MANCHESTER 1824

The University of Manchester

FlexCap

Improving Unikernels by harnessing Morello's Compartmentalisation

John Alistair Kressel and Pierre Olivier
john.kressel@manchester.ac.uk

FlexCap is funded by the UK DSbD programme via UKRI

Unikernels in a Nutshell

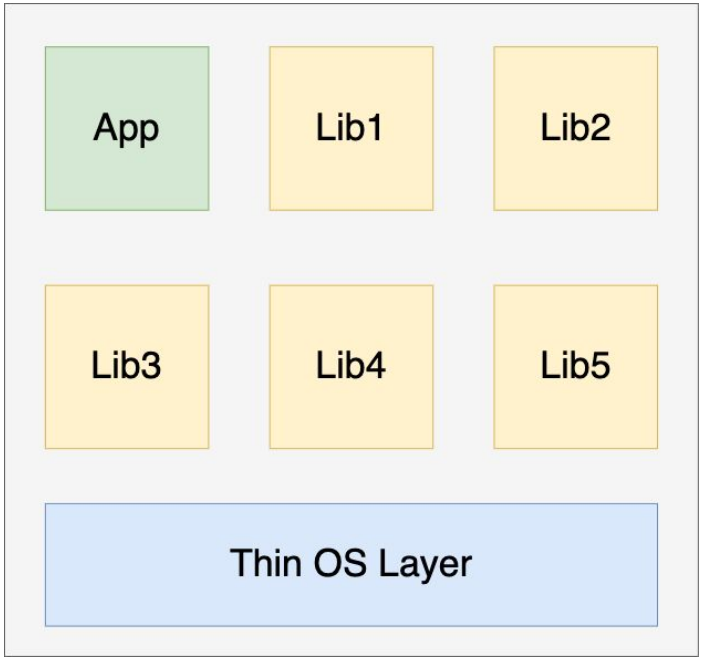
Main Market: Cloud applications running in Virtual Machines in Datacenters
Approach: OS becomes a library
Selling Point: Be lean and mean
“Deploy only your app with the subset of the OS that is needed”

What is Unikraft?

- Popular unikernel (aka Library OS) and POSIX compliant
  - Linux Applications run without modifications

Unikernel:

- Application compiled with libraries and ultra lightweight library OS layer
- Low resource consumption and high performance
- Reduced attack surface due to smaller code base
- Can be run bare metal or in a VM

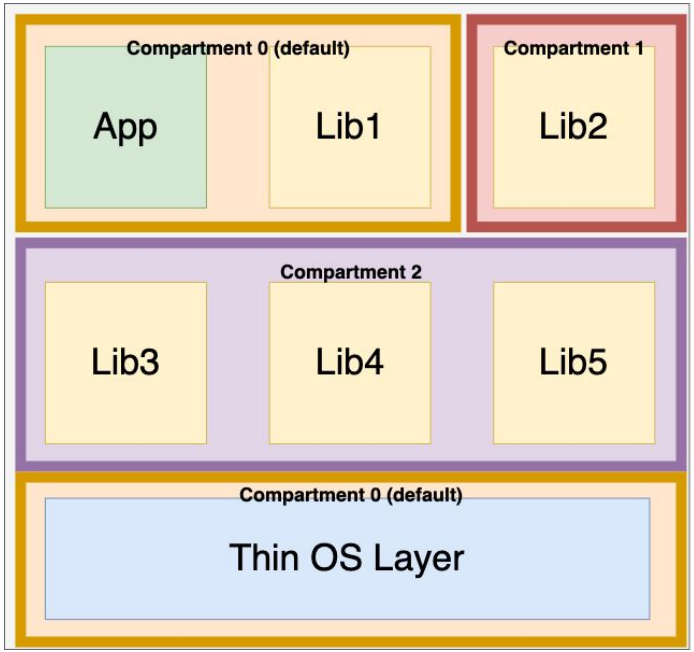


Unikraft paper: https://doi.org/10.48550/arXiv.2104.12721

What Is FlexOS?

