Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Secure Software Systems

The Promises and Pitfalls of Hardware-Assisted Security

Alexandra Dmitrienko
Julius-Maximilians-Universität Würzburg

alexandra.dmitrienko@uni-wuerzburg.de

The Great Promise of Trusted Computing

Historical Overview: Deployed Systems

1970 **1980** **1990** **2000** **2010**

Cambridge CAP

VAX/VMS

Trusted Platform Module (TPM)

PUFs

Java security architecture

Late launch/TXT

TPM 2.0

Intel SGX

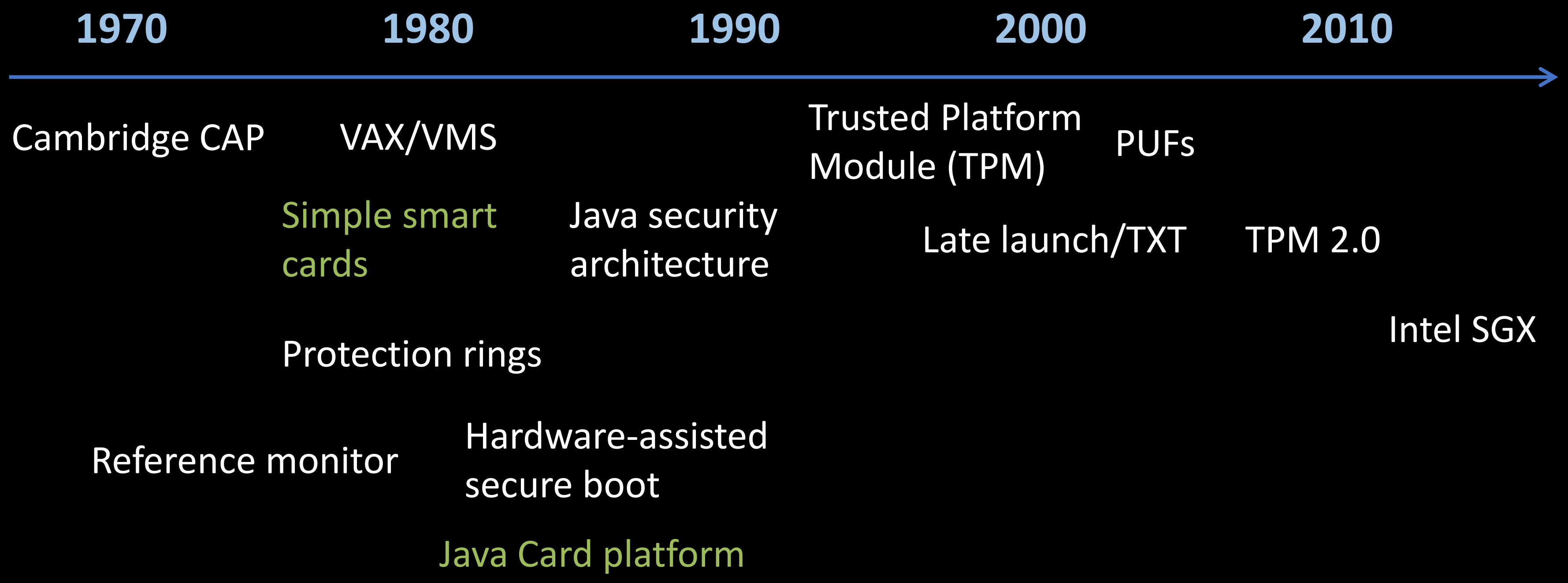
Protection rings

Reference monitor

Hardware-assisted secure boot

Computer security
Mobile security
Smart card security

Historical Overview: Deployed Systems



Computer security
Mobile security
Smart card security

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

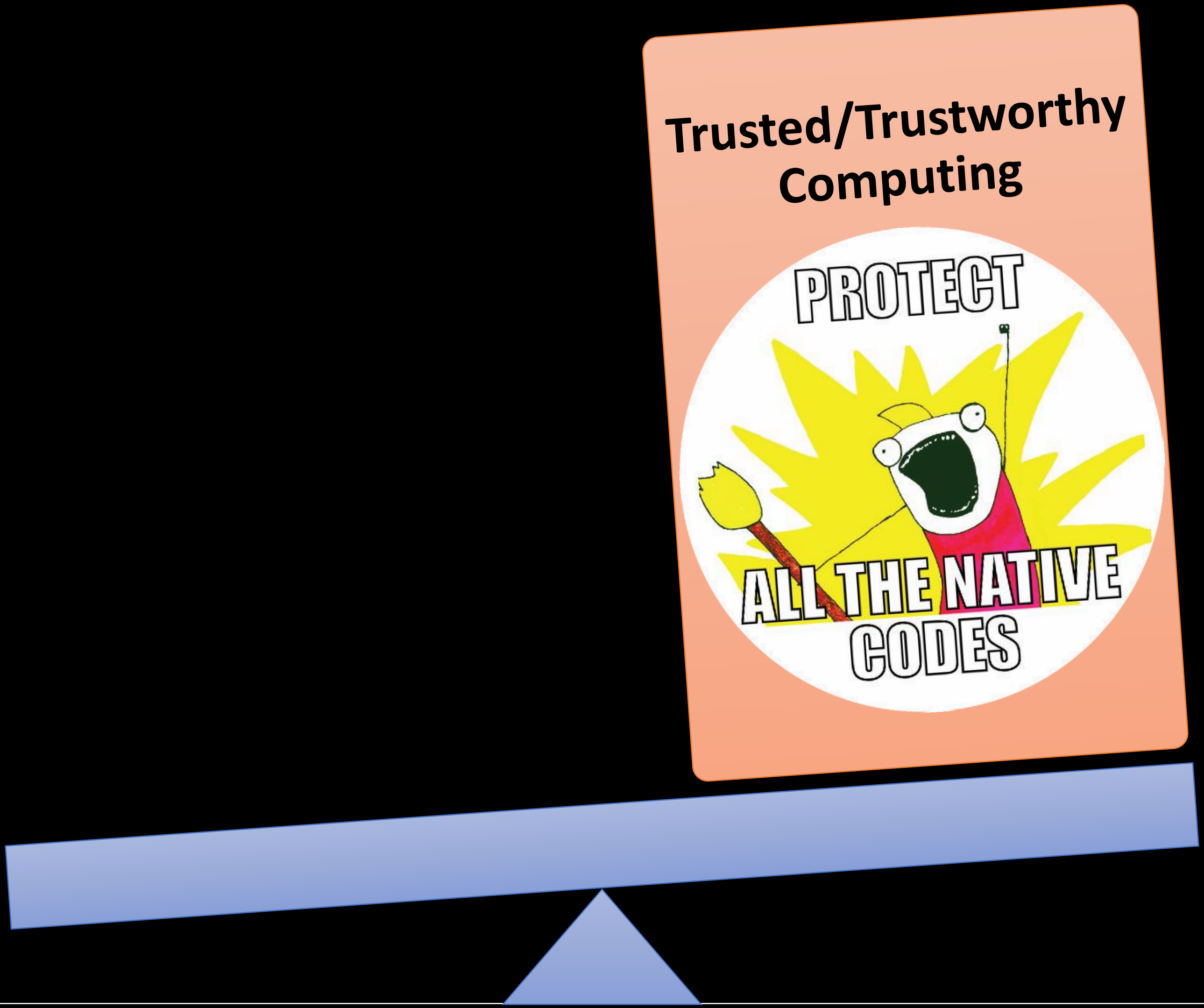
Java Card platform

Mobile OS security architectures

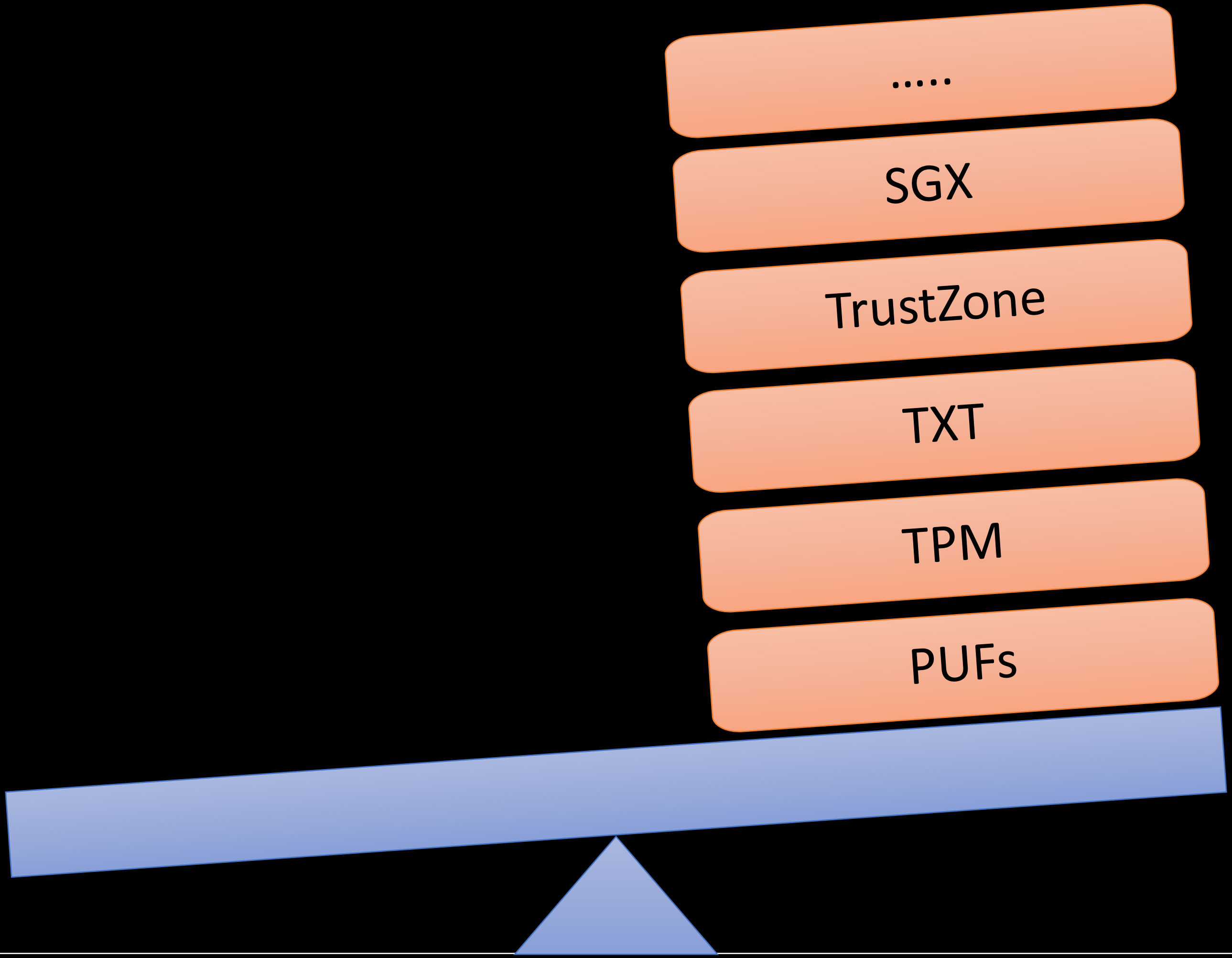
Mobile Trusted Module (MTM)

Computer security
Mobile security
Smart card security

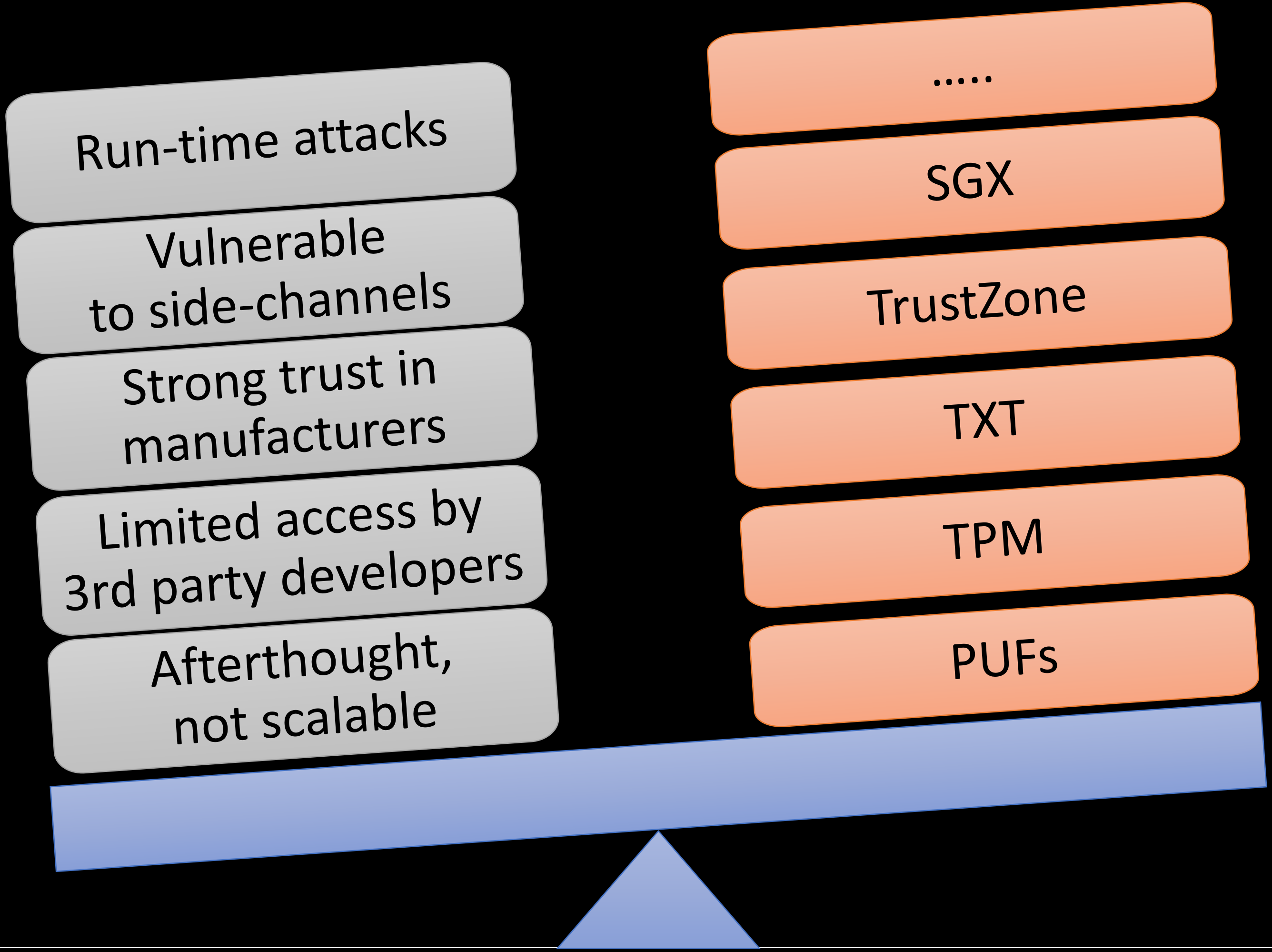
Trusted Computing under Attack



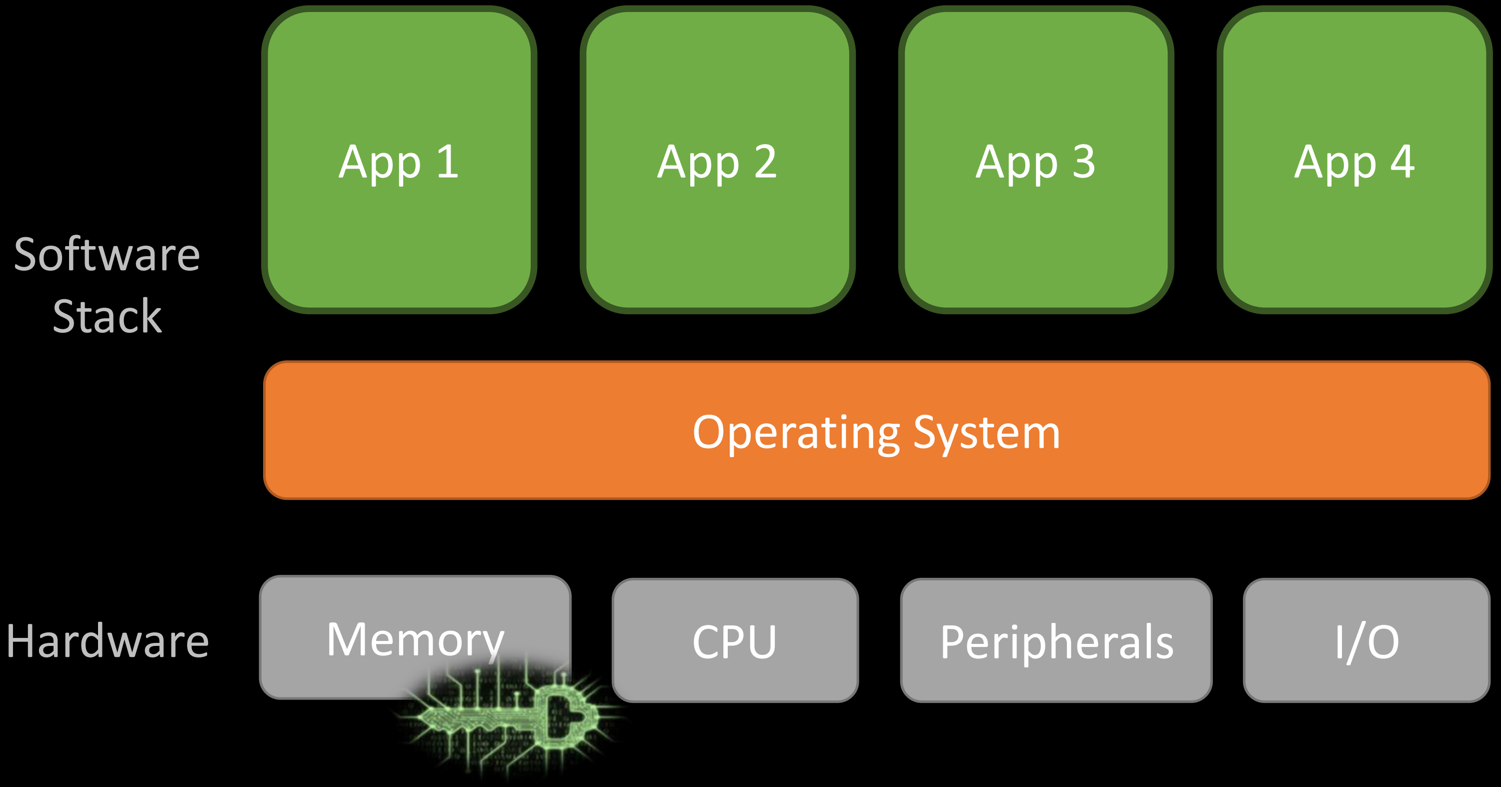
Trusted Computing under Attack



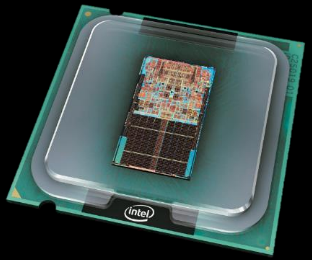
Trusted Computing under Attack


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

Goal: Self-Contained Security

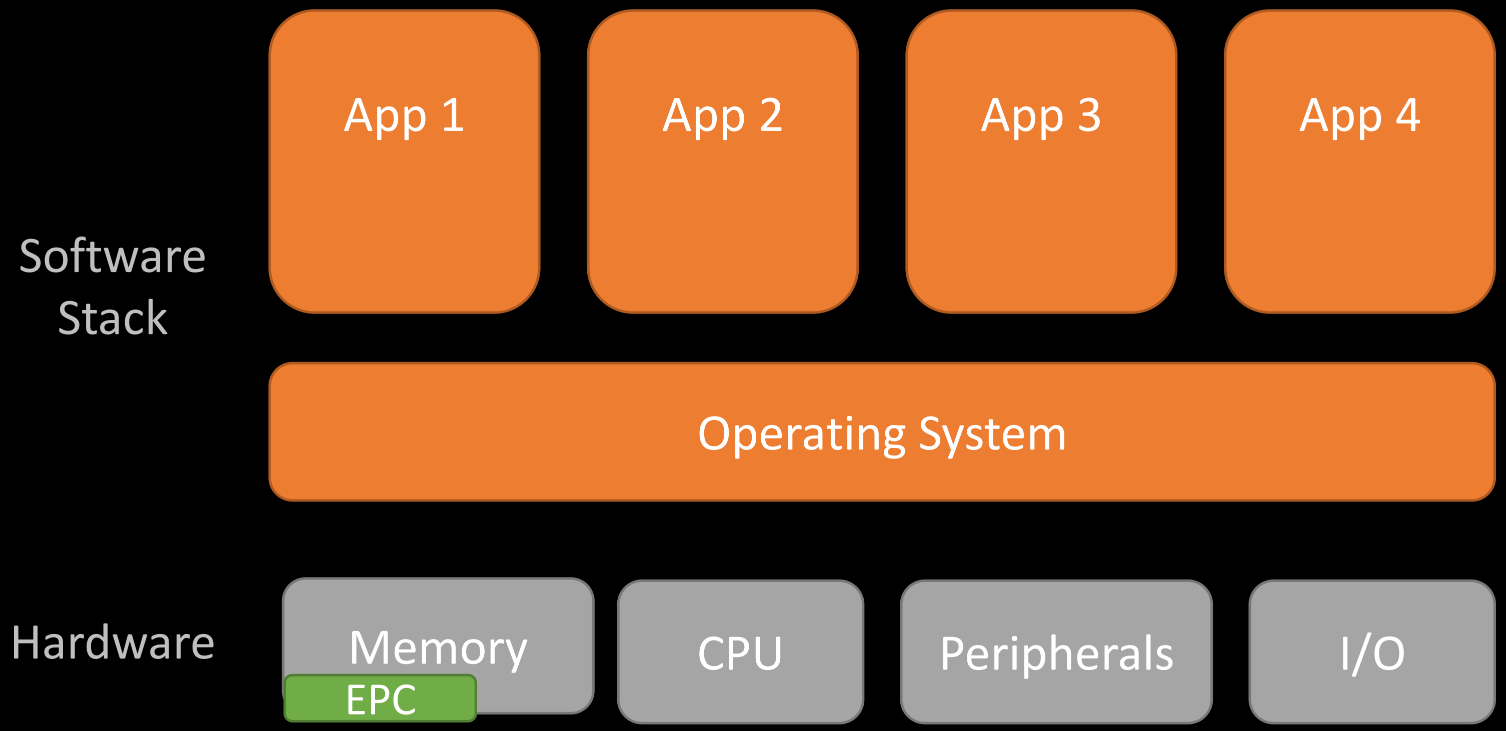


- Isolated execution
- Platform integrity
- Secure storage
- Device identification
- Device authentication capabilities



Intel SGX

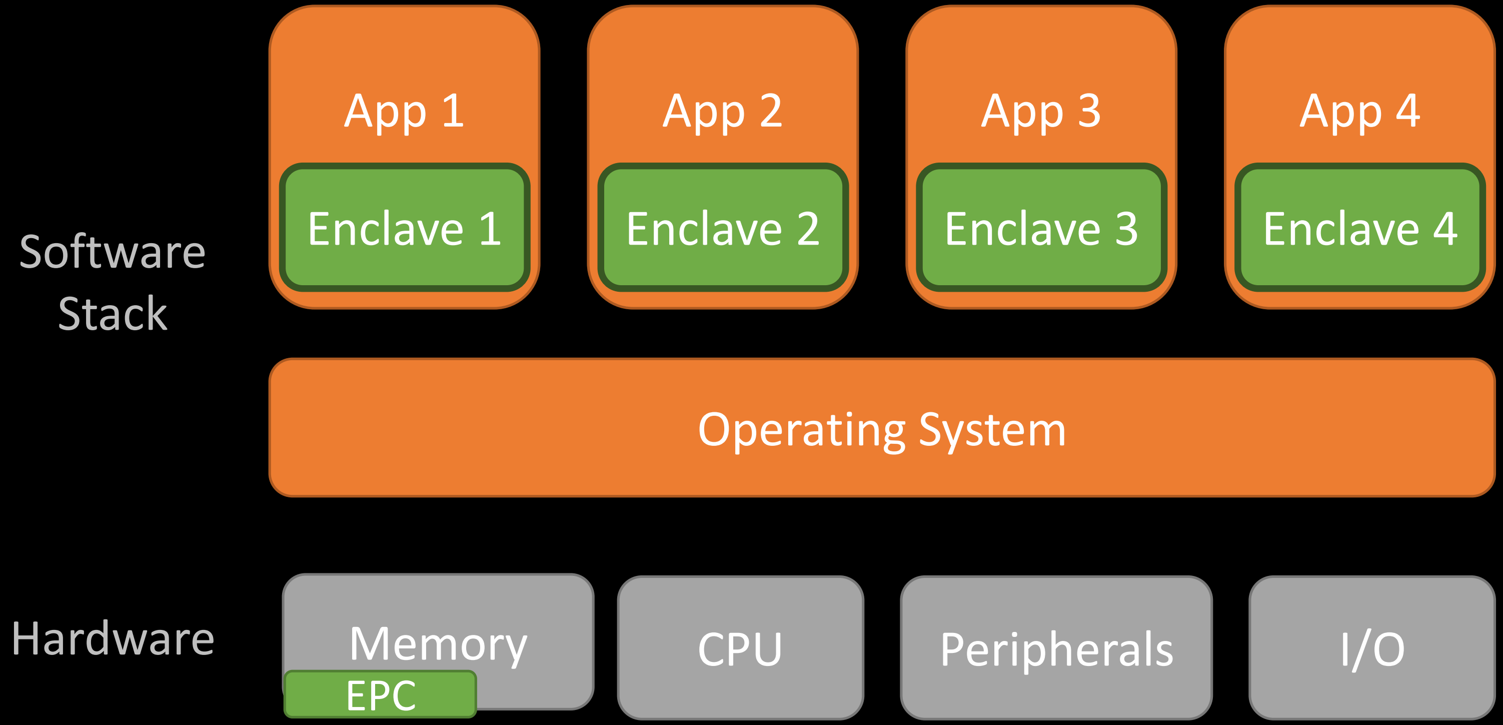
Intel Software Guard Extensions (SGX)



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

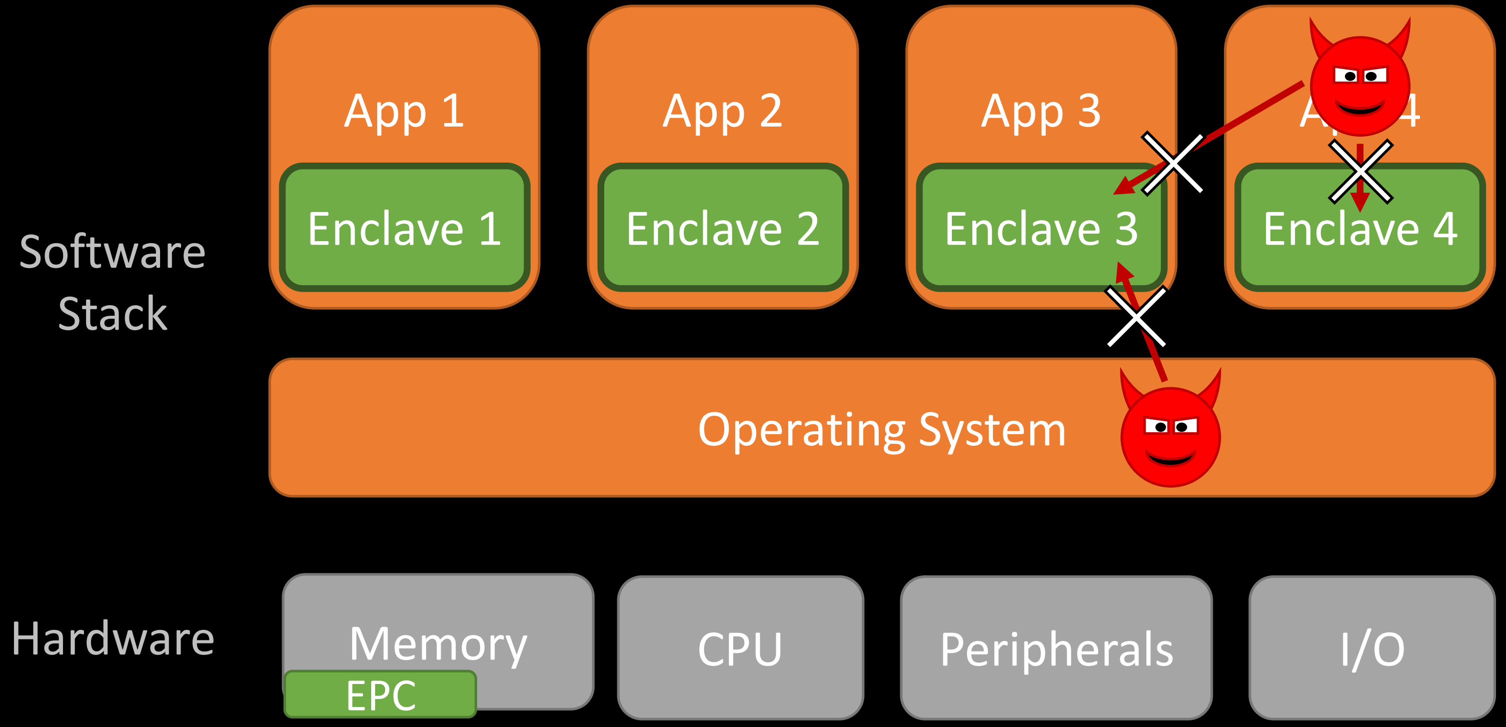
- OS creates and manages enclaves, allocates memory from Enclave Page Cache (EPC)
- OS maps physical to virtual memory, as well as loads data and code into enclave
- Trust assumptions: All software components untrusted



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

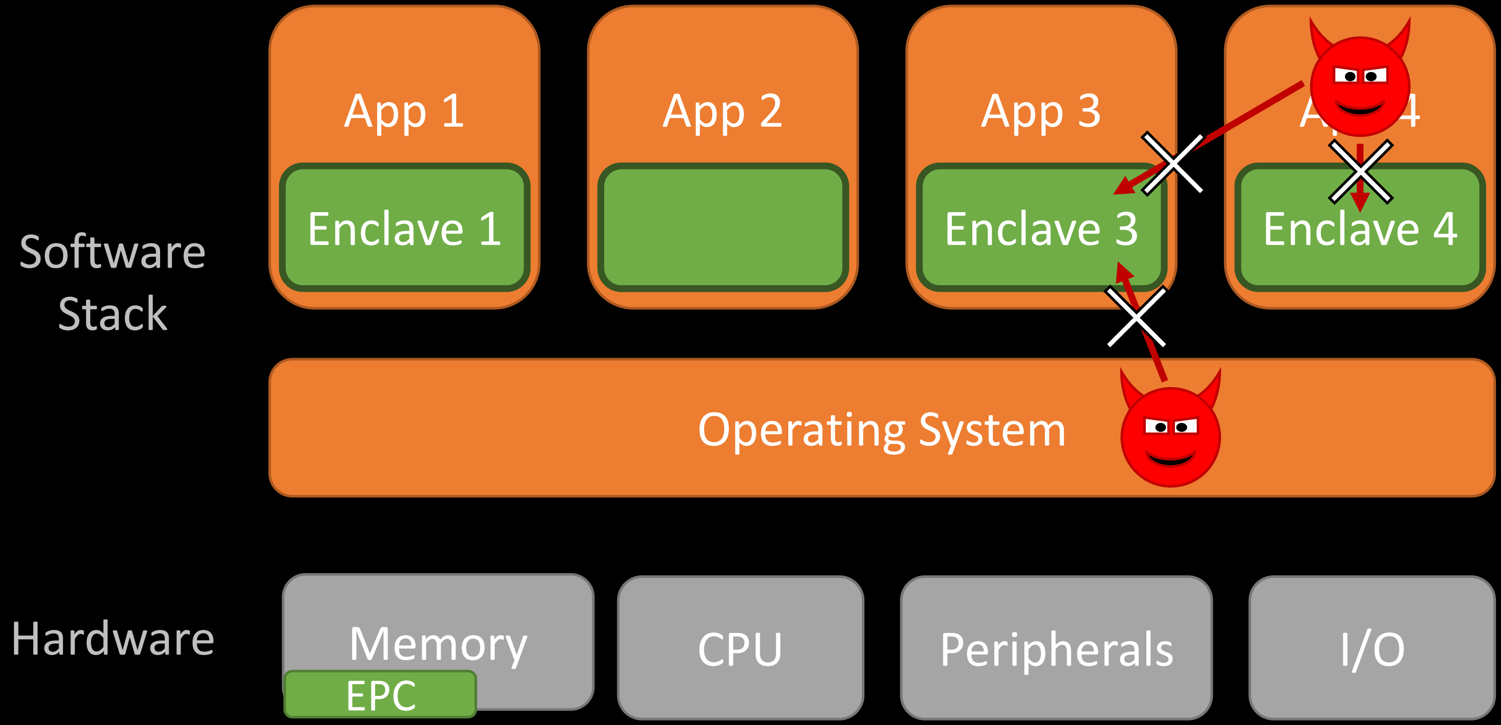
- OS creates and manages enclaves, allocates memory from Enclave Page Cache (EPC)
- OS maps physical to virtual memory, as well as loads data and code into enclave
- Trust assumptions: All software components untrusted



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

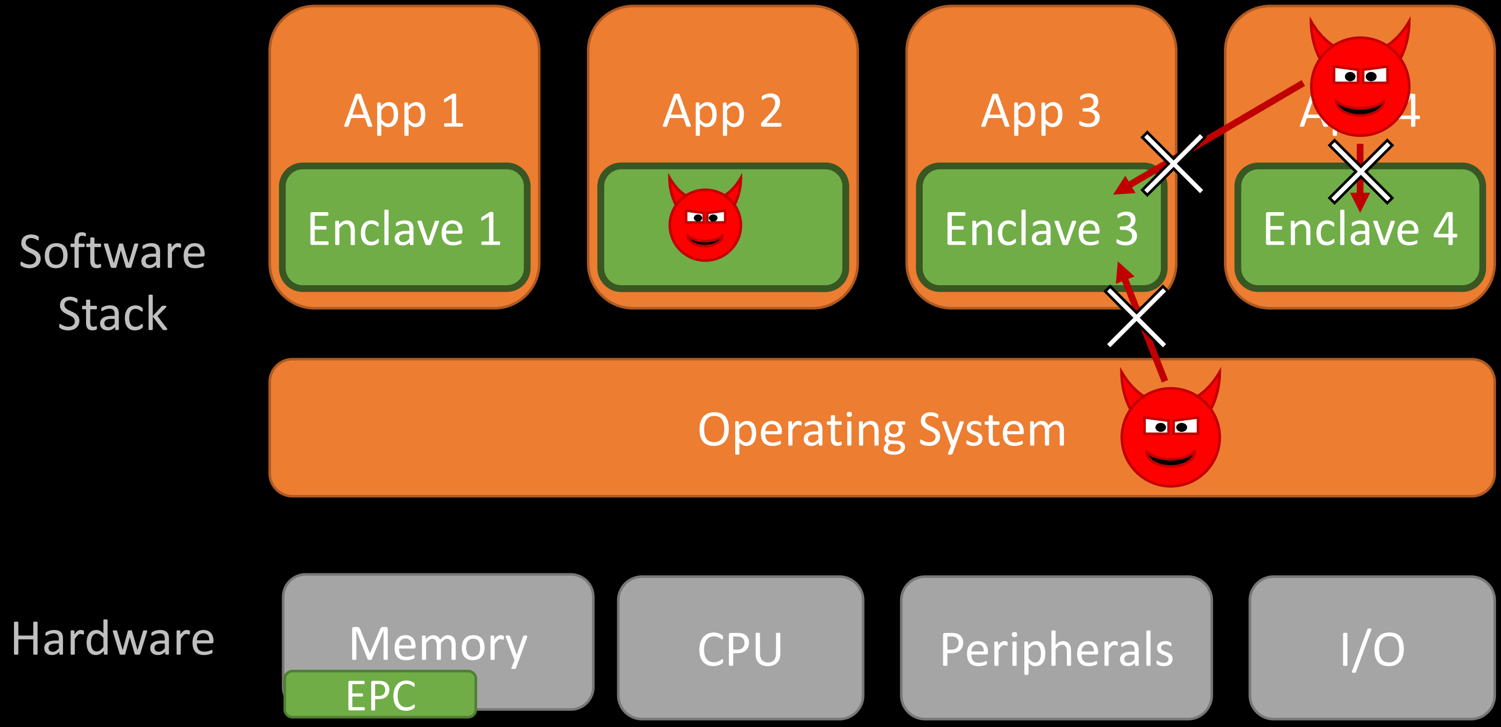
- Asynchrones Enclave Exit (AEX): Enclaves interruptable, CPU saves/deletes context in CPU registers



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

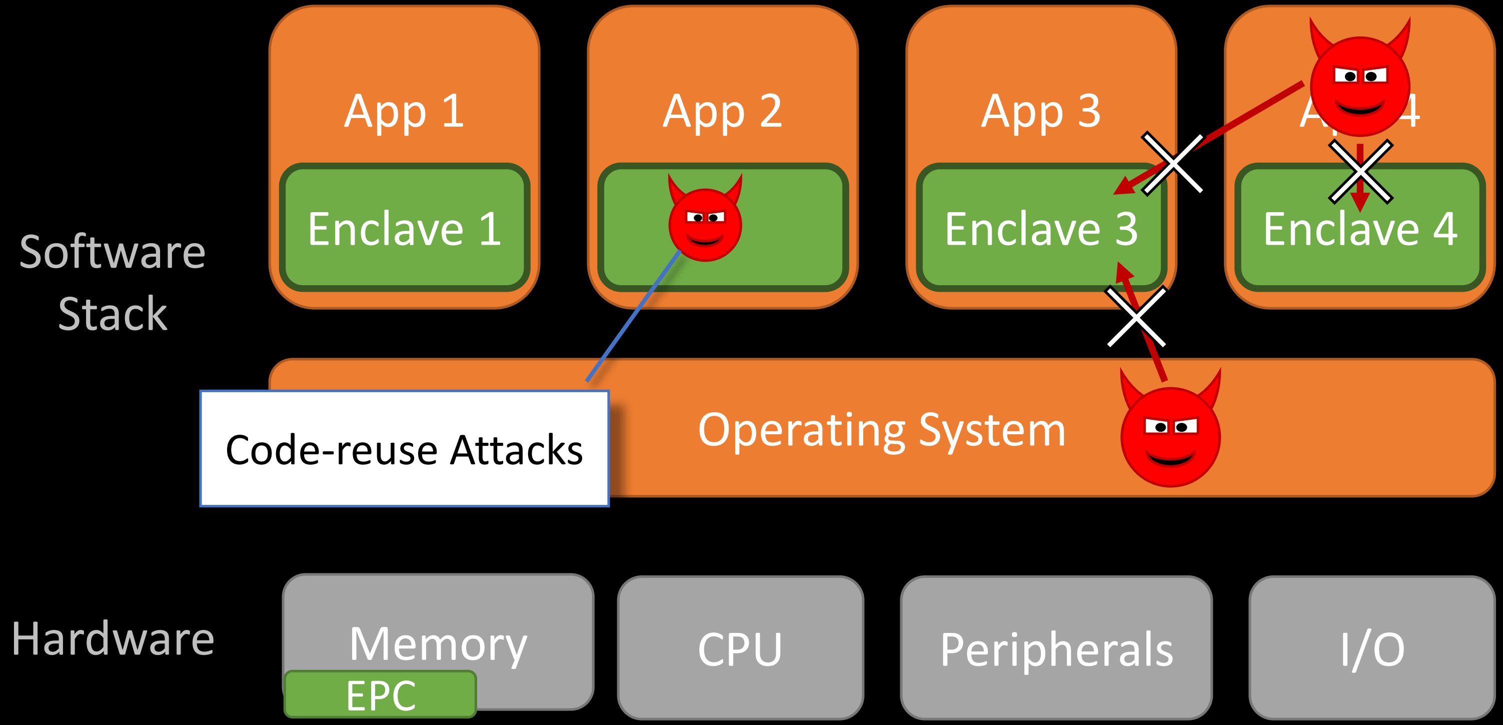
- Asynchrones Enclave Exit (AEX): Enclaves interruptable, CPU saves/deletes context in CPU registers



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

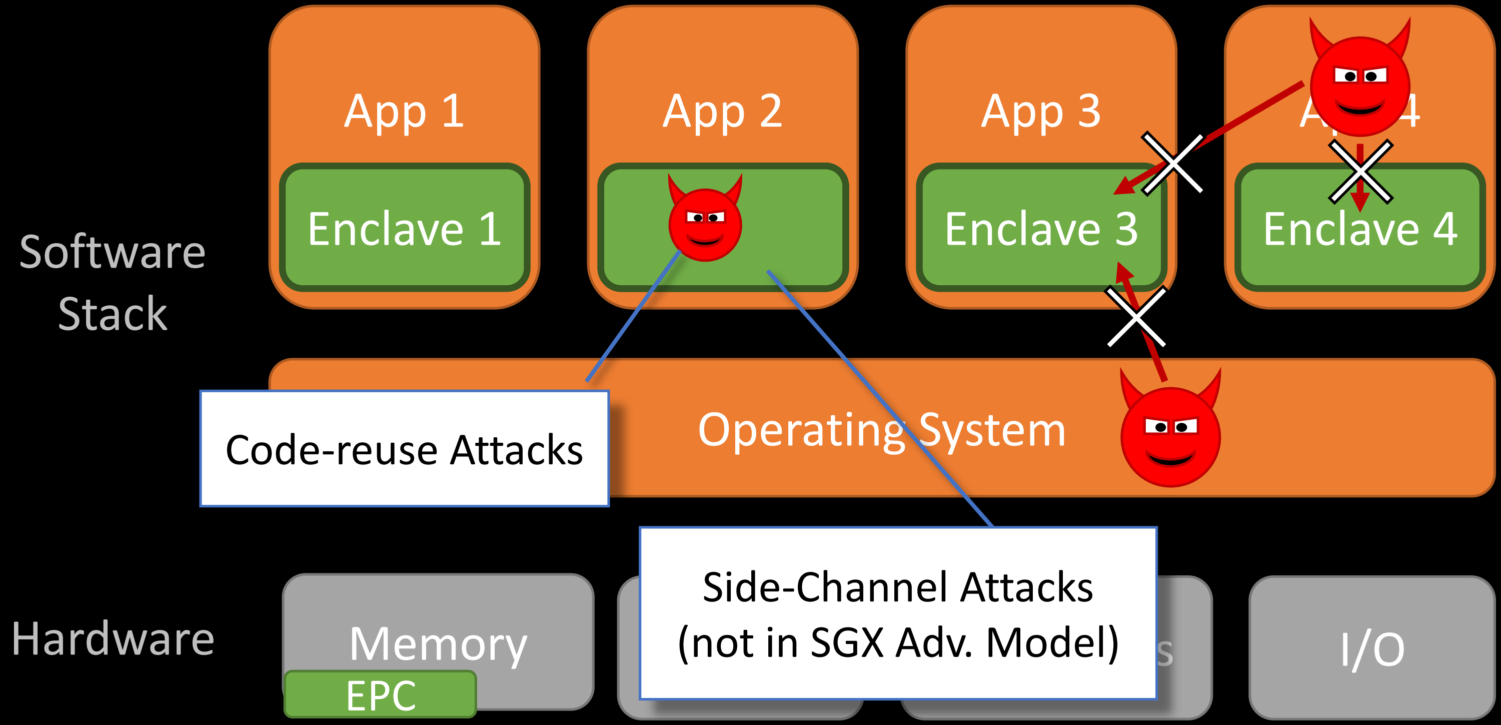
- Asynchrones Enclave Exit (AEX): Enclaves interruptable, CPU saves/deletes context in CPU registers



EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

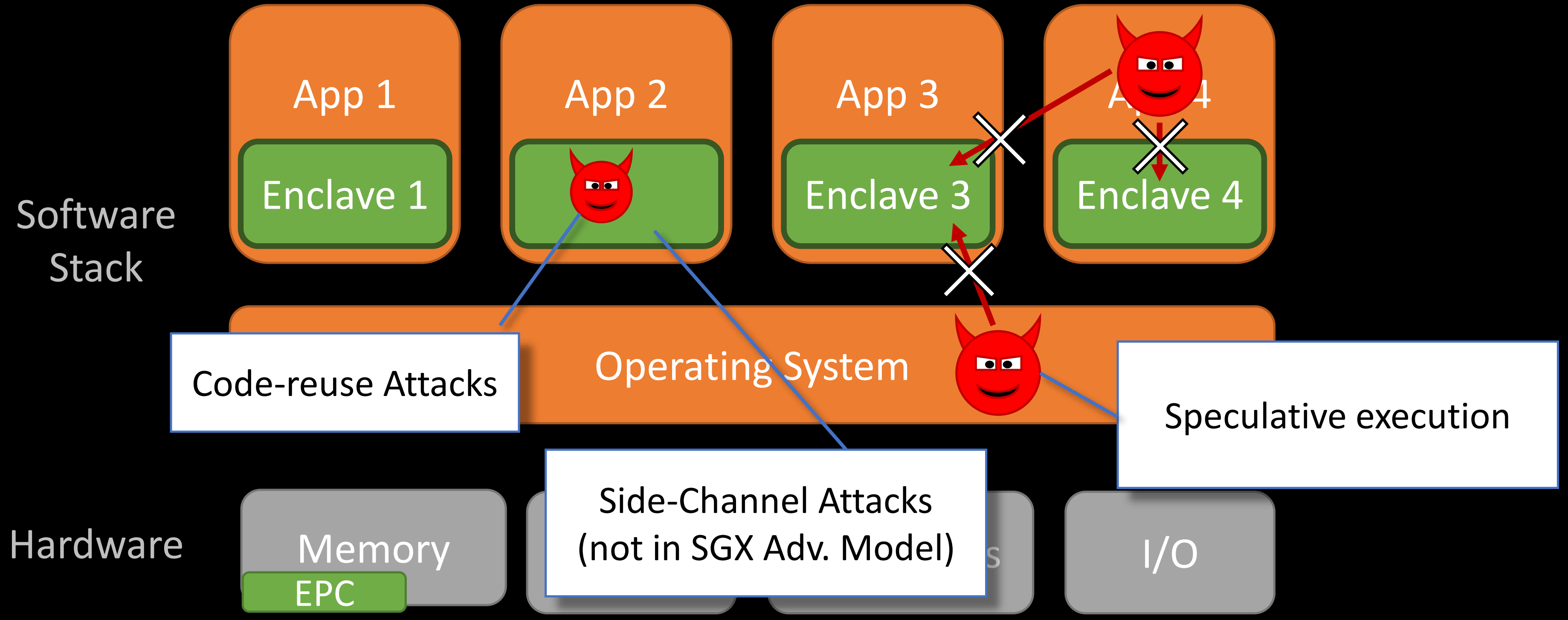
- Asynchrones Enclave Exit (AEX): Enclaves interruptable, CPU saves/deletes context in CPU registers



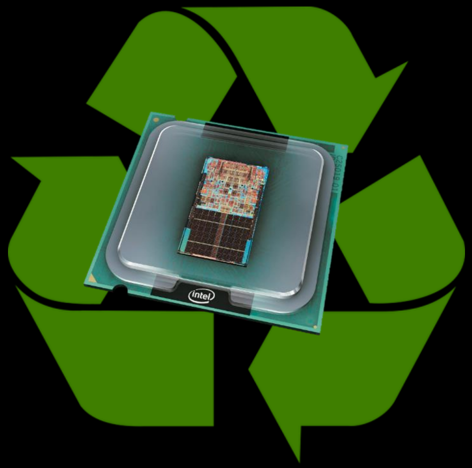
EPC: Enclave Page Cache (Dedicated Physical Memory)

Intel Software Guard Extensions (SGX)

- Asynchrones Enclave Exit (AEX): Enclaves interruptable, CPU saves/deletes context in CPU registers



EPC: Enclave Page Cache (Dedicated Physical Memory)



Code-reuse on SGX

Code-reuse Attacks: Big Picture

The New Y

Saturday, January 6, 2007

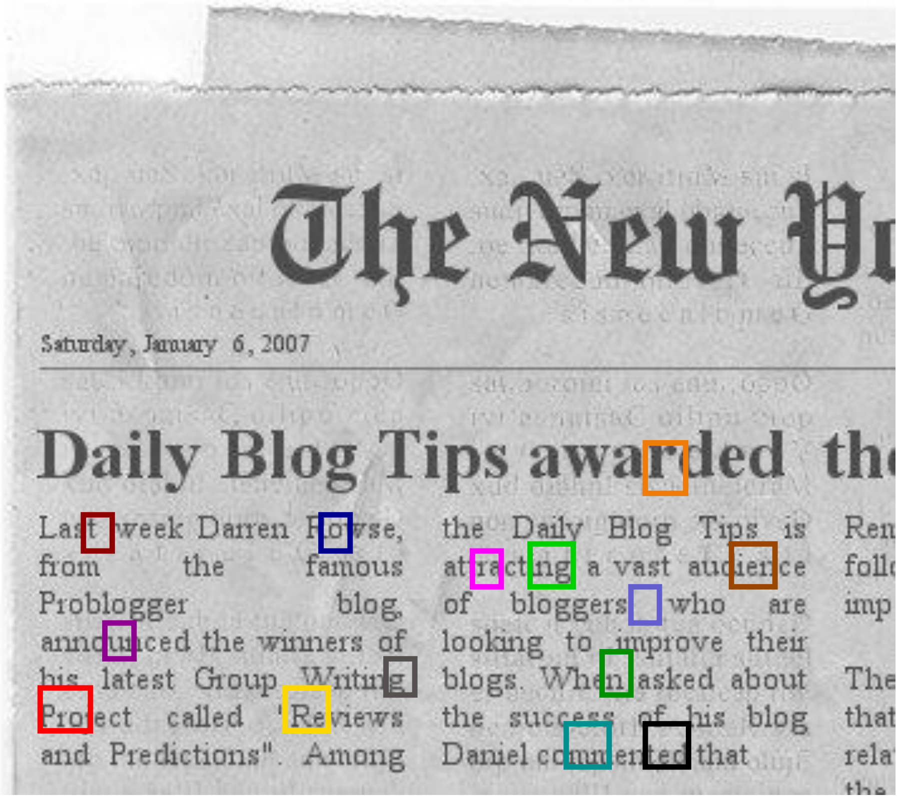
Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Ren
foll
imp

The
that
rela
the

Code-reuse Attacks: Big Picture



The New Y

Saturday, January 6, 2007

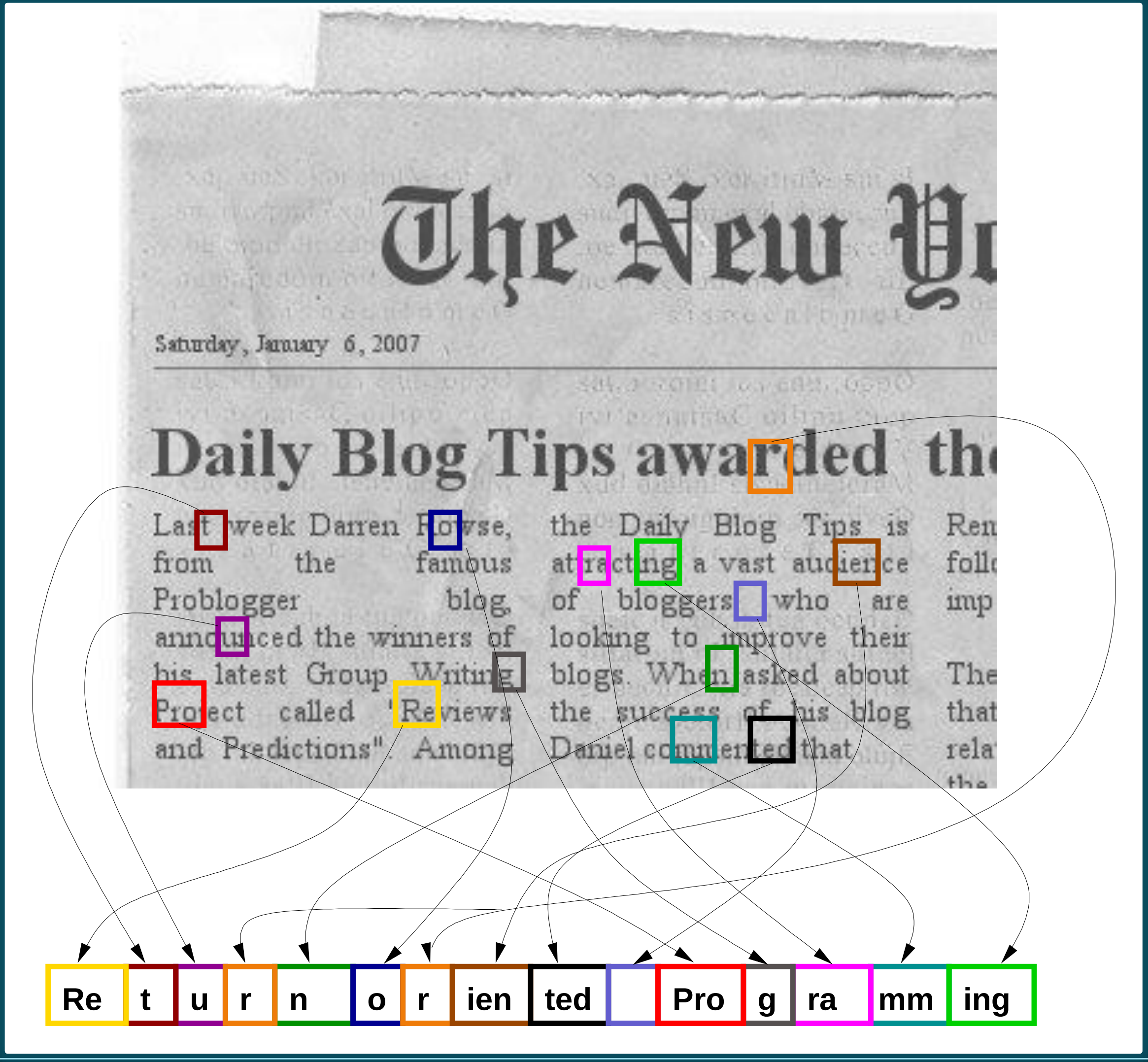
Daily Blog Tips awarded the

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Ren foll imp

The that rela the

Code-reuse Attacks: Big Picture


The New York
Saturday, January 6, 2007
Daily Blog Tips awarded the
Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among
the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that
Re
t
u
r
n
o
r
ien
ted
Pro
g
ra
mm
ing

Hacking in Darkness: ROP against Secure Enclaves

[Lee et al., USENIX Security 2017]

- Memory corruption attack against Intel SGX (Dark-ROP)
- Combines ROP techniques with oracles that inform about internal state of a victim enclave
- Requires kernel privileges
- Relies on running the target enclave multiple times and crashes to leak information
- Demonstrates how the security of SGX can be disarmed
  - Exfiltration of all memory contents from the enclave (code and data)
  - Bypassing the SGX attestation

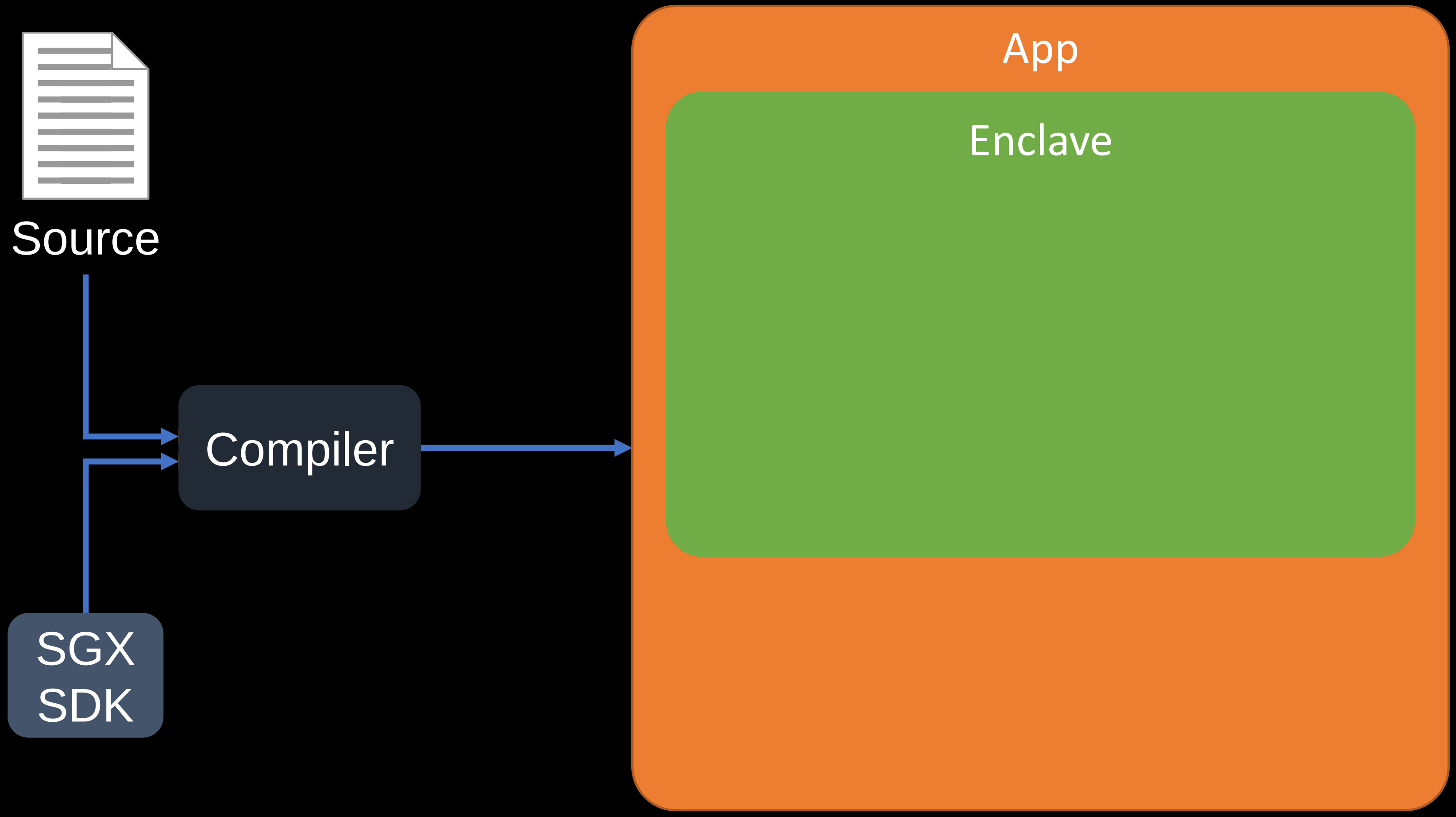
SGX-Shield: Randomization for SGX Enclaves

