Microarchitectural Side-Channel Vulnerabilities – Can Data Analytics Help?

Shih-Lien (Linus) Lu
Washington State University

Outlines

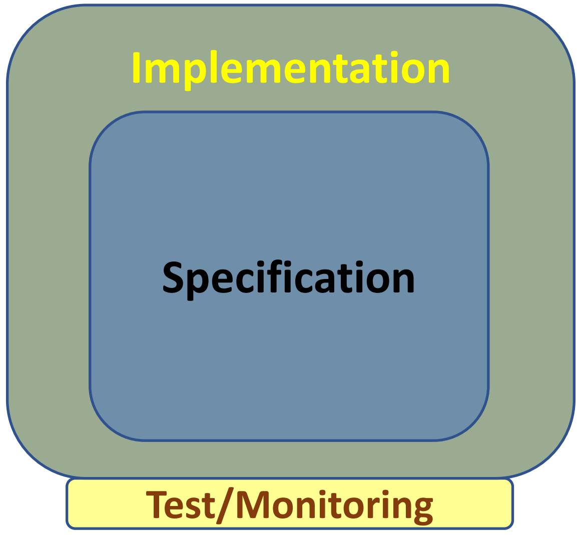
- **Security goal and challenge of securing systems**
- **Common attacks and side-channels**
  - **Physical side-channels**
- **Microarchitecture side-channels and …**
  - **Examples**
    - **Cache side-channel**
    - **Main memory side-channel**
    - **Speculation state side-channel**
  - **Some recent publications**
- **What can we do?**
  - **Static analysis**
  - **Dynamic monitoring**
- **Summary**

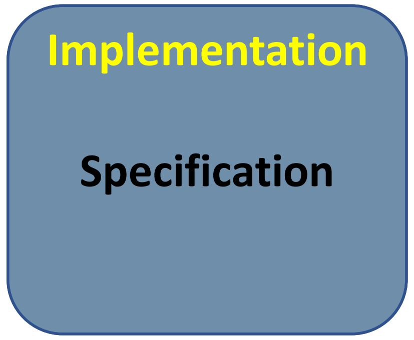
Objectives of Information Security - CIA

- Confidentiality
  - Only authorized parties should be able to access the required data
    - Access mean to understand the contents of the data
    - Should not disclose unnecessary content
- Integrity
  - The content should only be altered by authorized users intentionally
    - Not tempered or degraded
    - Purposely or inadvertently
  - Non-repudiation
- Availability
  - Timely accessibility of data to authorized entities when needed
    - System availability
    - Communication channel accessibility
    - Data readiness

Why is Security Challenging?

- **Correctness**
  - **Met specification (e.g. ISA spec)**
- **No vulnerability means …**
  - **No extra features beyond the specification**
  - **No side-channels**
    - **Speculation**
    - **Transient states**
    - **Physical manifestation**
- **But we want …**
  - **Performance**
  - **Observability**
  - **Testing/debugging**
  - **…**





Outlines

- Security goal and challenge of securing systems
- Common attacks and side-channels
    - Physical side-channels
- Microarchitecture side-channels and …
    - Examples
        - Cache side-channel
        - Main memory side-channel
        - Speculation state side-channel
    - Some recent publications
- What can we do?
    - Static analysis
    - Dynamic monitoring
- Summary

Cryptographic Failures (Sensitive Data Exposure)

- #2 of the Top 10 Attacks by OWASP® 2021
  - Failures related to cryptography / leakage of information
- One attack vector: side-channel
  - Security exploit that attempts to extract secrets from a chip/system
  - Leads to unintended data leakage
- Types of side-channel
  - Physical
    - Power usage of chip
    - Radiation of light
    - Timing
  - Microarchitectural
  - Software based
  - Hybrid

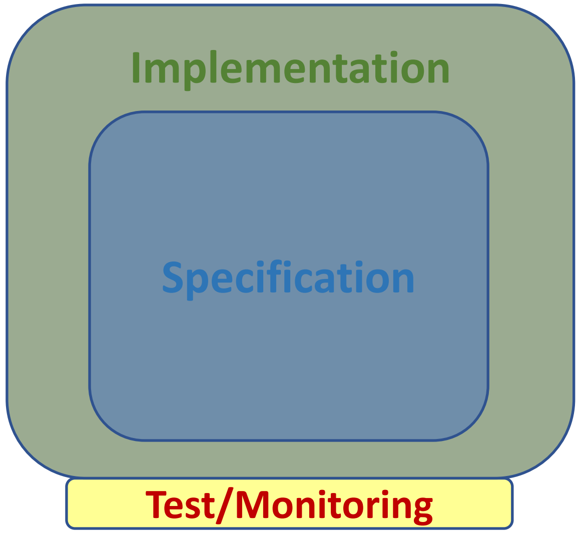
https://owasp.org/www-project-mobile-top-10/2014-risks/m4-unintended-data-leakage

Outlines

- Security goal and challenge of securing systems
- Common attacks and side-channels
  - Physical side-channels
- Microarchitecture side-channels and …
  - Examples
    - Cache side-channel
    - Main memory side-channel
    - Speculation state side-channel
  - Some recent publications
- What can we do?
  - Static analysis
  - Dynamic monitoring
- Summary

Microarchitectural Side-Channel

- What is microarchitecture? Why?
  - Implementation of (Instruction Set) Architecture (ISA)
    - Ex. Of ISA x86-64, ARMv8-A, RISC-V, MIPS
  - Multiple implementations of one ISA
- Program computes on private data
  - Interact with microarchitecture optimization
  - Data dependent hardware resource modulation
    - Observable by malicious threat actors
- Why this is so insidious?
  - Need no physical access
    - Cloud: multi-tenant
    - Threat agent is a “legitimate” tenant of the cloud



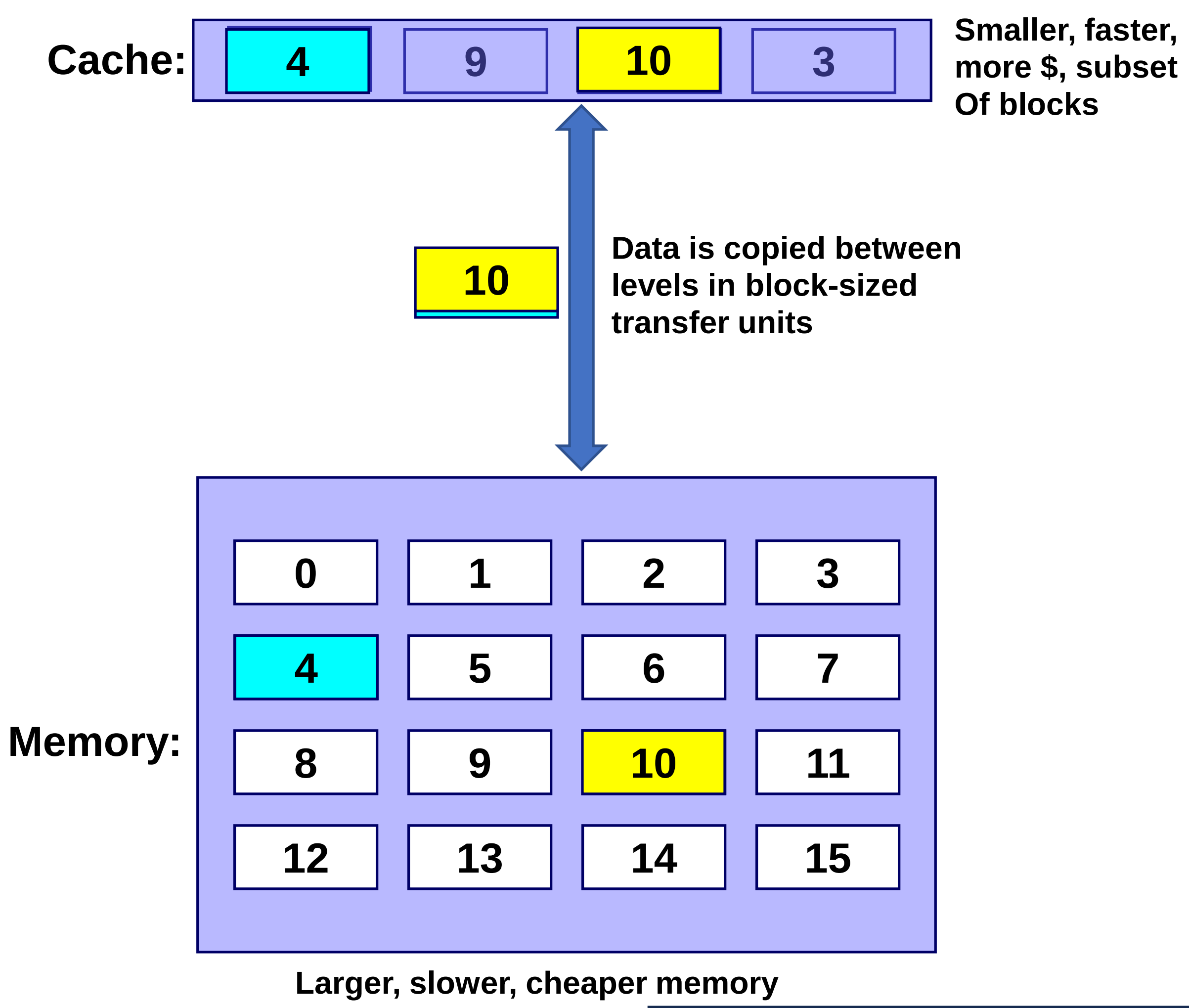
Microarchitectural Side-Channel (2)

- Why this is so insidious? (cont.)
  - Shared resource as the covert channel
    - Host has no explicit control of the share resource
    - Behavior can be affected/observed by another tenant
  - Conflicting goals – performance vs. security
  - Slowing down of Moore's law $\rightarrow$ more microarchitecture optimization
    - Increasing attack surfaces
- Cause
  - Speed up common cases
    - Lower latency when the "technique" works
    - Longer latency otherwise
  - Performance monitoring mechanisms

Passive Information Gathering & Active Information Leakage

Started with Cache

- High-level intro to cache

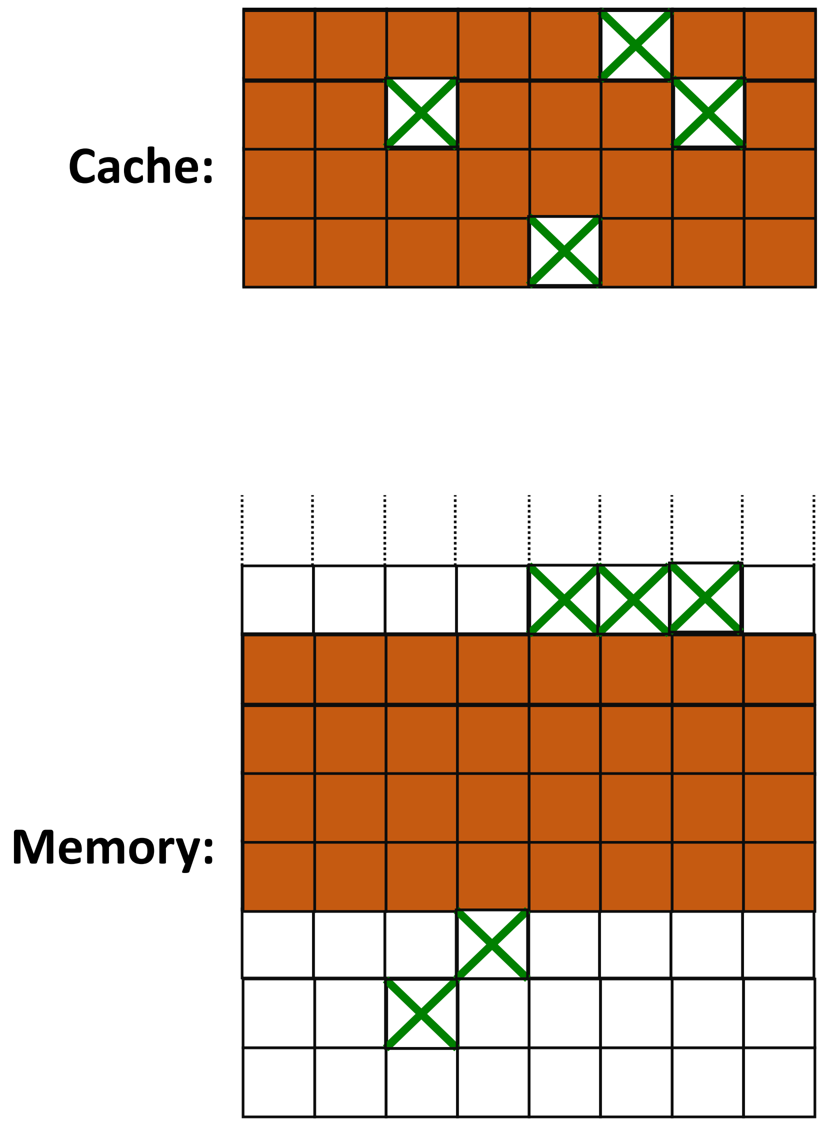

Cache:
4
9
10
3
Smaller, faster, more $, subset Of blocks
10
Data is copied between levels in block-sized transfer units
Memory:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper memory

