Microkernel Construction

Case Study: M^3

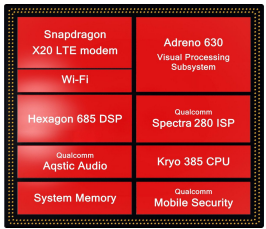
Nils Asmussen

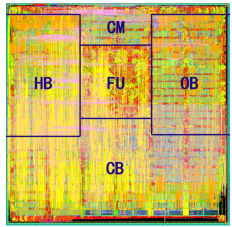
July 11th 2024

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

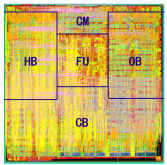
Heterogeneous Systems

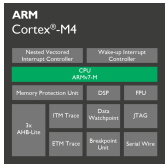




- Demanded by performance and energy requirements
- Big challenge for OSes: single shared kernel on all cores does no longer work
- OSes need to be prepared for processing elements with different feature sets

Third-party Components



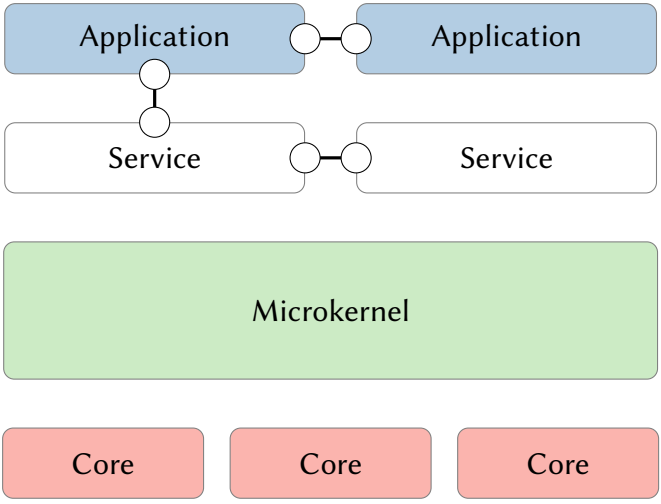


- Market pressure forces us to integrate third-party components
- We should not trust these components
- Currently, often no isolation between them
- Bug in such a component can compromise whole system (see Broadcom incident)

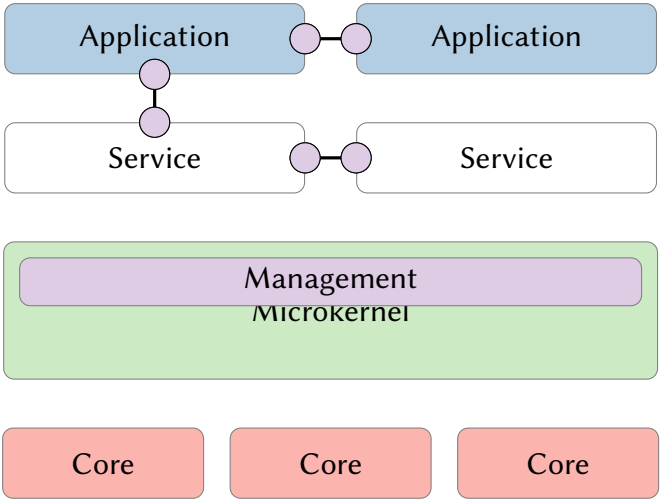
Security Issues of Complex General-purpose Cores

- 26 known attacks (and counting …)
- Allow to leak private data, sometimes bypassing all security measures of the core
- 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?

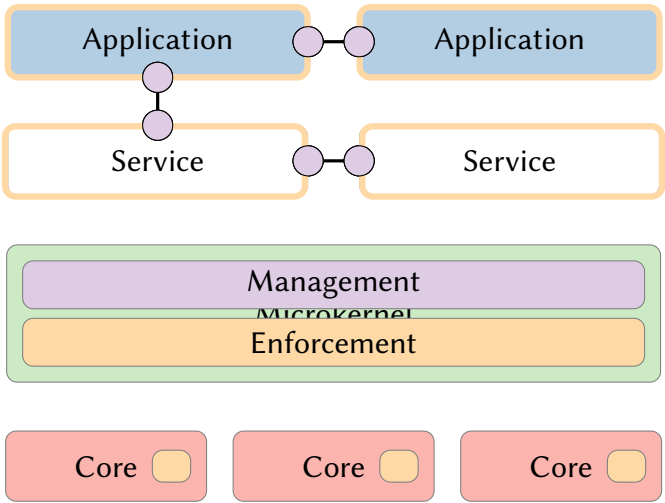
Microkernel-based System as Foundation



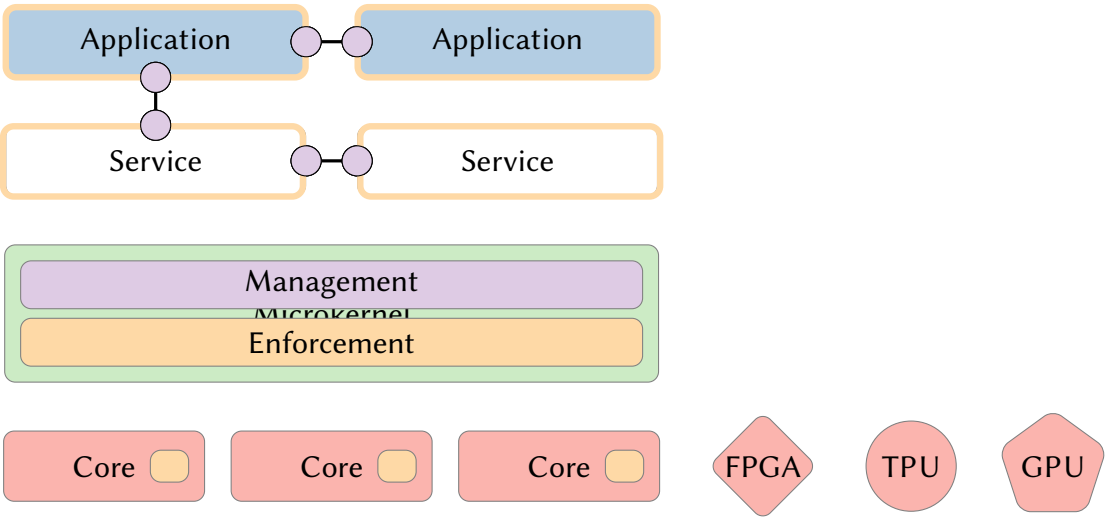
Microkernel-based System as Foundation



Microkernel-based System as Foundation



Microkernel-based System as Foundation

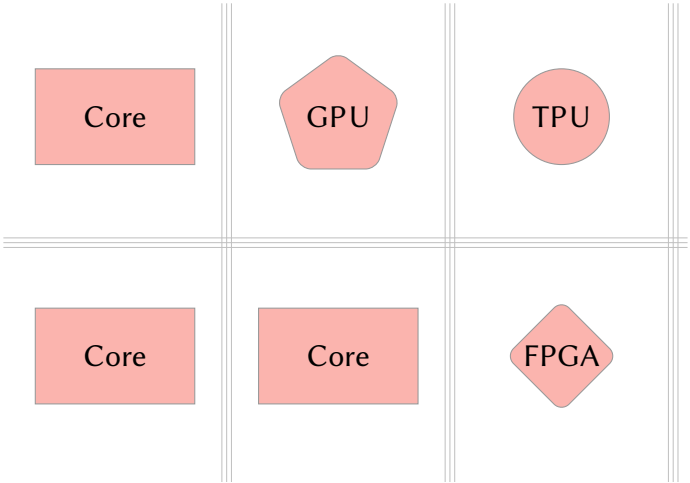


Outline

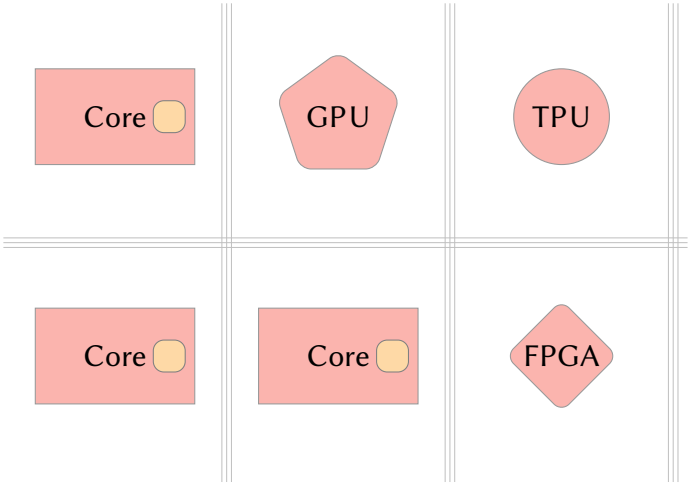
Outline

Hardware/Operating System Co-Design

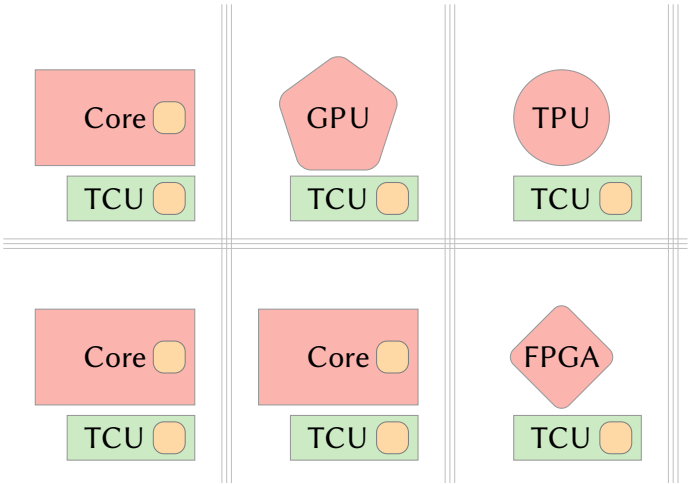
Hardware/Operating System Co-Design



Hardware/Operating System Co-Design



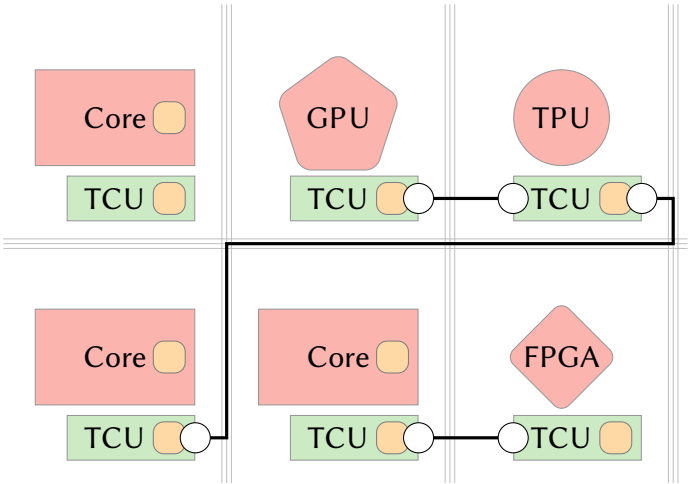
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component

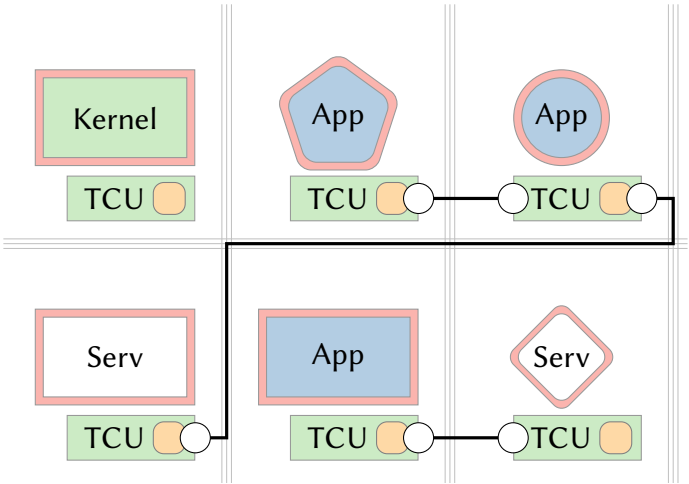
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component

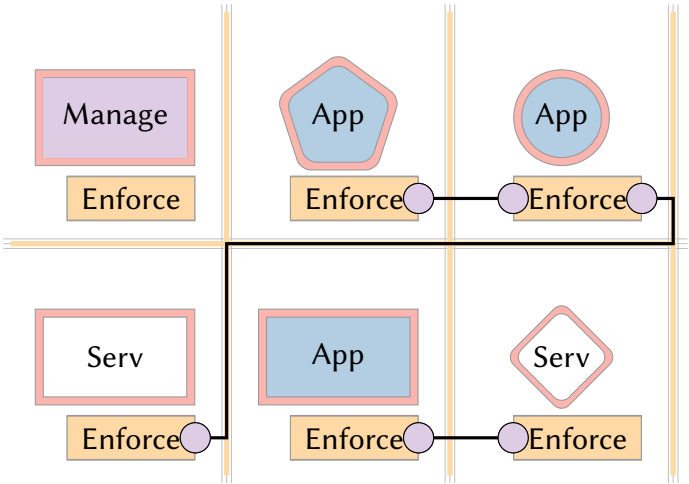
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component
- Kernel on dedicated tile

