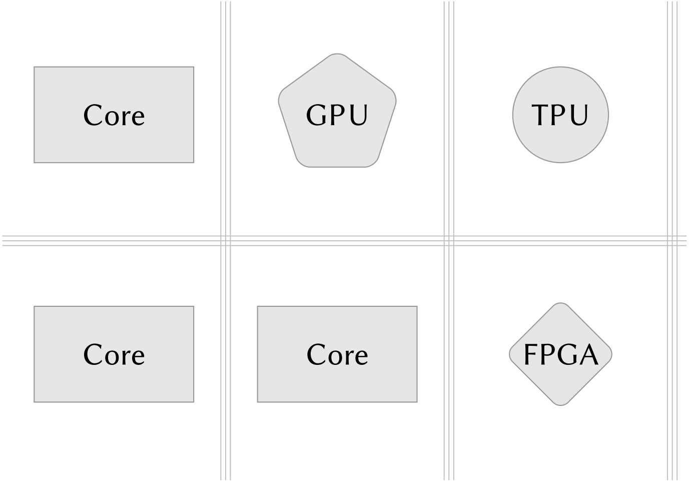
barkhauseninstitut.org

Efficient and Scalable Core Multiplexing with M^{3}v

Nils Asmussen, Sebastian Haas, Carsten Weinhold, Till Miemietz, Michael Roitzsch

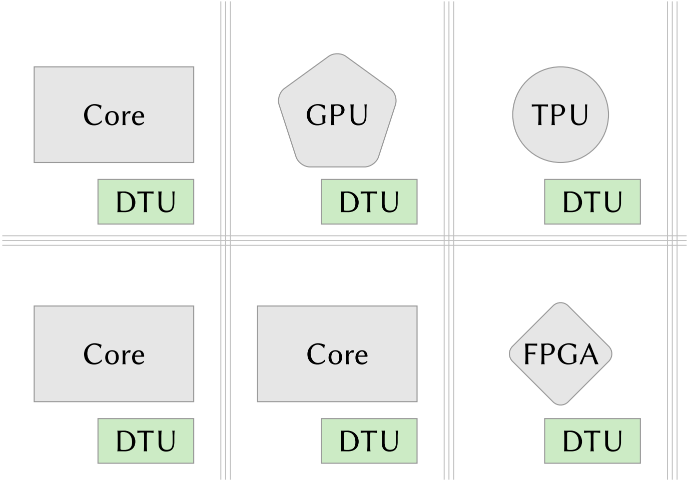
Fachgruppentreffen, Erlangen, 19.09.2022

M3 System Architecture [1]



[1] Asmussen et al.; M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M3 System Architecture [1]

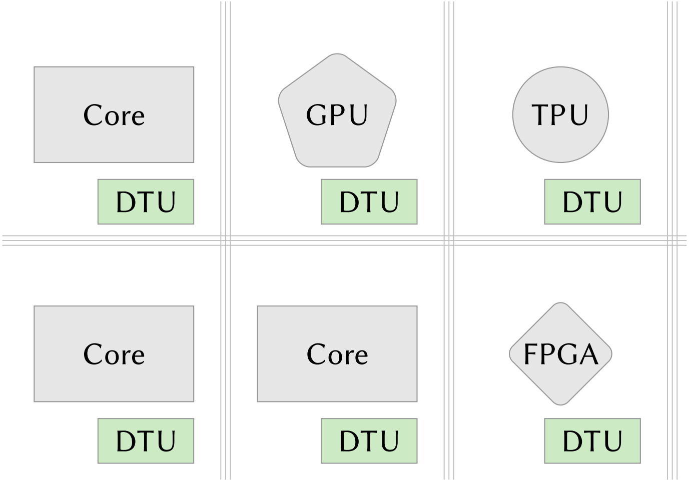


Key ideas:

- DTU as new hardware component

[1] Asmussen et al.; M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M^3 System Architecture [1]

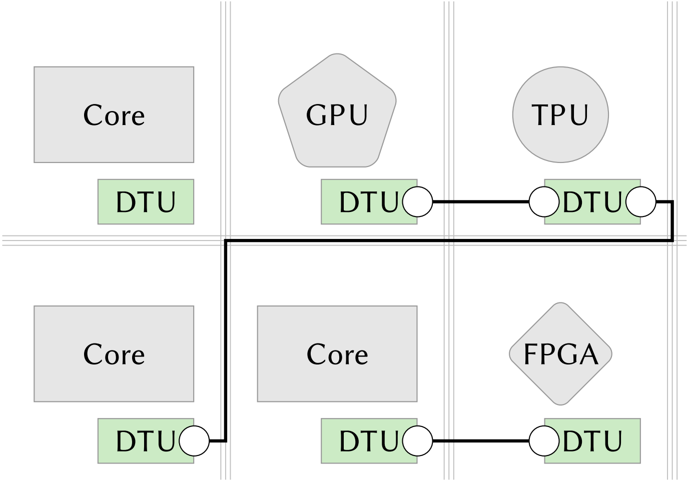


Key ideas:

- DTU as new hardware component
- Tiles are isolated by default

[1] Asmussen et al.; M^3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M^3 System Architecture [1]

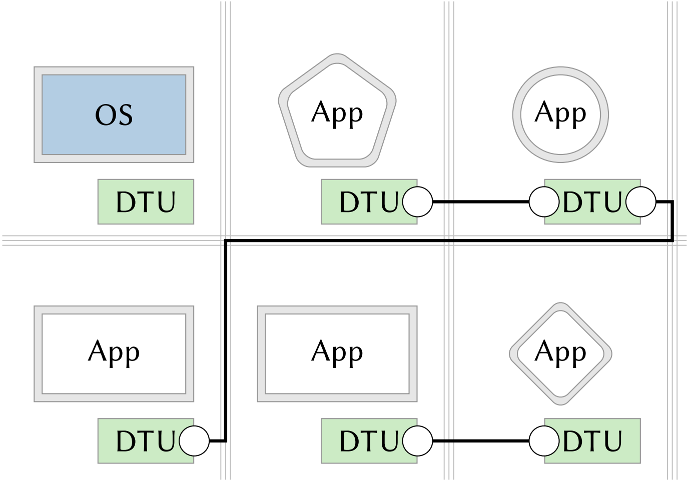


Key ideas:

- DTU as new hardware component
- Tiles are isolated by default

[1] Asmussen et al.; M^3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M^3 System Architecture [1]

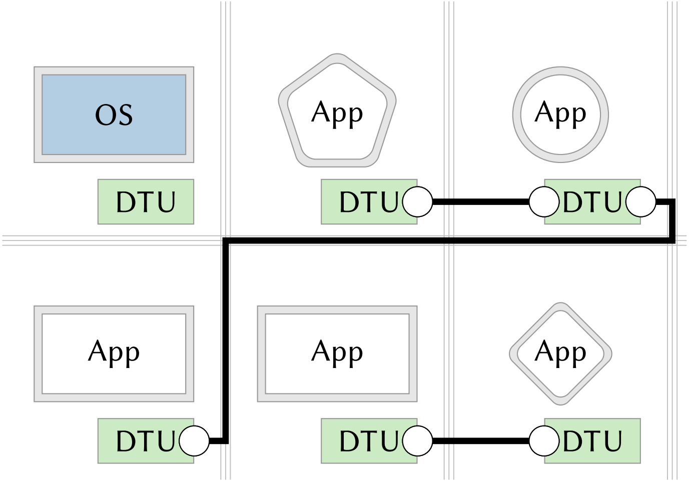


Key ideas:

- DTU as new hardware component
- Tiles are isolated by default
- OS on dedicated tile

[1] Asmussen et al.; M^3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M³ System Architecture [1]

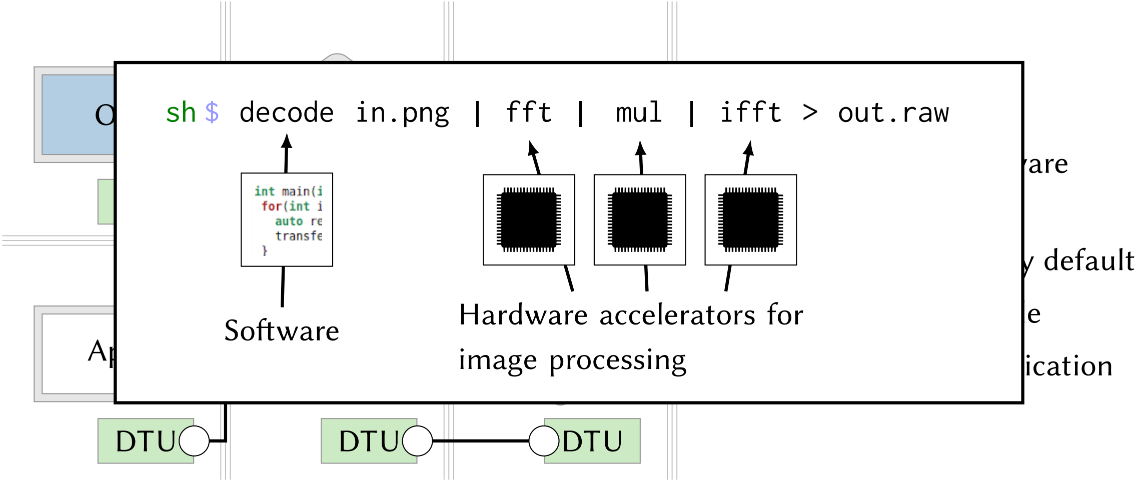


Key ideas:

- DTU as new hardware component
- Tiles are isolated by default
- OS on dedicated tile
- Fast-path communication

[1] Asmussen et al.; M³: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

M^3 System Architecture [1]



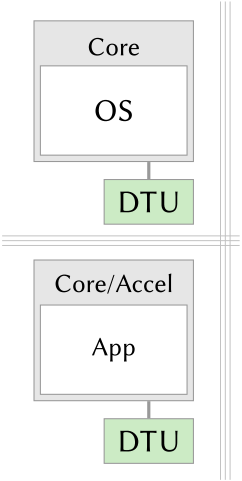
```
sh $ decode in.png | fft | mul | ifft > out.raw
```

```
int main(i
 for(int i
   auto re
   transfe
 }
```

[1] Asmussen et al.; M^3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

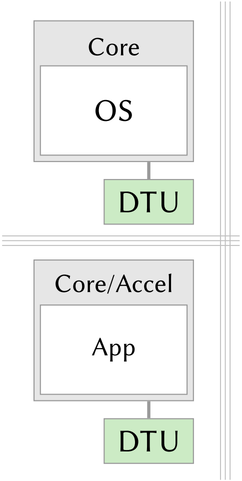
Comparison of Core Multiplexing Approaches

M^3 (ASPLOS'16)

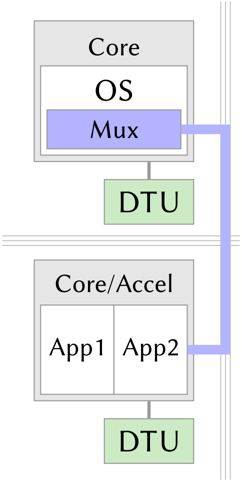


Comparison of Core Multiplexing Approaches

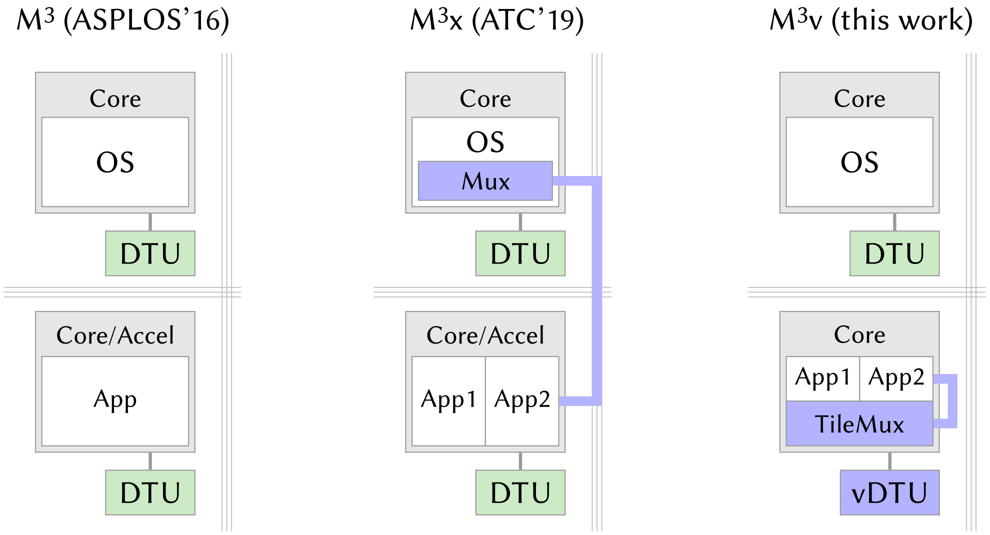
M^3 (ASPLOS’16)



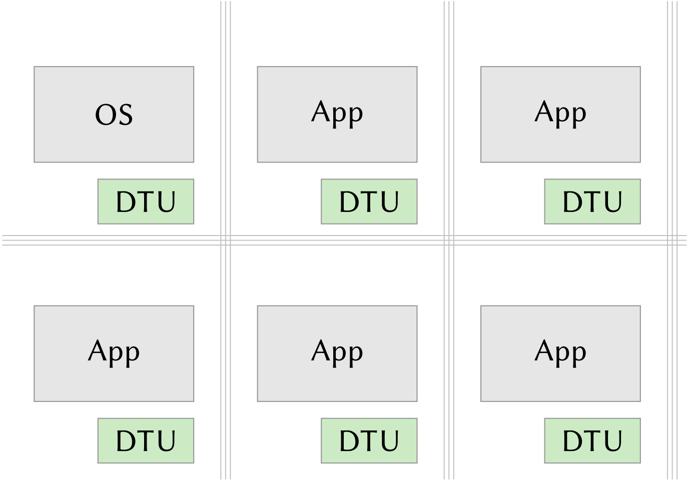
M^{3}x (ATC’19)



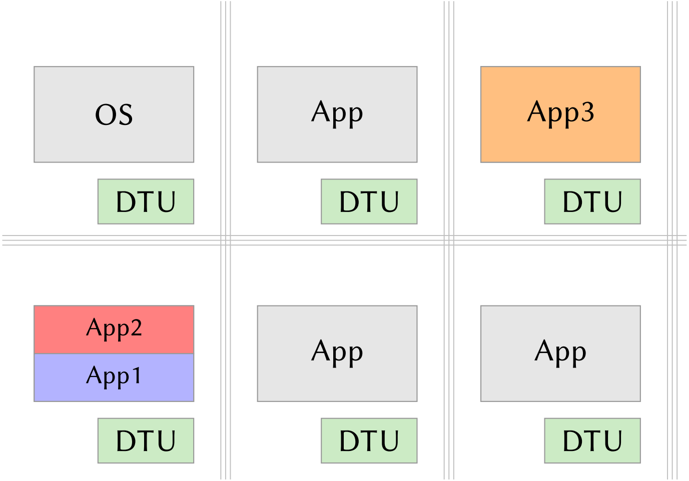
Comparison of Core Multiplexing Approaches



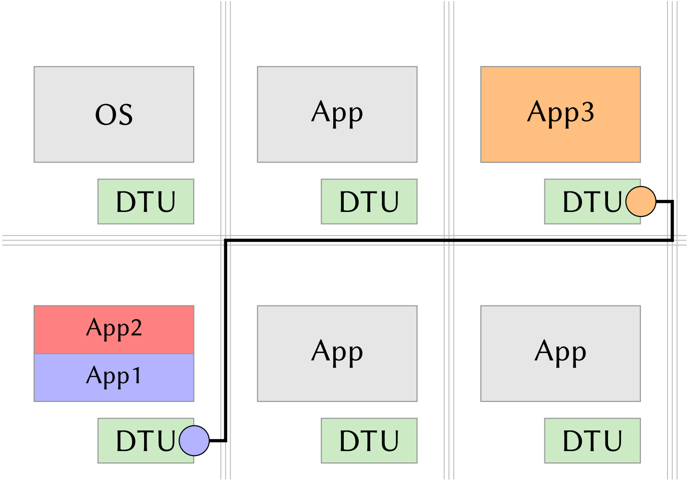
Core Multiplexing vs Fast-Path Communication



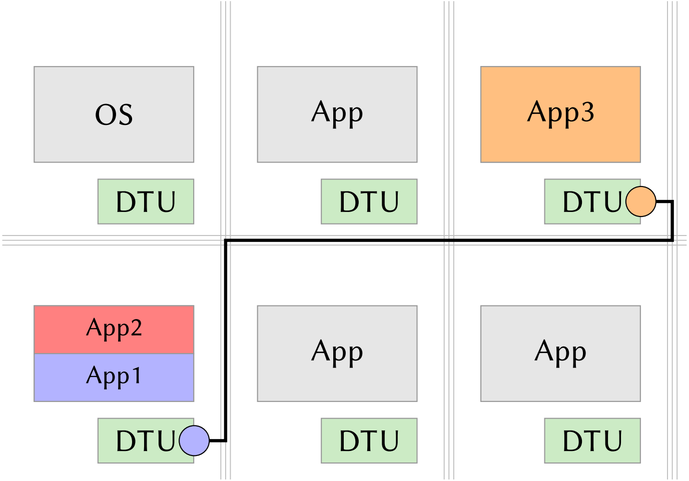
Core Multiplexing vs Fast-Path Communication



