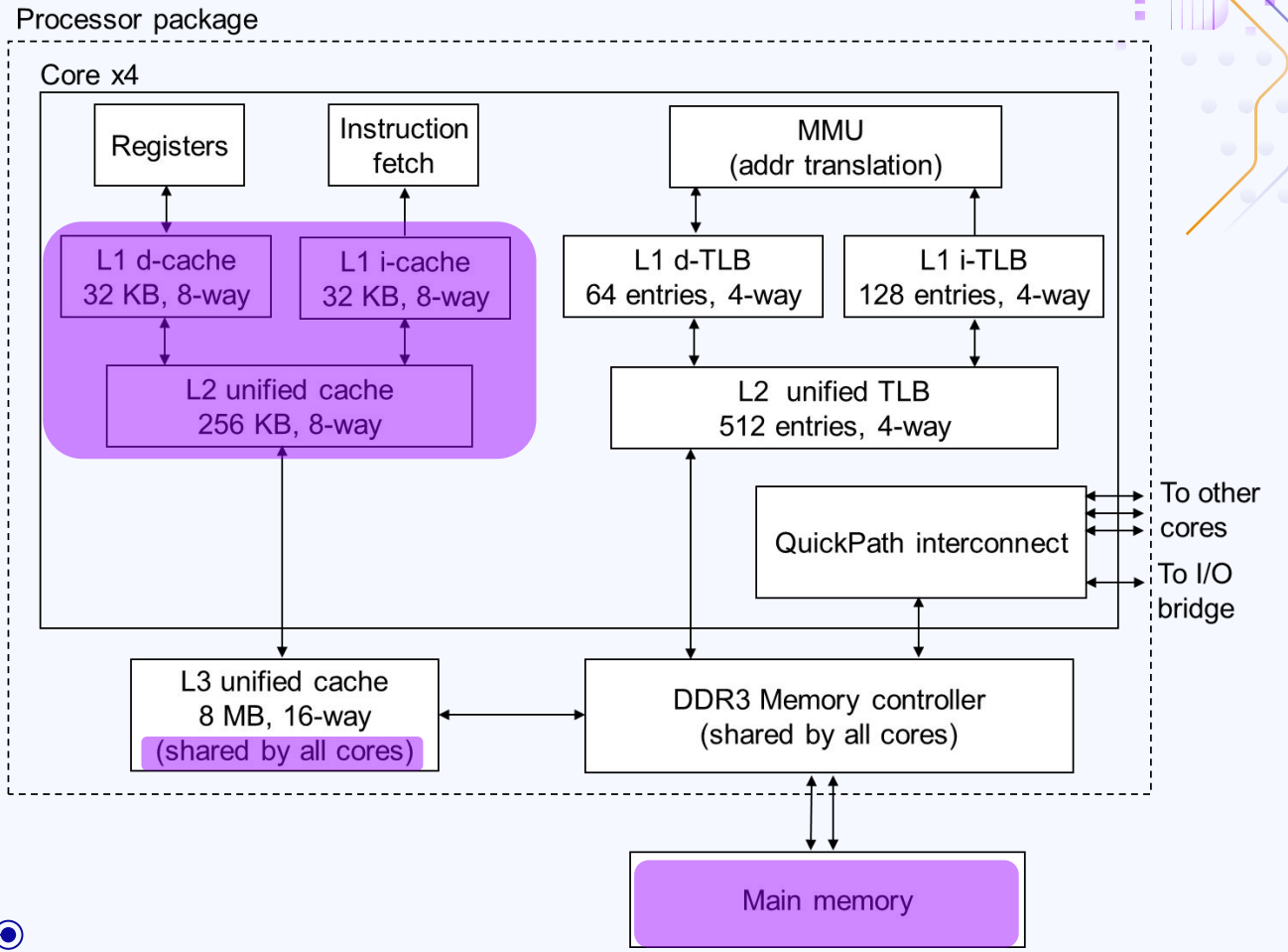
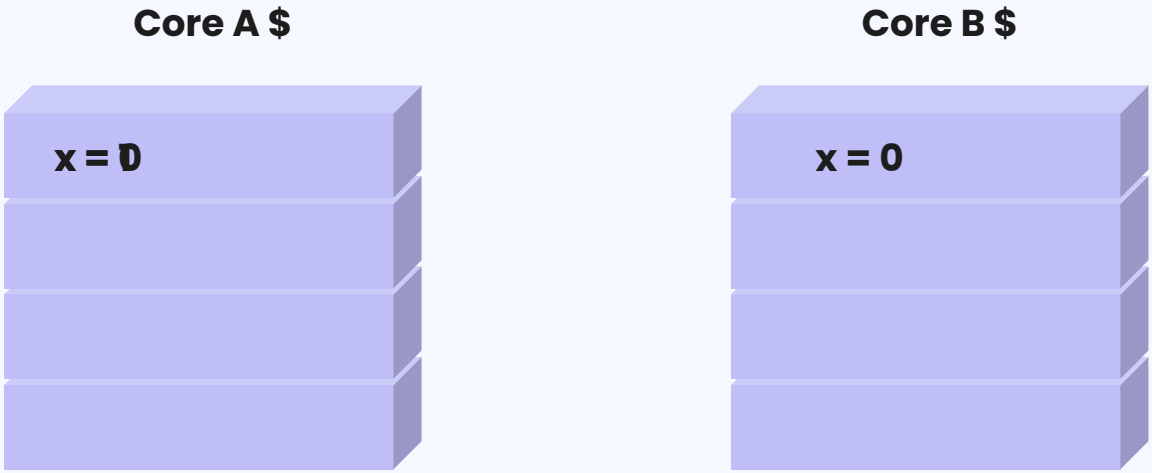
The perils of shared and unshared caches (coherence, side channels)

