Module 10 – Memory Safety and TEE

(ECE 209) Secure and Advanced Computer Architecture

Nader Sehatbakhsh
Department of Electrical and Computer Engineering
University of California, Los Angeles

How do we enforce security in computers today?

How do we enforce security in computers today?

1- "Principle of Least Privilege"

Users/Process should only have access to the data and resources needed to perform routine, authorized tasks.

How do we enforce security in computers today?

1- “Principle of Least Privilege”

Users/Process should only have access to the data and resources needed to perform routine, authorized tasks.

➔ *This leads to “privilege separation”. Typically, “user” and “root” level access.*

How do we enforce security in computers today?

2- Isolation

A process cannot access (read or write) the memory content of any other process.

How do we enforce security in computers today?

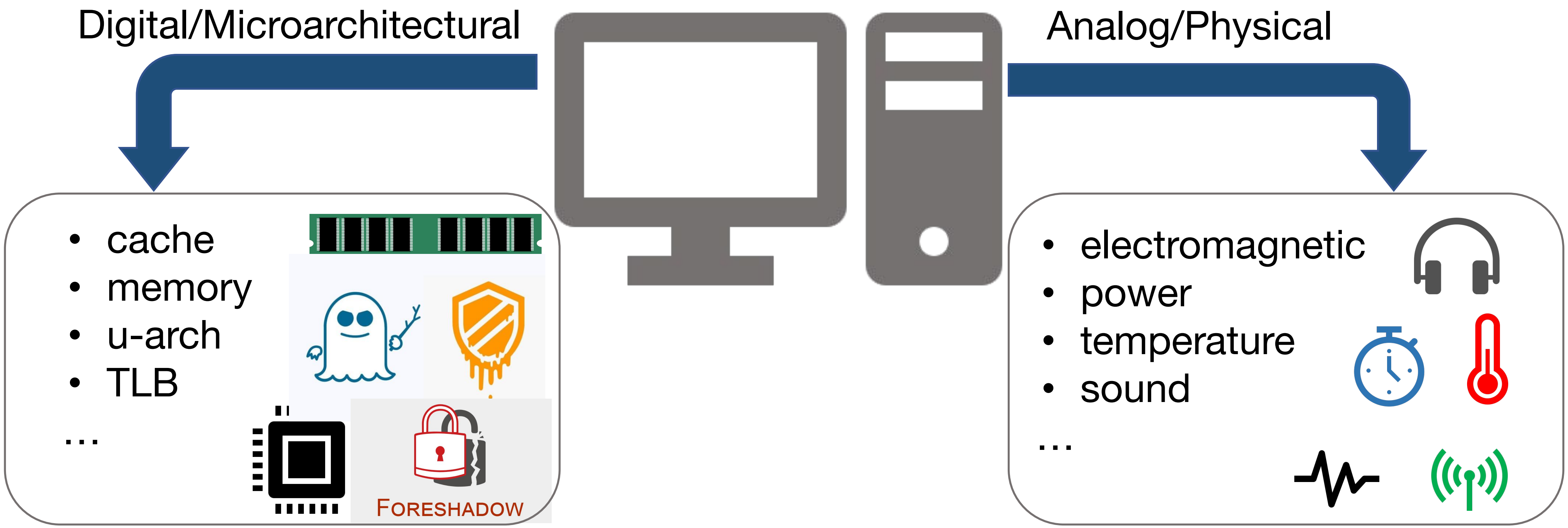
3- Trusted Computing Base (TCB)

Trust something (e.g., hardware), build everything on top/around that.

- *Only need to verify the TCB.*
- *Keep it simple and small so it can be easily(!) verified.*

Does it work?

Types of Side-Channels



Is there more?

- Side-channel attacks can read/leak information but can we WRITE?

Memory Safety

- Changing contents stored in memory to gain advantage!

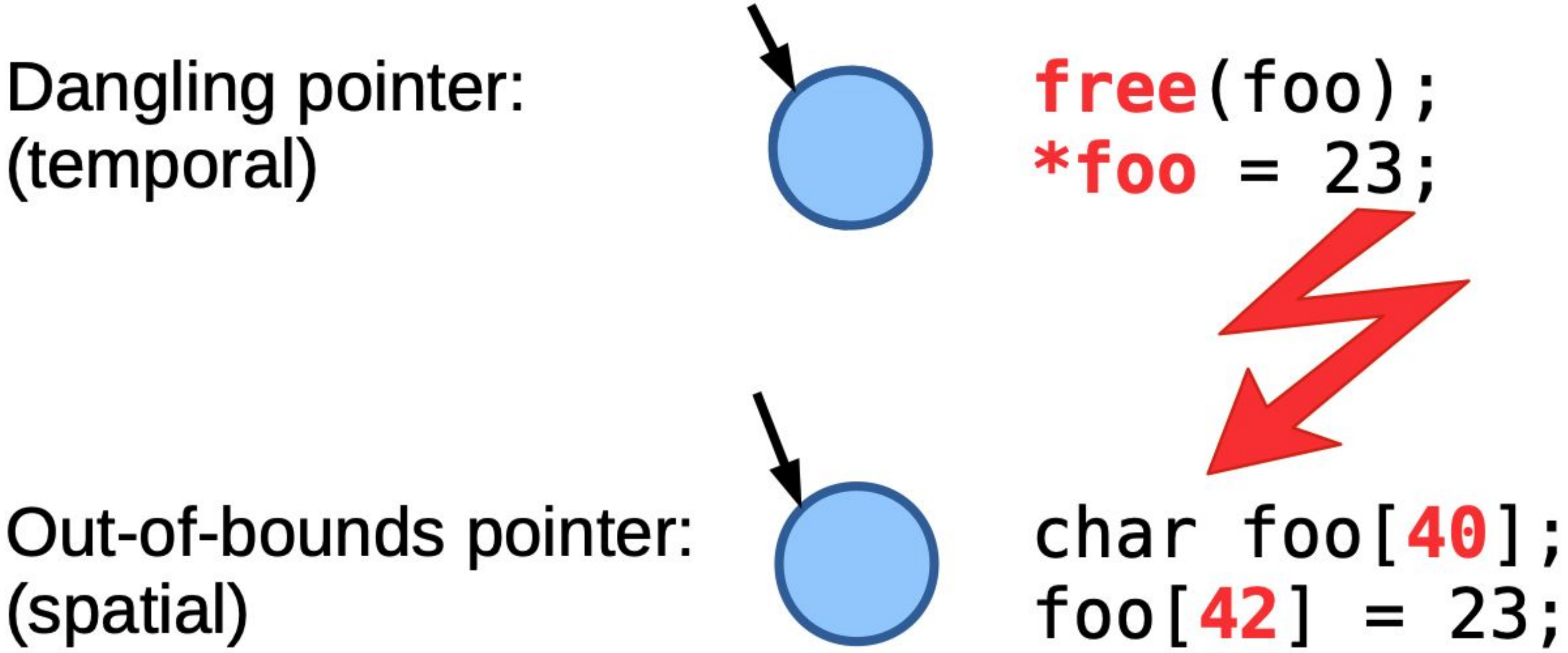
- We have already seen an example:
  - physical attacks → rowhammer, (other physical/fault attacks)
  - but can we use other means?

What can we do if we can modify memory?

