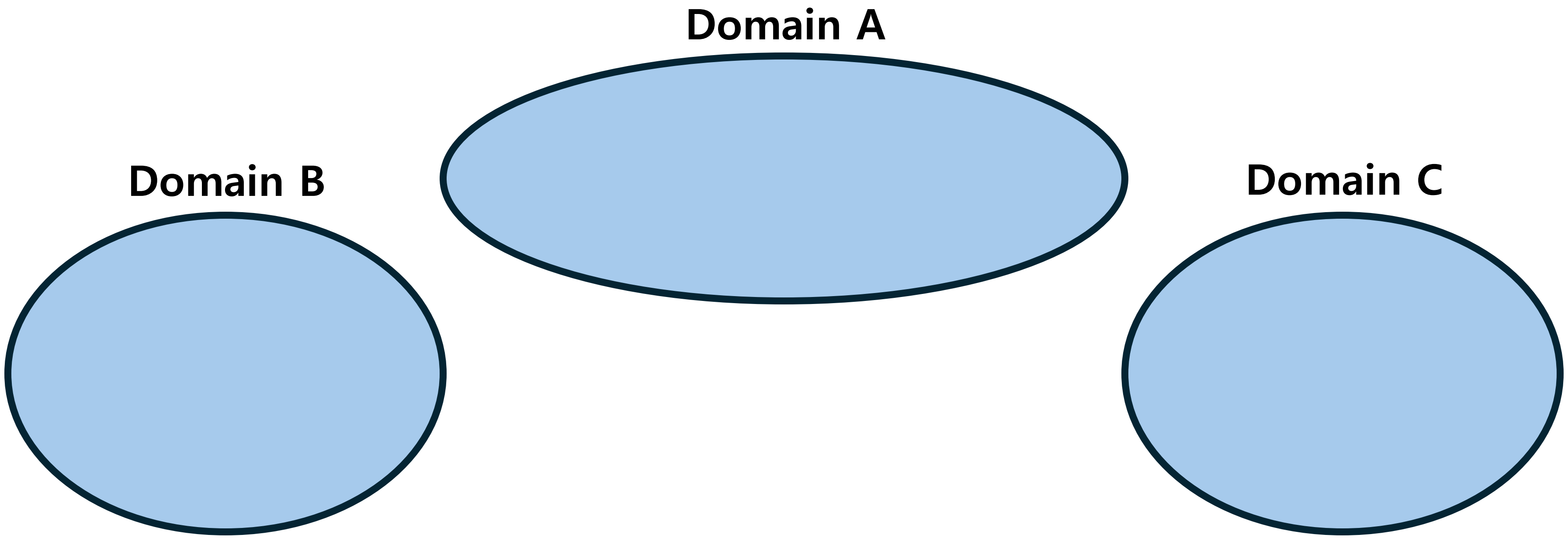
BASTAG: Byte-level Access Control on Shared Memory using ARM Memory Tagging Extension

Junseung You[1], Jiwon Seo[2], Kyeongryong Lee[1], Yeongpil Cho[3], Yunheung Paek[1]

[1]Seoul National University, [2]Dankook University, [3]Hanyang University

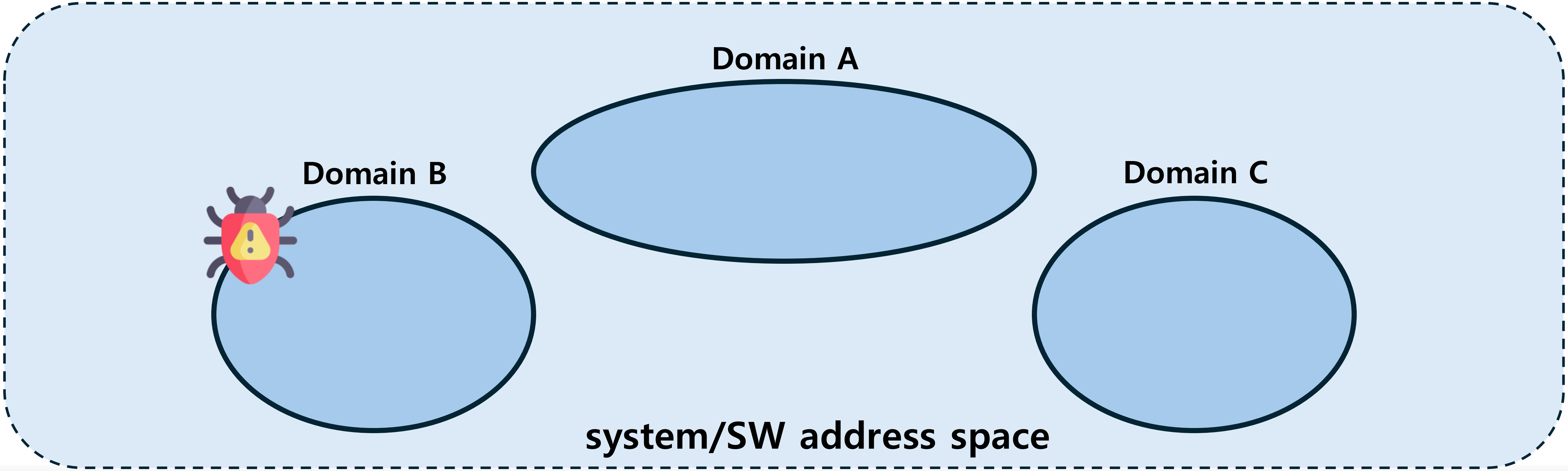
Modularization

- Modern software is modularized into distinct components
  - Libraries, modules, threads, etc.



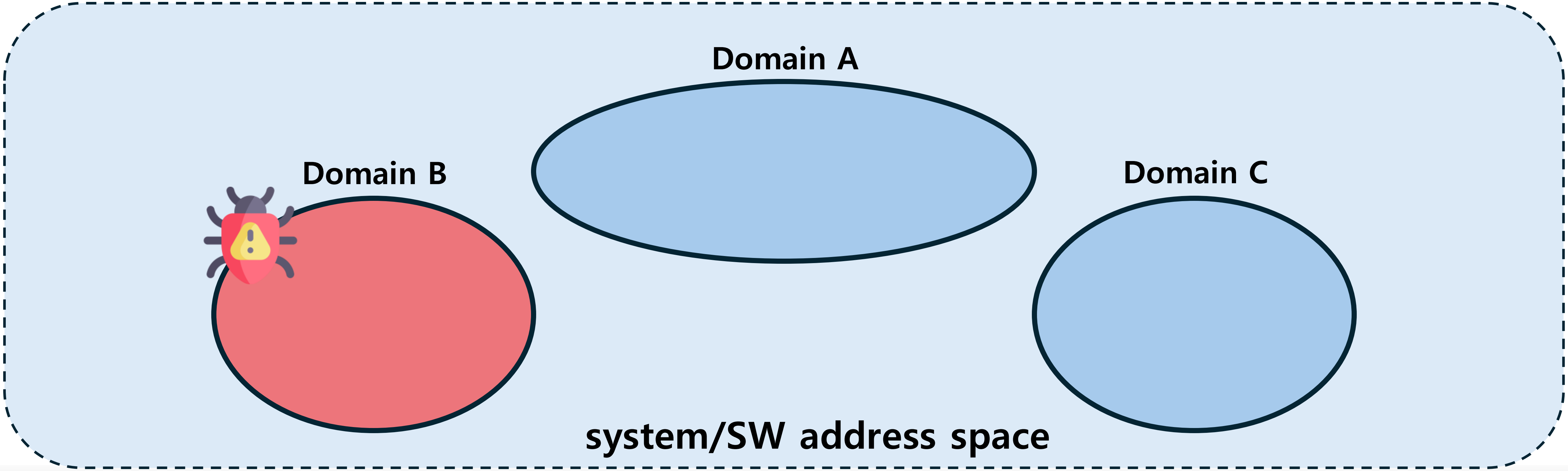
Need for Memory Protection

- Modularized components run in the same address space
- Vulnerability in one can compromise the whole system/software