[Seo et al., NDSS 2017]

- Address Space Layout Randomization (ASLR) for SGX enclaves
- Effective against ROP, since it relies on addresses of code snippets (gadgets)
- Limited entropy due to limited memory space
- Still effective against Dark-ROP
  - Since an enclave will be re-randomized after the crash

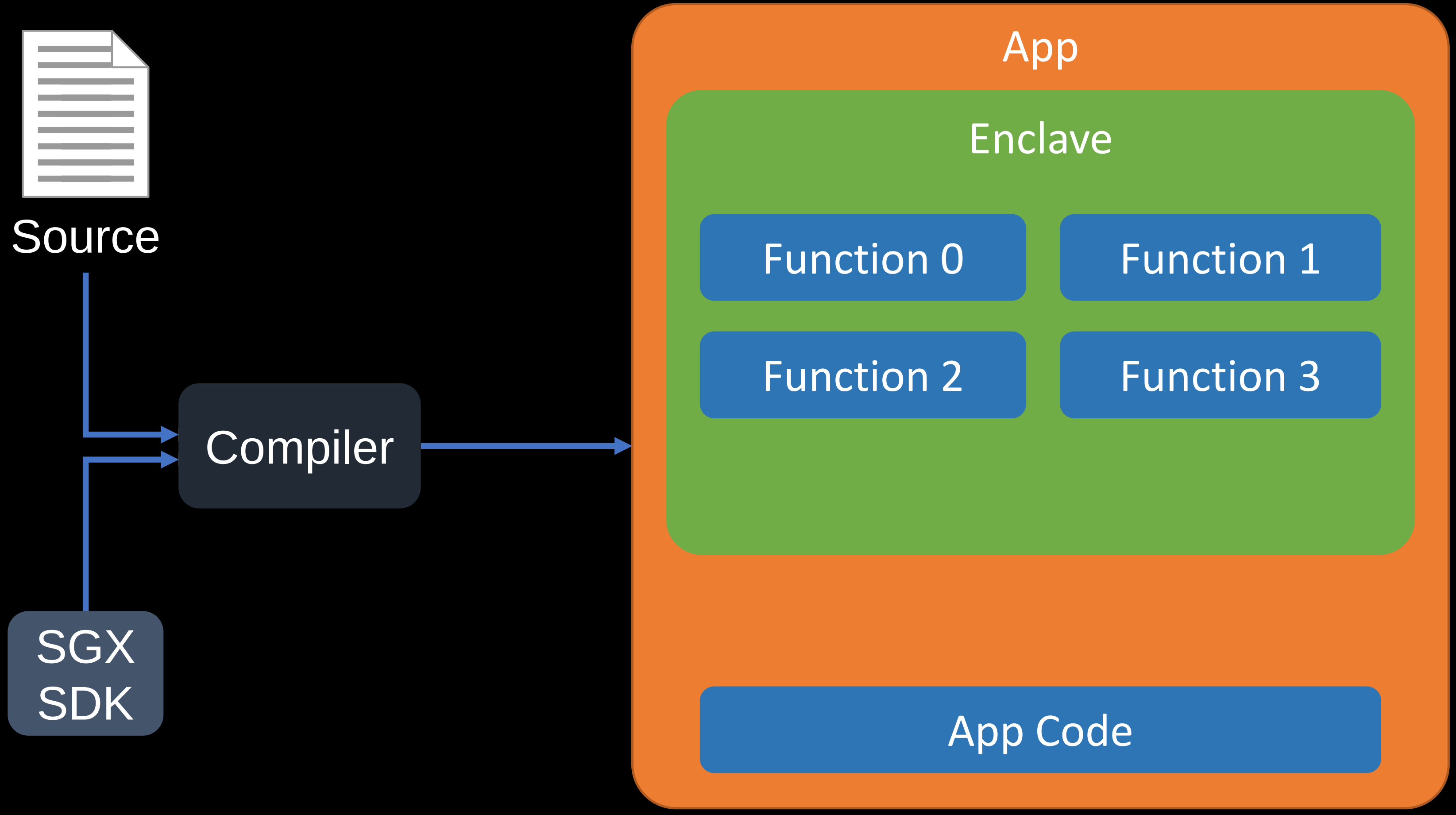
SGX SDK and The Guard's Dilemma

[Biondo et al., USENIX Security 2018]



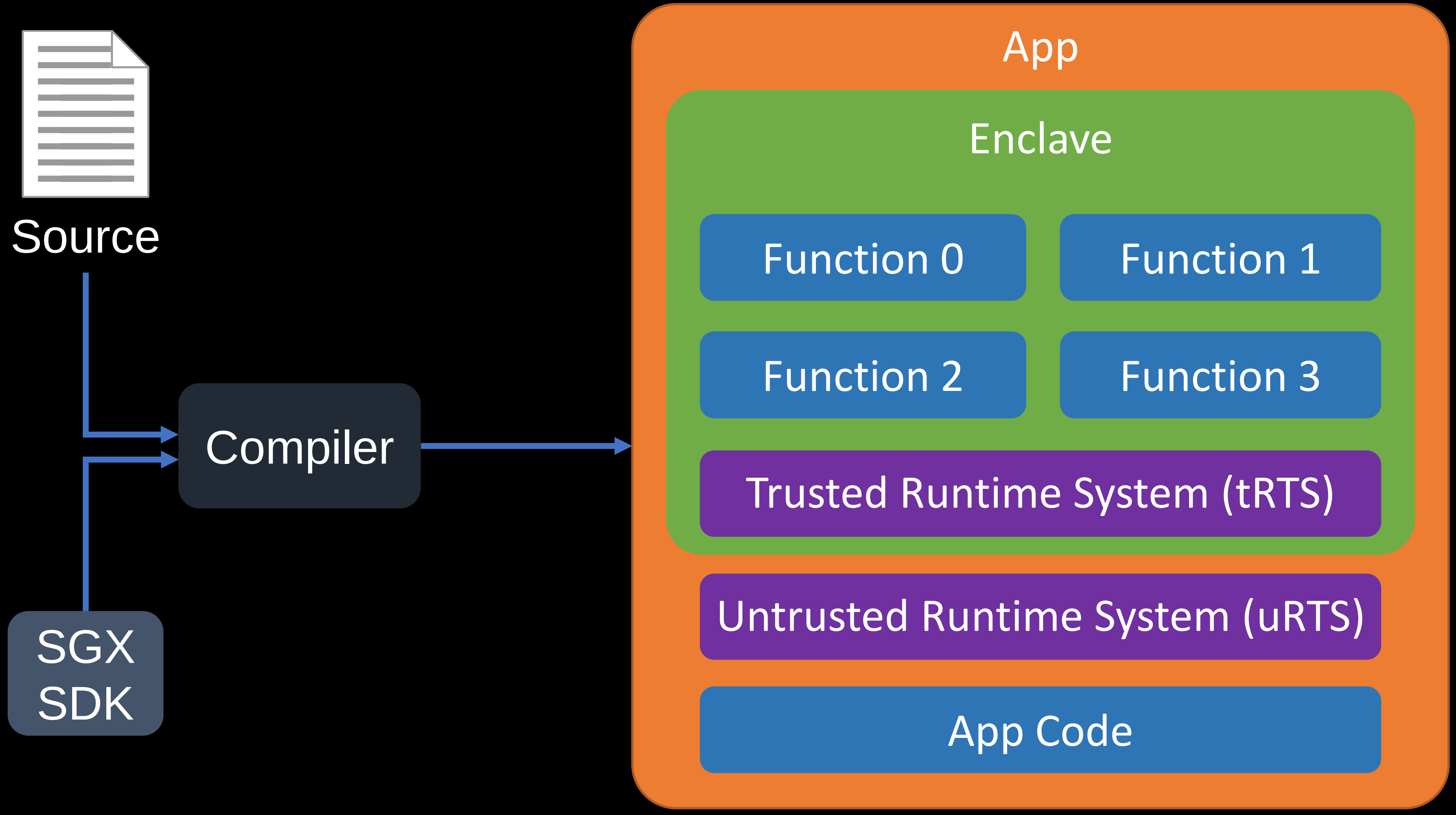
SGX SDK and The Guard's Dilemma

[Biondo et al., USENIX Security 2018]



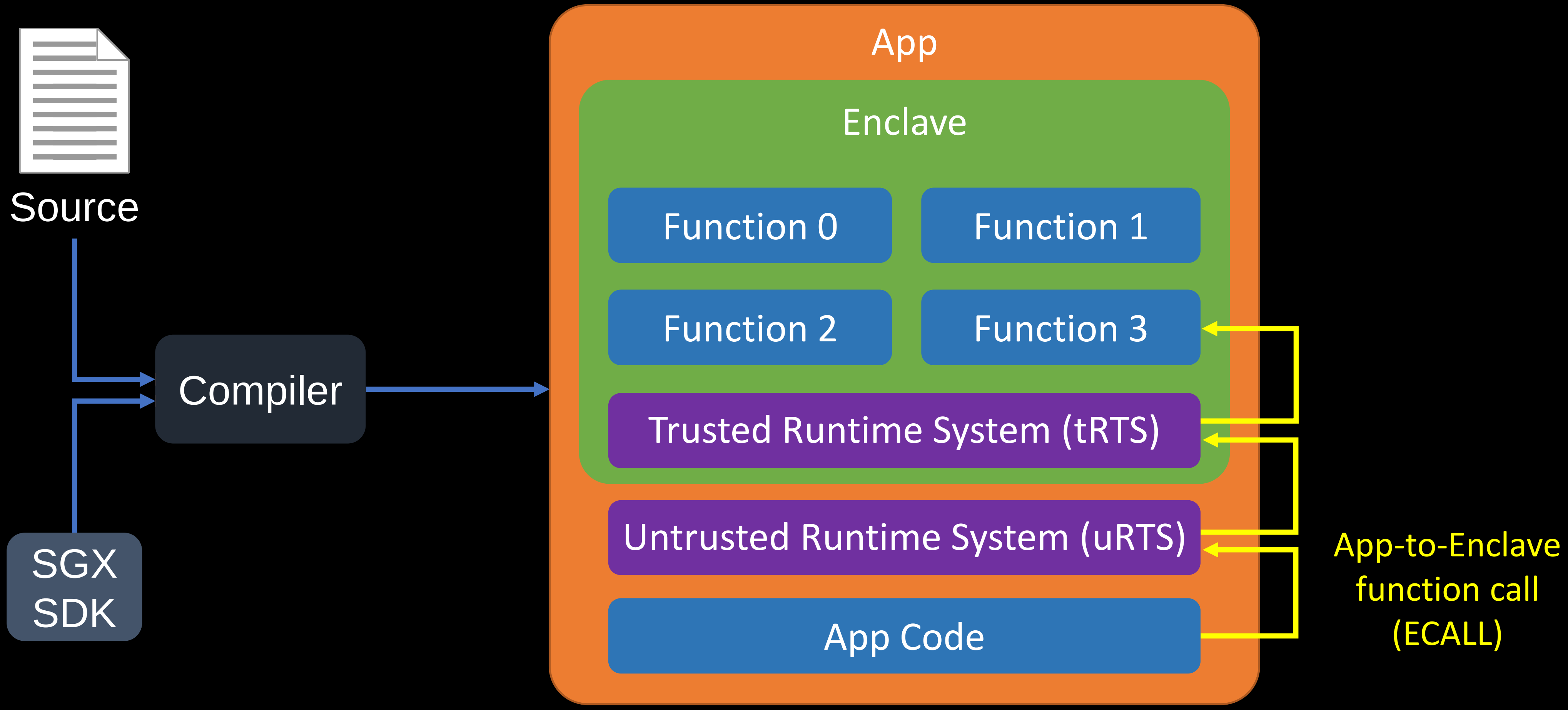
SGX SDK and The Guard’s Dilemma

[Biondo et al., USENIX Security 2018]



SGX SDK and The Guard’s Dilemma

[Biondo et al., USENIX Security 2018]

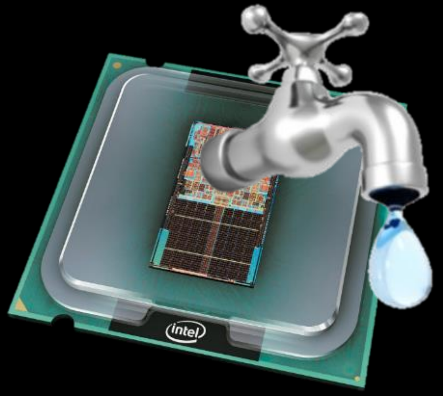

Source
Compiler
SGX SDK
App
Enclave
Function 0
Function 1
Function 2
Function 3
Trusted Runtime System (tRTS)
Untrusted Runtime System (uRTS)
App Code
App-to-Enclave function call (ECALL)

SGX SDK and The Guard’s Dilemma

[Biondo et al., USENIX Security 2018]

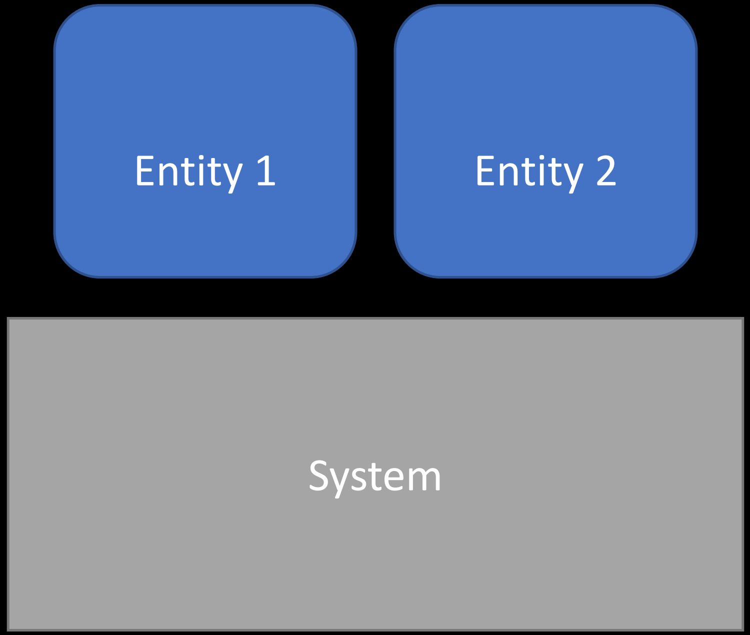
- tRTS is not randomized by SGX-Shield
- It cannot be randomized due to architectural specifics
  - E.g., enclave functions are invoked using fixed pre-defined entry points

- Contributions by Biondo et al.:
  - show that tRTS has enough gadgets to mount ROP
  - develop new techniques that do not require enclave crashes
  - new techniques do not require kernel privileges from an attacker

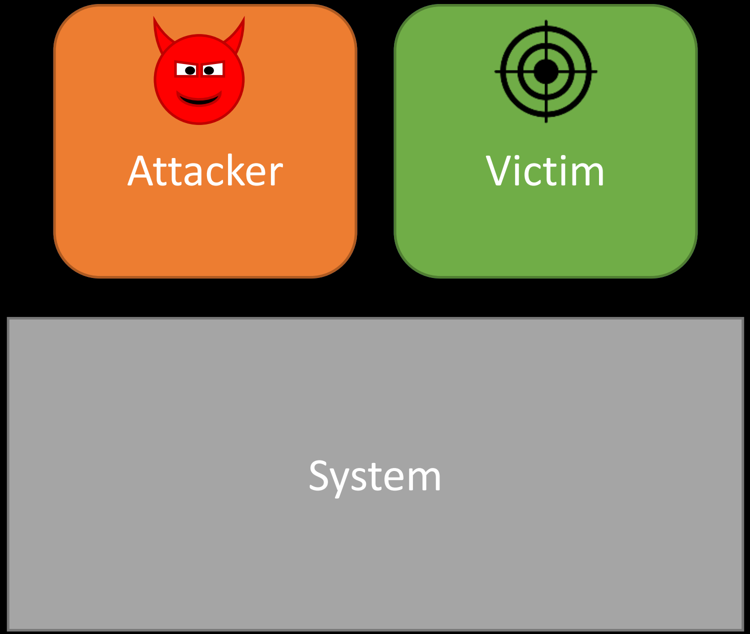


Leaky SGX

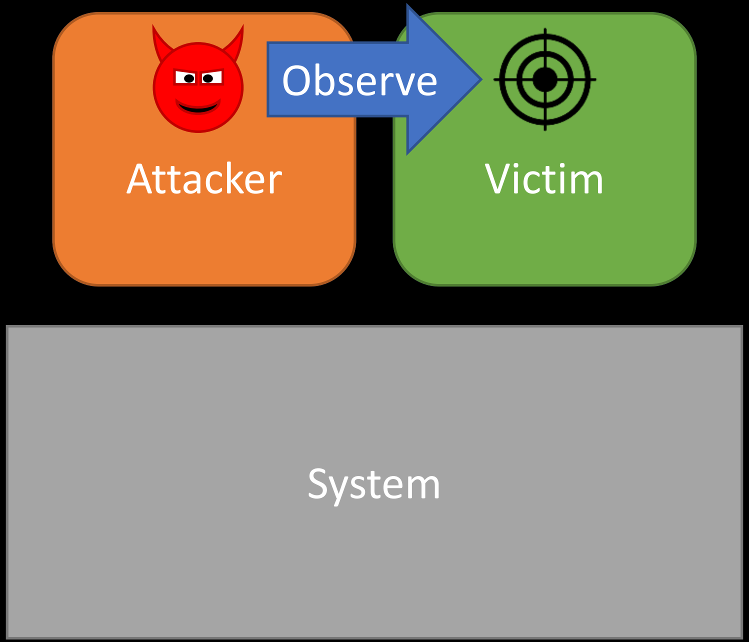
Side-Channel Attack: General Principle



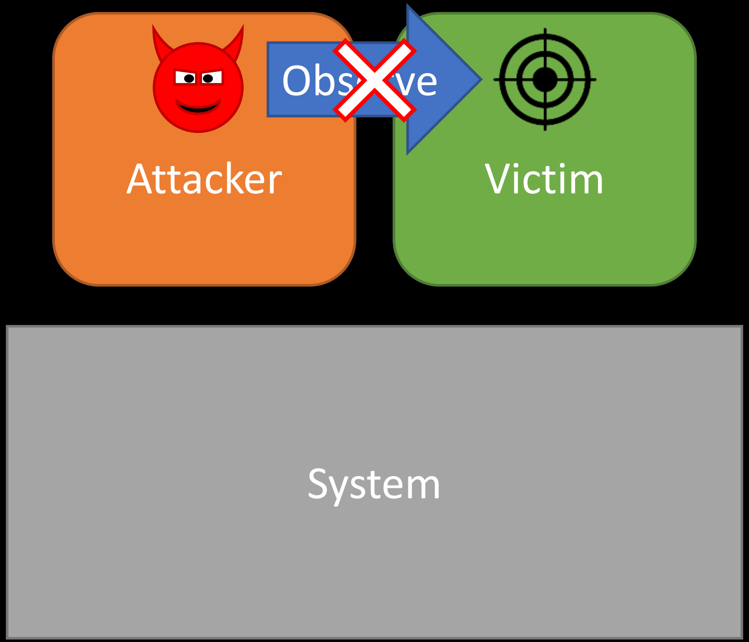
Side-Channel Attack: General Principle



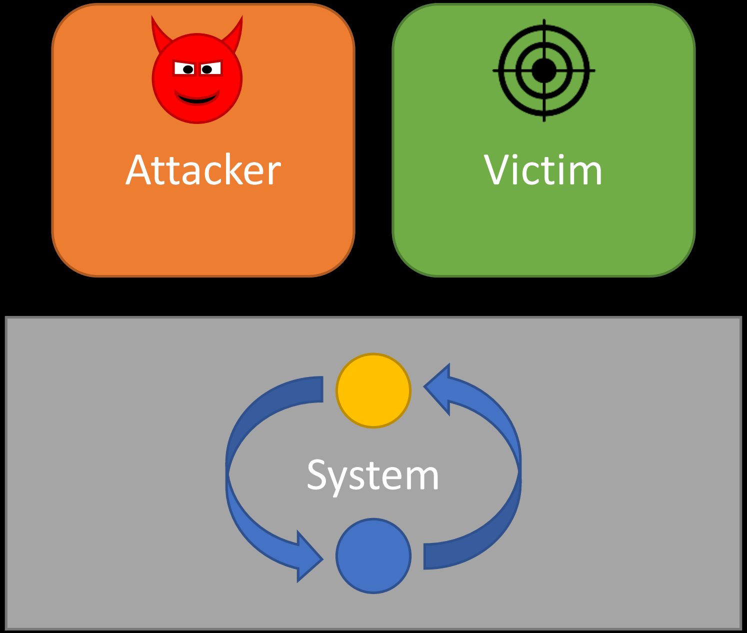
Side-Channel Attack: General Principle



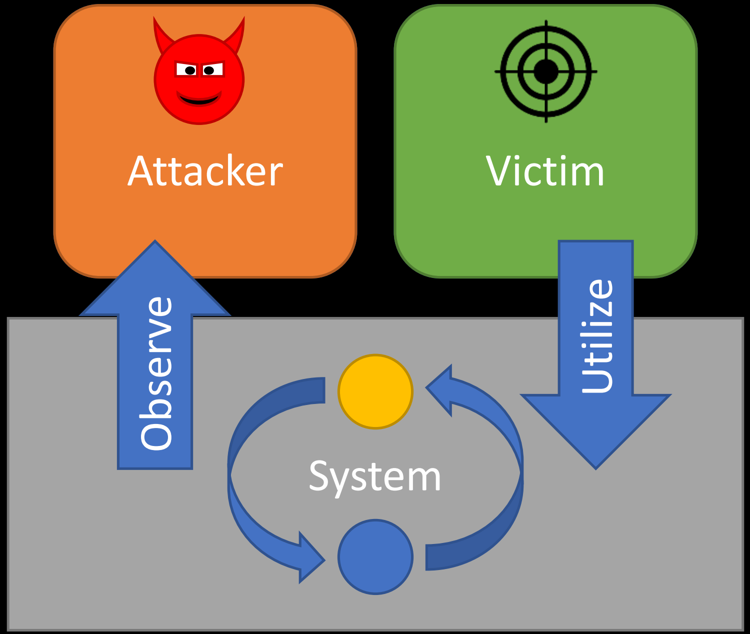
Side-Channel Attack: General Principle



Side-Channel Attack: General Principle

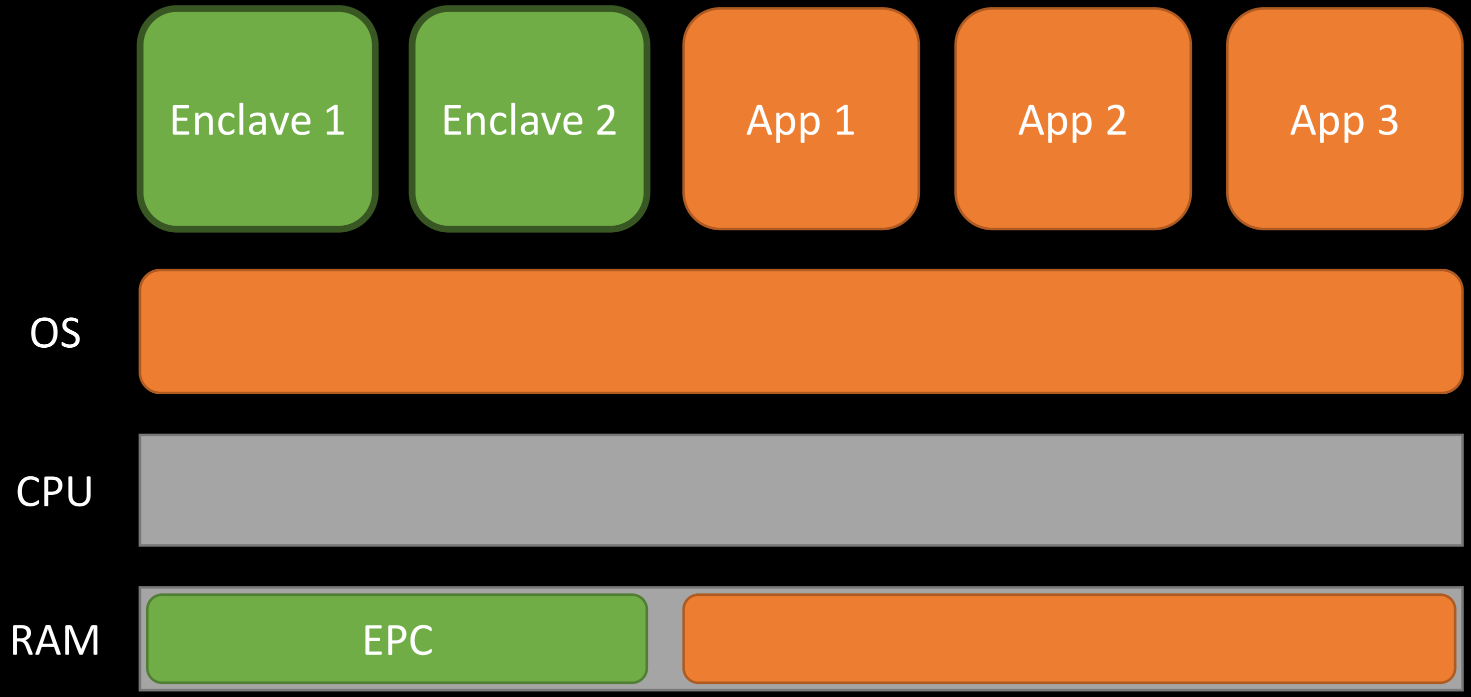


Side-Channel Attack: General Principle



Page Fault Attacks on SGX

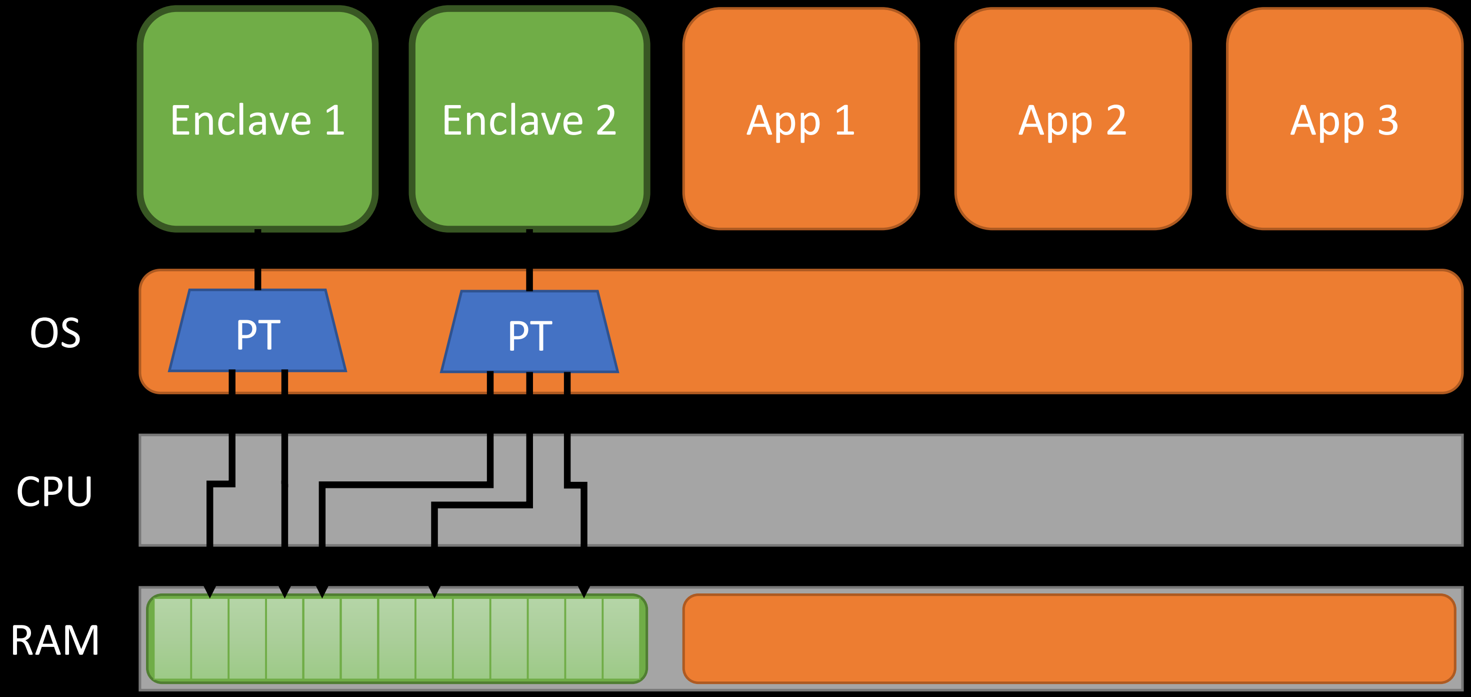
Granularity: page 4K, good for big data structures



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

Page Fault Attacks on SGX

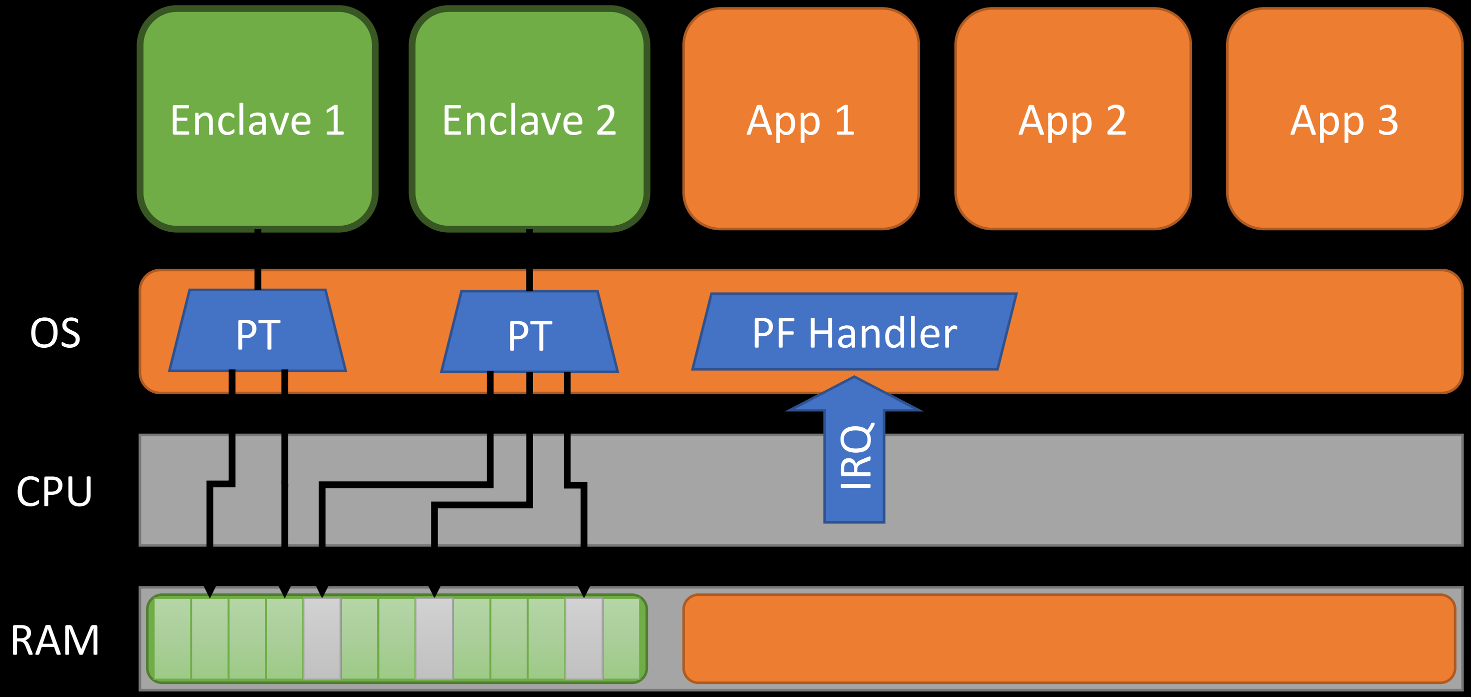
Granularity: page 4K, good for big data structures



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

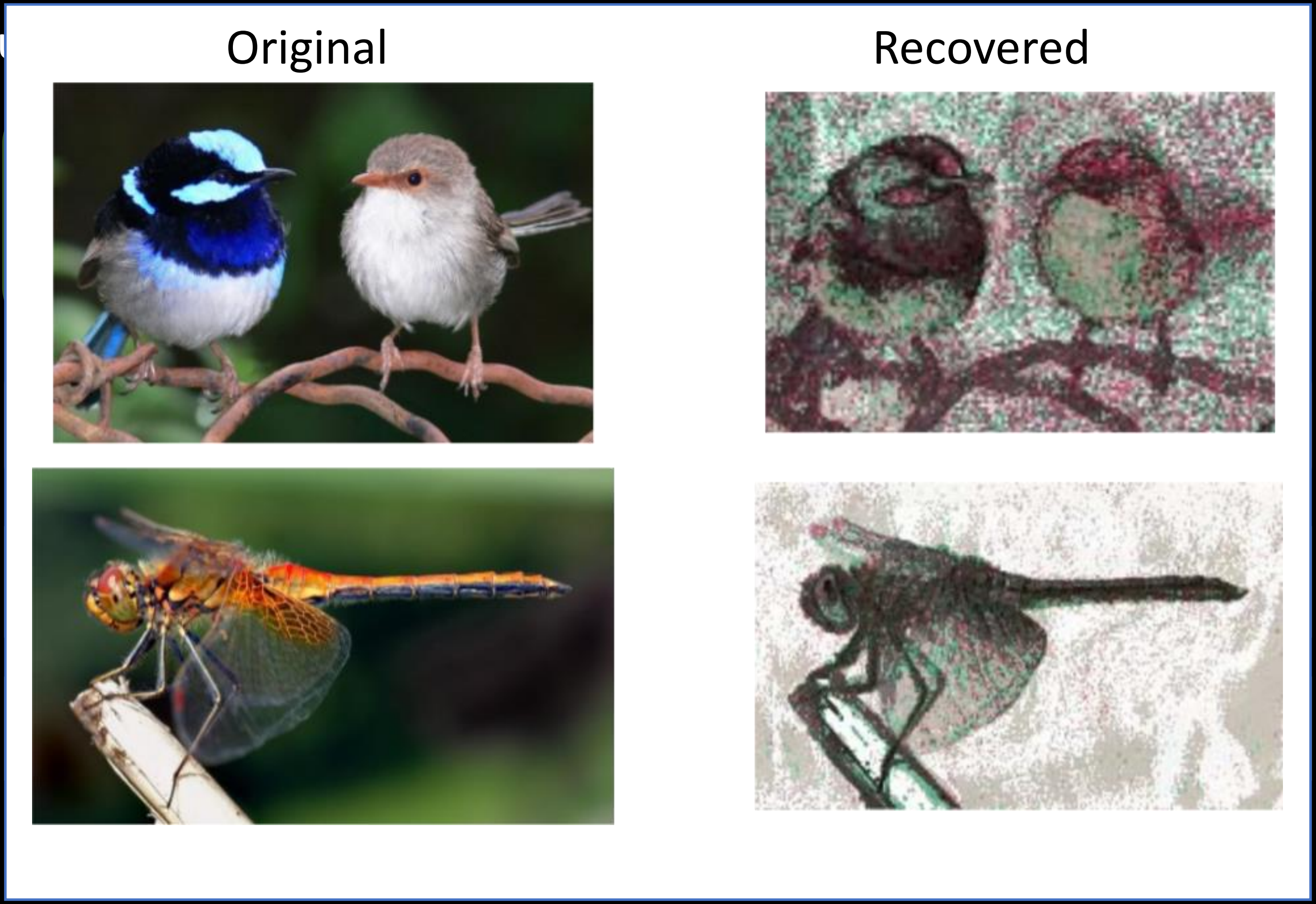
Page Fault Attacks on SGX

Granu

OS

CPU

RAM



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

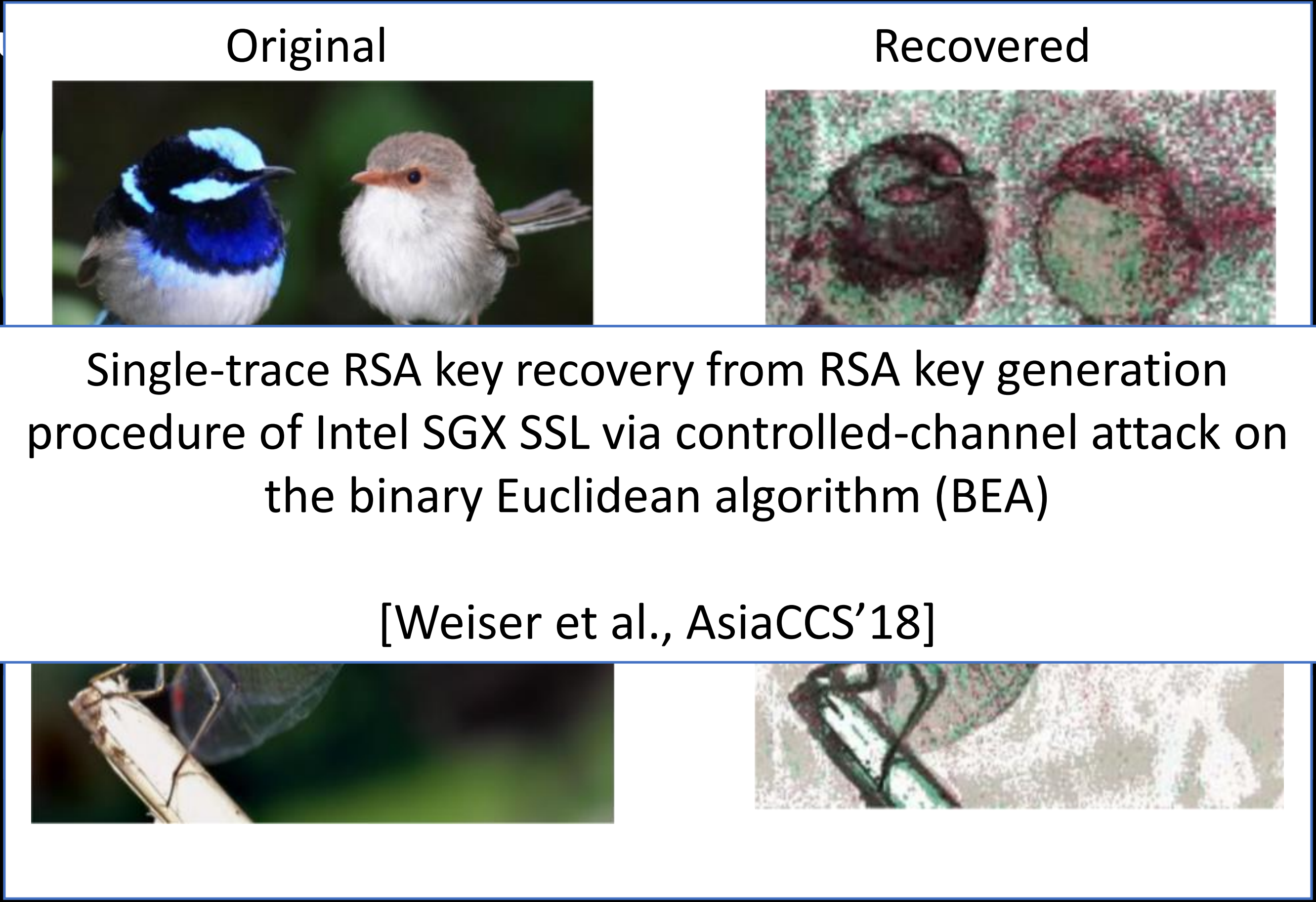
Page Fault Attacks on SGX

Granu

OS

CPU

RAM

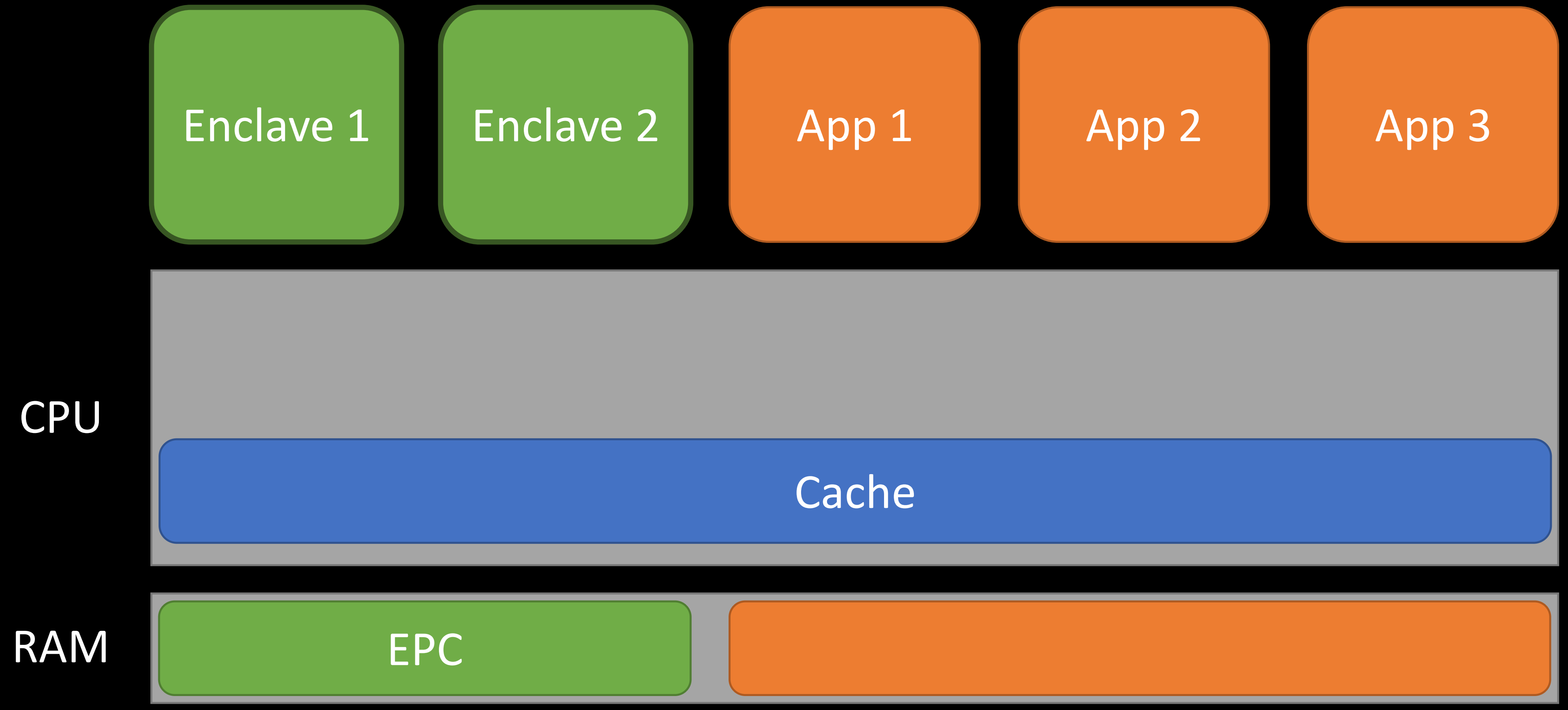


Single-trace RSA key recovery from RSA key generation procedure of Intel SGX SSL via controlled-channel attack on the binary Euclidean algorithm (BEA)

[Weiser et al., AsiaCCS'18]

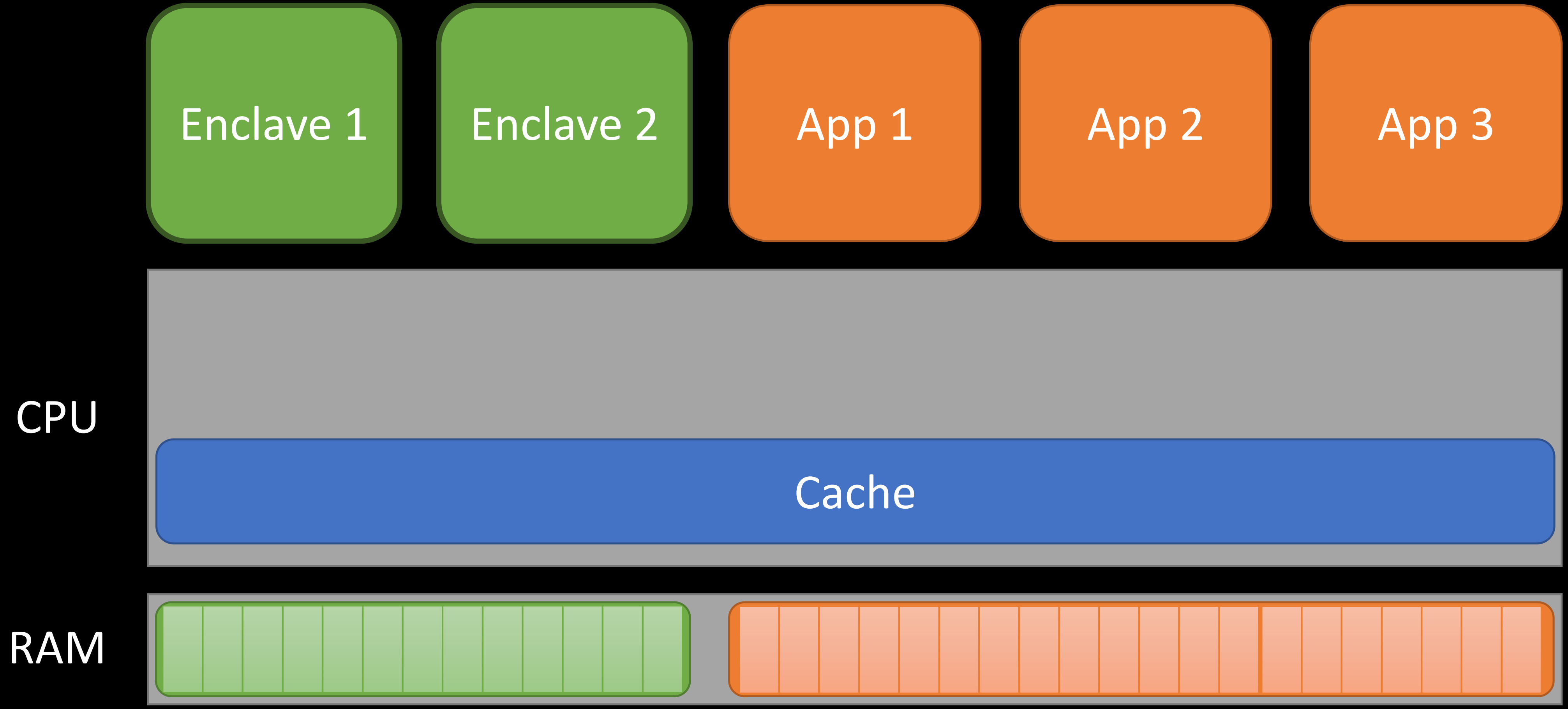
EPC: Enclave Page Cache PT: Page Tables PF: Page-Fault

Cache Attacks on SGX: Hack in The Box



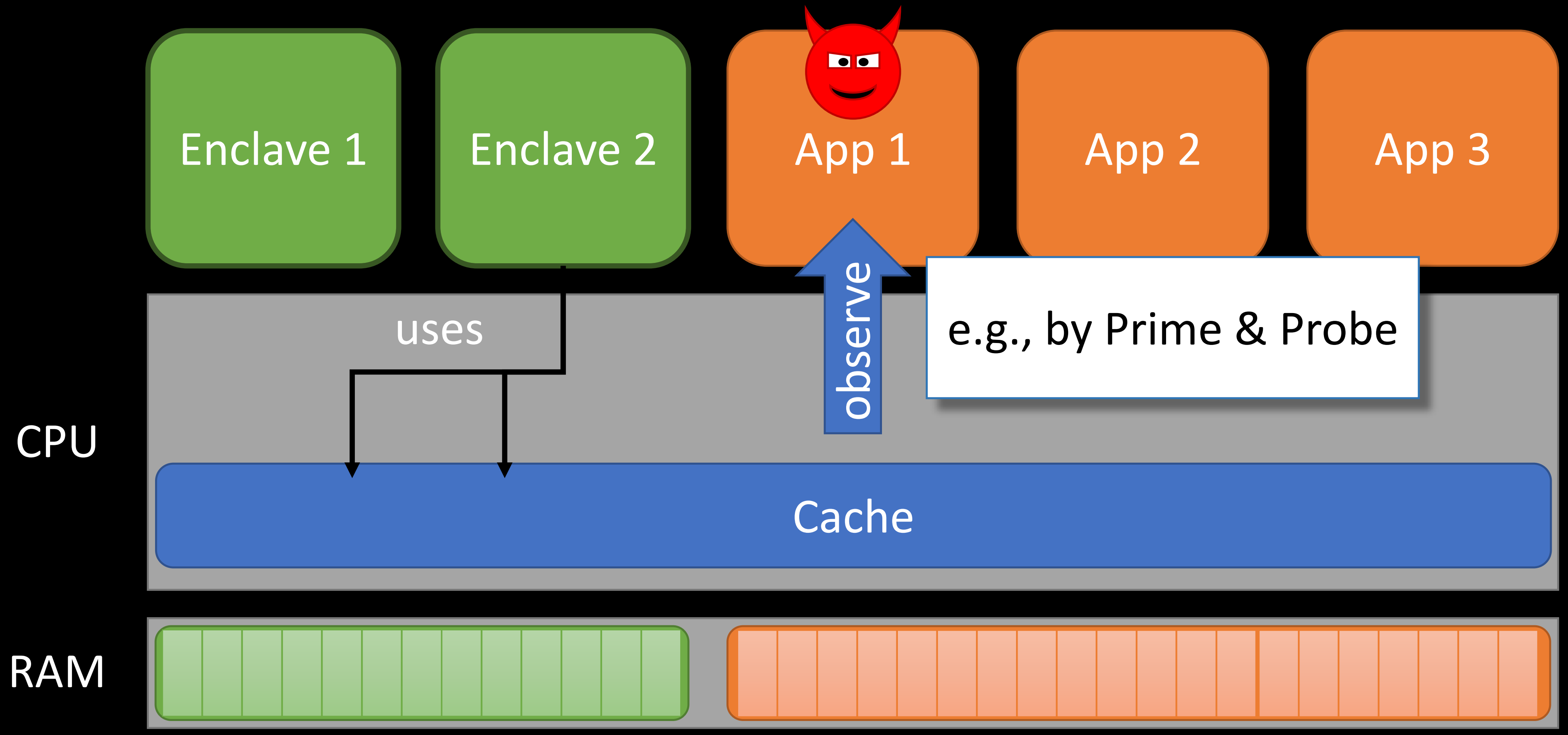
EPC: Enclave Page Cache

Cache Attacks on SGX: Hack in The Box



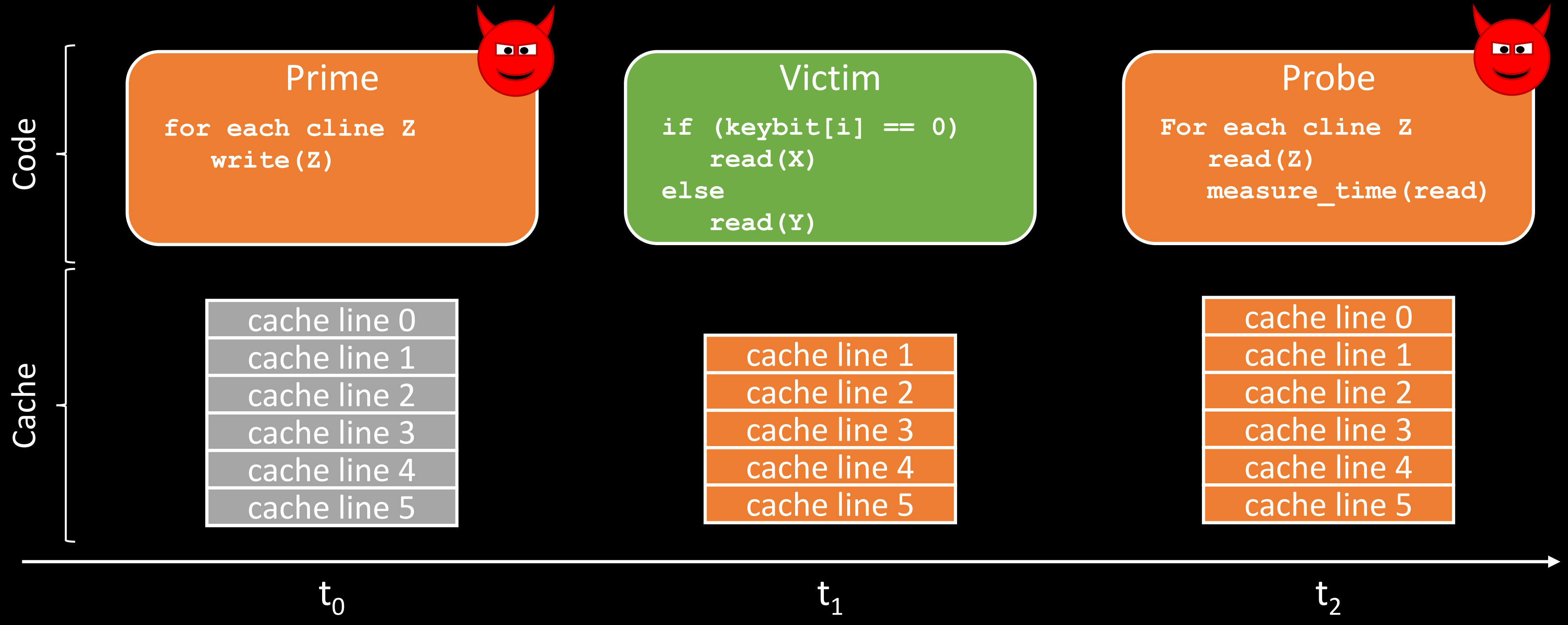
EPC: Enclave Page Cache

Cache Attacks on SGX: Hack in The Box



EPC: Enclave Page Cache

Prime + Probe

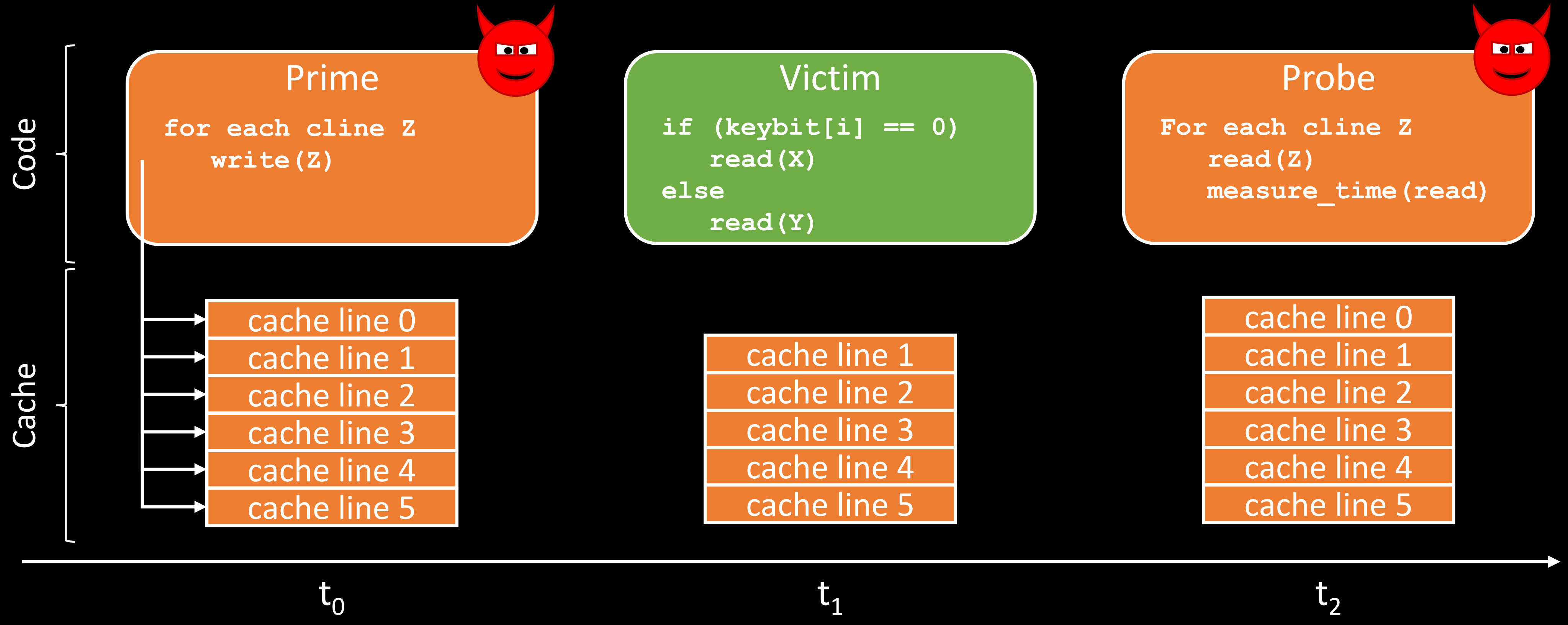


```
for each cline Z
   write(Z)
```

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

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

Prime + Probe



Prime

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

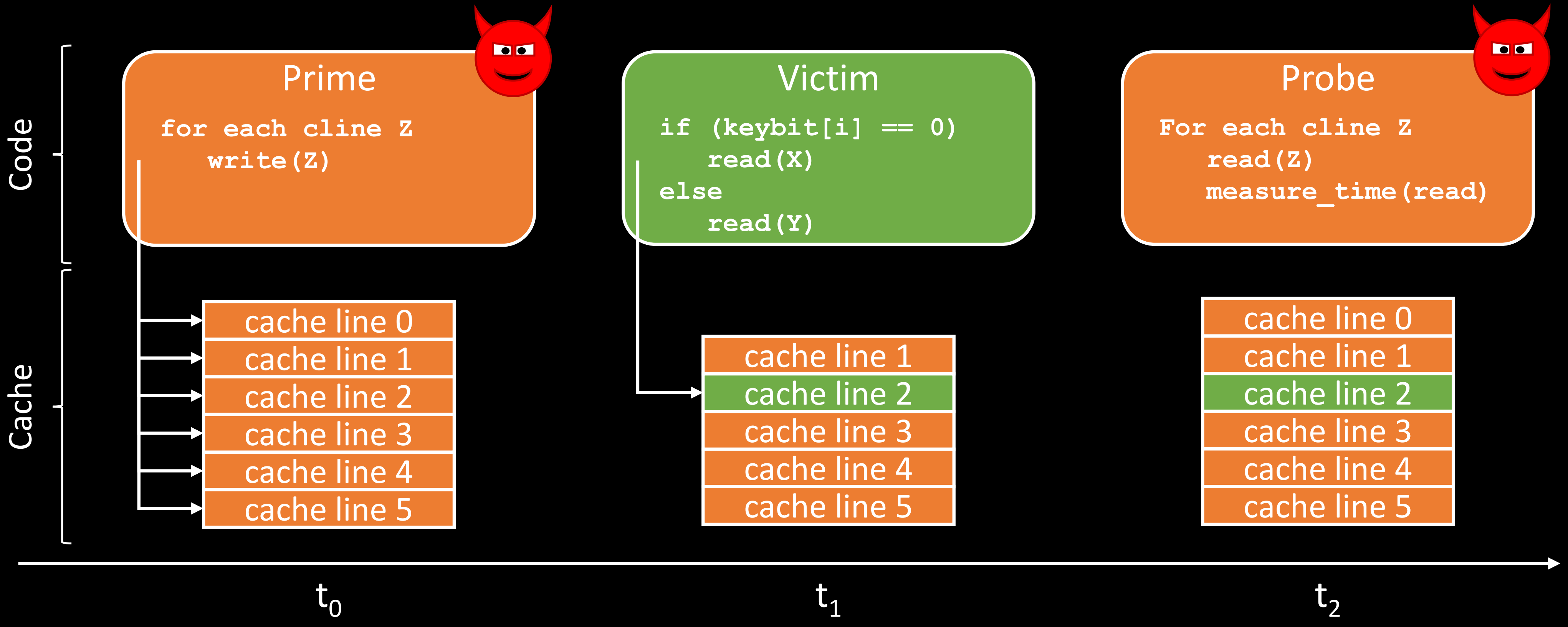
Victim

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

Probe

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

Prime + Probe



Prime:

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

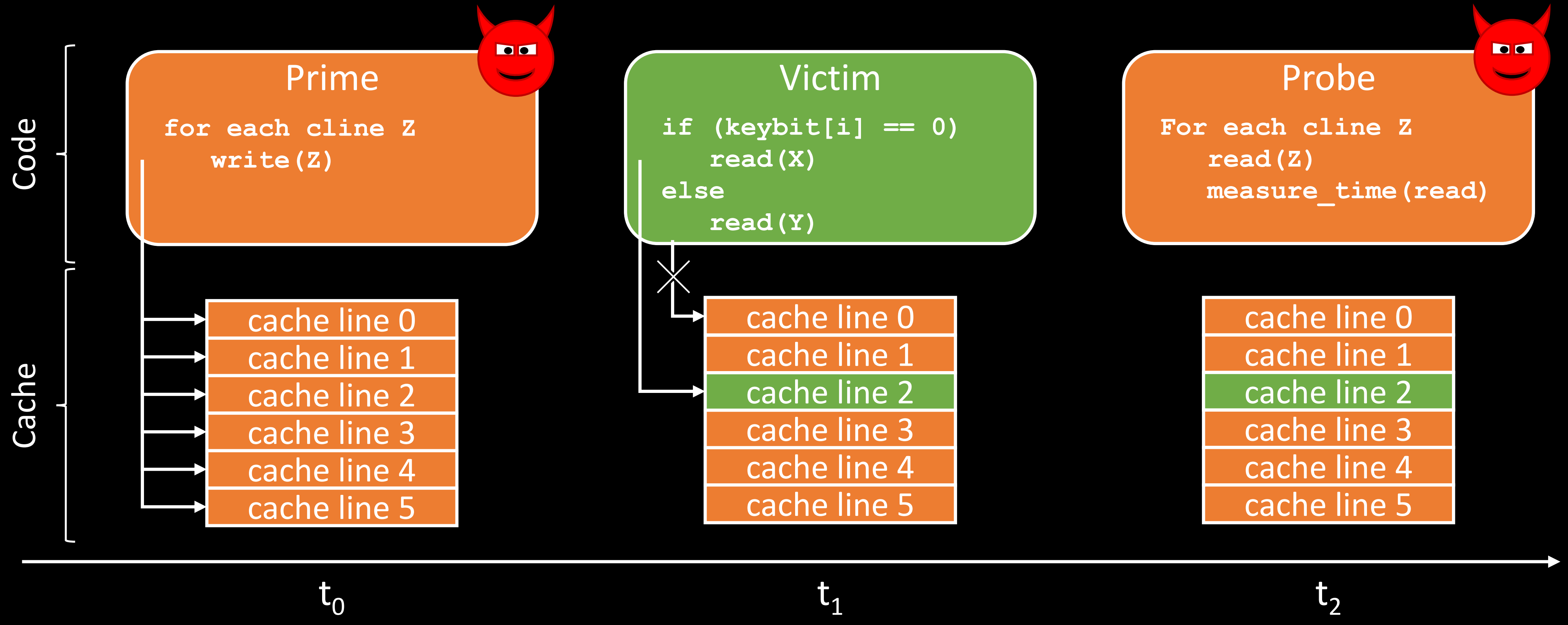
Victim:

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

Probe:

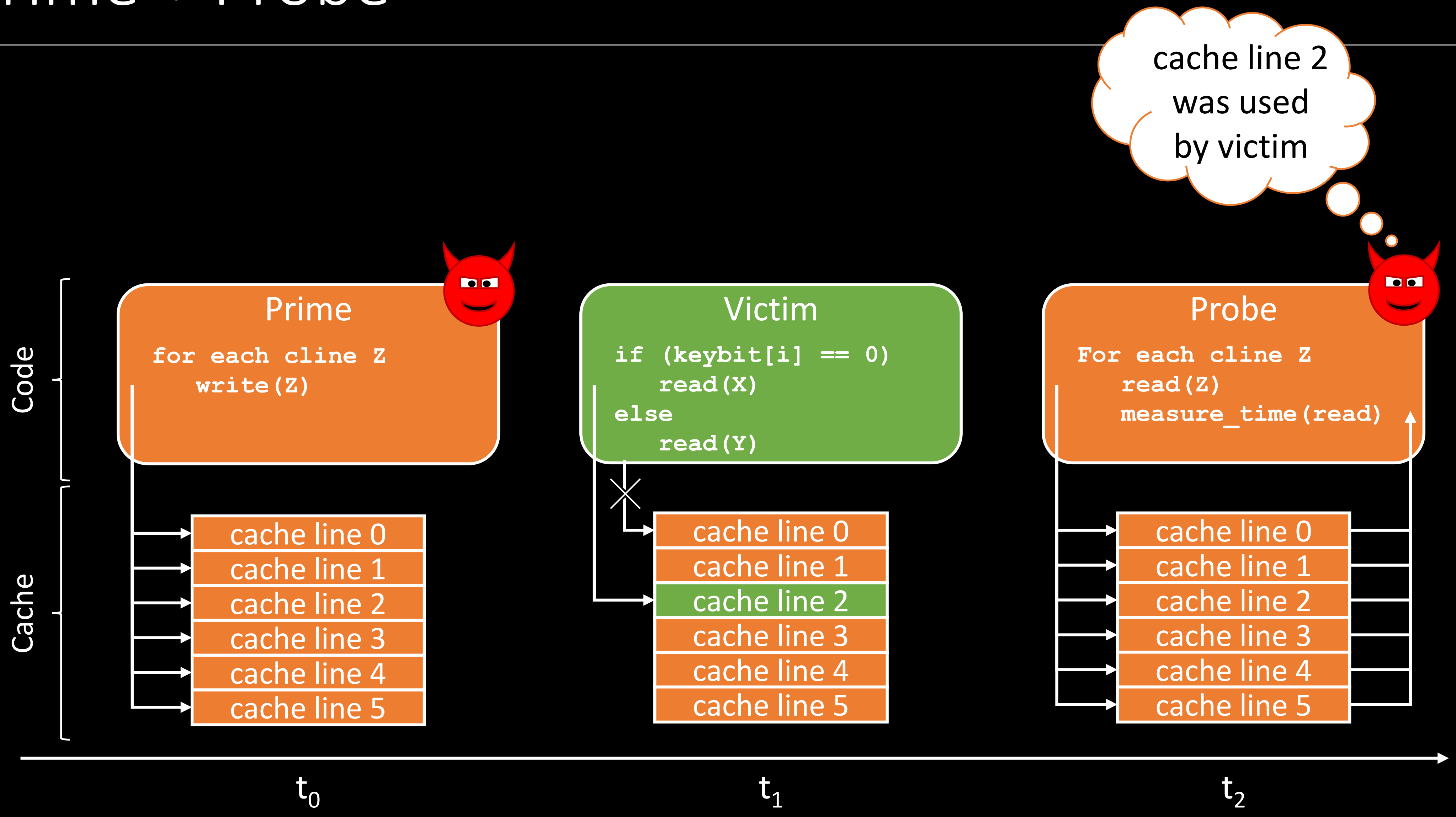
```
For each cline Z
   read(Z)
   measure_time(read)
```

Prime + Probe



Code
Cache
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)
cache line 0
cache line 1
cache line 2
cache line 3
cache line 4
cache line 5
t0
t1
t2

Prime + Probe



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)
```

How to measure the time difference?

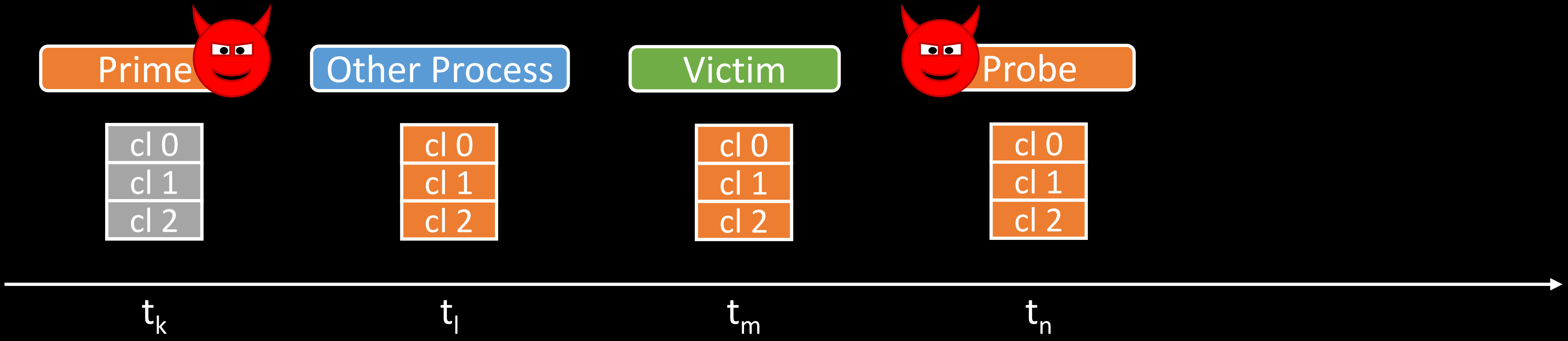
- #1: Time Stamp Counter (TSC)
  - Not precise enough to reliably distinguish the difference between L1 vs. L2 hits
  - Reading the time stamp counter by itself suffers from noise

- #2: Counting thread:
  - a thread that only performs a loop that constantly increments a value (basically a timer)
  - Slows down the victim, can be detected

- #3: Performance Monitoring Counter (PMC):
  - can be configured to count different events: executed cycles, cache hits or cache misses for the different caches, mis-predicted branches, etc.
  - Anti Side-channel Interference (ASCI) feature:
    - Can be configured to disable thread-specific performance monitoring of enclaves

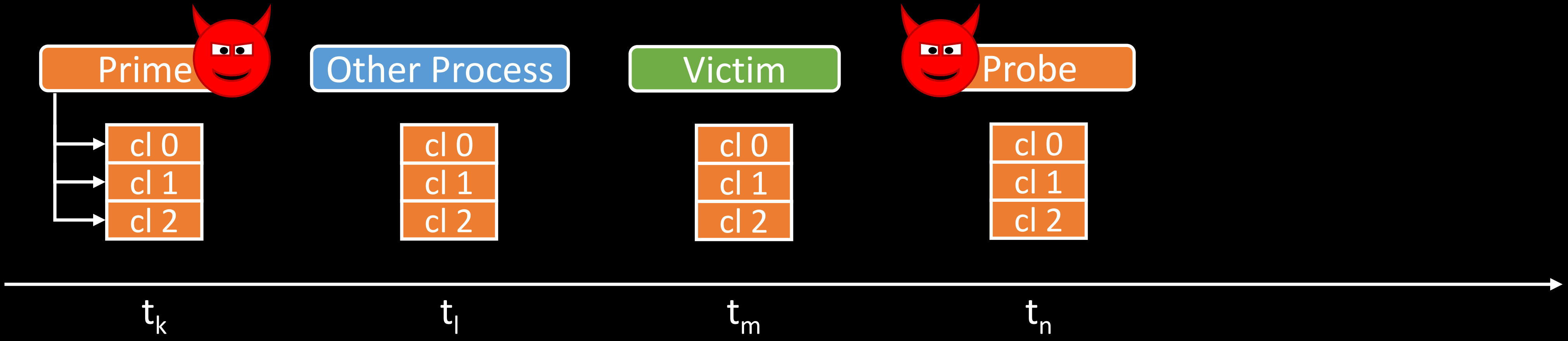
Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker's own code pollutes the cache



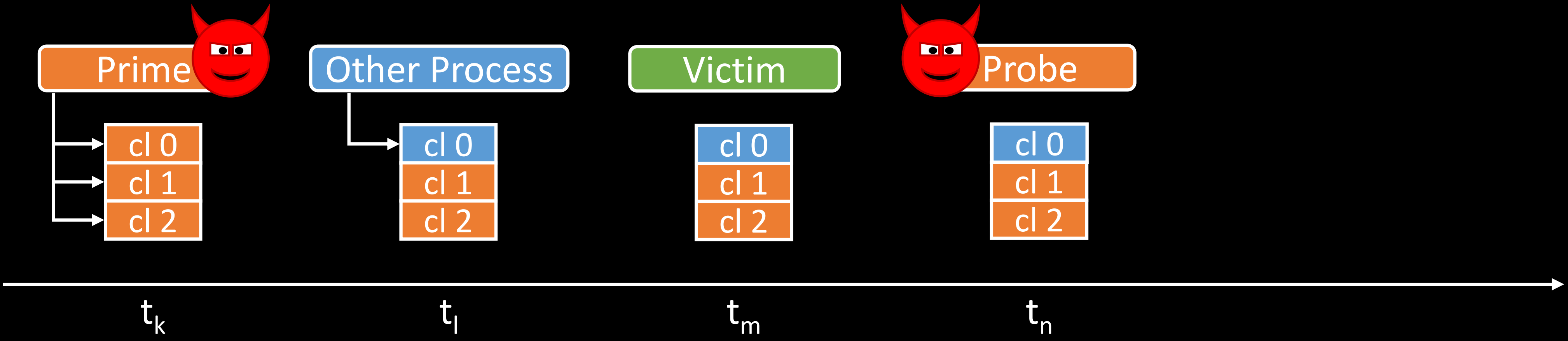
Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker’s own code pollutes the cache



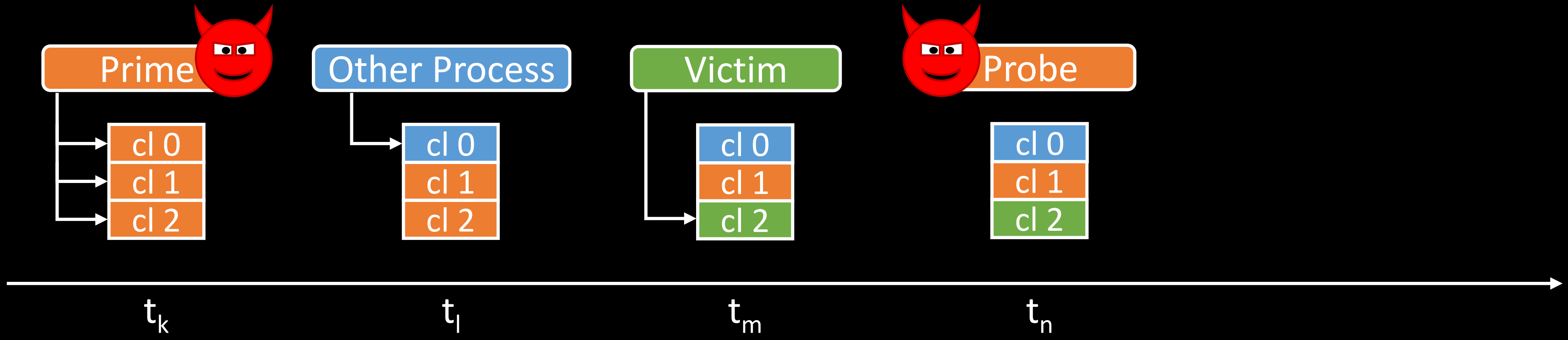
Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker's own code pollutes the cache



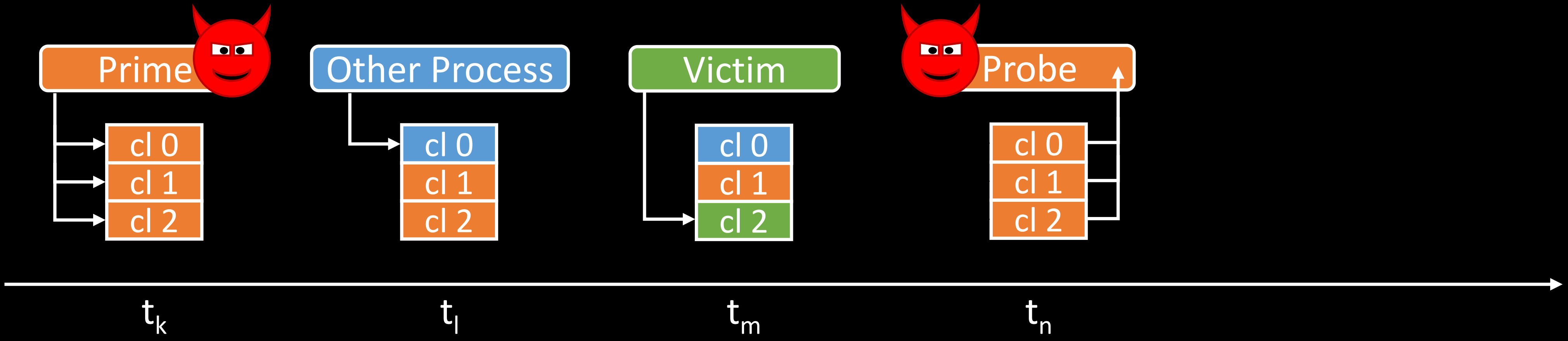
Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker's own code pollutes the cache



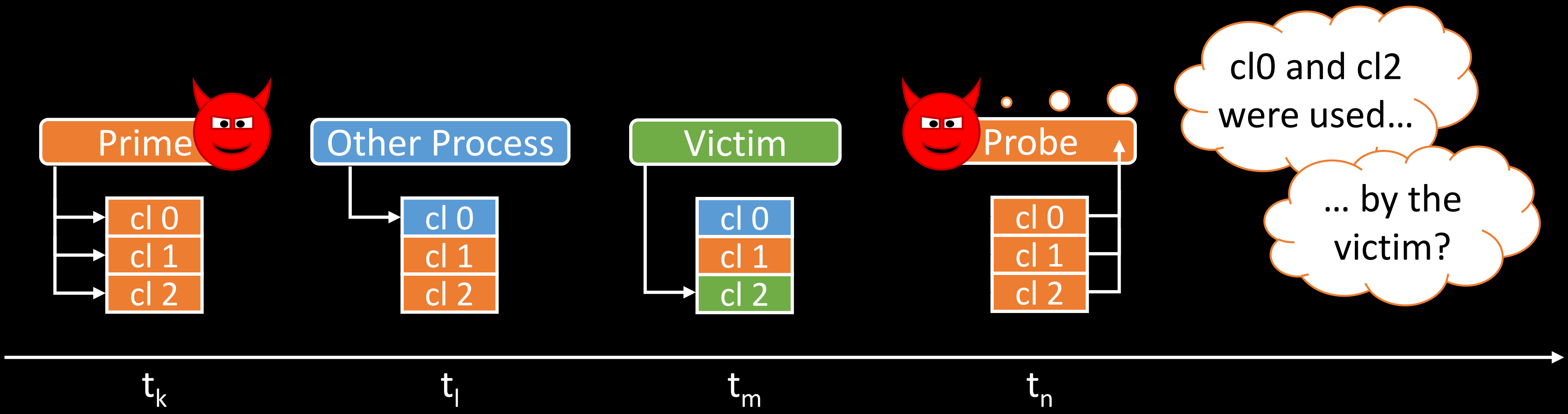
Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker's own code pollutes the cache

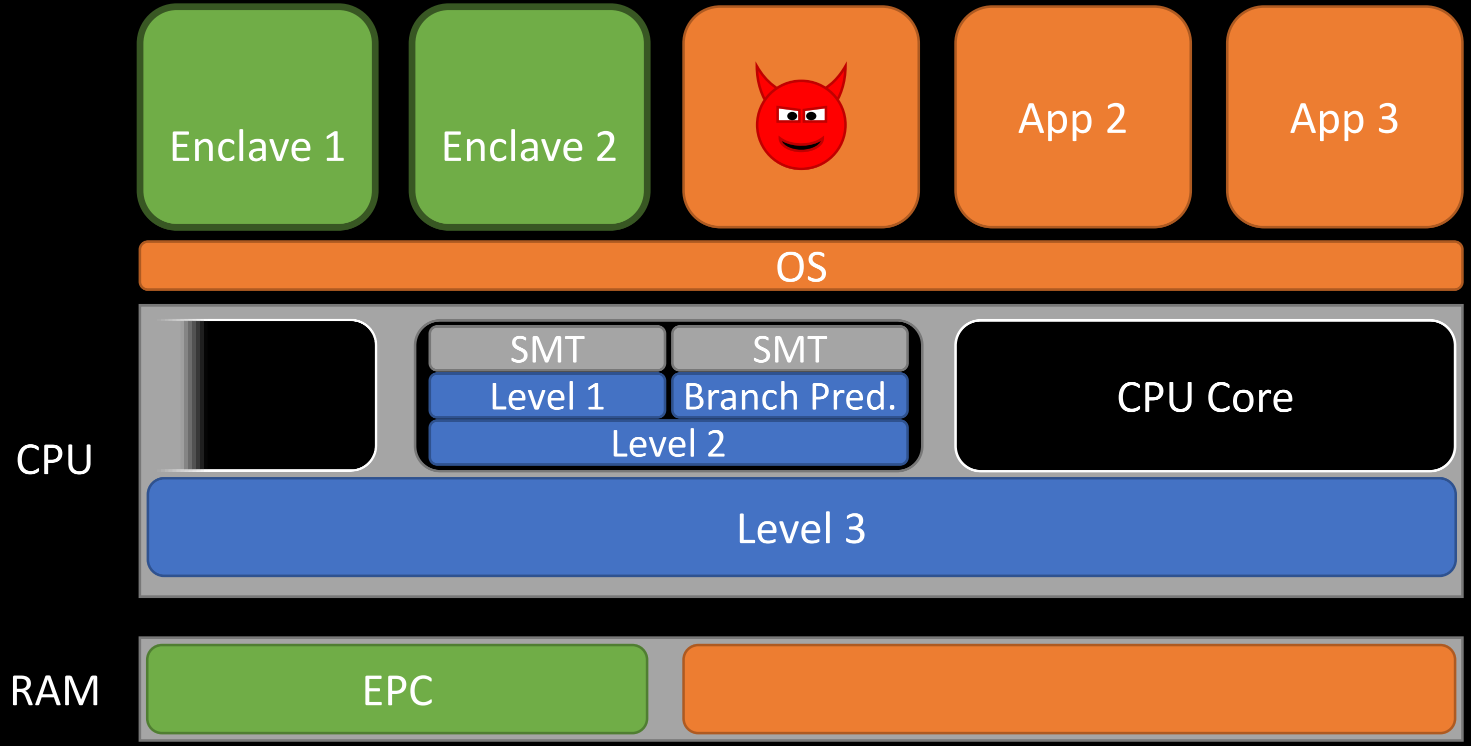


Side-Channel Grand Challenge: Noise

- Operating System and any other software running on the platform generate noise
- Even attacker's own code pollutes the cache

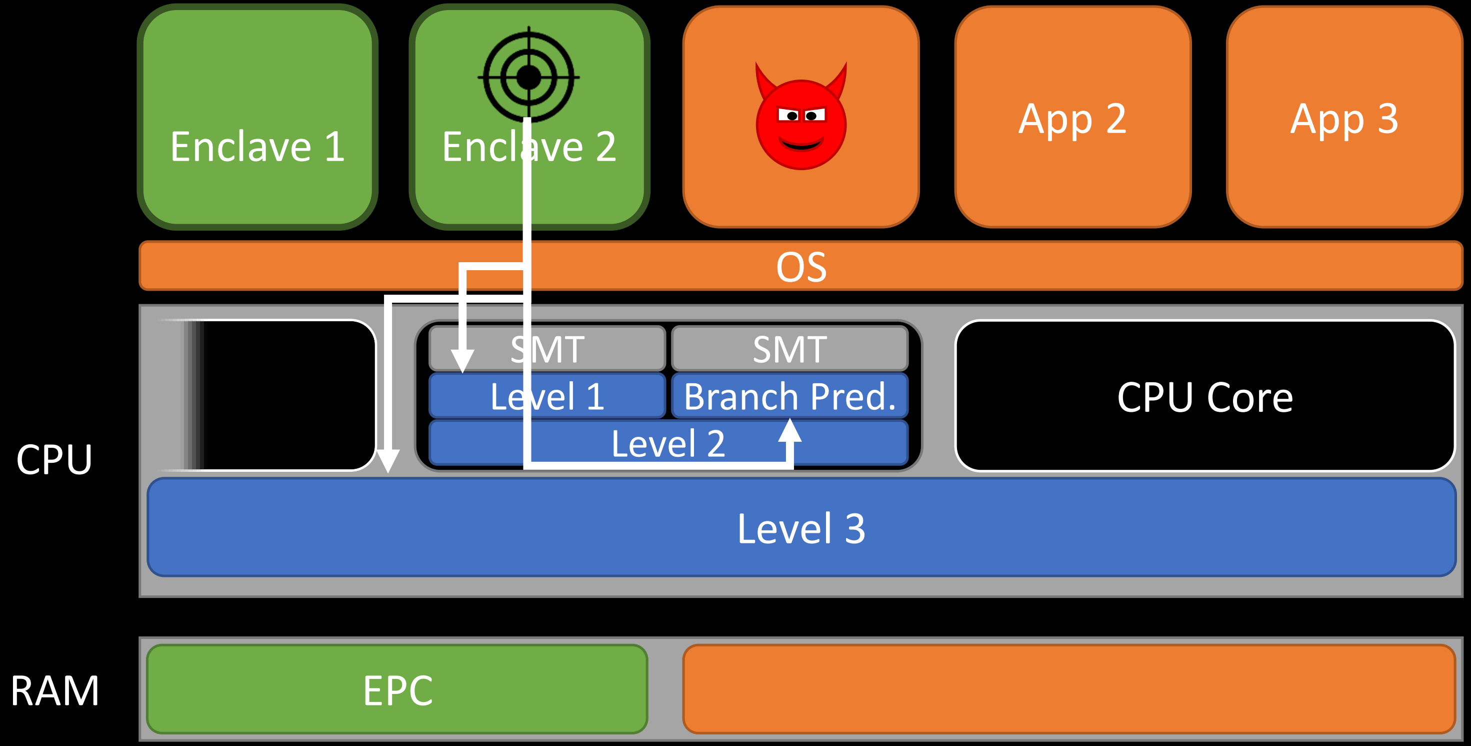


Cache Attacks on SGX



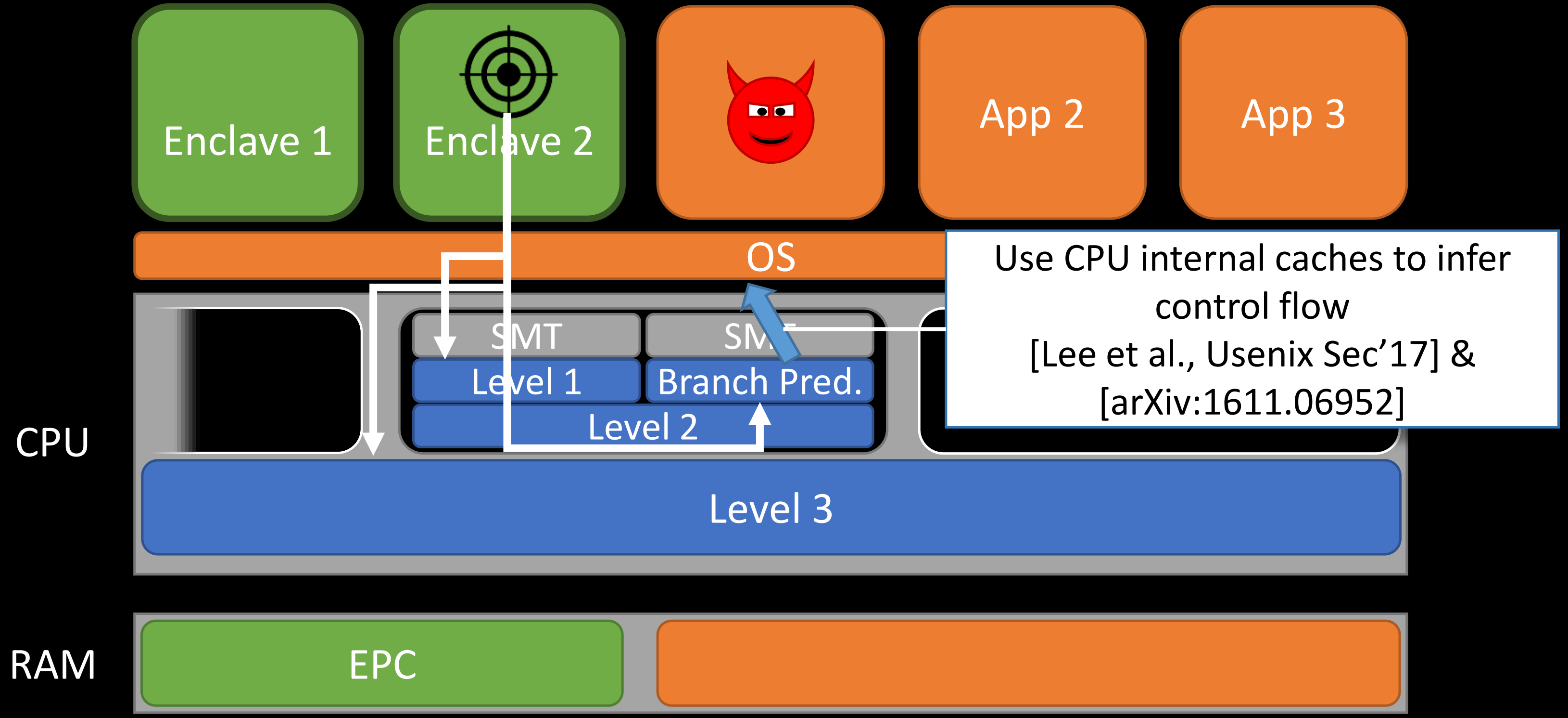
EPC: Enclave Page Cache SMT: Simultaneous Multithreading

Cache Attacks on SGX



EPC: Enclave Page Cache SMT: Simultaneous Multithreading

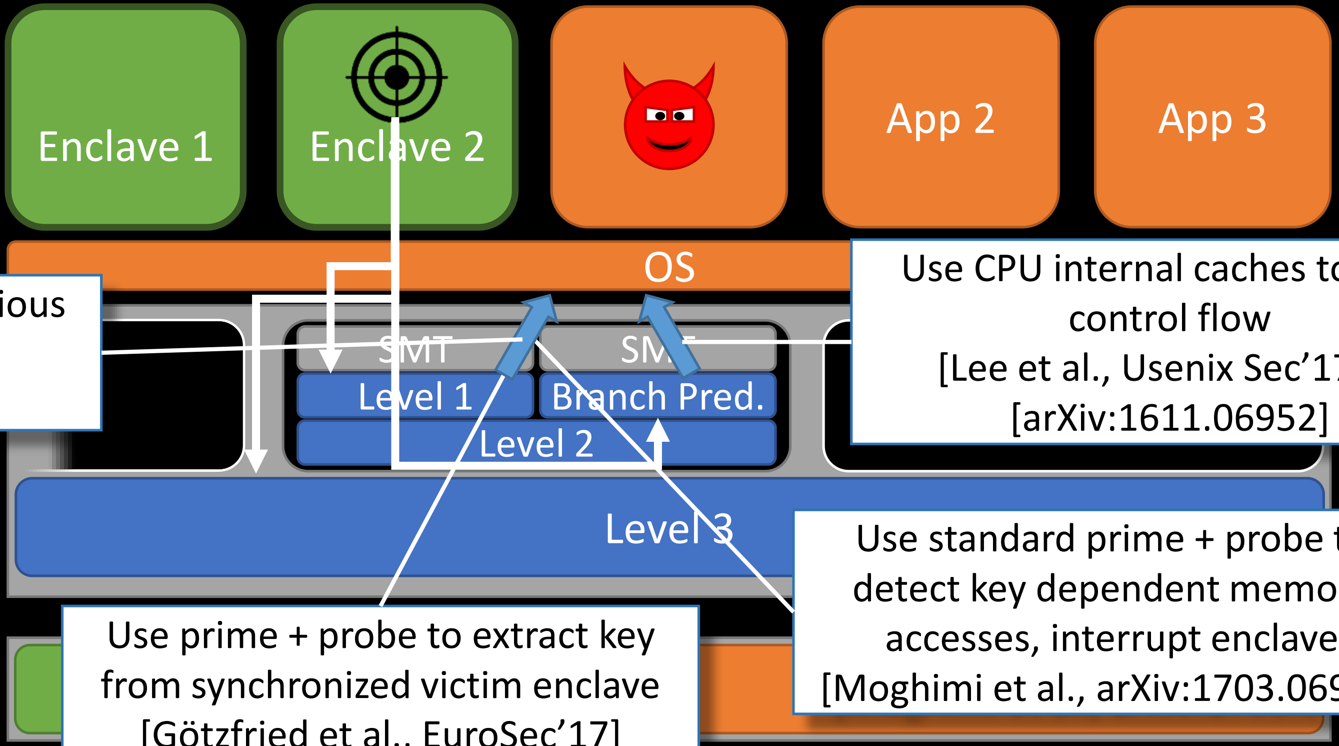
Cache Attacks on SGX



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



Prime + probe attack from malicious OS extracting genome data [Brasser et al., WOOT'17]

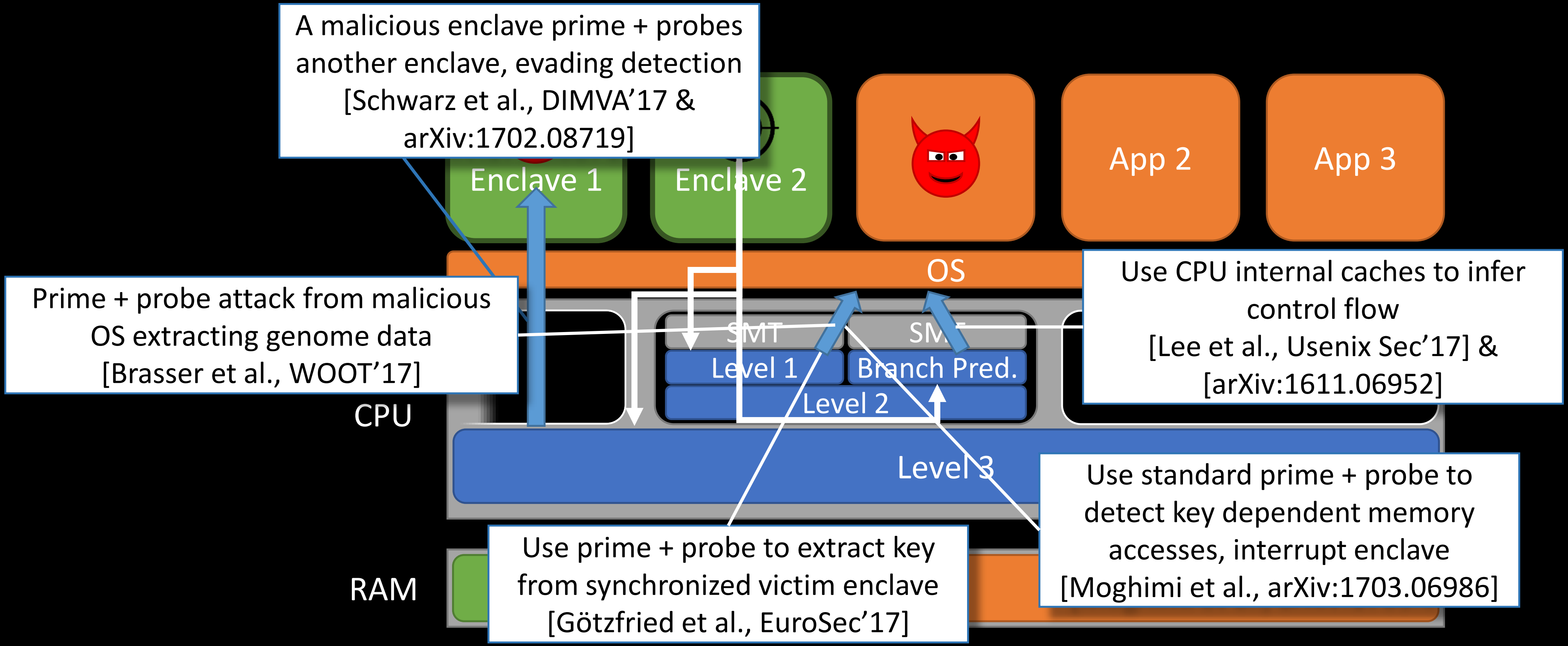
Use CPU internal caches to infer control flow [Lee et al., Usenix Sec'17] & [arXiv:1611.06952]

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]

Prime + probe attack from malicious OS extracting genome data [Brasser et al., WOOT'17]

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]

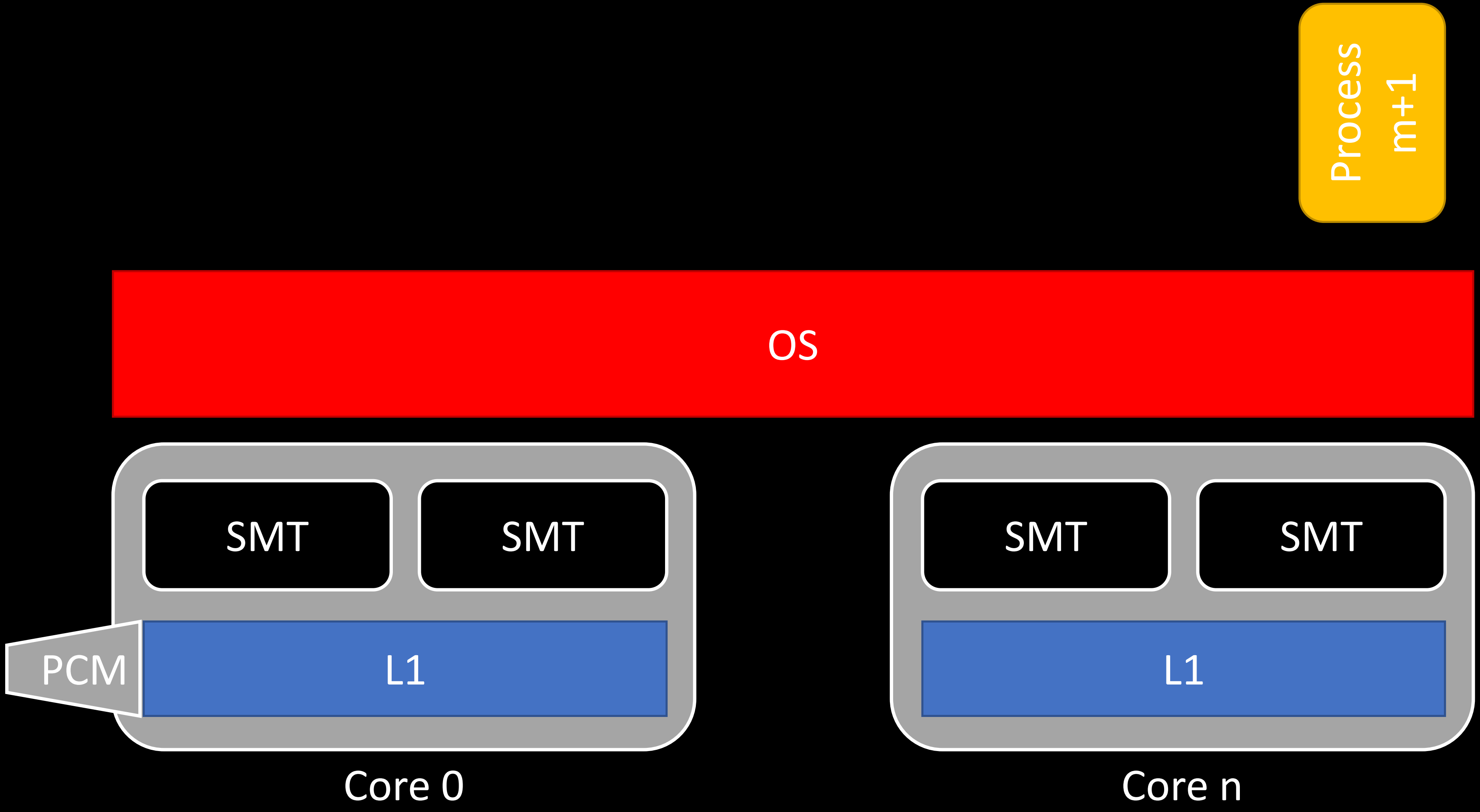
EPC: Enclave Page Cache SMT: Simultaneous Multithreading

SGX Side-Channel Attacks Comparison

	Attack Type	Observed Cache	Interrupting Victim	Time 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	TCS	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

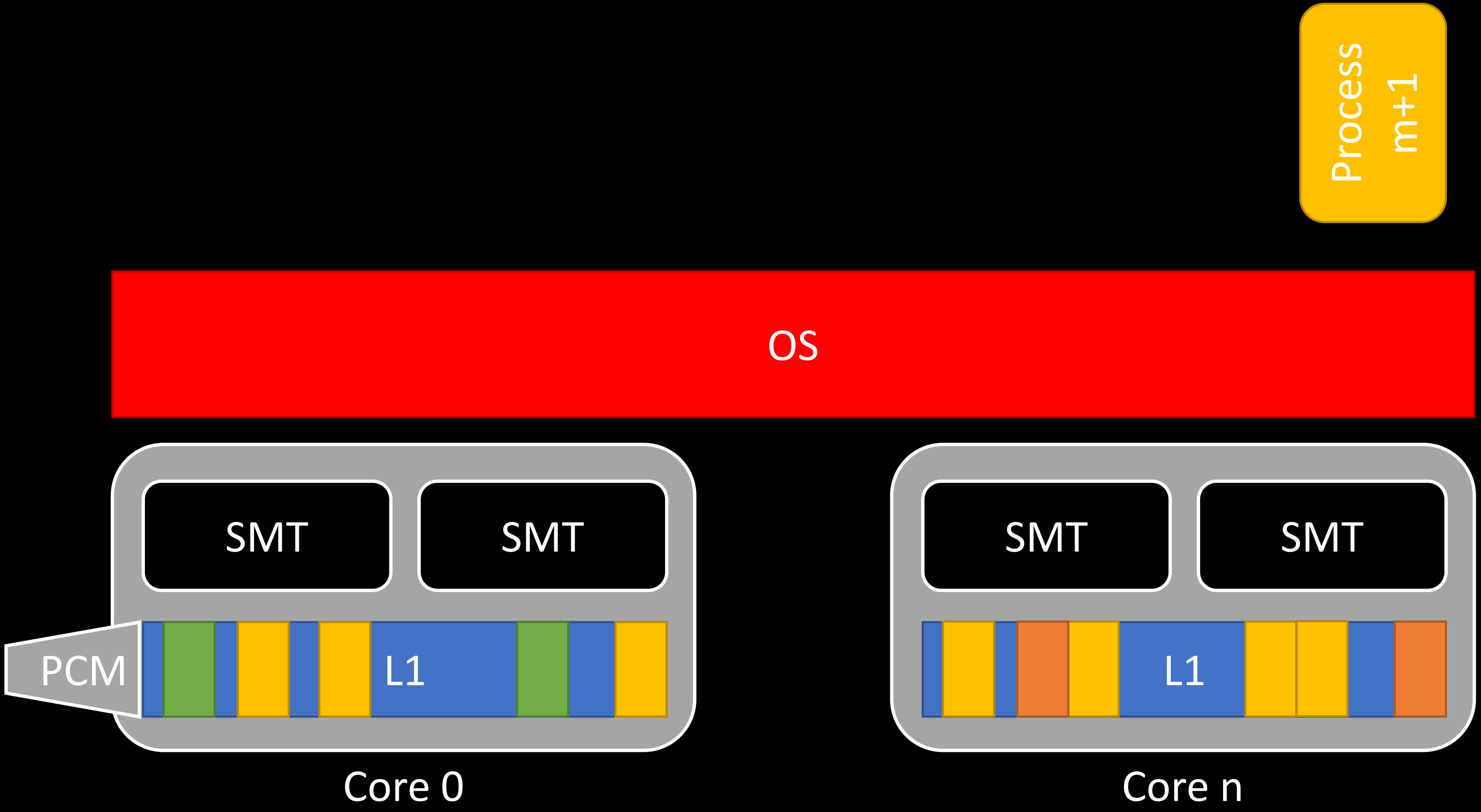
PCM: Performance Counter Monitor BTB: Branch Target Buffer LBR: Last Branch Record TSC: Time Stamp Counter

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



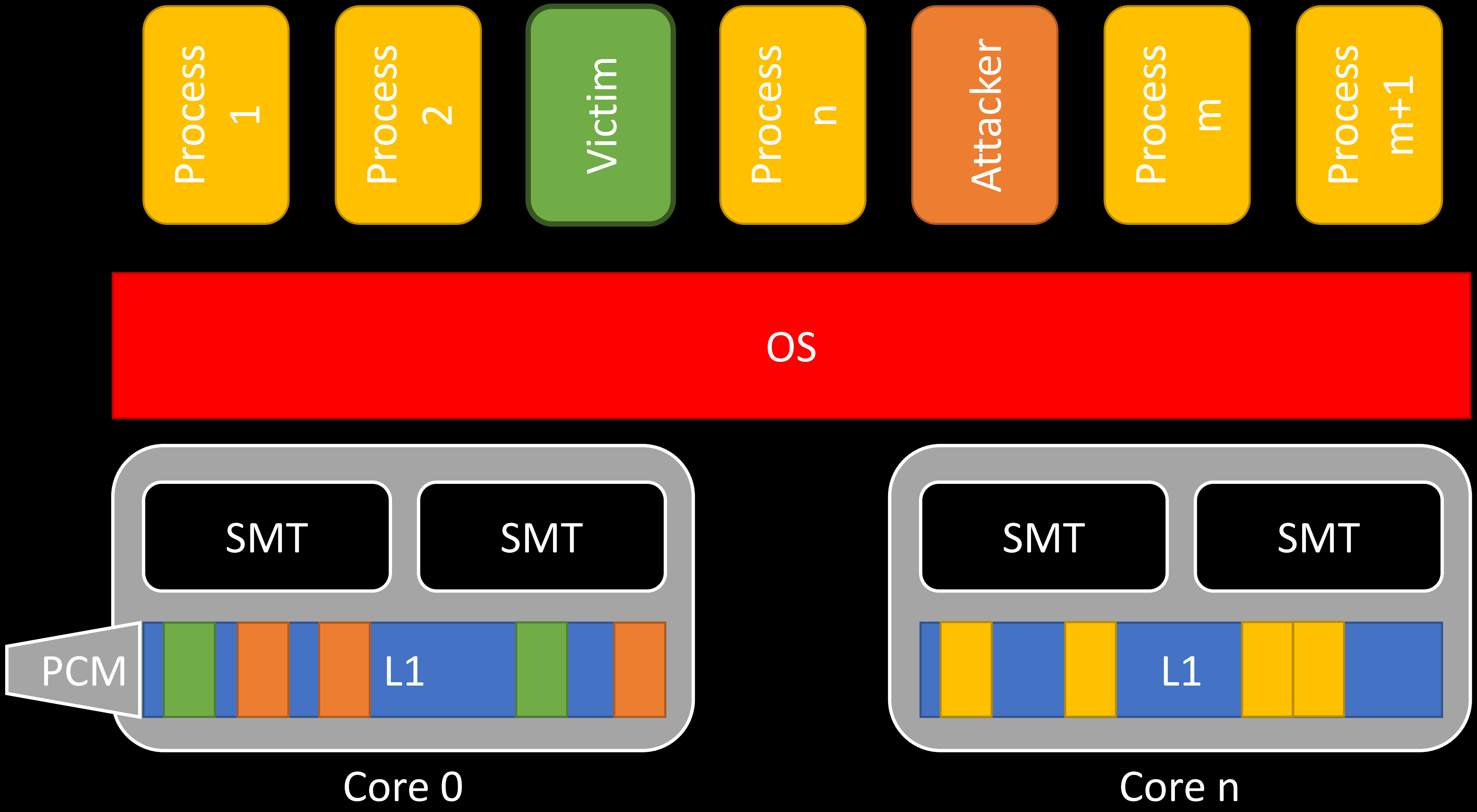
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

Our Attack [Brasser et al., WOOT’17]



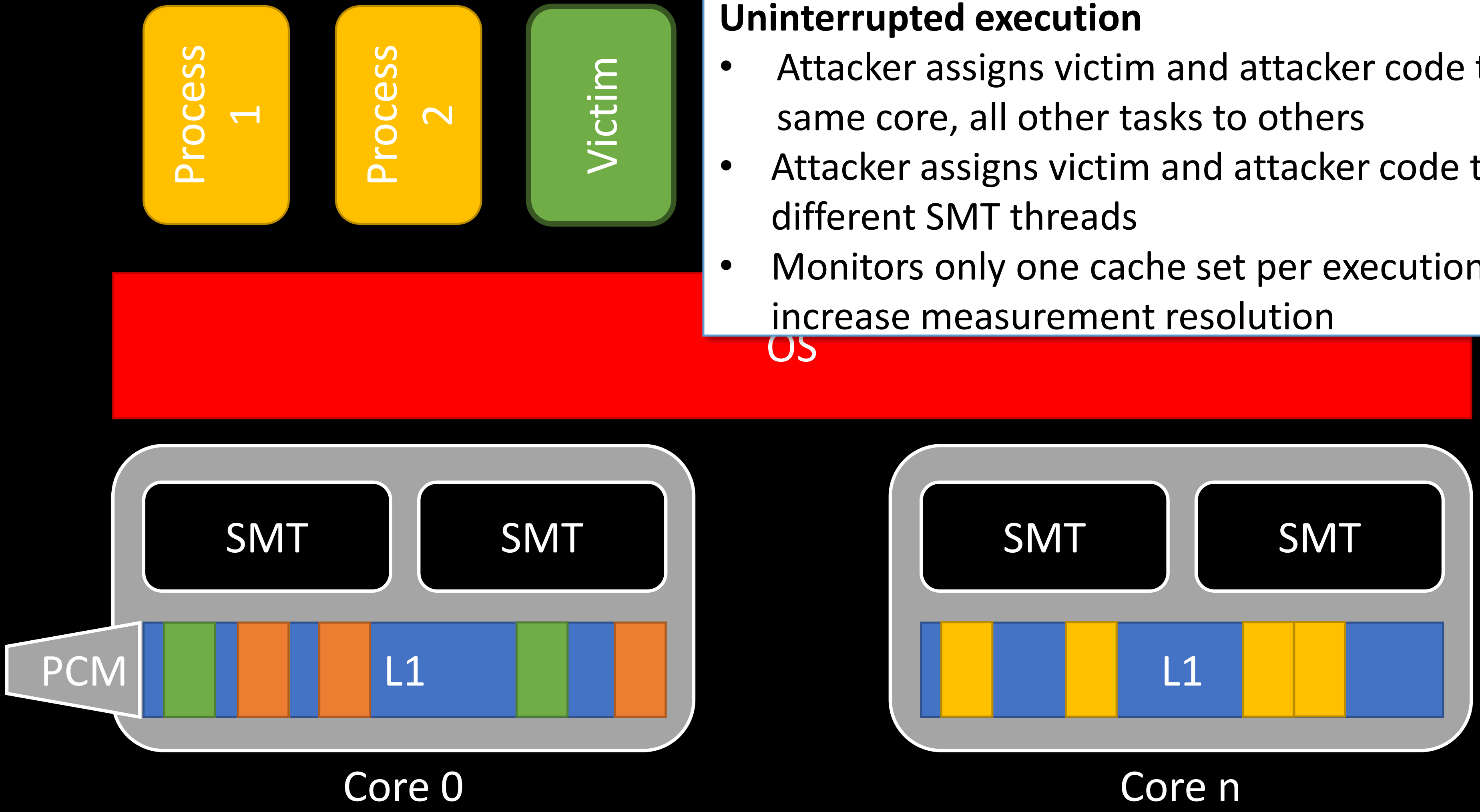
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

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]

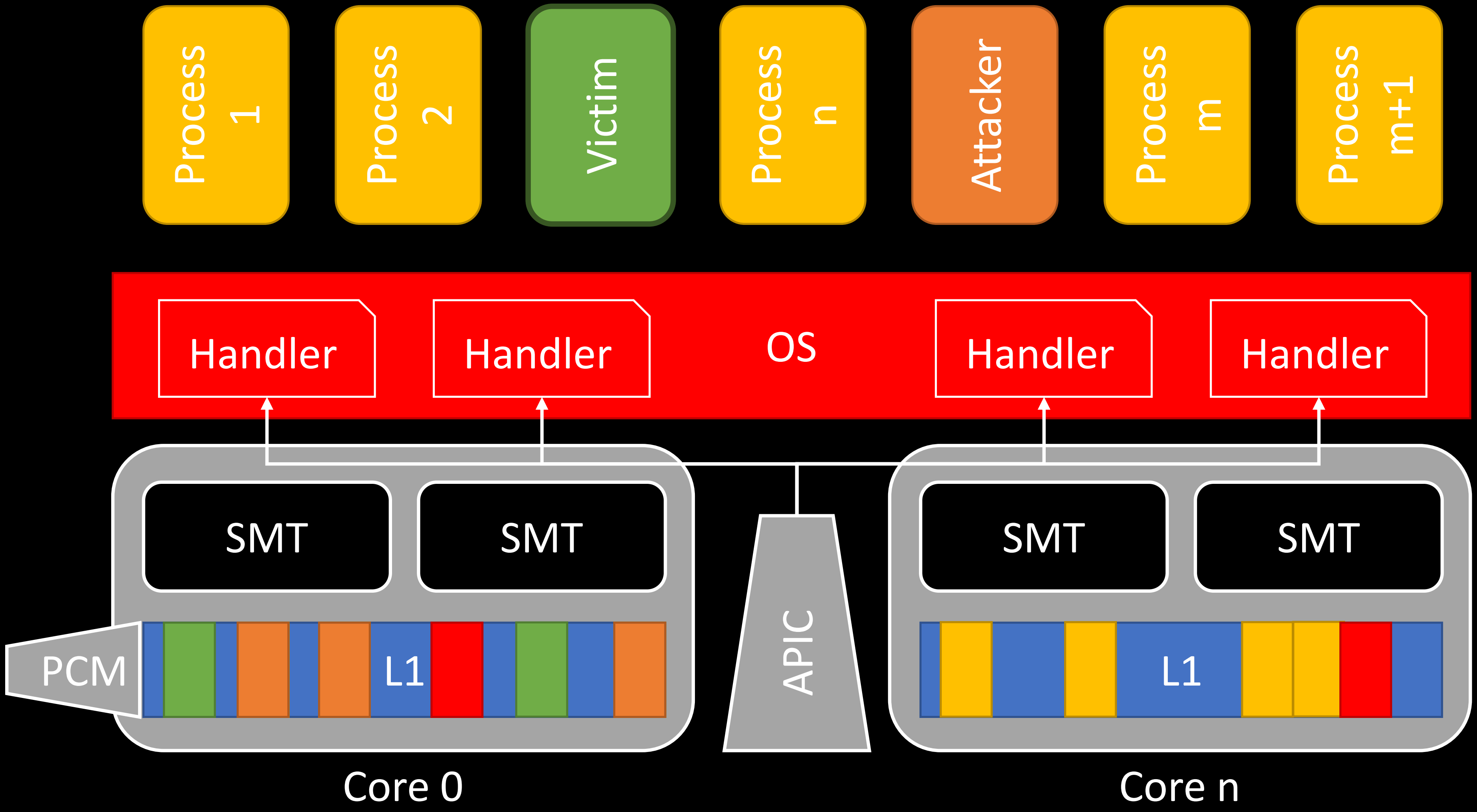


Uninterrupted execution

- Attacker assigns victim and attacker code to the same core, all other tasks to others
- Attacker assigns victim and attacker code to different SMT threads
- Monitors only one cache set per execution to increase measurement resolution

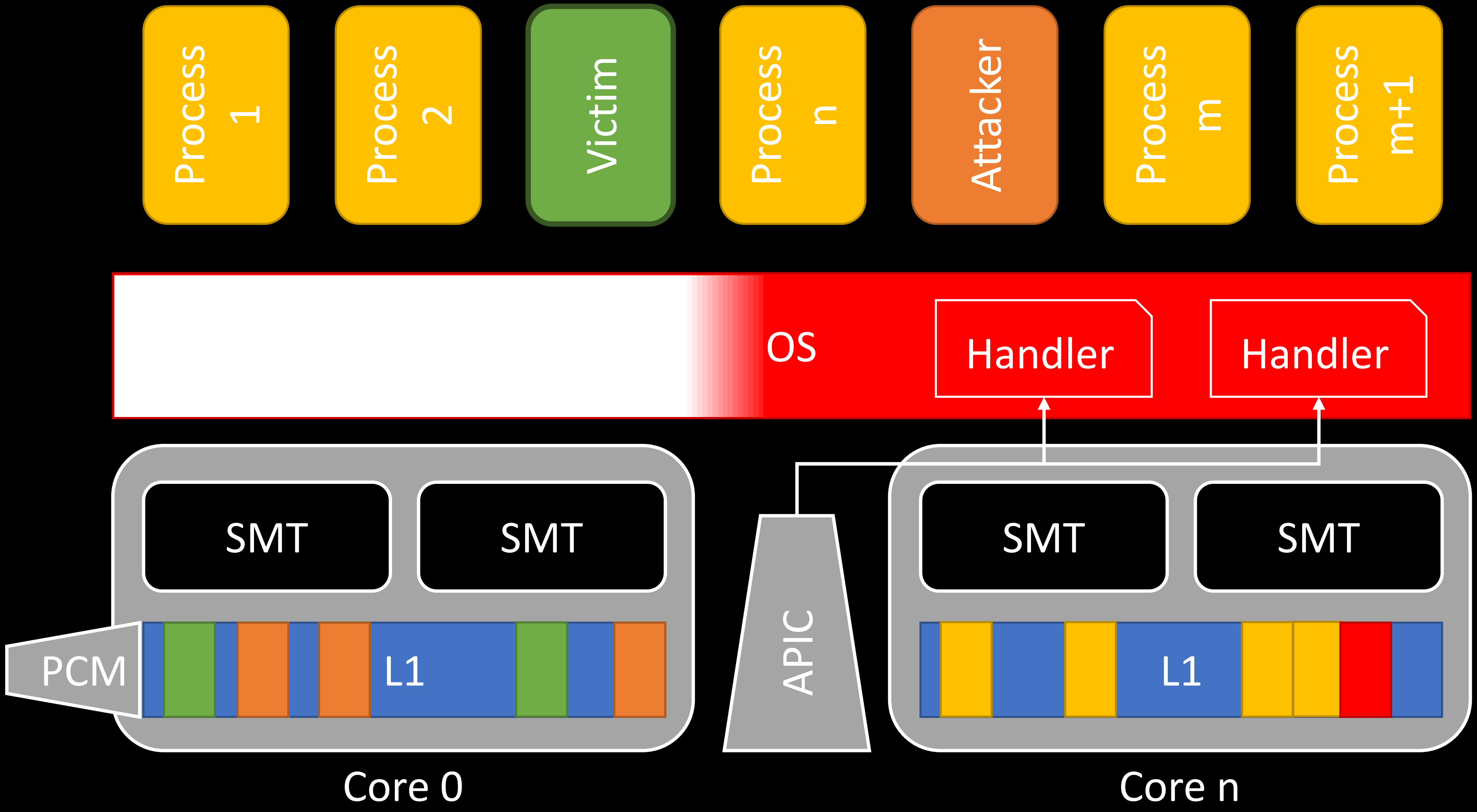
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

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



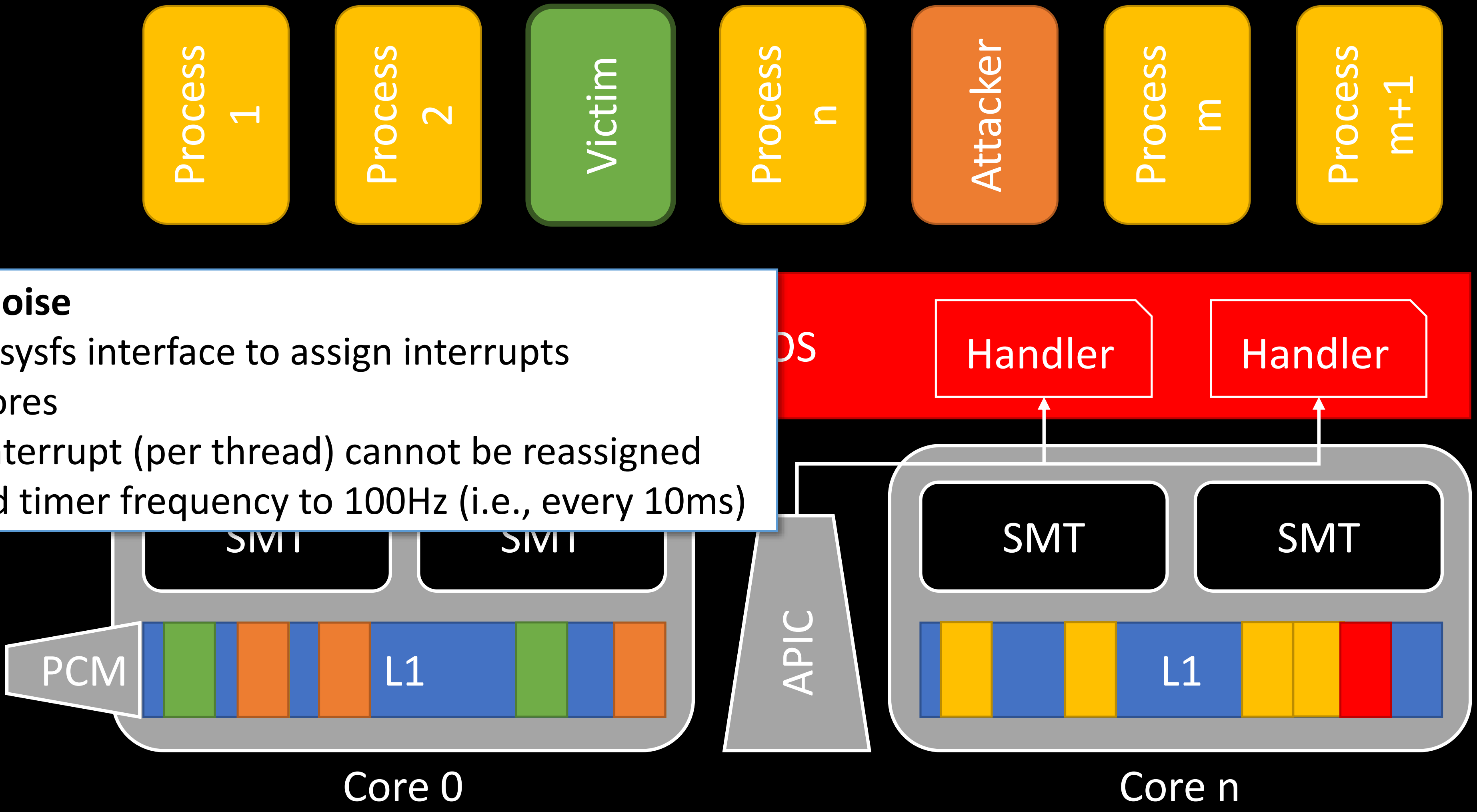
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

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]



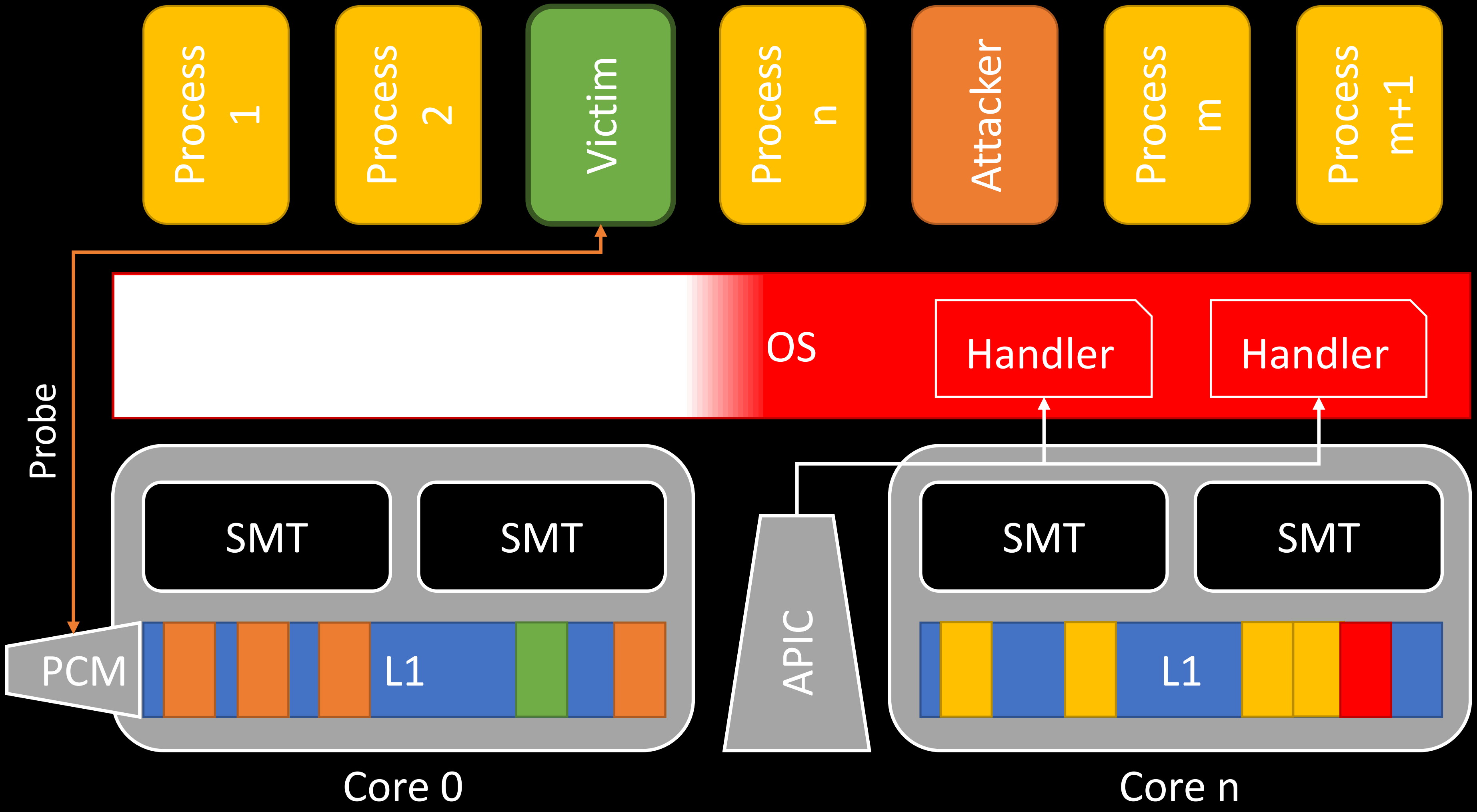
Reducing noise

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)

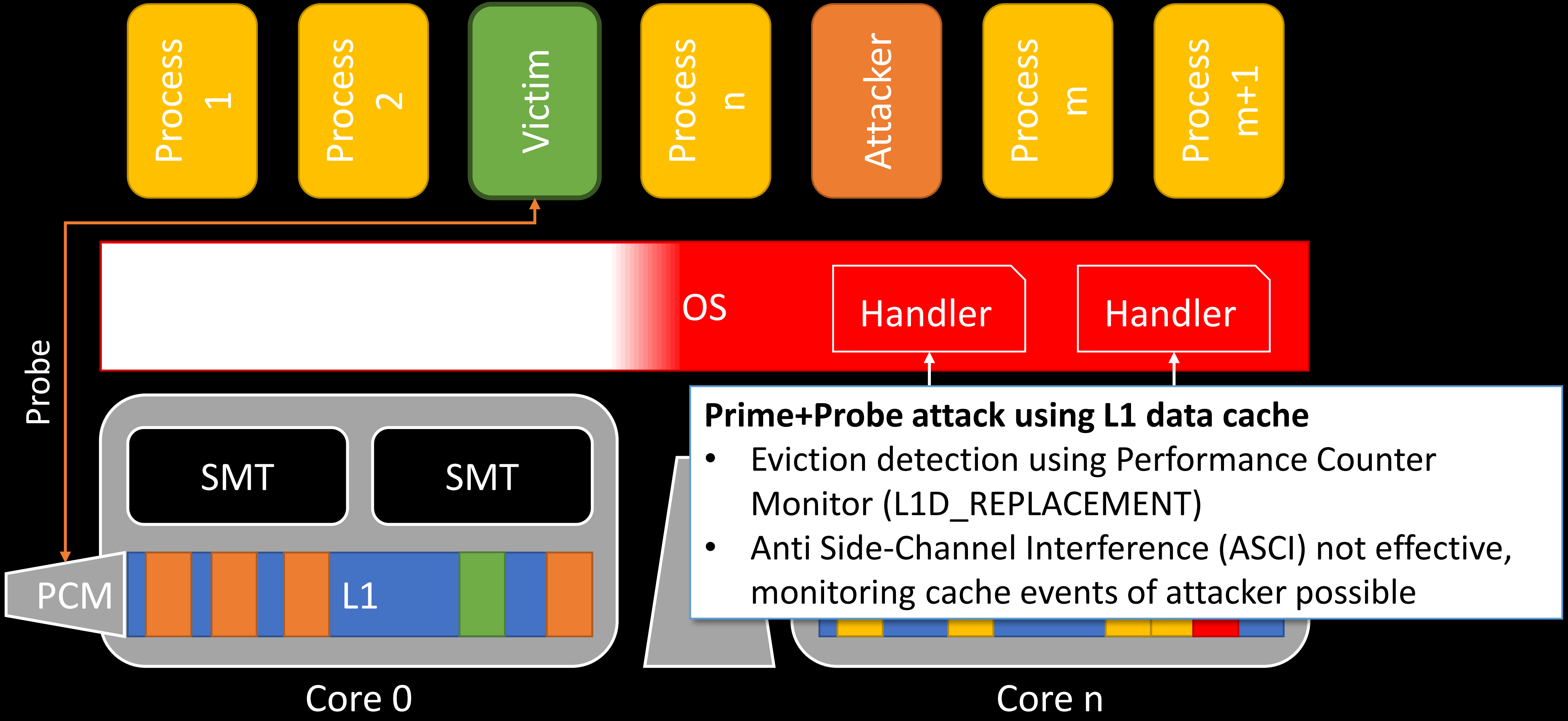
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

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]



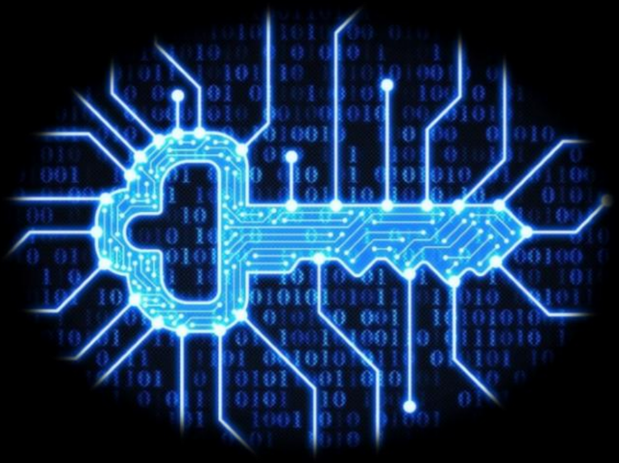
Prime+Probe attack using L1 data cache

- 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

Our Attack Use-Cases

[arXiv:1702.07521]



- Attacking RSA implementation from the Intel IIP crypto library in the Intel SGX SDK
- Extracting 2048-bit RSA decryption key

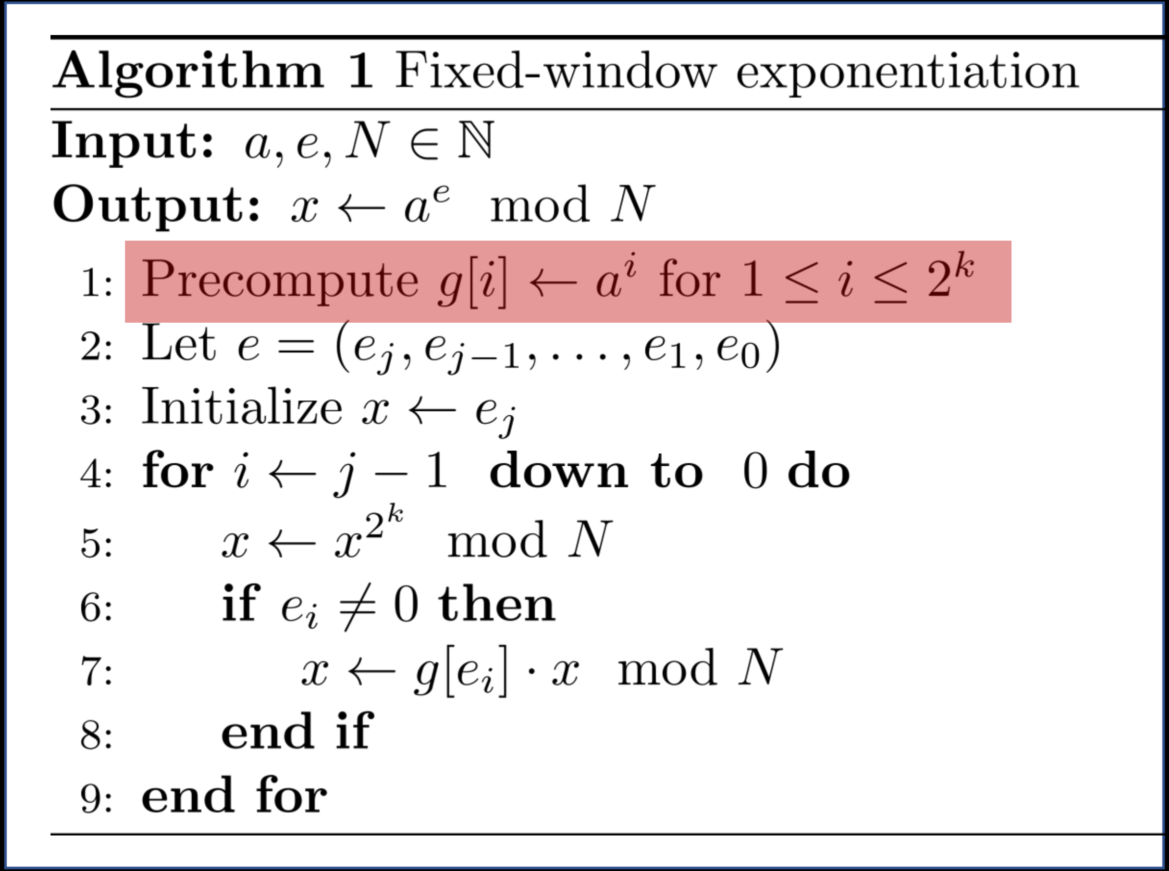
[Brasser et al., WOOT 2017]

- Attacking open source k-mer analysis tool PRIMEX [Lexa et al., Bioinformatics 2003]
- Extracting genome sequences

Extracting RSA decryption key

RSA Key Exfiltration: Victim Enclave

- RSA Decryption: $m = c^d \pmod N$



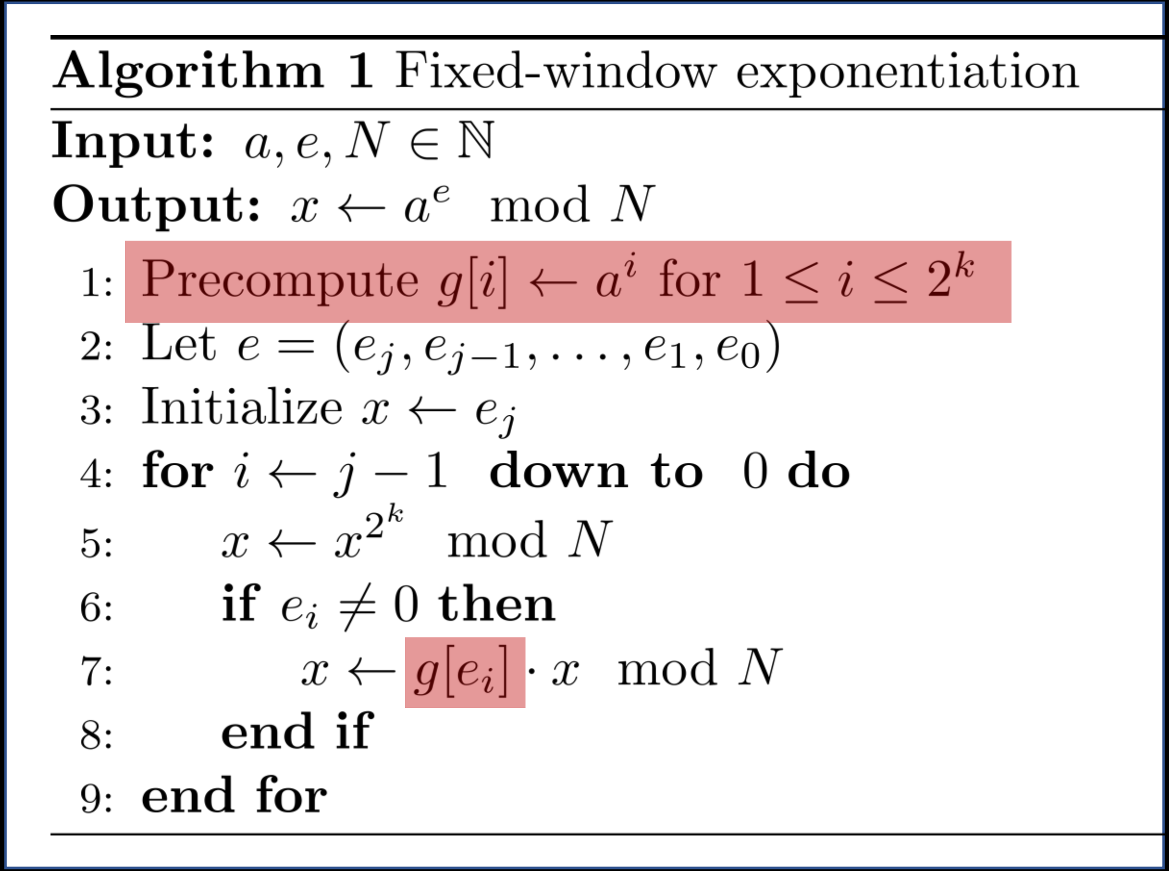
Algorithm 1 Fixed-window exponentiation

Input: $a, e, N \in \mathbb{N}$
Output: $x \leftarrow a^e \mod N$

