UNIVERSITY OF CAMBRIDGE

Enhancing temporal safety of CHERI-enabled language runtimes with ARM Memory Tagging Extension

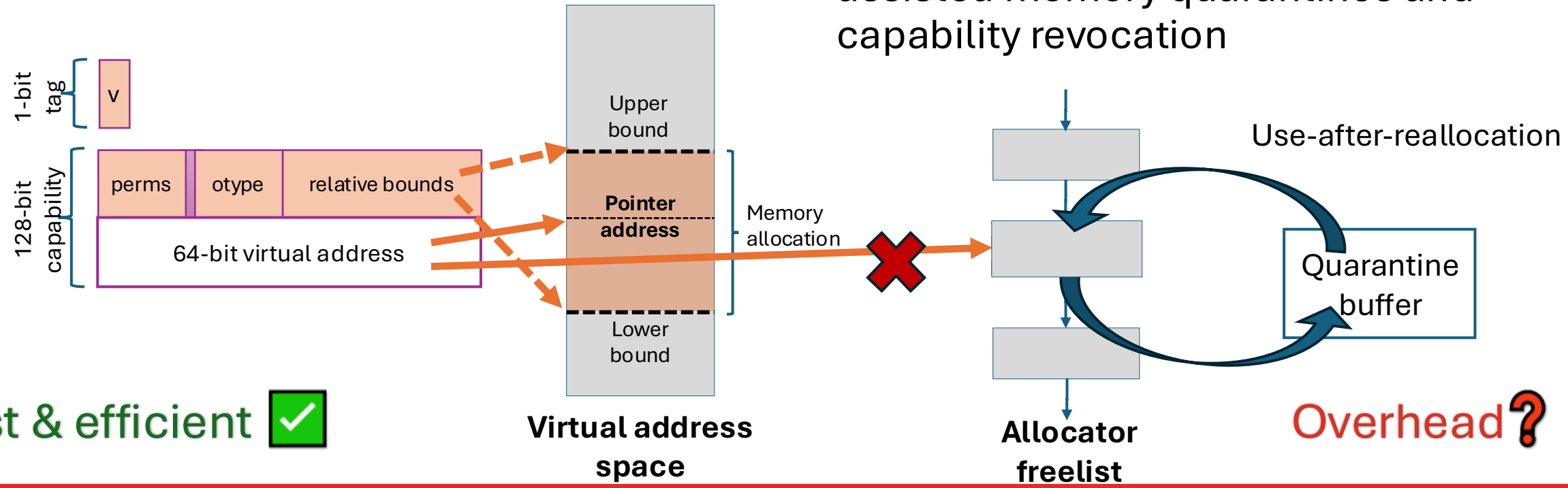
Qianhui Wang

Supervisor: Robert N.M. Watson

10 Mar 2026

Background: CHERI C/C++ memory safety

- CHERI mitigates memory safety vulnerabilities in C/C++ through **capabilities**
- Architecturally, capabilities distinguish pointers from memory addresses
- Spatial safety through hardware enforcement
- Temporal safety through software-assisted memory quarantines and capability revocation