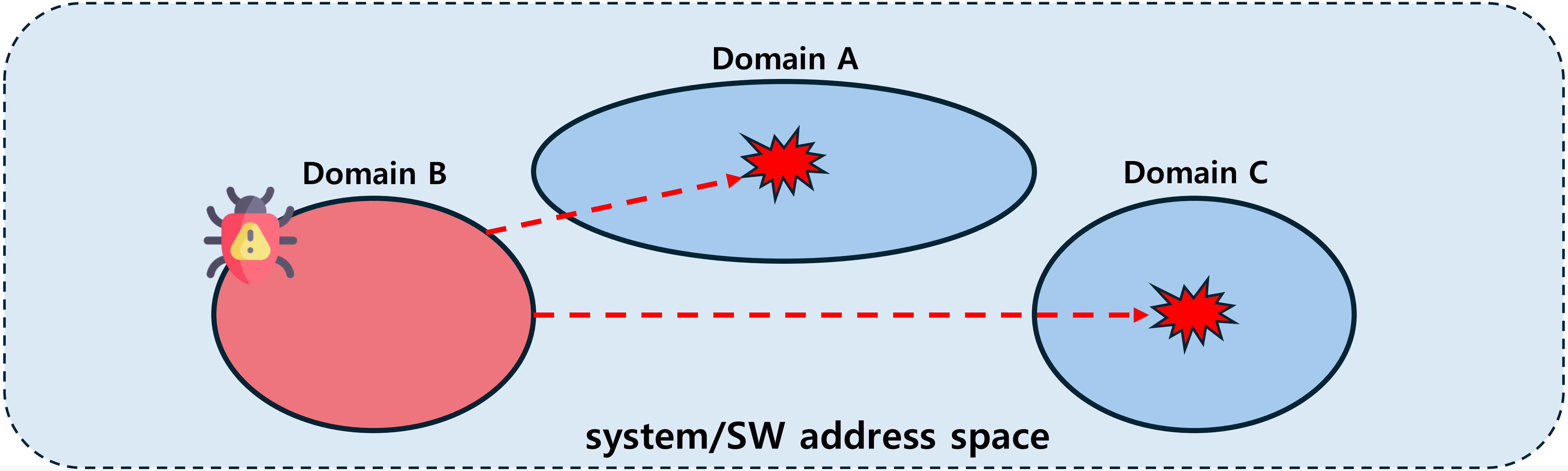
Need for Memory Protection

- Modularized components run in the same address space
- Vulnerability in one can compromise the whole system/software



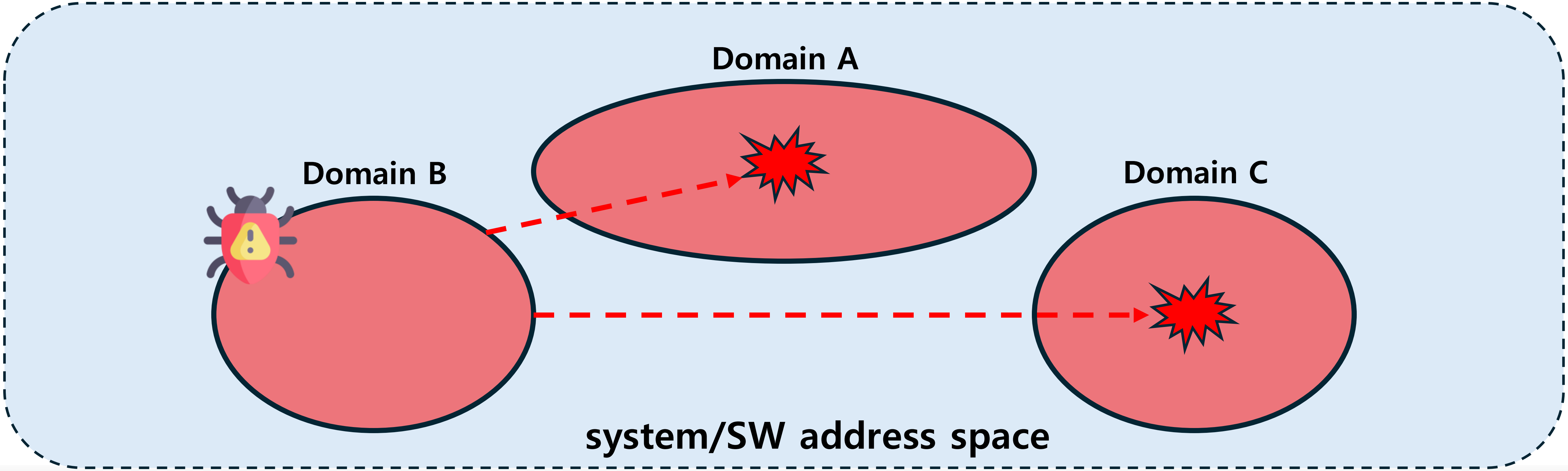
Need for Memory Protection

- Modularized components run in the same address space
- Vulnerability in one can compromise the whole system/software



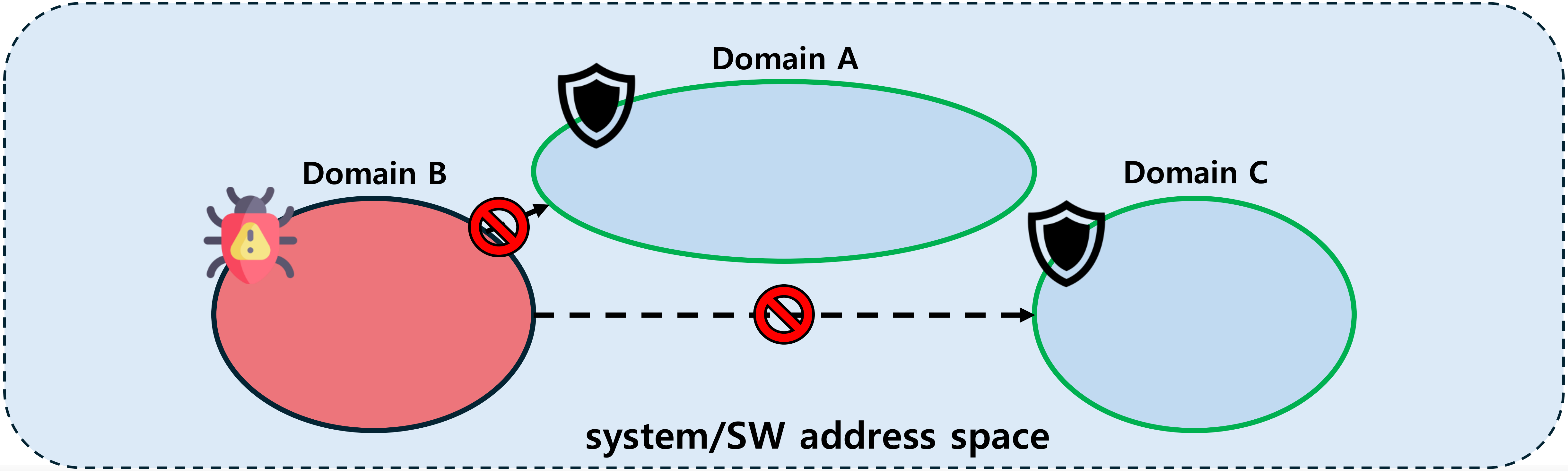
Need for Memory Protection

- Modularized components run in the same address space
- Vulnerability in one can compromise the whole system/software



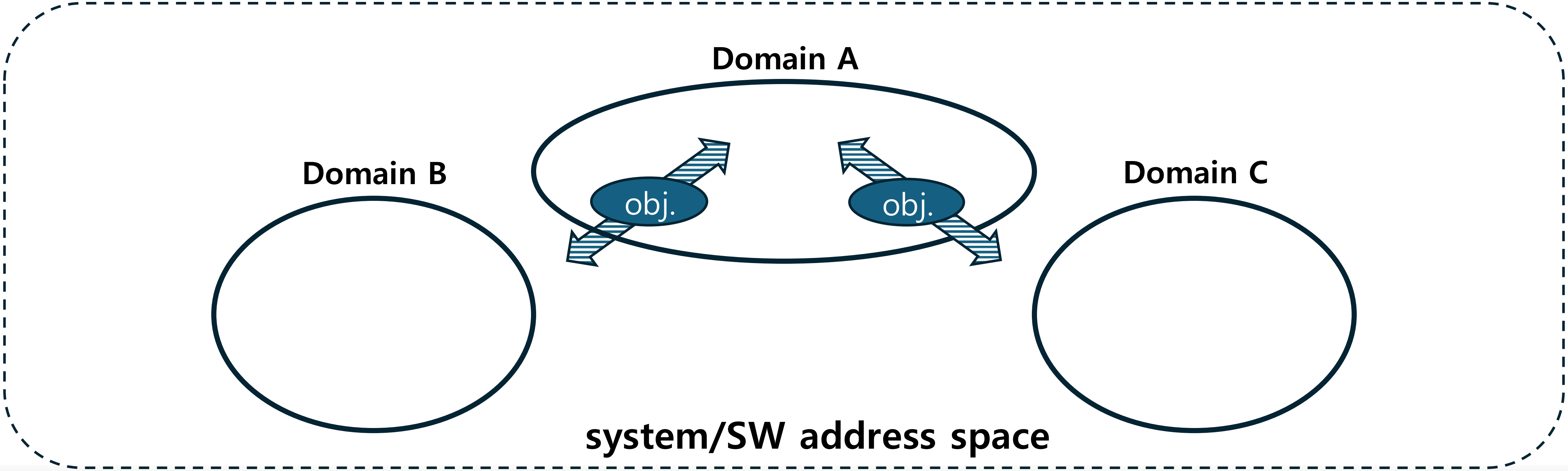
Protection for Private Memory

- Exclusive access to domain-private memory
- Addressed by numerous isolation techniques (e.g., SFI)