```
1: Precompute $g[i] \leftarrow a^i$ for $1 \leq i \leq 2^k$
2: Let $e = (e_j, e_{j-1}, \ldots, e_1, e_0)$
3: Initialize $x \leftarrow e_j$
4: for $i \leftarrow j - 1$ down to $0$ do
5:     $x \leftarrow x^{2^k} \mod N$
6:     if $e_i \neq 0$ then
7:         $x \leftarrow g[e_i] \cdot x \mod N$
8:     end if
9: end for
```

RSA Key Exfiltration: Victim Enclave

- RSA Decryption: $m = c^d \pmod N$



Algorithm 1 Fixed-window exponentiation

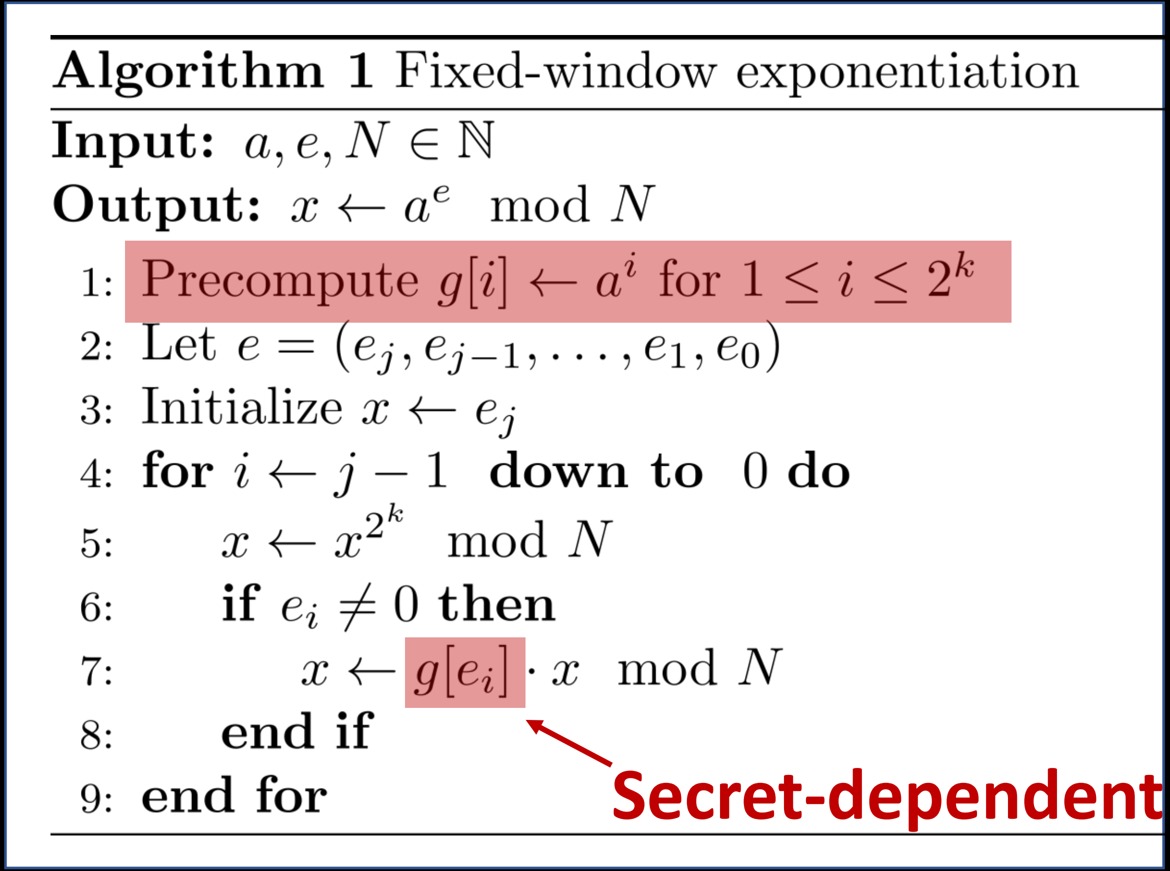
Input: $a, e, N \in \mathbb{N}$

Output: $x \leftarrow a^e \mod N$

```
1: Precompute $g[i] \leftarrow a^i$ for $1 \leq i \leq 2^k$
2: Let $e = (e_j, e_{j-1}, \ldots, e_1, e_0)$
3: Initialize $x \leftarrow e_j$
4: for $i \leftarrow j-1$ down to $0$ do
5:     $x \leftarrow x^{2^k} \mod N$
6:     if $e_i \neq 0$ then
7:         $x \leftarrow g[e_i] \cdot x \mod N$
8:     end if
9: end for
```

RSA Key Exfiltration: Victim Enclave

- RSA Decryption: $m = c^d \pmod N$



Algorithm 1 Fixed-window exponentiation

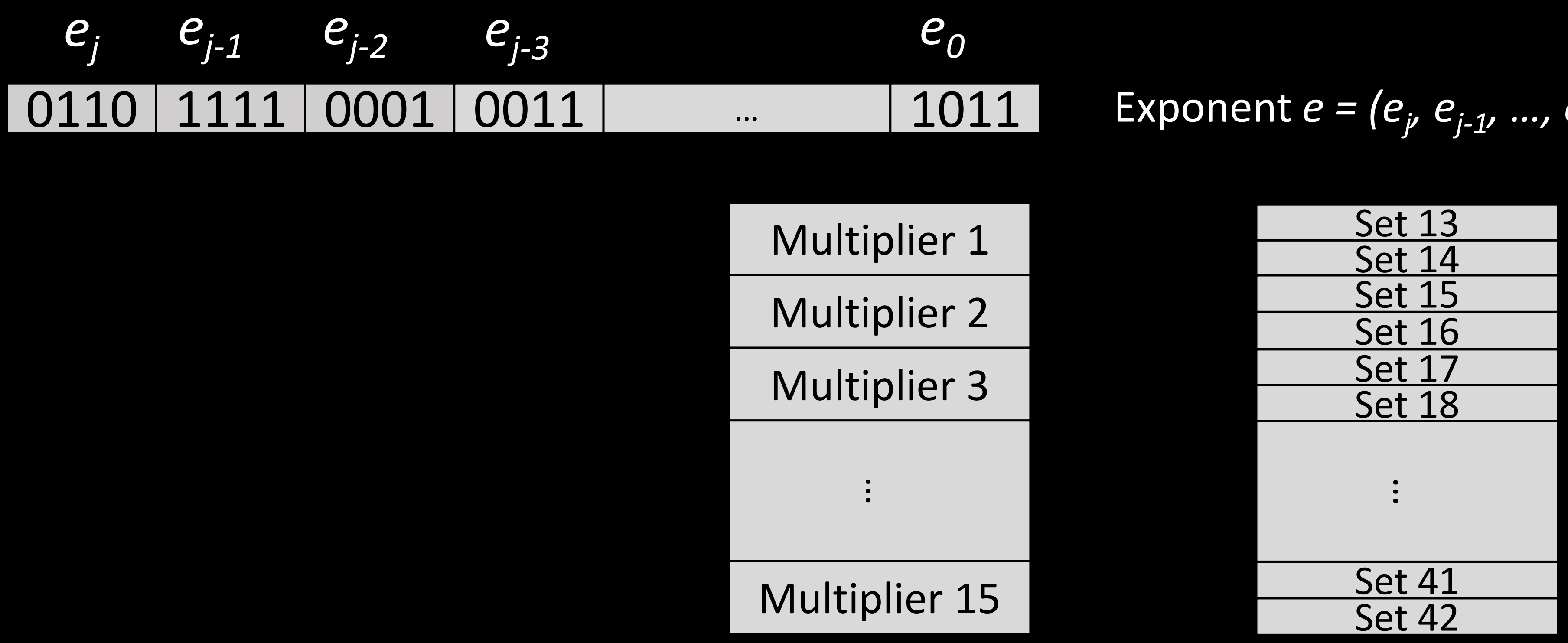
Input: $a, e, N \in \mathbb{N}$

Output: $x \leftarrow a^e \mod N$

