

# Foreshadow: speculative attacks on SGX and beyond

Mark Silberstein

Joint work with

Jo Van Bulck, **Marina Minkin**, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens,  
Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx

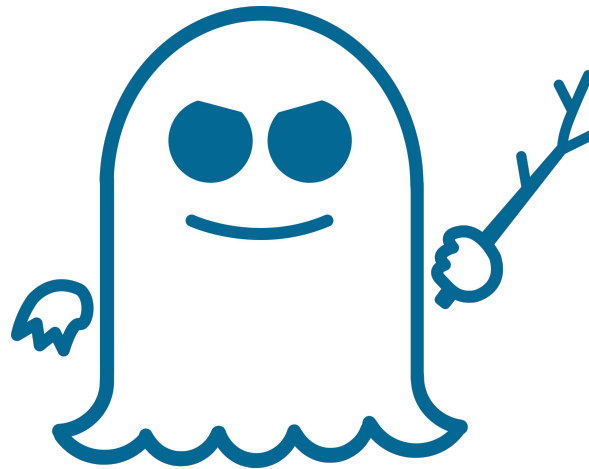


# Big picture in one slide

- Where do CPUs lose performance?
  - Branches, Memory translation
  - Technology scaling does not help

# Big picture in one slide

- Where do CPUs lose performance?
  - Branches, Memory translation
  - Technology scaling does not help
- Speculative execution for latency hiding
  - CPU speculates the outcome of slow operations
  - **Continues** execution assuming speculation is correct
  - **Rolls back** the modified architectural state otherwise



## Speculative execution attacks exploit

- *Speculation* past illegal memory accesses
- *Inability* to fully roll back  $\mu$ arch state
- *Covert/side channels* to leak the state

# Today

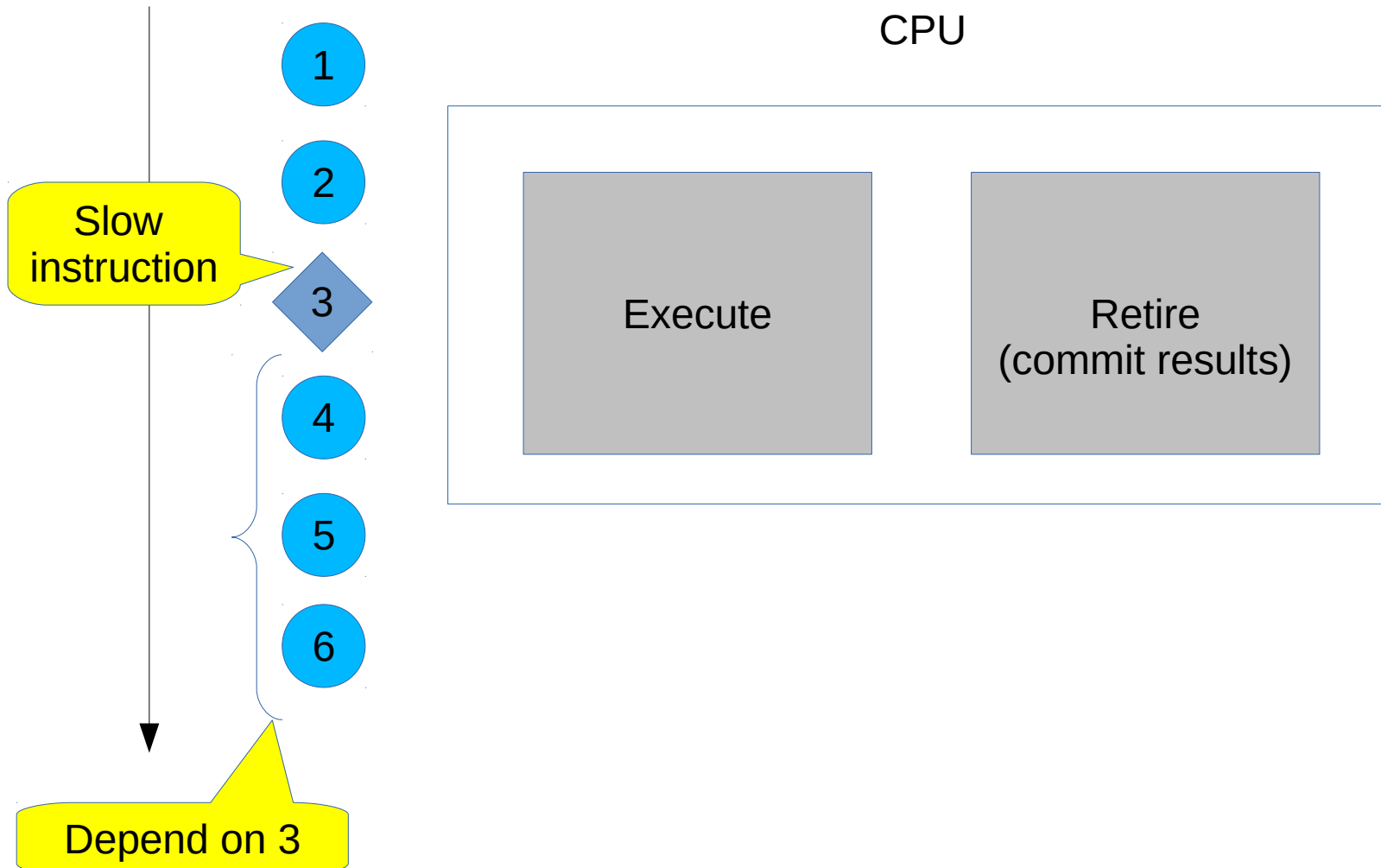
- Background
- From Meltdown to Foreshadow
- SGX: Collateral damage
- Foreshadow-NG (L1TF)
- Discussion

