CS 453/698: Software and Systems Security

Module: Hardware & Mobile Security

Lecture: Side-channel attacks

Adam Caulfield

University of Waterloo

Spring 2025

Reminders & Recap

Reminders:

- A4 is released
  - Due July 25th

Recap – last time we covered:

ARM TrustZone

- TZASC/TZMA: partition system resources
- NS-bit: internal to CPU, used by TZ-Aware MMU + Cache
- Secure world boots first

Android

- OS that leverages TZ
- Some features require SE

What is a side channel?

Definition: *a mechanism by which an attacker can extract information about a system or its operations by observing characteristics or indirect effects that are not part of the system's intended input or output.*

What is a side channel?

Definition: *a mechanism by which an attacker can extract information about a system or its operations by observing characteristics or indirect effects that are not part of the system's intended input or output.*

Metaphor: **Locard's exchange principle**

In forensic science, Locard's principle holds that: the perpetrator of a crime will bring something into the crime scene and leave with something from it, and that both can be used as forensic evidence → ***every contact leaves a trace***

What is a side channel?

Definition: *a mechanism by which an attacker can extract information about a system or its operations by observing characteristics or indirect effects that are not part of the system's intended input or output.*

Metaphor: **Locard's exchange principle**

In forensic science, Locard's principle holds that: the perpetrator of a crime will bring something into the crime scene and leave with something from it, and that both can be used as forensic evidence → ***every contact leaves a trace***

For computer security:

The execution of code will bring something to the hosting platform and leave with something from it, and both can be used as side channels.

Side Channels

Examples of side channels

Bandwidth consumptions (e.g., network traffic)

“James Bond” attacks

- Thermal/audio footprints
- Power consumption

General timing side channels

Cache-timing channels

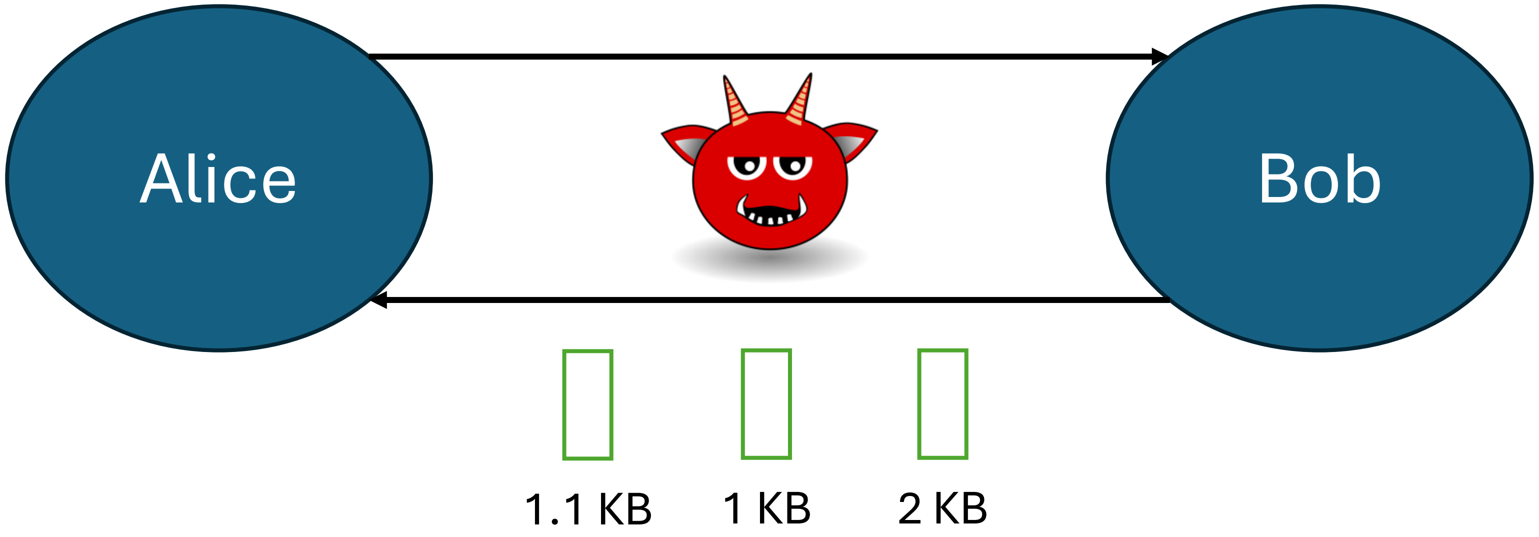
Side Channels: Bandwidth consumption

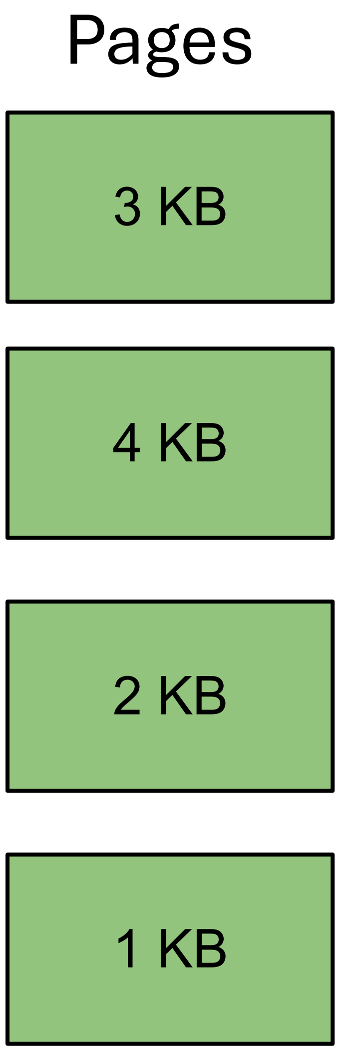
Alice and Bob Communicate:

Alice accesses health forum via encrypted channel with Bob

Adv. knows: bob hosts web forum & its content

But, cannot directly decrypt the downloaded content

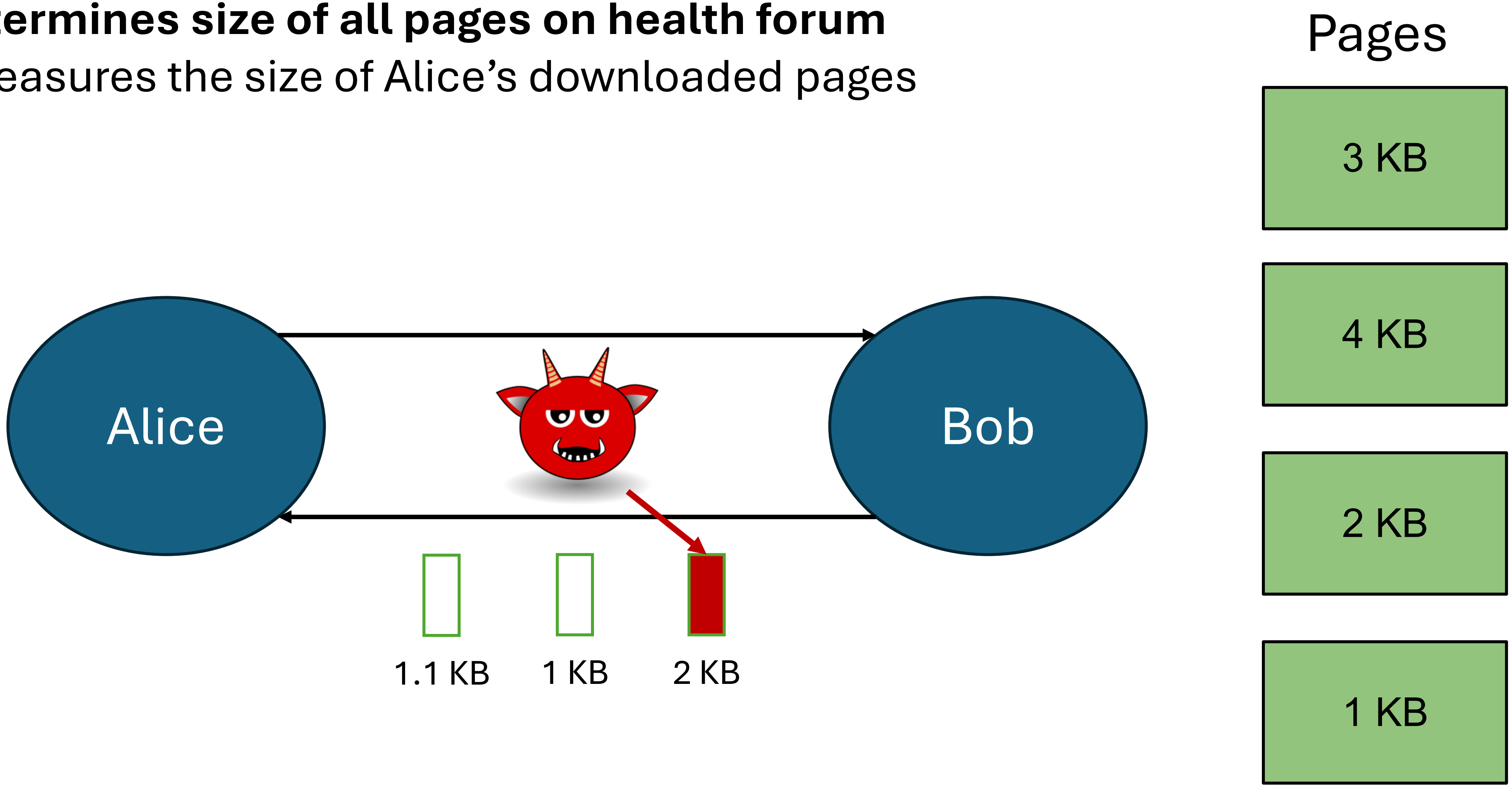




Side Channels: Bandwidth consumption

Adv. determines size of all pages on health forum

Then, measures the size of Alice's downloaded pages



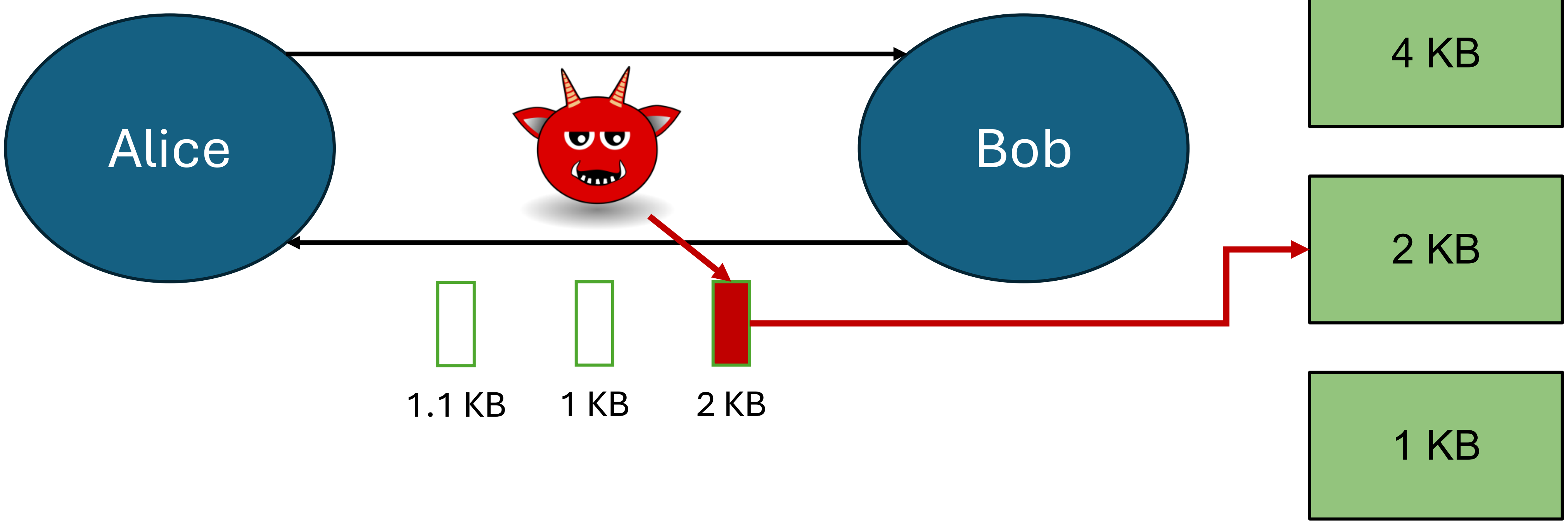
Side Channels: Bandwidth consumption

Adv. determines size of all pages on health forum

Then, measures the size of Alice's downloaded pages

Could potentially determine which webpage Alice is accessing

Potentially leaking private information

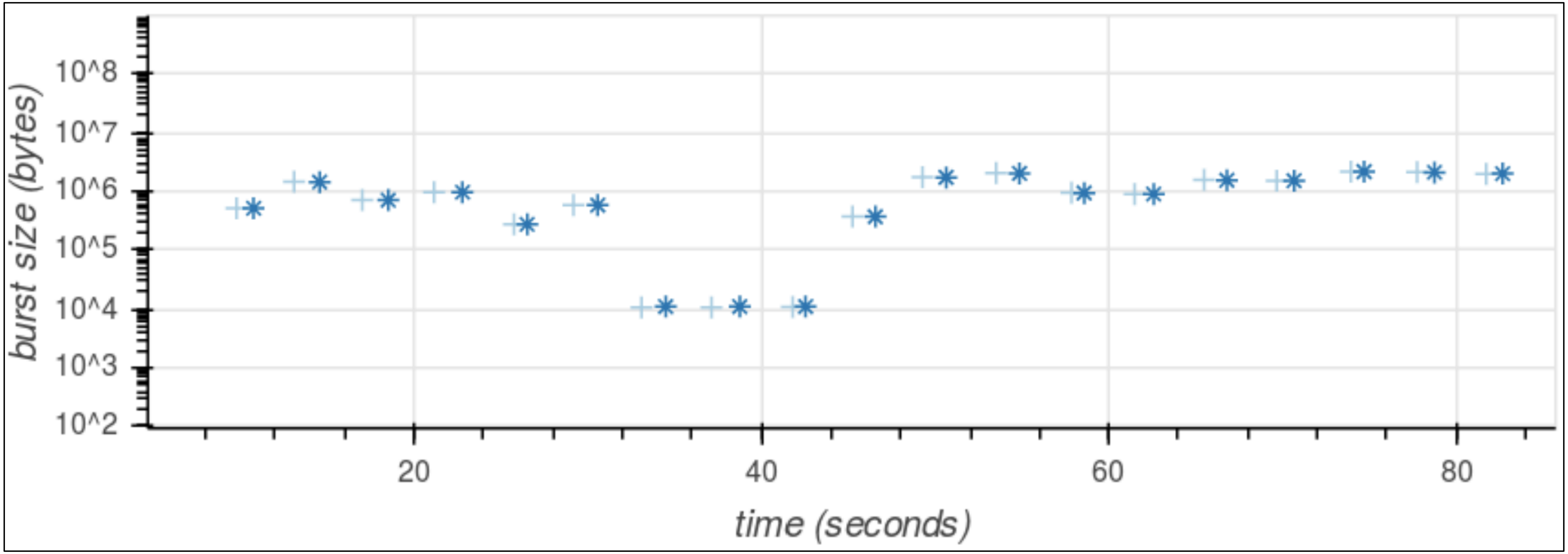


Side Channels: Bandwidth consumption

Another example:

Re-identification of Netflix video streaming

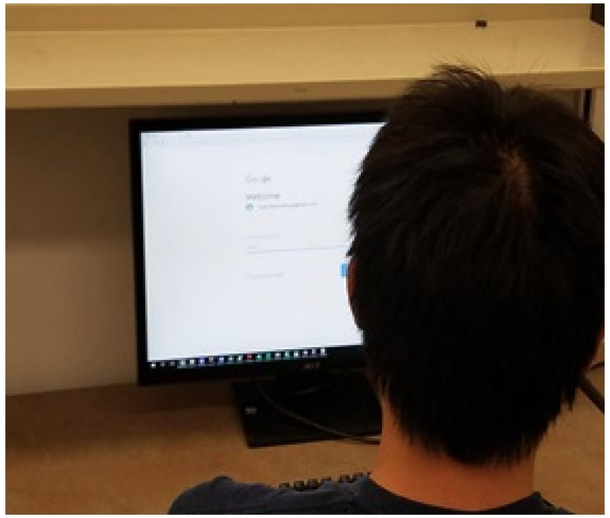
Burst sizes of a streamed scene of “Reservoir dogs”



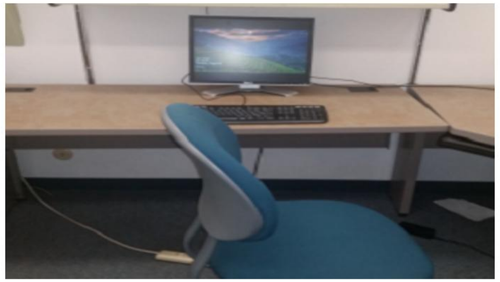
Schster et al., USENIX Security 2017

Side Channels: “”James Bonds” attacks

Any type of characteristic can be used as a side channel



(a) STEP 1: Victim Enters Password



(b) STEP 2: Victim Leaves (*Opportunistic*)

(b) STEP 2: Victim Drawn Away (*Orchestrated*)

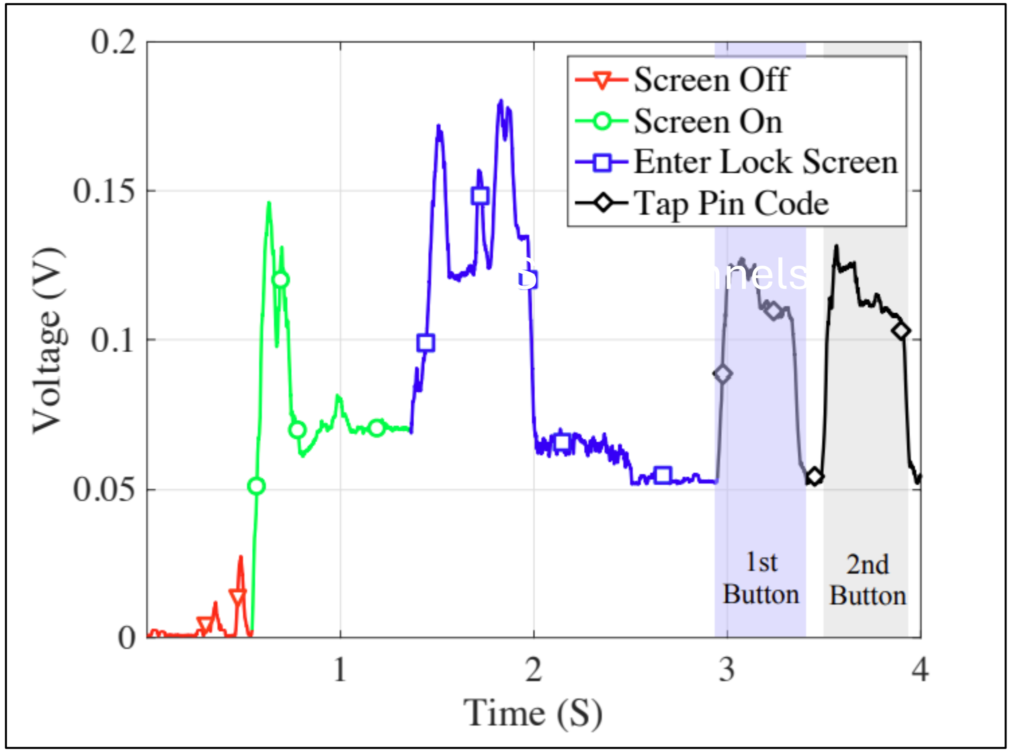
(c) STEP 3: Thermal Residues Captured

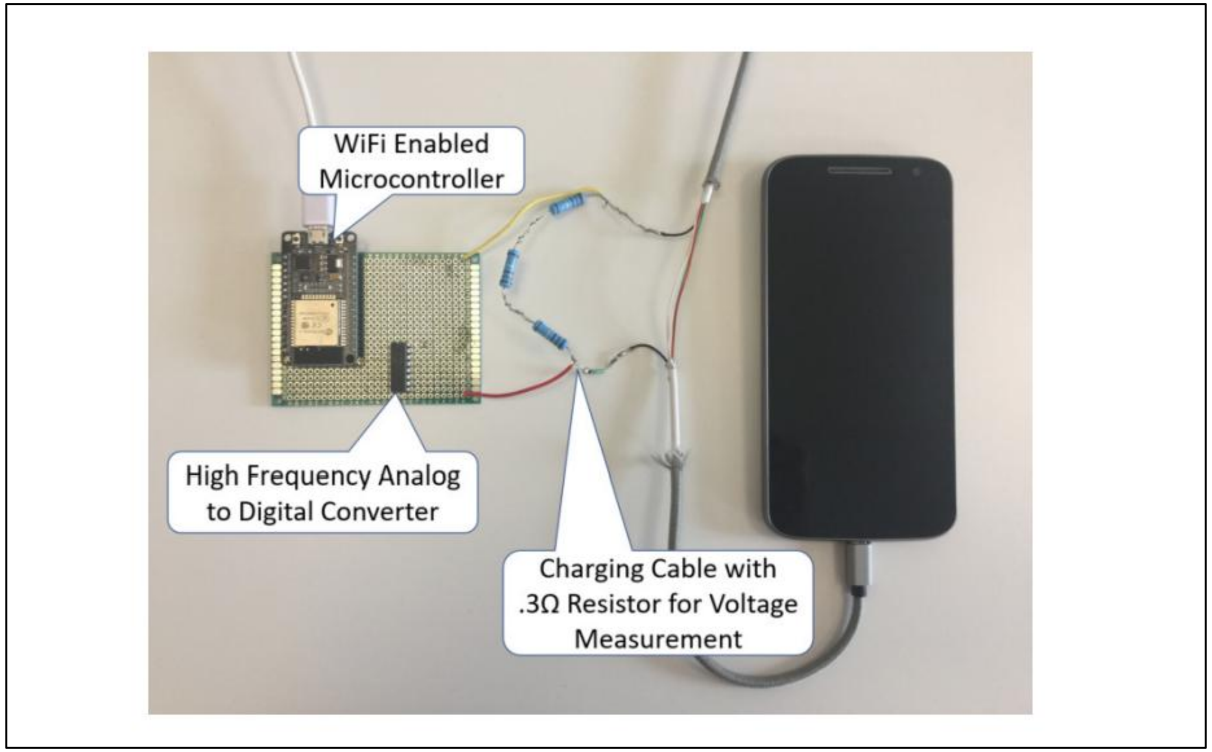
Figure 4: An Example of Thermanator Attack.

[Kaczmarek et al.: Thermal Side-Channel Attacks on Keyboard Input]

Side Channels: ‘”James Bonds” attacks

Any type of characteristic can be used as a side channel





[Cronin et al.: Charger-Surfing: Exploiting a Power Line Side-Channel for Smartphone Information Leakage]

Side Channels: Timing

Take this example function:

Finds the maximum value in a __secret__ buffer (*int * arr)*

```
int *find_max(__secret__ int *arr, int n) {
    int max_val = INT_MINIMUM;
    for (int i = 0; i < n; i++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    return max_val;
}
```

Assume an only sees Enc(max_val)…

Why could they learn max_val through timing this function?

Side Channels: Timing

Take this example function:
Finds the maximum value in a __secret__ buffer (*int * arr)*

```c
int *find_max(__secret__ int *arr, int n) {
    int max_val = INT_MINIMUM;
    for (int i = 0; i < n; i++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    return max_val;
}
```

Assume an only sees Enc(max_val)...
Why could they learn max_val through timing this function?

Side Channels: Timing

Root cause → Secret dependent execution paths

Can be exploited with remote access

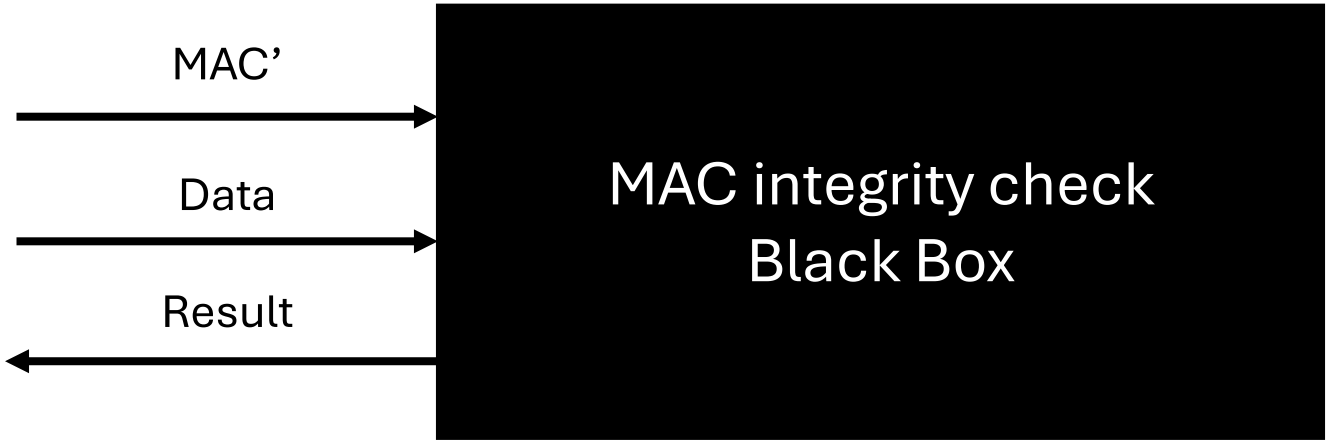
- Adv only needs to know the inputs
- Continue querying the function with different values

Side Channels: Timing

Root cause → Secret dependent execution paths

Can be exploited with remote access

- Adv only needs to know the inputs
- Continue querying the function with different values



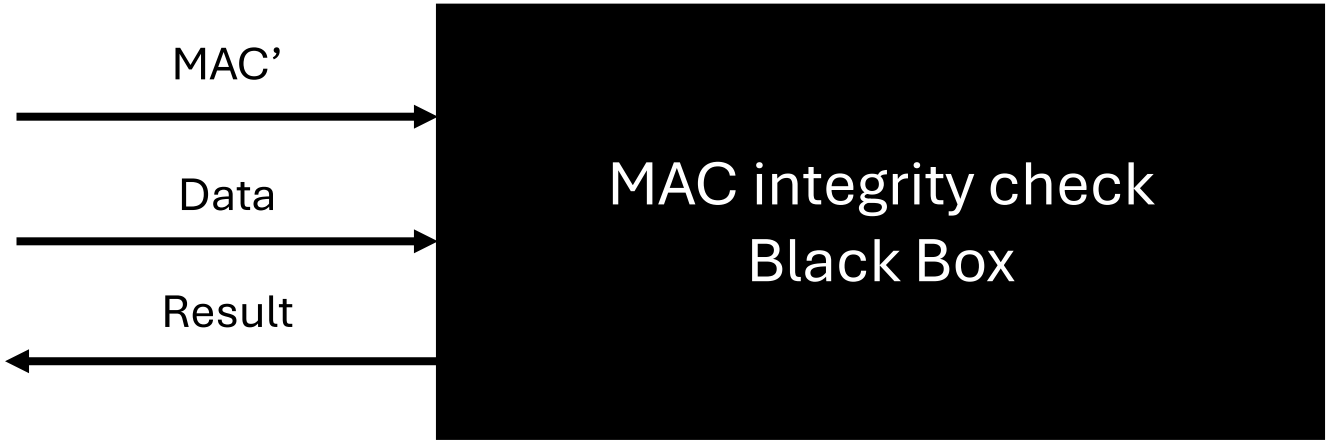

```
leaky_verify(MAC’, data):

  MAC_true = MAC(k, data)

  for i in range(0, len(MAC_true)):
    if MAC’[i] != MAC_true[i]:
      return False

  return True
```

Side Channels: Timing

Root cause → Secret dependent execution paths

Can be exploited with remote access

- Adv only needs to know the inputs
- Continue querying the function with different values




```
leaky_verify(MAC', data):

  MAC_true = MAC(k, data)

  for i in range(0, len(MAC_true)):
    if MAC'[i] != MAC_true[i]:
      return False

  return True
```

What could this black box be?.....

Side Channels: Timing

How to mitigate? → Constant-time programming

Constant-time programming

- Avoid secret-dependent if-statements
- Avoid secret-dependent memory accesses
- Avoid variable-time instructions
  - DIV, MULT (some archs.), Floating point operations

Side Channels: Constant-time Programming

Some examples: how to mitigate the previous example?

```
int *find_max(__secret__ int *arr, int n) {
    int max_val = INT_MINIMUM;
    for (int i = 0; i < n; i++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    return max_val;
}
```

Side Channels: Constant-time Programming

Some examples: how to mitigate the previous example?

```
int *find_max(__secret__ int *arr, int n) {
    int max_val = INT_MINIMUM;
    for (int i = 0; i < n; i++) {
        int predicate = arr[i] > max_val;
        max_val = (predicate * arr[i])
                | (!predicate * max_val);
    }
    return max_val;
}
```

Perform the same computation for each iteration

Record comparison Boolean into a *predicate* variable

Use value of *predicate* as a mask to set max_val for the current iteration

Side Channels: Constant-time Programming

Another examples: is this function constant-time?

```
int * get_element(
        int *arr, int size, __secret__ int index
) {
        int element = arr[index]
        return element
}
```

Side Channels: Constant-time Programming

Another examples: is this function constant-time?

```
int * get_element(
        int *arr, int size, __secret__ int index
) {
        int element = arr[index]
        return element
}
```

No → secret dependent memory access

How to patch?

