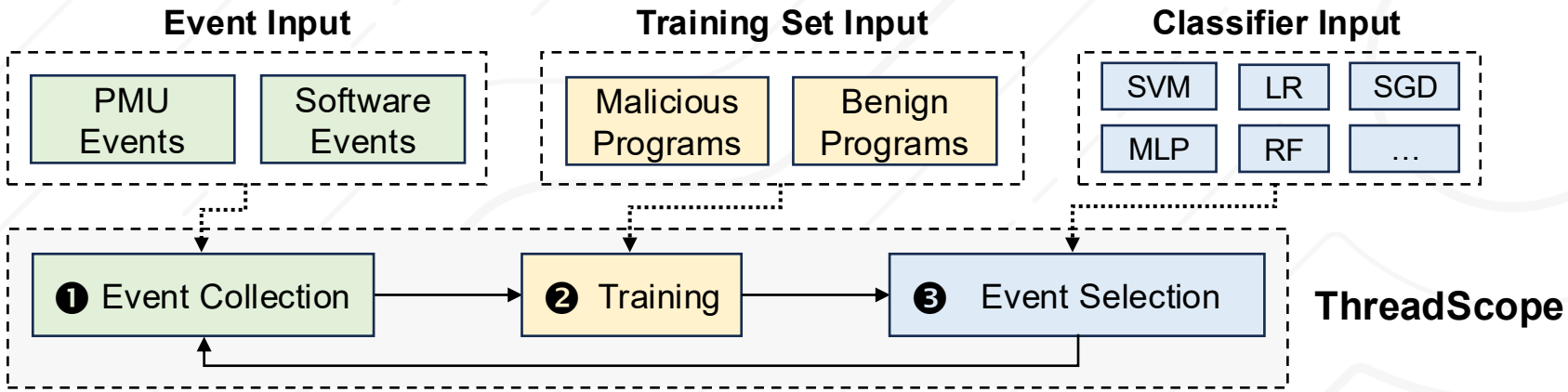


ThreadScope: Real-time detection of malicious counting threads

- ThreadScope
  - **Event collection**: perf (Linux), kperf (macOS), six events combined into one group
  - **Model training**:

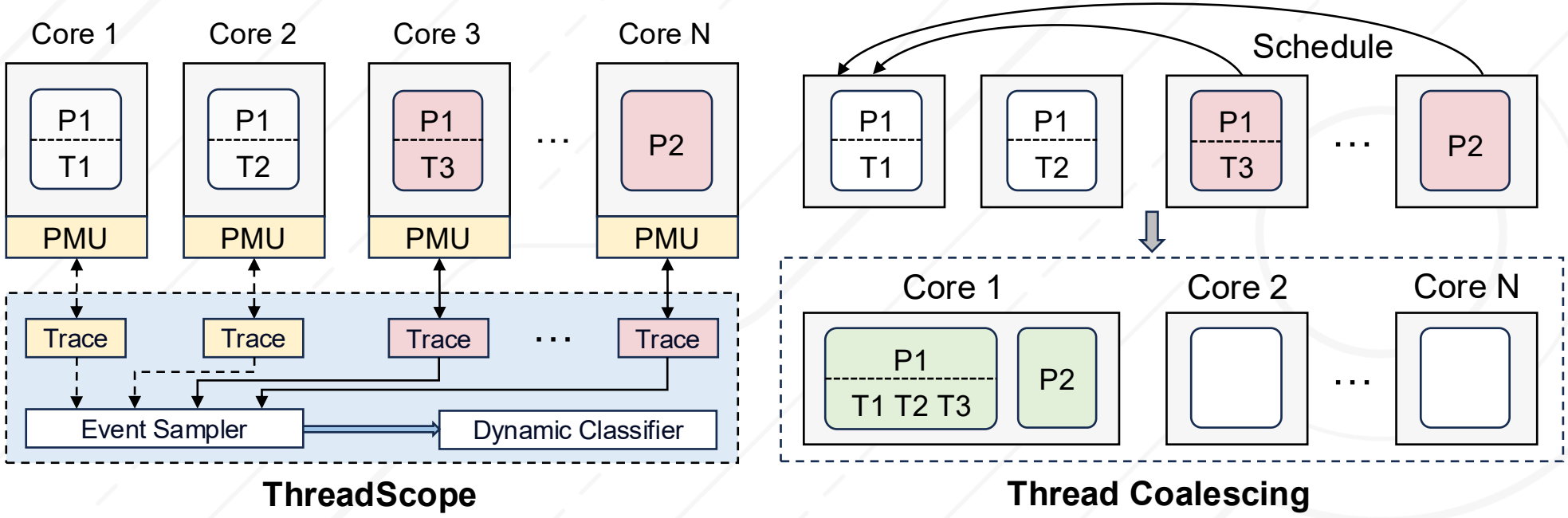
    samples = malicious programs + benign programs

