Module 8 – MicroArch Side-Channel Attack

(ECE 209) Secure and Advanced Computer Architecture

Nader Sehatbakhsh
Department of Electrical and Computer Engineering
University of California, Los Angeles

Why security is critical in the next decade?

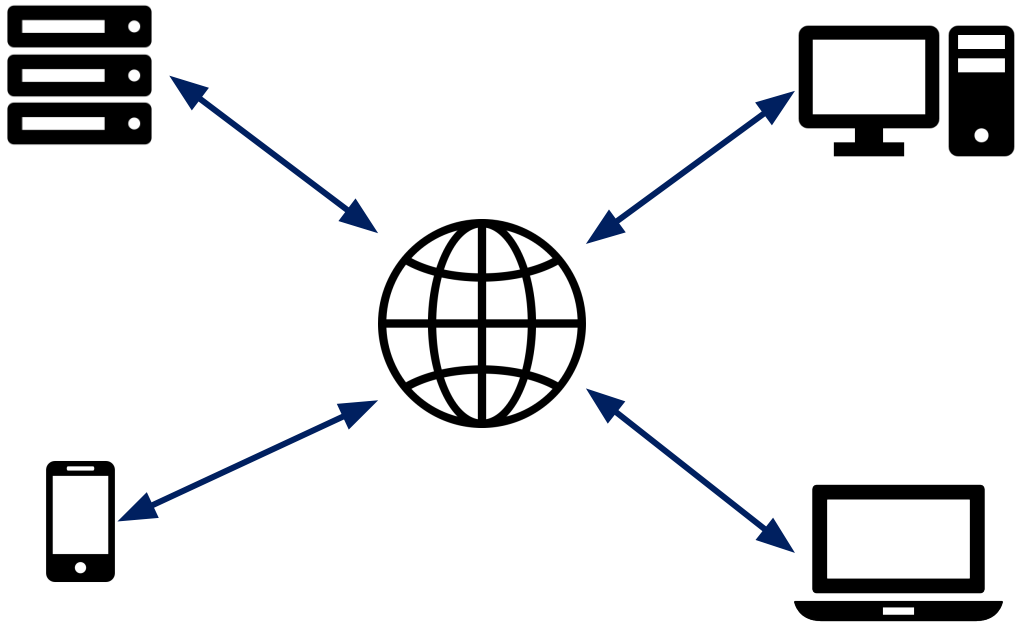
1. More Devices (CPS and industry 4.0)

2. More Security Critical Applications

★ ***More Difficult to Secure Computing Devices (computing paradigm).***

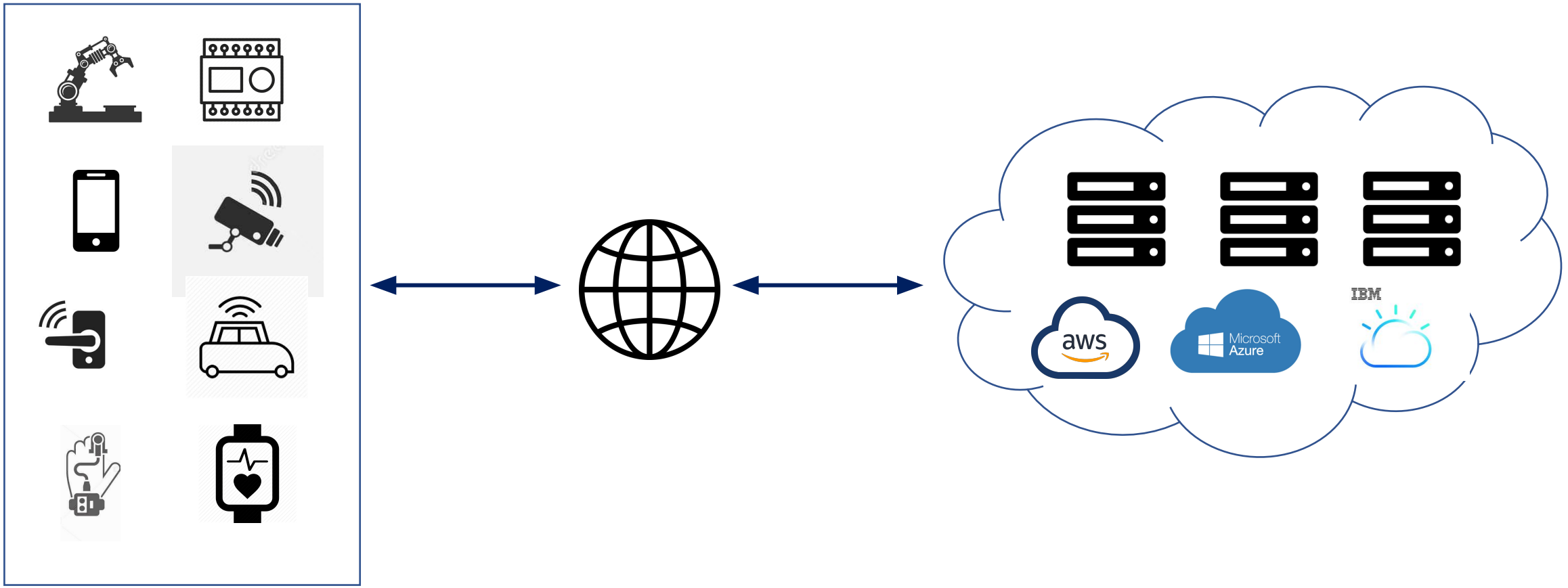
Shift in Computing Paradigm!

Past: Local and/or Secured



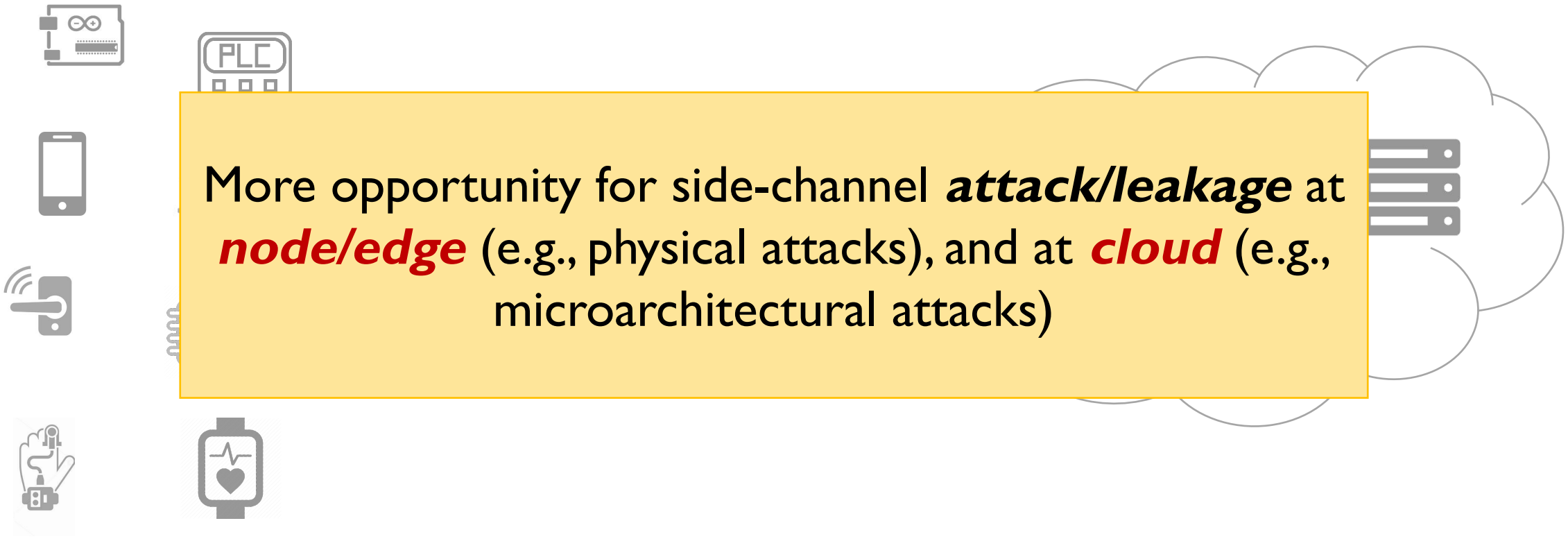
Shift in Computing Paradigm!

Present: Remote and Hostile



Shift in Computing Paradigm!

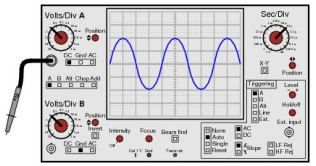
Present: Remote and Hostile



More opportunity for side-channel ***attack/leakage*** at ***node/edge*** (e.g., physical attacks), and at ***cloud*** (e.g., microarchitectural attacks)

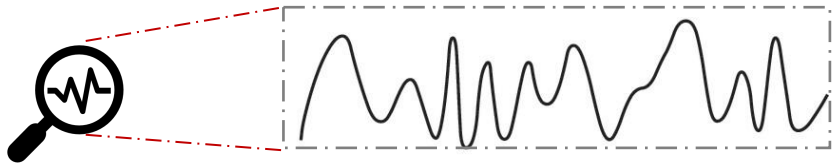
So, what is “side-channels”?

