Foreshadow:
speculative attacks on SGX and beyond

Mark Silberstein

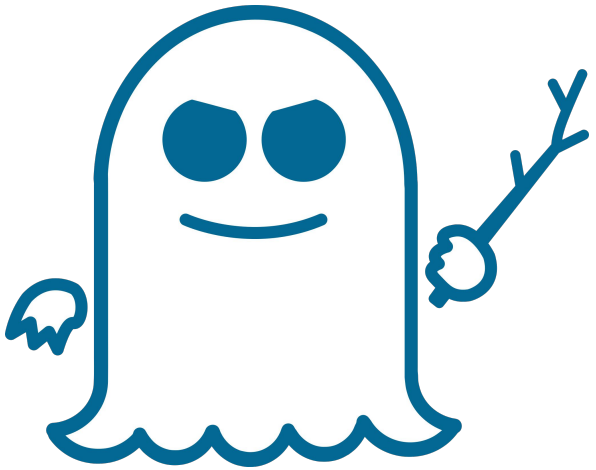
Joint work with
Jo Van Bulck, **Marina Minkin**, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx

Big picture in one slide

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

Big picture in one slide

- Where do CPUs loose 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

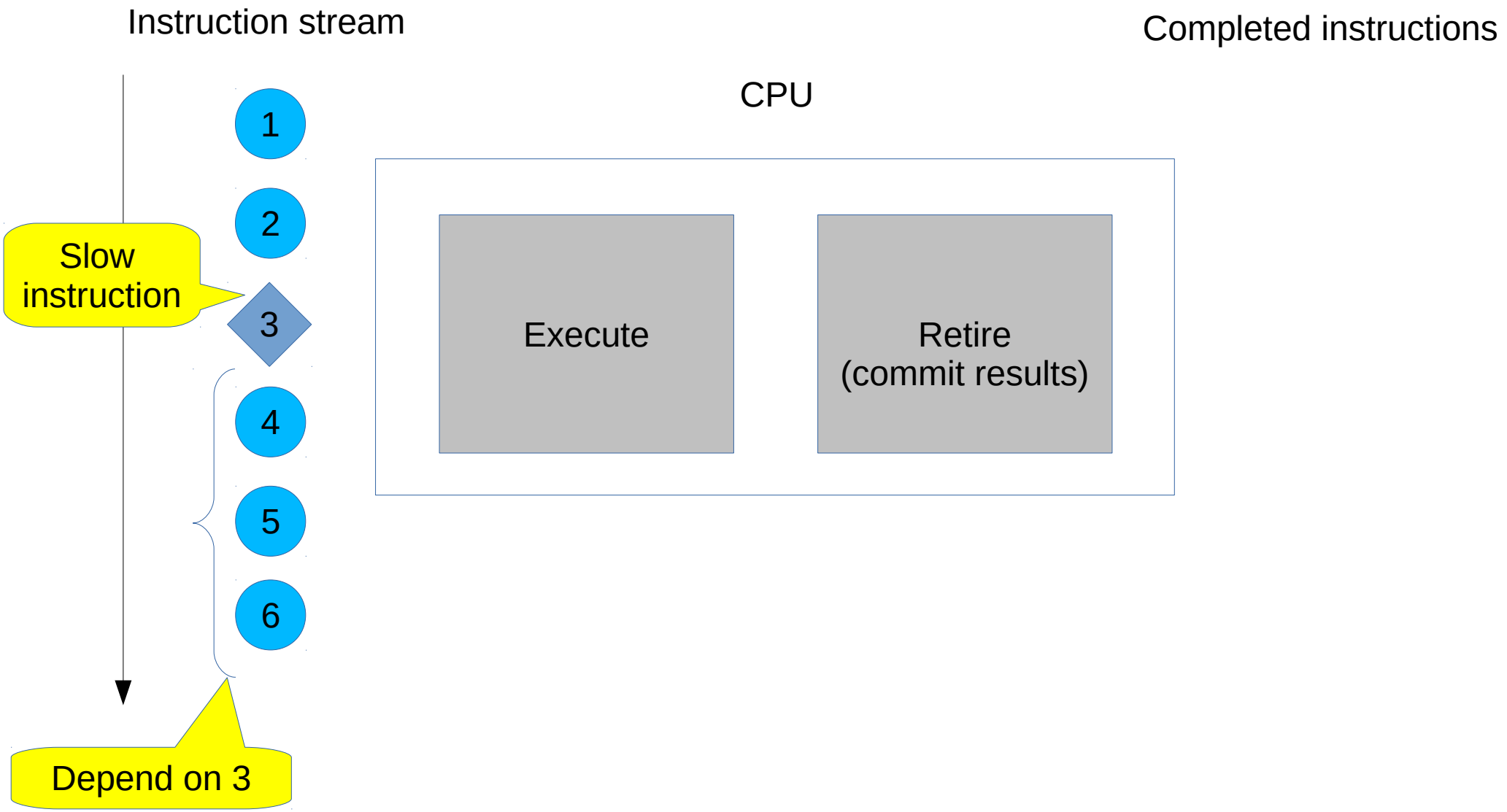

Speculative execution attacks exploit

- *Speculation* past illegal memory accesses
- *Inability* to fully roll back μarch state
- *Covert/side channels* to leak the state

