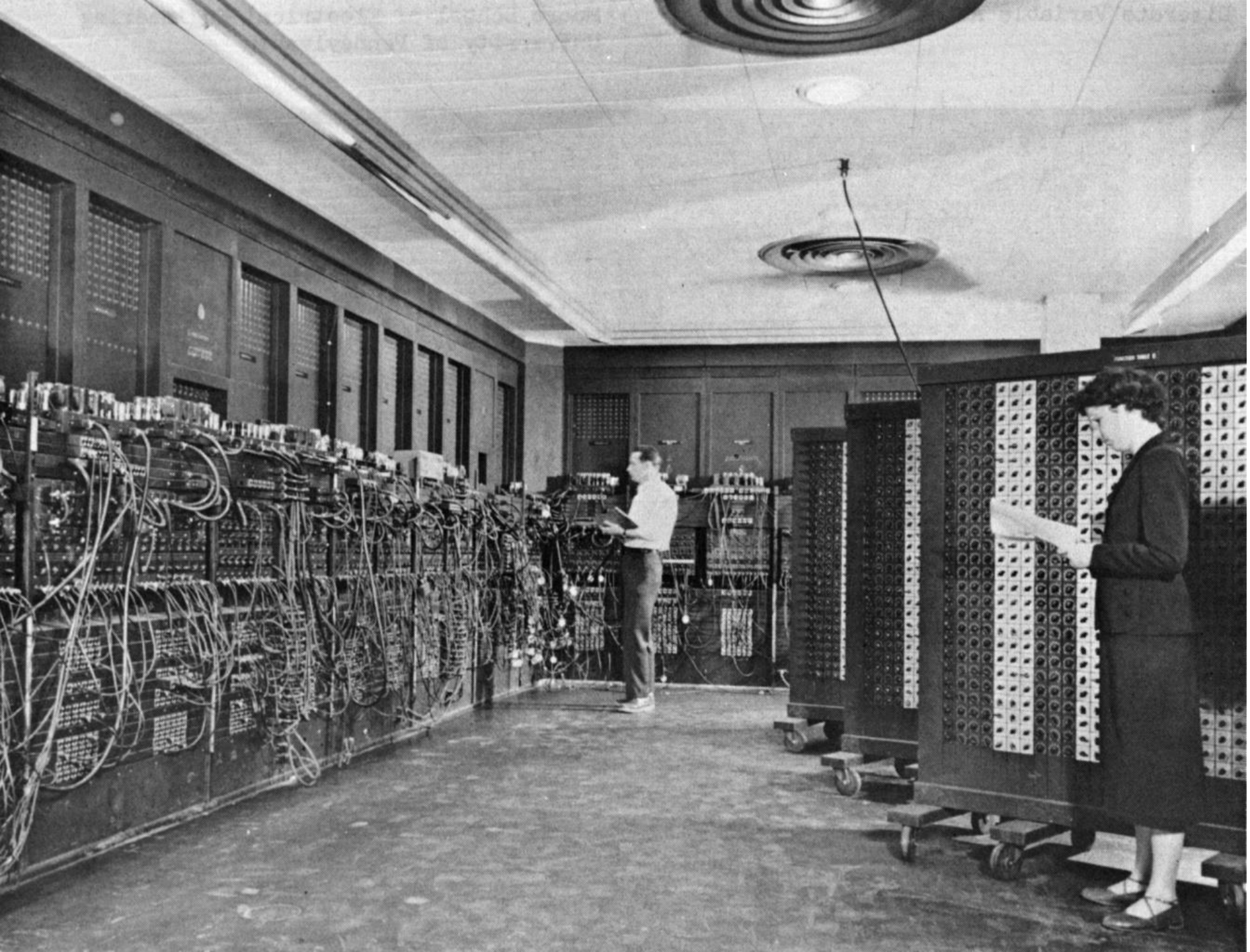
CS6465: Advanced Operating System Implementation

Lecture 10 – Modern isolation mechanisms

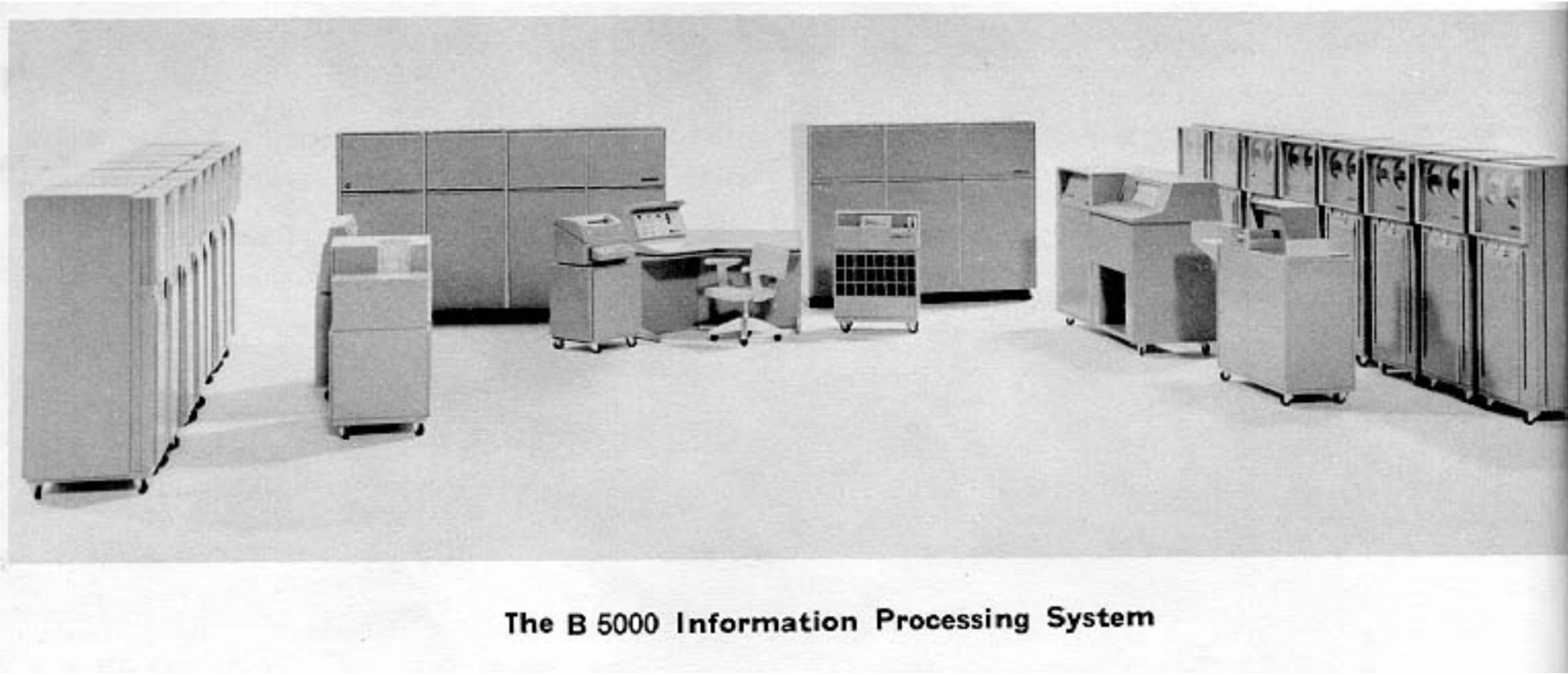
Anton Burtsev

November 2024



First Batch Architectures 1950- 1960

ENIAC 1945



The B 5000 Information Processing System

Segmentation: Burroughs B5000 1961

Paging: Atlas 1962 (University of Manchester)

A Kernel for Multics

Engineering a security kernel for a system requires isolating a minimum set of functions capable of supporting the system and finding a way to structure the kernel to facilitate verifying its correctness. Our plan is to produce a security kernel for Multics by removing nonkernel mechanisms from the supervisor, and restructuring the remaining kernel and partitioning it into multiple protection domains. This section describes these three interrelated categories of activities and provides specific examples of work underway or planned in each. The intention of the section is to communicate the spirit of the work rather than to to discuss thoroughly the various activities. The detailed results of individual activities are being communicated in other reports [16,17,18].

Final report of the Multics kernel design project

1977

Achieved IPC Performance

(Still The Foundation For Extensibility)

Jochen Liedtke * Kevin Elphinstone † Sebastian Schönberg ‡ Hermann
Gernot Heiser† Nayeem Islam* Trent Jaeger*

From L3 to seL4
What Have We Learnt in 20 Years of L4 Microkernels?

Kevin Elphinstone and Gernot Heiser
NICTA and UNSW, Sydney
{kevin.elphinstone,gernot}@nicta.com.au

Abstract

on the critical path of any service invocation, and low

Safe Hardware Access with the Xen Virtual Machine M

Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, Ma*
University of Cambridge Computer Laboratory, J J Thomson Avenue, Can

XFI: Software Guards for System Address Spaces

Úlfar Erlingsson
Microsoft Research
Silicon Valley

Martín Abadi
Microsoft Research
Silicon Valley
& UC Santa Cruz

Michael Vrable
UC San Diego

Mihai Budiu
Microsoft Research
Silicon Valley

George C. Necula
UC Berkeley

Abstract

a comprehensive protection system that offers
lexible access control and fundamental integrity

static analysis with inline software *guards* that
as in SFI) perform checks at runtime. The *XFI*
performs the static analysis as a linear inspection
structure of machine-code binaries; it ensures that

A Case for Language-Based Protection

Chris Hawblitzel and Thorsten von Eicken

Technical Report 98-1670
Department of Computer Science
Cornell University
Ithaca, NY

Abstract

The use of language mechanisms to enforce protection boundaries around software modules has

Singularity: Rethinking the Software Stack

Galen C. Hunt and James R. Larus
Microsoft Research Redmond

galenh@microsoft.com

ABSTRACT

modern operating systems have not evolved to accommodate the

The SawMill Multiserver Approach

Alain Gefflaut Trent Jaeger Yoonho Park
Jochen Liedtke * Kevin J. Elphinstone † Volkmar Uhlig †
Jonathon E. Tidswell † Luke Deller ‡ Lars Reuther ‡
IBM T.J Watson Research Center
Hawthorne, NY 10532

Nooks: An Architecture for Reliable Device Drivers *

Michael M. Swift, Steven Martin, Henry M. Levy, and Susan J. Eggers

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195, USA

{mikesw, stevaroo, levy, eggers} @cs.washington.edu

Architecture	Year	Policy Goal	Config?	Tag Impl			TCB					Compat.	Evaluation
				Size (bits)	Mgmt	Store	C	B	P	K	M		
Timber [118]	2019	IFC	No	2	ISA	W	●	●	●	●	●	Source	Simulation
ARM MTE [3]	2019	Memory Safety	Yes	4	ISA	N/A	●	●	●	●	●	Binary	ASIC
D-RI5CY [80]	2018	DIFT	Yes	1	API	W	●	●	●	●	●	Binary	FPGA
TMDFI [63]	2018	DIFT	Yes	8	ISA	W	●	●	●	●	●	Source	Simulation
HyperFlow [41]	2018	IFC	Yes	8	ISA	PT			●		●	None	FPGA
SDMP [90]	2018	Memory Safety	Yes	11	API	W	●	●	●		●	Source	Simulation
Typed Arch. [53]	2017	N/A	No	9	ISA	D	●	●	●	●	●	Source	FPGA
Dover [106]	2017	Programmable	Yes	Large	API	LT	●	●	●		●	Source	FPGA
Shakti-T [67]	2017	Memory Safety†	No	1	ISA	W	●	●	●	●	●	Source	FPGA
HDFI [99]	2016	DIFT	No	1	ISA	D	●	●	●	●	●	Source	FPGA
lowRISC [64, 101]	2015	Programmable	Yes	4	ISA	D	●	●	●	●	●	Source	FPGA
Taxi [43, 47]	2015	Memory Safety	No	8	Auto	D		●	●	●	●	Binary	Simulation
PUMP [30, 35, 37]	2014	Programmable	Yes	32	API	LT	●	●	●		●	Source	Simulation
CHERI [116, 117, 121]	2014	Programmable	No	1	Auto	W	●		●		●	Source	FPGA
SPARC M7 [75, 84, 88]	2014	Memory Safety	No	4	Auto	D		●	●	●	●	Binary	ASIC
Low-Fat Pointers [57]	2013	Memory Safety	No	8	Auto	W		●	●	●	●	None	FPGA
SAFE [5, 36]	2012	Programmable	Yes	59	API	W		●	●	●	●	None	FPGA
DataSafe [17]	2012	IFC	Yes	10	Auto	D			●		●	Binary	Simulation
Harmoni [33]	2012	DIFT	Yes	≤32	API	D		●	●	●	●	Binary	FPGA
Shioya et al. [93]	2011	IFC†	Yes	12	API	PT		●	●	●	●	Binary	Simulation
SIFT [78]	2011	DIFT	Yes	32	API	D		●	●	●	●	Binary	Simulation
FlexCore [32]	2010	Programmable†	Yes	Large	API	D	●	●	●	●	●	Source	FPGA
Execution Leases [109]	2009	IFC	No	1	Auto	D	●	●	●	●	●	None	FPGA

From SAMUEL JERO et al. TAG: Tagged Architecture Guide. ACM Computing Surveys, 2022

Yet for a while we only had segmentation and paging

- Hardware isolation primitives
  - Segmentation (46 cycles)
    - Jochen Liedtke. Improved address-space switching on Pentium processors by transparently multiplexing user address spaces. Technical Report, 1995
    - Deprecated when x86 moved to 64bits
  - Page tables (806 cycles)
    - https://sel4.systems/About/Performance/

- Network processing frameworks like DPDK spend less than 100 cycles per packet

Renewed interest in hardware support for isolation

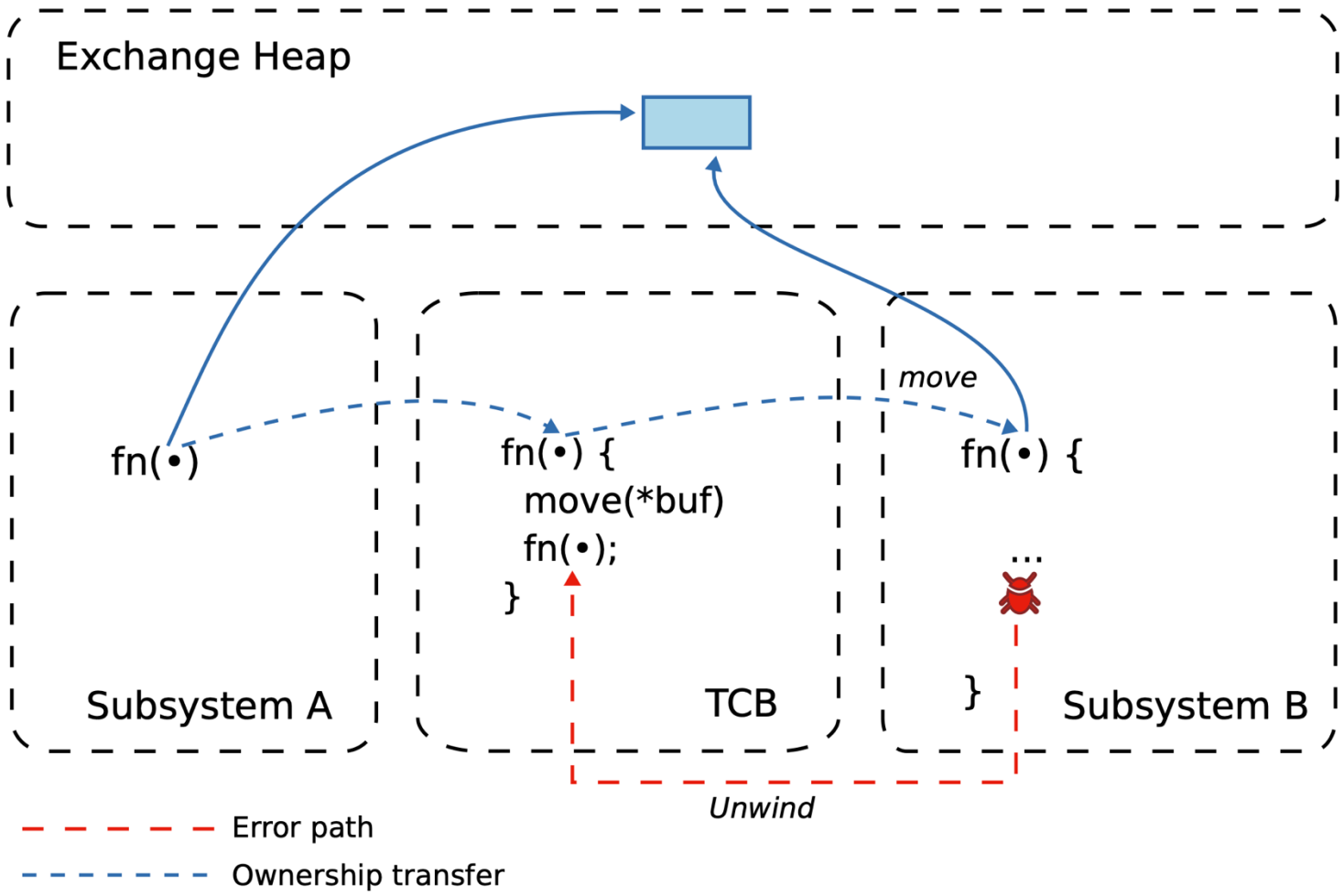
- Memory Protection Keys (MPK)
- Extended Page-Table (EPT) switching with VM functions (VMFUNC)
- Memory Tagging Extension (MTE)
- Intel 128-byte sub-page permissions (SPP)
- ARM pointer authentication (PAC)
- ARM Morello is the first silicon implementation of the CHERI capability model
- More…

Are we doing this right?

Design Principles

- Low-overhead enforcement
  - 1-2%
- Fast switching of isolation boundary
  - Ideally close to a function call
  - Cannot afford privilege level switch
- Zero-copy passing of data
- ``Move'' semantics and revocation

Isolation model



Evolving Mach 3.0 to a Migrating Thread Model

Bryan Ford *Jay Lepreau*

University of Utah

Abstract

We have modified Mach 3.0 to treat cross-domain remote procedure call (RPC) as a single entity, instead of a sequence of message passing operations. With RPC thus elevated, we improved the transfer of control during RPC by changing the thread model. Like most operating systems, Mach views threads as statically associated with a single task, with two threads involved in an RPC. An alternate model is that of migrating threads, in which, during RPC, a single thread abstraction moves between tasks with the logical flow of control, and "server" code is passively executed. We have compatibly replaced Mach's static threads with migrating threads, in an attempt to isolate this aspect of operating system design and implementation. The key element of our design is a decoupling of the thread abstraction into the *execution context* and the *schedulable thread of control*, consisting of a chain of contexts. A key element of our implementation is that threads are now "based" in the kernel, and temporarily make excursions into tasks via upcalls. The new system provides more precisely defined semantics for thread manipulation and additional control operations, allows scheduling and accounting attributes to follow threads, simplifies kernel code, and improves RPC performance. We have retained the old thread and IPC interfaces for backwards compatibility, with no changes required to existing client programs and only a minimal change to servers, as demonstrated by a functional Unix single server and clients. The logical complexity along the critical RPC path has been reduced by a factor of nine. Local RPC, doing normal marshaling, has sped up by factors of 1.7–3.4. We conclude that a migrating-thread model is superior to a static model, that kernel-visible RPC is a prerequisite for this improvement, and that it is feasible to improve existing operating systems in this manner.[1]

https://www.usenix.org/legacy/publications/library/proceedings/sf94/full_papers/ford.pdf

Modern Isolation Primitives

Limitations and Opportunities of Modern Hardware Isolation Mechanisms

Xiangdong Chen[1], Zhaofeng Li[1], Tirth Jain[†2], Vikram Narayanan[1], and Anton Burtsev[1]

[1]University of Utah, [2]Maya Labs

Abstract

A surge in the number, complexity, and automation of targeted security attacks has triggered a wave of interest in hardware support for isolation. Intel memory protection keys (MPK), ARM pointer authentication (PAC), ARM memory tagging extensions (MTE), and ARM Morello capabilities are just a few hardware mechanisms aimed at supporting low-overhead isolation in recent CPUs. These new mechanisms aim to bring practical isolation to a broad range of systems, e.g., browser plugins, device drivers and kernel extensions, user-defined database and network functions, serverless cloud platforms, and many more. However, as these technologies are still nascent, their advantages and limitations are yet unclear. In this work, we do an in-depth look at modern hardware isolation mechanisms with the goal of understanding their suitability for the isolation of subsystems with the tightest performance budgets. Our analysis shows that while a huge step forward, the isolation mechanisms in commodity CPUs are still lacking implementation of several design principles critical for supporting low-overhead enforcement of isolation boundaries, zero-copy exchange of data, and secure revocation of access permissions.

