Caplification

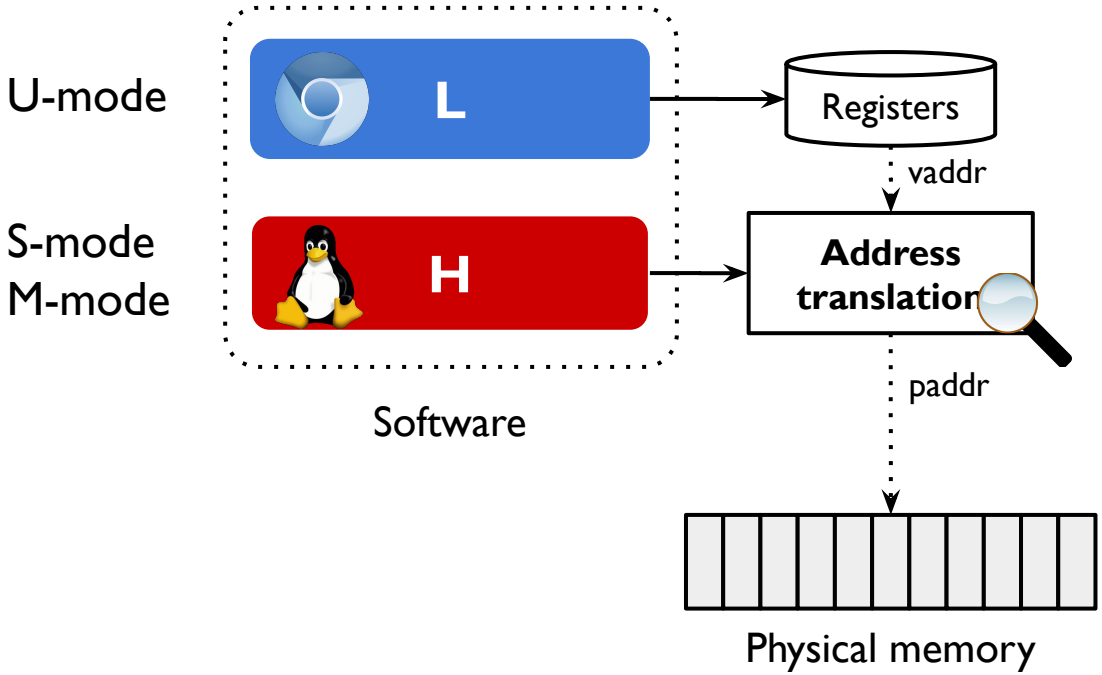
Bridging Capability-Aware and Capability-Oblivious Software

ACM SACMAT 2025

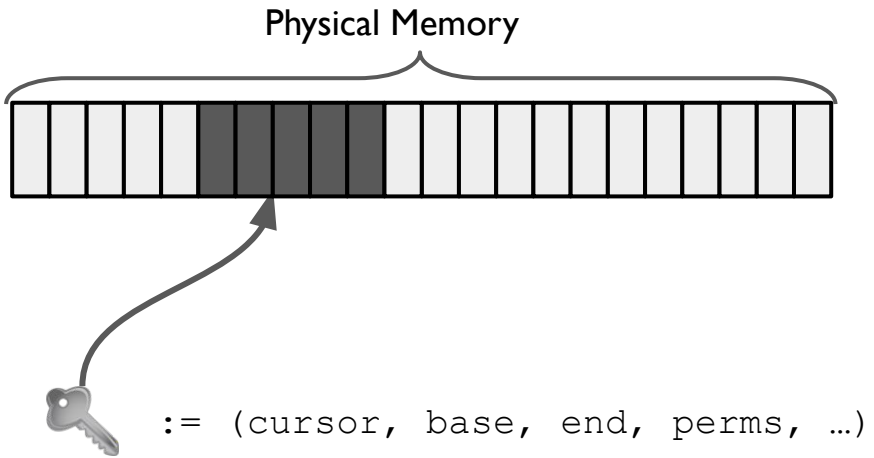
Jason Zhijingcheng Yu[1] Mingkai Li[2] Aditya Badole[1]
Trevor E. Carlson[1] Michael Swift[3] Prateek Saxena[1]

[1]National University of Singapore
[2]Columbia University
[3]University of Wisconsin–Madison