```
1: Precompute $g[i] \leftarrow a^i$ for $1 \leq i \leq 2^k$
2: Let $e = (e_j, e_{j-1}, \ldots, e_1, e_0)$
3: Initialize $x \leftarrow e_j$
4: for $i \leftarrow j - 1$ down to $0$ do
5:     $x \leftarrow x^{2^k} \mod N$
6:     if $e_i \neq 0$ then
7:         $x \leftarrow g[e_i] \cdot x \mod N$
8:     end if
9: end for
```

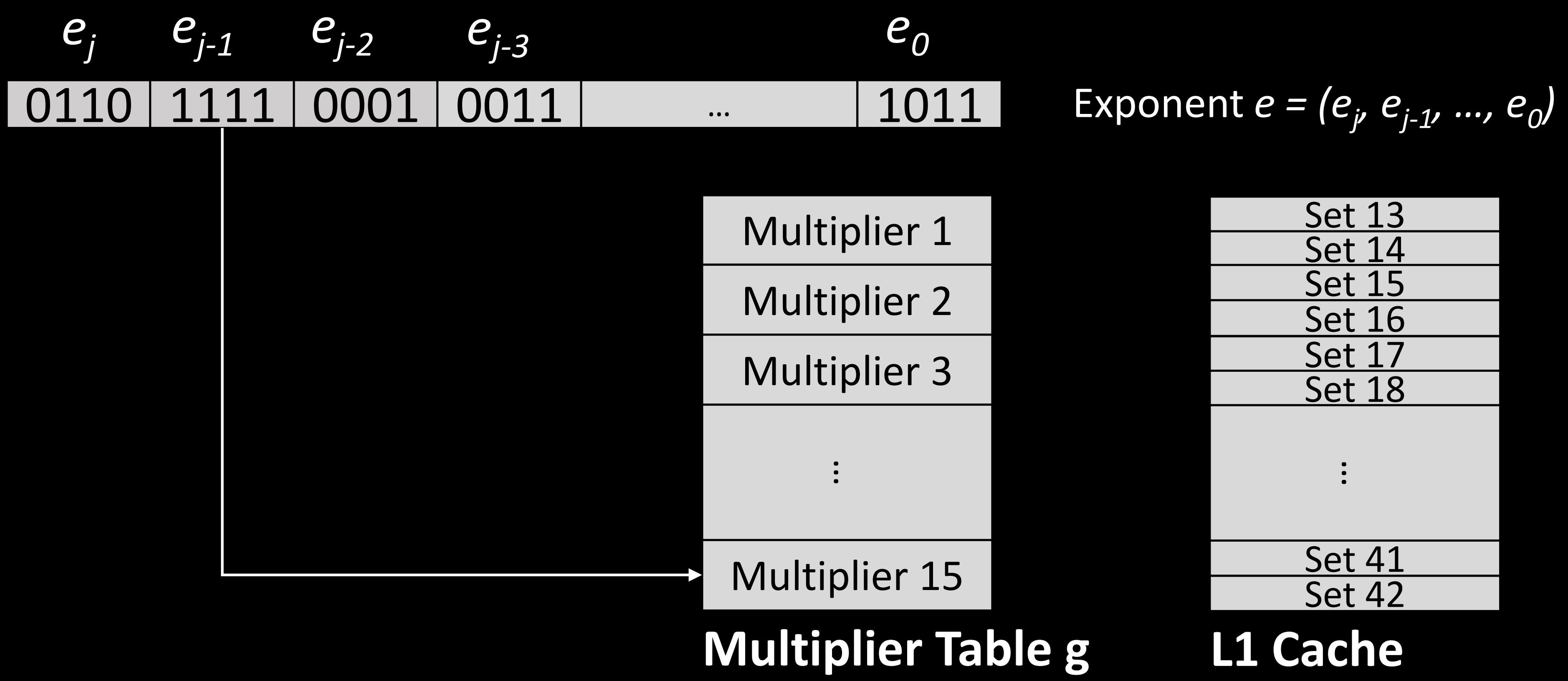
Secret-dependent memory access!

Fixed-size Sliding Window Exponentiation

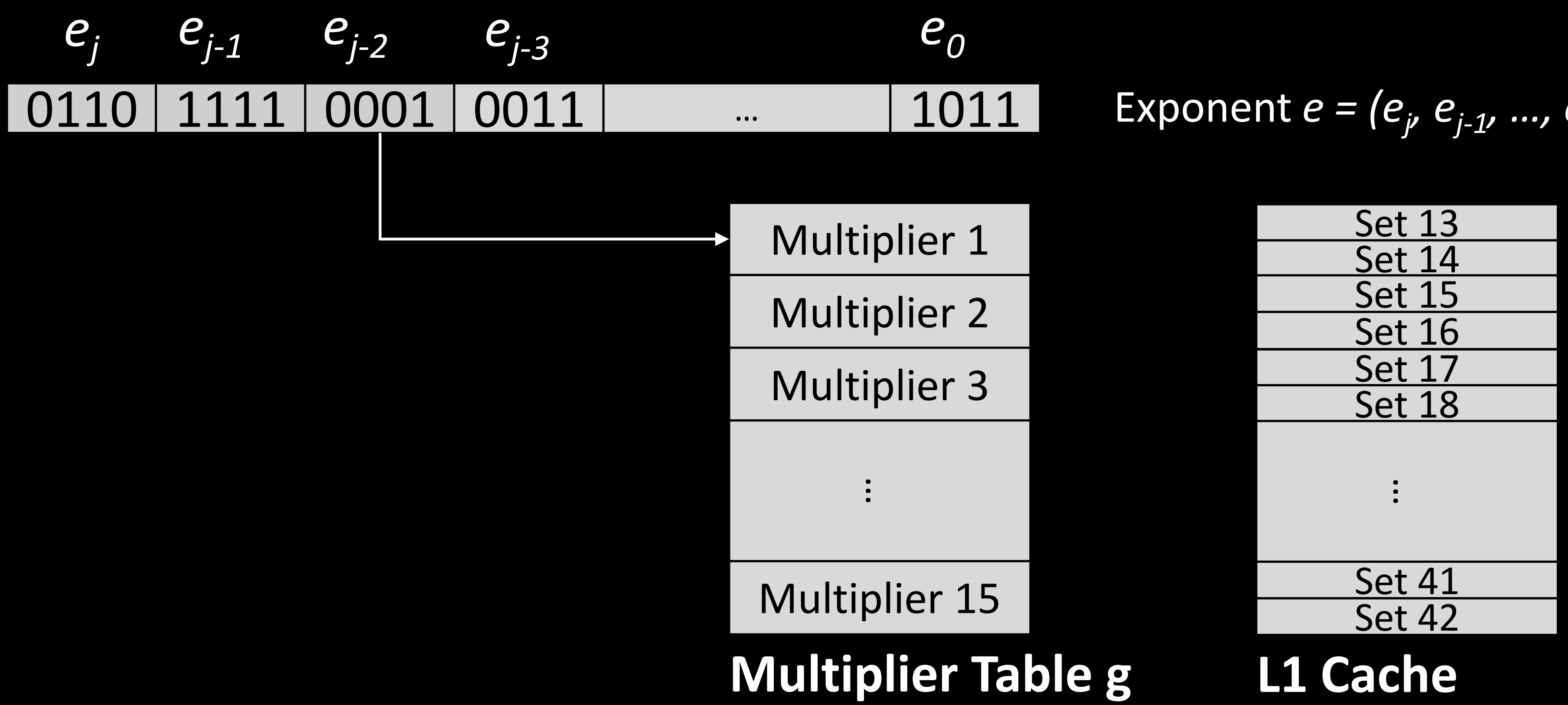


Multiplier Table g **L1 Cache**

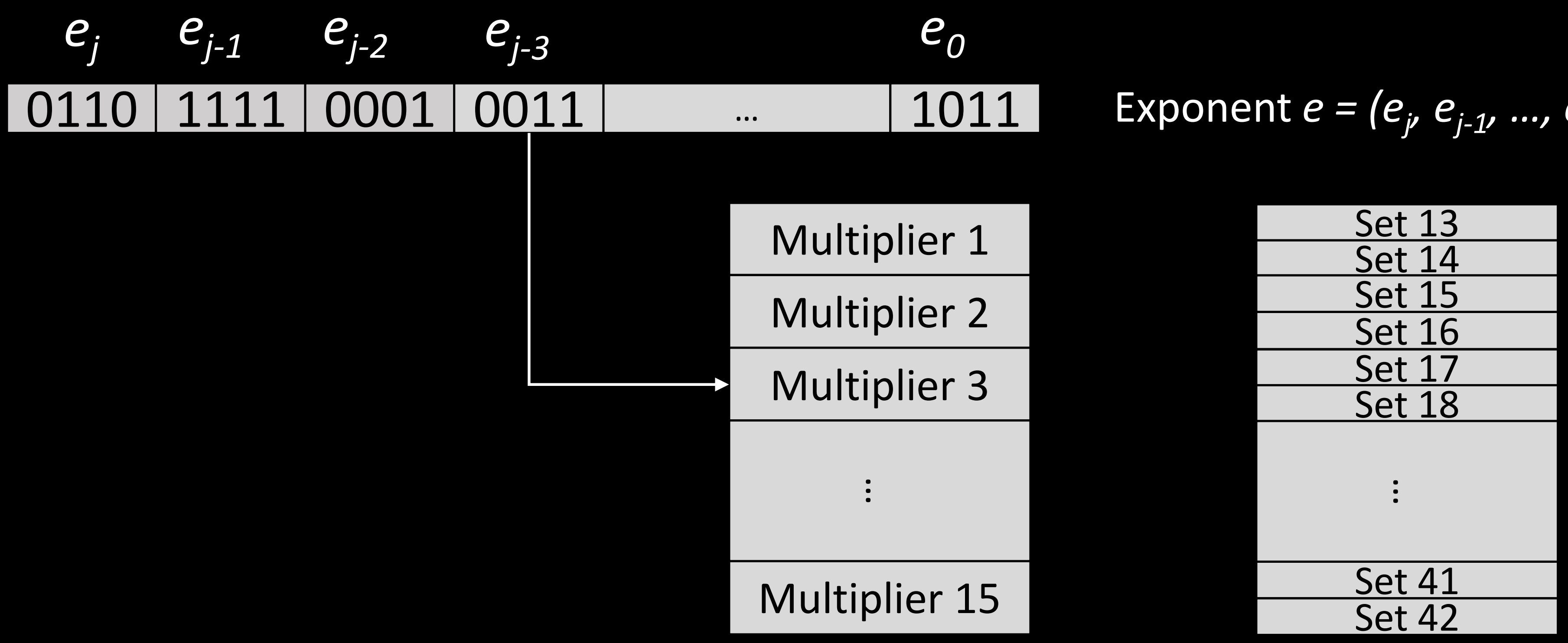
Fixed-size Sliding Window Exponentiation



Fixed-size Sliding Window Exponentiation



Fixed-size Sliding Window Exponentiation

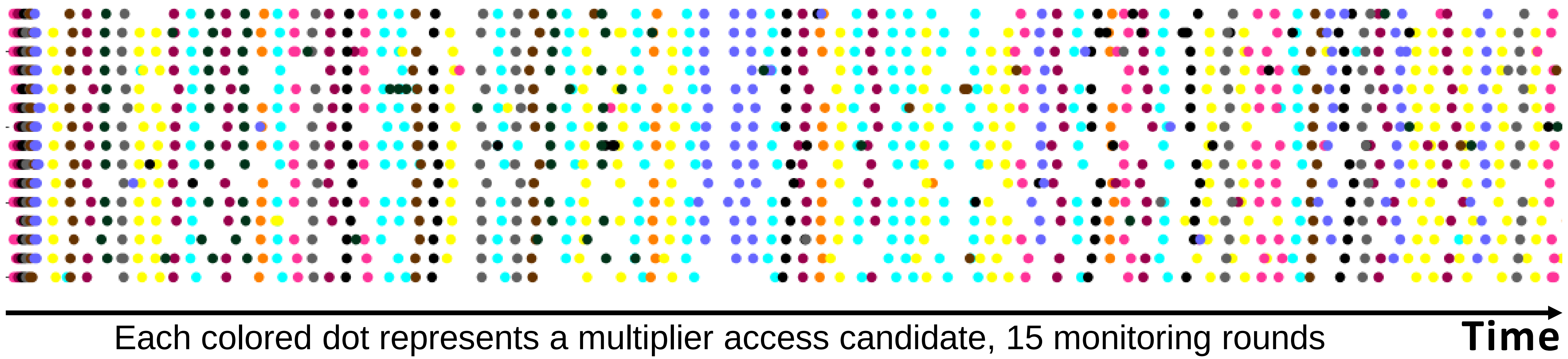


Multiplier Table g

L1 Cache

Attack Result

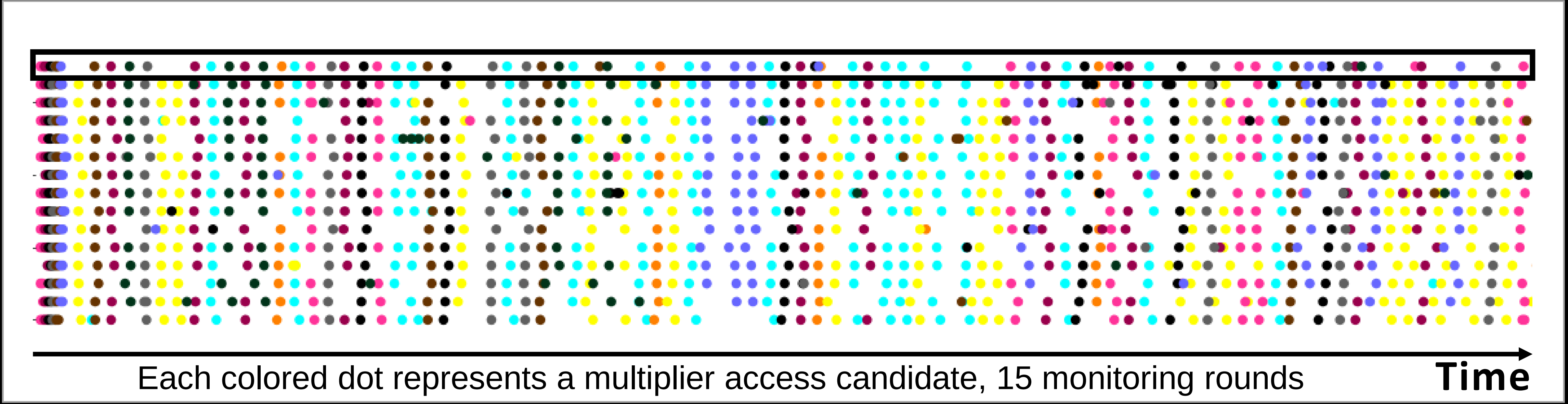
- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



Each colored dot represents a multiplier access candidate, 15 monitoring rounds

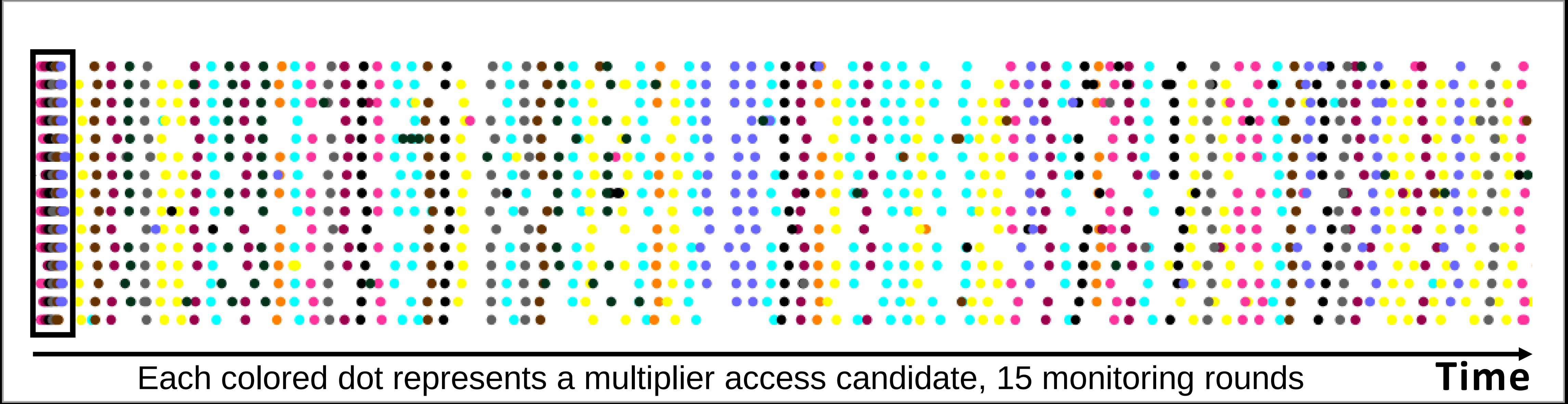
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



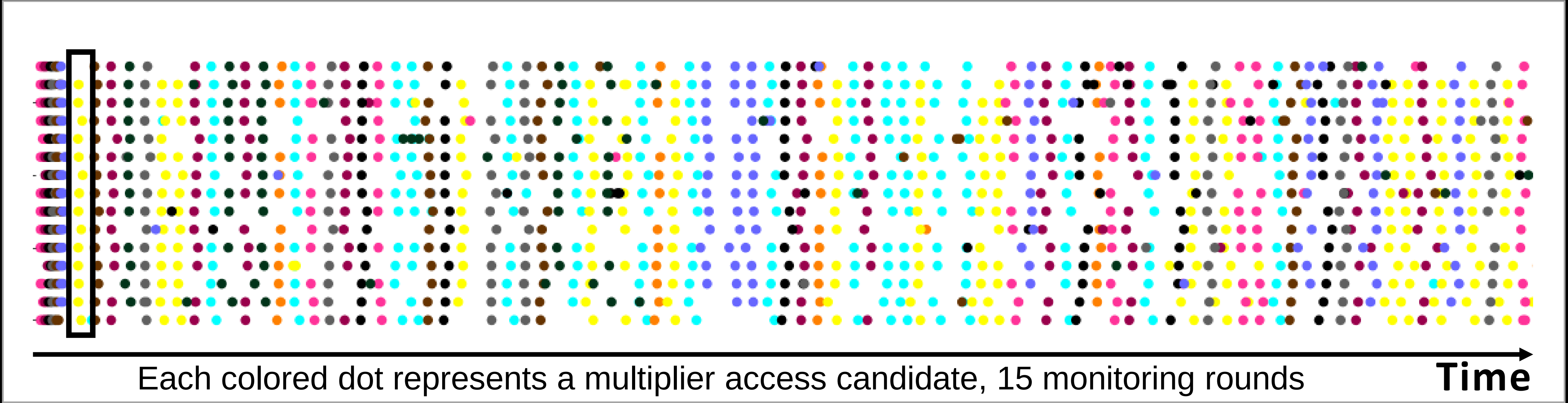
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



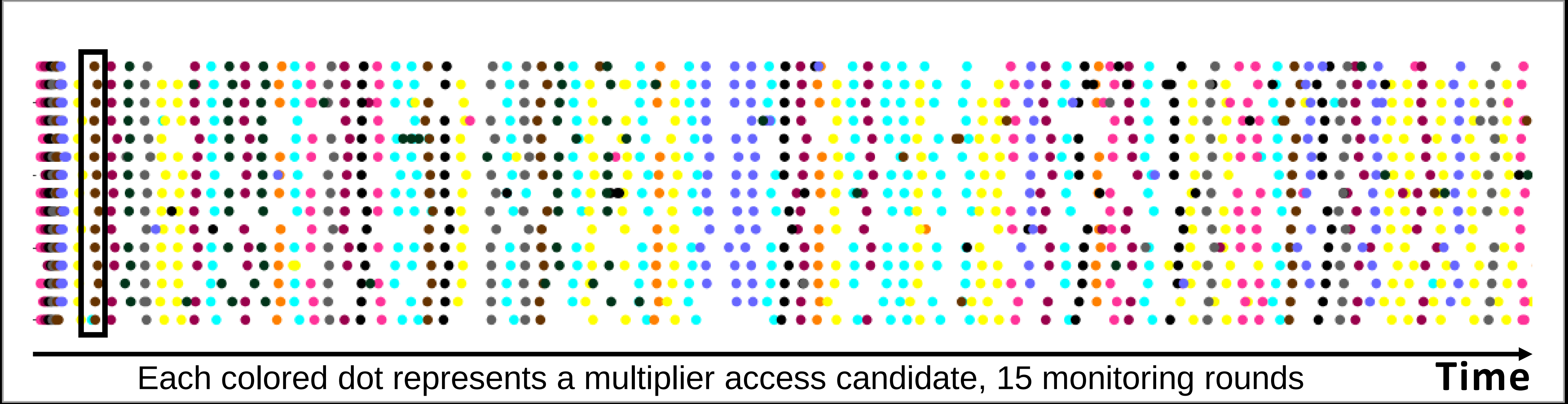
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO’09]



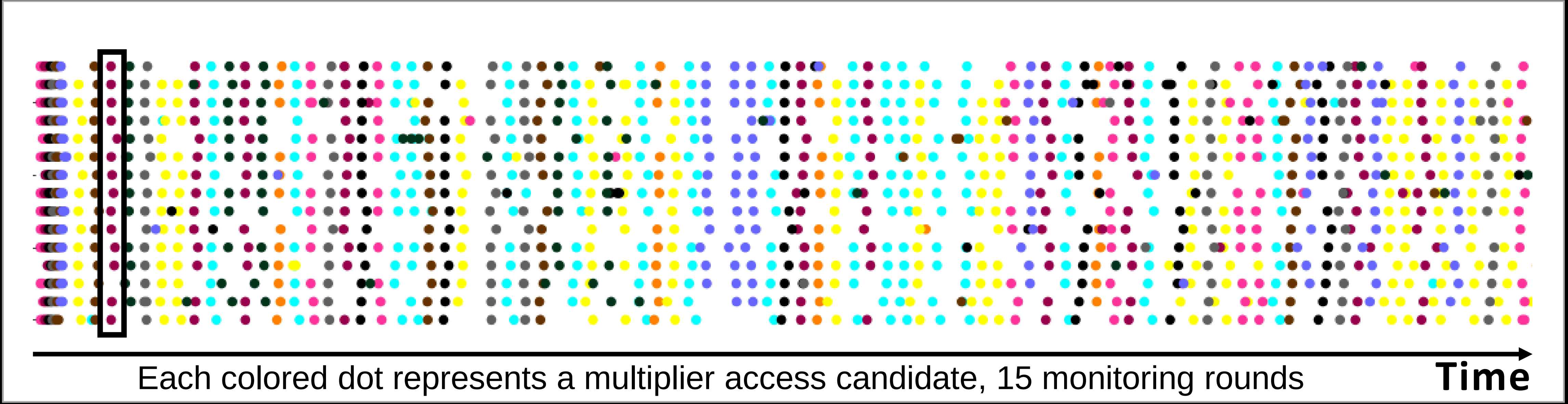
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



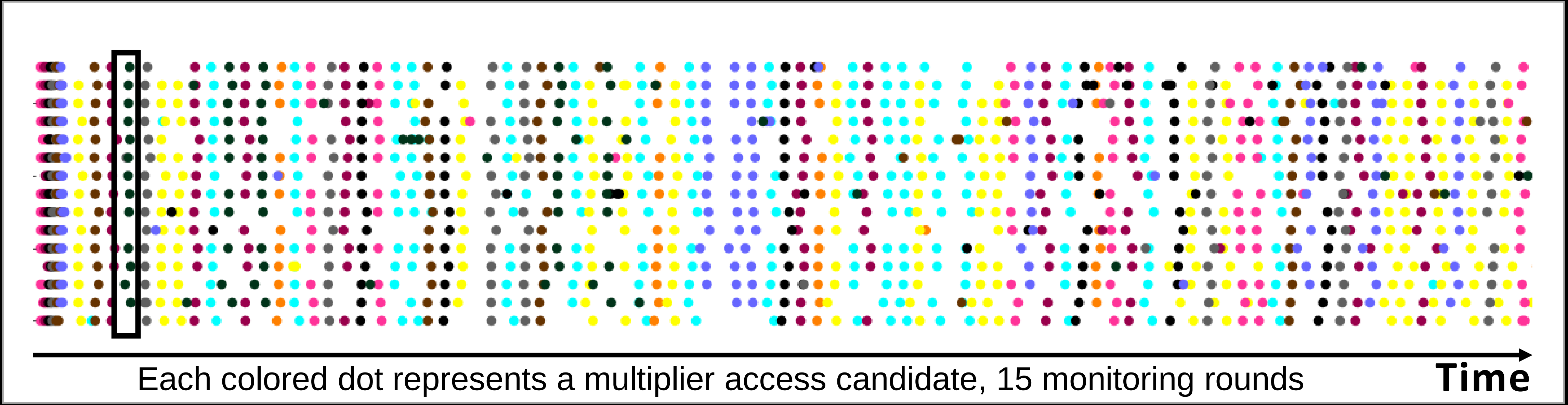
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



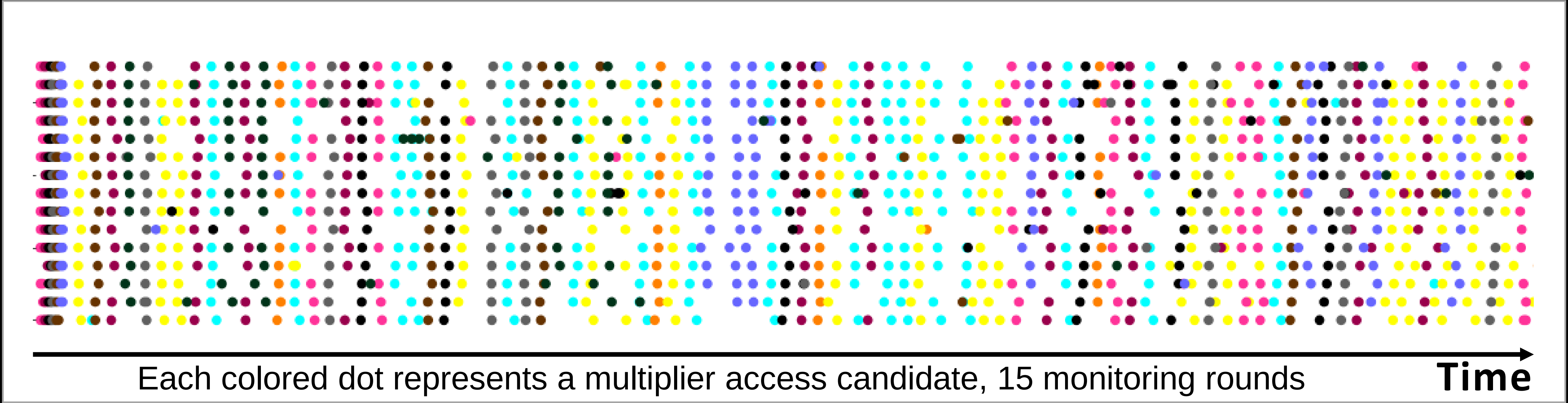
Attack Result

- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]

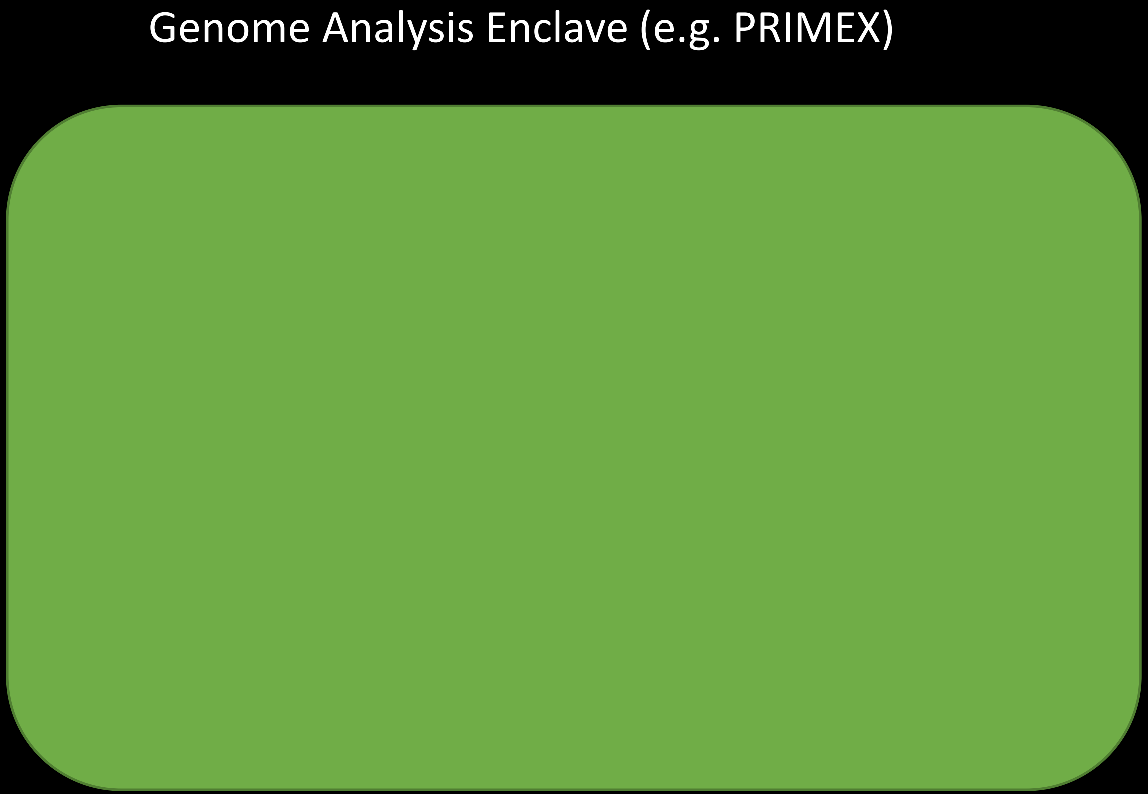


Attack Result

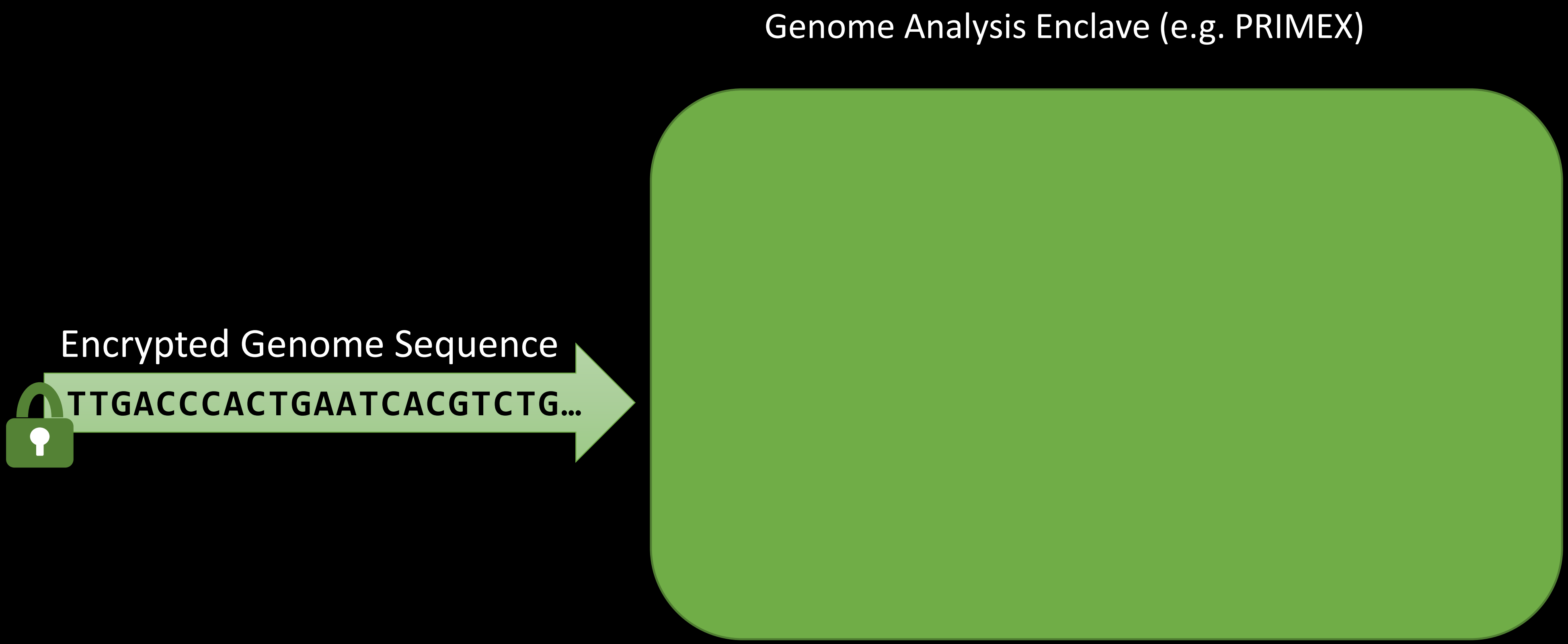
- 2048-bit Chinese Remainder Theorem RSA key
- Only 300 decryptions to leak 70% of key bits
- Enough to recover key [Heninger et. al., CRYPTO'09]



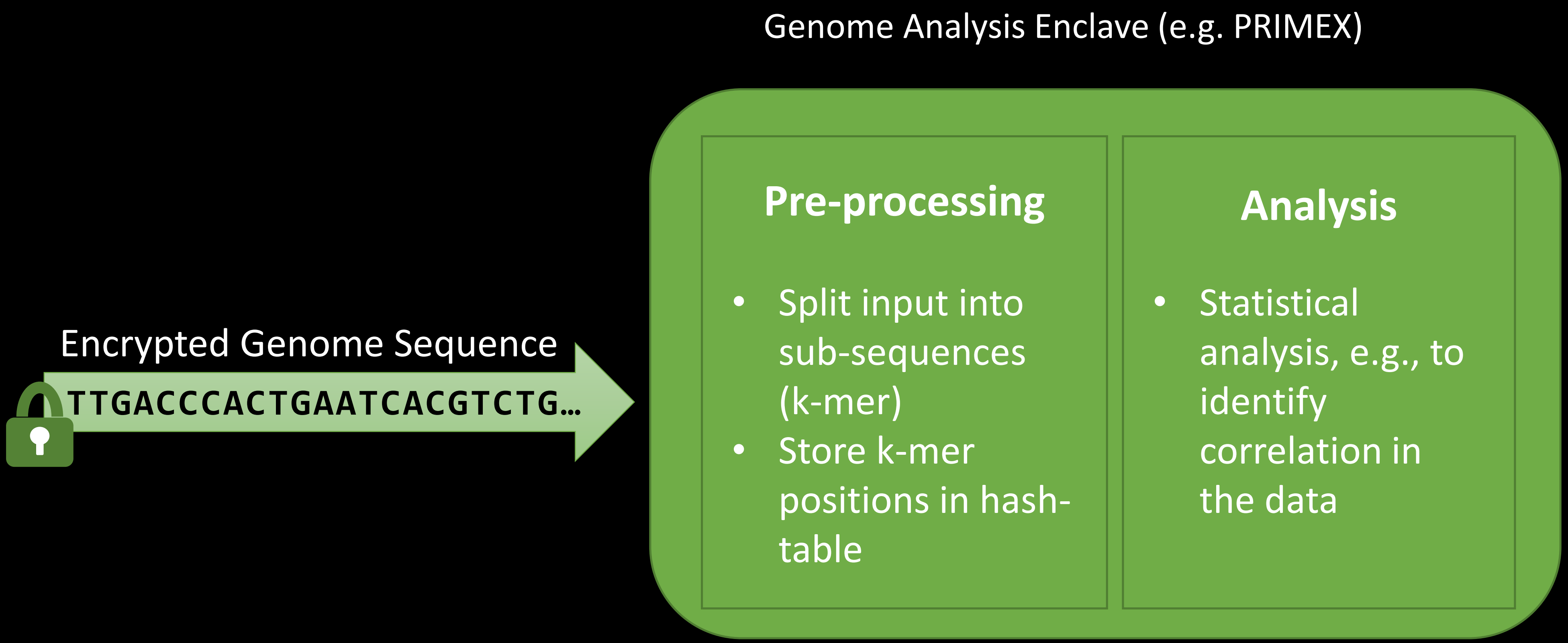
Genome Sequencing



Genome Sequencing


Genome Analysis Enclave (e.g. PRIMEX)
Encrypted Genome Sequence
TTGACCCACTGAATCACGTCTG…

Genome Sequencing


Genome Analysis Enclave (e.g. PRIMEX)
Encrypted Genome Sequence
TTGACCCACTGAATCACGTCTG…
Pre-processing
Analysis

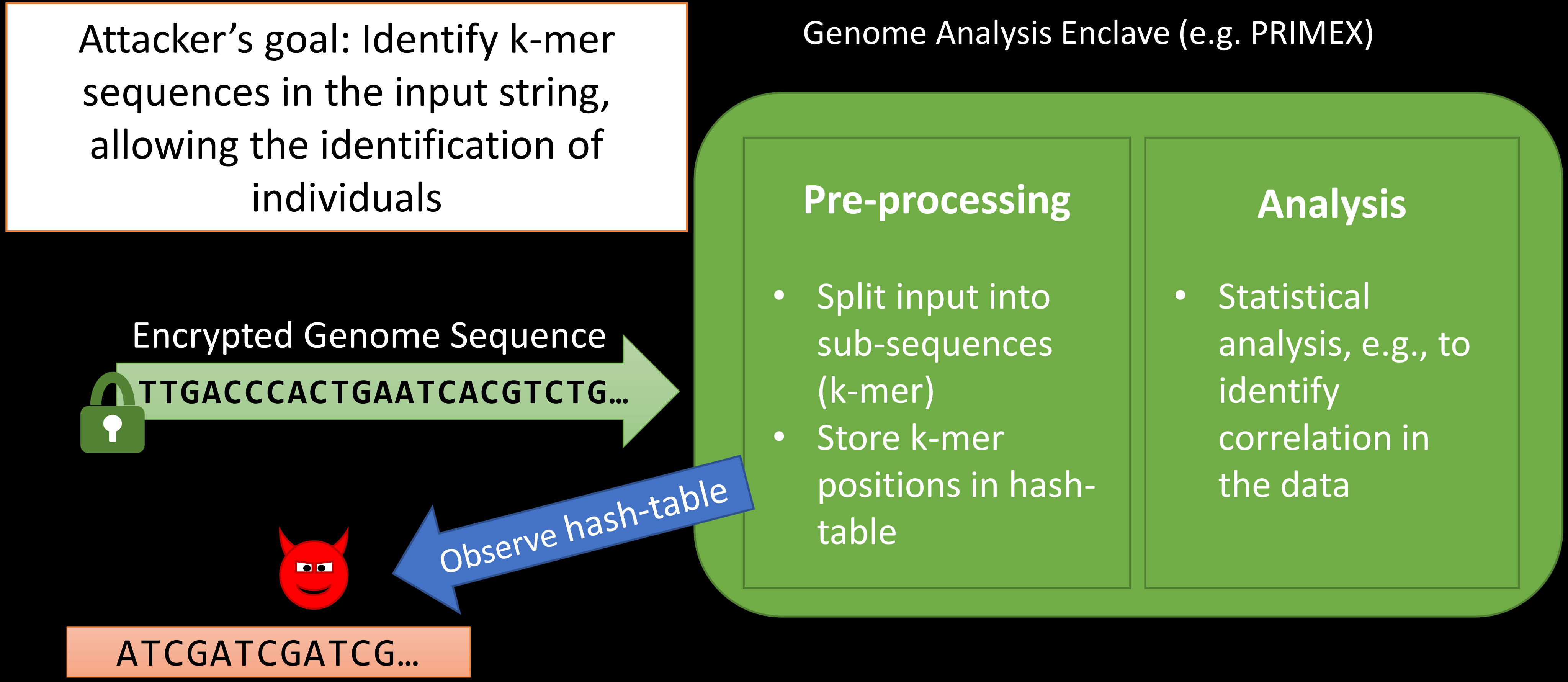
Pre-processing

- 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


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

Some Basics on Human Genomes

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

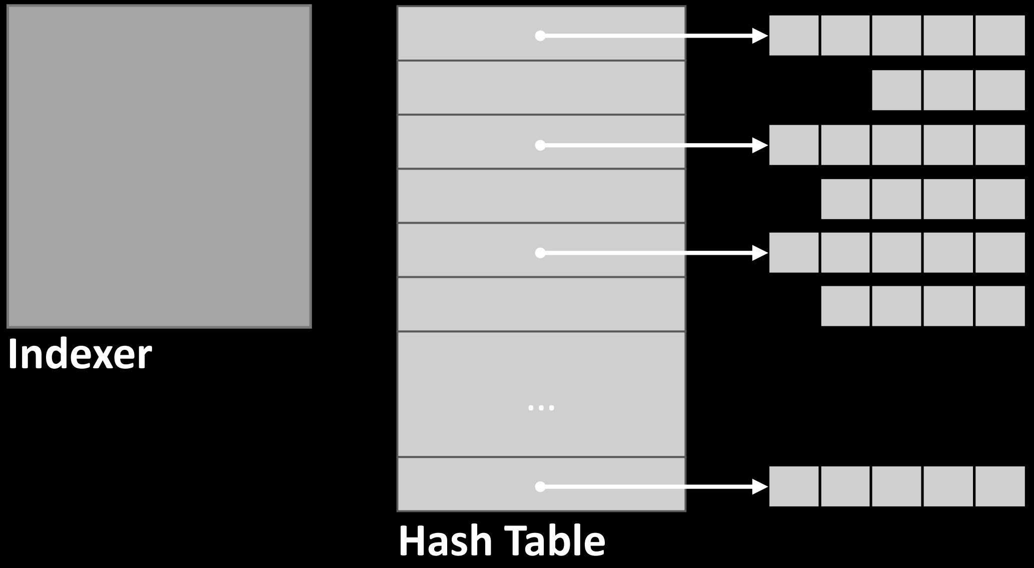
Some Basics on Human Genomes

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

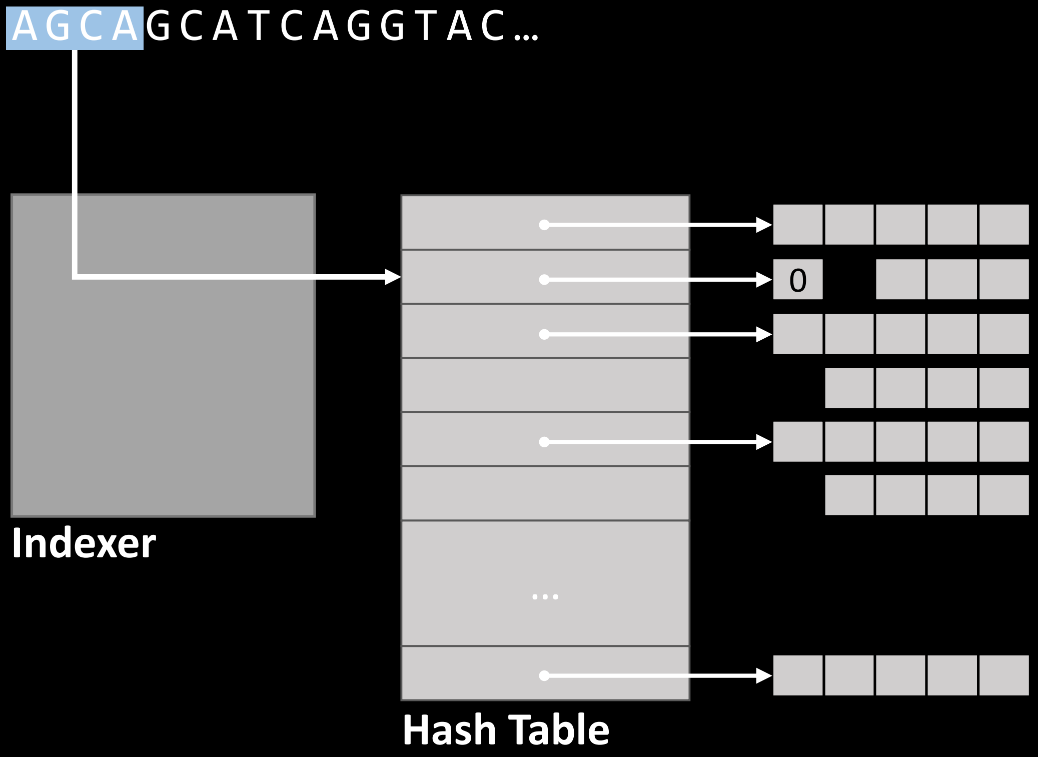
```
TTGACCCACTGAATCACGTCTGACCGCGCGTACGCGG
TCACTTGCGGTGCCGTTTTCTTTGTTACCGACGACCG
ACCAGCGACAGCCACCGCGCTCACTGCCACCAAAA
GAGTCATATCGATCGATCGATCGATCGATCGATCGAT
CGATCGATCGATCGATCGATCGATCGATCGATCATCA
CAGCCGACCAGTTTCTGGAACGTTCCCGATACTGGAA
CGGTCCTAATGCAGTATCCCACCCTCCTTCCATCGAC
GCCAGTCGAATCACGCCGCCAGCCACCGTCCGCCAGC
CGGCCAGAATACCGATGACTCGGCGGTCTCGTGTCGG
TGCCGGCCTCGCAGCCATTGTACTGGCCCTGGCCGCA
GTGTCGGCTGCCGCTCCGATTGCCGGGGCGCAGTCCG
CCGGCAGCGGTGCGGTCTCAGTCACCATCGGCGACGT
GGACGTCTCGCCTGCGAACCCAACCACGGGCACGCAG
GTGTTGATCACCCCGTCGATCAACAACTCCGGATCGG
CAAGCGGGTCCGCGCGTCAACGAGGTCACGCTGCG
CGGCGACGGTCTCCTCGCAACGGAAGACAGCCTGGGG
```

Genome Pre-Processing

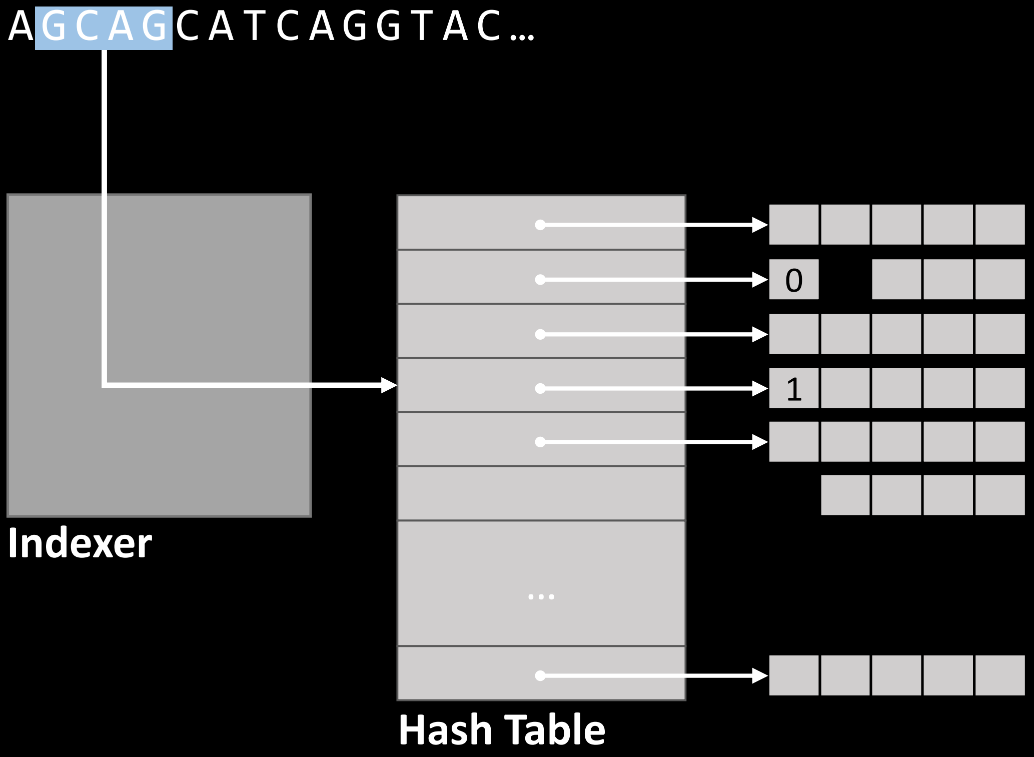
A G C A G C A T C A G G T A C …



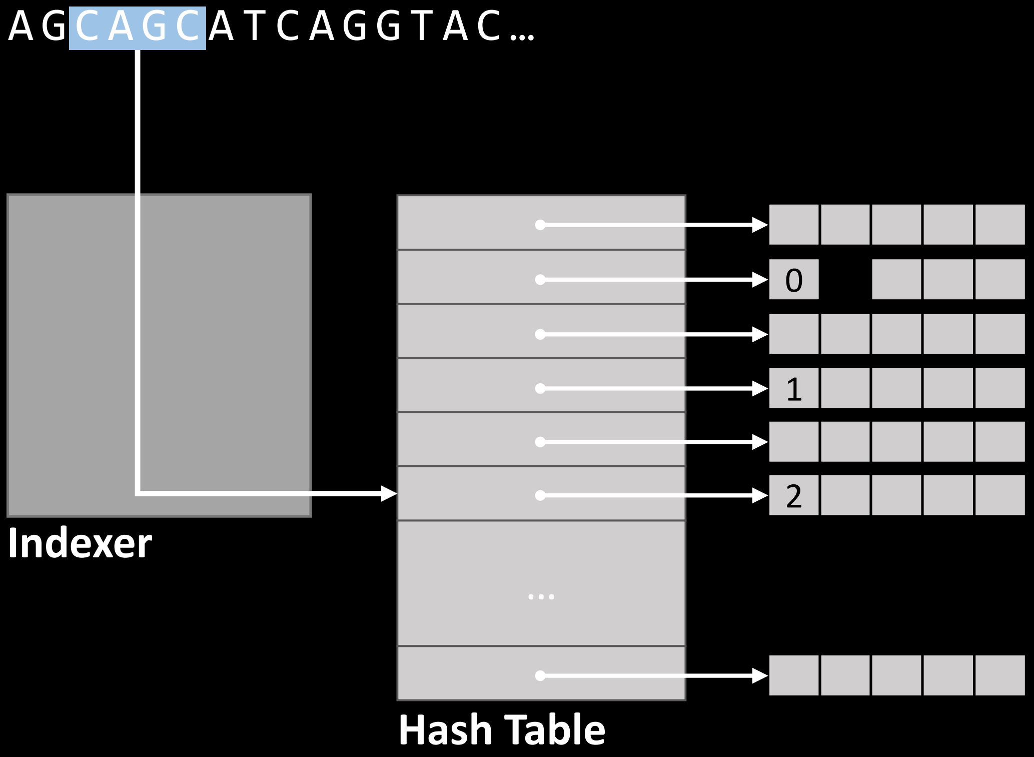
Genome Pre-Processing



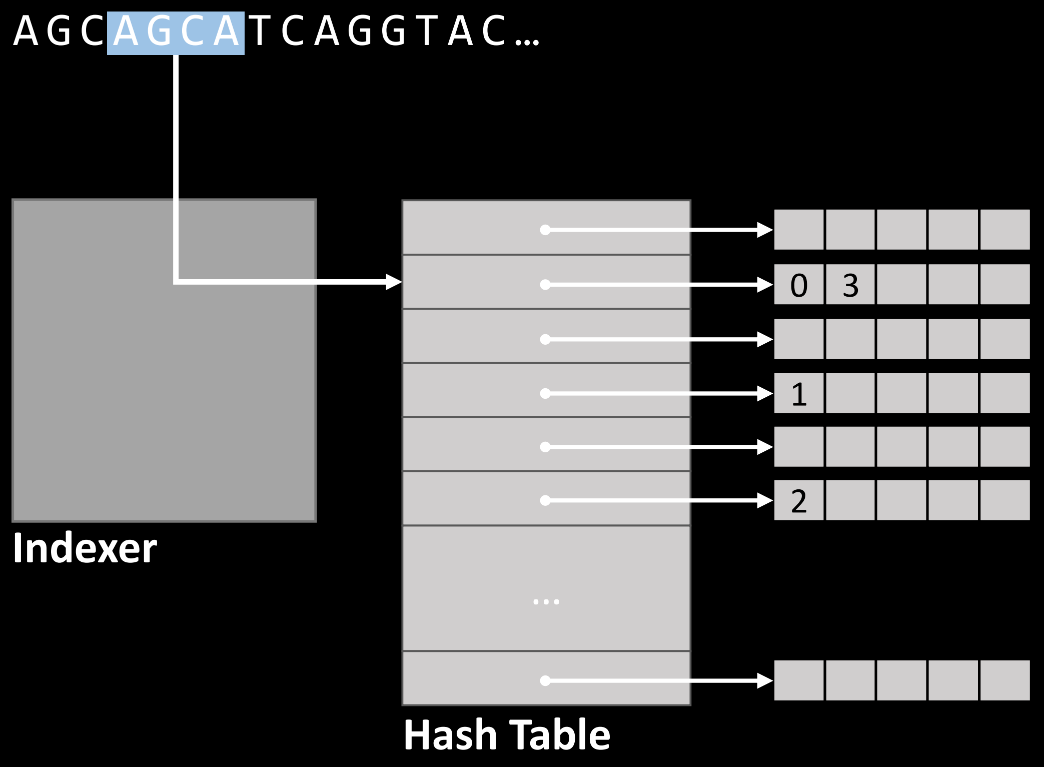
Genome Pre-Processing



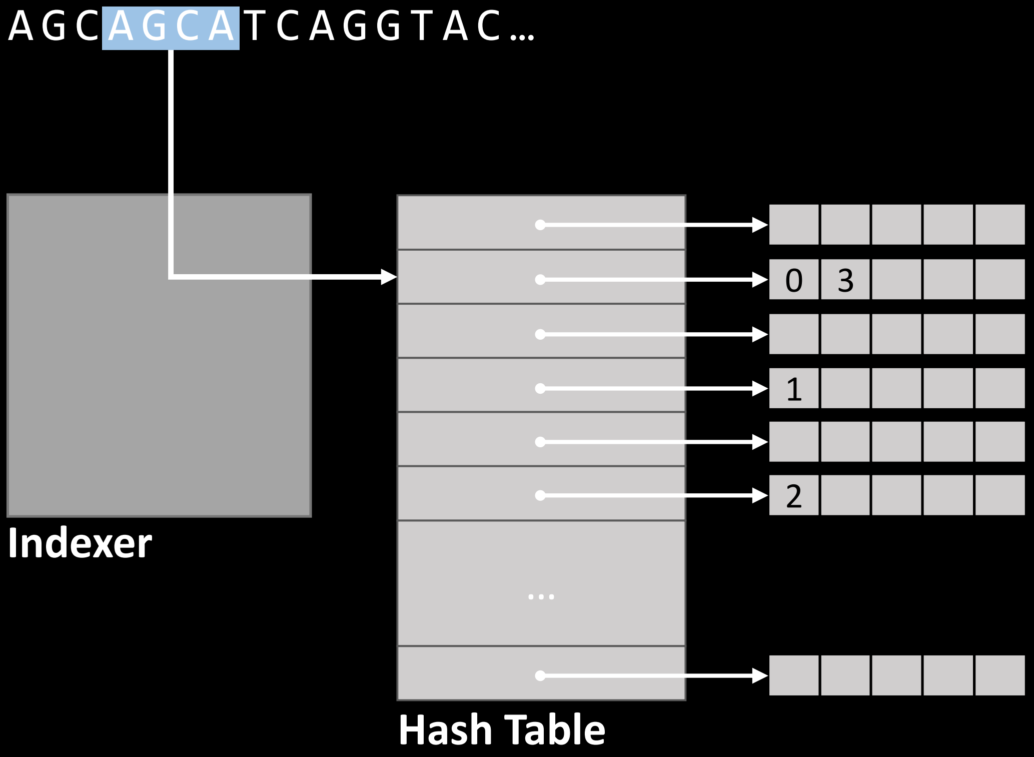
Genome Pre-Processing



Genome Pre-Processing

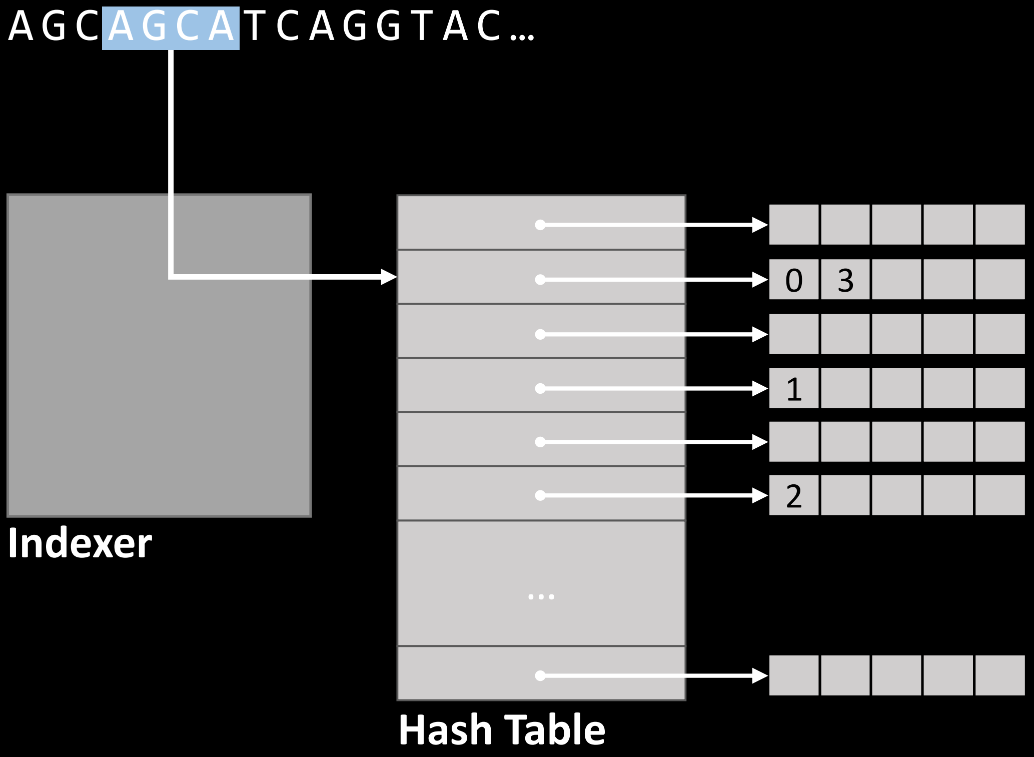


Genome Pre-Processing



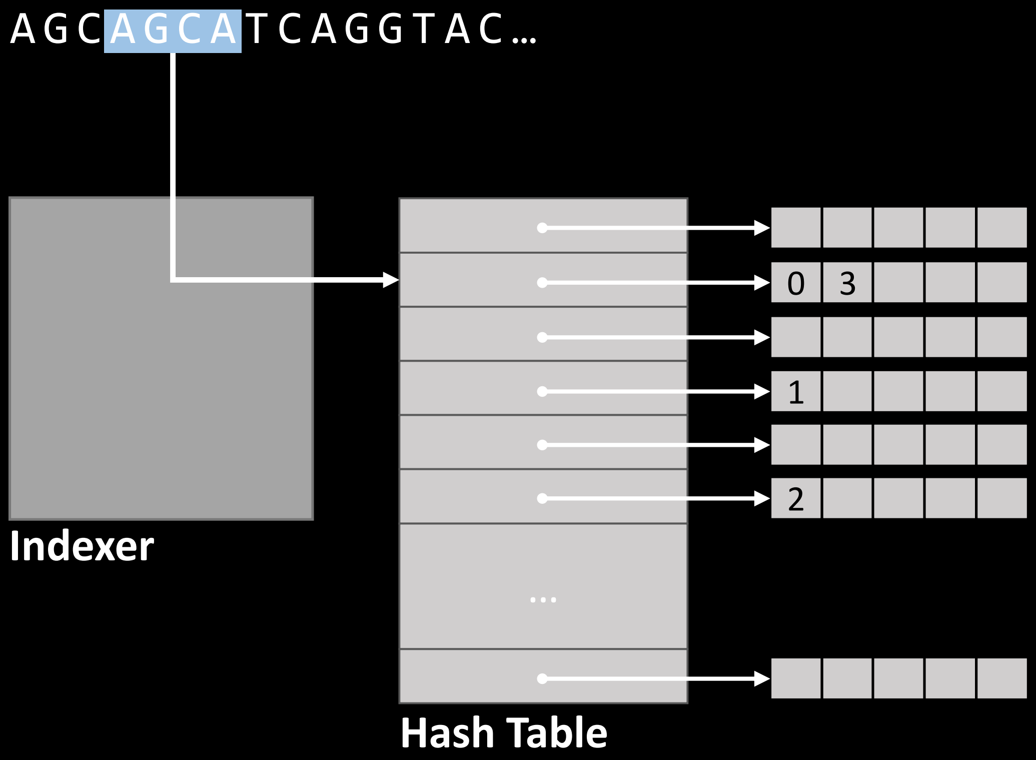
- Hash table access pattern
  - Hash table entry 8 bytes
  - Cache line size 64 bytes
  - Collisions
- Genome unstructured
- Microsatellites structured

Genome Pre-Processing



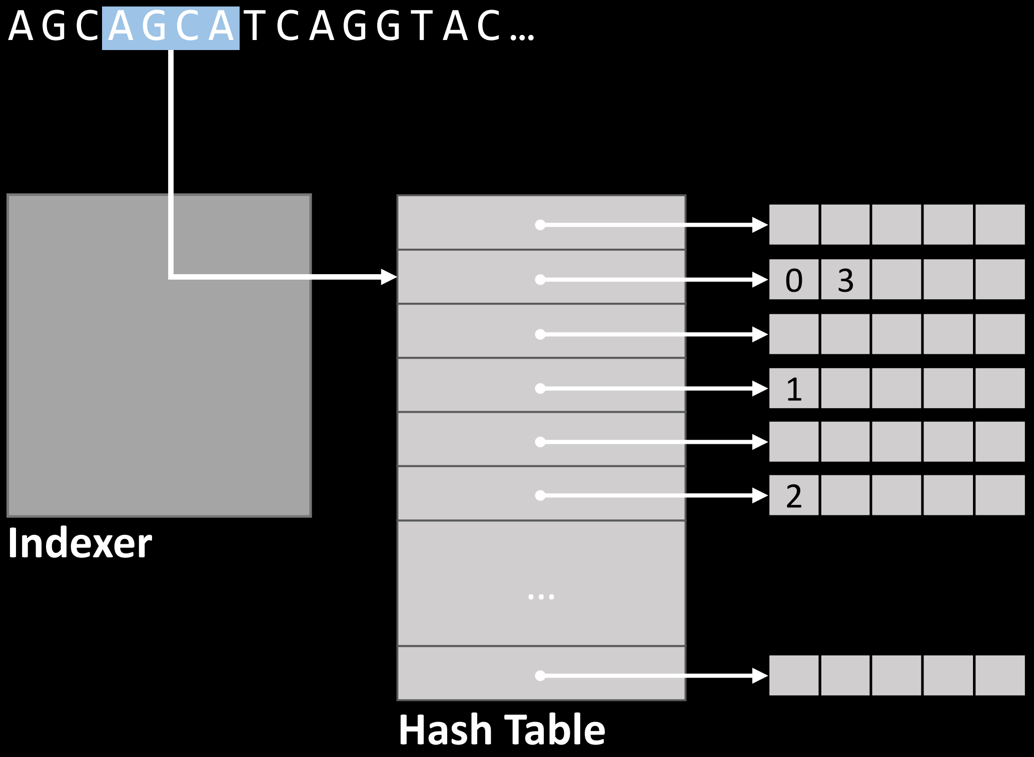
- Hash table access pattern
  - Hash table entry 8 bytes
  - Cache line size 64 bytes
  - Collisions
- Genome unstructured
- Microsatellites structured

Genome Pre-Processing



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

Genome Pre-Processing



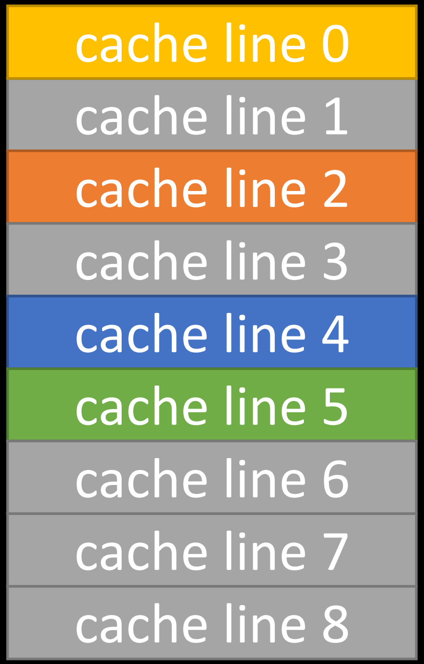
- Hash table access pattern
  - Hash table entry 8 bytes
  - Cache line size 64 bytes
  - Collisions
- Genome unstructured
- Microsatellites structured

