Microkernel Construction

Case Study: M³

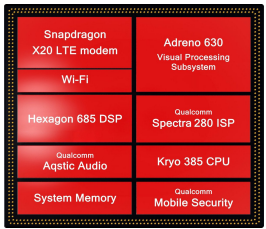
Nils Asmussen

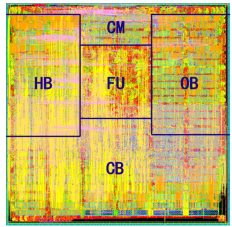
May 28th 2020

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

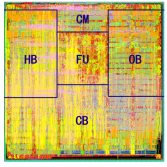
Heterogeneous Systems

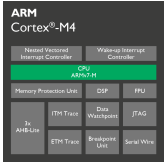




- Demanded by performance and energy requirements
- Big challenge for OSes: single shared kernel on all cores does no longer work
- OSes need to be prepared for processing elements with different feature sets

Third-party Components



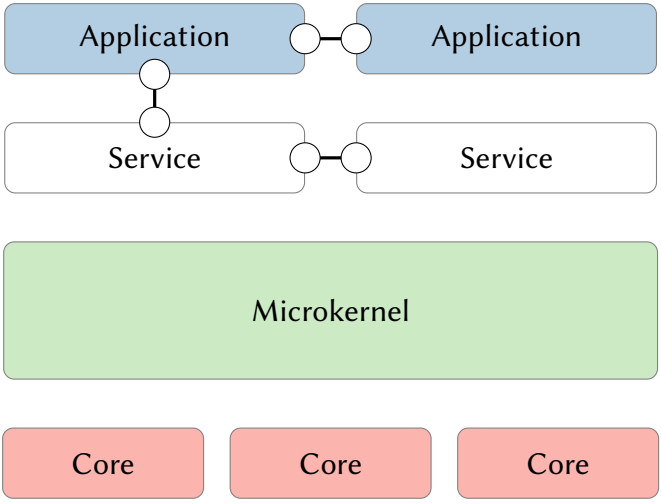


- Market pressure forces us to integrate third-party components
- We should not trust these components
- Currently, often no isolation between them
- Bug in such a component can compromise whole system (see Broadcom incident)

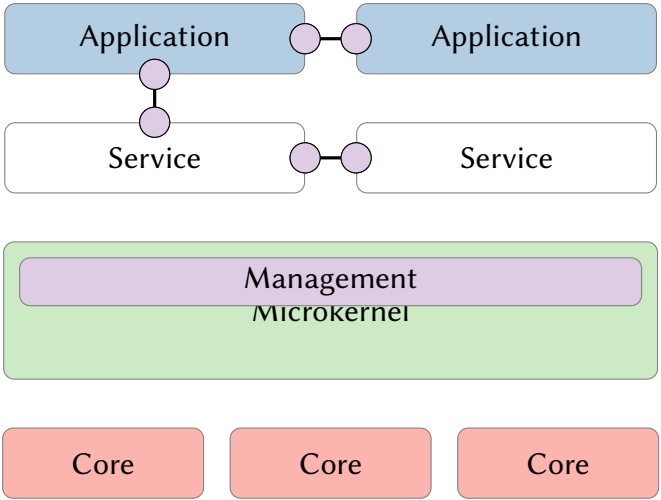
Security Issues of Complex General-purpose Cores

- 20 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?

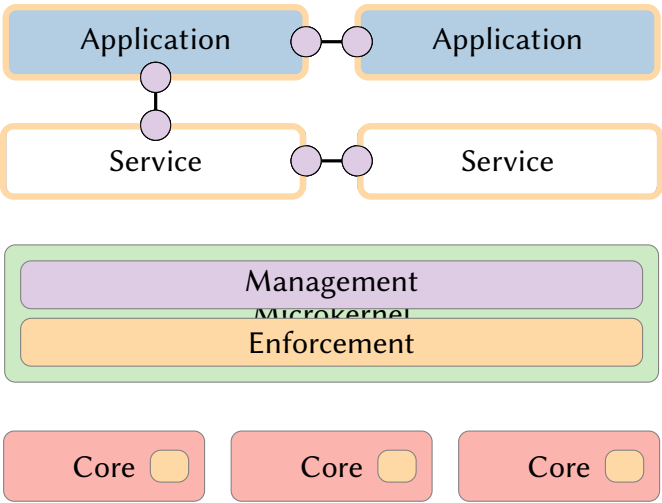
Microkernel-based System as Foundation



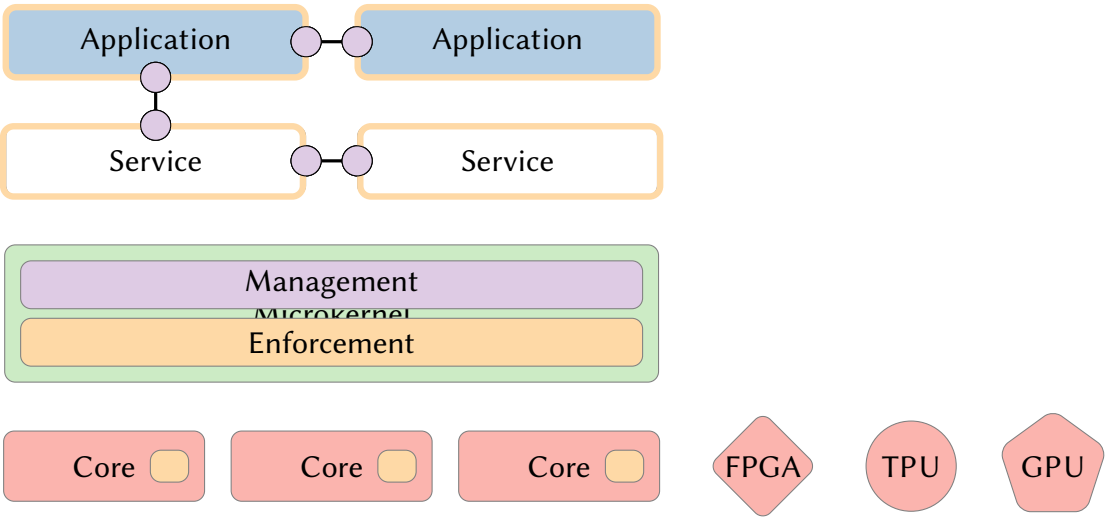
Microkernel-based System as Foundation



Microkernel-based System as Foundation



Microkernel-based System as Foundation



Outline

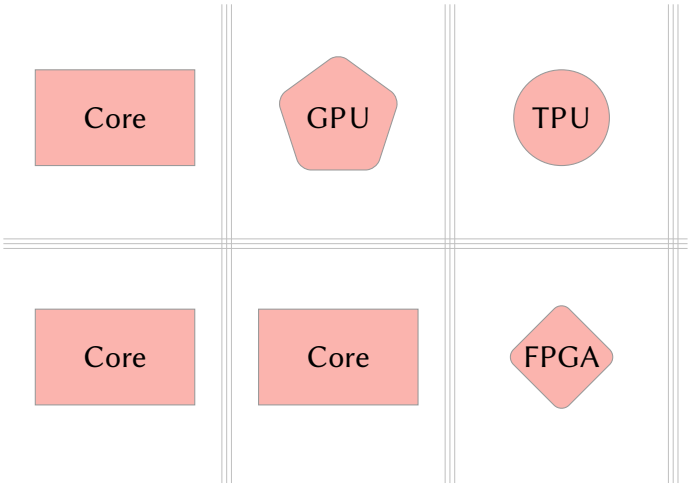
1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

Outline

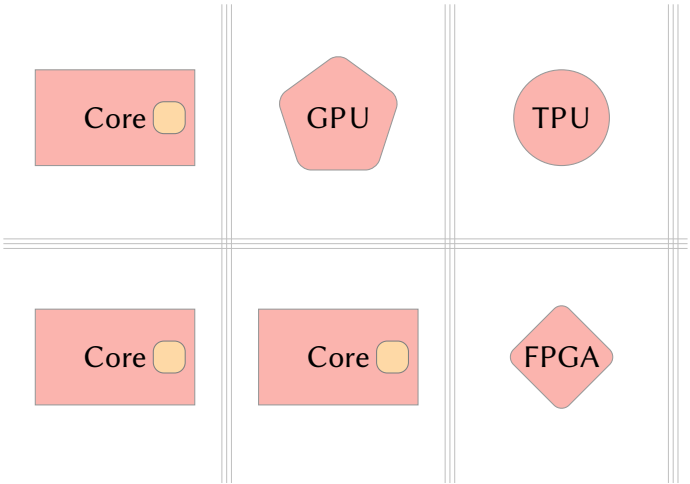
1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