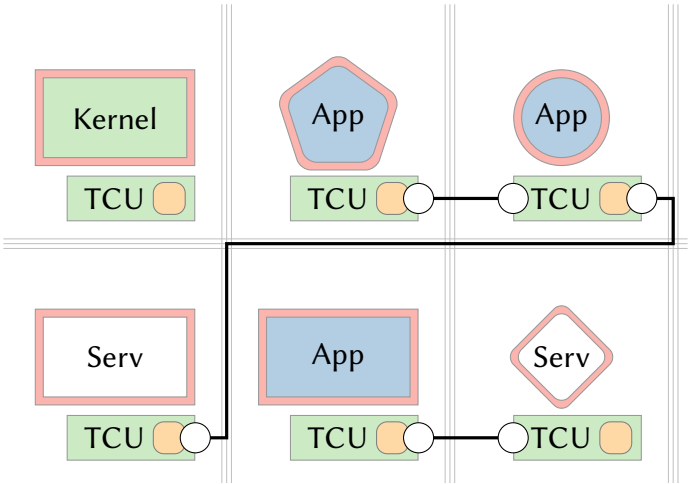
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component
- Kernel on dedicated tile
- Kernel manages, TCU enforces

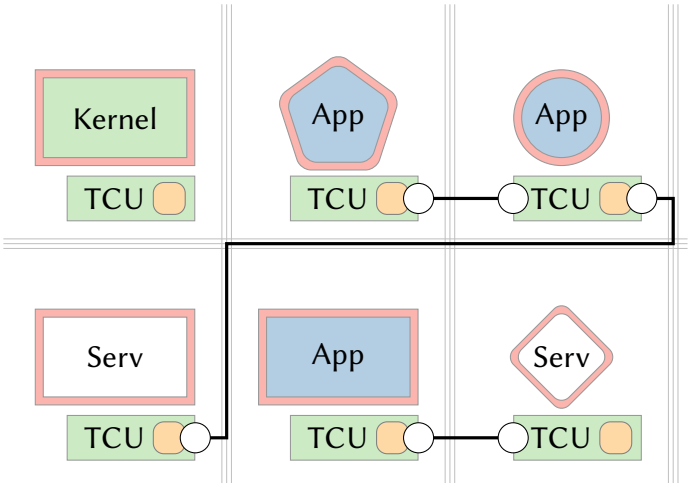
Hardware/Operating System Co-Design



Hardware challenges:

- Heterogeneity:
  Uniform interface

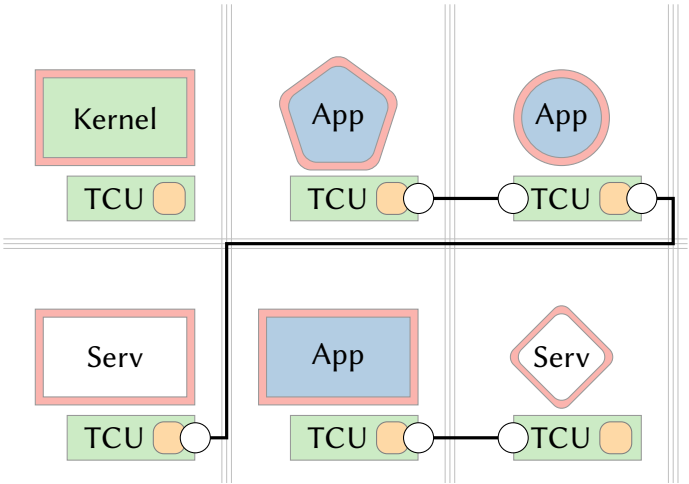
Hardware/Operating System Co-Design



Hardware challenges:

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

Hardware/Operating System Co-Design



Hardware challenges:

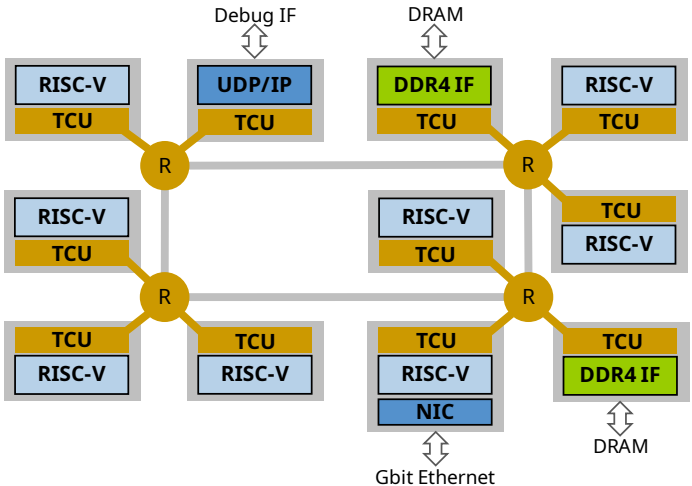
- Heterogeneity:
  Uniform interface
- Untrusted HW comp.:
  Protected by TCU
- Side channels:
  Physical isolation

Outline

gem5

- 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

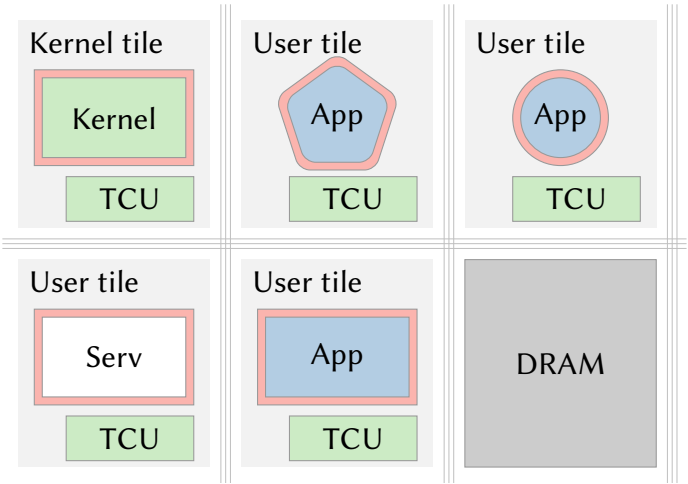
FPGA



- Xilinx VCU118 FPGA
- RISC-V: in-order Rocket or out-of-order BOOM
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- TCU contains 128 EPs

Outline

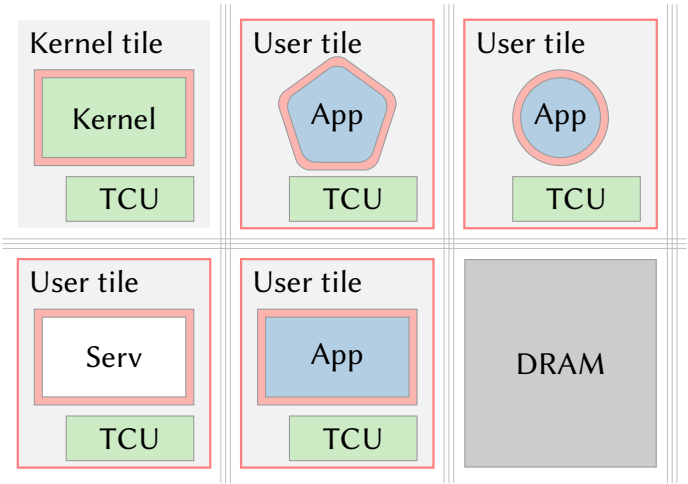
Isolation



TCU-based isolation:

- Additional protection layer

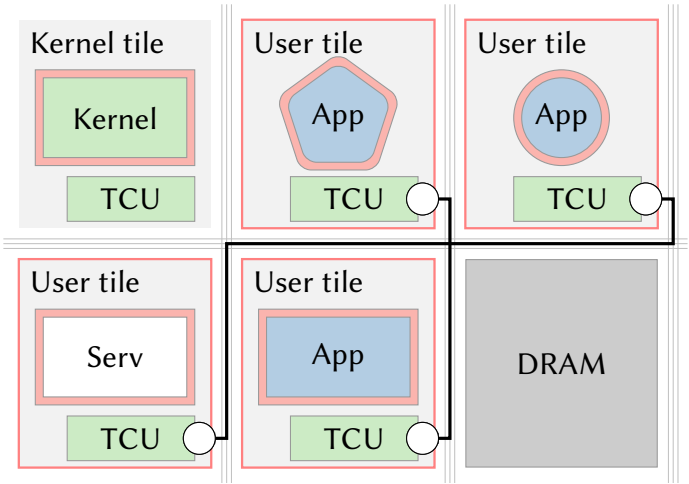
Isolation



TCU-based isolation:

- Additional protection layer

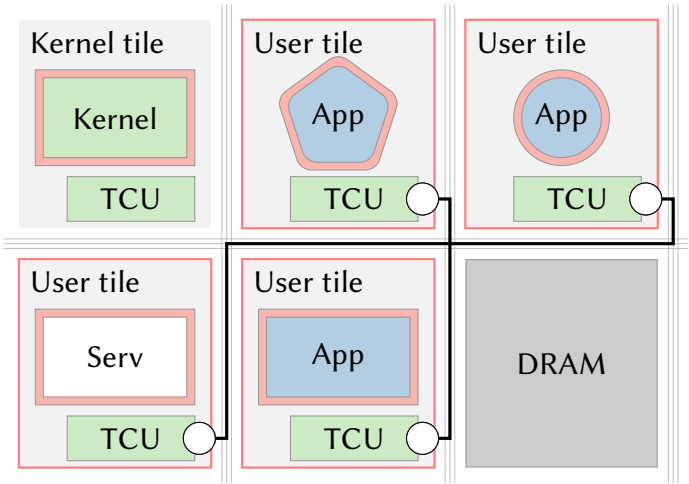
Isolation



TCU-based isolation:

- Additional protection layer
- Only kernel tile can establish communication channels

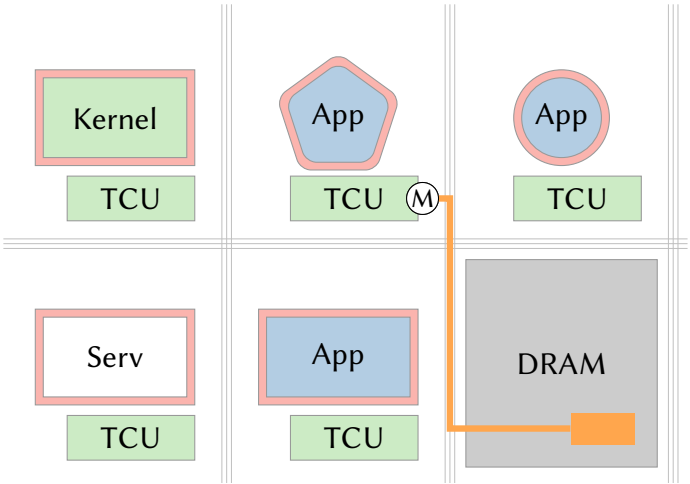
Isolation



TCU-based isolation:

- Additional protection layer
- Only kernel tile can establish communication channels
- User tiles can only use established channels

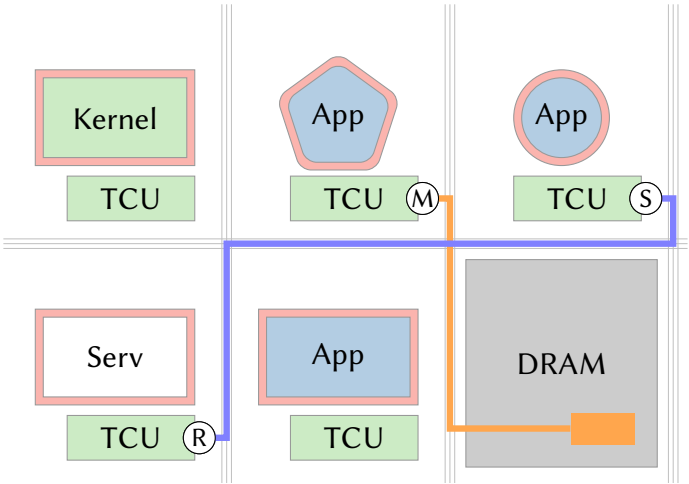
Communication



TCU provides *endpoints* to:

- Issue DMA requests to memory

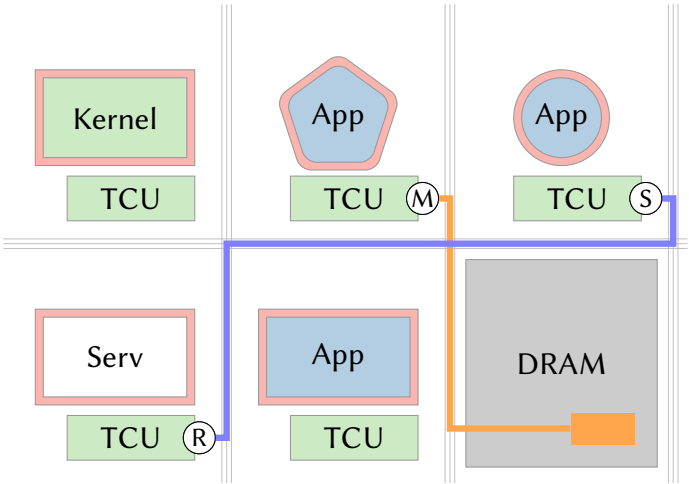
Communication



TCU provides *endpoints* to:

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

Communication



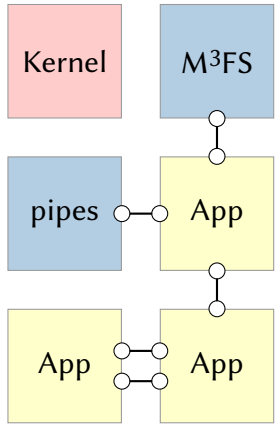
TCU provides *endpoints* to:

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

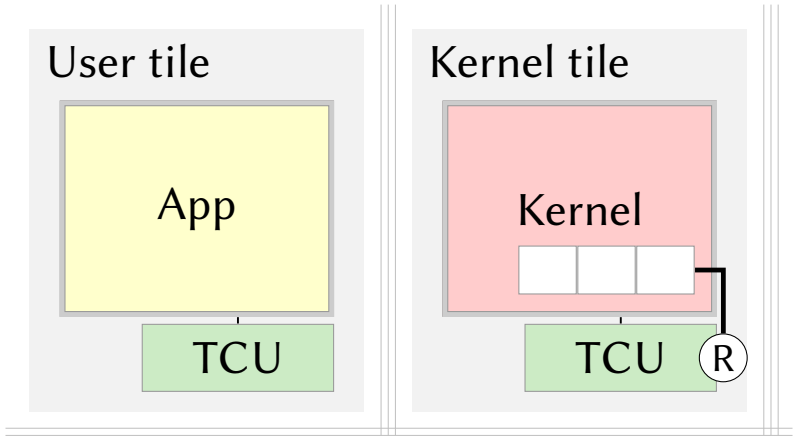
Outline

OS Design

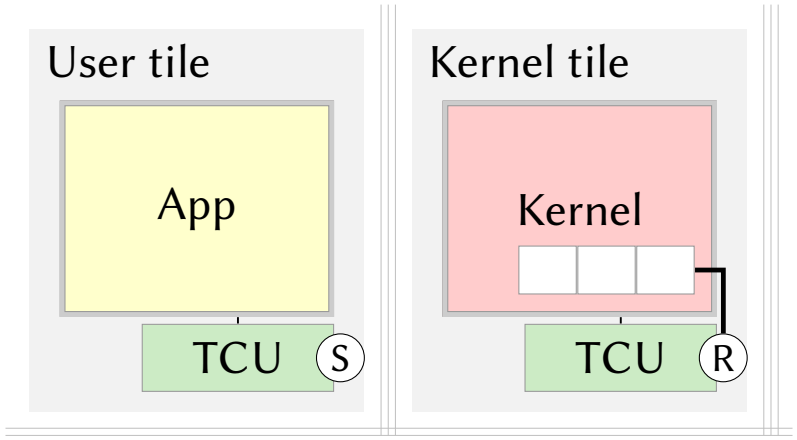
- M^3: **M**icrokernel-based syste**m** for het. **m**anycores (or L4 $\pm$ 1)
- Implemented from scratch in Rust and C++
- Drivers, filesystems, etc. implemented on user tiles
- Kernel manages permissions, using capabilities
- TCU enforces permissions (communication, memory access)
- Kernel is independent of other tiles



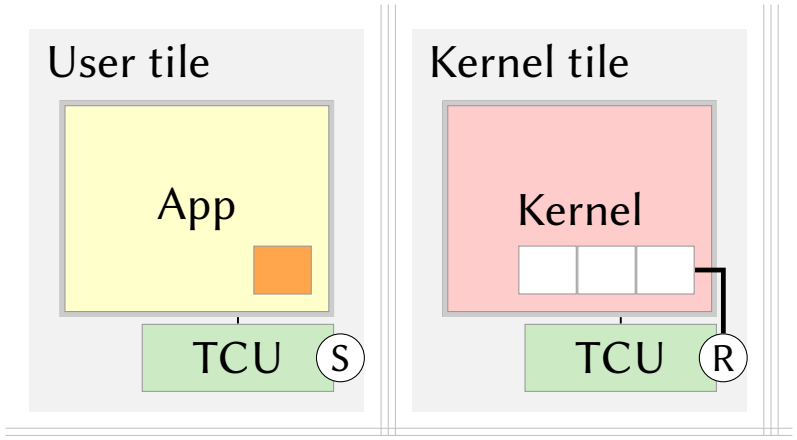
M³ System Call



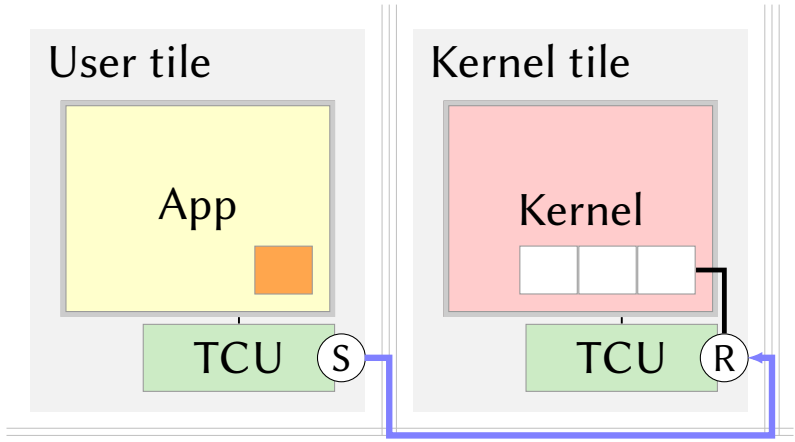
M³ System Call



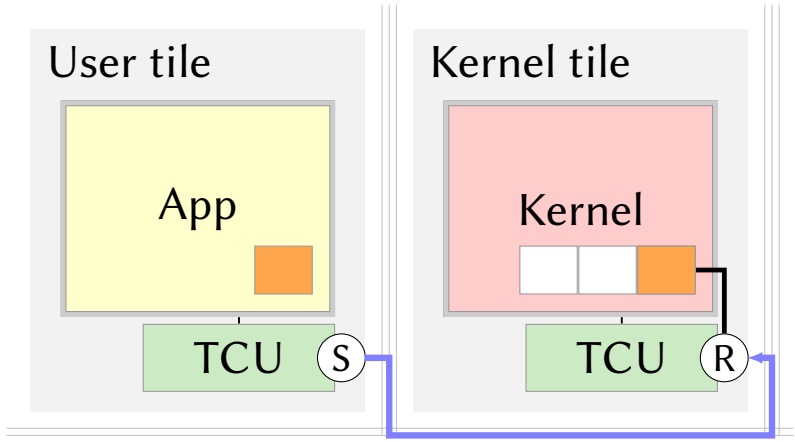
M³ System Call



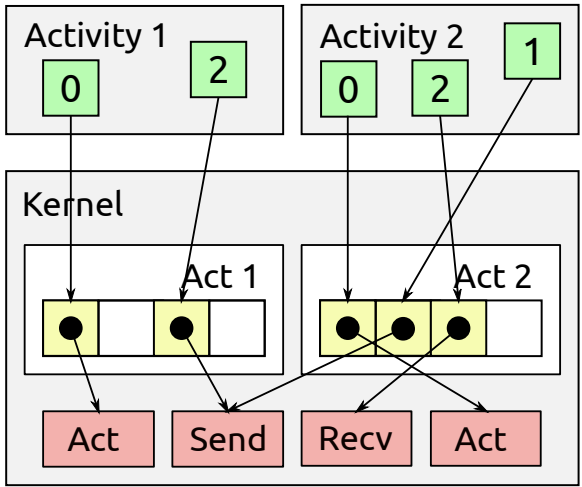
M³ System Call



M³ System Call



Capabilities Overview



Capabilities in M^3

- 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

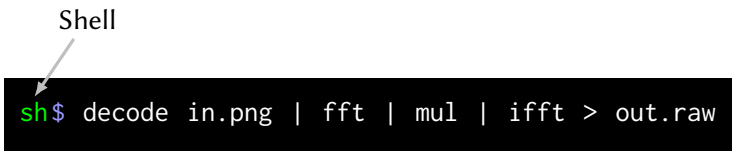
- Kernel provides syscalls to create, exchange, and revoke caps
- There are two ways to exchange caps:
  1. Directly with another activity (typically, a child activity)
  2. Over a session with a service
- The kernel offers two operations:
  1. Delegate: send capability to somebody else
  2. Obtain: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

Outline

OS Service Access for all Processing Element Types

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

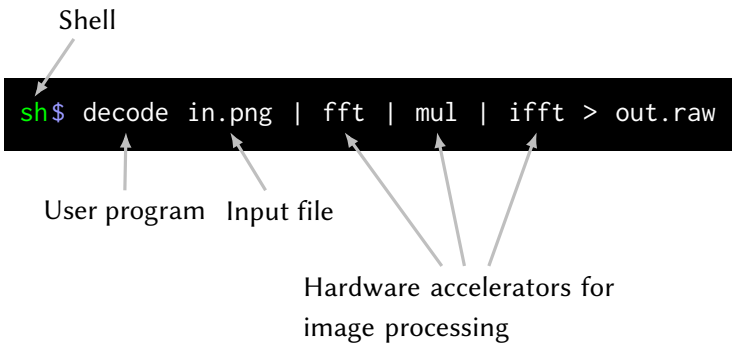
OS Service Access for all Processing Element Types



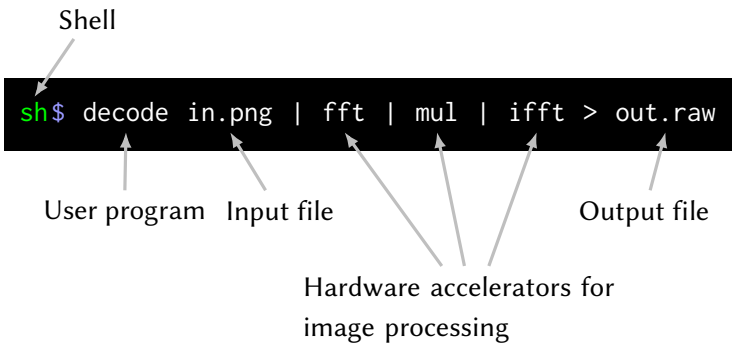
OS Service Access for all Processing Element Types

OS Service Access for all Processing Element Types

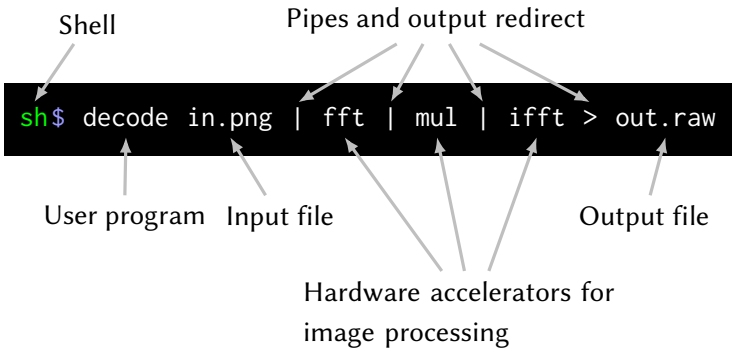
OS Service Access for all Processing Element Types



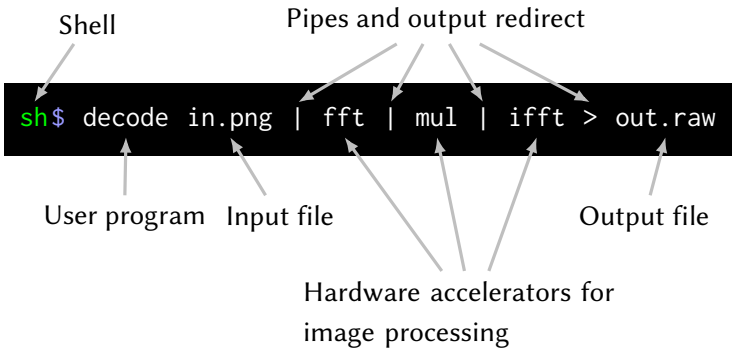
OS Service Access for all Processing Element Types



