Lecture Secure, Trusted and Trustworthy Computing

SGX Side-Channel Attacks

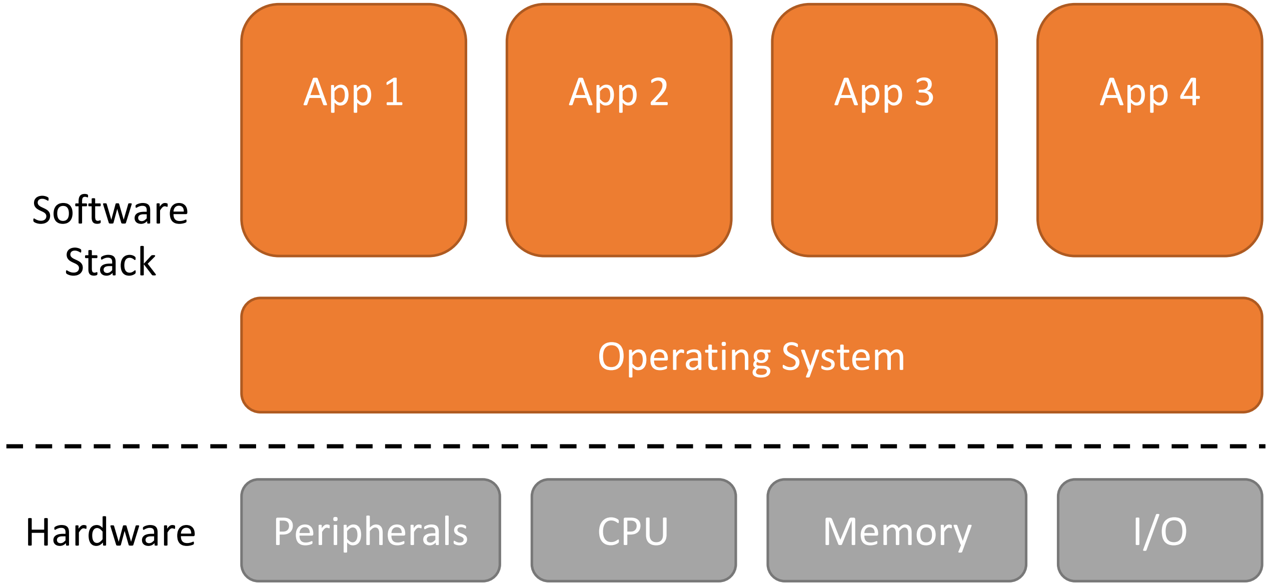
Prof. Dr.-Ing. Ahmad-Reza Sadeghi
System Security Lab
Technische Universität Darmstadt
Germany

Winter Term 2017/18

Intel Software Guard Extensions (SGX)

Assumptions:

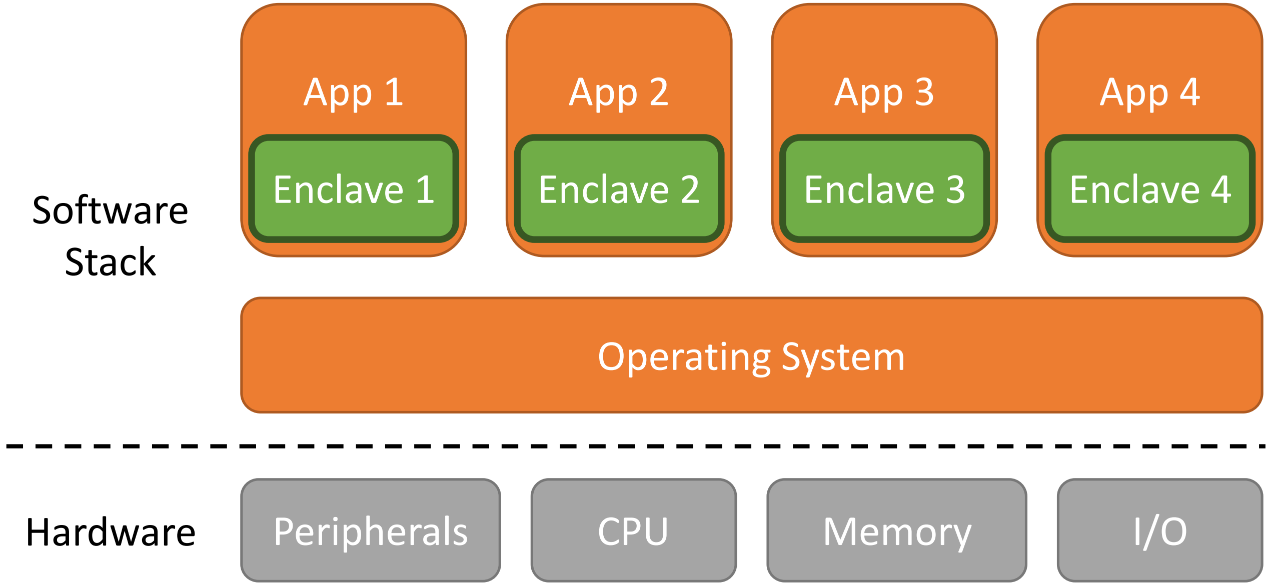
- All software components untrusted



Intel Software Guard Extensions (SGX)

Assumptions:

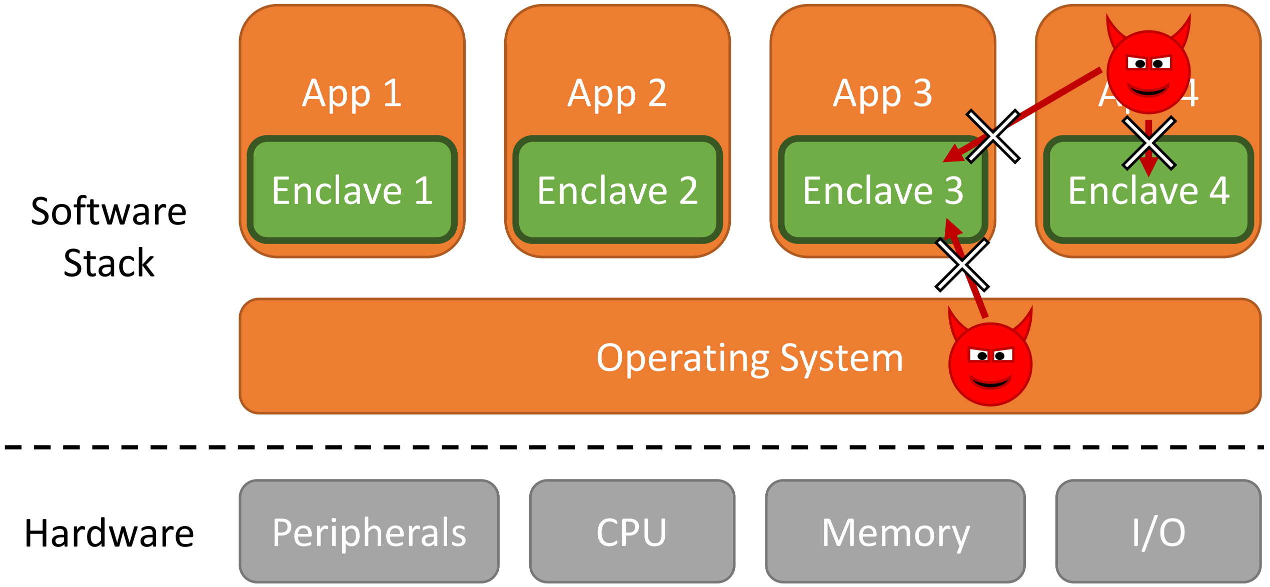
- All software components untrusted



Intel Software Guard Extensions (SGX)

Assumptions:

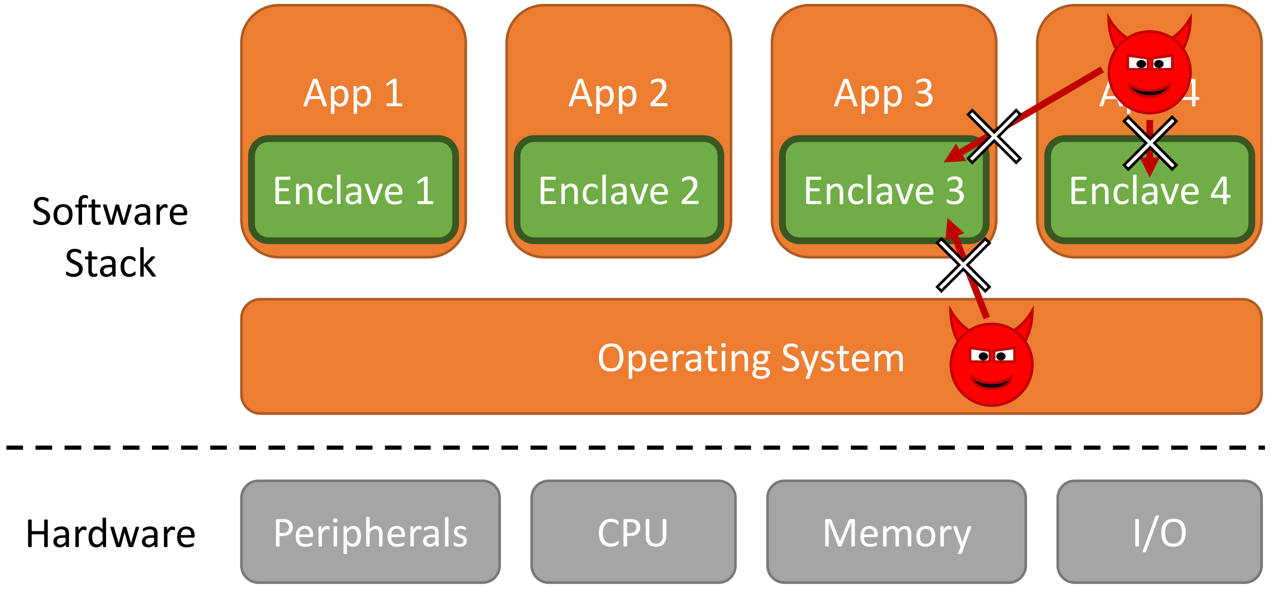
- All software components untrusted



Intel Software Guard Extensions (SGX)

Assumptions:

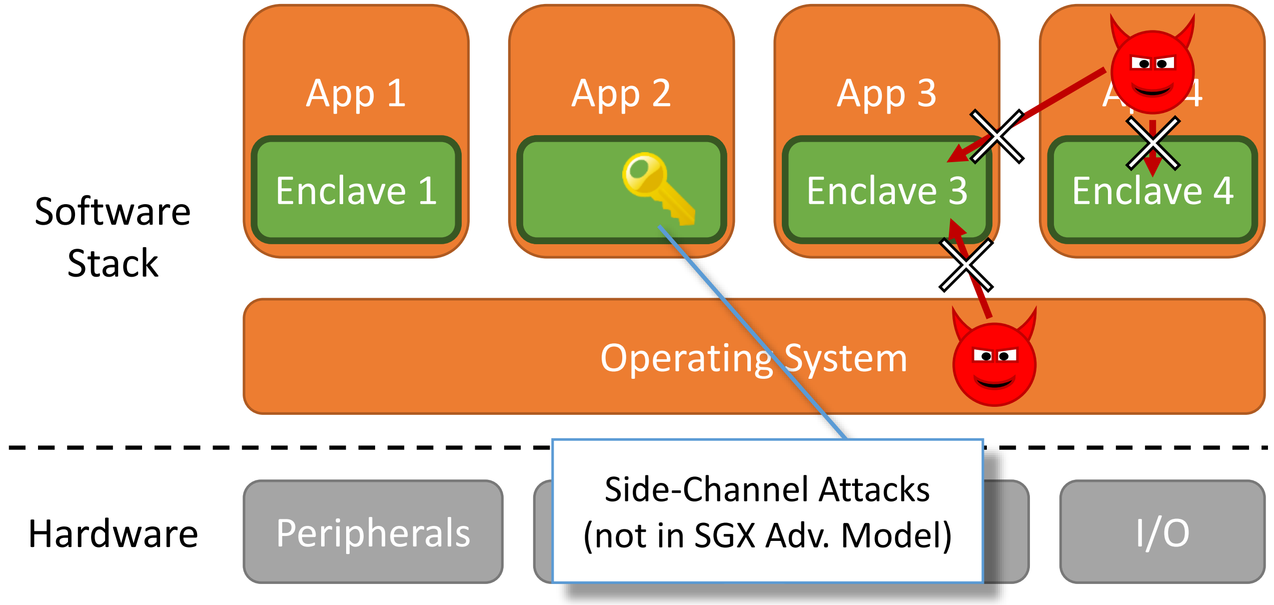
- All software components untrusted



Intel Software Guard Extensions (SGX)

Assumptions:

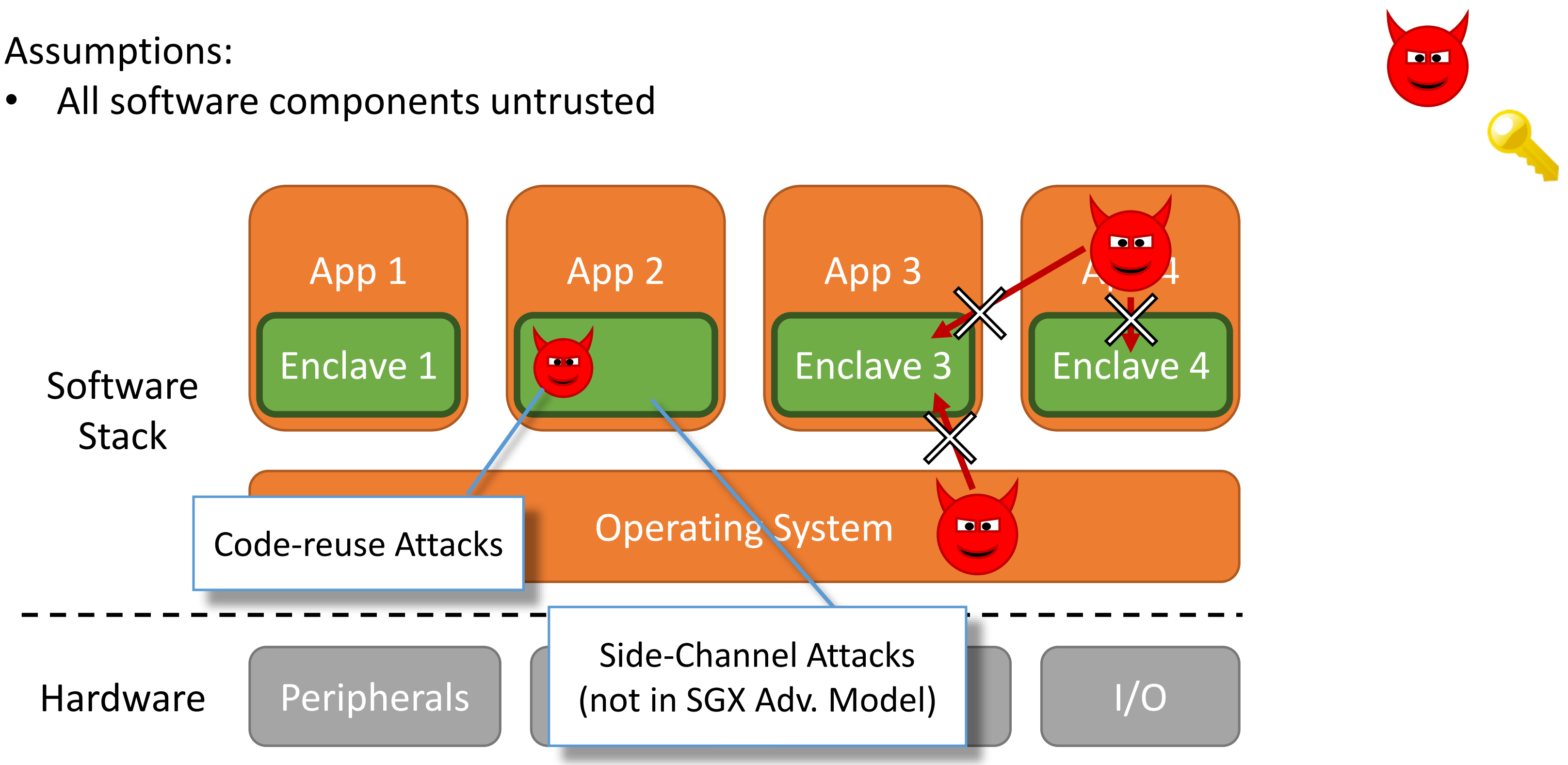
- All software components untrusted



Intel Software Guard Extensions (SGX)

Assumptions:

