Securing High-performance RISC-V Processors from Time Speculation

Christopher Celio, Jose Renau

Esperanto Technologies

{christopher.celio, jose.renau}
@esperantotech.com

8th RISC-V Workshop
Barcelona, Spain
May 7, 2018

Esperanto Technologies

Make a contribution Subscribe Find a job

The Guardian

News Opinion Sport Culture Lifestyle

US World Environment Soccer US politics Business **Tech** Science

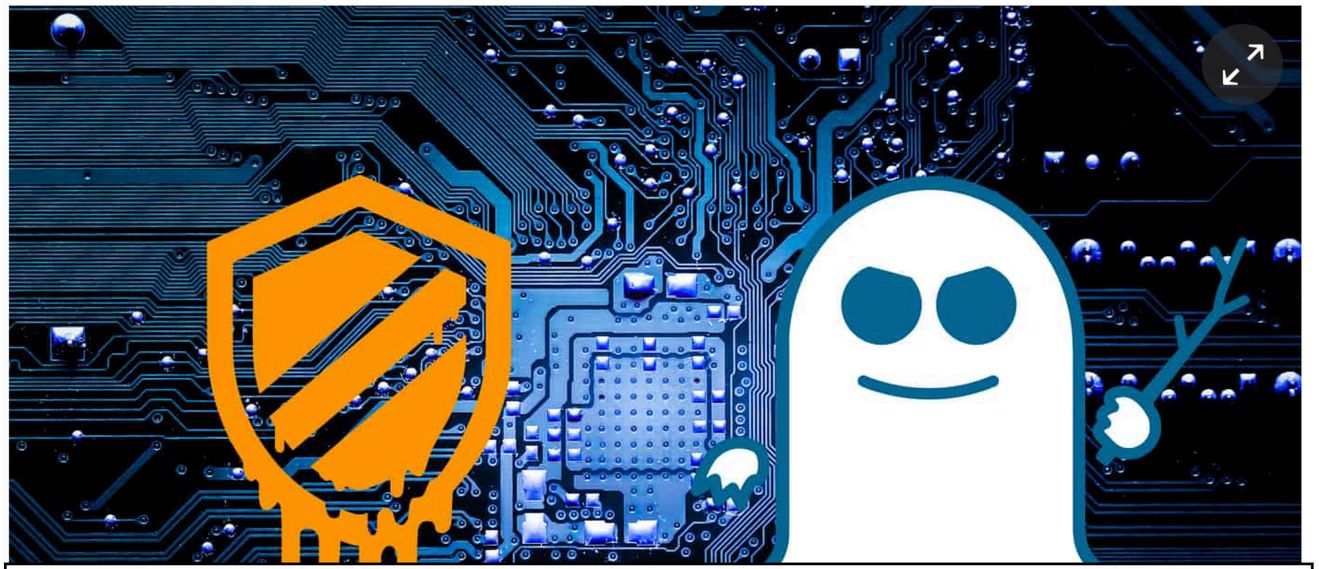
Data and computer security

Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers

Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors – and fix could slow devices

Samuel Gibbs
Thu 4 Jan 2018 07.06 EST

Spectre and Meltdown processor security flaws – explained



https://www.theguardian.com/technology/2018/jan/04/meltdown-spectre-worst-cpu-bugs-ever-found-affect-computers-intel-processors-security-flaw

The Register®
Biting the hand that feeds IT

ER SOFTWARE **SECURITY** DEVOPS BUSINESS PERSONAL TECH SCIENCE EM

Security

Intel admits a load of its CPUs have Spectre v2 flaw that can't be fixed

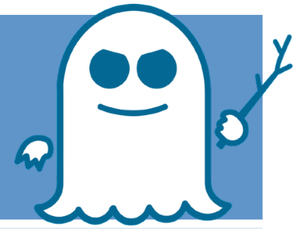
And won't fix Meltdown *nor* Spectre for 10 product families covering 230-plus CPUs

By Simon Sharwood, APAC Editor 4 Apr 2018 at 01:15 81 SHARE ▼

https://www.theregister.co.uk/2018/04/04/intel_spectre_microcode_updates/

Intel has issued fresh "microcode revision guidance" that reveals it won't address the Meltdown and Spectre design flaws in all of its vulnerable processors – in some cases because it's too tricky to remove the Spectre v2 class of vulnerabilities.

Timing Attacks Go Mainstream



	Meltdown	Spectre
Attacker	user attacking OS	sandboxed code; user attacking OS (or other user)
Exposes	all physical memory	sandbox escape; kernel memory to user (or user to other users)
Impacted	20 years of Intel CPUs; some ARM	**Virtually ANY CPU that uses speculation**
Mitigations	Patch OS to move OS memory out of user's address space	**???** microcode; speculation fences (ew!); **"Working as intended"**

See Spectre author Paul Kocher's talk for more details: https://www.youtube.com/watch?v=hqIavX_SCWc

https://spectreattack.com

https://googleprojectzero.blogspot.co.at/2018/01/reading-privileged-memory-with-side.html

Community Reactions

"The more I think about it, the less I understand it."
- Adrian Sampson

https://www.cs.cornell.edu/~asampson/blog/spectacular.html

Agenda

- What are timing-based attacks?
- A classification proposal
- Micro-architectural mitigations
- What should RISC-V do?