OS Service Access for all Processing Element Types

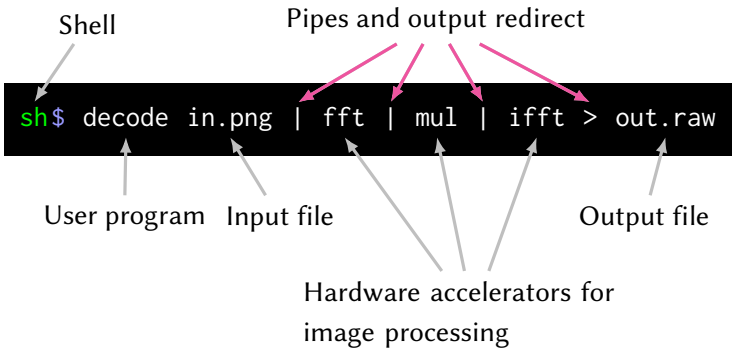


OS Service Access for all Processing Element Types



Challenges:

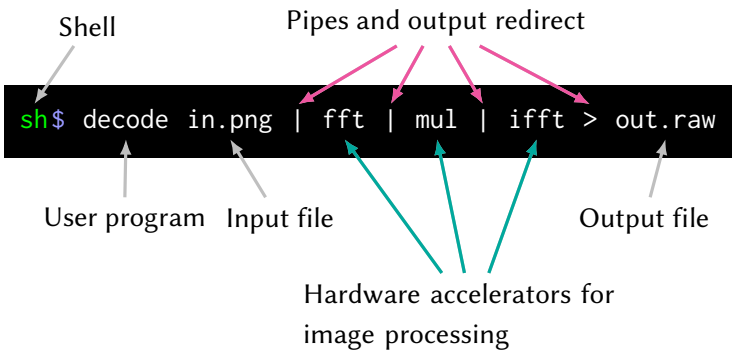
OS Service Access for all Processing Element Types



Challenges:

- OS must provide generic protocols

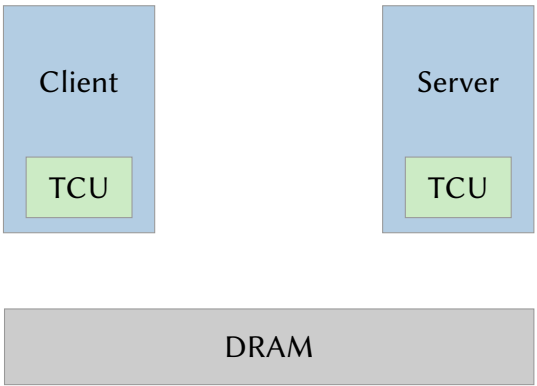
OS Service Access for all Processing Element Types



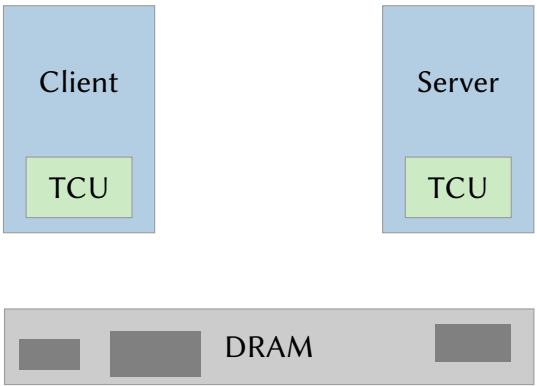
Challenges:

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

Generic Protocols



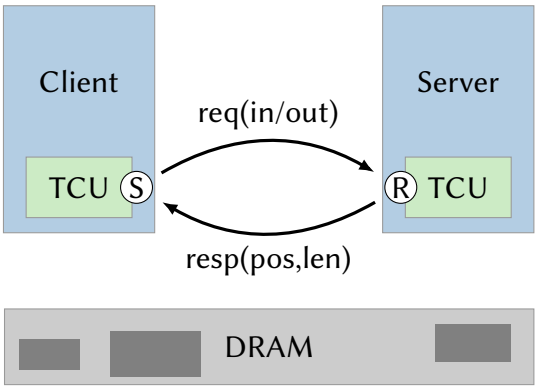
Generic Protocols



File protocol:

- Data in memory

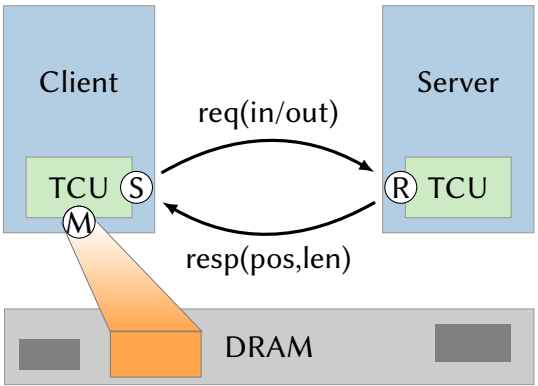
Generic Protocols



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

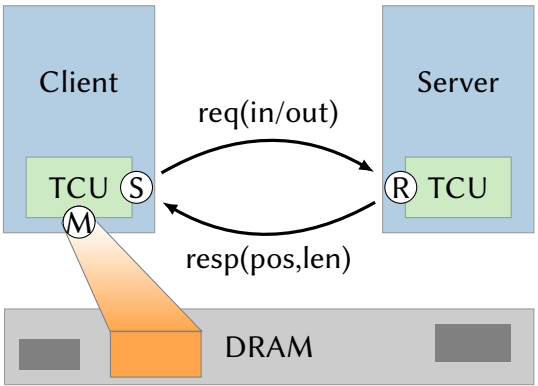
Generic Protocols



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

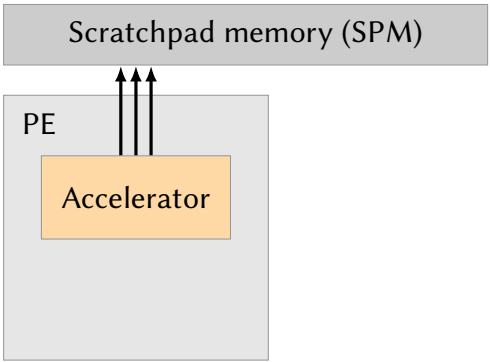
Implementation: M³FS – Overview

- M³FS organizes the file's data in extents
- 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: M^{3}FS – File Protocol

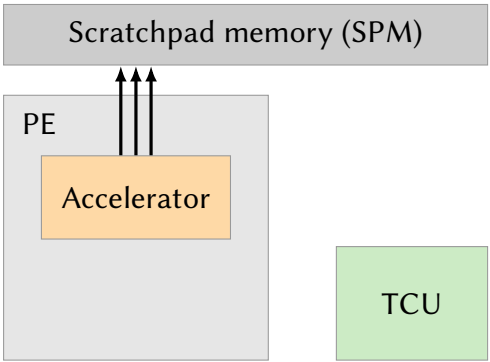
- The file session implements the file protocol (plus seeking)
- File session holds file position and advances it on read/write
- `req(in/out)` request next extent
- M^{3}FS configures client's EP for this extent
- Appending reserves new space, invisible to other clients
- `commit(nbytes)` commits a previous append

Additions to Accelerator



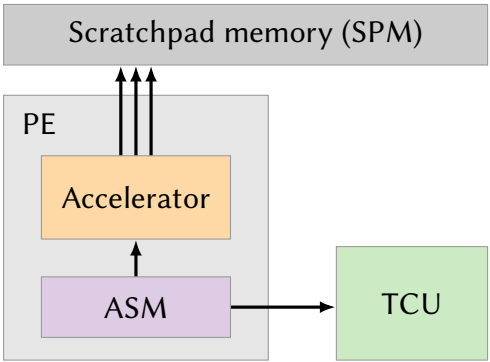
Off-the-shelf accelerators

Additions to Accelerator



Off-the-shelf accelerators

Additions to Accelerator

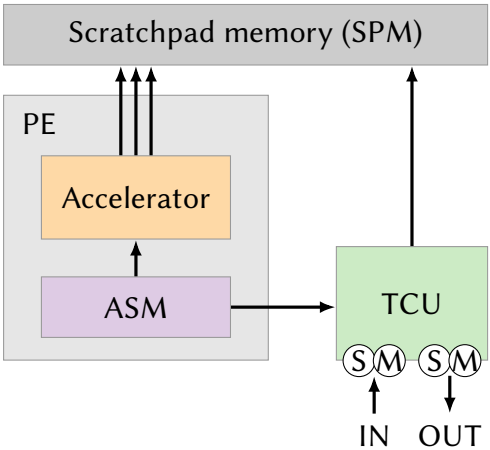


Off-the-shelf accelerators

Accelerator Support Module (ASM):

- Interacts with TCU and accelerator

Additions to Accelerator

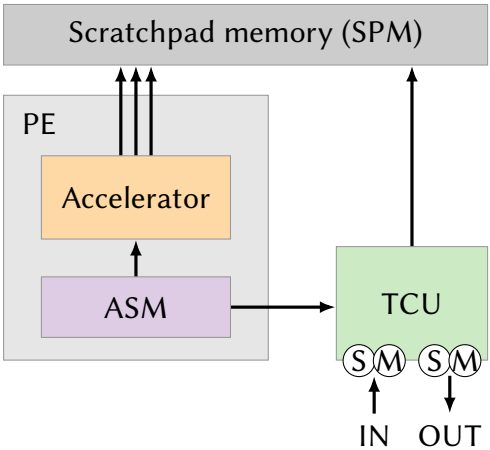


Off-the-shelf accelerators

Accelerator Support Module (ASM):

- Interacts with TCU and accelerator
- Implements file protocol for input and output channel

Additions to Accelerator



Off-the-shelf accelerators

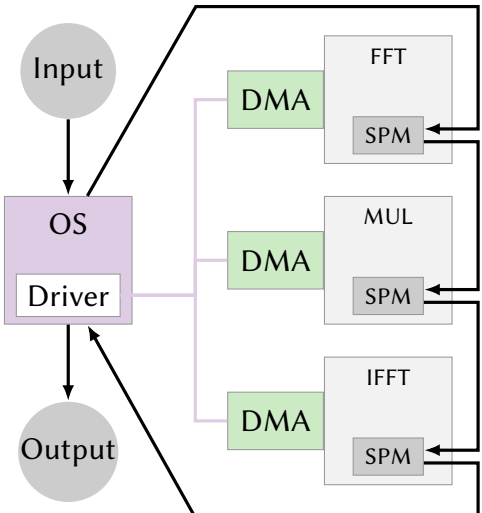
Accelerator Support Module (ASM):

- Interacts with TCU and accelerator
- Implements file protocol for input and output channel
- ASM assumes that endpoints are setup externally by software

