Security for Architects (Part 1)

Security/Privacy

Talking About Security…

- The goal is not to scare you!
- “Architecture in the wild”
- The cost of optimizations

Chat with your neighbor!

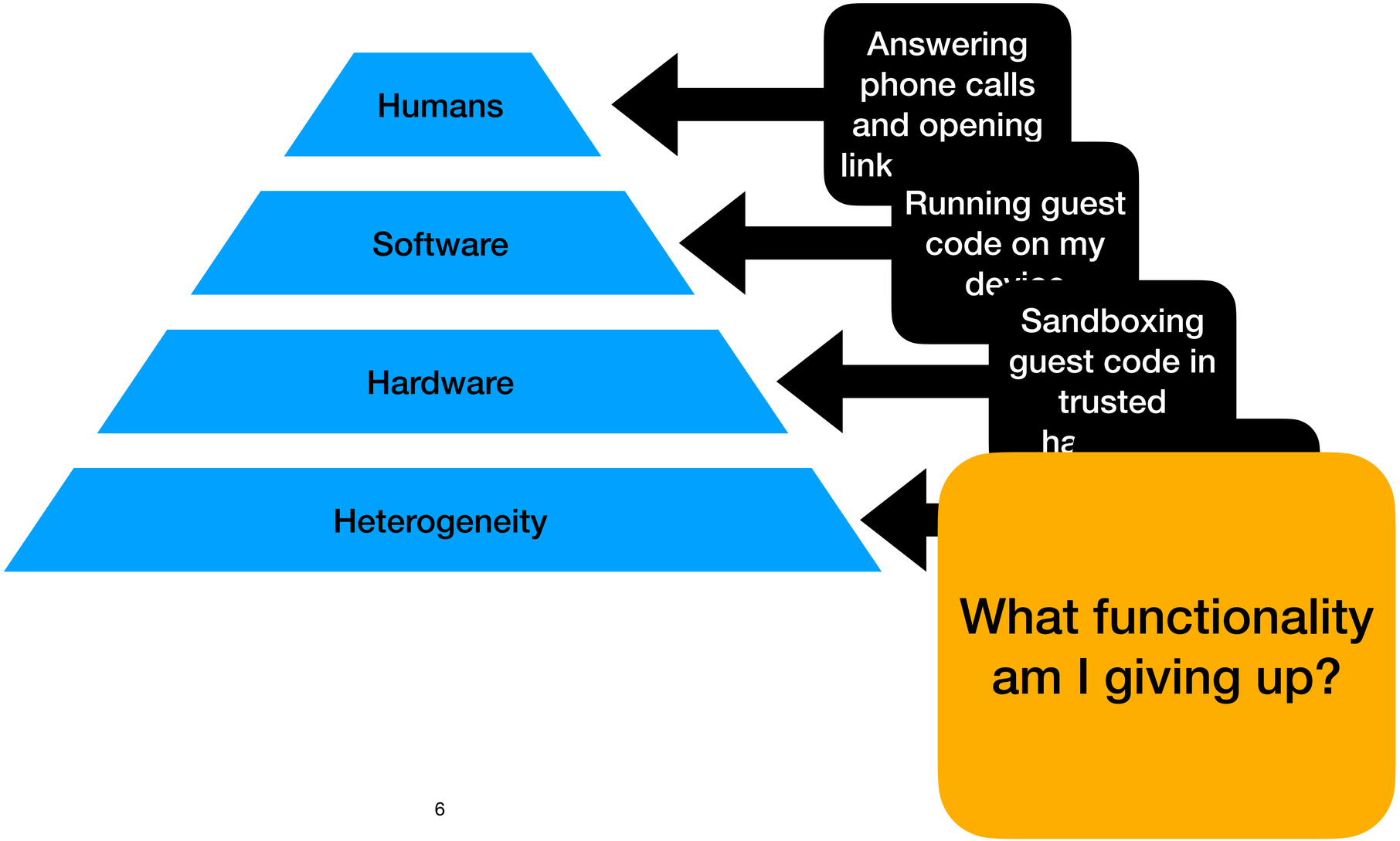
- How security-conscious are you?
- What steps do you take to ensure you privacy?
- What security expectations do you have of your computing devices?
- Should researchers work to uncover security vulnerabilities?