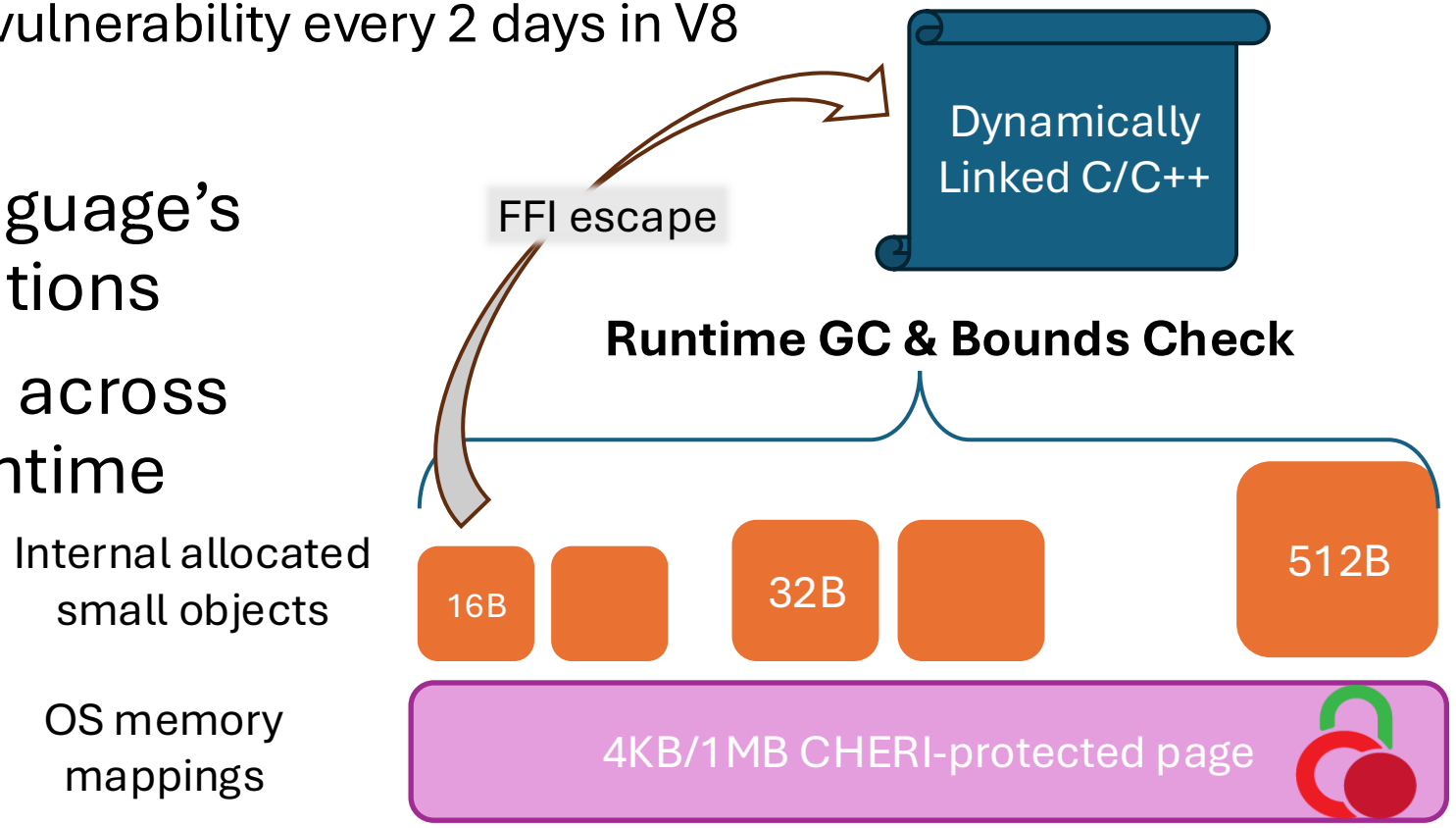
fast & efficient ✅

Overhead?