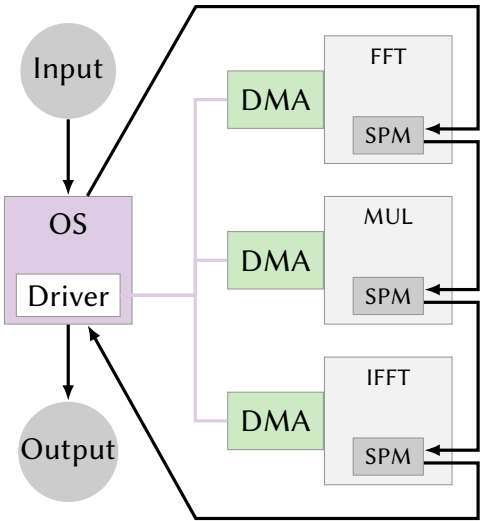
Demo

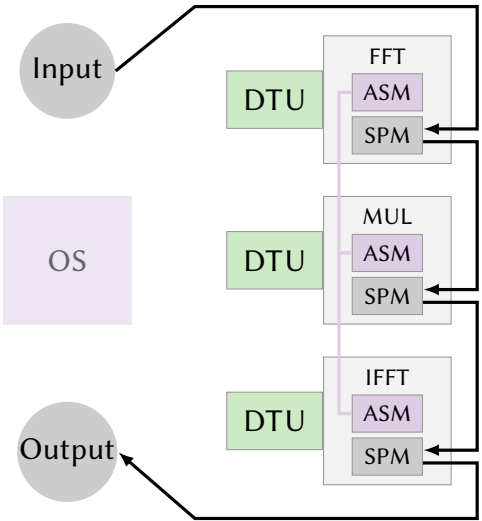
Outline

Assisted vs. Autonomous

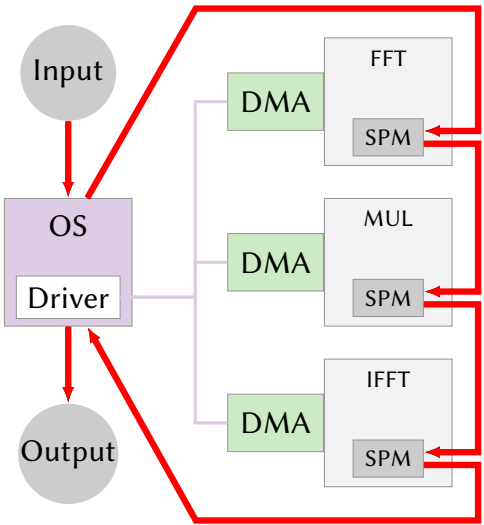


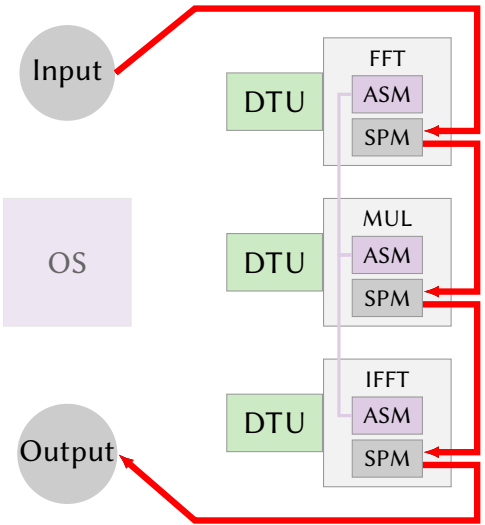
Assisted vs. Autonomous



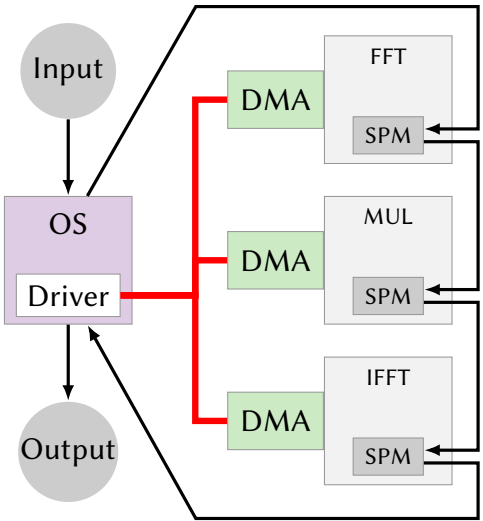


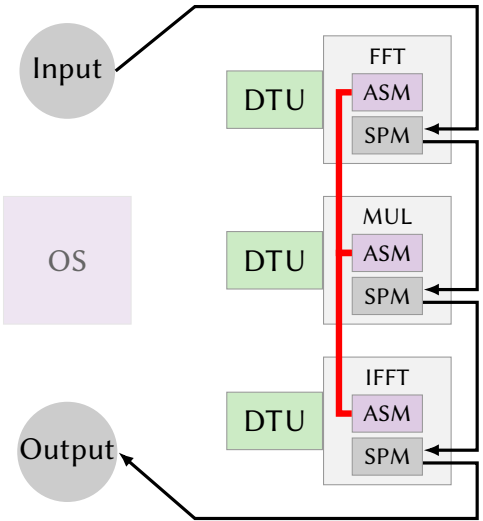
Assisted vs. Autonomous





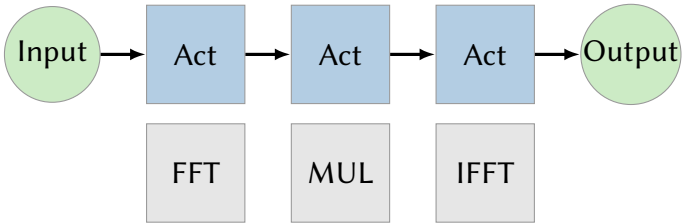
Assisted vs. Autonomous



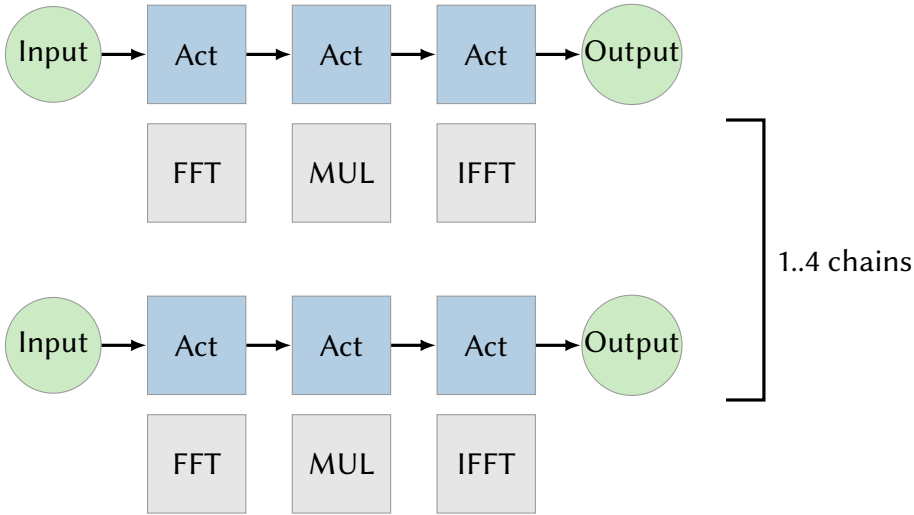


Accelerator Chains

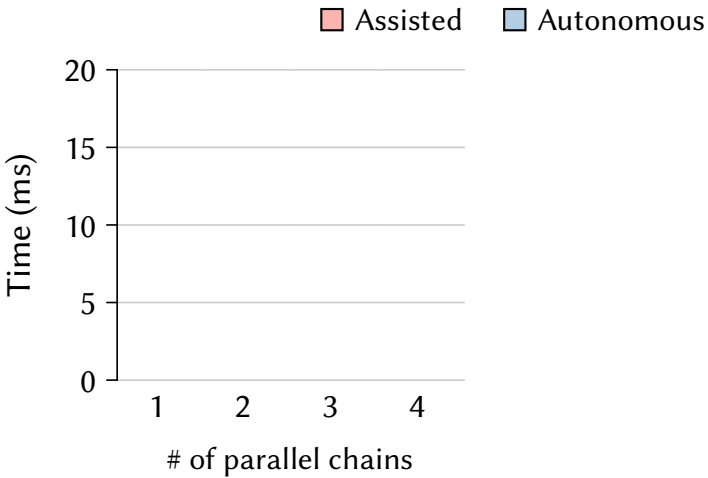
Accelerator Chains



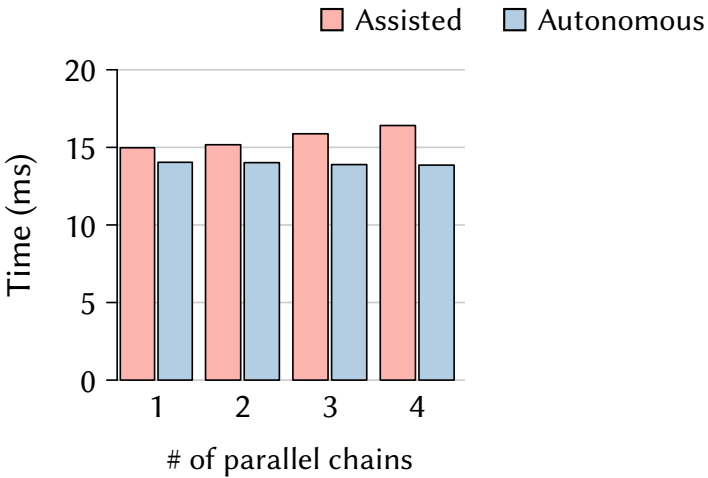
Accelerator Chains



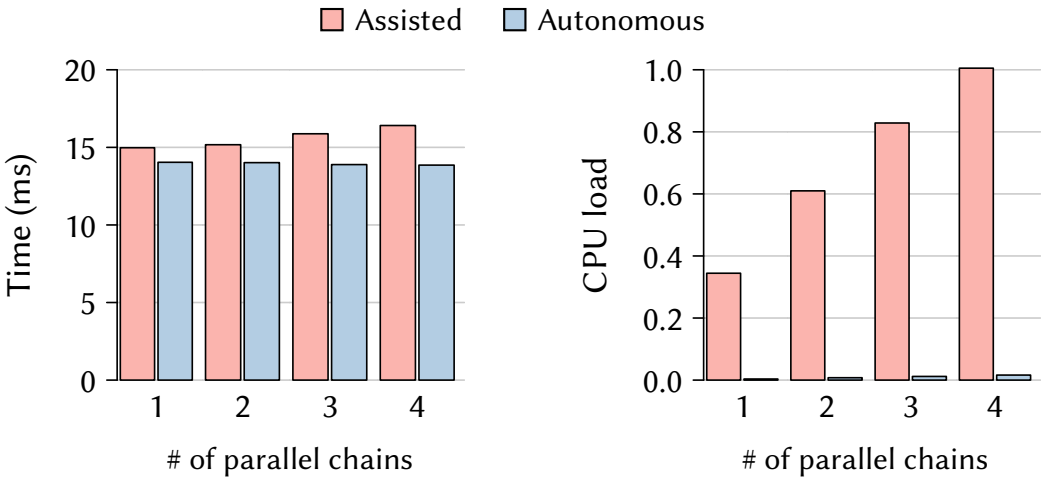
Accelerator Chains: Results



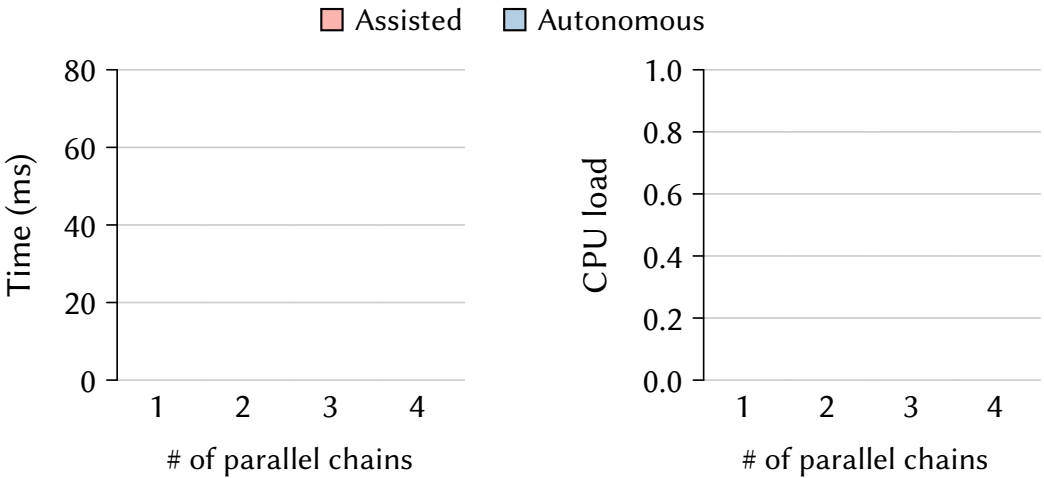
Accelerator Chains: Results



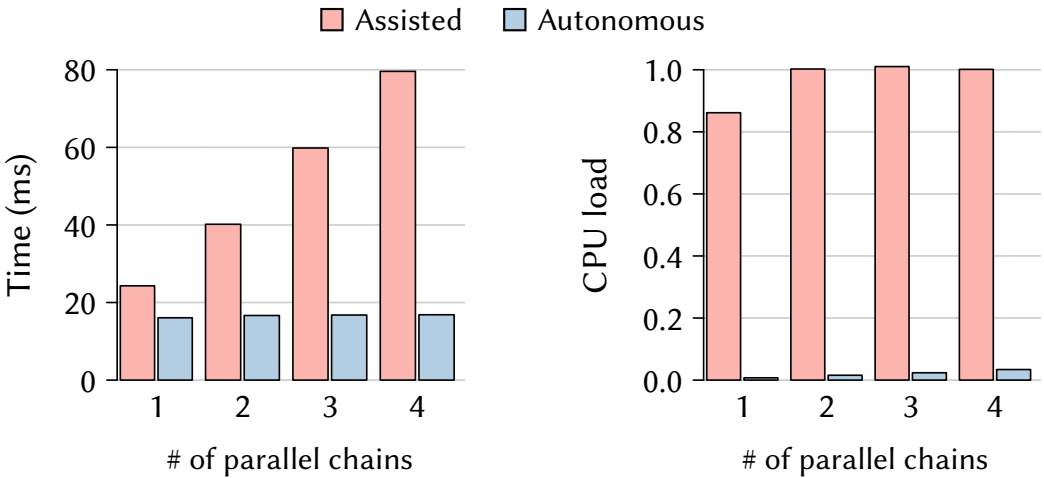
Accelerator Chains: Results



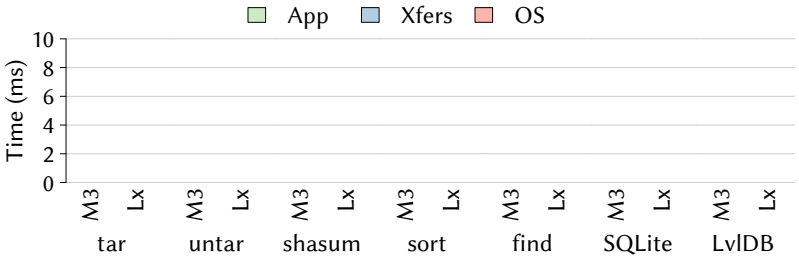
Accelerator Chains: Results (PCIe-like Latency)



Accelerator Chains: Results (PCIe-like Latency)

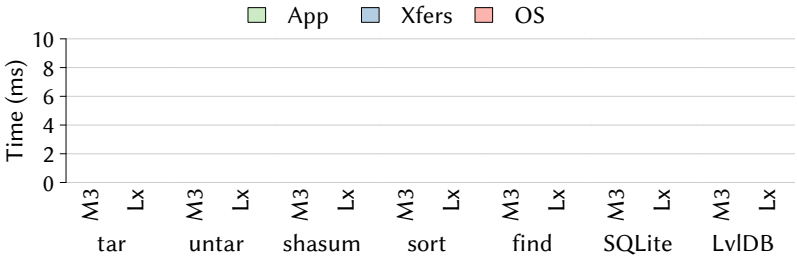


Linux Application Workloads

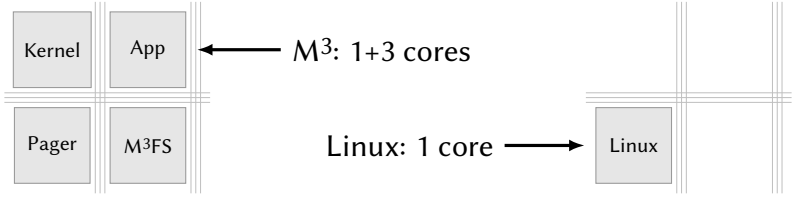


- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs

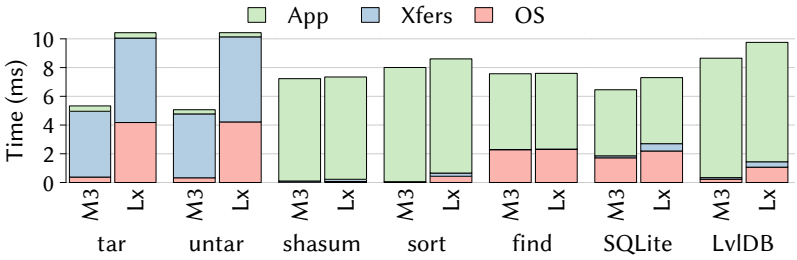
Linux Application Workloads



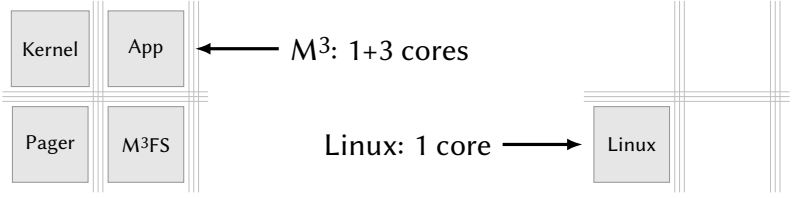
- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs



Linux Application Workloads



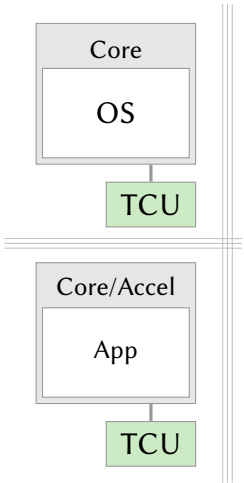
- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs



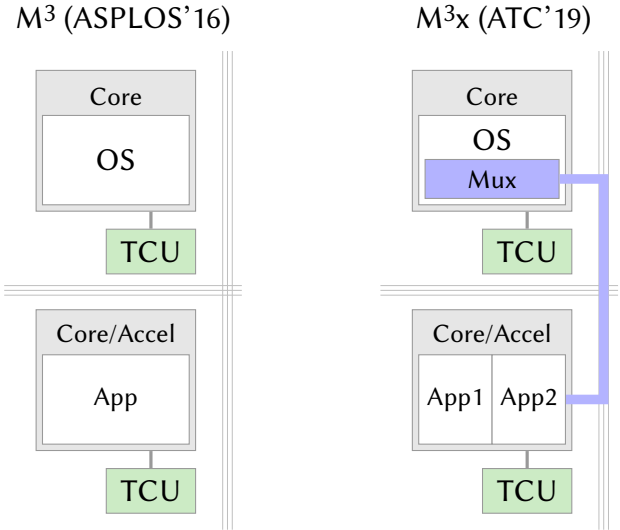
Outline

Comparison of Context-Switching Approaches

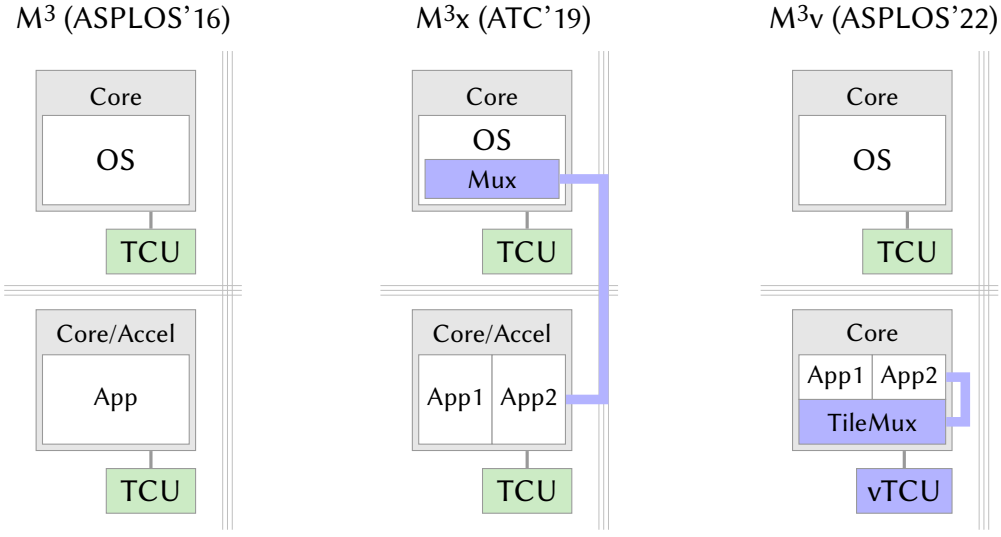
M^3 (ASPLOS'16)



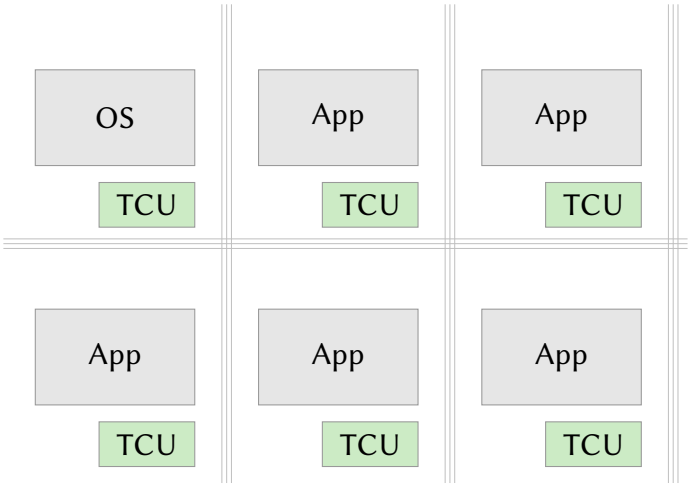
Comparison of Context-Switching Approaches



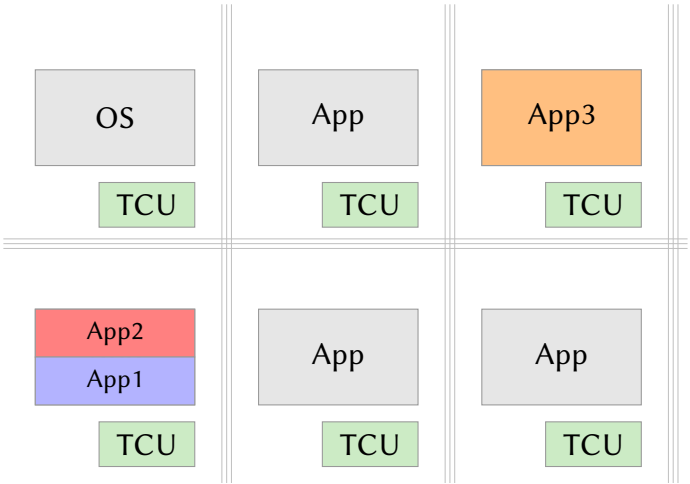
Comparison of Context-Switching Approaches



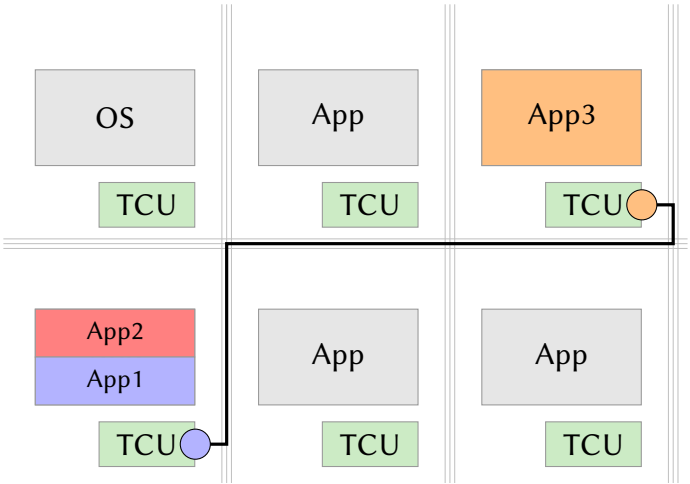
Context Switching vs. Fast-Path Communication



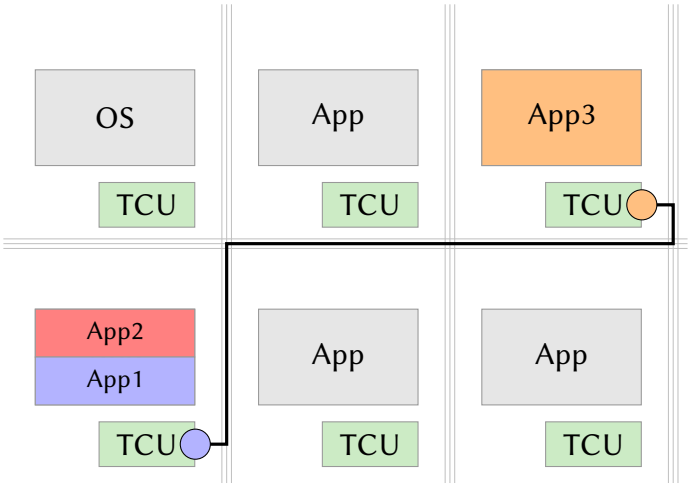
Context Switching vs. Fast-Path Communication