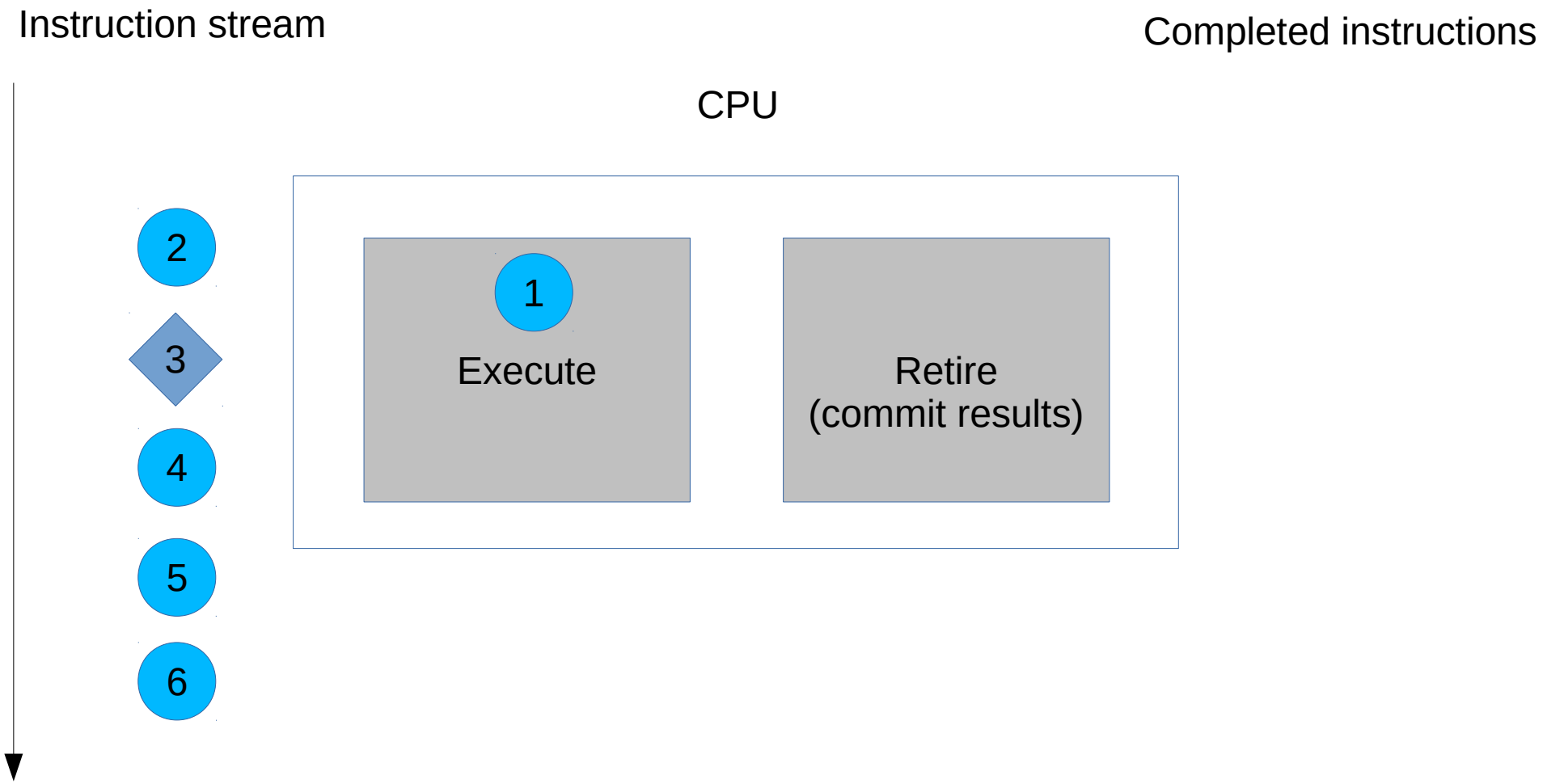
Today

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

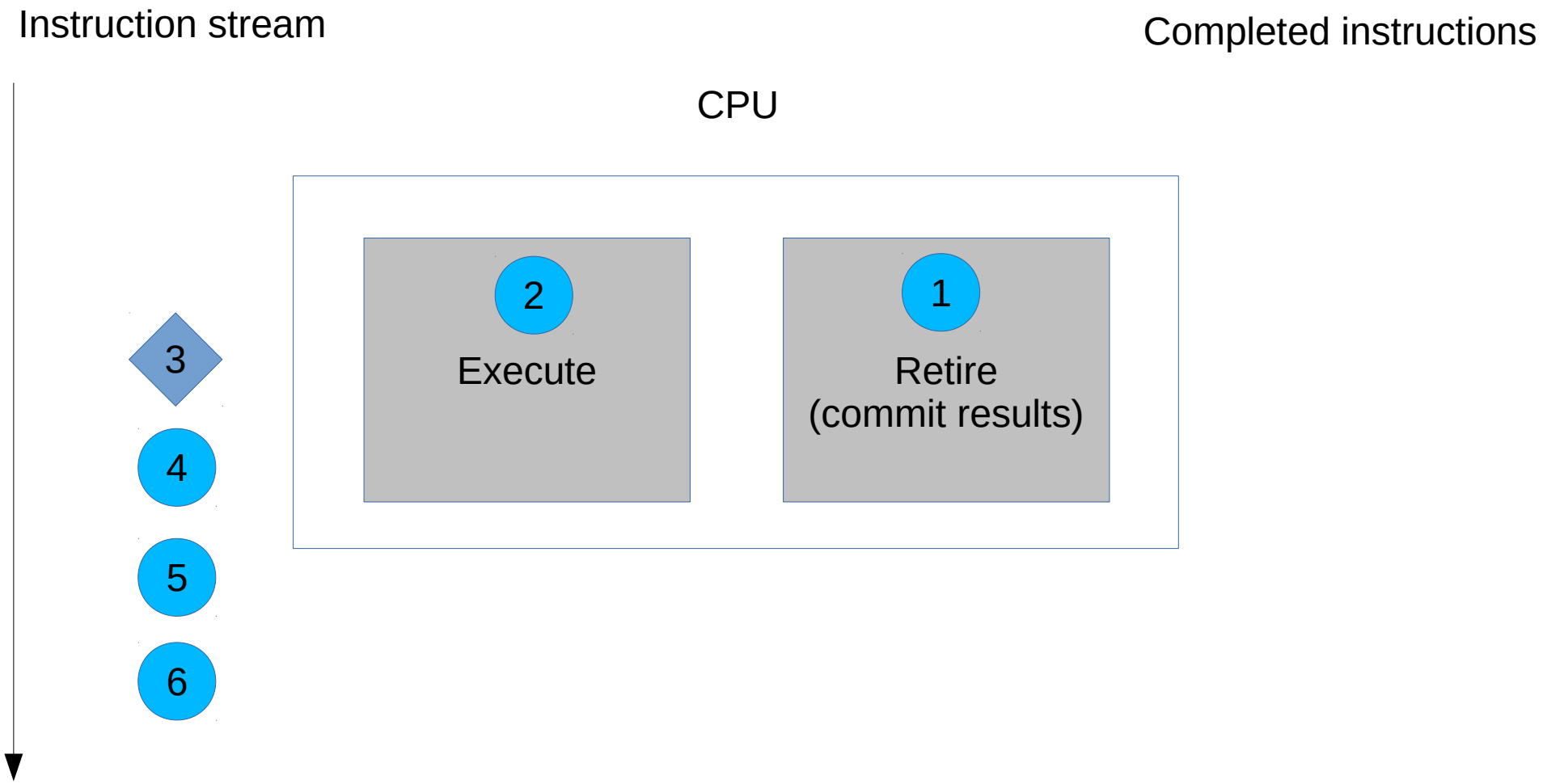
Speculative execution 101



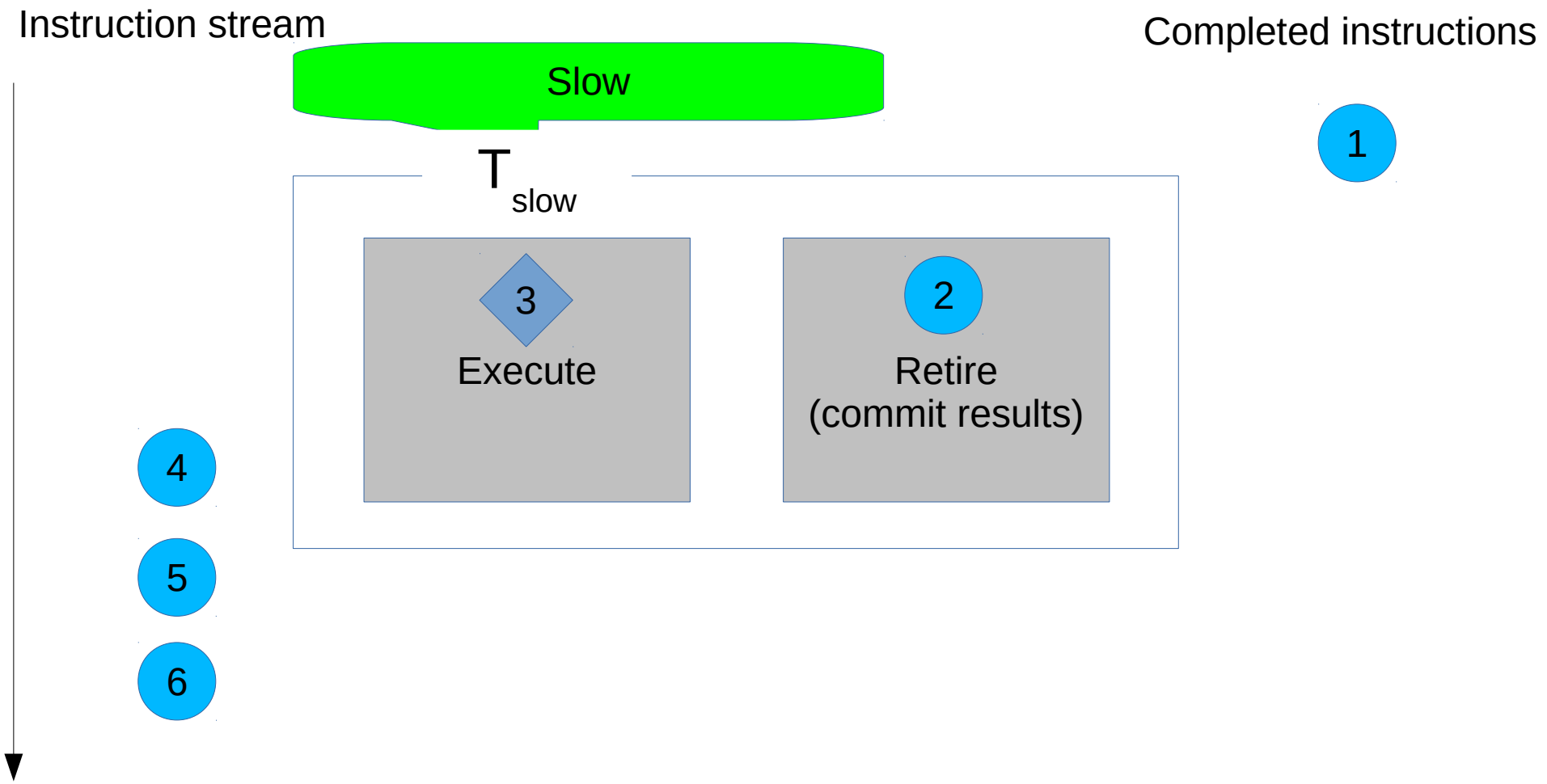
Speculative execution 101



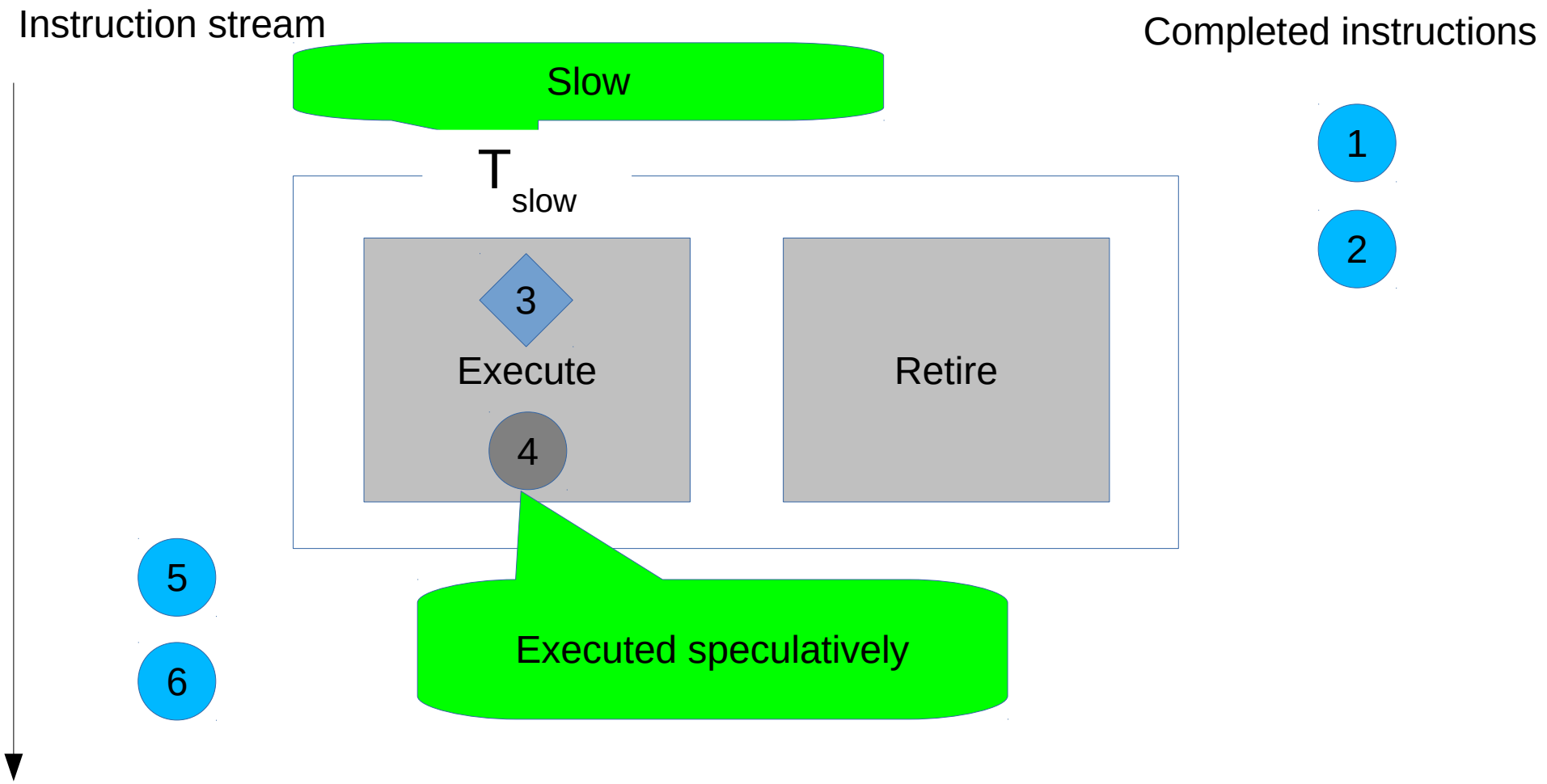
Speculative execution 101



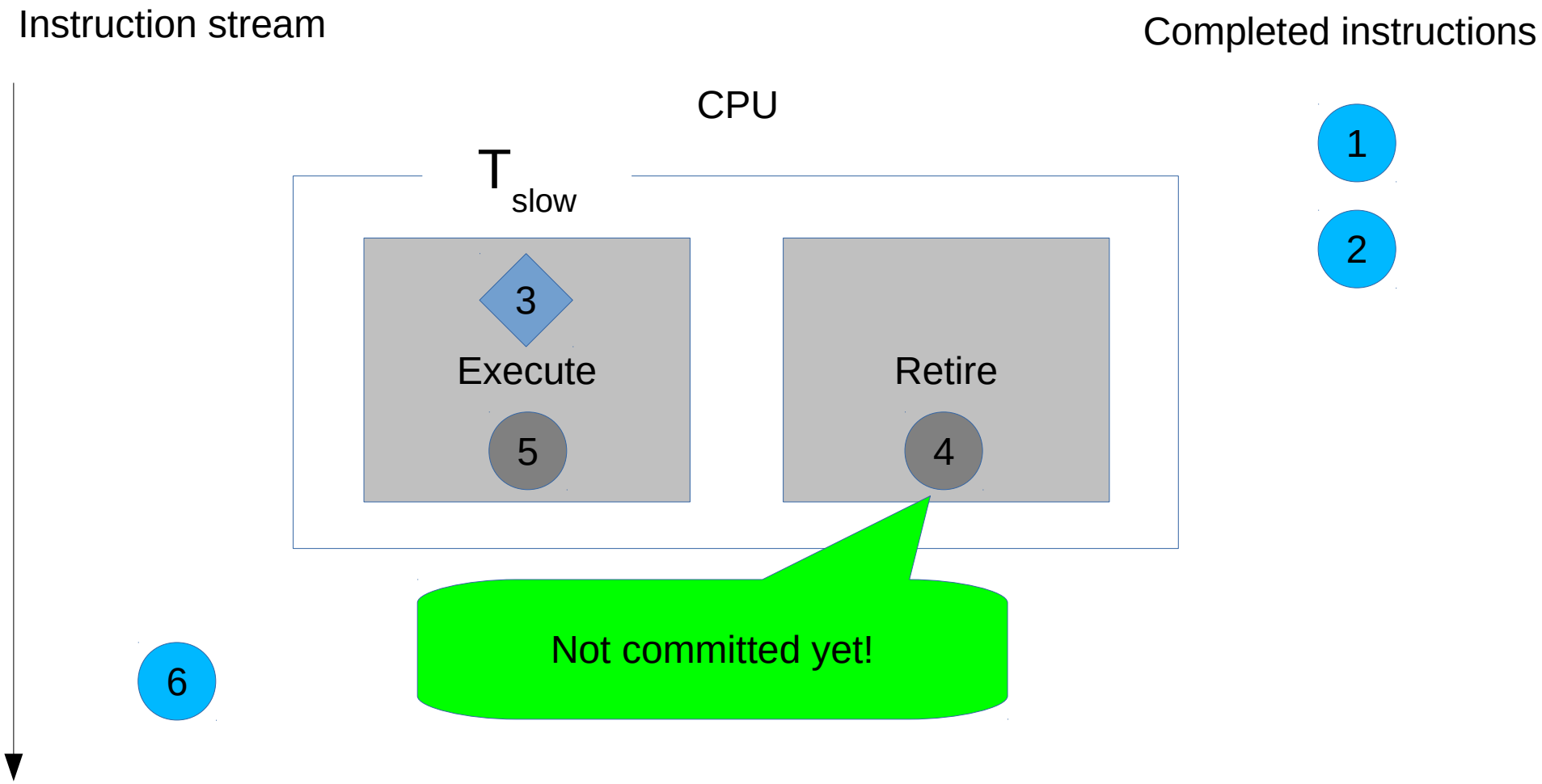
Speculative execution 101



