CYSEC
Cybersecurity
TU Darmstadt

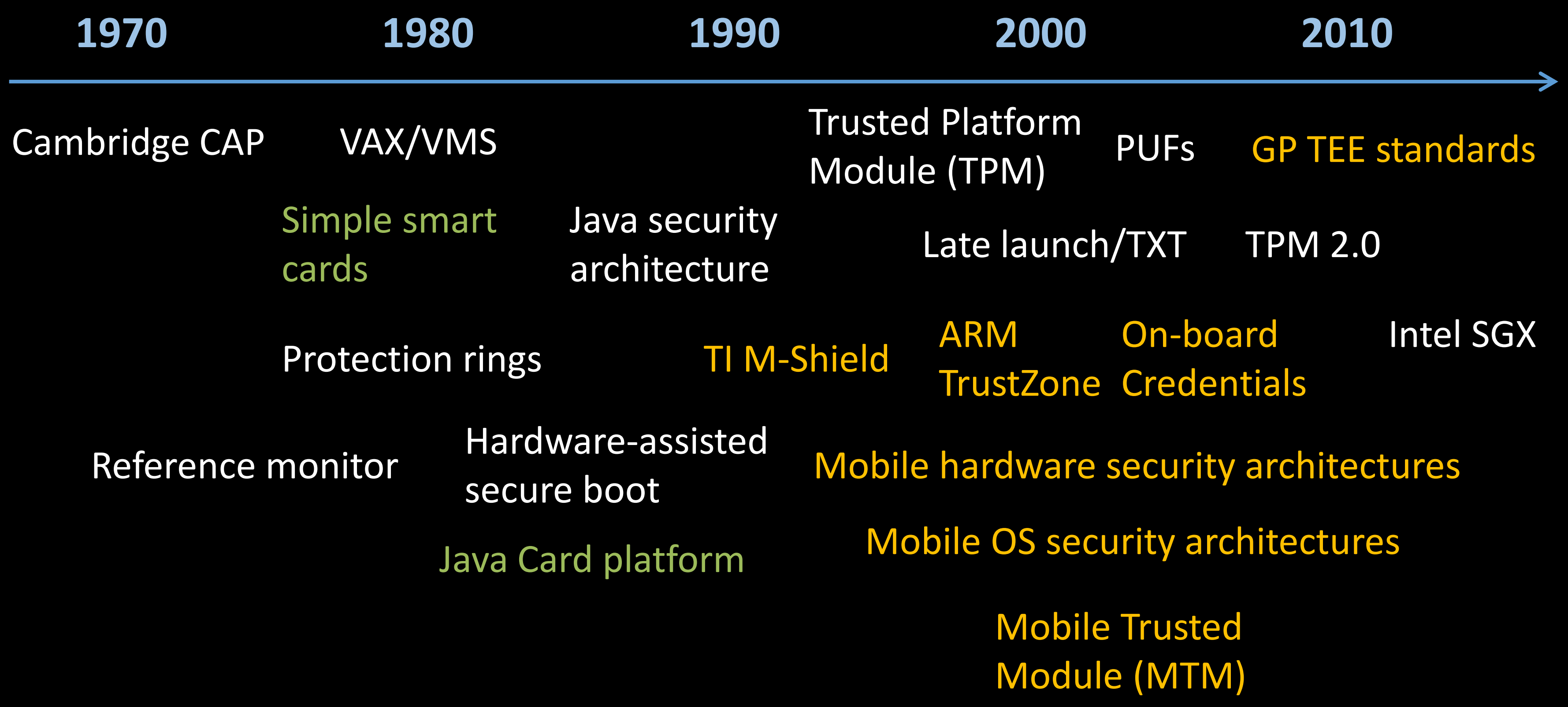
Hardware-assisted Security: From Trust Anchors to Meltdown of Trust

Ahmad-Reza Sadeghi

Technische Universität Darmstadt &

Intel Collaborative Research Institute for Collaborative & Resilient Autonomous Systems

Historical Overview: Deployed Systems



1970 1980 1990 2000 2010

Cambridge CAP

VAX/VMS

Trusted Platform Module (TPM)

PUFs

GP TEE standards

Simple smart cards

Java security architecture

Late launch/TXT

TPM 2.0

Protection rings

TI M-Shield

ARM TrustZone

On-board Credentials

Intel SGX

Reference monitor

Hardware-assisted secure boot

Mobile hardware security architectures

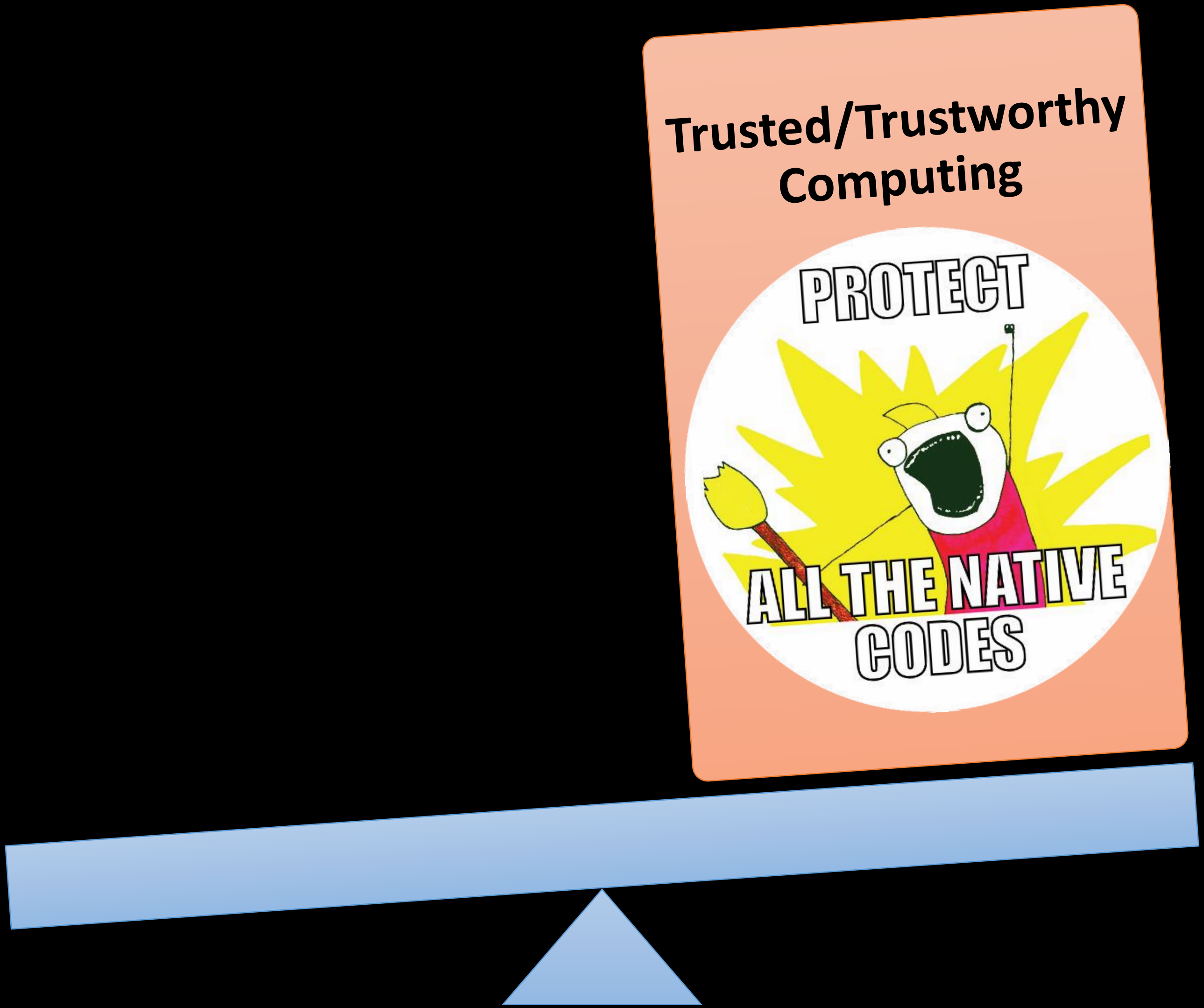
Java Card platform

Mobile OS security architectures

Mobile Trusted Module (MTM)

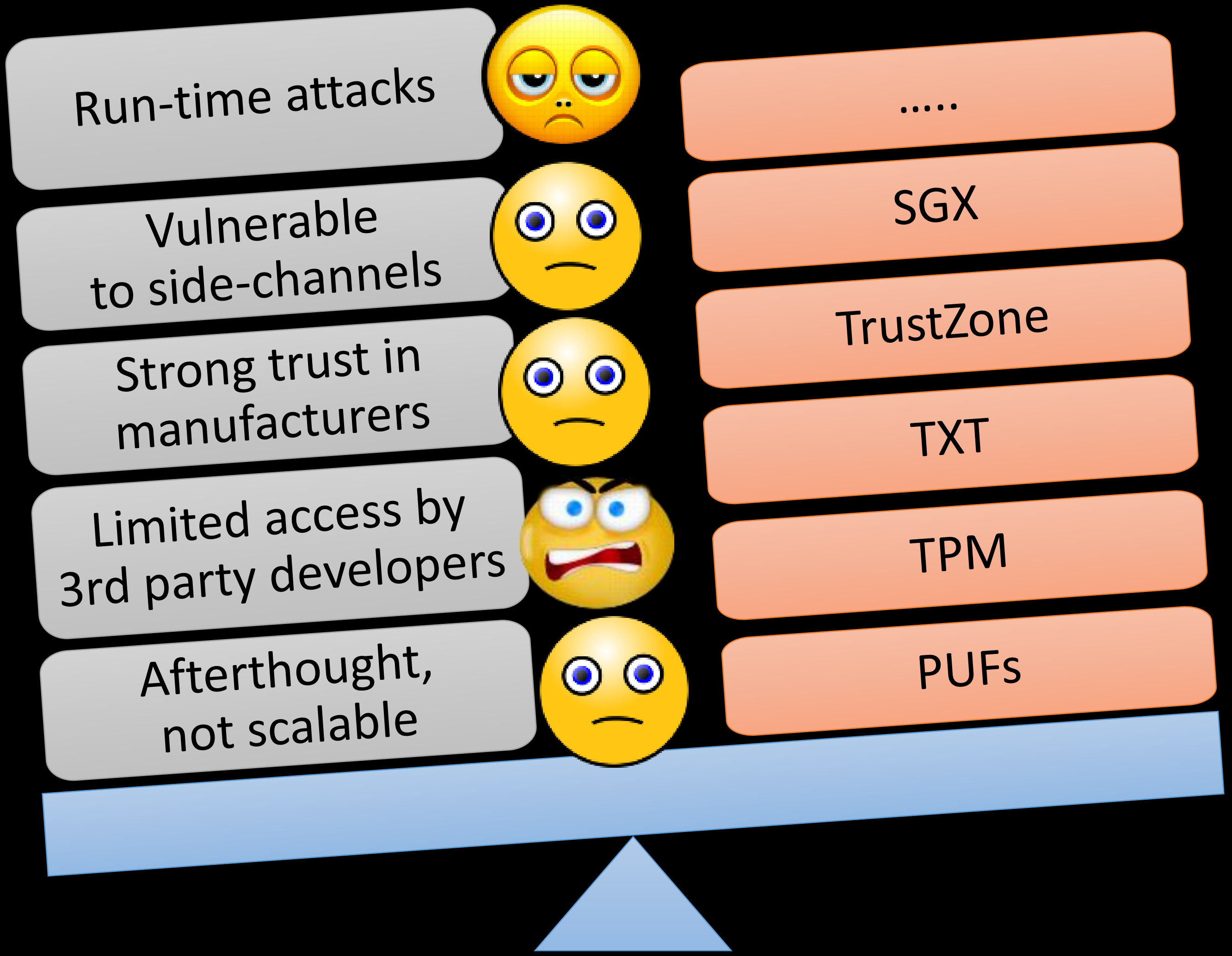
Computer security
Mobile security
Smart card security

Deployed HW-Assisted Security Technologies



Trusted/Trustworthy Computing

Deployed HW-Assisted Security Technologies



Run-time attacks

Vulnerable to side-channels

Strong trust in manufacturers

Limited access by 3rd party developers

Afterthought, not scalable

.....

SGX

TrustZone

TXT

TPM

PUFs

Fantastic

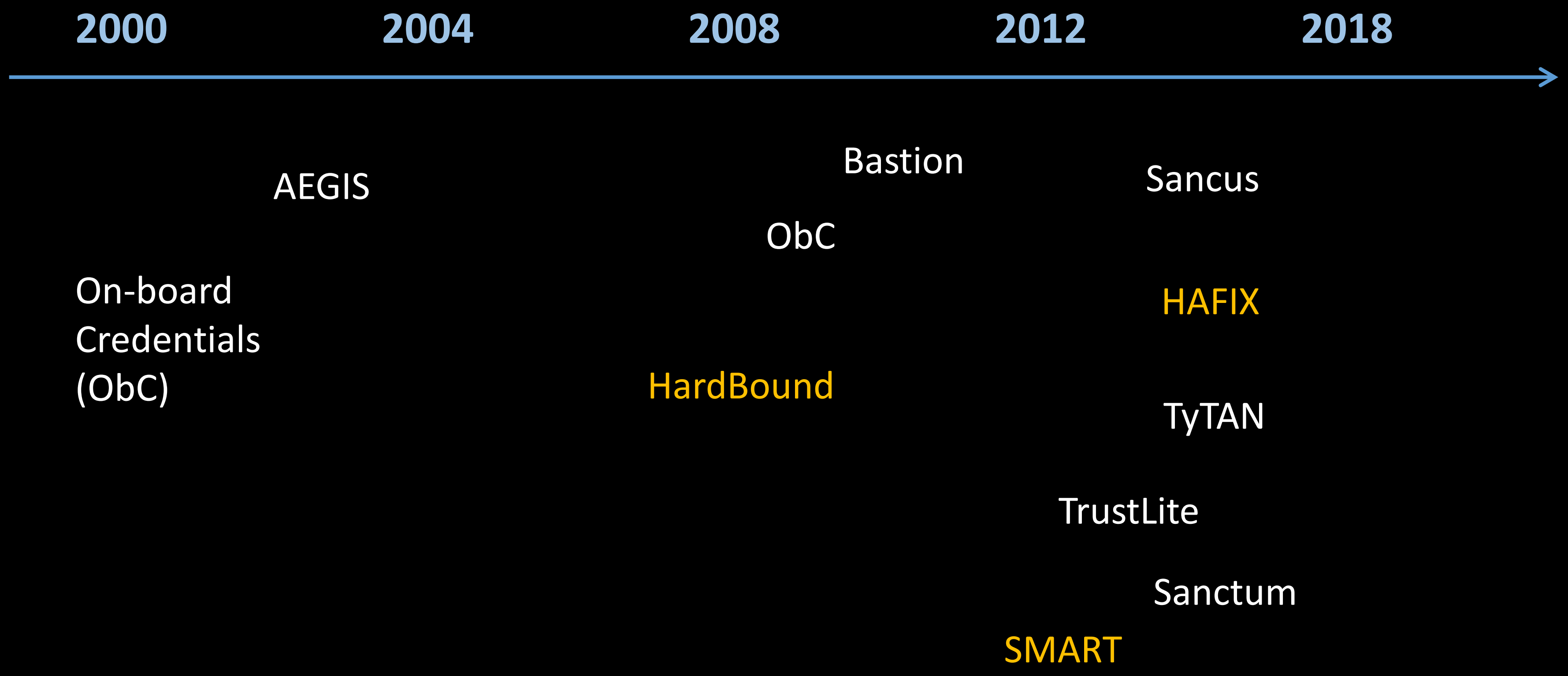
Sad

Very Sad

Complicated?

Total Disaster

Historical Overview: Research



2000

2004

2008

2012

2018

On-board Credentials (ObC)

AEGIS

HardBound

ObC

Bastion

SMART

TrustLite

Sancus

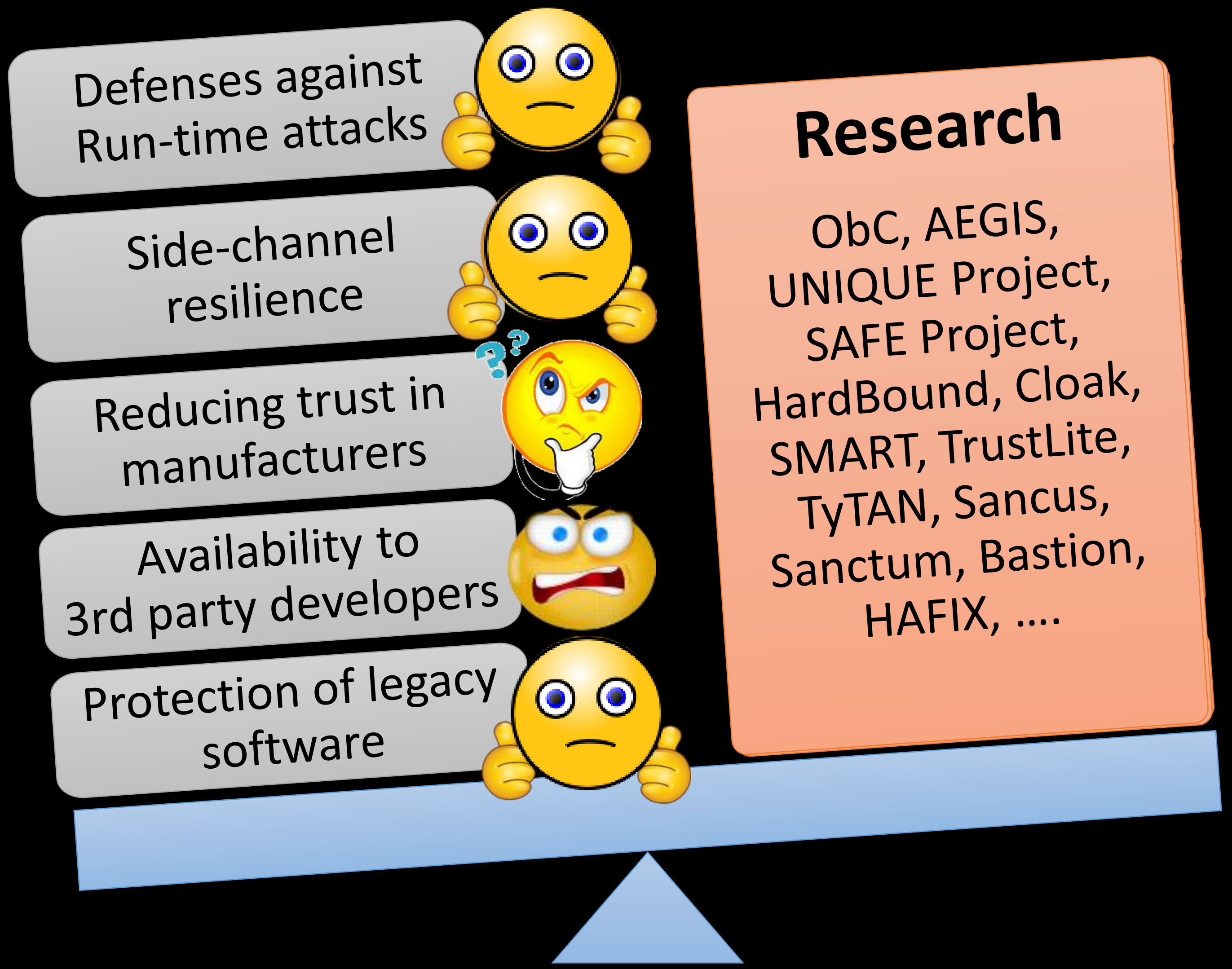
HAFIX

TyTAN

Sanctum

Trusted Execution
Security Extensions

HW-Assisted Security Technologies: Research


Defenses against Run-time attacks
Side-channel resilience
Reducing trust in manufacturers
Availability to 3rd party developers
Protection of legacy software
Research
ObC, AEGIS, UNIQUE Project, SAFE Project, HardBound, Cloak, SMART, TrustLite, TyTAN, Sancus, Sanctum, Bastion, HAFIX, ….

Fantastic

Almost Optimistic

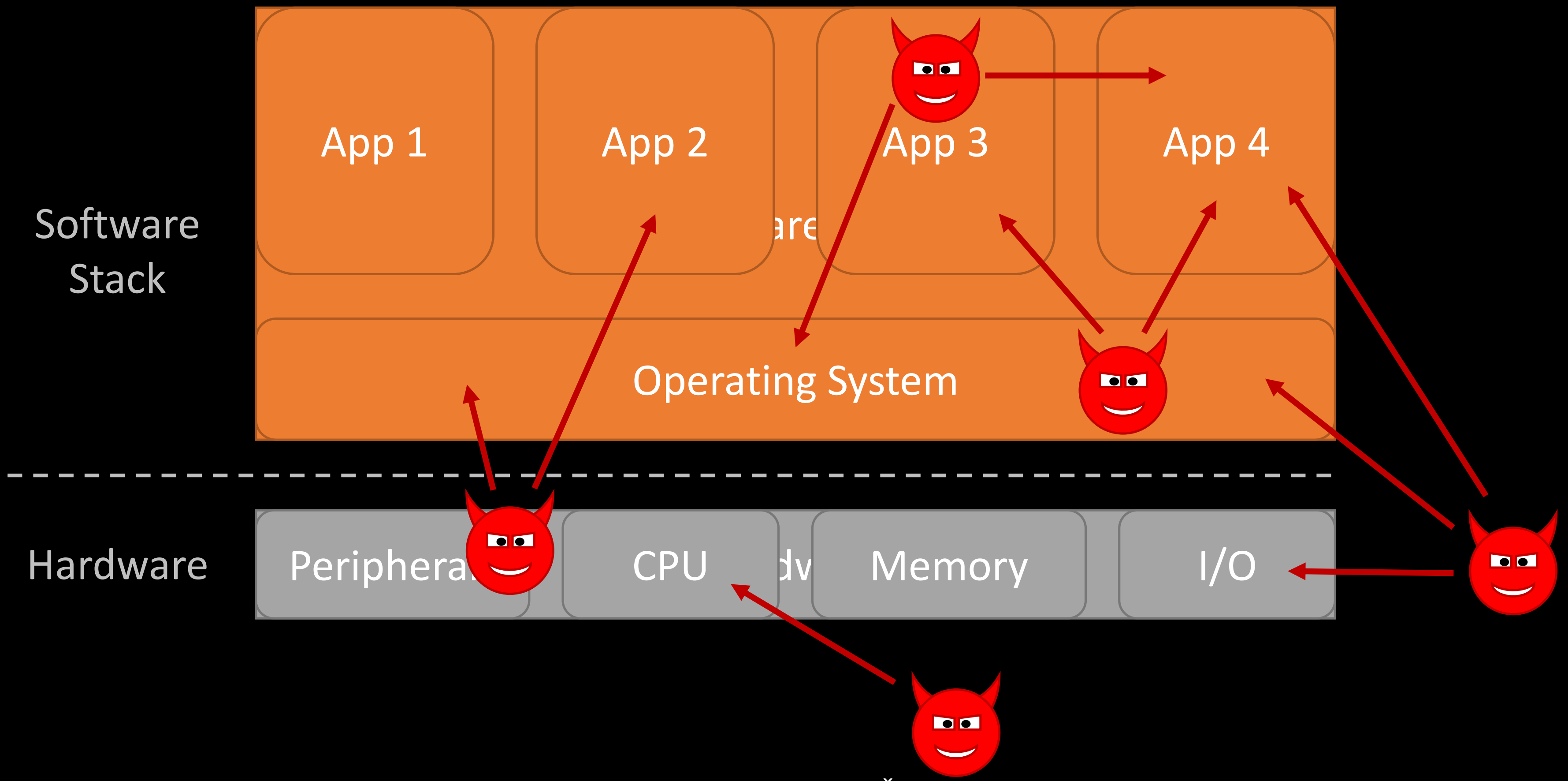
Sad

Complicated?

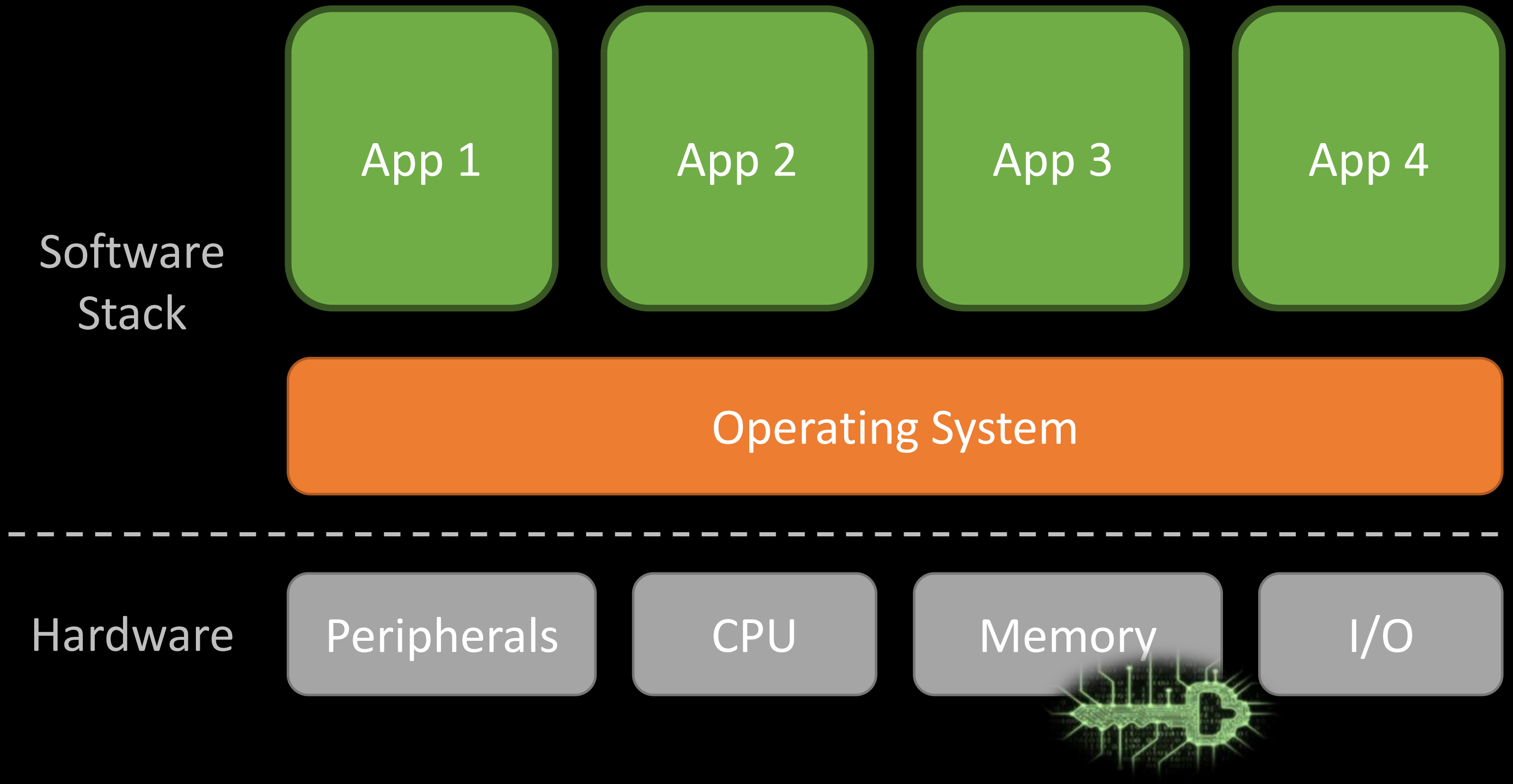
Total Disaster

We Need Change of Culture!

Today’s Systems: Attack Surface



Goal: Self-Contained Security

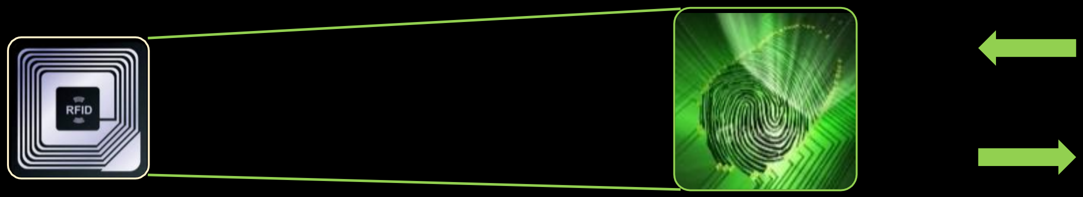


- Isolated execution
- Platform integrity
- Secure storage
- Device identification
- Device authentication capabilities

Intrinsic Security Primitives: The PUF Myth

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

Physically Unclonable Functions (PUFs)



Challenge c

Response r

Device

Physically Unclonable Function
(noisy function based on physical properties)

Hardware Fingerprint
(unique intrinsic identifier)

Inherently Unclonable

Due to unpredictable randomness during manufacturing of tag

Infeasible to predict

Challenge/response behavior is pseudo-random

Tamper-evident

Tampering with the PUF hardware changes challenge/response behavior

SELECTED PUFs

2001	🟢 Optical PUF [P.Ravikanth, 2001]
2002-2004	🟣 Arbiter PUF & RO-PUF [Gassend et al., CCS‘02] 🟣 Feed-Forward A-PUF [Lee et al., VLSIC’04]
2006	🟢 Coating PUF [Tuyls et al., CHES’06]
2007	🟠 SRAM PUF [Guajardo et al., CHES’07][Holcomb et al., RFIDSec’07] 🟠 Latch PUF [Su et al., ISSCC‘07] 🟣 XOR A-PUF [Suh et al., DAC’07]
2008	🟠 Flip-Flop PUF [Kumar et al., WiSec’08] 🟠 Butterfly PUF [Su et al., HOST‘08] 🟣 Lightweight PUF [Majzoobi et al., ICCAD‘08]
2010-2011	🟣 Glitch PUF [Anderson et al., ASP-DAC‘10] 🟠 Flash PUF [Prabhu et al., ICTTC‘11] 🟢 Current-based PUF [Majzoobi et al., ISCAS‘11] 🟣🟠 Bistable Ring PUF [Chen et al., HOST‘11]
2012-2013	🟠 Buskeeper PUF [Simons et al., HOST‘12] 🟠 DRAM PUF [Rosenblatt et al., SSC‘13] 🟢 Subthreshold Current PUF [Kalyanaraman et al., HOST‘13]
2014-2015	🟠 Bitline PUF [Holcomb et al., CHES‘14] 🟣 Processor-based PUF [Kong et al., DAC’14] 🟢 Current Mirrors PUF [Kumar et al., HOST‘14] 🟢 Voltage Transfer PUF [Vijaykumar et al., DATE‘15]
2016-now	🟢 MEMS PUF [Willers et al., CCS‘16] 🟠 Row Hammer-PUF [Schaller et al., HOST‘17] 🟣 MXPUF [Nguyen et al., eprint‘17] 🟣 Monte Carlo PUF [Rožić et al., FPT ‘17]

unique
www.unique-project.eu

EU UNIQUE Project

🟠 Memory-based PUFs
🟣 Delay-based PUFs
🟢 Other PUFs

PUFs: Main Categories

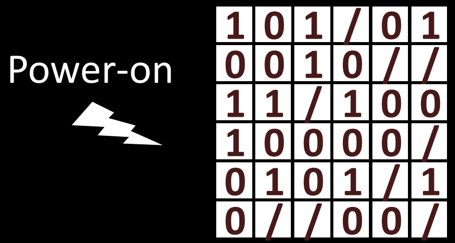
Memory-based PUFs

SRAM PUF
[Guajardo et al., CHES'07]
[Holcomb et al., RFIDSec'07]

Flip-Flop PUF
[Kumar et al., WiSec'08]

DRAM PUF
[Rosenblatt et al., SSC'13]

Row Hammer-PUF
[Schaller et al., HOST'17]



1	0	1	/	0	1
0	0	1	0	/	/
1	1	/	1	0	0
1	0	0	0	0	/
0	1	0	1	/	1
0	/	/	0	0	/

The output is based on the state of memory cells after a power cycle

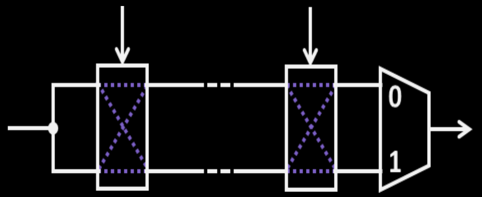
Delay-based PUFs

Arbiter PUF & RO-PUF
[Gassend et al., CCS'02]

Feed-Forward A-PUF
[Lee et al., VLSIC'04]

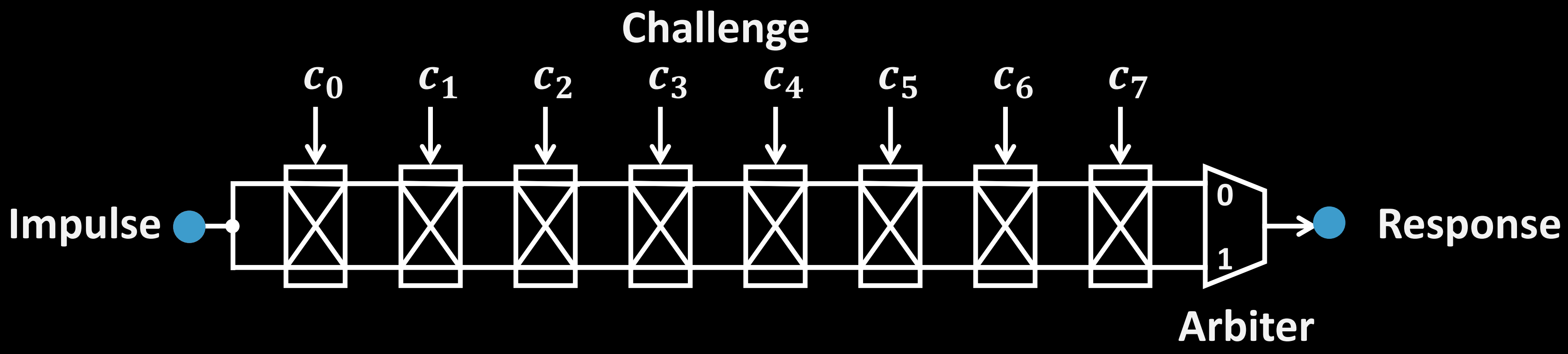
XOR A-PUF
[Suh et al., DAC'07]

Lightweight PUF
[Majzoobi et al., ICCAD'08]



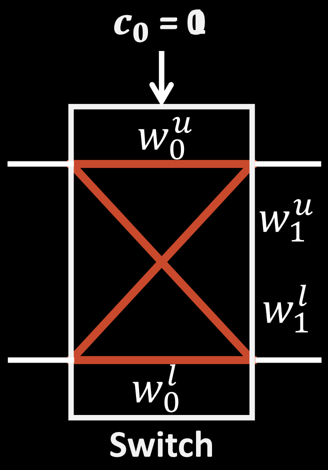
The output determined by the faster path

Example: Arbiter PUF



Pair of identically designed delay lines

- Ideally both paths have the same delay
- Arbiter determines signal arrives first
- Challenge dependent switches
- Different delay paths by switches



Manufacturing variations affect delay lines

- Either of the two paths will be faster
- One bit response at signal arrival

CYSEC
Cybersecurity
TU Darmstadt

How Good are PUFs in Practice?

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

PUF Security in Practice

Memory-based PUFs

SRAM PUF
[Guajardo et al., CHES'07]
[Holcomb et al., RFIDSec'07]

1	0	1	/	0	1
0	0	1	0	/	/
1	1	/	1	0	0
1	0	0	0	0	/
0	1	0	1	/	1
0	/	/	0	0	/

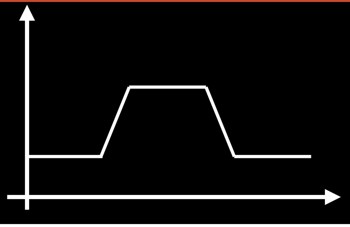
Physical Attacks

[Oren et al., CHES'13]
[Helfmeier et al., HOST'13]

Delay-based PUFs

Arbiter PUF
[Gassend et al., CC'04]

Linear Behavior!



Modeling Attacks

[Lee et al., VLSIC'04]

XOR A-PUF
[Suh et al., DAC'07]

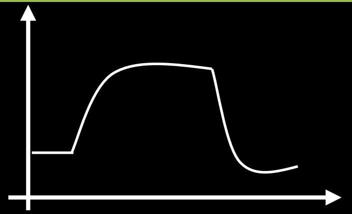
Add non-linear funcions

Modeling Attacks

[Ruhrmair et al., CCS'10]
[Becker, CHES'15]

Memristor A-PUF
[Suh et al., DAC'15]

Add non-linear components



Physical Attacks
[Merli et al., WESS'11]
[Tajik et al., CHES'14]
[Rührmair et al., CHES'14]

SELECTED ATTACKS & ANALYSIS

2004

- **ML-Modeling Attack (A-PUF)** [Lee et al., VLSIC’04]

2008

- **ML-Modeling Attack (FF A-PUF)** [Majzoobi et al., ITC’08]

2010-2012

- **ML-Modeling Attack delay-based PUFs** [Ruhrmair et al., CCS’10]
- **Formal Security Model** [Armknecht et al., S&P 2011]
- **PUFs: Myth, Fact or Busted?** [Katzenbeisser et al., CHES’12]
- **Semi-Invasive EM Attack (RO-PUF)** [Merli et al., WESS’11]

2013

- **Semi-Invasive Attack on PUFs** [Nedospasov et al., FDTC’13]
- **Cloning SRAM PUF** [Helfmeier et al., HOST’13]
- **Rémanence Decay SCA (SRAM PUF)** [Oren et al., CHES’13]
- **Noise SCA (A-PUF)** [Delvaux et al., HOST’13]

2014

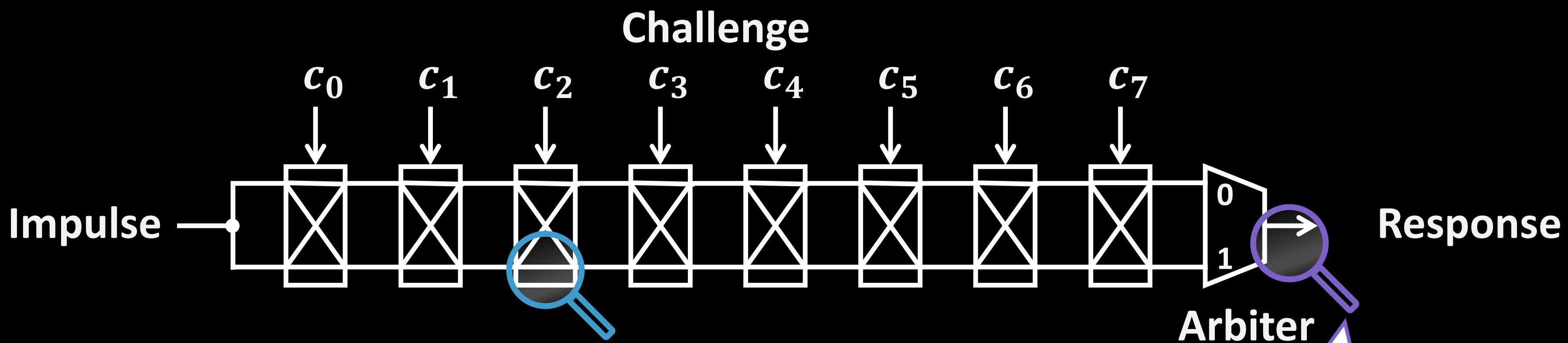
- **Photon Emission Analysis (A-PUF)** [Tajik et al., CHES’14]
- **ML-Modeling Attack (Bistable Ring PUF)** [Hesselbarth et al., TRUST’14]
- **Hybrid Modeling Attacks (Current-based PUF)** [Kumar et al., ICCD’14]
- **Power&Timing SCA (A-PUF)** [Rührmair et al., CHES’14]

2015-2018

- **Reliability-based ML-Modeling Attack (XOR A-PUF)** [Becker, CHES’15]
- **Unified Security Model for PUFs** [Armknecht et al., CT-RSA 2016]
- **ML-Modeling Attack (Bistable Ring PUF)** [Ganji et al., CHES’16]
- **ML-Modeling Attack on non-linear PUFs** [Vijaykumar et al., HOST’16]
- **Hammering RH-PUF** [Zeitouni et al., DAC’18]

Example: Arbiter PUF

Goal: Recovering the values of the wire delays inside the switch boxes



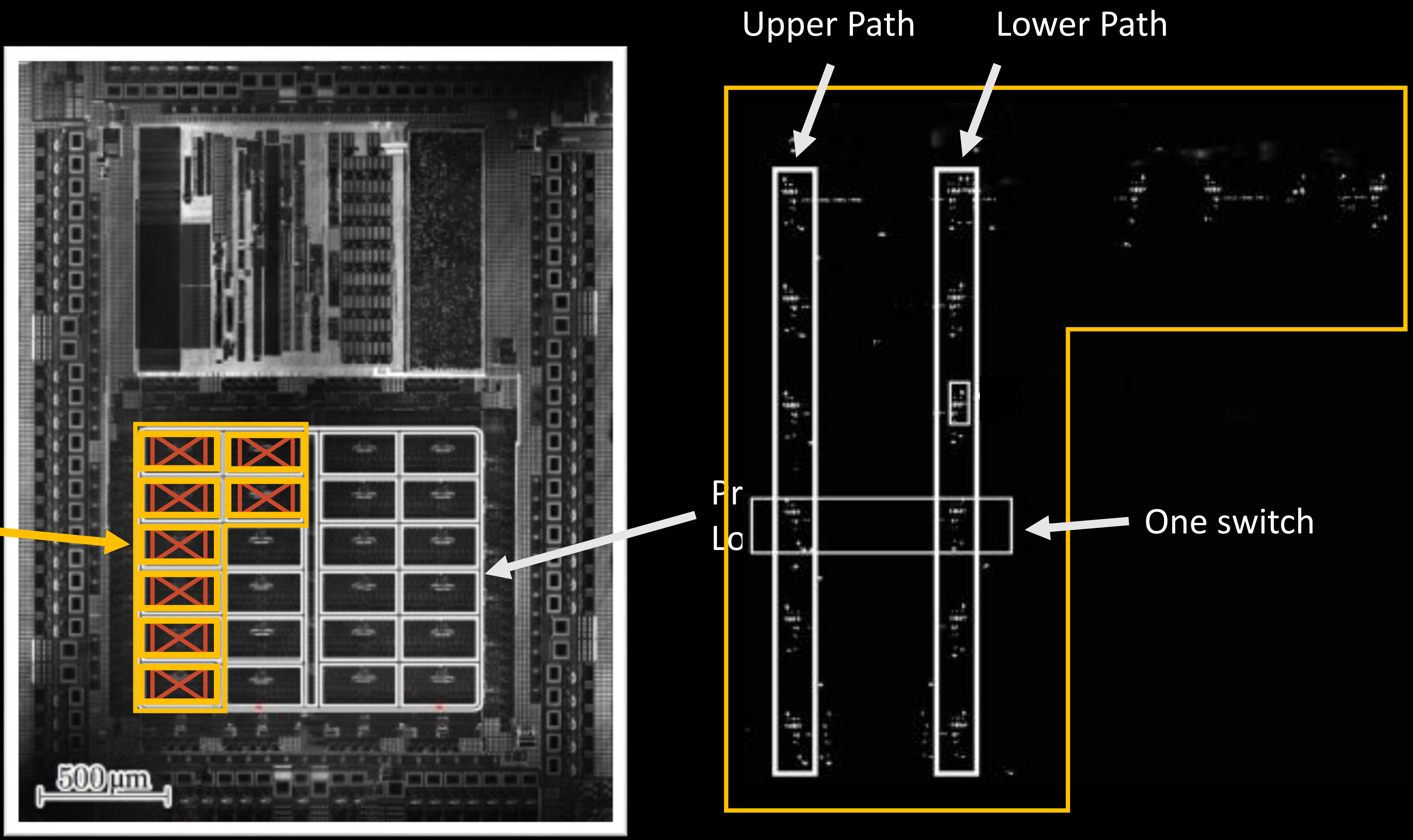
Physical Attacks
(Semi-invasive/Side-channel)

CRPs ≈ 10^2

Modeling Attacks
(Machine Learning)

CRPs ≈ 10^3 - 10^6

Arbiter PUF on a Complex Programmable Logic Device (CPLD): Backside View



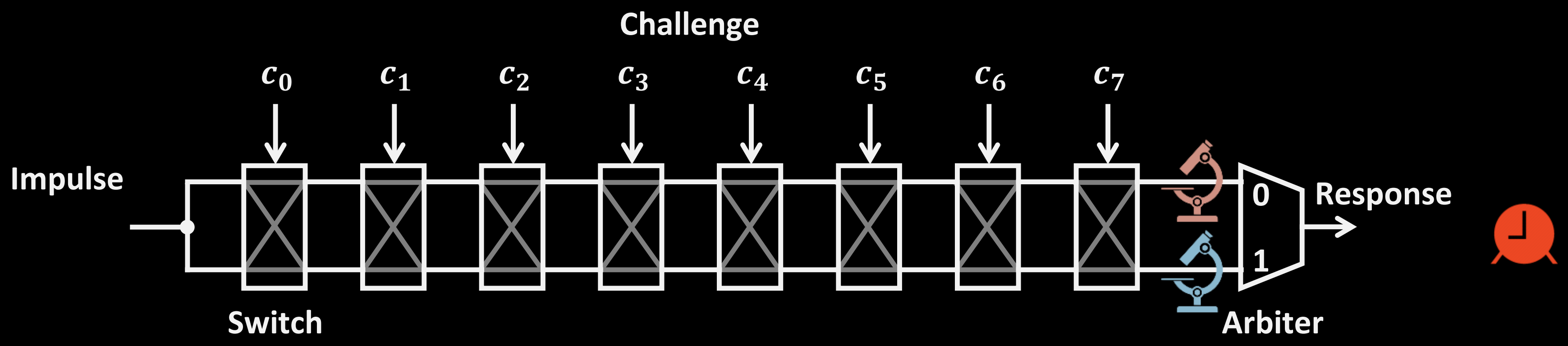
Placement of an Arbiter PUF with 8 switches

Upper Path

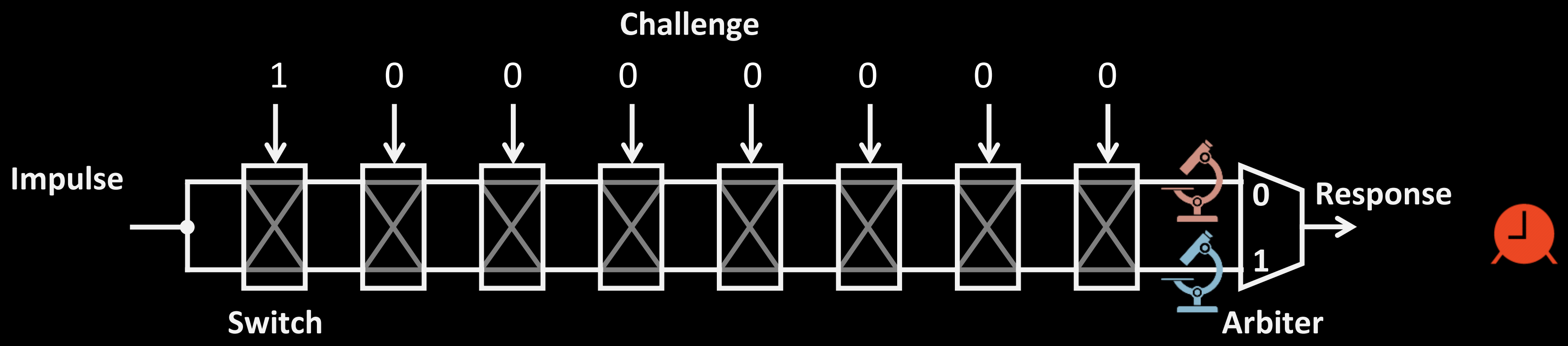
Lower Path

One switch

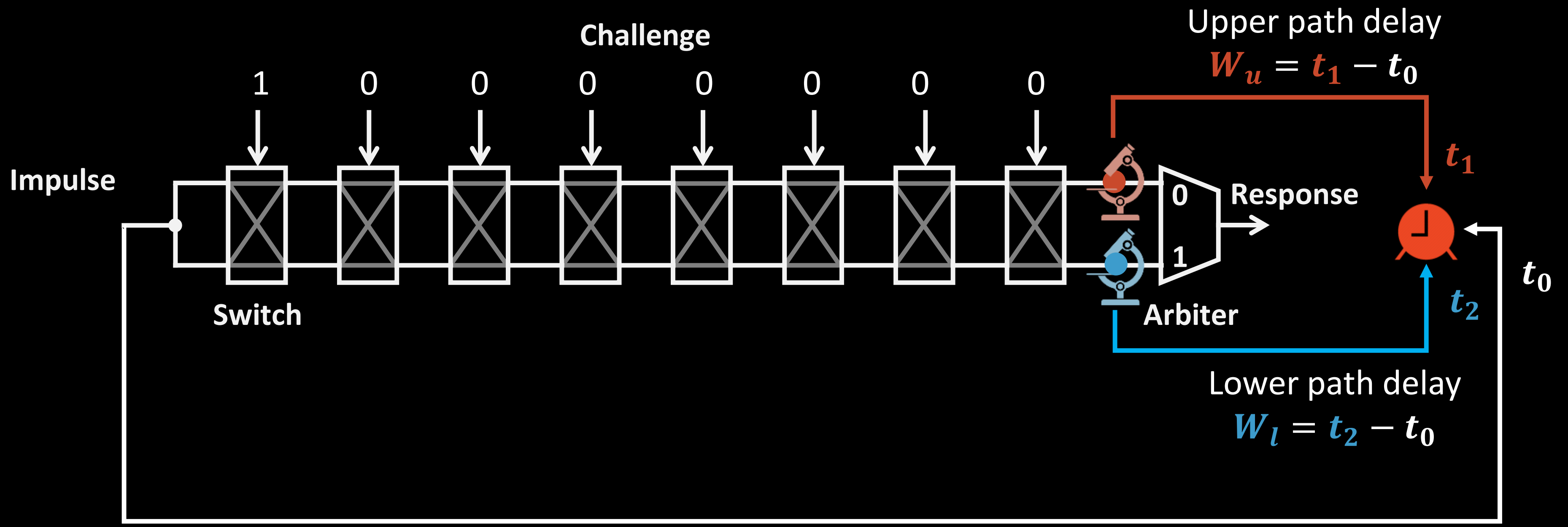
Physical Attacks: Example: [Tajik et al., CHES’14]



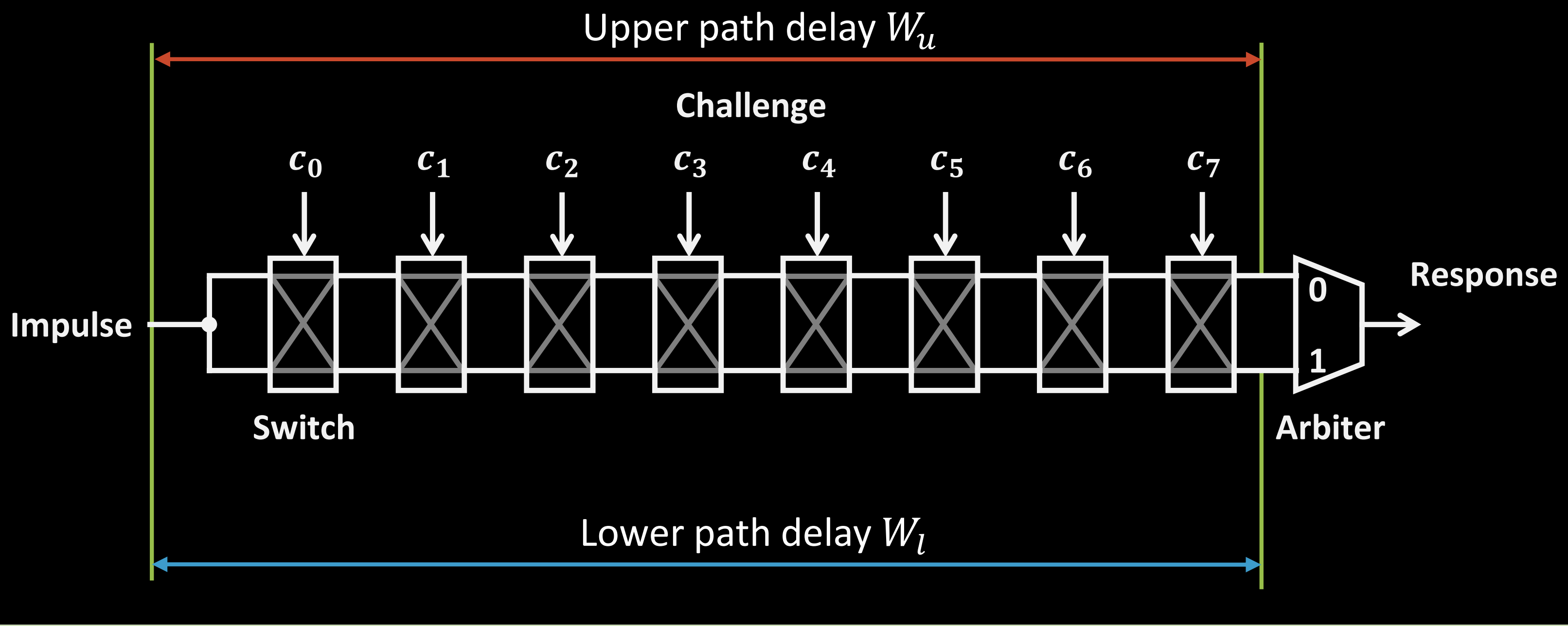
Physical Attacks: Example: [Tajik et al., CHES’14]



Physical Attacks: Example: [Tajik et al., CHES'14]

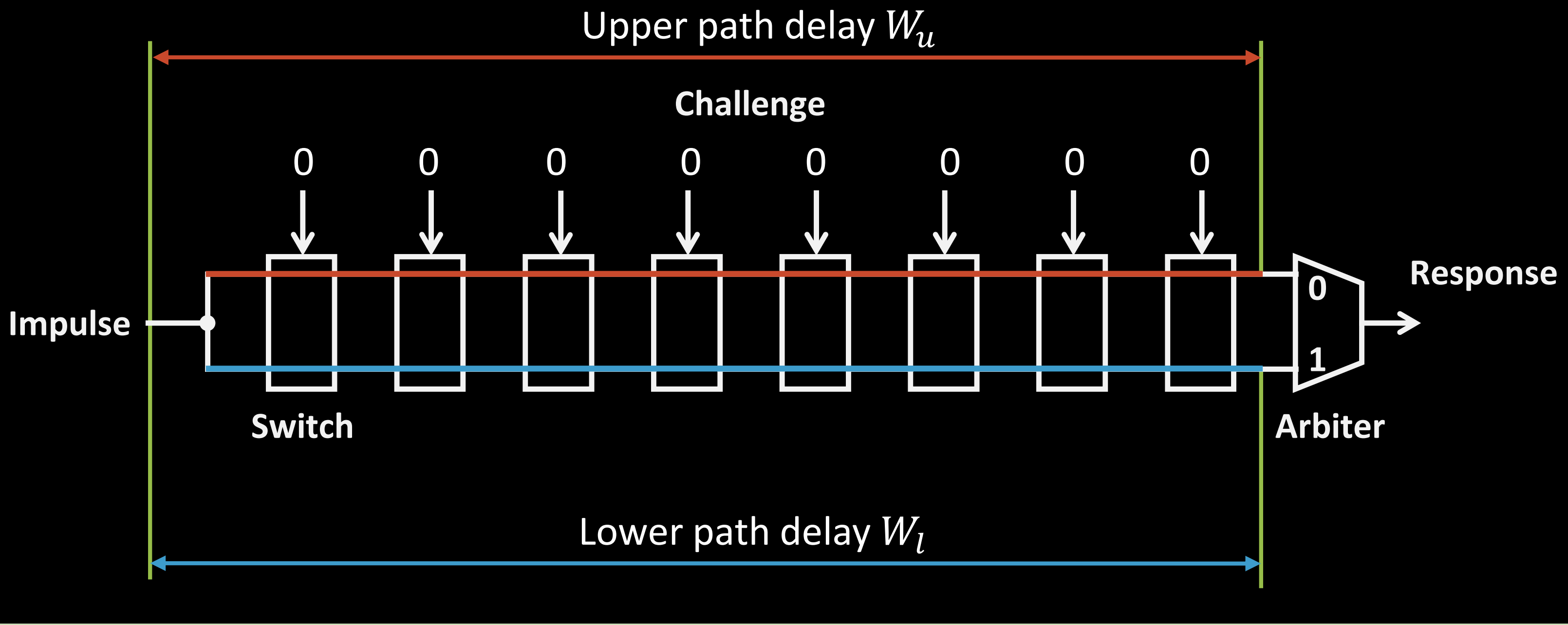


Physical Attacks: Example: [Tajik et al., CHES’14]



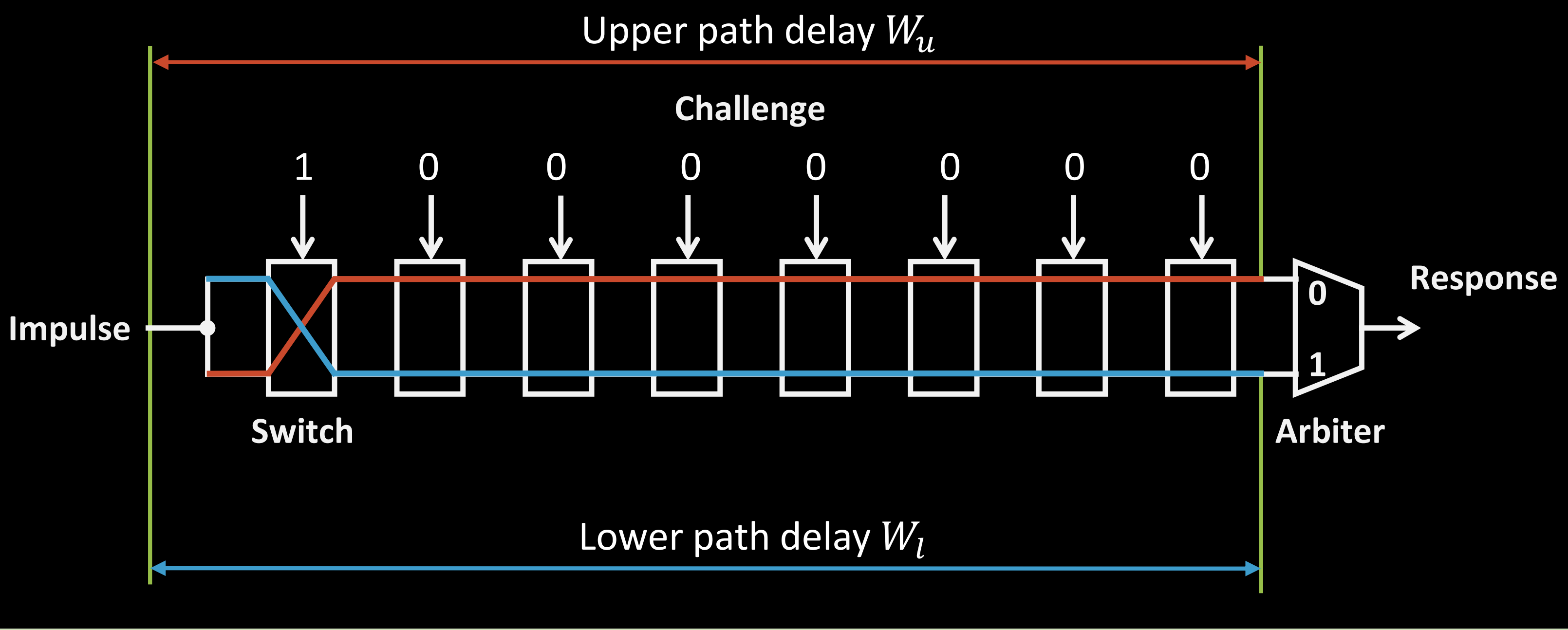
C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u									
W_l									

Physical Attacks: Example: [Tajik et al., CHES’14]



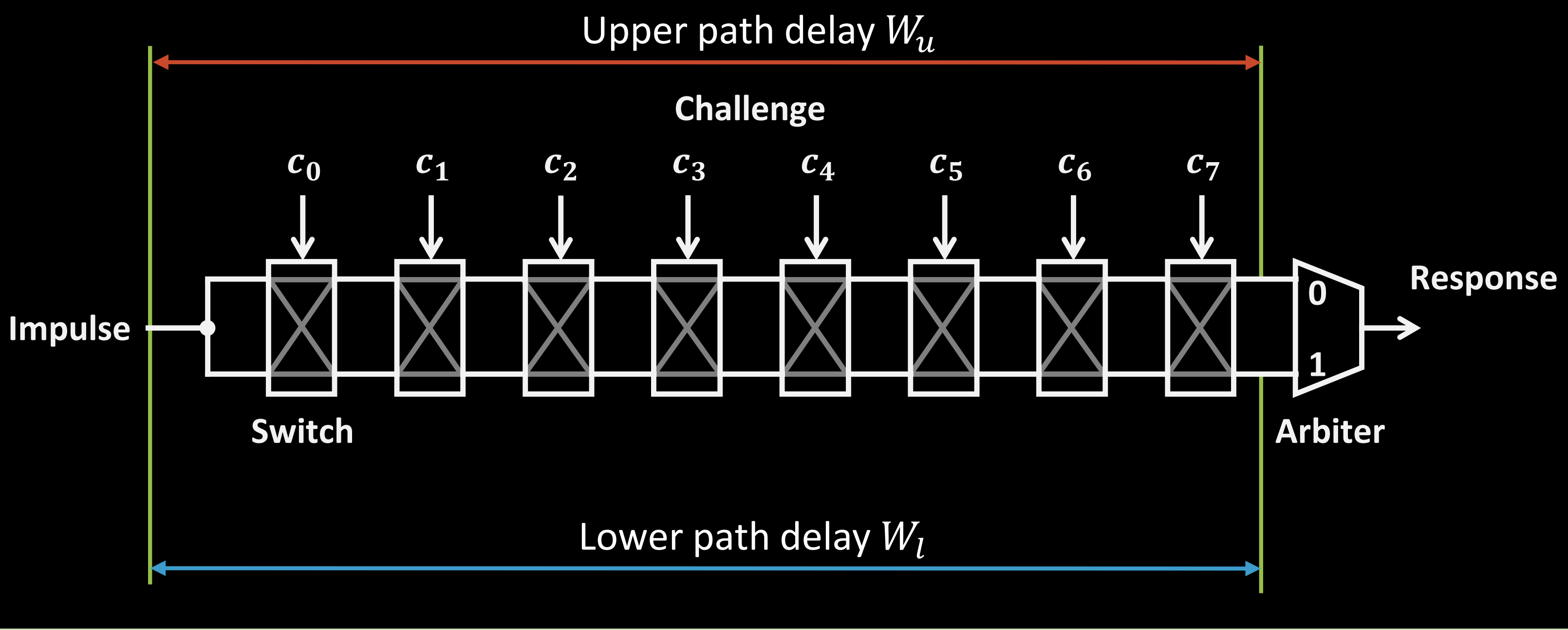
C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1								
W_l	u_1								

Physical Attacks: Example: [Tajik et al., CHES’14]



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

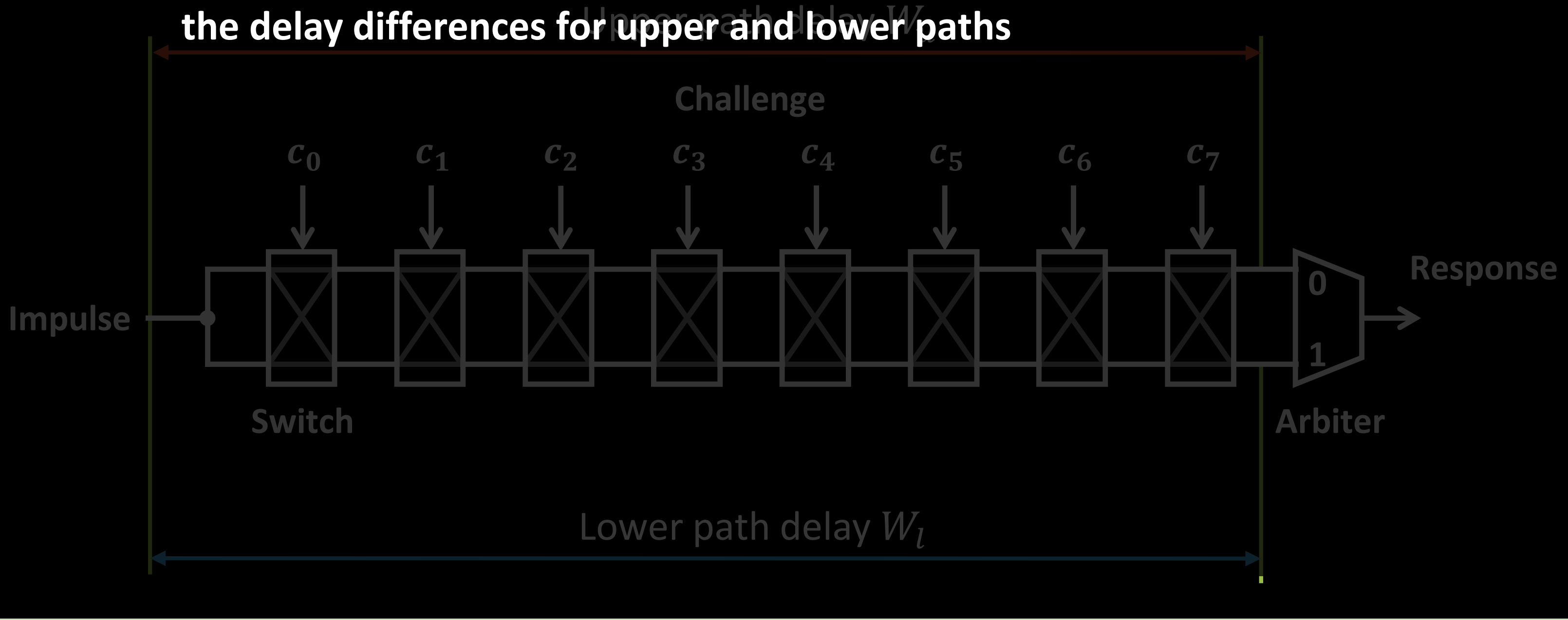
Physical Attacks: Example: [Tajik et al., CHES'14]



