RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: HTA-T09

The Foreshadow Attack: From a Simple Oversight to a Technological Nightmare

Dr Raoul Strackx, PhD

imec-DistriNet, KU Leuven, Belgium

Foreshadow Attacks

- Independently discovered
- Team of KU Leuven, Belgium
- Team of Universities of Technion, Michigan and Adelaide and DATA61
- Intel discovered other variants

foreshadowattack.eu

KU LEUVEN

Technion Israel Institute of Technology

UNIVERSITY OF MICHIGAN

THE UNIVERSITY of ADELAIDE

DATA 61

2018 was quite interesting...

source: https://software.intel.com/security-software-guidance/software-guidance

These were vulnerabilities in the processor itself
Hence, virtually *every* application was effected!

This led to various reactions...

How we told our upper management at the university (Nov '17)...

Source: https://pin.it/k4j53t23xiiqcd

How we told Intel (Jan '18)…

Source: https://pin.it/k4j53t23xiiqcd

How IT professionals reacted (to this class of vulnerabilities)…

Source: https://pin.it/hehzyfhdsvnlkc

How do these attacks work, in general?

Side-Channel Attacks

Source: The Italian Job (imdb.com)

Side-Channel Attacks

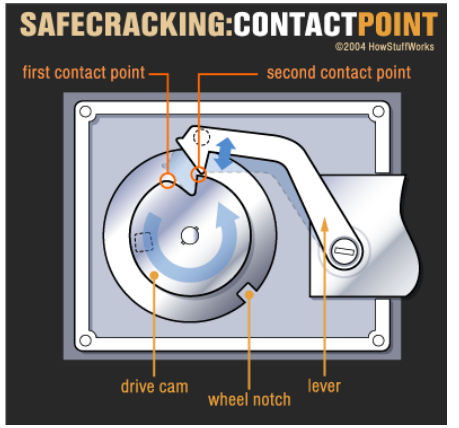
Attacker

Charleze Theron

Action: rotate

Carrier: sound

Victim



Vault

Security flaw: Lever may produce sound

Sources: https://home.howstuffworks.com, imdb.com

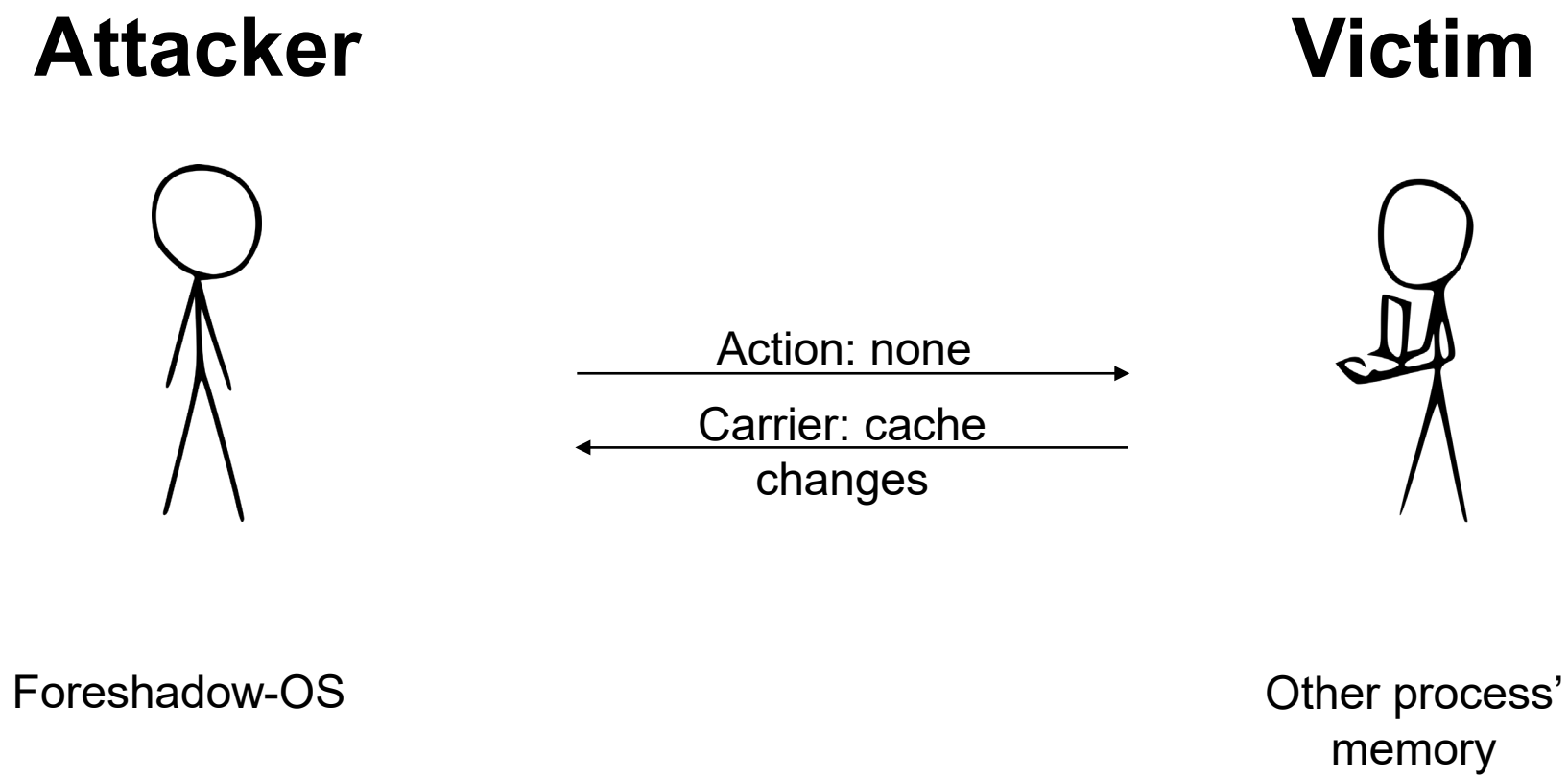
How does the Foreshadow attack work?

Attacks

- **Foreshadow-OS**
- **Foreshadow-VMM**
- **Foreshadow-SGX**
- **Foreshadow-SMM**
- →All exploit the same vulnerability, but the attack model varies

Foreshadow-OS: Reading L1 data through *bare-metal* not-present pages

Side-Channel Attacks



Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

Side-Channel Attacks

Attacker

Foreshadow-OS

Action: none

Carrier: cache changes

Victim

Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

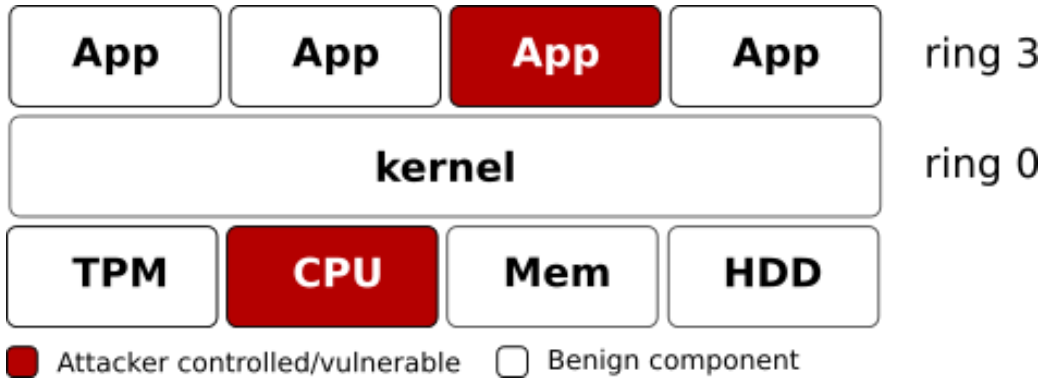
Setting: Attacker-controlled process

Attack model:

- Attacker operates within a malicious process
- Benign, bare-metal kernel ensures process isolation

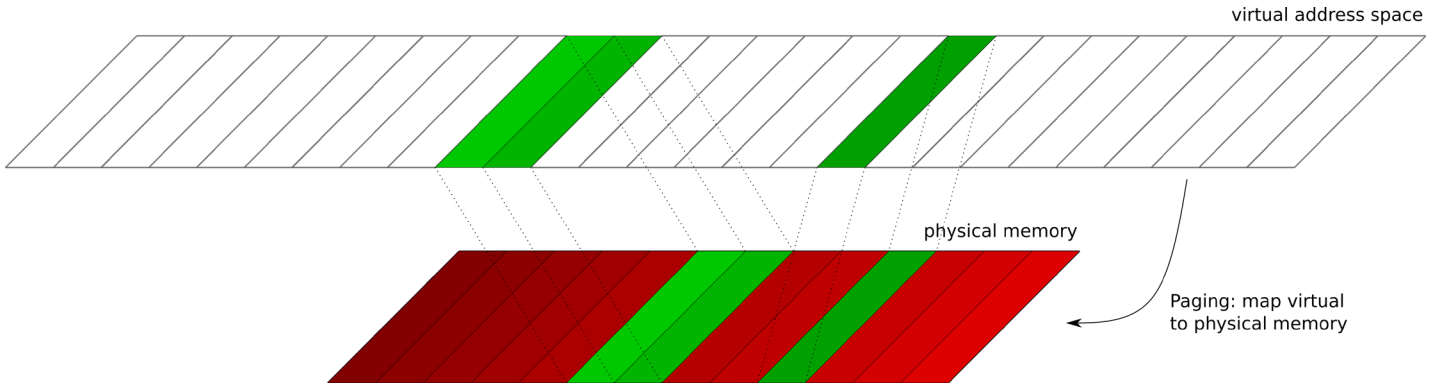
Attack objective:

- Read data outside the process' address space



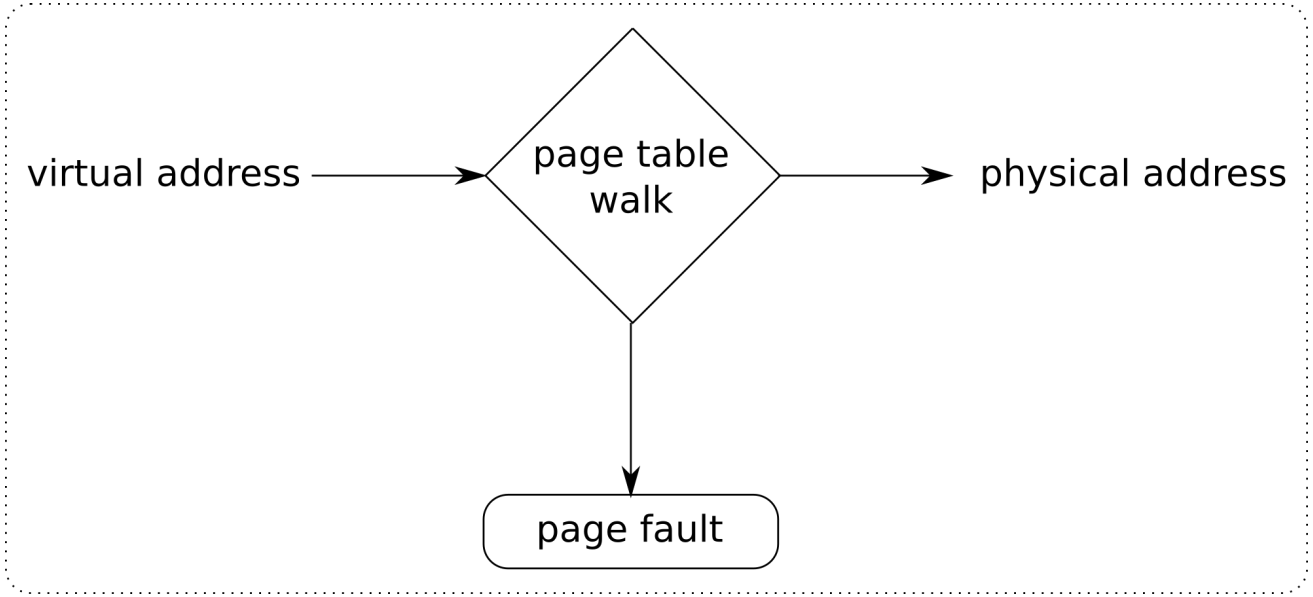
Background: How does process isolation work

- MMU: map virtual address space to physical memory
- Protect physical memory by:
  - Not providing a mapping
  - Restricting access (e.g., U/S-bit)



Background: How does process isolation work

- MMU: map virtual address space to physical memory
- Protect physical memory by:
  - Not providing a mapping
  - Restricting access (e.g., U/S-bit)



Background: How does process isolation work

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Source: Intel 64 and IA-32 architectures software developer's manual

Background: How does process isolation work

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Source: Intel 64 and IA-32 architectures software developer's manual

Background: How does process isolation work

”When P-bit is 0, the entry's physical address field may be re-used to keep track of the swapped out page”