Outline

- Introduce the “security stack”
- Fundamental architecture security vulnerabilities
- Some mitigating architectures

Security Stack

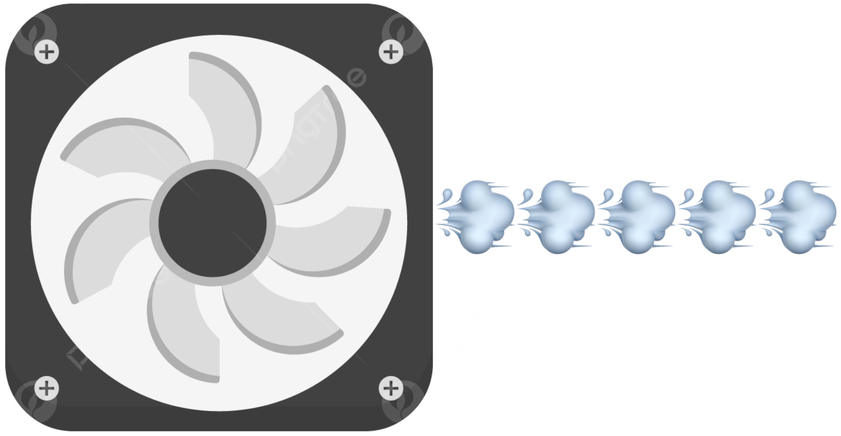
What do I trust?



Security Stack

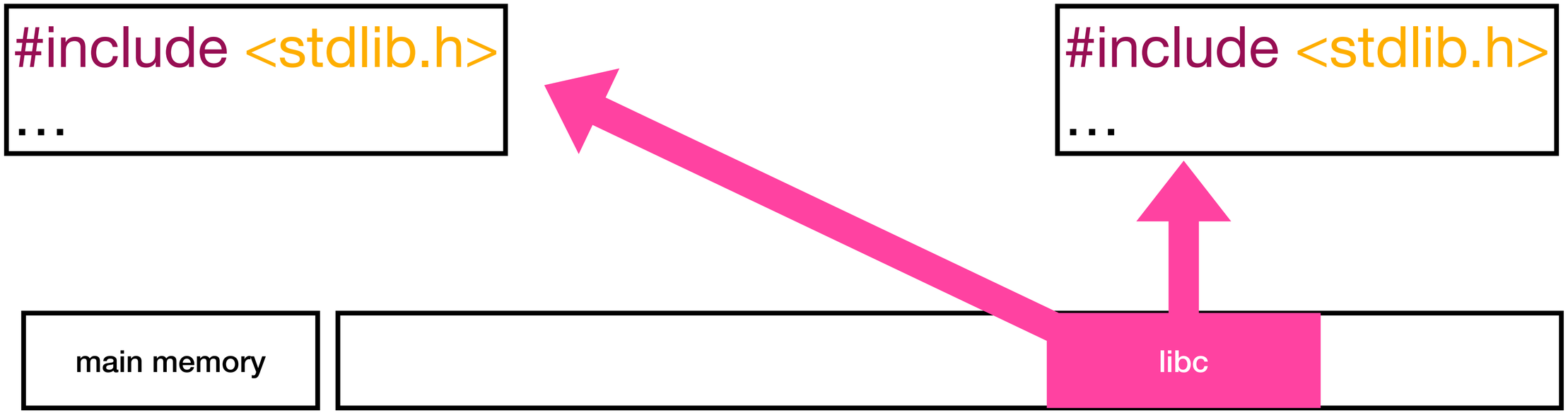
- How does hardware leak information?
- Side channels: Incidental information leakage inferred from observing normal execution

```
for (;;) {
   // super intense computation!
}
```