- Principles behind caching
  - Temporal locality
    - Re-use of specific data in time
  - Spatial locality
    - Use of data close to each other
    - Example loop:
      - for (…..) { sum=sum+x[i];}
- When CPU read
  - Cache Hit
    - Fast – lower latency
  - Cache Miss
    - Slow – longer latency

Cache/Cache Hierarchy is a Microarchitecture Technique

Prime+Probe [Per05, OST06]

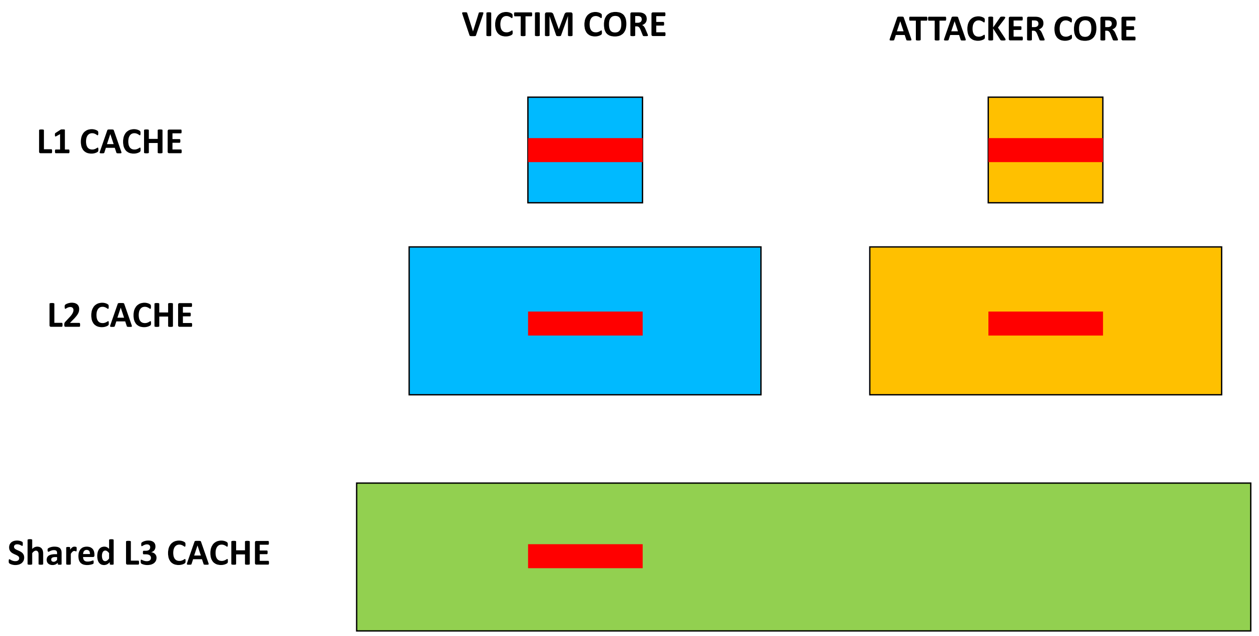
- Attacker chooses a cache-sized memory buffer
- Attacker accesses all the lines in the buffer, filling the cache with its data
- Victim executes, evicting some of the attackers lines from the cache
- Attacker measures the time to access the buffer
  - Accesses to cached lines is faster than to evicted lines
  - Learn victim’s address



[Per05] C. Percival, "Cache Missing for Fun and Profit", BSDCan, 2005
[OST06] D. A. Osvik, A. Shamir and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES", CT-RSA 2006

Flush+Reload [Y&F14]

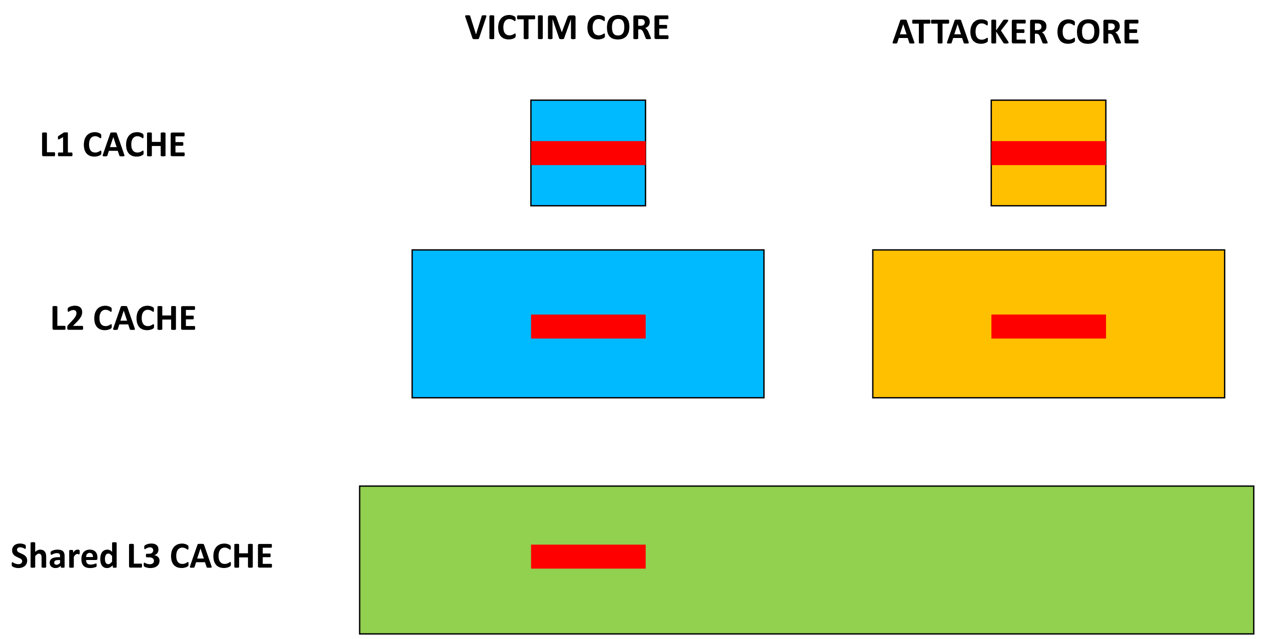
- Work on cache lines (vs. cache sets in Prime+Probe)
- Needs physical memory sharing with victim
- Attacker use "`CLFLUSH`" to evict line
- Due to inclusive property victim private cache line evicted also



Y. Yarom & K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," 23rd USENIX Sec Symp, 2014

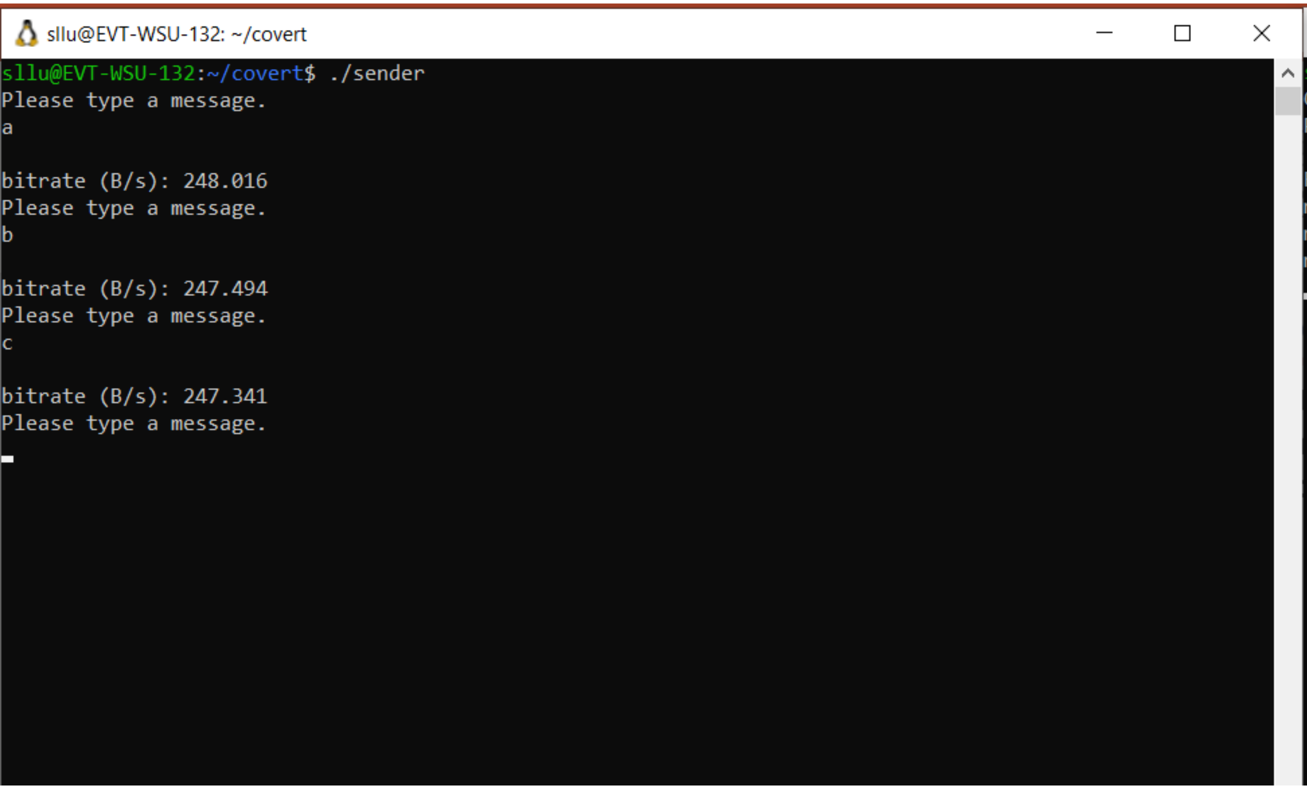
Flush+Reload (Cont.)