- All software components untrusted

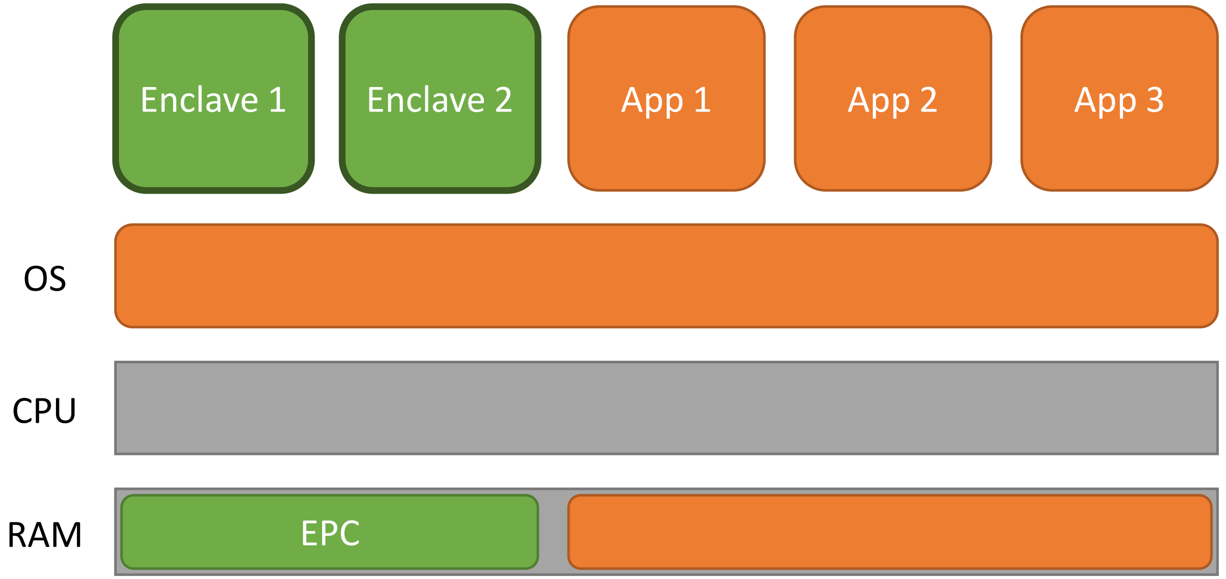


Leakage in Intel’s SGX

Page Fault Attacks on SGX

Granularity: page 4K, good for big data structures

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

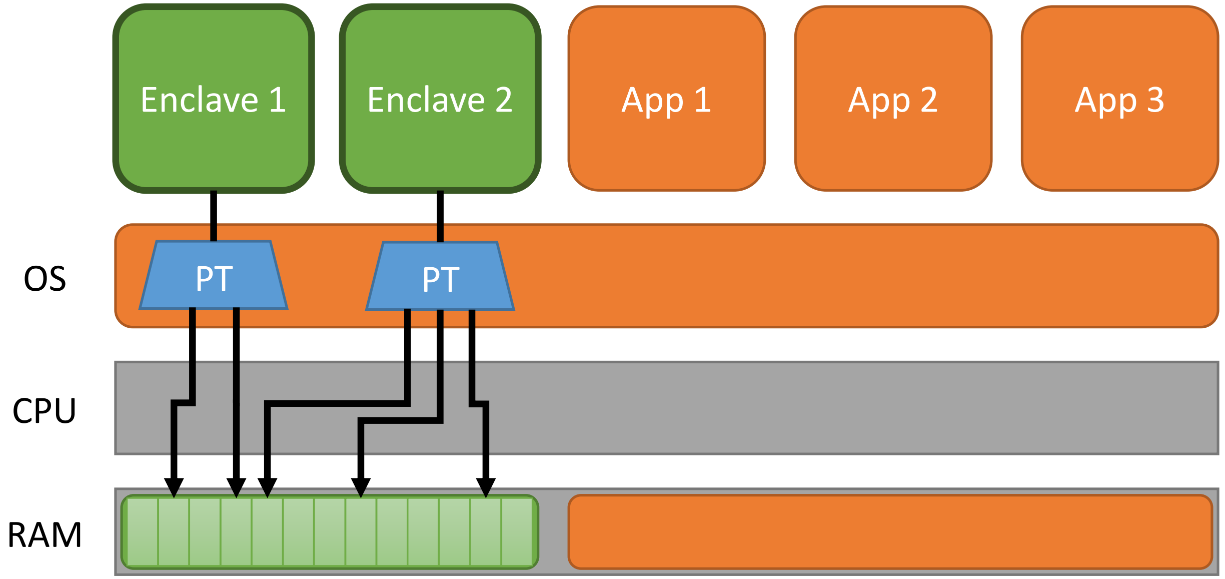


EPC: Enclave Page Cache   PT: Page Tables   PF: Page-Fault

Page Fault Attacks on SGX

Granularity: page 4K, good for big data structures

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

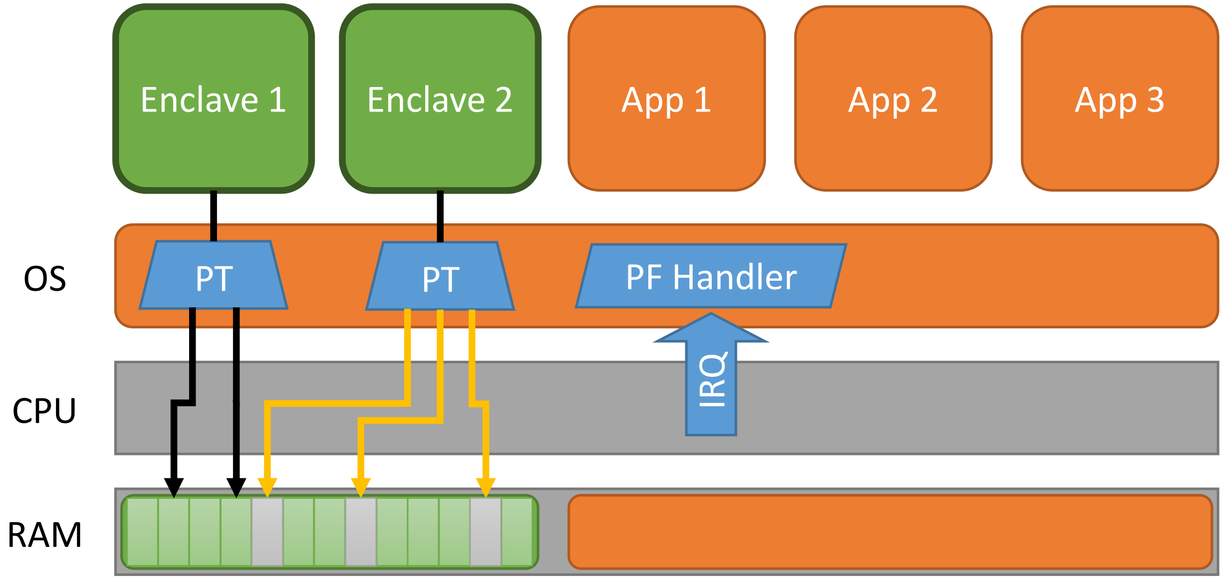


EPC: Enclave Page Cache PT: Page Tables PF: Page-Fault

Page Fault Attacks on SGX

Granularity: page 4K, good for big data structures

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



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

Page Fault Attacks on SGX

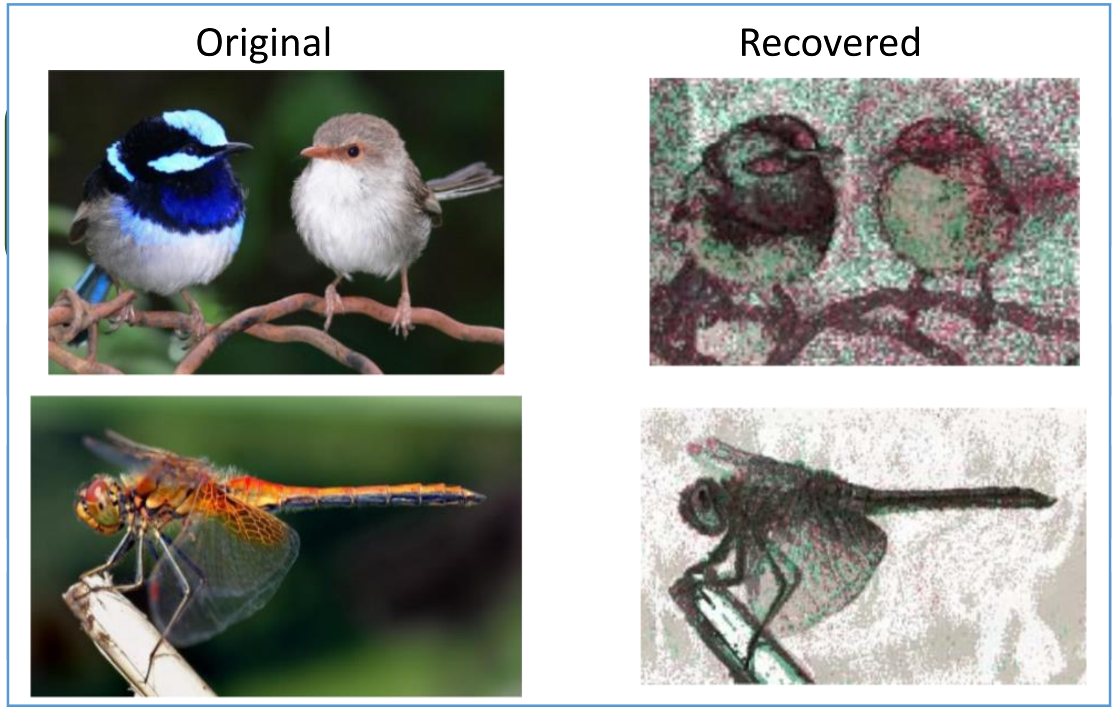
Granularity: page 4K, good for big data structures

[Xu et al., IEEE S&P’15]

OS

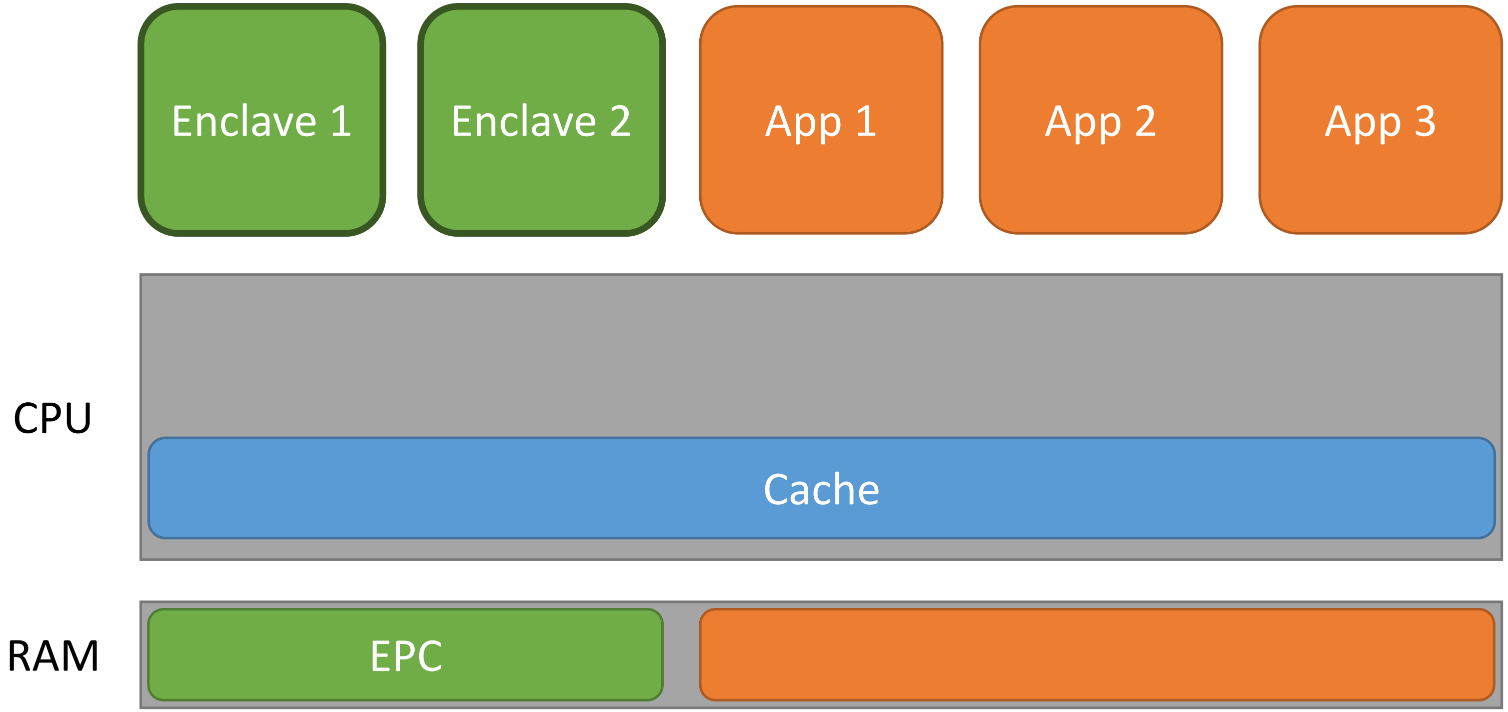
CPU

RAM



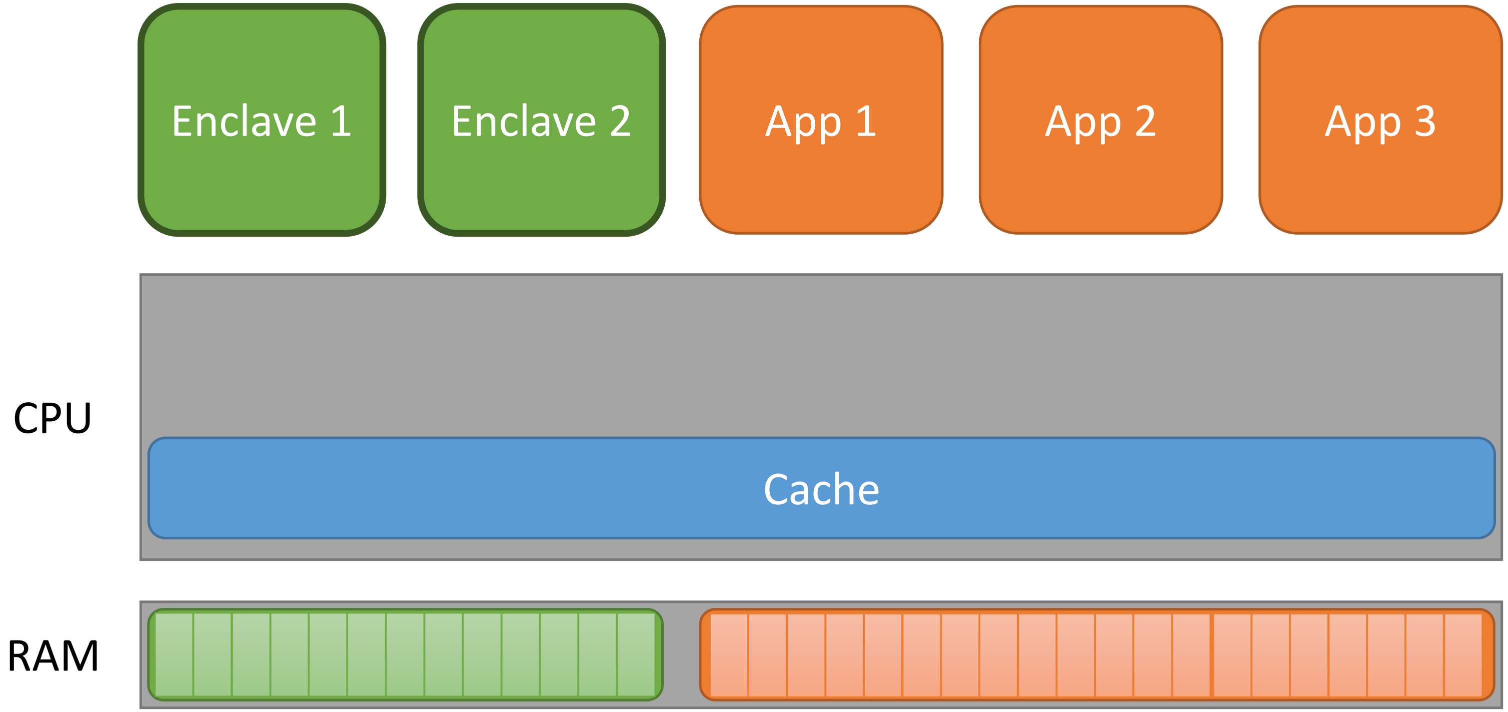
EPC: Enclave Page Cache PT: Page Tables PF: Page-Fault