Esperanto Technologies

THE TAKEAWAY

- We can still build high-performance, speculative processors that are protected against timing attacks.
- **Without modifying the ISA.**

<u>We only need micro-architectural techniques.</u>

What is a timing attack?

When a change in your **program input** affects the **time** of another user.

What is a timing attack?

When a change in your **program input** affects the **time** of another user.

```
IPC  = 1.1
Time…
Misses…
```

Esperanto Technologies

What is a timing attack?

When a change in your **program input** affects the **time** of another user.

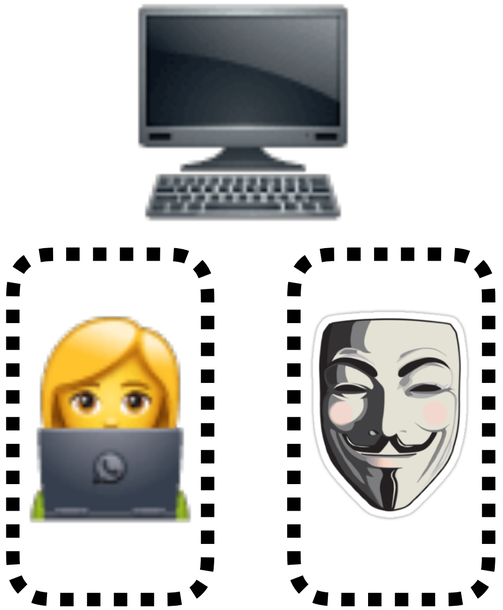
```
IPC  = 1.1
Time…
Misses…
```

```
IPC  = 0.6
Time…
Misses…
```

Esperanto Technologies

What is a timing attack?

When a change in your **program input** affects the **time** of another user.



```
IPC  = 0.9
Time…
Misses…
```

```
IPC  = 1.1
Time…
Misses…
```

```
IPC  = 0.6
Time…
Misses…
```

Esperanto Technologies

An example Spectre attack: a user attacks the OS

```
int close(int fd) {
  if (fd < process.file_num) { // bounds check; speculate "fd" is in bounds.
    file = process.files[fd];
    vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

Esperanto Technologies

An example Spectre attack: a user attacks the OS

```
int close(int fd) {
  if (fd < process.file_num) { // bounds check; speculate "fd" is in bounds.
    file = process.files[fd];
    vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)

Esperanto Technologies

An example Spectre attack: a user attacks the OS

```
int close(int fd) {
  if (fd < process.file_num) { // bounds check; speculate "fd" is in bounds.
    file = process.files[fd];
    vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

← untrusted input

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)
- malicious user invokes close() syscall using **untrusted input** (int fd)

Esperanto Technologies

An example Spectre attack: a user attacks the OS

```
int close(int fd){ ← untrusted input
  if (fd < process.file_num){ // bounds check; speculate "fd" is in bounds.
    file = process.files[fd];
    vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)
- malicious user invokes close() syscall using **untrusted input** (int fd)

An example Spectre attack: a user attacks the OS

```
int close(int fd){ ← untrusted input
  if (fd < process.file_num){ // bounds check; speculate "fd" is in bounds.
   file = process.files[fd];
    vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)
- malicious user invokes close() syscall using **untrusted input** (int fd)
- **process.files[fd]** loads a **secret key**

Esperanto Technologies

An example Spectre attack: a user attacks the OS

```
int close(int fd){ ← untrusted input
  if (fd < process.file_num){ // bounds check; speculate "fd" is in bounds.
   file = process.files[fd];
   vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)
- malicious user invokes close() syscall using **untrusted input** (int fd)
- **process.files[fd]** loads a **secret key**
- **.vtable loads an address (x)** based on the value of the **key**

An example Spectre attack: a user attacks the OS

```
int close(int fd){ ← untrusted input
  if (fd < process.file_num){ // bounds check; speculate "fd" is in bounds.
   file = process.files[fd];
   vtable = file.vtable;
    vtable.close(process.files[fd]); process.files[fd] = 0; return 0;
  } else {
    return -EBADF;
  }}
```

- malicious user clears out cache and fills it up with a **dummy** array (PRIME)
- malicious user invokes close() syscall using **untrusted input** (int fd)
- **process.files[fd]** loads a **secret key**
- **.vtable loads an address (x)** based on the value of the **key**
- malicious user can then PROBE his **dummy** array to deduce **(x)**

Anatomy of a timing 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.

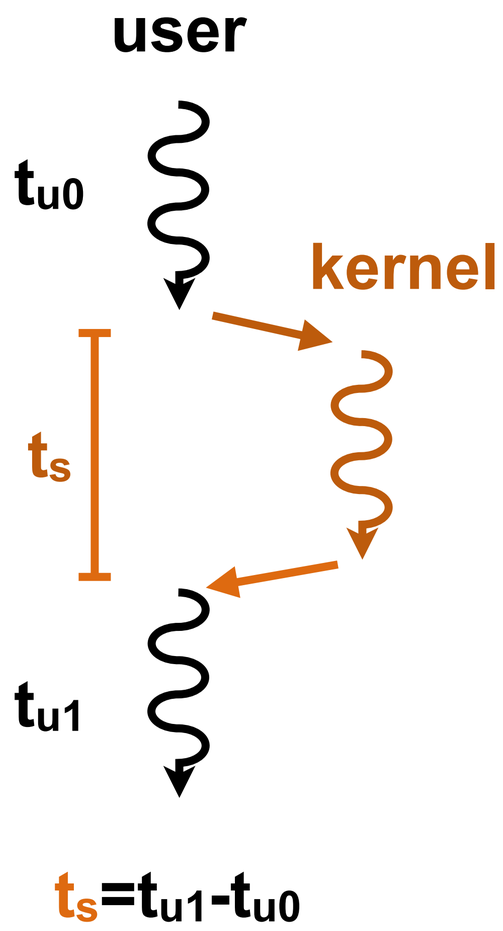
- Where should we break the chain?
  - Can we prevent victim from running bad code?
  - Can we prevent side-effects from being observable to other Time Domains?
  - Can we prevent attackers from measuring time?

