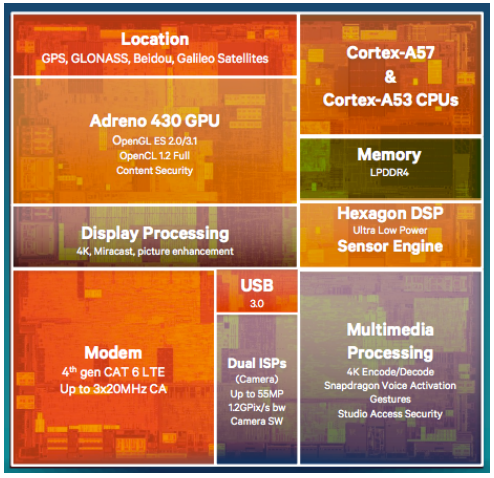
M^3 – **M**icrokernel-based Syste**m** for Heterogeneous **M**anycores

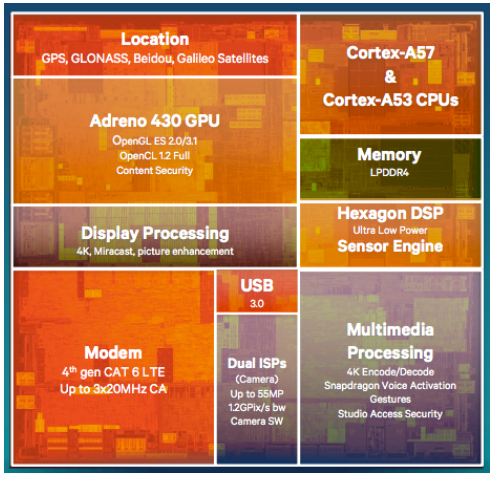
Nils Asmussen

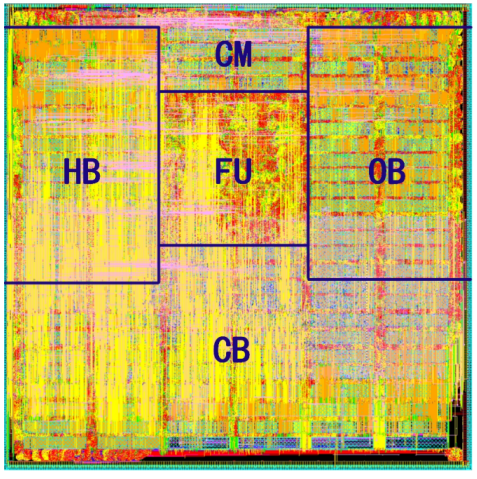
MKC, 06/29/2017

Heterogeneous Systems

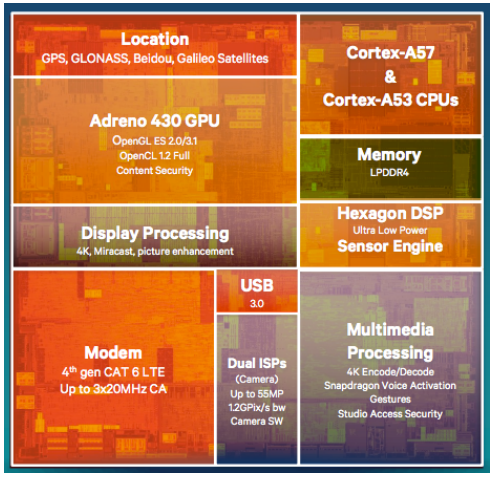


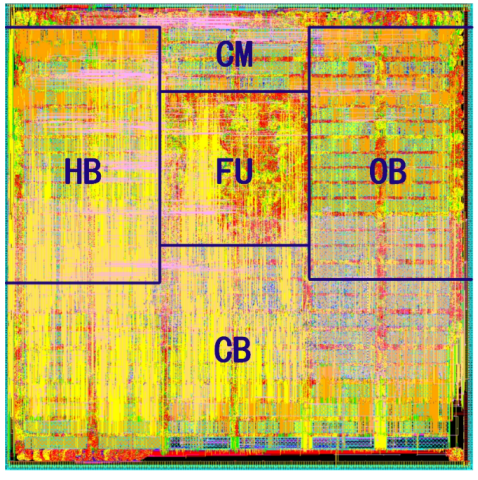
Heterogeneous Systems





Heterogeneous Systems





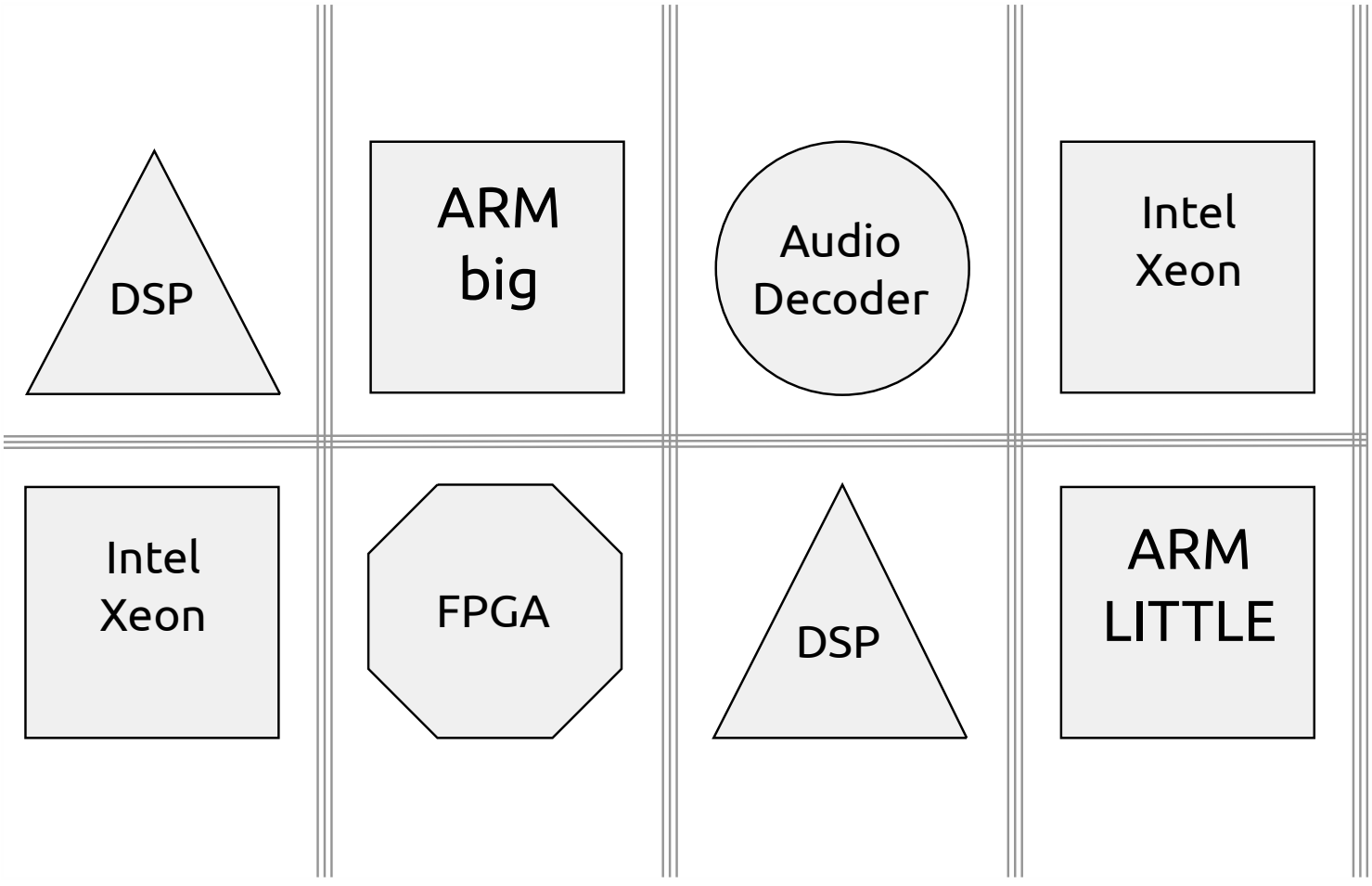
cfaed
CENTER FOR
ADVANCING
ELECTRONICS
DRESDEN

Why?

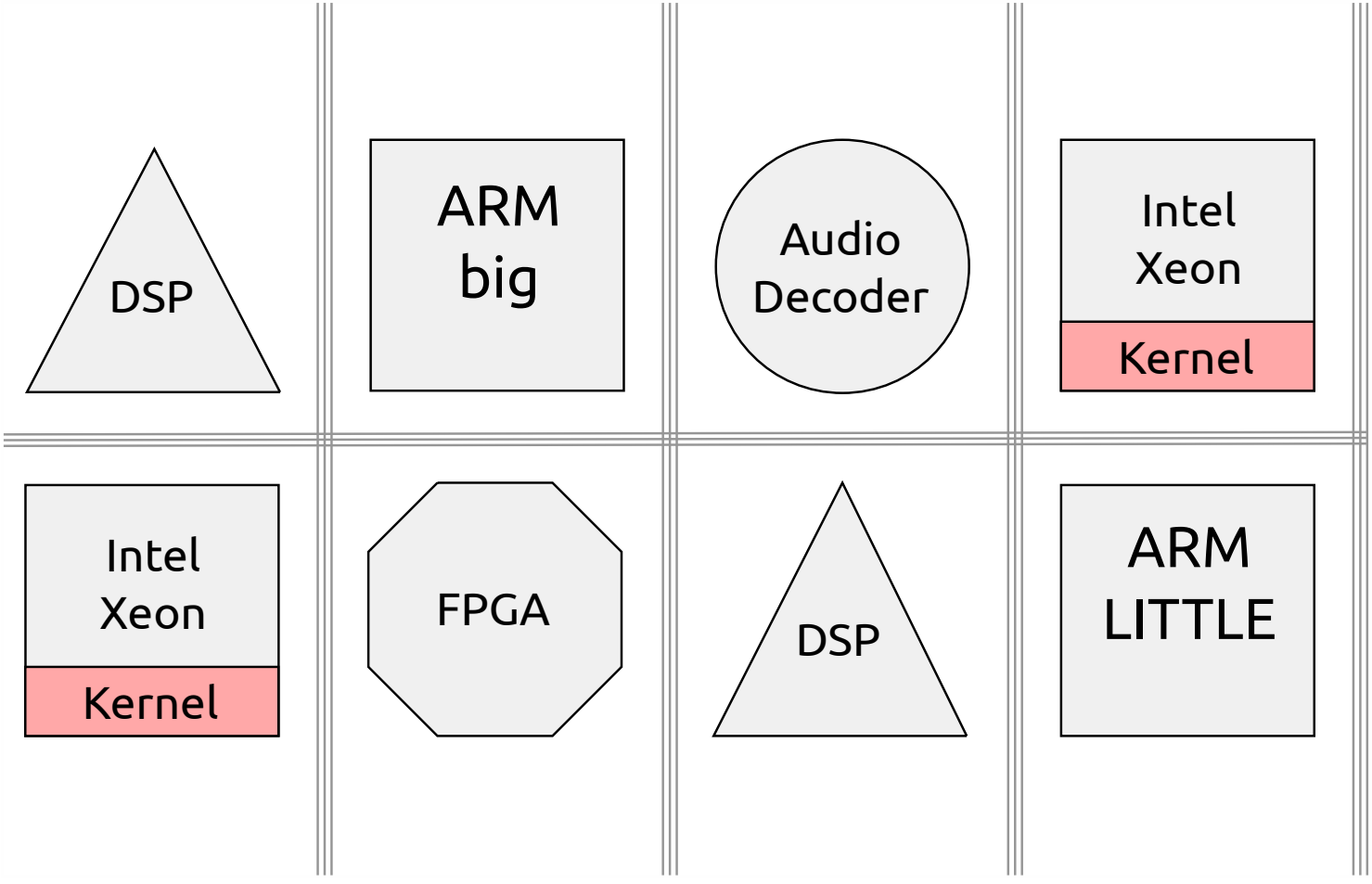
- FPGA-based memcached 16x better in performance per watt than Atom CPU [1]
- Machine learning accelerator is 20% faster than GPU and requires 128 times less energy [2]

[1] Thin servers with smart pipes: Designing SoC accelerators for memcached, ISCA'13
[2] PuDianNao: A polyvalent machine learning accelerator, ASPLOS'15