Intel i7

Source
(Bryant & O'Hallaron)



Cache coherence problem



Definitions

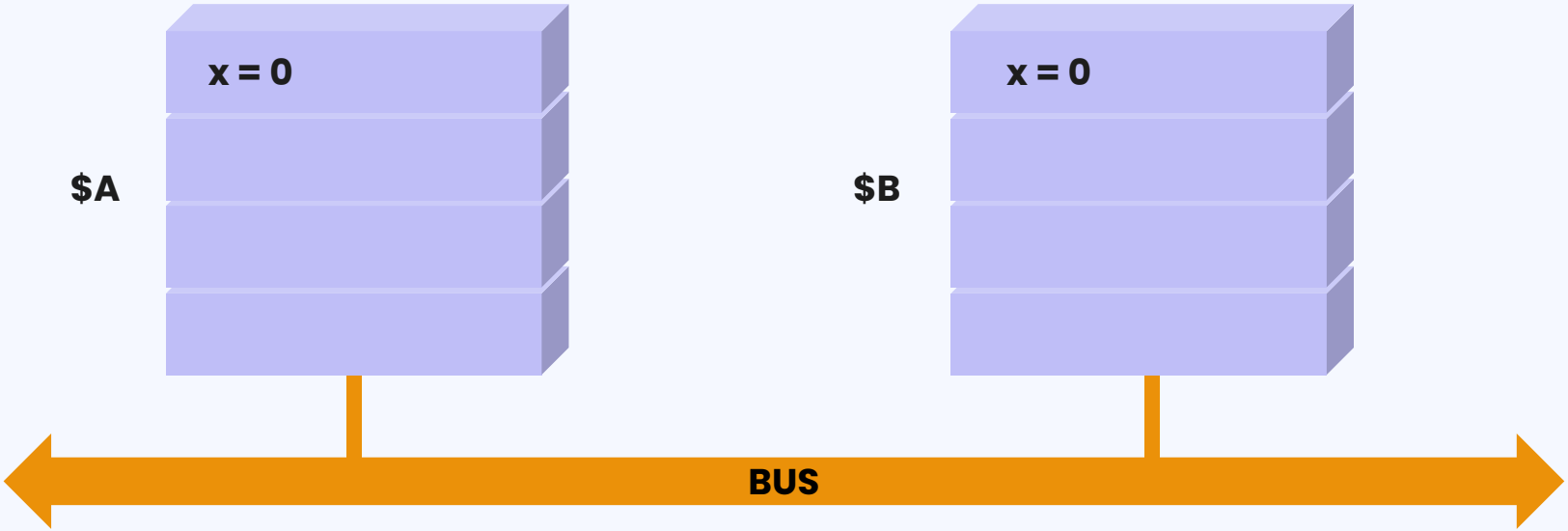
Intuitively: want any read of an item to return most recently written value to item

Coherence – what values can be returned by a read?