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

Physical Attacks: Example: [Tajik et al., CHES’14]

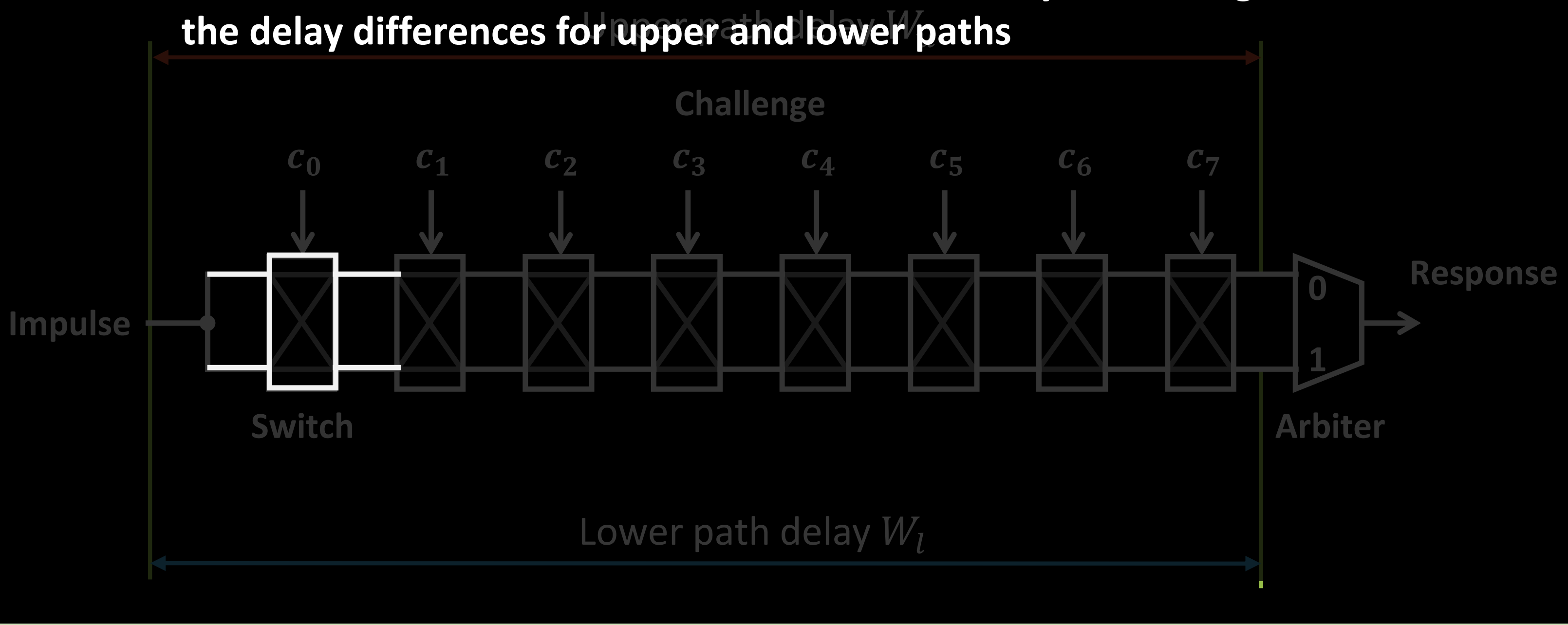
Characterize each switch box in the Arbiter PUF by calculating the delay differences for upper and lower paths



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

Physical Attacks: Example: [Tajik et al., CHES’14]

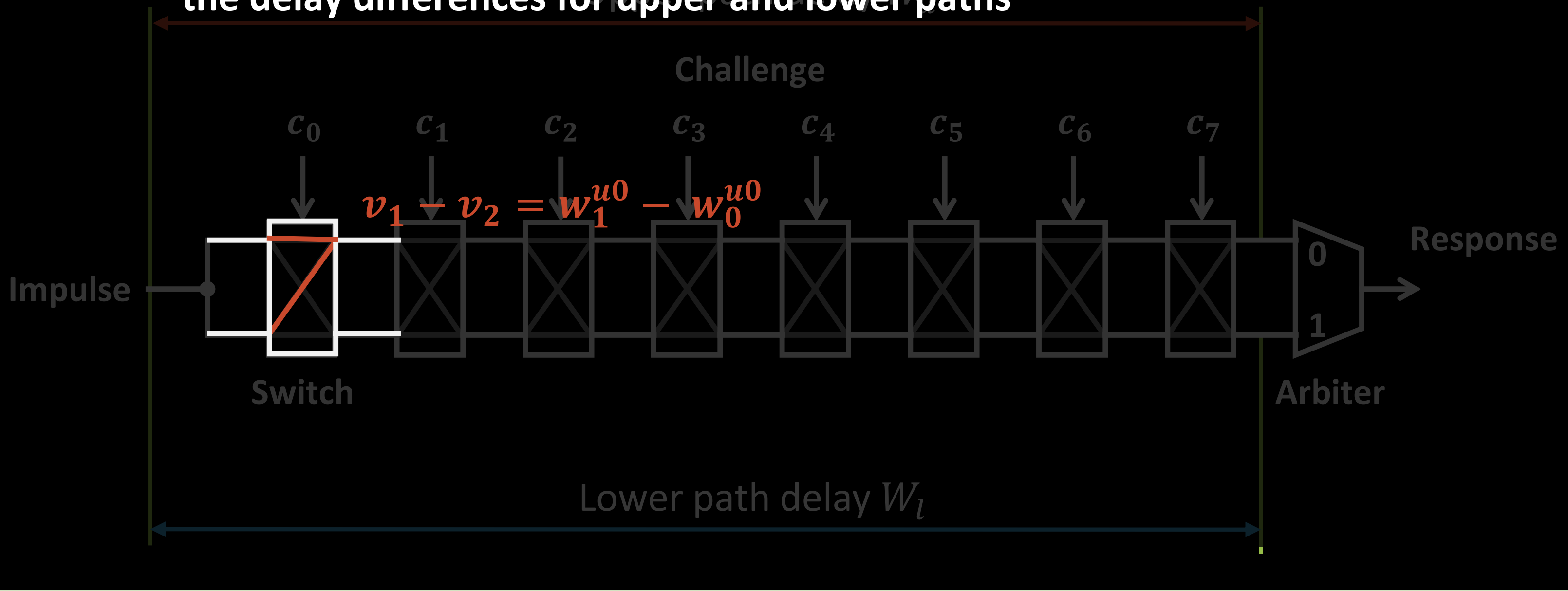
Characterize each switch box in the Arbiter PUF by calculating the delay differences for upper and lower paths



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

Physical Attacks: Example: [Tajik et al., CHES'14]

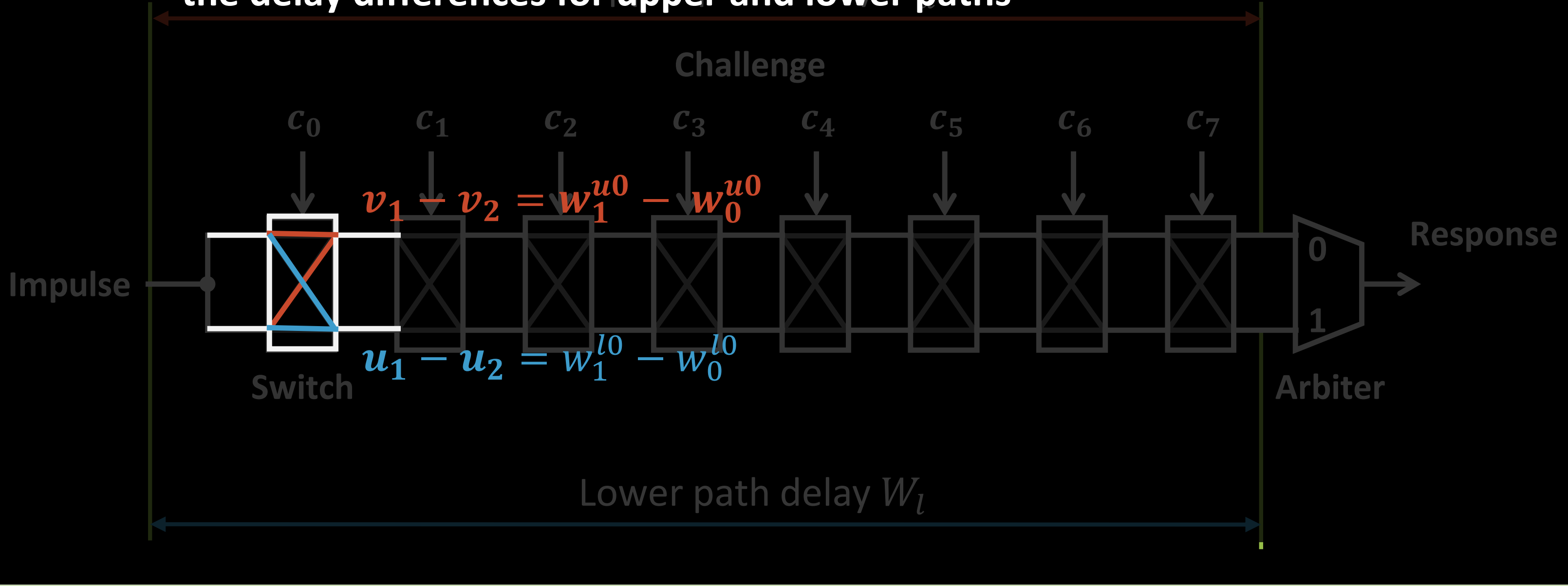
Characterize each switch box in the Arbiter PUF by calculating the delay differences for upper and lower paths



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

Physical Attacks: Example: [Tajik et al., CHES’14]

Characterize each switch box in the Arbiter PUF by calculating the delay differences for upper and lower paths



C	0x00	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
W_u	v_1	v_2							
W_l	u_1	u_2							

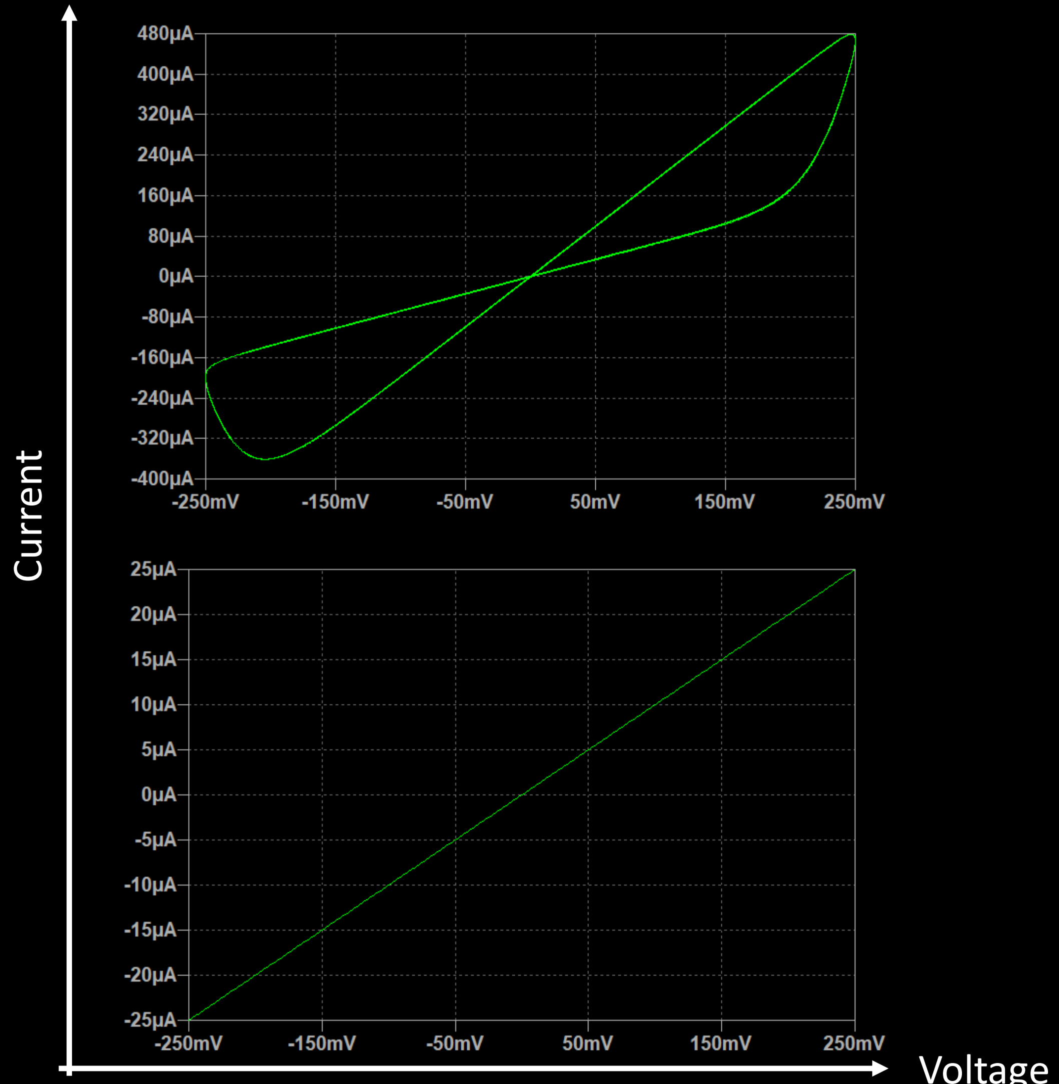
Beyond CMOS-based PUFs

CMOS-based PUFs exhibit linear behavior => vulnerable to machine learning

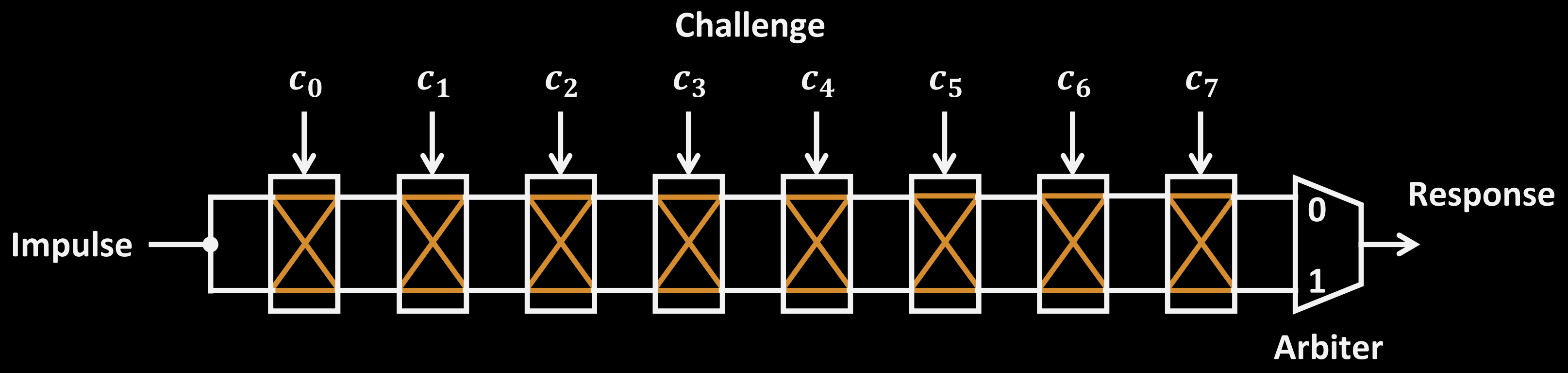
One Solution: Add components with non-linear behavior to complicate/escape machine learning attacks, e.g., Memristors

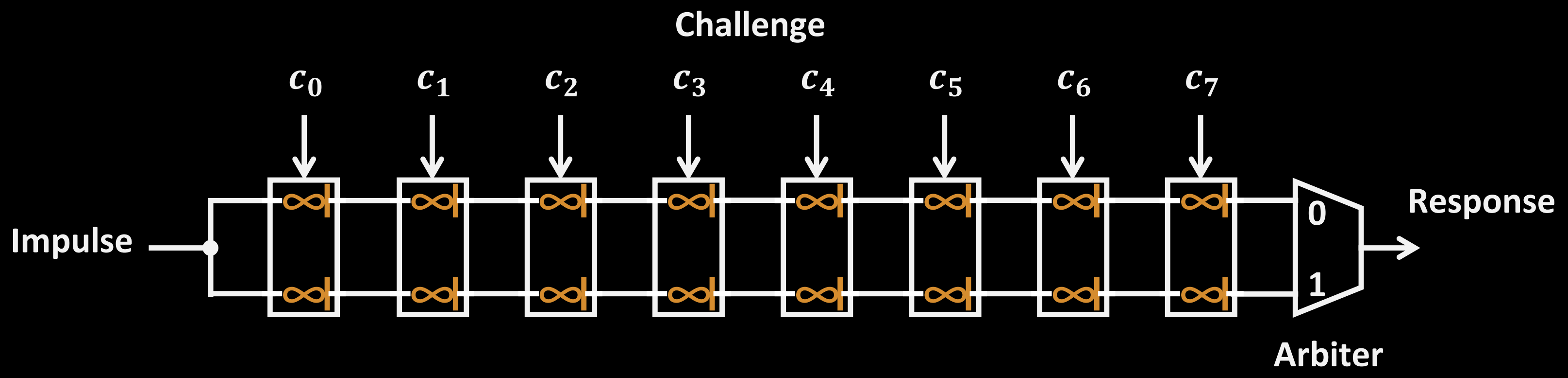
Memristors

- A resistor that changes it resistance as voltage is applied
- Applications:
  - Oscillators
  - Learners (Neural Networks)
  - Memories
  - PUFs!
- The top (bottom) figure shows Current-Voltage charcteristics of a memristor (resistor)

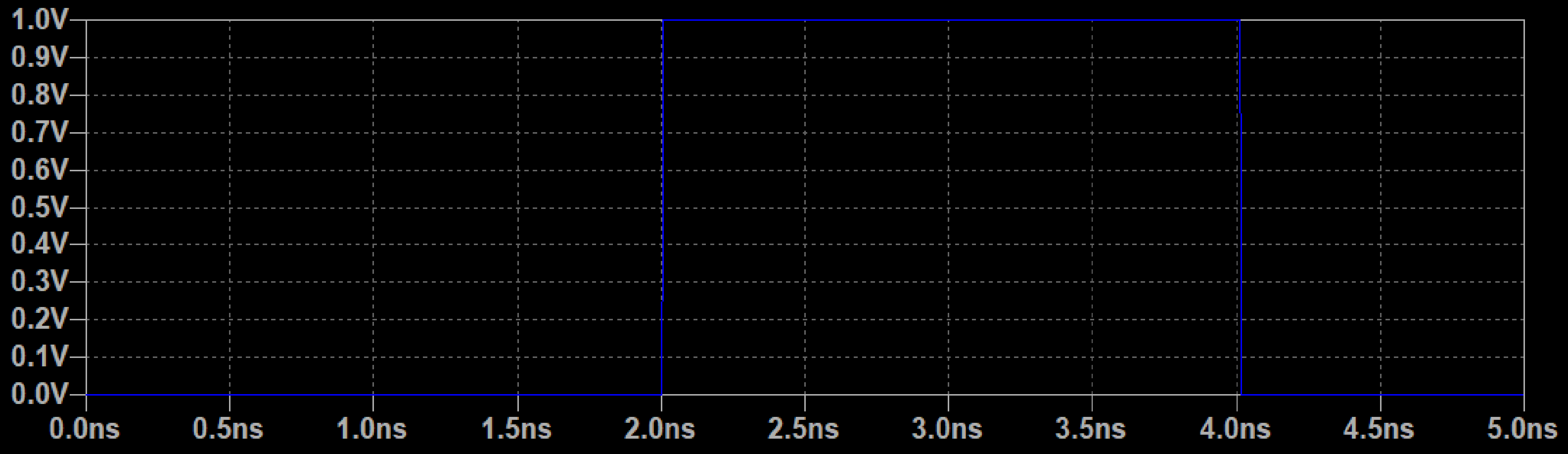


CMOS-based APUF vs. Memristor-based APUF

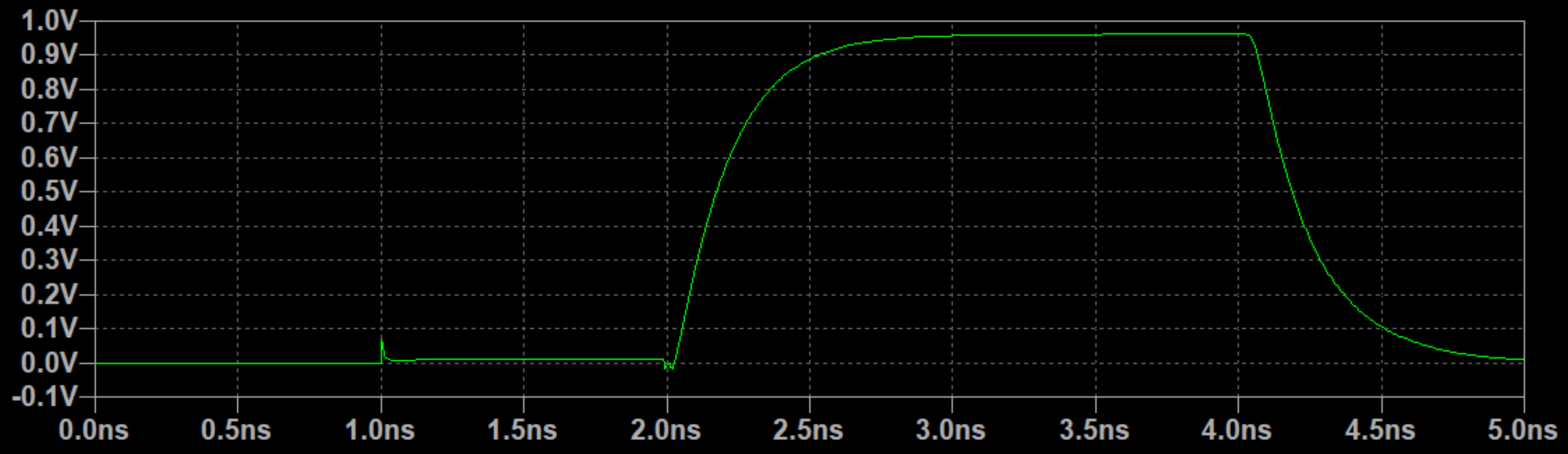




CMOS-based APUF vs. Memristor-based APUF



CMOS-based Arbiter PUF: Voltage at the upper path



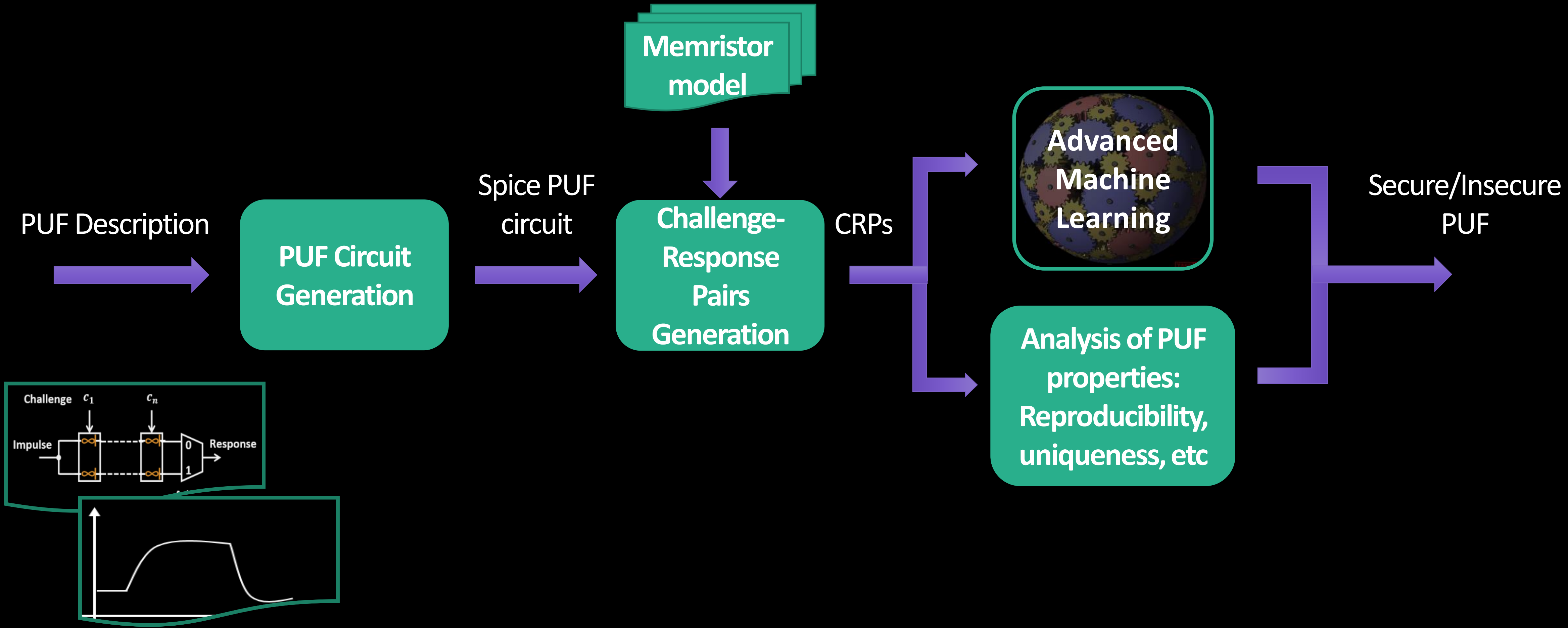
Memristor-based Arbiter PUF: Voltage at the upper path

Conclusion

- Many PUF designs, no unified security model
- Several successful attacks
  - Non-destructive physical attacks
  - Modeling attacks
- Designing secure PUFs is challenging?
  - What are the costs?
- PUFs based on advanced memory technologies
  - E.g., Memristors

Our Current Work:
Framework for Evaluation of Memristor-based PUFs

Framework for Evaluation of Memristor-based PUFs



PUF Description

PUF Circuit Generation

Spice PUF circuit

Memristor model

Challenge-Response Pairs Generation

CRPs

Advanced Machine Learning

Analysis of PUF properties: Reproducibility, uniqueness, etc

Secure/Insecure PUF

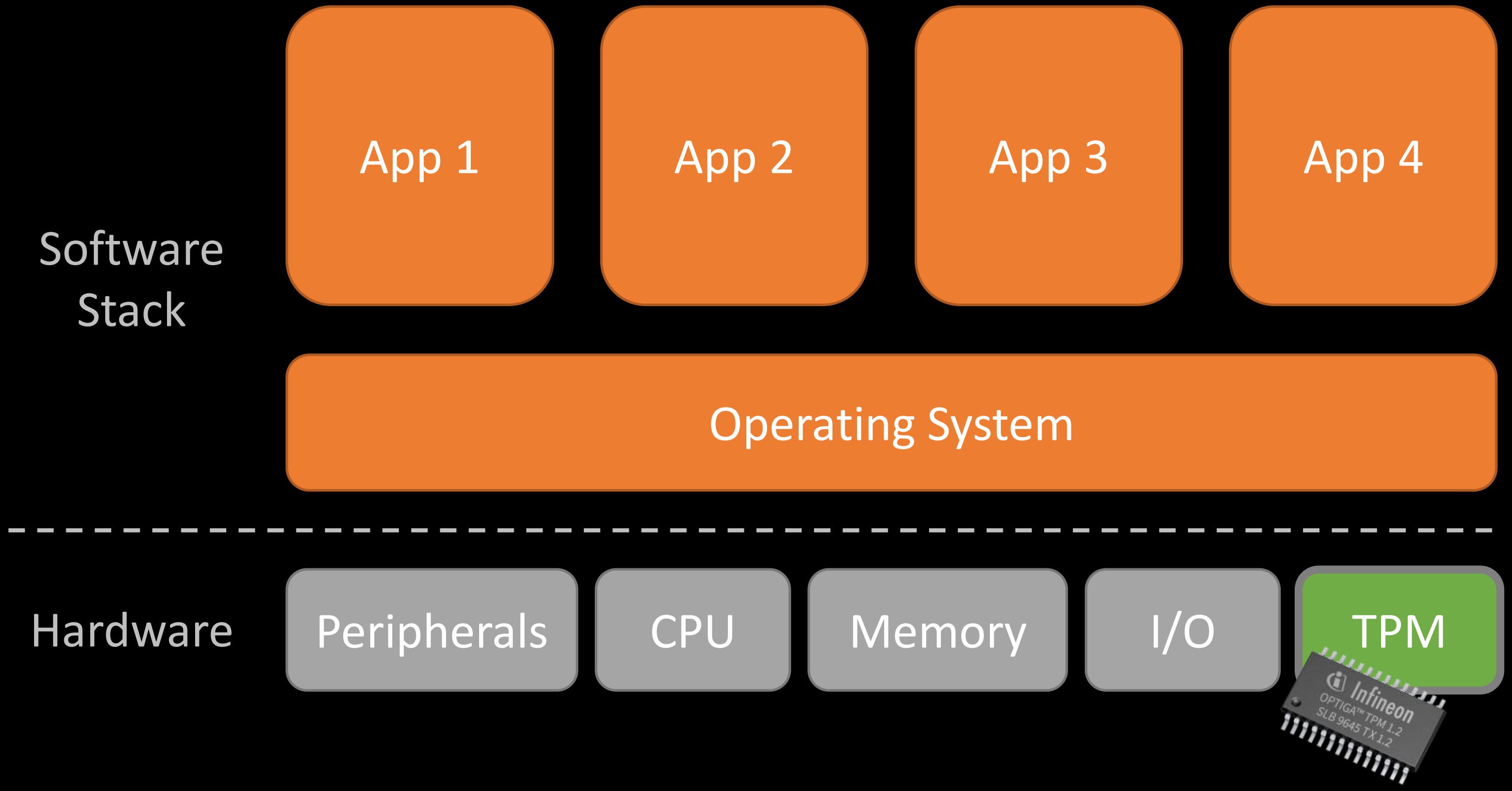
CYSEC
Cybersecurity
TU Darmstadt

Integrated Security Devices: The TPM Promise

Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

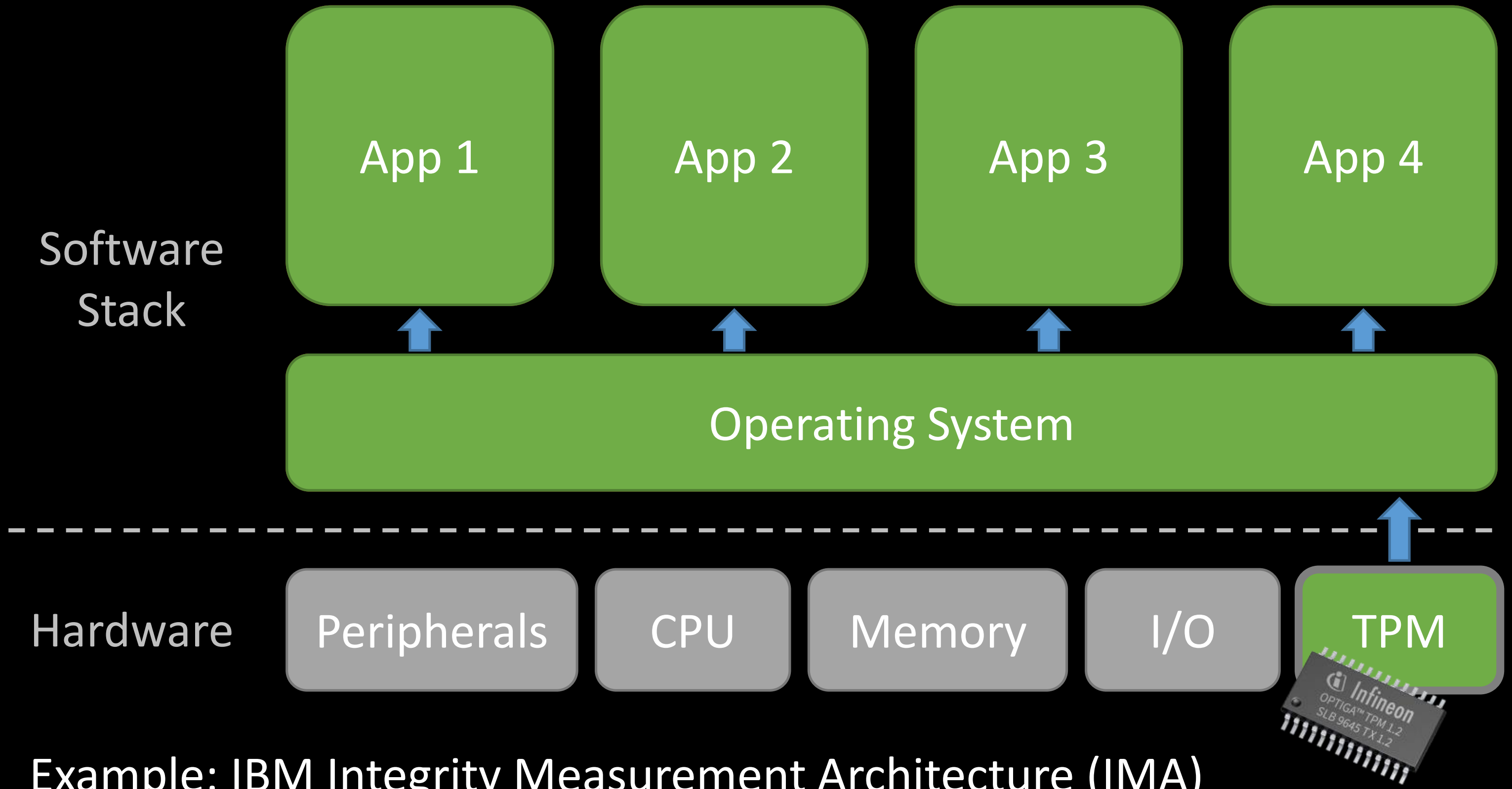
Trusted Computing

- Authenticated Boot and Attestation



Trusted Computing

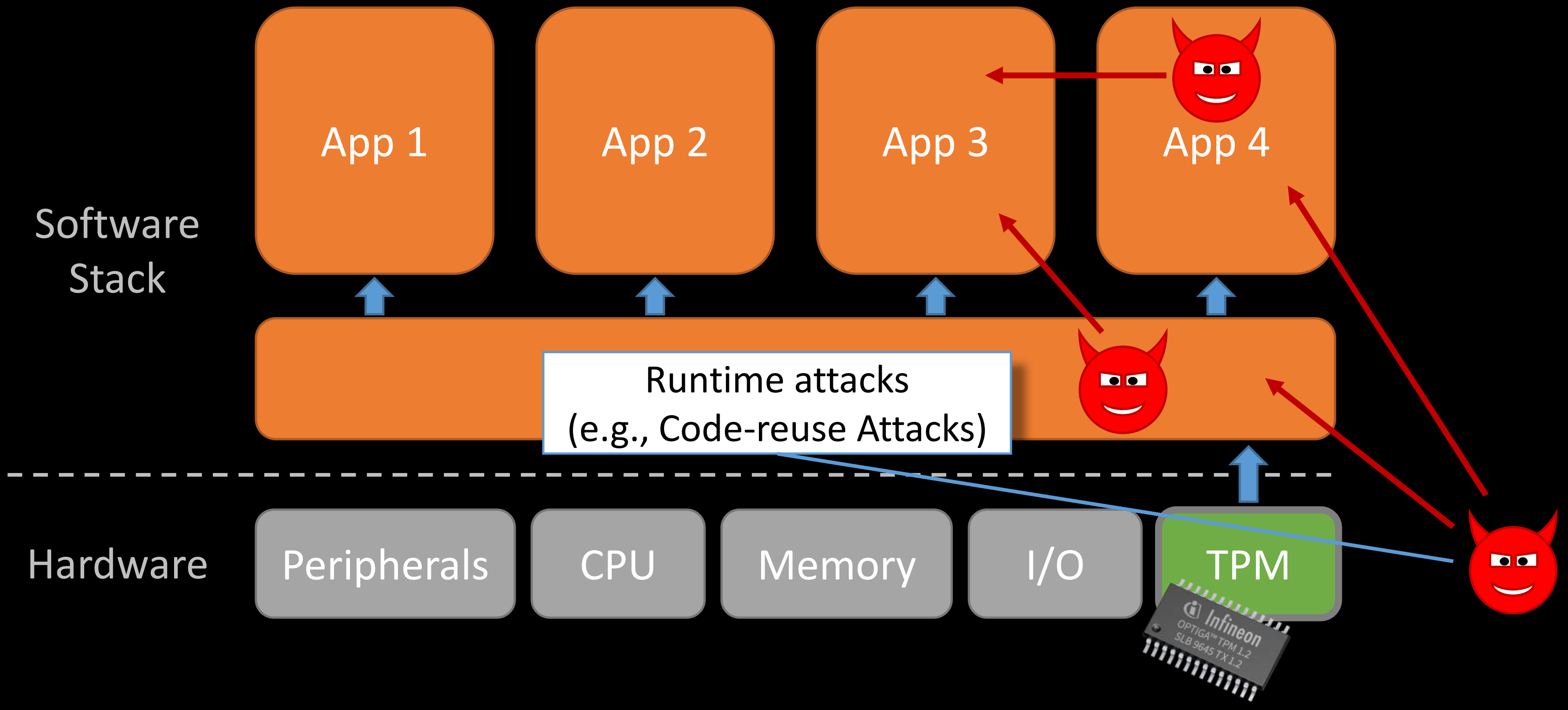
- Authenticated Boot and Attestation



Example: IBM Integrity Measurement Architecture (IMA)

Trusted Computing

- Authenticated Boot and Attestation



Summary: TPM-based Trusted Computing

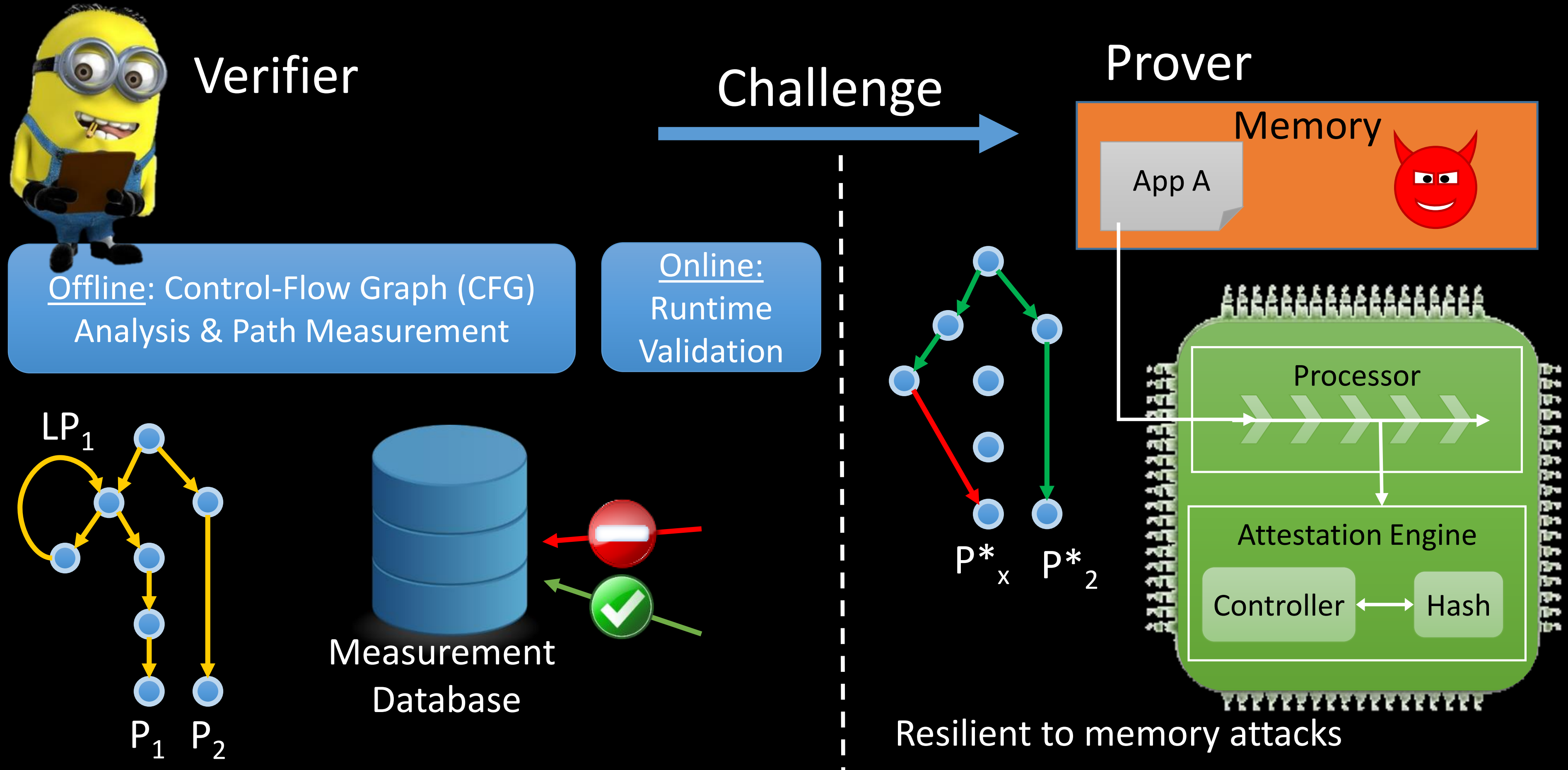
TPM assumptions and shortcomings

- Binary hashes express trustworthiness of code
  - Runtime attacks (e.g., code reuse) undermine this assumption
- Unforgeability of measurements
  - TPM 1.2 uses deprecated SHA1
- Protection against software attacks only
  - Hardware attacks on TPM

Our Current Work:
Control-Flow Attestation

Ongoing Work: Towards Run-time Attestation

- Control Flow Attestation [Davi et al, CCS 2016 & DAC 2017]



Verifier

Challenge

Prover

Memory

App A

Offline: Control-Flow Graph (CFG) Analysis & Path Measurement

Online: Runtime Validation

Processor

Attestation Engine

Controller

Hash

LP_1

P_1 P_2

Measurement Database

P^*_x P^*_2

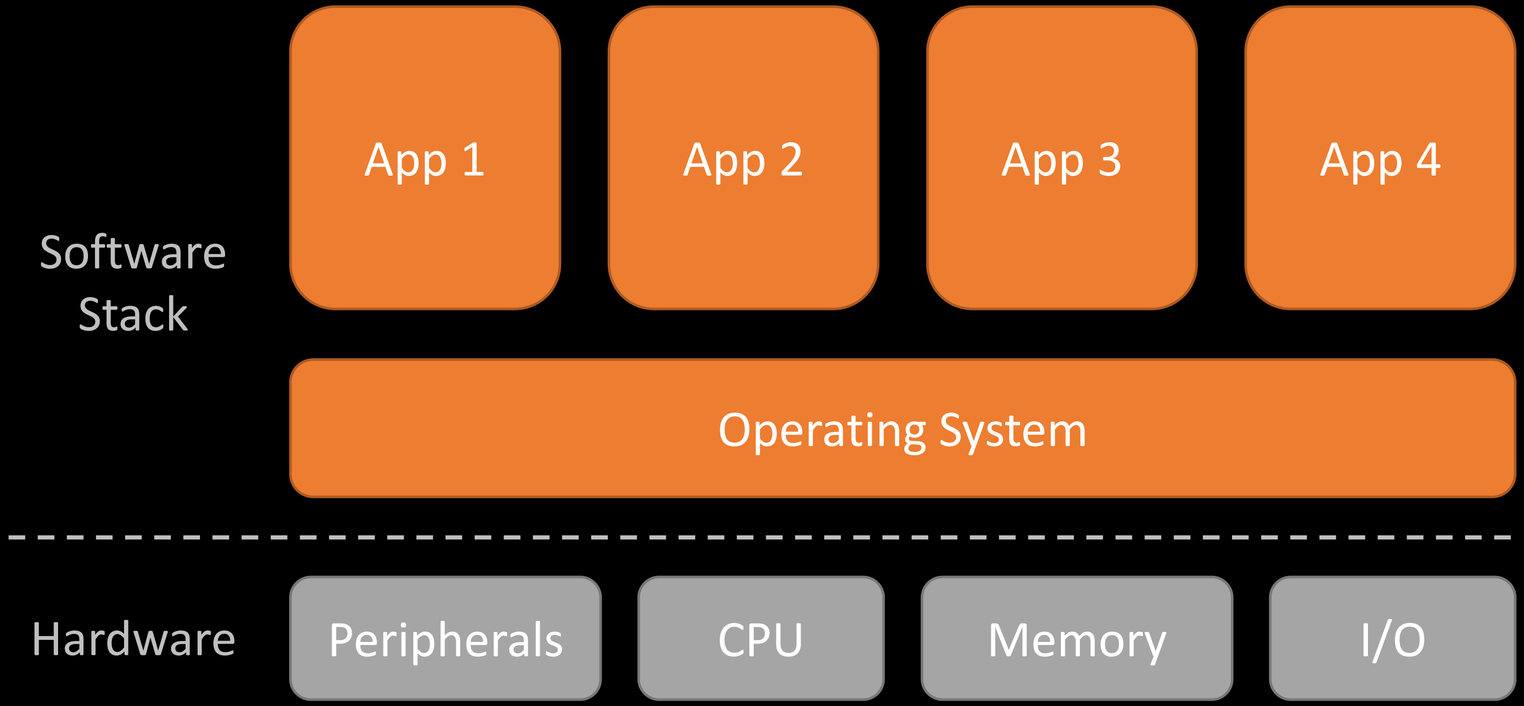
Resilient to memory attacks

Trusted Execution Environment (TEE)

ARM TrustZone

Assumptions:

- Apps in Secure World are trustworthy
- Normal World cannot influence Secure World



IMEI: International Mobile Equipment Identifier

ARM TrustZone

- Subsidy Lock
- IMEI Protection

iOS

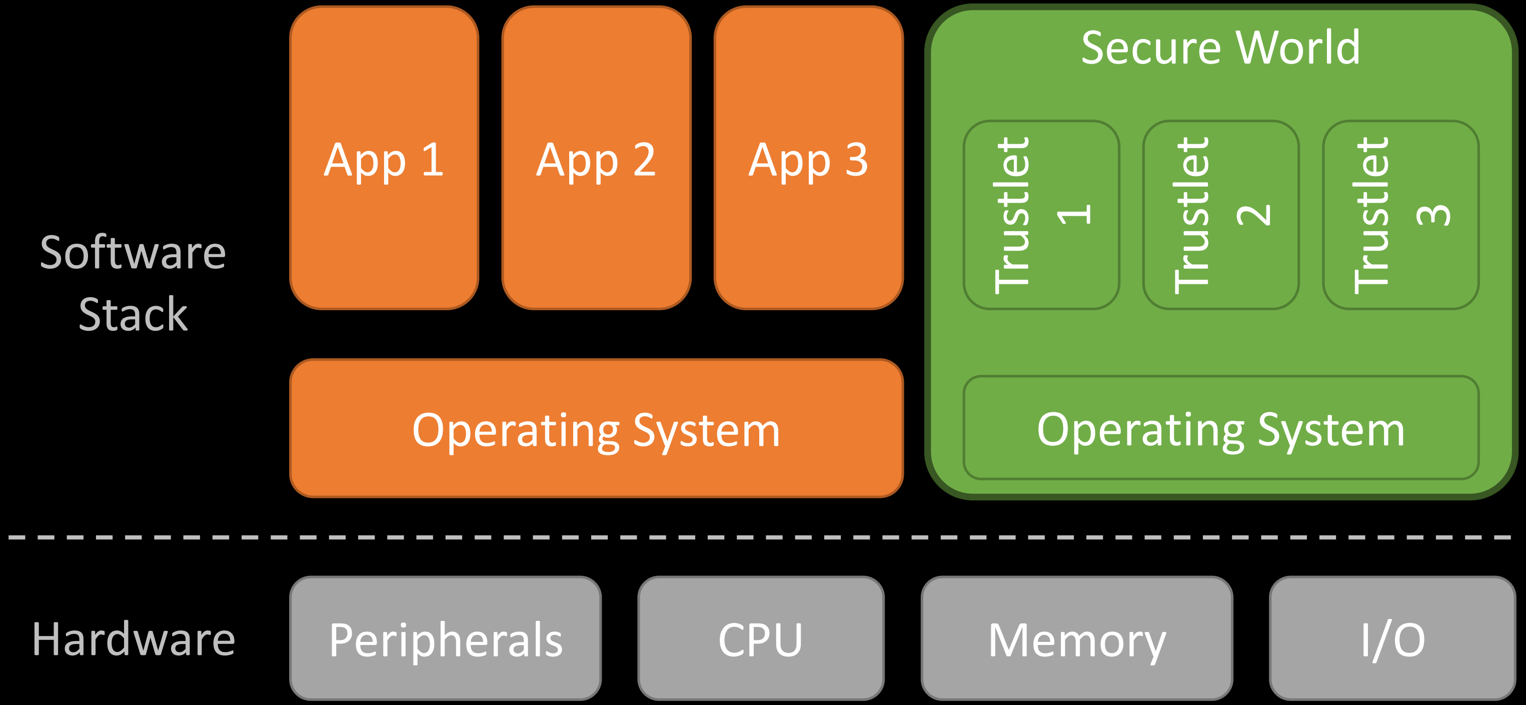
- Device Encryption
- Touch ID, Apple Pay

Android

- Full-Disk Encryption (FDE)
- Samsung KNOX
  - Secure-I/O, Attestation
  - Real-time Kernel Protection (TIMA)

Assumptions:

- Apps in Secure World are trustwor
- Normal World cannot influence Se



DRM

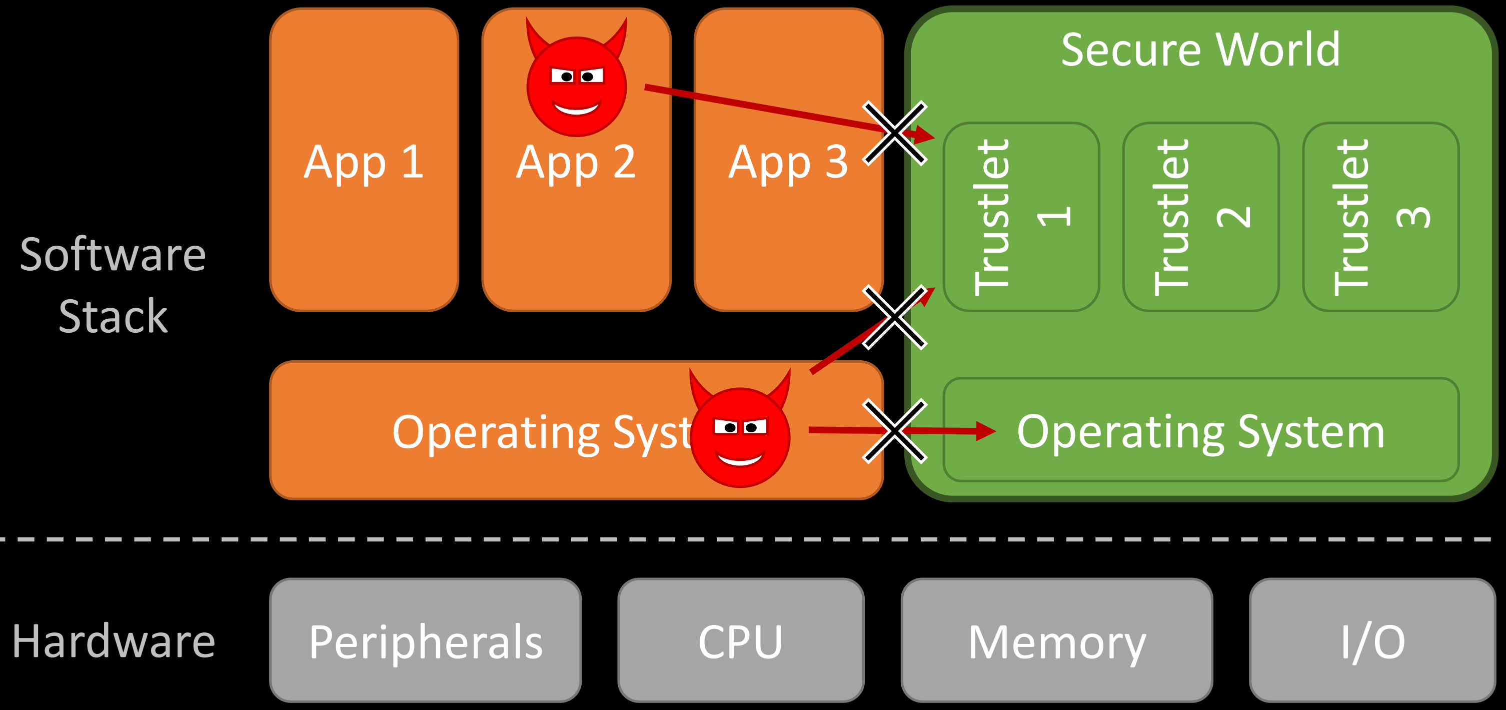
- Netflix
- Spotify
- Widevine

IMEI: International Mobile Equipment Identifier

ARM TrustZone

Assumptions:

- Apps in Secure World are trustworthy
- Normal World cannot influence Secure World

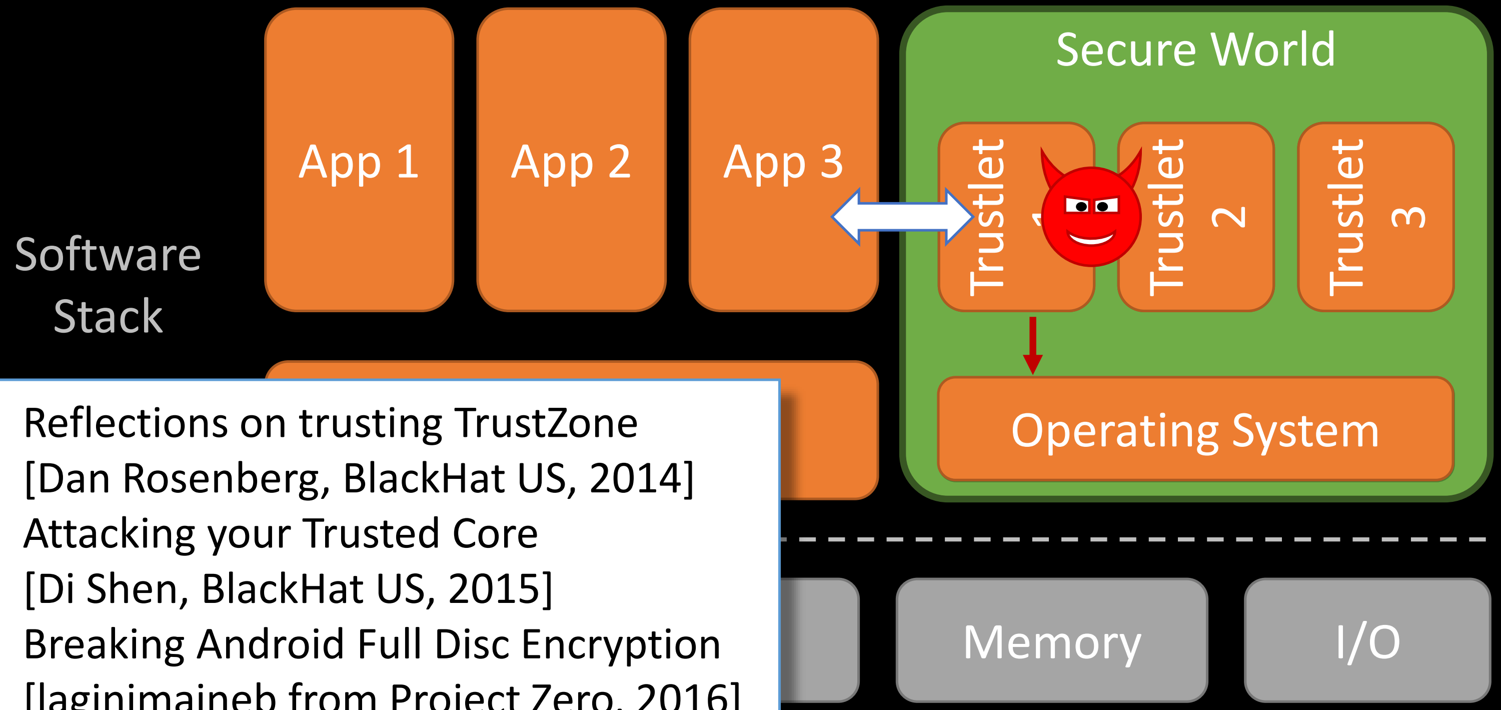


IMEI: International Mobile Equipment Identifier

ARM TrustZone

Assumptions:

- Apps in Secure World are trustworthy
- Normal World cannot influence Secure World



- Reflections on trusting TrustZone [Dan Rosenberg, BlackHat US, 2014]
- Attacking your Trusted Core [Di Shen, BlackHat US, 2015]
- Breaking Android Full Disc Encryption [laginimaineb from Project Zero, 2016]

IMEI: International Mobile Equipment Identifier

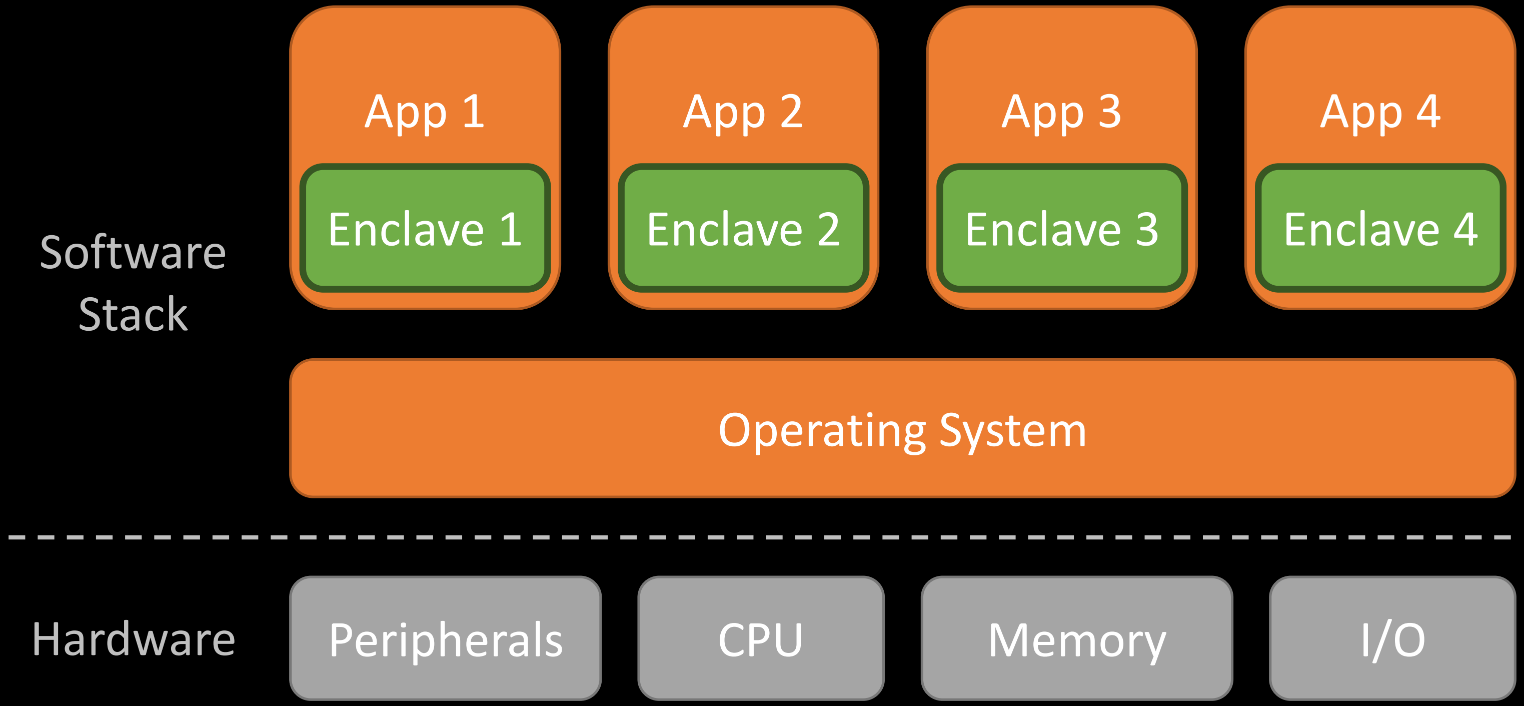
Summary: ARM TrustZone

- ARM TrustZone – Outdated?
  - Deployed for almost two decades
- Trusted computing for vendors and friends only
  - No access for app developer
- Many attacks have been shown over the last years

