CS211
Advanced Computer Architecture

L19 Hardware Security

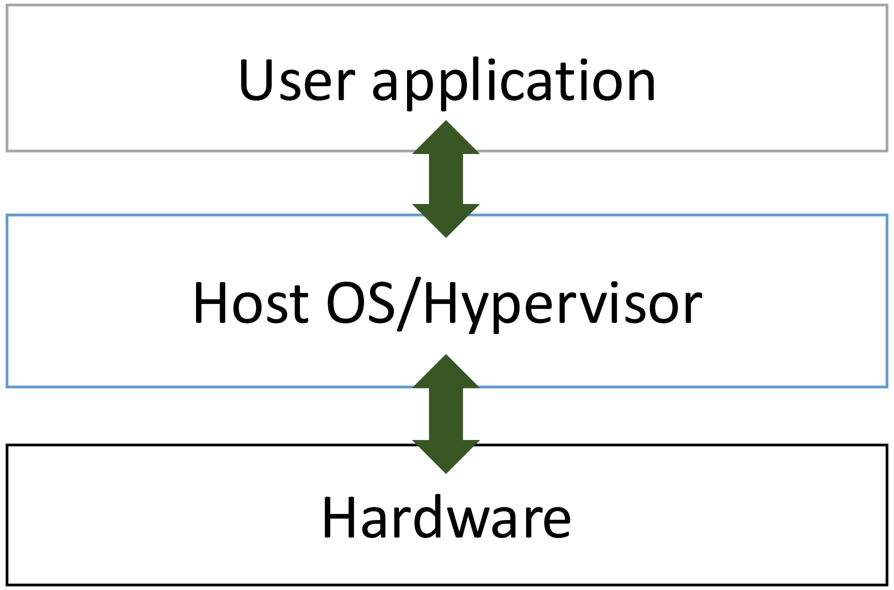
Chundong Wang

December 3rd, 2025

上海科技大学
ShanghaiTech University

Security breaches at hardware

- Cyber-attacks
  - In computers and computer networks an attack is any attempt to expose, alter, disable, destroy, steal or gain unauthorized access to or make unauthorized use of an asset. (ISO)

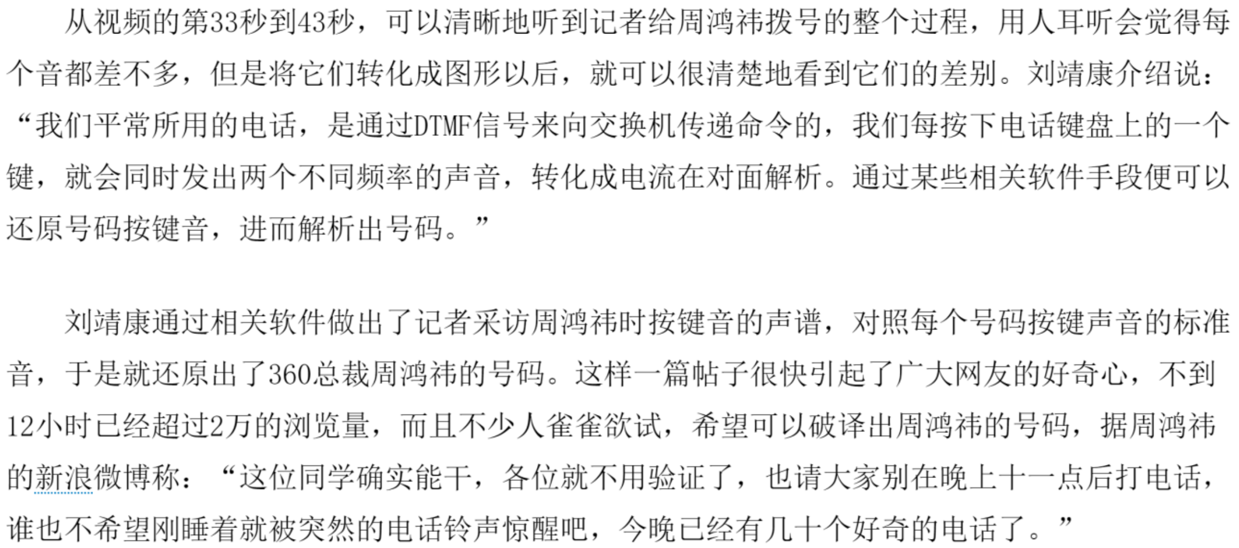


Side Channel Attacks

上海科技大学 ShanghaiTech University

Side channels are everywhere

- Example 1
  - In August 2012, an undergraduate student, Mr Liu Jingkang, from Nanjing University, figured out Mr Zhou Hongyi's cell phone number according to acoustic sounds of keystrokes captured in an *unanswered* phone-interview between a journalist and Mr Zhou.



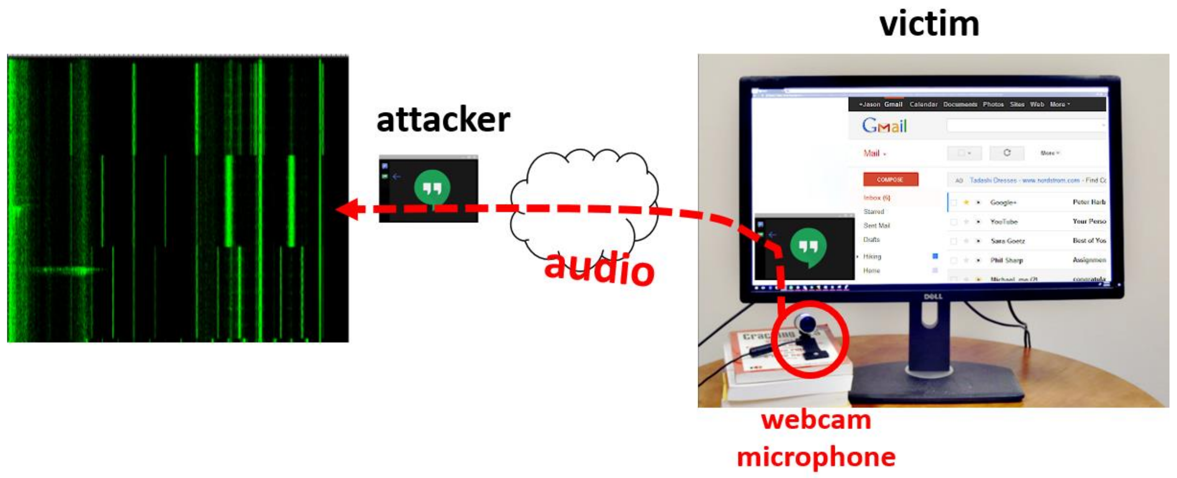
从视频的第33秒到43秒，可以清晰地听到记者给周鸿祎拨号的整个过程，用人耳听会觉得每个音都差不多，但是将它们转化成图形以后，就可以很清楚地看到它们的差别。刘靖康介绍说："我们平常所用的电话，是通过DTMF信号来向交换机传递命令的，我们每按下电话键盘上的一个键，就会同时发出两个不同频率的声音，转化成电流在对面解析。通过某些相关软件手段便可以还原号码按键音，进而解析出号码。"

刘靖康通过相关软件做出了记者采访周鸿祎时按键音的声谱，对照每个号码按键声音的标准音，于是就还原出了360总裁周鸿祎的号码。这样一篇帖子很快引起了广大网友的好奇心，不到12小时已经超过2万的浏览量，而且不少人雀雀欲试，希望可以破译出周鸿祎的号码，据周鸿祎的新浪微博称："这位同学确实能干，各位就不用验证了，也请大家别在晚上十一点后打电话，谁也不希望刚睡着就被突然的电话铃声惊醒吧，今晚已经有几十个好奇的电话了。"

From https://tech.qq.com/a/20120901/000028.htm

Side channels are everywhere

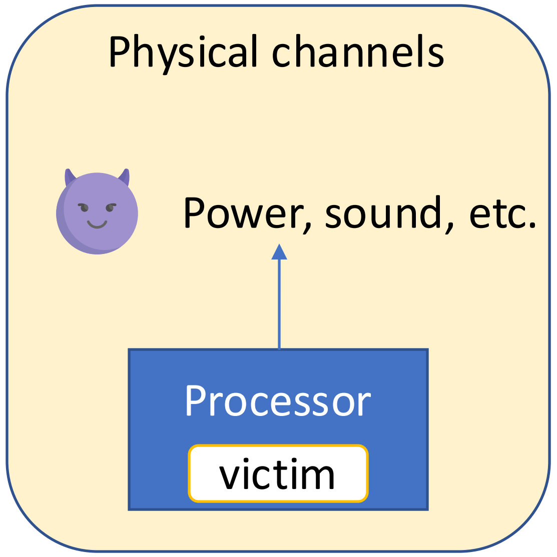
- Example 2
  - The **visual content** displayed on user's LCD screens leaks onto the faint **sound** emitted by the screens
  - *Synesthesia: Detecting Screen Content via Remote Acoustic Side Channels*, S&P '19

