Micro-architecture Attacks

Chengyu Song

Slides modified from
Nael B. Abu-Ghazaleh and Daniel Gruss

Micro-architecture

- Architecture: hardware features that are exposed to software via Instruction Sect Architecture (ISA)
- Micro-architecture: hardware features that are "transparent" to software

Architecture States

- What are architecture states?

Architecture States

- What are architecture states?
  - Registers
    - General registers
    - Configuration registers
  - Page tables
  - Memory (both virtual and physical)
  - Devices

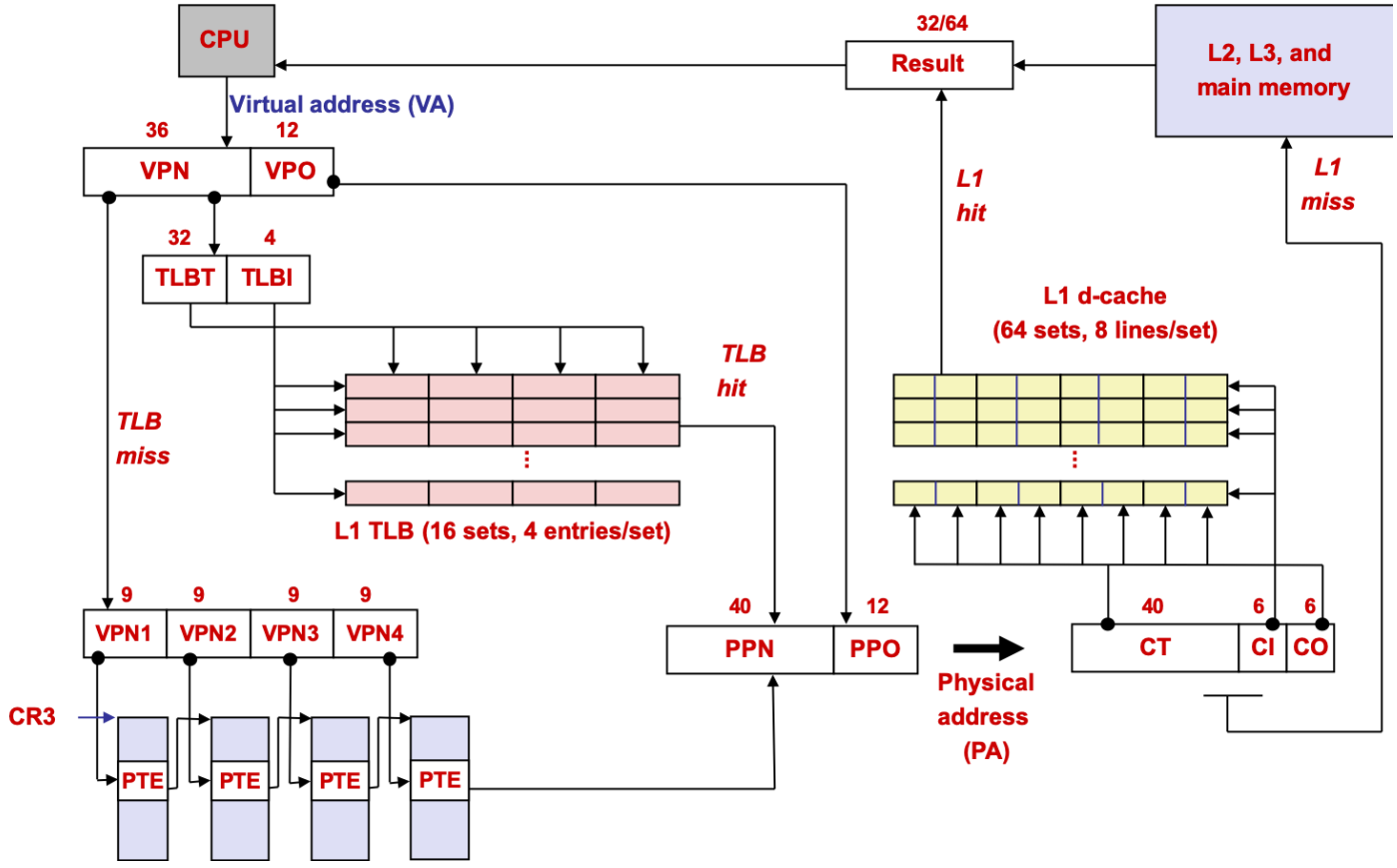
Micro-architecture States

- What are micro-architecture states?