Shared Memory

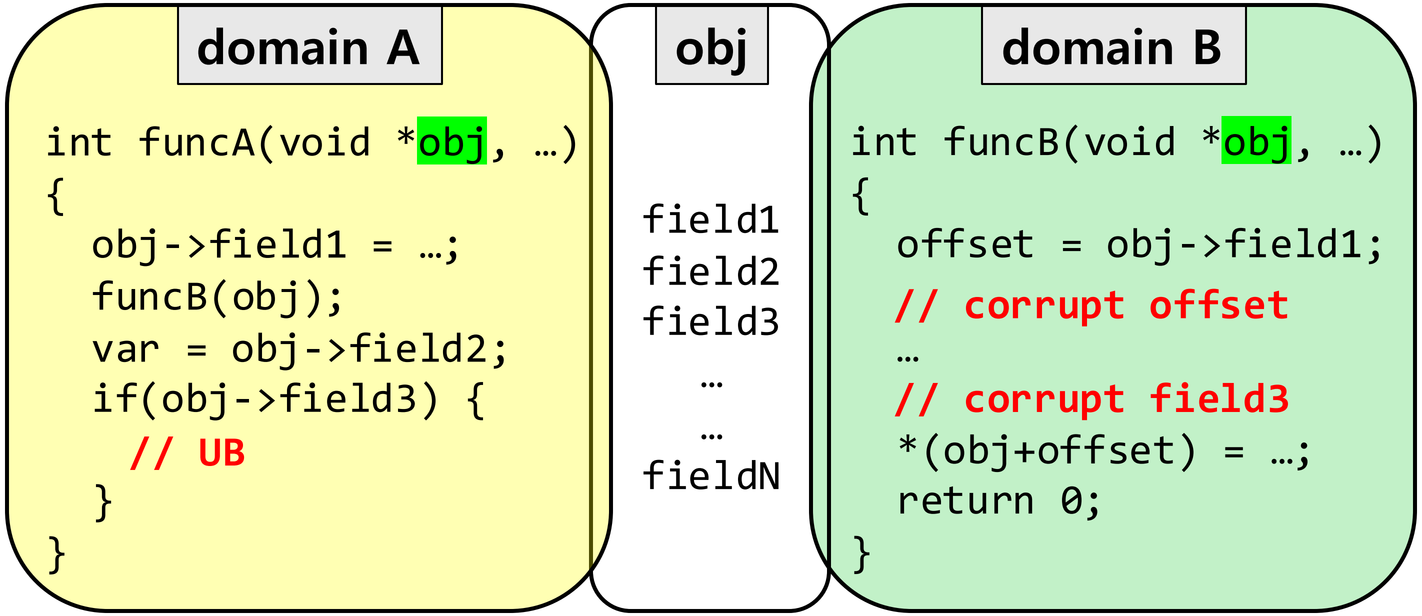
- Necessary for domain interaction and communication



Protection for 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



domain A

```
int funcA(void *obj, …)
{
  obj->field1 = …;
  funcB(obj);
  var = obj->field2;
  if(obj->field3) {
    // UB
  }
}
```

obj

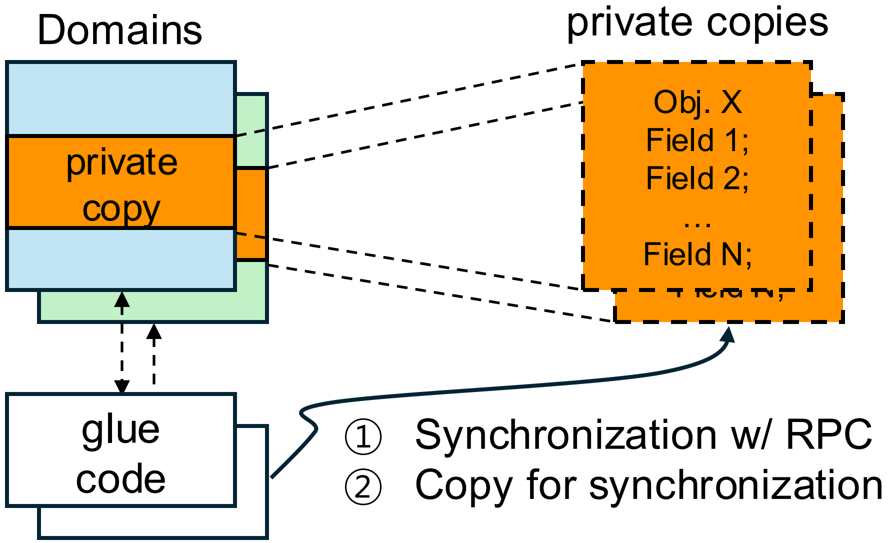
```
field1
field2
field3
  …
  …
fieldN
```

domain B

```
int funcB(void *obj, …)
{
  offset = obj->field1;
  // corrupt offset
  …
  // corrupt field3
  *(obj+offset) = …;
  return 0;
}
```