Memory Safety

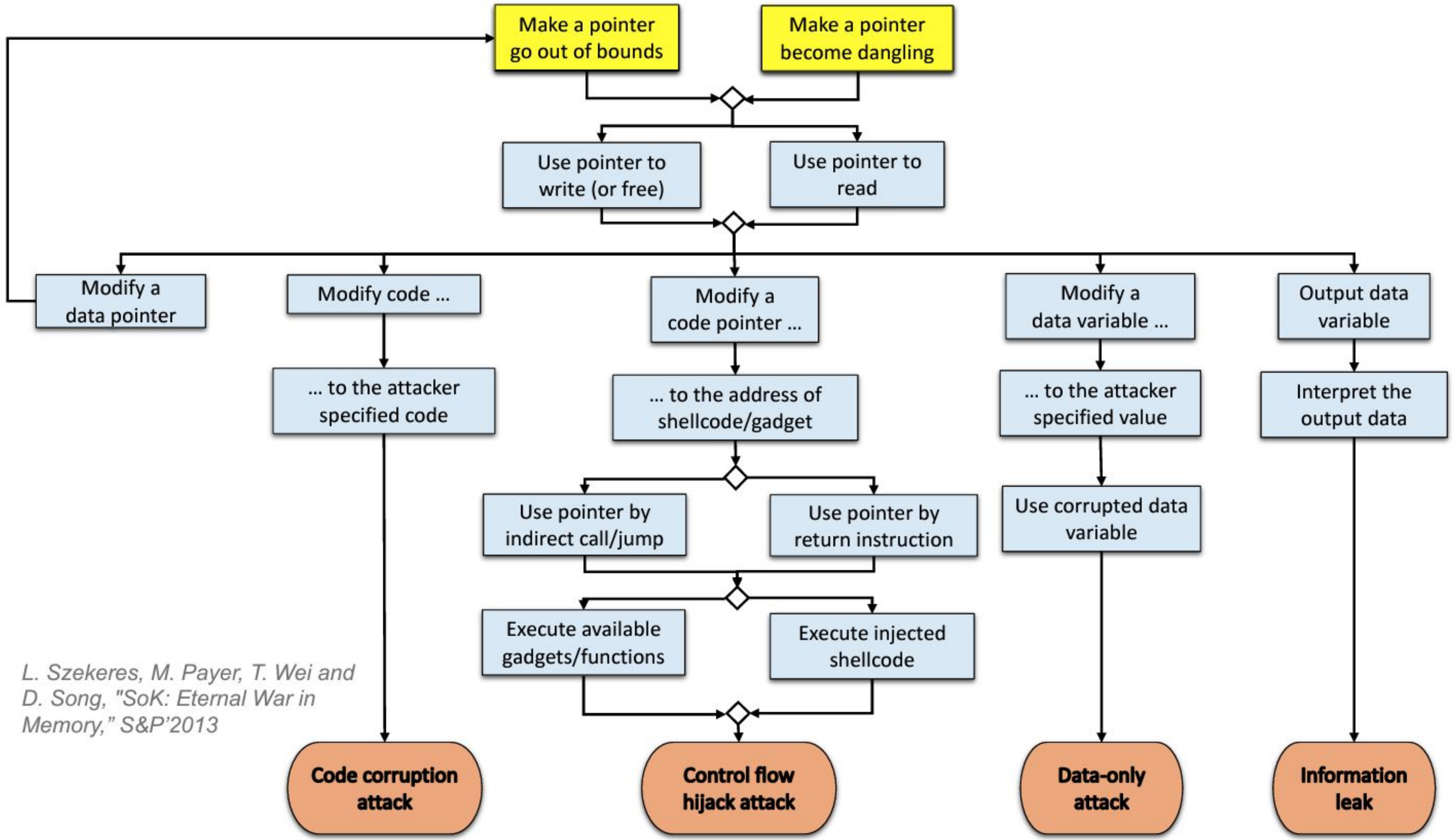
- How to modify memory? Remember that we need to somehow bypass isolation!

Invalid Dereferencing



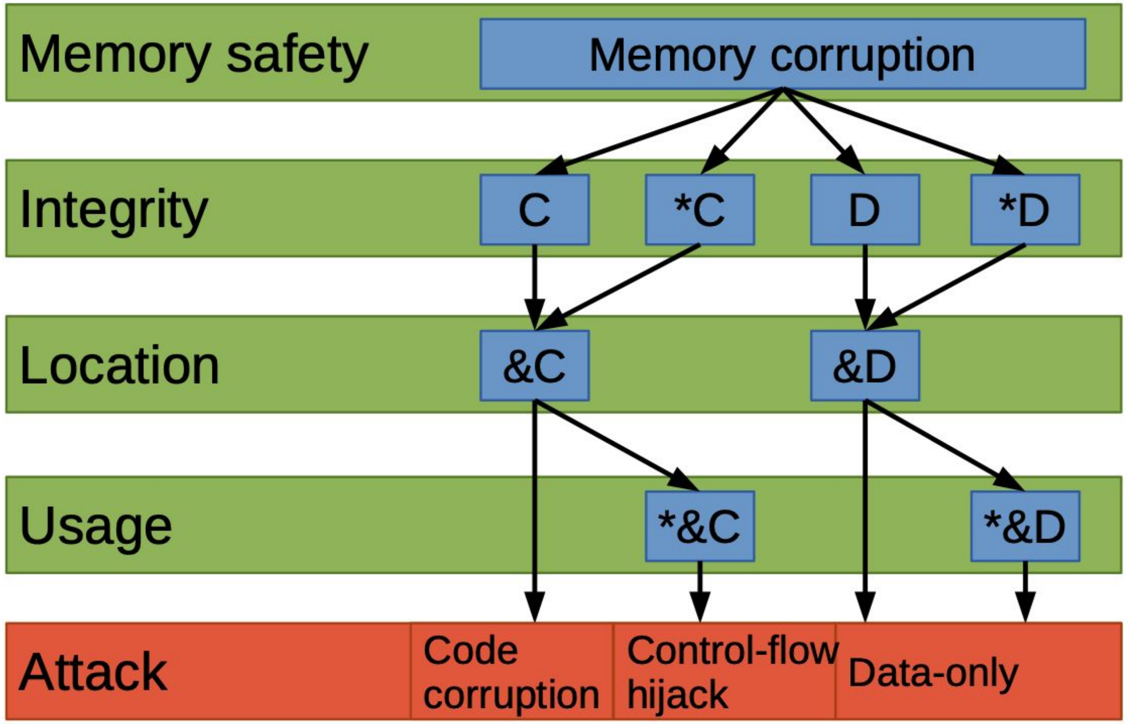
Violation iff: pointer is read, written, or freed

from SoK: Eternal War in Memory, Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song In: Oakland '14



L. Szekeres, M. Payer, T. Wei and D. Song, "SoK: Eternal War in Memory," S&P'2013

What can happen?



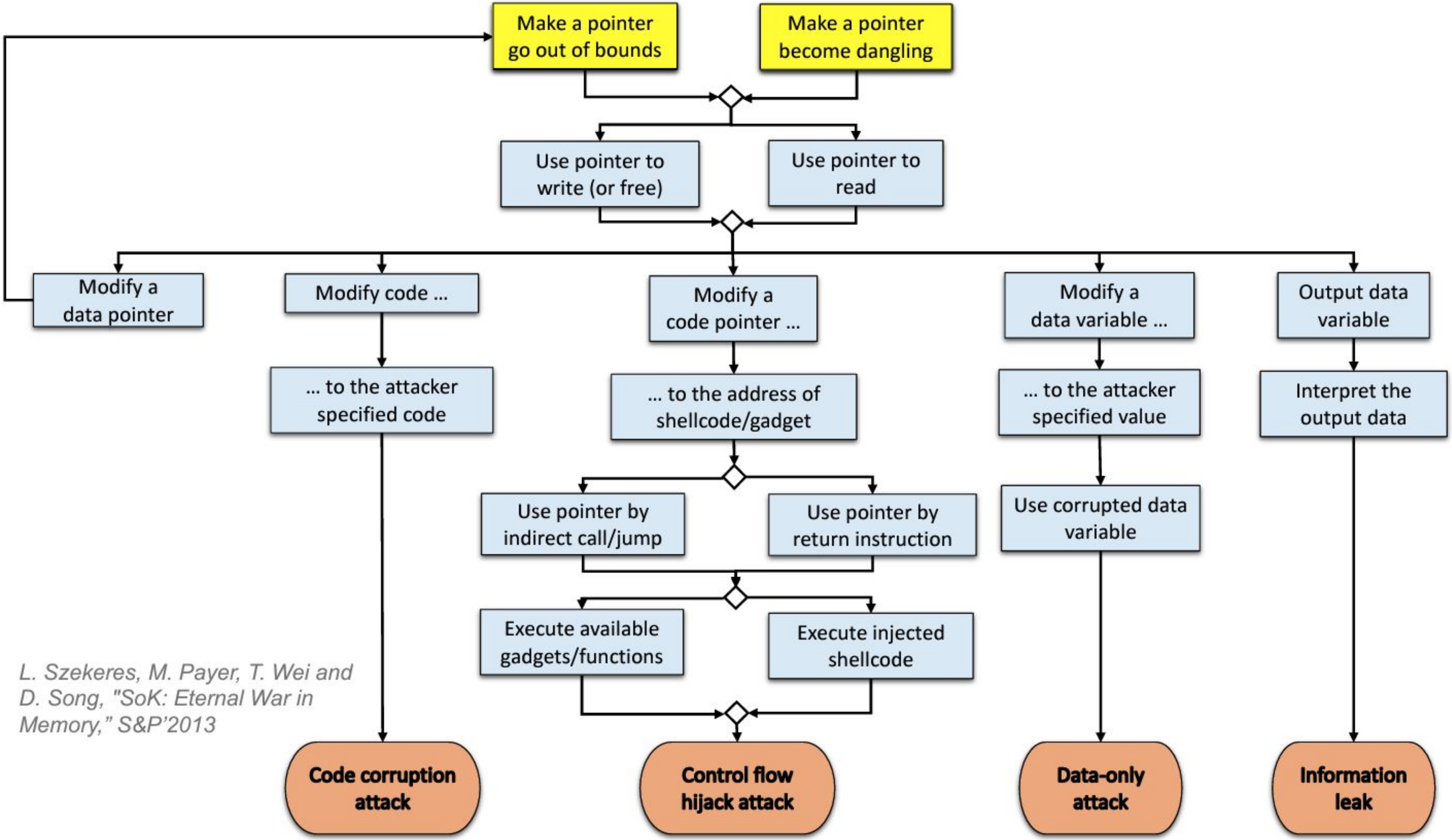
from SoK: Eternal War in Memory, Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song In: Oakland '14

How to defend?