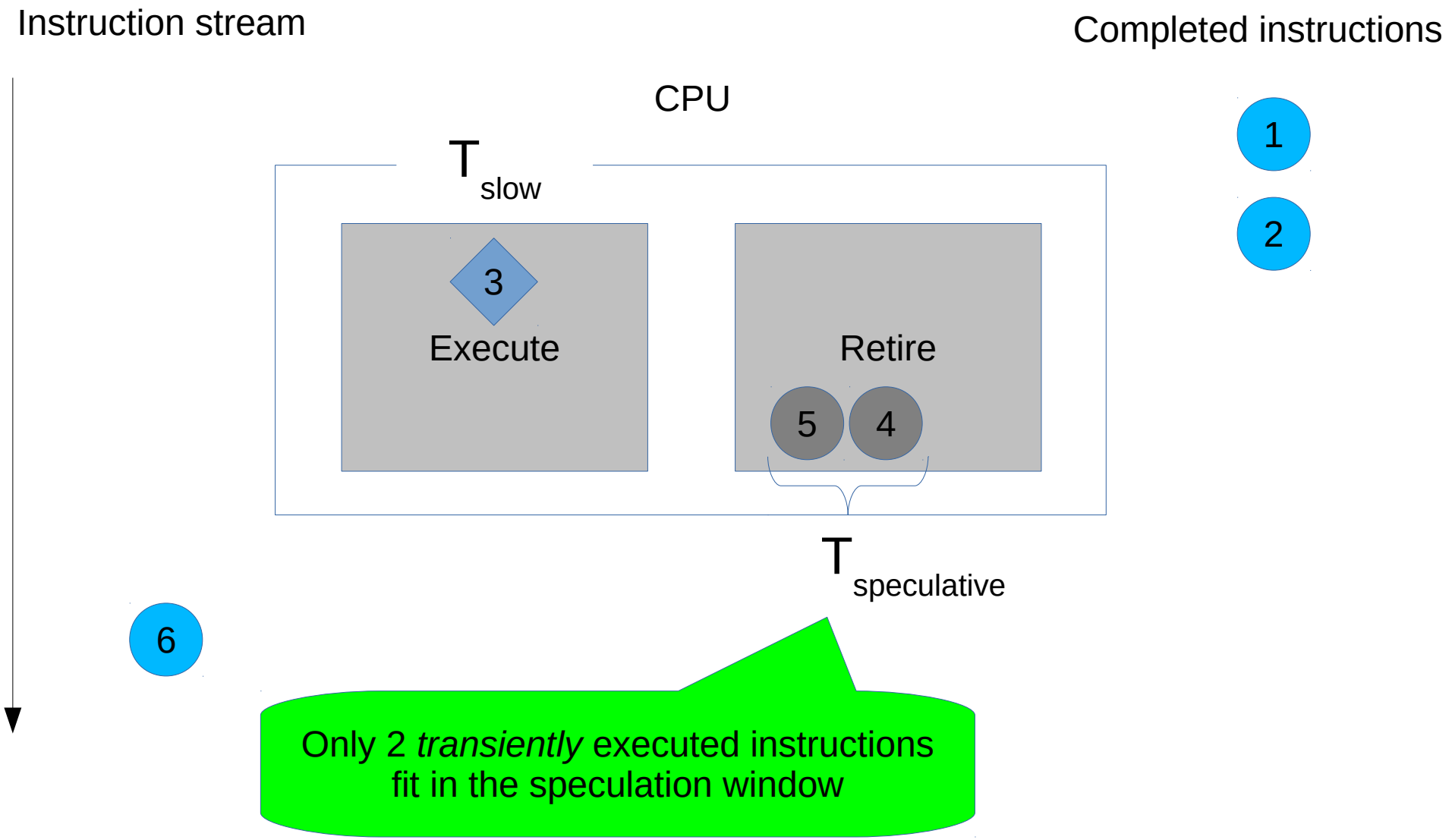
Speculative execution 101



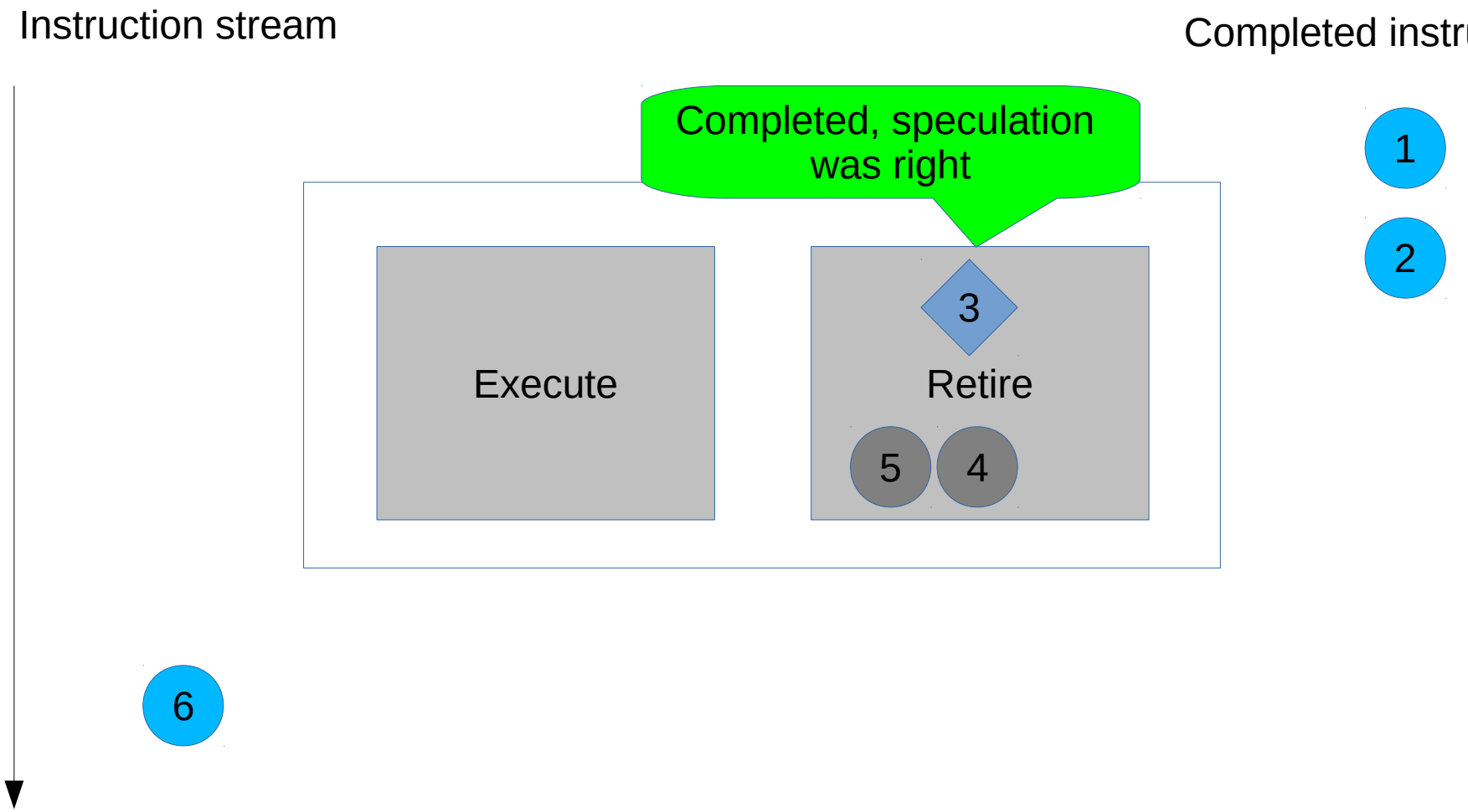
Speculative execution 101



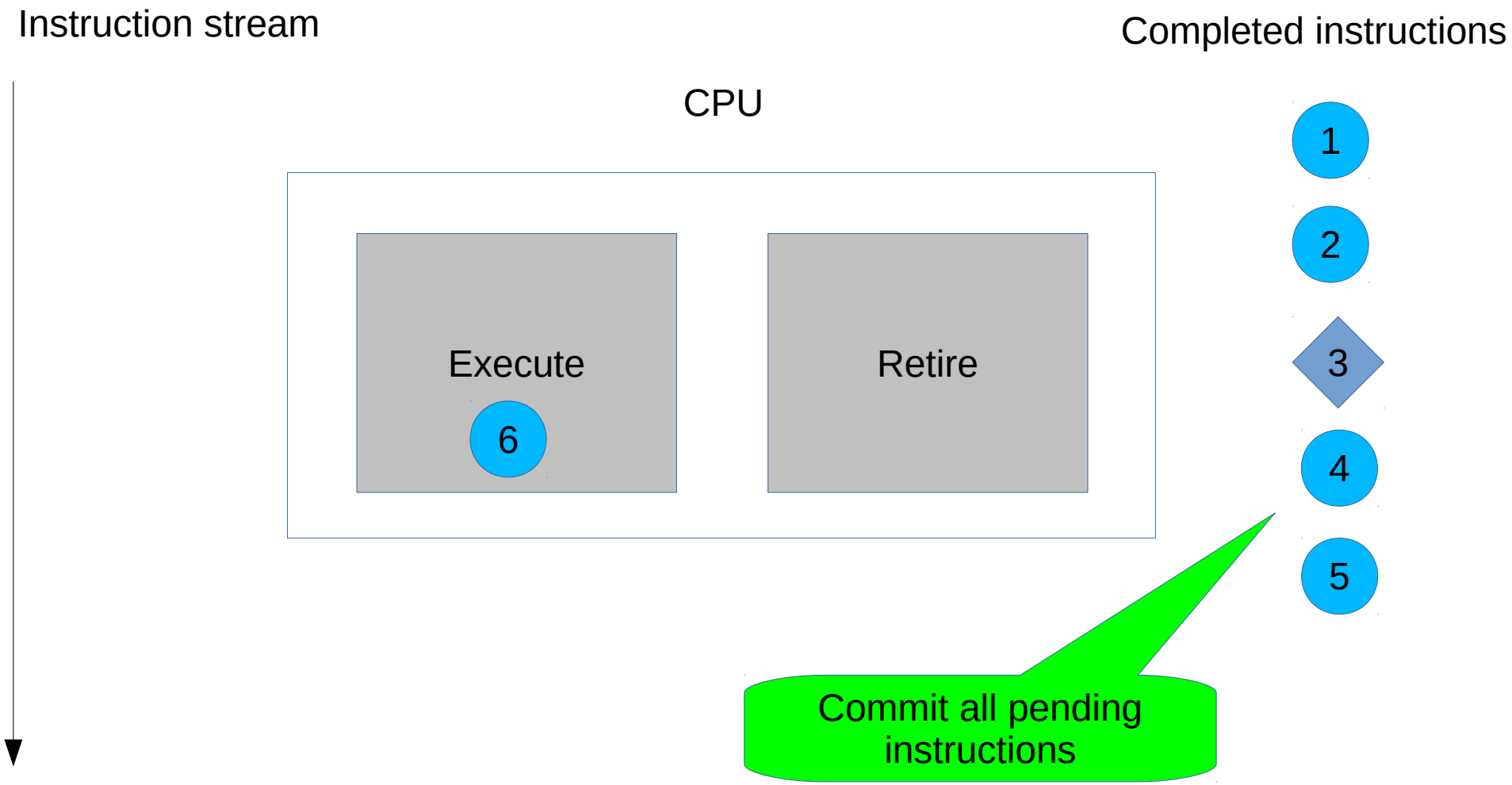
Speculative execution 101



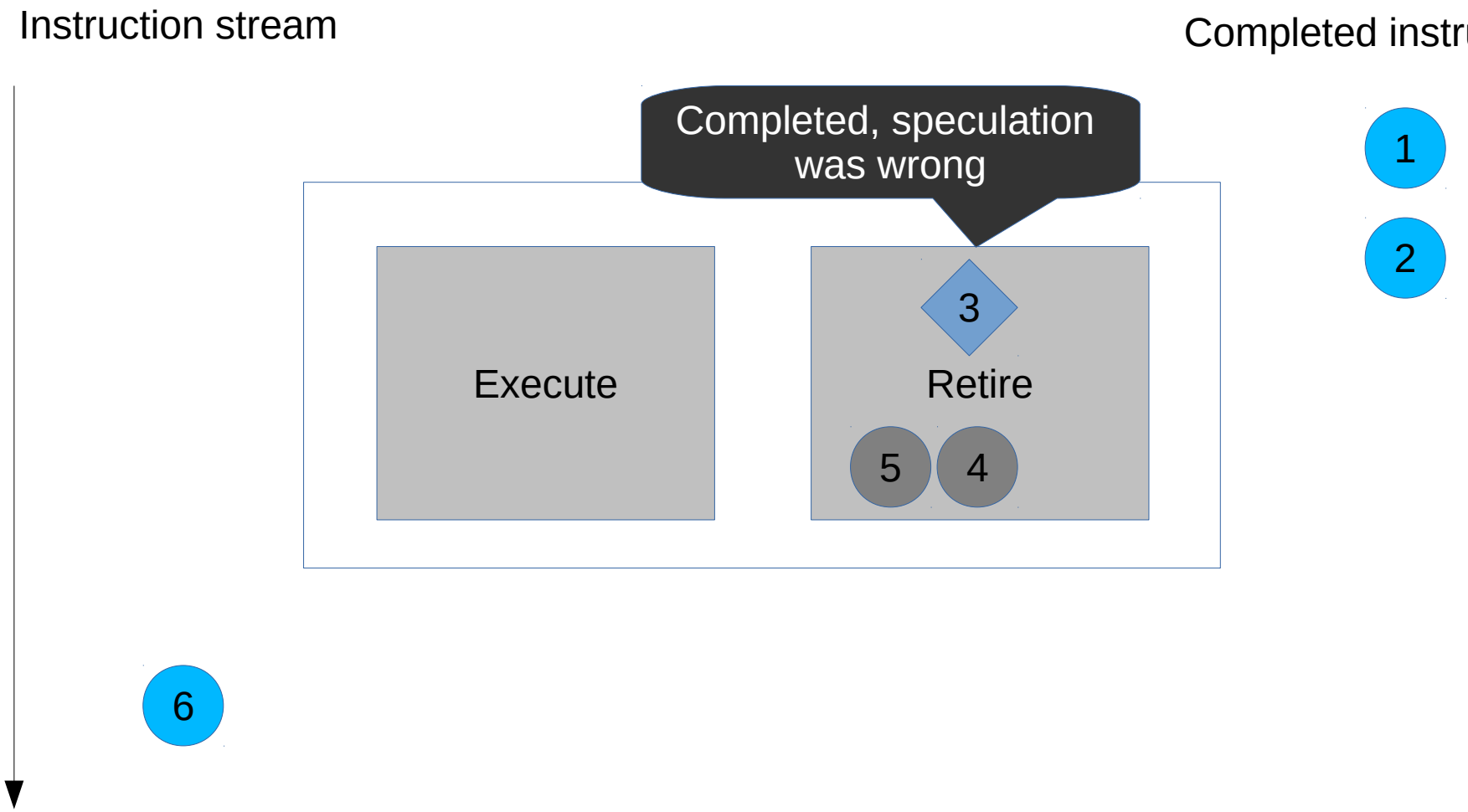
Speculative execution 101



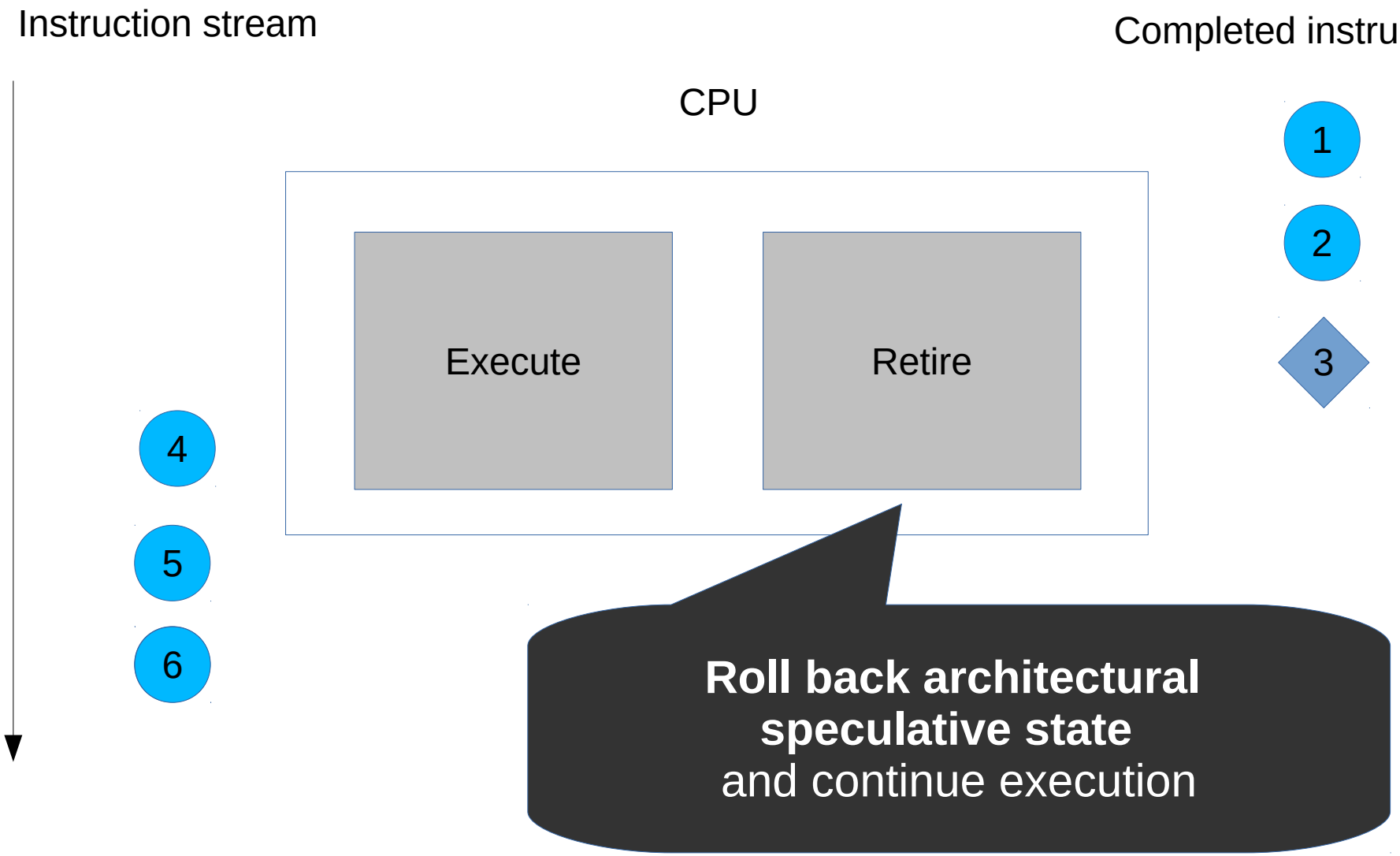
Speculative execution 101



Speculative execution 101