Cache Attacks on SGX: Hack in The Box



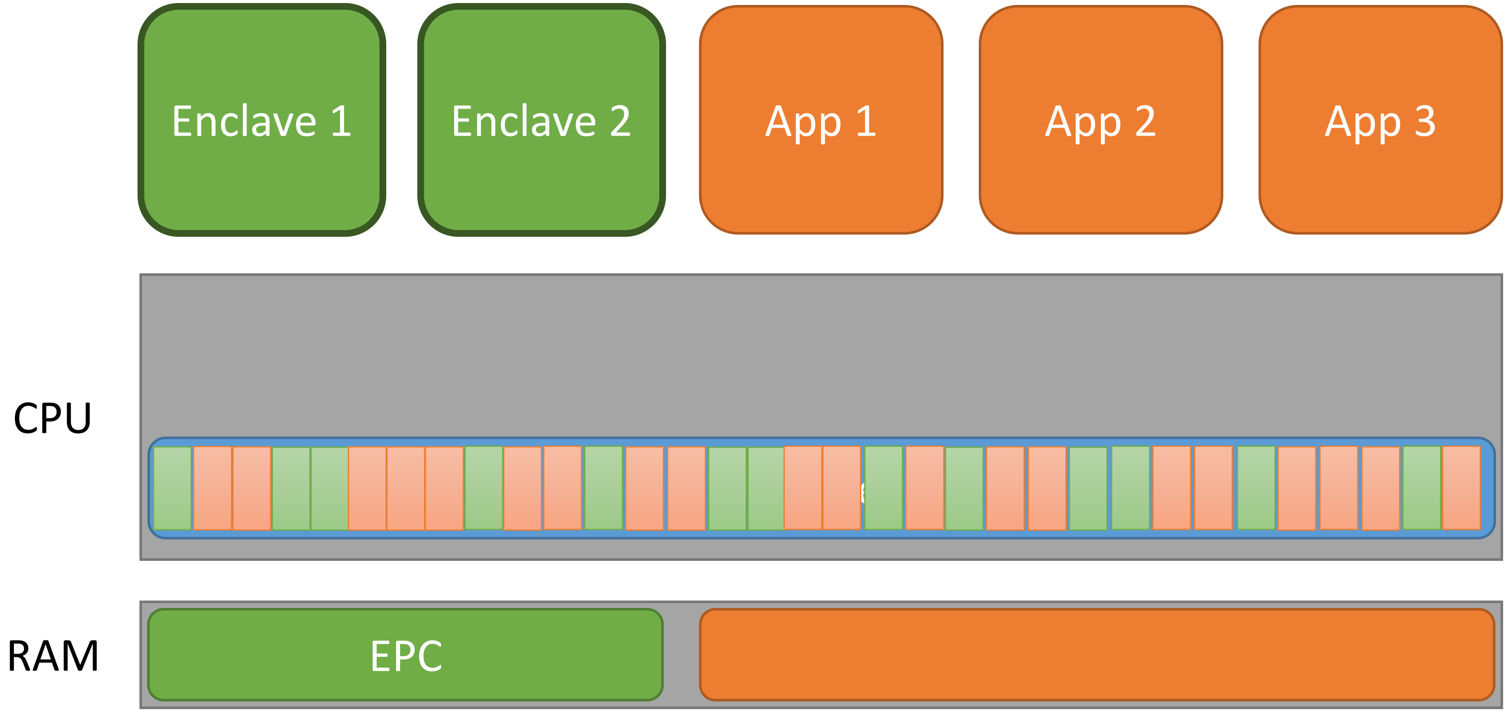
EPC: Enclave Page Cache

Cache Attacks on SGX: Hack in The Box



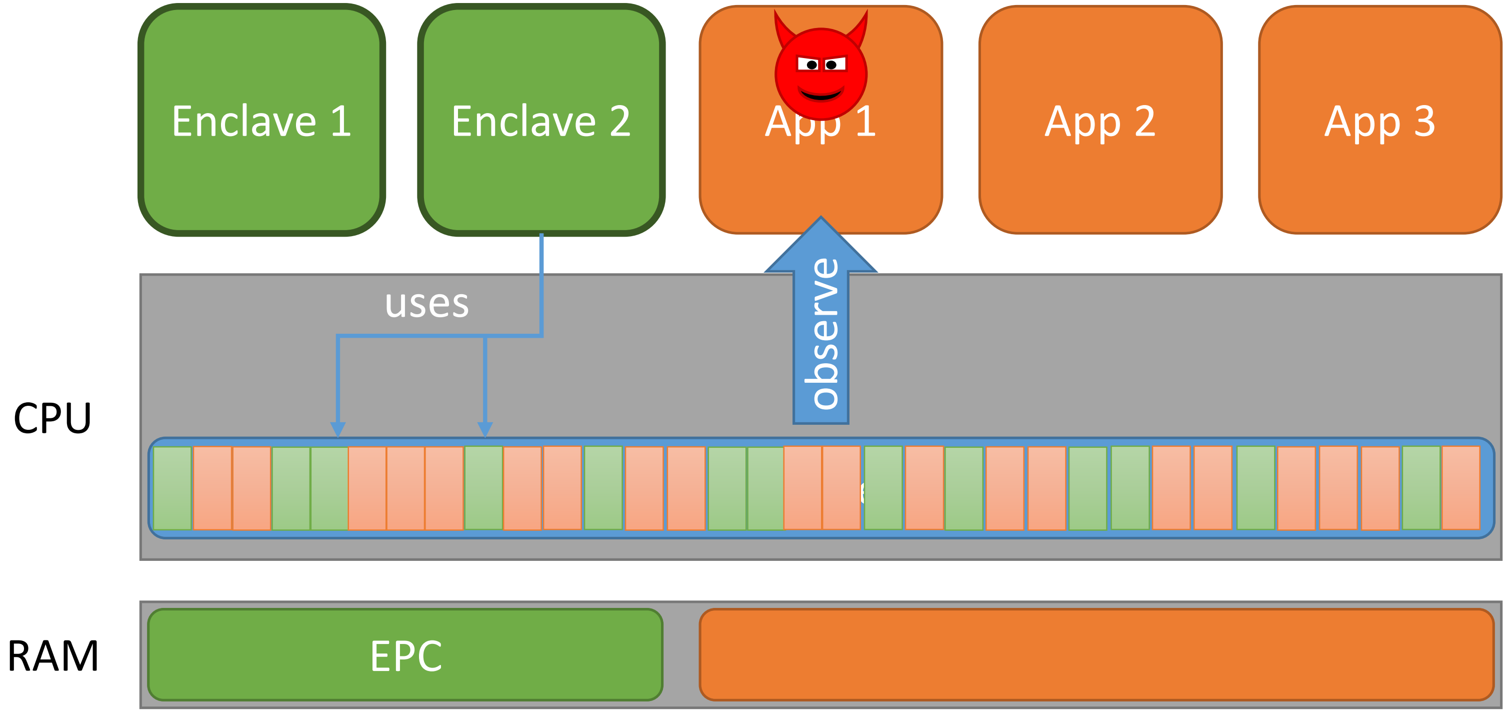
EPC: Enclave Page Cache

Cache Attacks on SGX: Hack in The Box



EPC: Enclave Page Cache

Cache Attacks on SGX: Hack in The Box

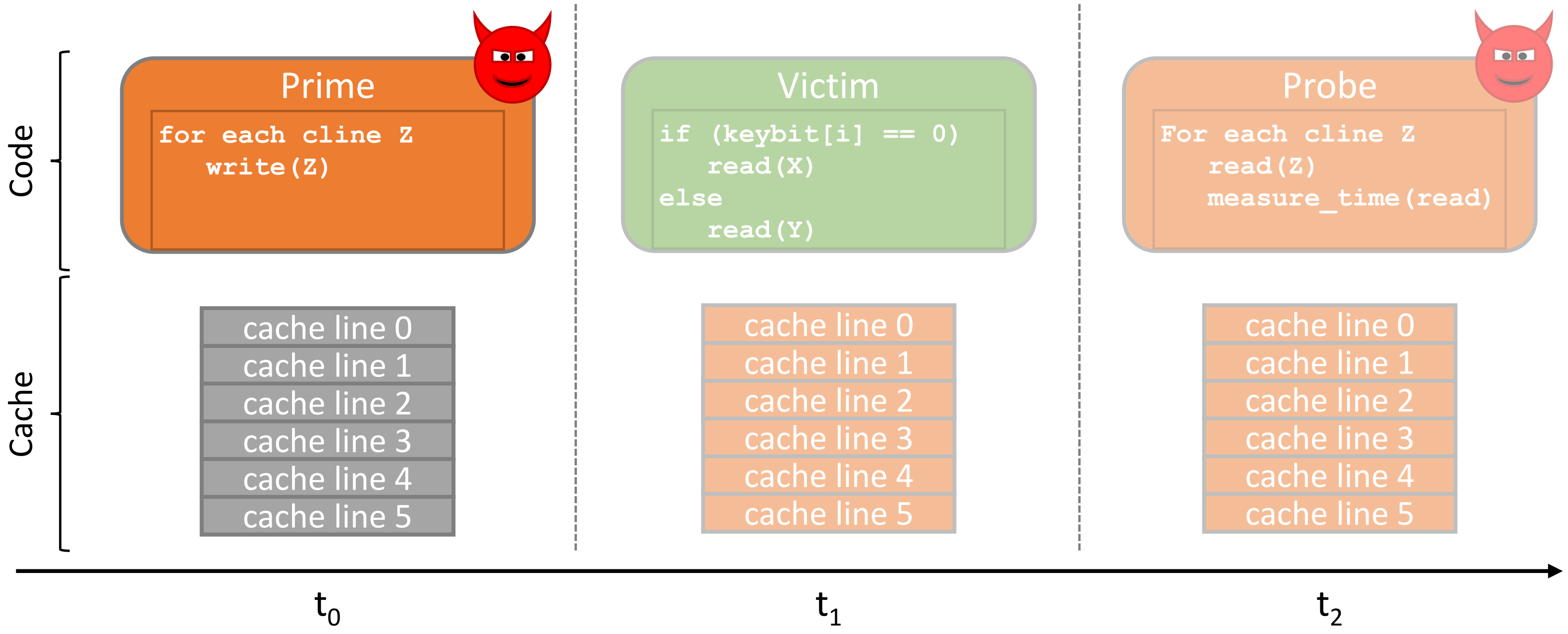


EPC: Enclave Page Cache

Side-Channel Attacks Basics:
Prime + Probe

Cache-based Side-Channel Attacks

Prime + Probe



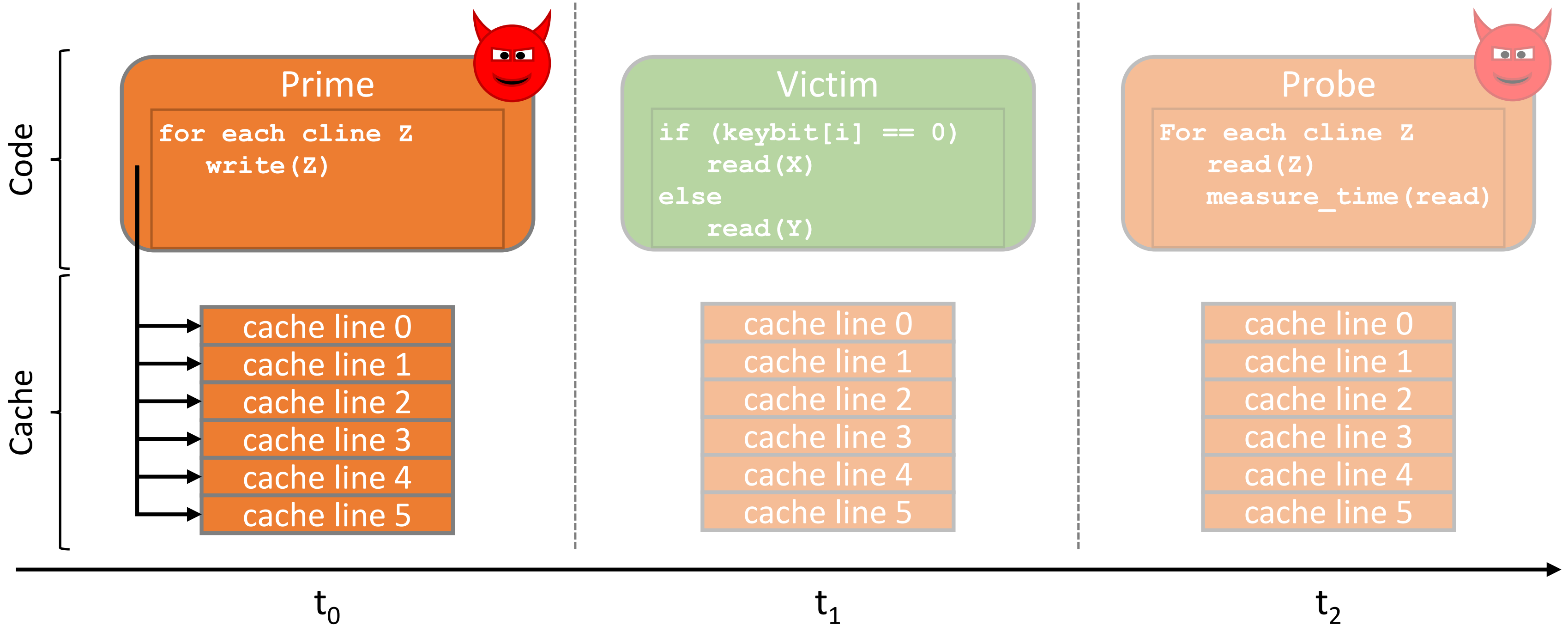
```
for each cline Z
    write(Z)
```

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

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

Cache-based Side-Channel Attacks

Prime + Probe



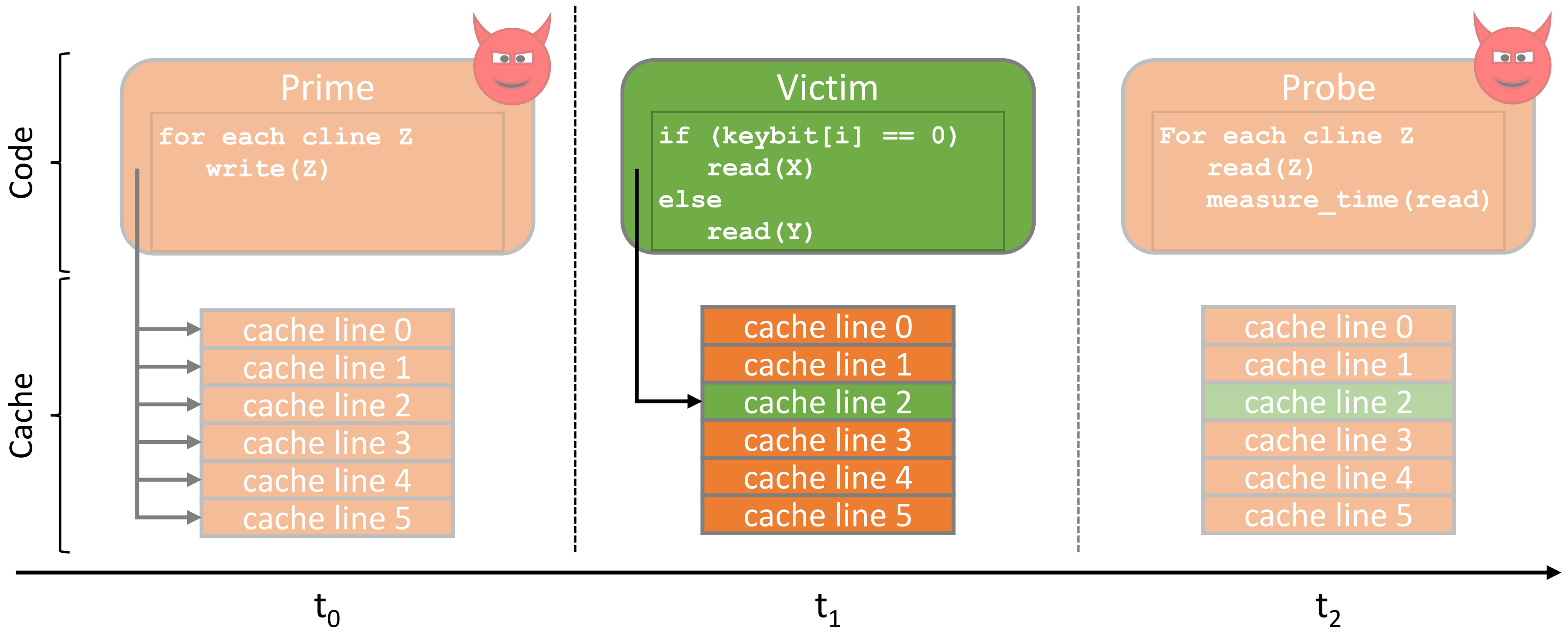
```
for each cline Z
   write(Z)
```

```
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



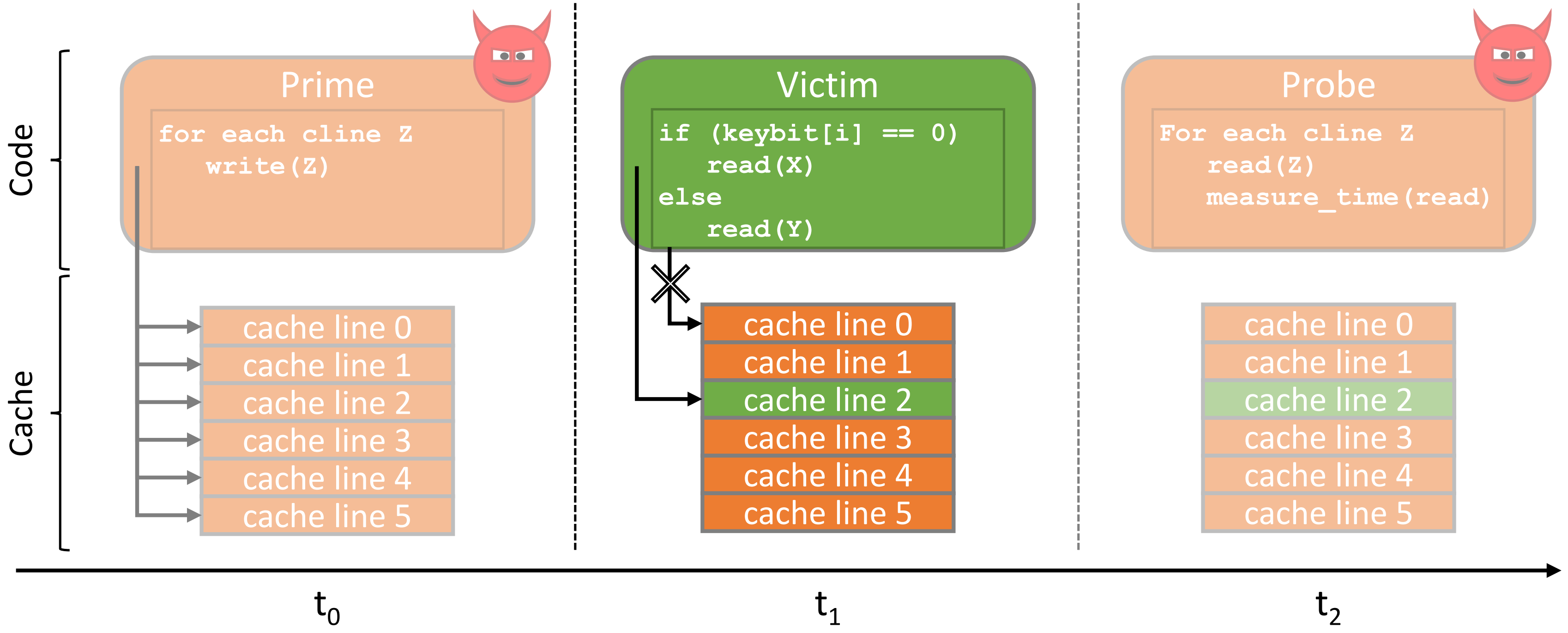
```
for each cline Z
    write(Z)
```

