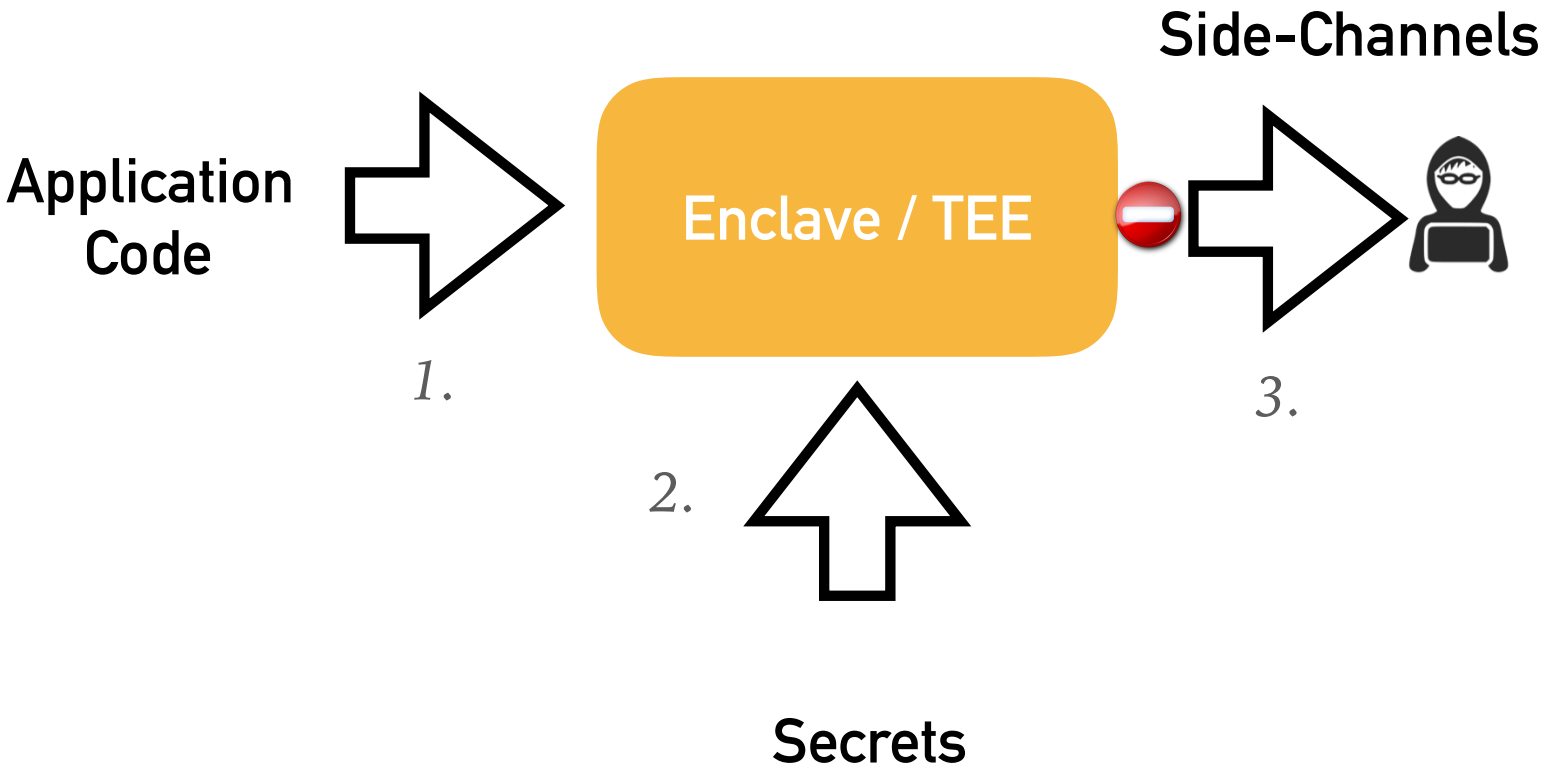
APPLICATION-ORIENTED SECURITY: SECRETS MANAGEMENT AND SIDE-CHANNEL PROTECTION FOR TEES

Christof Fetzer
TU Dresden

TECHNISCHE UNIVERSITÄT DRESDEN

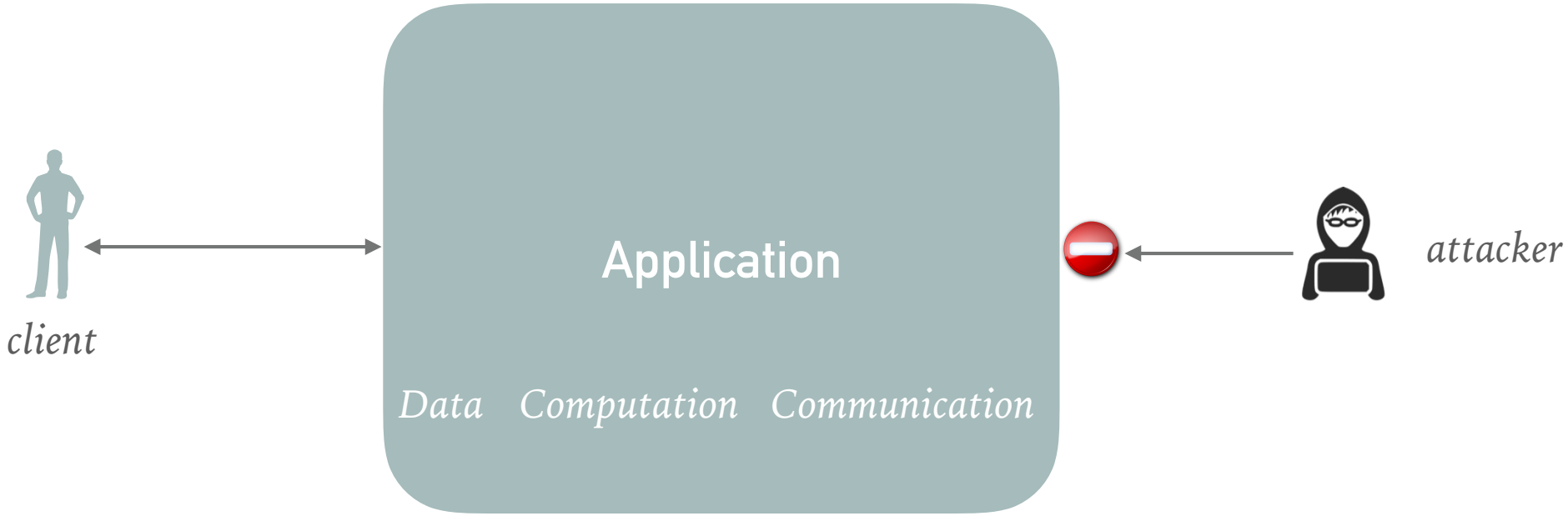
SecureCloud

OUTLINE



OBJECTIVES

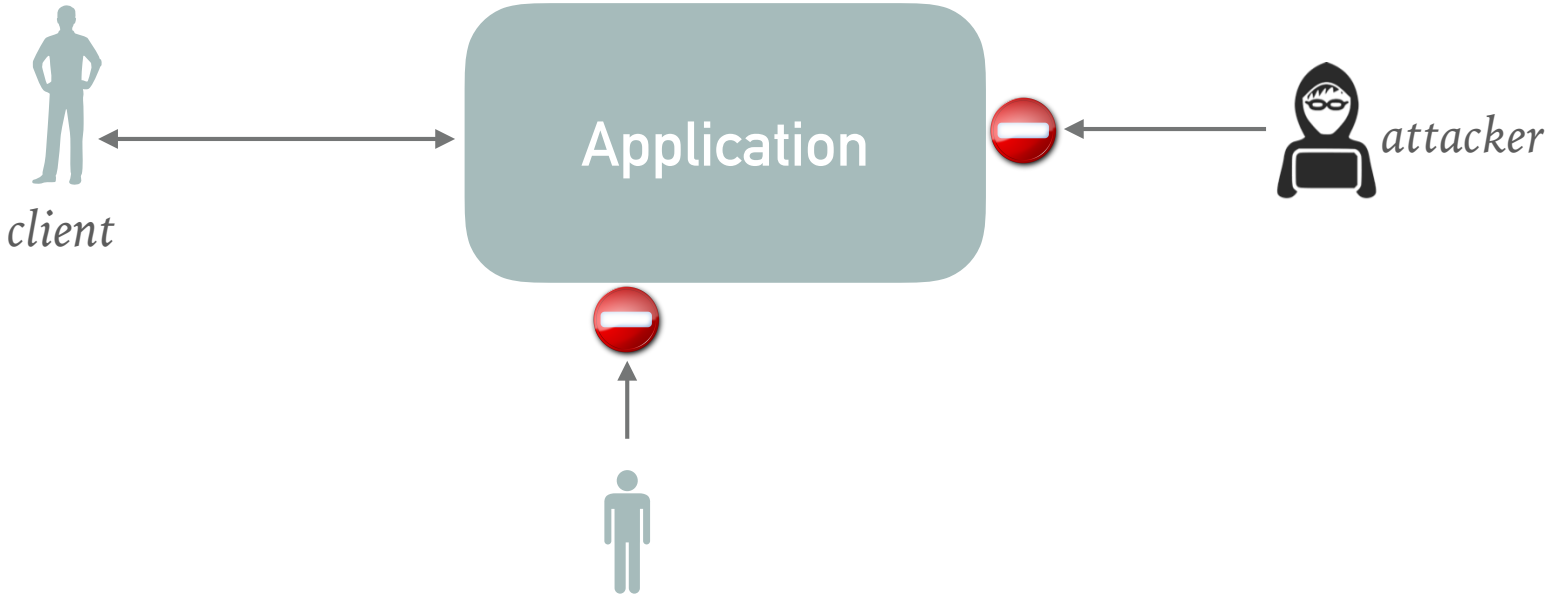
APPLICATION-ORIENTED SECURITY



***Objective**: Ensure integrity and confidentiality of applications*

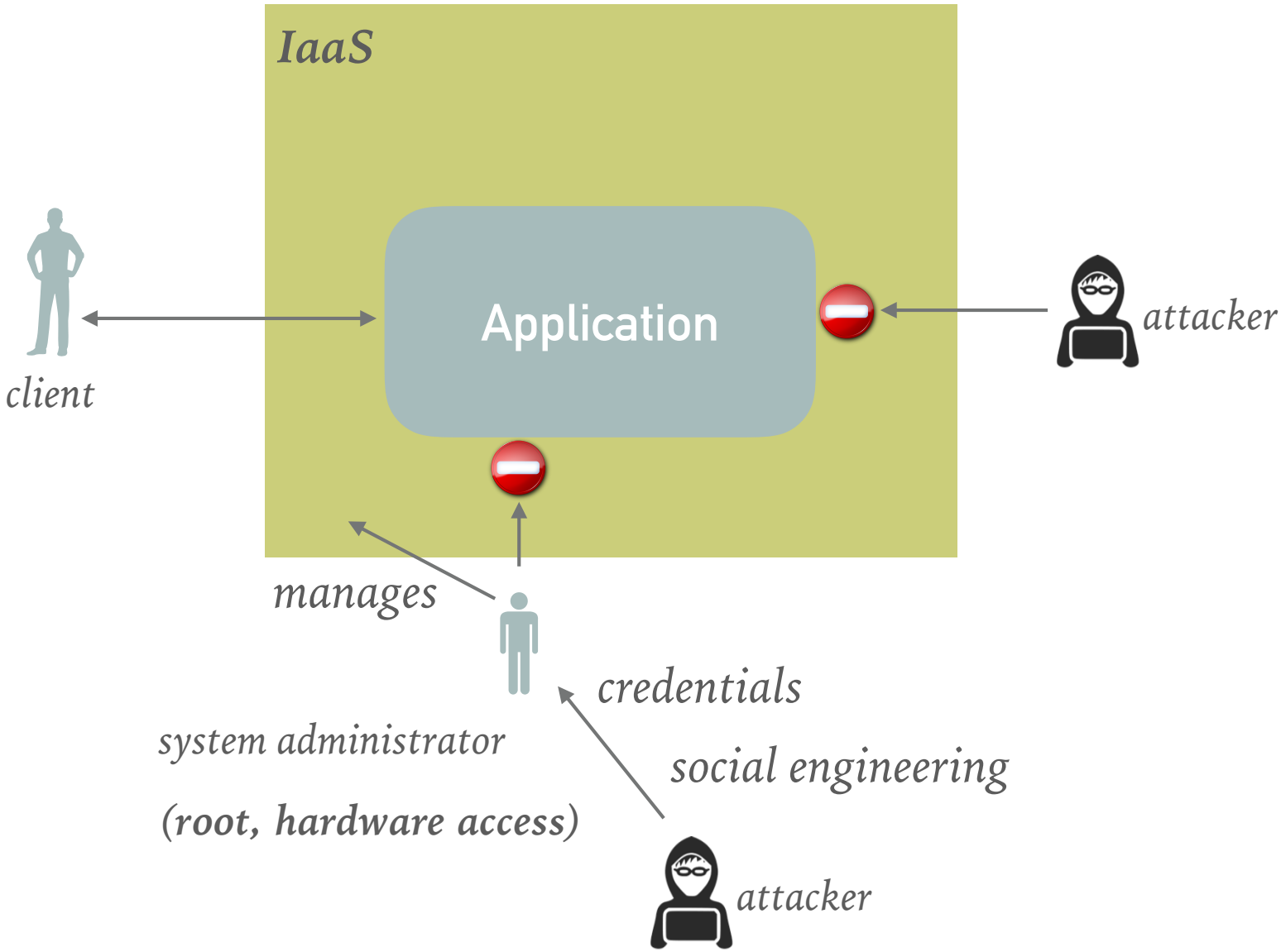
https://sconedocs.github.io

THREAT MODEL



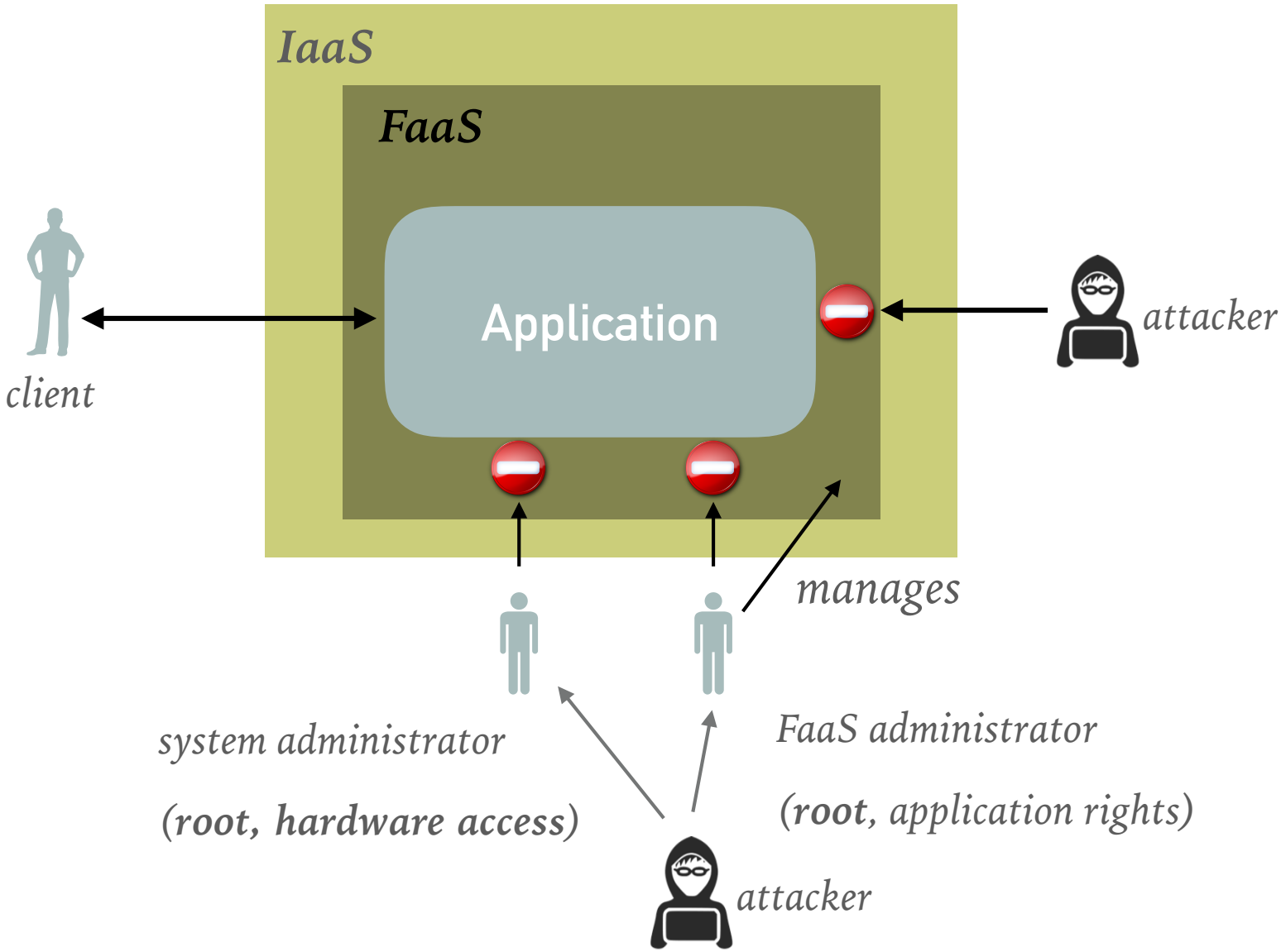
https://sconedocs.github.io

SYSTEM ADMINISTRATOR



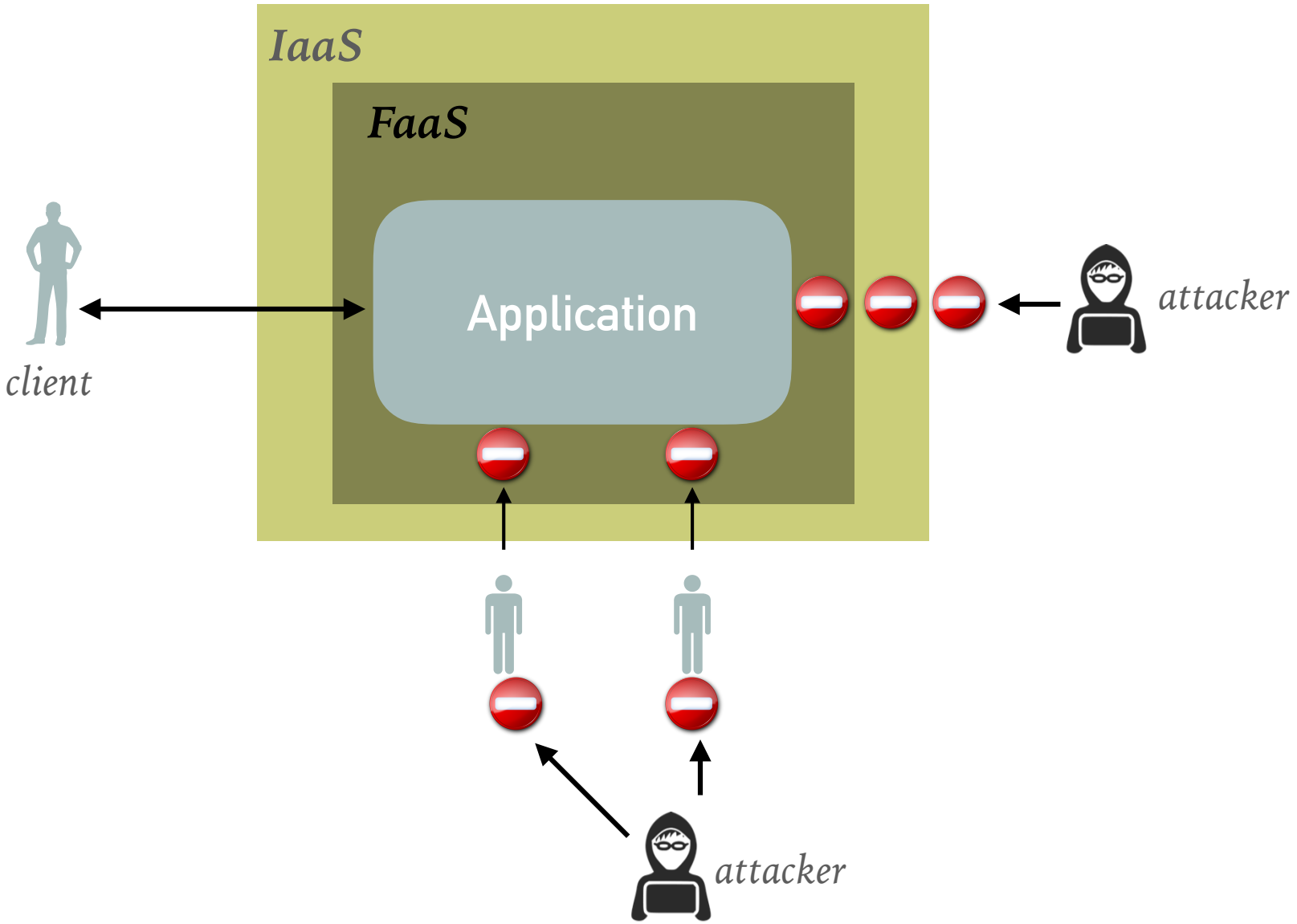
https://sconedocs.github.io

SERVICE PROVIDER ADMINISTRATOR



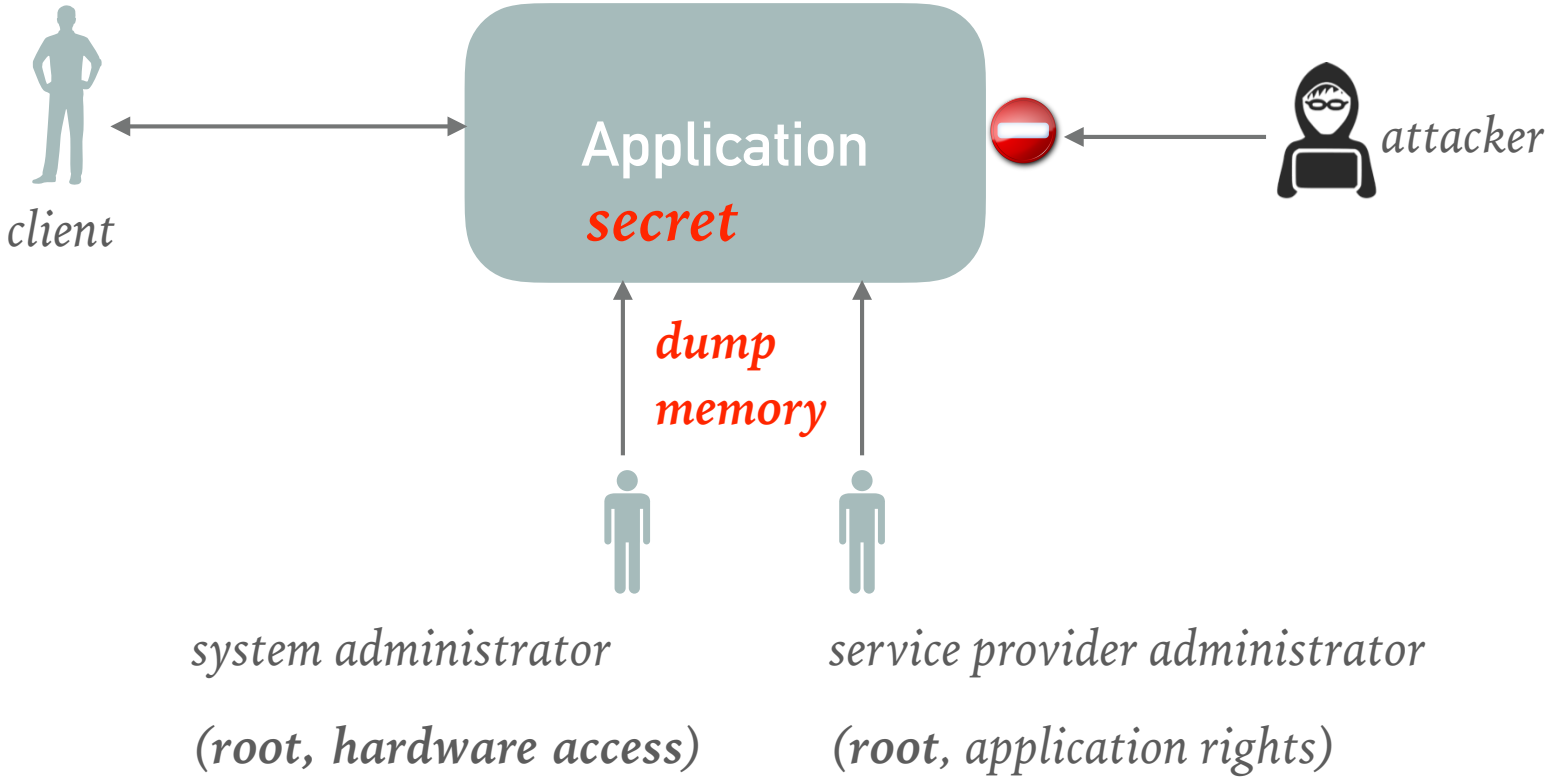
https://sconedocs.github.io

DEFENSE IN DEPTH!



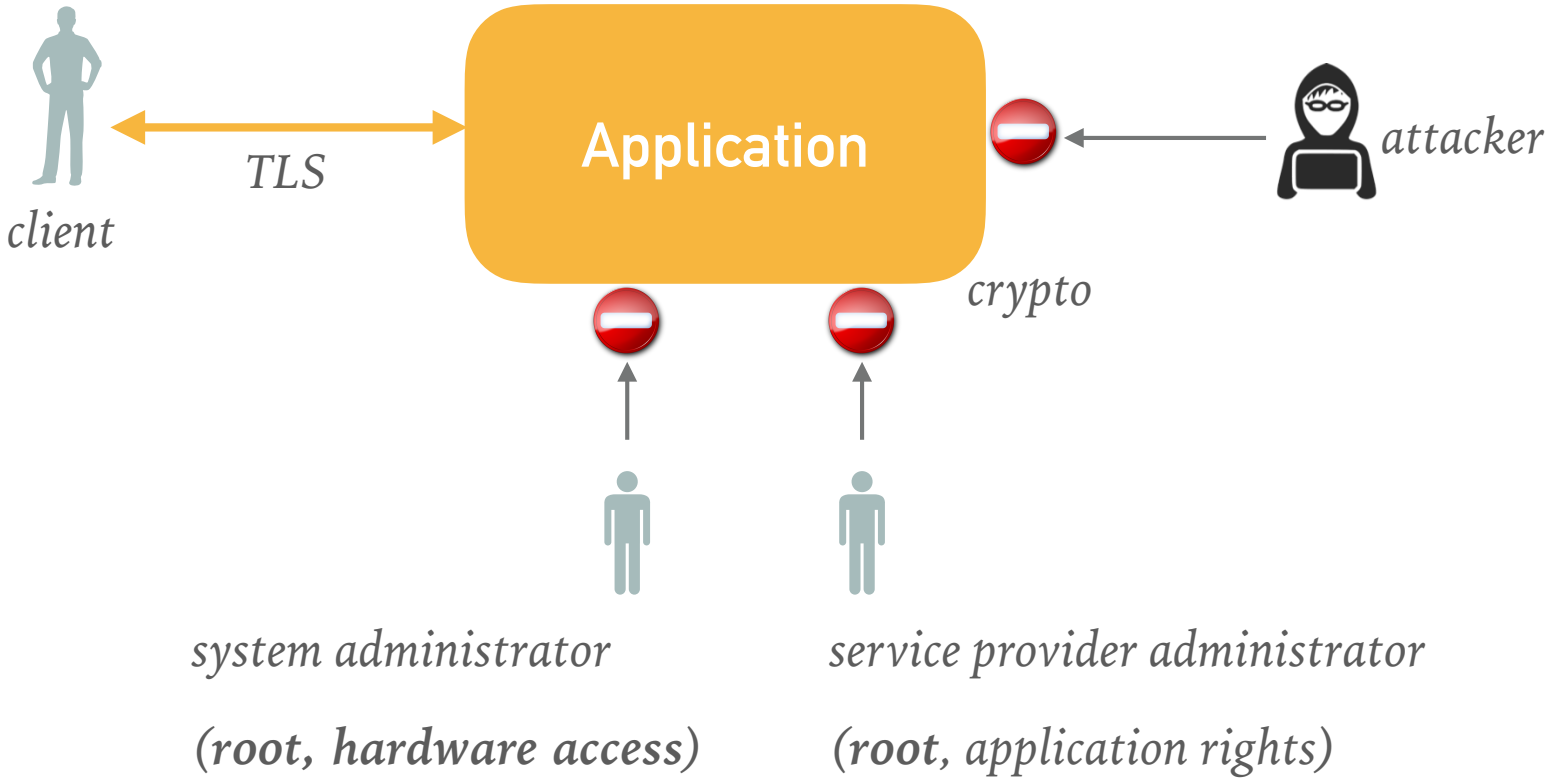
https://sconedocs.github.io

OS-BASED ACCESS CONTROL INSUFFICIENT



https://sconedocs.github.io

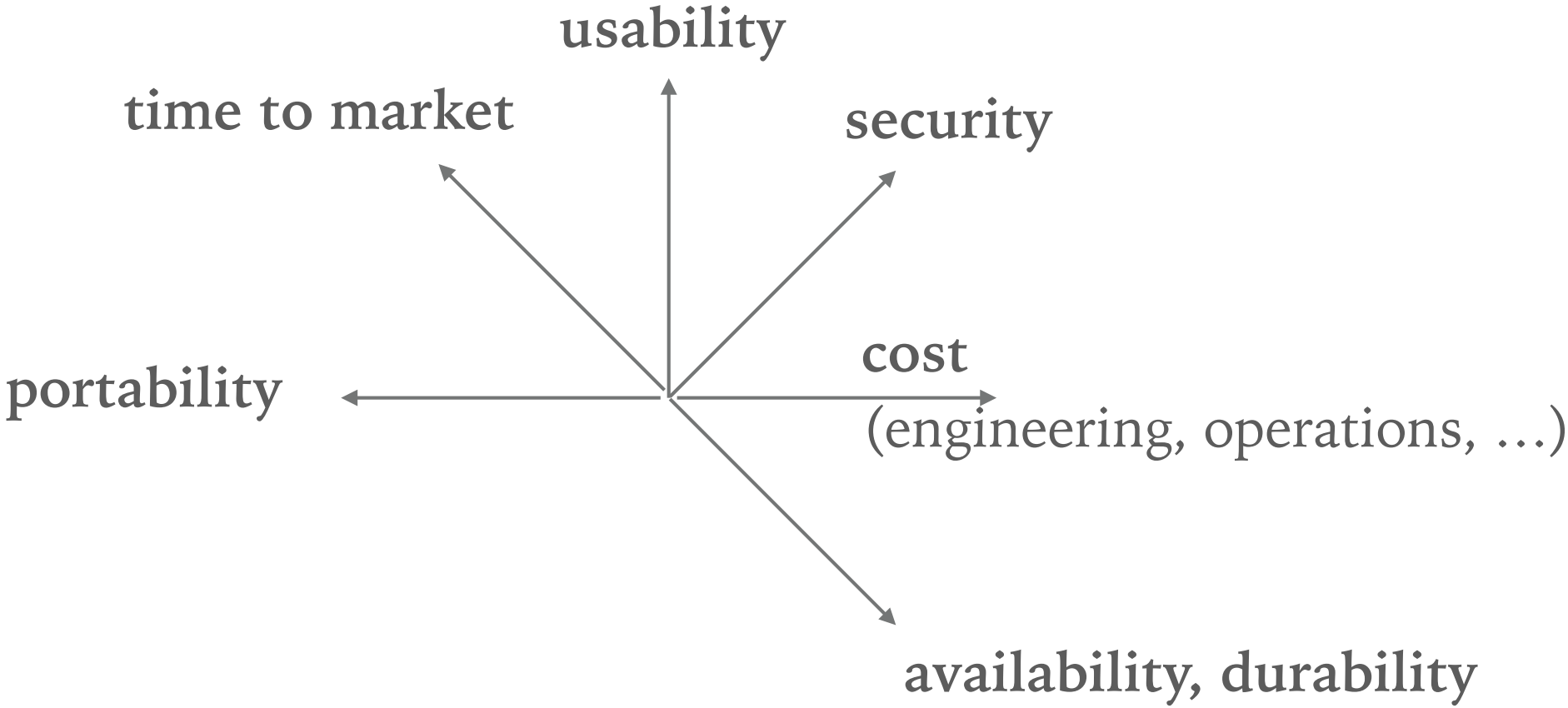
WE NEED A CRYPTOGRAPHIC APPROACH!



https://sconedocs.github.io

ENGINEERING IS ABOUT . . .

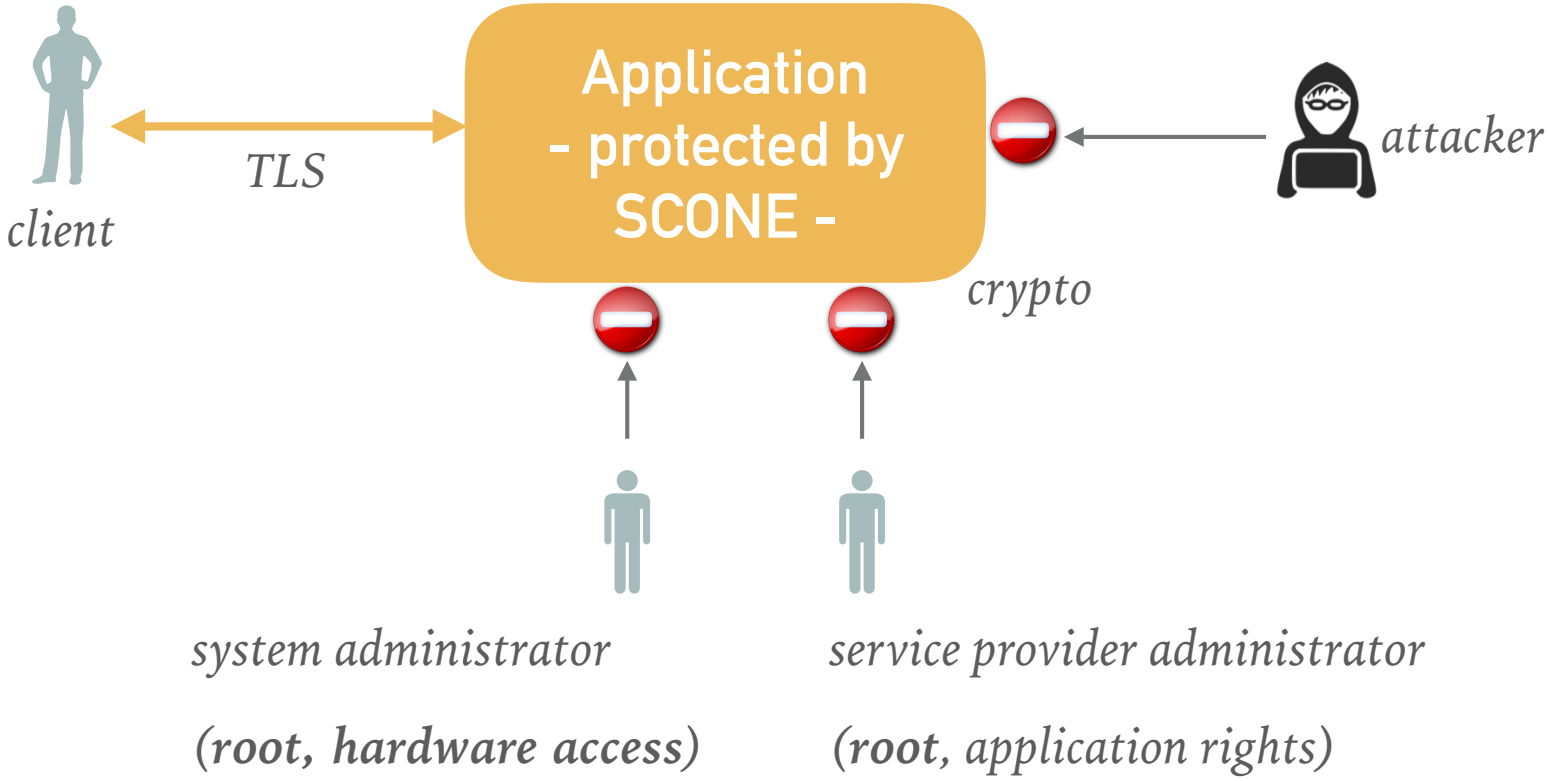
- . . . balancing multiple, often conflicting goals



. . . security is for many service providers not the most important issue

SCONE: E2E ENCRYPTION WITHOUT SOURCE CODE CHANGES

Languages: C, C++, Go, Rust, Java, Python, R, …



[SCONE] Sergei Arnautov, et al, „SCONE: Secure Linux Containers with Intel SGX“, USENIX OSDI 2016

WHY NO SOURCE CODE CHANGES?