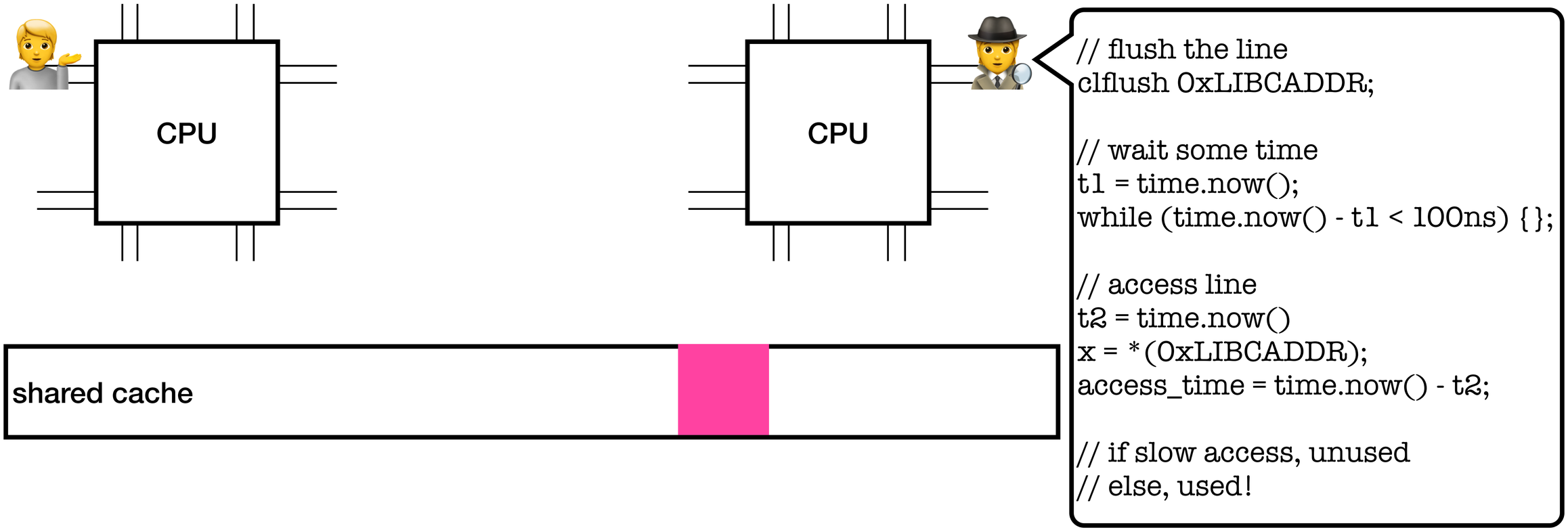


Architecture Security

- How precise can information be leaked from architecture behaviors?
- It depends…