(EPT) switching with VM functions (VMFUNC) [52] deployed in modern Intel CPUs provide support for memory isolation with overheads gradually approaching the overhead of a function call [41, 76, 83, 112]. Both X86 and ARM are exploring hardware support for sub-page isolation [7, 20]. The latest ARM CPUs introduce 16-byte-granularity memory isolation with the Memory Tagging Extension (MTE) [7, 10]. Intel introduced support for 128-byte sub-page permissions (SPP) [20]. Finally, ARM implements pointer authentication (PAC), i.e., cryptographic signing of pointers in hardware, that can be used to implement both control-flow integrity [67] and subsystem isolation [75]. ARM Morello is the first silicon implementation of the CHERI capability model [119].

In contrast to traditional hardware isolation techniques (e.g., segmentation and page tables), the new primitives are designed to support lightweight cross-subsystem invocations (in the low hundreds and even low tens of cycles). Hence these new mechanisms bring a promise of practical isolation of small untrusted extensions and third-party code that require frequent communication with the rest of the system, e.g., browser plugins [2, 78, 84, 124], database extensions [15, 105], virtualized network functions [6, 45, 48, 73, 92, 96], web applications [3, 27, 32, 56], serverless cloud and edge platforms [4,

https://mars-research.github.io/doc/2024-atc-hw-isolation.pdf

Detour: Software Fault Isolation

Will appear in the 2009 IEEE Symposium on Security and Privacy

Native Client: A Sandbox for Portable, Untrusted x86 Native Code

Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth,
Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar
Google Inc.

Abstract

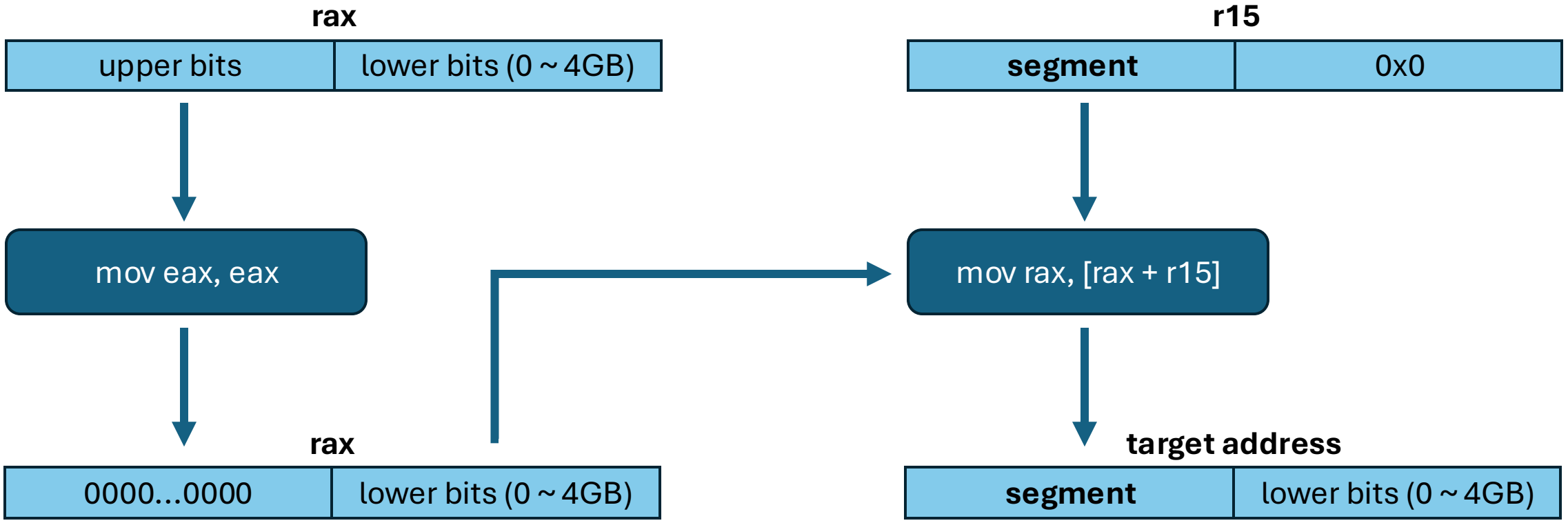
This paper describes the design, implementation and evaluation of Native Client, a sandbox for untrusted x86 native code. Native Client aims to give browser-based applications the computational performance of native applications without compromising safety. Native Client uses software fault isolation and a secure runtime to direct system interaction and side effects through interfaces managed by Native Client. Native Client provides operating system portability for binary code while supporting performance-oriented features generally absent from web application programming environments, such as thread support, instruction set extensions such as SSE, and use of compiler intrinsics and hand-coded assembler. We combine these properties in an open architecture that encourages community review and 3rd-party tools.

1. Introduction

As an application platform, the modern web browser brings together a remarkable combination of resources, including seamless access to Internet resources, high

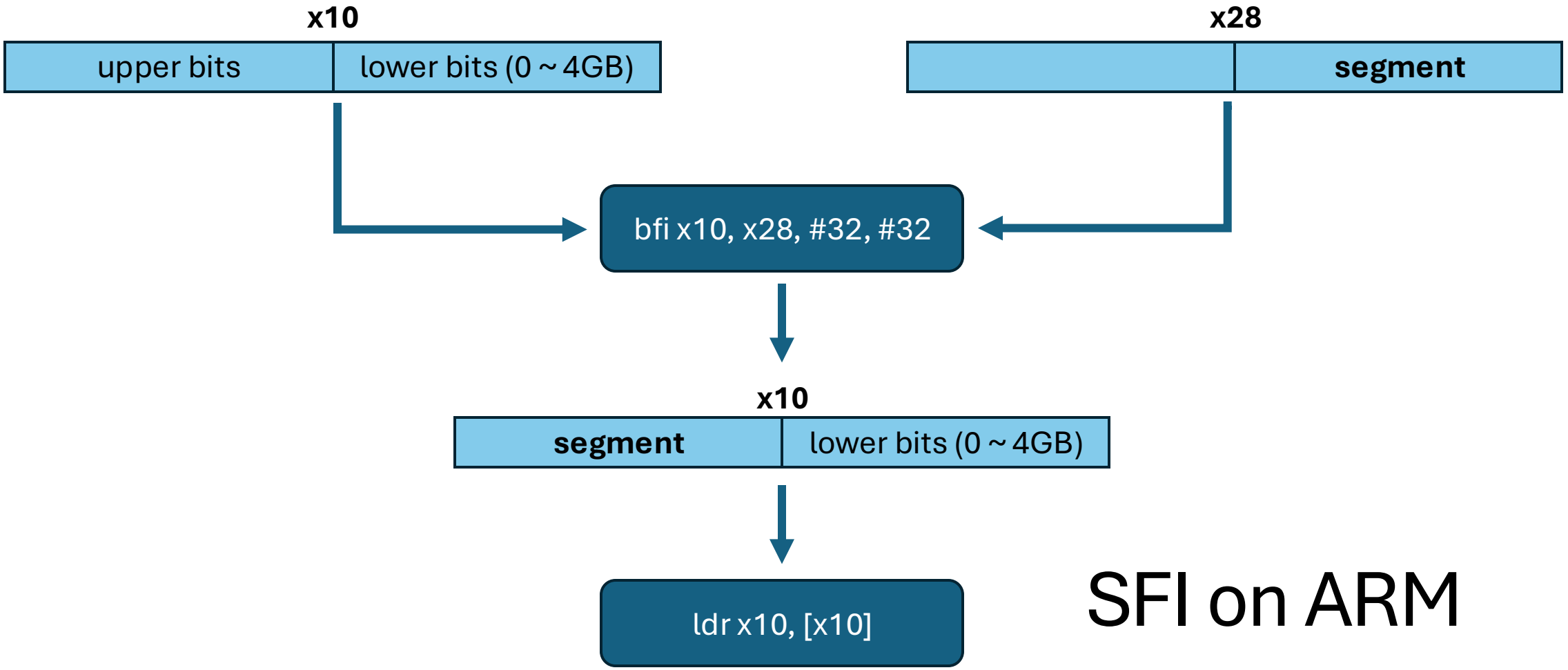
as a secondary consideration. Given this organization, and the absence of effective technical measures to constrain these plugins, browser applications that wish to use native-code must rely on non-technical measures for security; for example, manual establishment of trust relationships through pop-up dialog boxes, or manual installation of a console application. Historically, these non-technical measures have been inadequate to prevent execution of malicious native code, leading to inconvenience and economic harm [10], [54]. As a consequence we believe there is a prejudice against native code extensions for browser-based applications among experts and distrust among the larger population of computer users.

While acknowledging the insecurity of the current systems for incorporating native-code into web applications, we also observe that there is no fundamental reason why native code should be unsafe. In Native Client, we separate the problem of safe native execution from that of extending trust, allowing each to be managed independently. Conceptually, Native Client is organized in two parts: a constrained execution environment for native code to prevent unintended side effects, and a runtime for hosting these native code extensions through which allowable side effects may occur



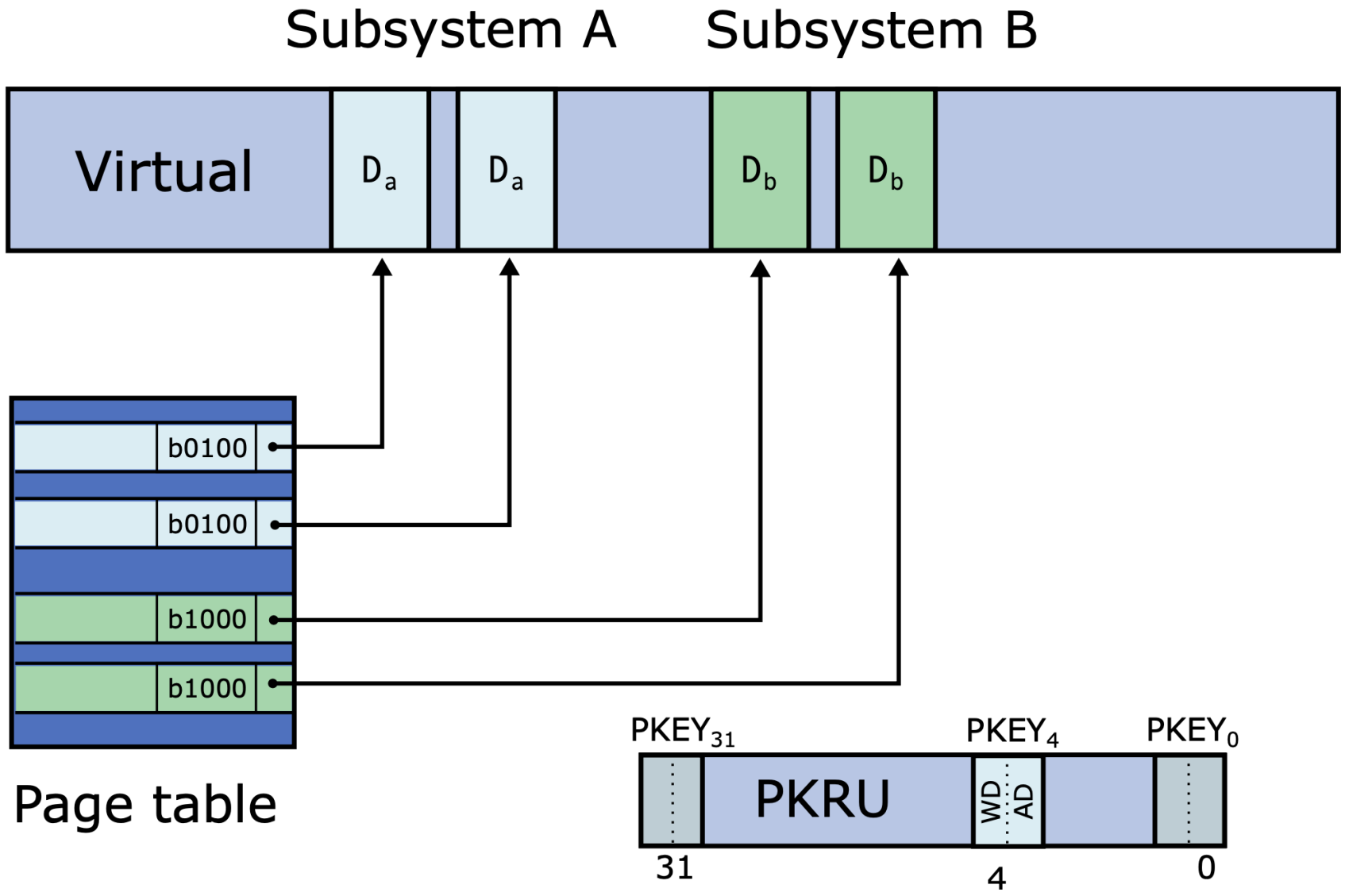

```
1  ; load value at [rax] to rcx
2  mov eax, eax ; eax contains 0-4GB
3  mov rcx, [r15, rax, 1] ;memory access within [r15 + 0-4GB]
```

Short primer on SFI (x86 NaCl, Wasm)

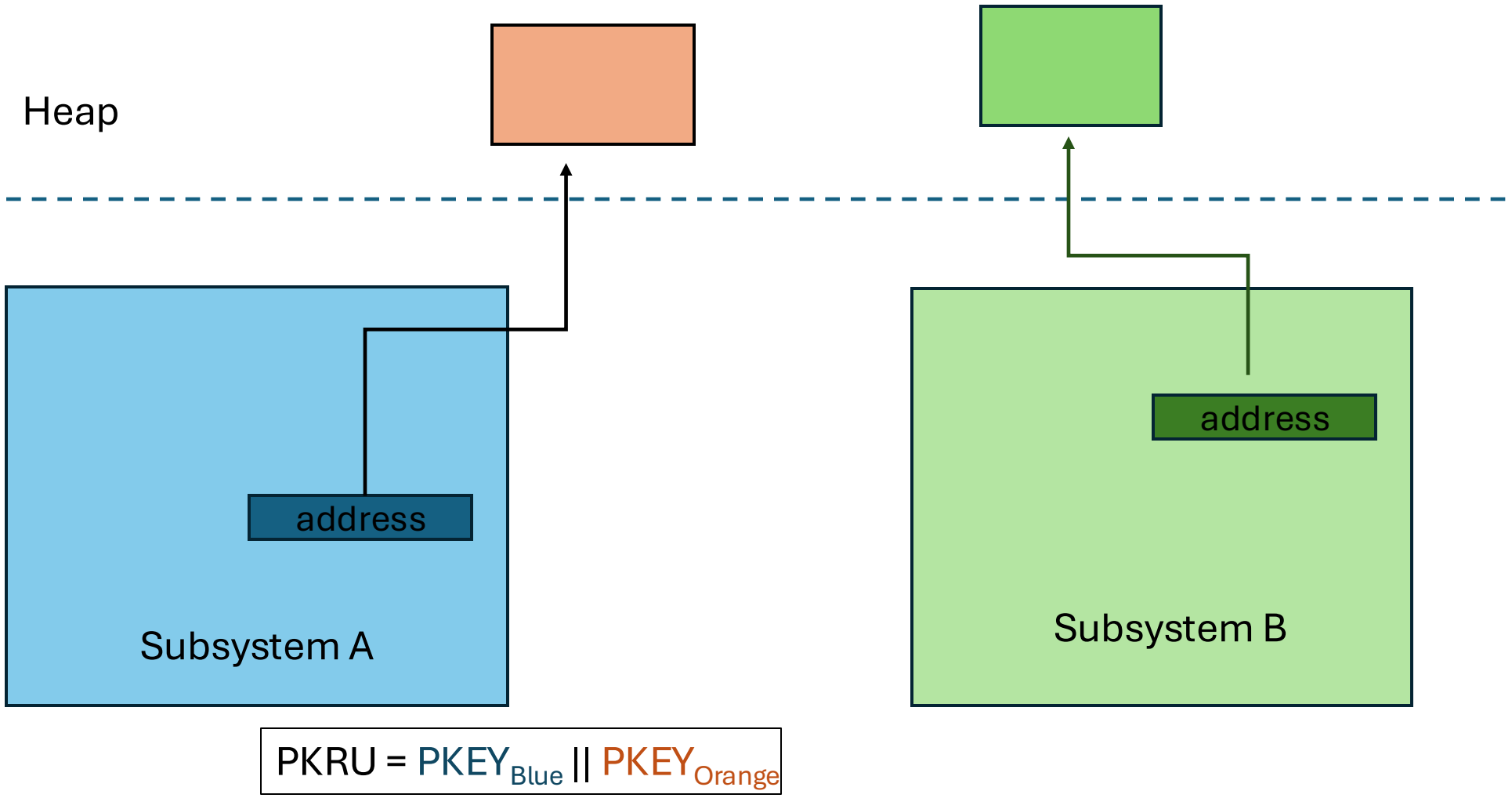

x10
upper bits
lower bits (0 ~ 4GB)
x28
segment
bfi x10, x28, #32, #32
x10
segment
lower bits (0 ~ 4GB)
ldr x10, [x10]

SFI on ARM

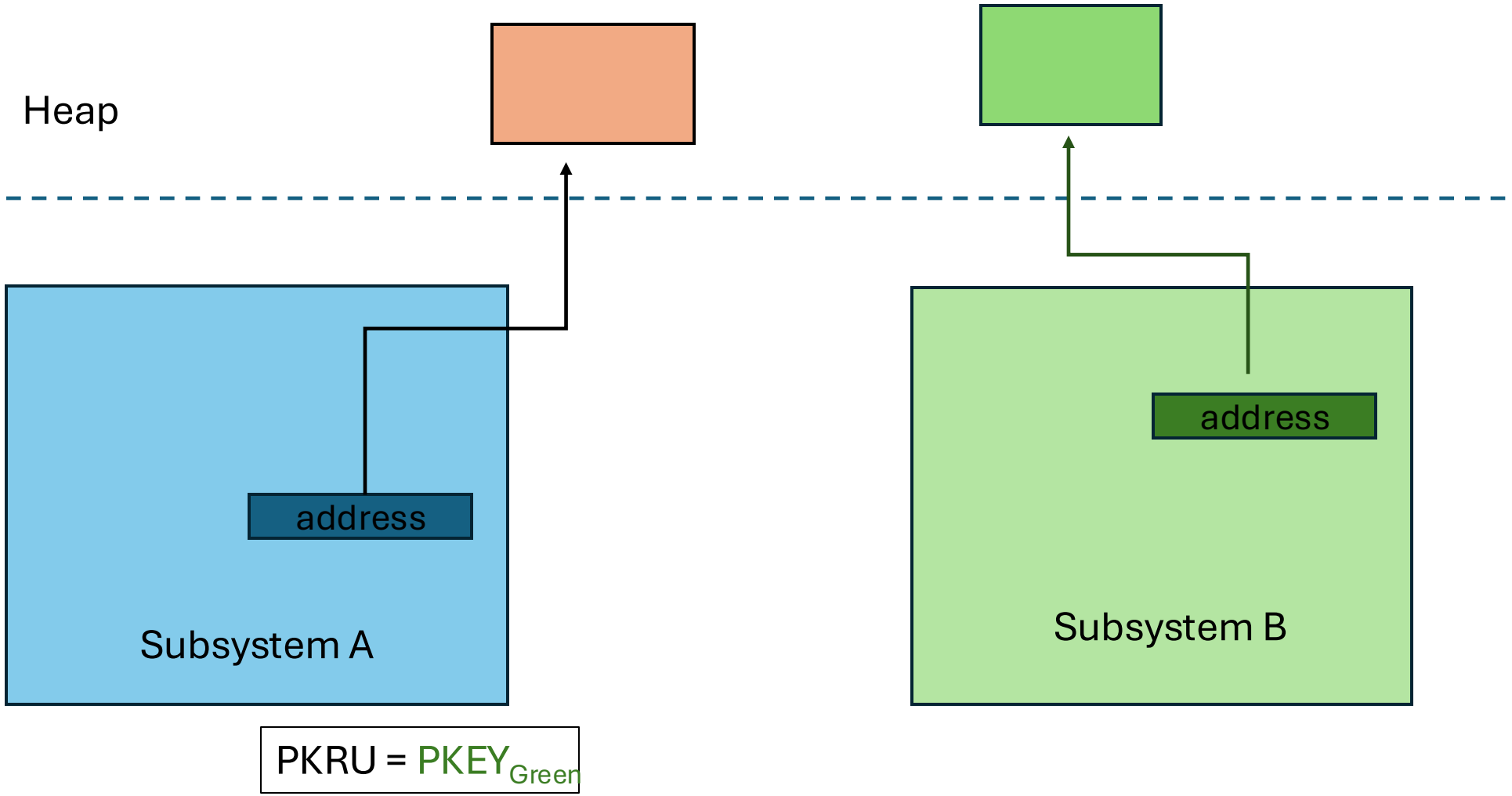
Intel MPK


Subsystem A
Subsystem B
Virtual
D_a
D_a
D_b
D_b
b0100
b0100
b1000
b1000
Page table
$PKEY_{31}$
$PKEY_4$
$PKEY_0$
PKRU
WD
AD
31
4
0