Side Channels: Constant-time Programming

Another examples: is this function constant-time?

```
int * get_element(
        int *arr, int size, __secret__ int index
) {
        int element = 0
        for (int i=0; i<size; i++){
                int value = arr[i];
                int match = (i == index);
                element = (match * value) + (~match * element)
        }
        return element
}
```

Similar idea: perform memory access for each value

Record comparison of correct access to expected one

Use comparison (in `match`) as a mask to update `element`

Side Channels: Cache Timing Attacks

Architectural-specific timing attacks:

For example, exploiting cache:

- Accessing values from cache vs. has specific timing

Example: Intel CPUs

- L1 cache → 4 cycles
- L2 cache → 12 cycles
- L3 cache → 26-31 cycles
- DRAM memory → 120+ cycles

Side Channels: Cache Timing Attacks

Architectural-specific timing attacks:

For example, exploiting cache:

- Accessing values from cache vs. has specific timing

Example: Intel CPUs

- L1 cache → 4 cycles
- L2 cache → 12 cycles
- L3 cache → 26-31 cycles
- DRAM memory → 120+ cycles

Some CPU instructions enable unprivileged cache maintenance

- `prefetch` → suggest CPU to load data into the catch
- `clflush` → throw out data from all caches

Side Channels: Cache Timing Attacks

Concrete scenario:

- You run a secure program on a machine, and the program does one of two things:
  - Encrypt()
  - Decrypt()
- You do not want anyone to know whether your program is encrypting a message or decrypting a message
  - Assuming trust in OS and hardware for now
- The binary of your program is available
- Attackers run their programs on the same machine
- Their goal is to infer which operation your program is running

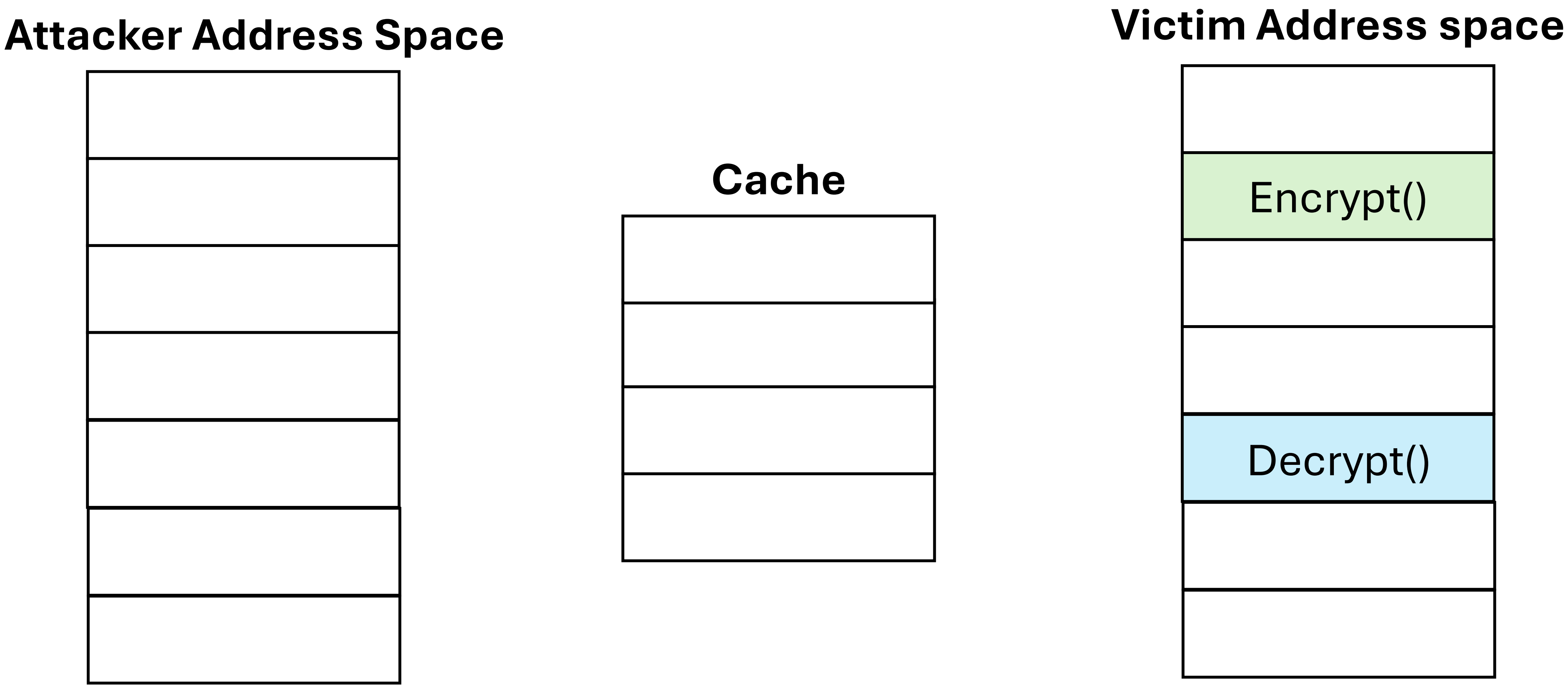
Side Channels: Cache Timing Attacks

Common access-driven cache attack strategies:

Flush + Reload

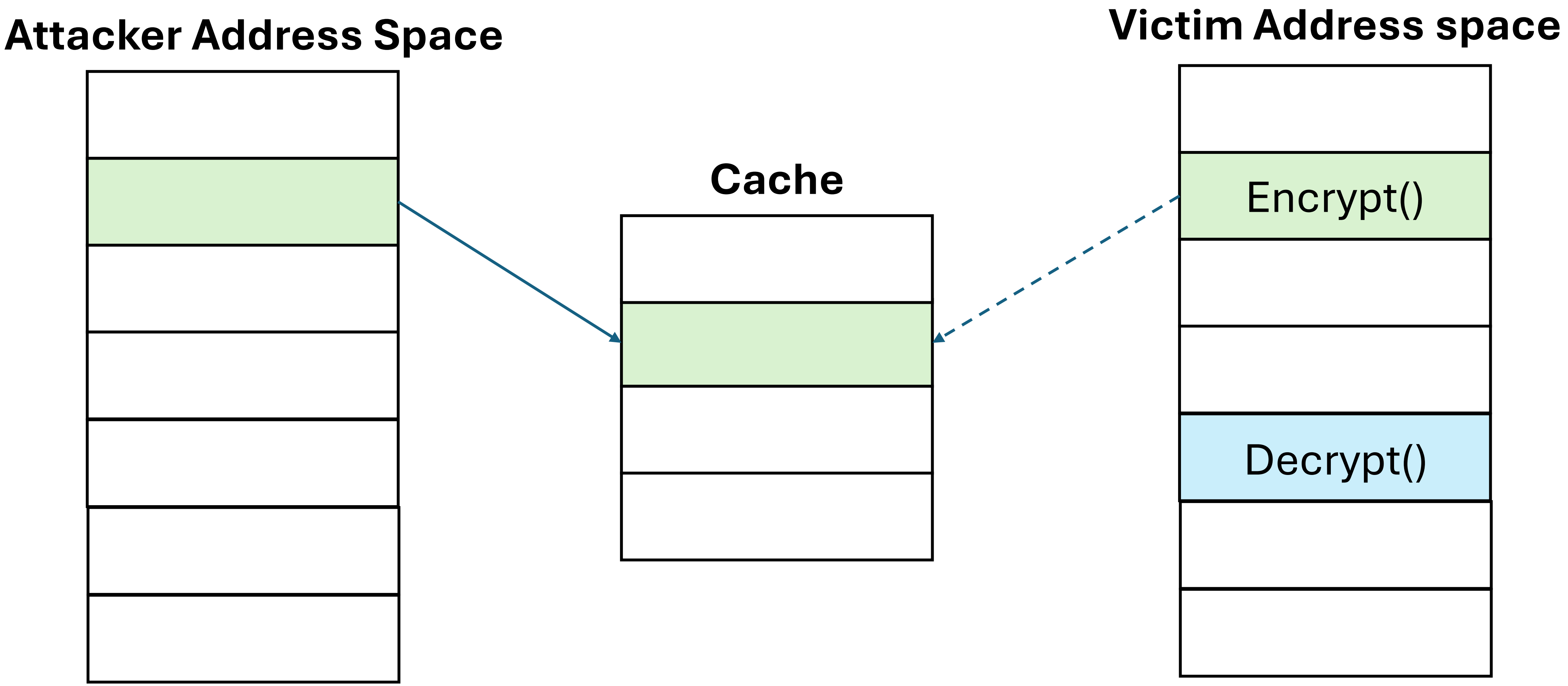
Prime + Probe

Flush + Reload



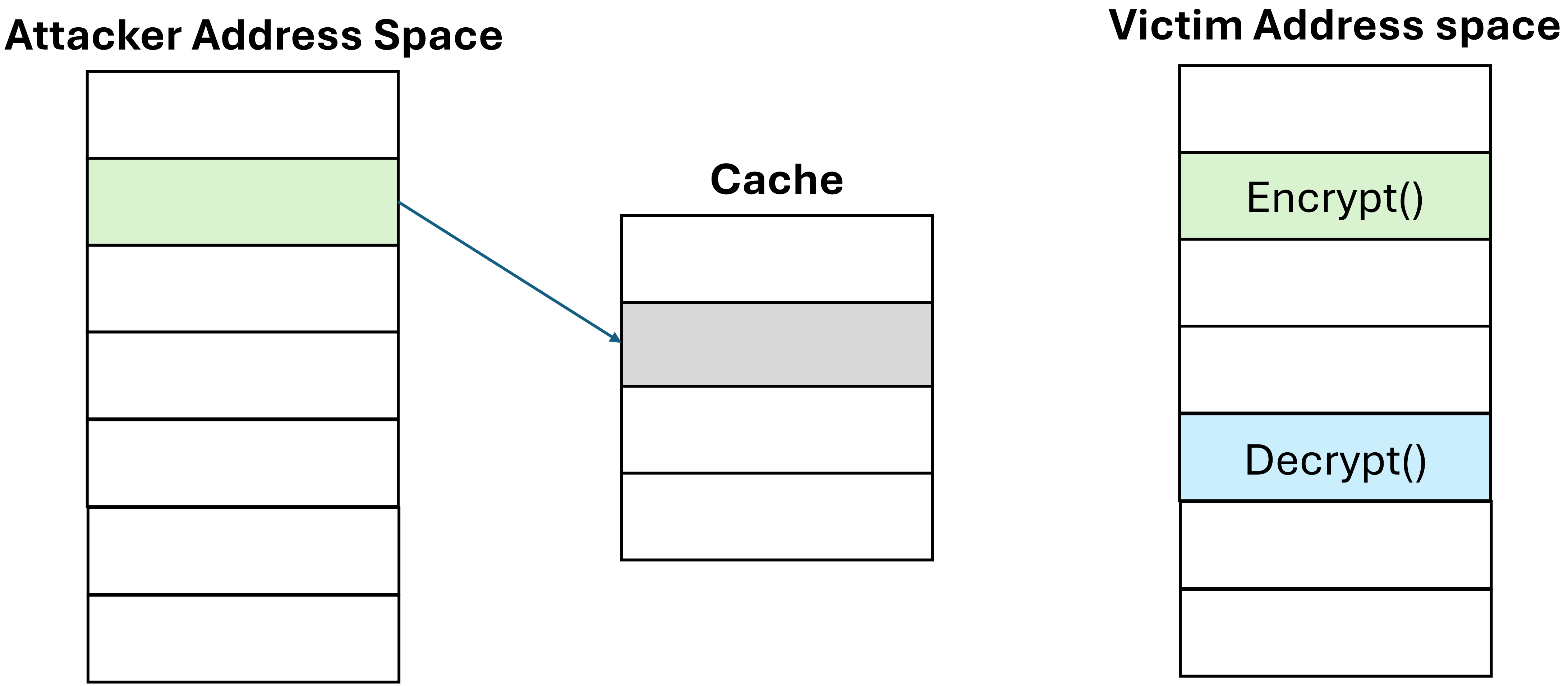
Init: victim program loaded while cache is empty

Flush + Reload



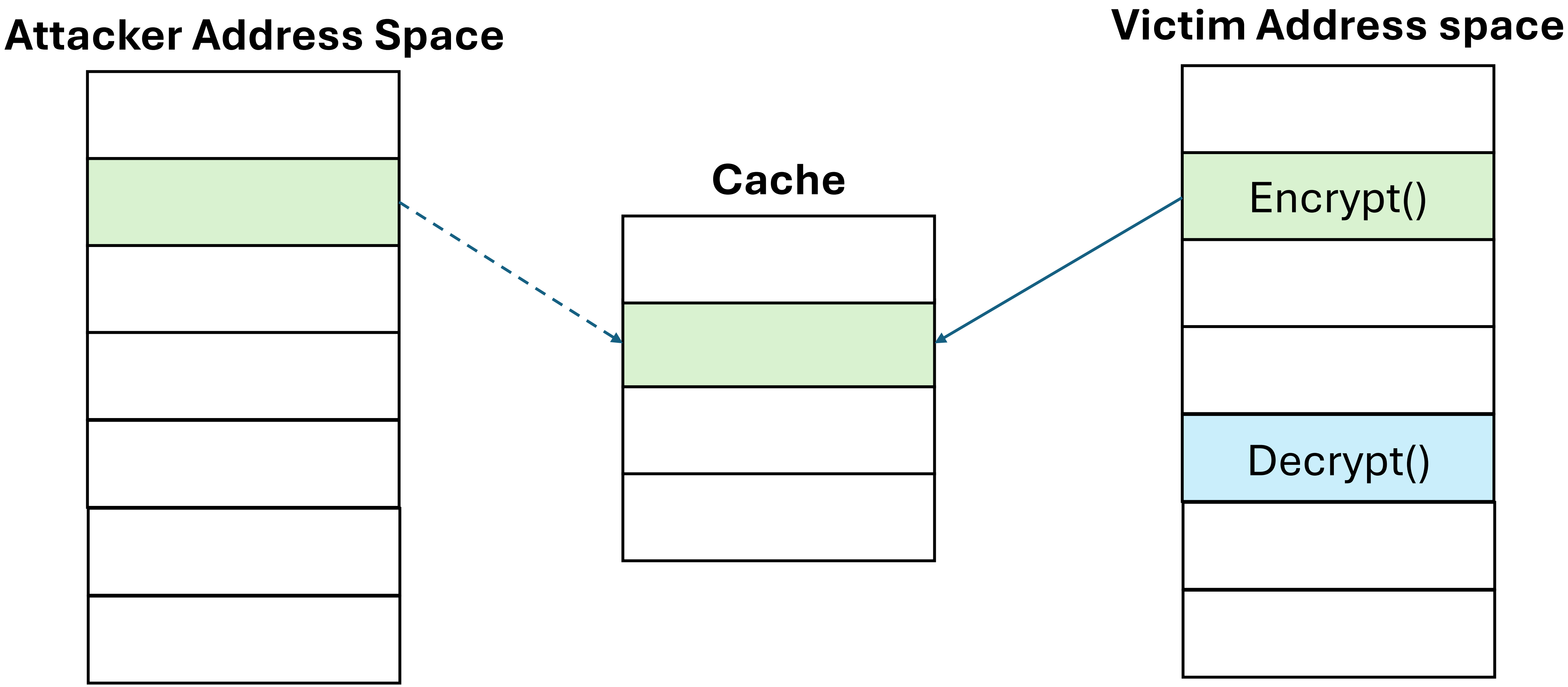
Step 1: attacker loads the Encrypt() code into its address space