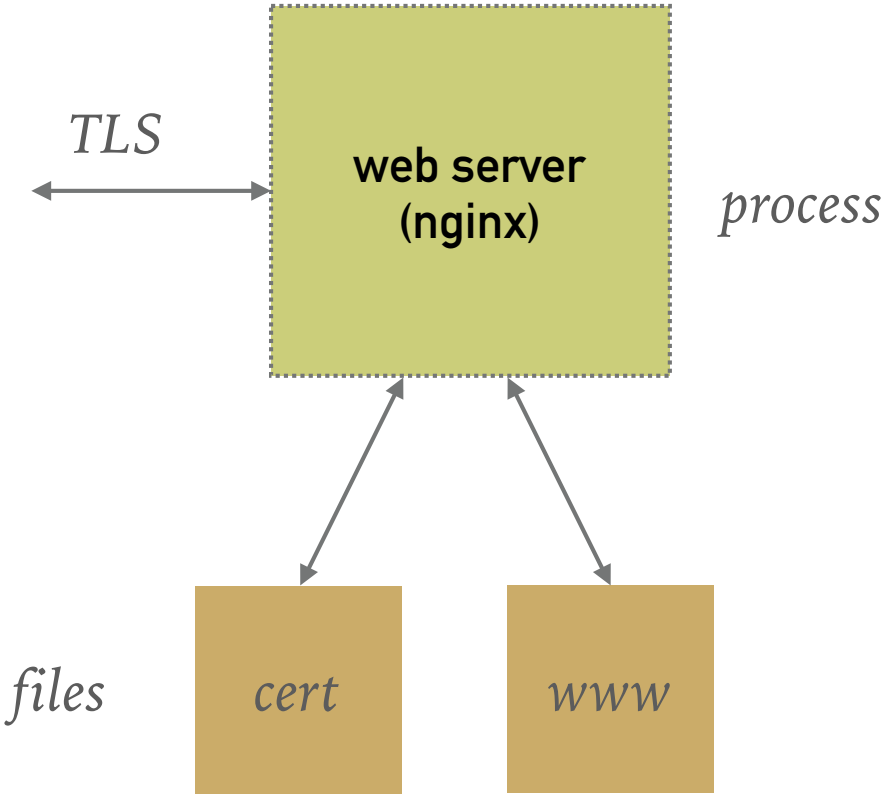
- **Considerations:**
  - reduce cost / time of improving security
  - reduce the skills required to improve security
  - no hardware / software lock-in
  - ..
  - **partitioning software for TEEs is quite difficult**

TOOL SUPPORT

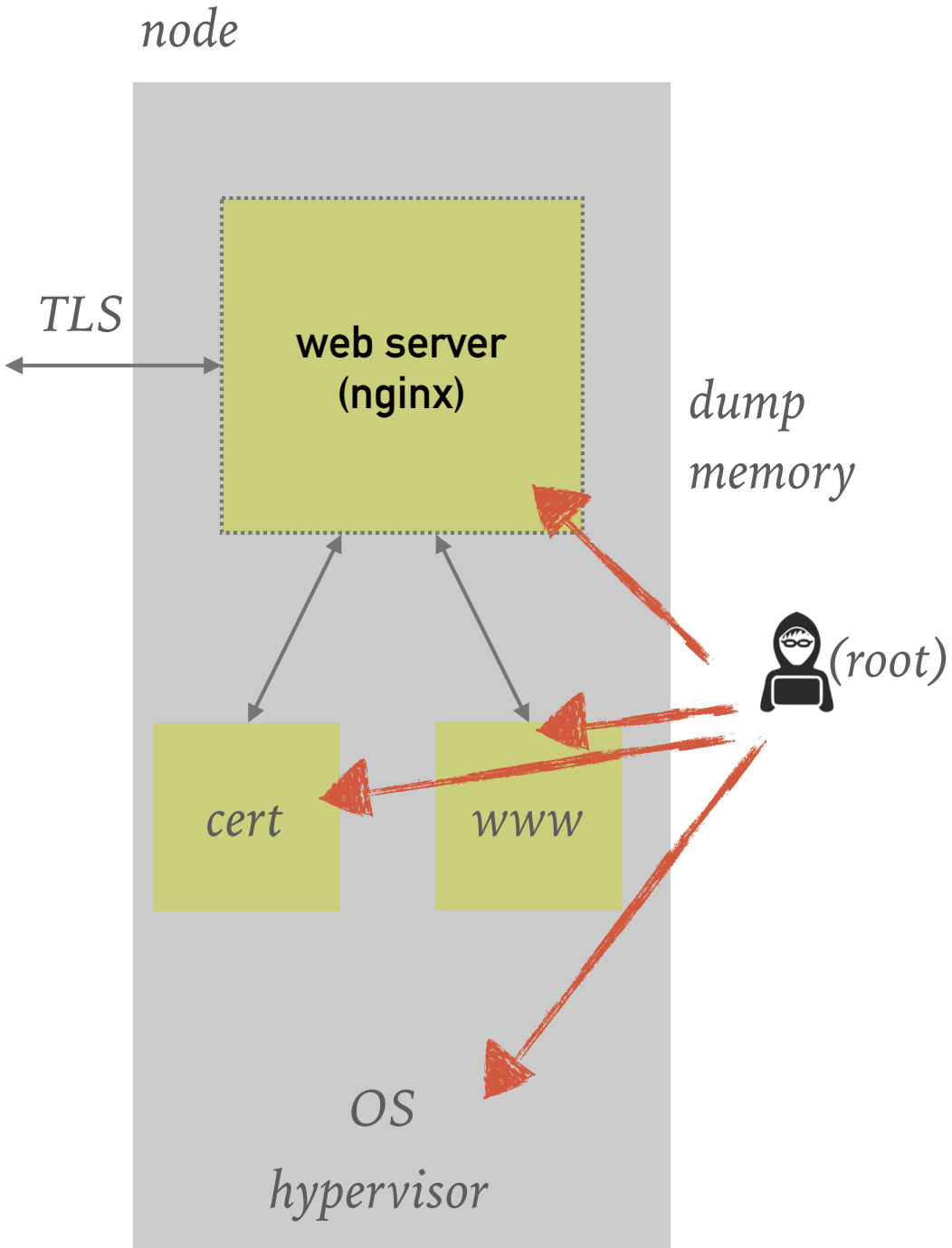
Joshua Lind, etc: „Glamdring: Automatic Application Partitioning for Intel SGX“, Usenix ATC 2017

…still hard to partition - should we really partition?

EXAMPLE



- Web Server (**nginx**)
- **Configuration:**
  - TLS certificate (private key)
  - config file
  - …
- **WWW files:**
  - must only be visible to authorised clients

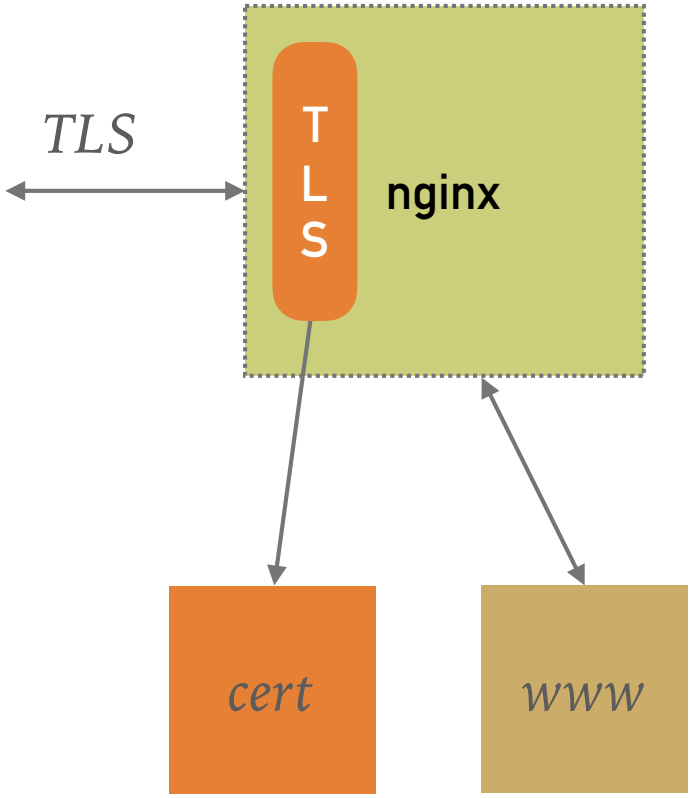


THREAT MODEL?

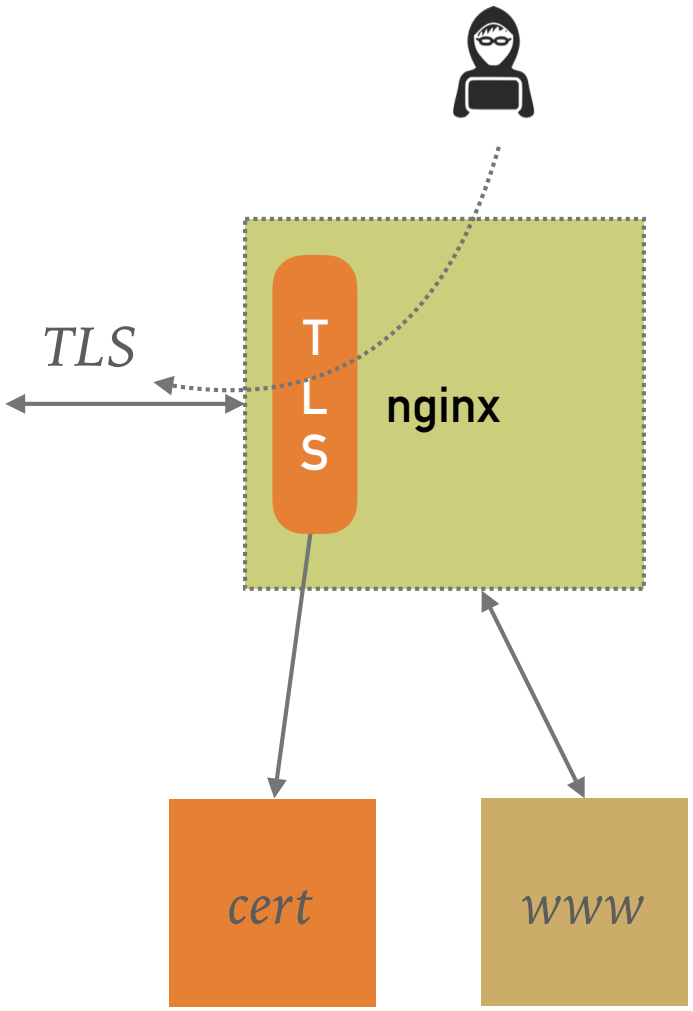
- Attacker has root access
  - controls OS
  - controls Hypervisor
- **Attacker can**
  - read/modify all files
  - can read/modify memory of processes
  - can see all network traffic

HOW TO PARTITION NGINX?

- We need to protect certificate!
  - must not leak private key
  - TLS should be protected!



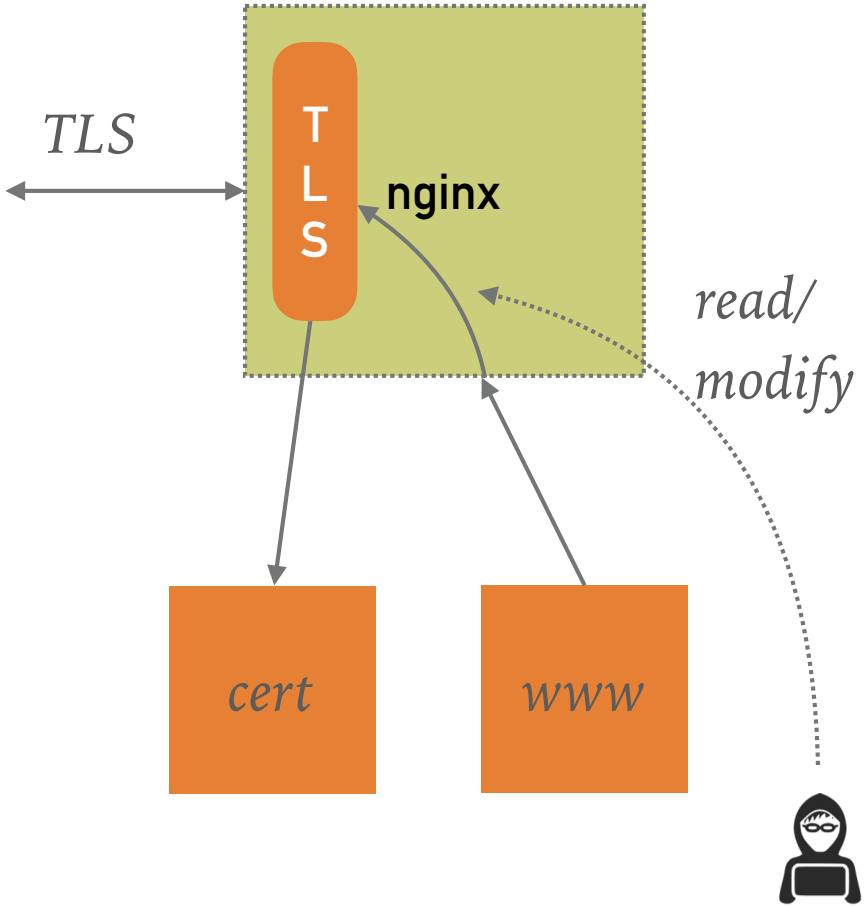
could impersonate original website if not protected



SHOULD WE PARTITION NGINX?

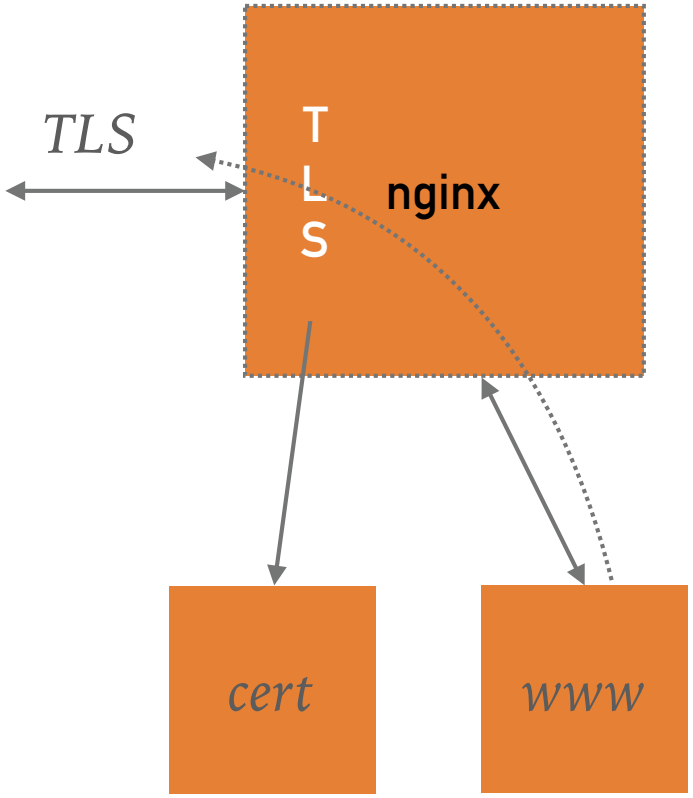
- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- **Attacker does not need cert:**
  - **establish connections via protected TLS stack**
- how to protect against this?
  - how to automate the protection?

SHOULD WE PARTITION NGINX?



- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- We need to encrypt www files
  - to ensure confidentiality
  - to ensure integrity

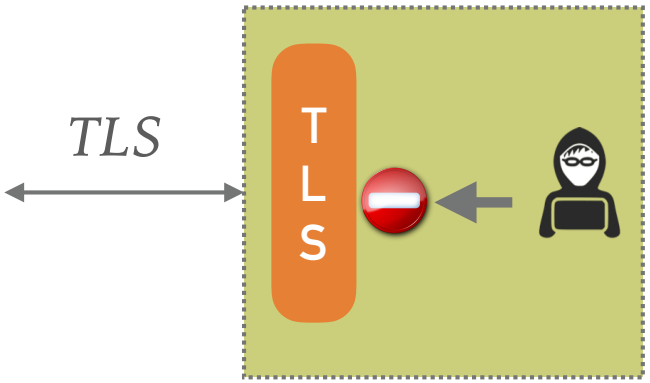
SHOULD WE PARTITION NGINX?



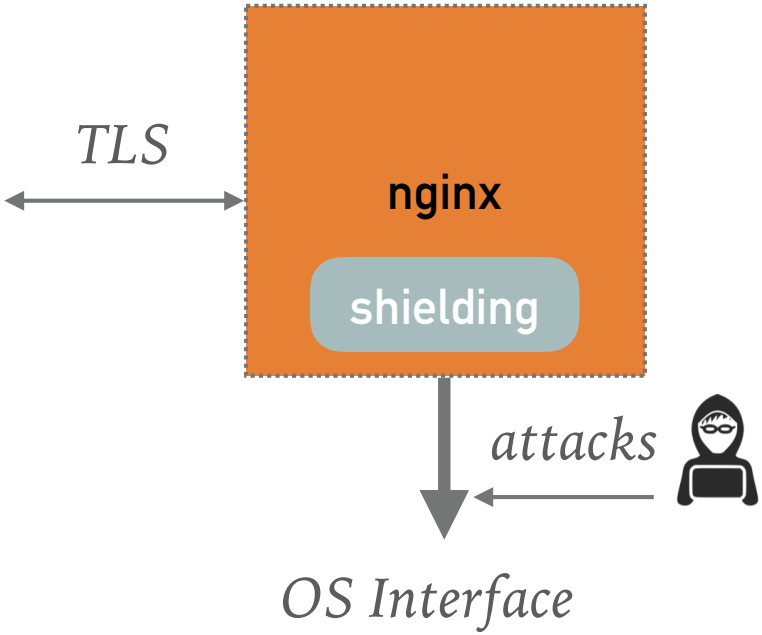
- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- We need to encrypt www files
  - to ensure confidentiality
  - to ensure integrity
- We need to protect content
  - never as plain text
  - detect modifications

Side-Channel attack!

SHIELDING

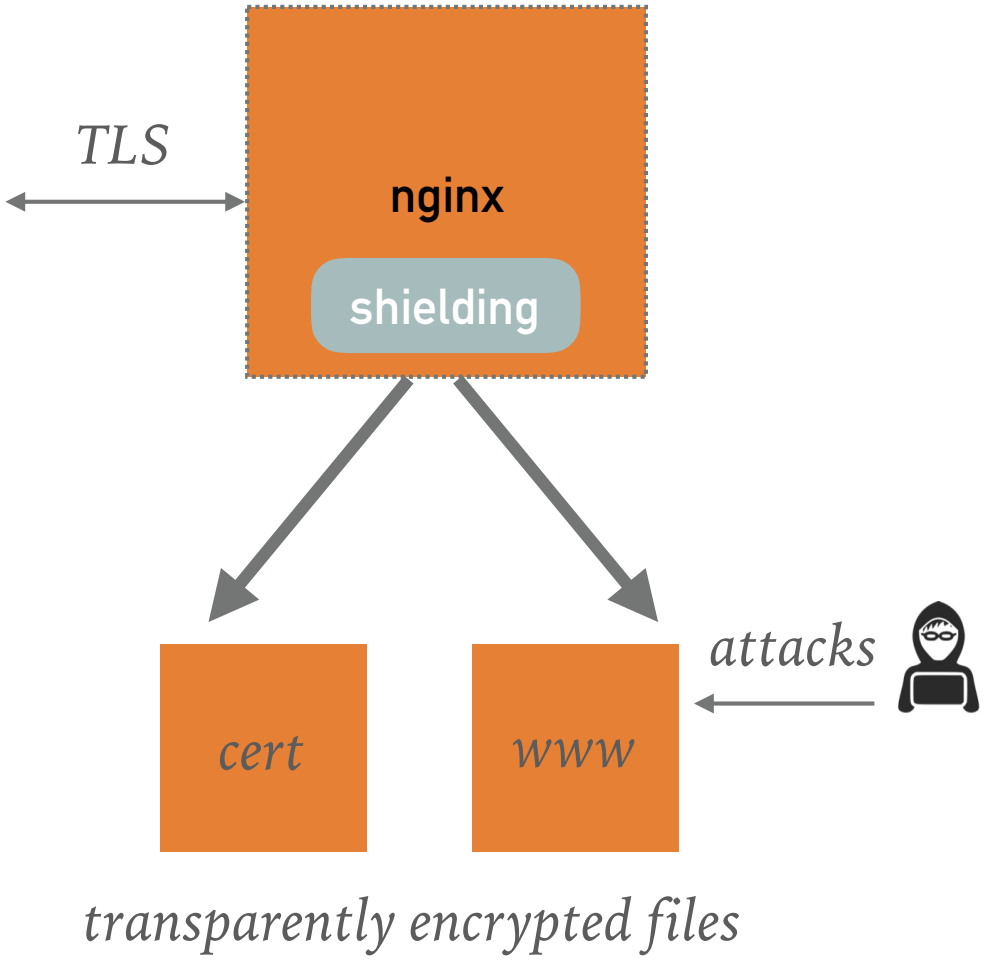


- Partitioning requires bespoke protection
  - need to ensure that TLS API calls are not malicious



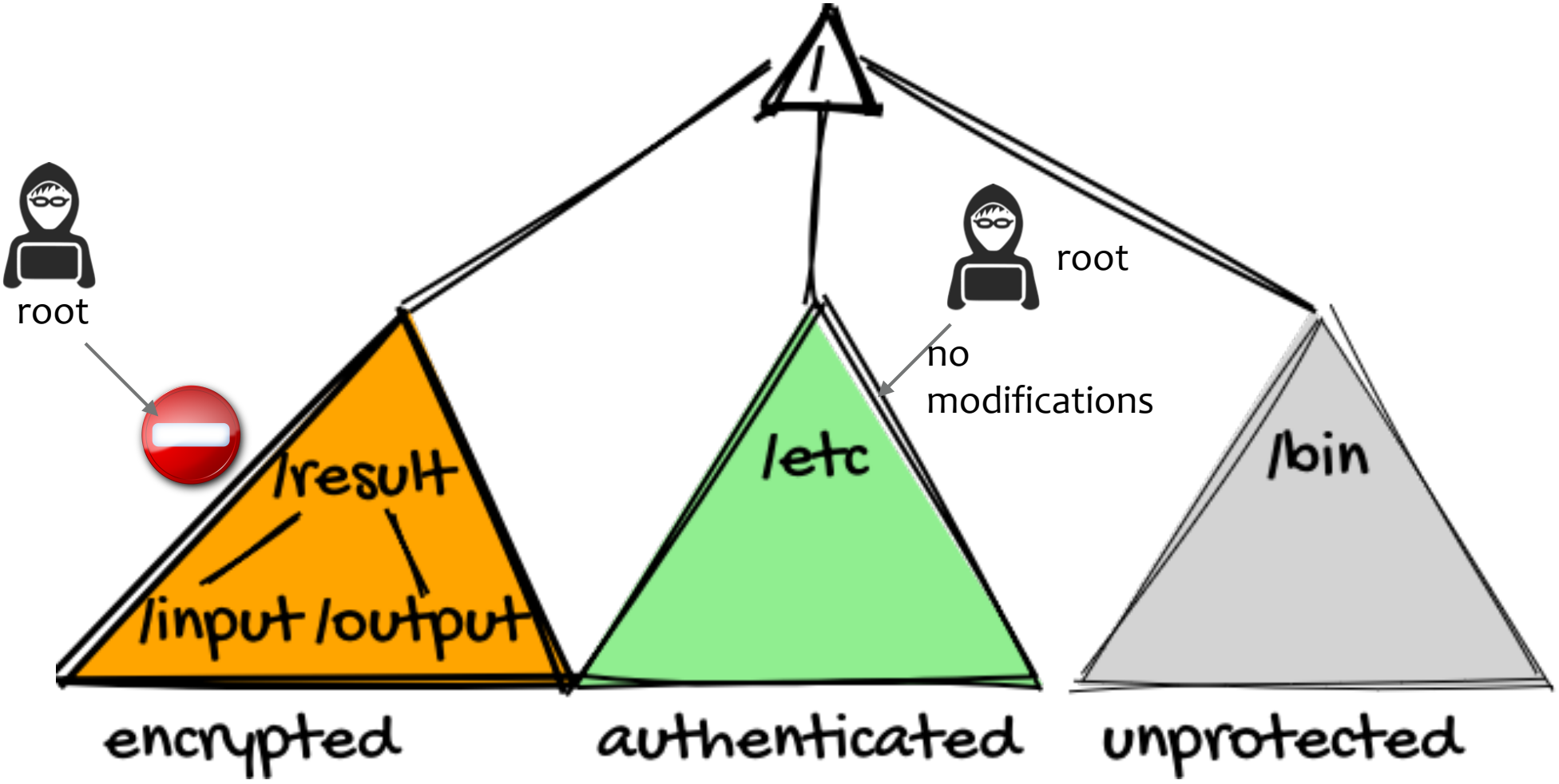
- **Protecting the OS Interface**
  - larger than TLS API
  - but reusable across many applications

FILESHIELD, TLS SHIELD,...



- Transparent encryption/ decryption of files
  - inside of enclaves
- In case app does not support TLS
  - can wrap TCP connections in TLS
  - e.g., memcache

SCONE: FILE ENCRYPTION



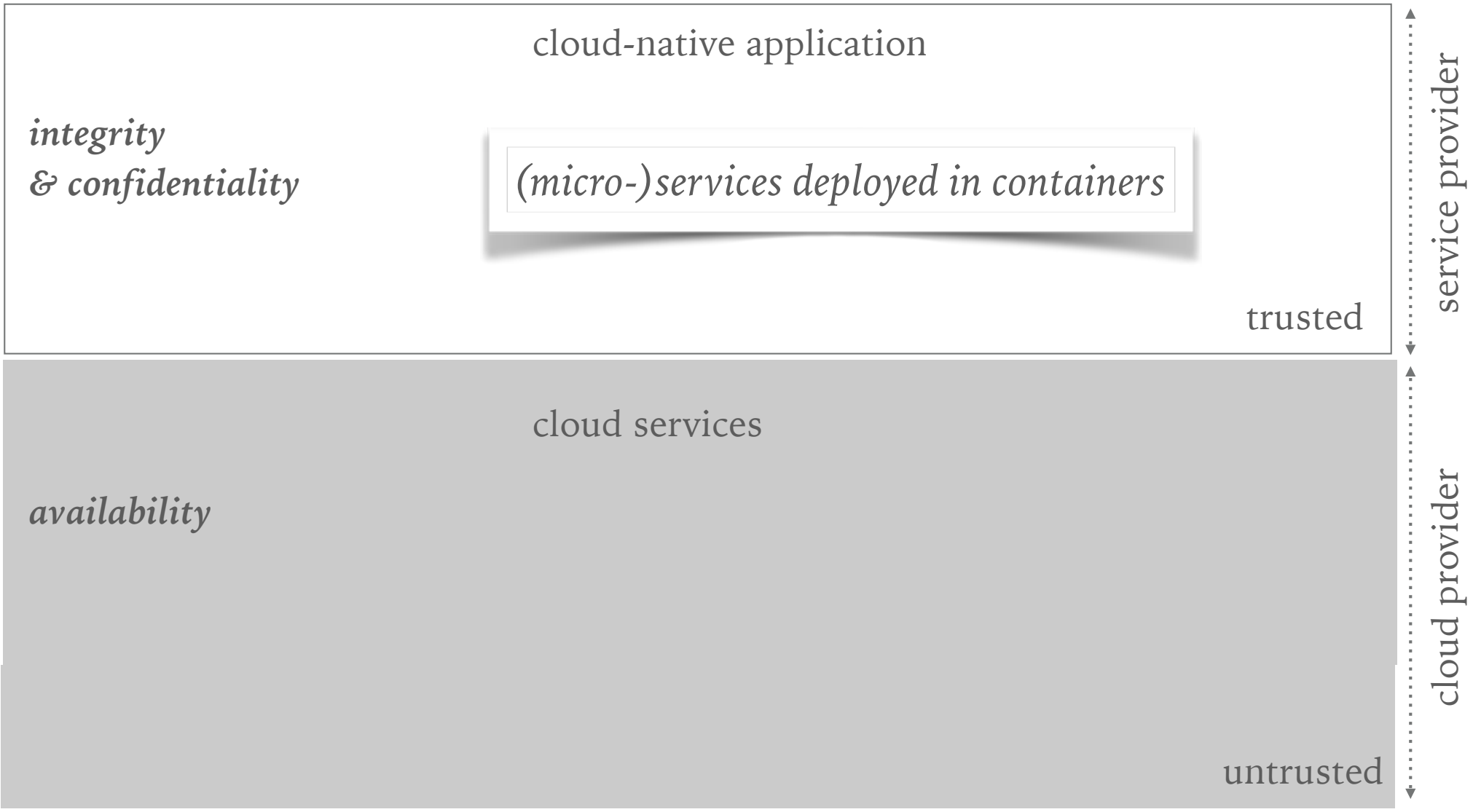
- Developer determines which files must be encrypted
- Encryption/authentication with source code changes

https://sconedocs.github.io

REALLY, NO PARTITIONING?