Core Multiplexing vs Fast-Path Communication

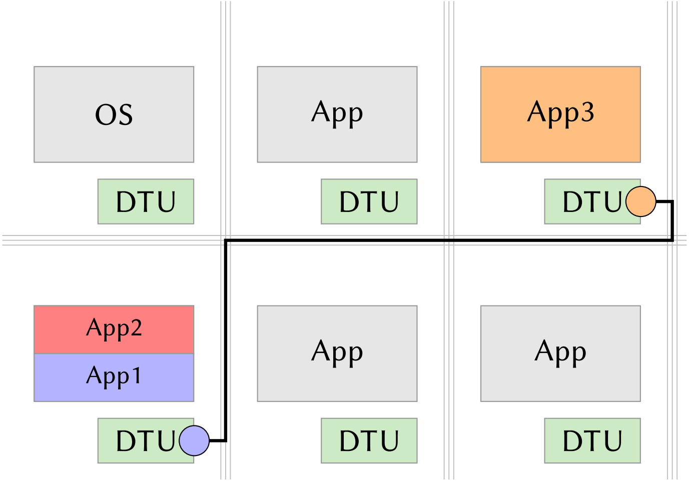


Core Multiplexing vs Fast-Path Communication



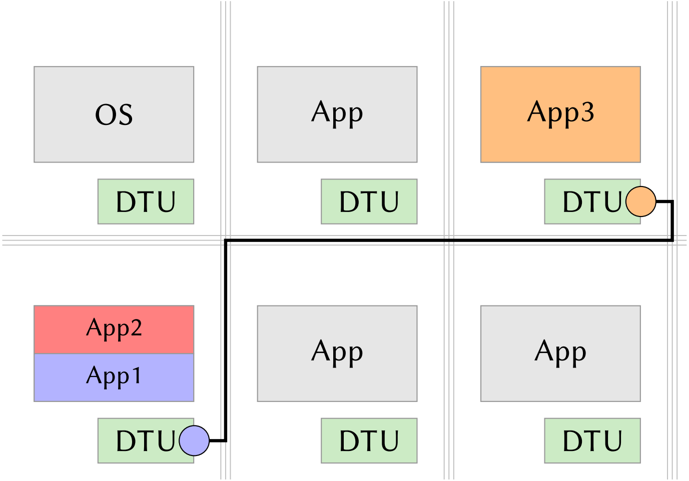
- Suspend App1 until new message, schedule App2

Core Multiplexing vs Fast-Path Communication



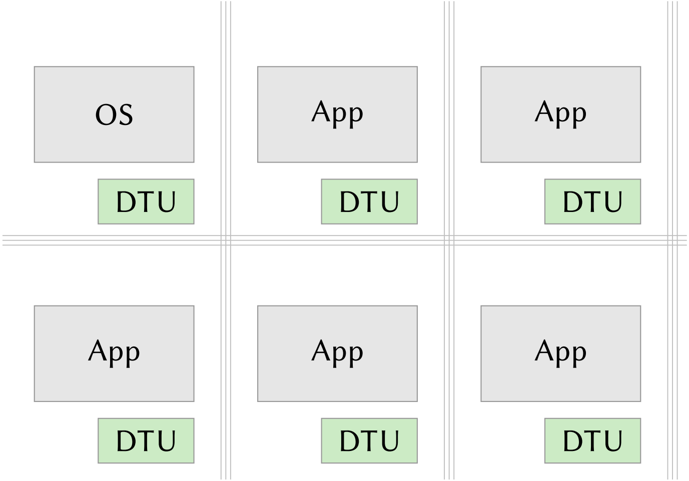
- Suspend App1 until new message, schedule App2
- Resume App1 upon new message

Core Multiplexing vs Fast-Path Communication

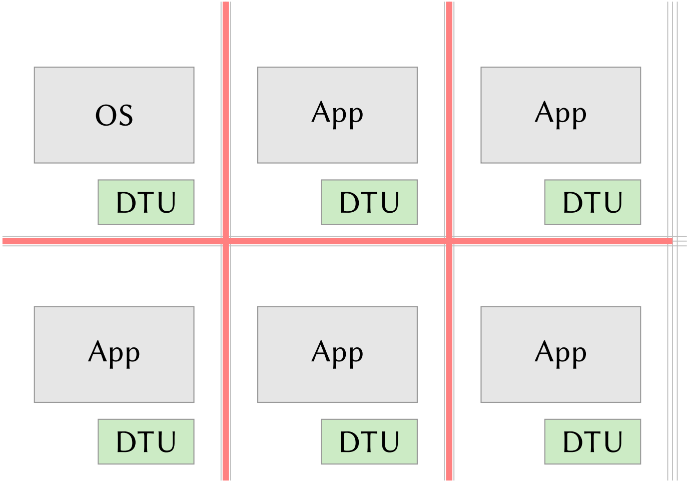


- Suspend App1 until new message, schedule App2
- Resume App1 upon new message
- Multiplexing conflicts with fast-path communication

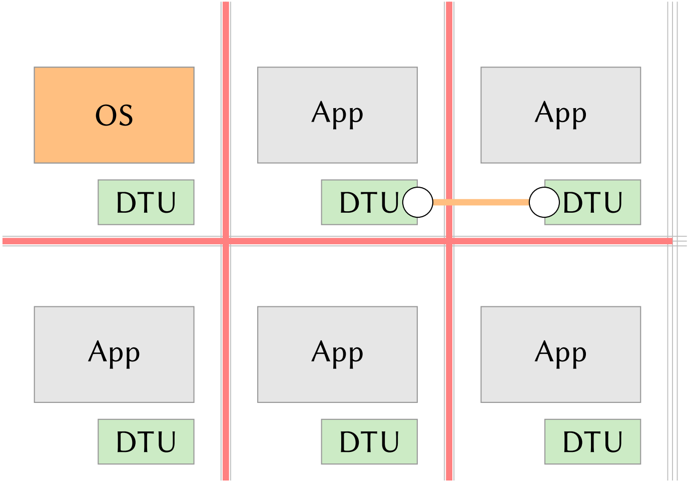
Strong Isolation between Tiles



Strong Isolation between Tiles

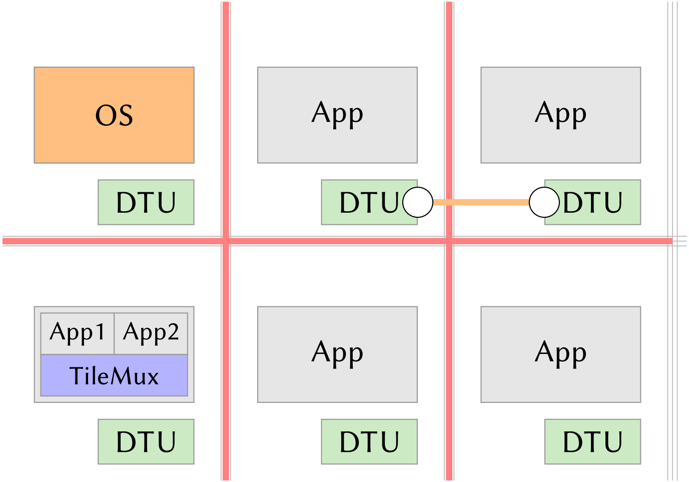


Strong Isolation between Tiles



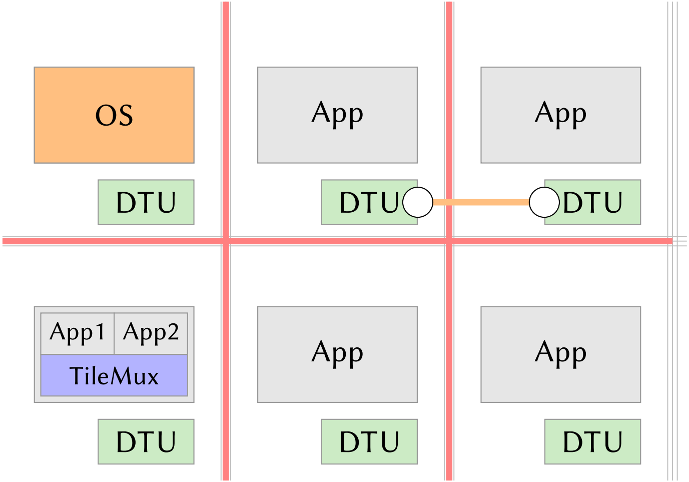
- Only the OS can provide access to tile-external resources

Strong Isolation between Tiles



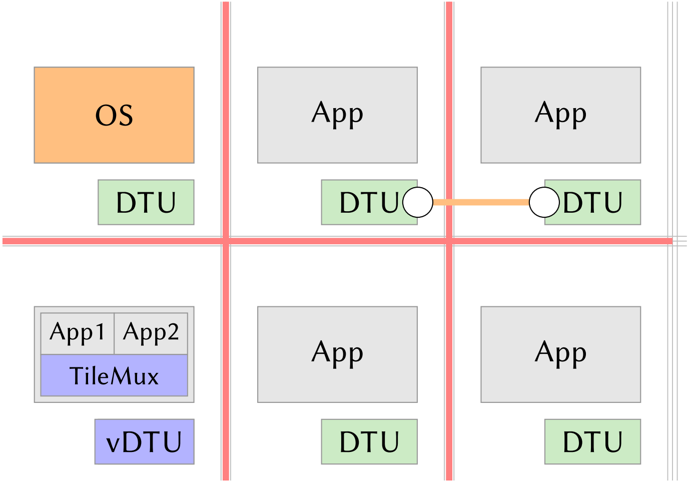
- Only the OS can provide access to tile-external resources
- Restoring DTU state provides access to **all** resources

Strong Isolation between Tiles



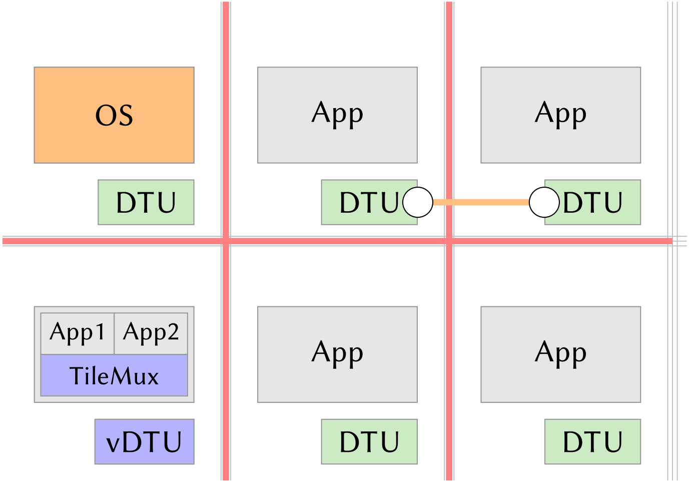
- Only the OS can provide access to tile-external resources
- Restoring DTU state provides access to **all** resources
- TileMux **must not** restore DTU state!