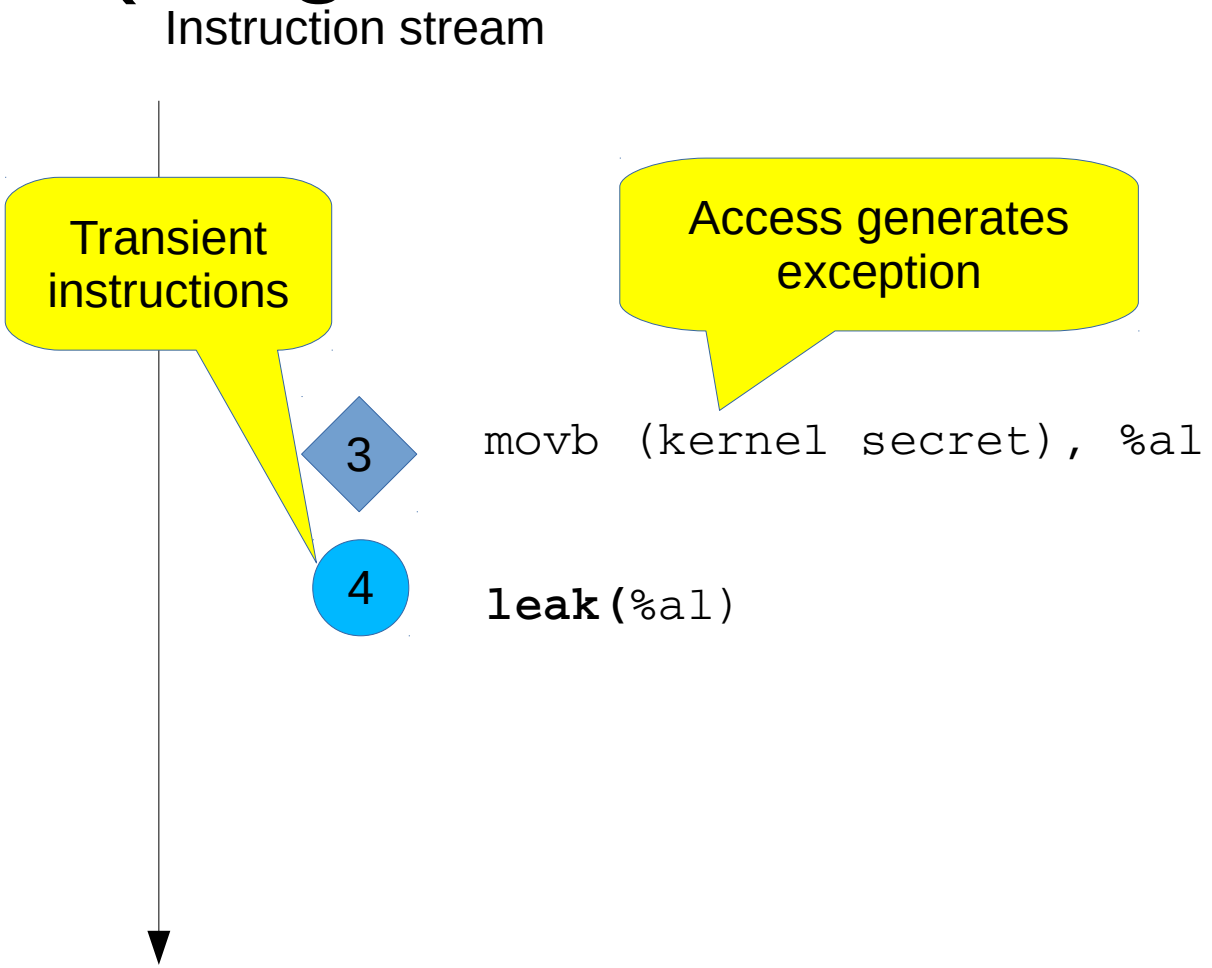
Speculative execution 101



Prerequisites to speculative execution attack

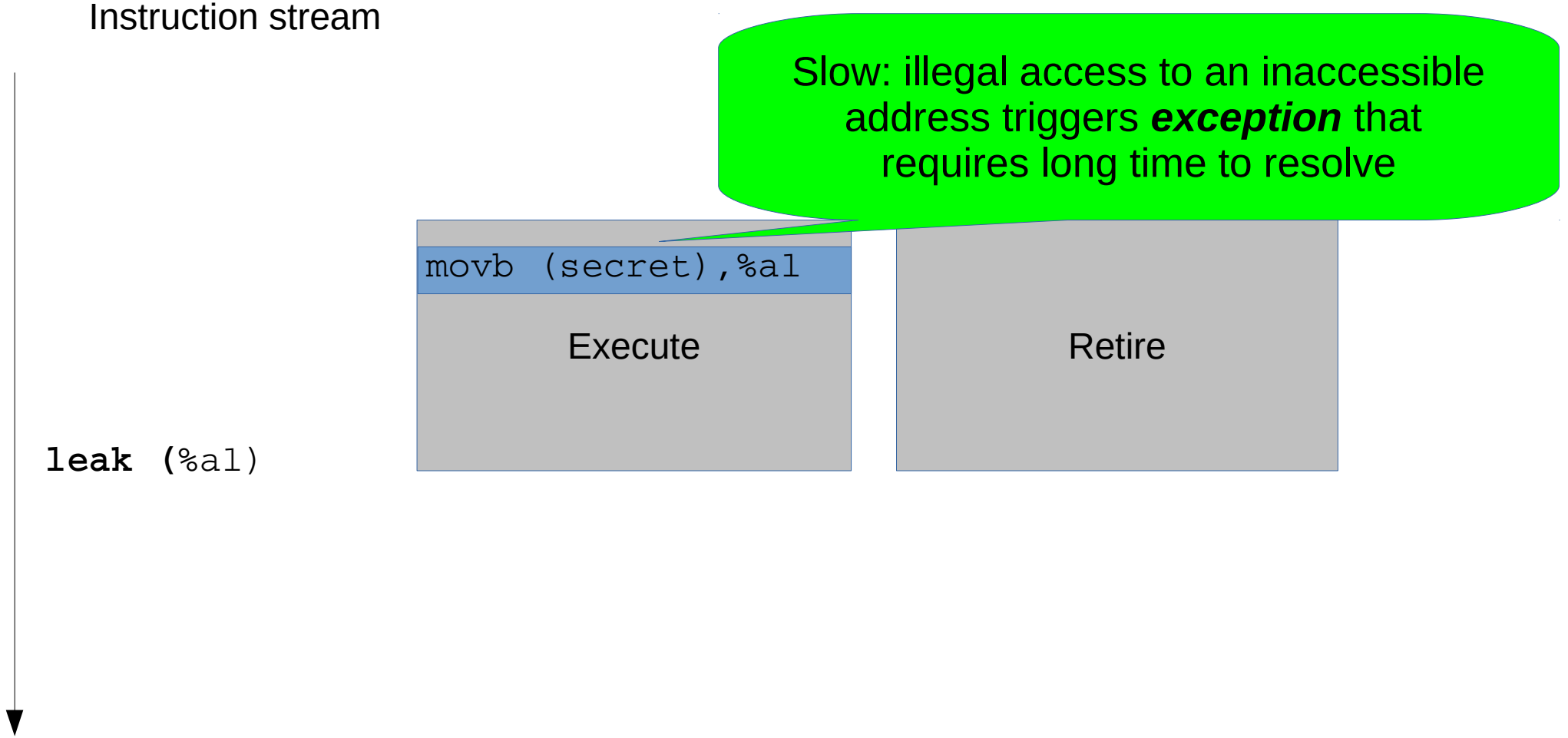
- CPU speculates ***insecurely***
- Speculative state **cannot be** rolled back: **data leak**
- Race condition: roll back vs. leaking logic
  - Attack succeeds only if $T_{speculative} < T_{slow\ access}$