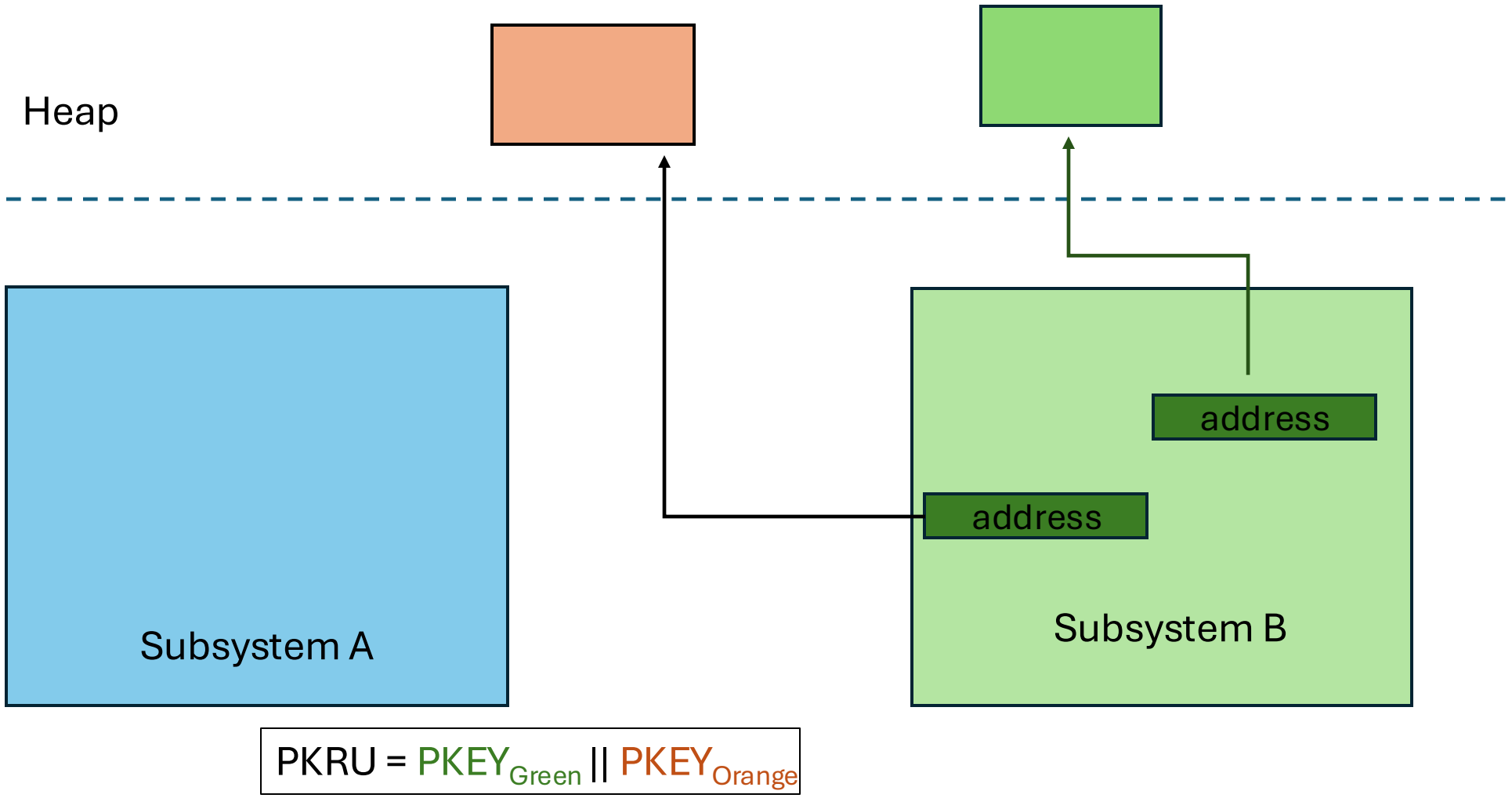
Isolation with MPK



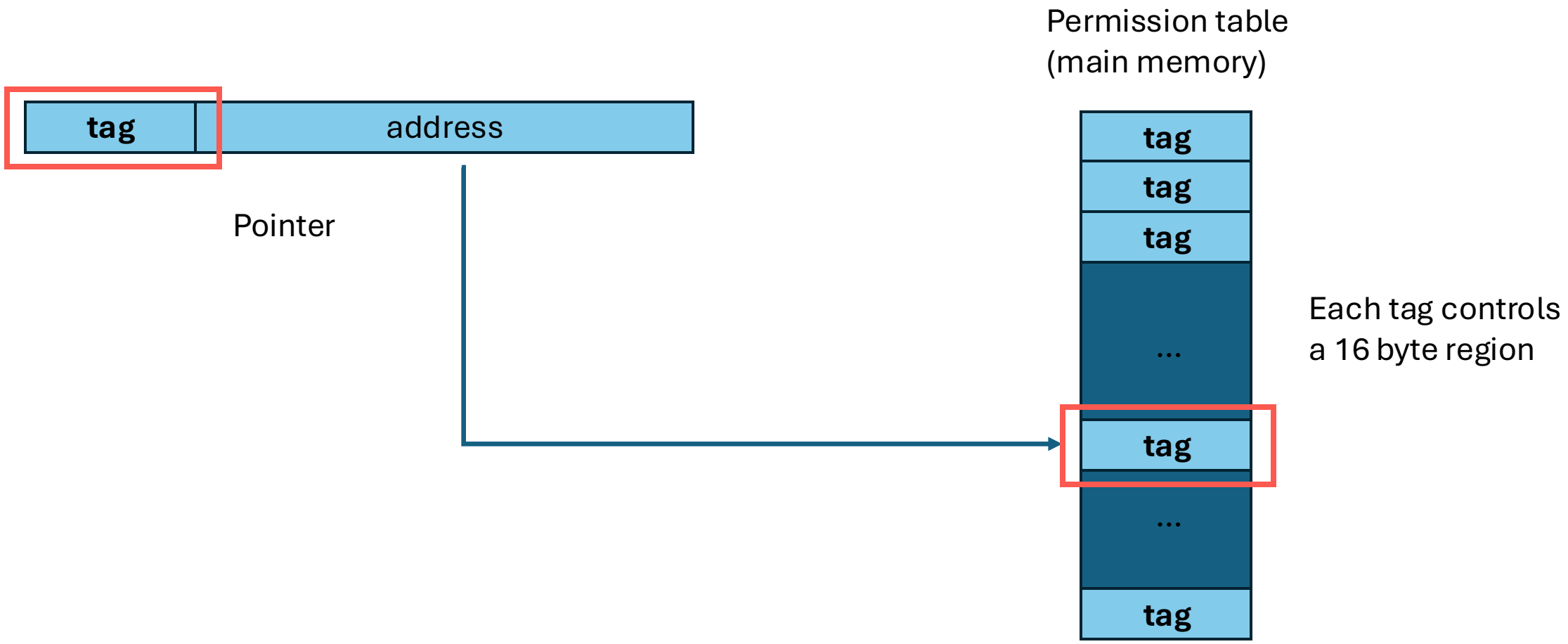
Isolation with MPK



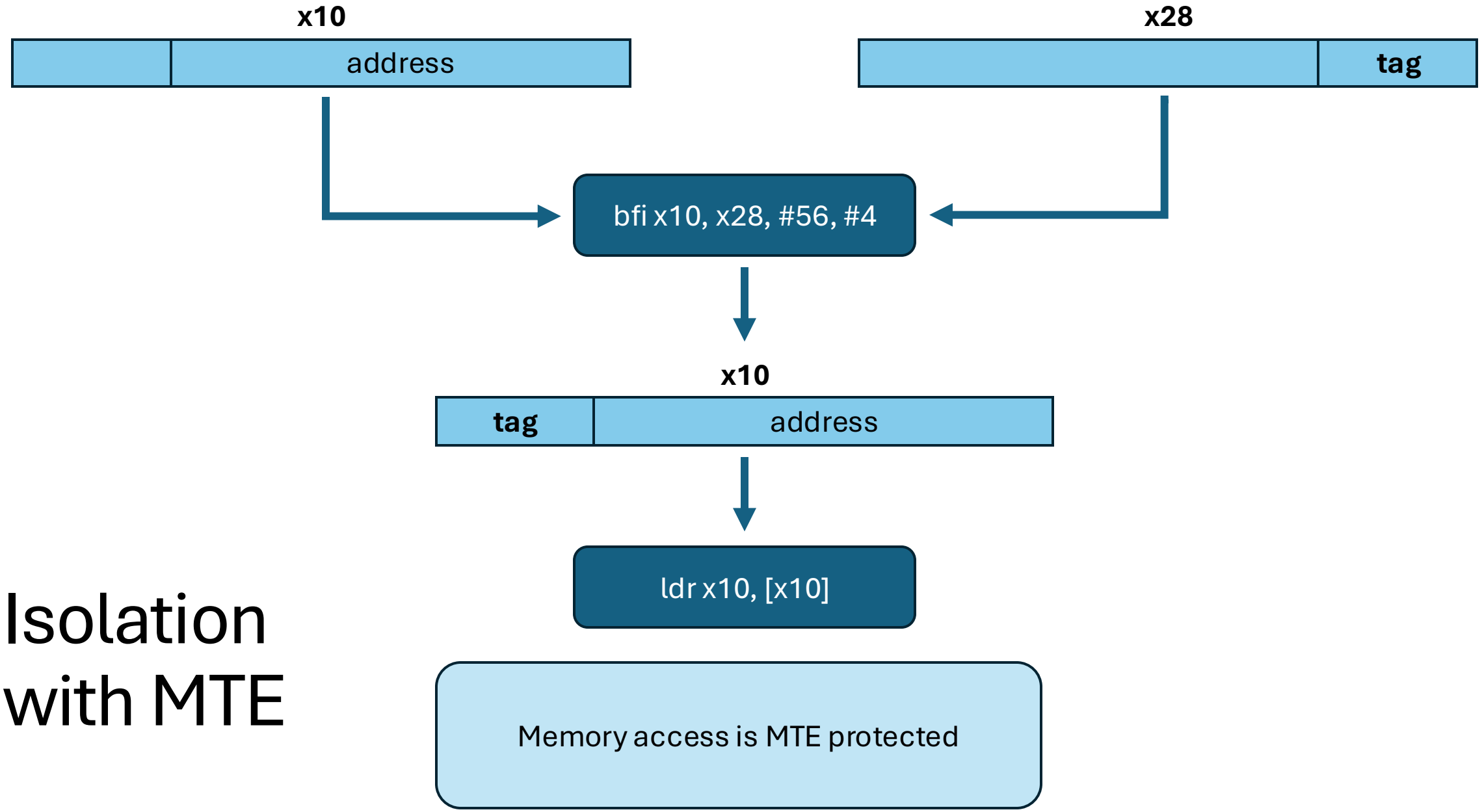
Isolation with MPK



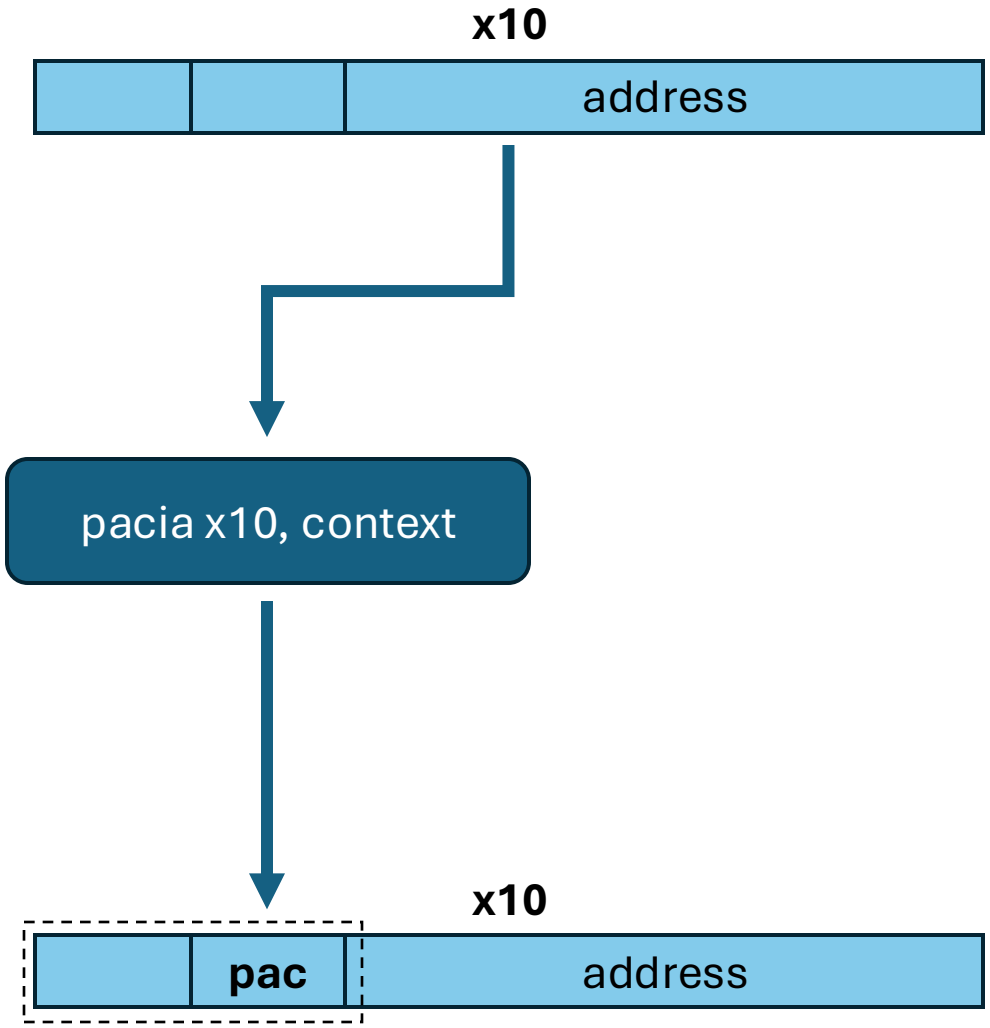
ARM Memory Tagging Extensions (MTE)

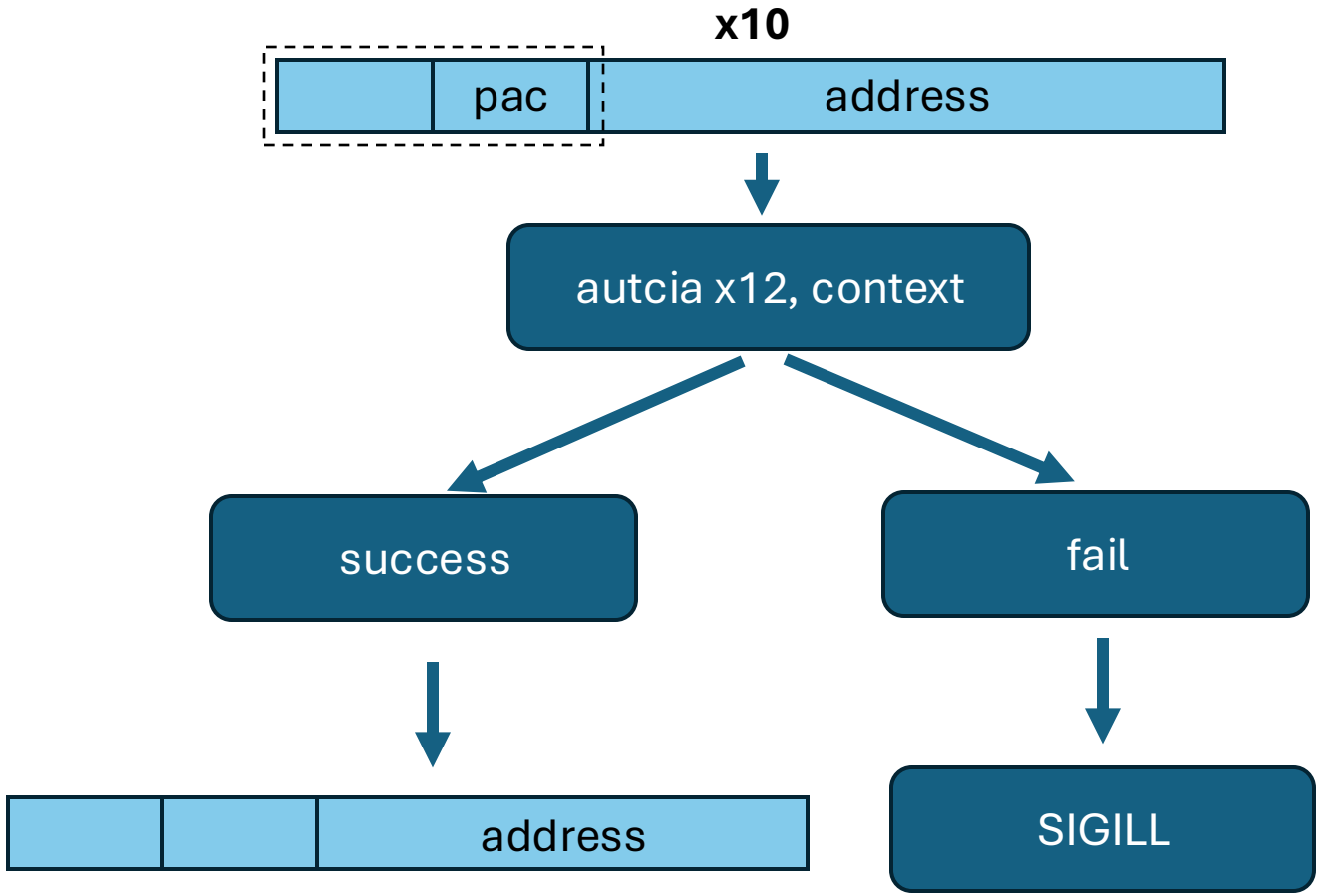


Isolation with MTE

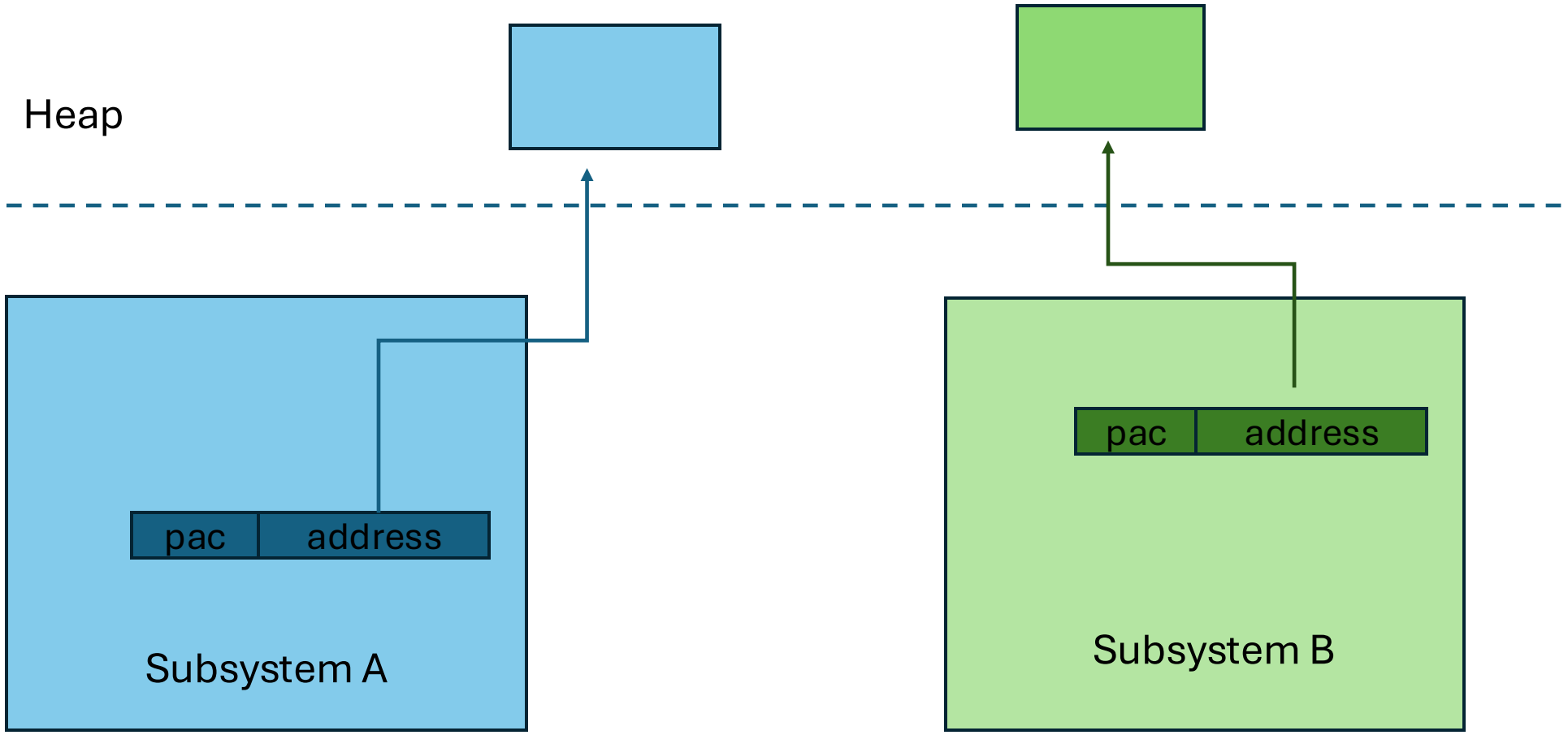


ARM Pointer Authentication (PAC)