```
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


Prime

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

Victim

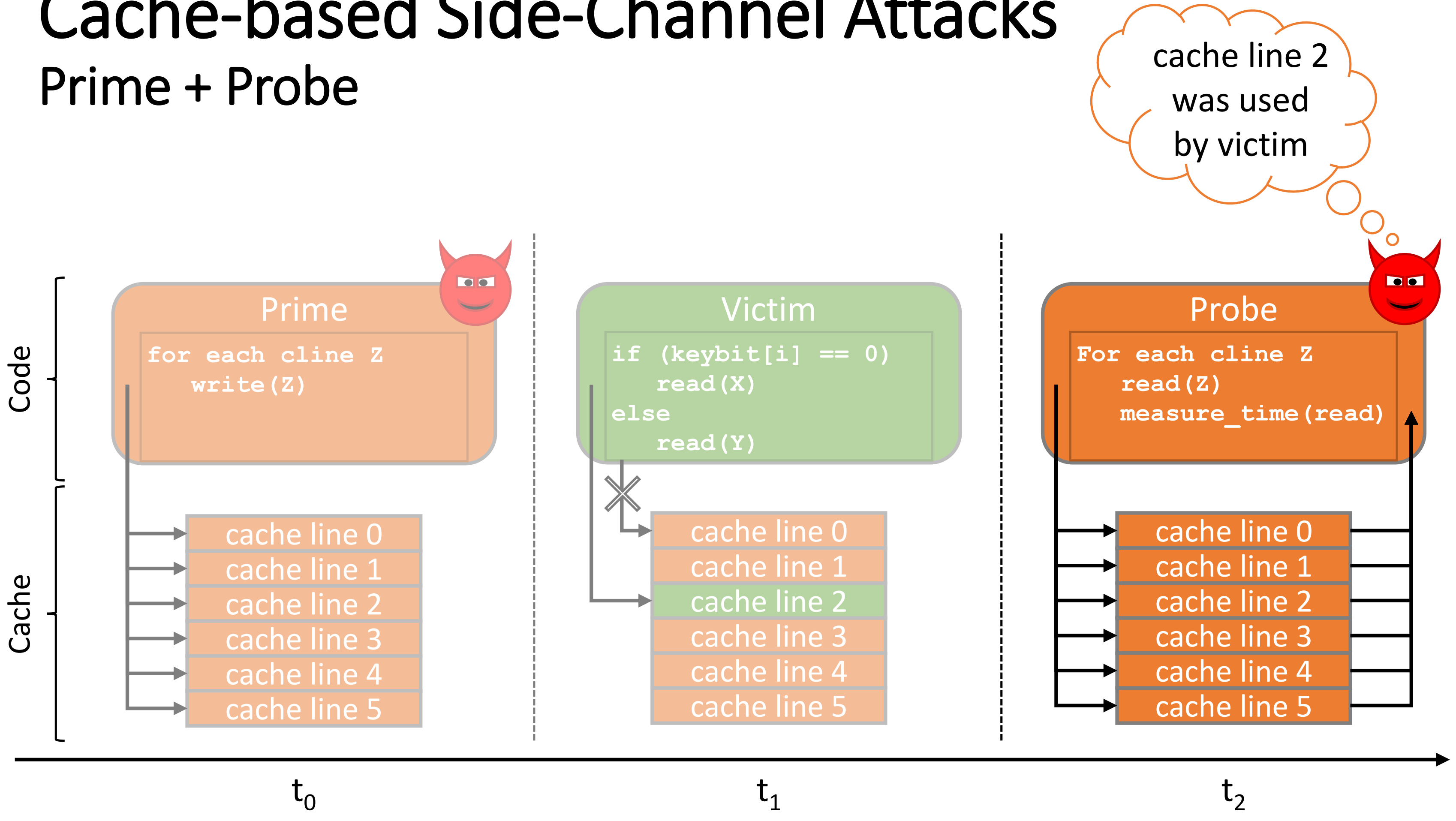
```
if (keybit[i] == 0)
    read(X)
else
    read(Y)
```

Probe

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

Cache-based Side-Channel Attacks

Prime + Probe



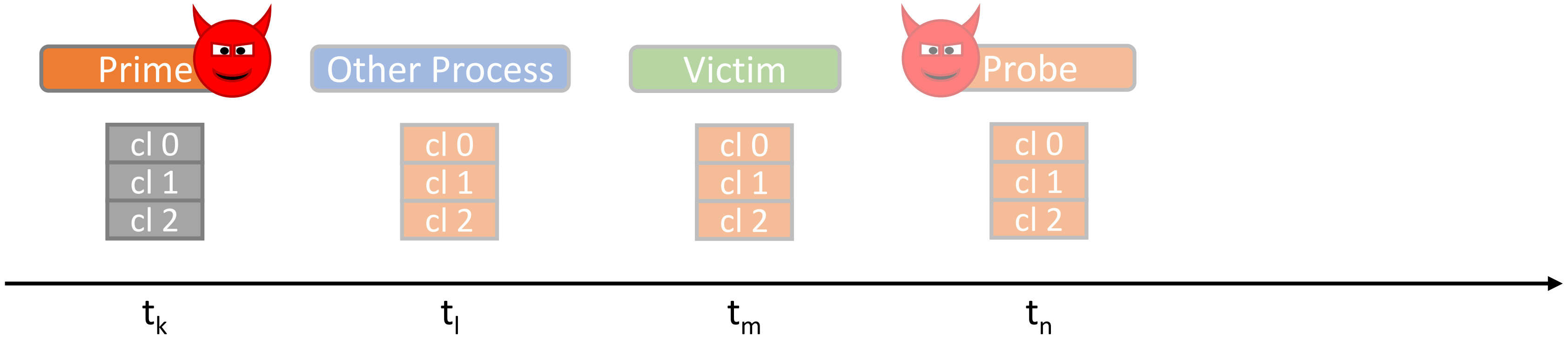
```
for each cline Z
    write(Z)
```

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

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

Side-Channel Attacker Challenge: Noise

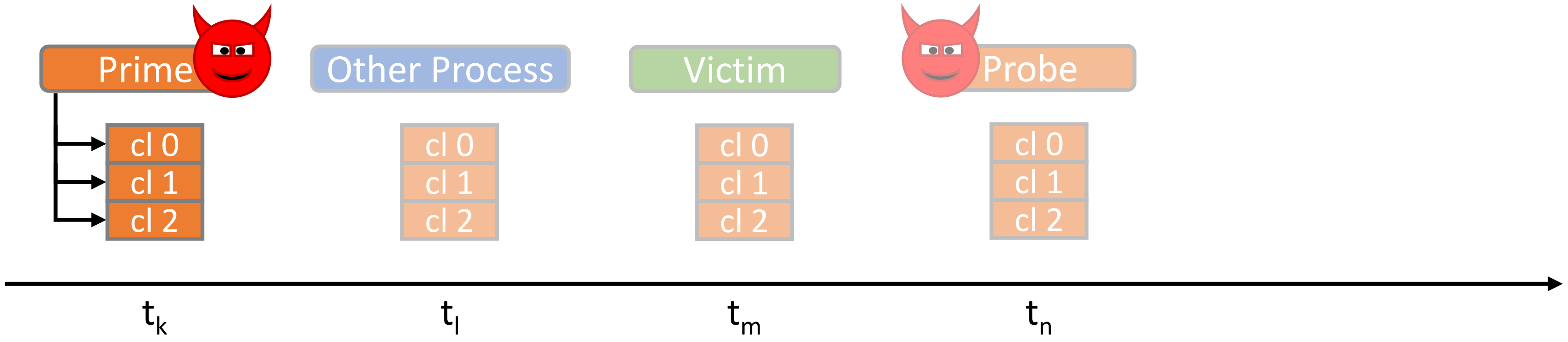
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

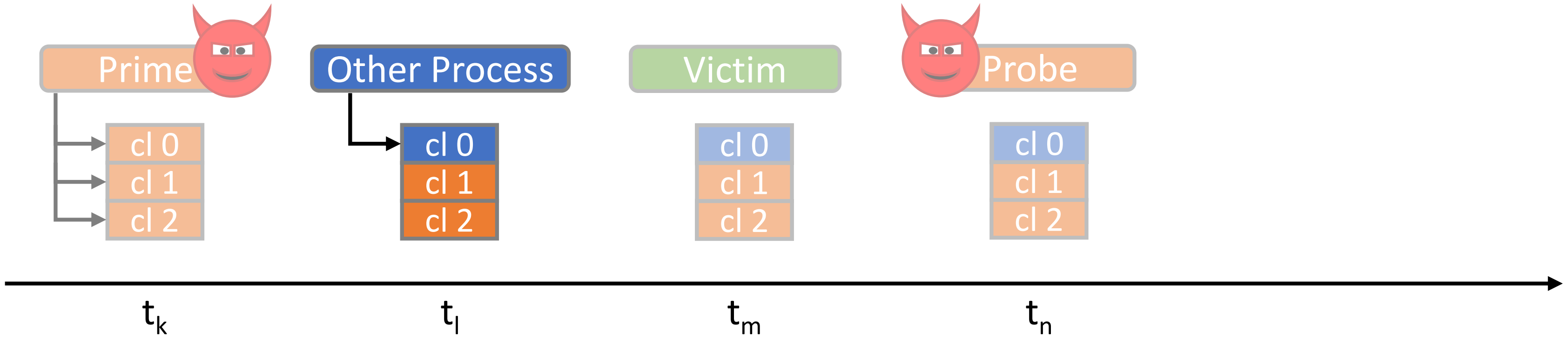
- "Classical" scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim's secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

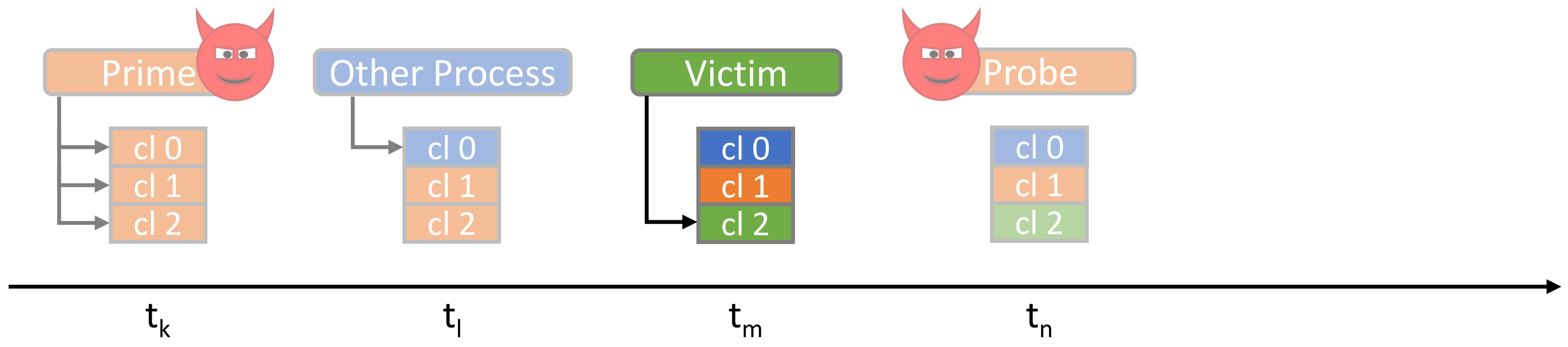
- "Classical" scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim's secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

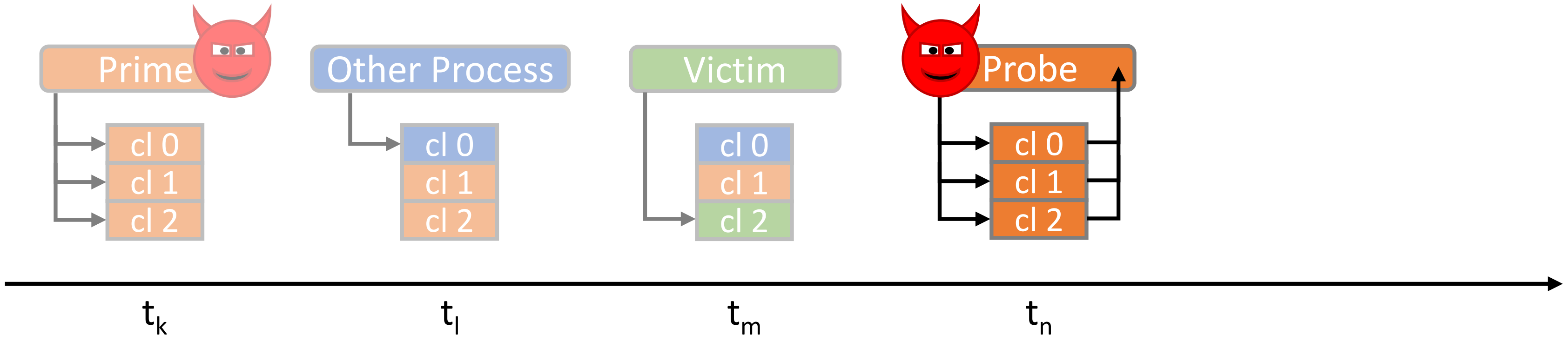
- “Classical” scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim’s secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

Side-Channel Attacker Challenge: Noise

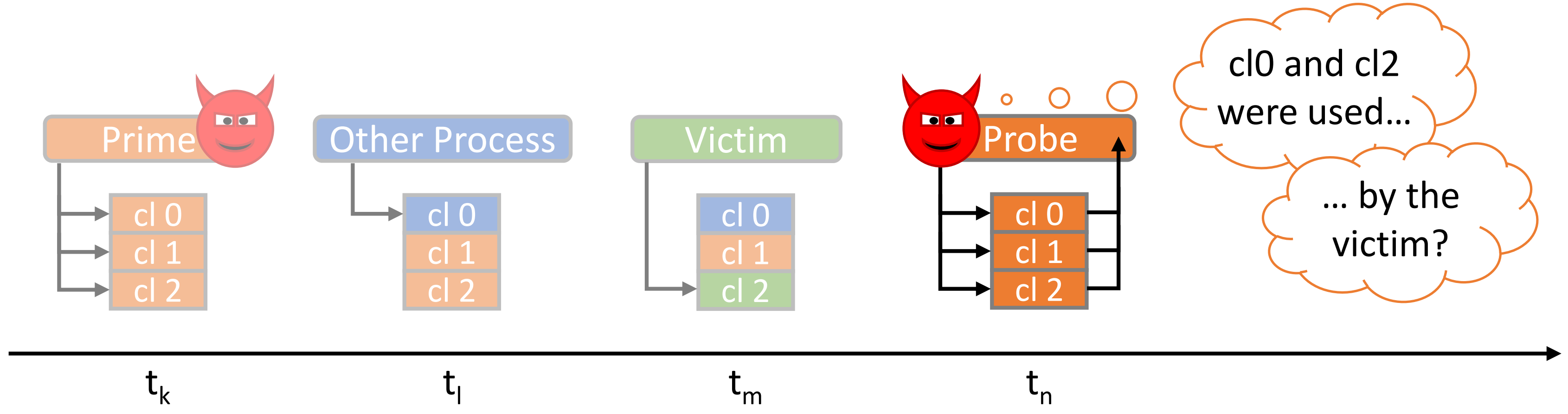
- "Classical" scenario: unprivileged attacker
- OS* is not collaborating with the attacker
  - OS can directly access process memory containing the victim's secret
  - System operates normally, impacting the caches (process scheduling, context switches, interrupts, etc.)



*OS: Operating System and any other privileged system software

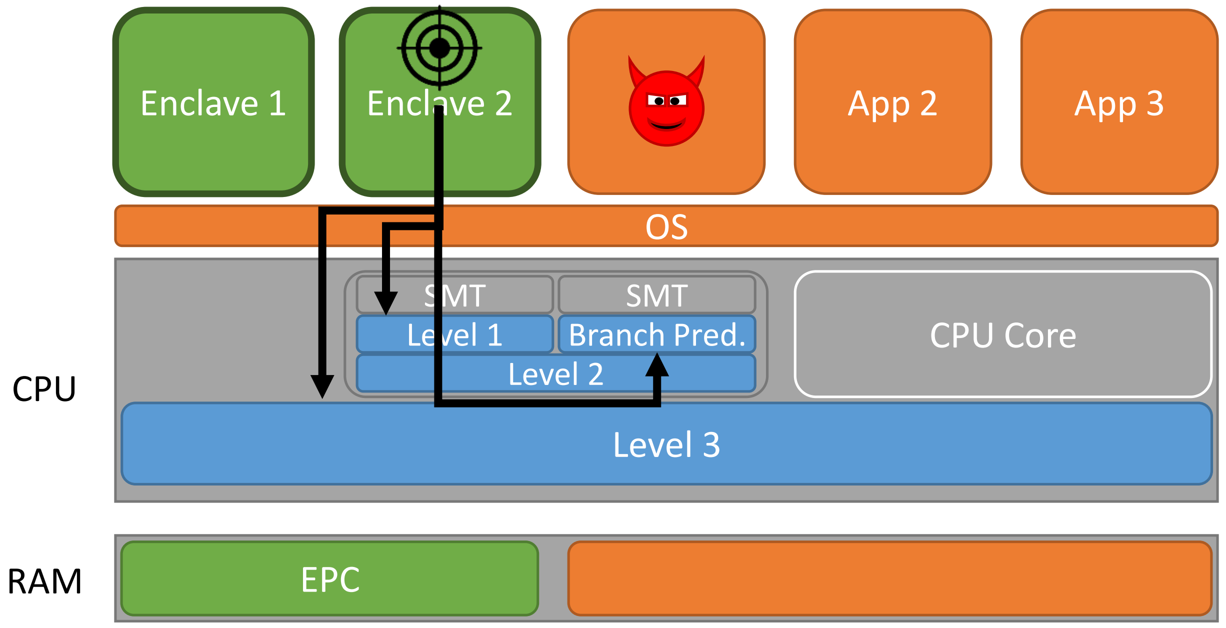
Side-Channel Attacker Challenge: Noise

- “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.)