# Speculative execution 101

Instruction stream

Completed instructions

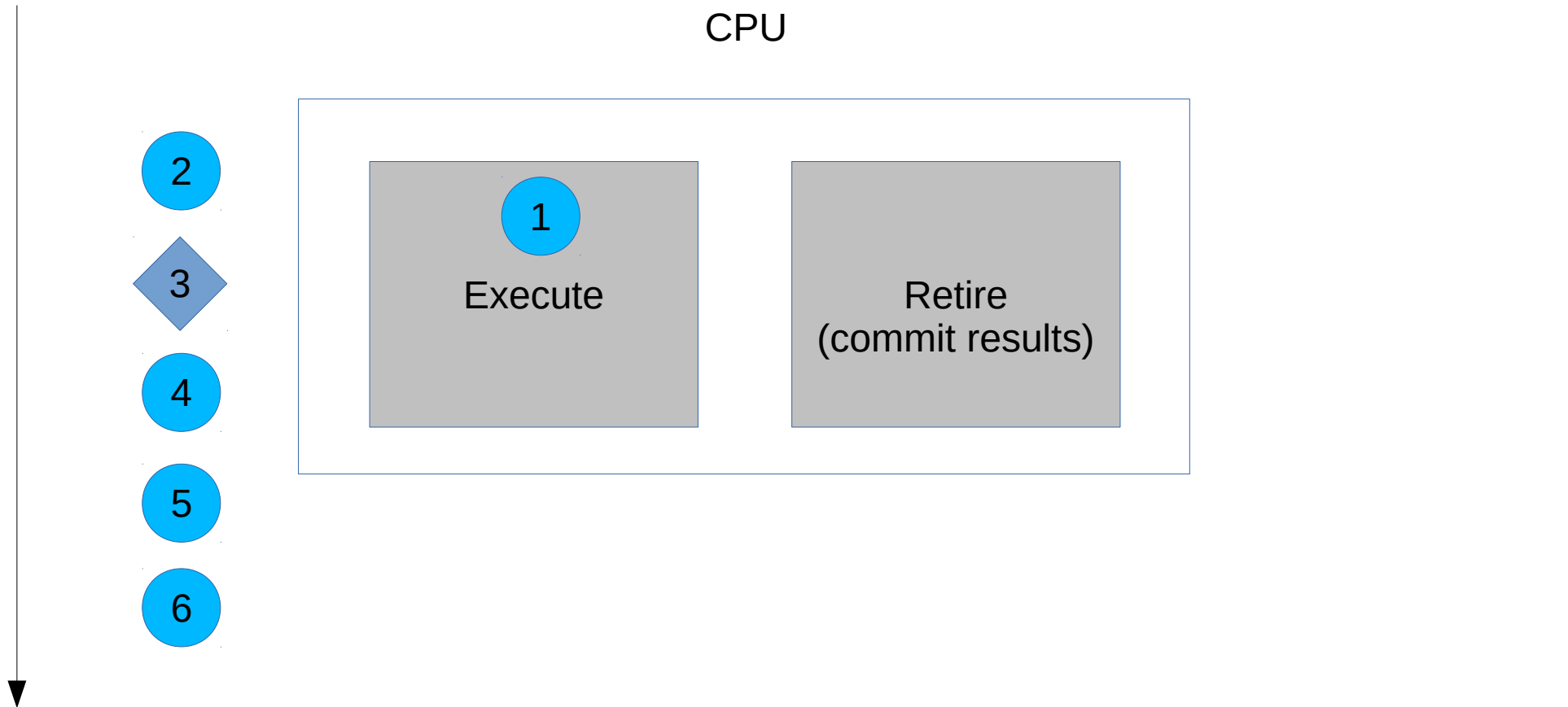
CPU



# Speculative execution 101

Instruction stream

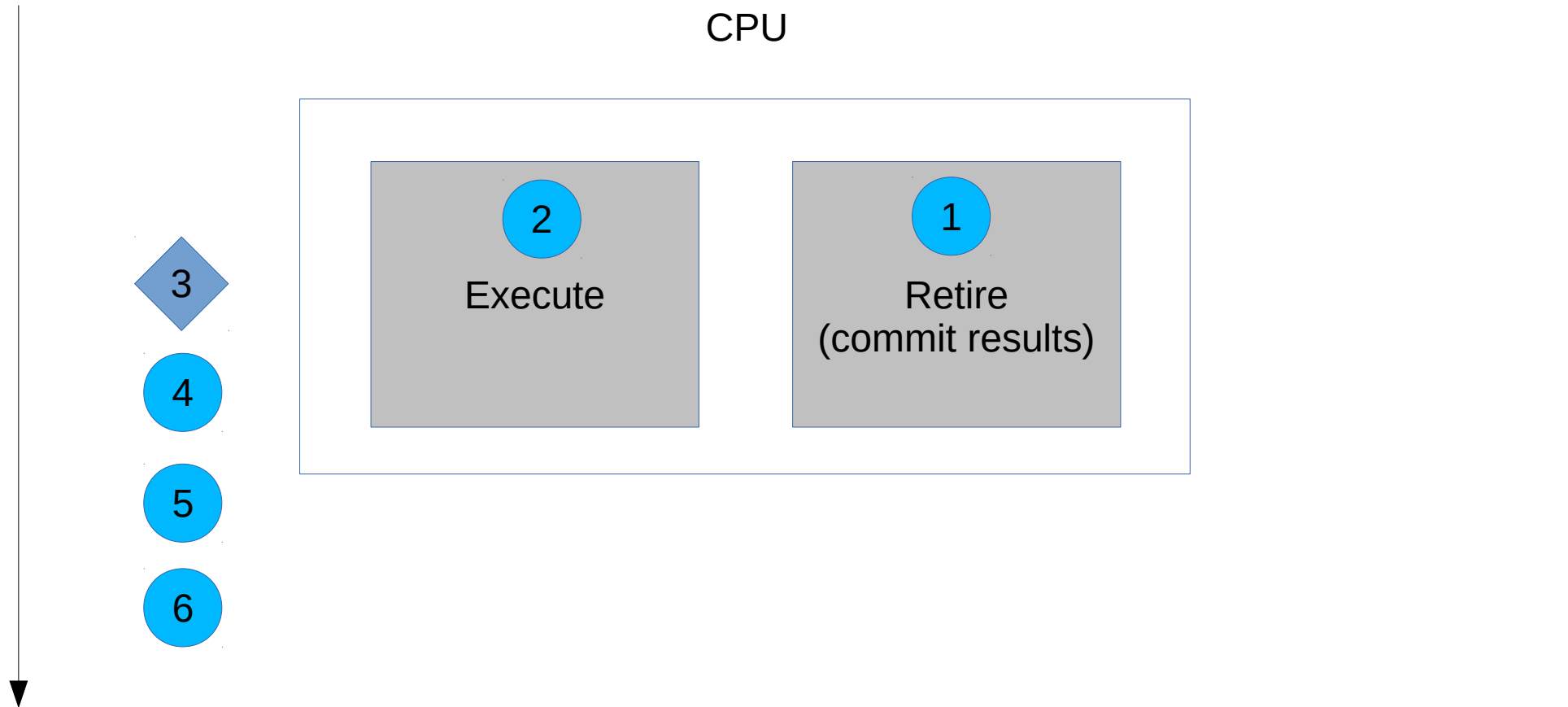
Completed instructions



# Speculative execution 101

Instruction stream

Completed instructions

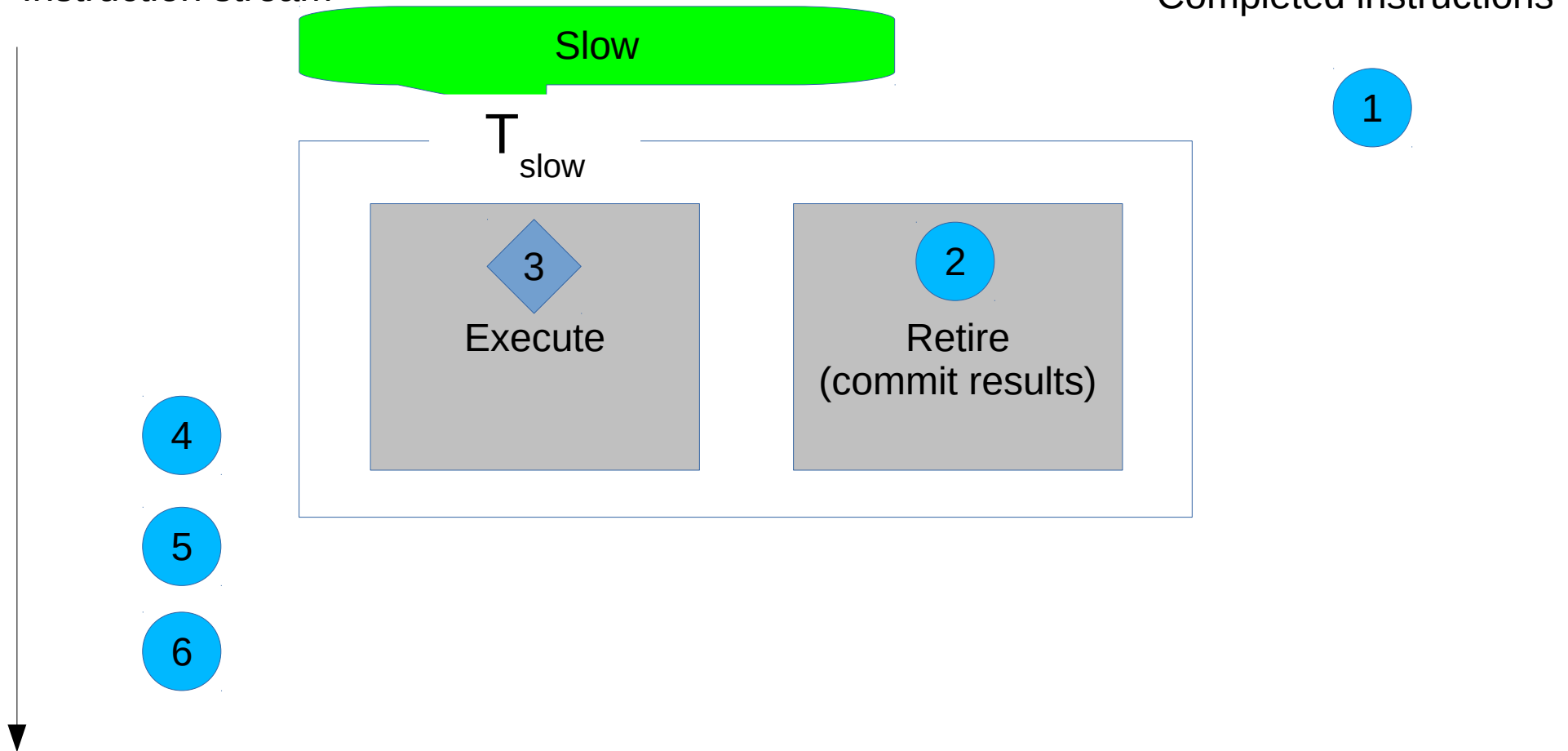




# Speculative execution 101

Instruction stream

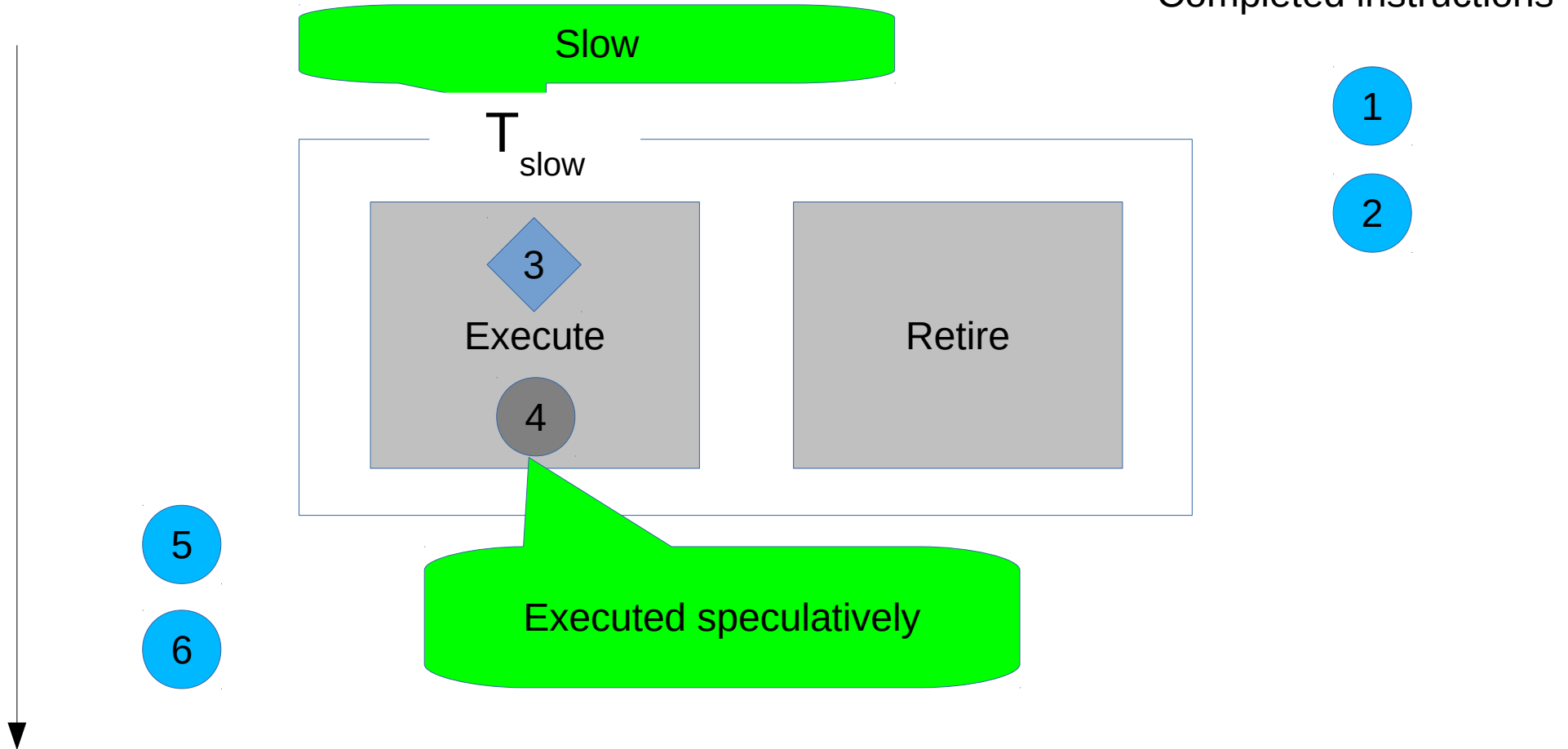
Completed instructions



# Speculative execution 101

Instruction stream

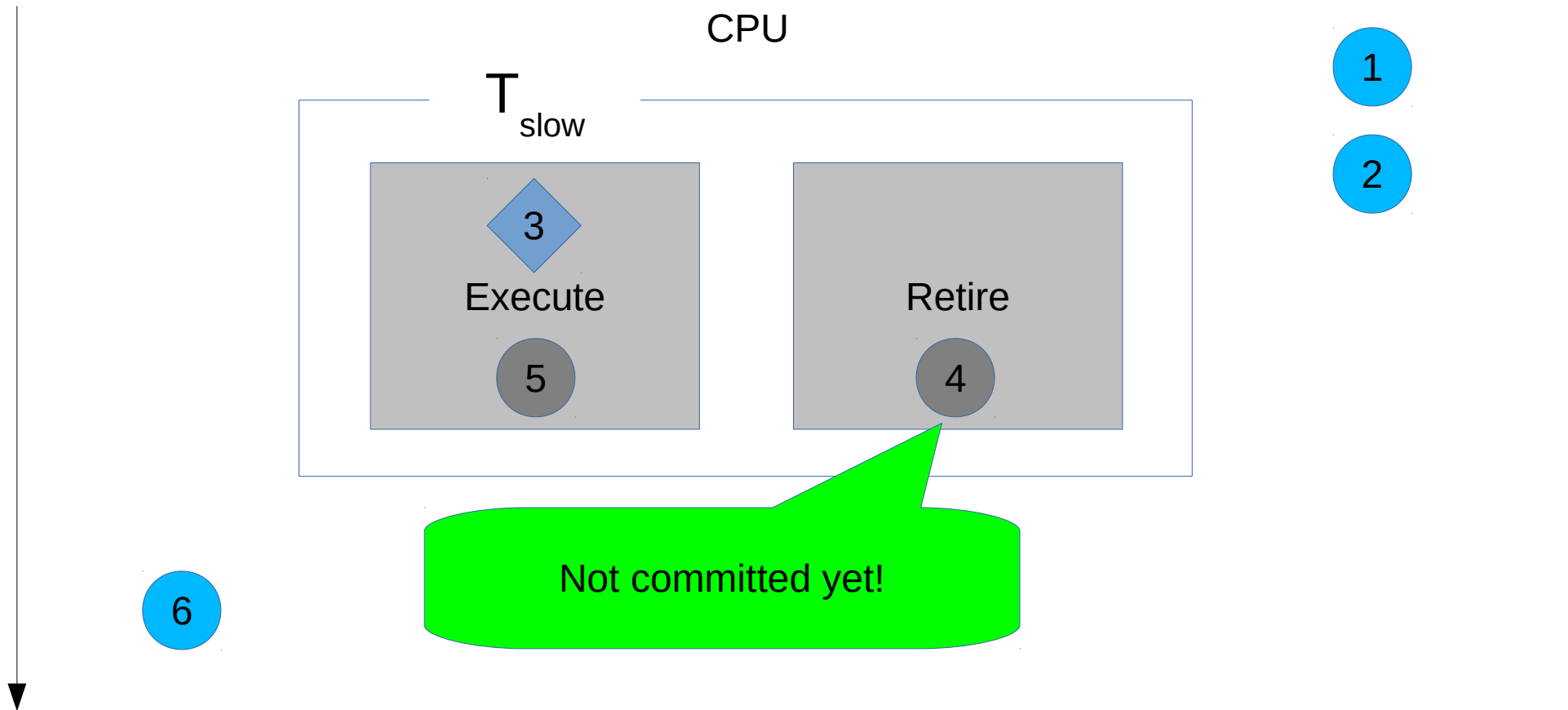
Completed instructions



# Speculative execution 101

Instruction stream

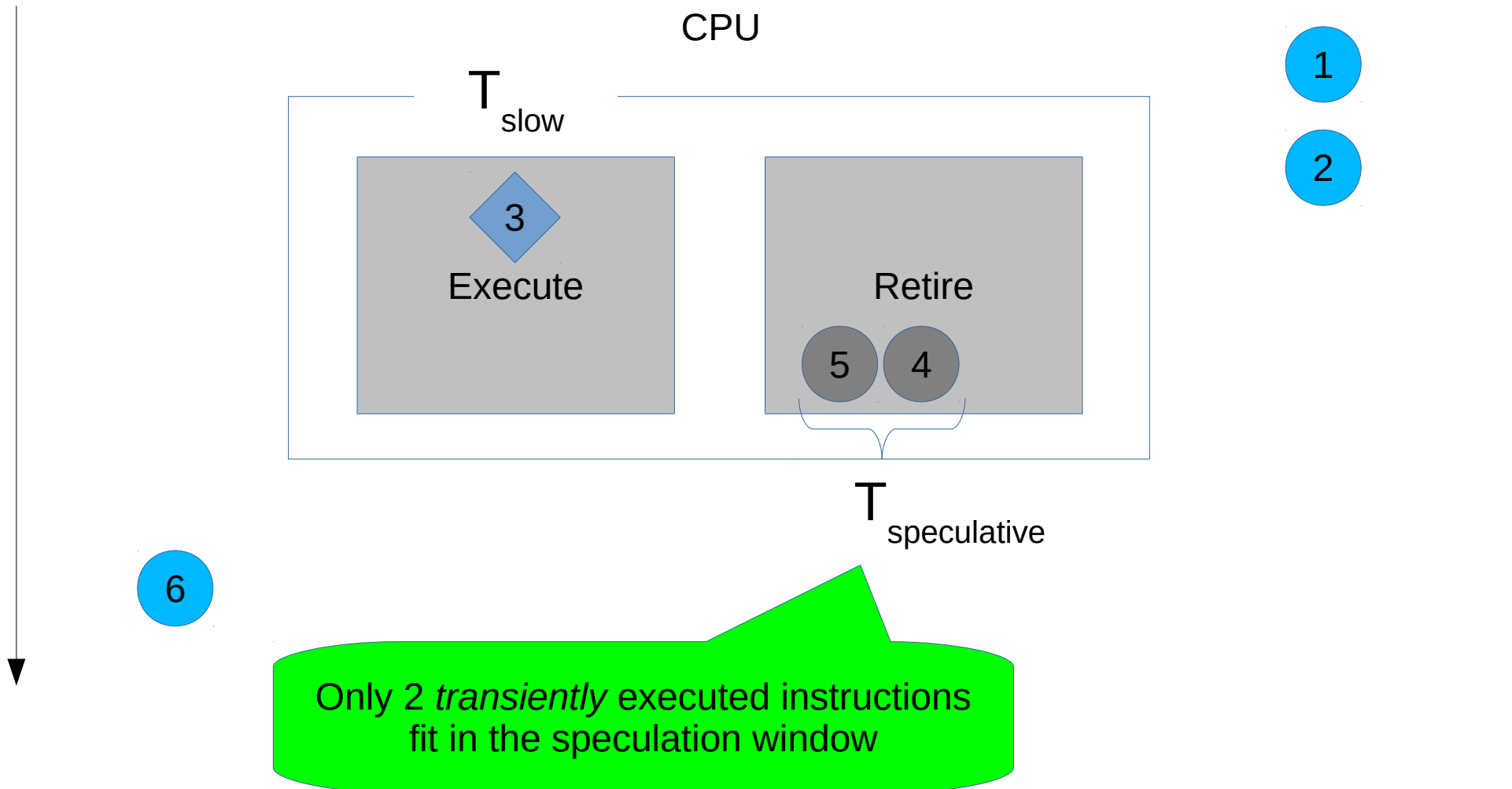