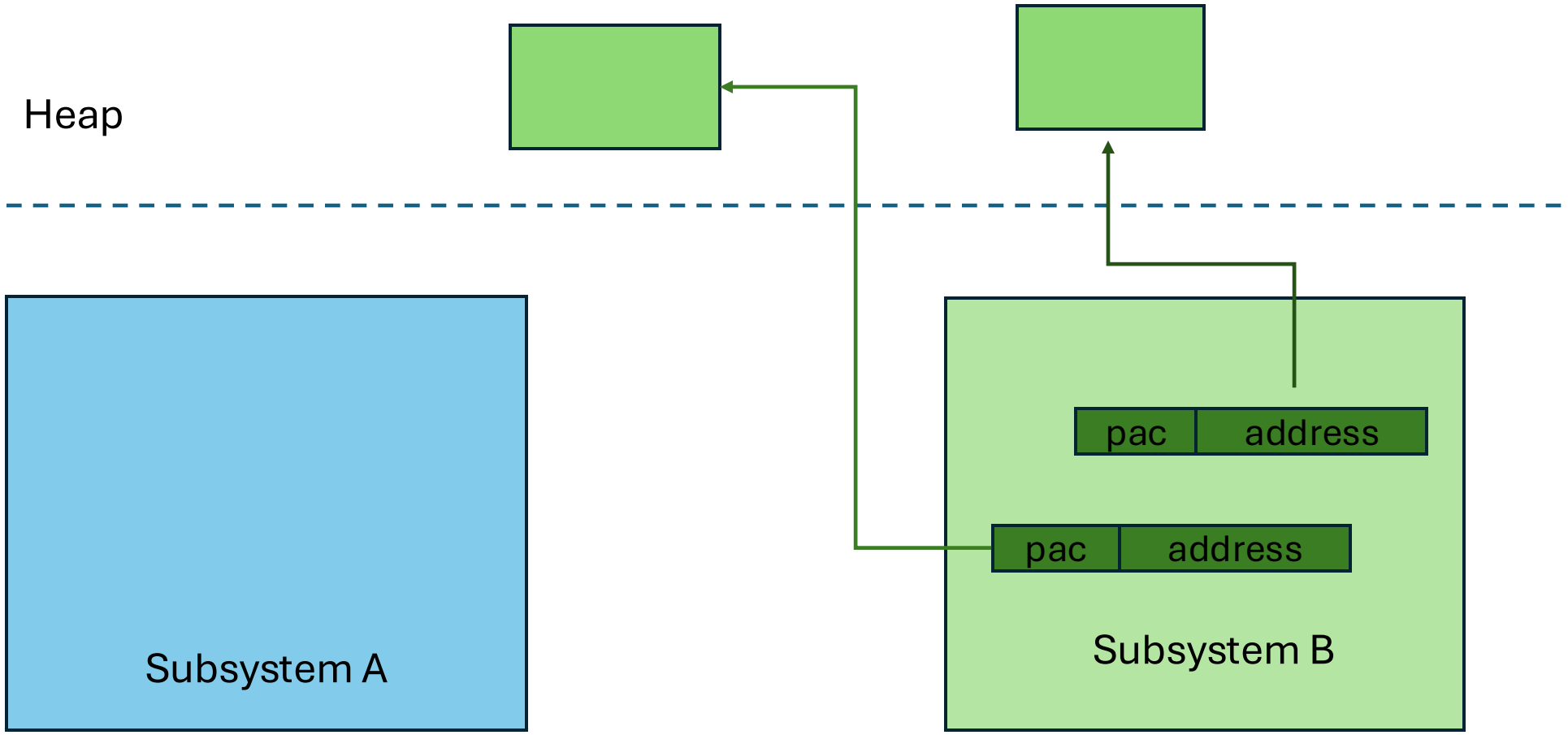


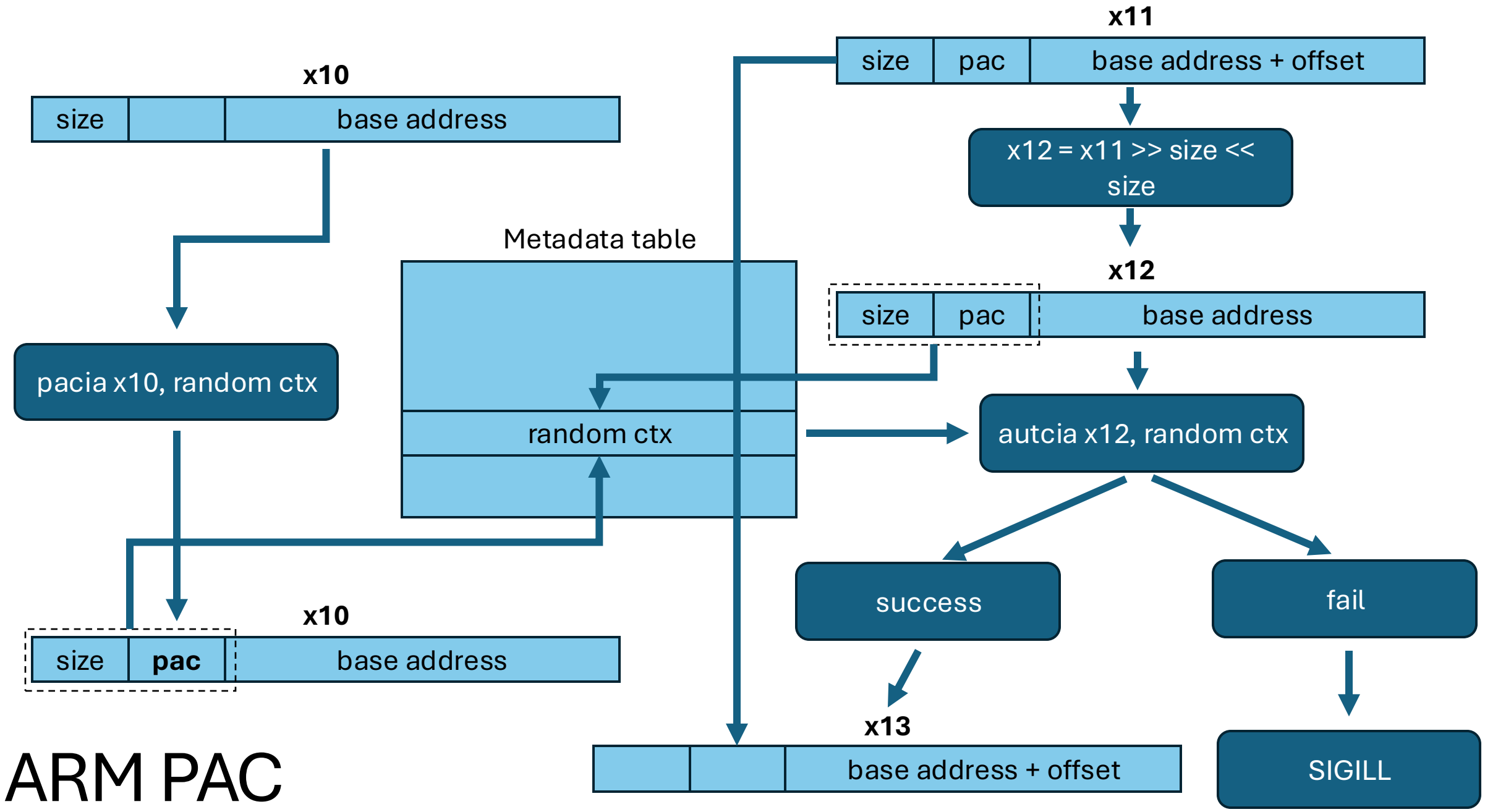


Isolation with PAC



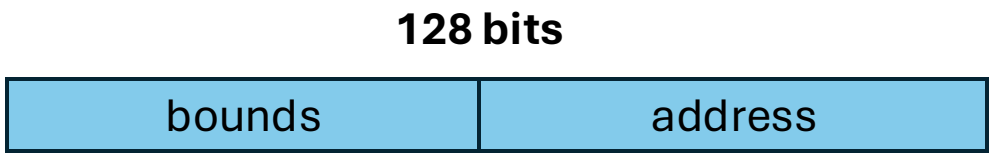
Isolation with PAC



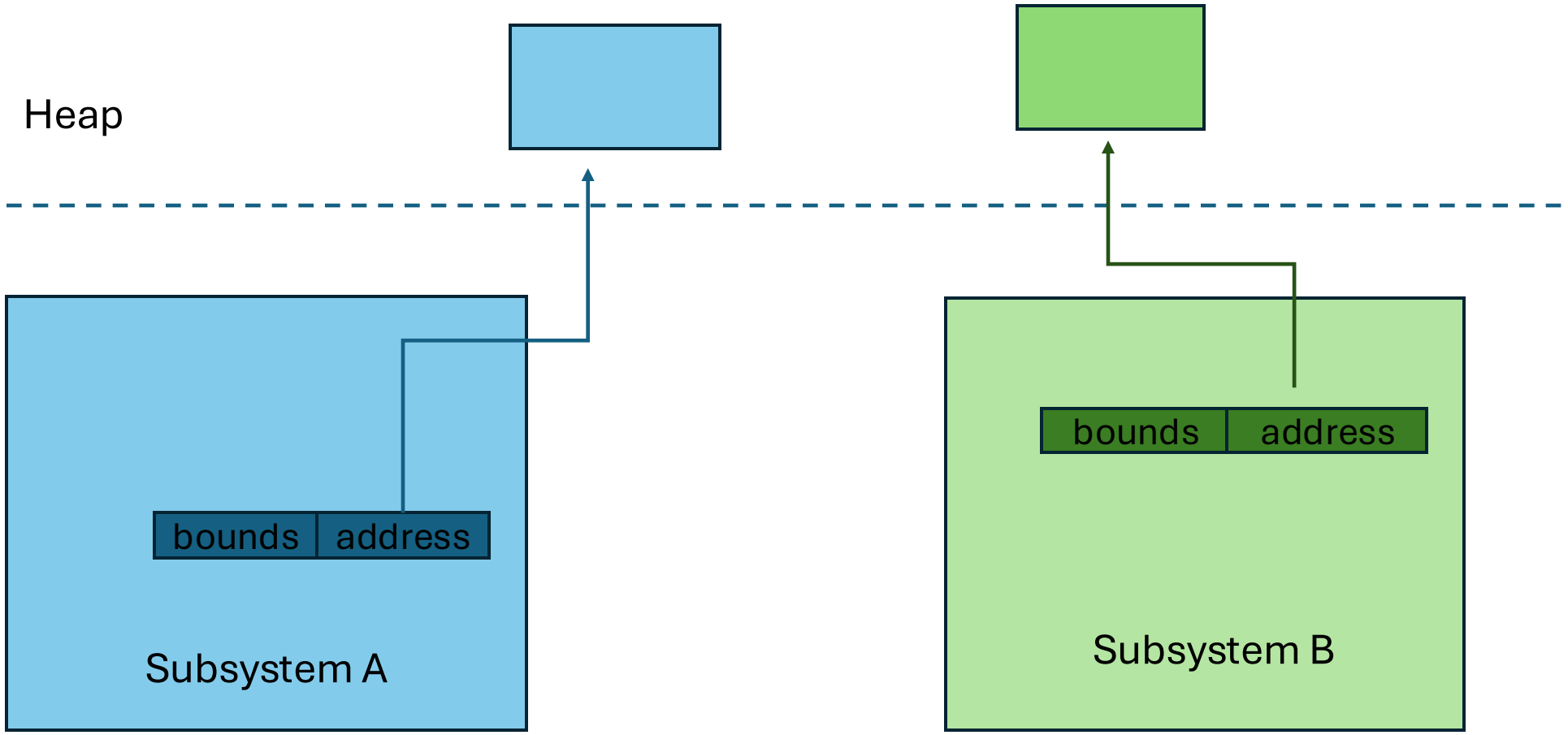

x11
size
pac
base address + offset
x10
size
base address
x12 = x11 >> size << size
Metadata table
x12
size
pac
base address
pacia x10, random ctx
random ctx
autcia x12, random ctx
success
fail
x10
size
pac
base address
x13
base address + offset
SIGILL
ARM PAC

ARM Morello CHERI Capabilities

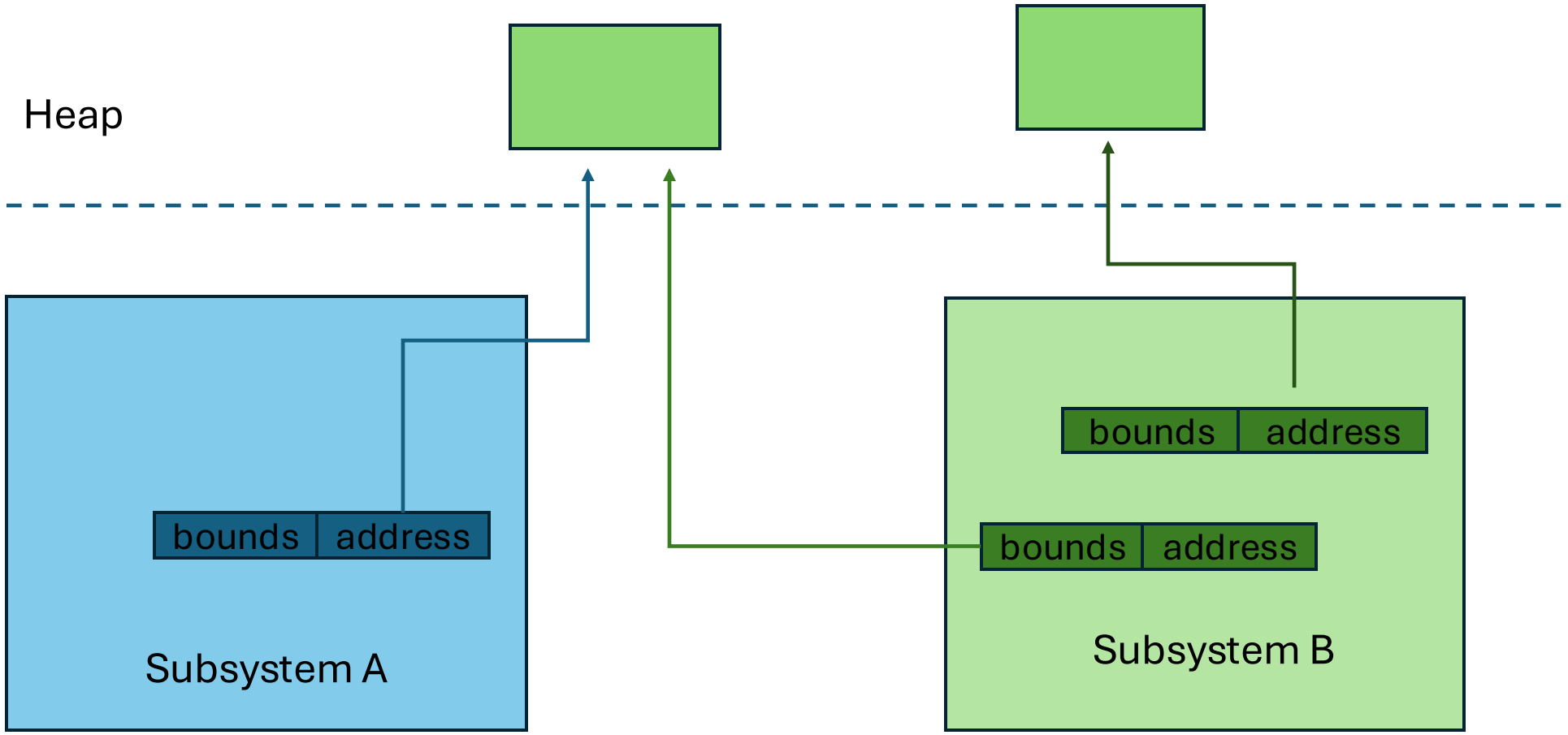
- Morello pointers



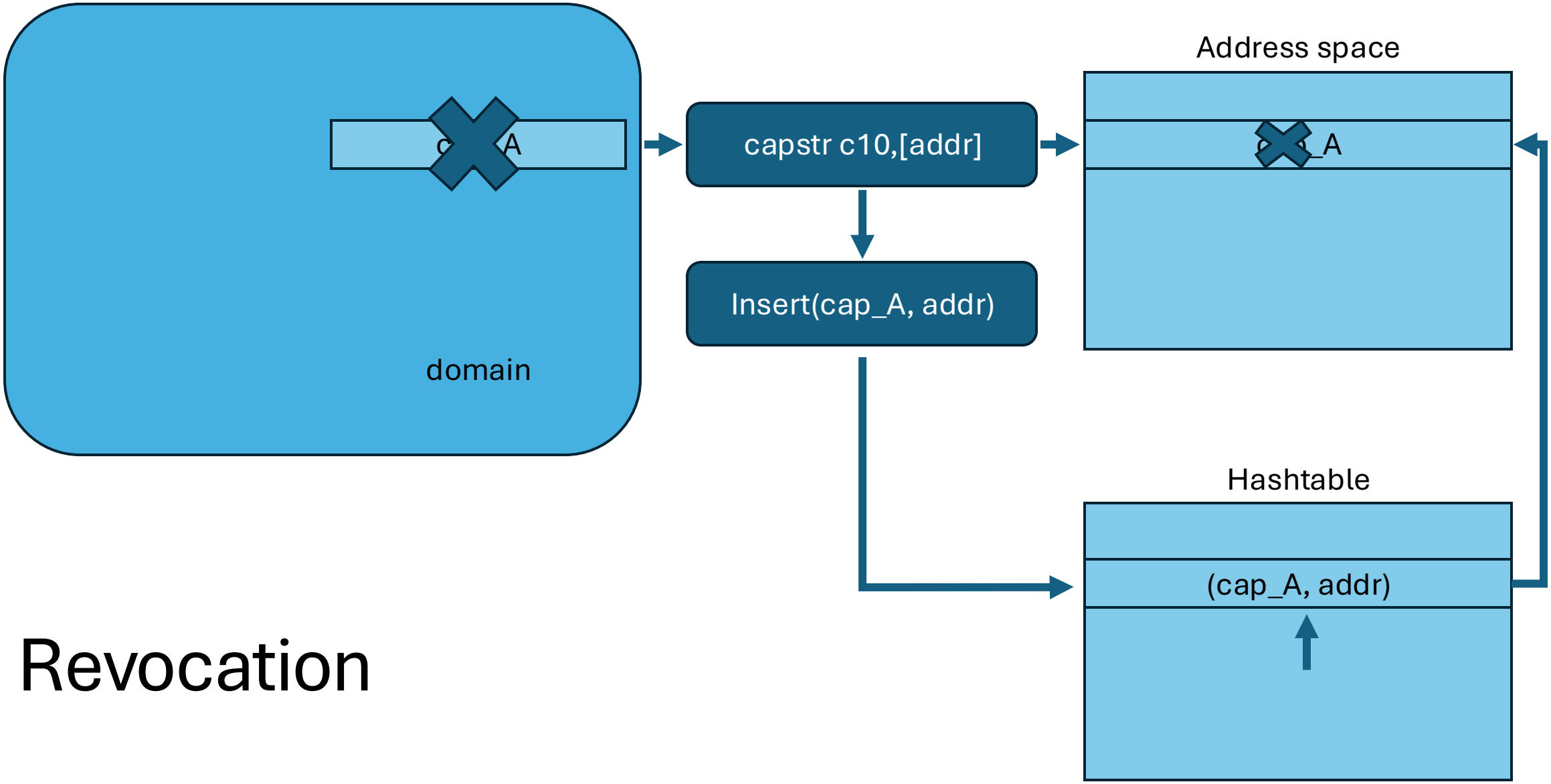
Isolation with Morello



Isolation with Morello



Can we make sure that A cannot access the object any more?