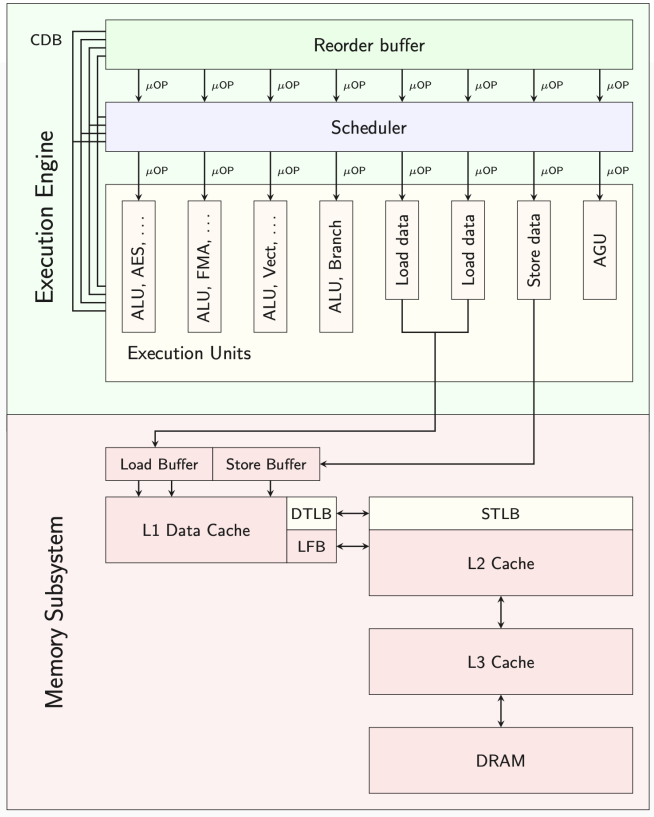
Micro-architecture States

- What are micro-architecture states?
  - Execution Units
  - TLB (translation lookahead buffer)
  - Caches
  - Predictors
  - Re-ordering buffer
  - Load/Store/Line-Fill Buffers

Review: address translation



Review: memory access



Review: security guarantees from hardware

- Privilege separation
  - Privileged instructions
  - Privileged registers/configurations
- Memory isolation
  - Kernel / userspace isolation
  - Virtual address space isolation
  - Virtual machine / host isolation
  - Trusted execution environment (SGX, TrustZone)

Micro-architecture attacks

- Breaking the isolation boundaries
- How?
  - Through **side-channels**

Review: side-channels

- What are side-channels?

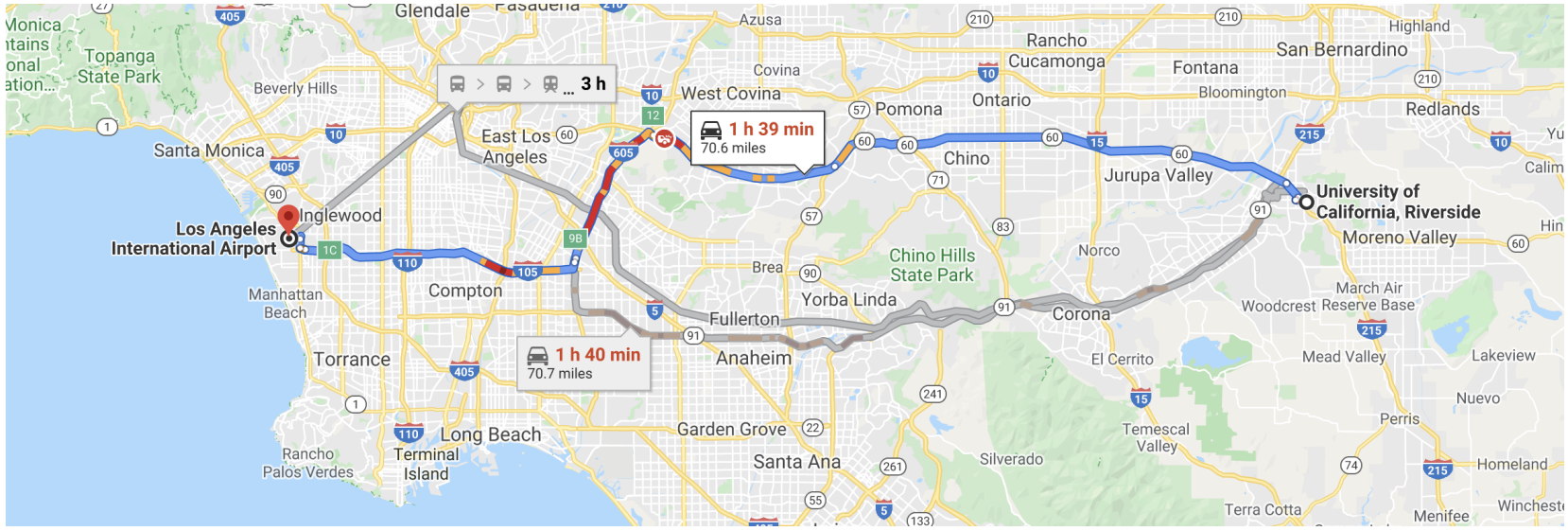
Review: side-channels

- What are side-channels?
- Types of side-channels?