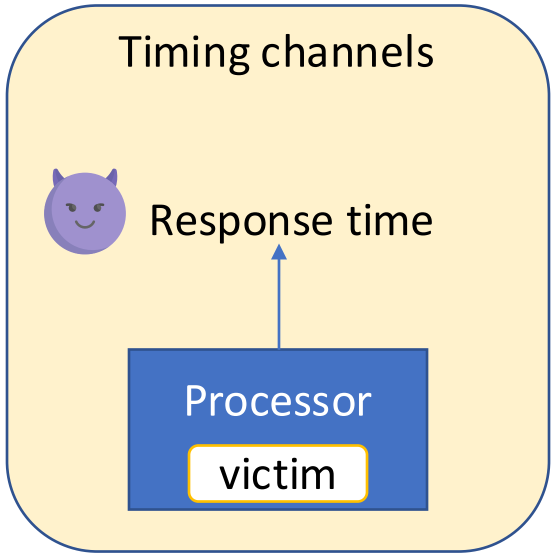


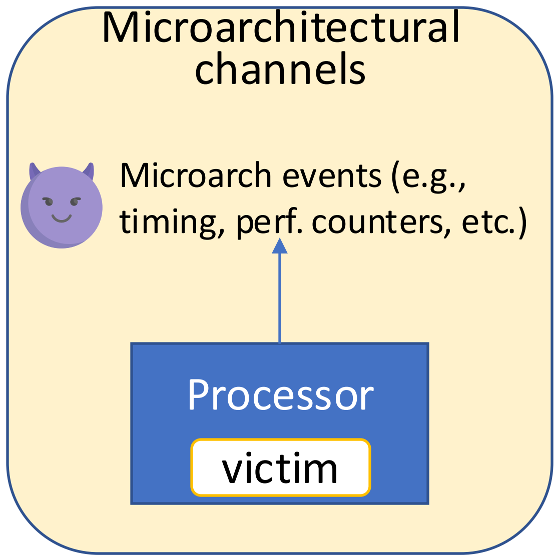
上海科技大学 ShanghaiTech University

Physical, timing, microarchitectural channels

- What can an adversary observe?






Microarchitectural channels
Microarch events (e.g., timing, perf. counters, etc.)
Processor
victim

Attacker requires measurement equipment → physical access

Attacker may be remote (e.g., over an internet connection)

Attacker may be remote, or be co-located

A typical victim application: RSA

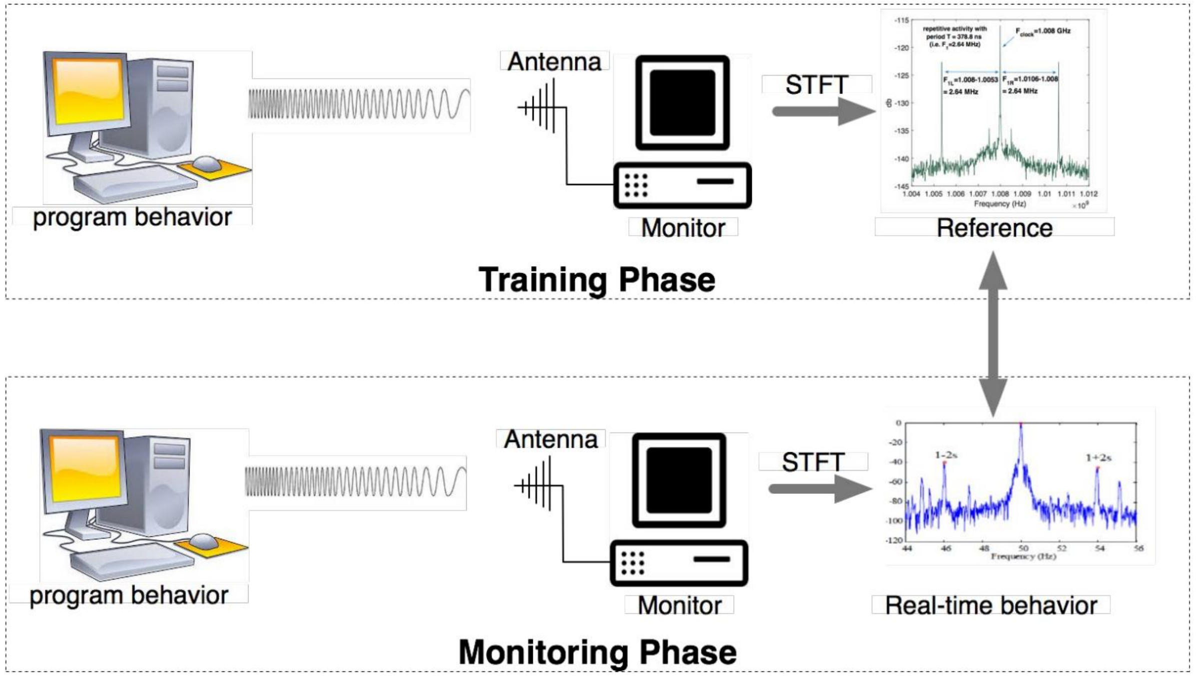
- Square-and-multiply based exponentiation

```
Input: base $b$, modulo $m$, exponent $e = (e_{n-1} \ldots e_0)_2$
Output: $b^e$ mod $m$
$r = 1$
for $i = n-1$ down to 0 do
        $r = (r * r)$ mod $m$
        if $e_i$ == 1 then
                $r = (r * b)$ mod $m$
        end if
end for
return $r$
```

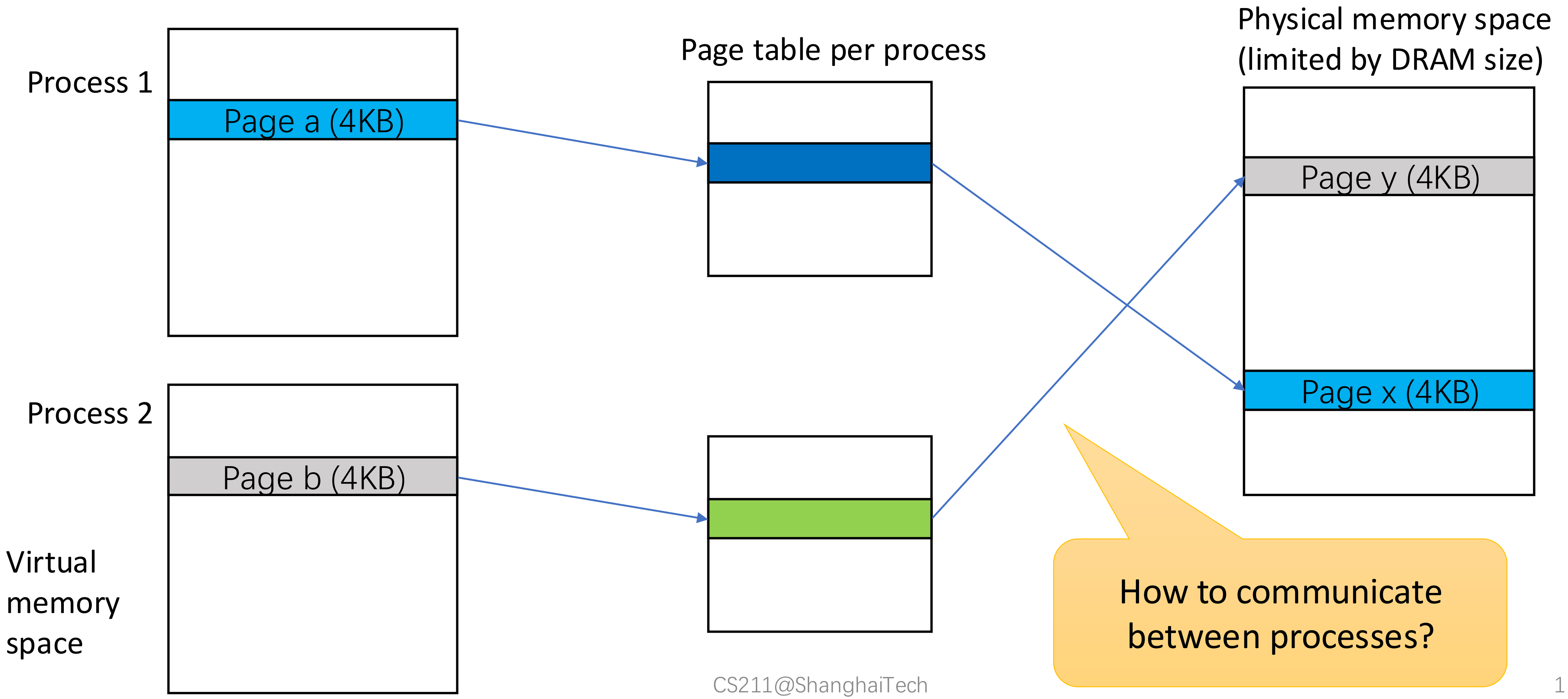
Assume that we measure time or power, can we figure out e_i?

Benign usage: non-intrusive software monitoring

- How to efficiently monitor application for anomaly detection?
- "Repetitive program activity (e.g. a loop) causes the unintentional EM signals to exhibit periodicity, i.e. the spectrum of these EM signals will have 'spikes' at frequencies that correspond to the time spent in each repetition of the program activity."
  - *Spectral Profiling: Observer-Effect-Free Profiling by Monitoring EM Emanations*, Micro '16



Recap: virtual memory for isolation



How to communicate between processes?