Can we block the clock?

- Chrome reduced the resolution of performance.now().
- But attackers will just build their own timers.

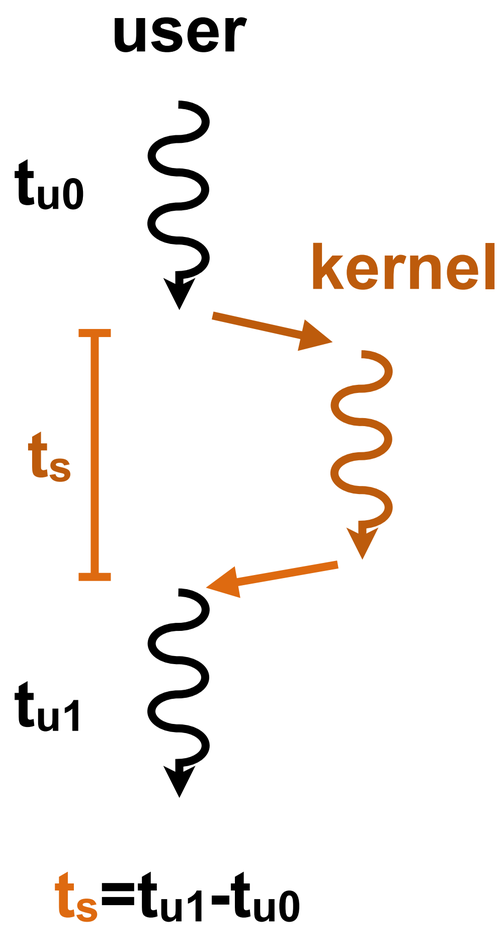
Global Clocks vs Local Clocks

- Global clock
  - establishes a consistent time across the machine
- Local clock
  - only valid for particularly locality (e.g., a user thread)
  - *could* be turned off when user thread not running
  - then a user cannot measure how long the OS took to service a request
- Coordination
  - malicious users with local clocks can coordinate to build global clocks



Global Clocks vs Local Clocks

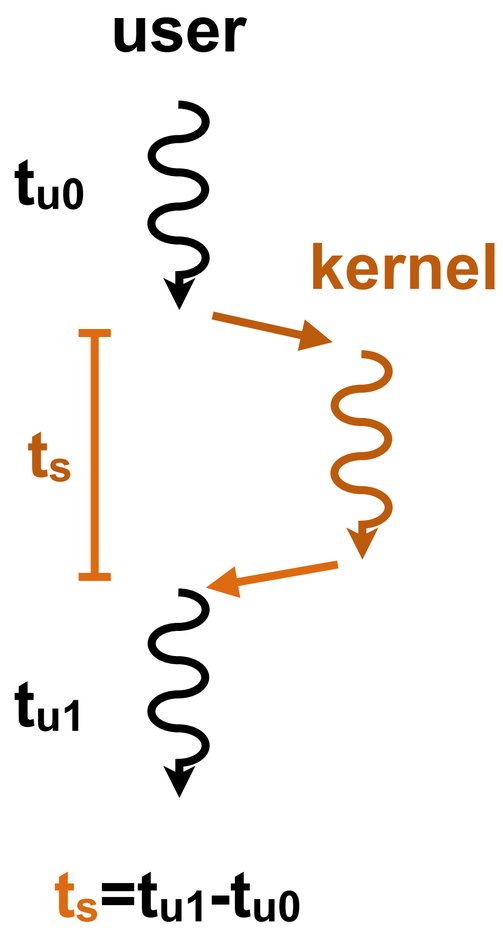
- Global clock
  - establishes a consistent time across the machine
- Local clock
  - only valid for particularly locality (e.g., a user thread)
  - *could* be turned off when user thread not running
  - then a user cannot measure how long the OS took to service a request
- Coordination
  - malicious users with local clocks can coordinate to build global clocks
- **Questions to ask yourself:**
  - What is RISC-V rdtime?
  - What is RISC-V rdcycle?



Global Clocks vs Local Clocks

- Global clock
  - establishes a consistent time across the machine
- Local clock
  - only valid for particularly locality (e.g., a user thread)
  - *could* be turned off when user thread not running
  - then a user cannot measure how long the OS took to service a request
- Coordination
  - malicious users with local clocks can coordinate to build global clocks
- **Questions to ask yourself:**
  - What is RISC-V rdtime?
  - What is RISC-V rdcycle?