Complete example (Rogue cache read – aka Meltdown)



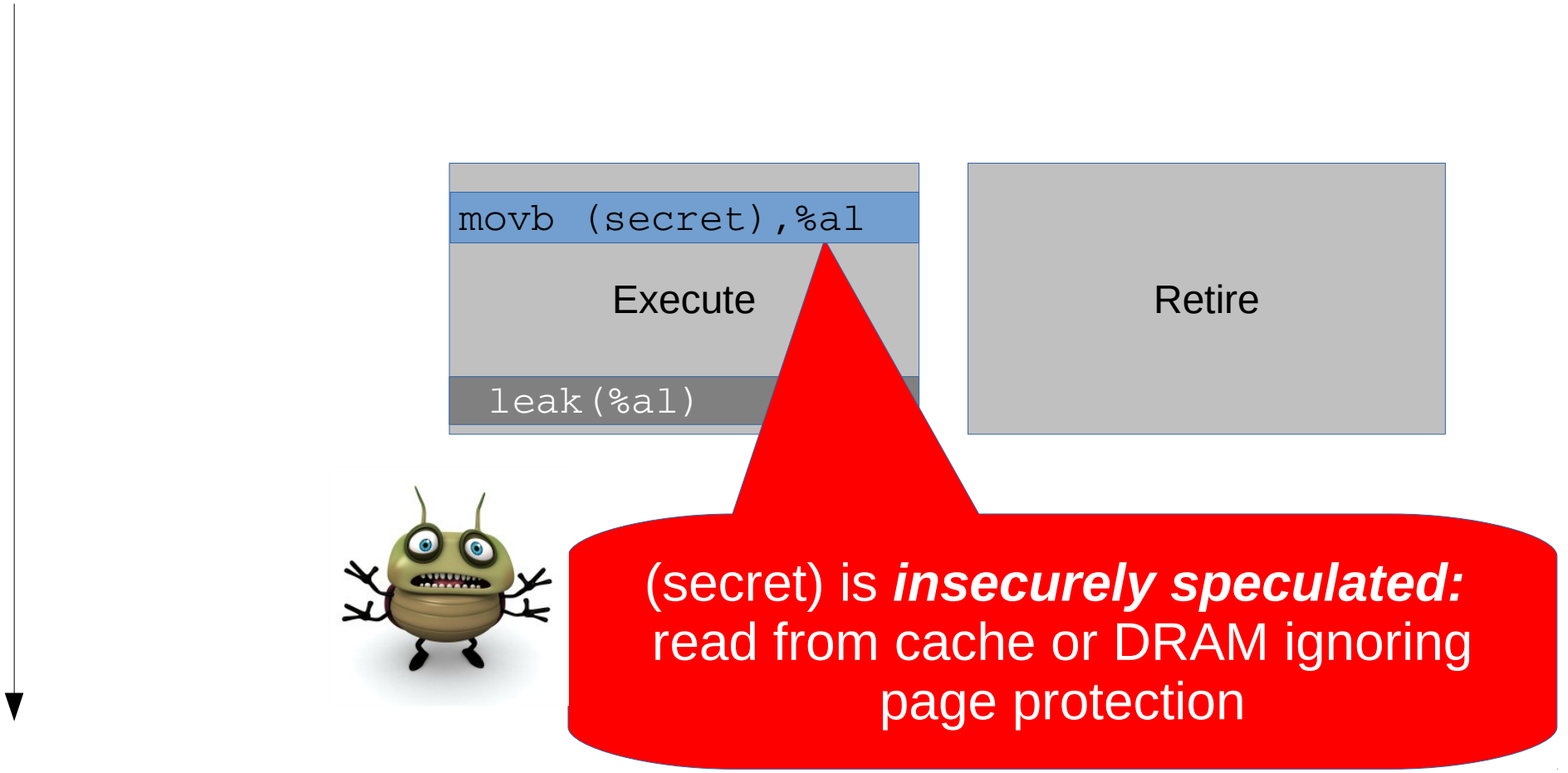
Complete example (Rogue cache read – aka Meltdown)

Instruction stream



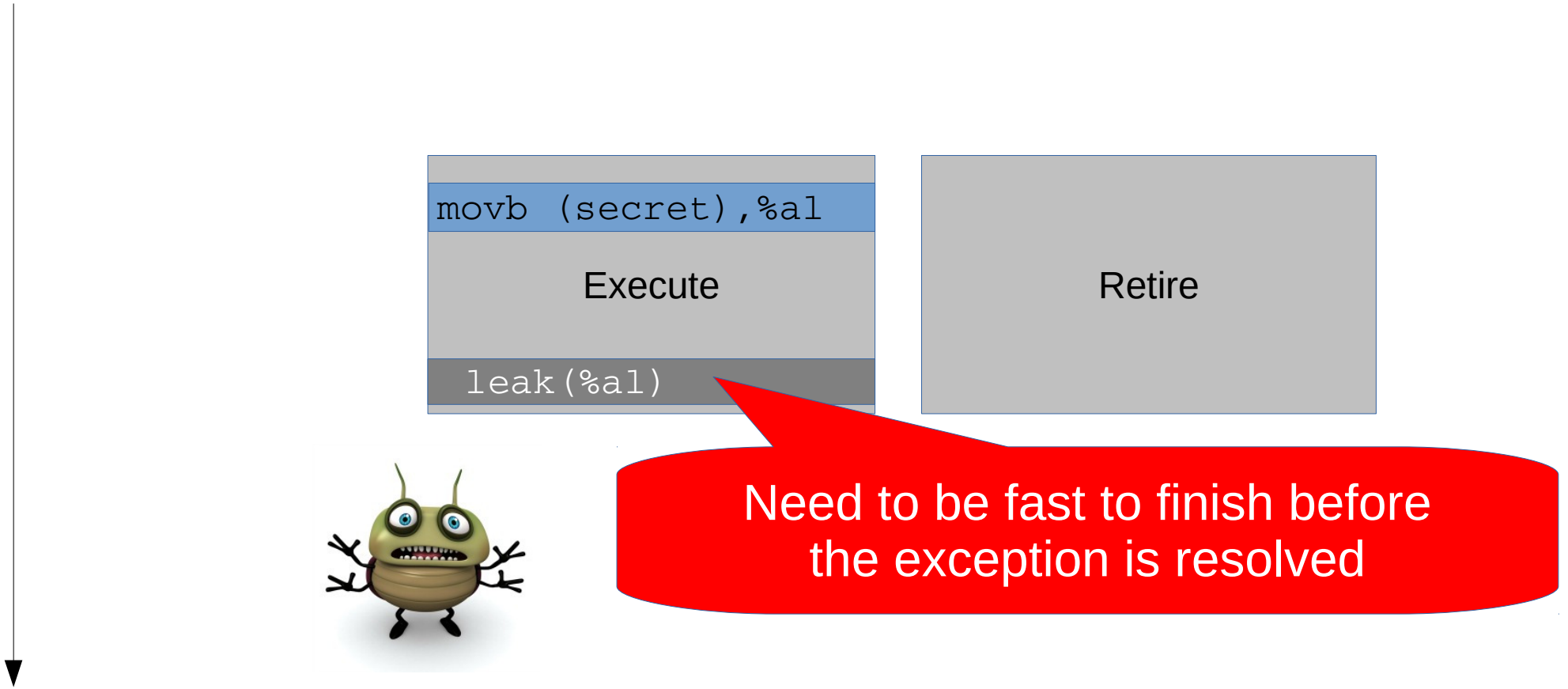
Complete example (Rogue cache read – aka Meltdown)

Instruction stream

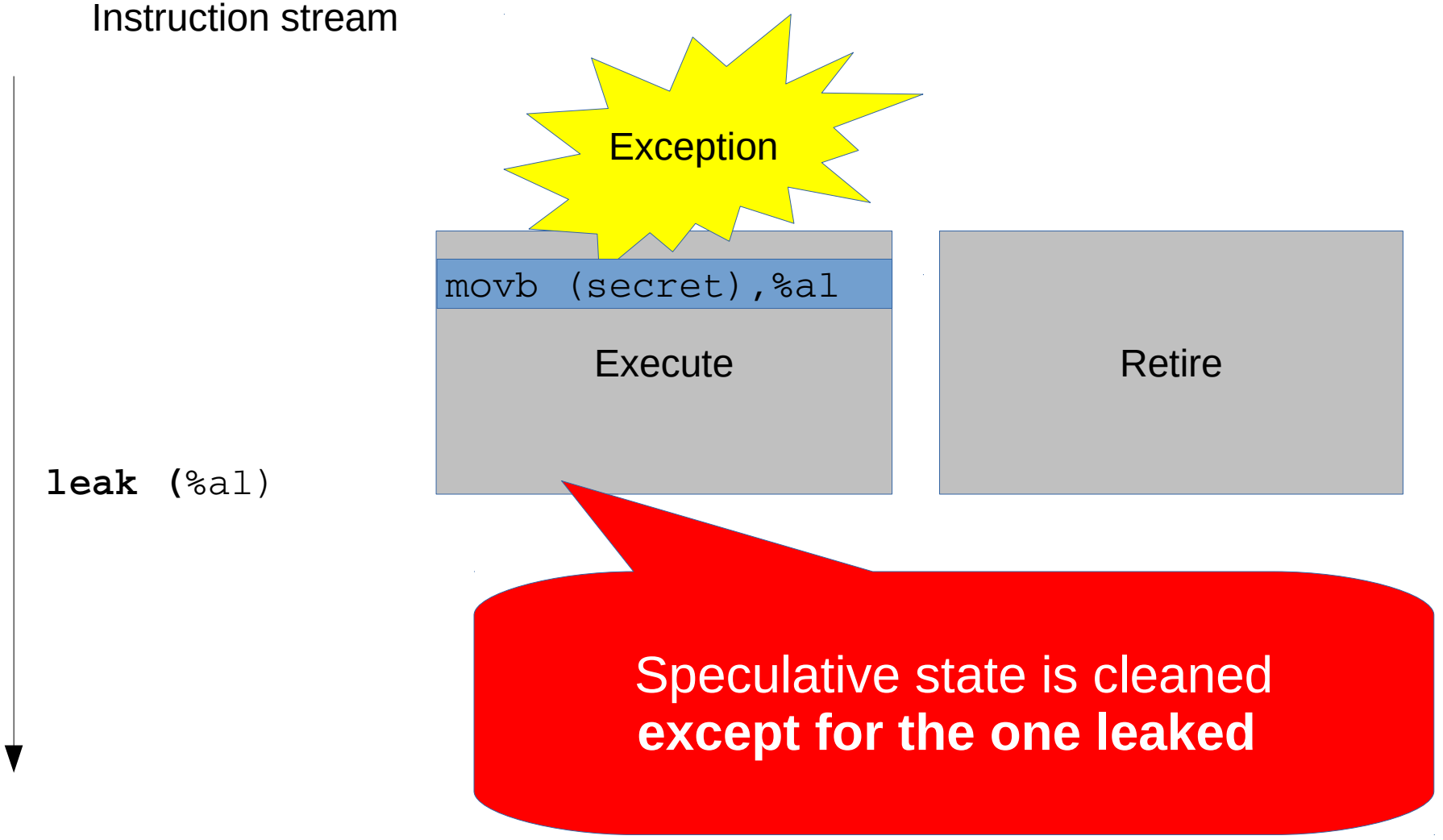


Complete example (Rogue cache read – aka Meltdown)

Instruction stream

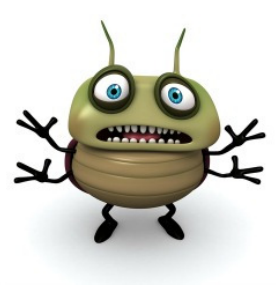


Complete example (Rogue cache read – aka Meltdown)

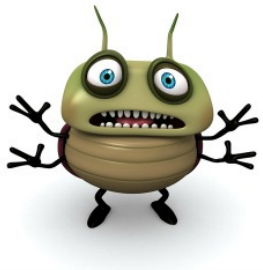


Recipe: Speculative read attacks

- **P**rovoke insecure speculation

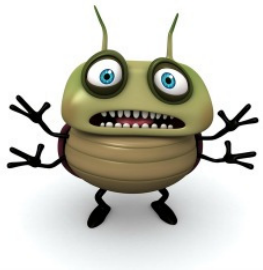


- **W**in the race
- **N**otify the attacker



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 μarch state can be rolled back
- μarch state becomes architecturally visible!
  - Caches
  - Branch predictors
  - Performance counters
  - Contention on shared resources
- Simplest: cache covert channel (Metldown/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