- Attacker waits
- Victim reload the cache line
- Attacker reload and use read time stamp counter to time (rdtsc)
  - If fast then line in L3 (used by victim) else in DRAM (not used by victim)



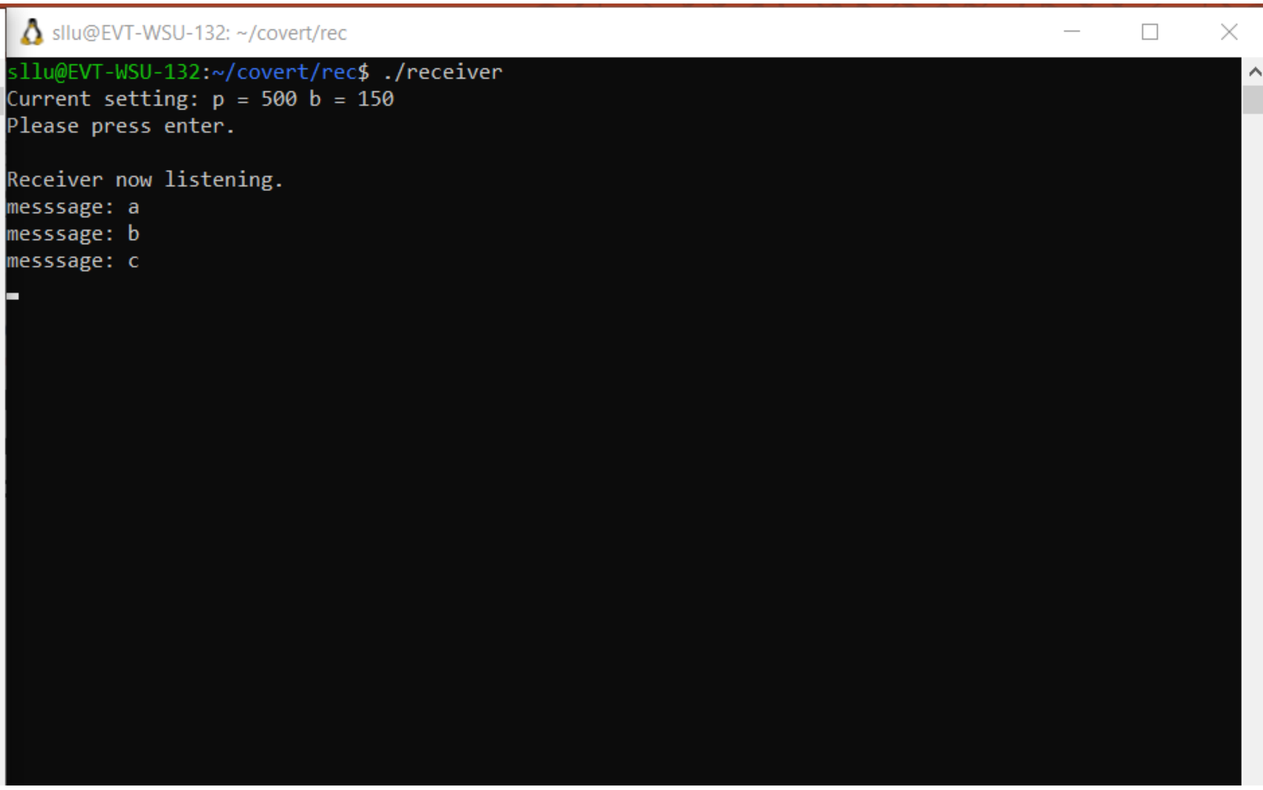
A Demo of Cache Covert Channel

- What can threat actor do with cache side channels?
  - Cryptographic key leakage
  - A simple way to establish a covert channel to leak data



sllu@EVT-WSU-132: ~/covert

```
sllu@EVT-WSU-132:~/covert$ ./sender
Please type a message.
a

bitrate (B/s): 248.016
Please type a message.
b

bitrate (B/s): 247.494
Please type a message.
c

bitrate (B/s): 247.341
Please type a message.
```

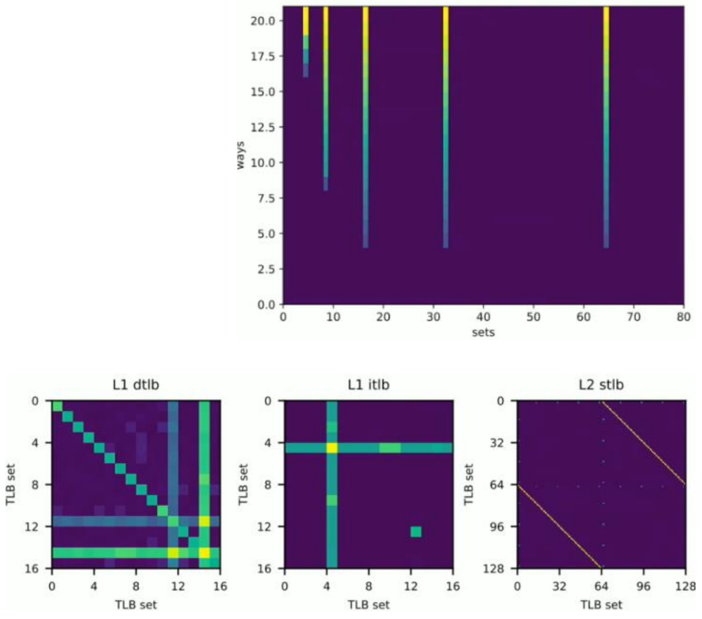


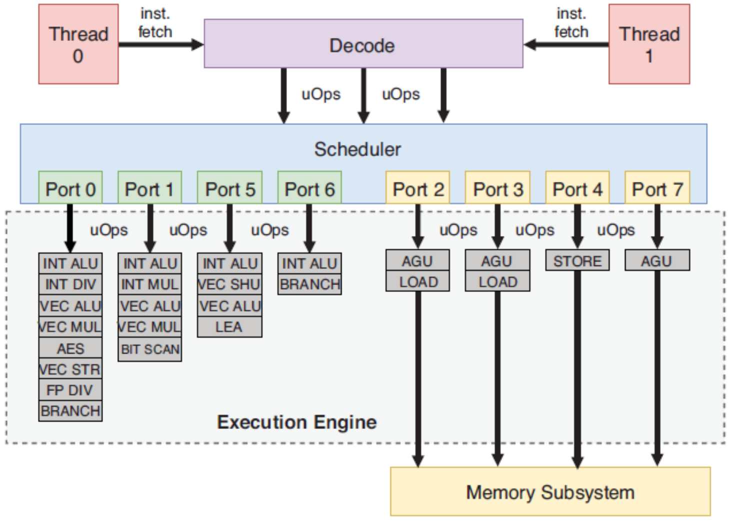
sllu@EVT-WSU-132: ~/covert/rec

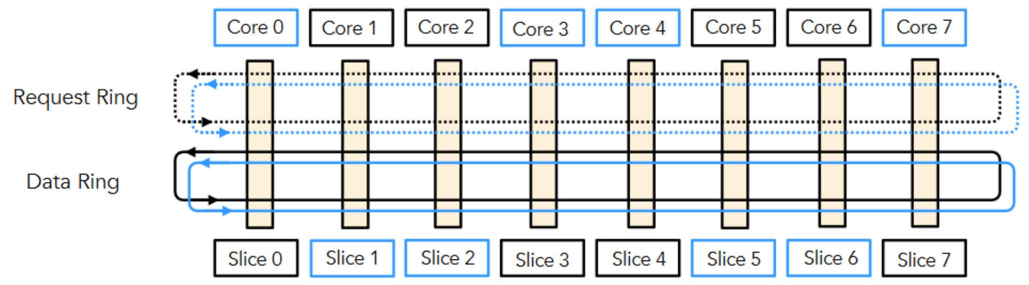
```
sllu@EVT-WSU-132:~/covert/rec$ ./receiver
Current setting: p = 500 b = 150
Please press enter.

Receiver now listening.
messsage: a
messsage: b
messsage: c
```

Other On-Chip Shared Structures

- TLBs [Gra 18]
  - L1iTLB, L1dTLB, L2TLB contention
  - Reverse engineered by experimenting with page walks and perf cntr
  - Access set of L1dTLB and observe sharing between two threads
  - Observe EdDSA ECC key multiplication
- Functional Units [Ald 19]
  - Port contention of SMT
- On-chip networks [PLF 20]
  - Ring contention
    - Overlapped segments