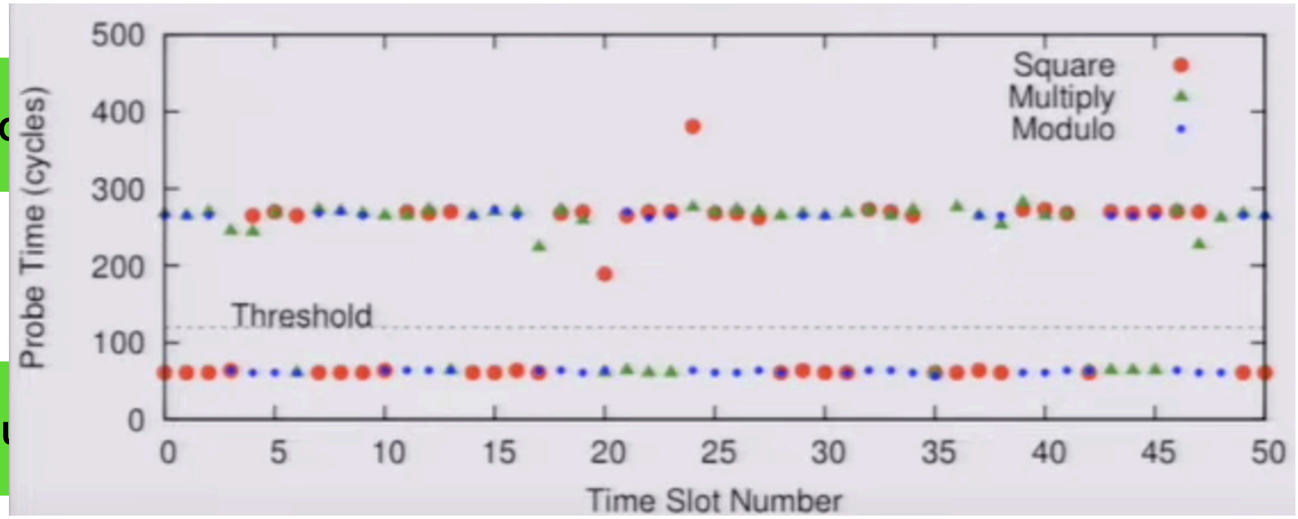
Flush + Reload Attack



Flush + Reload Attack

- So what? An attacker knows that I used libc…
- An example in the wild from RSA encryption/decryption $b^e \bmod n$ where e is secret!

```
x ⟵1
for i ⟵|e|-1 downto 0 do
    x ⟵x^2 mod n
    if (e_i = 1) then
        x = xb mod n
    endif
done
return x
```



courtesy: USENIX Security ‘14

Flush + Reload

- Shared libraries/software can be accessed by any processor
- Open-source software is out there to be analyzed
- Data dependent behavior can dictate the cache state/access times
- Spies can infer precise values from hits and misses to shared data

Flush + Reload: Chat with your neighbors!

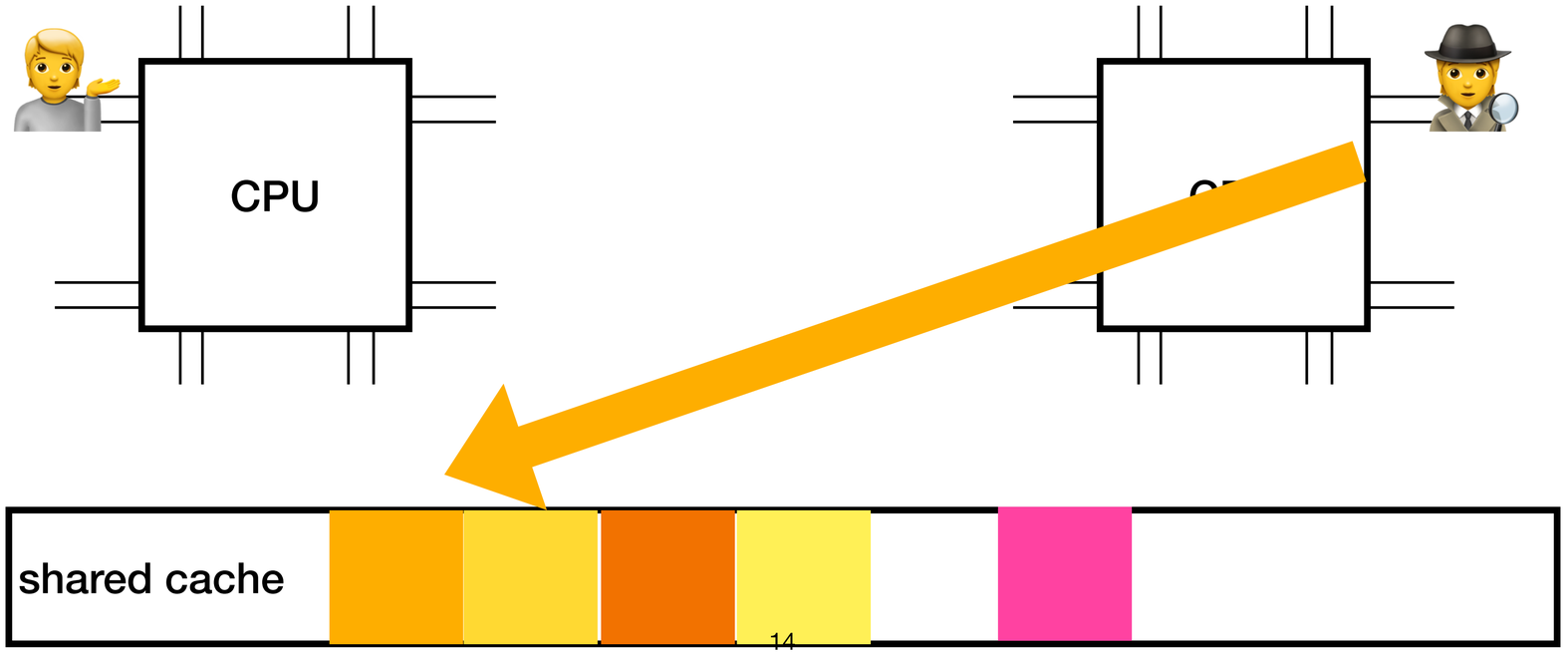
- Is this vulnerability dependent on the cache?
- If we care about privacy, are we totally screwed?