Context Switching vs. Fast-Path Communication

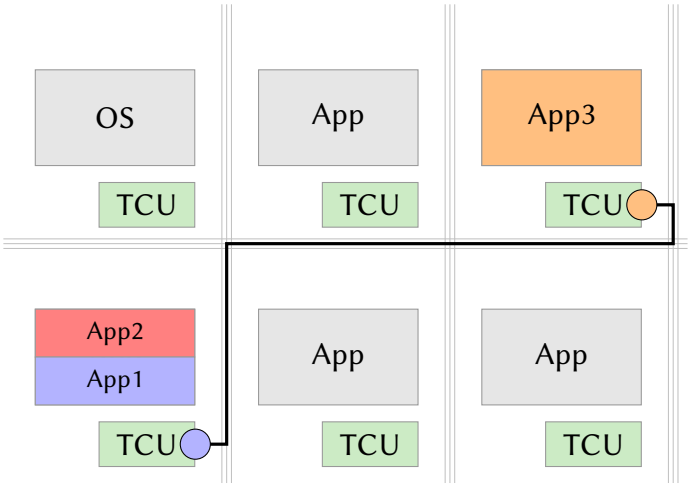


Context Switching vs. Fast-Path Communication



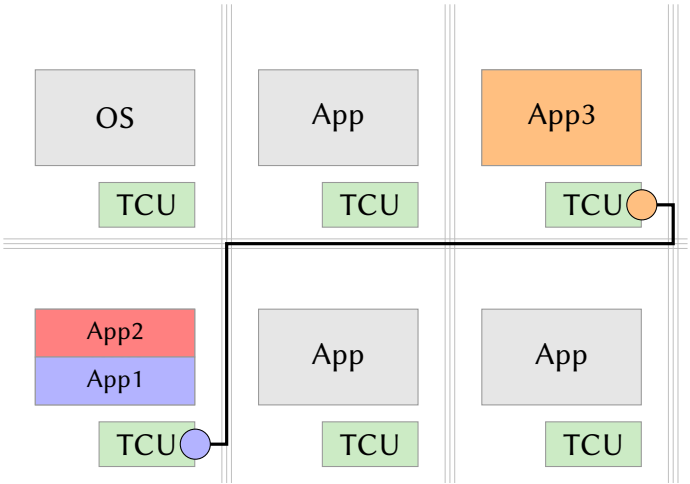
- Suspend App1 until new message, schedule App2

Context Switching vs. Fast-Path Communication



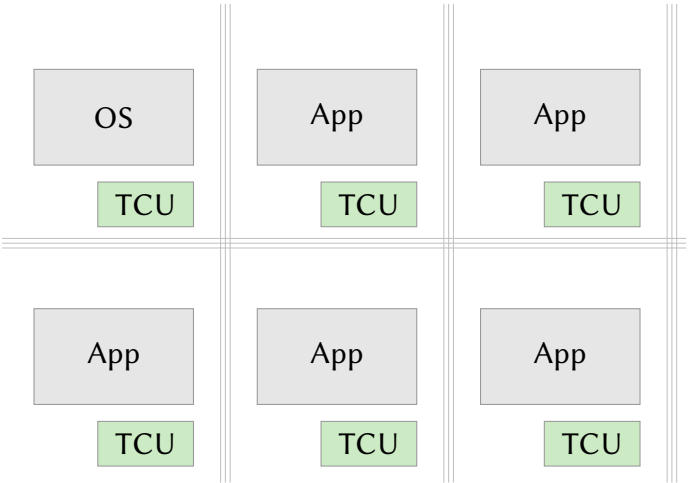
- Suspend App1 until new message, schedule App2
- Resume App1 upon new message

Context Switching vs. Fast-Path Communication

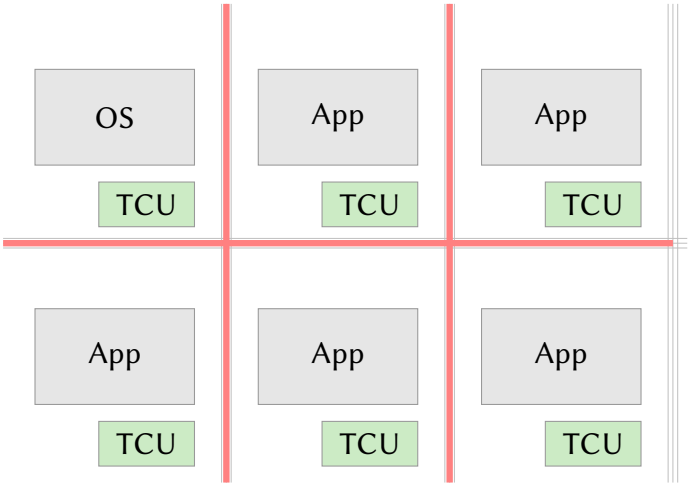


- Suspend App1 until new message, schedule App2
- Resume App1 upon new message
- Multiplexing conflicts with fast-path communication

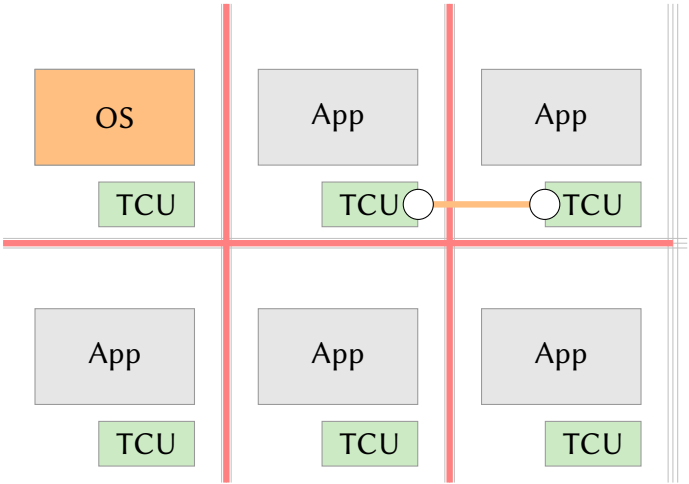
Context Switching vs. Isolation



Context Switching vs. Isolation

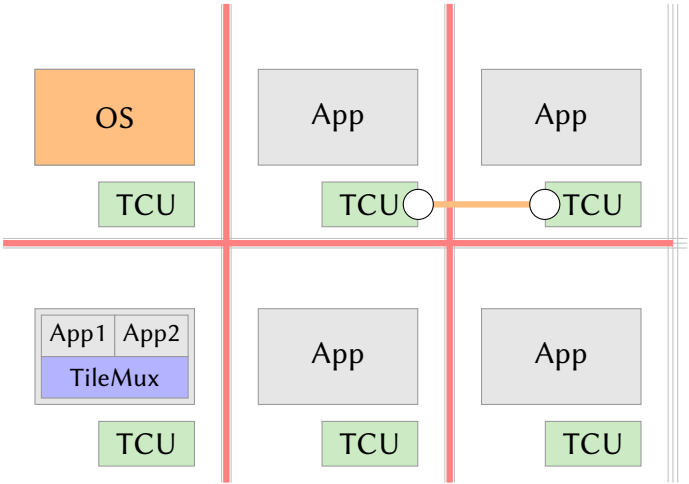


Context Switching vs. Isolation



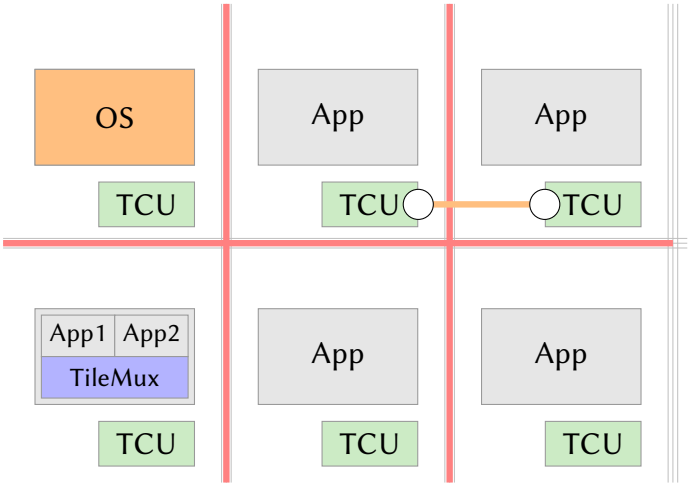
- Only the OS can provide access to tile-external resources

Context Switching vs. Isolation



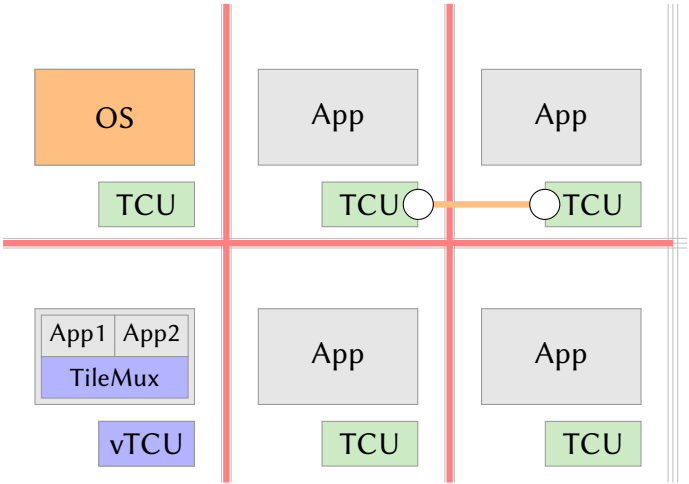
- Only the OS can provide access to tile-external resources
- Restoring TCU state provides access to **all** resources

Context Switching vs. Isolation



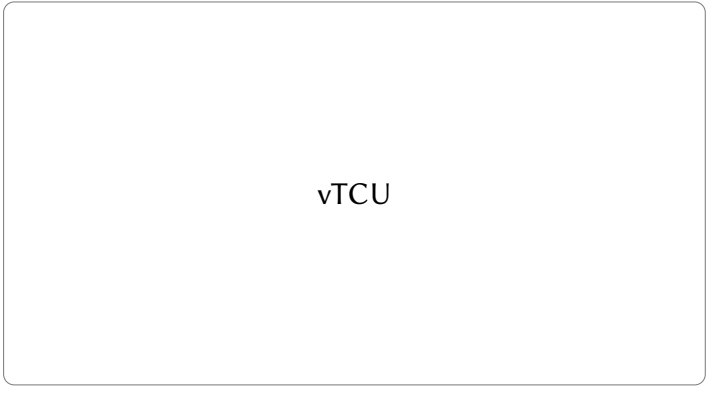
- Only the OS can provide access to tile-external resources
- Restoring TCU state provides access to **all** resources
- TileMux **must not** restore TCU state!

Context Switching vs. Isolation

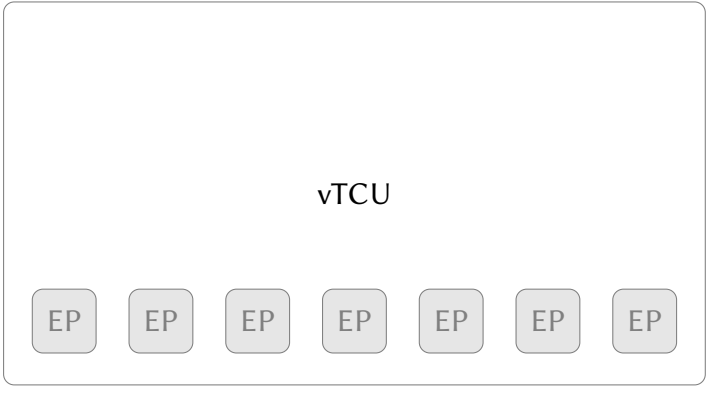


- Only the OS can provide access to tile-external resources
- Restoring TCU state provides access to **all** resources
- TileMux **must not** restore TCU state!