Completed instructions



# Speculative execution 101

Instruction stream

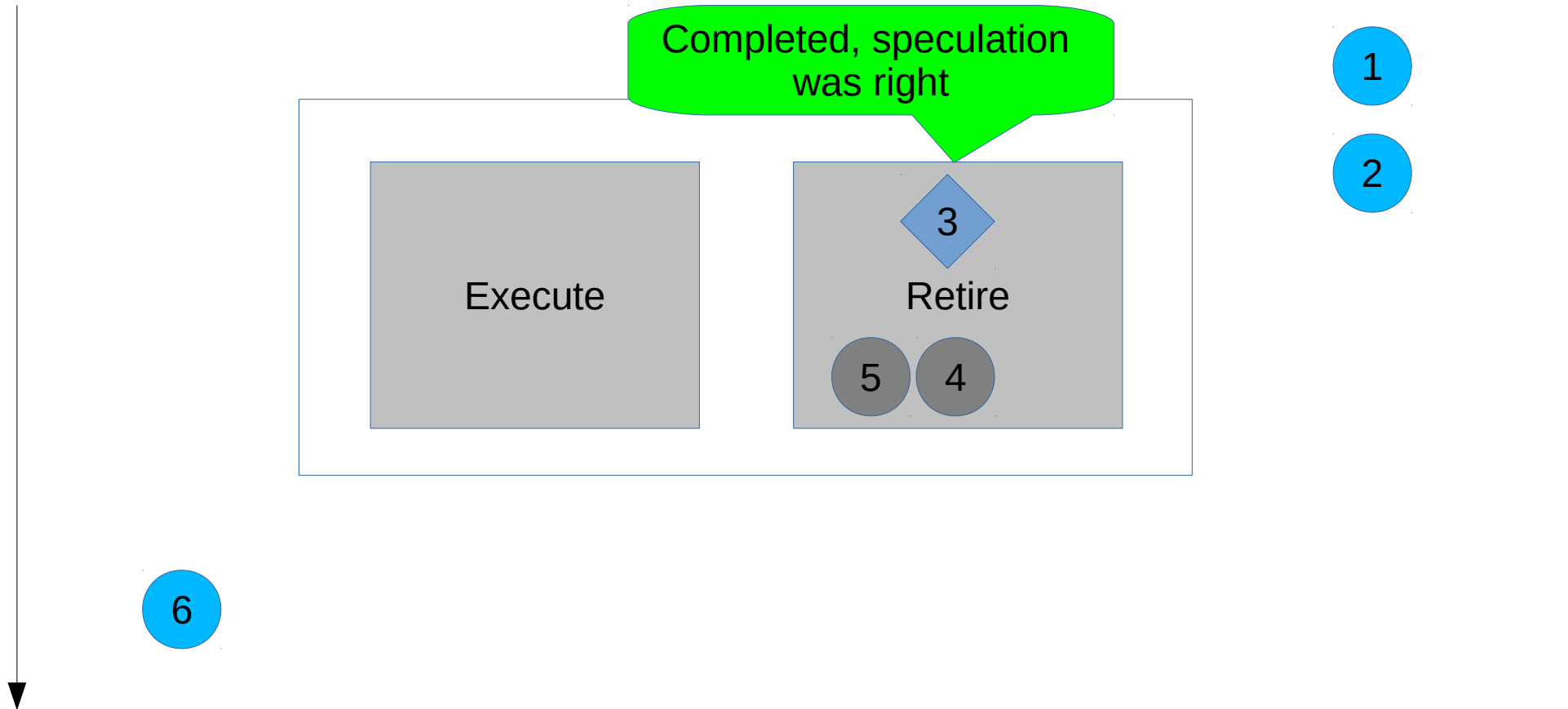
Completed instructions



# Speculative execution 101

Instruction stream

Completed instructions

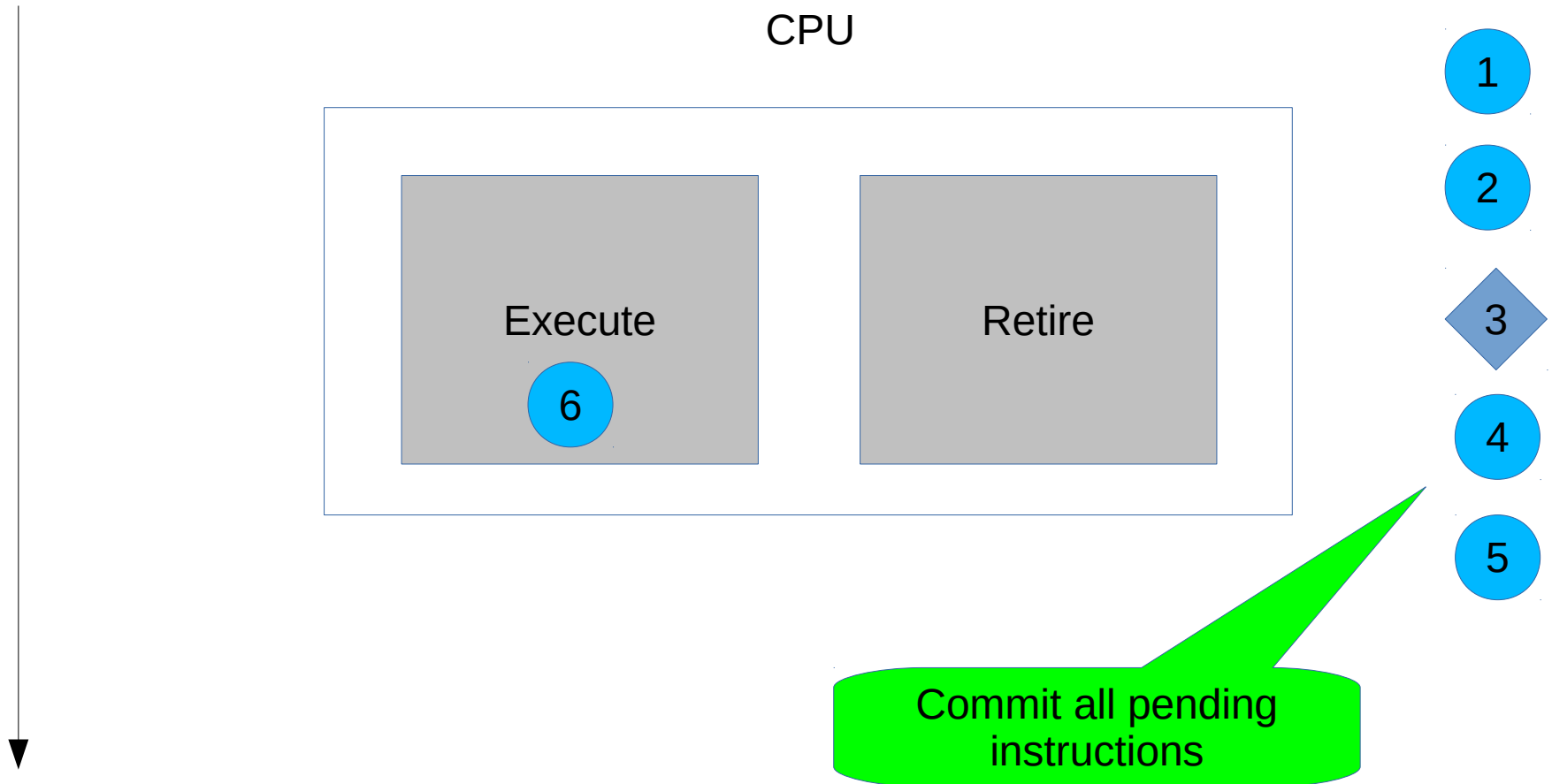


# Speculative execution 101

Instruction stream

Completed instructions

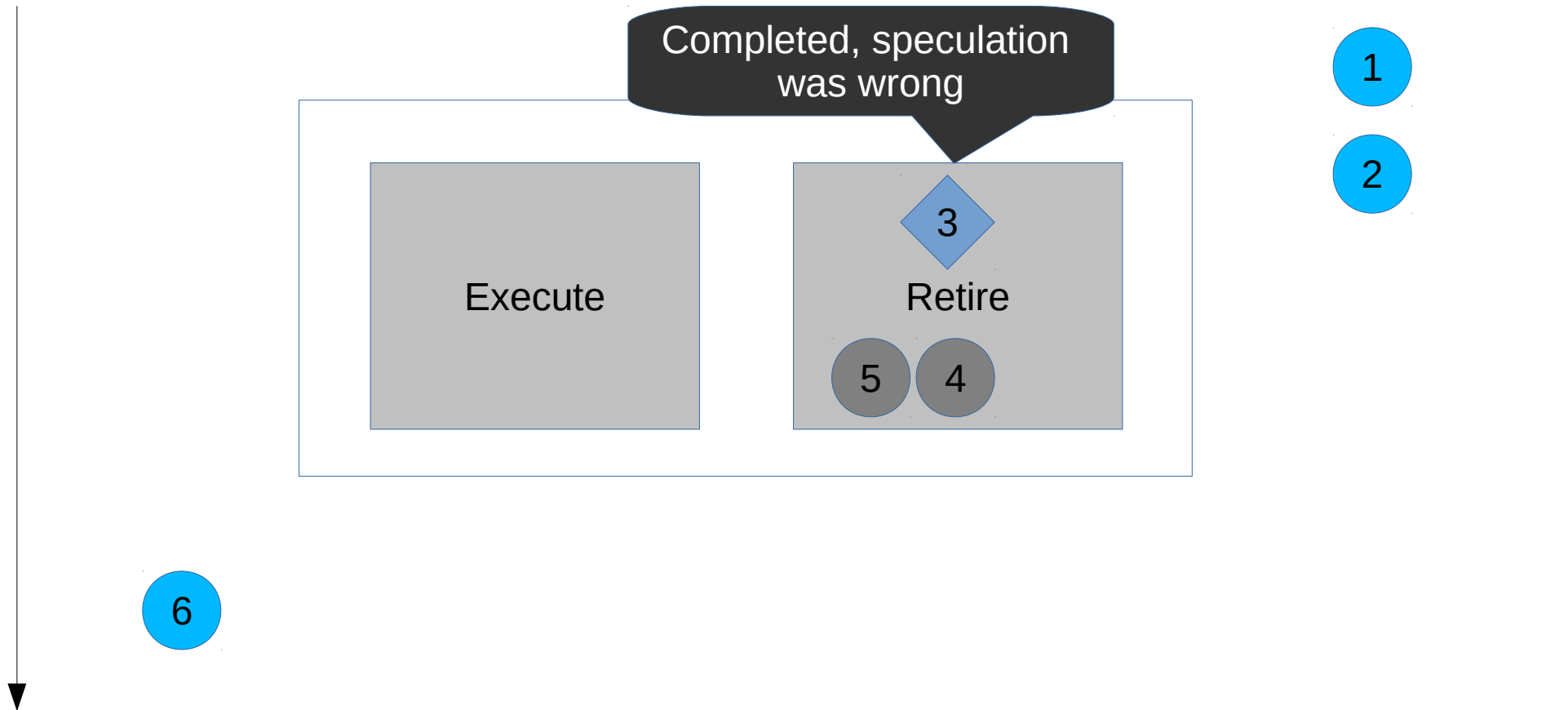
CPU



# Speculative execution 101

Instruction stream

Completed instructions

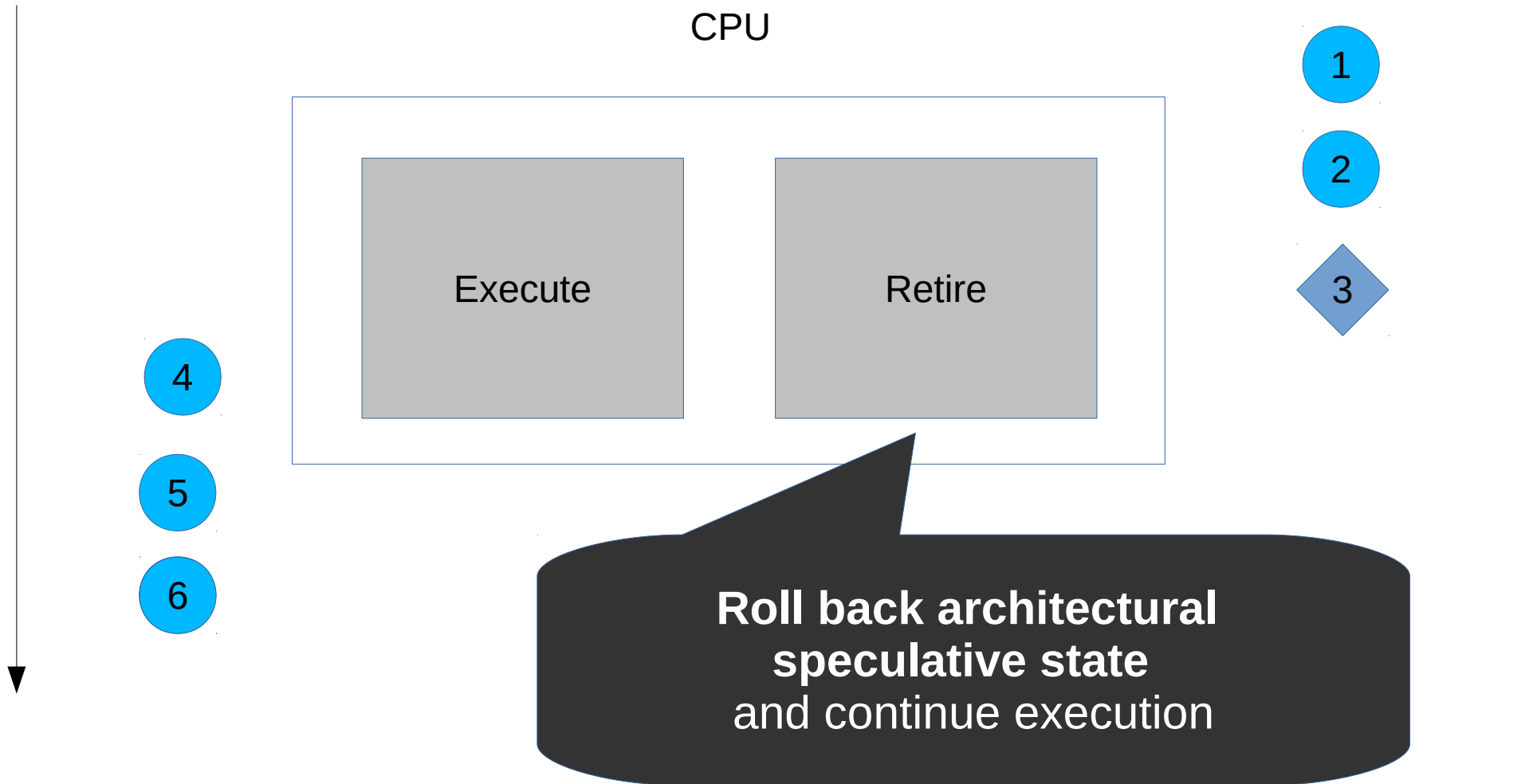


# Speculative execution 101

Instruction stream

Completed instructions

CPU





# Prerequisites to speculative execution attack

- CPU speculates *insecurely*
- Speculative state **cannot be** rolled back: **data leak**