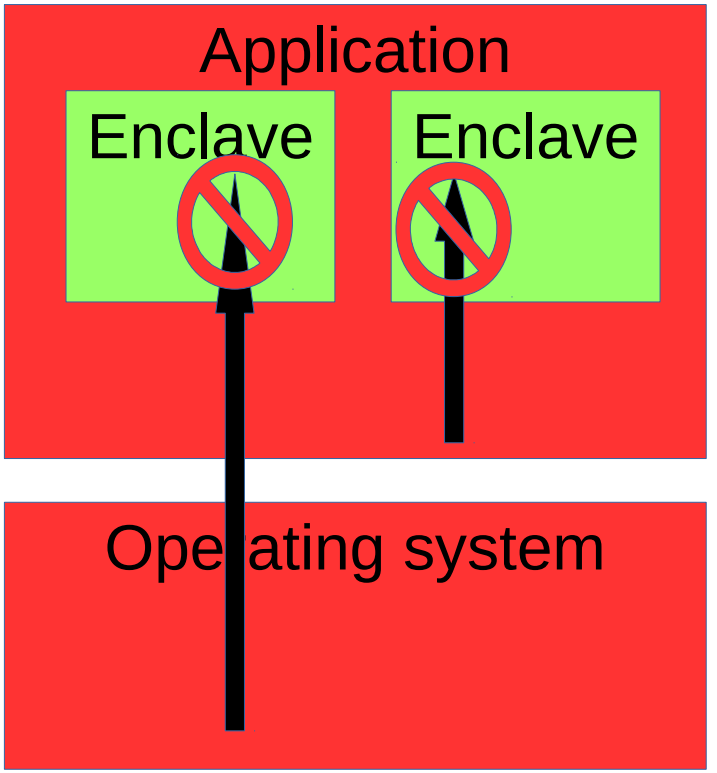
Plus some secret sauce that nobody really understands why it works

Agenda

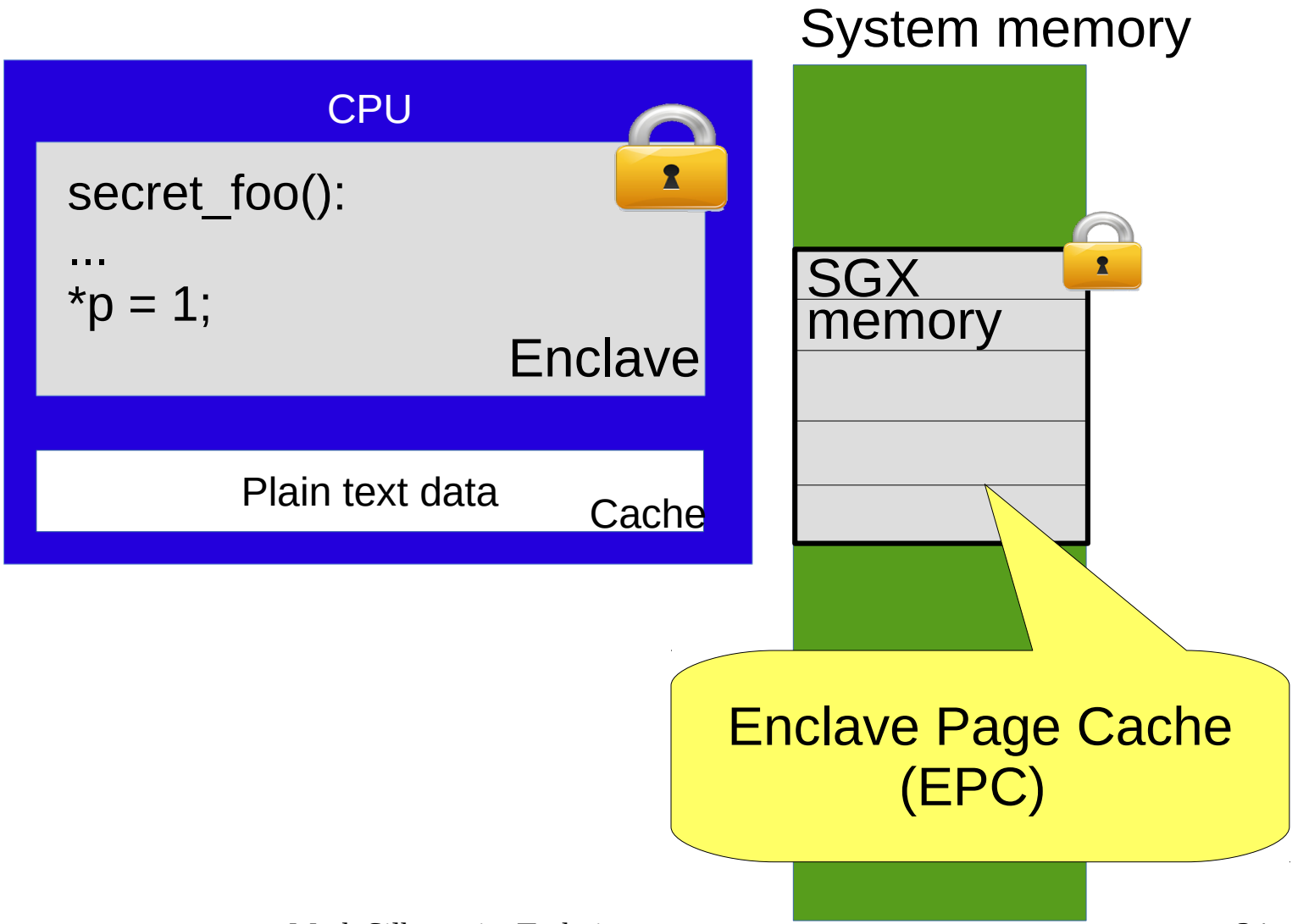
- Background on SGX
- Foreshadow
- Collateral damage on SGX
- Foreshadow-NG /L1TF
- Discussion

Background: SGX

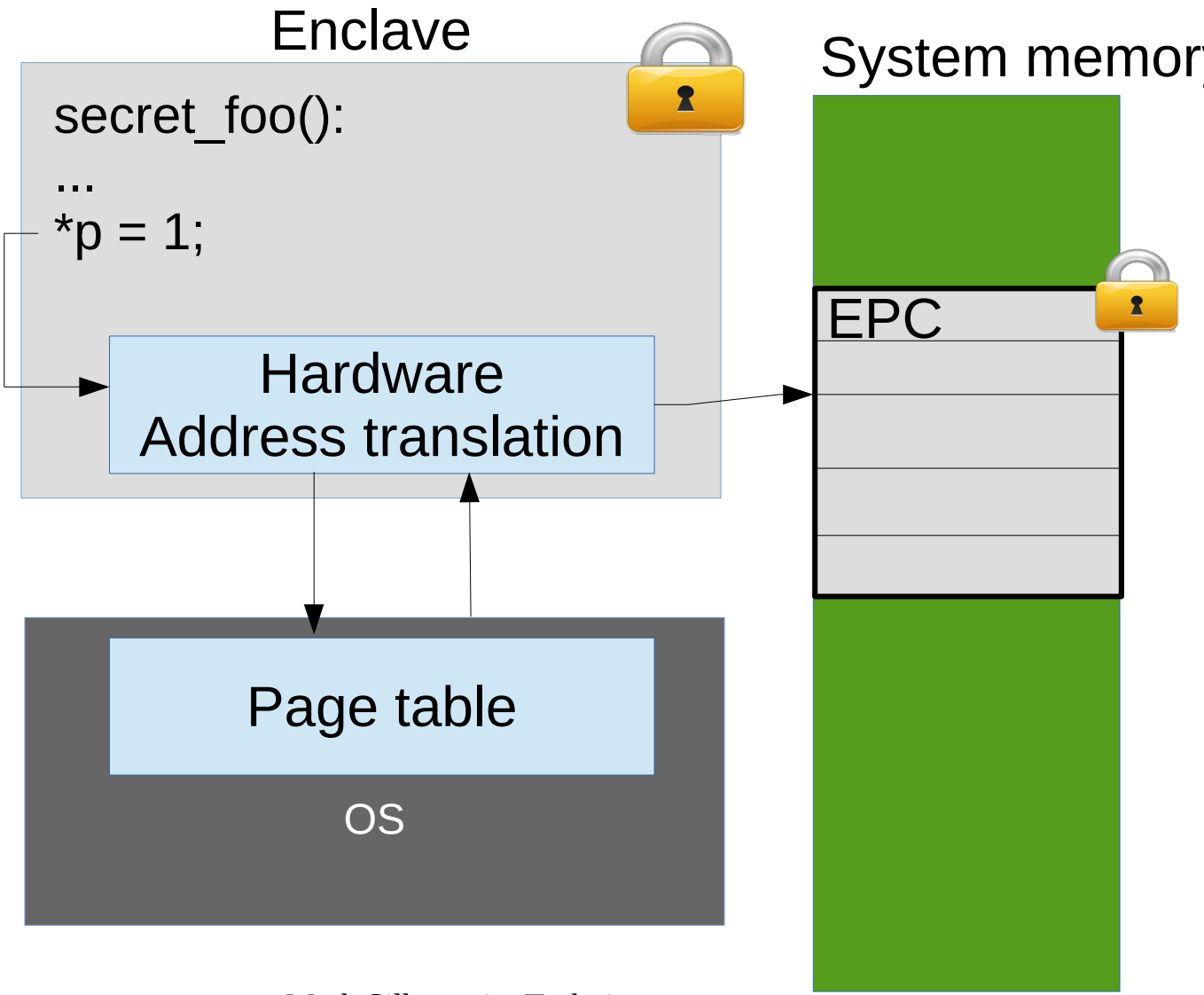
- Enclave: reversed sandbox
- Private code & data
  - Confidentiality
  - Integrity
  - Freshness
- Defends against privileged SW!
- HW acceleration
- Scales with CPU scaling



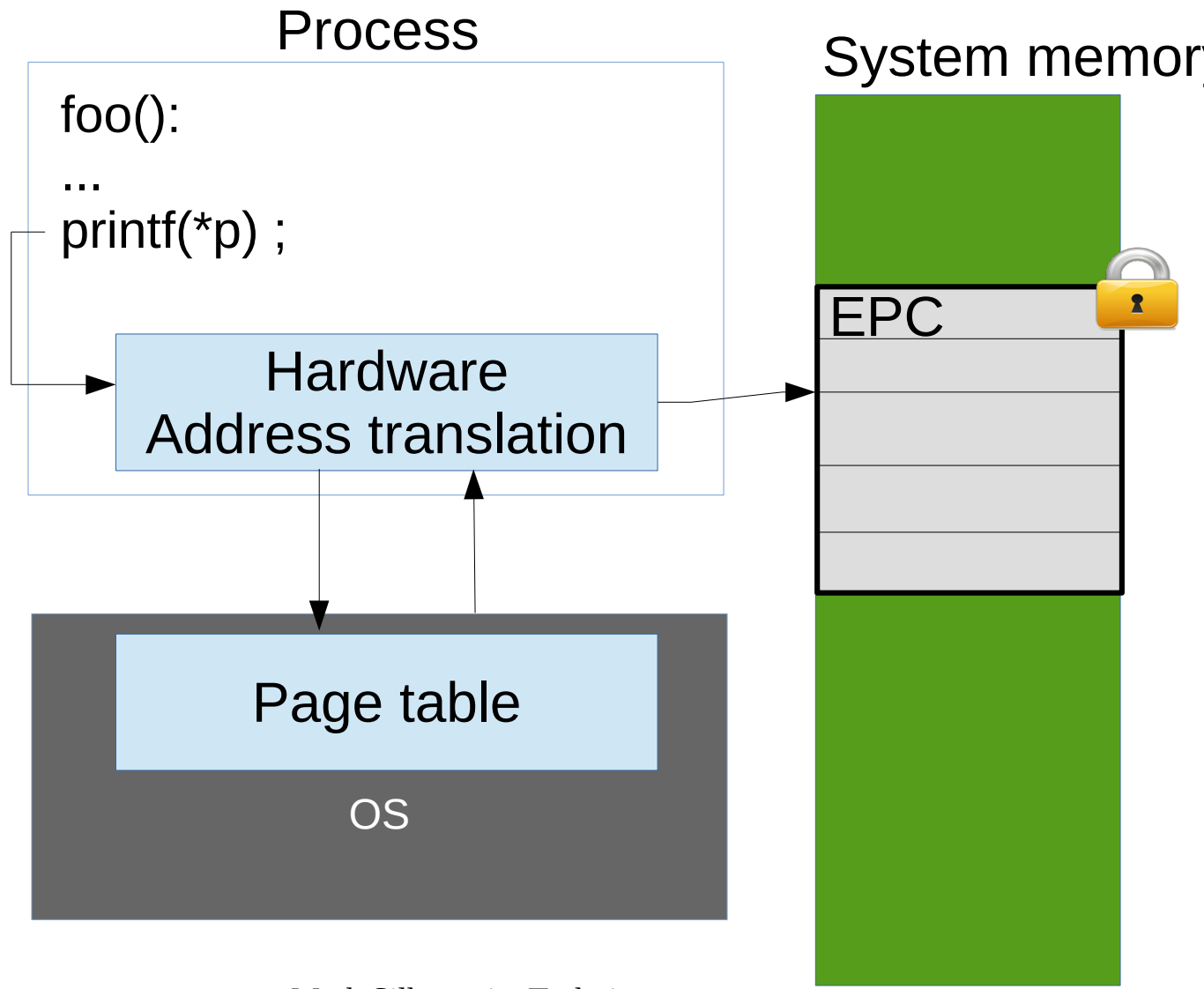
Background: SGX memory
DRAM encrypted, cache in plain text