Address space
capstr c10,[addr]
Insert(cap_A, addr)
domain
Hashtable
(cap_A, addr)

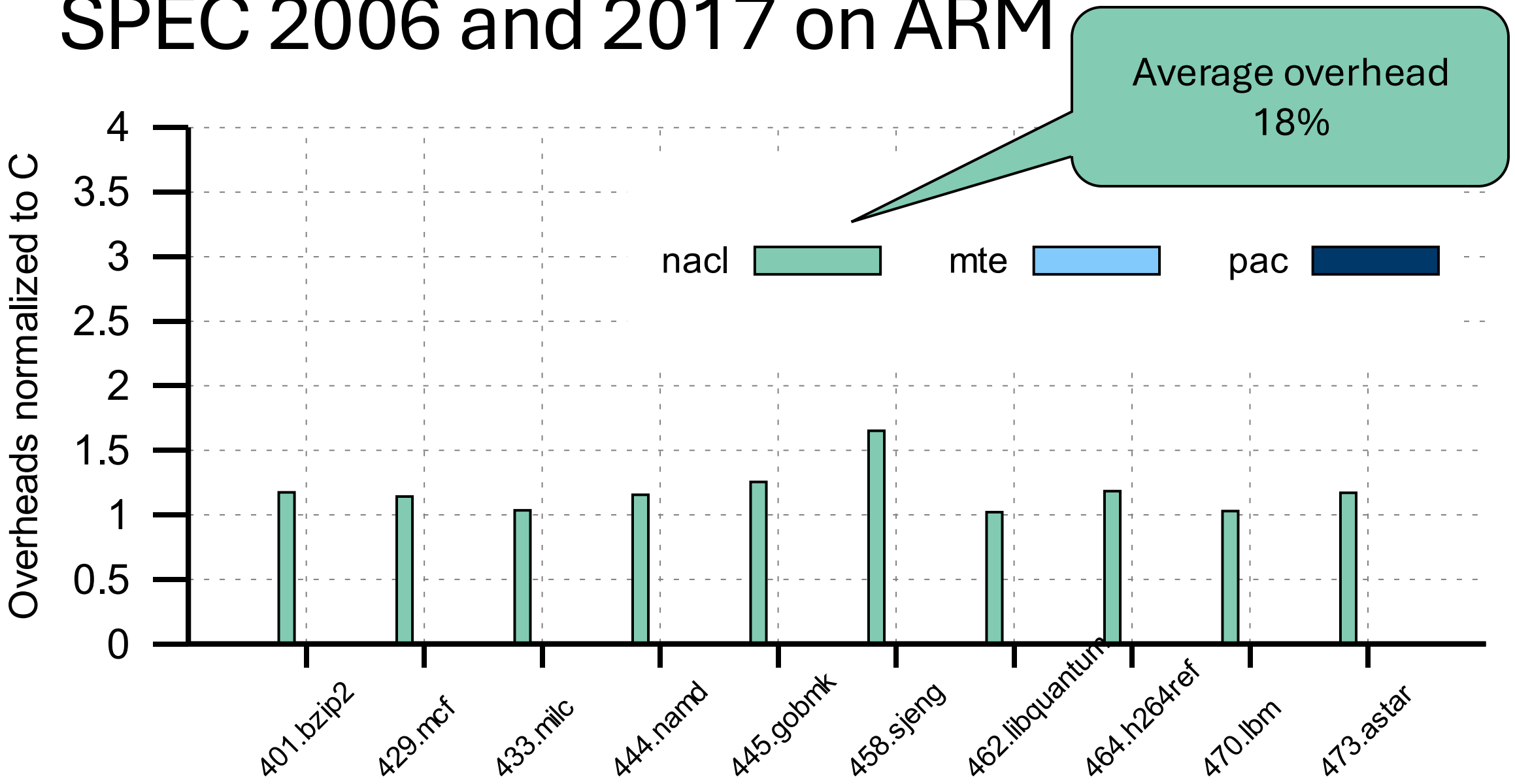
Revocation

Performance

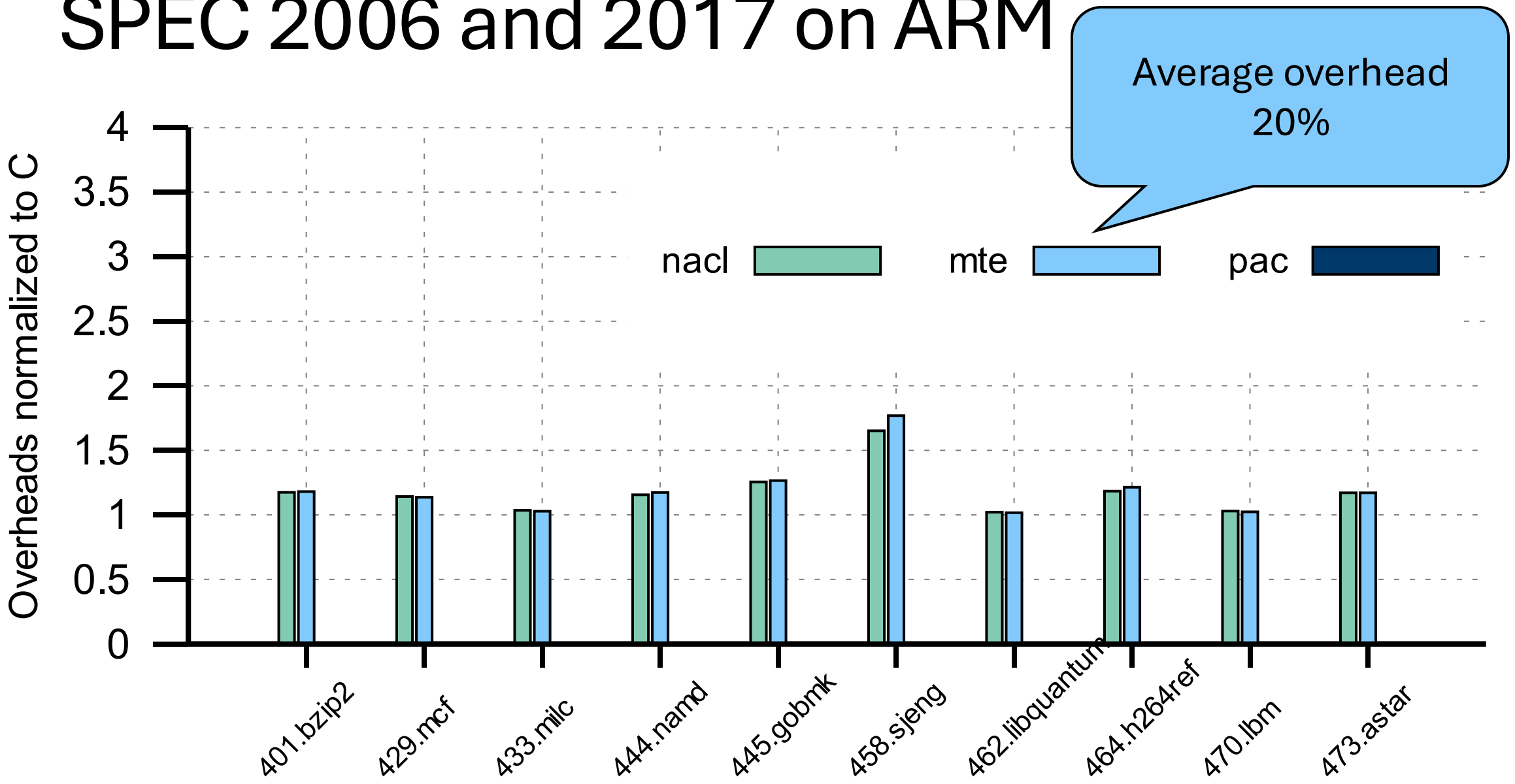
Performance

Overhead of Enforcement

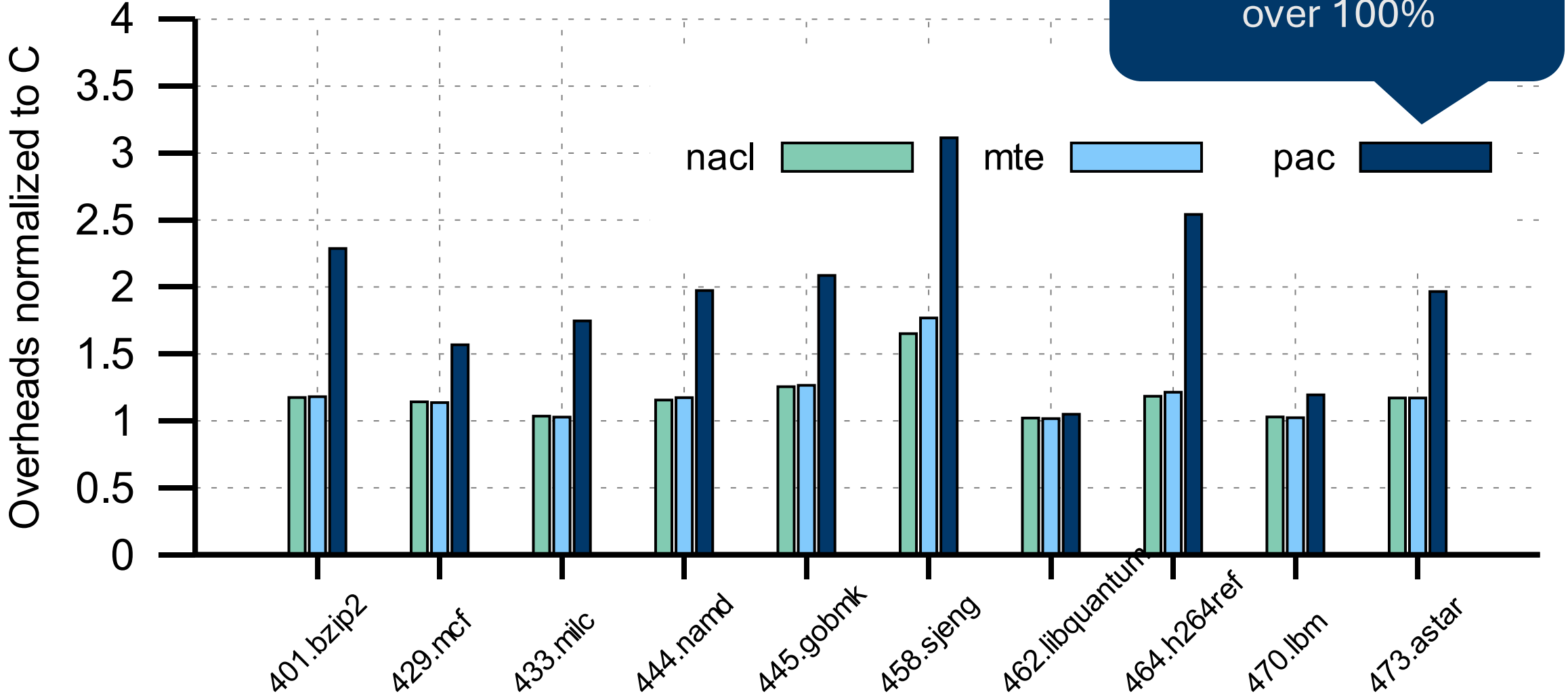
SPEC 2006 and 2017 on ARM



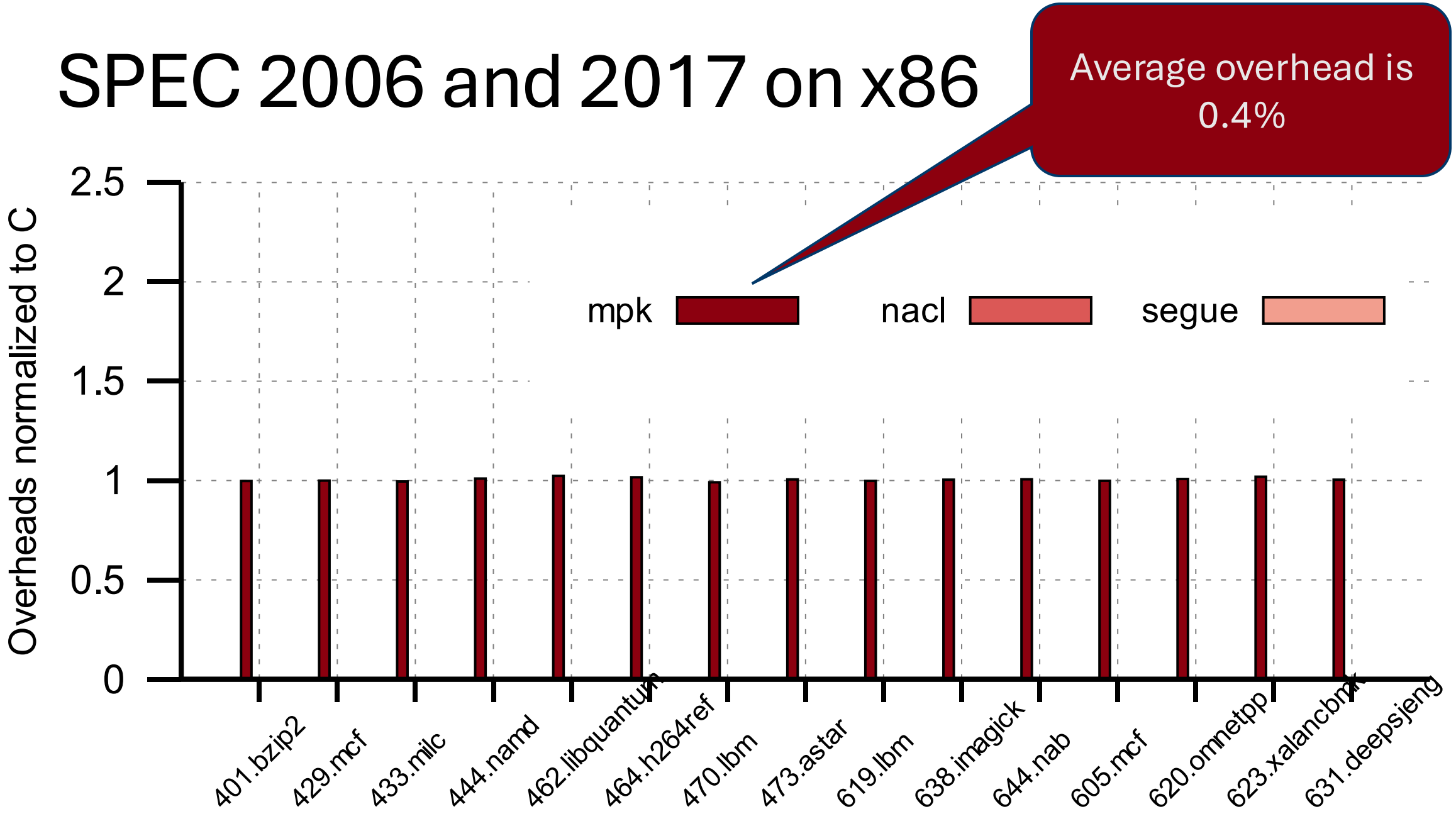
SPEC 2006 and 2017 on ARM



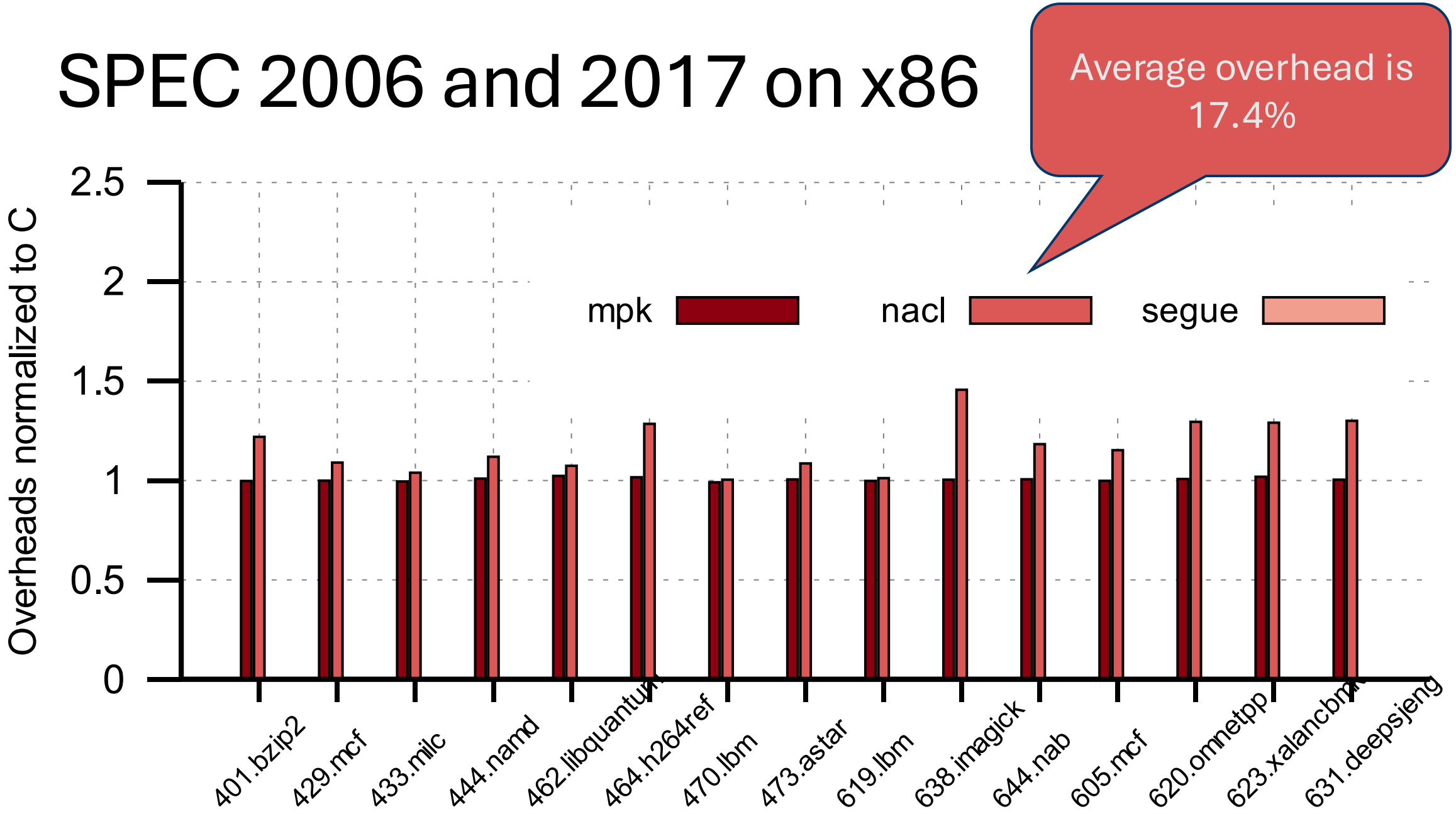
SPEC 2006 and 2017 on ARM



SPEC 2006 and 2017 on x86



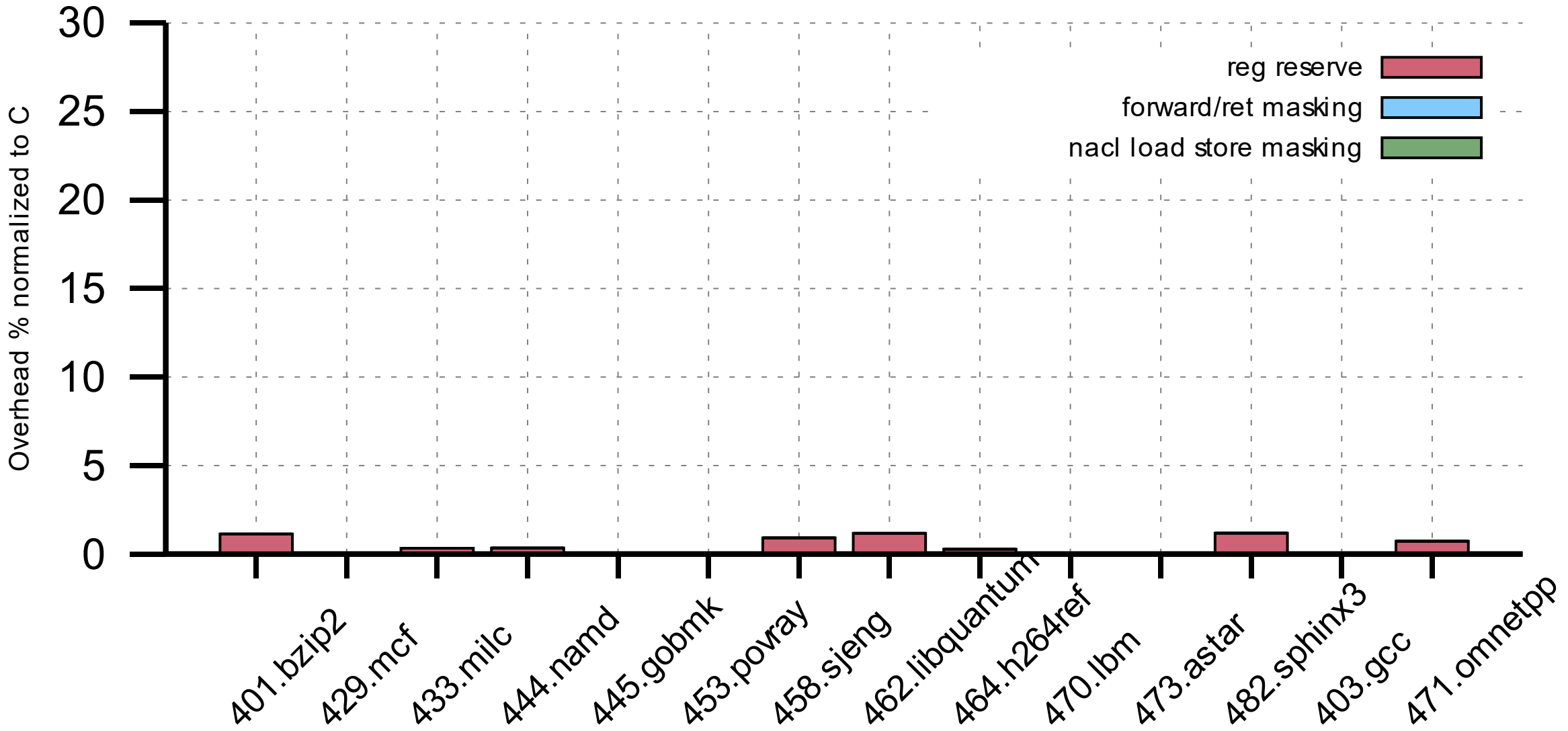
SPEC 2006 and 2017 on x86



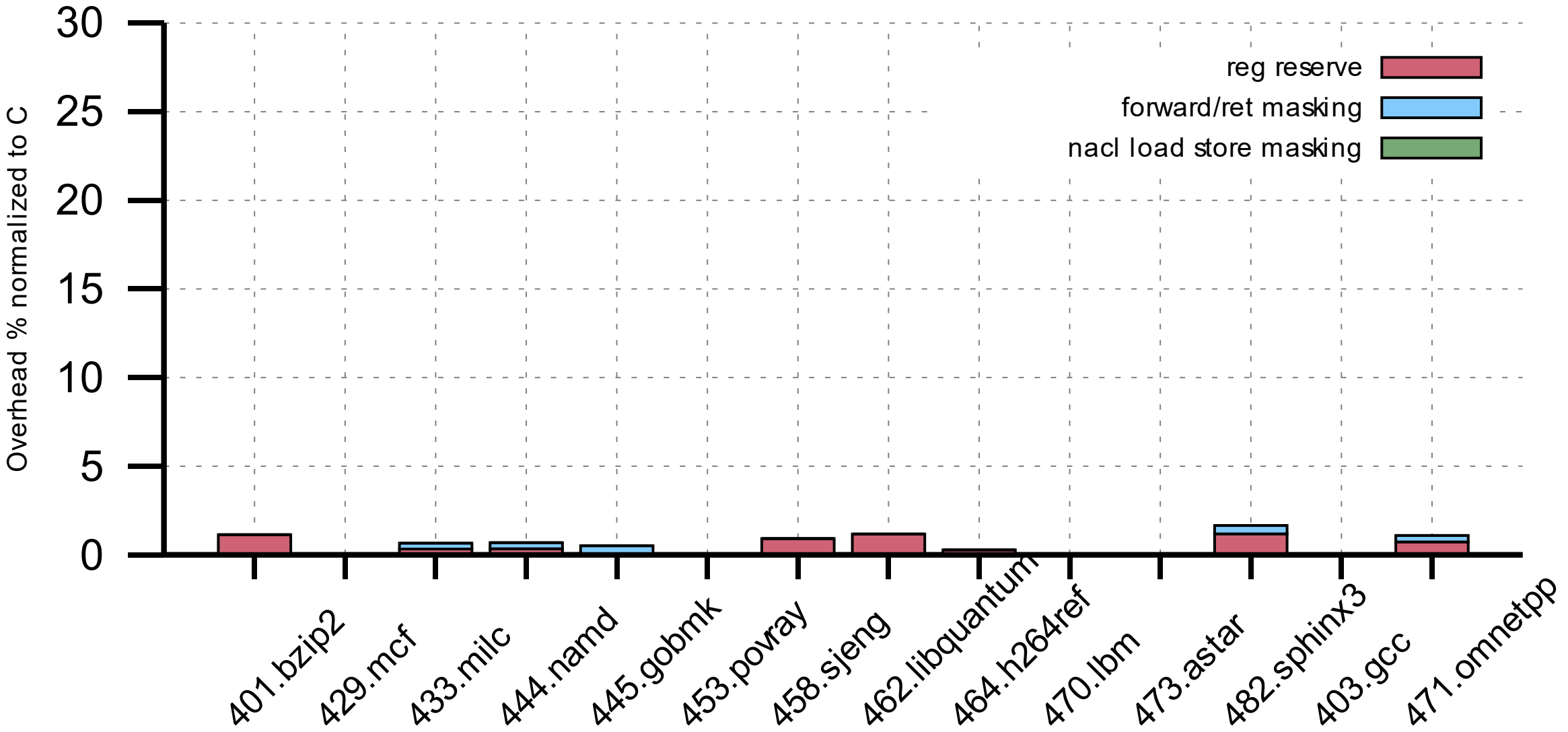
What are the reasons?

- Software SFI scheme
- Similar to Google NaCl
  - Instruction bundling
  - Software enforcement of bounds

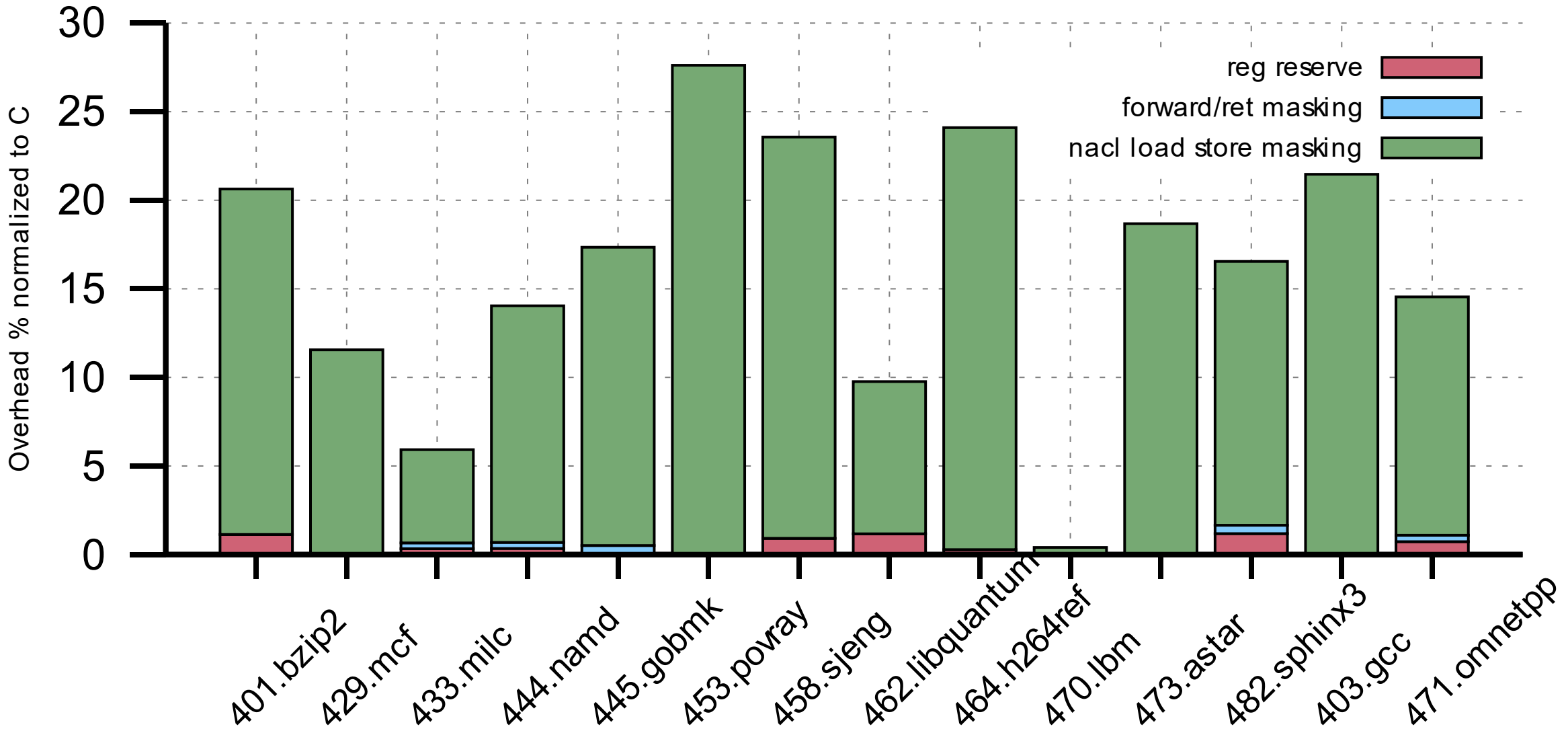
SPEC 2006 Performance Breakdown on x86



SPEC 2006 Performance Breakdown on x86



SPEC 2006 Performance Breakdown on x86



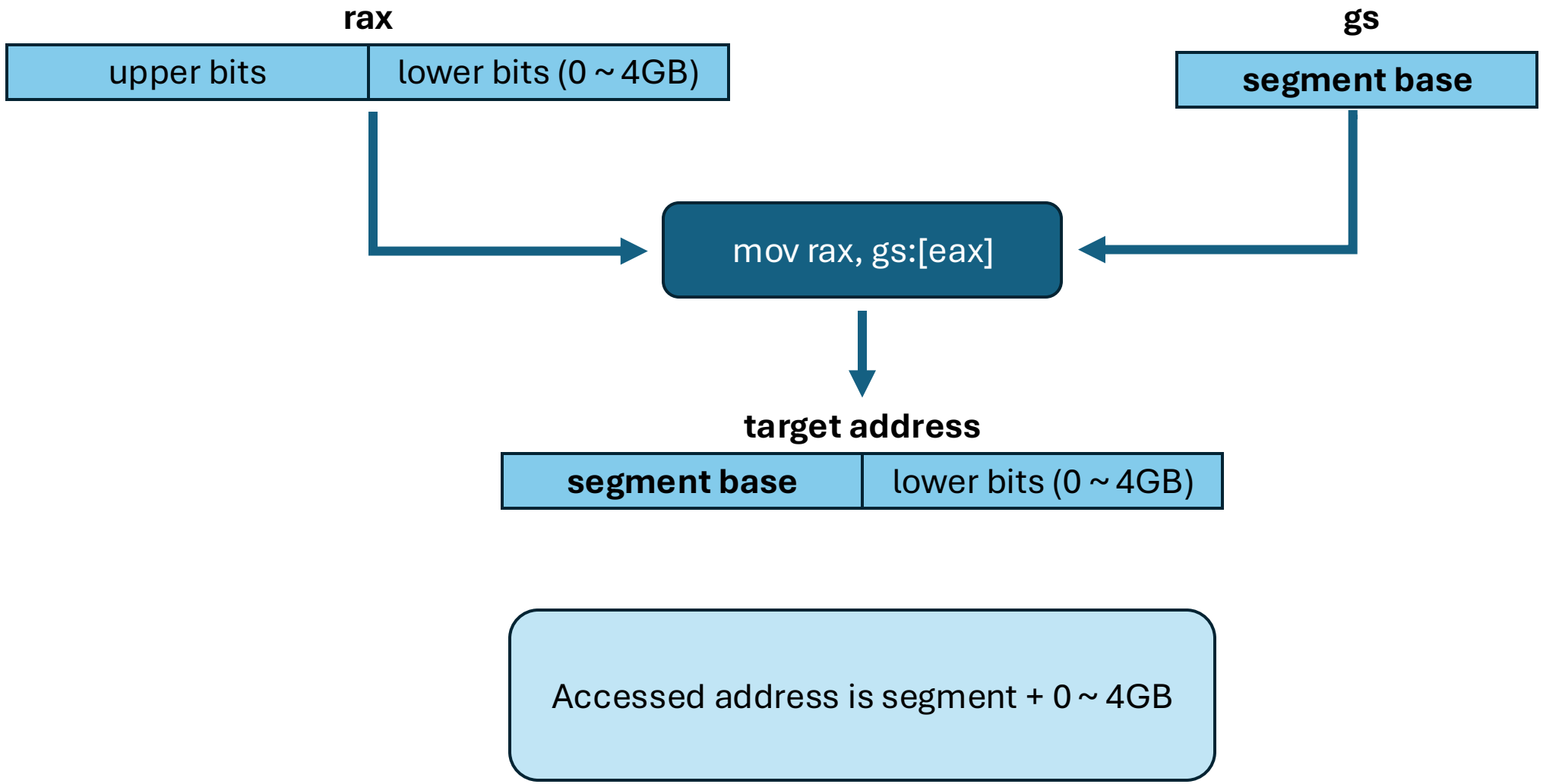
Observation