- Race condition: roll back vs. leaking logic
  - Attack succeeds only if  $T_{\text{speculative}} < T_{\text{slow access}}$

# Complete example (Rogue cache read – aka Meltdown)

Instruction stream

Transient instructions

3

```
movb (kernel secret), %al
```

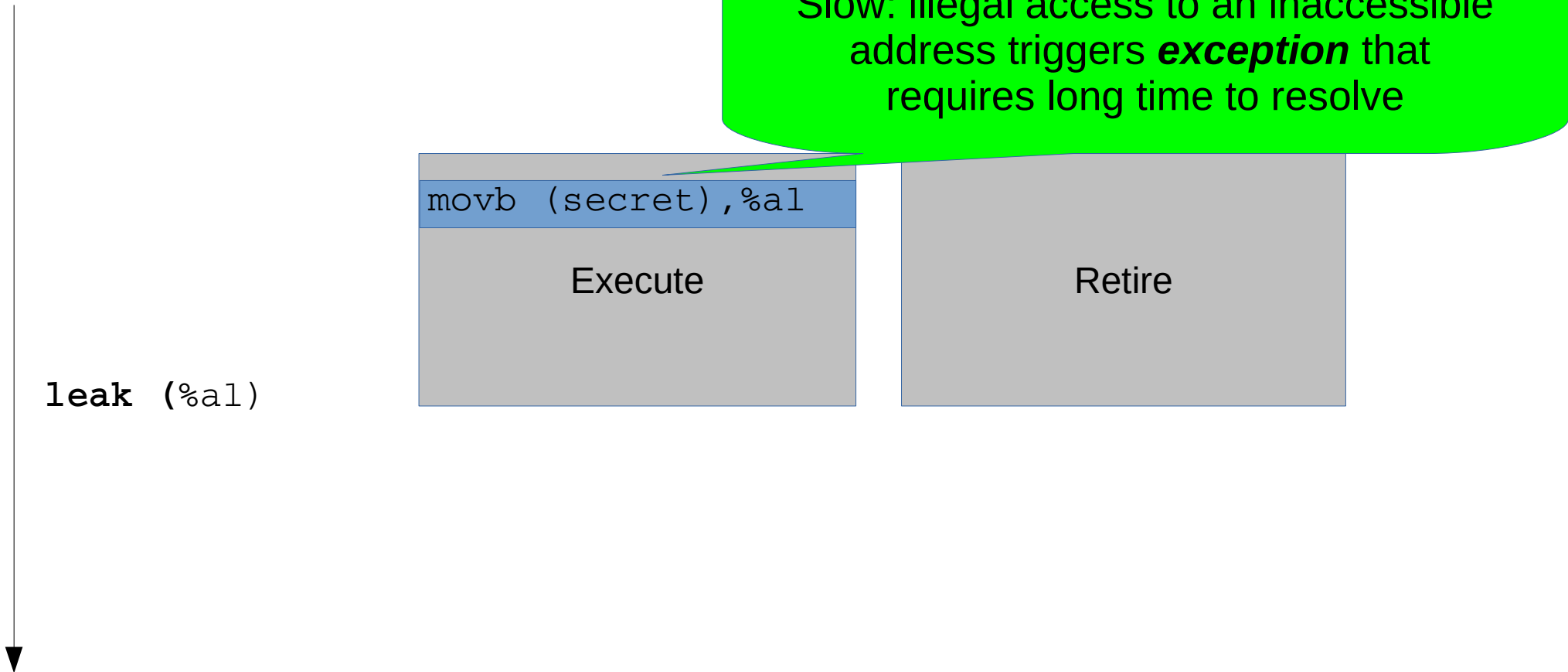
4

```
leak(%al)
```

Access generates exception

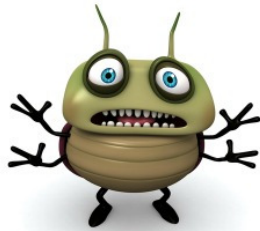
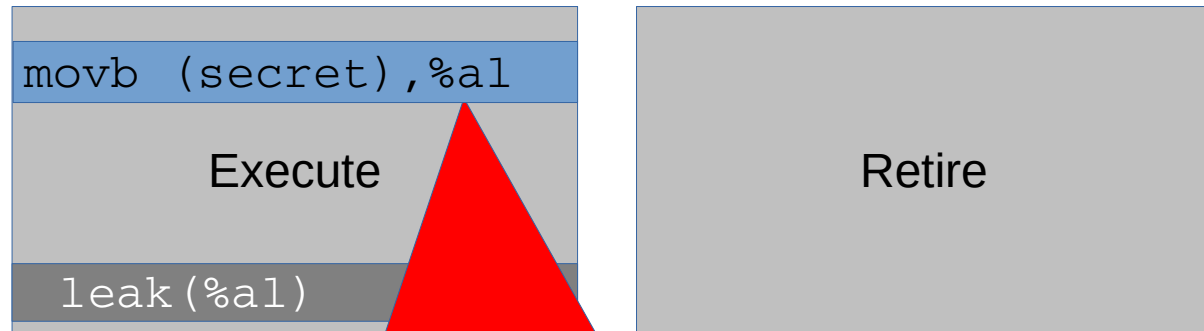
# Complete example (Rogue cache read – aka Meltdown)

Instruction stream



# Complete example (Rogue cache read – aka Meltdown)

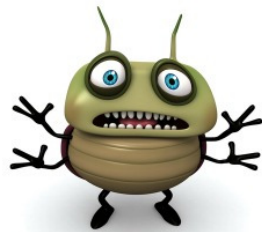
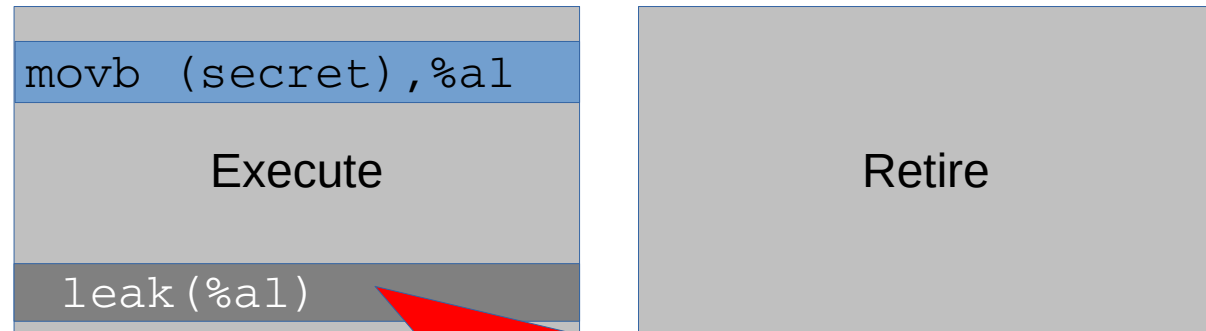
Instruction stream