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

Solutions for Shared Memory Protection

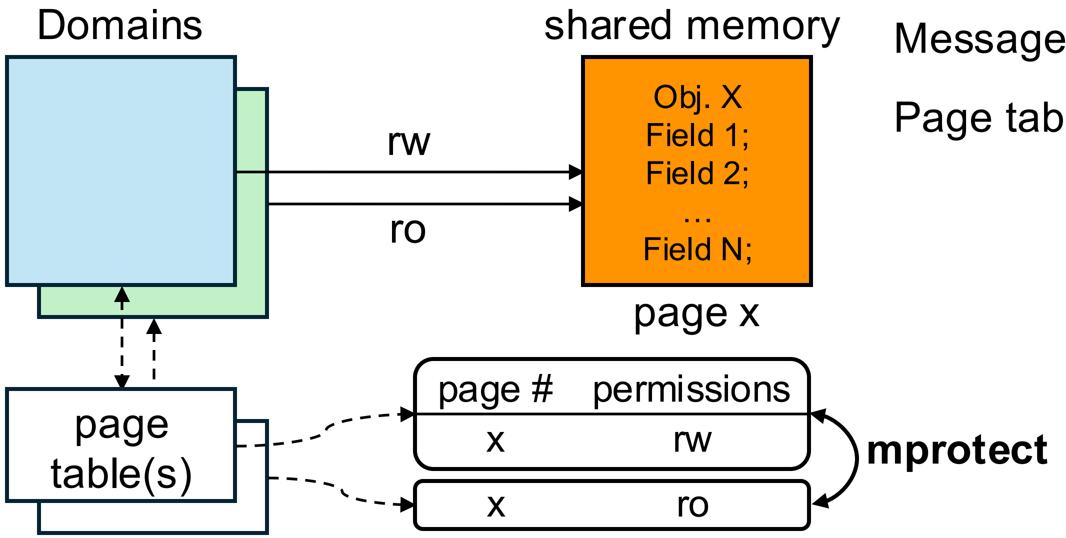


① Synchronization w/ RPC
② Copy for synchronization

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

Slow synchronization

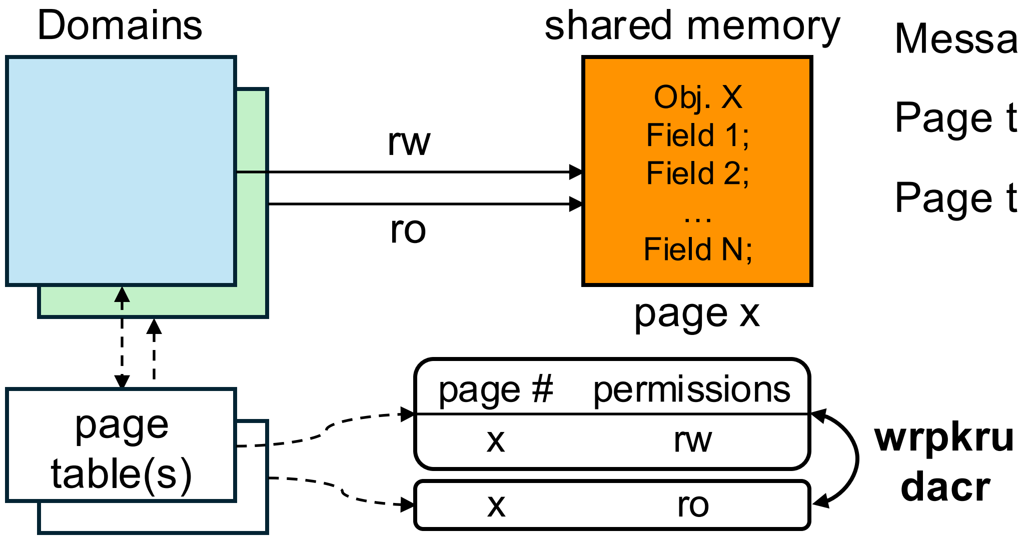
Solutions for Shared Memory Protection



Coarse-grained level of control

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

Solutions for Shared Memory Protection

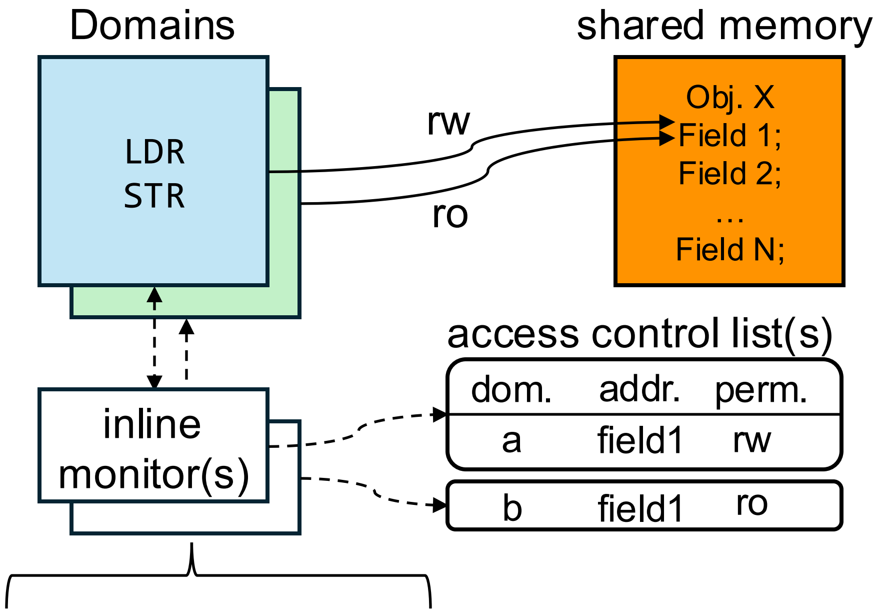


page #	permissions
x	rw
x	ro

Coarse-grained
level of 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**

Solutions for Shared Memory Protection



dom.	addr.	perm.
a	field1	rw
b	field1	ro

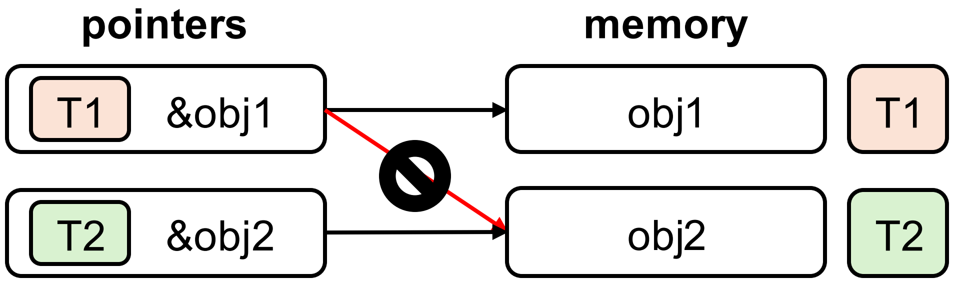
① Calculate index
② Lookup metadata
③ Check validity

➔ **Slow** checks

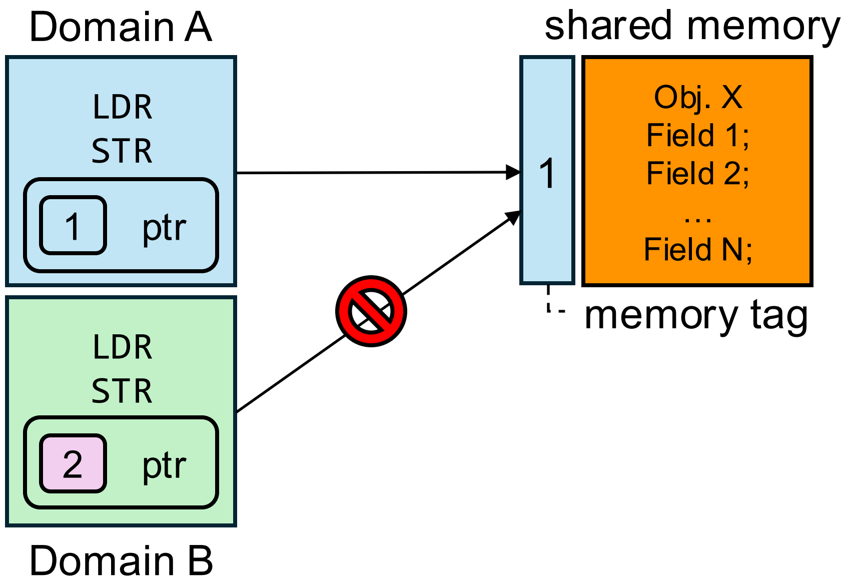
Mechanism	Permissions	Performance	Level of Control
Message-based	Per-domain/multiple	**Slow**	byte
Page table-based	Per-domain/multiple	**Slow** (w/ SW)	**page**
Page table-based	Per-domain/multiple	Fast (w/ HW)	**page**
Inline monitors	Per-domain/multiple	**Slow**	byte
Inline monitors	Per-domain/multiple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

ARM Memory Tagging Extension

- Introduced in ARMv8.5-A architecture
- Deployed in COTS devices (Google Pixel 8, Samsung Galaxy)
- 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