- Introduces flexible compartmentalisation achieved with Intel MPK and Intel EPT based isolation
- Developers define compartment boundaries and insert generic FlexOS annotations
- At build time FlexOS annotations are replaced with Intel MPK/Intel EPT specific code
- Isolation is achieved between library components

Threat model: Each compartment distrusts every other compartment



FlexOS paper: https://doi.org/10.48550/arXiv.2112.06566

FlexCap - Project Objectives

Overarching Research Question: Investigate Best Practice for using Compartmentalisation in Unikernels

Research Vehicles: Unikraft and FlexOS (open source unikernels)

Phases:

- Porting Unikraft and FlexOS (based upon Unikraft) to Morello
- Enable hybrid and purecap execution
- Evaluate the efficiency of capability enabled unikernels
- Research hybrid and purecap intra address space compartmentalisation
- Evaluate the performance and trade-offs of purecap vs. hybrid compartmentalisation

https://github.com/unikraft/unikraft

https://github.com/project-flexos/unikraft

Why Explore FlexOS On Morello?

- CHERI enables lightweight memory safety and isolation
  - perfect for lightweight unikernels/FlexOS
- Isolation can be more fine-grained than most existing solutions
  - Minimum isolation unit: Morello bytes vs Intel MPK one page
- Unikernels/FlexOS primarily used on datacentres
  - e.g. web servers, databases - security paramount!
- Unikernels/FlexOS run in single address space
  - desirable for performance, not security
  - hardware enforced isolation can restore security

Improving the security of unikernels/FlexOS is good for security sensitive applications typically run with unikernels/FlexOS!

FlexCap - Progress on Hybrid

Why Hybrid?

- Promises good compatibility with existing applications
  - No need to rewrite complex software
  - Faster adoption of Morello
  - Less chance of breaking poorly documented software

Criteria for a successful hybrid implementation:

1. Does not require significant rewriting of application
2. Maintains good performance

Morello - Compartments using Hybrid mode

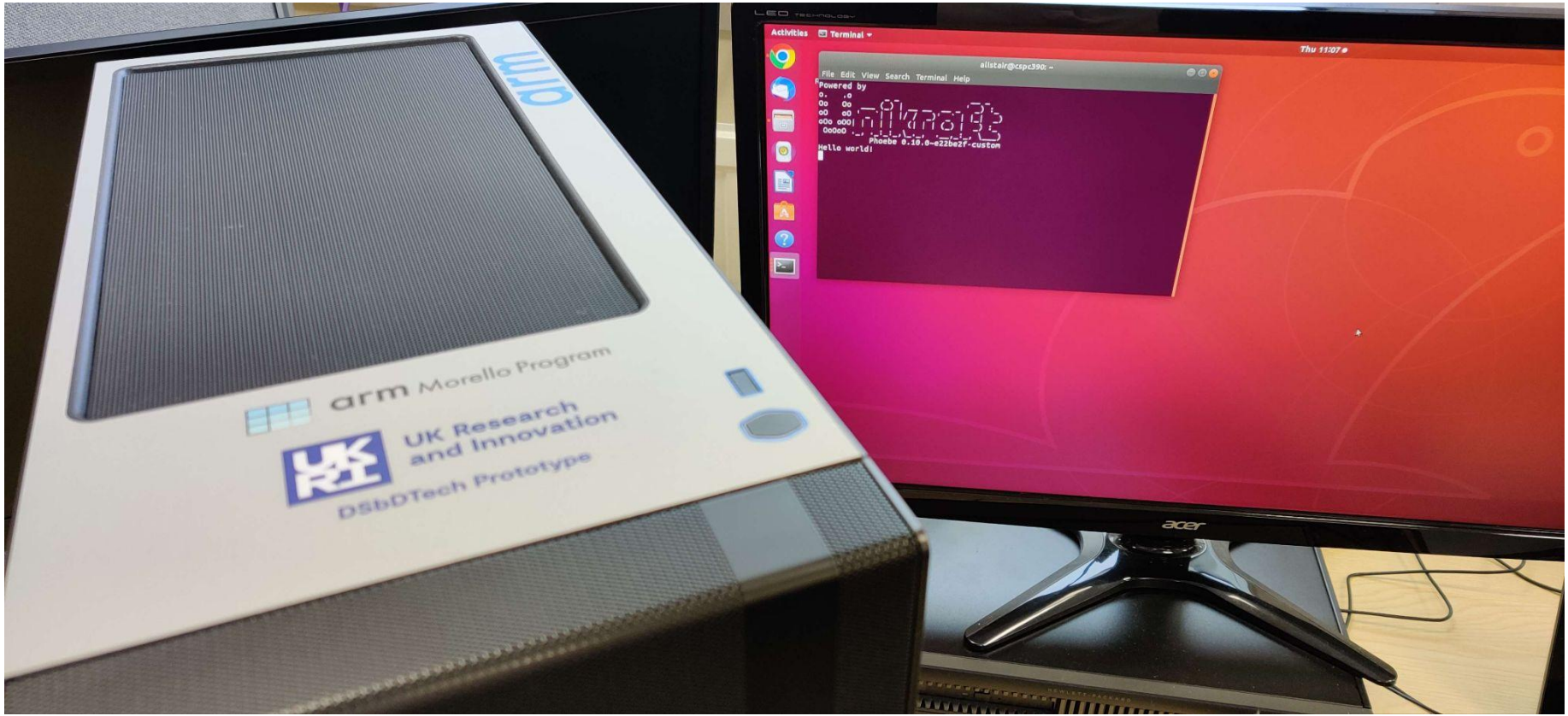
- Hybrid compartments are defined by PCC and DDC
- PCC (Program Counter Capability) restricts the program counter to specific code
- DDC (Default Data Capability) restricts memory operations to a specific **contiguous** region of memory