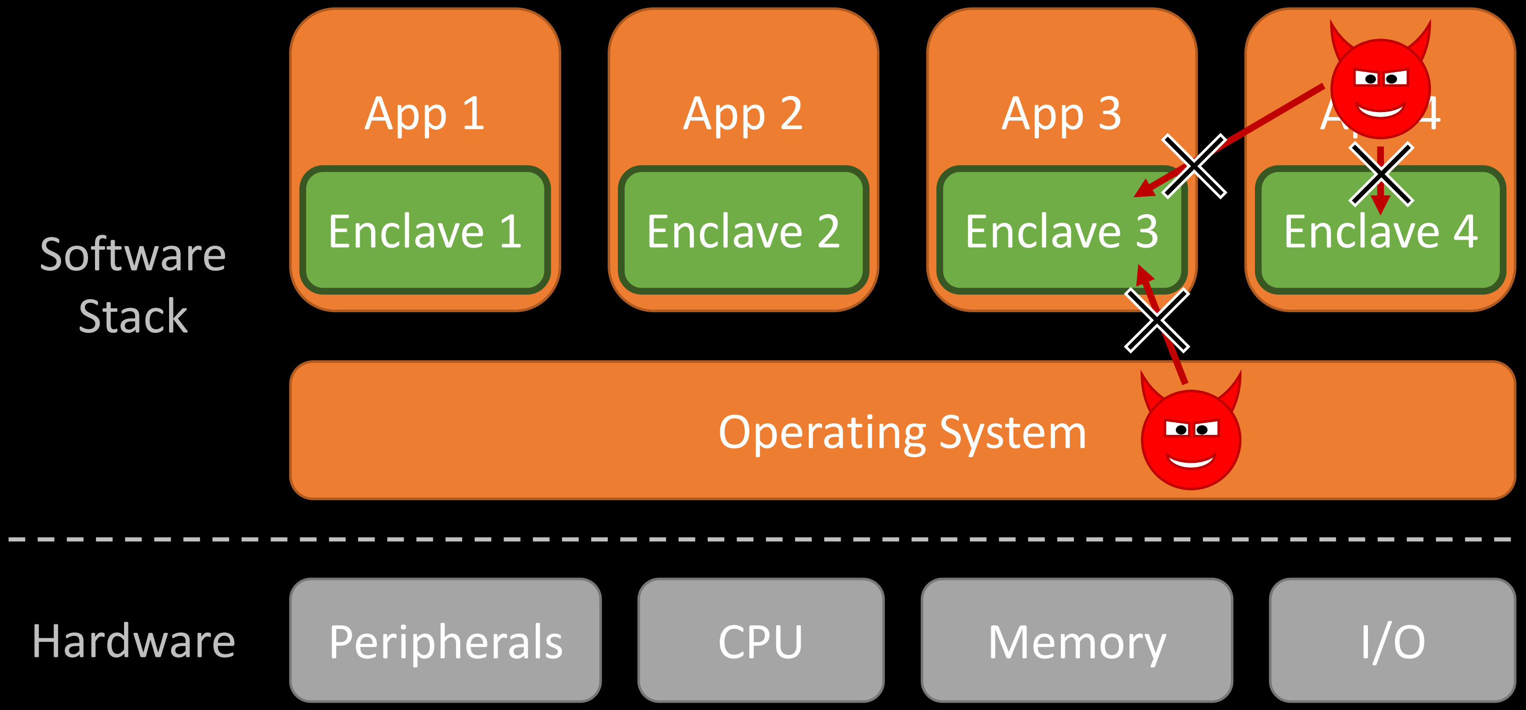
- On the positive side
  - Secure I/O

Our Current Work:
“Arbitrary” Number of TEEs in Normal World on ARM TZ

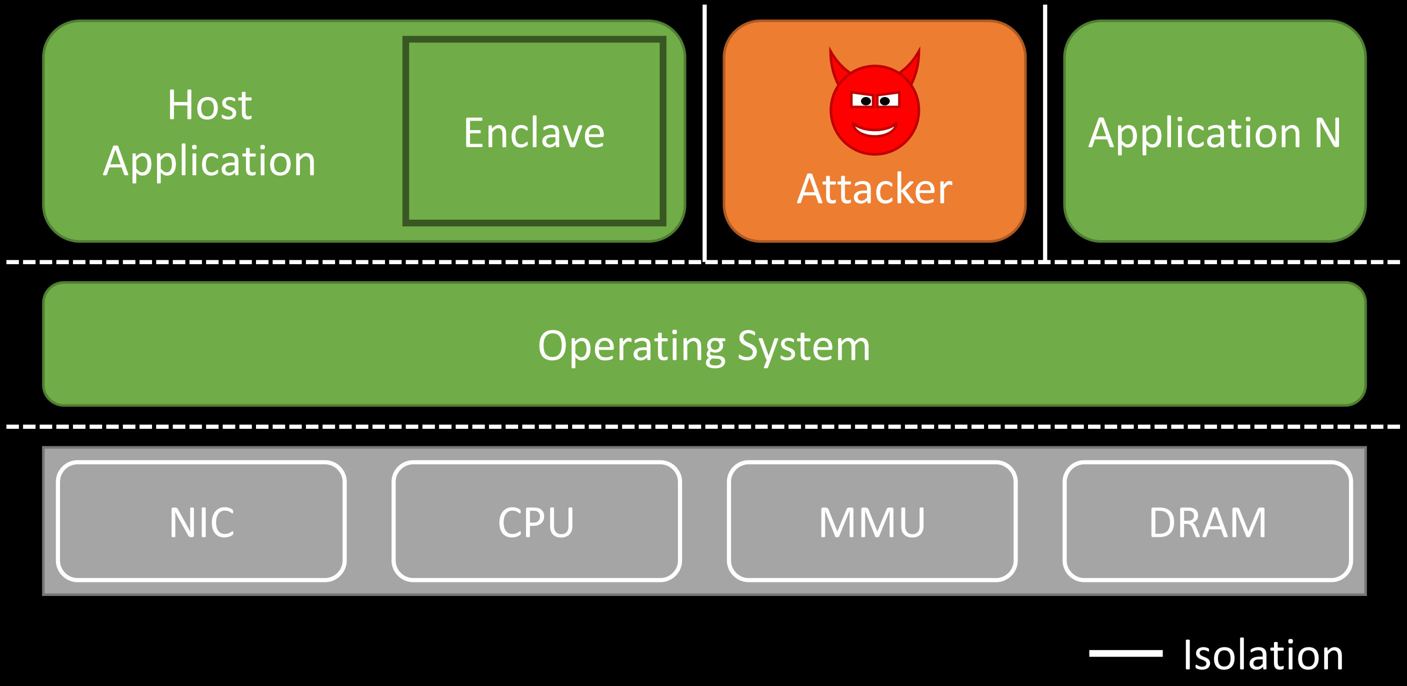
Intel Software Guard Extensions (SGX)



Intel Software Guard Extensions (SGX)

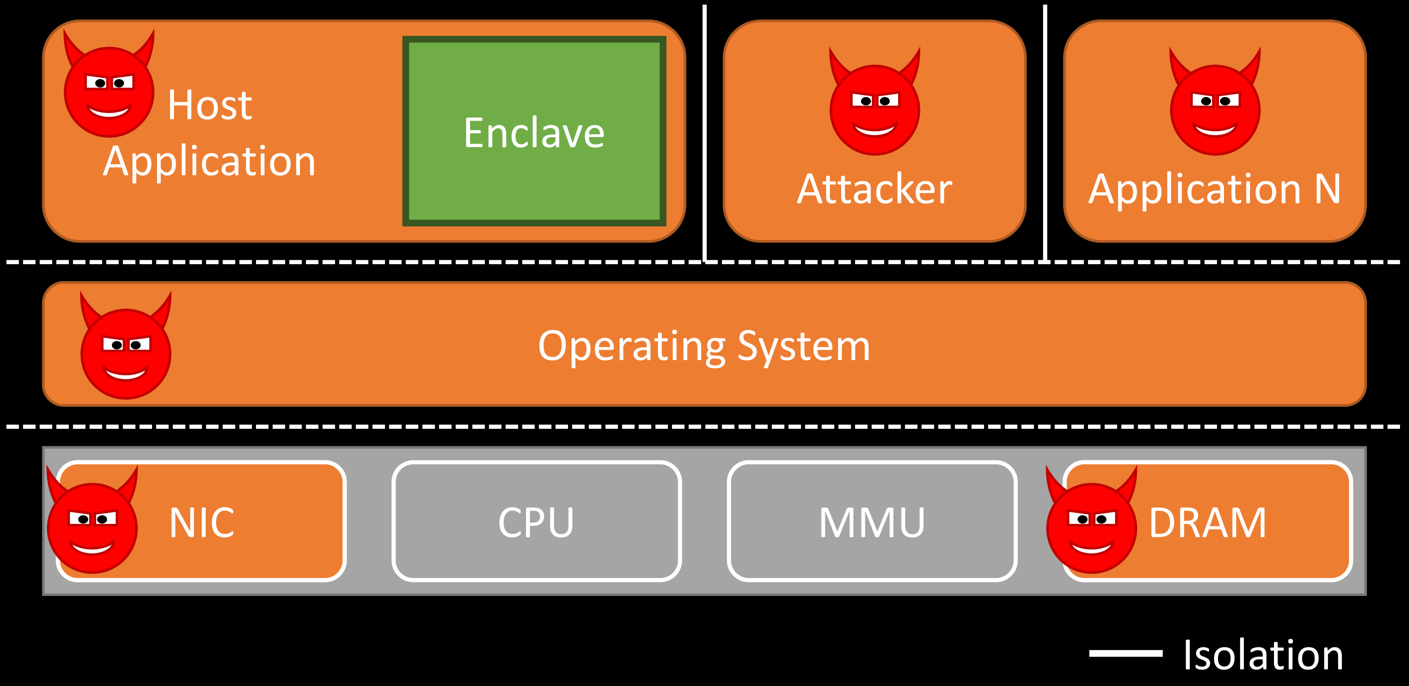


SGX (Adversary) Model



NIC: Network Interface Controller
MMU: Memory Management Unit

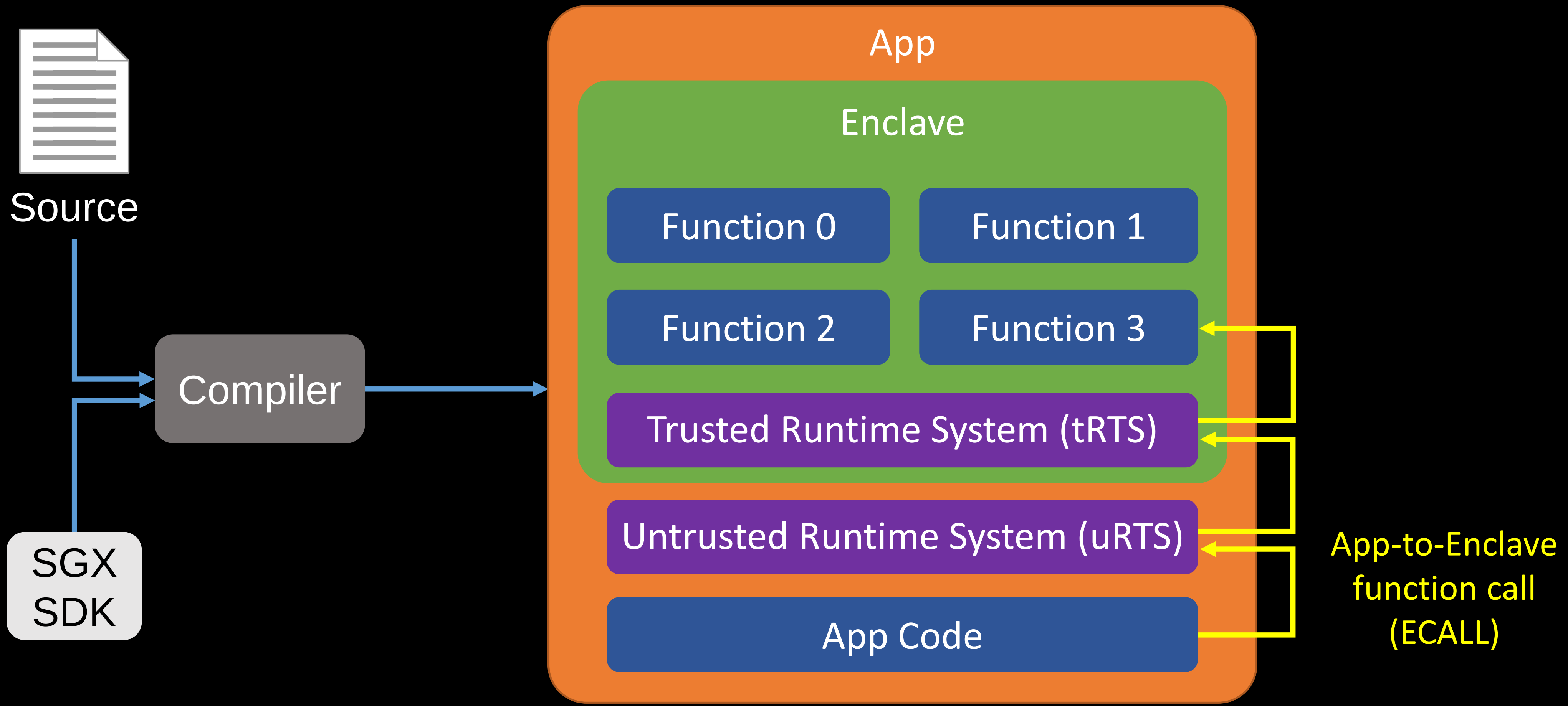
SGX (Adversary) Model



NIC: Network Interface Controller
MMU: Memory Management Unit

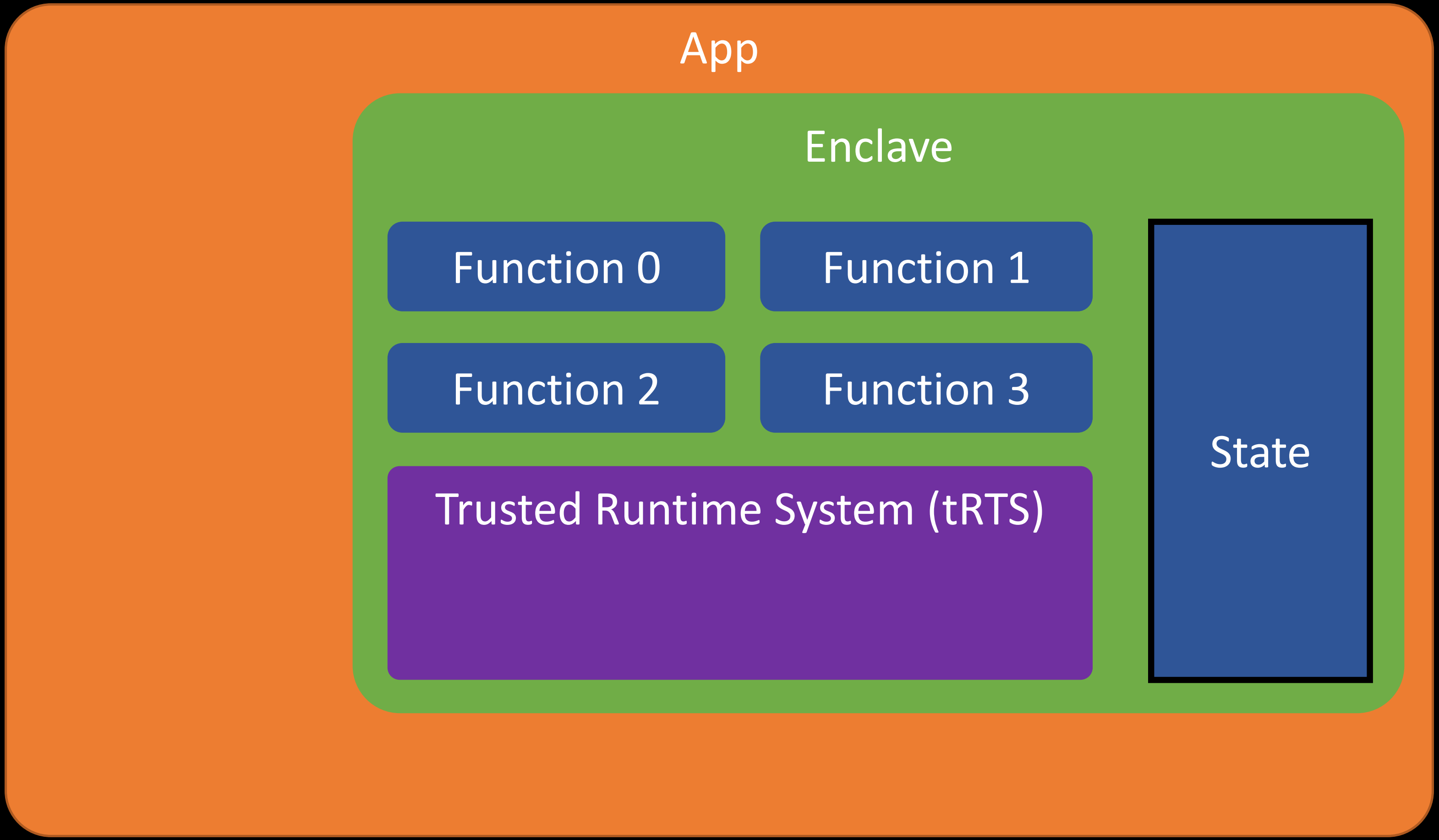
Run-time Attacks Inside the Enclave

SGX SDK and The Guard’s Dilemma


Source
SGX SDK
Compiler
App
Enclave
Function 0
Function 1
Function 2
Function 3
Trusted Runtime System (tRTS)
Untrusted Runtime System (uRTS)
App Code
App-to-Enclave function call (ECALL)

[Biondo et al., USENIX Sec. 2018]

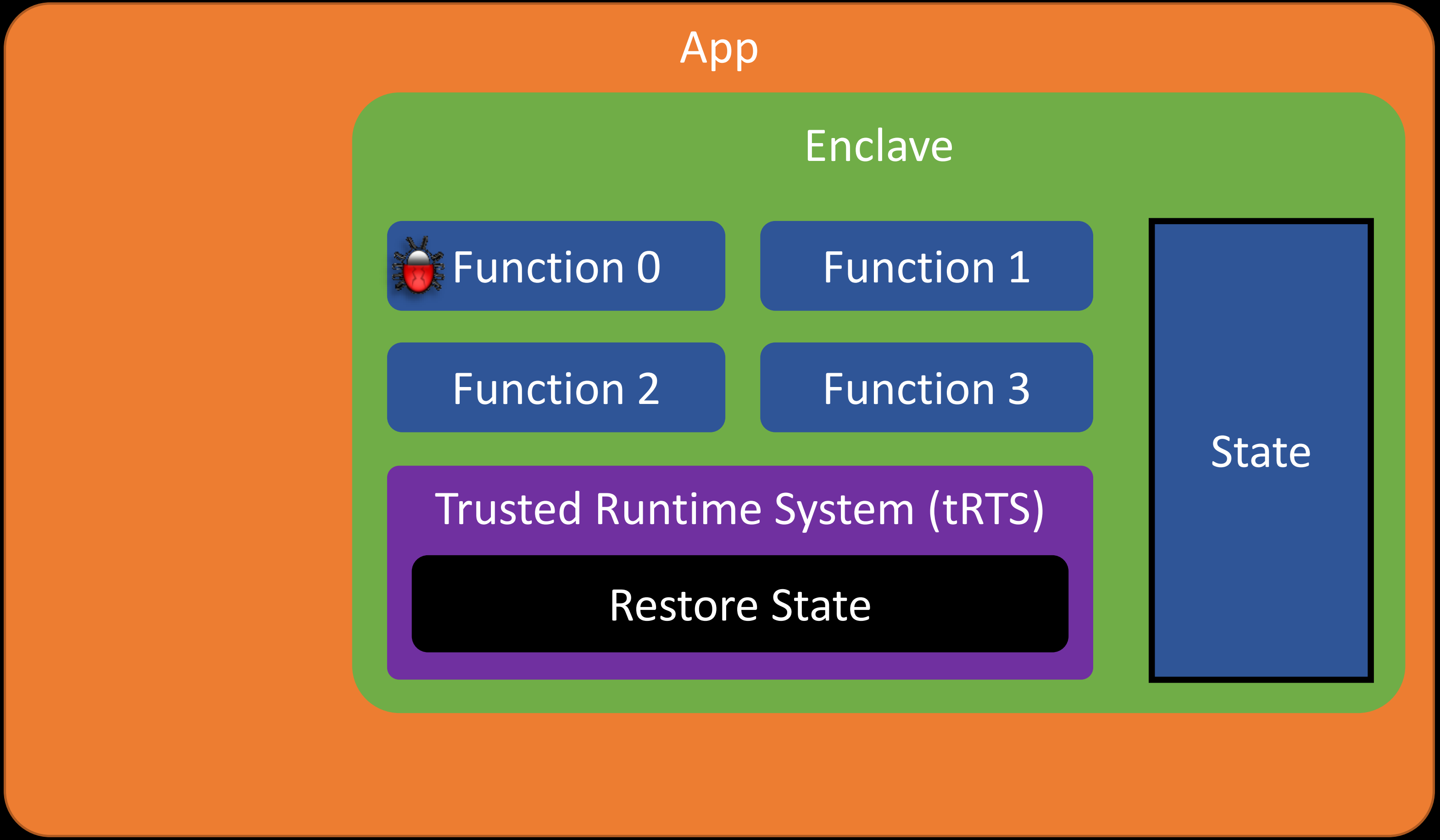
SGX SDK and The Guard’s Dilemma



[Biondo et al., USENIX Sec. 2018]

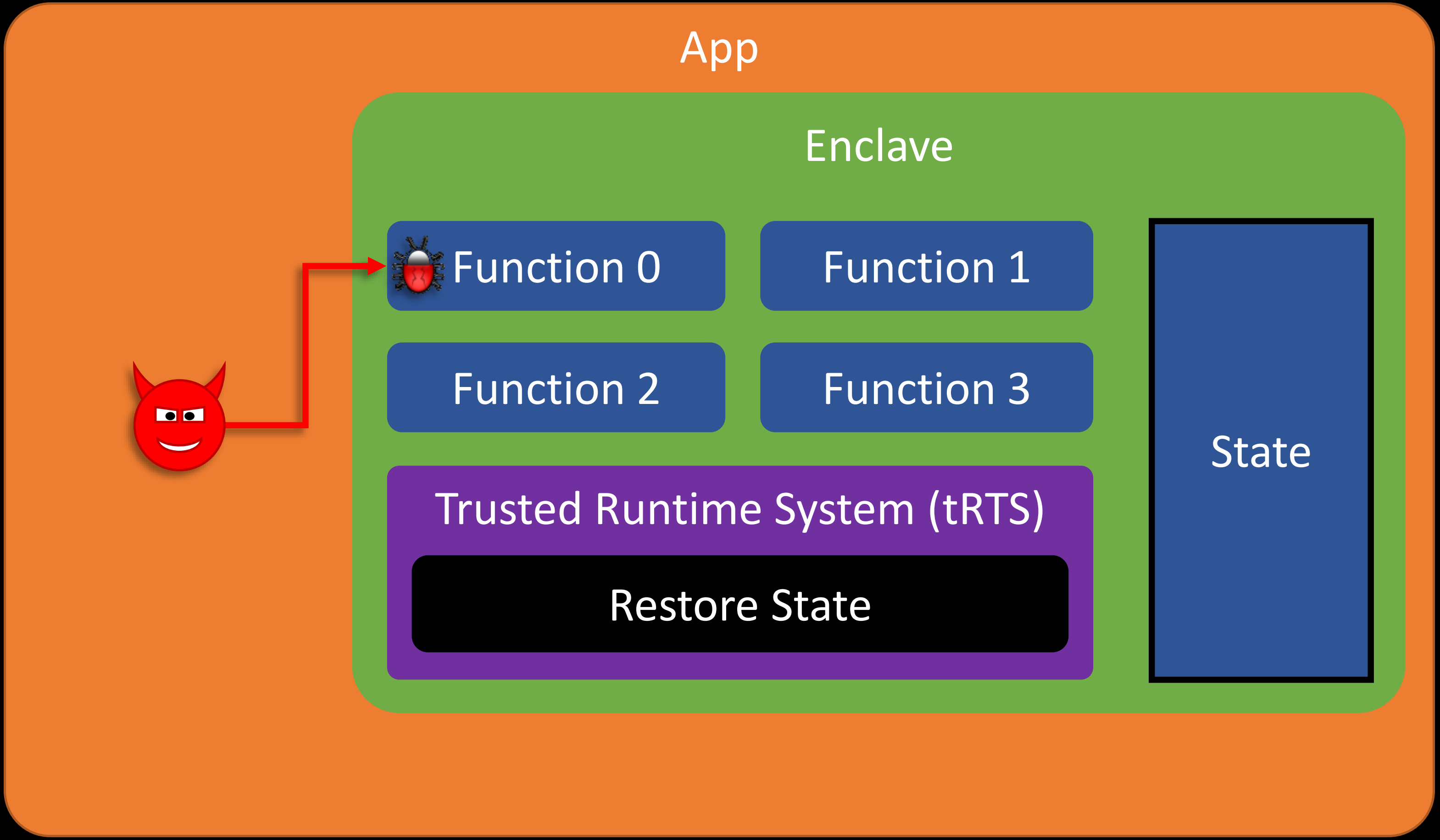
CYSEC
Cybersecurity
TU Darmstadt

SGX SDK and The Guard’s Dilemma



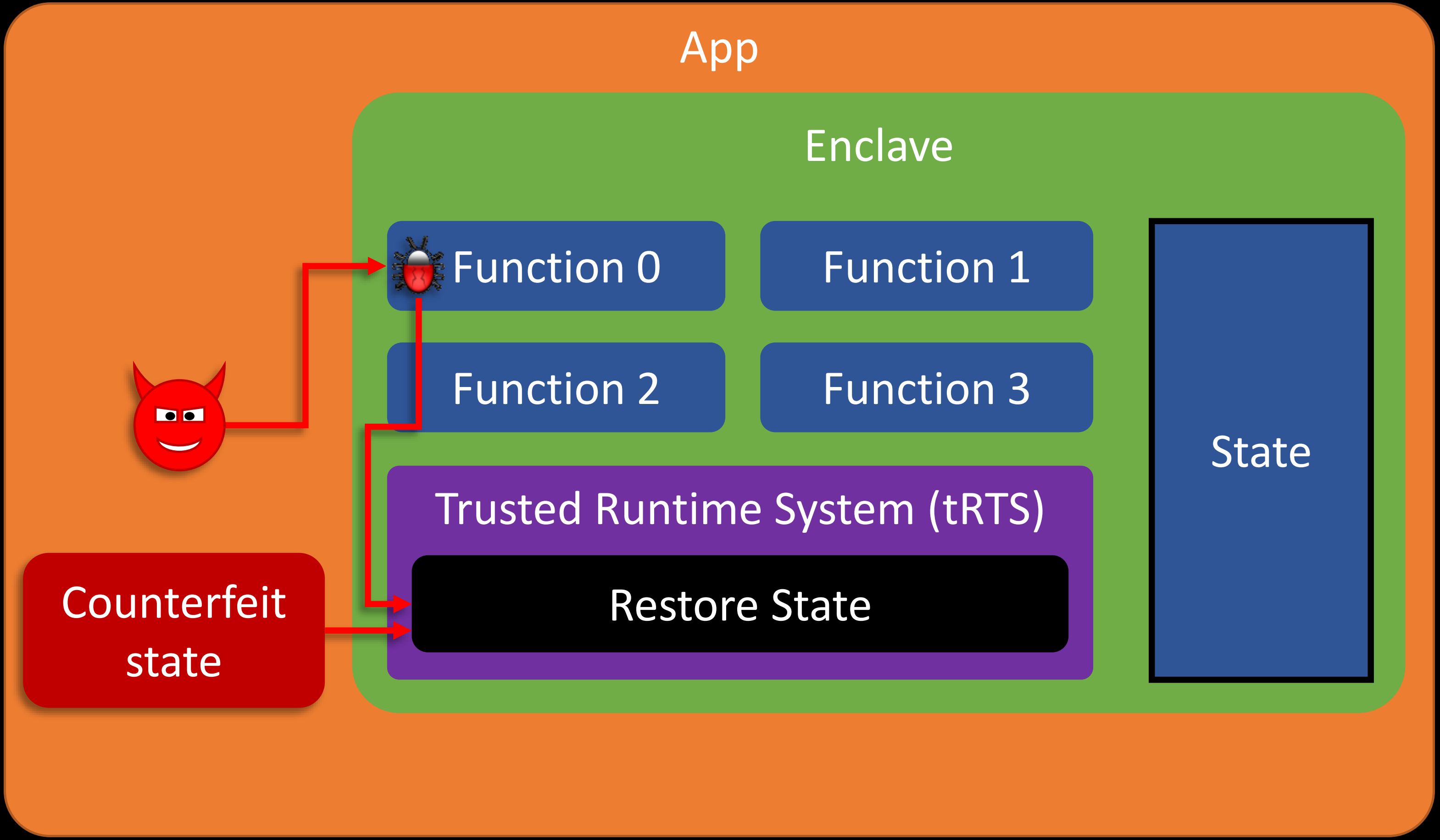
[Biondo et al., USENIX Sec. 2018]

SGX SDK and The Guard’s Dilemma



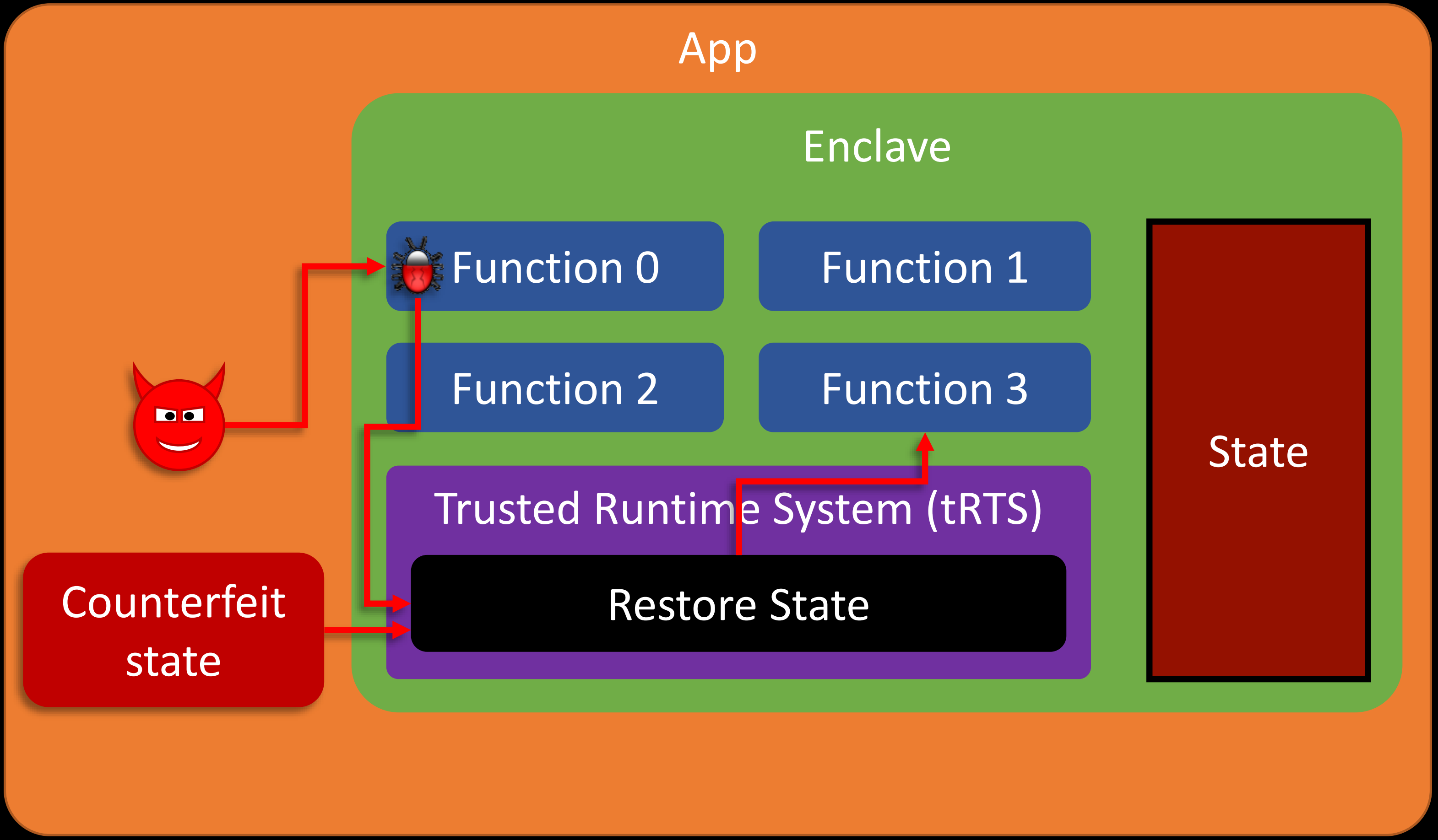
[Biondo et al., USENIX Sec. 2018]

SGX SDK and The Guard’s Dilemma



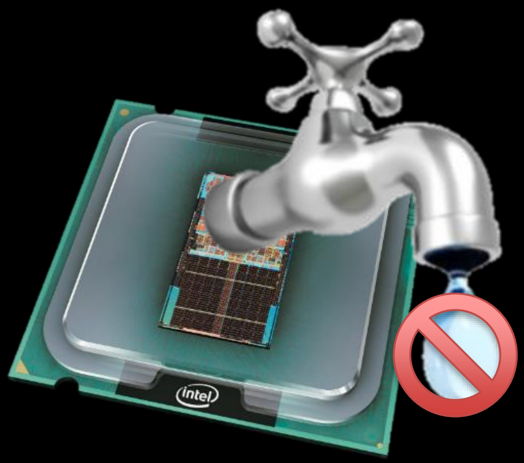
[Biondo et al., USENIX Sec. 2018]

SGX SDK and The Guard’s Dilemma



[Biondo et al., USENIX Sec. 2018]

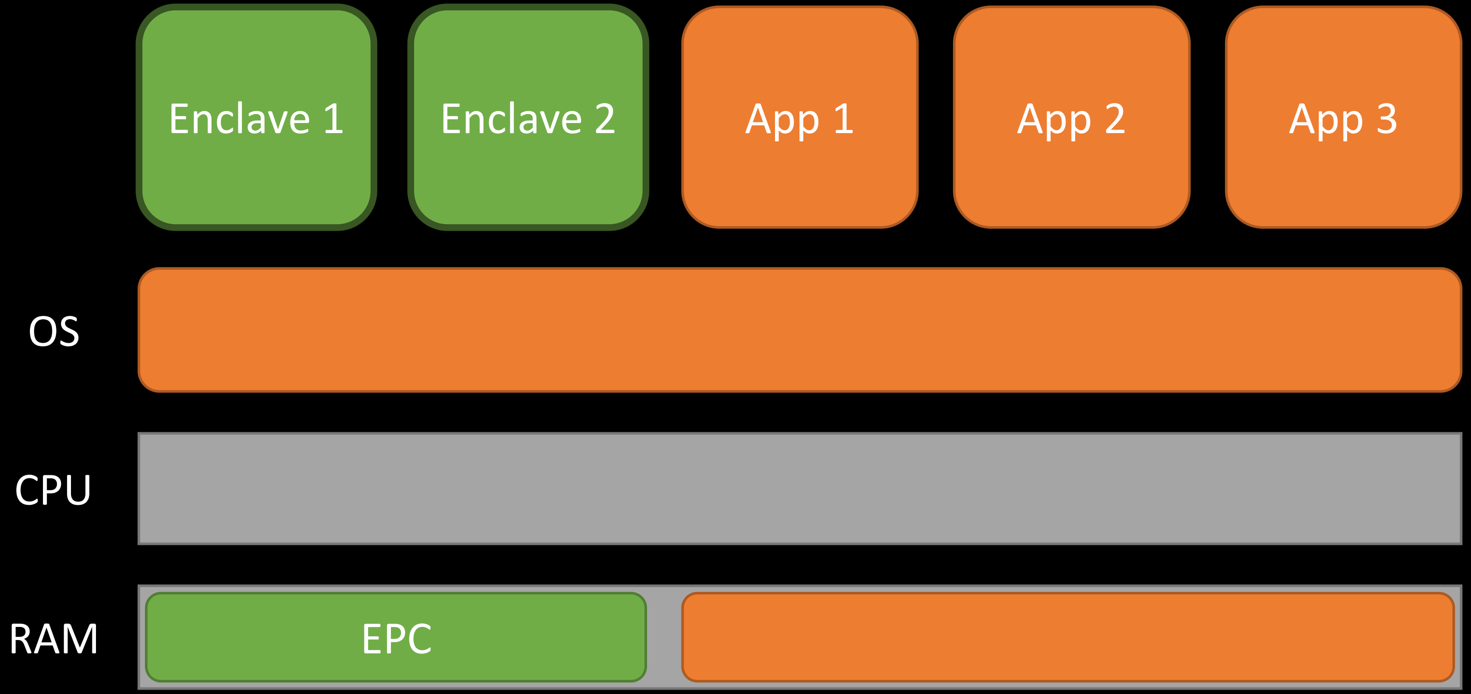
Leakage in Intel’s SGX



Summer School on real-world crypto and privacy, Šibenik (Croatia), June 11–15, 2018

Page Fault Attacks on SGX

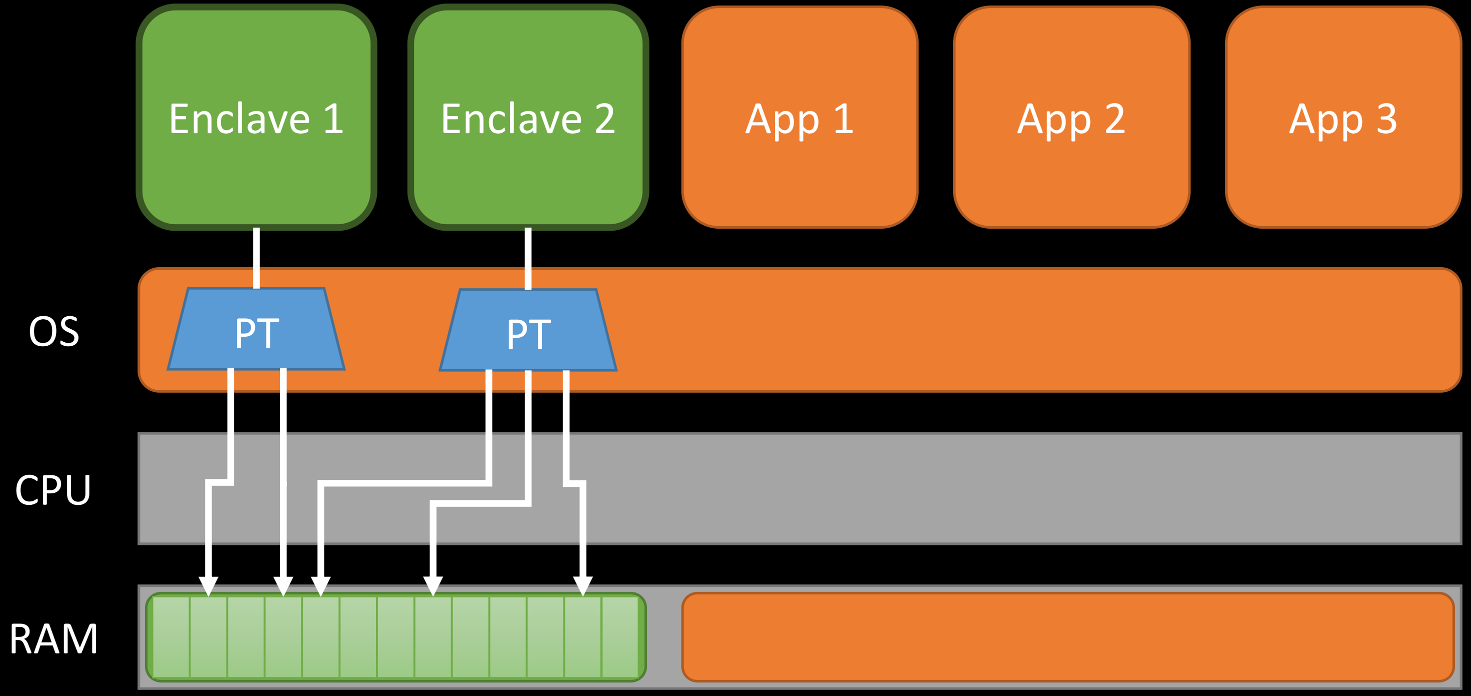
Granularity: page 4K, good for big data structures



EPC: Enclave Page Cache
PT: Page Tables
PF: Page-Fault

Page Fault Attacks on SGX

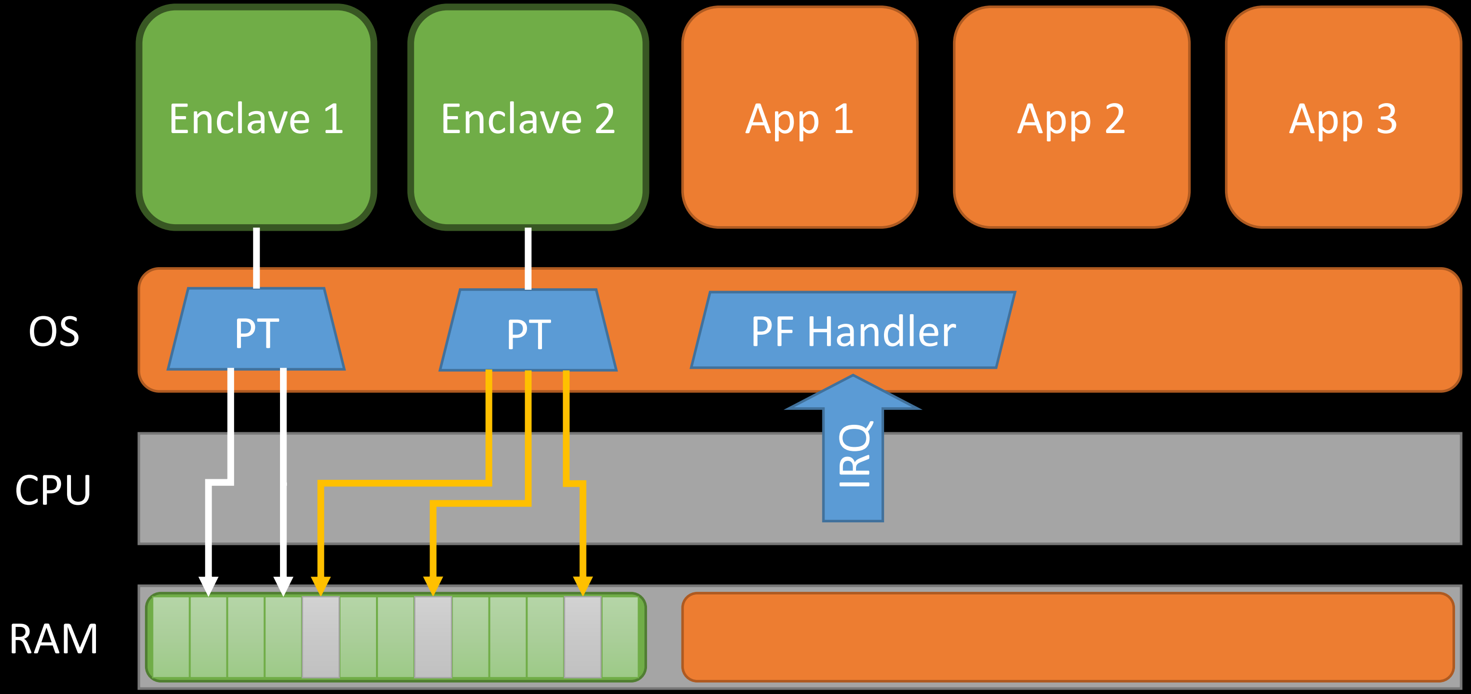
Granularity: page 4K, good for big data structures



EPC: Enclave Page Cache
PT: Page Tables
PF: Page-Fault

Page Fault Attacks on SGX

Granularity: page 4K, good for big data structures



EPC: Enclave Page Cache
PT: Page Tables
PF: Page-Fault

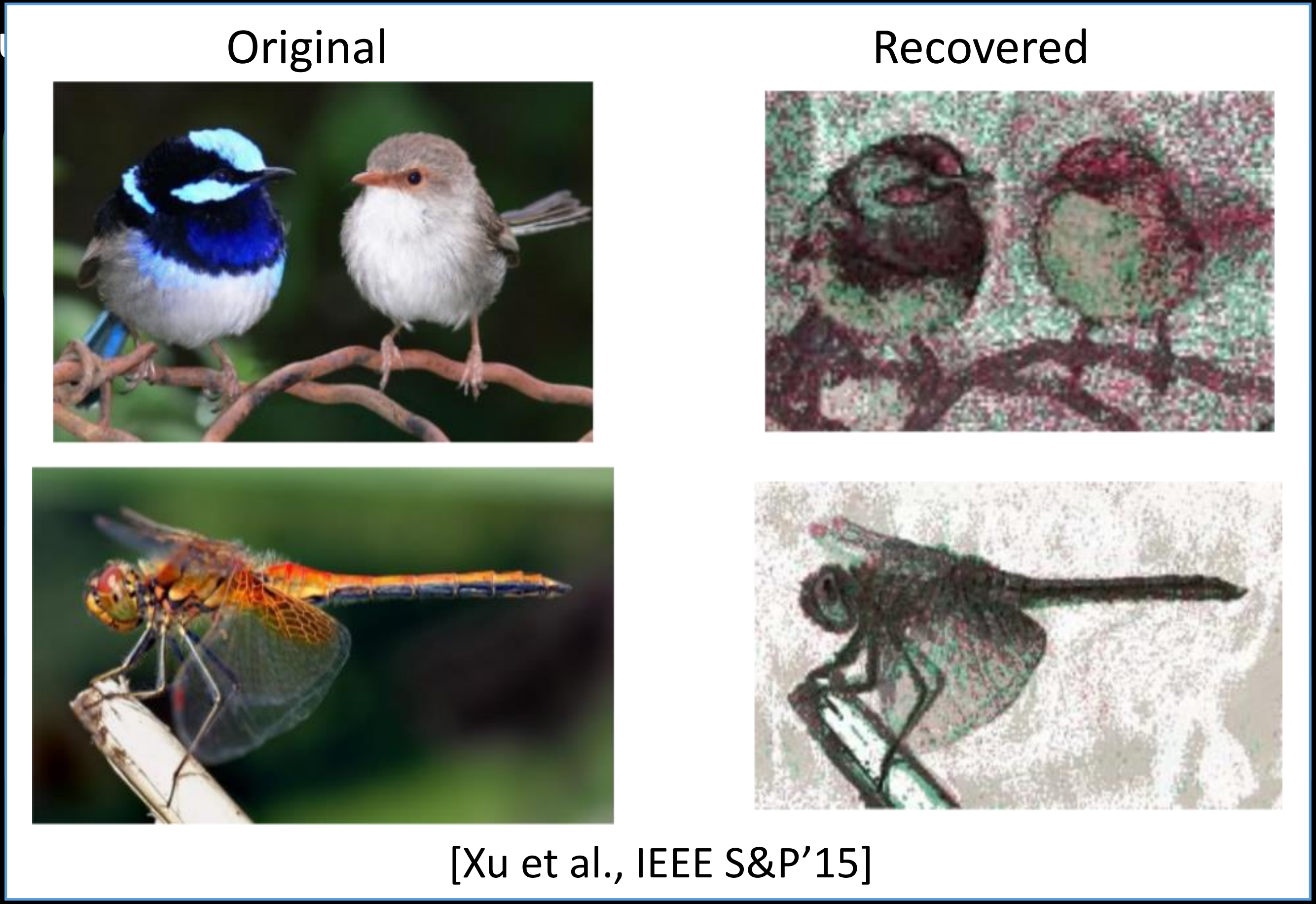
Page Fault Attacks on SGX

Granu

OS

CPU

RAM

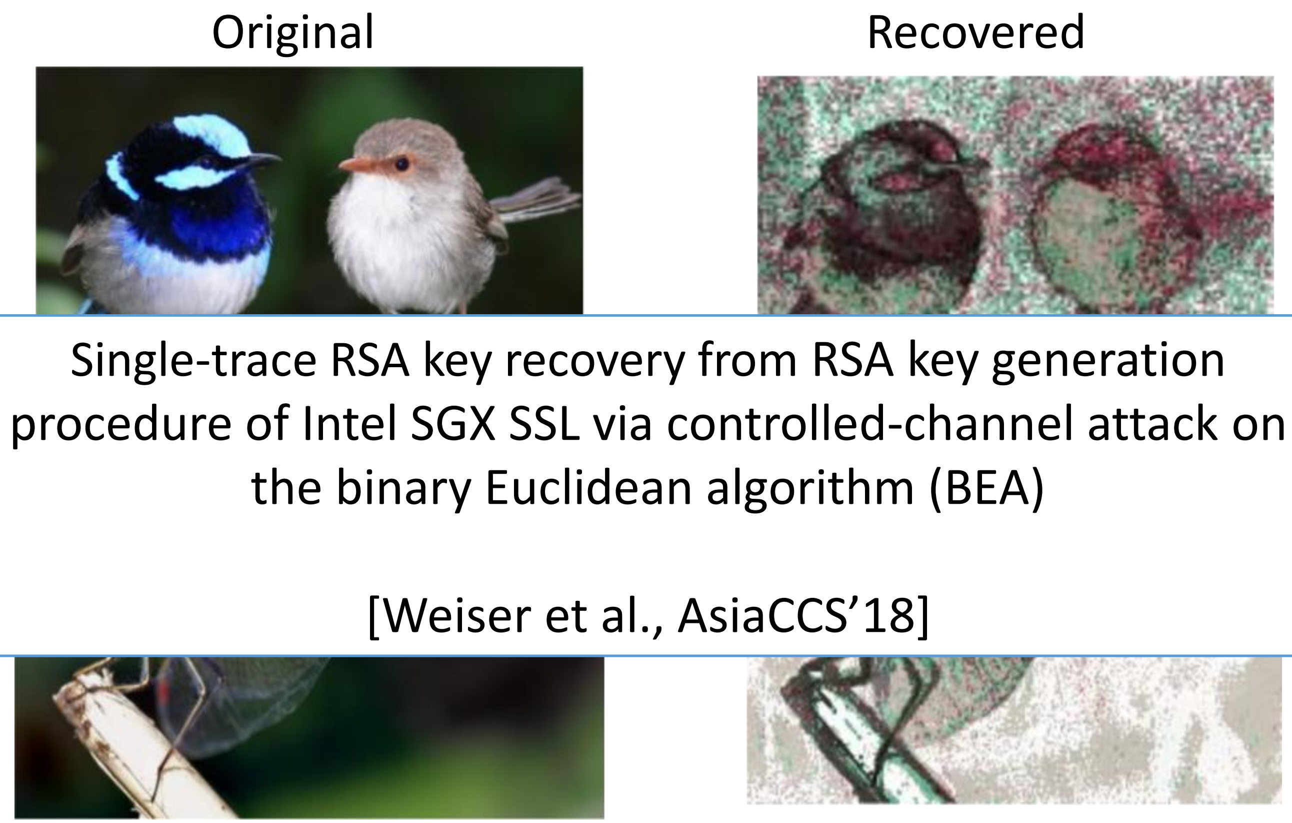


[Xu et al., IEEE S&P’15]

EPC: Enclave Page Cache
PT: Page Tables
PF: Page-Fault

Page Fault Attacks on SGX

Gran



Single-trace RSA key recovery from RSA key generation procedure of Intel SGX SSL via controlled-channel attack on the binary Euclidean algorithm (BEA)

[Weiser et al., AsiaCCS'18]

OS

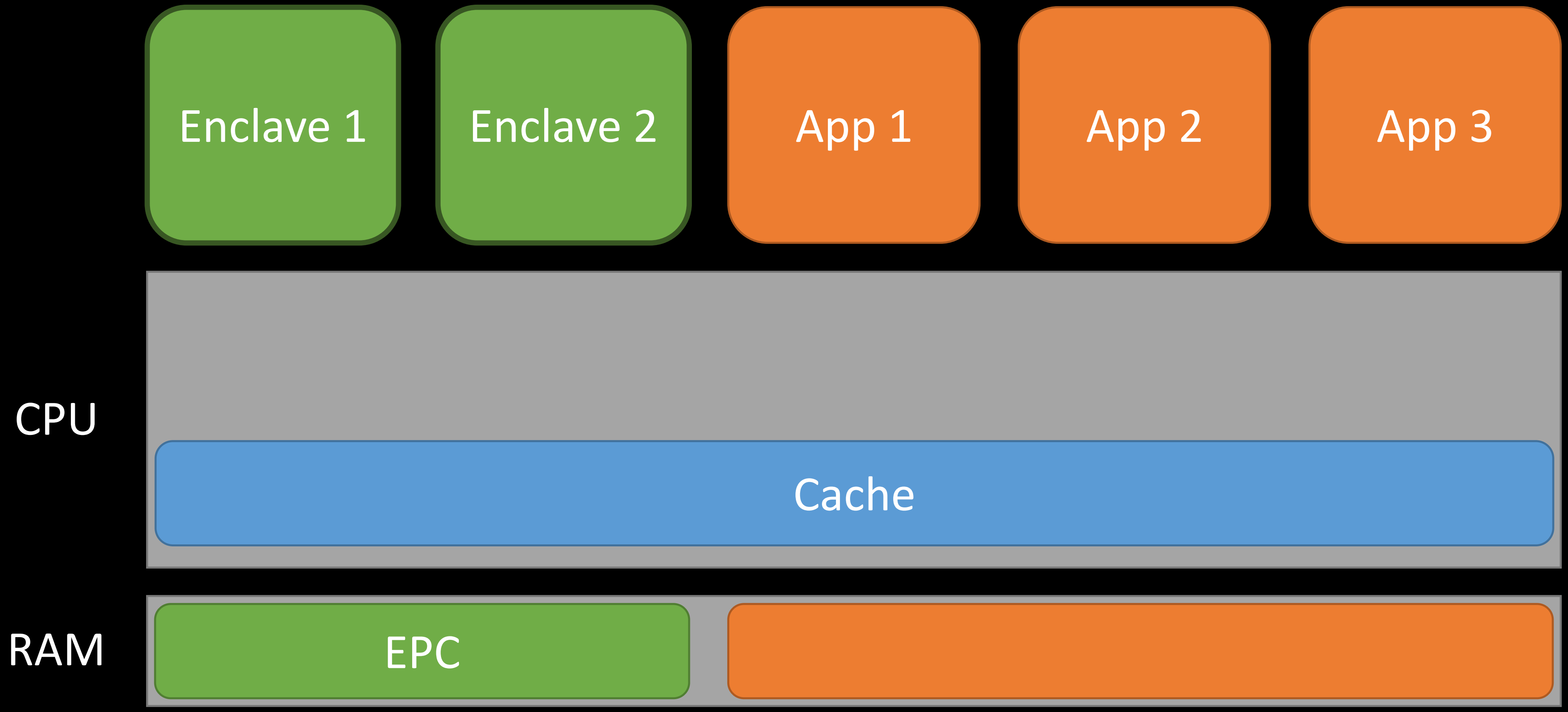
CPU

RAM

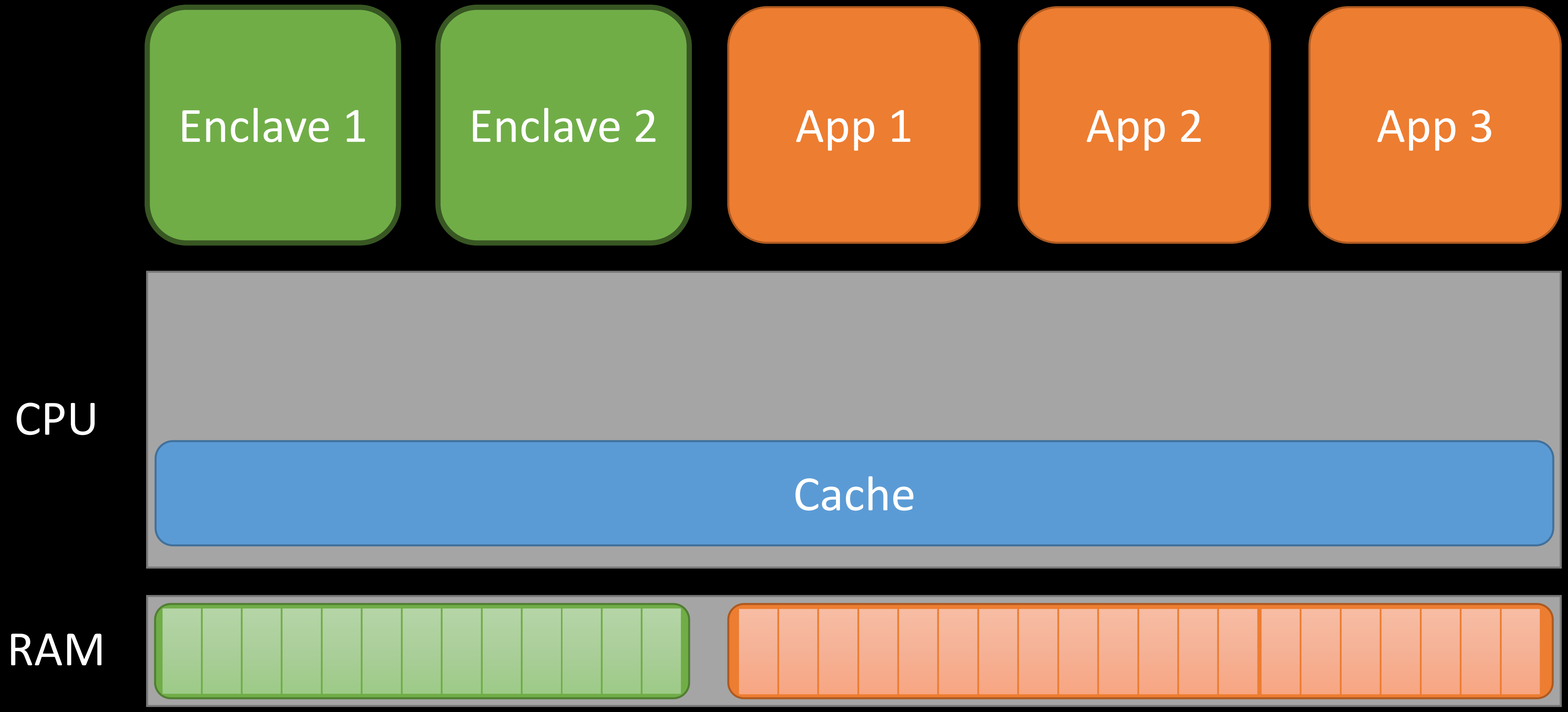
[Xu et al., IEEE S&P'15]

EPC: Enclave Page Cache
PT: Page Tables
PF: Page-Fault

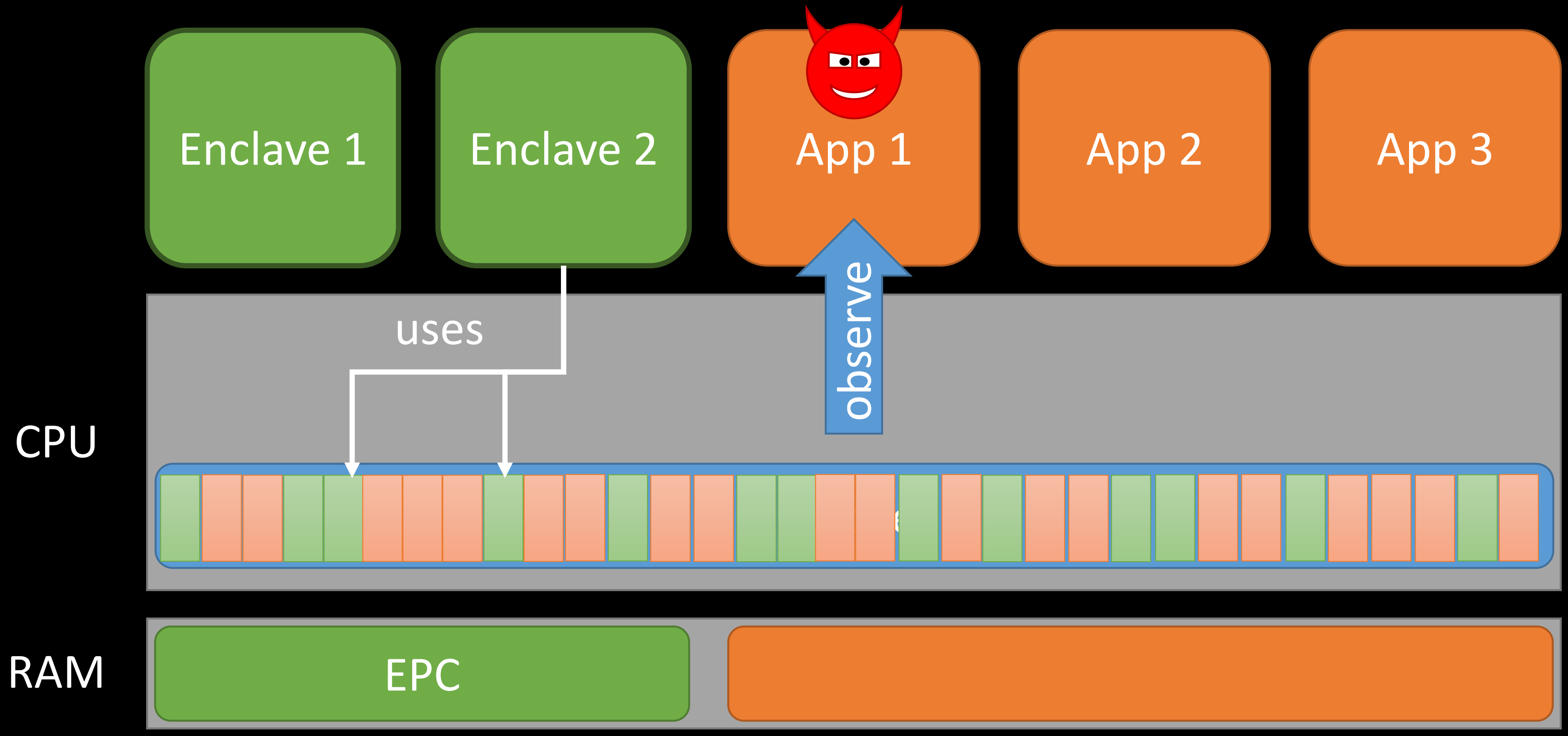
Cache Attacks on SGX: Hack in The Box



Cache Attacks on SGX: Hack in The Box



Cache Attacks on SGX: Hack in The Box

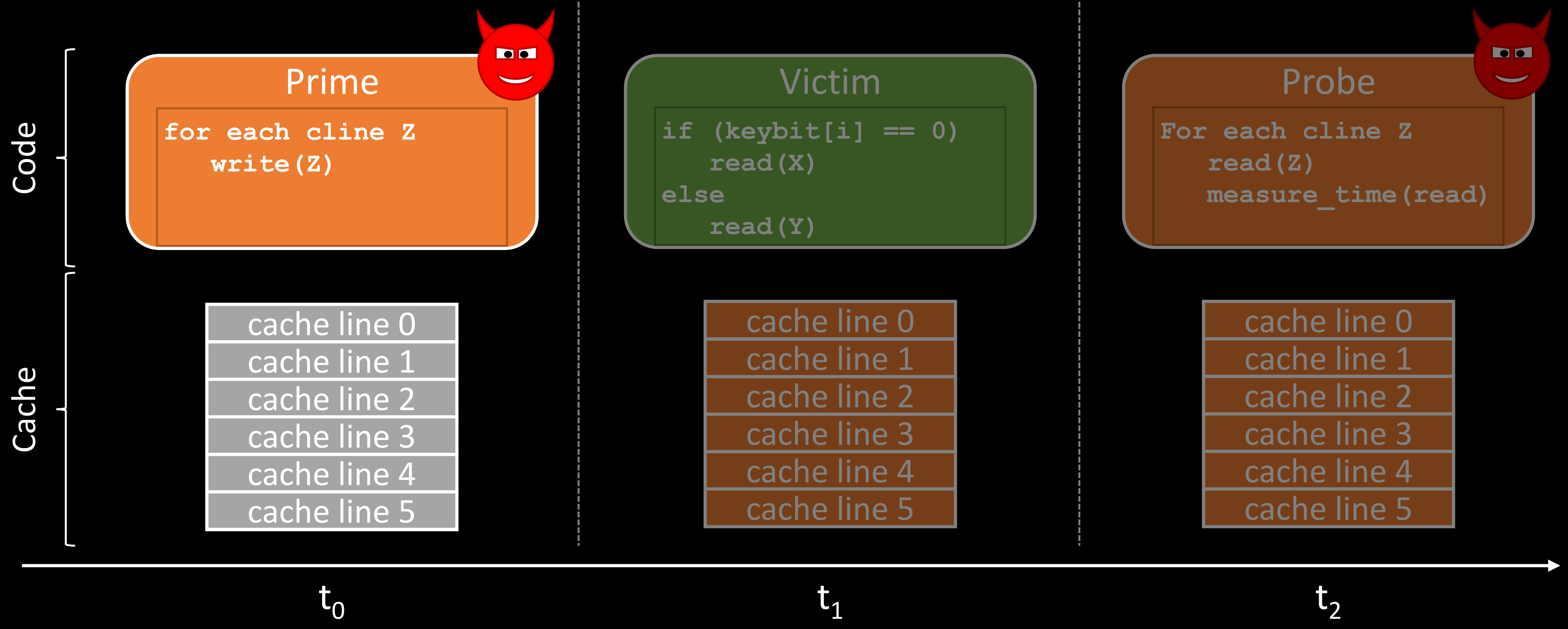


CYSEC
Cybersecurity
TU Darmstadt

Side-Channel Attacks Basics: Prime + Probe

Cache-based Side-Channel Attacks

Prime + Probe



```
for each cline Z
   write(Z)
```

```
if (keybit[i] == 0)
   read(X)
else
   read(Y)
```

```
For each cline Z
   read(Z)
   measure_time(read)
```

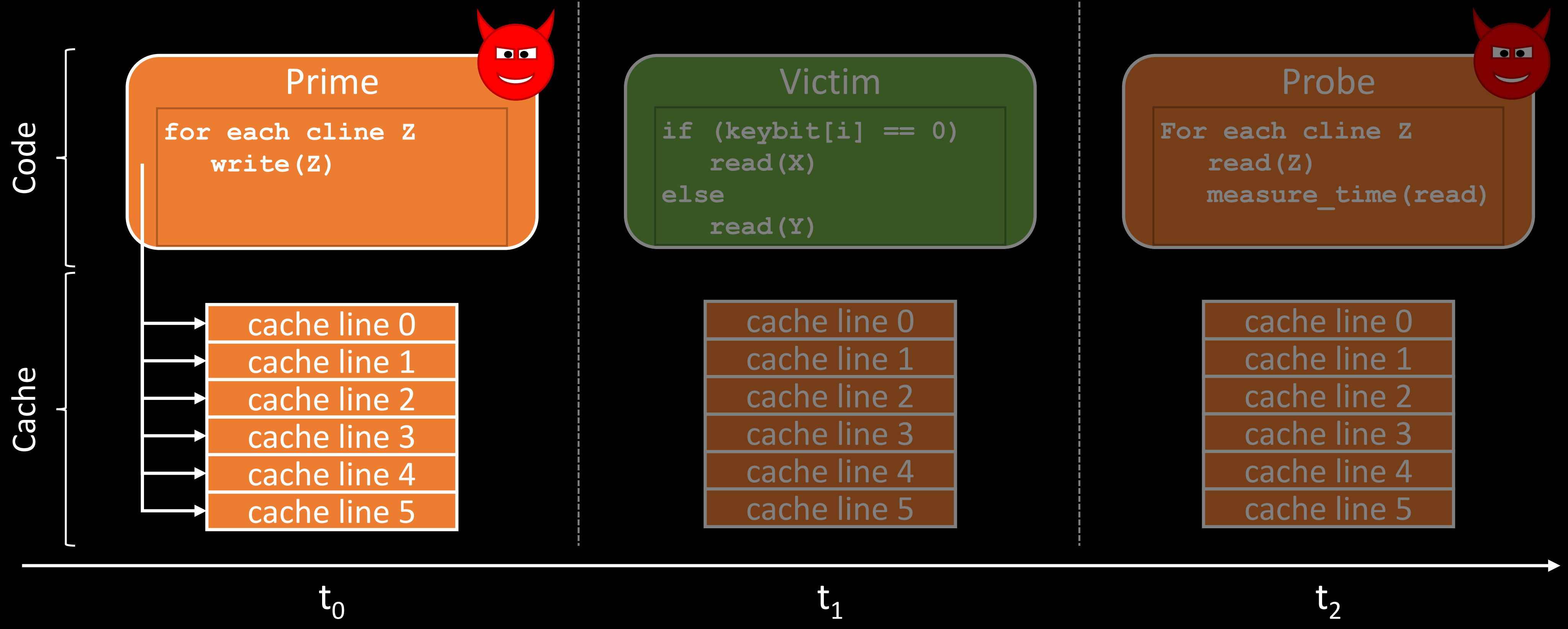
cache line 0
cache line 1
cache line 2
cache line 3
cache line 4
cache line 5

cache line 0
cache line 1
cache line 2
cache line 3
cache line 4
cache line 5

cache line 0
cache line 1
cache line 2
cache line 3
cache line 4
cache line 5

Cache-based Side-Channel Attacks

Prime + Probe



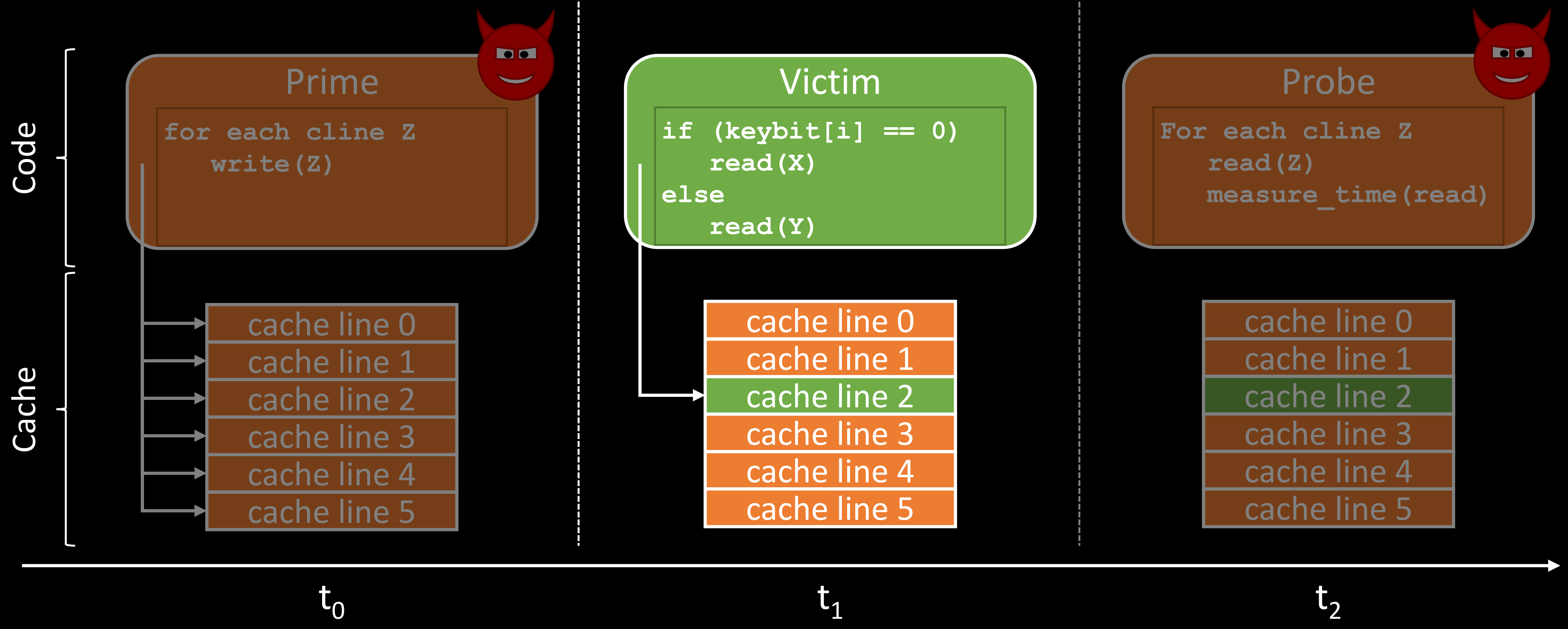
```
for each cline Z
   write(Z)
```

```
if (keybit[i] == 0)
   read(X)
else
   read(Y)
```

```
For each cline Z
   read(Z)
   measure_time(read)
```

Cache-based Side-Channel Attacks

Prime + Probe



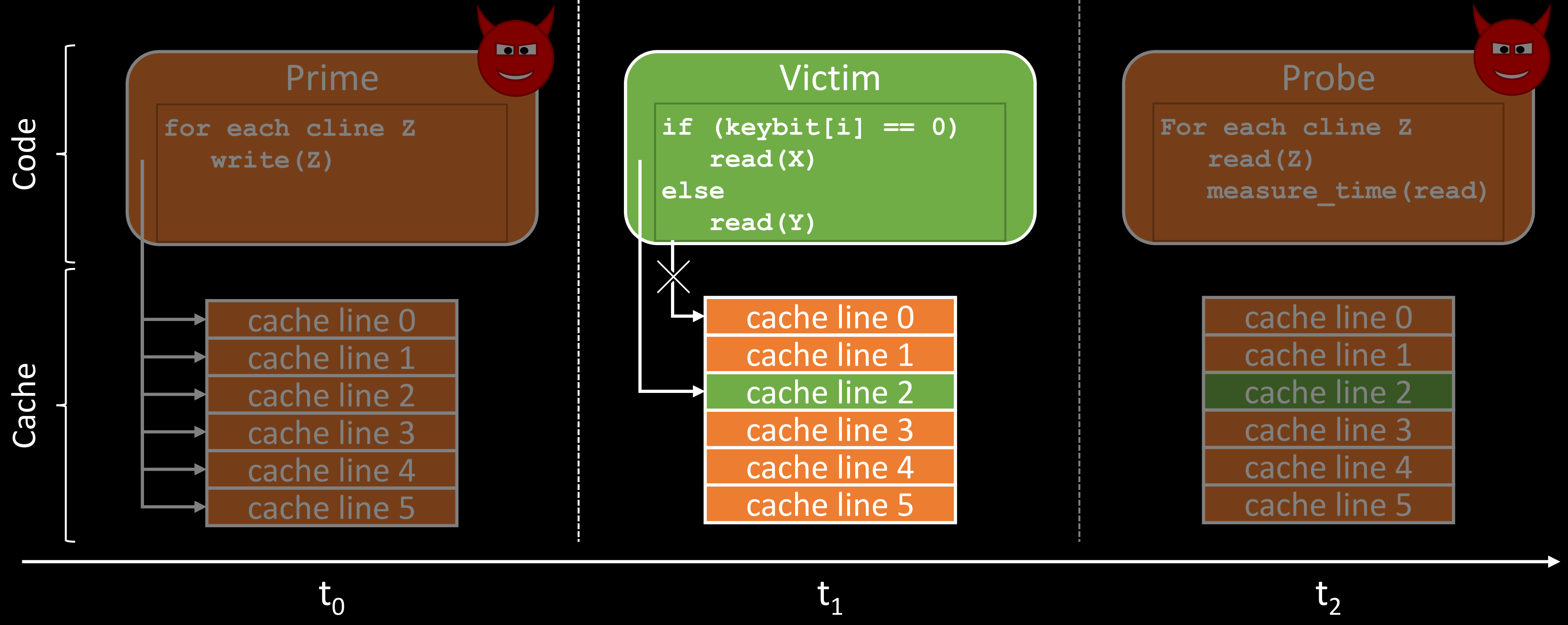
```
for each cline Z
   write(Z)
```

```
if (keybit[i] == 0)
   read(X)
else
   read(Y)
```

```
For each cline Z
   read(Z)
   measure_time(read)
```

Cache-based Side-Channel Attacks

Prime + Probe



```
for each cline Z
   write(Z)
```

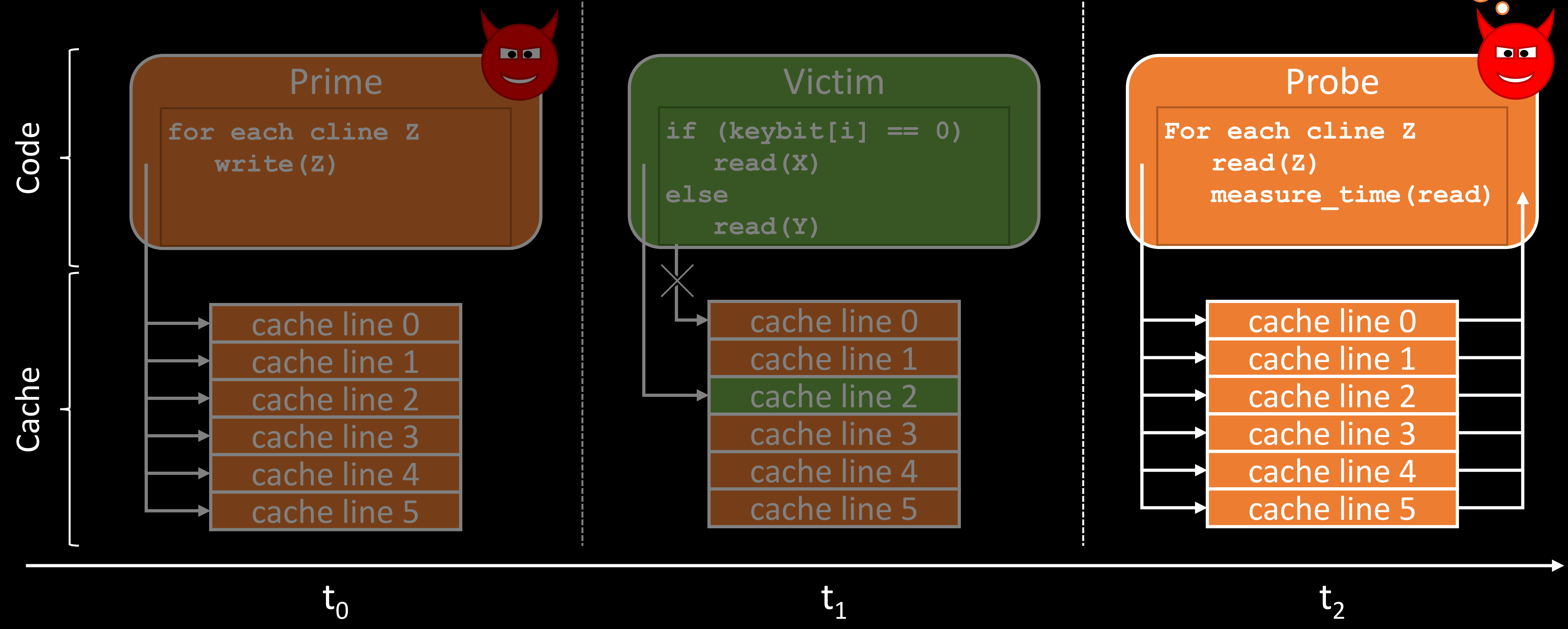
```
if (keybit[i] == 0)
   read(X)
else
   read(Y)
```

```
For each cline Z
   read(Z)
   measure_time(read)
```

Cache-based Side-Channel Attacks

Prime + Probe

cache line 2 was used by victim



Prime:

```
for each cline Z
   write(Z)
```

Victim:

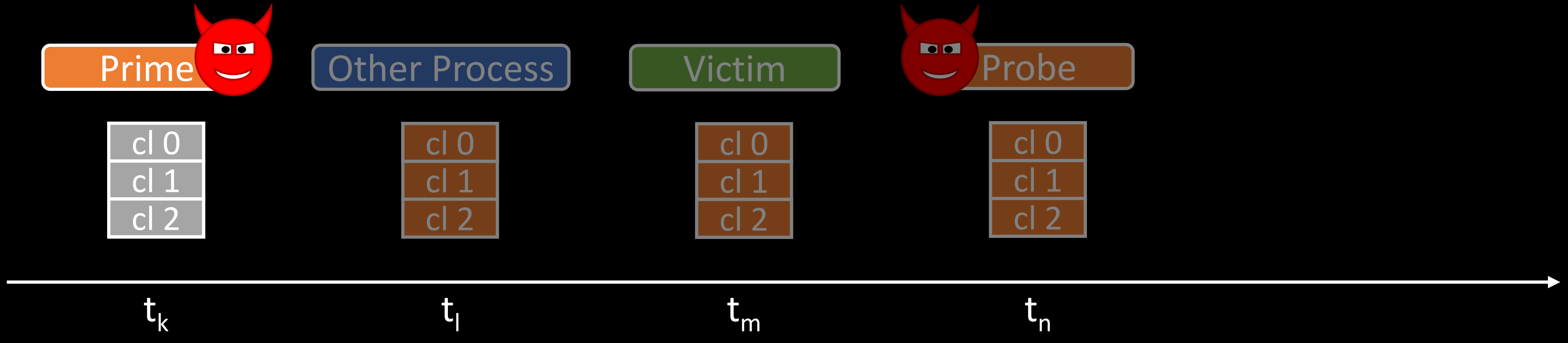
```
if (keybit[i] == 0)
   read(X)
else
   read(Y)
```

Probe:

```
For each cline Z
   read(Z)
   measure_time(read)
```

Side-Channel Attacker Challenge: Noise

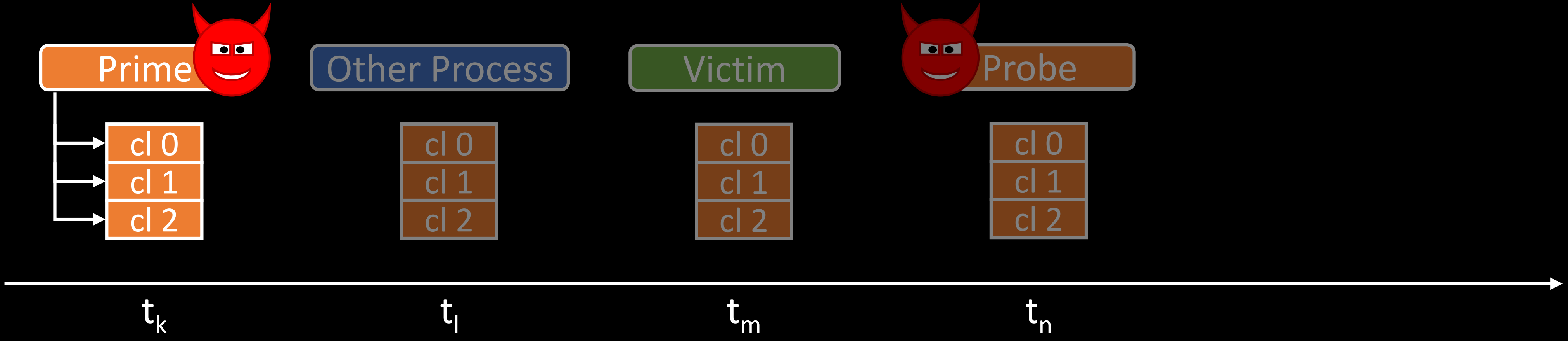
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

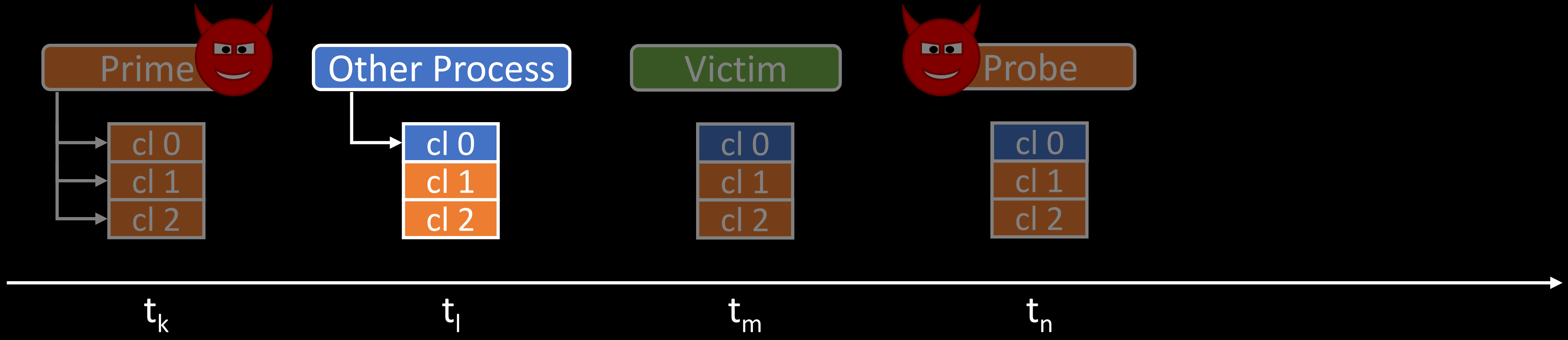
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

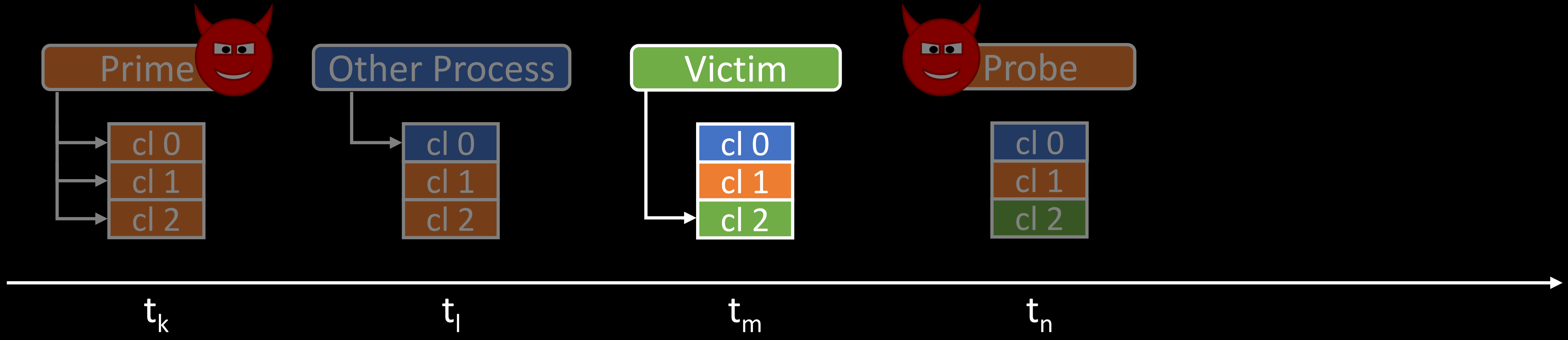
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

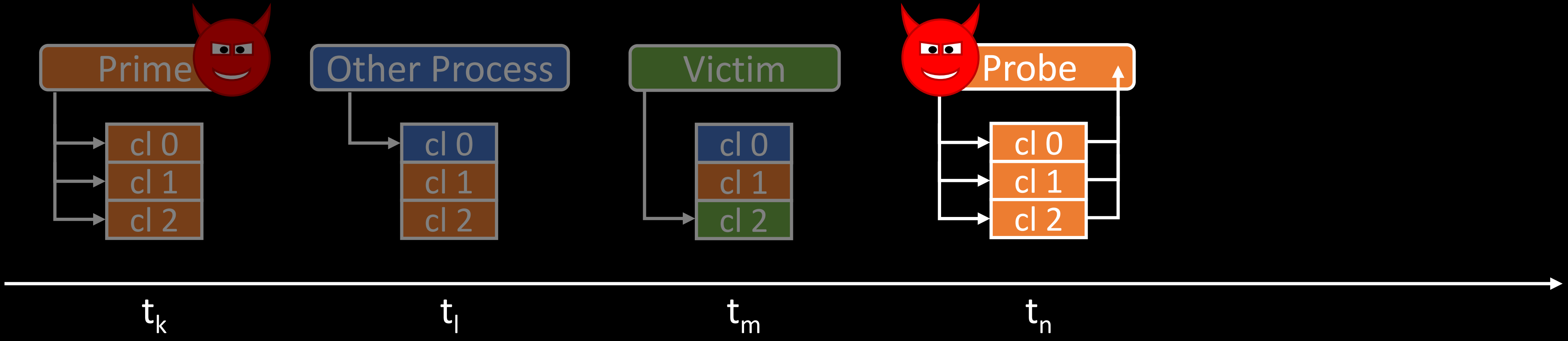
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

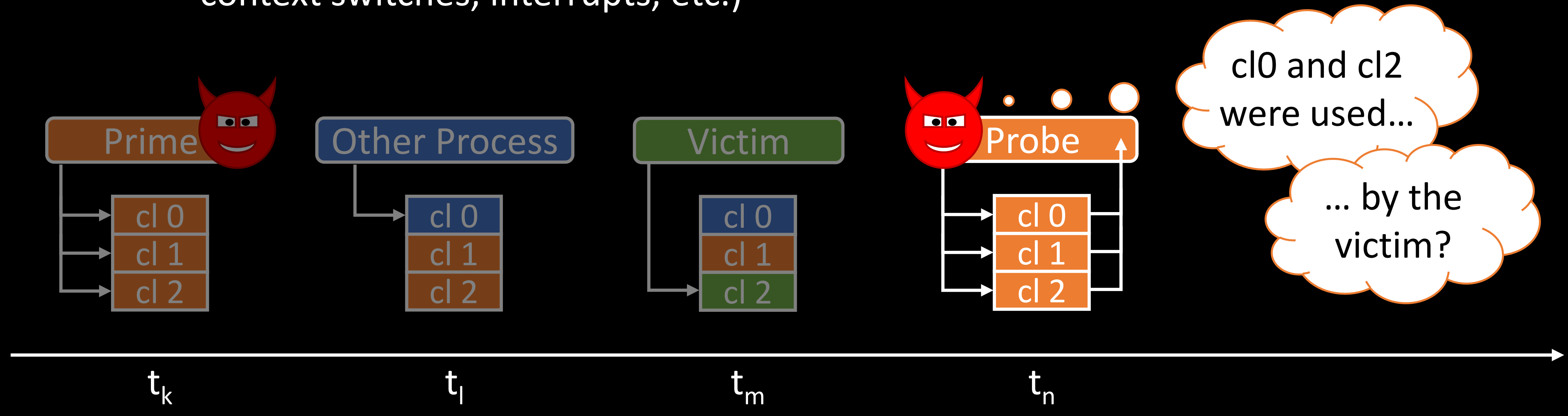
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

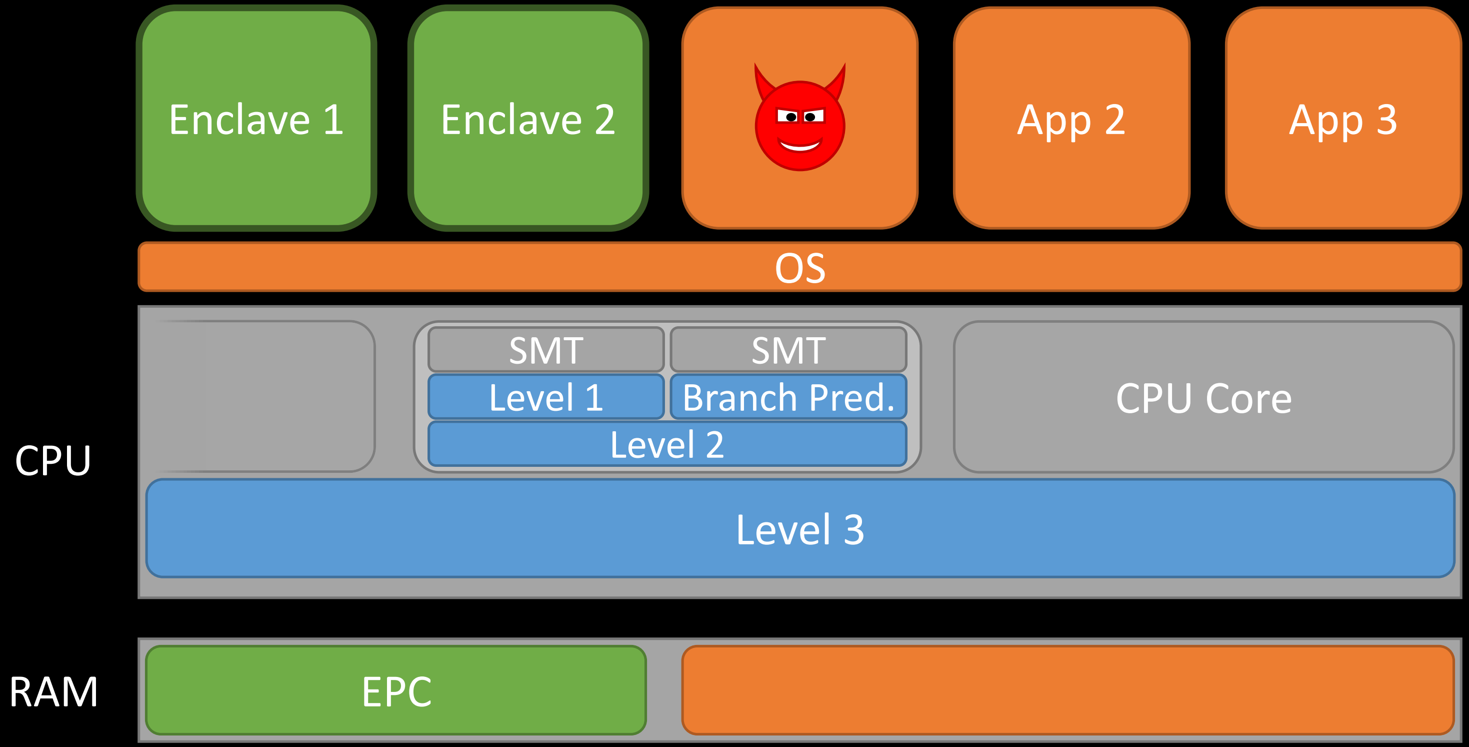
Side-Channel Attacker Challenge: Noise

- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



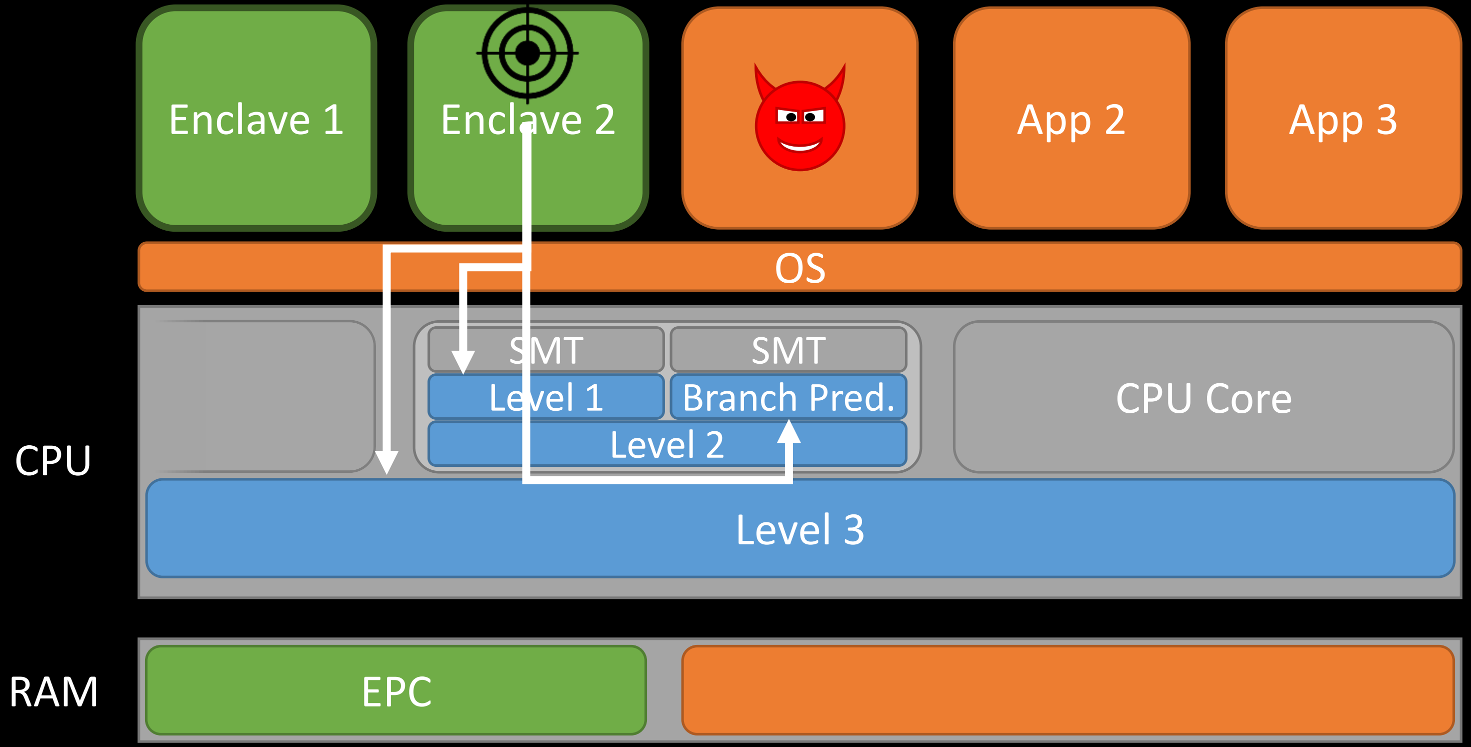
*OS: Operating System and any other privileged system software

Cache Attacks on SGX



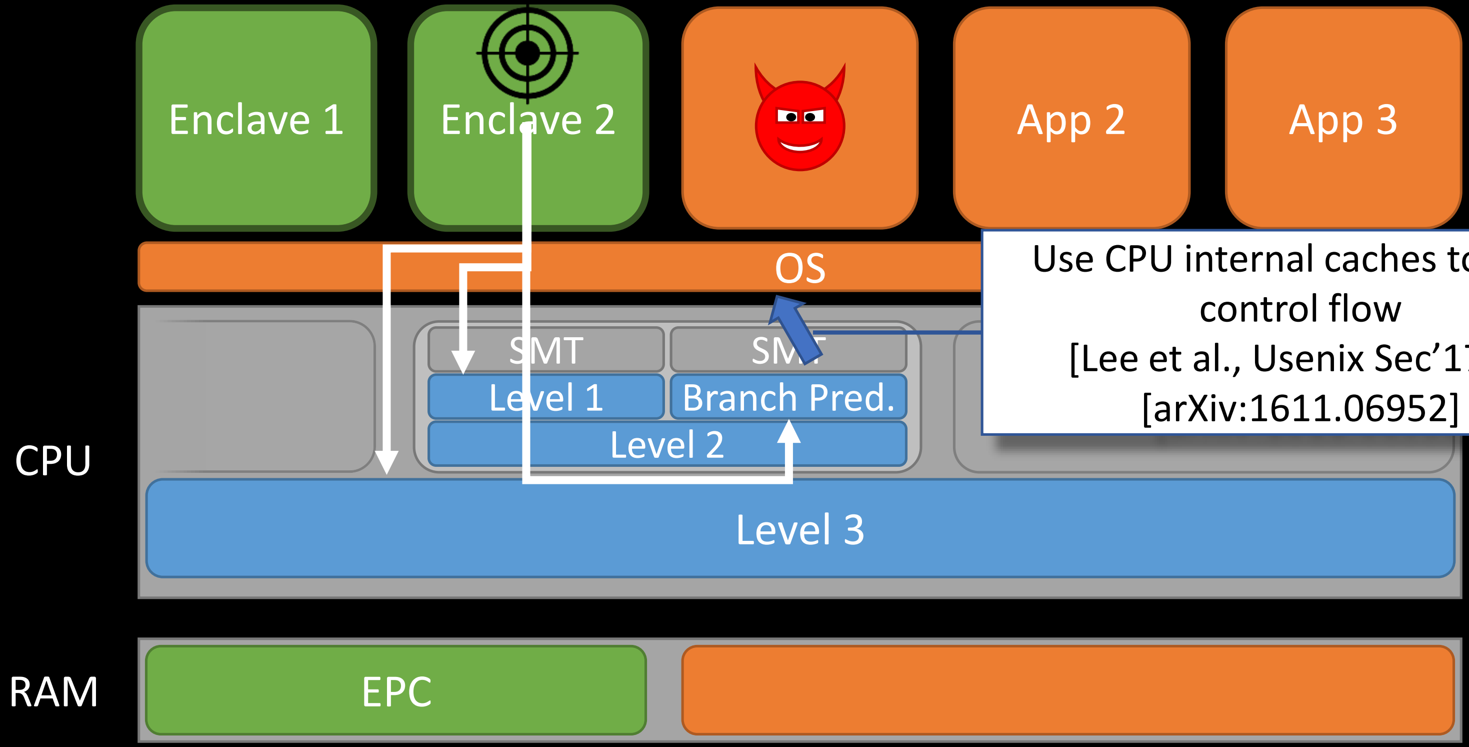
EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

Cache Attacks on SGX



EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

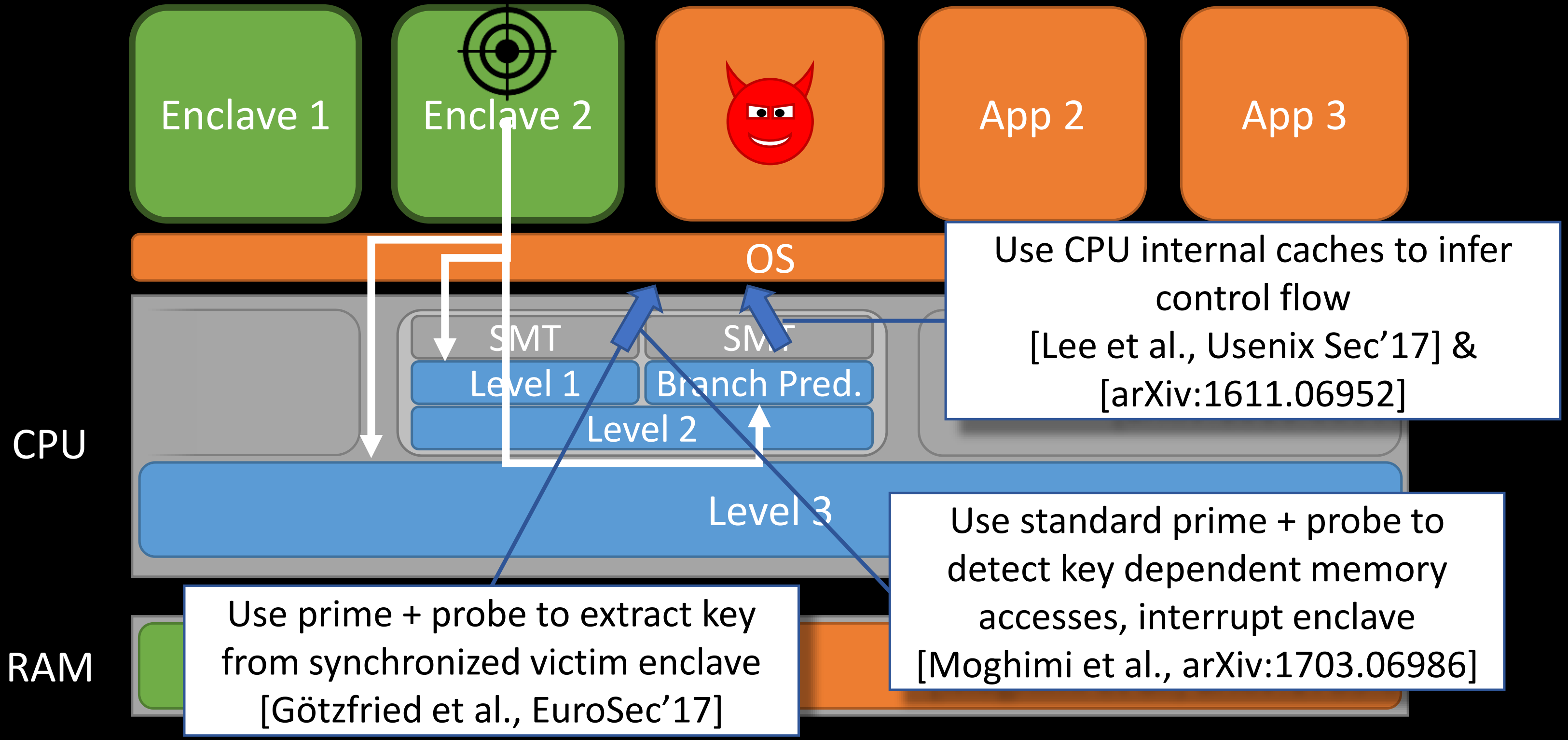
Cache Attacks on SGX



Use CPU internal caches to infer control flow
[Lee et al., Usenix Sec’17] &
[arXiv:1611.06952]

EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

Cache Attacks on SGX



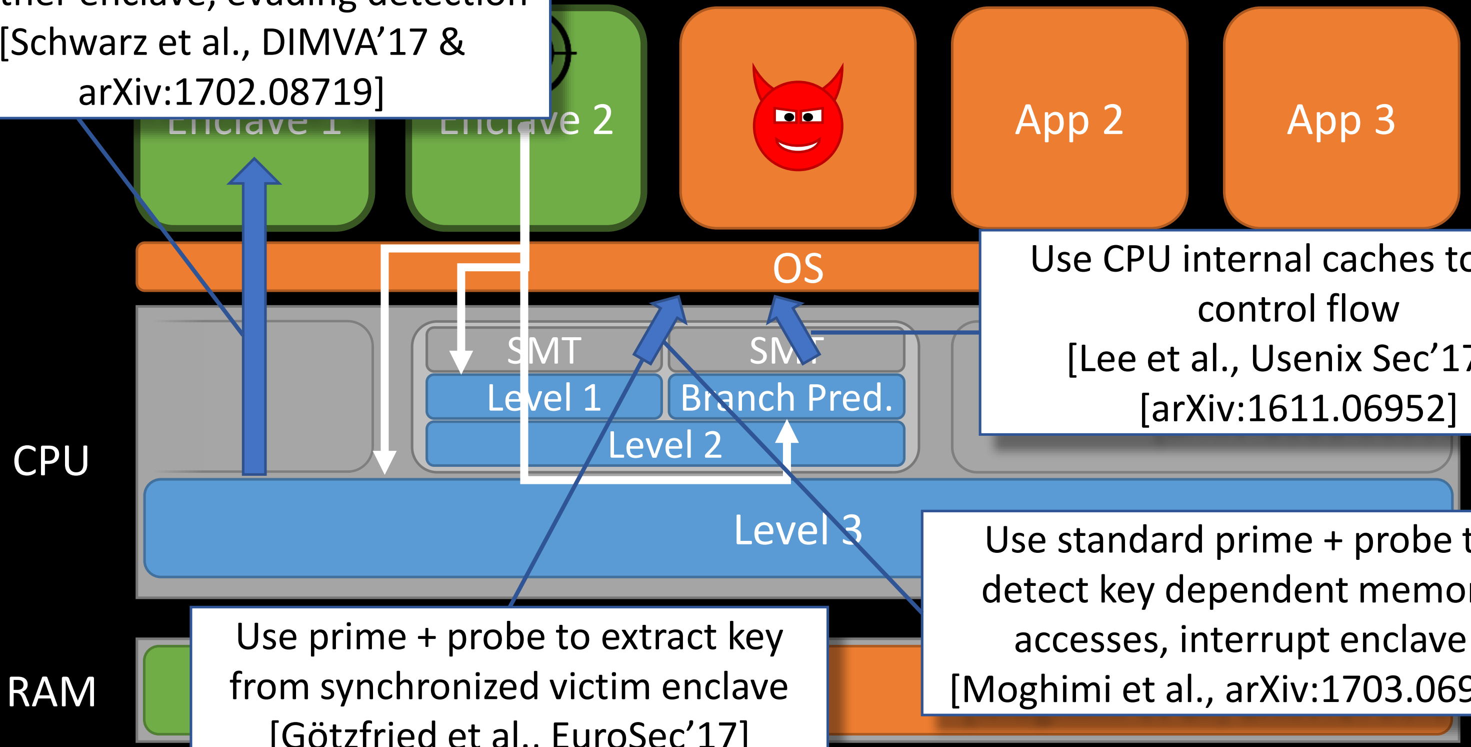
Use CPU internal caches to infer control flow
[Lee et al., Usenix Sec’17] & [arXiv:1611.06952]

Use standard prime + probe to detect key dependent memory accesses, interrupt enclave
[Moghimi et al., arXiv:1703.06986]

Use prime + probe to extract key from synchronized victim enclave
[Götzfried et al., EuroSec’17]

EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

Cache Attacks on SGX



A malicious enclave prime + probes another enclave, evading detection [Schwarz et al., DIMVA'17 & arXiv:1702.08719]

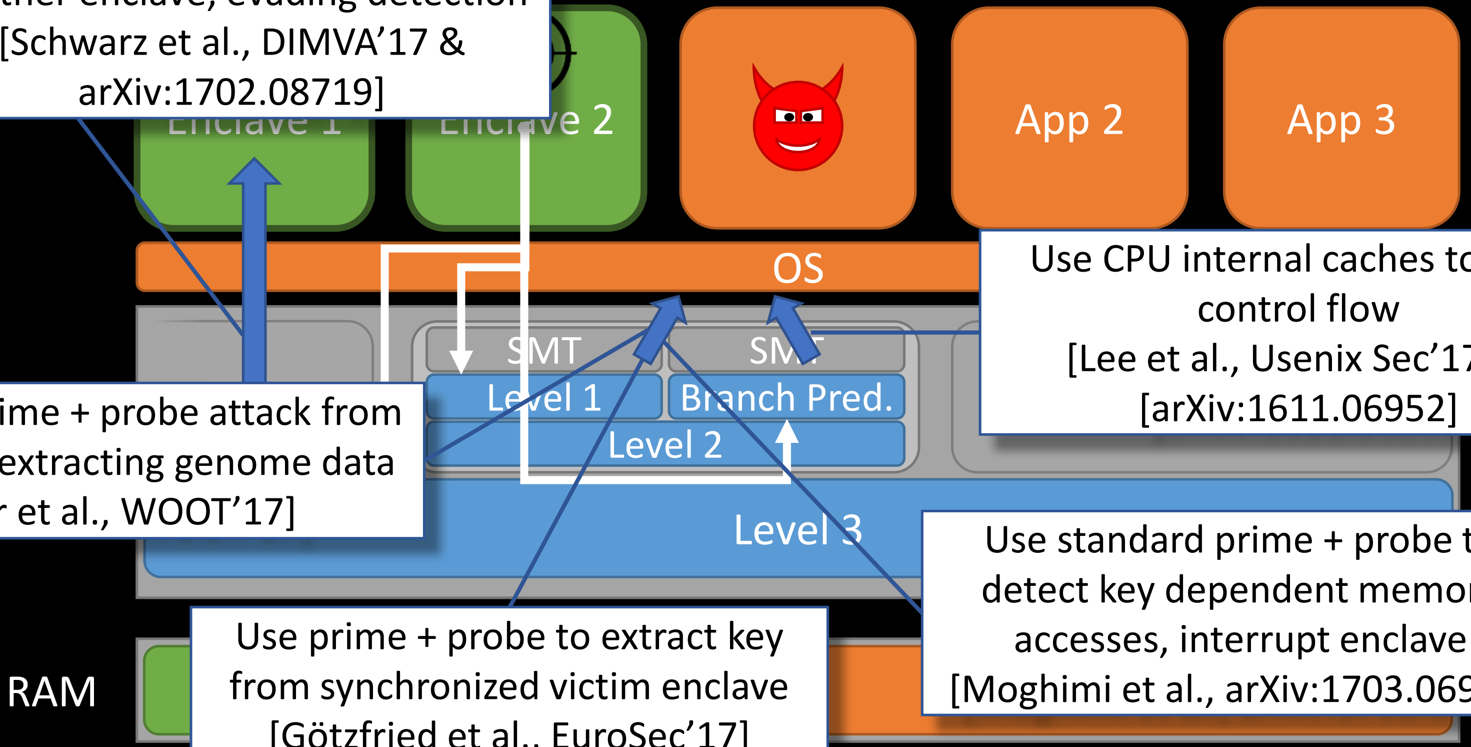
Use CPU internal caches to infer control flow [Lee et al., Usenix Sec'17] & [arXiv:1611.06952]

Use standard prime + probe to detect key dependent memory accesses, interrupt enclave [Moghimi et al., arXiv:1703.06986]

Use prime + probe to extract key from synchronized victim enclave [Götzfried et al., EuroSec'17]

EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

Cache Attacks on SGX



A malicious enclave prime + probes another enclave, evading detection [Schwarz et al., DIMVA’17 & arXiv:1702.08719]

Use CPU internal caches to infer control flow [Lee et al., Usenix Sec’17] & [arXiv:1611.06952]

Our attack: prime + probe attack from malicious OS extracting genome data [Brasser et al., WOOT’17]

Use standard prime + probe to detect key dependent memory accesses, interrupt enclave [Moghimi et al., arXiv:1703.06986]

Use prime + probe to extract key from synchronized victim enclave [Götzfried et al., EuroSec’17]

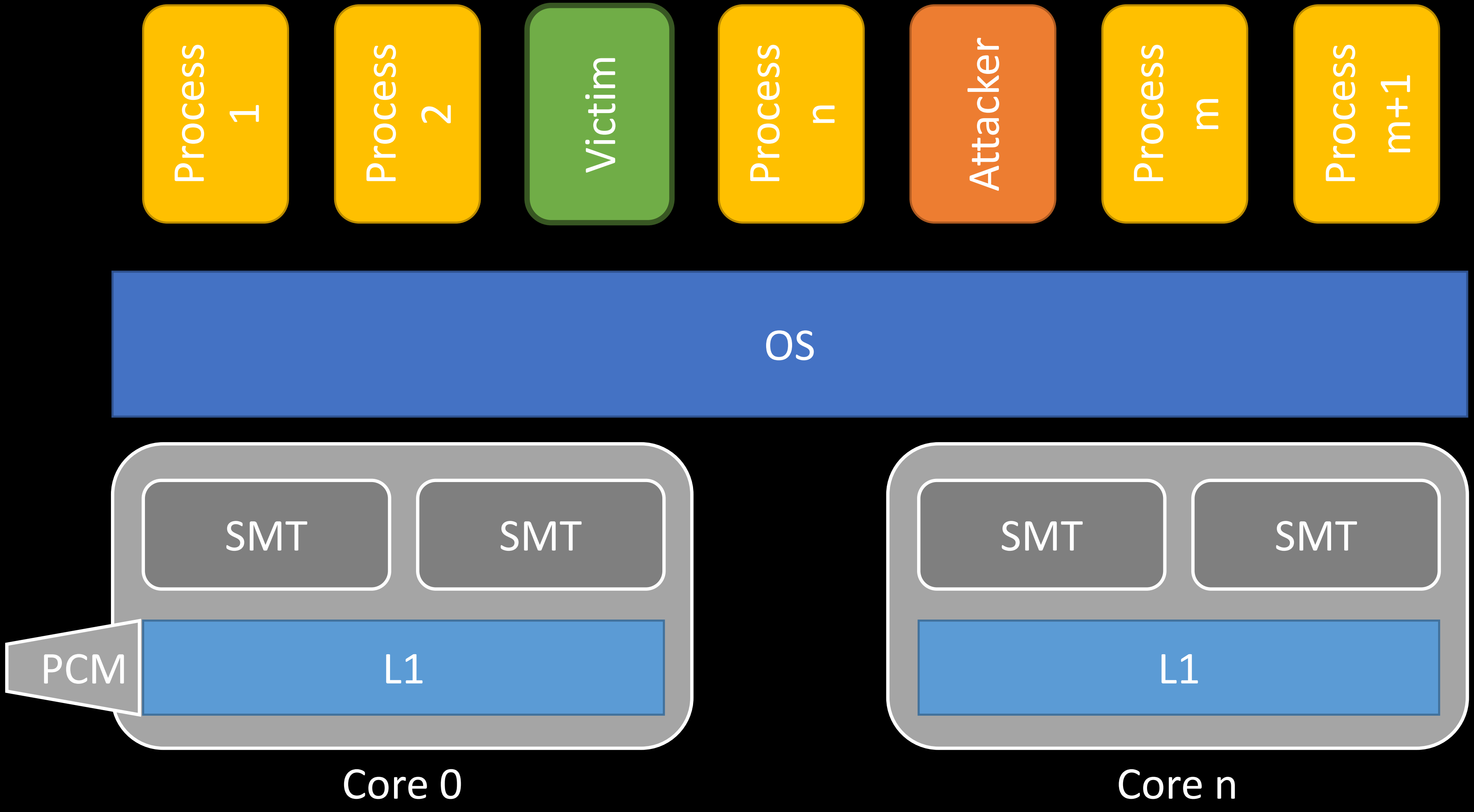
EPC: Enclave Page Cache
SMT: Simultaneous Multithreading

SGX Side-Channel Attacks Comparison

	Attack Type	Observed Cache	Interrupting Victim	Cache Eviction Measurement	Attacker Code	Attacked Victim
Lee et al.	Branch Shadowing	BTB / LBR	Yes	Execution Timing	OS	RSA & SVM classifier
Moghimi et al.	Prime + Probe	L1(D)	Yes	Access timing	OS	AES
Götzfried et al.	Prime + Probe	L1(D)	No	PCM	OS	AES
Our Attack	Prime + Probe	L1(D)	No	PCM	OS	RSA & Genome Sequencing
Schwarz et al.	Prime + Probe	L3	No	Counting Thread	Enclave	AES

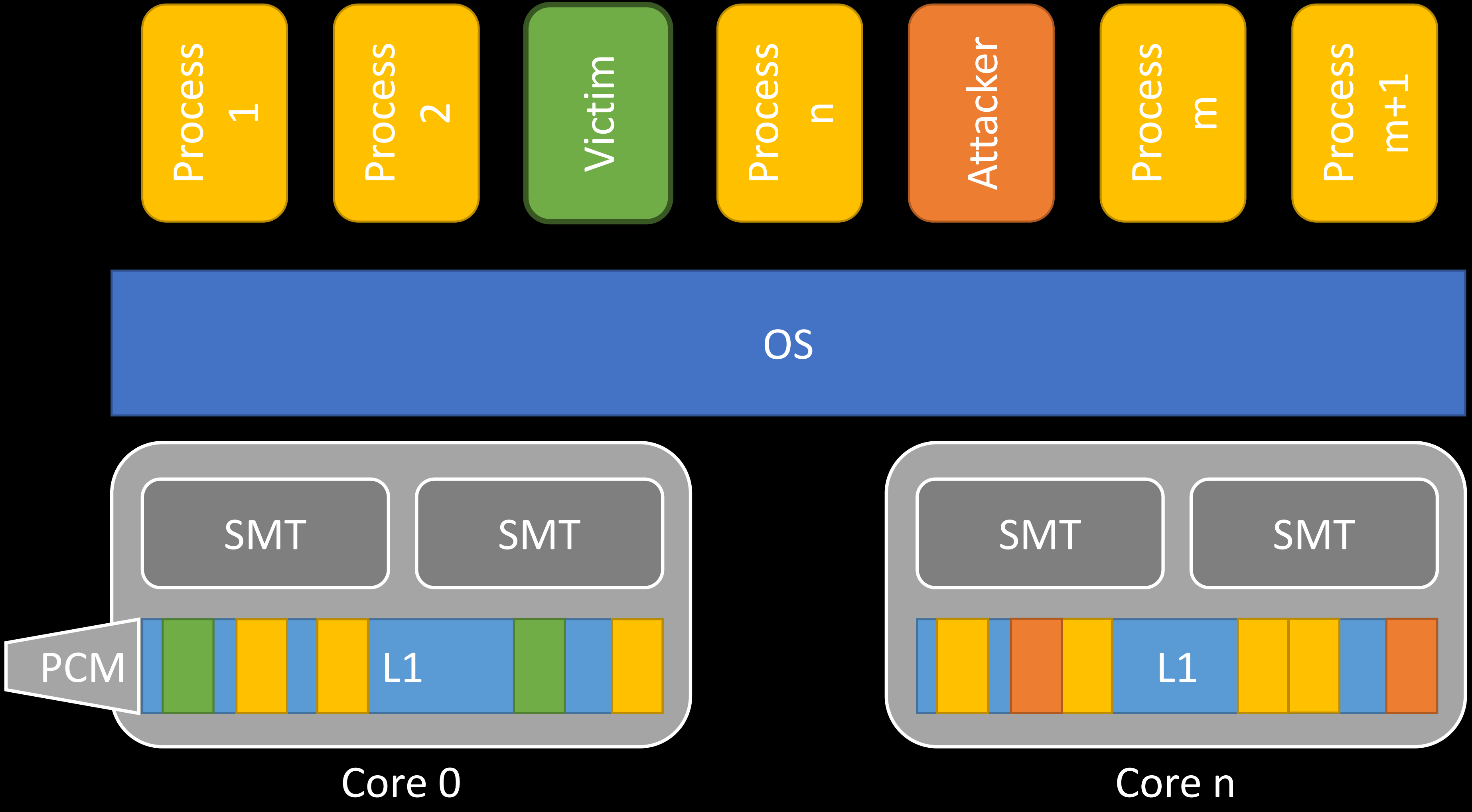
PCM: Performance Counter Monitor   BTB: Branch Target Buffer   LBR: Last Branch Record

Our Attack [Brasser et al., WOOT’17]



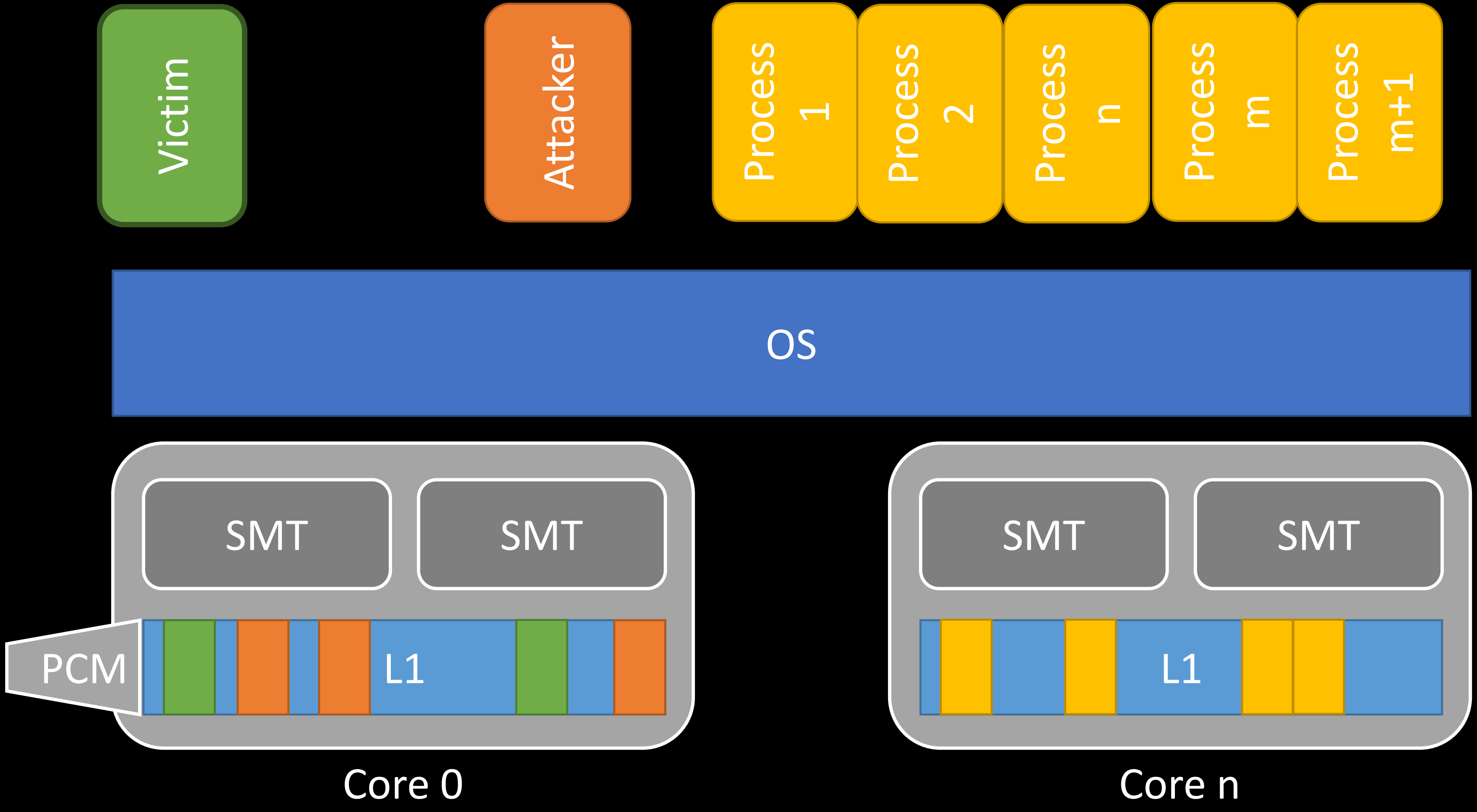
PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT’17]



PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT'17]

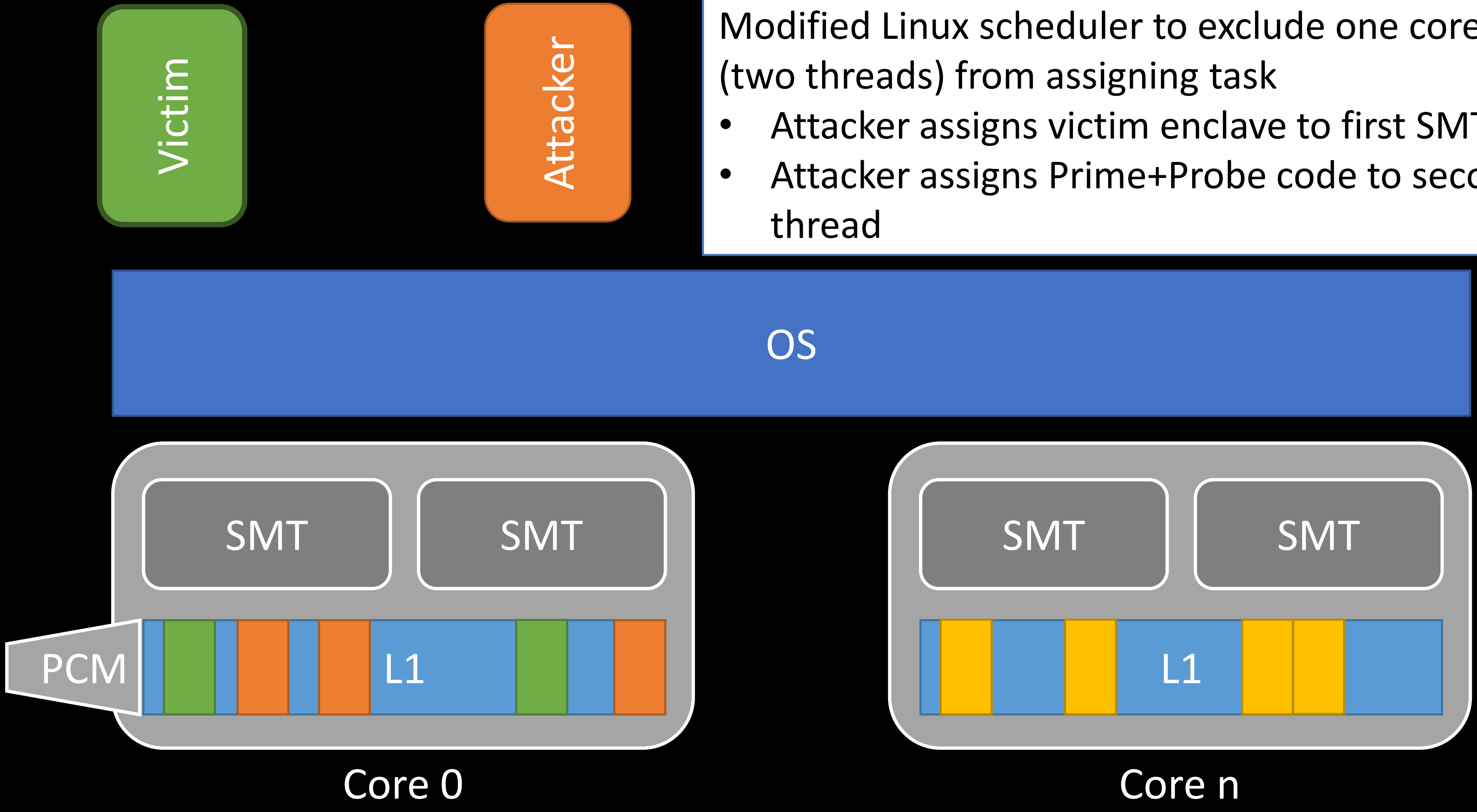


PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT'17]

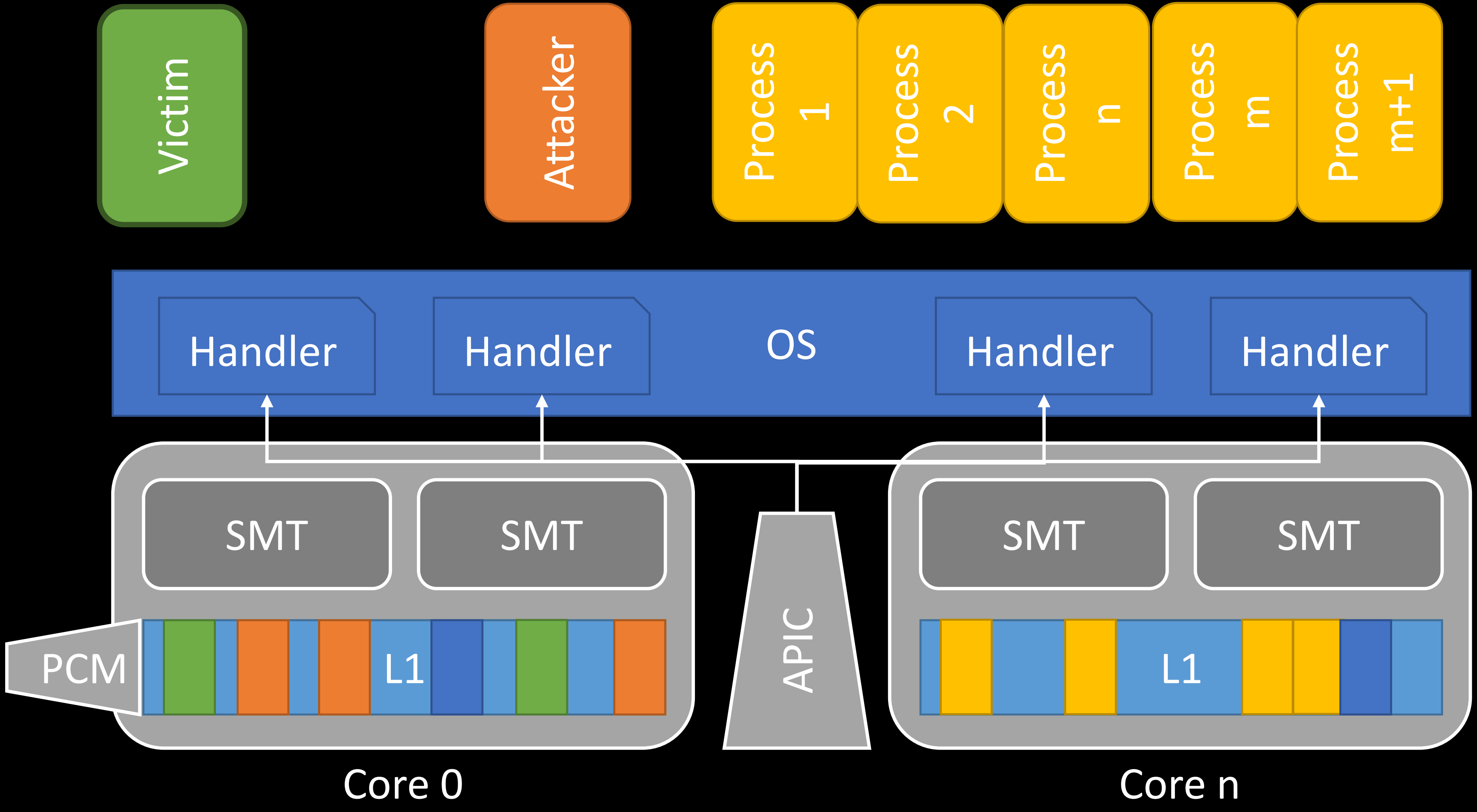
Modified Linux scheduler to exclude one core (two threads) from assigning task

- Attacker assigns victim enclave to first SMT thread
- Attacker assigns Prime+Probe code to second SMT thread