Solutions for Shared Memory Protection



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

16B granularity
Binary access permission
Single domain access control

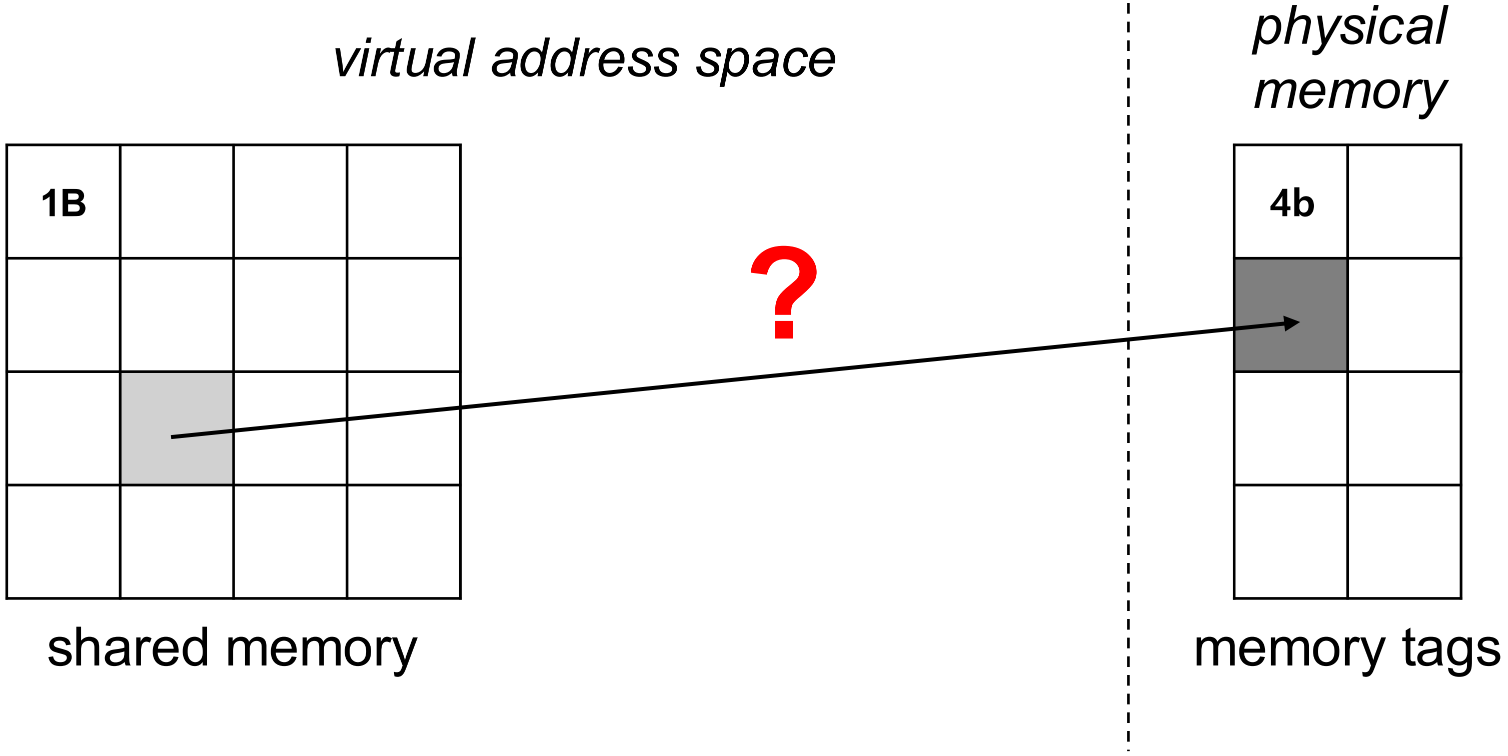
How can we leverage MTE for **efficient multi-domain, multi-policy byte-level** access control?