Hardware/Operating System Co-Design

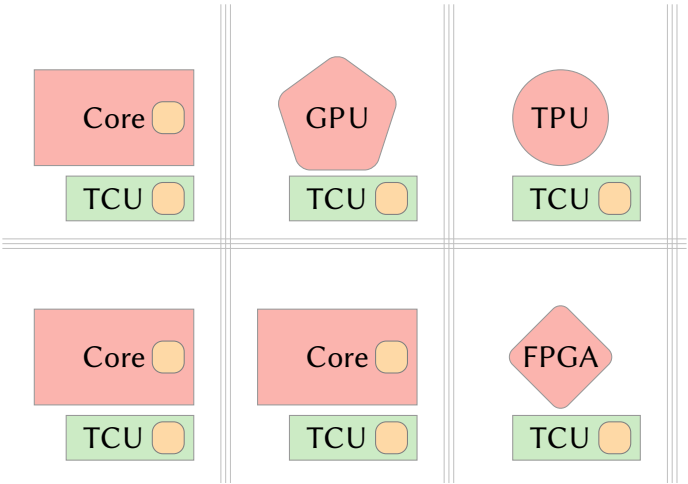
Hardware/Operating System Co-Design



Hardware/Operating System Co-Design



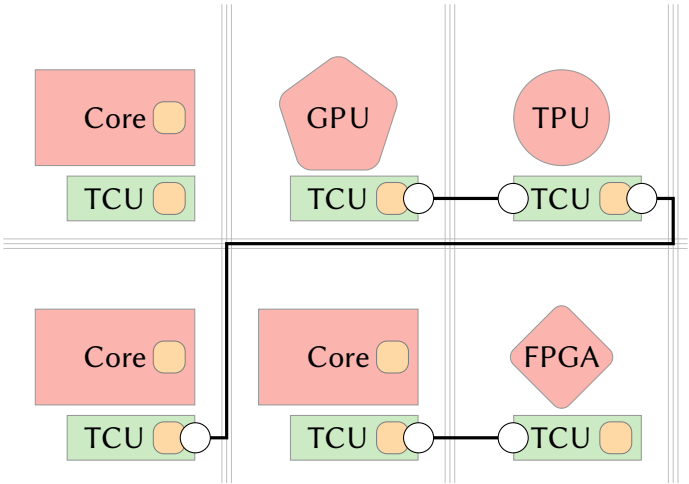
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component

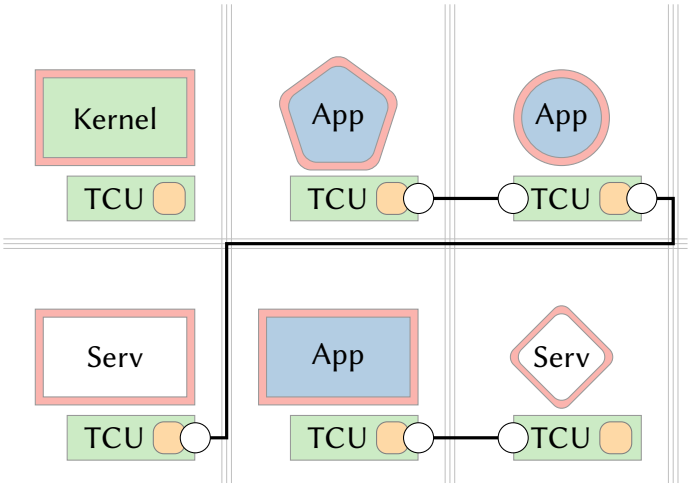
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component

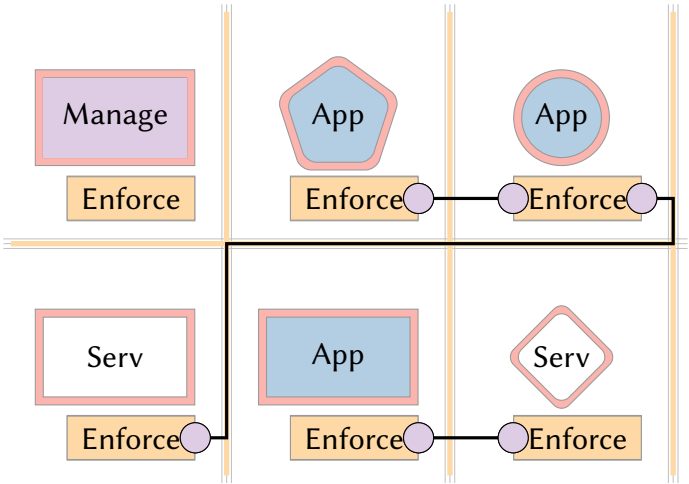
Hardware/Operating System Co-Design



Key ideas:

- TCU as new hardware component
- Kernel on dedicated tile

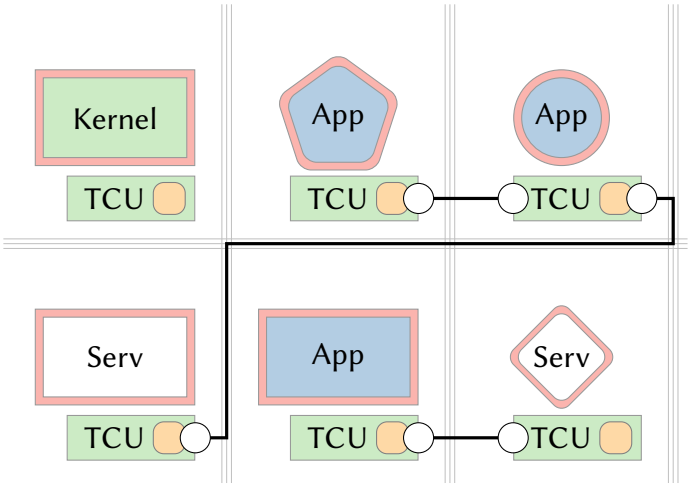
Hardware/Operating System Co-Design



Key ideas:

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

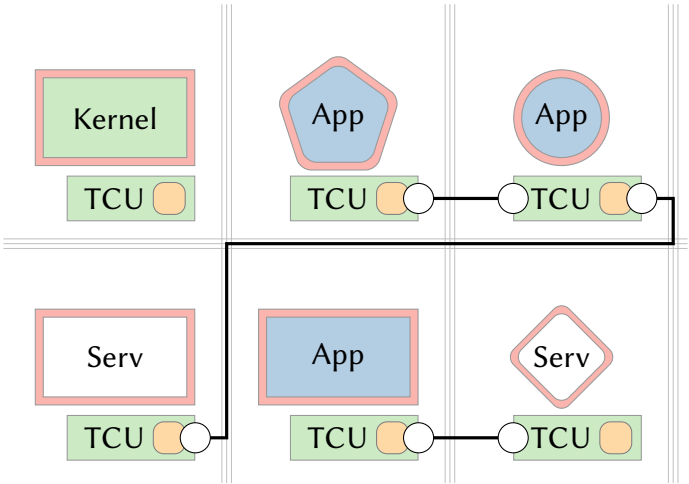
Hardware/Operating System Co-Design



Hardware challenges:

- Heterogeneity:
  Uniform interface

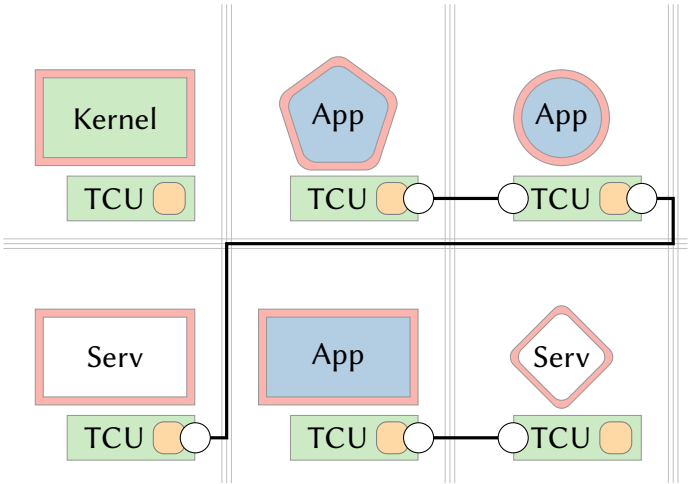
Hardware/Operating System Co-Design



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

1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

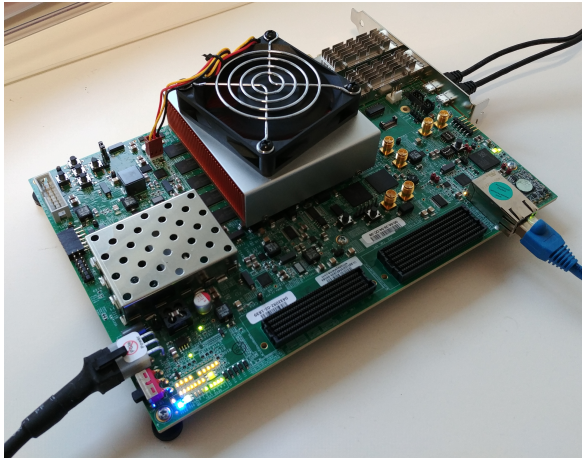
Linux

- M^3 runs on Linux using it as a virtual machine
- A process simulates a tile, having two threads (CPU + TCU)
- TCUs communicate over UNIX domain sockets
- No accuracy because
  - Programs are directly executed on host
  - Data transfers have huge overhead compared to HW
- Very useful for debugging and early prototyping