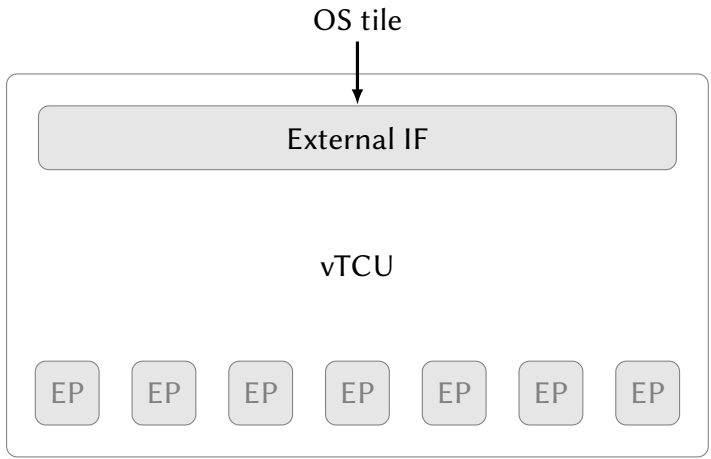
Virtualization of the TCU



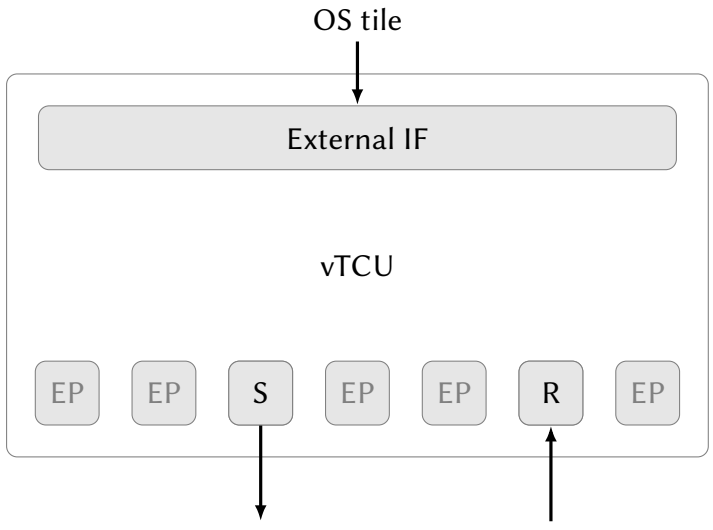
Virtualization of the TCU



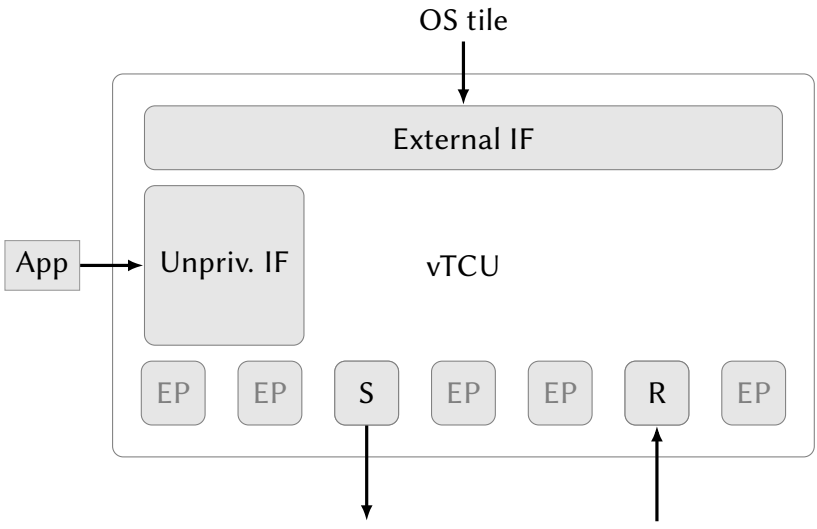
Virtualization of the TCU



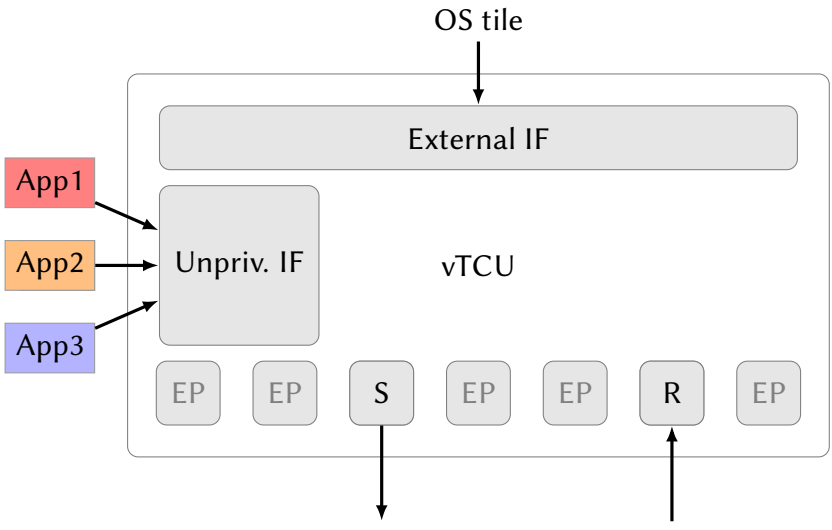
Virtualization of the TCU



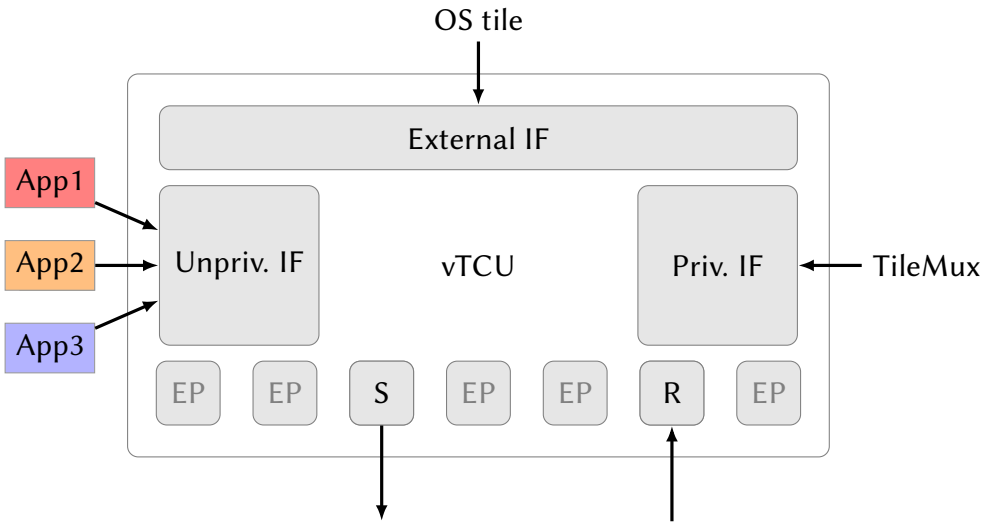
Virtualization of the TCU



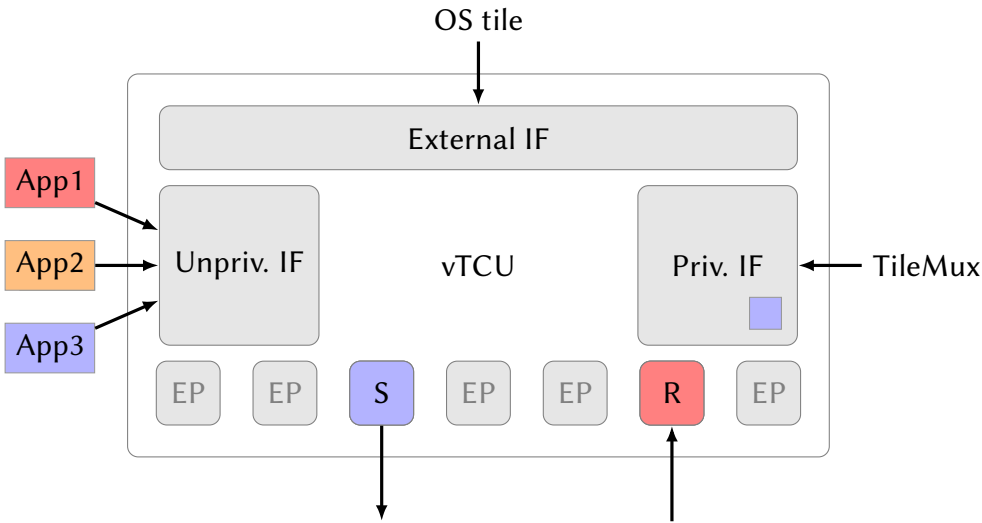
Virtualization of the TCU



Virtualization of the TCU



Virtualization of the TCU



vTCU Size and Complexity

	LUTs [k]	FFs [k]	BRAMs
BOOM	143.8	71.8	159
Rocket	46.6	22.0	152
NoC router	3.4	2.2	0
vTCU	15.2	5.8	0.5
Control Unit	10.3	3.3	0.5
NoC CTRL	3.2	1.5	0
CMD CTRL	7.1	2.8	0.5
Unpriv. IF	6.2	2.5	0.5
Priv. IF	0.9	0.3	0
Register file	2.0	1.0	0
Memory mapper + PMP	0.6	0.2	0
I/O FIFOs	2.3	0.3	0

vTCU Size and Complexity

	LUTs [k]	FFs [k]	BRAMs
BOOM	143.8	71.8	159
Rocket	46.6	22.0	152
NoC router	3.4	2.2	0
vTCU	15.2	5.8	0.5
Control Unit	10.3	3.3	0.5
NoC CTRL	3.2	1.5	0
CMD CTRL	7.1	2.8	0.5
Unpriv. IF	6.2	2.5	0.5
Priv. IF	0.9	0.3	0
Register file	2.0	1.0	0
Memory mapper + PMP	0.6	0.2	0
I/O FIFOs	2.3	0.3	0

vTCU Size and Complexity

	LUTs [k]	FFs [k]	BRAMs
BOOM	143.8	71.8	159
Rocket	46.6	22.0	152
NoC router	3.4	2.2	0
vTCU	15.2	5.8	0.5
Control Unit	10.3	3.3	0.5
NoC CTRL	3.2	1.5	0
CMD CTRL	7.1	2.8	0.5
Unpriv. IF	6.2	2.5	0.5
Priv. IF	0.9	0.3	0
Register file	2.0	1.0	0
Memory mapper + PMP	0.6	0.2	0
I/O FIFOs	2.3	0.3	0

vTCU Size and Complexity

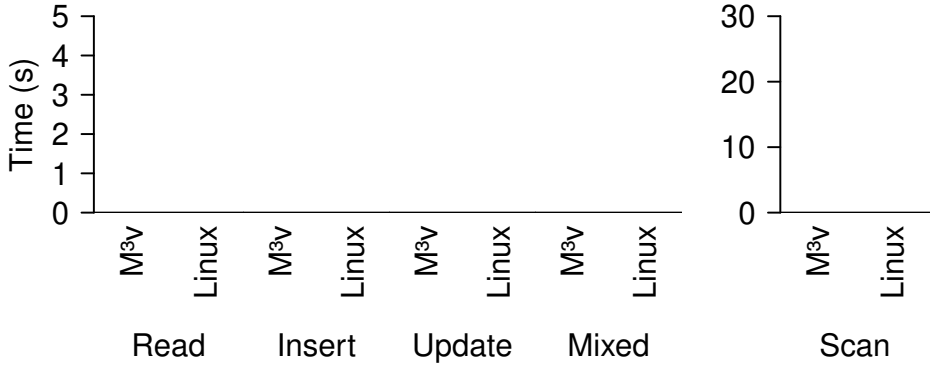
	LUTs [k]	FFs [k]	BRAMs
BOOM	143.8	71.8	159
Rocket	46.6	22.0	152
NoC router	3.4	2.2	0
vTCU	15.2	5.8	0.5
Control Unit	10.3	3.3	0.5
NoC CTRL	3.2	1.5	0
CMD CTRL	7.1	2.8	0.5
Unpriv. IF	6.2	2.5	0.5
Priv. IF	0.9	0.3	0
Register file	2.0	1.0	0
Memory mapper + PMP	0.6	0.2	0
I/O FIFOs	2.3	0.3	0

Performance Comparison with Linux

- 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

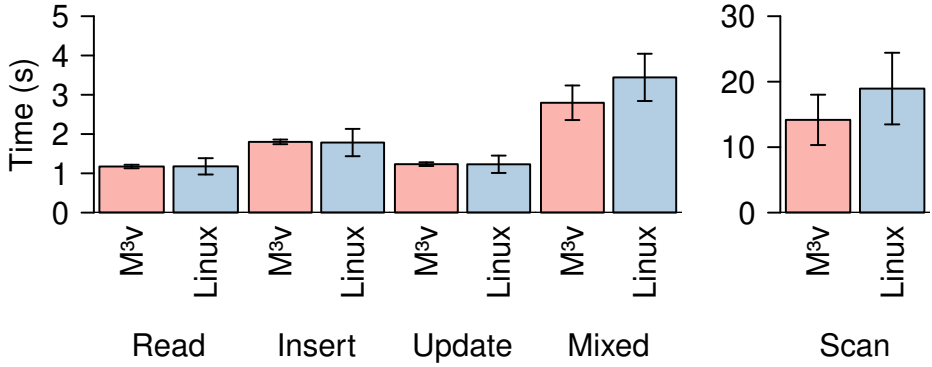
Performance Comparison with Linux

- 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 Linux

- 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



Ongoing Work at the Barkhausen Insitut

- Connected devices with remote attestation
- Turning the FPGA prototype into a silicon chip
- Providing real-time guarantees
- Running Linux on a user tile

Conclusion

- M^3 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
- General-purpose cores can be multiplexed efficiently
- Hardware implementation demonstrates modest additional cost
- Complete hardware/software stack available as open source:
  `https://github.com/Barkhausen-Institut/M3`

More Information

- **Core-Local Reasoning and Predictable Cross-Core Communication with M³**
  Nils Asmussen, Sebastian Haas, Adam Lackorzyński, Michael Roitzsch
  RTAS 2024
- **Efficient and Scalable Core Multiplexing with M³v**
  Nils Asmussen, Sebastian Haas, Carsten Weinhold, Till Miemietz, Michael Roitzsch
  ASPLOS 2022
- **M³x: Autonomous Accelerators via Context-Enabled Fast-Path Communication**
  Nils Asmussen, Michael Roitzsch, and Hermann Härtig
  USENIX ATC 2019
- **SemperOS: Distributed Capability System**
  Matthias Hille, Nils Asmussen, Pramod Bhatotia, and Hermann Härtig
  USENIX ATC 2019
- **M³: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores**
  Nils Asmussen, Marcus Völp, Benedikt Nöthen, Hermann Härtig, and Gerhard Fettweis
  ASPLOS 2016