Normal cross-process communication

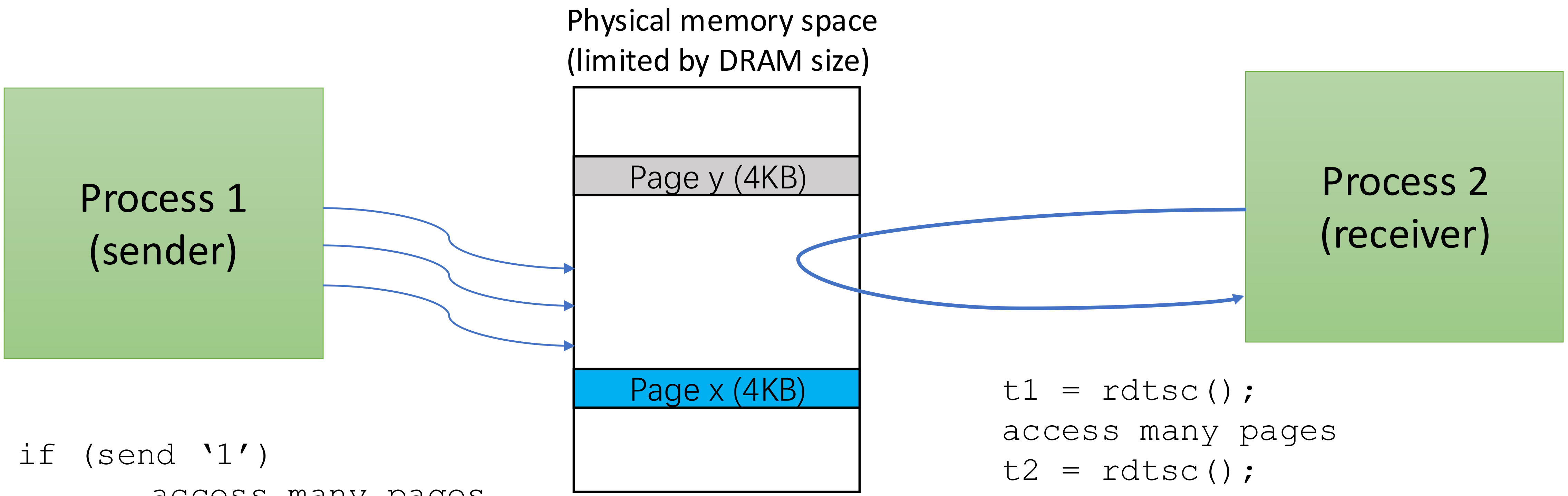
```
#include <socket.h>

void send (bit msg) {
        socket.send(msg);
}

bit recv() {
        return socket.recv(msg);
}
```

How to communicate **without** letting OS know?

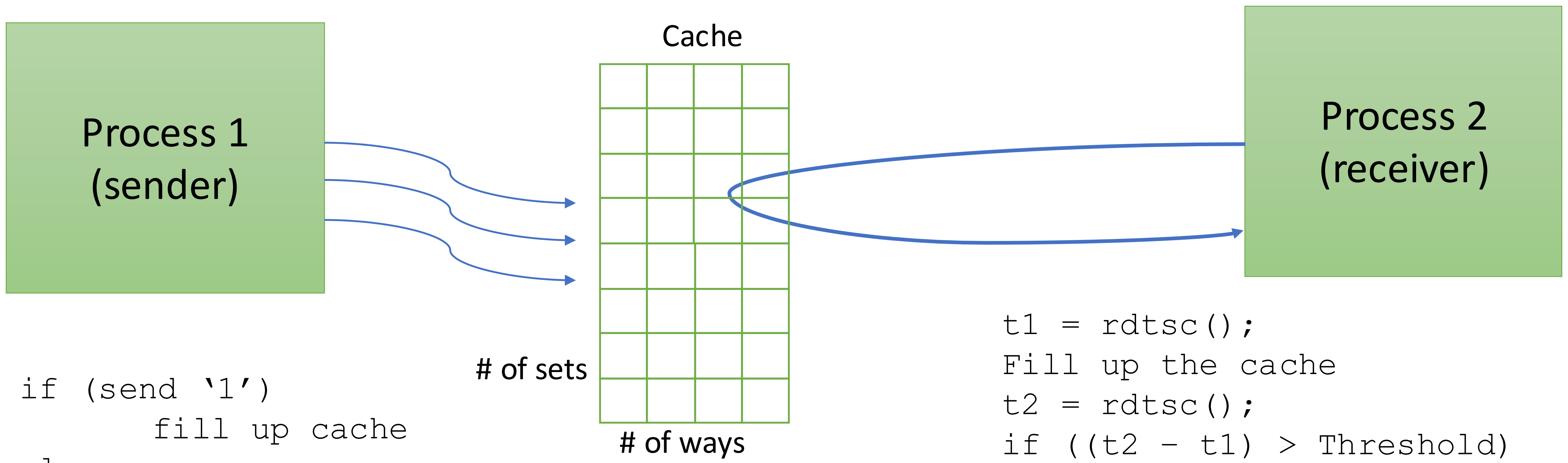
Solution 1: through the page fault



```
if (send '1')
        access many pages
else
        ; // idle
```

```
t1 = rdtsc();
access many pages
t2 = rdtsc();
if ((t2 - t1) > Threshold)
        read '1'
else
        read '0'
```

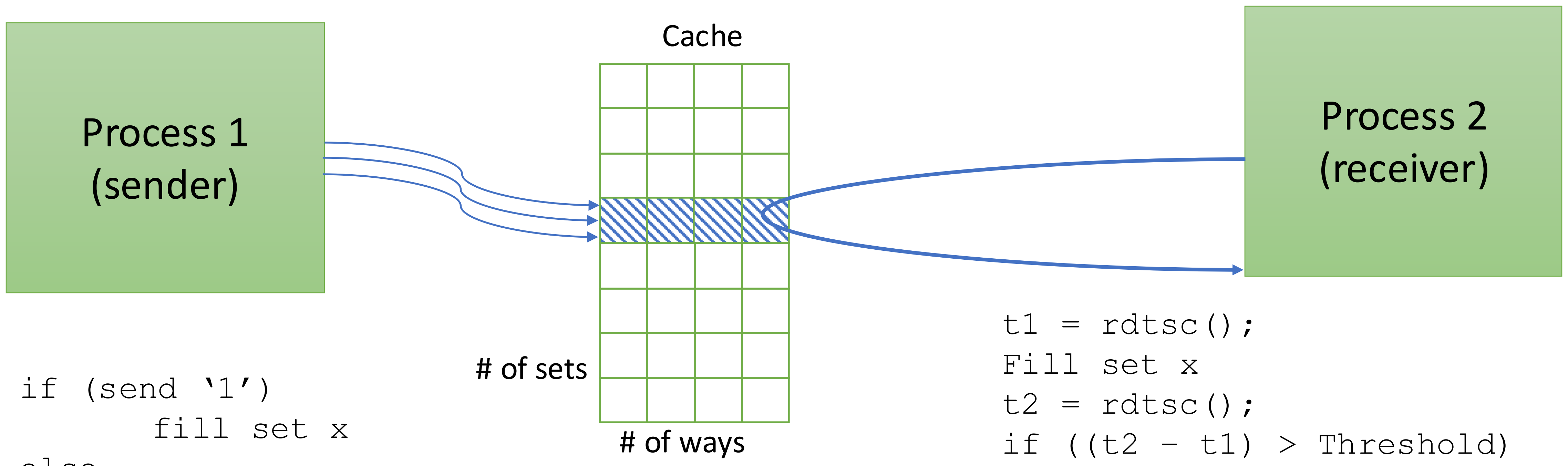
Solution 2: through the cache



```
if (send '1')
        fill up cache
else
        ; // idle
```

```
t1 = rdtsc();
Fill up the cache
t2 = rdtsc();
if ((t2 - t1) > Threshold)
        read '1'
else
        read '0'
```