Review: side-channels

- What are side-channels?
- Types of side-channels?
  - **Timing**, access pattern, power consumption, electromagnetic, acoustic, etc.

Review: timing side-channel



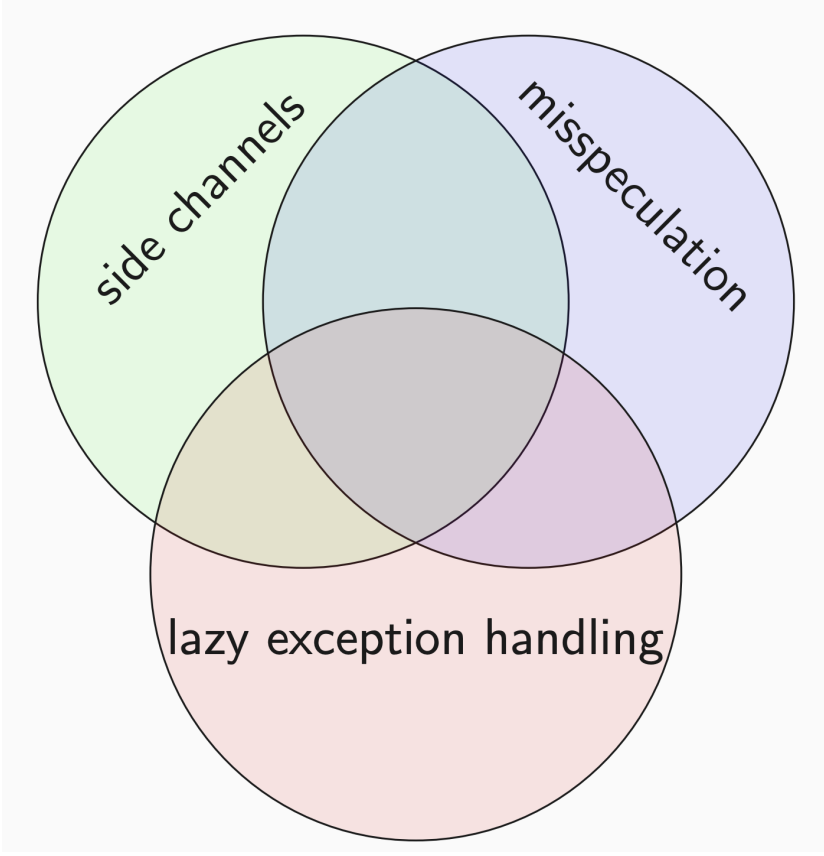
Micro-architecture timing side-channels

- What can cause timing differences?

Micro-architecture timing side-channels

- What can cause timing differences?
  - Caches: data/instruction caches, TLB, predictors
  - Execution unit: different instructions take different time to finish
  - Contention: competing -> waiting