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

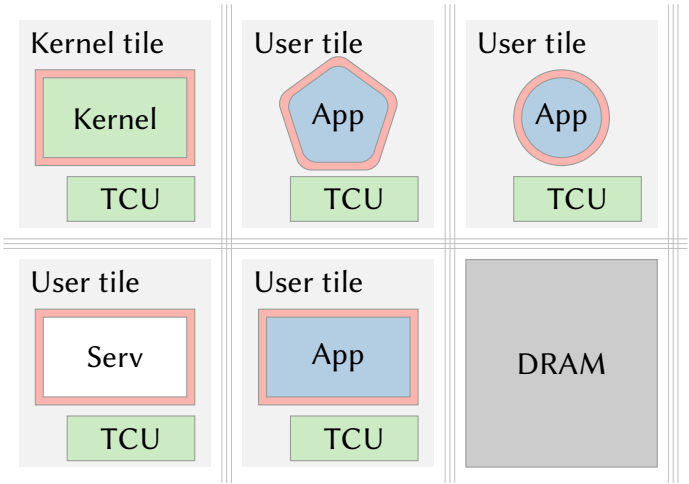
WIP: FPGA-based prototype



Outline

1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

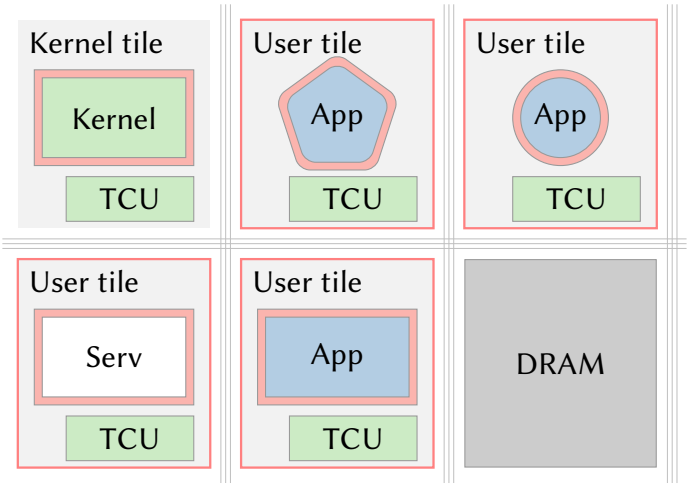
Isolation



TCU-based isolation:

- Additional protection layer

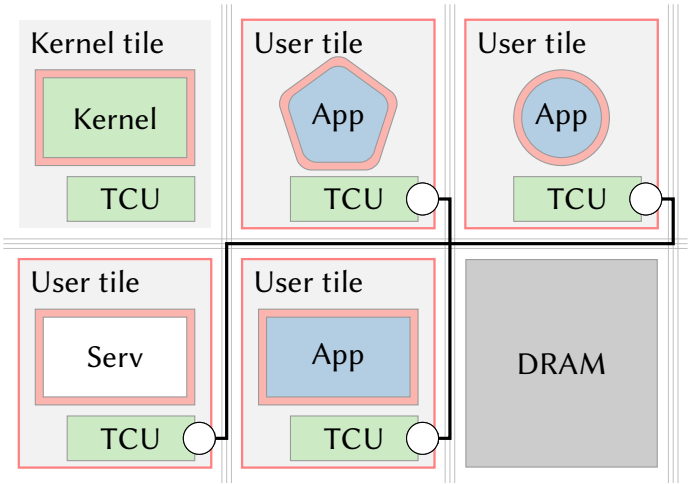
Isolation



TCU-based isolation:

- Additional protection layer

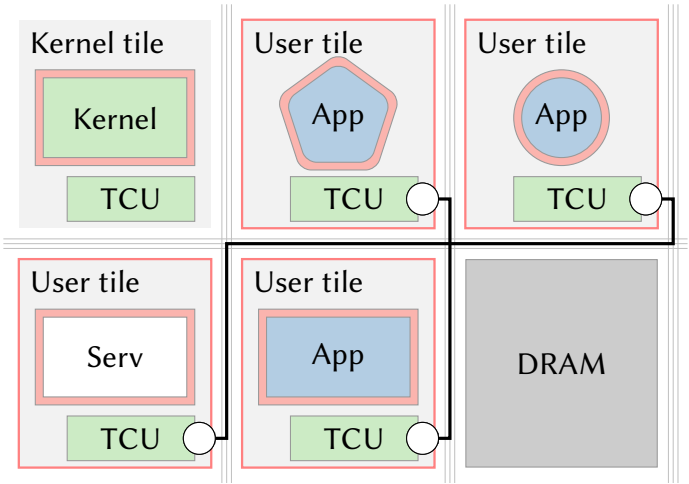
Isolation



TCU-based isolation:

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

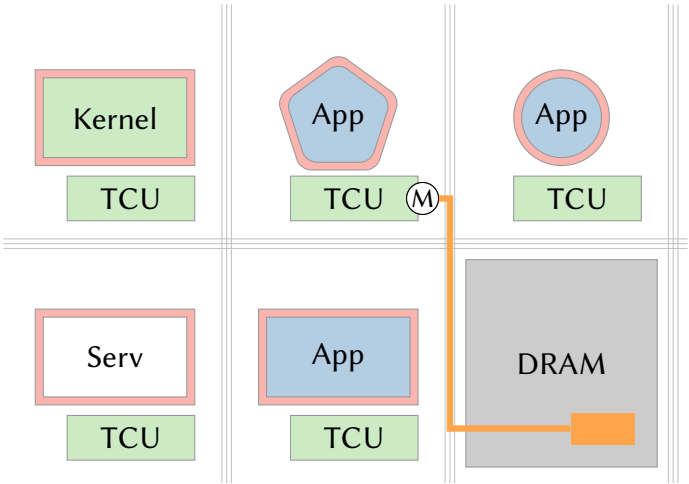
Isolation



TCU-based isolation:

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

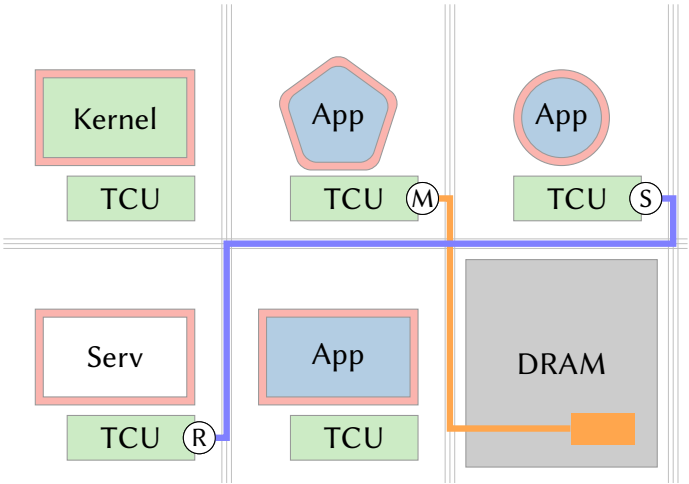
Communication



TCU provides *endpoints* to:

- Access memory (contiguous range, byte granular)

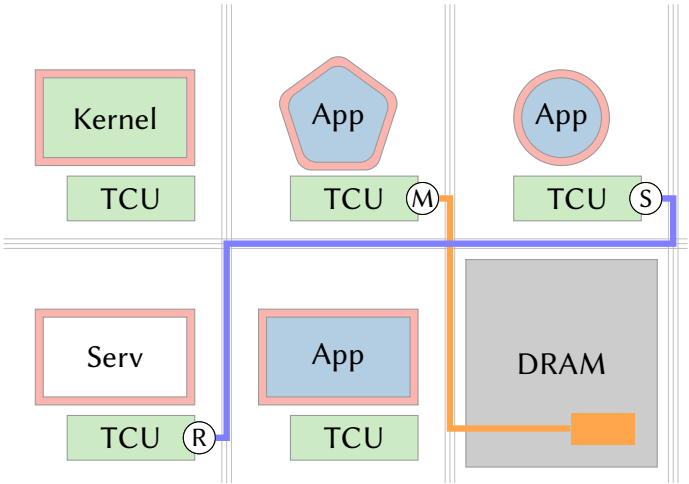
Communication



TCU provides *endpoints* to:

- Access memory (contiguous range, byte granular)
- Receive messages into a receive buffer
- Send messages to a receiving endpoint

Communication



TCU provides *endpoints* to:

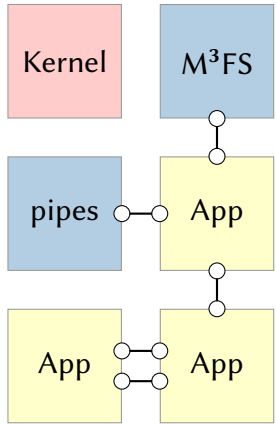
- Access memory (contiguous range, byte granular)
- Receive messages into a receive buffer
- Send messages to a receiving endpoint
- Replies for RPC