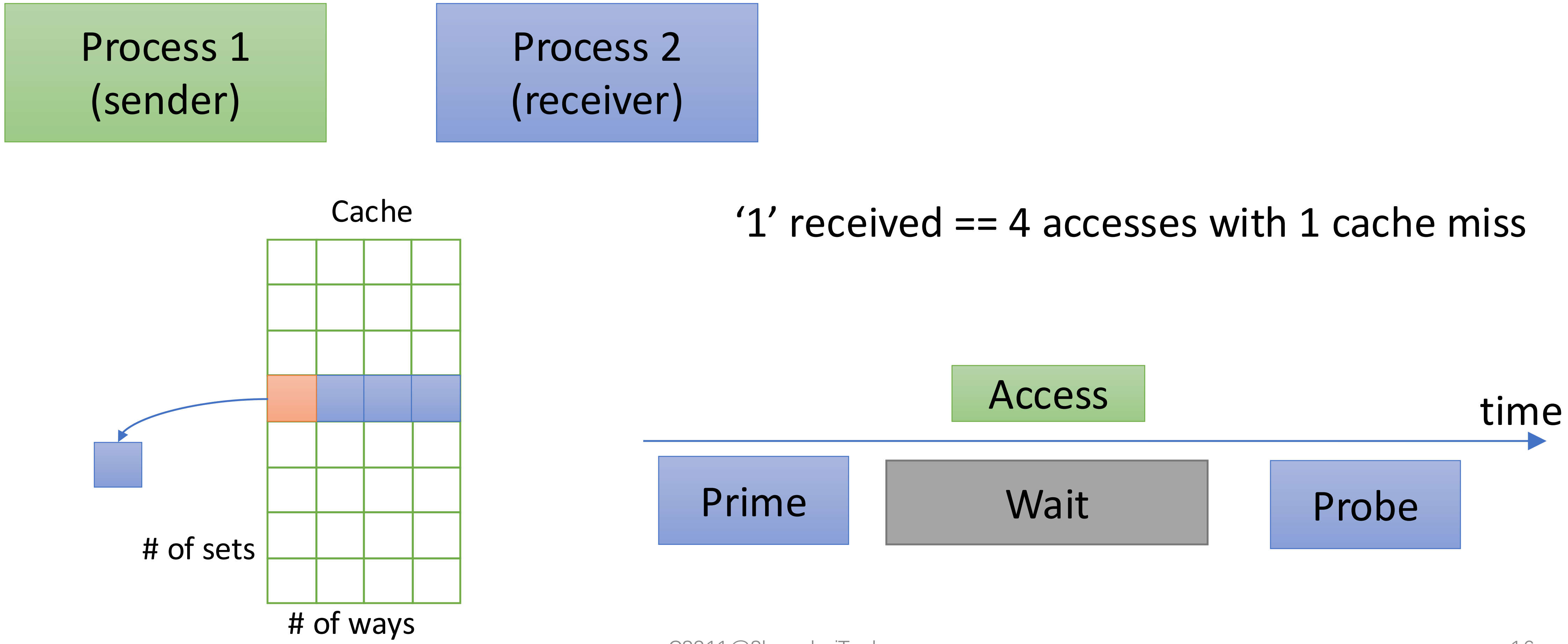
Solution 3: through a part of cache



```
if (send ‘1’)
        fill set x
else
        ; // idle
```

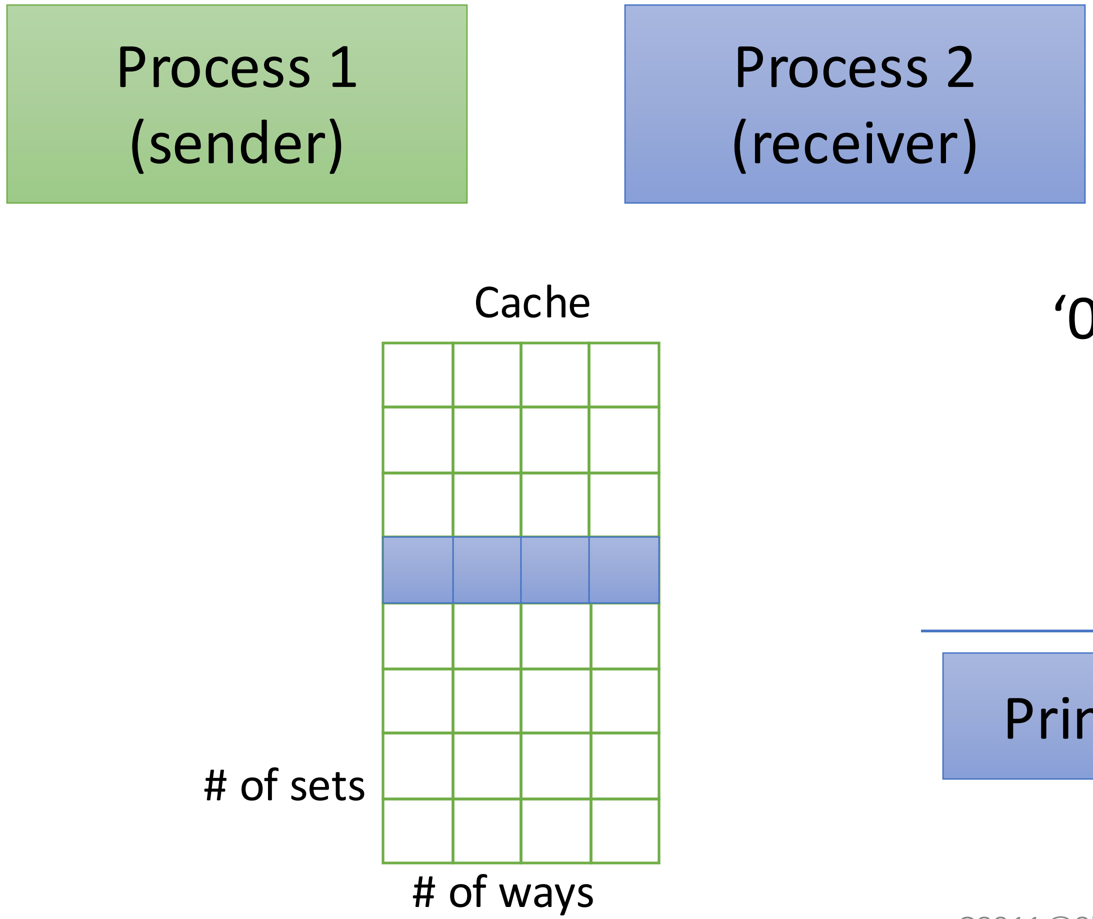
```
t1 = rdtsc();
Fill set x
t2 = rdtsc();
if ((t2 - t1) > Threshold)
        read ‘1’
else
        read ‘0’
```

Prime+Probe sends and receives ‘1’

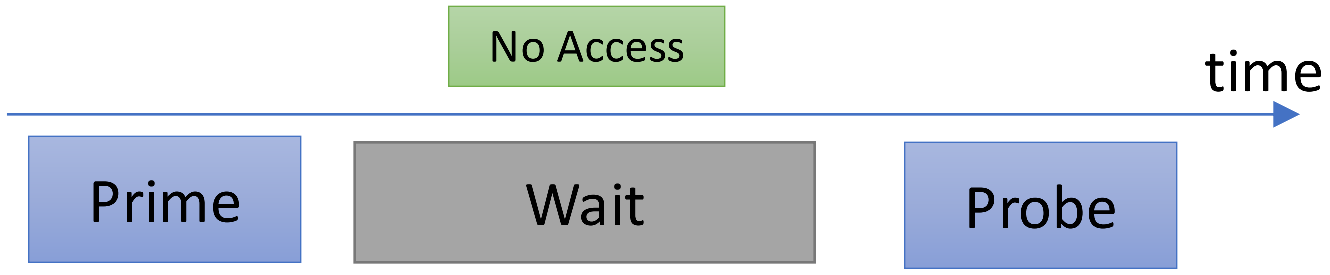


‘1’ received == 4 accesses with 1 cache miss

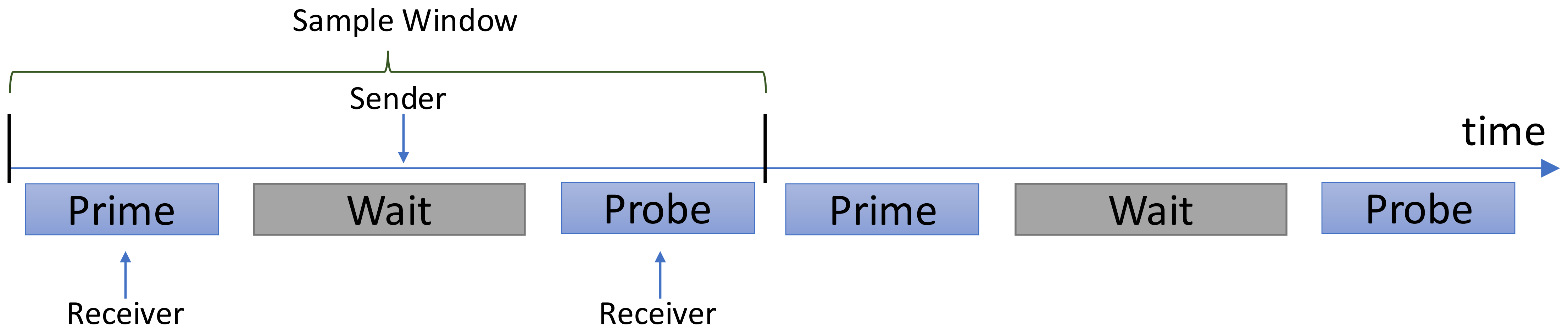
Prime+Probe sends and receives ‘0’



