Hardware-Assisted Security on Arm Mobile Platforms

– From Memory Safety to Confidential Computing

2025.11.27

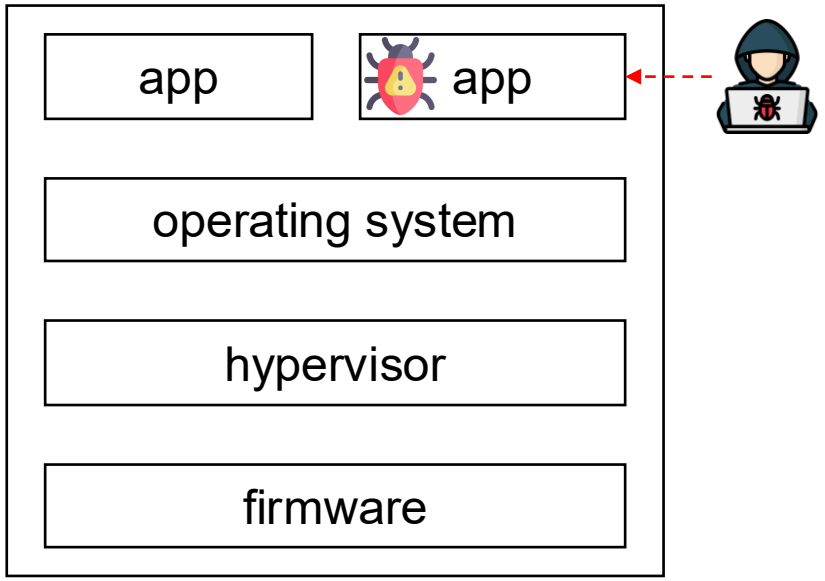
Junseung You

Who am I

- Education
  - 2014~2019 ECE B.E. @SNU
  - 2019~2026.2 (expected) Ph.D. candidate @SNU Security Optimization Research Lab
    Advisor: Prof. Yunheung Paek
- Research – System Security
  - Classic system security problems on ARM – memory safety, isolation, access control
  - Trusted execution environments
- Current Project
  - Secure on-device AI (using trusted execution environments)

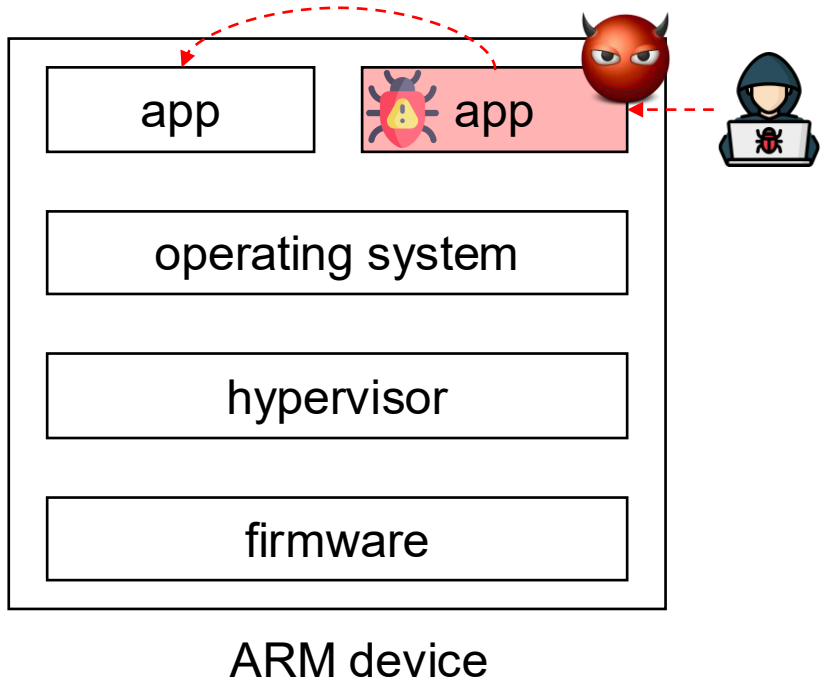
Systems are Vulnerable

- Bugs and vulnerabilities throughout the system software stack
  - Buffer overflows, use-after-frees, permission check bypass, privilege escalation, …



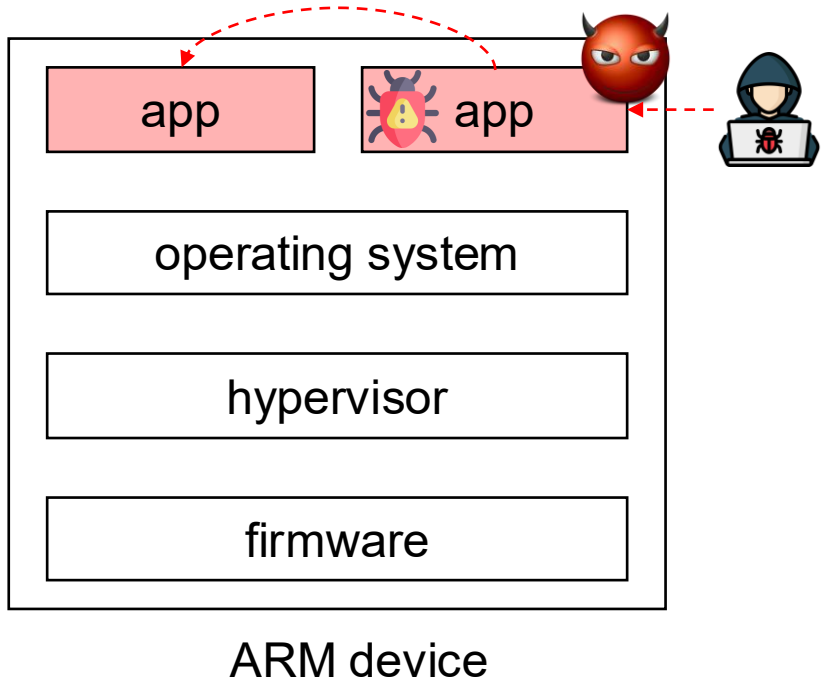
Systems are Vulnerable

- Bugs and vulnerabilities throughout the system software stack
  - Buffer overflows, use-after-frees, permission check bypass, privilege escalation, …



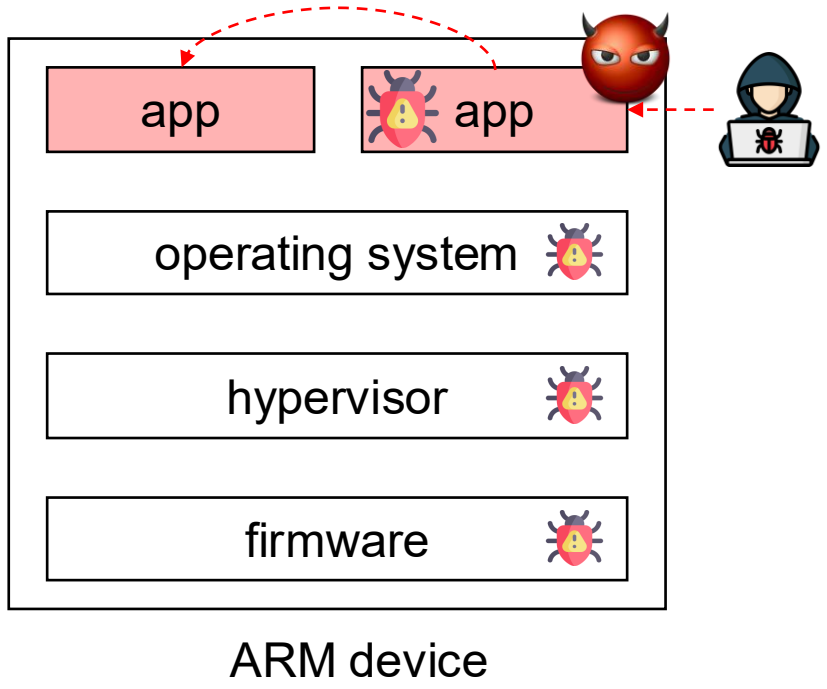
Systems are Vulnerable

- Bugs and vulnerabilities throughout the system software stack
  - Buffer overflows, use-after-frees, permission check bypass, privilege escalation, …



Systems are Vulnerable

- Bugs and vulnerabilities throughout the system software stack
  - Buffer overflows, use-after-frees, permission check bypass, privilege escalation, …



Systems are Vulnerable

- Bugs and vulnerabilities throughout the system software stack
  - Buffer overflows, use-after-frees, permission check bypass, privilege escalation, …

Computing > Internet > Online Security > Malware & Adware

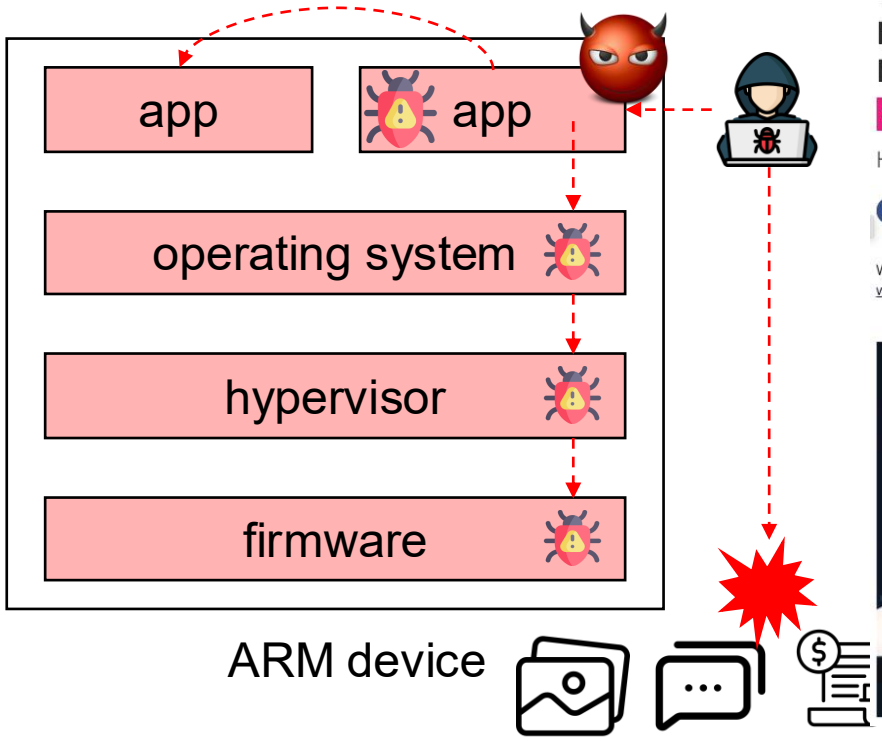
Samsung phones infected with 'Landfall' spyware through WhatsApp images — what you need to know

News By Amber Bouman published November 13, 2025

This zero-day attack doesn't require any user interaction

Comments (0)

When you purchase through links on our site, we may earn an affiliate commission. Here's how it works.



The 'Swiss army knife' of malware emerges - Hook v3 can do ransomware, keylogging, DDoS, screen capture, and far more

News By Efosa Udinmwen published September 5, 2025

Hook v3 adds 38 new commands, expanding its malicious reach

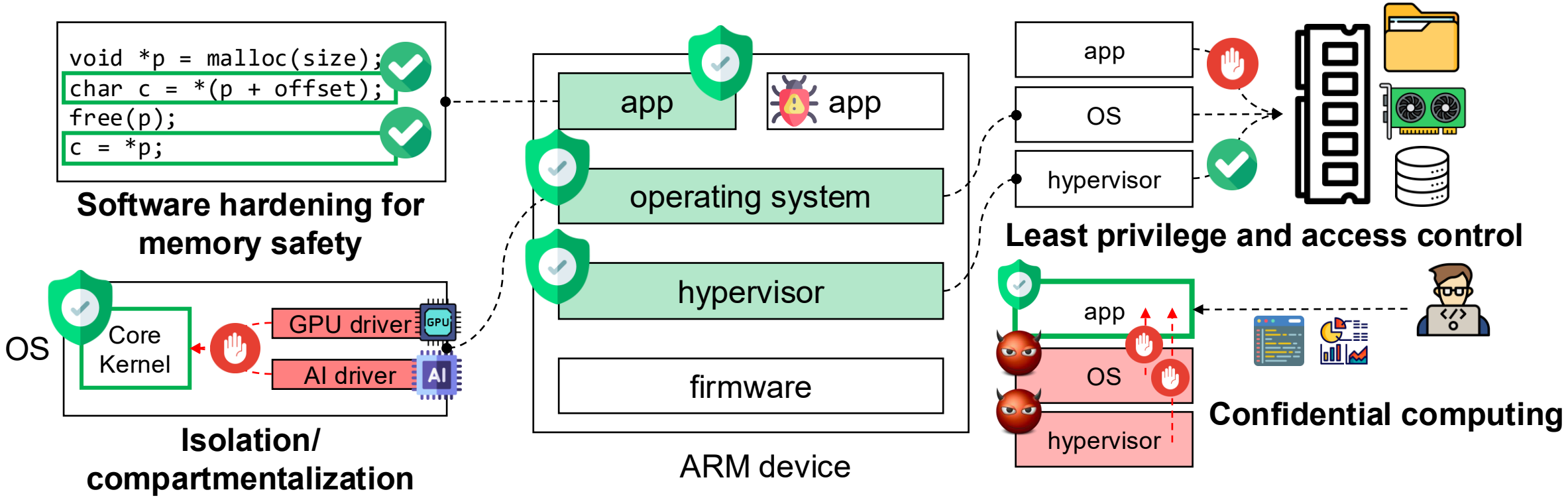
Comments (0)

When you purchase through links on our site, we may earn an affiliate commission. Here's how it works.

Building Secure Systems

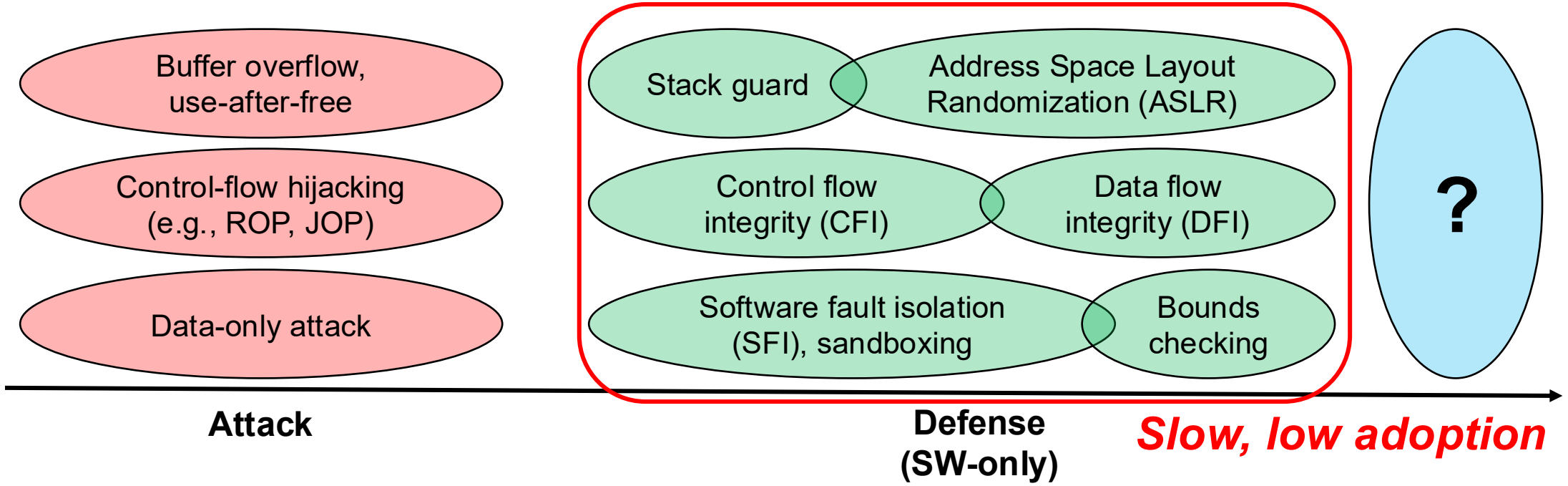
Confidentiality
Integrity
Availability

- Solutions to satisfy security principles/goals for secure systems
  - Isolation/compartmentalization, principle of least privilege, memory safety, …



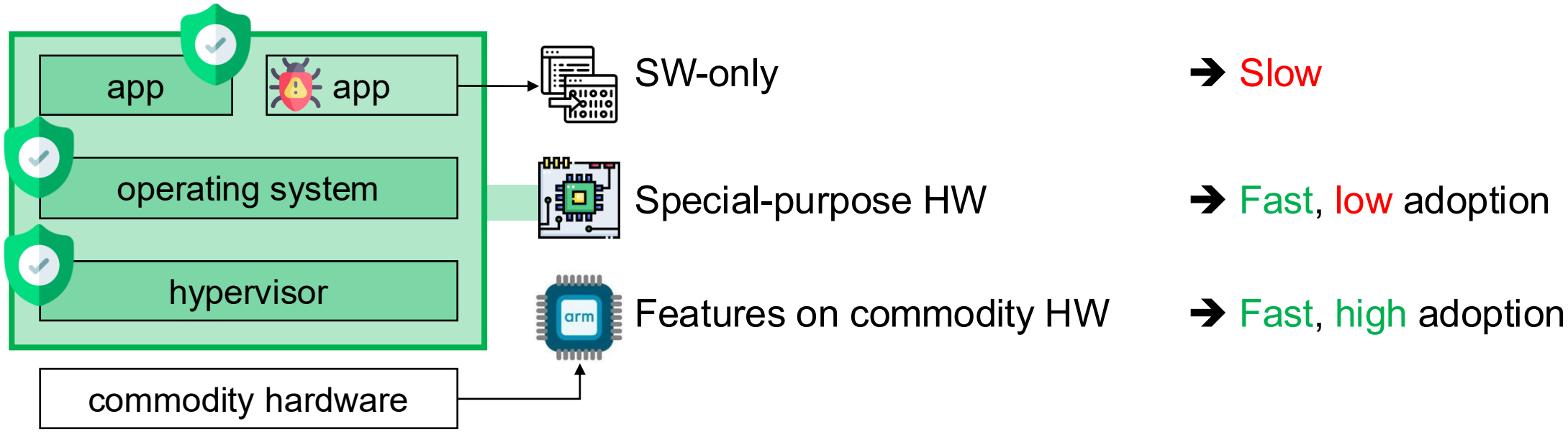
Security is not free

- Security and performance tradeoff
  - Achieving security requires additional measures (e.g., *checks*) to normal operations
  - Security mechanism can incur up to **2~3x** performance overhead (e.g., AddressSanitizer)



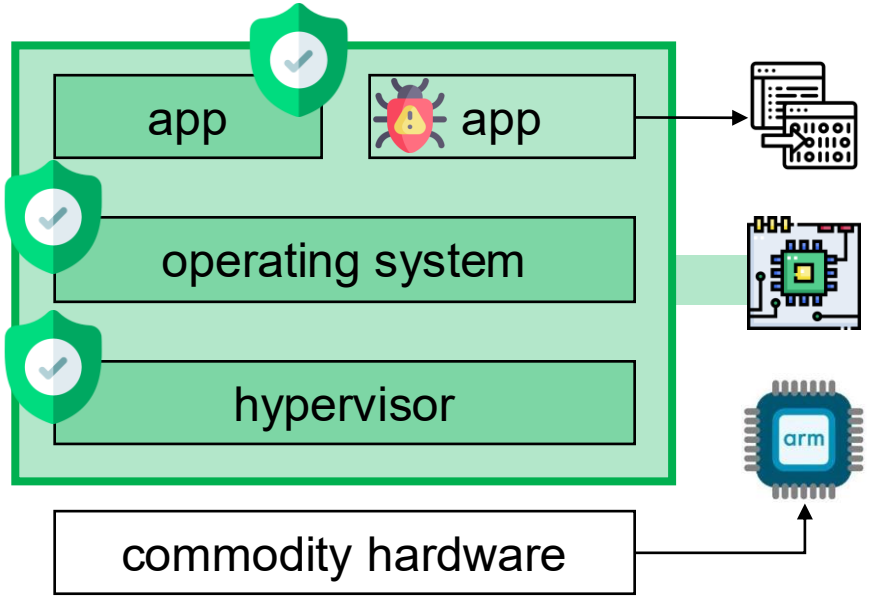
Hardware to the rescue…

- Purely software-based solutions are slow ➔ seek assist from hardware
- Special-purpose hardware is fast but slow to make real-world adoption
- Security extensions in commodity hardware are both fast & widely deployed



Hardware to the rescue…

- Purely software-based solutions are slow ➔ seek assist from hardware
- Special-purpose hardware is fast but slow to make real-world adoption
- Security extensions in commodity hardware are both fast & widely deployed



SW-only ➔ Slow

Special-purpose HW ➔ Fast, low adoption

Features on commodity HW ➔ Fast, high adoption

This talk!

Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

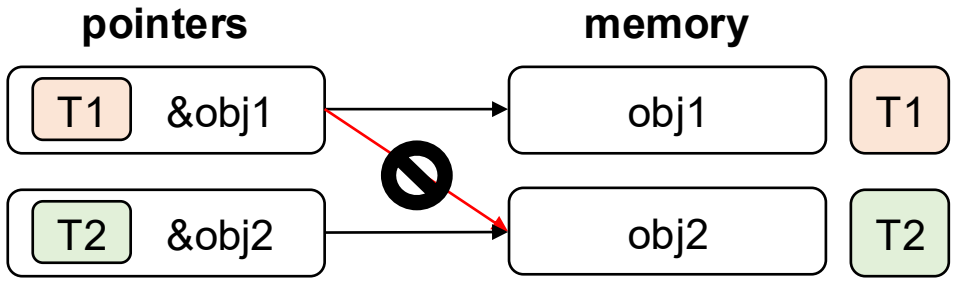
current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

ARM Memory Tagging Extension (MTE)

- Introduced in ARMv8.5-A architecture
- Deployed in COTS devices (Google Pixel 8 ~, Samsung Galaxy 22 ~)
- Associate 4-bit tags to pointers and 16-byte memory blocks
  - Pointer tags are stored in (unused) upper bits of pointers
  - Memory tags are stored in a dedicated area of physical memory
- Hardware checks pointer tag and memory tag on memory access
- Tag mismatch raises a tag check fault



Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

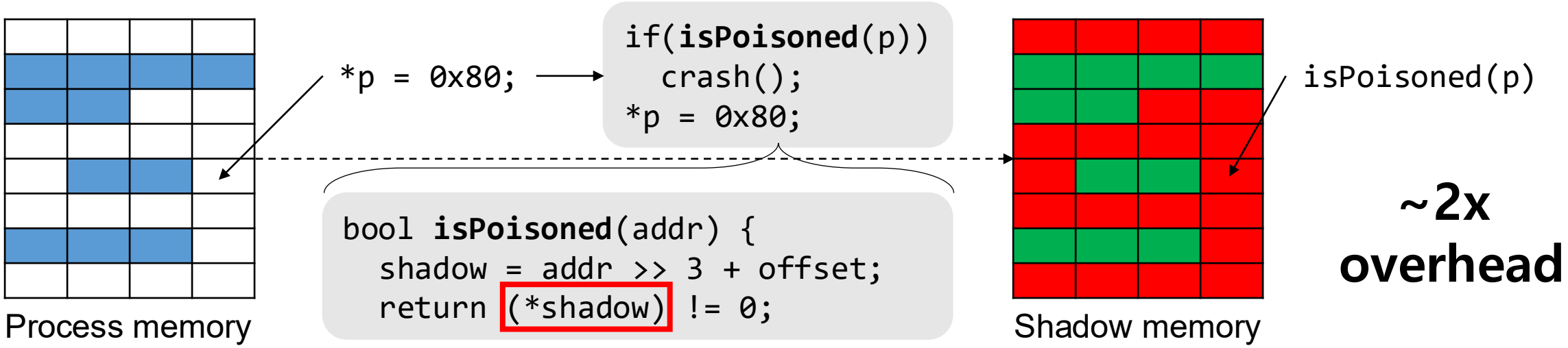
Hardware-assist is fast, but not a panacea

Spatial memory safety

- Buffer (stack/heap) overflow (out-of-bounds)

```
char *ptr = malloc(size);
*(ptr + size) = val; // buffer overflow
```

- Software-only solution: AddressSanitizer ➔ **extra** load per memory access

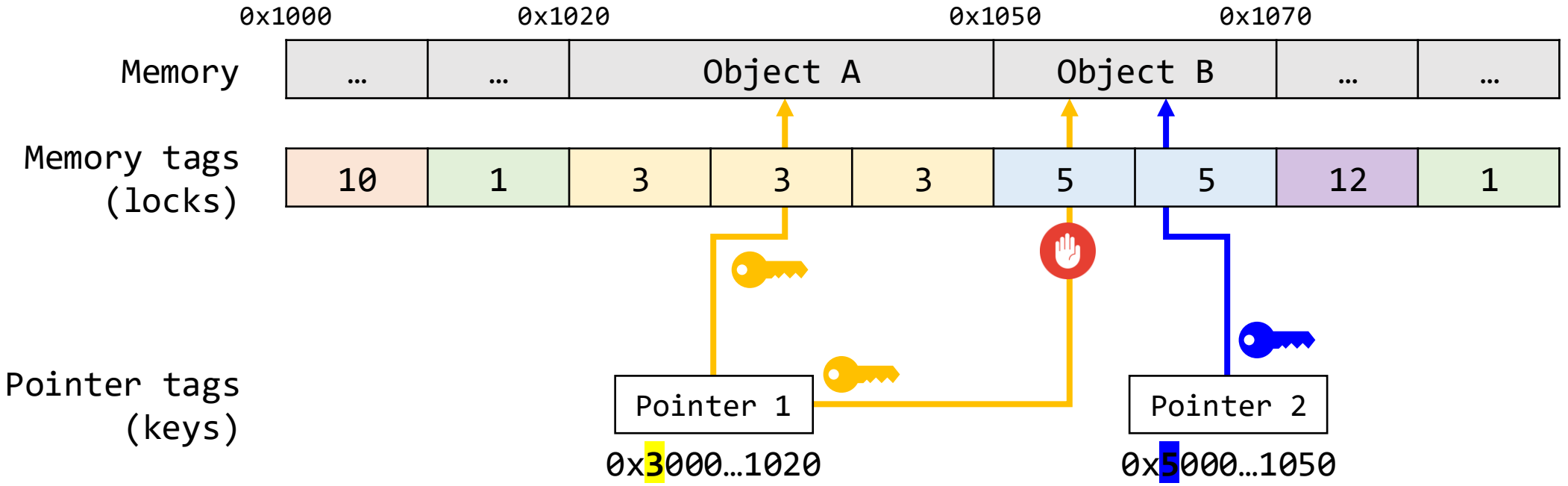


~2x overhead

reference: https://suelan.github.io/2020/08/18/20200817-address-sanitizer/

Spatial memory safety with MTE

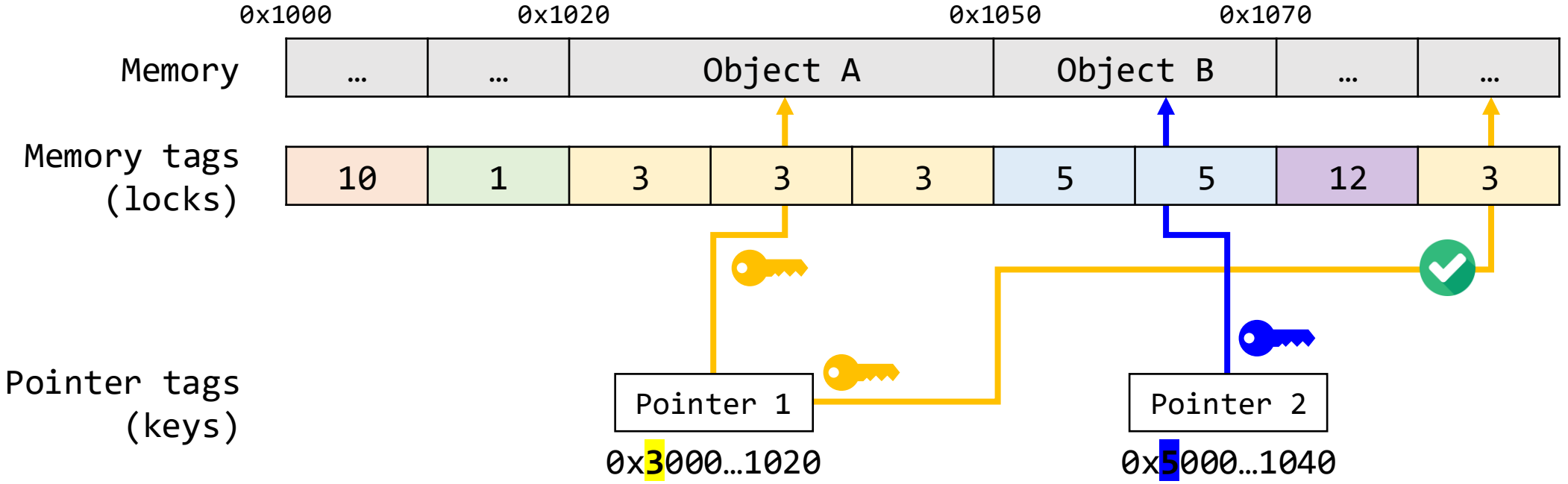
- Tag management during memory allocation
  - On `ptr = malloc(size)`, assign pointer tag to `ptr` and memory tag to allocated memory
  - MTE can detect overflow through hardware tag checks



reference: https://www.vusec.net/projects/stickytags/

Spatial memory safety with MTE – enough…?

- Limited number of tags (N = 16)
  - Cannot assign different tag values to respective (live) objects
  - Cannot detect overflow to (object) memory with a identical memory tag



reference: https://www.vusec.net/projects/stickytags/

Deterministic spatial memory safety with MTE

- ZOMETAG
  - Use 16 MTE tags ***per (4GB) zone***
  - Instrument pointer arithmetic to stay within the zone
    ➔ Use 32-bit register for address offset
    ➔ Check 32th bit after add/sub instruction

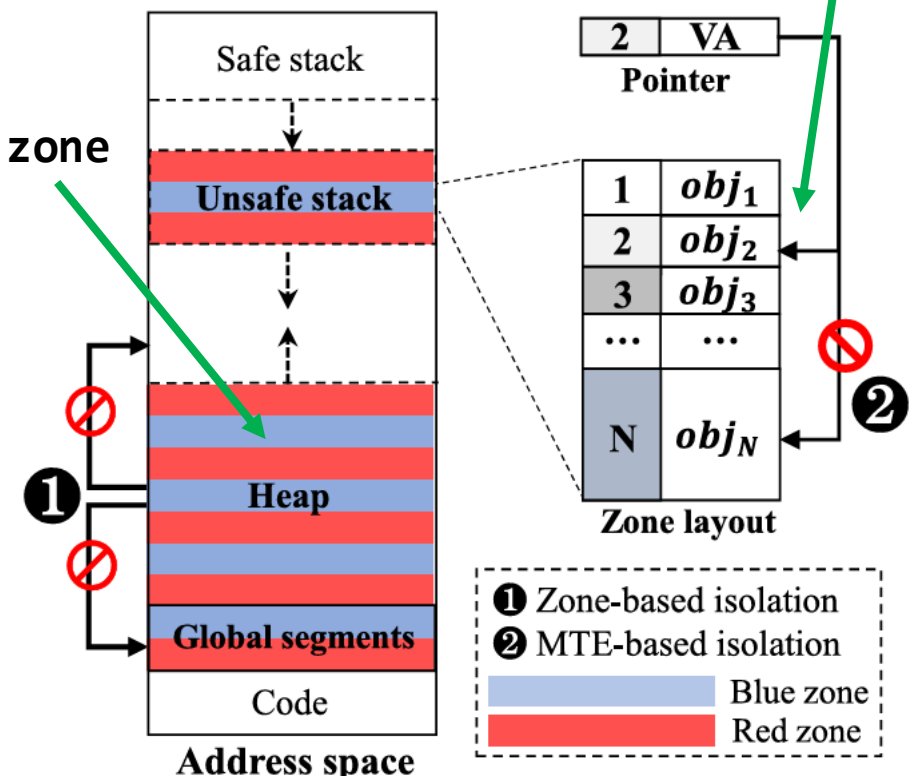
32-bit register

```
LDR     X0, [X1, X2]      ──→   LDR     X0, [X1, W2]
```

```
LDR     X7, [SP, #32]
ADD     X7, X7, #256      ──→   LDR     X7, [SP, #32]
LDR     X0, [X7]              ADD     X7, X7, #256
                              TBZ     X7, #32, <crash>
                              LDR     X0, [X7]
```

Check 32nd bit

16 objects With different MTE tags per zone



Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

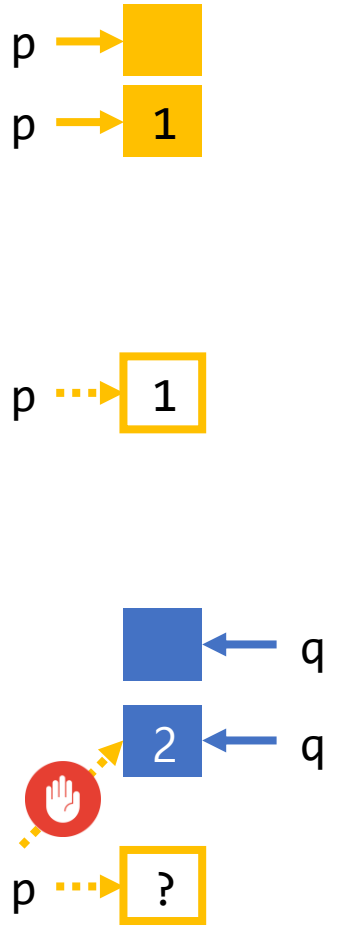
Temporal memory safety

- Use-after-free, double free, uninitialized access

```
char *ptr = malloc(size);
free(ptr);
*ptr = val; // use-after-free
```

- Software-only solution: one-time allocation (OTA)
  - Use virtual address ***only once*** for memory allocation
  - Consumes virtual address space for long-running apps
  - Breaks optimizations (e.g., fragmentation)

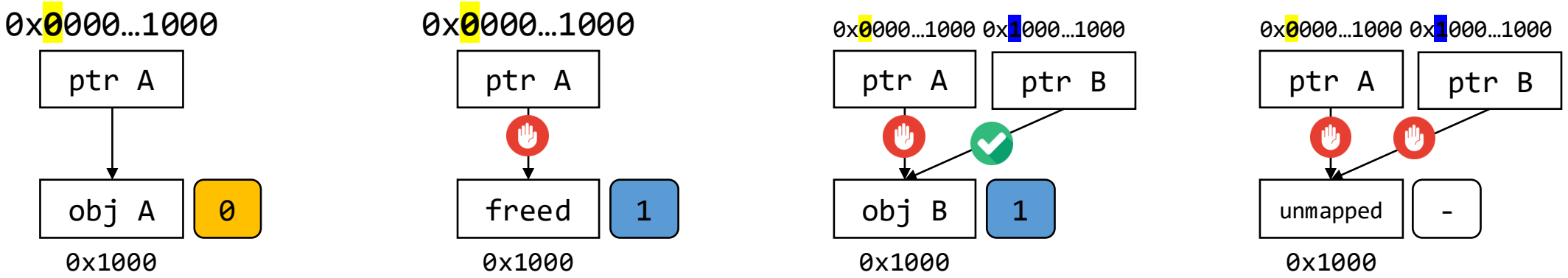
```
p = malloc(8);
*p = 1;
if (*p == 1) {
  …
  free(p);
}
…
q = malloc(8);
*q = ReadNet();
assert(p != q);
if (*p == 2) {
  …
```



reference: https://www.usenix.org/system/files/sec21_slides_wickman.pdf

Temporal memory safety with MTE

- VATAlloc (**V**irtual **A**ddress **T**agging **Alloc**ation)
  - Reuse virtual address for memory reallocation with MTE
    ➔ increment memory tag for each allocation (i.e., 0 ~ 15)
    ➔ virtual address is not reused after tag exhaustion (N = 16)
  - Slow down virtual address consumption by 16x



1) Object allocation (`malloc`)
2) Object allocation (`free`)
3) Object reallocation (`malloc, realloc`)
4) Tag exhaustion (`unmap`)

Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

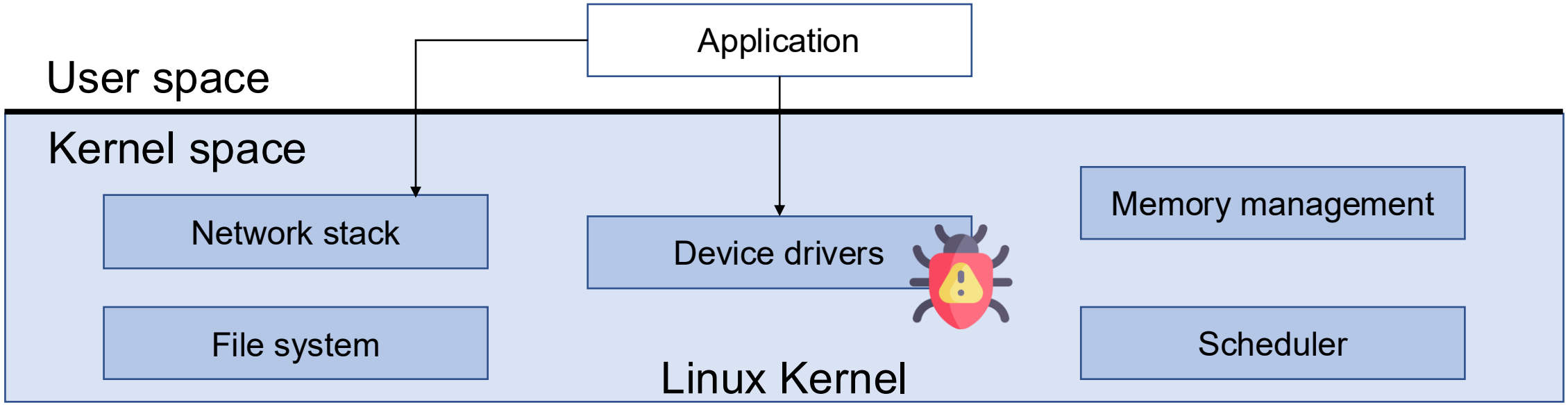
current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

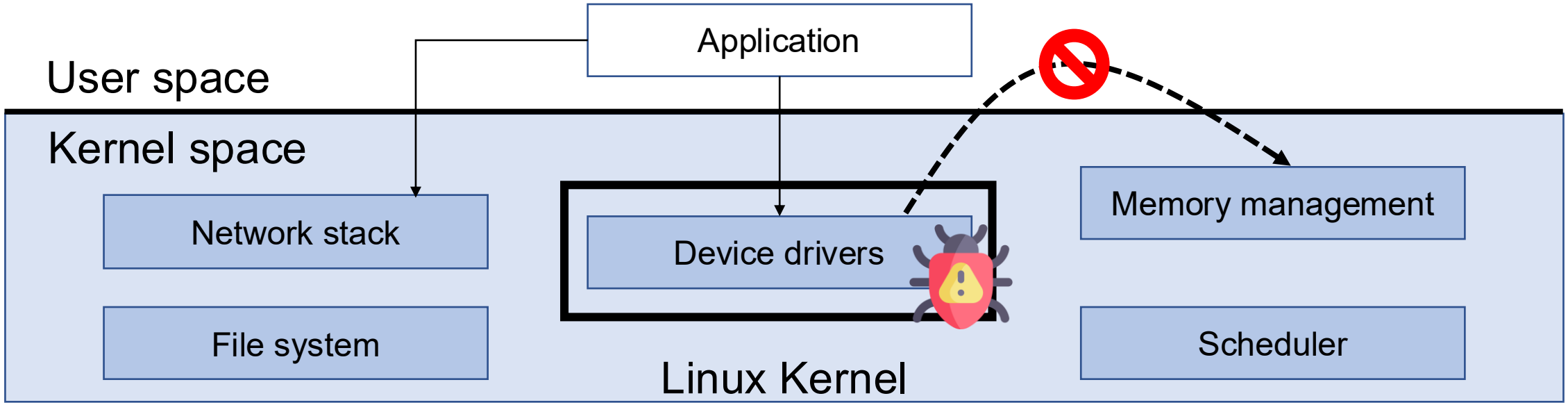
Commodity kernels

- Commodity kernels are **monolithic**
  - Kernel components (e.g., network stacks, drivers) run in the same address space
  - Vulnerability in single component propagates to the entire kernel

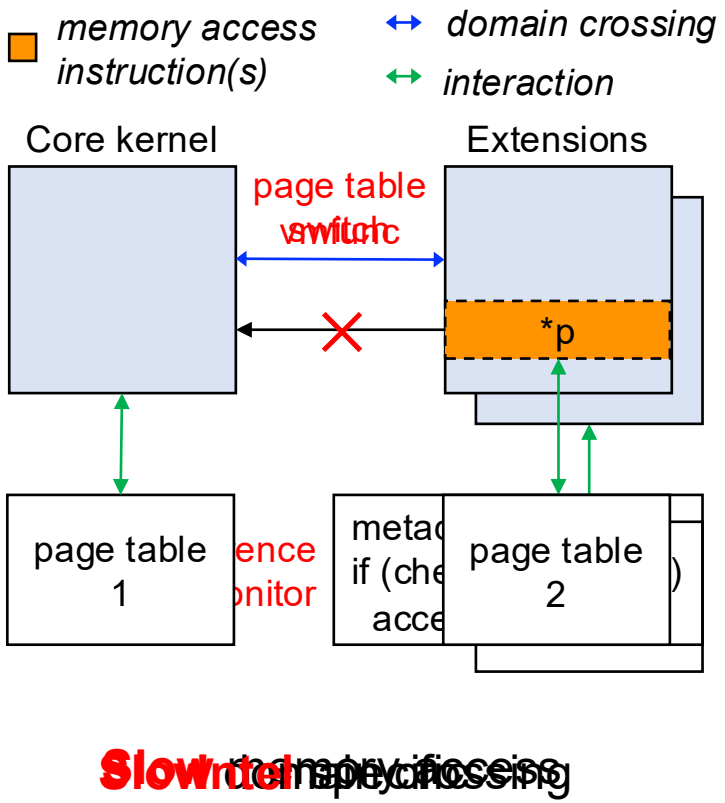


Isolation/Compartmentalization

- Need to isolate potentially vulnerable kernel components
  - E.g., Driver vulnerabilities ➔ **~ 50%** of all Linux kernel CVEs



Limitations of existing works

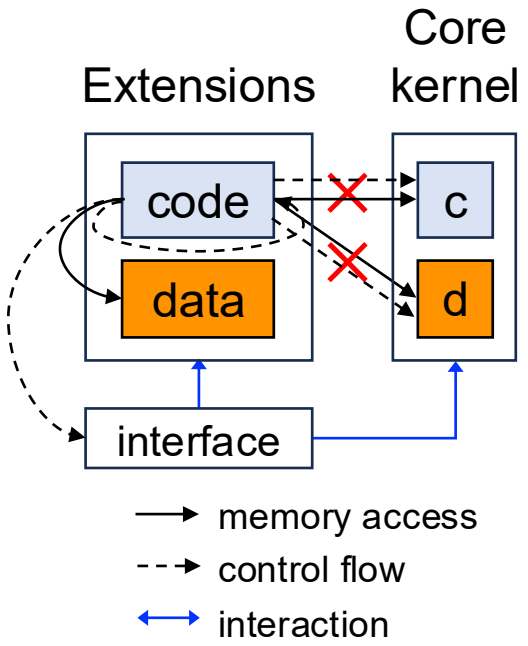


Mechanism	Memory access	Domain crossing	Architecture
Inline monitors (SW)	**Slow**	Fast	-
Page table-based	Fast	**Slow**	-
HW-assisted	Fast	Fast	**Intel**
Our solution	**Fast**	**Fast**	**ARM**

Can we leverage **MTE** for **efficient** isolation of kernel extensions on **ARM**?