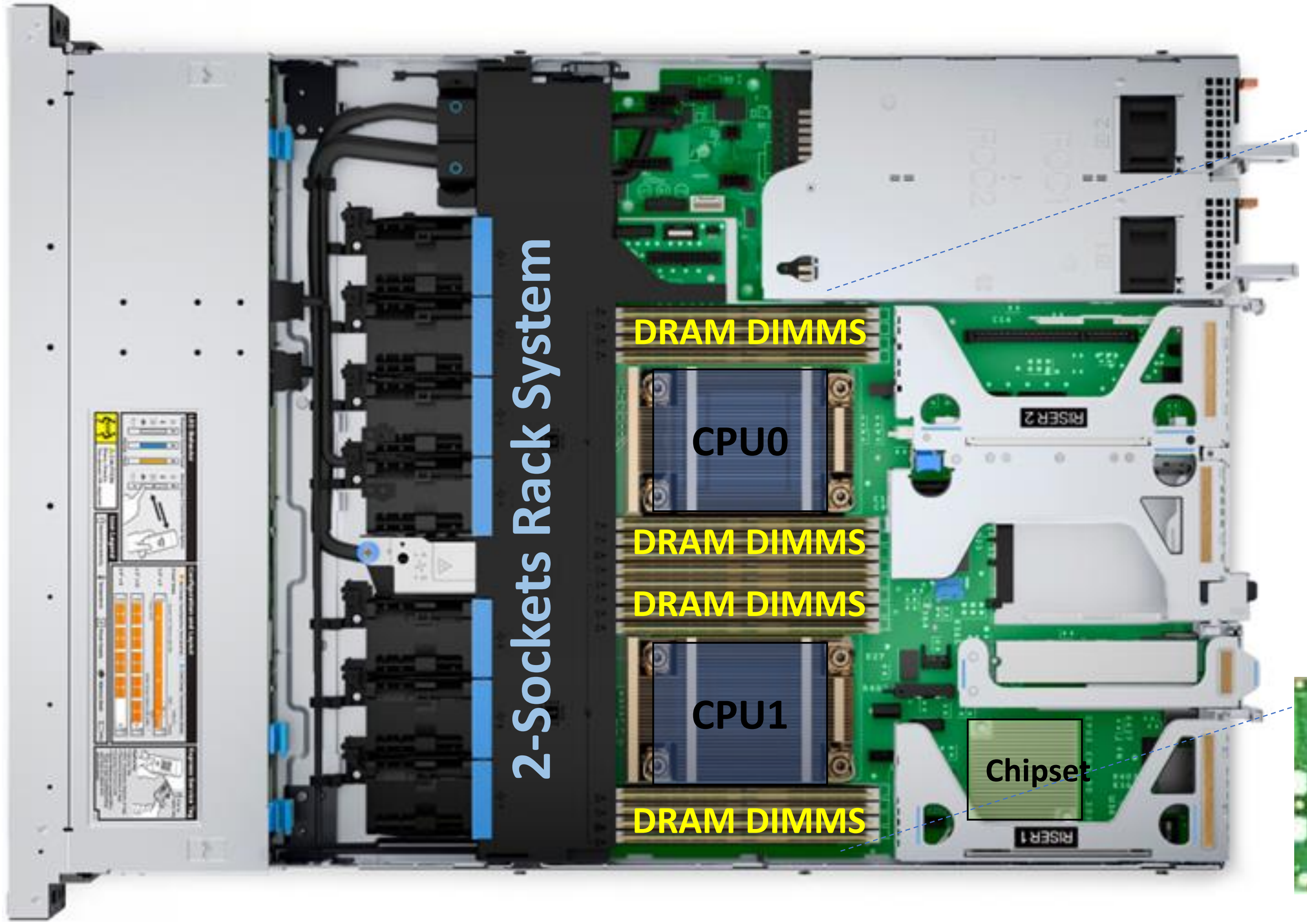


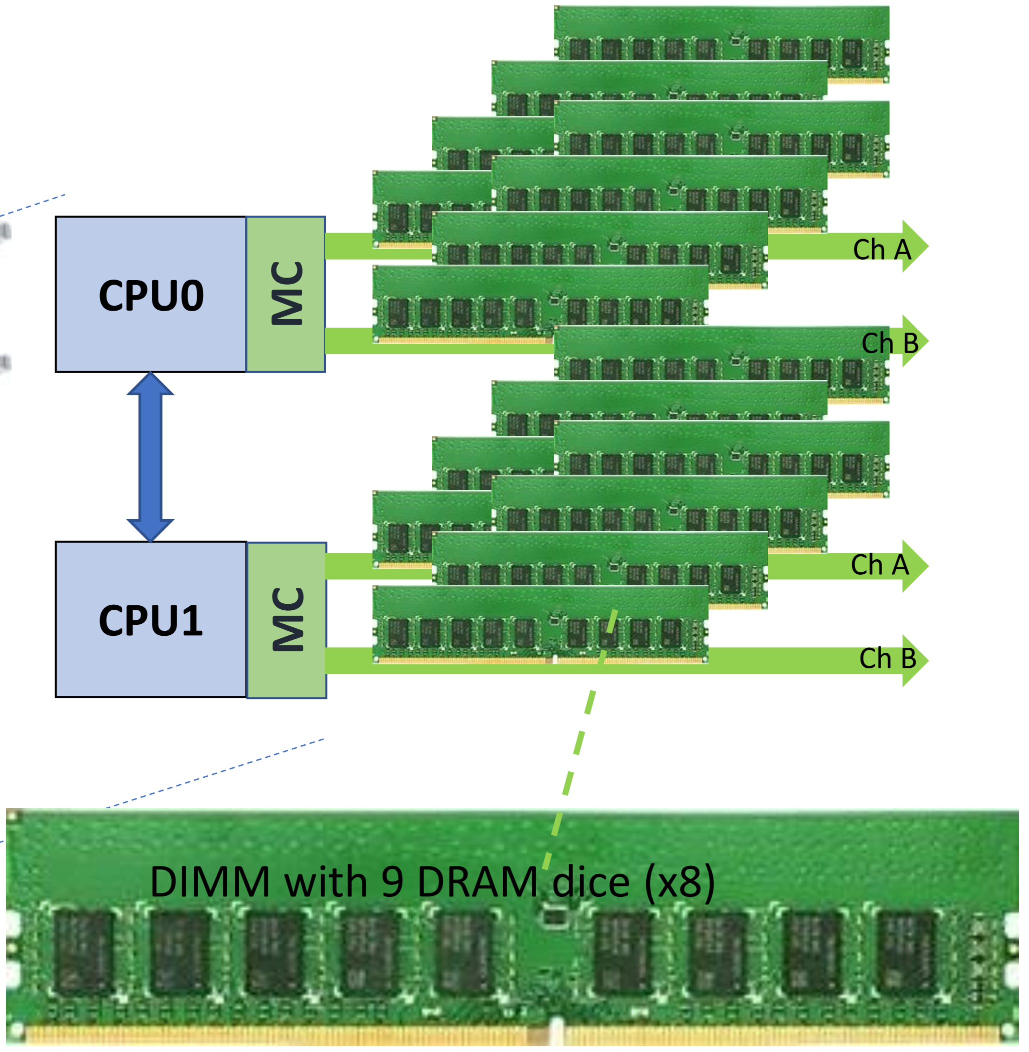


Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks, 27th USENIX Security Symp. 2018
Port Contention for Fun and Profit. In 2019 IEEE Symposium on Security and Privacy (SP). 2019
Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical, 30th USENIX Security Symp. 2021

DRAM in a Computer System

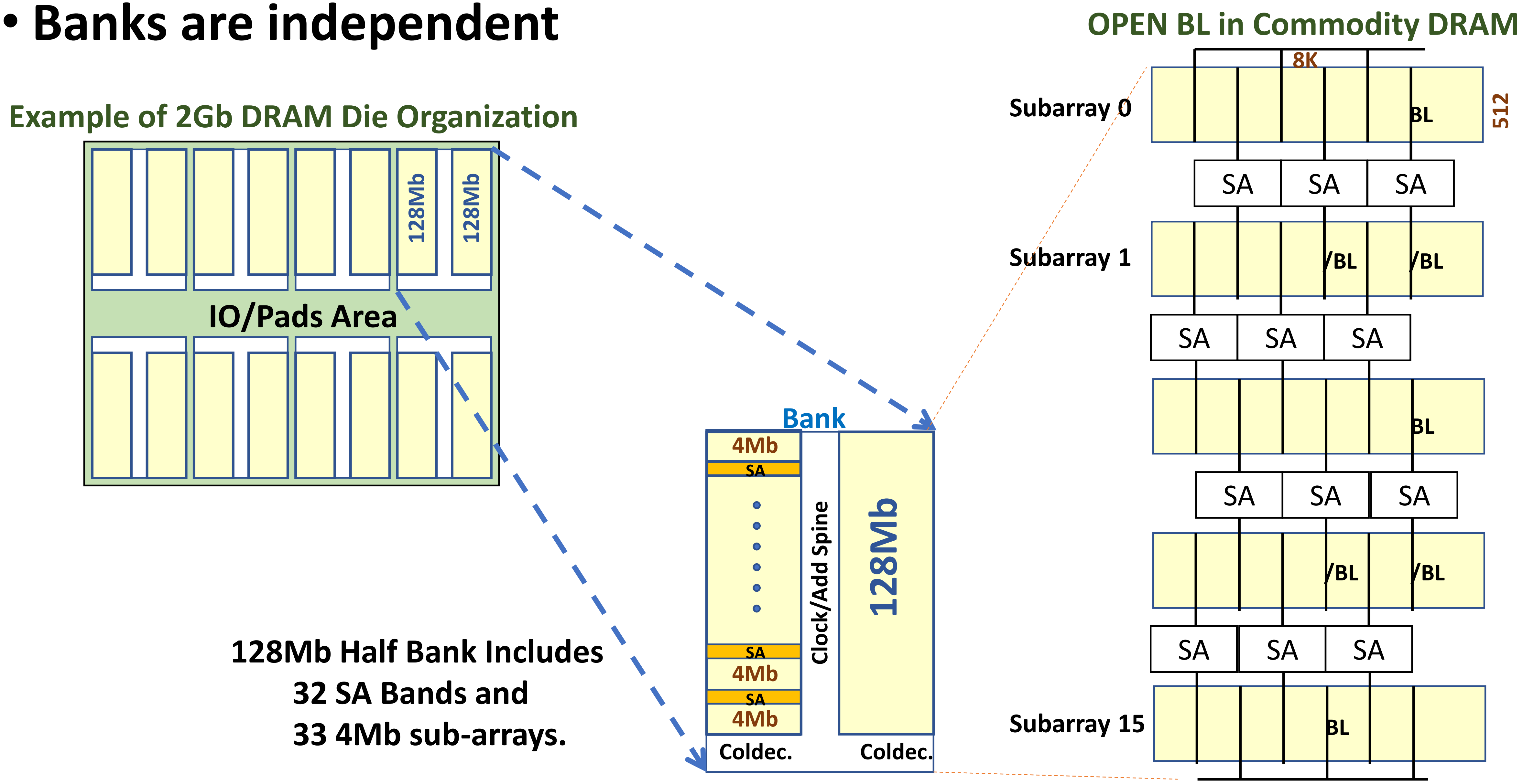
- Non-inclusive L3
- Isolate on-chip shared L3





DRAM Die Internal

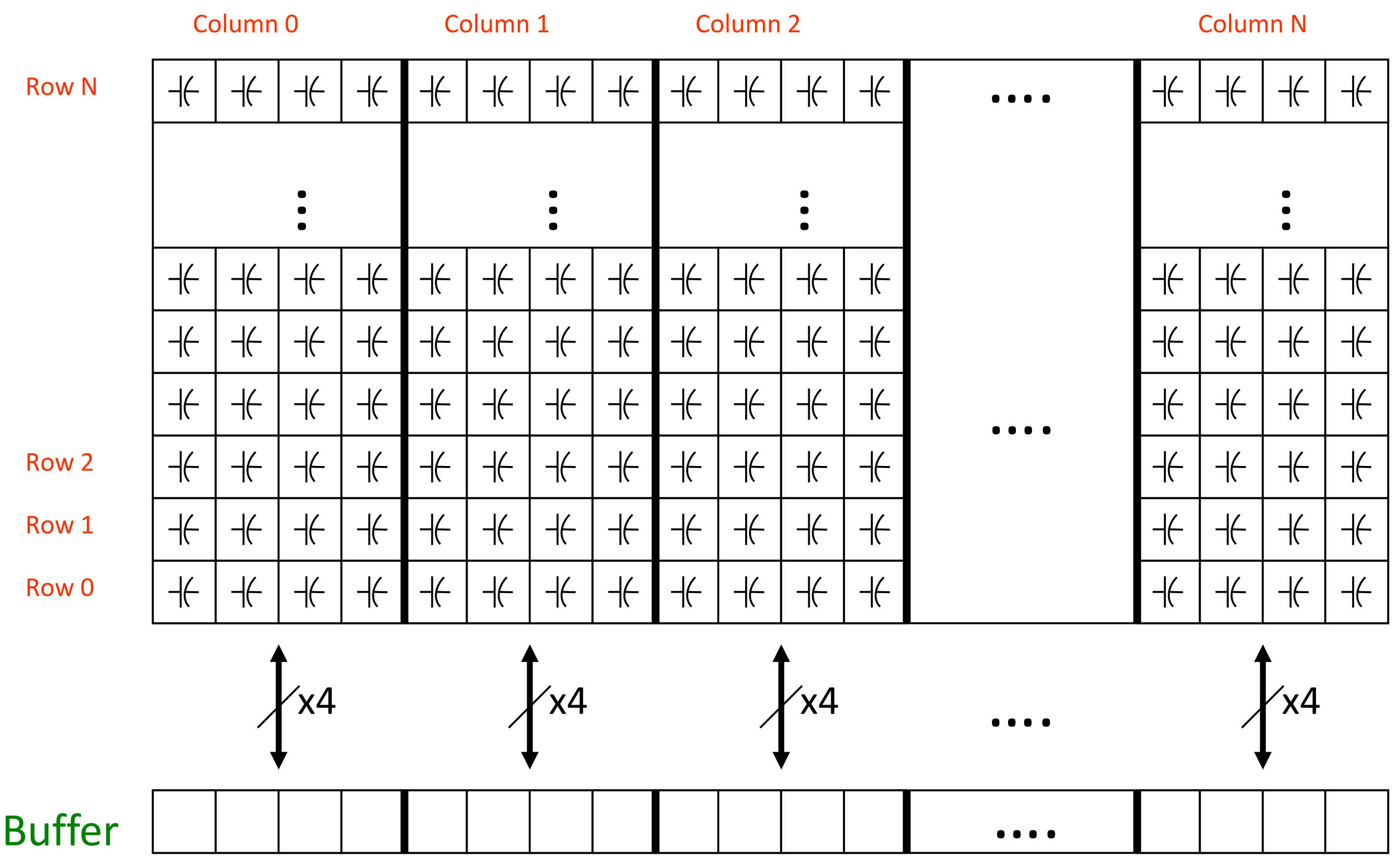
- Banks are independent



128Mb Half Bank Includes
32 SA Bands and
33 4Mb sub-arrays.