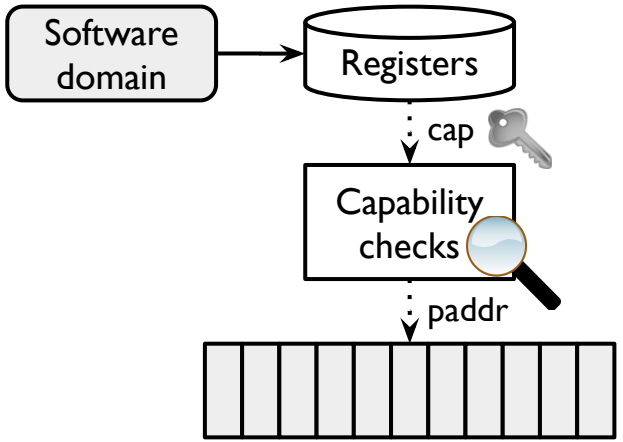
Virtual Memory Access Control



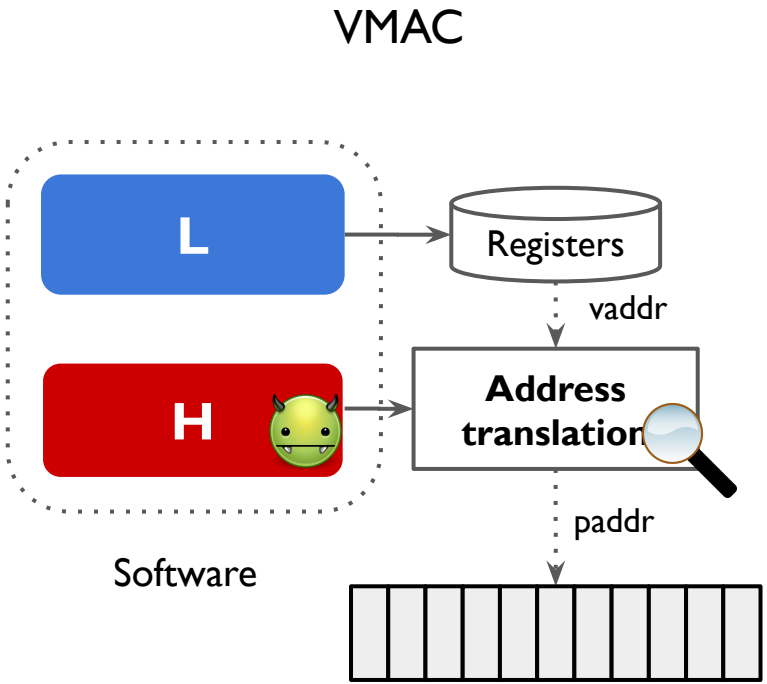
Alternative: Hardware Capabilities

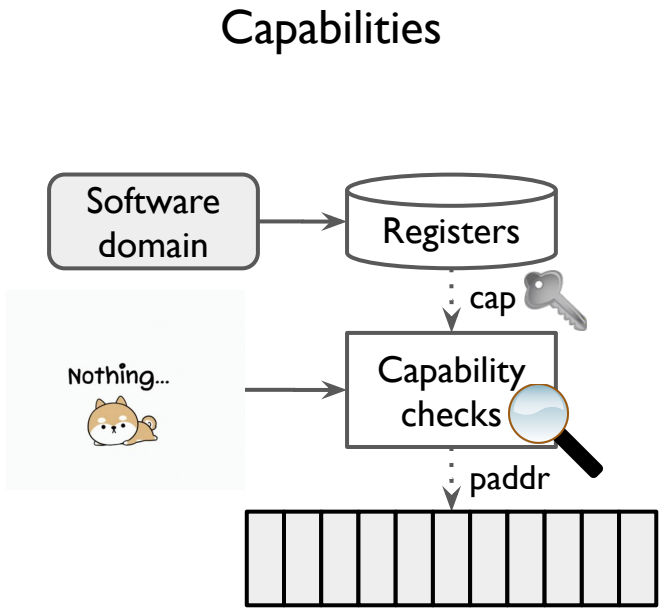



```
LD/ST addr, ...
LD/ST      , ...
```