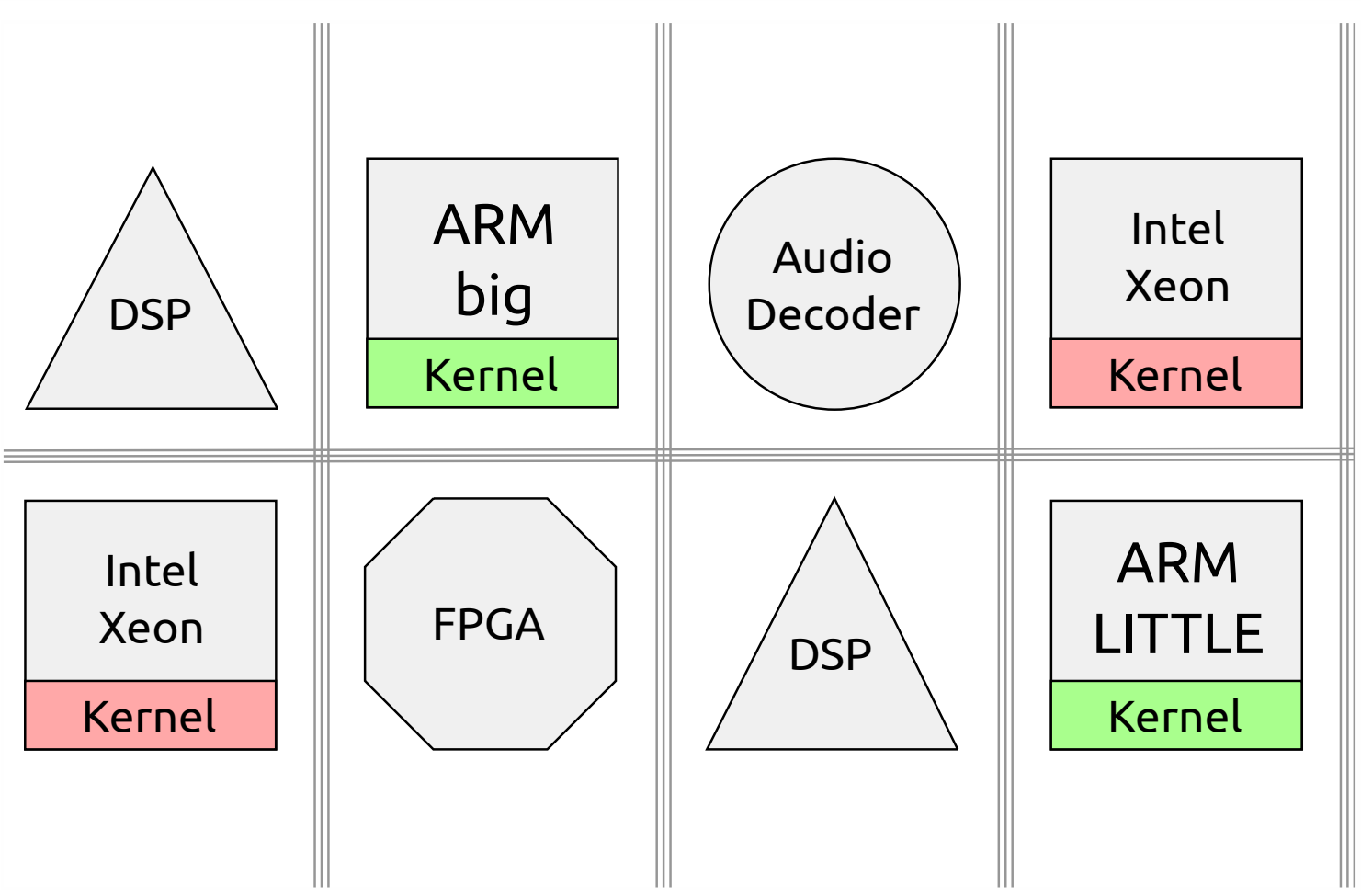
The Problem for OSes



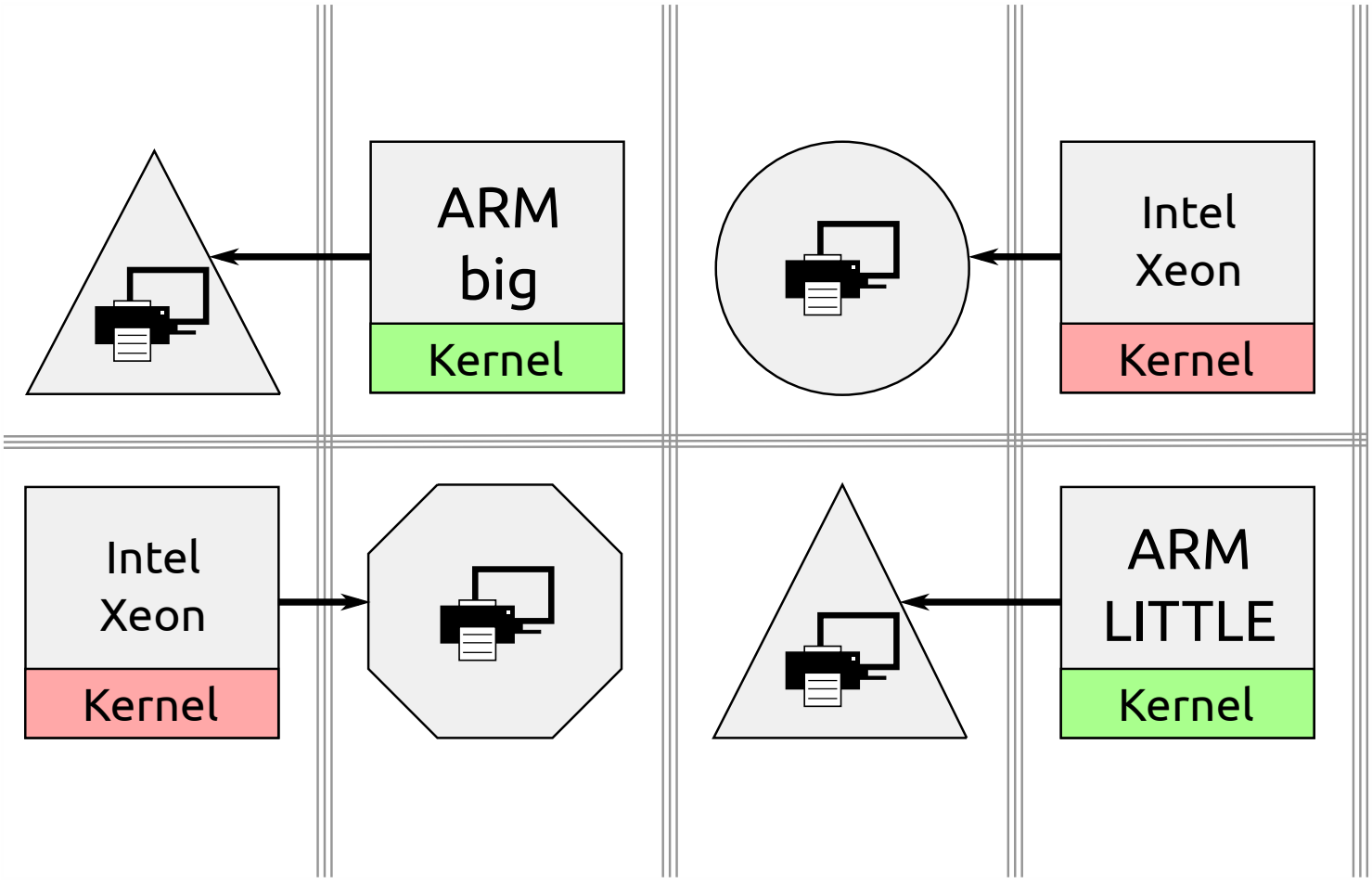
The Problem for OSes



The Problem for OSes



The Problem for OSes



Making Accelerators More First-Class

- File system access for GPUs [1]
- Network access for GPUs [2]
- Access to OS services from FPGAs [3,4]
- Computing directly on the SSD [5]

[1] GPUfs: integrating a file system with GPUs, ASPLOS'13
[2] GPUnet: Networking Abstractions for GPU Programs, OSDI'14
[3] ReconOS: An operating system approach for reconfigurable computing, MICRO'14
[4] A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers Using BORPH, TECS'08
[5] Willow: A user-programmable SSD, OSDI'14

Is There a Systematic Way?

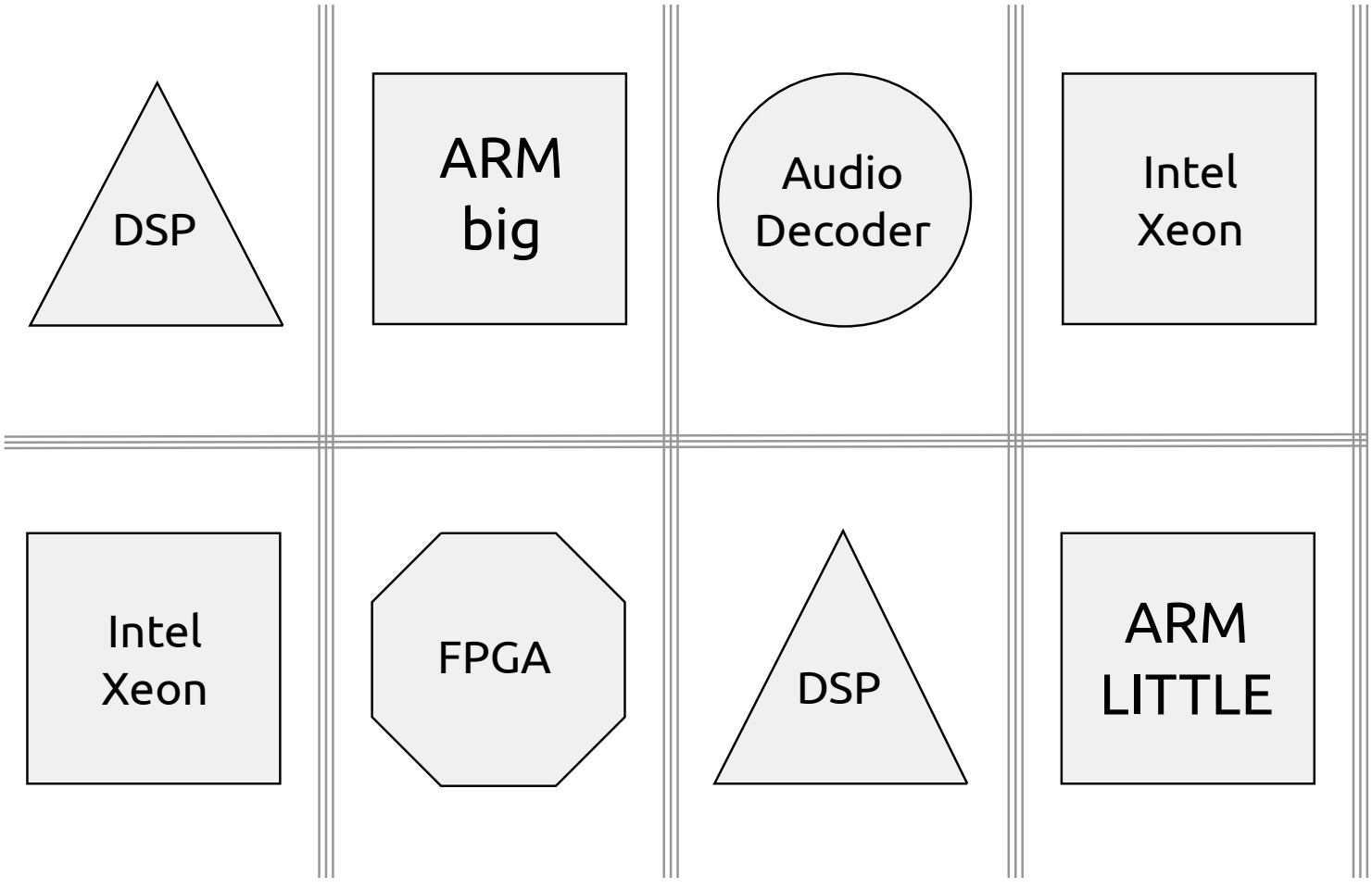
Can we design a system that treats all compute units (CU) as *first-class citizens* from the beginning?

1. Run untrusted code without causing harm
2. Access operating system services
3. Context switching support
4. Direct communication without involving CPU