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Side-Channel Attacks

Attacker

Foreshadow-OS

Action: none

Carrier: cache changes

Victim

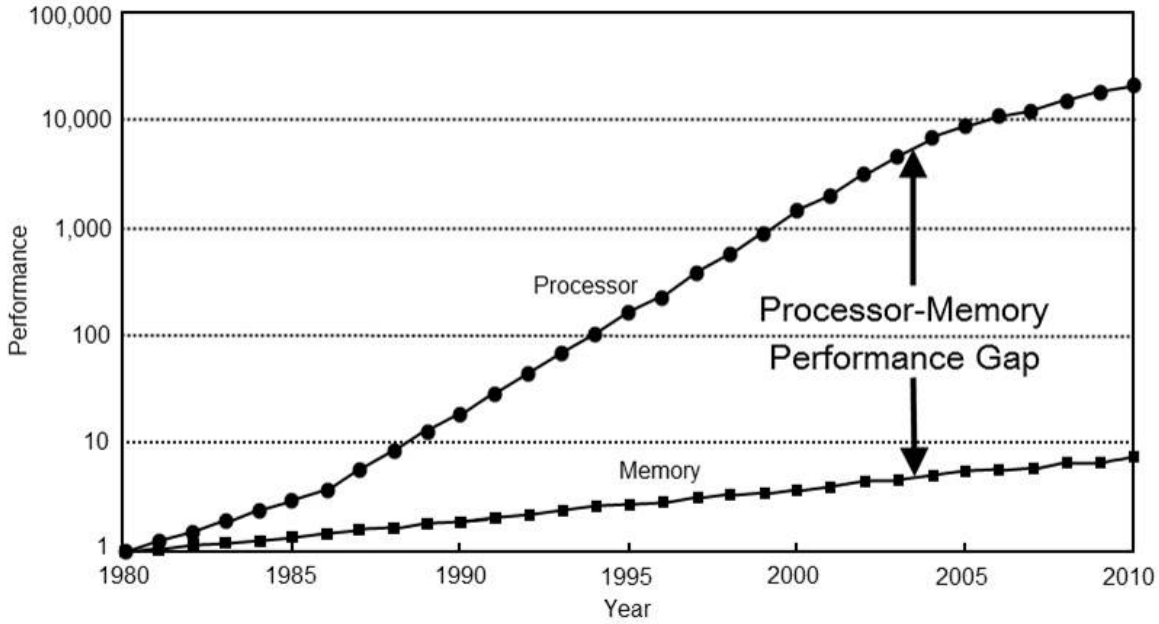
Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

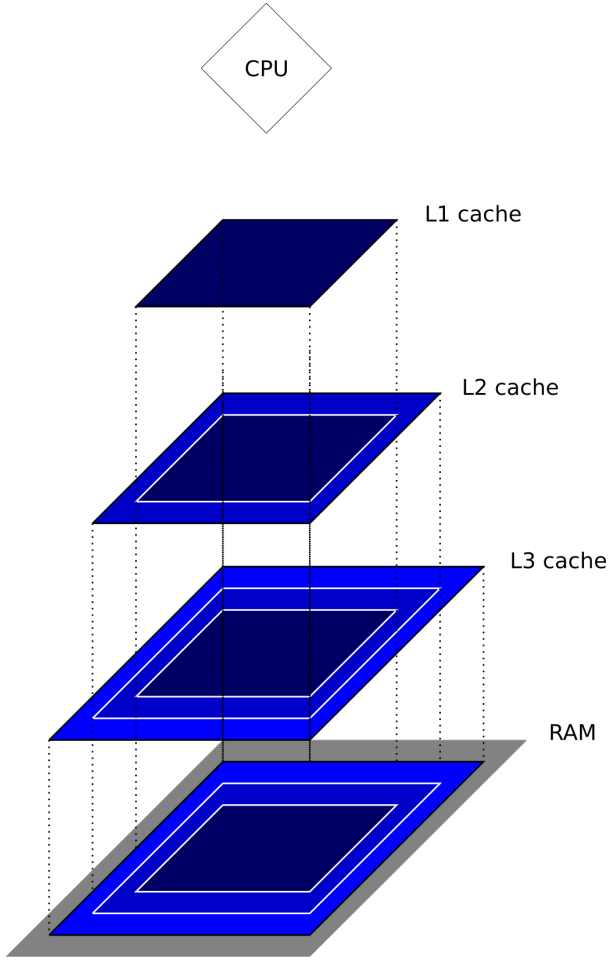
The message carrier: How does the cache work?

- **Problem**: Memory performance grows much slower than CPU performance
- **Solution**: Fast but small caches
  - Intel 486: L1 cache ('89)
  - Pentium Pro: L1 & L2 cache ('95)
  - Today: L1, L2 & L3 caches



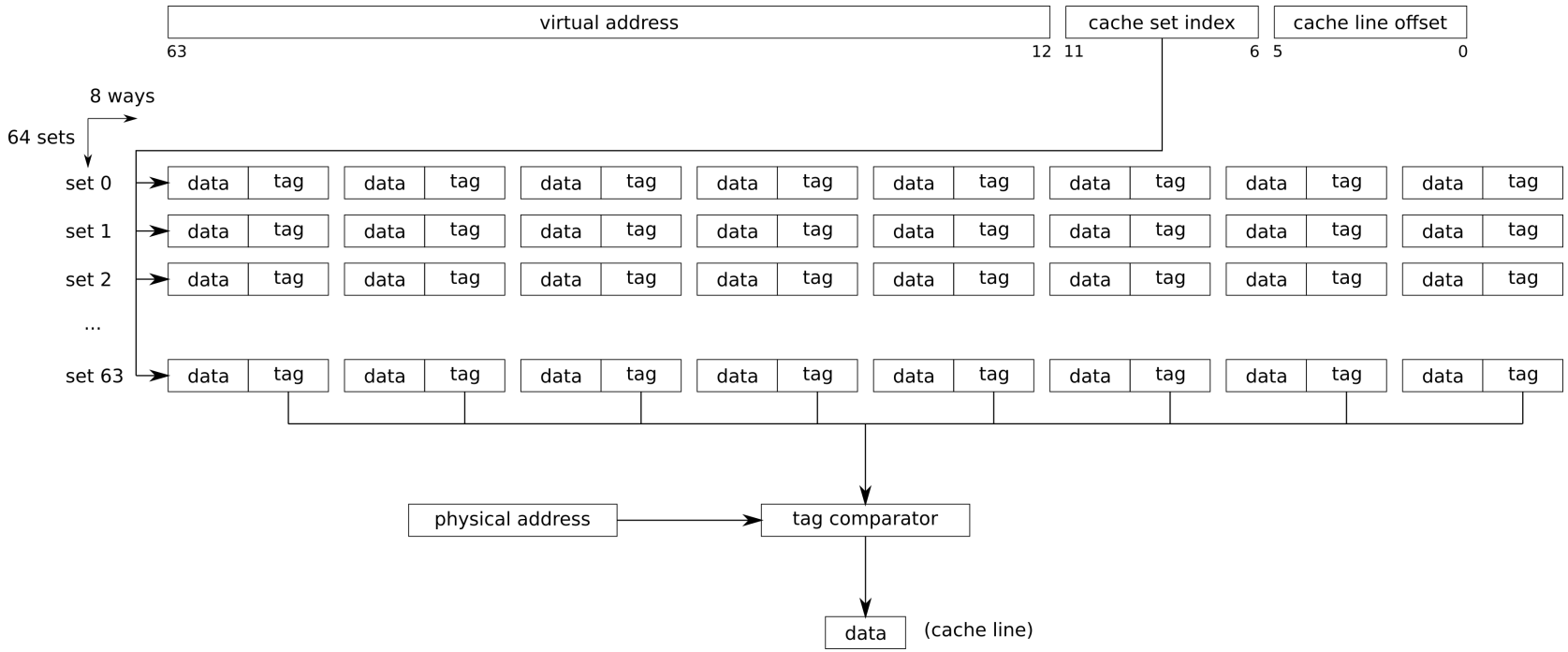
The message carrier: How does the cache work?

- **Problem**: Memory performance grows much slower than CPU performance
- **Solution**: Fast but small caches
  - Intel 486: L1 cache ('89)
  - Pentium Pro: L1 & L2 cache ('95)
  - Today: L1, L2 & L3 caches



The message carrier: How does the cache work?

- Cache lines: 64B
- L1: virtually-indexed, physically tagged
- 64 sets, 8 ways



The message carrier: How does the cache work?

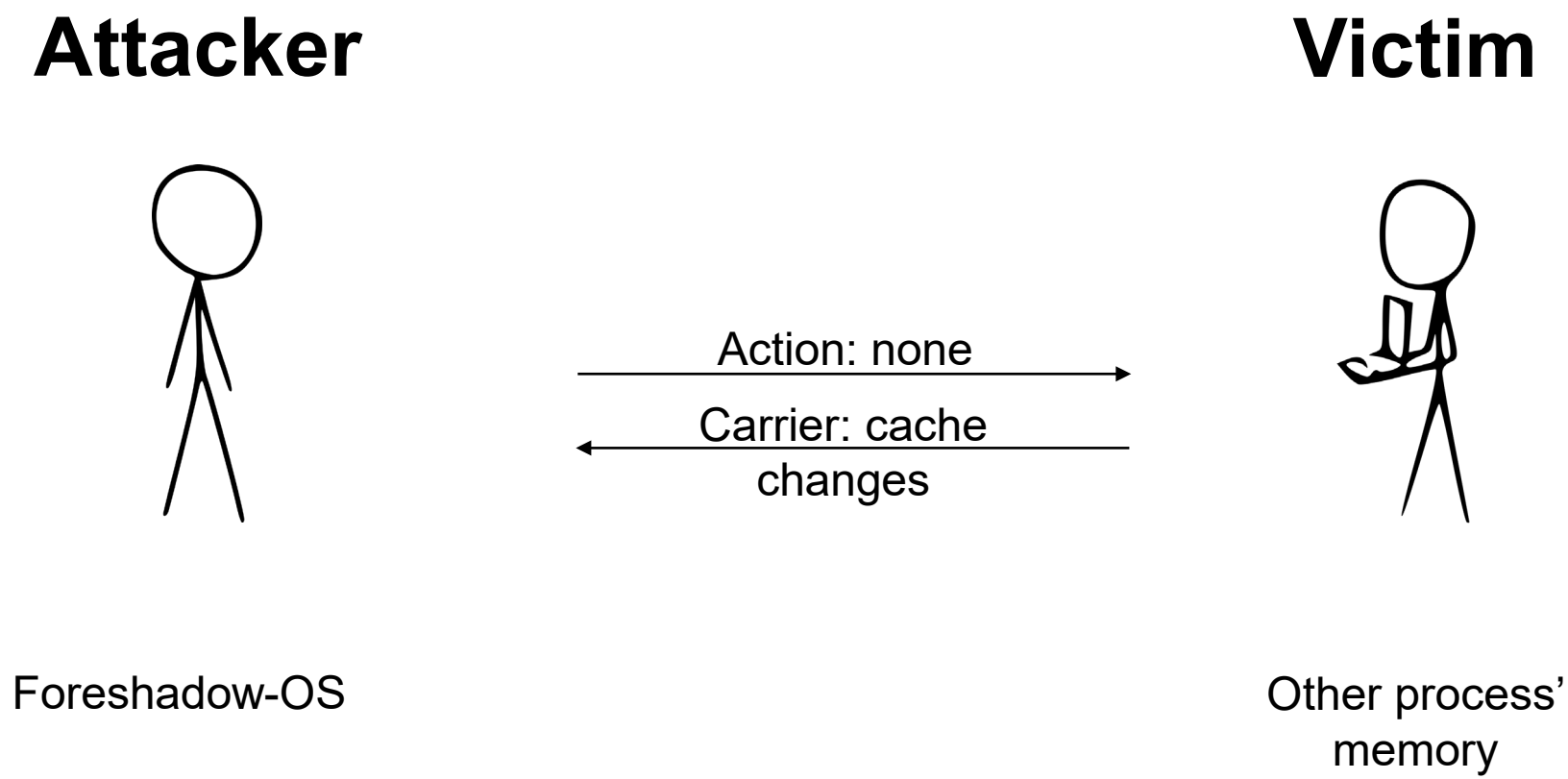
Manipulating the cache:

- Data accesses: load in all cache levels
- *clflush*: Flush data from all cache levels

memory	timing (in cycles)	std. dev.
L1	46	1.25
L2	53	1.14
RAM	246	6.22

→Any timing results <146 cycles clearly hits the cache

Side-Channel Attacks

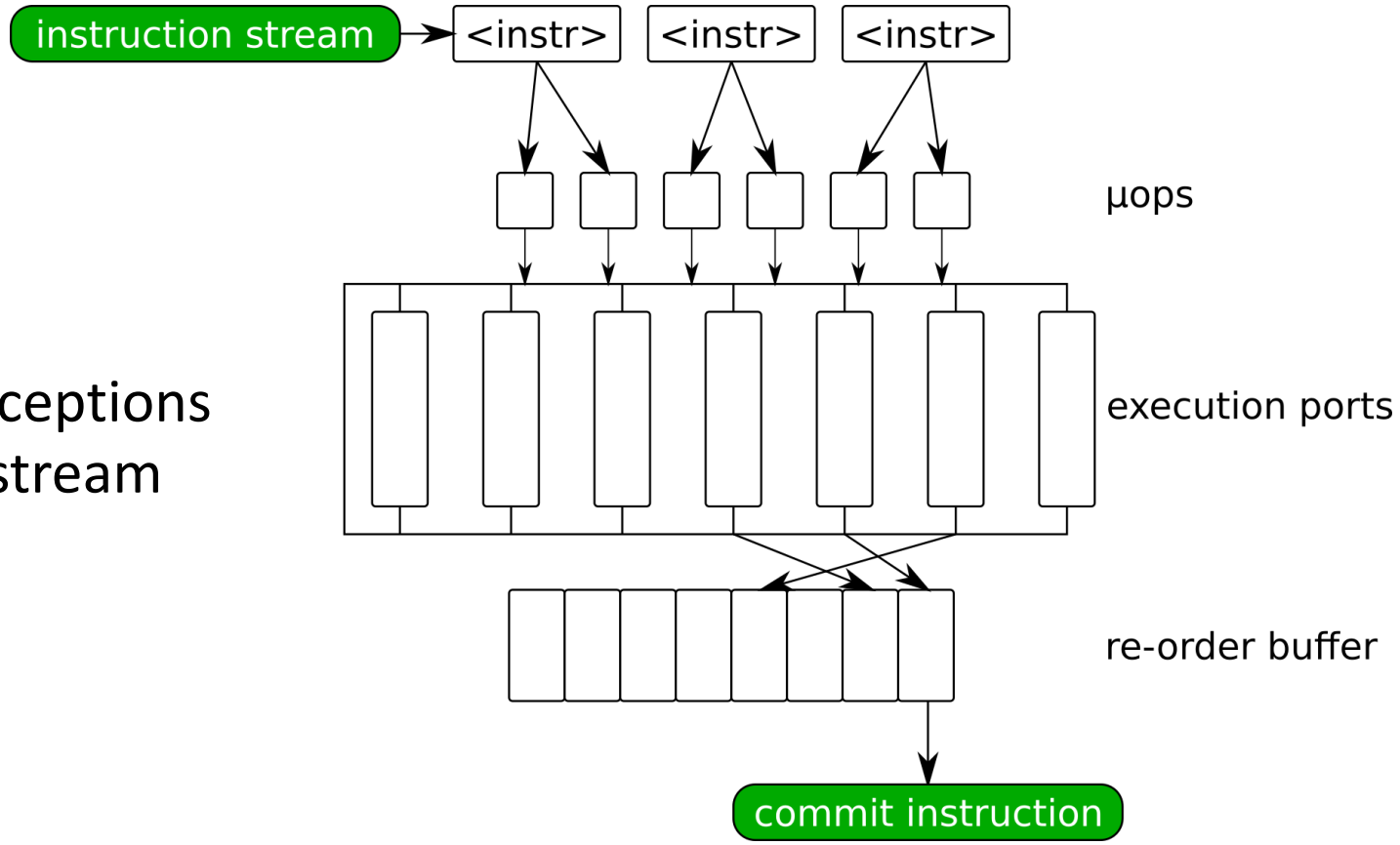


Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

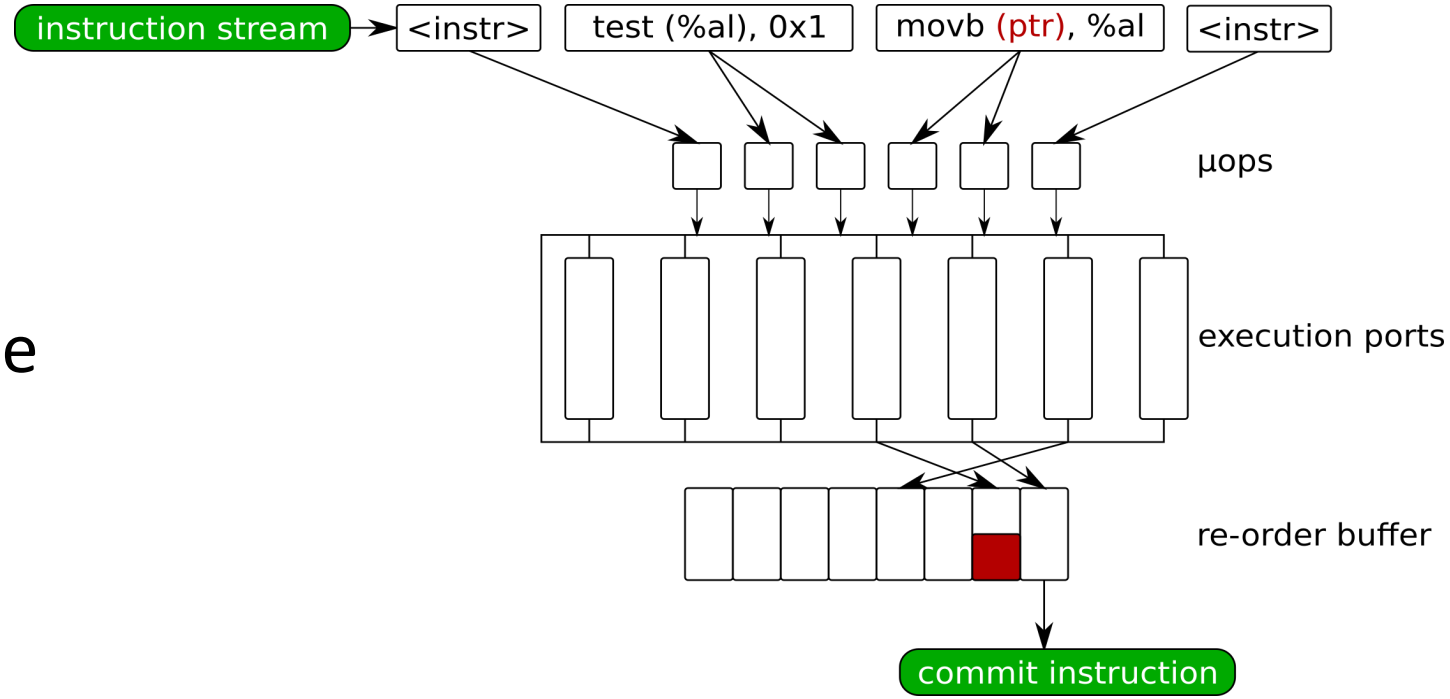
Background: Pipelining & Out of Order Execution

- Split instruction in µops
- Use multiple execution ports
- Execute µops as soon as possible
- Reorder buffer ensures results/exceptions are visible in-order of instruction stream



The Security Flaw: Transient Execution

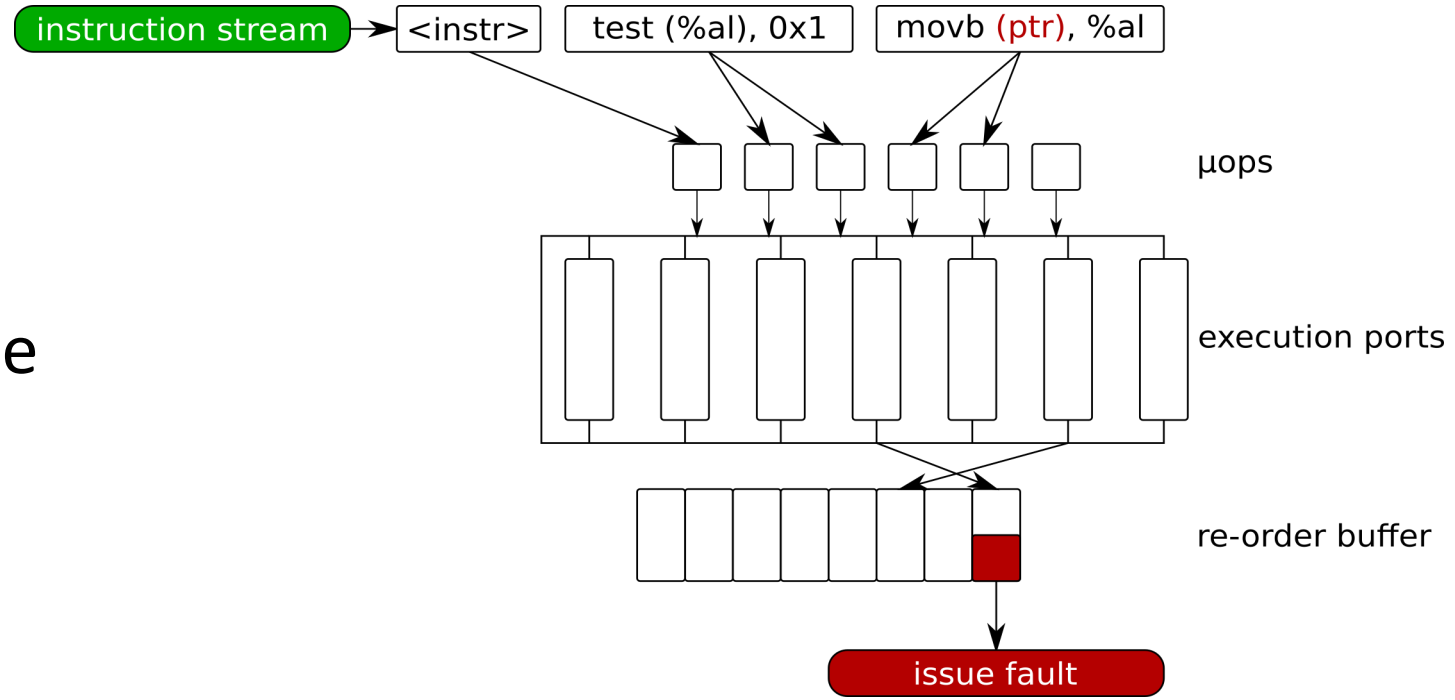
- Transient execution:
- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started



Key Issue: Not all side-effects of transient instructions are rolled back correctly! (e.g., cache changes)

The Security Flaw: Transient Execution

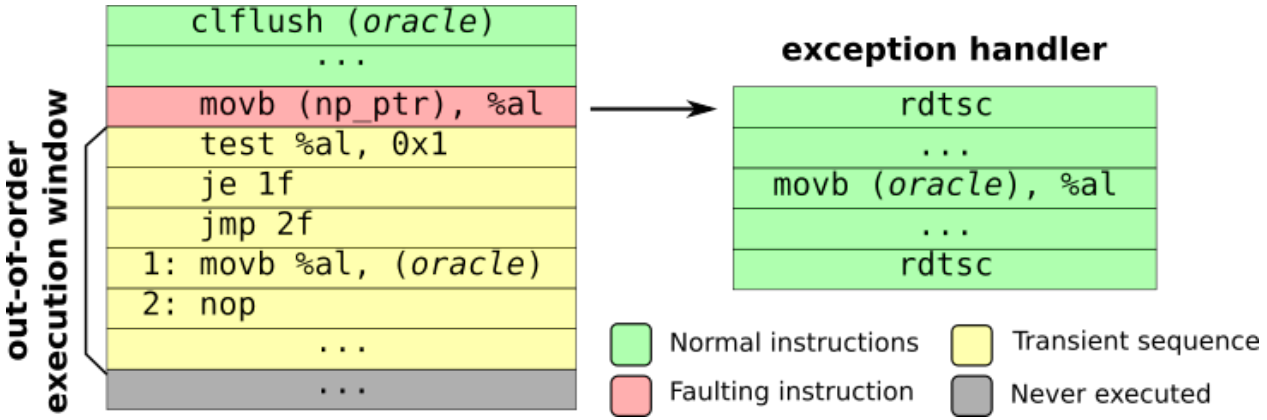
- Transient execution:
- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started



Key Issue: Not all side-effects of transient instructions are rolled back correctly! (e.g., cache changes)

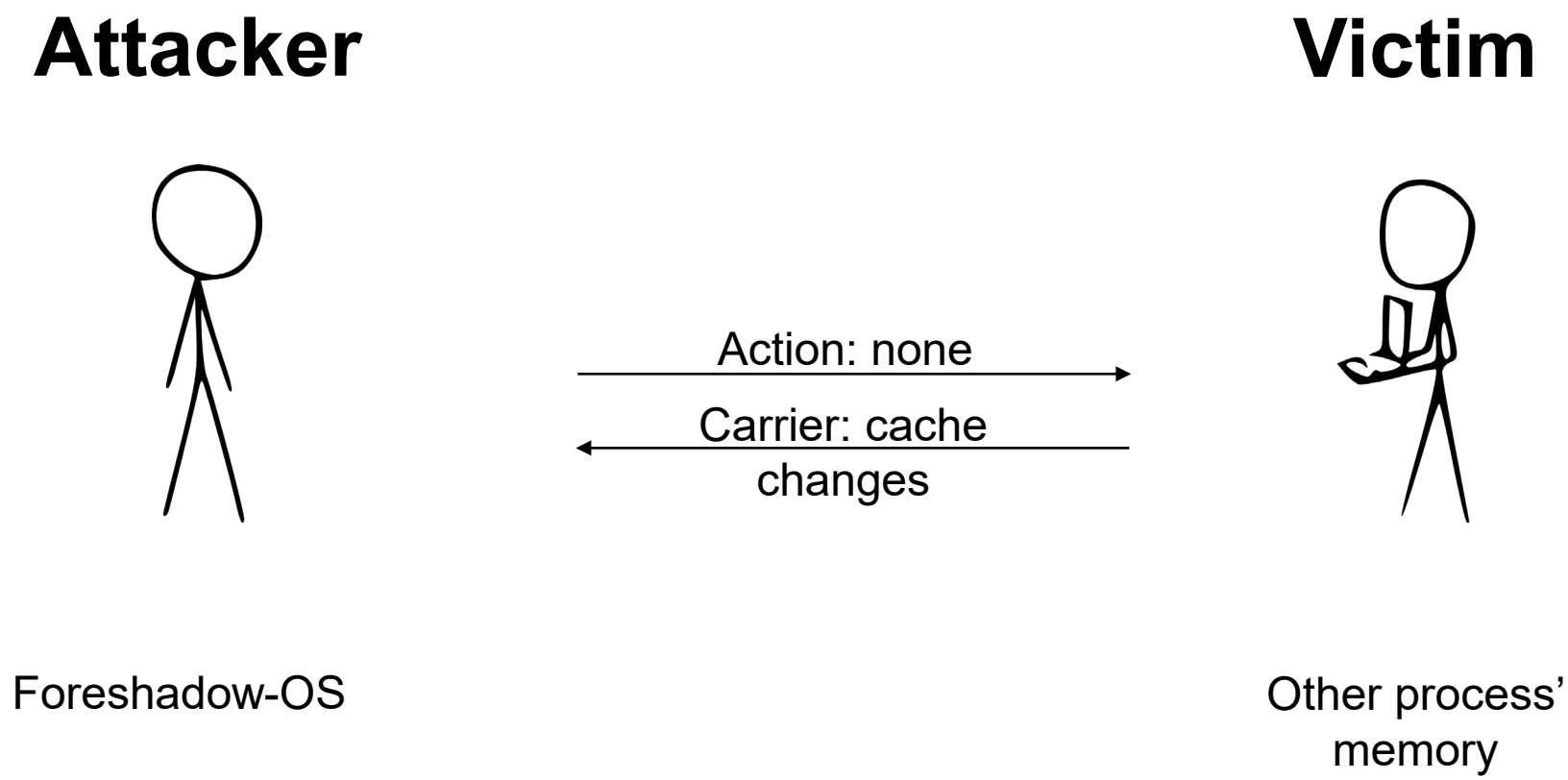
The Security Flaw: Transient Execution

- Transient execution:
- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started