Together the DDC and PCC are used to restrict A64 instructions

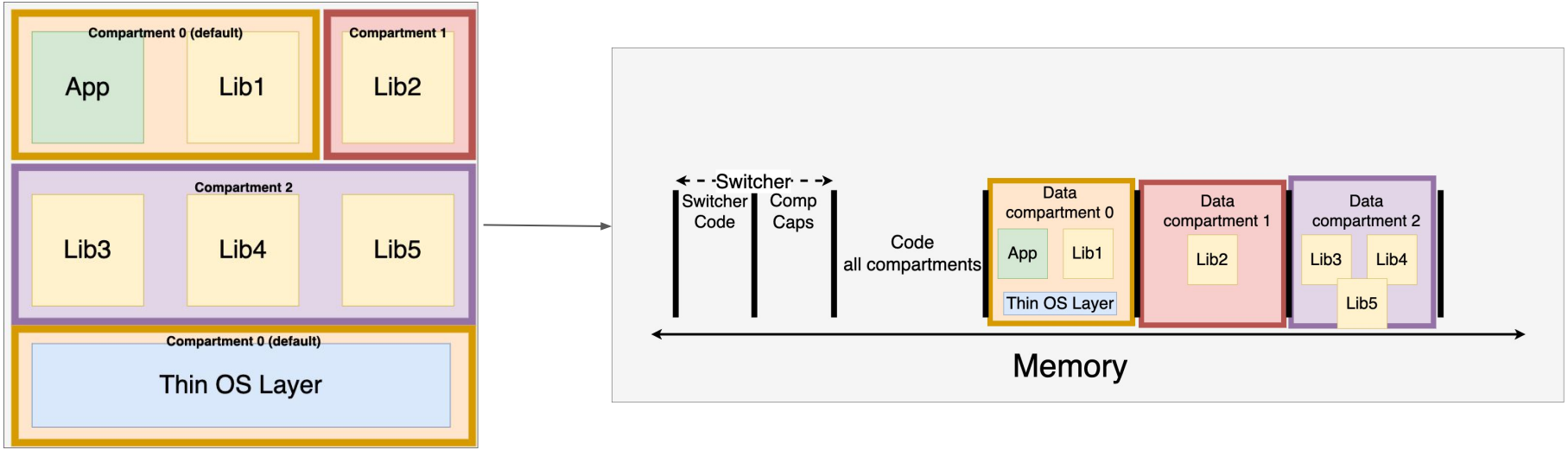
Progress so Far

- Project has been running for ~5 months
- Unikraft and FlexOS ported to Morello hardware
  - Bare metal
  - Boots in hybrid mode - capabilities enabled
  - Sets up capability features
  - Boots in <0.5s on Morello hardware
- Implemented hybrid compartments, enforced by PCC and DDC
- Implemented **switching mechanism**