Isolation goal and requirements

- Goal
  - Isolate kernel extensions (e.g., device drivers) from the core kernel on ARM
- Security requirements
  - ① Isolate memory access
  - ② Isolate control flow
  - ③ Manage interaction between the core kernel and extensions

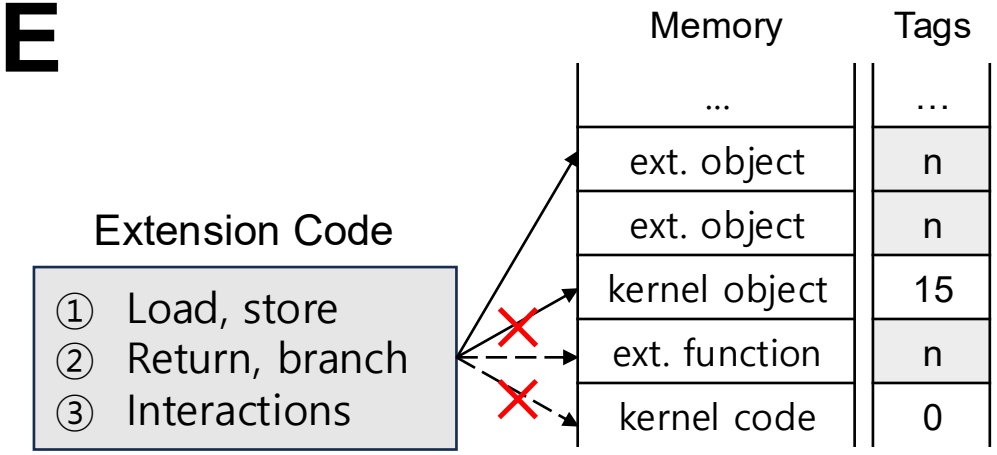


Kernel driver isolation with MTE

① Isolating memory access
- [pointer tag] extension: tag *n*, core kernel: tag *15*
- [memory tag] extension: tag *n*, core kernel: tag *15*
- Reserved register with pointer tag *n*

② Isolating control flow
- Tag valid branch targets with memory tag *15*
- Dummy load before branch to perform tag check

③ Interaction management
- Dynamically (un)tag objects passed during kernel-to-extension interactions (e.g., arguments)



Original instruction		Instrumented instructions	
LDR	$X_0, [X_m]$	BFXIL LDR	$\boldsymbol{X_{rsv}}, X_m, \#0, \#38$ $X_0, [\boldsymbol{X_{rsv}}]$
STR	$X_0, [X_m]$	BFXIL STR	$X_{rsv}, X_m, \#0, \#38$ $X_0, [X_{rsv}]$
RET		BFXIL LDR BR	$X_{rsv}, LR, \#0, \#38$ $XZR, [X_{rsv}]$ X_{rsv}

$* X_{rsv}$: | *n* | (pointer) |

→ memory access

⇢ control transfer

Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

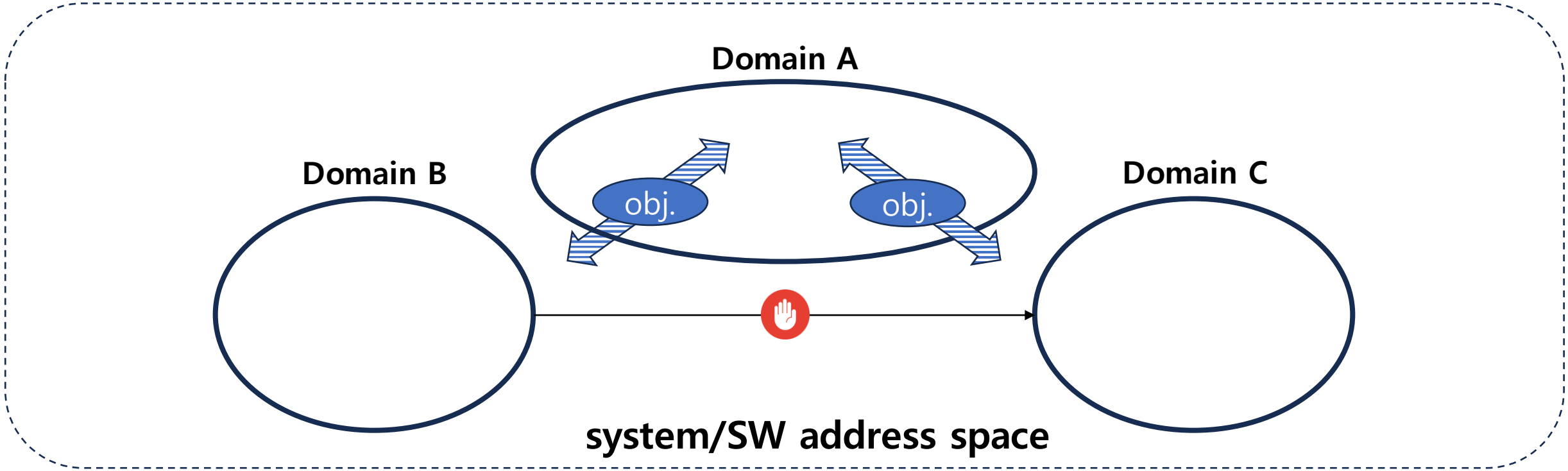
current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

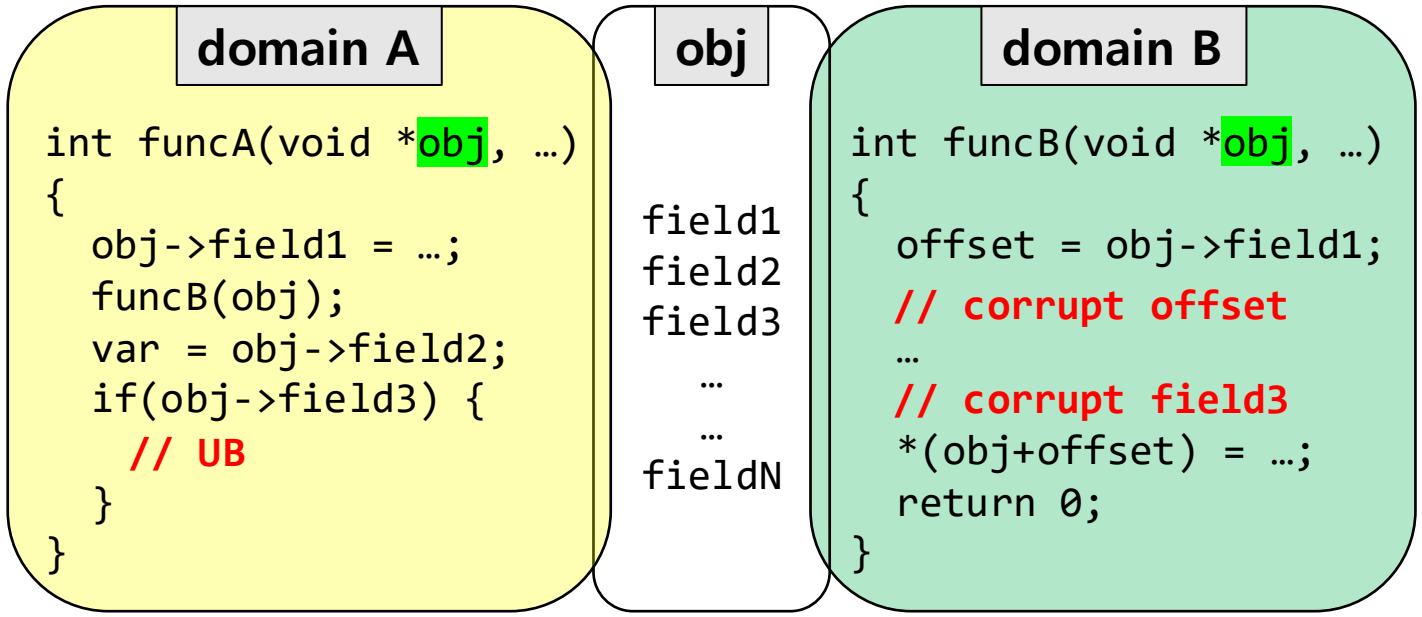
Shared Memory

- Necessary for domain interaction and communication
- Private memory protected by isolation / compartmentalization



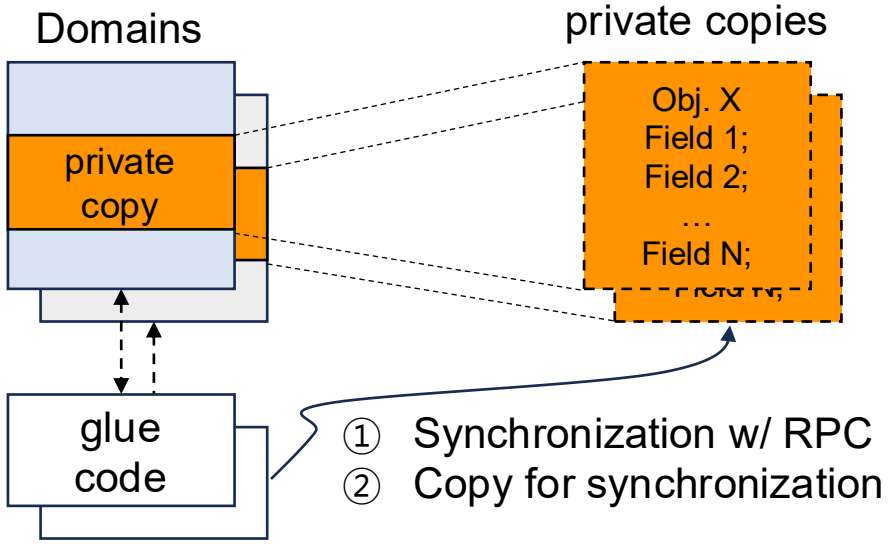
Access control on Shared Memory

- Blunt access control can compromise interacting domain(s)
  - e.g., unrestricted access permissions
  - CVE-2021-21309, CVE-2022-21769, CVE-2022-48198, …
- Protection Requirements
  - Per-domain permissions
  - Multiple permissions (read-write, read-only, na)
  - Byte-level granularity



field1➔A:rw,B:ro | field2➔A:ro,B:rw | field3➔A:rw,B:na

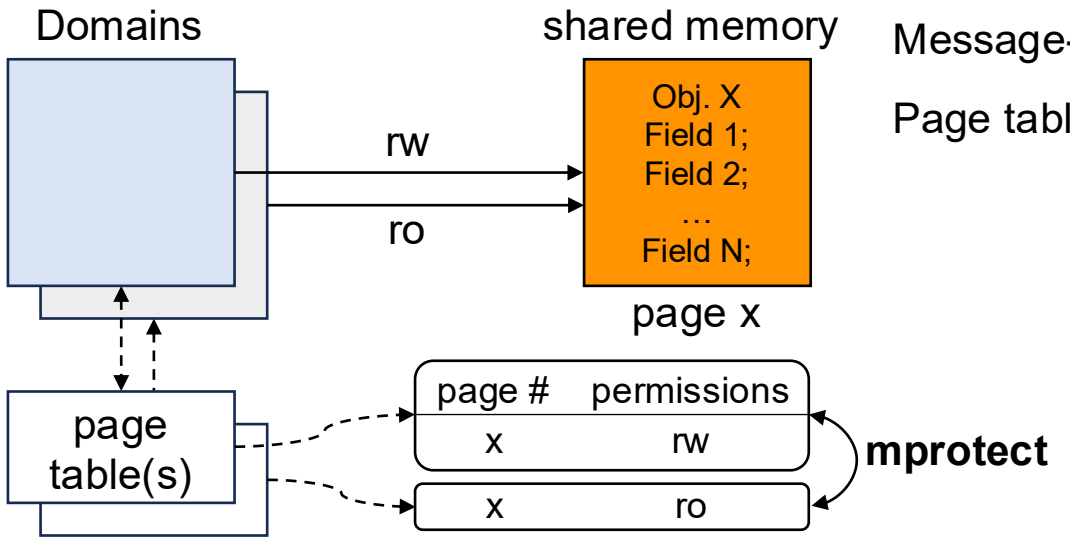
Solutions for access control on shared memory



Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte

Slow synchronization

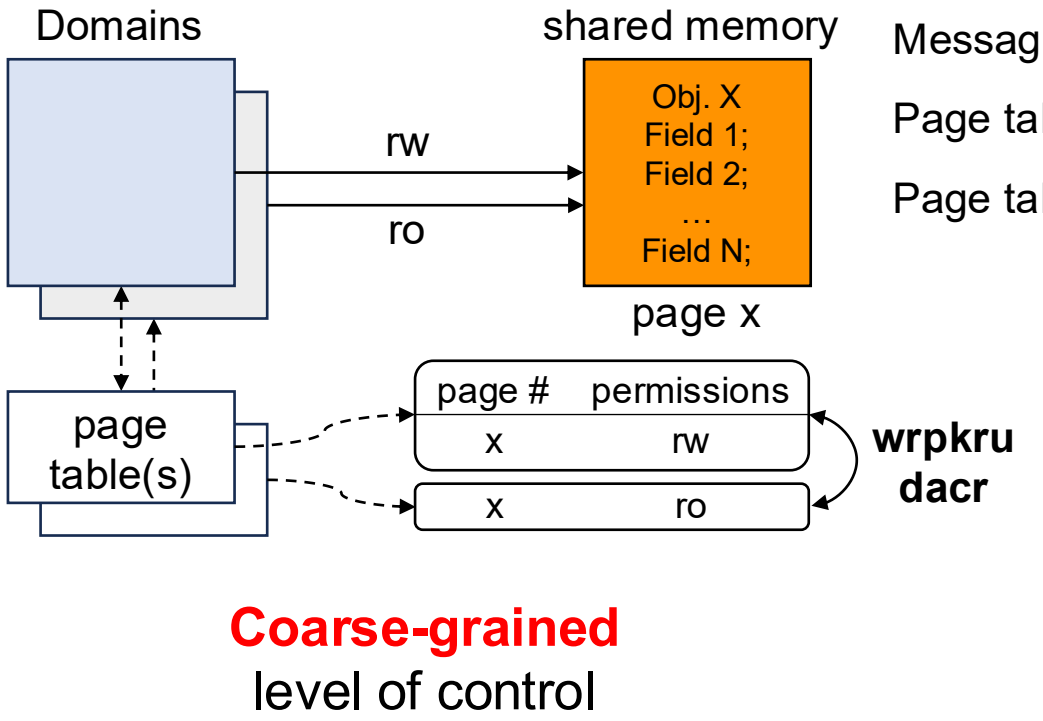
Solutions for access control on shared memory



Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte
Page table-based	Per-domain/multiple	**Slow** (w/ SW)	**page**

Coarse-grained
level of control

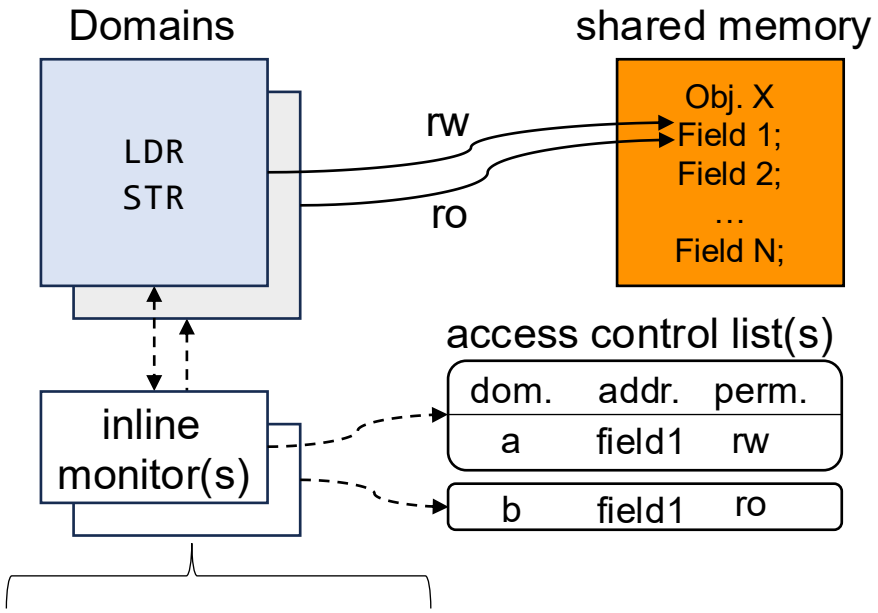
Solutions for access control on shared memory



Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte
Page table-based	Per-domain/multiple	**Slow** (w/ SW)	**page**
Page table-based	Per-domain/multiple	**Fast** (**w/ HW**)	**page**

Coarse-grained
level of control

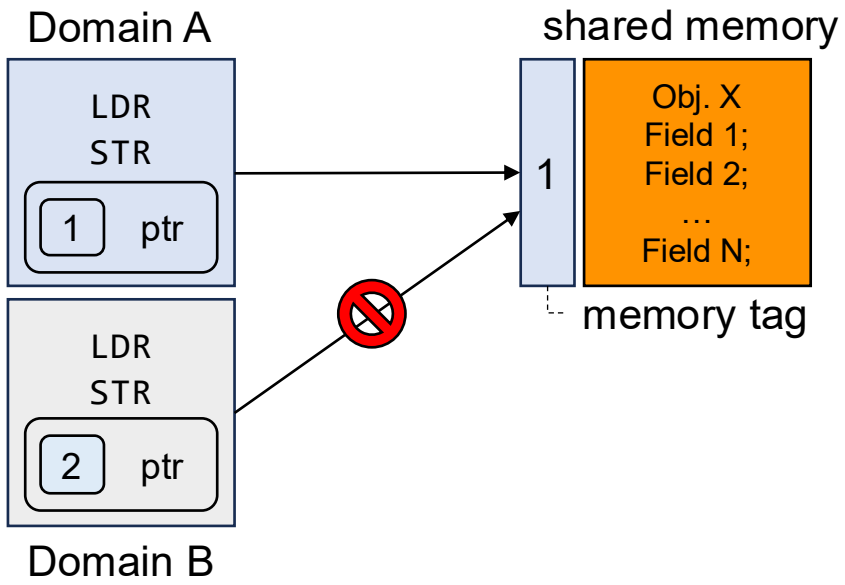
Solutions for access control on shared memory



① Calculate index
② Lookup metadata ➔ **Slow** checks
③ Check validity

Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte
Page table-based	Per-domain/multiple	**Slow** (w/ SW)	**page**
Page table-based	Per-domain/multiple	Fast (w/ HW)	**page**
Inline monitors	Per-domain/multiple	**Slow**	byte
Inline monitors	Per-domain/multiple	**w/ HW ?**	byte

Solutions for access control on shared memory



16B granularity
Binary access permission
Single domain access control

Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte
Page table-based	Per-domain/multiple	**Slow** (w/ SW)	**page**
Page table-based	Per-domain/multiple	Fast (w/ HW)	**page**
Inline monitors	Per-domain/multiple	**Slow**	byte
Inline monitors	Per-domain/multiple	**w/ HW ?**	byte
MTE-only	**one-domain/binary**	Fast	**16B**
BASTAG	**Per-domain/multiple**	**Fast**	**byte**

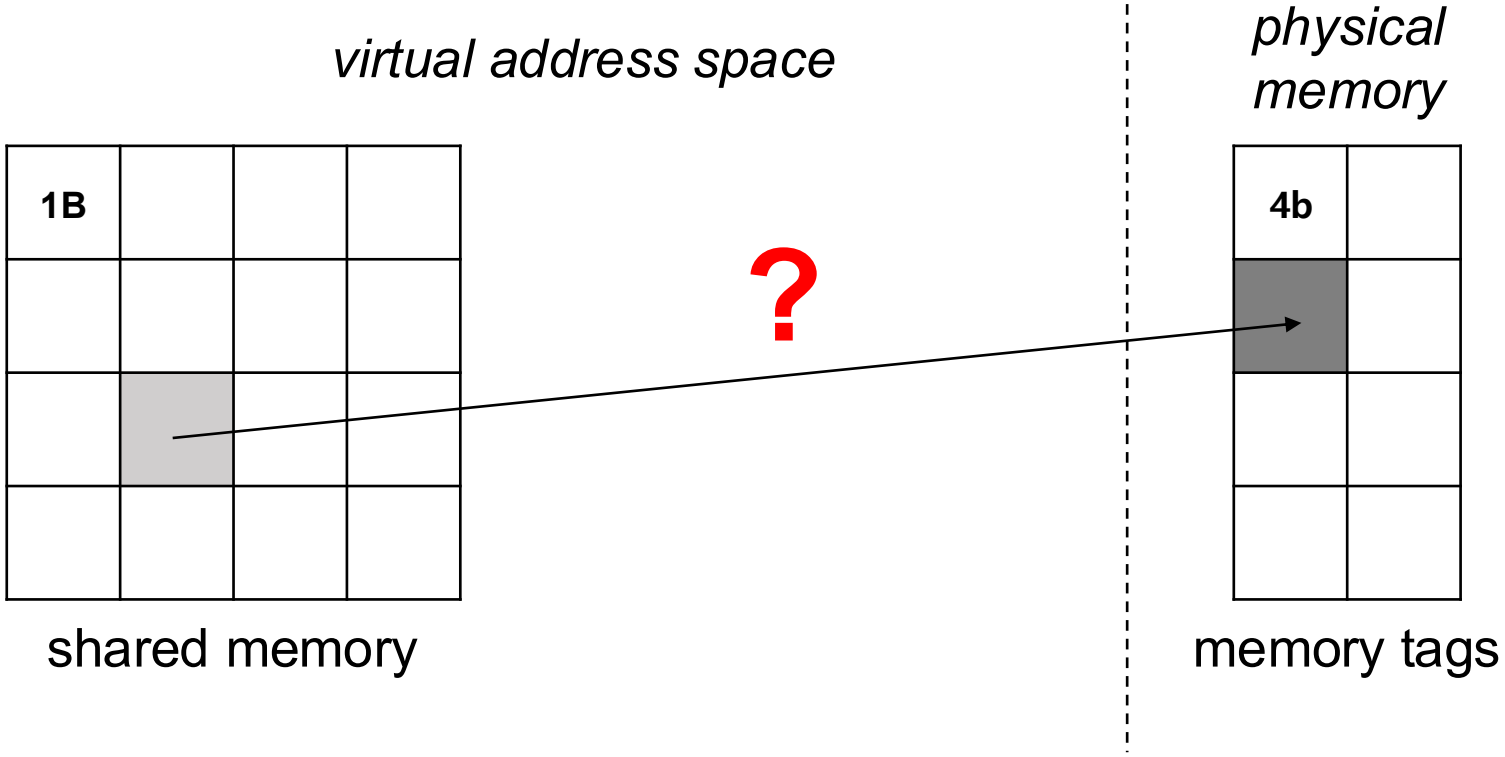
How can we leverage **MTE** for **efficient multi-domain, multi-policy byte-level** access control?