Strong Isolation between Tiles



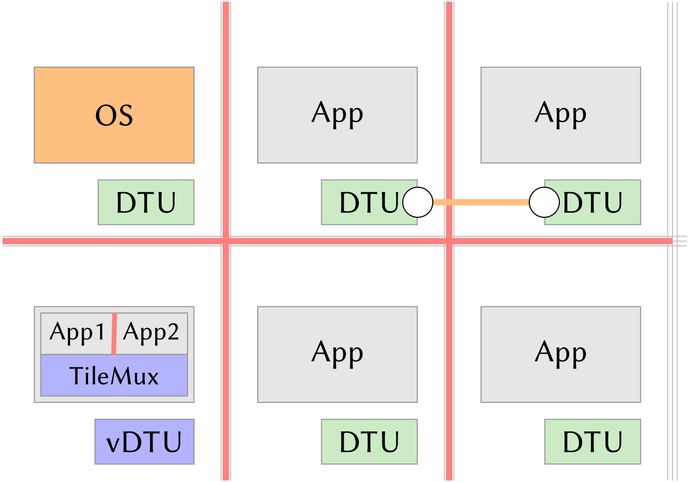
- Only the OS can provide access to tile-external resources
- Restoring DTU state provides access to **all** resources
- TileMux **must not** restore DTU state!

Strong Isolation between Tiles



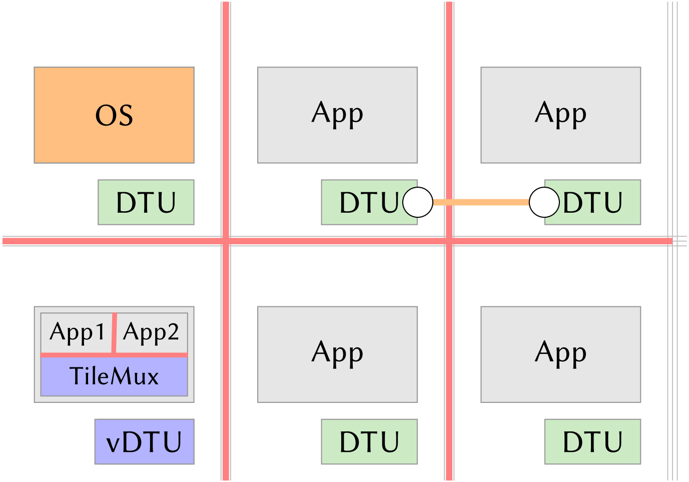
- M^{3*} provides better isolation than conventional architectures

Strong Isolation between Tiles



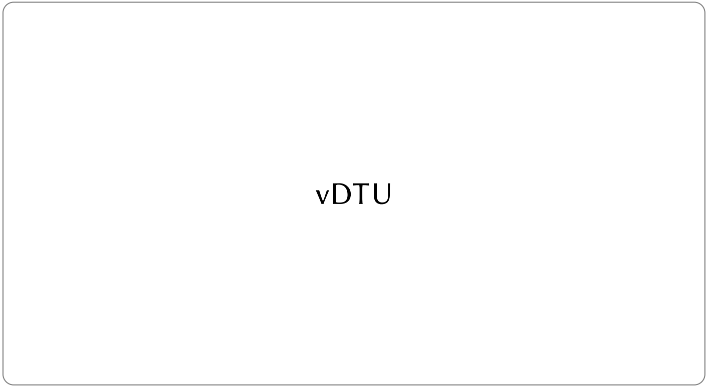
- M^{3*} provides better isolation than conventional architectures
- M^3x and M^3v trade some isolation for better resource utilization

Strong Isolation between Tiles



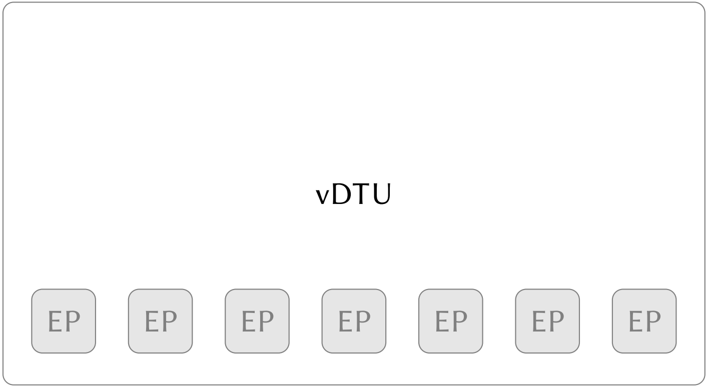
- M3* provides better isolation than conventional architectures
- M3x and M3v trade some isolation for better resource utilization
- M3v trades some more isolation for better efficiency