‘0’ received == 4 accesses without cache miss

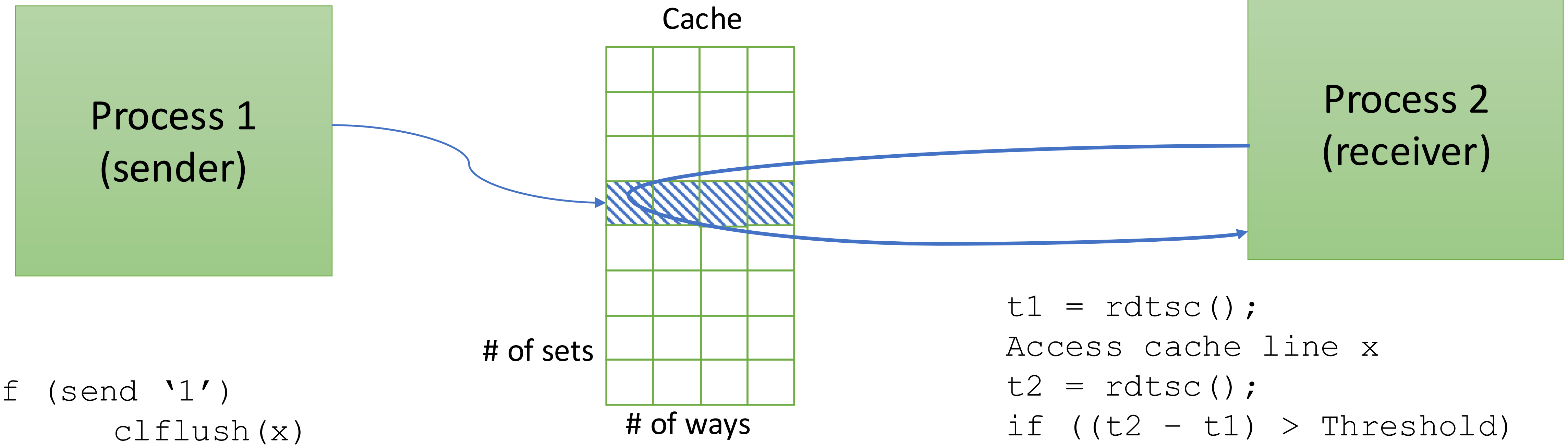


A Complete Protocol -- Synchronization



- Window size agreed on by sender and receiver
- Each window transmits some bits 🡐 bandwidth matters!
- Calibration
  - Both sides need to perform a window alignment at the start

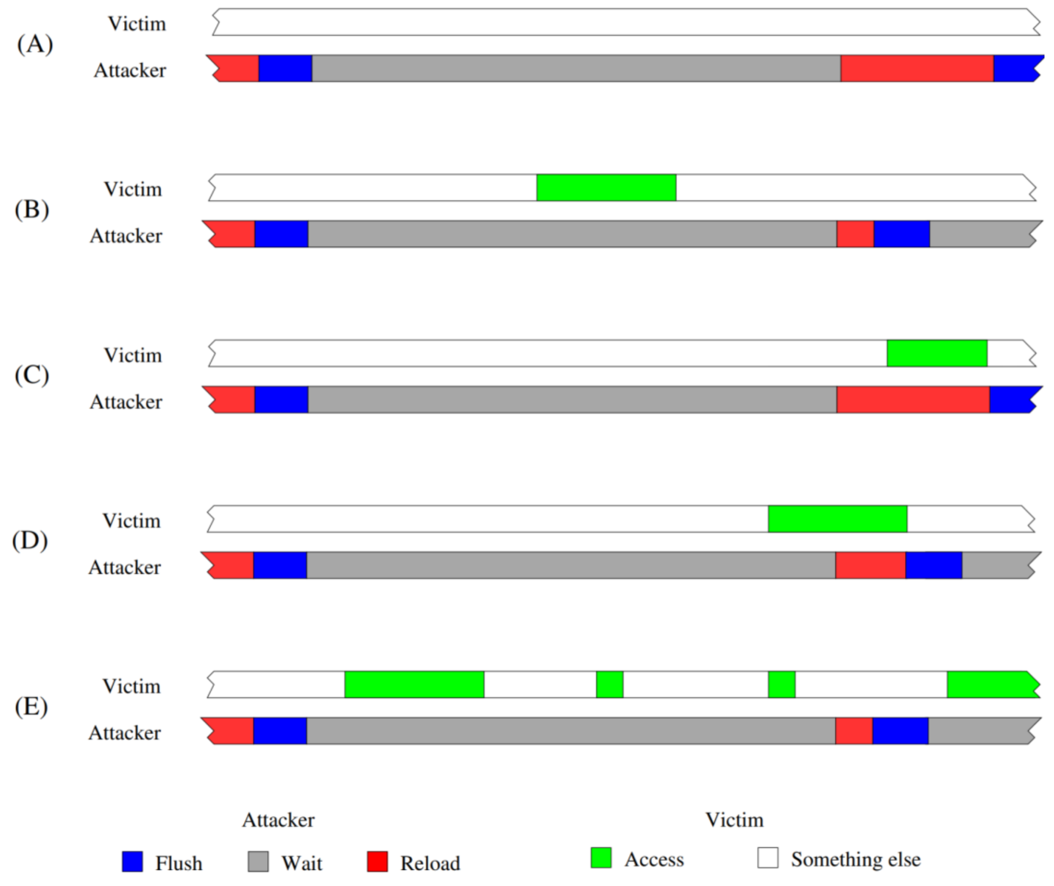
Solution 4: through a smaller part of cache



```
if (send ‘1’)
        clflush(x)
else
        ; // idle
```

```
t1 = rdtsc();
Access cache line x
t2 = rdtsc();
if ((t2 - t1) > Threshold)
        read ‘1’
else
        read ‘0’
```

Flush+Reload



(A) No Victim Access

(B) With Victim Access

(C) Victim Access Overlap

(D) Partial Overlap

(E) Multiple Victim Accesses

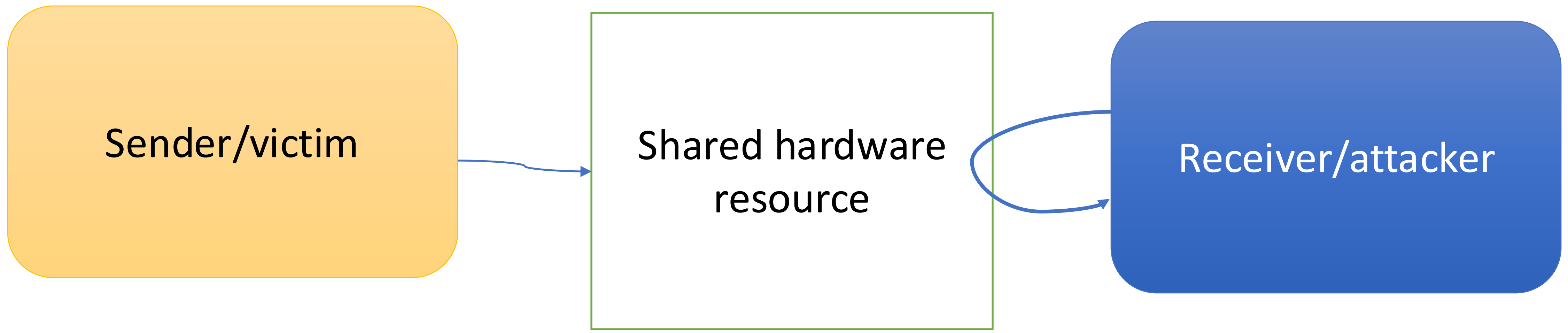
Flush+Reload

- To build a side channel through `clflush`
- Manually share memory pages between process 1 and process 2
- Process 1 forcefully flushes and reloads shared cache line
  - Process 2 accesses the cache line to get cache miss/hit

  ➔ to speculate a ‘1’ or ‘0’

Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 719–732.

Key factor for a successful communication/attack

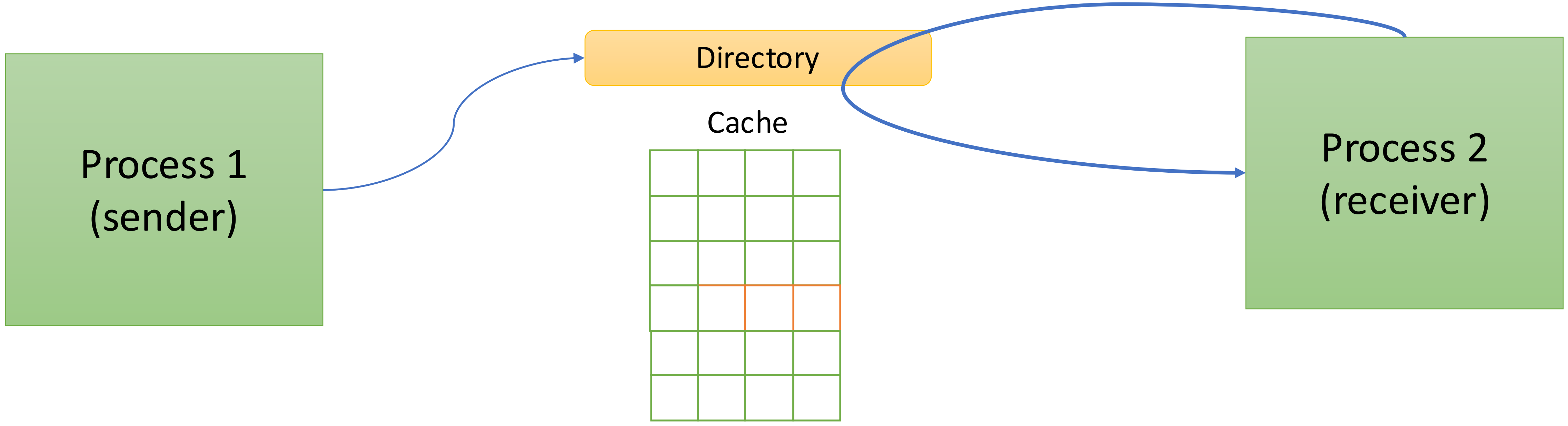


```
if (send '1') // or secret
        do sth with the resource
else
        idle, or do sth else with the resource
```

```
t1 = rdtsc();
do sth with the resource
t2 = rdtsc();
if (t2 - t1 > Threshold)
        read '1'
else
        read '0'
```