- Range of solutions
  - Finding and fixing the bug
  - Software solutions
  - Hardware solutions

Hardware Support for Memory Safety

- Idea: include metadata and perform security checks at runtime
  - Spatial safety (bound information)
  - Temporal safety (allocation/de-allocation information)



L. Szekeres, M. Payer, T. Wei and D. Song, "SoK: Eternal War in Memory," S&P'2013

Main Ideas

- Control-flow integrity (CFI) → check all transitions and allow only those that are valid.

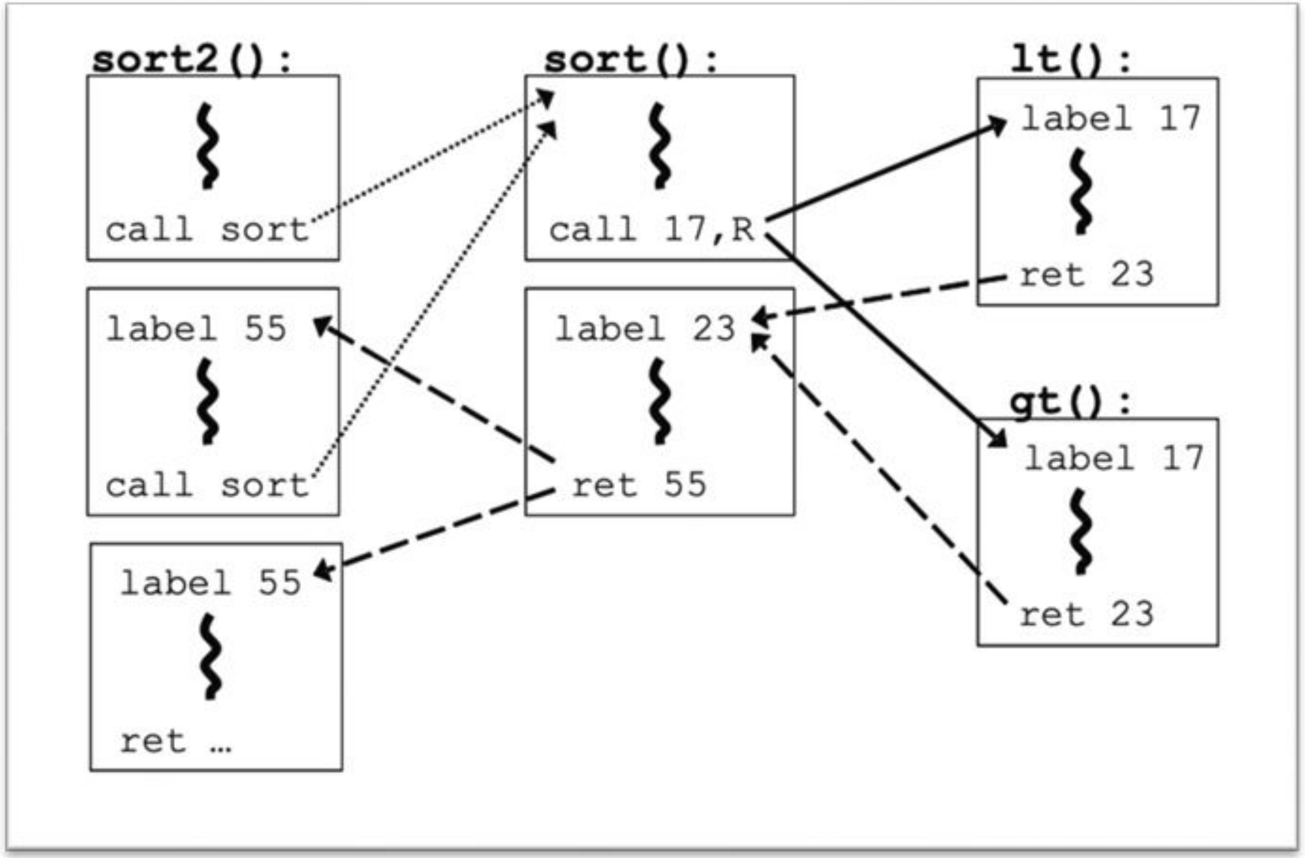
- Two main questions:
    - How to find/know all the valid transitions? (why)
    - How to check and enforce this in runtime?

```
bool lt(int x, int y) {
    return x < y;
}

bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len)
{
    sort( a, len, lt );
    sort( b, len, gt );
}

sort(int x[], int len, fun_ptr)
{
    for(int i=0; ….)
        for (int j=i; ….)
            if (fun_ptr(x[i], x[j]))
                … //swap x[i] and x[j]
```



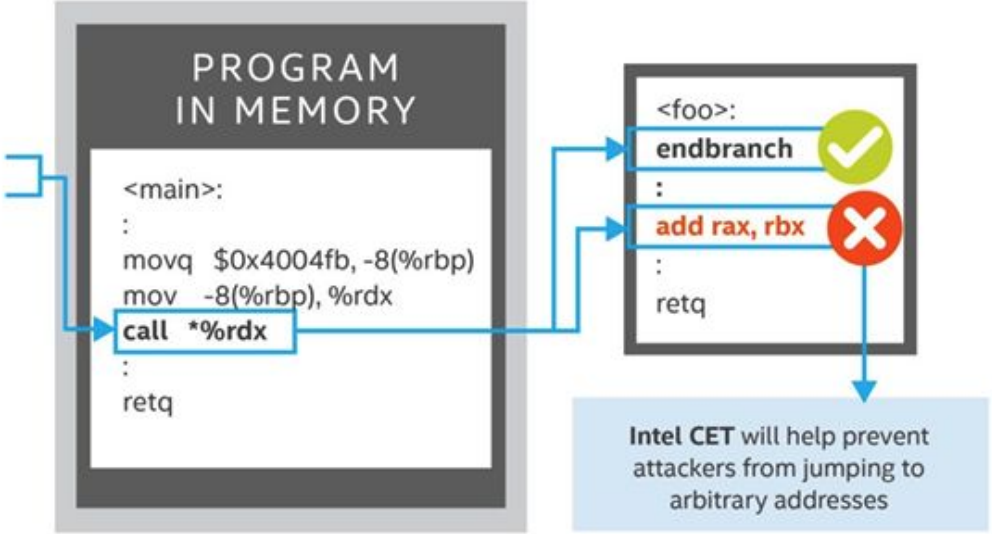
Control-Flow Integrity Principles, Implementations, and Applications; Mart´ın Abadi, et al. CCS'05

Intel® Control-Flow Enforcement Technology (Intel CET)

INTEL CET = **INDIRECT BRANCH TRACKING (IBT)** + **SHADOW STACK (SS)**

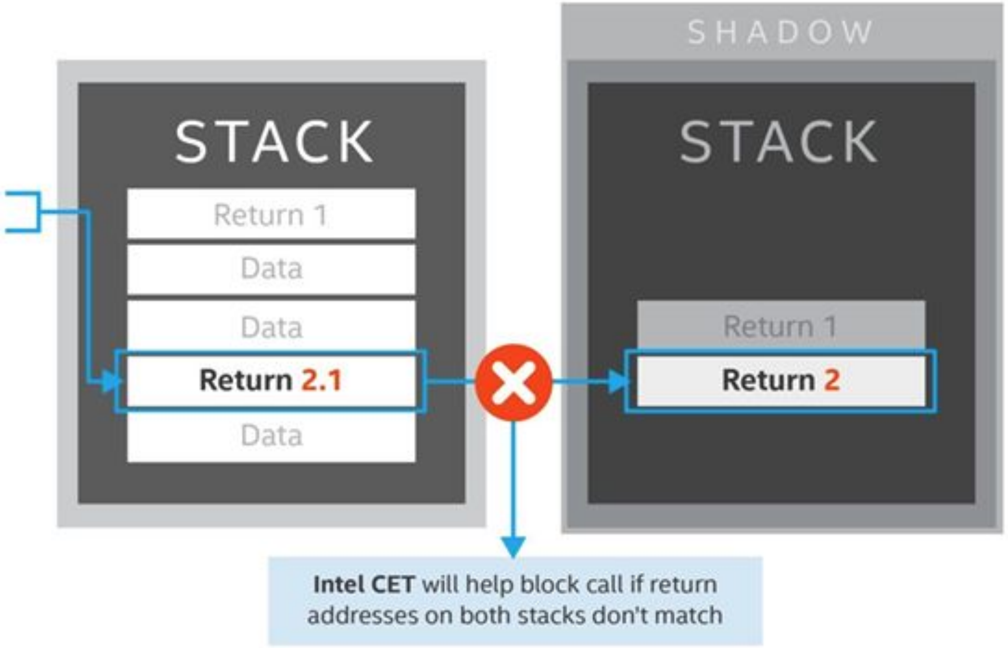
INDIRECT BRANCH TRACKING (IBT)

IBT delivers indirect branch protection to defend against jump/call oriented programming (JOP/COP) attack methods.