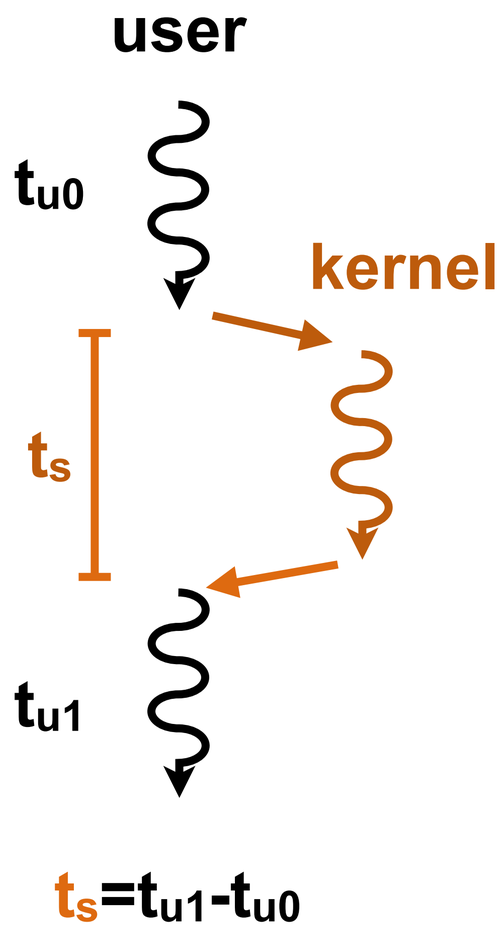
counts up from "some arbitrary time in the past"



Global Clocks vs Local Clocks

- Global clock
  - establishes a consistent time across the machine
- Local clock
  - only valid for particularly locality (e.g., a user thread)
  - *could* be turned off when user thread not running
  - then a user cannot measure how long the OS took to service a request
- Coordination
  - malicious users with local clocks can coordinate to build global clocks
- **Questions to ask yourself:**
  - What is RISC-V rdtime?
  - What is RISC-V rdcycle?
  - "Up to the platform"!

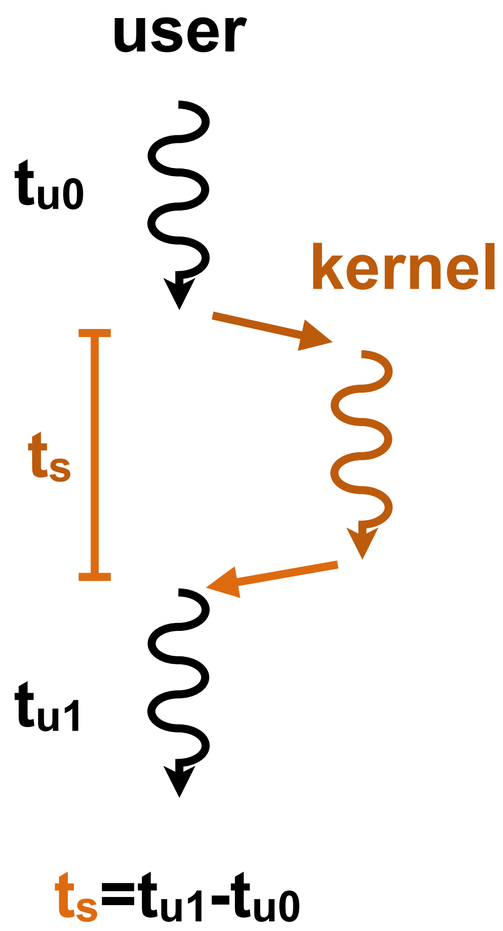
counts up from "some arbitrary time in the past"