Outline

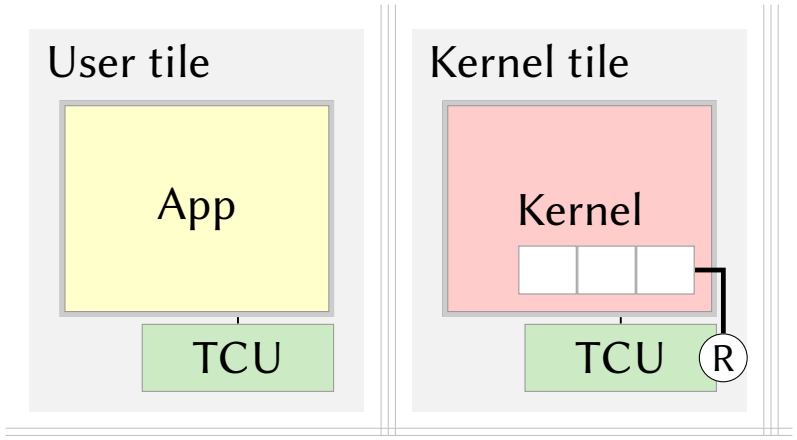
1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

OS Design

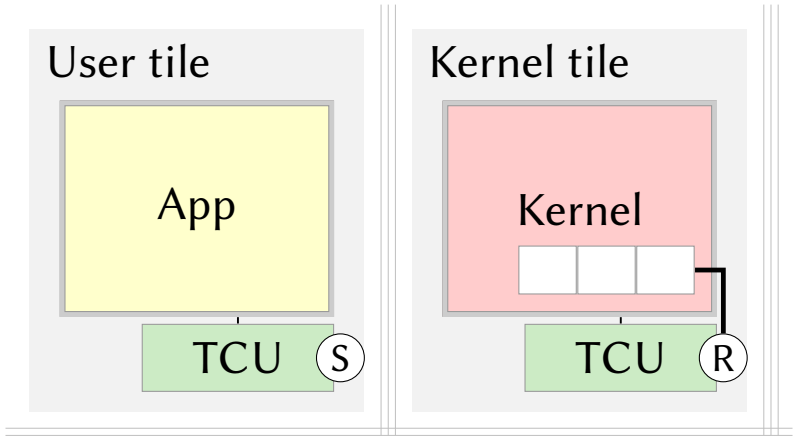
- M³: **M**icrokernel-based syste**m** for het. **m**anycores (or L4 ± 1)
- Implemented from scratch in Rust and C++
- Drivers, filesystems, etc. implemented on user tiles
- Kernel manages permissions, using capabilities
- TCU enforces permissions (communication, memory access)
- Kernel is independent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



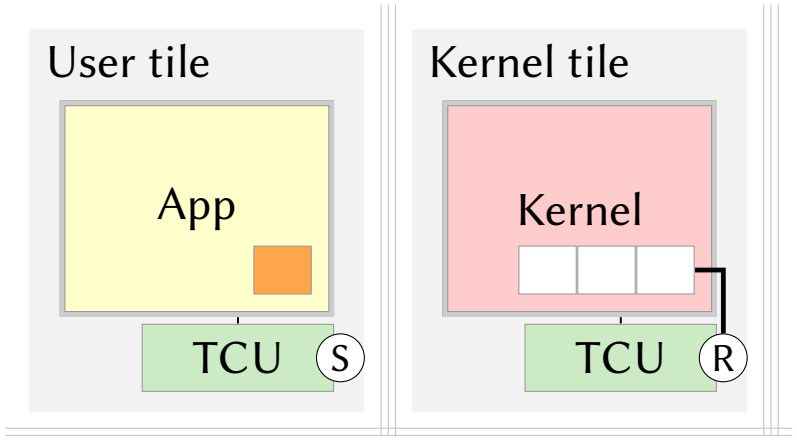
M³ System Call



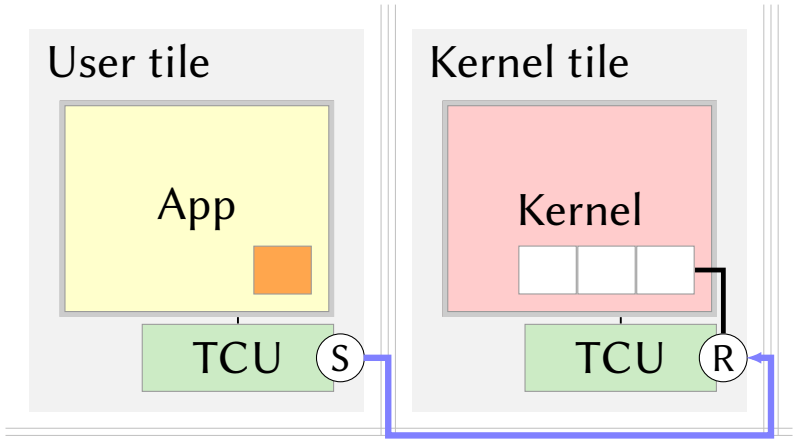
M³ System Call



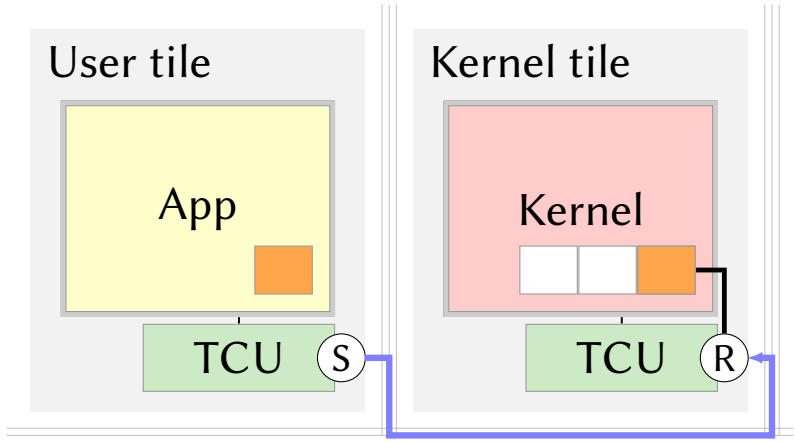
M³ System Call



M³ System Call



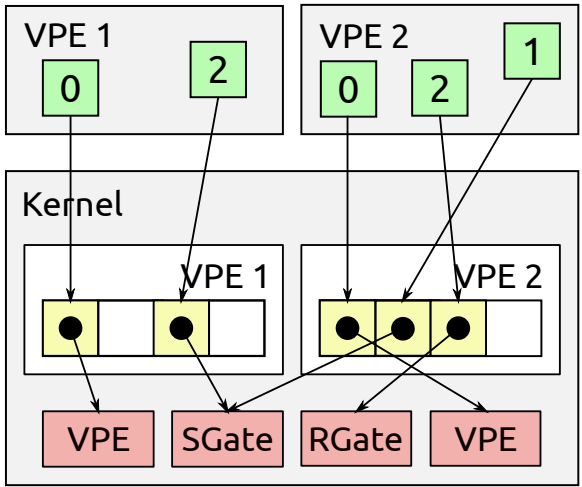
M³ System Call



Outline

1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

Overview



Capabilities

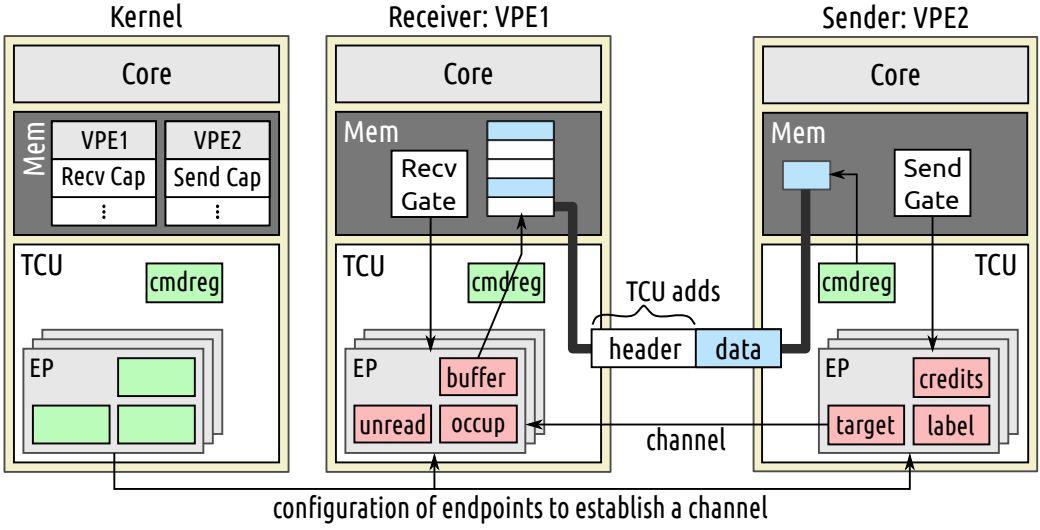
M^3 capabilities:

- Send: send messages to a receive EP
- Receive: receive messages from send EPs
- Memory: access remote memory via TCU
- Service: create sessions
- Session: exchange caps with service
- Endpoint: configure EPs of own or foreign TCU
- VPE: use a processing element

Capability Exchange

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

Communication



Virtual PEs

- M³ kernel manages user PEs in terms of VPEs
- VPE is combination of a process and a thread
- VPE creation yields a VPE capability and memory capability
- Library provides primitives like `fork` and `exec`
- VPEs are used for *all* PEs:
  - Accelerators are not handled differently by the kernel
  - All VPEs can perform system calls
  - …