Flush + Reload

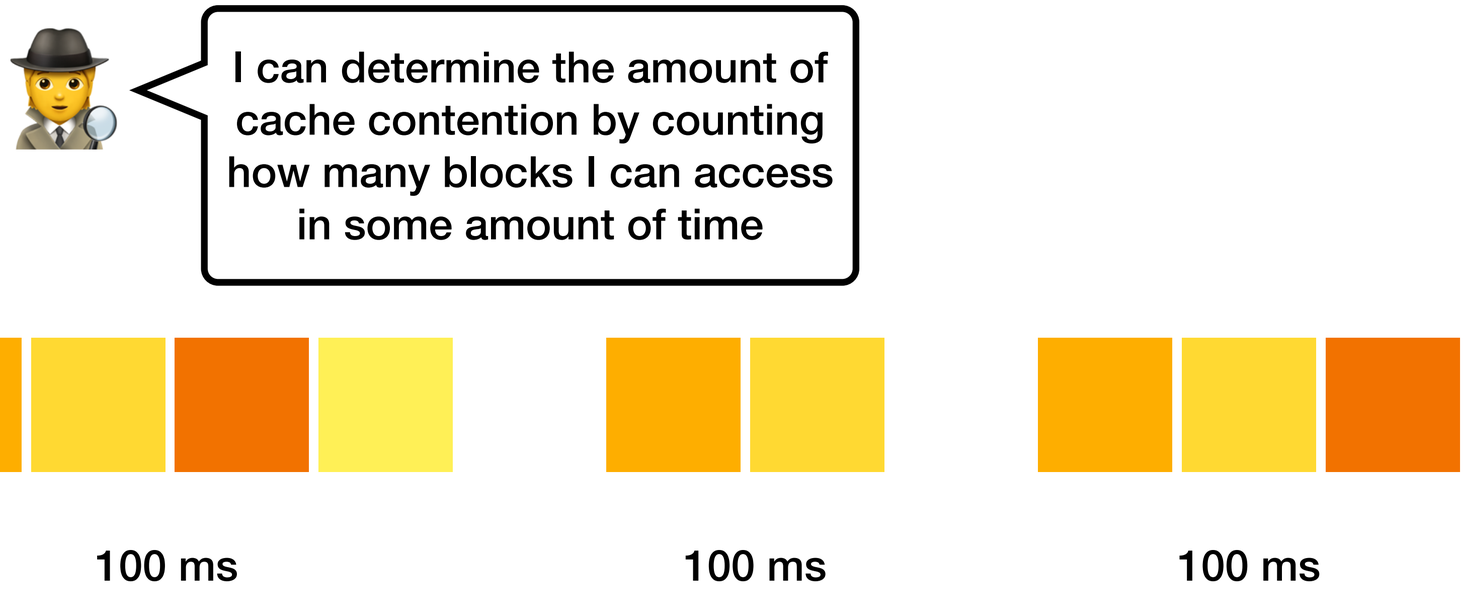
- Dependent on `clflush` operation (10 points to RISC-V!)
- Dependent on very precise timers (nano seconds granularity!)
- What if the ISA/hardware didn’t allow for such precise timing?