Global Clocks vs Local Clocks

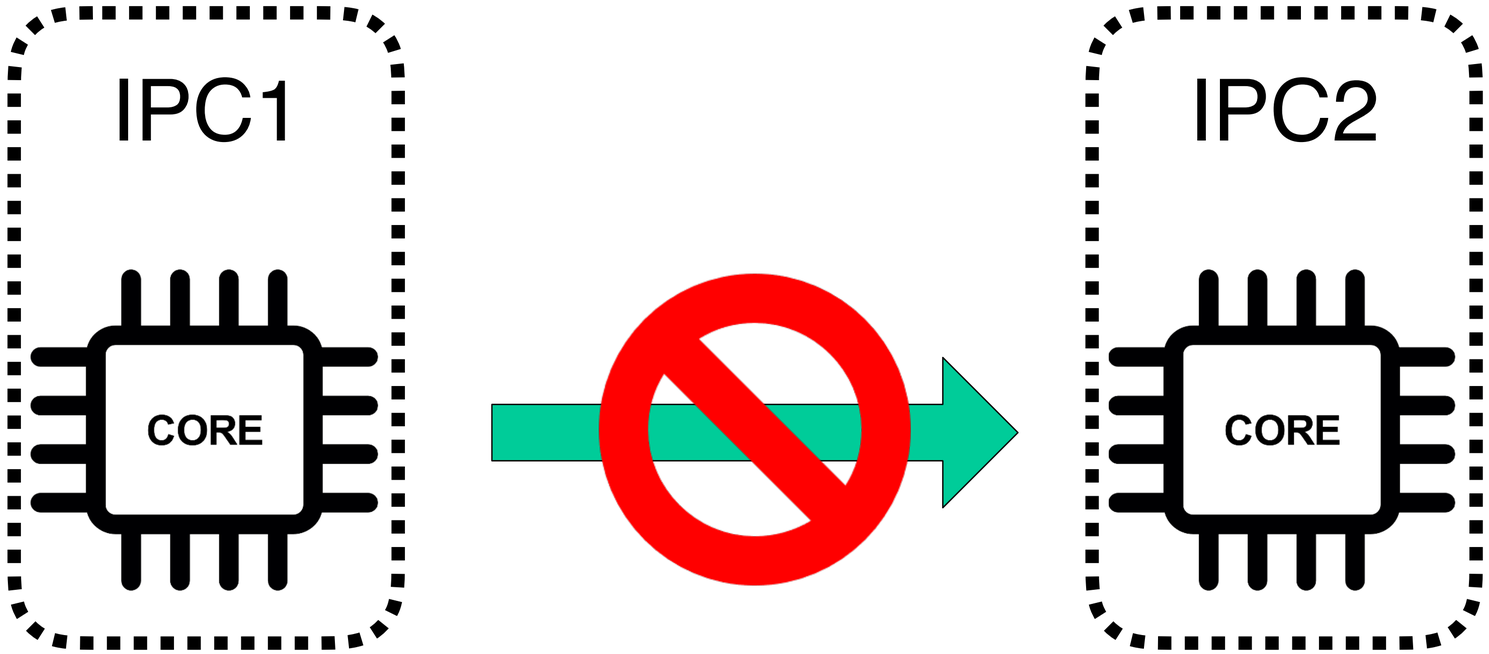
- Global clock
  - establishes a consistent time across the machine
- Local clock
  - only valid for particularly locality (e.g., a user thread)
  - *could* be turned off when user thread not running
  - then a user cannot measure how long the OS took to service a request
- Coordination
  - malicious users with local clocks can coordinate to build global clocks
- **Questions to ask yourself:**
  - What is RISC-V rdtime?
  - What is RISC-V rdcycle?
  - "Up to the platform"!
  - writeable in m-mode
  - can be trapped by OS/HV
  - How should your platform define/use these?

counts up from "some arbitrary time in the past"



Time Domains

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

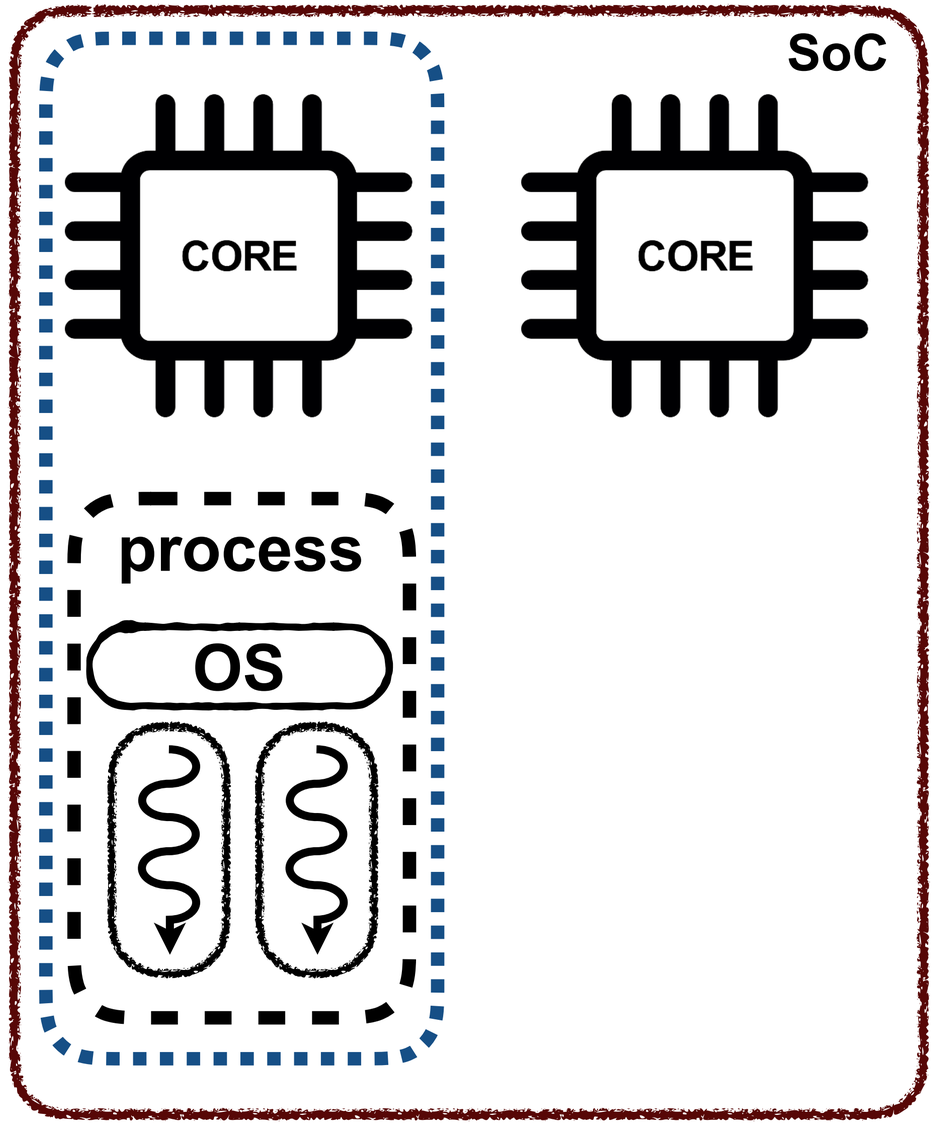


The customer+architect decides where to draw the Time Domain boundaries.