SHADOW STACK (SS)

SS delivers return address protection to defend against return-oriented programming (ROP) attack methods.



Shadow Stack Operation

SHADOW STACK

- Setup by OS/VMM
- Protected by new memory access control
- Different shadow stacks for each privilege level

Stack usage on near CALL

- Call
  - pushes return address on both stacks
- No parameters passing on shadow stack

- Return
  - pops return address from both stacks
  - Controlflow Protection (#CP) exception in case the two return addresses don't match

Keeps stack ABI intact – no changes to data stack layout

ENDBRANCH

- New Instruction to mark legal targets of indirect jumps
- Added by the compiler
- Decodes as “NOP” on legacy processors
- An indirect jump to a target not marked by ENDBR signals an exception

Recompile

```
main() {
        int (*f)();
        f = foo;
        f();
}
int foo() {
        return
}
```

```
<main>:
endbr64
:
movq    $0x4004fb, -8(%rbp)
mov     -8(%rbp), %rdx
call    *%rdx
:
retq

<foo>:
endbr64
:
add rax, rbx
:
retq
```

Main Ideas

- Control-flow integrity (CFI) → check all transitions and allow only those that are valid.

- Two main questions:
    - How to find/know all the valid transitions? (why)
    - How to check and enforce this in runtime?

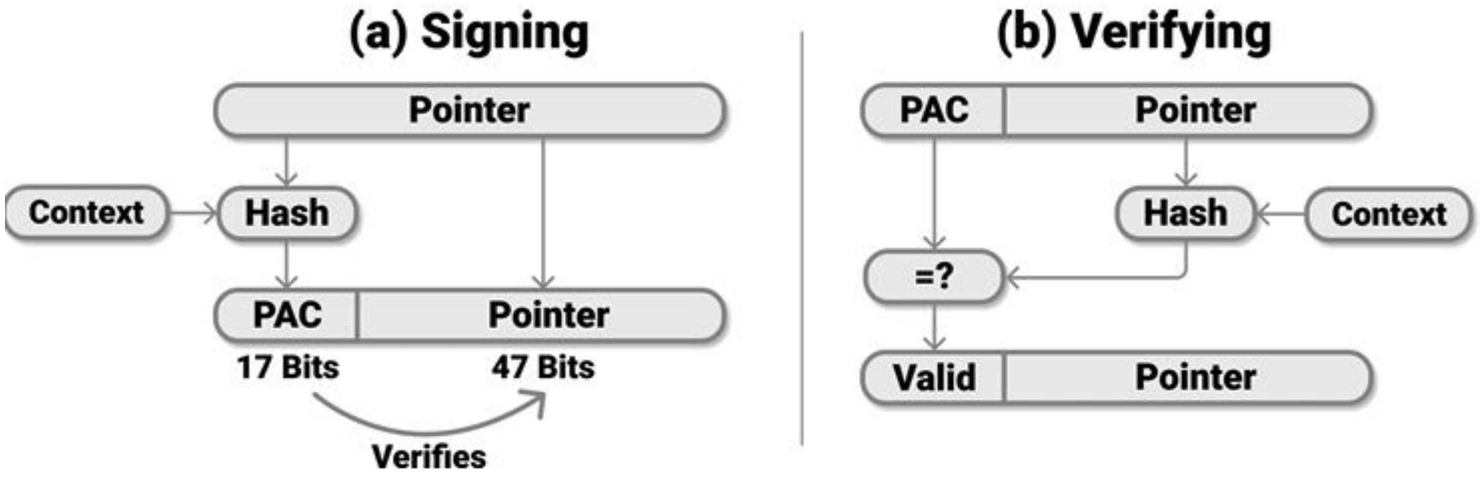
Main Ideas

- Memory tagging → Augment every pointer with its tag. Only legal instructions can update the tag!

Main Ideas

- Memory tagging → Augment every pointer with its tag. Only legal instructions can update the tag!
- Further expand this to every memory content → data, pointer, and code

ARM PAC



Before function call

```
pacia lr, sp
sub   sp, sp, #0x40
str   lr, [sp, #0x30]
...
```

Before function return

```
ldr   lr, [sp, #0x30]
add   sp, sp, #0x40
autia lr, sp
ret
```

Challenges

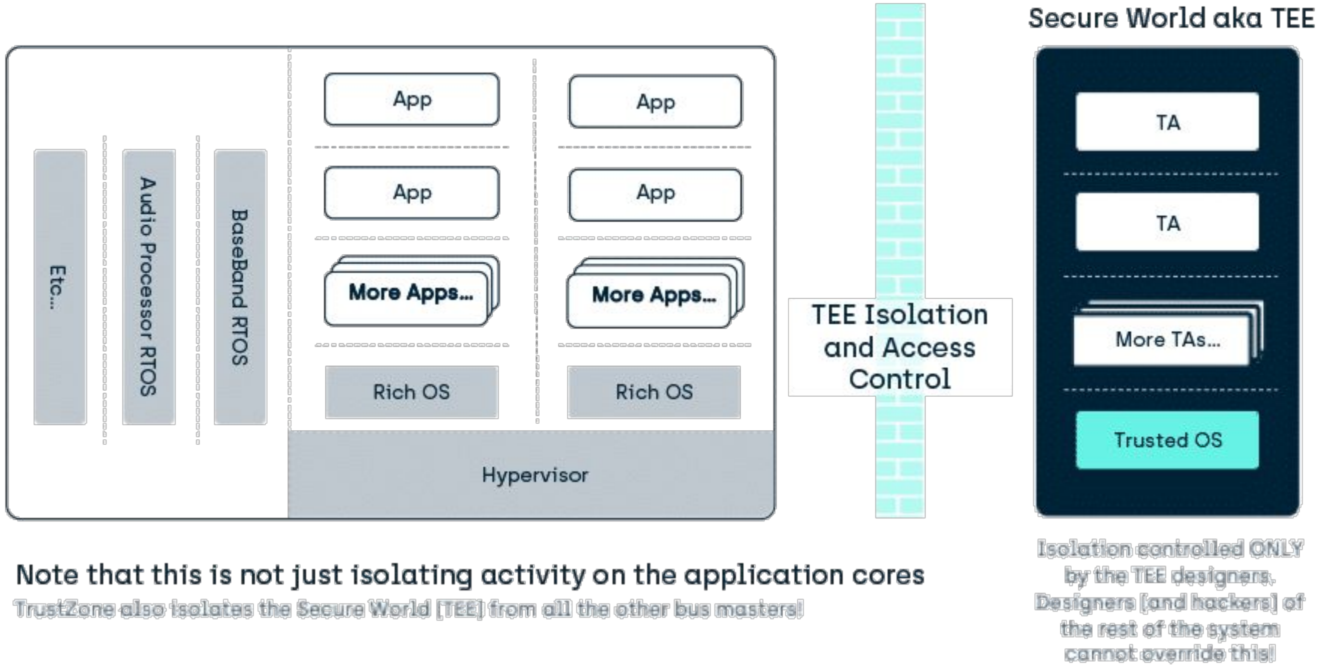
- How to store tags? How to move them? How much is the slowdown?
- What MAC to use? What to do if they don't match?
- Does this always work?

Trusted Execution Environment

TEE

- Why do we need them?

  1. Taking isolation to next level



From https://www.trustonic.com/technical-articles/what-is-a-trusted-execution-environment-tee/

Why we need more isolation?

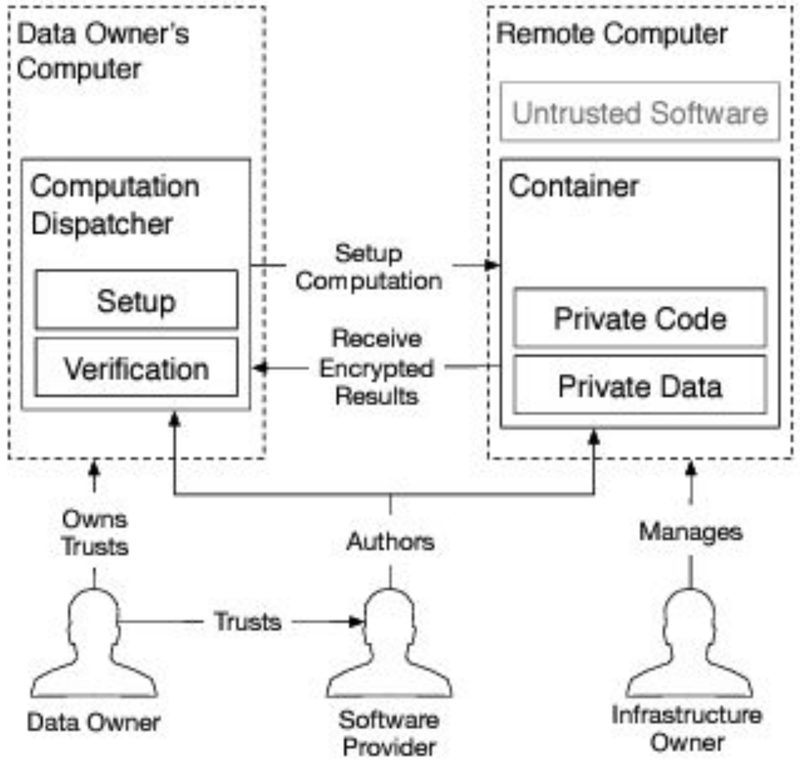
- Larger the code, harder to remove bugs.
- Side channels.
- Different vendors/designers which we don't trust.
- More sharing.