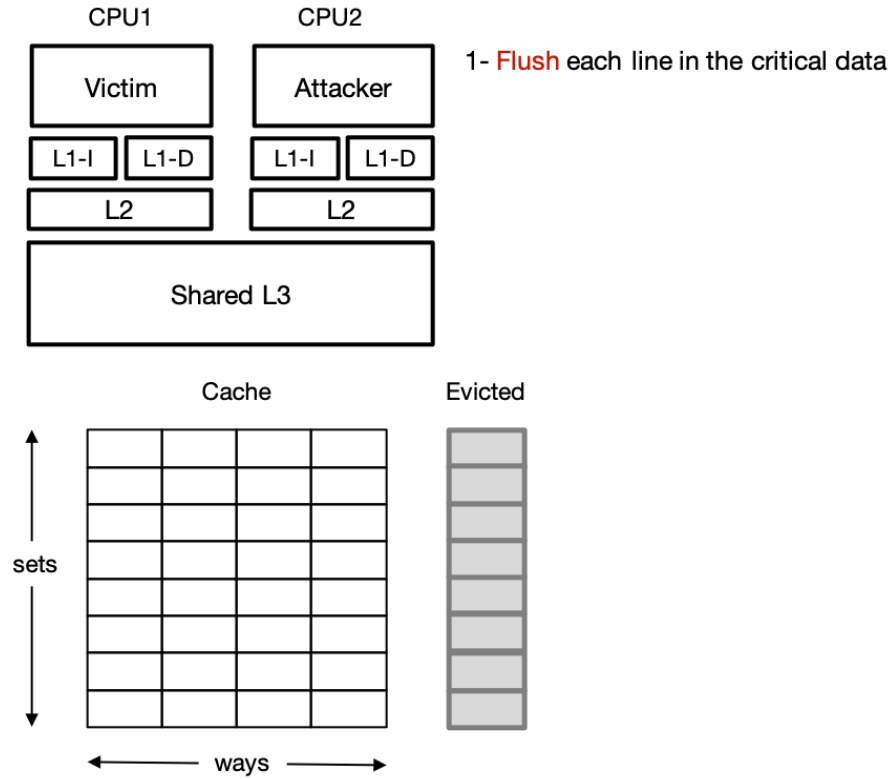
The Era of micro-architecture attacks



General steps

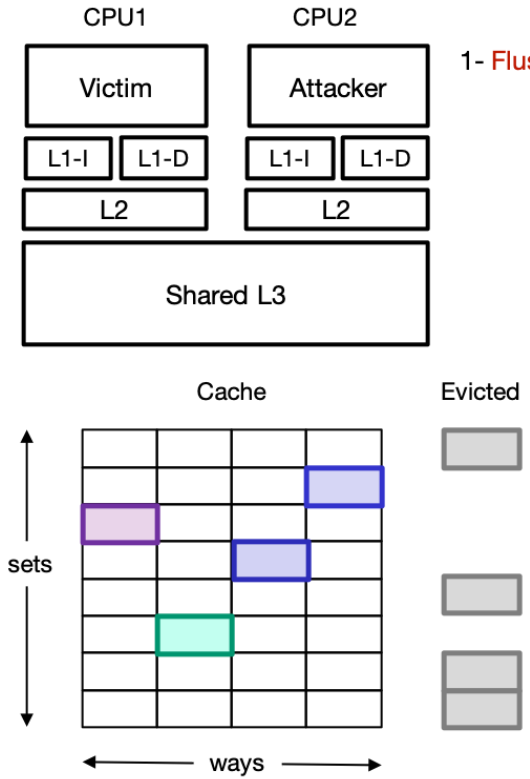
1. Access the secret
2. **Leak the secret**

Flush + Reload



Flush + Reload

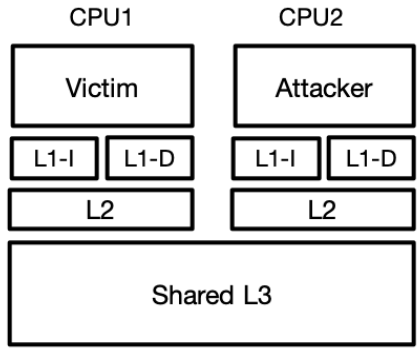
2- Victim accesses critical data



1- Flush each line in the critical data

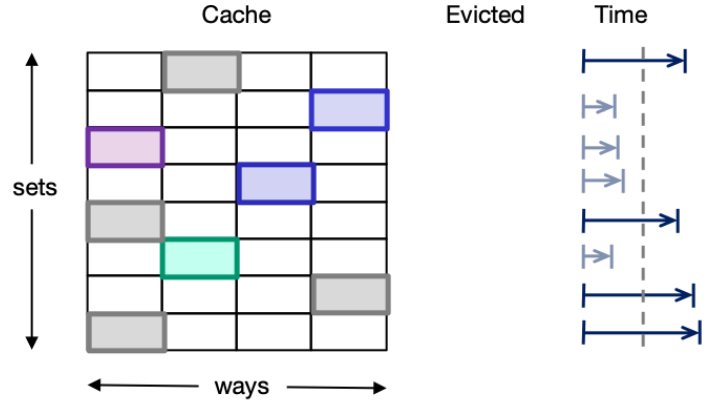
Flush + Reload

2- Victim accesses critical data

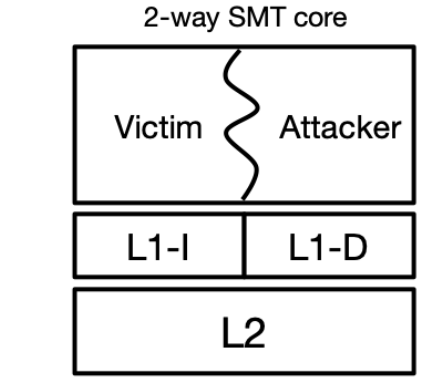


1- Flush each line in the critical data

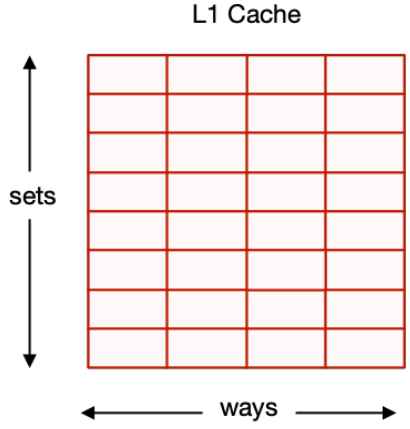
3- Reload critical data (measure time)