DRAM Internal Abstraction – Page of 8K

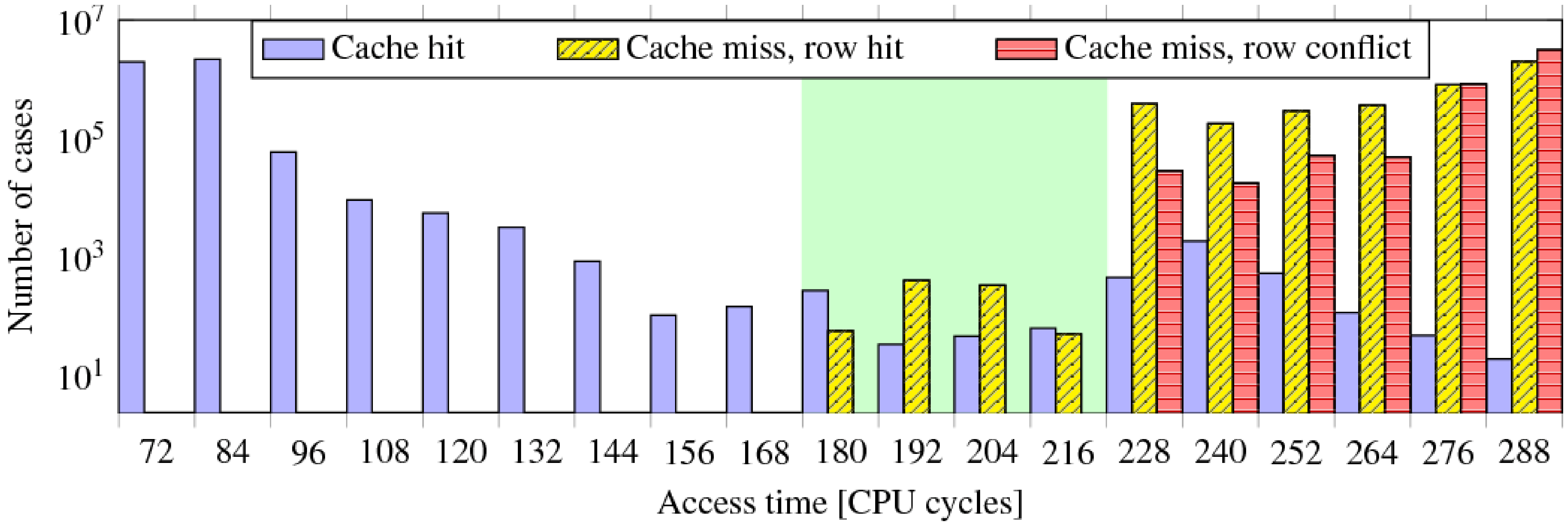
- ACT – activate a row/page (tRAS)
- Read a portion of bits out (burst)
- Page policy
  - Close page
  - Open page
    - Locality
- Access to same bank slow
- Access to different banks
  - faster



Page Size = number of bits

Access Time Disparity Due to Row Buffer [PGM16]

- **Behavior of memory access time**
  - **Row hits – lower latency (180-216 cycles no row conflicts)**
  - **Row miss/conflicts – higher latency**



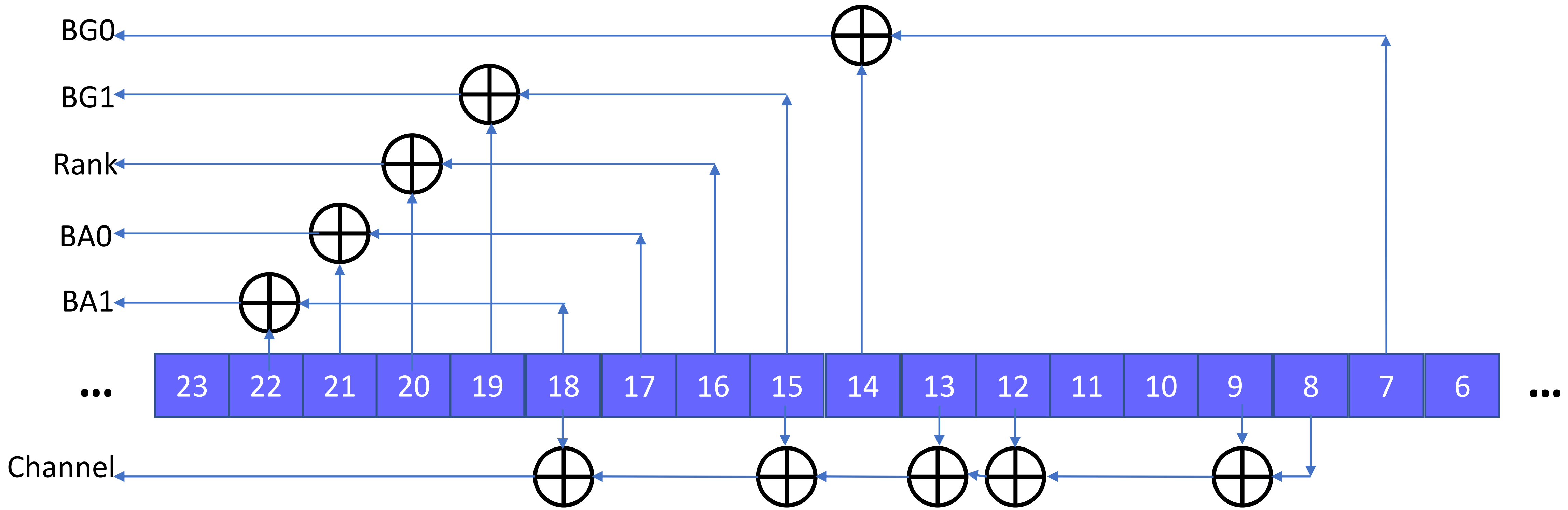
P. Pessl et. al, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," 25th USENIX Sec Symp, 2016

Reverse-Engineering DRAM Addressing (1)

- Reconnaissance
  - Memory controller (MC) uses physical address
  - Need to determine the physical address mapping
    - CPU vendors (MC) do not disclose mapping functions
      - As an advantage to competitor
      - Security enhancement
  - Assume linear equation
- Actual procedure
  - Download SW from github which generate random addresses and time access latency
  - Investigate HW system parameters
    - DRAM DIMM
    - Channel
    - Rank
    - Banks

Reverse Engineer Address Mapping (2)

- **Experiment on my Dell OptiPlex 7760 aoi**
  - **32GB DRAM – 2 DIMMs/2 Ranks/1 Channel/DDR4**
  - **(7, 14), (15, 19), (16, 20), (17, 21), (18, 22), (8, 9, 12, 13, 15, 18)**
- **HW/Project**
  - **Github download -https://github.com/IAIK/drama**



Outlines

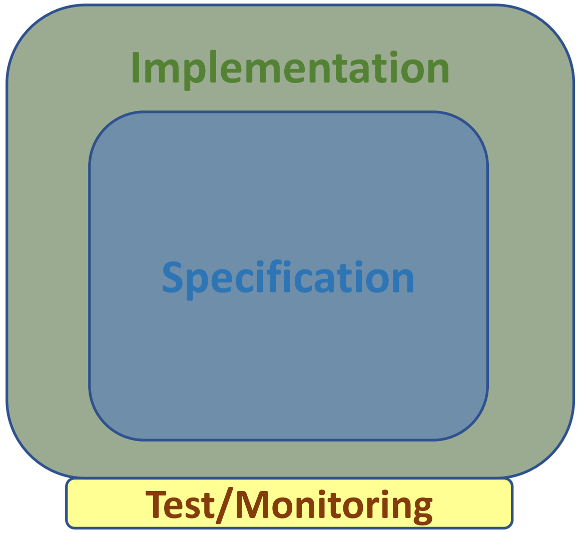
- Security goal and challenge of securing systems
- Common attacks and side-channels
  - Physical side-channels
- Microarchitecture side-channels and …
  - Examples
    - Cache side-channel
    - Main memory side-channel
    - Speculation state side-channel
  - Some recent publications
- What can we do?
  - Static analysis
  - Dynamic monitoring
- Summary

Speculation to Enhance Performance

- **IPC (or CPI) - How to do more per clock?**
  - **Reducing memory delays → Caches**
  - **Working during delays → Speculative execution to increase parallelism**
- **Example of speculation**

```
if (uncached_value_usually_1 == 1)
      foo()
```

  - **Branch predictor**
  - **Assume result of "if()" is 'true' (based on prior history)**
  - **CPU starts foo() execution speculatively -- but doesn't commit changes**
  - **When value arrives from memory and condition of "if()" is evaluated**
    - **Correct: Commit speculative work – performance gain**
    - **Incorrect: Discard speculative work**



Vulnerability in Transient State

Software security requirement - CPUs runs instructions correctly

Does making + discarding mistakes violate this assumption?

Regular execution

Set up the conditions so the processor will make a desired speculation mistake

Fetch the sensitive data from the covert channel

Erroneous speculative execution

Mistake leaks sensitive data into a covert channel (e.g. state of the cache)

Kocher, "Spectre Attacks: Exploiting Speculative Execution," https://arxiv.org/abs/1801.01203

Conditional Branch (Variant 1) Attack Example [Koc 18]

- **Array bound checking**

```
if (x < array1_size)
   y = array2[array1[x]*4096];
```

- **Assume code segment above is in kernel API**
  - **Unsigned *int x* comes from untrusted caller**

- **Execution without speculation is safe**
  - **CPU will not evaluate *array2[array1[x]*4096]* unless *x* < *array1_size***

- **What about with speculative execution?**

Kocher, "Spectre Attacks: Exploiting Speculative Execution," https://arxiv.org/abs/1801.01203

Conditional Branch (Variant 1) Attack (Spectre) [Koc 18]

- **Array bound checking**

```
if (x < array1_size)
   y = array2[array1[x]*4096];
```

- **Before attack:**
  - **Train branch predictor to expect *if()* is true (e.g. call with *x < array1_size*)**
  - **Evict *array1_size* and *array2[]* from cache**

Memory & Cache Status

`array1_size  = 00000008`

Memory at `array1` base address:

8 bytes of data (value doesn't matter)

[… lots of memory up to `array1` base+N…]

`02 F1 98 CC 90...` (something secret)

```
array2[ 0*4096]
array2[ 1*4096]
array2[ 2*4096]
array2[ 3*4096]
array2[ 4*4096]
array2[ 5*4096]
array2[ 6*4096]
array2[ 7*4096]
array2[ 8*4096]
array2[ 9*4096]
array2[10*4096]
array2[11*4096]
     ...
```

Contents don't matter only care about cache ***status***

🟪 Uncached 🟩 Cached

Kocher, "Spectre Attacks: Exploiting Speculative Execution," https://arxiv.org/abs/1801.01203

Conditional Branch (Variant 1) Attack (Spectre)

- **Array bound checking**

```
if (x < array1_size)
    y = array2[array1[x]*4096];
```

- **Attacker calls victim with *x=9 (>8)***
  - **Speculative exec - waiting for *array1_size* (not in $)**
    - **Predict that *if()* is true**
    - **Read address *(array1 base + x)* with out-of-bounds *x (speculatively)***
    - **Read returns secret byte = *02* (fast – in cache)**
    - **Request memory at *(array2 base + 02*4096)***
    - **Brings *array2[02*4096]* into the cache**
    - **Realize**
      work
  - **Finish ope**

Memory & Cache Status

```
array1_size = 00000008
```

Memory at `array1` base address:

8 bytes of data (value doesn't matter)

[… lots of memory up to *`array1` base+N*…]