```
TTGACCCACTGAATCACGTCTGACCGCGCGTACGCGGTCACTTGC
GGTGCCGTTTTCTTTGTTACCGACGACCGACCAGCGACAGCCACC
GCGCGCTCACTGCCACCAAAAGAGTCATATCGATCGATCGATCGA
TCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGAT
CATCACAGCCGACCAGTTTCTGGAACGTTCCCGATACTGGAACGG
TCCTAATGCAGTATCCCACCCTCCTTCCATCGACGCCAGTCGAAT
CACGCCGCCAGCCACCGTCCGCCAGCCGGCCAGAATACCGATGAC
TCGGCGGTCTCGTGTCGGTGCCGGCCTCGCAGCCATTGTACTGGC
CCTGGCCGCAGTGTCGGCTGCCGCTCCGATTGCCGGGGCGCAGTC
CGCCGGCAGCGGTGCGGTCTCAGTCACCATCGGCGACGTGGACGT
CTCGCCTGCGAACCCAACCACGGGCACGCAGGTGTTGATCACCCC
```

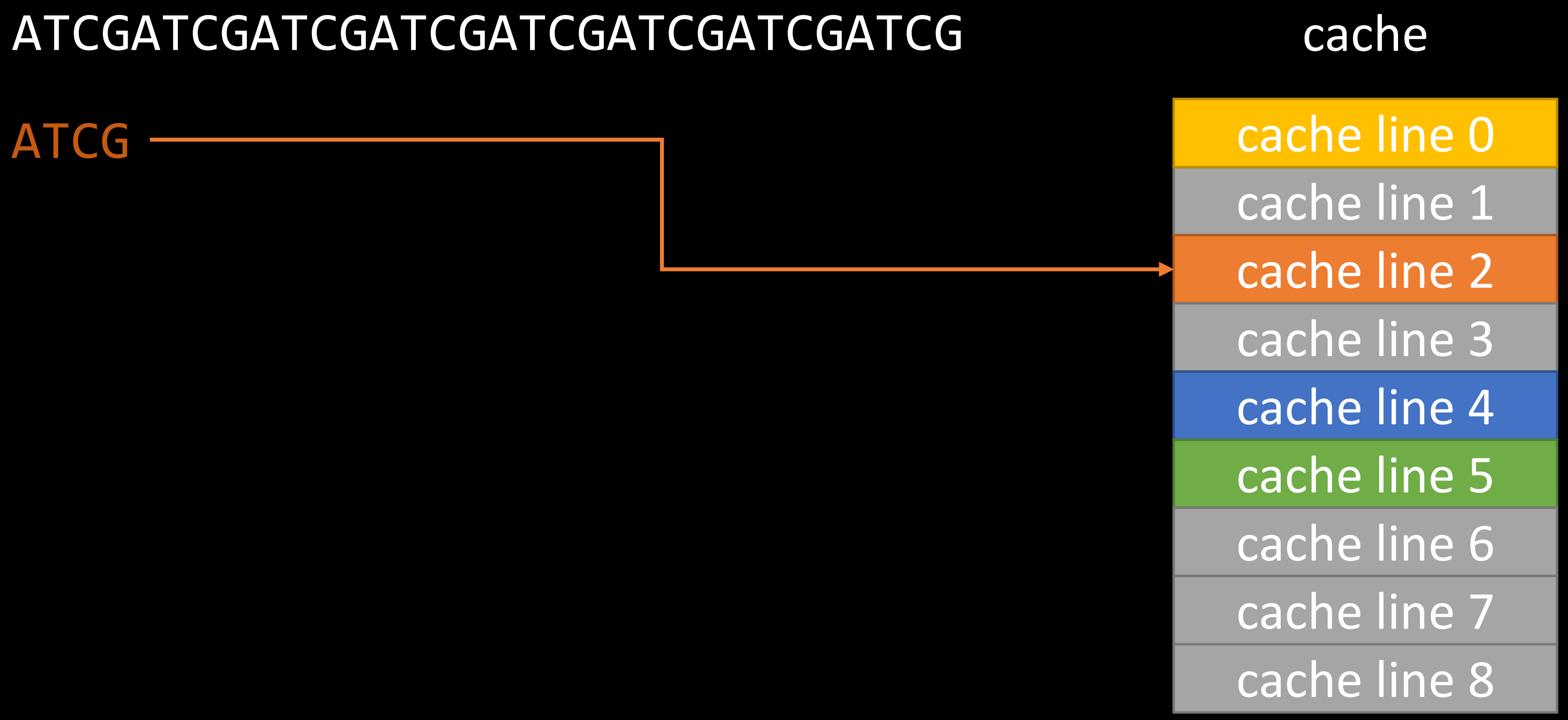
Microsatellites and Processed k-mers

ATCGATCGATCGATCGATCGATCGATCGATCG

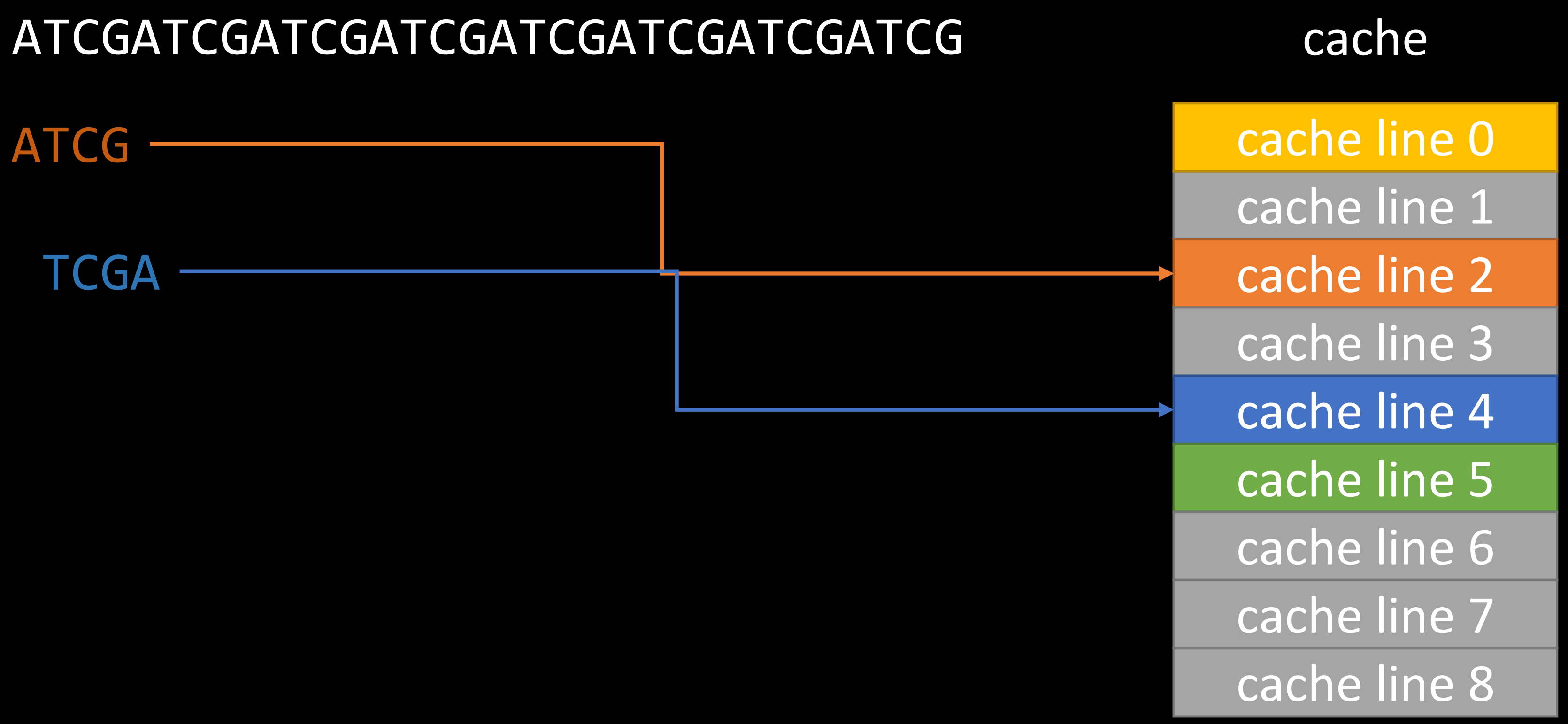
cache



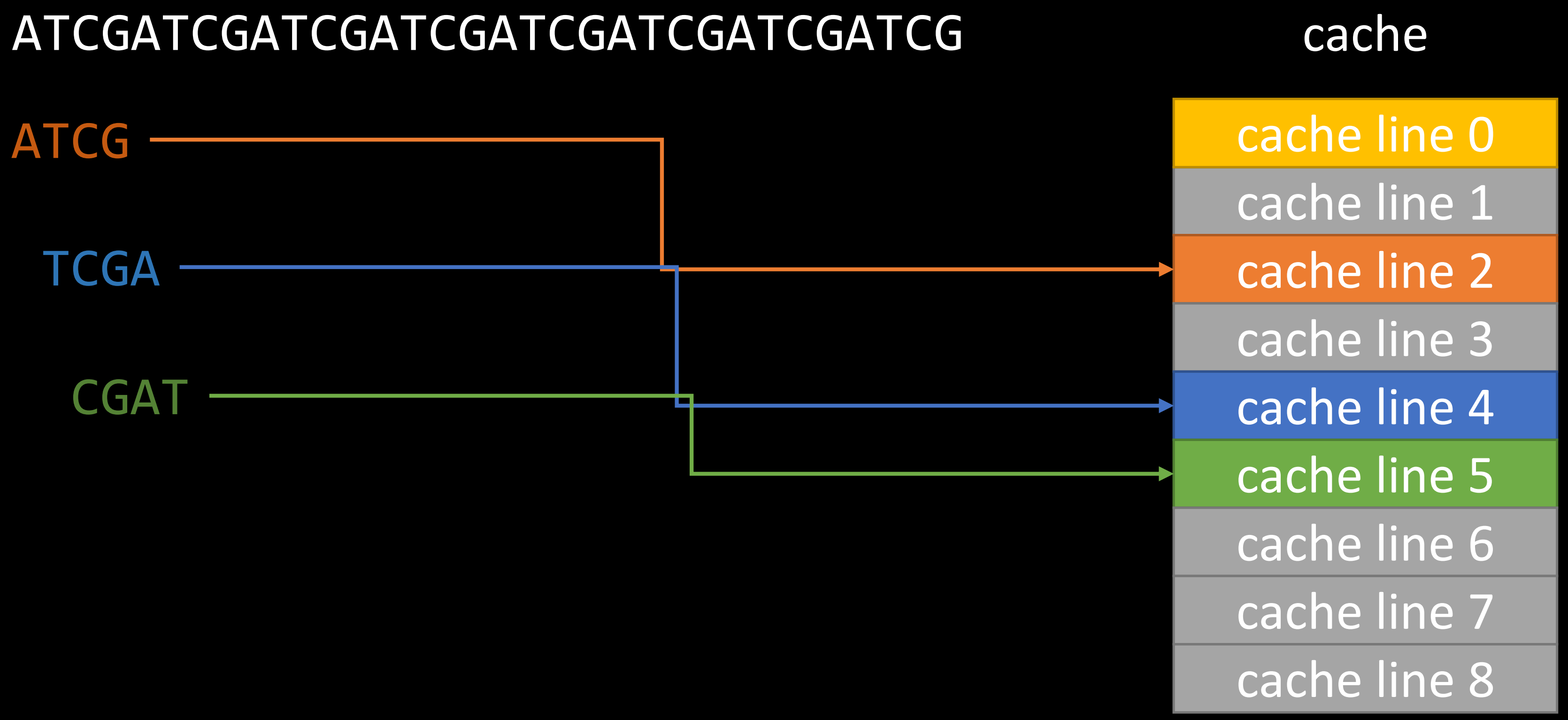
Microsatellites and Processed k-mers



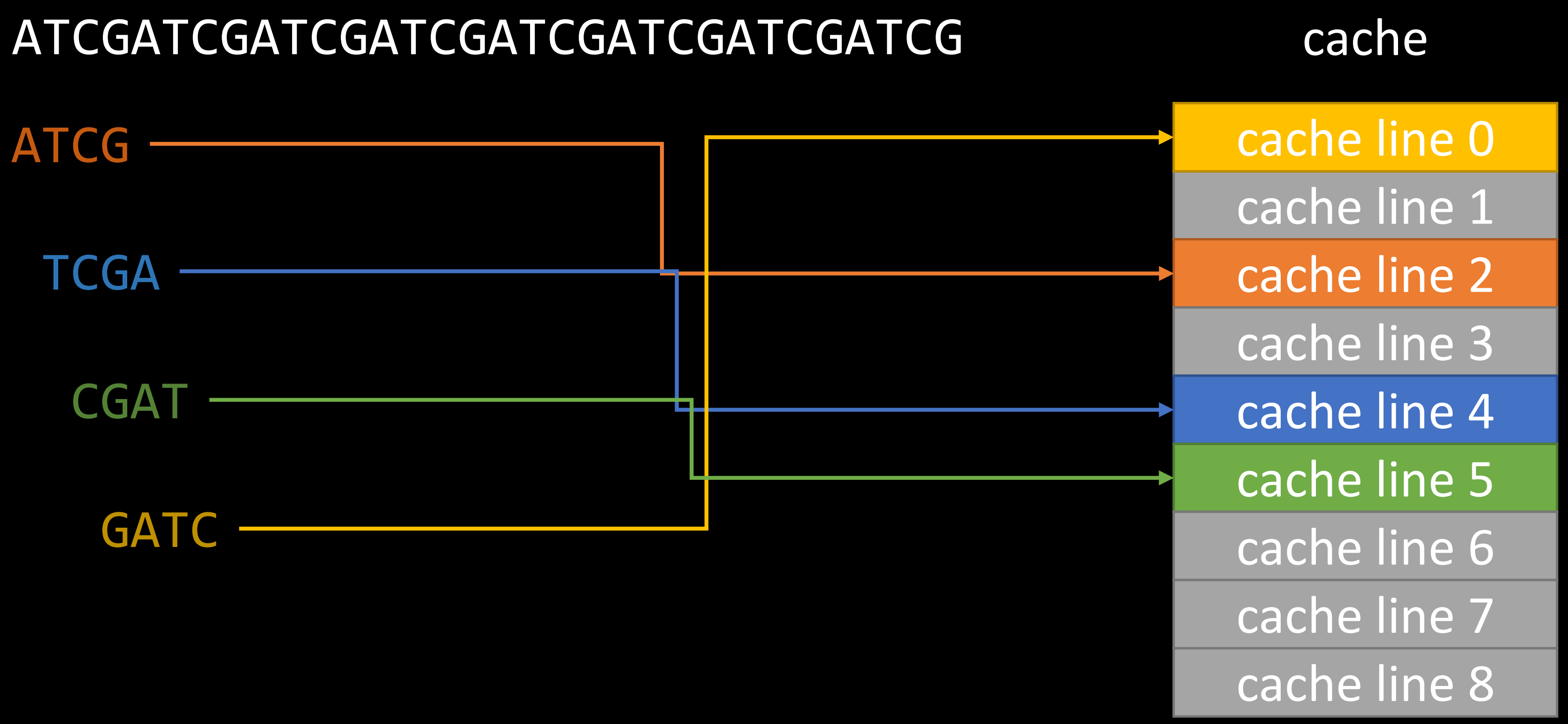
Microsatellites and Processed k-mers



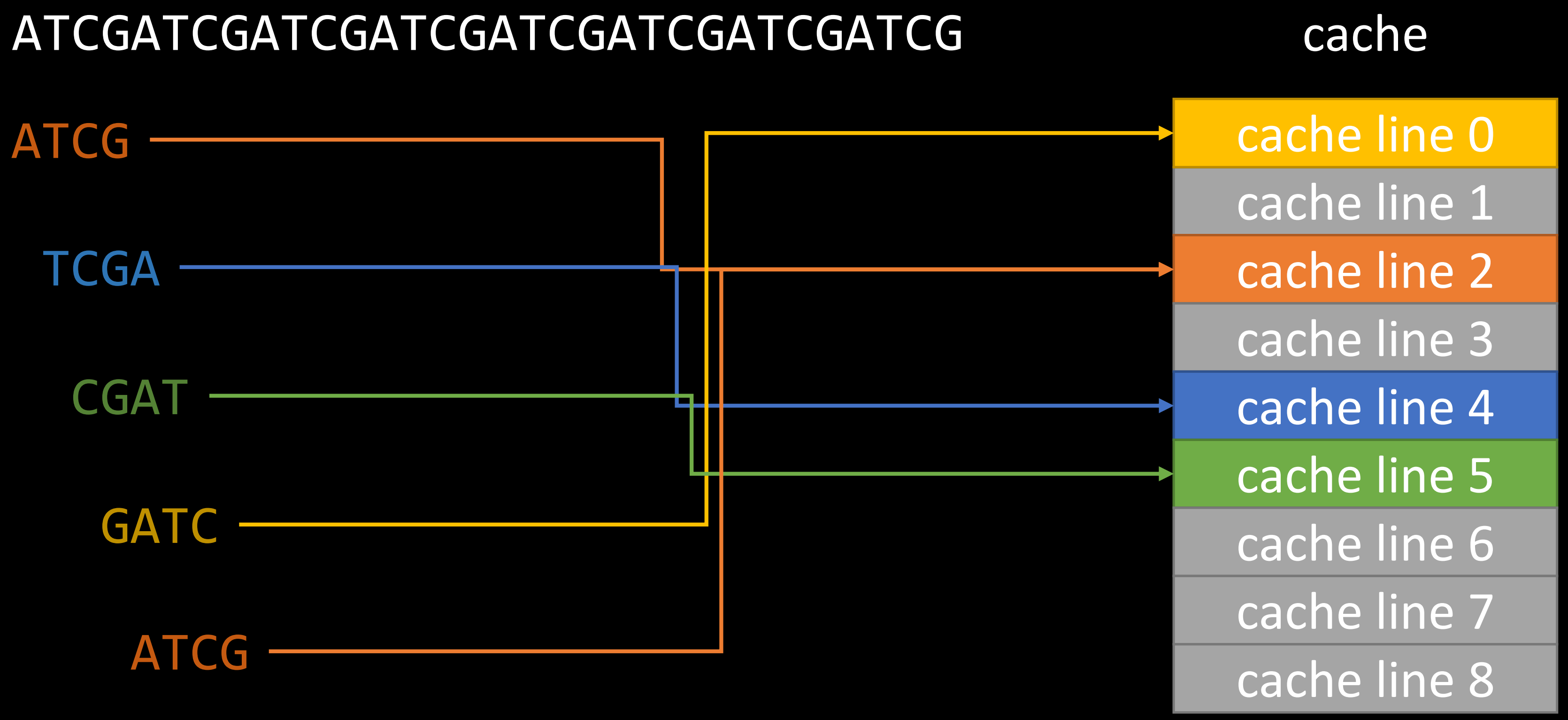
Microsatellites and Processed k-mers



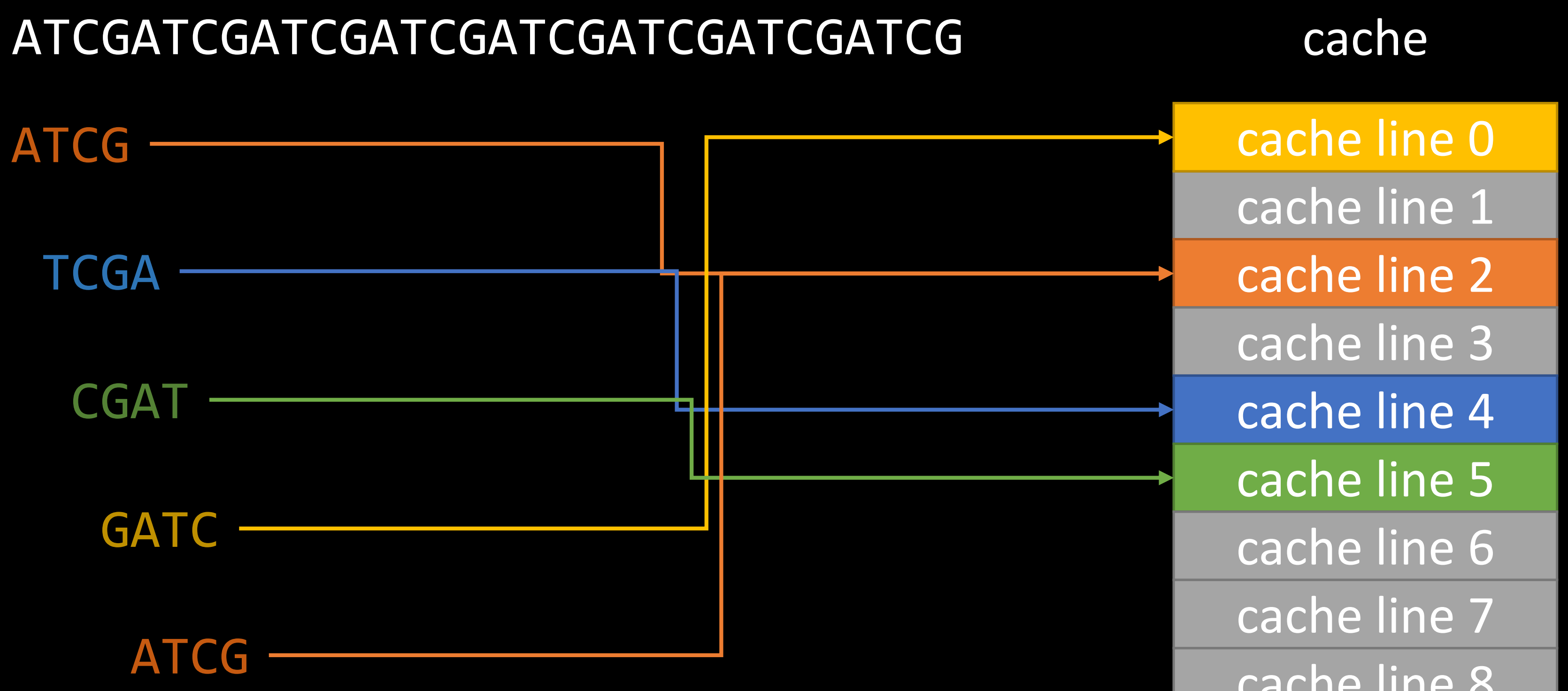
Microsatellites and Processed k-mers



Microsatellites and Processed k-mers



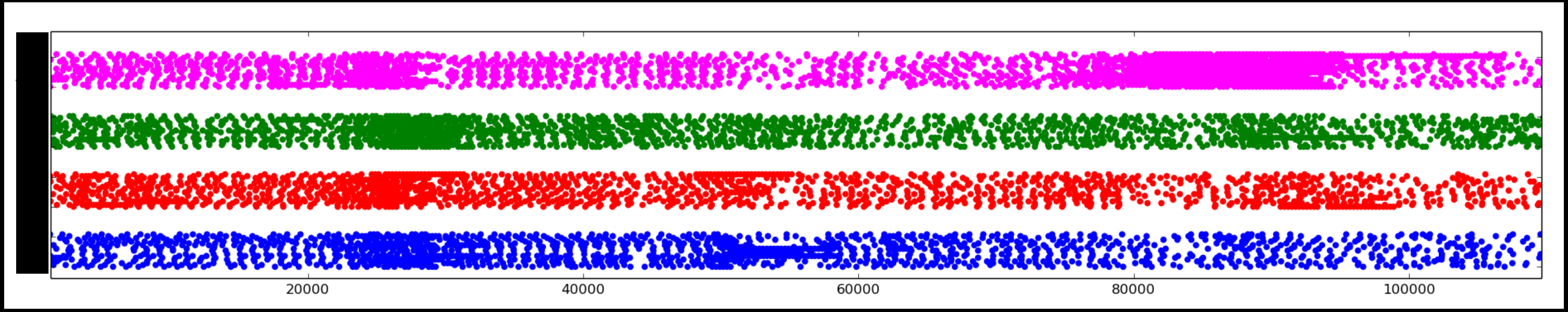
Microsatellites and Processed k-mers



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

Genome Sequencing Attack Results

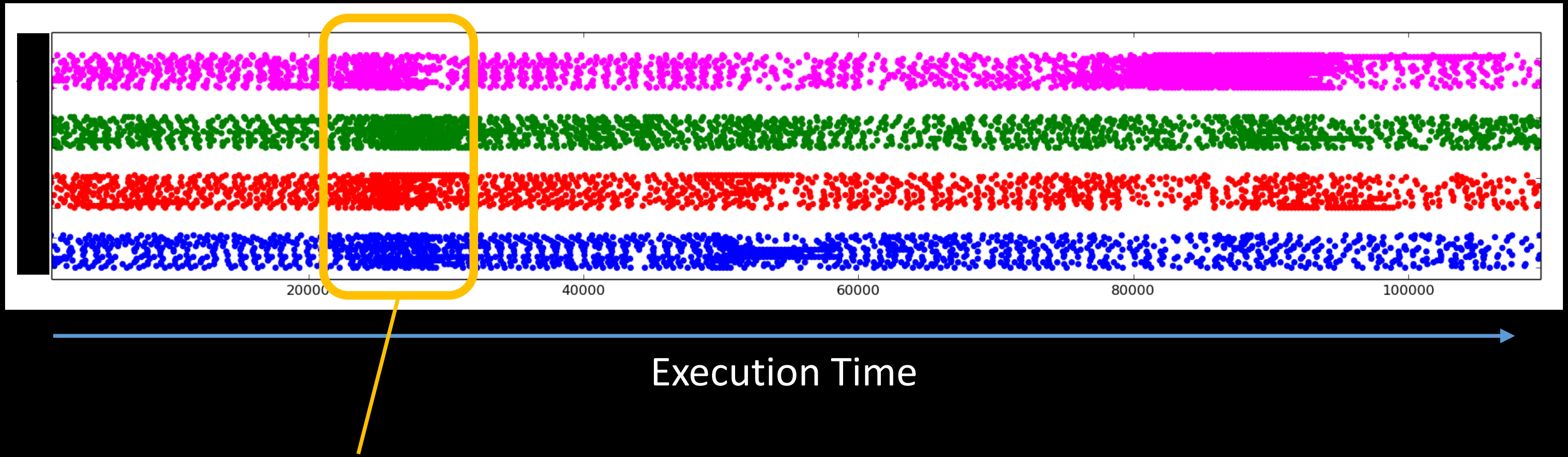
- Monitor cache lines associated to satellite
- High activity in cache lines reveal occurrence of satellite in input string



Execution Time

Genome Sequencing Attack Results

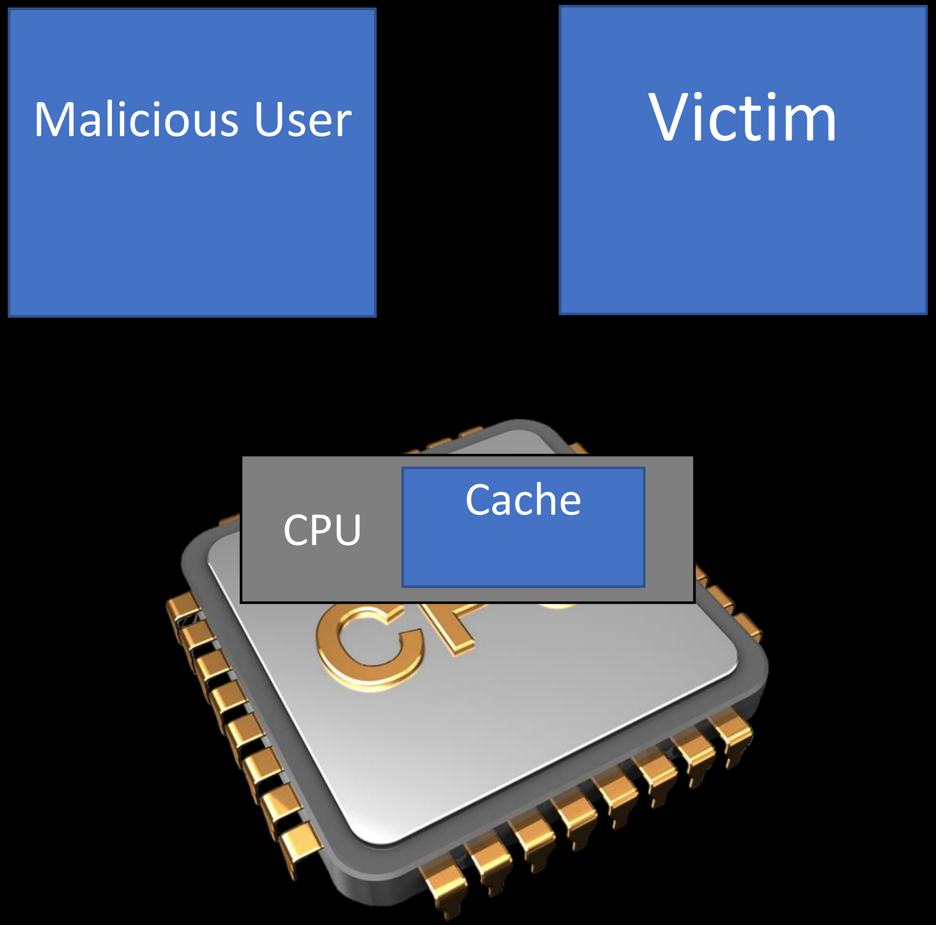
- Monitor cache lines associated to satellite
- High activity in cache lines reveal occurrence of satellite in input string



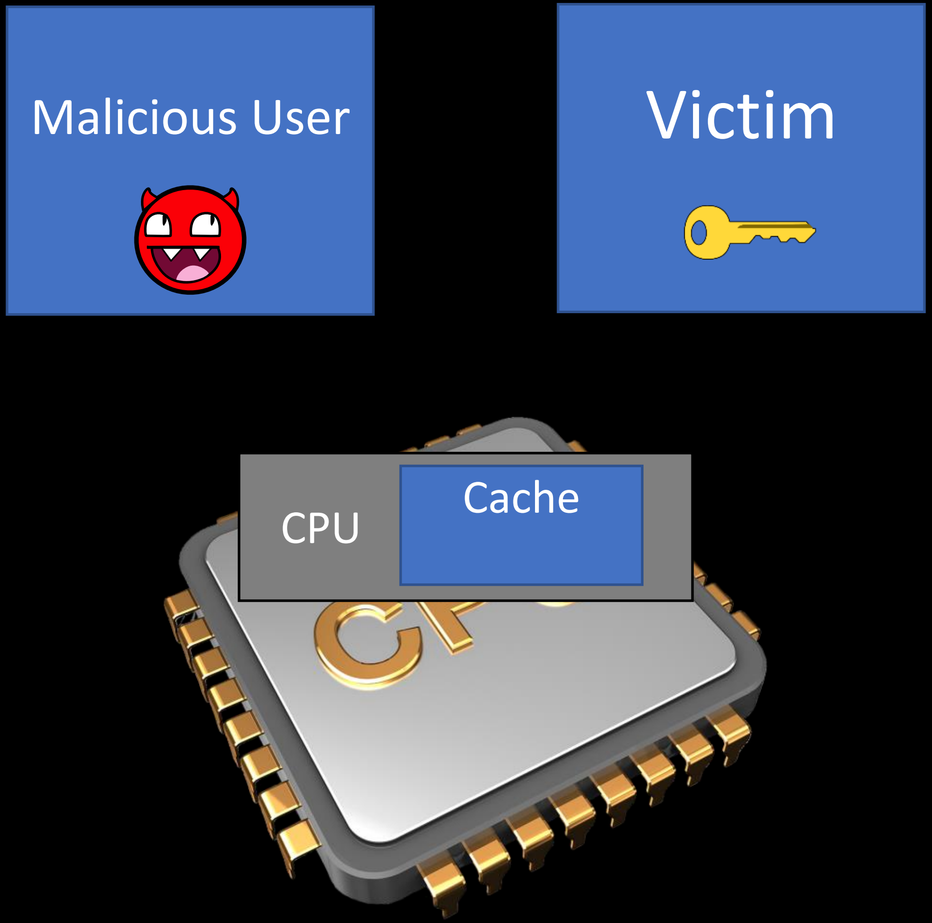
Activity in all related cache lines

Speculative Execution Attacks

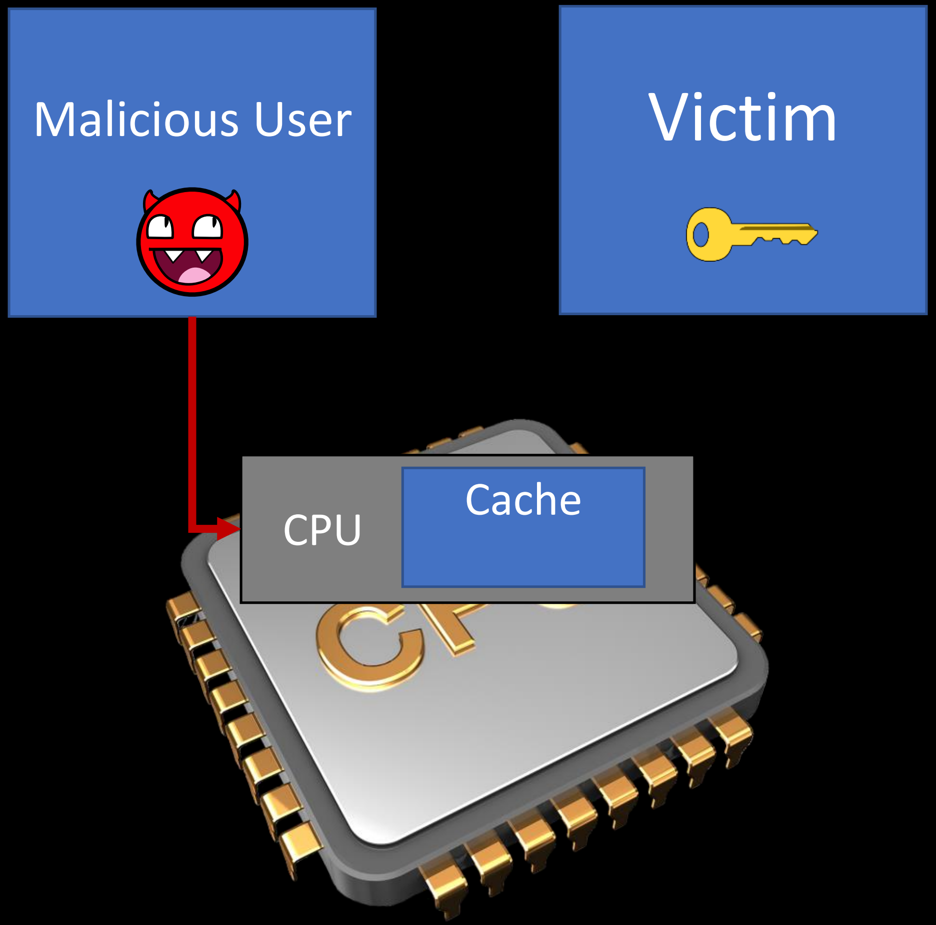
Speculative Execution Bug



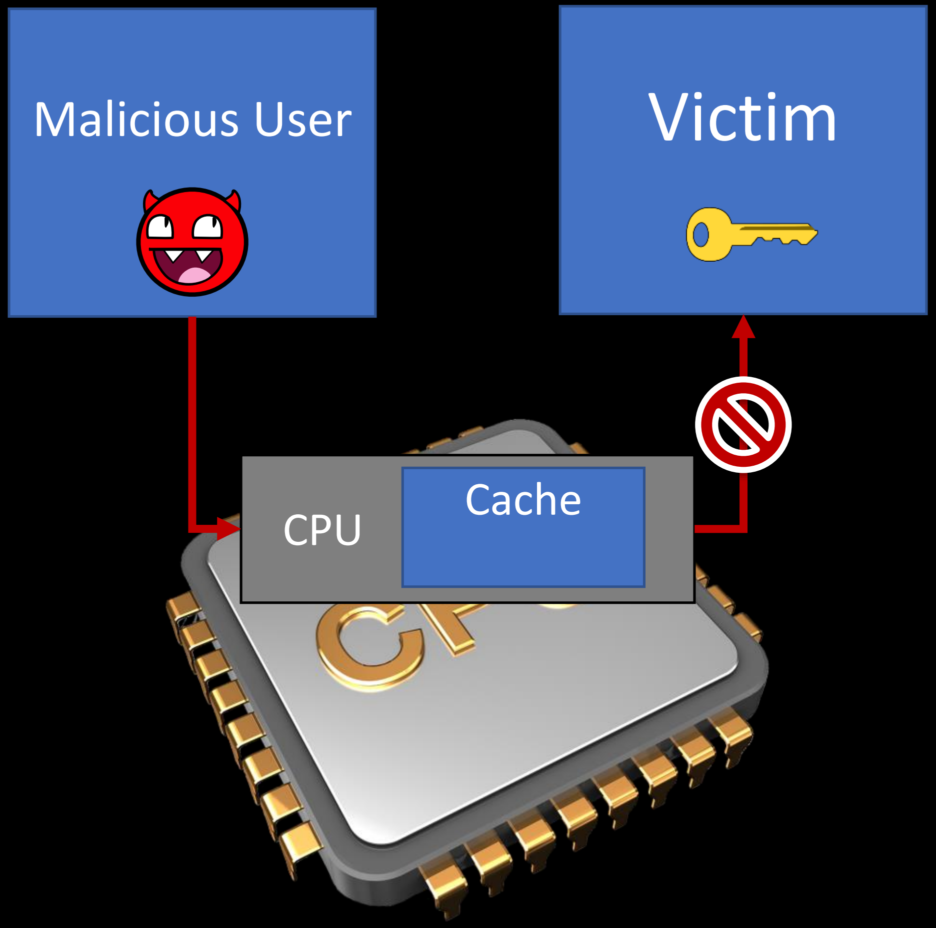
Speculative Execution Bug



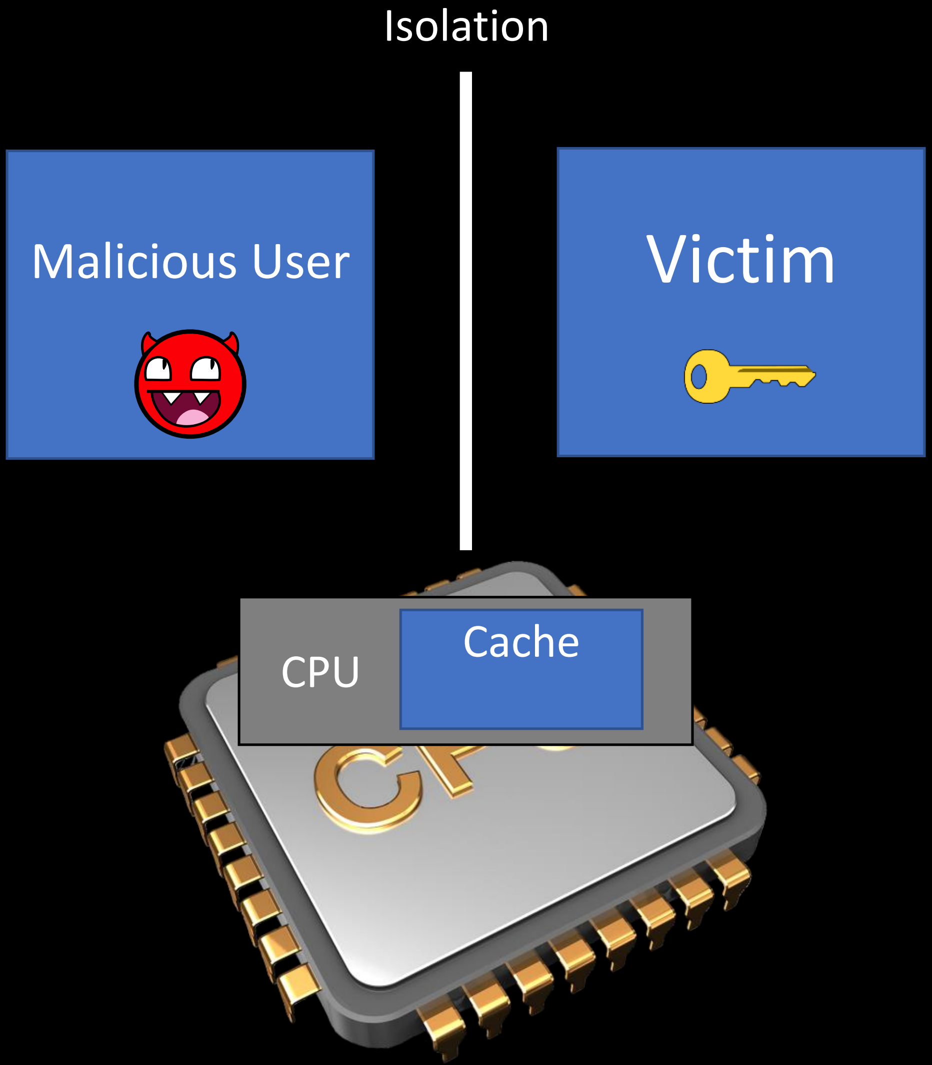
Speculative Execution Bug



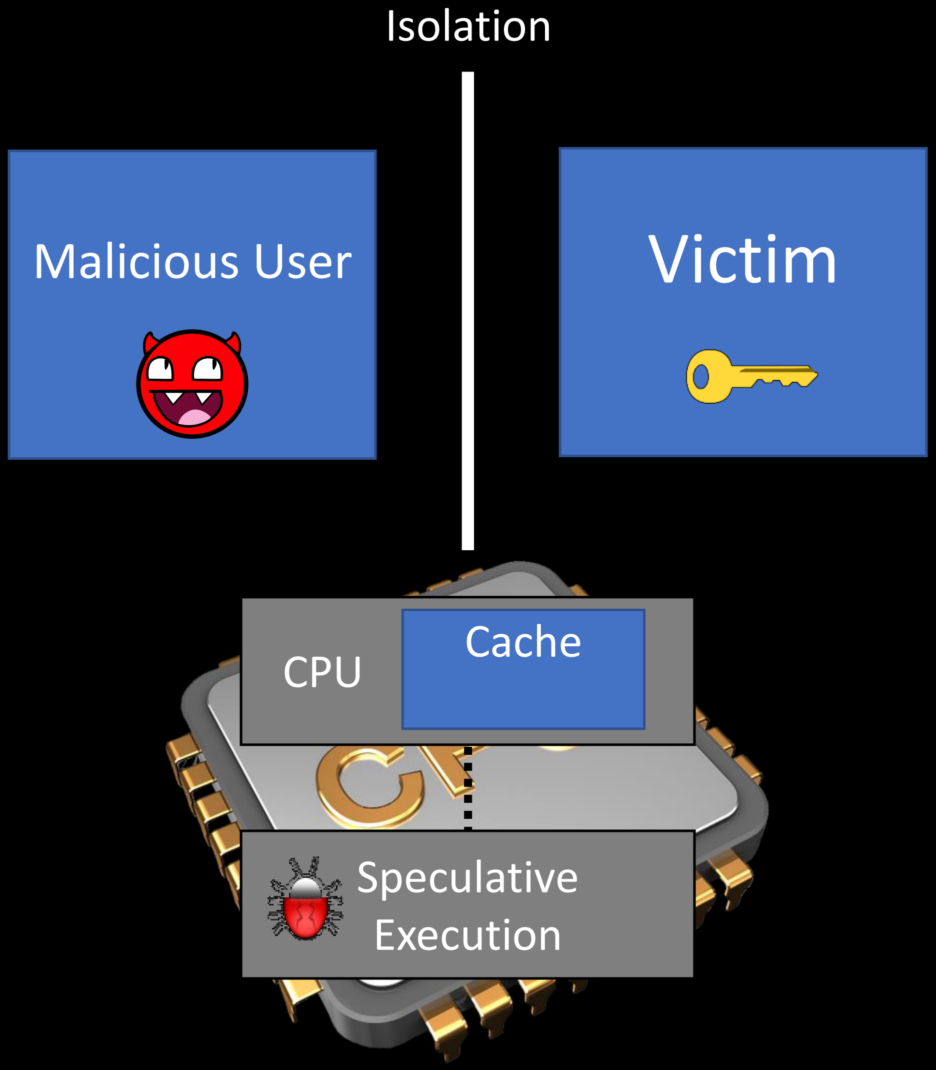
Speculative Execution Bug



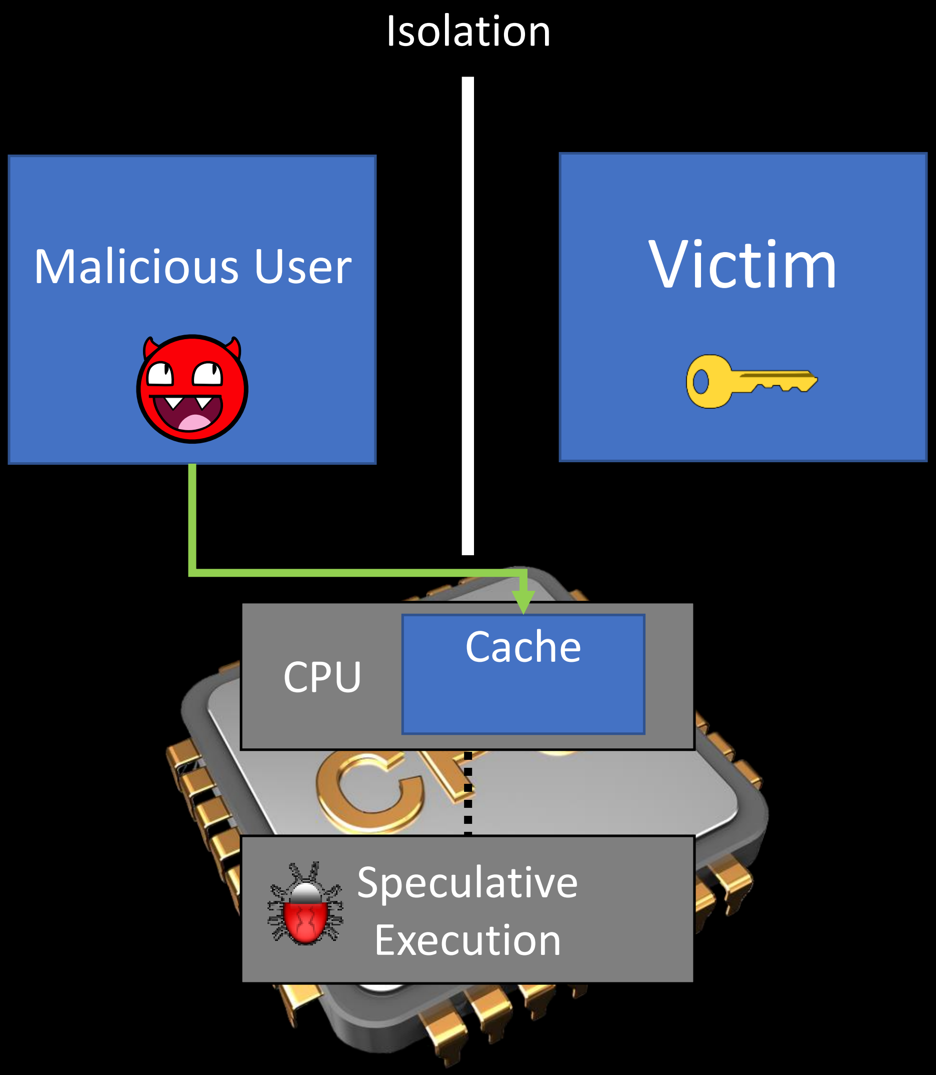
Speculative Execution Bug



Speculative Execution Bug



Speculative Execution Bug



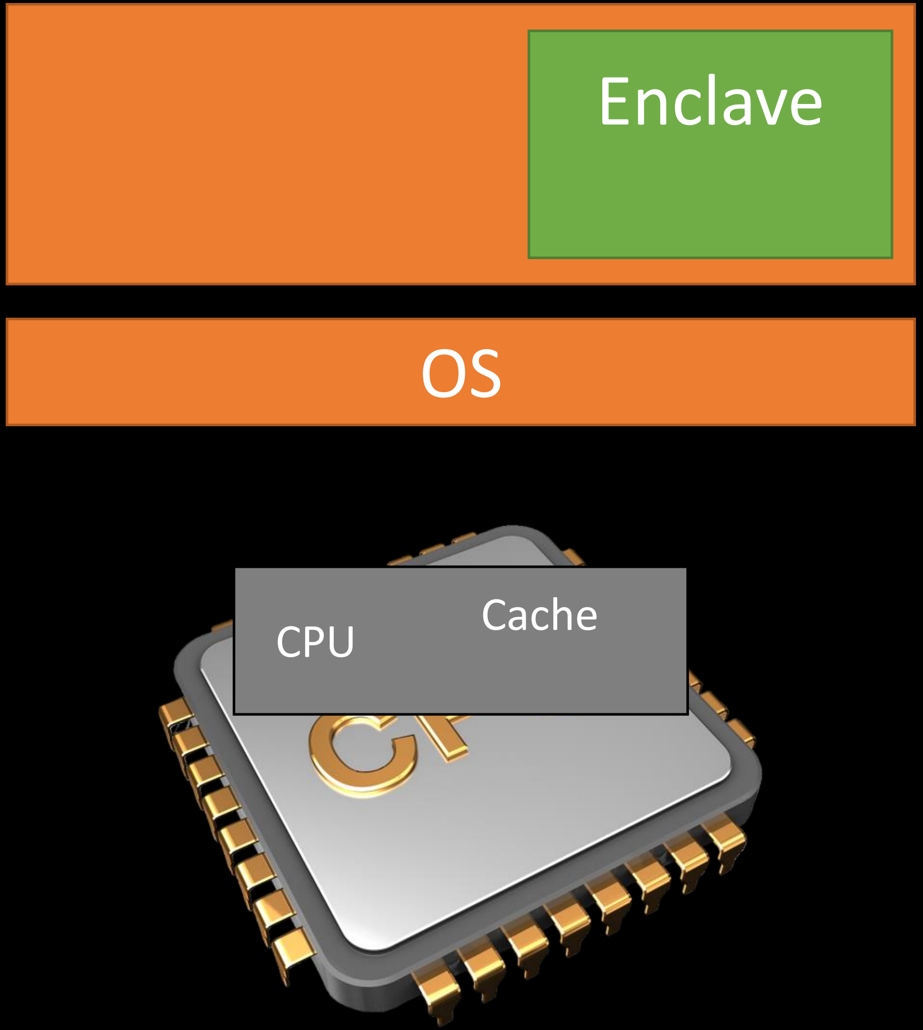
Meltdown

- Exploits speculative execution bug
  - attacker can read arbitrary physical memory (including kernel memory) from an unprivileged user process
  - this can be used, e.g., to break kernel ASLR from unprivileged process
  - or, to extract secrets from Intel SGX enclaves!

Foreshadow: Meltdown against SGX

- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache

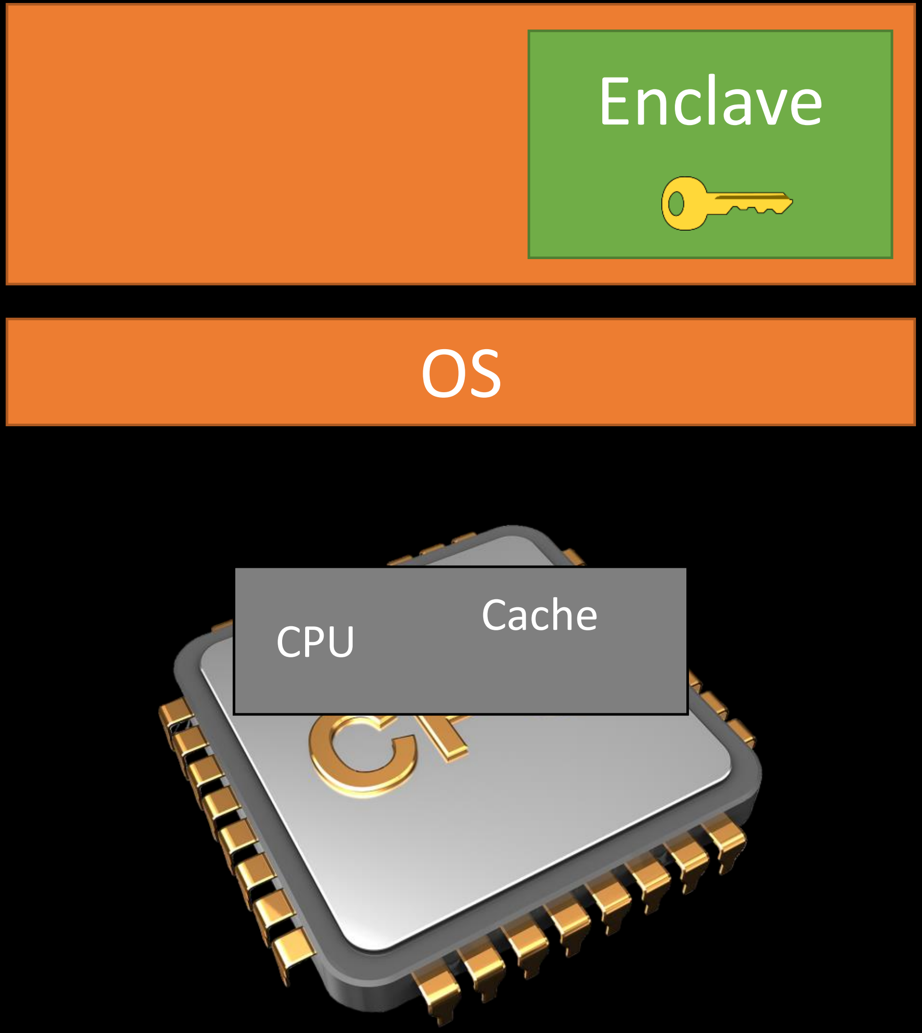
- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties



Foreshadow: Meltdown against SGX

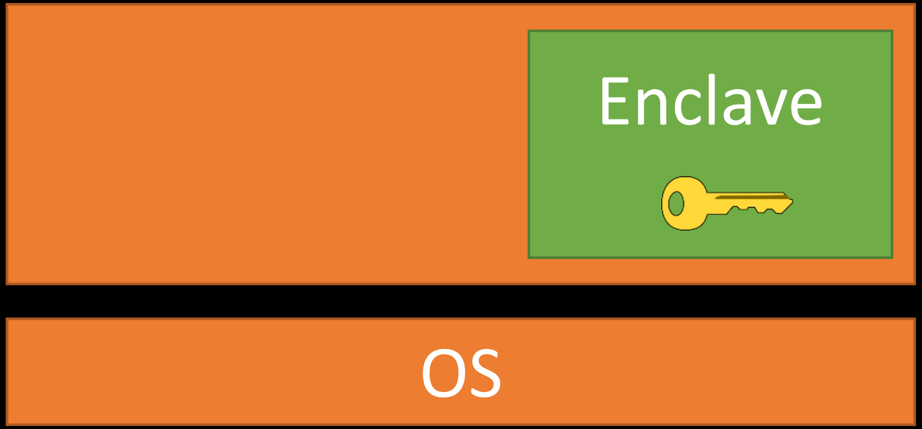
- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache

- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties

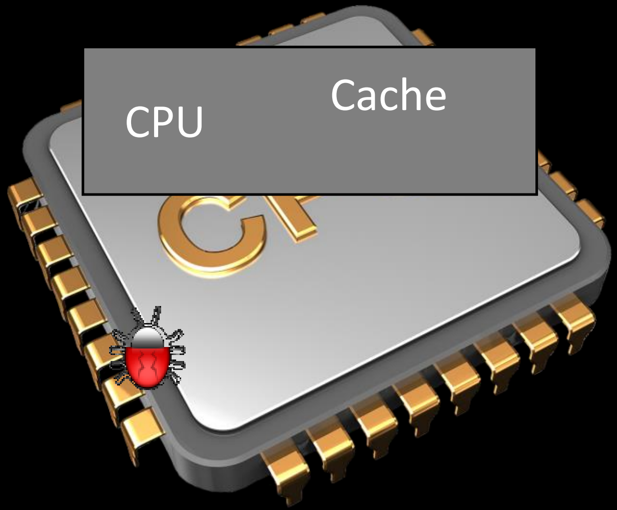


Foreshadow: Meltdown against SGX

- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache

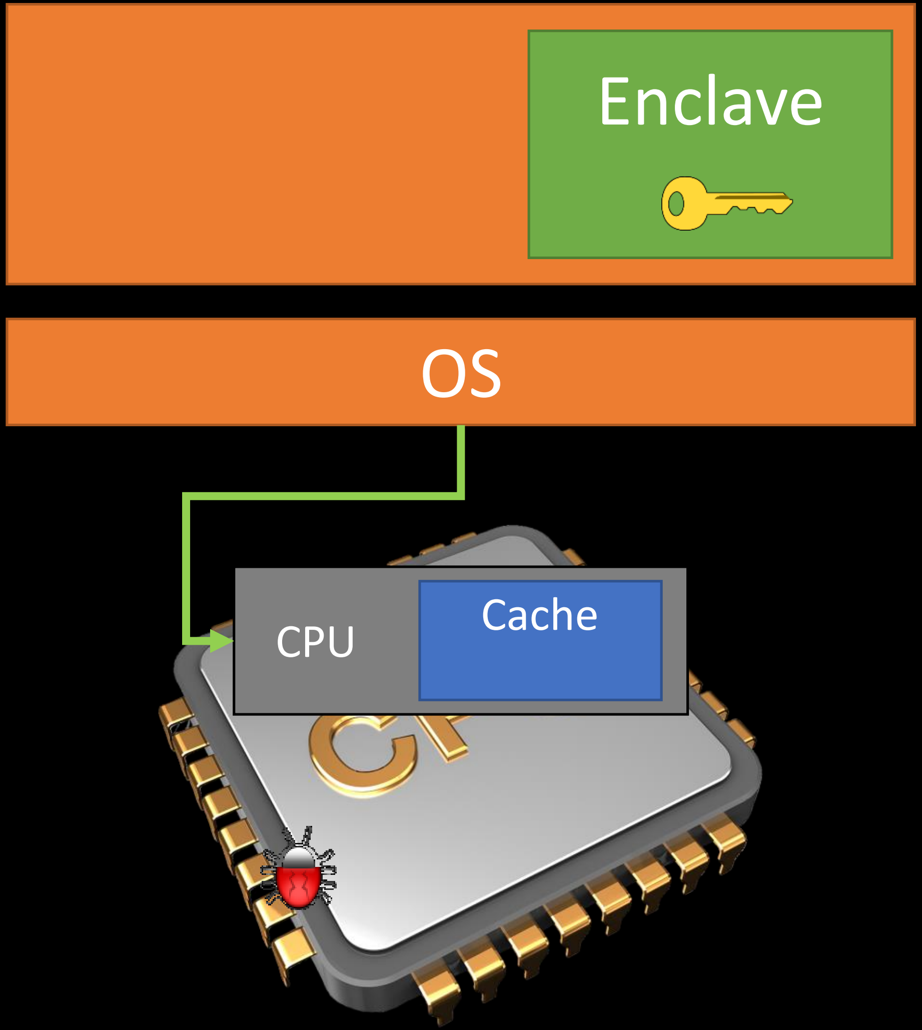


- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties



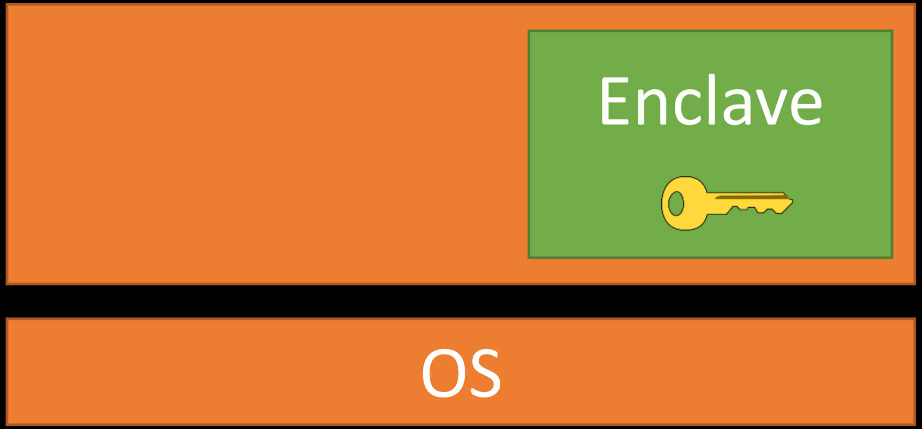
Foreshadow: Meltdown against SGX

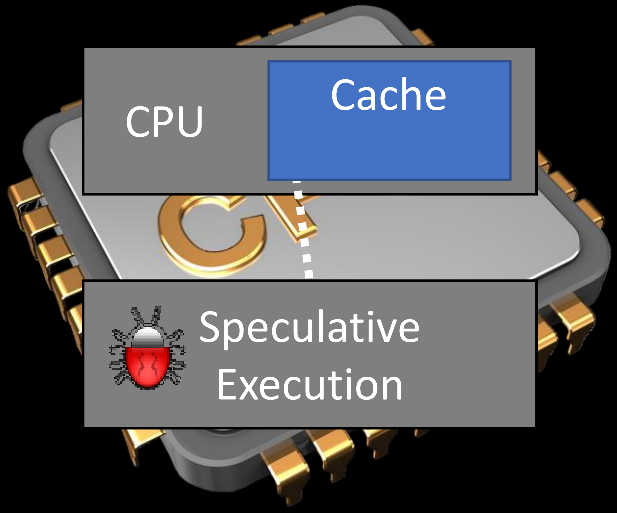
- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache
- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties



Foreshadow: Meltdown against SGX

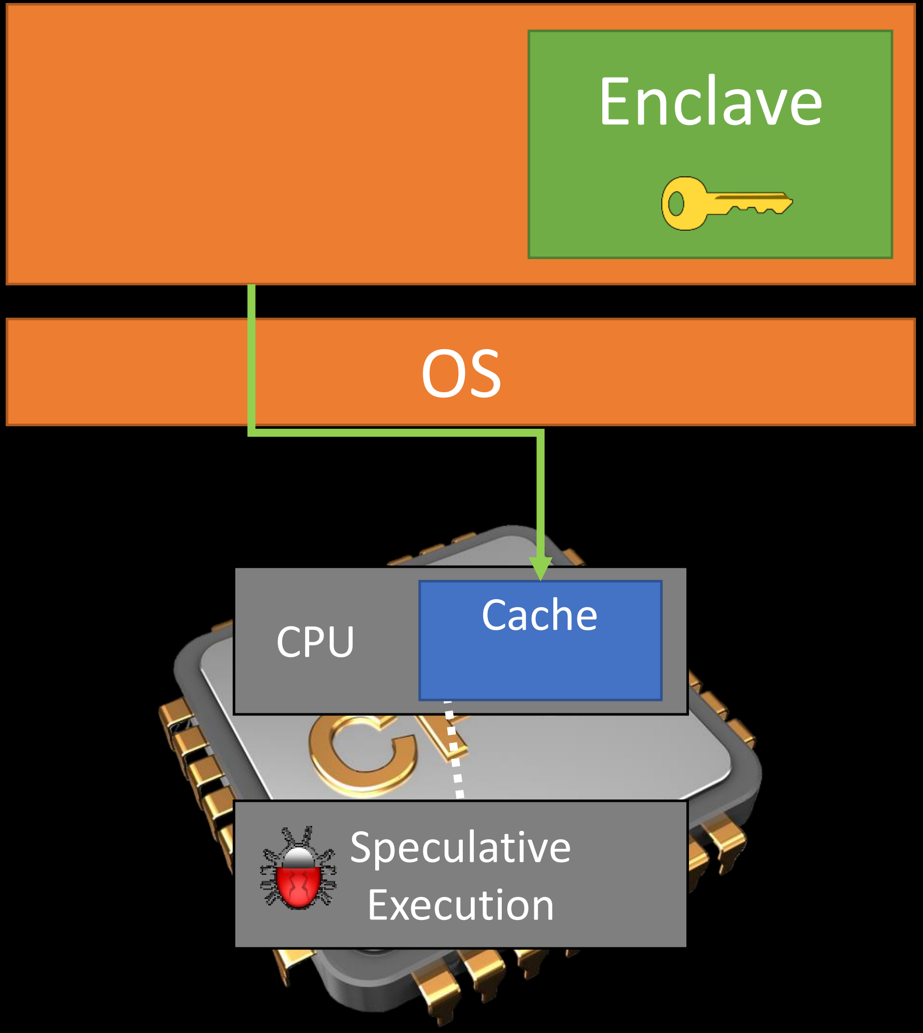
- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache
- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties





Foreshadow: Meltdown against SGX

- Foreshadow [Van Bulck, USENIX Security 2018]
  - Extract long-term secrets from Intel **Launching** and **Quoting** Enclaves
  - Speculative access only possible for data in L1 cache
- Implications
  - Attacker can bypass vetting of enclaves by Intel
  - Attacker can forge local and remote attestations sent to other enclaves and to remote parties



How to Get Enclave Data into L1 Cache?

- Run enclave and interrupt when target data was used
  - The enclave's usage of the target data brings it into the cache

- Use SGX paging mechanism
  - OS can swap in/out pages of enclaves
  - When an enclave page is swapped in, its content is loaded into L1 cache
  - Malicious OS can run attack without even running the enclave

Defenses Against Foreshadow

- Flush L1 cache on enclave exit
  - Provided via microcode update
  - Only effective without hyperthreading

- Include hyperthreading configuration in attestation report
  - “[…] the Intel SGX attestation will indicate whether hyperthreading has been enabled by the BIOS.” [Intel*]

- Renew SGX keys
  - “The microcode update changes the Security Version Number (SVN) associated with the Intel SGX implementation and provides enclaves on the platform with new sealing and attestation keys.” [Intel*]

Defenses Against Foreshadow

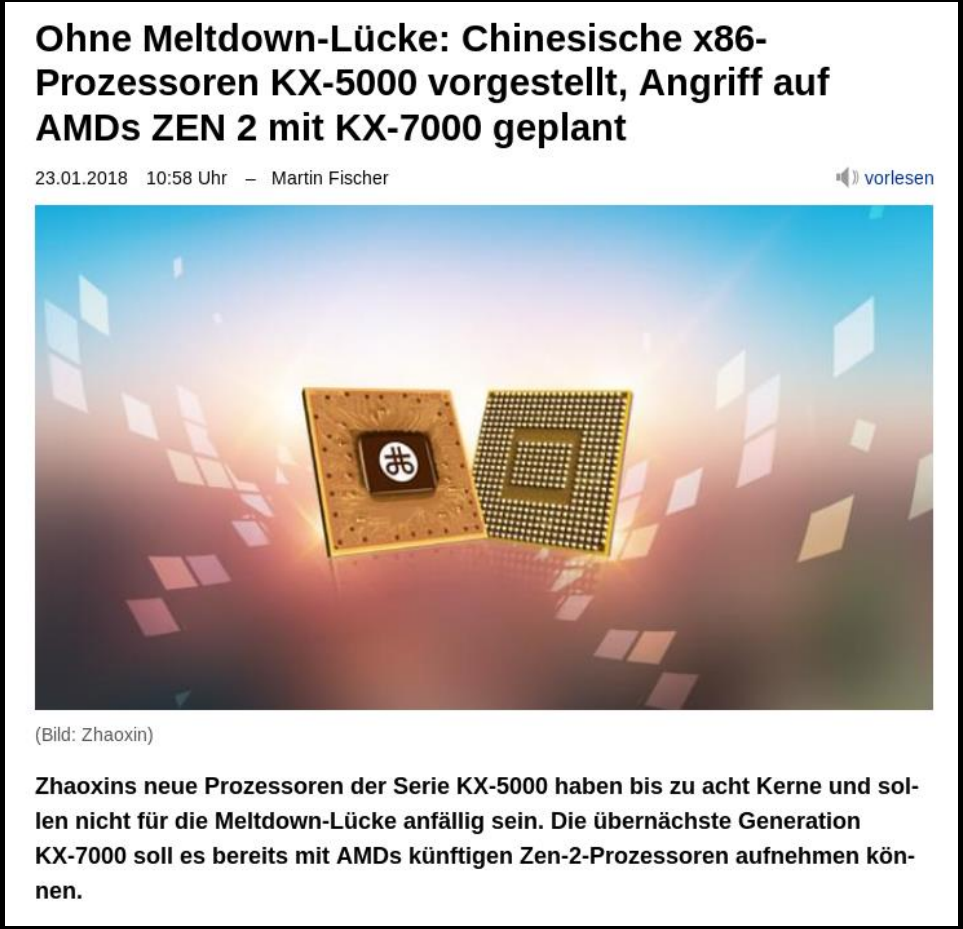
- Flush L1 cache on enclave exit
  - Provided via microcode update
  - Only effective without hyperthreading

- Include hyperthreading configuration in attestation report
  - "[…] the Intel SGX attestation will indicate whether hyperthreading has been enabled by the BIOS." [Intel*]

- Renew SGX keys
  - "The microcode update changes the Security Version Number (SVN) associated with the Intel SGX implementation and provides enclaves on the platform with new sealing and attestation keys." [Intel*]

* https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault

Alternative Solutions?



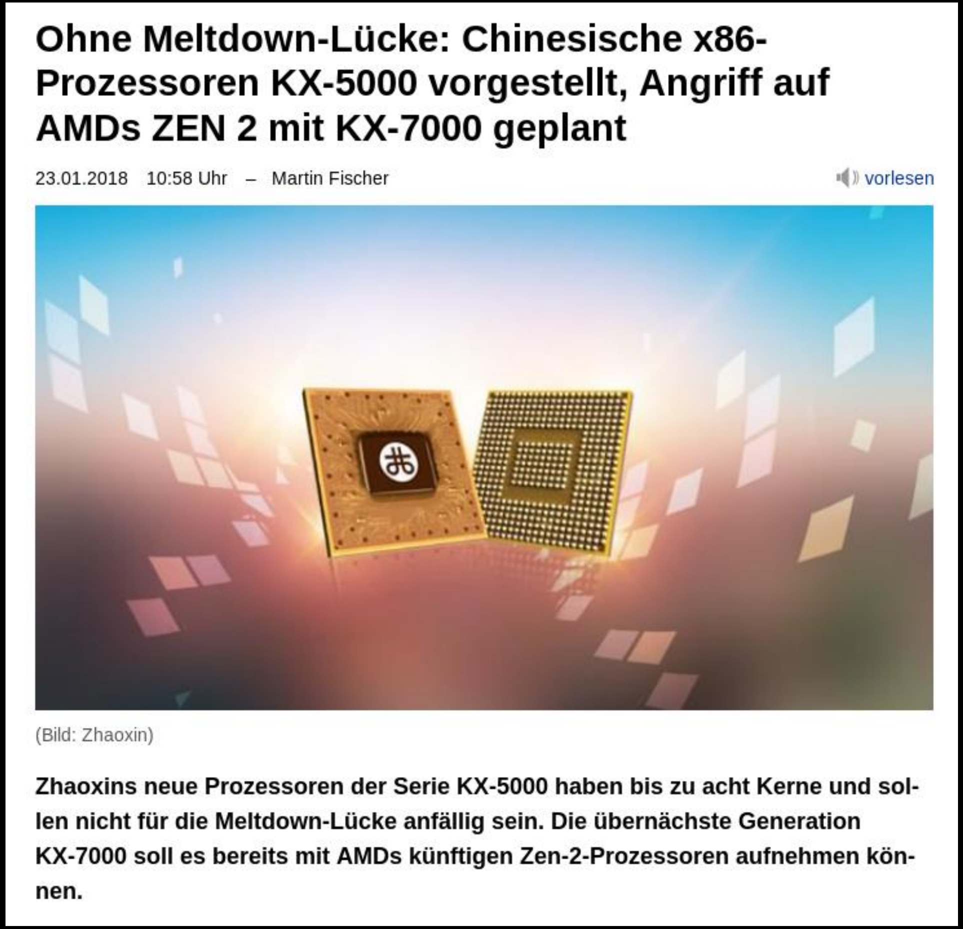
Ohne Meltdown-Lücke: Chinesische x86-Prozessoren KX-5000 vorgestellt, Angriff auf AMDs ZEN 2 mit KX-7000 geplant

23.01.2018 10:58 Uhr – Martin Fischer vorlesen

(Bild: Zhaoxin)

Zhaoxins neue Prozessoren der Serie KX-5000 haben bis zu acht Kerne und sollen nicht für die Meltdown-Lücke anfällig sein. Die übernächste Generation KX-7000 soll es bereits mit AMDs künftigen Zen-2-Prozessoren aufnehmen können.