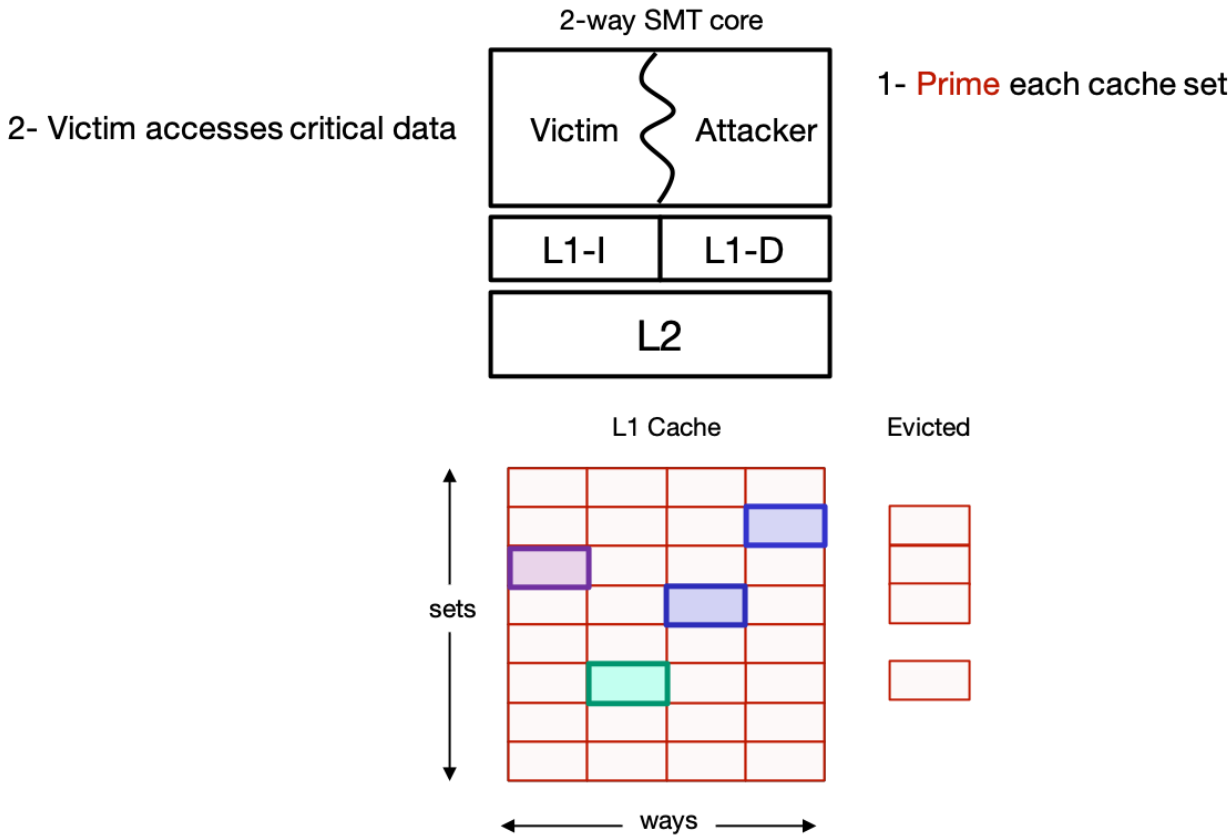
Prime + Probe



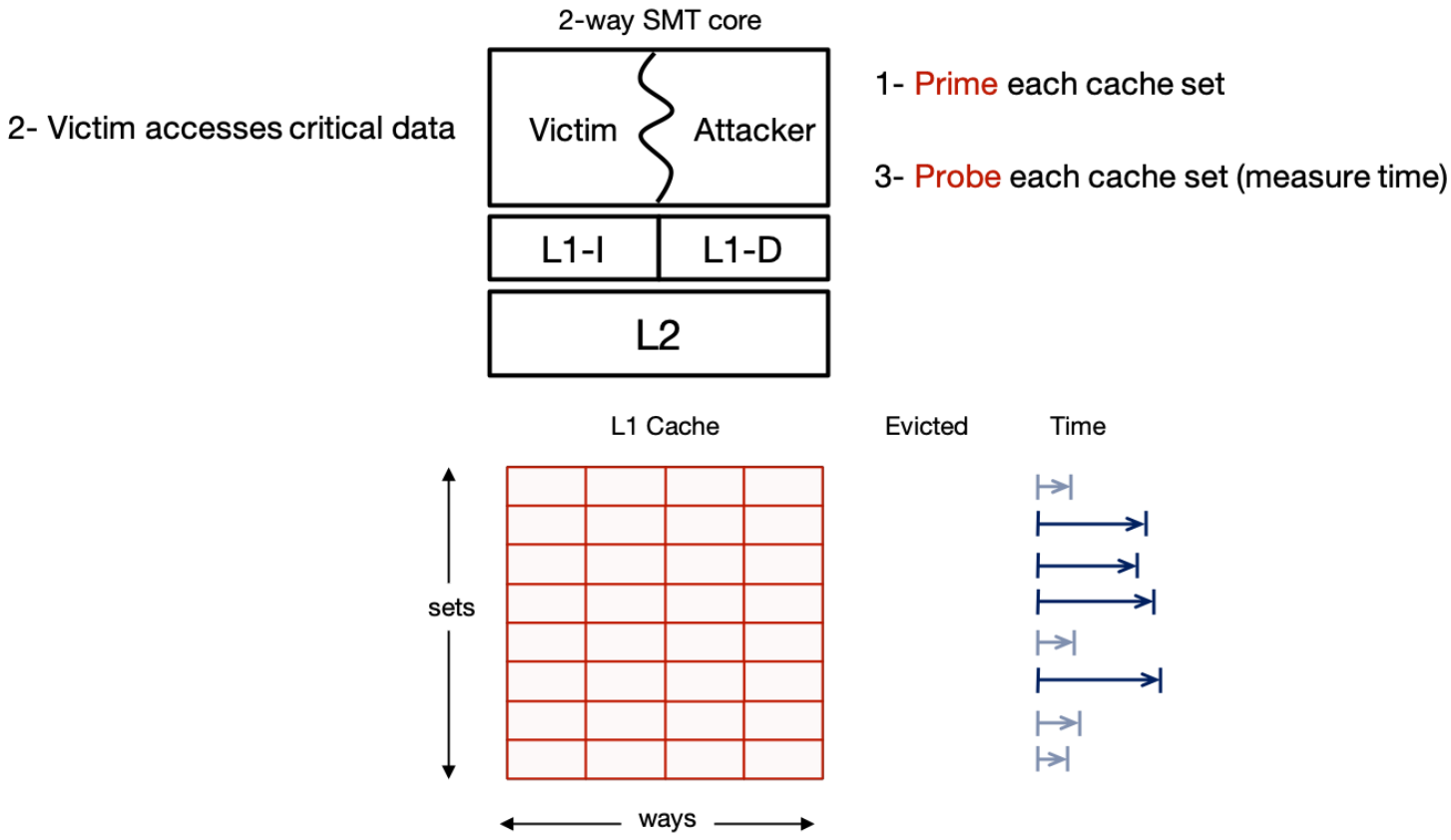
1- Prime each cache set



Prime + Probe


2-way SMT core
2- Victim accesses critical data
Victim
Attacker
1- Prime each cache set
L1-I
L1-D
L2
L1 Cache
Evicted
sets
ways

Prime + Probe


2-way SMT core
Victim
Attacker
L1-I
L1-D
L2
2- Victim accesses critical data
1- Prime each cache set
3- Probe each cache set (measure time)
L1 Cache
Evicted
Time
sets
ways

Challenges

- How to get high resolution timer?
- How to find the secret data?
  - L1: VIVT or VIPT
  - L2/L3: PIPT
  - L3 (LLC): shared/not shared, inclusive/non-inclusive

Other side-channels

- In TLB or not
- In BTB (branch target buffer) or not
- Execution Unit / Port Contention
- Path length

General steps

1. **Access the secret**
2. Leak the secret

How to access the data

- Talk to (invoke) the victim
- What if the victim will not access the secret?
  - *Force* it to access (by manipulate predictors) -> Spectre-style attacks
  - Just access it (and handle exception) -> Meltdown-style attacks

Out-of-order execution

- Idea: don't stall the pipeline, exploit instruction level
  - Schedule/issue an instruction as soon as dependencies are "ready"
- Challenge: what if there're branches?
  - Conditional: which branch to take?
  - Indirect: what's the target?

Speculative execution

- Idea: let's just guess
  - Direction prediction: pattern history table (PHT)
  - Target prediction: branch target buffer (BTB), return stack buffer (RSB)
  - More than just branch
- **The most important technique for high-performance processors**