Access control on shared memory with MTE

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE

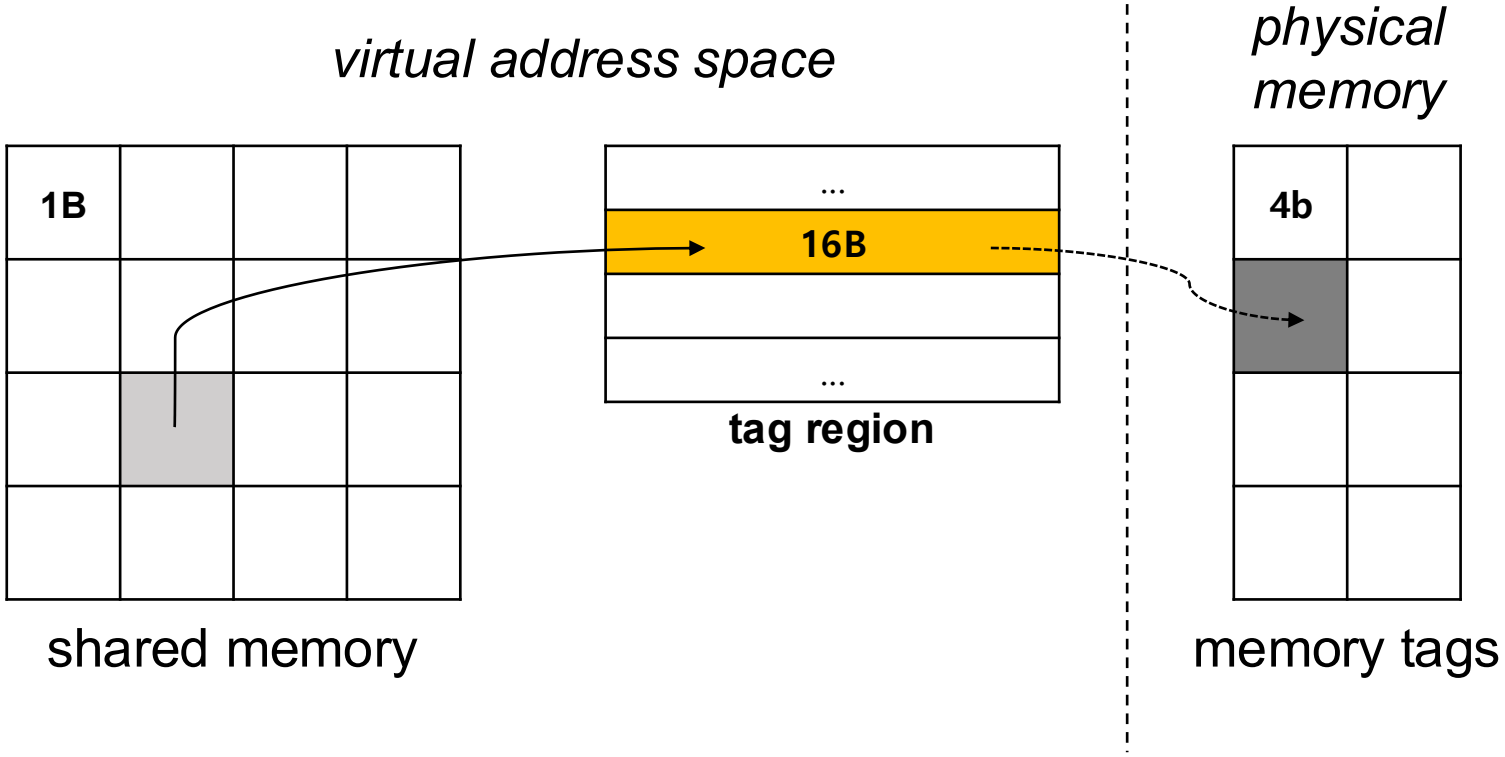
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



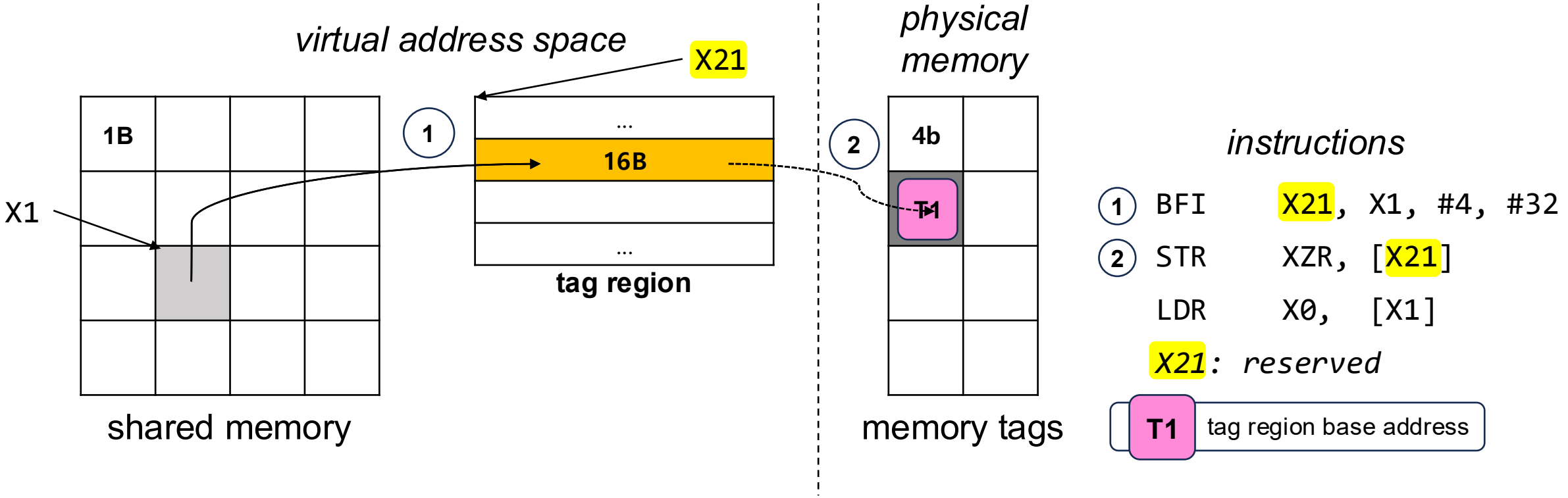
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



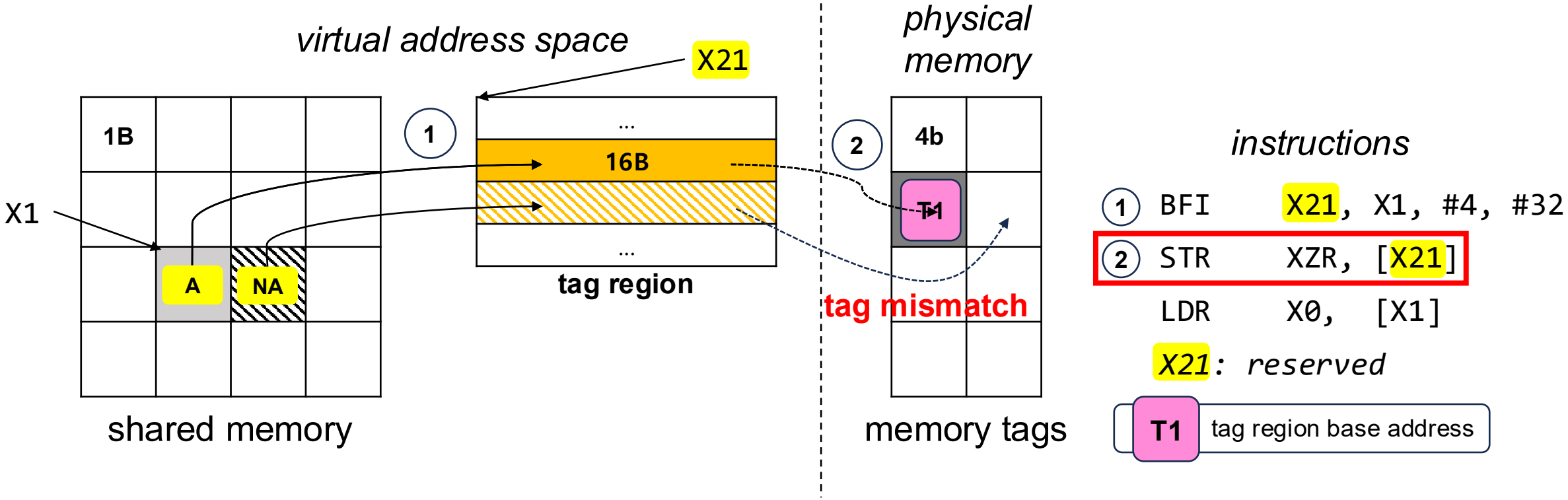
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



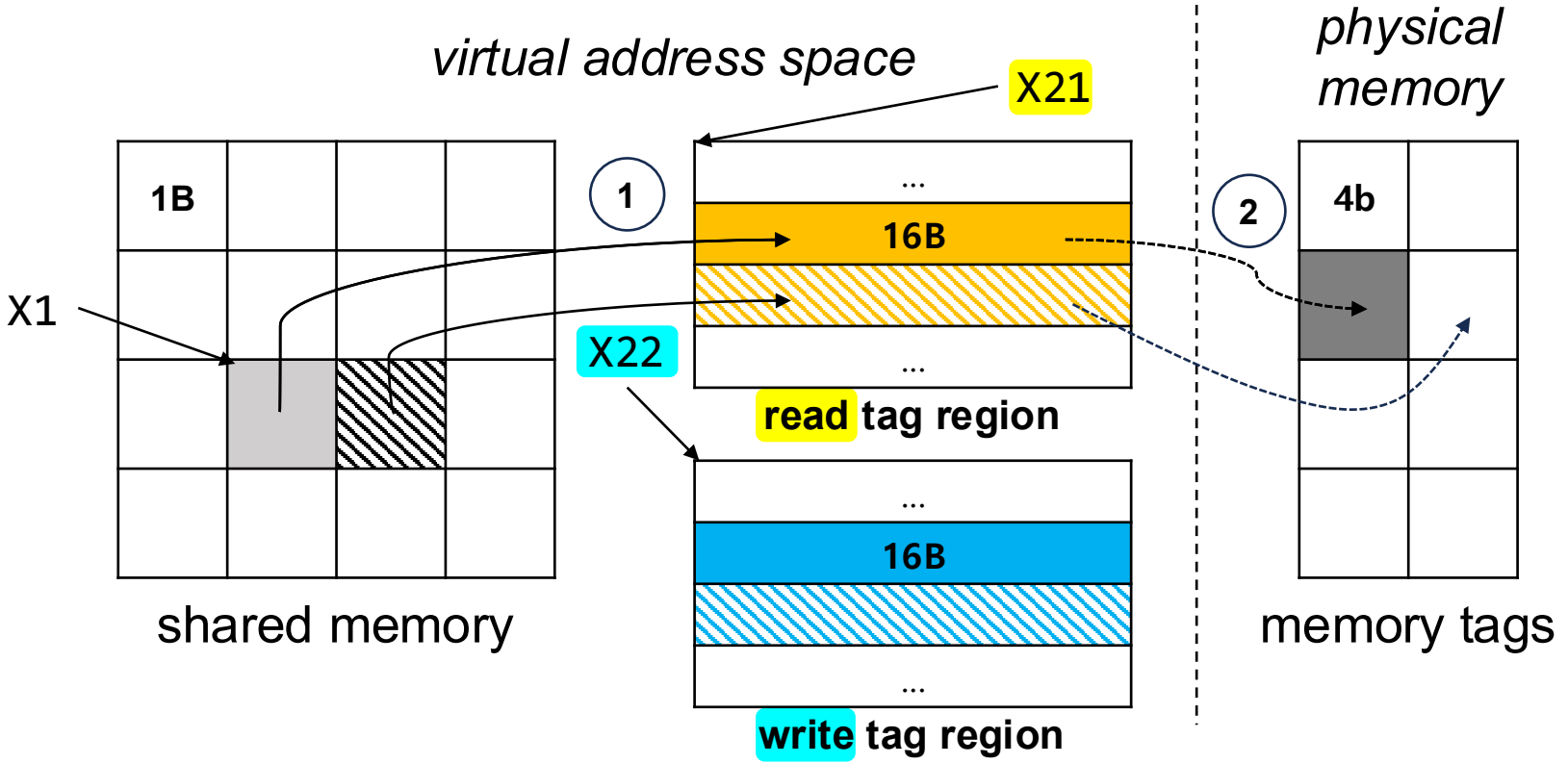
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE

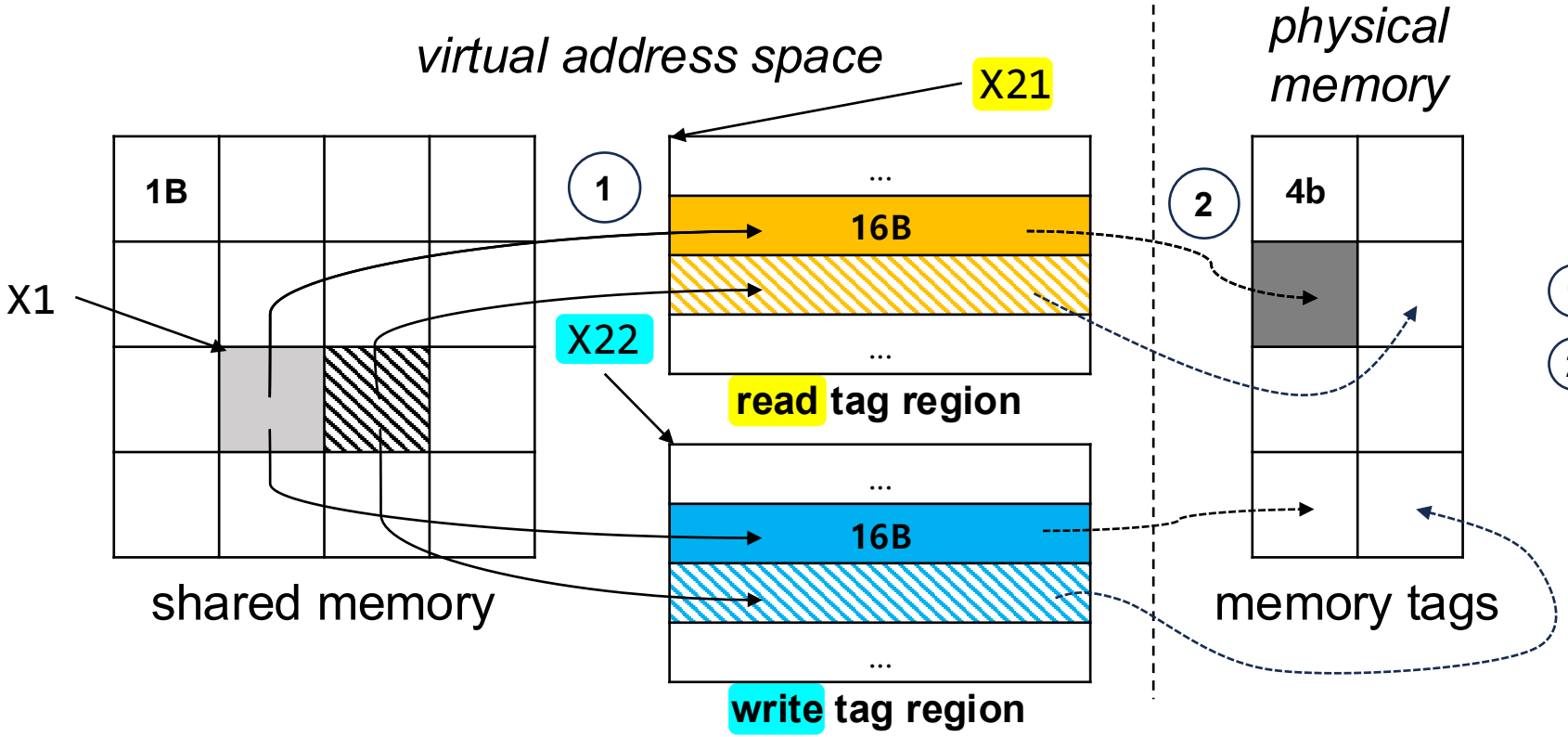


instructions

```
1  BFI    X21, X1, #4, #32
2  STR    XZR, [X21]
   LDR    X0,  [X1]
X21: reserved
```

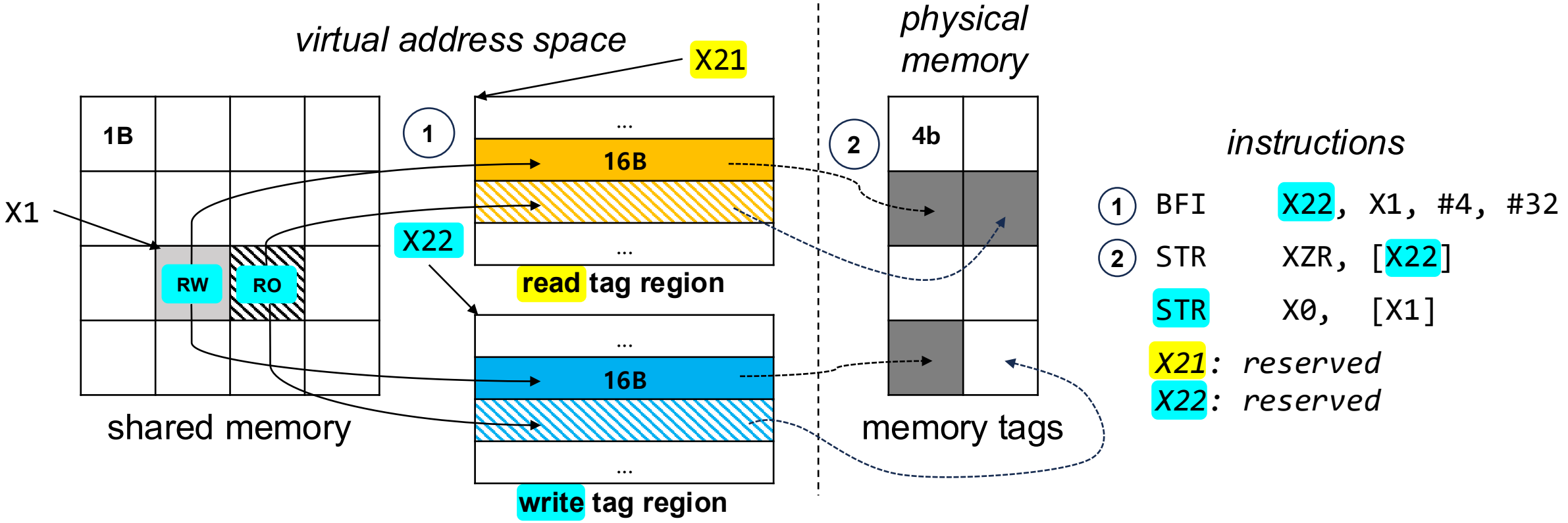
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