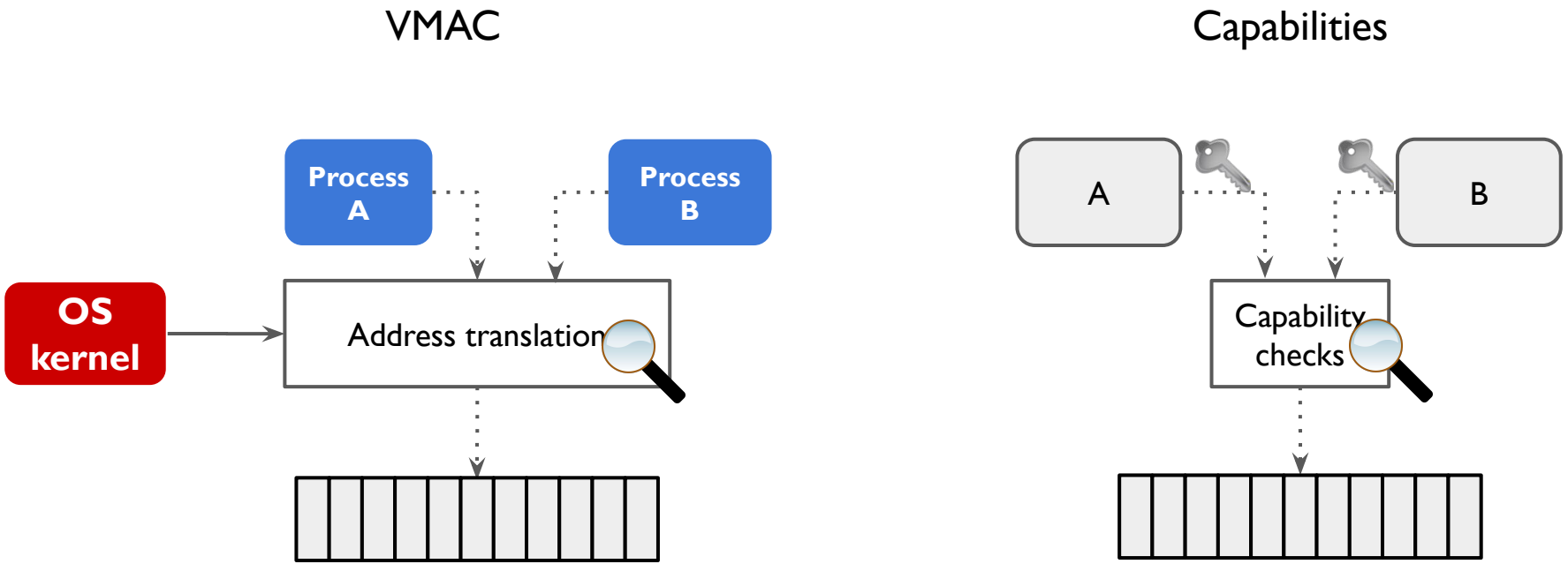


Hardware Capabilities Remove Ambient Authority

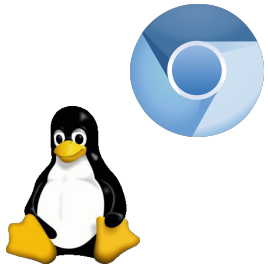


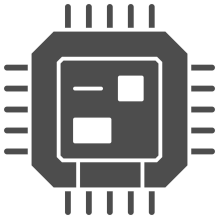


Hardware Capabilities Provide Context Independence



Problem: Compatibility

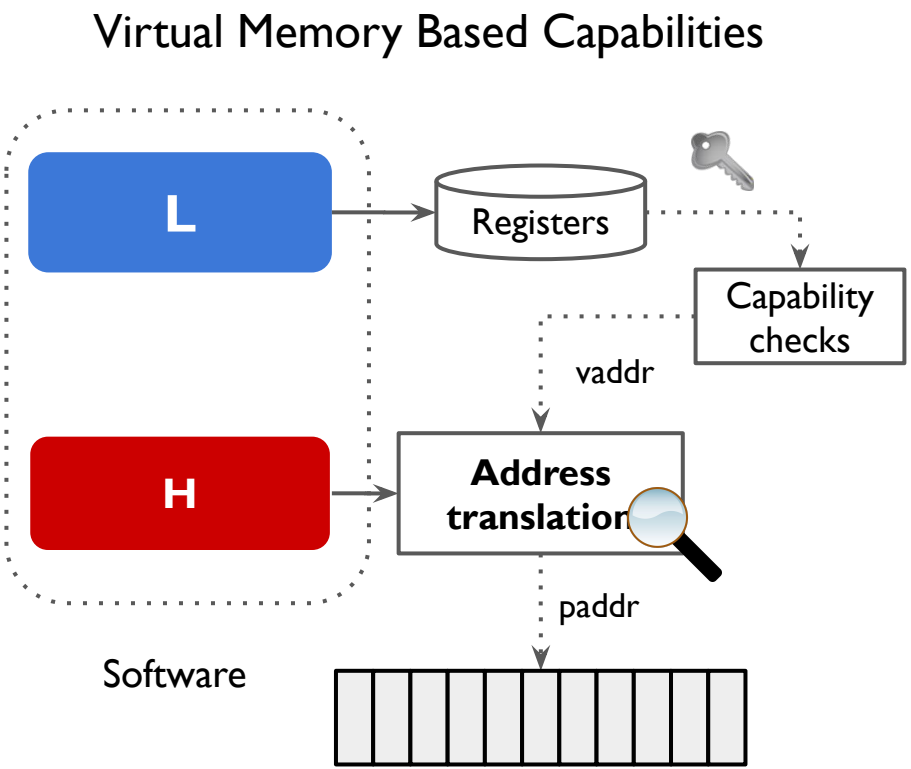




Capability-oblivious software

Capability-based hardware

Prior attempts sacrifice benefits of capabilities



✗ Central trusted authority

✗ Capabilities only meaningful in the same virtual address space

Goal: Bridge capability-aware and capability-oblivious software

Without sacrificing the strengths of capability-based isolation?