VPEs – Examples

Executing ELF-Binaries

```
VPE vpe("test");
char *args[] = {"/bin/hello", "foo", "bar"};
vpe.exec(3, args);
```

VPEs – Examples

Executing ELF-Binaries

```
VPE vpe("test");
char *args[] = {"/bin/hello", "foo", "bar"};
vpe.exec(3, args);
```

Lambdas

```
VPE vpe("test");
MemGate mem = MemGate::create_global(0x1000, RW);
vpe.delegate(CapRngDesc(mem.sel(), 1));
vpe.run([&mem]() {
    mem.read(buf, sizeof(buf));
});
```

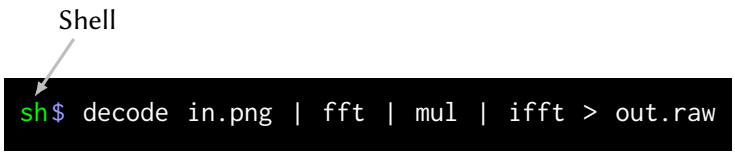
Outline

1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

OS Service Access for all Processing Element Types

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

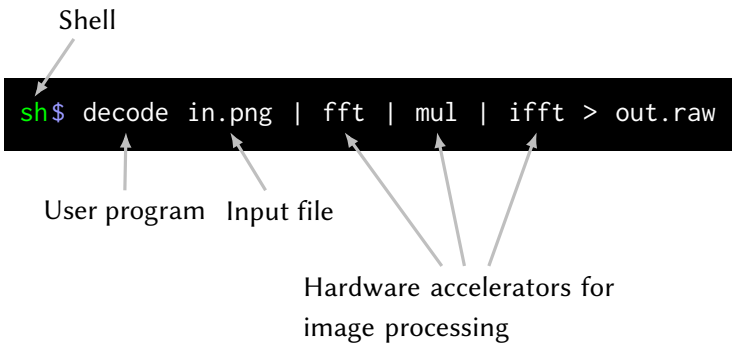
OS Service Access for all Processing Element Types



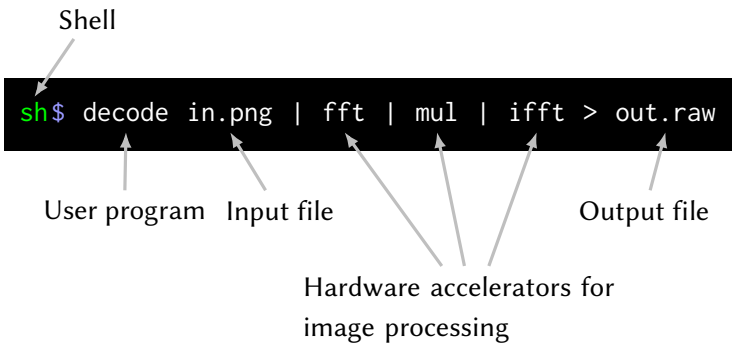
OS Service Access for all Processing Element Types

OS Service Access for all Processing Element Types

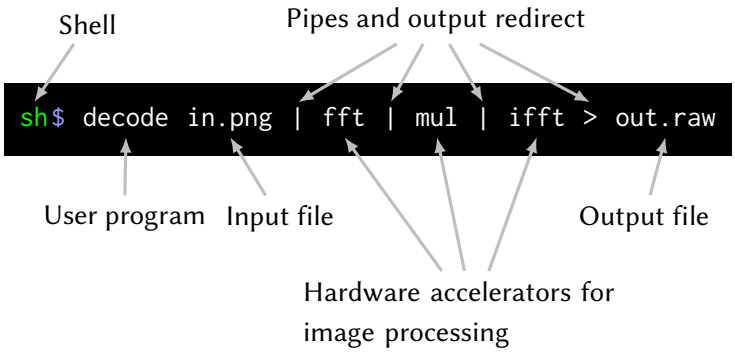
OS Service Access for all Processing Element Types



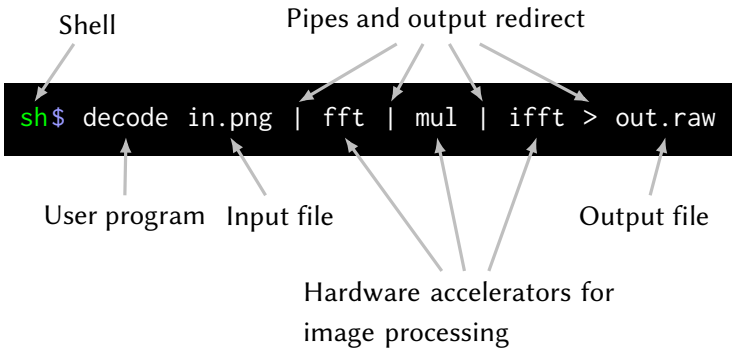
OS Service Access for all Processing Element Types



OS Service Access for all Processing Element Types

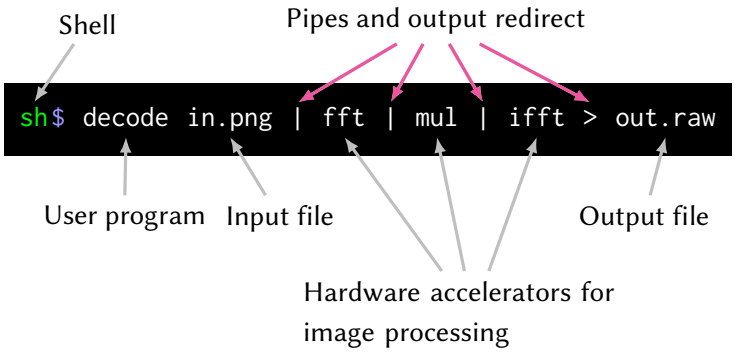


OS Service Access for all Processing Element Types



Challenges:

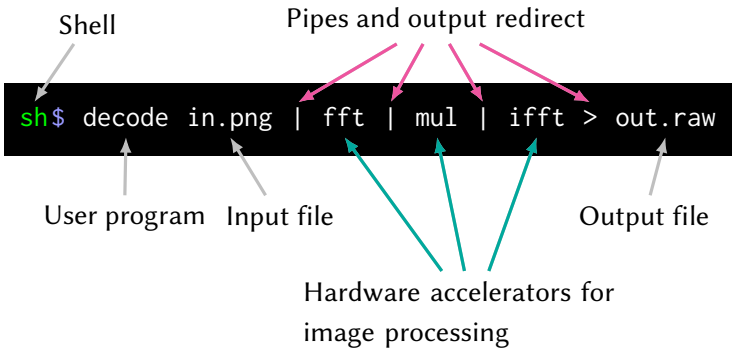
OS Service Access for all Processing Element Types



Challenges:

- OS must provide generic protocols

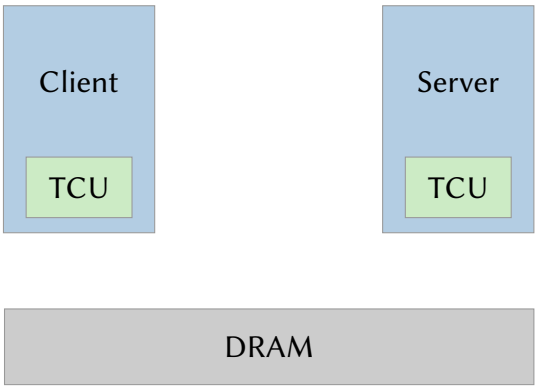
OS Service Access for all Processing Element Types



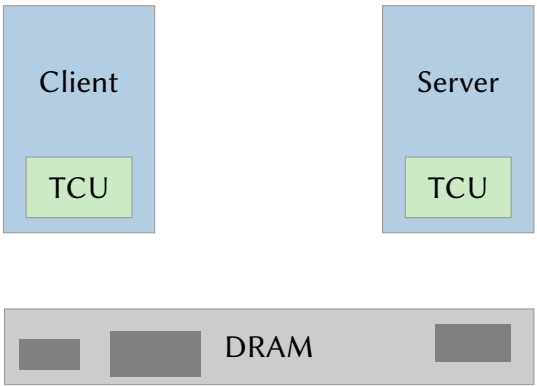
Challenges:

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

Generic Protocols



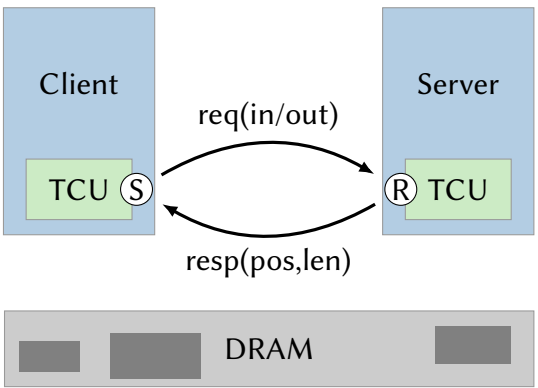
Generic Protocols



File protocol:

- Data in memory

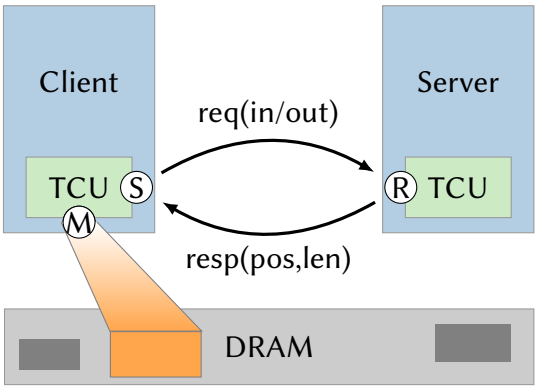
Generic Protocols



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

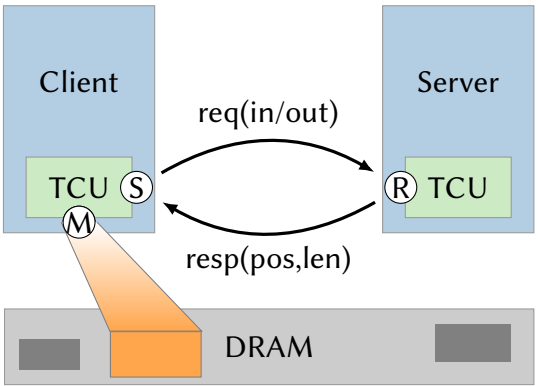
Generic Protocols



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

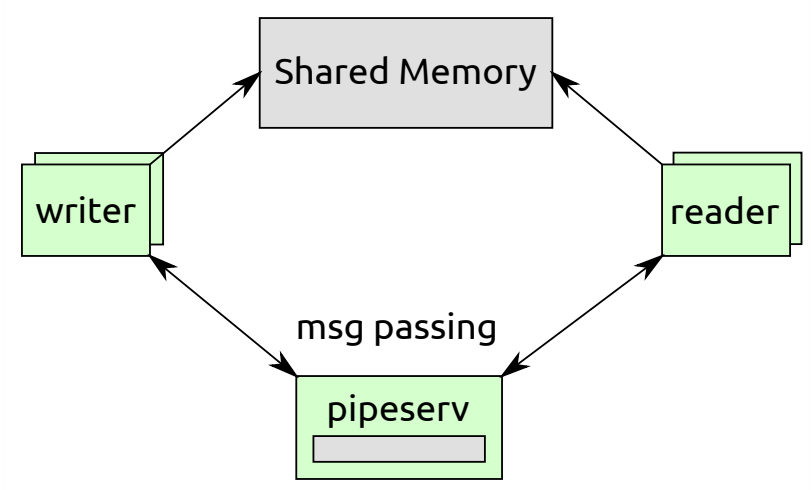
- M³FS organizes the file's data in extents
- M³FS can be used with a memory and disk backend
  - With memory backend, FS image is a contiguous region in DRAM
  - Clients get access to parts of the image
  - With disk backend, M³FS uses a buffer cache in DRAM
  - Clients get access to parts of buffer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

Implementation: Pipe – Overview

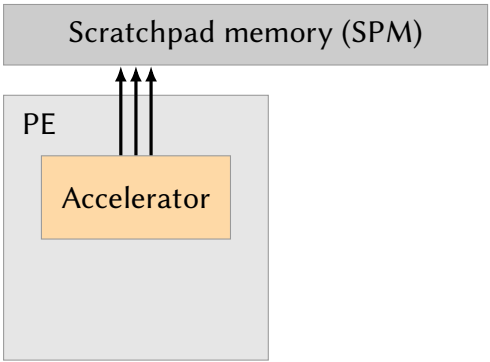
Implementation: Pipe – Overview



Implementation: Pipe

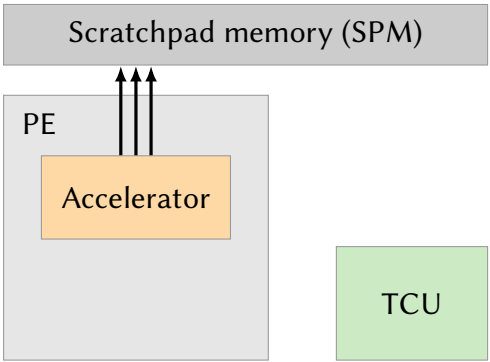
- Two types of sessions: *pipe session*, *channel session*
- Pipe session represents whole pipe, allows to create channels
- Channel session implements file protocol
- Channel session can be cloned
- Server configures client's EP just once at the beginning
- `req(in/out)` request access to next data
- `commit(nbytes)` commits previous request

Additions to Accelerator



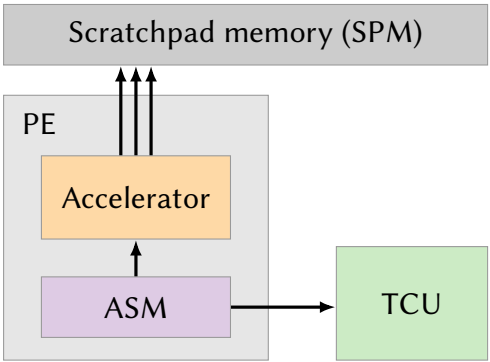
Off-the-shelf accelerators

Additions to Accelerator



Off-the-shelf accelerators

Additions to Accelerator

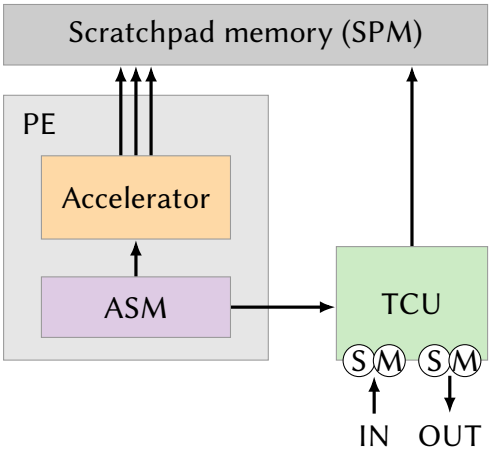


Off-the-shelf accelerators

Accelerator Support Module (ASM):

- Interacts with TCU and accelerator

Additions to Accelerator

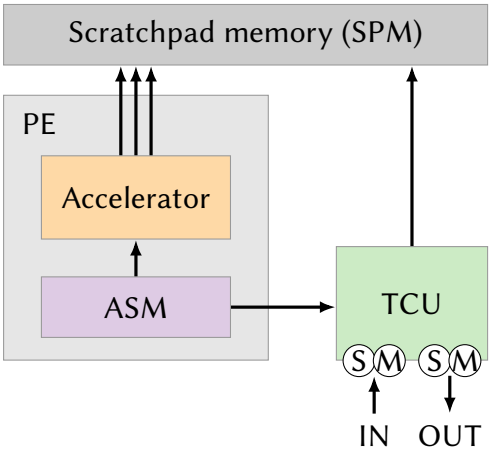


Off-the-shelf accelerators

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):

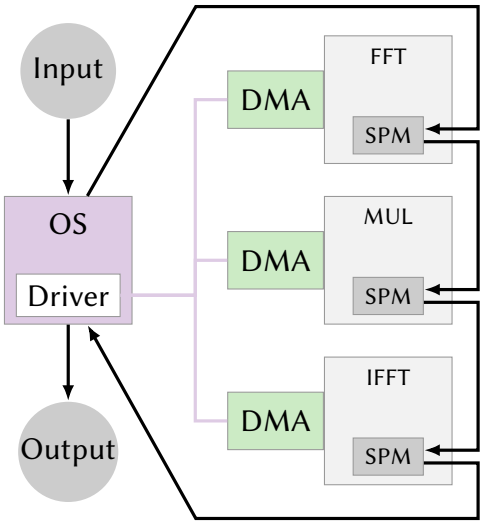
- Interacts with TCU and accelerator
- Implements file protocol for input and output channel
- ASM assumes that endpoints are setup externally by software

Demo

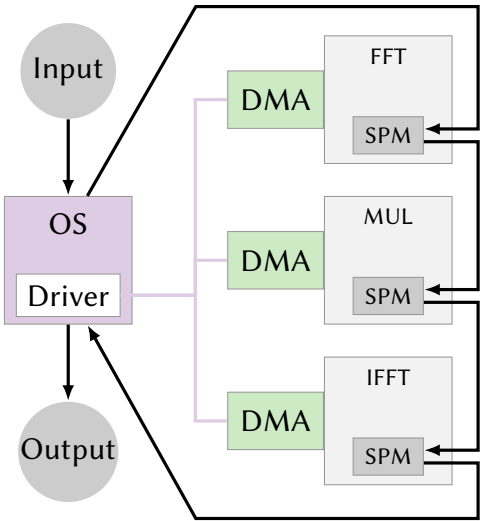
Outline

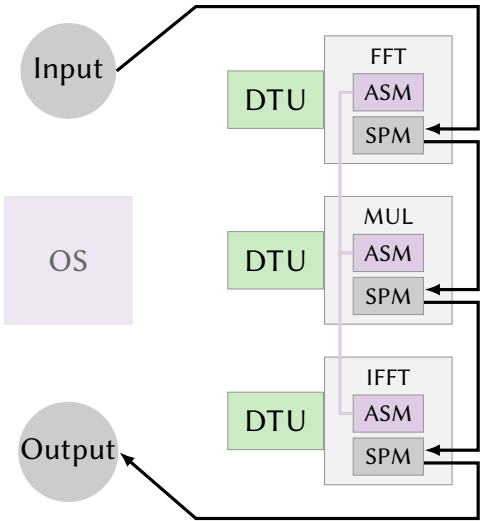
1. The New System Architecture
2. Prototype Platforms
3. Isolation and Communication
4. Operating System Overview
5. Capabilities
6. OS Services and Accelerators
7. Evaluation