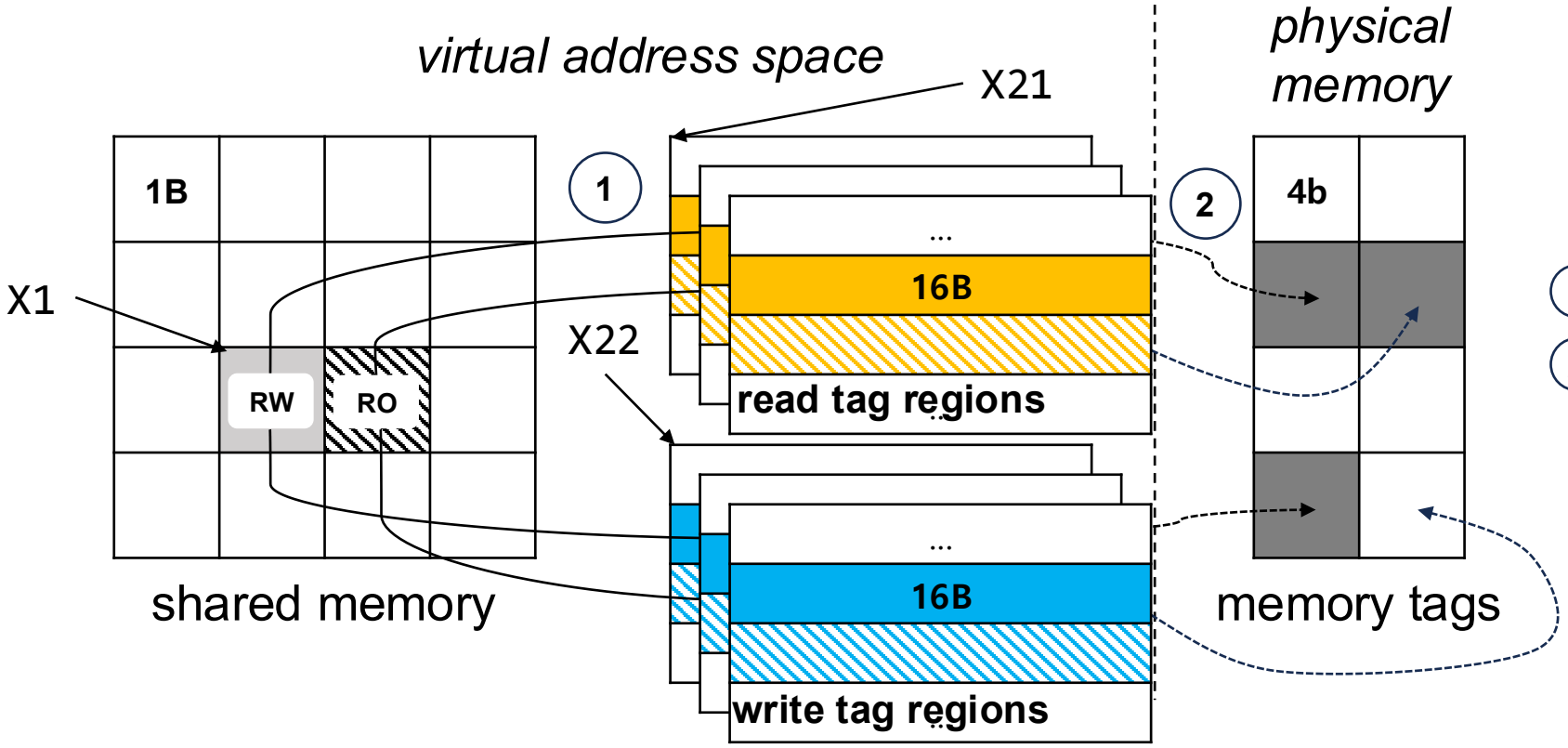
Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



Shadow Memory Tagging

- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE

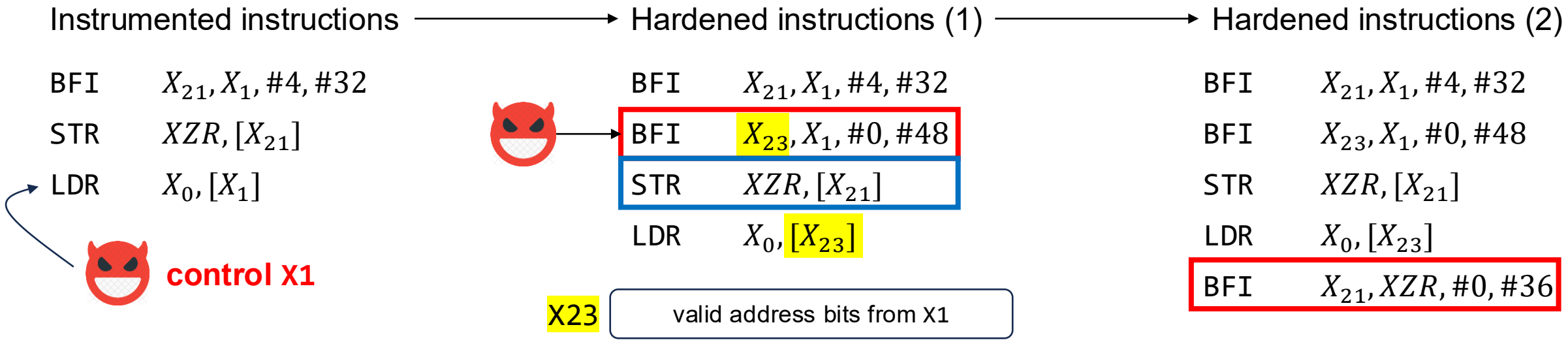


instructions

```
1  BFI    X22, X1, #4, #32
2  STR    XZR, [X22]
   STR    X0,  [X1]
X21: reserved
X22: reserved
```

Bypass Prevention

- Attacker may subvert the control flow to bypass the access control checks



Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

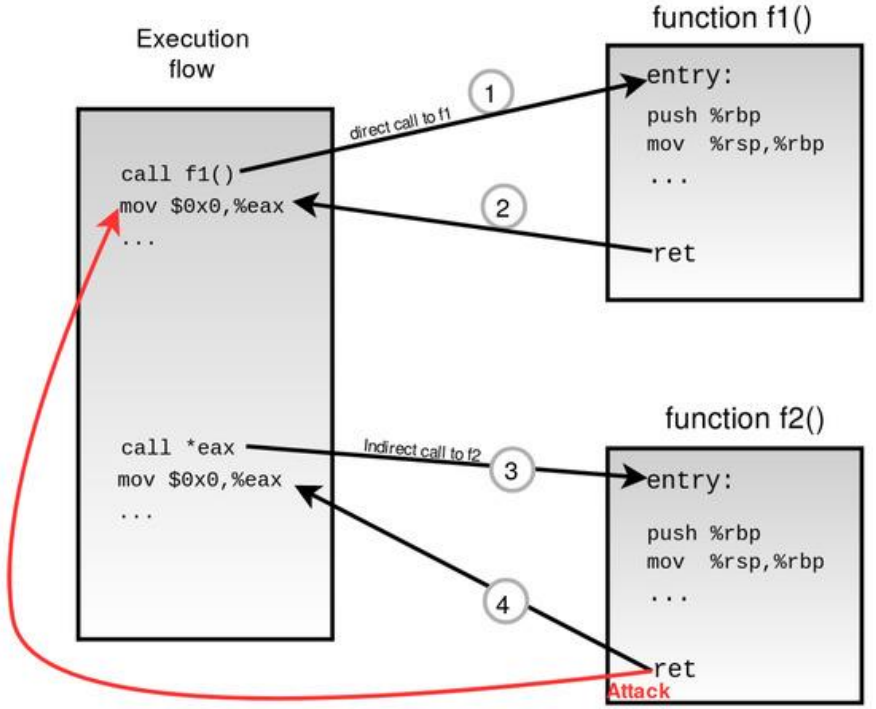
current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

Control-flow integrity

- Return oriented programming (ROP) and jump oriented programming (JOP)
  - Manipulate return and jump addresses to hijack the program execution
  - Code injection attack, code reuse attack, …
- Backward-edge protection
  - Protect return addresses = protect the stack!
  - Shadow stack, stack canaries…
- Forward-edge protection
  - How can we detect jump target address (function pointer) modification?



reference: https://www.mdpi.com/2076-3417/9/20/4229

ARM Branch Target Identification (BTI)

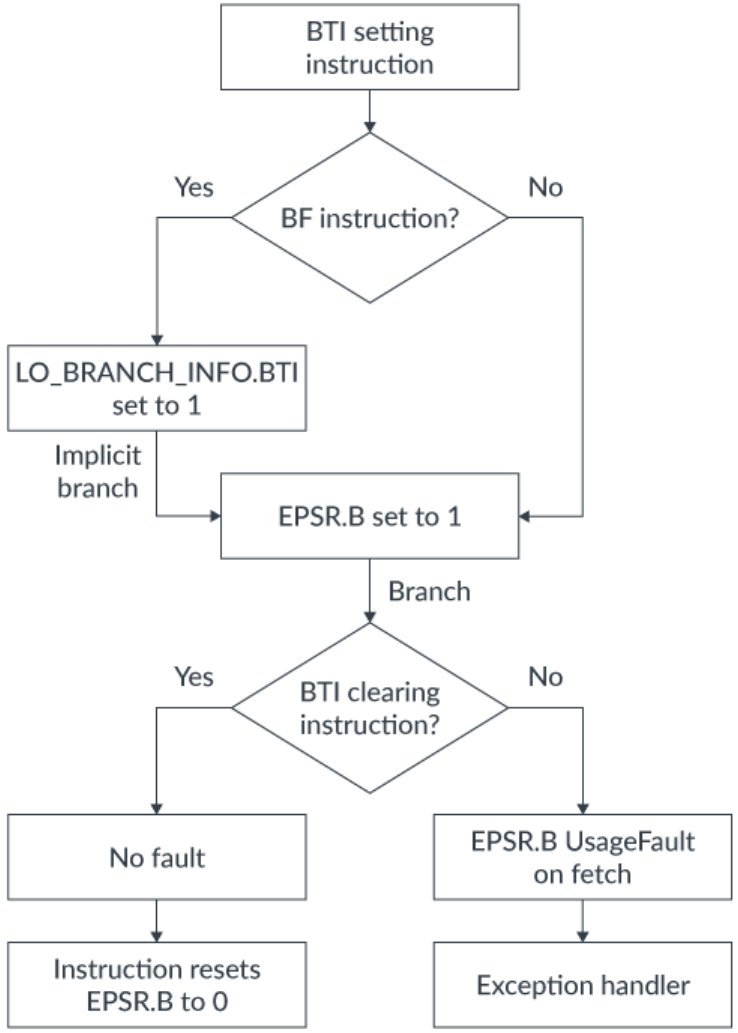
- Introduced in ARMv8.5-A (with MTE)
- Hardware enforces the indirect branch instructions to only target valid ***landing pad*** instructions
- Memory holding such landing pad instructions are marked with ***Guarded Page*** bit in the page tables

```
main:
LDR     X7, @func_addr
BLR     X7
```

```
malicious_func:
ADD     X7, X7, #imm
BR      X7
```

```
original_func:
BTI
ADD     X0, X0, #imm
STR     X0, [X1]
```

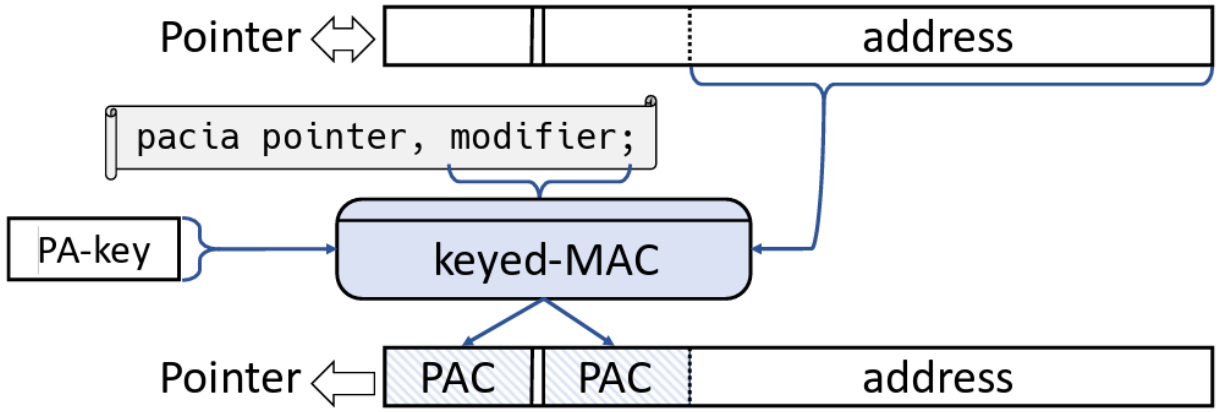
Attackers can still chain BTI-enabled gadgets



reference: https://developer.arm.com/community/arm-community-blogs/b/architectures-and-processors-blog/posts/armv8-1-m-pointer-authentication-and-branch-target-identification-extension

ARM Pointer Authentication (PA)

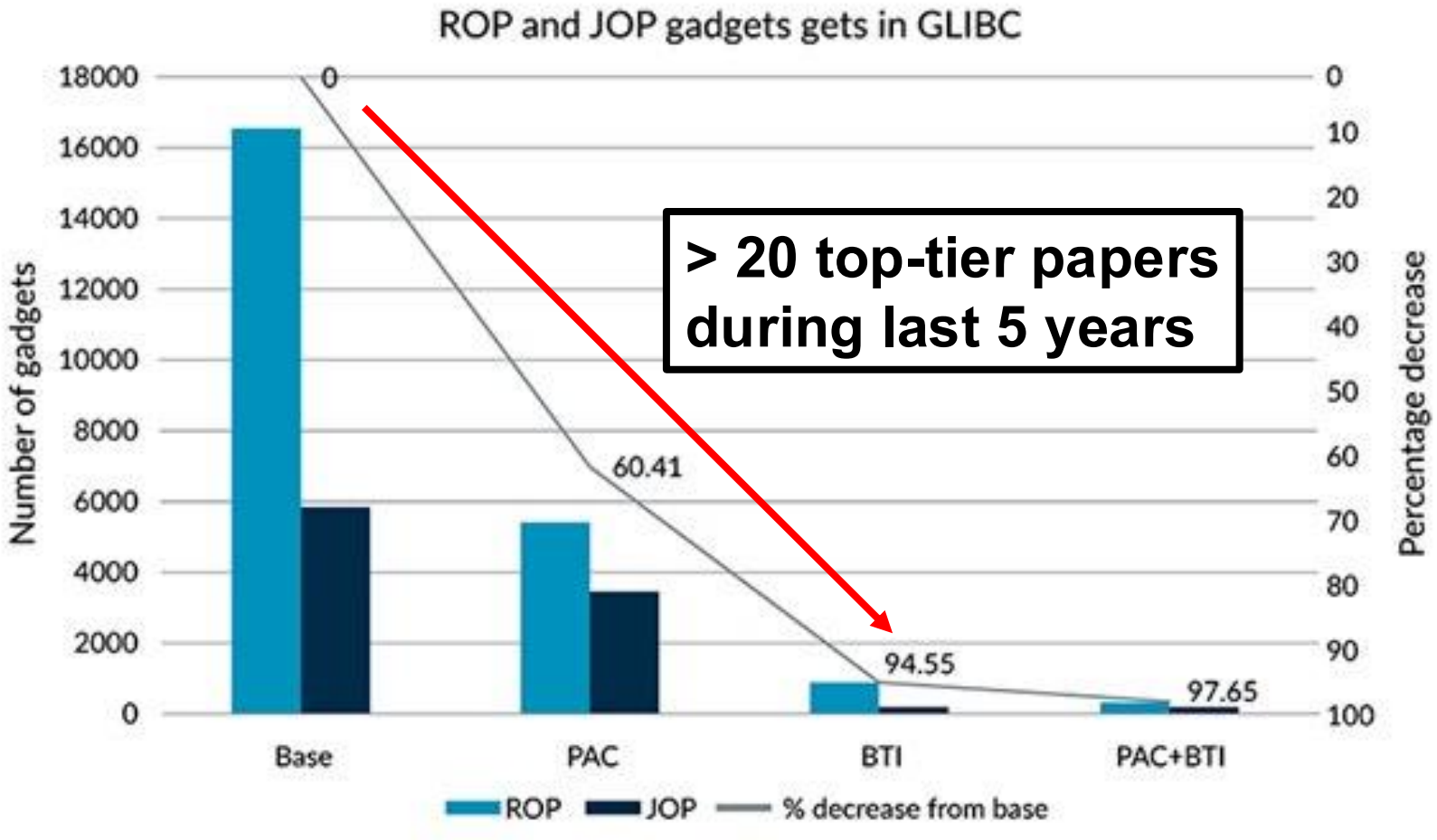
- Introduced in ARMv8.3-A
- Check the integrity of pointers with ***pointer authentication codes*** (PACs)
  - PACs are stored in unused upper bits of the pointer
  - Additional instructions to create and validate PACs
- PACs are generated on pointer creation (store), authenticated on use (load)



reference: https://www.usenix.org/system/files/sec19-liljestrand_0.pdf

```
function:
PACIASP                    // Create PAC
STP     FP, LR, [SP, #0] // Store LR
…
LDP     FP, LR, [SP, #0] // Load LR
AUTIASP                   // Authenticate PAC
RET
```