    training set, validation set, test set = 50% samples, 30% samples, 20% samples
  - **Event selection:** Select event groups that yield the highest accuracy



Thread coalescing: Runtime mitigation of malicious counting threads

- Thread coalescing
  - **Process-level**: Binds all threads of the identified process to a fixed physical core
  - **User-level**: Migrates all processes belonging to the same user to a fixed physical core



1. Introduction
2. Design & Implementation
3. **Evaluation**
4. Conclusion

Environment Setup

- Environment
  - CPU: Include servers, personal computers, and embedded systems
  - OS: Linux and macOS
- Side-channel attack
  - Cache side-channel attacks (**`m4`**)
  - TLB side-channel attacks (**`m1`**)
  - MDP side-channel attacks (**`a72`** and **`n1`**)

ID	Vendor	Device	CPU	OS
`a72`	Broadcom	Raspberry Pi 4B	Cortex-A72	Debian 6.6.51
`n1`	GIGABYTE	R152-P30 Server	Ampere Altra Max	Ubuntu 20.04
`m1`	Apple	MacBook Pro	Apple M1	macOS 14.6.1
`m4`	Apple	Mac mini	Apple M4	macOS 15.5

Accuracy (Random Forest)

- **Conclusion 1**: The classification accuracy under the APT model is close to that under the TT model
- **Conclusion 2**: In the range of 10 Hz to 10,000 Hz, the sampling rate has little impact on accuracy

Frequency	Index	TT Model				APT Model			
		m1	m4	n1	a72	m1	m4	n1	a72
100 Hz	Accuracy	100%	100%	100%	100%	100%	100%	100%	99.710%
	False Positive	0	0	0	0	0	0	0	0.290%
	False Negative	0	0	0	0	0	0	0	0
1000 Hz	Accuracy	100%	100%	99.995%	100%	100%	100%	99.896%	99.985%
	False Positive	0	0	0.005%	0	0	0	0.016%	0
	False Negative	0	0	0	0	0	0	0.088%	0.015%
10000 Hz	Accuracy	100%	100%	100%	100%	100%	100%	99.984%	99.946%
	False Positive	0	0	0	0	0	0	0	0.046%
	False Negative	0	0	0	0	0	0	0.016%	0.008%

Performance Overhead

- Method
  - Enable ThreadScope on all cores and measure performance overhead with SPEC 2017
- Discussion
  - Higher sampling rates induce higher runtime overhead
  - Bottleneck: Dumping PMU events and transferring them to the model
  - Tradeoff: Higher frequency leads to faster response but higher overhead

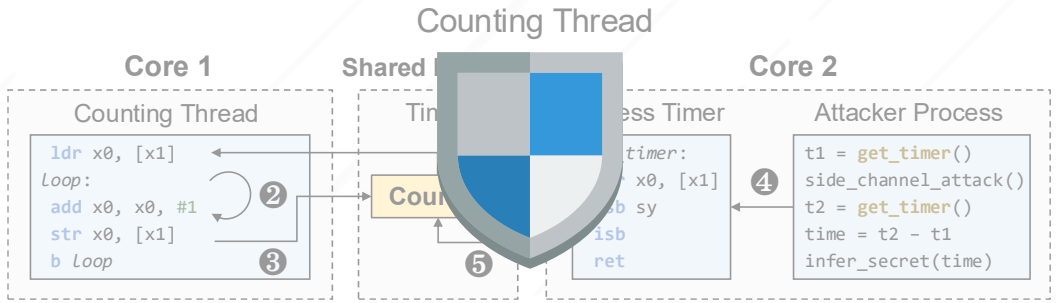
ID	100 Hz	500 Hz	1000 Hz	5000 Hz	10000 Hz
a72	1.33%	5.33%	8.00%	16.00%	18.00%
n1	0.98%	1.71%	2.45%	3.18%	2.45%*
m1	0.23%	1.42%	2.13%	5.46%	6.52%
m4	1.42%	6.43%	7.86%	10.71%	12.86%