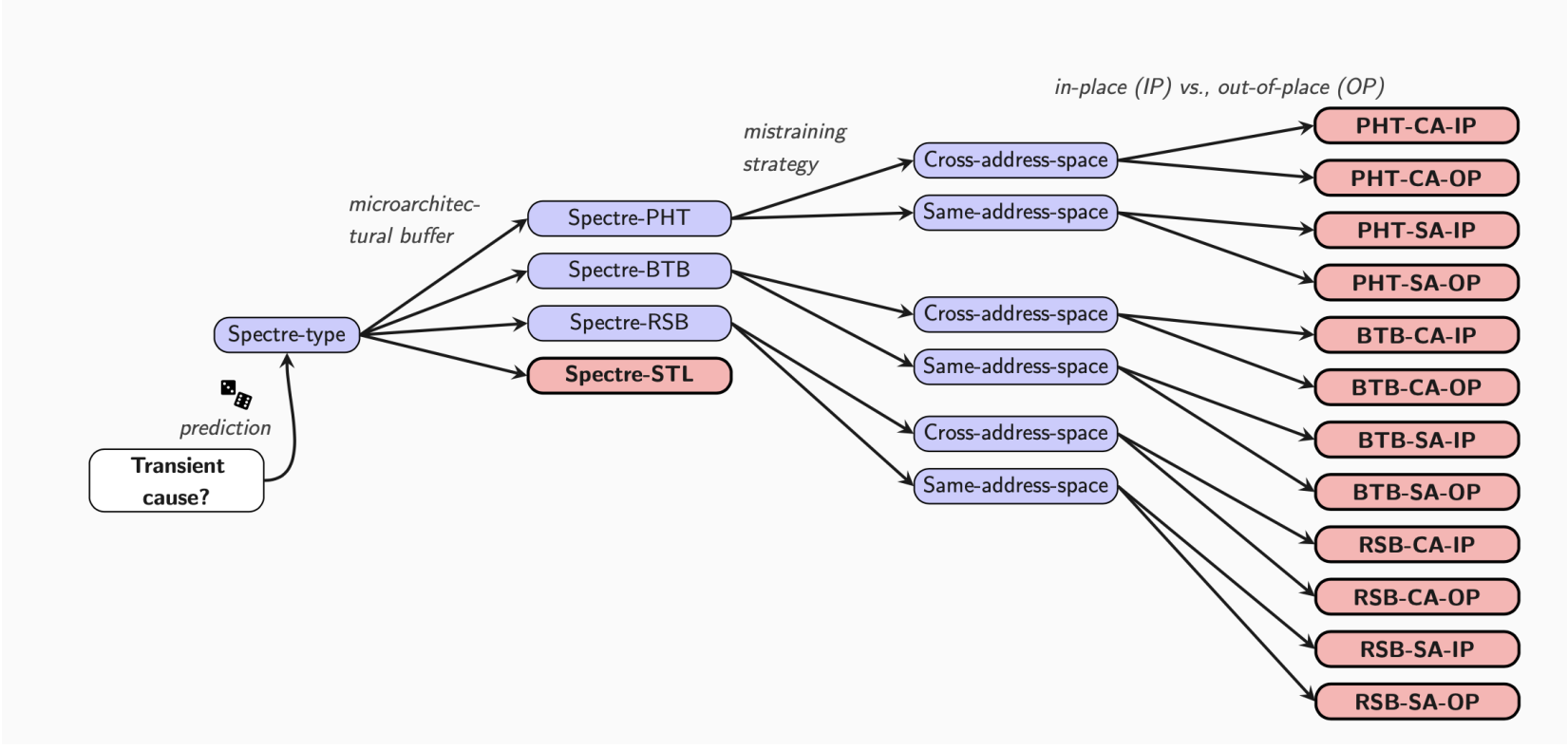
Spectre attacks

- Q: what happens when the processor mispredict?
- A: it will squash the speculatively executed instructions
  - No architecture side-effects
  - However, it will leave micro-architecture side-effects!
- Idea: use micro-architecture side-channel to extract the side-effects (secret)

Spectre v1

```
if (x < array1_size) {
  // array1[x] is the secret value
  // it won't be accessible at architecture level
  secret = array1[x];
  // however, we can leak it through another layer of indirection
  // 4096 is page size, used to denoise cache side-channel
  y = array2[secret * 4096];
```

