Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alistair Kressel, Hugo Lefeuvre and Pierre Olivier
The University of Manchester

This work was partly funded by the EPSRC/Innovate UK grant EP/X015610/1(FlexCap), the UK's EPSRC grants EP/V012134/1 (UniFaaS), EP/V000225/1 (SCorCH), a studentship from NEC Labs Europe and a Microsoft Research PhD Fellowship.

Background & Motivation

CHERI & Morello

- Capability Hardware Enhanced RISC Instructions (CHERI) brings hardware capabilities to RISC ISAs[1]
- ARM implementation called Morello, with hardware available
- Hardware capabilities constrain memory accesses
  - Enforce bounds and permissions checks
  - Encode bounds and permissions information alongside addresses
  - Capabilities can be used only where needed to lower the porting costs (hybrid mode)
- Code and data accesses can be tightly controlled for **effective compartmentalization**

[1] R. N. M. Watson et al., "CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization,", 2015

Compartmentalization

- Isolates portions of code and data
  - Reduces the privileges of isolated components to reduce damage which can be caused
  - Systems software is often not memory safe - so it is important
- Ideal requirements:
  - Low **porting and refactoring cost**
  - Low **performance overhead vs. uncompartmentalized execution**
  - Strong **security guarantees** (eg. granularity of sharing)
  - Good **scalability** to many compartments
- Not all methods will suit all needs

Related Work

- CheriRTOS[1], CompartOS[2], CheriOS[3], CheriBSD[4], Cap-VMs[5] and CHERI JNI[6] and CherIoT[7] utilize CHERI for compartmentalization

Limitations:

- Lack of exploration of the CHERI hybrid mode compartmentalization design space
  - Which models are available?
  - How can data be shared between compartments?
- None of these works evaluates CHERI compartmentalization on available hardware
  - Hard to draw meaningful performance conclusions from FPGAs and simulations

[1] H. Xia *et al.* 2018, [2] H. Almatary *et al.* 2022, [3] L. G. Esswood, 2021, [4] R. N. M. Watson *et al.* 2015, [5] V. A. Sartakov *et al.* 2022, [6] D. Chisnall *et al.* 2017, [7] S. Amar *et al.* 2023

Research Questions

- How can CHERI hardware capabilities in hybrid mode be leveraged to facilitate compartmentalization?
  - Which models are possible?
  - What are the refactoring costs given the compatibility promised by hybrid mode?
  - How does the performance compare to other intra-address space mechanisms such as MPK?
  - How well do compartment models scale to many compartments?
  - What are the security properties of the compartment models?

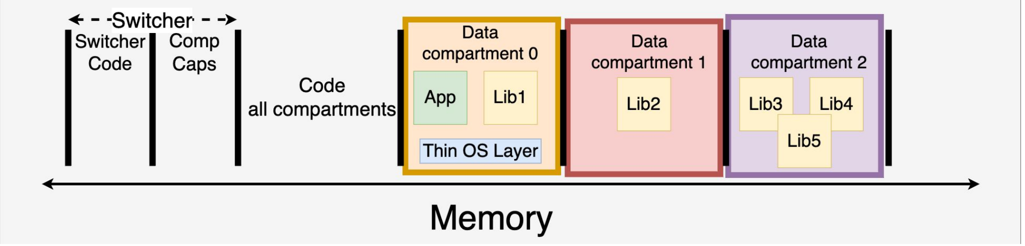
Evaluation performed on real hardware to gain better insight into performance implications