(secret) is *insecurely speculated*:  
read from cache or DRAM ignoring  
page protection

# Complete example (Rogue cache read – aka Meltdown)

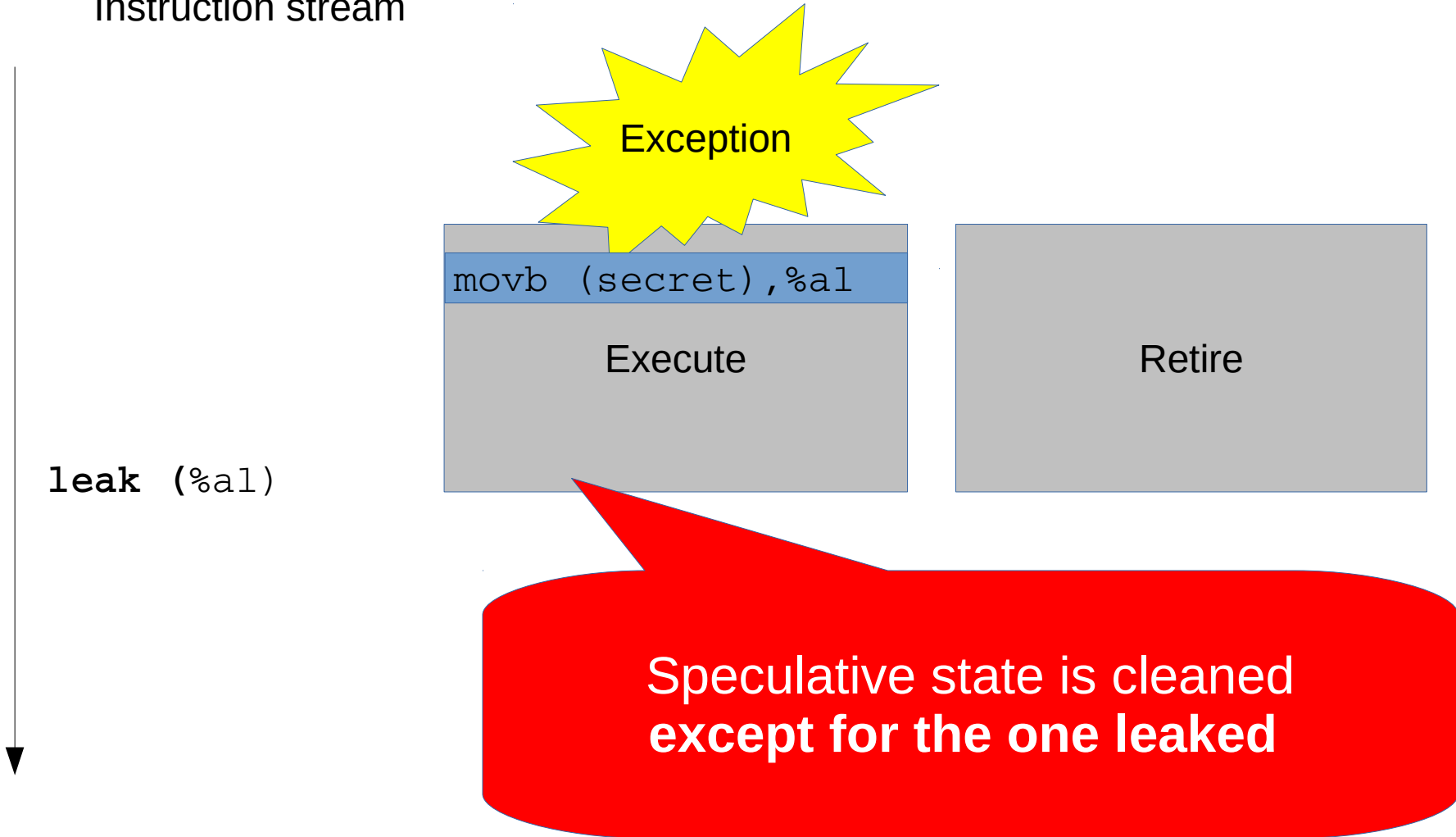
Instruction stream



Need to be fast to finish before  
the exception is resolved

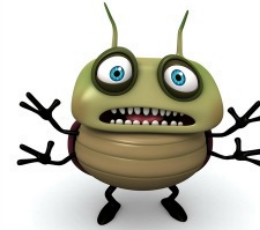
# Complete example (Rogue cache read – aka Meltdown)

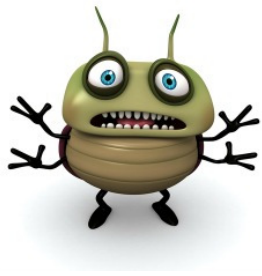
Instruction stream



# Recipe: Speculative read attacks

- **P**rovoke insecure speculation
- **W**in the race
- **N**otify the attacker

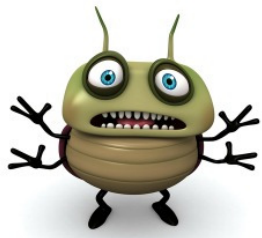




# Question 1: where does insecure speculation occur?

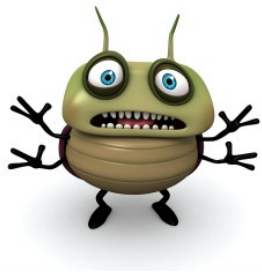
- Meltdown: **exception** due to access to a page with Supervisor **bit**





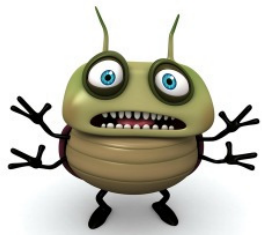
# Question 1: where does insecure speculation occur?

- Meltdown: **exception** due to access to a page with Supervisor **bit**
- Spectre V1: mis-speculated branch



# Question 1: where does insecure speculation occur?

- Meltdown: **exception** due to access to a page with Supervisor **bit**
- Spectre V1: mis-speculated branch
- Foreshadow/L1TF: **exception** due to access to a **non-present** page, or via an **incorrect mapping**



# Question 1: where does insecure speculation occur?

- Meltdown: **exception** due to access to a page with Supervisor **bit**
- Spectre V1: mis-speculated branch
- Foreshadow/L1TF: **exception** due to access to a **non-present** page, or via an **incorrect mapping**

The data is speculatively fetched from cache/memory *violating protection guarantees (OS/program)*

# Question 2: How to avoid misspeculation rollback?

- Not all  $\mu$ arch state can be rolled back
- $\mu$ arch state becomes architecturally visible!
  - Caches
  - Branch predictors
  - Performance counters
  - Contention on shared resources
- Simplest: cache covert channel (Meltdown/Spectre)

# Flush-Reload covert channel

- Flush the cache before the attack

- Sender/receiver: declare

```
char leak_array[4K*256]
```

- Sender:

```
void leak_byte(char secret) {  
    leak_array[4K*secret]=1;  
}
```

- Receiver: probe the array to identify cached values

```
- argmin(access_time(leak_array[4K*i]))
```

# Question 3: How to win the leak-to-rollback race condition

- Access to `leak_array` must be fast (in TLB)
- Access to secrets must be fast (in cache)
- Try many times
  - suppress the exception bailout
- Unsuccessful attempts are zero-biased

# Question 3: How to win the leak-to-rollback race condition

- Access to `leak_array` must be fast (in TLB)
- Access to secrets must be fast (in cache)
- Try many times
  - suppress the exception bailout
- Unsuccessful attempts are zero-biased

**Plus some secret sauce that nobody really understands why it works**

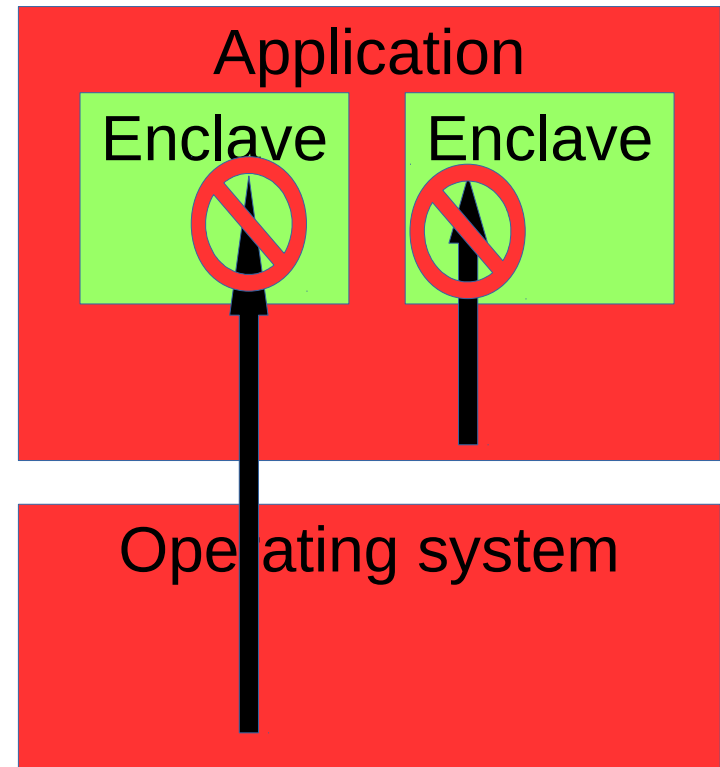
# Agenda

- Background on SGX
- Foreshadow
- Collateral damage on SGX
- Foreshadow-NG /L1TF
- Discussion