Side-Channels

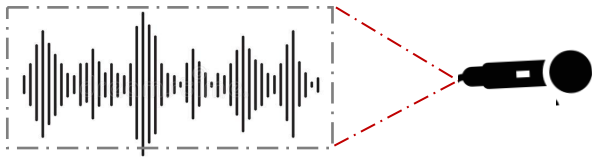


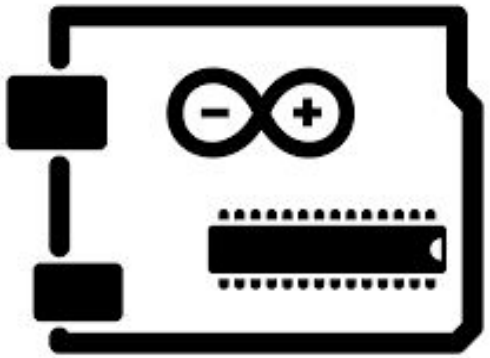
power

electromagnetic

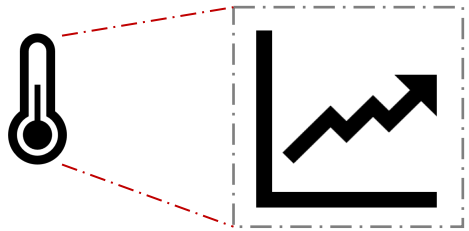


sound/acoustic



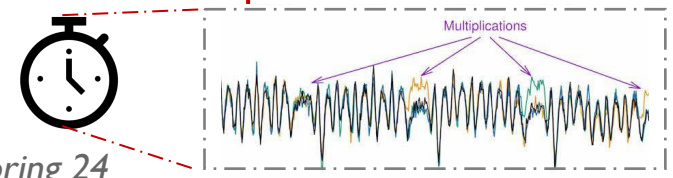


temperature



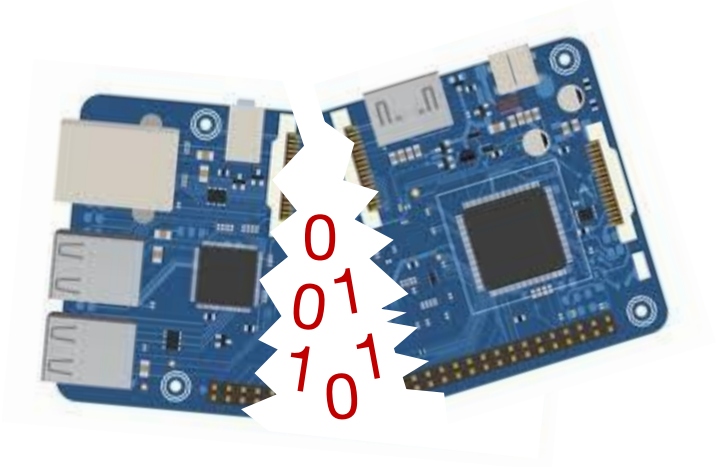
...

time

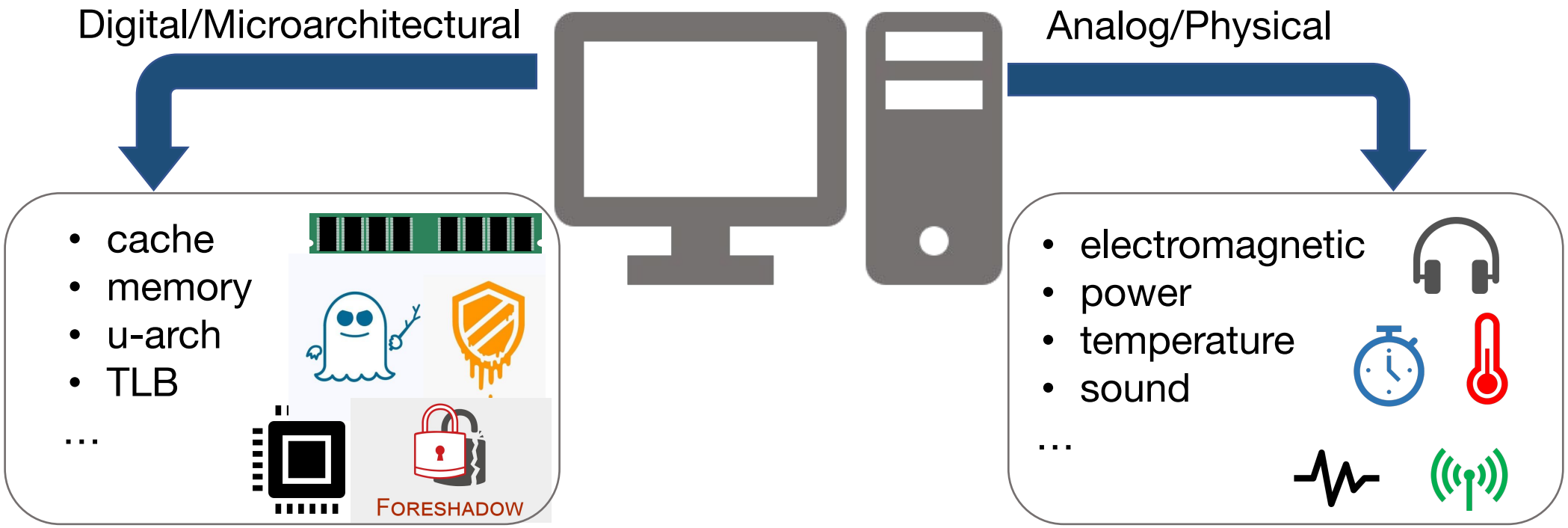


Side-Channels can ***leak*** secrets!

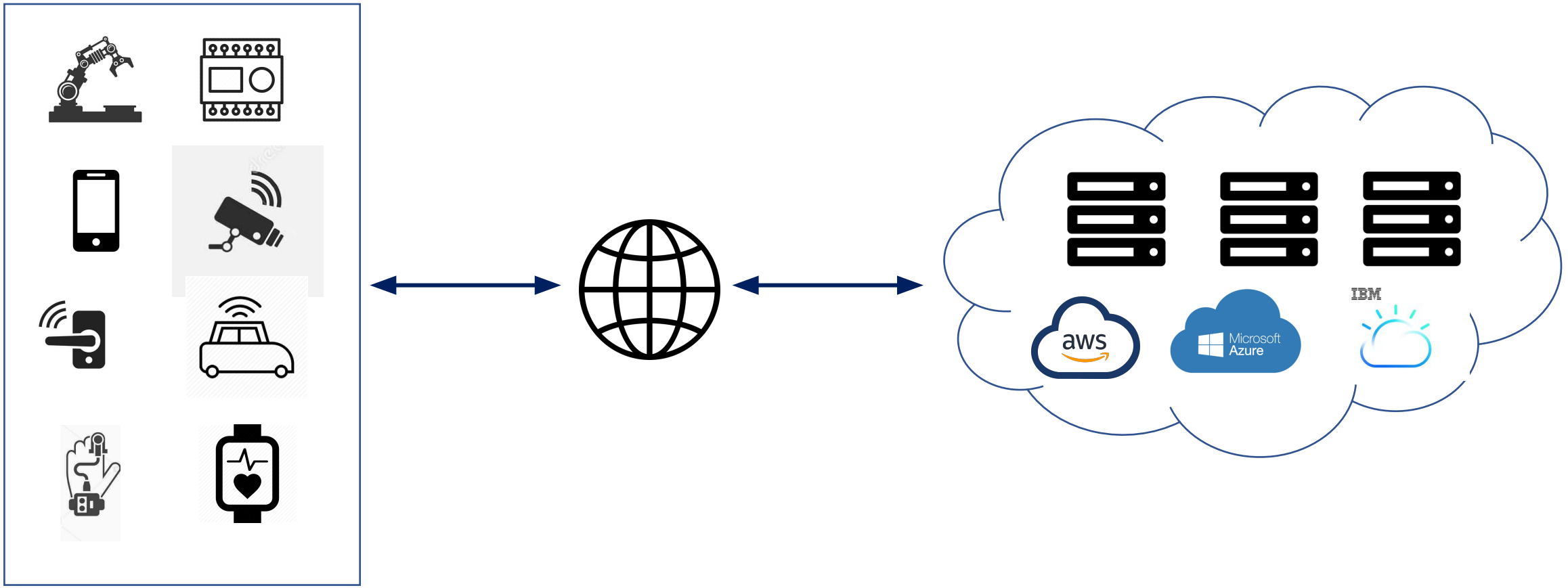
An adversary can leverage the existing ***correlation*** between the side-channel signals and critical information in the application to discover the secrets.



Types of Side-Channels



Main issues



How do we enforce security in computers today?

How do we enforce security in computers today?

1- "Principle of Least Privilege"

Users/Process should only have access to the data and resources needed to perform routine, authorized tasks.

How do we enforce security in computers today?

1- "Principle of Least Privilege"

Users/Process should only have access to the data and resources needed to perform routine, authorized tasks.

➔ *This leads to "privilege separation". Typically, "user" and "root" level access.*

How do we enforce security in computers today?

2- Isolation

A process cannot access (read or write) the memory content of any other process.

How do we enforce security in computers today?

3- Trusted Computing Base (TCB)

Trust something (e.g., hardware), build everything on top/around that.

- *Only need to verify the TCB.*
- *Keep it simple and small so it can be easily(!) verified.*

Does it work?

Digital Side-Channels

- Digital side-channels can break the isolation assumption.

★ *How?*

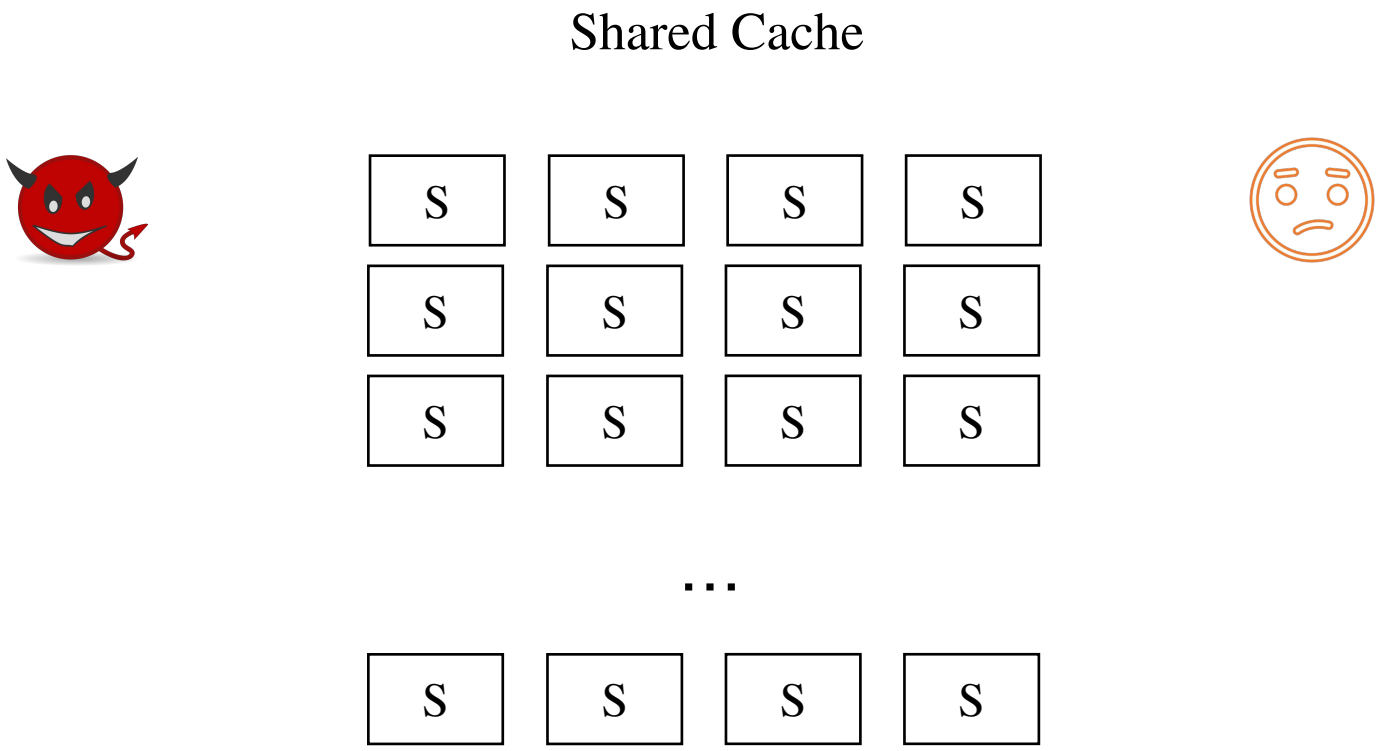
Digital Side-Channels

- Digital side-channels can break the isolation assumption.