Unikraft executing Hello World on Morello

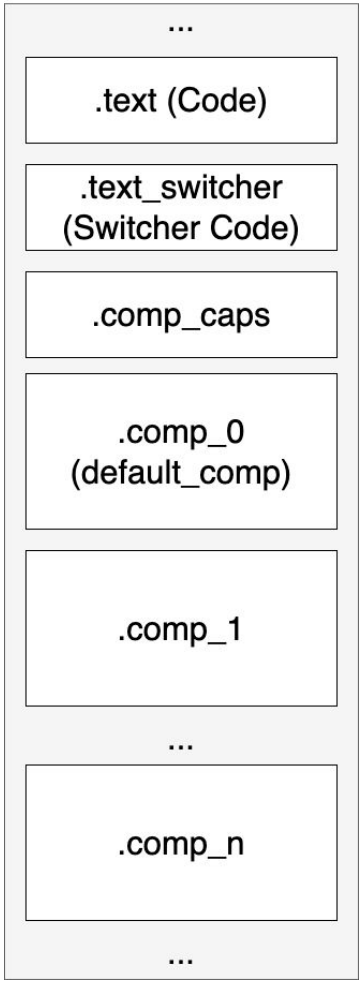


FlexOS Morello Compartment Overview



Memory Layout - Specified at Link Time

- Isolated libraries get own local linker script
- Script puts compartment data into a compartment data section
- Default compartment data goes into compartment 0 data sections
- Final linkage puts compartment data sections into separate sections of the ELF file



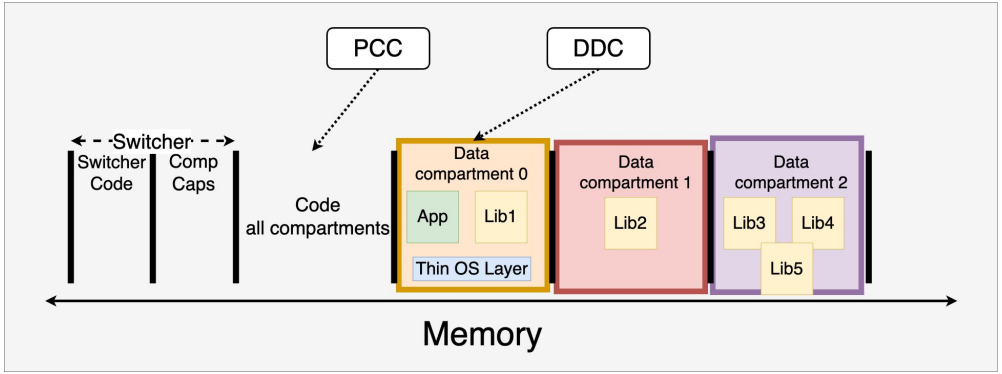
Compartment Initialisation

Remember

- Compartments defined by developers at compile time
- Compartments occupy different sections of the executable