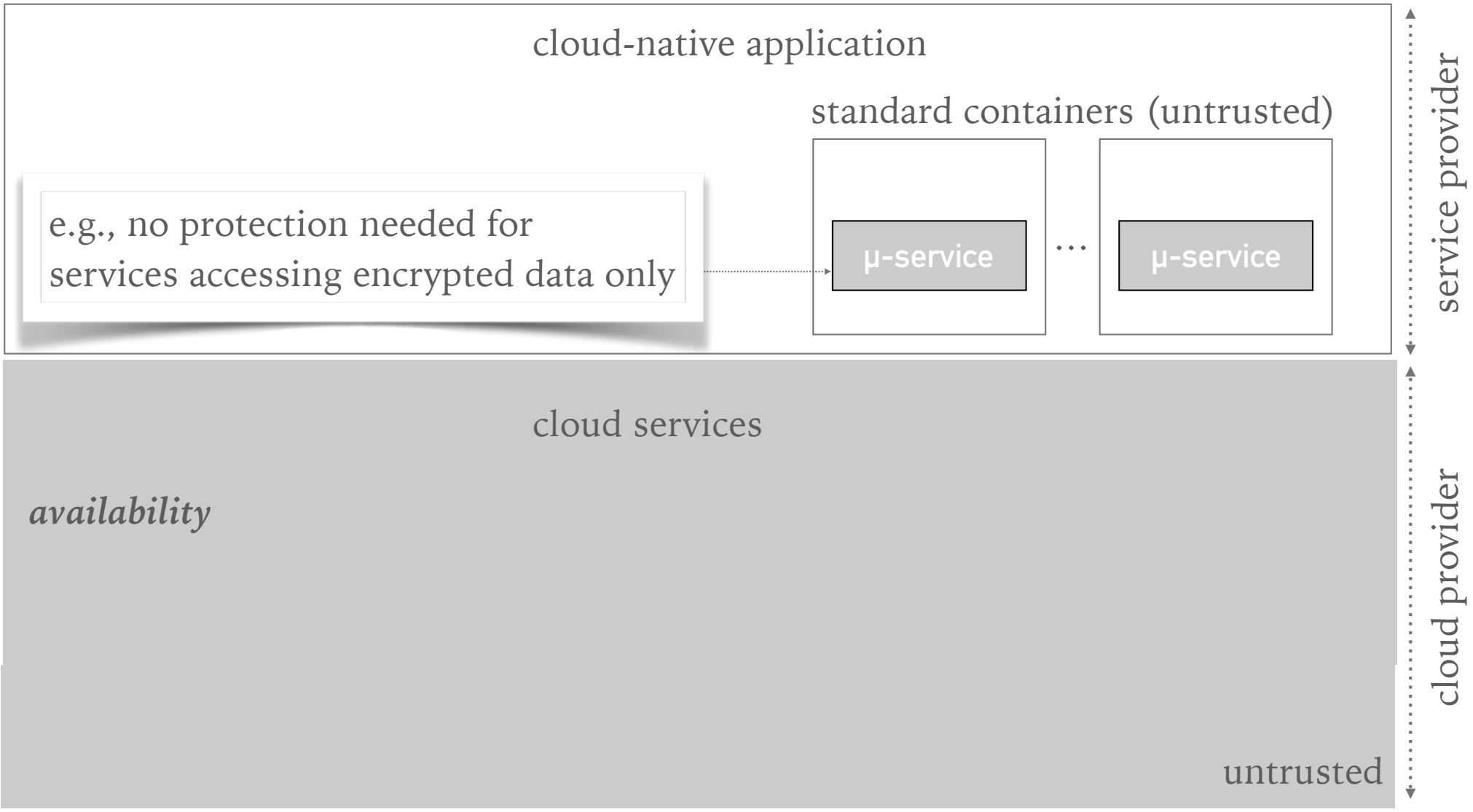
- focus on microservices -

CLOUD NATIVE APPLICATIONS



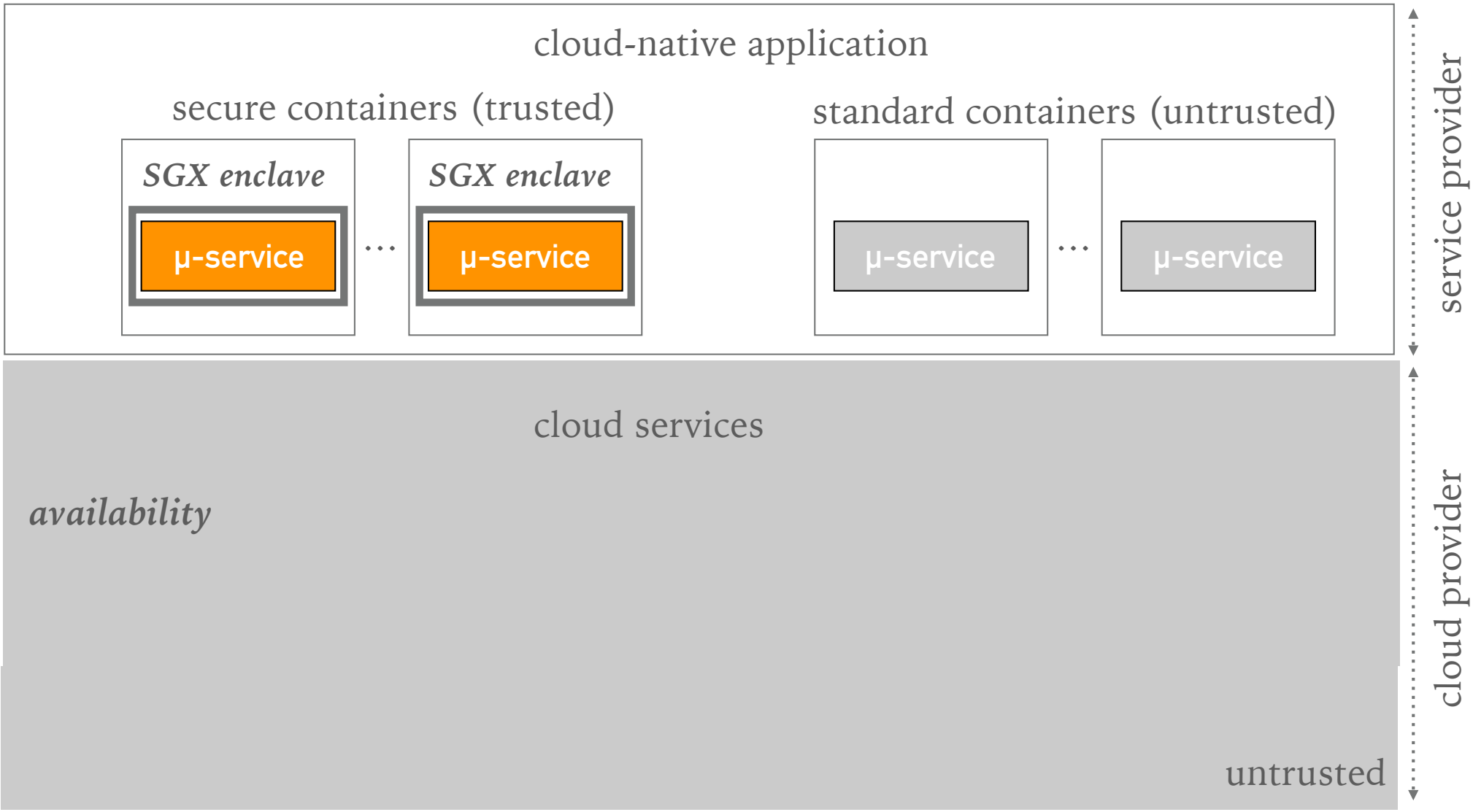
https://sconedocs.github.io

EACH MICROSERVICE RUNS IN A CONTAINER



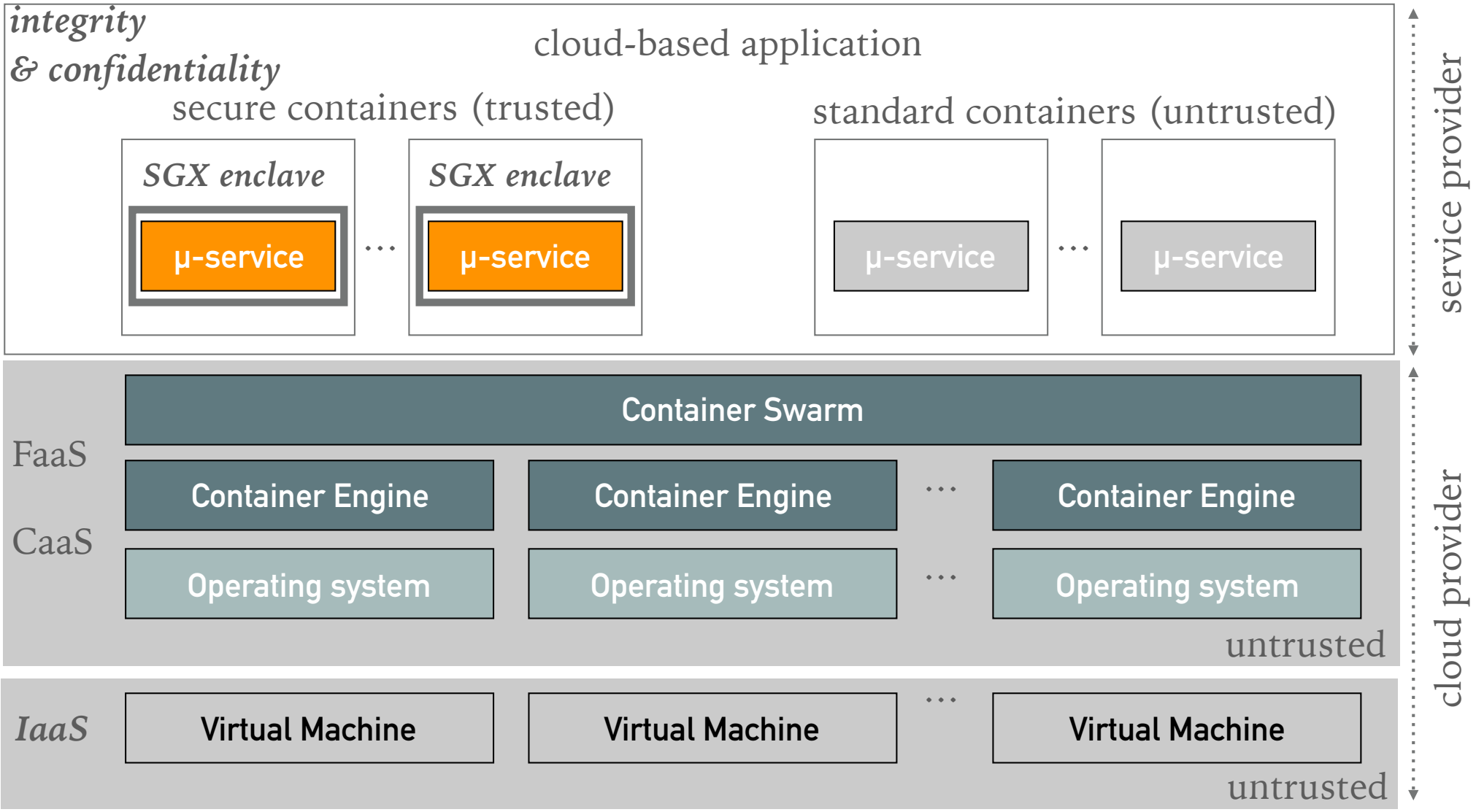
https://sconedocs.github.io

MICROSERVICE-BASED PARTITIONING



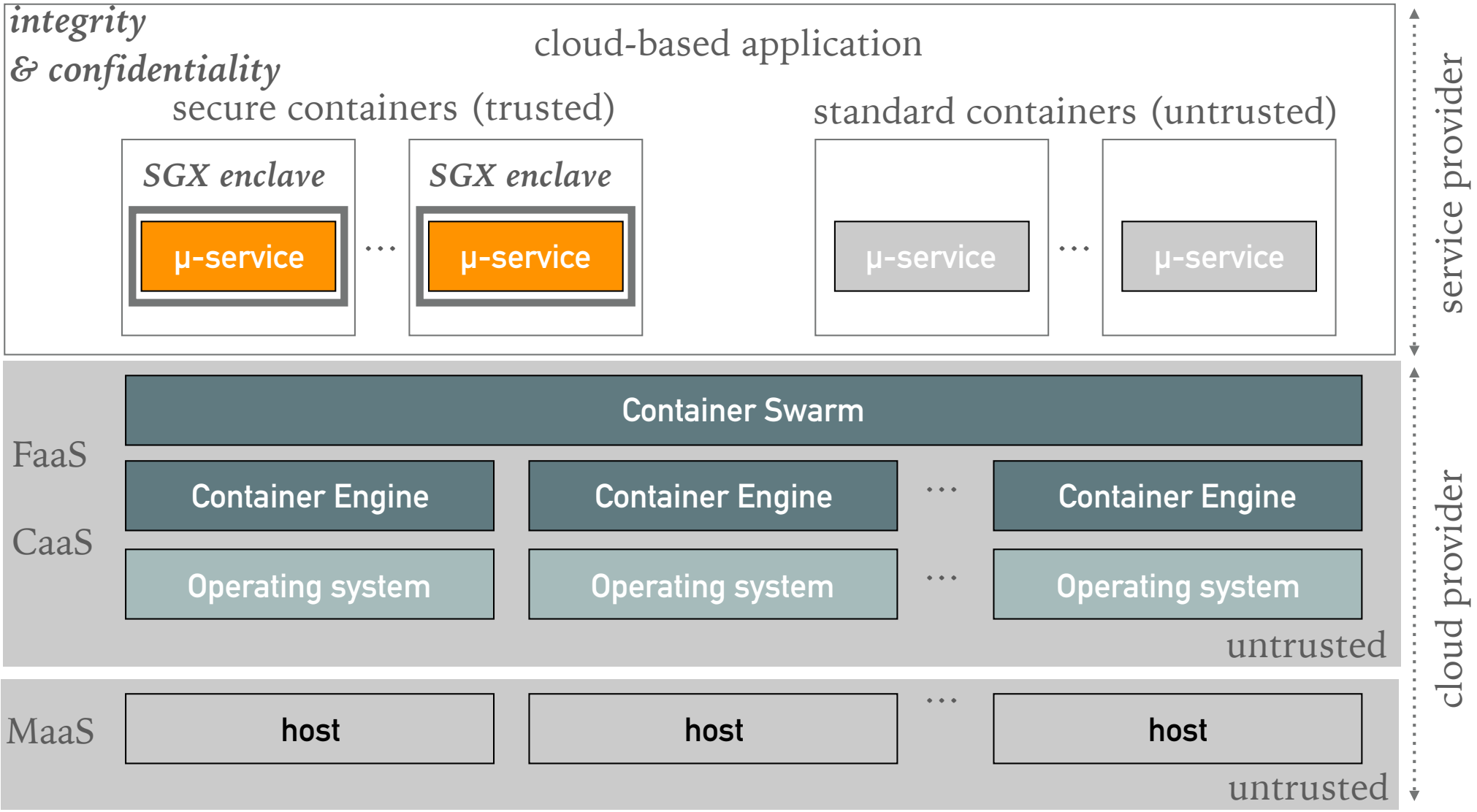
https://sconedocs.github.io

CONTAINER-AS-A-SERVICE AND/OR IAAS



https://sconedocs.github.io

METAL-AS-A-SERVICE



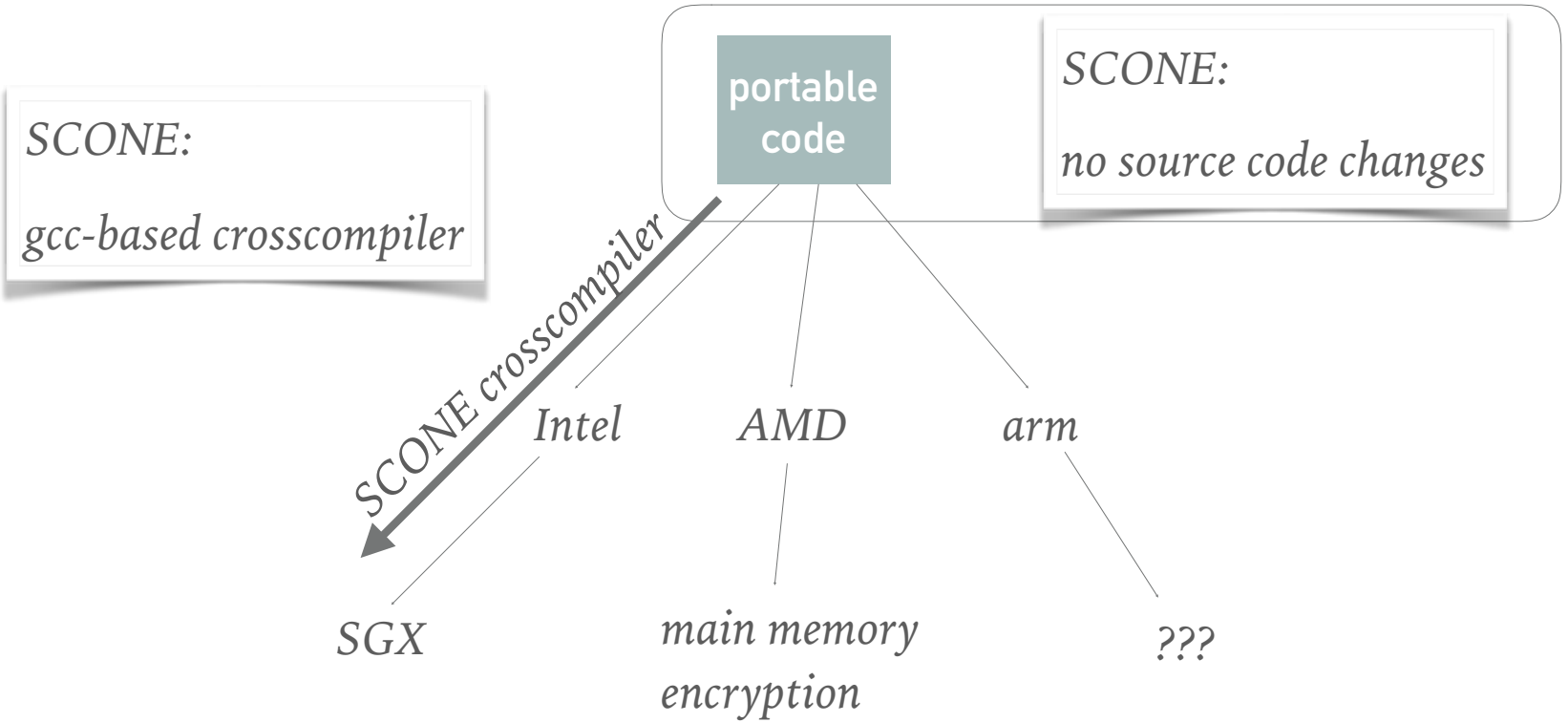
https://sconedocs.github.io

PORTABILITY

- reduce cost / avoid lock in -

SCONE PLATFORM: DESIGNED FOR MULTIPLE ARCHITECTURES

Languages: C, C++, Go, Rust, **interpreted/JIT:** Java, Python, R, …



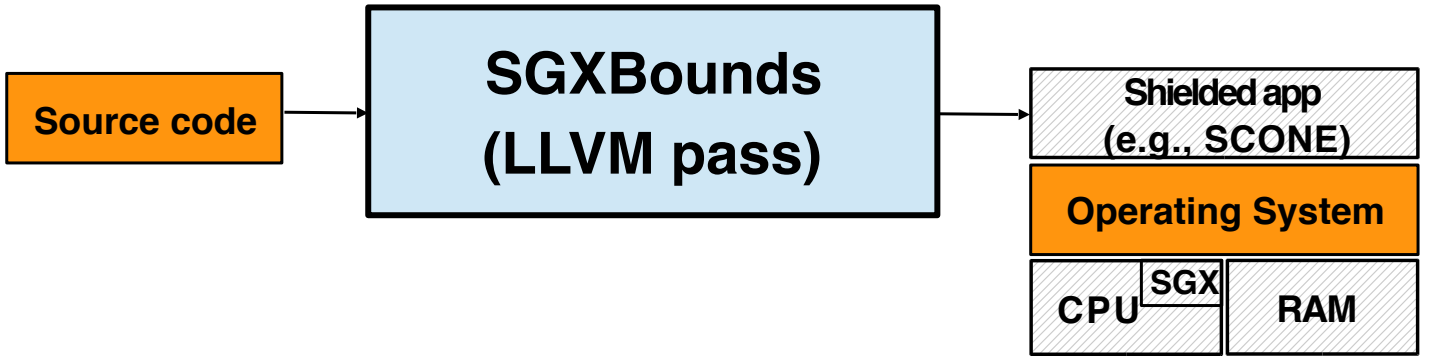
Portability through cross-compilation

https://sconedocs.github.io

MEMORY SAFETY?

- exploiting application bugs -

SGXBOUNDS FOR MEMORY SAFETY (C AND C++)



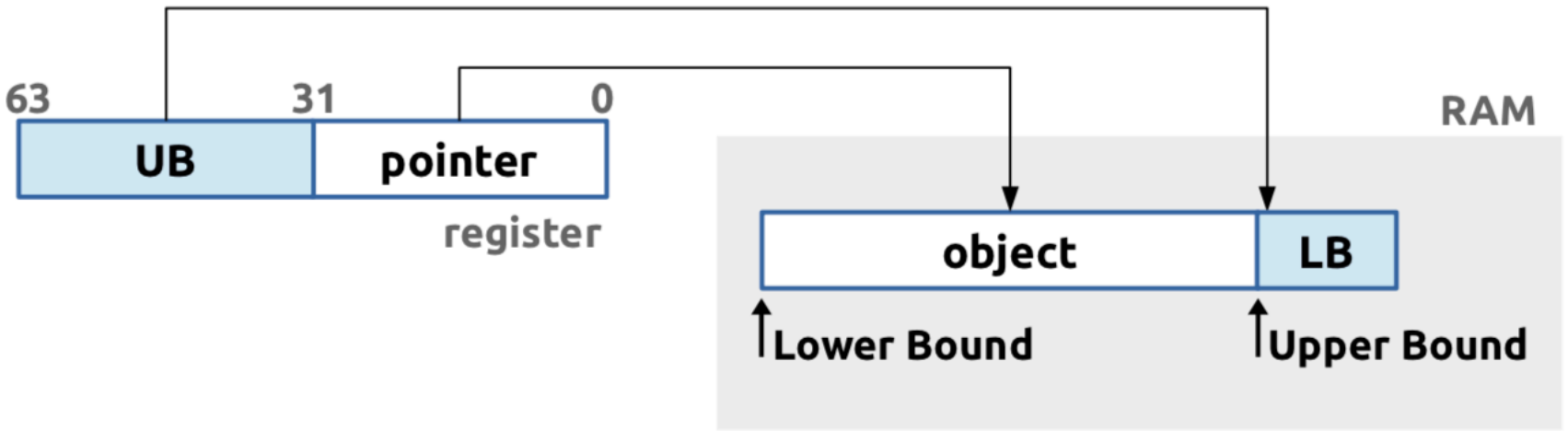
Advanced features:

- ➥ **Tolerating** errors with **boundless memory**
- ➥ **Metadata management** support
- ➥ Compile-time **optimizations**

See paper for details

Dmitrii Kuvaiskii, Oleksii Oleksenko, Sergei Arnautov, Bohdan Trach, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. 2017. SGXBOUNDS: Memory Safety for Shielded Execution. In *Proceedings of the Twelfth European Conference on Computer Systems* (**EuroSys '17**). ACM, New York, NY, USA, 205-221.

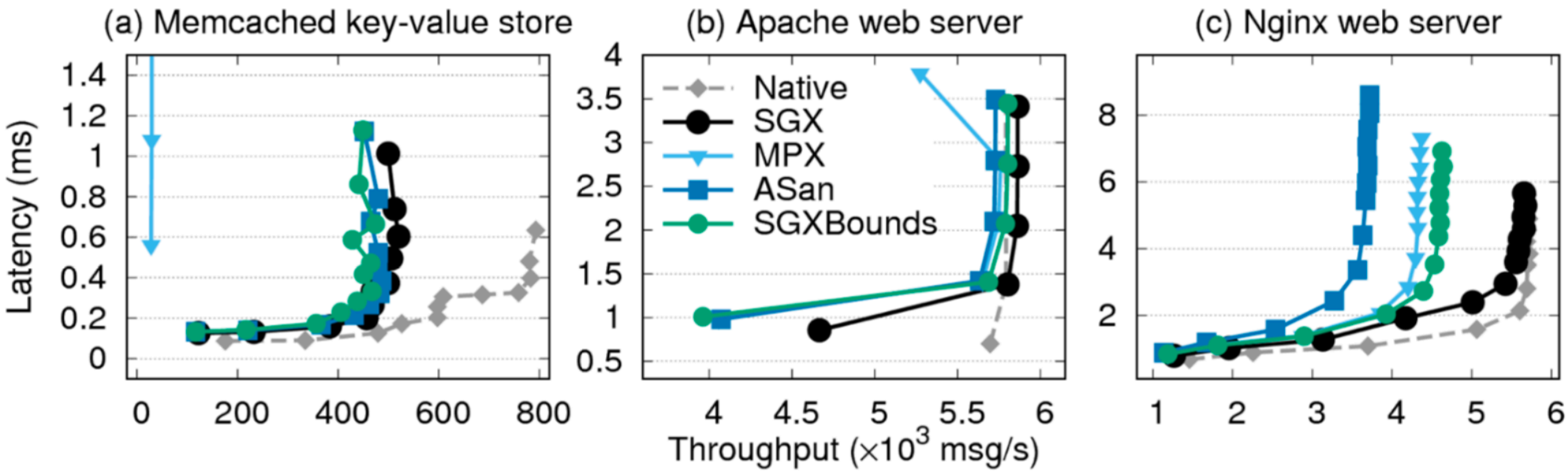
SGXBOUNDS IN A NUTSHELL



- **Lower bound (LB)**: placed after allocated **object**
- **Upper bound (UB):** embedded into 64-bit **pointer**
  - "tagged pointer" as opposed to "fat pointer"
  - SGX enclaves address 32-bit space (max 36 bits)

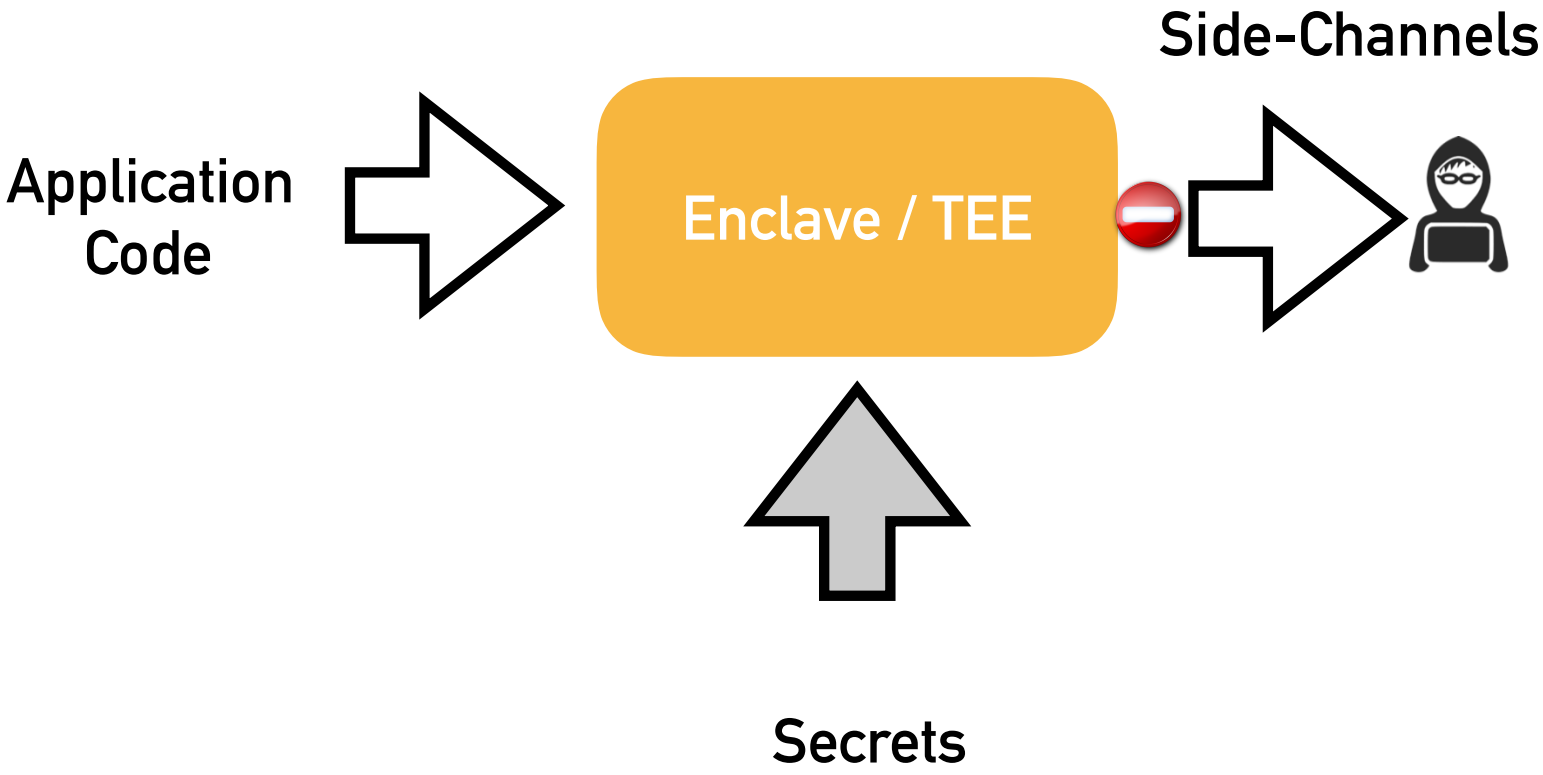
☺ Cache-friendly layout → minimal memory overhead

PERFORMANCE



	ASan	MPX	SGXBounds
Phoenix	1.41	2.27	**1.13**
PARSEC	1.60	1.43*	**1.20**
SPEC	1.76	1.52*	**1.41**

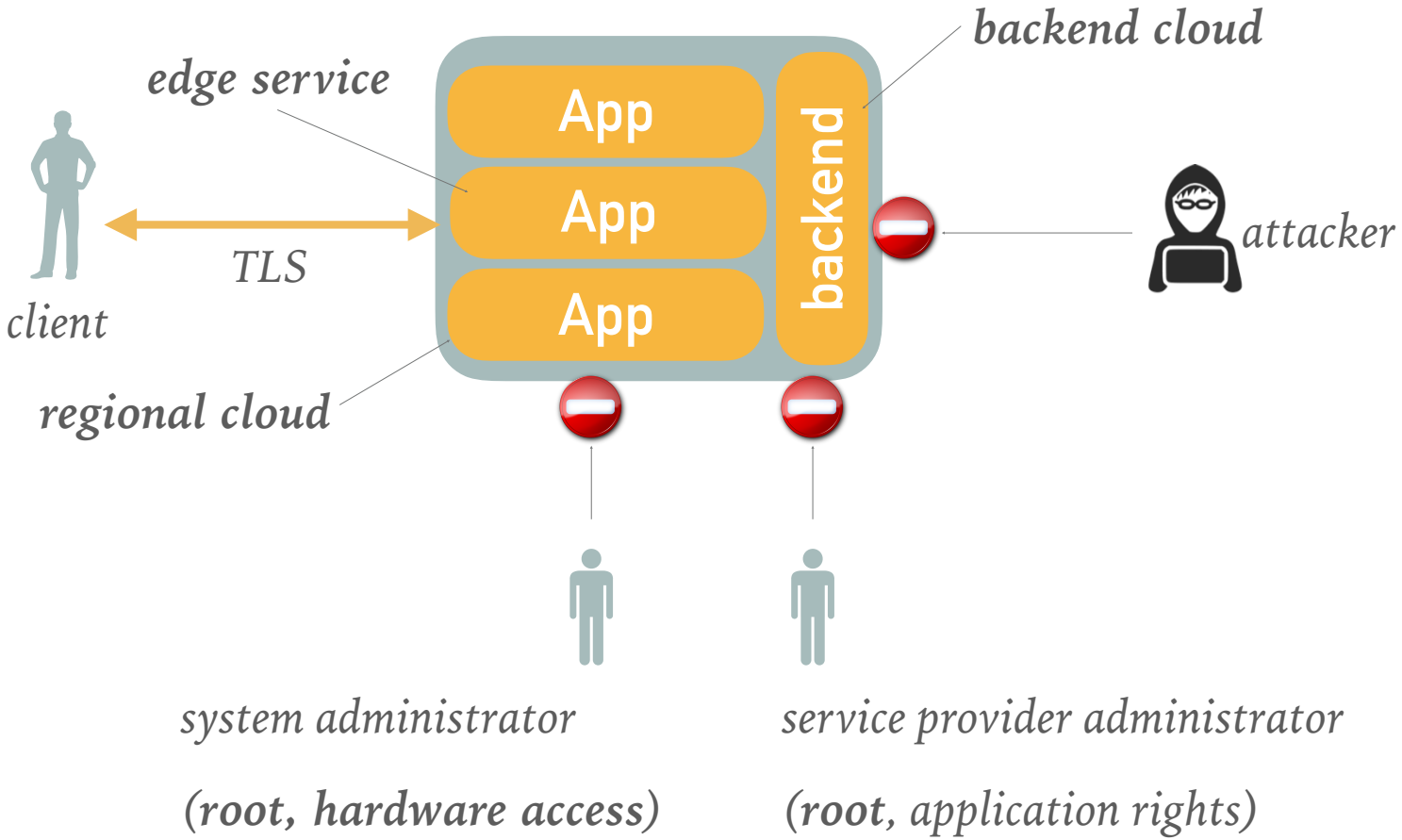
SECRETS MANAGEMENT



DISTRIBUTED APPLICATIONS

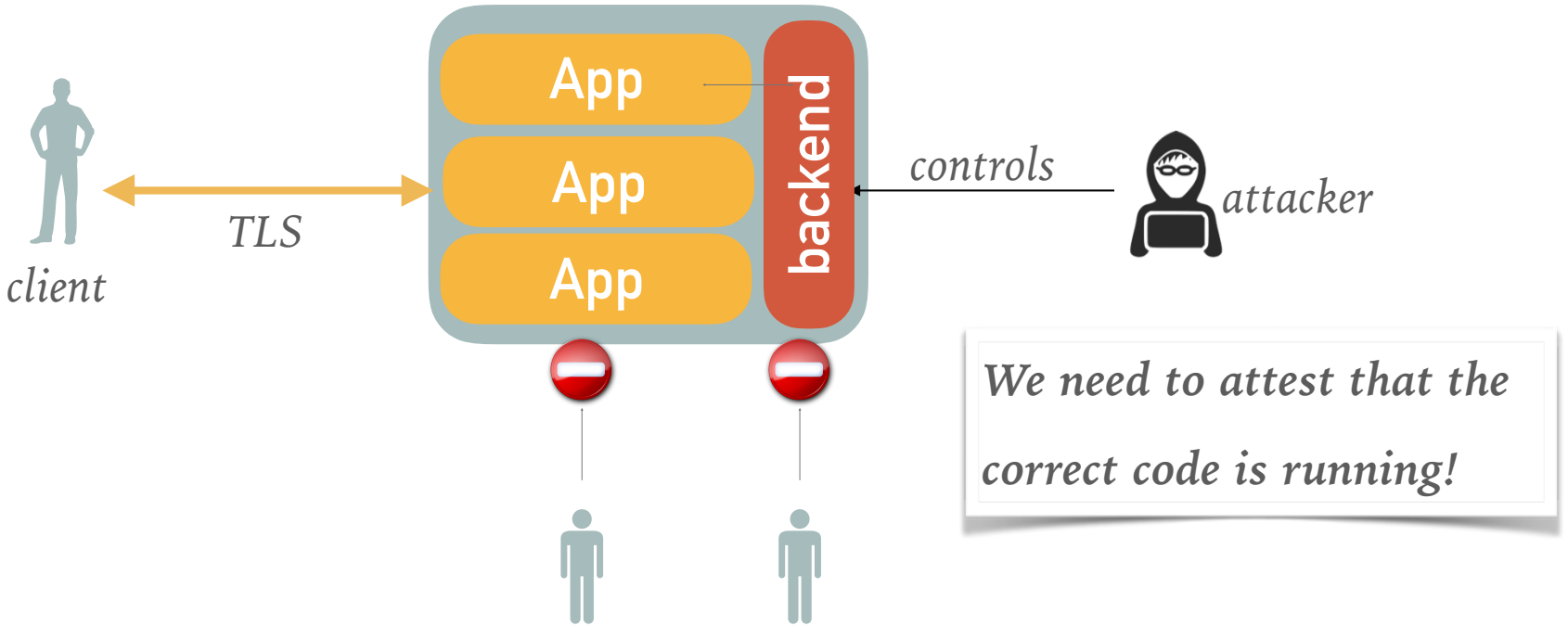
- motivation -

DISTRIBUTED APPLICATIONS – SPREAD ACROSS CLOUDS



https://sconedocs.github.io

HOW DO WE KNOW THAT CORRECT CODE EXECUTES?



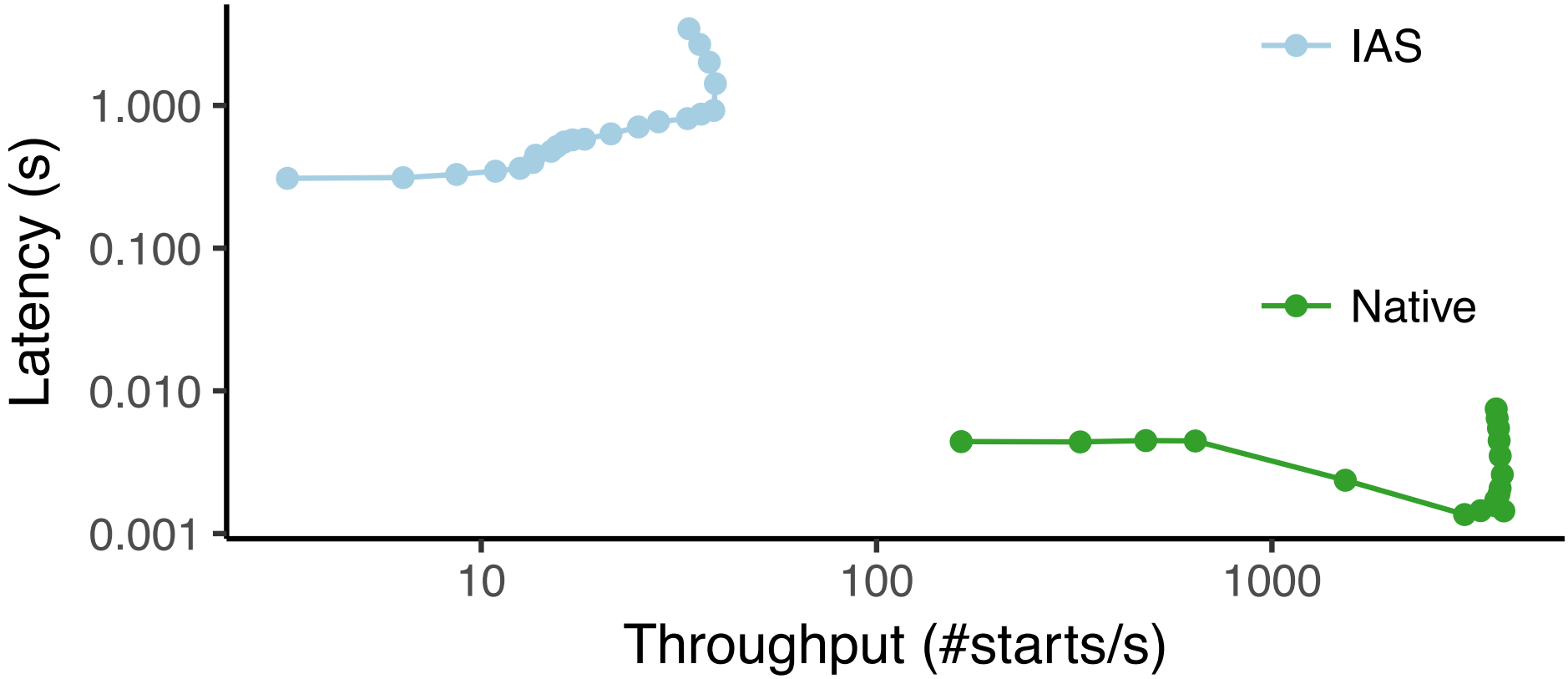
https://sconedocs.github.io

TRANSPARENT ATTESTATION & CONFIGURATION

- problems? -

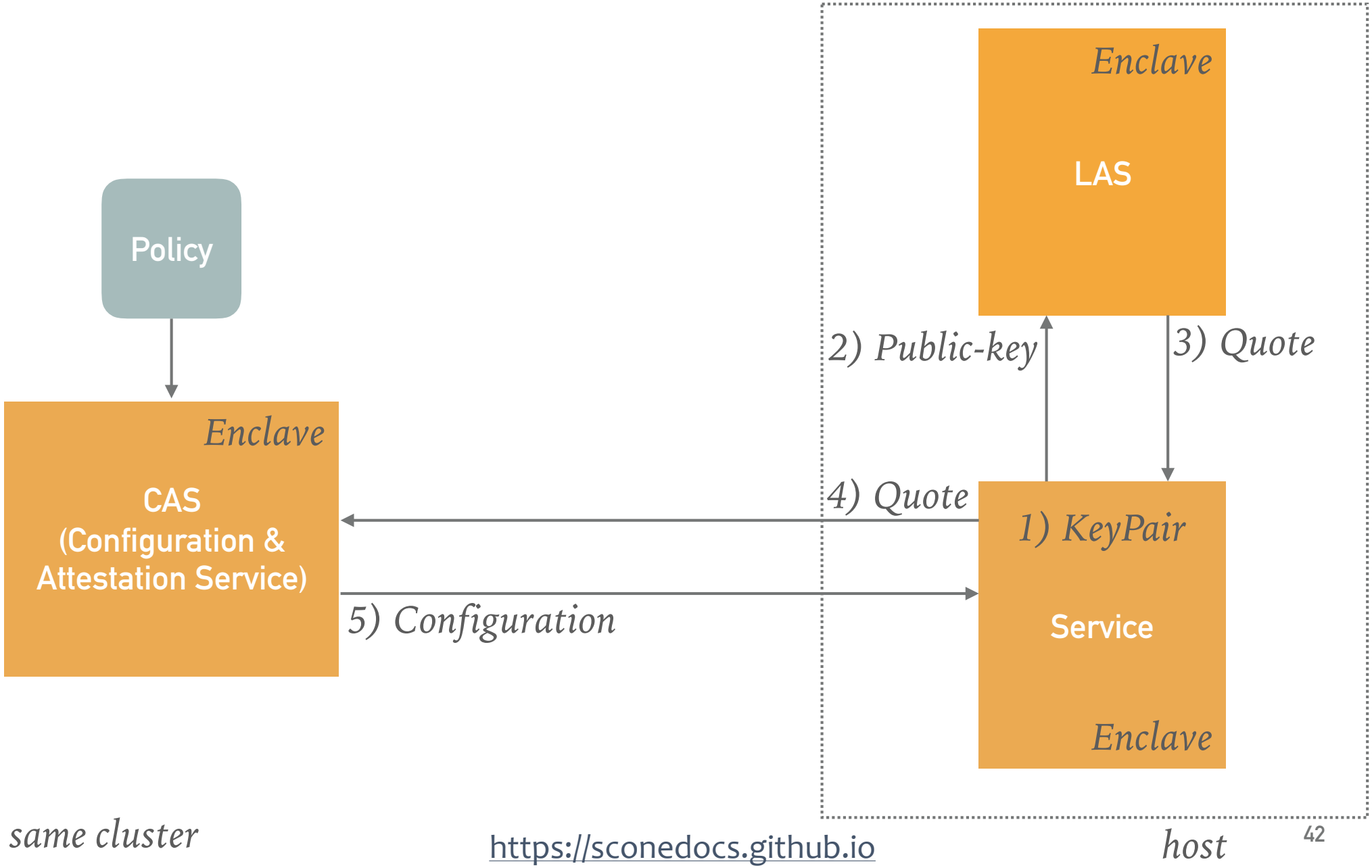
PROBLEM: STARTUP LATENCY!

- **Attestation via Intel Attention Service**
  - rate in which we can spawn containers limited/depends on Intel

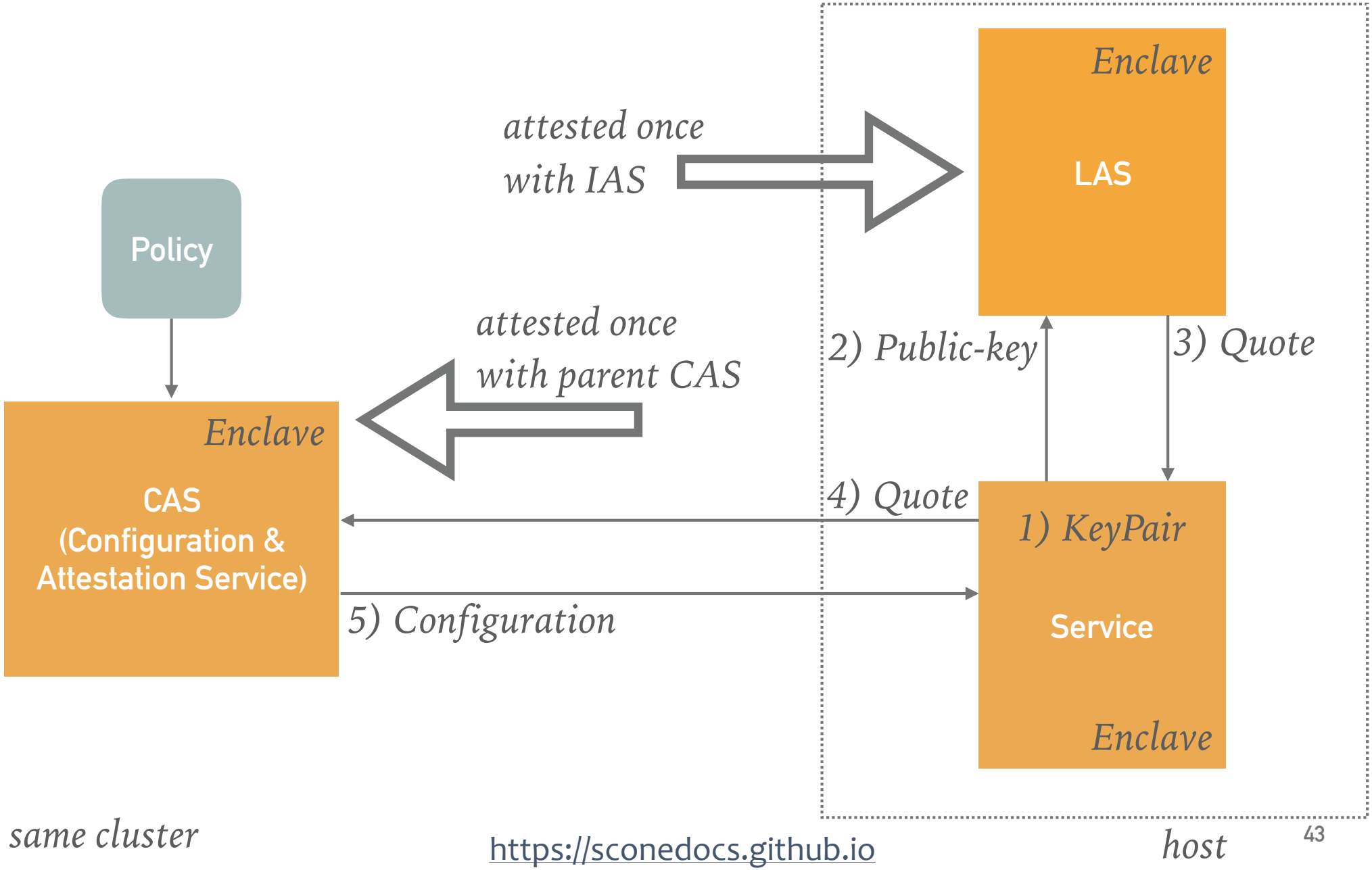


https://sconedocs.github.io

TRANSPARENT ATTESTATION

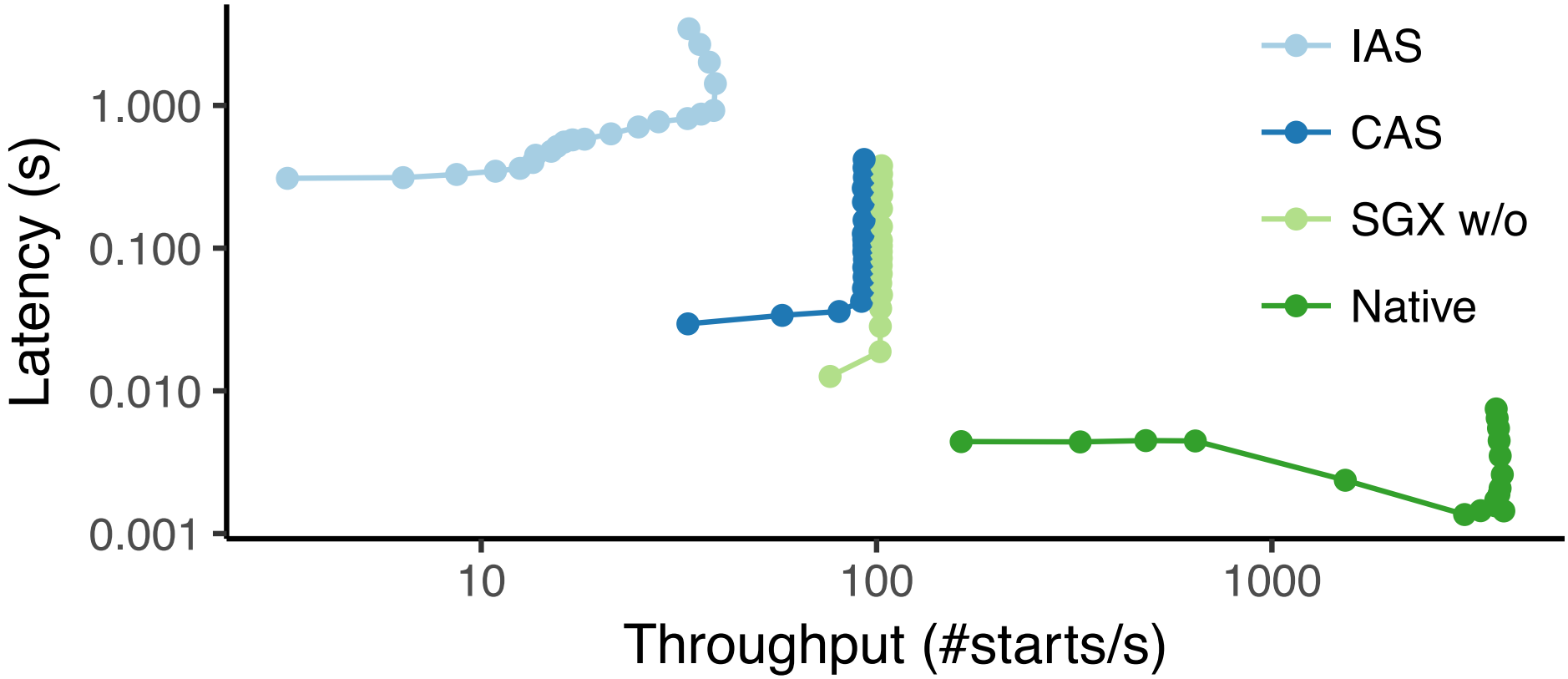


TRANSPARENT ATTESTATION



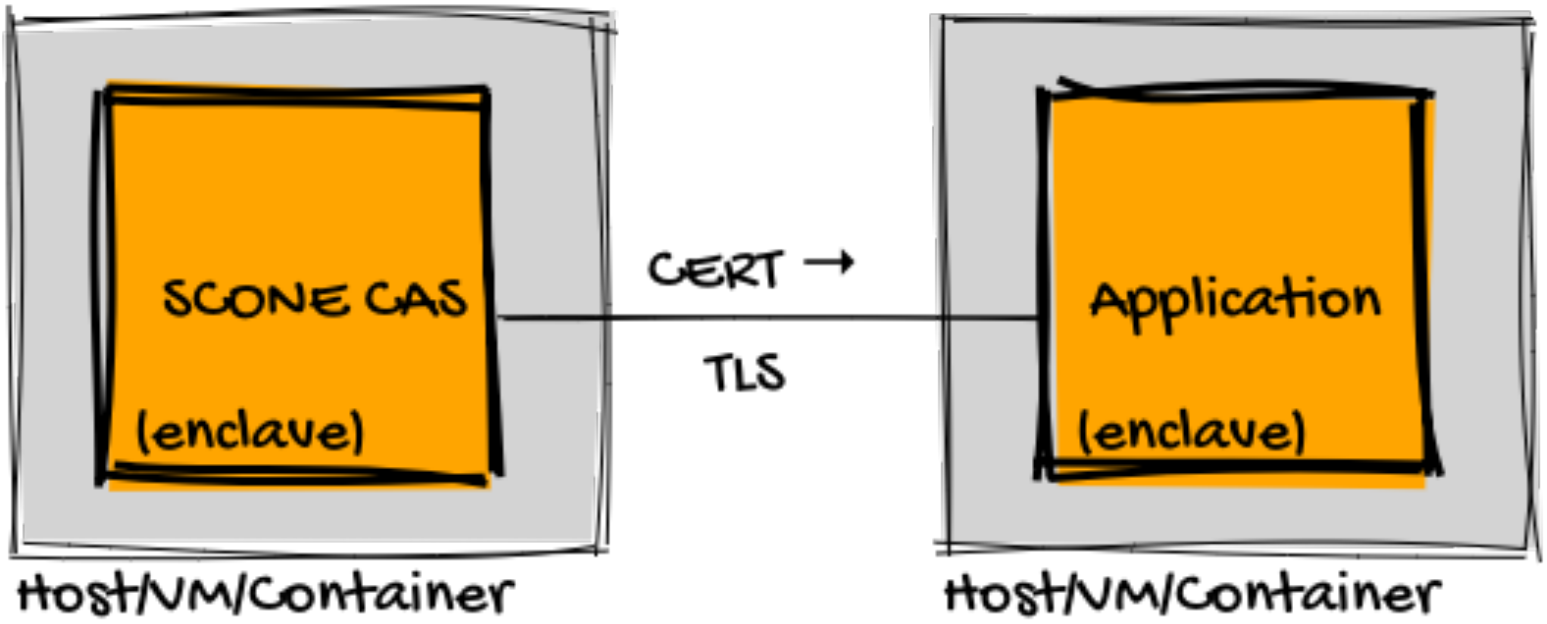
PROBLEM: STARTUP LATENCY!