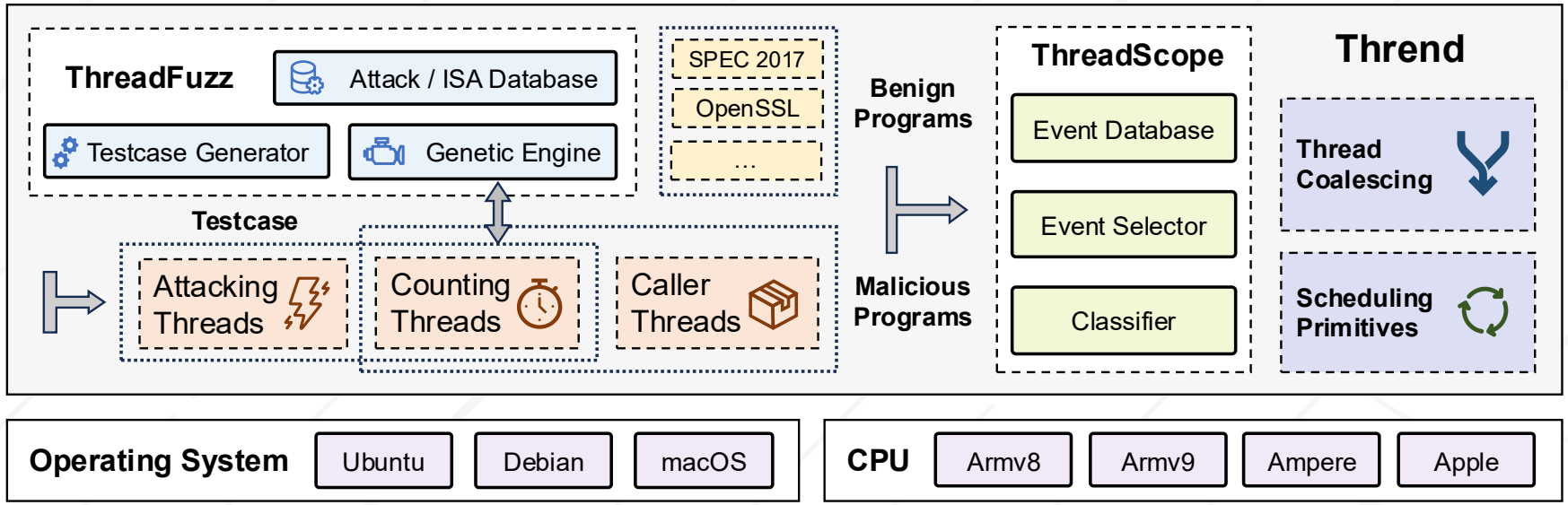
* Reason remains unknown. We leave a detailed analysis for future work.

1. Introduction
2. Design & Implementation
3. Evaluation
4. **Conclusion**

Conclusion

- Thrend
  - The first defense against counting threads
  - Accuracy: 99.71% (100 Hz)
  - Performance overhead: 1.42% (100 Hz)





THE CHIPS TO SYSTEMS CONFERENCE
63
SPONSORED BY:
CEDA
IEEE Council on Electronic Design Automation
SIGDA
acm

Thanks

Thrend: Mitigating Counting Thread-based Fine-grained Timing on Arm and Apple CPUs

Chang Liu*, Shuaihu Feng*, Jiaqi Cui, Hongpei Zheng, Aodong Cui,
Jian Dong, Yuan Li, Trevor E. Carlson, Dongsheng Wang