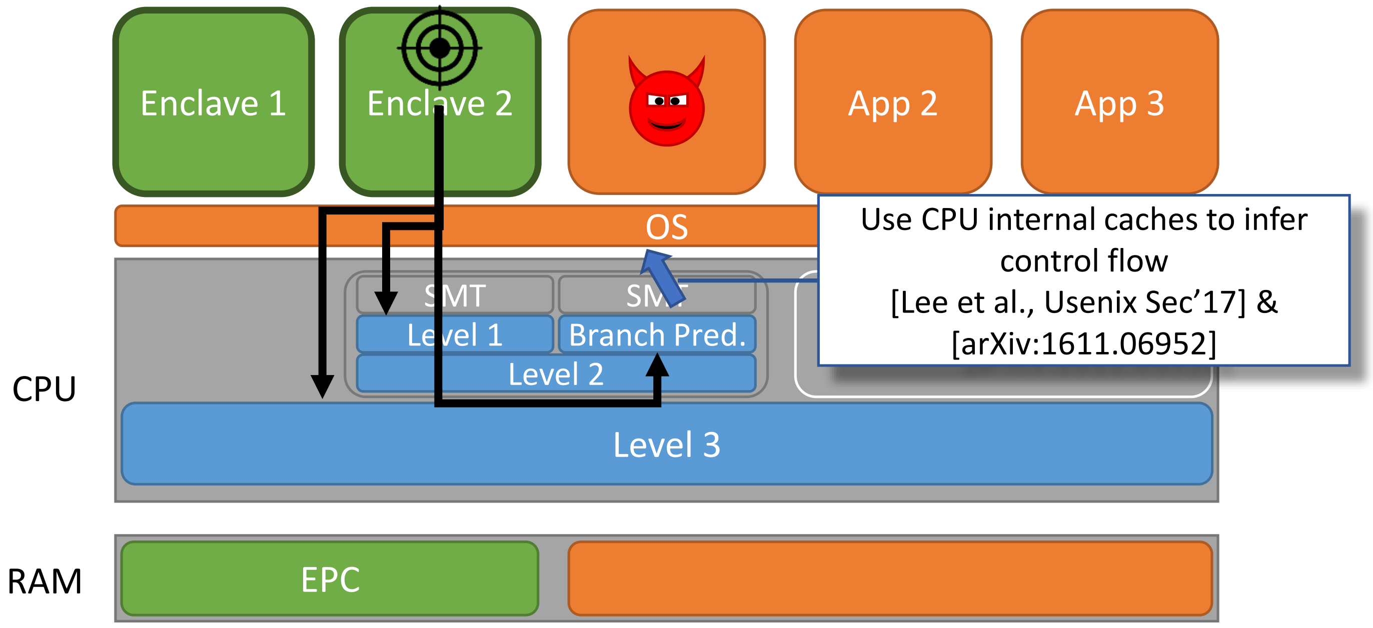
*OS: Operating System and any other privileged system software

Cache Attacks on SGX



EPC: Enclave Page Cache   SMT: Simultaneous Multithreading

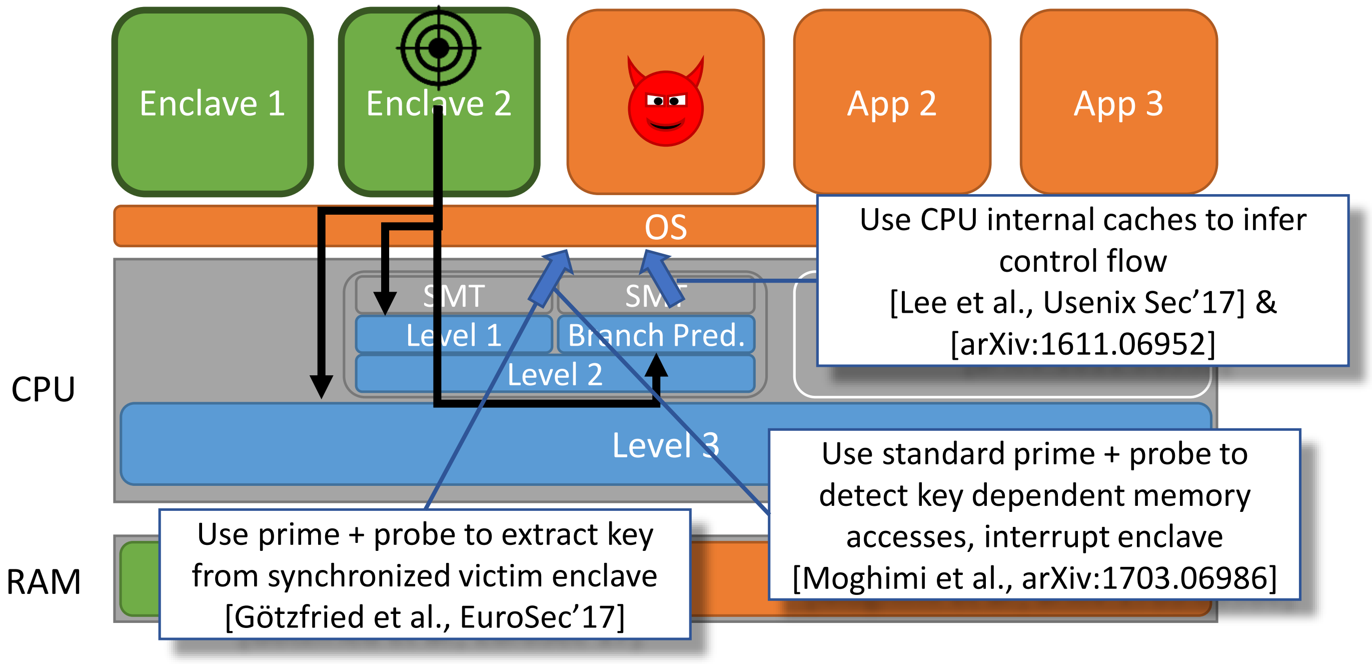
Cache Attacks on SGX



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



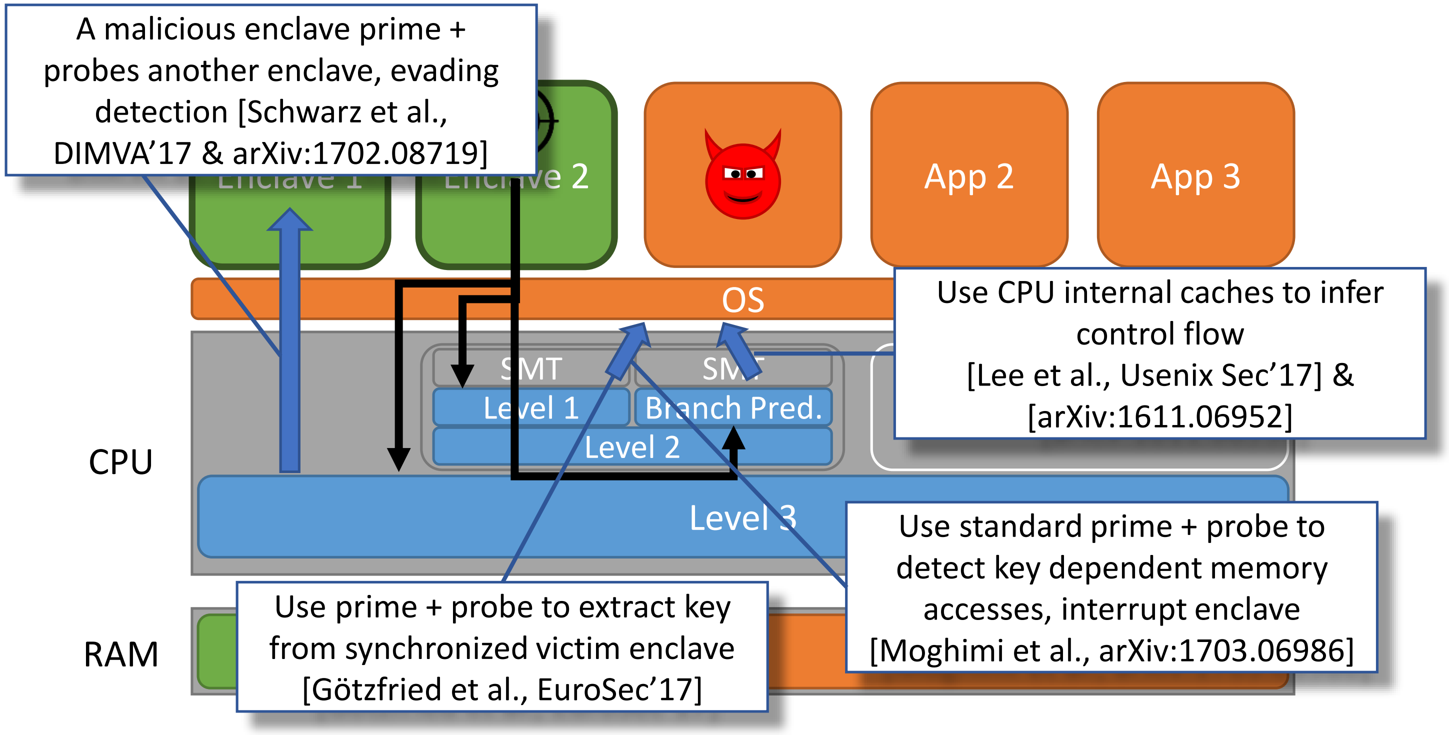
Use CPU internal caches to infer control flow [Lee et al., Usenix Sec'17] & [arXiv:1611.06952]

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]

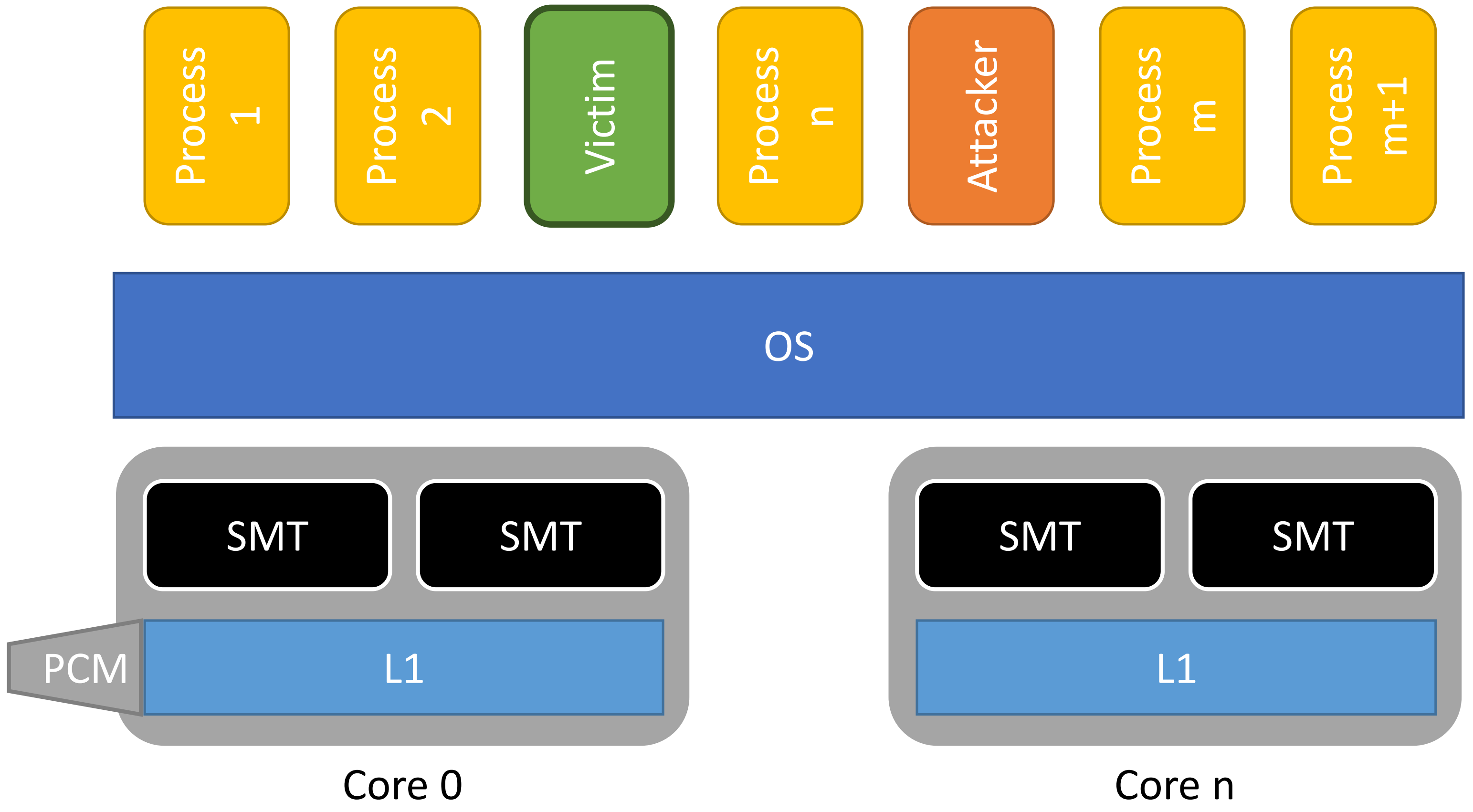
EPC: Enclave Page Cache SMT: Simultaneous Multithreading

Current attacks

- Rely on (frequently) interrupting enclaves
  - Can be detected
- Make strong assumptions
  - Assume synchronization between victim and attacker

Our Attack

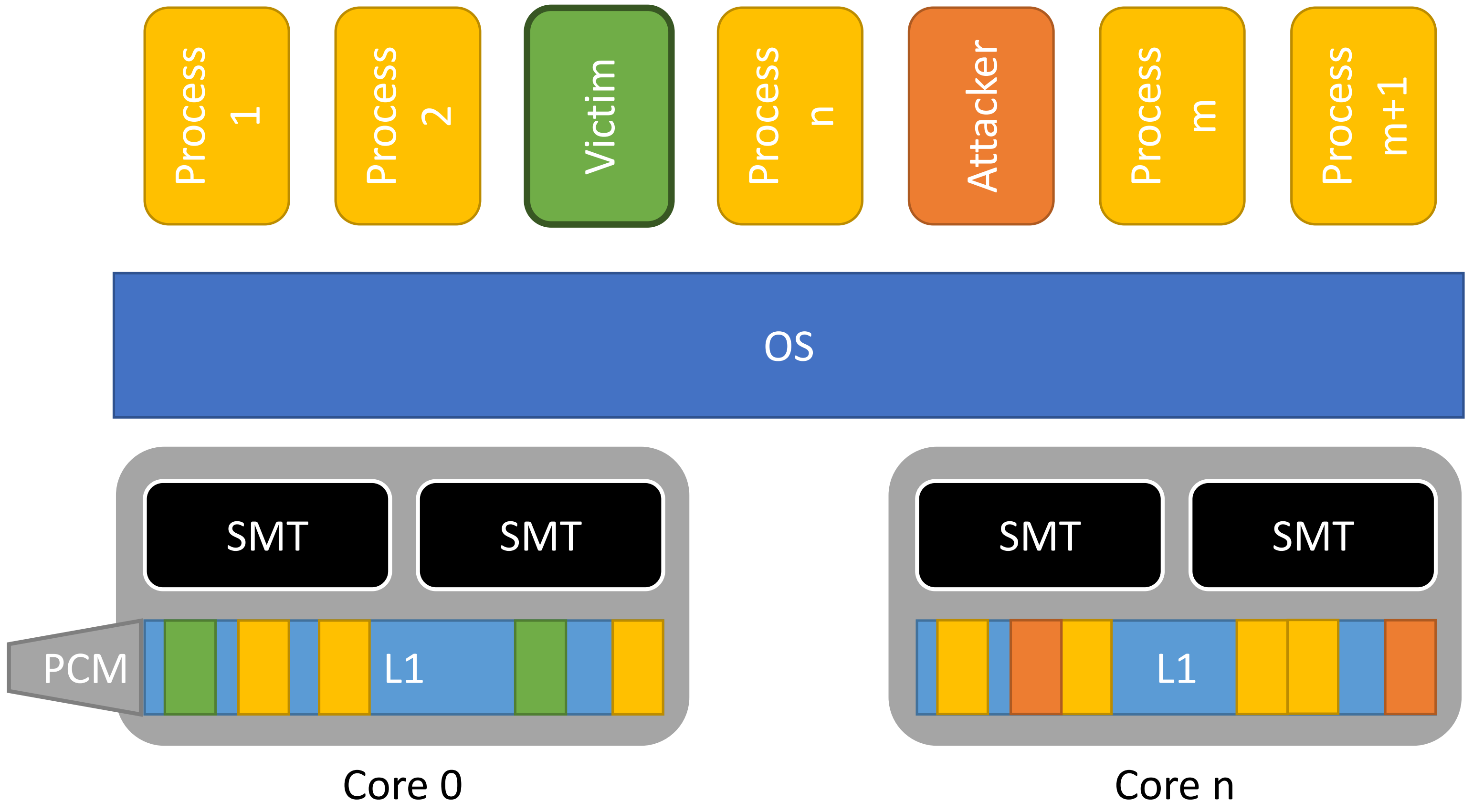
[Brasser et al., WOOT'17]