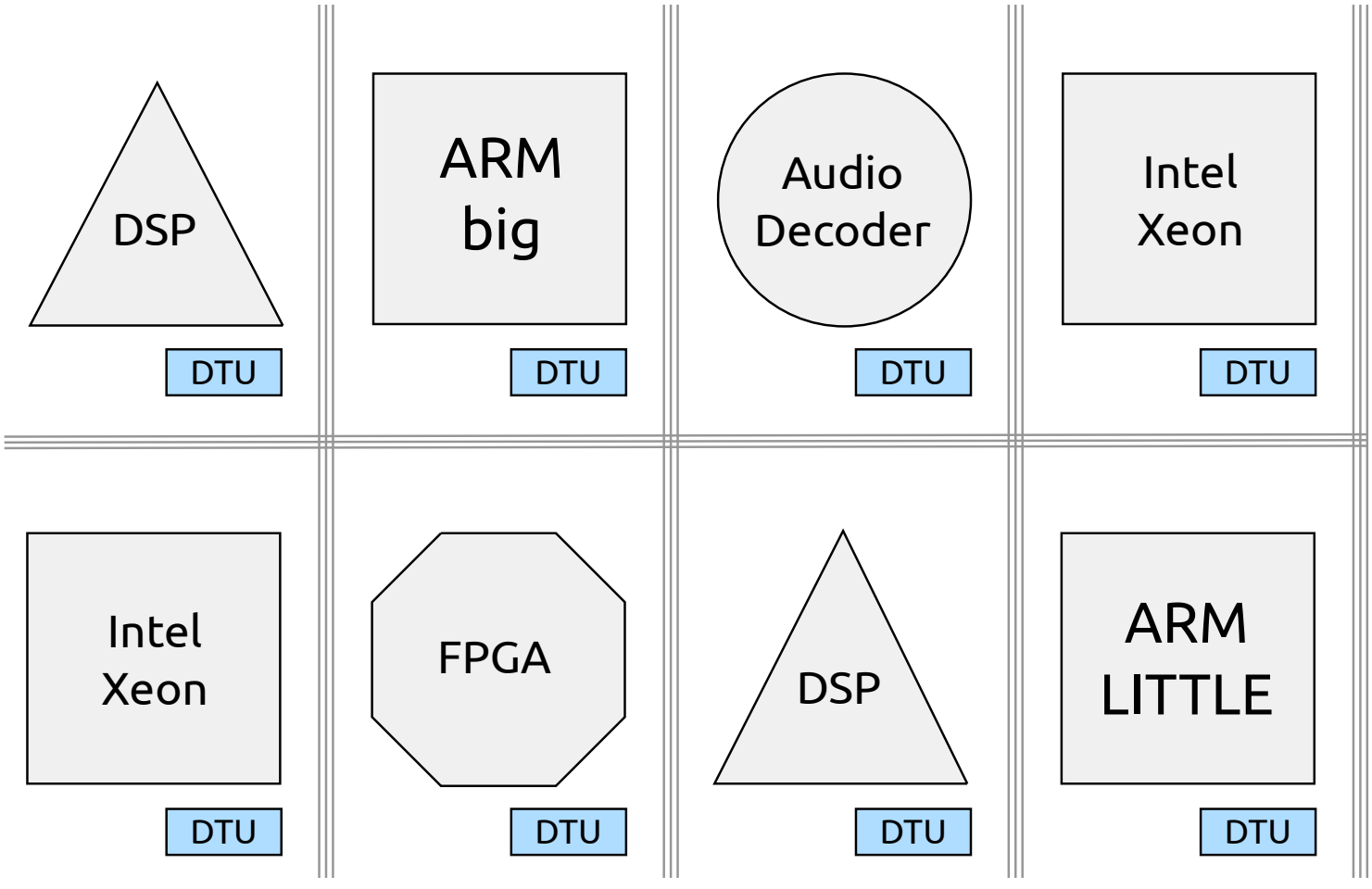
Outline

Outline

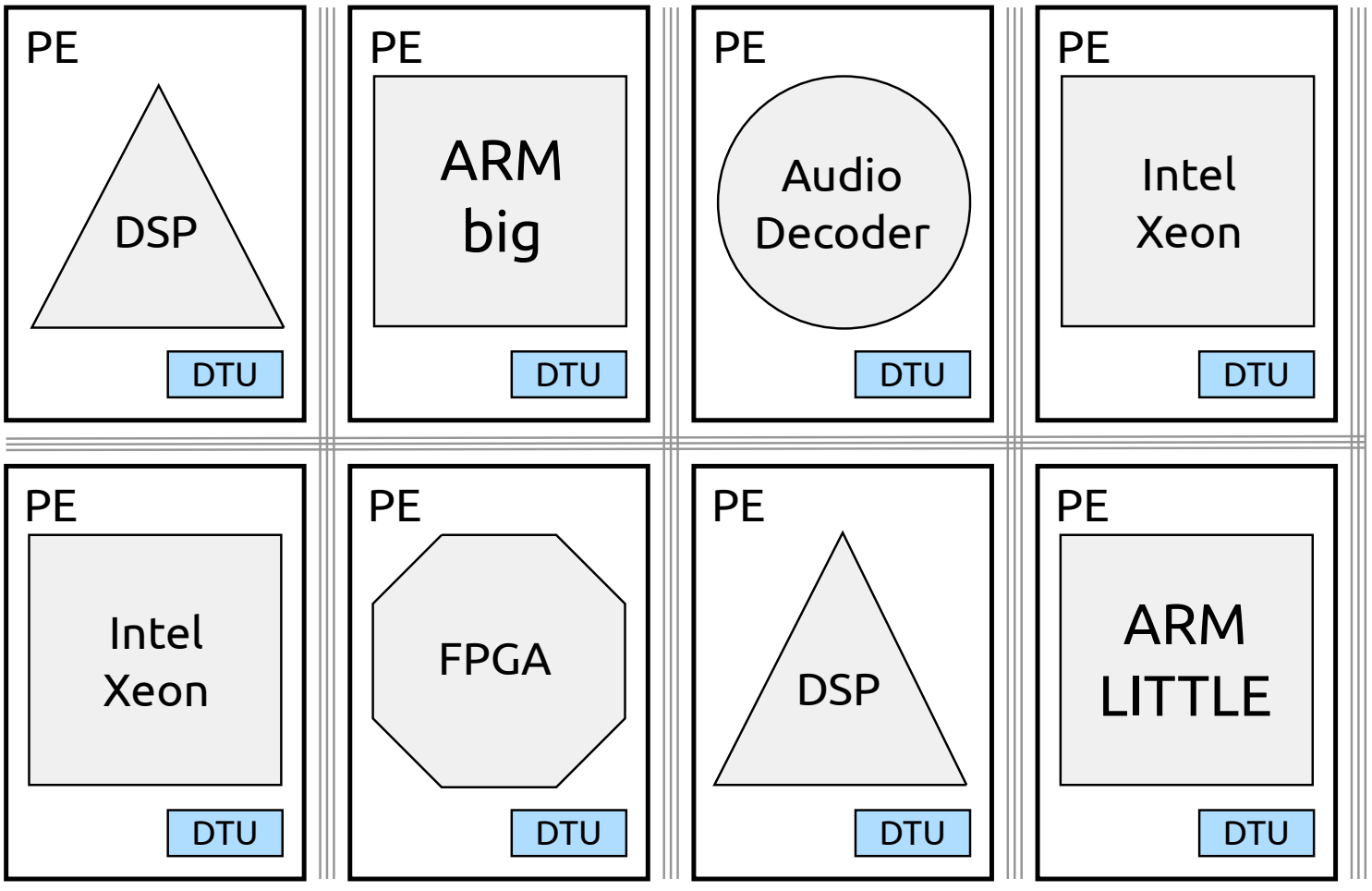
My Approach – Hardware



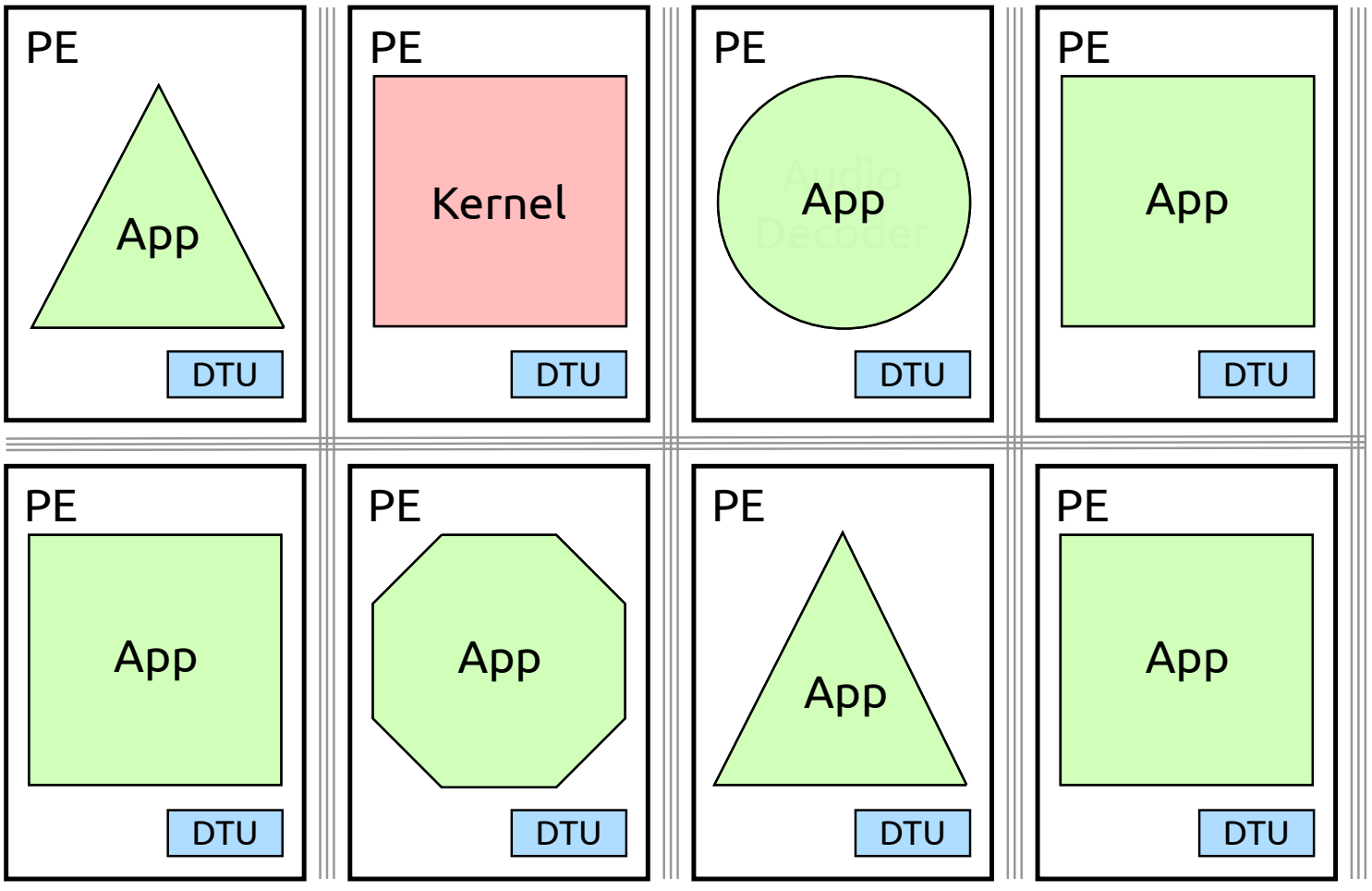
My Approach – Hardware



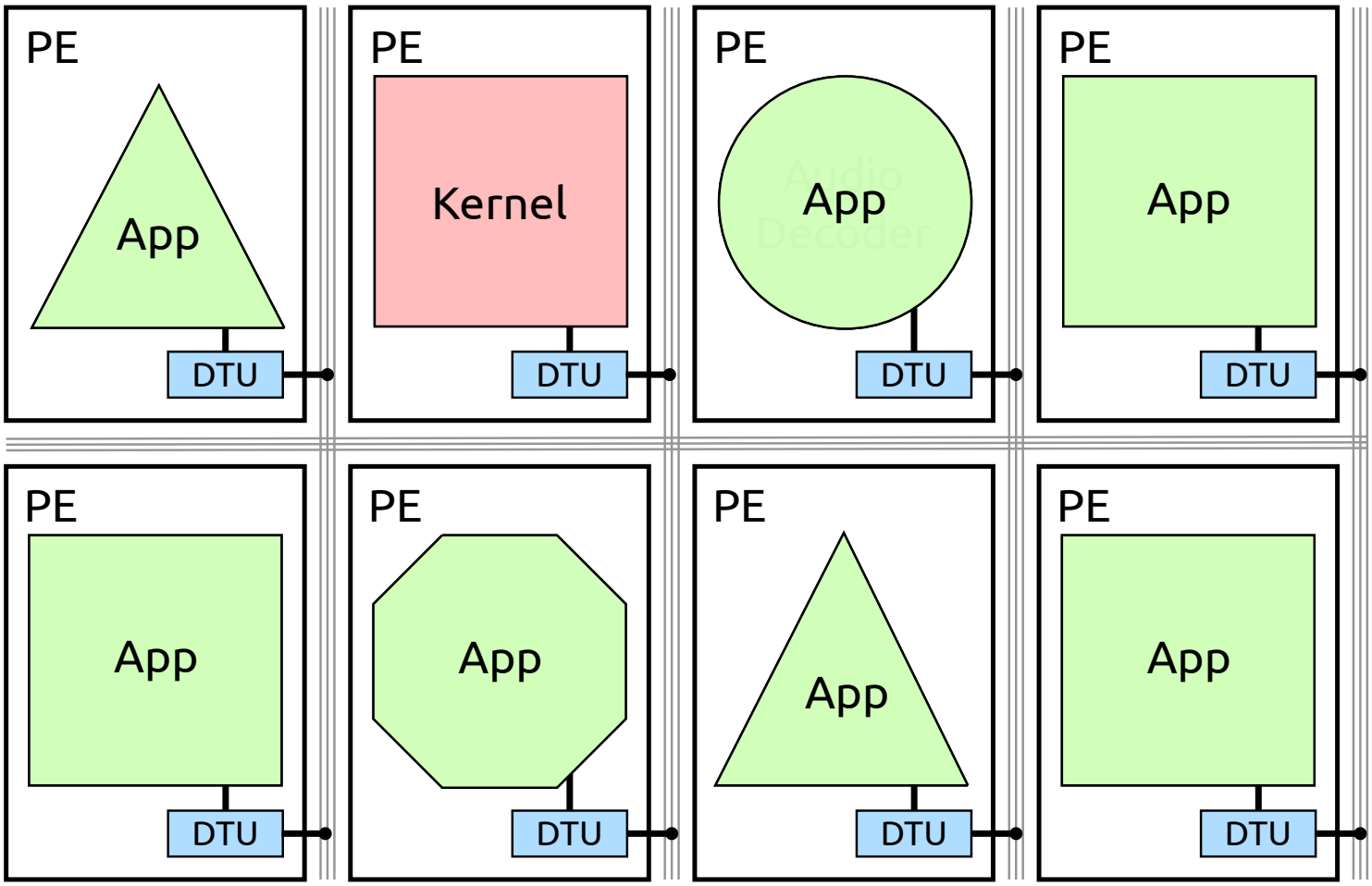
My Approach – Hardware



My Approach – Software



My Approach – Software



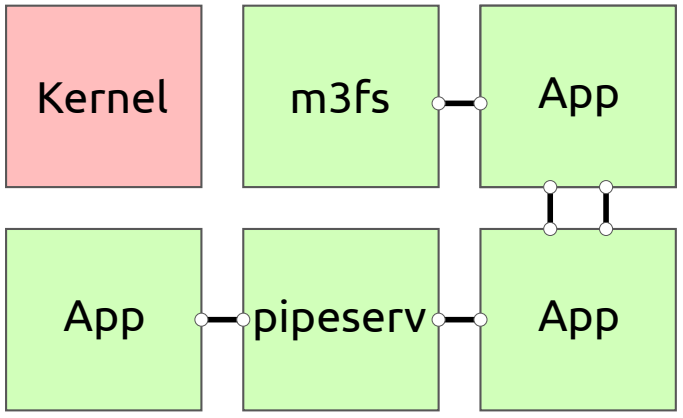
Asmussen et al.: M3: A Hardware/OS Co-Design to Tame Heterogeneous Manycores, ASPLOS'16

Data Transfer Unit

- Supports memory access and message passing
- Provides a number of endpoints
- Each endpoint can be configured for:
  1. Accessing memory (contiguous range, byte granular)
  2. Receiving messages into a receive buffer
  3. Sending messages to a receiving endpoint
- Direct reply on received messages
- Configuration only by kernel, usage by application
- Credit system to prevent DoS attacks

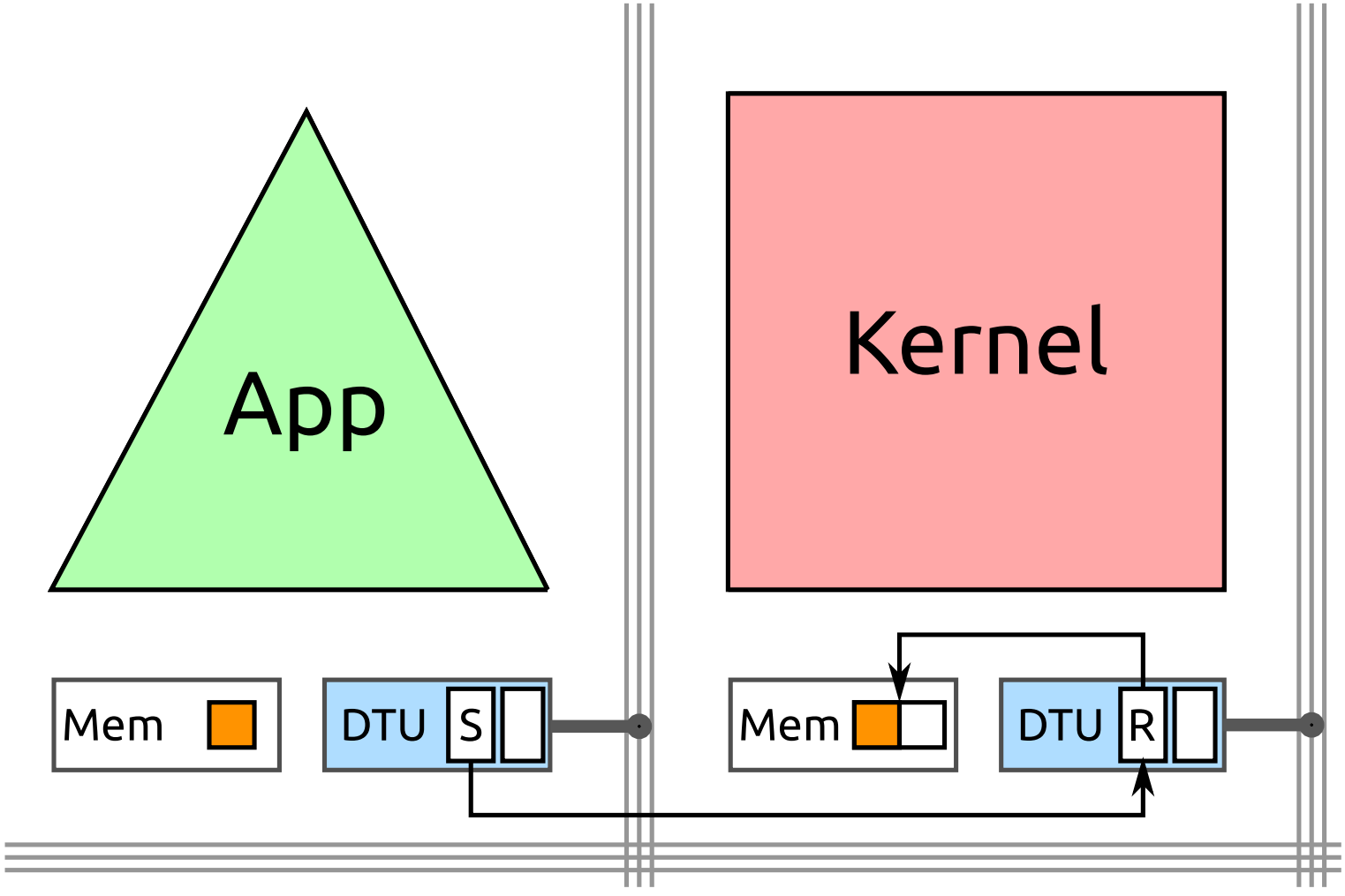
OS Design

- M^3: **M**icrokernel-based syste**m** for het. **m**anycores (or L4 ± 1)
- Implemented from scratch
- Drivers, filesystems, ... are implemented on top
- Kernel manages permissions, using capabilities