Spectre family attacks



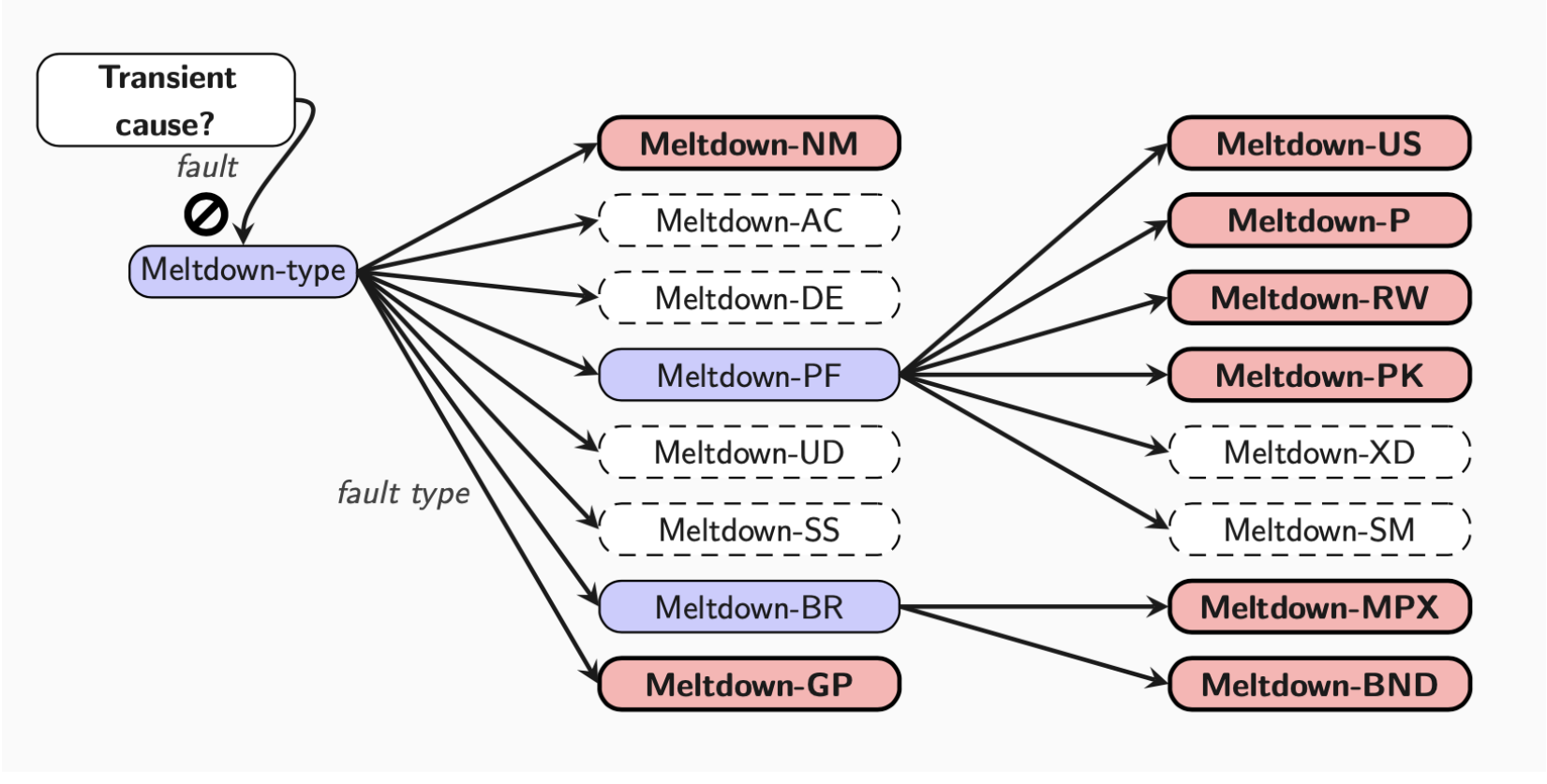
Meltdown attacks

- Q: what if an instruction triggers an exception?
- Most ISA guarantees *precise* exception
  - Out of order instructions will be squashed -> no architecture changes
  - However, it will also leave micro-architecture side-effects!
- Idea: use micro-architecture side-channel to extract the side-effects (secret)

Meltdown-U/S

```
xor rax, rax
retry:
mov al, byte [rcx] ; rcx = kernel address, will cause page fault
shl rax, 0xc ; << 12 == * 4096
jz retry
mov rbx, qword [rbx + rax] ; leak, rbx = probe array
```

Meltdown family attacks



Defenses

Table 1: Spectre-type defenses and what they mitigate.

Attack		InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
Intel	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	●	◐	◐	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
ARM	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
AMD	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	■	◧	◧	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◧	◇	■	◐	◇	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Symbols show if an attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

Performance of defenses

Table 2: Reported performance impacts of countermeasures

Defense \ Impact	Performance Loss	Benchmark
InvisiSpec	22%	SPEC
SafeSpec	3% (improvement)	SPEC2017 on MARSSx86
DAWG	2–12%, 1–15%	PARSEC, GAPBS
RSB Stuffing	no reports	
Retpoline	5–10%	real-world workload servers
Site Isolation	only memory overhead	
SLH	36.4%, 29%	Google microbenchmark suite
YSNB	60%	Phoenix
IBRS	20–30%	two sysbench 1.0.11 benchmarks
STIPB	30– 50%	Rodinia OpenMP, DaCapo
IBPB	no individual reports	
Serialization	62%, 74.8%	Google microbenchmark suite
SSBD/SSBB	2–8%	SYSmark®2014 SE & SPEC integer
KAISER/KPTI	0–2.6%	system call rates
L1TF mitigations	-3–31%	various SPEC