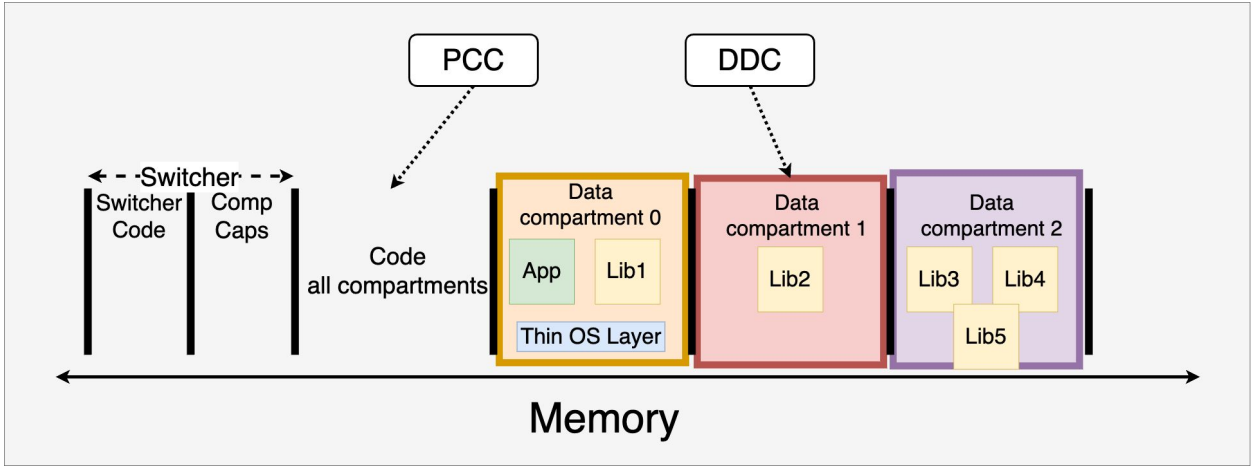
Initialisation process

- Capabilities to enforce compartment bounds initialised during boot
- Boundaries set in the linker script
- Enter default compartment at the end of boot process (compartment 0)

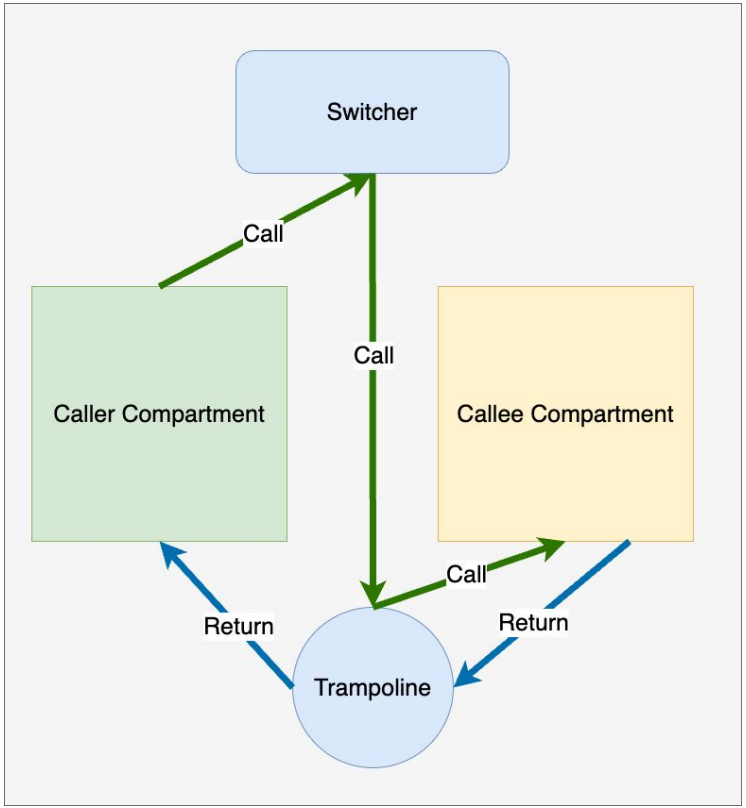


Compartment Isolation

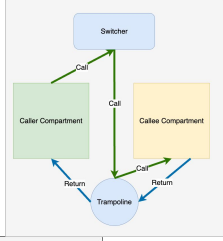
- Data for each compartment isolated via the DDC
- Each compartment data section is a separate memory location
- Code remains in one section - PCC remains unchanged between compartments
- Moving between compartments requires switching (DDC updates, etc.)



Compartment Switching Overview



Switching Compartment (caller)