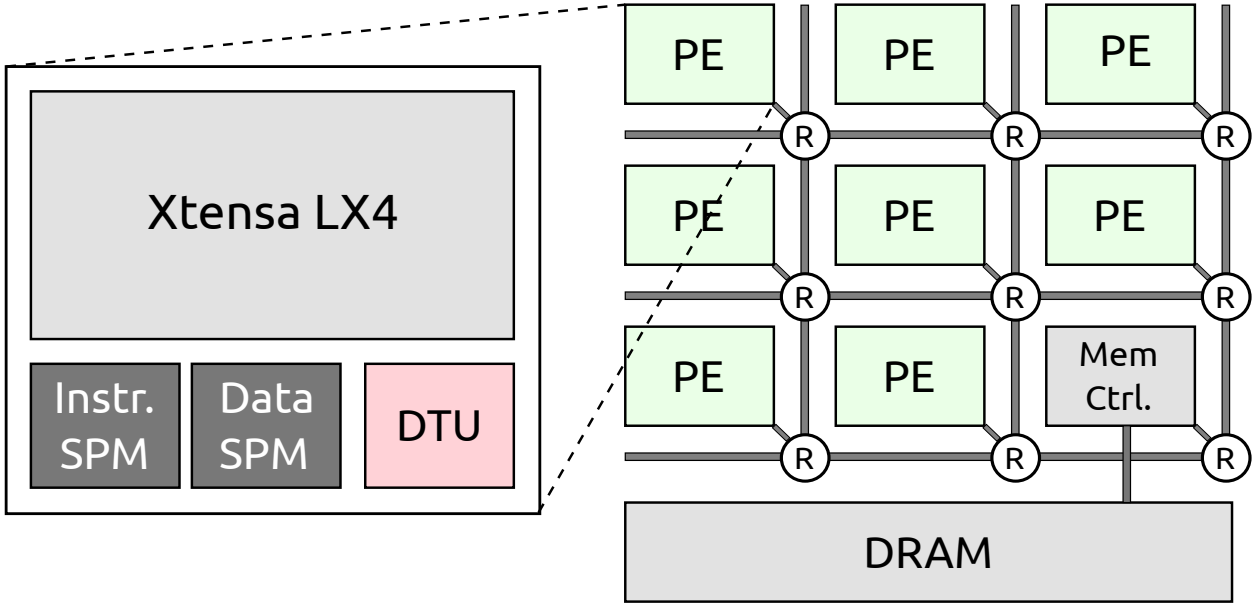
- DTU enforces permissions (communication, memory access)
- Kernel is independent of other CUs in the system

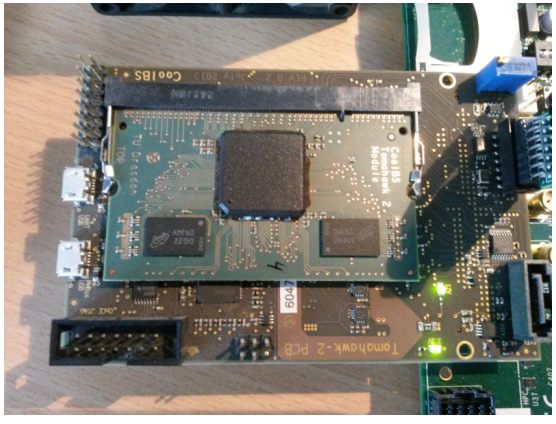
M³ System Call



Outline

Tomahawk 2 and 4





PEs have no OS support:

- No privileged mode
- No MMU, no caches, but SPM
- T2: simple DTU; T4: most features

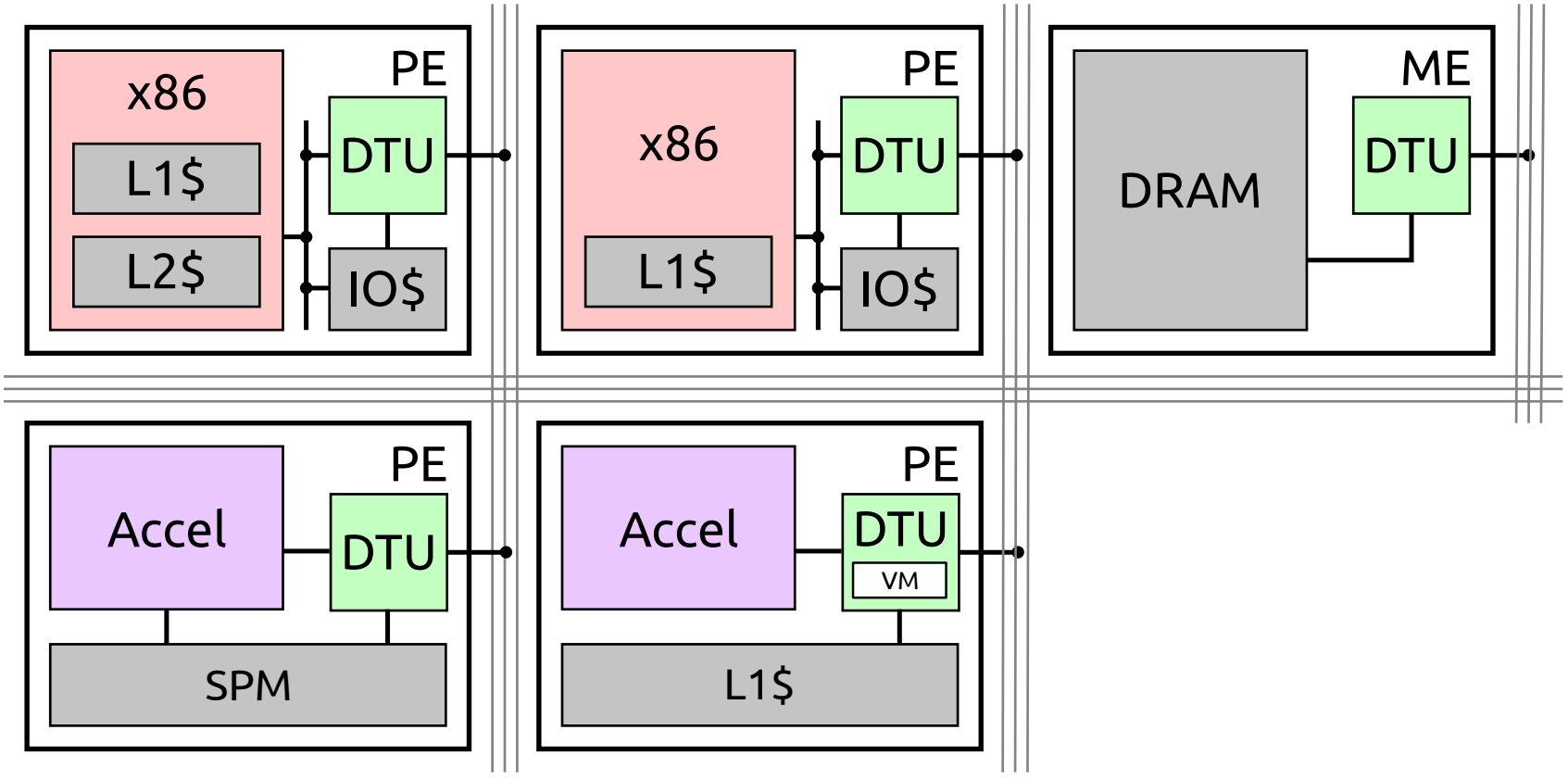
Linux

- M^3 runs on Linux using it as a virtual machine
- A process simulates a PE, having two threads (CPU + DTU)
- DTUs 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

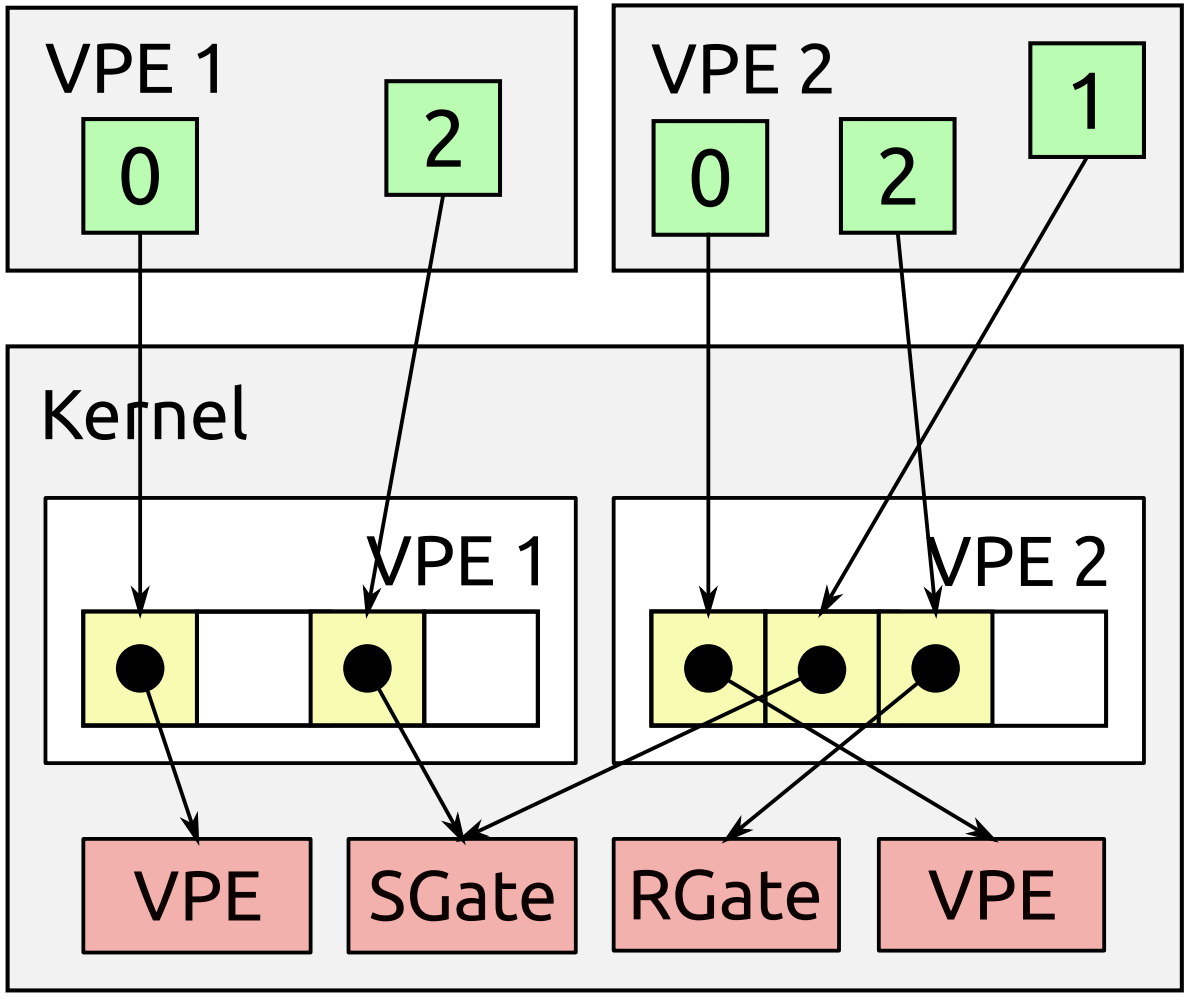
- Modular platform for computer architecture research
- Supports various ISAs (x86, ARM, Alpha, SPARC, ...)
- Provides detailed CPU and memory models
- Cycle-accurate simulation
- We built a DTU for gem5
- We also added hardware accelerators

gem5 – Example Configuration



Outline

Overview



Capabilities

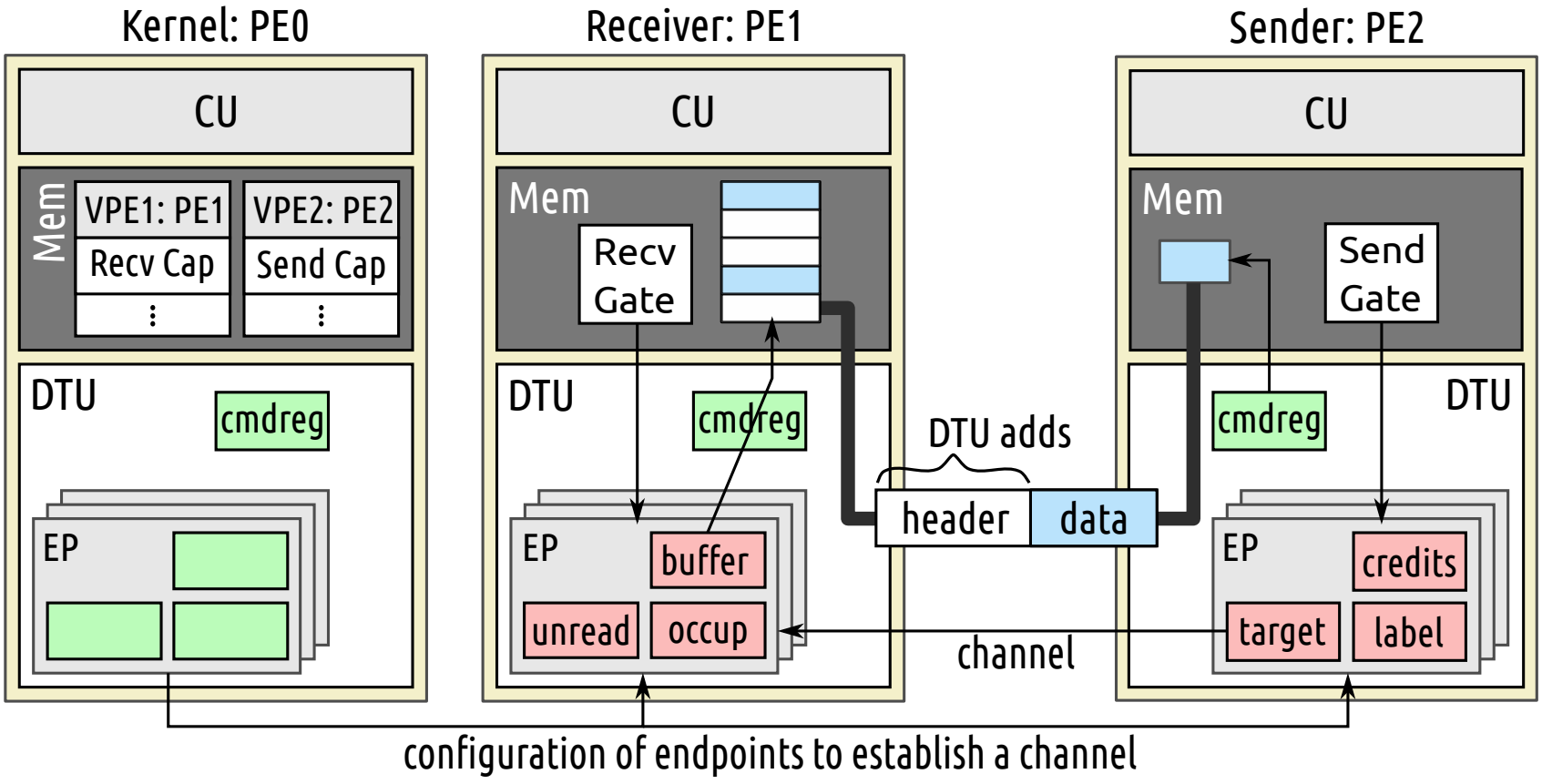
M^3 has the following capabilities:

- Send: send messages to a receive EP
- Receive: receive messages from send EPs
- Memory: access remote memory via DTU
- Mapping: access remote memory via load/store
- Service: create sessions
- Session: exchange caps with service
- VPE: use a PE

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 cap. and memory cap.
- 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
  - All VPEs can have time slices and priorities
  - ...

