Securing SoCs from Time Side Channels

Jose Renau

Esperanto Technologies

An Attacker can use Time Leaks to Infer Usage

```
IPC  = 0.9
Time…
Misses…
```

```
IPC  = 1.1
Time…
Misses…
```

```
IPC  = 0.6
Time…
Misses…
```

Anatomy of an Attack

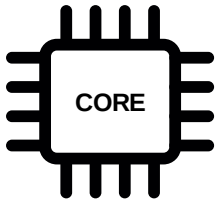
- Victim **runs code that leaks** observable side-effects.
- Attacker runs code that is **affected by timing leaks**.
- Attacker **measures time** his code took to run.

- Can we prevent to run code? (not clear)
- Can we avoid time leaks? (the goal of this)
- Can we avoid measuring time? (not really)

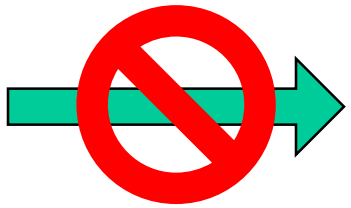
Time Domain

One Time Domain should not affect the performance of another Time Domain

Time Domain 1
IPC1

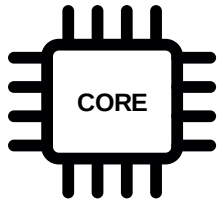



```
add  s0,sp,48
sw   a0,-36(s0)
sd   a1,-48(s0)
lw   a5,-36(s0)
sw   a5,-20(s0)
sw   zero,-24(s0)
j    .L2
```



Time Impact

Time Domain 2
IPC2

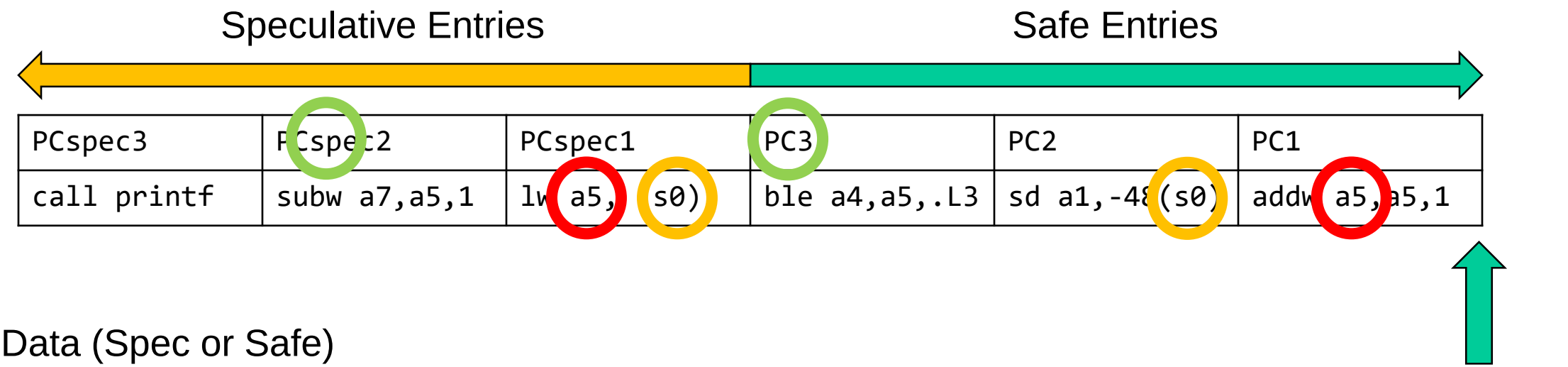



```
lw   a4,-20(s0)
lw   a5,-36(s0)
addw a5,a4,a5
sw   a5,-20(s0)
lw   a5,-24(s0)
addw a5,a5,1
sw   a5,-24(s0)
```

Outline

- Categorize Time Leaks
- High level techniques
- Sample cores

What can Time Leak?