- MTE + CET is the only scheme with low overhead
- Isolation schemes are inherently limited by the overhead of software instrumentation
  - Even a single instruction that enforces an MTE tag adds prohibitive overhead
  - It’s on the critical path of the pipeline

Possible solutions

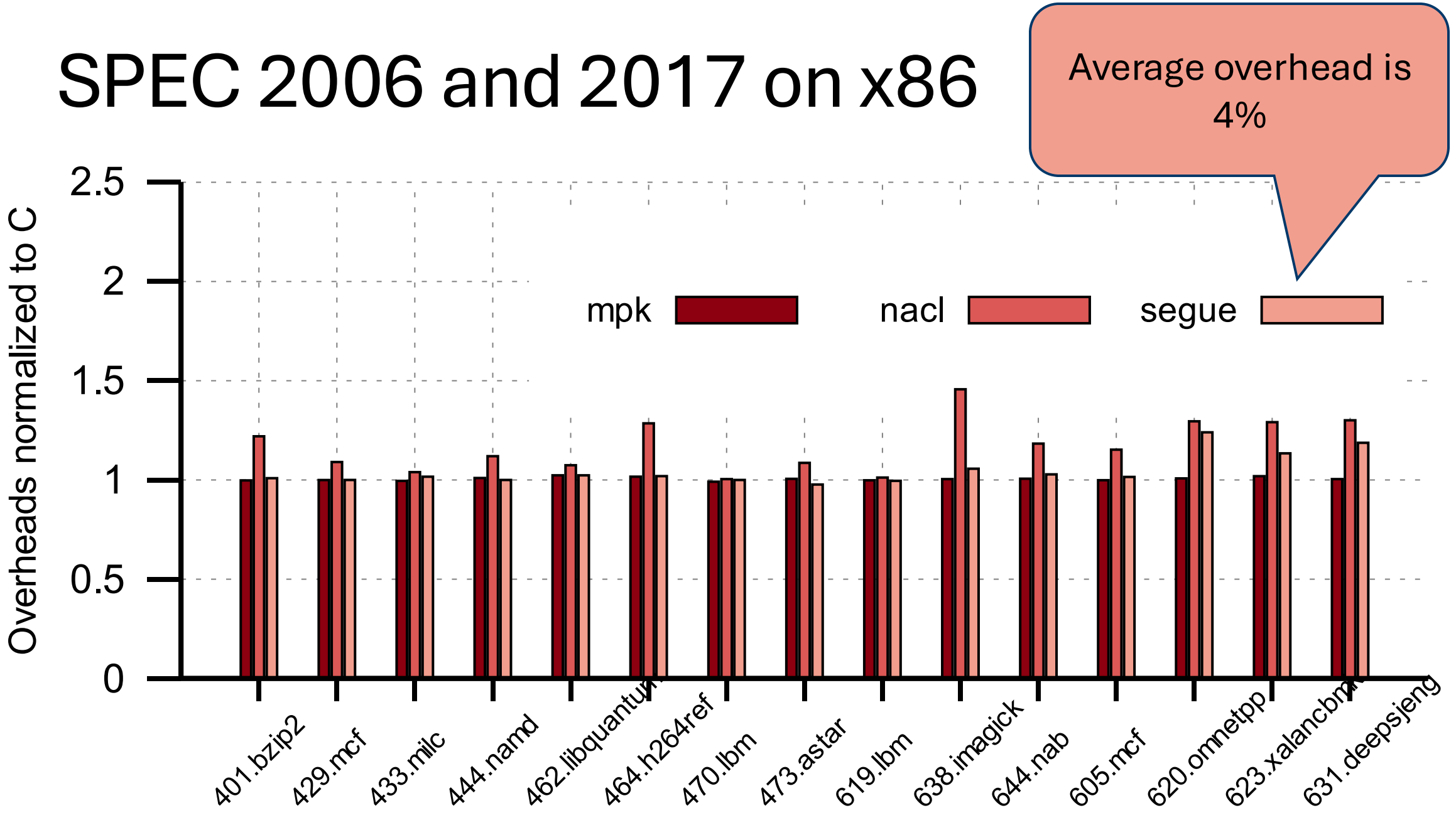
- x86 hardware support for segment enforcement
  - gs trick on x86 (Segue[1])

[1] Shravan Narayan et al. Segue and colorguard: Optimizing SFI performance and scalability on modern x86. In *Proceedings of Workshop on Programming Languages and Analysis for Security (PLAS)*, 2022.




```
1  ; load value at [rax + rbx] to rcx
2  mov rcx, gs:[eax, ebx, 1] ;memory access within [gs + 0-8GB]
```

SPEC 2006 and 2017 on x86



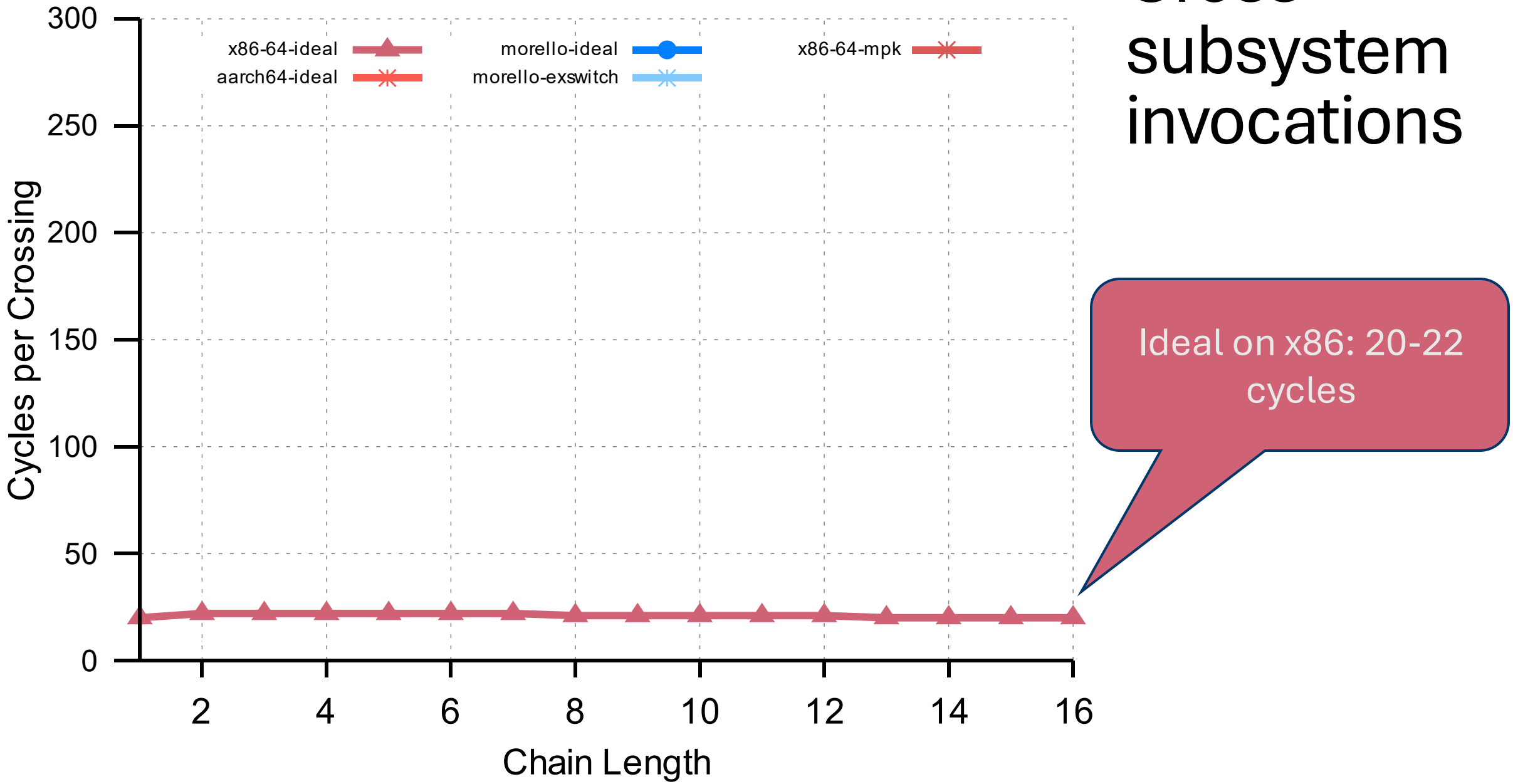
Possible solutions

- ARM MTE
  - Protected hardware register that holds the tag
  - Simulated: 2% overhead on SPEC (CFI enforcement and reservation of one register)