Alternative Solutions?



Ohne Meltdown-Lücke: Chinesische x86-Prozessoren KX-5000 vorgestellt, Angriff auf AMDs ZEN 2 mit KX-7000 geplant

23.01.2018 10:58 Uhr – Martin Fischer

vorlesen

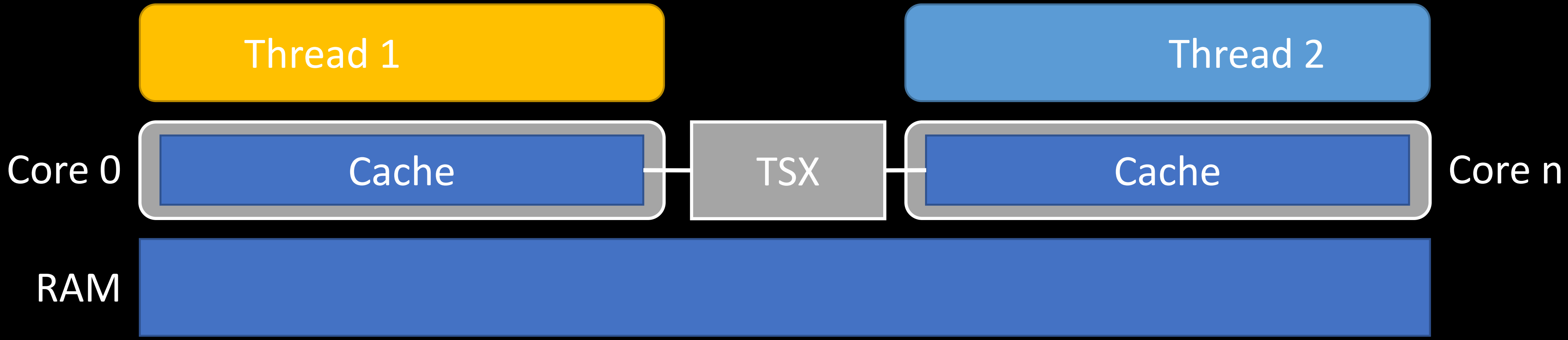
(Bild: Zhaoxin)

Zhaoxins neue Prozessoren der Serie KX-5000 haben bis zu acht Kerne und sollen nicht für die Meltdown-Lücke anfällig sein. Die übernächste Generation KX-7000 soll es bereits mit AMDs künftigen Zen-2-Prozessoren aufnehmen können.

Buy Chinese Quality Chips, not cheap American copies!

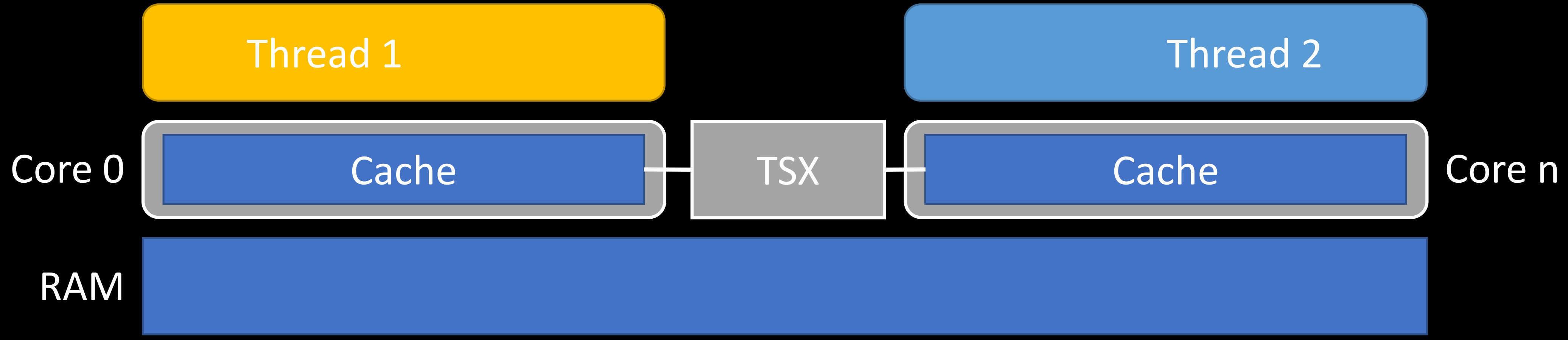
Side-Channel Defenses Using TSX

Intel TSX



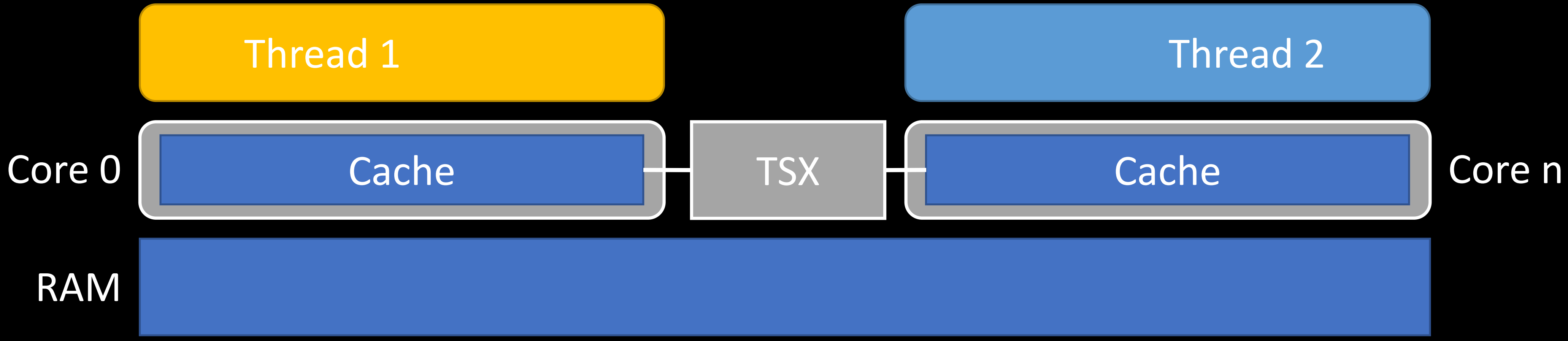
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



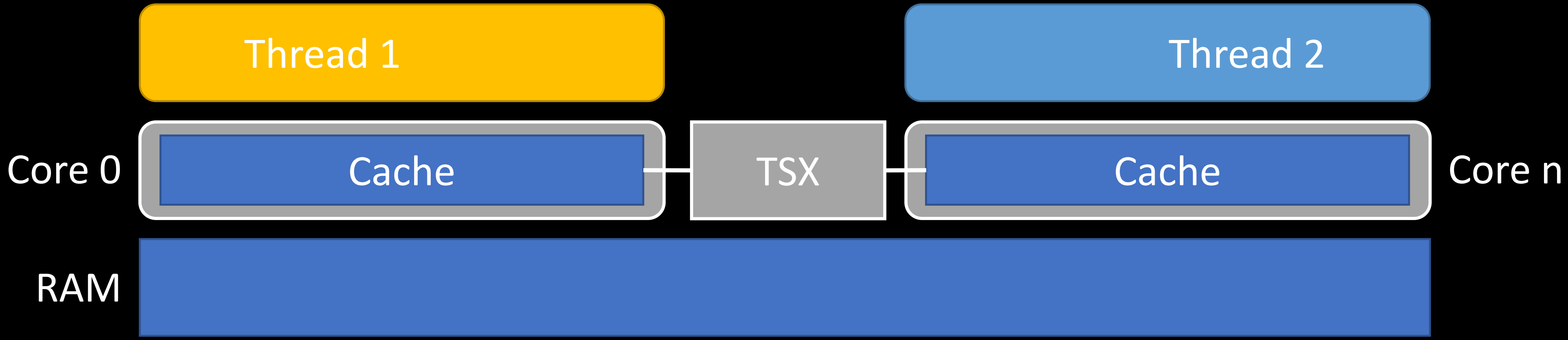
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



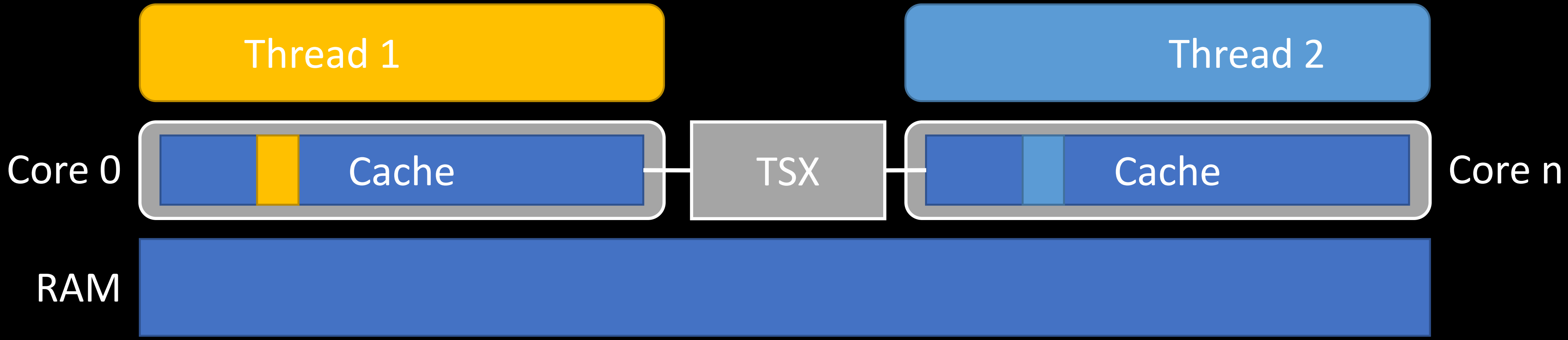
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



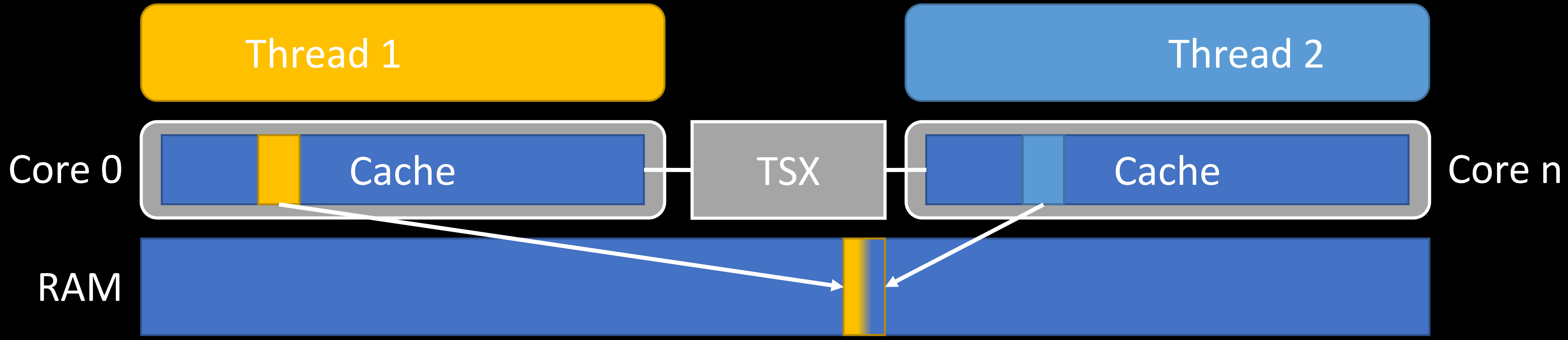
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



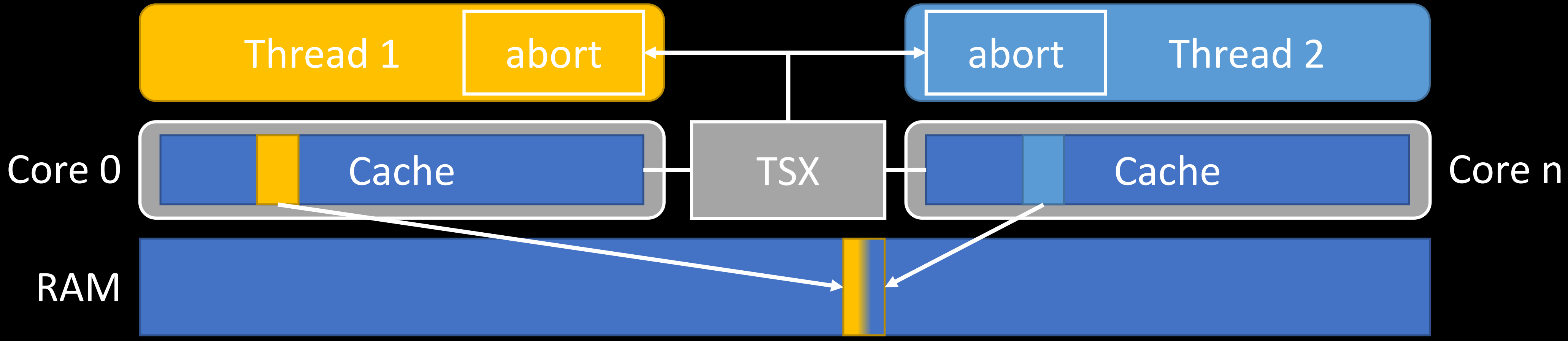
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



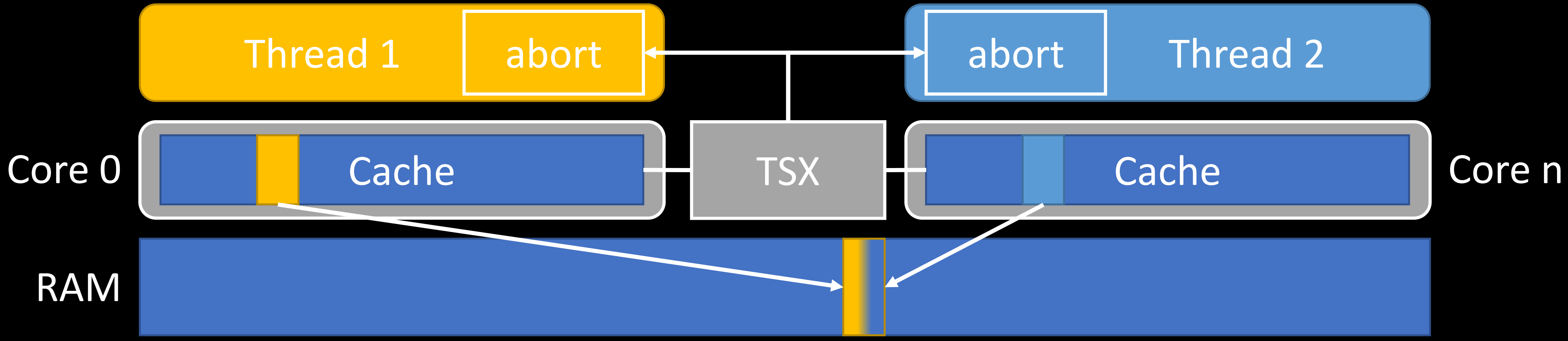
Intel TSX

- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



Intel TSX

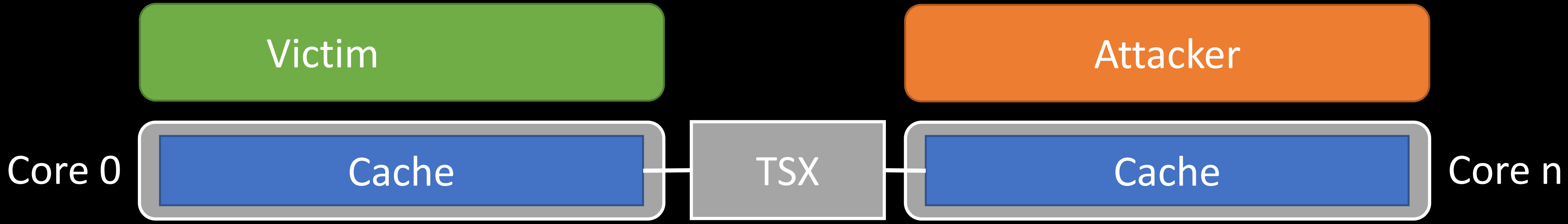
- Intel implementation of Hardware Transactional Memory (HTM)
- Designed for high-performance concurrency
- Allows synchronous memory transactions
- TSX is **not** available on all SGX-enable processors



SGX Specific Side-Channel Defenses Using TSX

Detecting enclave's interruption

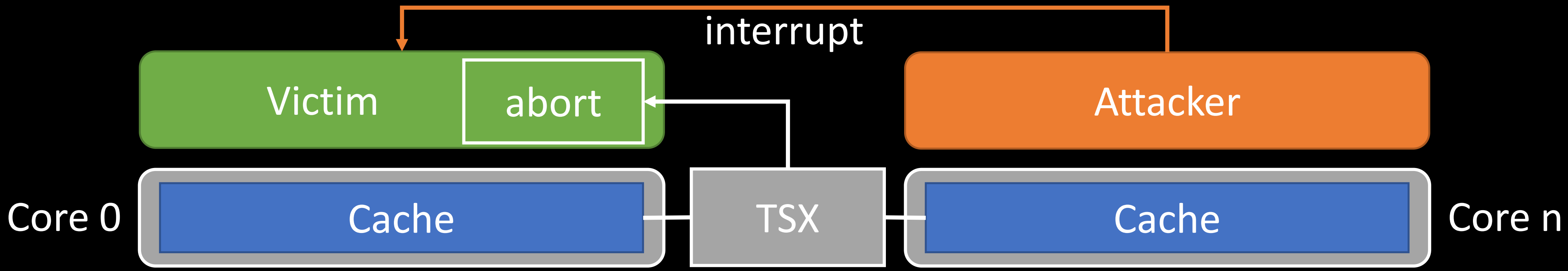
- Frequent interrupts evidence for side-channel attack
- T-SGX: Uses TSX feature to detect enclave interrupt [Shih et al., NDSS'17]
- Déjà Vu : Uses TSX to detect enclave slowdown [Chen et al., AsiaCCS'17]



SGX Specific Side-Channel Defenses Using TSX

Detecting enclave's interruption

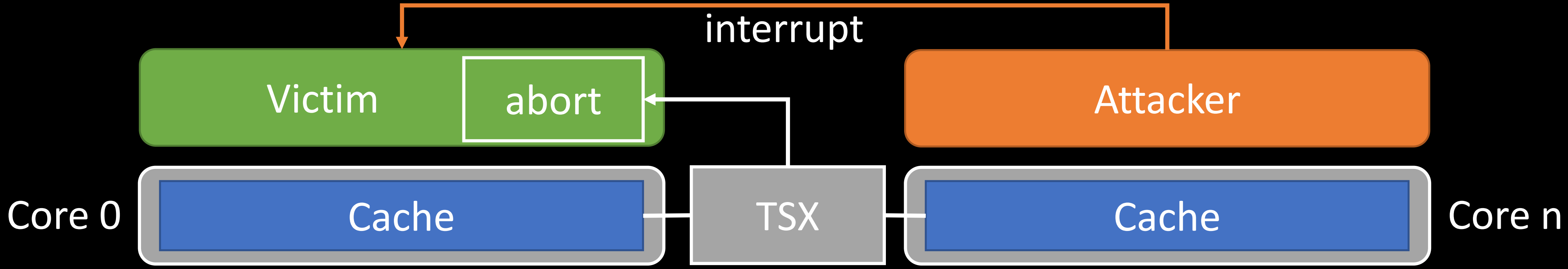
- Frequent interrupts evidence for side-channel attack
- T-SGX: Uses TSX feature to detect enclave interrupt [Shih et al., NDSS'17]
- Déjà Vu : Uses TSX to detect enclave slowdown [Chen et al., AsiaCCS'17]



SGX Specific Side-Channel Defenses Using TSX

Detecting enclave's interruption

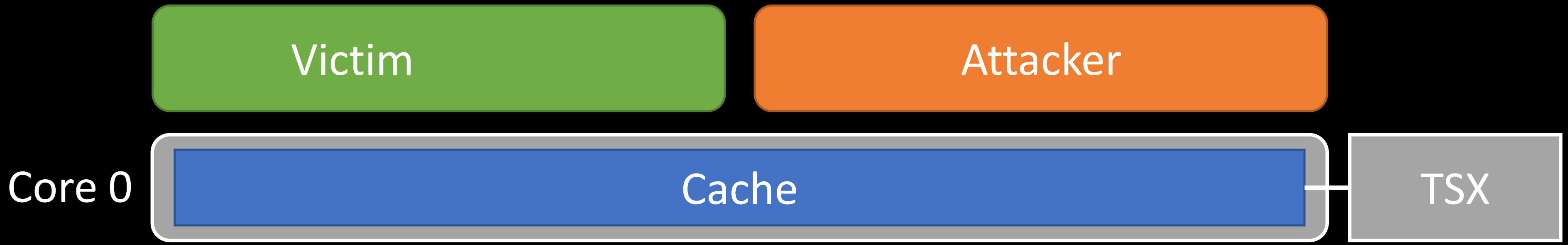
- Frequent interrupts evidence for side-channel attack
- T-SGX: Uses TSX feature to detect enclave interrupt [Shih et al., NDSS'17]
- Déjà Vu : Uses TSX to detect enclave slowdown [Chen et al., AsiaCCS'17]



SGX Specific Side-Channel Defenses Using TSX

Detecting cache evictions

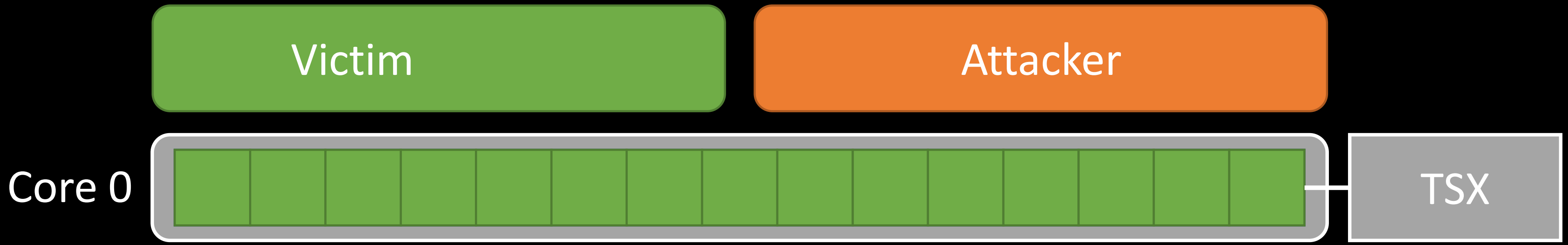
- Eviction of the victim’s cache entries could lead to information leakage
- Cloak: Prime cache before accessing sensitive data [Schuster et al., USENIX 2017]



SGX Specific Side-Channel Defenses Using TSX

Detecting cache evictions

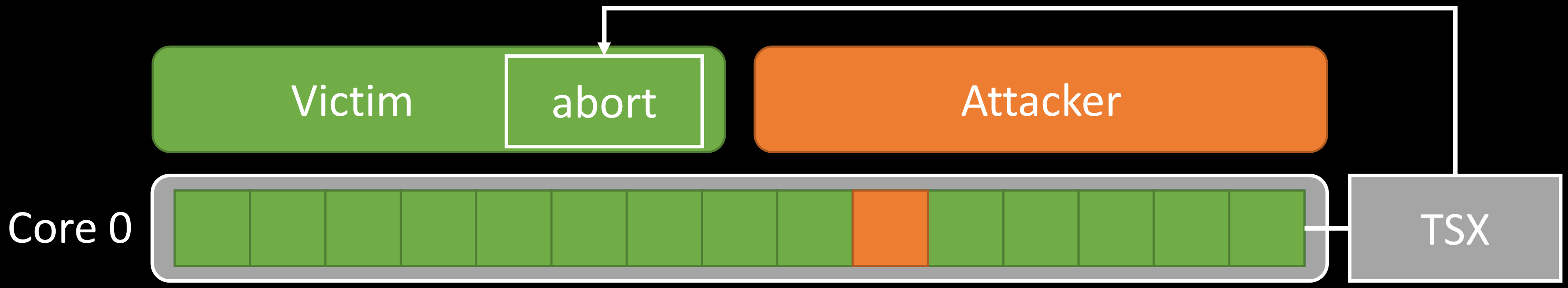
- Eviction of the victim's cache entries could lead to information leakage
- Cloak: Prime cache before accessing sensitive data [Schuster et al., USENIX 2017]



SGX Specific Side-Channel Defenses Using TSX

Detecting cache evictions

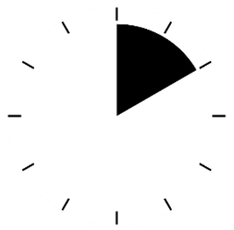
- Eviction of the victim's cache entries could lead to information leakage
- Cloak: Prime cache before accessing sensitive data [Schuster et al., USENIX 2017]



General Hardware-based Side-Channel Defenses

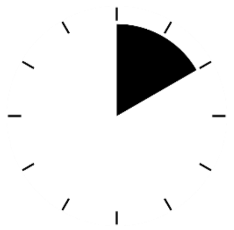
General Hardware-based Side-Channel Defenses

Temporal cache isolation

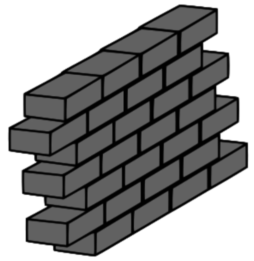


General Hardware-based Side-Channel Defenses

Temporal cache isolation

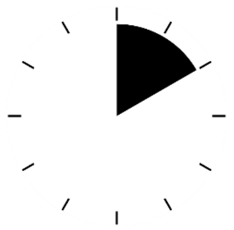


Cache partitioning / coloring

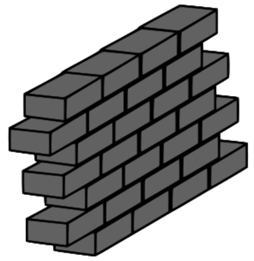


General Hardware-based Side-Channel Defenses

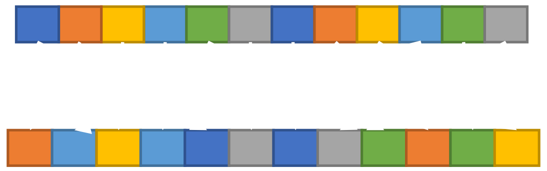
Temporal cache isolation



Cache partitioning / coloring

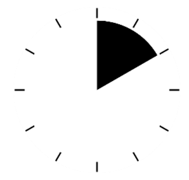


Randomized cache mappings

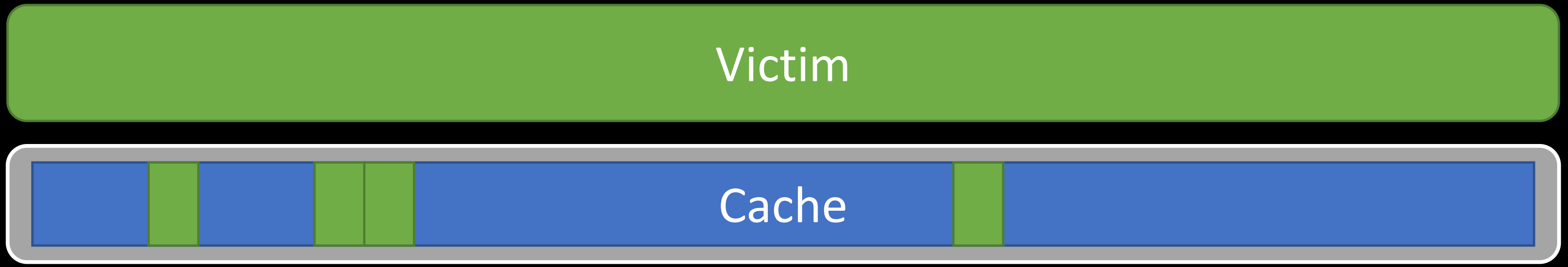


Temporal Cache Isolation

Temporal cache isolation

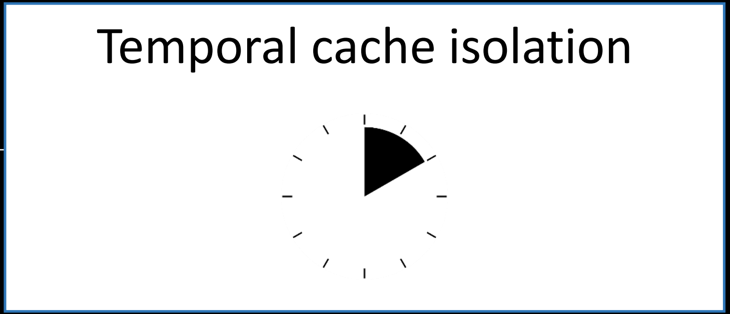


- Flush on each context switch
- Ineffective on SMT-enabled systems where caches are shared contemporaneously
- E.g., [Costan et al., USENIX Sec’16]

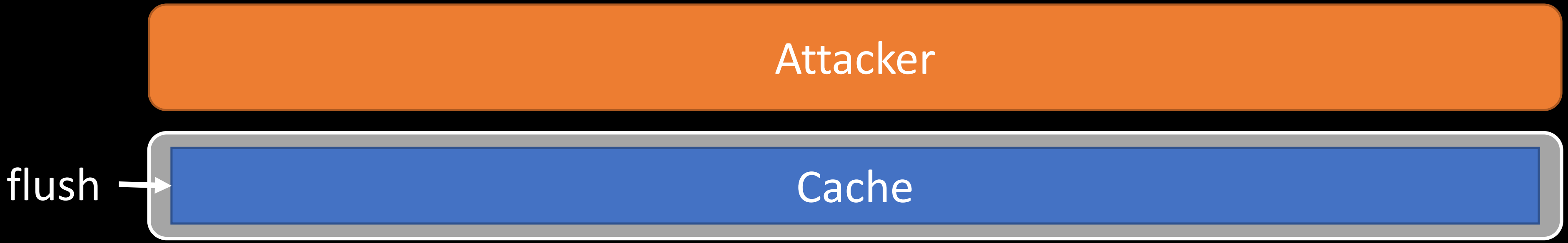


SMT: Simultaneous Multithreading

Temporal Cache Isolation



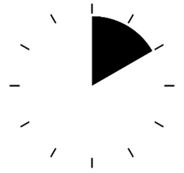
- Flush on each context switch
- Ineffective on SMT-enabled systems where caches are shared contemporaneously
- E.g., [Costan et al., USENIX Sec’16]



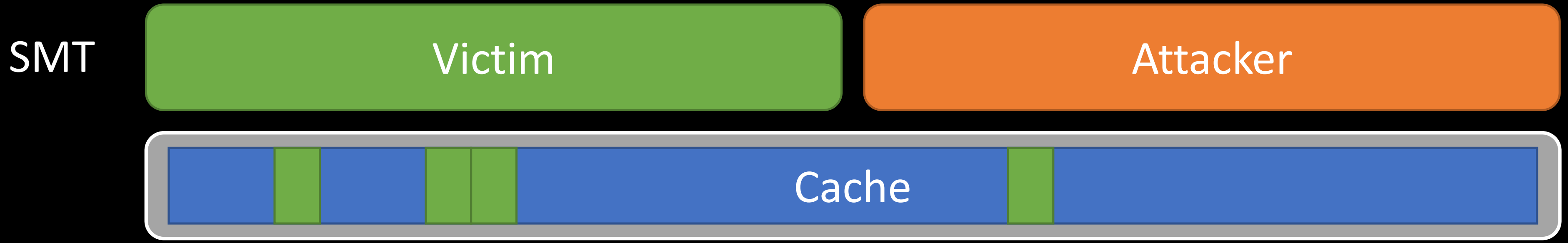
SMT: Simultaneous Multithreading

Temporal Cache Isolation

Temporal cache isolation

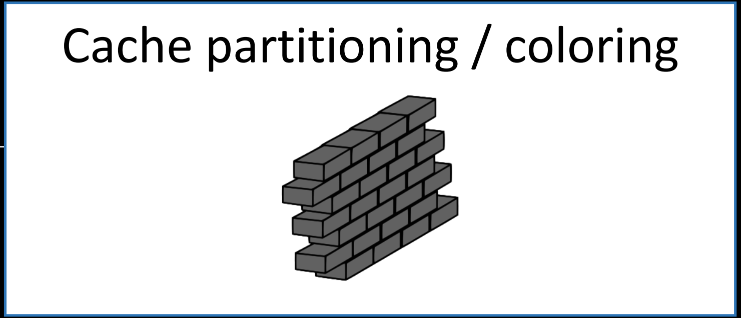


- Flush on each context switch
- Ineffective on SMT-enabled systems where caches are shared contemporaneously
- E.g., [Costan et al., USENIX Sec'16]

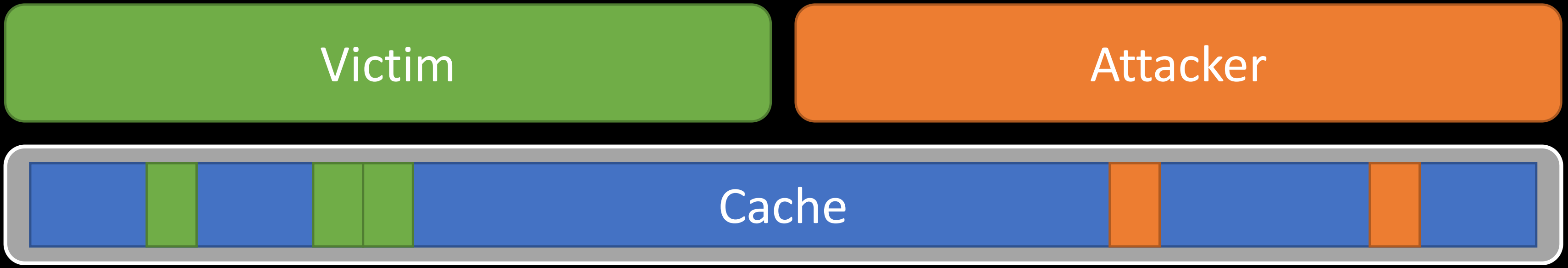


SMT: Simultaneous Multithreading

Cache Partitioning / Coloring

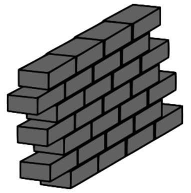


- Reduces the amount of cache available to individual software
- E.g., [Domnister et al., TACO’12]

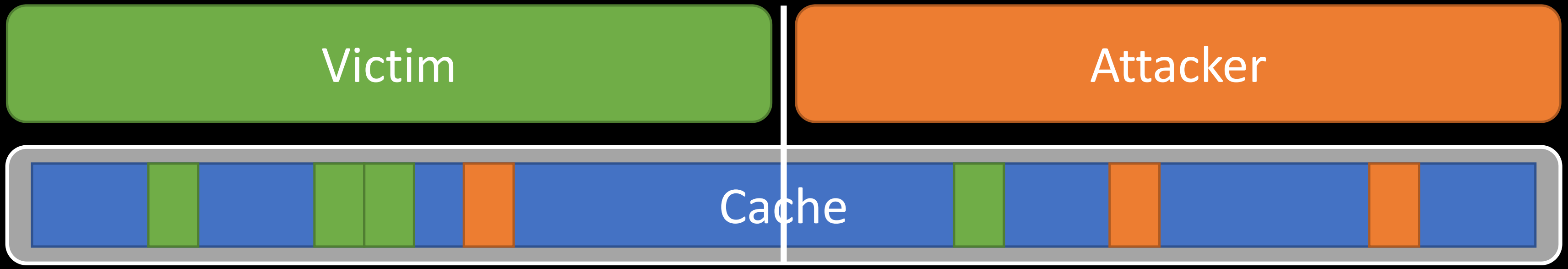


Cache Partitioning / Coloring

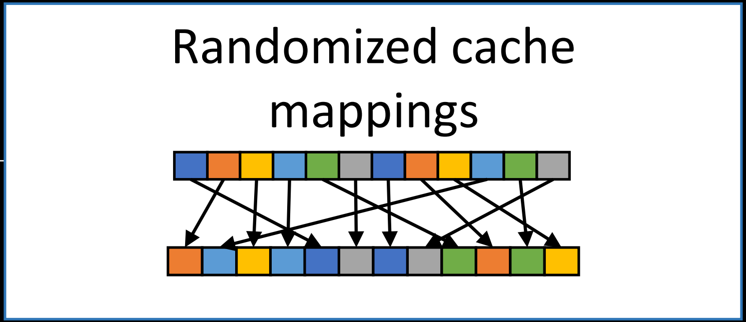
Cache partitioning / coloring



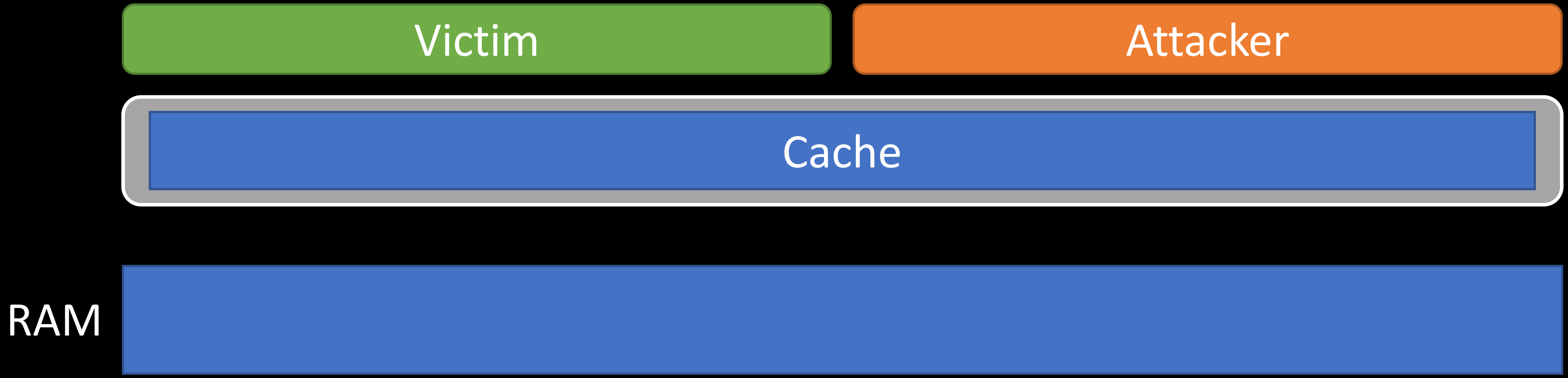
- Reduces the amount of cache available to individual software
- E.g., [Domnister et al., TACO’12]



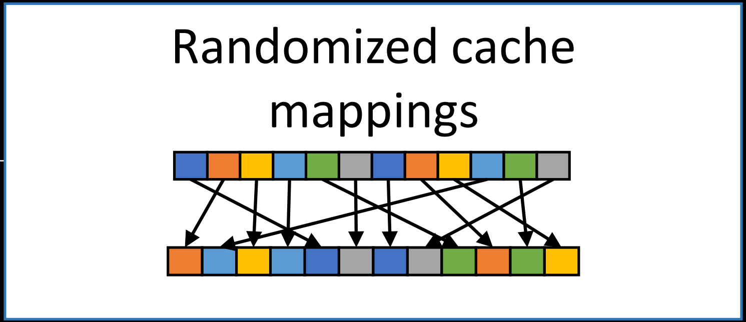
Randomized Cache Mappings



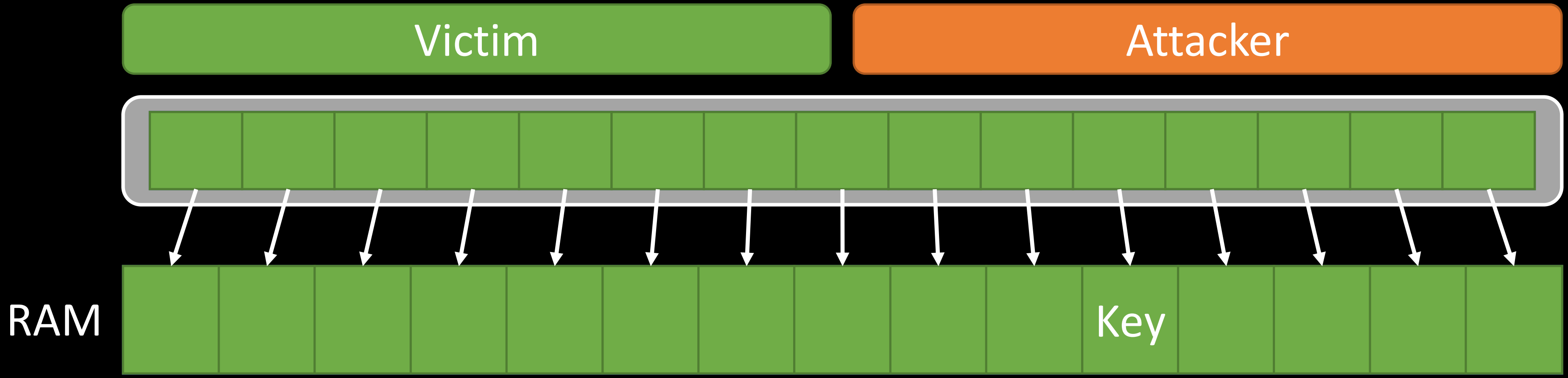
- Adversary cannot link cache observation with memory locations
- Frequency analysis or predictable access patterns can reveal randomization secret
- E.g., [Wang et al., ISCA'07]



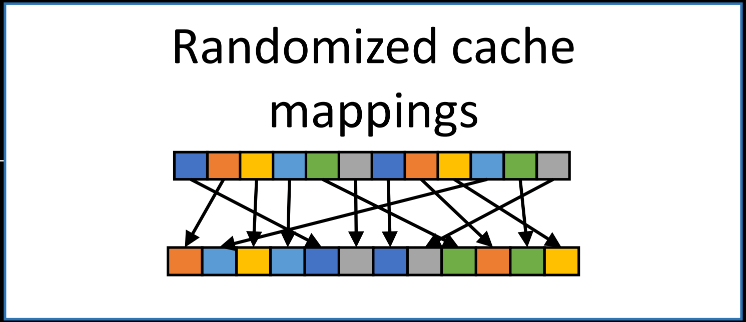
Randomized Cache Mappings



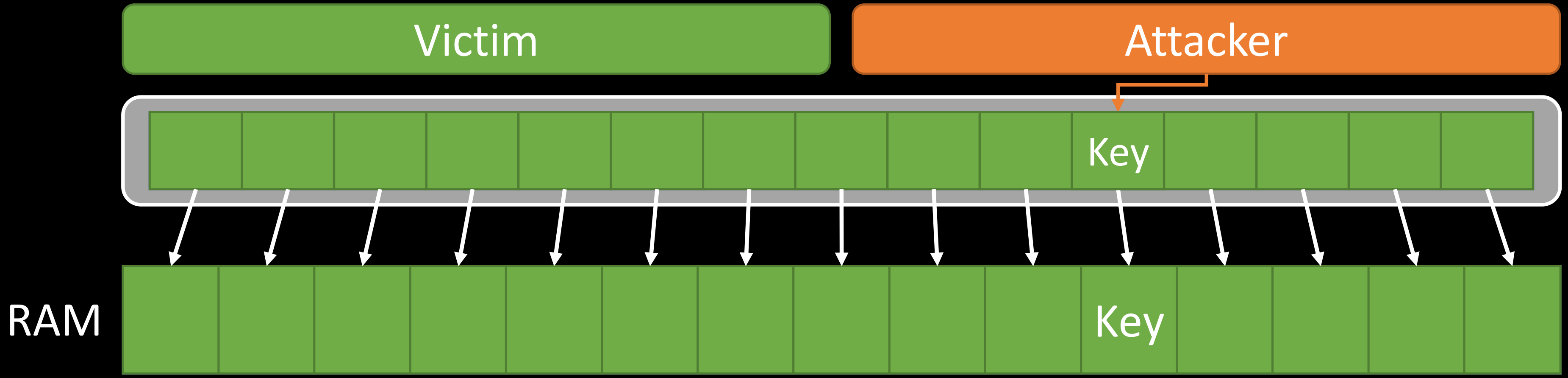
- Adversary cannot link cache observation with memory locations