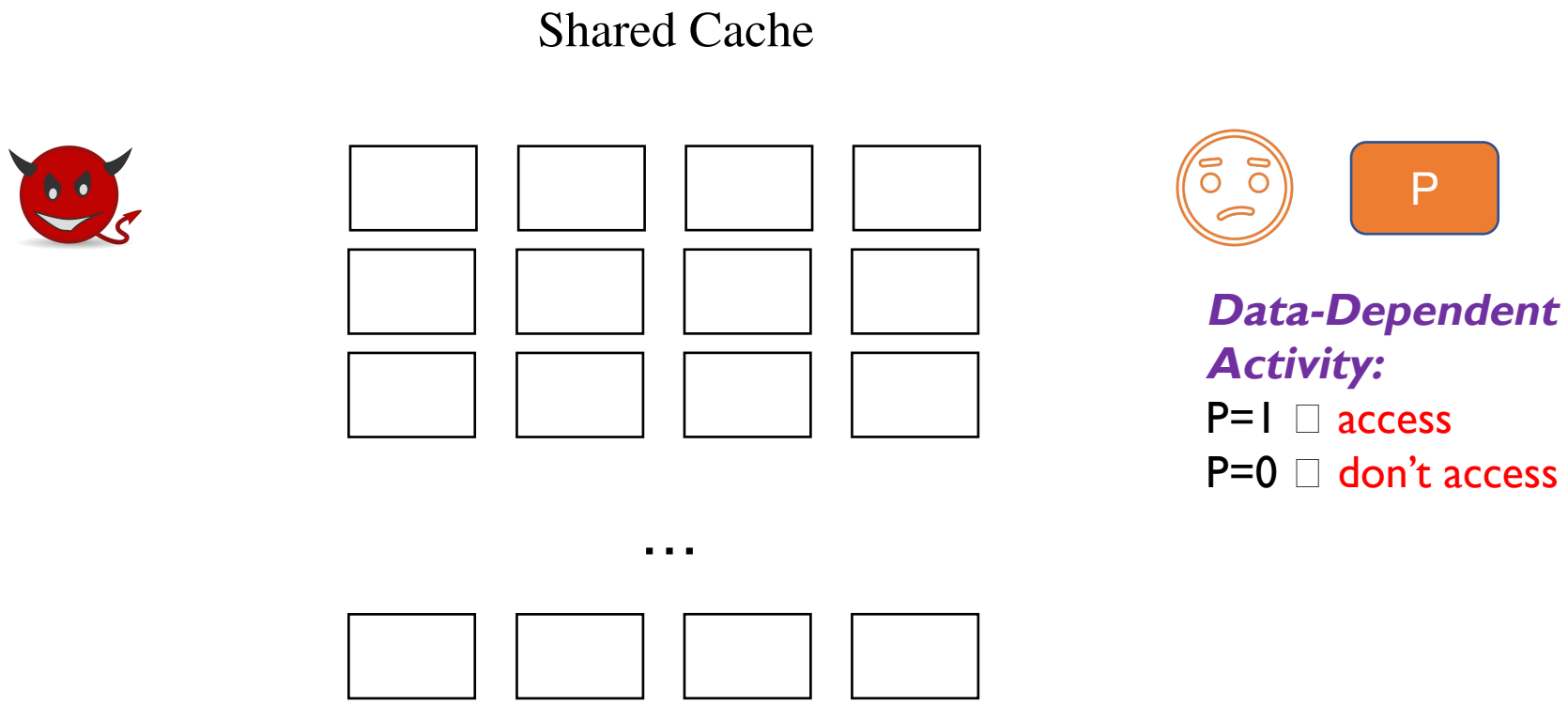
★ *How?*

➔ *Through different **contentions** in hardware/microarchitecture units (e.g., caches).*

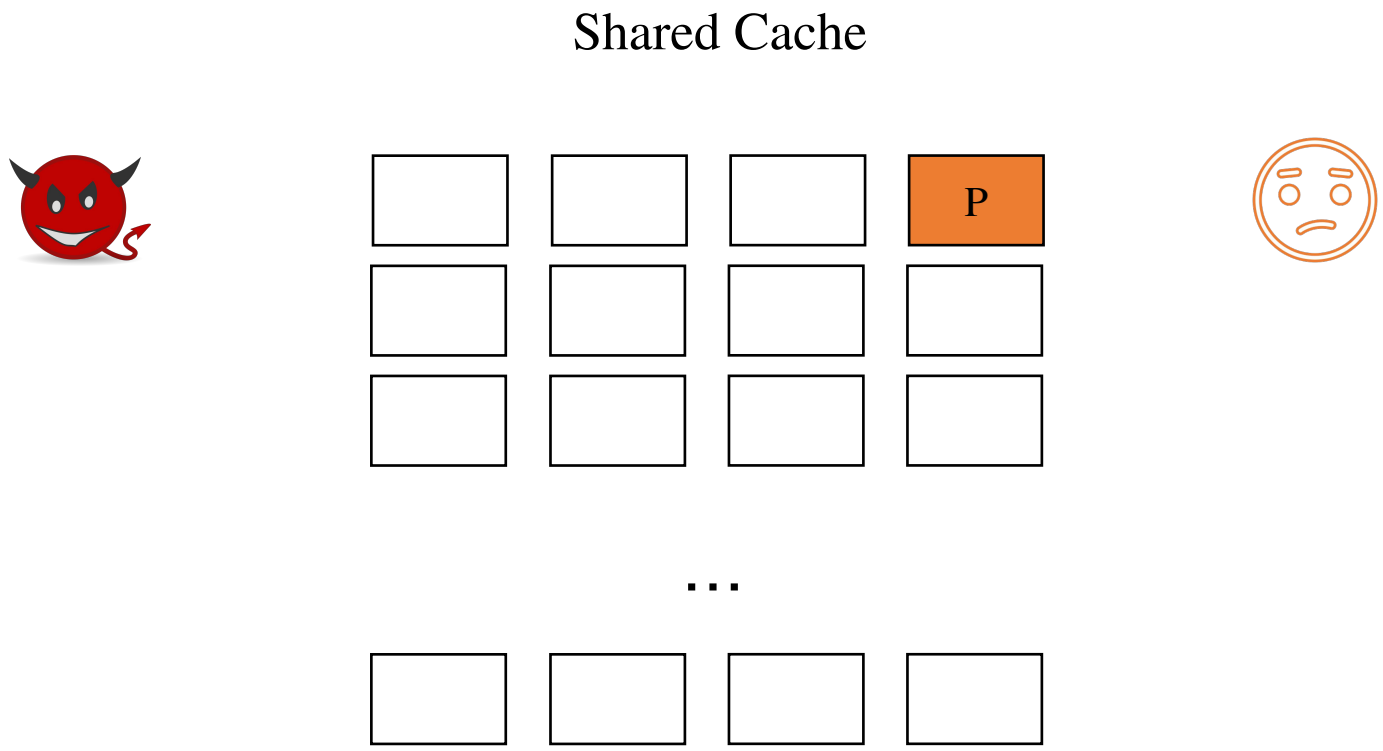
How cache side-channel works?



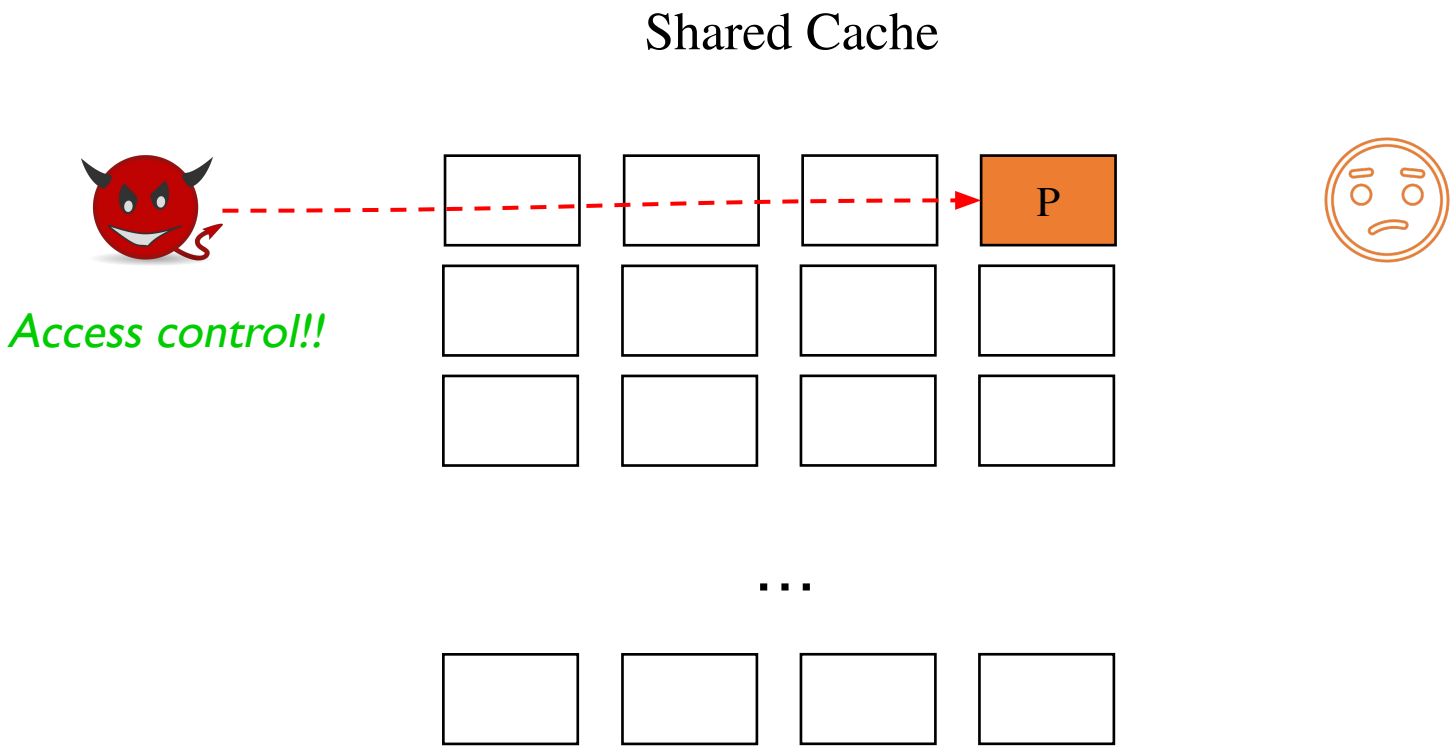
How cache side-channel works?