1. On a uniprocessor: reads after writes return written value
2. On a multiprocessor: reads by **B** after writes by **A** return written value **when given sufficient time**
3. Two writes to same location by one processor are seen in the same order by all processors
    a. **A** writes 1 and then 2 to a memory location
    b. We don't want **B** to decide the value is 1 and **C** to decide the value is 2

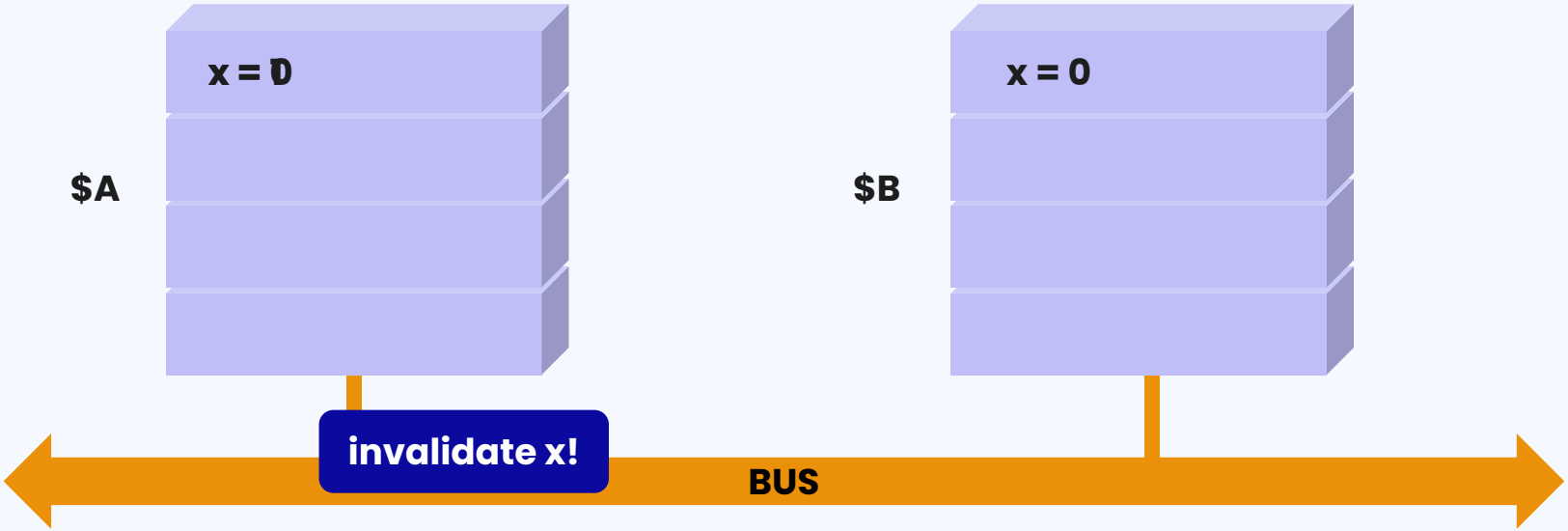
Snooping 🔍

1. A reads x
2. B reads x



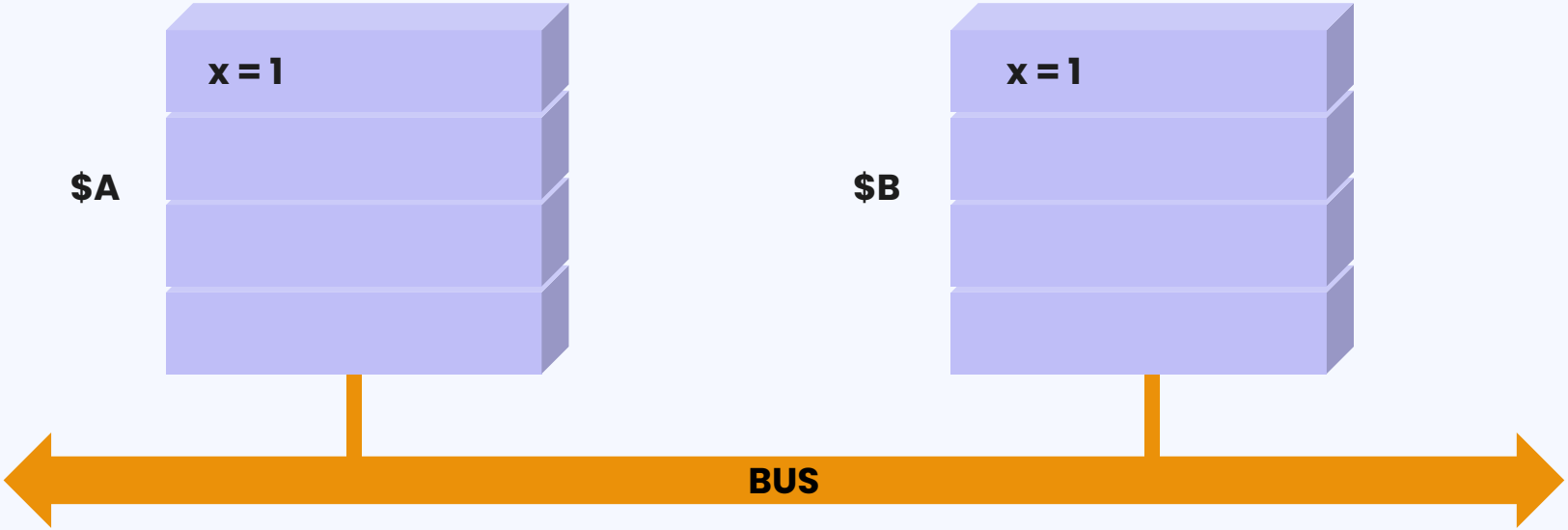
3. A writes x

Snooping🔍



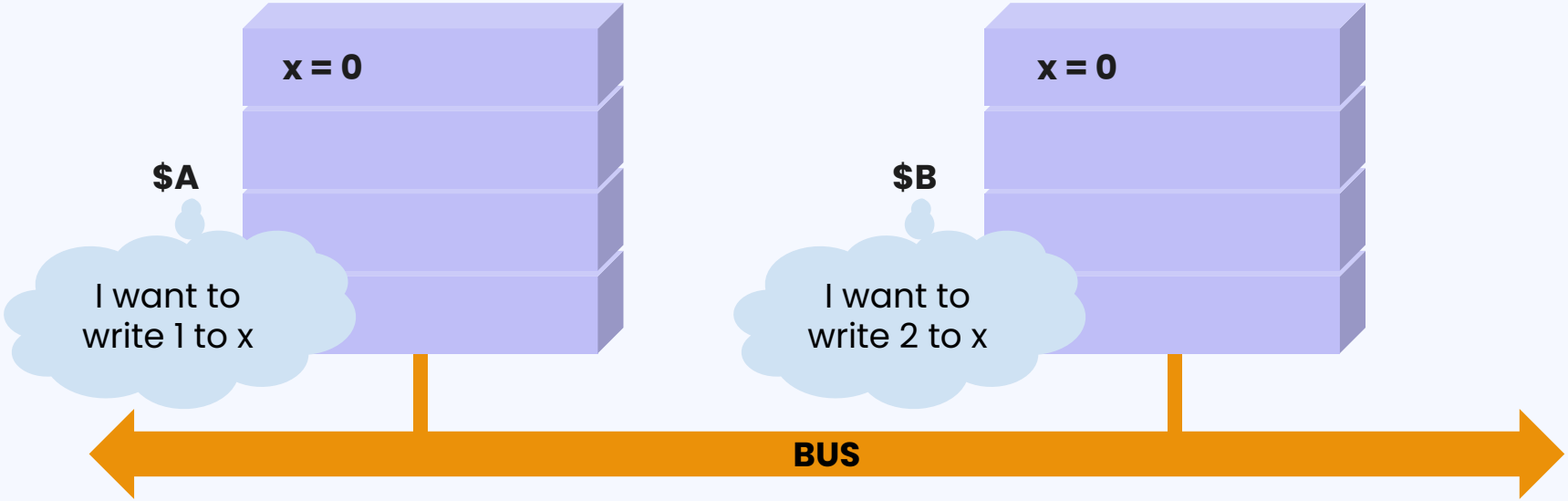
4. B reads x

Snooping🔍



Cache needs to keep state for each block based on bus messages (common protocol: MESI (upgrade of MSI)

What should happen here?



What effects does block size have on coherence protocols?