The University of Manchester

MANCHESTER 1824

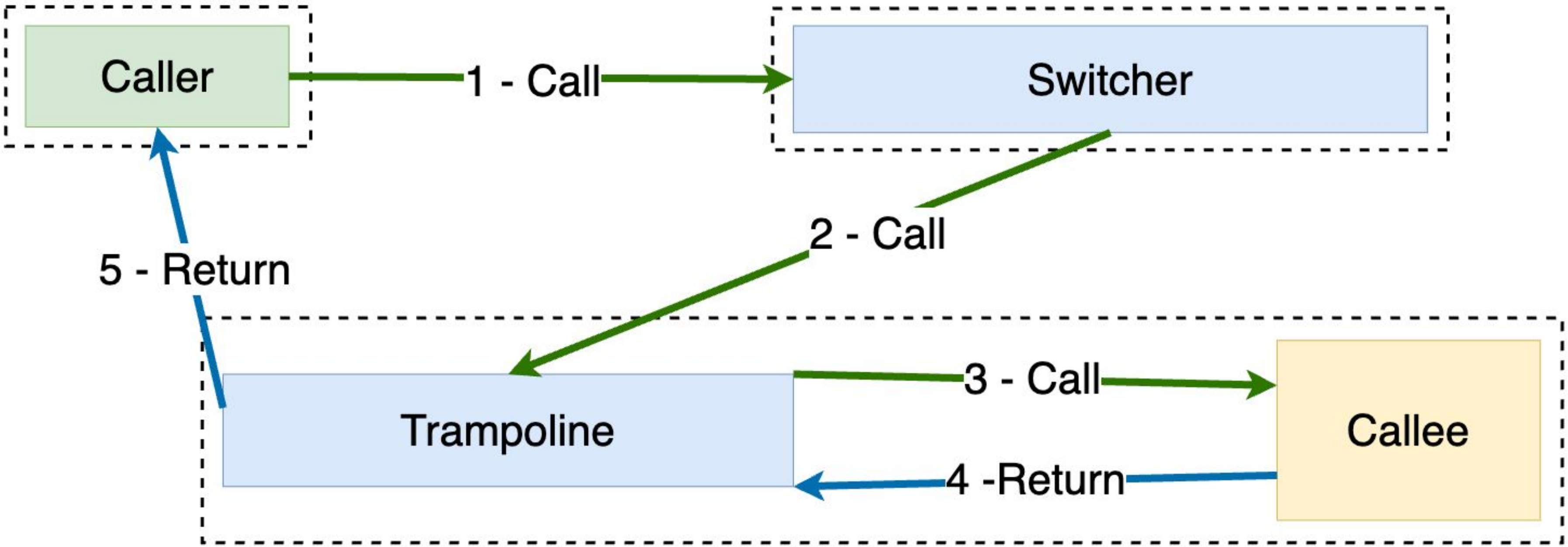
Design

Overview



- Compartments defined statically at build time by developer
- Initialised during the boot process
- Compartments are enabled by two global architectural capabilities
  - Default Data Capability (DDC) restricts data access to compartment memory
  - Program Counter Capability (PCC) restricts code access
- Compartments have private stacks, heaps and allocators
- Each compartment occupies its own portion of the address space
- A **compartment switcher** resides in memory not accessible to any compartment
- This design ensures secure management of hybrid mode CHERI compartments