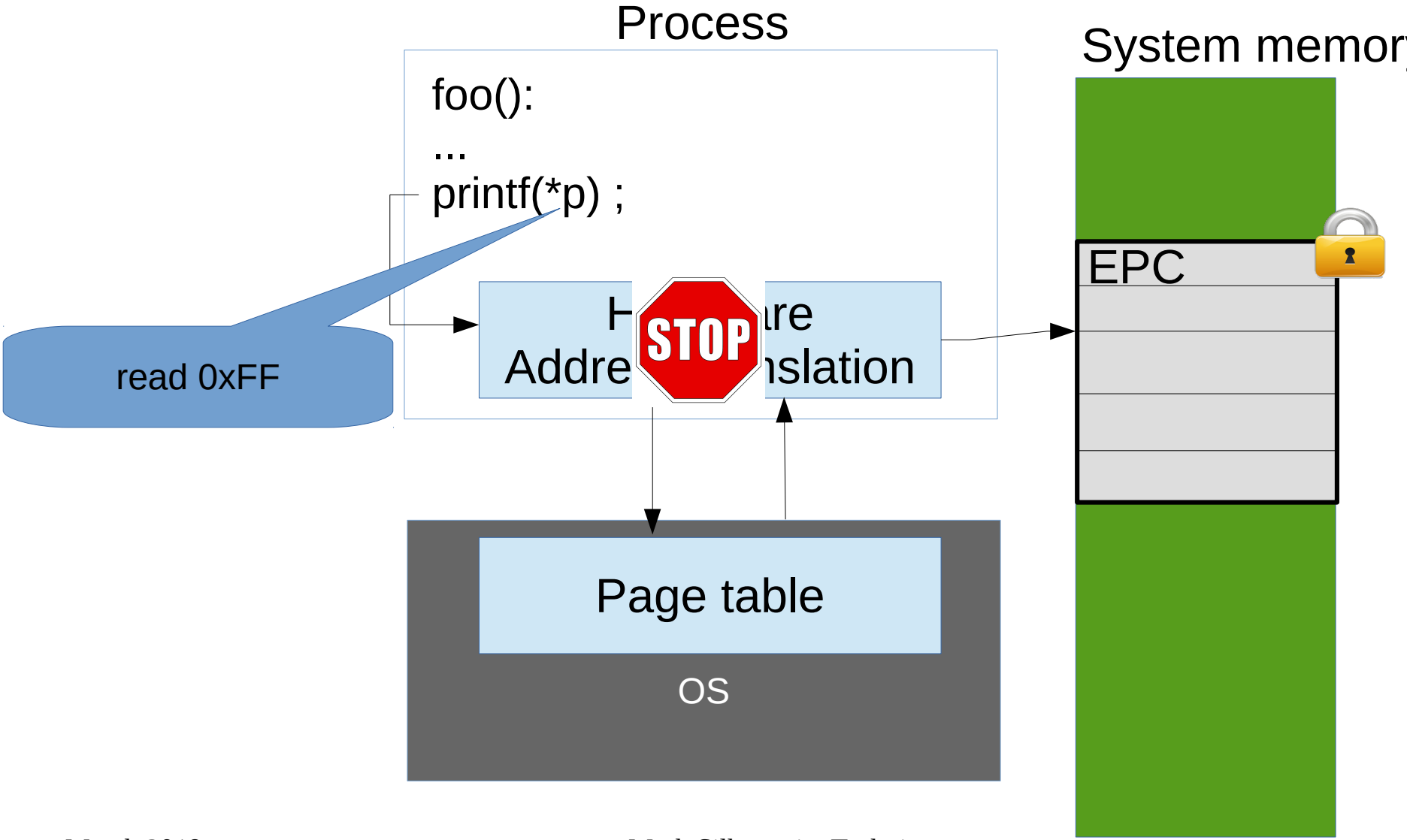
Background: Address translation in enclaves

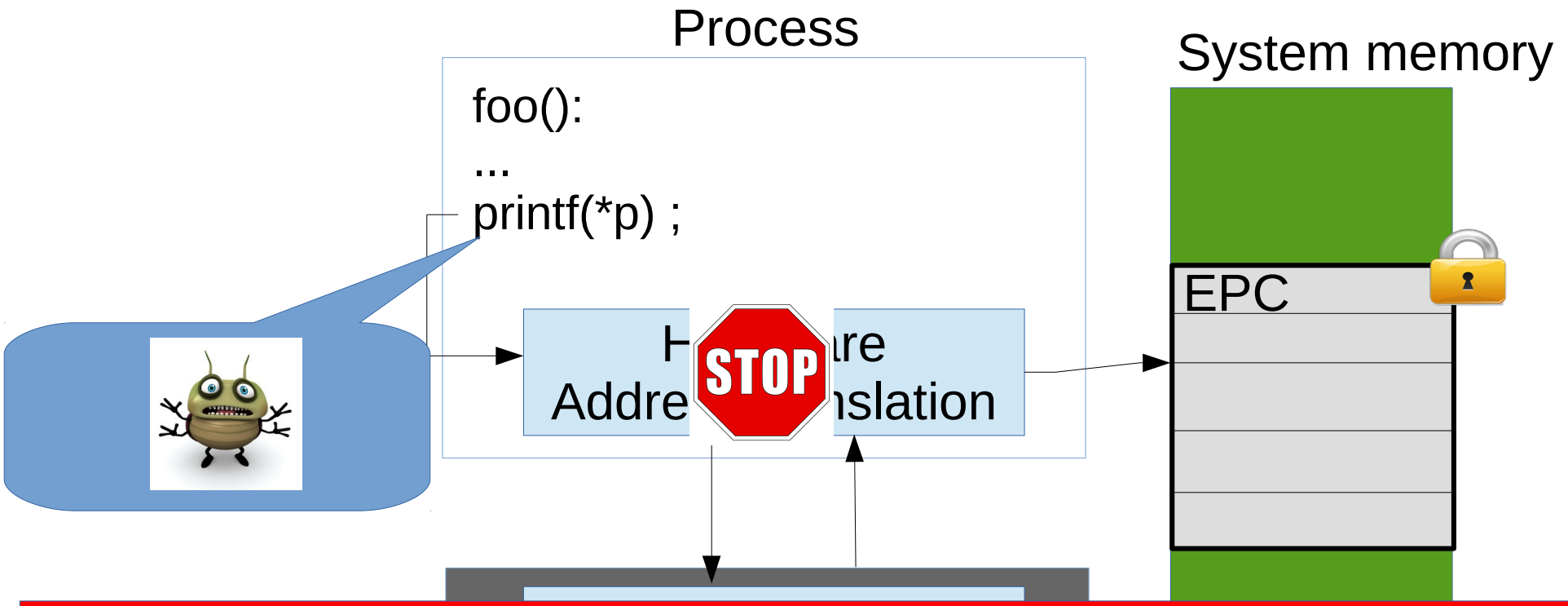


Background: SGX abort page semantics



Background:
SGX *abort page semantics*





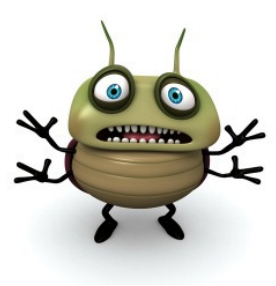
Foreshadow uses speculative execution to **leak secrets from SGX secure memory (EPC)**

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

SGX is resilient to strawman Meltdown attack

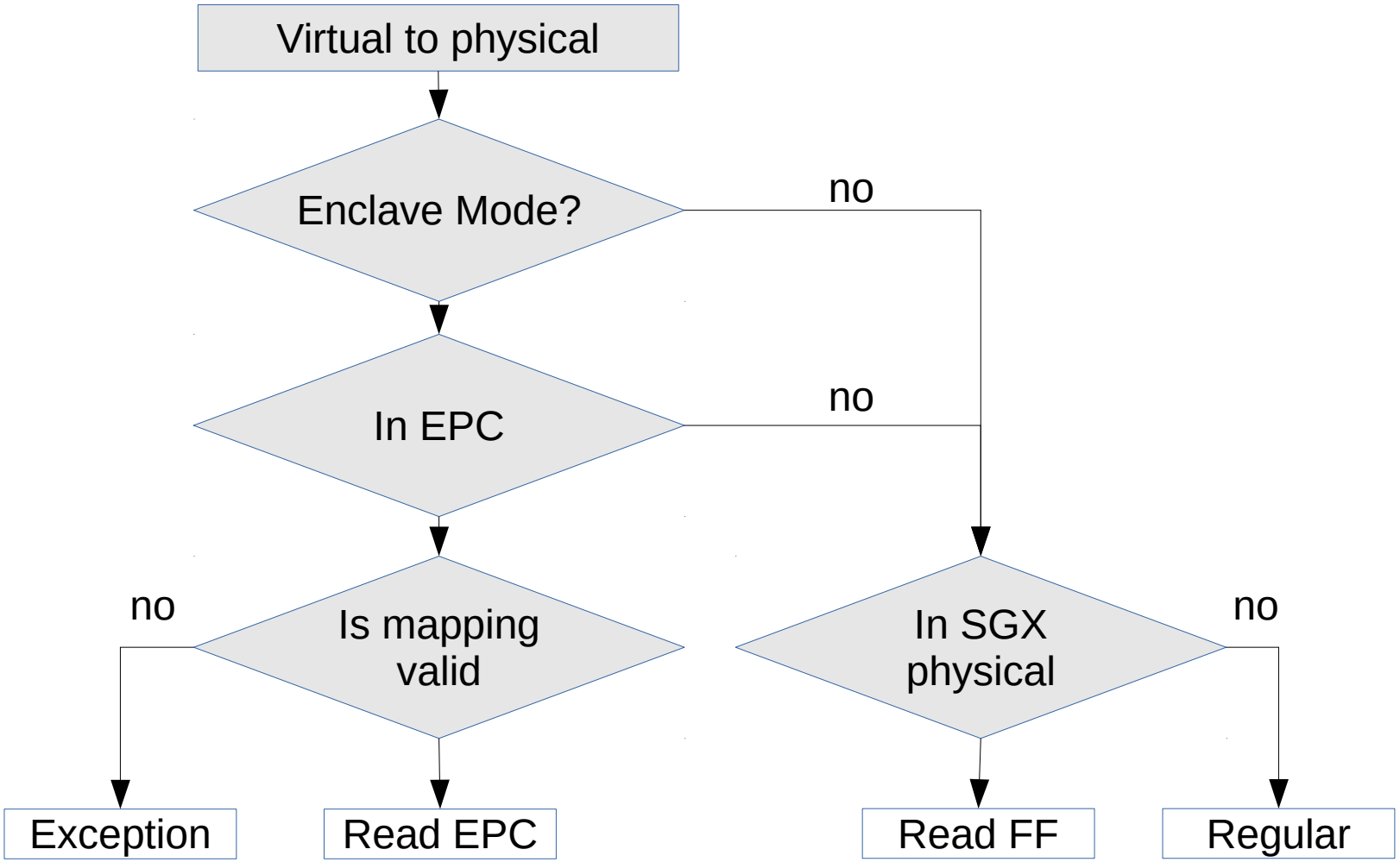
- Provoke
  - Abort page behavior suppresses exception: no speculation
- Provoke/Win – Secrets must be in L1 cache
- Notify – Same as Meltdown