`02 F1 98 CC 90...` (something secret)

```
array2[ 0*4096]
array2[ 1*4096]
array2[ 2*4096]
array2[ 3*4096]
array2[ 4*4096]
array2[ 5*4096]
array2[ 6*4096]
array2[ 7*4096]
array2[ 8*4096]
array2[ 9*4096]
array2[10*4096]
```

Contents don't matter
only care about cache ***status***

d   Cached

Attacker measures read time for *`array2[i*4096]`*

- **Read for `i`=02 is fast (cached), revealing secret byte**
- **Repeat with many `x` (eg ~10KB/s)**

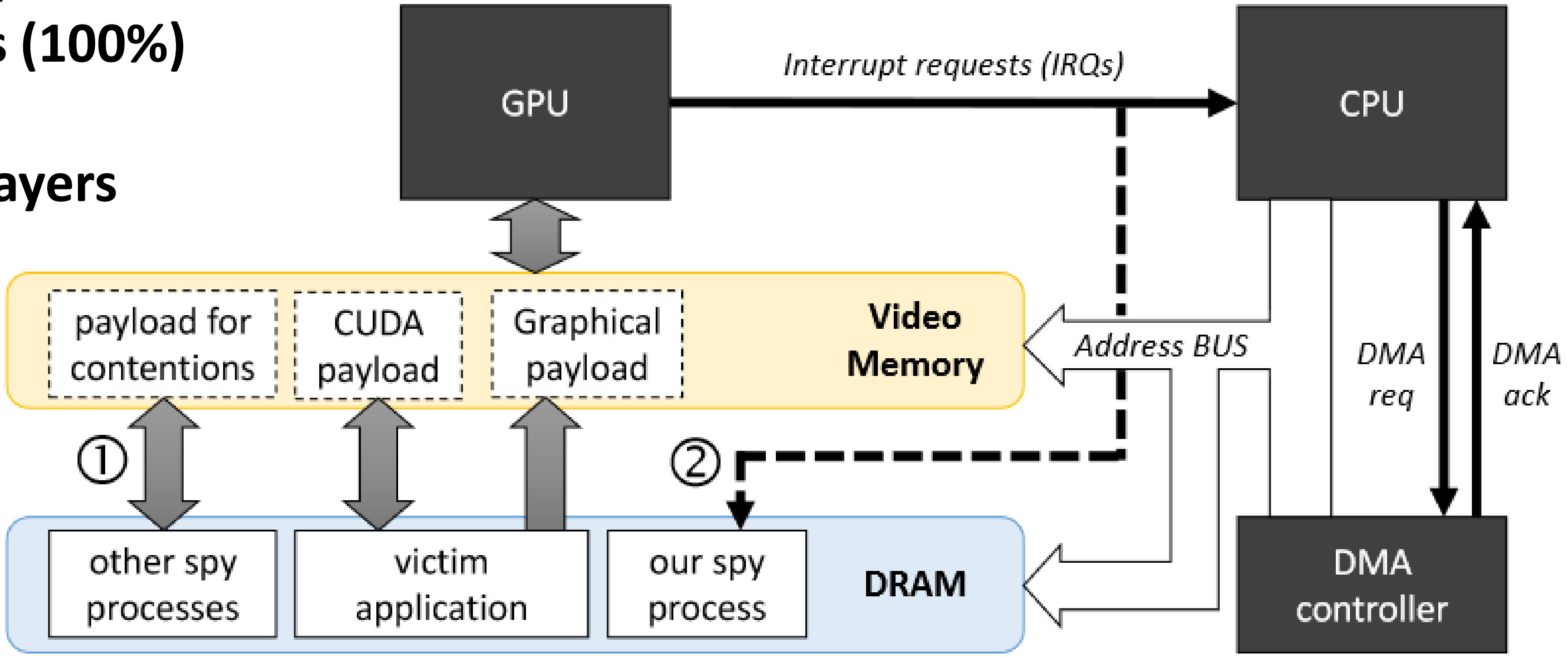
ution," https://arxiv.org/abs/1801.01203

Outlines

- Security goal and challenge of securing systems
- Common attacks and side-channels
  - Physical side-channels
- Microarchitecture side-channels and …
  - Examples
    - Cache side-channel
    - Main memory side-channel
    - Speculation state side-channel
  - Some recent publications
- What can we do?
  - Static analysis
  - Dynamic monitoring
- Summary

Interrupt Requests Side-Channels [Ma 2015]

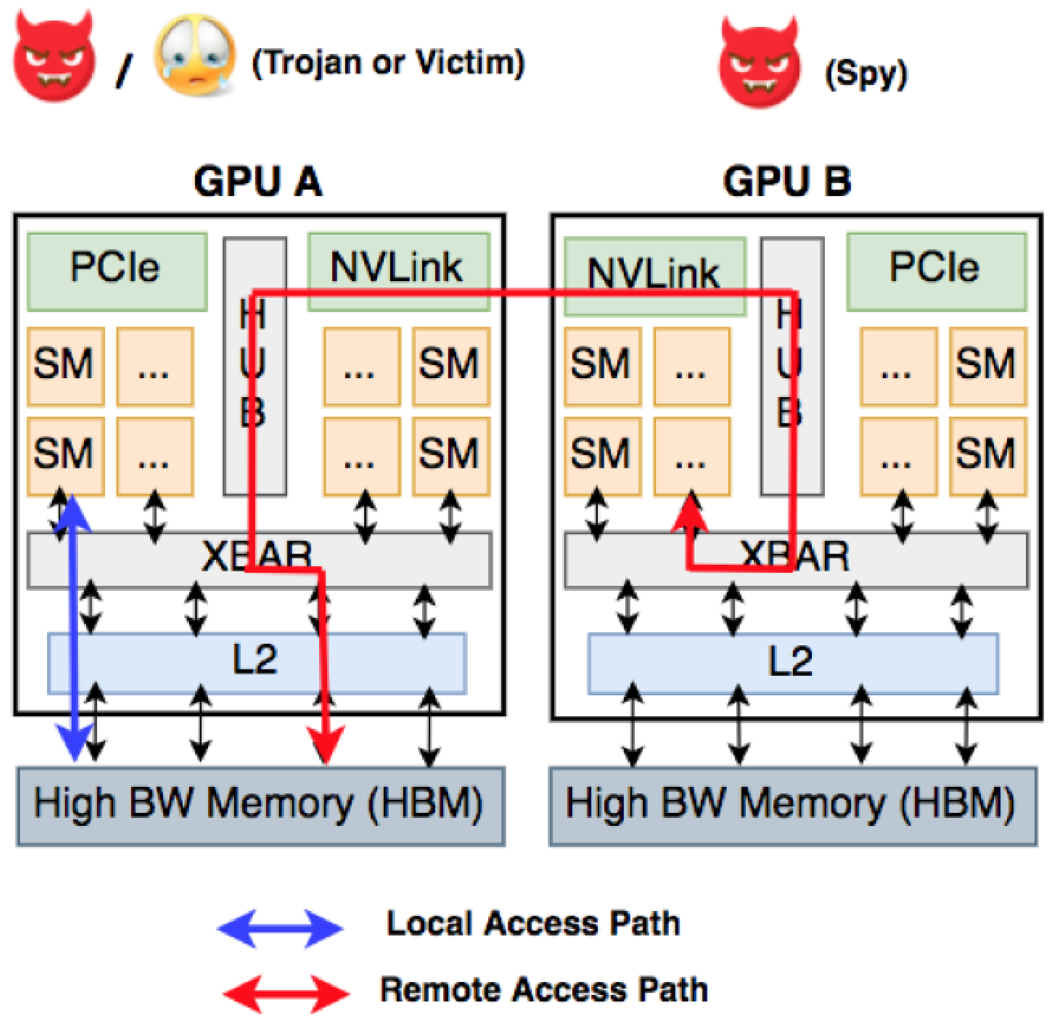
- No active contented process - ①
- Passive reading aggregated graphics interrupt counts (OS) - ②
  - Infer from statistics collected to predict GPU tasks (ML)
  - Events (prediction rate)
    - GUI apps (99.95%)
    - GPGPU workloads (100%)
    - Webpage visits
    - Different video players
    - PDF documents



On the Effectiveness of Using Graphics Interrupt as a Side Channel for User Behavior Snooping, IEEE T. on Dep & Sec computing, v. 19, 2022
https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=7752&context=sis_research

Spy In the GPU-box [Dut 23]

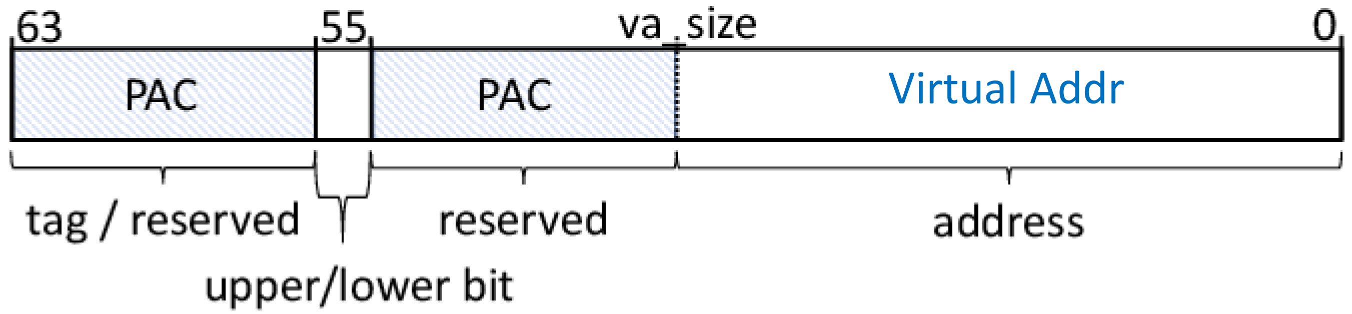
- Heterogeneous system
  - GPU + CPU
- Covert channel between GPUs
- Need to reverse engineer GPU cache

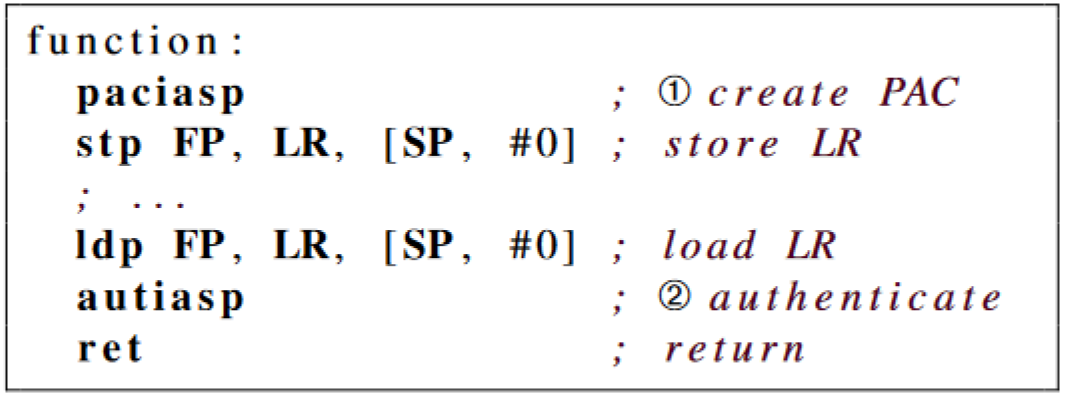


S. Dutta et. al., Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems, ISCA 2023

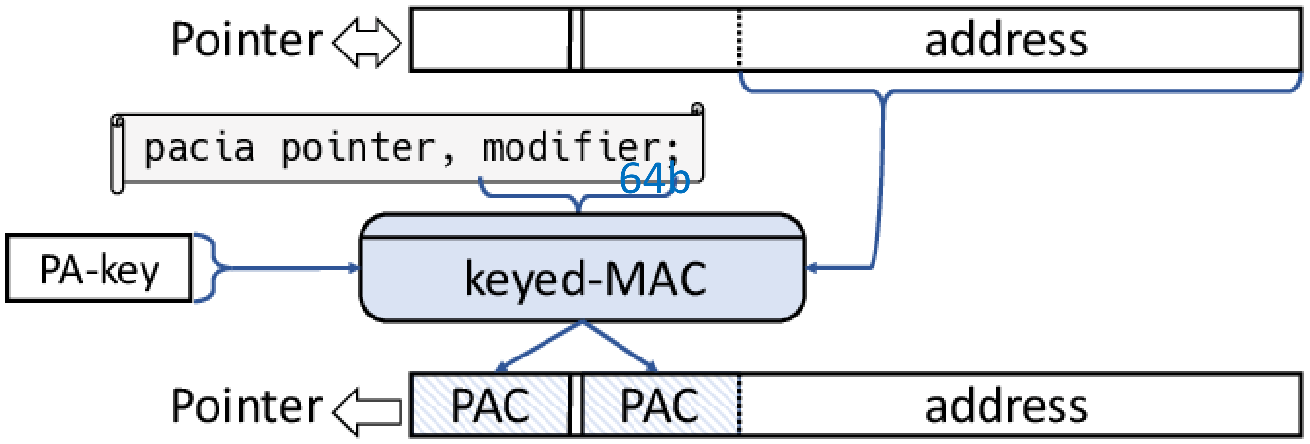
Security Enhancement Structure - PAC Vulnerability