G1: unmodified capability-oblivious software stack

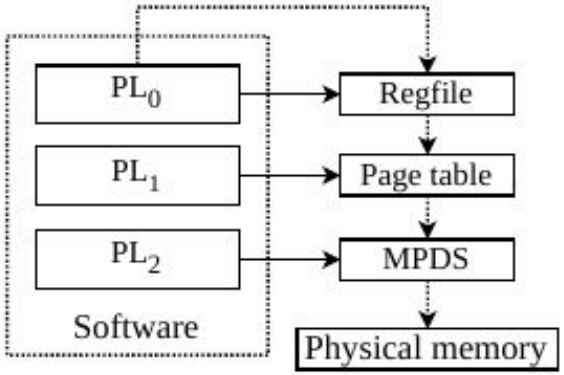
G2: capability-based isolation in kernel- and user-space without trusting privileged software

G3: capability-based memory sharing between capability-aware and capability-oblivious software

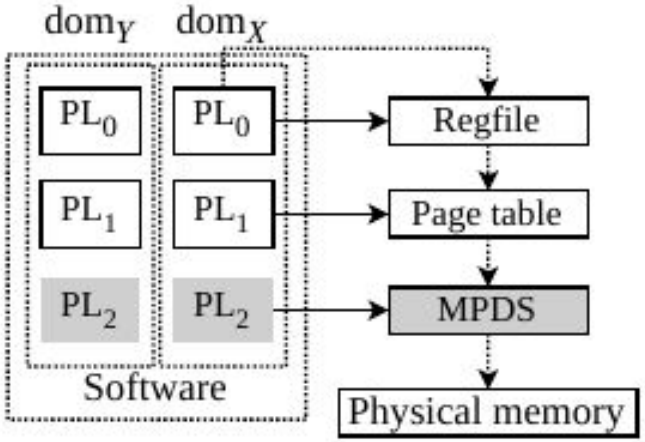
Prior Work Falls Short

Design	G1	G2	G3
CheriBSD	✓	✗	✓
CheriOS	✗	✗	✗
CHERI-TrEE	✗	✗	✗
CAP-VM	✓	✗	✗
ORC	✗	✗	✗
CODOM	✓	✗	✓
Caplification (our work)	✓	✓	✓

Core Idea: Caplification



(a) An abstract VMAC system before caplification



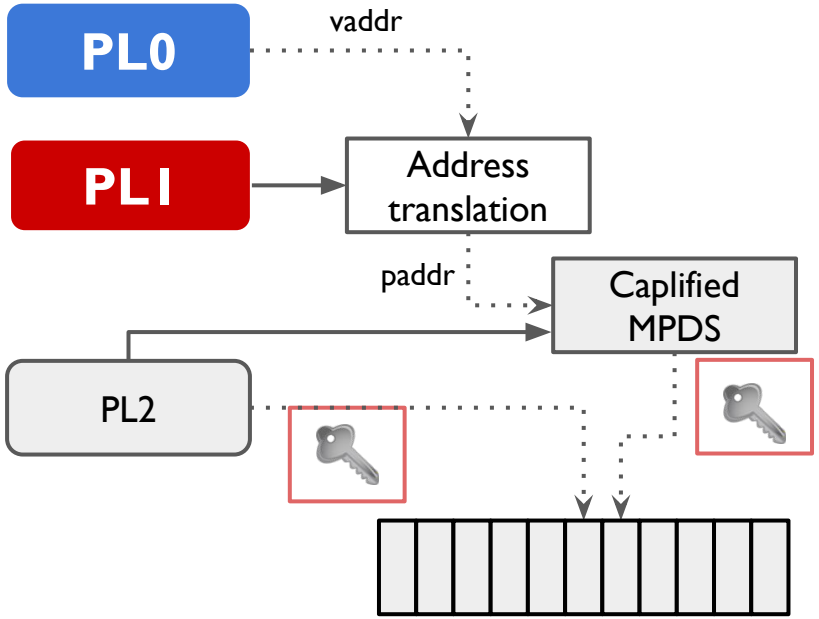
(b) An abstract VMAC system after caplification