D: Data (Spec or Safe)

A: Addresses (Spec or Safe)

P: Program Counter (Spec or Safe)

E: Execution Time (Spec or Safe)

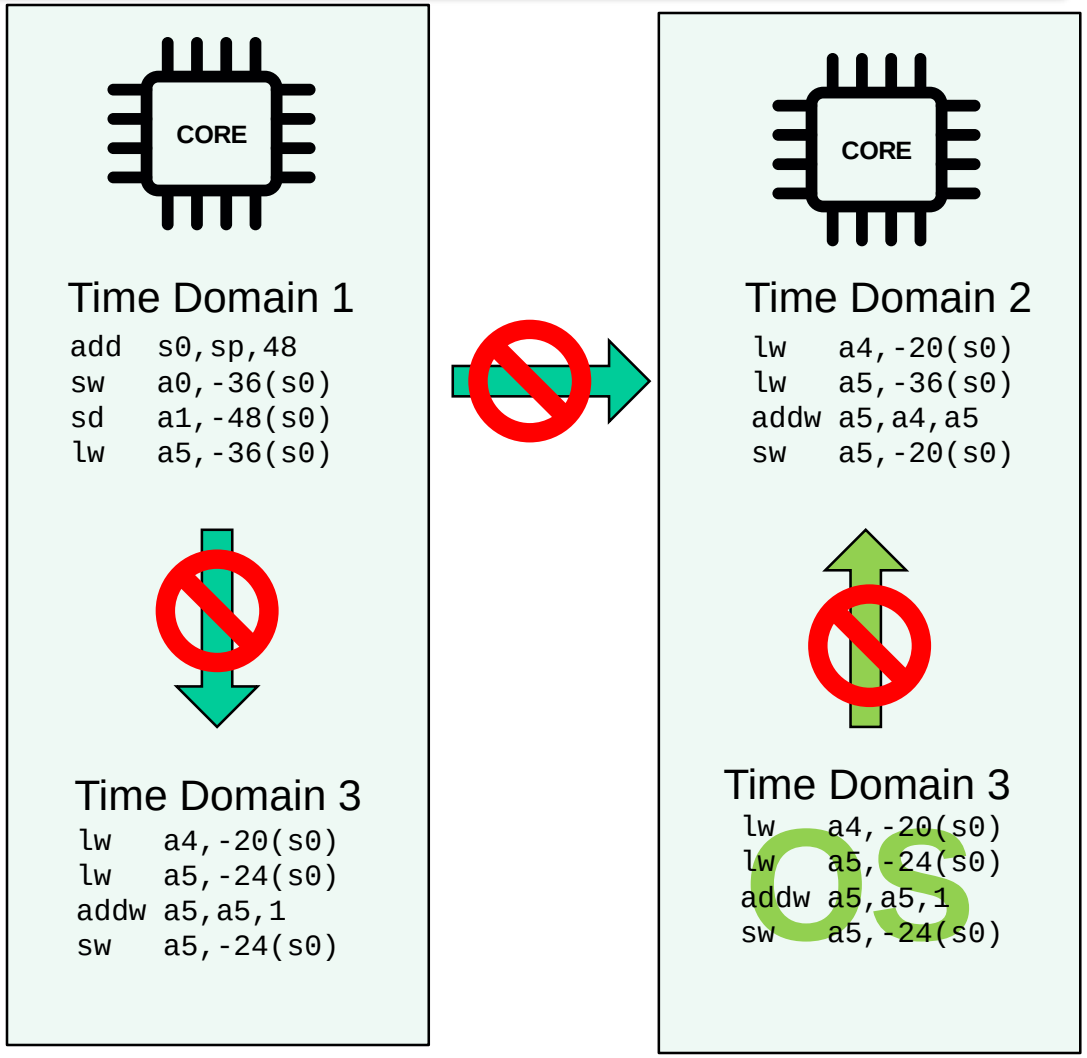
Where can Time Leaks go?