ARM PA+BTI



reference: https://newsroom.arm.com/blog/pac-bti

Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

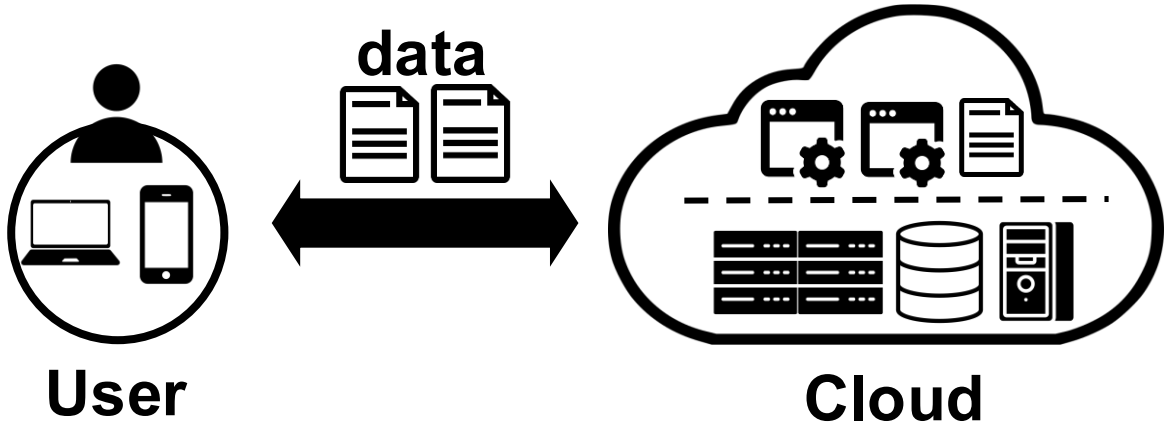
Confidential Computing

- Cloud and remote computing

Cloud

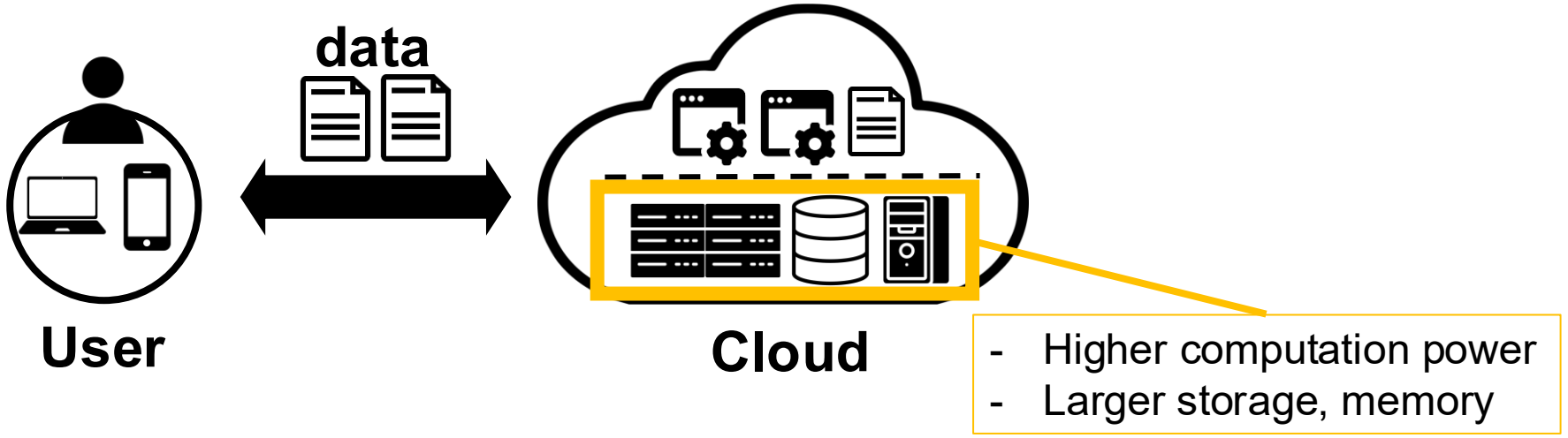
Confidential Computing

- Cloud and remote computing



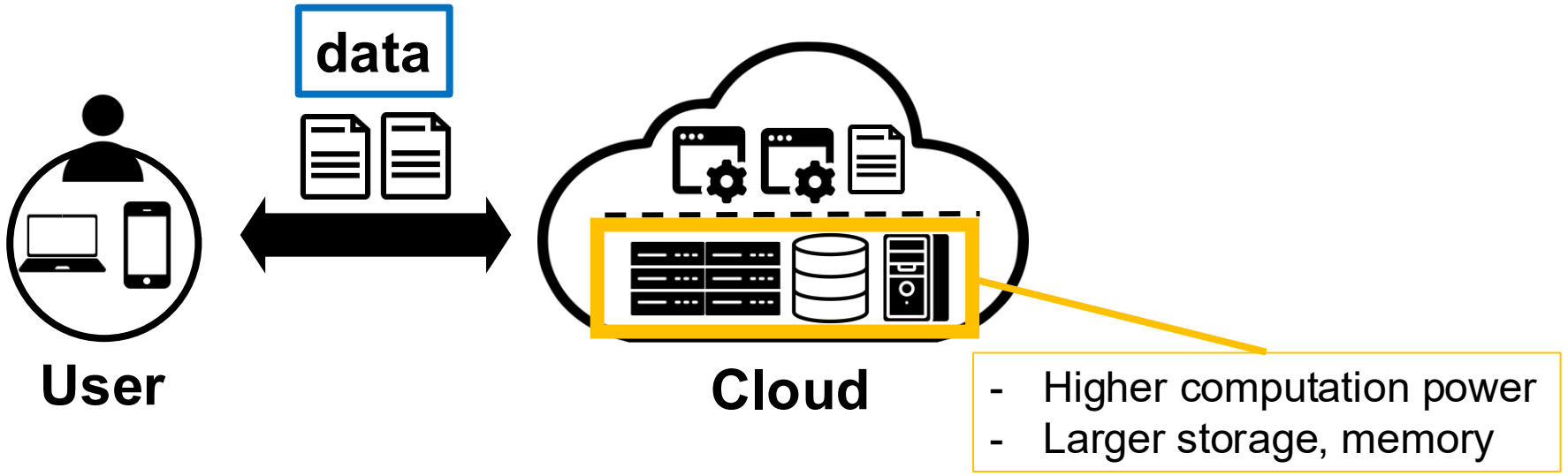
Confidential Computing

- Cloud and remote computing



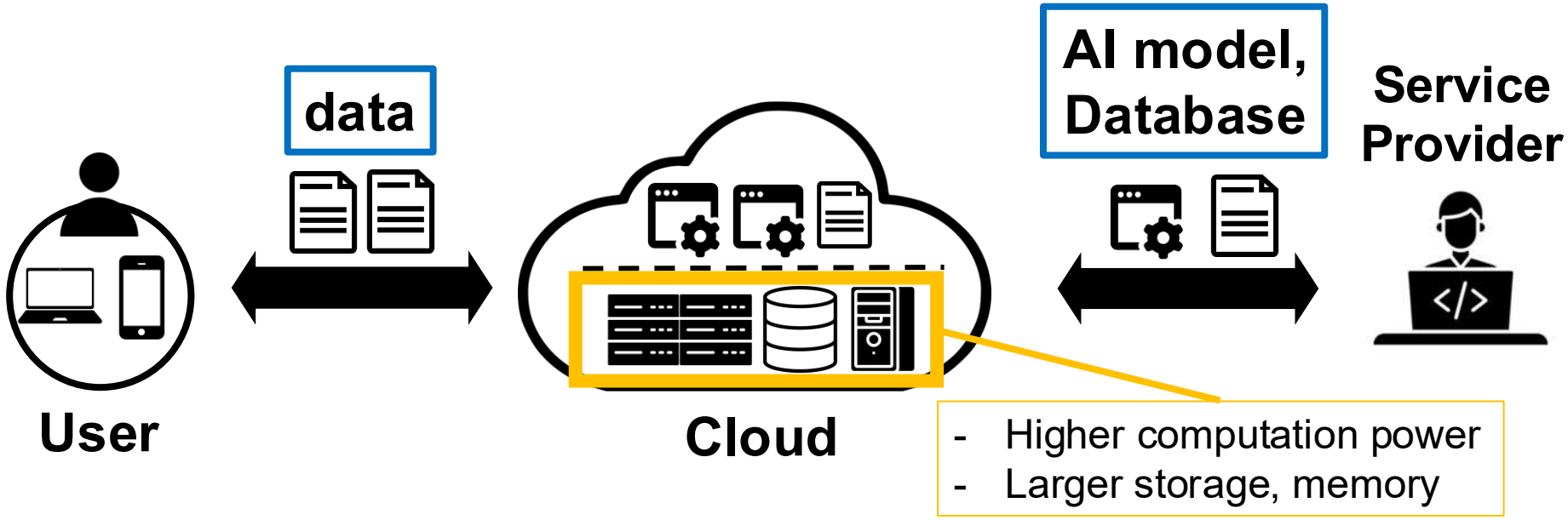
Confidential Computing

- Cloud and remote computing



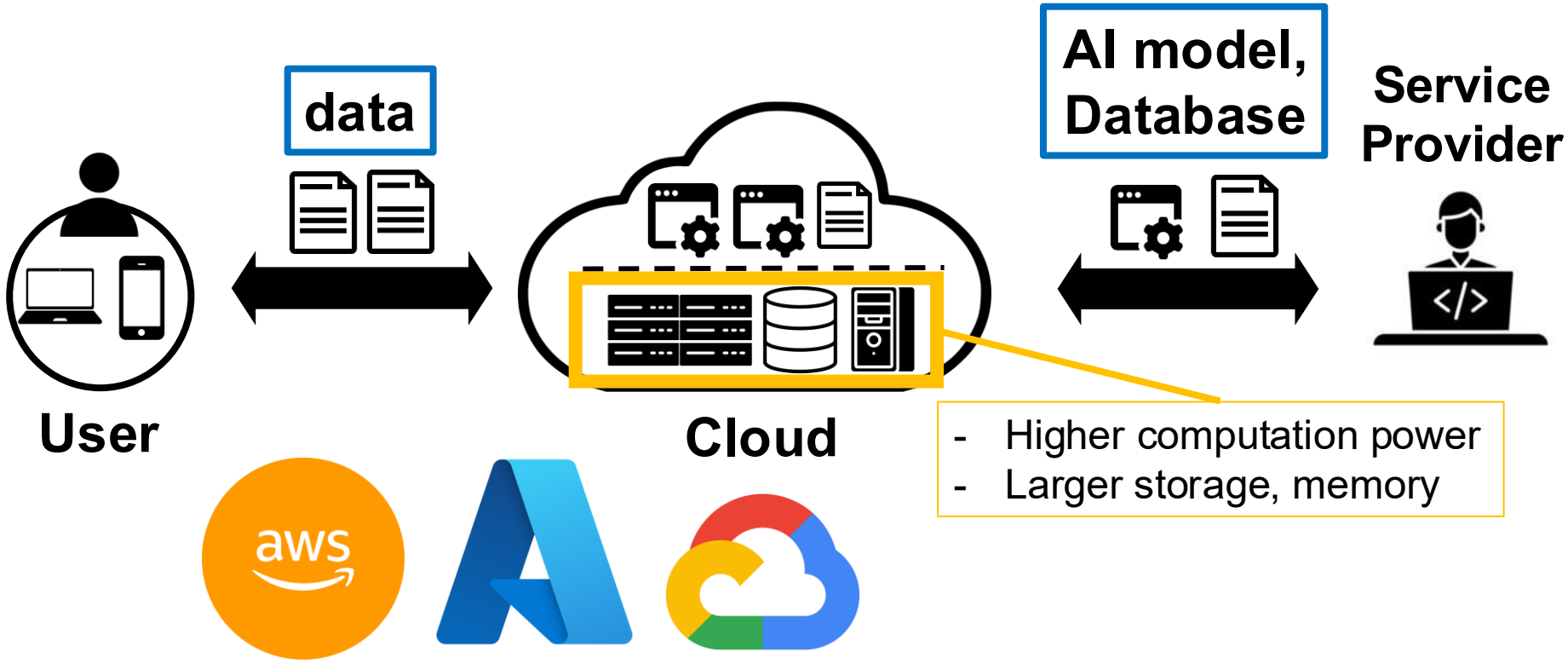
Confidential Computing

- Cloud and remote computing



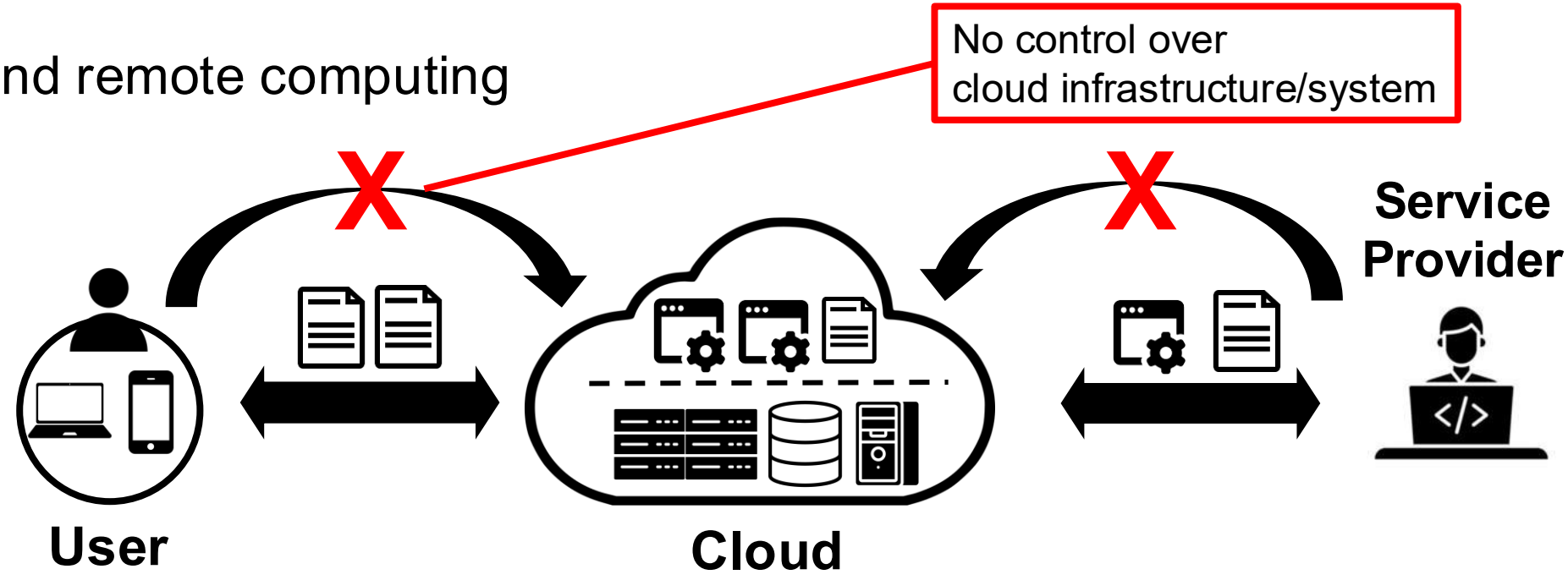
Confidential Computing

- Cloud and remote computing



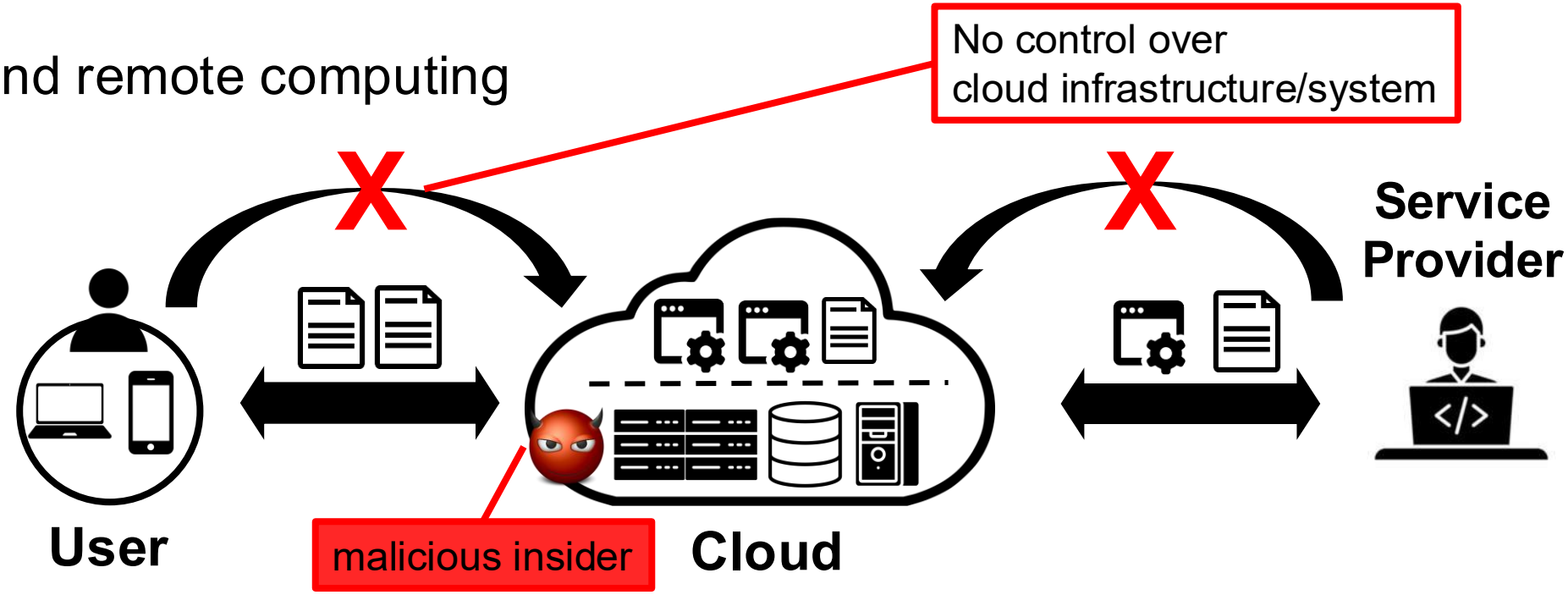
Confidential Computing

- Cloud and remote computing



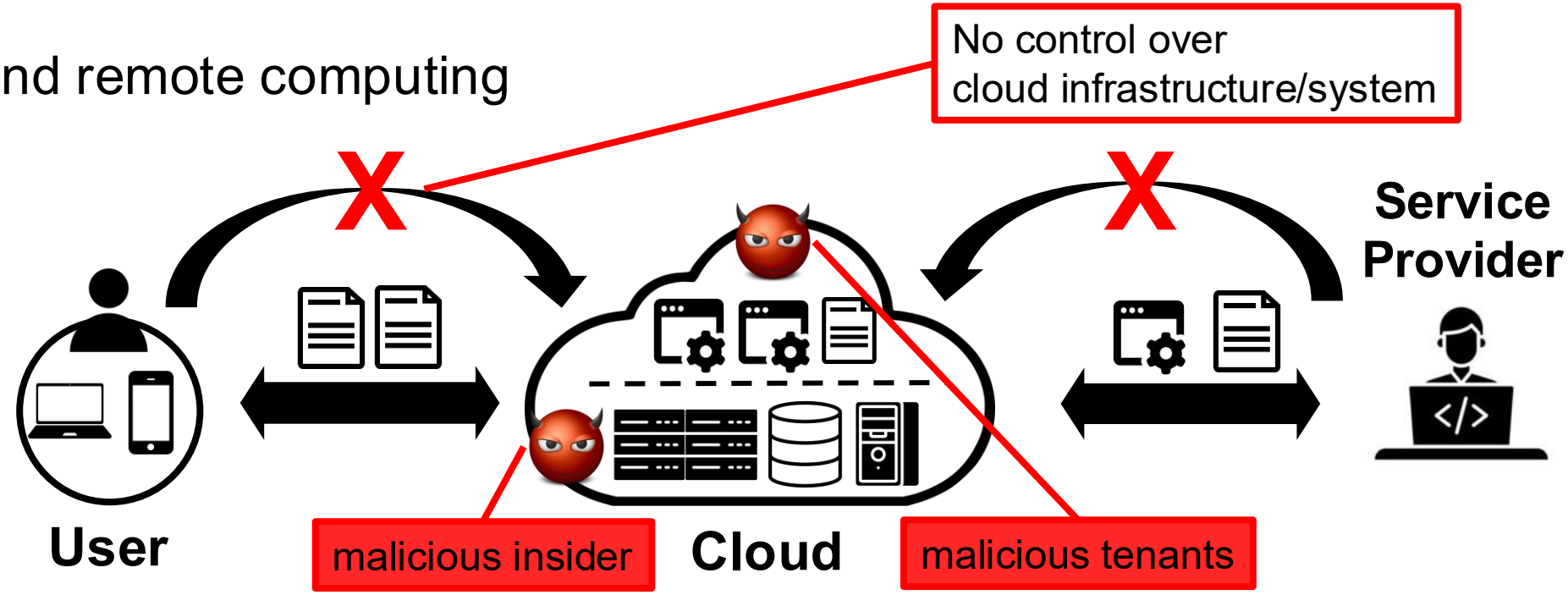
Confidential Computing

- Cloud and remote computing



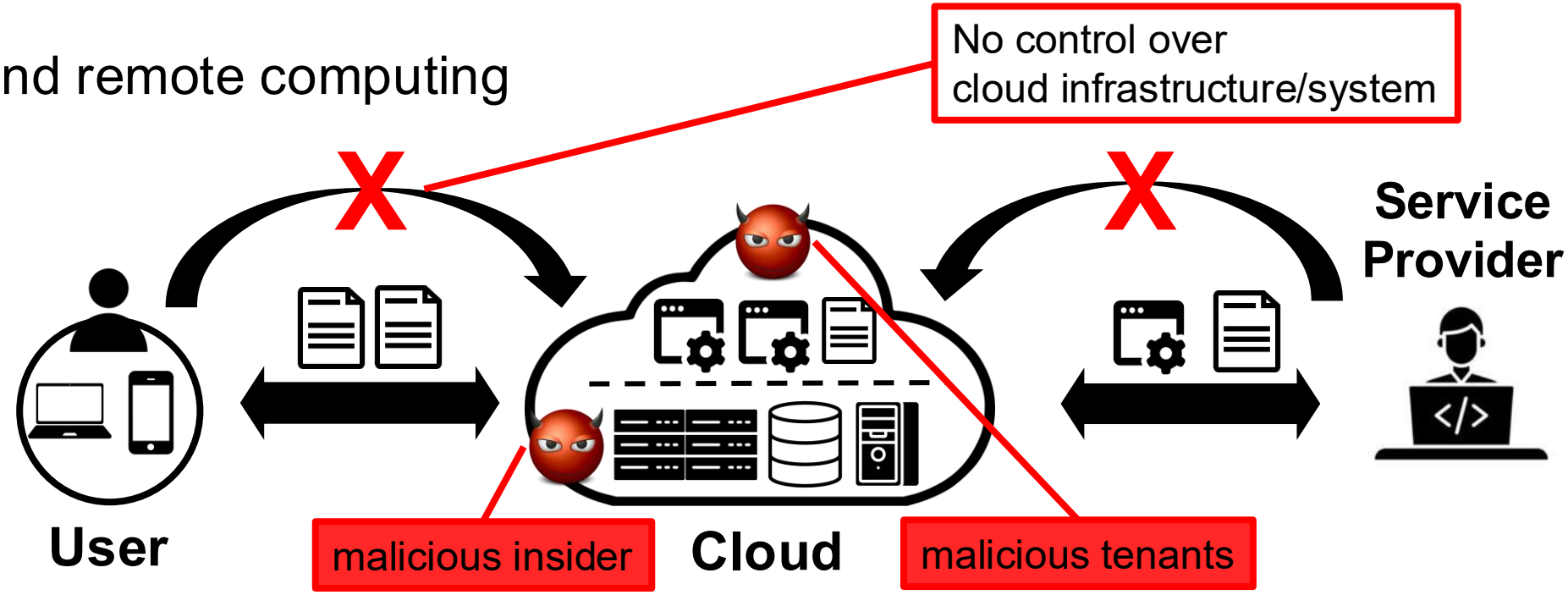
Confidential Computing

- Cloud and remote computing



Confidential Computing

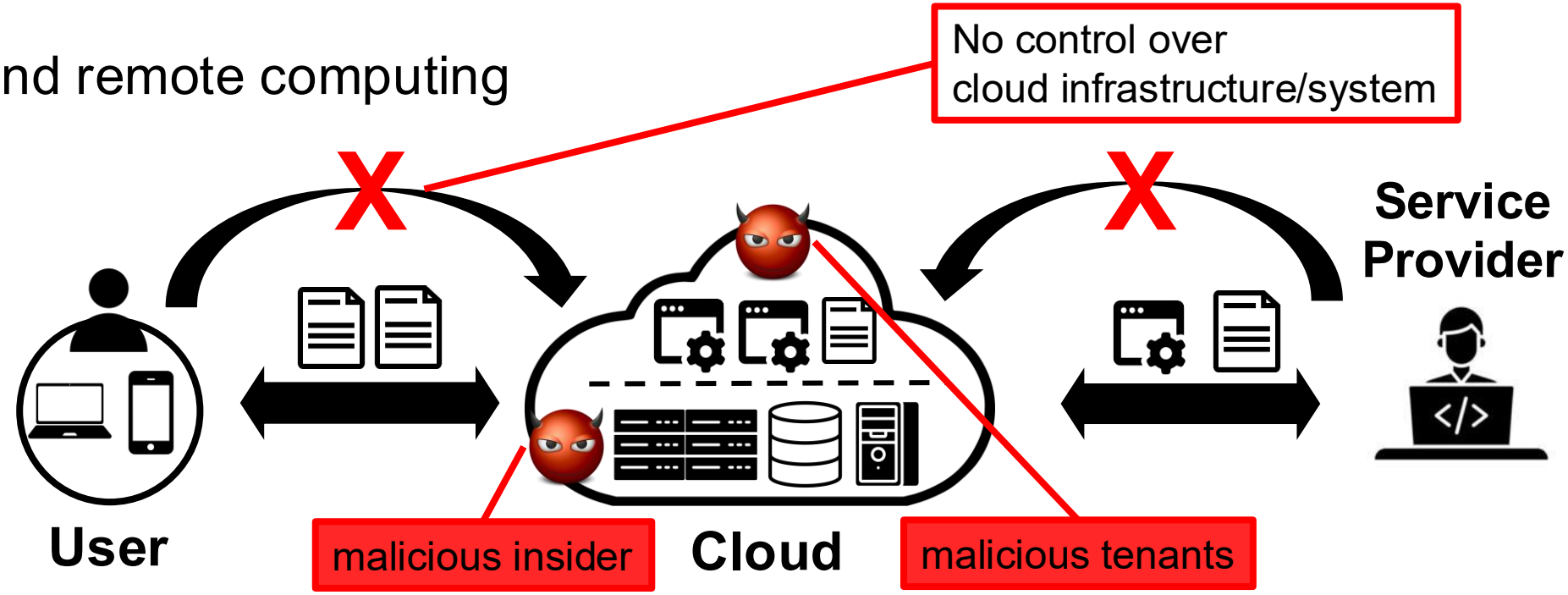
- Cloud and remote computing



How do we protect data/code in the cloud from adversarial insiders?