TEE

- Why do we need them?

1. Taking isolation to next level

2. Remote computing



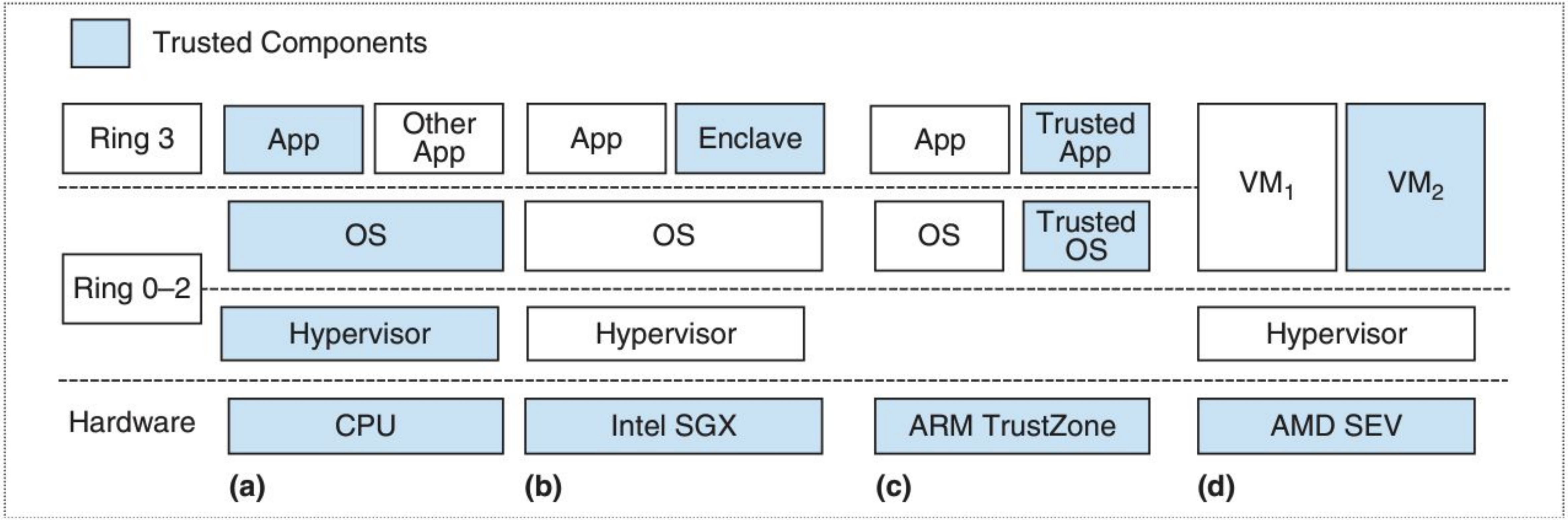
From SGX Explained.

Why we need remote computing?

- We won't trust software but hardware and cryptographic methods.

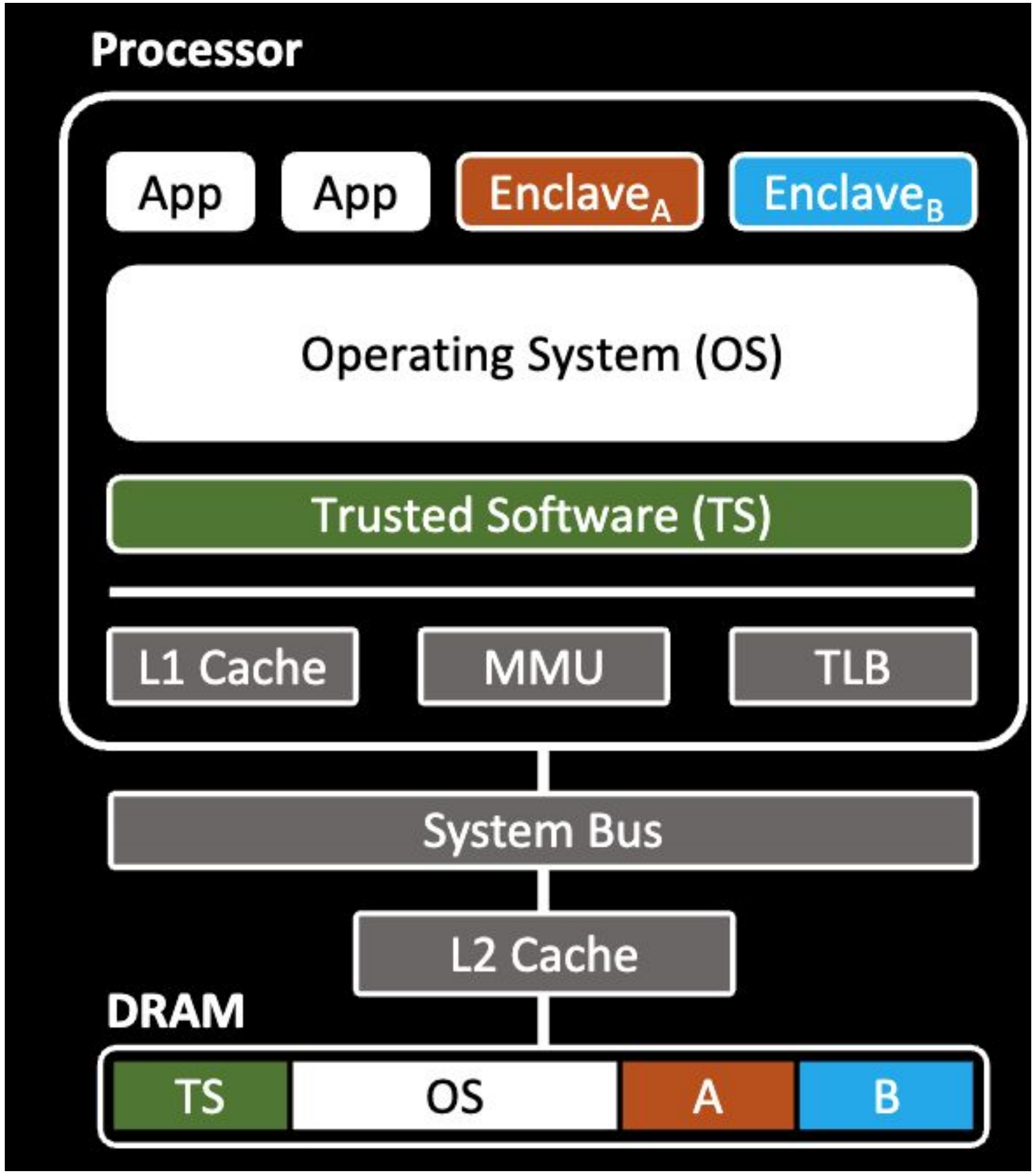
- The trust issue is either due to bugs or just having a third-party and/or regulations.