Flush + Reload



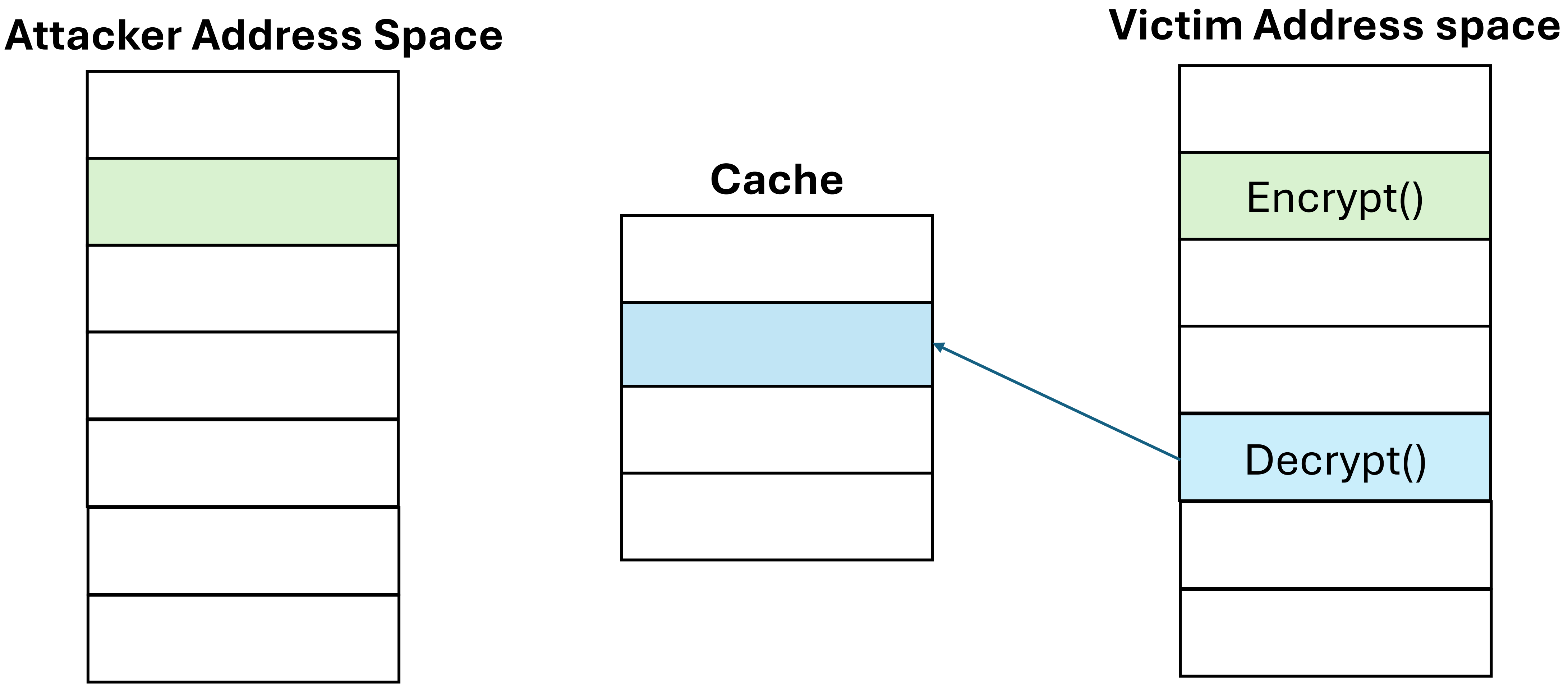
Step 2: attacker flushes the cache

Flush + Reload



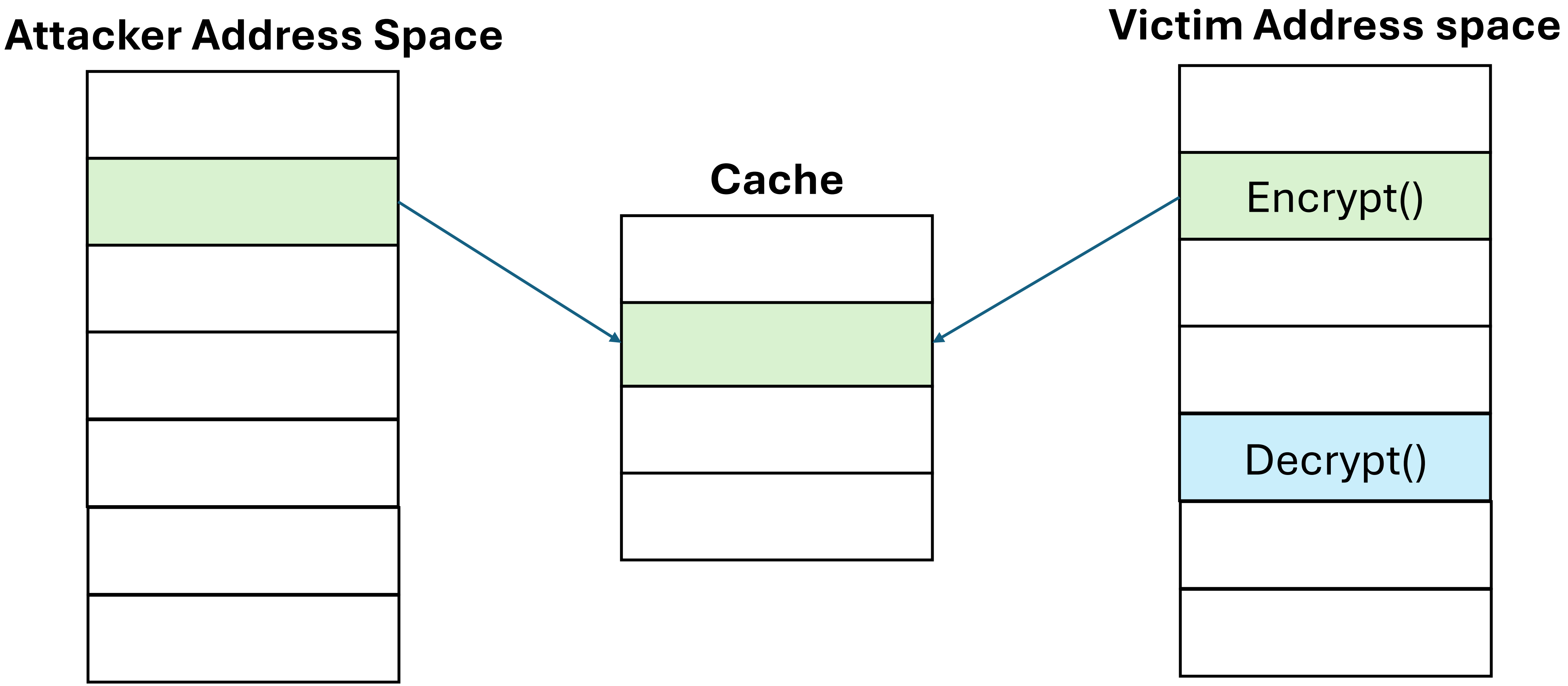
Step 3a: victim performs Encrypt() operation

Flush + Reload



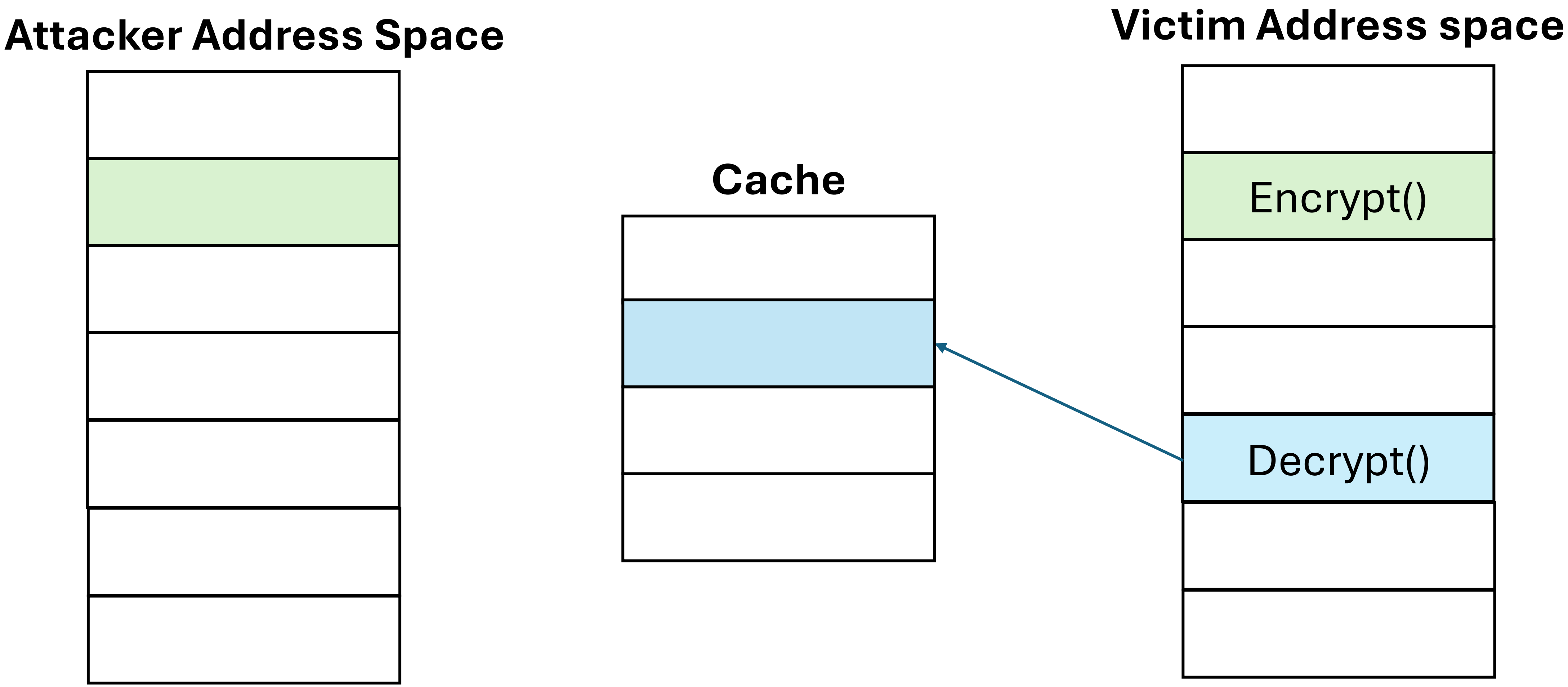
Step 3b: victim performs Decrypt() operation

Flush + Reload



Step 4: attacker calls encrypt and times it → if occurred after 3a, will be fast

Flush + Reload



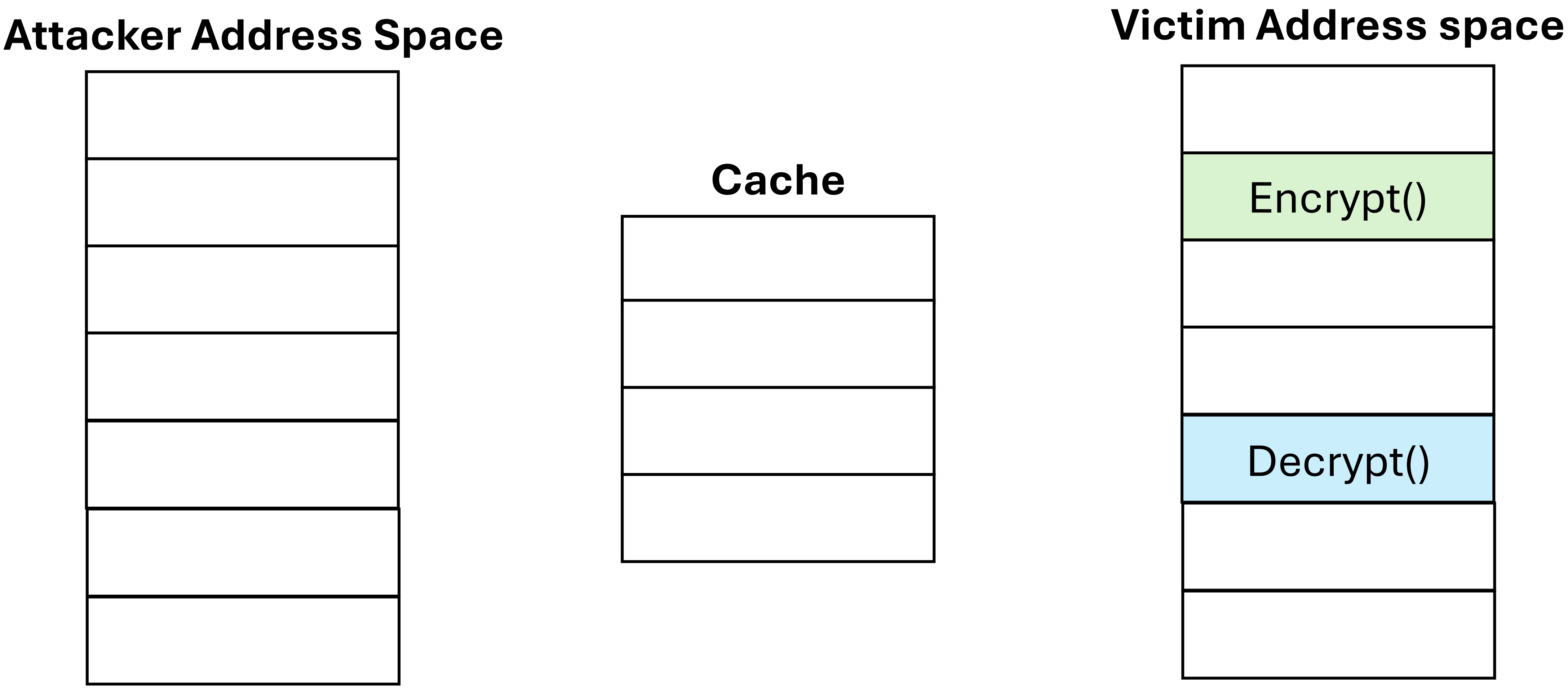
Step 4: if after step 3b, slow because it is no longer in cache