BASTAG

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

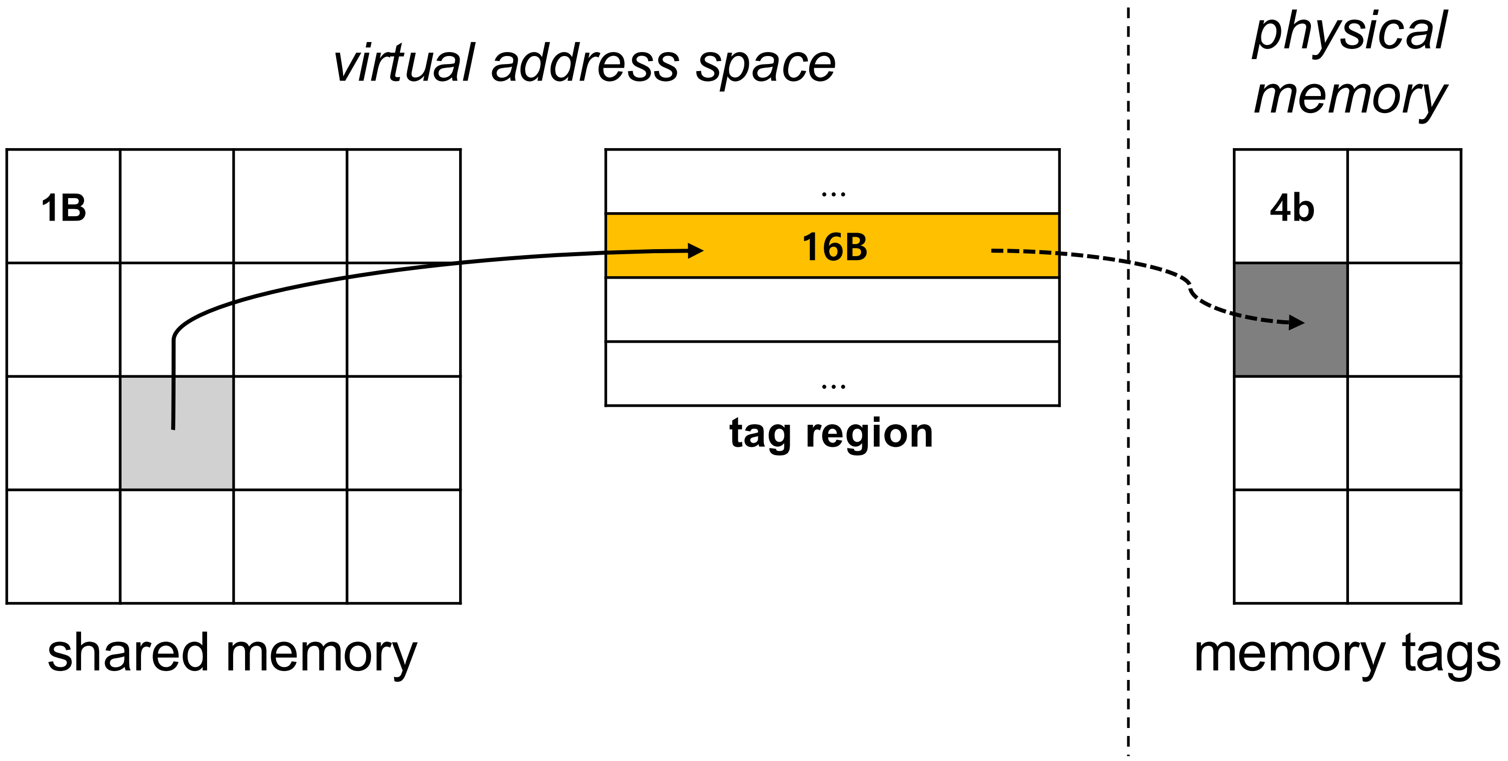
Shadow Memory Tagging

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



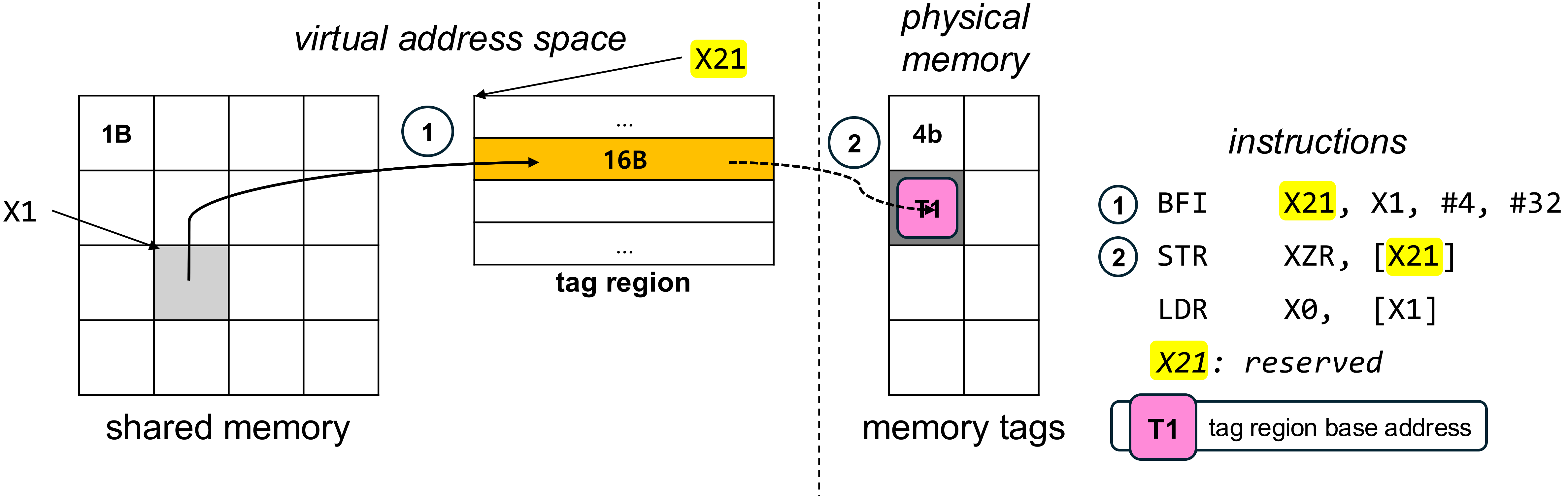
Shadow Memory Tagging

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



Shadow Memory Tagging

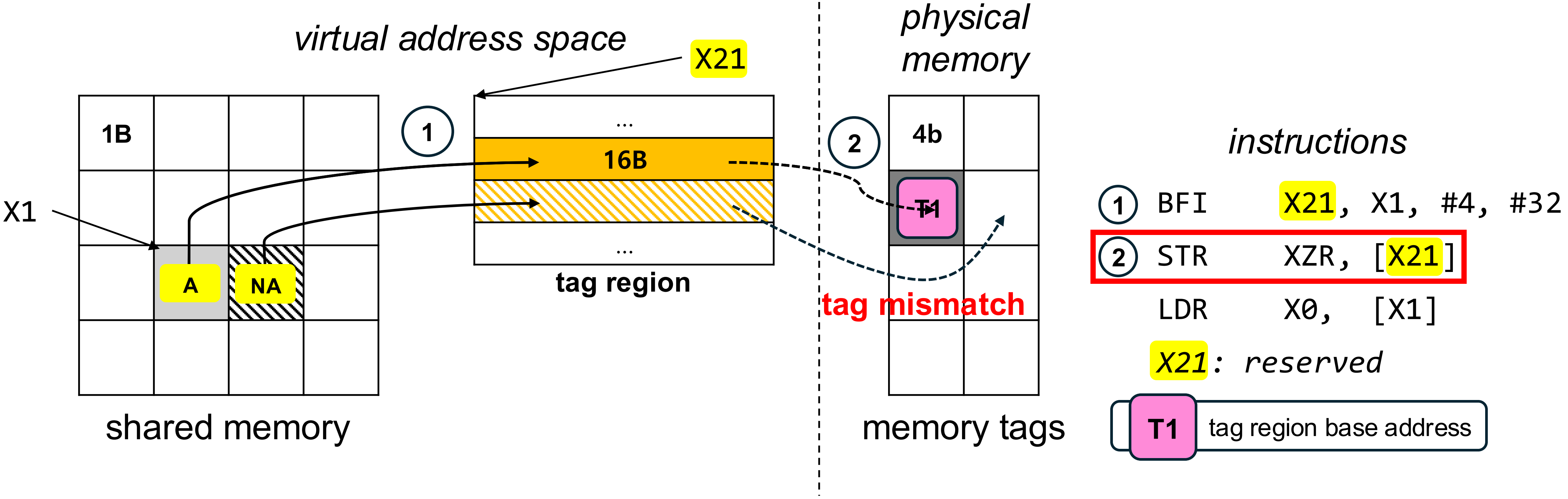
- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



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

Shadow Memory Tagging

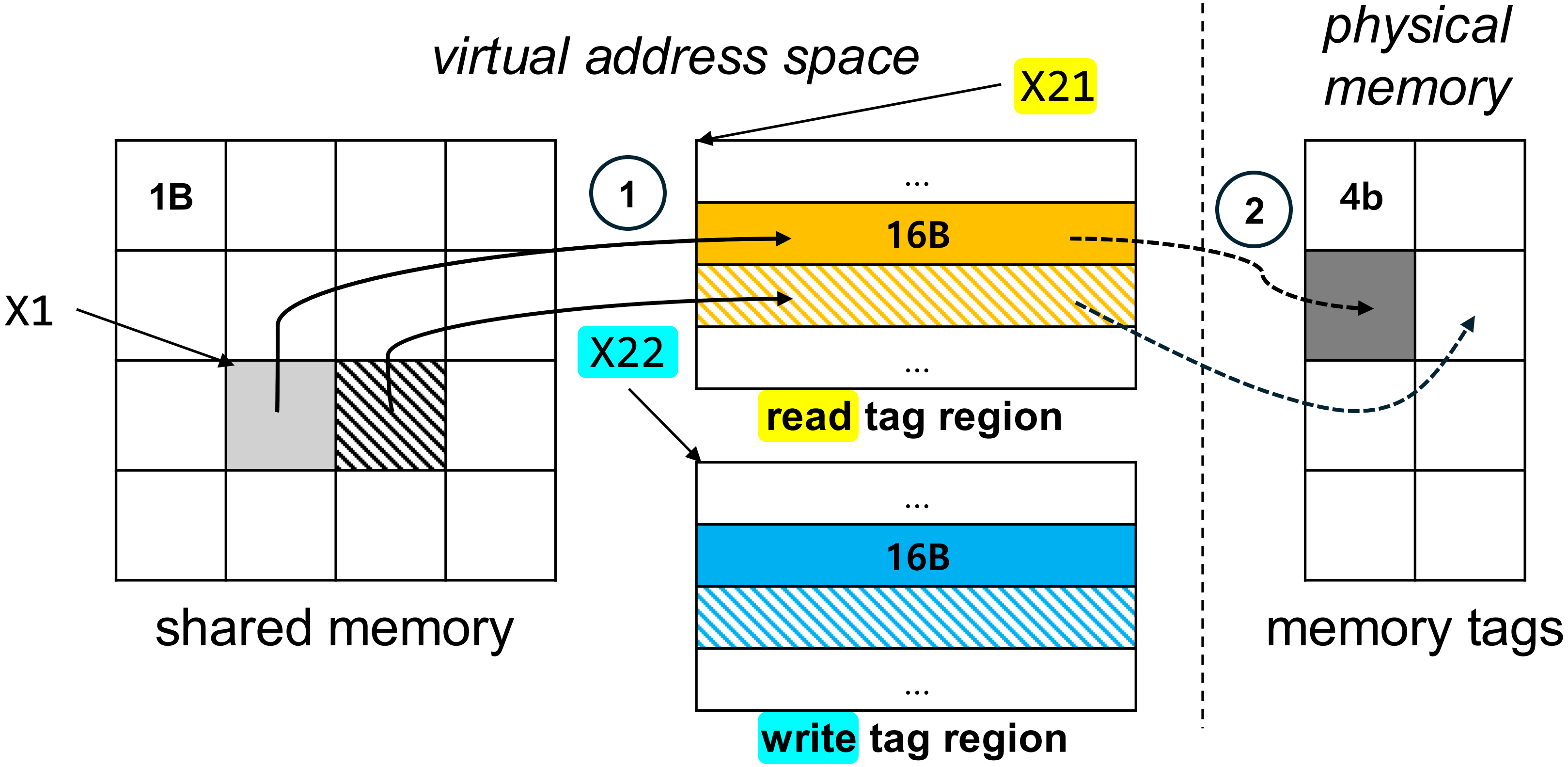
- Goal
  - Byte-level, per-domain, multi-policy access control on shared memory using ARM MTE



```
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

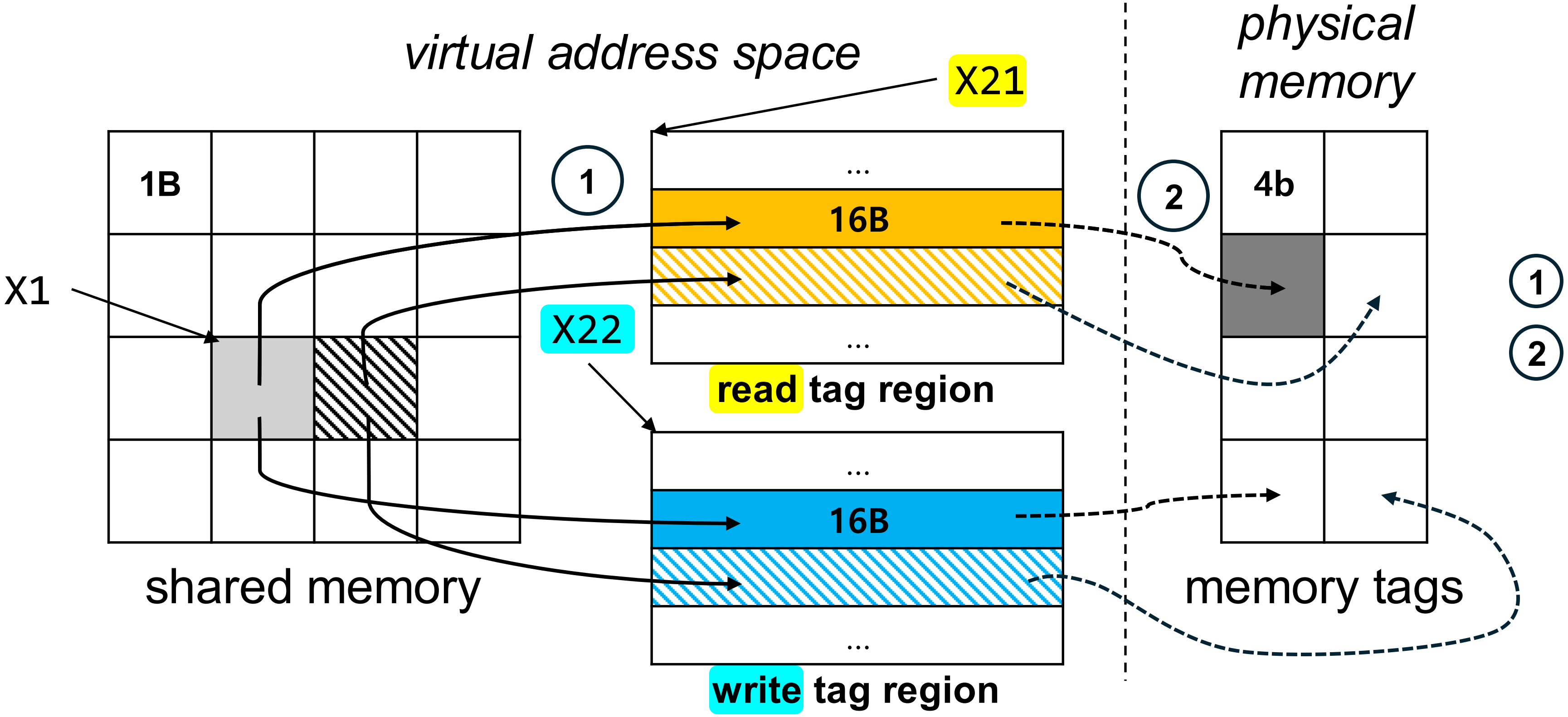


instructions