Key Issue: Not all side-effects of transient instructions are rolled back correctly! (e.g., cache changes)

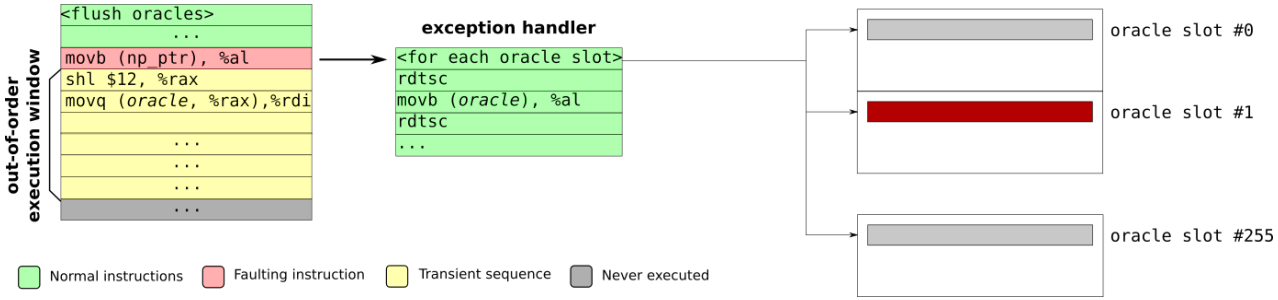
Side-Channel Attacks



Security flaw: OoO execution leaves traces of transient instructions

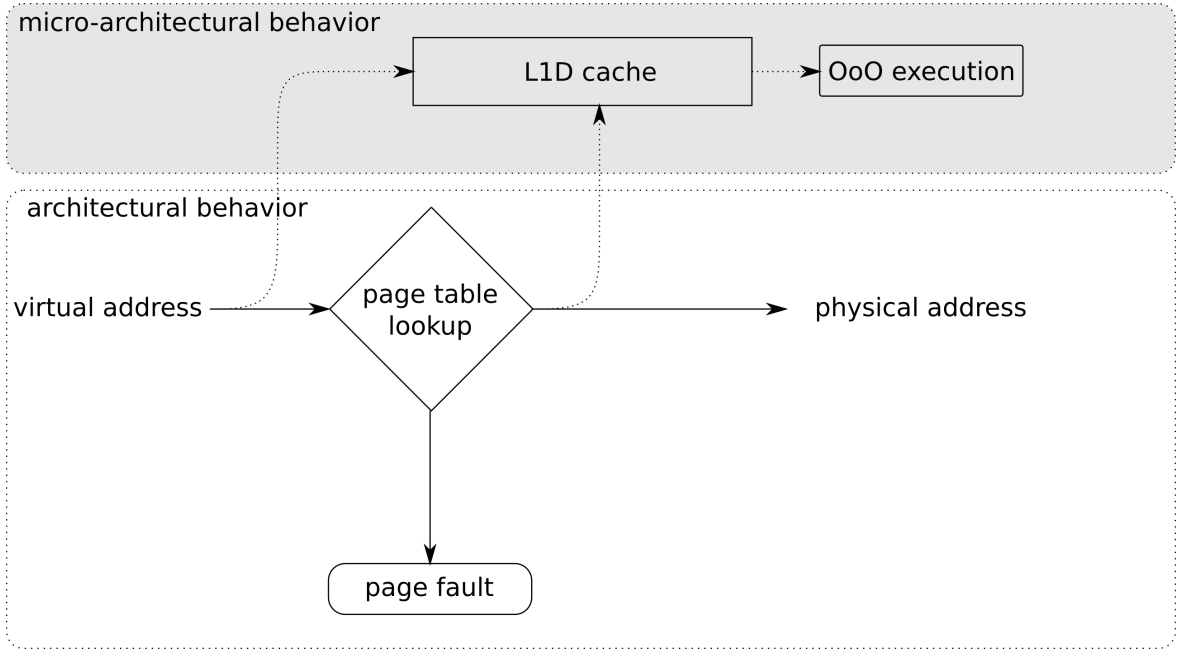
Source: xkcd.com

Foreshadow-OS: The exploit




```
1  int8_t *oracles = ...;
2  int8_t *np_ptr = ...; // the secret
3  int8_t _tmp;
4
5  // Step 1: Remove oracle slots from cache
6  for ( int i = 0; i < 256; ++i )
7    clflush( &oracles[4096 * i] );
8
9  // Step 2: Trick system in sensitive data in L1 but PTE present bit to 0
10
11 // Step 3: attempt to read not present memory
12 _tmp = oracle[4096 * (*np_ptr)];
13
14 // suppress fault
15
16 // Step 4: which oracle slot is cached?
17 for ( int i = 0; i < 256; ++i ) {
18   if ( time_access( oracle[4096 * i] ) < 146 )
19     print( "*np_ptr = %i\n", i );
20 }
```

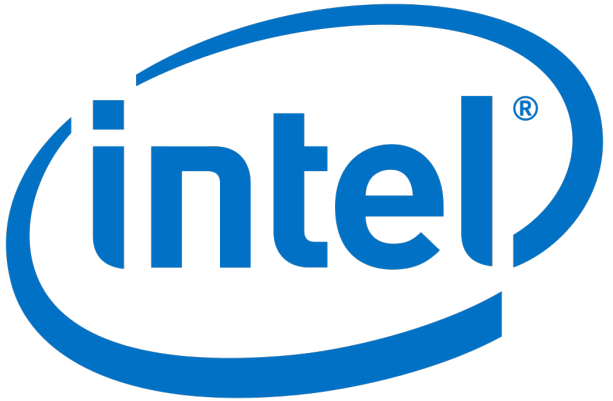
Foreshadow-OS: The exploit




```
1 int8_t *oracles = ...;
2 int8_t *np_ptr = ...; // the secret
3 int8_t _tmp;
4
5 // Step 1: Remove oracle slots from cache
6 for ( int i = 0; i < 256; ++i )
7   clflush( &oracles[4096 * i] );
8
9 // Step 2: Trick system in sensitive data in L1 but PTE present bit to 0
10
11 // Step 3: attempt to read not present memory
12 _tmp = oracle[4096 * (*np_ptr)];
13
14 // suppress fault
15
16 // Step 4: which oracle slot is cached?
17 for ( int i = 0; i < 256; ++i ) {
18   if ( time_access( oracle[4096 * i] ) < 146 )
19     print( "*np_ptr = %i\n", i );
20 }
```

Who's Affected?

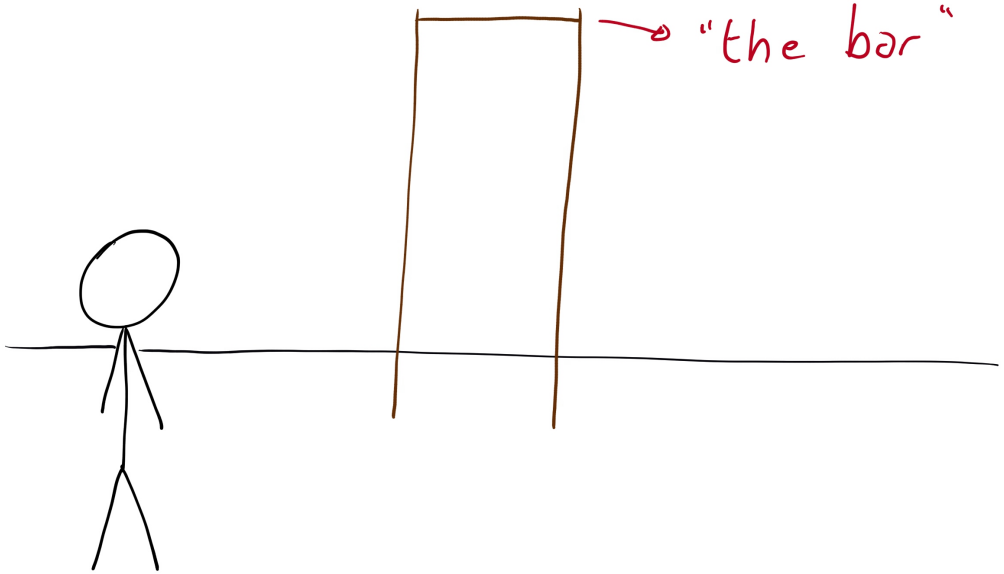
- Intel Core processors of the last 7 years
- Intel server processors
- NOT AMD, not ARM



Impact of This Attack

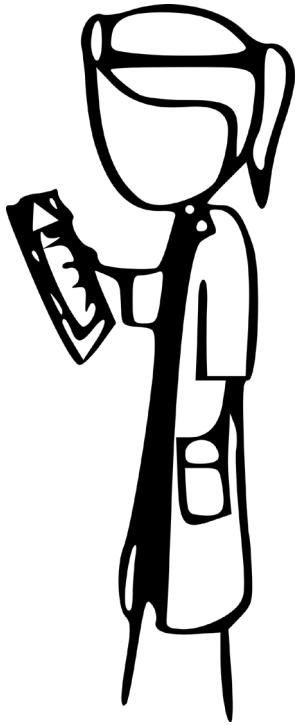
Requirements:

- Secret data in L1D
- Page must be not-present
- ⇝ Must difficult attack, “easiest” to understand
- ⇝ Low impact!



Mitigations

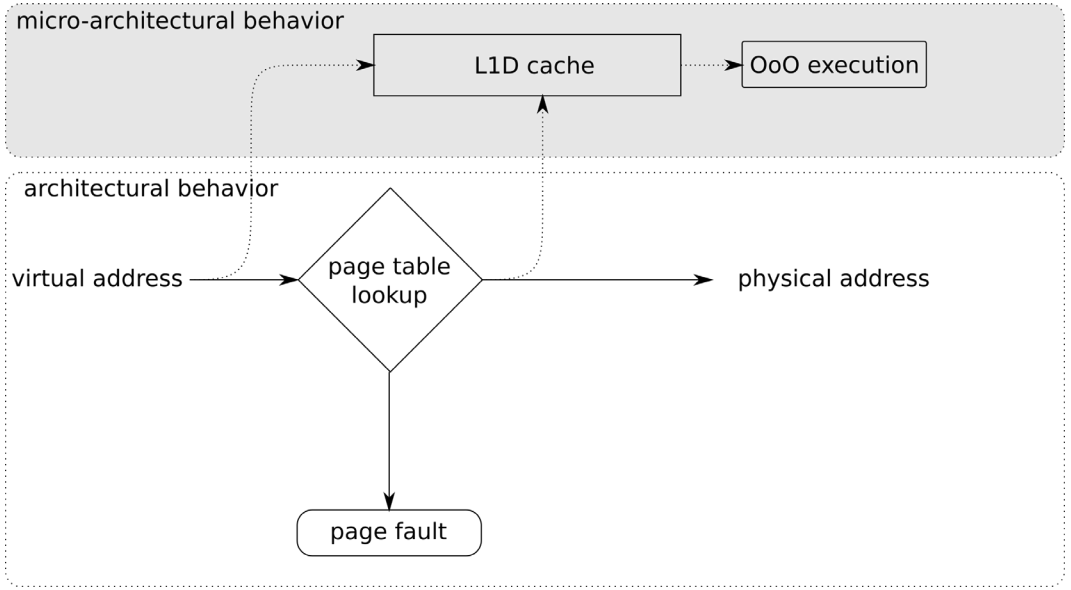
- Long term: Replace chips!
- Short term:
  - Software approaches:
    - Ensure PTE entry do not point to existing physical address
    - Flush complete L1D cache



Source: xkcd.com

Mitigations

- Long term: Replace chips!
- Short term:
  - Software approaches:
    - Ensure PTE entry do not point to existing physical address
    - Flush complete L1D cache



Foreshadow-VMM: Reading physical L1 data through *virtualized* not-present pages...

Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-VMM

Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-VMM

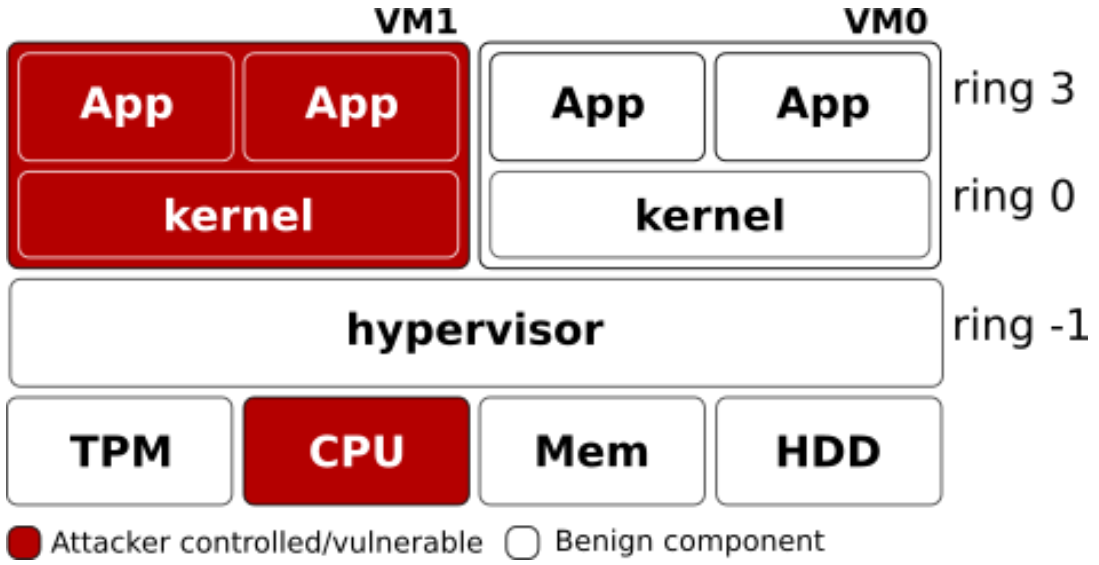
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

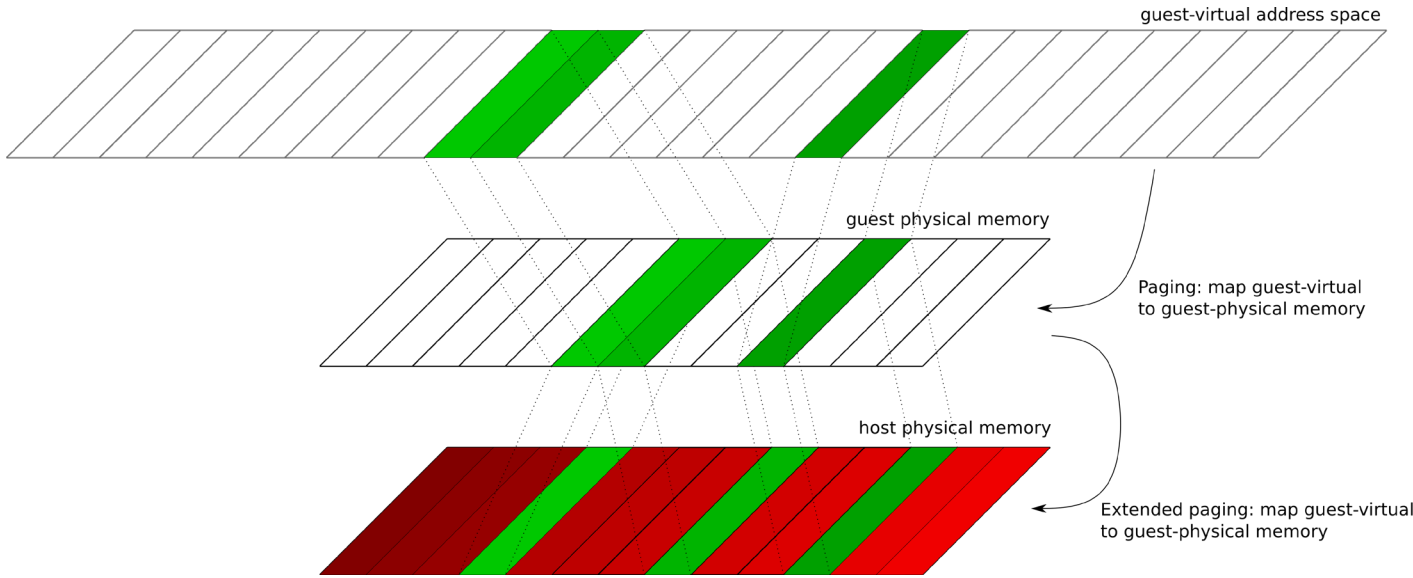
Setting: Attacker-controlled VM

- Multiple VMs on one physical server
- Attacker-controlled VM
- Hypervisor ensures VM isolation
- ⇝ Goal: read other VM's memory



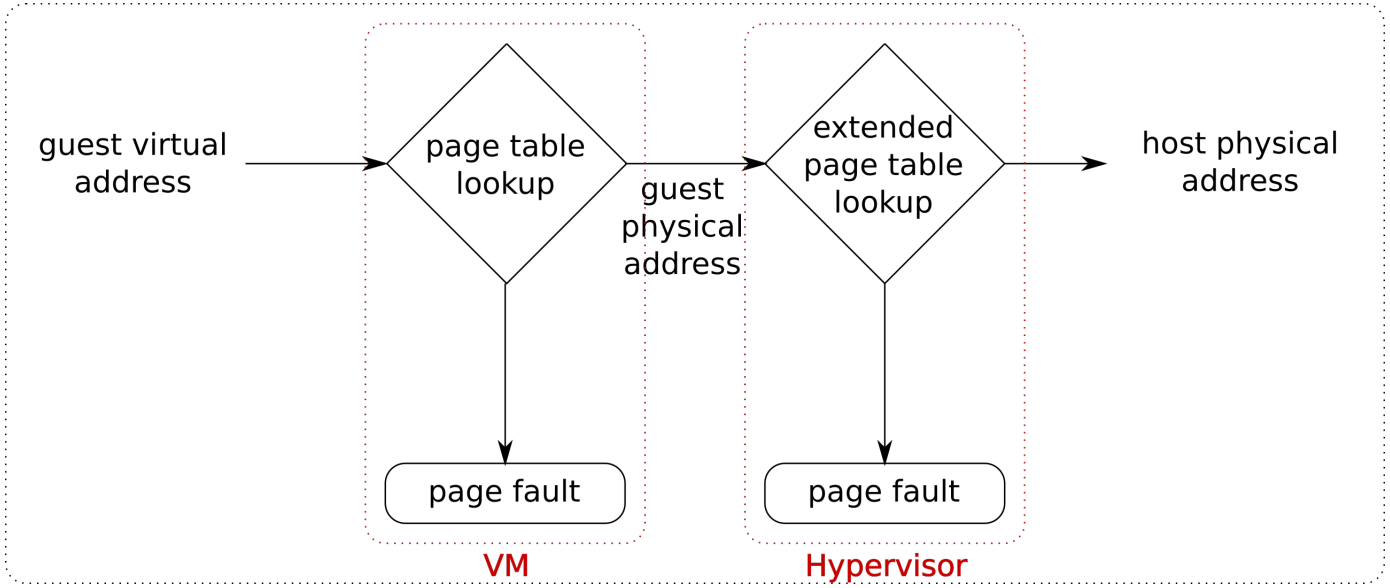
How do extended page tables work

- Adds another layer:
  - PT: guest-virtual addr. → guest-phys. addr.
  - EPT: guest-phys. addr. → host-phys. addr.



How do extended page tables work

- Adds another layer:
  - PT: guest-virtual addr. ➞ guest-phys. addr.
  - EPT: guest-phys. addr. ➞ host-phys. addr.



Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-VMM

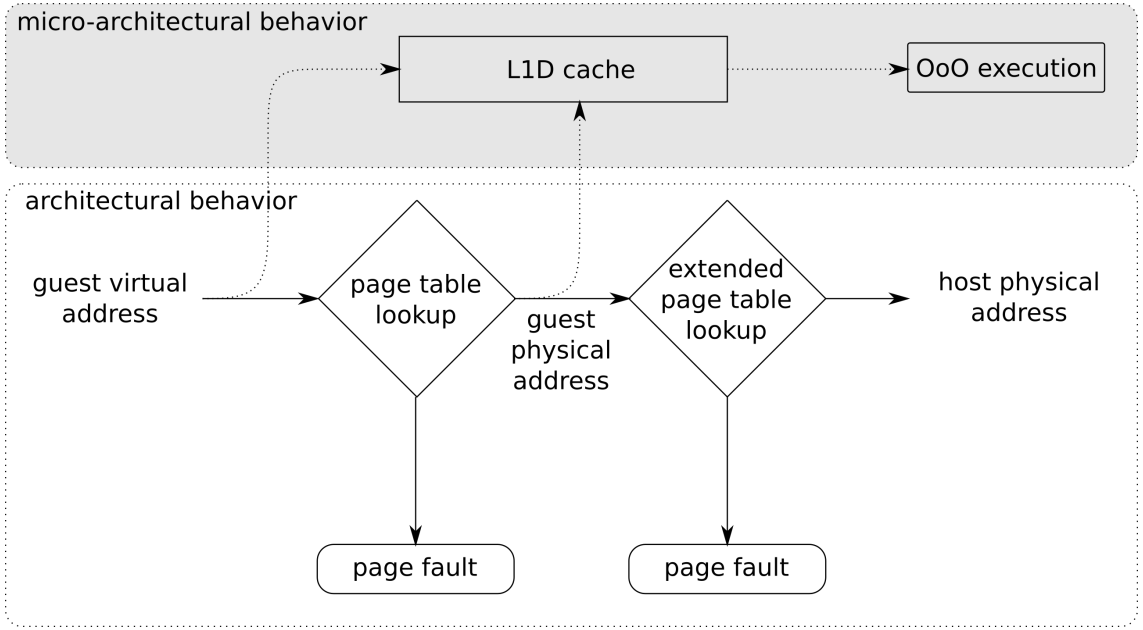
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

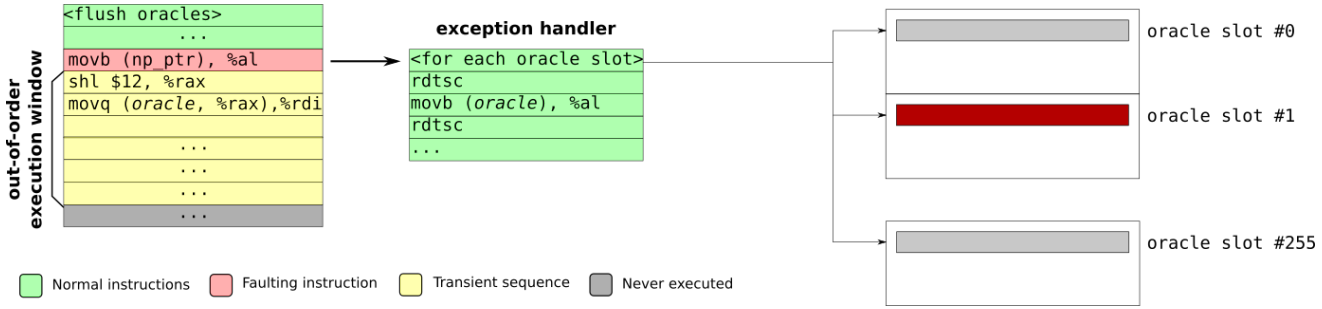
Source: xkcd.com

The Security Flaw: Interpreting guest-physical as host-physical addresses

- VM-level: non-present PTE entry
- VMM-level: irrelevant
- Upon access:
  - Tag data access as a violation
  - Pass *guest-physical* as *host-physical* address to L1D cache
  - Continue transient execution!!
- ⇝ This breaks the VM's address space abstraction!