SC: no Same Core leaks

MC: no Muti-Core leaks

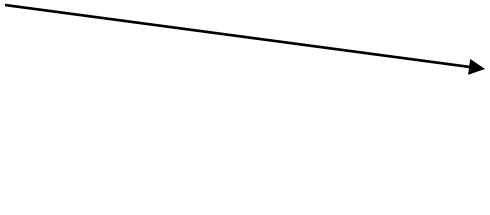
OS: no Operating System leaks

AI: no Application Interface leaks



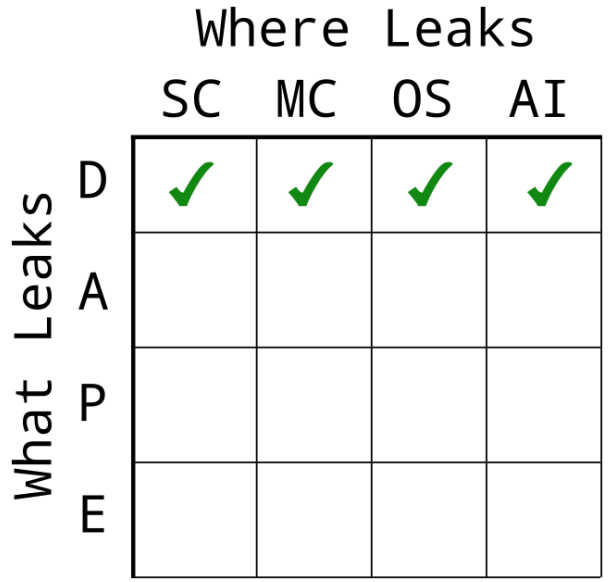
What is a Safe.AI? (Application Interface)

```
x = get_time
Call Alien_API
```