- Attestation via Intel takes too long time
  - rate in which we can spawn containers limited/depends on Intel



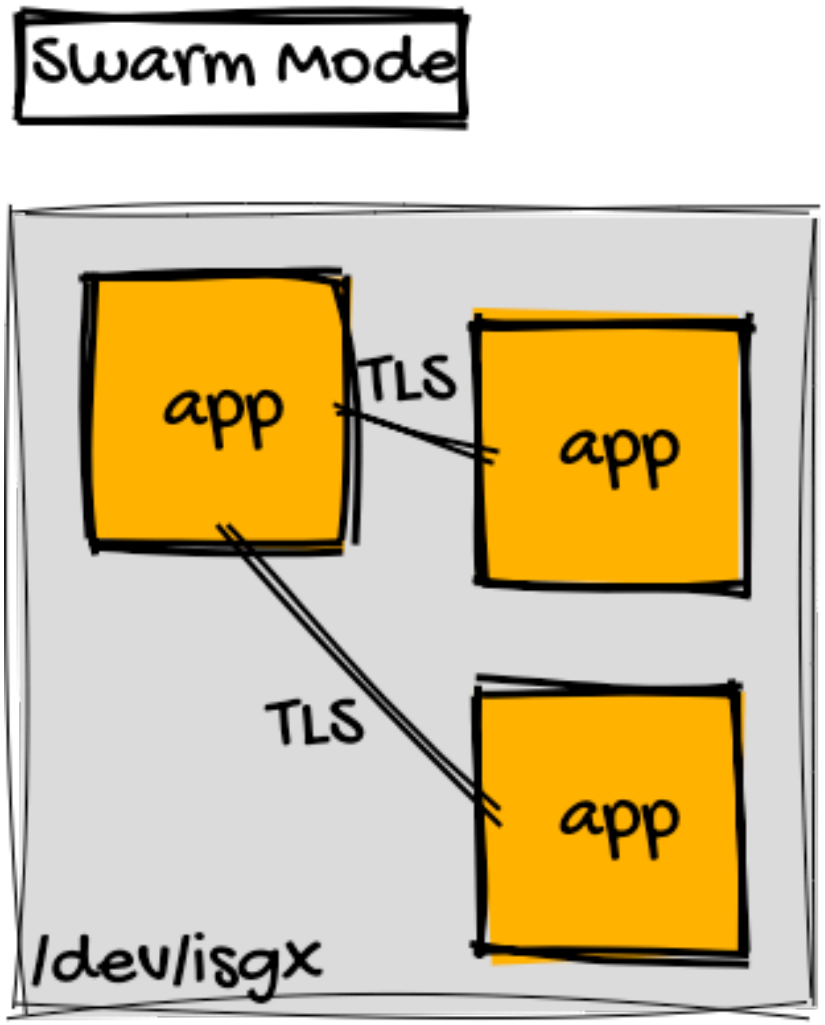
https://sconedocs.github.io

NO SOURCE CODE CHANGES?



- **Problem**: how can the application show that it runs inside an enclave and runs correct code?
- **Approach**: attest application and give it a certificate

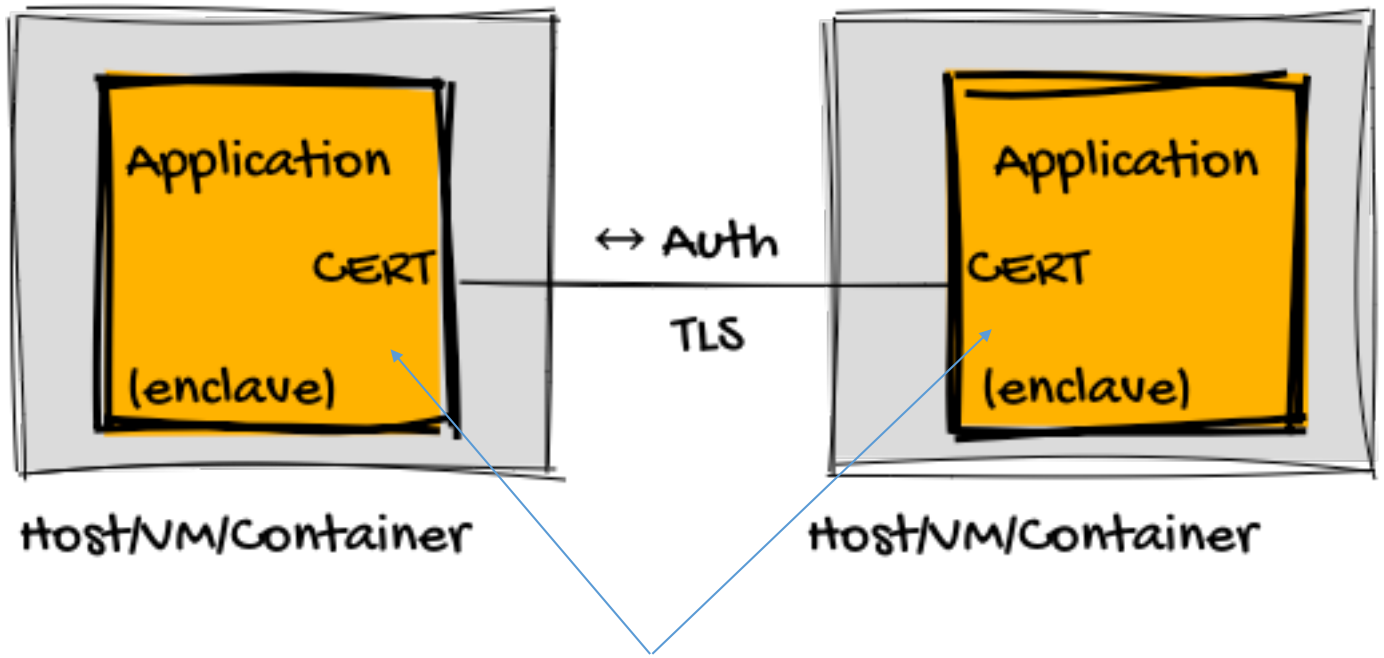
REST INTERFACES



- Application is partitioned into microservices
- Microservices run on same or on different machines
- REST APIs protected by TLS
  - could add transparently if needed („SCONE *TLS shield*“)

TRANSPARENT P2P ATTESTATION VIA TLS

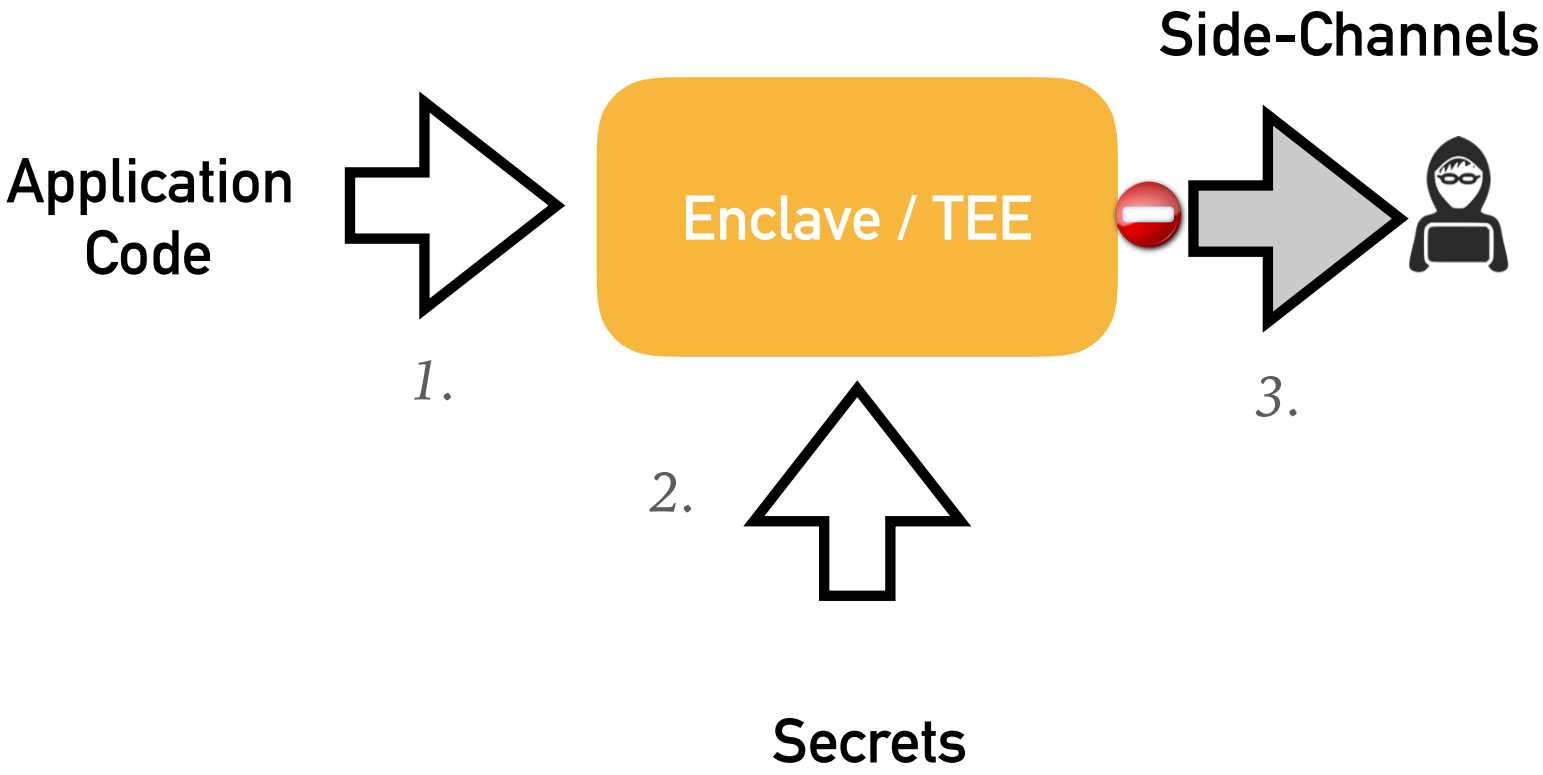
We run our internal CA and only components belonging to the same application can talk to each other …



Certificate Authority
(integrated in SCONE CAS)

https://sconedocs.github.io

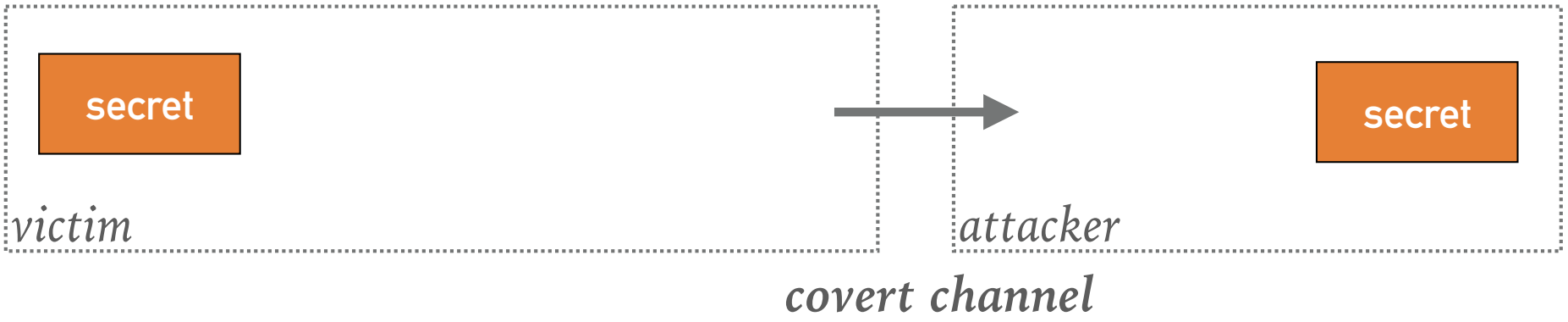
SECRETS MANAGEMENT



SIDE CHANNELS

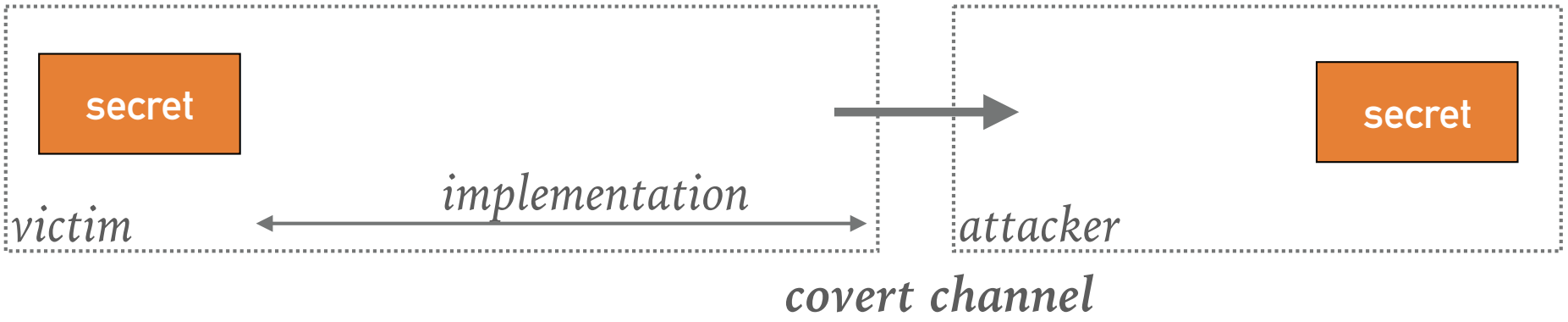
- dealing various side channels -

SIDE CHANNEL ATTACK?



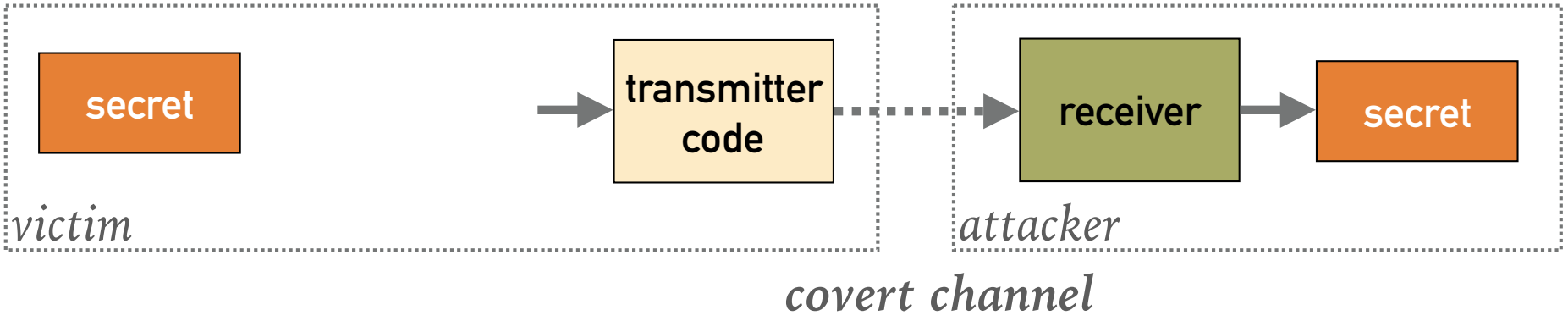
- An **attacker** steals a **secret** from a **victim** via a **covert channel**:
  - i.e., a channel that is not intended to communicate information
- Examples: **timing**, **power**, **cache**, …

SIDE CHANNEL ATTACK?



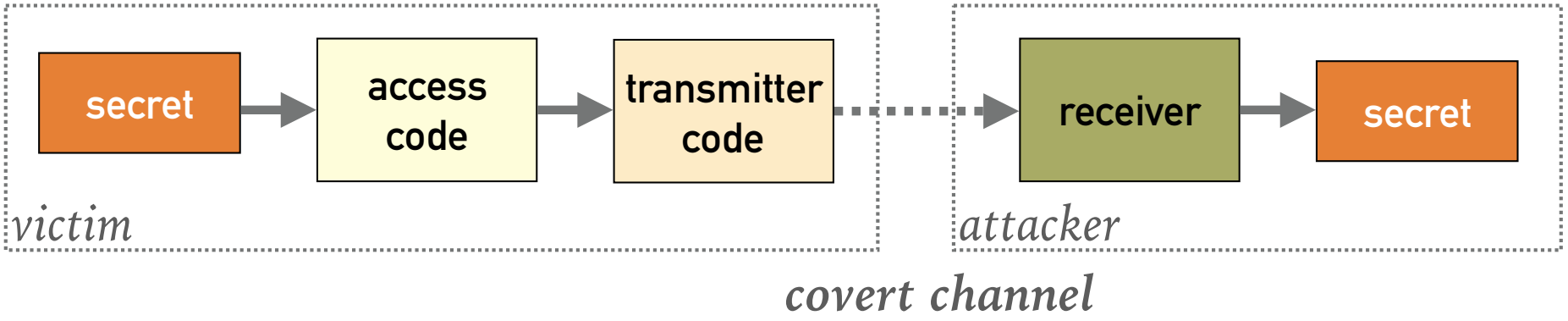
- Side channel attack:
  - information gained from the implementation
    - not from the implemented algorithm

SIDE CHANNEL ATTACK



- We need a
  - **transmitter code** in the victim to send the secret, and a
  - receiver to store the secret

SIDE CHANNEL ATTACK



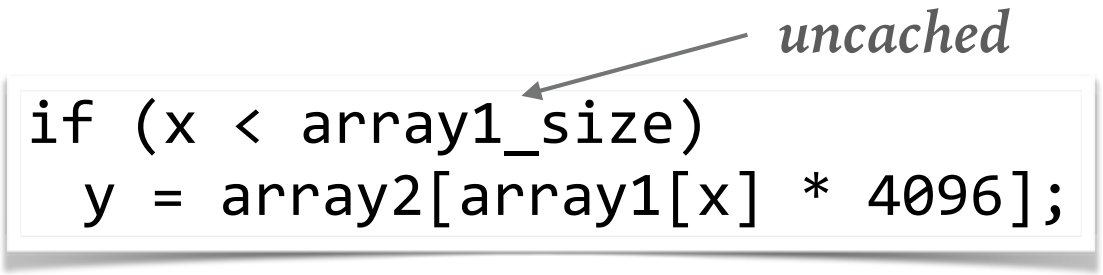
- Finally, we need
  - **access code** to get the secret to the transmitter code

WHAT SIDE-CHANNELS?

- **Spectre Variant 1**
- Spectre Variant 2
- Meltdown
- Spectre NG
- Foreshadow
- …

- Let’s focus on Spectre Variant 1

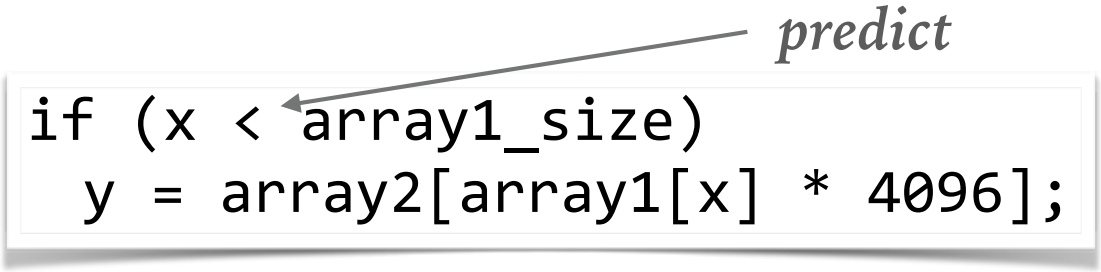
NON-SPECULATIVE EXECUTION



***Note**: execution must wait until array1_size is fetched from memory*

Paul Kocher et al, „Spectre Attacks: Exploiting Speculative Execution“, 40th IEEE Symposium on Security and Privacy (S&P'19)

SPECULATIVE EXECUTION



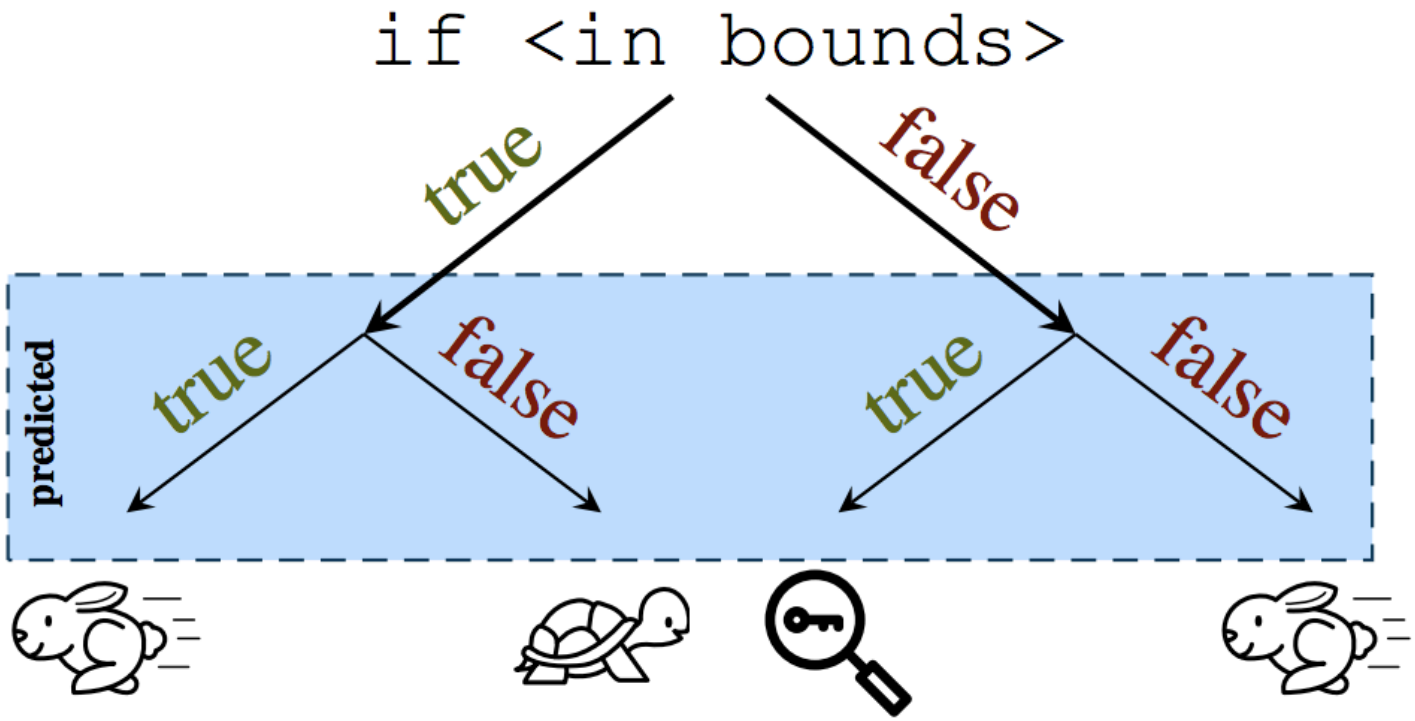
***Approach**: CPU predicts the outcome of the comparisons*

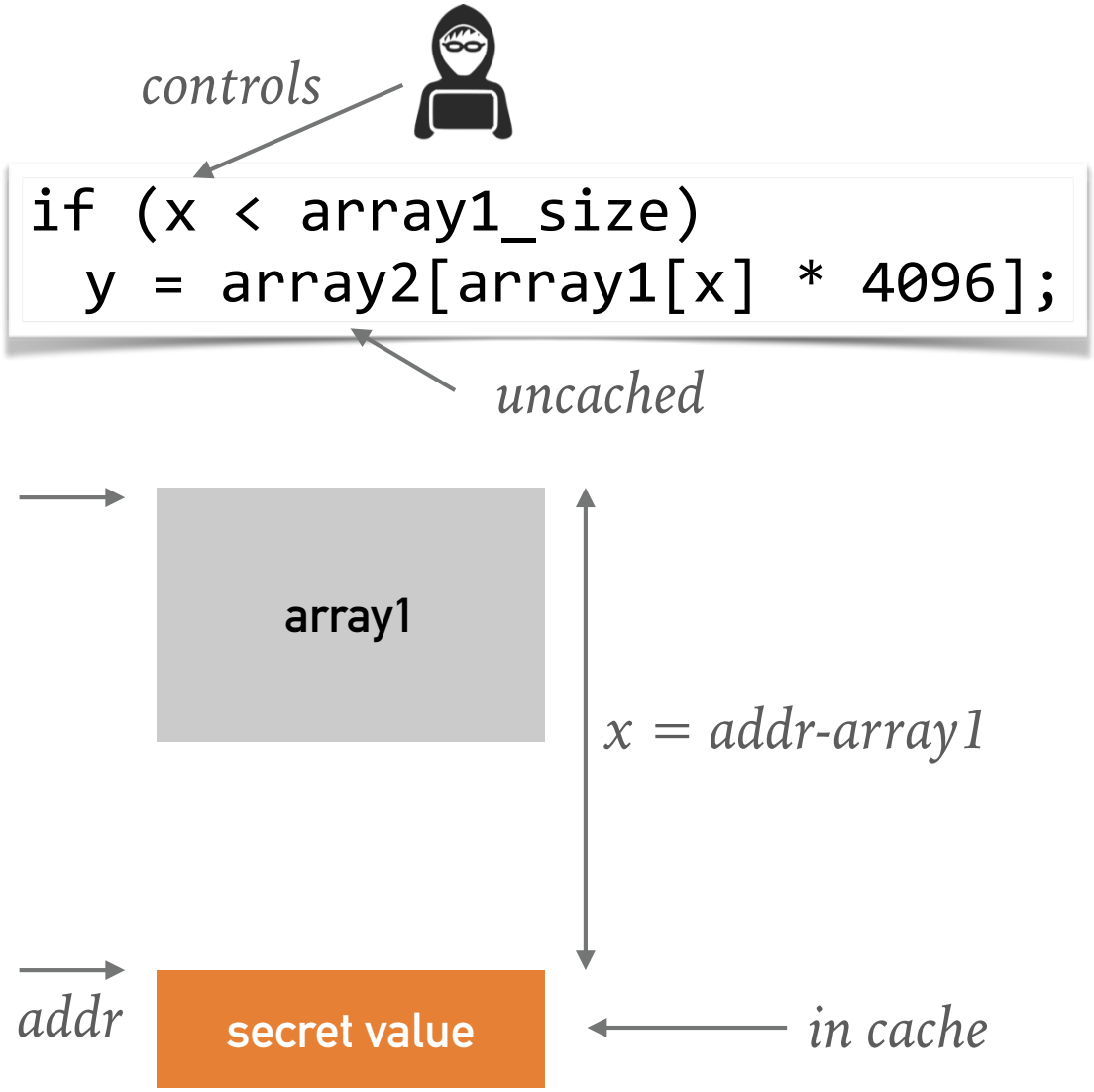
Paul Kocher et al, „Spectre Attacks: Exploiting Speculative Execution“, 40th IEEE Symposium on Security and Privacy (S&P'19)

EXPLOITING CONDITIONAL BRANCH MISPREDICTION

```
if (x < array1_size)
  y = array2[array1[x] * 4096];
```

- Branch prediction to speed up computations:

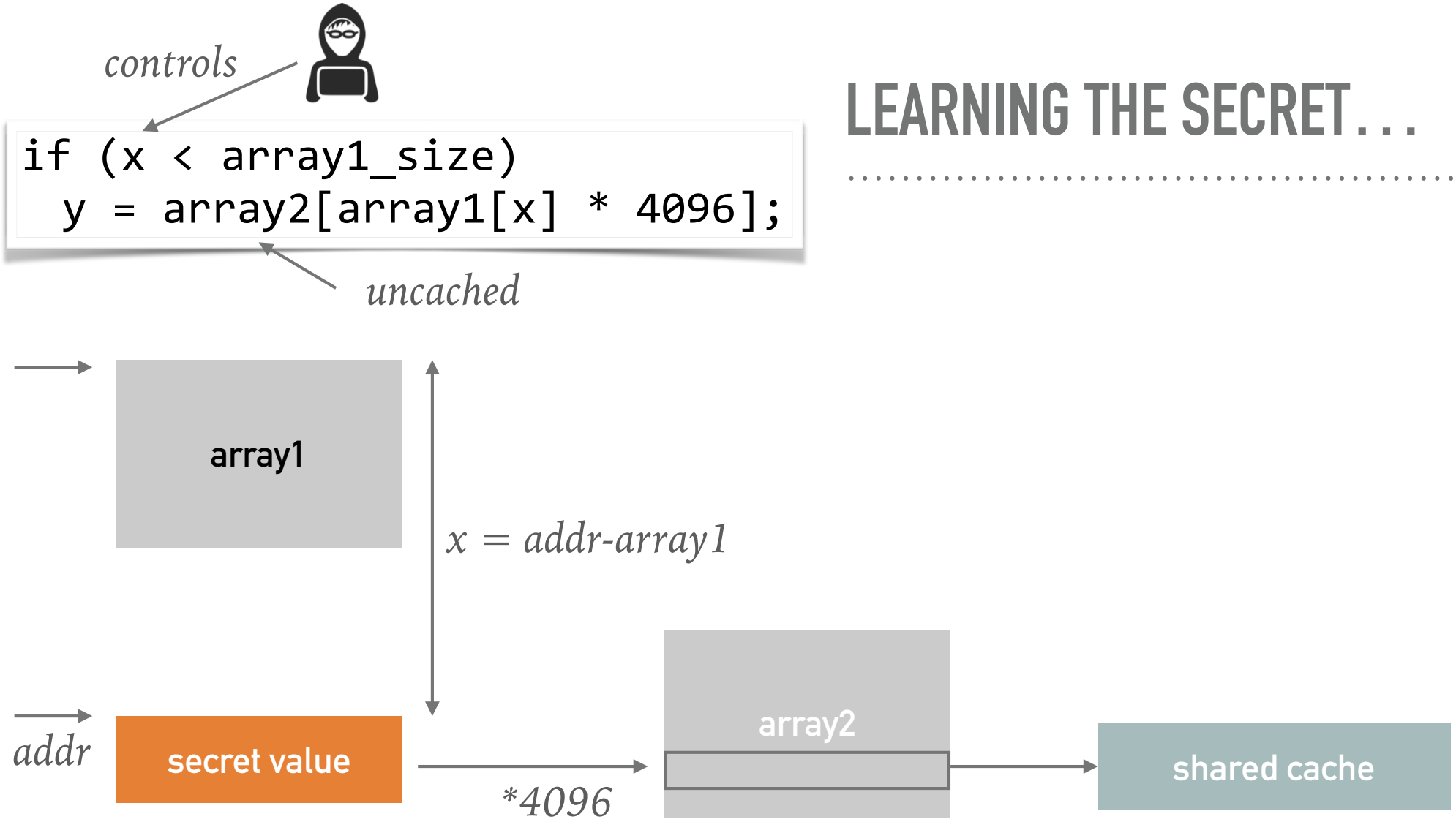




MISPREDICTION

- CPU might (mis)predict **true**:
  - accesses **array1** out-of-bound
  - will speculatively access
    - **array2[secret value*4k]**
- **CPU will eventually detect this**
  - rolls back updates

LEARNING THE SECRET...