Confidential Computing

- Cloud and remote computing

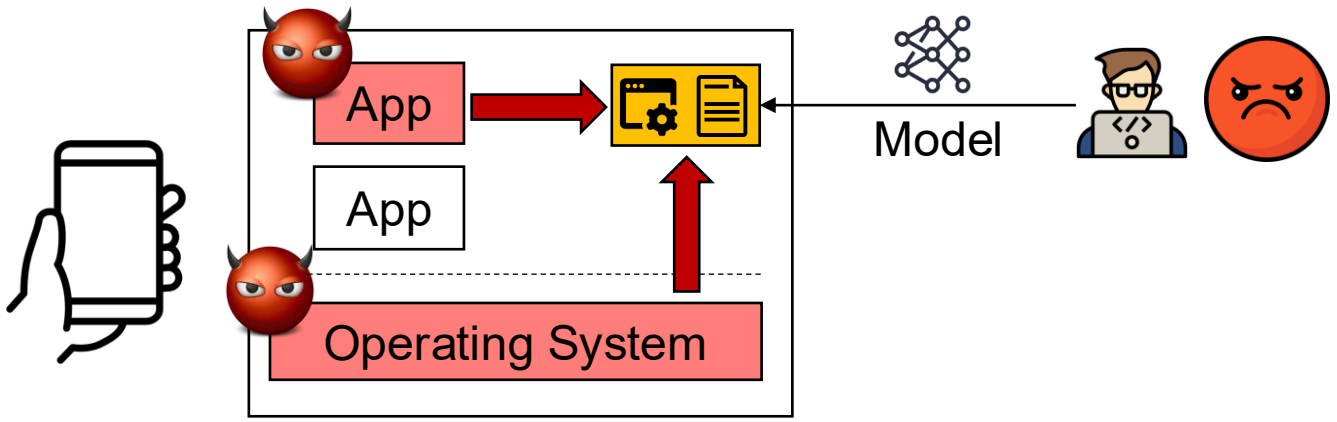


How do we protect data/code in the cloud from adversarial insiders?

Confidential Computing

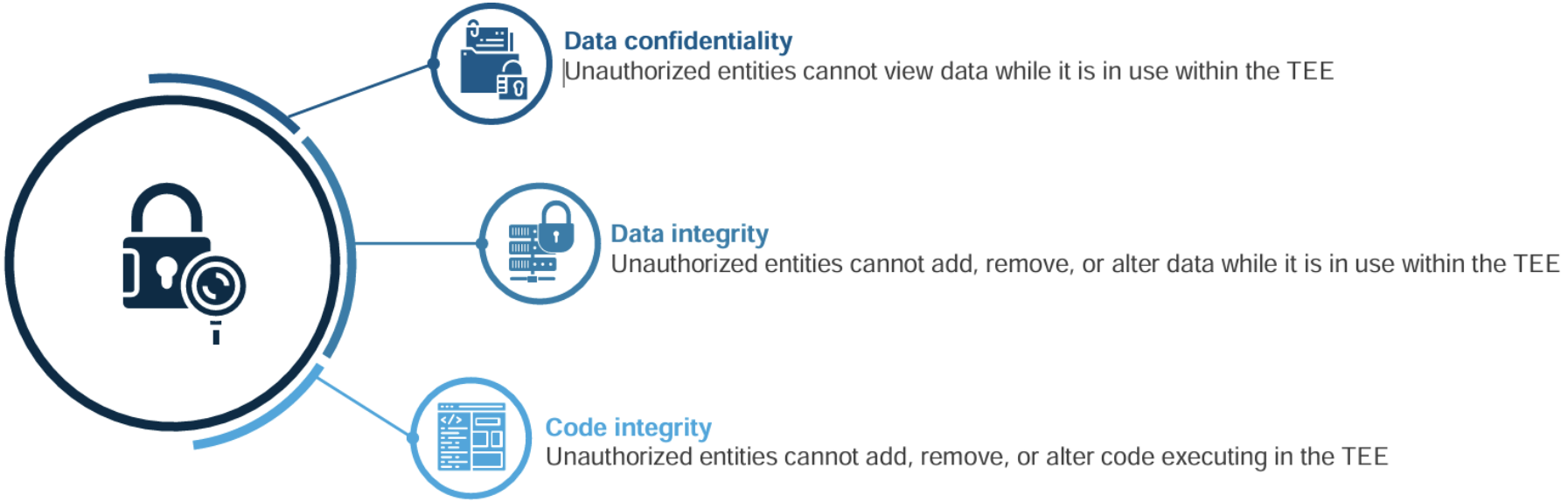
Confidential Computing on ARM

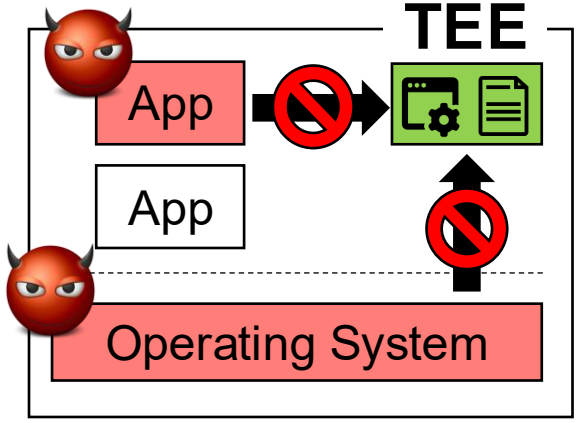
- Why do we need confidential computing on ARM?
  - High-performance ARM processors
  - Integrated System-on-Chip (SoC) accelerators (NPUs, GPUs, …)
  - Rather than sending sensitive information, download the service on to the device!
    - E.g., ***on-device AI***
    - What about proprietary AI model privacy? ➔ confidential computing on ARM



Trusted Execution Environment (TEE)

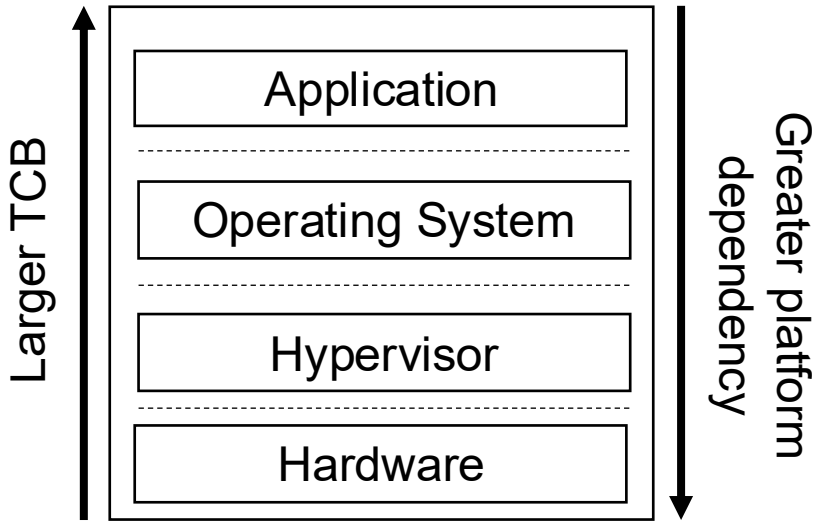
- Enabling technology for confidential computing
- Isolated execution environment
- Guarantees ***confidentiality*** and ***integrity*** of data/code inside TEE





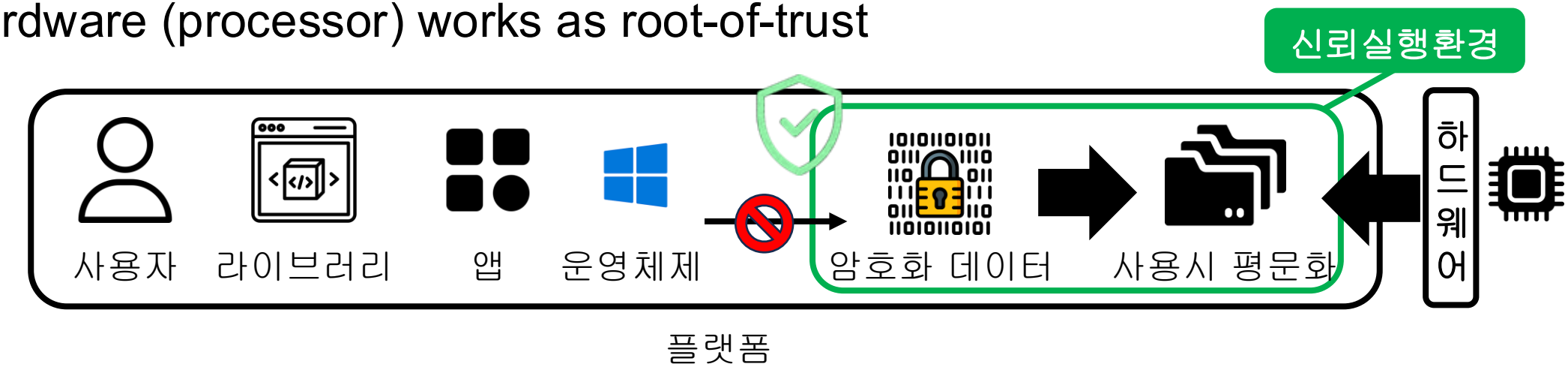
Types of TEEs

- Software-based TEEs
  - Trusts privileged software (e.g., hypervisor)
  - No (or minimal) hardware dependency
  - Large trusted computing base (TCB)
- Hardware-based TEEs
  - Only need to trust the device hardware
  - Hardware must support TEE extension
  - Better performance and security
  - Intel TDX, AMD SEV, ARM CCA…



Hardware-assisted TEEs

- Hardware (processor) works as root-of-trust



- Supported by major hardware vendors
  - Intel, ARM, AMD, (NVIDIA for GPUs)…
  - Continuous support for TEEs
    - Intel SGX ➔ Intel TDX
    - AMD SEV ➔ AMD SEV-SNP

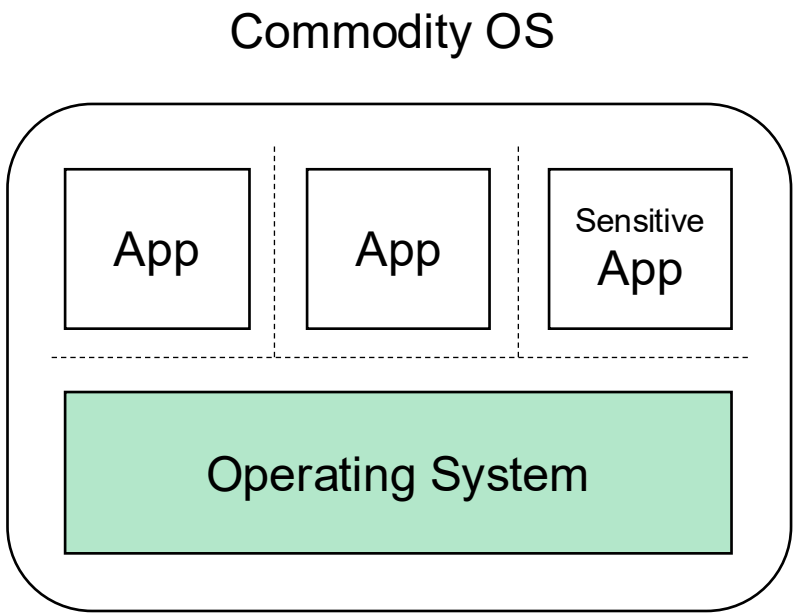
TrustZone® System Security by ARM

intel SGX

ARM TrustZone

- Hardware-assisted TEE for ARM platforms
- Already widely used for security-critical applications (banking, payment, DRM)

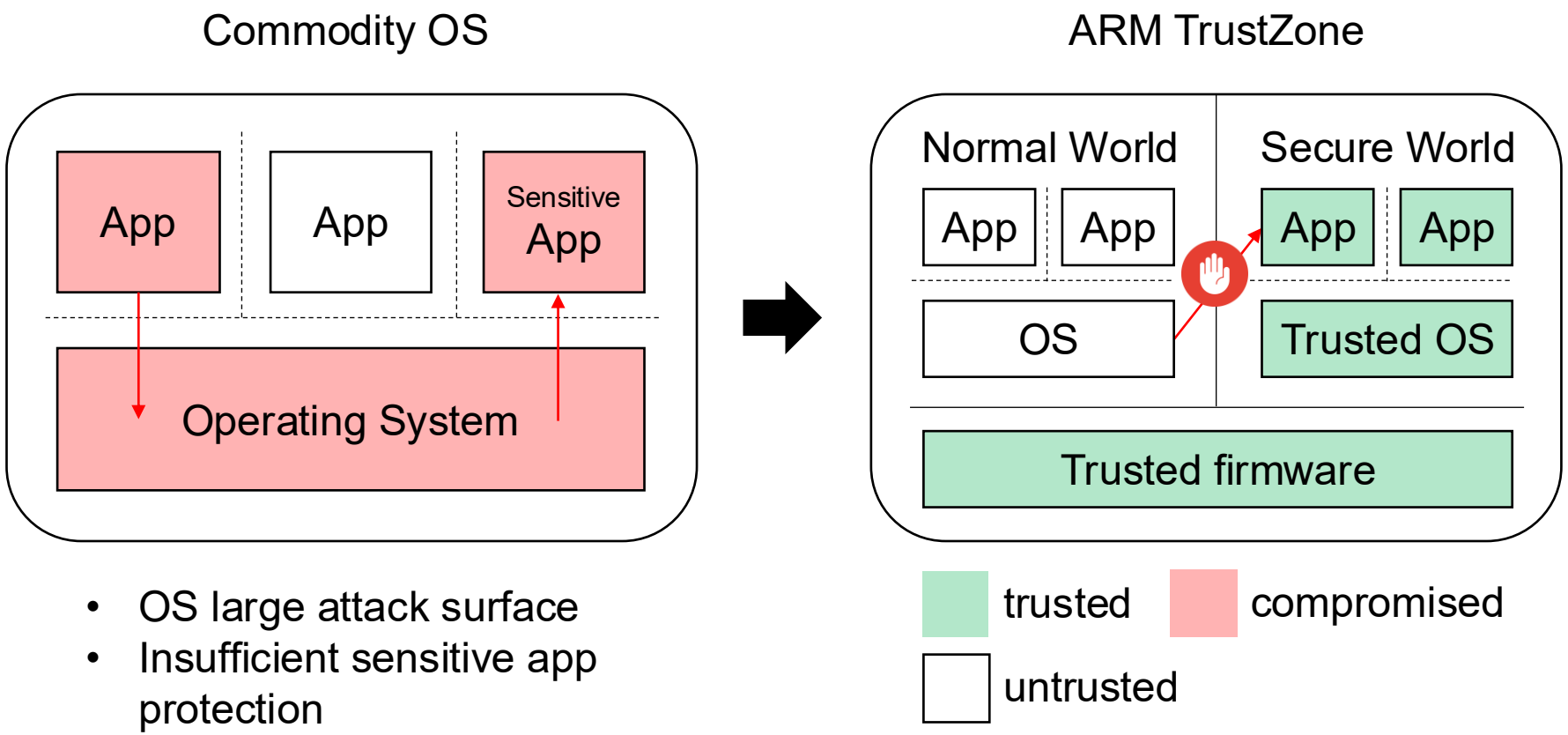
ARM TrustZone Overview



- OS large attack surface
- Insufficient sensitive app protection

trusted
compromised
untrusted

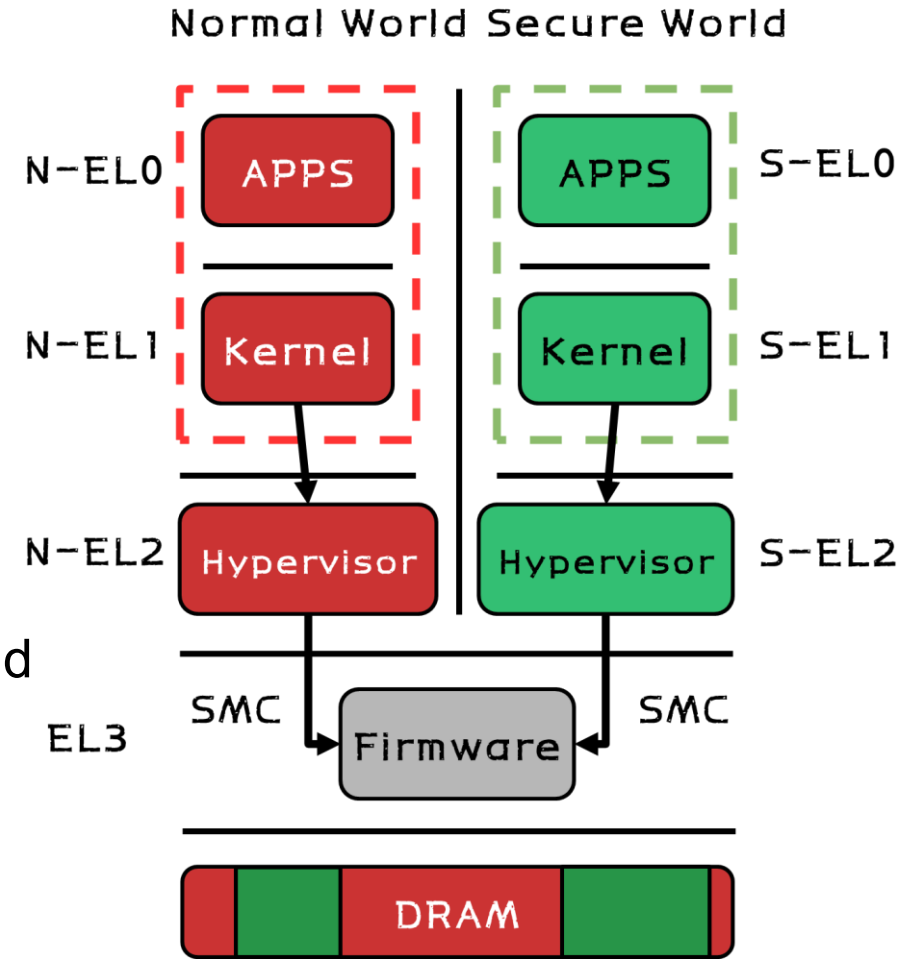
ARM TrustZone Overview



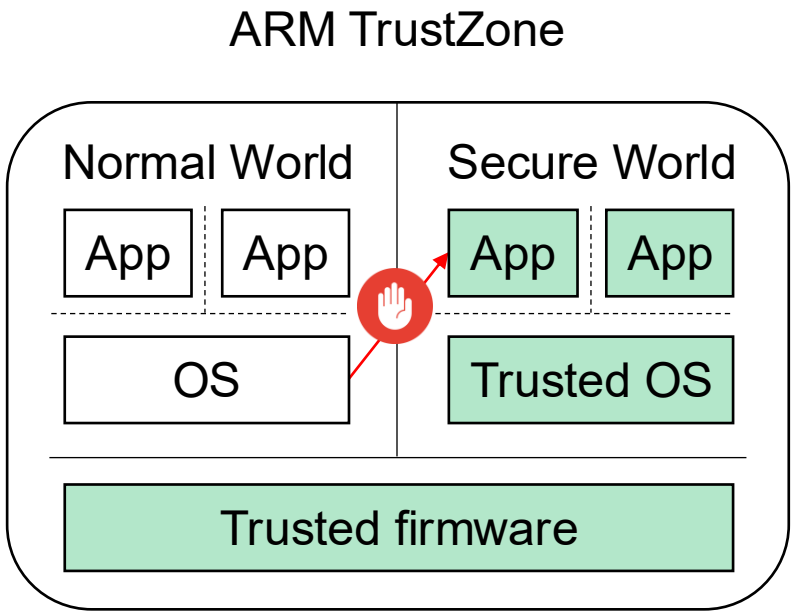
- OS large attack surface
- Insufficient sensitive app protection

ARM TrustZone Architecture

- Exception Levels (ELs)
  - Processor-level management of privilege separation between system components
  - Higher EL is given with more control over resources and can manage the behavior of lower ELs
- Trustzone Address Space Controller (TZASC)
  - HW memory controller provided by Trustzone
  - Restrict access to secure world memory from normal world
- Secure Monitor Call (SMC)
  - Communication between normal and secure world
  - Traps execution to firmware in EL3