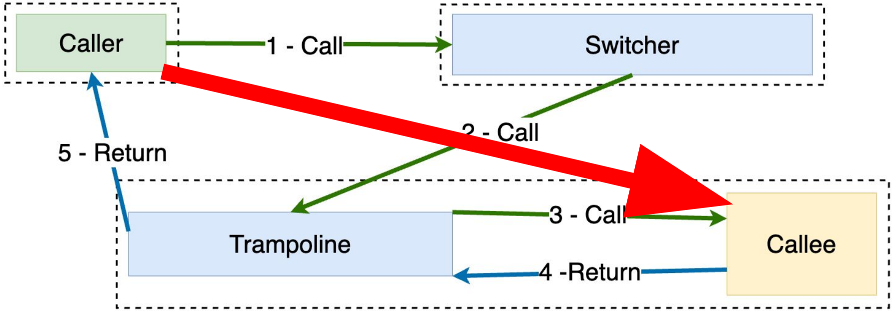
Switching Mechanism



Challenges

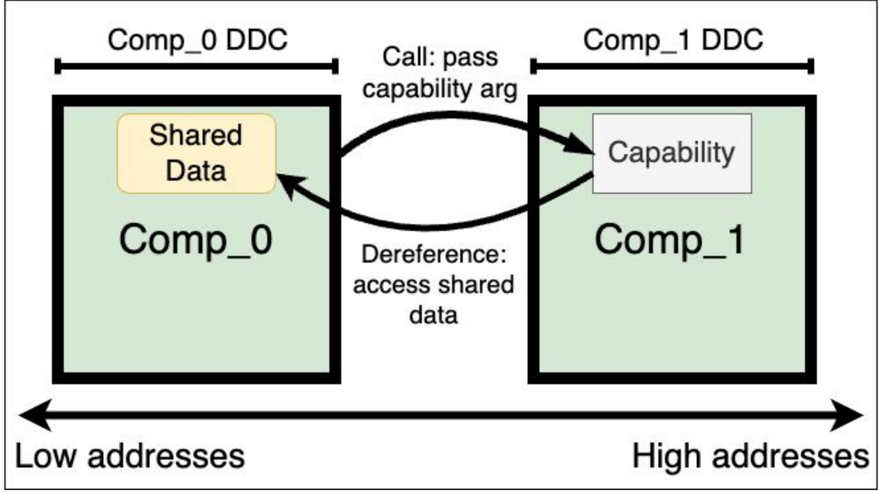
The University of Manchester

Data Sharing Overview



- Isolated compartments must communicate
- What compartment models can be used without weakening the compartment security, whilst not imposing a high overhead on performance and porting effort
- Two methods are explored and evaluated to achieve data sharing

Method 1: Manual Capability Propagation

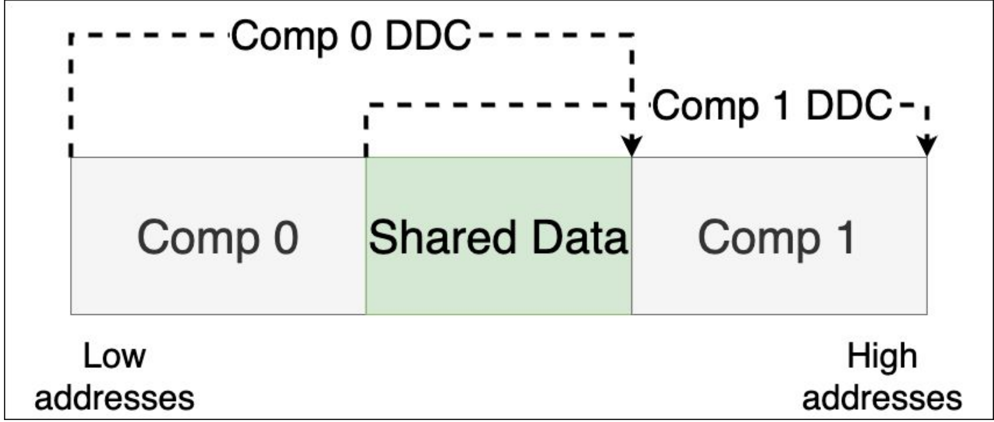


- Pointers manually annotated in source code to be transformed to capabilities
- Can be burdensome due to capability propagation
- Compromise: Individual functions are sandboxed
  - Wrapper function transforms pointer arguments to capability arguments
  - Requires minimal rewriting because scope is individual functions
- Trust model: **sandbox**

MANCHESTER 1824

The University of Manchester

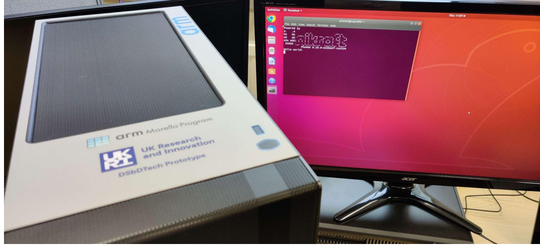
Method 2: Overlapping Shared Memory



- Shared memory region is initialised
- Shared memory lies between a pair of compartments
- Compartment DDC bounds **overlap** the shared memory
- Data is annotated by developer to place it in shared memory
- Trust model: **mutual distrust**