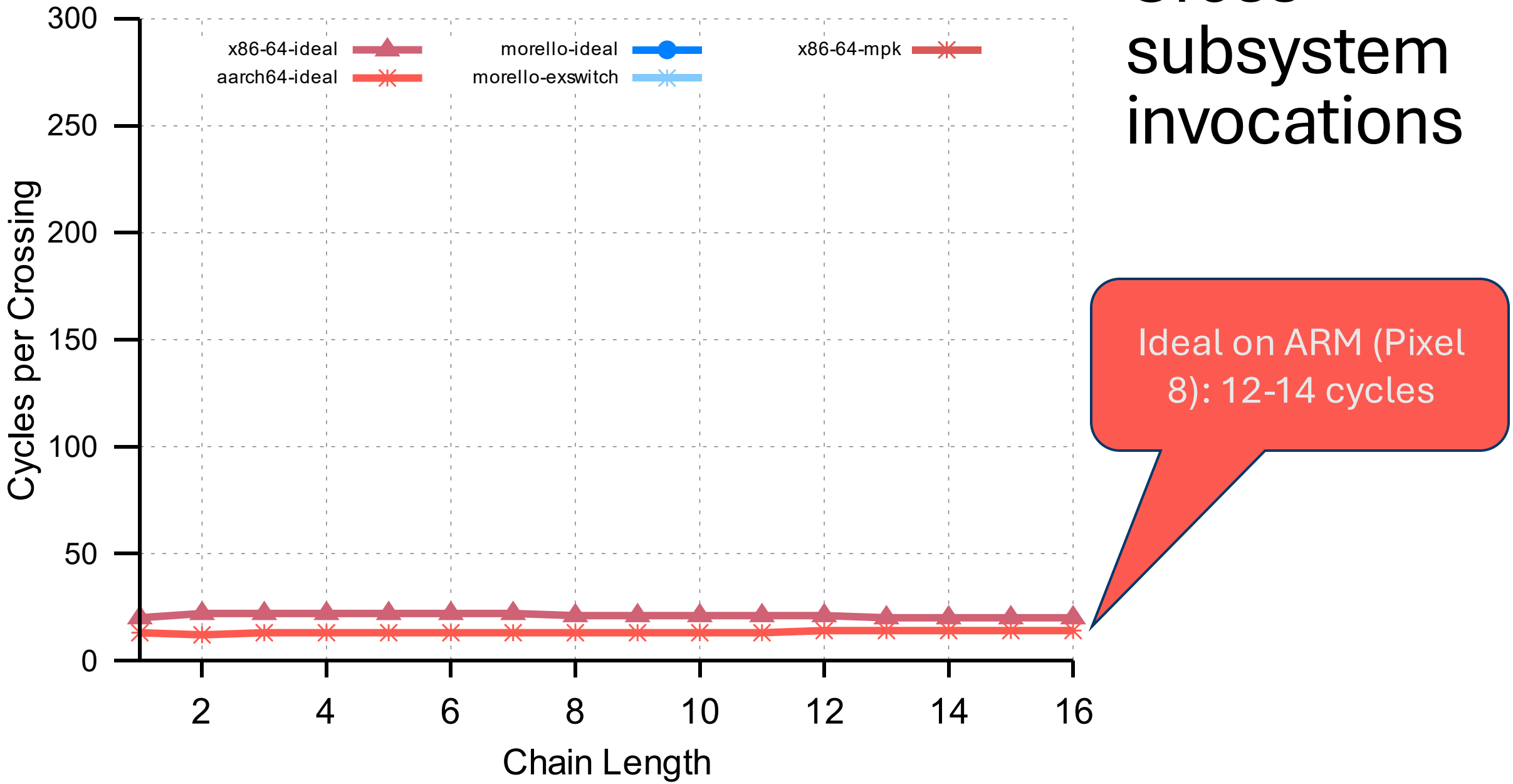
Performance

Cross-Subsystem Invocations

Cross-subsystem invocations

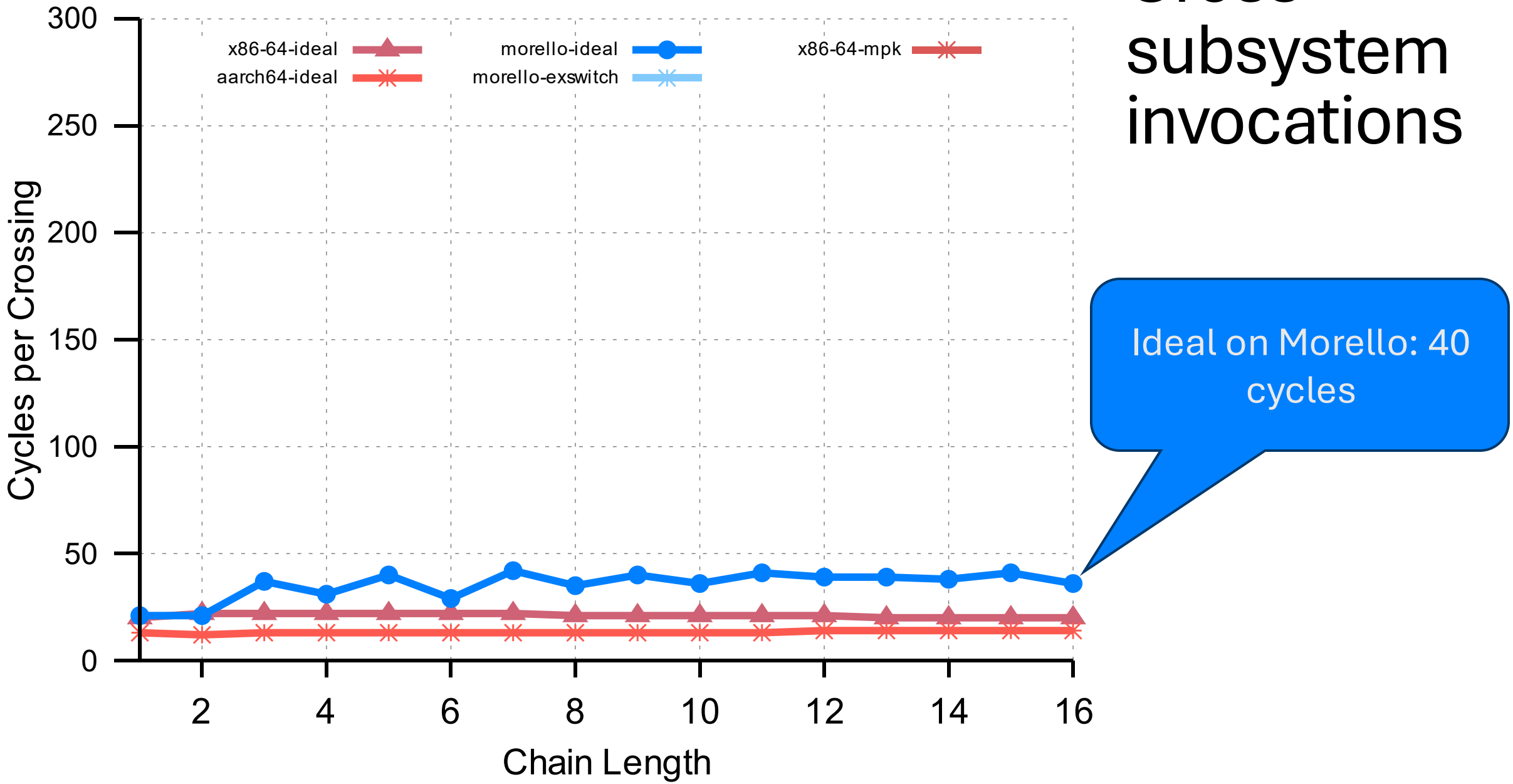


Cross-subsystem invocations

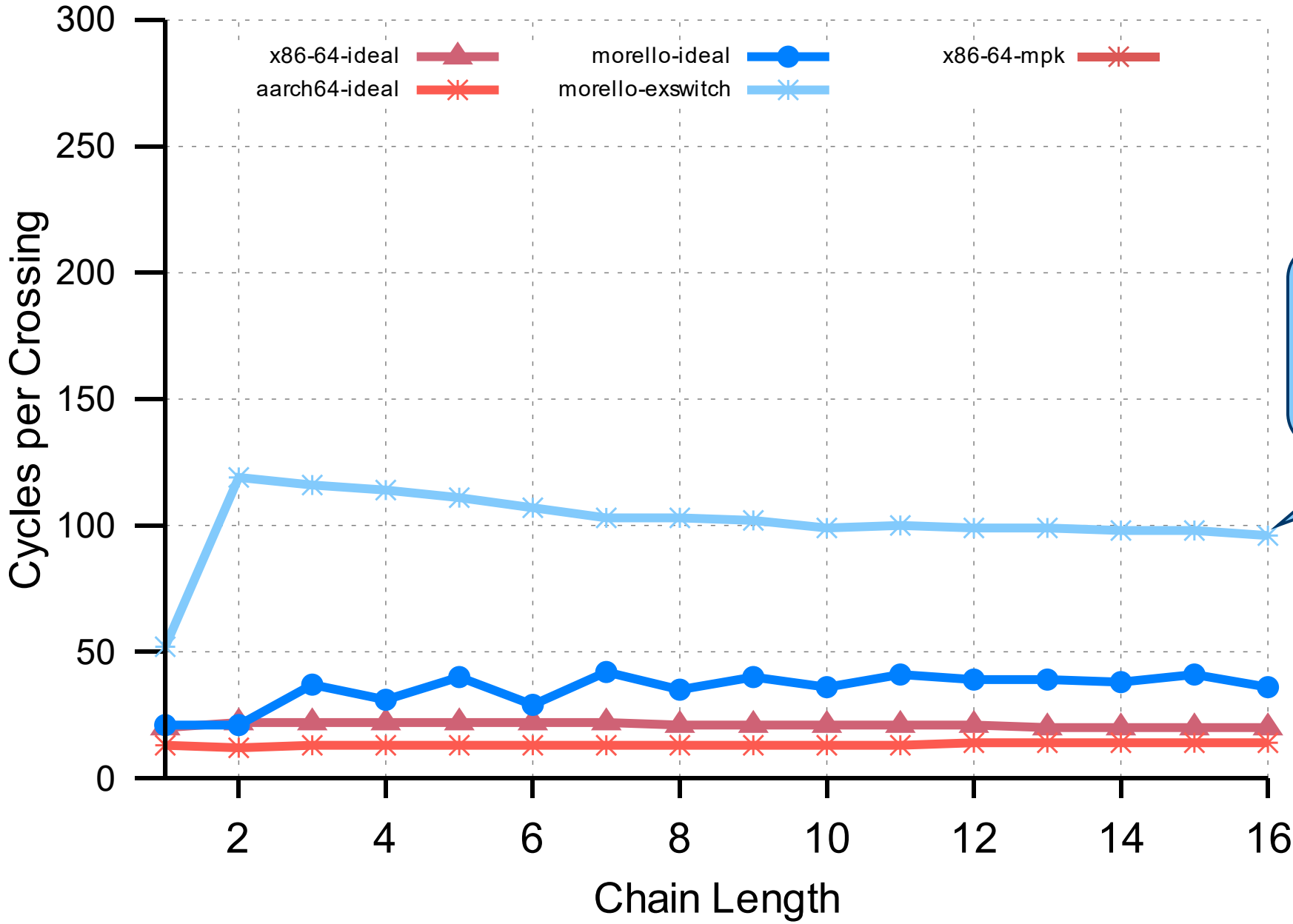


Ideal on ARM (Pixel 8): 12-14 cycles

Cross-subsystem invocations

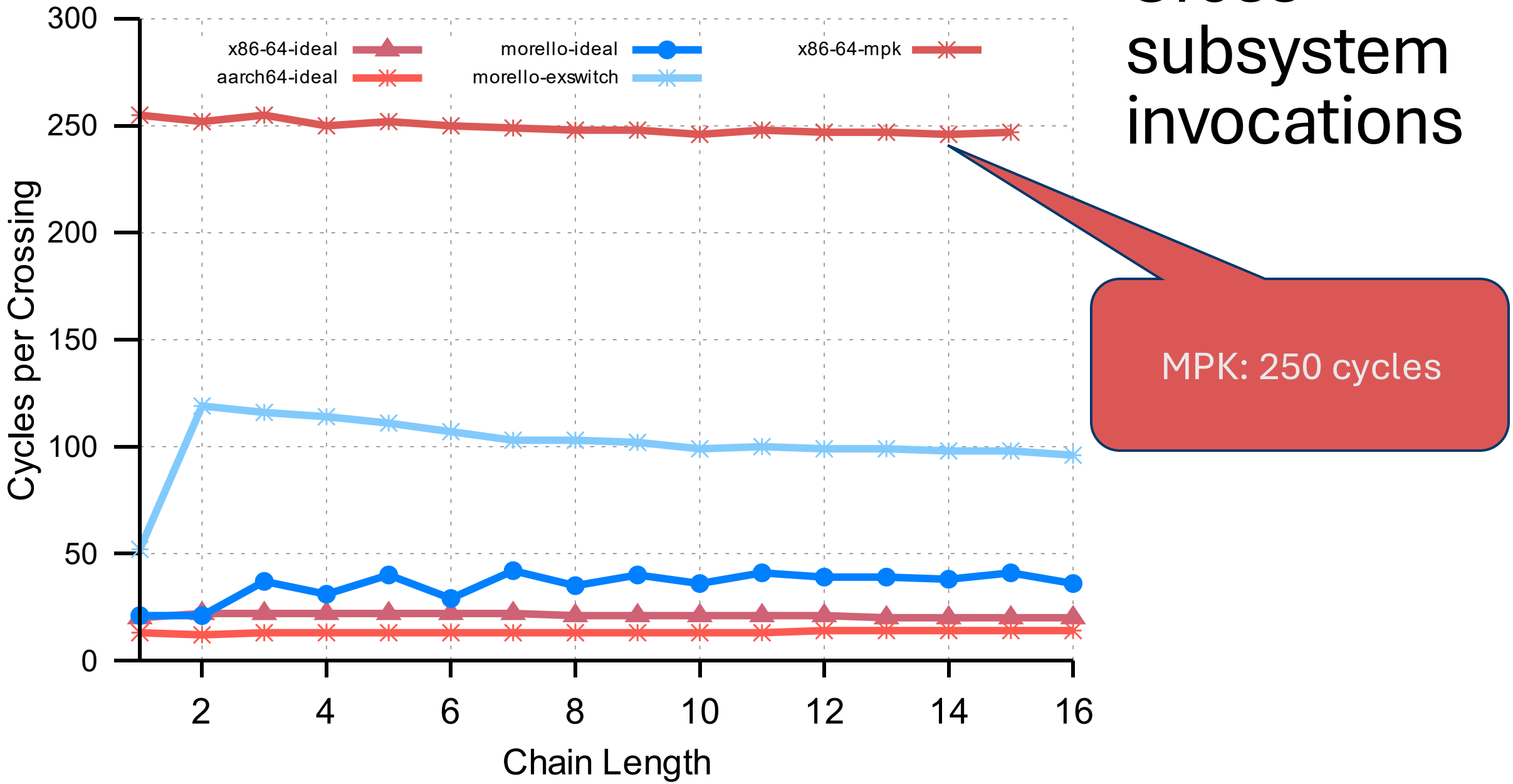


Cross-subsystem invocations



Morello switch to executive mode: 50-120 cycles

Cross-subsystem invocations



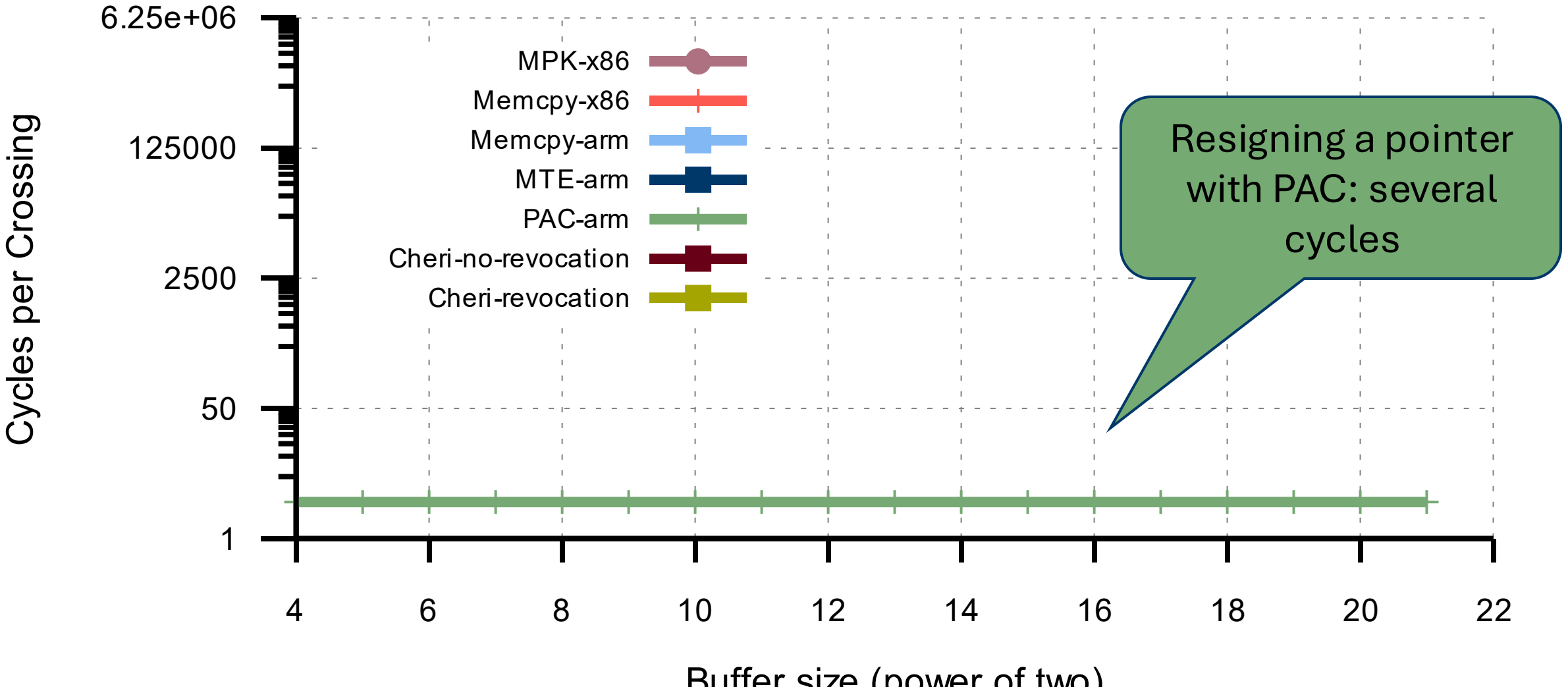
Observation

- We are far from an ``ideal'' cost of function invocation
- Most overhead
  - The cost of changing the hardware isolation boundary
  - The cost of saving and restoring general and extended registers
    - 137 cycles on x86
    - 40 cycles on Morello
    - 14 cycles on ARM (good!)