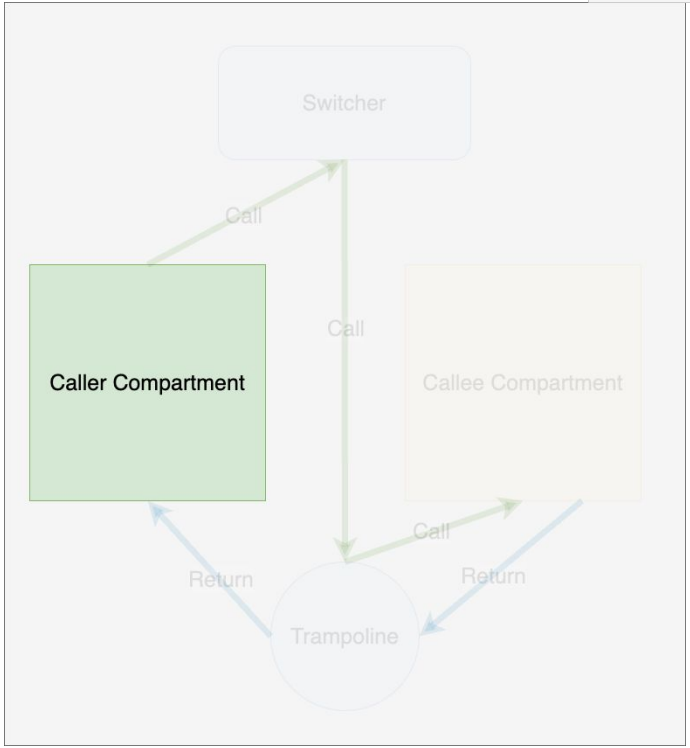


When calling:

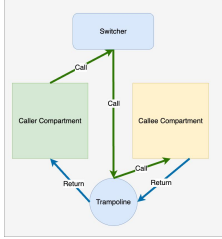
- Saves compartment context
- Caller compartment sets function arguments
- Loads target compartment ID and function pointer to call
- Invokes compartment switcher via sealed capabilities

On Return:

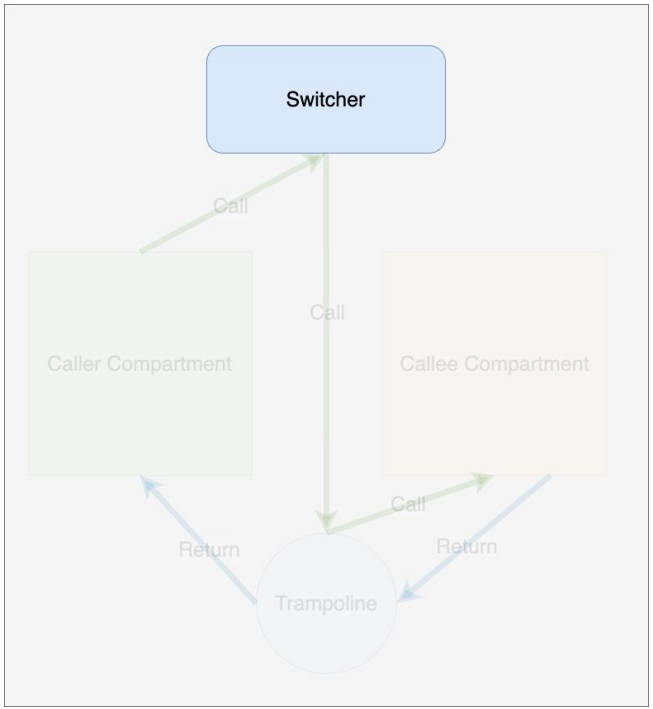
- Handles return values
- Restores compartment context



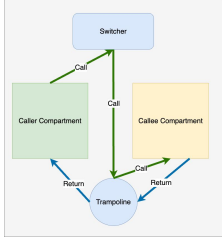
Compartment Switcher



- Is given: **callee compartment ID, function pointer and function arguments**
- Saves caller compartment DDC and PCC
  - needed for return
- Sets up callee compartment stack
- Switches DDC to callee DDC capability
- Branches to trampoline