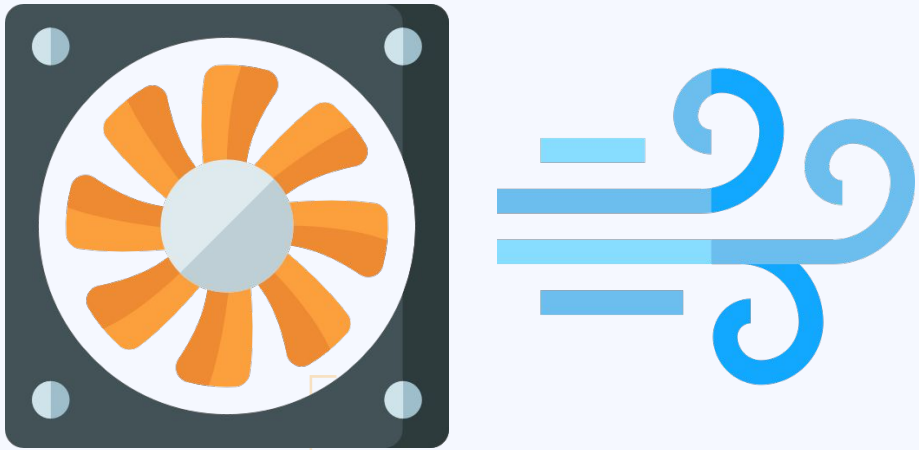
(Pivoting away from coherence)
How can we *measurably* determine whether or not a piece of data is already present in a cache?

Security and side channels

Security: protection against threats from malicious actor

Obviously a **large** field, this is just a taste

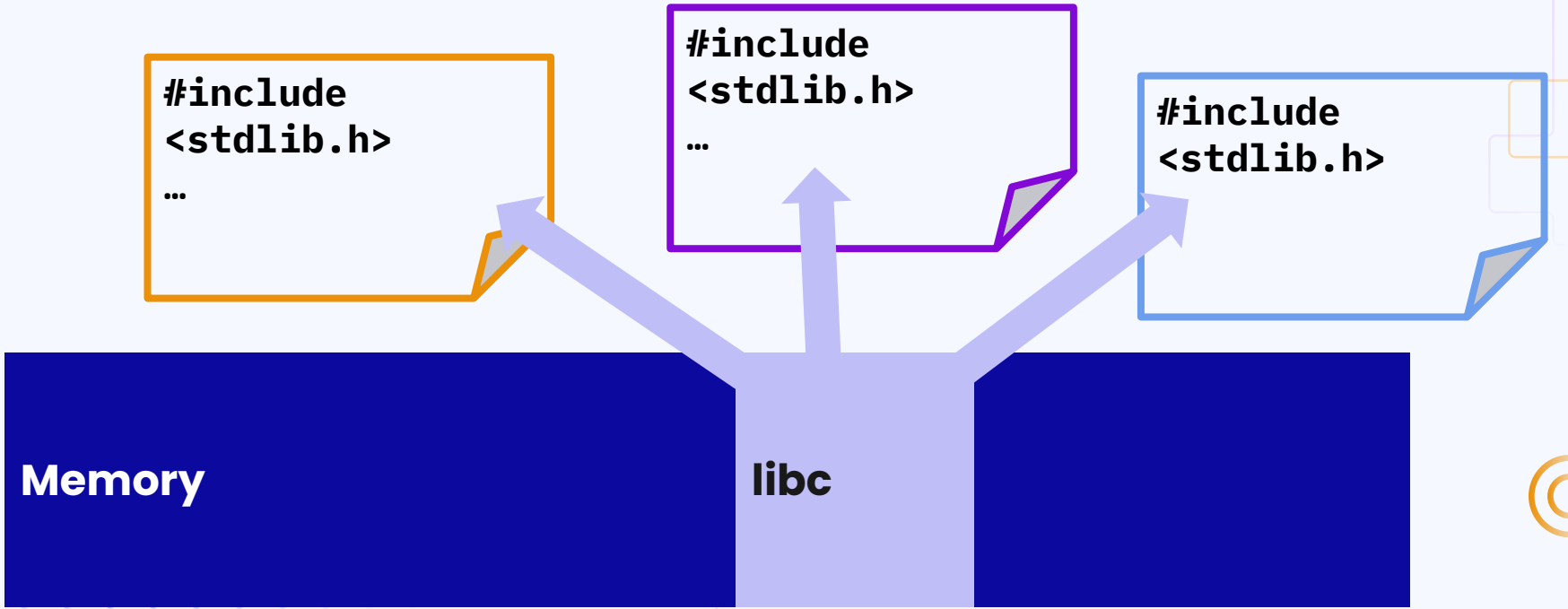
Side channels: Incidental information leakage inferred from observing normal execution



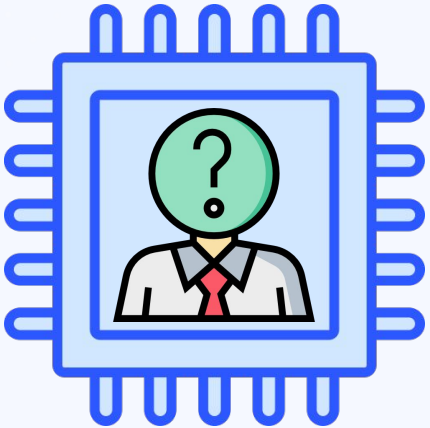
Slides adapted from Sam Thomas

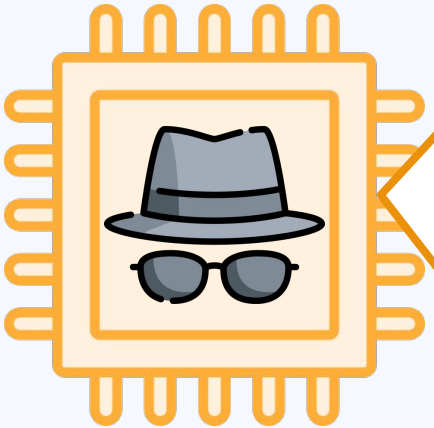
Shared cache side channels

What precisely can CPU B learn about CPU A when data is shared?



Flush & Reload attack



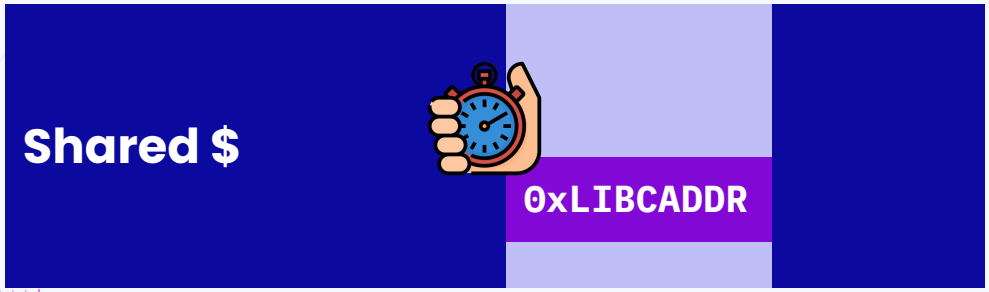


```
// flush the block
asm(clflush 0xLIBCADDR);

// wait some time
t1 = time.now();
while(time.now() - t1 <
100ns);

// access block
t2 = time.now();
x = * 0xLIBCADDR;
accessTime = time.now()
- t2;

// if slow: unused
// if fast: used!
```



Flush & Reload in the wild

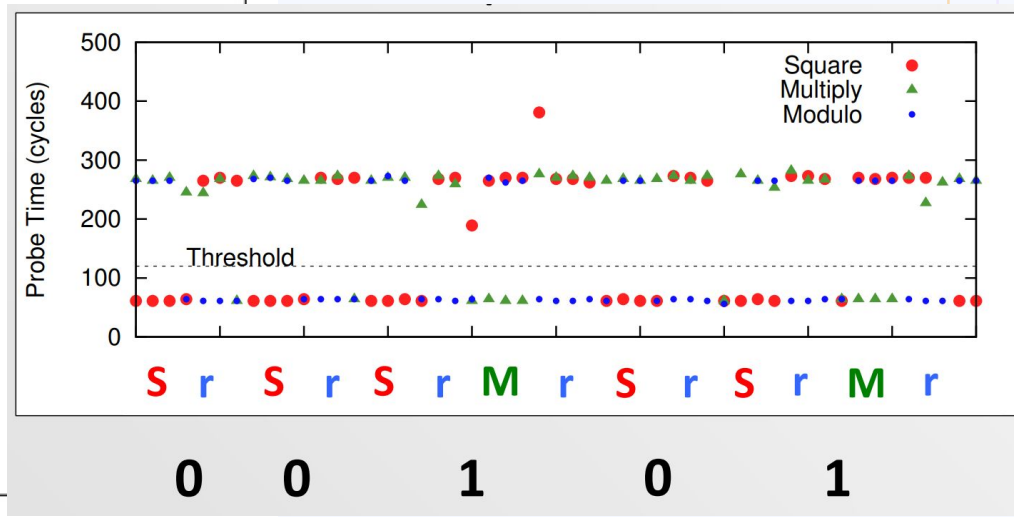
So what? An attacker knows I used libc...

```
1  function exponent(b, e, m)
2  begin
3    x ← 1
4    for i ← |e| − 1 downto 0 do
5      x ← x^2
6      x ← x mod m
7      if (e_i = 1) then
8        x ← xb
9        x ← x mod m
10     endif
11   done
12   return x
13 end
```