Flush + Reload

Summary:

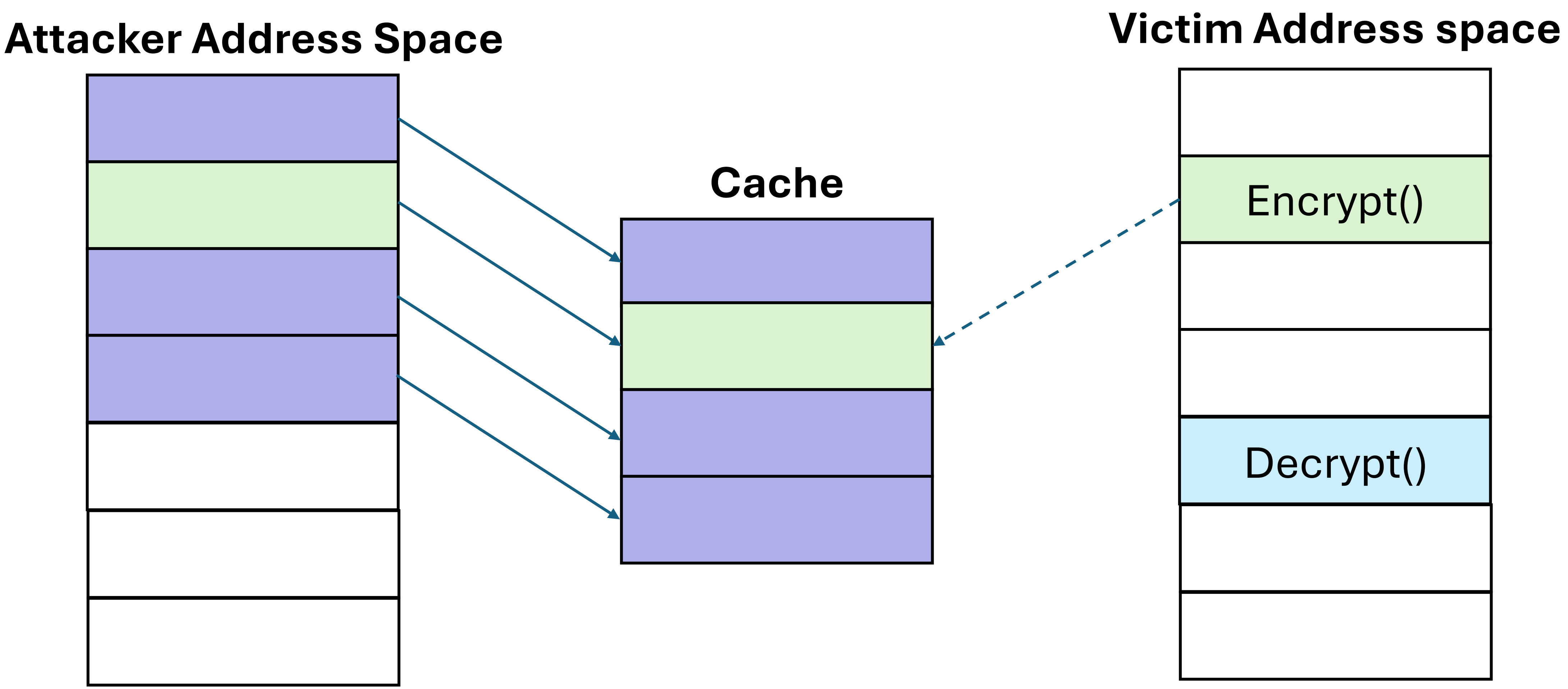
- Load Encrypt() to gain virtual address to the same physical page
- Flush the cache line corresponding to Encrypt()
- Reload by calling again, measuring the time to detect if the victim has loaded it
- Why? → Determine which code/data is in use, then attack further

Prime + Probe



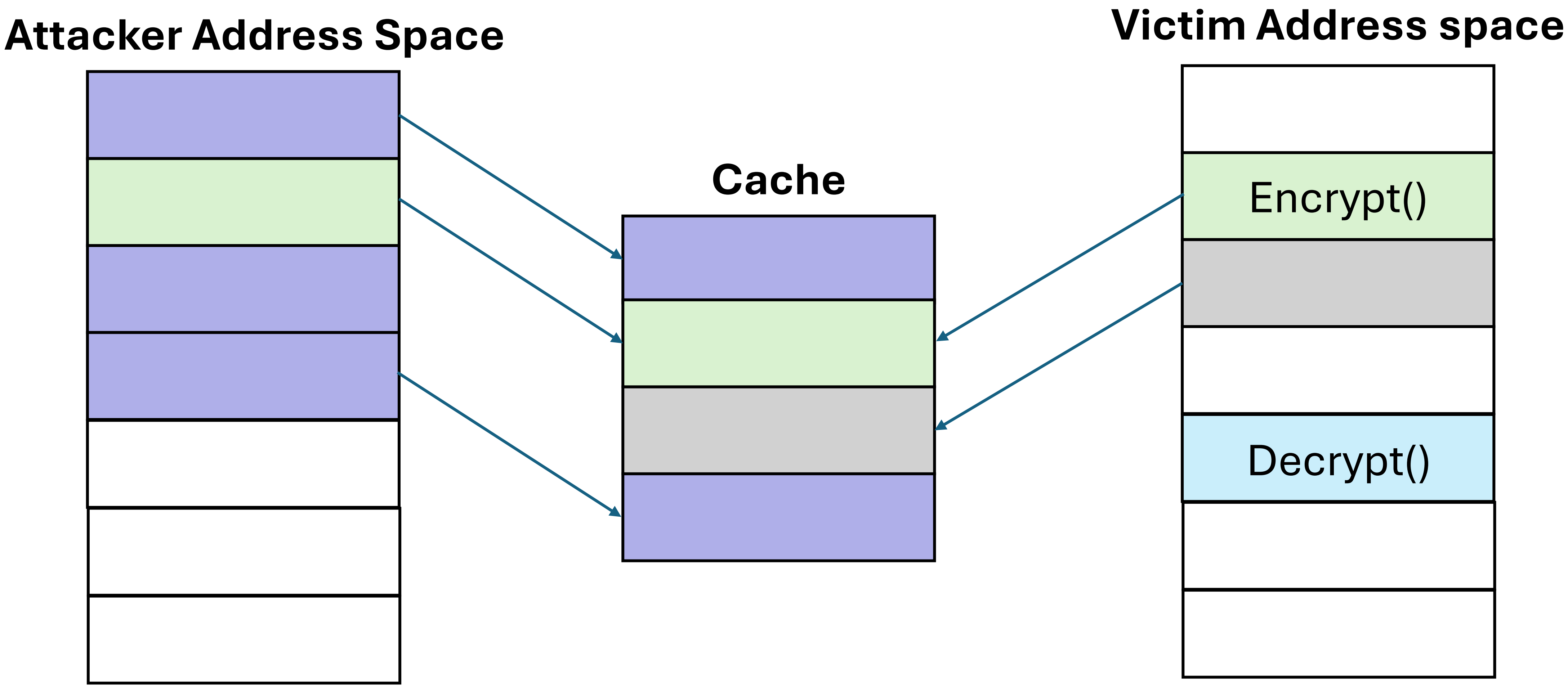
Init: victim program loaded while the cache is empty

Prime + Probe



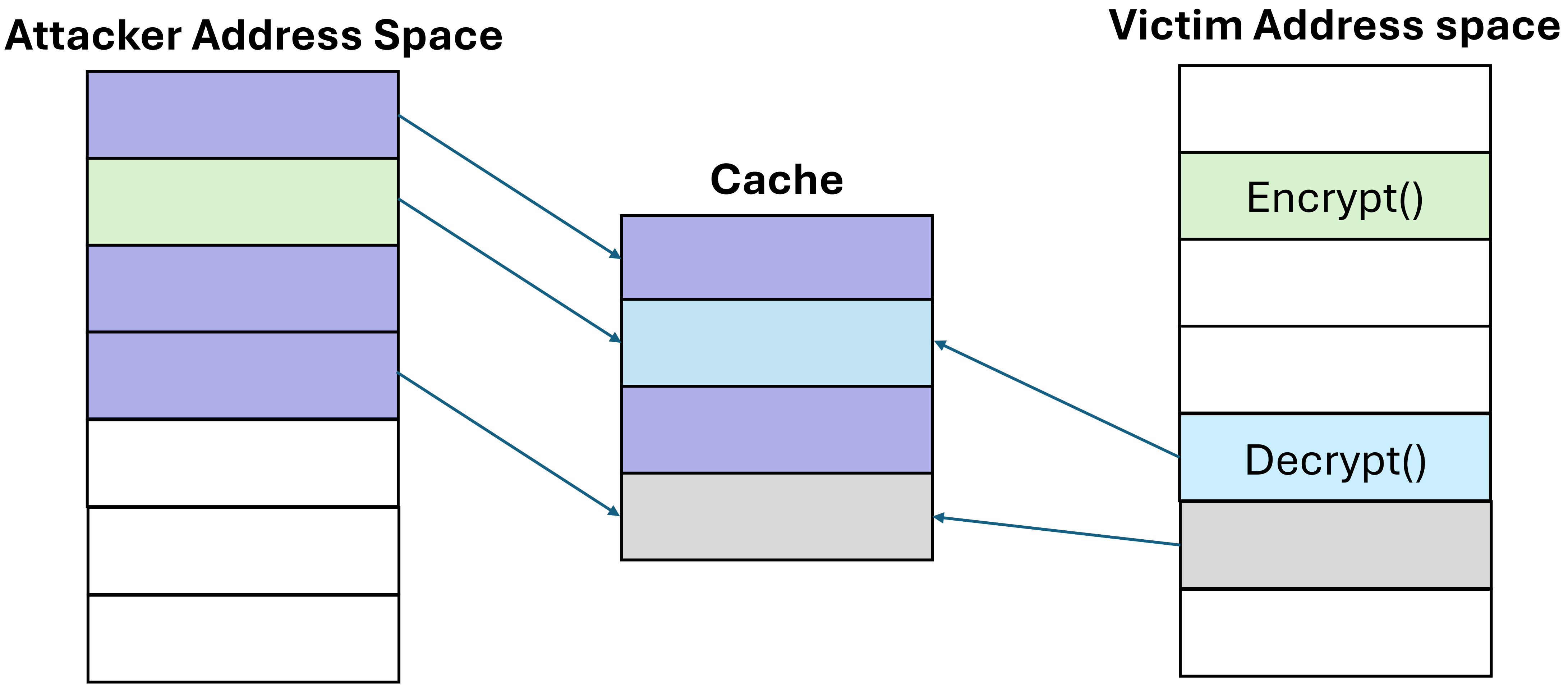
Step 1: Attacker fills all available cache (prime)

Prime + Probe



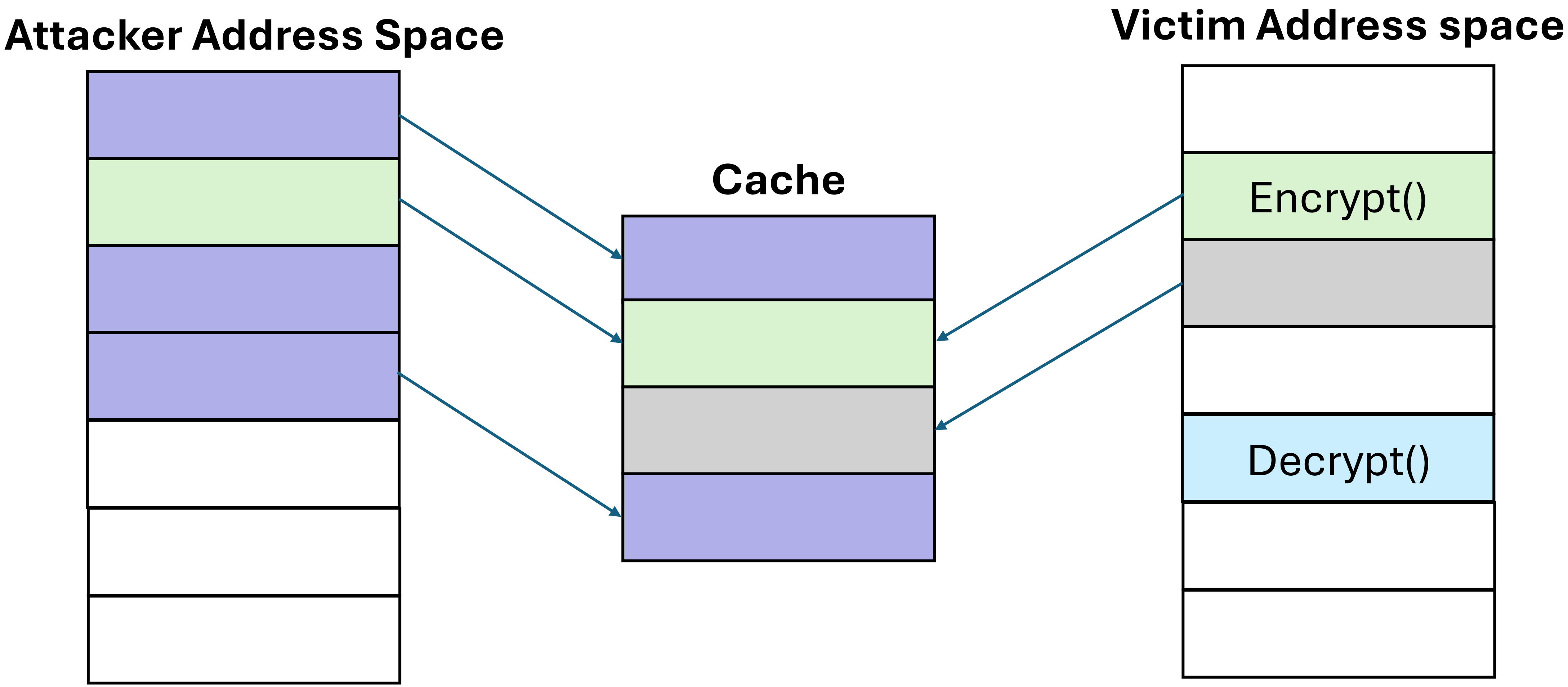
Step 2a: Victim evicts cache lines while performing Encrypt()