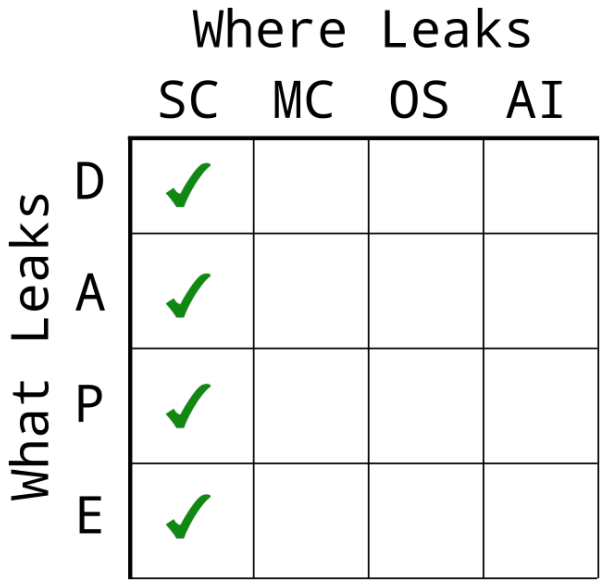


```
Wait Response
y = get_time
```

Time Leaks Categorization

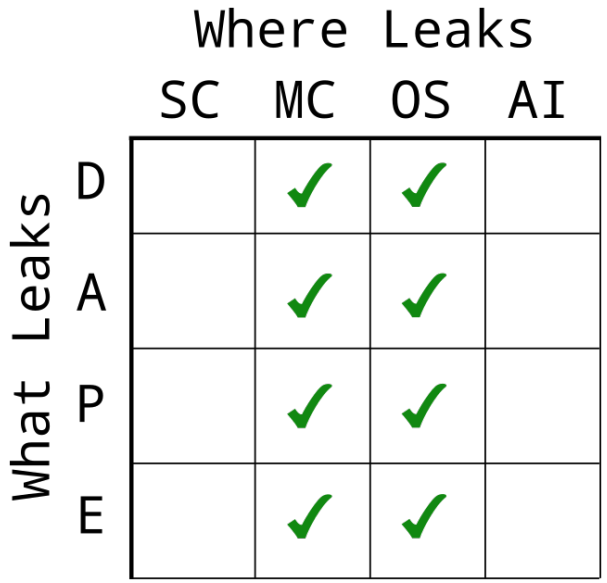


Safe[D].ALL core

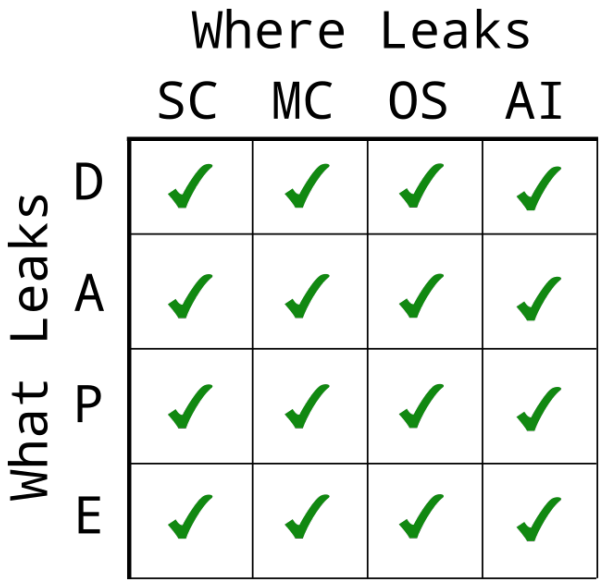


Safe.SC core

Time Leaks Categories



Safe.MC.OS



Spec.ALL

Interesting Cases

- Google/amazon may want
  - Safe.MC.OS, Spec.ALL
- Desktop/Phone
  - Safe.SC.MC, Spec.ALL
- Autonomous cars
  - Spec.MC, Spec.ALL and use OS to separate per core
- Router/switch
  - Safe.nothing?
- IOT
  - Unclear, web browser IOT, maybe Spec.??, Spec.ALL

Outline

- Categorize Time Leaks
- High level techniques
- Sample cores

High Level Idea

- For Spec:
  - Do not make speculation visible
- For Safe:
  - Do not make any traffic visible to other time domains

Tech 1: Point of No Return (PNR)

- Not all the instructions are speculative
- PNR in ROB indicates point were instruction is guaranteed to eventually commit

- SPEC2006 for an A72-like core ~25% of the instructions are beyond the PNR

Tech 2: No Speculative Update Predictors

- Delay and/or perfect fixup all the predictors
  - Good: Minor/no performance impact
  - Bad: More buffering
- Delay TAGE/BTB/… updates until retirement
  - No performance impact!
- Return Address Stack (RAS) and GHR
  - Have a copy at retirement, perfect fix at misspredication
- Delay AMPM/Stride/xxx prefetch updates to retirement
  - Even better performance, no pollution

Where Leaks

	SC	MC	OS	AI
D	✓	✓	✓	✓
A	✓	✓	✓	✓
P	✓	✓	✓	✓
E				

SPEC

Tech 3: No Speculative Cache

- No inclusive/exclusive L1/L2/L3 caches
- Allocate lines only to Store Completion Buffer (SCB) on cache miss
  - When combined with PNR, very few entries in SCB
  - For a 4-way OoO core and SPEC2006, 3 entries cover over 99% of the cases

- Issues: Request does not change any state until passes PNR
  - If it triggers coherence, it has to wait until passes PNR
  - Cache LRU updated at retirement, not execute

  - Possible to add “speculative buffers” for each cache

Where Leaks

	SC	MC	OS	AI
D	✓	✓	✓	✓
A	✓	✓	✓	✓
P	✓	✓	✓	✓
E				

SPEC

Tech 4: Constant Miss-Speculation Recovery

- No resources pin down by speculative flushes
  - No long latency resource pindown (sqrt/div/…)
  - No miss queue/MSHR

Where Leaks

	SC	MC	OS	AI
D	✓	✓	✓	✓
A	✓	✓	✓	✓
P	✓	✓	✓	✓
E	✓	✓	✓	✓

SPEC

Tech 5: Partition and BW isolation

- Private L1 and L2
- Many papers showing how to data partition LLC
- Bias designs towards systems with smaller amount of shared LLC
- Partition the directory, LLC, and any resource

Where Leaks

	SC	MC	OS	AI
D	✓	✓	✓	✓
A	✓	✓	✓	✓
P	✓	✓	✓	✓
E	✓	✓	✓	✓

SPEC

Tech 6: Partial/Full Flush

- Flush entries at Time Domain switch
- Flush can be partial like 2 LRU ways or X entries

Tech 7: Partial/Full Off-load

- Off-load/Re-load part of structures at Time Domain switch
- E.g:
  - Off-Load Data L1 MRU lines on Time Domain switch
  - Flush the other Data L1 entries

Tech 8: Dynamic every ?M cycles

- Dynamic partition/repartition/adaptation can increase throughput/utilization

- OK to dynamically adapt as long as it does not leak
  - Adapt to a slow moving average infrequently (millions of cycles)
  - Trade-off performance vs security

Other Techniques (not show due to time)

- Tech 9: Dealing with Row Hammer
- Tech 10: BW isolation per Time Domain
- Tech 11: QoS for network on chip
- Tech 12: Non-value dependent operation latency (no telescopic units)
- Tech 13: Regularization
- Tech 14: Homogenization