Trampoline and Callee



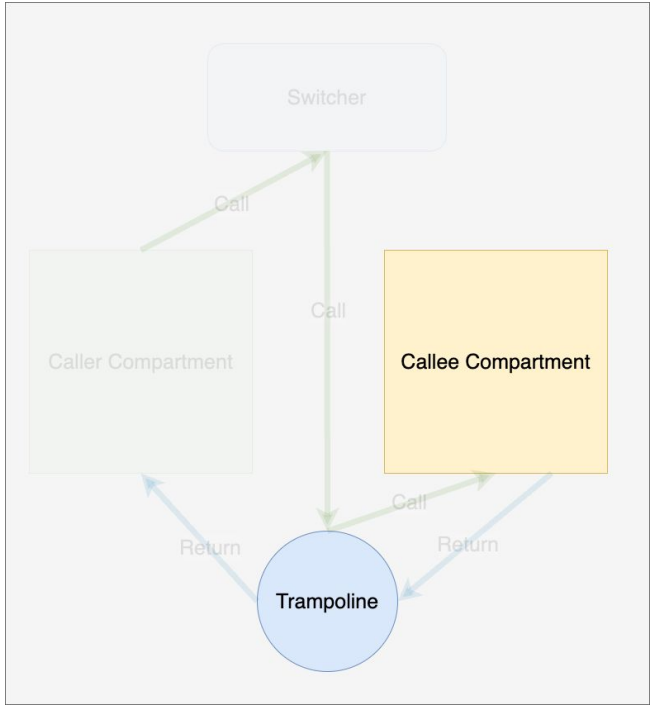
Need to create a capability to return to caller Compartment

When called:

- Saves capability needed to restore caller compartment
- Jumps to called function pointer

On return:

- Uses saved capability to load and restore caller capability pair
- Returns via capability to caller compartment



Remember - The Promise of Hybrid

- Highly **compatible** with existing software
- Don’t need to port to purecap
- Maintains high performance

Existing applications pass pointers as arguments into functions to avoid copying data

How do we pass pointers among compartments where we can only set compartment boundaries with annotations (ala FlexOS)?

Passing pointers

Challenge for Hybrid Mode

How to pass pointers when adding compartments to an existing application?

- Compartments can only access addresses within their own memory region
- Existing pointers to addresses which now belong to compartments now inaccessible

```c
int *comp_1_data = 42;

void compartment_1_func() {
    int result = compartment_2_func(comp_1_data);
}


int compartment_2_func(int *ptr) {
    return *ptr + *ptr;
}
```

Option 1 - Source code rewriting

- Analyse the flow of data crossing compartment boundaries
- Rewrite sections of the application to use capabilities in place of pointers
- Solves issue of pointers

But!

- Requires complex rewriting - negates hypothetical advantage of hybrid
- Not compatible with existing code

```
int *comp_1_data = 42;

void compartment_1_func() {
    int result = compartment_2_func(comp_1_data);
}


int compartment_2_func(int *ptr) {
    return *ptr + *ptr;
}
```

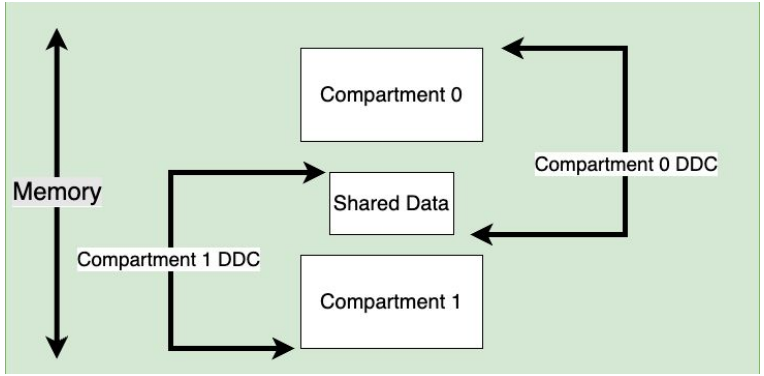
```
int *comp_1_data = 42;

void compartment_1_func() {
    int result = compartment_2_func(cheri_ptr(comp_1_data, sizeof(int)));
}


int compartment_2_func(int *__capability ptr) {
    return *ptr + *ptr;
}
```

Option 2 - Shared data section (2 compartments)

- Position shared data section between 2 compartments so that each can access
- Allows data sharing
- Existing FlexOS annotations can be used to achieve this

But!

- Access is granted to all of shared data
  - Potential for oversharing
- Only works for 2 compartments - very limited!