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

Our Attack

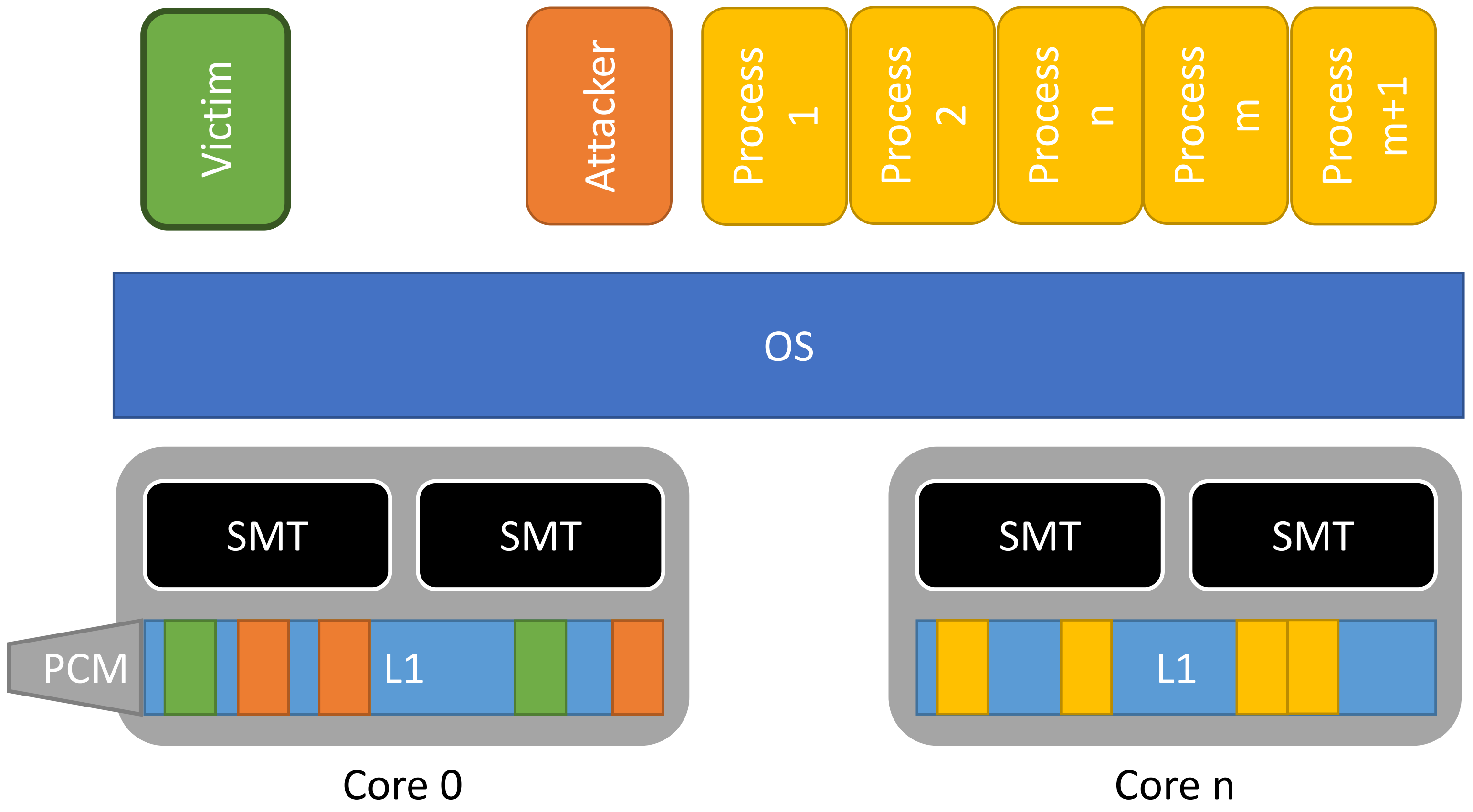
[Brasser et al., WOOT’17]



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

Our Attack

[Brasser et al., WOOT'17]



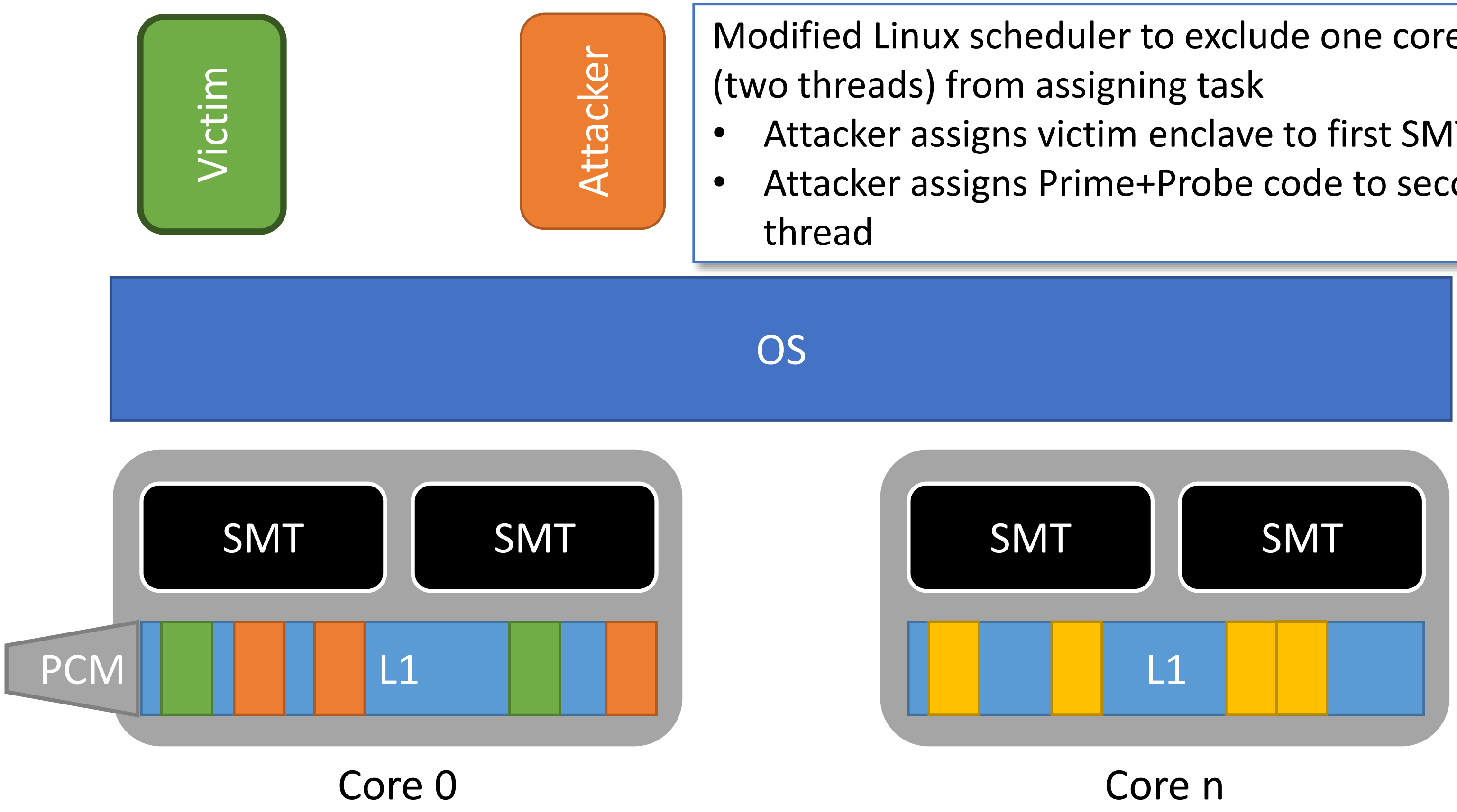
PCM: Performance Counter Monitor | SMT: Simultaneous Multithreading | APIC: Advanced Programmable Interrupt Controller

Our Attack

[Brasser et al., WOOT’17]

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

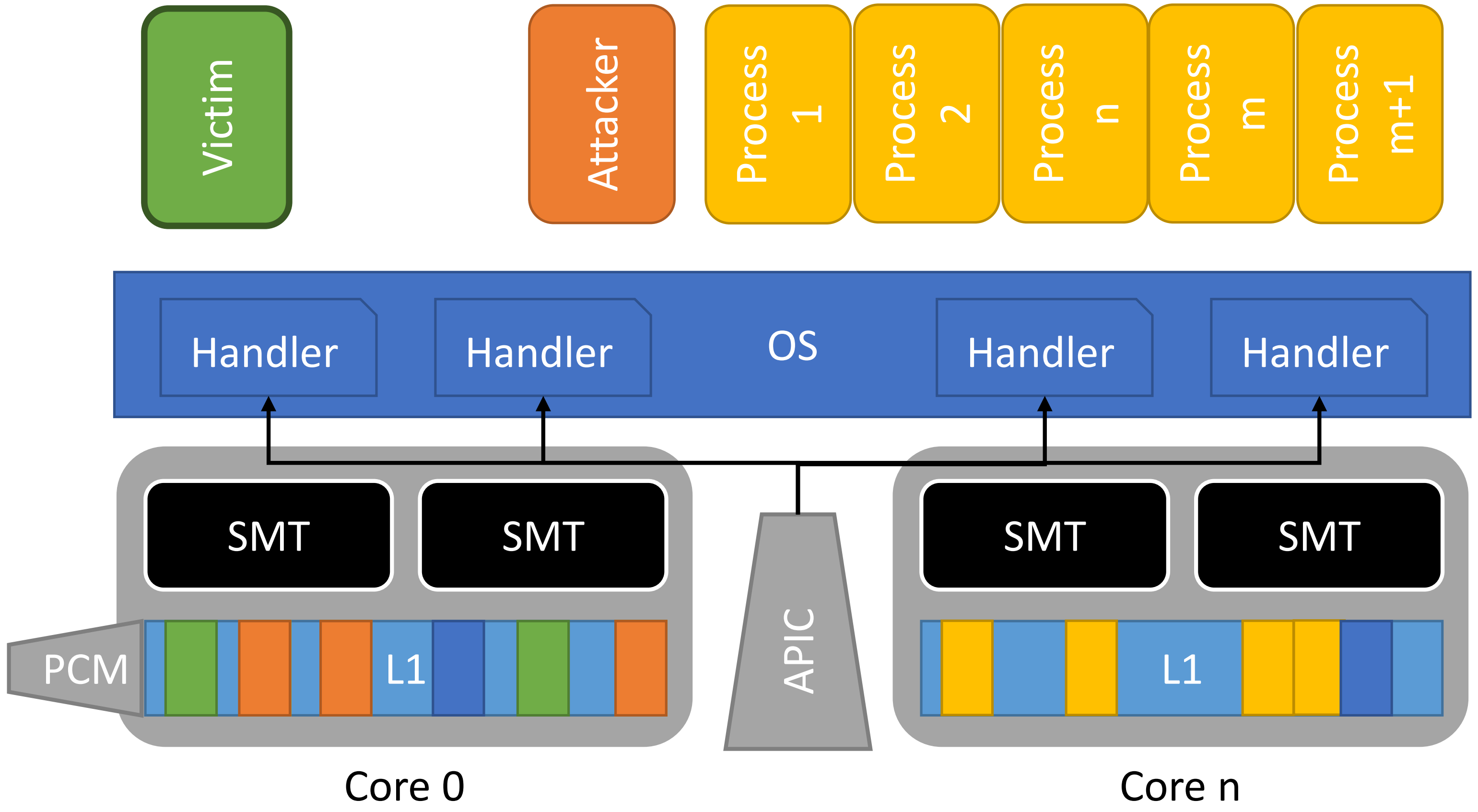
- Attacker assigns victim enclave to first SMT thread
- Attacker assigns Prime+Probe code to second SMT thread



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

Our Attack

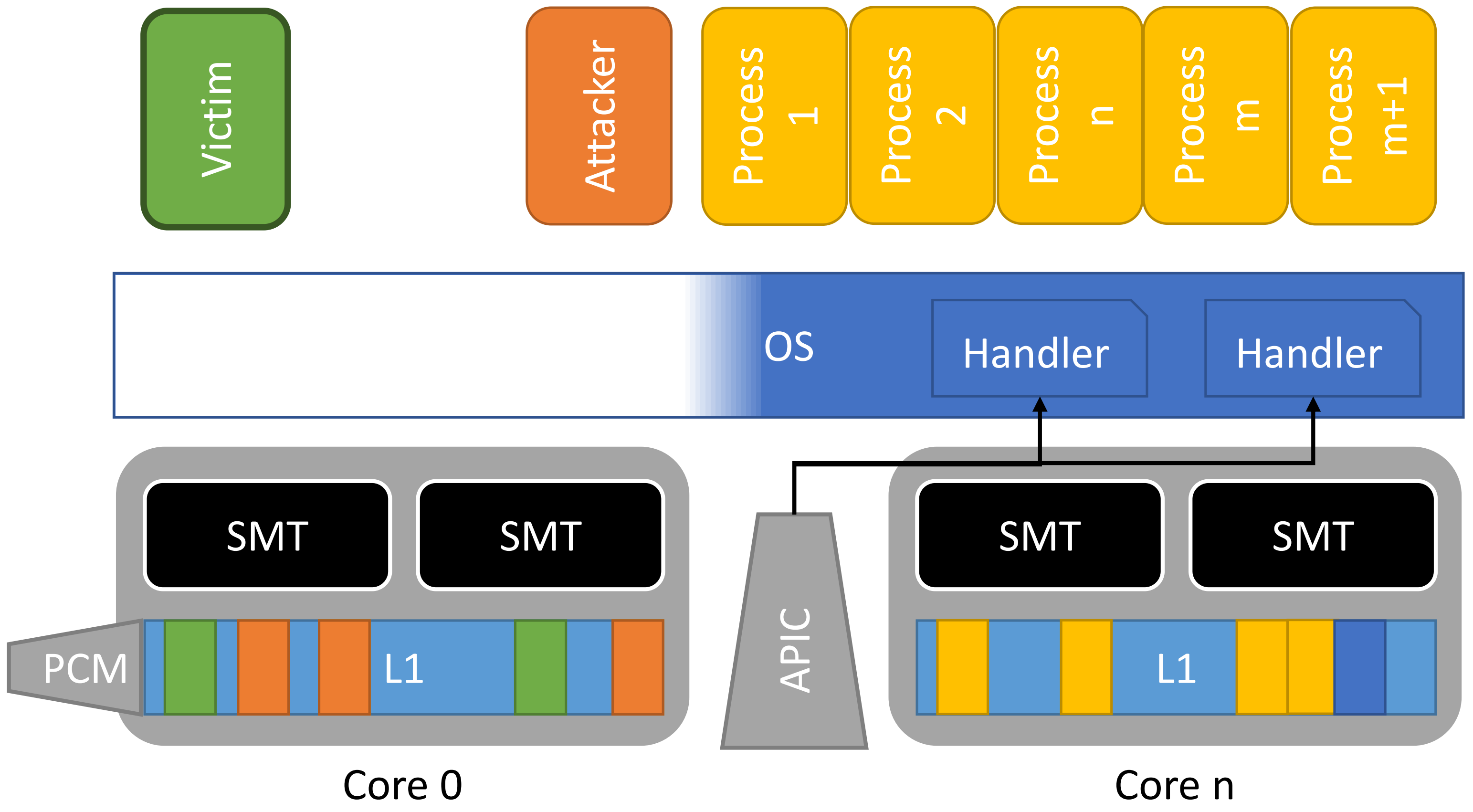
[Brasser et al., WOOT’17]



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

Our Attack

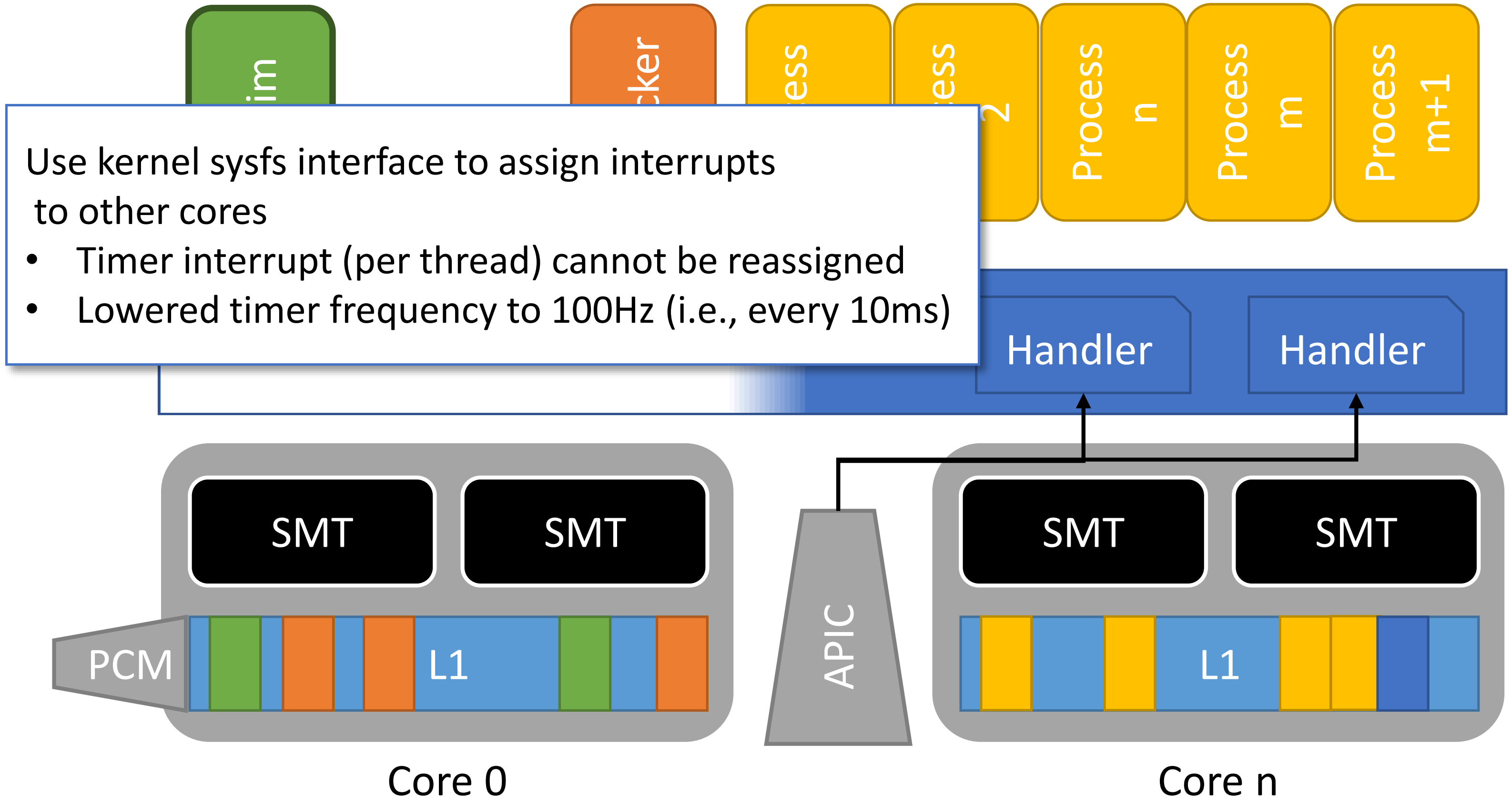
[Brasser et al., WOOT’17]