Outline

- Categorize Time Leaks
- High level techniques
- Sample cores

Spec.ALL Techniques

- Overall different structure
  - Private L1/L2 non-inclusive caches, small non-inclusive L3
  - Extra “fake” physical address bits to indicate Time Domain id (core and OS/User)
  - Inclusive L1 and L2 TLB. Fast L2 to L1 TLB update
  - L2 instruction cache (IL2)
- Core
  - Tech 1: PNR
  - Tech 2: No Speculative Update Predictors
    - TAGE/ITTAGE/BTB/Prefetcher, stride prefetcher uses retire address + extra offset
  - Tech 3: No Speculative Cache
    - Non inclusive L1/L2/L3. Allocate on L1 SCB
  - Tech 4: Constant spec flush time
- SoC
  - Tech 5: Dynamic Partition
    - Fixed partition LLC, directory
    - Dynamic partition: bandwidth, Memory Controller (open pages, row hammer)
  - Tech 10: Memory BW partitioning per core

Safe.MC Techniques

- Comes for free once you have the previous Spec.ALL
  - Tech 5 (Dynamic Partitioning) and Tech 10 (BW)

Mostly, Spec.ALL already provides Safe.MC

Safe.OS Techniques

- Tech 5: Dynamic Partition if frequent OS calls
  - Dynamic partition: L2 TLB, L2, Memory Dependence Predictor
  - No L2 allocate by OS unless frequent OS
- Tech 6: Full/Partial Flush on OS return
  - Memory Dependence predictor, L1 TLB, Memory Controller
  - Data L1, Instruction L1, BTB, ITTAGE
- Tech 8: Dynamic adapt every SoC partitioned resources every 10M cycles
  - L2, L2 TLB, BTB, ITTAGE

Mostly, allocate a chunk of core predictors to OS or flush a fixed chunk at OS return

Safe.SC.MC.OS Techniques

- Combine Safe.MC and Safe.OS techniques, but also add
- Tech 5: Dynamic Partition
  - Dynamic partition: L2 TLB, DL2, IL1
- Tech 6: Partial Flush if infrequent Time Domain switch
  - Full Flush Memory Dependence predictor, L1 TLB
  - Partial flush if partition DL2, IL2, BTB, ITTAGE
- Tech 7: Partial Off-load
  - Off-load MRU sets in data and instruction DL1 and IL1 on Time Domain switch
    - Off-load to IL1/DL2 if partition mode, flush otherwise
  - Off-load data and instruction prefetch tables for quick re-load

Mostly, flush if infrequent, partition/off-load if frequent switching

Conclusions

- Categorization of Time Leaks to classify/understand/test current SoCs
- Show several techniques that can achieve different degrees of protection
- Show techniques for Spec.ALL with small performance impact (but large redesign)
- Show techniques for Safe.OS, Safe.MC, Safe.SC
- Protection level could change at run-time

What do we need from ISA?

- Some way to indicate TimeDomain ID
- Option 1
  - Have CSR
- Option 2
  - Use the upper bits in the address space

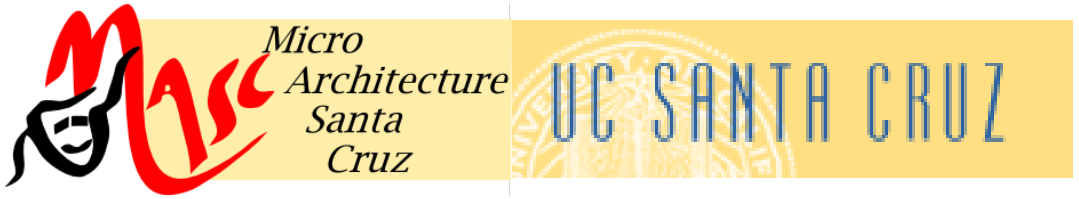
- TimeDomain ID is important to avoid overheads when unnecessary
  - E.g: no need to wait for PNR in most data sharing multithreaded apps

- Some way to expose the reconfiguration/repartitioning. An AMPM++?

Questions?

Jose Renau

Department of Computer Engineering,
University of California, Santa Cruz
http://masc.soe.ucsc.edu



- Securing Processors from Time Side Channels, Jose Renau, April 2018.