Virtualization of the DTU

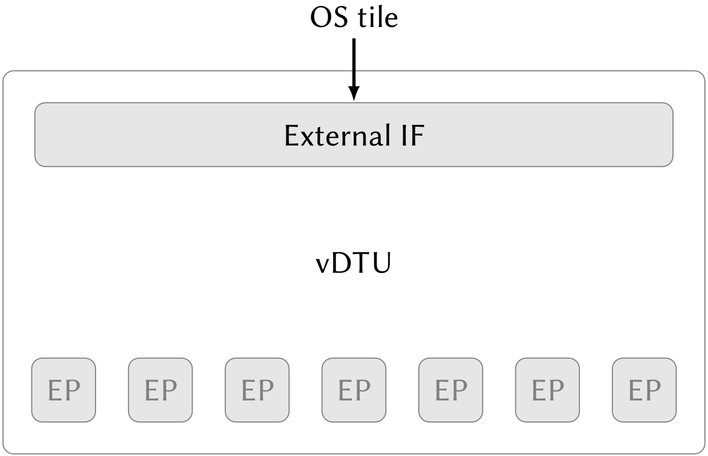


vDTU

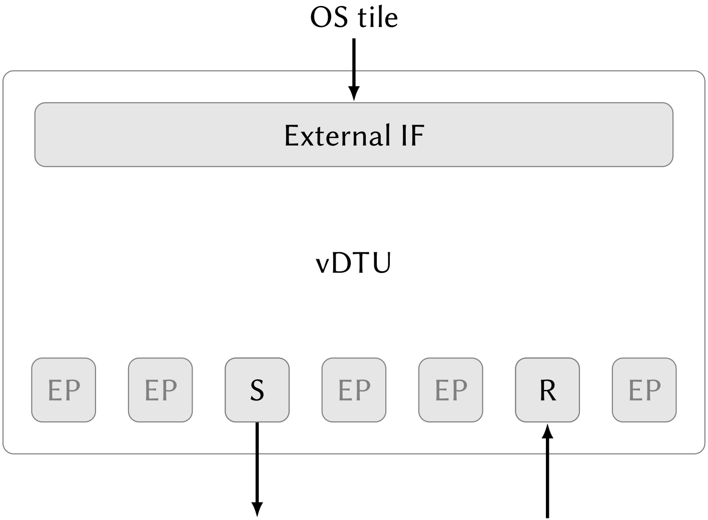
Virtualization of the DTU



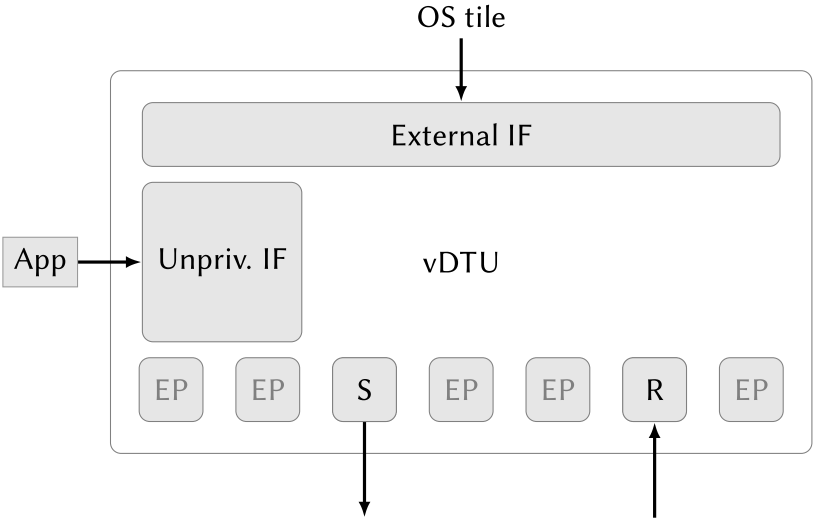
Virtualization of the DTU



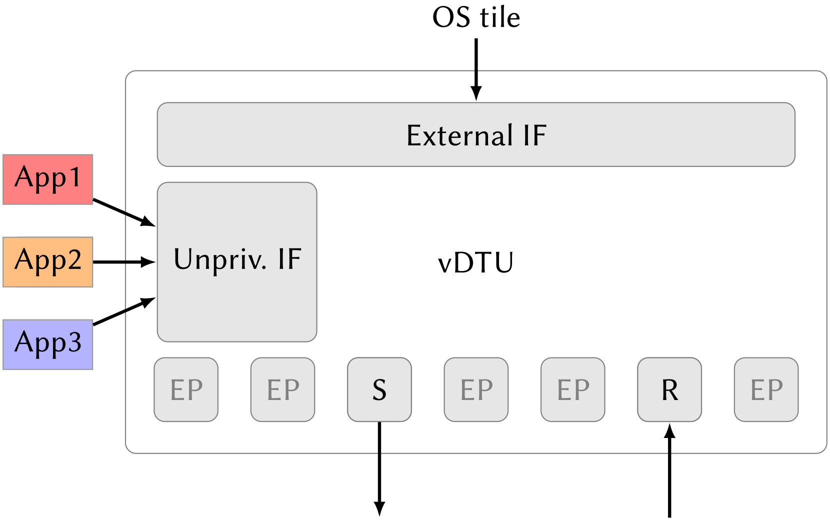
Virtualization of the DTU



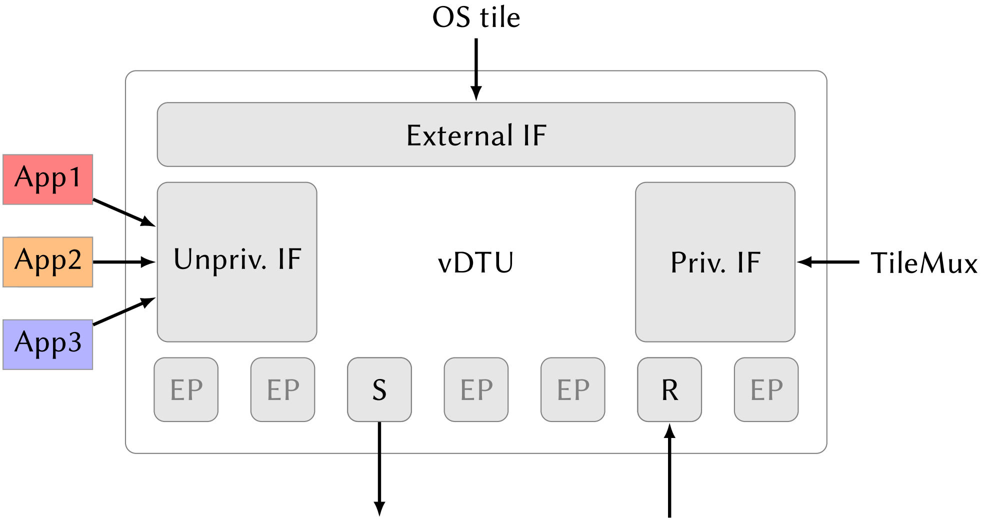
Virtualization of the DTU



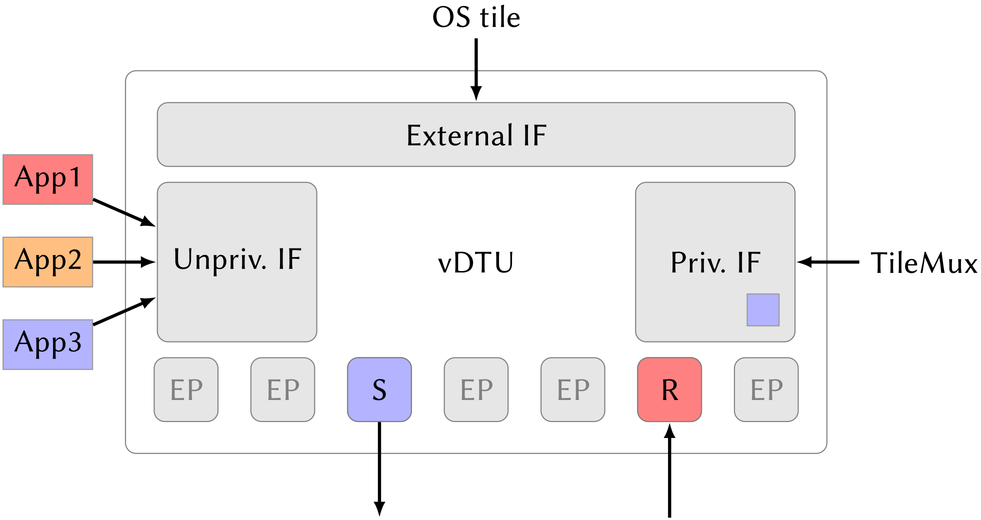
Virtualization of the DTU



Virtualization of the DTU



Virtualization of the DTU



Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run

Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run
- Applications fetch new messages directly from the vDTU

Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run
- Applications fetch new messages directly from the vDTU
- If there is none *and* other apps are ready, TileMux is used to block

Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run
- Applications fetch new messages directly from the vDTU
- If there is none *and* other apps are ready, TileMux is used to block
- Race condition: checking for new msgs and blocking (like lost-wakeup problem)

Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run
- Applications fetch new messages directly from the vDTU
- If there is none *and* other apps are ready, TileMux is used to block
- Race condition: checking for new msgs and blocking (like lost-wakeup problem)
    - The vDTU tracks the number of new messages of the current app

Blocking Applications

- If the current app waits for new messages, other apps should get the chance to run
- Applications fetch new messages directly from the vDTU
- If there is none *and* other apps are ready, TileMux is used to block
- Race condition: checking for new msgs and blocking (like lost-wakeup problem)
    - The vDTU tracks the number of new messages of the current app
    - The priv. IF offers a command to atomically switch to a new app

Unblocking Applications

M^{3}x

- Incoming messages cannot be stored in memory, if the receiver is blocked

Unblocking Applications

M^3x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available

Unblocking Applications

M^{3}x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available
- M^{3}x resorts to a “slow-path” by forwarding messages over the OS tile

Unblocking Applications

M^{3}x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available
- M^{3}x resorts to a “slow-path” by forwarding messages over the OS tile

M^{3}v

- The vDTU knows all EPs and can always store the message

Unblocking Applications

M^3x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available
- M^3x resorts to a “slow-path” by forwarding messages over the OS tile

M^3v

- The vDTU knows all EPs and can always store the message
- If the owner of the receive EP is blocked, the vDTU injects an interrupt

Unblocking Applications

M^3x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available
- M^3x resorts to a “slow-path” by forwarding messages over the OS tile

M^3v

- The vDTU knows all EPs and can always store the message
- If the owner of the receive EP is blocked, the vDTU injects an interrupt
- TileMux marks the receiver as ready

Unblocking Applications

M^{3}x

- Incoming messages cannot be stored in memory, if the receiver is blocked
- The receive EP is not available
- M^{3}x resorts to a “slow-path” by forwarding messages over the OS tile

M^{3}v

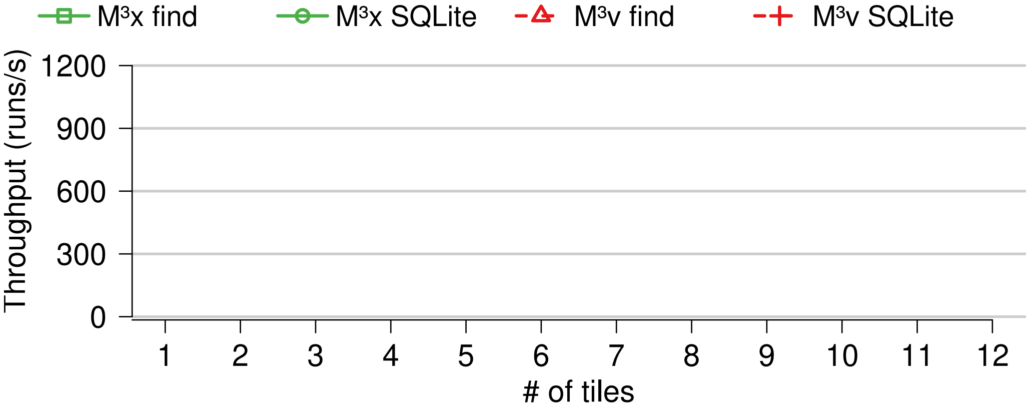
- The vDTU knows all EPs and can always store the message
- If the owner of the receive EP is blocked, the vDTU injects an interrupt
- TileMux marks the receiver as ready
- Best case: neither the OS tile nor TileMux is involved in the communication