Prime + Probe



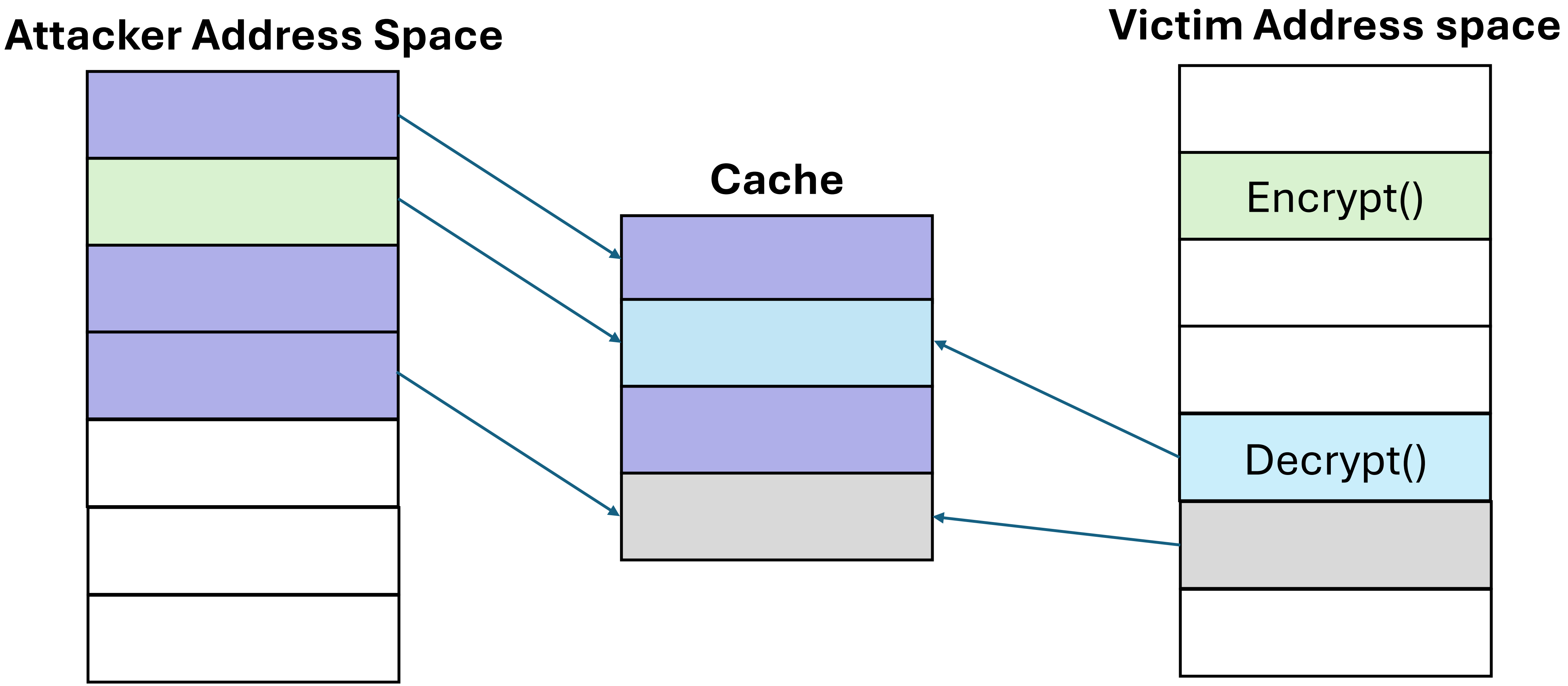
Step 2b: Victim evicts cache lines while performing Decrypt()

Prime + Probe



Step 3: attacker calls Encrypt after step 2a → fast!

Prime + Probe



Step 3: attacker calls Encrypt after step 2b → slow

Flush + Reload

Summary:

- Prime: attacker fills targeted cache sets with their own data
- Victim executes and evicts some of the attacker’s cache lines
- Attacker re-accesses their cache lines, and timing reveals victim activity
- Why? → Same as before
  - Determine which code/data is in use, then attack further

Side channels in TPMs & TEEs

So, how does this relate to HW security measures?

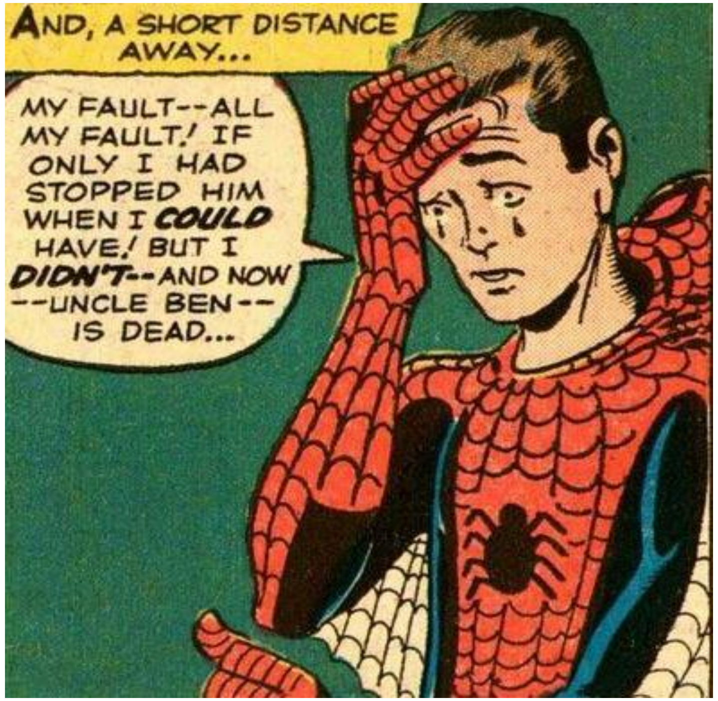
Side channels in TPMs & TEEs

So, how does this relate to HW security measures?

Don't forget the "Uncle Ben" principle of TEEs

TEE Manufacturer

TEE Programmer



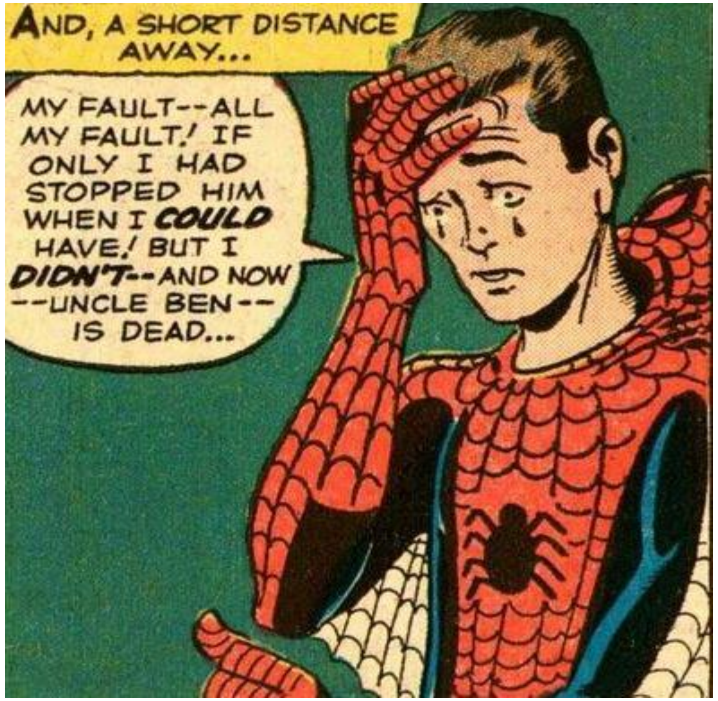
Side channels in TPMs & TEEs

So, how does this relate to HW security measures?

Don't forget the "Uncle Ben" principle of TEEs

TEE Manufacturer

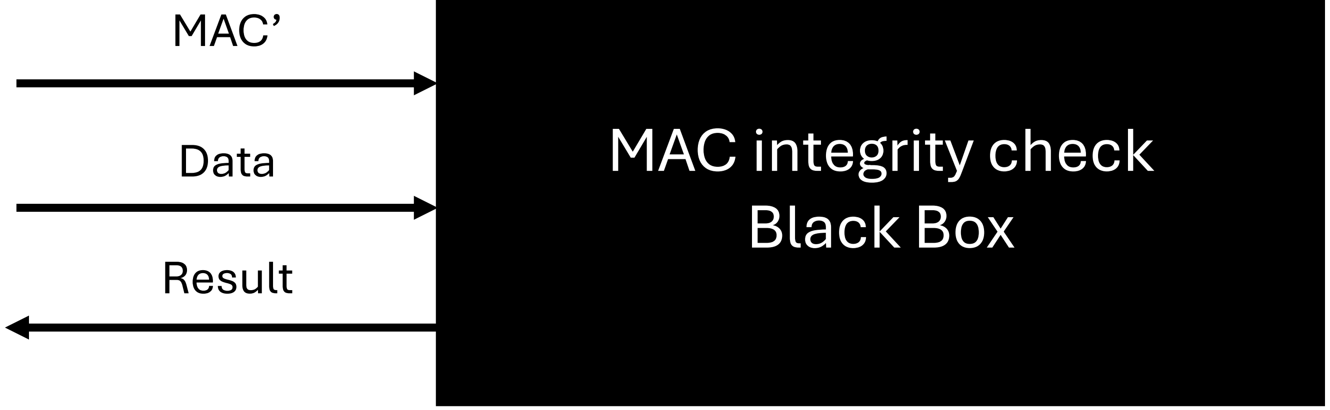
TEE Programmer



Insecure code inside TEE boundary (including software-induced side channels) break the hardware-provided guarantees

Side channels in TPMs & TEEs

Recall from earlier...

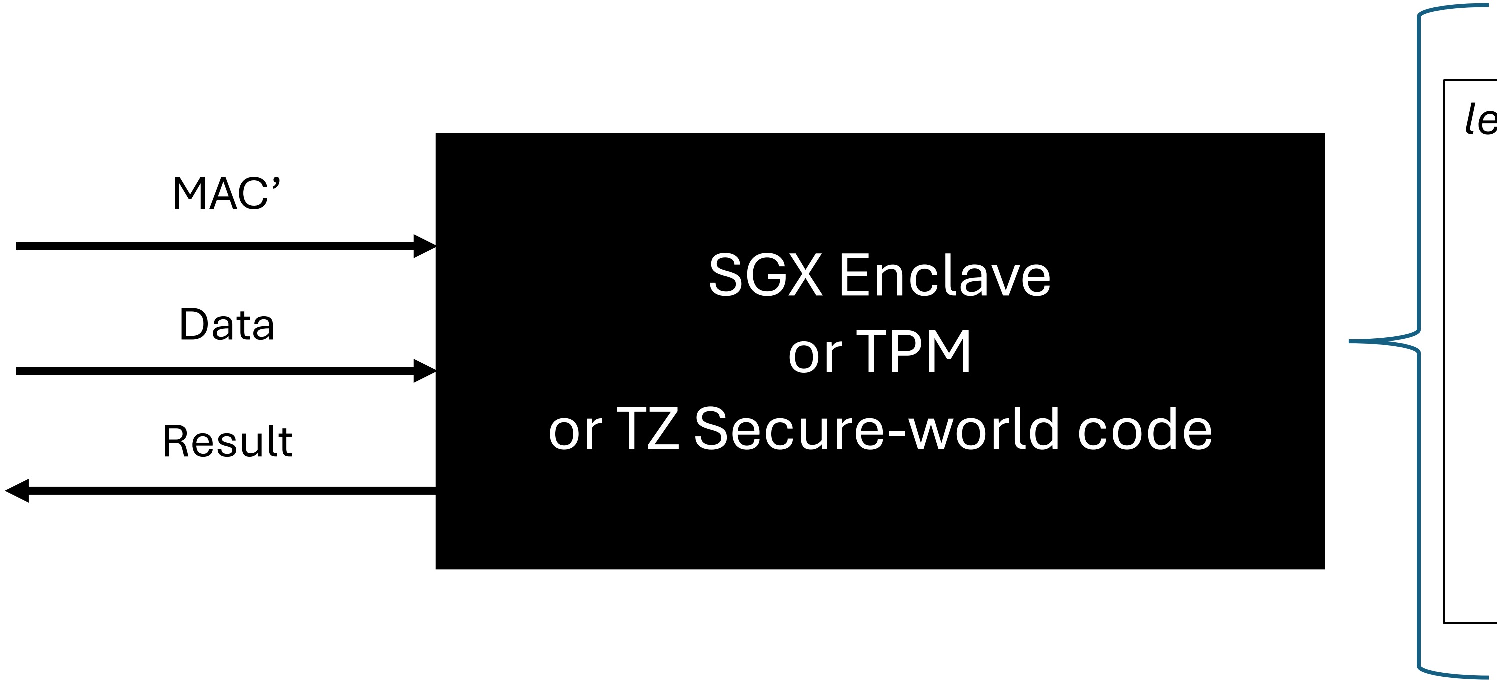


What could this black box be?.....

```
leaky_verify(MAC’, data):

  MAC_true = MAC(k, data)

  for i in range(0, len(MAC_true)):
    if MAC’[i] != MAC_true[i]:
      return False

  return True
```