Foreshadow-VMM: The exploit

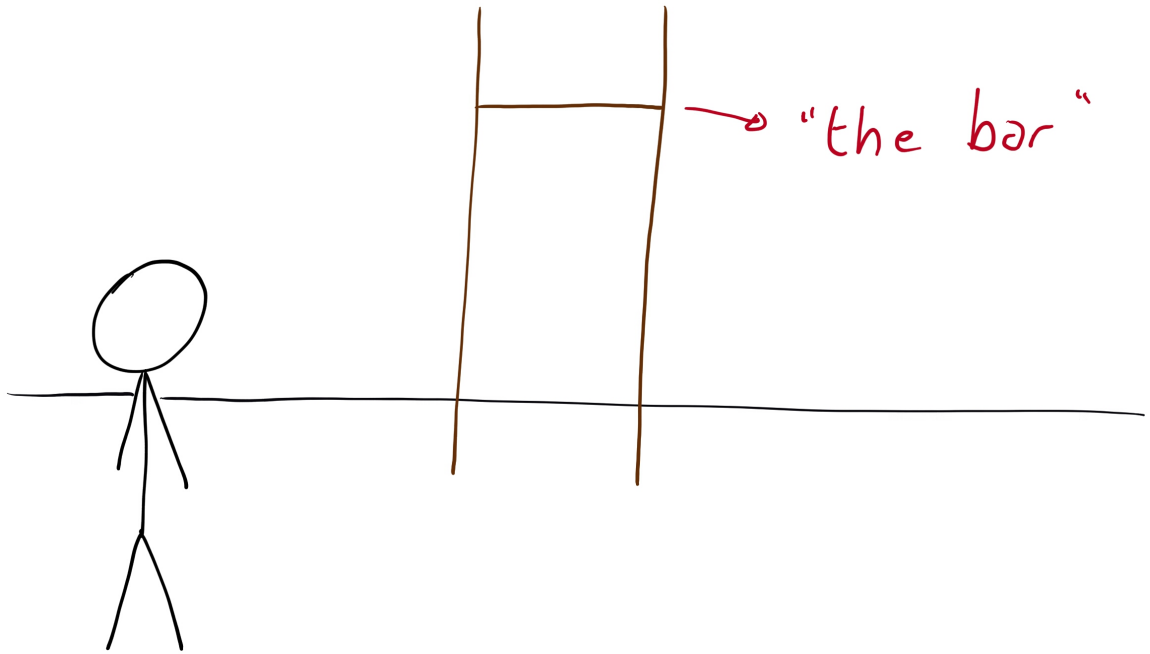



```
1  int8_t *oracles = ...;
2  int8_t *np_ptr = ...;
3  int8_t _tmp;
4
5  // Step 1: Setup PT to physical address of interest
6
7  // Step 2: Remove oracle slots from cache
8  for ( int i = 0; i < 256; ++i )
9    clflush( &oracles[4096 * i] );
10
11 // Step 3: Wait for sensitive data in L1D
12
13 // Step 4: attempt to read not present memory
14 _tmp = oracle[4096 * (*np_ptr)];
15
16 // suppress fault
17
18 // Step 5: is oracle cached?
19 for ( int i = 0; i < 256; ++i ) {
20   if ( time_access( oracle[4096 * i] ) < 146 )
21     print( "*np_ptr = %i\n", i );
22 }
```

Impact of Foreshadow-VMM

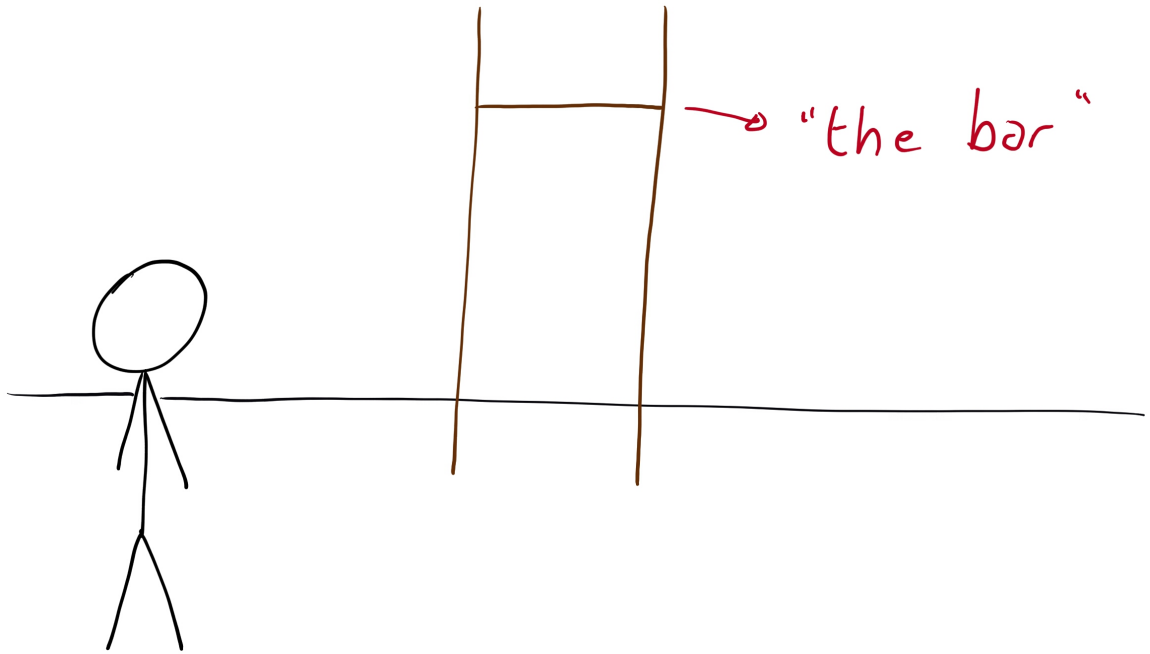
- Requirements:
  - Attacker must have full VM under her control
  - Secret data must reside in L1D

- ⇝ Modest impact!



Impact of Foreshadow-VMM

- Requirements:
  - Attacker must have full VM under her control
  - Secret data must reside in L1D
    → **This may not be that complicated!!**
- ⇝ Modest impact!



Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-VMM

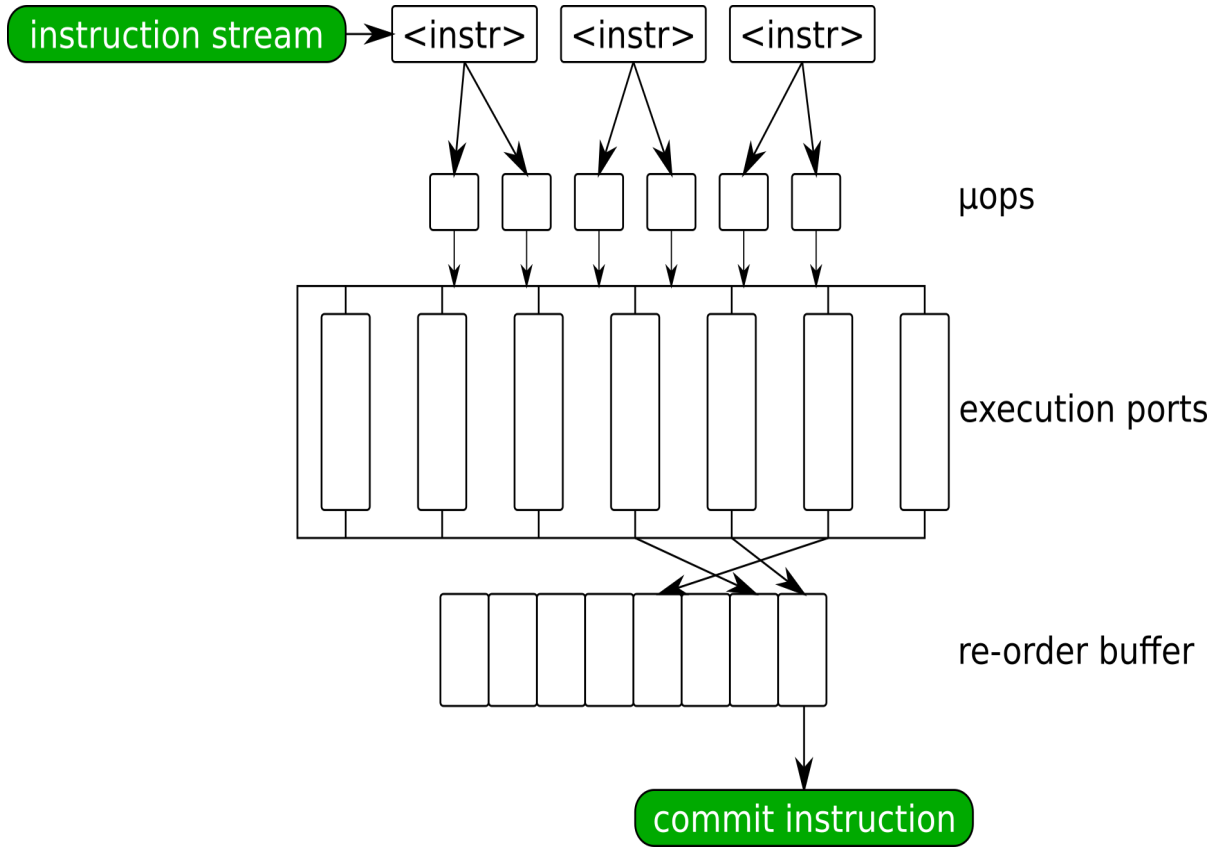
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

Intel HyperThreading as an Enabler

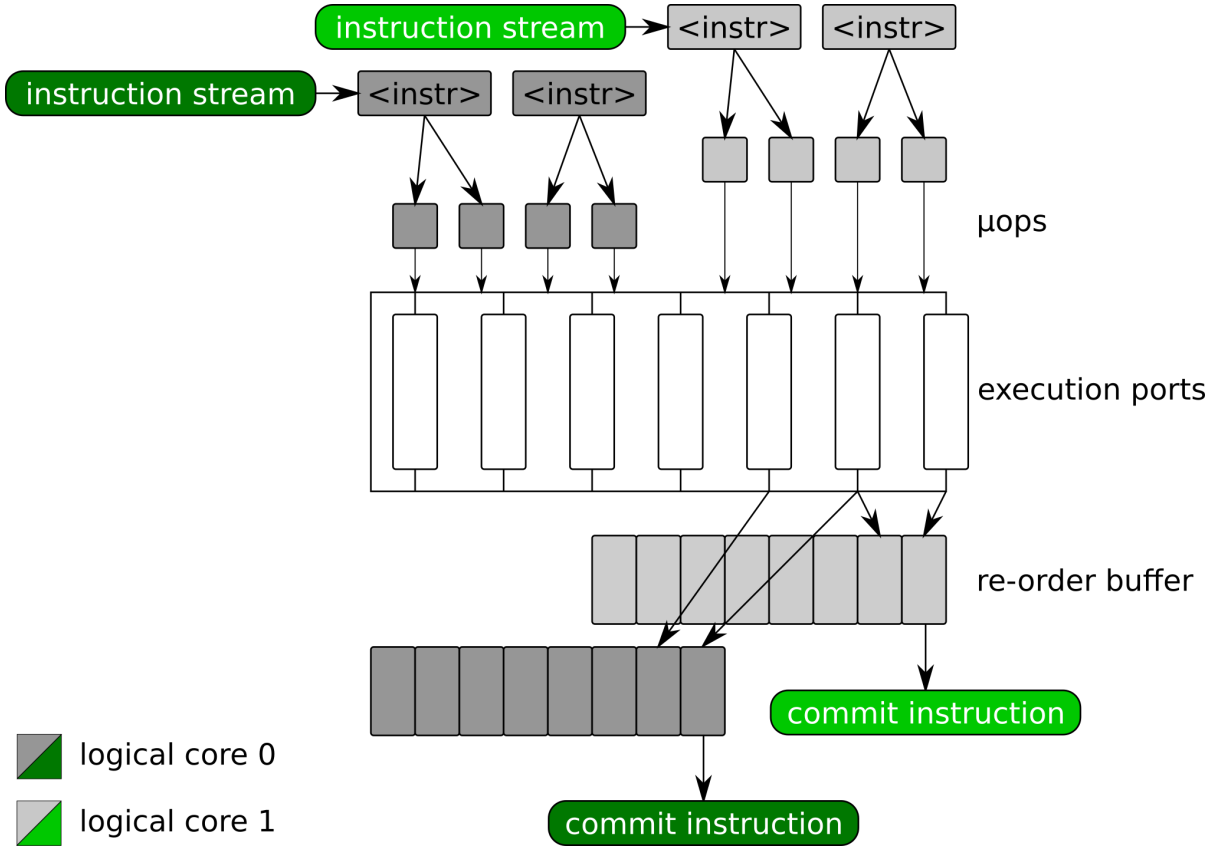
- **Problem**: Execution ports are still under-utilized
- **Solution**: Split physical core in two, duplicate
  - register file
  - re-order buffer
- Shared:
  - Execution ports
  - **L1 cache!** (and other levels)
- ⇝Up to 30% performance increase



source: https://www.cs.sfu.ca/~fedorova/Teaching/CMPT886/Spring2007/papers/hyper-threading.pdf

Intel HyperThreading as an Enabler

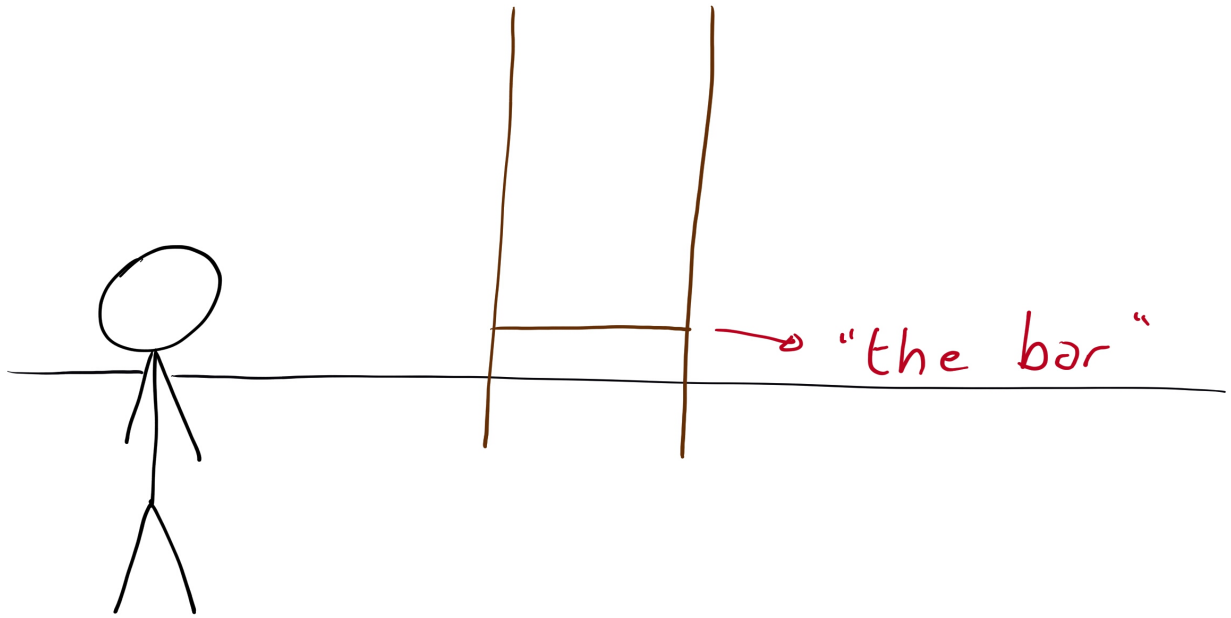
- **Problem**: Execution ports are still under-utilized
- **Solution**: Split physical core in two, duplicate:
  - register file
  - re-order buffer
- Shared:
  - Execution ports
  - **L1 cache!** (and other levels)
- ⇝Up to 30% performance increase



source: https://www.cs.sfu.ca/~fedorova/Teaching/CMPT886/Spring2007/papers/hyper-threading.pdf

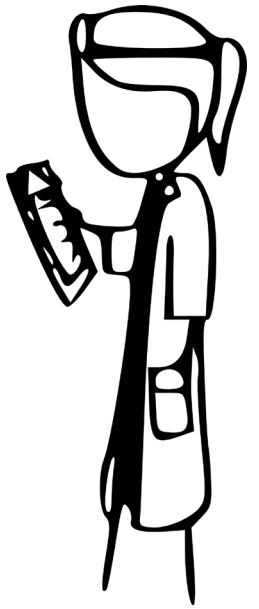
Impact of Foreshadow-VMM

- Requirements:
  - Attacker must have full VM under her control
  - Secret data must reside in L1D
    → **Just have a little bit of patience!**
- 〰 High impact!