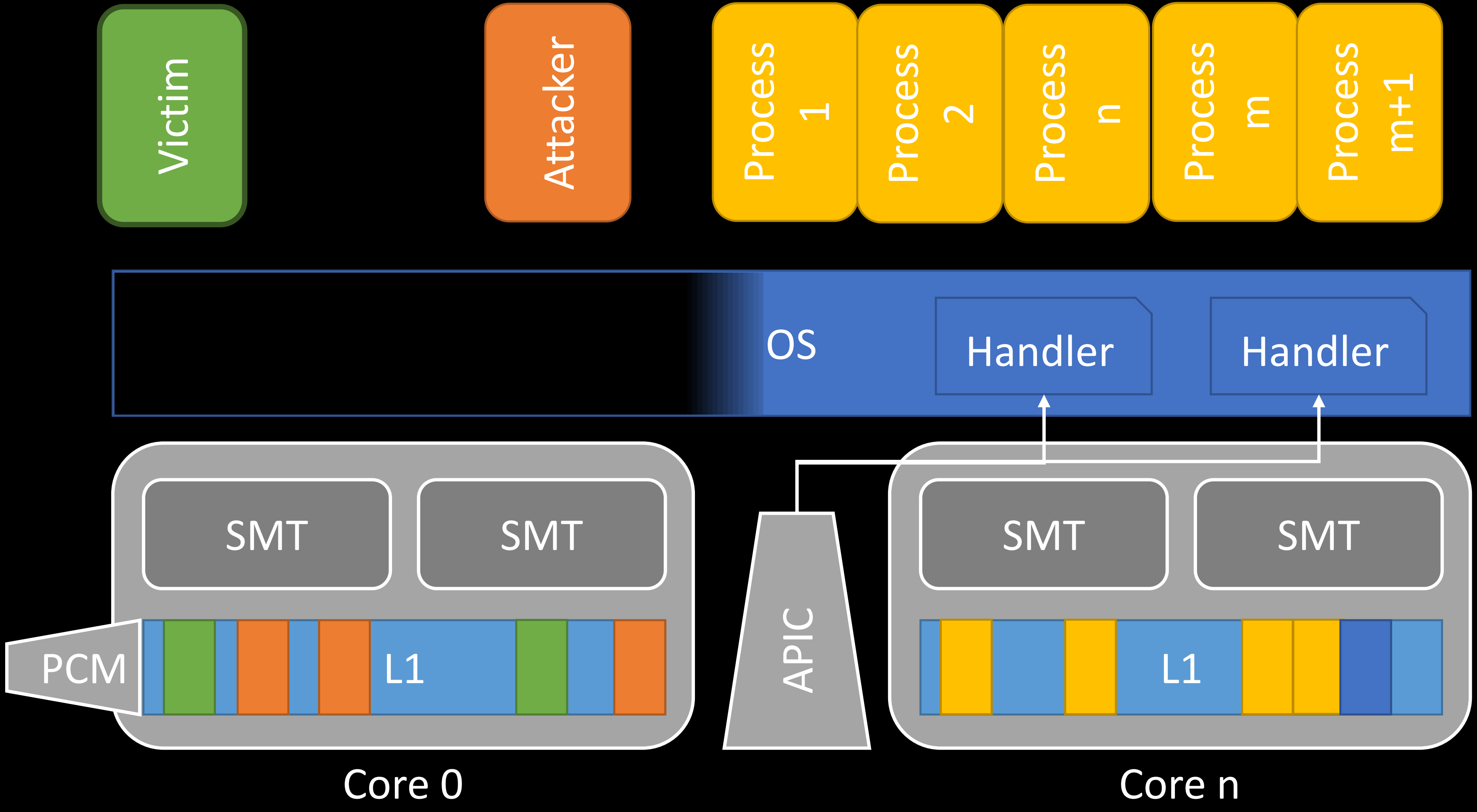
PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT'17]



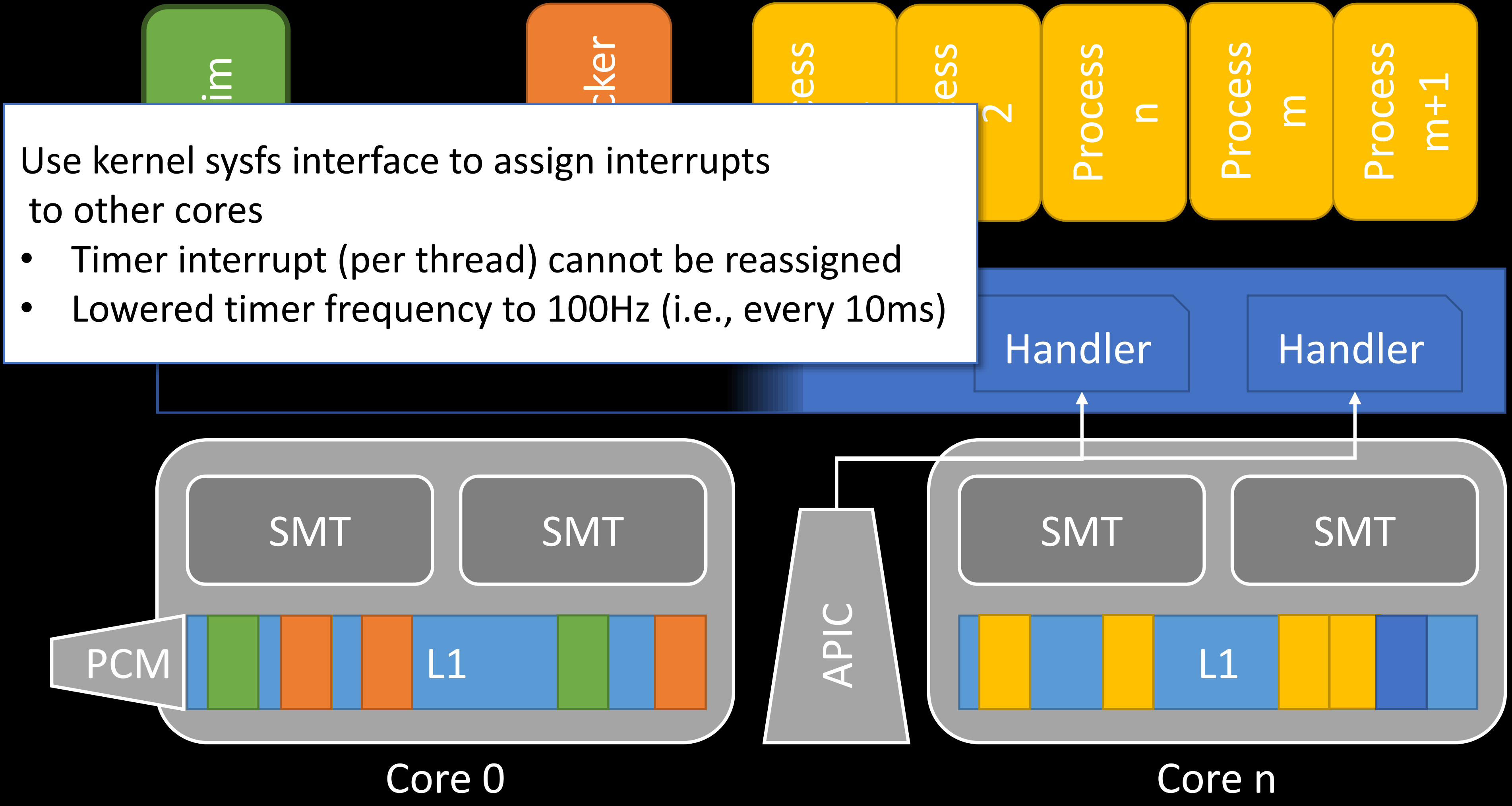
PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT'17]



PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT'17]

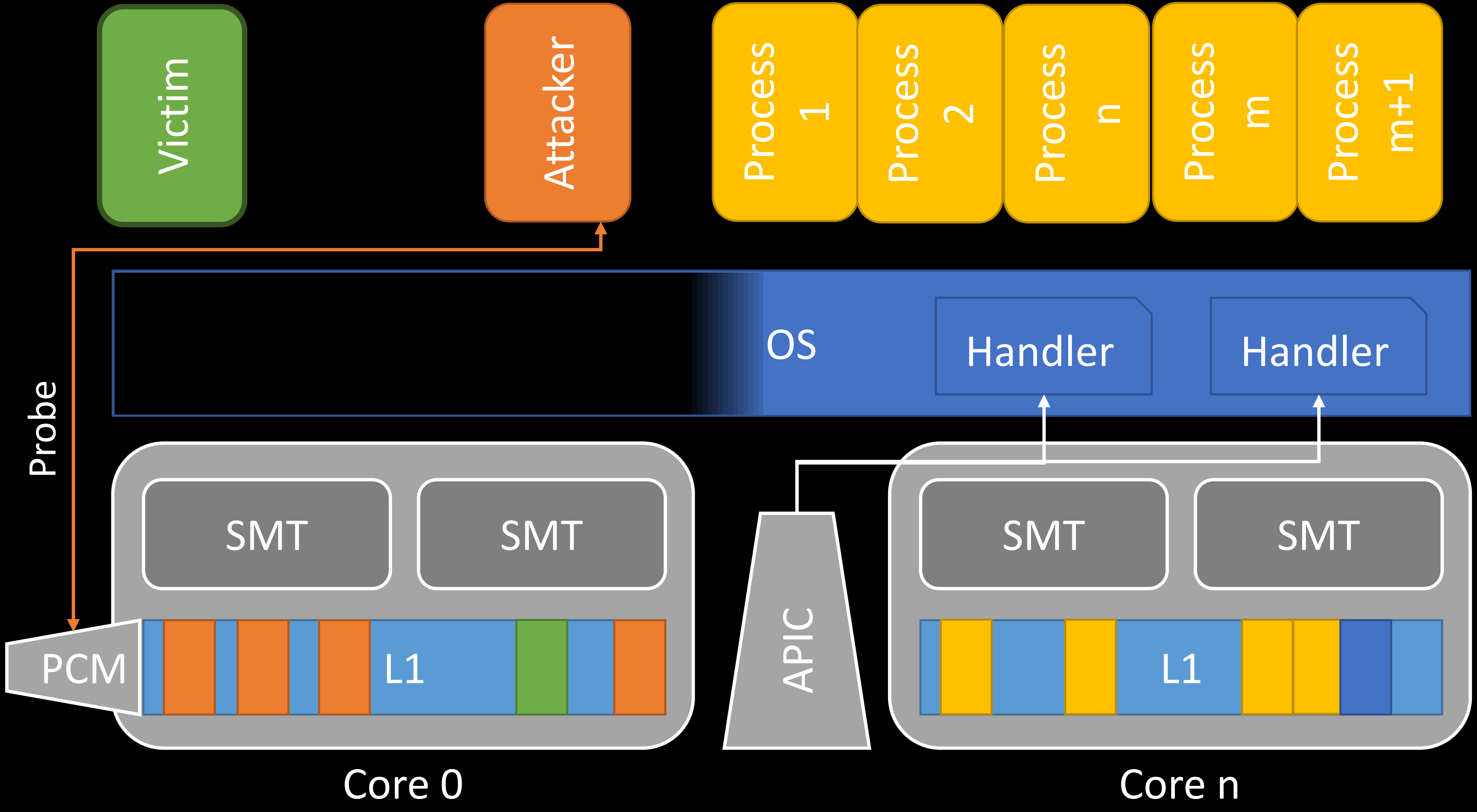


Use kernel sysfs interface to assign interrupts to other cores

- Timer interrupt (per thread) cannot be reassigned
- Lowered timer frequency to 100Hz (i.e., every 10ms)

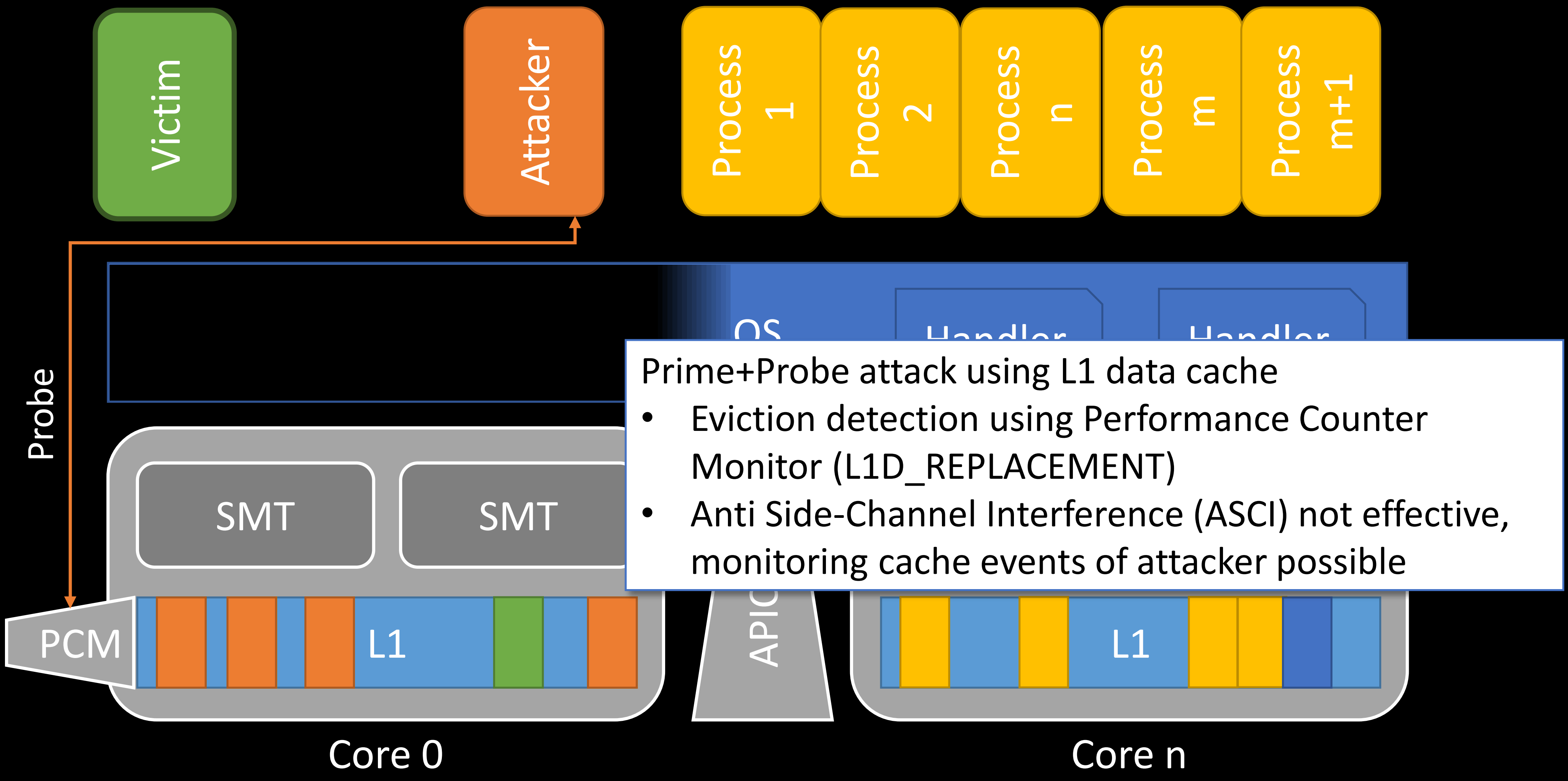
PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT’17]



PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT’17]

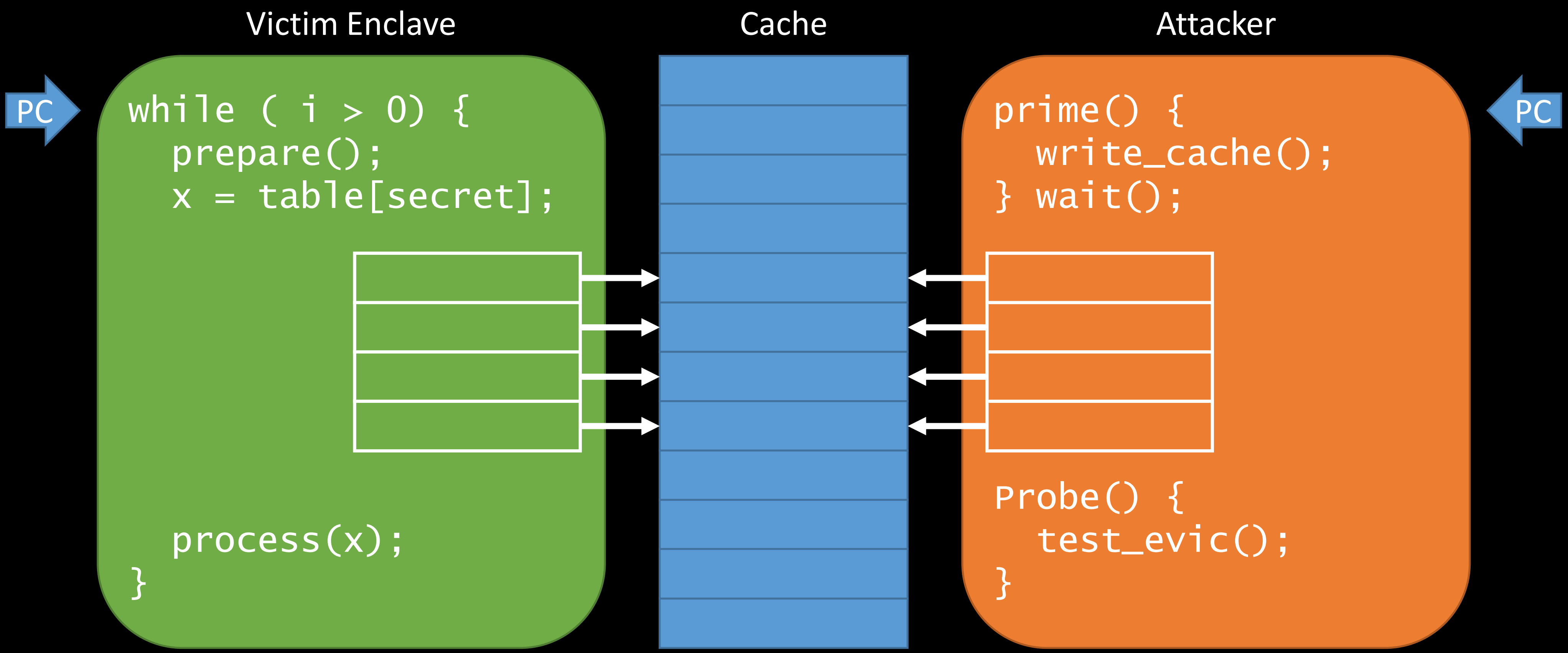


Prime+Probe attack using L1 data cache

- Eviction detection using Performance Counter Monitor (L1D_REPLACEMENT)
- Anti Side-Channel Interference (ASCI) not effective, monitoring cache events of attacker possible

PCM: Performance Counter Monitor
SMT: Simultaneous Multithreading
APIC: Advanced Programmable Interrupt Controller

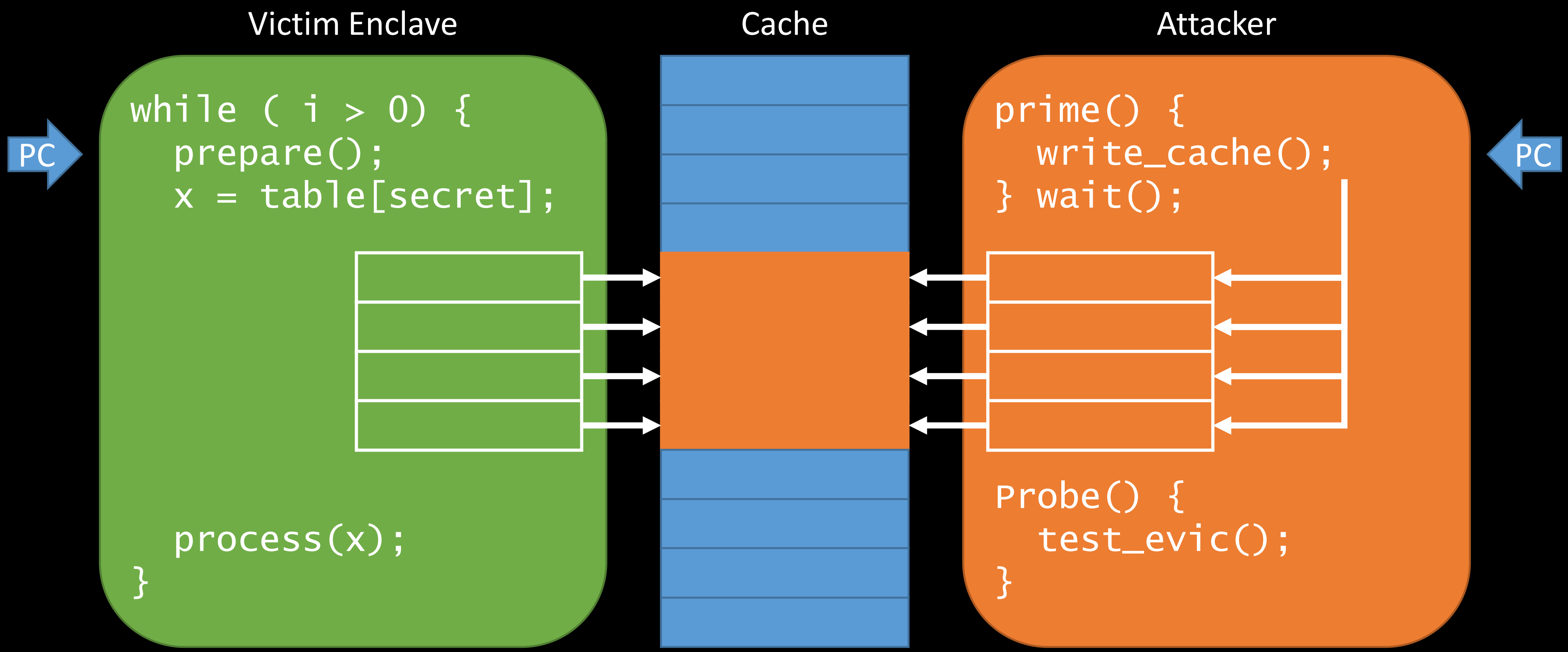
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

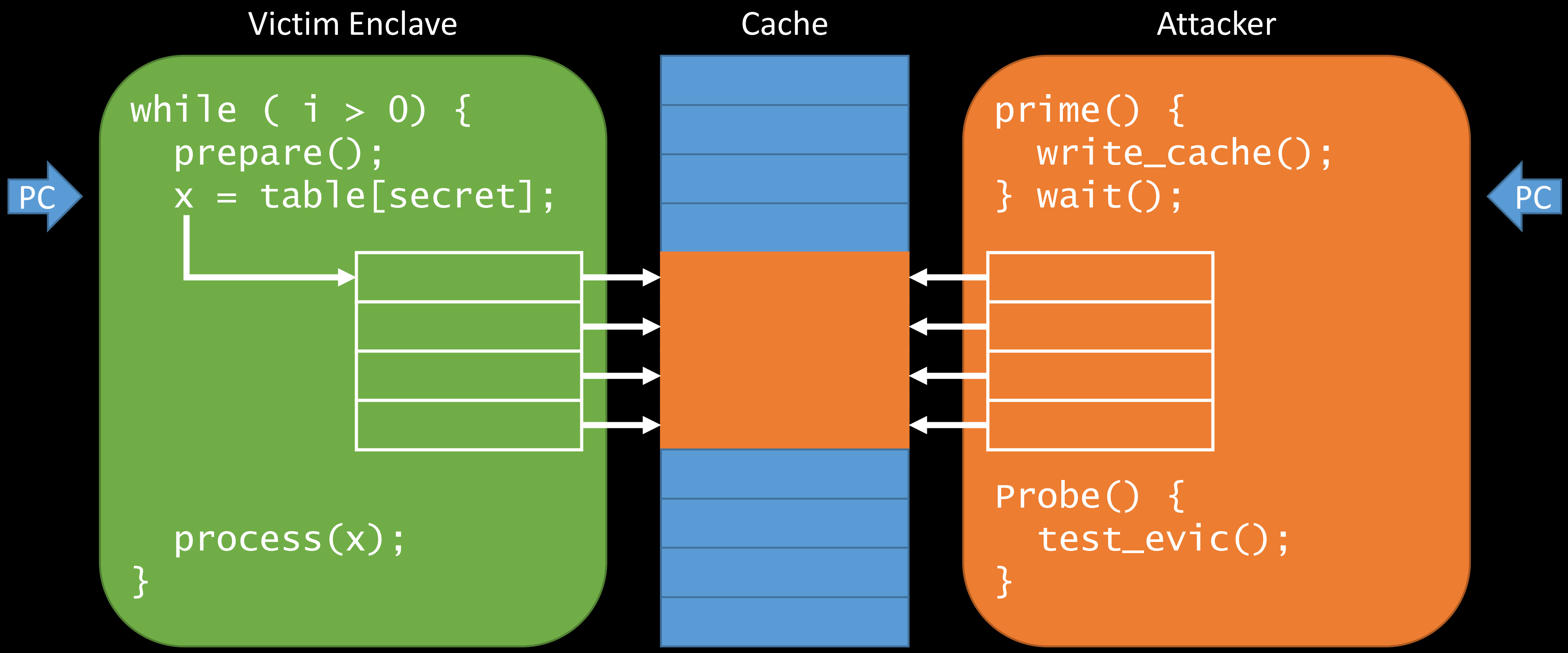
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];



  process(x);
}
```

```
prime() {
  write_cache();
} wait();



Probe() {
  test_evic();
}
```

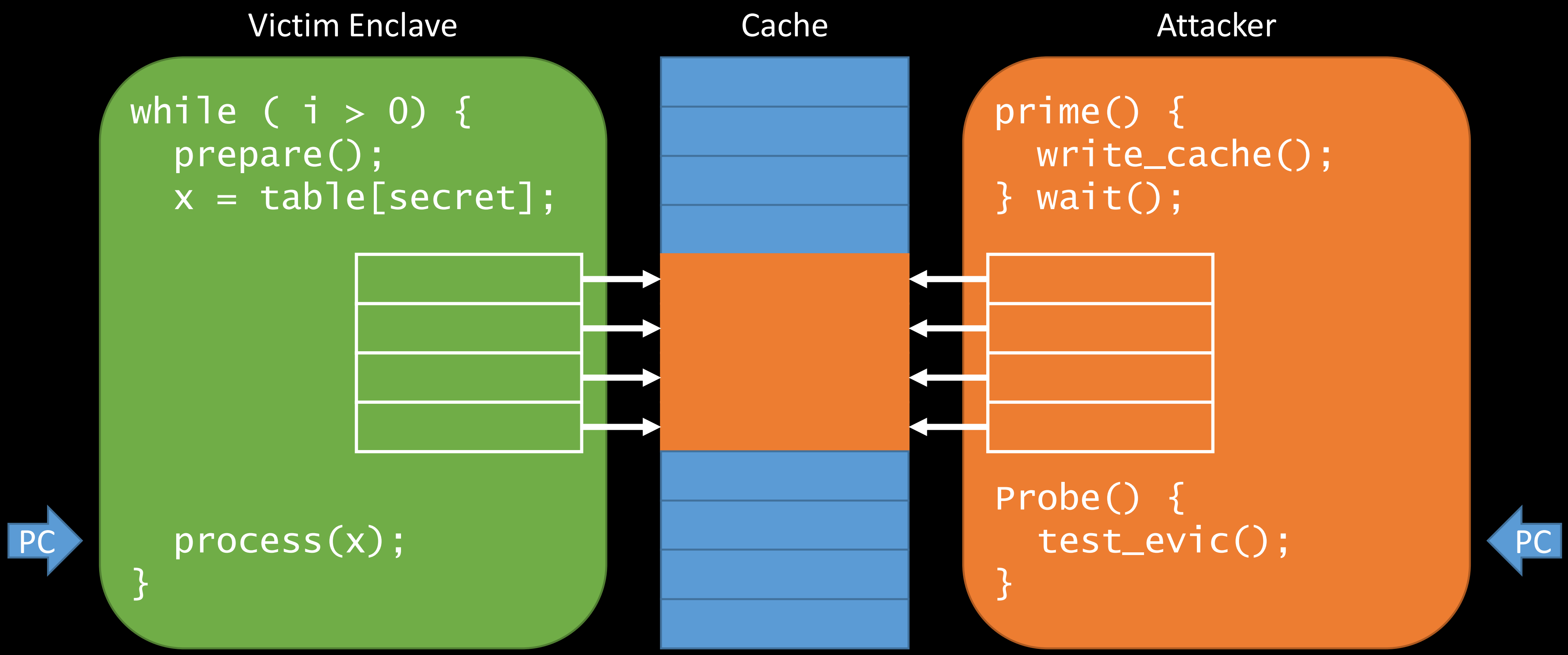
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];

  process(x);
}
```

```
prime() {
  write_cache();
} wait();

Probe() {
  test_evic();
}
```

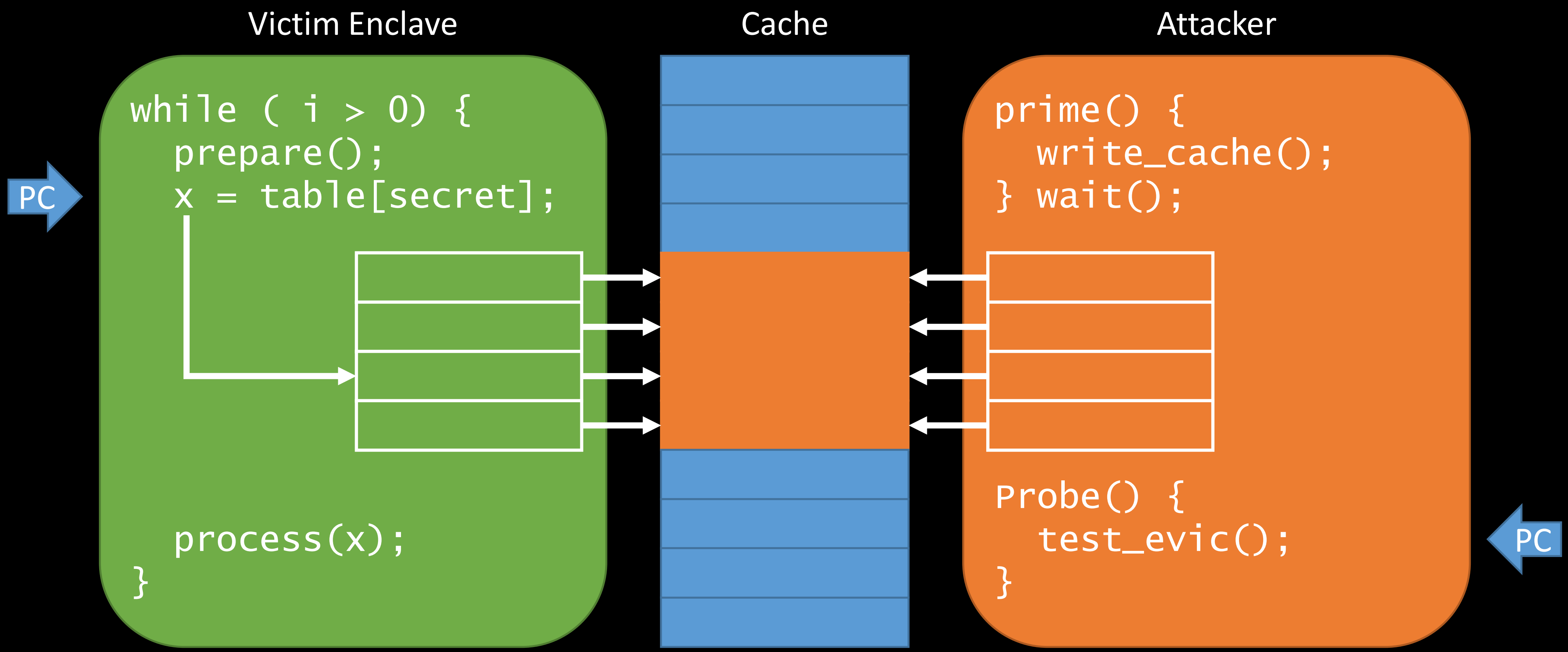
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

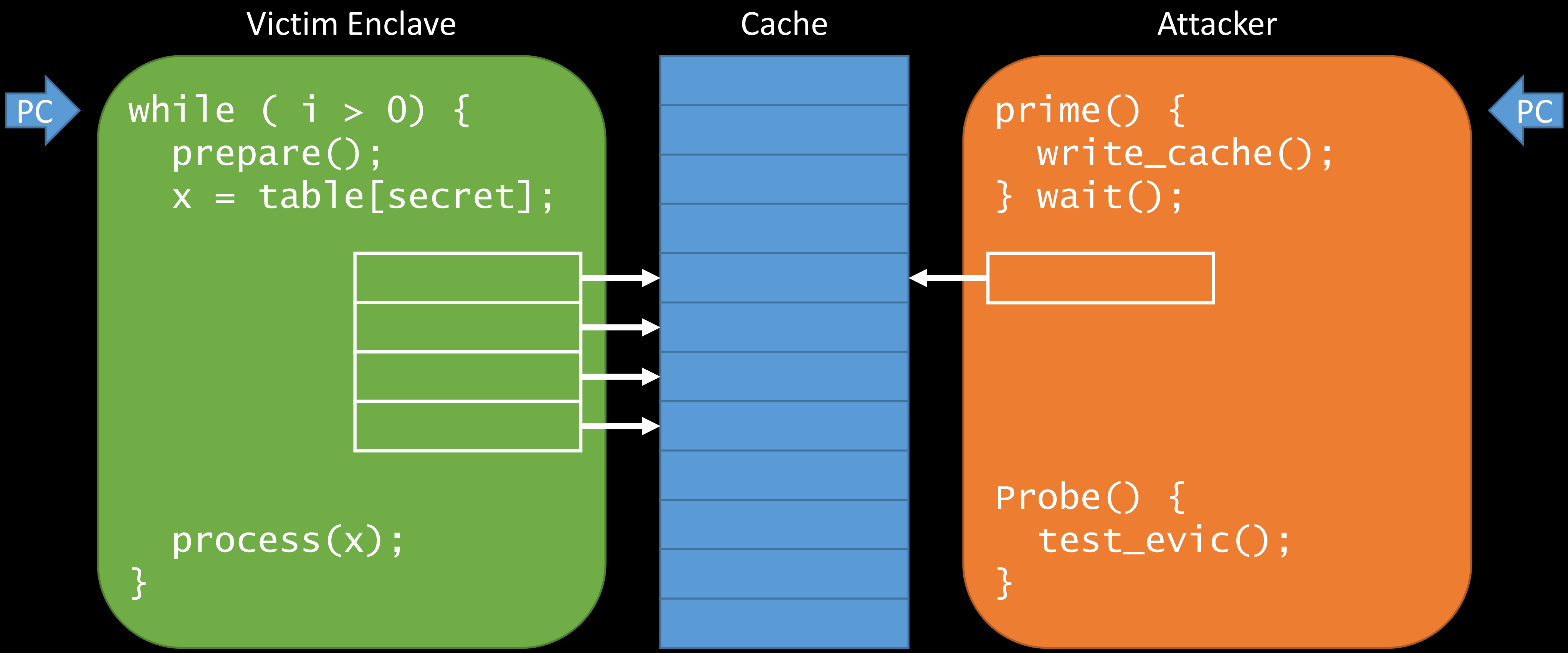
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

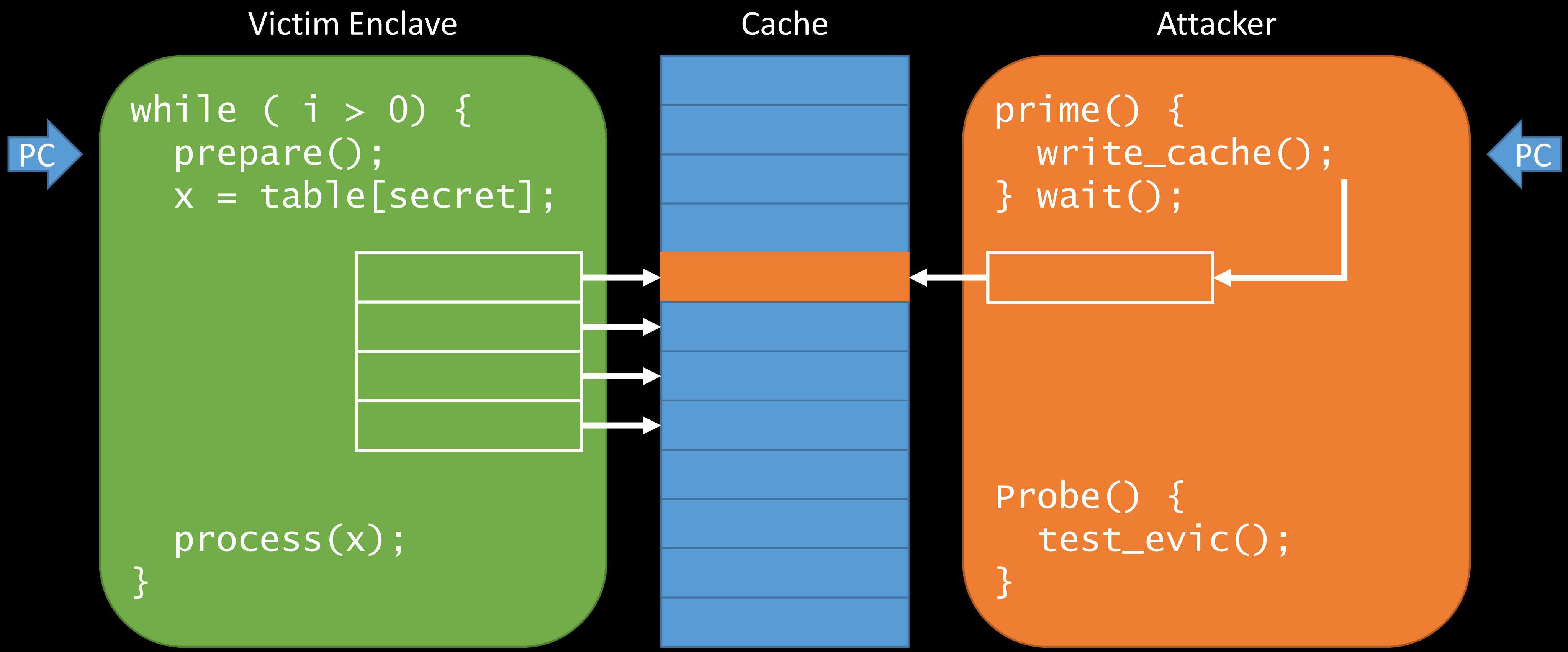
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

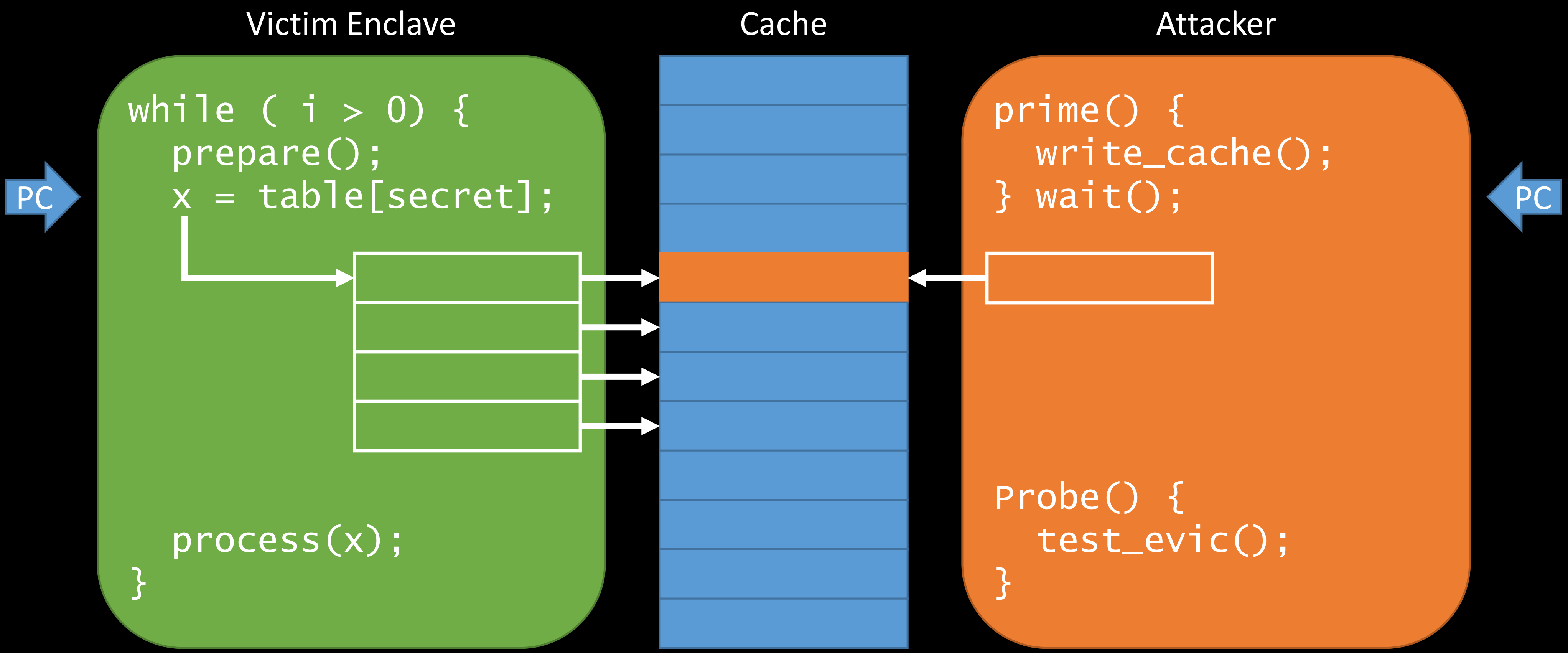
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

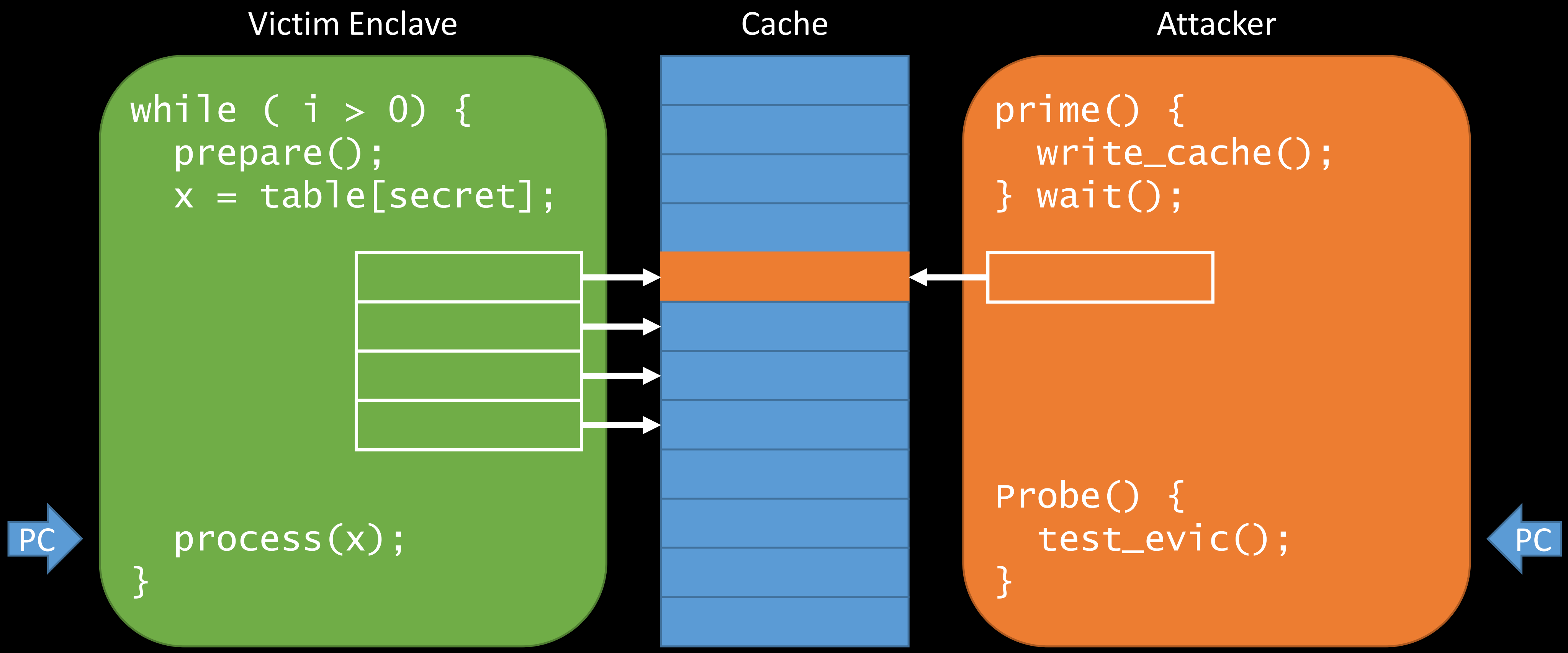
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

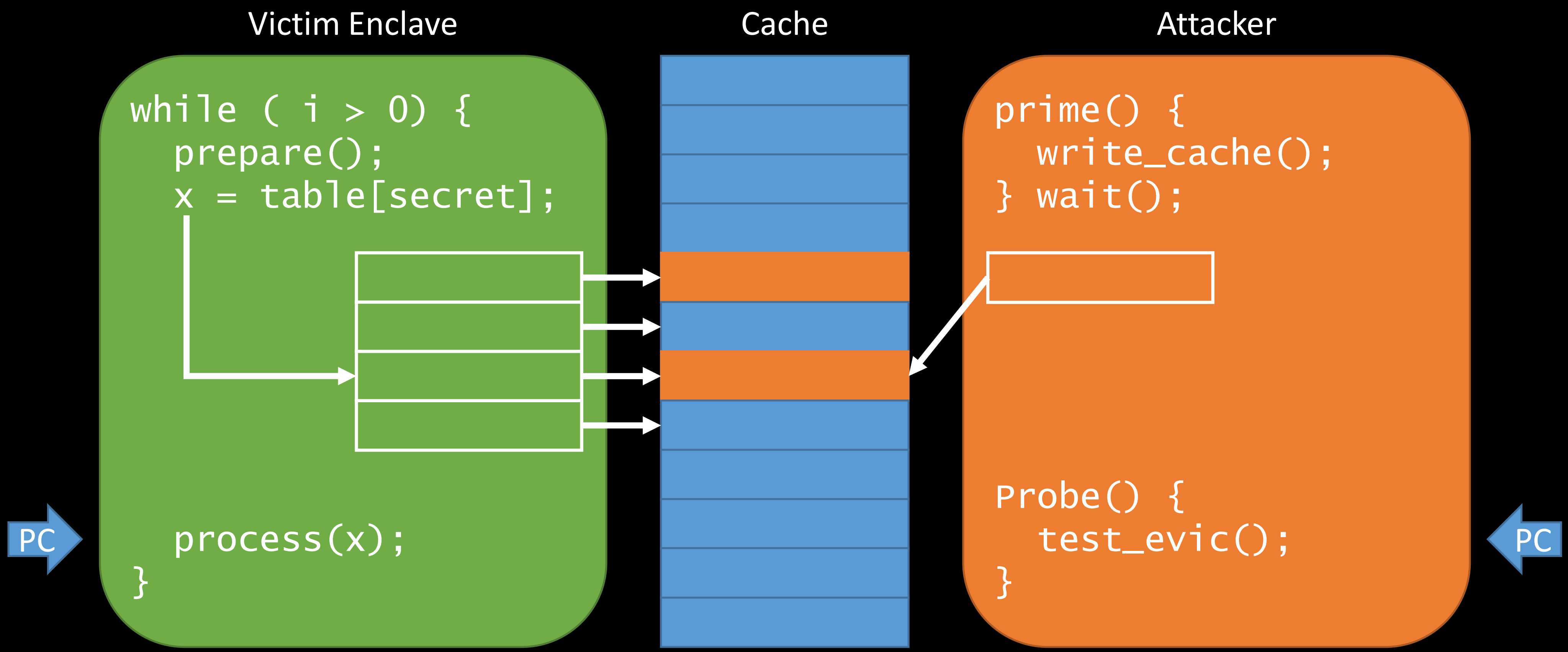
Spatial vs. Temporal Resolution



```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

Spatial vs. Temporal Resolution

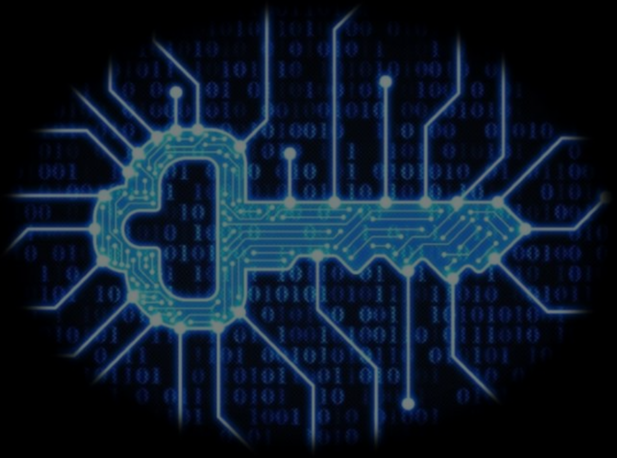


```
while ( i > 0) {
  prepare();
  x = table[secret];


  process(x);
}
```

```
prime() {
  write_cache();
} wait();


Probe() {
  test_evic();
}
```

Our Attack Use-Cases

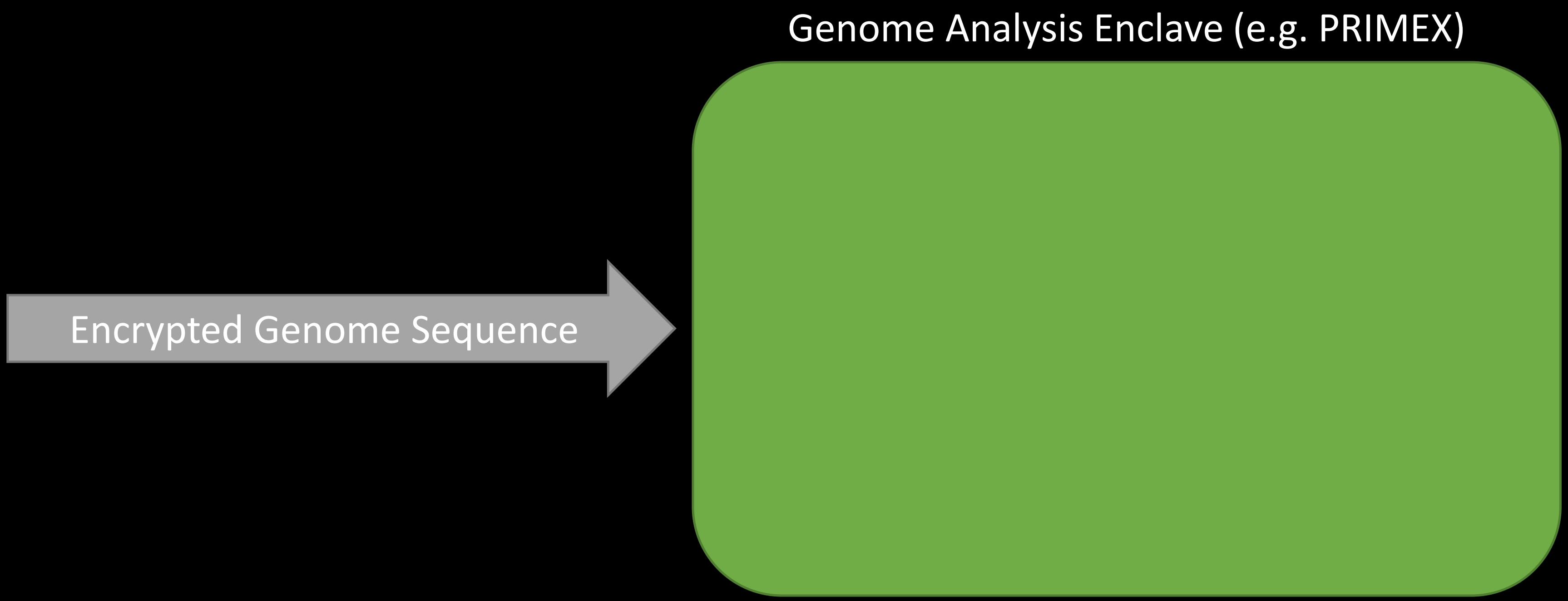
Extracting 2048-bit RSA decryption key



[arXiv:1702.07521]

Extracting genome sequences

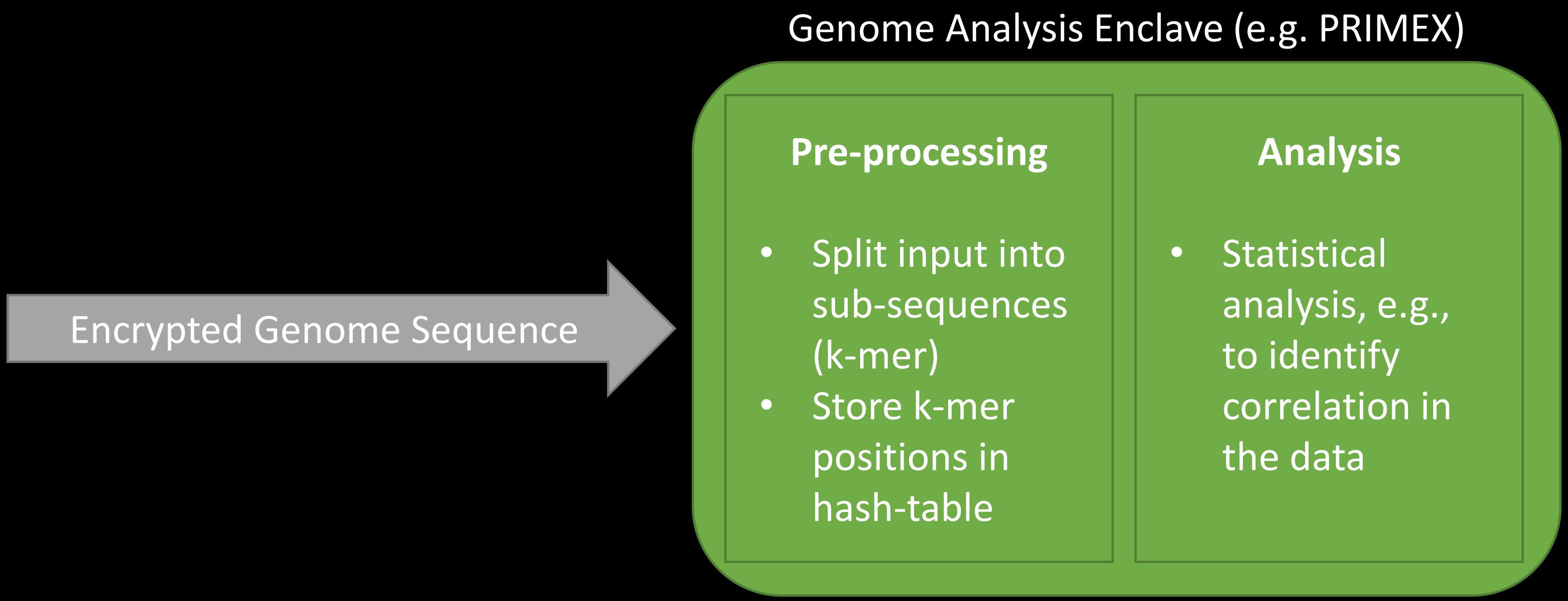
Genome Sequencing



Genome Analysis Enclave (e.g. PRIMEX)

Encrypted Genome Sequence

Genome Sequencing



Genome Analysis Enclave (e.g. PRIMEX)

Encrypted Genome Sequence

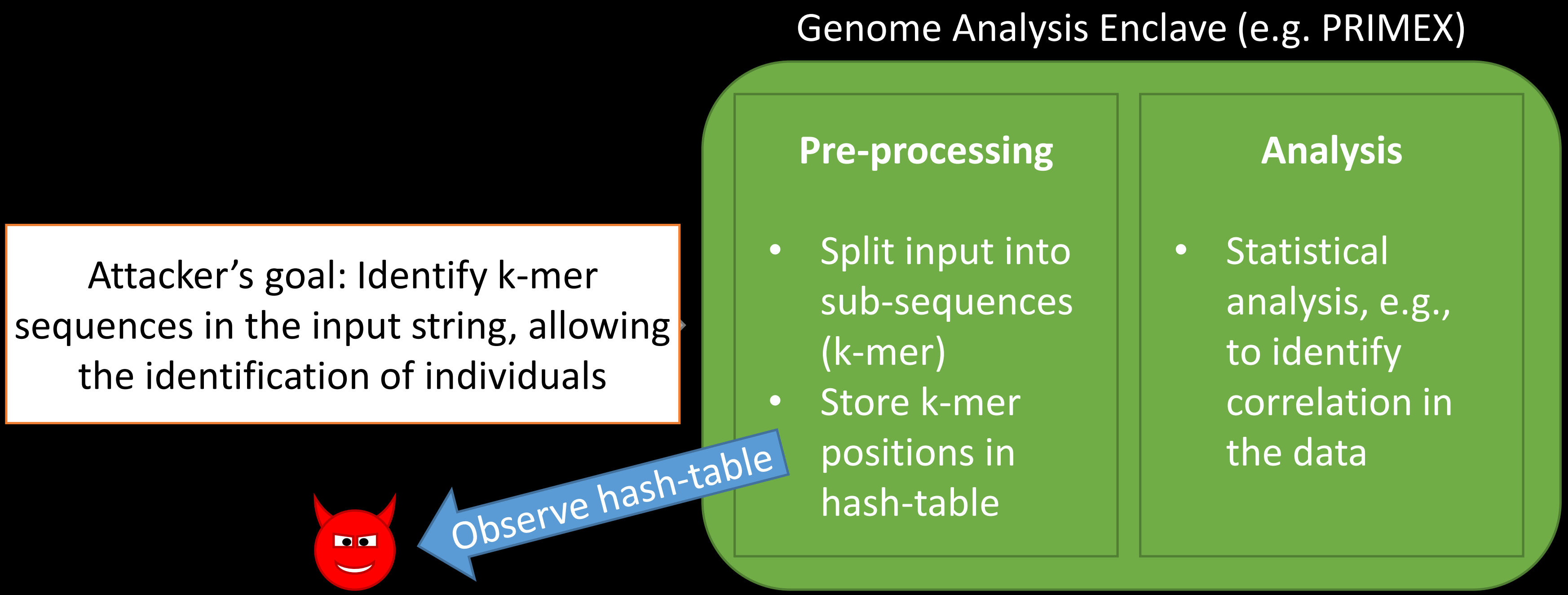
Pre-processing

- Split input into sub-sequences (k-mer)
- Store k-mer positions in hash-table

Analysis

- Statistical analysis, e.g., to identify correlation in the data

Genome Sequencing



Genome Analysis Enclave (e.g. PRIMEX)

Pre-processing

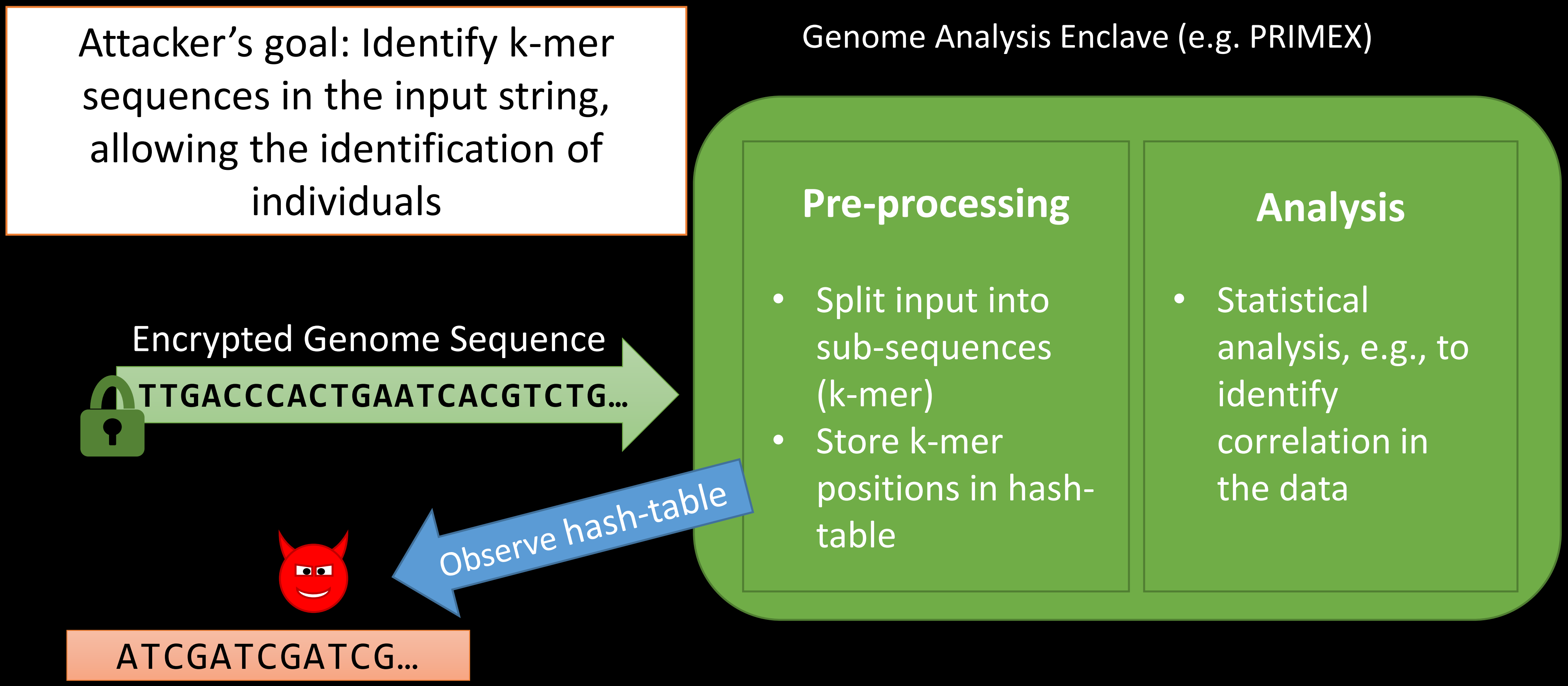
- Split input into sub-sequences (k-mer)
- Store k-mer positions in hash-table

Analysis

- Statistical analysis, e.g., to identify correlation in the data

Attacker’s goal: Identify k-mer sequences in the input string, allowing the identification of individuals

Genome Sequencing



Attacker’s goal: Identify k-mer sequences in the input string, allowing the identification of individuals

Genome Analysis Enclave (e.g. PRIMEX)

Pre-processing

- Split input into sub-sequences (k-mer)
- Store k-mer positions in hash-table

Analysis

- Statistical analysis, e.g., to identify correlation in the data

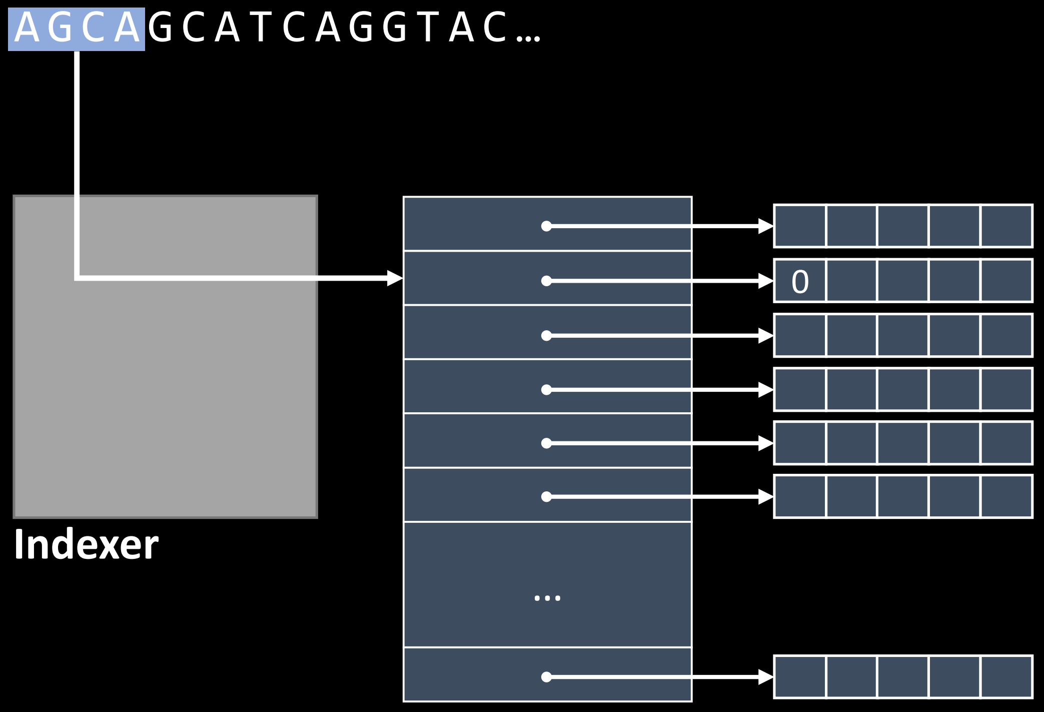
Human Genome

- Nucleobases
  - Adenine (A)
  - Cytosine (C)
  - Guanine (G)
  - Thymine (T)
- Microsatellite
  - Forensic analysis
  - Genetic fingerprinting
  - Kinship analysis