Performance/Scalability Comparison with M^3x

- Setup: gem5 simulator, 3 GHz out-of-order x86-64 cores
- Every tile runs: SQLite/find benchmark and in-memory filesystem

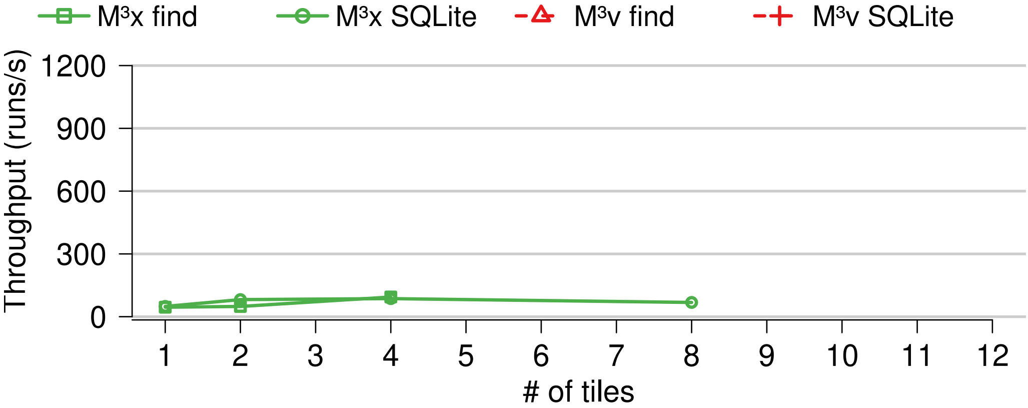
Performance/Scalability Comparison with M³x

- Setup: gem5 simulator, 3 GHz out-of-order x86-64 cores
- Every tile runs: SQLite/find benchmark and in-memory filesystem



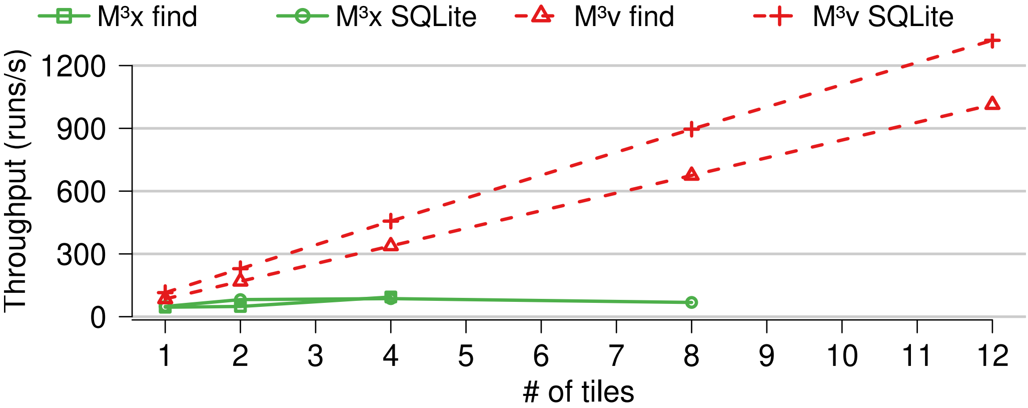
Performance/Scalability Comparison with M³x

- Setup: gem5 simulator, 3 GHz out-of-order x86-64 cores
- Every tile runs: SQLite/find benchmark and in-memory filesystem



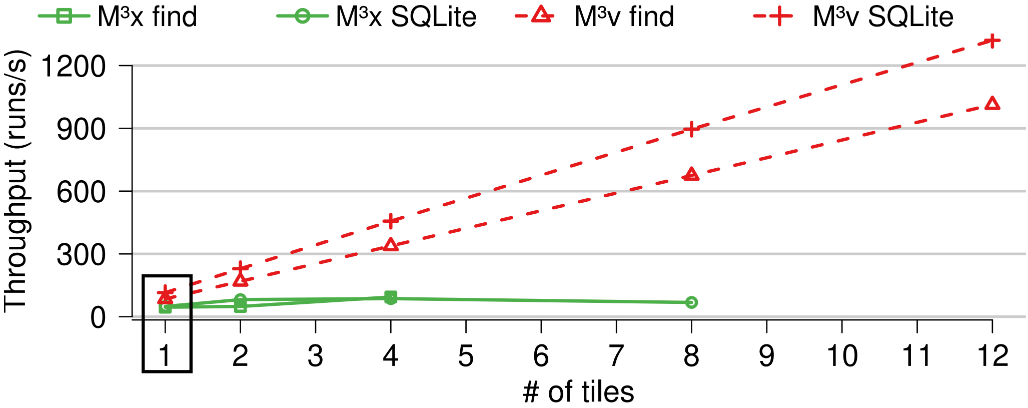
Performance/Scalability Comparison with M³x

- Setup: gem5 simulator, 3 GHz out-of-order x86-64 cores
- Every tile runs: SQLite/find benchmark and in-memory filesystem



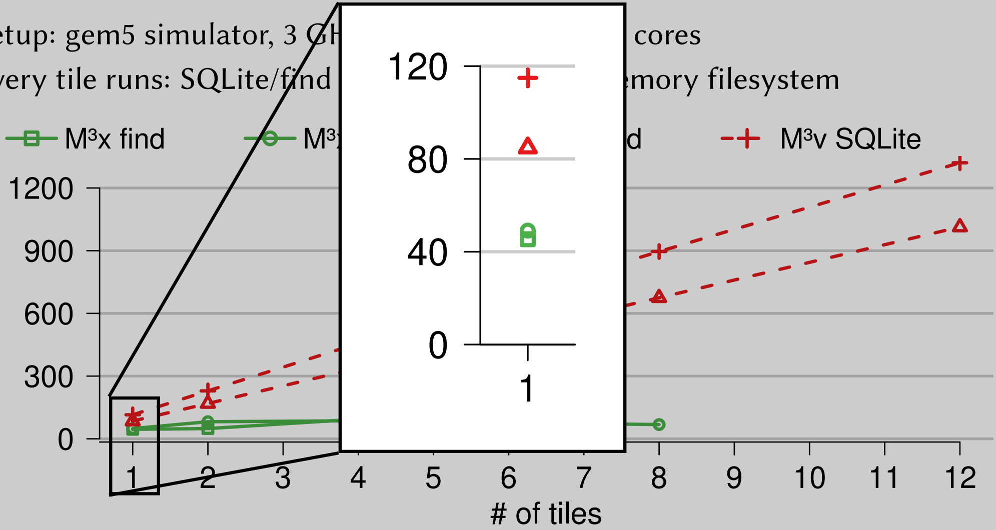
Performance/Scalability Comparison with M³x

- Setup: gem5 simulator, 3 GHz out-of-order x86-64 cores
- Every tile runs: SQLite/find benchmark and in-memory filesystem

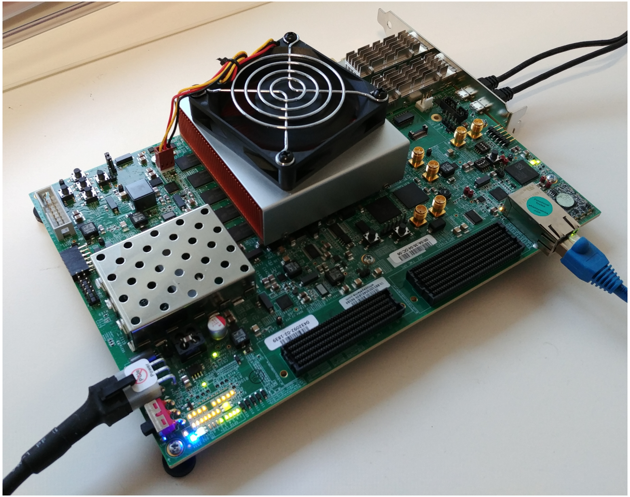


Performance/Scalability Comparison with M³x

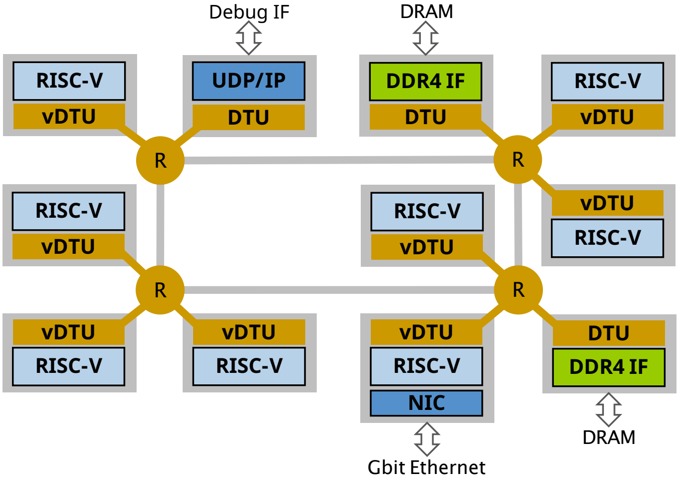
- Setup: gem5 simulator, 3 GH cores
- Every tile runs: SQLite/find emory filesystem