Assisted vs. Autonomous

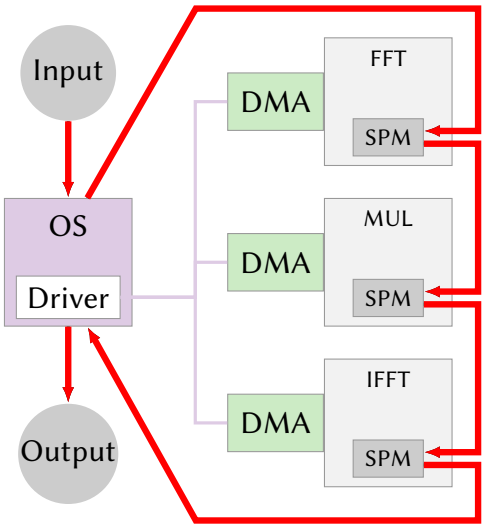


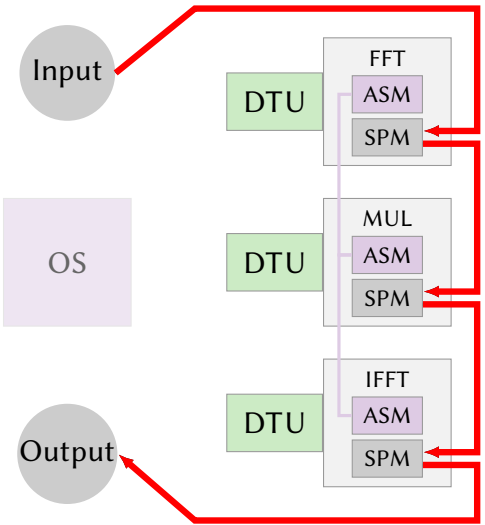
Assisted vs. Autonomous



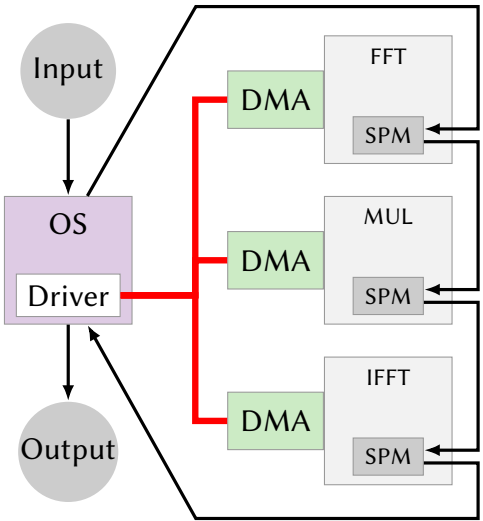


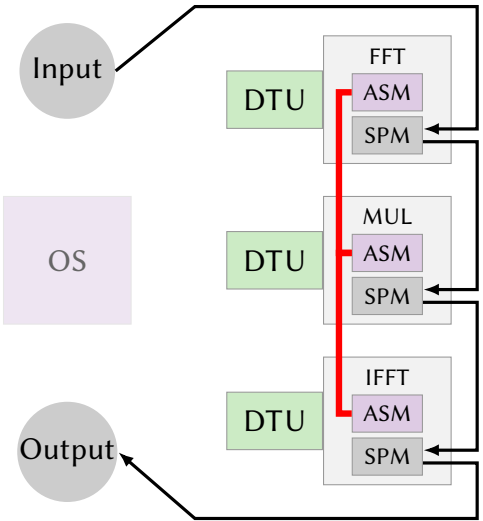
Assisted vs. Autonomous





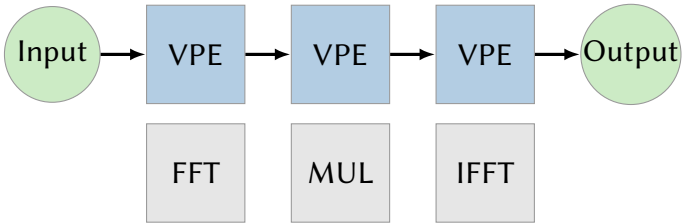
Assisted vs. Autonomous



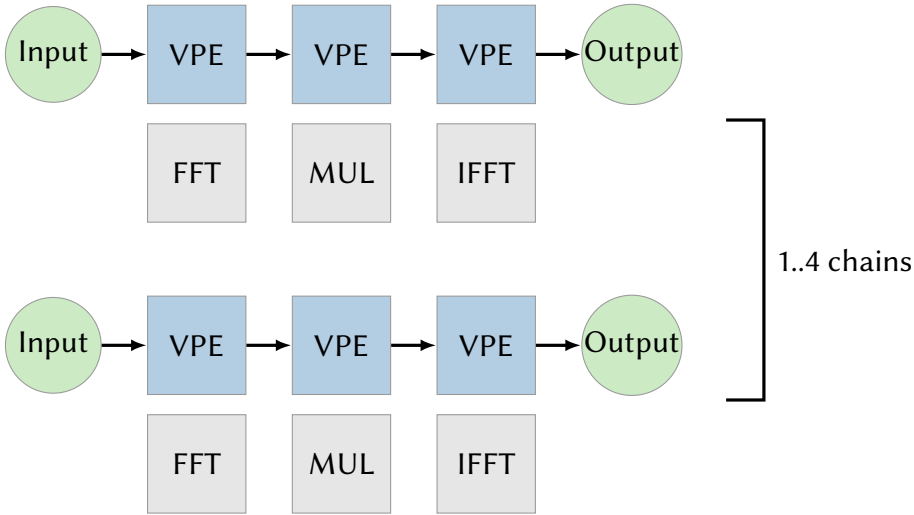


Accelerator Chains: Evaluation

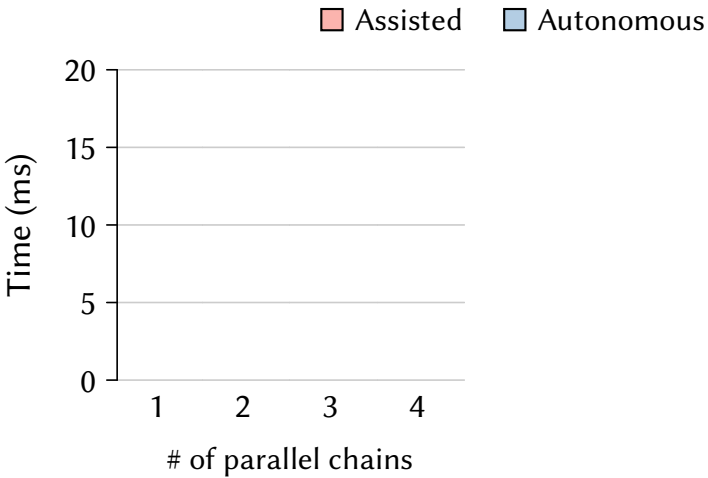
Accelerator Chains: Evaluation



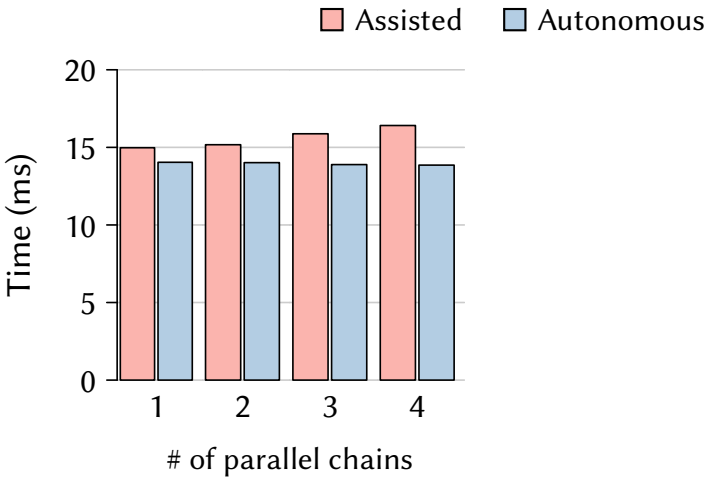
Accelerator Chains: Evaluation



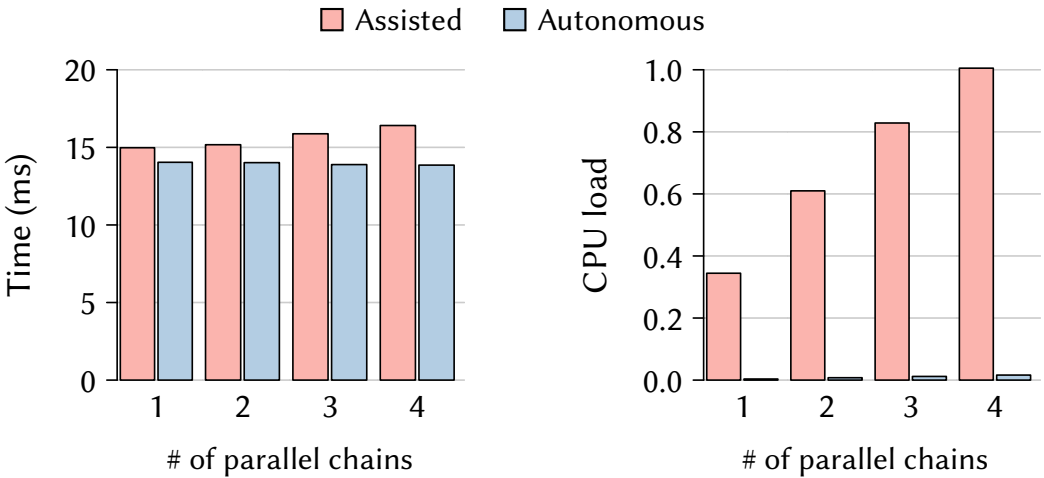
Accelerator Chains: Results



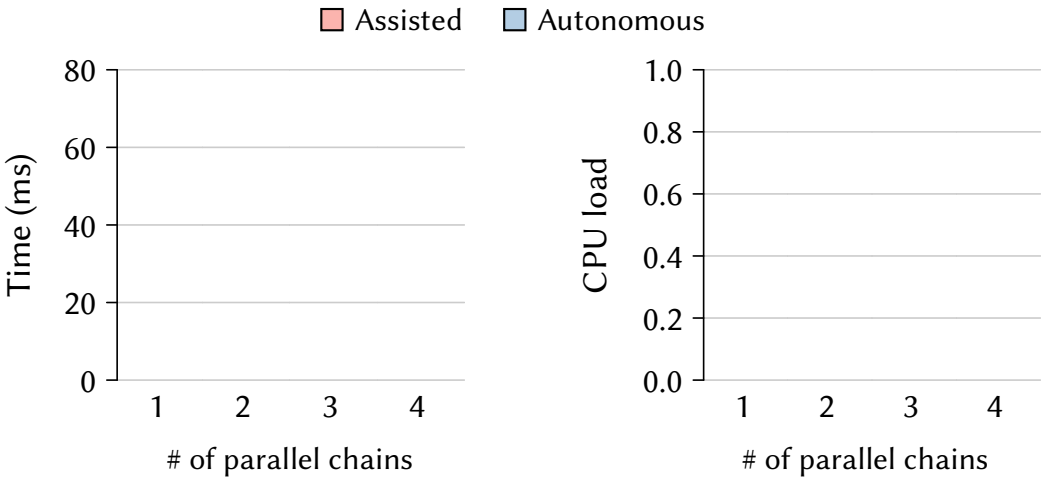
Accelerator Chains: Results



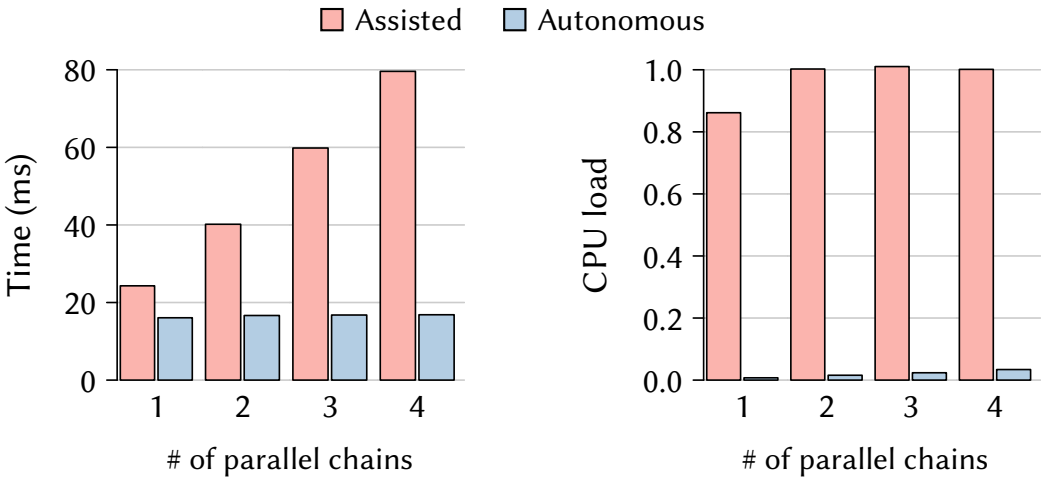
Accelerator Chains: Results



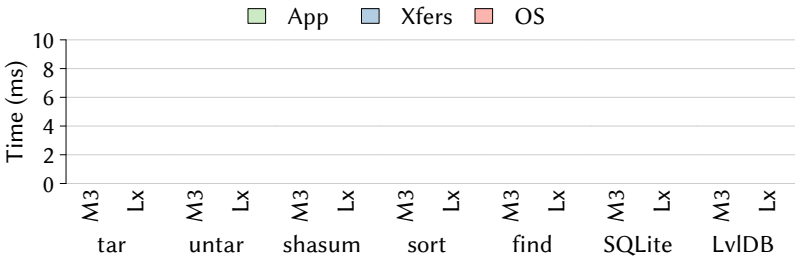
Accelerator Chains: Results (PCIe-like Latency)



Accelerator Chains: Results (PCIe-like Latency)

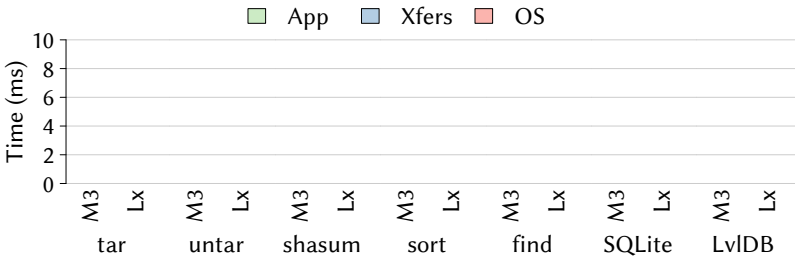


Linux Application Workloads

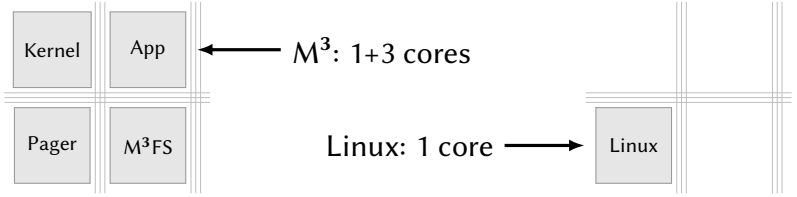


- M^3 vs. Linux 4.10