Problem with ARM TrustZone



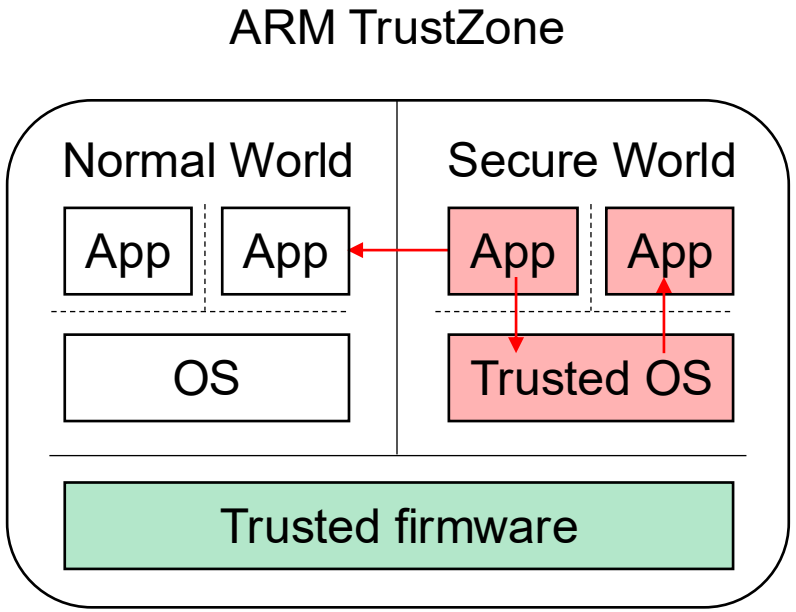
trusted

untrusted

compromised

Problem with ARM TrustZone

Need to go through (very) strict verification by device vendors (e.g., Samsung, Google)



Difficult to be used for as-a-service applications

- Trust OS still a large attack surface
- High costs to protect apps with TrustZone
- Single security domain

trusted

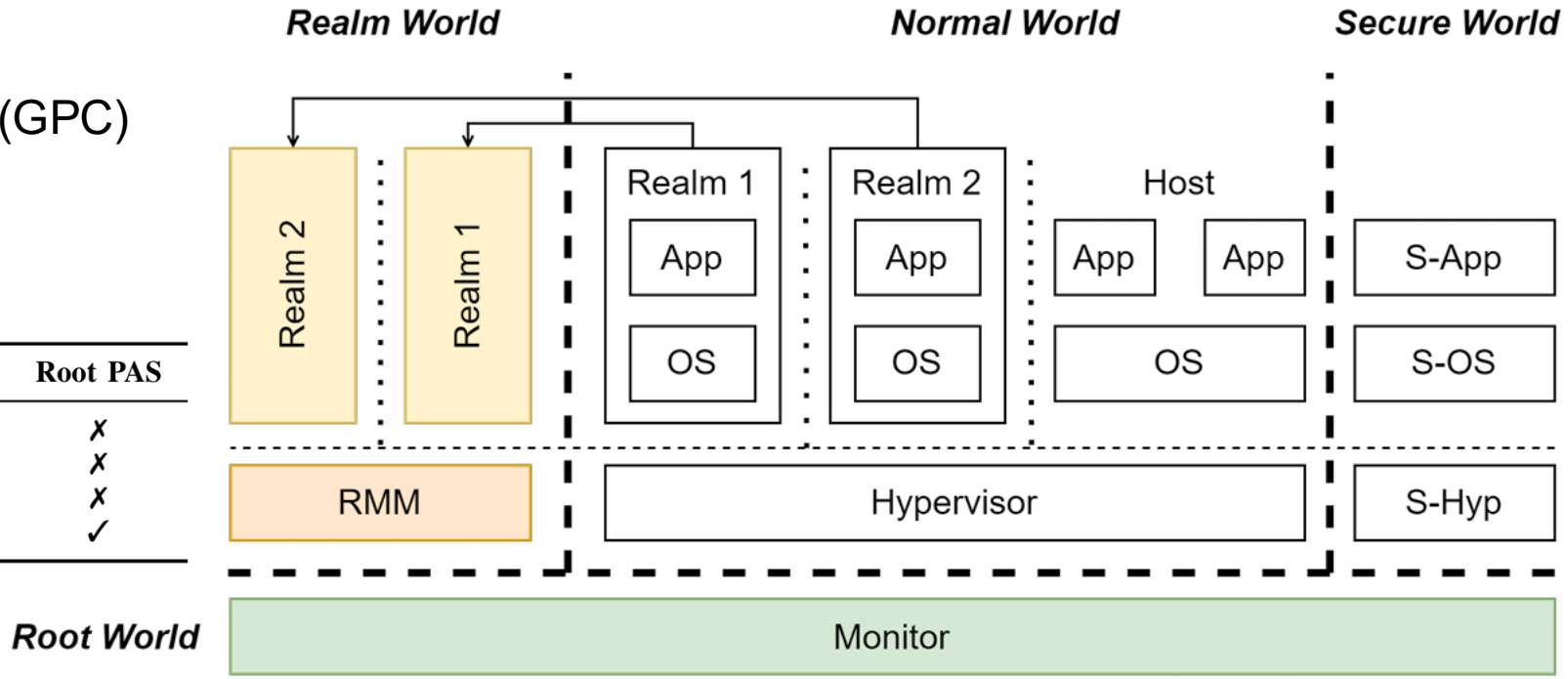
untrusted

compromised

ARM Confidential Compute Architecture (CCA)

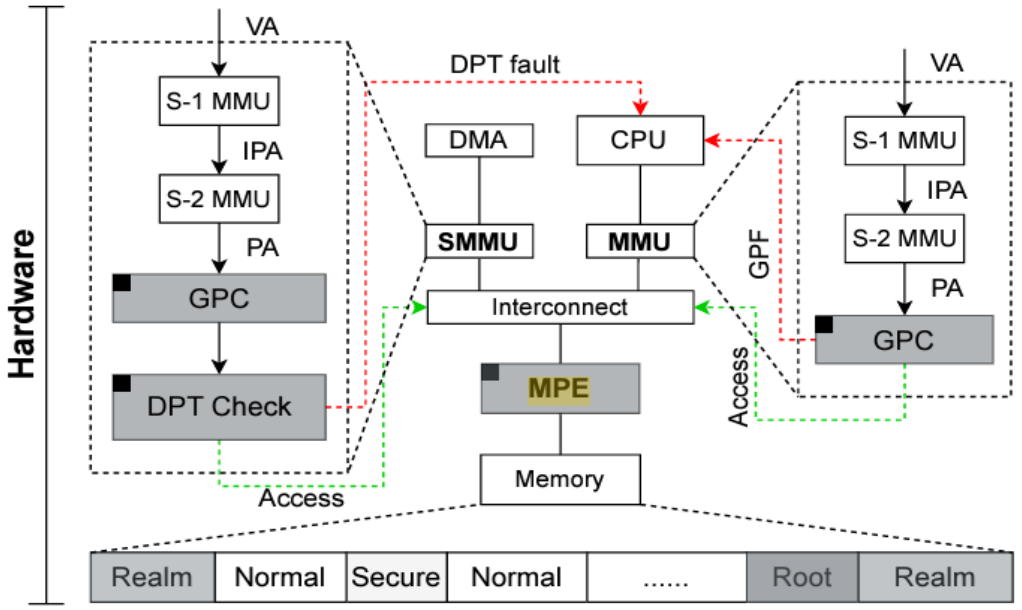
- Provide confidential computing for next generation (ARMv9.2-A) ARM devices
  - New security state for confidential computing: ***Realm world***
  - Hardware-isolated ***Root world***
  - New security supports…
    - Granule Protection Check (GPC)
    - Memory encryption

Security State	Normal PAS	Secure PAS	Realm PAS	Root PAS
Normal	✓	✗	✗	✗
Secure	✓	✓	✗	✗
Realm	✓	✗	✓	✗
Root	✓	✓	✓	✓



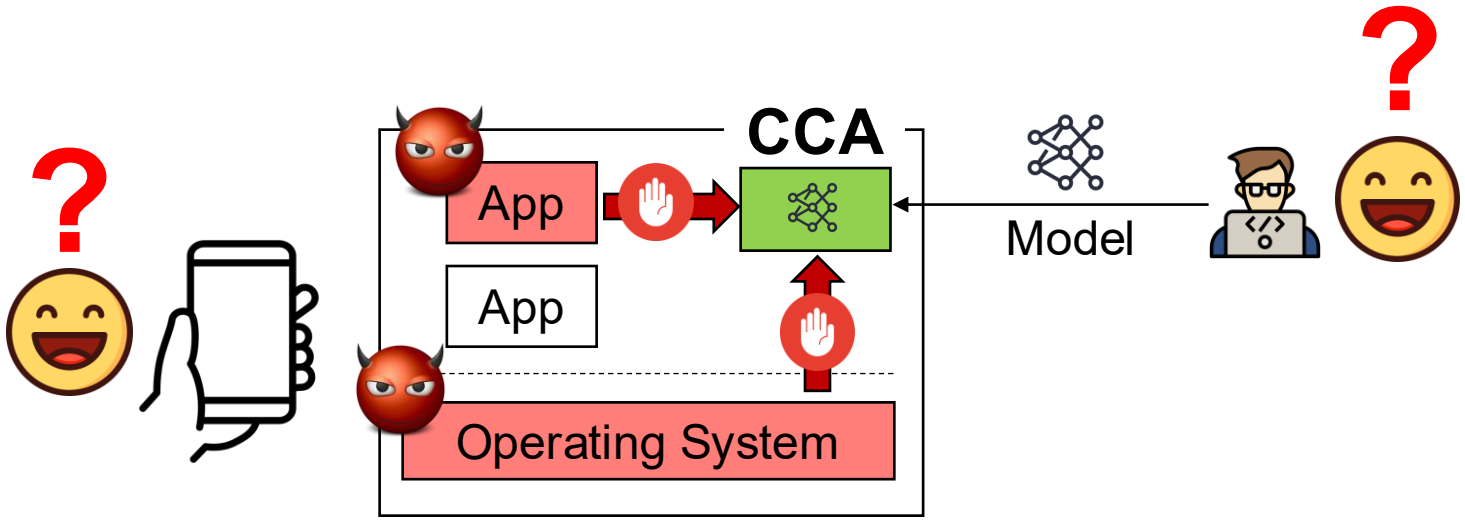
ARM CCA Architecture

- Granule Protection Table (GPT)
  - Data structure that manages physical address space (PAS) that each memory page belongs to (i.e., GPT entry = { physical address, (normal/secure/root/Realm) }
  - Hardware MMU checks the validity of the access against GPT (Granule Protection Check)
  - Can only be modified by the EL3 monitor
- Memory Protection Engine (MPE)
  - Support memory encryption per PAS
  - On-going development for per-Realm VM encryption
- Device Protection Table (DPT)
  - Extension for per-device memory access control
  - Only authorized devices can access memory



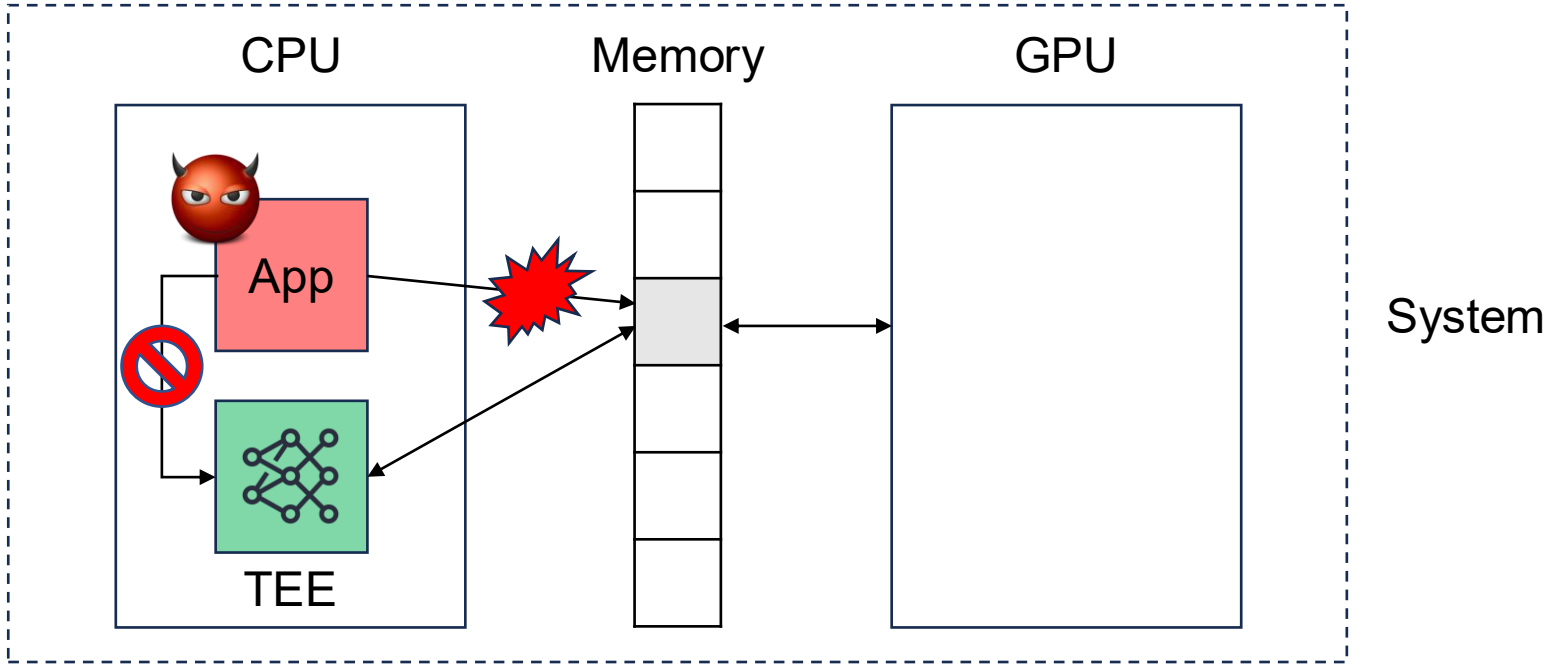
Open Problems

- Is CCA enough to provide confidential on-device AI?



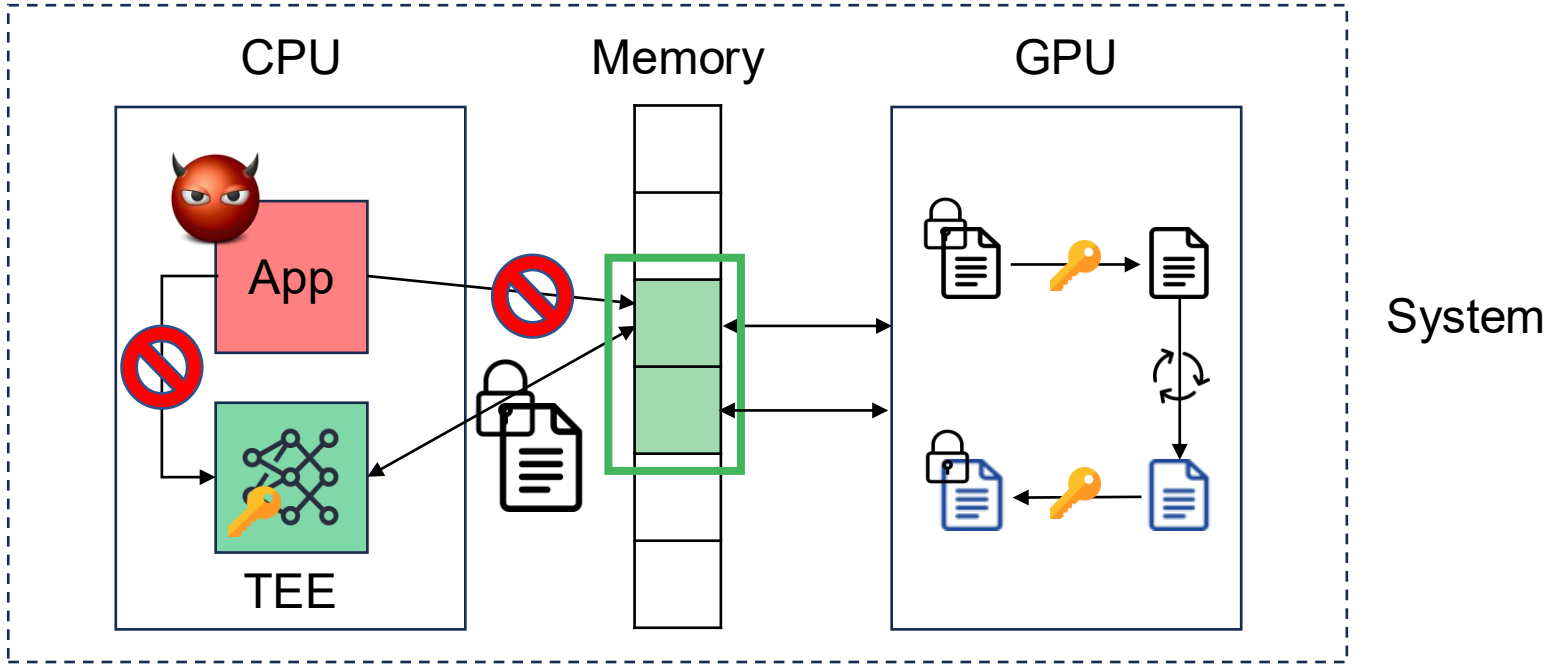
Open Problem – Extending TEE to GPU

- GPU TEE
  - GPU is essential for fast AI service
  - CCA does ***not*** protect memory used by Realm VMs



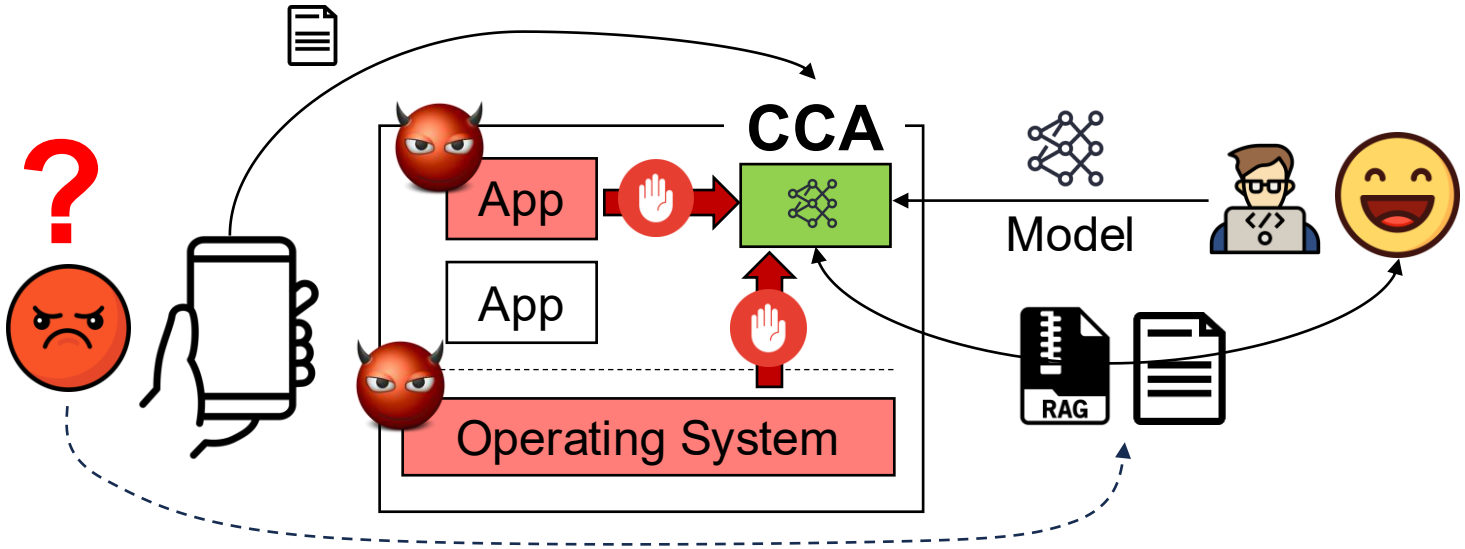
Open Problem – Extending TEE to GPU

- GPU TEE ➔ only available for Intel, AMD platforms (NVIDIA H100/200)
  - GPU is essential for fast AI service
  - CCA does ***not*** protect memory used by Realm VMs



Open Problem – RAG for On-device AI

- SOTA on-device AI services leverage Retrieval-Augmented Generation (RAG) for better services
  - Need communication with the external cloud
  - Data provider (device user) cannot examine what is being executed inside TEE
    ➔ Cannot verify whether their (private) input data is leaked during RAG communication



Outline

- Hardware-assisted solutions for building secure systems
  - Focus on latest extensions on ARM – MTE, PAC, BTI, CCA

current done

Goal	Problem	HW feature
Memory safety	Spatial memory safety	Memory Tagging Extension (MTE)
	Temporal memory safety	Memory Tagging Extension (MTE)
Control-flow integrity	Coarse-grained CFI	Branch Target Indirection (BTI)
	Fine-grained CFI	Pointer Authentication Code (PAC)
Compartmentalization	Kernel driver isolation	Memory Tagging Extension (MTE)
Least privilege	Access control on shared memory	Memory Tagging Extension (MTE)
Confidential computing	Trusted execution environment	TrustZone, CC Architecture (CCA)

Hardware-assist is fast, but not a panacea

Take-aways – on system security

- System security is hard. You need to know…
  - Low-level programming (e.g., C/C++)
  - Assembly language
  - Compiler (e.g., LLVM)
  - Operating system (memory management, process/thread, …)
  - Hardware (cache, DMA, MMU, …)

Need to know the whole system ☹

- But is useful, regardless of the environment
  - Systems vary in detail, but is consistent on overarching principles
  - Drones, autonomous vehicles, mobile, cloud, datacenters, robots…

Take-aways – on hardware-assisted security

- Software-only security solutions are ***slow***.
  - Look out for new hardware extensions for both performance and security
  - We investigated…
    - ARM Memory Tagging Extension
    - ARM Branch Target Identification
    - ARM Pointer Authentication to solve classic system security problems
- Hardware-assisted TEE is a key enabler for confidential computing
  - Stay tuned for advances in TEE extensions from major hardware vendors
  - Look out for new problems/challenges before deciding to use TEE

Thank you

- Questions?