Memory Access in Caplification

Two pathways to access memory

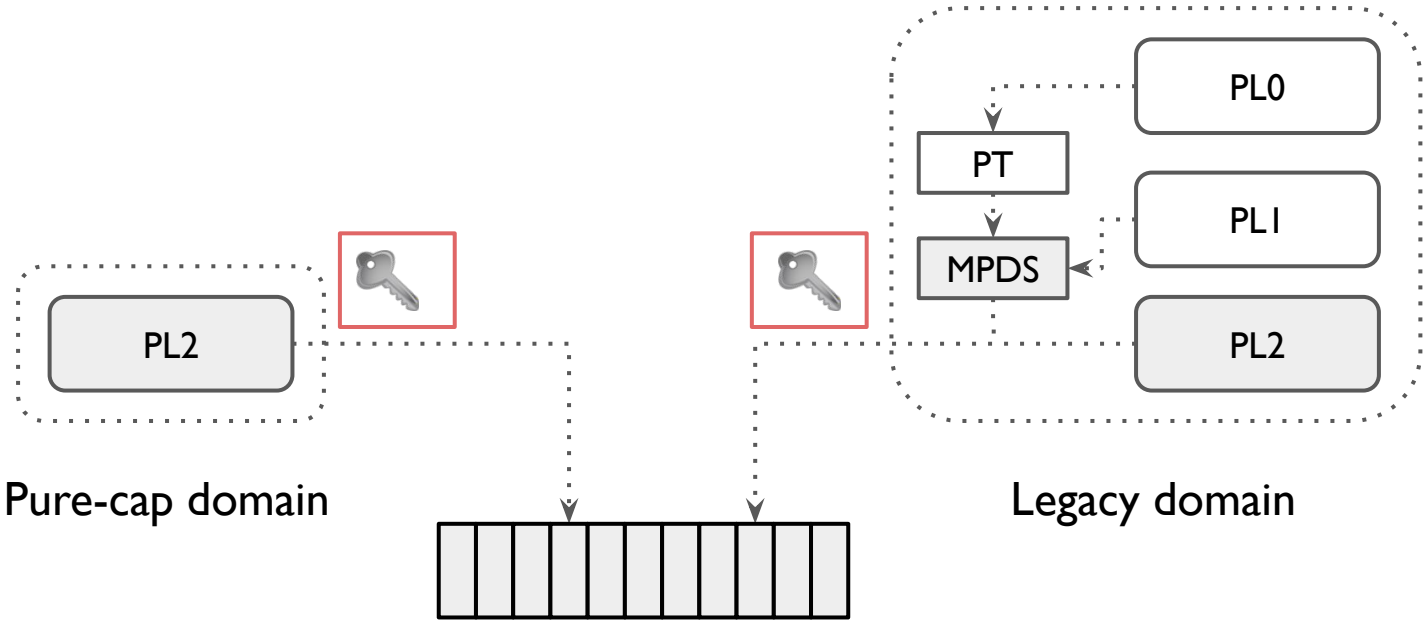
Capability-oblivious memory access

Capability-aware memory access



All memory accesses are ultimately done through capabilities.

Mixing Domains



Challenges

Exception and Interrupt Handling

Who should handle exceptions and interrupts when the domains do not trust one another?

DANGER

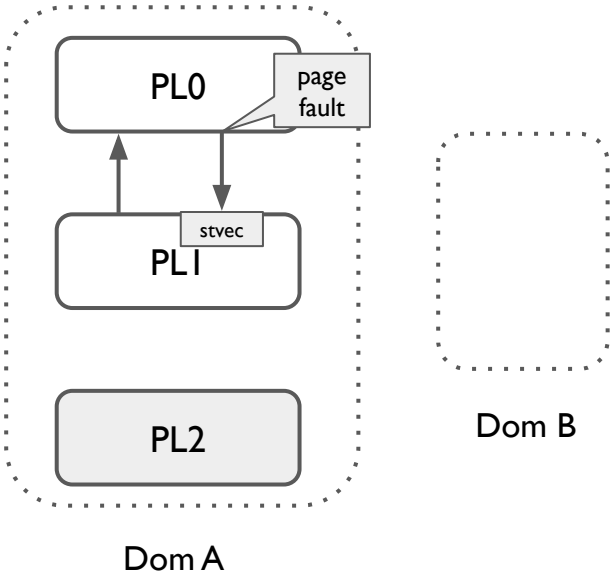
SGX controlled channels (AsiaCCS 2016)
SmashEx (CCS 2021)
Ahoi attacks
- USENIX 2024
- S&P 2024
- AsiaCCS 2025

Exceptions:
- Generated due to execution of current domain
- Targeting current domain
- Unsafe to deliver to other domains

Interrupts:
- Events external to CPU cores
- Targeting CPU core / whole system
- Unsafe to deliver to any application domain

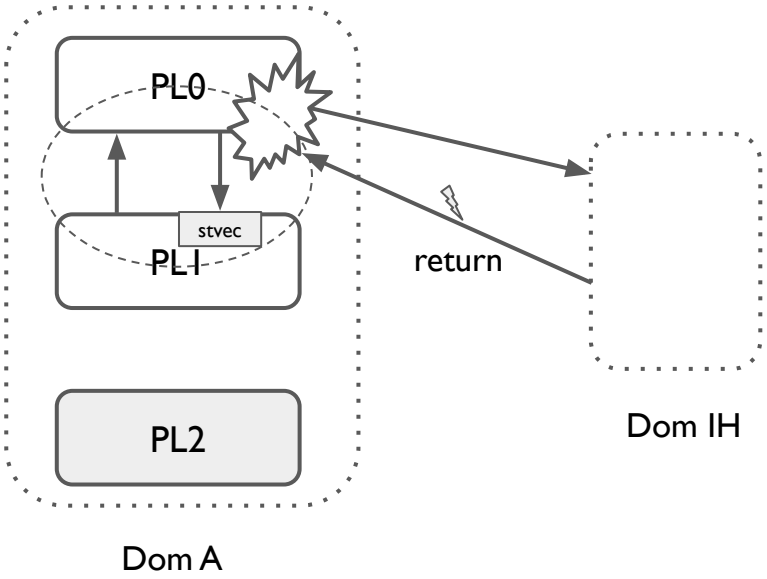
Local vs Global Delivery of Exceptions and Interrupts

Local Delivery
(exceptions, V-interrupts)