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

Our Attack

[Brasser et al., WOOT’17]



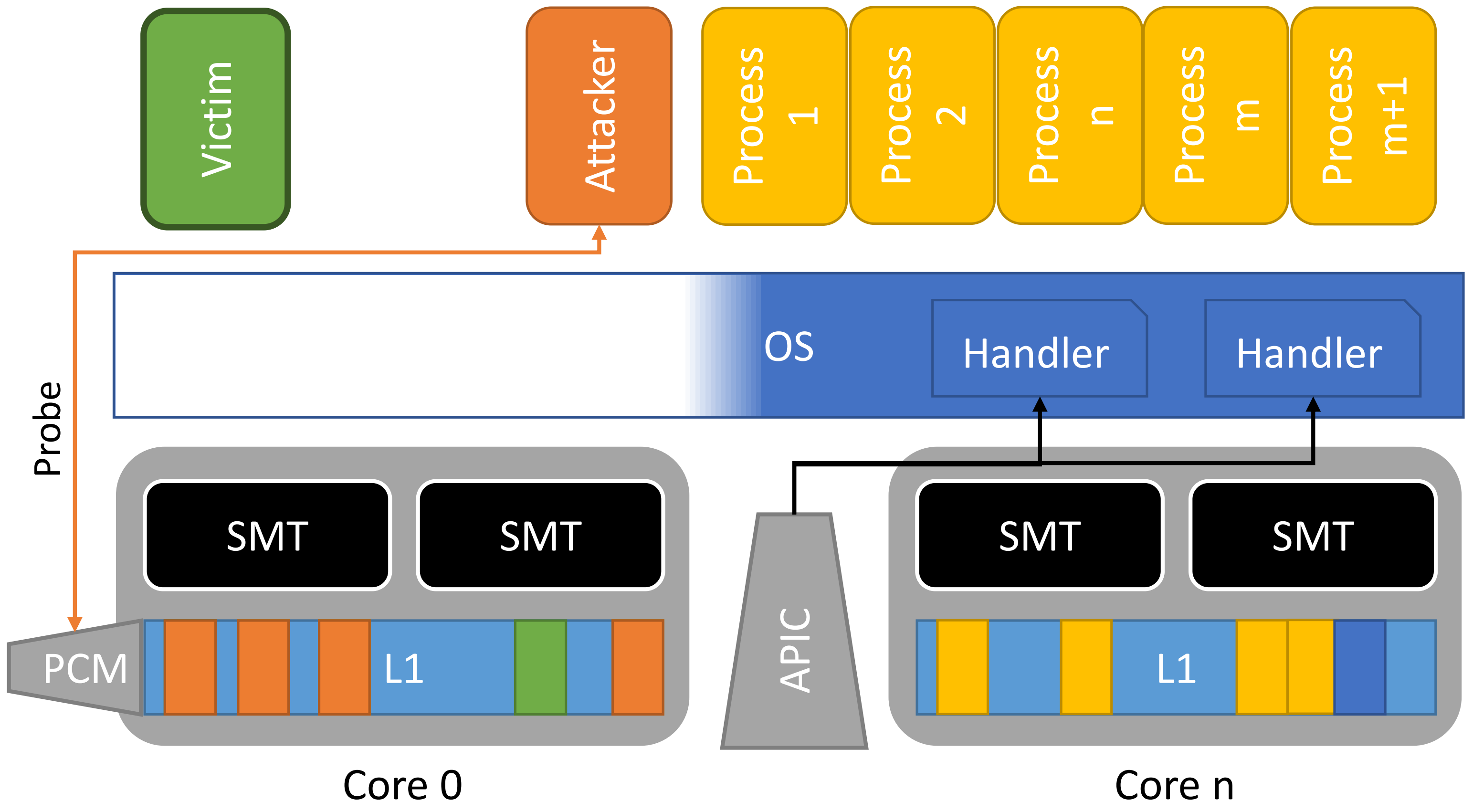
Use kernel sysfs interface to assign interrupts to other cores

- 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

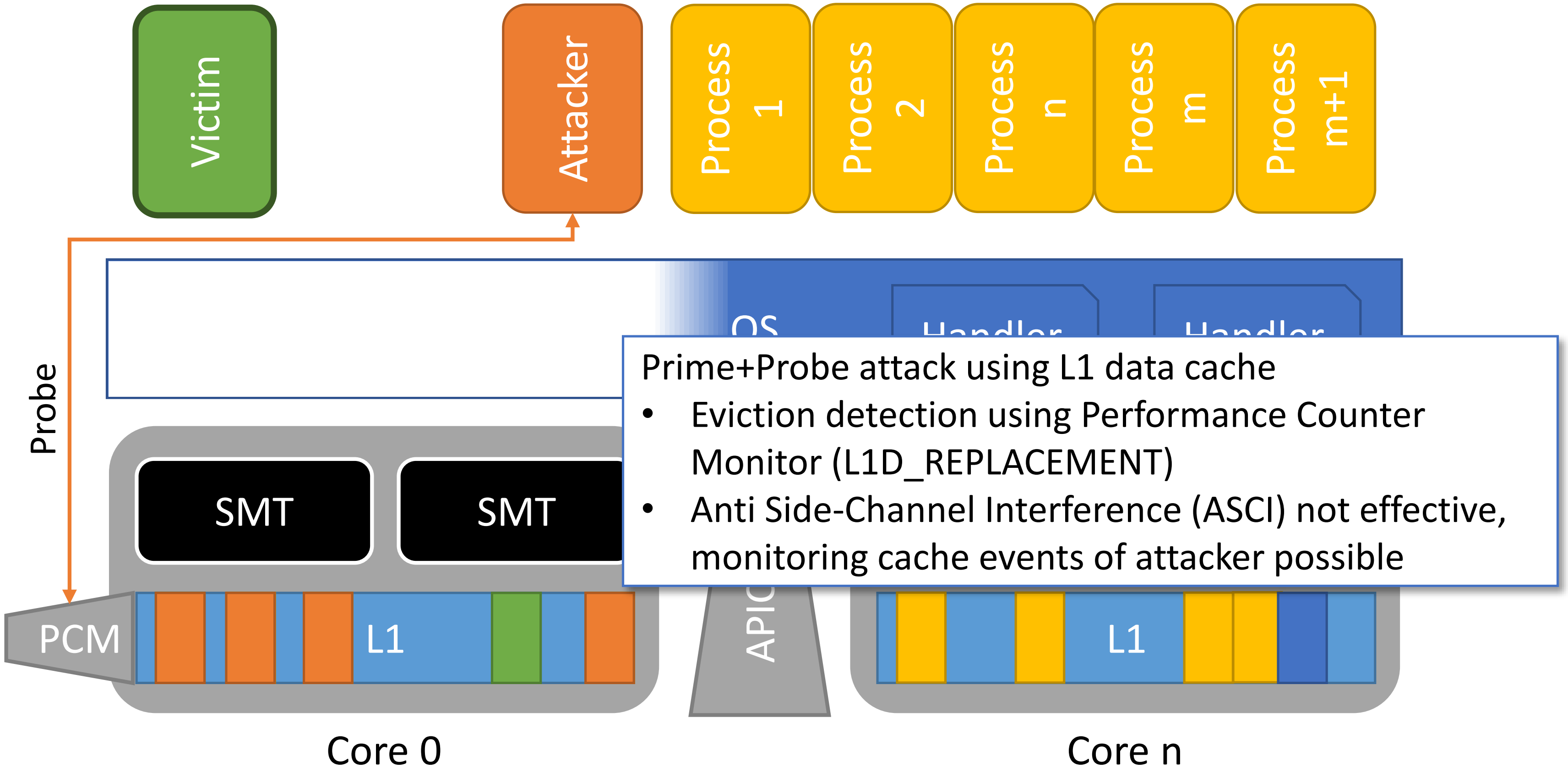
[Brasser et al., WOOT’17]



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

Our Attack

[Brasser et al., WOOT'17]



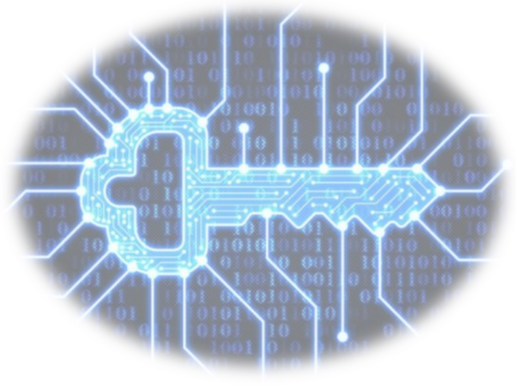
Prime+Probe attack using L1 data cache

- 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

Our Attack Use-Cases

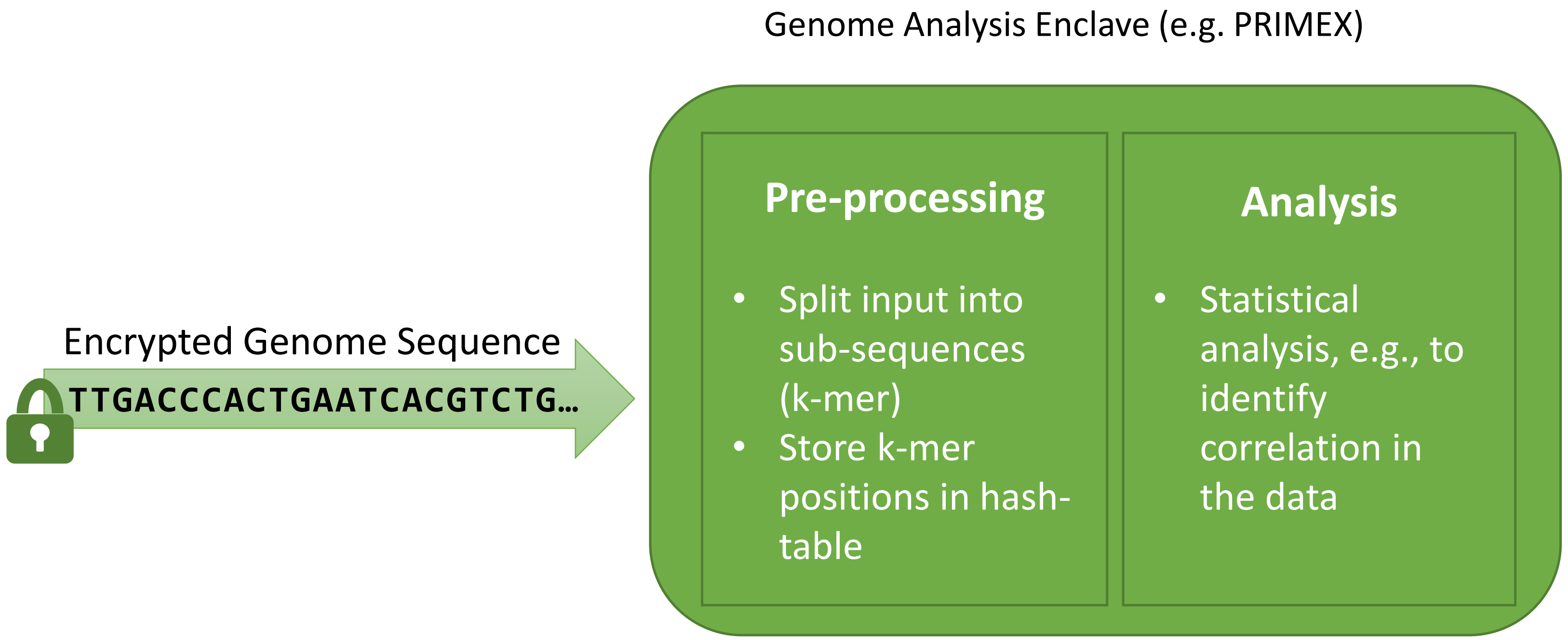
Extracting 2048-bit RSA decryption key



[arXiv:1702.07521]

Extracting genome sequences

Genome Sequencing


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

Pre-processing

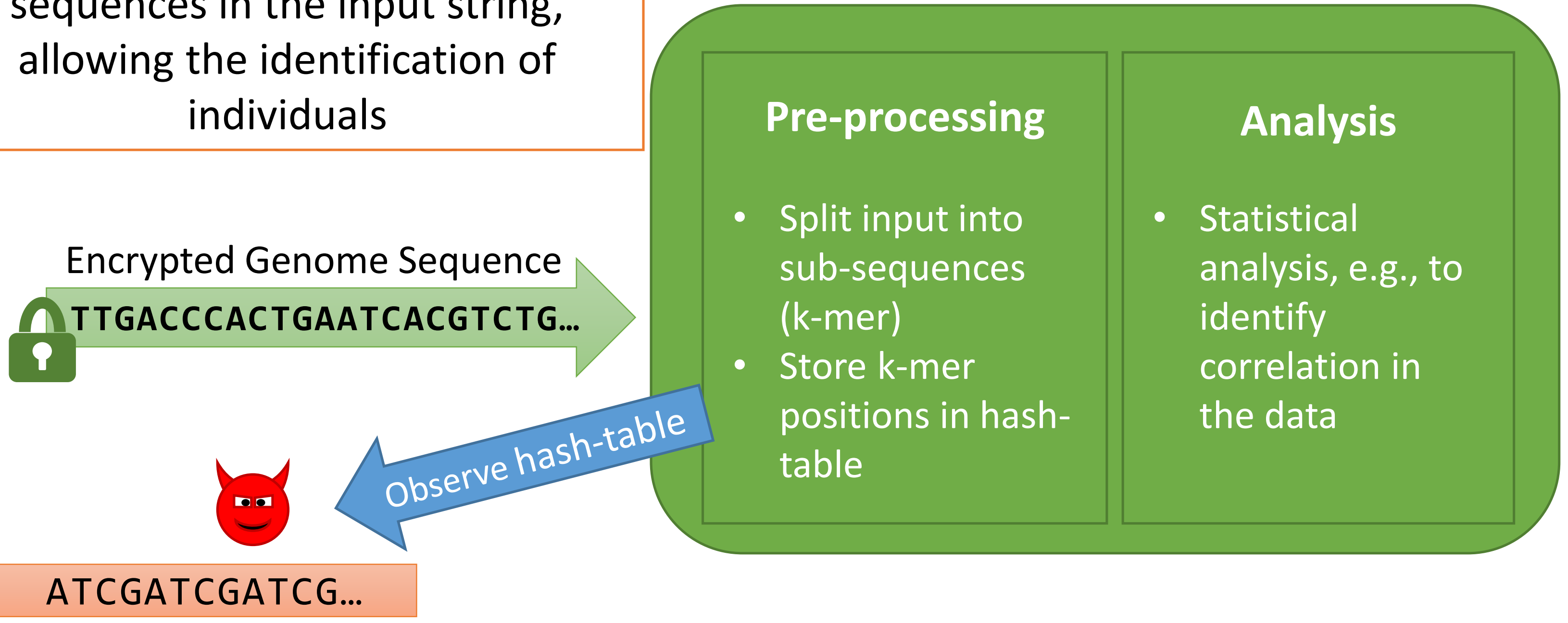
- Split input into sub-sequences (k-mer)
- Store k-mer positions in hash-table

Analysis

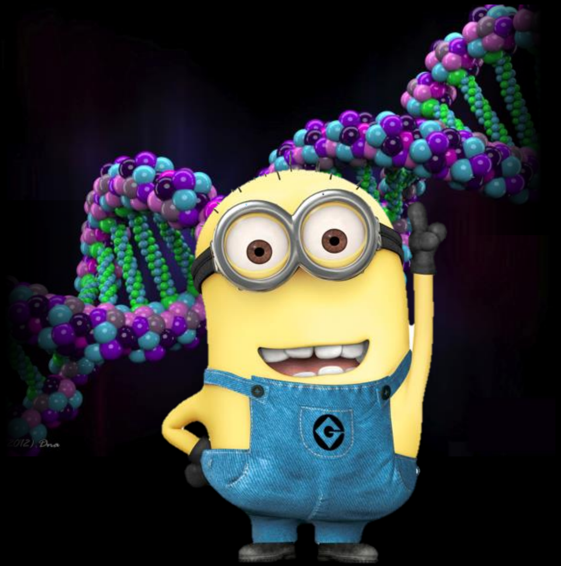
- Statistical analysis, e.g., to identify correlation in the data