How cache side-channel works?



How cache side-channel works?



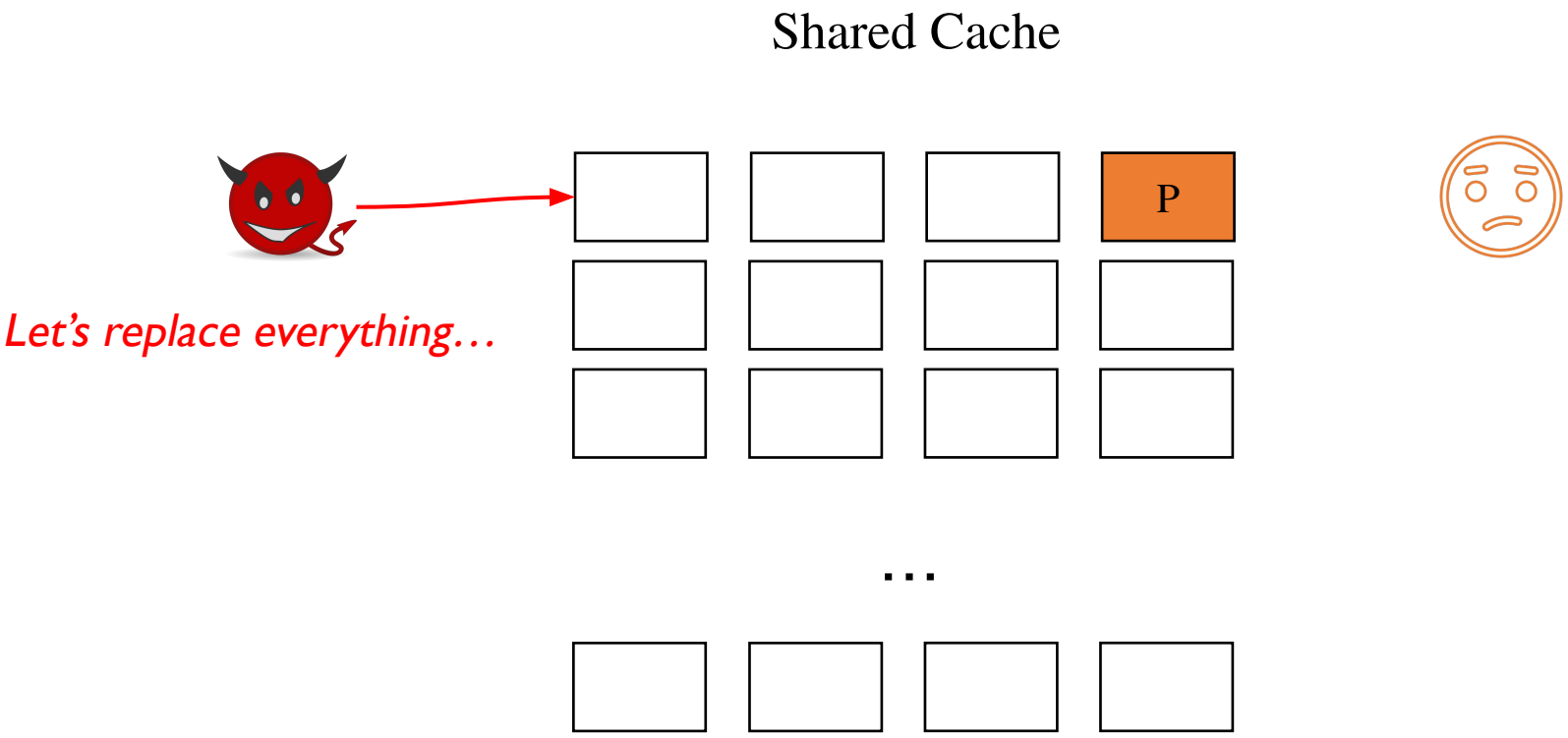
How do we enforce security in computers today?

2- Isolation

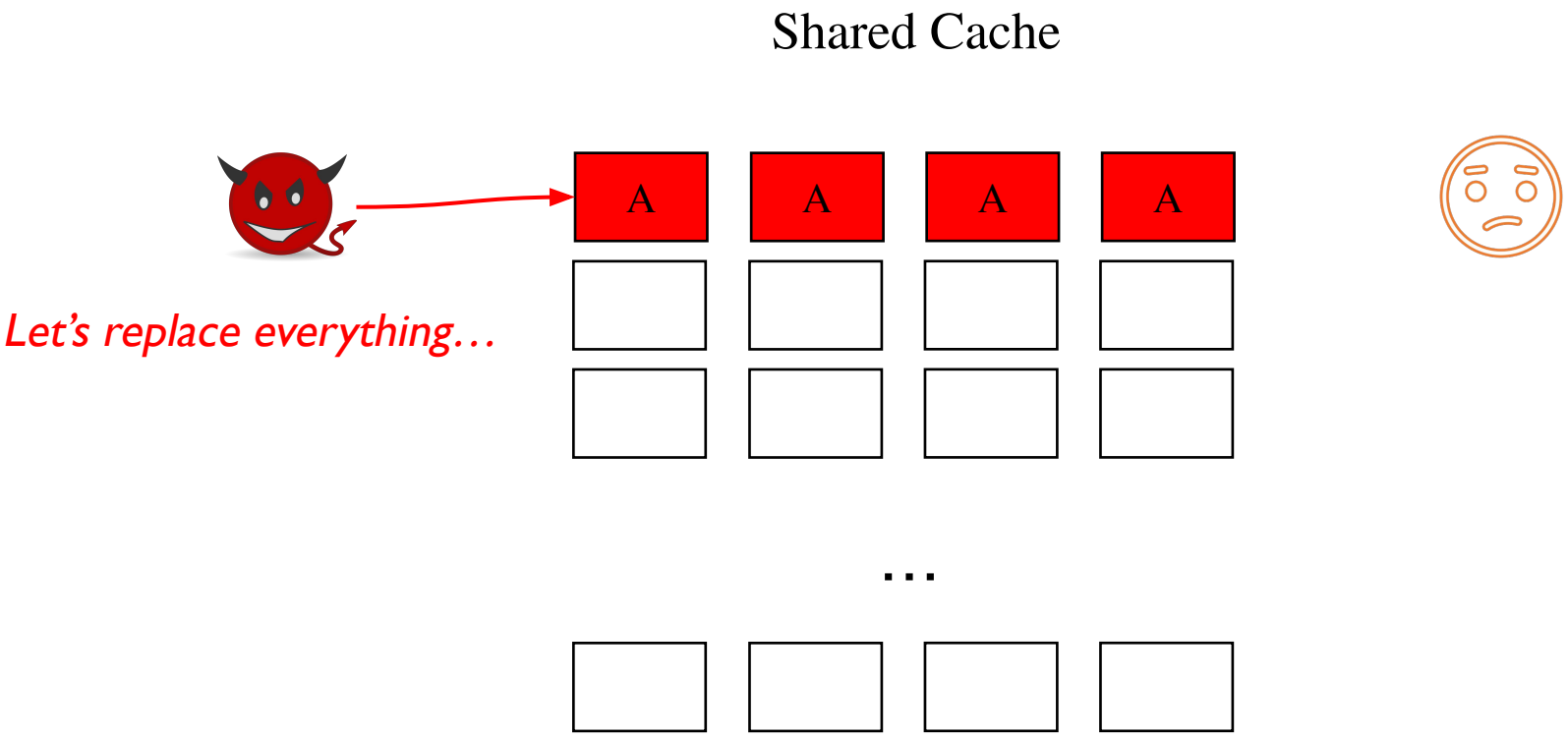
A process can not access (read or write) the memory content of any other process.

Isolation is typically enforced by OS through address translation.

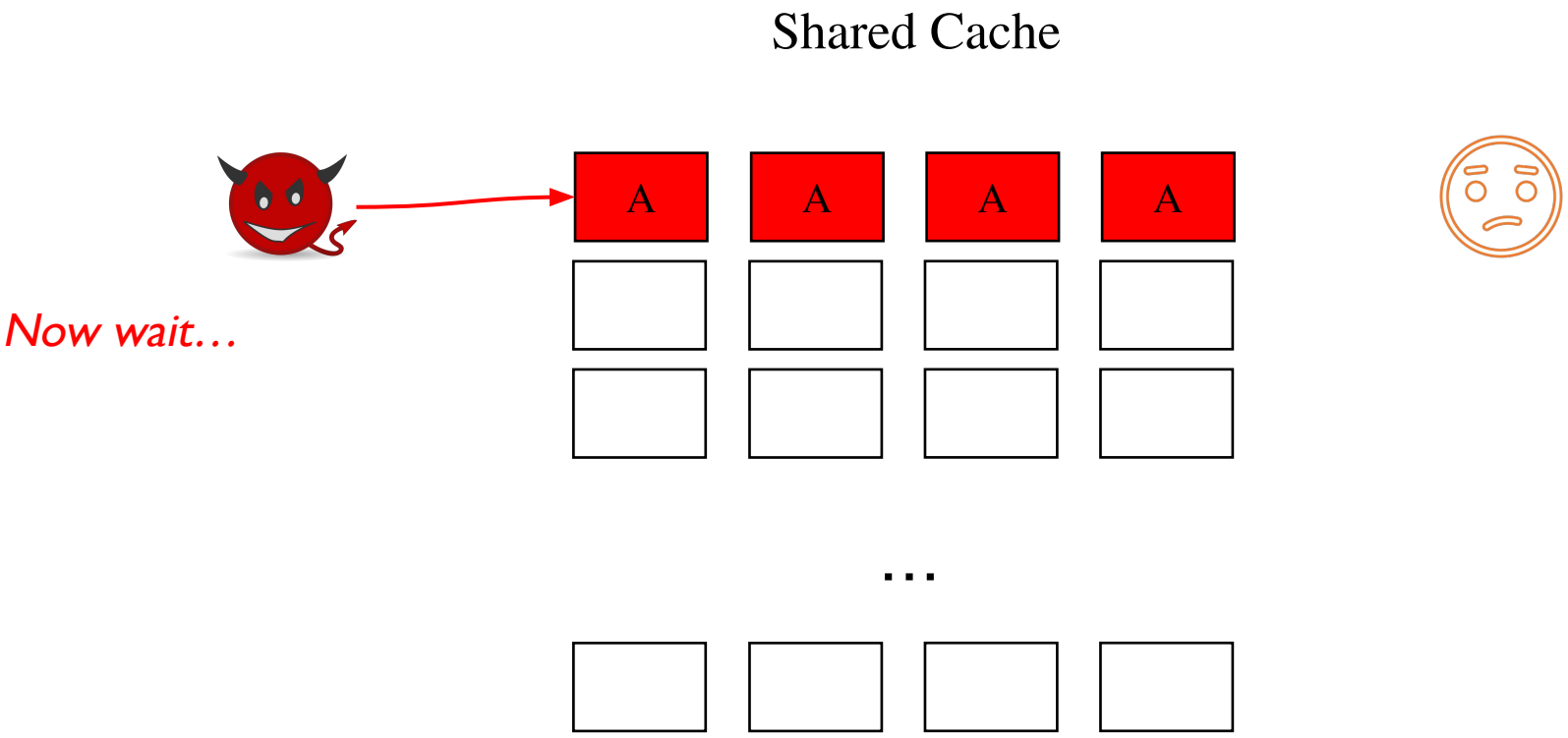
How cache side-channel works?