Same mechanism from base ISA

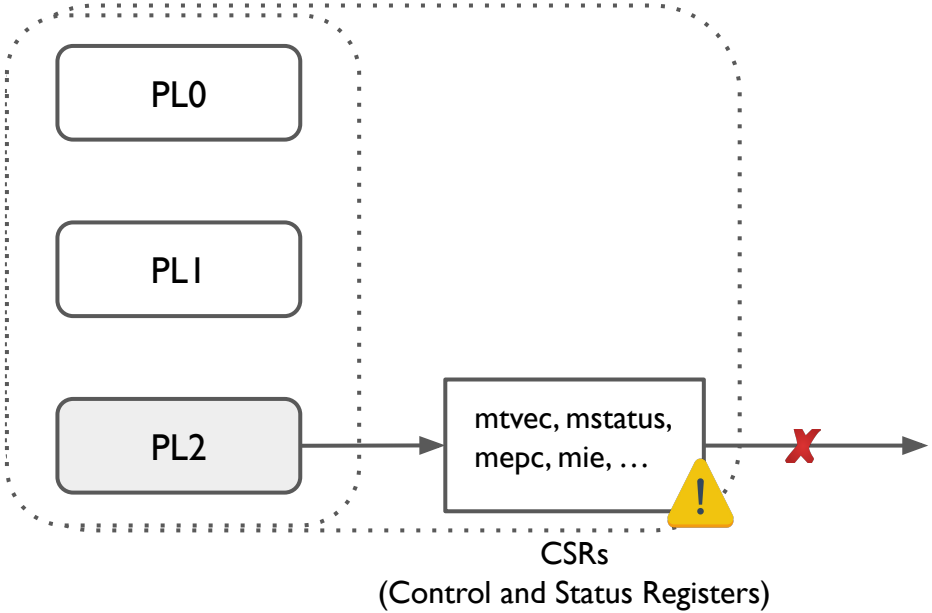
Global Delivery
(H-interrupts)



Delivered to global interrupt handler domain

Privileged States

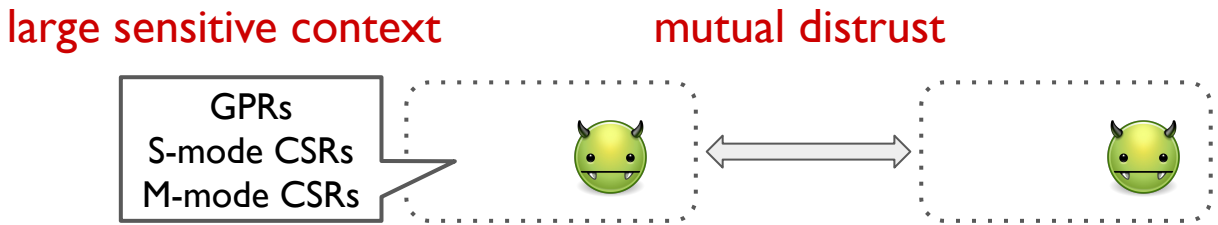
Localizing the effects of CSRs



Example: mie now only affects V-interrupts in the domain

Domain Switching

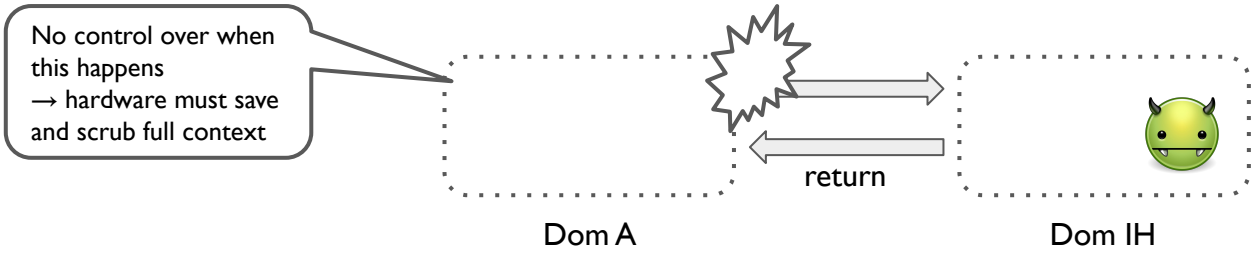
Why challenging?