Solution 5: through directories, not cache

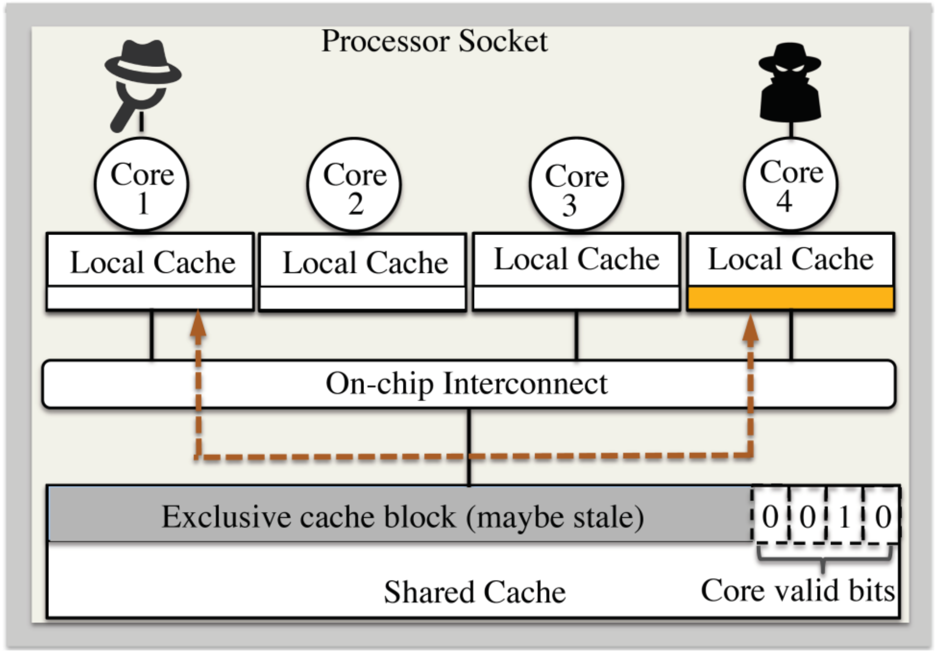
- Inclusive vs non-inclusive cache
- Although cache is non-inclusive, some other structure is shared
  - For cache coherence, we studied "directory"

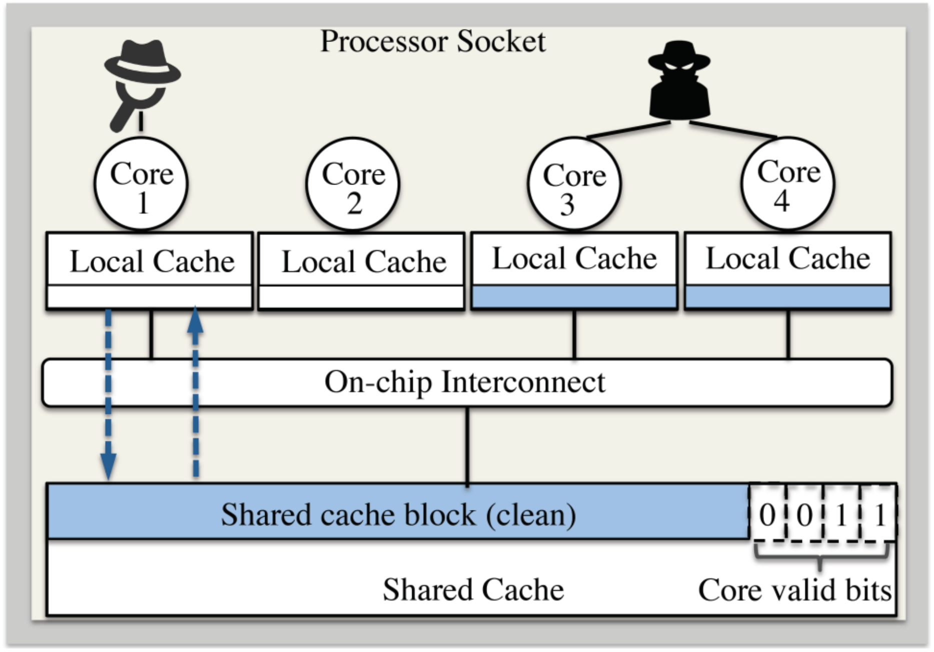


Attack Directories, Not Caches: Side-Channel Attacks in a Non-Inclusive World, S&P '19

Solution 6: through cache coherence protocol

- Cache coherence is also a viable resource
  - Remember E and S states of MESI protocol?





Different states, different latencies

- Two considerations
  - Shared memory, read-only, between two sender and receiver
  - Synchronization prior to transmission
- An encoding *contract* of ‘1’ or ‘0’
  - To transmit every ‘1’, cache block put in a coherence state for x times;
  - To transmit every ‘0’, cache block put in a coherence state for y times;
  - In-between every bit transmission, sender places the cache block in another coherence state for z times.
    - 🡸 to denote bit boundaries.

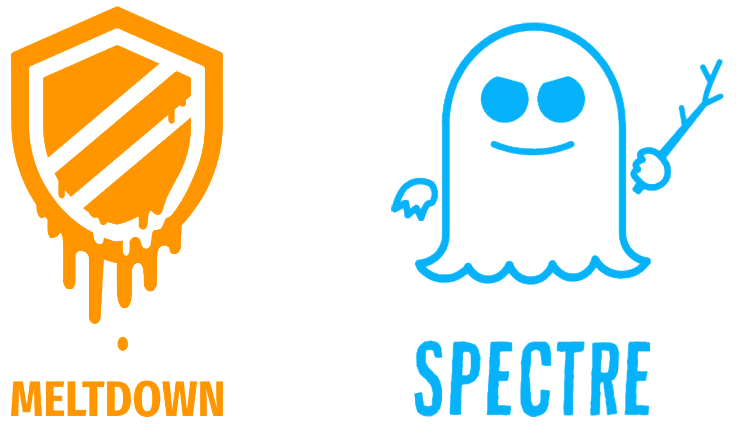
Any more solutions?

How to communicate **without** letting OS know?

Meltdown and Spectre

Meltdown and Spectre

- Hardware vulnerability
  - Affecting Intel x86 microprocessors, IBM POWER processors, and some ARM-based microprocessors
- All Operating Systems affected!
- They are considered “catastrophic”!
- Allow to read all memory (e.g. from other process or other Virtual Machines (e.g. other users data on Amazon cloud service!))
- How Meltdown and Spectre work covers all knowledge of architecture:
  - Virtual Memory; Protection Levels; Instruction Pipelining; Out-of-order Execution; Speculative Execution; CPU Caching.

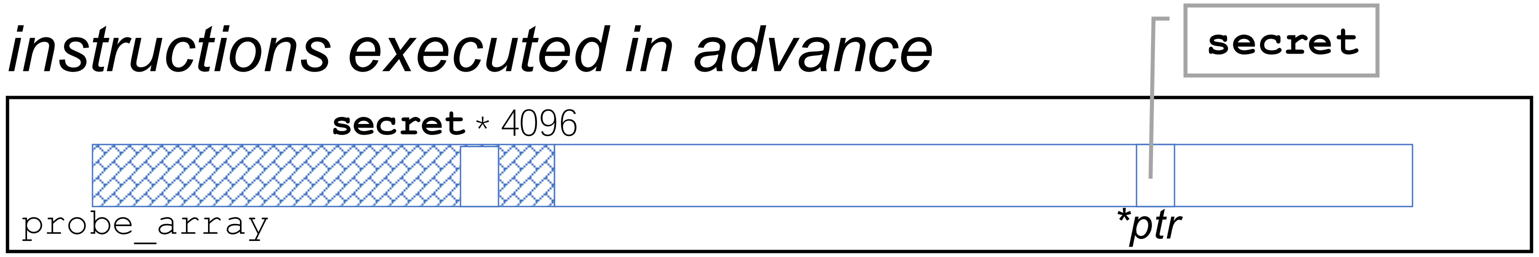


Transient execution, albeit architectural equivalence, leads to information leakage that are reflected by microarchitectural state transitions.