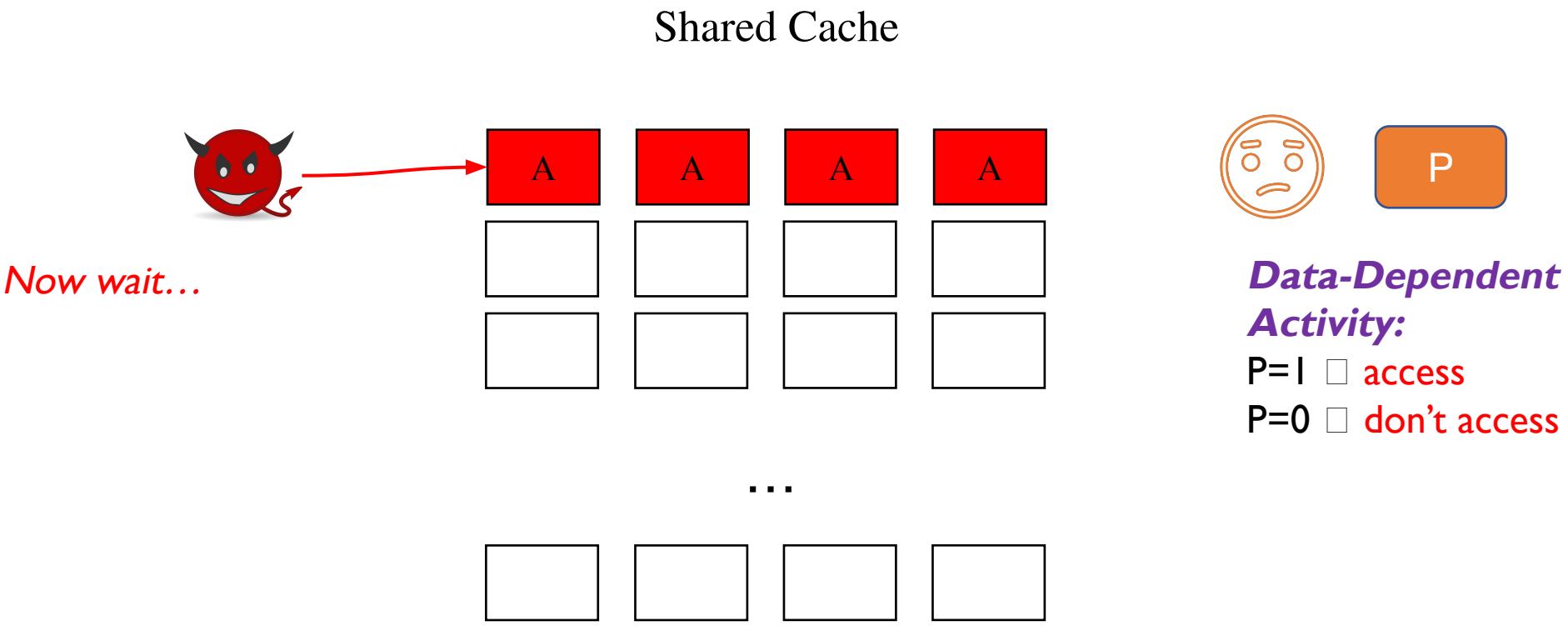
How cache side-channel works?



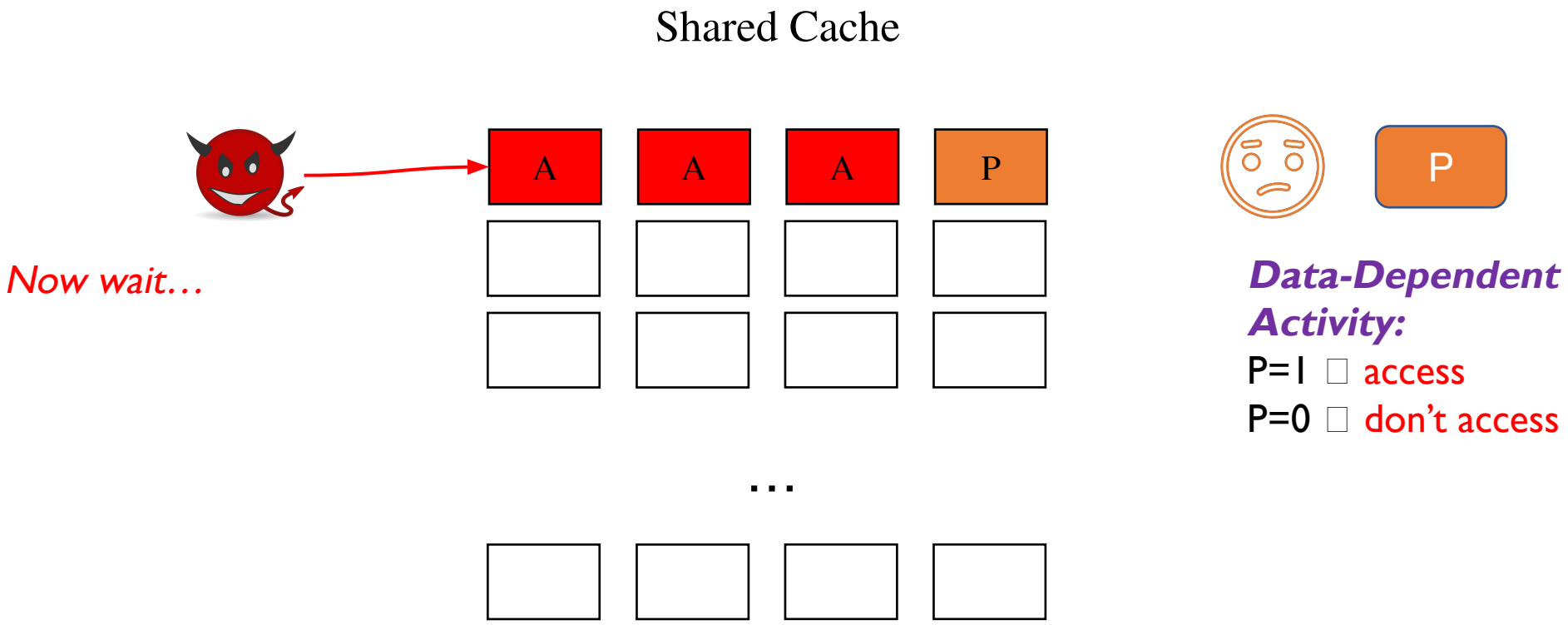
How cache side-channel works?



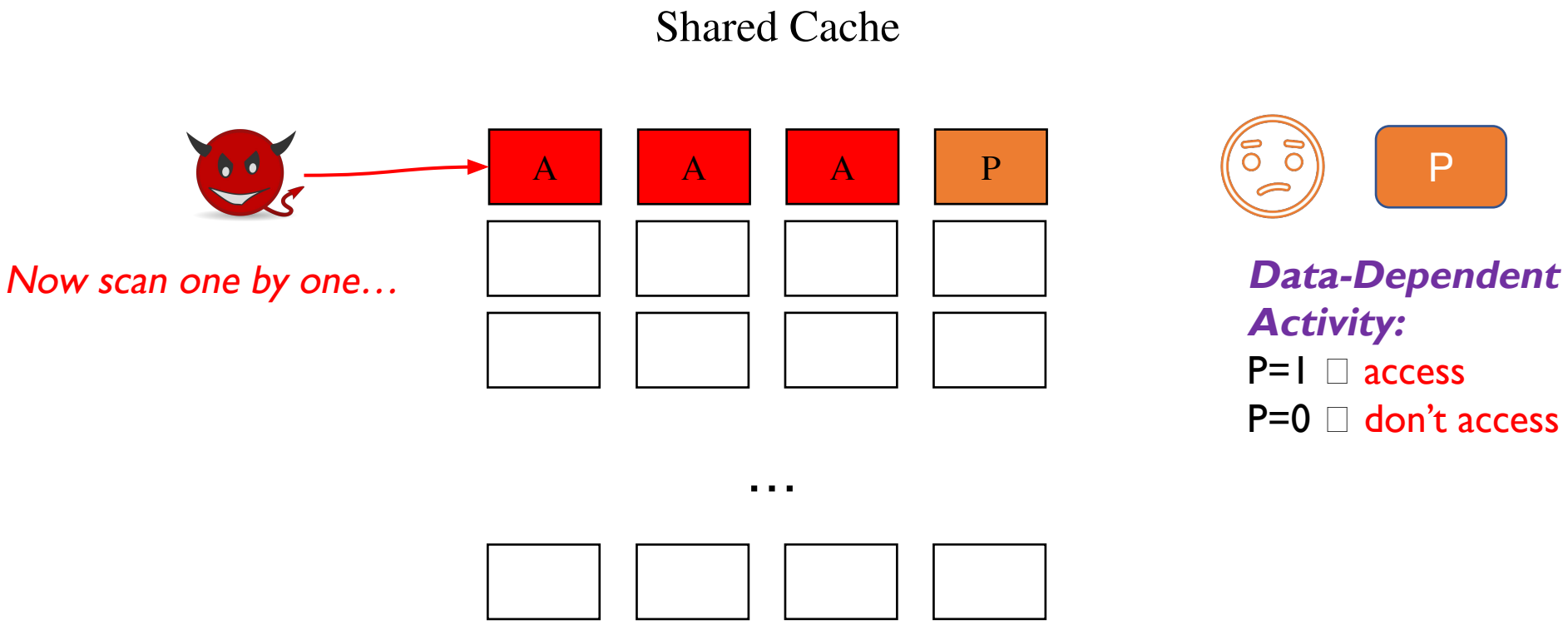
How cache side-channel works?