CHERI C/C++ is also useful for managed languages

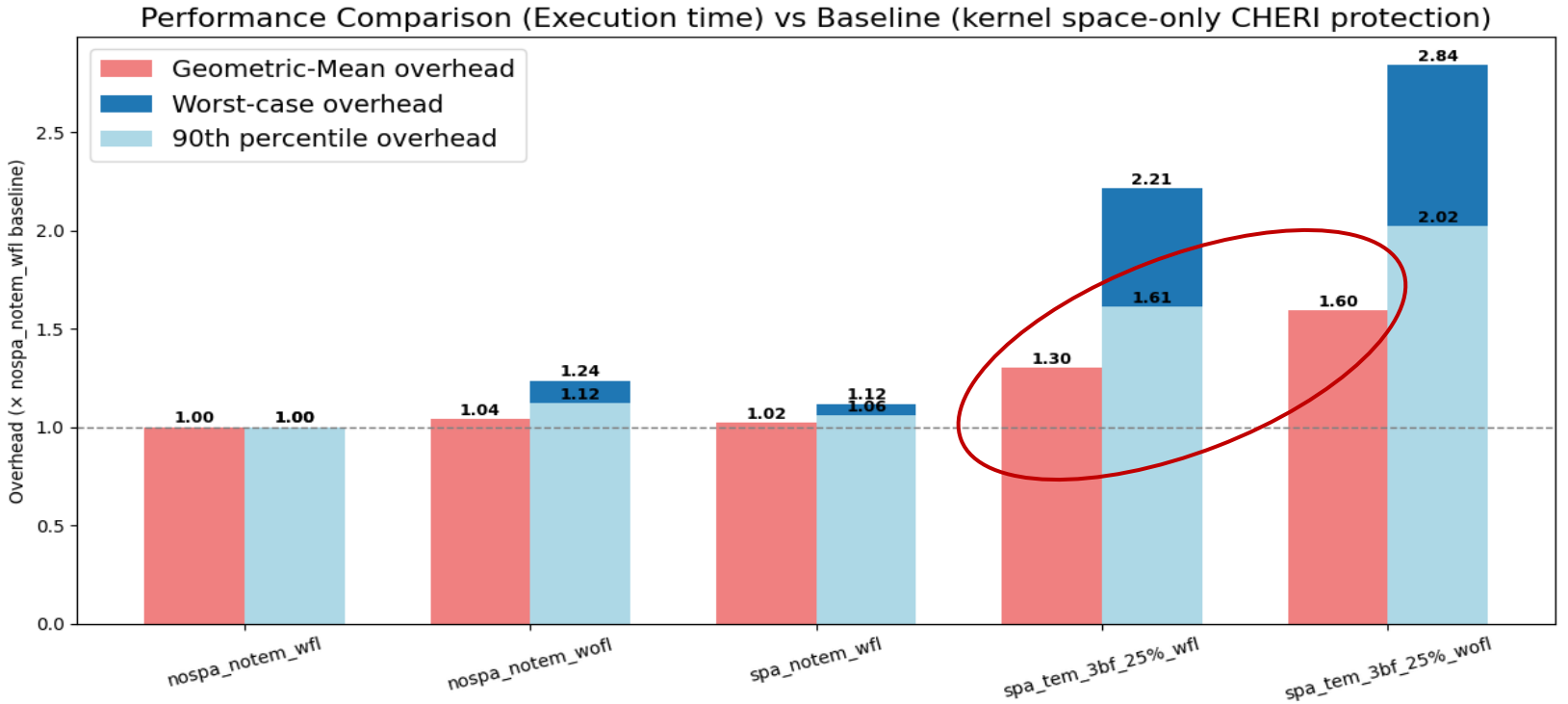
- Managed languages rely on ~M lines of memory-unsafe C/C++ that are prone to bugs, e.g., 0.7M in CPython, 1.2M in JDK, 2M in V8
  - 1 critical memory-safety vulnerability every 2 days in V8

- Need to protect the language's smaller memory allocations
- and memory that flows across language borders at runtime



Limitation: CHERI temporal safety incurs high overhead for language runtimes [a study on CHERI-CPython]

- High churn rate of small object allocations and frees in runtimes
- **90%** increase in memory traffic due to the need for quarantine
- **30%** execution time overhead + another **30%** due to unfriendly interaction with the language's type-based freelist mechanism



Key observations: prototyping [**temporary**] vs essential overheads[**research-worthy**]

Prototype software:

- Preliminary CHERI support in language allocators
- Yet to explore integration of language-level GC with capability revocation mechanism

Prototype hardware:

- ARM Morello board has known microarchitectural limitations

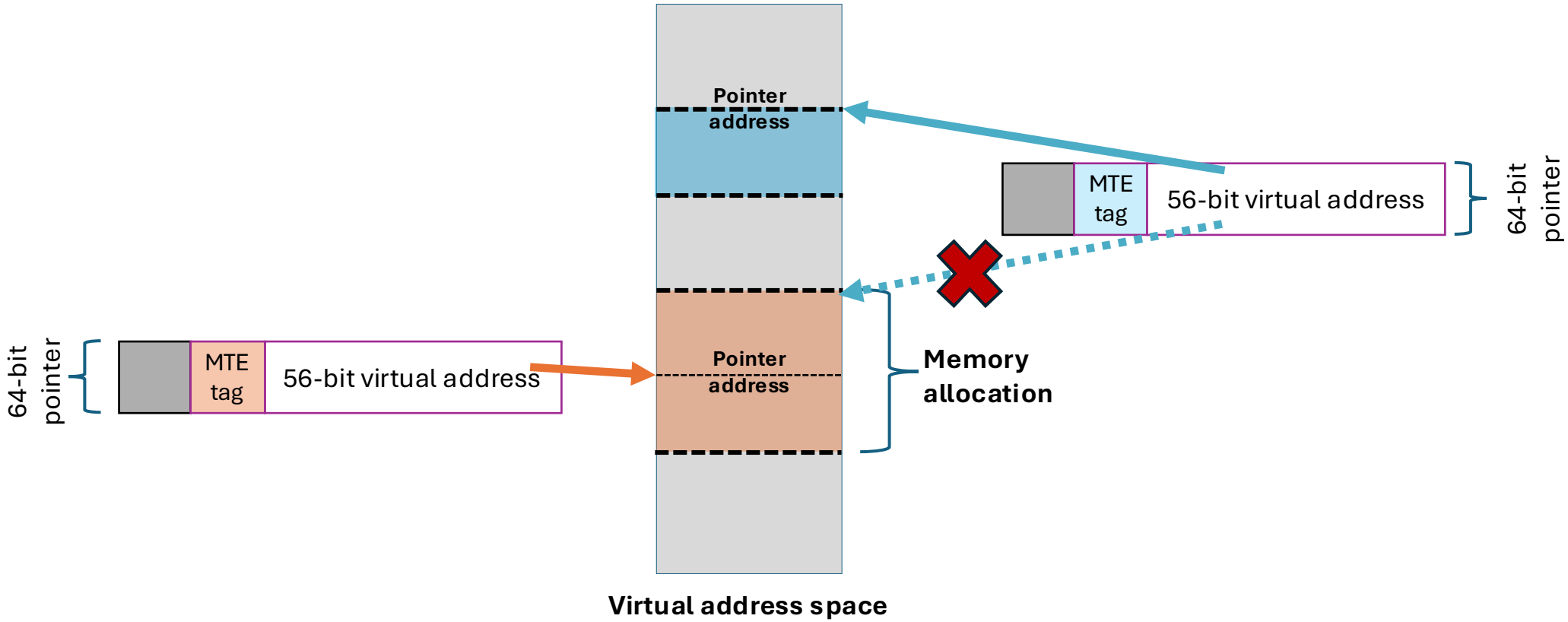
?

There are intrinsic limitations to the CHERI temporal safety mechanism on language runtime allocators:

- Could we reduce the amount of memory quarantined and thus the total amount of memory traffic?
- Could we reduce the number of revocation sweeps required and thus improve the execution time?

ARM Memory Tagging Extension (MTE)

- MTE provides hardware protection against memory safety errors at run-time; software decides faulting modes for speed/security



How could CHERI work with MTE in synergy?

- What are our goals?
  - retain CHERI deterministic security guarantees vs MTE probabilistic nature
  - achieve MTE performance [best ~2-17% overhead*] & close the gap from UAR to UAF
- Our initial use-case study:
  - explore using the MTE colours for immediate reallocation

  ⇒ this could lead to an ideal 15x reduction in #revocation sweeps
  - Architecturally, add MTE pointer tags into capability metadata and retain MTE physical memory tags

*Noh, Taehyun, et al. "ARM MTE Performance in Practice (Extended Version)." *arXiv preprint arXiv:2601.11786* (2026).