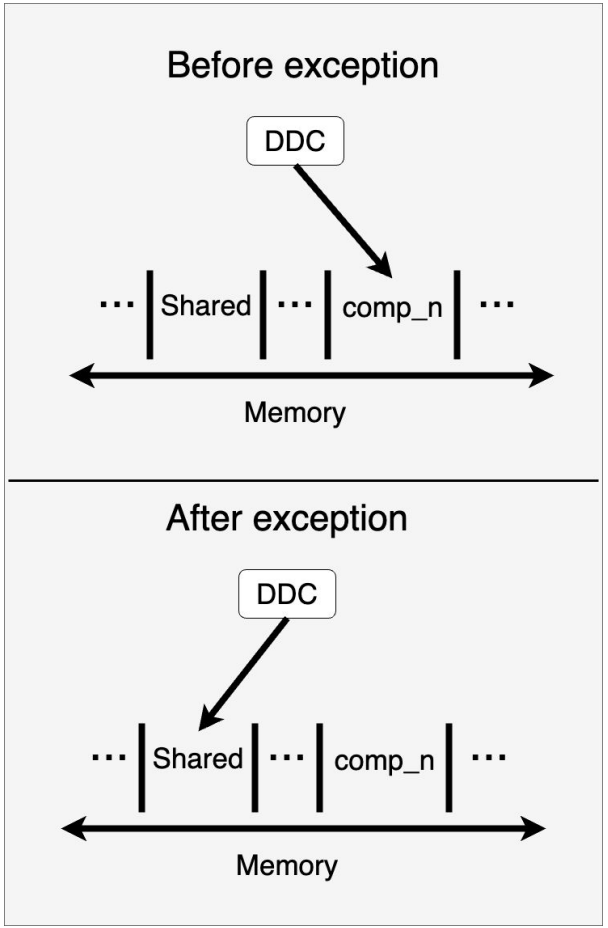


Option 3 - Exception based handling

- Single shared data section
- Use DDC banking to have compartment DDC and shared data DDC
- Switch between them when attempted access triggers capability bound fault
- Preserves the promise of compatibility

But!

- Use of exceptions hurts performance



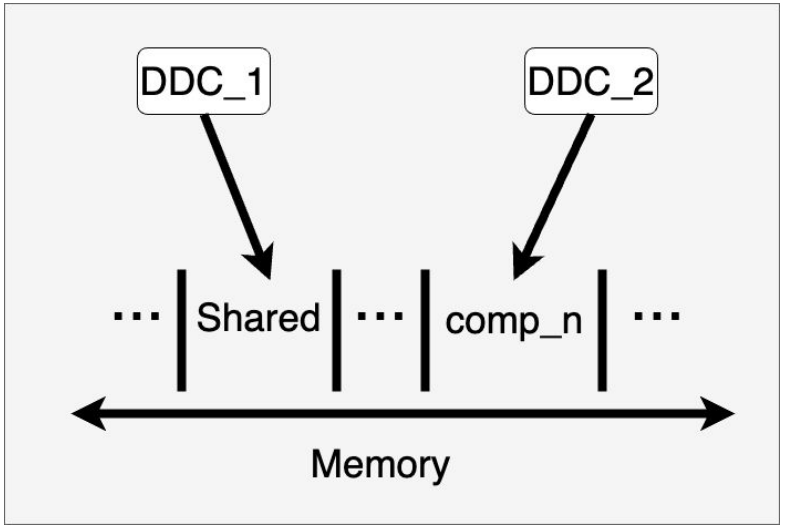
A Possible Solution

Hybrid compartmentalisation of existing complex applications leads to:

- Extensive work required OR
- Does not maintain high performance

To facilitate hybrid compartmentalisation of capability unaware application code:

- Support for non-contiguous memory for DDC
- For example, multiple DDCs
  - One protects the compartment private data DDC_2
  - Second protects shared data DDC_1
- No switching required, no exceptions



Future Plans

Purecap FlexOS

- Port Unikraft and FlexOS to purecap Morello
- Explore compartmentalisation of unikernels in purecap, eliminating pointer issues

Summary

Summary

- First unikernels running on Morello
- Unikraft and FlexOS work with hybrid capabilities on Morello
- Investigated the feasibility of hybrid compartmentalisation for existing applications
- Future work will focus on unikernels and how to compartmentalise existing applications with minimal modifications using purecap

https://unikraft.org/blog/2022-12-01-unikraft-on-morello/

FlexCap is funded by the UK DSbD programme via UKRI

Any Questions?