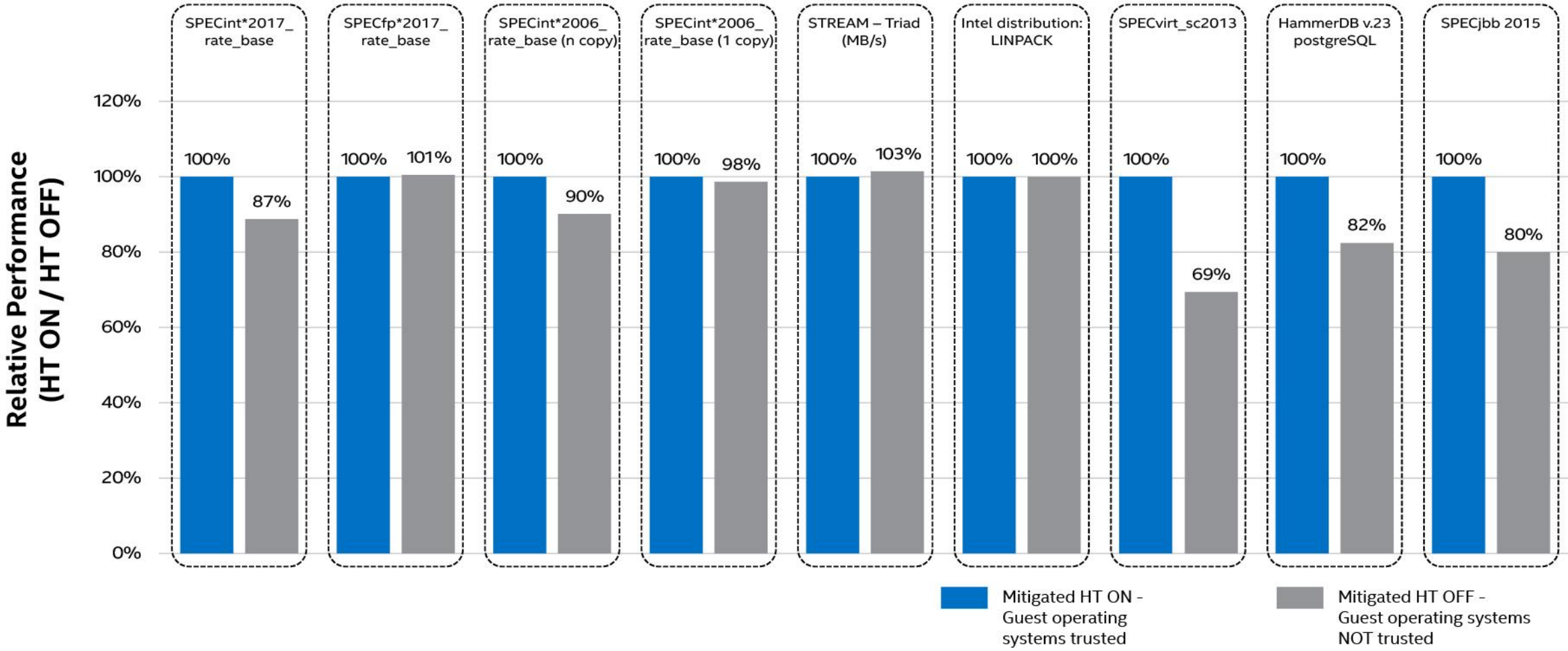
Mitigations

- Long term: Replace chips!
- Short term:
  - Make sure no secrets are in L1D cache
  - → Flush L1D before upon VM-entry
  - → Make sure no two different VMs execute on same physical core
    - Disable HyperThreading
    - Patch VM scheduler



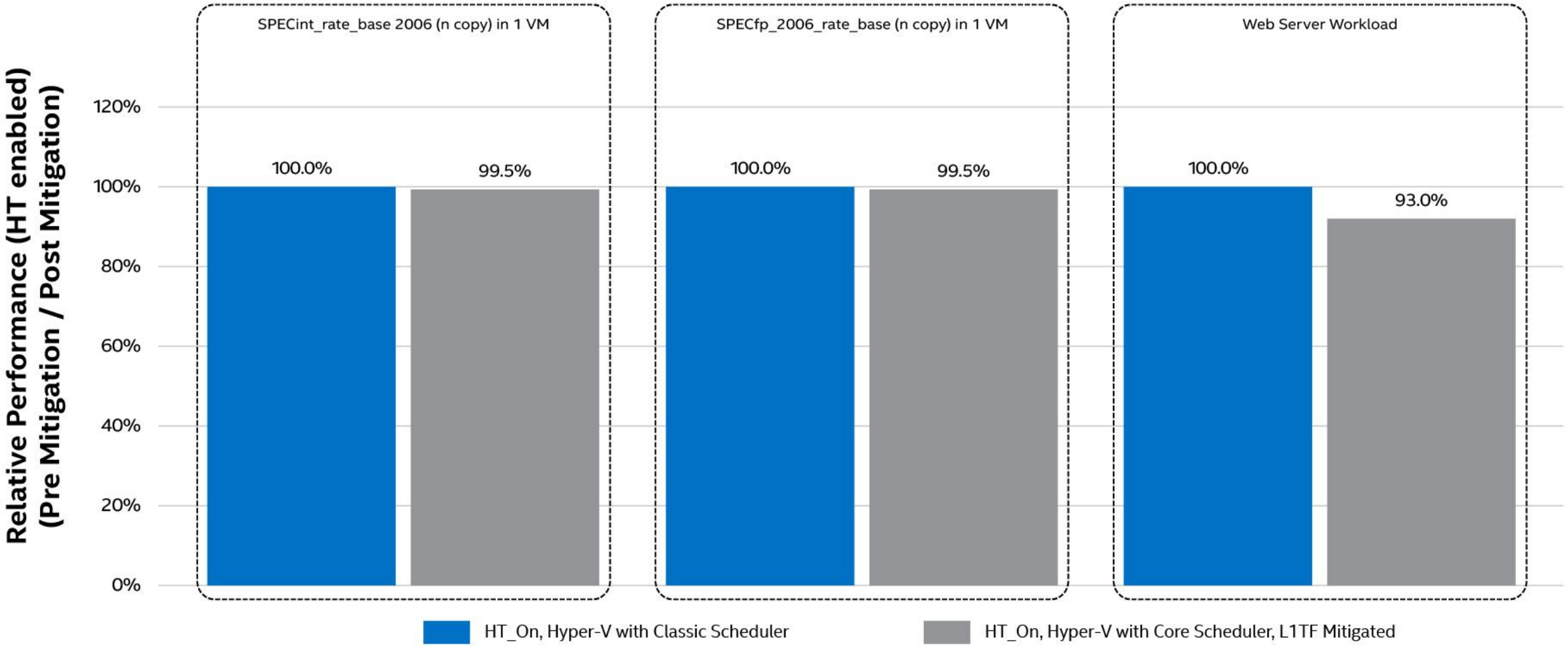
Source: xkcd.com

Mitigations – Disabling HyperThreading



source: https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

Mitigations – Update VM scheduler



source: https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

Foreshadow-SGX: Dismantling Intel SGX's security objectives

Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-SGX

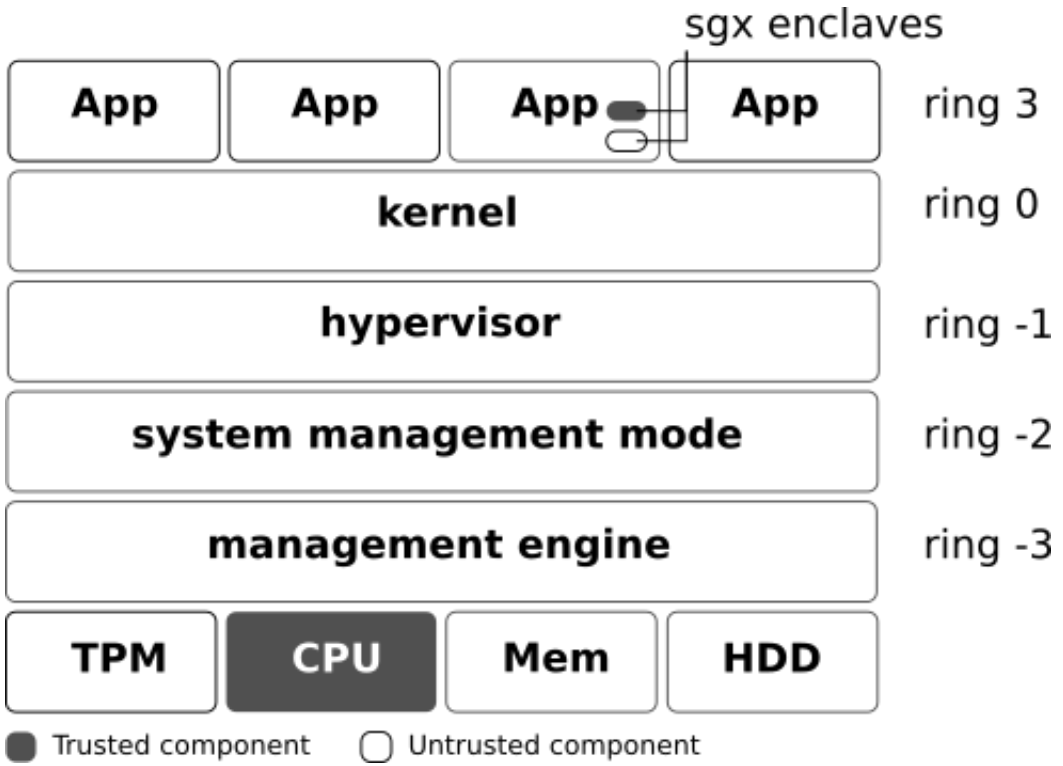
SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

Background: Intel SGX

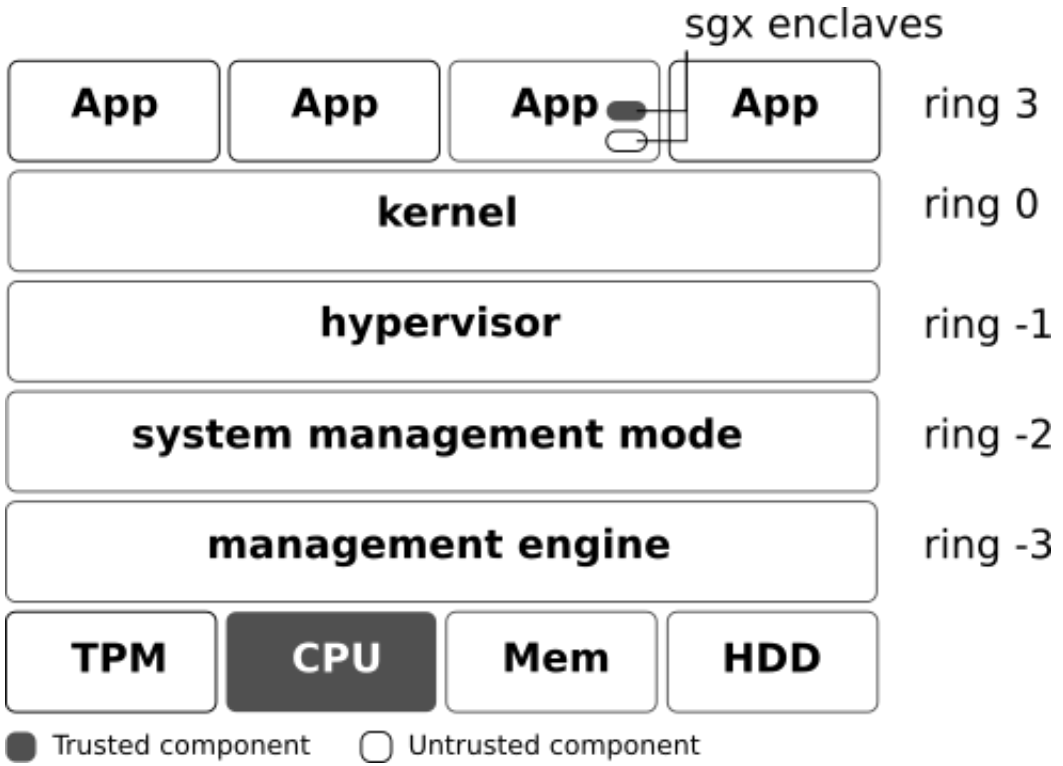
- **Problem**: Huge software TCB
- **Solution**: Protected-Module Architecture (e.g., Intel SGX)
- Only trust Intel hardware/enclaves
- Use cases:
  - protecting finger prints,
  - DRM,
  - password vault,
  - …