- **What is PAC (Pointer Authentication Codes)?**
  - **A "security" feature from ARM to guard against pointer integrity**
    - **ARMv8.3-A**
    - **Unused address bits**
    - **Dynamically instrumented**
      - **Add PAC (pacia__)**
      - **Authenticate (autia__)**
  - **Usage example – func call**
    - **Entry – create a PAC in LR**
    - **Exit – check LR**



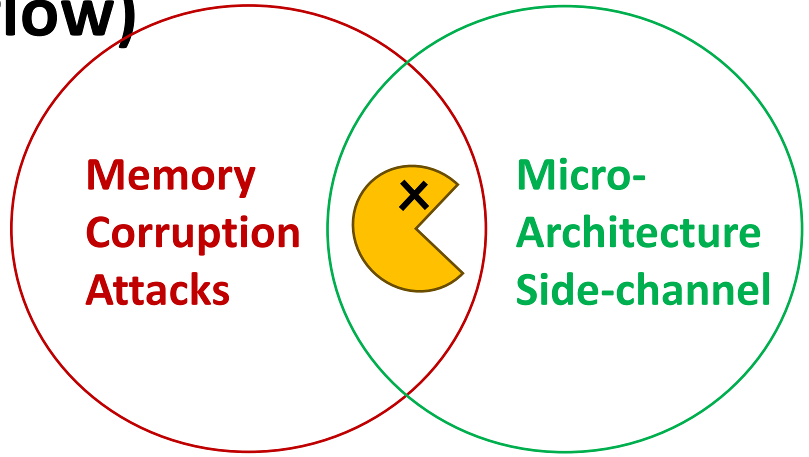


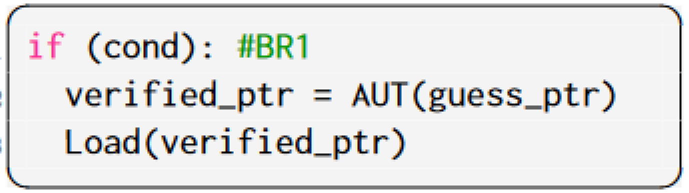
```
function:
  paciasp                  ; ① create PAC
  stp FP, LR, [SP, #0]     ; store LR
  ; ...
  ldp FP, LR, [SP, #0]     ; load LR
  autiasp                  ; ② authenticate
  ret                      ; return
```



Apple M1 PACMAN Attack [Rav 22]

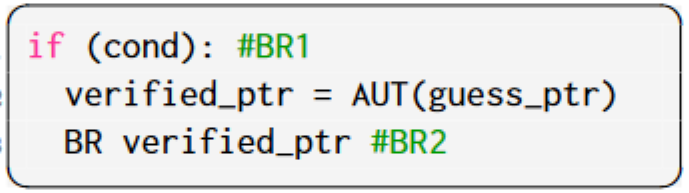
- Leveraging microarchitecture side-channel & memory corruption
  - PAC protects illegal memory access (buffer overflow)
    - Verify the signature (hash)
    - If different system halts
  - But … there is the speculation state!
    - Guess the hash in speculation state





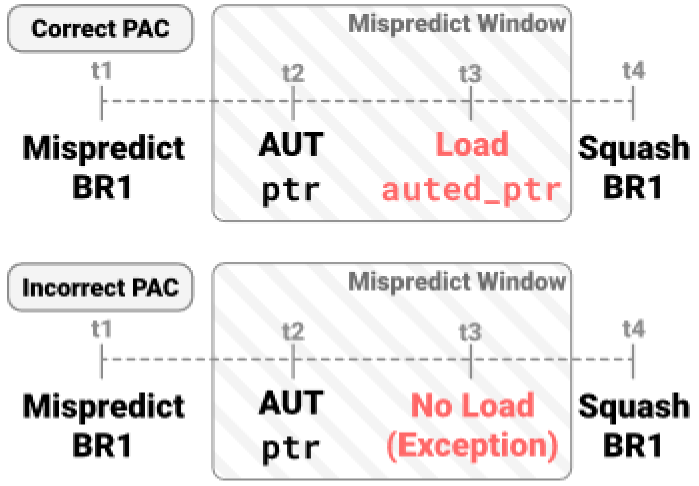
```
if (cond): #BR1
  verified_ptr = AUT(guess_ptr)
  Load(verified_ptr)
```

(a) A data PACMAN gadget.

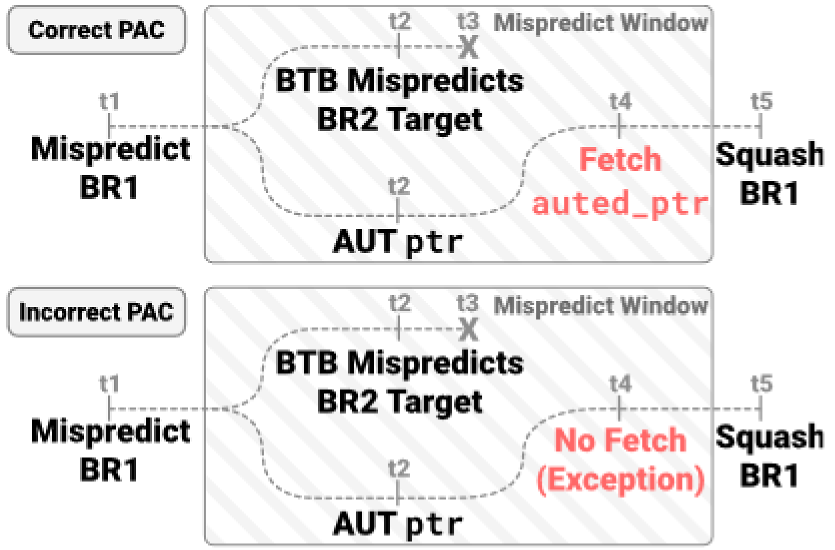


```
if (cond): #BR1
  verified_ptr = AUT(guess_ptr)
  BR verified_ptr #BR2
```

(b) An instruction PACMAN gadget.



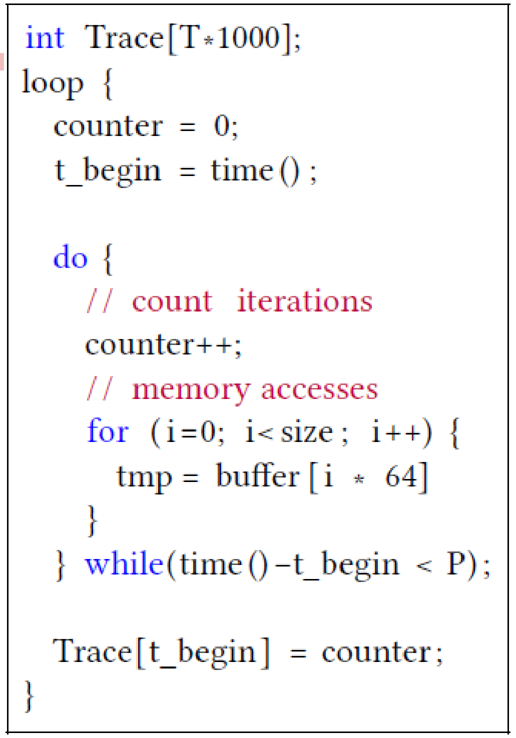
(c) Timeline for data access leak.



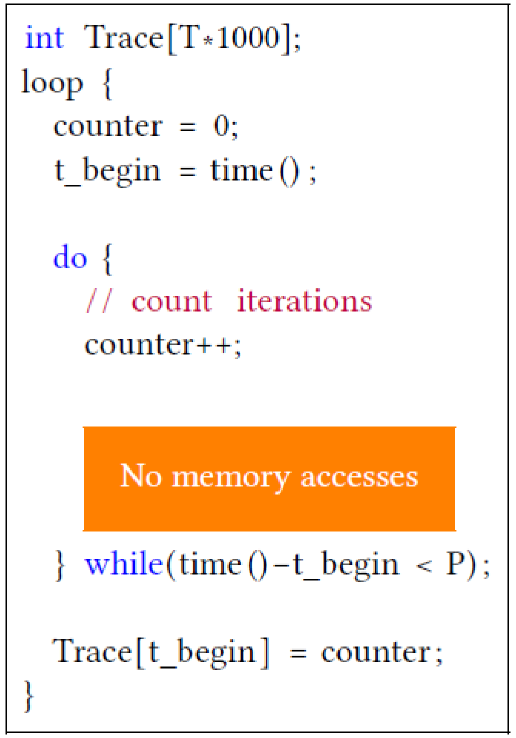
(d) Timeline for instruction fetch leak.