Only multiply when bit is 1

Figure 2: Exponentiation by Square-and-Multiply

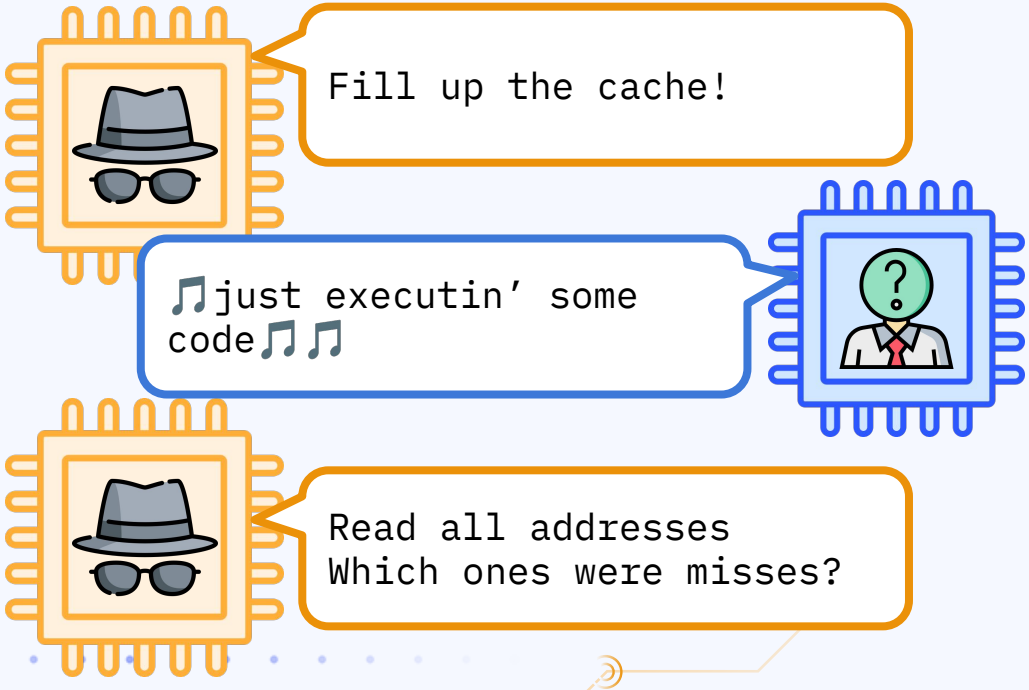
Yarom, Yuval, and Katrina Falkner. "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack." 23rd USENIX security symposium (USENIX security 14). 2014. *link*

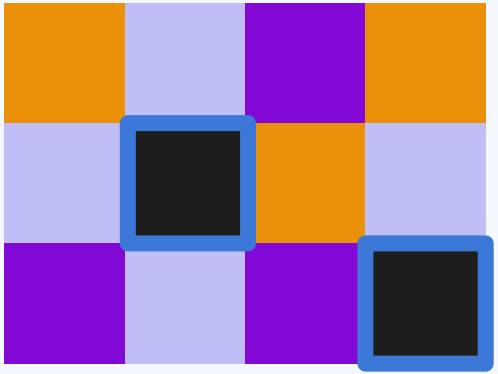


What does flush & reload depend on?

Prime & Probe

Doesn’t require precise timing or access to victim’s memory; measures cache contention (less granular)





Caveats

It takes a *lot* to instrument a side-channel attack

Often can't learn *everything*, but narrow down search space

What can be done to guard against attacks?

- Oblivious RAM (largely theoretical)
- Cache partitioning/slicing
- Use an abacus