Background: Intel SGX

3 key properties:

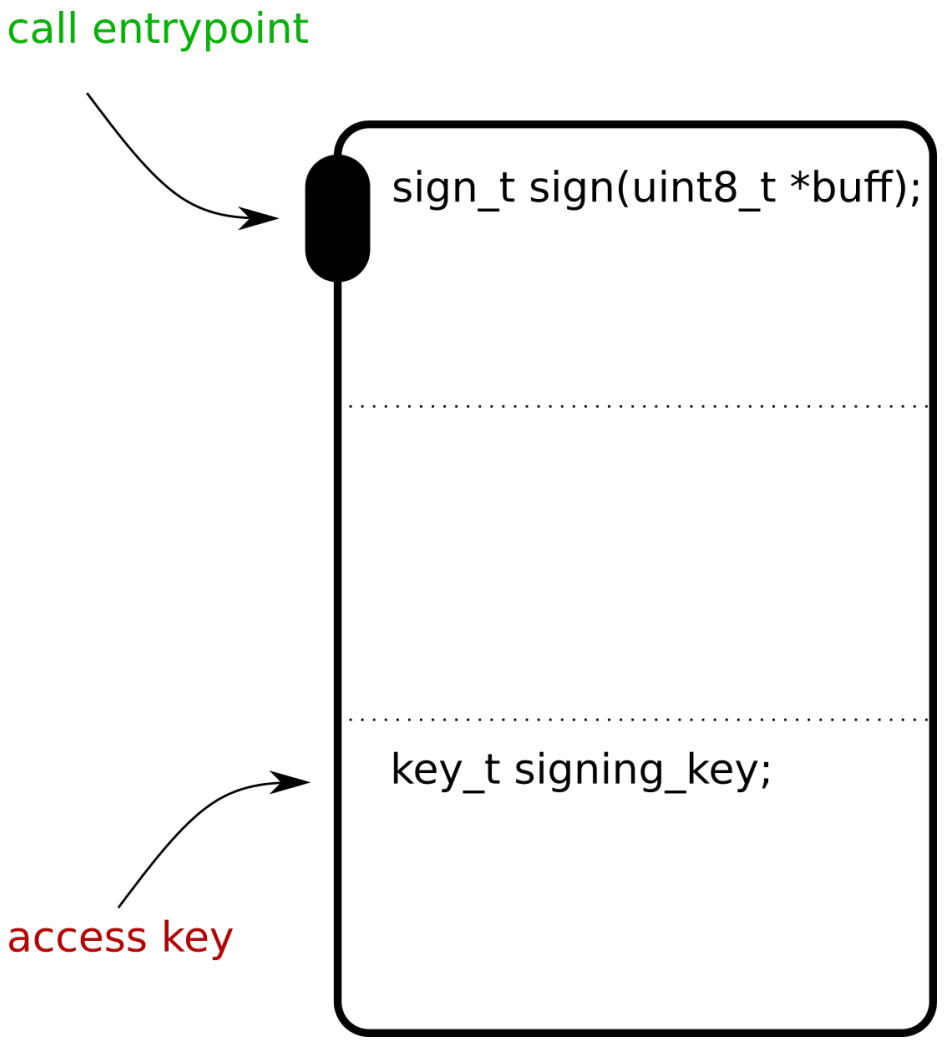
- Isolation
- Secure Storage
- Attestation



Background: Intel SGX

Isolation

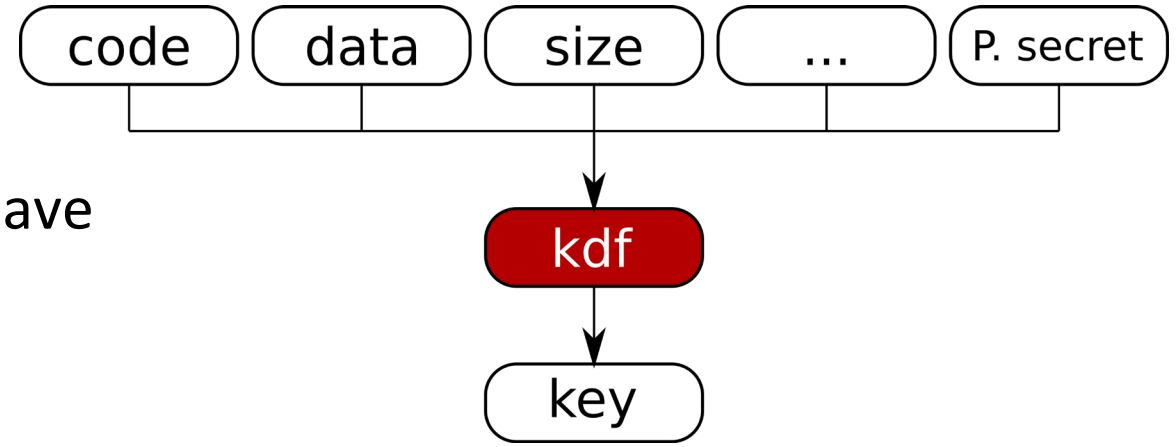
- Enclaves live in process' address space
- Only accessible through specific entry points
- Abort page semantics: Reading enclave memory outside the enclave results in -1.



Background: Intel SGX

Secure Storage:

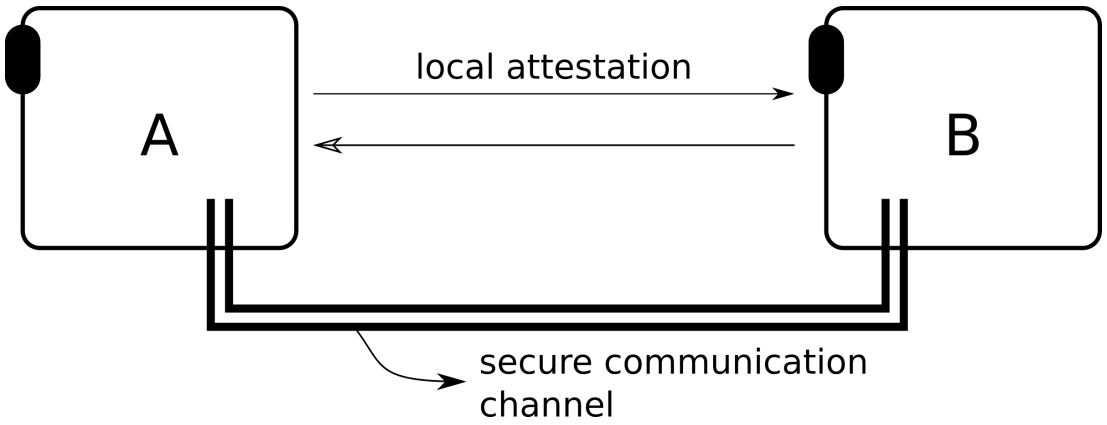
- Enclaves die at power off
- Seal/Unseal confidential data
- Key derivation ensures unique key per enclave
- ⇝ Derived keys stored in enclave memory!



Background: Intel SGX

Attestation:

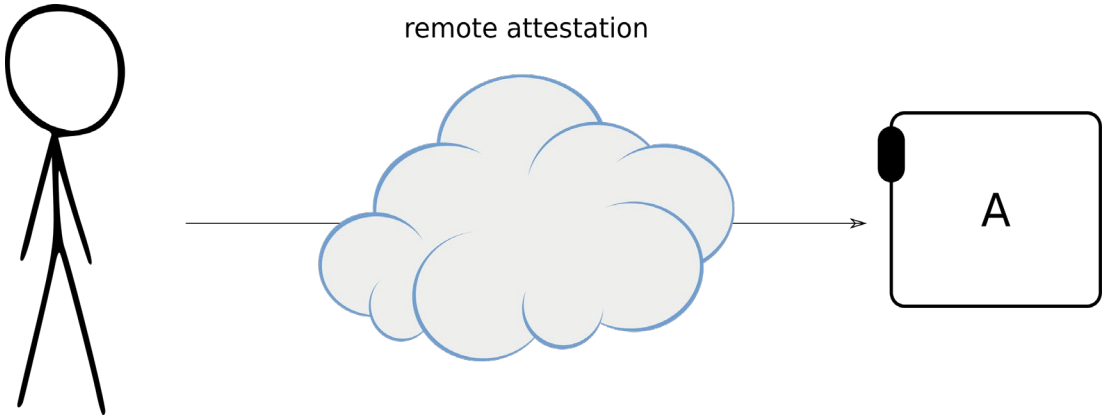
- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as a building block for remote attestation



Background: Intel SGX

Attestation:

- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as a building block for remote attestation



Side-Channel Attacks

Attacker

Victim

Action: manipulate PT

Carrier: cache changes

Foreshadow-SGX

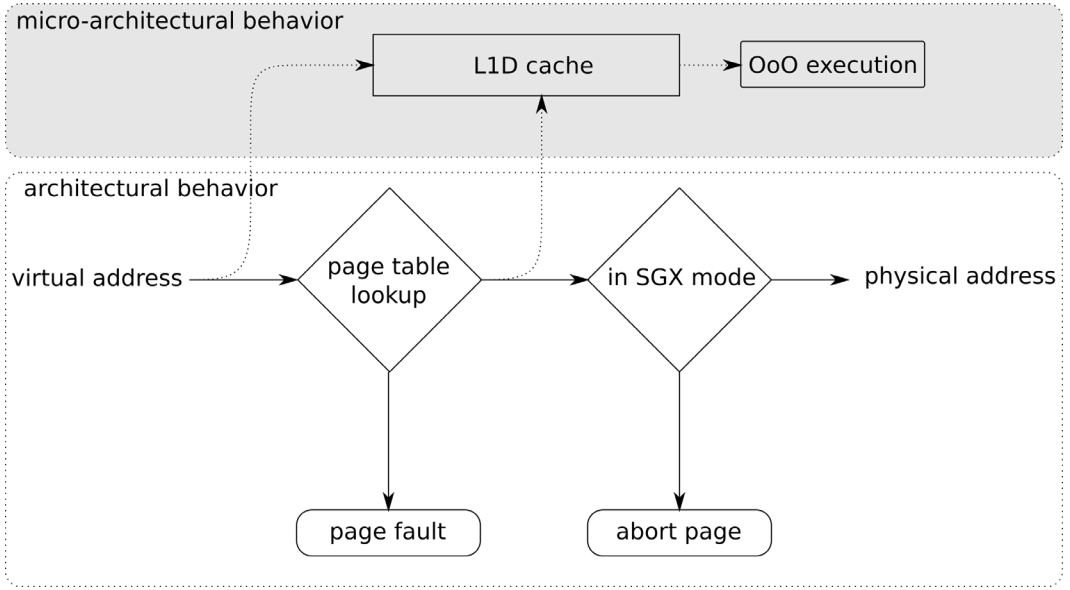
SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

Source: xkcd.com

The Attack Approach

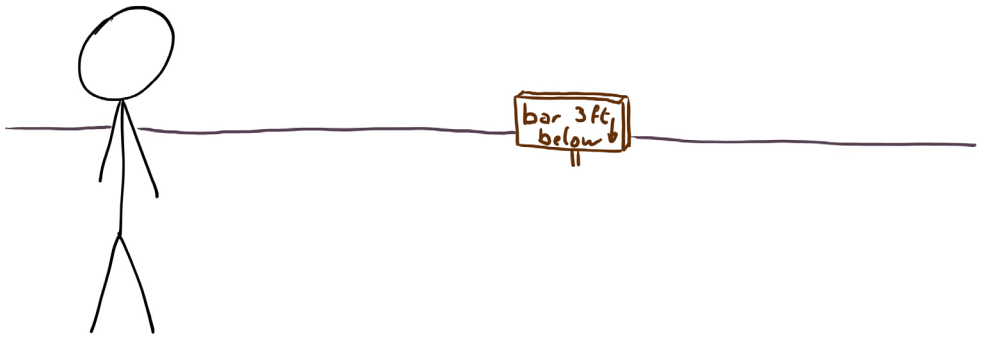
- Bypass abort page semantics
- Ensure data in L1D:
  - Single-step through enclave
  - *eldu* instruction as a side effect loads enclave page in L1D!



Impact of Foreshadow-SGX

- Requirements:
  - Mark enclave page not-present
  - Call enclave/issue “special” SGX instruction

  ⇝ Completely breaks enclave isolation, sealed storage and remote/local attestation



Conclusion

- We were lucky Foreshadow was responsibly disclosed
- Foreshadow breaks the virtual memory abstraction
- Modern x86 processors have become too complex to completely understand
- Need to completely rethink past design decisions

RSAConference2019

Questions?

Raoul Strackx
raoul.strackx@cs.kuleuven.be
@raoul_strackx

RSAConference2019