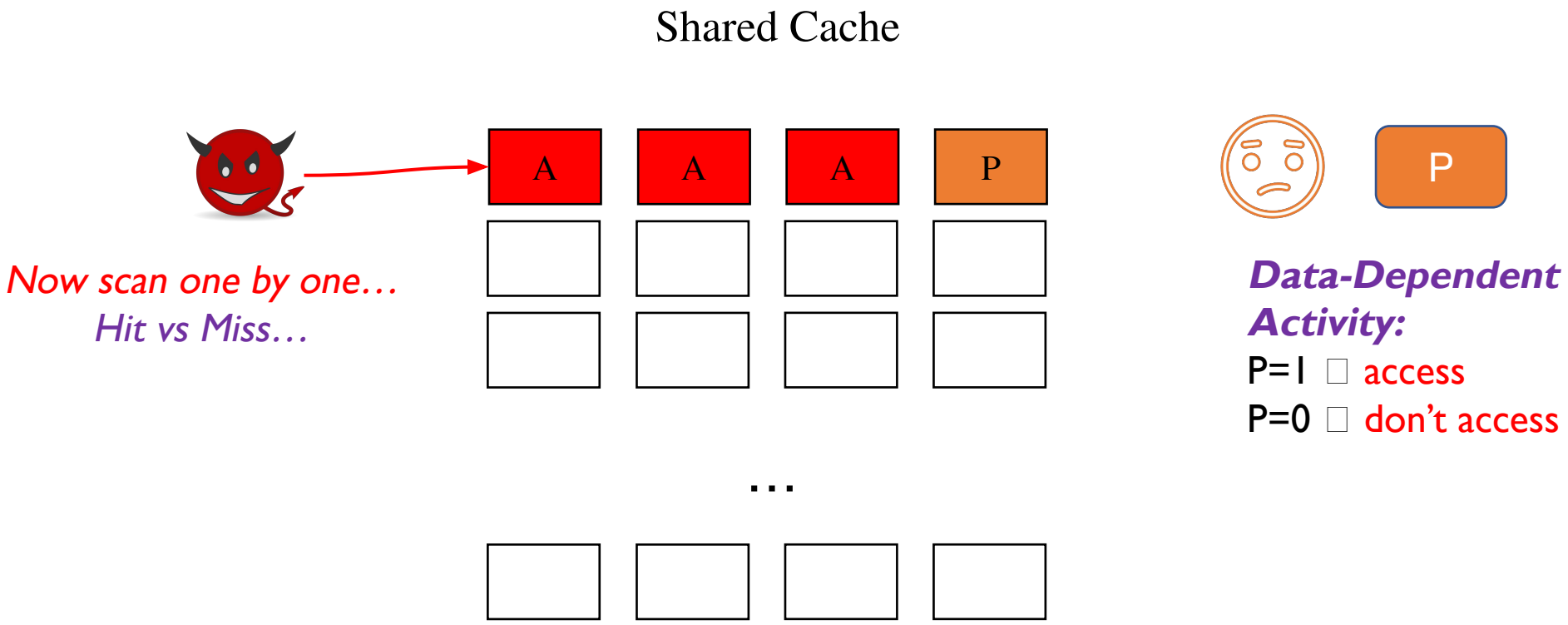
How cache side-channel works?



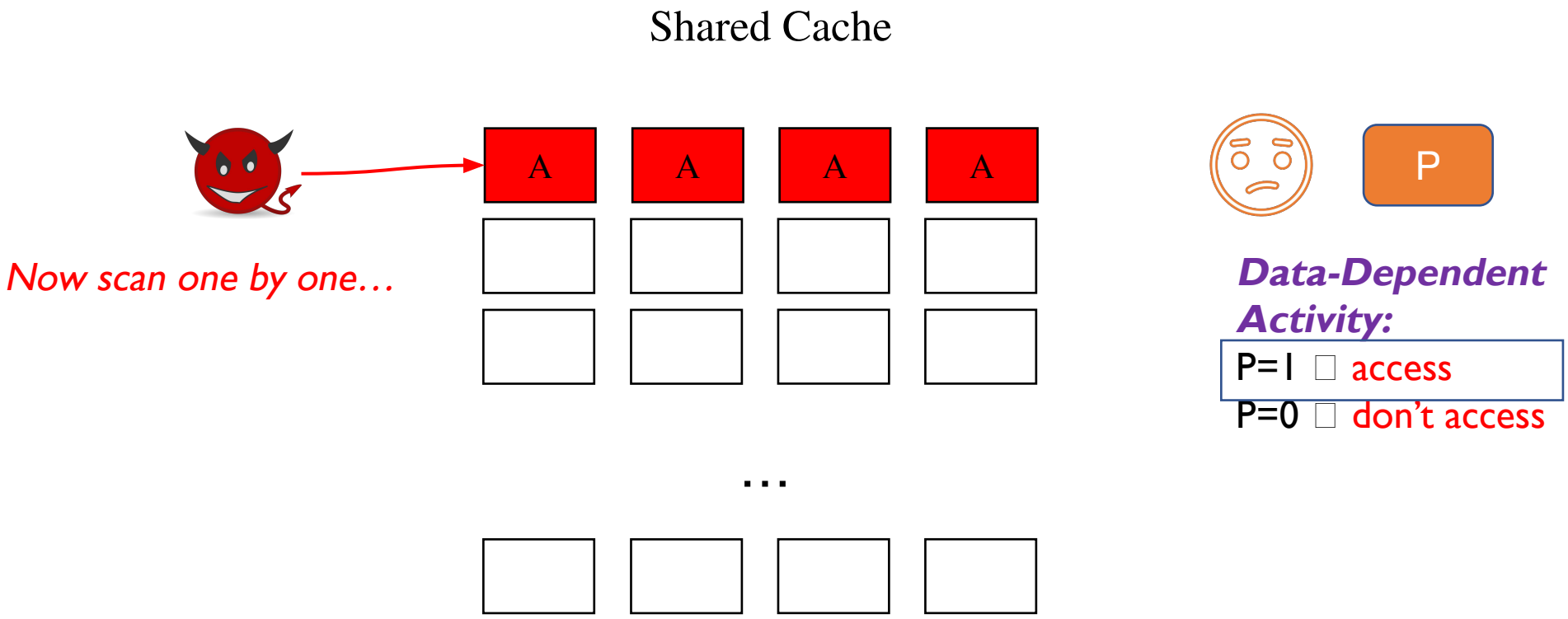
How cache side-channel works?



How cache side-channel works?



How cache side-channel works?



How cache side-channel works?

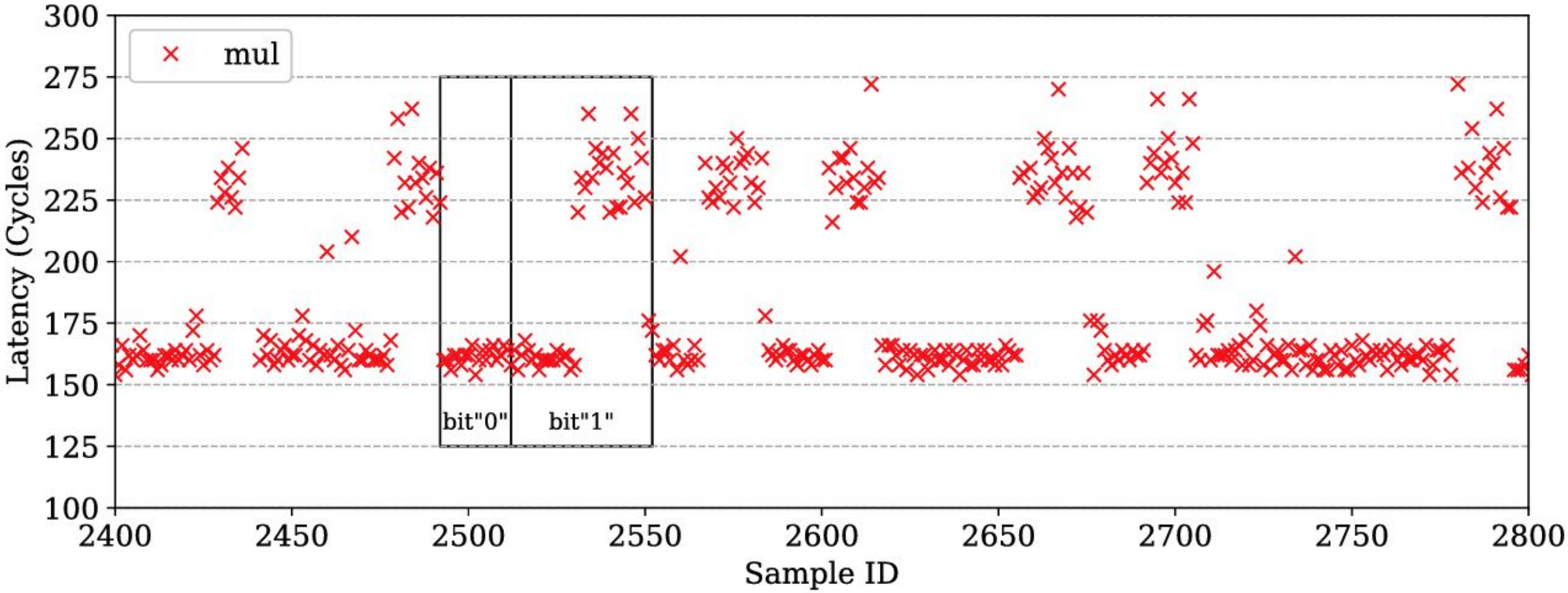
Prime+Probe

Summary:

How realistic is this?