```
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



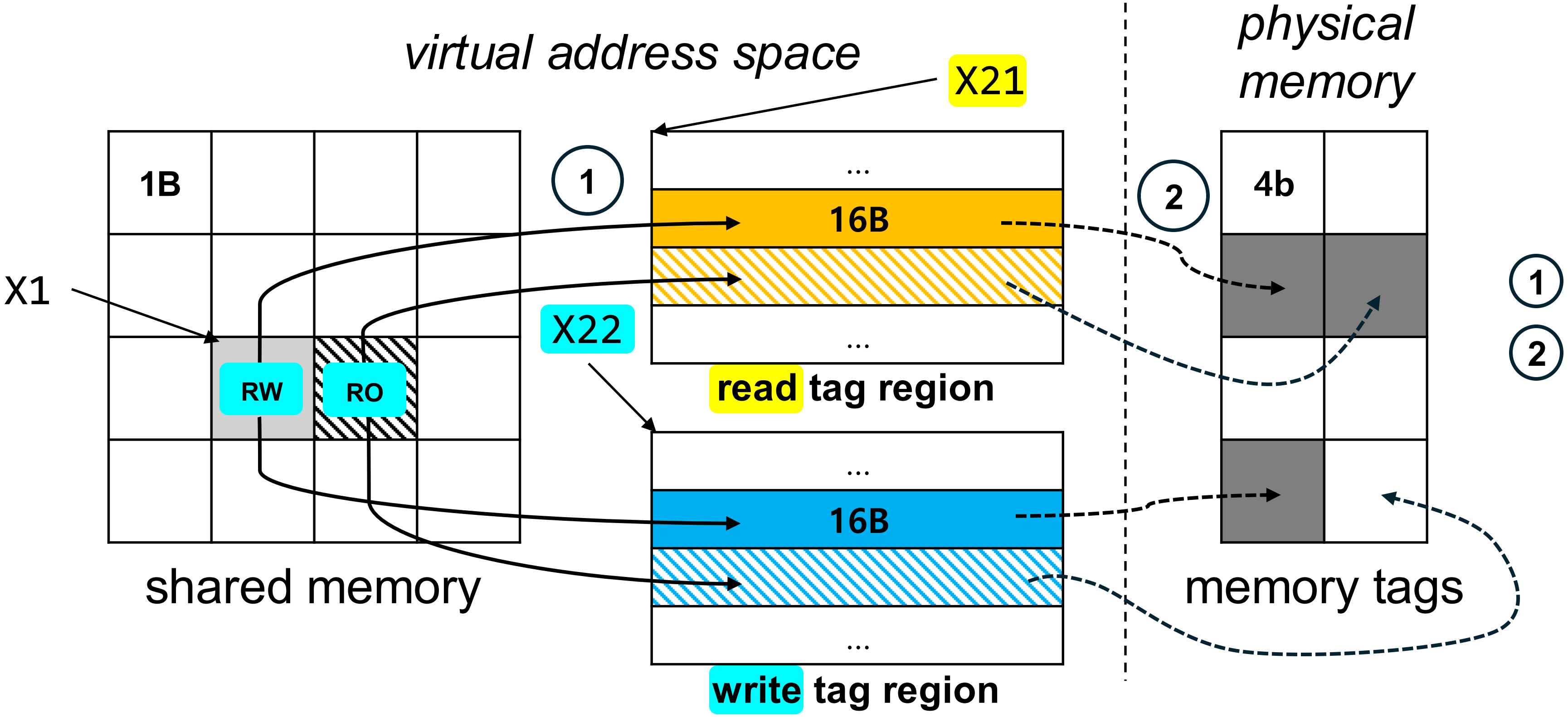
instructions

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

X21: reserved
X22: reserved

Shadow Memory Tagging

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

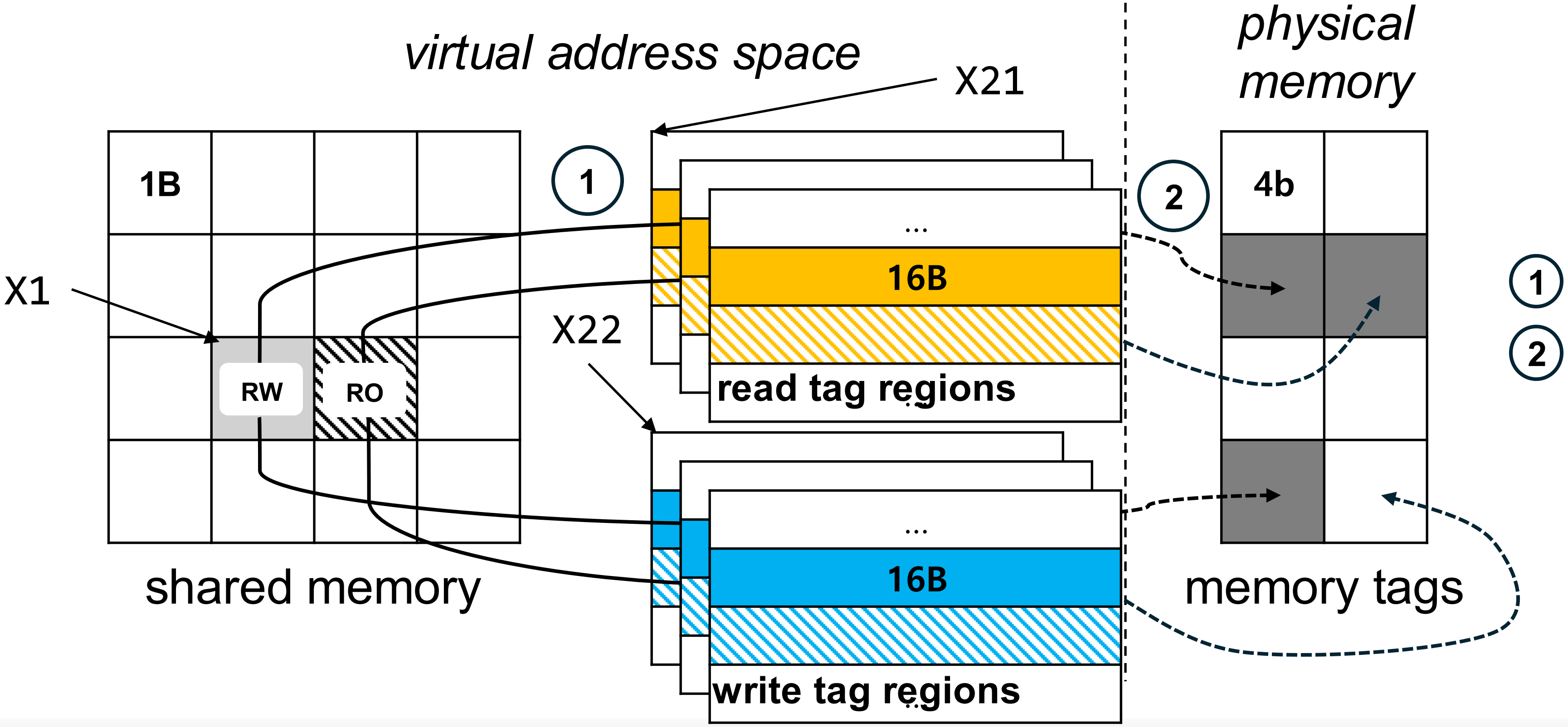


instructions

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

Shadow Memory Tagging

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



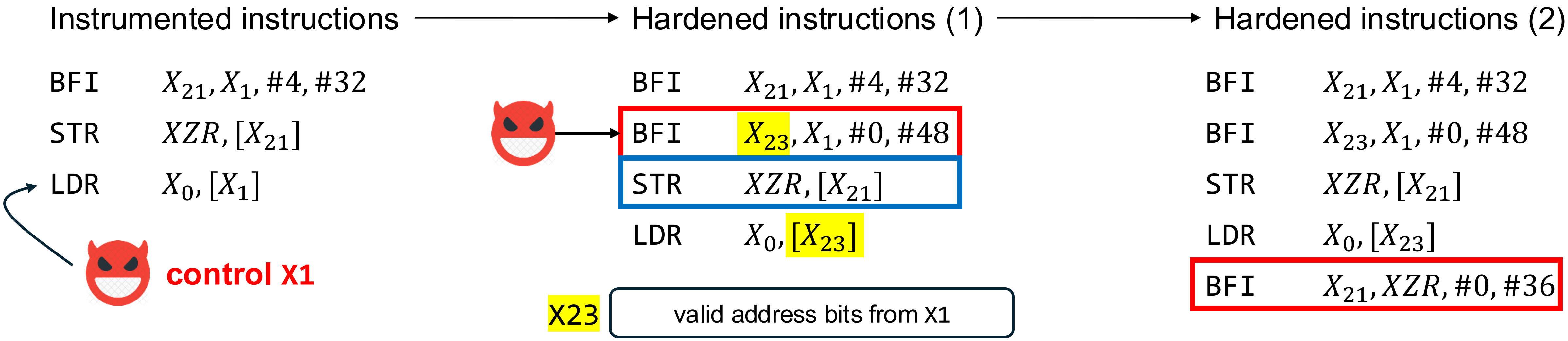
instructions

```
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



Instrumented instructions → Hardened instructions (1) → Hardened instructions (2)

Instrumented instructions

BFI $X_{21}, X_1, \#4, \#32$

STR $XZR, [X_{21}]$

LDR $X_0, [X_1]$

control X1

Hardened instructions (1)

BFI $X_{21}, X_1, \#4, \#32$

BFI $X_{23}, X_1, \#0, \#48$

STR $XZR, [X_{21}]$

LDR $X_0, [X_{23}]$

X23 valid address bits from X1

Hardened instructions (2)

BFI $X_{21}, X_1, \#4, \#32$

BFI $X_{23}, X_1, \#0, \#48$

STR $XZR, [X_{21}]$

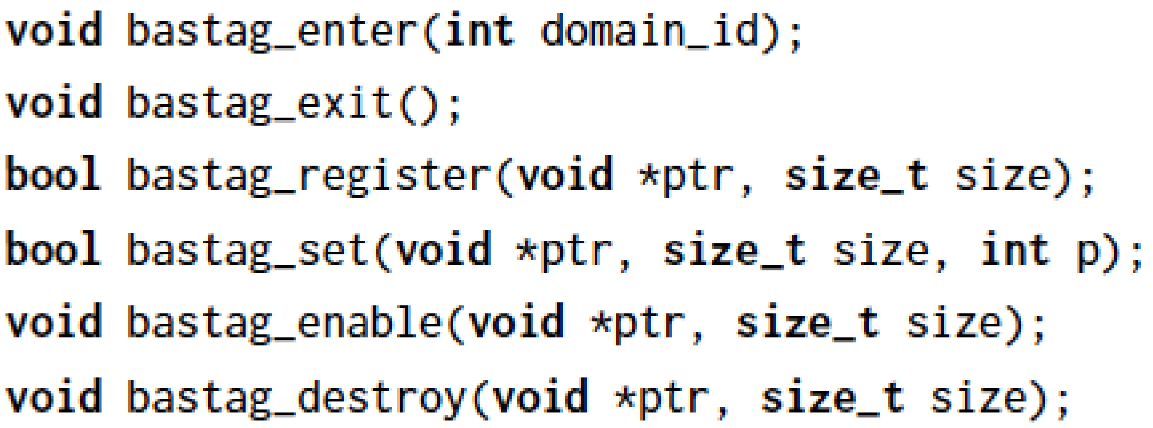
LDR $X_0, [X_{23}]$

BFI $X_{21}, XZR, \#0, \#36$

Optimizations and APIs

- Optimizations
  - Tag region sharing ➔ use same physical page for tag regions with identical permissions
  - Lazy tag mapping ➔ map the page for tag regions only when non-zero tag is necessary

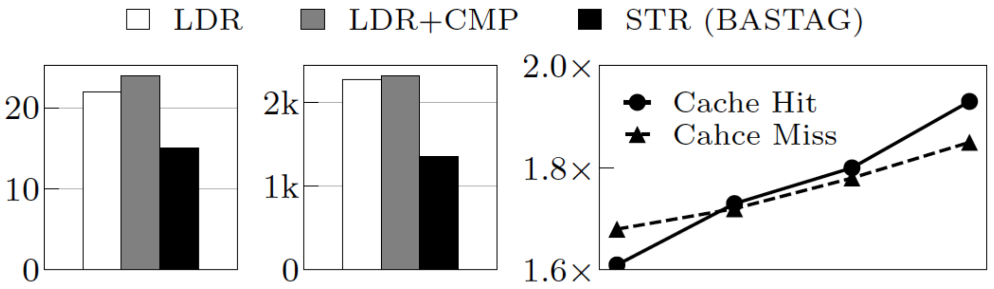
- APIs
  - Provide set of APIs for programmers to manage shared memory and its access permissions



```
void bastag_enter(int domain_id);
void bastag_exit();
bool bastag_register(void *ptr, size_t size);
bool bastag_set(void *ptr, size_t size, int p);
void bastag_enable(void *ptr, size_t size);
void bastag_destroy(void *ptr, size_t size);
```

Evaluation

Mechanism	Baseline	IRM-based	Msg-based	**BASTAG**
Δ*Counter*	37	54	78	40



(a) Cache Hit (b) Cache Miss (c) Relative Overhead

- Setup
  - Google Pixel 8 (w/ MTE support)
  - Kernel version 5.10.110
- Micro-benchmarks
  - Faster than alternatives as well as SW-only shadow memory schemes in terms of cycles
- Macro-benchmarks (3 case studies)
  - Kernel drivers (nullnet, nullblk) ➔ + 5.7% vs. 22.9% (SW-based)
  - Inter-task communication (PX4 middleware) ➔ + 7.1% vs. 21.1% (SW-based)
  - Multi-threaded application (Memcached) ➔ + 5.8% vs. 17.0% (SW-based)
  - Integration with isolation (for private memory) on SPEC2017rate ➔ + 8.6% vs. 20.3% (SW-based)

Conclusion

- BASTAG is an efficient solution that provides **byte-level, per-domain, multi-policy access control** on shared memory using ARM MTE

- BASTAG proposes a novel technique, ***shadow memory tagging***, to overcome the inherent limitations of MTE

- BASTAG outperforms existing byte-level access control solutions while demonstrating acceptable overhead when applied to realistic use cases