There is a Bigger Fish [Coo 22]

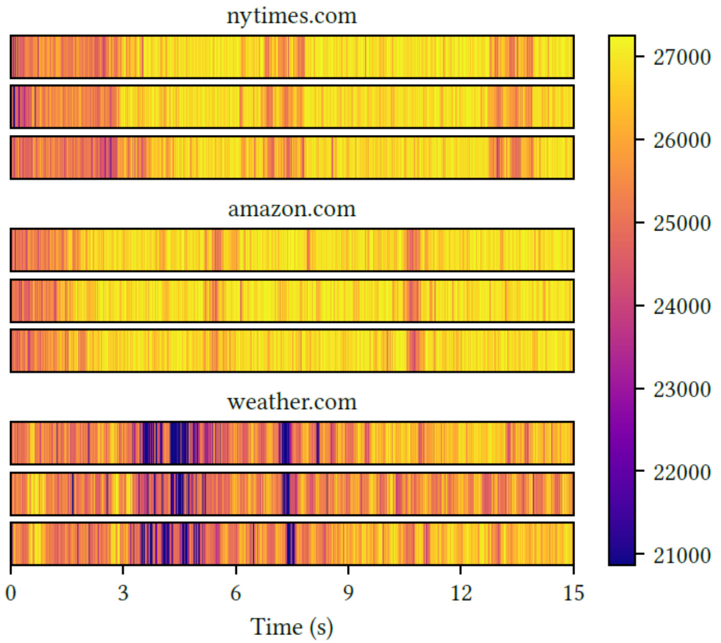
- What is the true side channel?
- Website fingerprinting example
  - Previous publication pointed to cache
  - Scientific approach
    - Hypothesis
    - Verify
  - Turns out to be interrupts

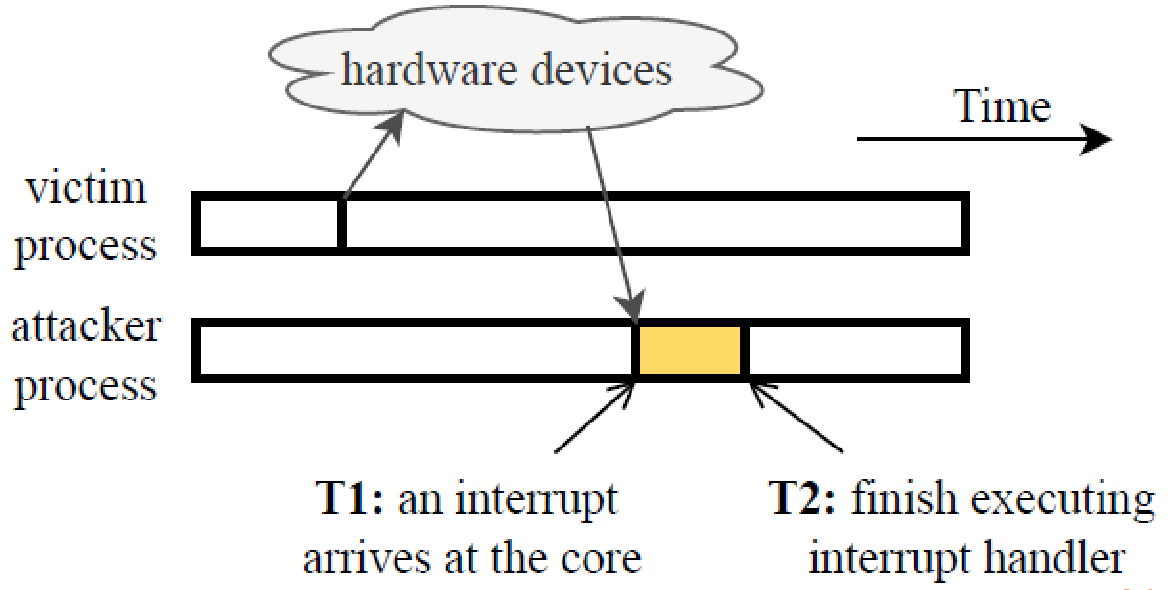


```
int Trace[T*1000];
loop {
  counter = 0;
  t_begin = time();

  do {
    // count iterations
    counter++;
    // memory accesses
    for (i=0; i<size; i++) {
      tmp = buffer[i * 64]
    }
  } while(time()-t_begin < P);

  Trace[t_begin] = counter;
}
```



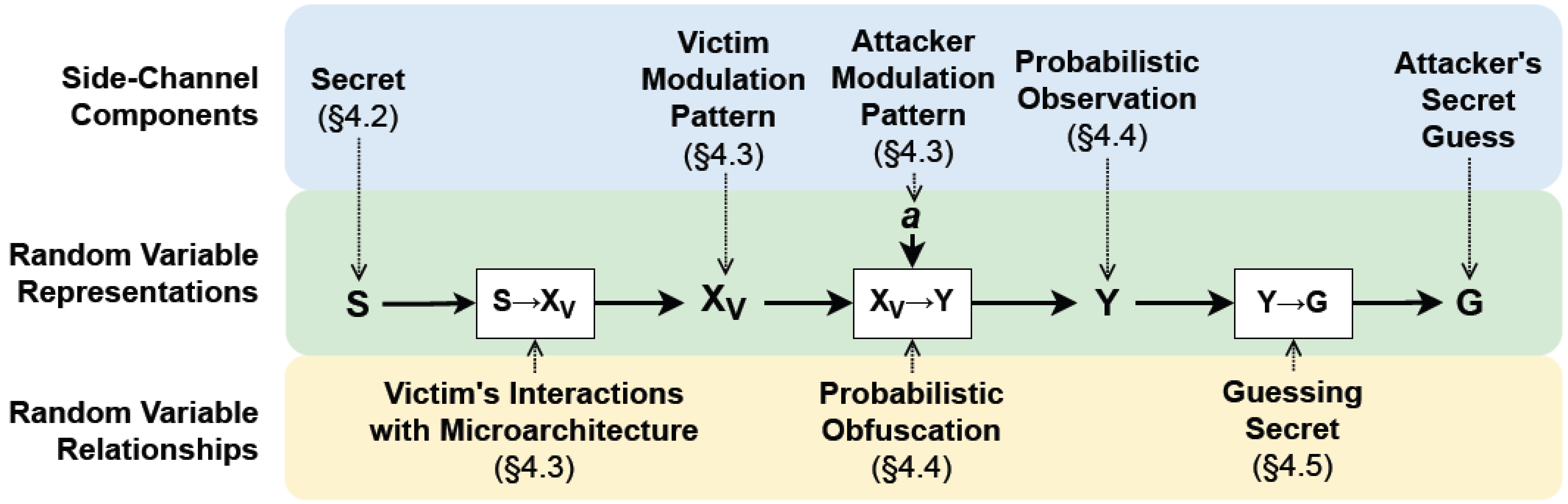
```
int Trace[T*1000];
loop {
  counter = 0;
  t_begin = time();

  do {
    // count iterations
    counter++;

      No memory accesses

  } while(time()-t_begin < P);

  Trace[t_begin] = counter;
}
```




hardware devices
Time
victim process
attacker process
T1: an interrupt arrives at the core
T2: finish executing interrupt handler

Metior: Evaluate Obfuscating Defense Schemes

- It is an arm race
  - A weakness is discovered
  - Methods proposed to block the channel
  - New weakness is found (even on the methods proposed)
- A formal model


Side-Channel Components
Random Variable Representations
Random Variable Relationships
Secret (§4.2)
Victim Modulation Pattern (§4.3)
Attacker Modulation Pattern (§4.3)
Probabilistic Observation (§4.4)
Attacker's Secret Guess
a
S
S→X_V
X_V
X_V→Y
Y
Y→G
G
Victim's Interactions with Microarchitecture (§4.3)
Probabilistic Obfuscation (§4.4)
Guessing Secret (§4.5)

Outlines

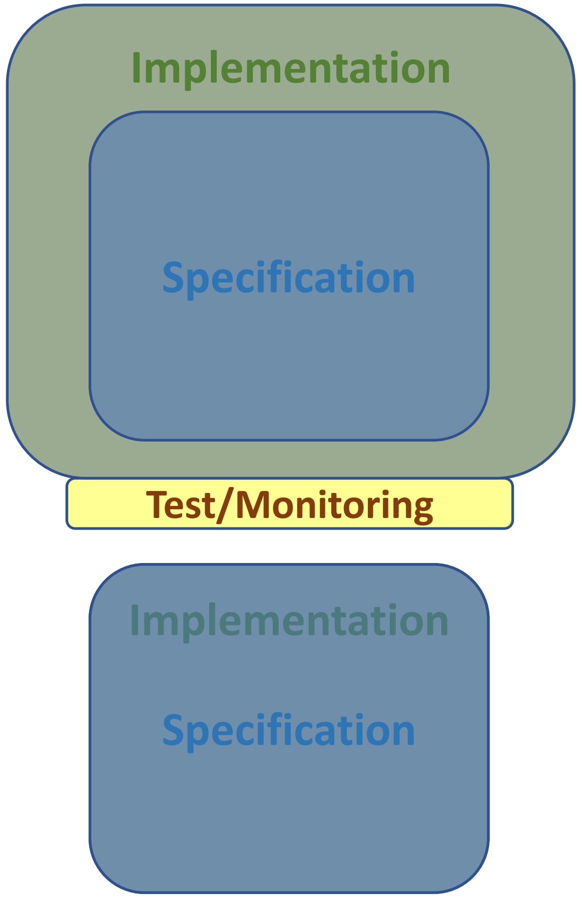
- Security goal and challenge of securing systems
- Common attacks and side-channels
  - Physical side-channels
- Microarchitecture side-channels and …
  - Examples
    - Cache side-channel
    - Main memory side-channel
    - Speculation state side-channel
  - Some recent publications
- **What can we do?**
  - **Static analysis**
  - **Dynamic monitoring**
- **Summary**

Classification

- Side-channels enabled by
  - Shared structures
  - Speculation
  - Performance counters
- Channel types
  - Persistent
    - States that persist until next events
  - Ephemeral
    - Transient states – attacker/victim must cohabit
- Attack types
  - Active
    - Affects victim
  - Passive

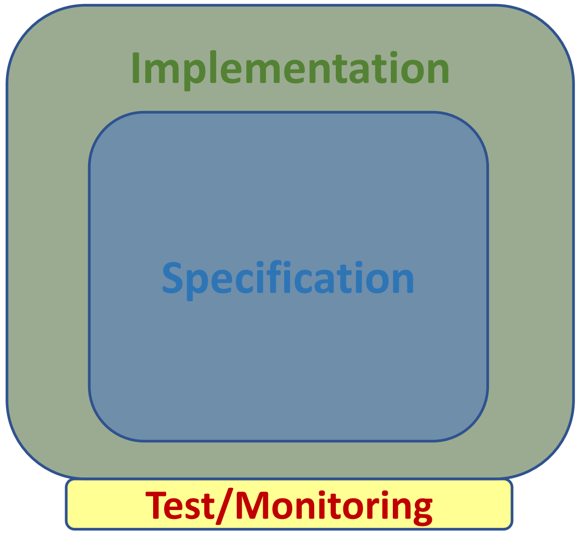
Mitigation

- Complete clean slate design
  - Compatibility challenge
  - Cost – formal verification?
- Mitigation
  - Partition
    - Strick isolation
    - No shared structures
      - Performance impact
  - Obfuscation
    - Add noise



Possible Obfuscation Approaches

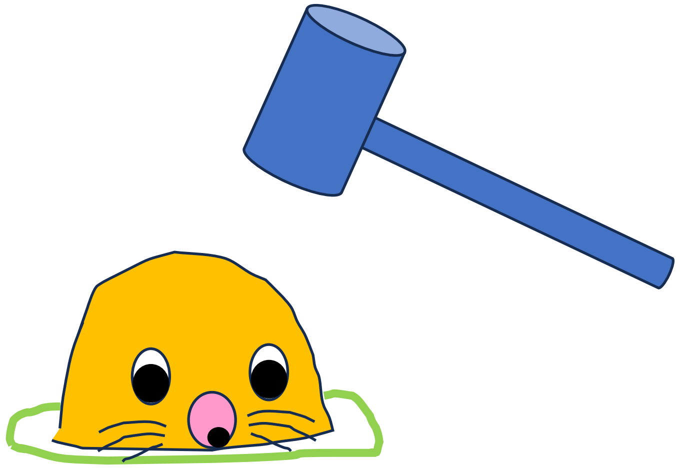
- Static analysis/data analytic
  - Given an implementation
  - Formalize all potential modulation
    - To mitigate one must find the modulation
    - Aid in reverse engineering
  - Build graph - evaluate
- Dynamic monitoring
  - Detection of active leakage
    - Patterns?
    - Does it need performance counters?
- Dynamic randomization - obfuscation
  - Source side
    - Intelligent fuzzing of software
  - System side
    - Insertion of disturbance
      - Maximize effect of noise while minimize effects on performance



https://web.eecs.umich.edu/~barisk/public/morpheus.pdf

Summary

- Security is important when there is no trust
- Security is challenging
- Microarchitecture side channel is particularly dangerous
  - No need for physical access
  - Against design goals
  - Increasing attack surfaces
- Possible approach
  - Static
  - Dynamic
  - Need efficient and general solution



Thank you!

Q&A