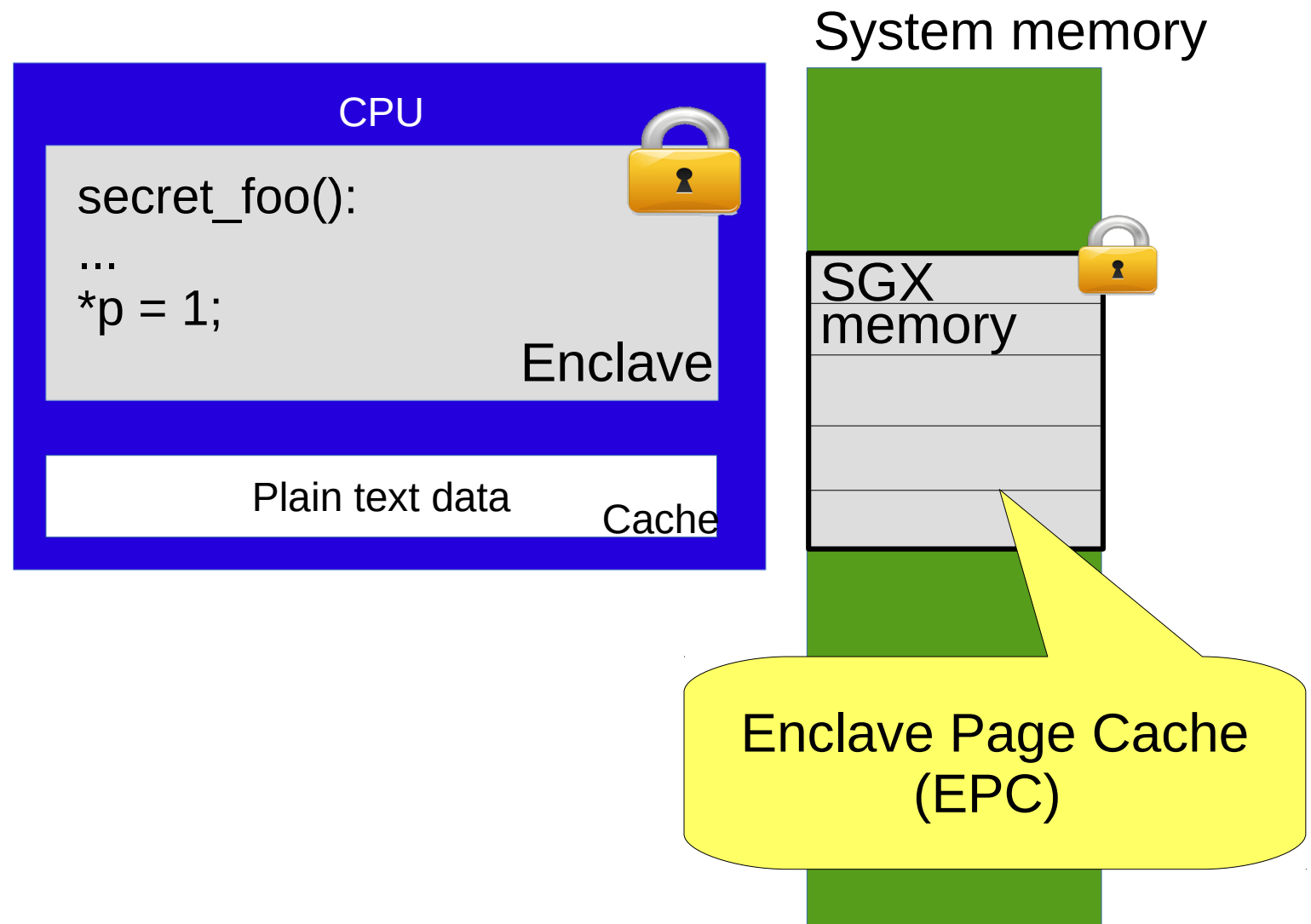


# Background: SGX

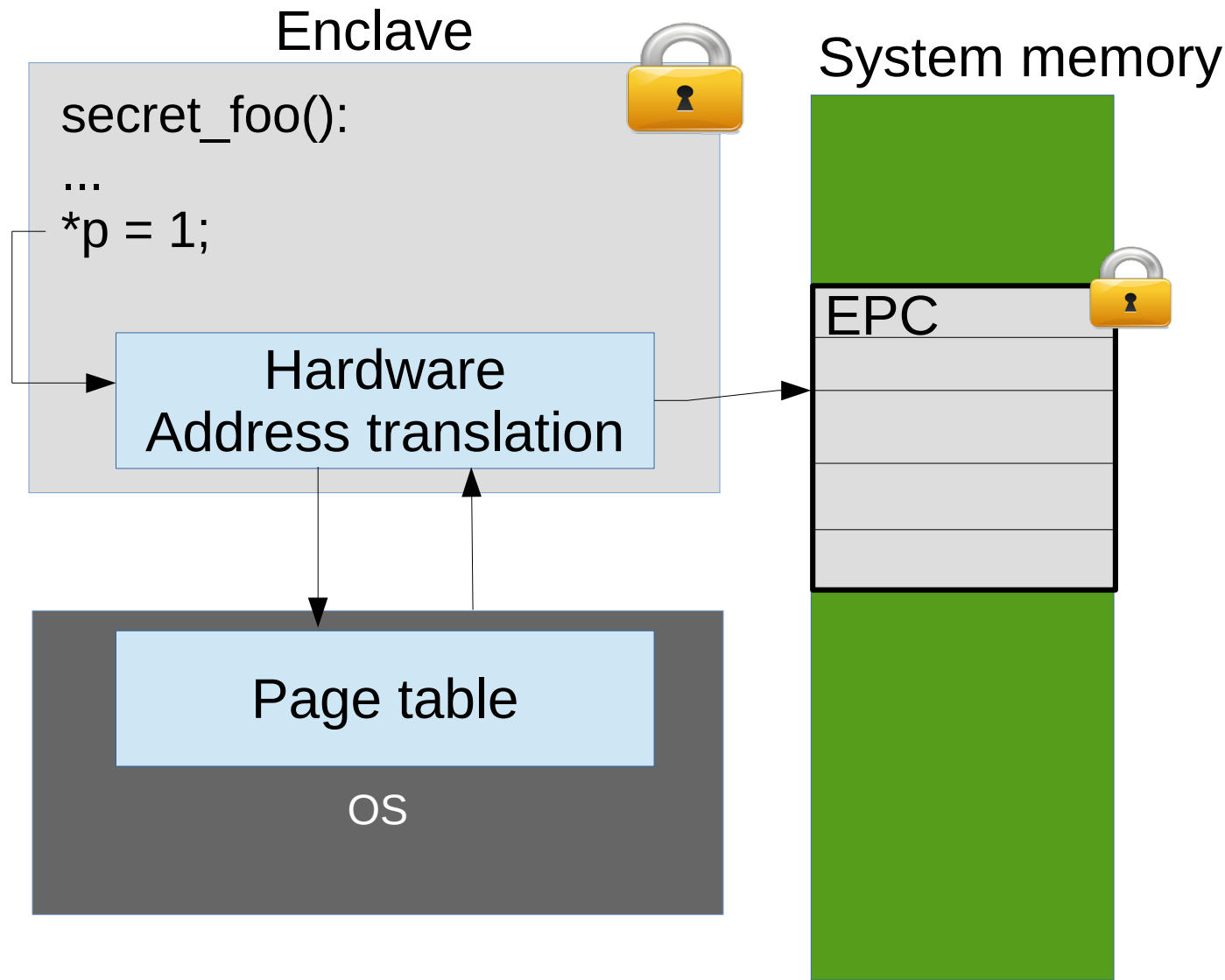
- Enclave: reversed sandbox
- Private code & data
  - Confidentiality
  - Integrity
  - Freshness
- Defends against privileged SW!
- HW acceleration
- Scales with CPU scaling



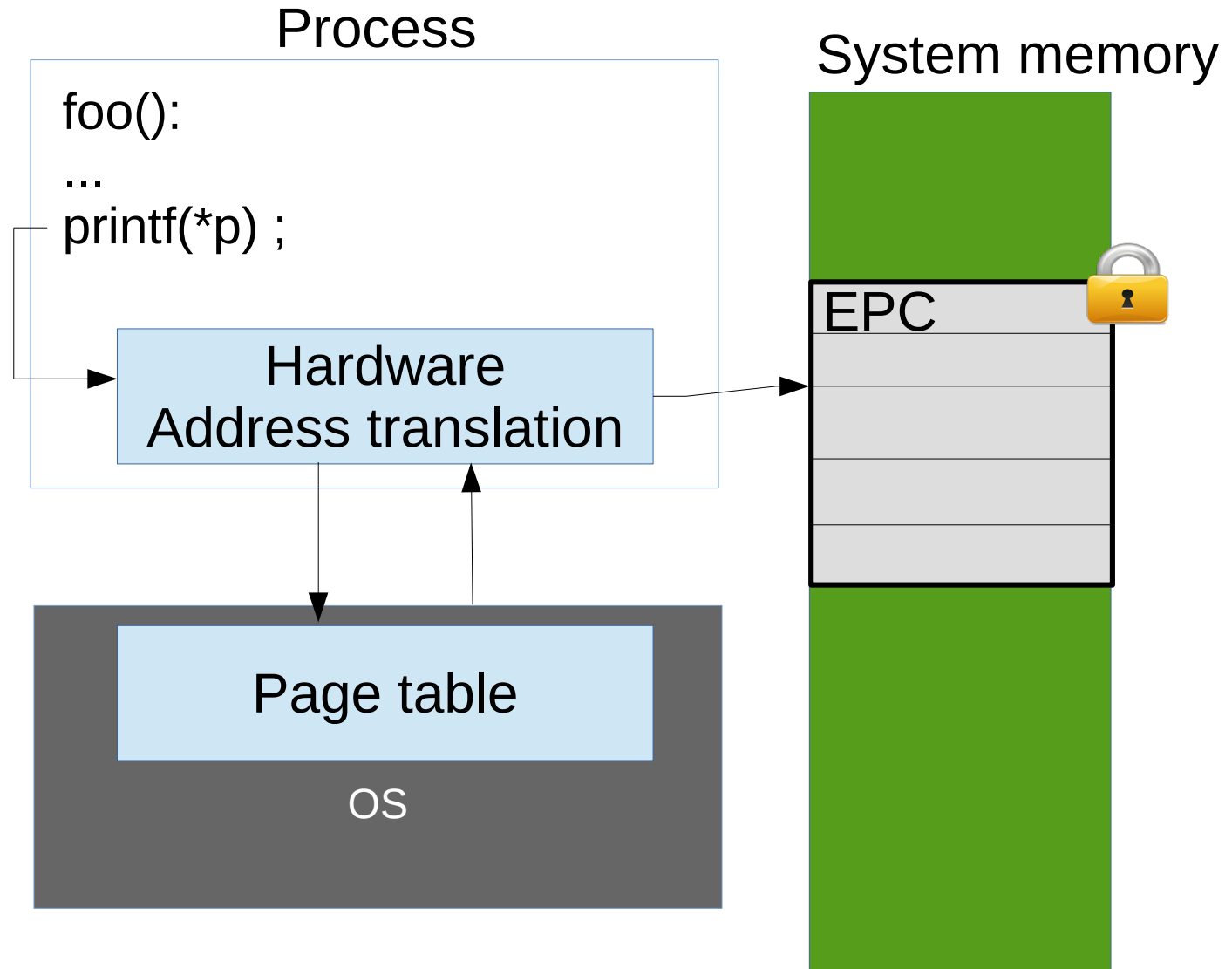
# Background: SGX memory DRAM encrypted, cache in plain text



# Background: Address translation in enclaves

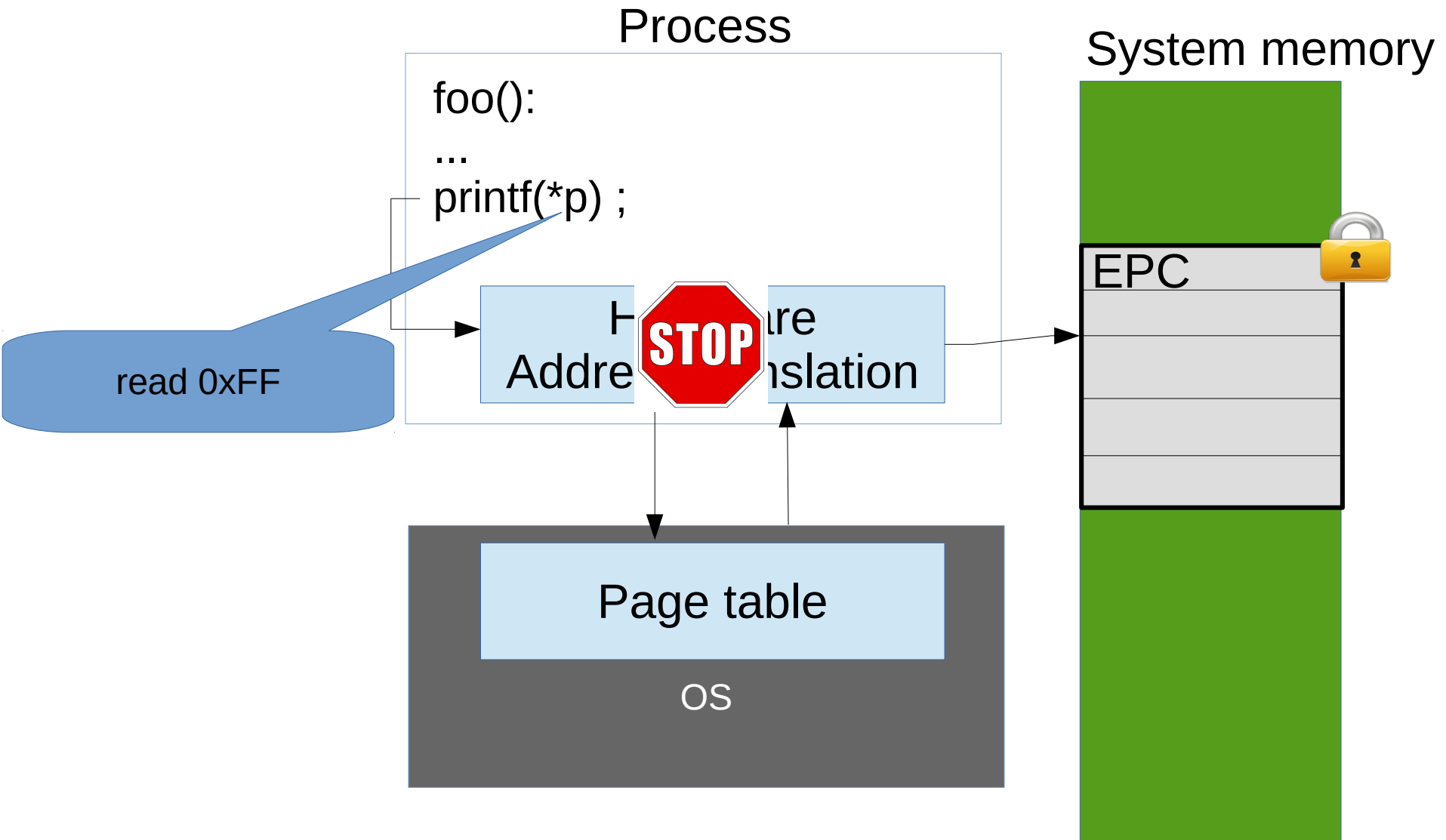


# Background: SGX abort page semantics



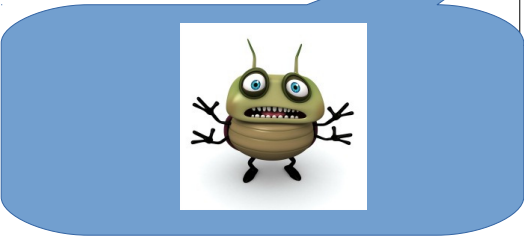
# Background:

## *SGX abort page semantics*

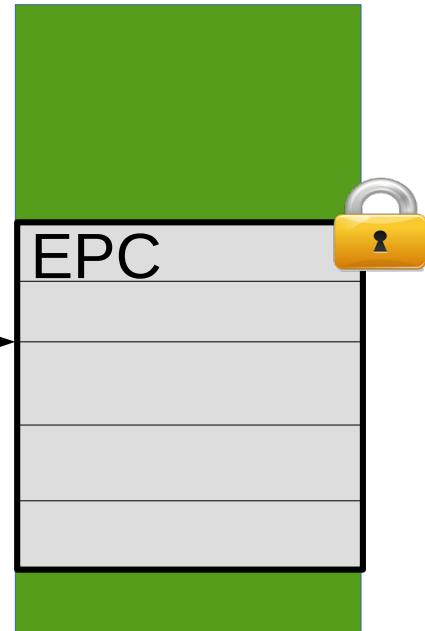


## Process

```
foo():  
...  
printf(*p);
```



## System memory



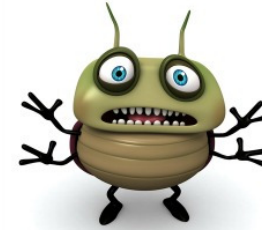
**Foreshadow uses speculative execution to leak secrets from SGX secure memory (EPC)**

# Agenda

- Foreshadow
- Collateral damage on SGX
- Foreshadow-NG /L1TF
- Discussion

# Reminder: Speculative read attacks

- **P**rovoke insecure speculation
- **W**in the race
- **N**otify the attacker





# Challenges of SGX attacks

- Provoke

- Abort page behavior suppresses exception: no speculation

SGX is resilient to strawman Meltdown attack

- Provoke/Win – Secrets must be in L1 cache
- Notify – Same as Meltdown

# Challenges of SGX attacks

- **Provoke**

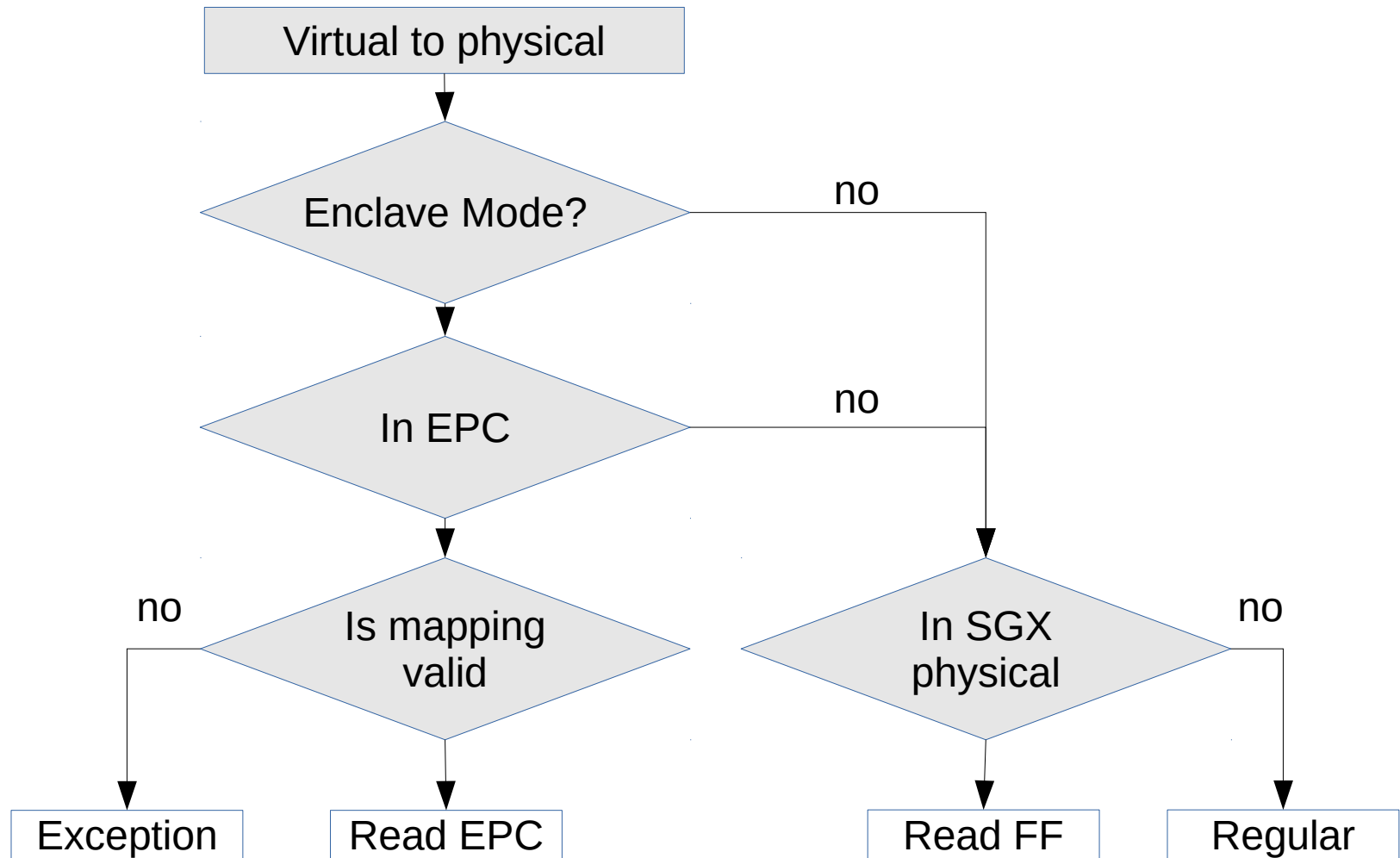
- Abort page behavior suppresses exception: no speculation

SGX is resilient to strawman Meltdown attack

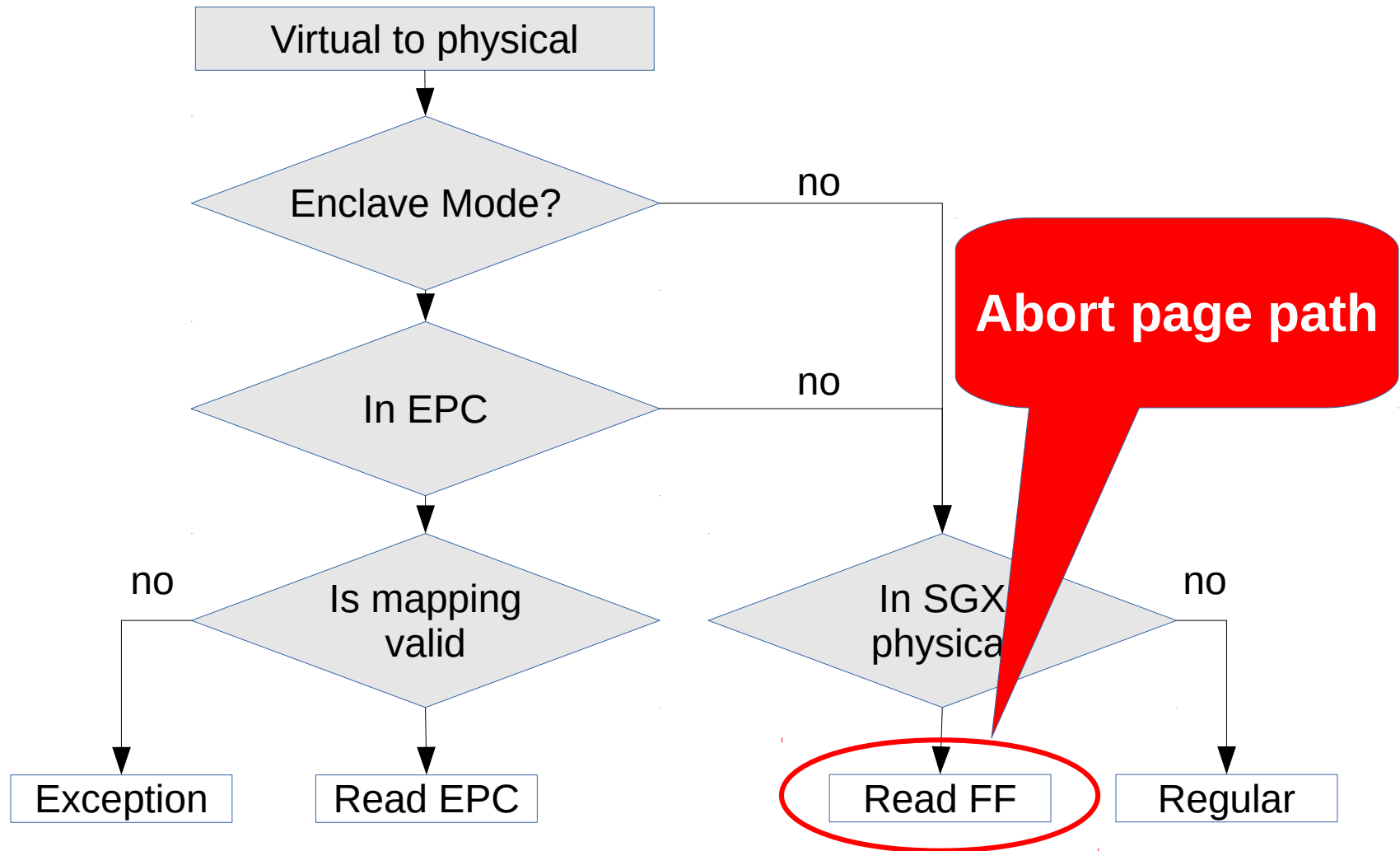
- **Provoke/Win** – Secrets must be in L1 cache