Challenges of SGX attacks

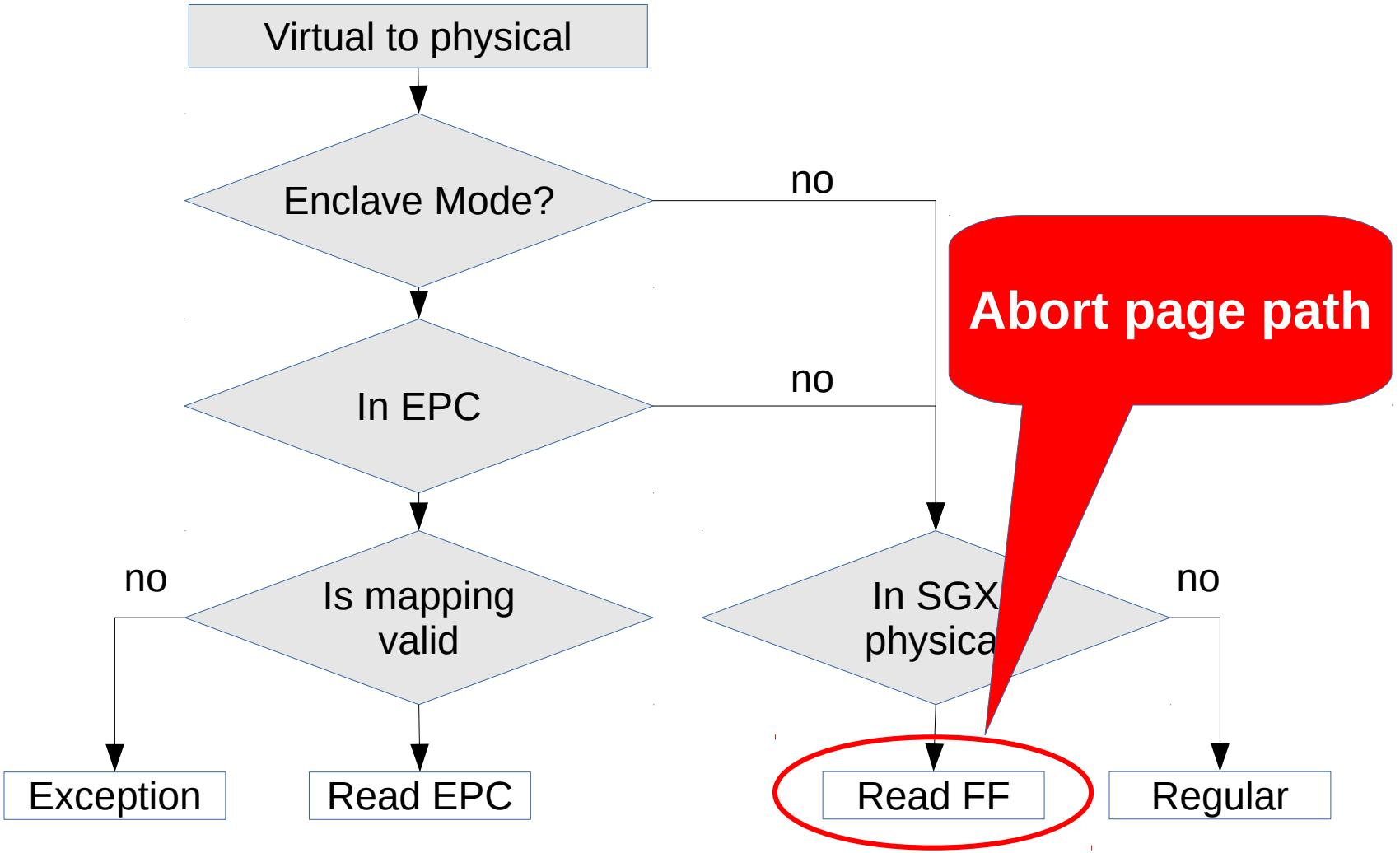
SGX is resilient to strawman Meltdown attack

- Provoke
  - Abort page behavior suppresses exception: no speculation
- Provoke/Win – Secrets must be in L1 cache
- Notify – Same as Meltdown

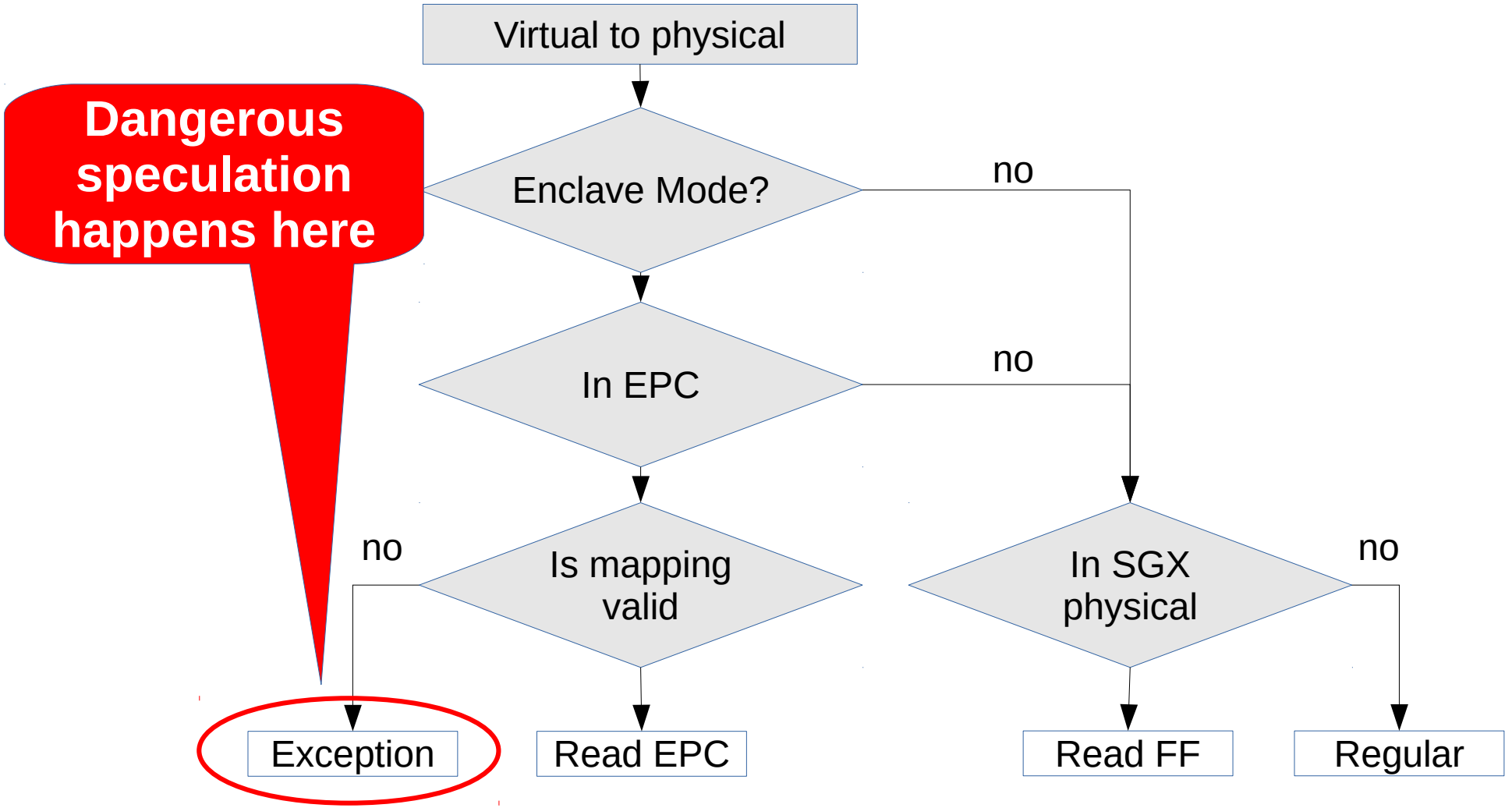
Understanding memory translation with SGX



Understanding memory translation with SGX



Understanding memory translation with SGX



Provoke 1
Overriding abort page semantics

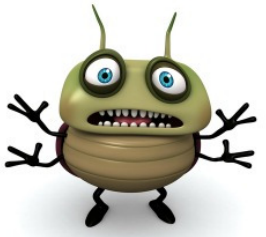
- Idea 1: access to a “not-present” EPC page
  - ***user*** calls `mprotect(epc_mem, PROT_NONE)`

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**

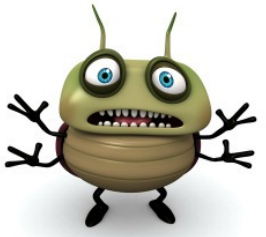


- Why does it work?
  Speculative path ignores SGX memory checks

Provoke 1
Overriding abort page semantics

This attack works from user space!

- Idea 1: access to [illegible] 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**



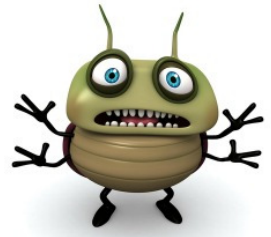
- Why does it work?
  Speculative path ignores SGX memory checks

Provoke 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

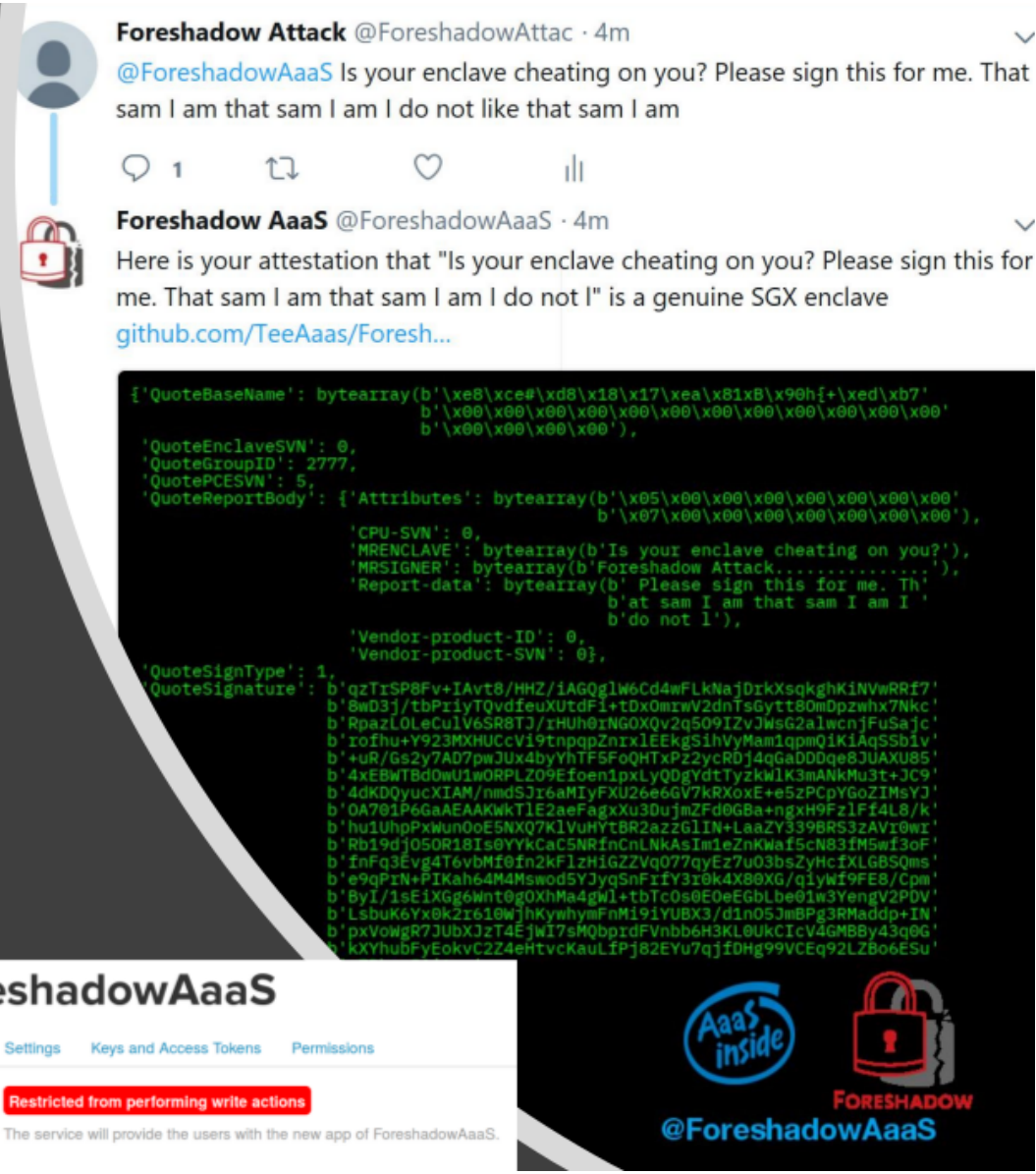
How to retrieve Intel Secret?

- 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



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 3rd by KU Leuven, Jan 23rd 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