Types of Time Domains

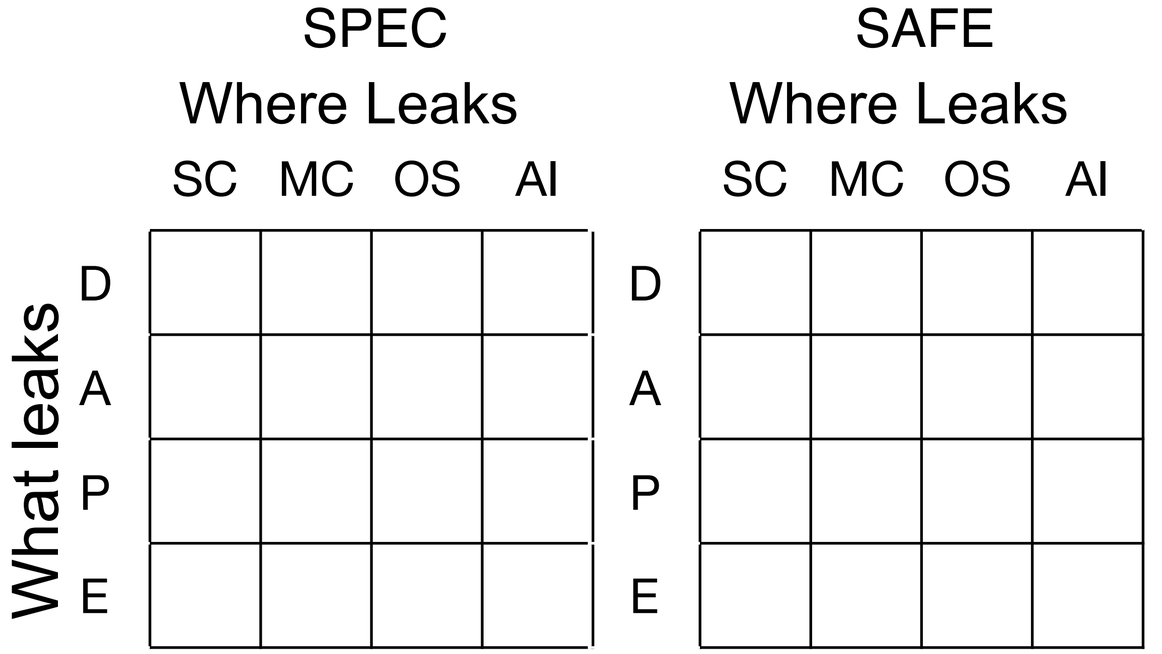
- Entire SoC (aka, today)
- Per core
- Per virtual machine
- Per user process
- Per thread
- Per hyper-thread

- Cloud providers
  - Per core/VM is fine.
- Controlled Environments
  - Per SoC may be fine.
- Our laptops?
  - Per user thread (stupid ads).



Categorizing Time Leaks

- Is the attack built using **spec**ulative execution or the committed (**safe**) instructions?
- Where are we leaking from?
  - **SC:** no same core leaks
  - **MC:** no multi-core leaks
  - **OS:** no operating system leaks
  - **AI:** no application interface leaks
- What are we directly leaking?
  - **D**ata
  - **A**ddress
  - **P**rogram counter
  - **E**xecution time



Micro-architectural Mitigations

- I will focus on techniques for Spec-based timing attacks
- These may not be exhaustive or even sufficient
- I'm not your ~~lawyer~~ CPU architect

High-level Ideas

- Don't leak any observable side-effects in the machine if speculation is aborted.
- Avoid bandwidth interference between different Time Domains.

Idea 1: The Point of No Return (PNR)

- Modern speculative cores have 100-300 instructions inflight
- Not all inflight instructions are speculative ("spec")
- PNR in re-order buffer (ROB) denotes instructions that are guaranteed to eventually commit ("safe")

- SPEC2006 for an A72-like core has **25% of instructions** in ROB are beyond the PNR

- **Conclusion:**
  - We only have to worry about 75% of instructions. \o/
  - There are enough inflight "safe" instructions that we can use to our advantage.

Esperanto Technologies

Idea 2: Do not update predictors speculatively

- The problem:
  - Spectre used caches as side-channel, but predictors can be used too.
  - branch predictors can leak PC, address, data information (as byproduct of spec execution).
  - data prefetches can leak address information.
- Delay updates until Commit (PNR) or perform "perfect" fix-up.
  - good: minor/no performance impact
  - bad: more buffering, potentially more ports needed in some cases
- Branch predictors, target buffers (BTB)
  - update at Commit
- Return Address Stack
  - Have a commit copy, rebuild perfect copy on misprediction
- Data prefetchers
  - update at Commit
  - predict using the commit stream
  - better performance, no pollution