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);
```

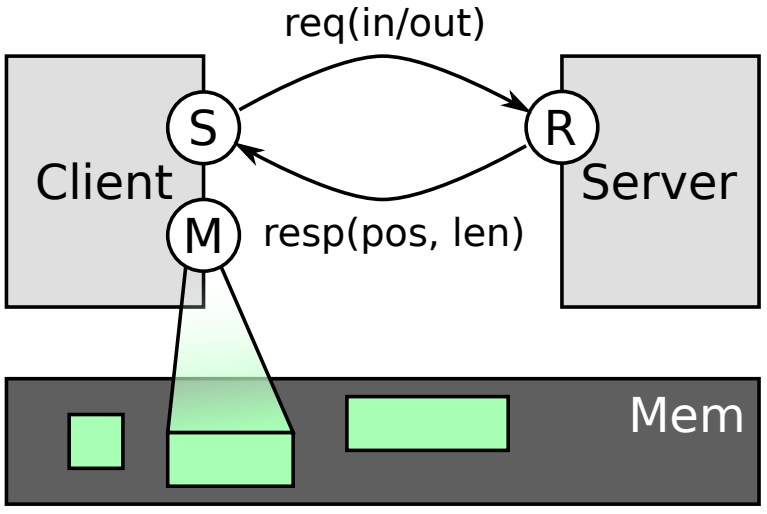
Asynchronous Lambdas