TEE Trust Boundaries



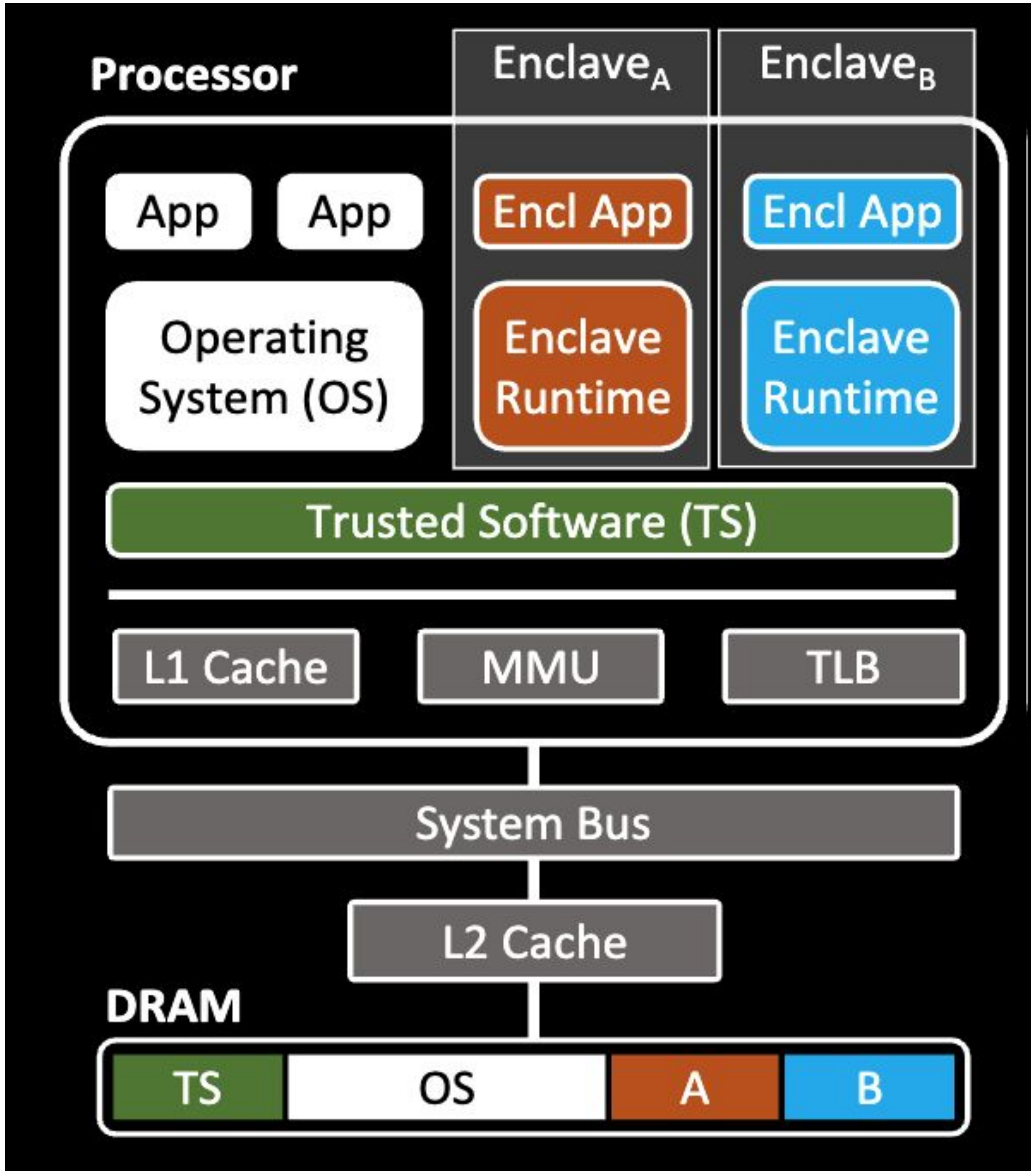
From Building Open Trusted Execution Environments

Options



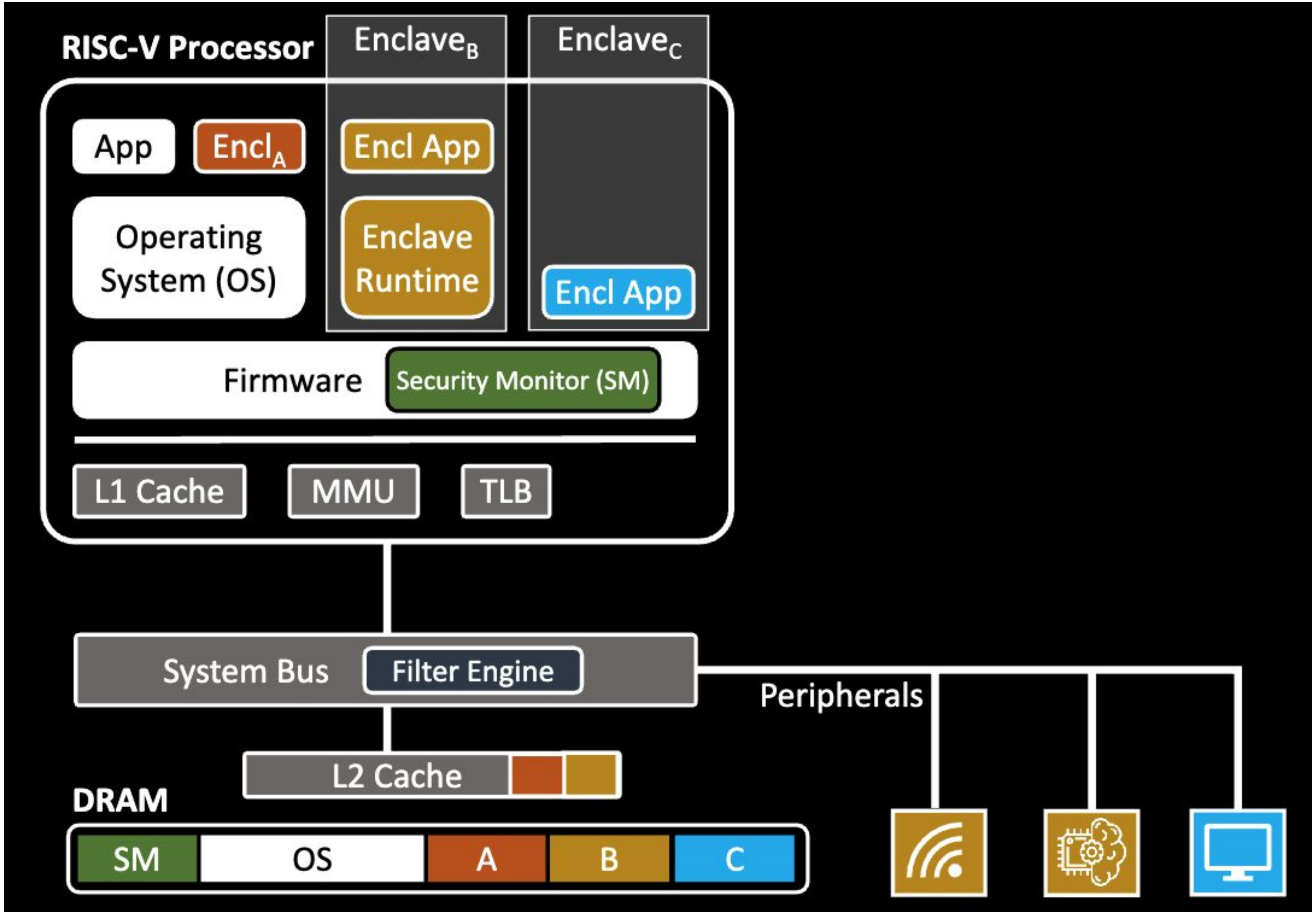
From CURE: A Security Architecture with Customizable and Resilient Enclaves