√ partial mitigation

√ full mitigation

Where Leaks

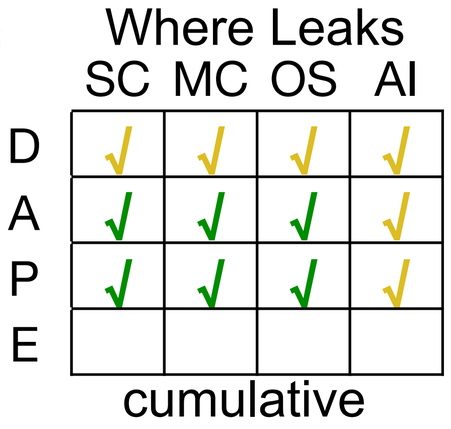
	SC	MC	OS	AI
D	√	√	√	√
A	√	√	√	√
P	√	√	√	√
E				

Idea 3: No speculative Cache updates

- Problem
  - **the method of choice** for side-channel leaking of addresses (and thus indirectly data)
  - victim and attacker share caches
  - attacker can control data in victim's caches and vice-versa
  - partitioning an option: but too heavy-handed
  - we **cannot** delay until commit or performance is dead

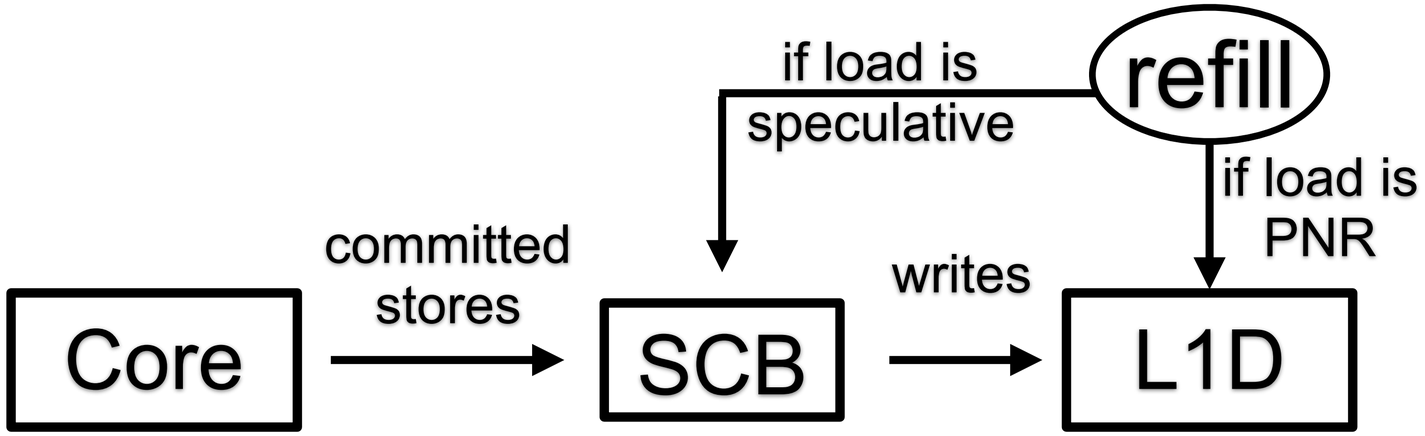
- No fully inclusive L1/L2/L3 caches
  - Neither exclusive nor inclusive (NENI)
  - "free running"
  - Great for performance anyways (but more coherence agents)

- Allocate only into the L1 cache
  - difficult to buffer (and then kill) speculative allocations in L2/L3 caches
  - continued...



Idea 3: No speculative Cache updates

- Allocate only into the L1 cache
  - difficult to buffer (and then kill) speculative allocations in L2/L3 caches
- Option 0
  - if new cache line comes back and the load is PNR, write directly into L1D
- Option 1
  - allocate speculative cache lines into **Store Completion Buffer** (SCB)
  - SCB already handles writing store data to L1D, write coalescing, and bypassing to dependent loads
  - takes entries away from regular stores (SPEC2006: only need 1-3 entries)



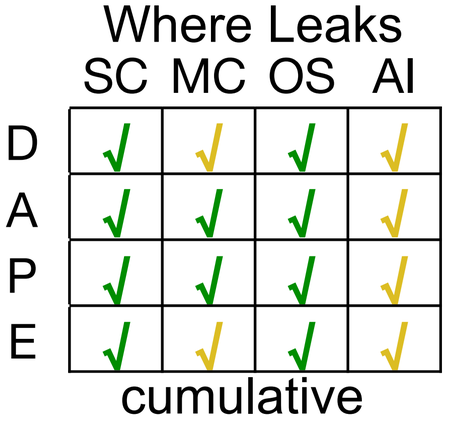
Where Leaks

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

Idea 4: Misspeculation recovery should be deterministic (for Spec.SC)

- Caches aren't the only side-channel available
- Any shared resource can leak time
  - (ALUs, predictors, Regfile ports...)
  - contention for these resources can leak time
  - resources must be deterministically released on a misspeculation