Hardware Implementation

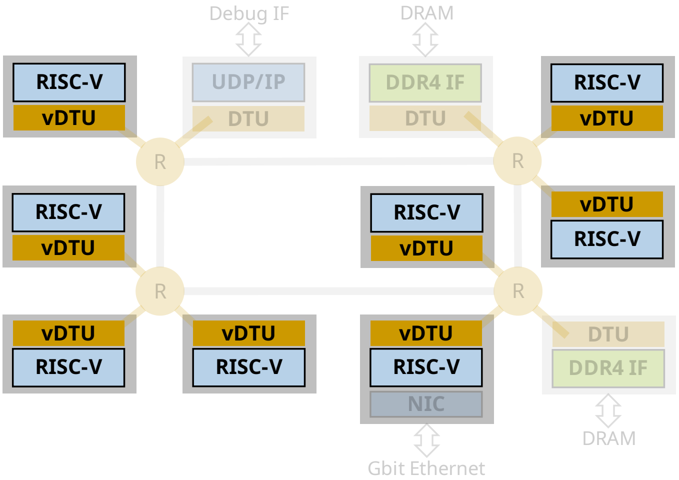


Hardware Implementation



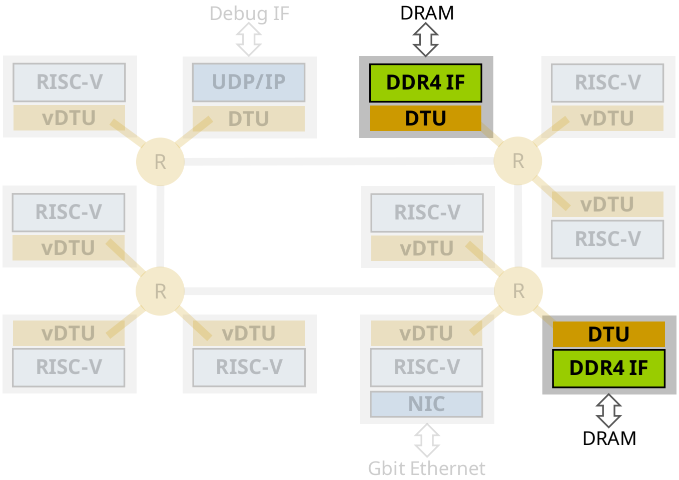
- Xilinx VCU118 FPGA
- RISC-V: in-order Rocket or out-of-order BOOM
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- vDTU contains 128 EPs

Hardware Implementation



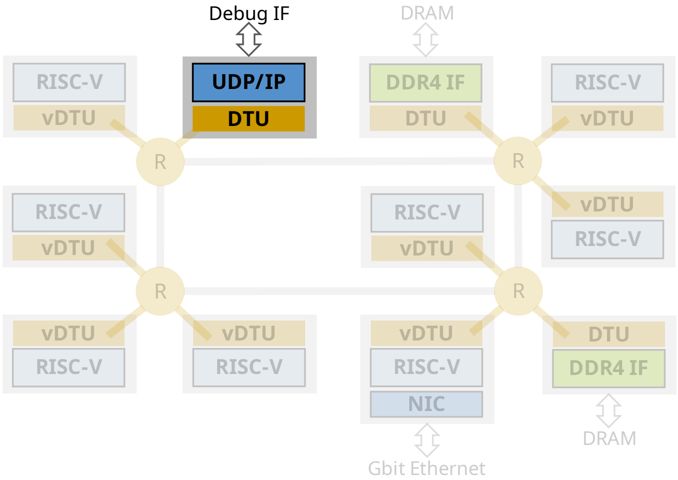
- Xilinx VCU118 FPGA
- RISC-V: in-order Rocket or out-of-order BOOM
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- vDTU contains 128 EPs

Hardware Implementation



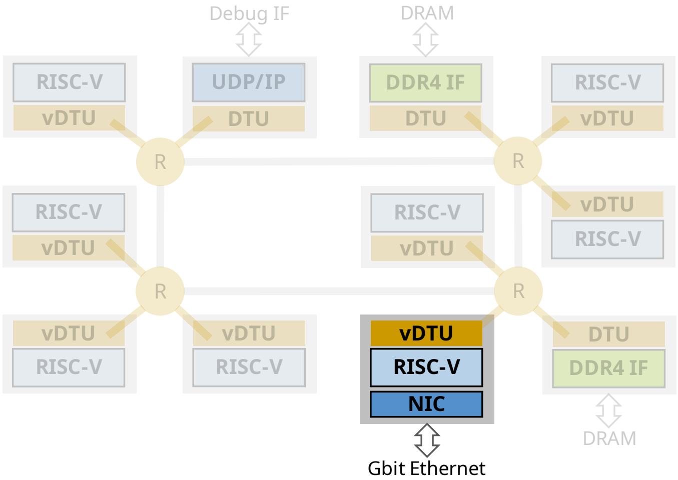
- Xilinx VCU118 FPGA
- RISC-V: in-order Rocket or out-of-order BOOM
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- vDTU contains 128 EPs

Hardware Implementation



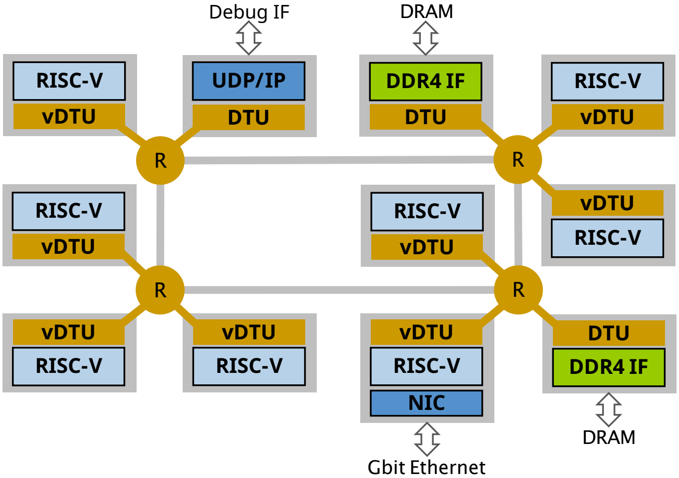
- Xilinx VCU118 FPGA
- RISC-V: in-order Rocket or out-of-order BOOM
- Rocket at 100 MHz, BOOM at 80 MHz
- 2x16 kB L1, 512 kB L2
- vDTU contains 128 EPs

Hardware Implementation



- 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
- vDTU contains 128 EPs

Hardware Implementation



- 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
- vDTU contains 128 EPs

vDTU 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
vDTU	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

vDTU 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
vDTU	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

vDTU 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
vDTU	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

vDTU Size and Complexity

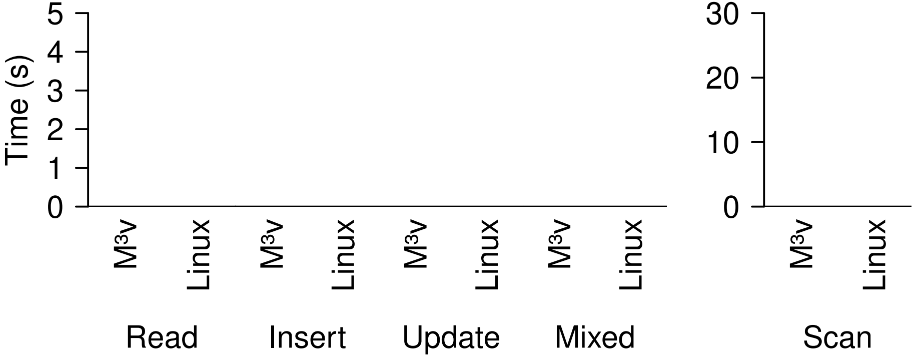
	LUTs [k]	**FFs [k]**	**BRAMs**
BOOM	143.8	71.8	159
Rocket	46.6	22.0	152
NoC router	3.4	2.2	0
vDTU	15.2	5.8	0.5
Control Unit	10.3	3.3	0.5
NoC CTRL	3.2	1.5	0
CMD CTRL	7.1	2.8	0.5
Unpriv. IF	6.2	2.5	0.5
Priv. IF	0.9	0.3	0
Register file	2.0	1.0	0
Memory mapper + PMP	0.6	0.2	0
I/O FIFOs	2.3	0.3	0