Options



From CURE: A Security Architecture with Customizable and Resilient Enclaves

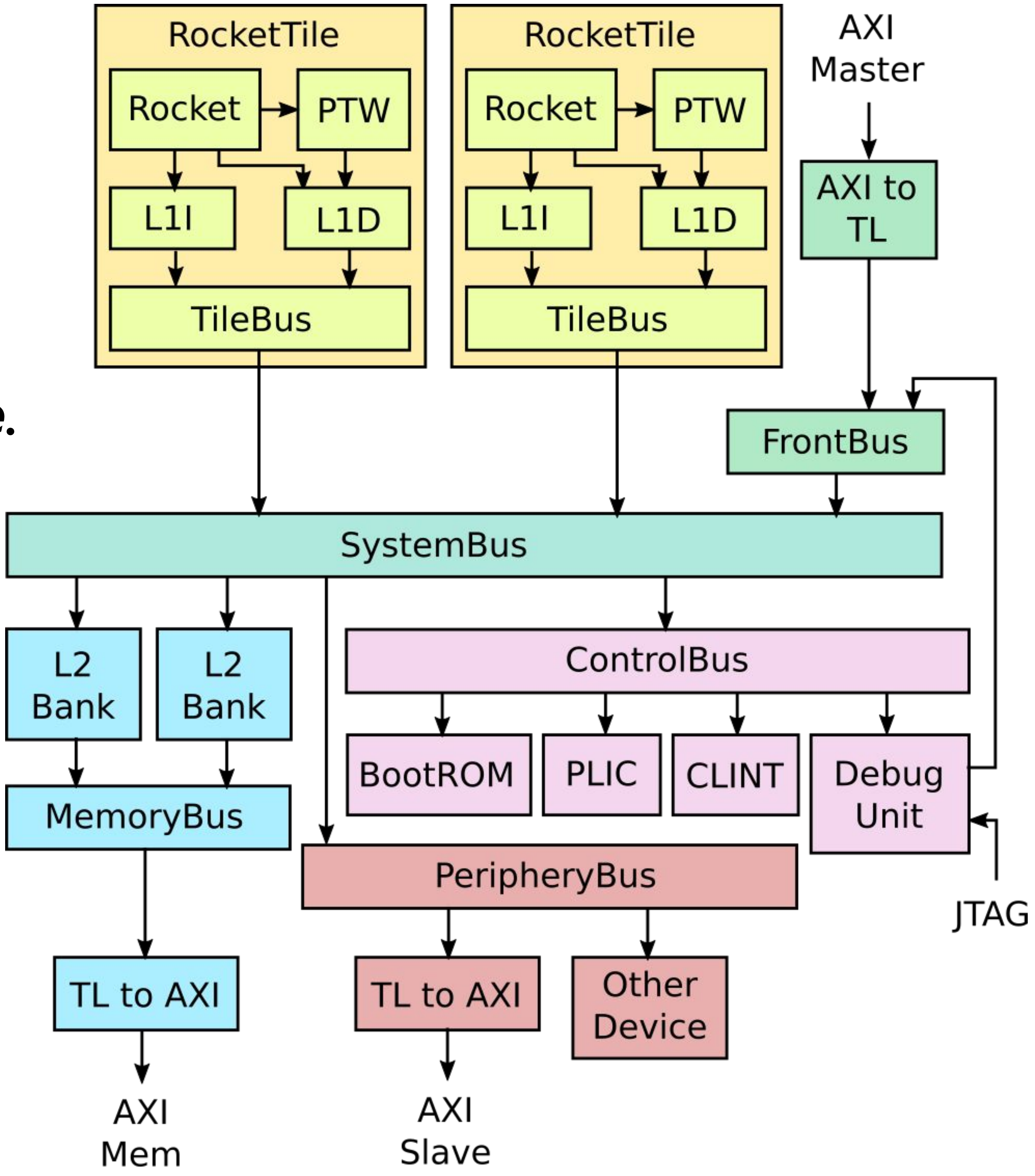
Options

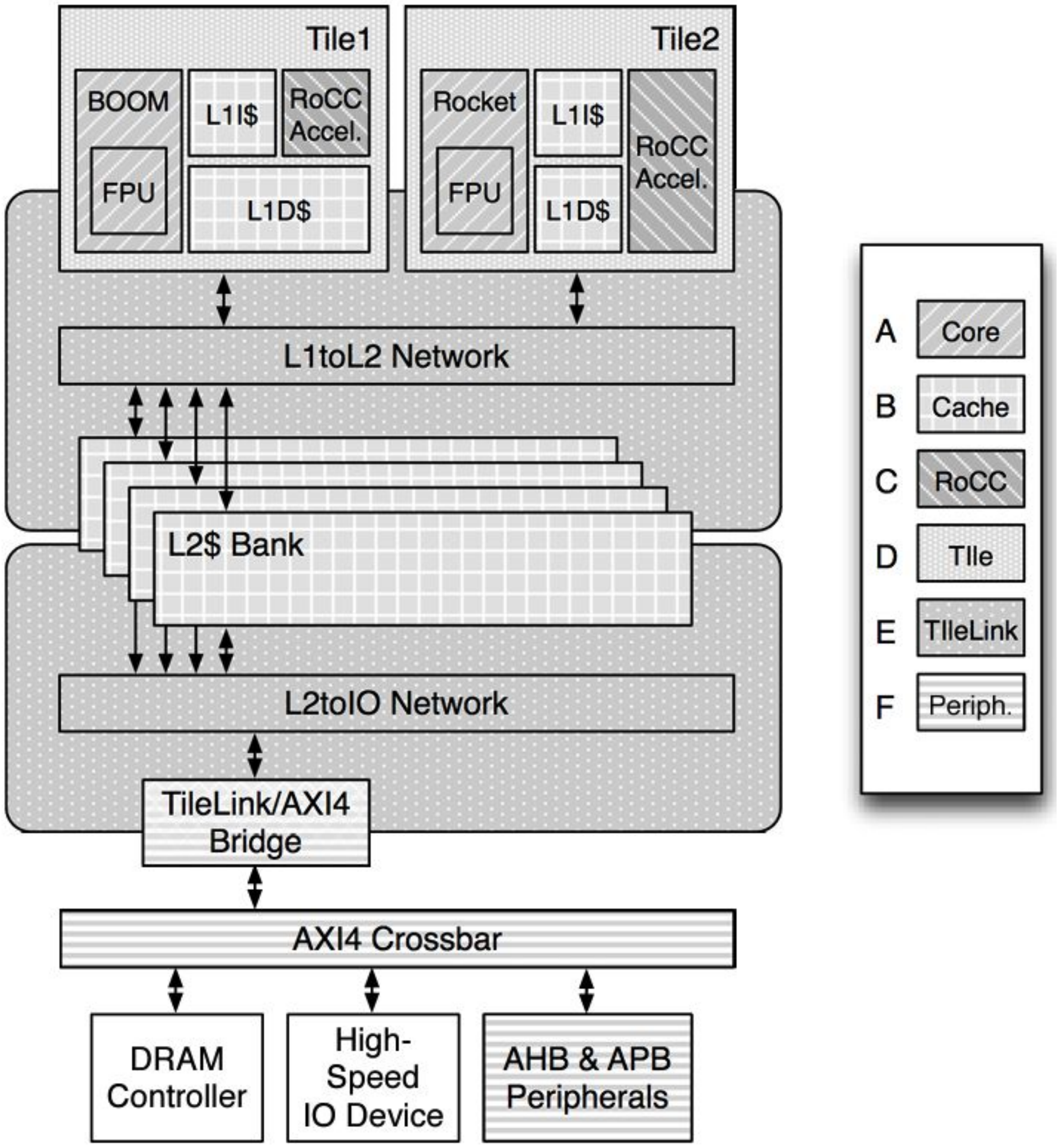


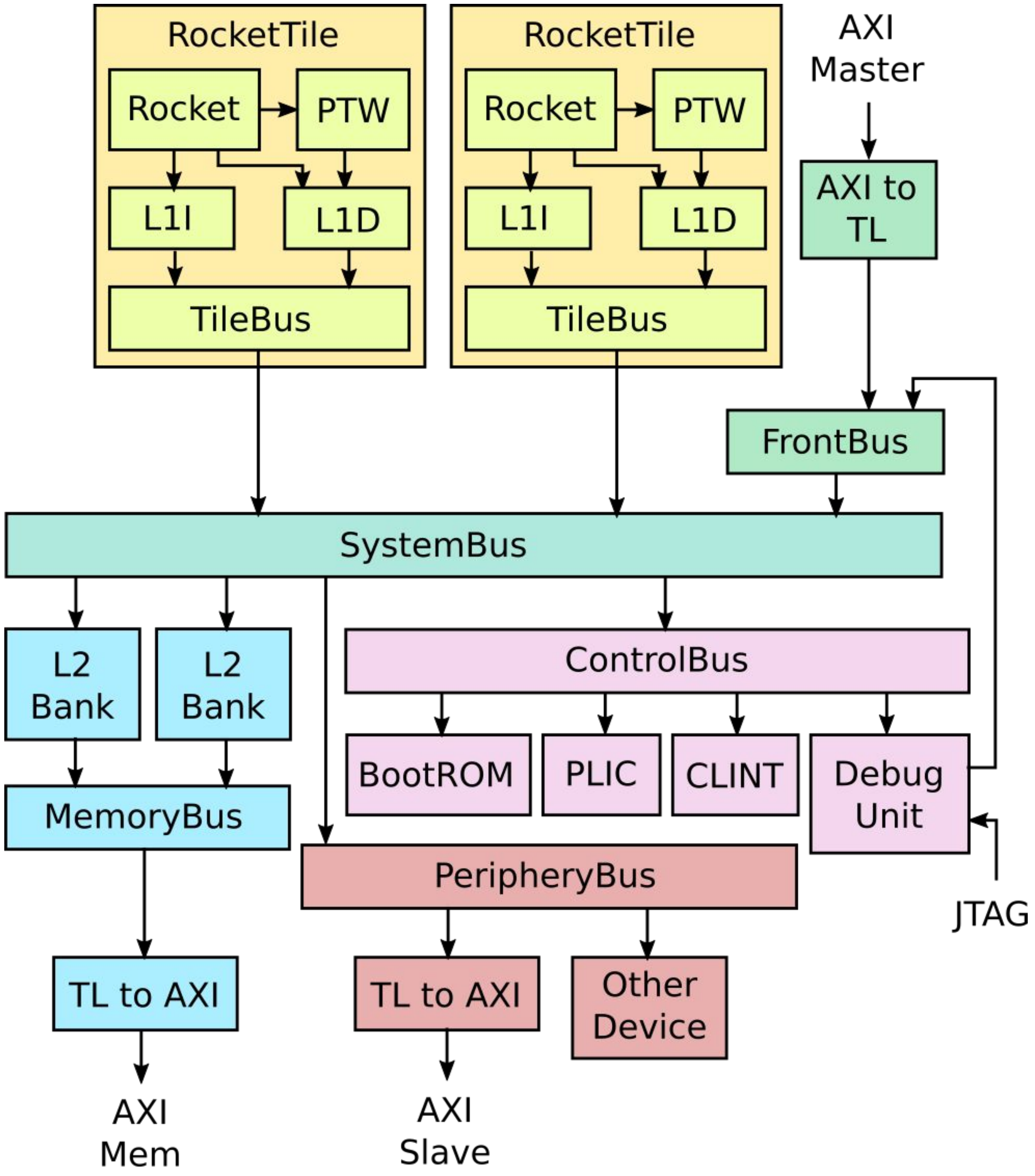
From CURE: A Security Architecture with Customizable and Resilient Enclaves