Meltdown: Out of order execution

The aim of Meltdown: to leak/dump memory

- Out of order execution
  - *Some instructions executed in advance*



```
// secret is one-byte. probe_array is an array of char.
1. raise_exception();
2. // the line below is never reached
3. access(probe_array[secret * 4096])
```

probe_array should never be accessed, but accessed at some location probe_array + **secret** * 4096.

probe_array is fully controlled by attacker who can use Flush+Reload to see which cache line of probe_array is hit, so as to figure out the value of **secret.**
secret can be the value at any memory location, i.e., **ptr*

```
; rcx: inaccessible kernel address
; rbx: probe_arry.
1. xor rax, rax ; rax ← 0
2. retry:
3.     mov al, byte [rcx]
4.     shl rax, 0xc ; ×4096
5.     jz retry
6. mov rbx, qword[rbx + rax]
```

上海科技大学
ShanghaiTech University

The Impact of Meltdown

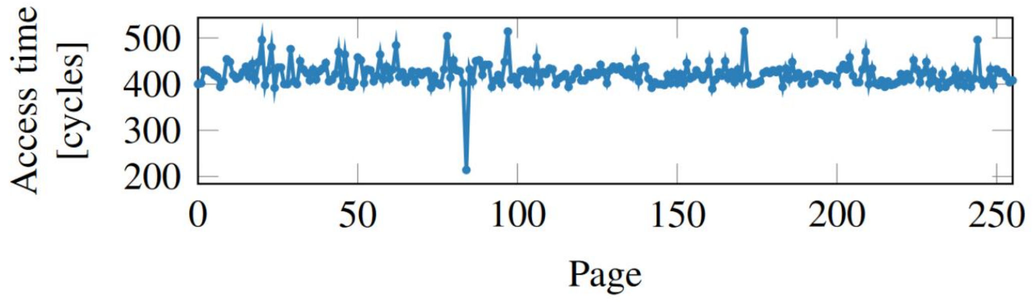


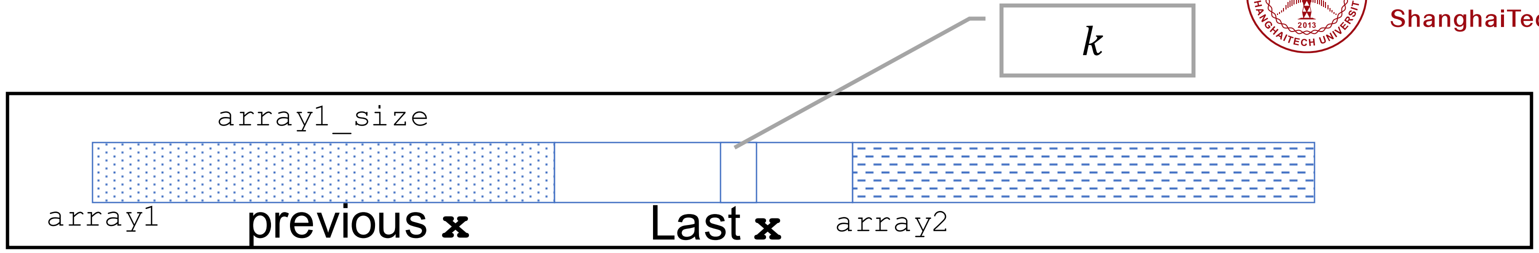
Figure 4: Even if a memory location is only accessed during out-of-order execution, it remains cached. Iterating over the 256 pages of probe_array shows one cache hit, exactly on the page that was accessed during the out-of-order execution.

Justification:
The researchers put a value of 84 in **secret** and managed to use Flush+Reload to get a cache hit at the 84th page.

The researchers developed competent programs to read memory locations that should be inaccessible to their program. They managed to dump the entire physical memory, for kernel and users.

Source: https://meltdownattack.com/meltdown.pdf

Spectre: Speculation



- Speculative execution
  - Example: branch prediction

Prerequisites:
i. array1[**x**], with an out-of-bound **x** larger than array1_size, resolves to a secret byte k that is cached;
ii. array1_size and array2 uncached.
iii. Previous **x** values have been valid.

```
// x is controlled by attacker.
1. if (x < array1_size)
2.    y = array2[array1[x] * 4096]
```

🡐 cache miss, so run next line due to prediction history

🡐 array1[**x**] cache hit, as k is cached, so load array2[k * 4096]

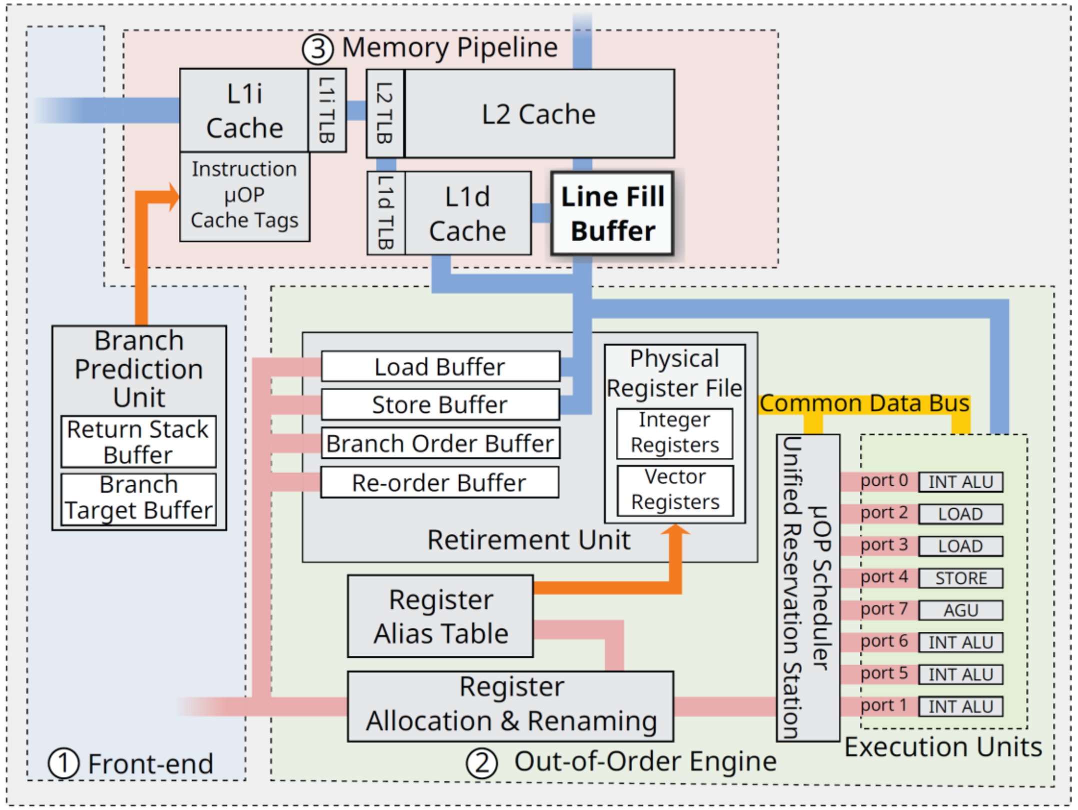
Regarding a misprediction with an illegal **x**, array2[k * 4096] will not be used, but has been loaded into CPU cache.
We can use Flush+Reload to guess k with array2.

The aim of Spectre:
to read out a victim's sensitive information

RIDL: Rogue In-Flight Data Load

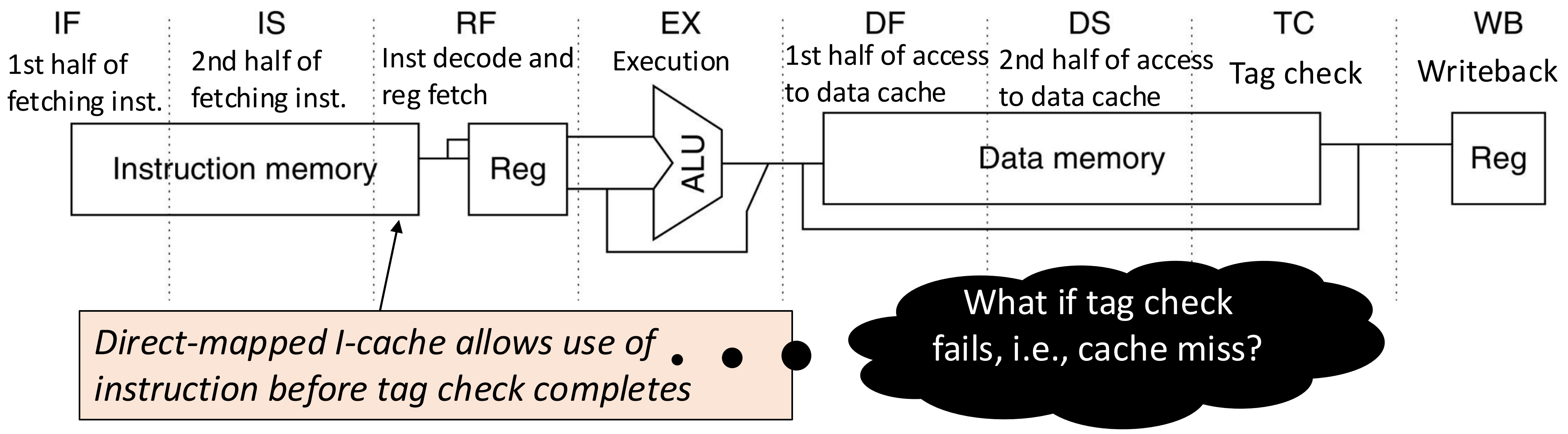
- Not data in cache, but in line fill buffer (LFB)

Line Fill Buffers (LFBs) are internal buffers that the CPU uses to keep track of outstanding memory requests and perform a number of optimizations such as merging multiple in-flight stores.