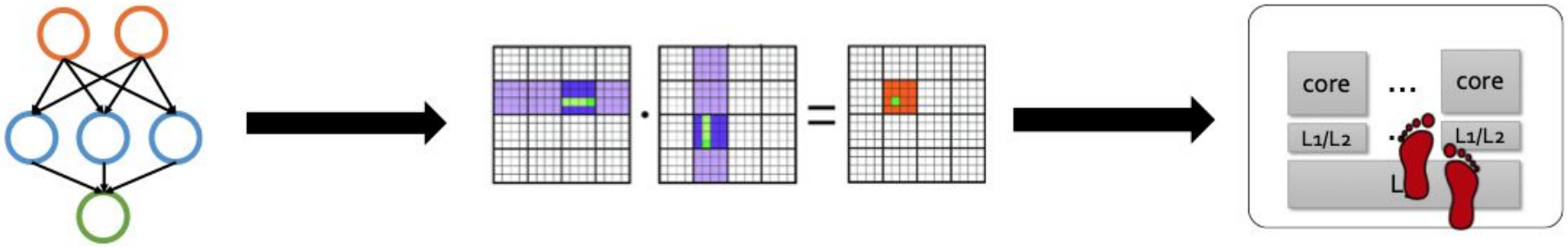
```
1  function exponent(b, e, m)
2  begin
3    x ← 1
4    for i ← |e| − 1 downto 0 do
5      x ← x^2
6      x ← x mod m
7      if (e_i = 1) then
8        x ← xb
9        x ← x mod m
10     endif
11   done
12   return x
13 end
```

*Taken from *FLUSH+RELOAD:A High Resolution, Low Noise, L3 Cache Side-Channel Attack, USENIX'14.*



Access latencies measured in the probe operation in Prime+Probe on a cache line inside the multiplication function.
A sequence of "01010111011001" can be deduced as part of the exponent.

How realistic is this?

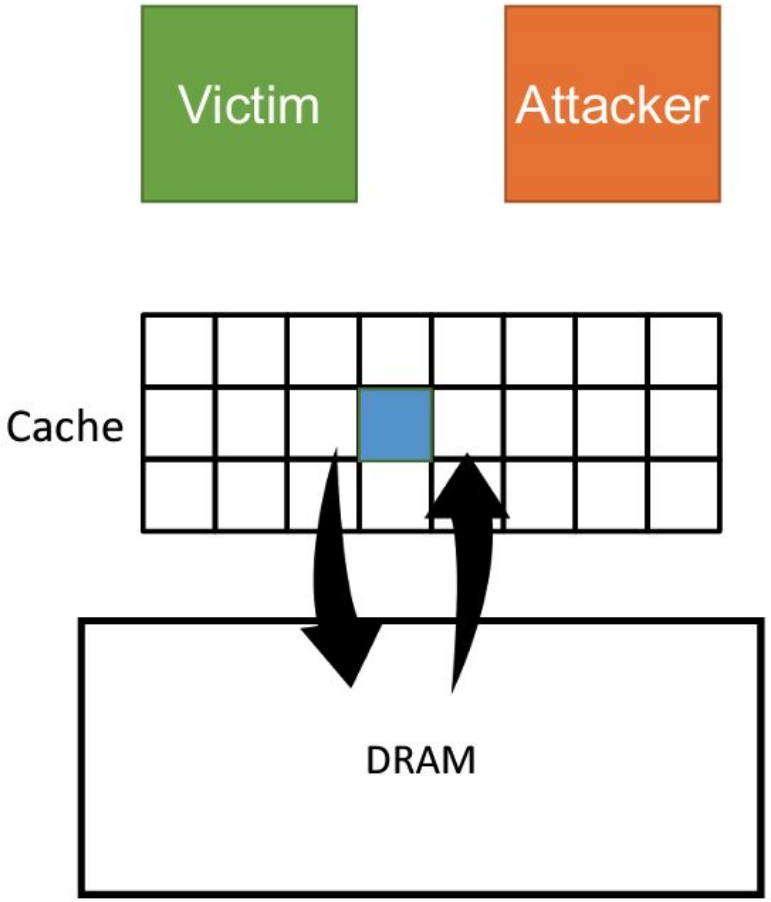


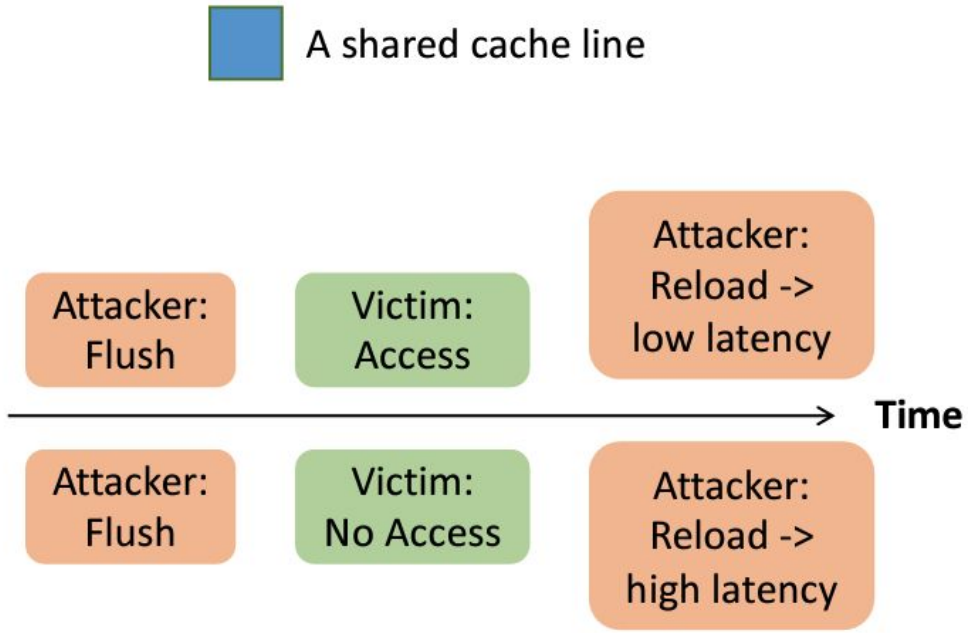
*Taken from *Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures, USENIX'20.*

Other Variants of Cache Side-Channel Attacks

- *Prime+Probe*
- *Flush+Reload*
- *Flush+Flush*
- *Take-A-Way*
- *Streamline*
- …

Flush+Reload





From MIT's Secure Hardware Design Class (6.5950/6.5951)

Is cache the only channel?

- NO!

- Virtually any *shared* resource can be attacked!

Is cache the only channel?

- Examples:
  - *Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks, USENIX'18.*
  - *DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks, USENIX'16.*
  - *Rendered Insecure: GPU Side Channel Attacks are Practical, CCS'18.*
  - *BranchScope: A New Side-Channel Attack on Directional Branch Predictor, ASPLOS'18.*
  - …

Can we defend against it?

- *Almost!*

Defense Against Side-Channels

- Software/Compiler Solutions
  - Make everything constant-time and/or data-oblivious.
    - *GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation, ASPLOS'15.*
    - *Data Oblivious ISA Extensions for Side Channel-Resistant and High Performance Computing, NDSS'19.*
    - …

Defense Against Side-Channels

- Partitioning
  - Isolate/Partition shared resources.
    - *CATalyst: Defeating last-level cache side channel attacks in cloud computing, HPCA'16.*
    - *…*

Defense Against Side-Channels

- Randomizing/Noise Addition
  - Make it hard for the adversary to setup the attack.
    - *Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping, MICRO'19.*
    - *Scattercache: Thwarting Cache Attacks via Cache Set Randomization , USENIX'19.*
    - …

Defense Against Side-Channels

- Monitoring
  - Detect the attack when it happened.
    - *ReplayConfusion: Detecting cache-based covert channel attacks using record and replay, MICRO'16.*
    - *PerSpectron: Detecting Invariant Footprints of Microarchitectural Attacks with Perceptron, MICRO'20.*
    - *…*

Defense Against Side-Channels

- *Others?*

Defense Against Side-Channels

- Challenges:
  - *Overheads: performance and area*
  - *Accuracy*
  - *Feasibility*

What is *Covert Channels?*

- Creating a covert communication between two parties (e.g., two processes).

➔ *Why do we need covert communications?*

What is *Covert Channels?*

- *Why do we need covert communications?*
  - Breaking the "least privilege principle"
    - One process has access to secrets but is monitored and can't connect to "outside" world.
    - The other process can not access the secrets but can freely connect to outside. (examples: …)
    - An adversary (e.g., a rogue employee) can create two malicious processes and "extract" information.

Key Takeaways

- Any shared resources can leak secrets!
- Extracting secrets are possible through digital side-channels.
- We need to find effective defense mechanisms, and find the best balance between performance and security.

Transient Execution Attacks

Recall OoO

- In-order frontend, OoO backend

- How did we fix exception and mispeculation?
  - ROB → commit in-order, flush if things are wrong!

Transient Execution Attacks

- Create transient windows → modern processors have tons of this!

- Access something *illegal* during this window → normally the OS would not allow it but due to speculation, we first issue the load and then later check → still fine because either the instruction never happens (due to a parent flush) or OS throws an error.

- BUT: issuing a load will bring the data into the cache!!! → launch a cache side-channel now!

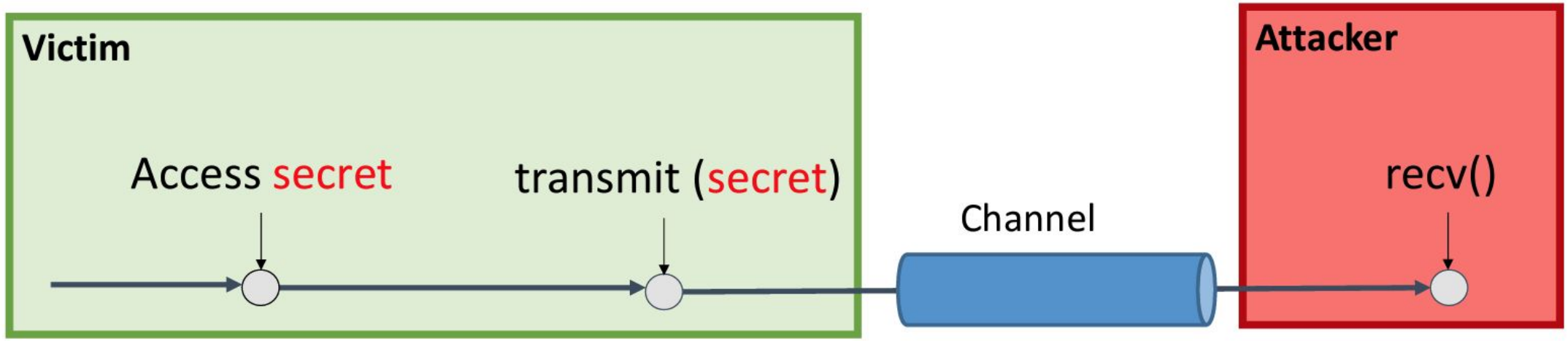
What can we do with this?