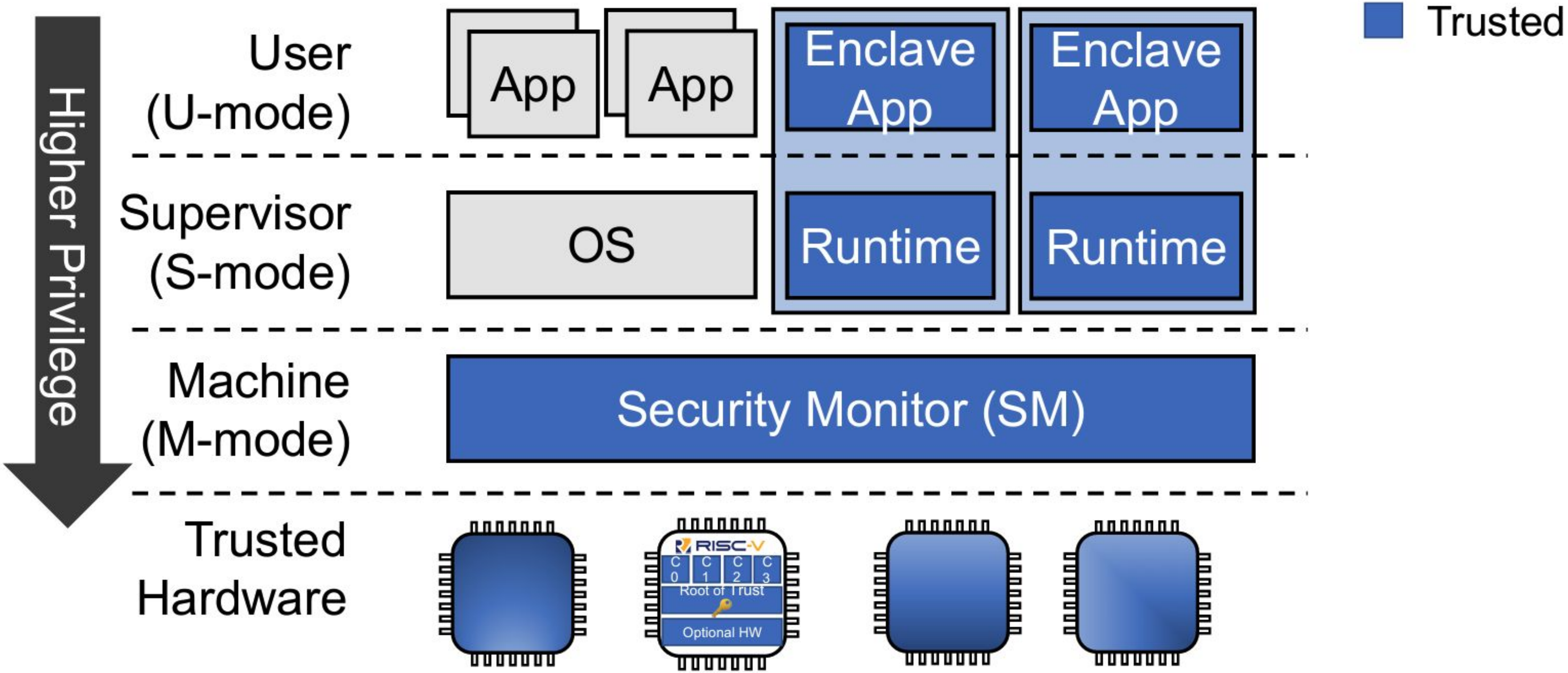
Microarch

- How to architect a TEE?
- We are focusing on ***Keystone***.

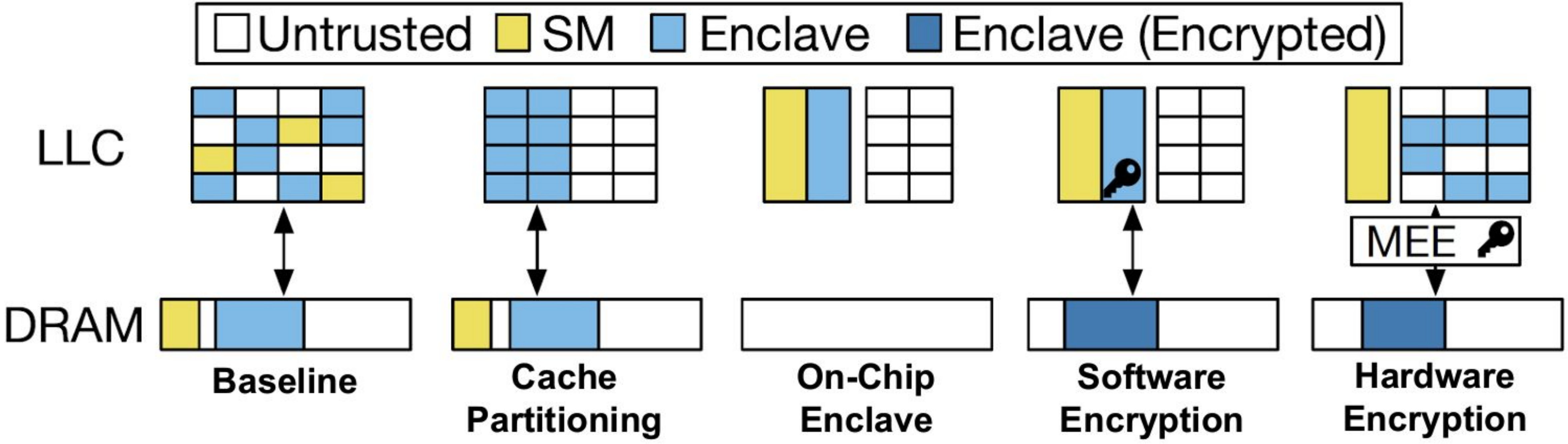



Tile1
BOOM
FPU
L1I$
RoCC Accel.
L1D$
Tile2
Rocket
FPU
L1I$
L1D$
RoCC Accel.
L1toL2 Network
L2$ Bank
L2toIO Network
TileLink/AXI4 Bridge
AXI4 Crossbar
DRAM Controller
High-Speed IO Device
AHB & APB Peripherals
A Core
B Cache
C RoCC
D TIle
E TIleLink
F Periph.


RocketTile
Rocket
PTW
L1I
L1D
TileBus
RocketTile
Rocket
PTW
L1I
L1D
TileBus
AXI Master
AXI to TL
FrontBus
SystemBus
L2 Bank
L2 Bank
MemoryBus
ControlBus
BootROM
PLIC
CLINT
Debug Unit
JTAG
PeripheryBus
TL to AXI
AXI Mem
TL to AXI
AXI Slave
Other Device


Higher Privilege
User
(U-mode)
App
App
Enclave
App
Enclave
App
Trusted
Supervisor
(S-mode)
OS
Runtime
Runtime
Machine
(M-mode)
Security Monitor (SM)
Trusted
Hardware
RISC-V
C 0
C 1
C 2
C 3
Root of Trust
Optional HW

How to protect memory?



What are the challenges?

- First, is this really secure?
  - Not really, there are still many side channel attacks because TEE is not truly isolated!

What are the challenges?

- First, is this really secure?
    - Not really, there are still many side channel attacks because TEE is not truly isolated!

- Second, how to balance performance and security?
    - Challenges: true isolation reduces utilization, parallelism, etc.

What are the challenges?

- First, is this really secure?
  - Not really, there are still many side channel attacks because TEE is not truly isolated!
- Second, how to balance performance and security?
  - Challenges: true isolation reduces utilization, parallelism, etc.
- Third, how to handle security for other components?

Summary

- In addition to hardware concerns, side channel, transient, and faults, computing systems are vulnerable to software attacks due to bugs in software.

- There are two main solutions for software vulnerabilities:
  - Either fix them using a plethora of solutions (hardware, software, etc.)
  - Or use TEEs to isolate the environment.
    - TEEs have additional bonuses including remote computing and/or protection against faults, side-channel, and/or transient attacks.

End of Presentation

UCLA Samueli School of Engineering

Acknowledgement

- This course is partly inspired by the following courses created by my colleagues:
  - 6.888 MIT (Yan)
  - CS 598 UIUC (Fletcher)
  - CS 7290 Georgia Tech (Kim)
  - ECE 752 Wisconsin (Lipasti)
  - ECE 7103A Georgia Tech (Qureshi)