- Value array2[secret value*4096] is read into cache

ACCESS CODE

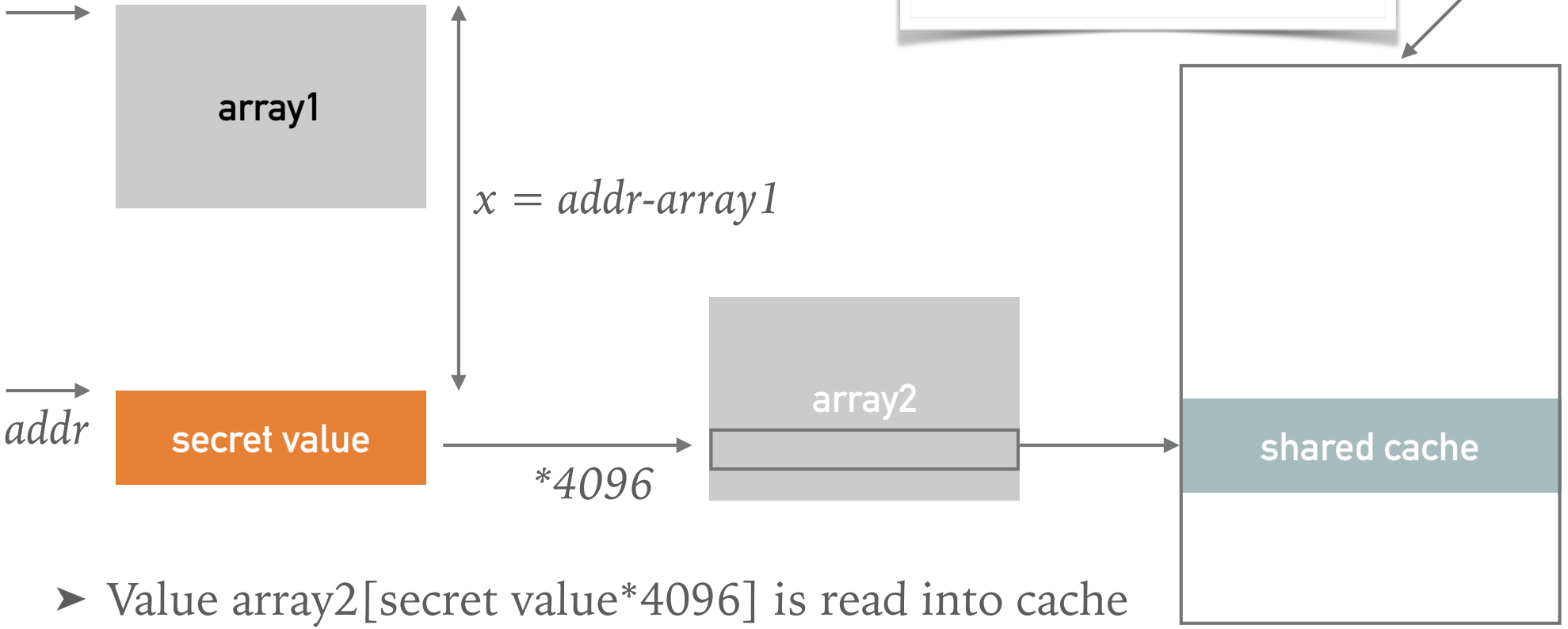
```
if (x < array1_size)
  y = array2[array1[x] * 4096];
```

uncached

LEARNING THE SECRET...

probes cache to detect

which index was accessed



- Value array2[secret value*4096] is read into cache

DECOMPOSING SPECTRE 1

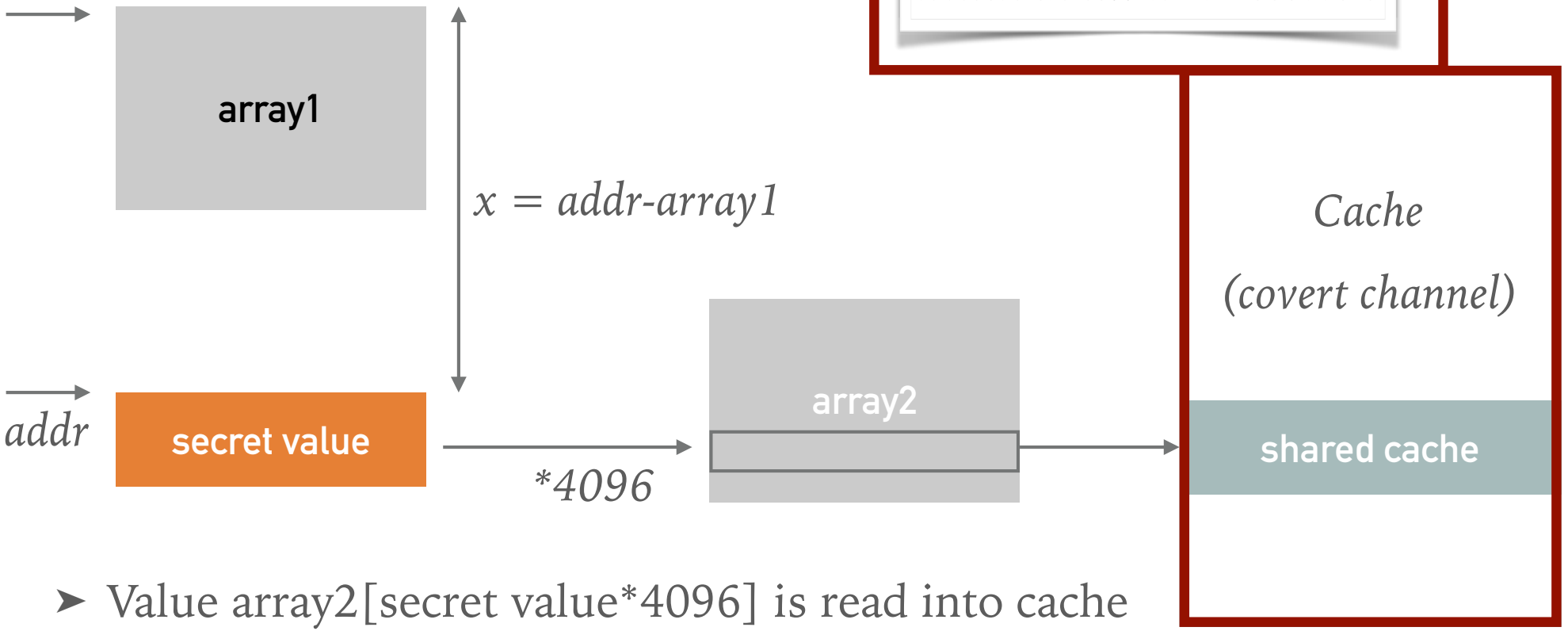
Access Code

```
if (x < array1_size)
  y = array2[array1[x] * 4096];
```

Transmitter Code

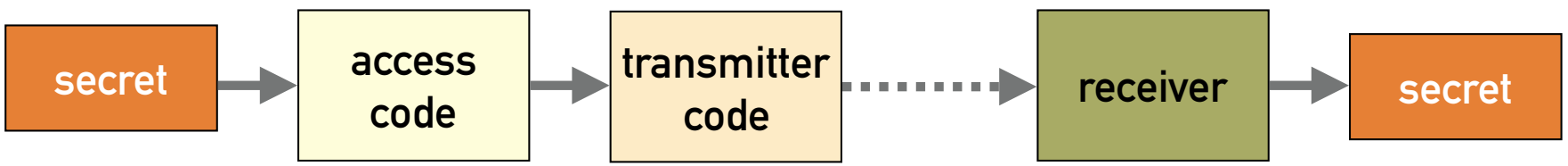
probes cache to detect which index was accessed

Receiver

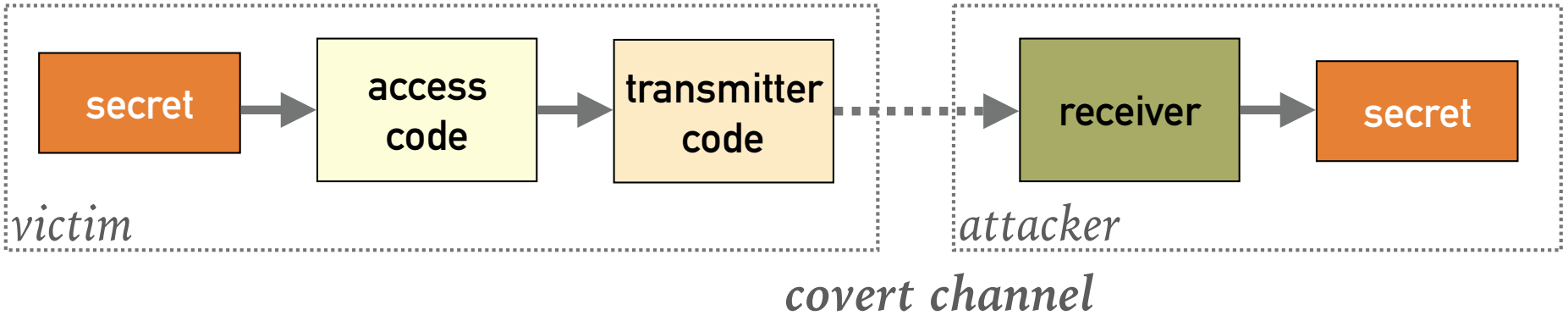


- Value array2[secret value*4096] is read into cache

covert channel



ADDRESSING SIDE CHANNEL ATTACKS?



- Alternatives:
  - disable access code
  - disable transmitter code
  - disable covert channel
  - disable receiver code

Varys

Protecting SGX Enclaves From Practical Side-Channel Attacks

Oleksii Oleksenko, Bohdan Trach
Robert Krahn, Andre Martin, Christof Fetzer

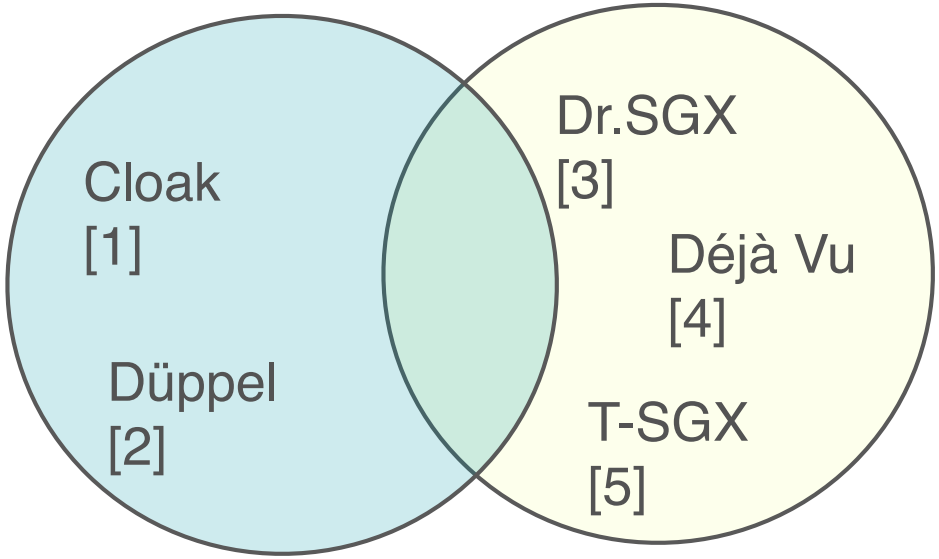
TECHNISCHE UNIVERSITÄT DRESDEN

Mark Silberstein

TECHNION Israel Institute of Technology

Existing solutions

Low overhead



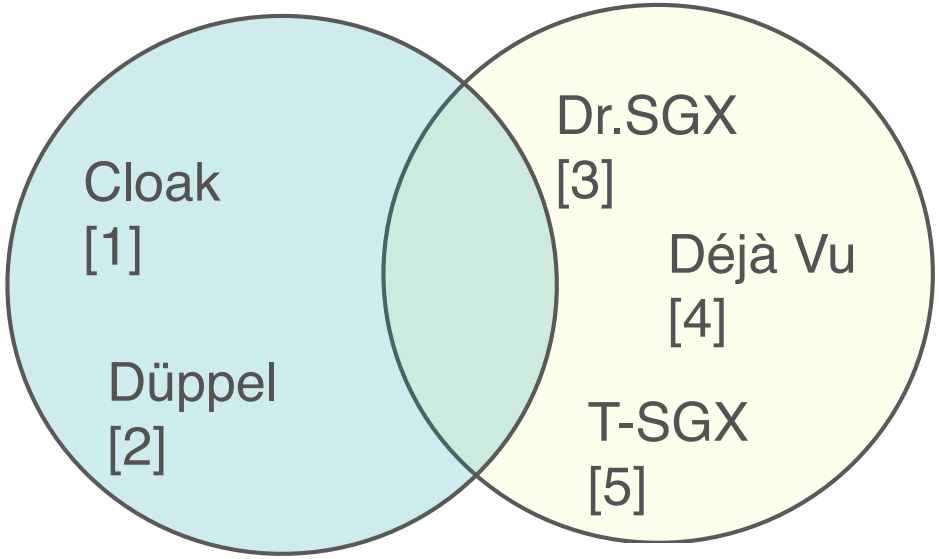
Low effort

(no code changes required)

[1] Gruss, D., Lettner, J., Schuster, F., Ohrimenko, O., Haller, I., & Costa, M. Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory. In *Usenix Security 2017*.
[2] Zhang, Y., Reiter, M. K., Zhang, Y., & Reiter, M. K. Düppel: Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud. *In CCS 2013.*
[3] Brasser, F., Capkun, S., Dmitrienko, A., Frassetto, T., Kostiainen, K., Müller, U., & Sadeghi, A.-R. DR.SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization. In arXiv 2017.
[4] Chen, S., Reiter, M. K., Zhang, X., & Zhang, Y. Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu. In *ASIA CCS '17*.
[5] Shih, M., Lee, S., & Kim, T. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *NDSS 2017*.

Existing solutions

Low overhead



Low effort

(no code changes required)

Varys

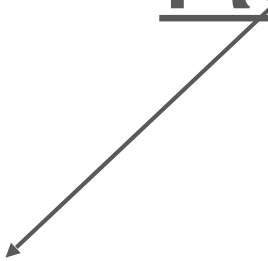
- 15% average slowdown
- No code changes

Approach

Rely but verify

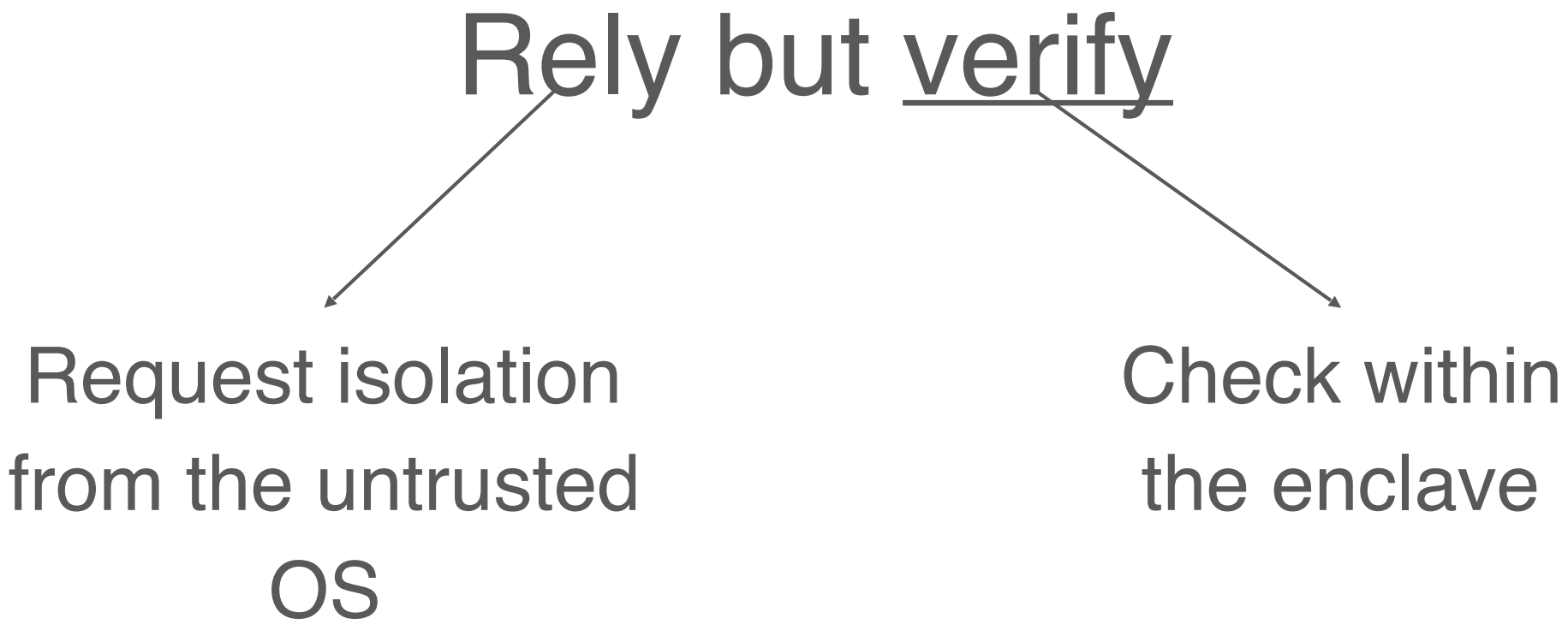
Approach

Rely but verify



Request isolation from the untrusted OS

Approach


Rely but verify
Request isolation from the untrusted OS
Check within the enclave

Complete description

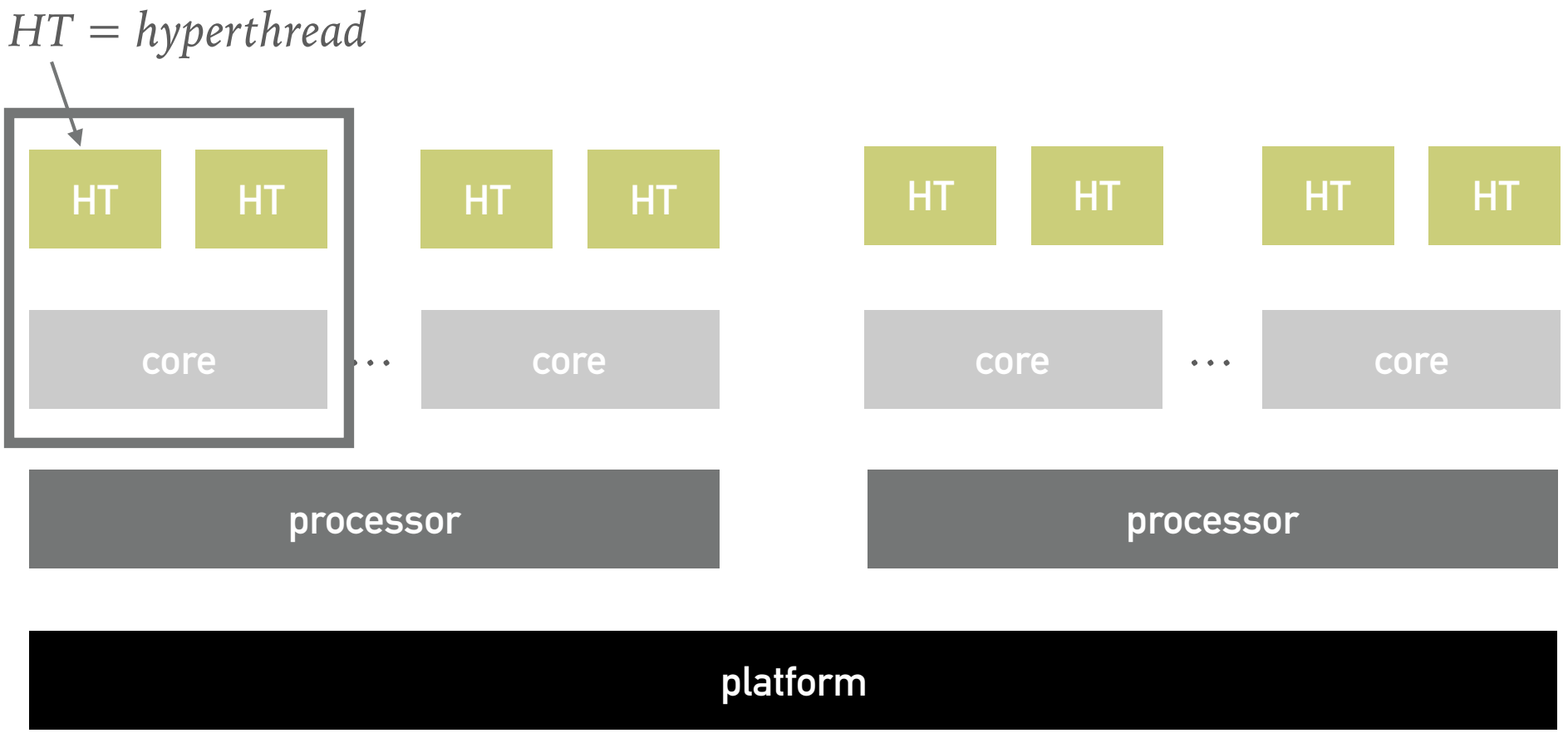
Varys implements a low-cost protection for Intel SGX enclaves against side-channel attacks by creating an isolated environment and verifying it at runtime.

Varys implements a low-cost protection for Intel SGX enclaves against side-channel attacks by creating an isolated environment and verifying it at runtime.

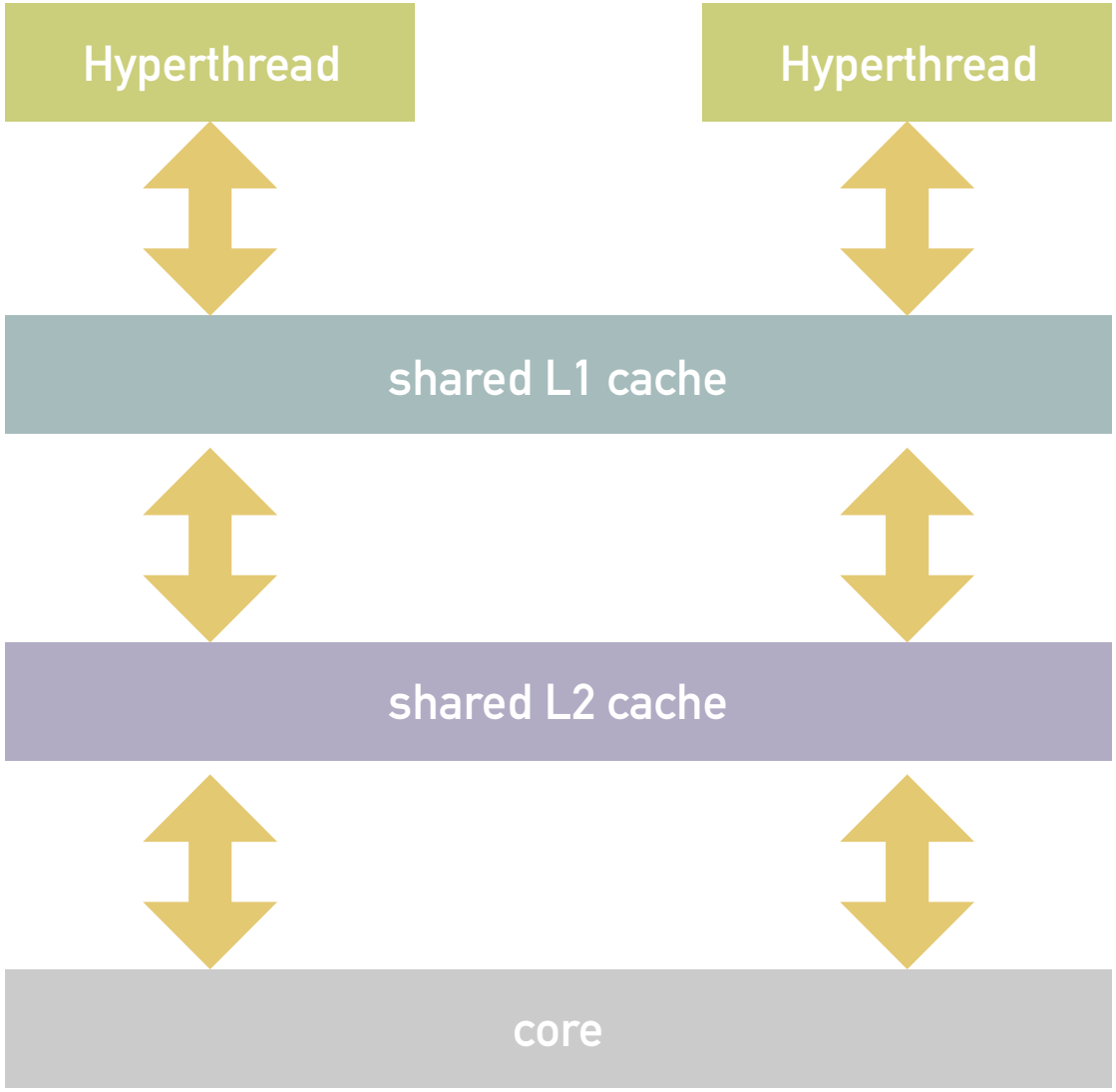
Let’s explains this sentence

Varys implements a low-cost protection for
Intel SGX enclaves against **side-channel attacks**
by creating an isolated environment and
verifying it at runtime.

WHAT SIDE CHANNELS DO WE CONSIDER?



L1 & L2 CACHE: SHARED BETWEEN HYPERTHREADS



L1 cache:

- *instruction cache, data cache*
- *32 KB data cache*
- *32 KB instruction cache*
- *4 cycles*

(skylake)

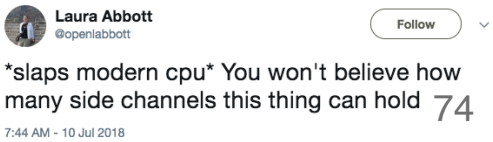
L2 cache (skylake):

- *size = 256KB*
- *64 bytes/cacheline*
- *12 cycles latency*

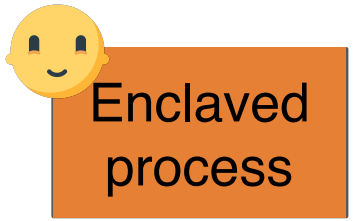
Vulnerable shared resources

- CPU caches (L1, L2)
- Page tables
- FPU
- Memory bus
- ...

} **Varys**

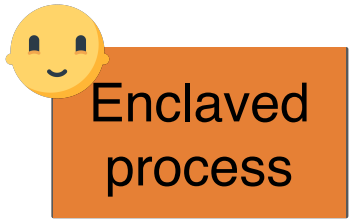


Side-channel attacks



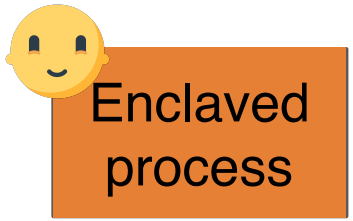
Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



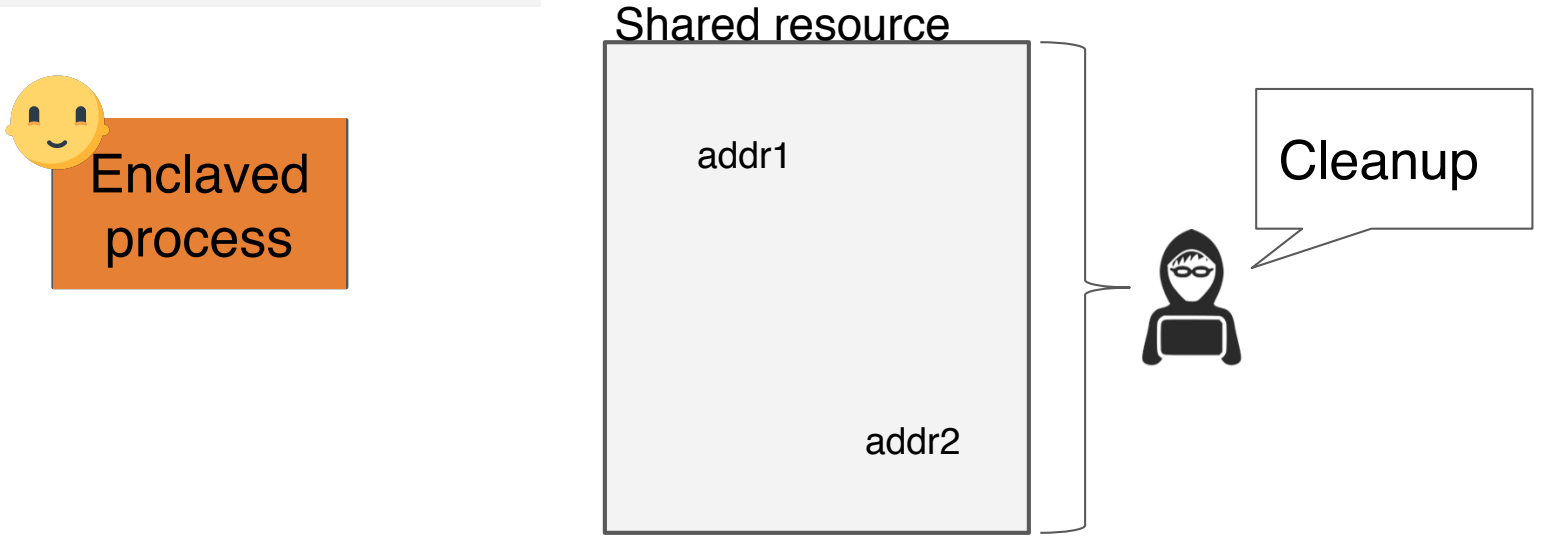
Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



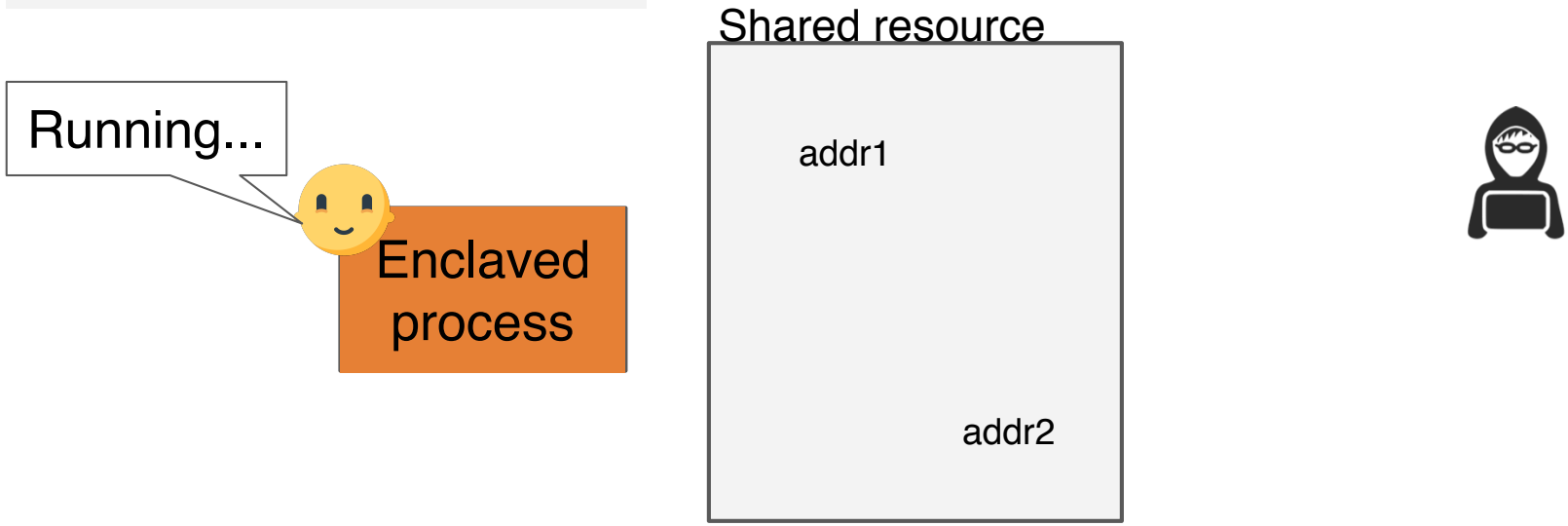
Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



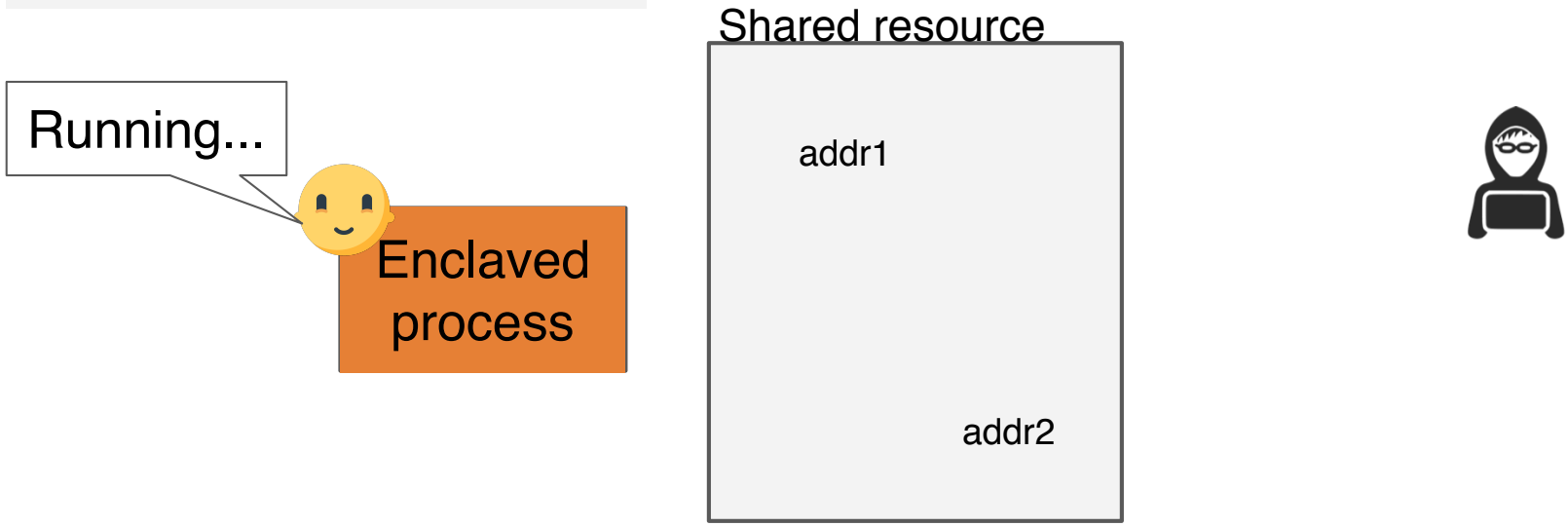
Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



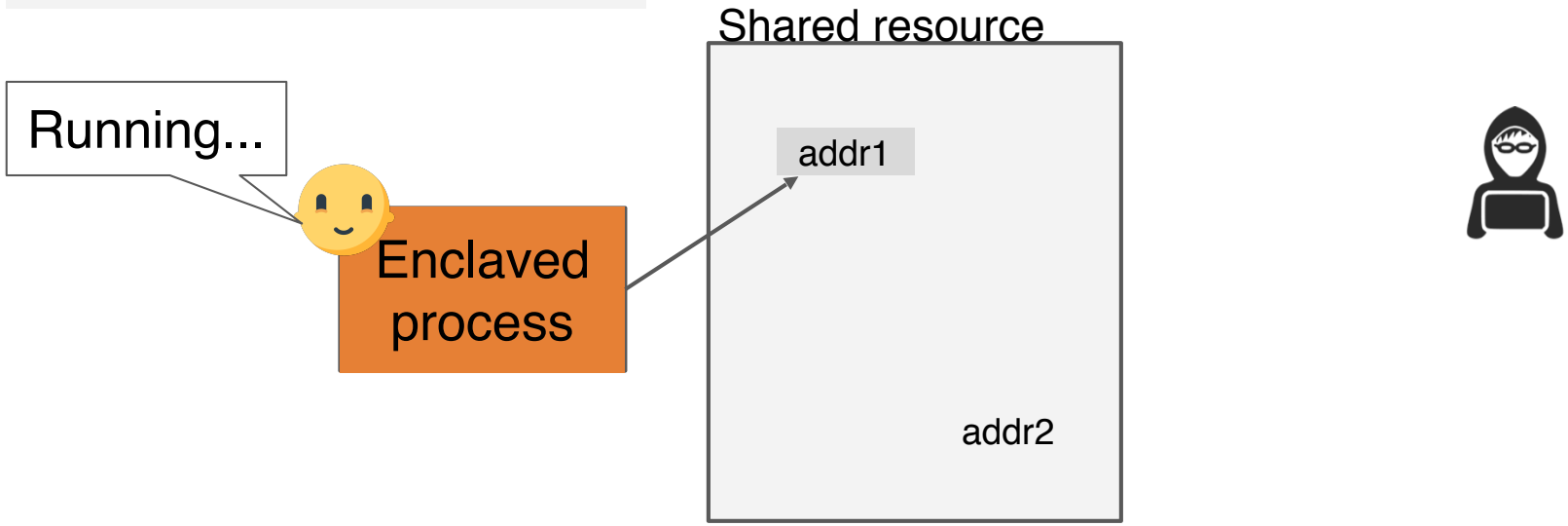
Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



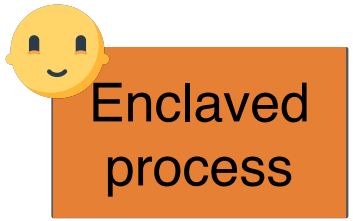
Side-channel attacks

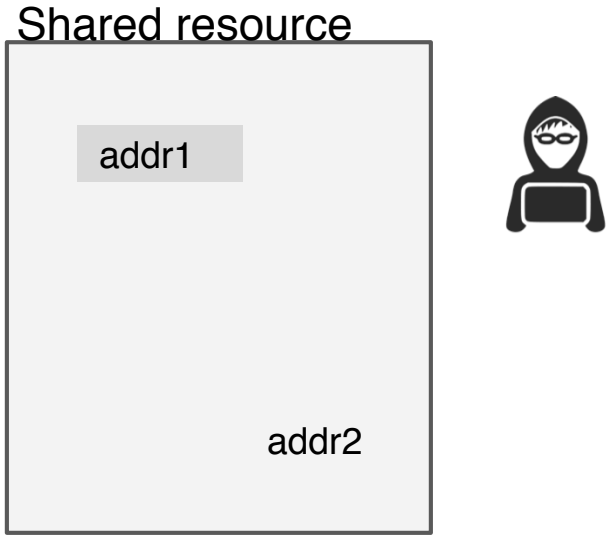
```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



Side-channel attacks