- Many things! Virtually reading any arbitrary memory as long as we can find the right sequence of gadgets.

- Examples: Spectre (several variants) and Meltdown

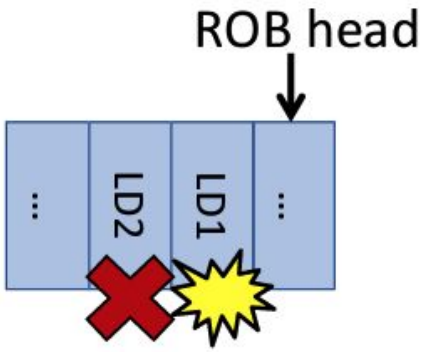
General Idea

- Access the secret during the transient window!



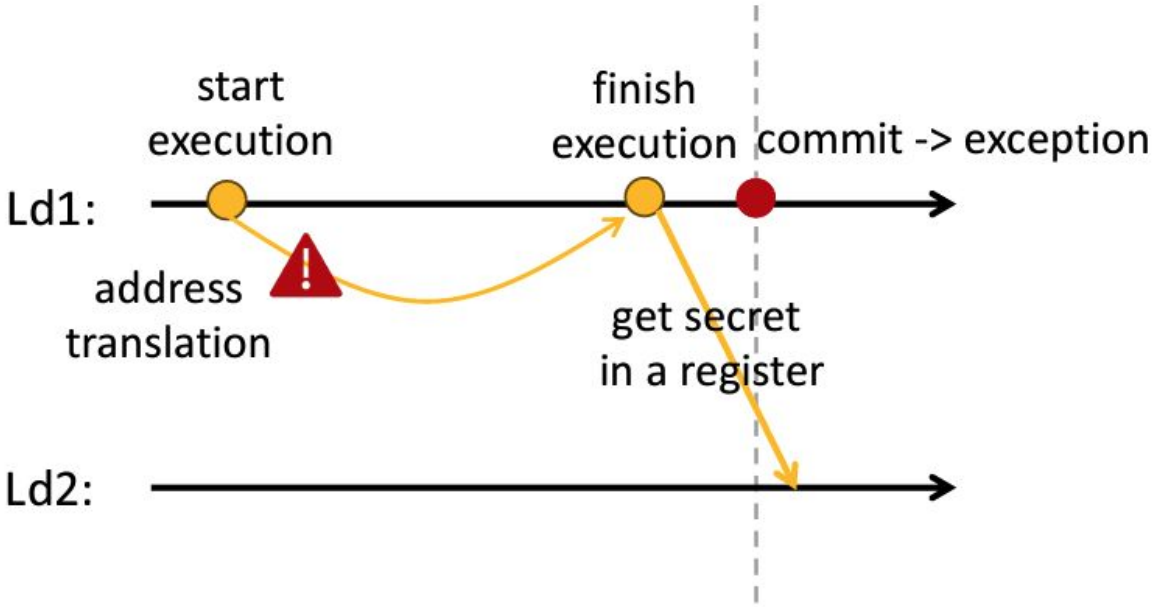
How can we read arbitrary memory?

```
……
Ld1: uint8_t secret = *kernel_address;
Ld2: unit8_t dummy = probe_array[secret*64];
```

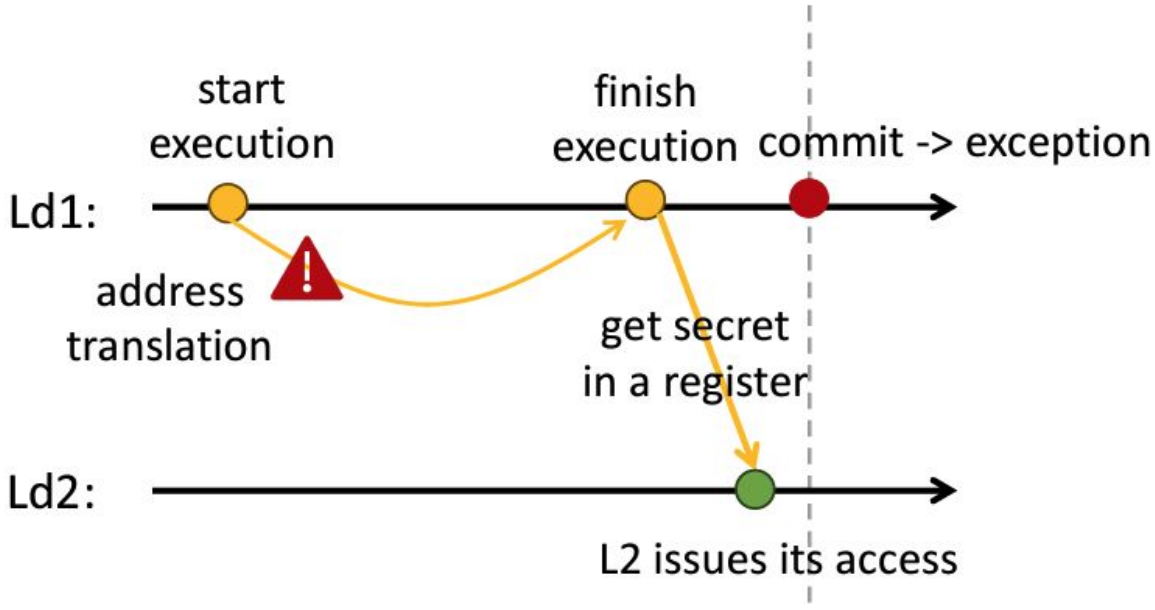


Meltdown

Case 1: Fail. Ld2 is squashed before the corresponding memory access is issued.



Case 2: Attack works. Ld2's request is sent out before the instruction is squashed.



Meltdown Steps

```
……
Ld1: uint8_t secret = *kernel_address;
Ld2: unit8_t dummy = probe_array[secret*64];
```

1. Setup: Attacker allocates probe_array, with 256 cache lines. Flushes all its cache lines.

2. Transmit: Attacker executes (code shown above).

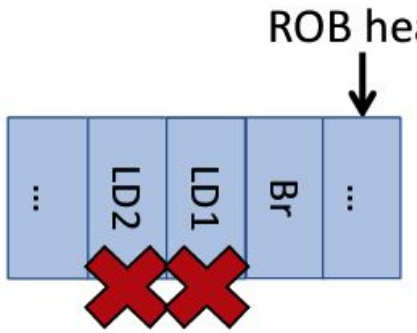
3. Receive: After handling protection fault, attacker performs cache side channel attack to figure out which line of probe_array is accessed → recovers byte.

Spectre

- Consider the following kernel code, e.g., in a syste

```
Br:  if (x < size_array1) {
Ld1:       secret = array1[x]
Ld2:       y = array2[secret*64]
     }
```

Always malicious?
No. It may be a benign misprediction.
We do not consider Spectre as a bug.

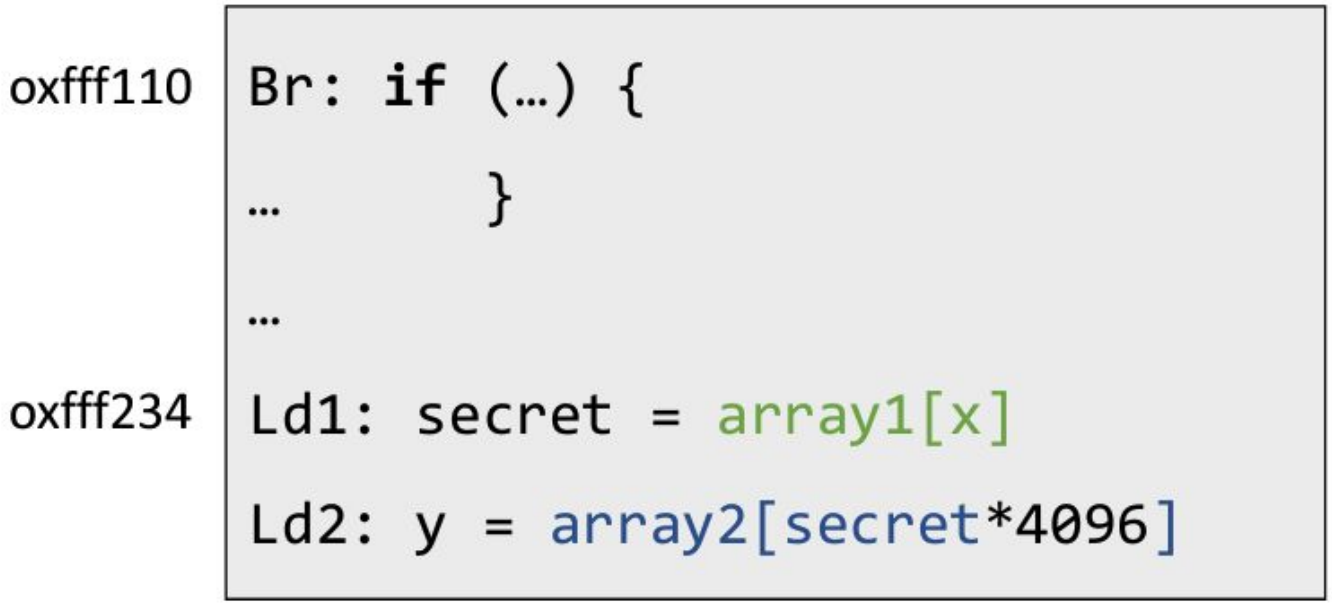


Attacker to read arbitrary memory:

1. Setup: Train branch predictor
2. Transmit: Trigger branch misprediction; *&array1[x]* maps to some desired kernel address
3. Receive: Attacker probes cache to infer which line of *array2* was fetched

Variant 2: Branch target

- Jumping to arbitrary targets!



How to fix this?

- Not easy! very deep integrated in the pipeline!
- Many possibilities, many gadgets!

Mitigations?

- Disable speculation!
  - Always
  - Maybe sometimes?

- Protect the branch predictor
  - Not allow it to be re-trained easily

- Make speculation invisible?
  - How?

Takeaways

- Speculation, fundamentally can be exploited to bypass isolation.
- When combined by a side-channel attack, these transient execution attacks can leak a sensitive secret.
- Defense against this is quite challenging (due to fundamental limitations) but very active.

End of Presentation

UCLA Samueli School of Engineering

Acknowledgement

- This course is partly inspired by the following courses created by my colleagues:
  - 6.888 MIT (Yan)
  - CS 598 UIUC (Fletcher)
  - CS 7290 Georgia Tech (Kim)
  - ECE 752 Wisconsin (Lipasti)
  - ECE 7103A Georgia Tech (Qureshi)