- Frequency analysis or predictable access patterns can reveal randomization secret
- E.g., [Wang et al., ISCA'07]



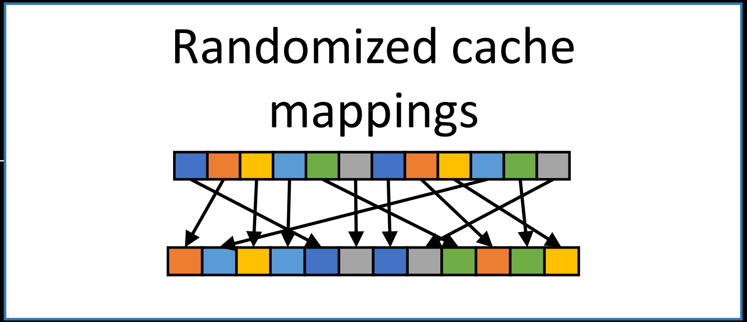
Randomized Cache Mappings



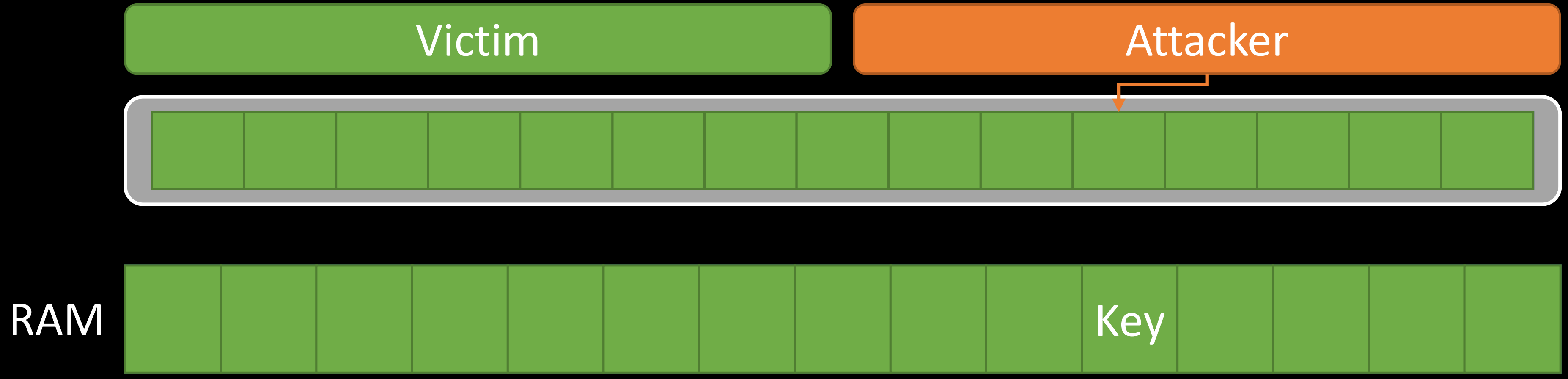
- Adversary cannot link cache observation with memory locations
- Frequency analysis or predictable access patterns can reveal randomization secret
- E.g., [Wang et al., ISCA'07]



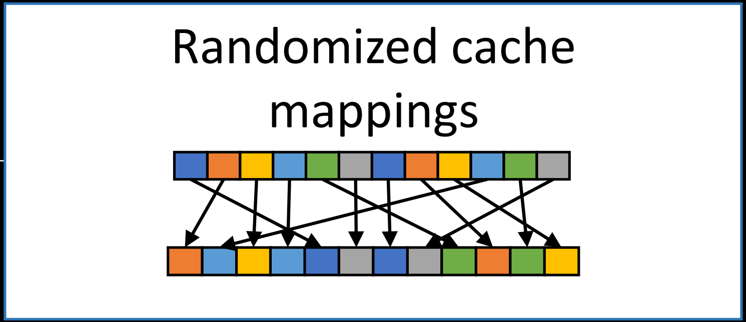
Randomized Cache Mappings



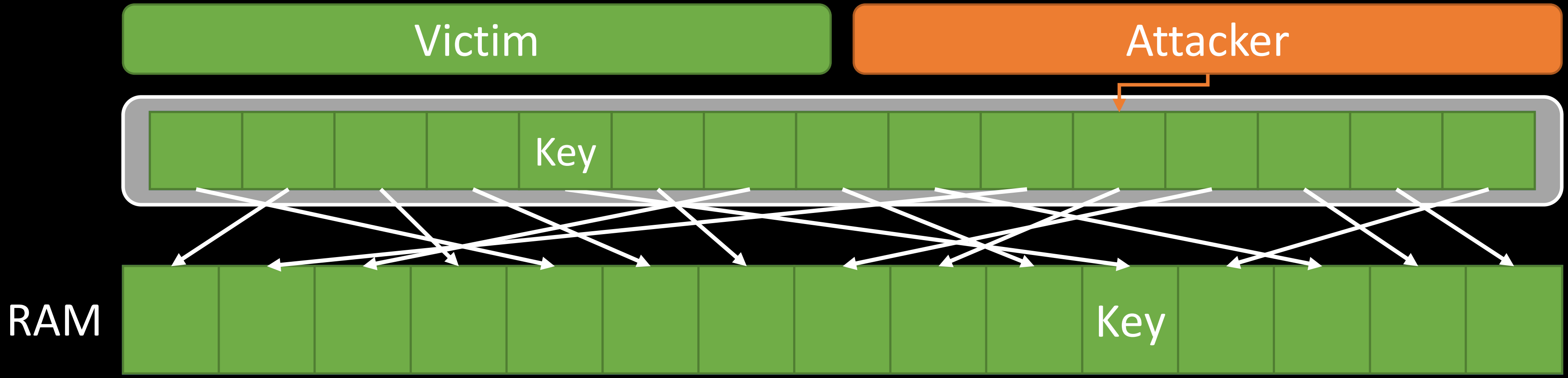
- Adversary cannot link cache observation with memory locations
- Frequency analysis or predictable access patterns can reveal randomization secret
- E.g., [Wang et al., ISCA'07]



Randomized Cache Mappings



- Adversary cannot link cache observation with memory locations
- Frequency analysis or predictable access patterns can reveal randomization secret
- E.g., [Wang et al., ISCA’07]

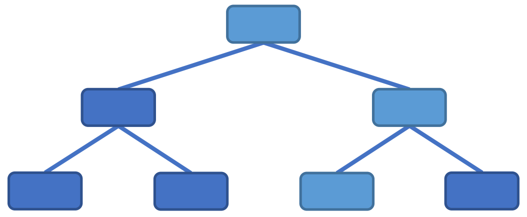


General Software-only Side-Channel Defenses

Side-channel resilient software design

Monitoring for attack effects

Oblivious execution / ORAM



General Software-only Side-Channel Defenses

Side-channel resilient
e design

Example

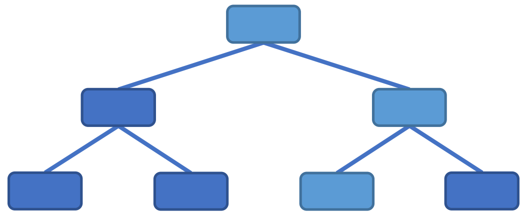
- Scatter and gather: data accesses effect *all* cache lines

Problems

- Not applicable to all applications
- Manual software hardening required

Monitoring for attack effects

Oblivious execution / ORAM



General Software-only Side-Channel Defenses

Side-channel resilient
e design

Example

- Scatter and gather: data accesses effect *all* cache lines

Problems

- Not applicable to all applications
- Manual software hardening required

Monitoring for attack effects

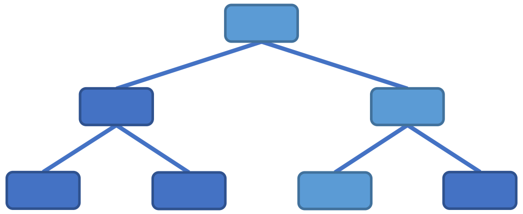
Example

- Use hardware performance counter to detect unusually high cache eviction rate

Problems

- Requires privileged entity (not available in SGX model)

Oblivious execution / ORAM



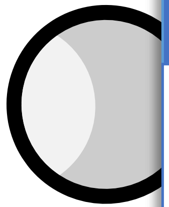
General Software-only Side-Channel Defenses

Side-channel resilient
e design

Monitoring for attack effects

Obli

RAM



Example

- Scatter and gather: data accesses effect *all* cache lines

Problems

- Not applicable to all applications
- Manual software hardening required

Oblivious RAM

- All memory accesses (code and/or data) indistinguishable

Problems

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

Example

- Use hardware performance counter to detect unusually high cache eviction rate

Problems

- Requires privileged entity (not available in SGX model)

Our Recent Work:
DR.SGX: Automated and Adjustable Side-Channel Protection for SGX using Data Location Randomization

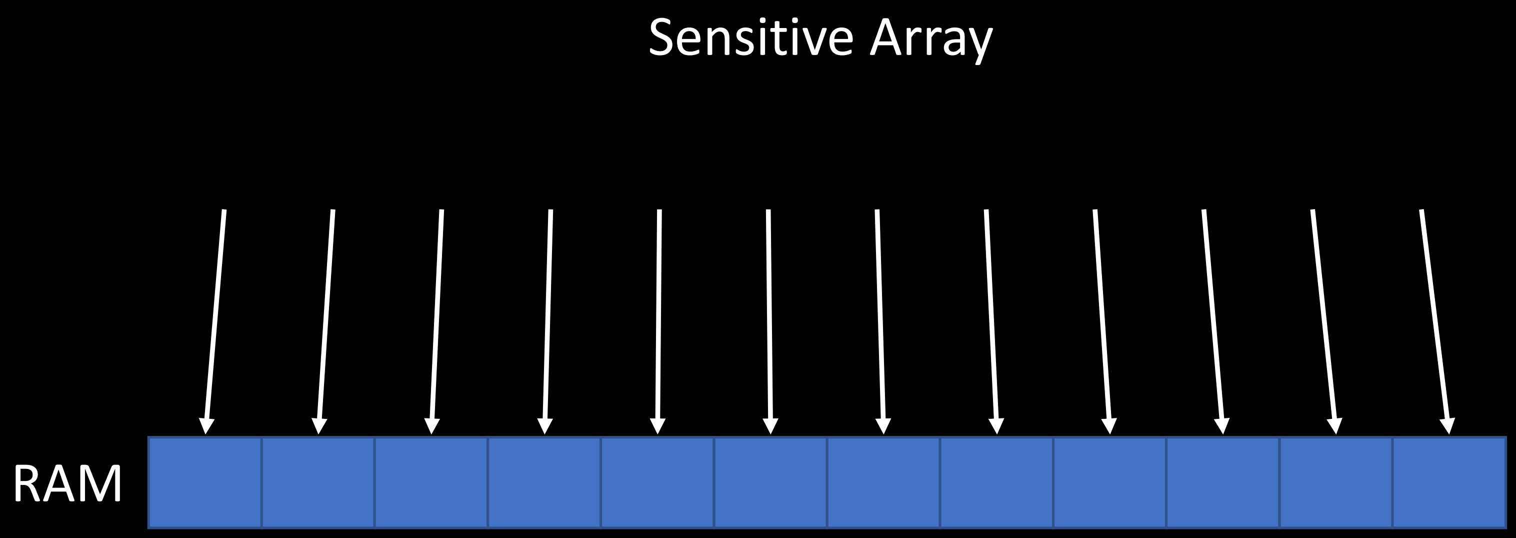
[Brasser et al., ACSAC 2019]

DR.SGX: Objective and Approach

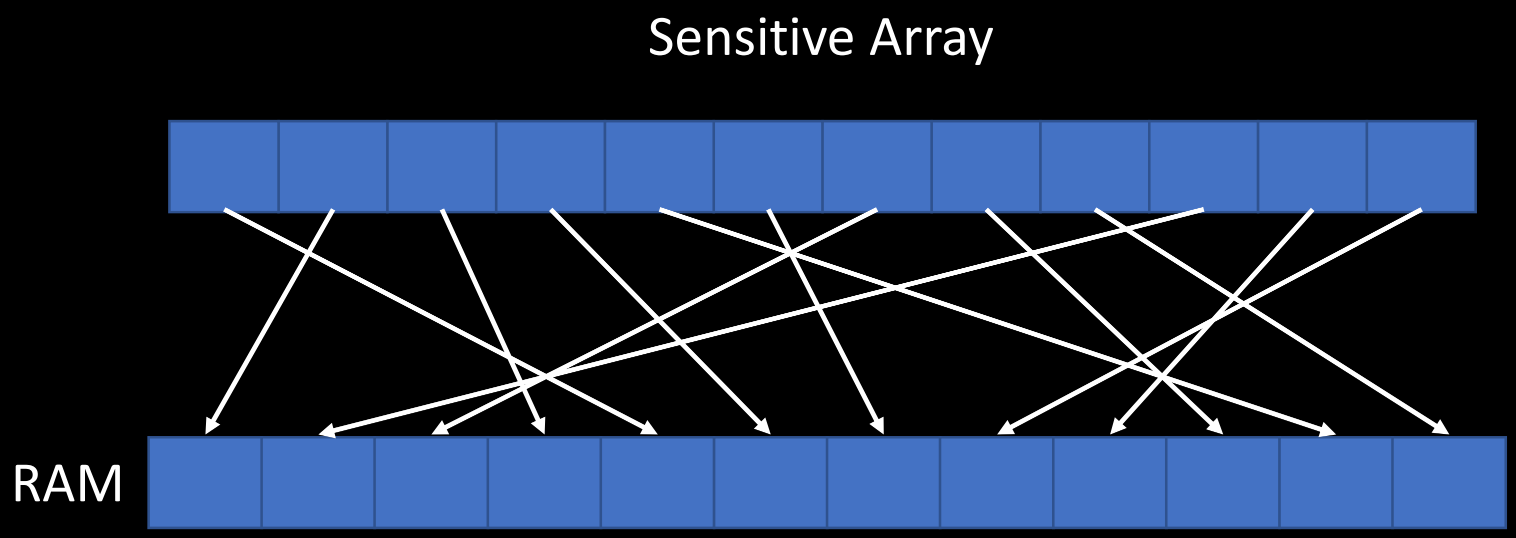
- **Objective:** Similarly to ORAM, make memory accesses indistinguishable
  - but at a cheaper cost
  - without relying on meta-data that needs protection

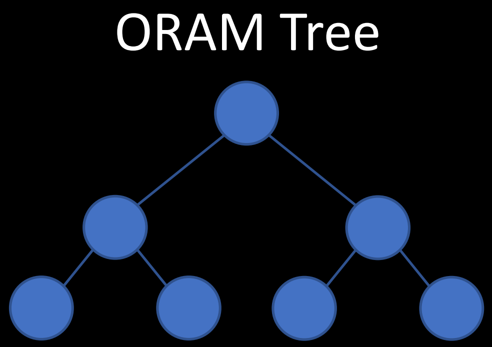
- **Approach:** Runtime fine-grained data location randomization
  - format-preserving encryption to determine location of randomized data
    - only small constant-size metadata needed
  - compiler-based approach (no annotations needed)
  - gradual randomization, interleaved with enclave execution
  - configurable re-randomization rate

Randomizing Memory: ORAM vs. DR.SGX

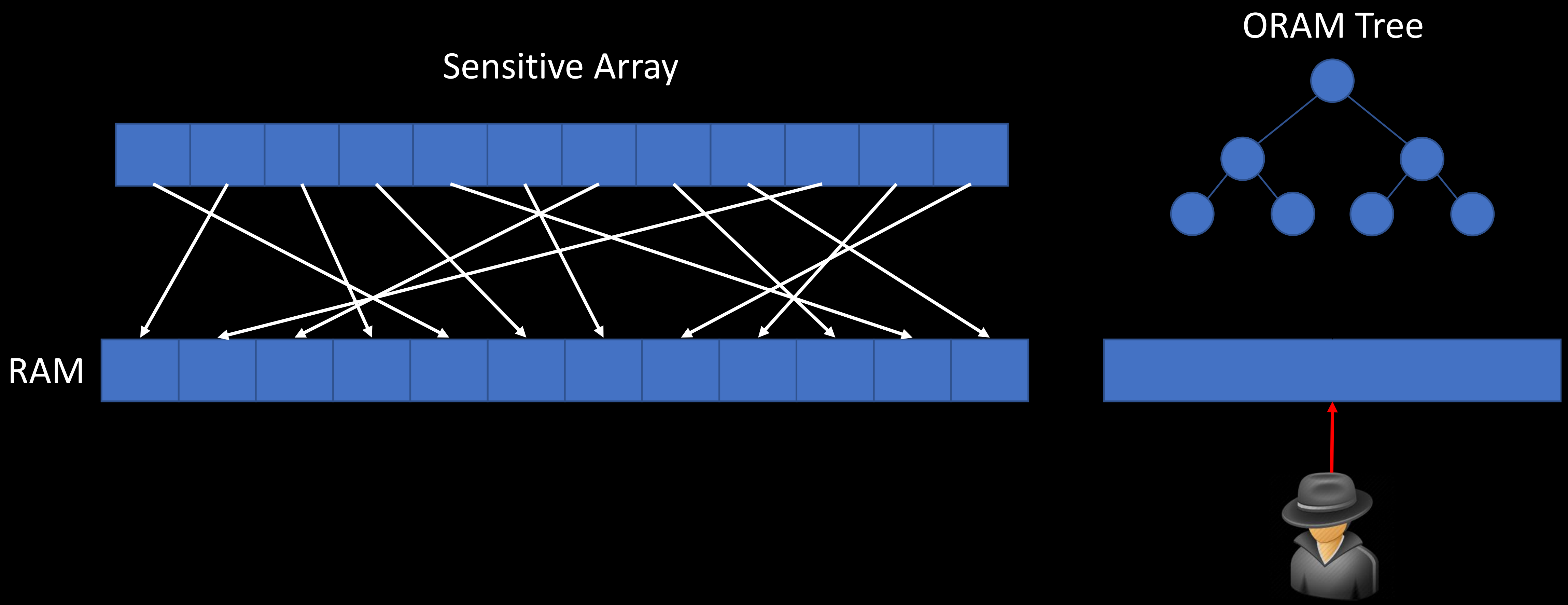


Randomizing Memory: ORAM vs. DR.SGX

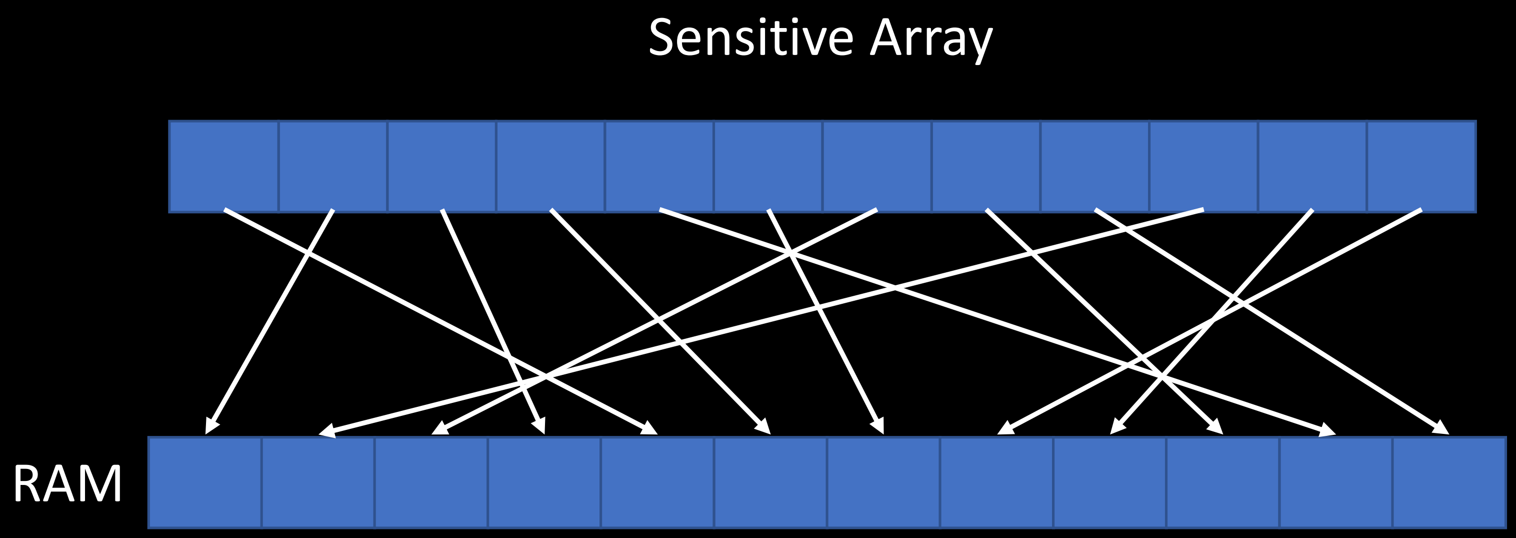


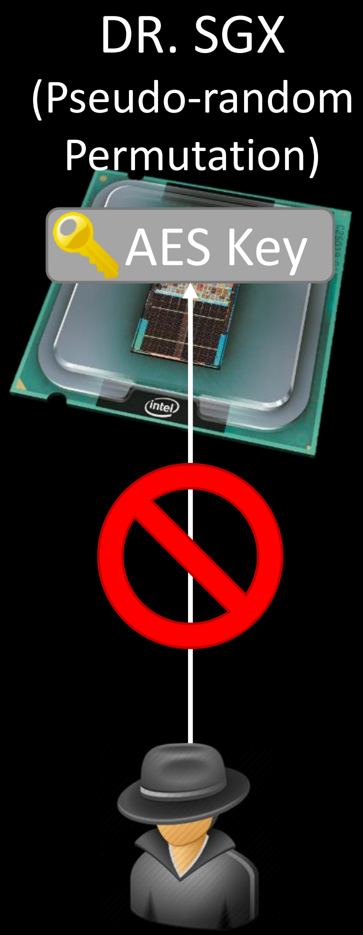


Randomizing Memory: ORAM vs. DR.SGX



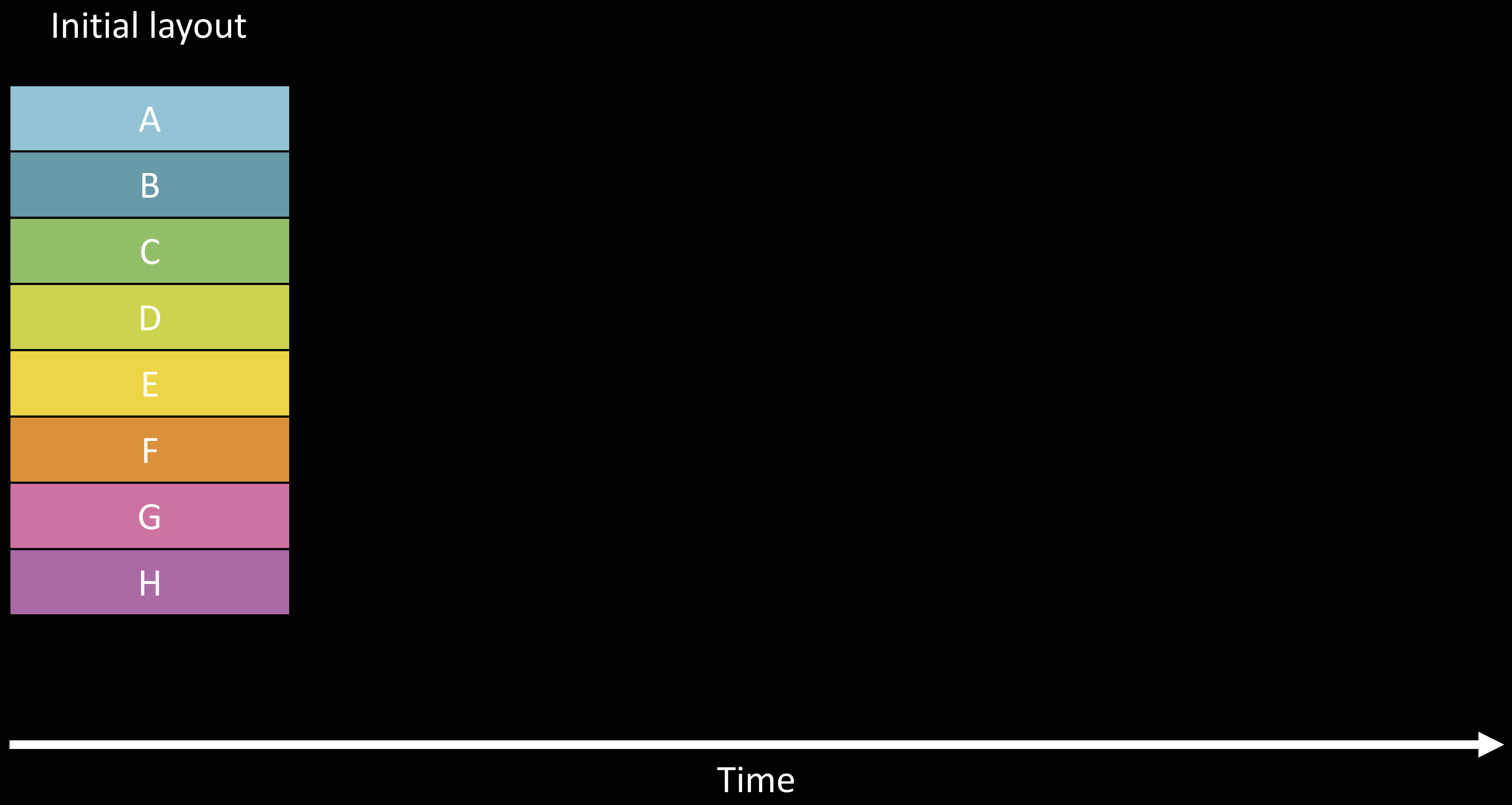
Randomizing Memory: ORAM vs. DR.SGX





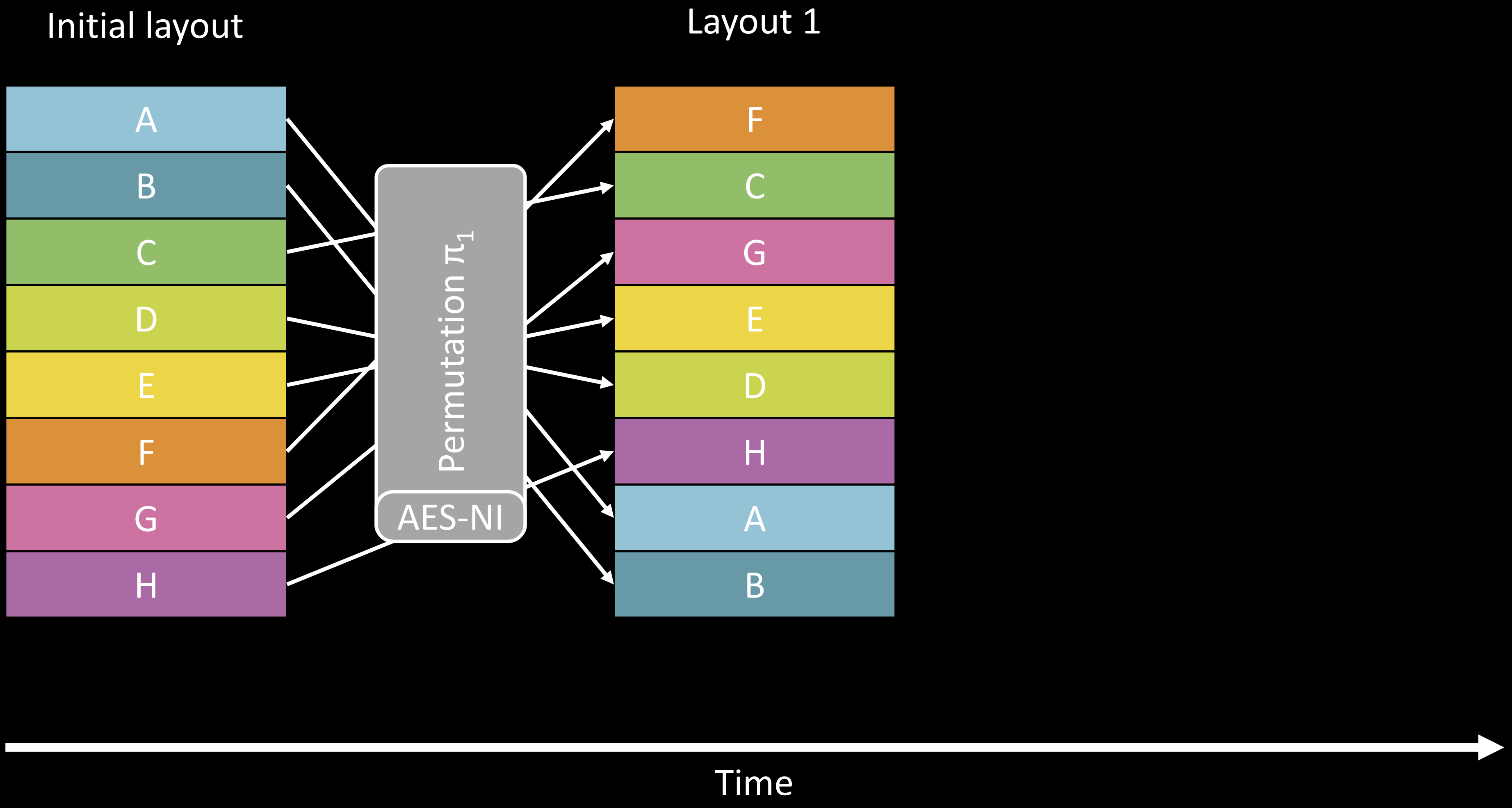
DR.SGX Re-randomization

FFX Format-Preserving Encryption scheme with AES as a block cipher



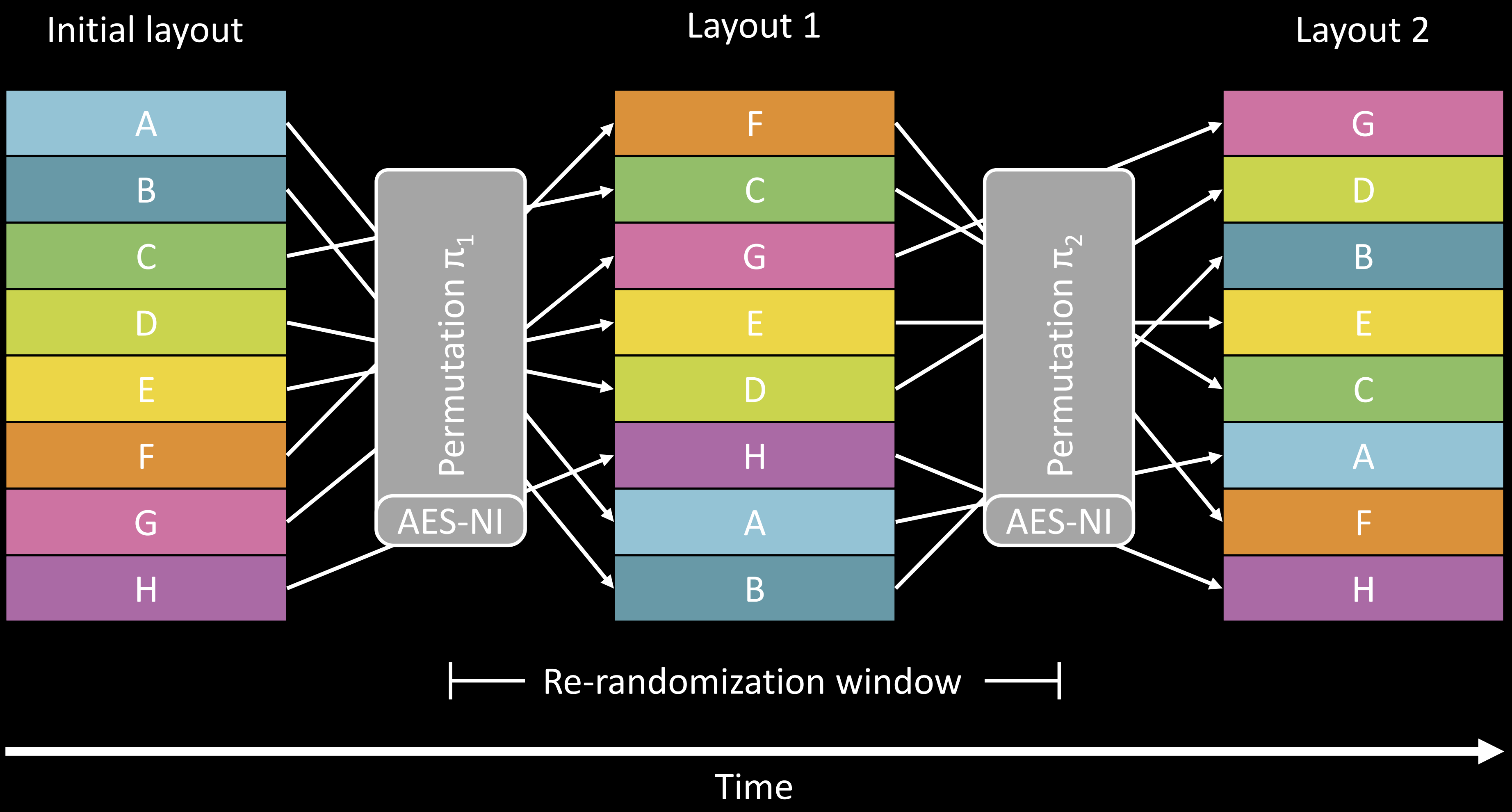
DR.SGX Re-randomization

FFX Format-Preserving Encryption scheme with AES as a block cipher

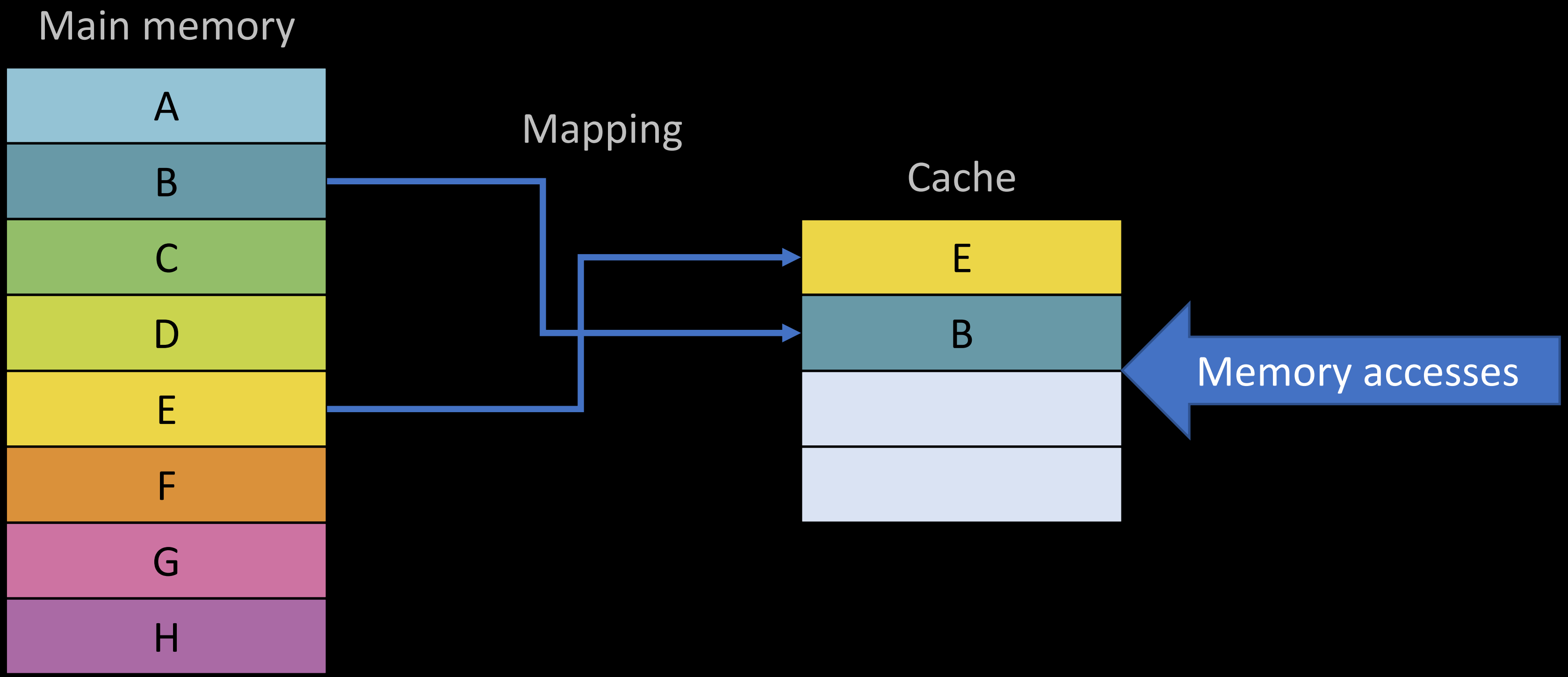


DR.SGX Re-randomization

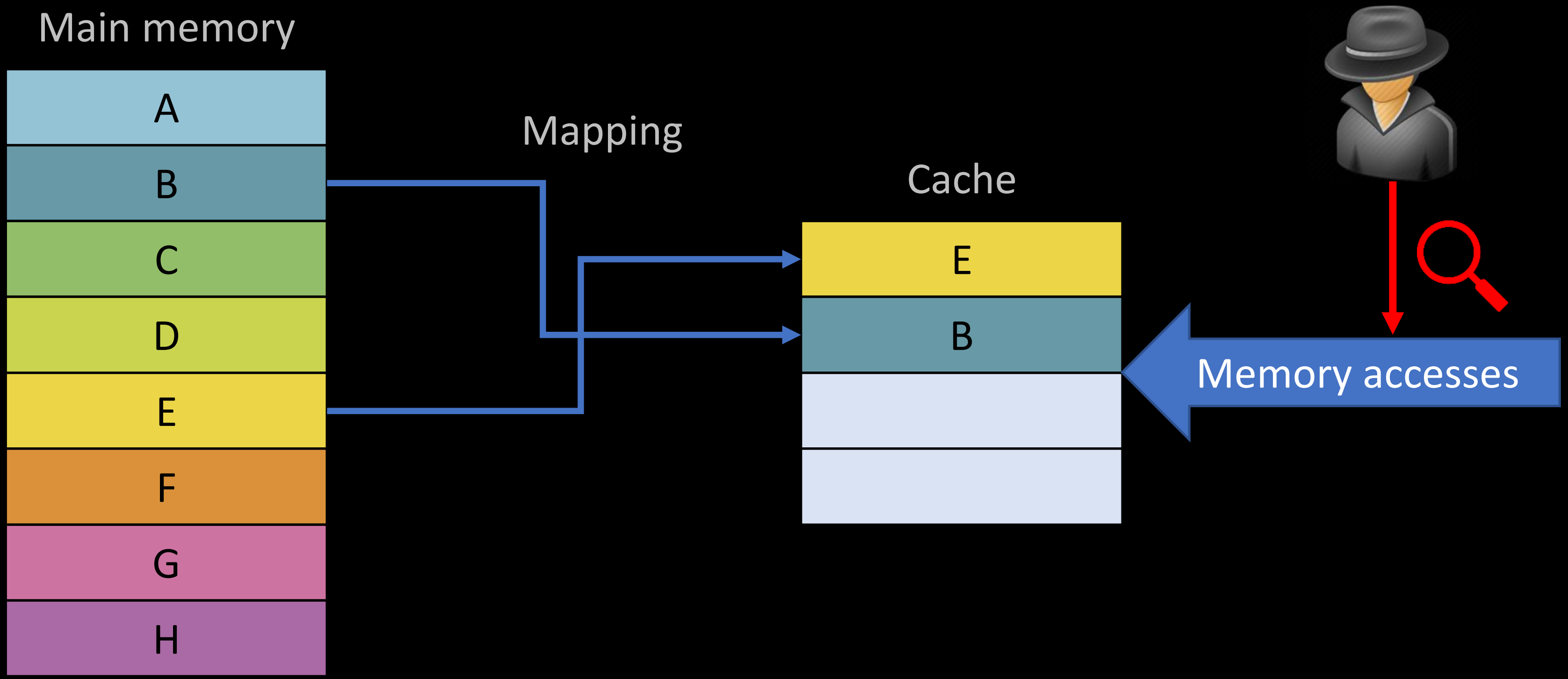
FFX Format-Preserving Encryption scheme with AES as a block cipher



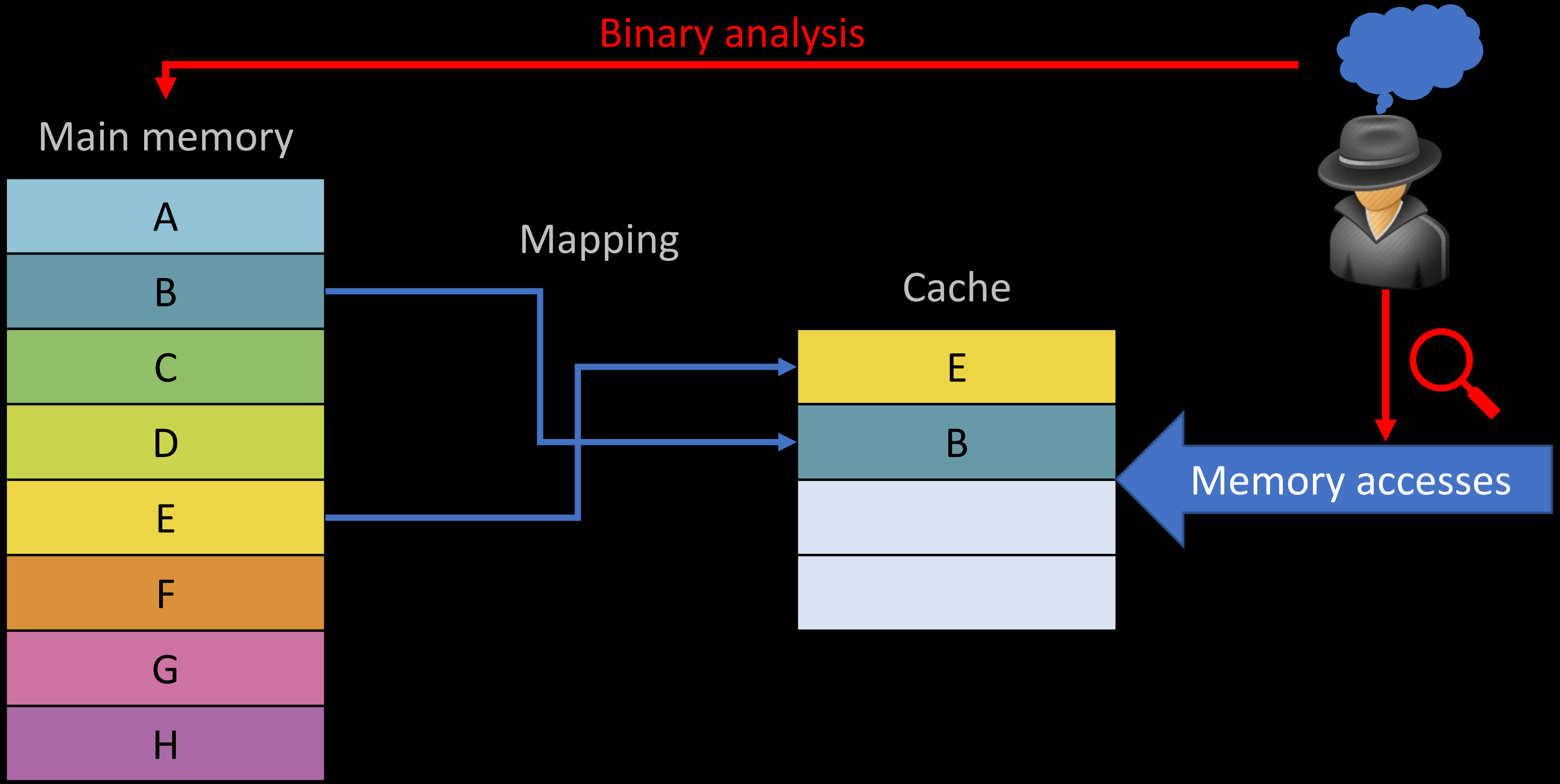
Data Randomization



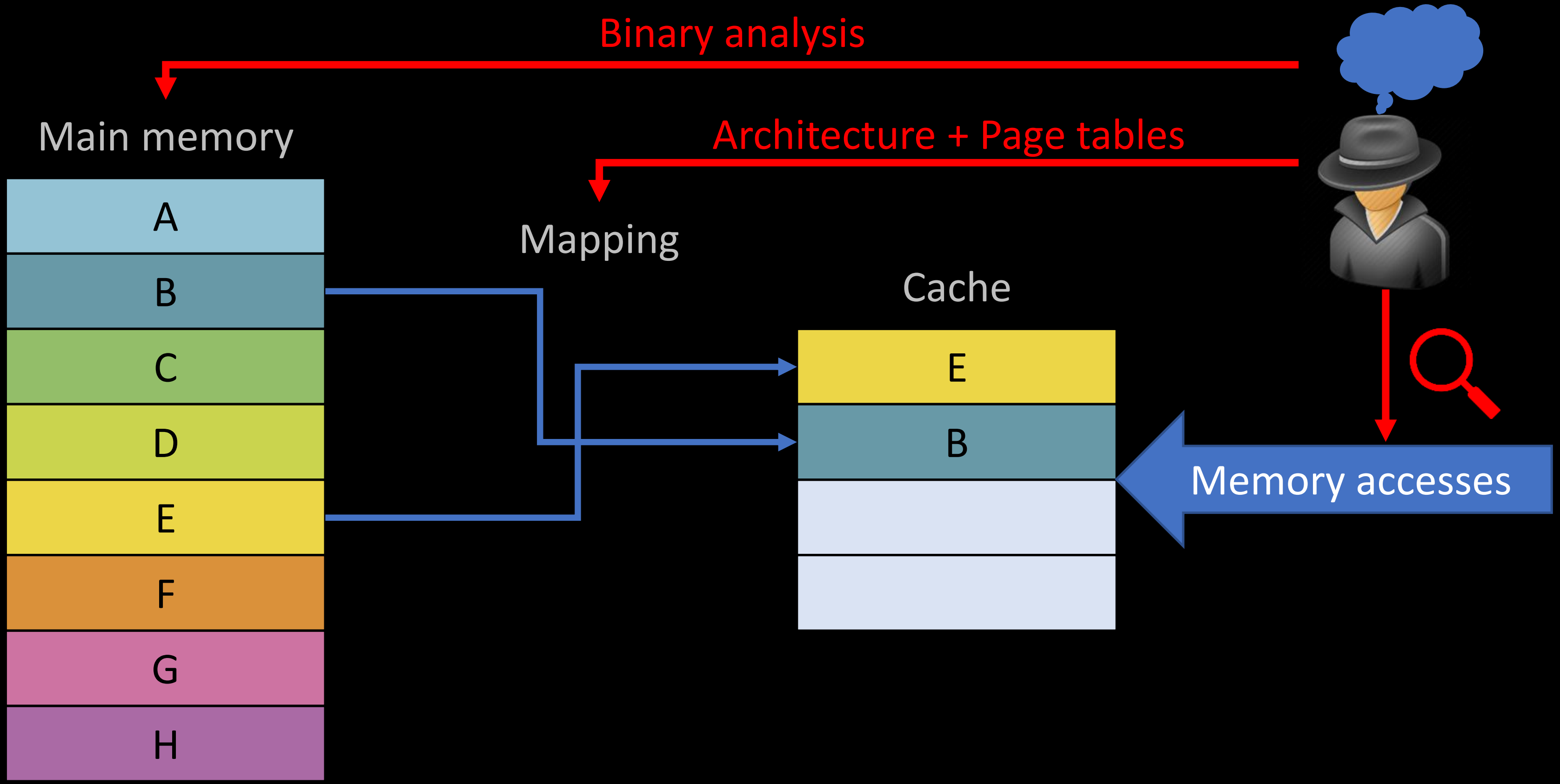
Data Randomization



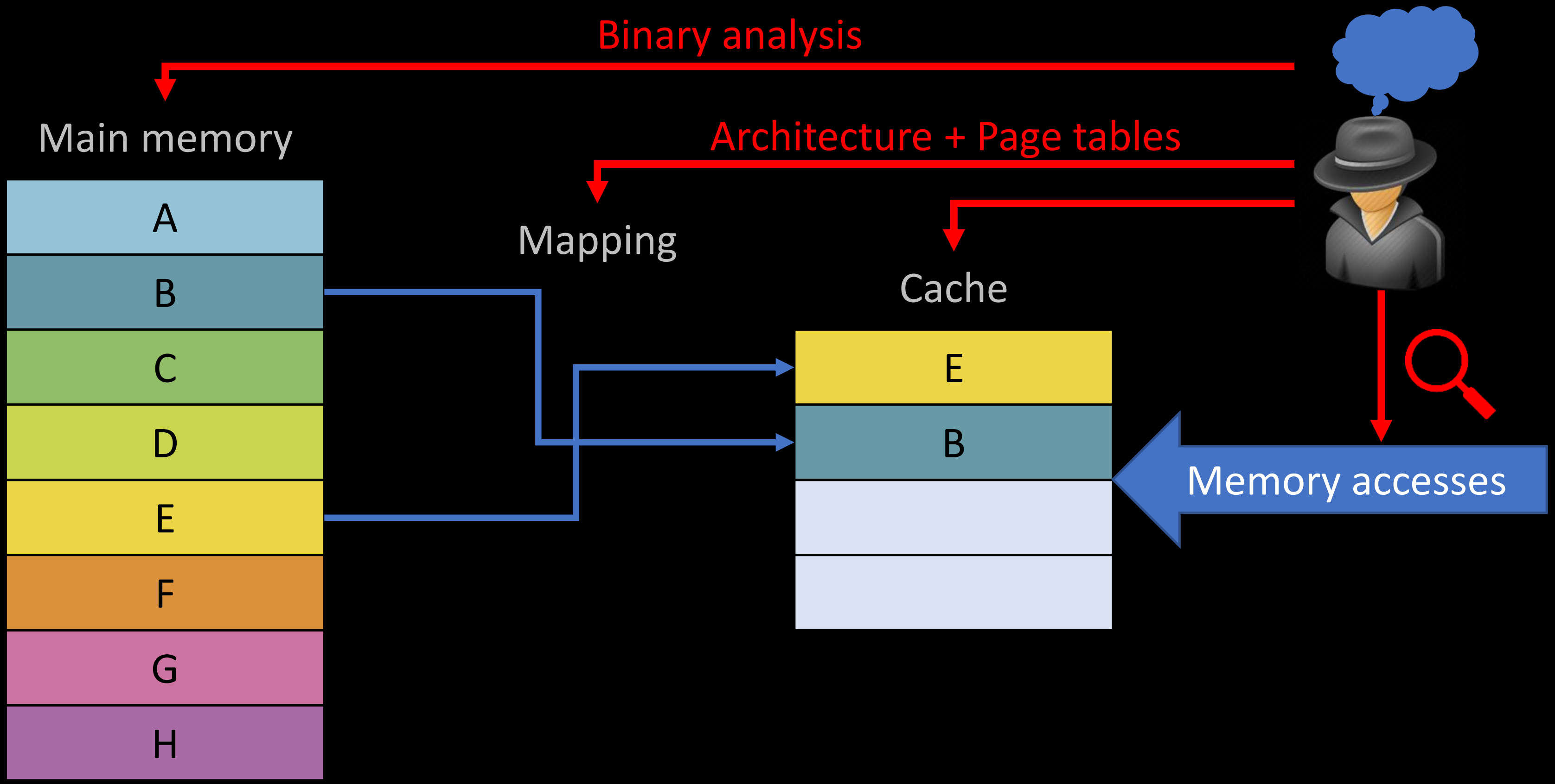
Data Randomization



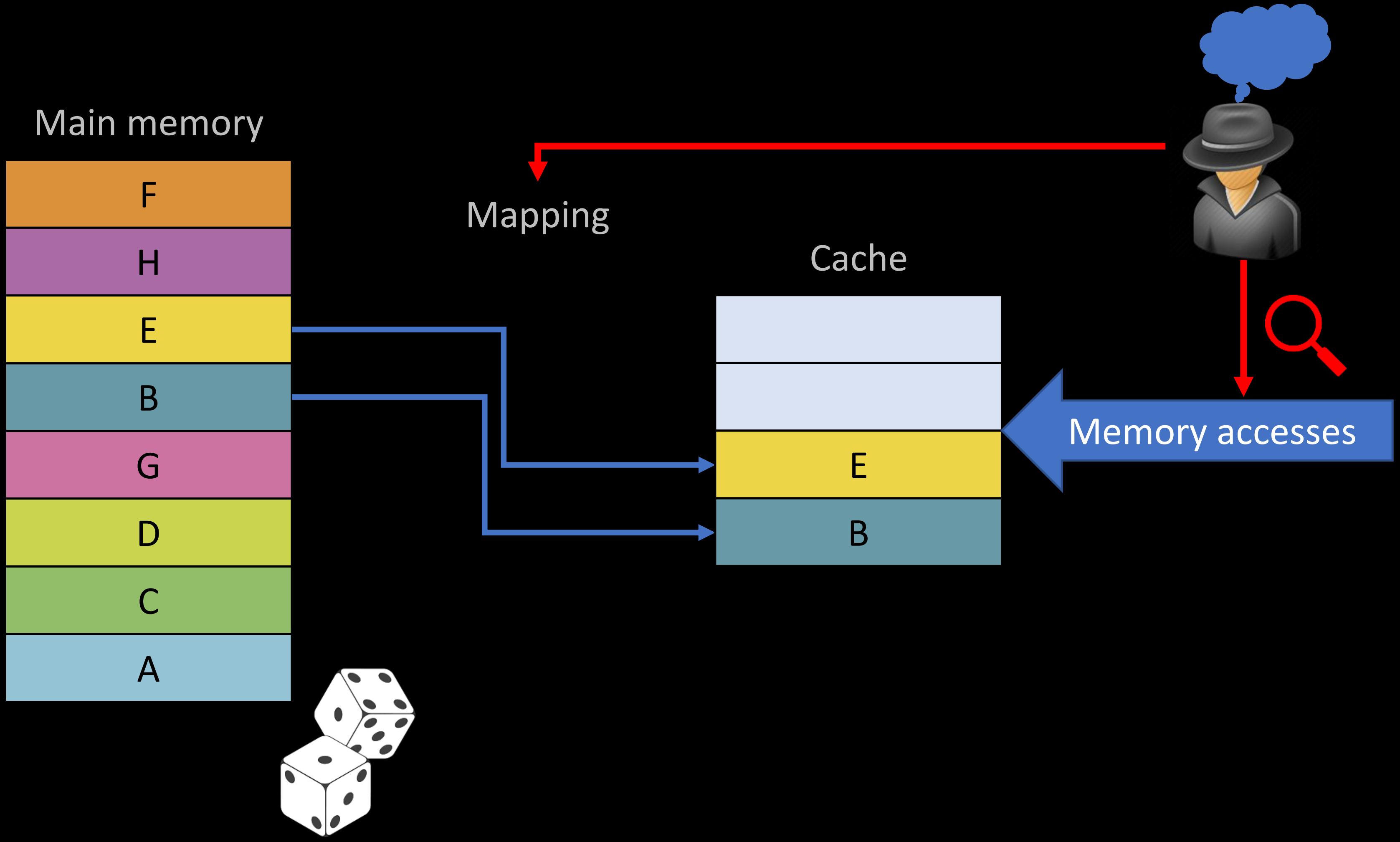
Data Randomization



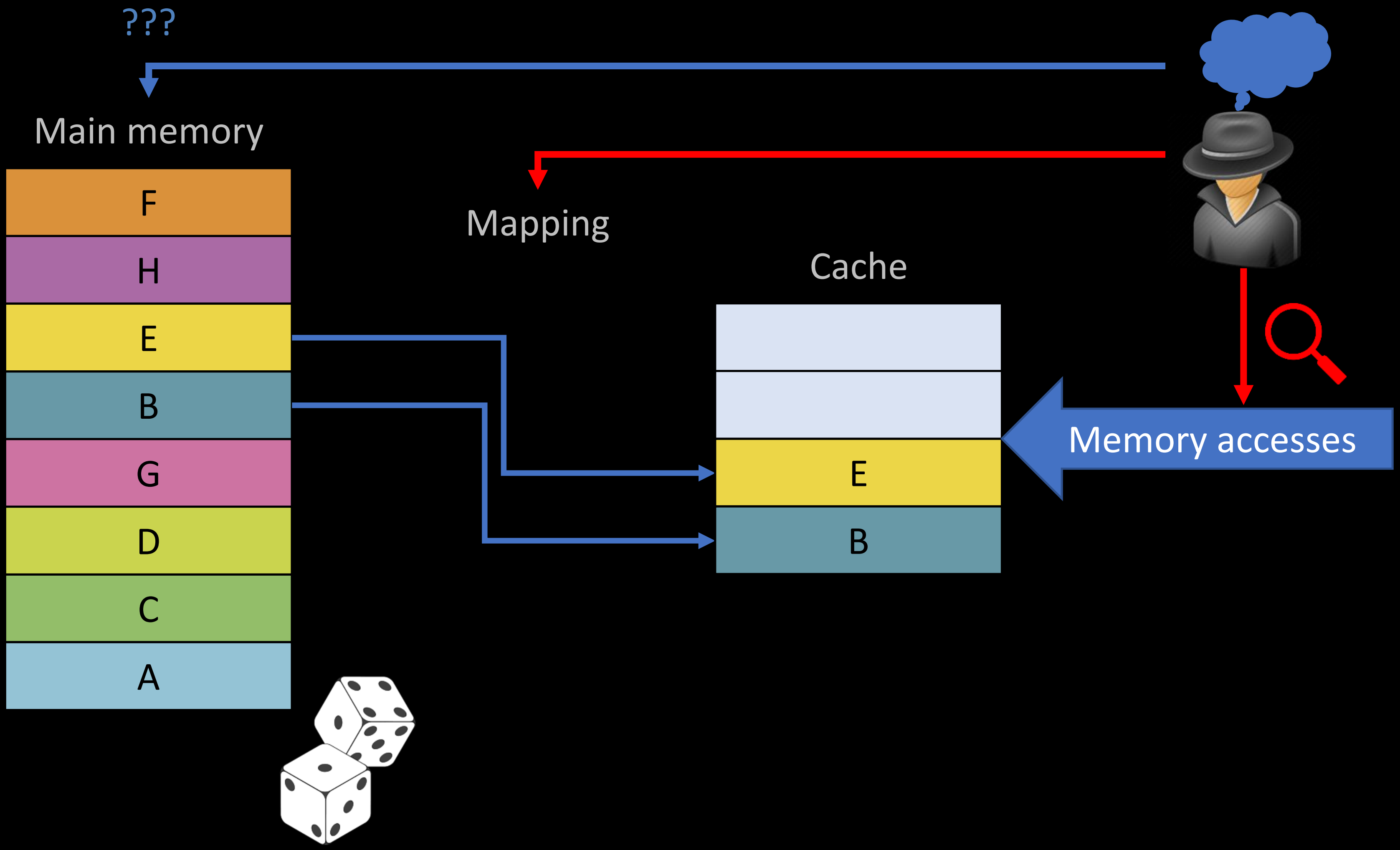
Data Randomization



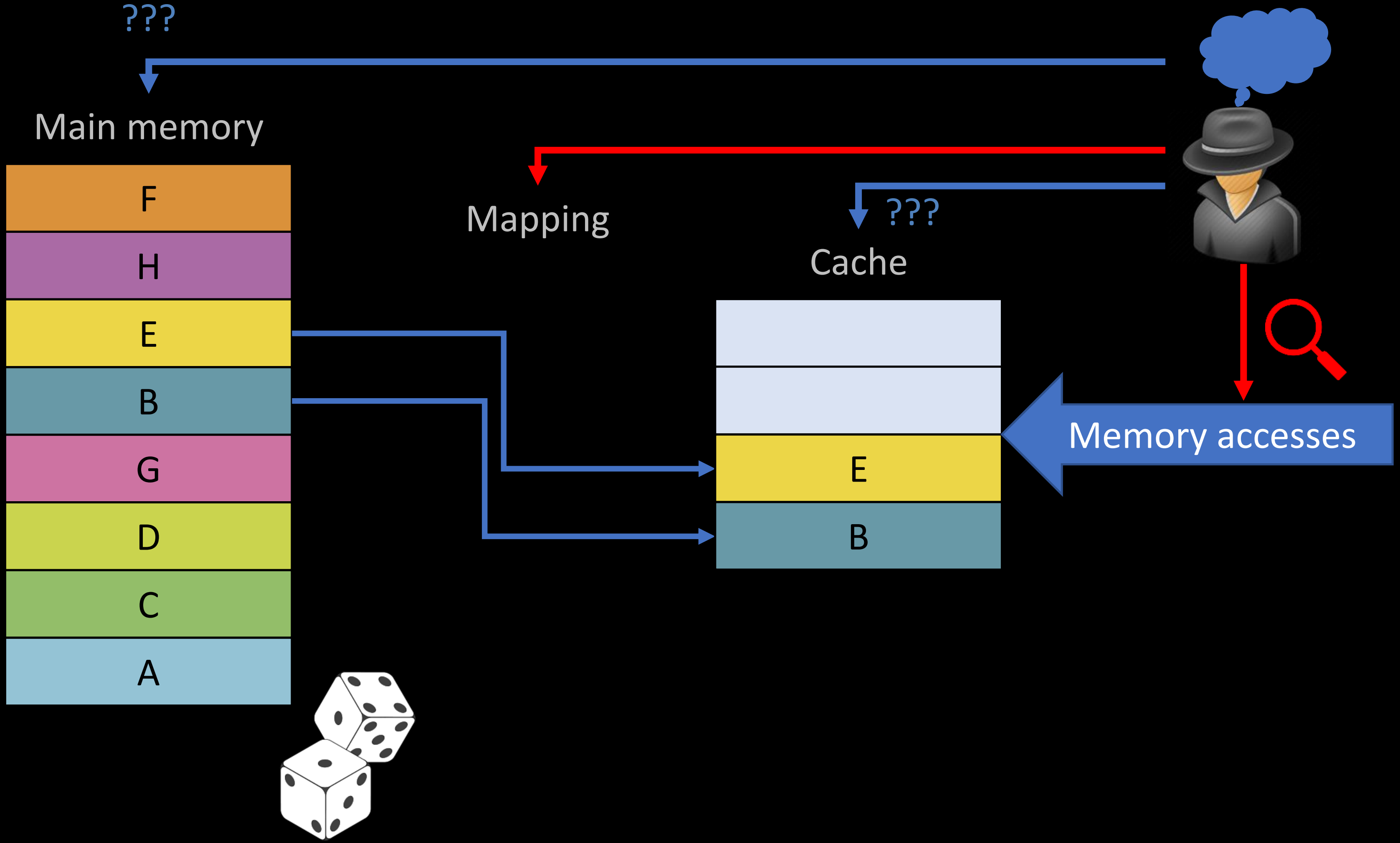
Data Randomization



Data Randomization

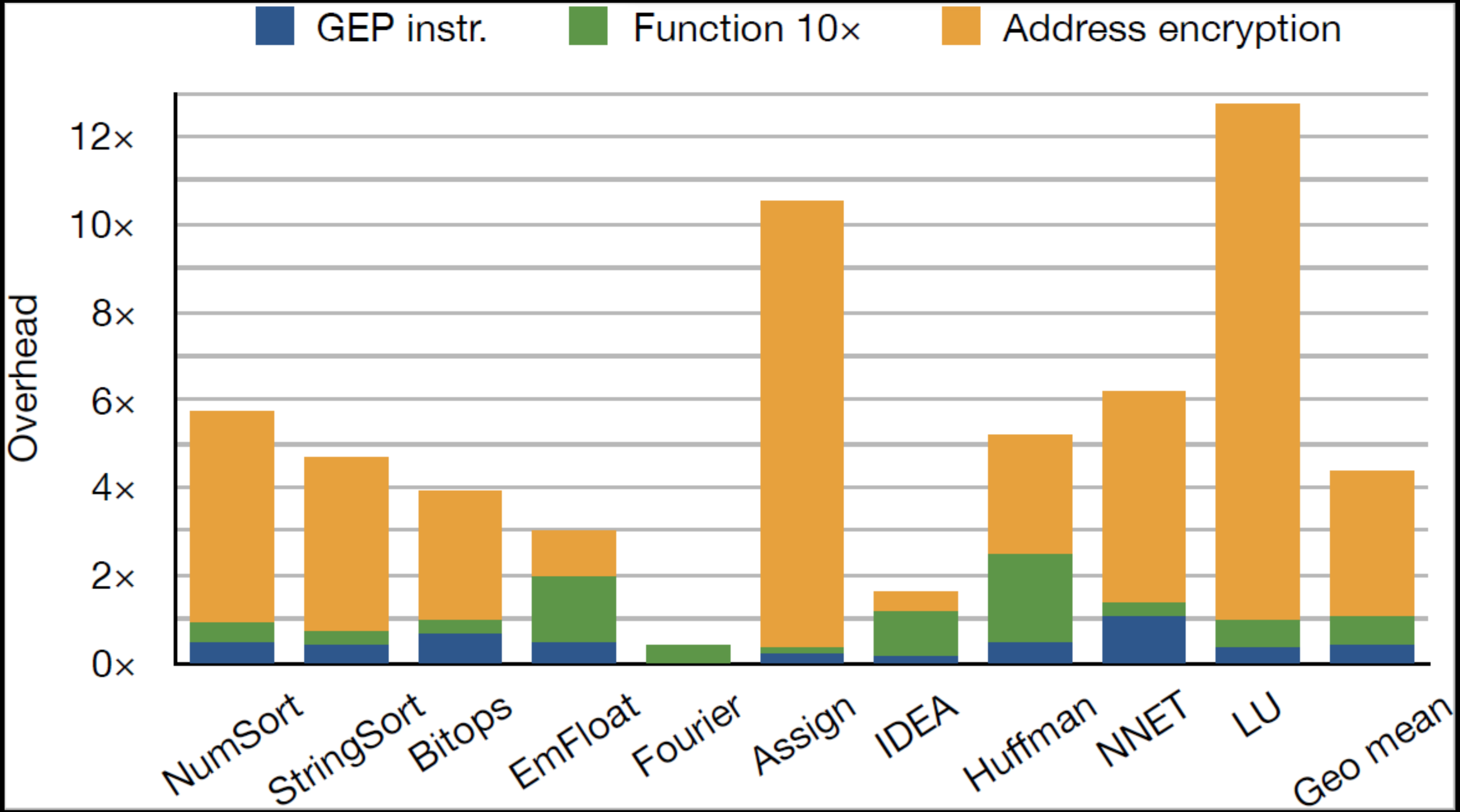


Data Randomization



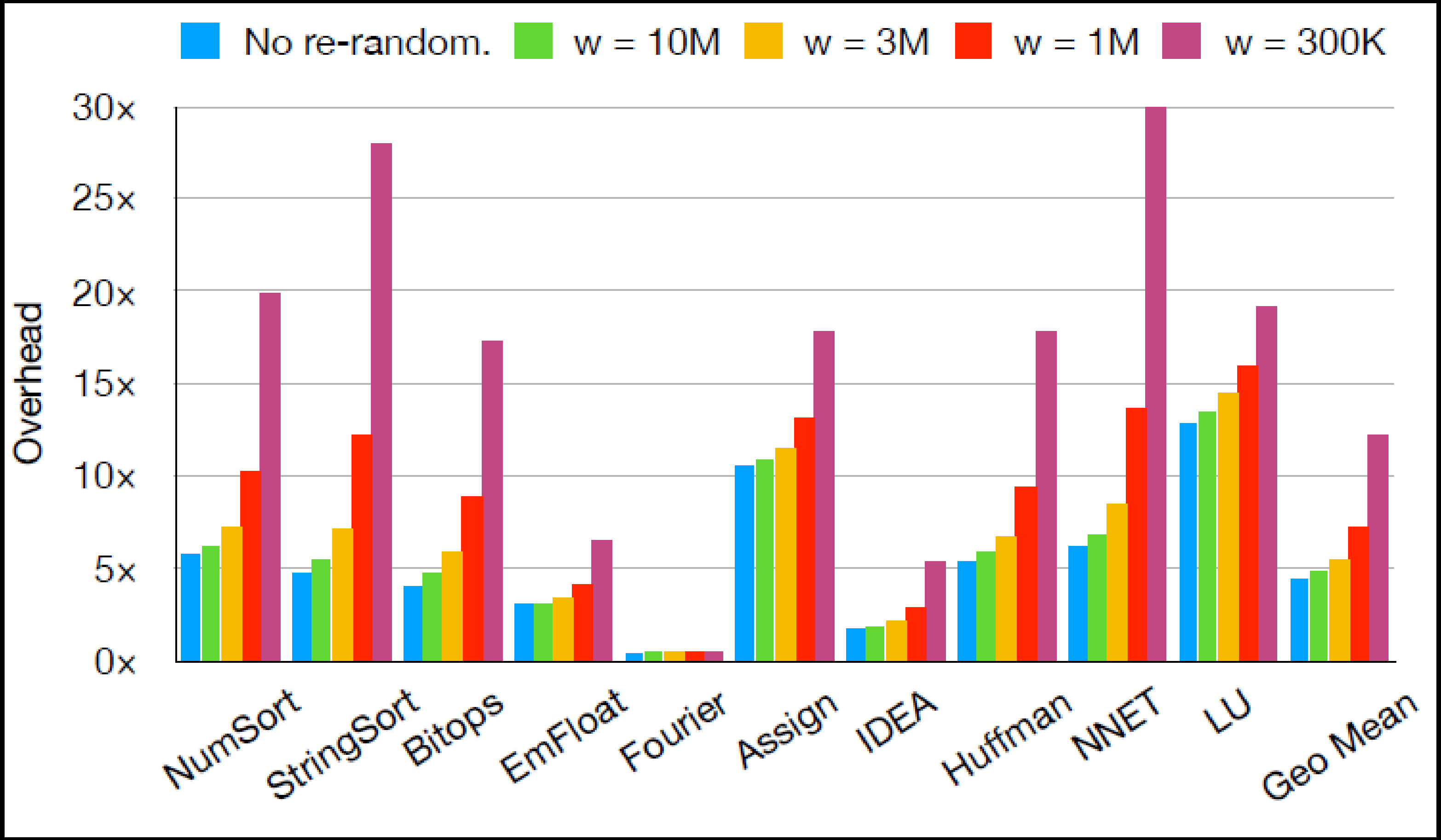
Performance Evaluation using Nbench

- Without runtime re-randomization (geometric mean about 4x)



Performance Evaluation using Nbench

- With different re-randomization windows (geometric mean up to 12x)



ORAM vs. Dr.SGX: Performance Comparison

- Obfuscuro [Ahmad et al., NDSS 2019]
  - Obfuscation engine on Intel SGX
  - Implements both, ORAM and oblivious execution
  - Performance overheads of 83x on average and up to 220x

- Dr. SGX
  - Performance overhead 4x – 12x
    - at least one order of magnitude lower than Obfuscuro
  - Allows developers to balance between increased side-channel protection and the performance cost based on adjustable security parameter

Conclusion

- Great concepts suffer from implementation problems
- Intel SGX is no exception
- Side-channel attacks are a major threat to Intel SGX
  - Were deemed as ‘too difficult’ and were left out of the attacker model
  - Research has shown it otherwise
  - Attacks still can be improved through more automation
- Countermeasures
  - Range from specific protections against particular problems to generic solutions
  - Generic solutions, however, come at significant (prohibitive?) cost
  - There is a need for more efficient generic solutions

E
?
R
S
D
F