```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



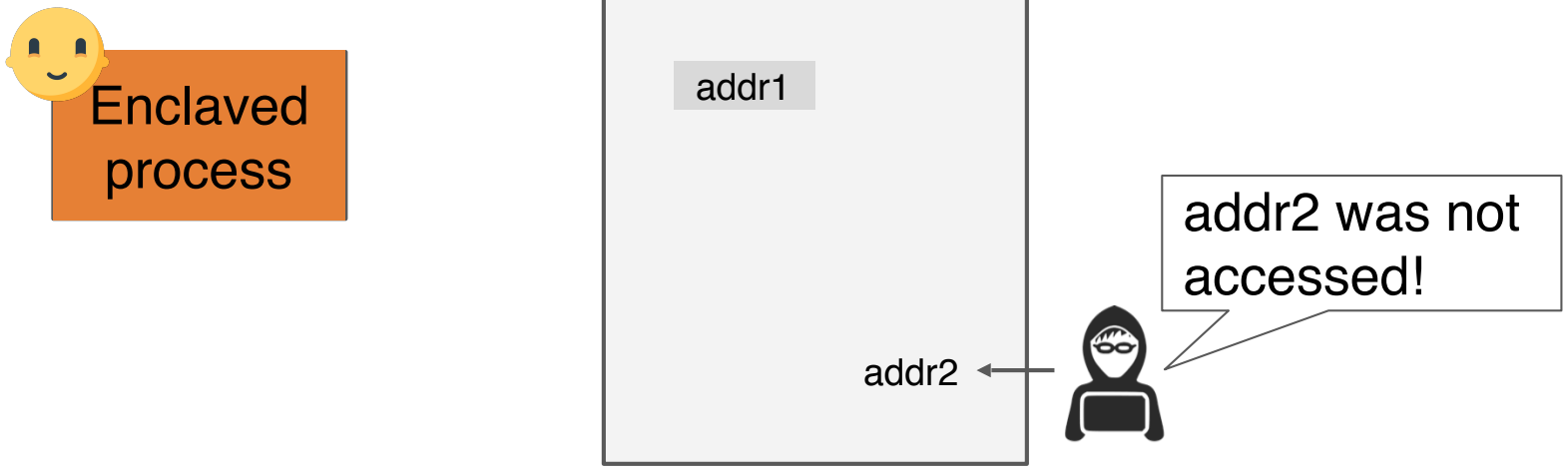


Side-channel attacks

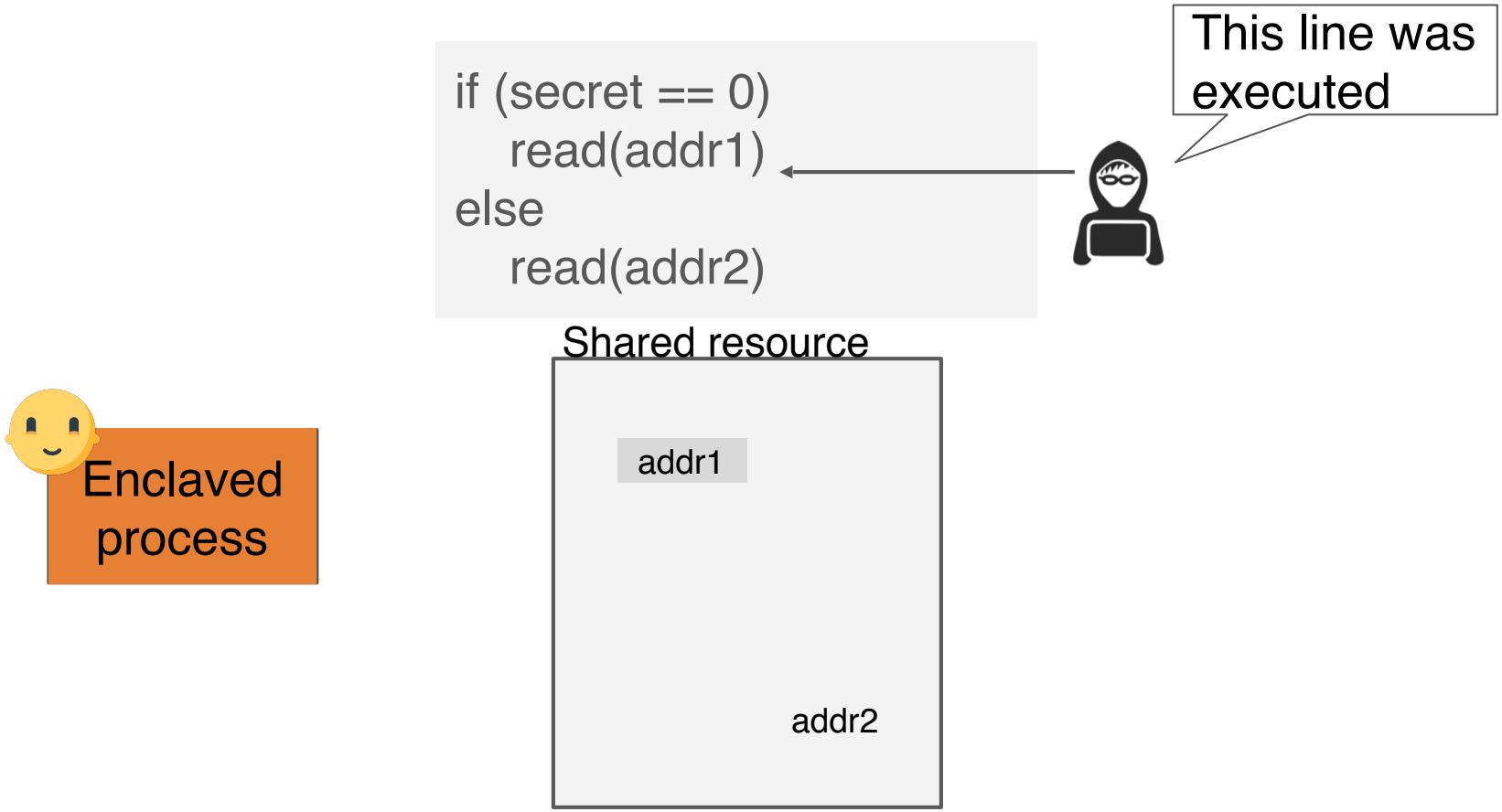
```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```

Side-channel attacks

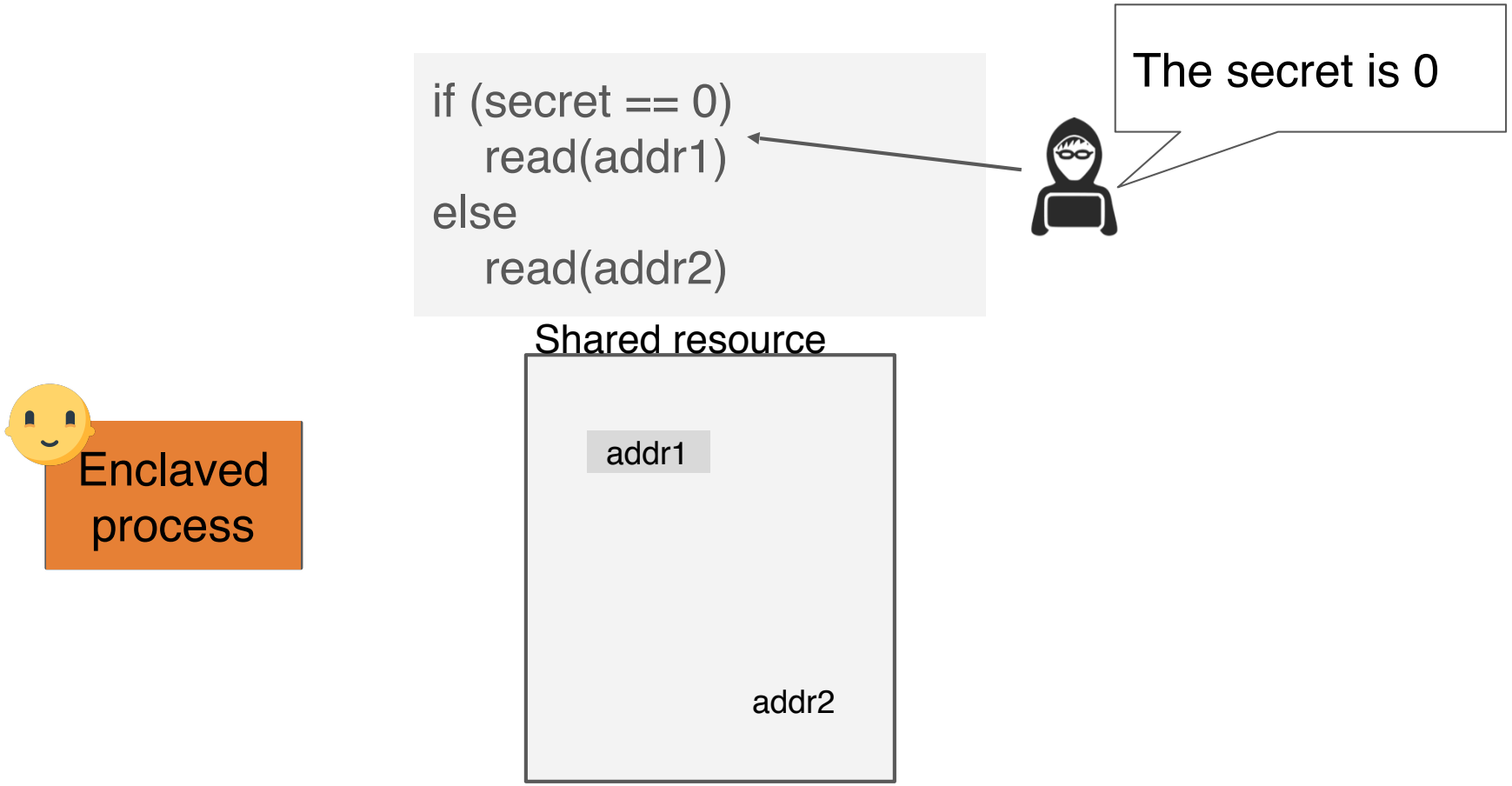
```
if (secret == 0)
    read(addr1)
else
    read(addr2)
```



Side-channel attacks



Side-channel attacks



Varys implements a low-cost protection for Intel SGX enclaves against side-channel attacks by creating an **isolated environment** and verifying it at runtime.

Attack requirements

- High interrupt rate
- Predefined cache state
- Shared core

Attack requirements

- ~~High interrupt rate~~
- ~~Predefined cache state~~
- ~~Shared core~~

Isolated environment

Varys implements a low-cost protection for Intel SGX enclaves against side-channel attacks by **creating** an isolated environment and **verifying it at runtime**.

Design

- High preemption rate
- Predefined cache state
- Shared core

Restrict and terminate

Cache eviction

Trusted reservation

Design

- High preemption rate
- Predefined cache state
- Shared core

Restrict and terminate

Cache eviction

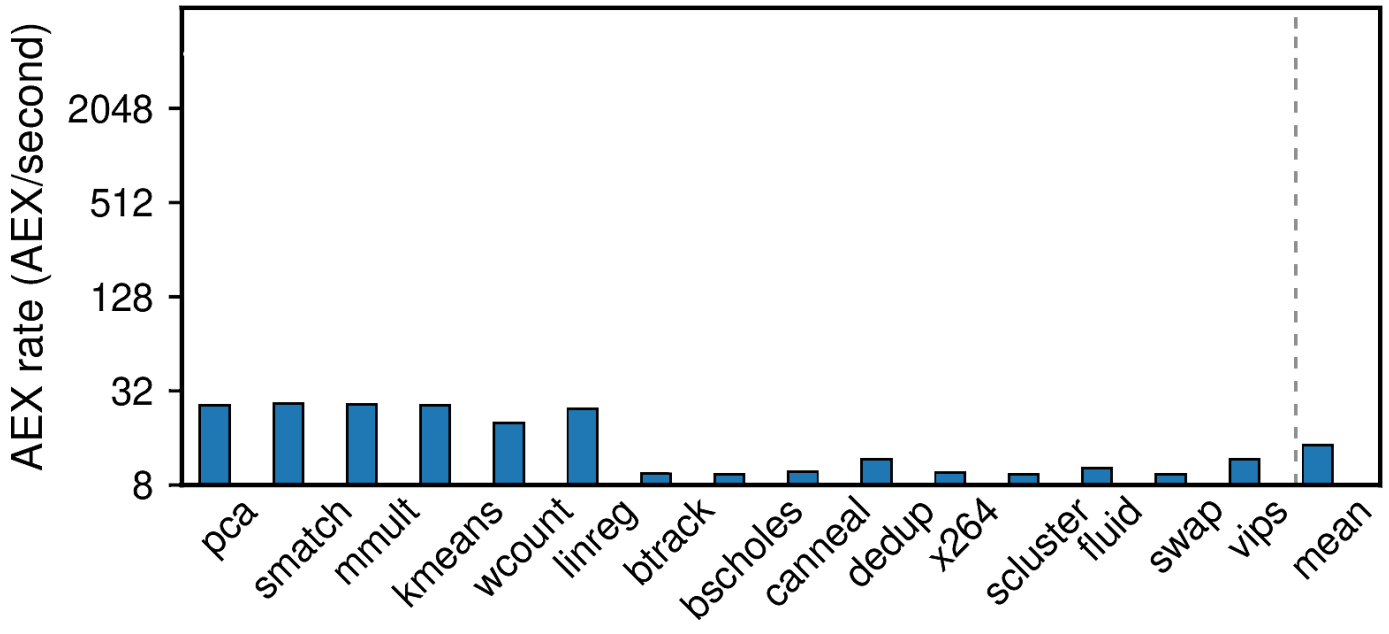
Trusted reservation

Restricting preemption rate

- Attack exit rate: ~ 5000 exits/s.

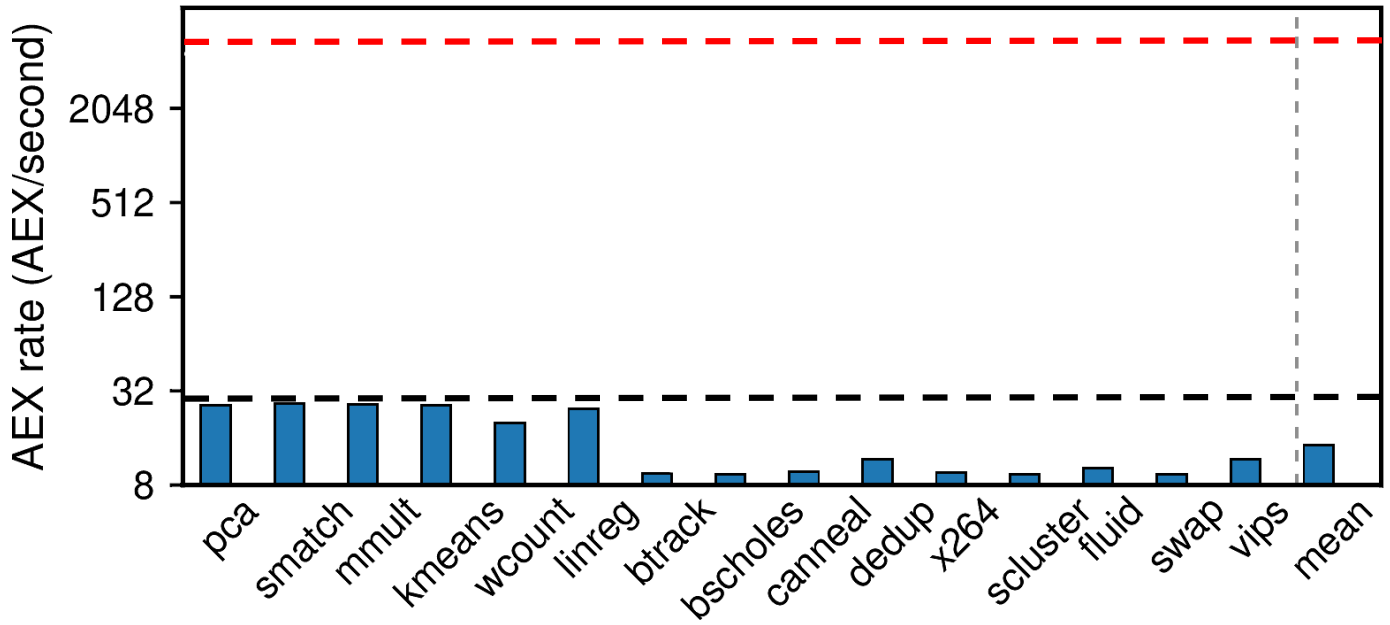
Restricting preemption rate

- Attack exit rate: ~ 5000 exits/s.
- Normal exit rate: ~ 30 exits/s.

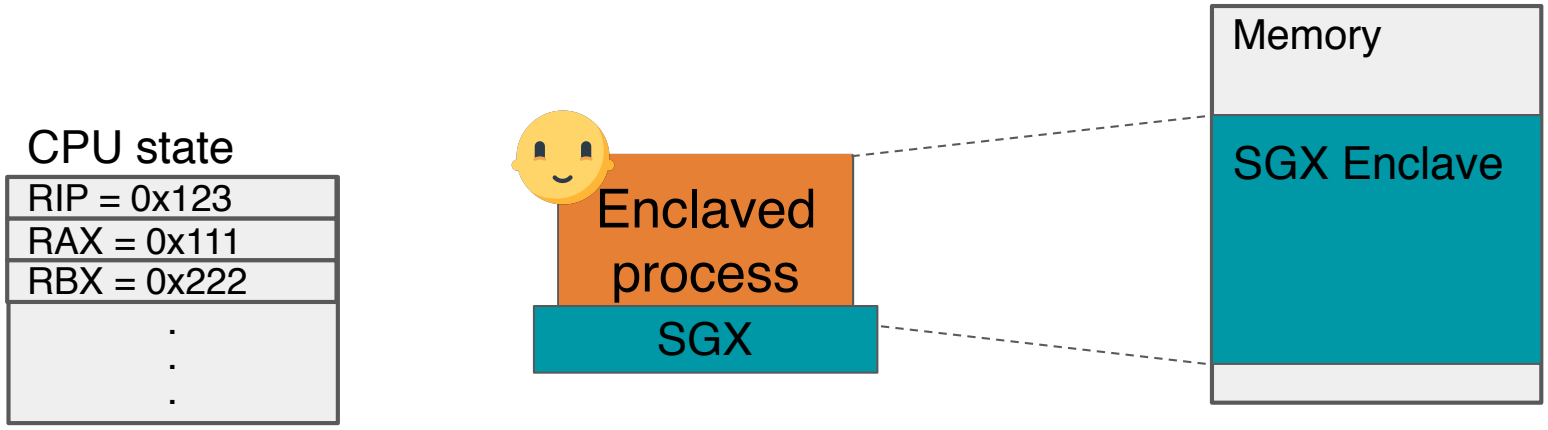


Restricting preemption rate

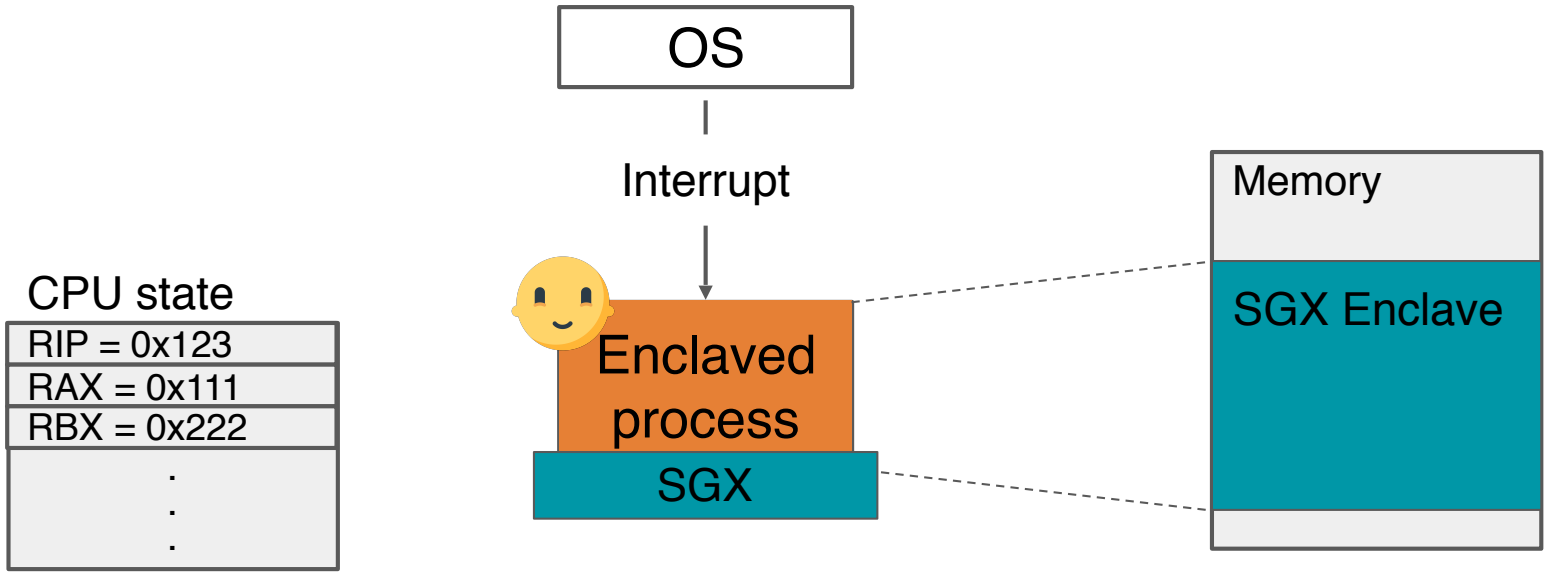
- Attack exit rate: ~ 5000 exits/s.
- Normal exit rate: ~ 30 exits/s.



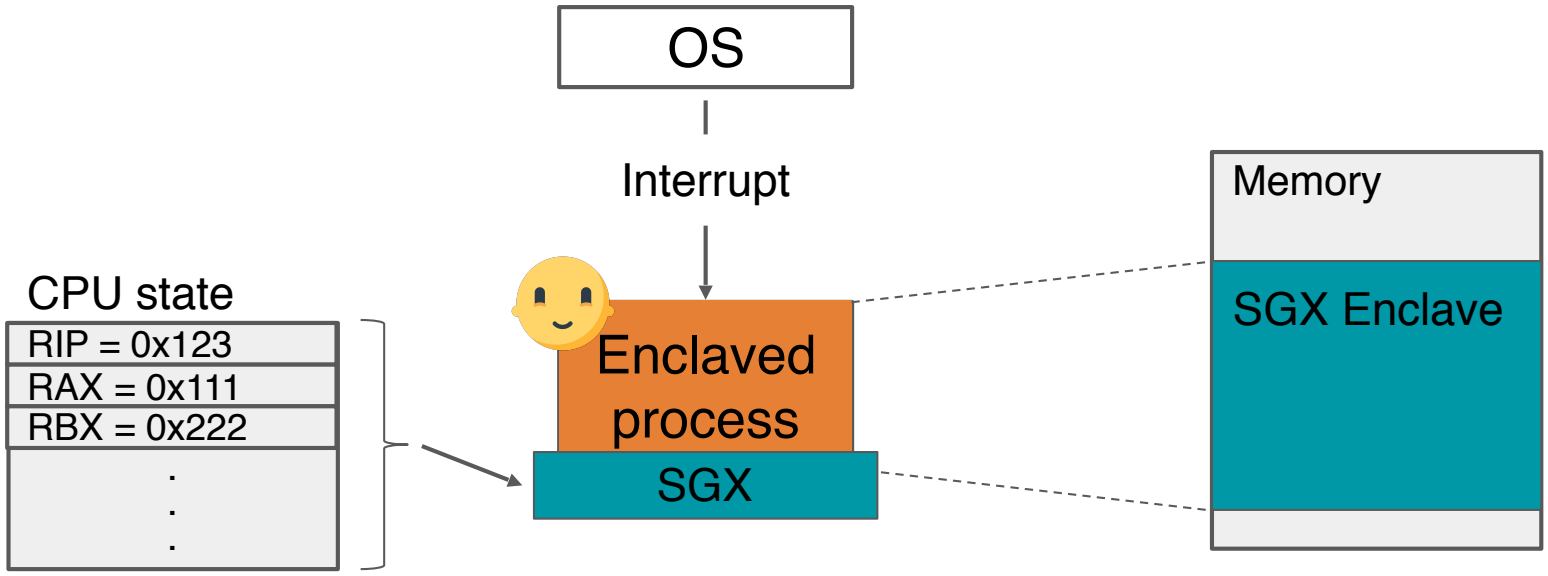
Asynchronous Enclave Exit (AEX)



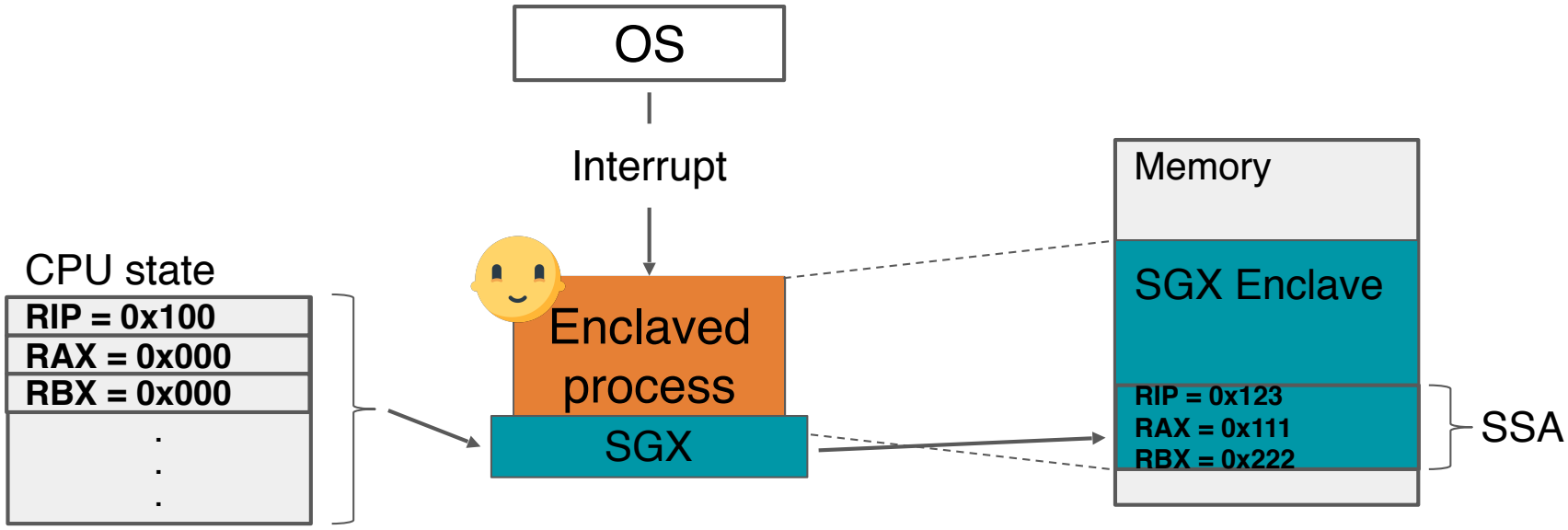
Asynchronous Enclave Exit (AEX)



Asynchronous Enclave Exit (AEX)

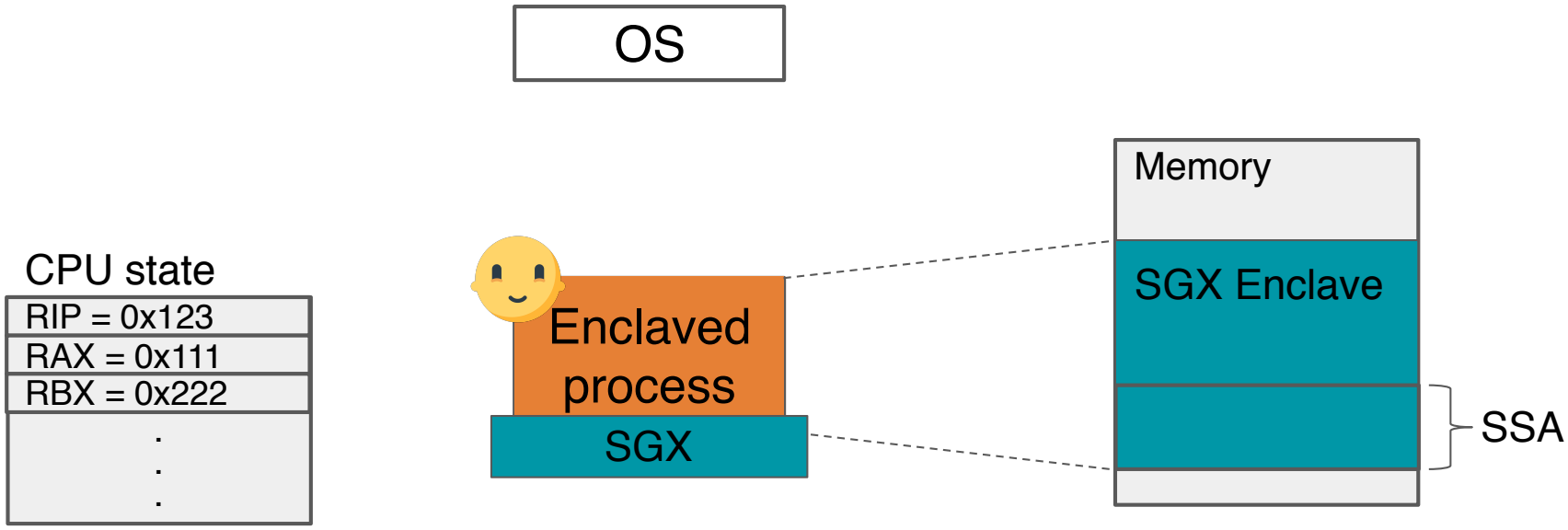


Asynchronous Enclave Exit (AEX)

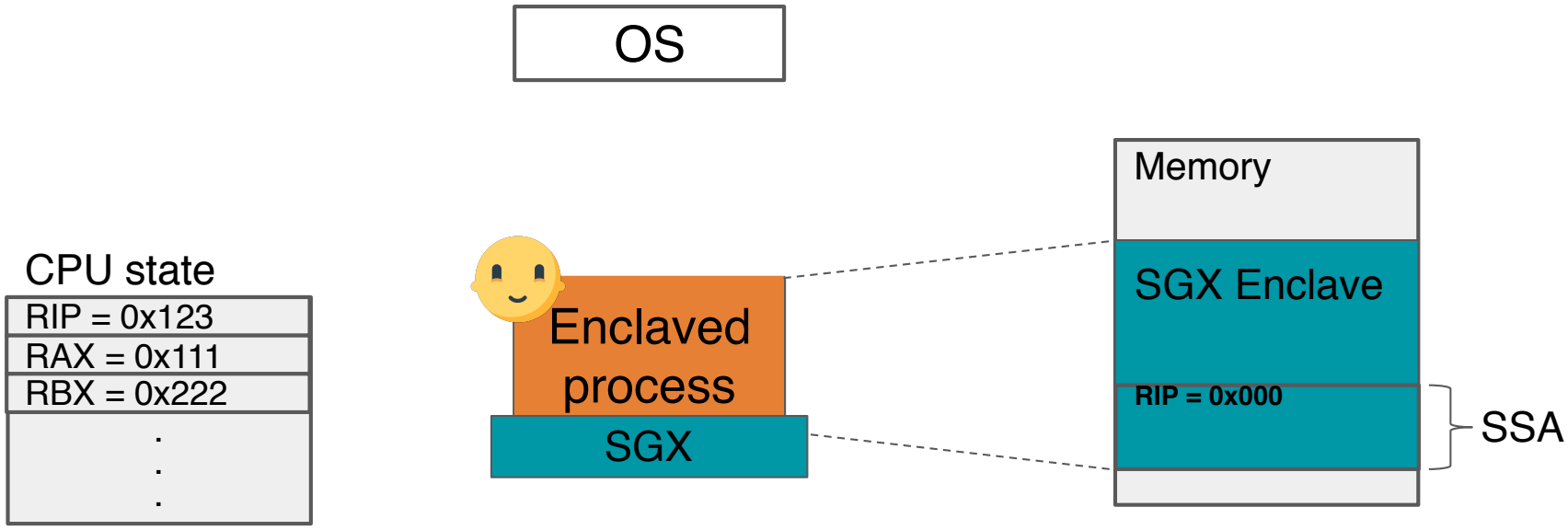


$SSA = State\ Save\ Area$

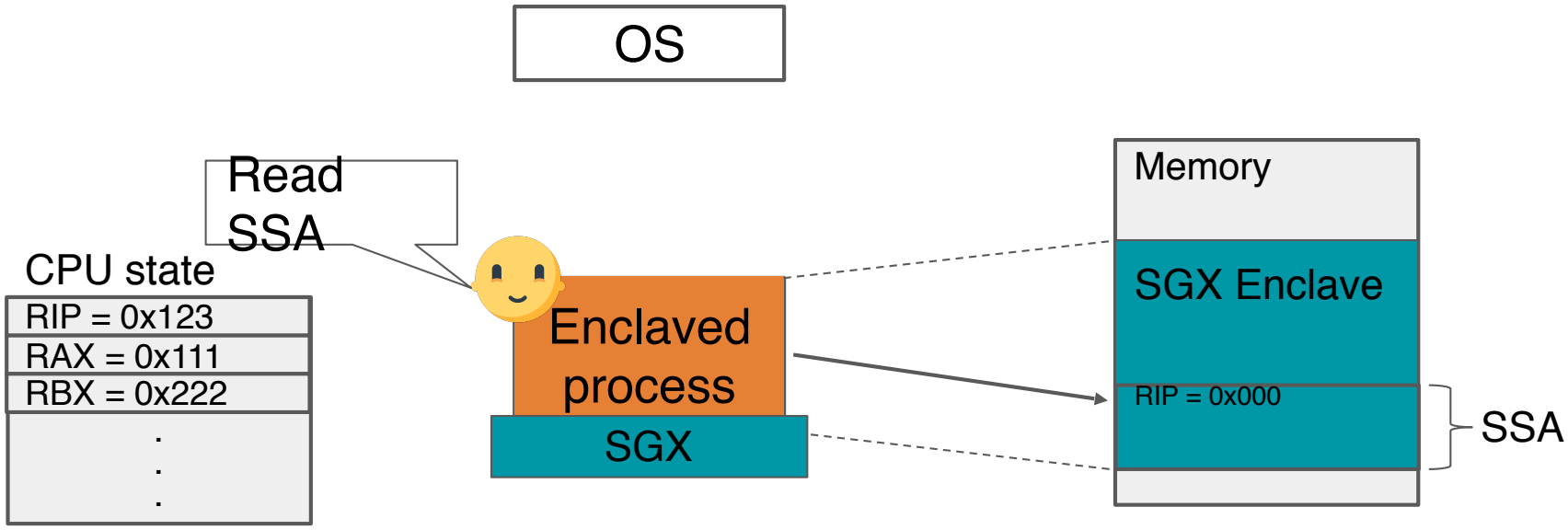
Detecting interrupts



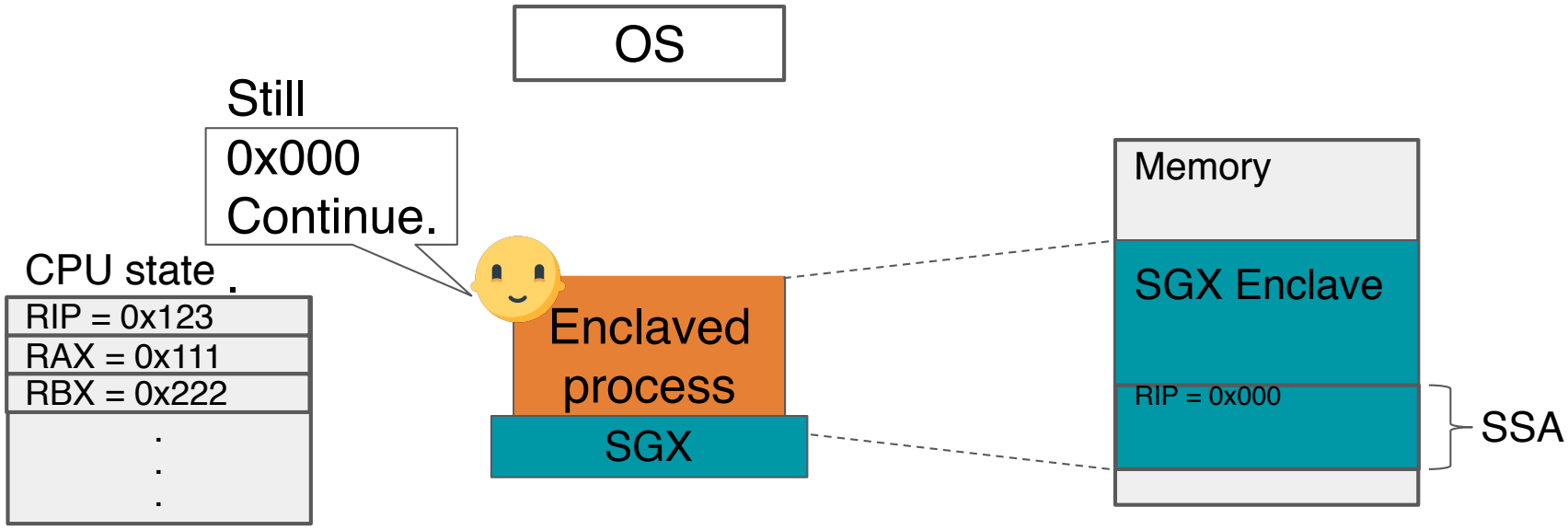
Detecting interrupts