Side channels in TPMs & TEEs

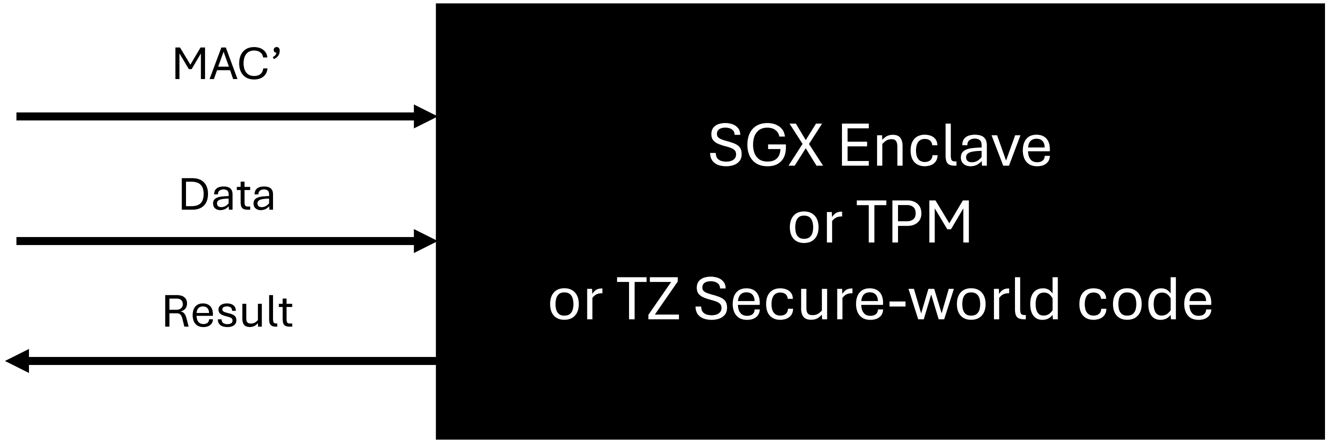
Recall from earlier…




```
leaky_verify(MAC’, data):

  MAC_true = MAC(k, data)

  for i in range(0, len(MAC_true)):
    if MAC’[i] != MAC_true[i]:
      return False

  return True
```

Side channels in TPMs & TEEs

Recall from earlier…




```
leaky_verify(MAC’, data):

  MAC_true = MAC(k, data)

  for i in range(0, len(MAC_true)):
    if MAC’[i] != MAC_true[i]:
      return False

  return True
```

Timing attacks on TPMs and TEEs are possible if not careful

Side channels in TPMs & TEEs

Example: TPM timing side channels

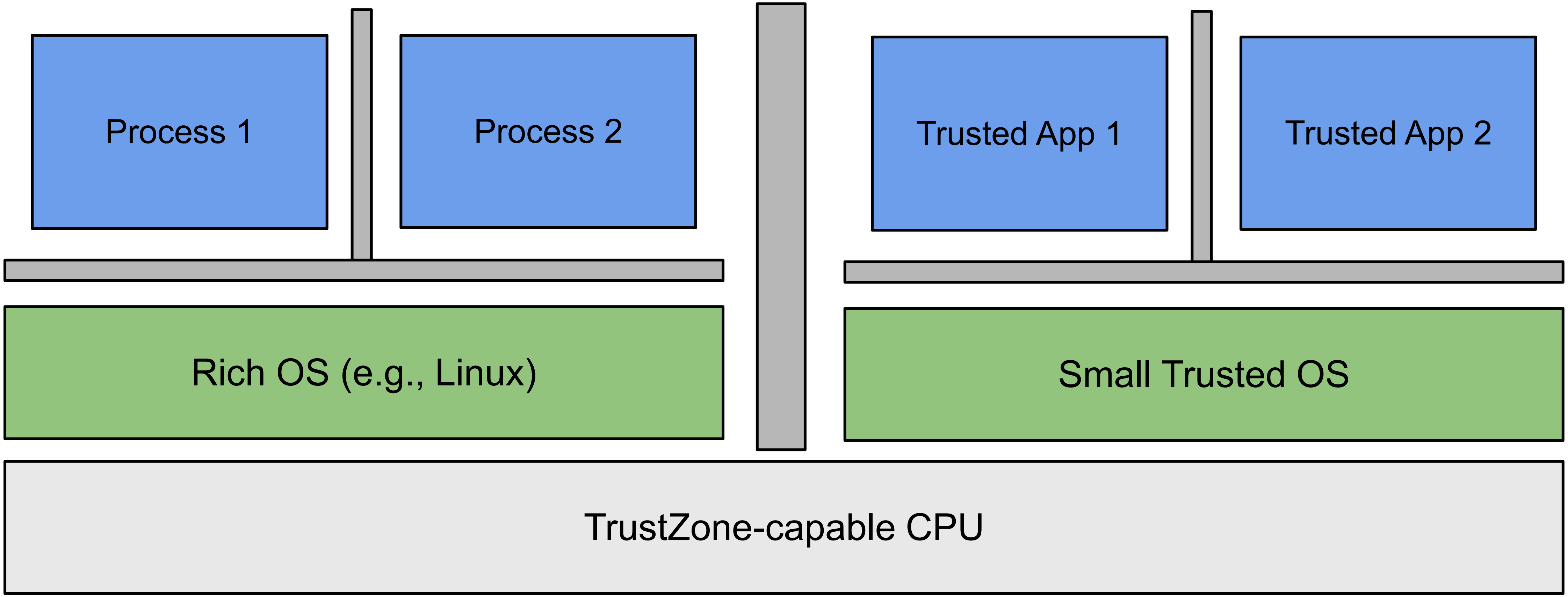
Particularly as it relates to firmware-TPMs (fTPMs)

- Software-based implementation of TPMs
- Addressing some limitations of physical TPM: low-bandwidth
- The idea → run the entire TPM functionality in software inside a TEE
- Software-virtualized TPM
- Intel fTPM

Side channels in TPMs & TEEs

Example: TPM timing side channels

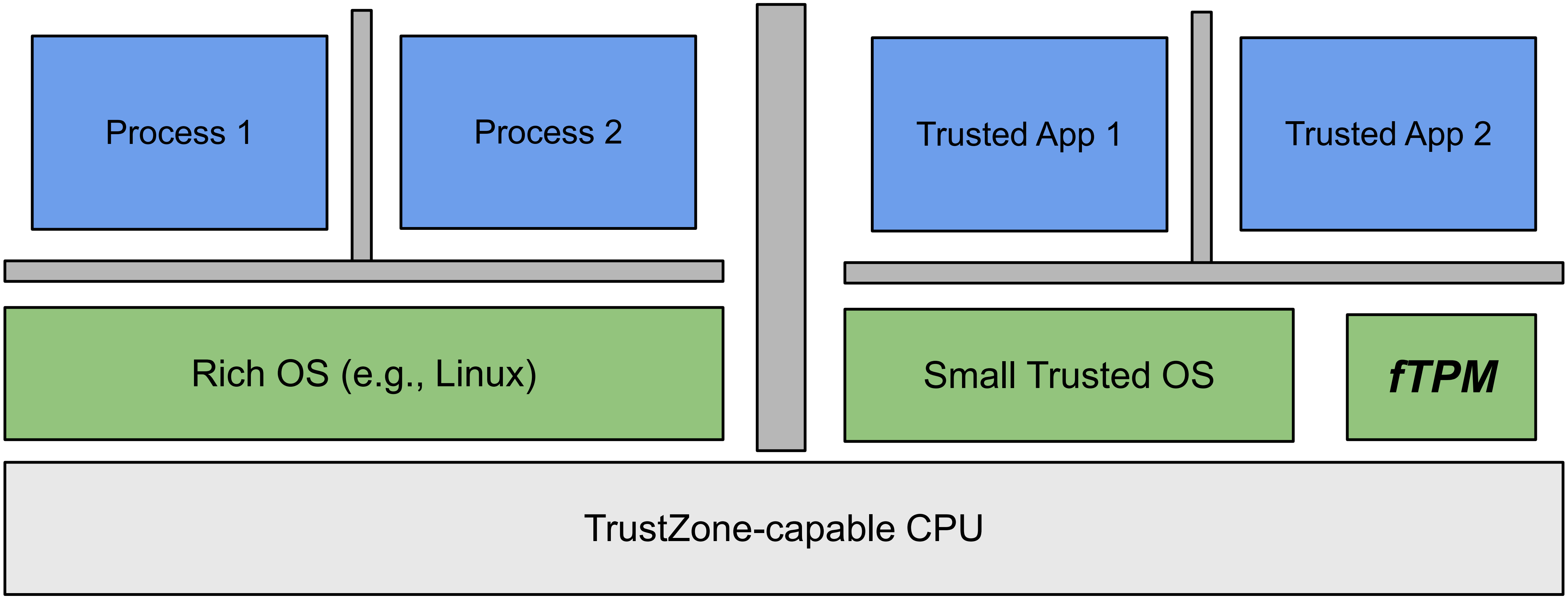
Firmware TPMs depicted:



Side channels in TPMs & TEEs

Example: TPM timing side channels

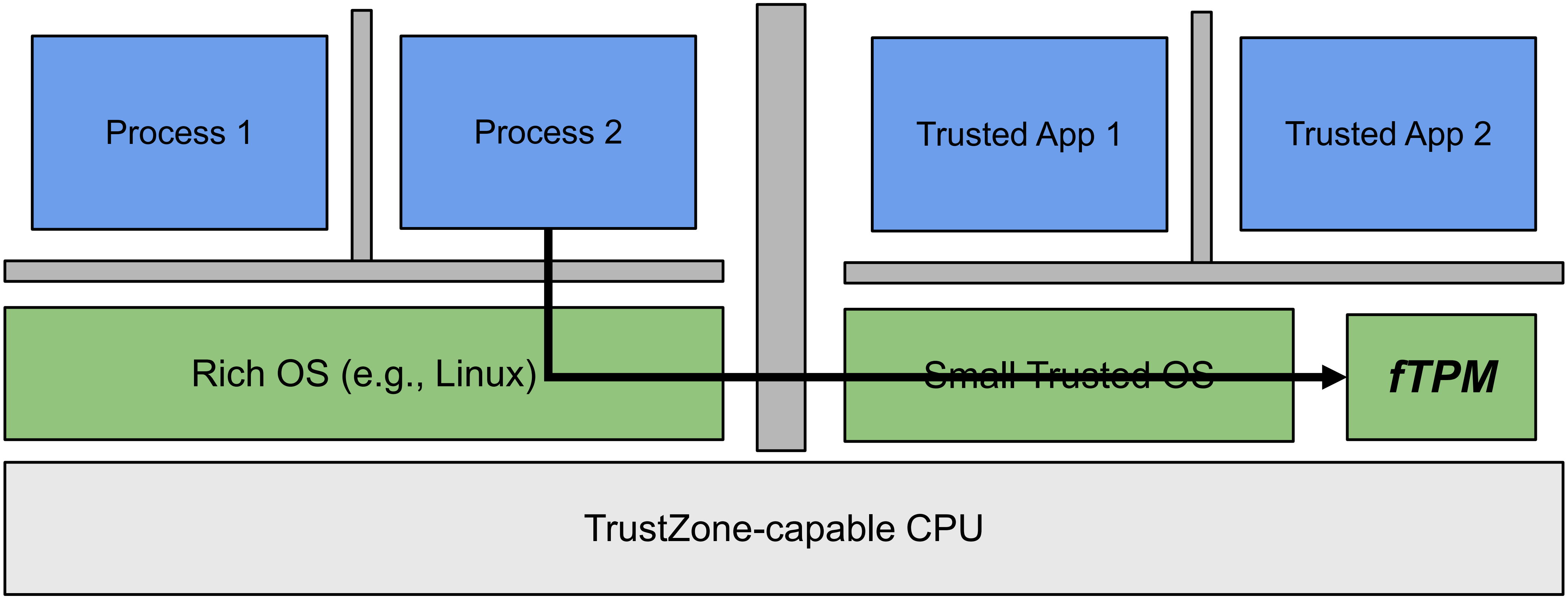
Firmware TPMs depicted:



Side channels in TPMs & TEEs

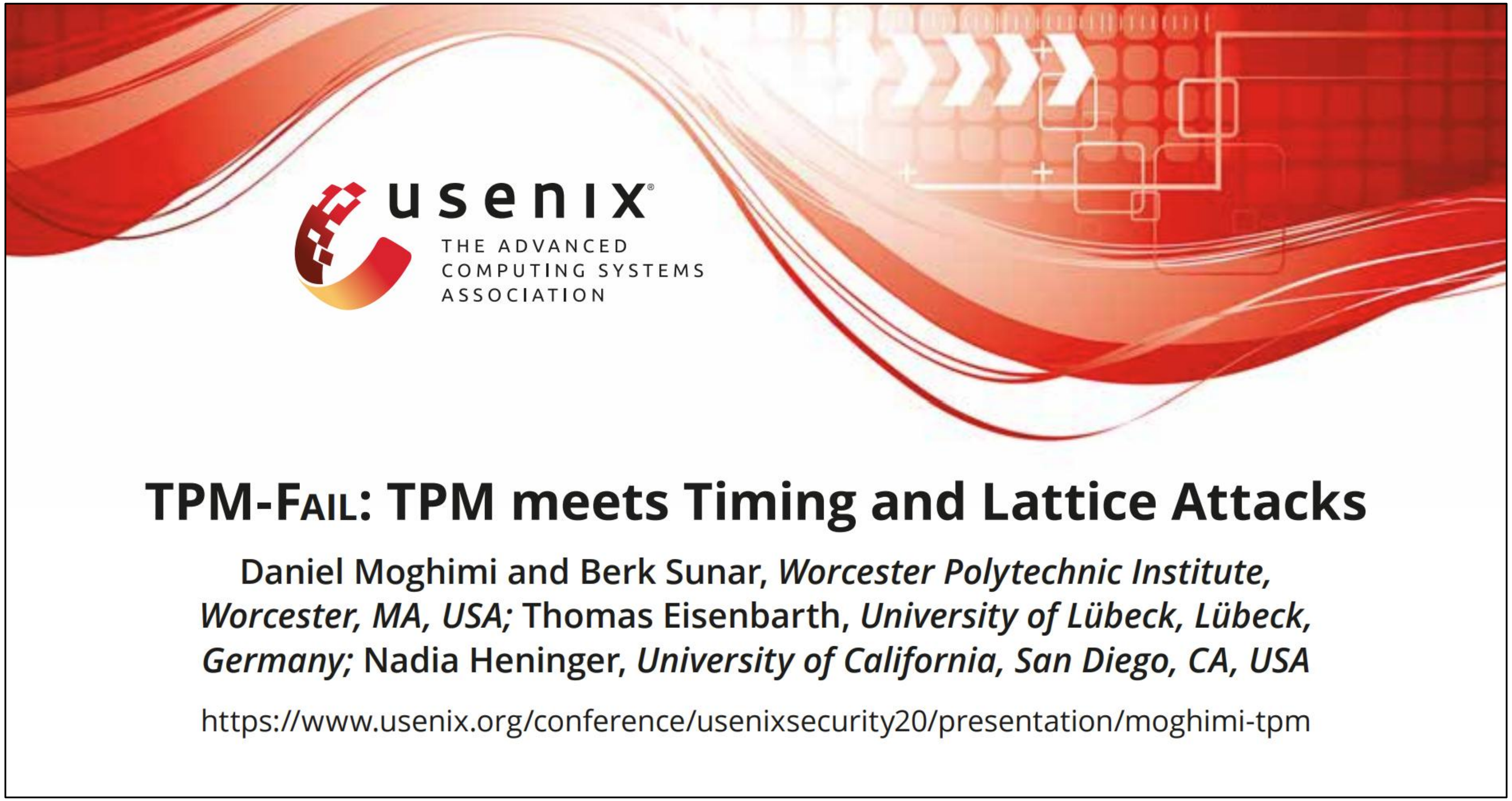
Example: TPM timing side channels

Firmware TPMs depicted:



Side channels in TPMs & TEEs

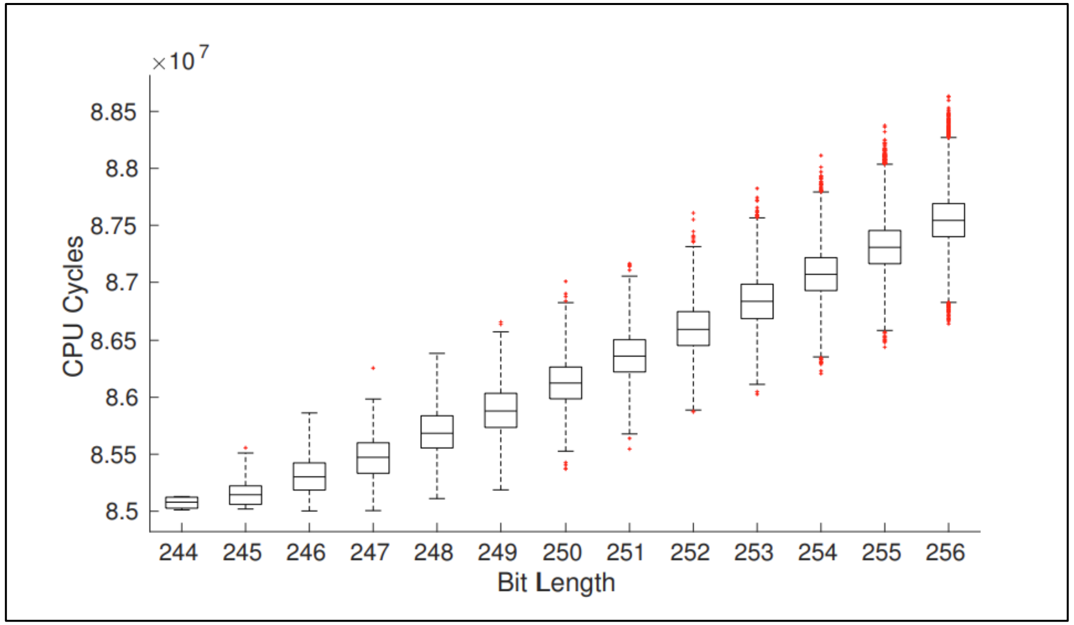
Example: TPM timing side channels



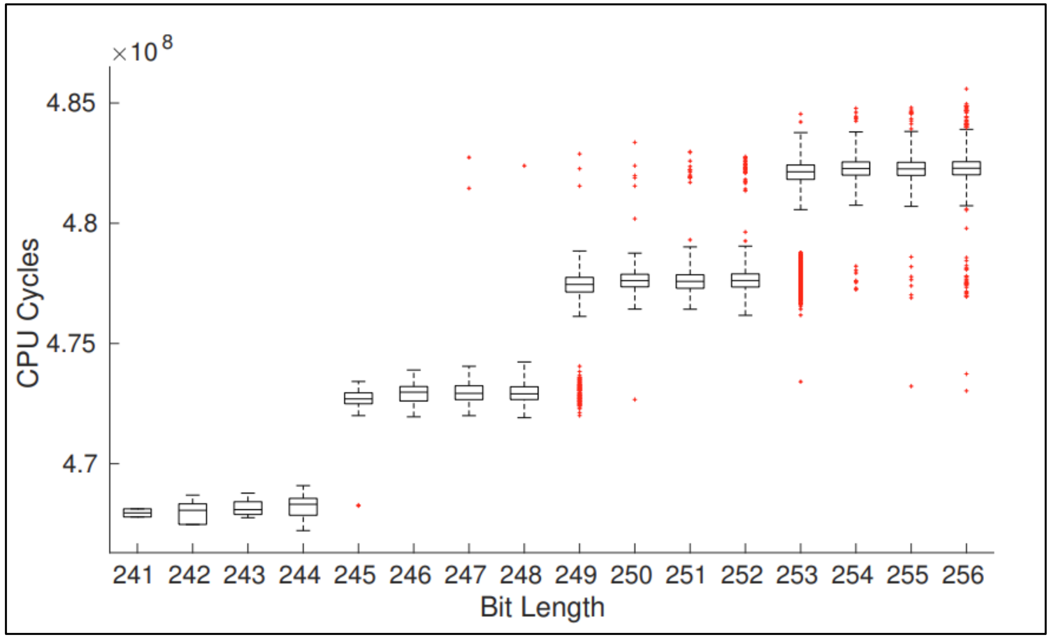
[TPM-Fail paper]

Side channels in TPMs & TEEs

Example: fTPM timing side channels in Intel and STMicroelctronic



STMicroelectronics fTPM signature generation



Intel fTPM signature generation

fTPMs were found to have timing side channels for ECDSA signature generation

- ECDSA scalar multiplication depends on nonce length

Side channels in TPMs & TEEs

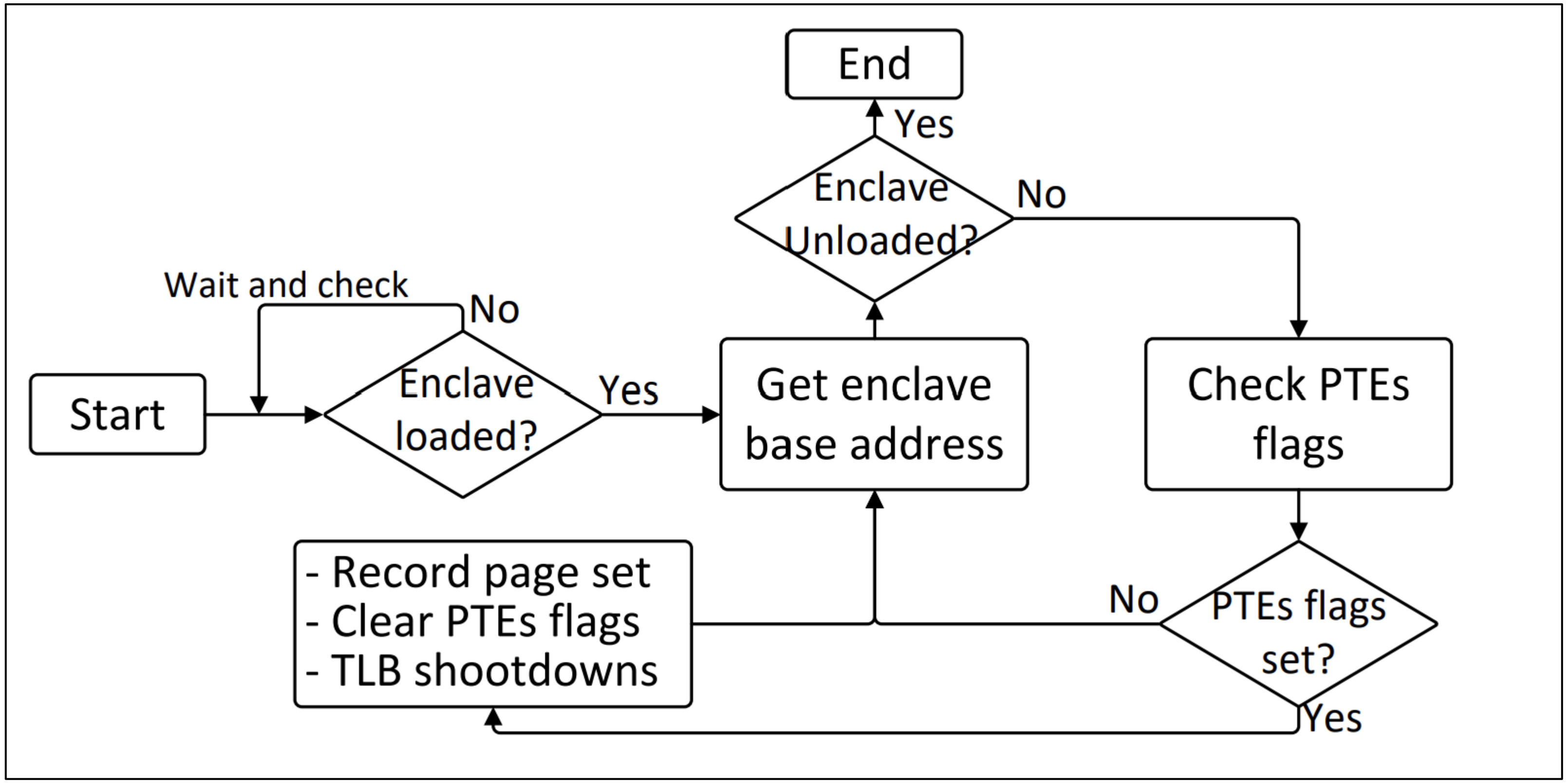
Example: Intel SGX

- Recall →
  - Enclave pages is placed in EPC
  - Metadata stored in EPCM
  - Both cannot be directly modified (only through EADD before EINIT)
- Malicious OS cannot directly modify
- **However:** entire memory hierarchy is shared
  - Enclave and non-enclave share cache
  - Enclave and non-enclave share other memory modules (DRAM module)
- Additionally:
  - Outer world can invoke exits from enclave → Asynchronous exits (AEX)
  - Pages have "accessed" and "dirty" bits observable by OS

Side channels in TPMs & TEEs

Example: Intel SGX → *sneaky page monitoring (SPM)*

Goal: exploit page faults to learn control flow of enclave



"Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX"

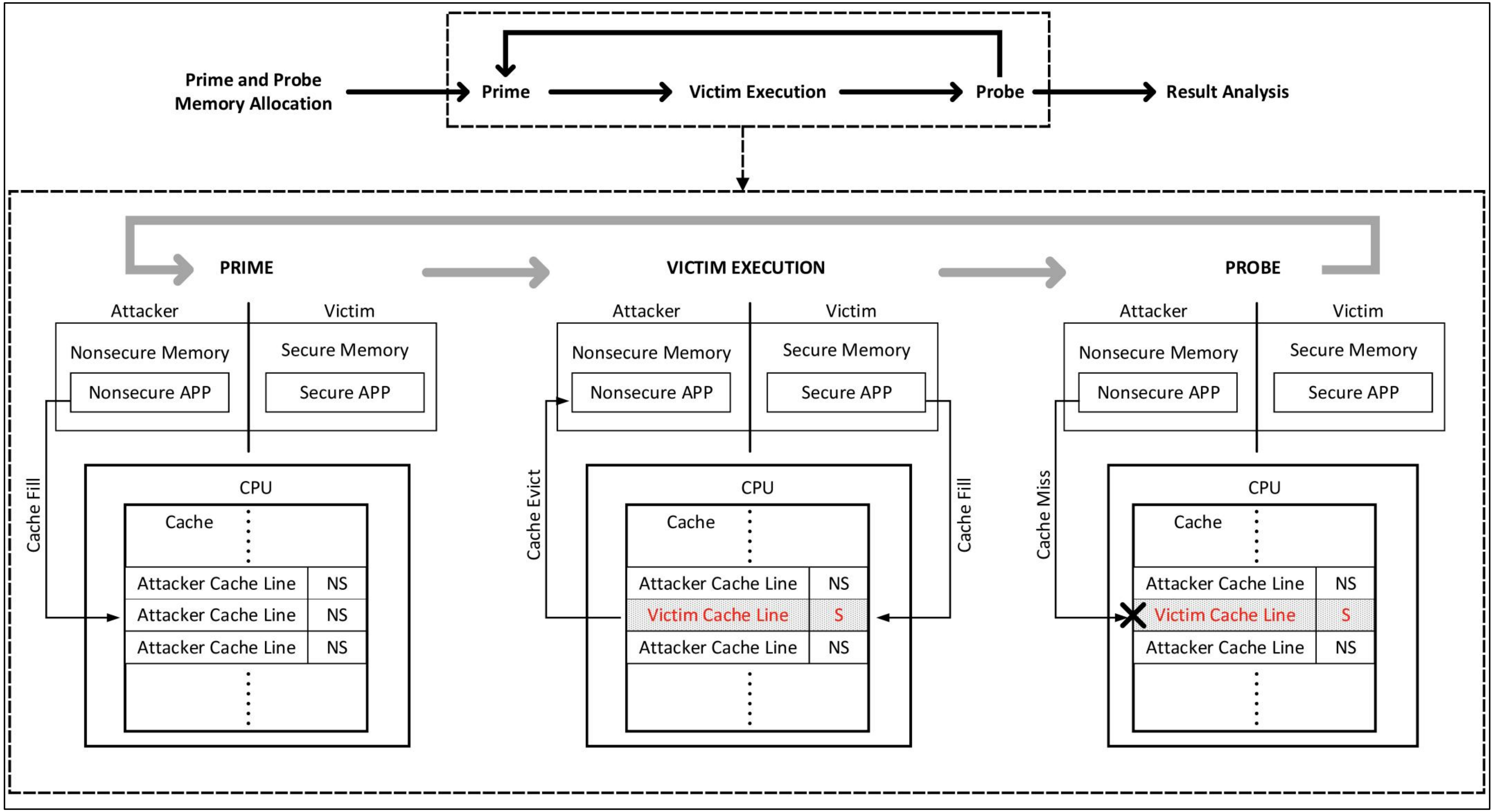
Side channels in TPMs & TEEs

Example: TrustZone

- Timing side channels
- Typical cache-timing attacks don't work in the same way
  - NS-bit in the cache
- Slightly modified version of Prime-Probe is still possible
- Also interrupt-based attacks
  - If misconfigured interrupt controller, can invoke interrupts to return to Normal World

Side channels in TPMs & TEEs

Example: TrustZone version of Prime+Probe



Summary

Many sources of side-channels must be considered

Some require physical access, others are possible to observe remotely

- Constant-time programming
- Some tools to automate, but mostly done manually
- Some ISA support → e.g., conditional instructions in ARM

For fTPMs and TEEs:

- For timing side channels: Uncle Ben's principle
- For other side channels: understand architectural behavior

That’s all for today!

Coming up….

- Ethics, law, regulations, and compliance

Reminders:

- A4 is due on July 25

That's all for today!

Resources:

- Leaky Cauldron
- Another SGX attack: SGX Step
- Load-step attack in TrustZone
- TruSpy cache attack in TrustZone
- TPM-Fail