```
TTGACCCACTGAATCACGTCTGACCGCGCGTACGCGG
TCACTTGCGGTGCCGTTTTCTTTGTTACCGACGACCG
ACCAGCGACAGCCACCGCGCGCTCACTGCCACCAAAA
GAGTCATATCGATCGATCGATCGATCGATCGATCGAT
CGATCGATCGATCGATCGATCGATCGATCGATCATCA
CAGCCGACCAGTTTCTGGAACGTTCCCGATACTGGAA
CGGTCCTAATGCAGTATCCCACCCTCCTTCCATCGAC
GCCAGTCGAATCACGCCGCCAGCCACCGTCCGCCAGC
CGGCCAGAATACCGATGACTCGGCGGTCTCGTGTCGG
TGCCGGCCTCGCAGCCATTGTACTGGCCCTGGCCGCA
GTGTCGGCTGCCGCTCCGATTGCCGGGGCGCAGTCCG
CCGGCAGCGGTGCGGTCTCAGTCACCATCGGCGACGT
GGACGTCTCGCCTGCGAACCCAACCACGGGCACGCAG
GTGTTGATCACCCCGTCGATCAACAACTCCGGATCGG
CAAGCGGGTCCGCGCGCGTCAACGAGGTCACGCTGCG
CGGCGACGGTCTCCTCGCAACGGAAGACAGCCTGGGG
```

Genome Preprocessing

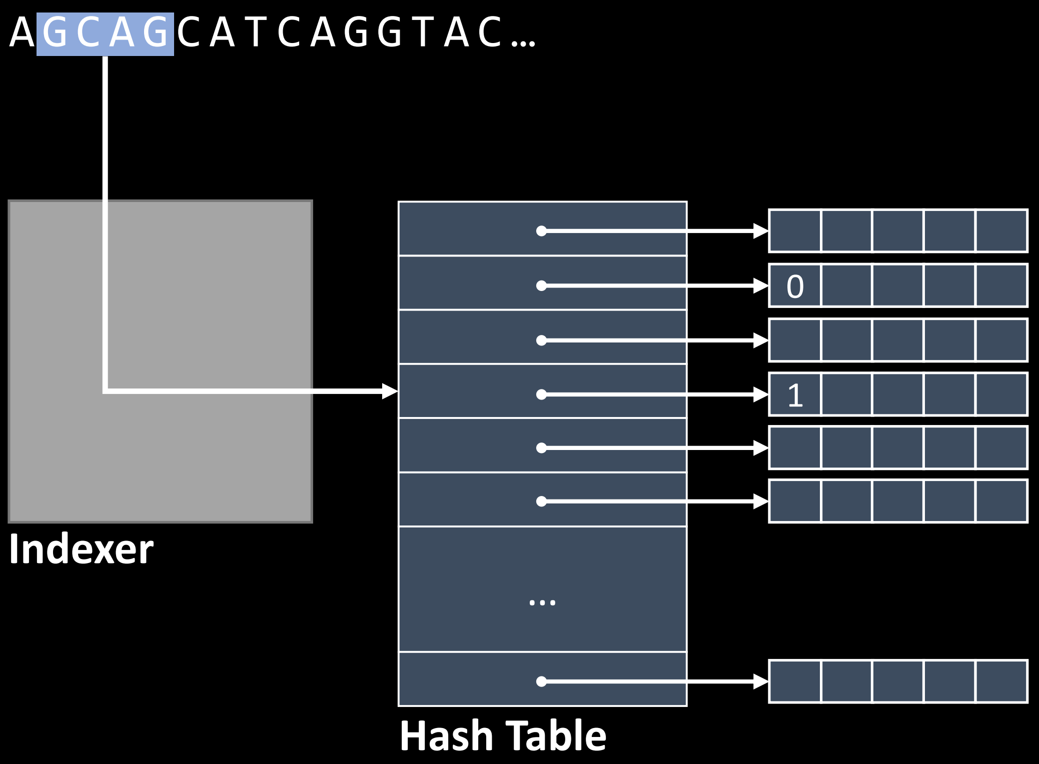
AGCAGCATCAGGTAC…



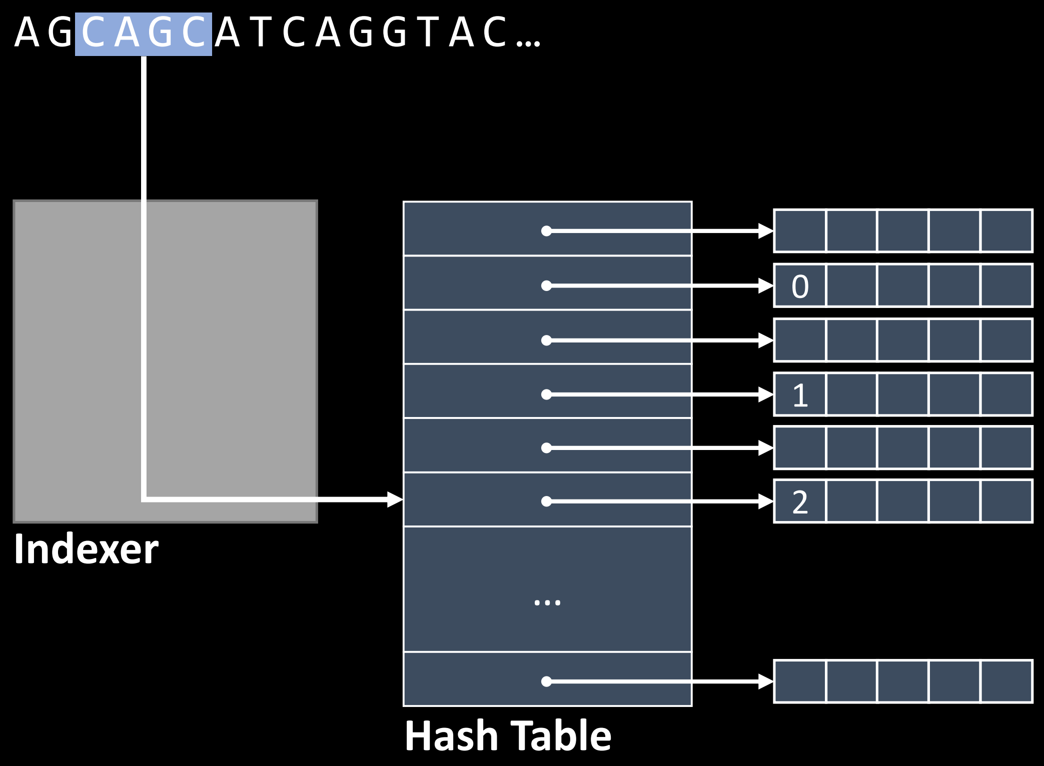
Indexer

Hash Table

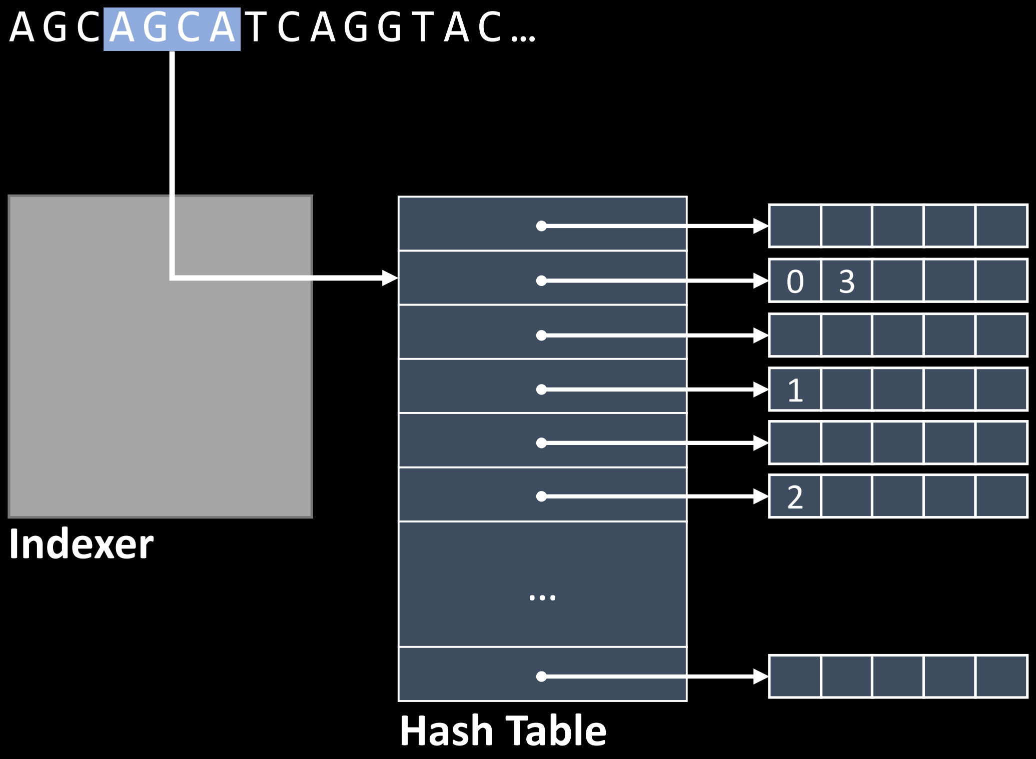
Genome Preprocessing



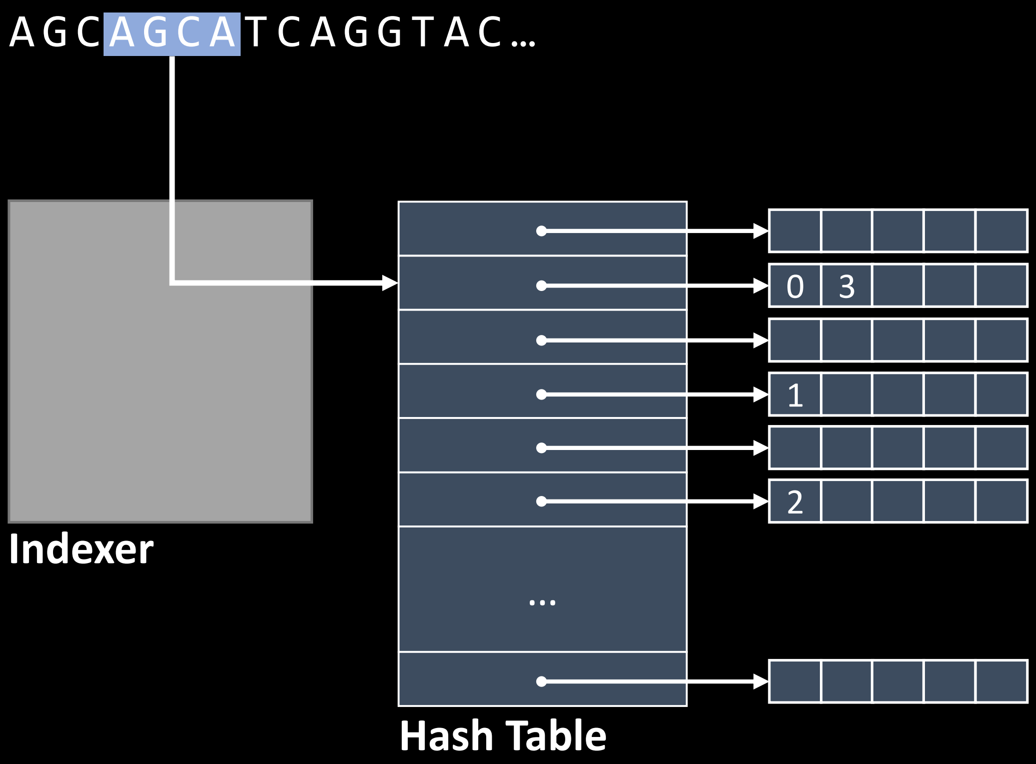
Genome Preprocessing



Genome Preprocessing



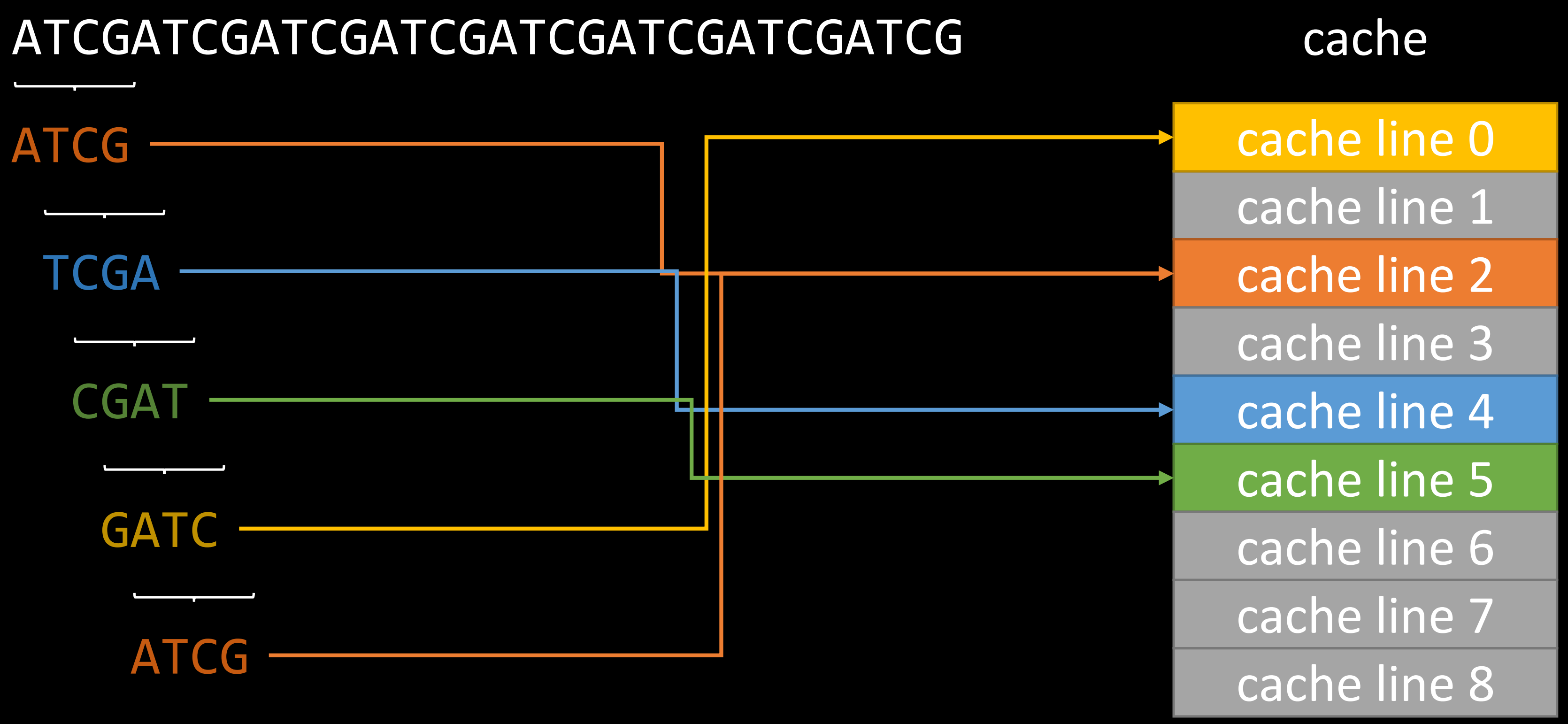
Genome Preprocessing



- Hash table access pattern
  - Hash table entry 8 bytes
  - Cache line size 64 bytes
  - Collisions
- Genome unstructured
- Microsatellites structured

TTGACCCACTGAATCACGTCTGACCGCGCGTACGCGGTCACTTGC
GGTGCCGTTTTCTTTGTTACCGACGACCGACCAGCGACAGCCACC
GCGCGCTCACTGCCACCAAAAGAGTCATATCGATCGATCGATCGA
TCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGAT
CATCACAGCCGACCAGTTTCTGGAACGTTCCCGATACTGGAACGG
TCCTAATGCAGTATCCCACCCTCCTTCCATCGACGCCAGTCGAAT
CACGCCGCCAGCCACCGTCCGCCAGCCGGCCAGAATACCGATGAC
TCGGCGGTCTCGTGTCGGTGCCGGCCTCGCAGCCATTGTACTGGC
CCTGGCCGCAGTGTCGGCTGCCGCTCCGATTGCCGGGGCGCAGTC
CGCCGGCAGCGGTGCGGTCTCAGTCACCATCGGCGACGTGGACGT
CTCGCCTGCGAACCCAACCACGGGCACGCAGGTGTTGATCACCCC

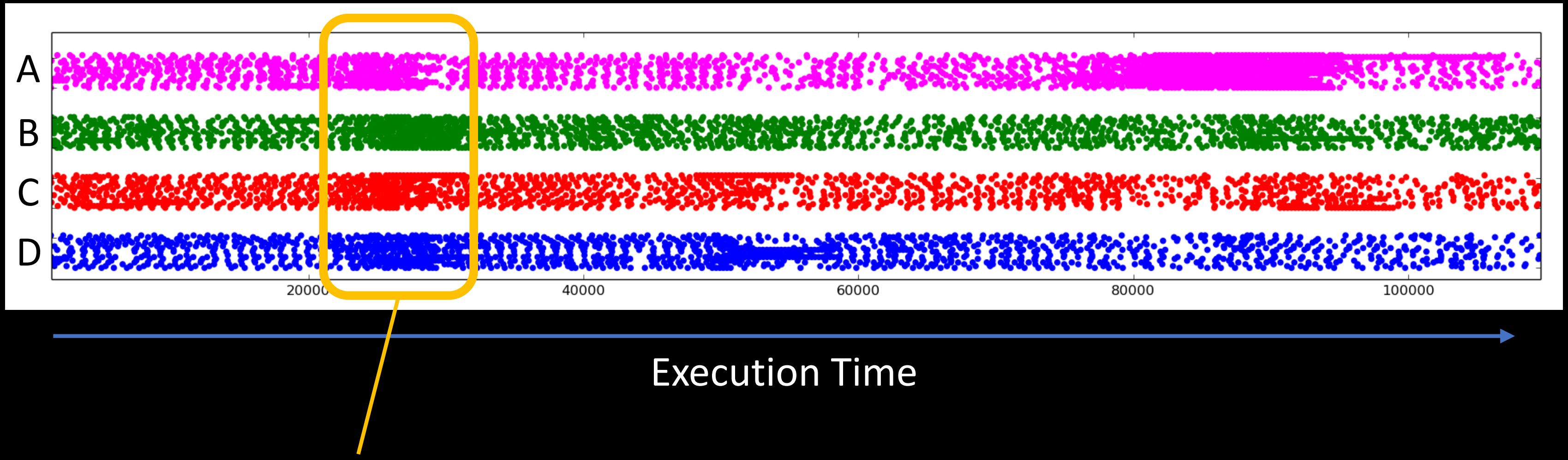
Microsatellites and Processed k-mers



The microsatellite will activate cache lines 2, 4, 5 and 0 repeatedly

Genome Sequencing Attack Results

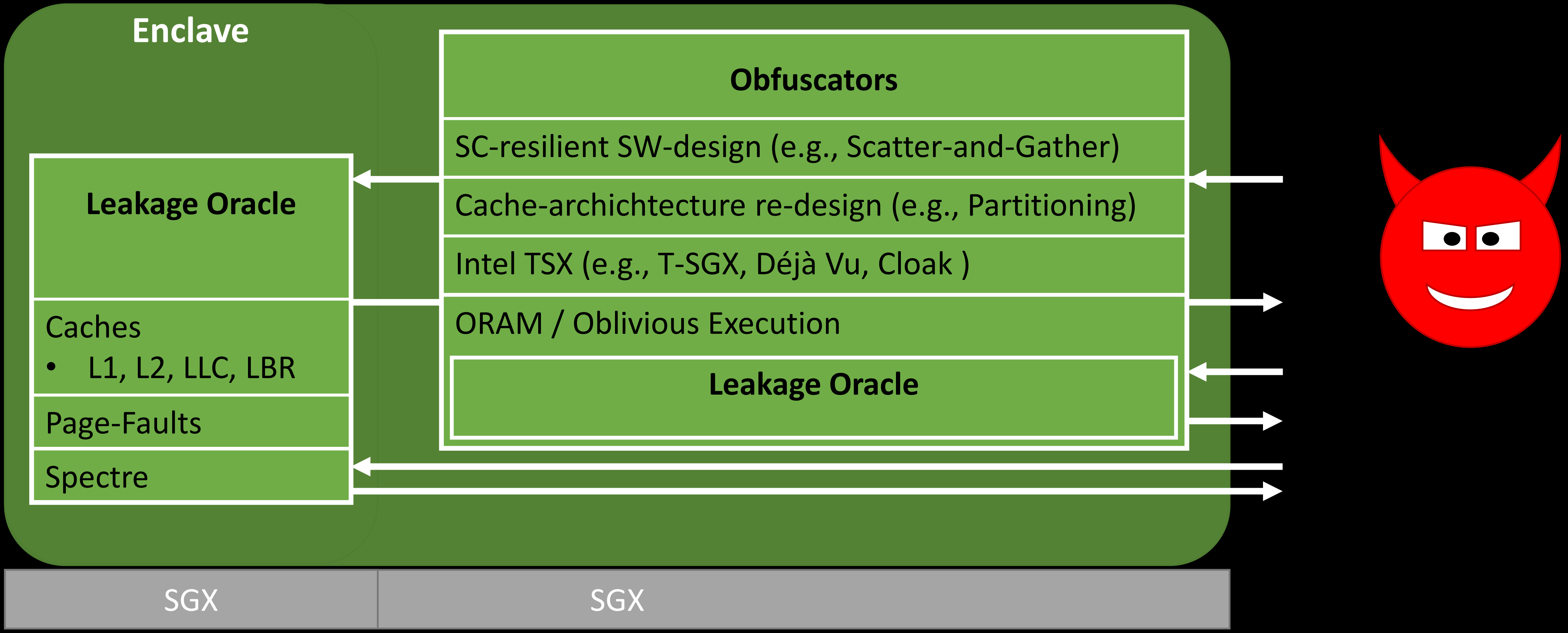
- Monitor cache lines associated to satellite
- High activity in cache lines reveal occurrence of satellite in input string



Execution Time

Activity in all related cache lines

SGX Side Channels & Defenses



Enclave

Leakage Oracle

Caches
- L1, L2, LLC, LBR

Page-Faults

Spectre

Obfuscators

SC-resilient SW-design (e.g., Scatter-and-Gather)

Cache-archichtecture re-design (e.g., Partitioning)

Intel TSX (e.g., T-SGX, Déjà Vu, Cloak)

ORAM / Oblivious Execution

Leakage Oracle

SGX

SGX

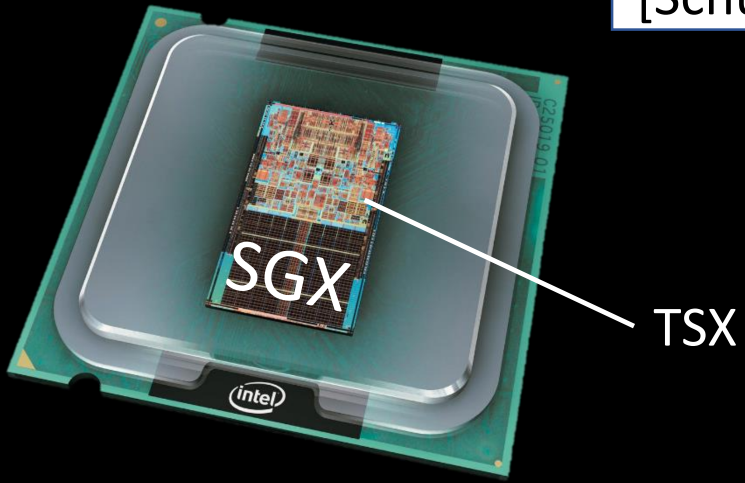
SGX Specific Side-Channel Defenses Using TSX

- Intel TSX is a hardware mechanism to allow synchronous memory transactions
- TSX is **not** available on all SGX-enable processors

T-SGX: Uses TSX to detect enclave interrupt [Shih et al., NDSS’17]

Cloak: Prime cache before accessing sensitive data [Schuster et al., USENIX 2017]

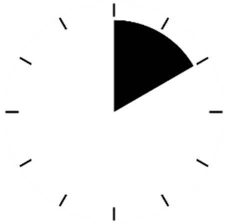
Déjà Vu : Uses TSX to detect enclave slowdown [Chen et al., AsiaCCS’17]



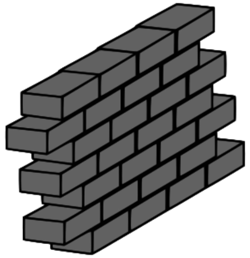
TSX: Transactional Synchronization Extensions

General Hardware-based Side-Channel Defenses

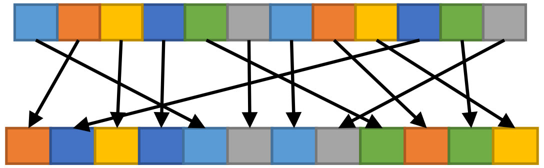
Temporal cache isolation



Cache partitioning / coloring

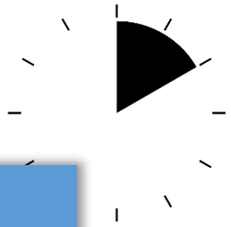


Randomized cache mappings



General Hardware-based Side-Channel Defenses

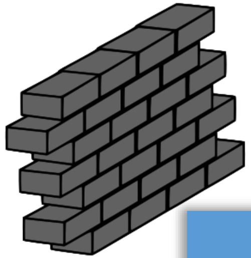
Temporal cache isolation



Problems

- Ineffective on SMT-enabled systems

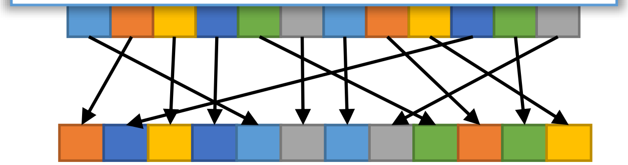
Cache partitioning / coloring



Problems

- Reduces the amount of cache available to individual software

Ranc pings



Problems

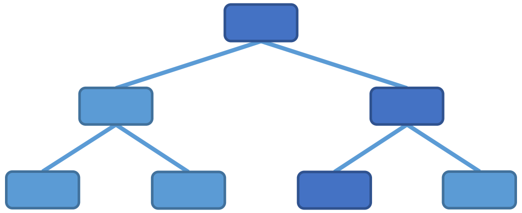
- Frequency analysis for randomization secret

General Software-only Side-Channel Defenses

Side-channel resilient software design

Monitoring for attack effects

Oblivious execution / ORAM



General Software-only Side-Channel Defenses

Side-channel resilient software design

Problems

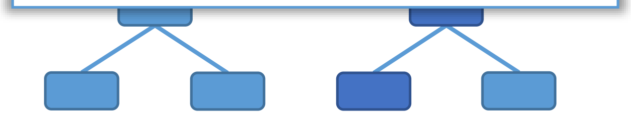
- Not applicable to all applications
- Manual software hardening required

Monitoring for attack effects

Problems

- Requires privileged entity (not available in SGX model)

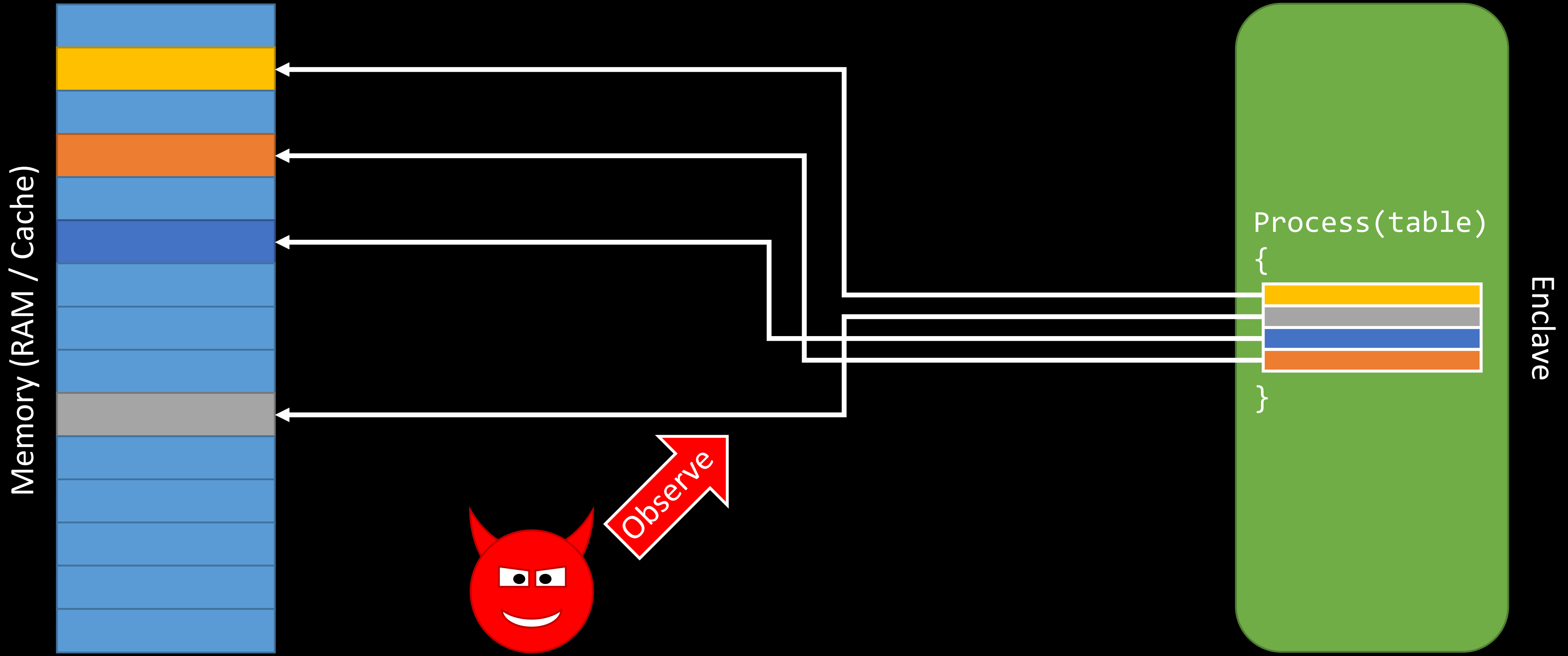
Obliv RAM



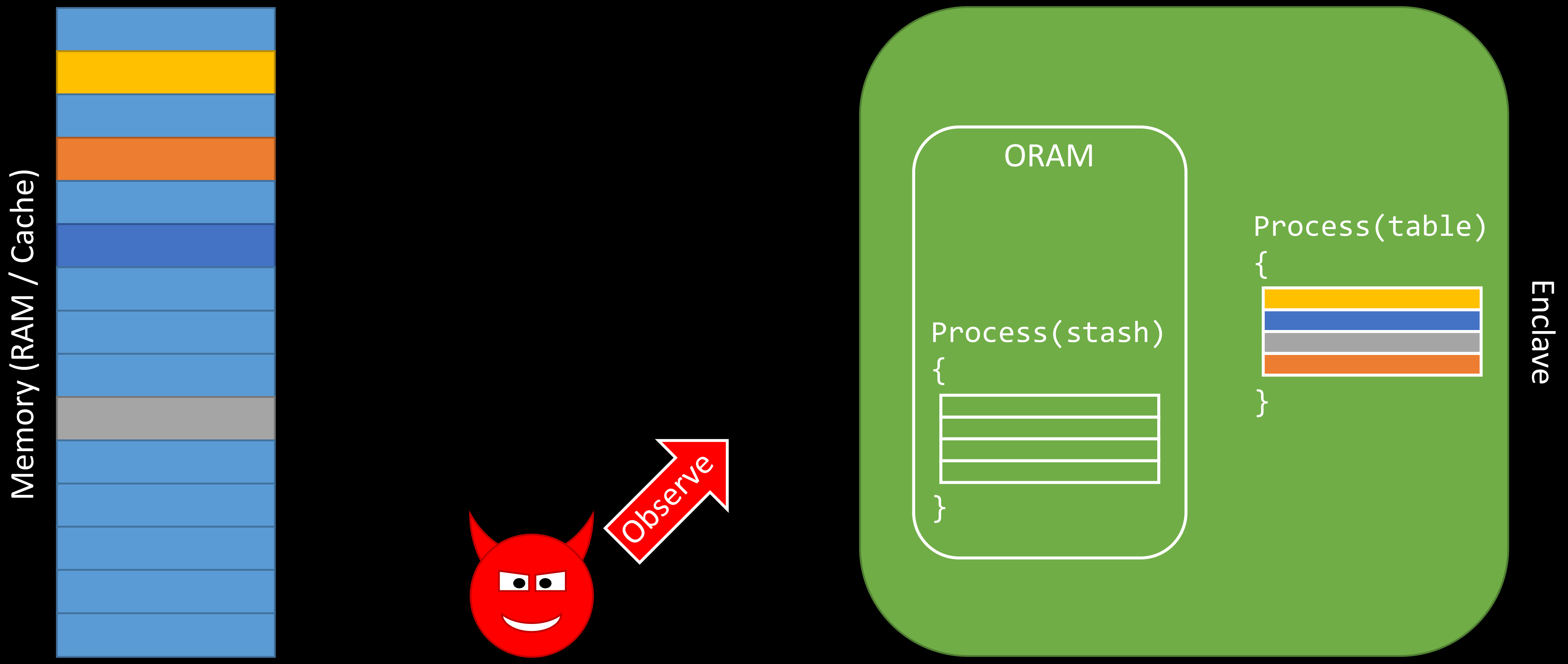
Problems

- Too inefficient, ORAM metadata needs to be protected as well

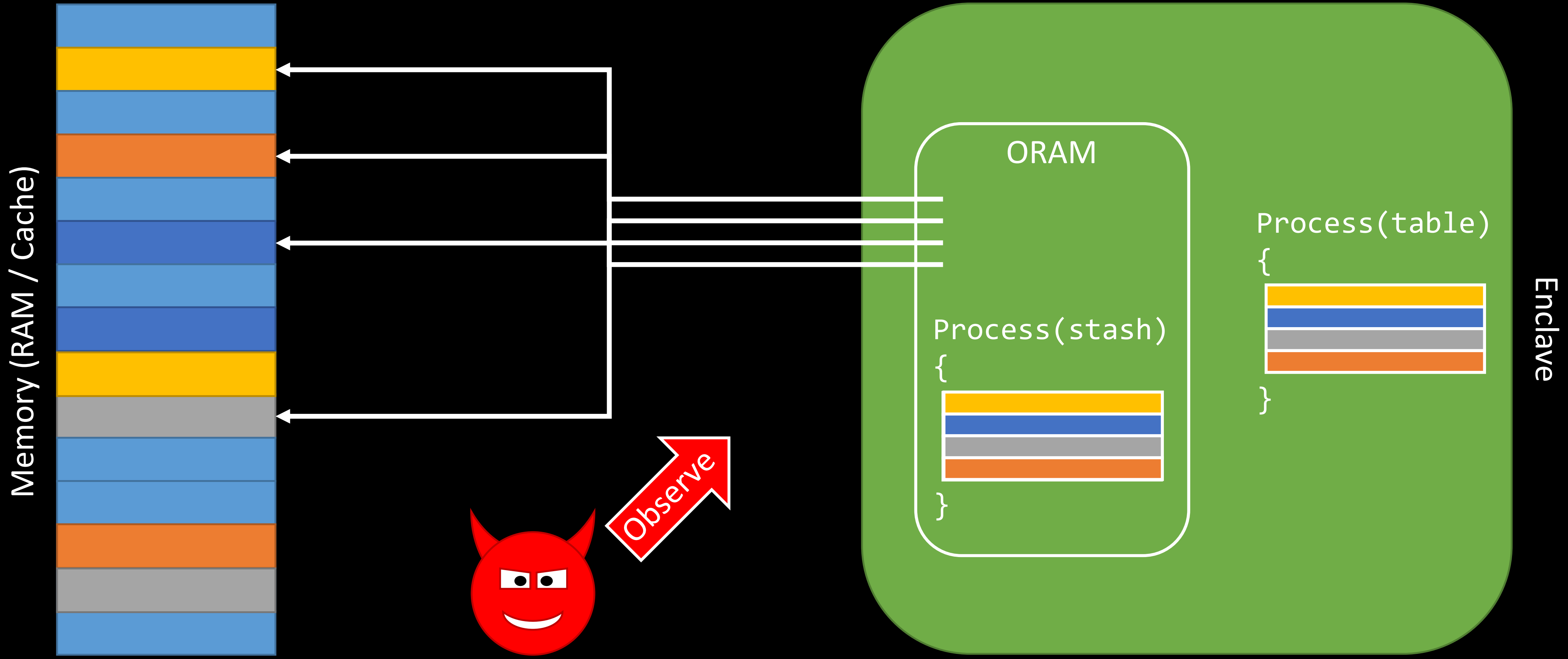
While(leak) { add_ORAM_layer(); }



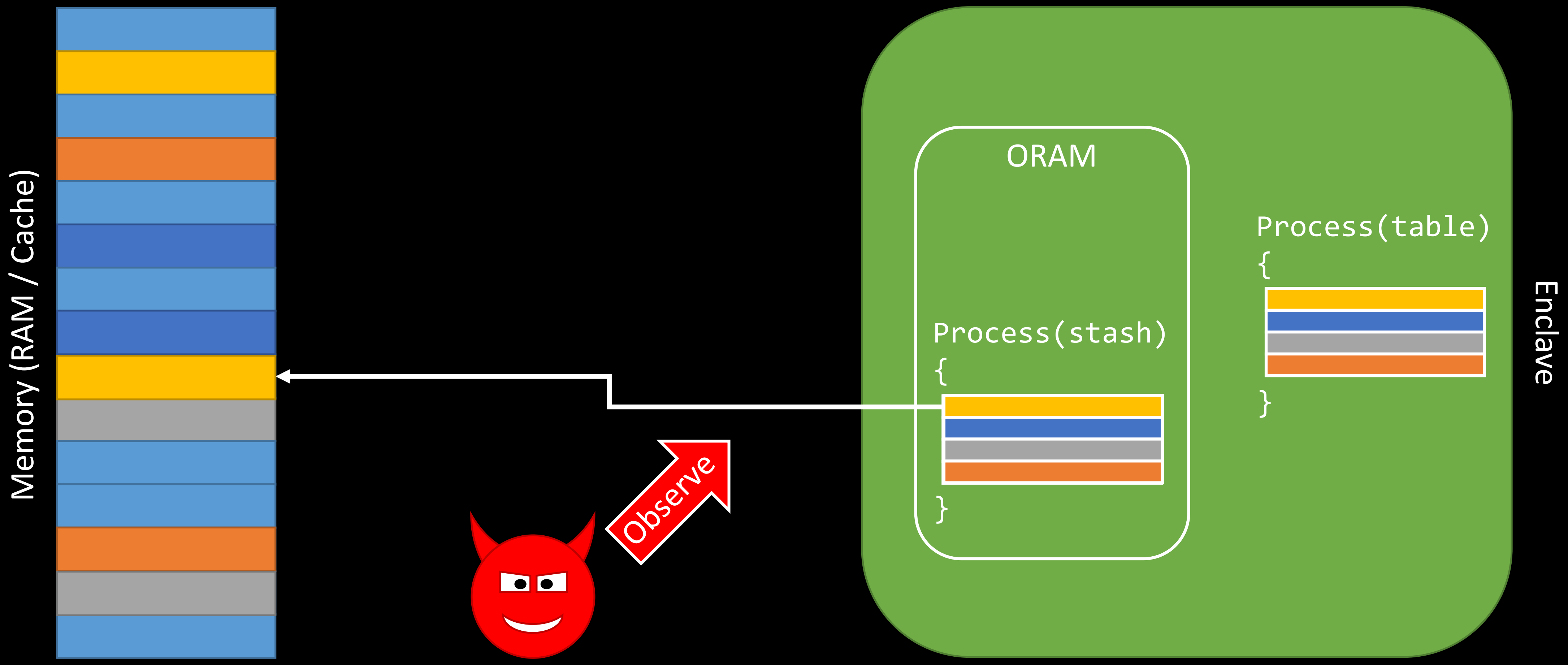
While(leak) { add_ORAM_layer(); }



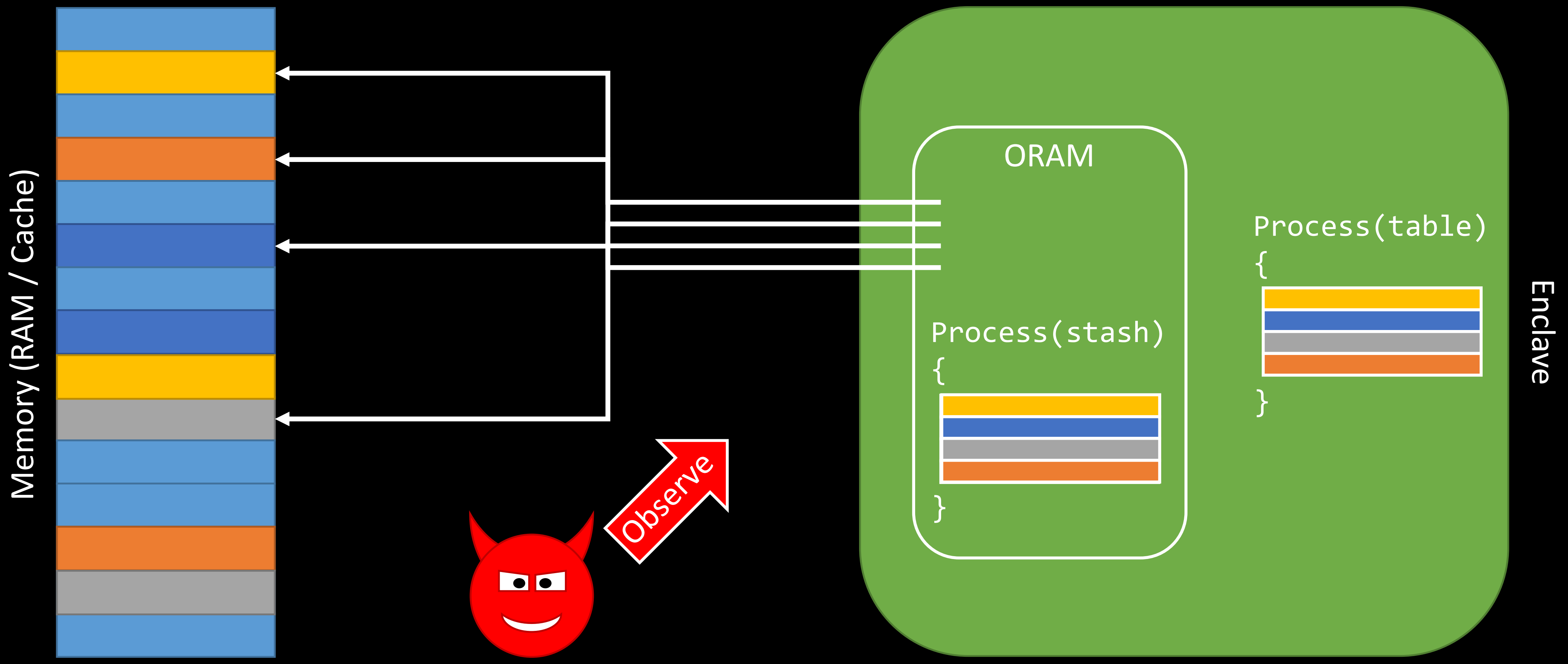
While(leak) { add_ORAM_layer(); }



While(leak) { add_ORAM_layer(); }

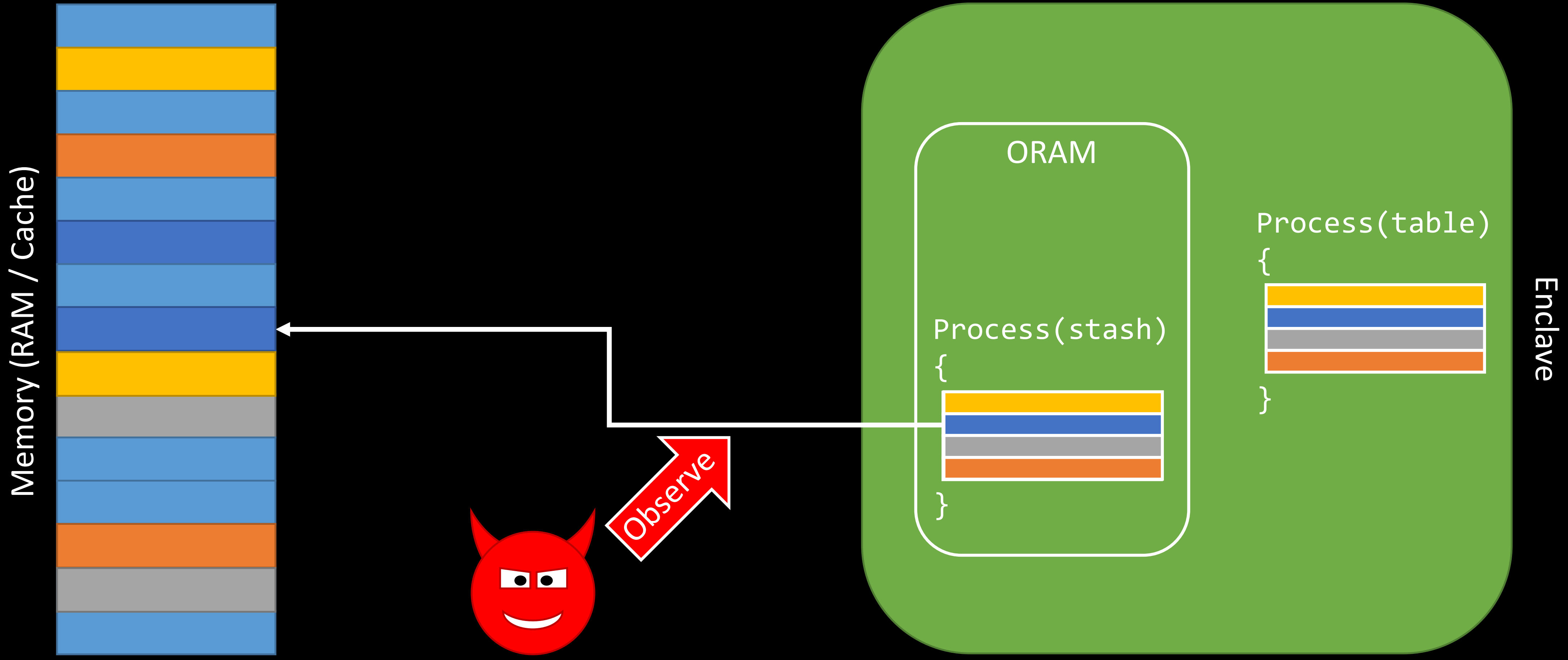


While(leak) { add_ORAM_layer(); }



CYSEC Cybersecurity TU Darmstadt

While(leak) { add_ORAM_layer(); }



Summary: SGX – All Problems Solved?

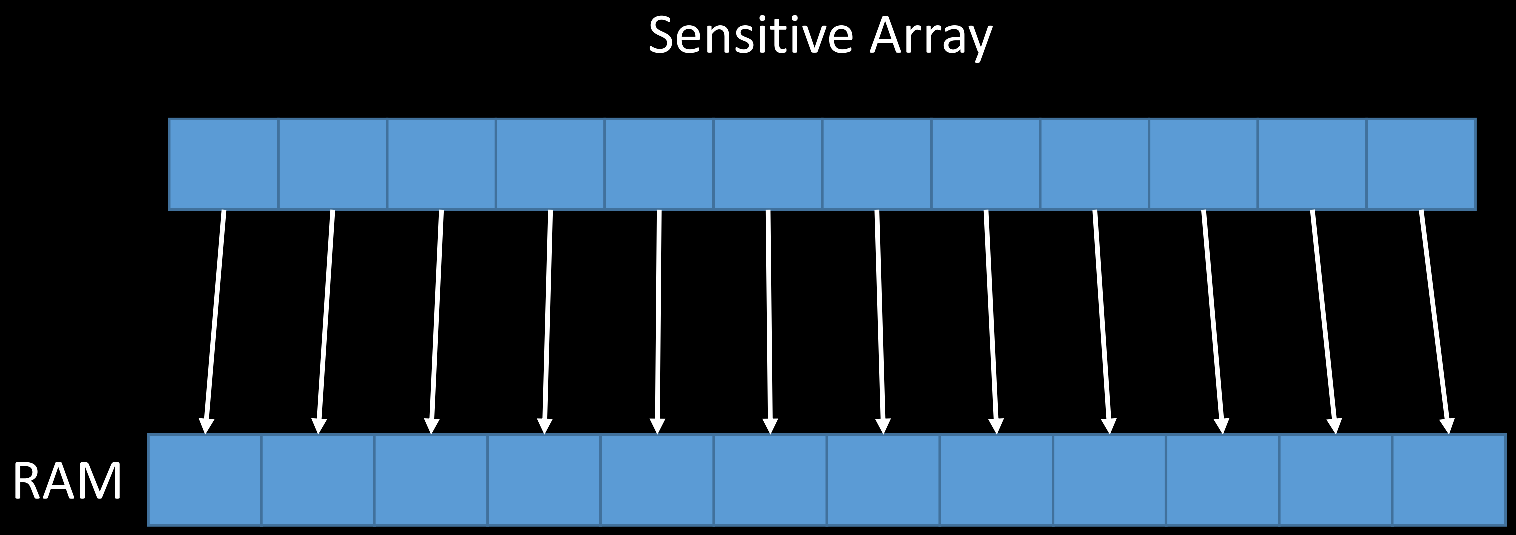
- Side channels more drastic than originally thought
- Current add-on defenses not practical or effective
- Academic research solutions mostly not deployed
- Generic software-only side-channel defenses required
  - No security expertise of enclave developers (no annotations)
  - Hardware extensions/features not available in *all* SGX CPUs

CYSEC
Cybersecurity
TU Darmstadt

Our Current Work:
Generic Software-only Side-Channel Defenses

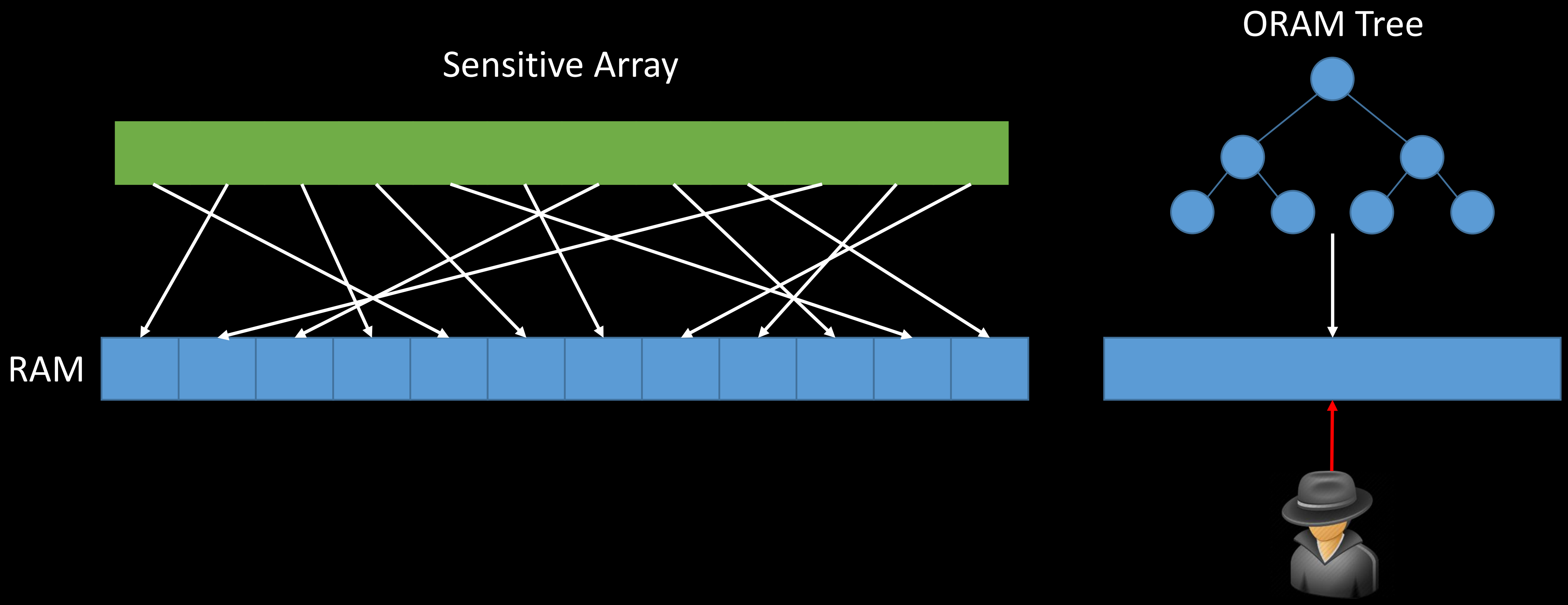
Our Current Work: Software-based Side-Channel Mitigations

[Brasser et al., DR. SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization, ArXiv]



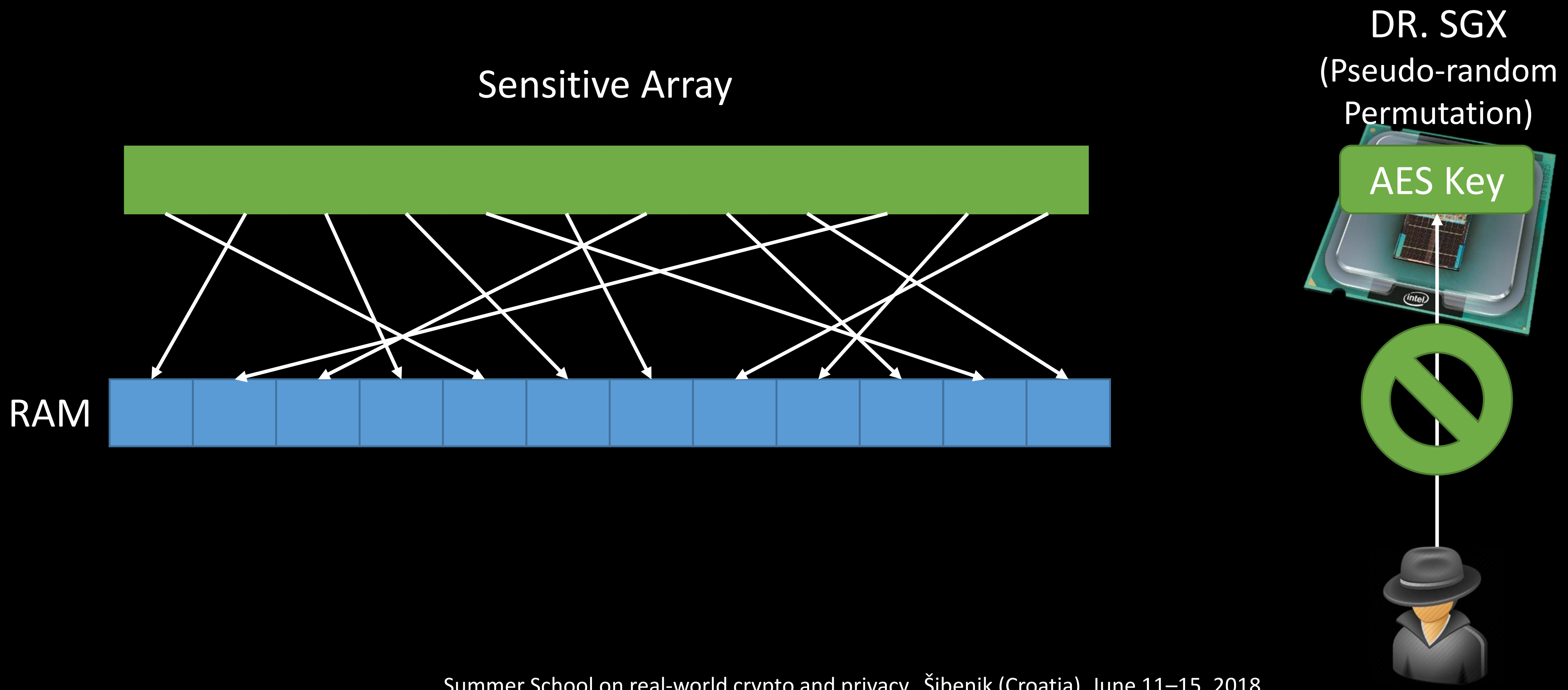
Our Current Work: Software-based Side-Channel Mitigations

[Brasser et al., DR. SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization, ArXiv]

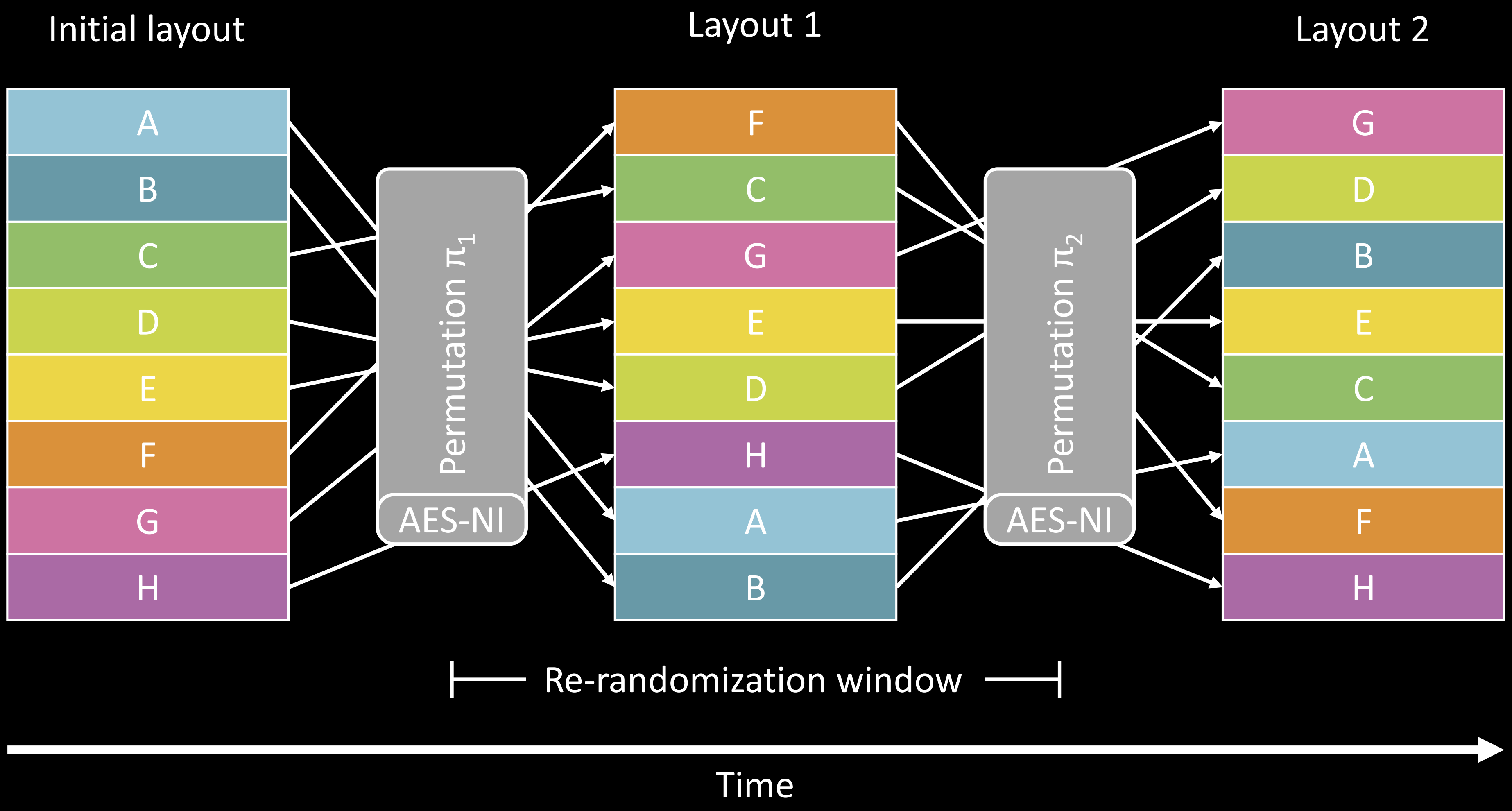


Our Current Work: Software-based Side-Channel Mitigations

[Brasser et al., DR. SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization, ArXiv]

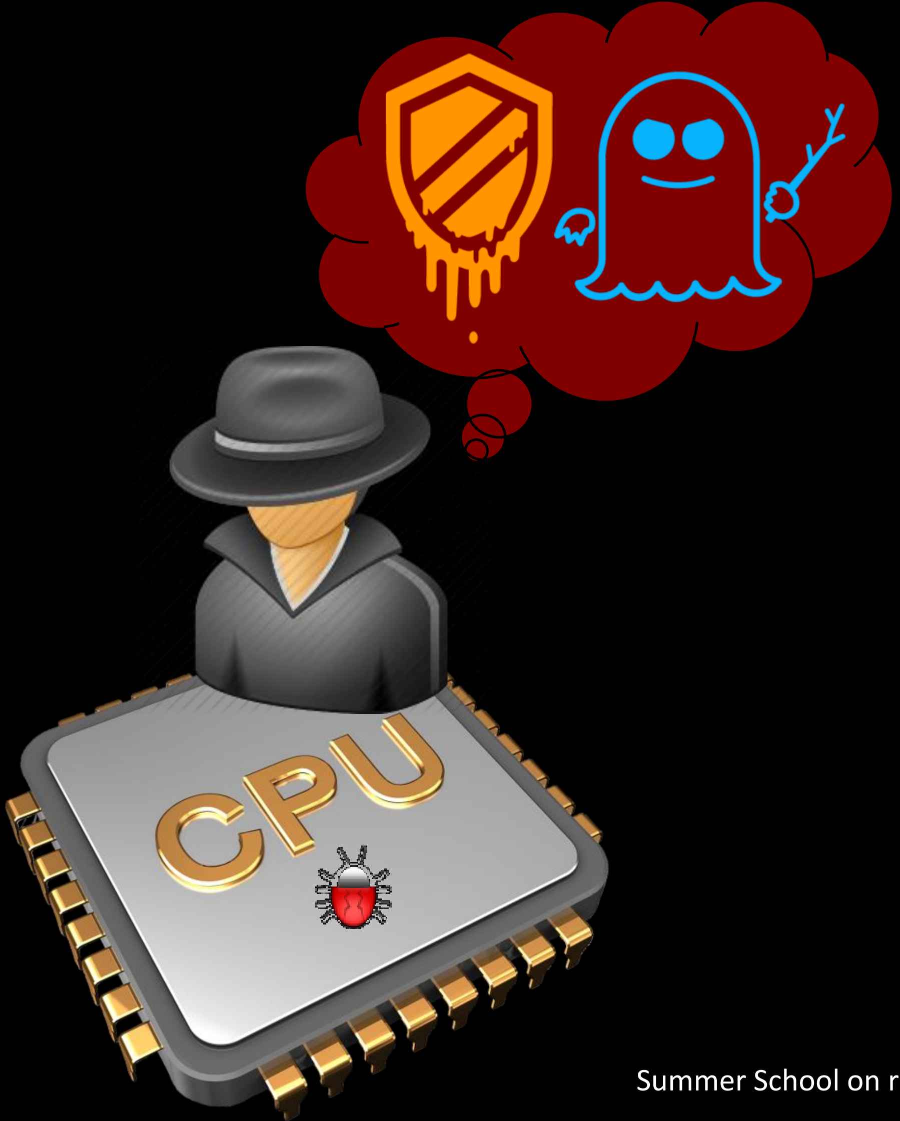


DR.SGX Re-randomization

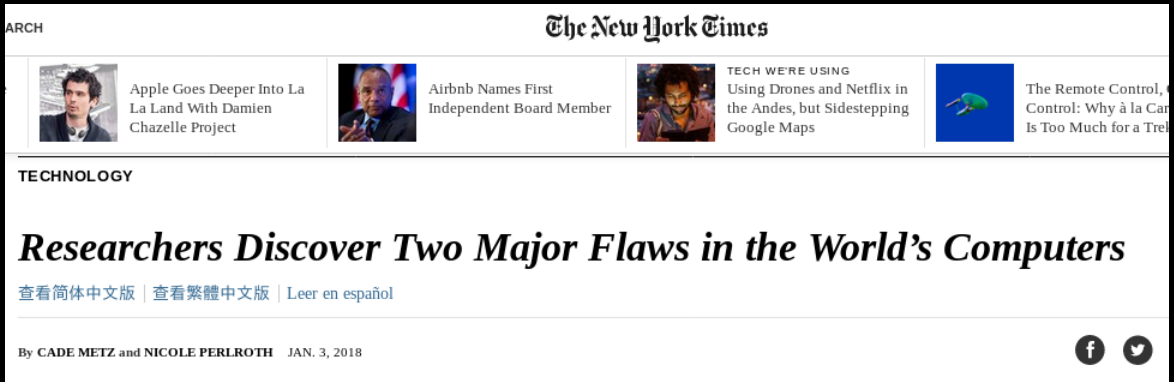


Meltdown and Spectre

So, you might have noticed...



So, you might have noticed...



The New York Times

Apple Goes Deeper Into La La Land With Damien Chazelle Project

Airbnb Names First Independent Board Member

TECH WE'RE USING
Using Drones and Netflix in the Andes, but Sidestepping Google Maps

The Remote Control, Control: Why à la Car Is Too Much for a Trel

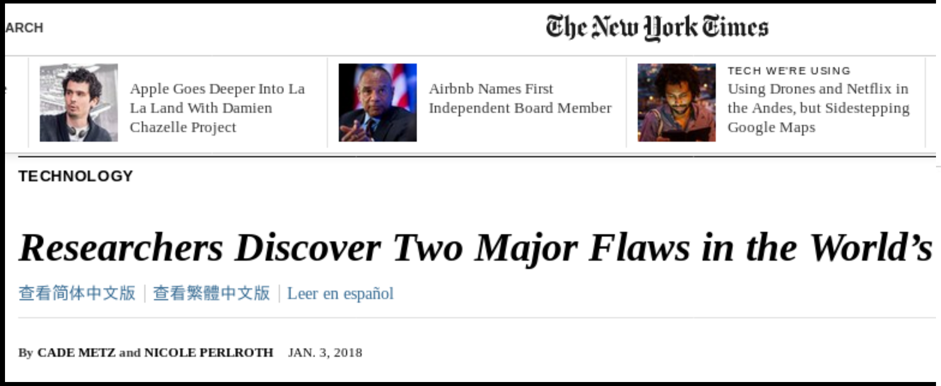
TECHNOLOGY

Researchers Discover Two Major Flaws in the World's Computers

查看简体中文版 | 查看繁體中文版 | Leer en español

By CADE METZ and NICOLE PERLROTH JAN. 3, 2018

So, you might have noticed…



ARCH The New York Times

Apple Goes Deeper Into La La Land With Damien Chazelle Project

Airbnb Names First Independent Board Member

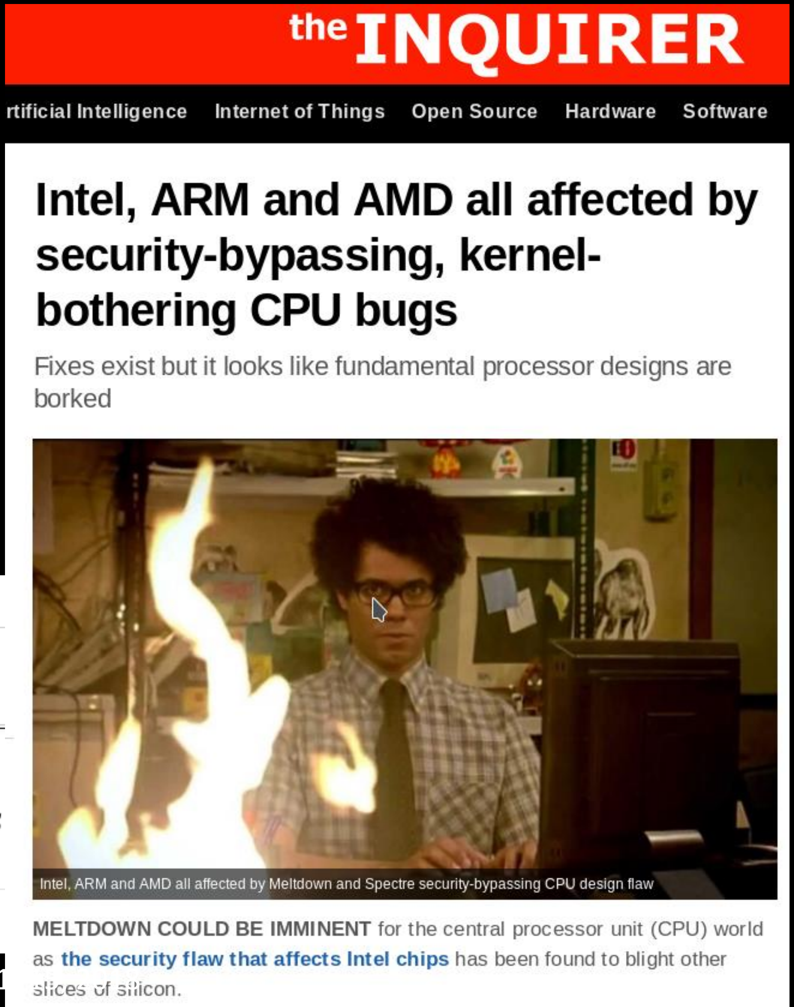
TECH WE'RE USING
Using Drones and Netflix in the Andes, but Sidestepping Google Maps

TECHNOLOGY

Researchers Discover Two Major Flaws in the World's

查看简体中文版 | 查看繁體中文版 | Leer en español

By CADE METZ and NICOLE PERLROTH JAN. 3, 2018



the INQUIRER

rtificial Intelligence Internet of Things Open Source Hardware Software

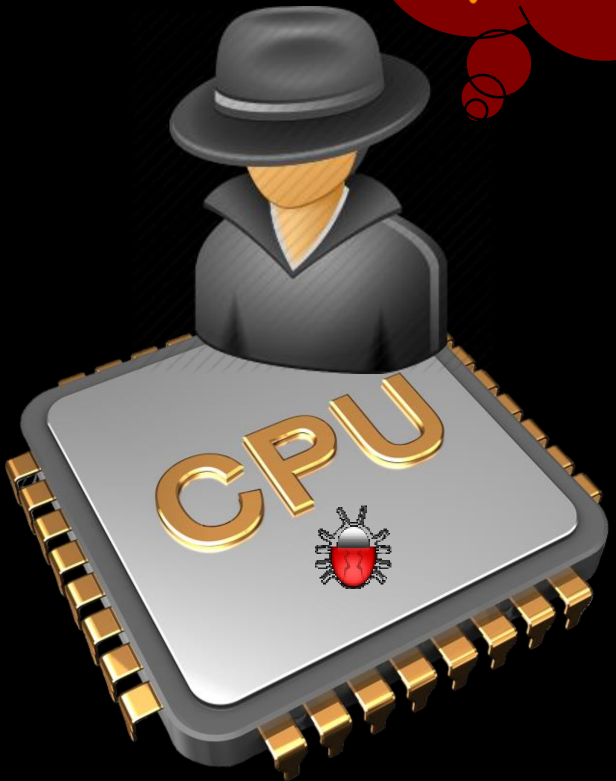
Intel, ARM and AMD all affected by security-bypassing, kernel-bothering CPU bugs

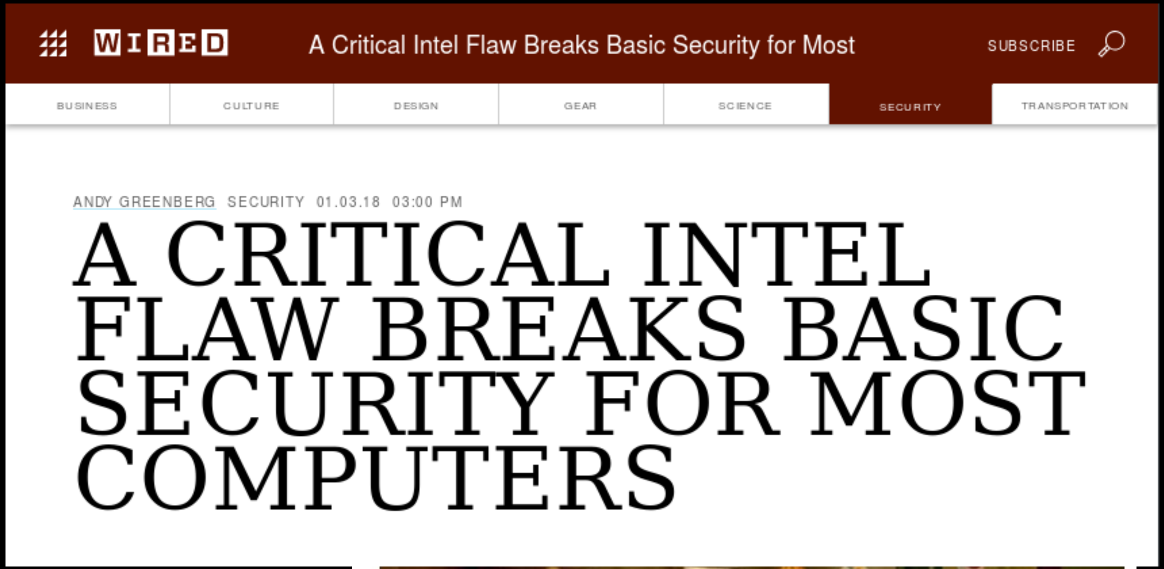
Fixes exist but it looks like fundamental processor designs are borked

Intel, ARM and AMD all affected by Meltdown and Spectre security-bypassing CPU design flaw

MELTDOWN COULD BE IMMINENT for the central processor unit (CPU) world as the security flaw that affects Intel chips has been found to blight other slices of silicon.

So, you might have noticed...

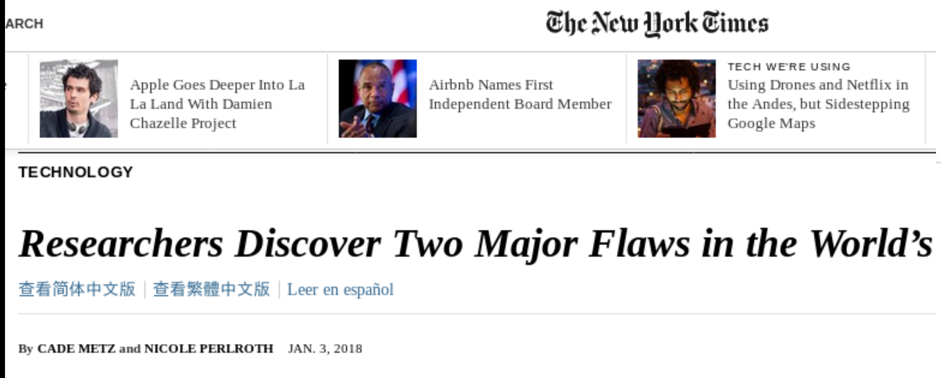




A Critical Intel Flaw Breaks Basic Security for Most

ANDY GREENBERG SECURITY 01.03.18 03:00 PM

A CRITICAL INTEL FLAW BREAKS BASIC SECURITY FOR MOST COMPUTERS



Apple Goes Deeper Into La La Land With Damien Chazelle Project

Airbnb Names First Independent Board Member

TECH WE'RE USING
Using Drones and Netflix in the Andes, but Sidestepping Google Maps

TECHNOLOGY

Researchers Discover Two Major Flaws in the World's

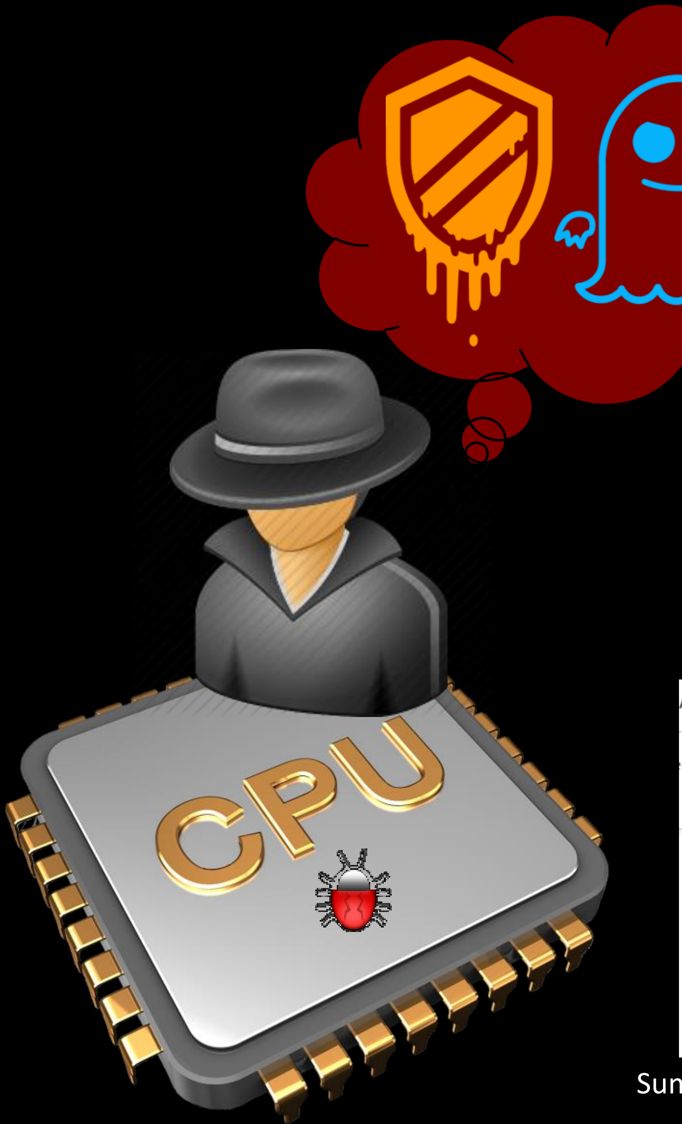
查看简体中文版 | 查看繁體中文版 | Leer en español

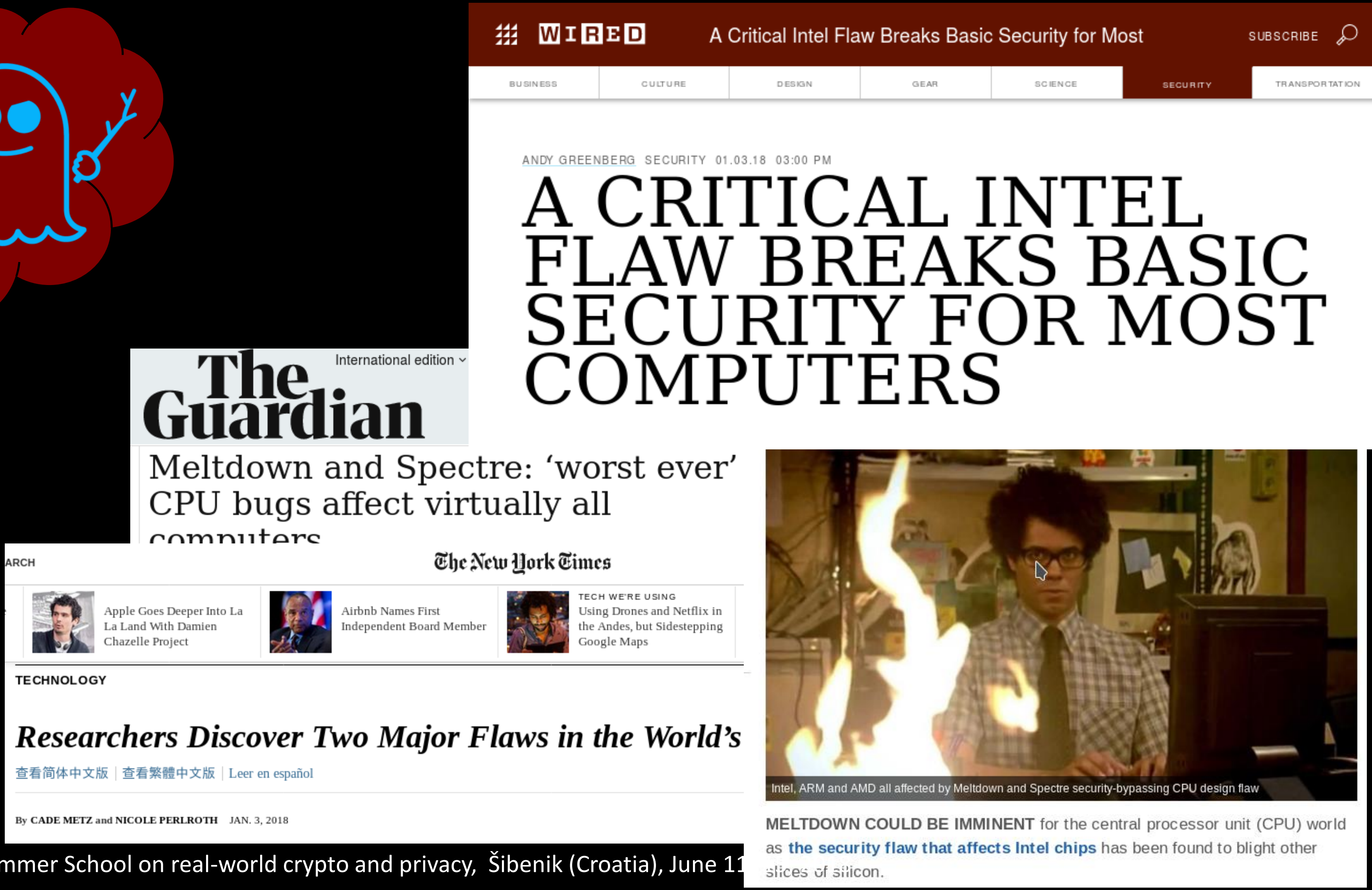
By CADE METZ and NICOLE PERLROTH JAN. 3, 2018

Intel, ARM and AMD all affected by Meltdown and Spectre security-bypassing CPU design flaw

MELTDOWN COULD BE IMMINENT for the central processor unit (CPU) world as **the security flaw that affects Intel chips** has been found to blight other slices of silicon.

So, you might have noticed…





A Critical Intel Flaw Breaks Basic Security for Most

ANDY GREENBERG SECURITY 01.03.18 03:00 PM

A CRITICAL INTEL FLAW BREAKS BASIC SECURITY FOR MOST COMPUTERS

The Guardian

International edition

Meltdown and Spectre: ‘worst ever’ CPU bugs affect virtually all computers

ARCH

The New York Times

Apple Goes Deeper Into La La Land With Damien Chazelle Project

Airbnb Names First Independent Board Member

TECH WE'RE USING
Using Drones and Netflix in the Andes, but Sidestepping Google Maps

TECHNOLOGY

Researchers Discover Two Major Flaws in the World’s

查看简体中文版 | 查看繁體中文版 | Leer en español

By CADE METZ and NICOLE PERLROTH JAN. 3, 2018

Intel, ARM and AMD all affected by Meltdown and Spectre security-bypassing CPU design flaw

MELTDOWN COULD BE IMMINENT for the central processor unit (CPU) world as the security flaw that affects Intel chips has been found to blight other slices of silicon.

Three Attacks

- CVE-2017-5754 (aka. *Meltdown*)
  - Exploits rogue data-cache loads **during speculative execution**
- CVE-2017-5753 (aka. *Spectre*)
  - Exploits bounds-check bypasses **during speculative execution**
- CVE-2017-5715 (aka. *Spectre*)
  - Exploits branch-target injection **during speculative execution**

~~Intel Inside~~ Bug inside *Speculative Execution!*

Speculative Execution? Sounds fishy..

And what is a processor anyways?

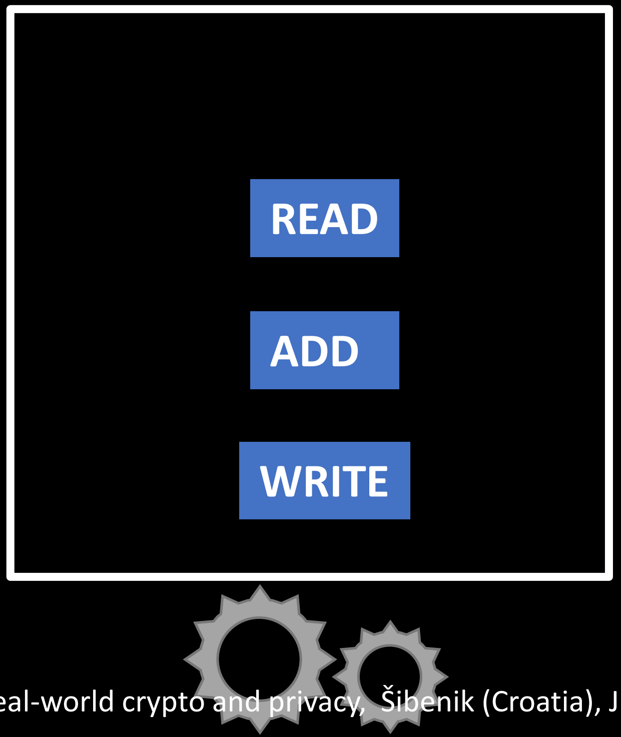
Input:

Data:
17
42

Code:

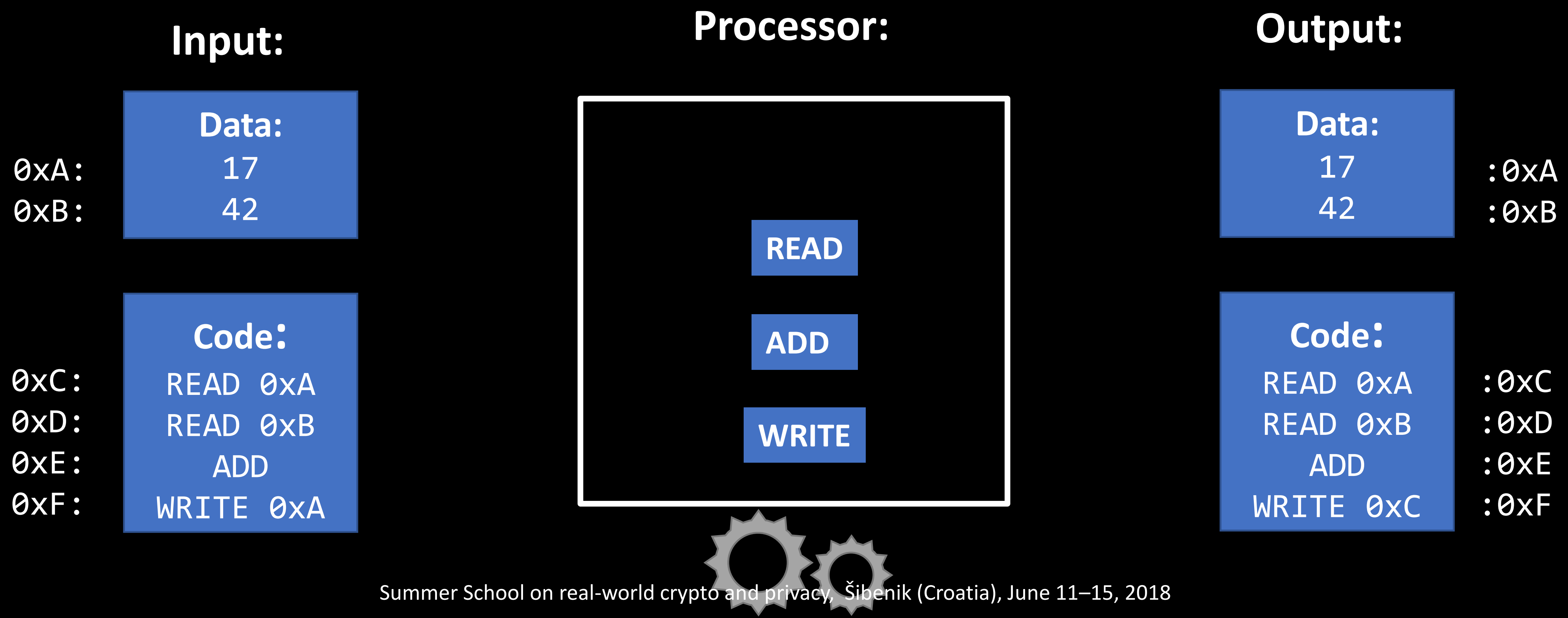
```
READ 0xA
READ 0xB
ADD
WRITE 0xA
```

Processor:



Speculative Execution? Sounds fishy..

And what is a processor anyways?



Input:

Data:

```
0xA: 17
0xB: 42
```

Code:

```
0xC: READ 0xA
0xD: READ 0xB
0xE: ADD
0xF: WRITE 0xA
```

Processor:

READ

ADD

WRITE

Output:

Data:

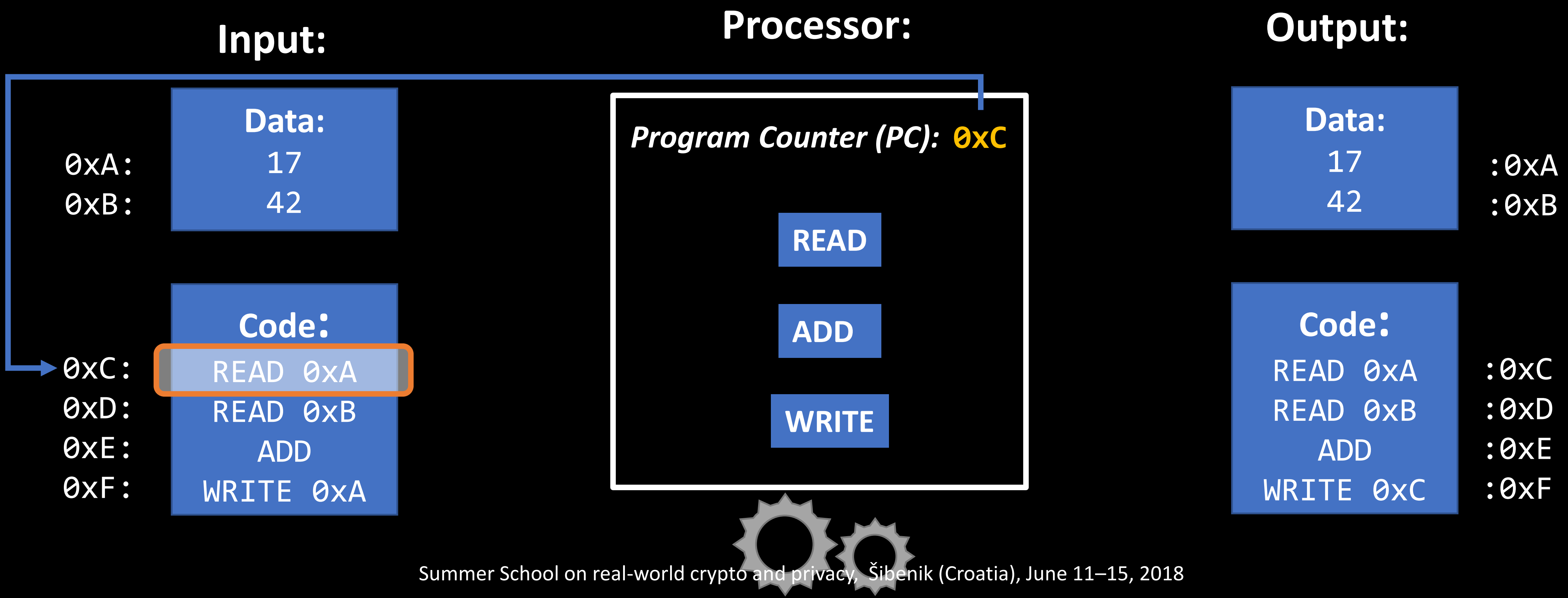
```
17 :0xA
42 :0xB
```

Code:

```
READ 0xA  :0xC
READ 0xB  :0xD
ADD       :0xE
WRITE 0xC :0xF
```

Speculative Execution? Sounds fishy..

And what is a processor anyways?



Input:

Data:

0xA: 17
0xB: 42

Code:

0xC: READ 0xA
0xD: READ 0xB
0xE: ADD
0xF: WRITE 0xA

Processor:

Program Counter (PC): 0xC

READ

ADD

WRITE

Output:

Data:

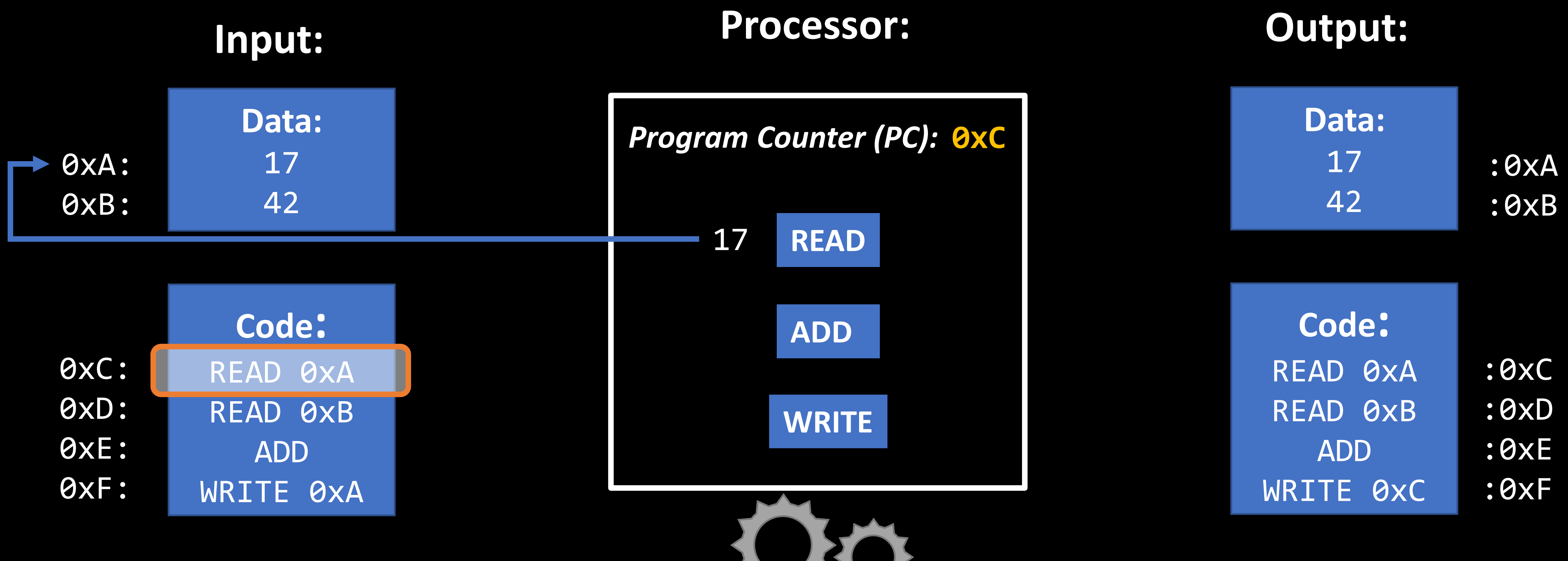
17 :0xA
42 :0xB

Code:

READ 0xA :0xC
READ 0xB :0xD
ADD :0xE
WRITE 0xC :0xF

Speculative Execution? Sounds fishy..

And what is a processor anyways?



Input:

Data:

```
0xA: 17
0xB: 42
```

Code:

```
0xC: READ 0xA
0xD: READ 0xB
0xE: ADD
0xF: WRITE 0xA
```

Processor:

Program Counter (PC): `0xC`

17 **READ**

ADD

WRITE

Output:

Data:

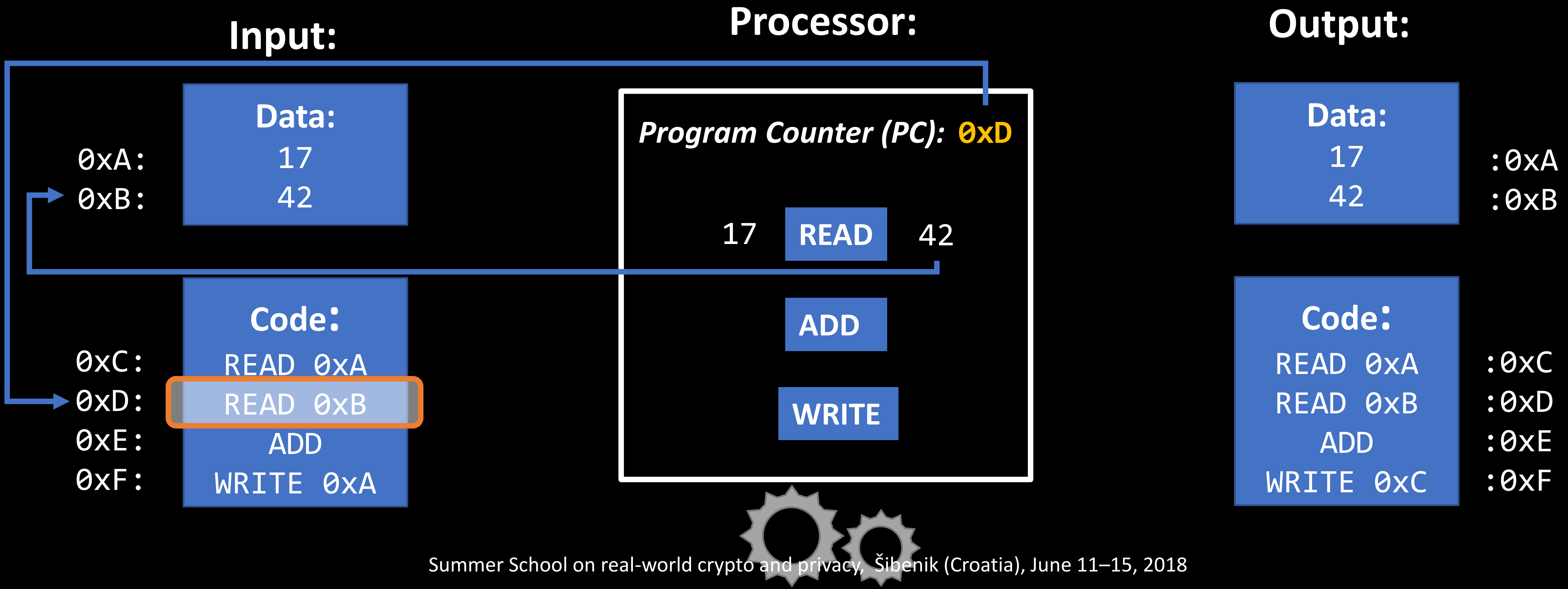
```
17  :0xA
42  :0xB
```

Code:

```
READ 0xA   :0xC
READ 0xB   :0xD
ADD        :0xE
WRITE 0xC  :0xF
```

Speculative Execution? Sounds fishy..

And what is a processor anyways?



Input:

Data:
0xA: 17
0xB: 42

Code:
0xC: READ 0xA
0xD: READ 0xB
0xE: ADD
0xF: WRITE 0xA

Processor:

Program Counter (PC): 0xD

17 **READ** 42

ADD

WRITE

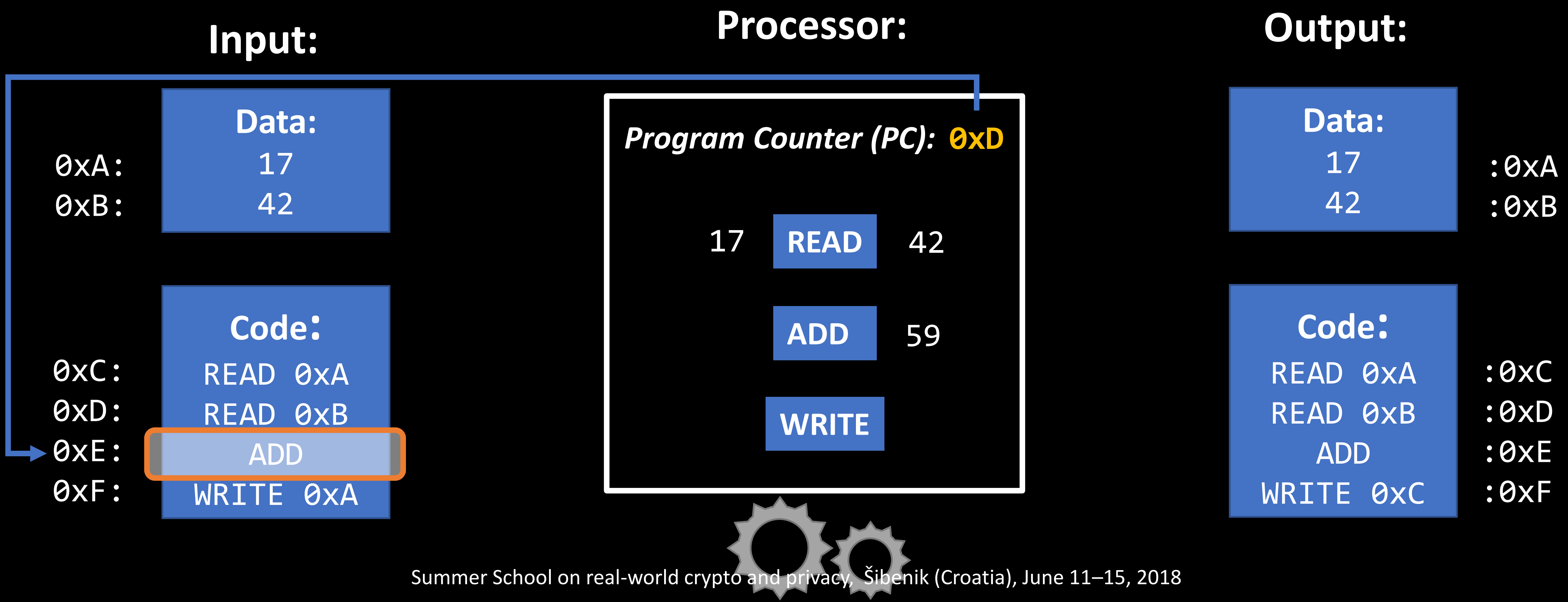
Output:

Data:
17 :0xA
42 :0xB

Code:
READ 0xA :0xC
READ 0xB :0xD
ADD :0xE
WRITE 0xC :0xF

Speculative Execution? Sounds fishy..

And what is a processor anyways?



Input:

Data:

0xA: 17
0xB: 42

Code:

0xC: READ 0xA
0xD: READ 0xB
0xE: ADD
0xF: WRITE 0xA

Processor:

Program Counter (PC): 0xD

17 **READ** 42

ADD 59

WRITE

Output:

Data:

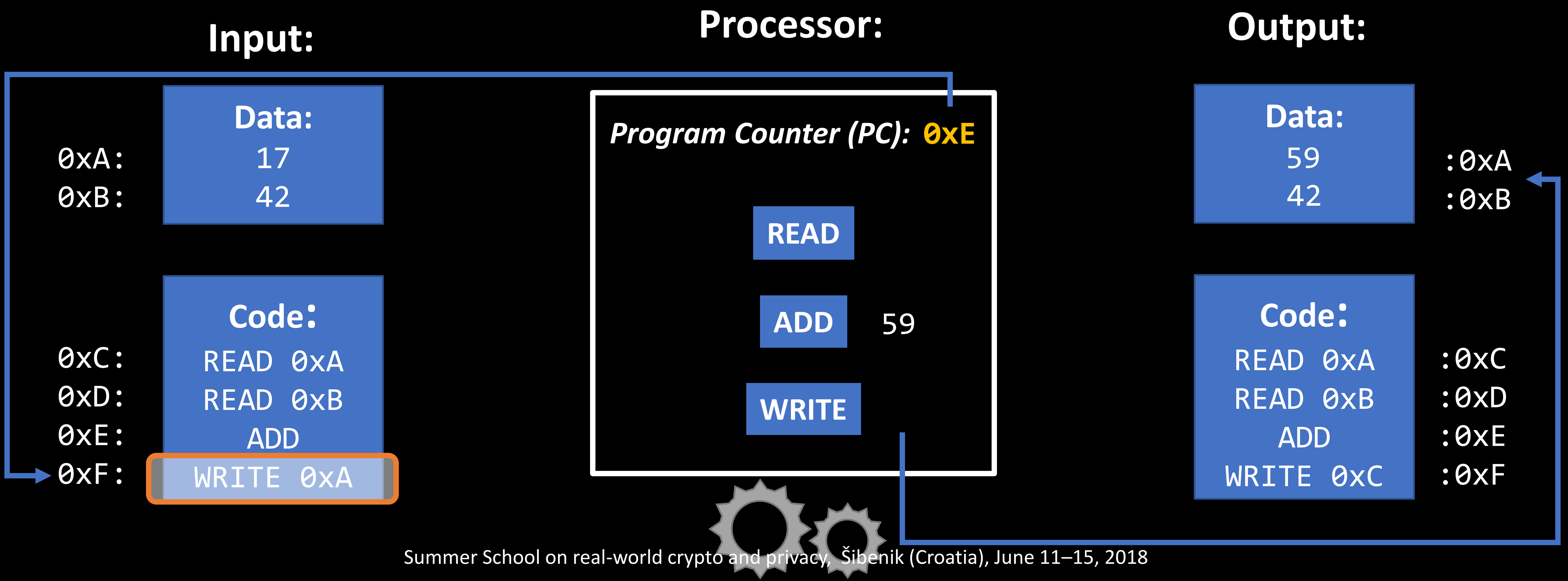
17 :0xA
42 :0xB

Code:

READ 0xA :0xC
READ 0xB :0xD
ADD :0xE
WRITE 0xC :0xF

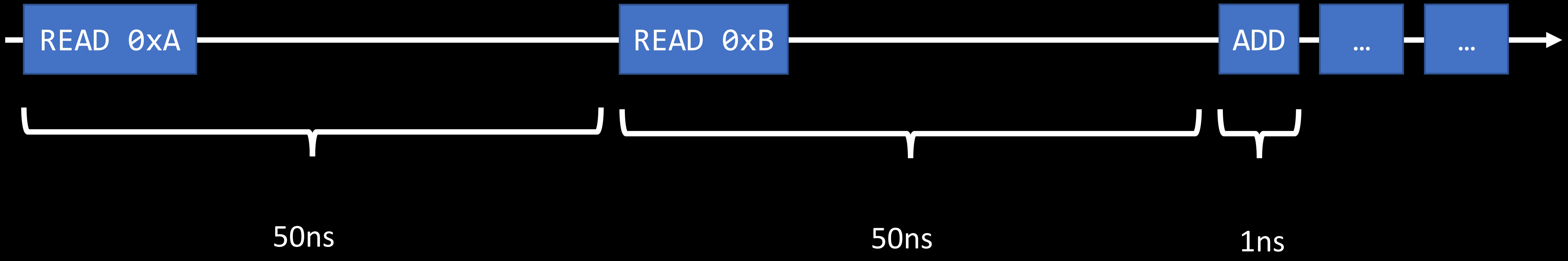
Speculative Execution? Sounds fishy..

And what is a processor anyways?



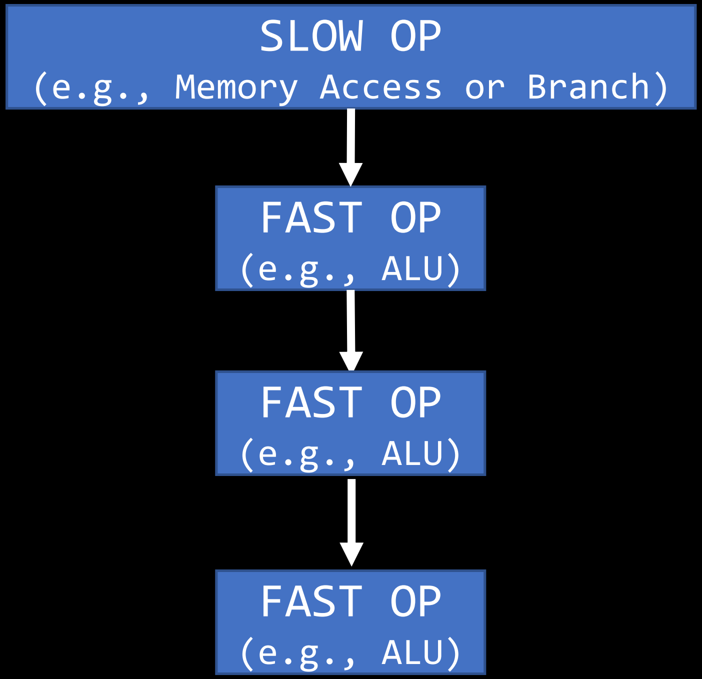
Some operations are SLOOOOOOW

- Two read operations can easily stall the CPU for more than 100ns
- An integer addition takes two orders of magnitude less time (~1ns)
- So, in the time domain the execution looks like this:
- **Processor does *NOTHING* for 100ns!**



Optimizing for Performance..

Instruction Stream:



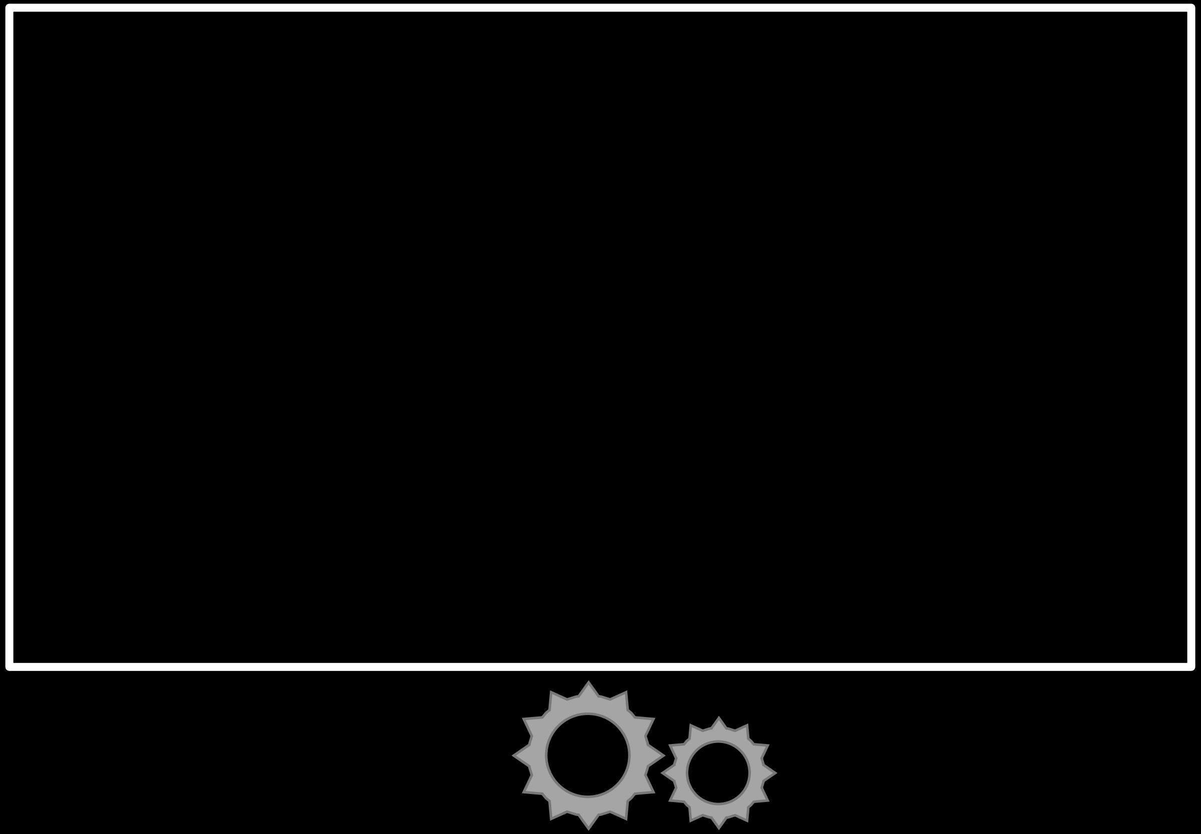
SLOW OP
(e.g., Memory Access or Branch)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

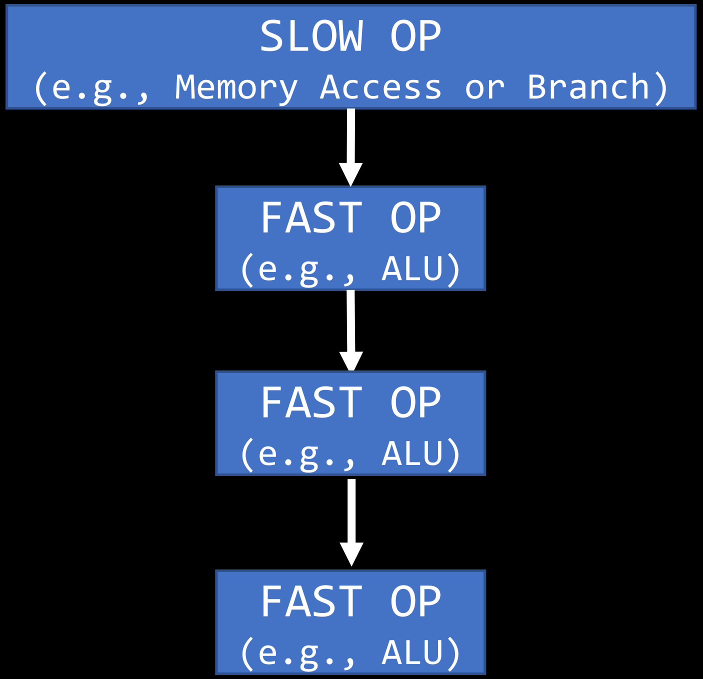
FAST OP
(e.g., ALU)

Out-of-Order Execution:

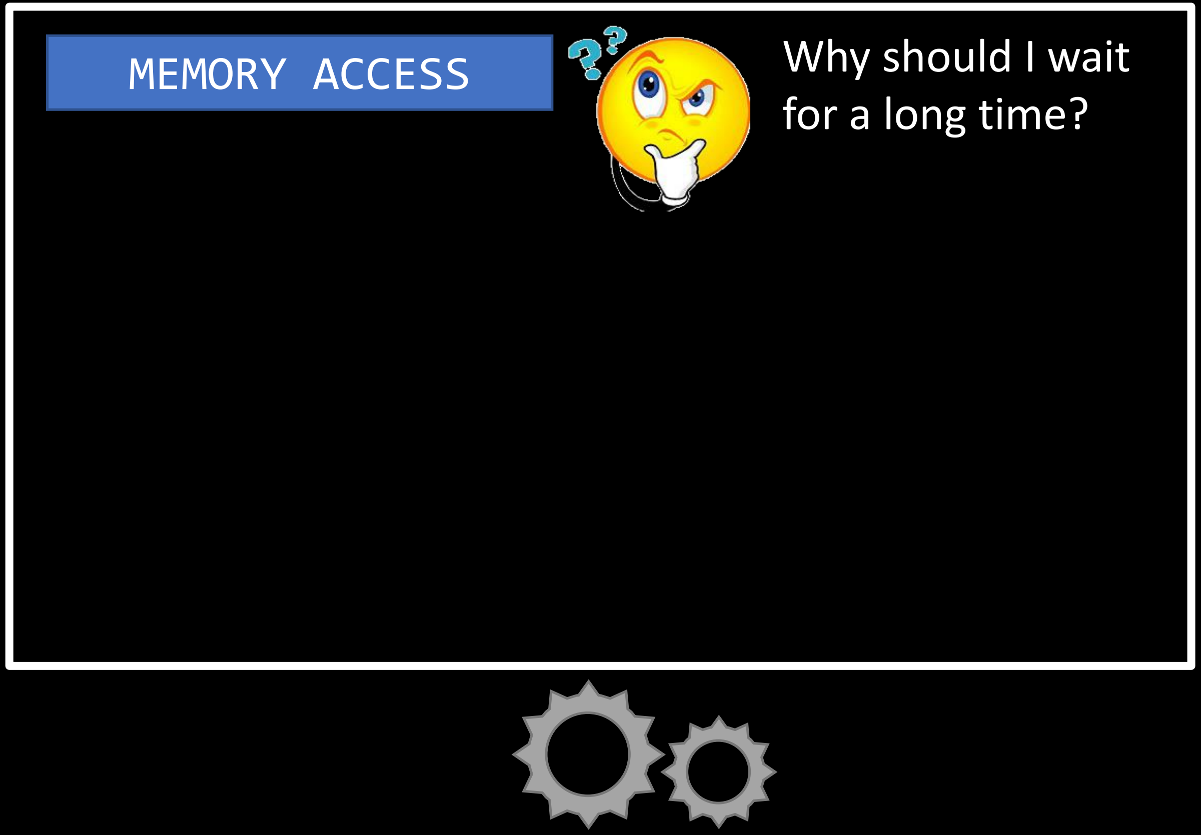


Optimizing for Performance..

Instruction Stream:



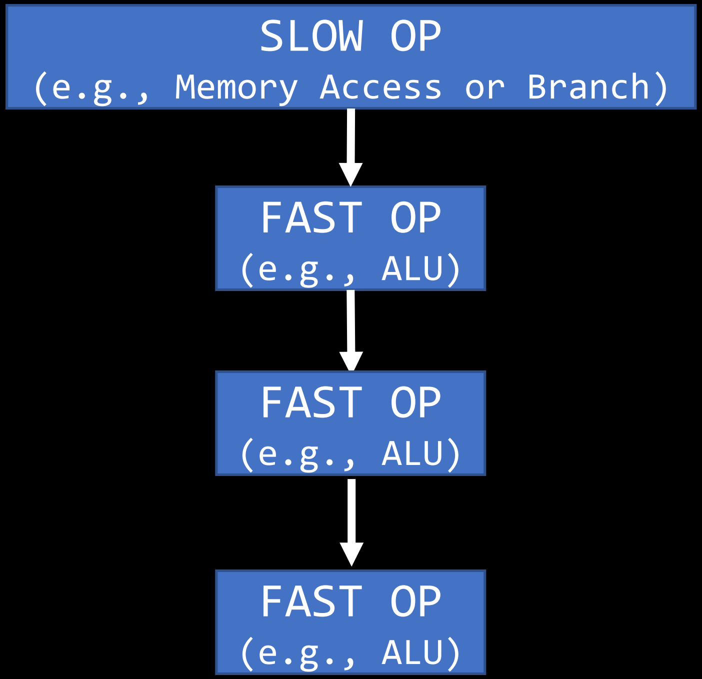
Out-of-Order Execution:



Why should I wait for a long time?

Optimizing for Performance..

Instruction Stream:



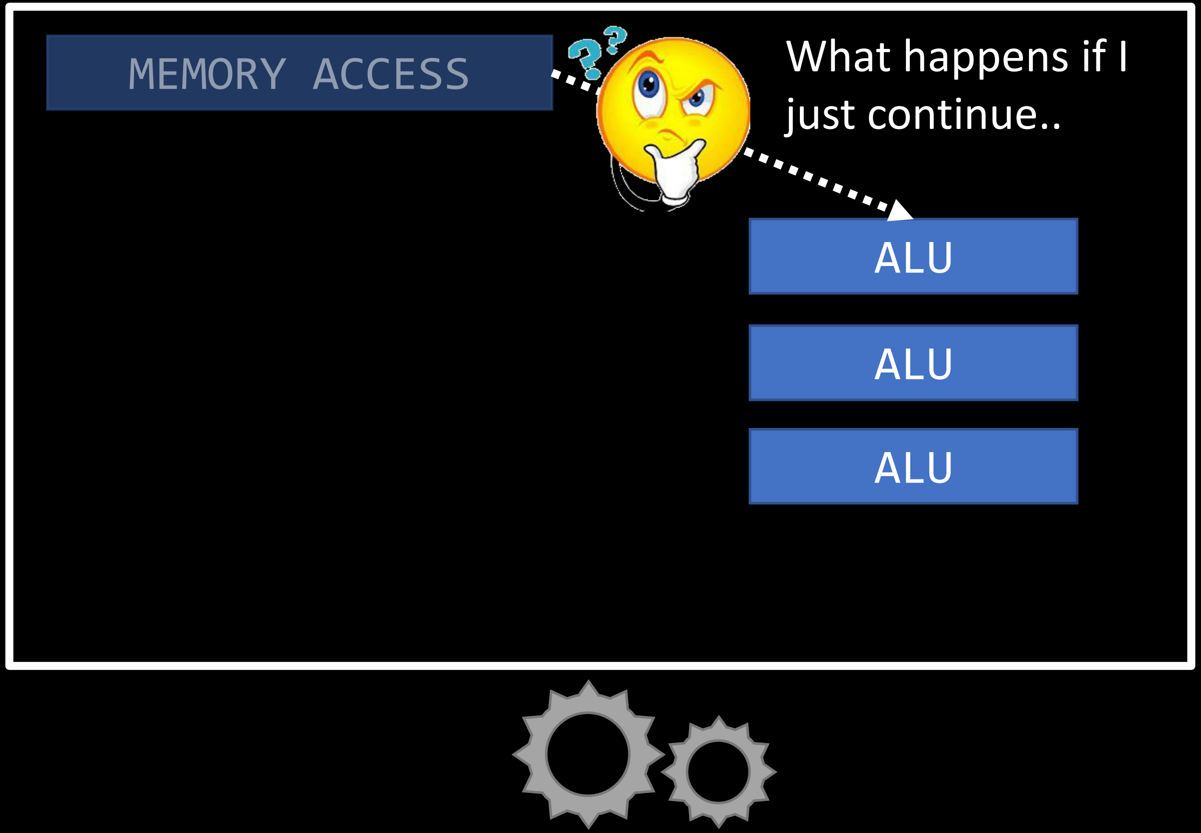
SLOW OP
(e.g., Memory Access or Branch)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

Out-of-Order Execution:



MEMORY ACCESS

What happens if I just continue..

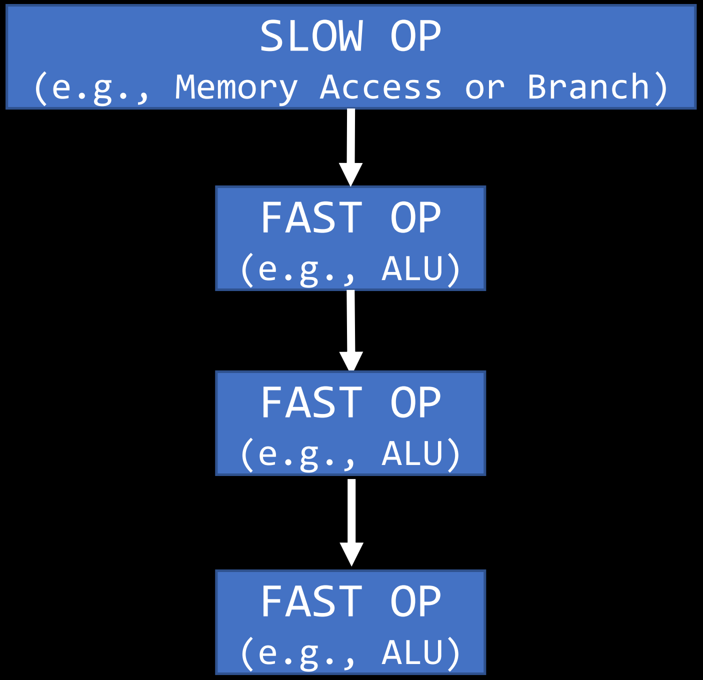
ALU

ALU

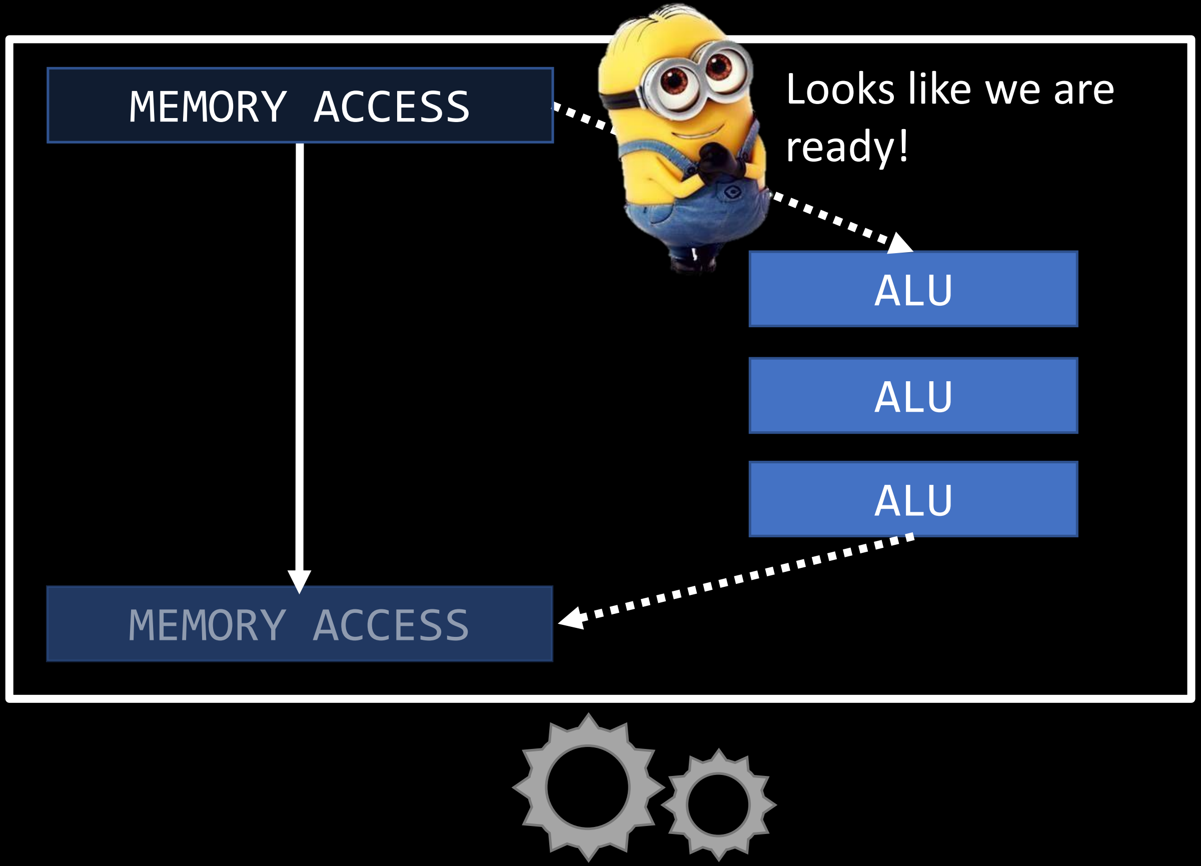
ALU

Optimizing for Performance..

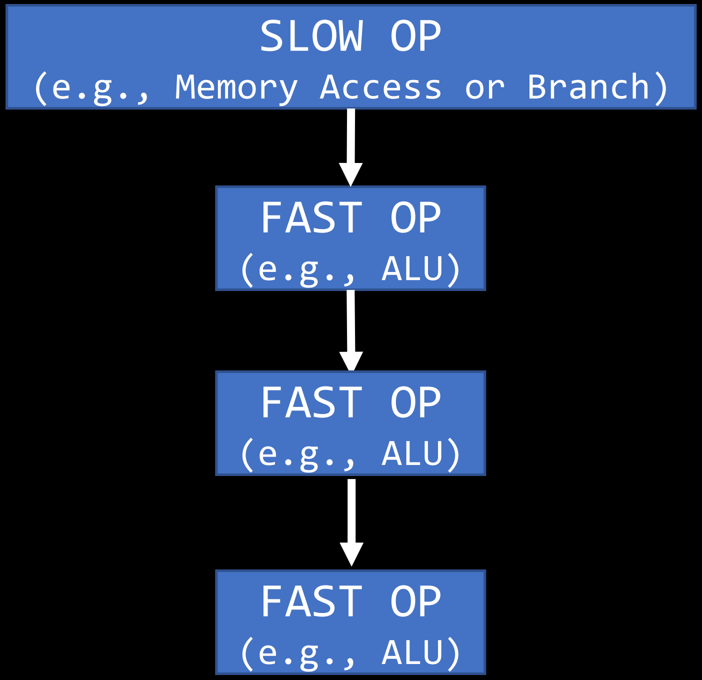
Instruction Stream:

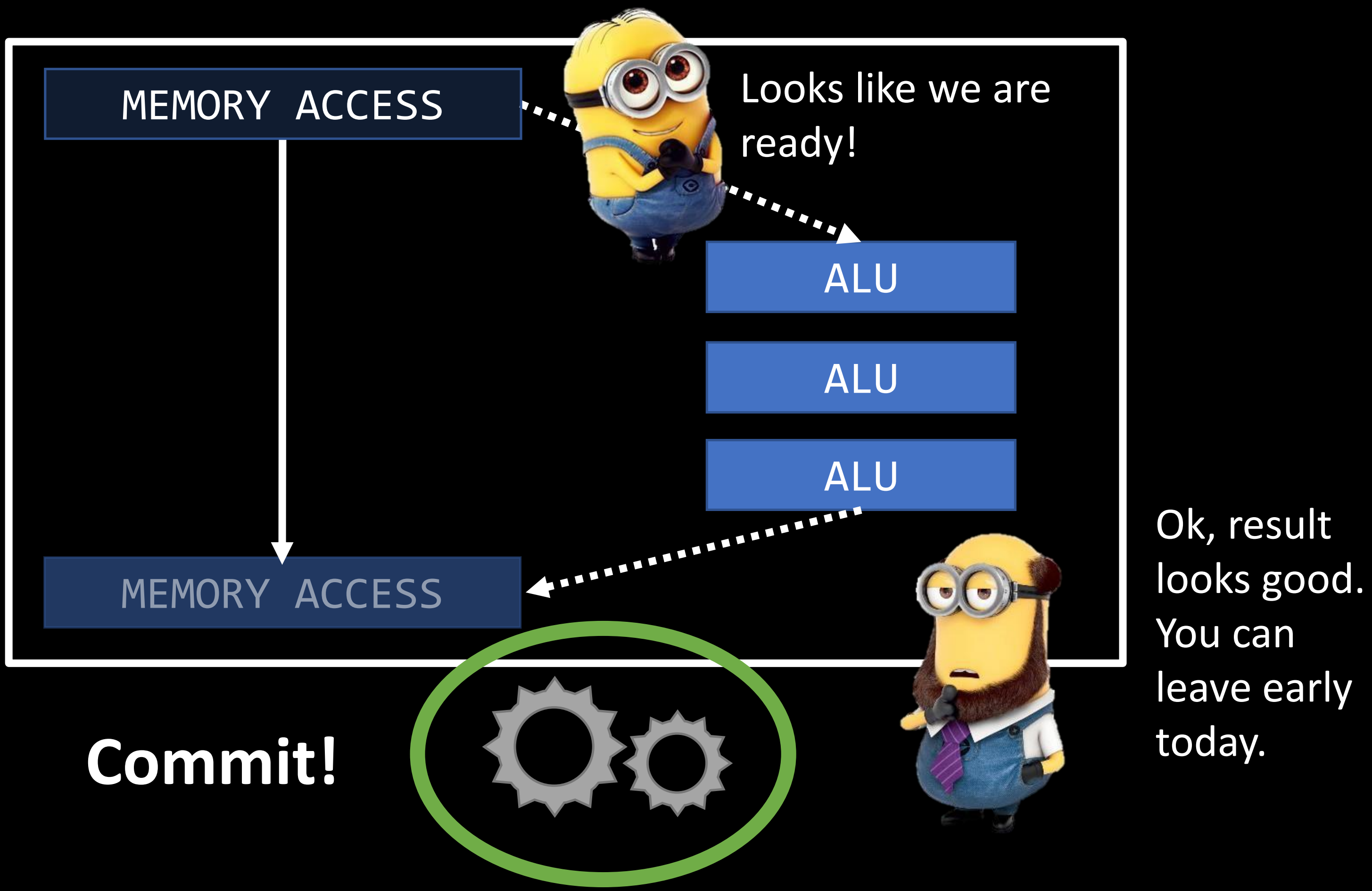


Out-of-Order Execution:



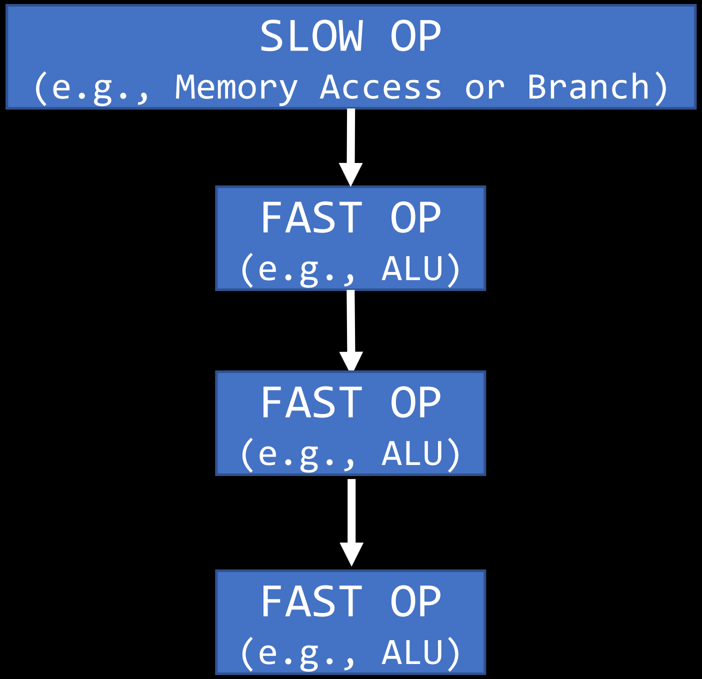
Optimizing for Performance..



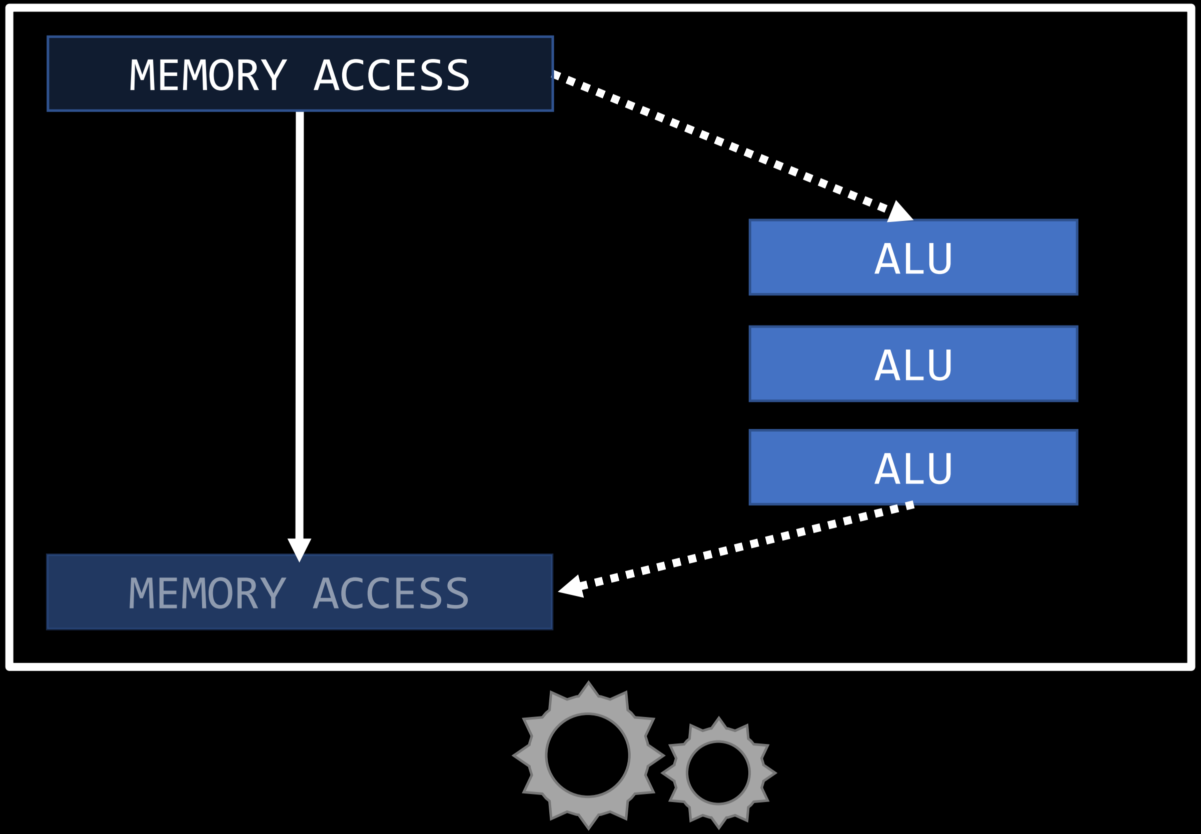


Optimizing for Performance..

Instruction Stream:



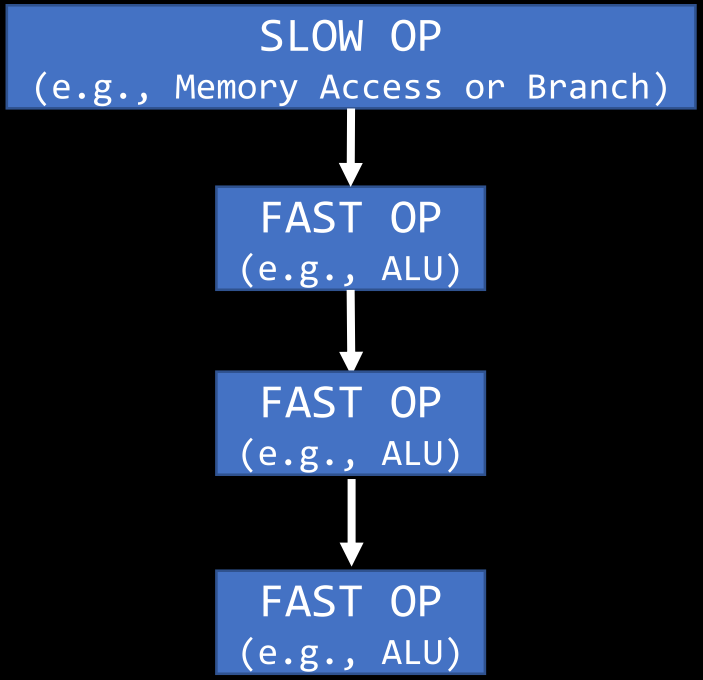
Out-of-Order Execution:



To Boost Performance Modern Processors Execute Instructions *Out-of-Order*!

..what if it does not work?

Instruction Stream:



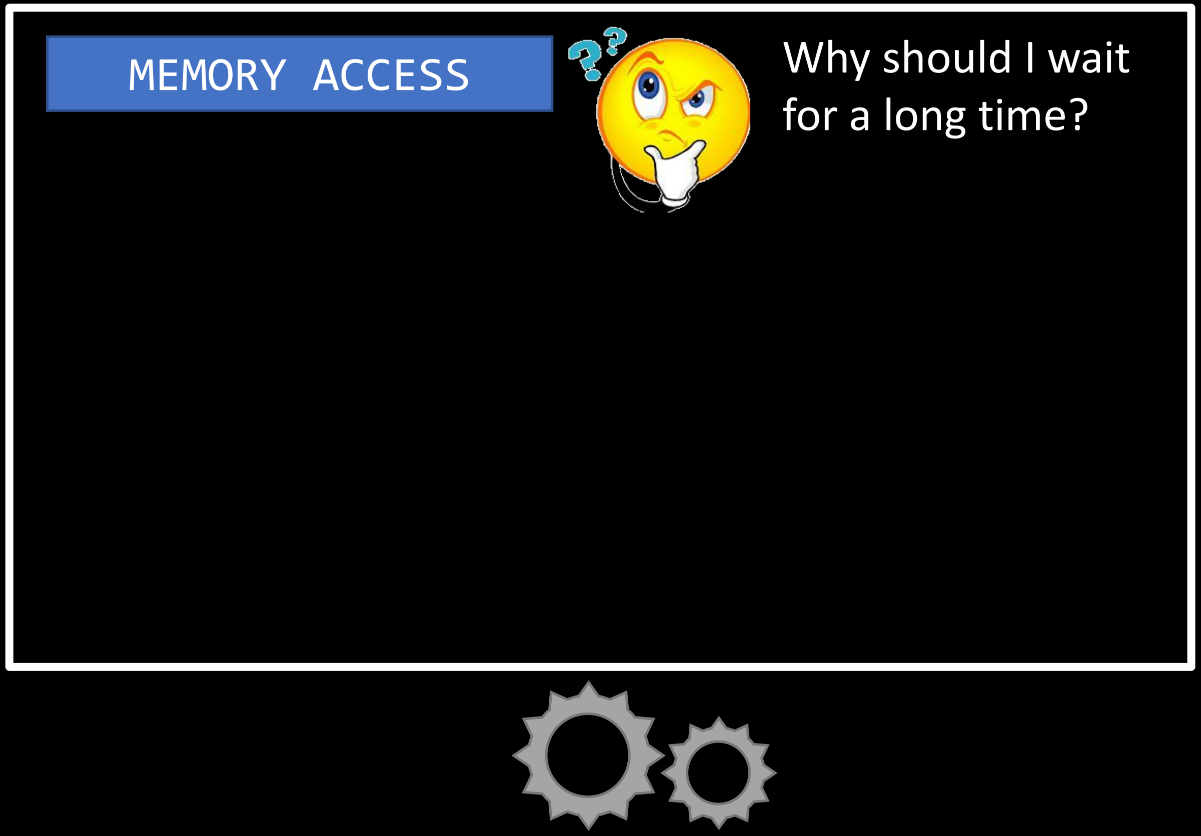
SLOW OP
(e.g., Memory Access or Branch)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

Out-of-Order Execution:

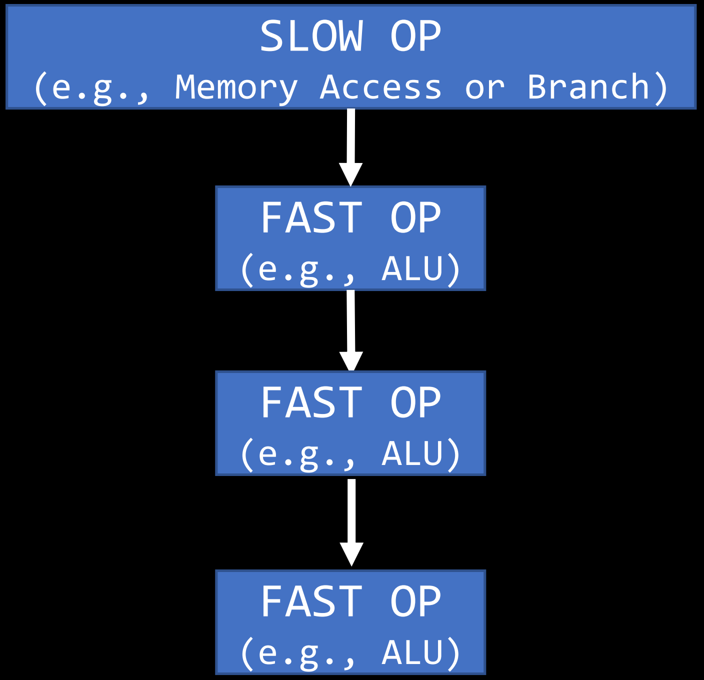


MEMORY ACCESS

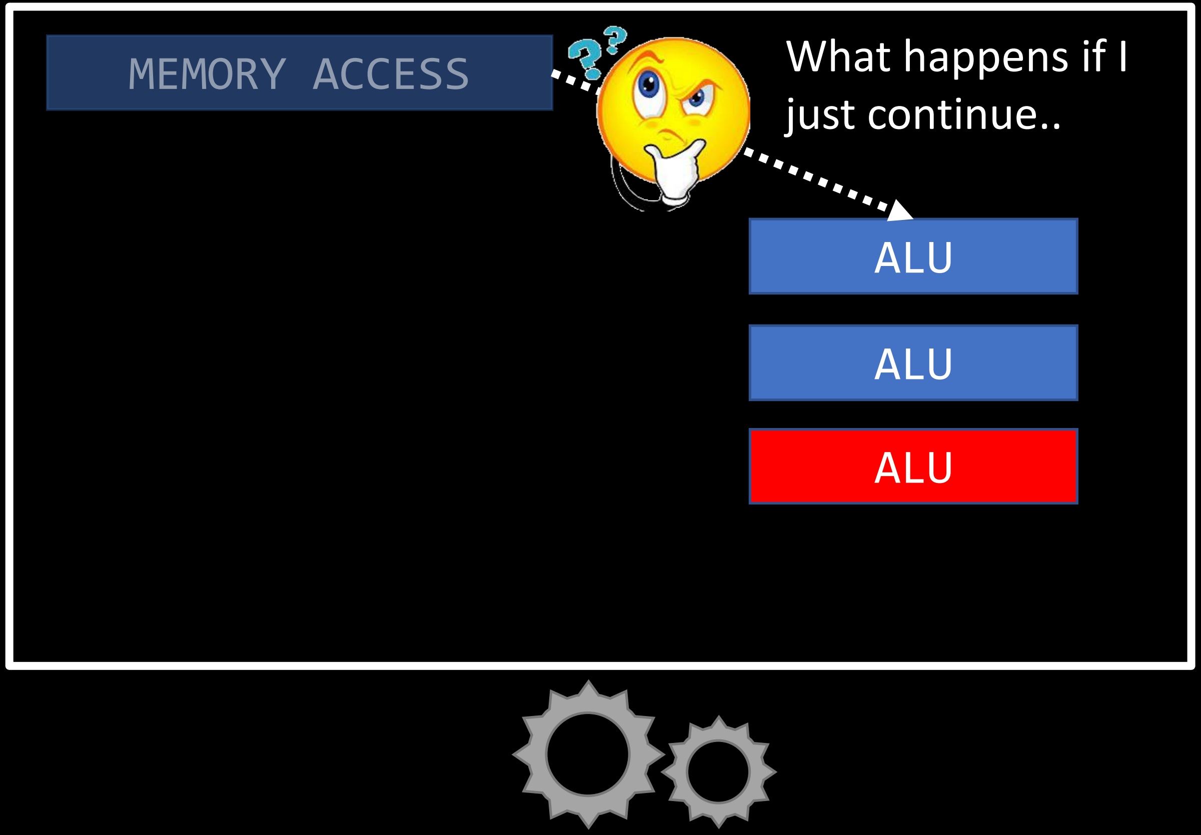
Why should I wait for a long time?

..what if it does not work?

Instruction Stream:

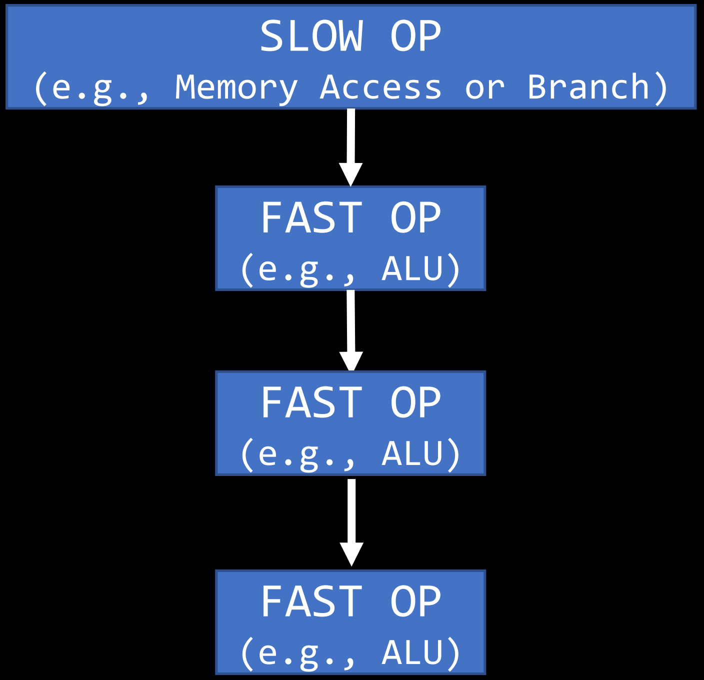


Out-of-Order Execution:

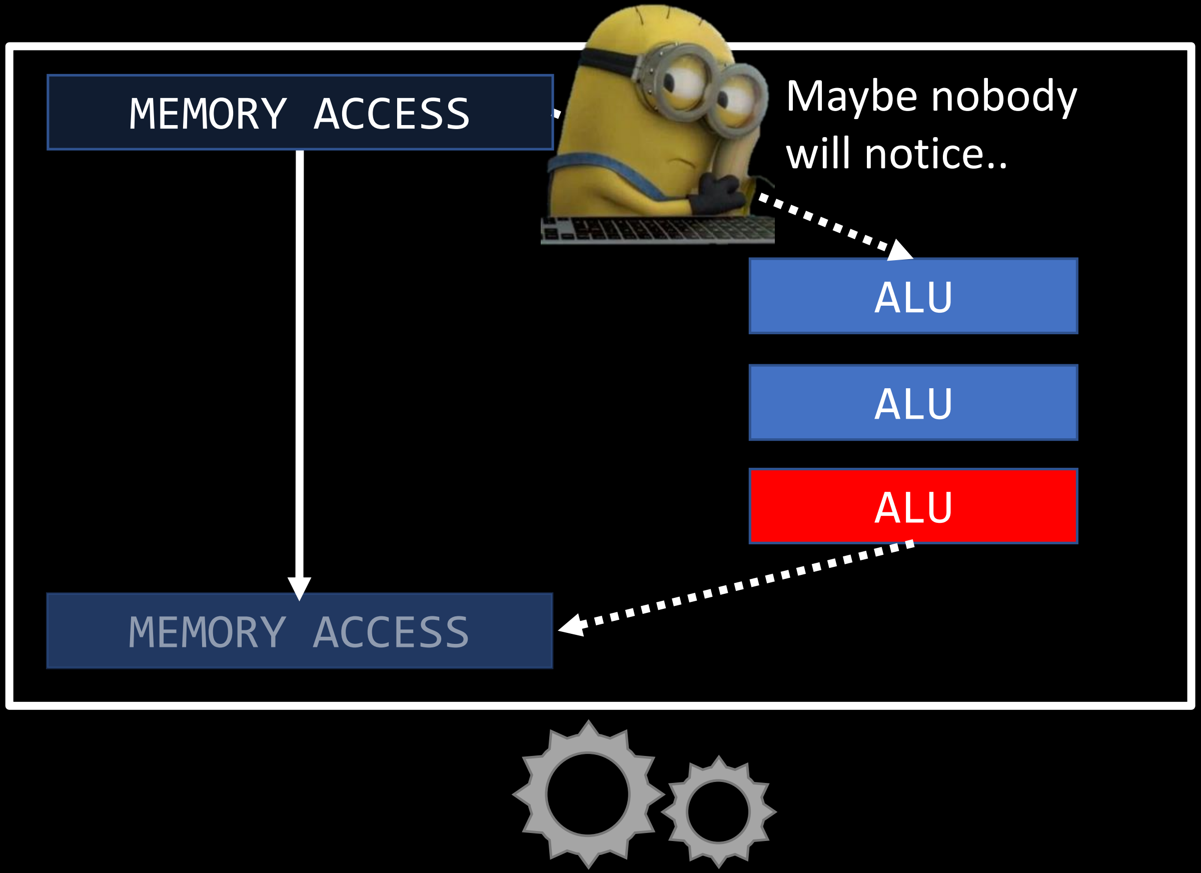


..what if it does not work?

Instruction Stream:

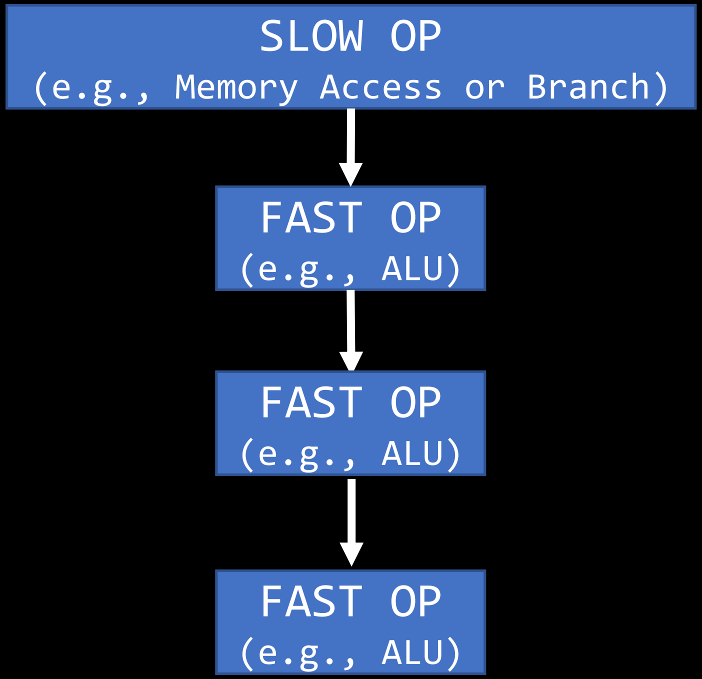


Out-of-Order Execution:



..what if it does not work?

Instruction Stream:



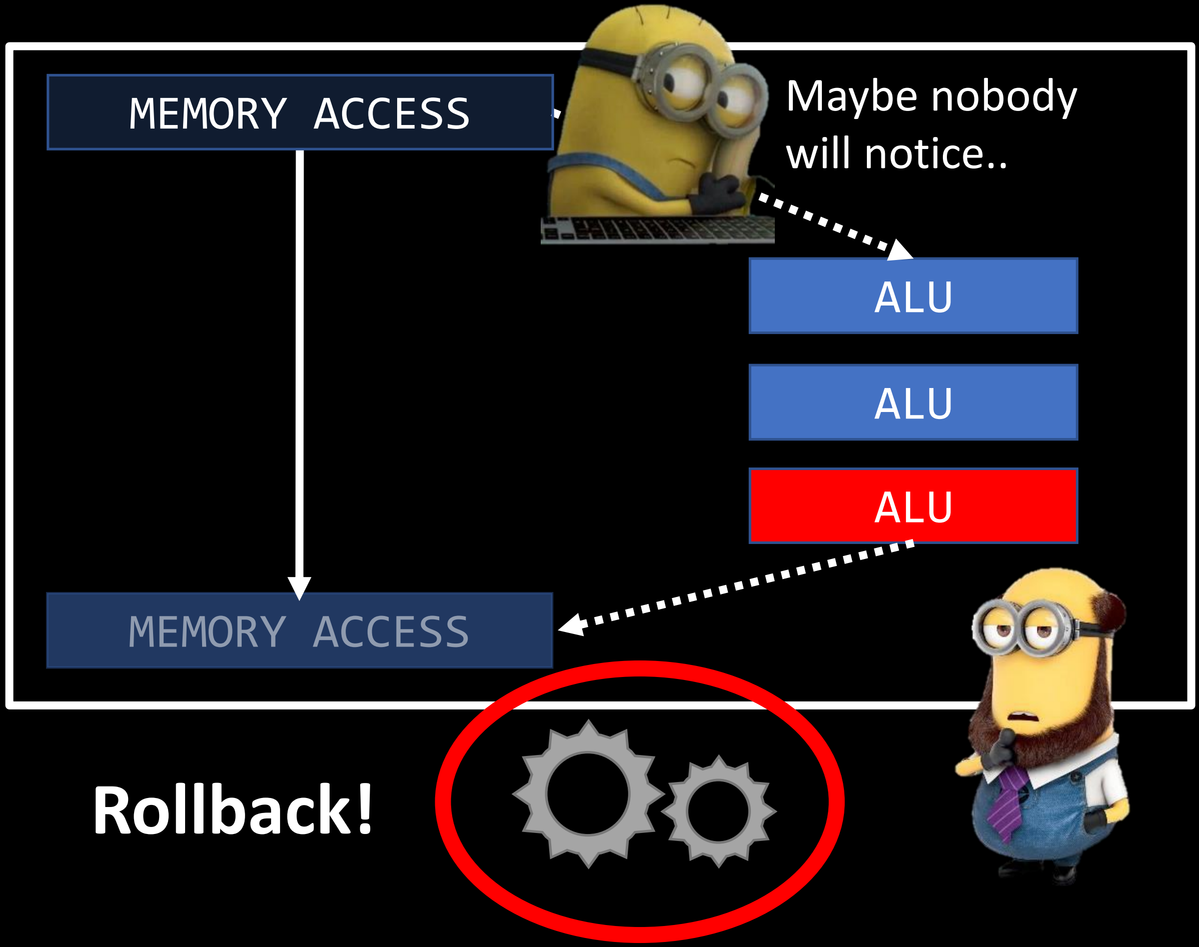
SLOW OP
(e.g., Memory Access or Branch)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

Out-of-Order Execution:



MEMORY ACCESS

Maybe nobody will notice..

ALU

ALU

ALU

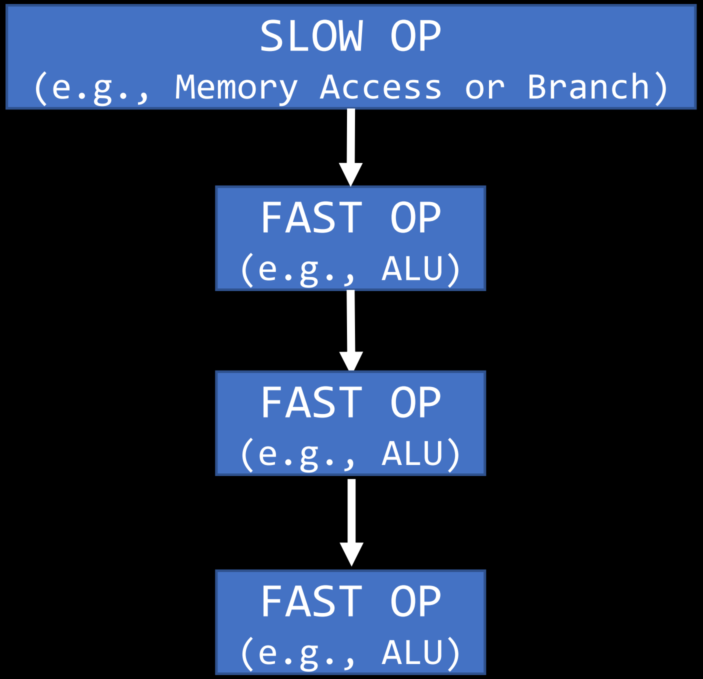
MEMORY ACCESS

Rollback!

Do it in order, stupid!

..what if it does not work?

Instruction Stream:



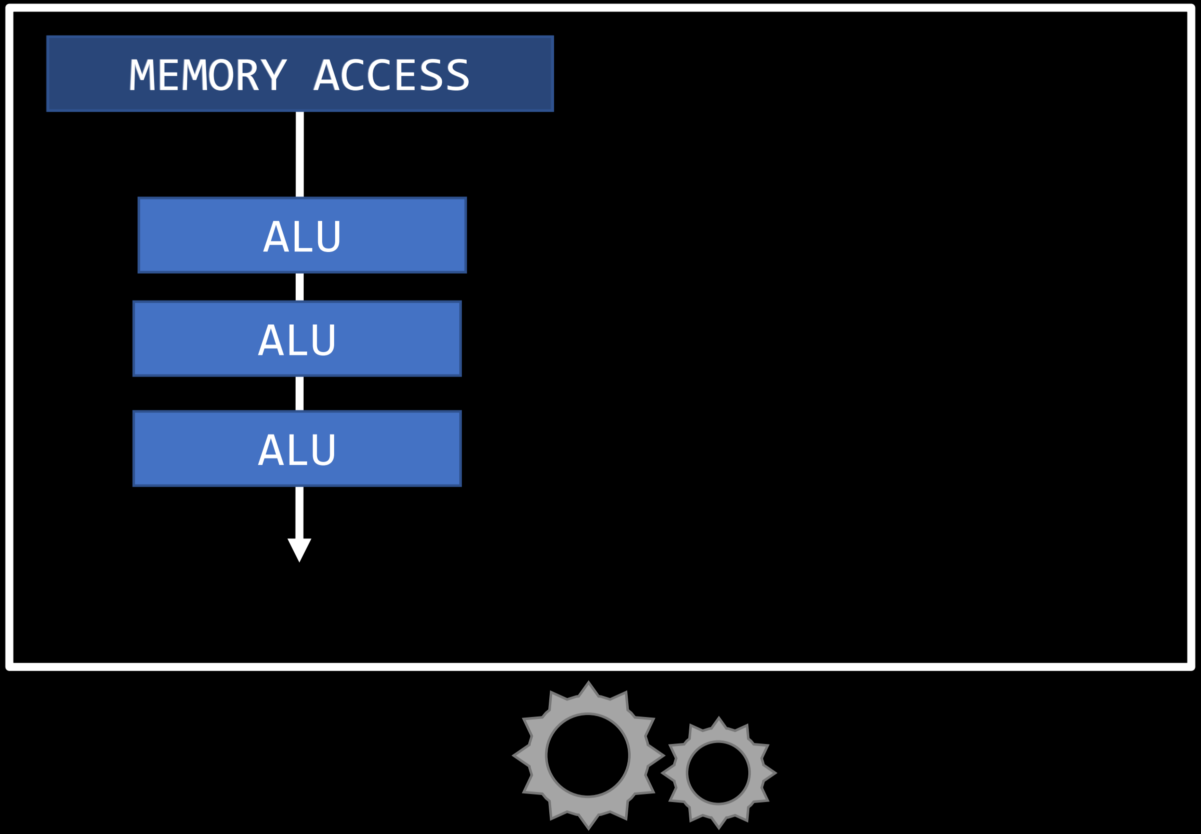
SLOW OP
(e.g., Memory Access or Branch)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

FAST OP
(e.g., ALU)

In Order Execution:



MEMORY ACCESS

ALU

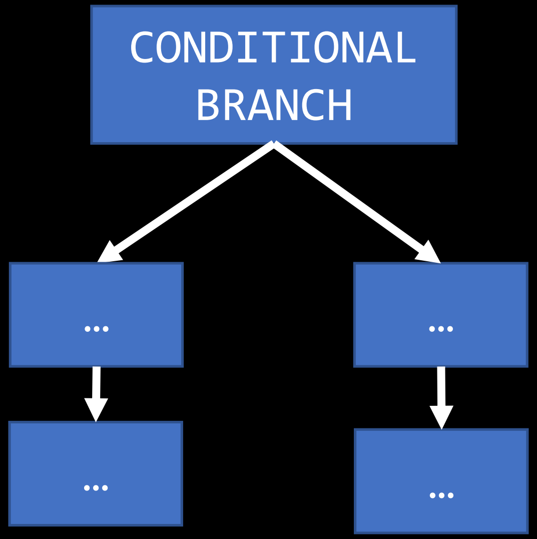
ALU

ALU

Only *correct* optimizations are commited!

Out-of-Order vs. *Speculative* Execution

- If the instruction that is re-ordered is a **branching instruction**, the resulting Out-of-Order stream is called *Speculative Execution*



- Many processors **do not** optimize this
- Bigger processors invest a lot of work into optimizing branches!
- Simple optimization:
  - Always execute both branches
  - *only commit the correct one*

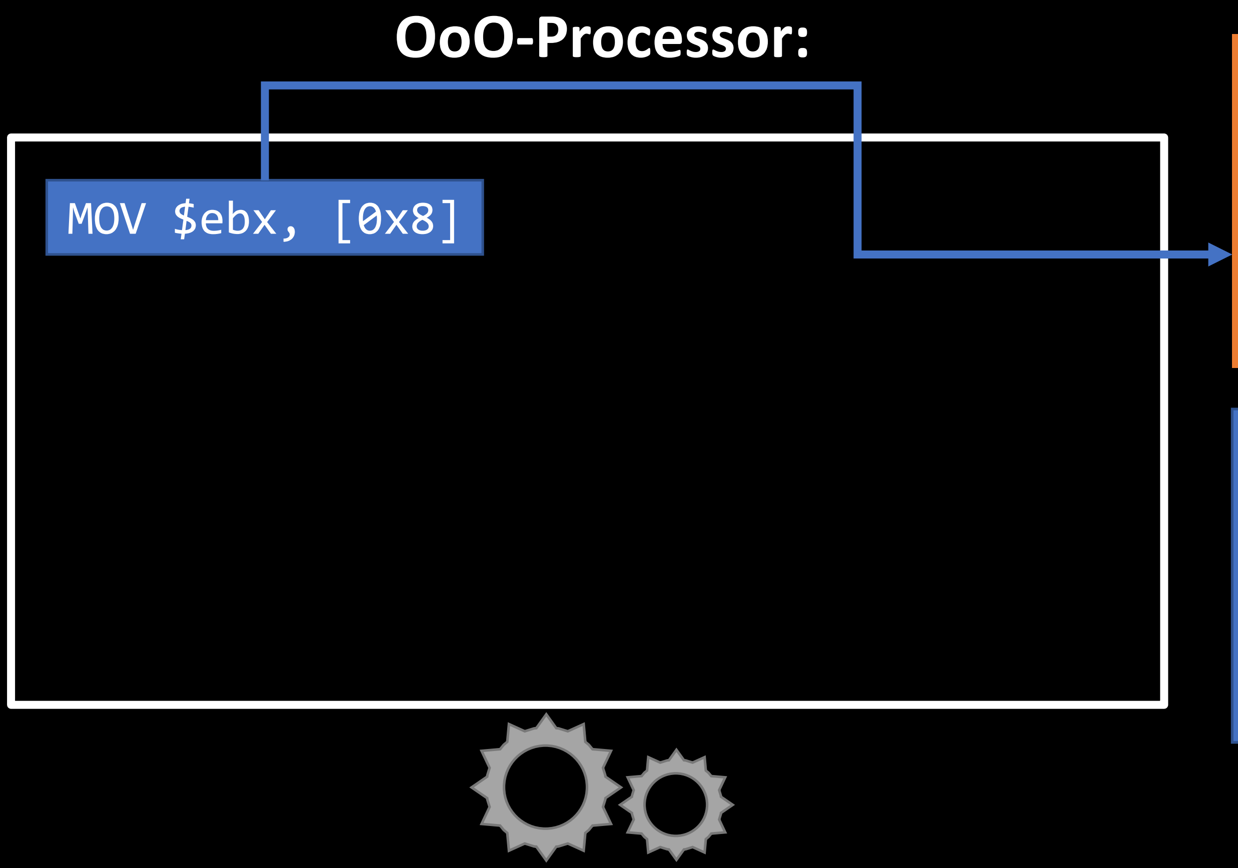
What could possibly go wrong?

Cache:

0x70:	00	00	00	00
0x74:	00	00	00	00
0x78:	00	00	00	00
0x7C:	00	00	00	00

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```



OoO-Processor:

```
MOV $ebx, [0x8]
```

OS Memory:

0A	0B	0C	0D	:0x0
0E	0F	10	1A	:0x4
00	00	00	0C	:0x8
1F	20	2A	2B	:0xC

User Memory:

0A	0B	0C	0D	:0x70
0E	0F	10	1A	:0x74
1B	1C	1D	1E	:0x78
1F	20	2A	2B	:0x7C

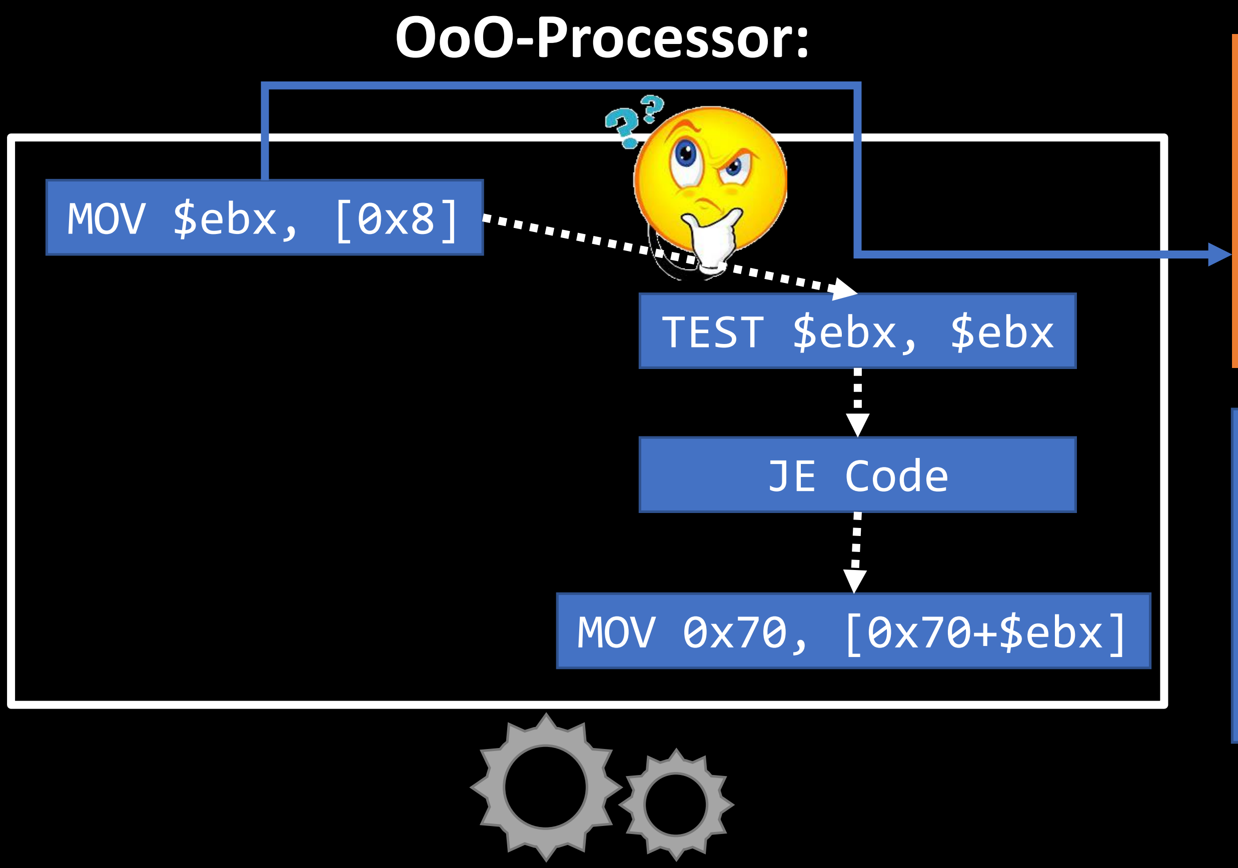
What could possibly go wrong?

Cache:

0x70:	00	00	00	00
0x74:	00	00	00	00
0x78:	00	00	00	00
0x7C:	00	00	00	00

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```



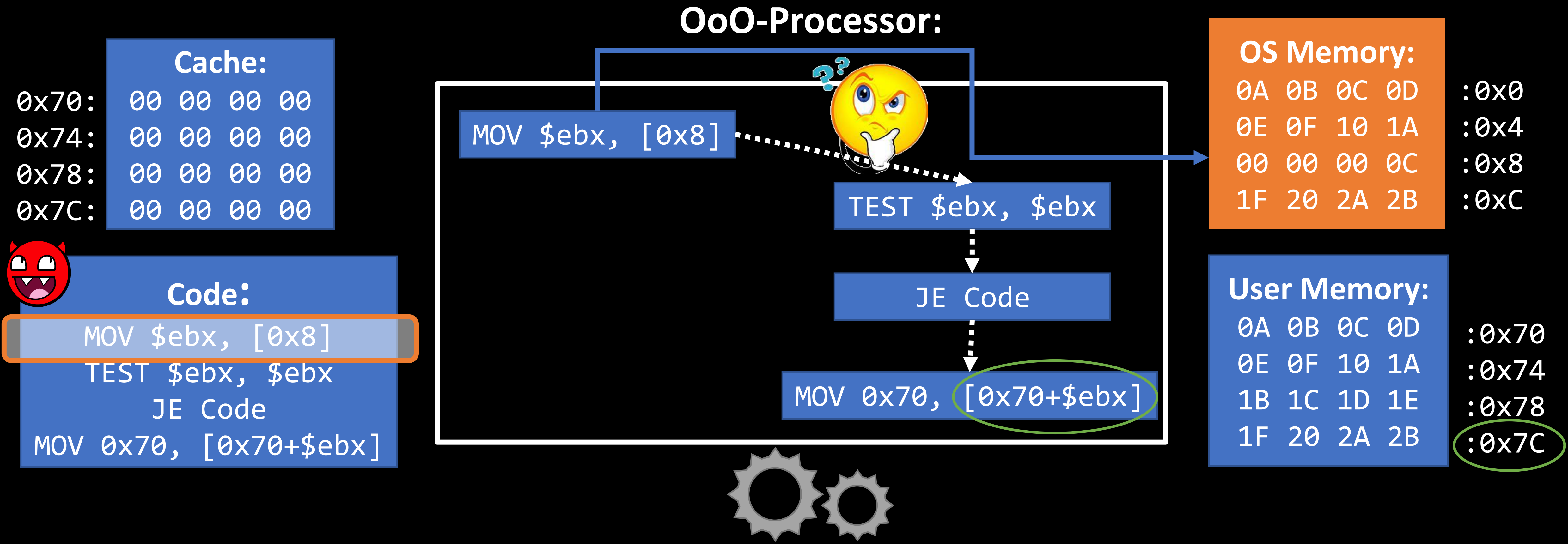
OS Memory:

0A	0B	0C	0D	:0x0
0E	0F	10	1A	:0x4
00	00	00	0C	:0x8
1F	20	2A	2B	:0xC

User Memory:

0A	0B	0C	0D	:0x70
0E	0F	10	1A	:0x74
1B	1C	1D	1E	:0x78
1F	20	2A	2B	:0x7C

What could possibly go wrong?



Cache:

0x70:	00	00	00	00
0x74:	00	00	00	00
0x78:	00	00	00	00
0x7C:	00	00	00	00

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OoO-Processor:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

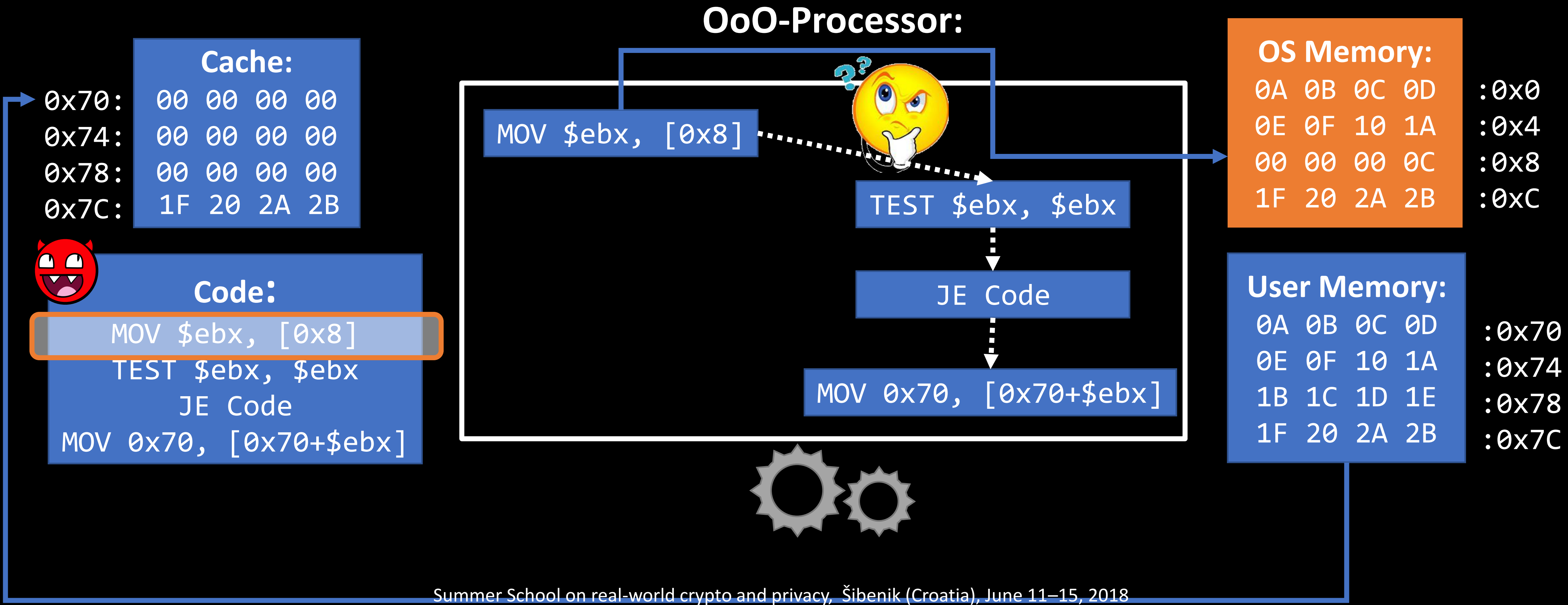
OS Memory:

0A	0B	0C	0D	:0x0
0E	0F	10	1A	:0x4
00	00	00	0C	:0x8
1F	20	2A	2B	:0xC

User Memory:

0A	0B	0C	0D	:0x70
0E	0F	10	1A	:0x74
1B	1C	1D	1E	:0x78
1F	20	2A	2B	:0x7C

What could possibly go wrong?



Cache:

```
0x70:  00 00 00 00
0x74:  00 00 00 00
0x78:  00 00 00 00
0x7C:  1F 20 2A 2B
```

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OoO-Processor:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

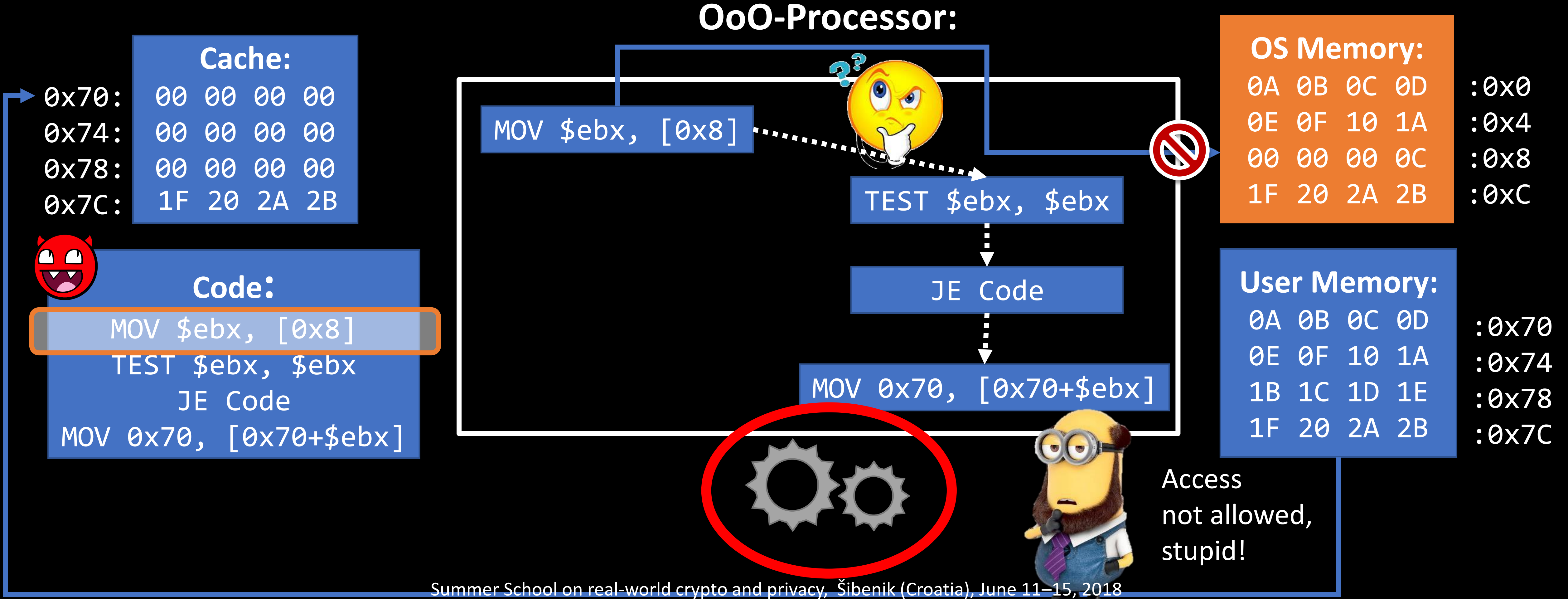
OS Memory:

```
0A 0B 0C 0D  :0x0
0E 0F 10 1A  :0x4
00 00 00 0C  :0x8
1F 20 2A 2B  :0xC
```

User Memory:

```
0A 0B 0C 0D  :0x70
0E 0F 10 1A  :0x74
1B 1C 1D 1E  :0x78
1F 20 2A 2B  :0x7C
```

What could possibly go wrong?



OoO-Processor:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

Cache:

0x70:	00	00	00	00
0x74:	00	00	00	00
0x78:	00	00	00	00
0x7C:	1F	20	2A	2B

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OS Memory:

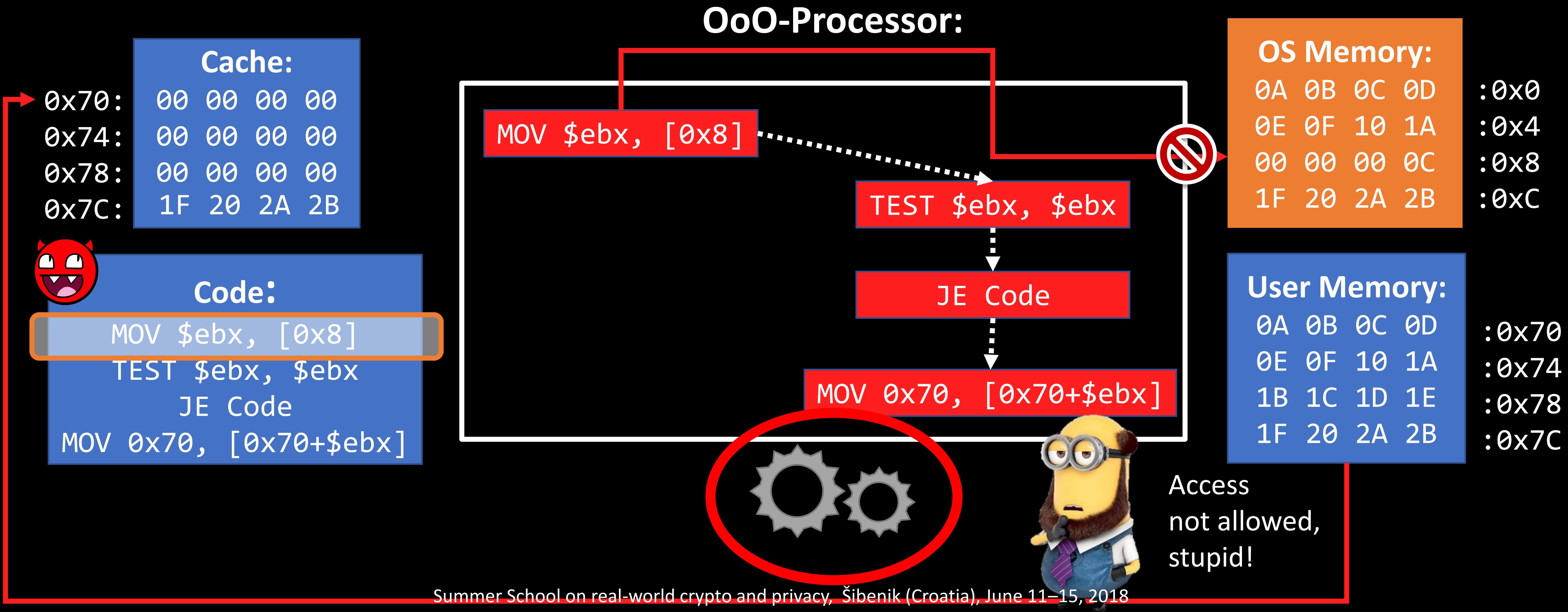
0A	0B	0C	0D	:0x0
0E	0F	10	1A	:0x4
00	00	00	0C	:0x8
1F	20	2A	2B	:0xC

User Memory:

0A	0B	0C	0D	:0x70
0E	0F	10	1A	:0x74
1B	1C	1D	1E	:0x78
1F	20	2A	2B	:0x7C

Access
not allowed,
stupid!

What could possibly go wrong?



OoO-Processor:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

Cache:

```
0x70:  00 00 00 00
0x74:  00 00 00 00
0x78:  00 00 00 00
0x7C:  1F 20 2A 2B
```

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OS Memory:

```
0A 0B 0C 0D  :0x0
0E 0F 10 1A  :0x4
00 00 00 0C  :0x8
1F 20 2A 2B  :0xC
```

User Memory:

```
0A 0B 0C 0D  :0x70
0E 0F 10 1A  :0x74
1B 1C 1D 1E  :0x78
1F 20 2A 2B  :0x7C
```

Access
not allowed,
stupid!

What could possibly go wrong?

OoO-Processor:

Cache:

0x70:	00	00	00	00
0x74:	00	00	00	00
0x78:	00	00	00	00
0x7C:	1F	20	2A	2B

Code:

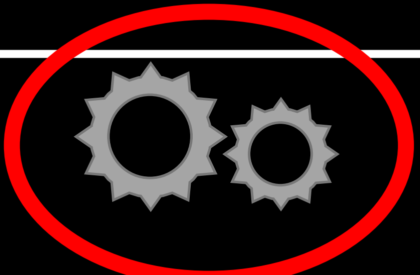
```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OS Memory:

0A	0B	0C	0D	:0x0
0E	0F	10	1A	:0x4
00	00	00	0C	:0x8
1F	20	2A	2B	:0xC

User Memory:

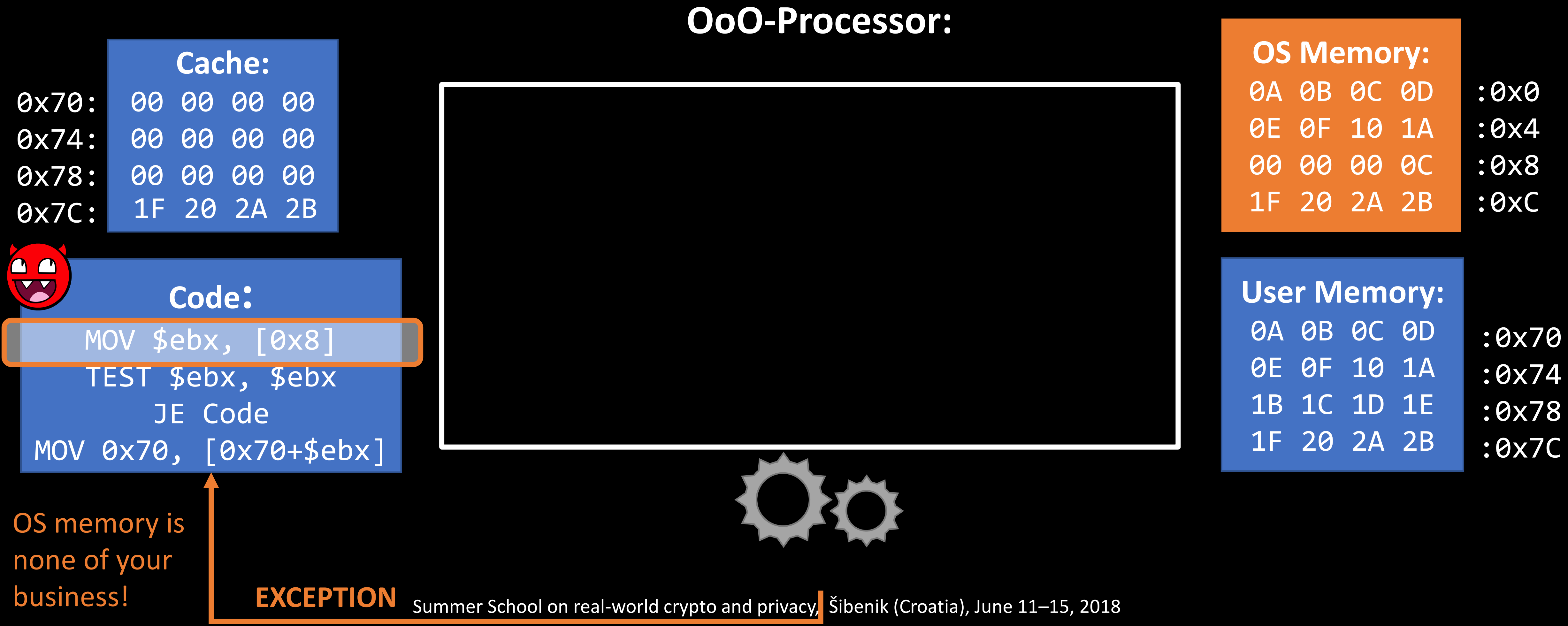
0A	0B	0C	0D	:0x70
0E	0F	10	1A	:0x74
1B	1C	1D	1E	:0x78
1F	20	2A	2B	:0x7C



Rollback!

Access
not allowed,
stupid!

What could possibly go wrong?



OoO-Processor:

Cache:

```
0x70: 00 00 00 00
0x74: 00 00 00 00
0x78: 00 00 00 00
0x7C: 1F 20 2A 2B
```

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

OS Memory:

```
0A 0B 0C 0D :0x0
0E 0F 10 1A :0x4
00 00 00 0C :0x8
1F 20 2A 2B :0xC
```

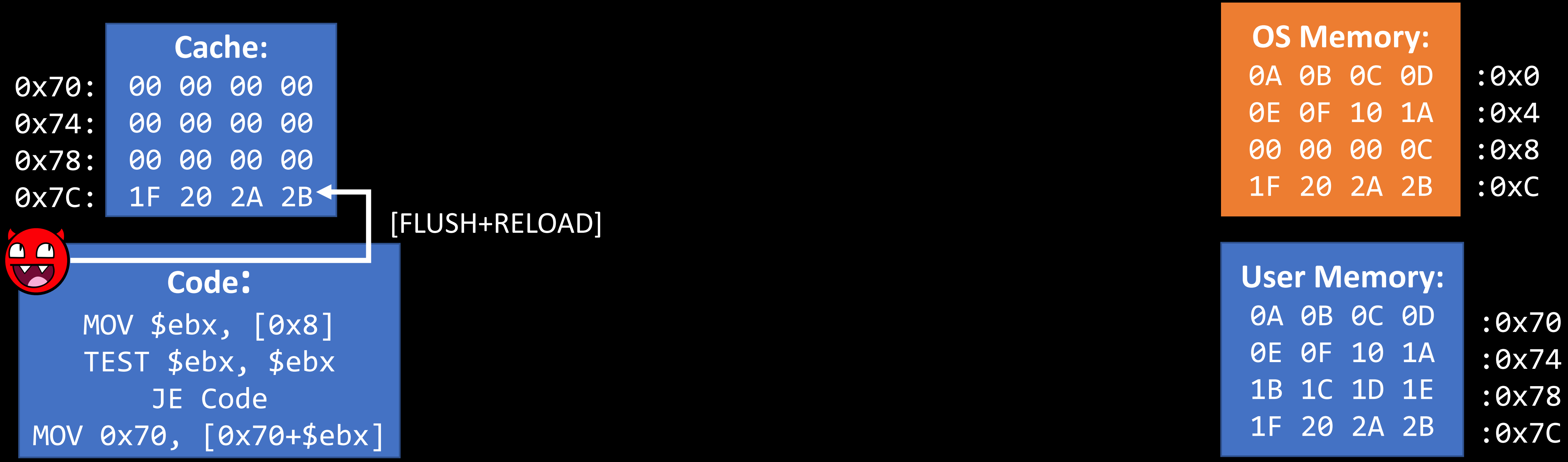
User Memory:

```
0A 0B 0C 0D :0x70
0E 0F 10 1A :0x74
1B 1C 1D 1E :0x78
1F 20 2A 2B :0x7C
```

OS memory is none of your business!

EXCEPTION

What could possibly go wrong?



Cache:

```
0x70:  00 00 00 00
0x74:  00 00 00 00
0x78:  00 00 00 00
0x7C:  1F 20 2A 2B
```

[FLUSH+RELOAD]

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

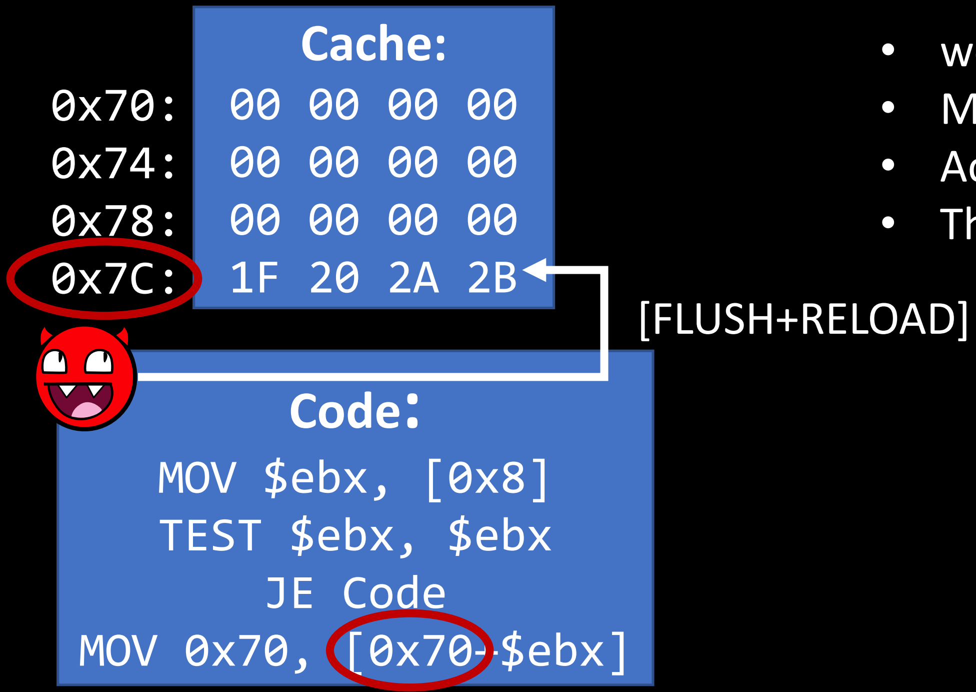
OS Memory:

```
0A 0B 0C 0D  :0x0
0E 0F 10 1A  :0x4
00 00 00 0C  :0x8
1F 20 2A 2B  :0xC
```

User Memory:

```
0A 0B 0C 0D  :0x70
0E 0F 10 1A  :0x74
1B 1C 1D 1E  :0x78
1F 20 2A 2B  :0x7C
```

What could possibly go wrong?



Cache:

```
0x70:  00 00 00 00
0x74:  00 00 00 00
0x78:  00 00 00 00
0x7C:  1F 20 2A 2B
```

Code:

```
MOV $ebx, [0x8]
TEST $ebx, $ebx
JE Code
MOV 0x70, [0x70+$ebx]
```

- well well what do we have here..
- Memory access happened at 0x7C
- Actually, my start address was 0x70
- The value at 0x8 must have been:

0x7C-0x70 = 0x0C!

OS Memory:

```
0A 0B 0C 0D  :0x0
0E 0F 10 1A  :0x4
00 00 00 0C  :0x8
1F 20 2A 2B  :0xC
```

User Memory:

```
0A 0B 0C 0D  :0x70
0E 0F 10 1A  :0x74
1B 1C 1D 1E  :0x78
1F 20 2A 2B  :0x7C
```