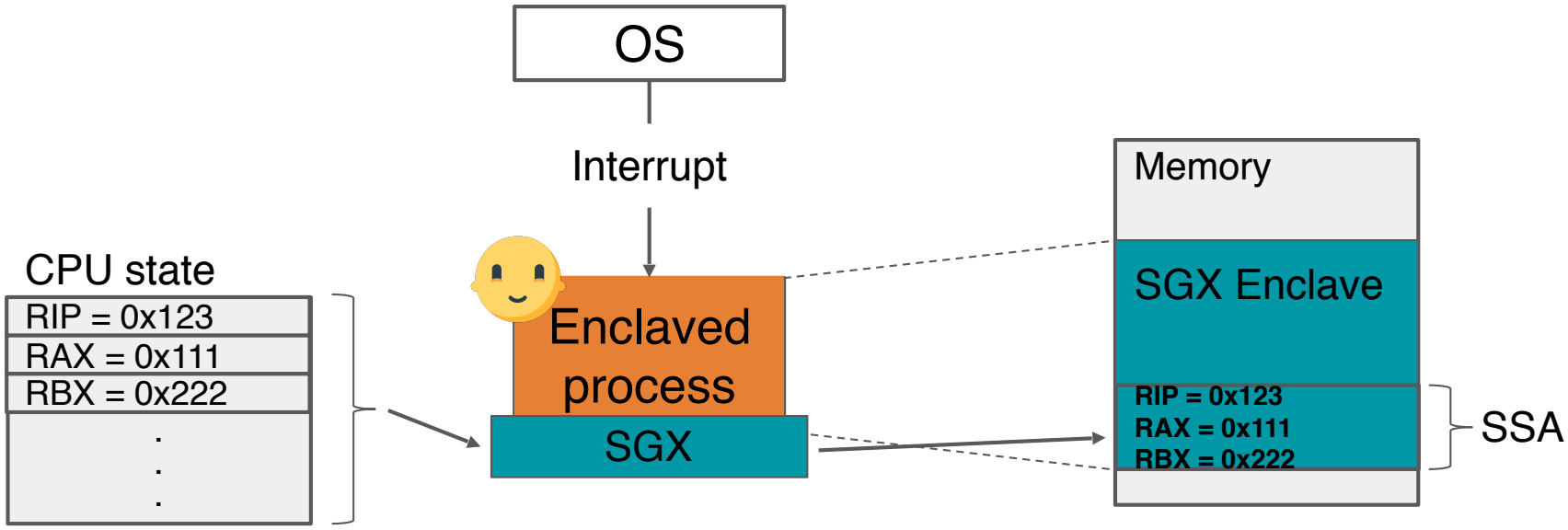
Detecting interrupts



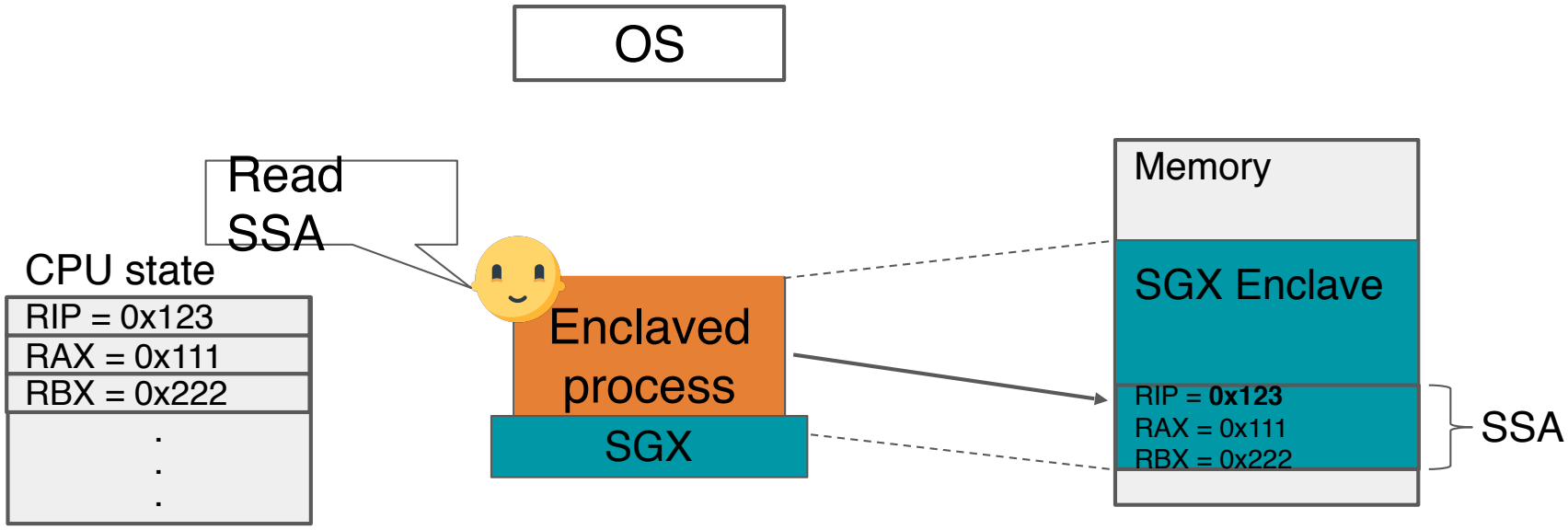
Detecting interrupts



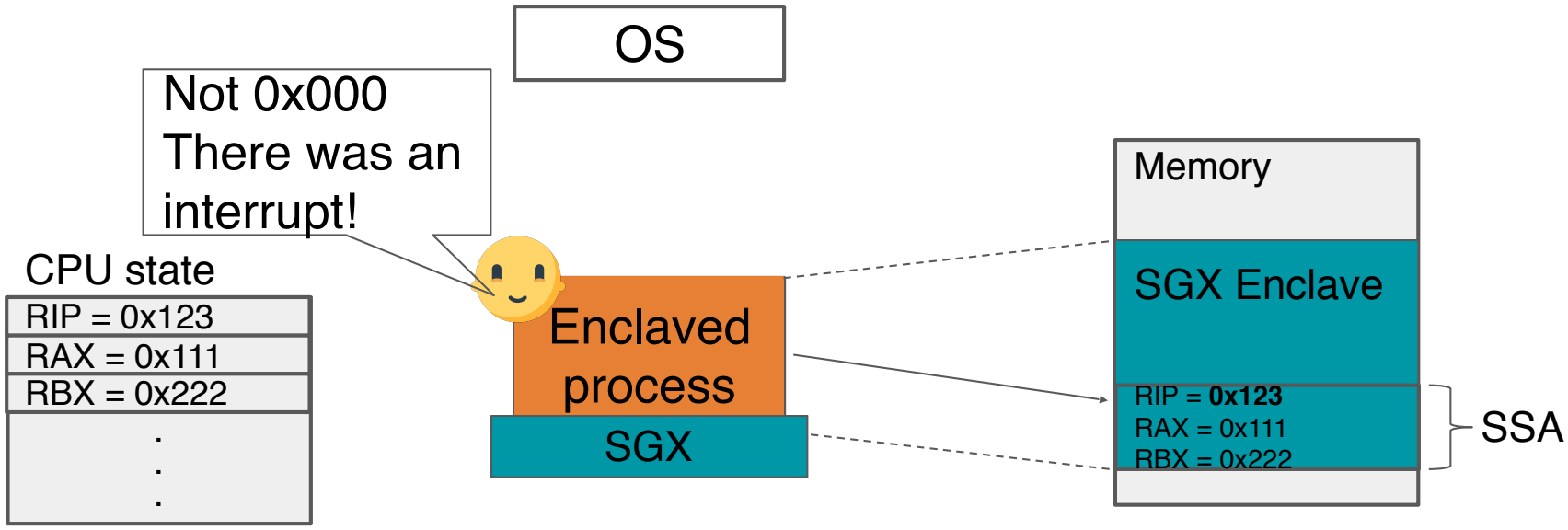
Detecting interrupts



Detecting interrupts



Detecting interrupts



Design

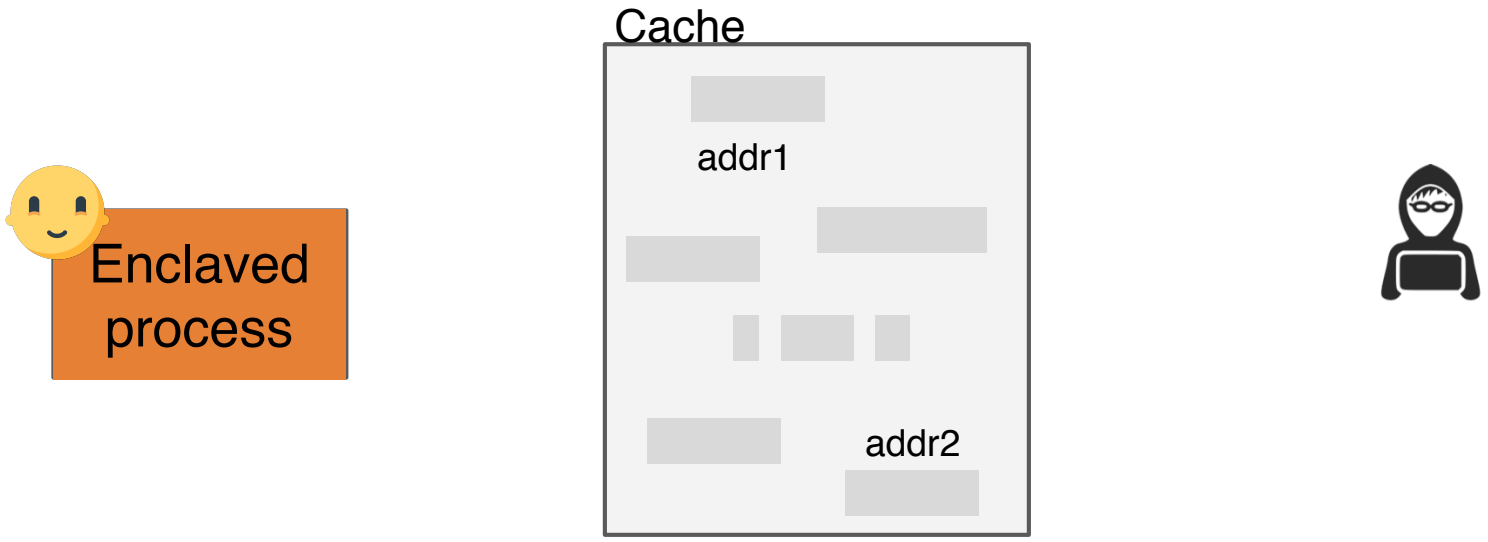
- High preemption rate
- Predefined cache state
- Shared core

Restrict and terminate

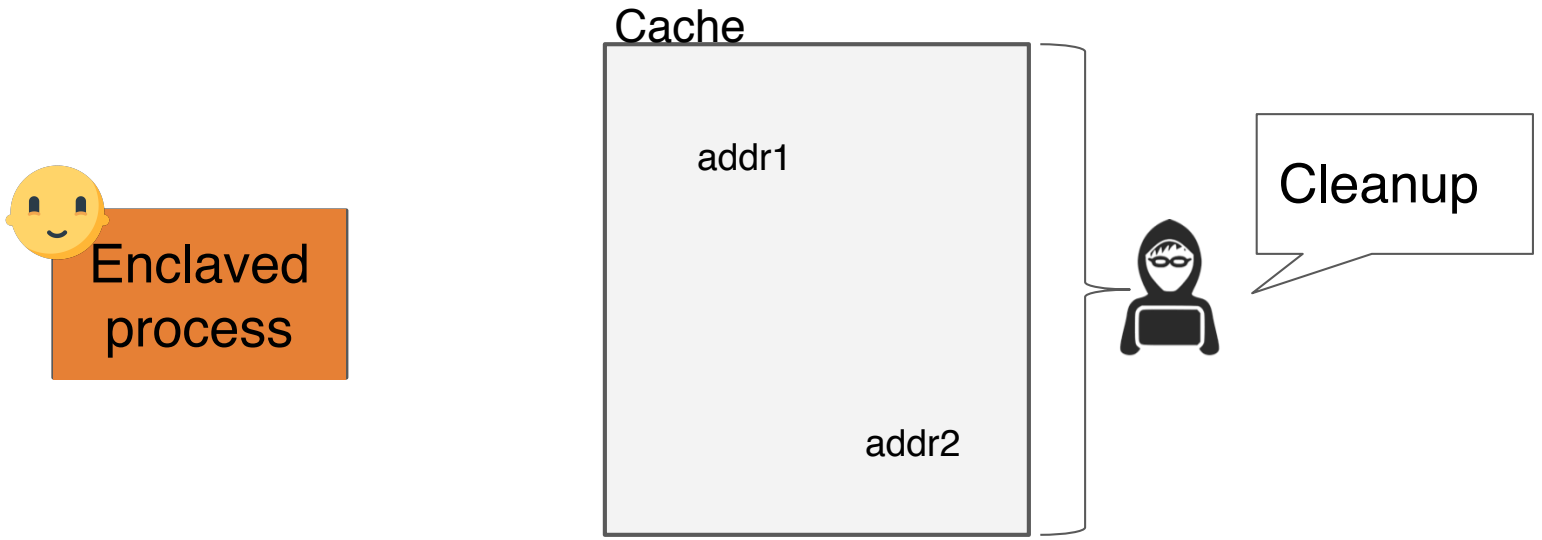
Cache eviction

Trusted reservation

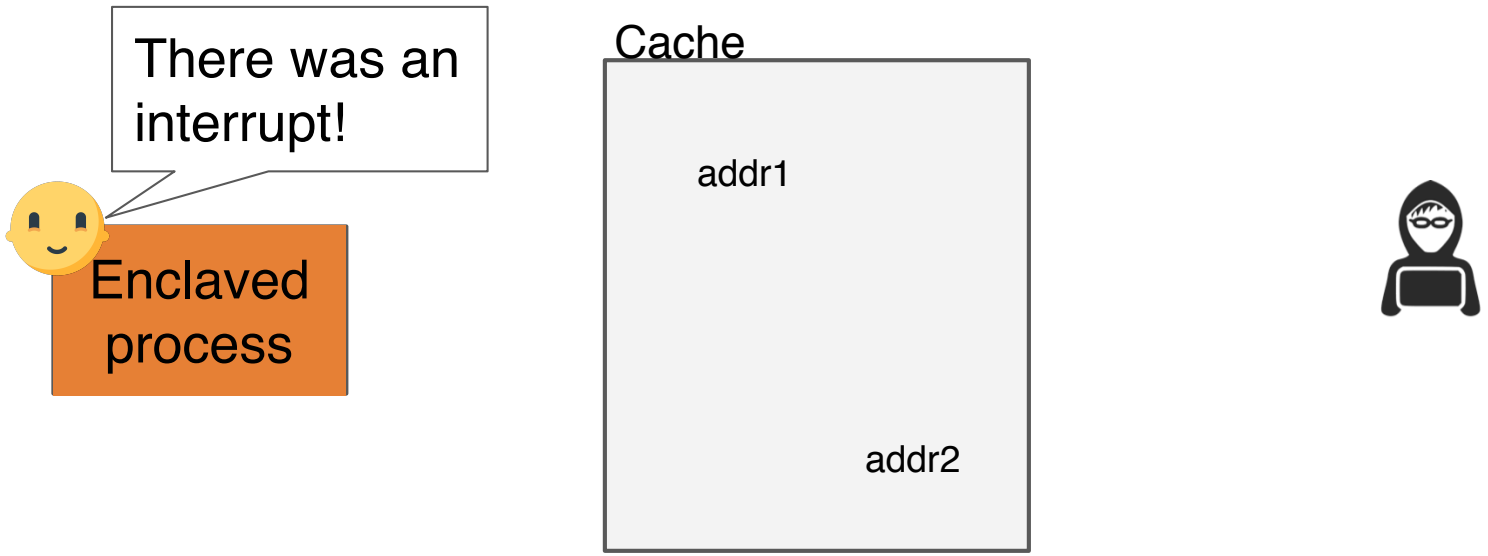
Hiding cache traces



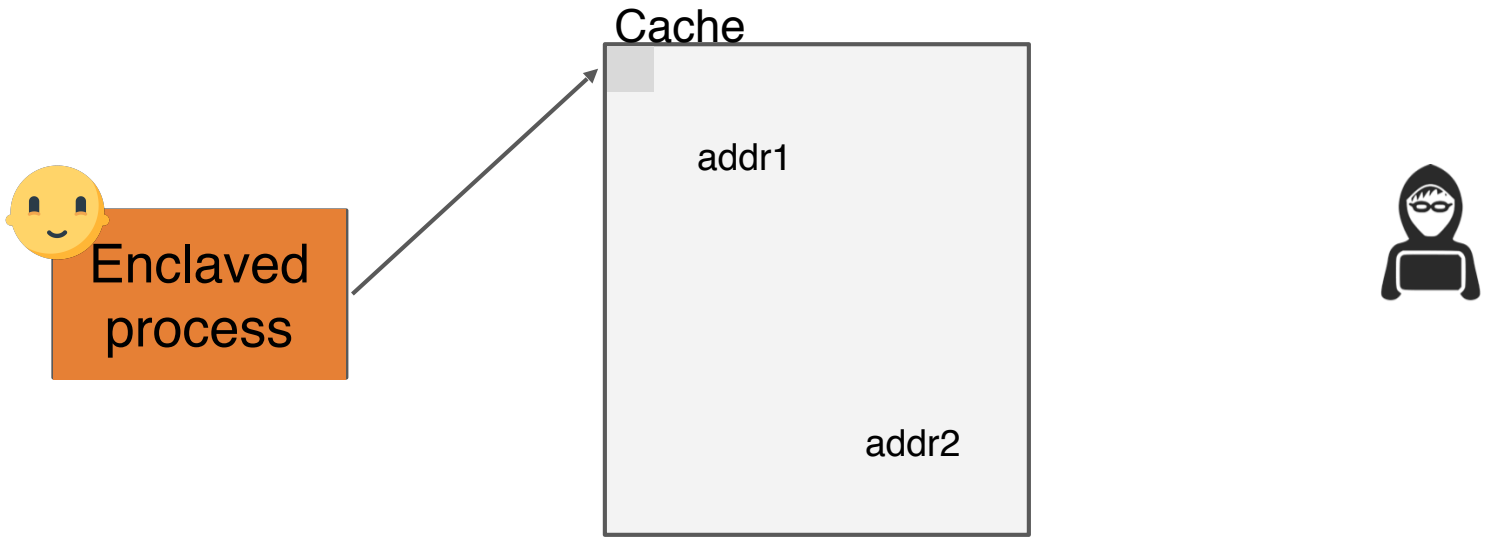
Hiding cache traces



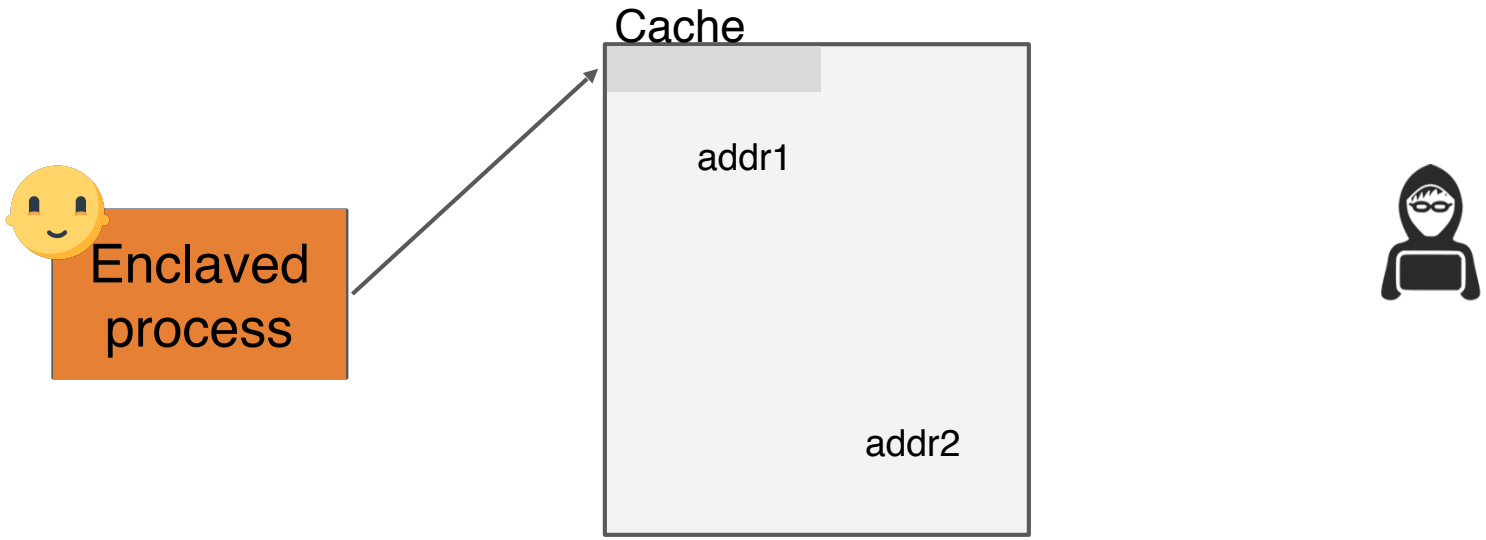
Hiding cache traces



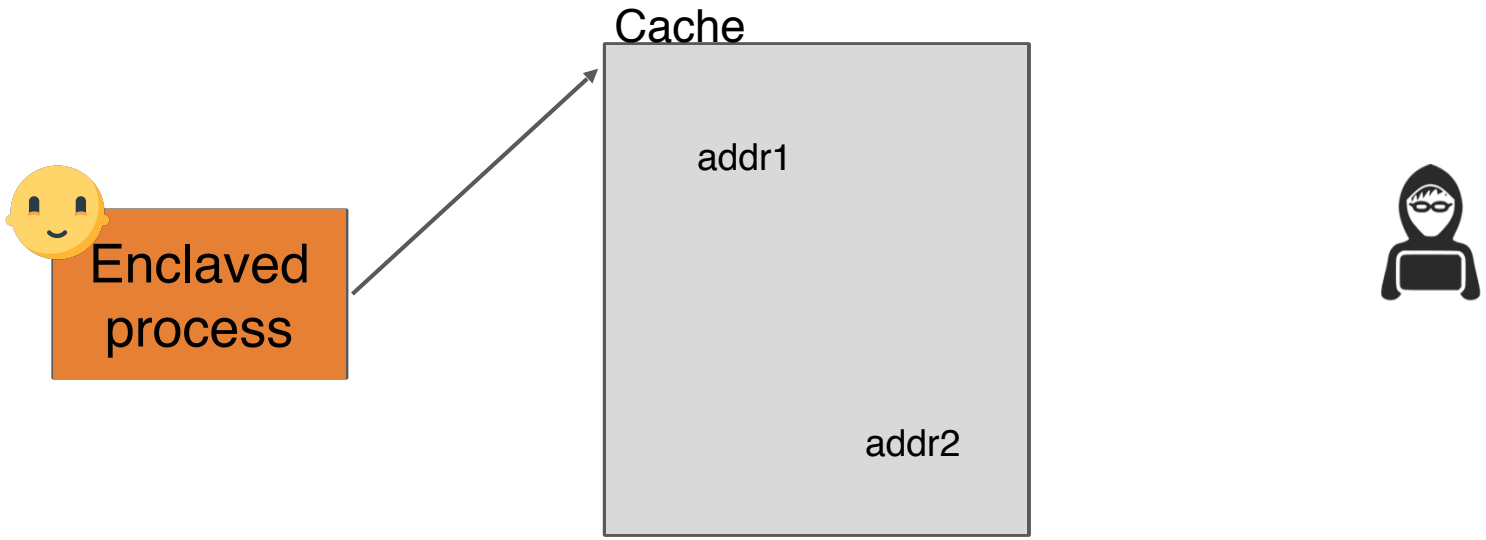
Hiding cache traces



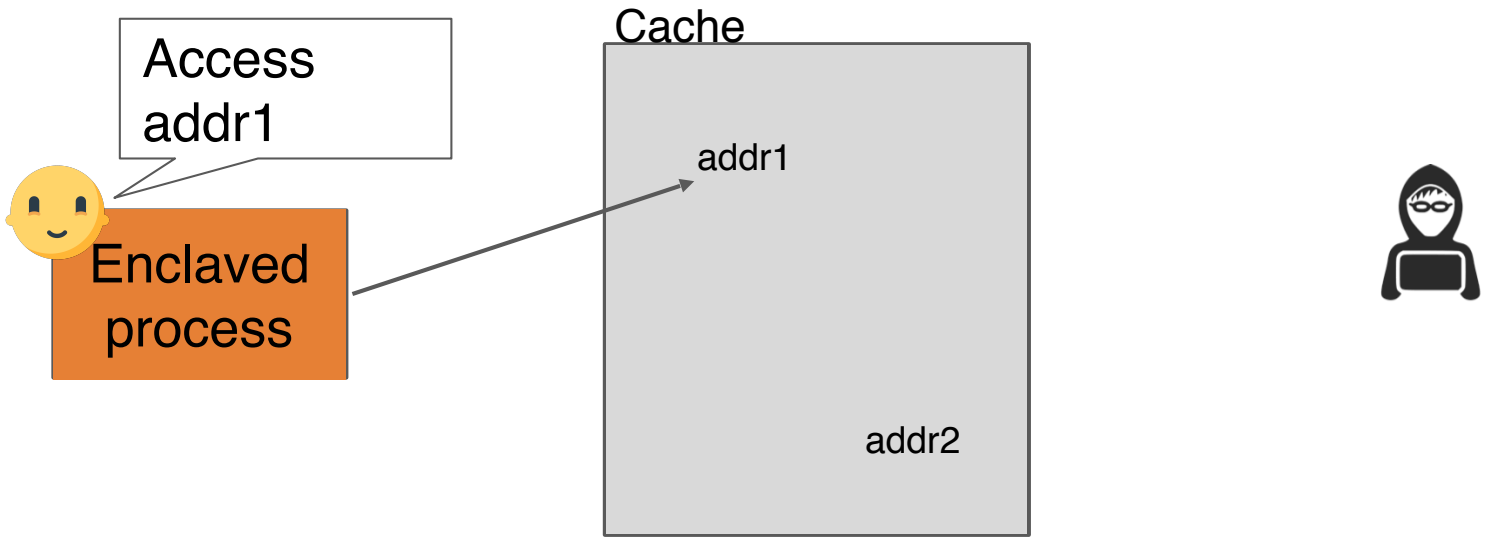
Hiding cache traces



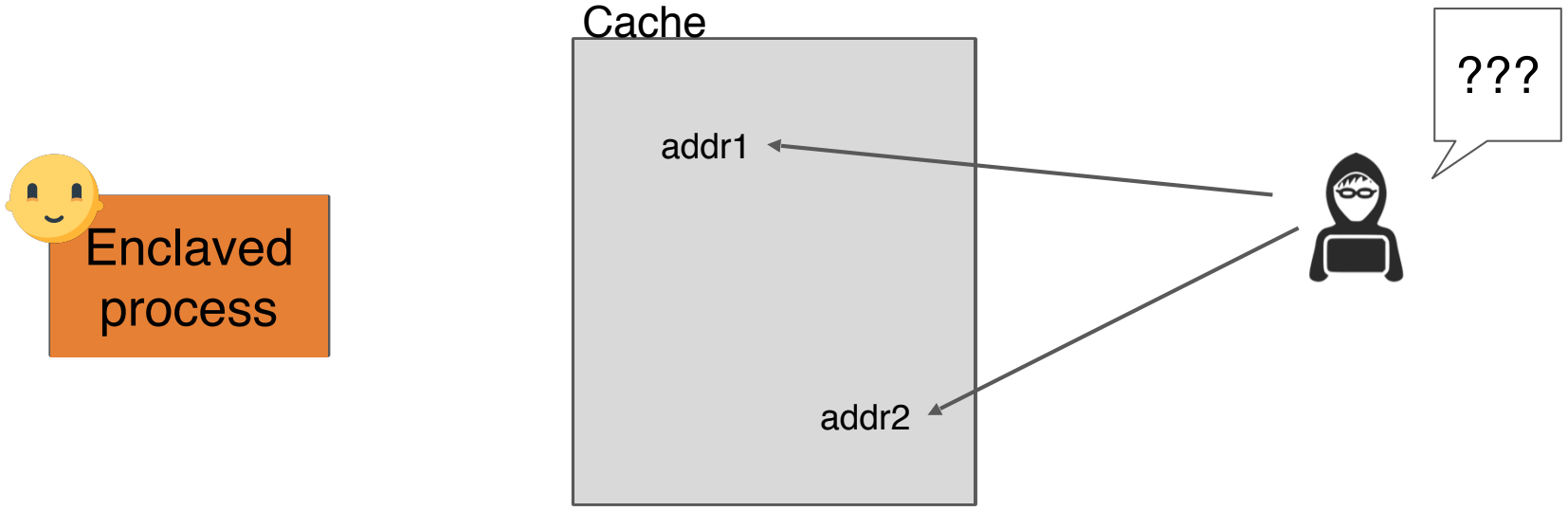
Hiding cache traces



Hiding cache traces



Hiding cache traces



Design

- High preemption rate
- Predefined cache state
- Shared core



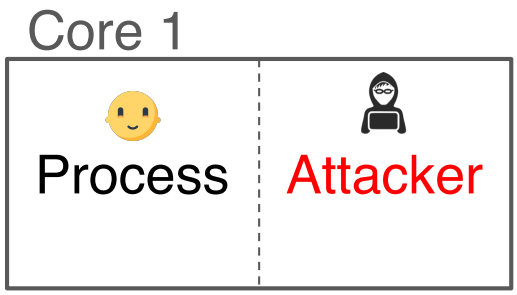
Restrict and terminate

Cache eviction

Trusted reservation

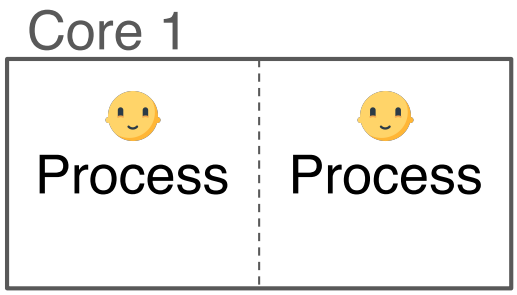
Preventing core sharing

- Occupy both hyperthreads

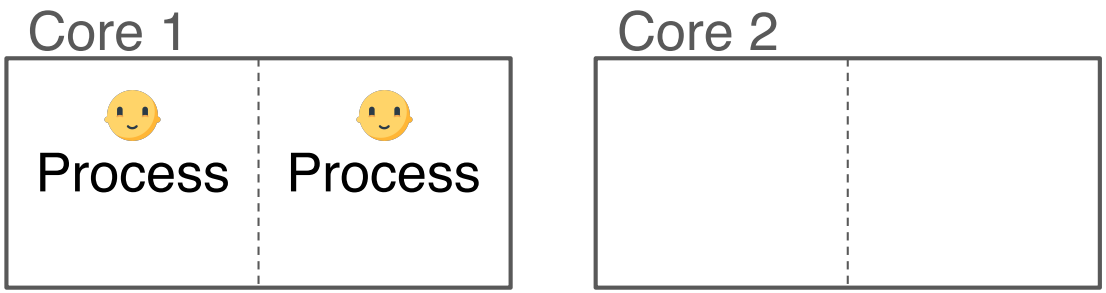


Preventing core sharing

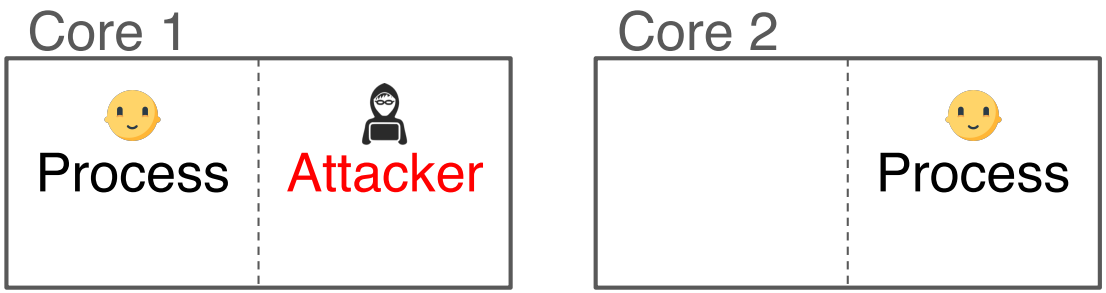
- Occupy both hyperthreads
  - Use process affinity



How do we ensure reservation?

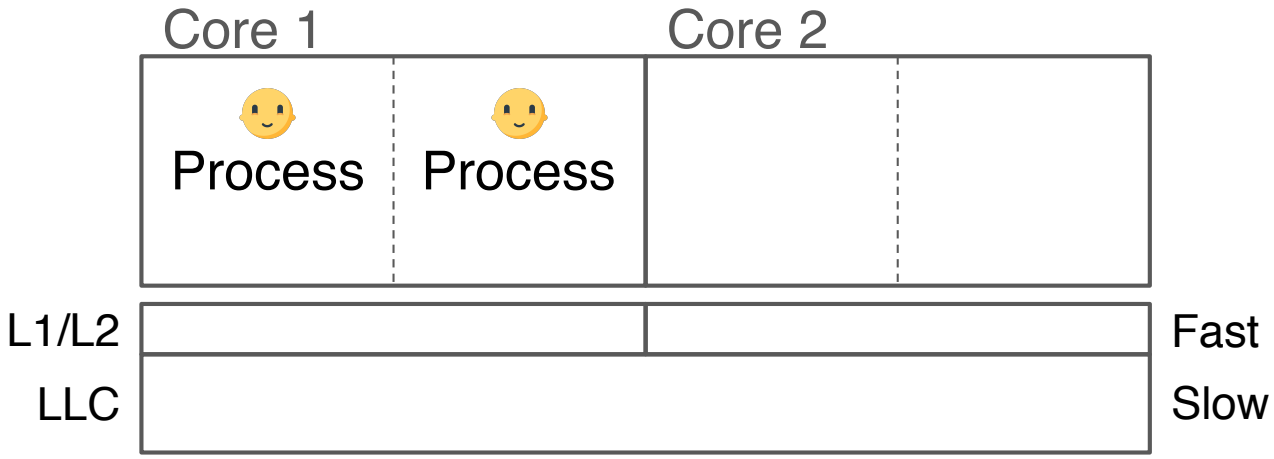


How do we ensure reservation?



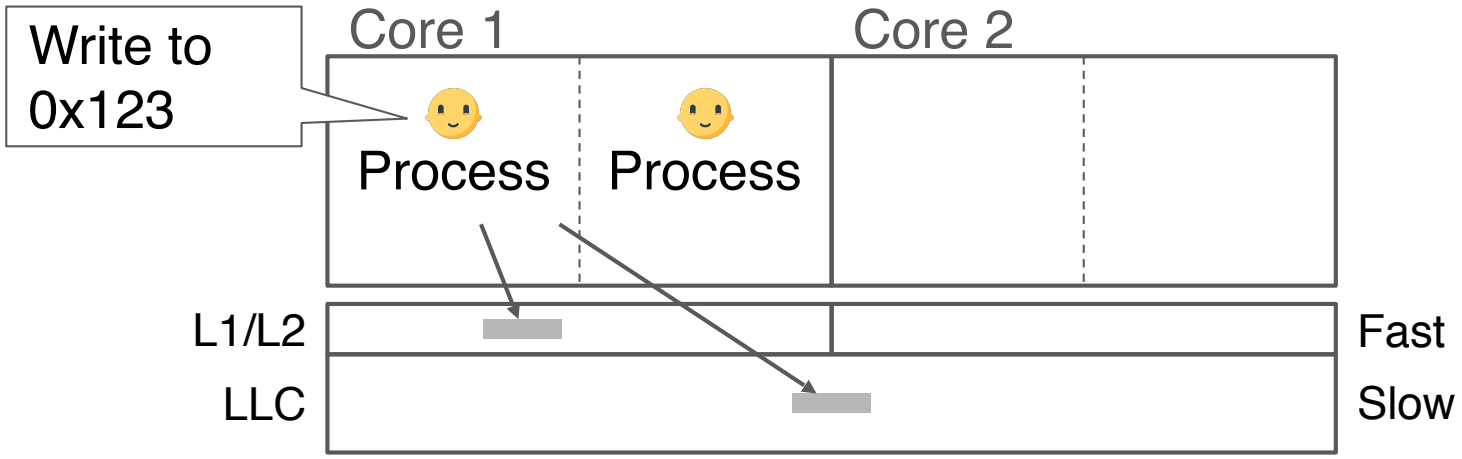
Handshake

- Use shared access timing



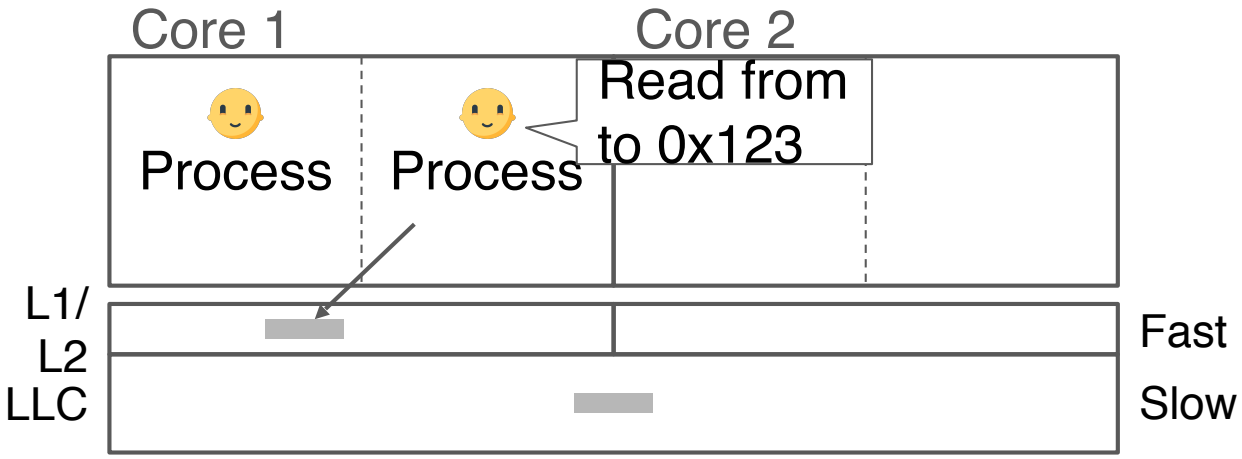
Handshake

- Use shared access timing



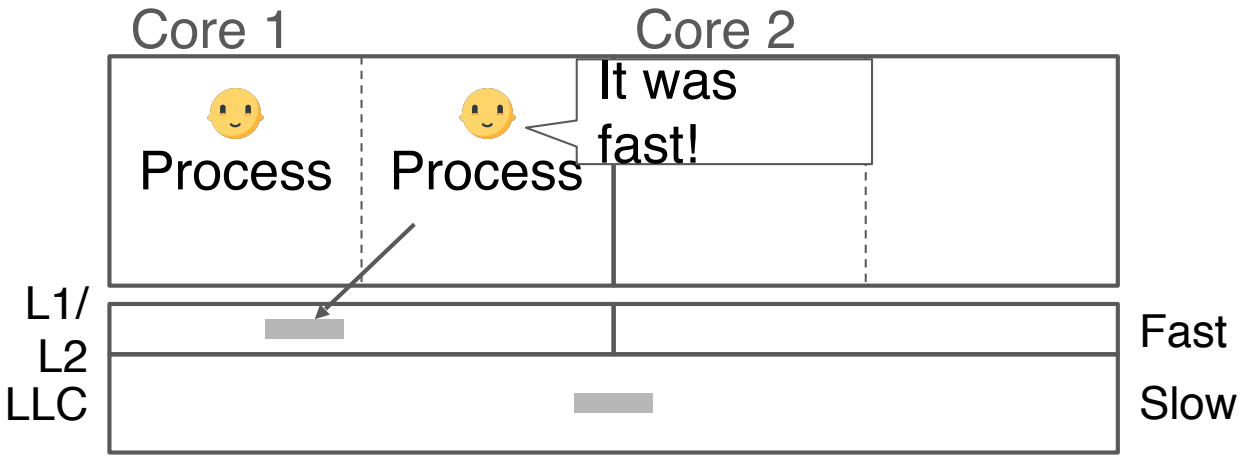
Handshake

- Use shared access timing



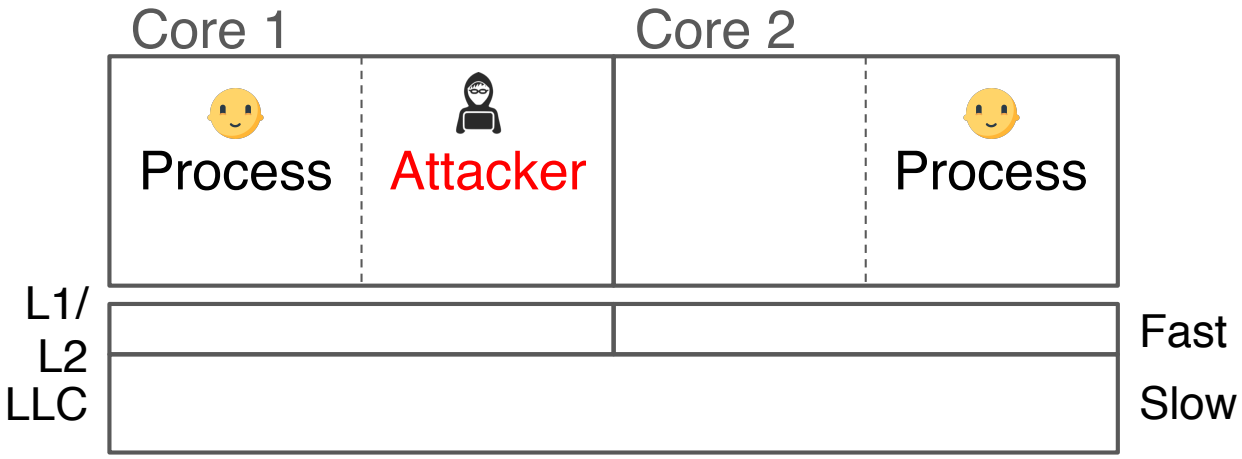
Handshake

- Use shared access timing



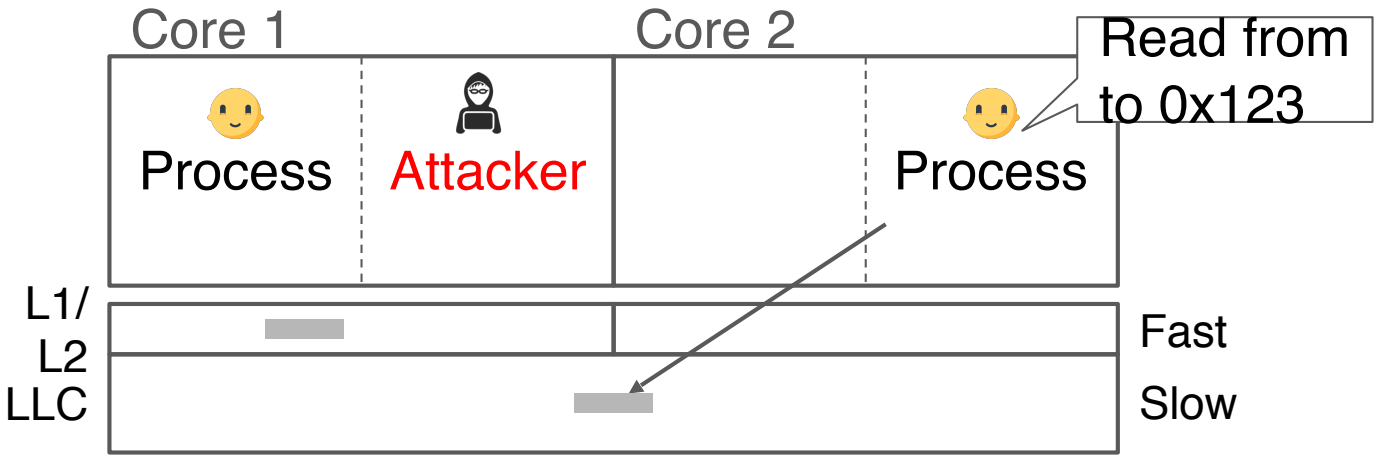
Handshake

- Use shared access timing



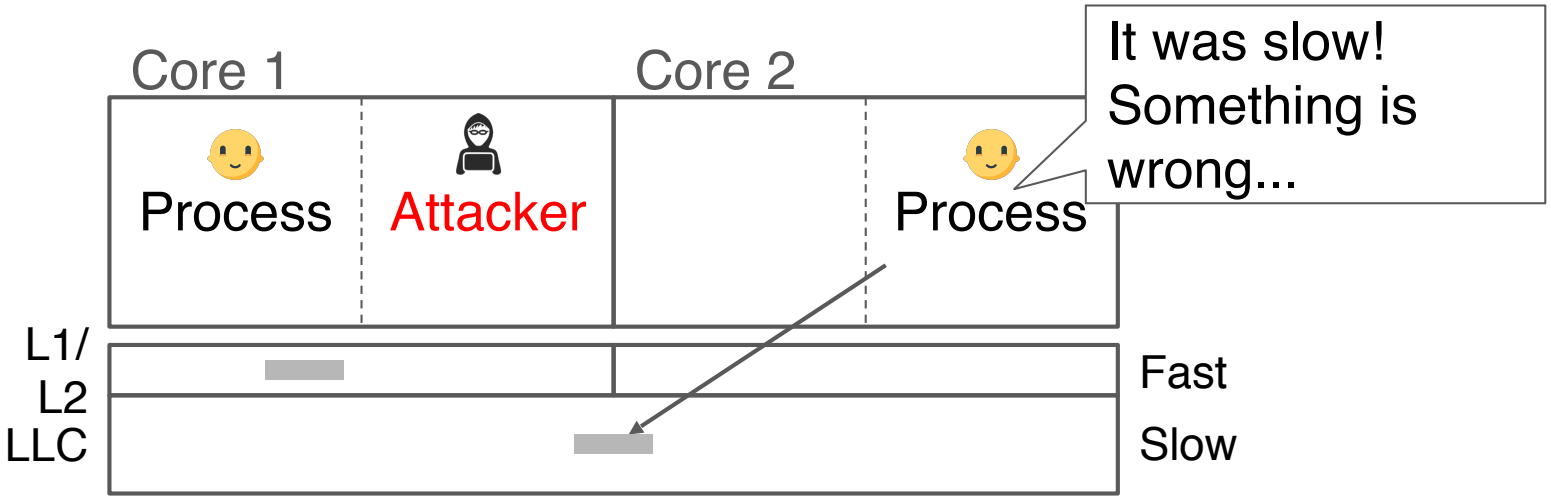
Handshake

- Use shared access timing



Handshake

- Use shared access timing



Design

- High preemption rate
- Predefined cache state
- Shared core

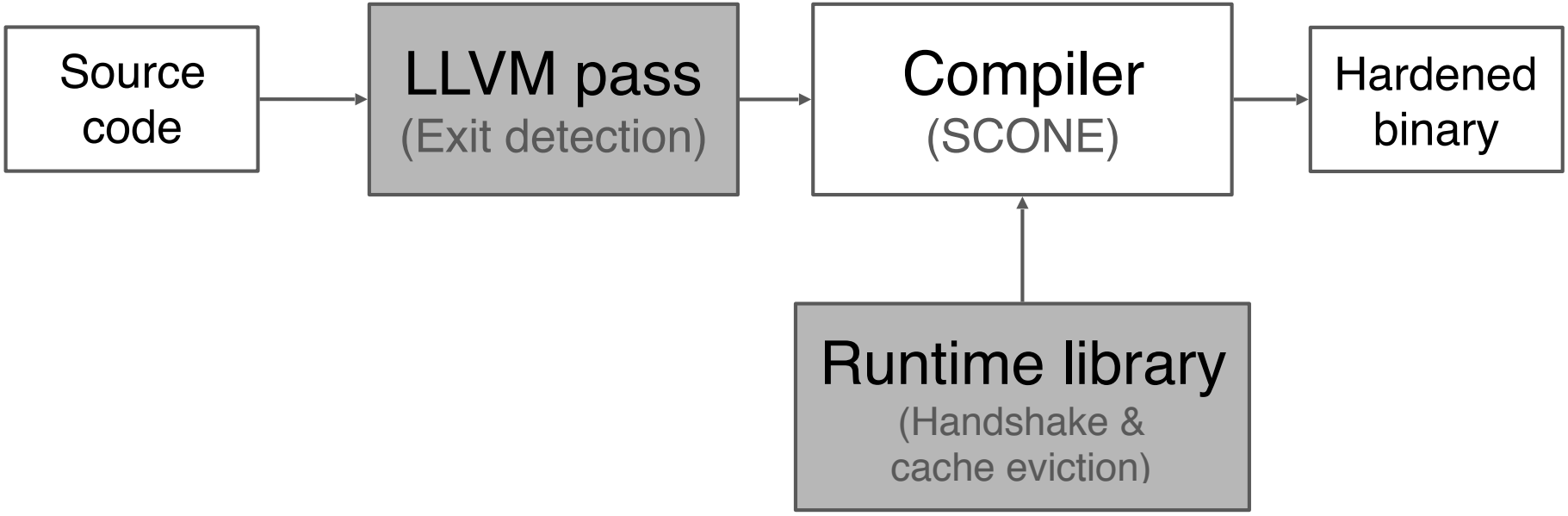
Restrict and terminate

Cache eviction