```
VPE vpe("test");
MemGate mem = MemGate::create_global(0x1000, RW);
vpe.delegate(CapRngDesc(mem.sel(), 1));
vpe.run_async([&mem]() {
    mem.read(buf, sizeof(buf));
    cout << "Done reading!\n";
});
```

Outline

File Protocol

- File protocol is used for all file-like objects
- Simple for accelerators, yet flexible for software
- Software uses POSIX-like API on top of the protocol
- Server provides client access to data by configuring client's memory endpoint



- Client accesses data via DTU, without involving others
- `req(in/out)` requests next input/output piece and implicitly commits previous piece
- `commit(nbytes)` commits `nbytes` of previous piece
- Receiving `resp(n, 0)` indicates EOF

Implementation: m3fs – Overview

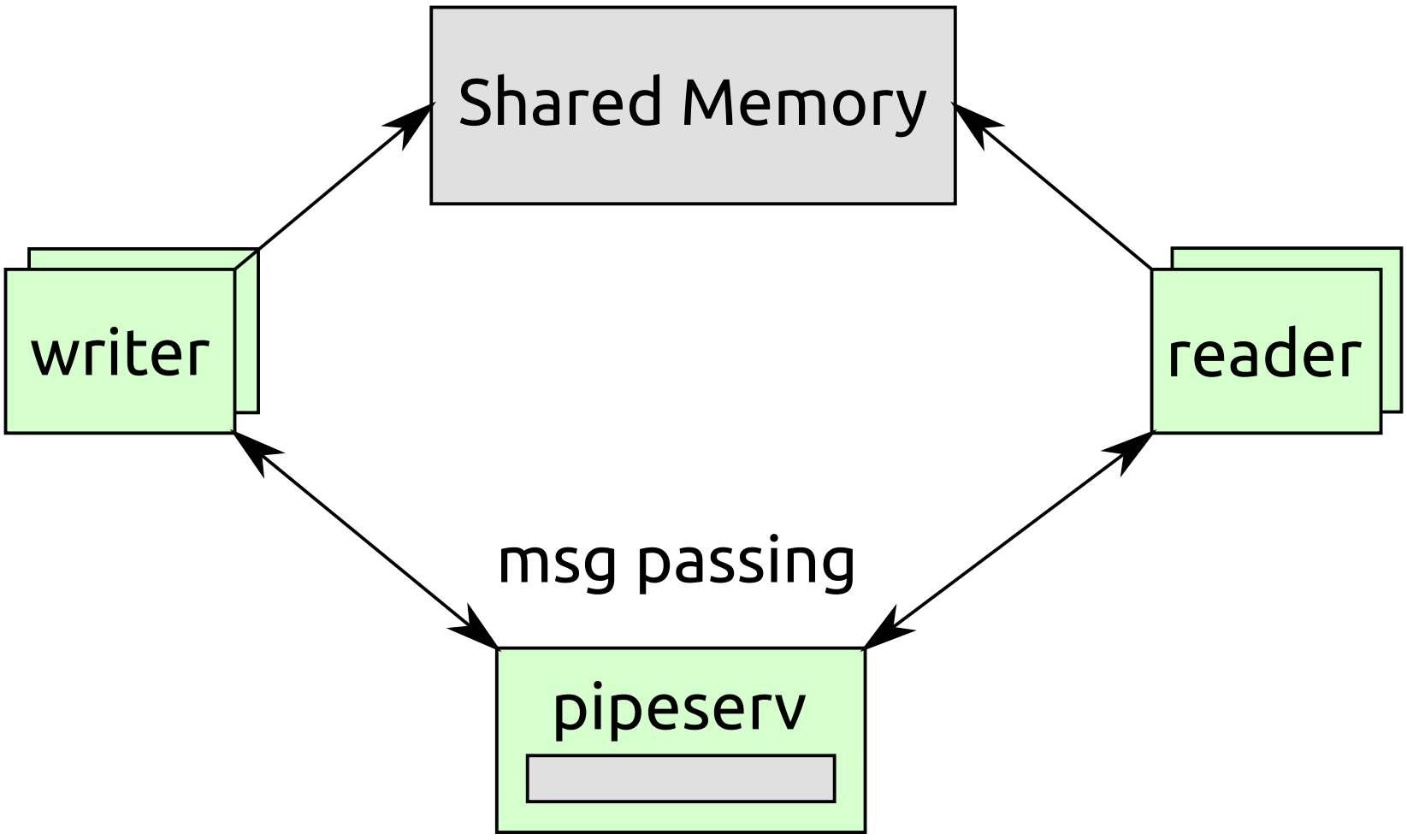
- *m3fs* is an in-memory file system
- m3fs organizes the file's data in extents
- Two types of sessions: *metadata session*, *file session*
- Metadata session is created first, allows `stat`, `open`, ...
- `open` creates a new file session
- Both sessions can be cloned to provide other VPEs access

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

File Multiplexing

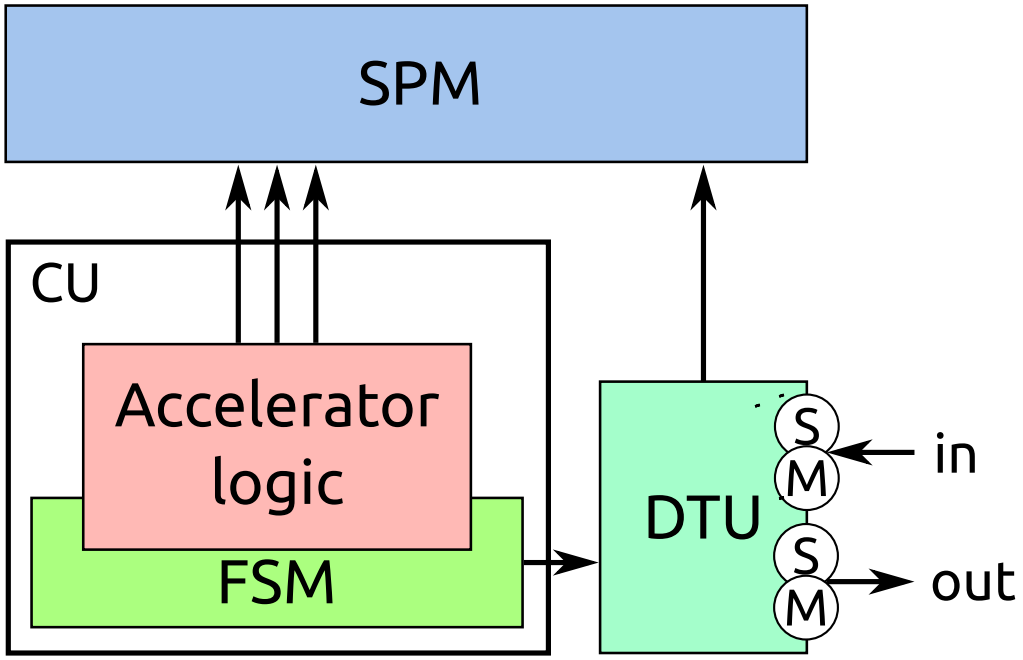
- File protocol maps directly to EPs (limited resource)
- Number of open files shouldn't be limited (that much)
- libm3 dedicates at most 4 EPs to files and multiplexes them
- Multiplexing requires:
  1. `commit(nbytes)` to commit read/written data
  2. revocation of EP capability (old server)
  3. delegation of EP capability (new server)
  4. next `read`/`write` will contact server again
- Fortunately, file multiplexing does almost never happen

Accel. Example: Stream Processing

- Accelerator works on scratchpad memory
- Input data needs to be loaded into scratchpad
- Result needs to be stored elsewhere