Prime + Probe

- Not so fast…
- We may not be able to time individual accesses, but we can measure the impact of misses due to *cache contention*

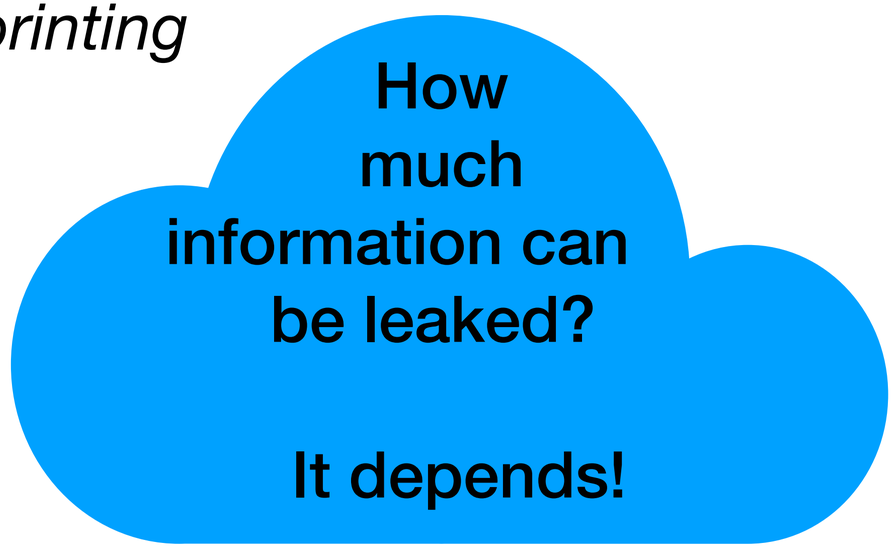


Prime + Probe


I can determine the amount of cache contention by counting how many blocks I can access in some amount of time
100 ms
100 ms
100 ms

Prime + Probe

- Wait, but this is information is much less precise… how would a spy use that information?
- These attacks are generally used for leaking bigger picture information like *website fingerprinting*



Architecture Mitigations

- How can we defend against these attacks? We need to understand what enables these attacks!
- Shared libraries can be an avenue for leaking information via Flush + Reload attacks
- Certain instructions make these attacks possible
- Shared hardware allows attackers to flush/probe your data

Architecture Mitigations

- Oblivious RAM: remove correlation between behavior and impact on architectural state
  - i.e., scan all of memory for every operation
- Trusted Execution: isolate the sensitive process in a specialized region of hardware
  - i.e., Intel SGX, ARM TrustZone, RISC-V Keystone, etc.
- Cache partitioning: each processor gets assigned to a region in each shared cache and can only access its region

Oblivious RAM

- Hide information by removing correlation between observations and inference
- Often done at the algorithmic level (not really an architecture)
- Very theoretical

Trusted Execution

- Often very architecture dependent on what the specs are and what the guarantees are
- Separate caches for isolation, separate register state, etc…
- Often very slow!
- Subject to any underlying architecture vulnerabilities!

Cache Partitioning

- Provides isolation in the cache, which eliminates shared access
- Leads to inefficient use of the cache… fragmentation!
- Research into scalable cache partitions, but even these leak data!

Still to come!

- Spectre/Meltdown attacks
- Rowhammer attacks
- HertzBleed
- Bus Leakage

Concluding Thoughts

- You don’t have to be scared about what is in your cache state, but you should be aware of what’s in your cache state
- These vulnerabilities are born out of powerful, smart optimizations! But performance benefits aren’t always for free…
- Working on mitigating these vulnerabilities well is an active ongoing effort!

- So should I just give up and use an abacus?