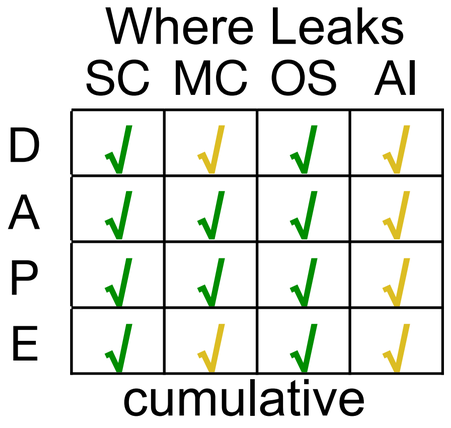
```
// Scenario: sandbox bypassing a bounds check
// Assume: fsqrt is variable-latency; single-occupancy; cannot be killed
t0 = rdtime()
if (x is in bounds) { // bounds check
   key = array[x] // access out-of-bounds memory
   fsqrt(key)  // execution time depends on key value
}
fsqrt(dontCare) // contention on fdiv betrays key value
time = rdtime() - t0
```



Idea 4: Misspeculation recovery should be deterministic (for Spec.SC)

- On a misspeculation, how quickly can you deallocate/recover?
  - inflight cache misses (MSHRs, inflight load entries)
  - renamed physical registers
  - occupied functional units
  - return address stack entries
- Caveat
  - These examples are only required for Spec.SC
  - Maybe sandboxes and privileged JITs are overrated?

    (Chris's opinion, not Jose's)



Idea 5: Partition

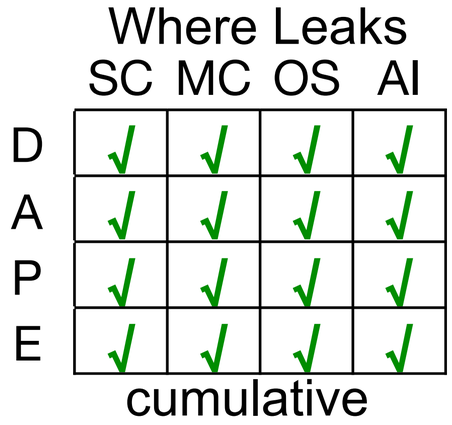
- Give cores Private L1 and L2 caches
  - Assuming the whole core is a Time Domain
  - Lots of existing literature on managing, partitioning LLCs
- Partition uncore bandwidth
  - Speculative memory accesses use bandwidth too
  - prevent other Time Domains from perturbing the observed bandwidth

	Where Leaks			
	SC	MC	OS	AI
D	√	√	√	√
A	√	√	√	√
P	√	√	√	√
E	√	√	√	√

Esperanto Technologies

Idea 6: Drain speculative pipeline on Time Domain Switch

- You probably already do this, but just to state it clearly.

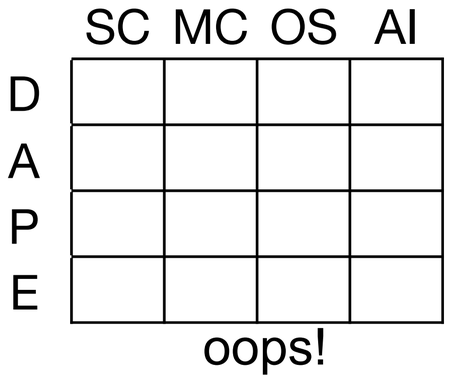


Idea 7: Dynamic Partitioning

- Actually, dynamic partitioning leaks information.
  - How much one Time Domain has been allocated leaks information about what the other Time Domains are (or are not) doing.

- But...

- Dynamic partitioning improves throughput/utilization/ performance.
- To improve security, adapt partitions infrequently using a slow-running average.



Idea 8: Partial & Full Flushes

- Flush entries on Time Domain switch
  - Many entries are not useful across Time Domains anyways.

- Branch Target Buffer Flush and Partition Example
  - Goal: prevent the User from controlling (or measuring) the OS.
  - But full flush on a privilege mode switch hurts User performance.
  - Instead, flush 1/4 BTB entries and hand them to the OS.
  - Lazily deallocate OS entries if application performs frequent OS syscalls.

Idea 9: Partial/Full Offload

- Problem
  - Some resources are long-lived (across Time Domains) and painful to repopulate (only for safe).

- TLBs are small (little state to offload) but painful if unlearned.
- L1D: off-load the MRU entries on Time Domain switch and flush the rest of the L1D.

This is it. This is the ONLY ISA-visible change I will propose. And it's not even necessary. Just nice for performance.

- **This requires an ARCHITECTURAL CHANGE**
  - On a Time Domain switch the OS/hypervisor must negotiate with the CPU when and where to offload "Opaque State" to.
  - Intel has a XSAVE/XRESTORE instruction.

Other ideas (not enough time to discuss)

- Full (or partial) flushes
- Full (or partial) offloading
- Dealing with Row Hammer
- Bandwidth isolation per Time Domain
  – specifying a TD id in the address bits
- Non-value dependent operation latency
  – no early-outs
- Regularization
  – create fake events/traffic (e.g., keyboard interrupts)
- Homogenization
  – equalize the time of API calls to protect Application Interfaces
- How do you test for timing leaks?
- How do you protect against non-speculative timing attacks?

Conclusion

- What should RISC-V do?
  - Nothing. Don't add instructions (except perhaps SAVE/RESTORE for perf).
  - Facilitate and communicate good uarch design.
  - More collaboration on RISC-V Platform Specifications
    - many implementation decisions affect security.
    - Perf counter visibility should not cross time domain boundaries.
- Let's share our ideas
  - This is too important to get wrong.
  - Use our ideas (we want RISC-V to be a successful ISA).
- Please provide feedback to us!
  - We want to get this right.