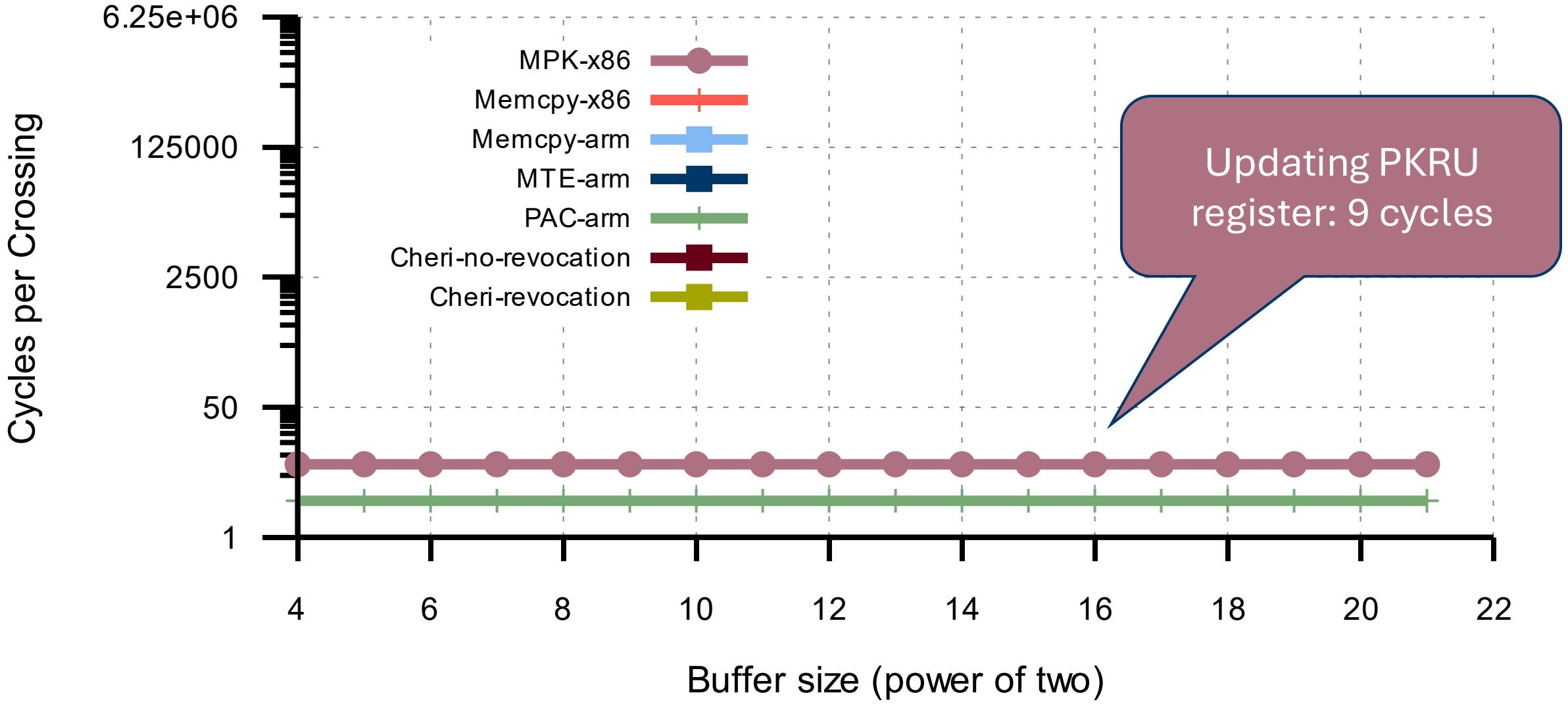
Performance

Passing data (zero copy)

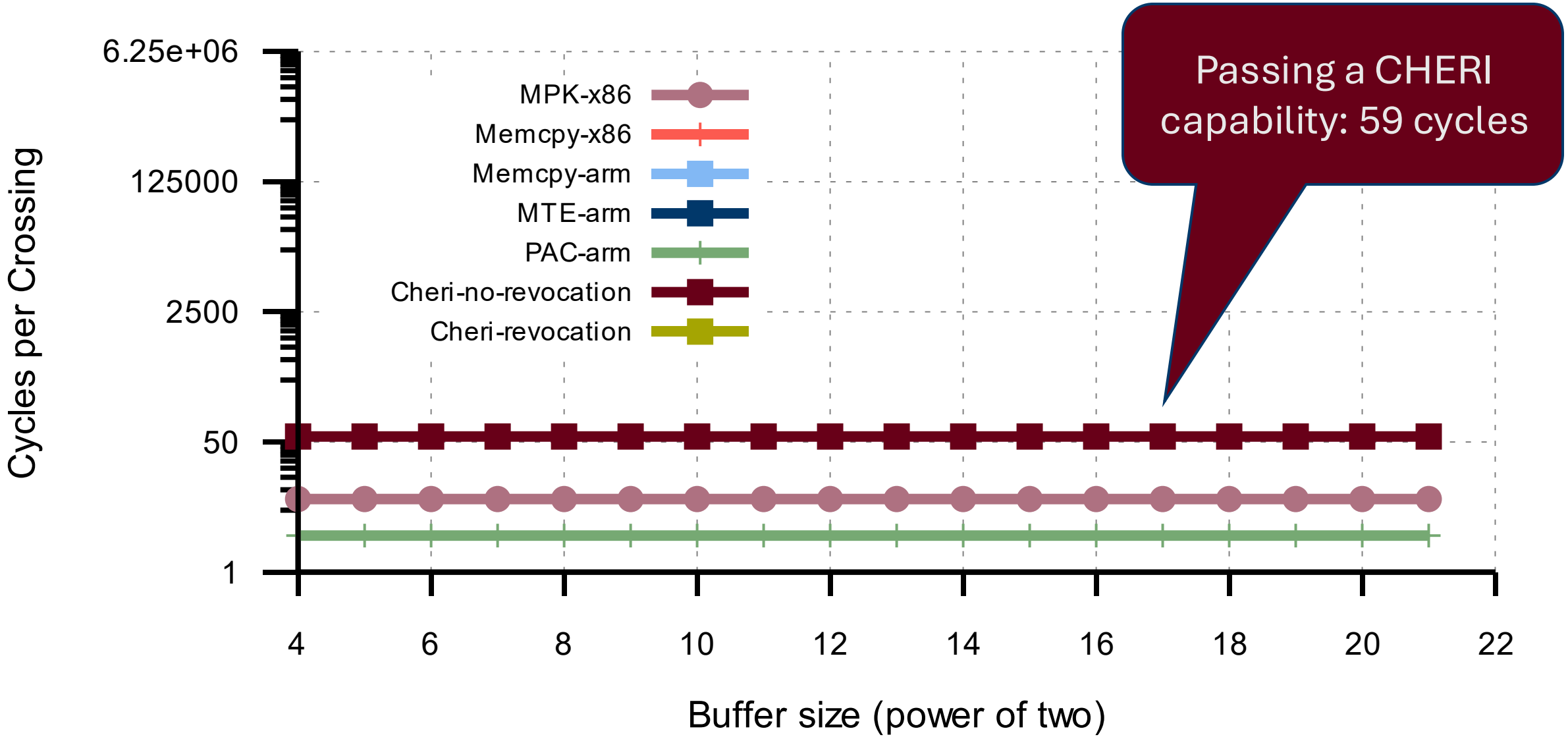
Passing data



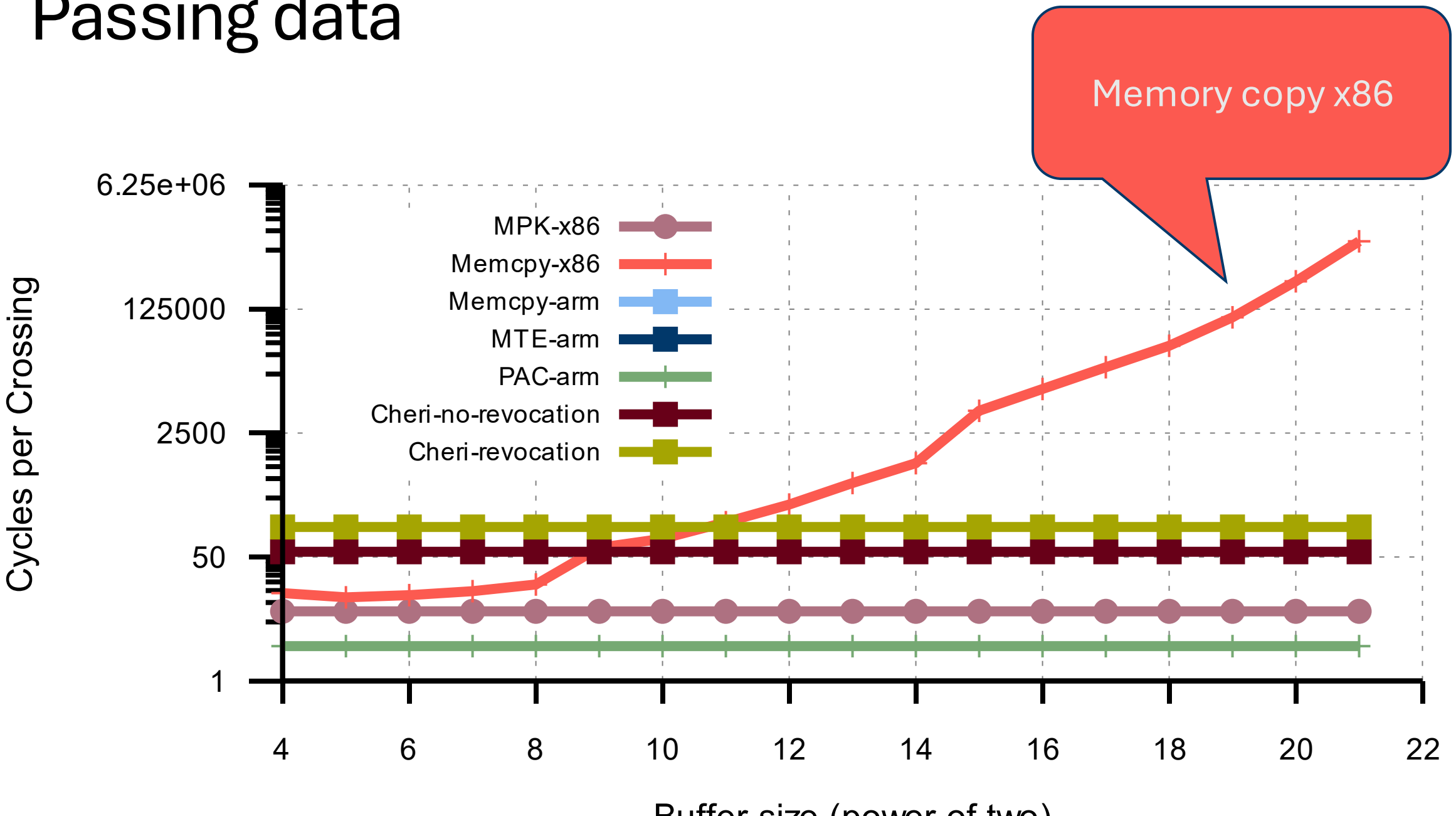
Passing data



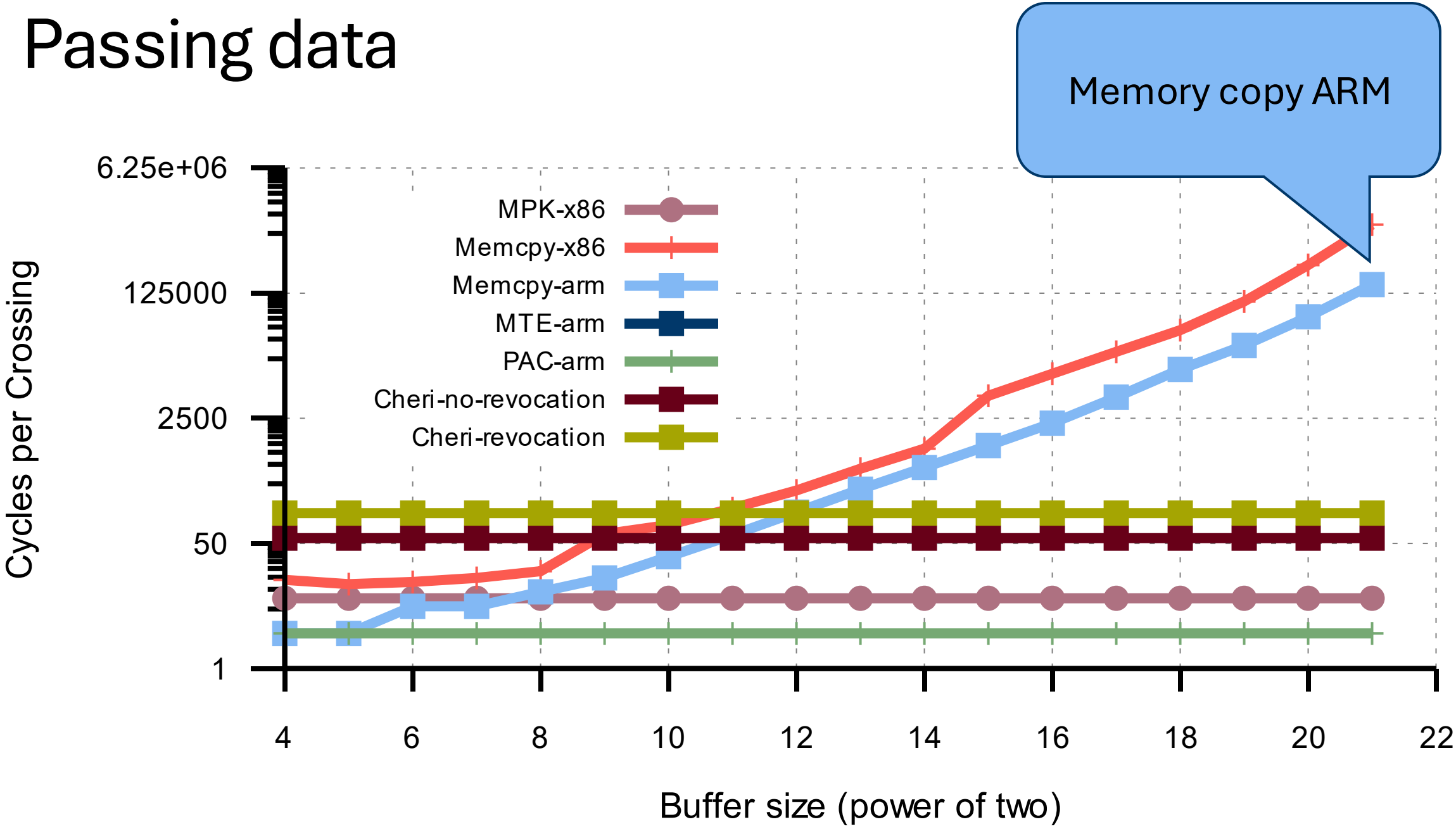
Passing data



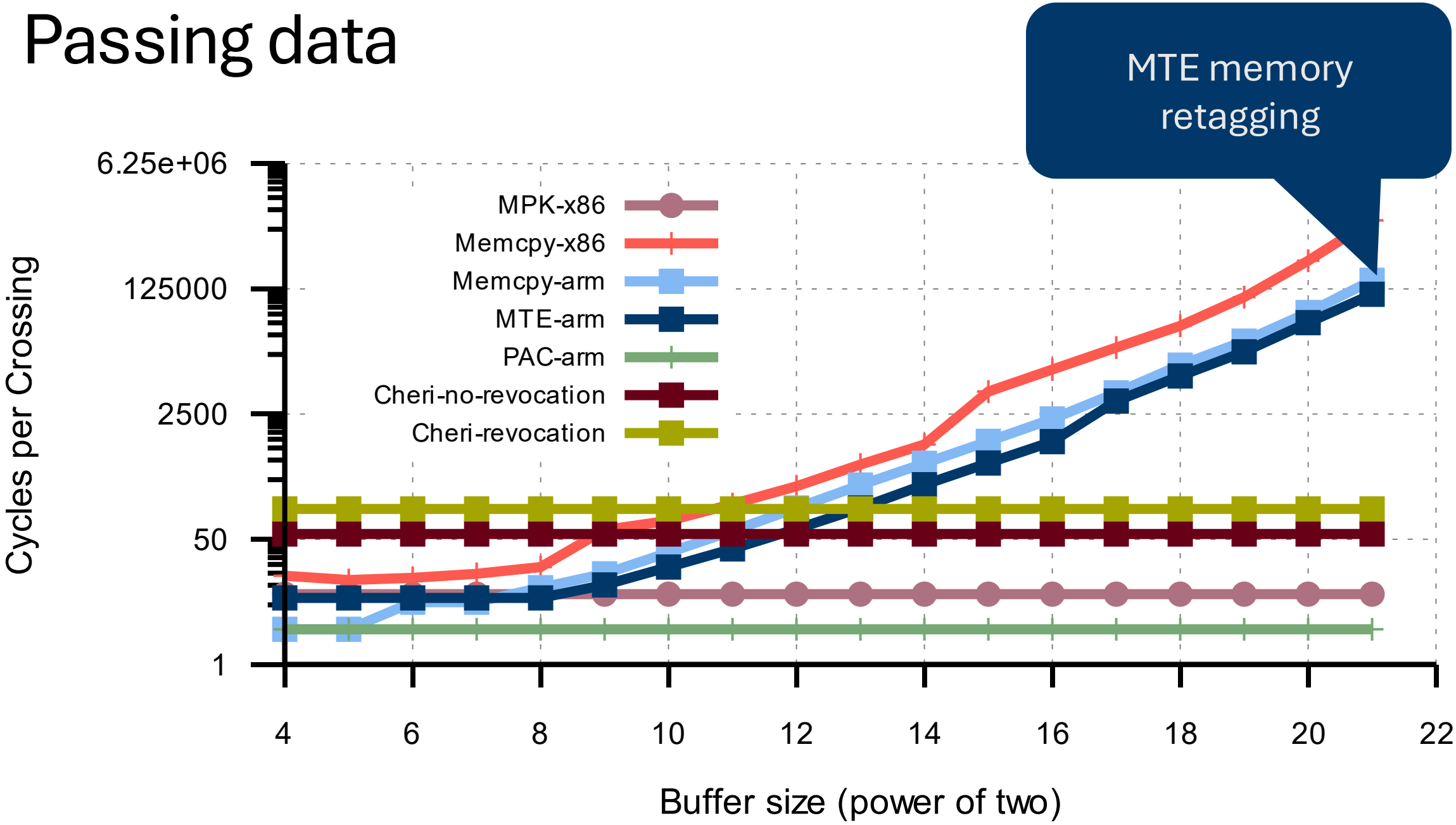
Passing data



Passing data



Passing data



Observations

- CHERI, MTE, PAC and MPK support zero copy
- Limitations
  - MPK: restricted programming model limited to a single core
    - Synchronization across cores requires TLB-shootdown-like scheme
  - MTE is surprisingly slow (as slow as memory copy)

Design principle

Core-coherent synchronization of rights *Hardware should support synchronization of access permissions across all cores of the system*

Critical for implementing general programming model

Possible solution

- MPK
  - Unprivileged instructions to update tags in the page table
  - Core-coherent TLBs (tried in the past, probably can work again)

Observation

- Tagged architectures (MPK and MTE) are limited with the number of isolated subsystems: 15 for MPK and MTE

Design principle

Practically large number of isolated subsystems *Hardware should implement a practically large number of isolation domains*

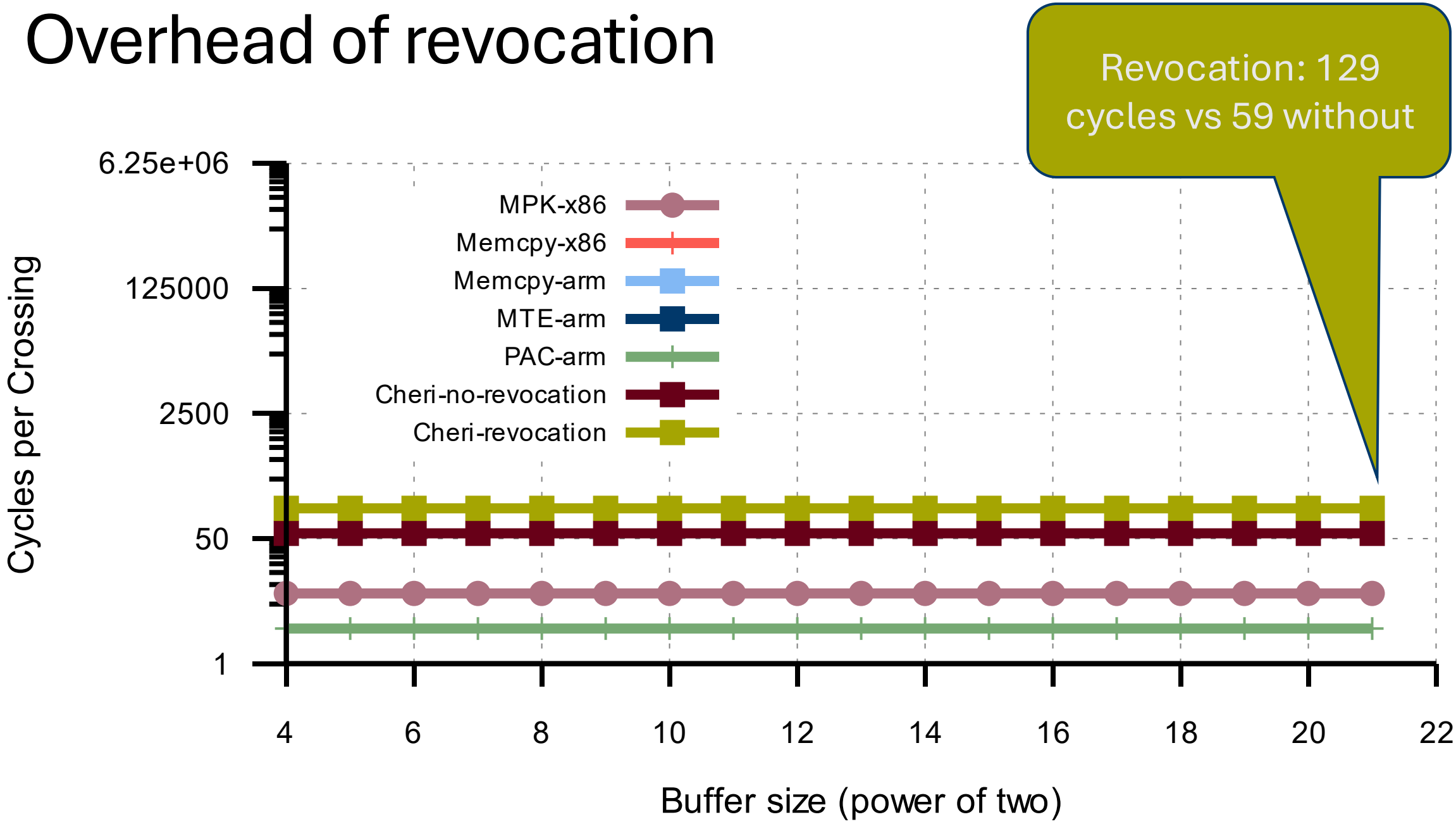
Possible solutions

- MPK
  - Requires update to the page table structure – challenging
- MTE
  - Can use 8 available bits of the pointer
  - 256 isolated subsystems

Performance

Revocation

Overhead of revocation



Observation

- MTE is the only scheme that supports revocation in a core-coherent manner

- MPK, PAC and CHERI are limited to a single core
  - Huge limitation!
- CHERI requires expensive instrumentation
- PAC access rights can remain resident in registers
  - So maybe PAC can’t actually implement clean revocation

Design principle

Revocation *Hardware must support revocation as a first-class citizen*

Possible approaches

- MPK
  - Core-coherent TLBs
- CHERI
  - Proxy capabilities