Domain Switching

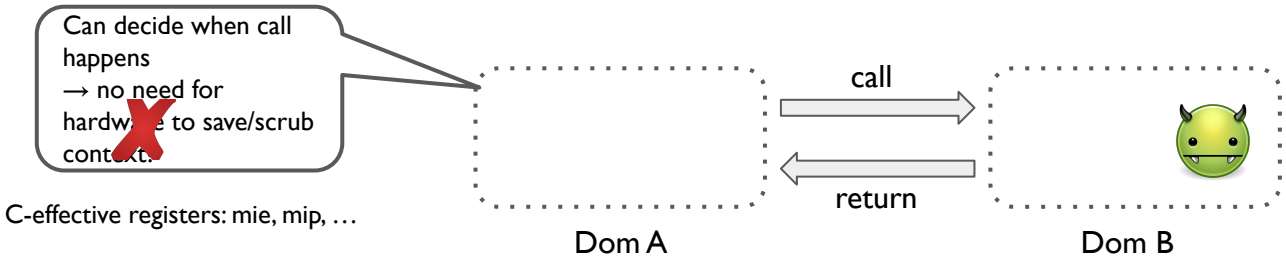
Asynchronous

Swap full context



Synchronous

Swap C-effective registers



Caplifive

Caplification of Capstone on RISC-V

Capstone supports flexible security needs

Minimal set of properties

P1: **Exclusive Access**

P2: **Revocable Delegation**

P3: **Extensible Hierarchy**

P4: **Secure Domain Switching**

CAPSTONE

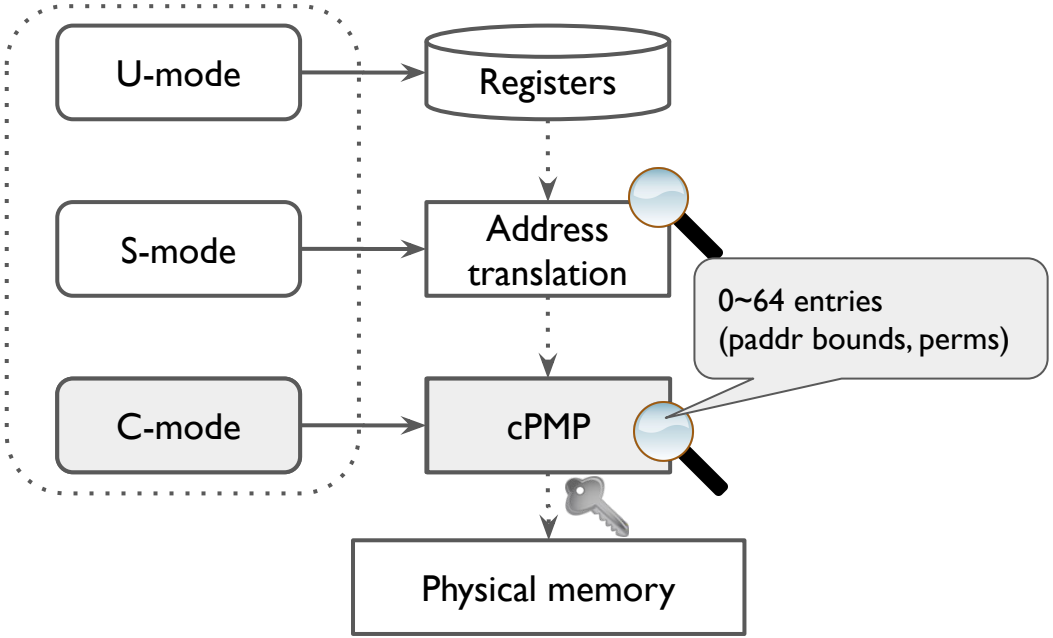
Capstone (USENIX '23)

- Spatial Memory Safety
- Temporal Memory Safety
- Concurrent Thread Safety
- Intra-process Sandboxing
- Process Sandboxing
- Virtualization
- Red-Green Secure Worlds
- Nested / App Virtualization

RISC-V-based Internal Structure

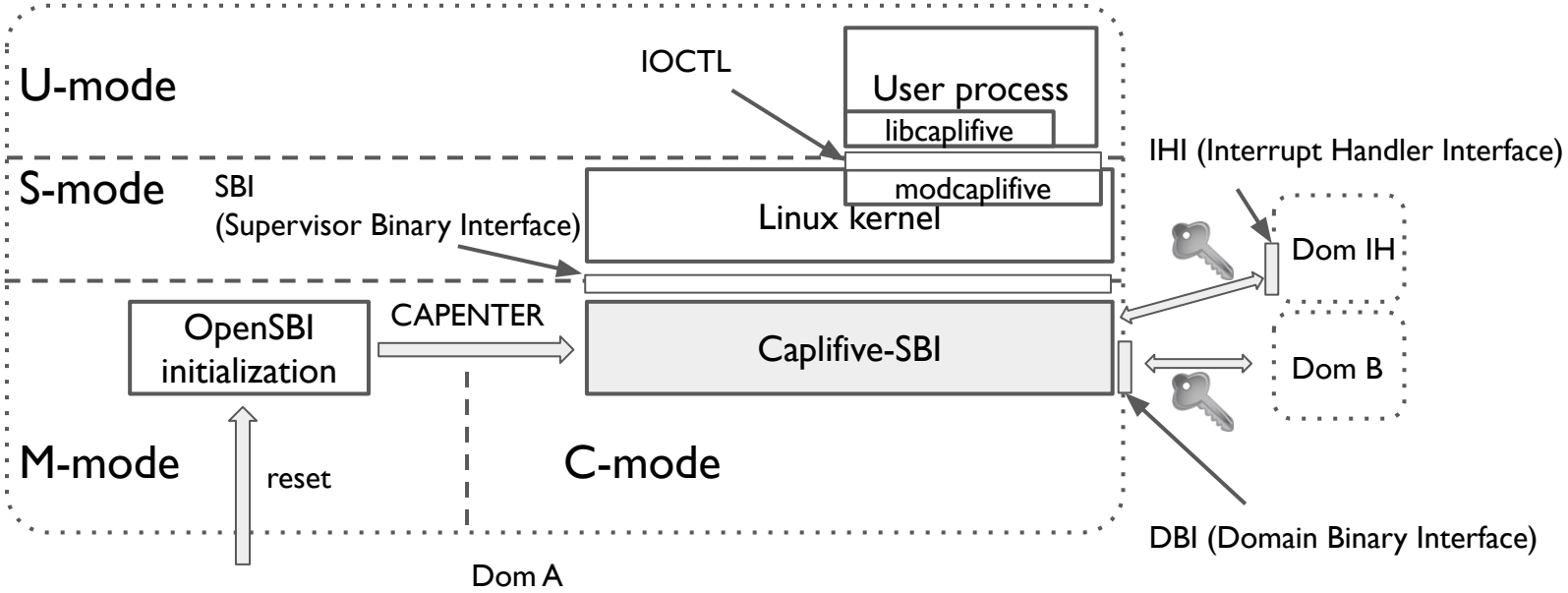
RISC-V: open; open-source SW & HW ecosystem