Accel. Example: Stream Processing

- Accelerator works on scratchpad memory
- Input data needs to be loaded into scratchpad
- Result needs to be stored elsewhere



Shell Integration

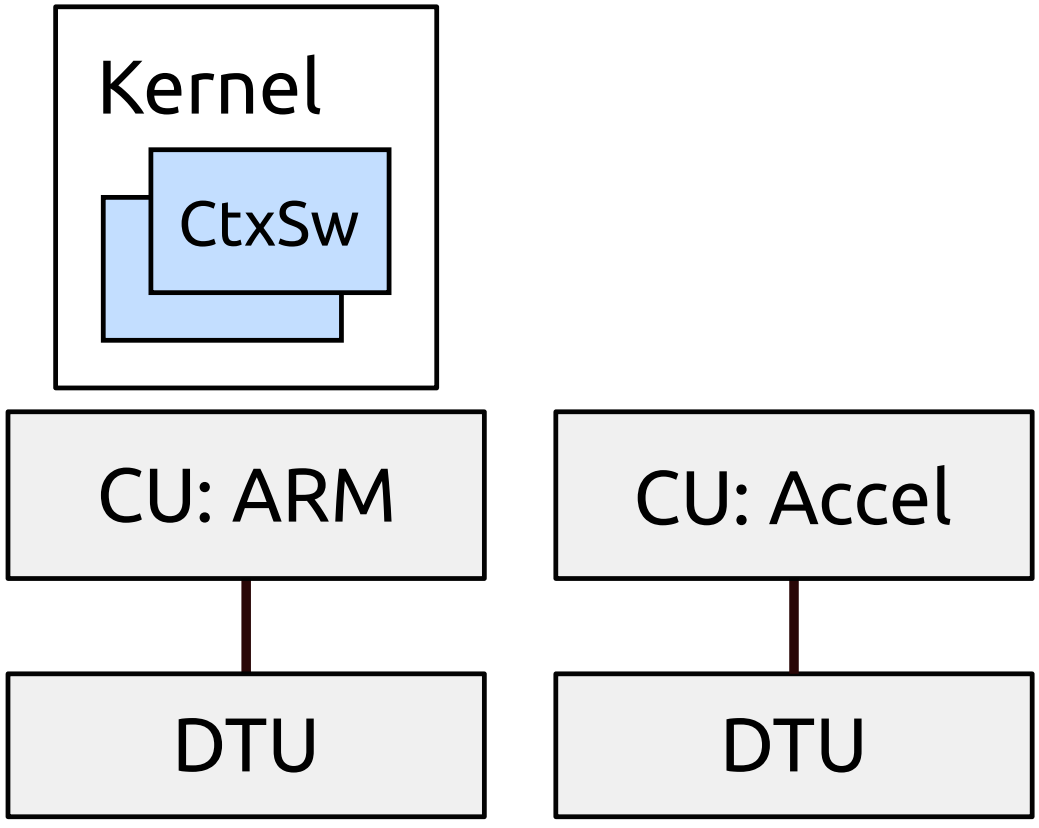
- M^3 allows to use accelerators from the shell:
  `preproc | accel1 | accel2 > output.dat`
- Shell connects the EPs according to stdin/stdout
- Accelerators work autonomously afterwards
- Requires about 30 additional lines in the shell

Demo

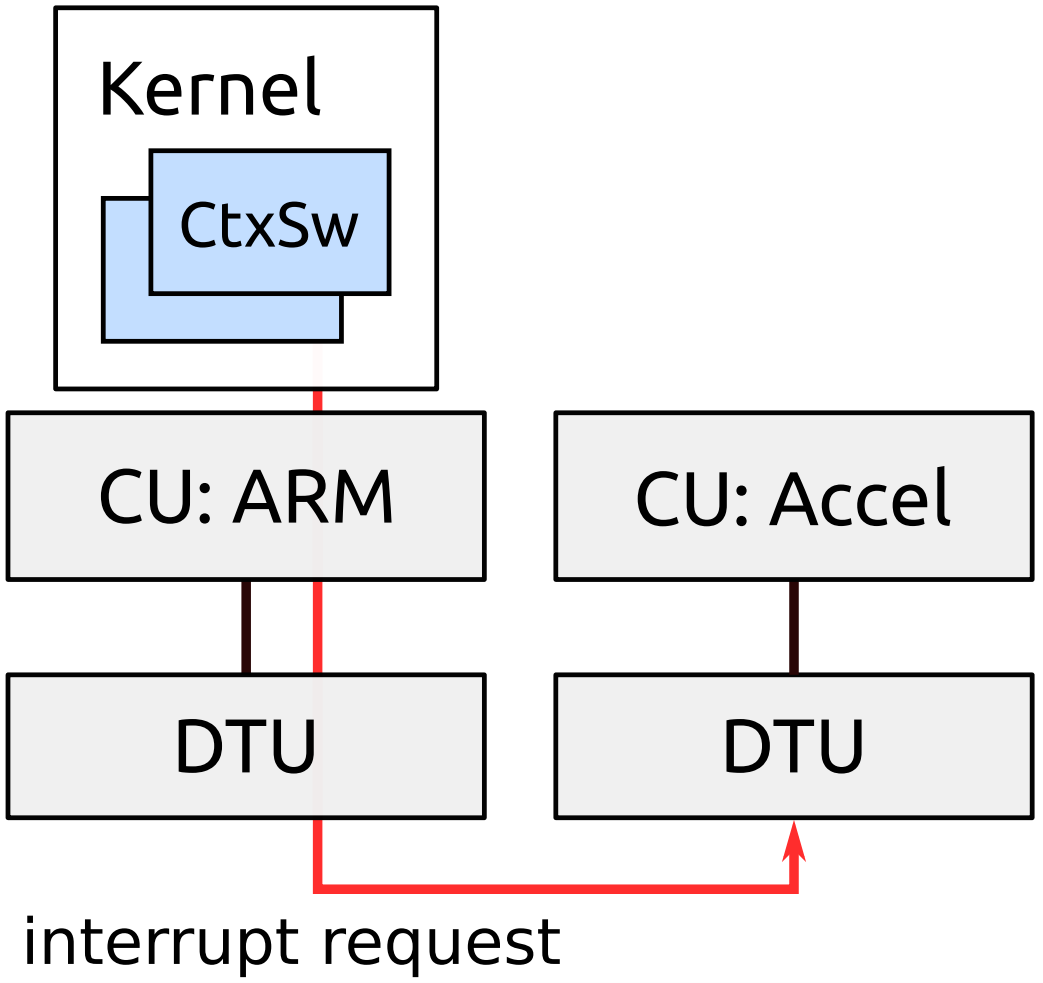
Demo

Outline

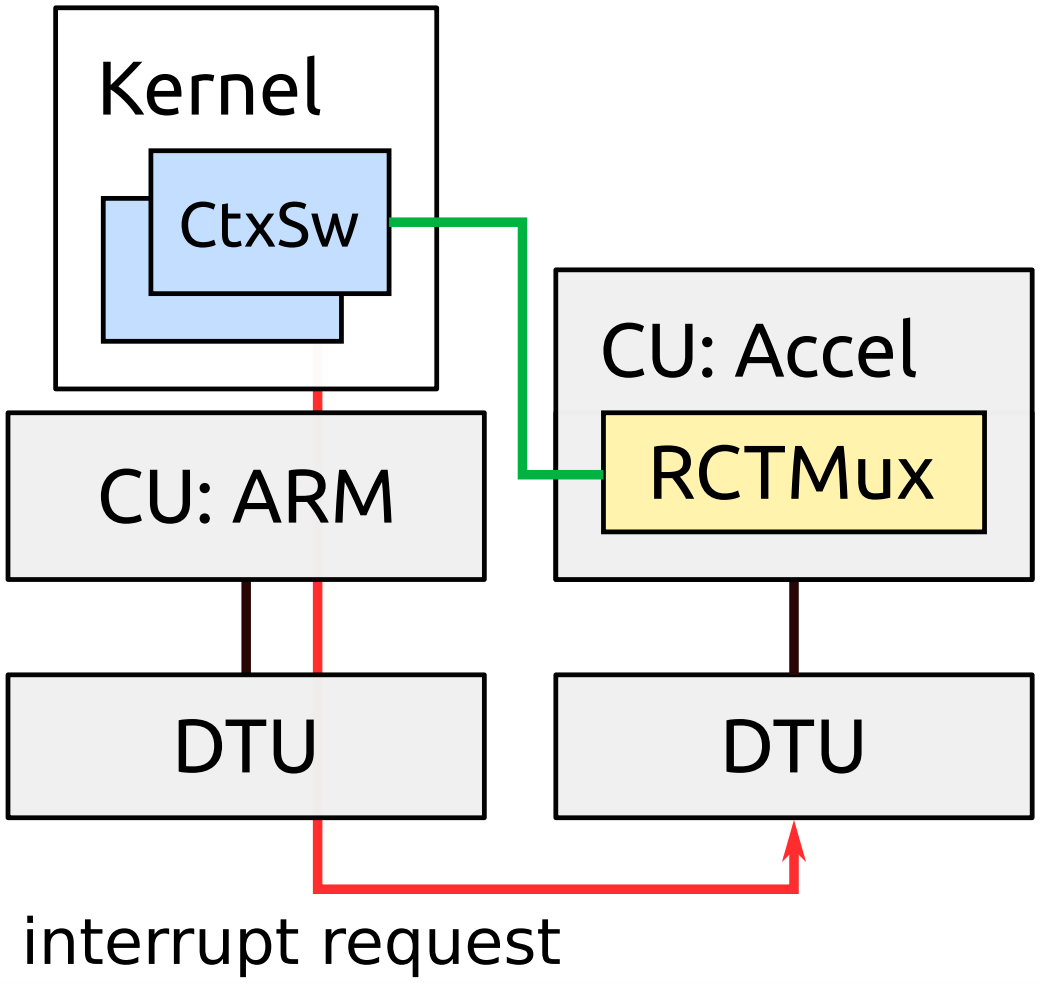
Context Switching



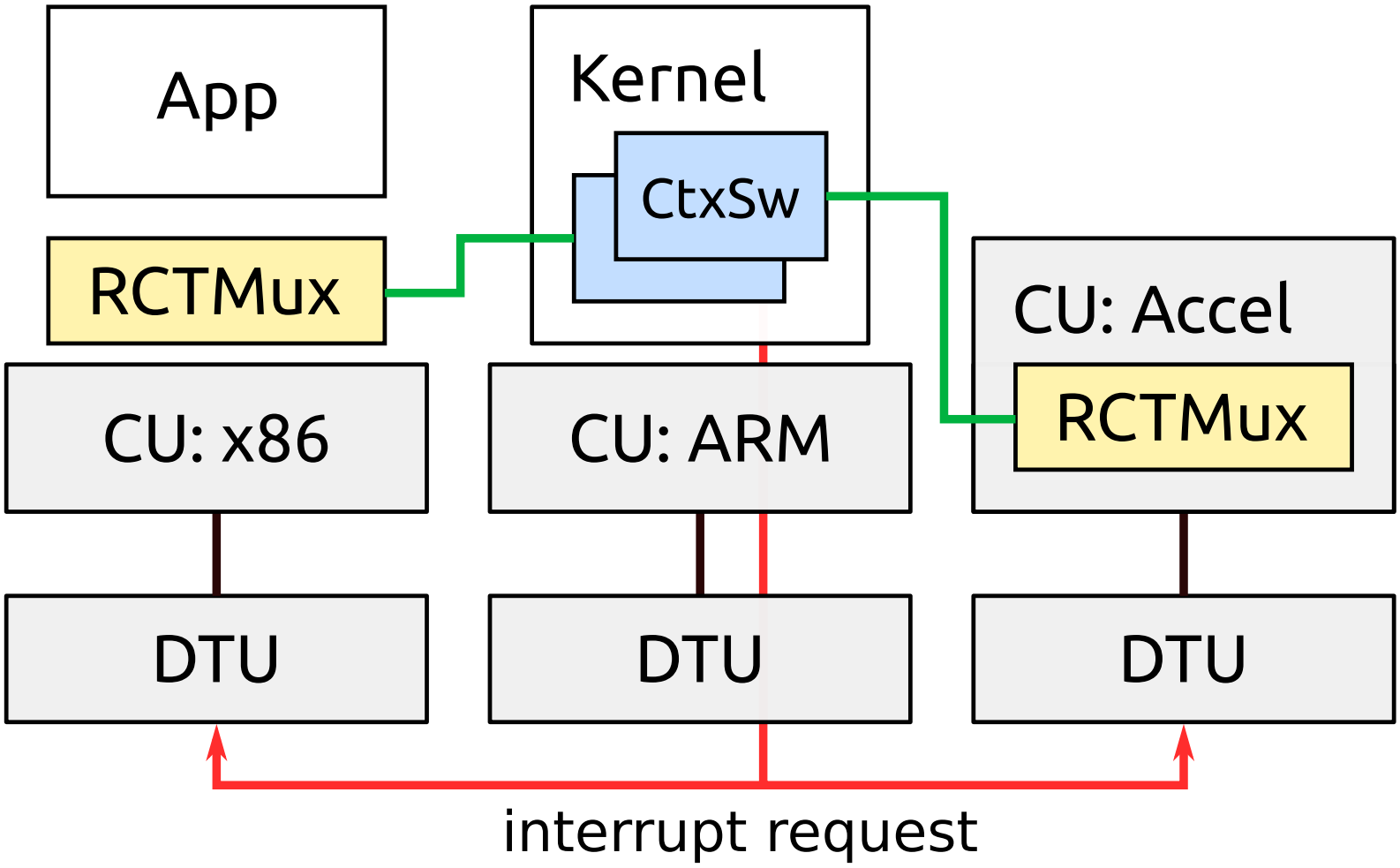
Context Switching



Context Switching



Context Switching



Communication with Suspended VPEs

- If a VPE is suspended, communication channels stay valid
- Each DTU knows the ID of the currently running VPE
- Messages contain the target VPE ID
- If these do not match, DTU responds with an error
- In this case, the sender lets the kernel *forward* the message
- Kernel will resume the VPE and afterwards transmit the message on behalf of the sender

Computing vs. Idling

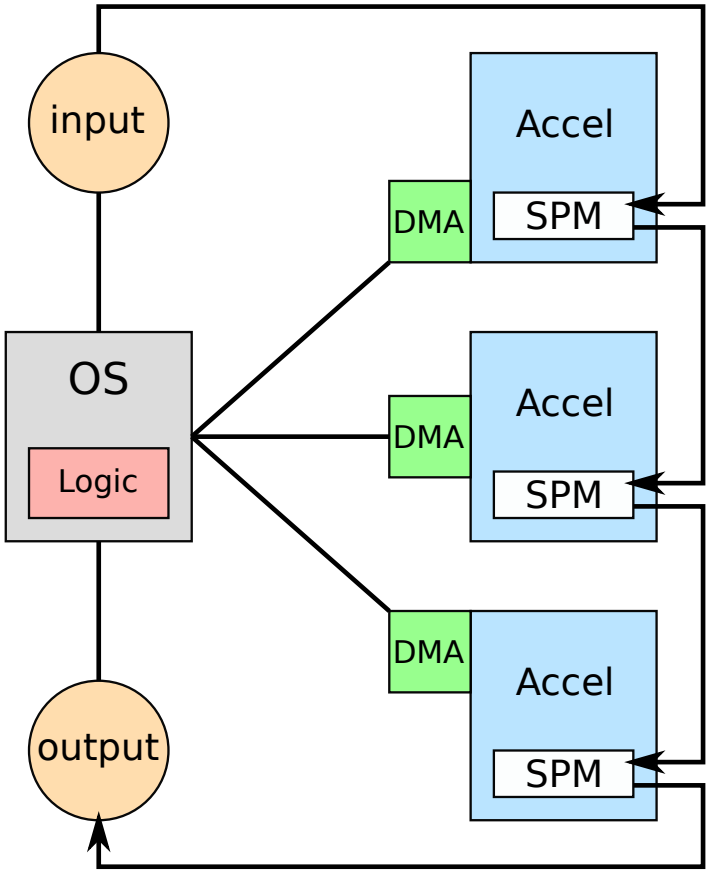
- How does the kernel know what VPEs are doing?
- VPEs communicate directly, without involving the kernel and wait for the next msg via DTU
- The kernel asks VPEs to report idling, if other VPEs are ready
- As soon as a VPE starts to idle, it checks whether it should report that
- If so, the VPE waits for the time chosen by the kernel and performs a system call afterwards

Outline

Experimental Setup

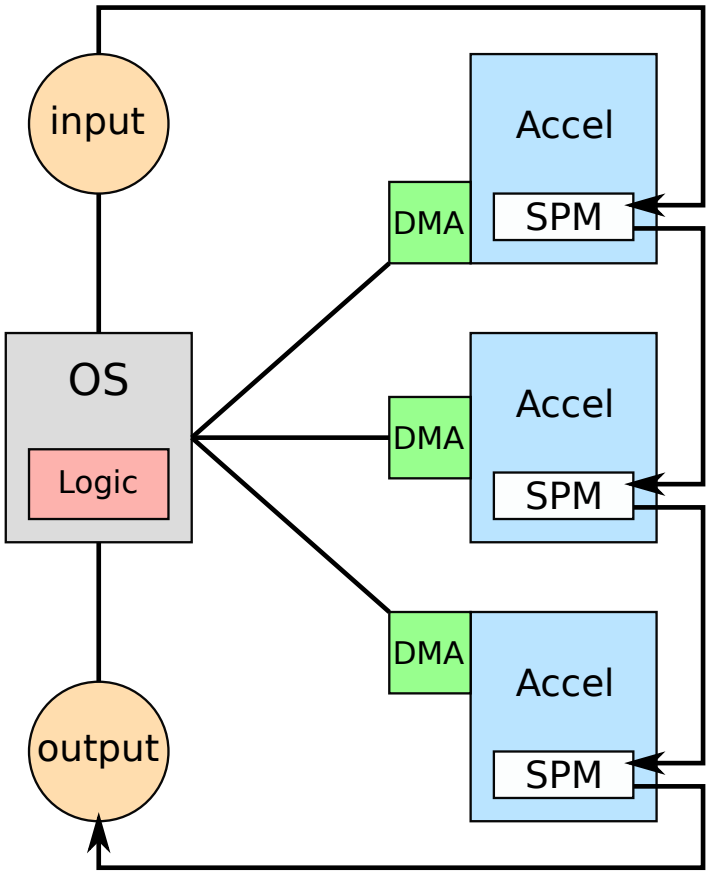
- Evaluation platform is gem5
- Each general-purpose PE has x86_64 core @ 3GHz, 32+32 KiB L1 cache, 256 KiB L2 cache
- Accelerator PEs are clocked with 1GHz
- DRAM (`DDR3_1600_8x8`) clocked with 1GHz