- **Notify** – Same as Meltdown

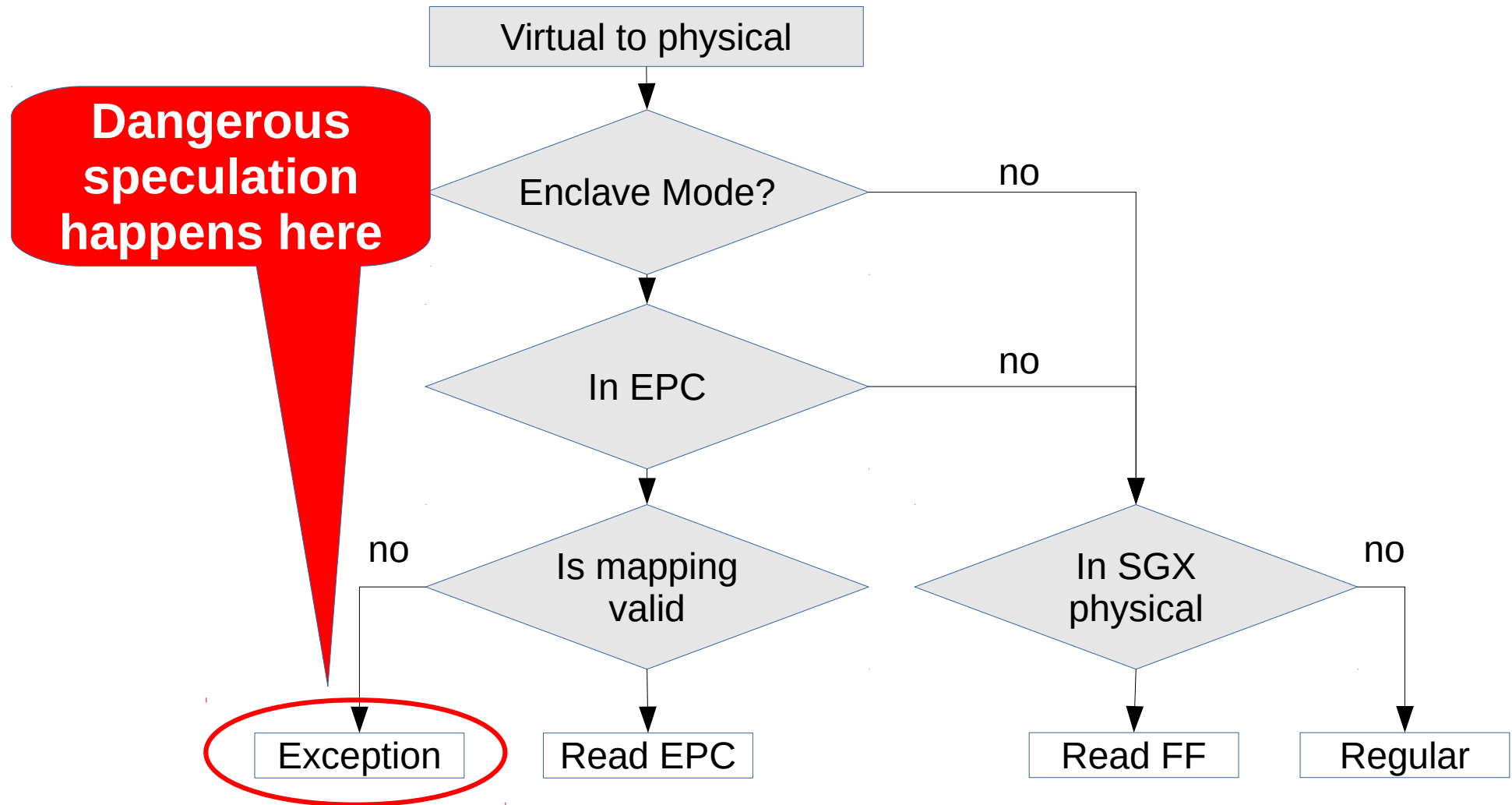
# Understanding memory translation with SGX



# Understanding memory translation with SGX



# Understanding memory translation with SGX



# Provoke 1

## Overriding abort page semantics

- Idea 1: access to a “not-present” EPC page
  - ***user*** calls `mprotect(epc_mem, PROT_NONE)`

# Provoke 1

## Overriding abort page semantics

- Idea 1: access to a “not-present” EPC page
  - *user* calls `mprotect(epc_mem, PROT_NONE)`
- Access to the `epc_mem` triggers exception
- Speculative path *reads* `epc_mem` from L1 despite **SGX** protection

# Provoke 1

## Overriding abort page semantics

- Idea 1: access to a “not-present” EPC page
  - **user** calls `mprotect(epc_mem, PROT_NONE)`
- Access to the `epc_mem` triggers exception
- Speculative path **reads** `epc_mem` from L1 despite **SGX** protection



- **Why does it work?**  
Speculative path ignores SGX memory checks



# Provoke 1

## Overriding abort page semantics

- Idea 1: access to `epc_mem` from user space!
  - **user** calls `mprotect(epc_mem, PROT_NONE)`
- Access to the `epc_mem` triggers exception
- Speculative path **reads** `epc_mem` from L1 despite **SGX** protection

This attack works from user space!



- **Why does it work?**  
Speculative path ignores SGX memory checks

# Provoke 2

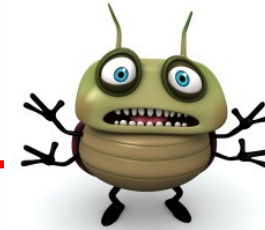
## Overriding abort page semantics

- Idea 2: access maliciously mapped page
  - kernel maps an EPC page into another enclave

# Provoke 2

## Overriding abort page semantics

- Idea 2: access maliciously mapped page
  - kernel maps an EPC page into another enclave



- Why does it work?
  - Speculative path ignores inter-enclave protection checks

# Attack works with secrets in L1!

## How to ensure they are in L1?

1. Single-stepping an enclave with SGX-Step
2. Controlled side channel attack
- 3. *Dumping enclave's memory without enclave execution via enclave swapping***



# Leak secret

- Same as in Meltdown:
  - flush-and-reload cache covert channel
- Some tweaking to win the race

# Summary so far

- SGX is vulnerable to speculative execution attacks
- Enclave's data in L1 cache can be accessed via speculative access
- L1 cache can be ***populated*** via enclave paging mechanism **without executing the enclave**
- Result: dump ***all*** enclave memory

# Collateral damage: attacking SGX attestation

# Remote attestation

- Essential for SGX ecosystem
- Enables a party trusting Intel to trust an enclave executed on a remote machine



# Remote attestation

- Example: **Netflix** video player runs on **your** computer, receives secrets from Netflix.
- Remote attestation proves to Netflix that
  - The player is running on genuine Intel's hardware
  - The player's binary is a genuine one

Sponsored add:

An excellent primer on SGX 2.0 attestation: first talk at

<http://cyber.technion.ac.il/2018-summer-school-on-cyber-computer-security>

# SGX Architectural Enclaves

- Implement remote attestation in software
- **Rely on SGX security guarantees**
  - **keep Intel-provisioned Secret in the Architectural Enclave**
- Trusted by Intel

# Observations

- Knowing **Intel Secret** allows signing faked enclaves
- **Intel Secret** is *designed for unlinkability*
  - Intel cannot tell apart enclaves signed with the same key
- Corollary: with the Intel Secret in attacker's hands, enclave *users* (Netflix) cannot tell apart genuine and faked enclaves!

# How to retrieve Intel Secret?

- The Secret is stored on a disk encrypted with *sealing* key
- Sealing key is found in enclave's memory of the Intel Architectural Enclave

# How to retrieve Intel Secret?

- The Secret is stored on a disk encrypted with *sealing* key
- Sealing key is found in enclave's memory of the Intel Architectural Enclave

We attack the Quoting Enclave:  
A combination of

1. Controlled side channel
2. Foreshadow

# AaaS (Attestation as a Service)

- [@ForeshadowAaaS](#) Will attest to anything tweeted at it
- Reduced cost of hackership – no need to buy an SGX machine
- Hacker's privacy guaranteed by EPID protocol
- Attestation server returns `Group_Out_Of_Date`
- **SGX Keys are still not revoked (despite weeks of advances notice)**
- Blocked by Twitter



**Foreshadow Attack** @ForeshadowAttac · 4m

@ForeshadowAaaS Is your enclave cheating on you? Please sign this for me. That sam I am that sam I am I do not like that sam I am



**Foreshadow AaaS** @ForeshadowAaaS · 4m

Here is your attestation that "Is your enclave cheating on you? Please sign this for me. That sam I am that sam I am I do not I" is a genuine SGX enclave

[github.com/TeeAaaS/Foresh...](https://github.com/TeeAaaS/ForeshadowAaaS)

```
{'QuoteBaseName': bytearray(b'\xe8\xce#\xd8\x18\x17\xea\x81x8\x90h{+\xed\xb7'
                             b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
                             b'\x00\x00\x00\x00'),
 'QuoteEnclaveSVN': 0,
 'QuoteGroupID': 2777,
 'QuotePCESVN': 5,
 'QuoteReportBody': {'Attributes': bytearray(b'\x05\x00\x00\x00\x00\x00\x00\x00'
                                             b'\x07\x00\x00\x00\x00\x00\x00\x00'),
                    'CPU-SVN': 0,
                    'MRENCLAVE': bytearray(b'Is your enclave cheating on you?'),
                    'MRSIGNER': bytearray(b'Foreshadow Attack.....'),
                    'Report-data': bytearray(b'Please sign this for me. Th'
                                             b'at sam I am that sam I am I'
                                             b'do not I'),
                    'Vendor-product-ID': 0,
                    'Vendor-product-SVN': 0},
 'QuoteSignType': 1,
 'QuoteSignature': b'qzTrSP8Fv+IAvt8/HHZ/iAG0glW6Cd4wFLkNajDrkXsqkghKiNVwRRf7'
                  b'8wD3j/tbPriyTQvdfeuxUtdF1+tDx0mrvV2dnTsGytt80mDpzwhx7Nkc'
                  b'RpazL0LeCu1V6SR8Tj/rHUh0rNGOXQv2q509IZvJWsG2alwcnjFuSajc'
                  b'roFhu+Y923MXHUCcVi9tnppqZnrx1EEkgSiHvyMam1qpmQ1kiAqS5b1v'
                  b'+uR/Gs2y7AD7pwJUx4byYhTF5FoQHTxP22ycRDj4qGaDDQe8JUAXU85'
                  b'4xEBWTBd0wU1wORPLZ09Efoen1pxLyQDgYdtTyzkwl1K3mAnkMu3t+JC9'
                  b'4dKDQyucXIAM/nmdSJr6aMIyFXU26e6GV7kRXXoxE+e5zPCpYGoZIMsYJ'
                  b'0A701P6GaAEAAKwKt1E2aeFagxXu30ujmZFd0GBa+ngxH9Fz1F4L8/k'
                  b'hU1UhpPxWun0oE5NX07K1VuHYtBR2azZG1IN+LaaZY339BR53zAVz0wv'
                  b'Rb19dJ050R18Is0YYkCaC5NRfnCnLNkAsIm1eZnKwaf5cN83fM5wI3of'
                  b'fnFq3Evg4T6vbMf0fn2kFlzHiGZVq077qyEz7u03bsZyHcFXLGBS0ms'
                  b'e9qPrN+PIKah64M4Mswod5YJyqSnFiY3r0k4X80XG/qiywI9FE8/Cpm'
                  b'ByI/1sEiXGg6wnt0g0XhMa4gw1+tbTc0s0E0eEgblbe01w3YengV2PDV'
                  b'LsbuK6Yx0k2r610w`hKywhymFnM191YUBX3/d1n05JmBPg3RMaddp+IN'
                  b'pxVowgR7JubXJzT4E`jWI7sM0bprdfVnbb6H3KL0UkCIcV4GMBBy43q0G'
                  b'kXyhubFyEokvC2Z4eHtvKauLIPj82EYU7qjIDHg99VCEq92LZB06ESu'
```

## ForeshadowAaaS

Details Settings Keys and Access Tokens Permissions



Restricted from performing write actions

The service will provide the users with the new app of ForeshadowAaaS.



FORESHADOW

@ForeshadowAaaS

# Summary so far

- SGX is vulnerable to speculative execution attacks
- Allows dumping enclave's memory
- Attack enables leaking sealing key and Secret from infrastructural enclaves
- **Breaks the SGX remote attestation** without an easy way to revoke (anonymous) Secret

# Foreshadow-NG: L1TF

- Foreshadow reported on Jan 3<sup>rd</sup> by KU Leuven, Jan 23<sup>rd</sup> by Technion/Michigan/Adelaide
- Intel's follow up (Aug 11, but known since March): there are three other flavors, same bug
  
- Process-to-process
- Process-to-SMM
- **VM guest to host**



# L1 Terminal Fault

- When an accessed page is marked ***not present (terminal fault)***, PA is used to access L1 cache, while ignoring..
  - SGX: EPC access checks
  - OS: Protection checks
  - VirtualMachine: GuestP-to-HostP translation
- Implication: guest controls which Host Physical addresses to access
- Major issue: forced months of disclosure embargo

# Foreshadow vs. Meltdown

- Spectre/Meltdown – same address space leaks
- Foreshadow – both intra and inter-address space leaks. Memory isolation non-existent

# Mitigation: Foreshadow

- SGX microcode updates
  - flush L1 on each enclave exit/eldu
    - => prevents non-concurrent attacks on L1
  - hyperthreading is part of the enclave trusted state
    - => prevents concurrent attacks on L1
  - increase security version (TCB update)

# Open questions

- Foreshadow: bug or design (methodology) flaw?
- Does *SGX inherit* the bug from X86?
- What do we actually know about the reasons?
  - Hint: not much
- SGX remote attestation relies on SGX – poor design choice?
- Disclosure process: who is in charge for the world piece?

# Summary: Foreshadow

- PWN SGX enclaves
- Breaks SGX confidentiality
- Steals seal-key – breaks the integrity of persistent storage
- Breaks the remote attestation guarantees which relies on the enclave
- Same bug causes VM, OS and SMM protection violation

Questions?

[mark@ee.technion.ac.il](mailto:mark@ee.technion.ac.il)