Trusted reservation

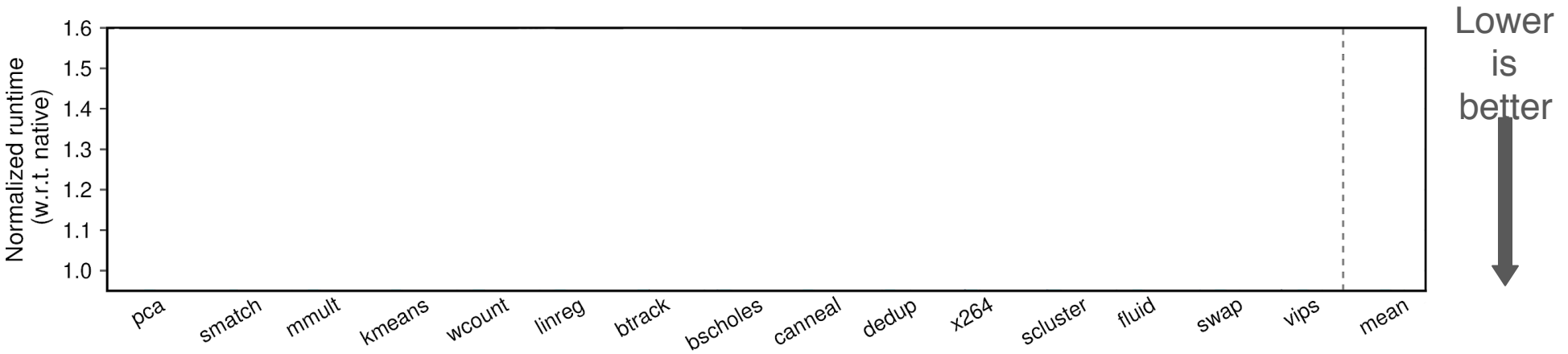
Varys **implements** a low-cost protection for Intel SGX enclaves against side-channel attacks by creating an isolated environment and verifying it at runtime.

Implementation

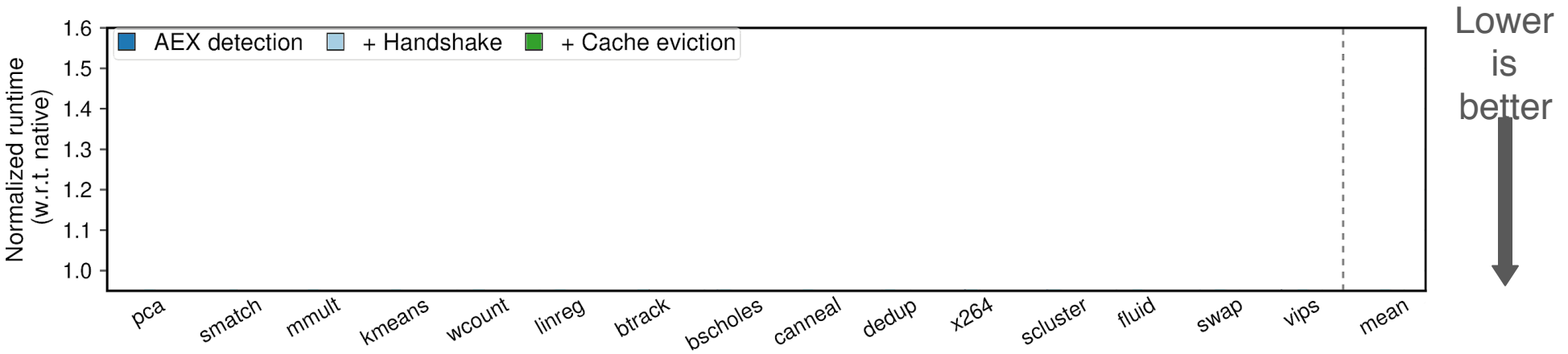


Varys implements a **low-cost** protection for Intel SGX enclaves against side-channel attacks by creating an isolated environment and verifying it at runtime.

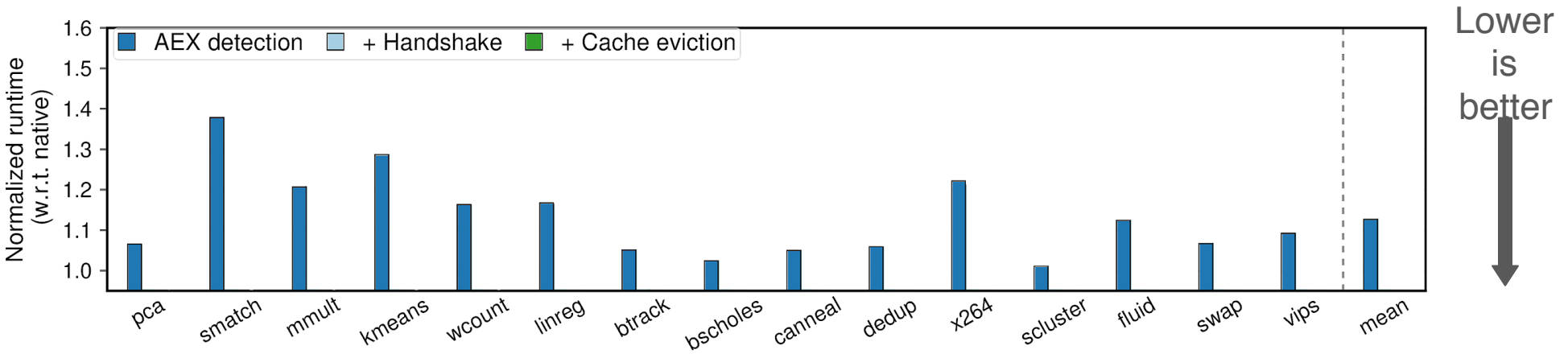
Evaluation: performance



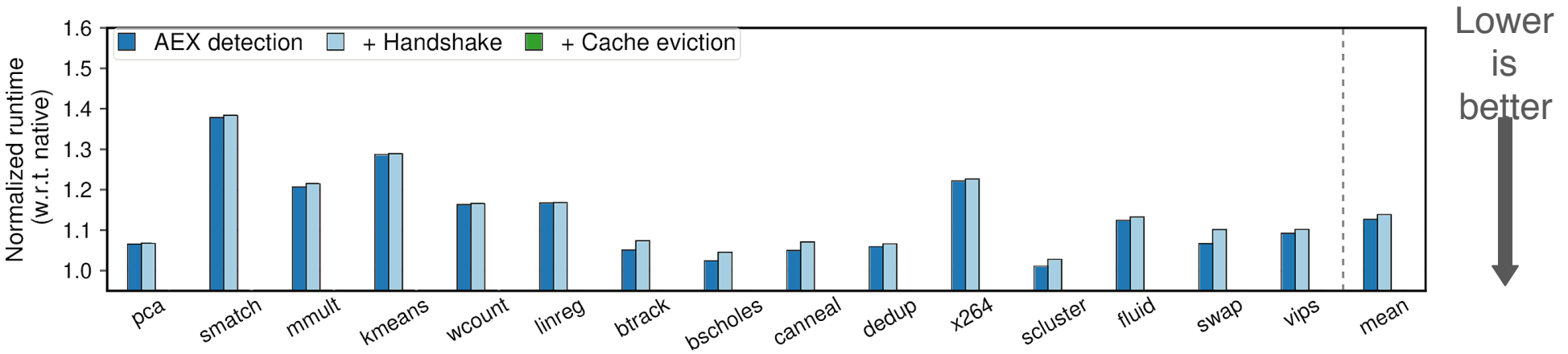
Evaluation: performance



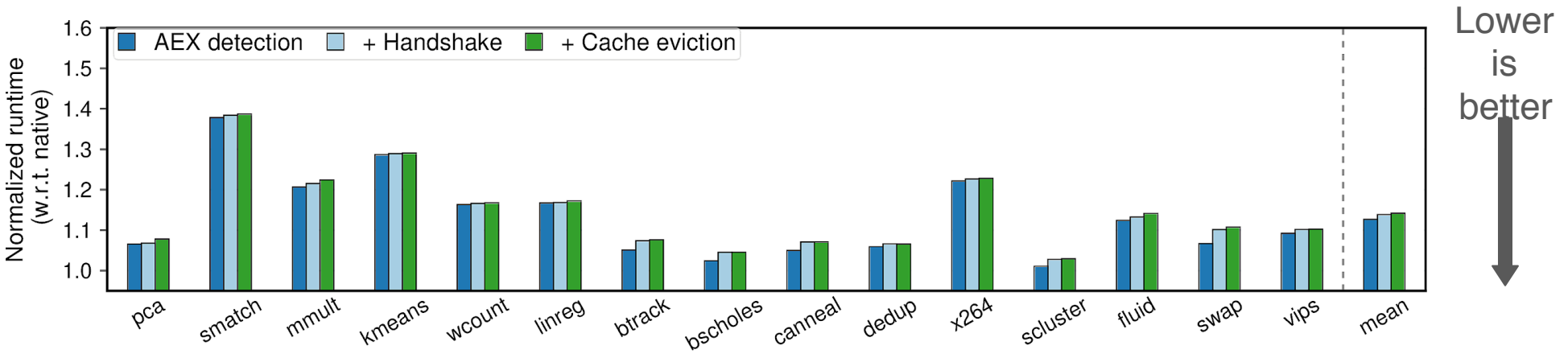
Evaluation: performance



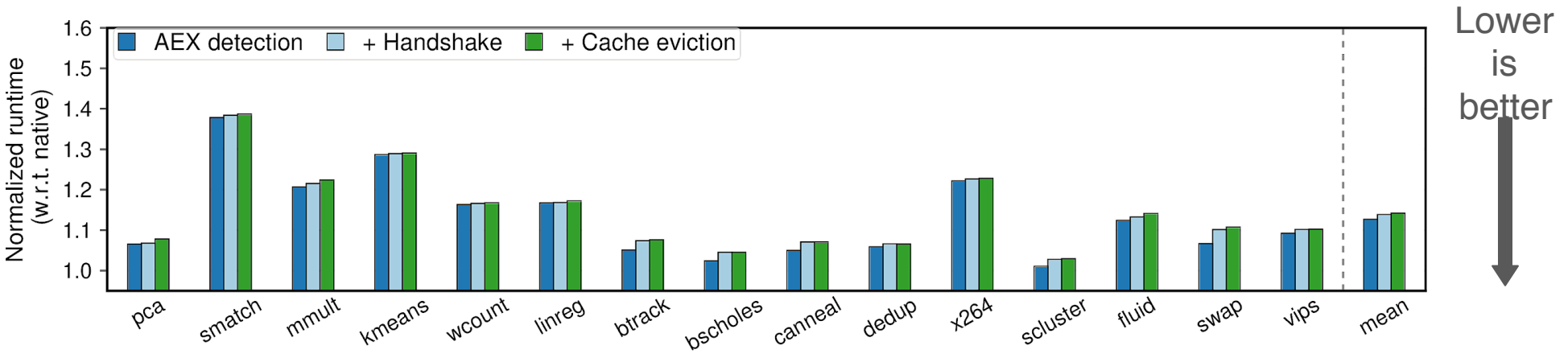
Evaluation: performance



Evaluation: performance



Evaluation: performance

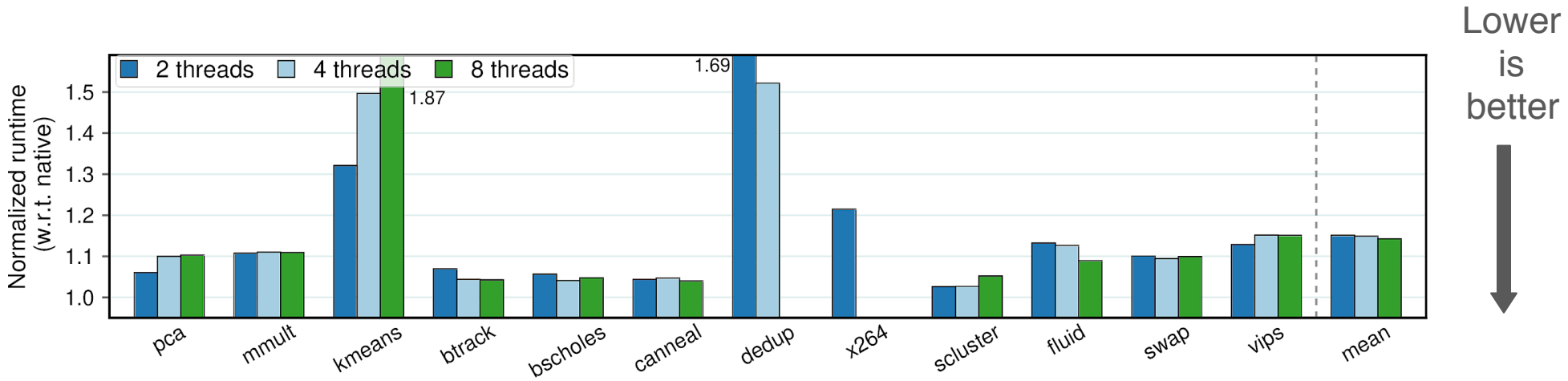


Handshake and eviction **only at enclave exits**

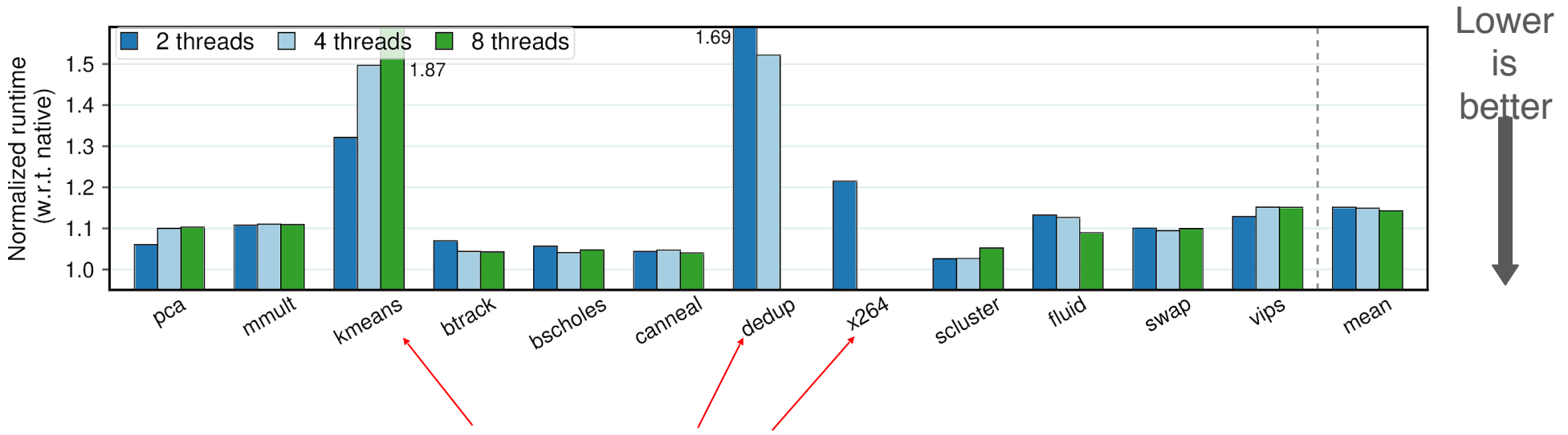
- 20-30 times per second

Evaluation: multithreading

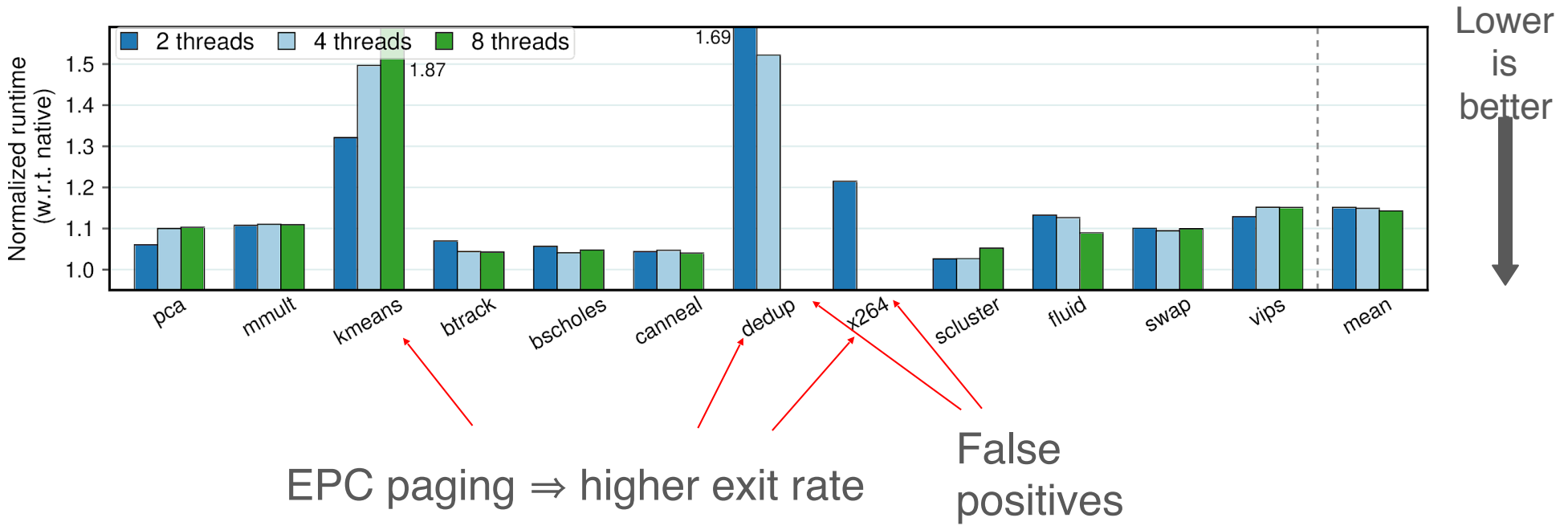
Evaluation: multithreading



Evaluation: multithreading



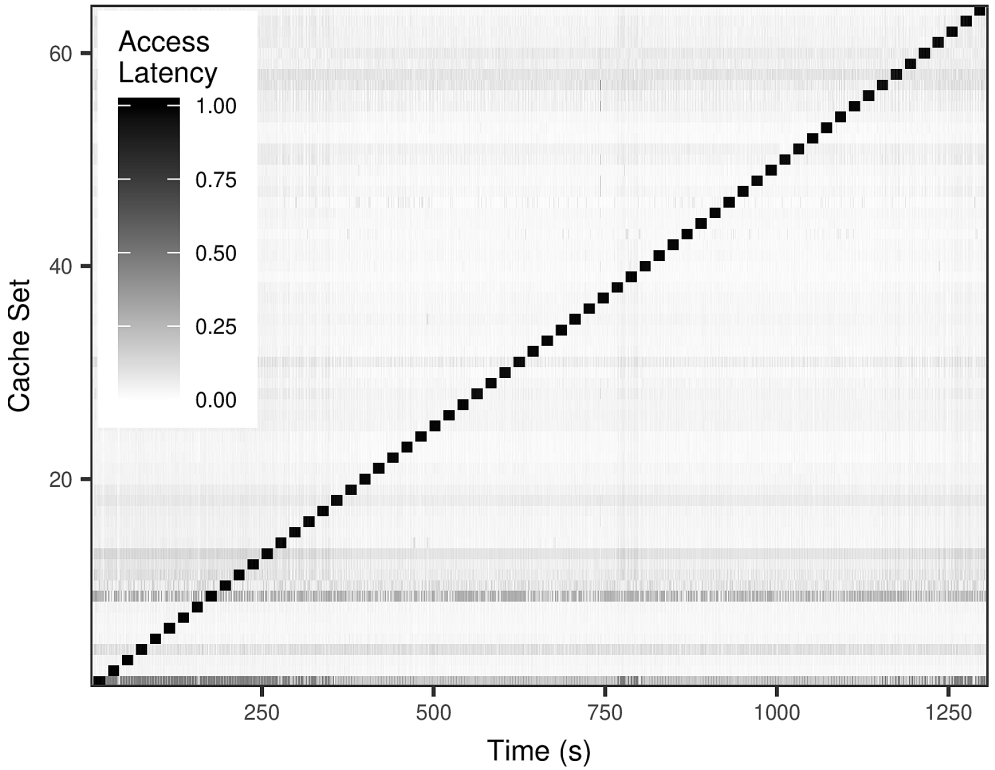
Evaluation: multithreading



Varys implements a low-cost **protection** for Intel SGX enclaves against side-channel attacks by creating an isolated environment and verifying it at runtime.

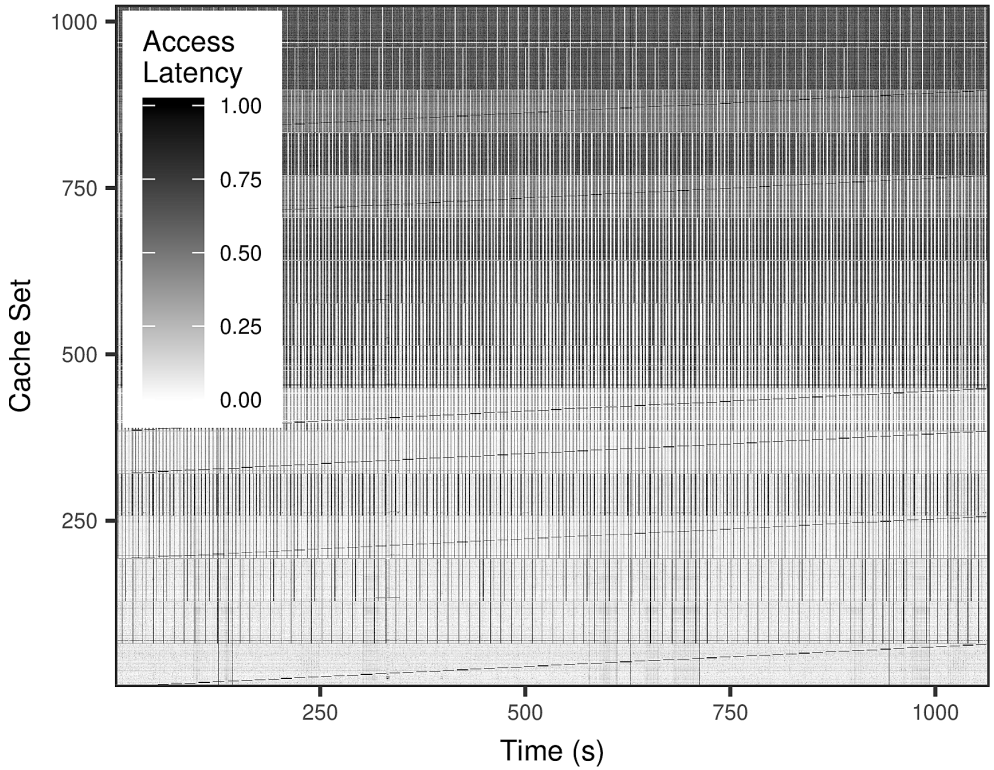
Evaluation: security

- Privileged cache SCA
  - Target: L1 cache
- No eviction



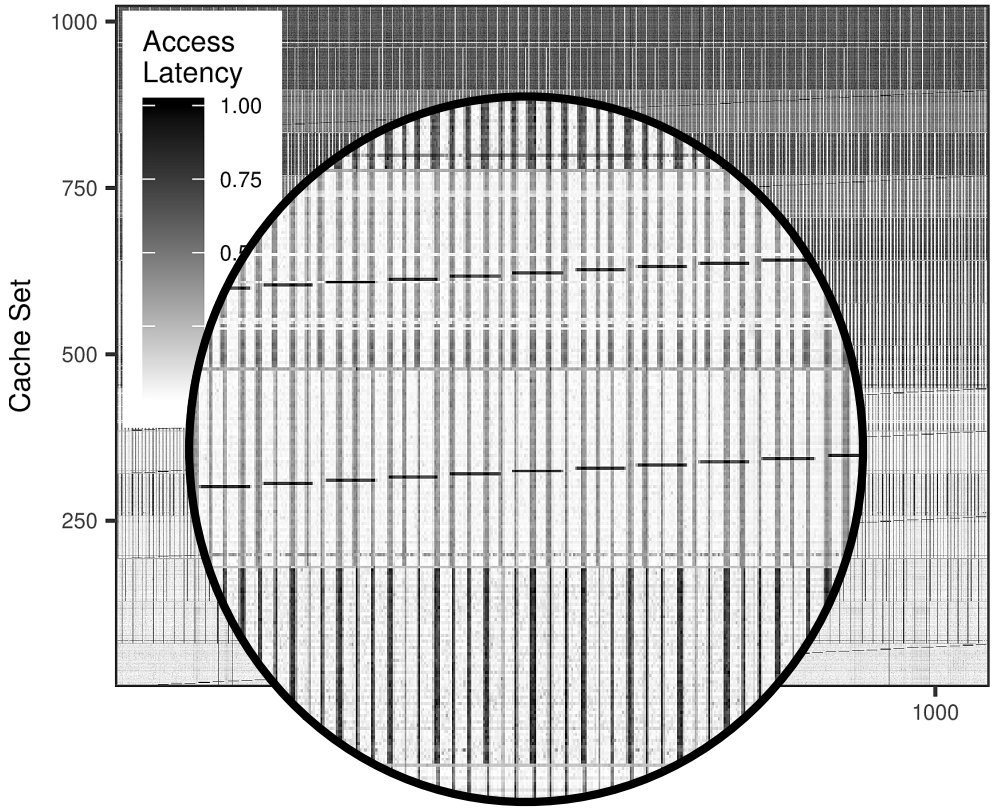
Evaluation: security

- Privileged cache SCA
  - Target: L2 cache
- No eviction



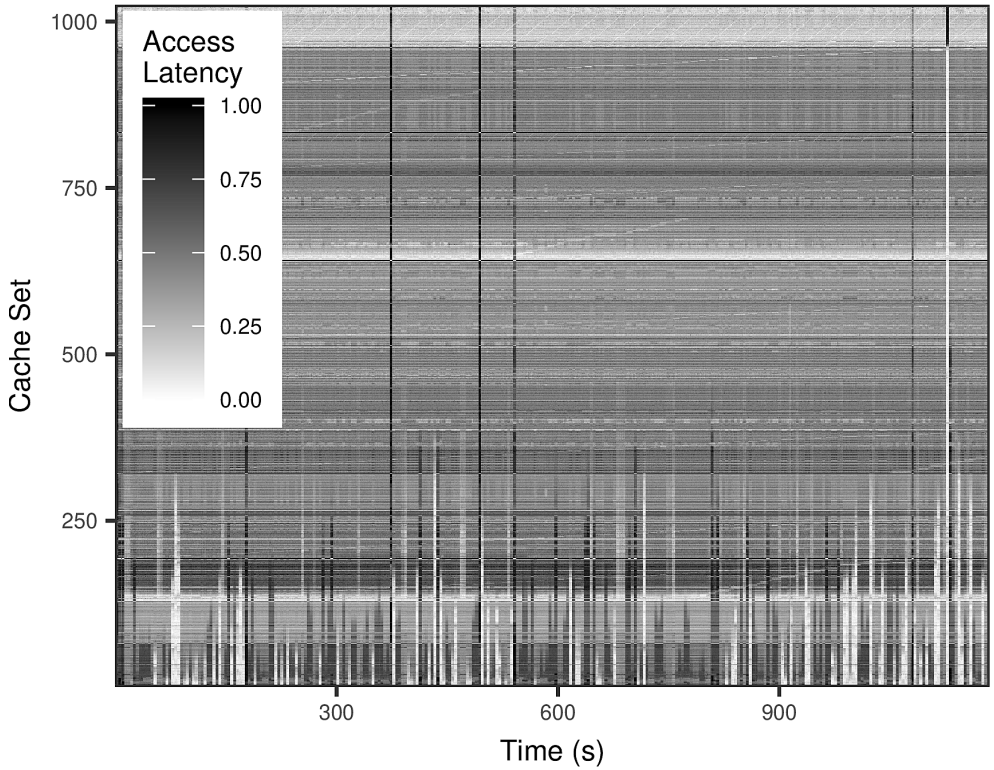
Evaluation: security

- Privileged cache SCA
  - Target: L2 cache
- No eviction



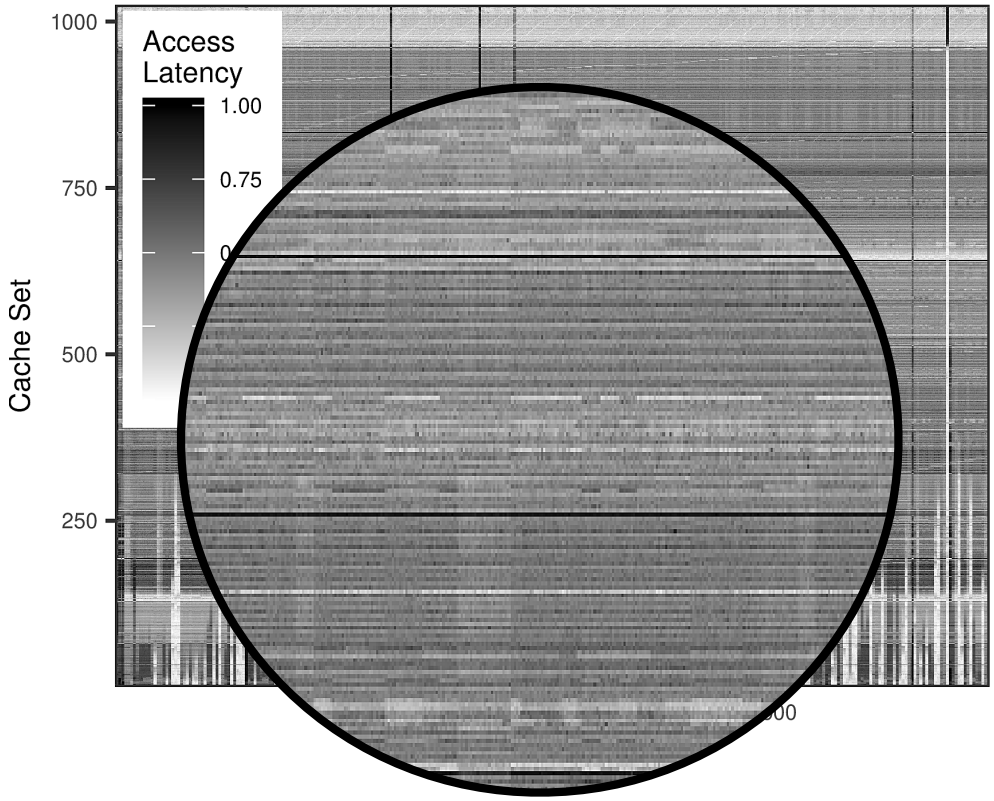
Evaluation: security

- Privileged cache SCA
  - Target: L2 cache
- Varys protection



Evaluation: security

- Privileged cache SCA
  - Target: L2 cache
- Varys protection



Varys Summary

- Varys: side-channel protection for SGX enclaves
- "Rely but verify" approach
  - Ask OS for
    - Lower interrupt rate
    - Paired thread allocation
  - Verify the request
- Loads cache on enclave entrance