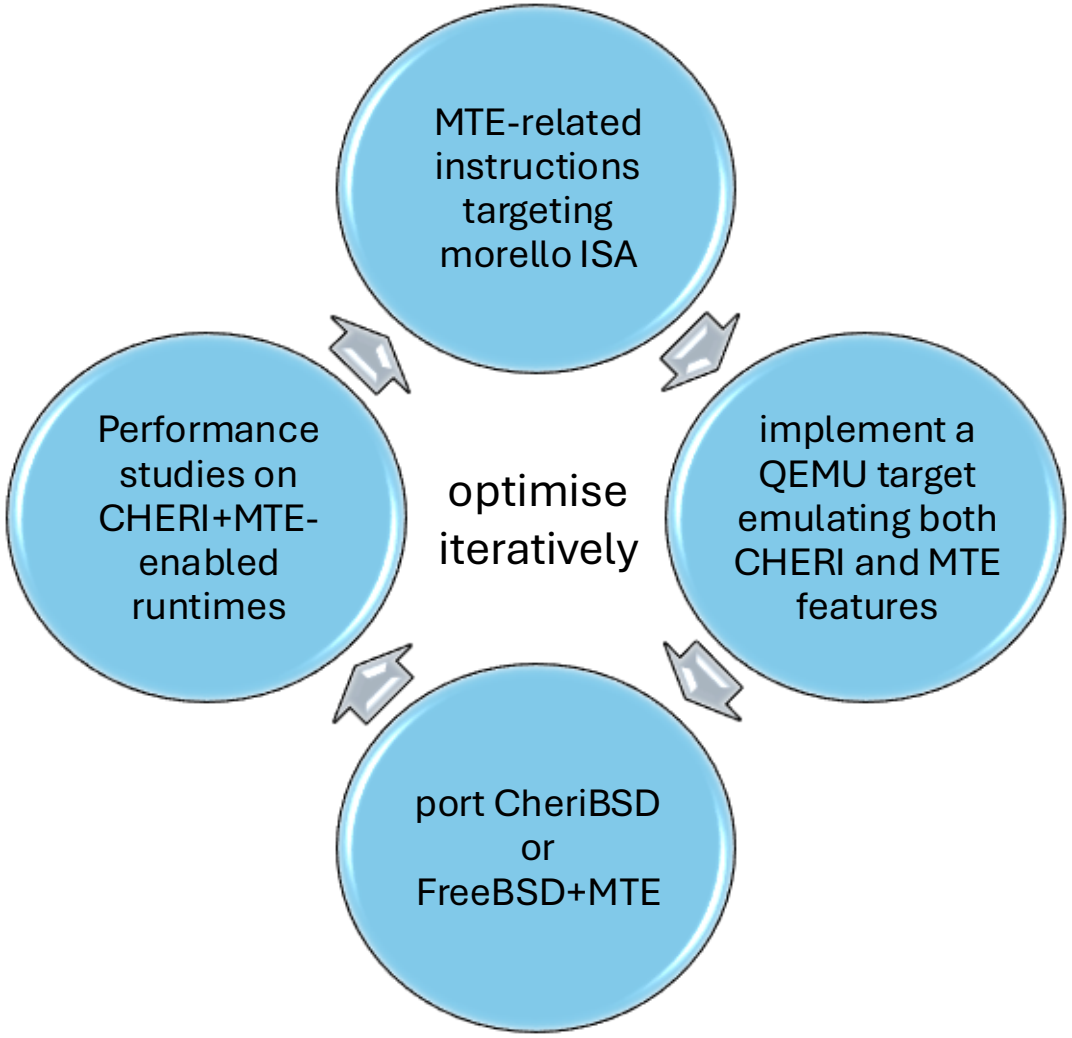
Progress update: CHERI/MTE threat model study

- Difficulties:
  - support for MTE in mainstream OSes, compilers, hypervisors is mostly under development
  - devices and emulators implement different versions of FEAT_MTE

	CHERI (morello)	Apple EMTE (m5 chip)	QEMU-MTE3 Linux	QEMU-MTE3 FreeBSD[buggy]
UAF	UAR	✓ 🎲	✓ 🎲	✓ 🎲
OOB	✓	Heap: ✓ 🎲 Stack: pthread no support Global: ?	Heap: ✓ 🎲 Stack: Global: ×	Heap: ✓ 🎲 Stack: × Global: ×
read-only PTR	✓	×		
IPC	✓	×	Work in Progress	
monotonicity	✓	×		
pointer type confusion	✓	×		

Looking ahead: design an architectural model that combines CHERI & MTE and build a software prototype for evaluation

- An architectural model such as QEMU is a fastest way to validate our hypothesis on whether CHERI+MTE could reduce memory traffic on language runtimes
- Later accurate microarchitectural performance studies could be done by feeding QEMU-generated traces into Gem5 cache model



UNIVERSITY OF CAMBRIDGE

Qianhui Wang

qw304@cam.ac.uk

cascade