- Traced on Linux, replayed on M^3
- M^3FS vs. Linux tmpfs

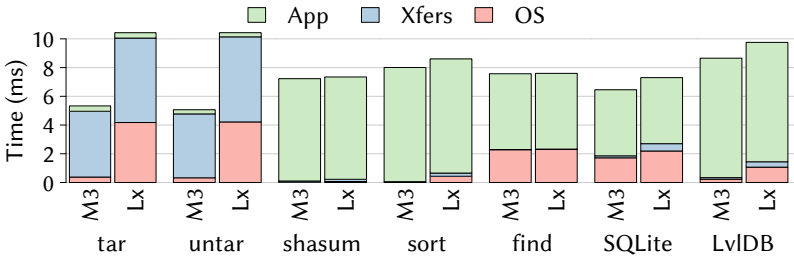
Linux Application Workloads



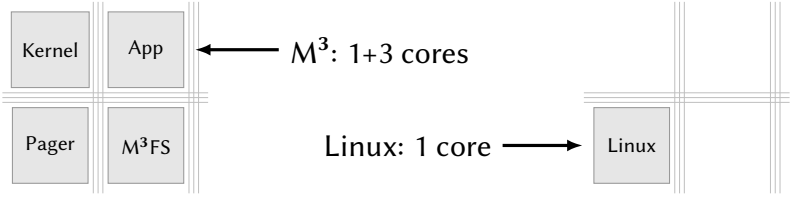
- M³ vs. Linux 4.10
- Traced on Linux, replayed on M³
- M³FS vs. Linux tmpfs



Linux Application Workloads



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



Ongoing Work at the Barkhausen Insitut

- Tile sharing among multiple applications
- Hardware implementation (FPGA and silicon)
- Connected devices with remote attestation

Conclusion

- M^3 uses a hardware/operating-system co-design
- TCU introduces common interface for all processing elements (PEs)
- Allows to integrate all (untrusted) PEs as first-class citizens
- Access to OS services for all PEs
- Allows simple management of complex systems

More Information

- **M³: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores**
  Nils Asmussen, Marcus Völp, Benedikt Nöthen, Hermann Härtig, and Gerhard Fettweis
  ASPLOS 2016
- **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