- Short running, but representative benchmarks

Accelerator Chaining – Variants

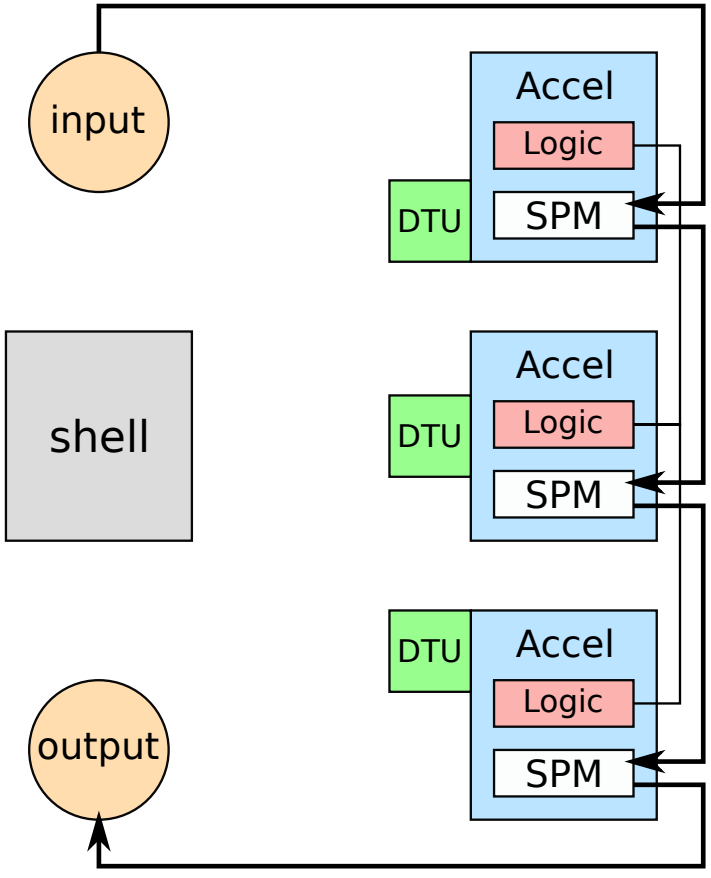


Assisted

Accelerator Chaining – Variants

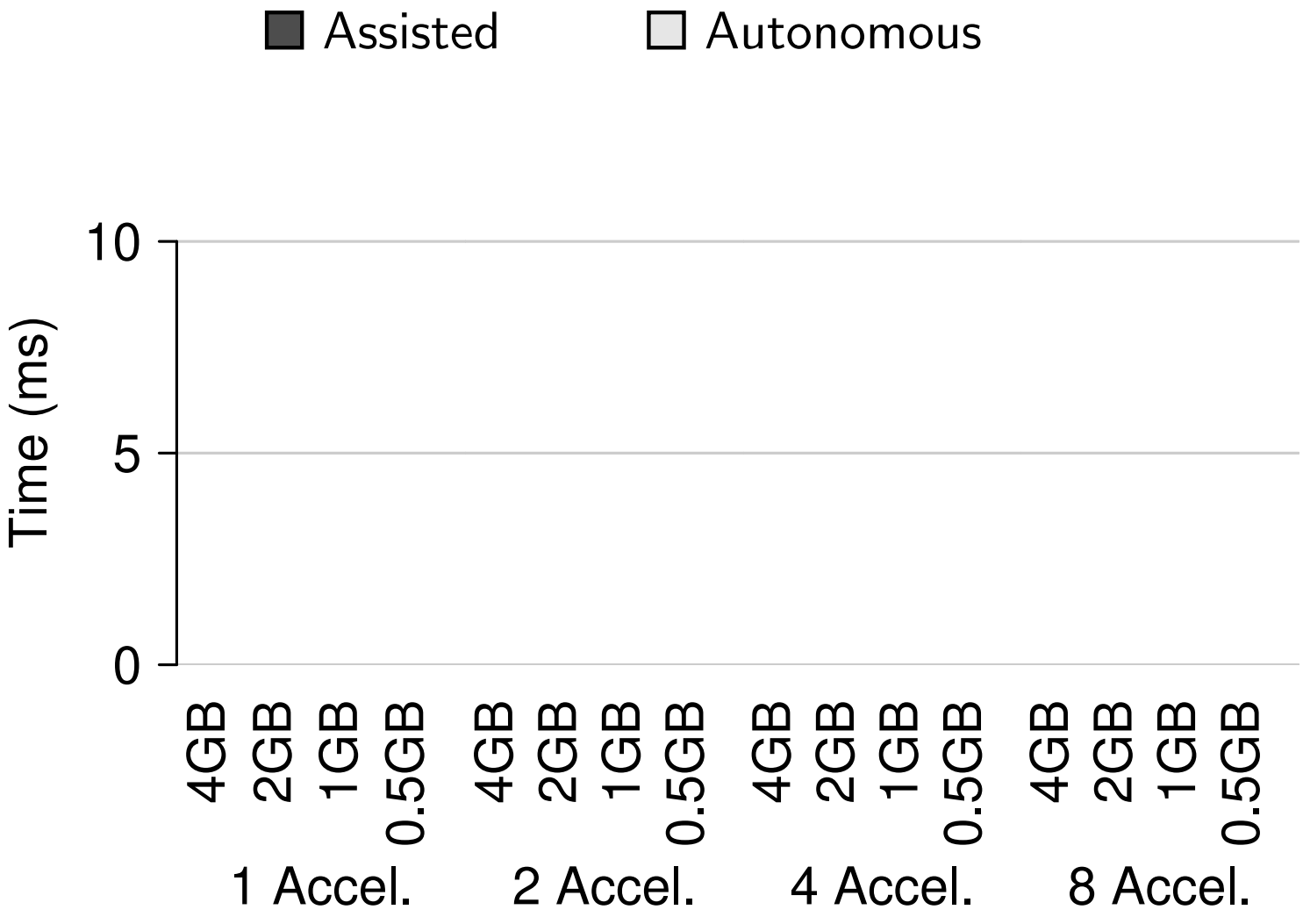


Assisted

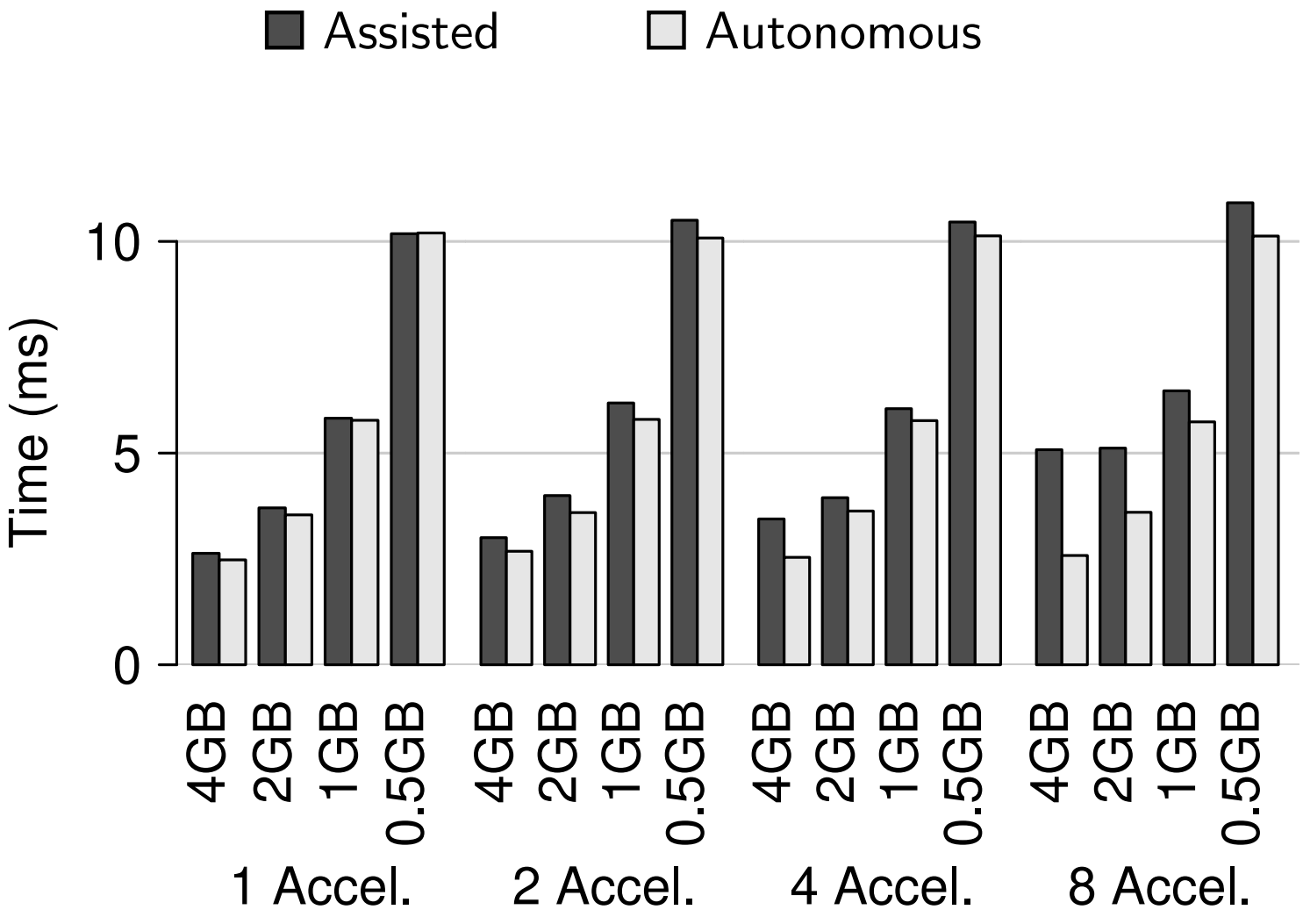


Autonomous

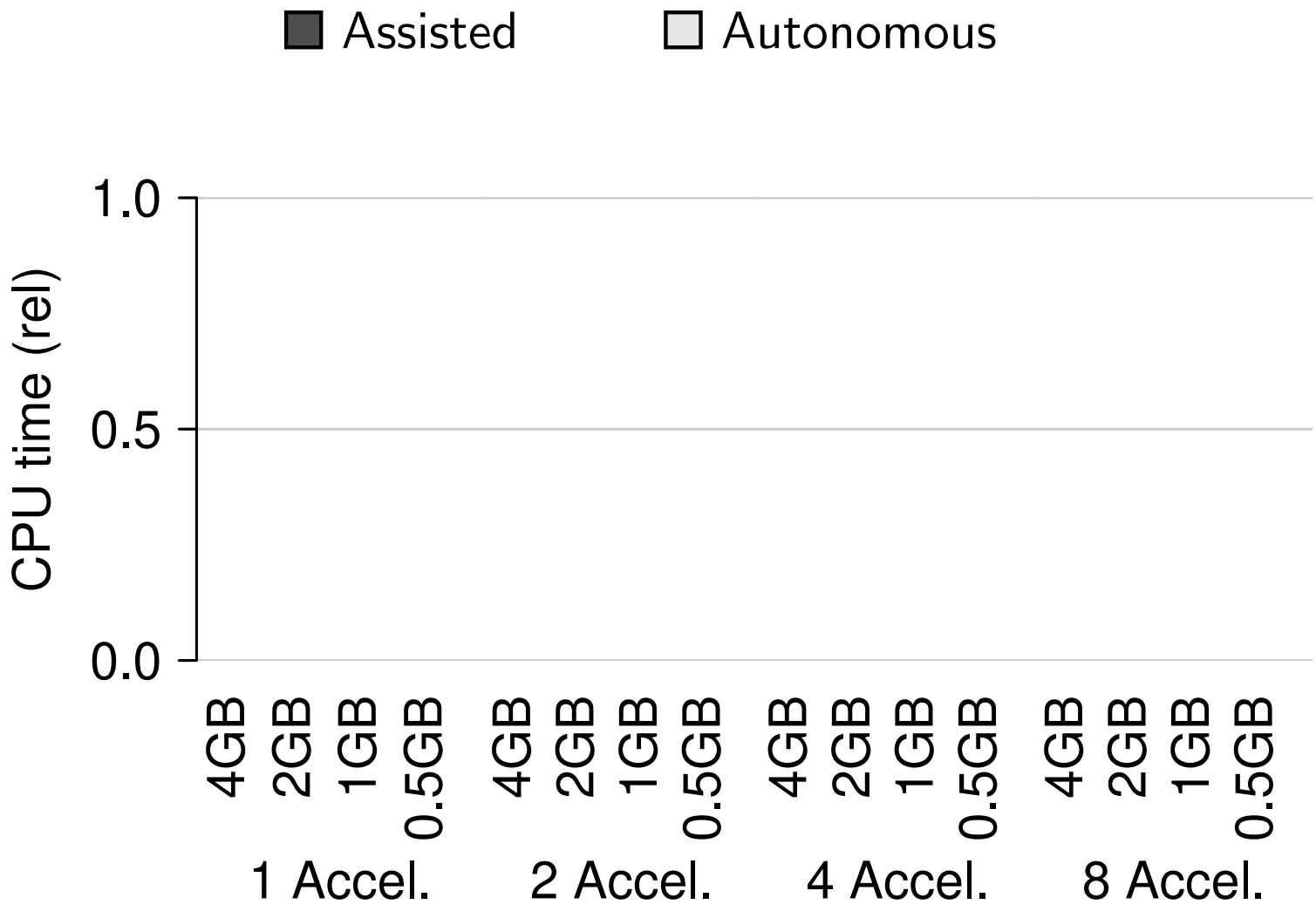
Accelerator Chaining – Results



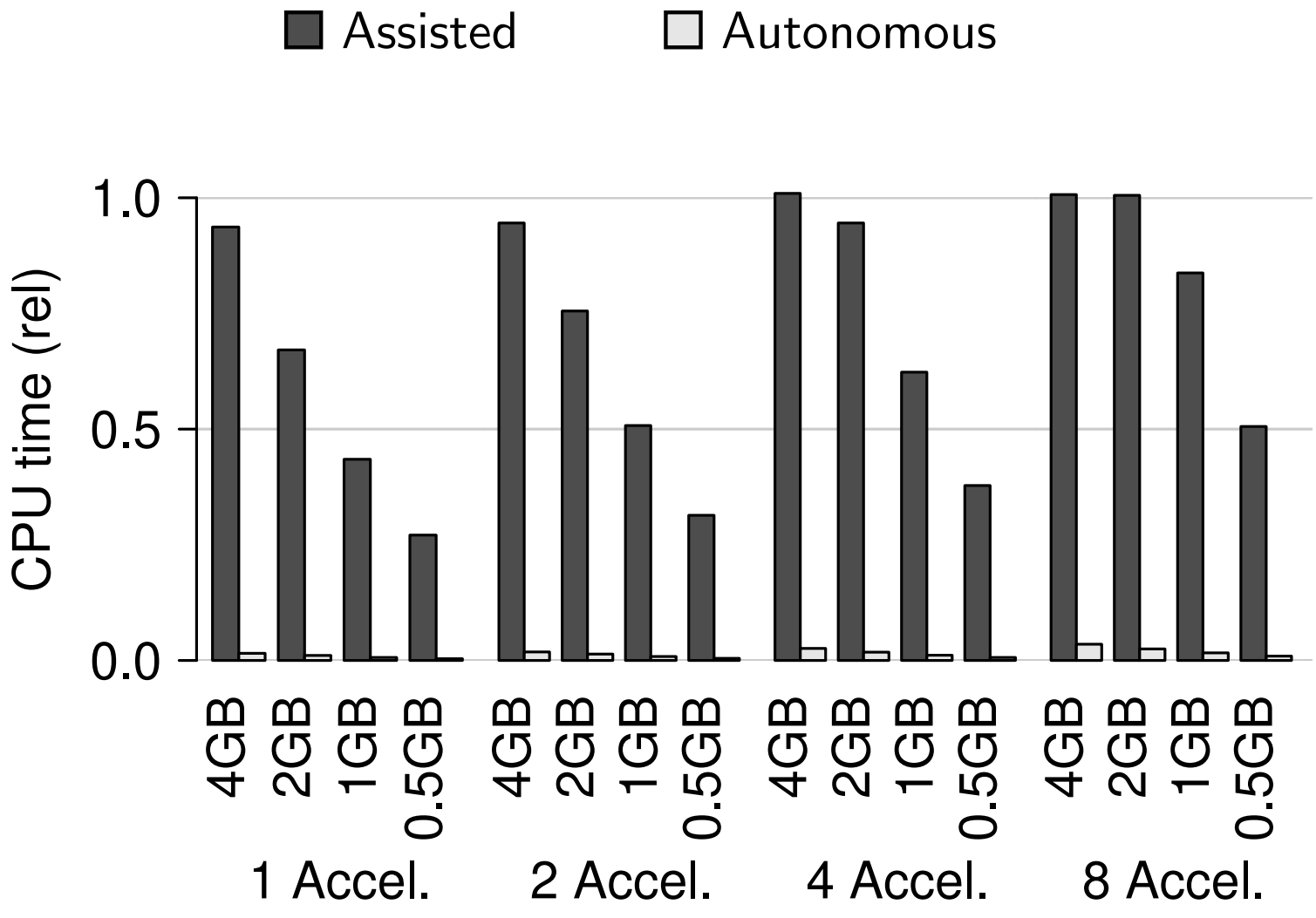
Accelerator Chaining – Results



Accelerator Chaining – Results



Accelerator Chaining – Results

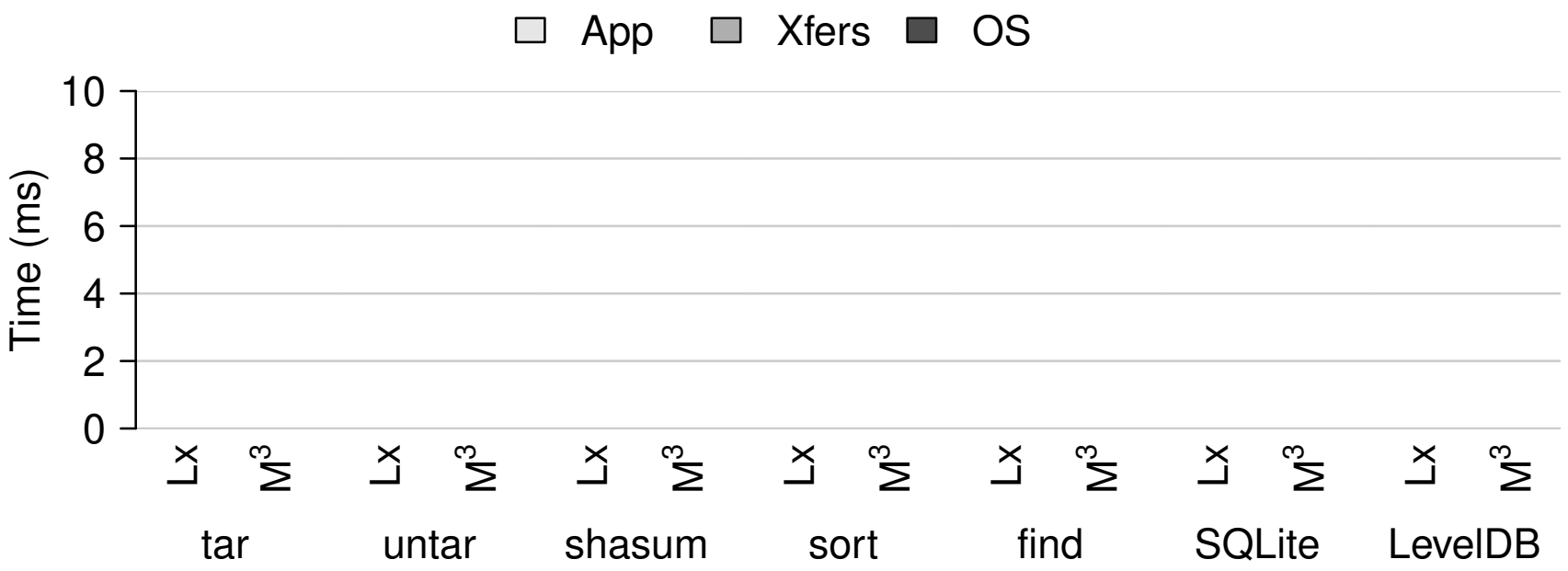


Application Performance

- Comparison to Linux 4.10, using tmpfs
- Traced obtained on Linux and replayed on M^3
- M^3: 3 user PEs; Linux: 1 core (same config)

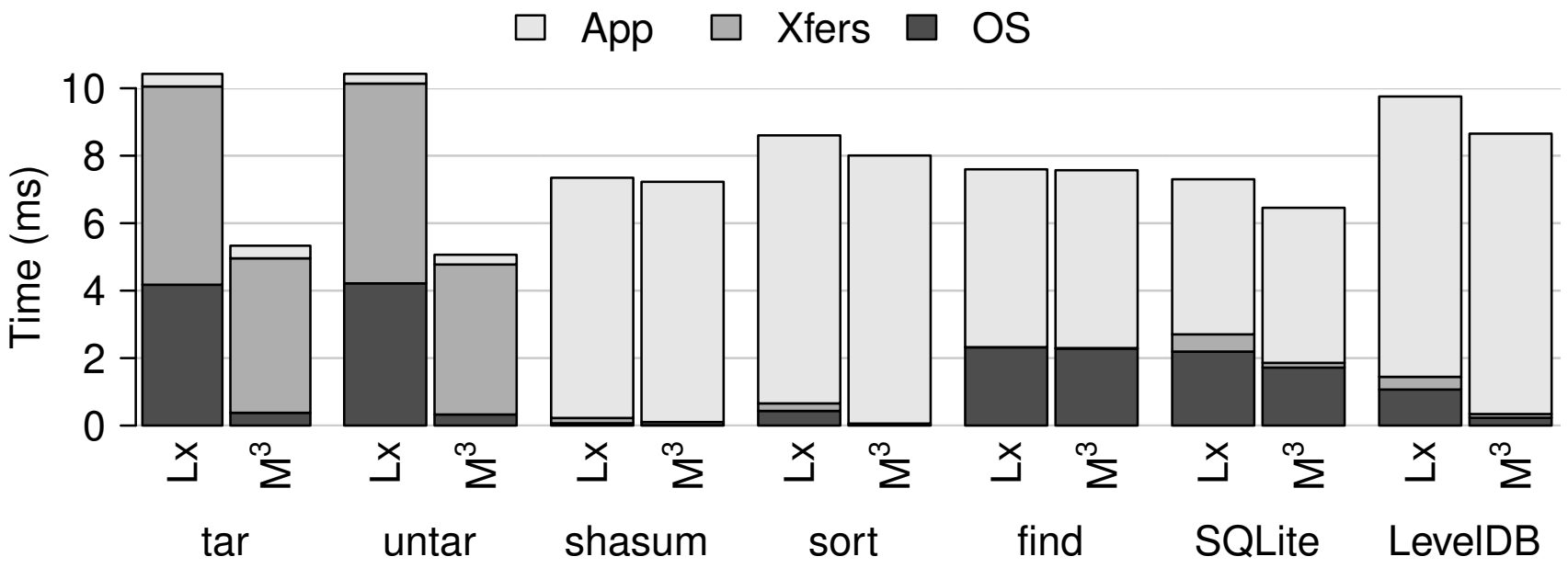
Application Performance

- Comparison to Linux 4.10, using tmpfs
- Traced obtained on Linux and replayed on M³
- M³: 3 user PEs; Linux: 1 core (same config)



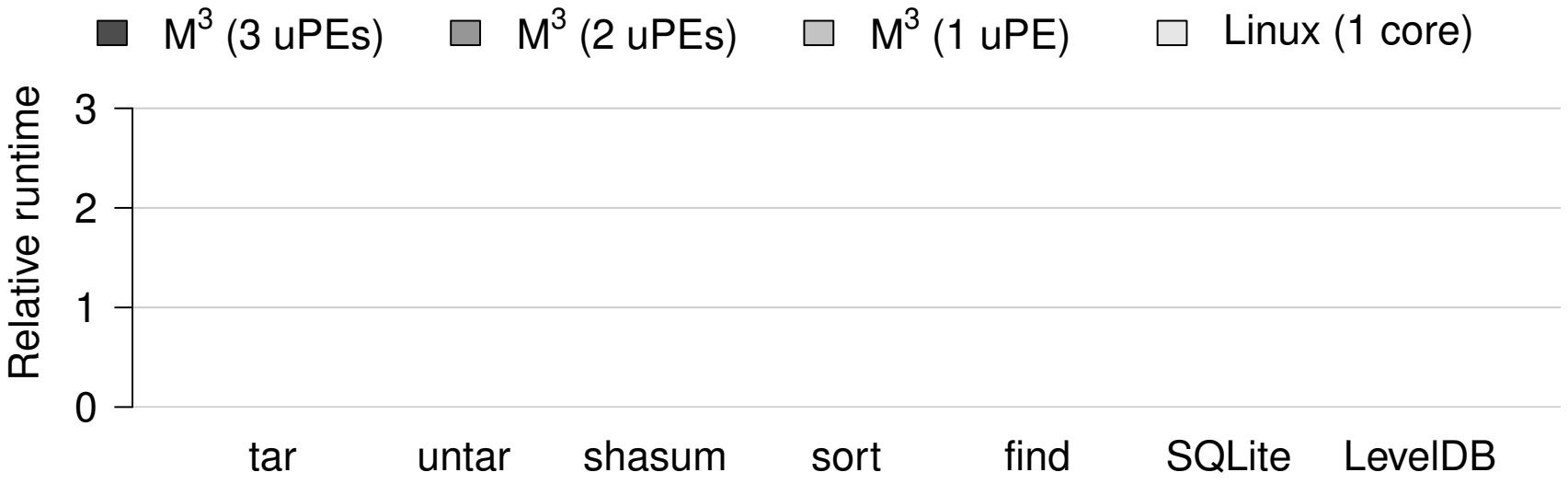
Application Performance

- Comparison to Linux 4.10, using tmpfs
- Traced obtained on Linux and replayed on M^3
- M^3: 3 user PEs; Linux: 1 core (same config)



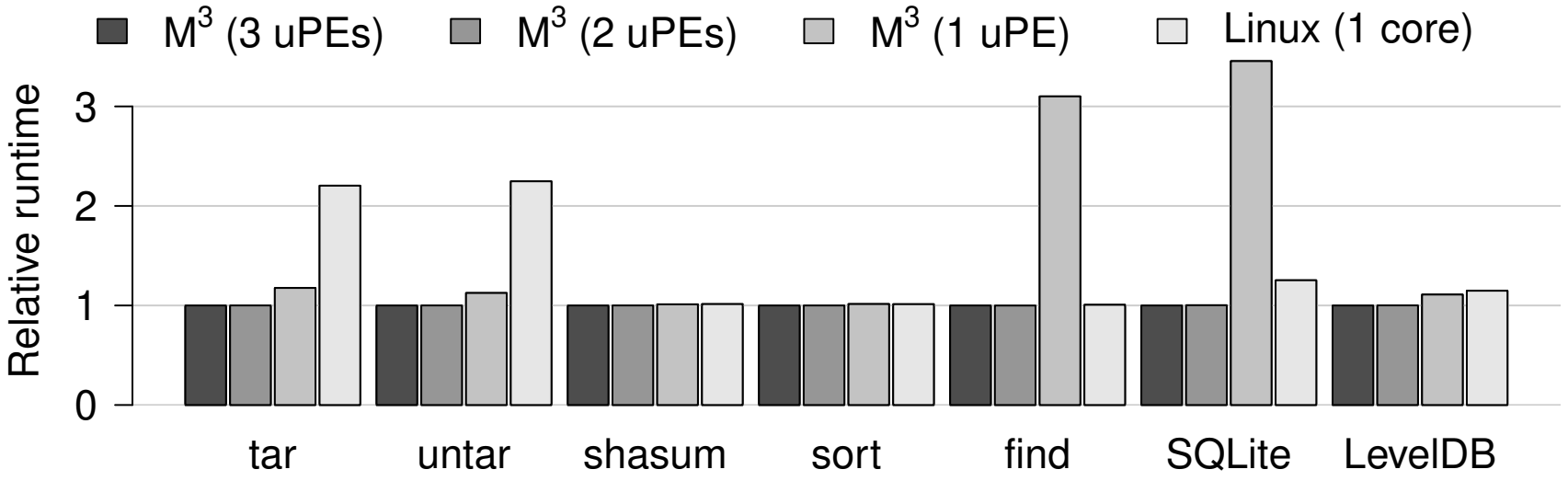
PE Sharing

- 3 user PEs: pager, m3fs, app (baseline)
- 2 user PEs: pager+m3fs, app
- 1 user PEs: pager+m3fs+app



PE Sharing

- 3 user PEs: pager, m3fs, app (baseline)
- 2 user PEs: pager+m3fs, app
- 1 user PEs: pager+m3fs+app



Ongoing Work

- Multiple instances of the kernel/services (by Matthias Hille)
- Improved network support (by Georg Kotheimer)
- Extension of m3fs for storage devices (by Sebastian Reimers)

Conclusion

- M³ uses a hardware/software co-design
- DTU introduces common interface for all CUs
- Allows to treat all CUs as first-class citizens
- Access to OS services for all CUs
- M³ uses the same concepts for all CUs
- Allows simple management of complex systems