Caplified MPDS: PMP (Physical Memory Protection)



ISA documentation

Software Stack



Hardware Implementation

Implementation serves more than one purpose

Purpose	Requirement	Strategy
Platform for experimenting with software stacks	Performance	Functional simulator – QEMU
Evaluating overhead of hardware design	Cycle-level accuracy	RTL implementation – CVA6

CapliFive-QEMU

CapliFive-RTL

Evaluations

Evaluation Questions

EQ1: Overhead of individual operations

EQ2: Performance of applications compared with VMAC baselines

Baselines

PMP: PMP-based isolation similar to Keystone

No-isolation: run everything in M-mode without isolation

Microbenchmarks (EQ1)

CapliFive-RTL simulated with Verilator

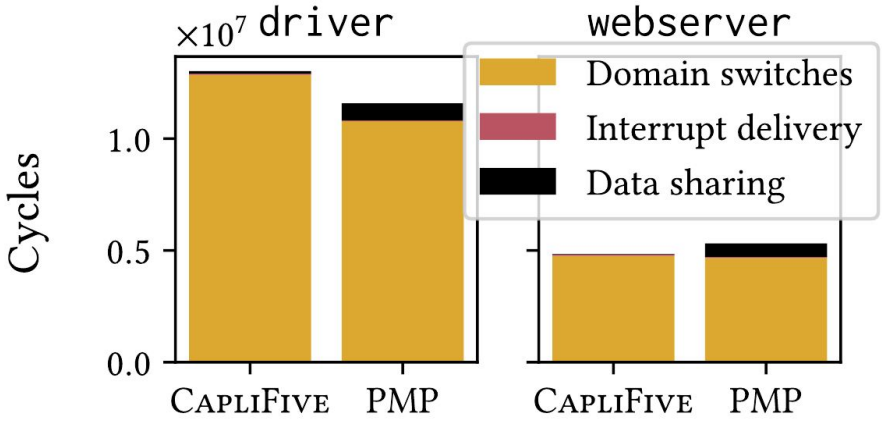
Macrobenchmarks (EQ2)

Profiling through CapliFive-QEMU

Evaluation Results

Microbenchmark	CapliFive	PMP	No-isolation
Sequential loads	511	511	511
Sequential stores	471	445	440
Sequential ptr. loads	766	511	511
Sequential ptr. stores	846	445	440
U-to-S trap	99	93	18
S-to-M/C trap	113	113	18
S-to-U return	67	67	14
M/C-to-S return	158	158	14
Sync. dom. switch (C)	88–308	N/A	18
Sync. dom. switch (S/U)	614	471	18
Interrupt delivery	951	731	119
Data sharing (1KB)	25	1042	0
Data transfer (1KB)	4	1042	0

Microbenchmarks



Macrobenchmarks

Most operations: similar overhead
Data sharing: less costly on CapliFive than on PMP
Interrupts: more costly on CapliFive than on PMP

Comparable performance
CapliFive reduces overhead of data sharing by up to 98%

Conclusion

Goal: bridging capability-aware and capability-oblivious software
without sacrificing the strengths of capability-based isolation

G1: unmodified capability-oblivious software stack

G2: capability-based isolation in kernel- and user-space without trusting privileged software

G3: capability-based memory sharing between capability-aware and capability-oblivious software

Caplification:
- Converting MPDS in existing ISAs to be capability-based
- Ensuring *all* memory accesses are ultimately *capability-based*
- Providing capability-oblivious *internal structures*

CapliFive: Caplifying Capstone on RISC-V

Thank you!

Code artifacts

ISA documentation