An overview of the Intel Skylake microarchitecture from *RIDL: Rogue In-Flight Data Load*, S&P ’19

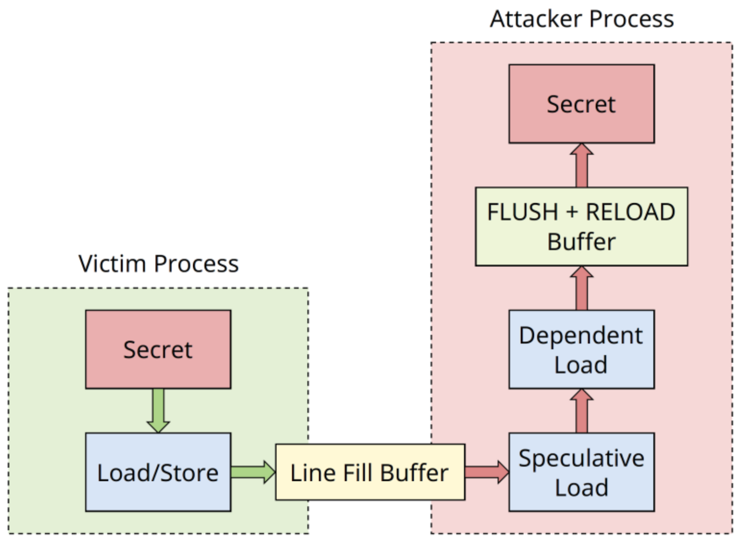
Speculative load (L05)


IF
1st half of fetching inst.
IS
2nd half of fetching inst.
RF
Inst decode and reg fetch
EX
Execution
DF
1st half of access to data cache
DS
2nd half of access to data cache
TC
Tag check
WB
Writeback
Instruction memory
Reg
ALU
Data memory
Reg
Direct-mapped I-cache allows use of instruction before tag check completes
What if tag check fails, i.e., cache miss?

The eight-stage pipeline structure of the MIPS R4000 (1991) uses pipelined instruction and data caches.

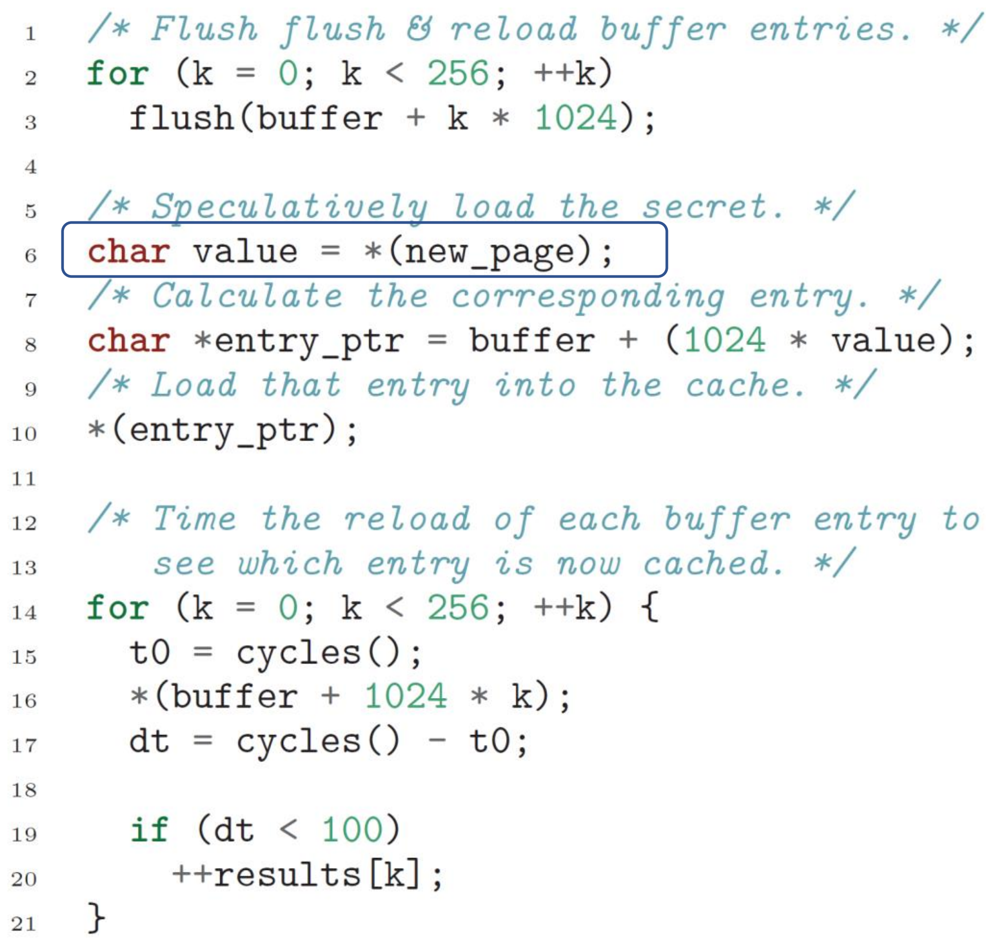
RIDL: an example

Any problems with RIDL?



An overview of the RIDL attack from *RIDL: Rogue In-Flight Data Load*, S&P '19

when executing Line 6, the CPU speculatively loads a value from memory in the hope it is from our newly allocated page, while really it is in-flight data from the LFBs belonging to an arbitrarily different security domain



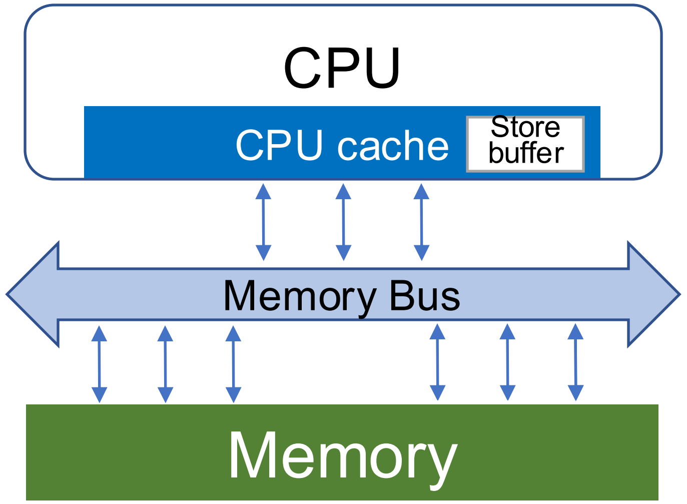
```
1   /* Flush flush & reload buffer entries. */
2   for (k = 0; k < 256; ++k)
3     flush(buffer + k * 1024);
4
5   /* Speculatively load the secret. */
6   char value = *(new_page);
7   /* Calculate the corresponding entry. */
8   char *entry_ptr = buffer + (1024 * value);
9   /* Load that entry into the cache. */
10  *(entry_ptr);
11
12  /* Time the reload of each buffer entry to
13      see which entry is now cached. */
14  for (k = 0; k < 256; ++k) {
15    t0 = cycles();
16    *(buffer + 1024 * k);
17    dt = cycles() - t0;
18
19    if (dt < 100)
20      ++results[k];
21  }
```

More MDS attacks

- Microarchitectural data sampling (MDS)
- Fallout
  - *Fallout: Leaking Data on Meltdown-resistant CPUs*, CCS '19
- ZombieLoad
  - *ZombieLoad: Cross-Privilege-Boundary Data Sampling*, CCS '19
- Medusa
  - *Medusa: Microarchitectural Data Leakage via Automated Attack Synthesis*, USENIX Security '20
- CacheOut
  - *CacheOut: Leaking Data on Intel CPUs via Cache Evictions*, S&P '21

LVI: Value Injection

CPU thinks this one in store buffer seems to be the one it needs.

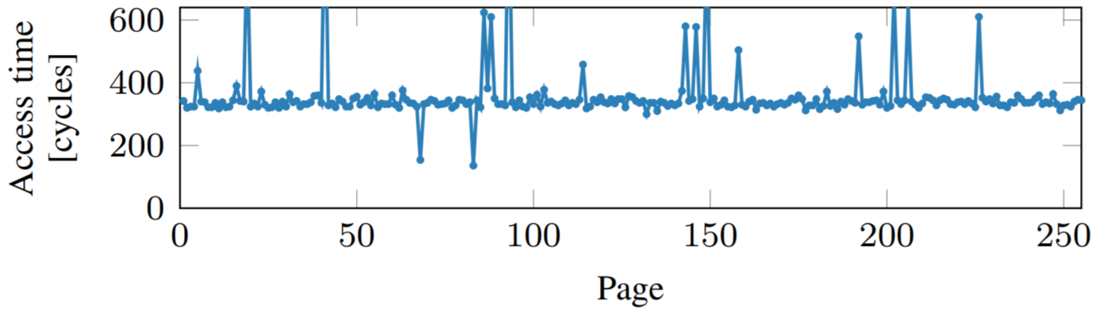


```
void call_victim(size_t untrusted_arg) {
    *arg_copy = untrusted_arg;
    array[**trusted_ptr * 4096];
}
```

Dereference: but at 1st level, page fault.

An attacker-controlled value is fed, and some micro-architectural state change happens.

CPU may use data in the store buffer upon a cache miss of memory load, although the two are completely unrelated.



Conclusion

- Security is a serious and critical issue
- This lecture only sees the tip of the iceberg

Simple, but not easy

The Prestige

Acknowledgements

- These slides contain materials developed and copyright by:
  - Prof. Mengjia Yan (MIT)
  - Authors of all mentioned papers