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

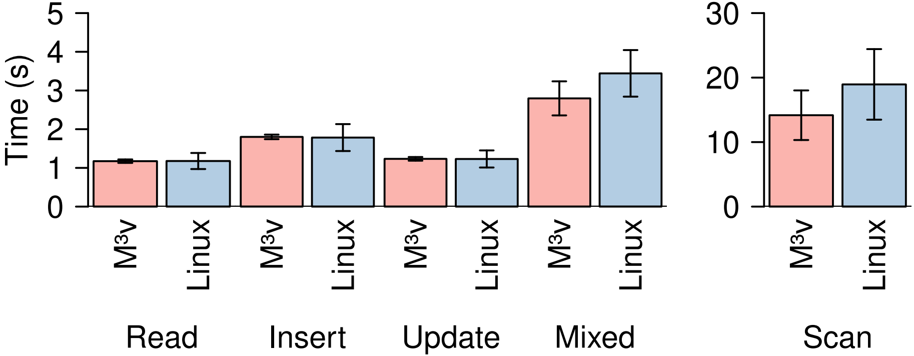
Performance Comparison with Linux

- 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

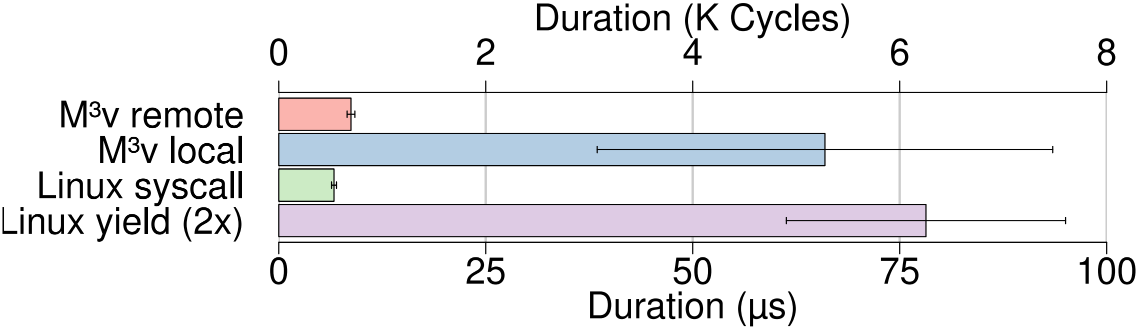


Conclusion

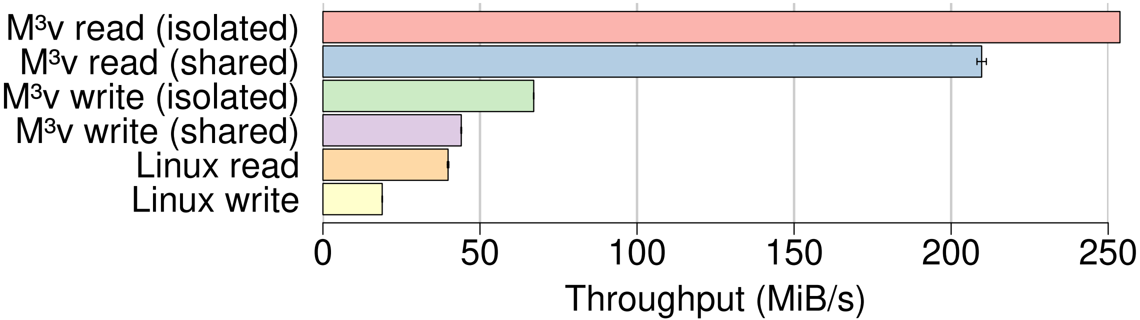
- M^3 explores a new system architecture with a new per-tile hardware component
- M^{3}v shows how general-purpose cores can be multiplexed efficiently
- Hardware implementation demonstrates modest additional hardware costs
- Competitive performance to Linux with context-switch-heavy workloads
- The complete hardware/software stack is available as open source: `https://github.com/Barkhausen-Institut/M3`

Backup Slides

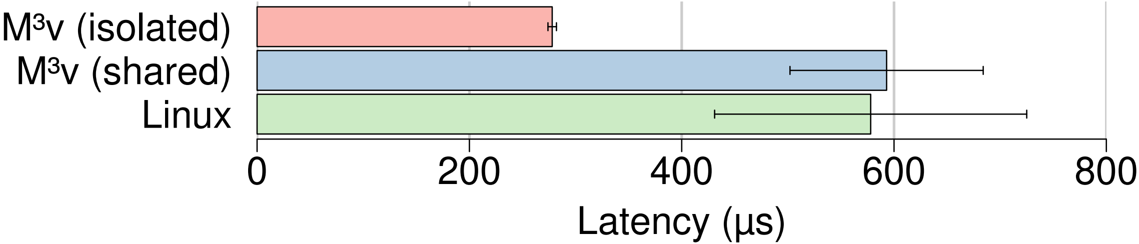
Microbenchmarks: IPC and Context Switches



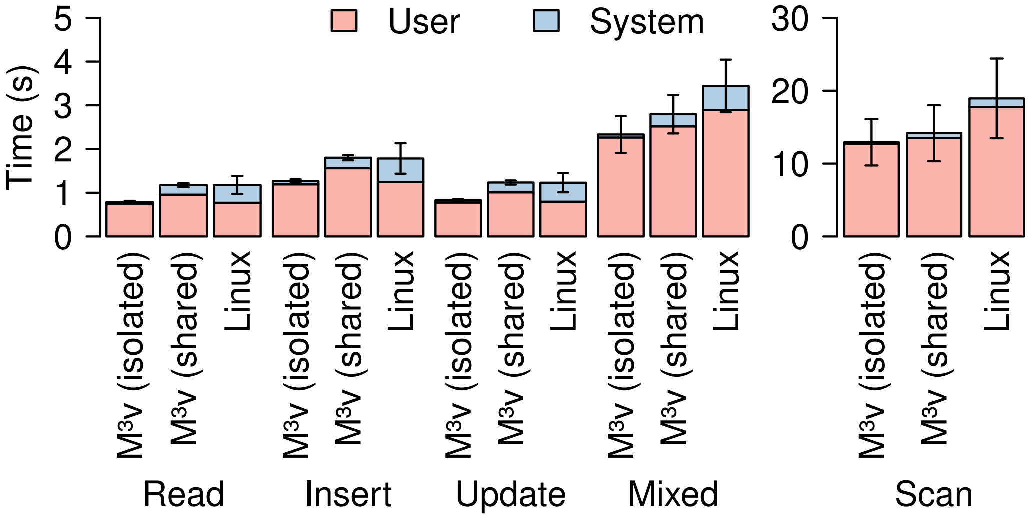
Microbenchmarks: File System



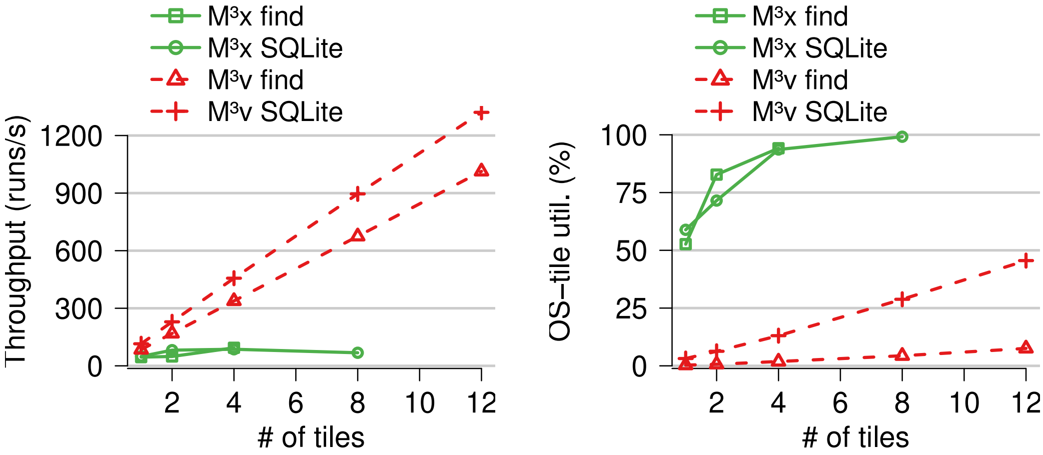
Microbenchmarks: Networking



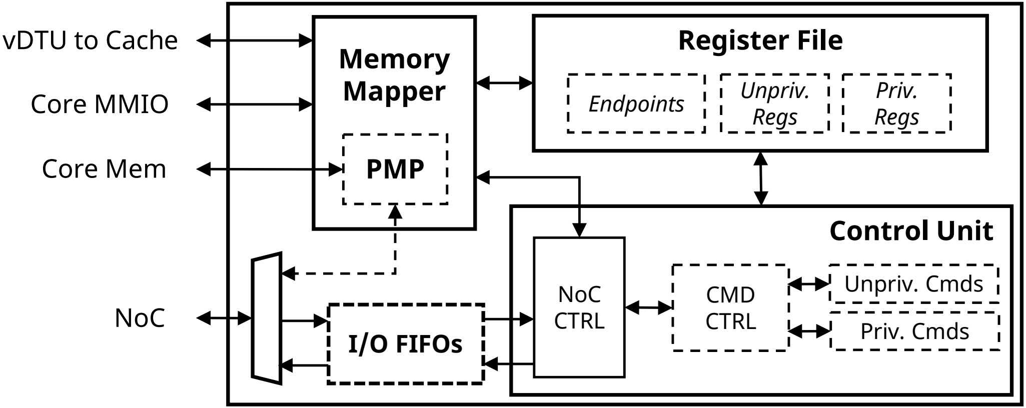
Macrobenchmarks: YCSB



Comparison with M³x: OS-tile utilization



Hardware Implementation


vDTU to Cache
Core MMIO
Core Mem
NoC
Memory Mapper
PMP
Register File
Endpoints
Unpriv. Regs
Priv. Regs
Control Unit
NoC CTRL
CMD CTRL
Unpriv. Cmds
Priv. Cmds
I/O FIFOs