Genome Sequencing

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


Genome Analysis Enclave (e.g. PRIMEX)
Encrypted Genome Sequence
TTGACCCACTGAATCACGTCTG…
Observe hash-table
ATCGATCGATCG…
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 Genomes



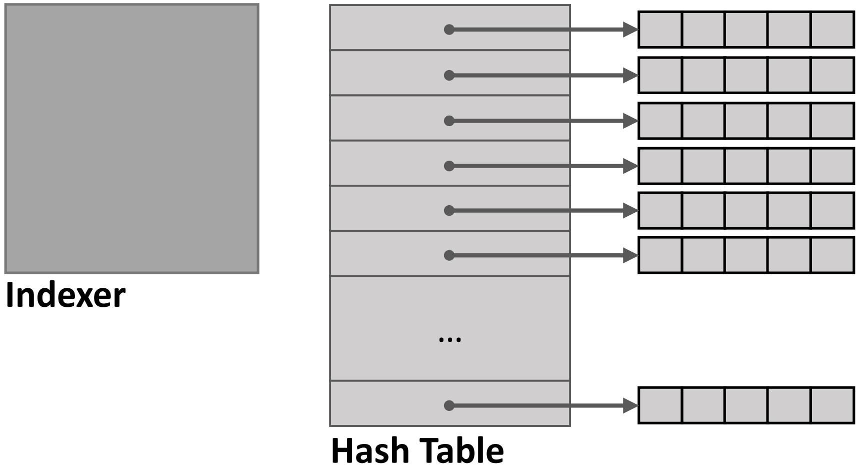
Human Genome

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

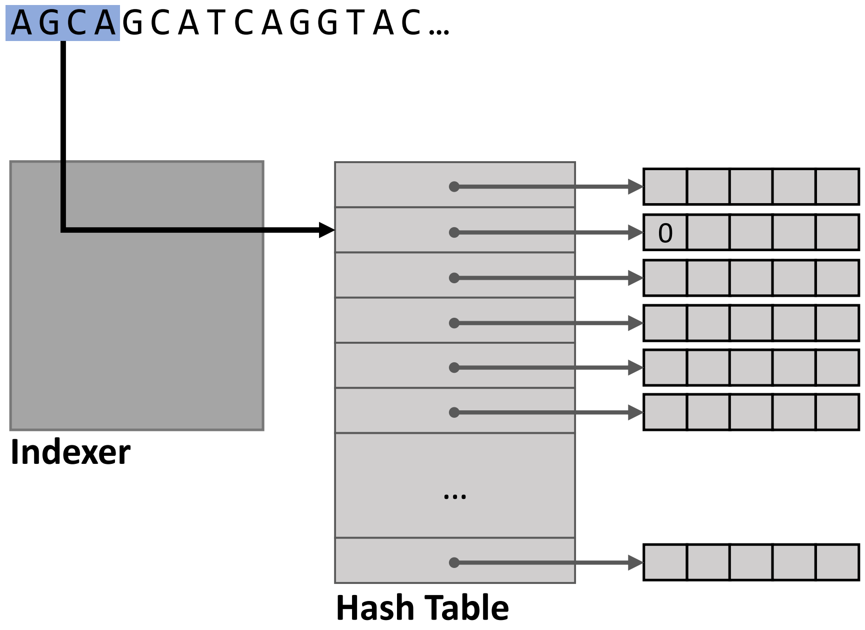
```
TTGACCCACTGAATCACGTCTGACCGCGCGTACGCGG
TCACTTGCGGTGCCGTTTTCTTTGTTACCGACGACCG
ACCAGCGACAGCCACCGCGCGCTCACTGCCACCAAAA
GAGTCATATCGATCGATCGATCGATCGATCGATCGAT
CGATCGATCGATCGATCGATCGATCGATCGATCATCA
CAGCCGACCAGTTTCTGGAACGTTCCCGATACTGGAA
CGGTCCTAATGCAGTATCCCACCCTCCTTCCATCGAC
GCCAGTCGAATCACGCCGCCAGCCACCGTCCGCCAGC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

Genome Preprocessing

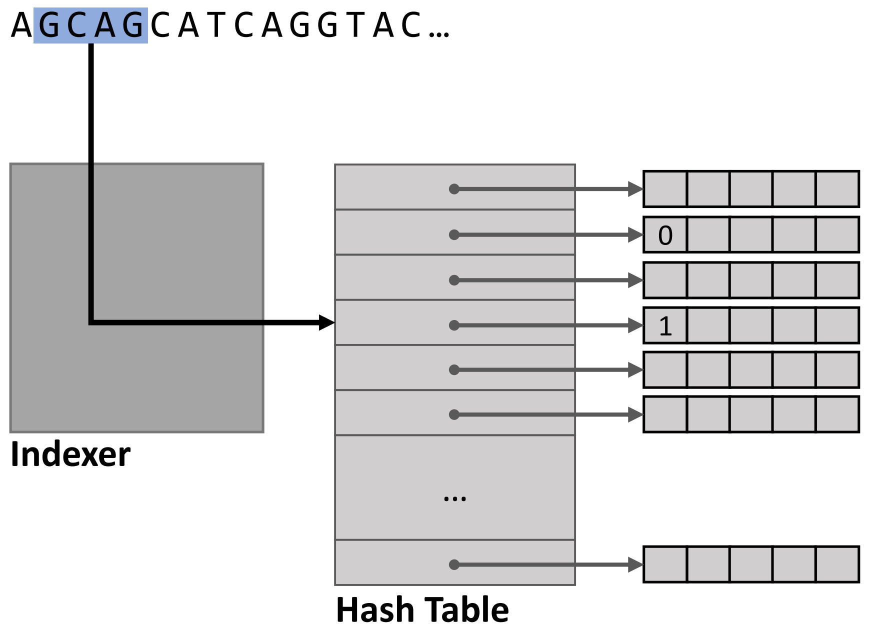
A G C A G C A T C A G G T A C …



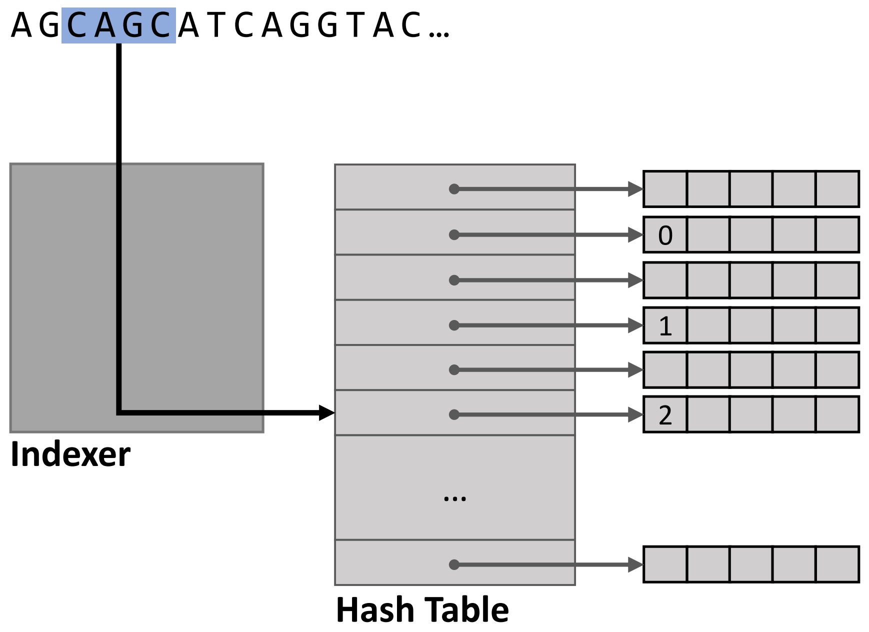
Genome Preprocessing



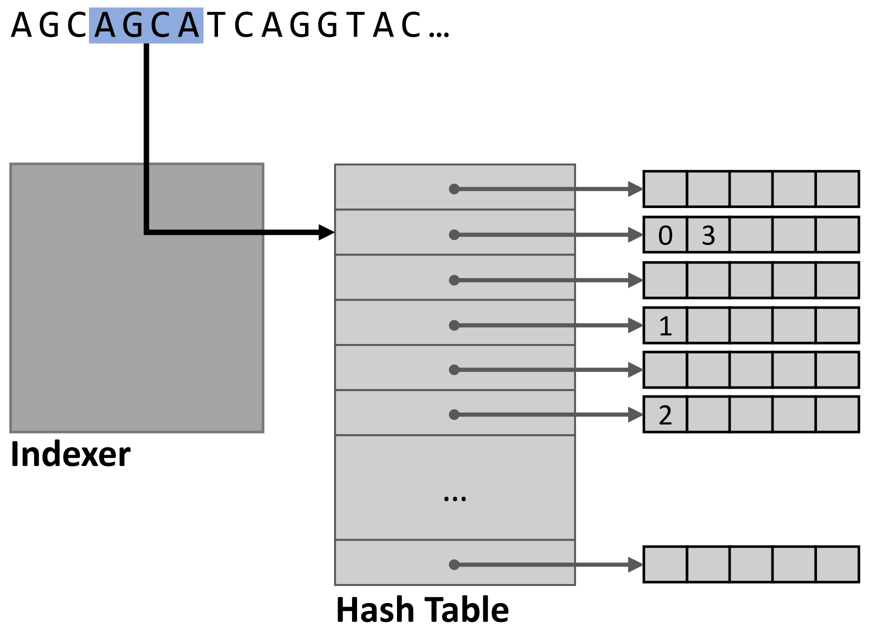
Genome Preprocessing



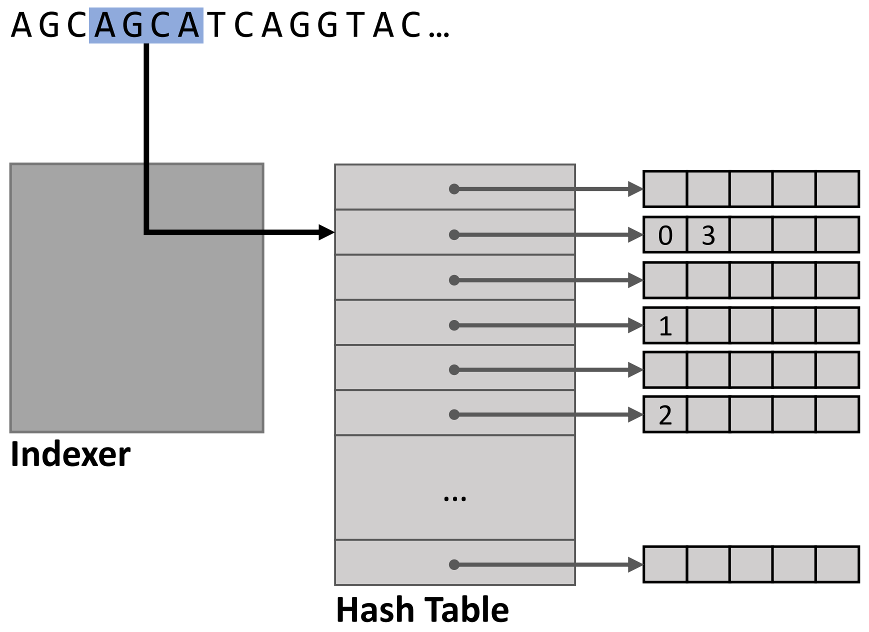
Genome Preprocessing



Genome Preprocessing



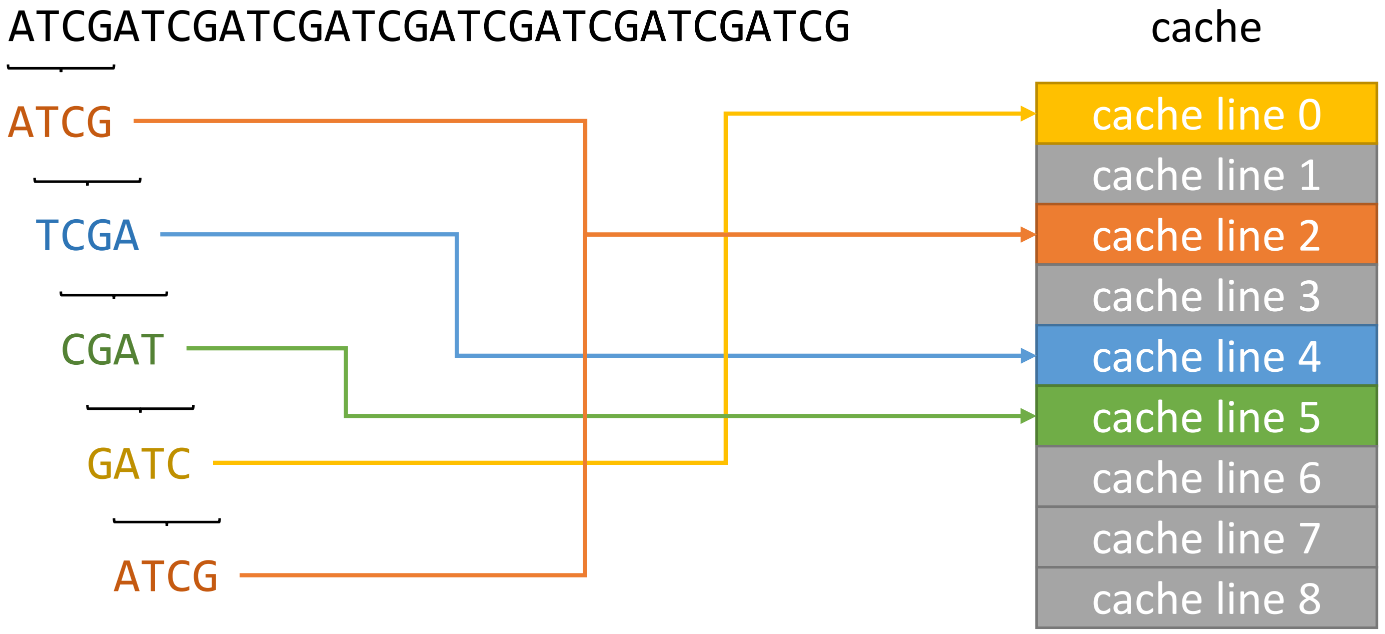
Genome Preprocessing



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

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

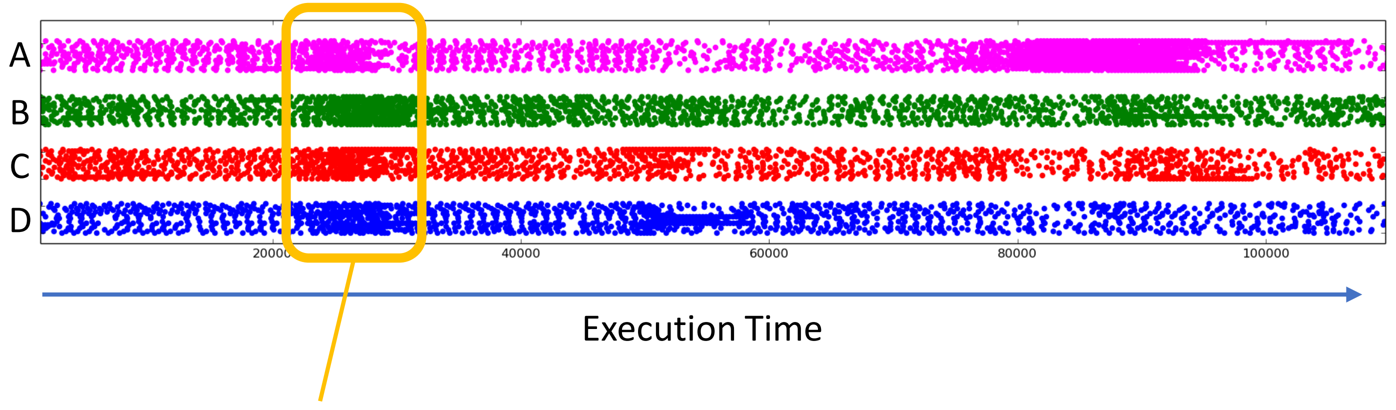
Microsatellites and Processed k-mers



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



Activity in all related cache lines

SGX Specific Side-Channel Defenses

- Page-fault side-channel defenses
  - 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]
    - Detect interrupts as indicator for an attack
    - Rely on Intel TSX, not available on all SGX-enable processors

- Cache side-channel defense
  - Cloak: Prime cache before accessing sensitive data [Schuster et al., USNIX 2017]
    - Requires annotations of sensitive data
    - Relies on Intel TSX, not available on all SGX-enabled processors

TSX: Transactional Synchronization Extensions

Hardware-based Side-Channel Defenses

- Time-interleaved cache sharing, flush on each context switch
  - Ineffective on SMT-enabled systems where caches are shared contemporaneously
  - E.g., [Costan et al., USENIX Sec’16]
- Cache partitioning / coloring
  - Reduces the amount of cache available to individual software
  - E.g., [Domnister et al., TACO’12]
- Randomized cache mappings
  - Frequency analysis or predictable access patterns can reveal randomization secret
  - E.g., [Wang et al., ISCA’07]

SMT: Simultaneous Multithreading

General Software-only Side-Channel Defenses

- Side-channel resilient software design
  - Not applicable to all applications
  - Manual hardening of software required

- Monitoring for attack effects
  - Requires privileged entity (not available in SGX model)

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

Summary: SGX – All Problems Solved?

- Side channels more drastic than originally thought
- Current add-on defenses not practical or effective
- Academic research provides many solutions that are 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-enabled CPUs

Thank You!
Questions?