Evaluation

MANCHESTER 1824
The University of Manchester



Evaluation Overview

- FlexOS[1] (compartmentalization-aware LibOS) ported to Morello
- Libsodium crypto library test suite derived benchmark used to evaluate ***manual capability propagation***
  - 5 functions manually annotated
  - Different configurations run
- SQLite benchmark used to evaluate ***overlapping shared memory*** approach
  - Filesystem isolated
  - Performs 5000 INSERT operations - system call and filesystem heavy

All experiments were run bare-metal on Morello hardware

[1] Hugo Lefeuvre et al. 2022

Porting Effort

Software	Sharing approach	Compartments	Porting cost	Changes (LoC)
libsodium	Function sandboxing	`sodium_hex2bin`	< 1h	9
		`sodium_bin2hex`	< 1h	8
		`chacha20_encrypt_bytes`	< 2h	73
		`store32_le`	< 1h	5
		`store64_be`	< 1h	5
SQLite	Overlapping DDCs	`vfscore + ramfs`	< 2d	< 300

Manual capability propagation:

- Proportionally higher than overlapping shared memory; all shared data pointers must be annotated - **max 73/141 LoC** (>50%)
- Porting effort as implemented is low due to small scope of functions

Overlapping shared memory:

- Low; shared data only annotated at declaration - **<300/5.8k LoC** (~5%)

Insight:

- Important to choose code to isolate carefully

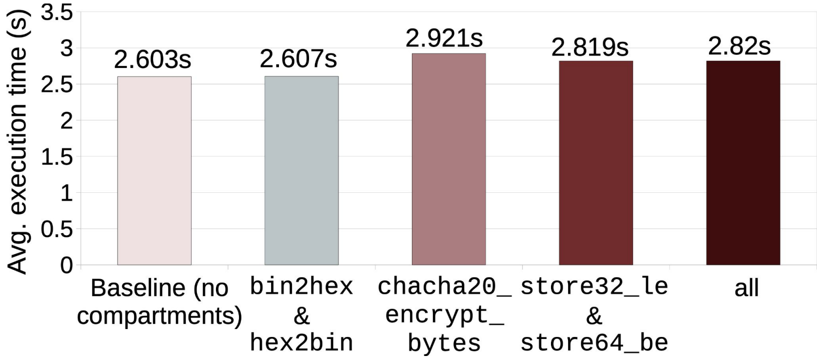
Performance

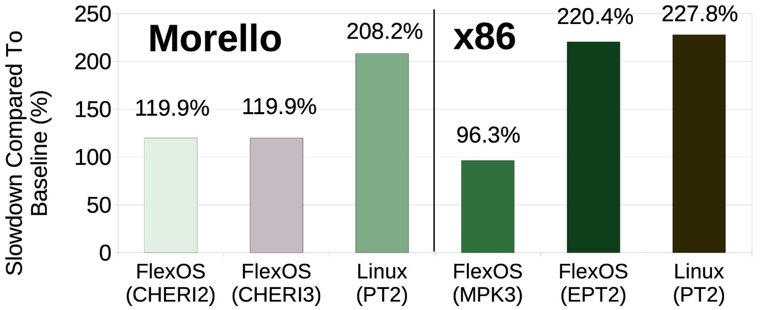
Manual Capability Propagation:

- With carefully selected functions overhead is low (Libsodium)
  - Overhead evaluated is **0.1%-12.2%**

Overlapping shared memory:

- Performance overhead same order of magnitude to MPK and lower than EPT on FlexOS (SQLite)
- Runs faster than same benchmark on Linux (SQLite)
  - Same hardware
  - Isolation is between user and kernel





MANCHESTER 1824

The University of Manchester

Summary

Summary of Contributions

1. Exploration of hybrid mode CHERI compartmentalization design space
2. Performance evaluation of approaches
3. Evaluation of security properties compared to Intel MPK and EPT

Project Website: