barkhausen institut

Microkernel Construction

M³

Nils Asmussen

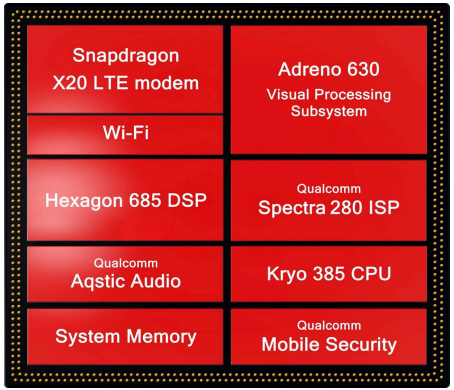
07/02/2026

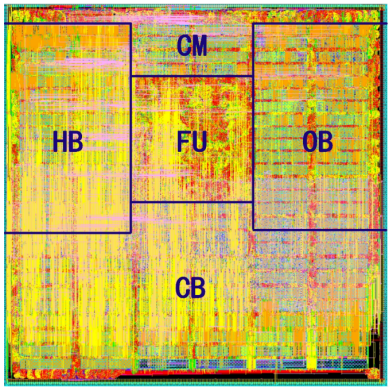
- **Introduction**
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- Evaluation

Motivation

- Microkernel-based systems have proven valuable for several objectives
  - Security
  - Robustness
  - Real time
  - Flexibility
- Recently, new challenges are coming from the hardware side
  - Heterogeneous systems
  - Third-party components
  - Security issues of complex general-purpose cores

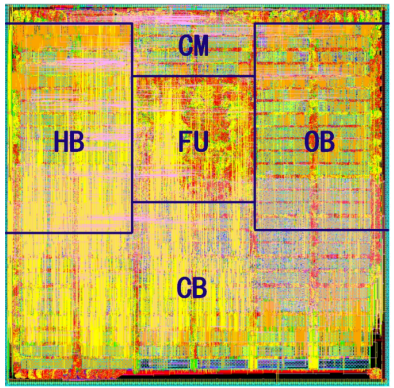
Heterogeneous Systems

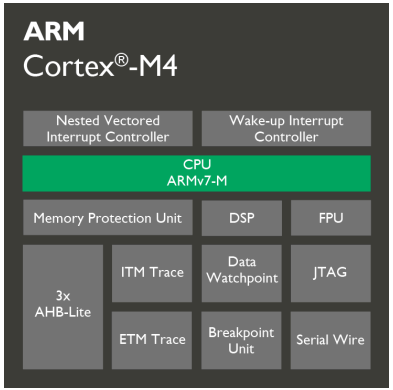




- Demanded by performance and energy requirements
- Big challenge for OSes: single shared kernel on all cores does no longer work
- OSes need to be prepared for processing elements with different feature sets

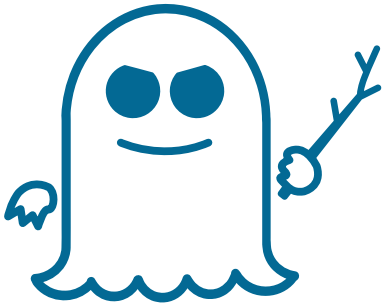
Third-party Components





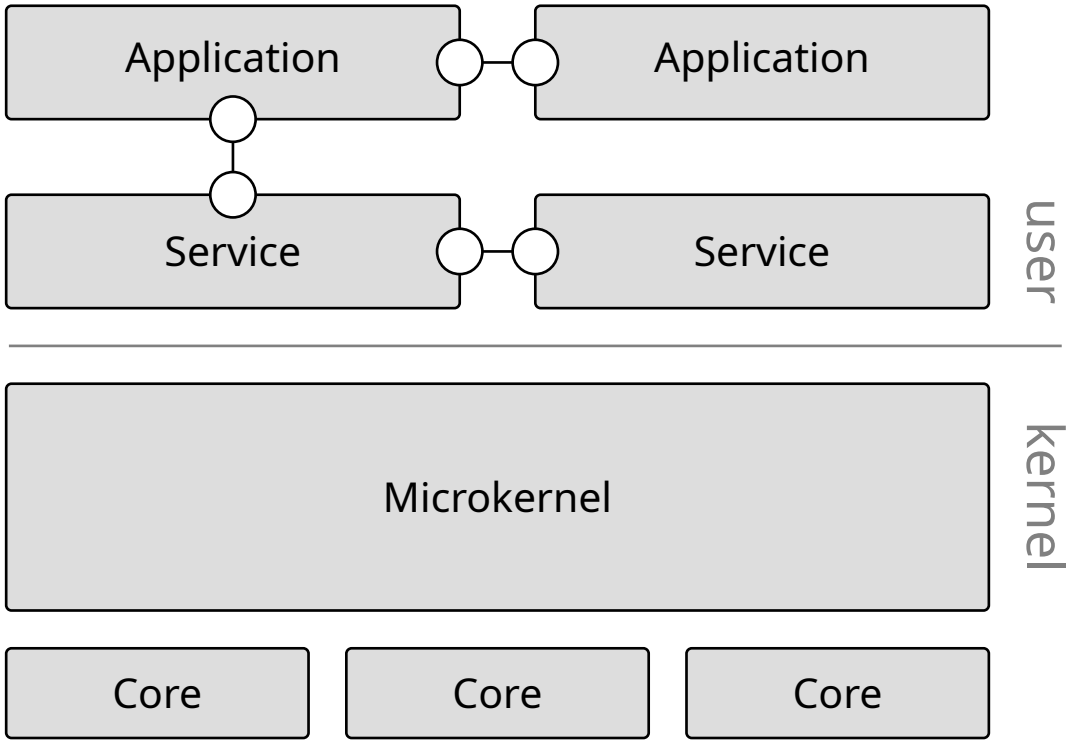
- Market pressure forces us to integrate third-party components
- We should not trust these components
- Currently, often no isolation between them
- Bug in them can compromise whole system (see Broadpwn)

Security Issues of Complex General-purpose Cores

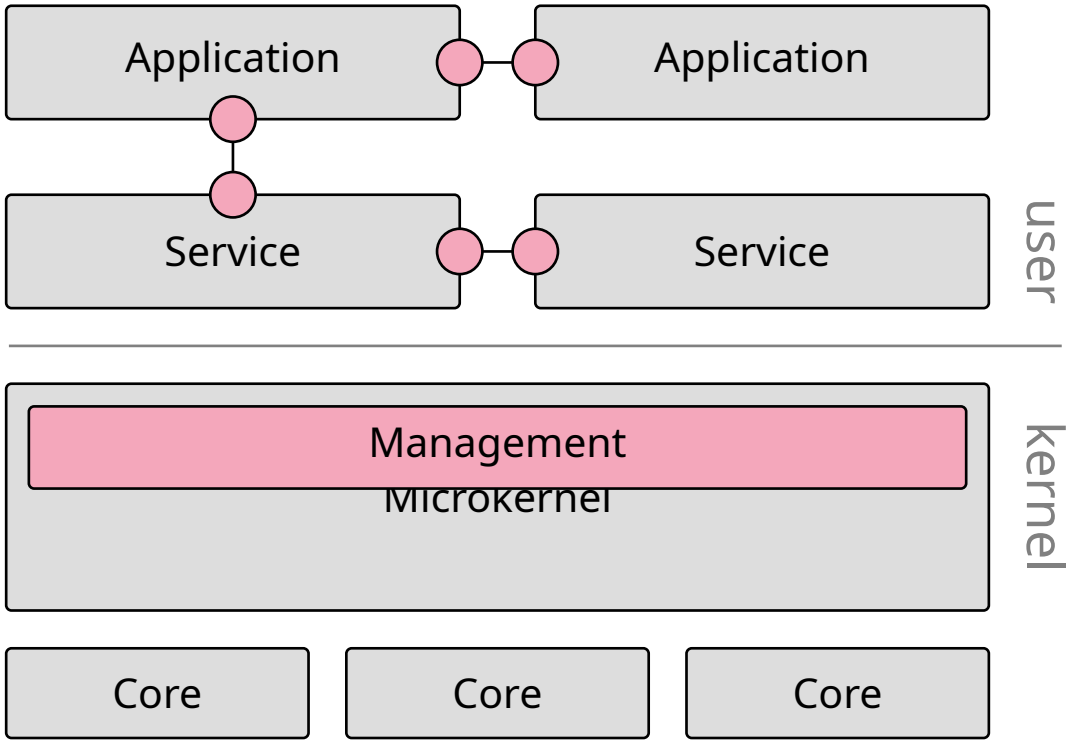


- 37 known CVEs (and counting …)
- Allow to leak private data (bypassing the core’s security measures)
- Mitigations exist, but these are complex and costly
- These security holes have been lurking in CPUs for many years
- Should we still trust these complex cores to properly enforce the isolation between different software components?

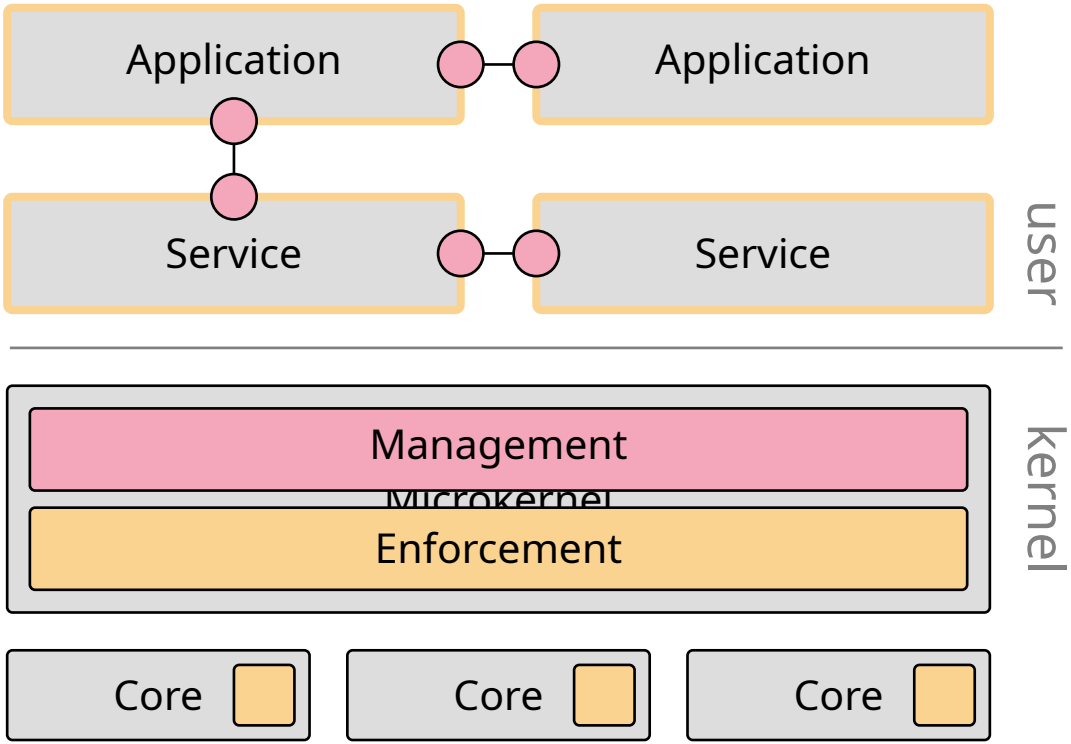
Microkernel-based System as Foundation



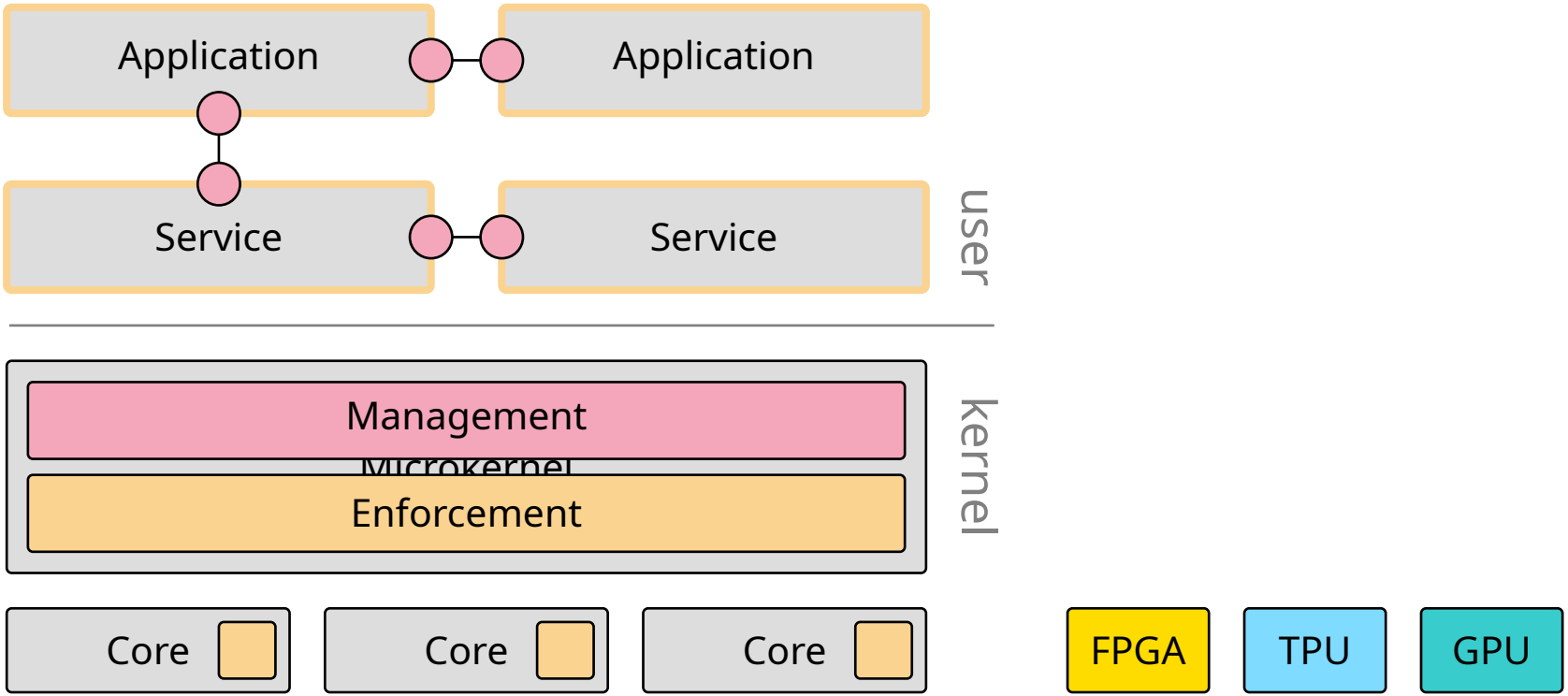
Microkernel-based System as Foundation



Microkernel-based System as Foundation



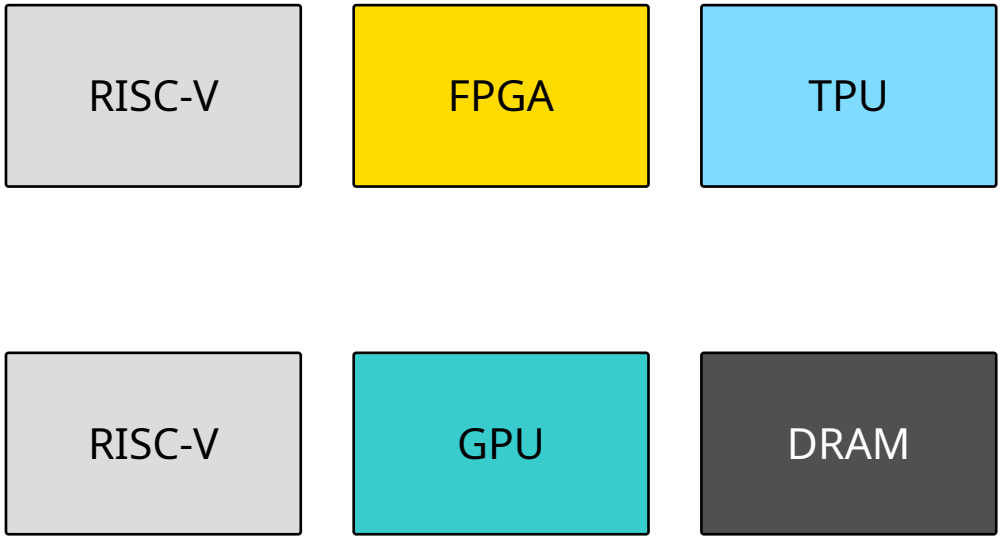
Microkernel-based System as Foundation



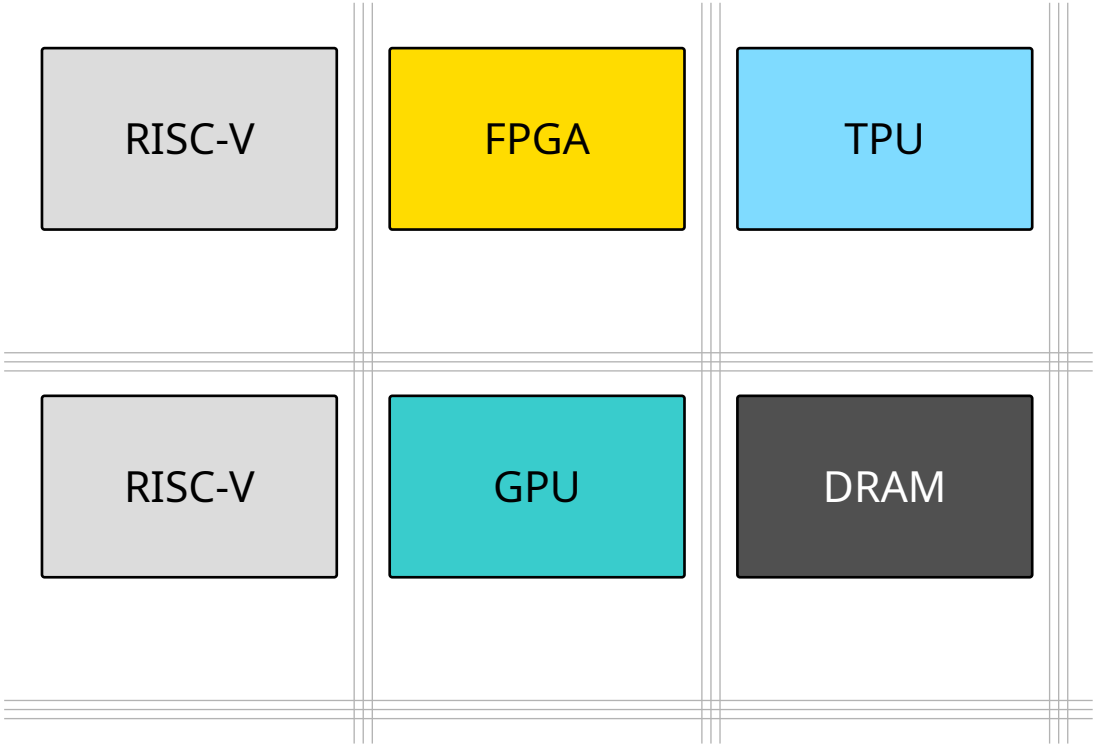
- Introduction
- **The New System Architecture**
- Prototype Platforms
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- Evaluation

M³: Hardware/Operating-System Co-Design

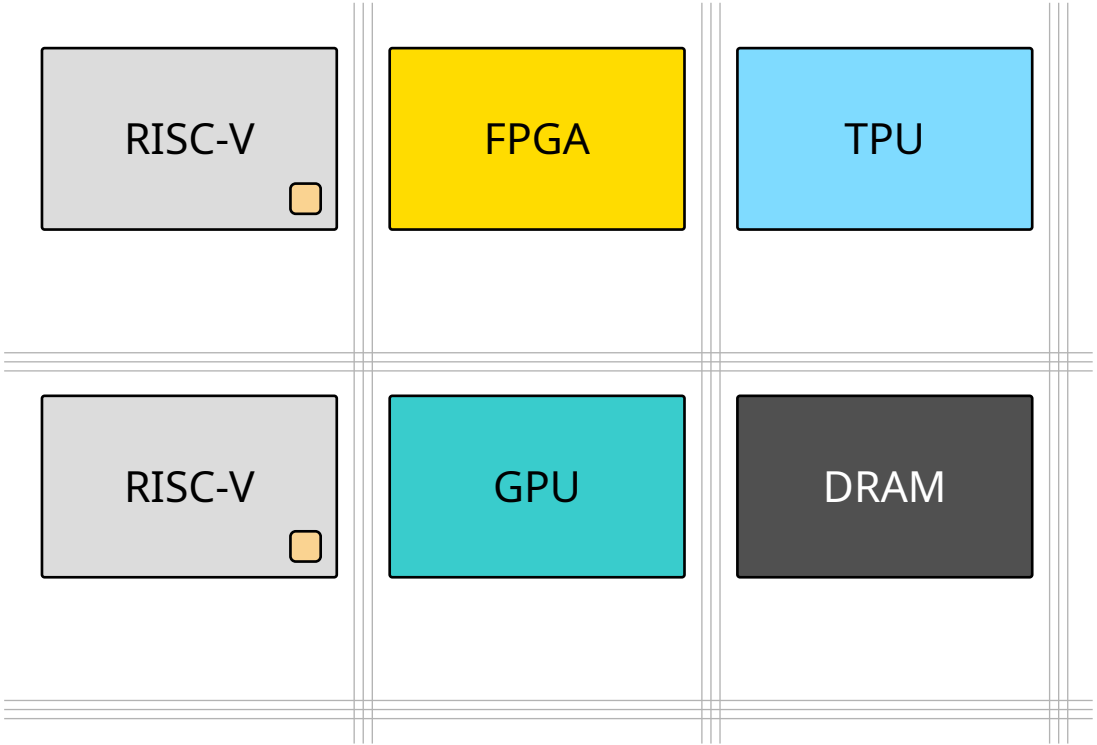
M³: Hardware/Operating-System Co-Design



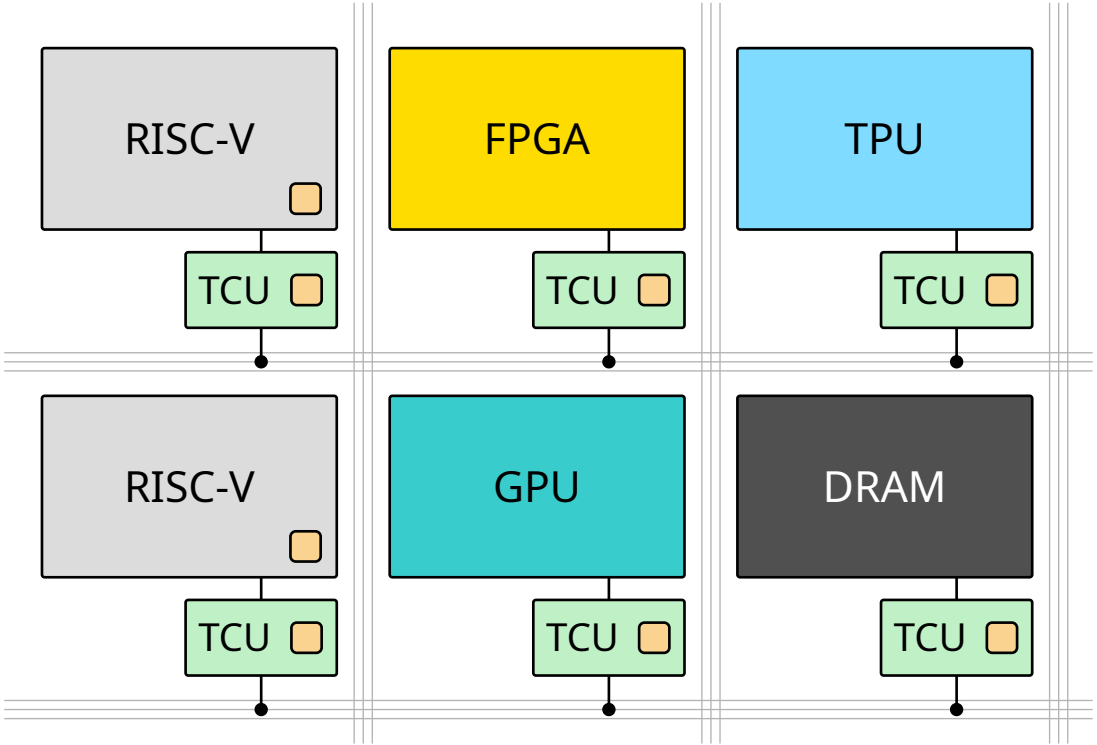
M³: Hardware/Operating-System Co-Design



M³: Hardware/Operating-System Co-Design



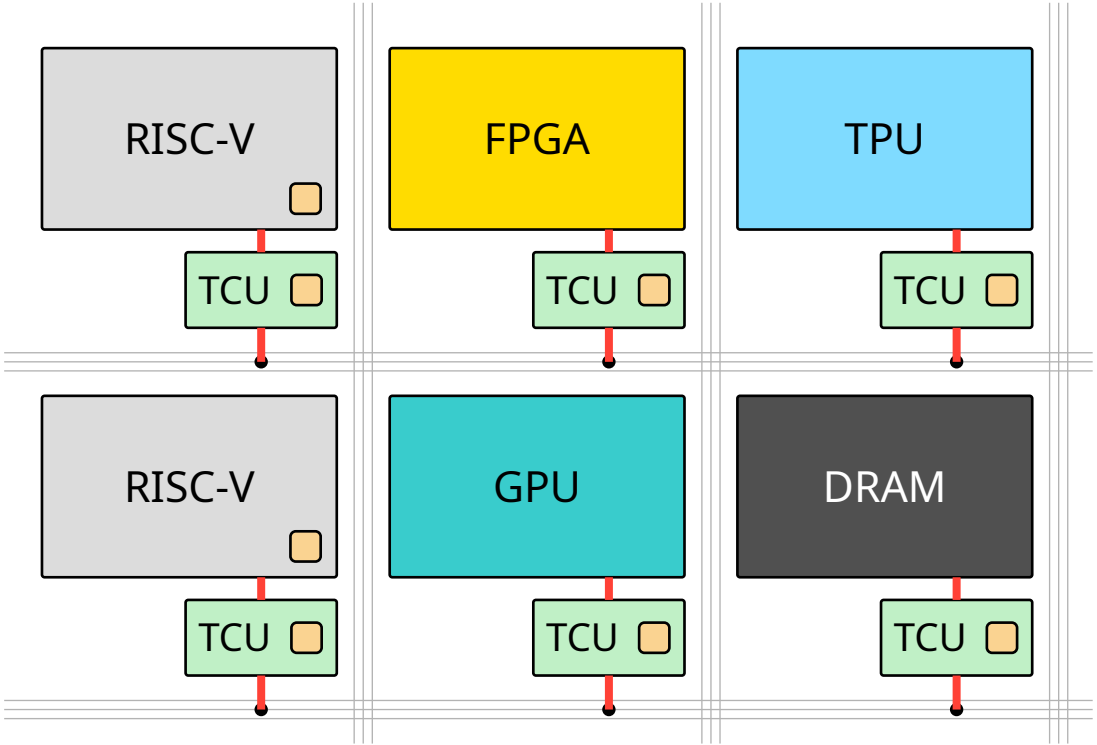
M³: Hardware/Operating-System Co-Design



Key ideas:

- **TCU as new HW component**

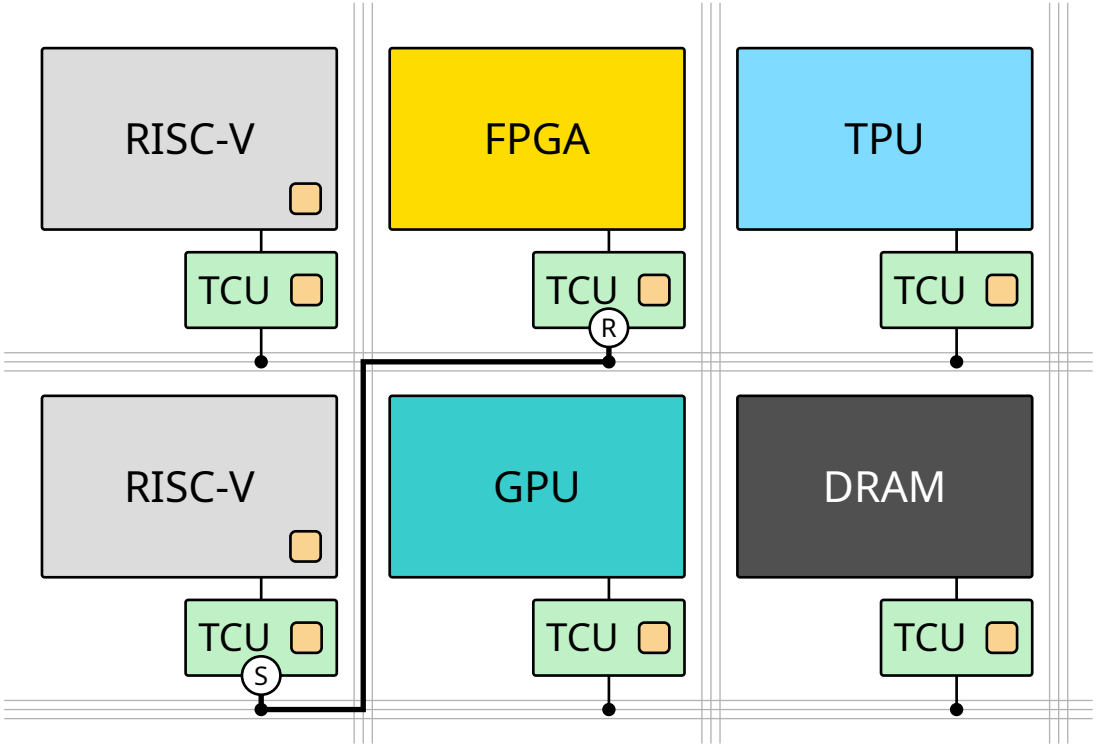
M³: Hardware/Operating-System Co-Design



Key ideas:

- TCU as new HW component
- **Tiles are isolated**

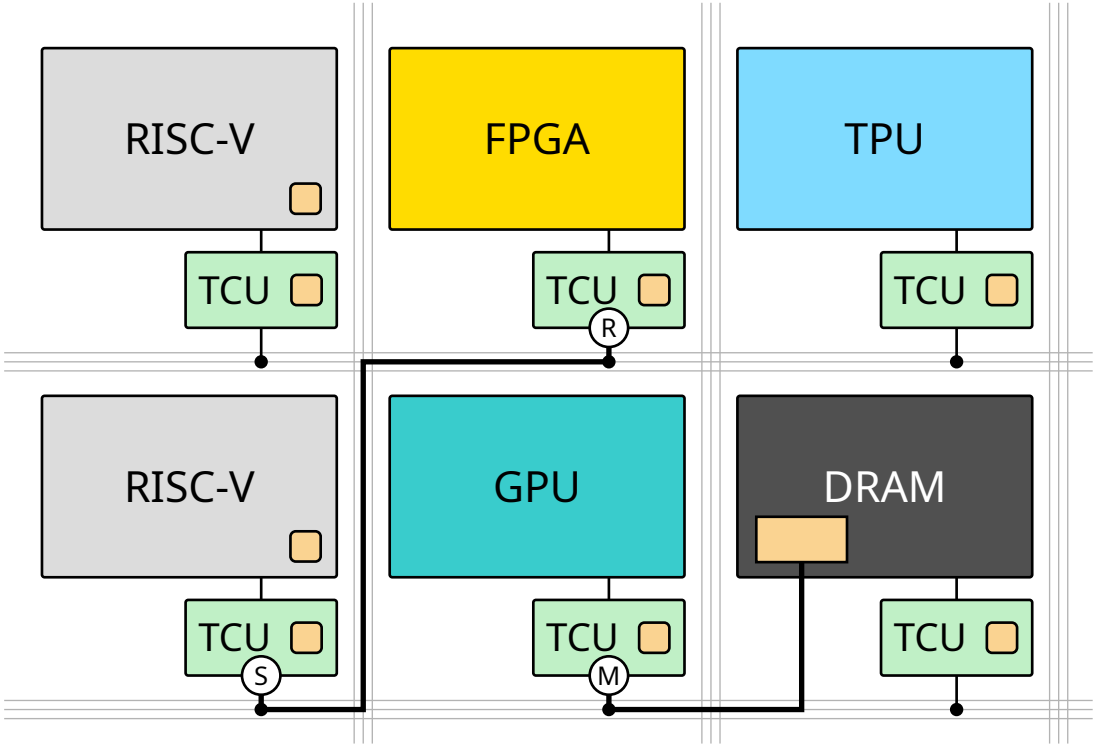
M³: Hardware/Operating-System Co-Design



Key ideas:

- TCU as new HW component
- Tiles are isolated
- **Selective communication**

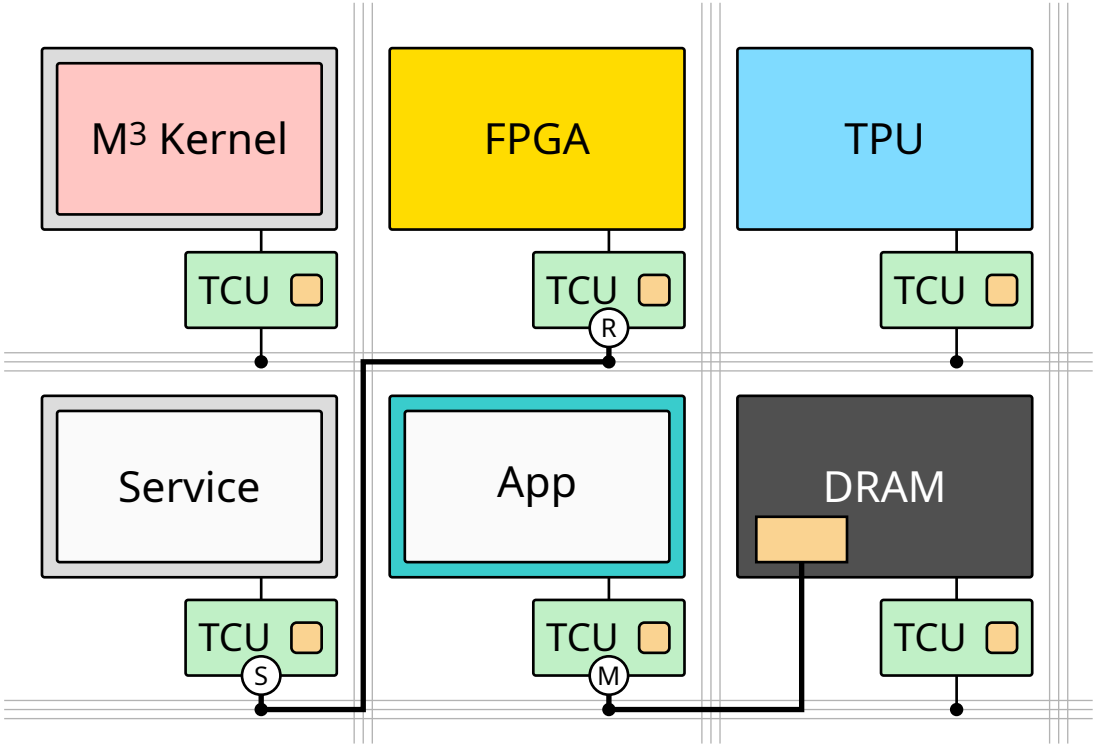
M³: Hardware/Operating-System Co-Design



Key ideas:

- TCU as new HW component
- Tiles are isolated
- **Selective communication**

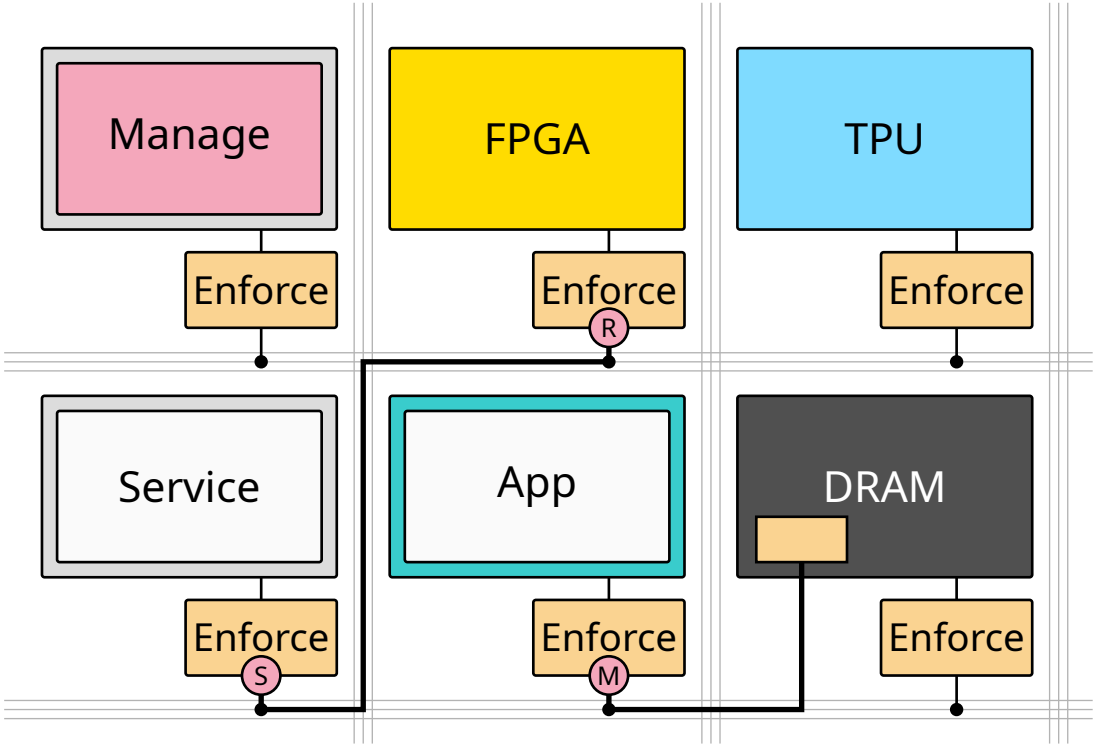
M³: Hardware/Operating-System Co-Design



Key ideas:

- TCU as new HW component
- Tiles are isolated
- Selective communication
- **Dedicated kernel tile**

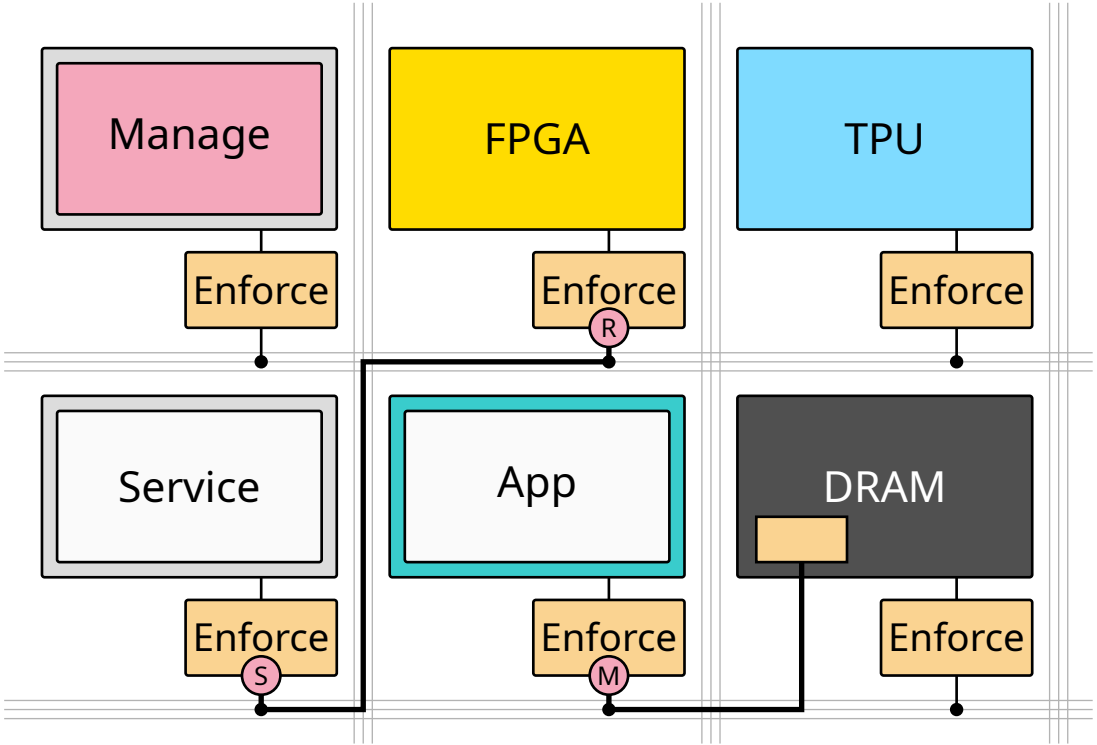
M³: Hardware/Operating-System Co-Design



Key ideas:

- TCU as new HW component
- Tiles are isolated
- Selective communication
- Dedicated kernel tile
- **Kernel manages, TCU enforces**

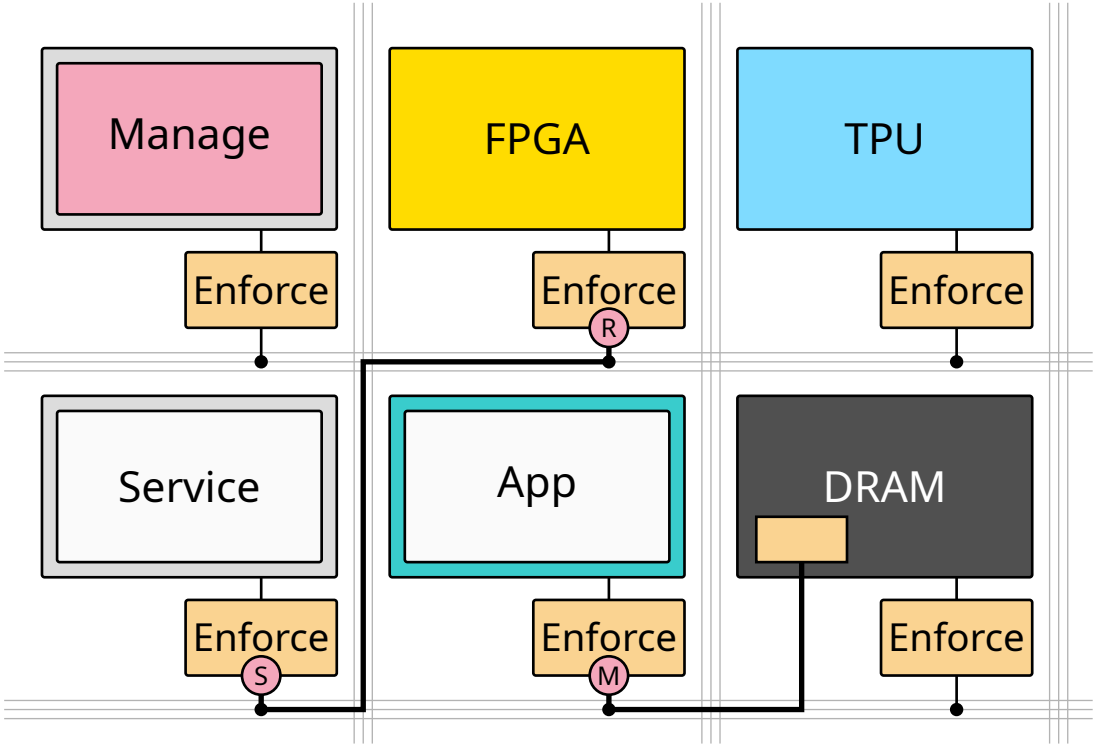
M³: Hardware/Operating-System Co-Design



Hardware challenges:

- **Heterogeneity: Uniform interface**

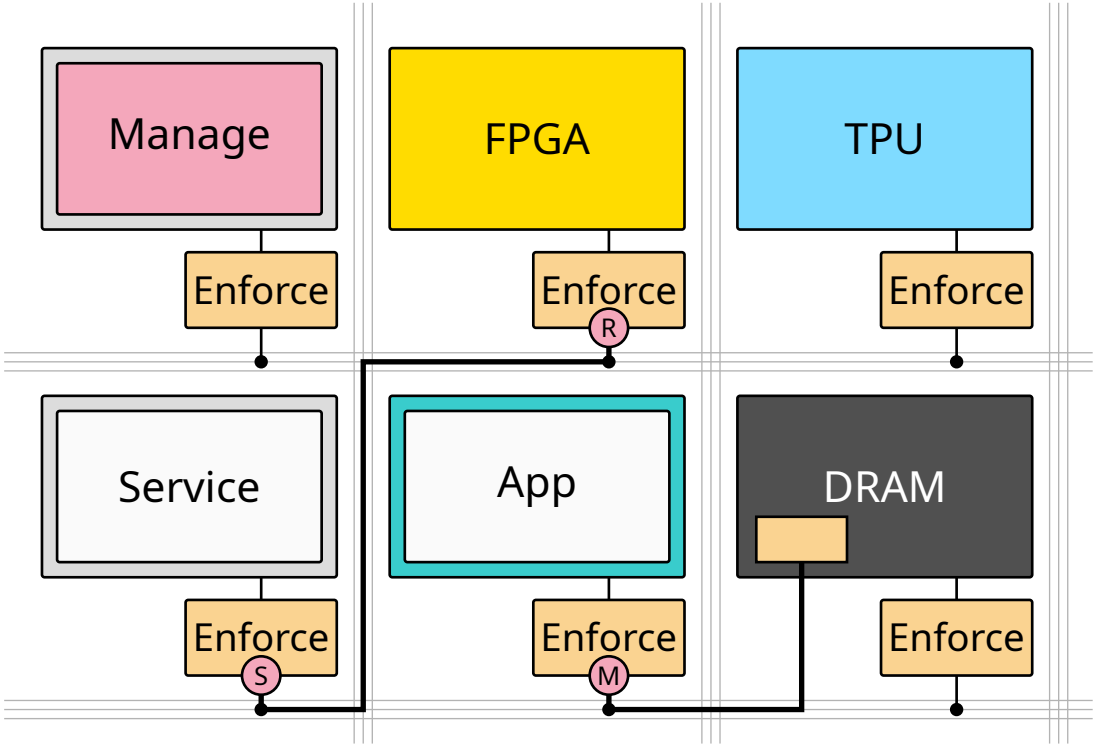
M³: Hardware/Operating-System Co-Design



Hardware challenges:

- Heterogeneity: Uniform interface
- **Untrusted HW comp: Protected by TCU**

M³: Hardware/Operating-System Co-Design



Hardware challenges:

- Heterogeneity: Uniform interface
- Untrusted HW comp: Protected by TCU
- **Side channels: Physical isolation**

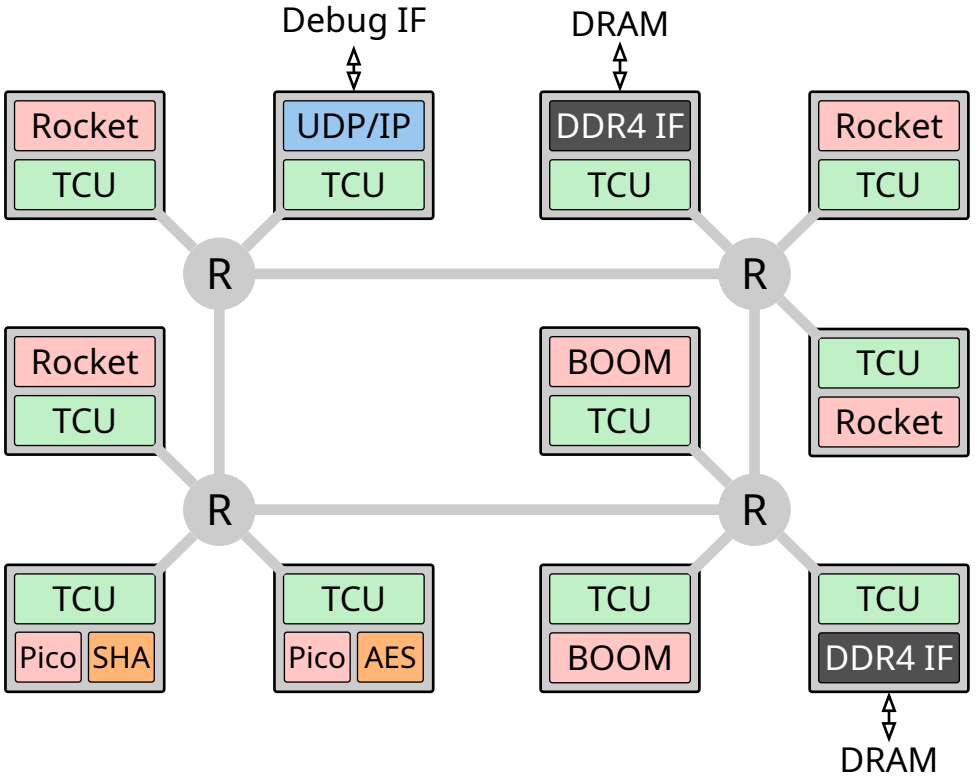
- Introduction
- The New System Architecture
- **Prototype Platforms**
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- Evaluation

gem5 Platform

- Modular platform for computer-architecture research
- Supports various ISAs (x86, ARM, Alpha, RISC-V, ...)
- Provides detailed CPU and memory models
- Cycle-accurate simulation
- Added TCU model to gem5
- Added hardware accelerators

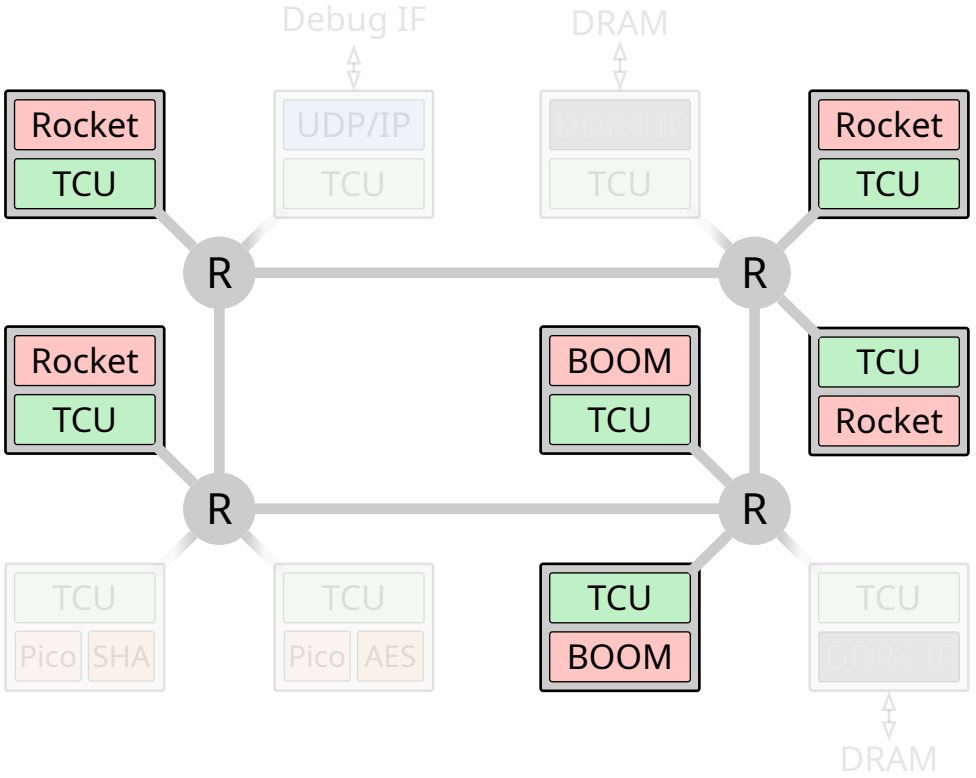
FPGA Platform

- Xilinx VCU118 FPGA
- TCU with 128 EPs



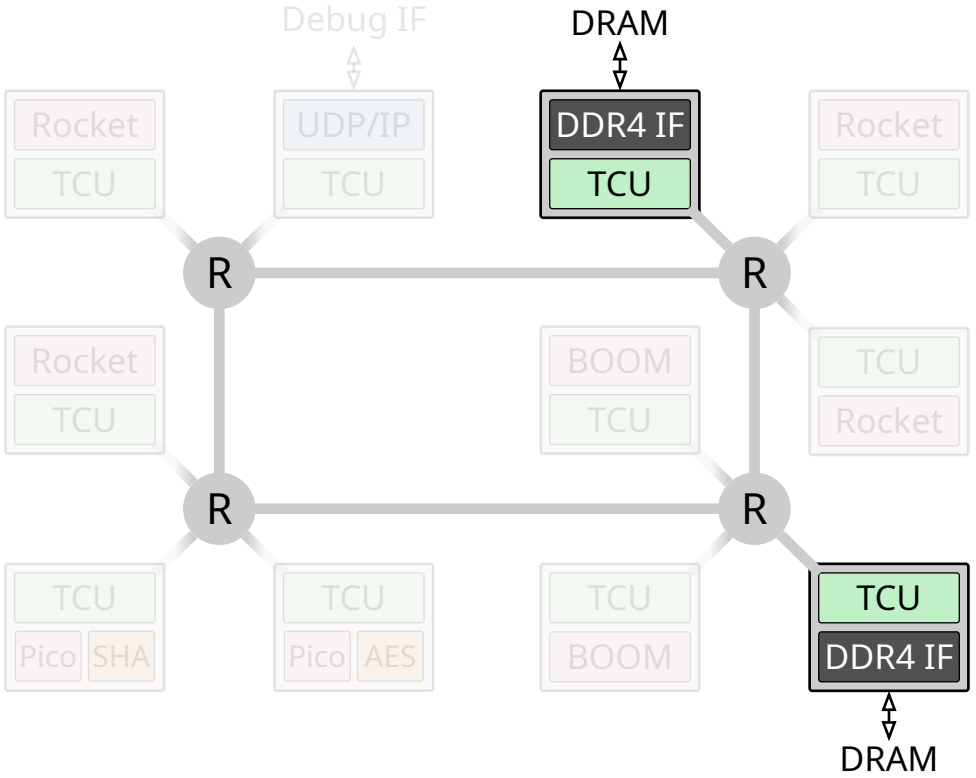
FPGA Platform

- Xilinx VCU118 FPGA
- TCU with 128 EPs
- RISC-V Rocket/BOOM Cores
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2



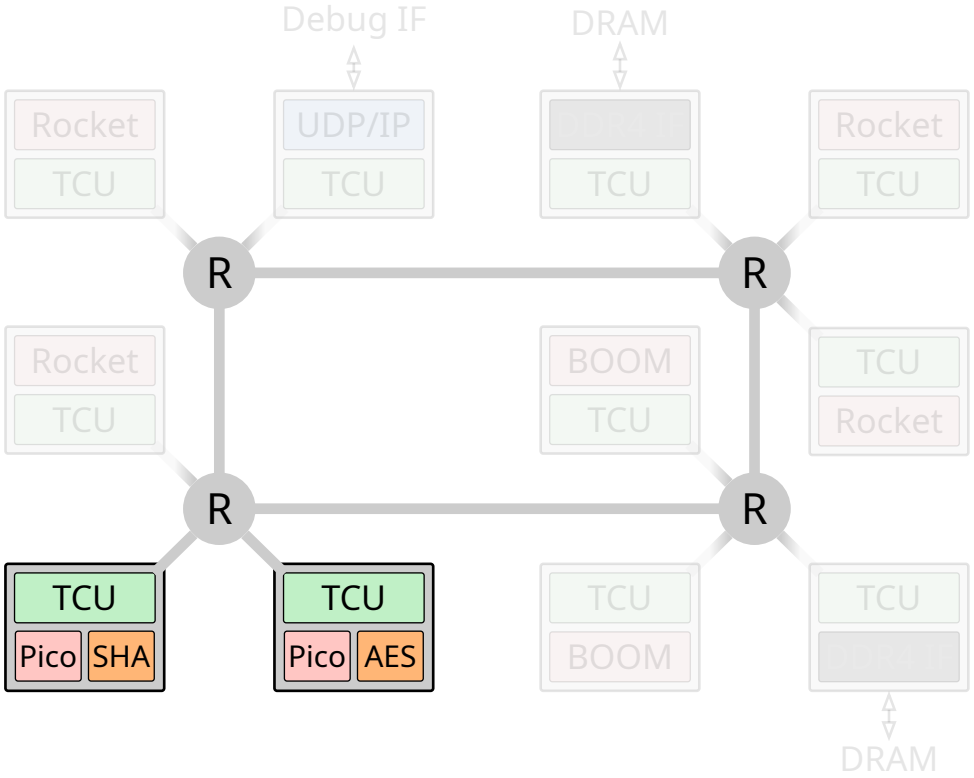
FPGA Platform

- Xilinx VCU118 FPGA
- TCU with 128 EPs
- RISC-V Rocket/BOOM Cores
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- 2 GiB off-chip DRAM

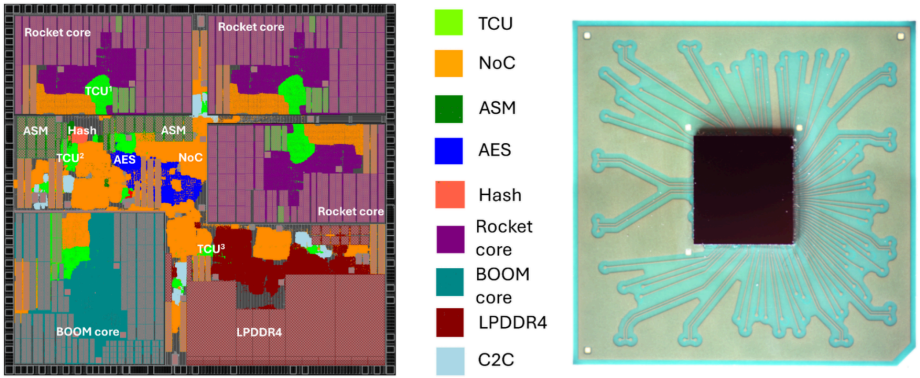


FPGA Platform

- Xilinx VCU118 FPGA
- TCU with 128 EPs
- RISC-V Rocket/BOOM Cores
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- 2 GiB off-chip DRAM
- Accelerators with PicoRV32
- SHA-3 (RoT) and AES



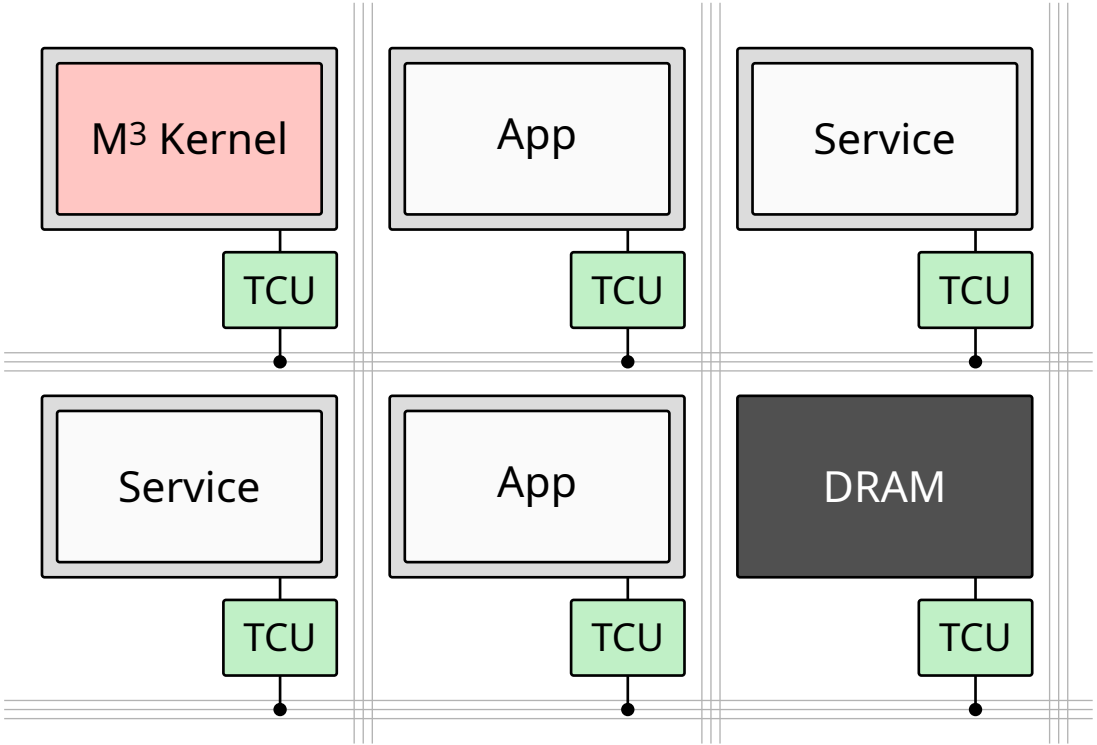
Silicon Platform



- The Barkhausen Institut builds actual chips!
- Fabricated at Global Foundaries in Dresden
- 22 nm FD-SOI process technology
- Resulting in a 3.94 mm × 3.64 mm flip-chip (top right)

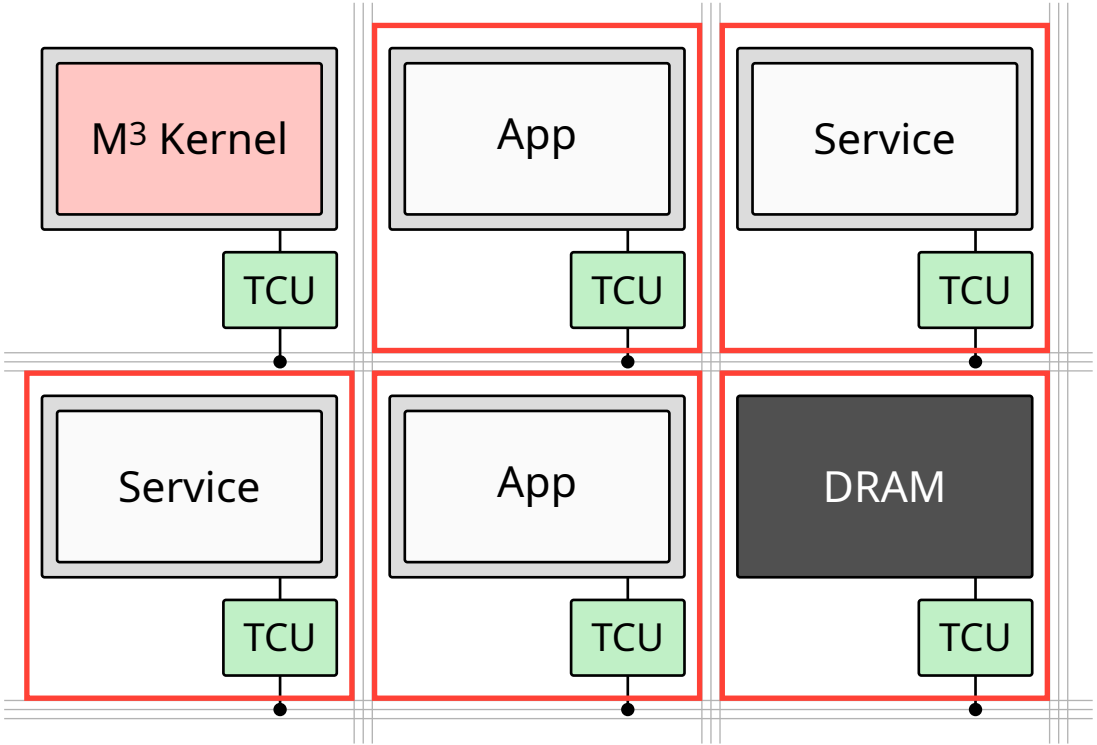
- Introduction
- The New System Architecture
- Prototype Platforms
- **Isolation and Communication**
- Operating System
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- Evaluation

Isolation



TCU-based isolation:

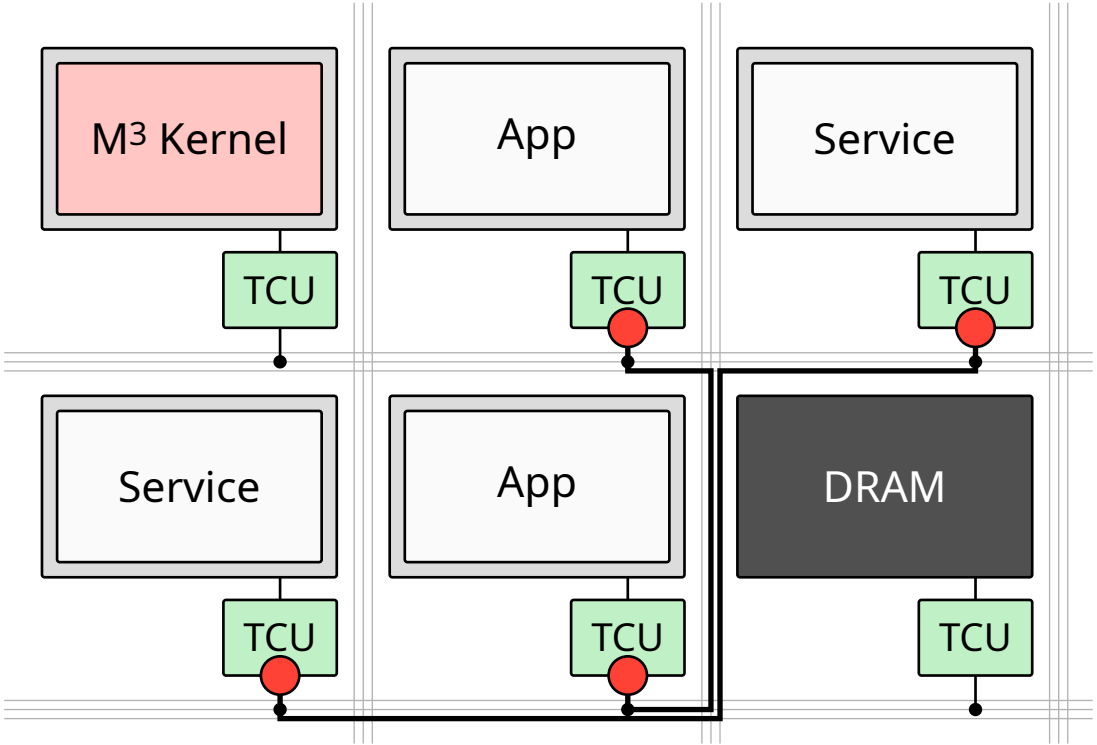
Isolation



TCU-based isolation:

- **Additional prot. layer**

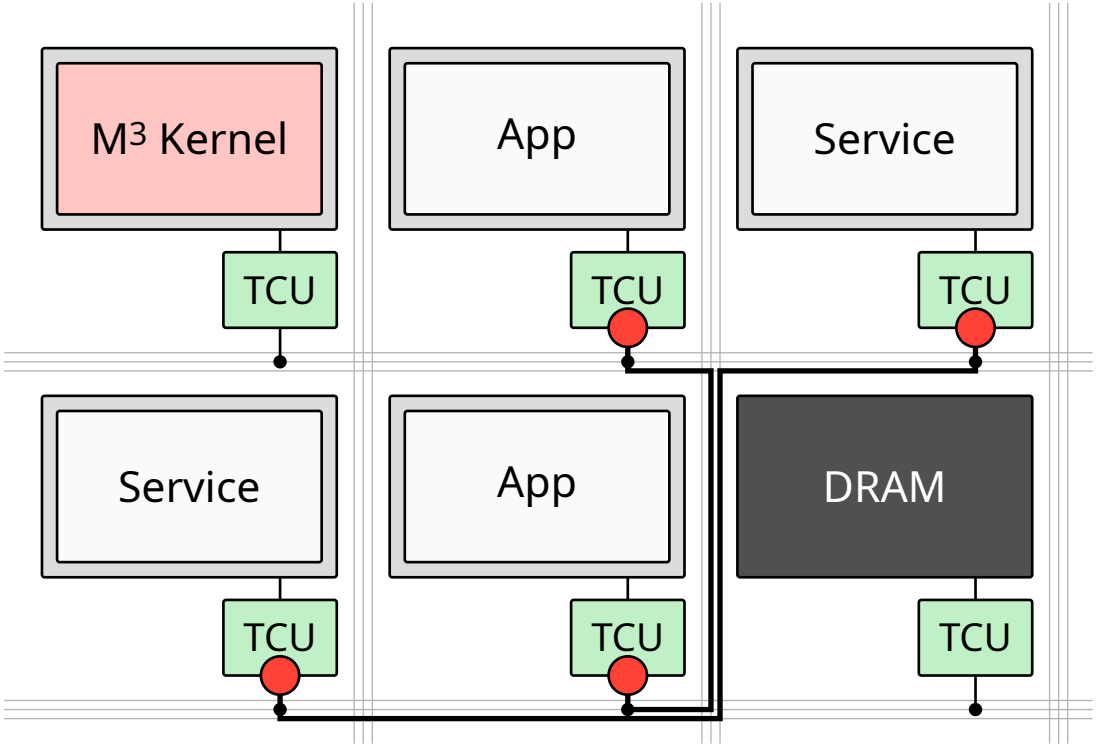
Isolation



TCU-based isolation:

- Additional prot. layer
- **Only kernel tile can establish channels**

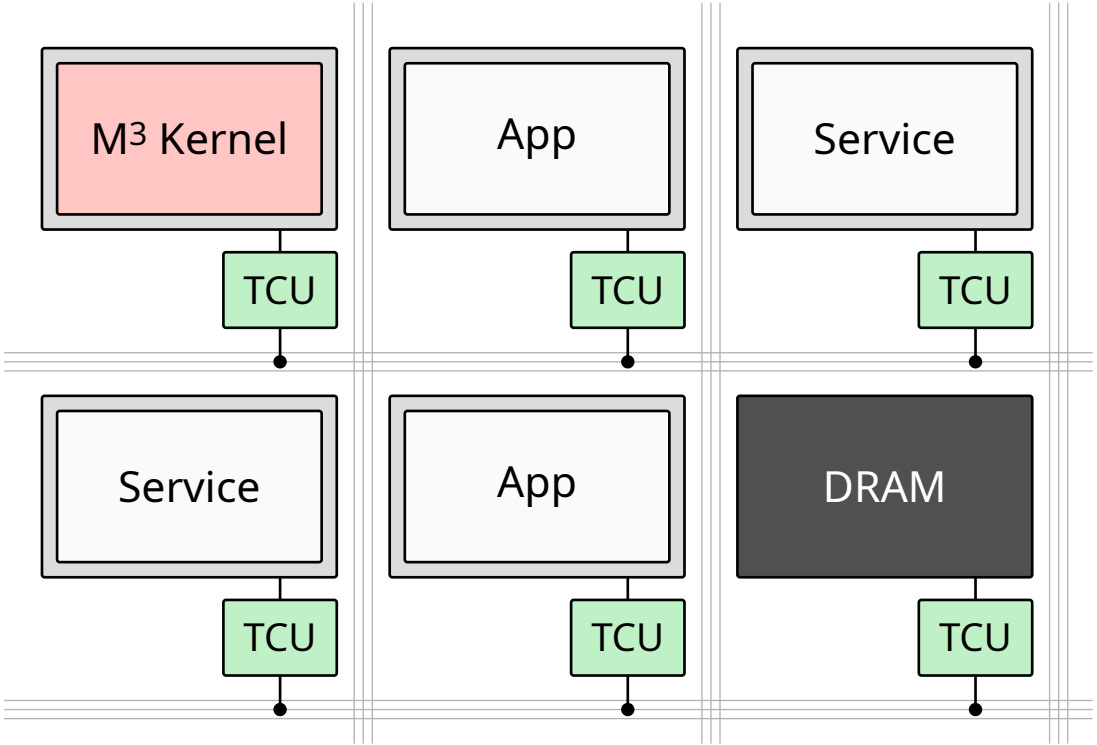
Isolation



TCU-based isolation:

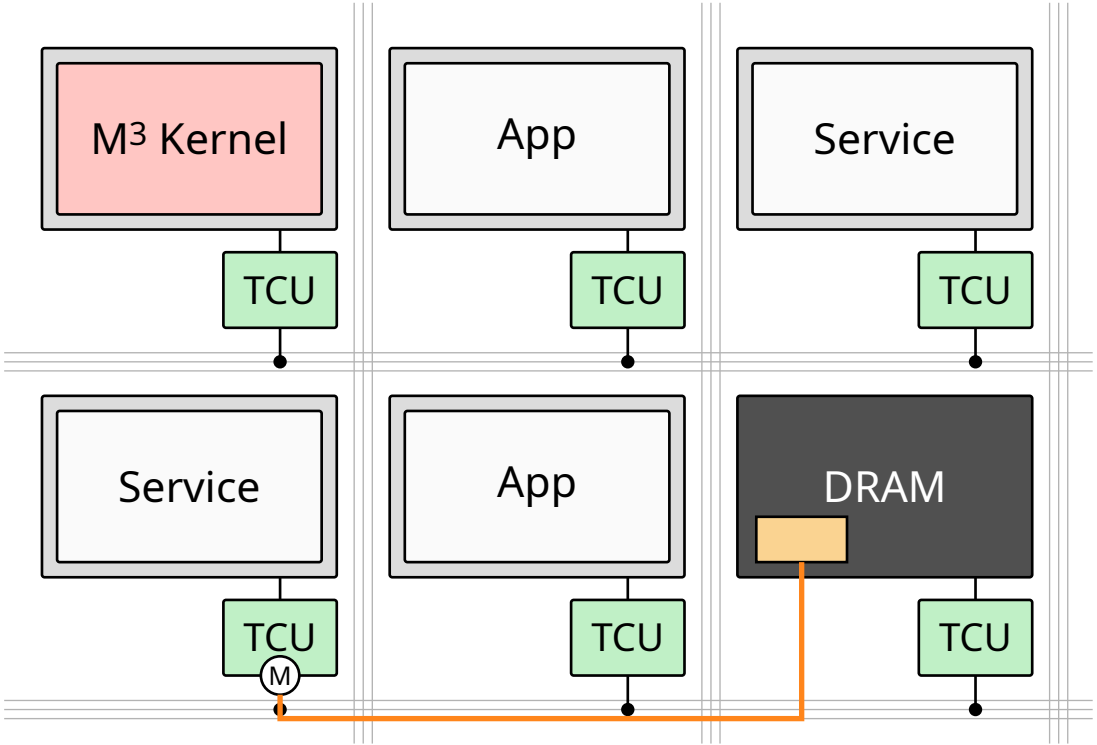
- Additional prot. layer
- Only kernel tile can establish channels
- **User tiles can only used established channels**

Communication



TCU provides *endpoints*

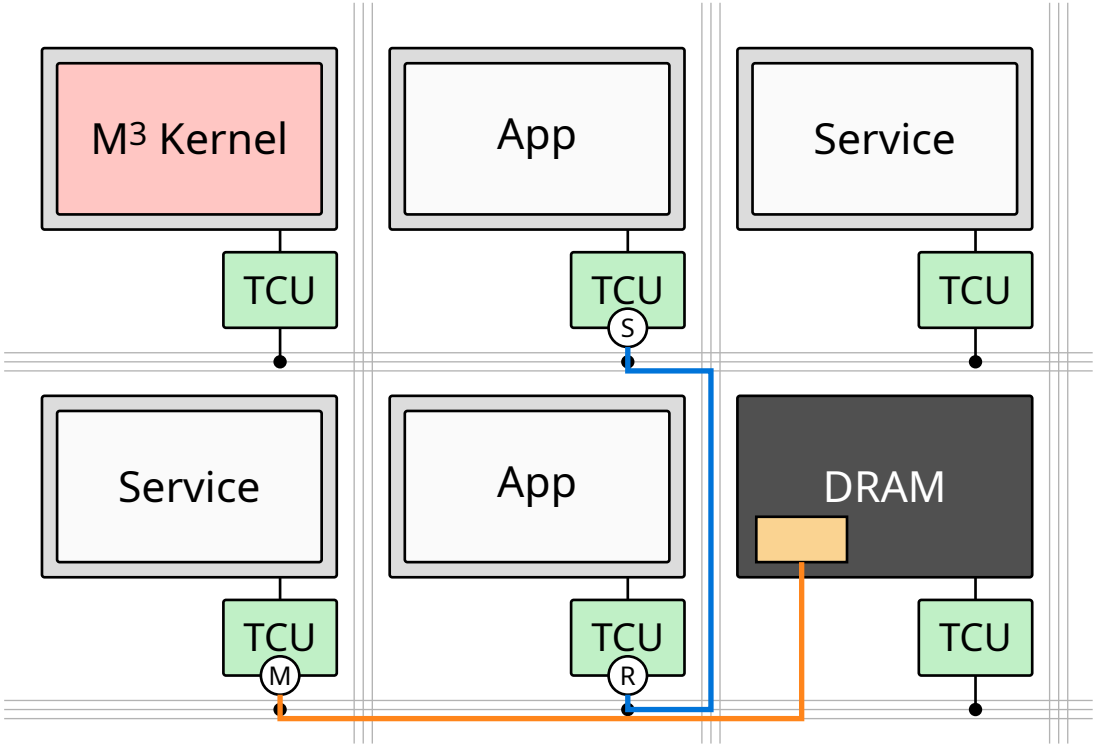
Communication



TCU provides *endpoints*

- **Issue DMA requests to memory**

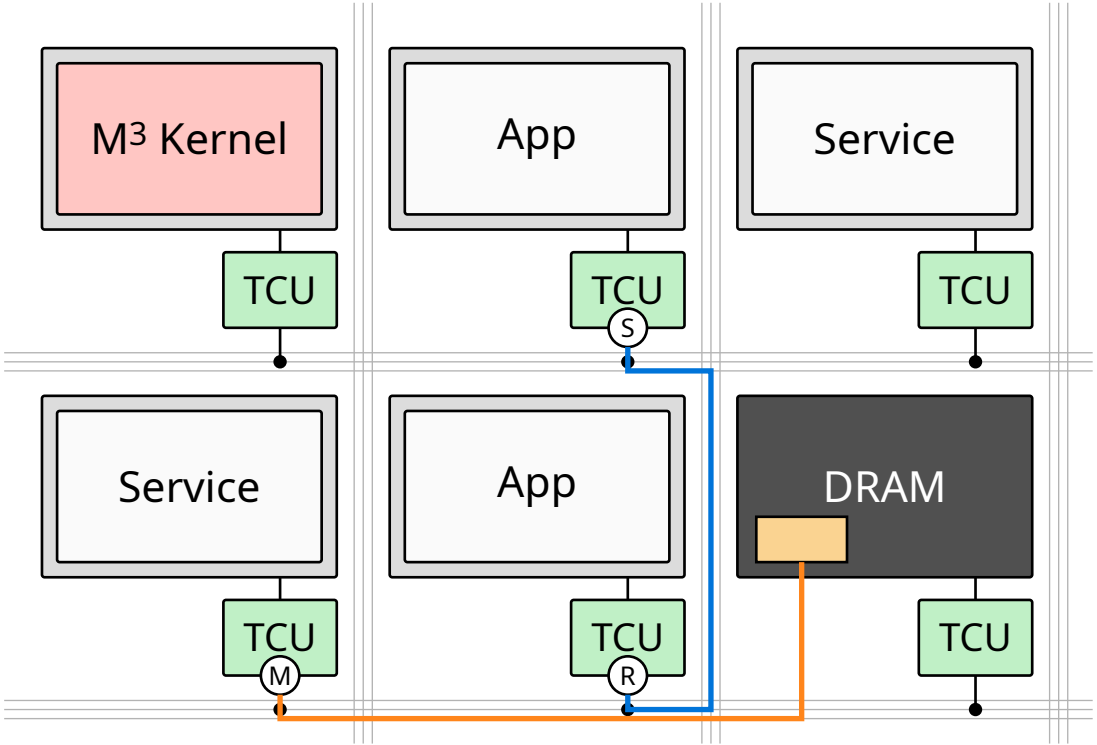
Communication



TCU provides *endpoints*

- Issue DMA requests to memory
- **Receive messages into a receive buffer**

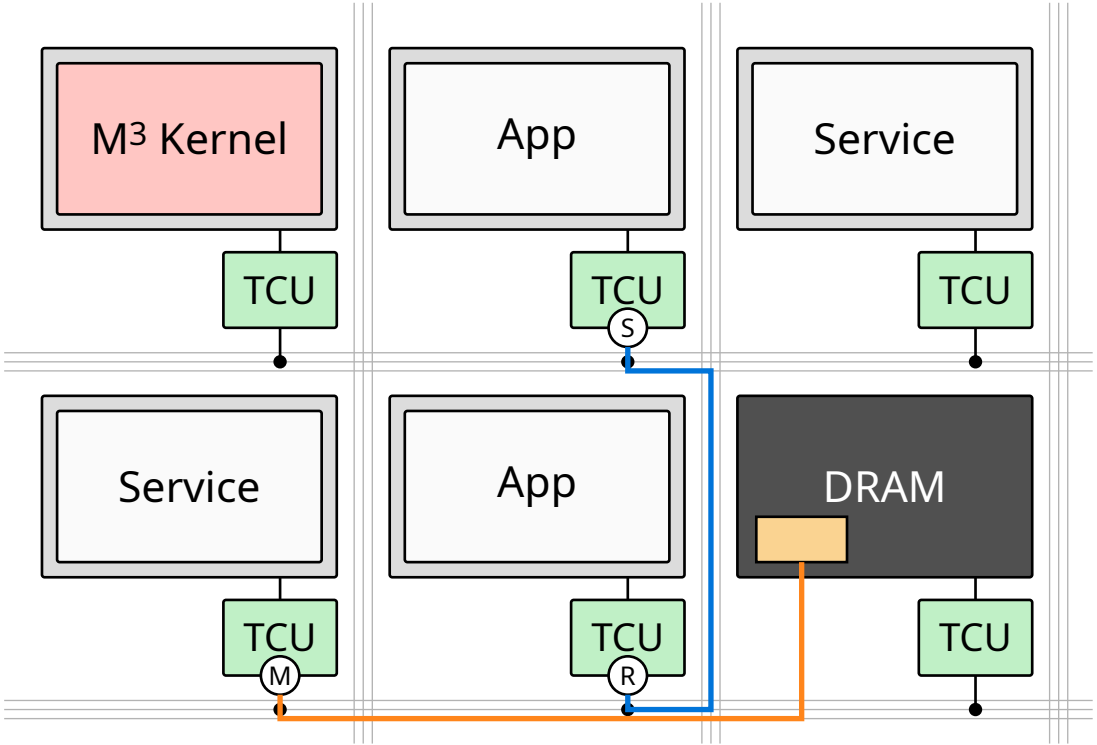
Communication



TCU provides *endpoints*

- Issue DMA requests to memory
- Receive messages into a receive buffer
- **Send messages to a receiving endpoint**

Communication



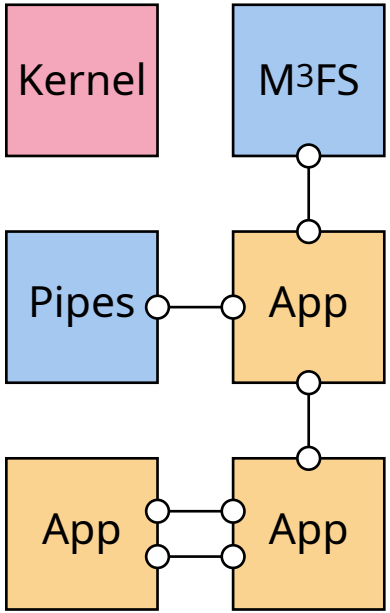
TCU provides *endpoints*

- Issue DMA requests to memory
- Receive messages into a receive buffer
- Send messages to a receiving endpoint
- **Replies for RPC**

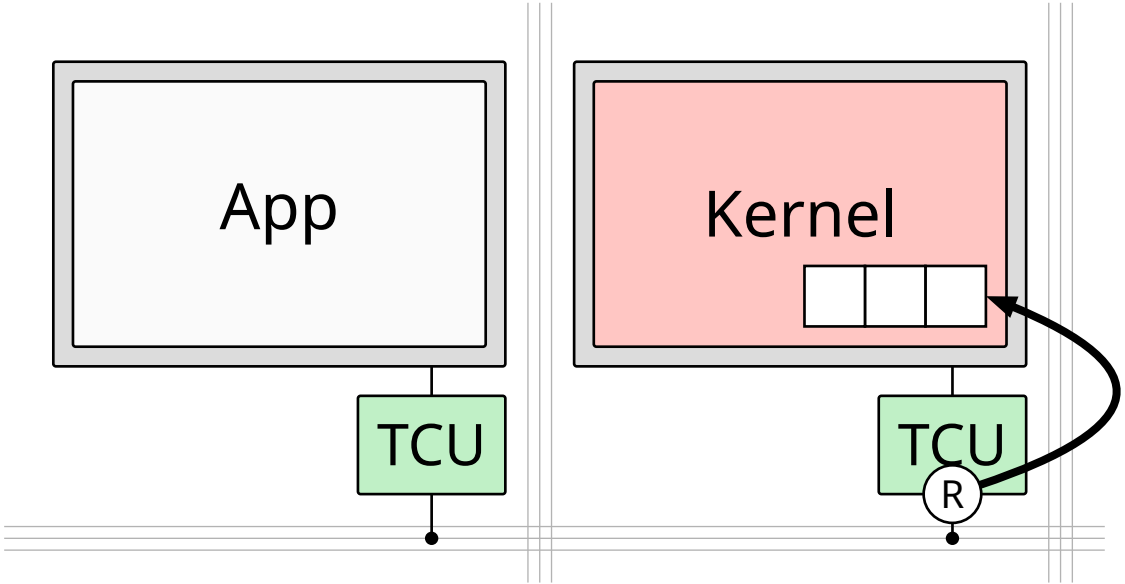
- Introduction
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- **Operating System**
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- Evaluation

Operating-System Design

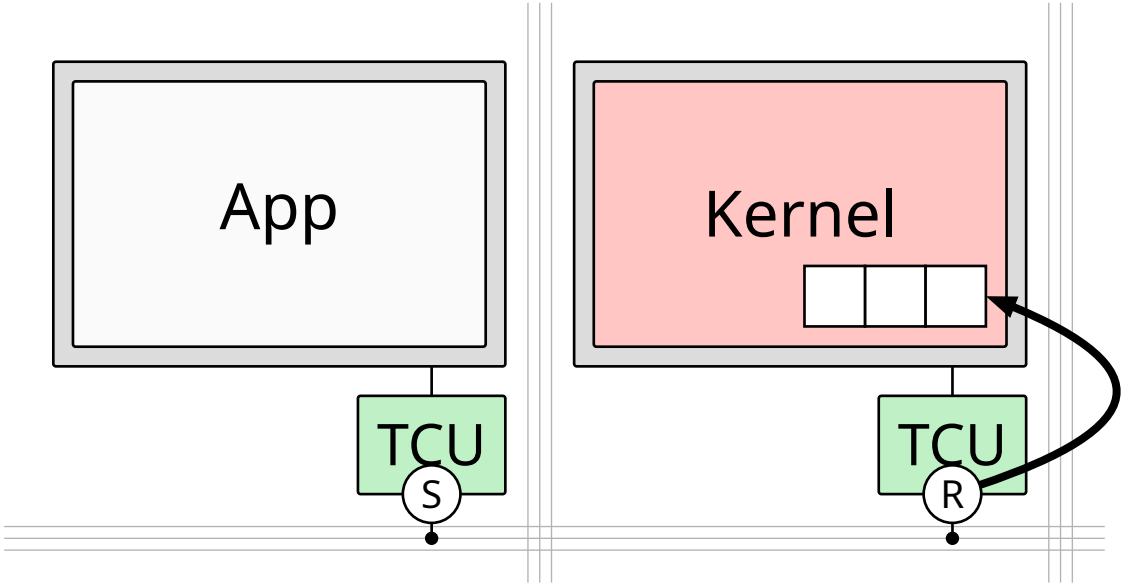
- M³: **M**icrokernel-based syste**m** for heterogeneous **m**anycores (or L4 ± 1)
- Implemented from scratch in Rust and C++
- Drivers, filesystems, etc. implemented on user tiles
- Kernel manages permissions, using capabilities
- TCU enforces permissions (communication, memory access)
- Kernel is independent of other tiles



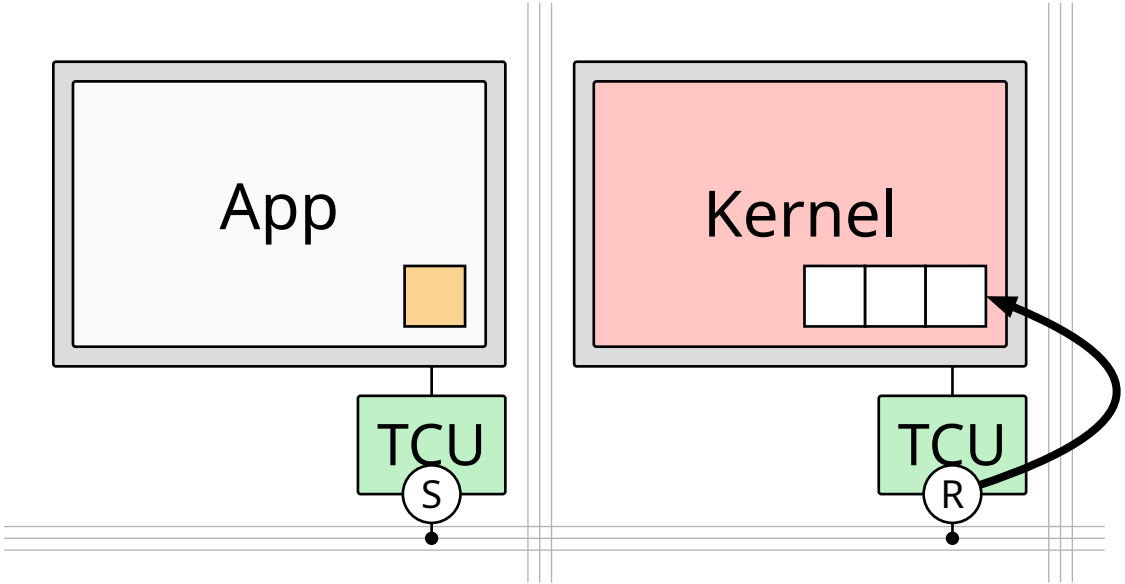
M³ System Call



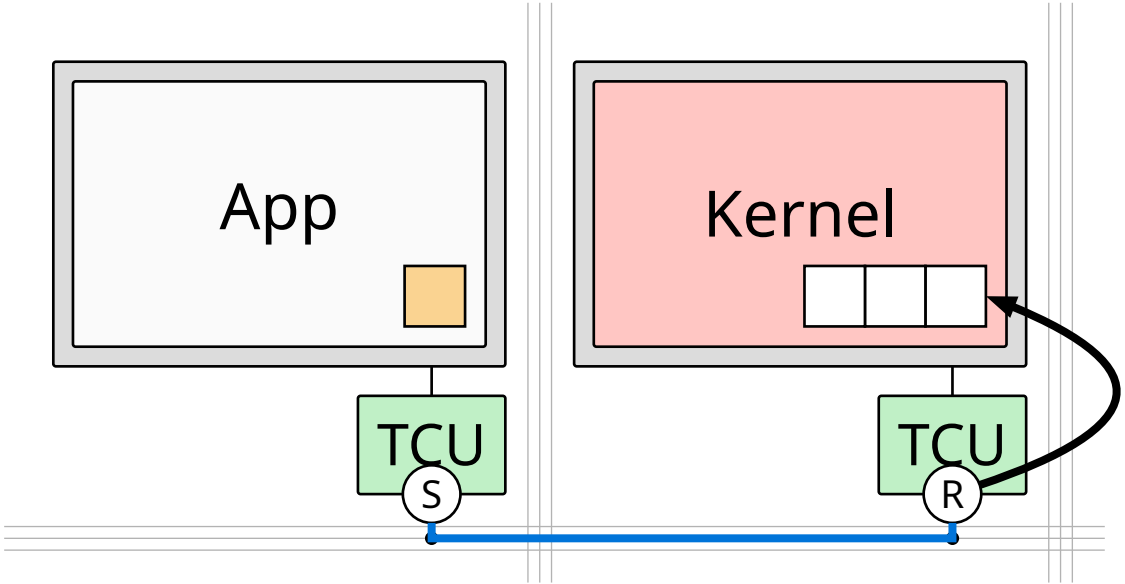
M³ System Call



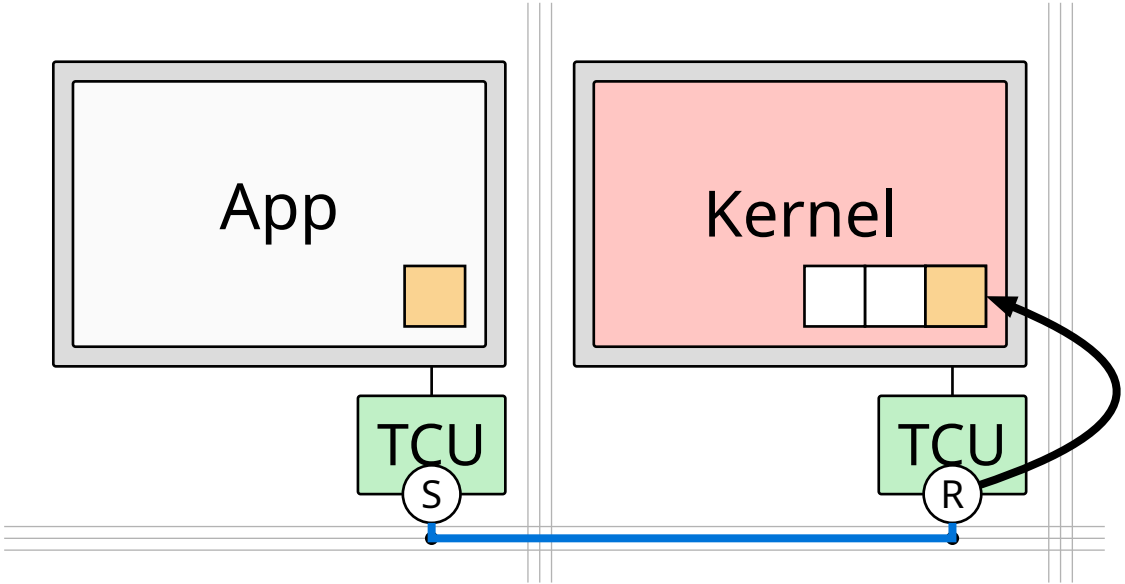
M³ System Call



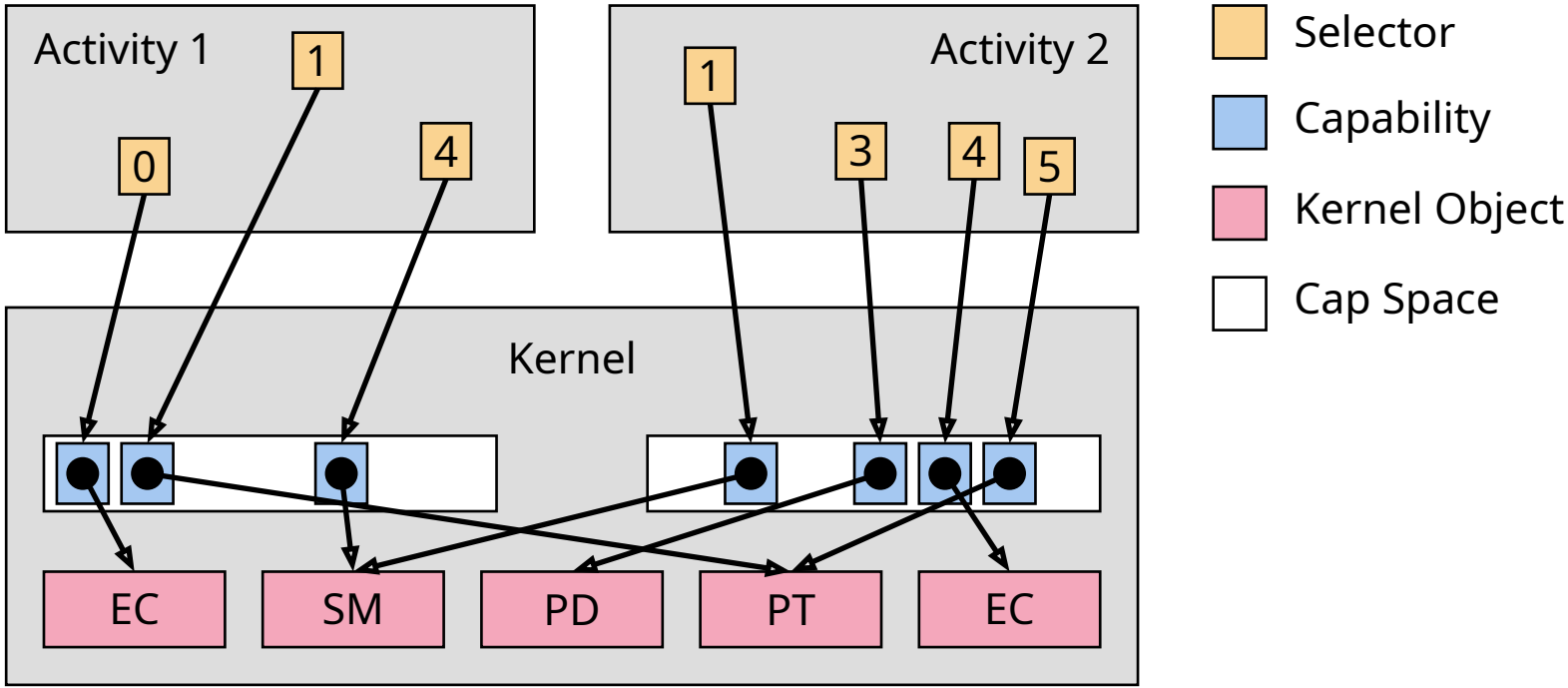
M³ System Call



M³ System Call



Capabilities Overview



Capabilities in M³

Send	Send messages to a receive EP
Receive	Receive messages from send EPs
Memory	Issue DMA requests to memory
Service	Create sessions
Session	Exchange caps with service
Endpoint	Configure EPs of own or foreign TCU
Tile	Create activities
Activity	Executes code on/uses logic of a tile

Capability Exchange

- Kernel provides syscalls to create, exchange, and revoke caps
- There are two ways to exchange caps:
  1. Directly with another activity (typically, a child activity)
  2. Over a session with a service

Capability Exchange

- Kernel provides syscalls to create, exchange, and revoke caps
- There are two ways to exchange caps:
  1. Directly with another activity (typically, a child activity)
  2. Over a session with a service
- The kernel offers two operations:
  1. **Delegate**: send capability to somebody else
  2. **Obtain**: receive capability from somebody else

Capability Exchange

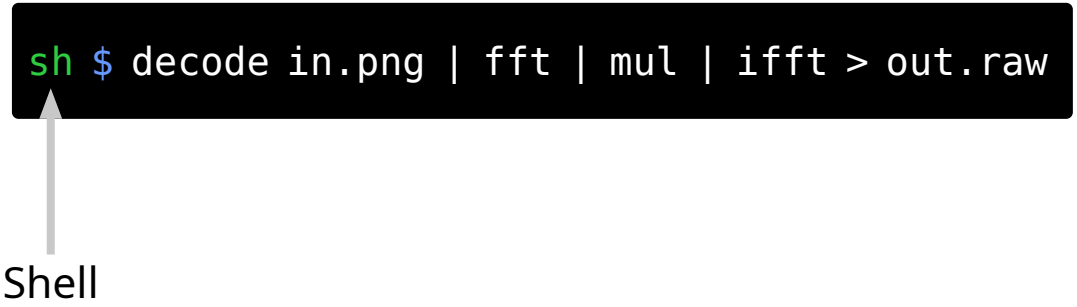
- Kernel provides syscalls to create, exchange, and revoke caps
- There are two ways to exchange caps:
  1. Directly with another activity (typically, a child activity)
  2. Over a session with a service
- The kernel offers two operations:
  1. **Delegate**: send capability to somebody else
  2. **Obtain**: receive capability from somebody else
- Difference to L4:
  - Applications communicate directly, without involving the kernel
    $\rightarrow$ Capability exchange cannot be done during IPC
  - Special communication channel between kernel and servers
  - Kernel uses this channel to send exchange requests to server

- Introduction
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- Operating System
- **OS Services and Accelerators**
- Context Switching
- Trusted Execution Environments
- Evaluation

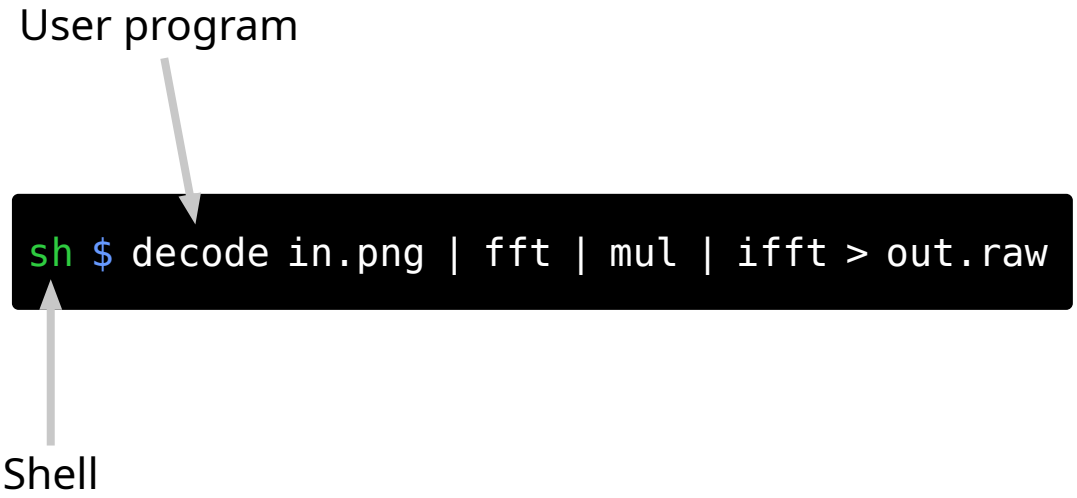
OS-Service Access for all Compute Units

```
sh $ decode in.png | fft | mul | ifft > out.raw
```

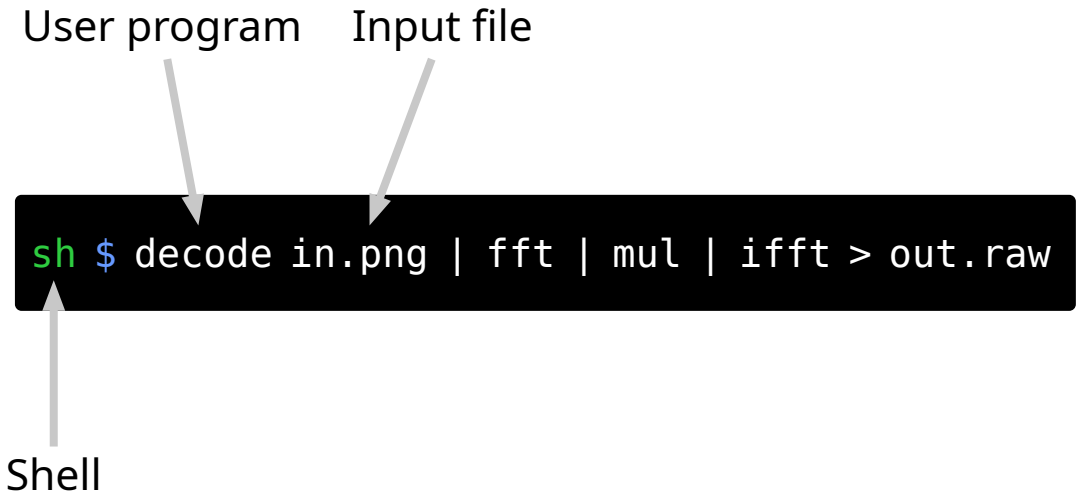
OS-Service Access for all Compute Units



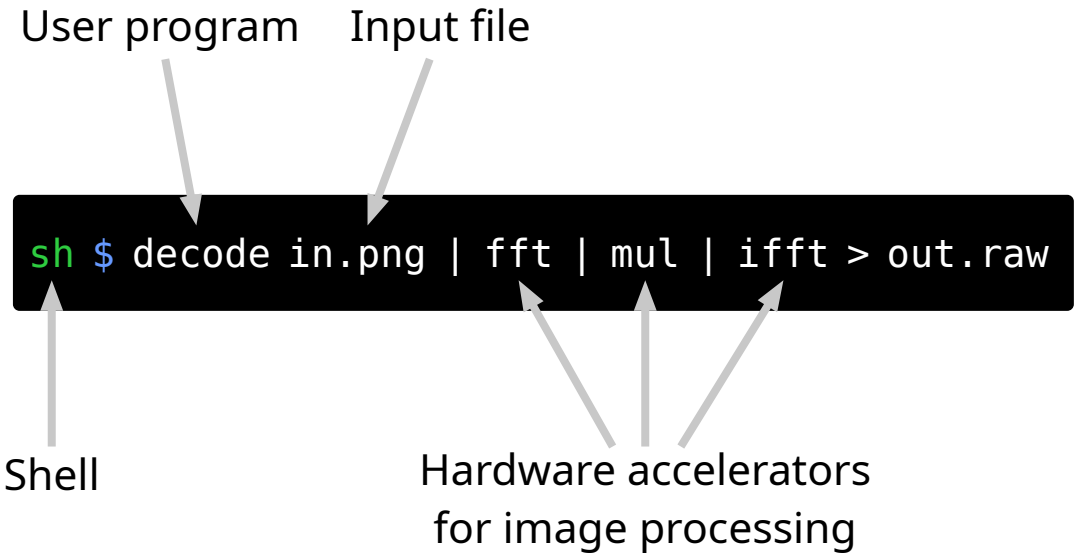
OS-Service Access for all Compute Units



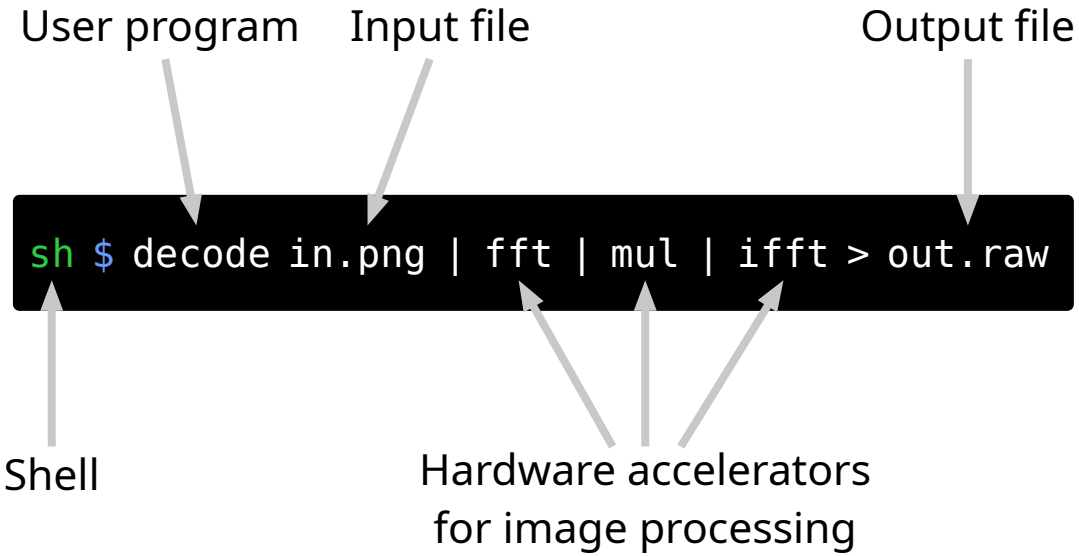
OS-Service Access for all Compute Units



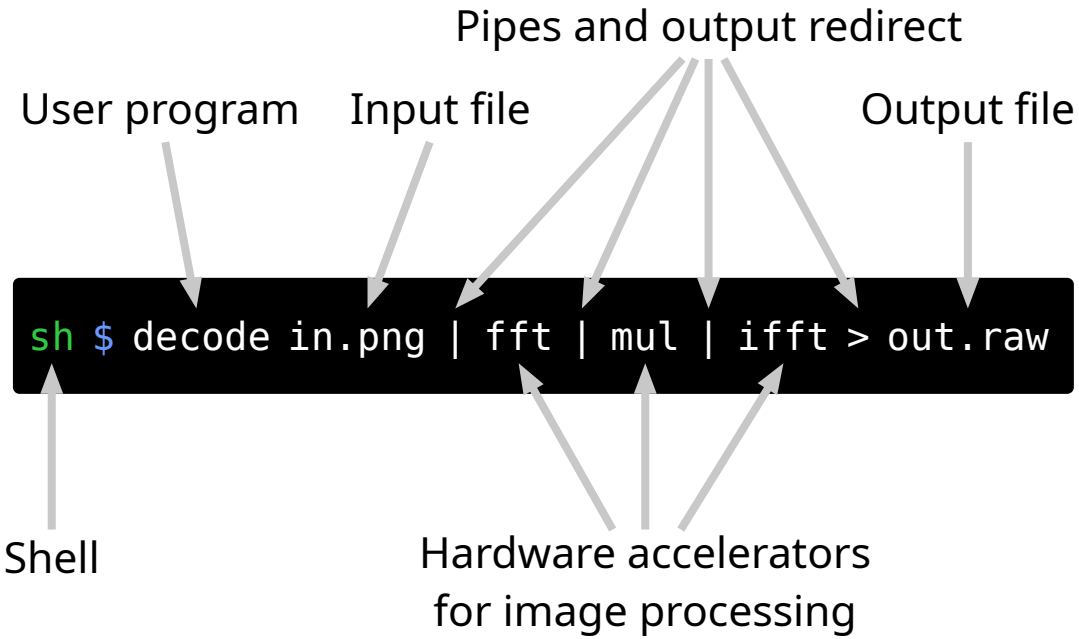
OS-Service Access for all Compute Units



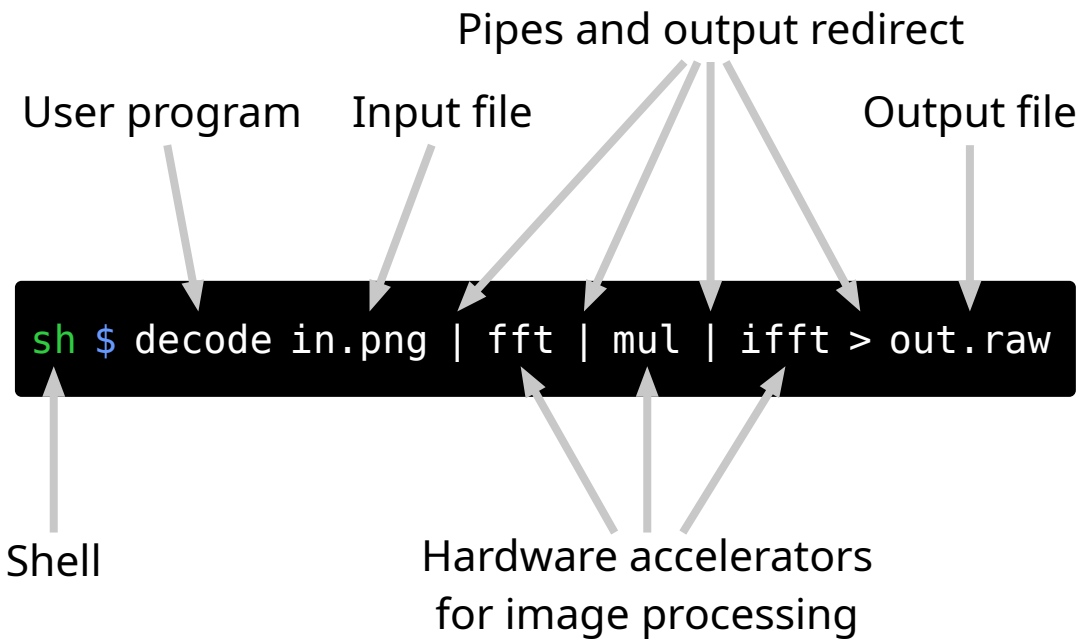
OS-Service Access for all Compute Units



OS-Service Access for all Compute Units

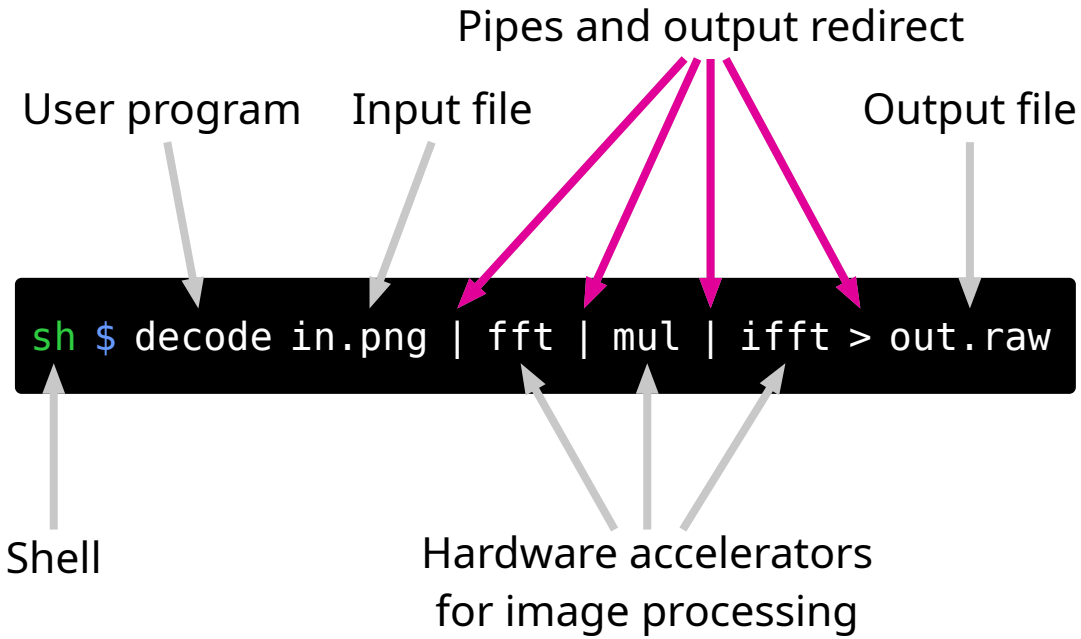


OS-Service Access for all Compute Units



Challenges:

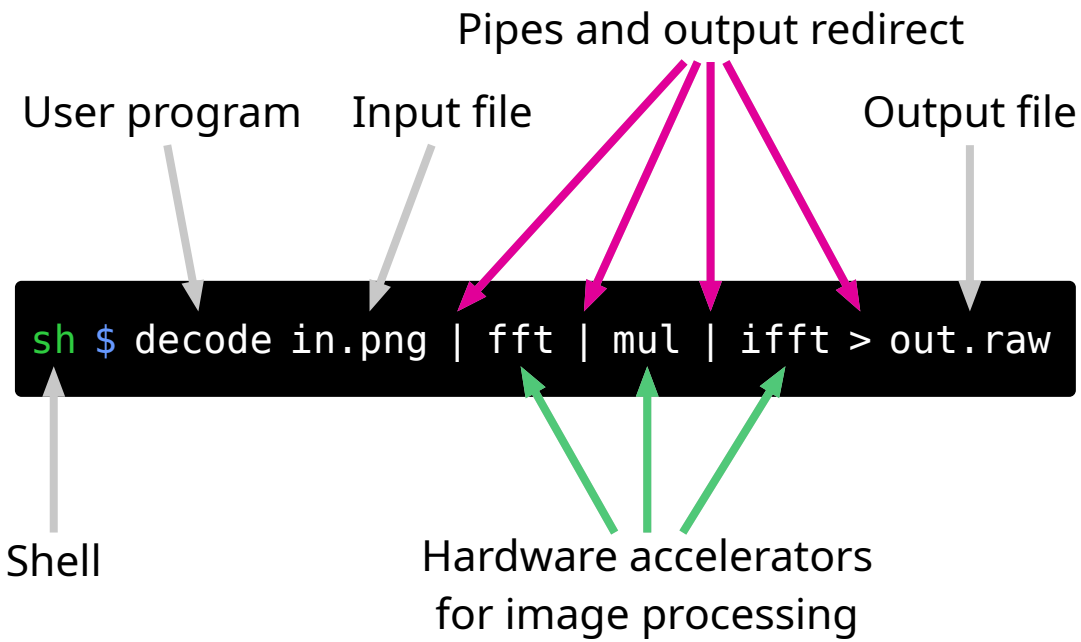
OS-Service Access for all Compute Units



Challenges:

- OS must provide **generic protocols**

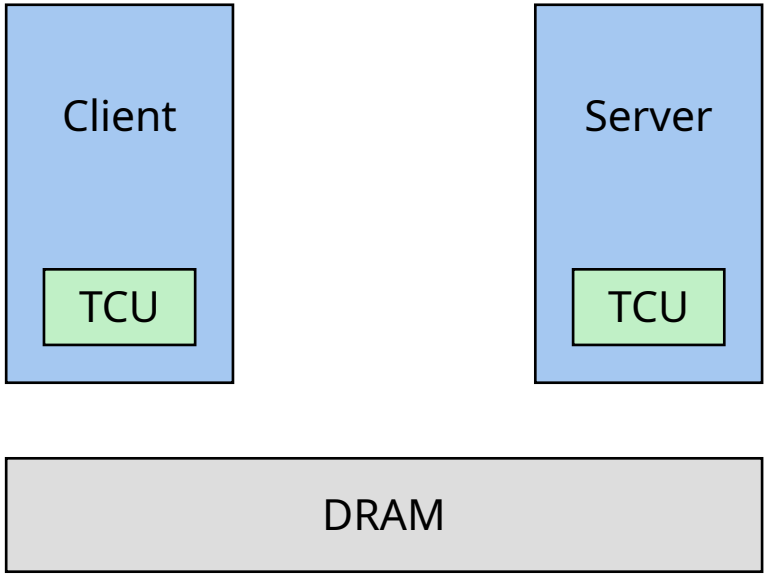
OS-Service Access for all Compute Units



Challenges:

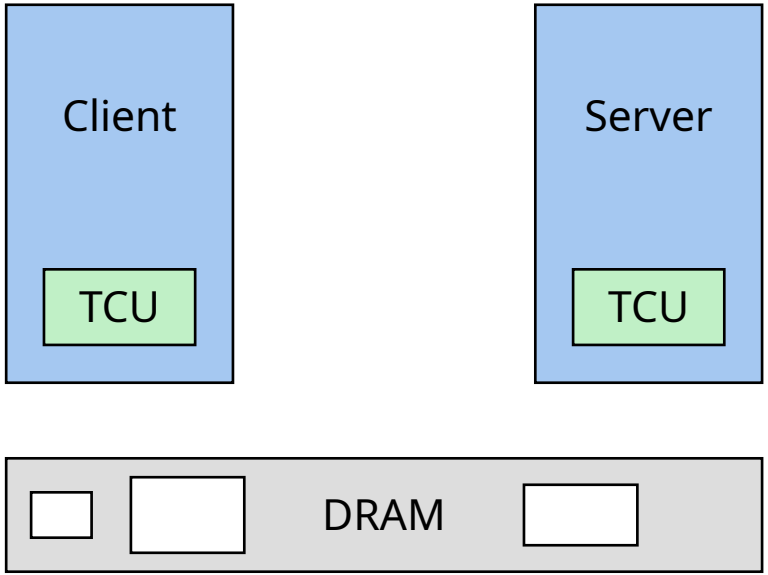
- OS must provide **generic protocols**
- **Accelerators** need support for protocols

Generic Protocols



File Protocol:

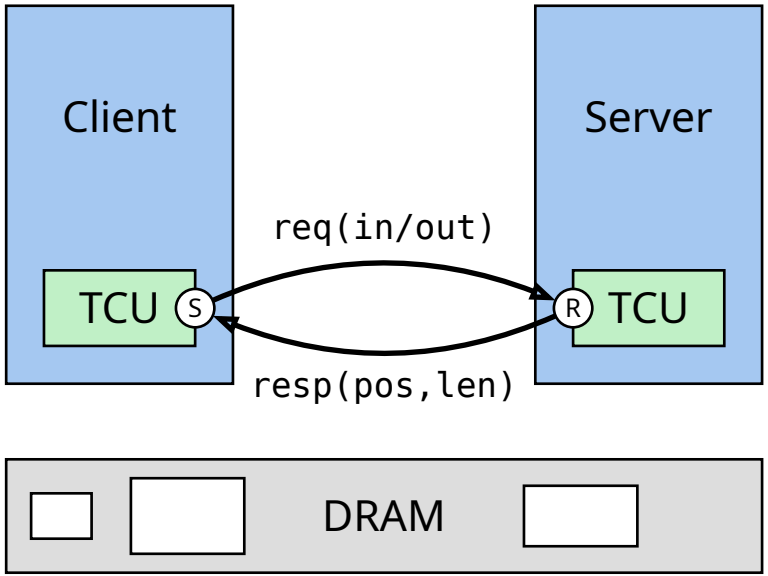
Generic Protocols



File Protocol:

- Data in memory

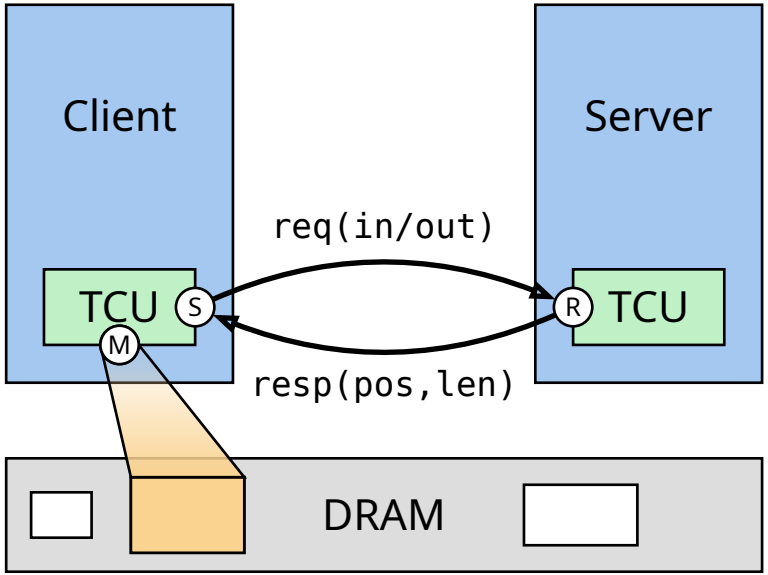
Generic Protocols



File Protocol:

- Data in memory
- RPC between client and server
  - `req(in/out)` requests next piece, implicitly commits previous piece
  - `commit(nbytes)` commits nbytes of previous piece

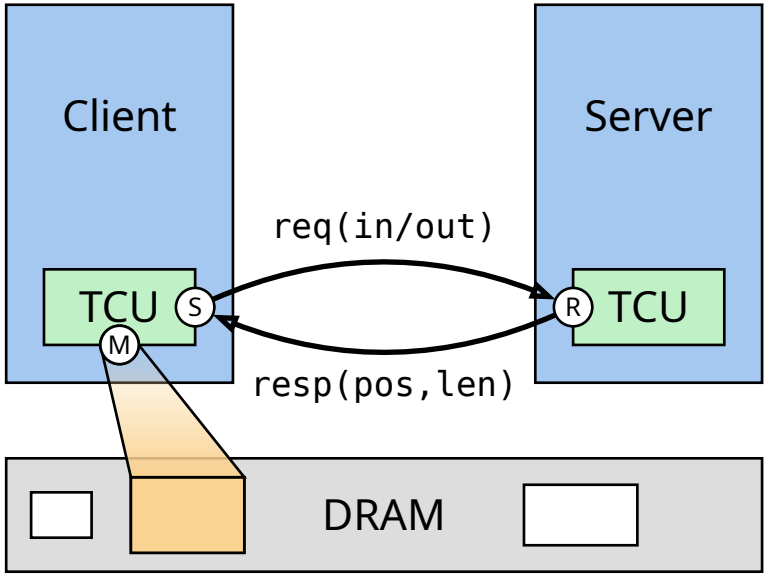
Generic Protocols



File Protocol:

- Data in memory
- RPC between client and server
  - `req(in/out)` requests next piece, implicitly commits previous piece
  - `commit(nbytes)` commits nbytes of previous piece
- Server configures client's memory EP

Generic Protocols



File Protocol:

- Data in memory
- RPC between client and server
  - `req(in/out)` requests next piece, implicitly commits previous piece
  - `commit(nbytes)` commits nbytes of previous piece
- Server configures client's memory EP
- Client accesses data via TCU

Implementation: M³FS – Overview

- M³FS organizes the file’s data in extents

Implementation: M³FS – Overview

- M³FS organizes the file's data in extents
- M³FS can be used with a memory and disk backend
  - With memory backend, FS image is a contiguous region in DRAM
  - Clients get access to parts of the image
  - With disk backend, M³FS uses a buffer cache in DRAM
  - Clients get access to parts of buffer cache

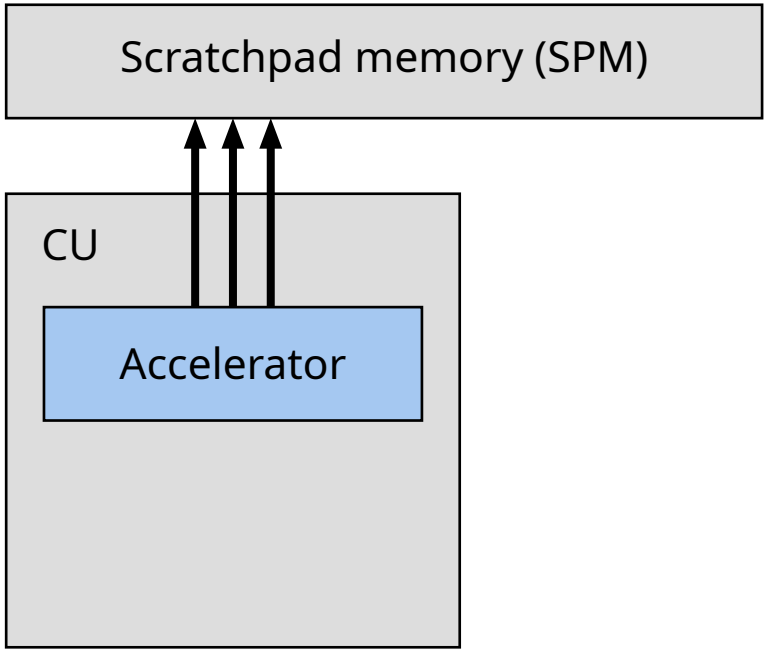
Implementation: M³FS – Overview

- M³FS organizes the file's data in extents
- M³FS can be used with a memory and disk backend
  - With memory backend, FS image is a contiguous region in DRAM
  - Clients get access to parts of the image
  - With disk backend, M³FS uses a buffer cache in DRAM
  - Clients get access to parts of buffer cache
- Two types of sessions: *metadata* session, *file* session
- Metadata session is created first, allows stat, open, . . .
- `open` creates a new file session
- Both sessions can be cloned to provide other activities access

Implementation: M³FS – File Protocol

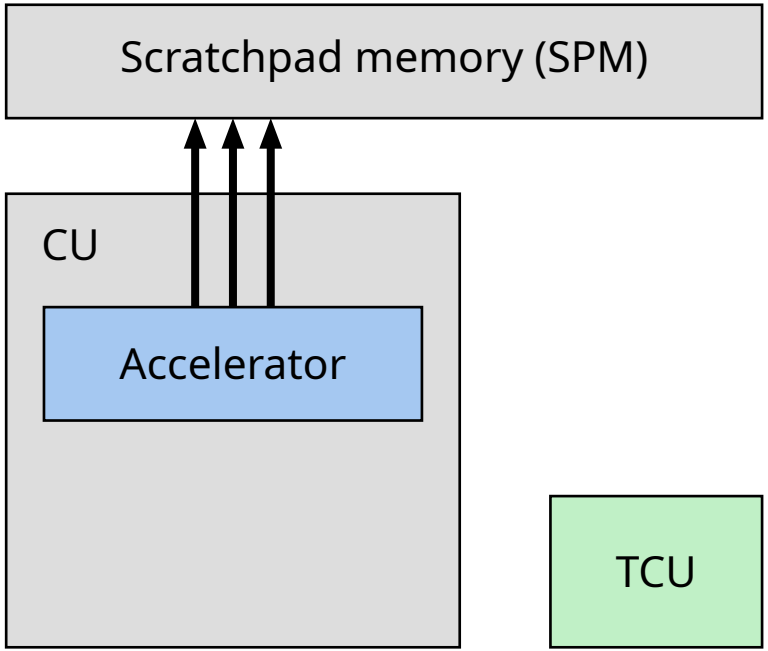
- The file session implements the file protocol (plus seeking)
- File session holds file position and advances it on read/write
- `req(in/out)` request next extent
- M³FS configures client's EP for this extent
- Appending reserves new space, invisible to other clients
- `commit(nbytes)` commits a previous append

Additions to Accelerator



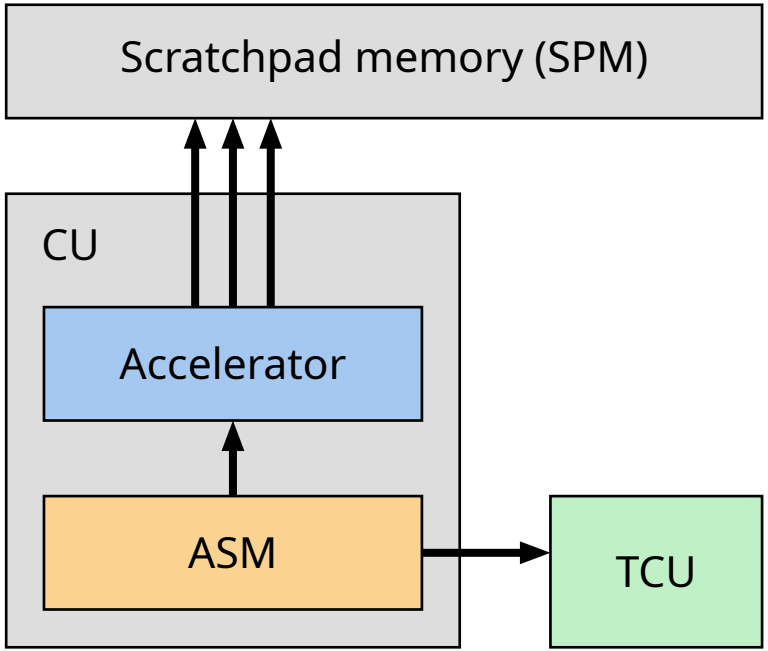
- Off-the-shelf accelerators

Additions to Accelerator



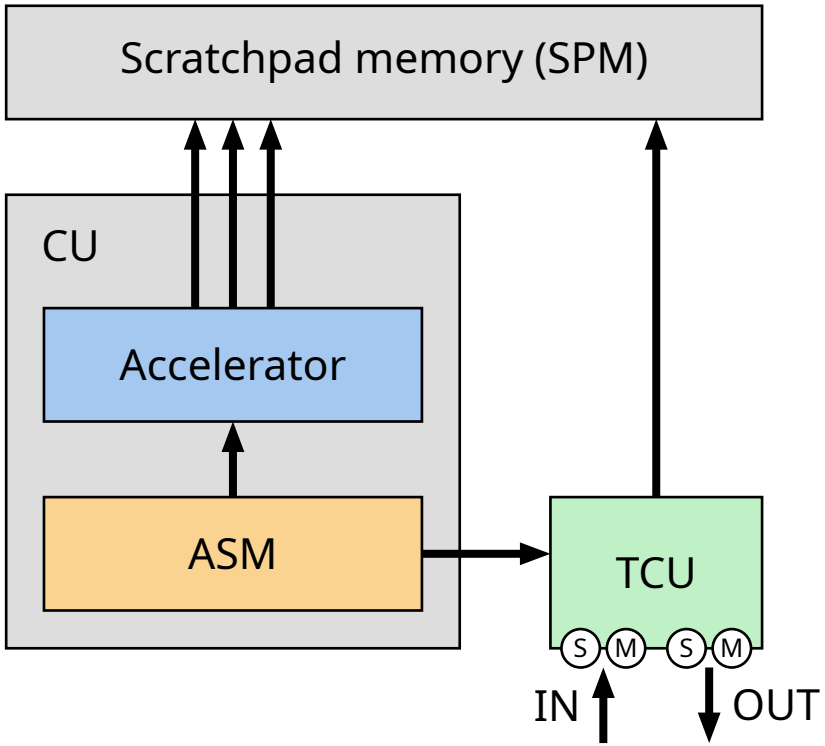
- Off-the-shelf accelerators

Additions to Accelerator



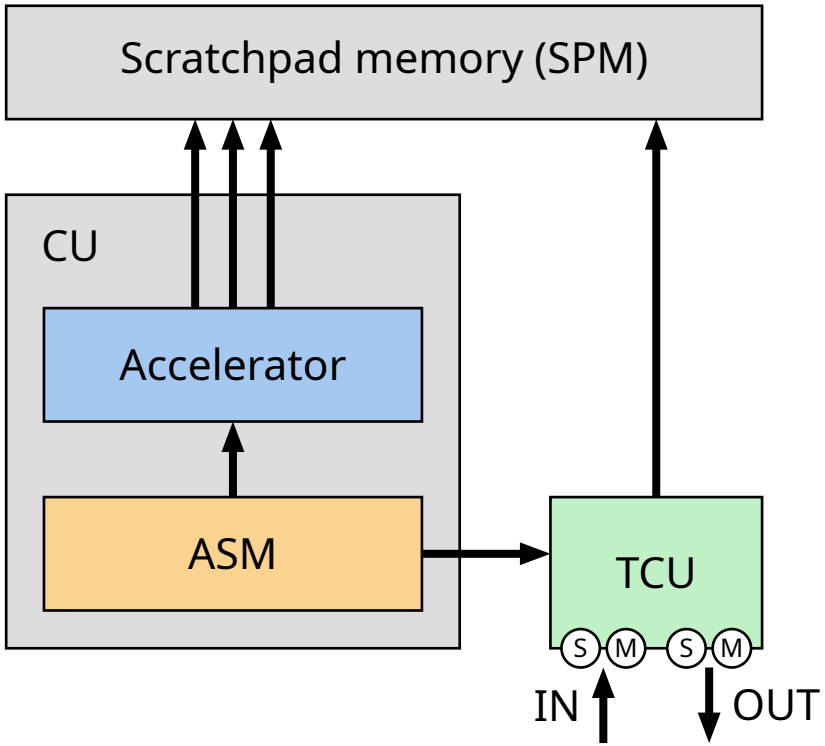
- Off-the-shelf accelerators
- Accelerator Support Module (ASM):
  - Interacts with TCU and accelerator

Additions to Accelerator



- Off-the-shelf accelerators
- Accelerator Support Module (ASM):
  - Interacts with TCU and accelerator
  - Implements file protocol for input and output channel

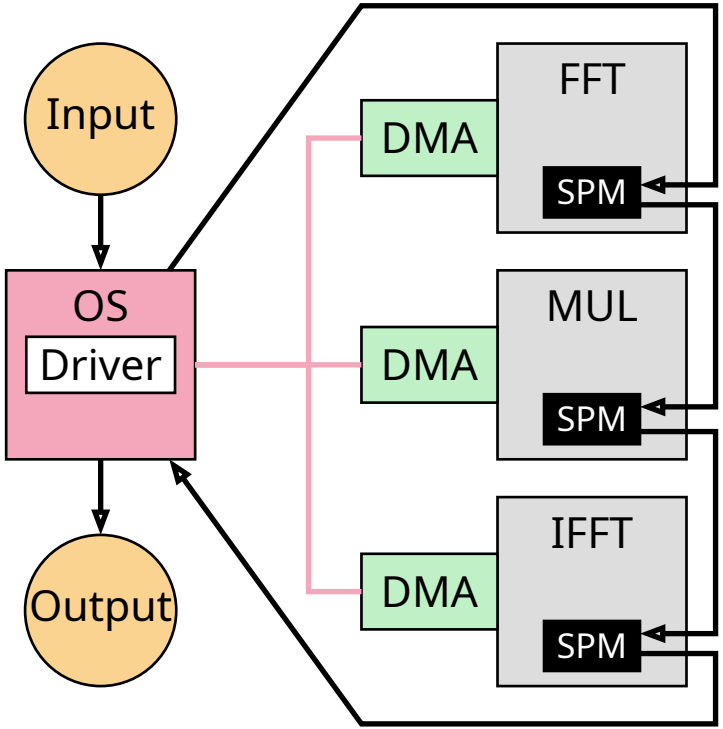
Additions to Accelerator



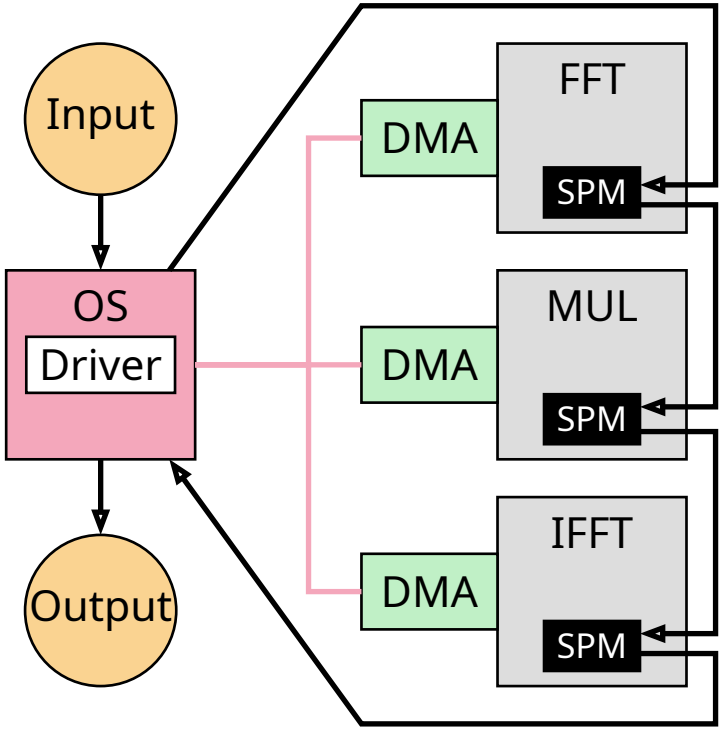
- Off-the-shelf accelerators
- Accelerator Support Module (ASM):
  - Interacts with TCU and accelerator
  - Implements file protocol for input and output channel
  - ASM assumes that endpoints are setup externally by software

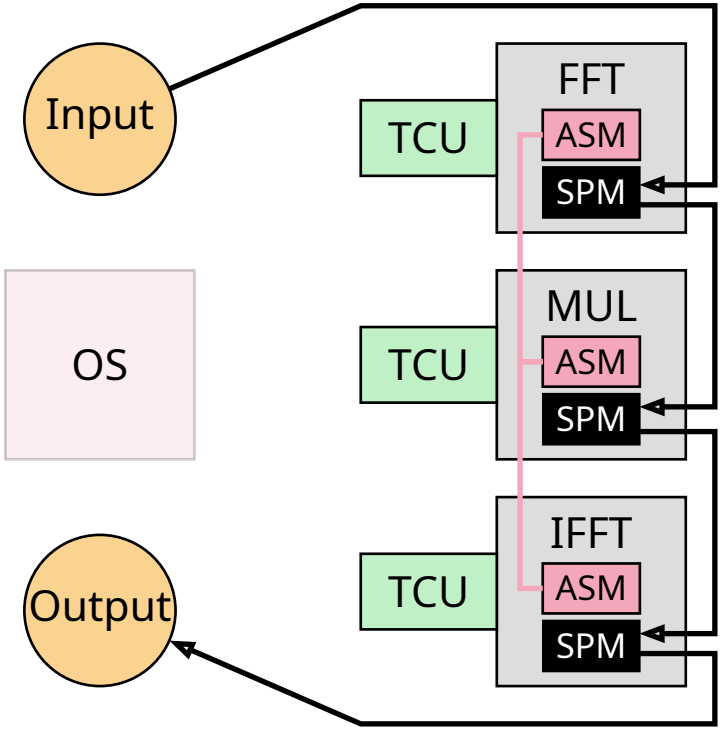
Demo

Assisted vs. Autonomous

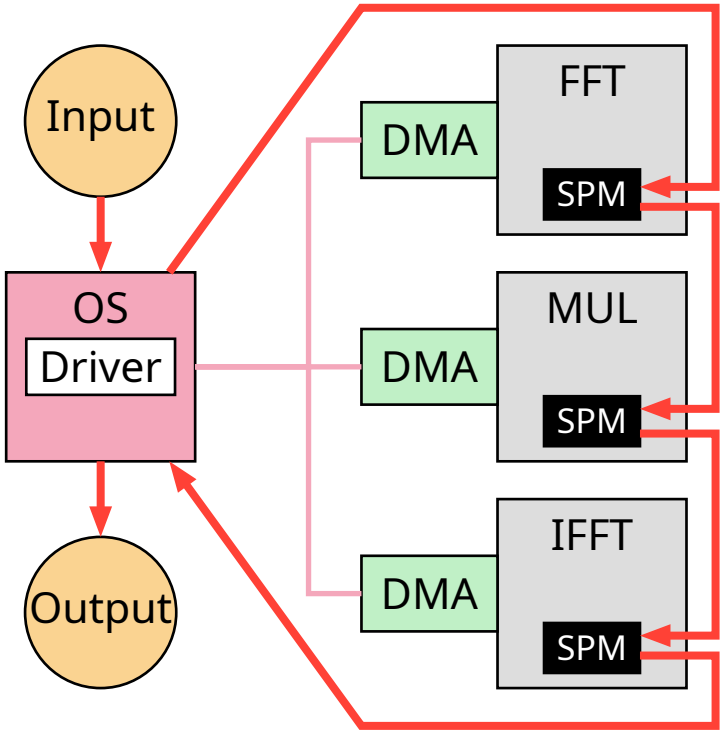


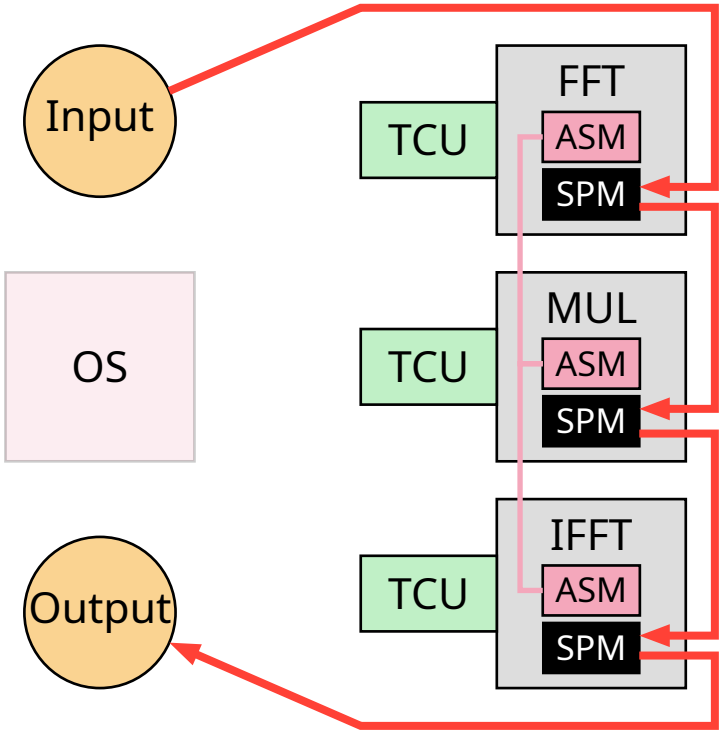
Assisted vs. Autonomous



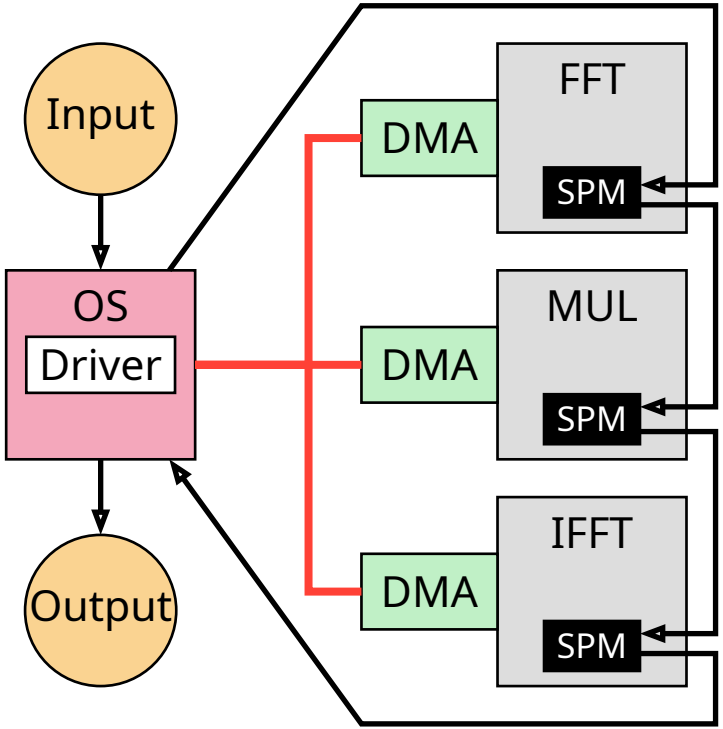


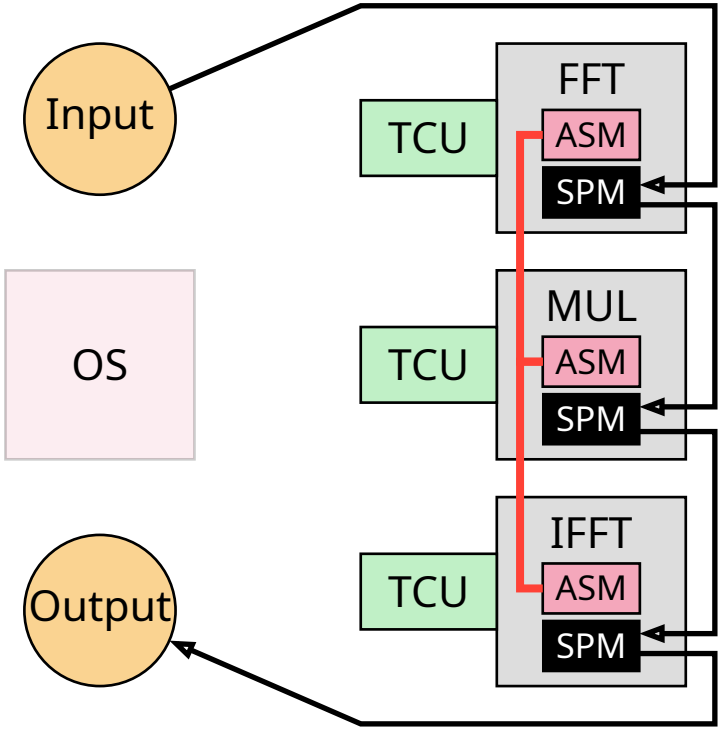
Assisted vs. Autonomous





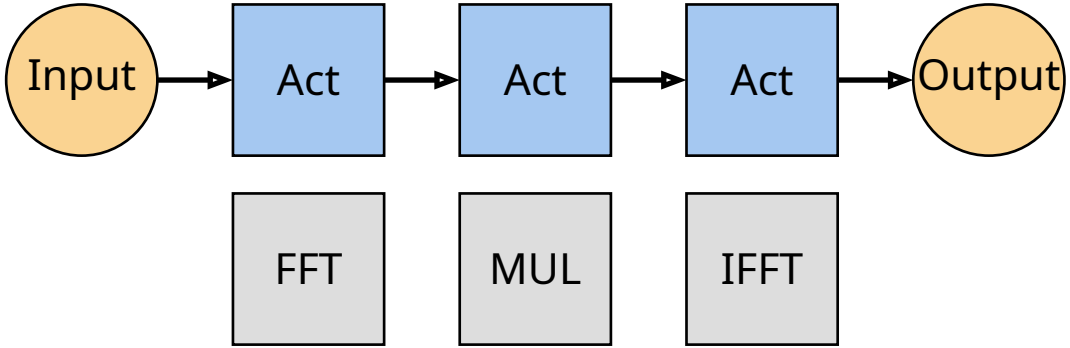
Assisted vs. Autonomous



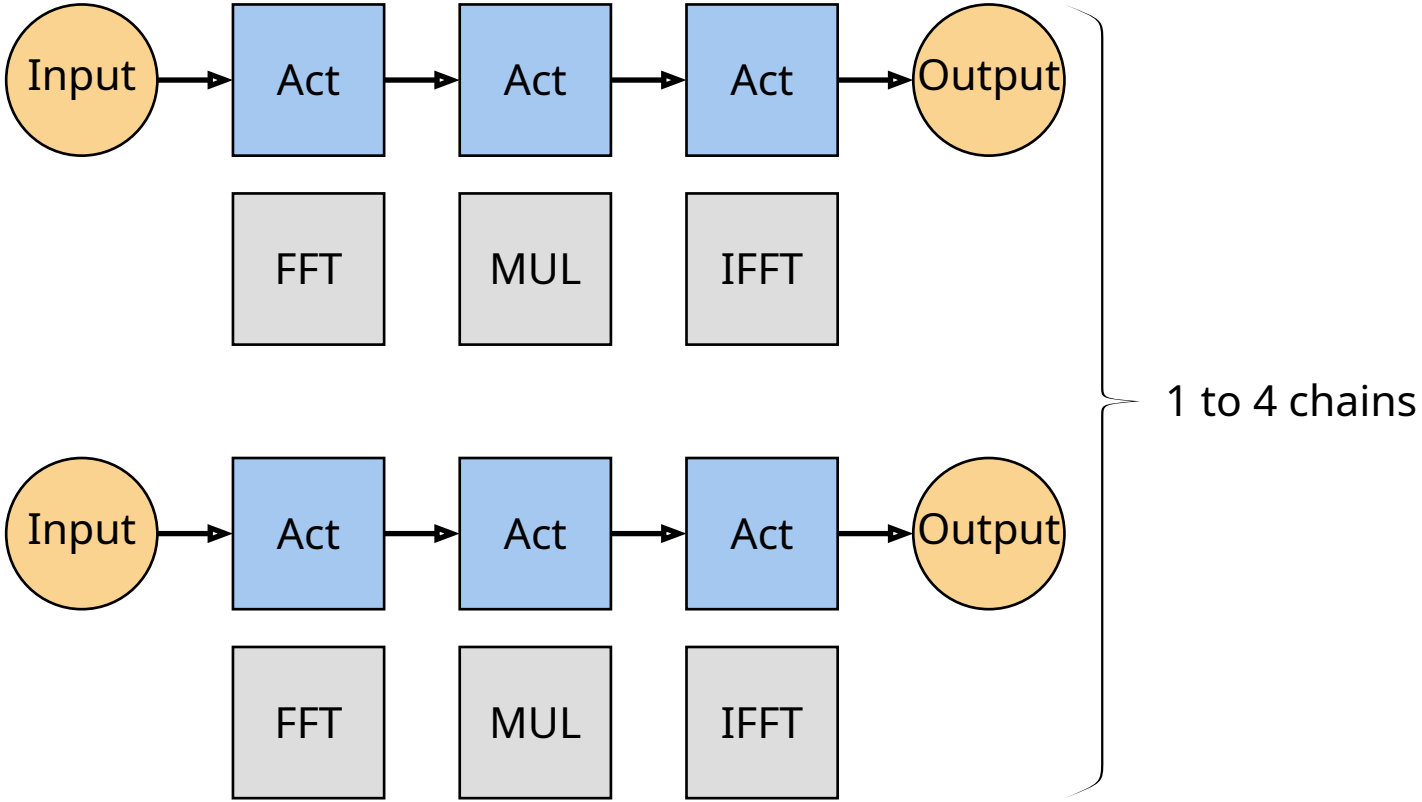


Accelerator Chains

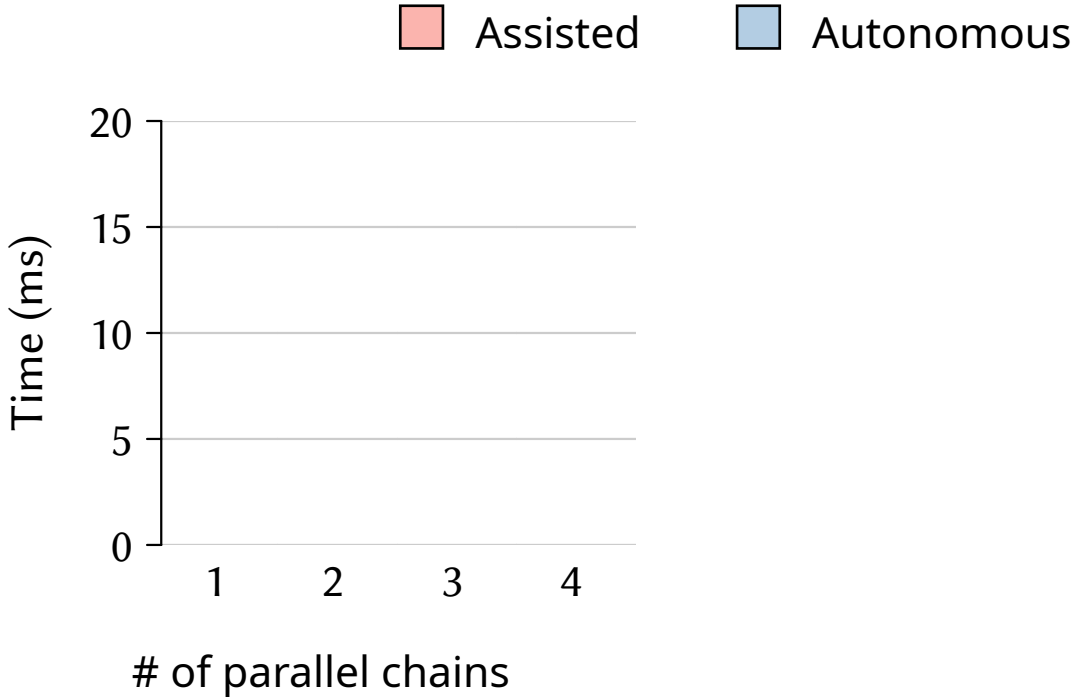
Accelerator Chains



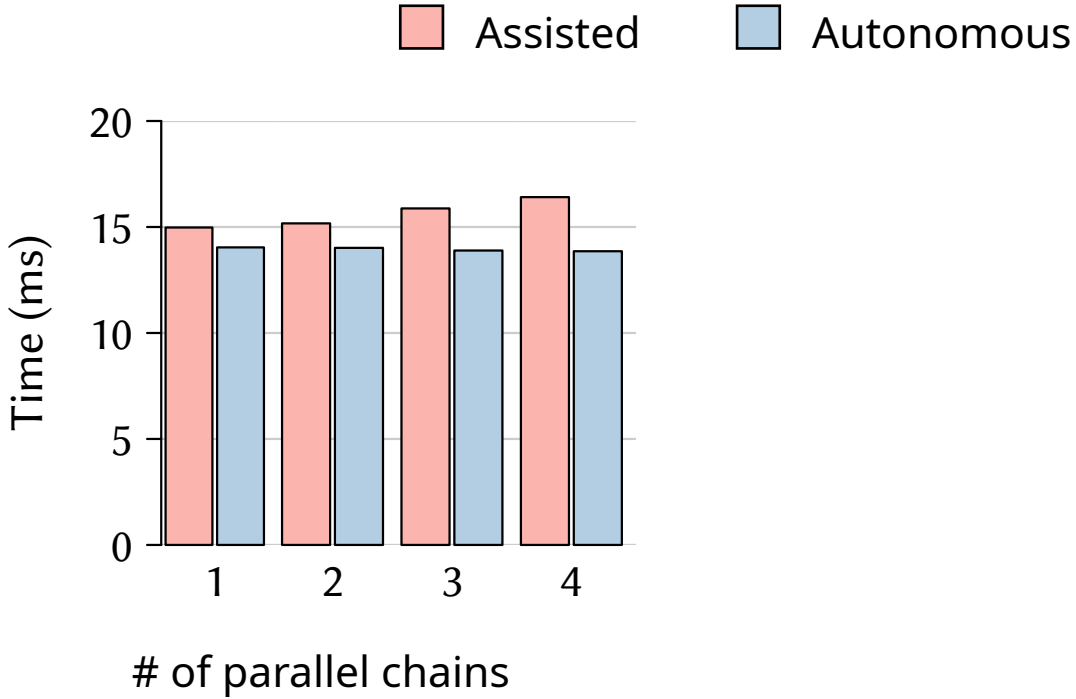
Accelerator Chains



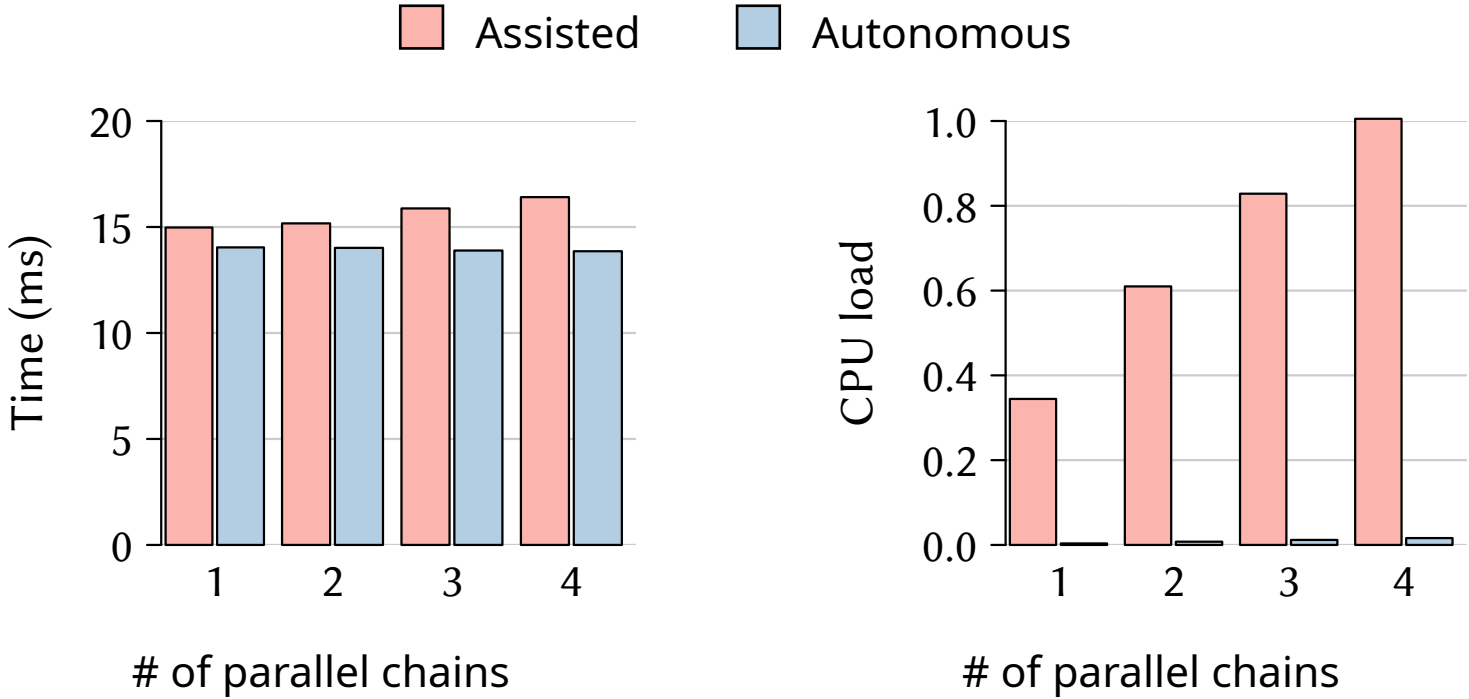
Accelerator Chains: Results



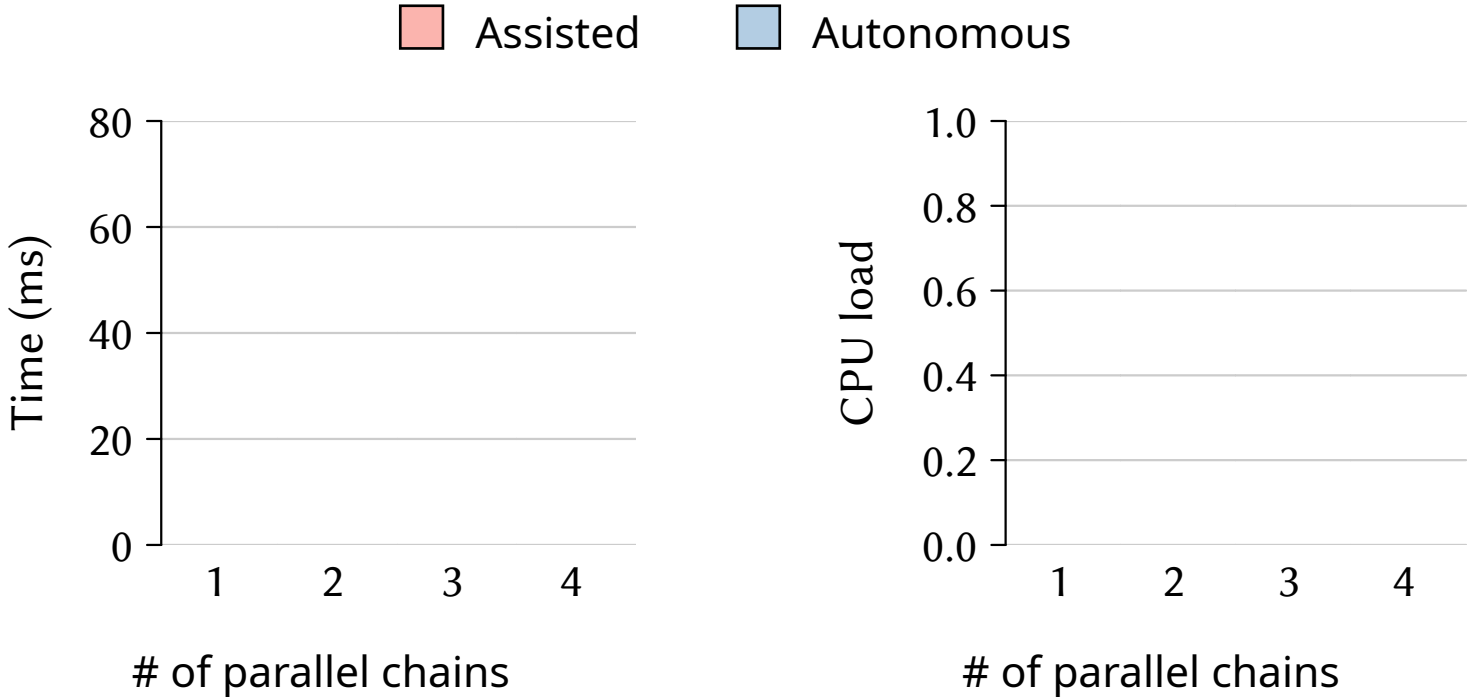
Accelerator Chains: Results



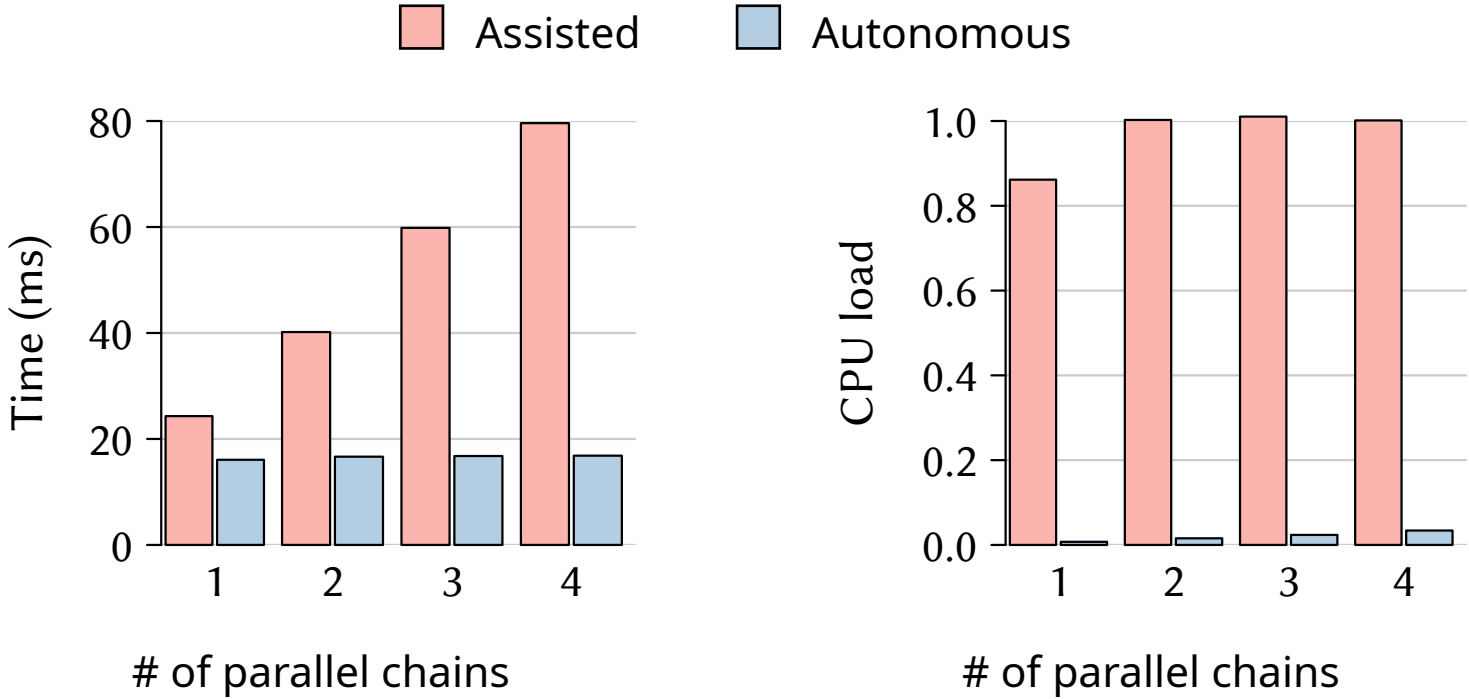
Accelerator Chains: Results



Accelerator Chains: Results (PCIe-like Latency)



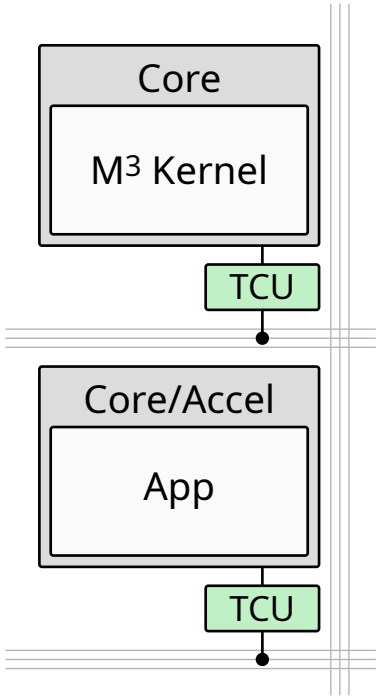
Accelerator Chains: Results (PCIe-like Latency)



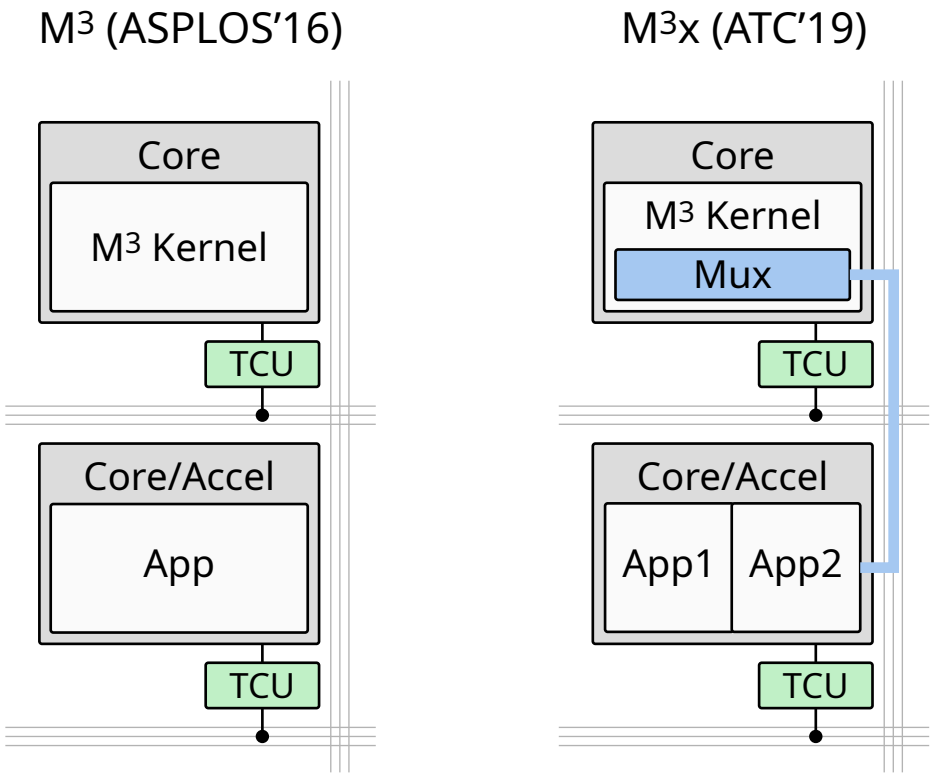
- Introduction
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- **Context Switching**
- Trusted Execution Environments
- Evaluation

Comparison of Context-Switching Approaches

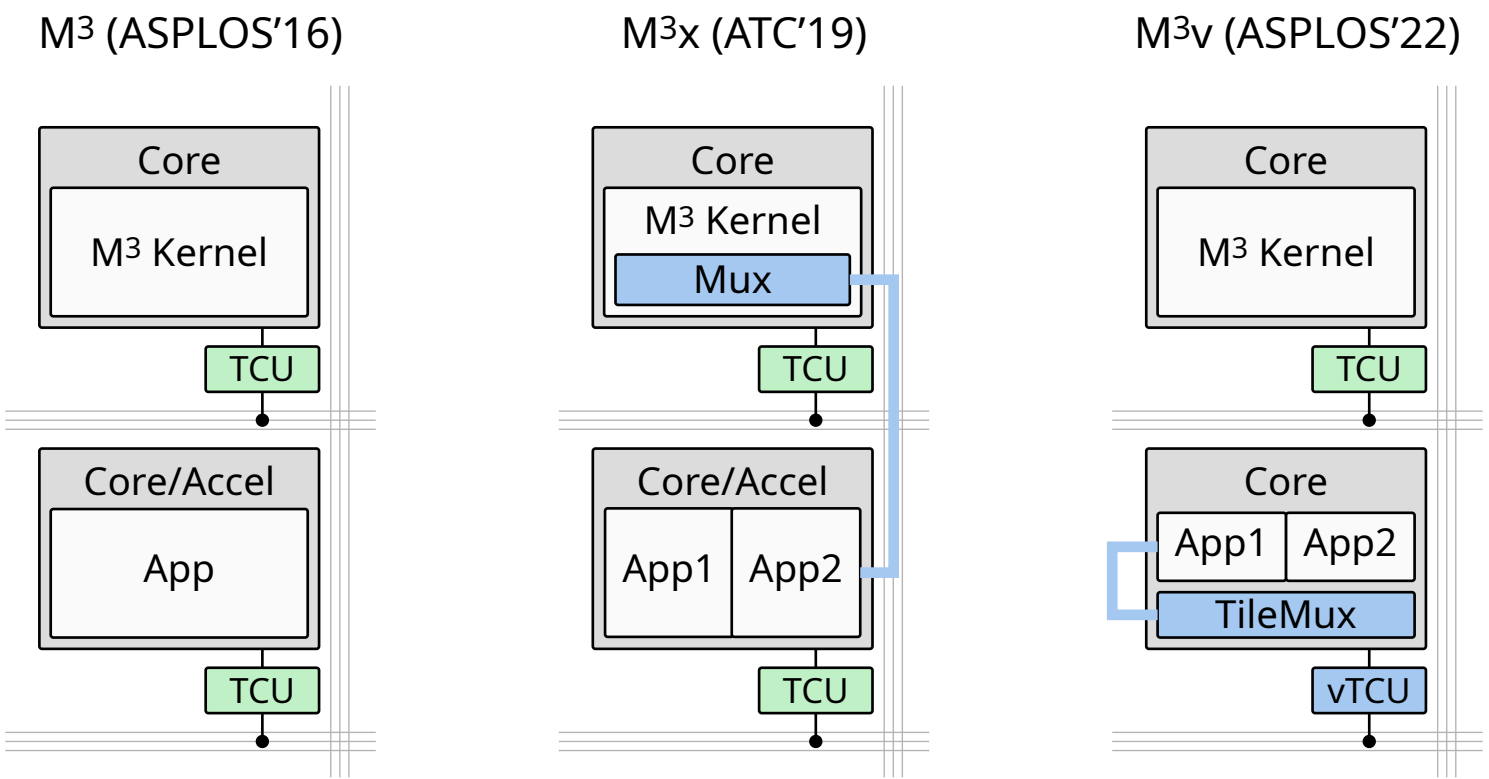
M³ (ASPLOS'16)



Comparison of Context-Switching Approaches



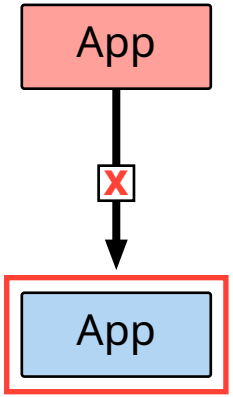
Comparison of Context-Switching Approaches



- Introduction
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- Context Switching
- **Trusted Execution Environments**
- Evaluation

What is a Trusted Execution Environment?

What is a Trusted Execution Environment?



Properties:

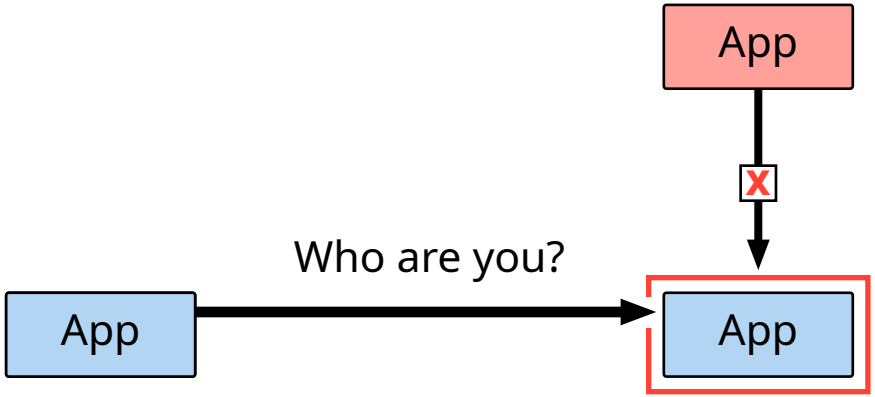
- Isolated environment

What is a Trusted Execution Environment?

Properties:

- Isolated environment

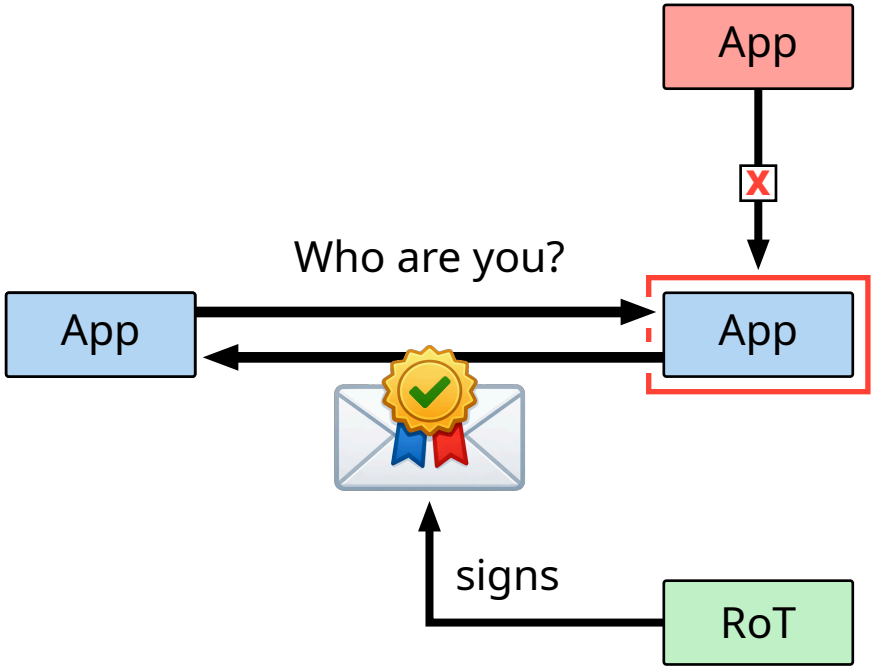
What is a Trusted Execution Environment?



Properties:

- Isolated environment
- Externally verifiable

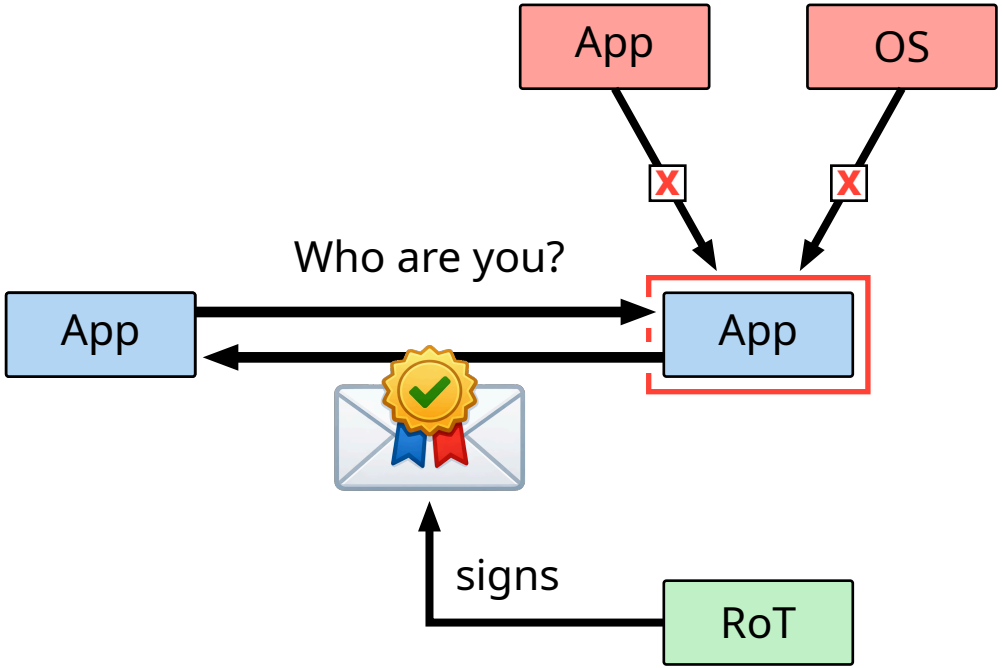
What is a Trusted Execution Environment?



Properties:

- Isolated environment
- Externally verifiable

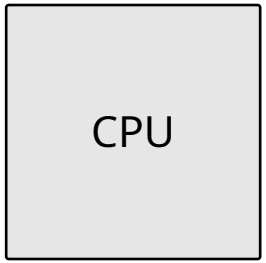
What is a Trusted Execution Environment?



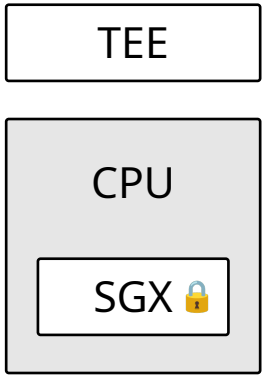
Properties:

- Isolated environment
- Externally verifiable
- Small *trusted computing base*

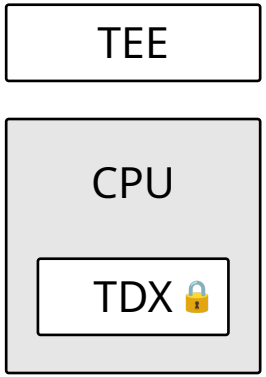
Existing Trusted Execution Environments



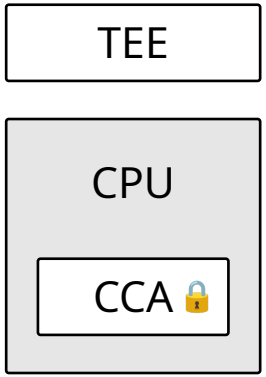
Existing Trusted Execution Environments



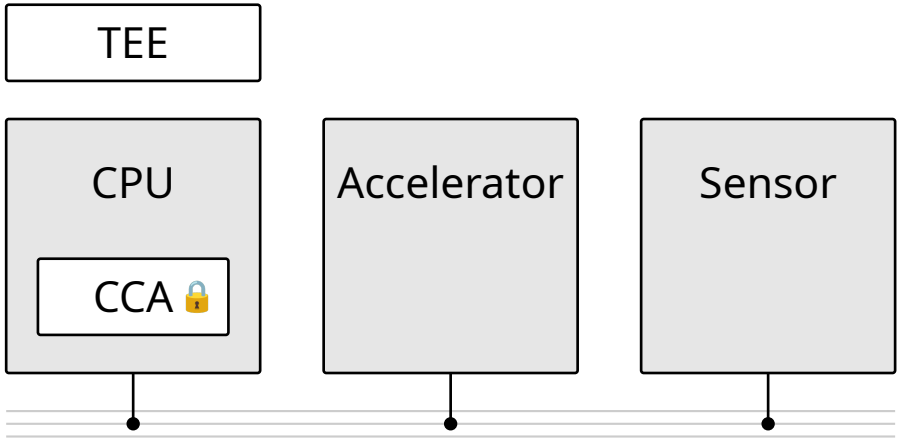
Existing Trusted Execution Environments



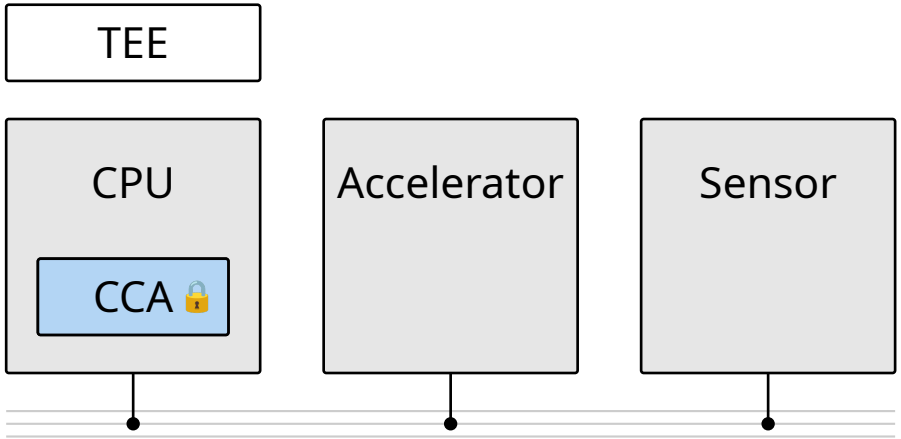
Existing Trusted Execution Environments



Existing Trusted Execution Environments



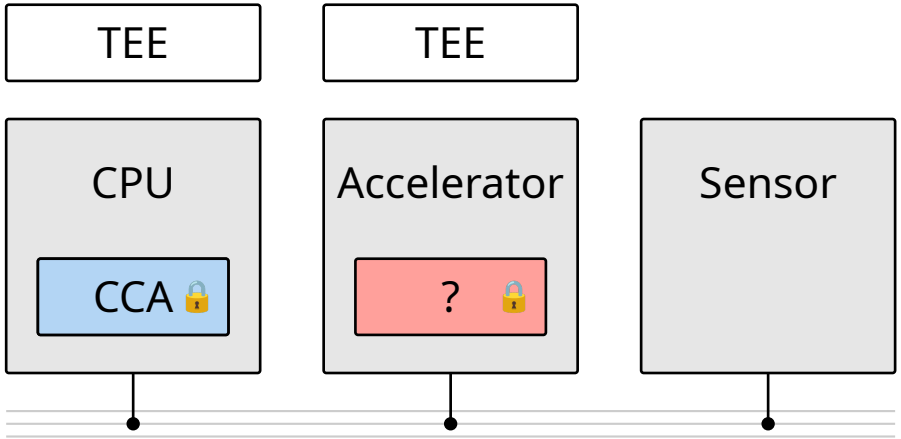
Existing Trusted Execution Environments



Problems:

1. **CPU centric**

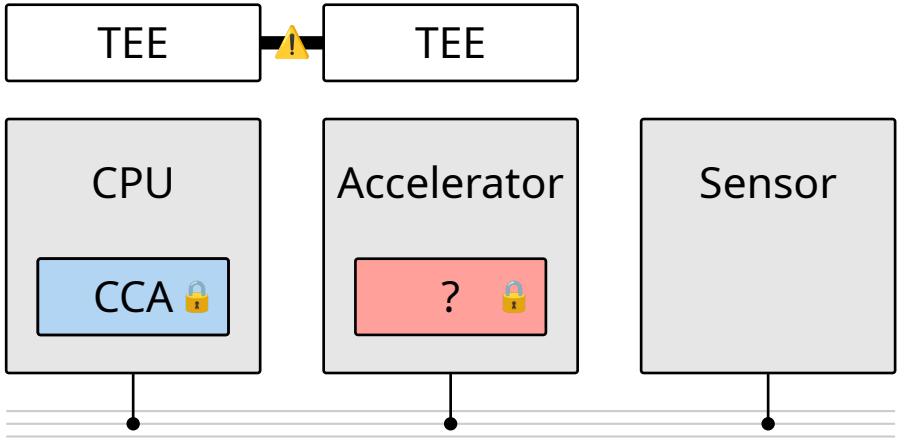
Existing Trusted Execution Environments



Problems:

1. CPU centric
2. **Heterogeneous TEEs inflate TCB**

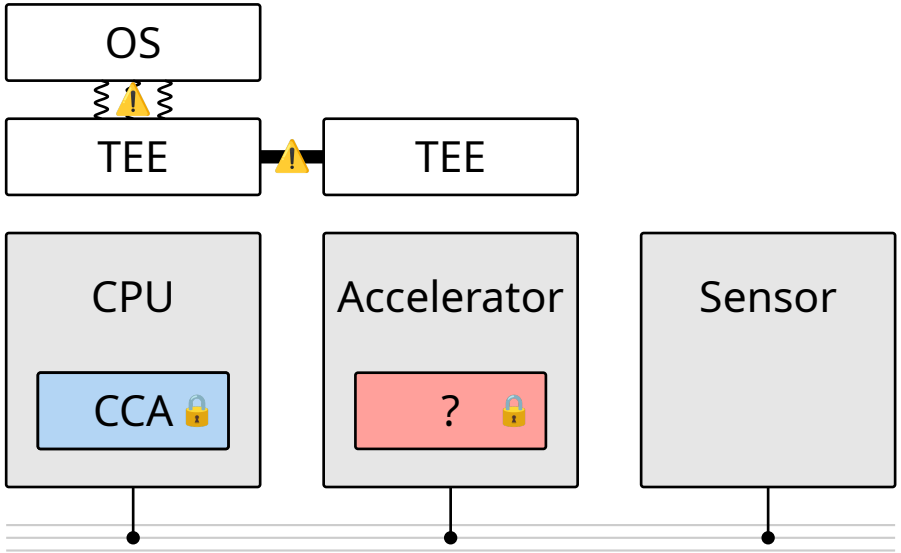
Existing Trusted Execution Environments



Problems:

1. CPU centric
2. **Heterogeneous TEEs inflate TCB**

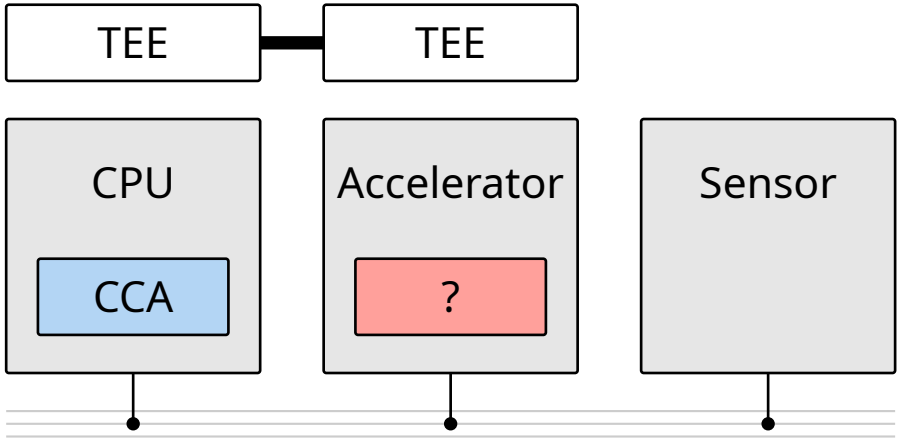
Existing Trusted Execution Environments



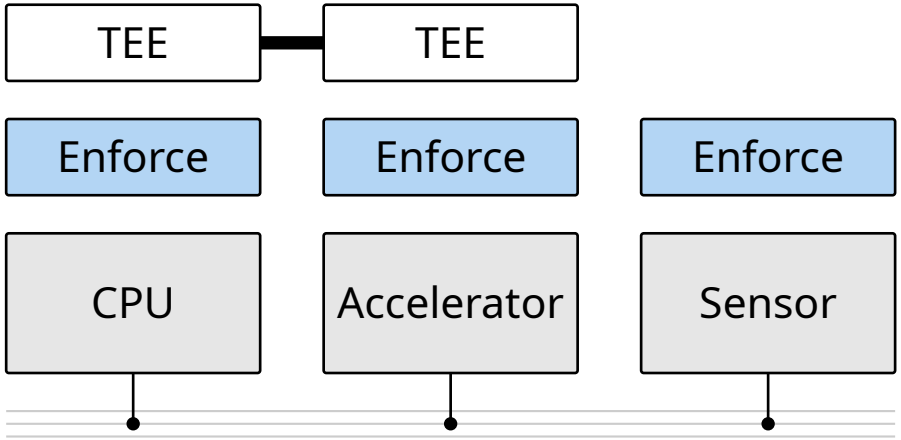
Problems:

1. CPU centric
2. Heterogeneous TEEs inflate TCB
3. **Inherently prone to side-channel issues**

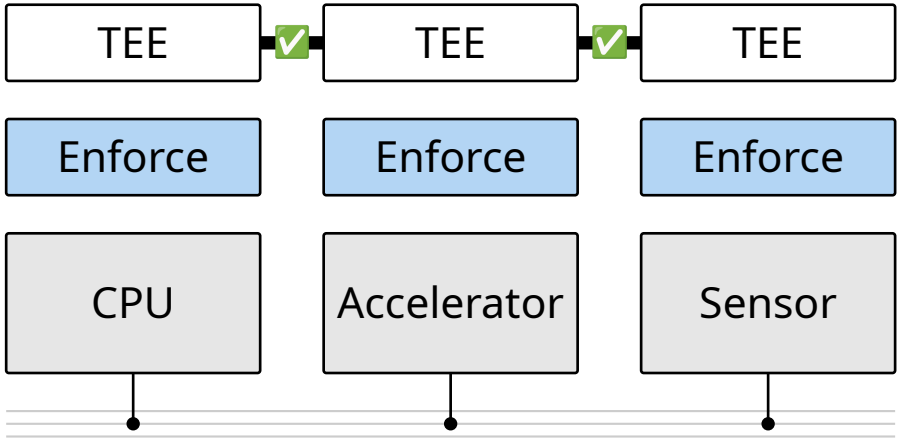
Core-Independent TEEs



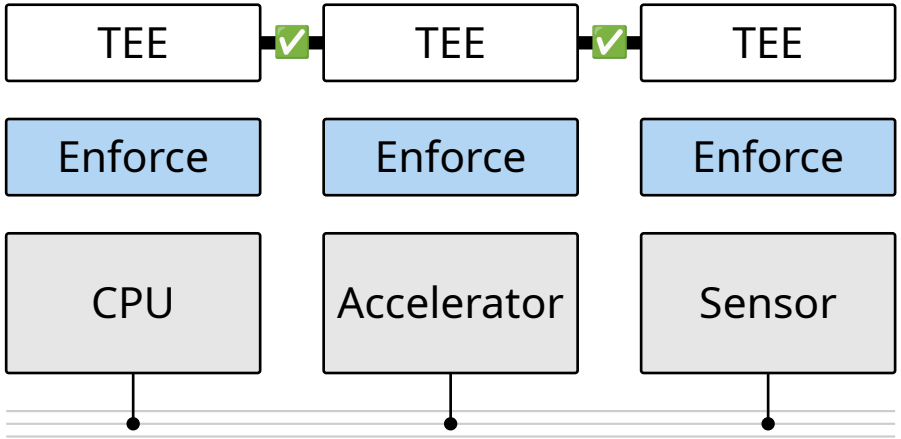
Core-Independent TEEs



Core-Independent TEEs



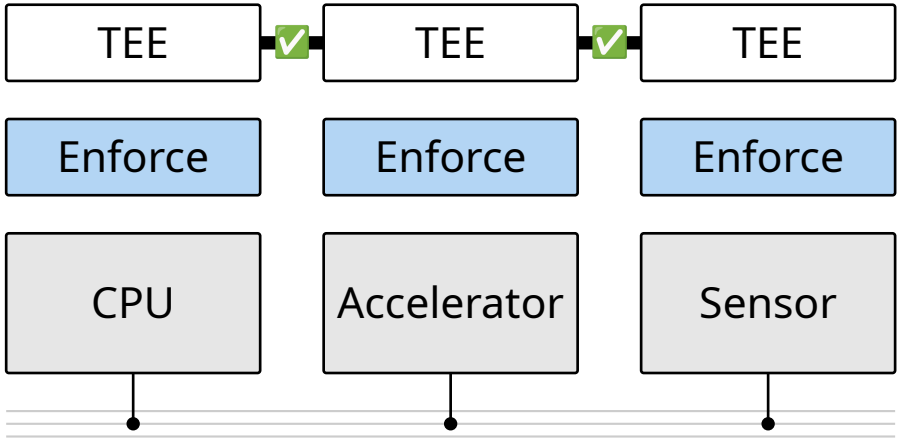
Core-Independent TEEs



Minimal Ingredients:

1. **Exclusive memory**

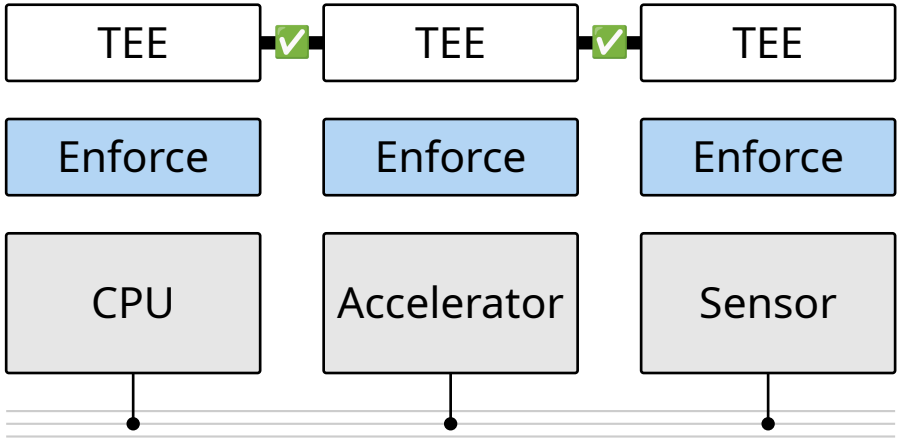
Core-Independent TEEs



Minimal Ingredients:

1. Exclusive memory
2. **Exclusive channels**

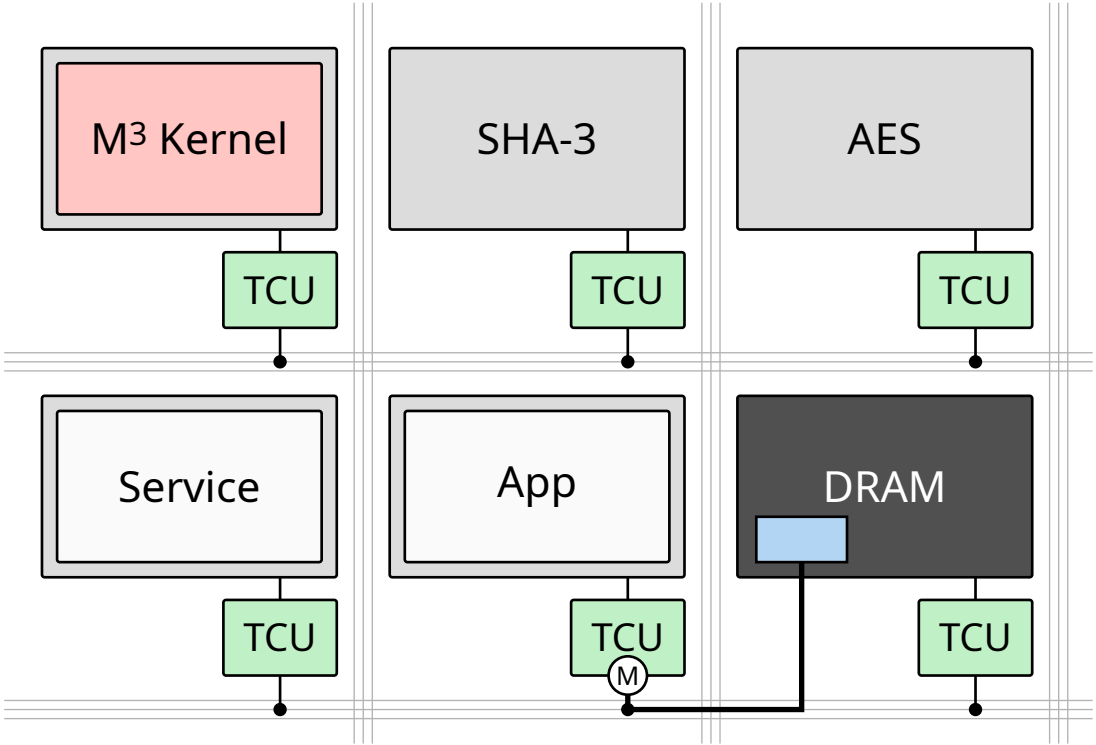
Core-Independent TEEs



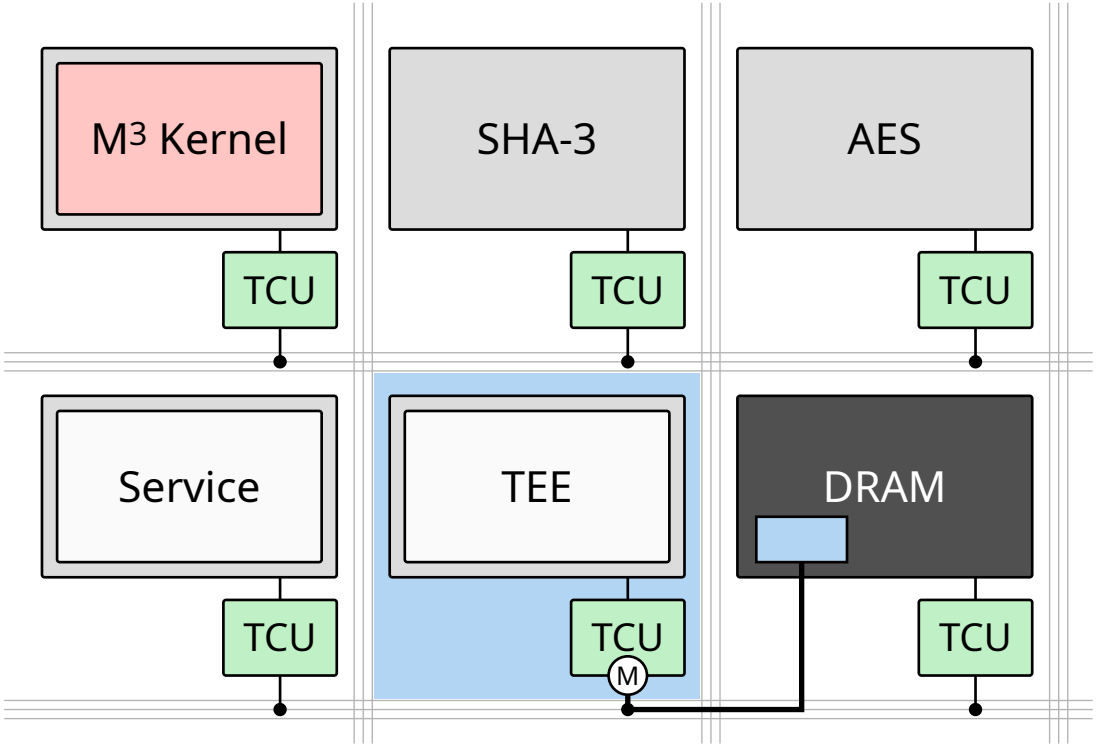
Minimal Ingredients:

1. Exclusive memory
2. Exclusive channels
3. **Remote OS services**

Tile-Based TEEs



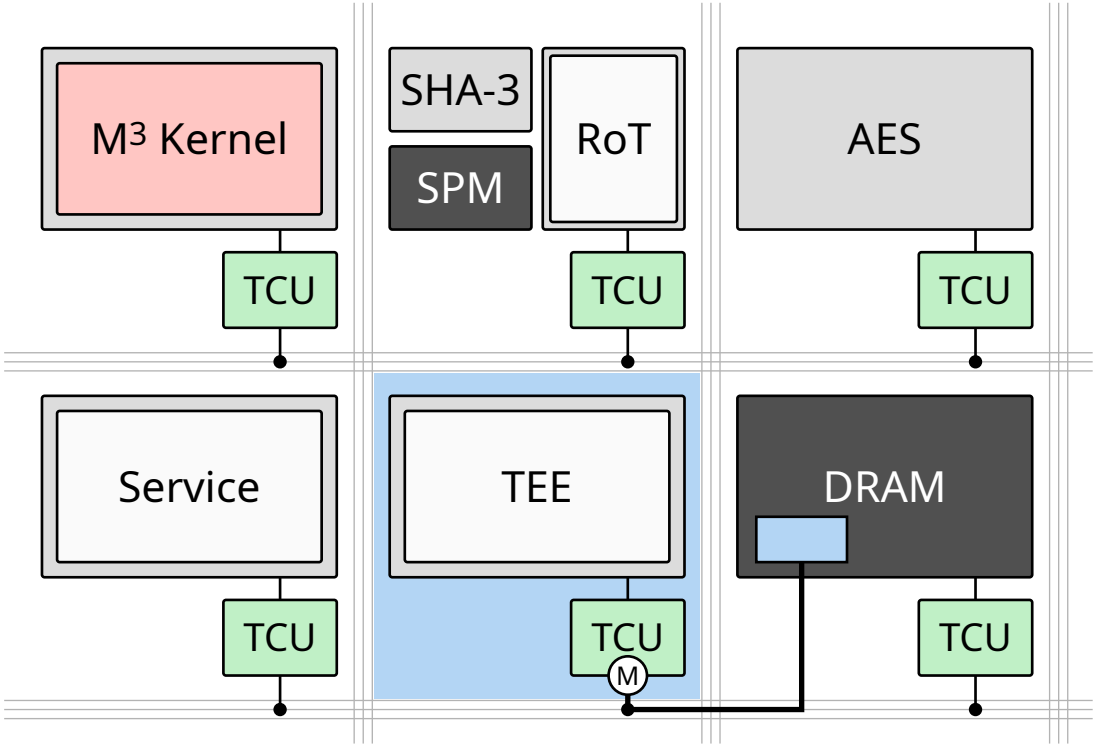
Tile-Based TEEs



Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services

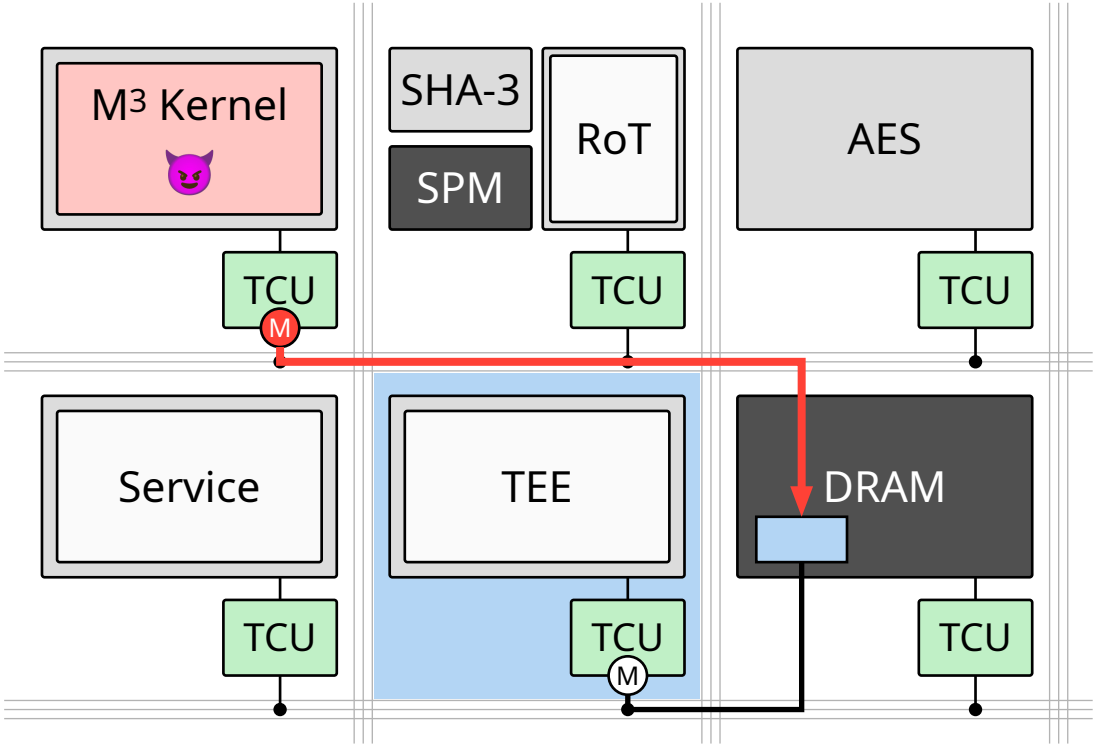
Tile-Based TEEs



Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

Tile-Based TEEs



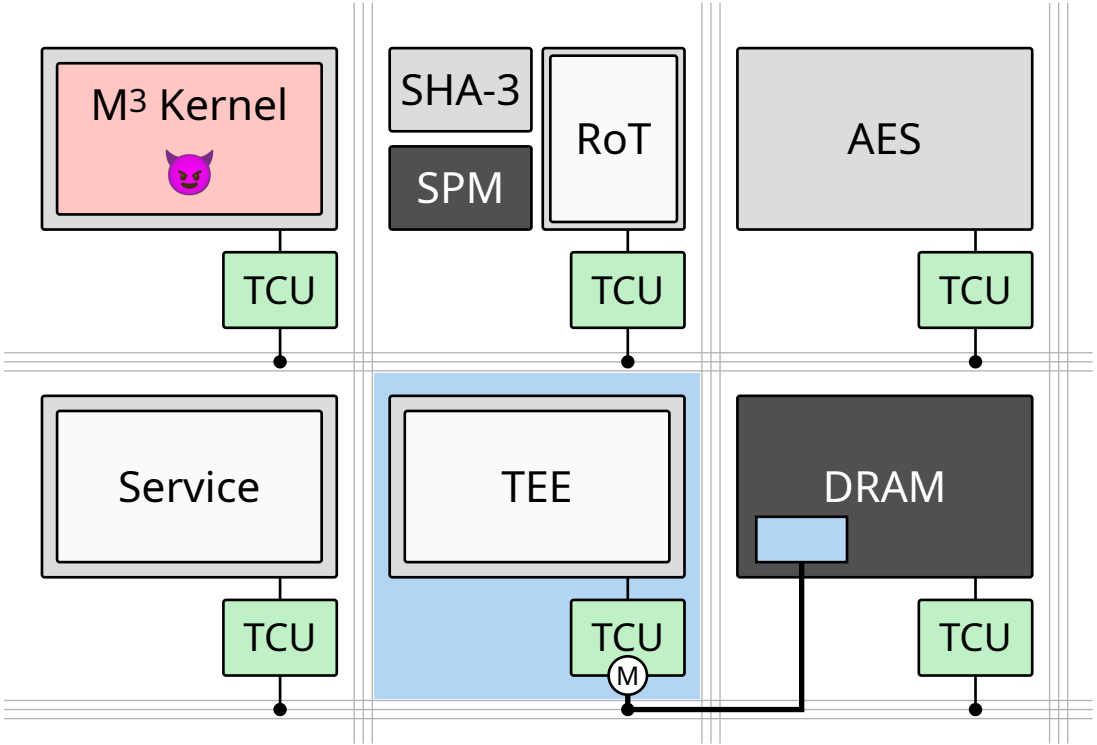
Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

Challenges:

1. **Kernel Privileges**

Tile-Based TEEs



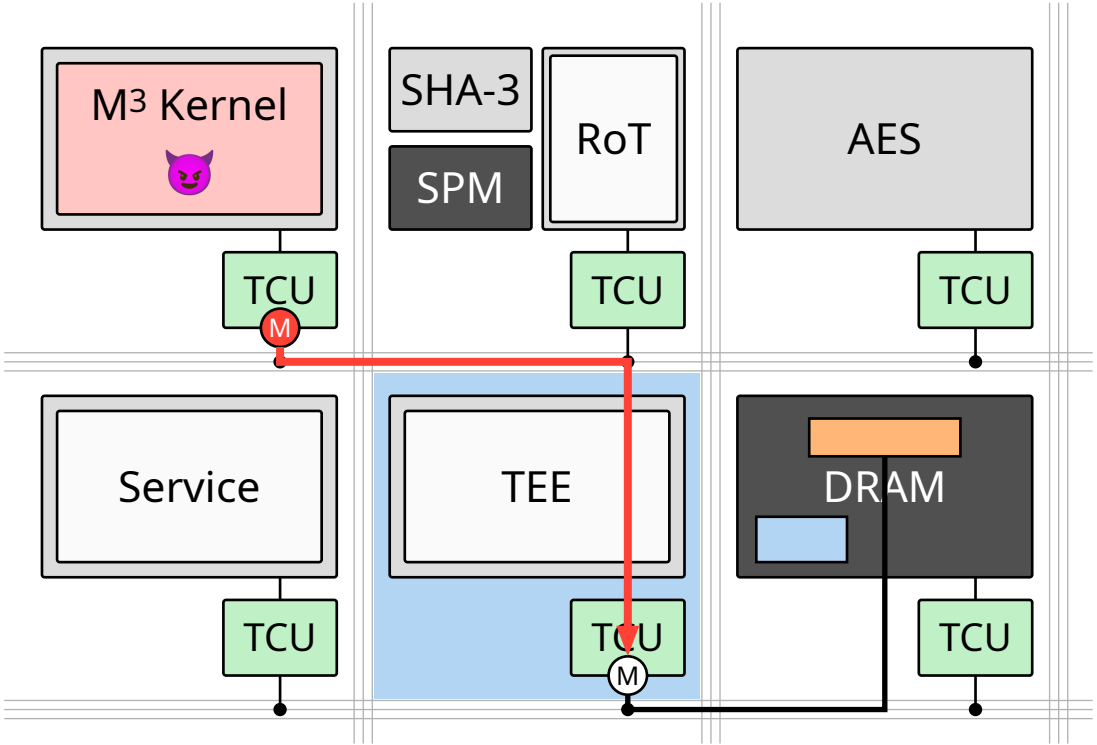
Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

Challenges:

1. Kernel Privileges
2. **Isolation vs. Management**

Tile-Based TEEs



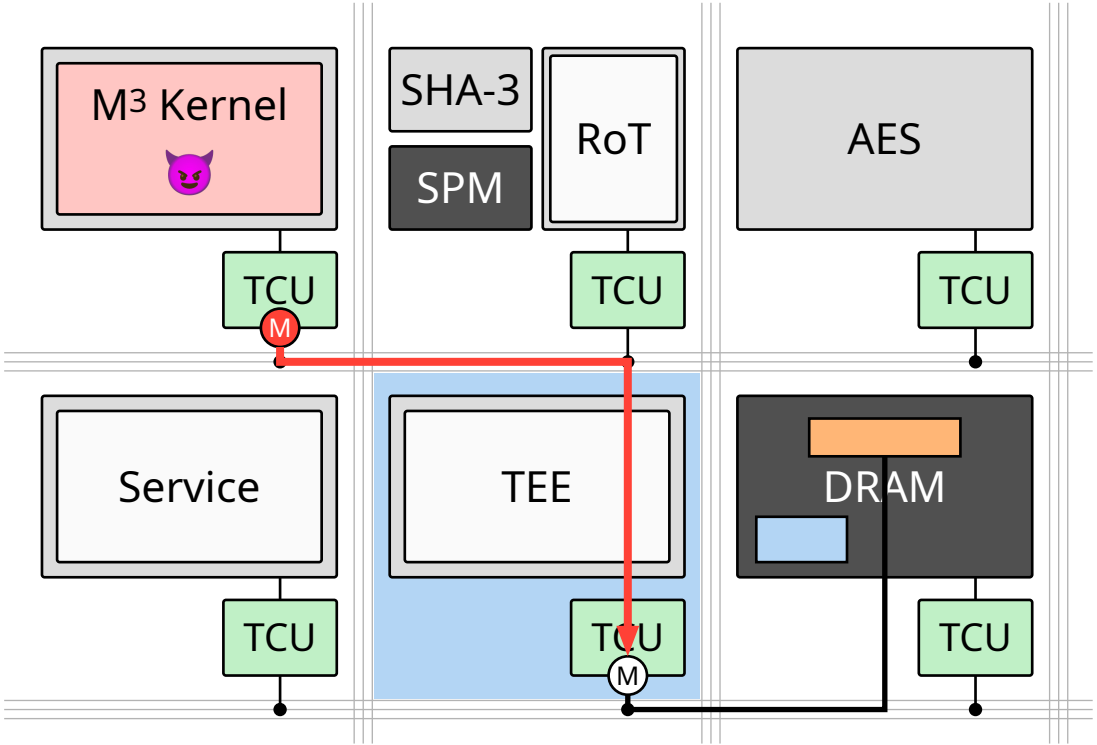
Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

Challenges:

1. Kernel Privileges
2. **Isolation vs. Management**

Tile-Based TEEs



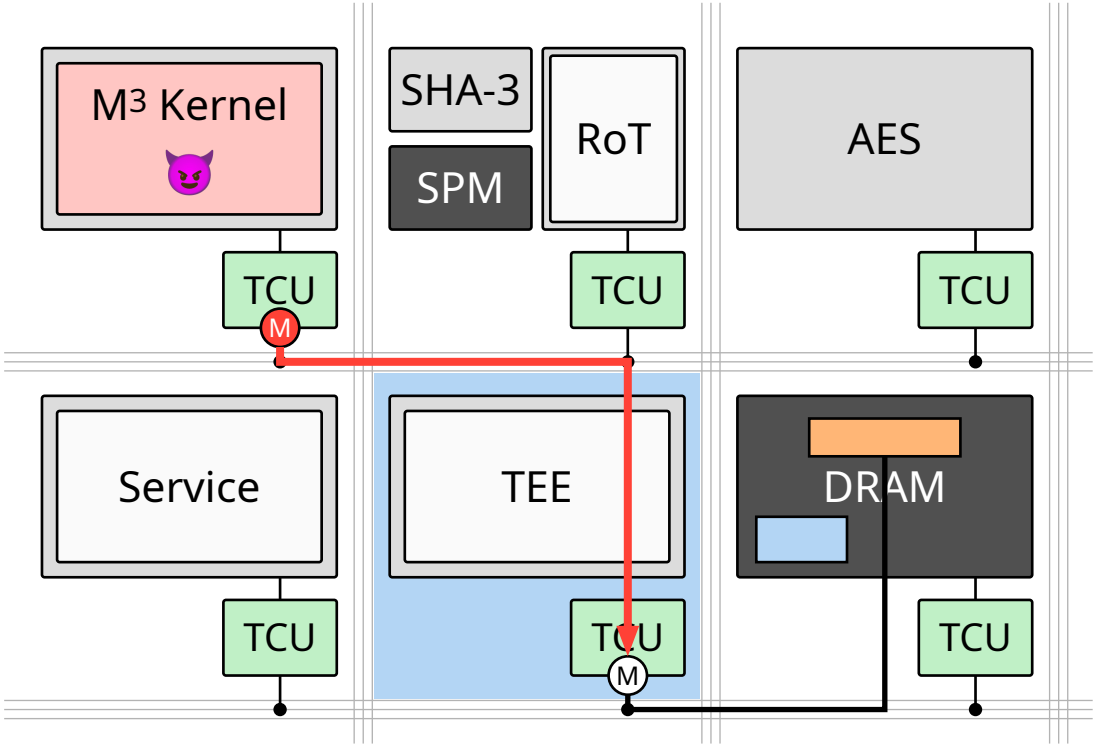
Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

Challenges:

1. Kernel Privileges
2. Isolation vs. Management
3. **Runtime Measurement**

Tile-Based TEEs



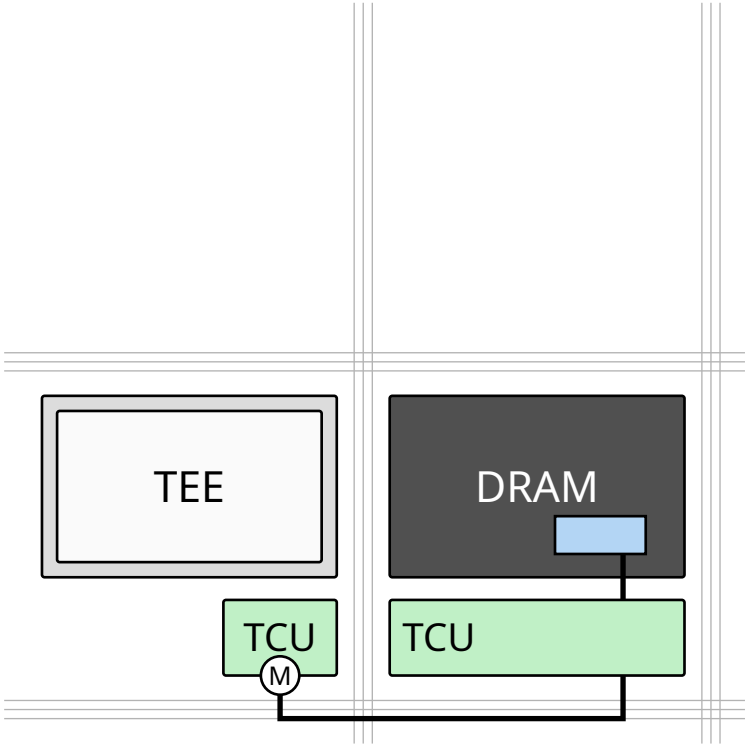
Good starting point:

- ✅ Tile isolation via TCU (core independent)
- ✅ Remote OS services
- ✅ Root of trust

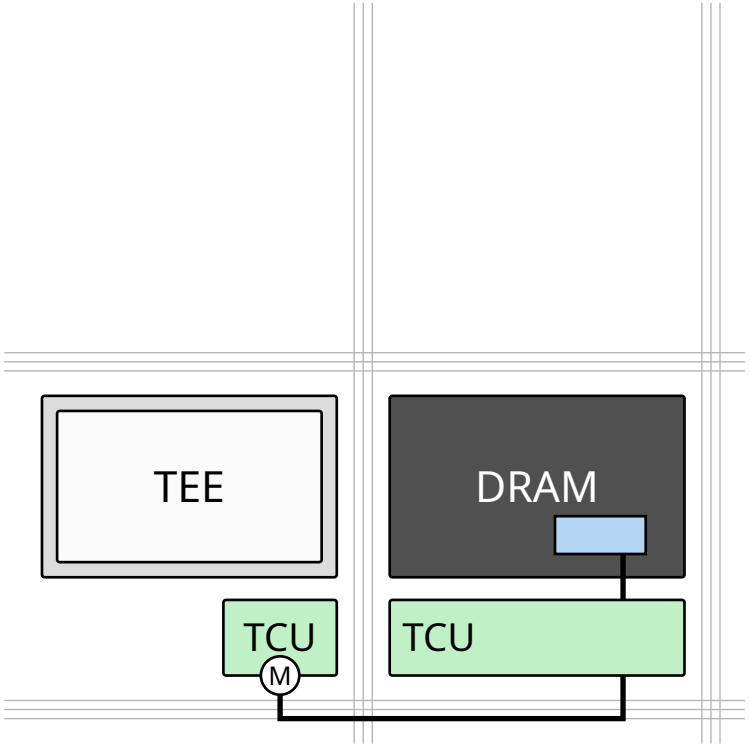
Challenges:

1. Kernel Privileges
2. Isolation vs. Management
3. ~~Runtime Measurement~~

#C1 – Kernel Privileges



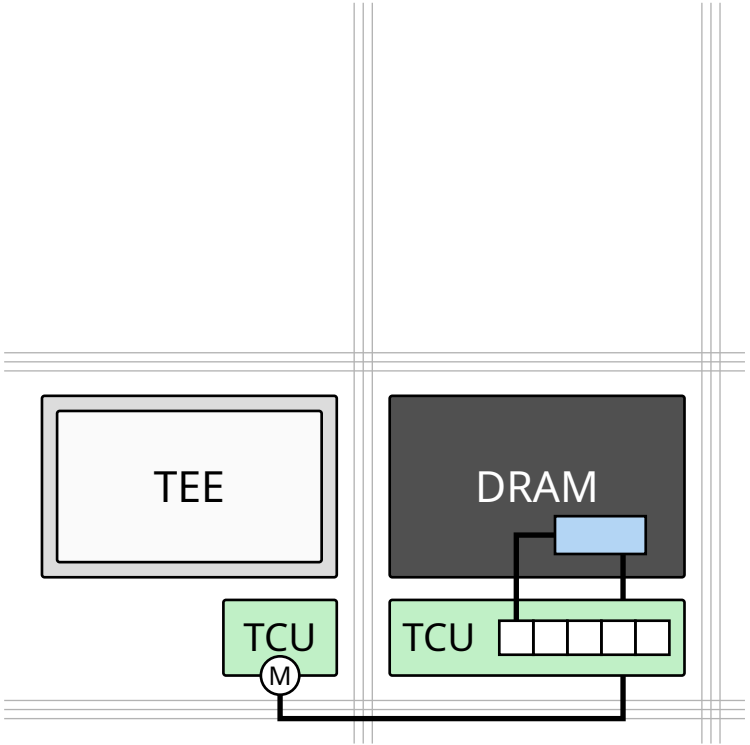
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile

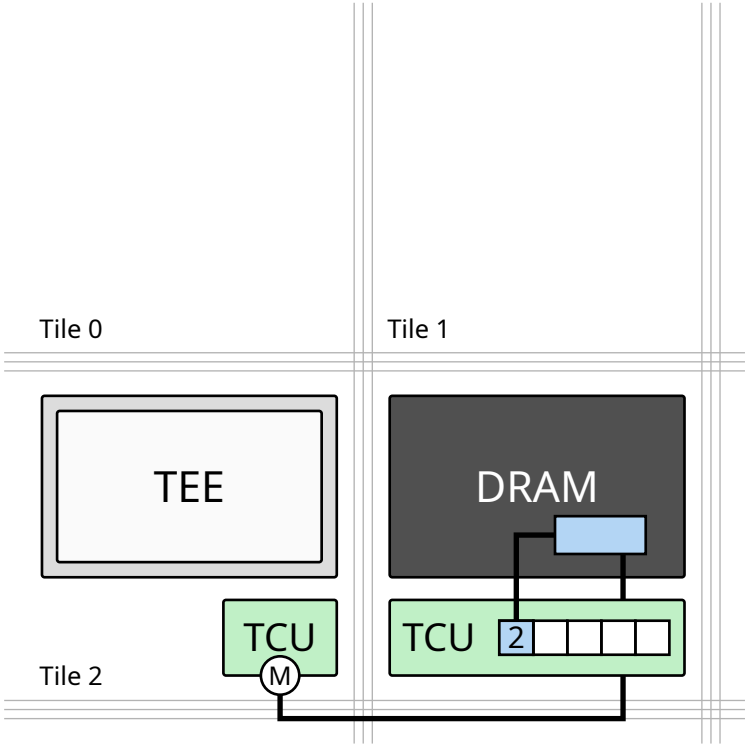
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*

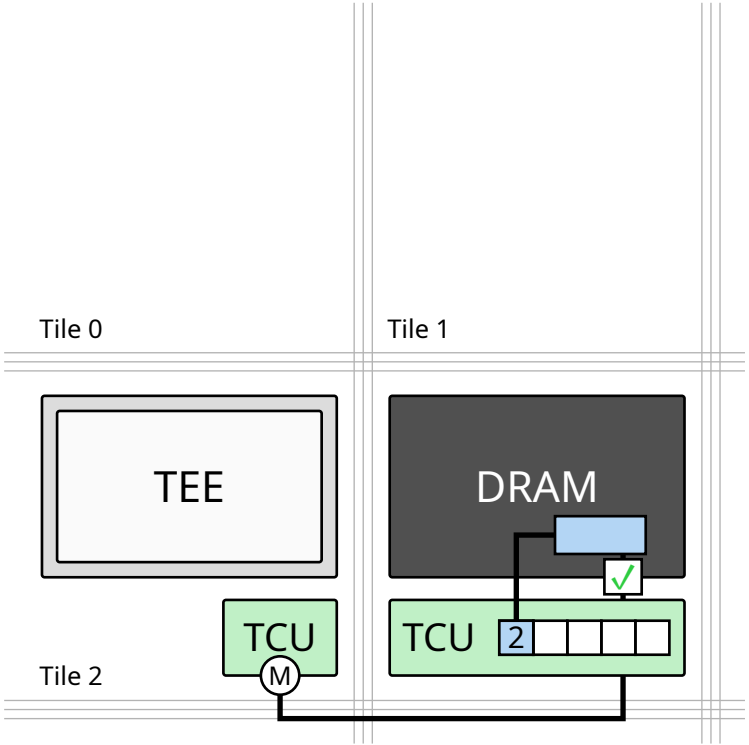
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

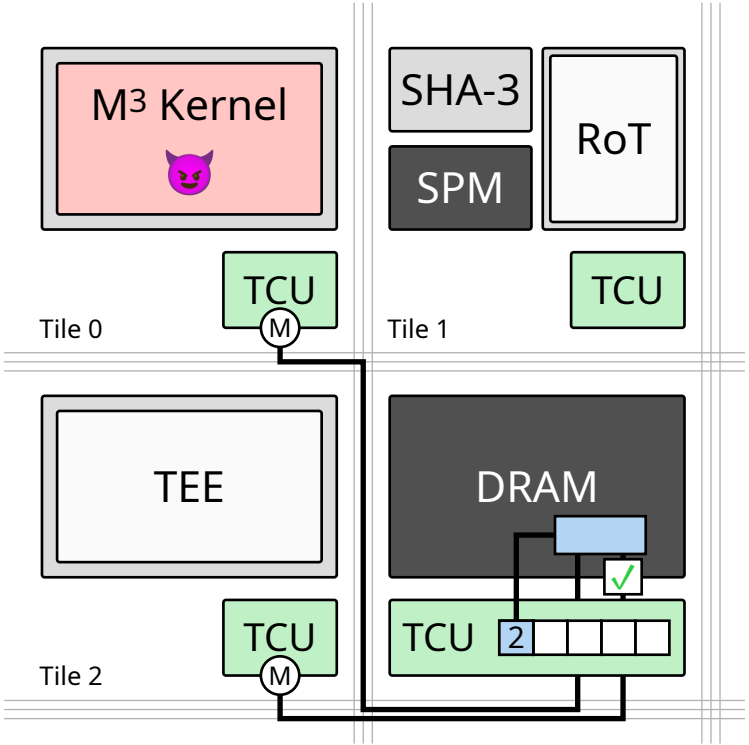
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

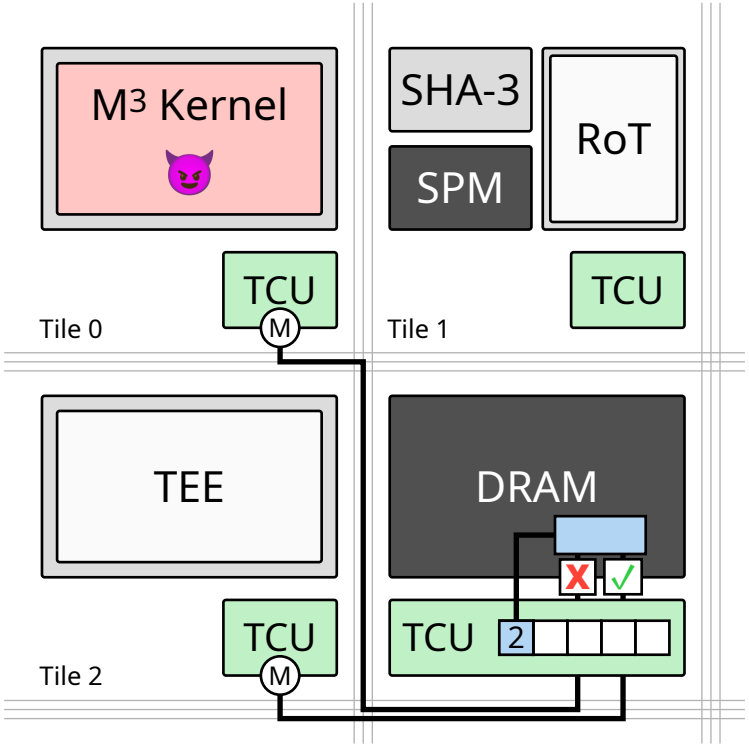
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

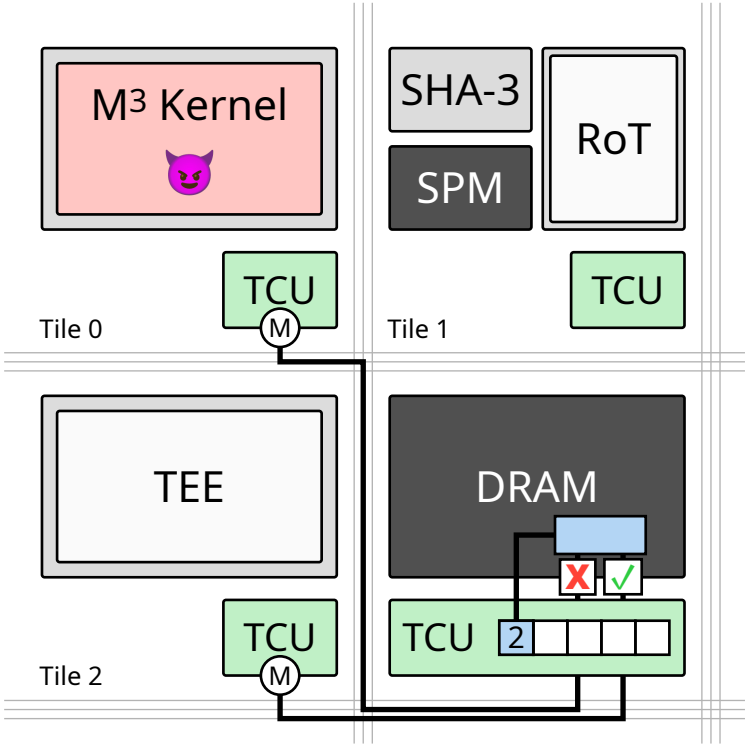
#C1 – Kernel Privileges



Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

#C1 – Kernel Privileges

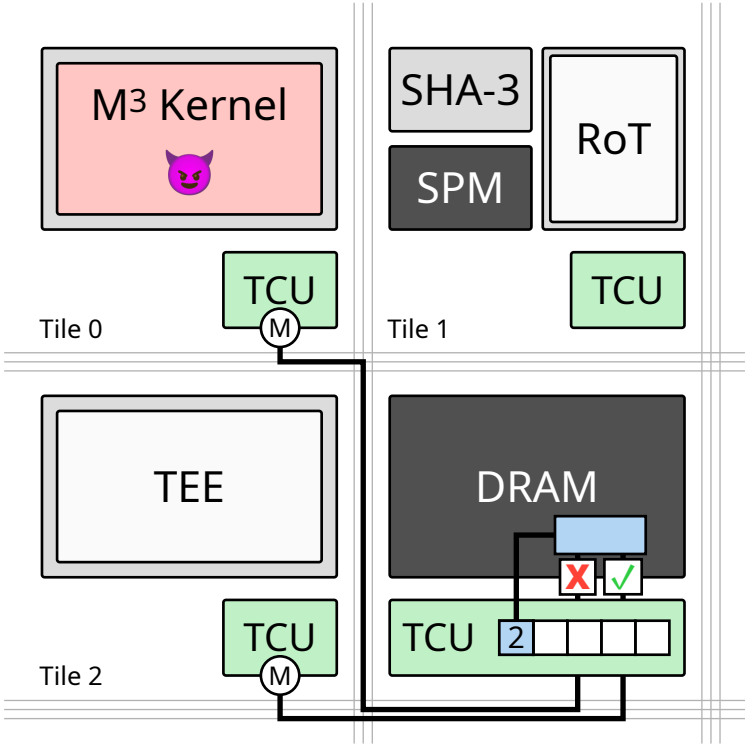


Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

Region Management:

#C1 – Kernel Privileges



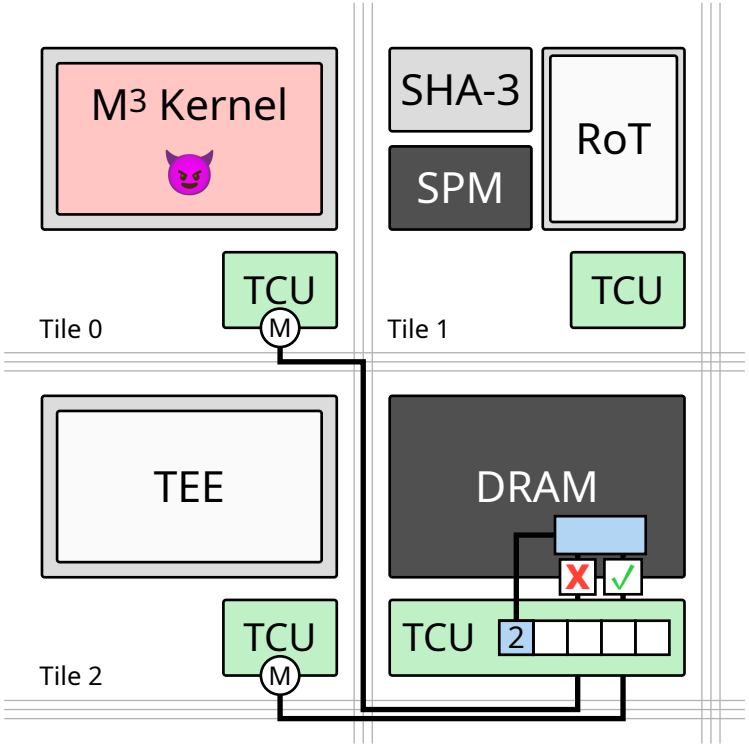
Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

Region Management:

- Only RoT can modify region registers

#C1 – Kernel Privileges



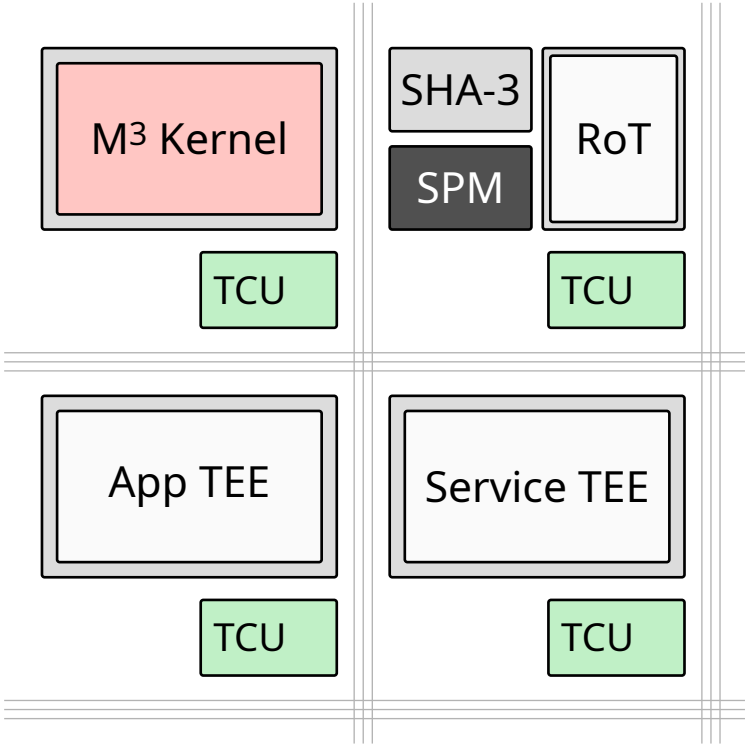
Approach:

- All memory accesses are interposed by TCU at memory tile
- Introduce *exclusive memory regions*
- Access restricted to specific tile

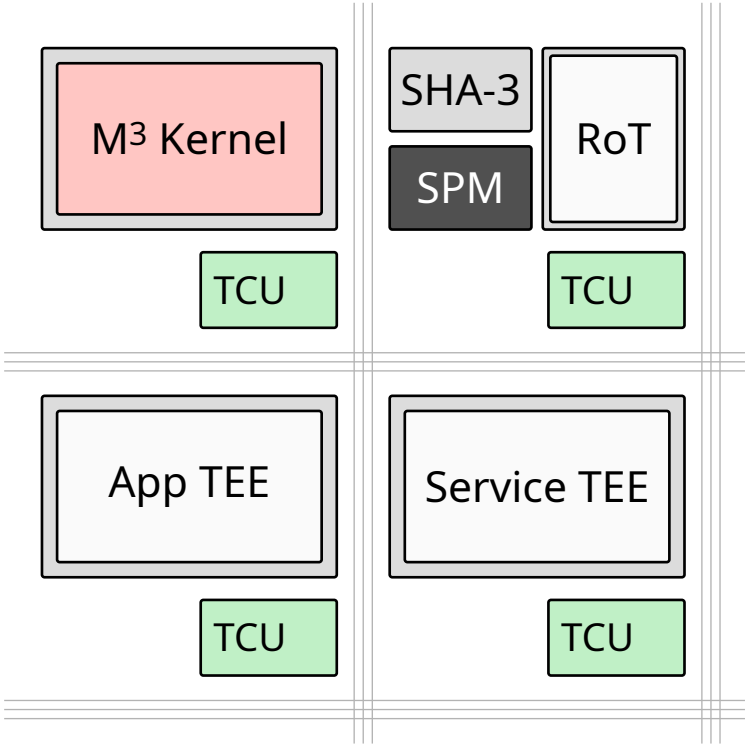
Region Management:

- Only RoT can modify region registers
- Zeros memory before reassignment

#C2 – Isolation vs. Management



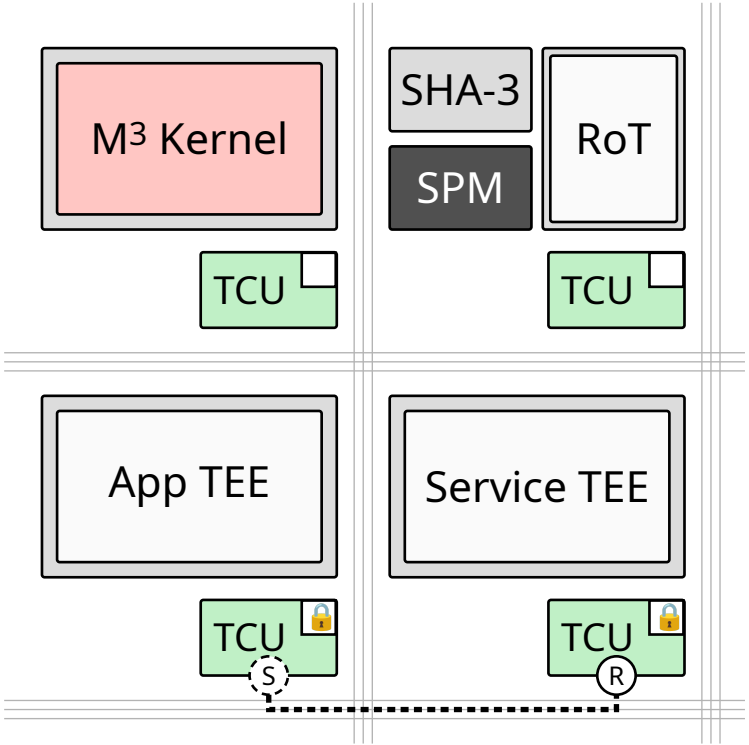
#C2 – Isolation vs. Management



Approach:

- Kernel configures TCU endpoints

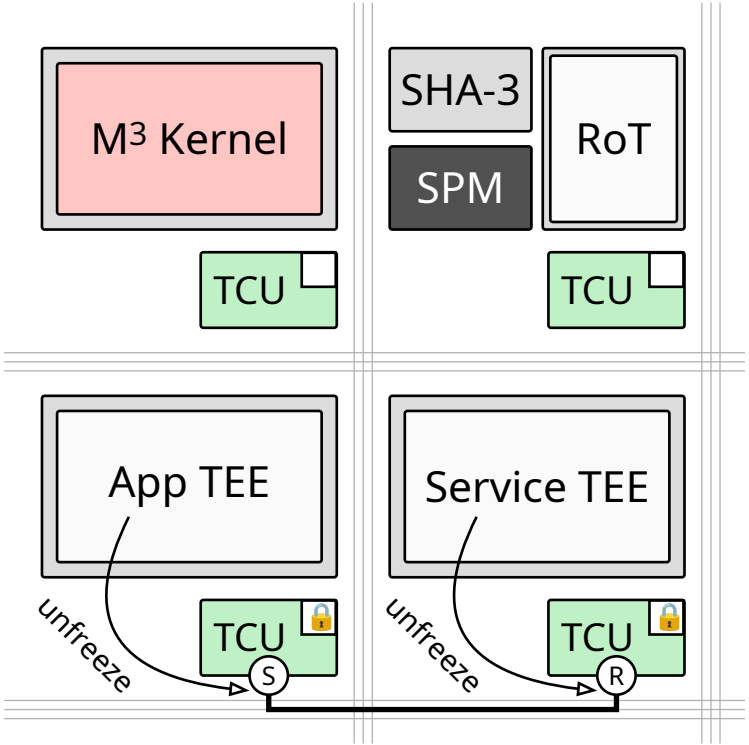
#C2 – Isolation vs. Management



Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes

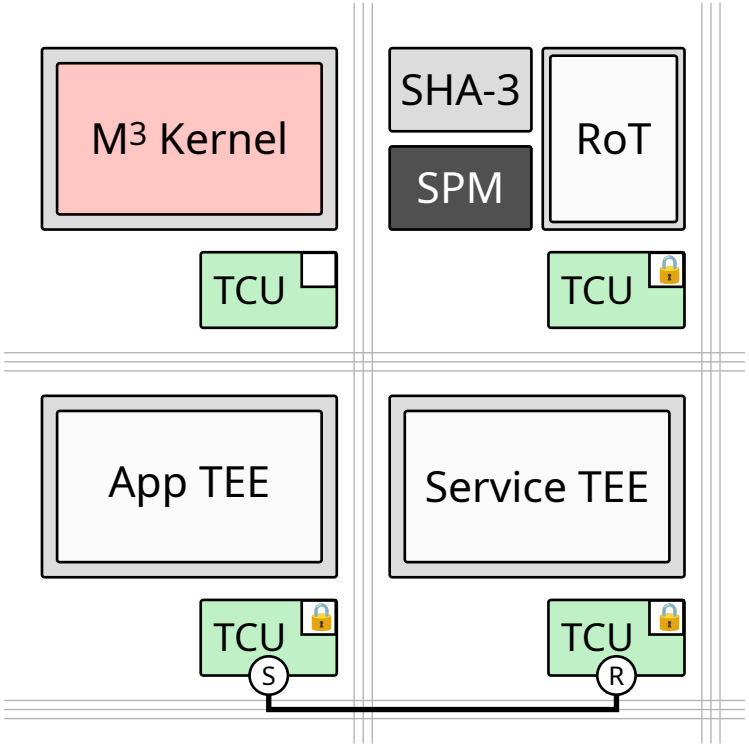
#C2 – Isolation vs. Management



Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes

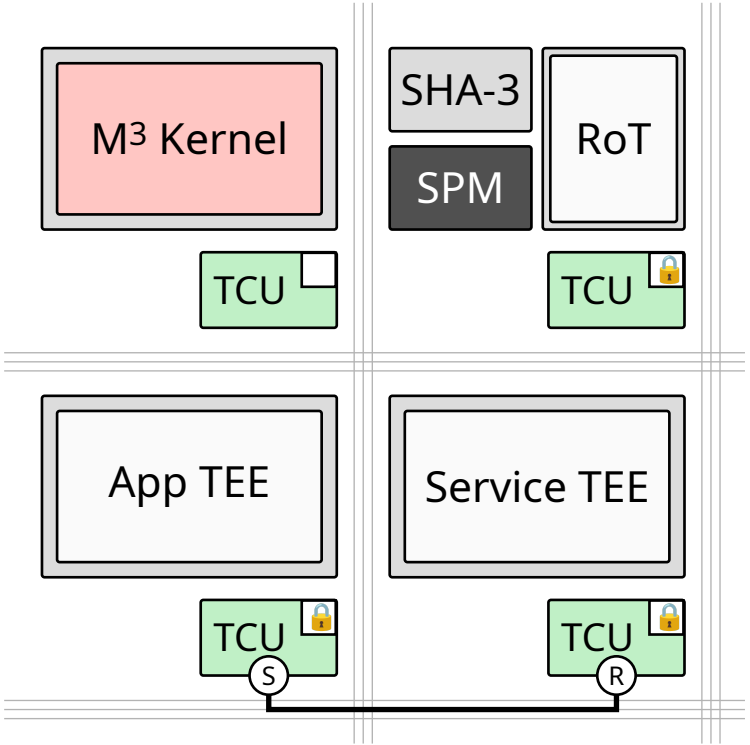
#C2 – Isolation vs. Management



Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes
- RoT locks its tile to protect itself

#C2 – Isolation vs. Management

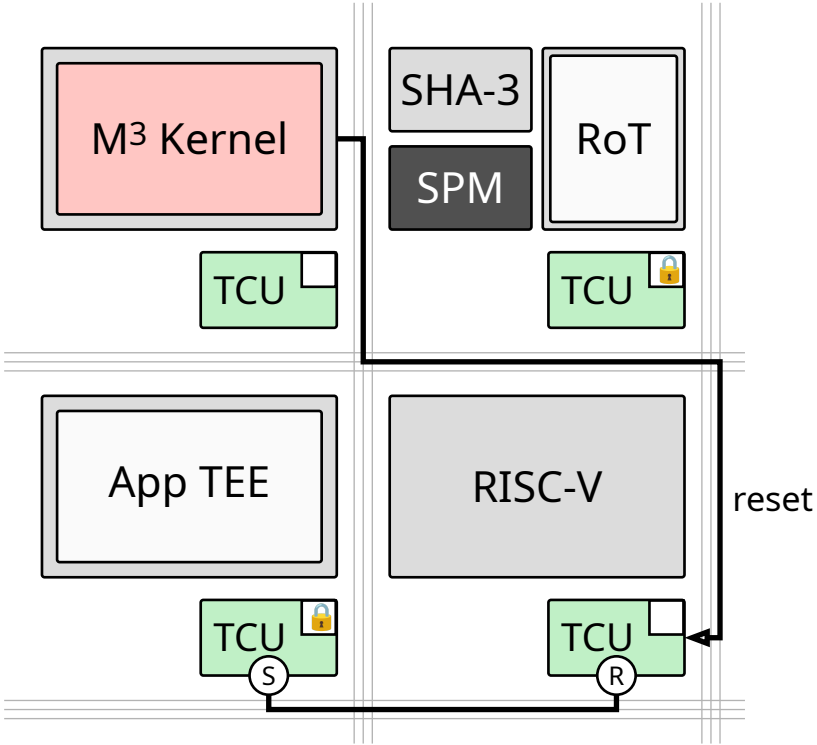


Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes
- RoT locks its tile to protect itself

Misbehaving Applications:

#C2 – Isolation vs. Management



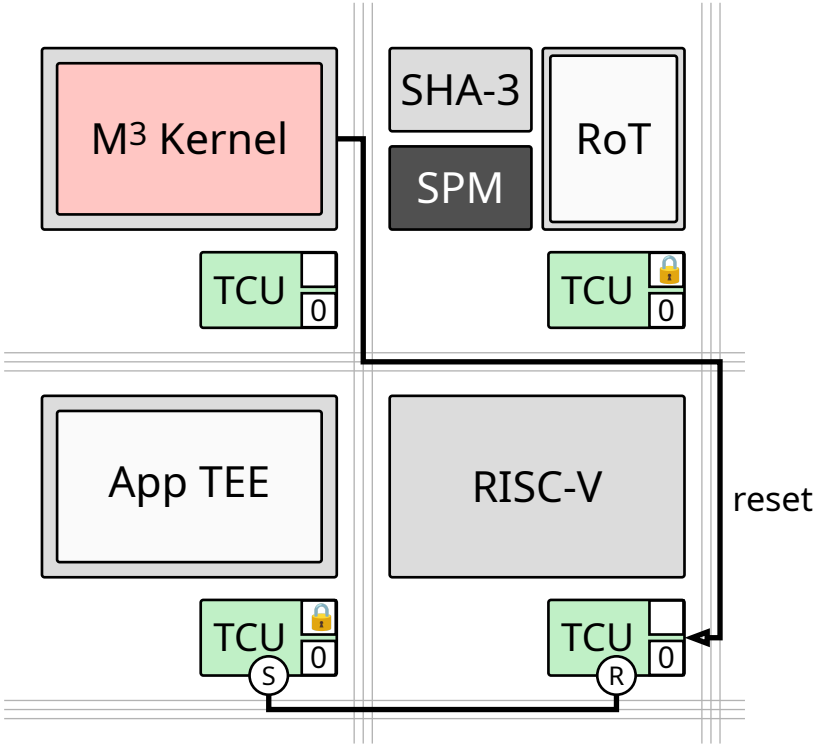
Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes
- RoT locks its tile to protect itself

Misbehaving Applications:

- Kernel can *reset* tiles

#C2 – Isolation vs. Management



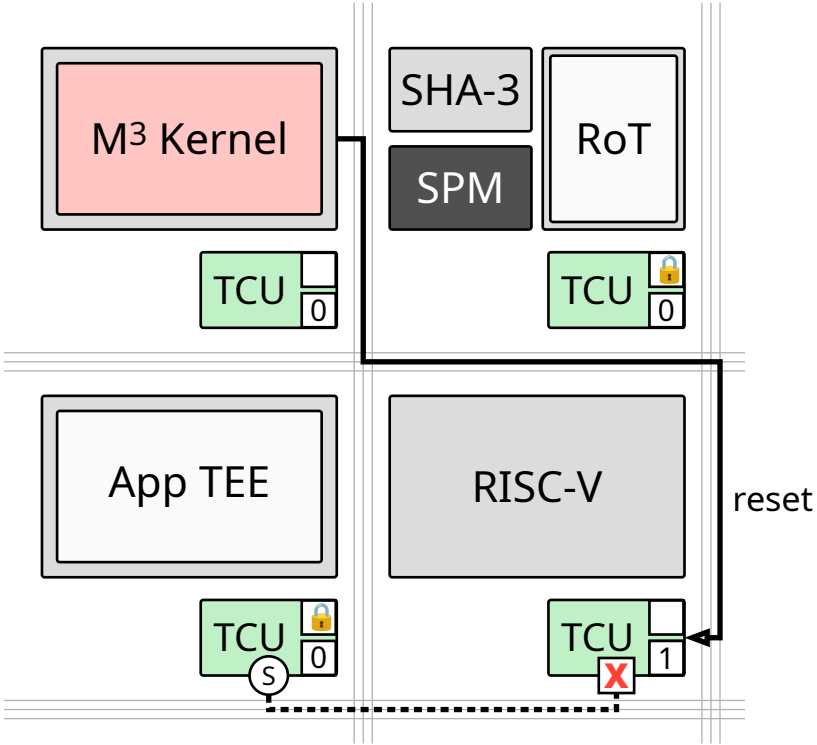
Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes
- RoT locks its tile to protect itself

Misbehaving Applications:

- Kernel can *reset* tiles
- Detection: generation counter

#C2 – Isolation vs. Management



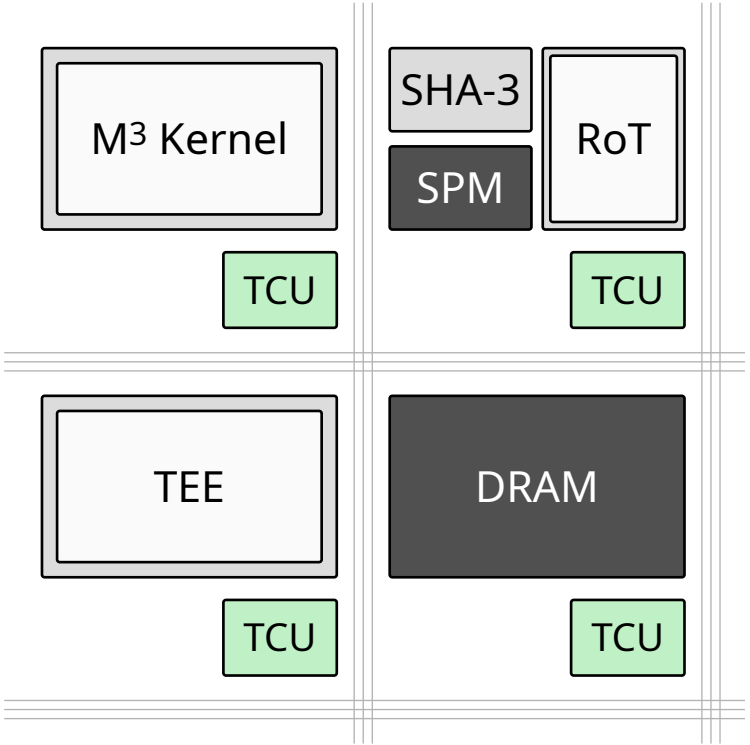
Approach:

- Kernel configures TCU endpoints
- Introduce *tile locking*:
  TEE needs to ACK endpoint changes
- RoT locks its tile to protect itself

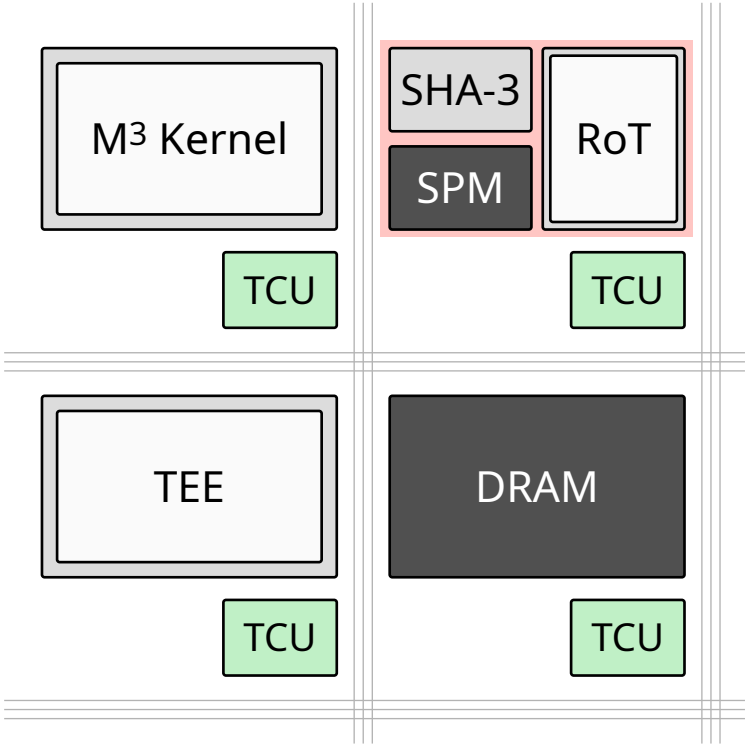
Misbehaving Applications:

- Kernel can *reset* tiles
- Detection: generation counter
- Increased on reset, stored in communication channels

TCB and Limitations

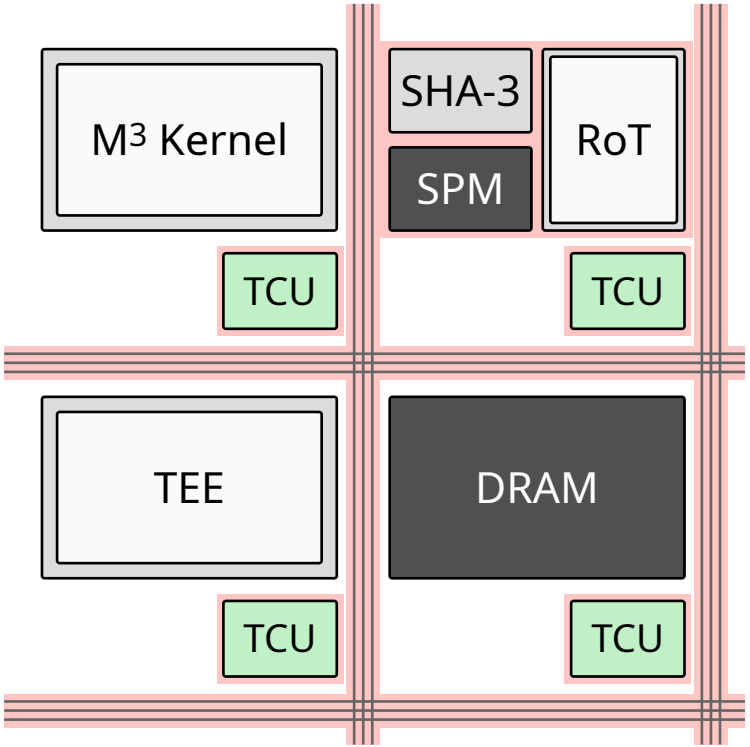


TCB and Limitations



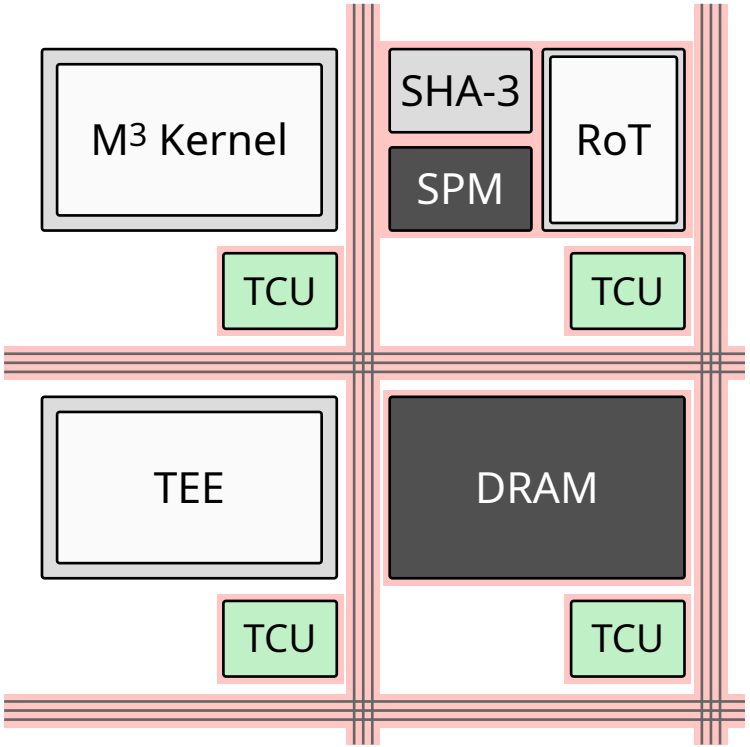
- **Root of Trust**

TCB and Limitations



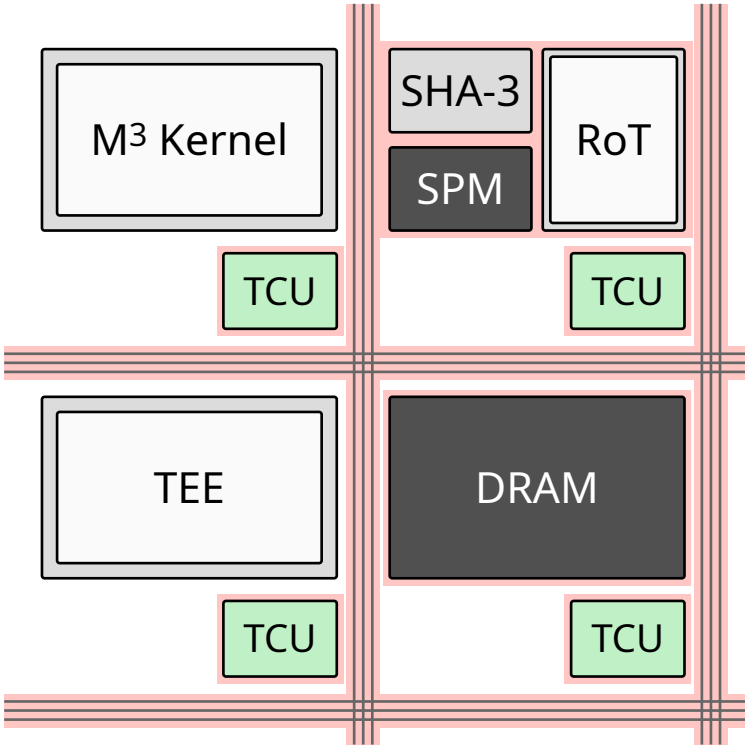
- Root of Trust
- **NoC, and TCUs are trusted**
  - Reasonable for SoCs
  - Going off-chip requires encryption

TCB and Limitations



- Root of Trust
- NoC, and TCUs are trusted
  - Reasonable for SoCs
  - Going off-chip requires encryption
- **Memory encryption optional**
  - Encryption at chip boundary

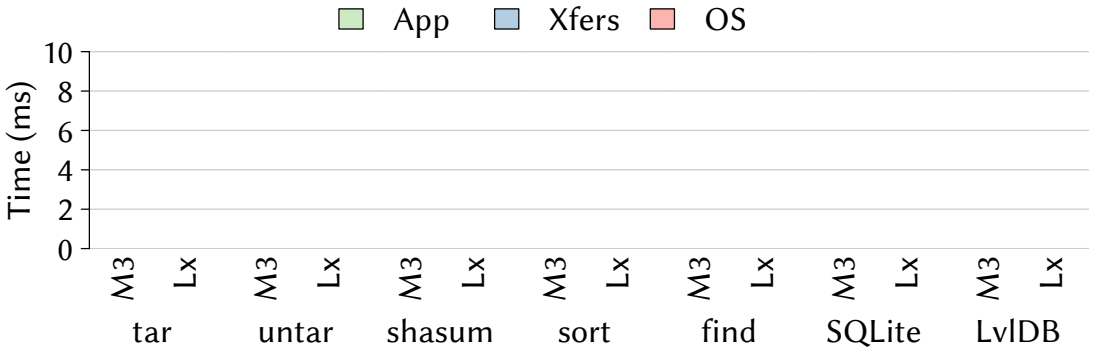
TCB and Limitations



- Root of Trust
- NoC, and TCUs are trusted
  - Reasonable for SoCs
  - Going off-chip requires encryption
- Memory encryption optional
  - Encryption at chip boundary
- **Kernel only trusted for availability**
  - Cores/accelerators not in TCB

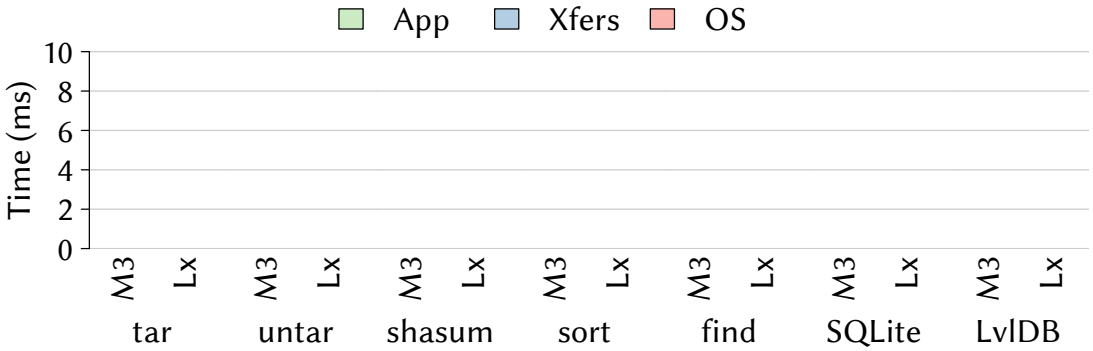
- Introduction
- The New System Architecture
- Prototype Platforms
- Isolation and Communication
- Operating System
- OS Services and Accelerators
- Context Switching
- Trusted Execution Environments
- **Evaluation**

Linux Application Workloads

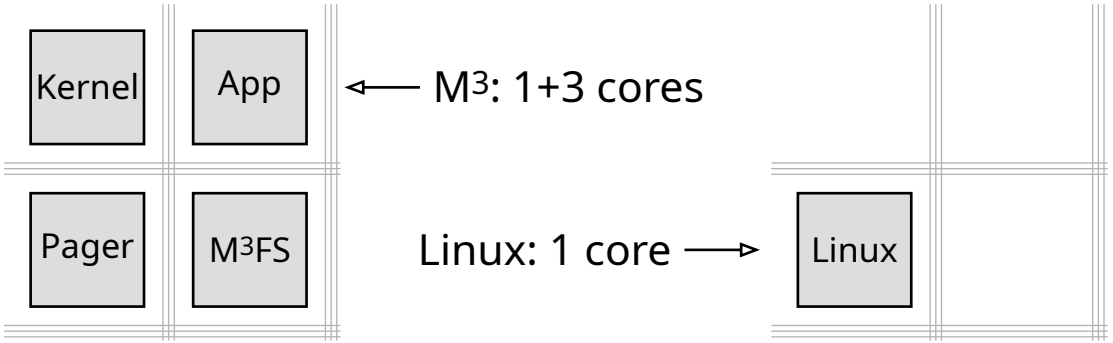


- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs

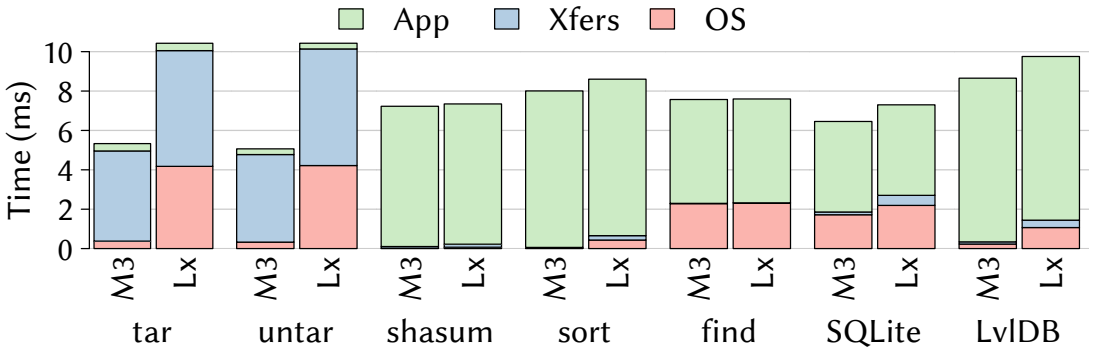
Linux Application Workloads



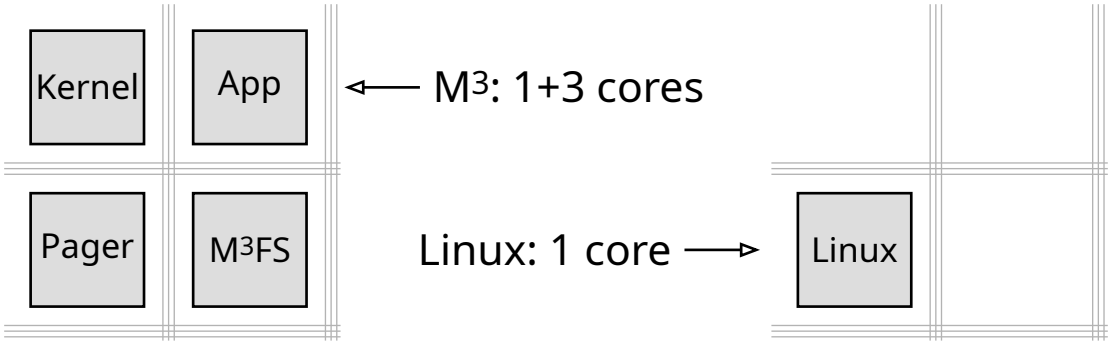
- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs



Linux Application Workloads

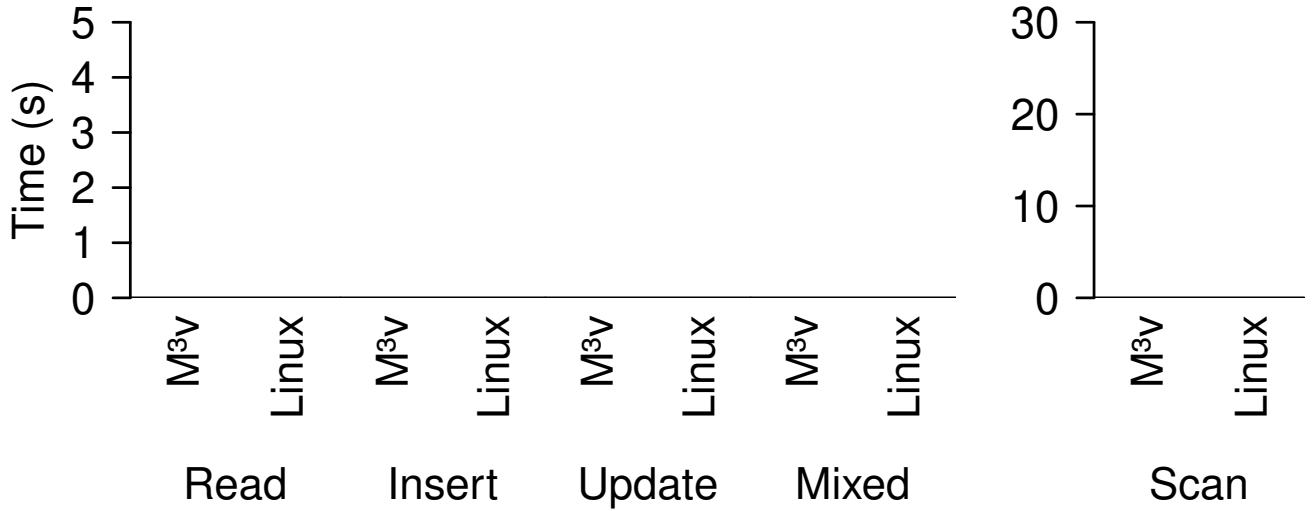


- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs



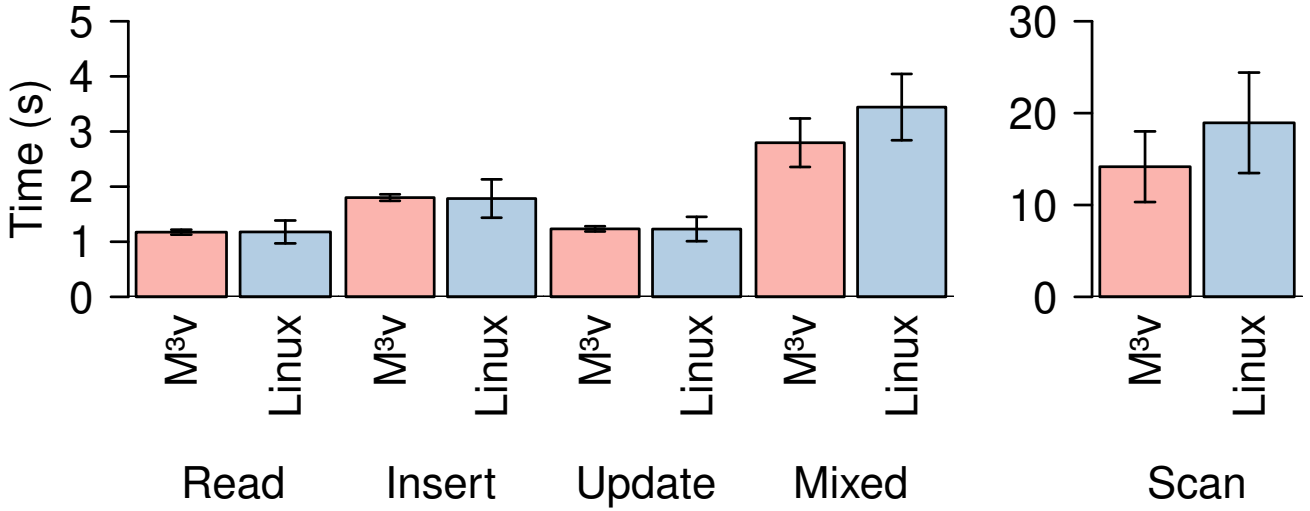
Performance Comparison with Core Sharing

- LevelDB receives requests from remote machine and sends result back
- Requests generated with YCSB; different shares of read/insert/update/scan
- Single BOOM core runs: LevelDB, pager, filesystem, network stack



Performance Comparison with Core Sharing

- LevelDB receives requests from remote machine and sends result back
- Requests generated with YCSB; different shares of read/insert/update/scan
- Single BOOM core runs: LevelDB, pager, filesystem, network stack



Hardware Costs

Component	Total area [mm^2]	SRAM area [mm^2]	Percent
Rocket tile	1.123	0.710	-
• Rocket core	0.880	0.599	78%
• TCU	0.104	0.053	9%
BOOM tile	1.606	0.973	-
• BOOM core	1.360	0.861	84%
• TCU	0.104	0.053	6%
AES tile	0.421	0.207	-
• Pico core	0.138	0.130	33%
• AES accelerator	0.107	0.012	25%
• TCU	0.036	0.008	9%

Hardware Costs

Component	Total area [mm^2]	SRAM area [mm^2]	Percent
Rocket tile	1.123	0.710	-
• Rocket core	0.880	0.599	78%
• TCU	0.104	0.053	9%
BOOM tile	1.606	0.973	-
• BOOM core	1.360	0.861	84%
• TCU	0.104	0.053	6%
AES tile	0.421	0.207	-
• Pico core	0.138	0.130	33%
• AES accelerator	0.107	0.012	25%
• TCU	0.036	0.008	9%

Ongoing Work at the Barkhausen Institut

- Selective cache coherence
- Running Linux on M^3
- Running L4 on M^3
- OS control over NoC routing
- Providing real-time guarantees
- Remote Attestation with sealed memory

Conclusion

- M³ explores a system architecture with a new per-tile hardware component
- TCU introduces common interface for all cores/accelerators
- Allows to integrate untrusted cores/accelerators, including OS-service access
- Provides generic TEEs for cores and accelerators
- Hardware implementation demonstrates modest additional cost
- Complete hardware/software stack available as open source: https://github.com/Barkhausen-Institut/M3