GPU native I/0 2.0

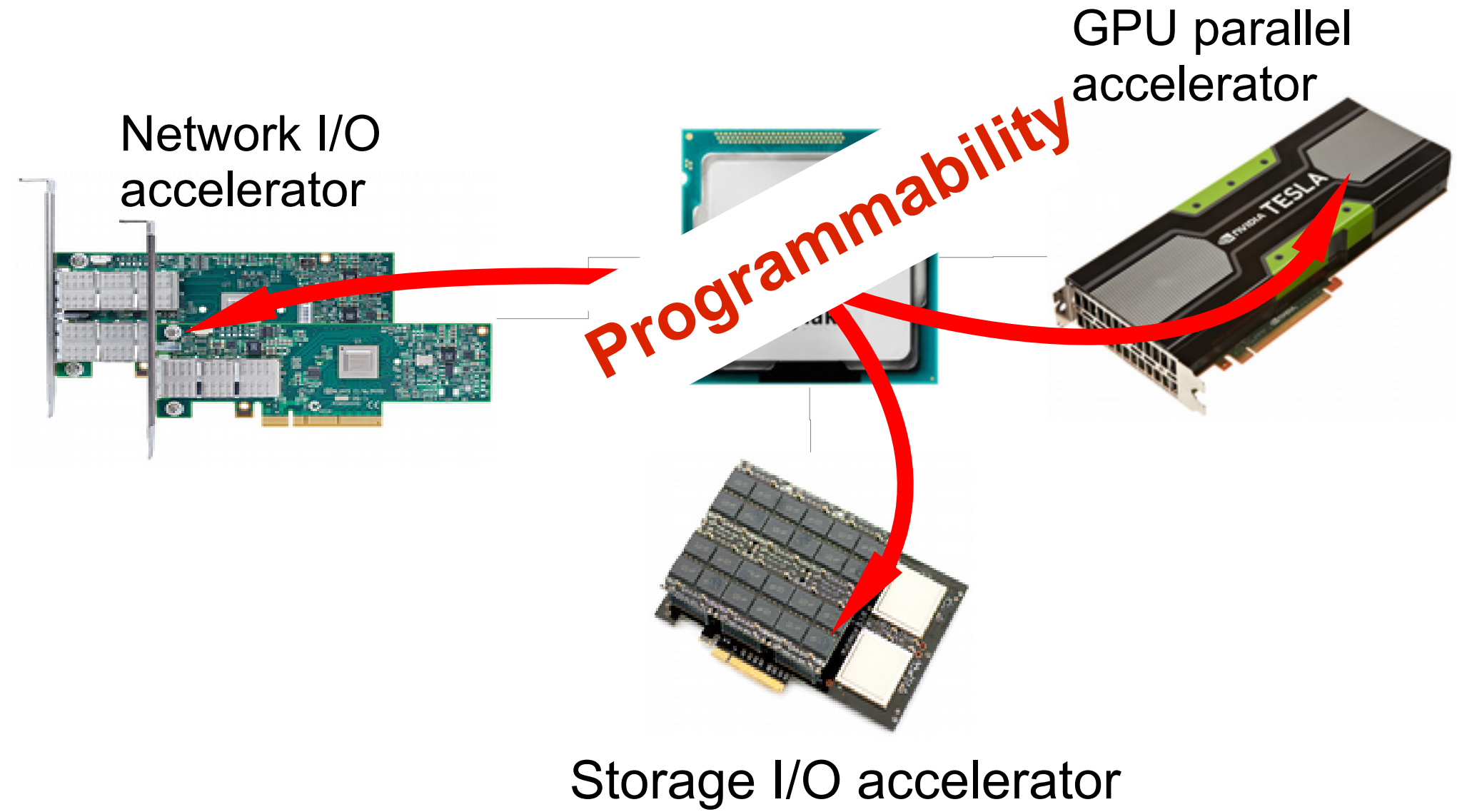
Leveraging new hardware for efficient GPU I/O abstractions

Mark Silberstein

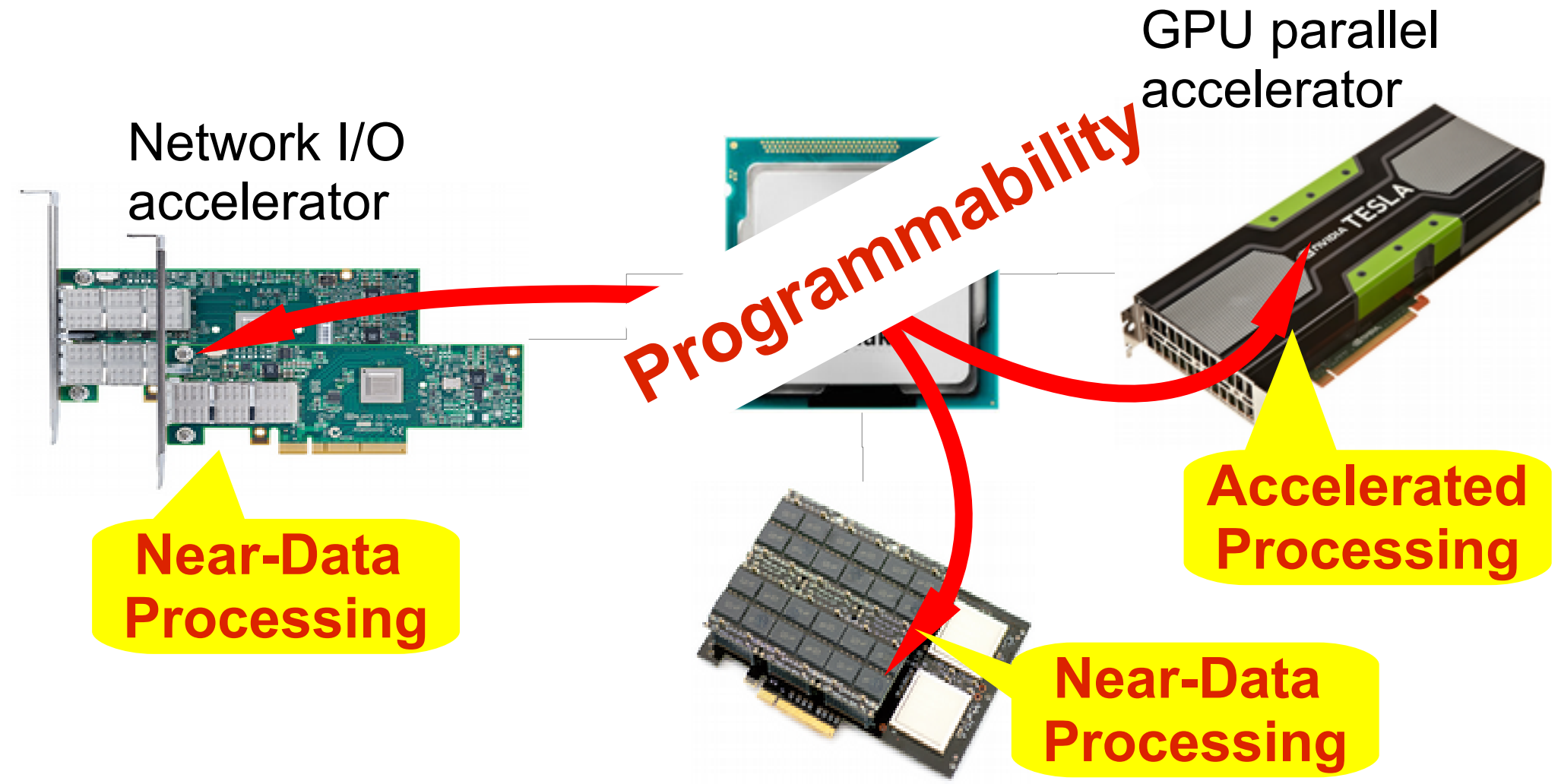
Technion

November 2019
NVIDIA

Central Processing Units (CPUs) are no longer **Central**

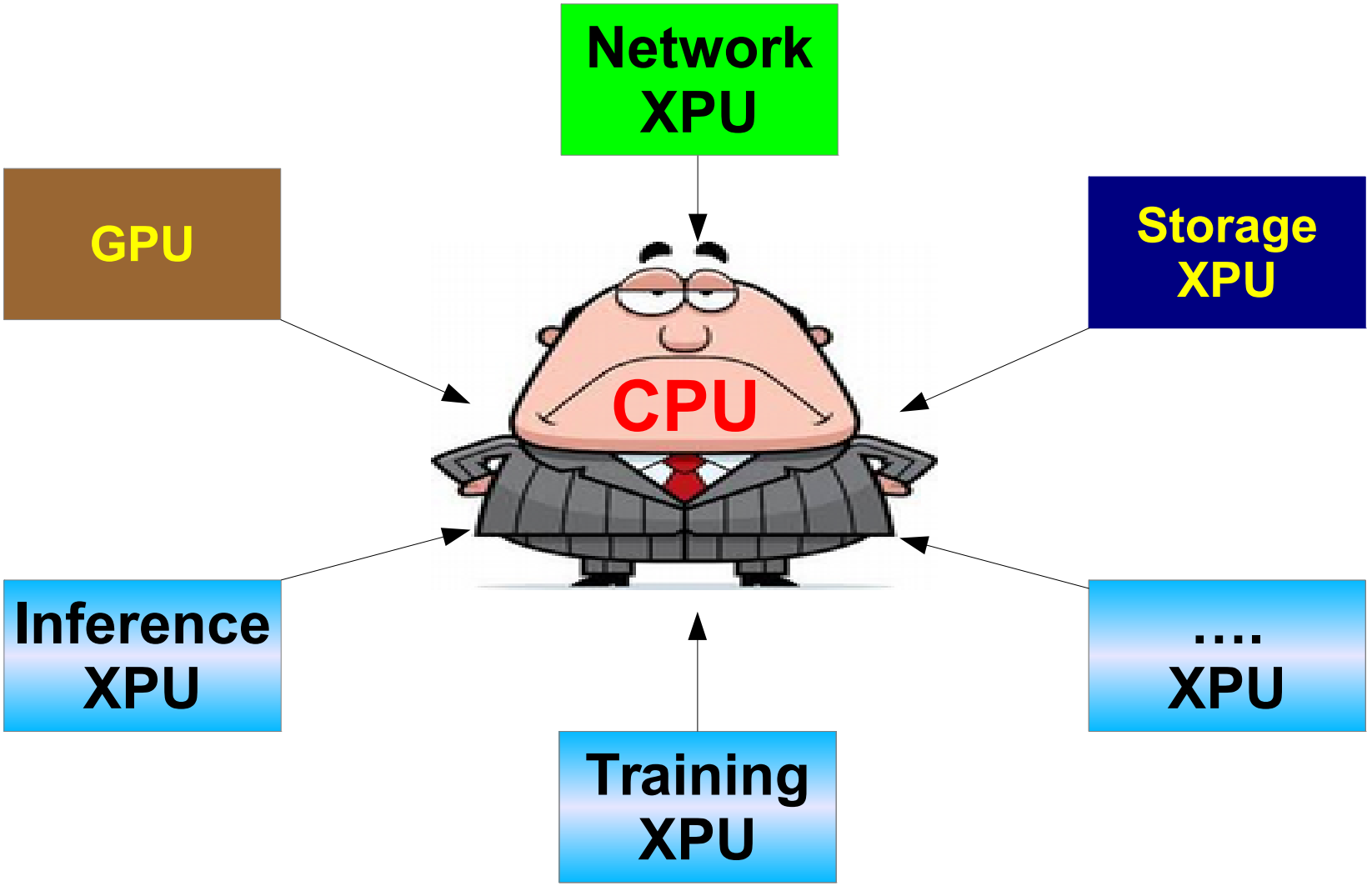


Omni-programmable system
X- Processing Units: XPUs



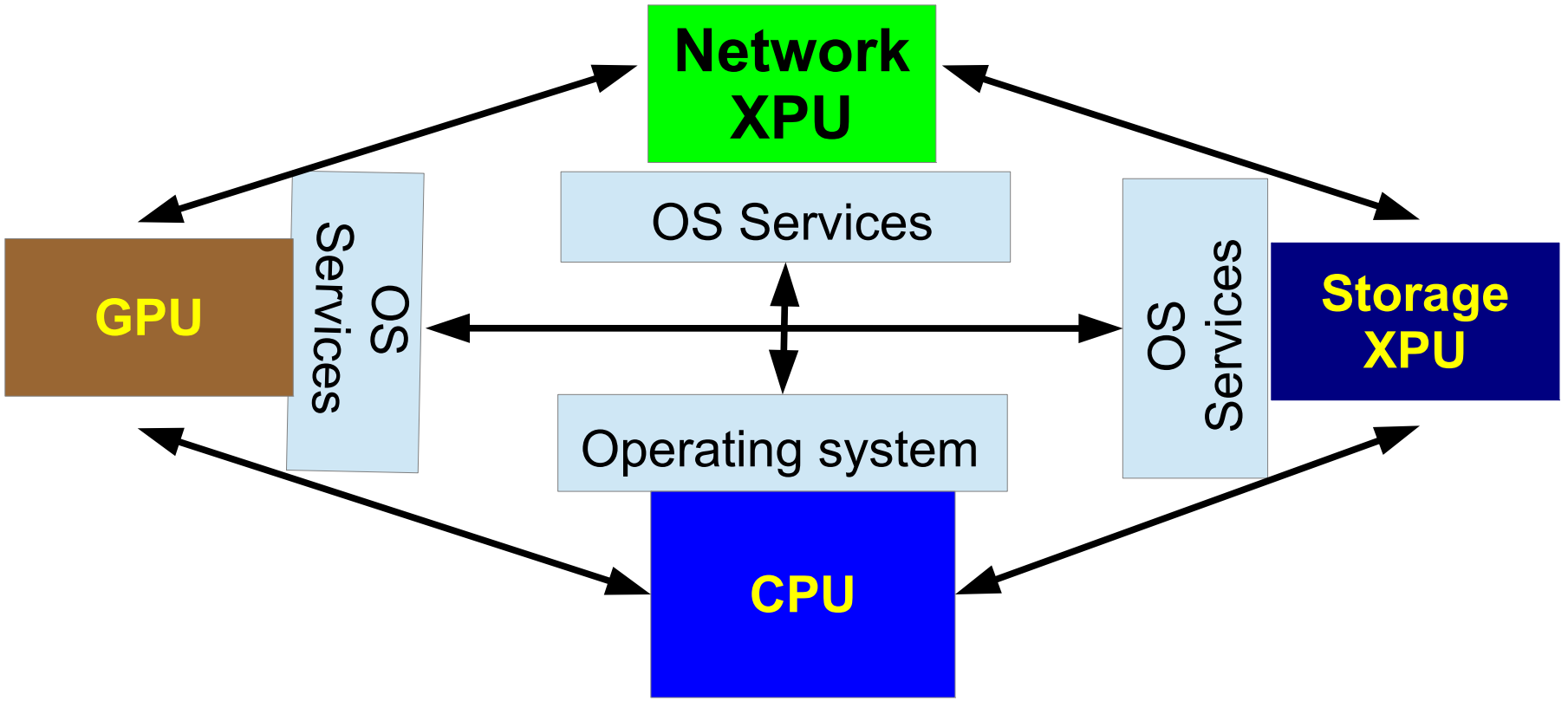
But XPUs also create new walls!

THE problem ***is general***:
OS architecture is CPU - centric



OmniX: accelerator-centric OS architecture

HotOS17



Accelerator-centric OS architecture

Types of OS abstractions for accelerators

Accelerator-centric: no CPU in data/control path

Accelerator-friendly: accelerator-aware host OS

Data-centric: inline near-data processing

Types of OS abstractions for accelerators

ASPLOS13,TOCS14,OSDI14,TOCS15,ISCA16,SYSTOR16,ROSS16,ATC17,HotOS17,ATC19,PACT19

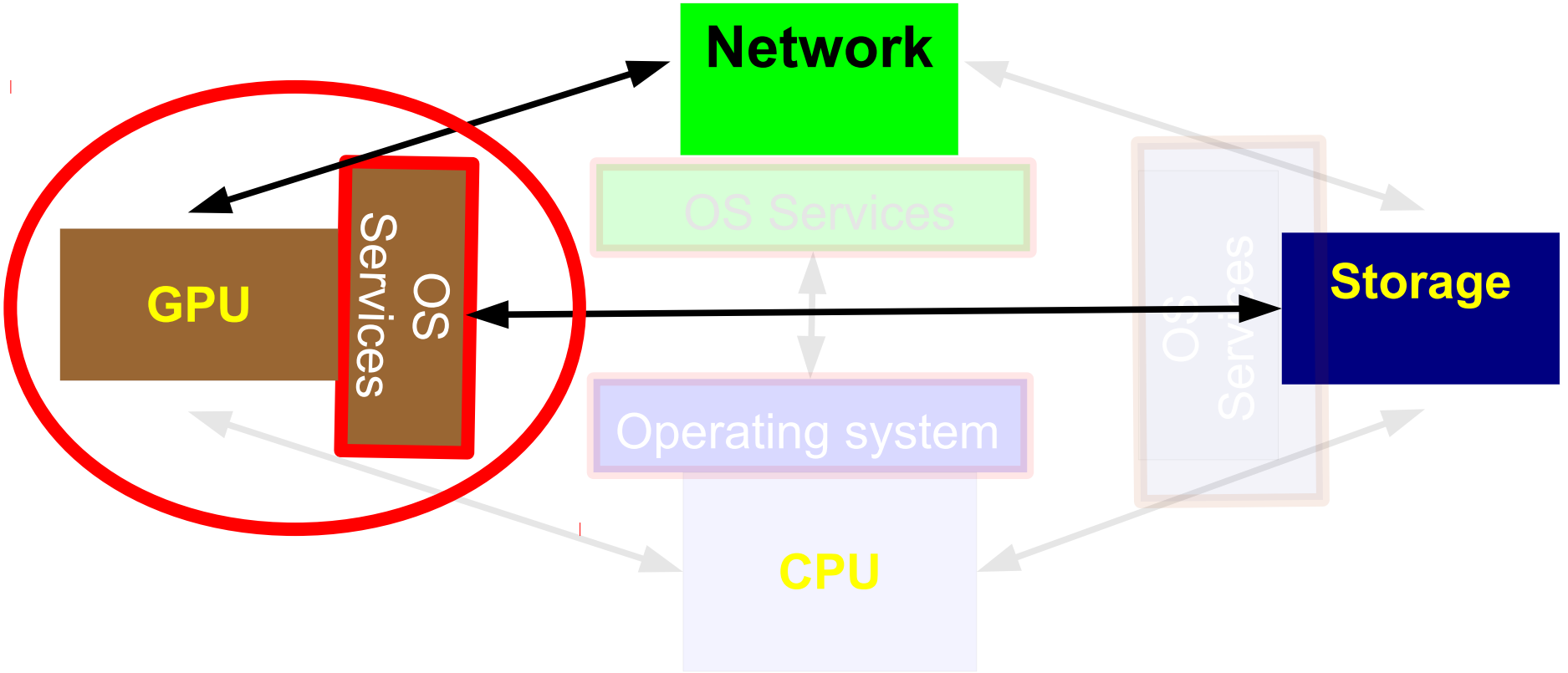
Accelerator-centric: no CPU in data/control path

Accelerator-friendly: accelerator-aware host OS

SFMA17, SFMA18, FCCM19, USENIX ATC19

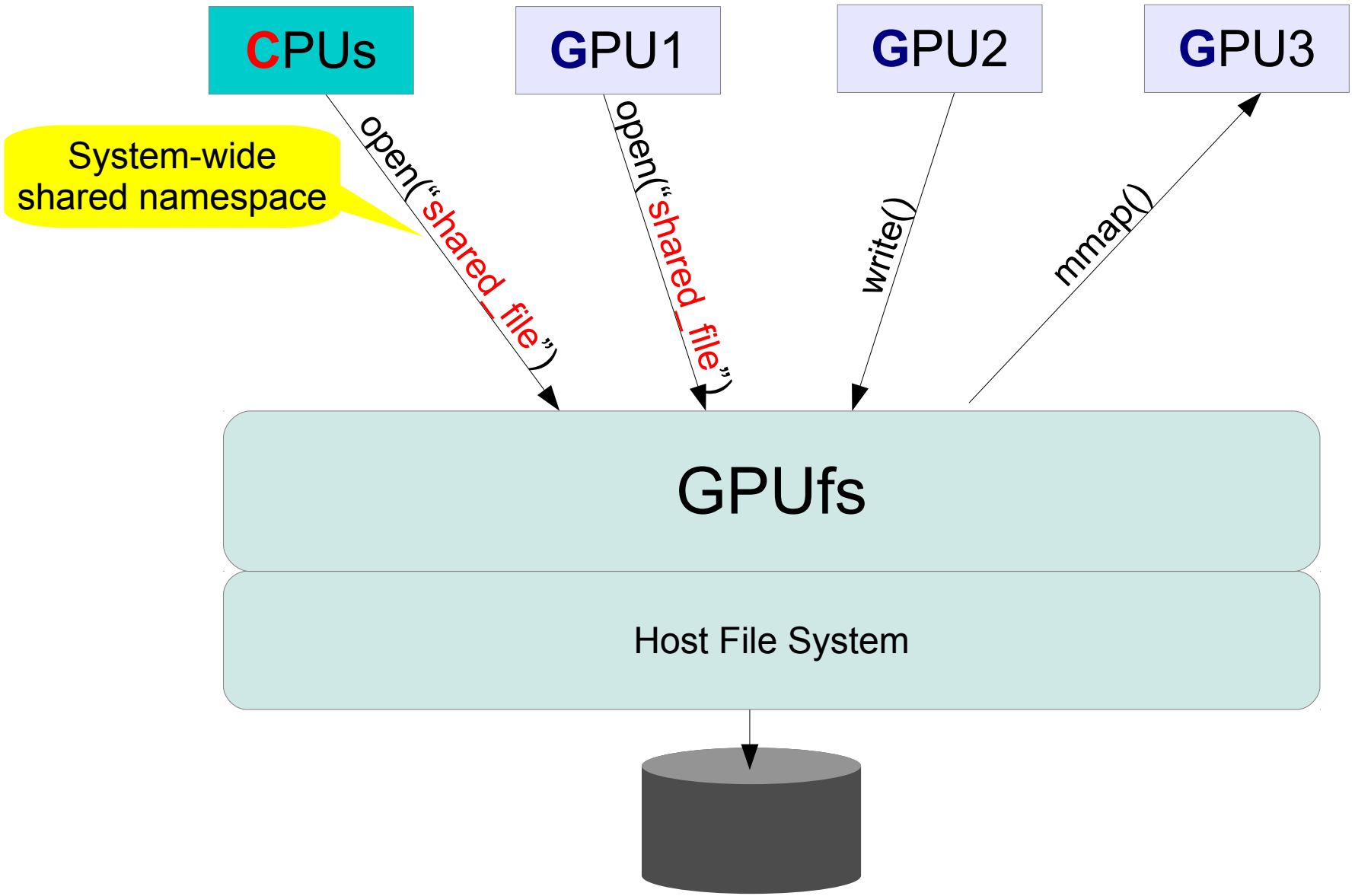
Data-centric: inline near-data processing

GPU I/O services V1.0



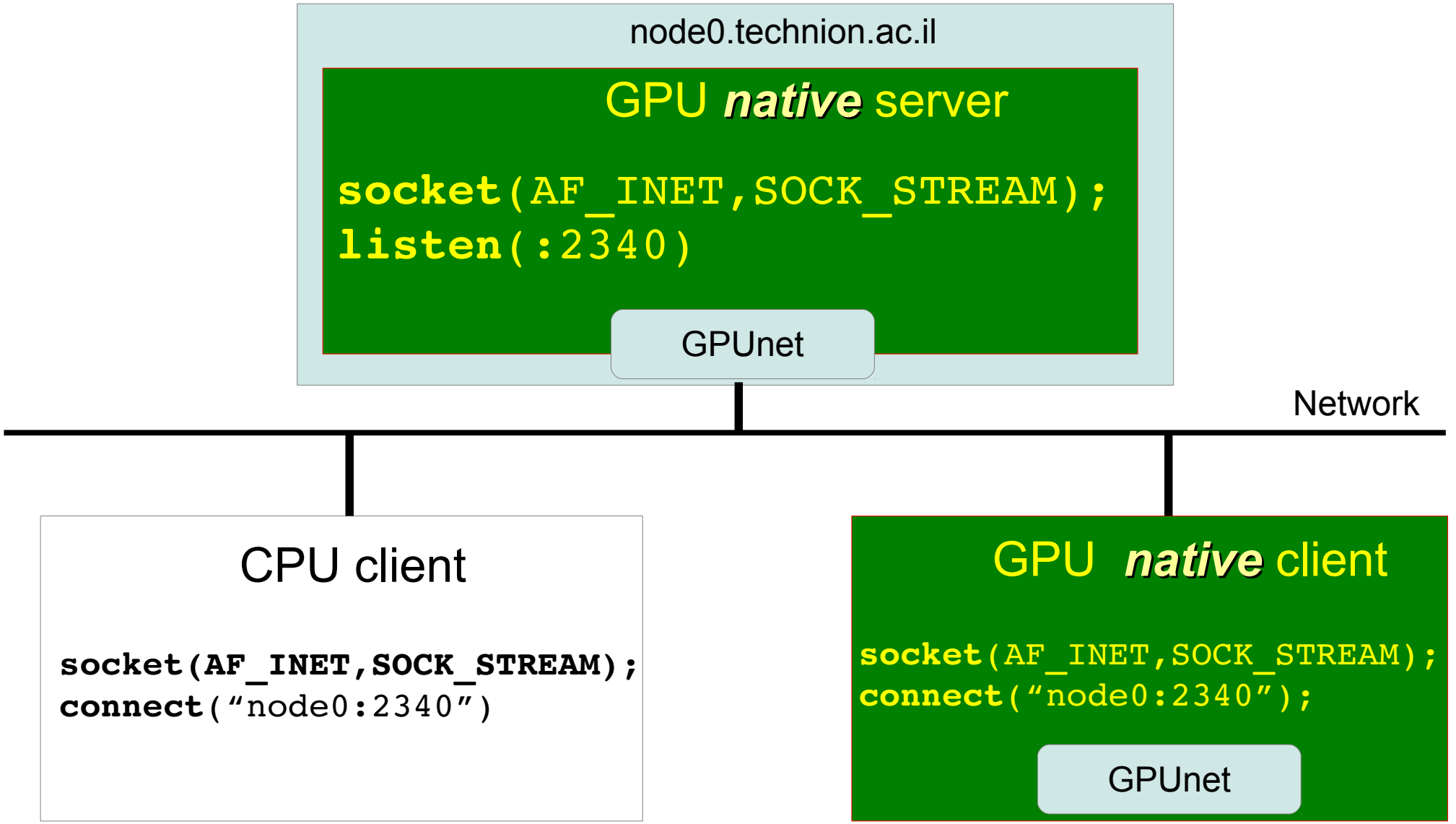
GPUfs: File system library for GPUs

ASPLOS13: S., Keidar, Ford, Witchel



GPUnet: Network library for GPUs

OSDI14



Accelerator in full control over its I/O

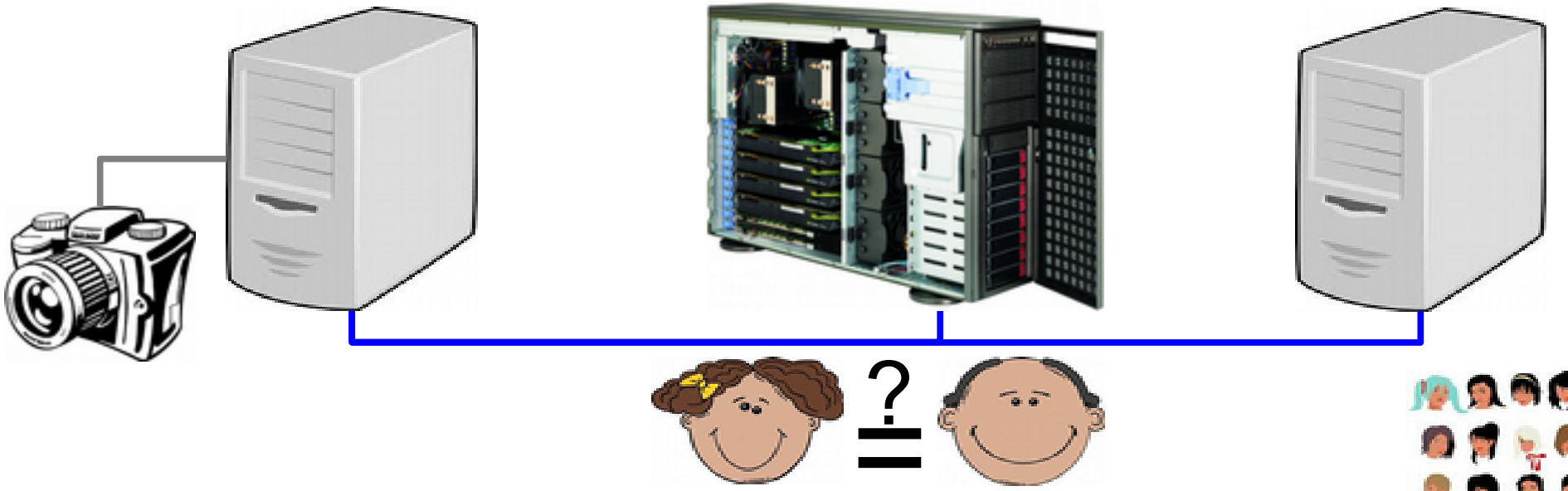
- I/O without «leaving» the GPU kernel
  - Data-driven access to huge DBs
  - Full-blown multi-tier GPU network servers
  - Multi-GPU Map/Reduce (no user CPU code)
- POSIX-like APIs with slightly modified semantics
- Transparency for the rest of the system
- Reduced code complexity
- Unleashed GPU performance potential

Example: face verification server

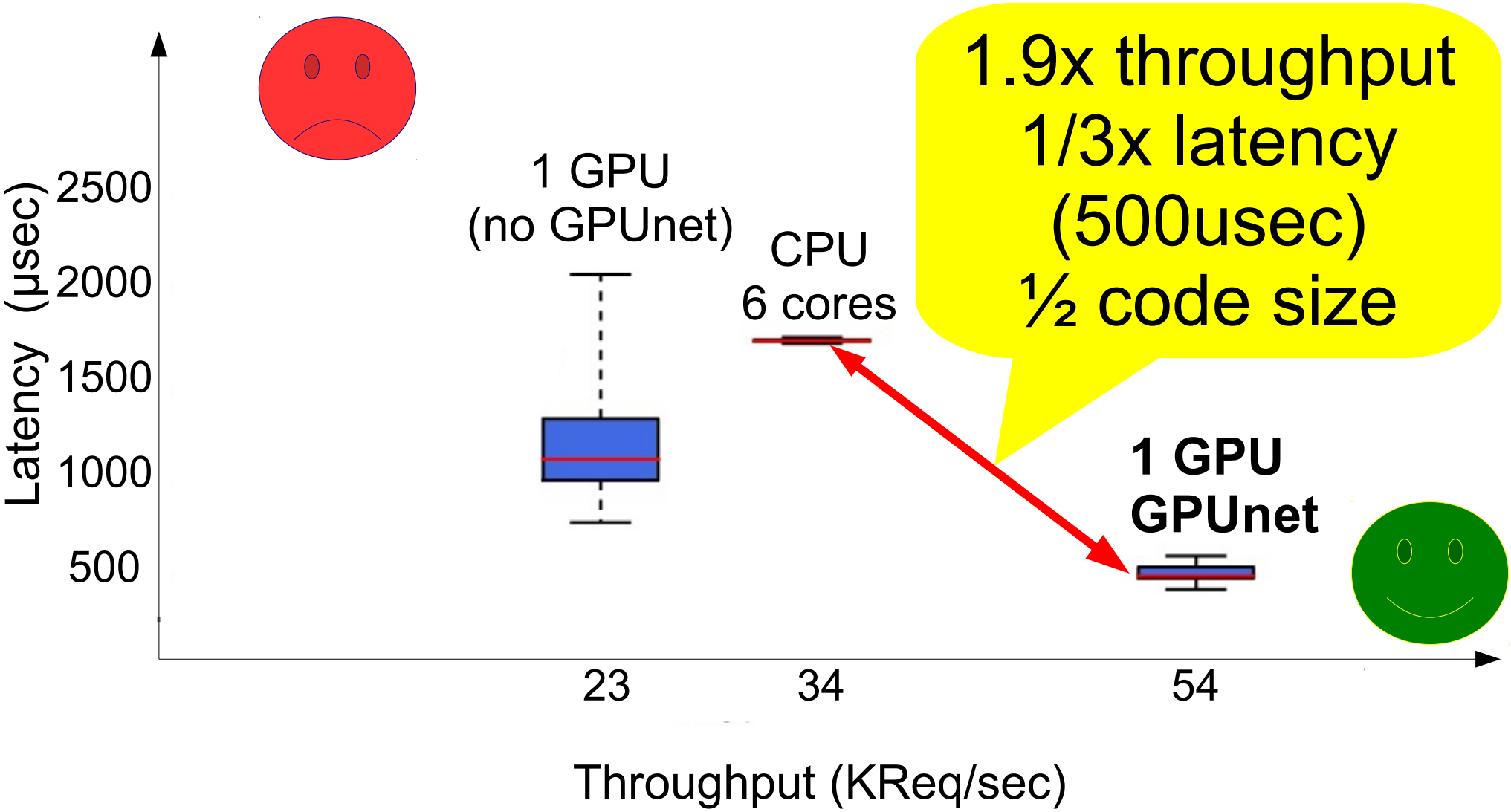
CPU client (unmodified)

GPU server (GPUnet)

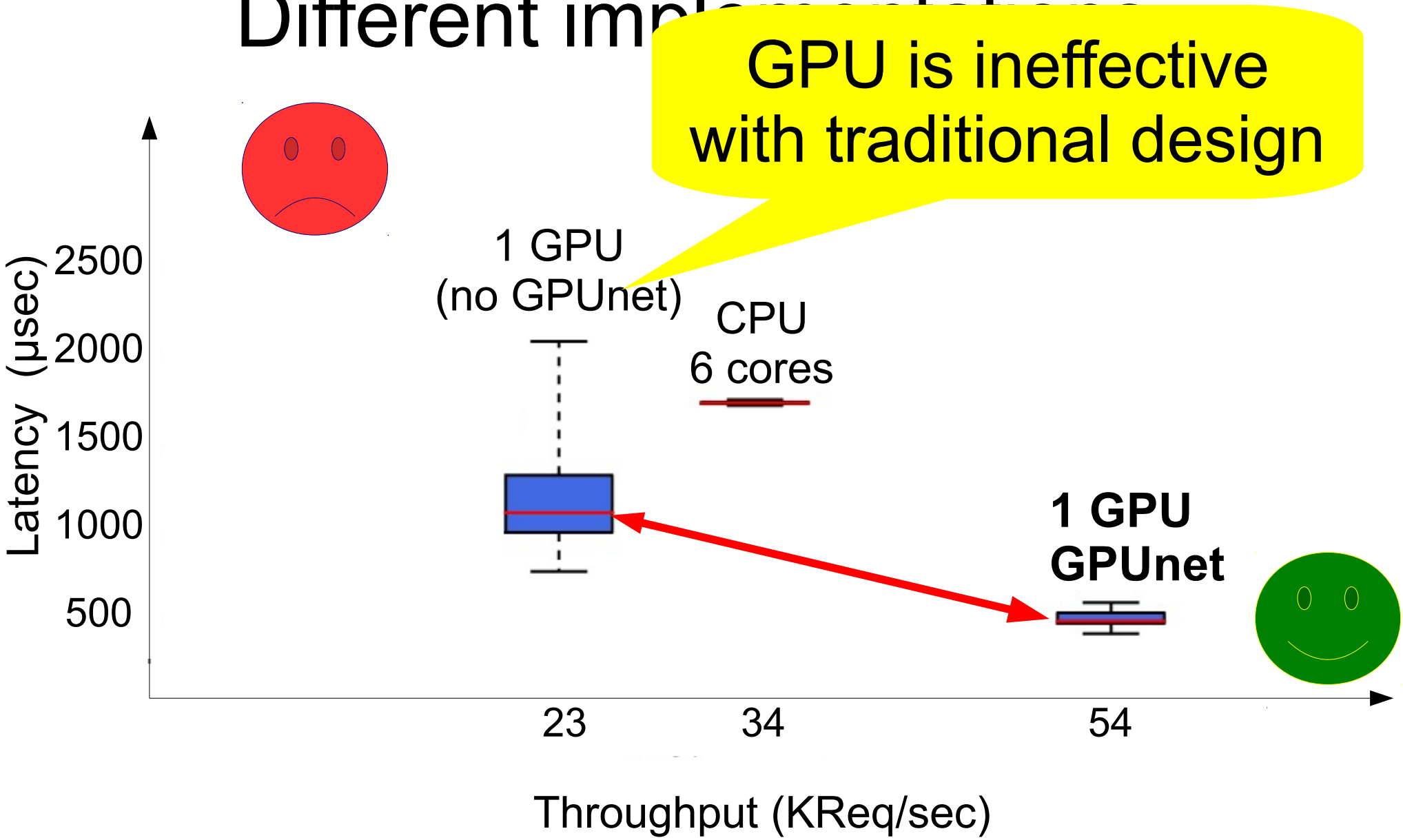
memcached (unmodified)



Face verification: Different implementations



Face verification: Different imp



Main design principles

- Micro-kernel design
  - RPC to File/Network services on the ***CPU***
  - User-land abstraction implementation (libOSes)

Main design principles

- Micro-kernel design
  - RPC to File/Network services on the CPU
  - User-land abstraction implementation (libOSes)
- Single name space with the CPU OS
  - Same socket space, same file name space

Main design principles

- Micro-kernel design
  - RPC to File/Network services on the CPU
  - User-land abstraction implementation (libOSes)
- Single name space with the CPU OS
  - Same socket space, same file name space
- Extensive SW layer on the GPU
  - Handles massive API parallelism
  - Implements consistency model (FS)
  - Implements flow control (sockets)

Main design principles

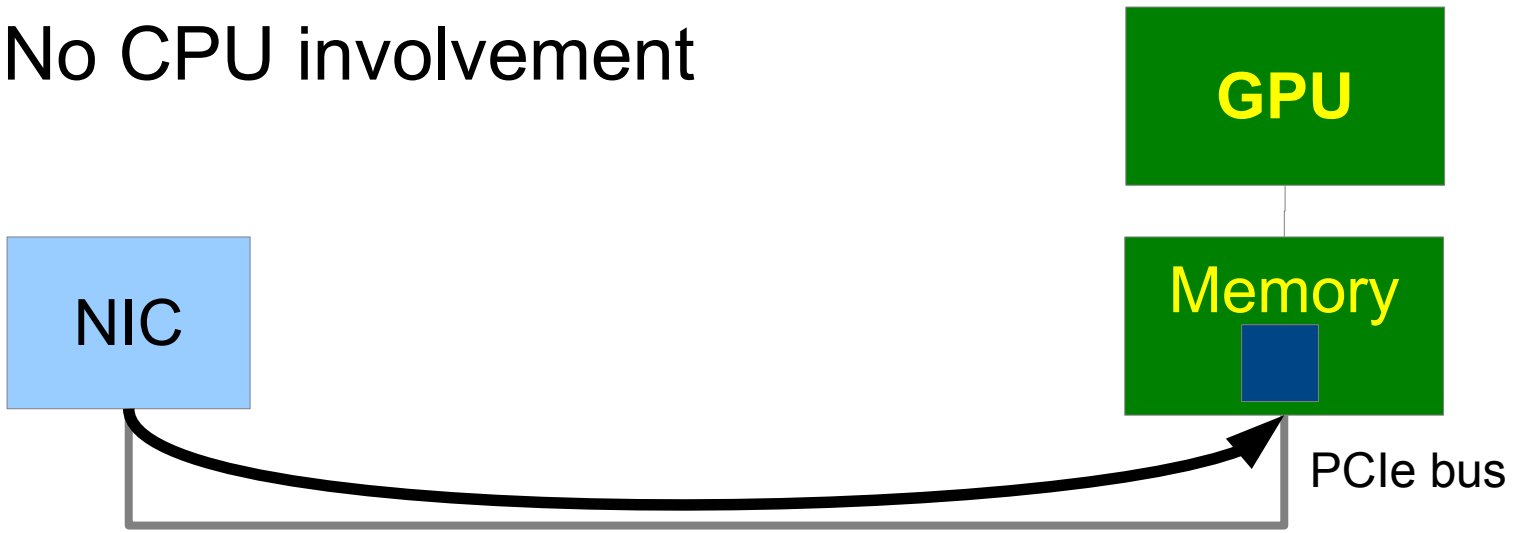
- Micro-kernel design
  - RPC to File/Network services on the CPU
  - User-land abstraction implementation (libOSes)
- Single name space with the CPU OS
  - Same socket space, same file name space
- Extensive SW layer on the GPU
  - Handles massive API parallelism
  - Implements consistency model (FS)
  - Implements flow control (sockets)
- Seamless data path optimization
  - Eliminates CPU from data path
  - Exploits data locality

Main design principles

- Micro-kernel design
  - RPC to File/Network services on the CPU
  - User-land abstraction implementation (libOSes)
- Single name space with the CPU OS
  - Same socket space, same file name space
- Extensive SW layer on the GPU
  - Handles massive API parallelism
  - Implements consistency model (FS)
  - Implements flow control (sockets)
- Seamless data path optimization
  - Eliminates CPU from data path
  - Exploits data locality

Optimized I/O: no CPU in data path

- SSD/NIC may perform DMA directly into/from GPU memory without the CPU (P2P DMA)
- Why?
  - Lower latency
  - Less buffering/complexity for thpt
  - No CPU involvement



Optimized I/O: no CPU in data path

- SSD/NIC may perform DMA directly into/from GPU memory without the CPU (P2P DMA)

Challenge: the OS is on the CPU!

I/O device sharing, multiplexing, translation, transport layer

Examples:

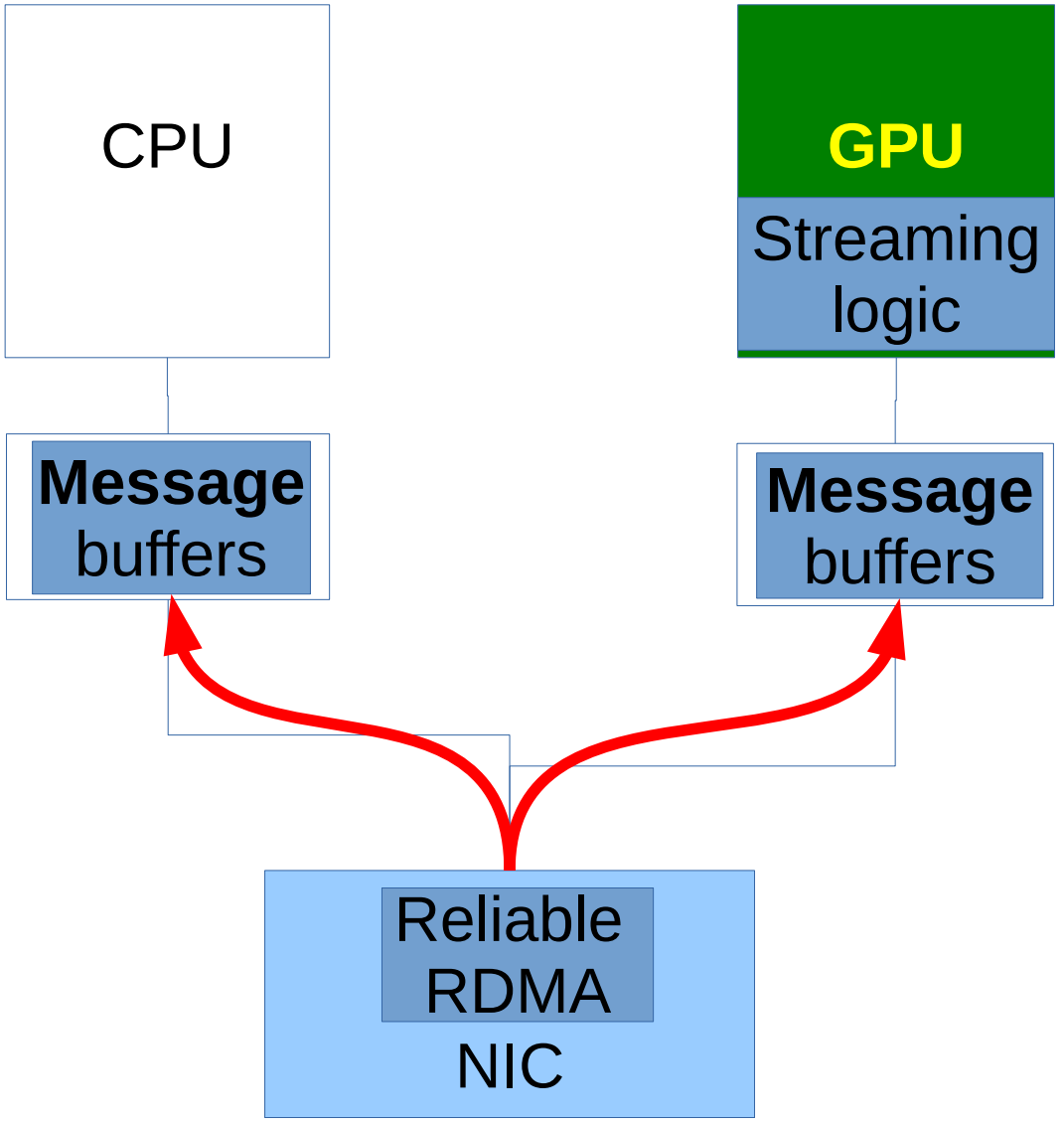
- Data from FS could be in the CPU page cache
- GPU and CPU both need to access the network
- TCP on GPU?

Storage: OS integration of P2P between SSD and GPUs

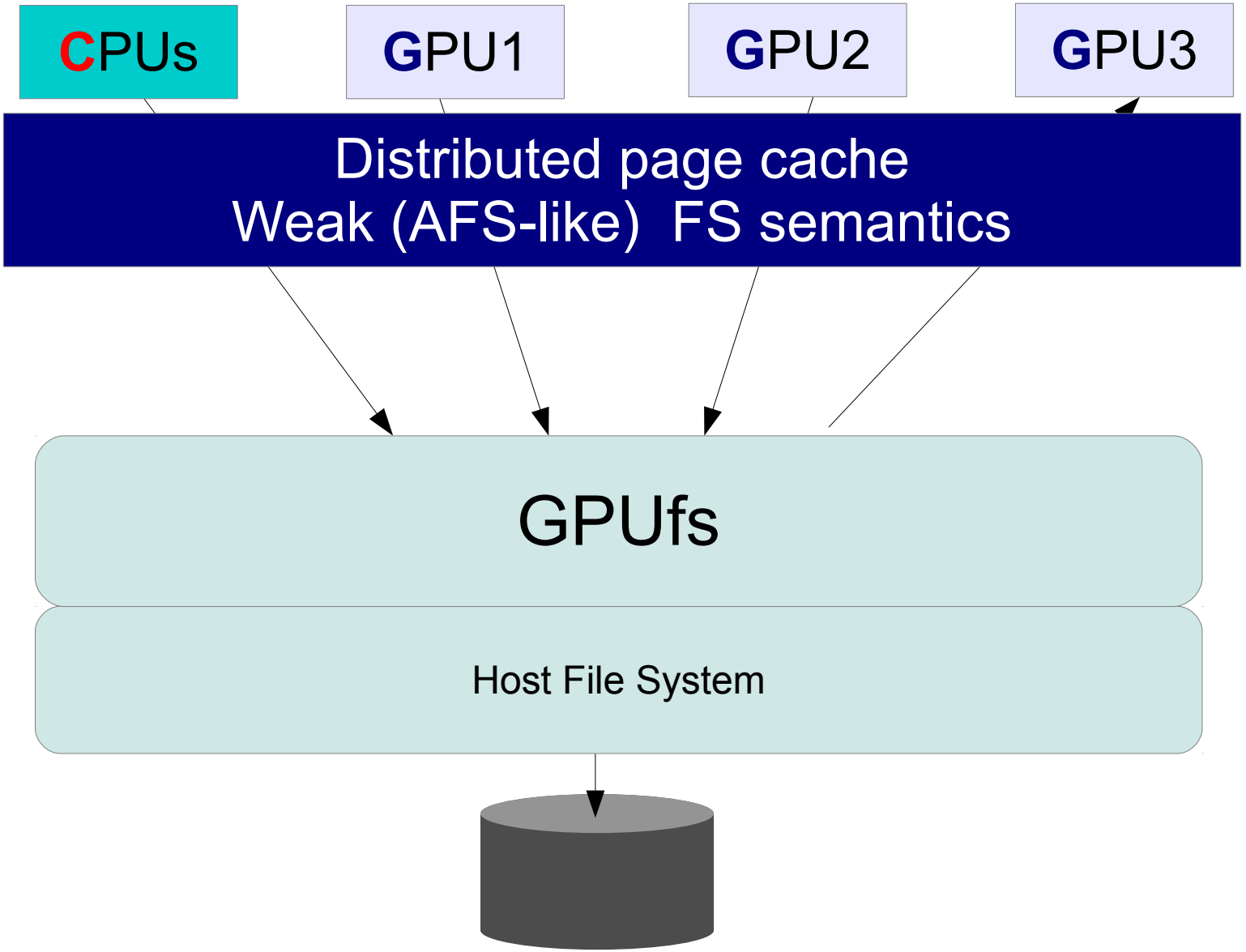
USENIX ATC17, similar NVIDIA product in 2019

- Regular OS file APIs may use **GPU** memory buffers
  - mapping GPU memory into CPU address space
- Maintains POSIX FS consistency
- **Transparently fetches the page cache or P2P DMA**
- **Integrates with OS prefetcher**
- Compatible with OS block layer (i.e., software RAID)
- Results:
  - 5.2GB/s from SSDs to GPU
  - 2-3x in applications

Networking: offloading transport layer to the NIC (via RDMA)



Transparent locality optimization



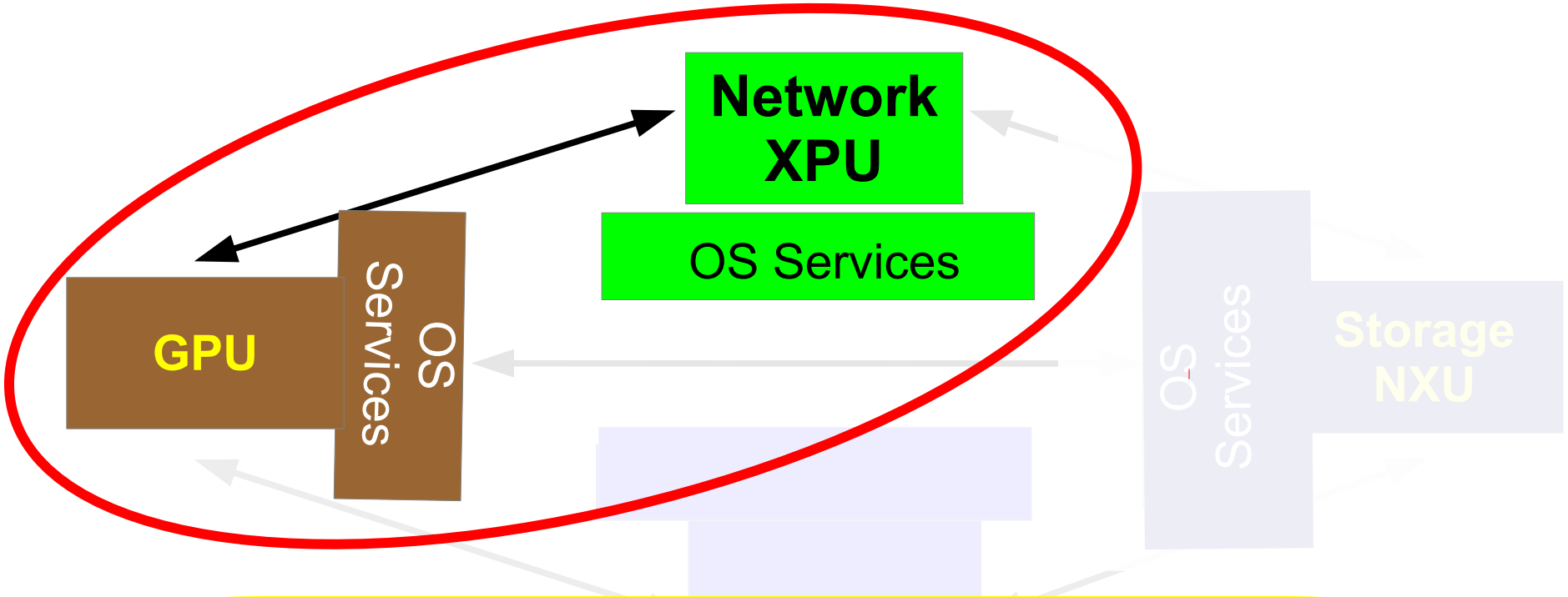
Summary so far...

- GPU-centric I/O services V1.0
  - Simplify code development
  - Enable transparent performance optimization

CPU is used as the driver for all operations

Can we do better?

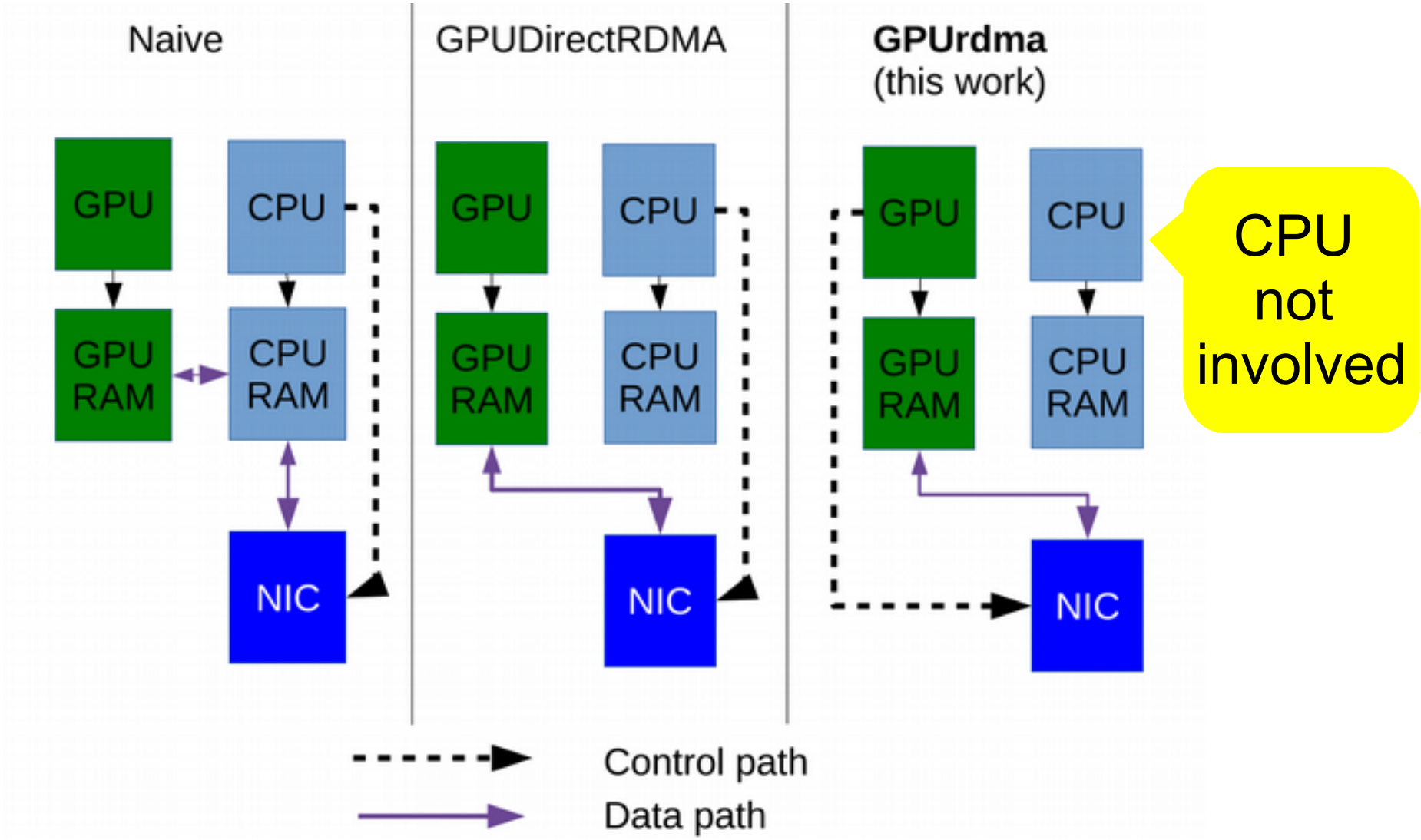
CPU-less design: (GPU I/O V1.5) no CPU in control and data path



Lower latency (no CPU roundtrip)
Better scalability (no CPU load)

GPUrdma: direct GPU access to RDMA

ROSS16



GPUrdma: direct GPU access to RDMA

GPU-to-GPU roundtrip latency via Infiniband

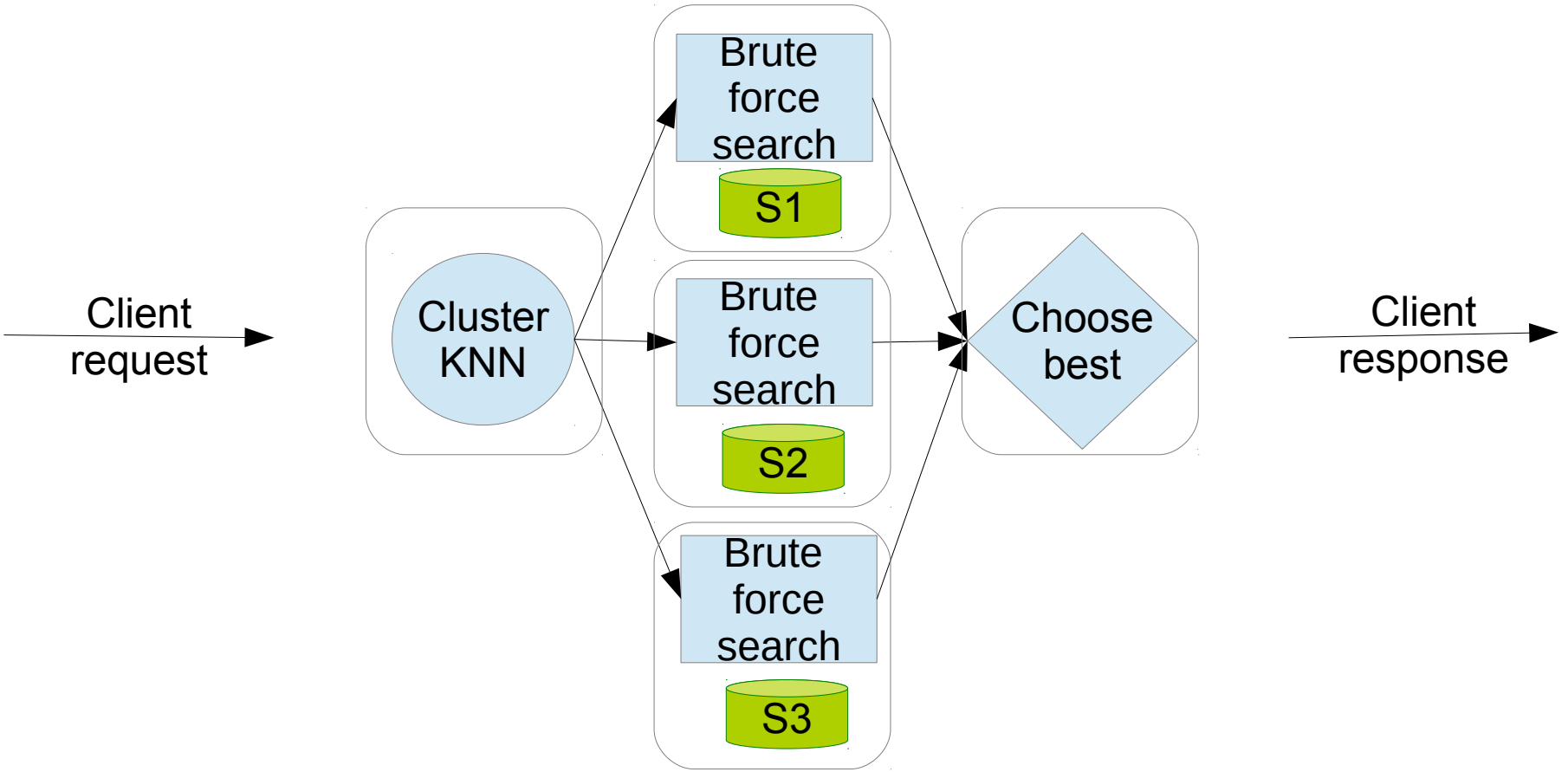
GPUnet (CPU-mediated RDMA): 50 usec

GPUrdma (NIC controlled by GPU): 5 usec

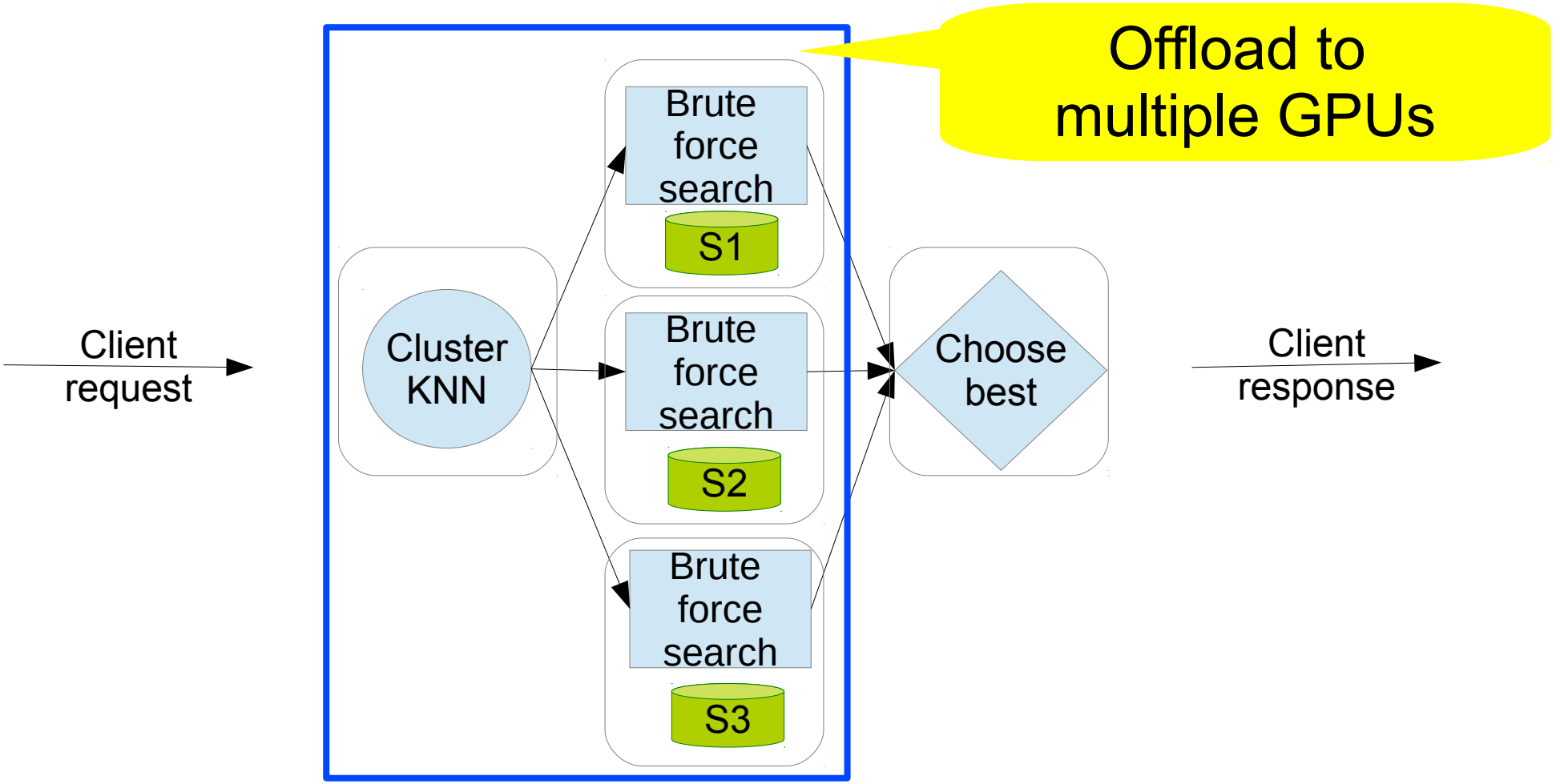
CPU-less design: lower latency

The case for CPU-less multi-GPU server design

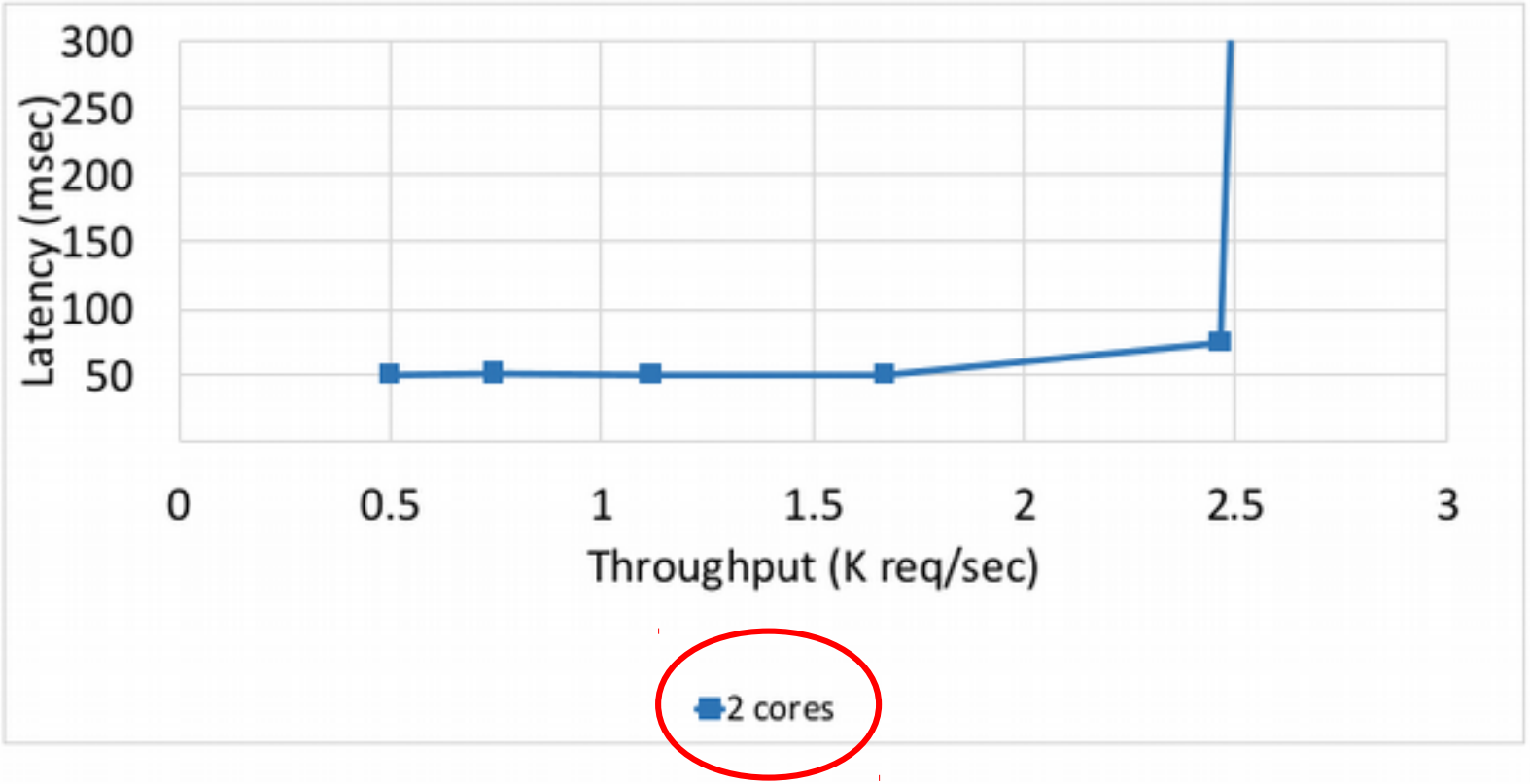
Image Similarity Search



Traditional design: CPU controls GPU invocation and data movements

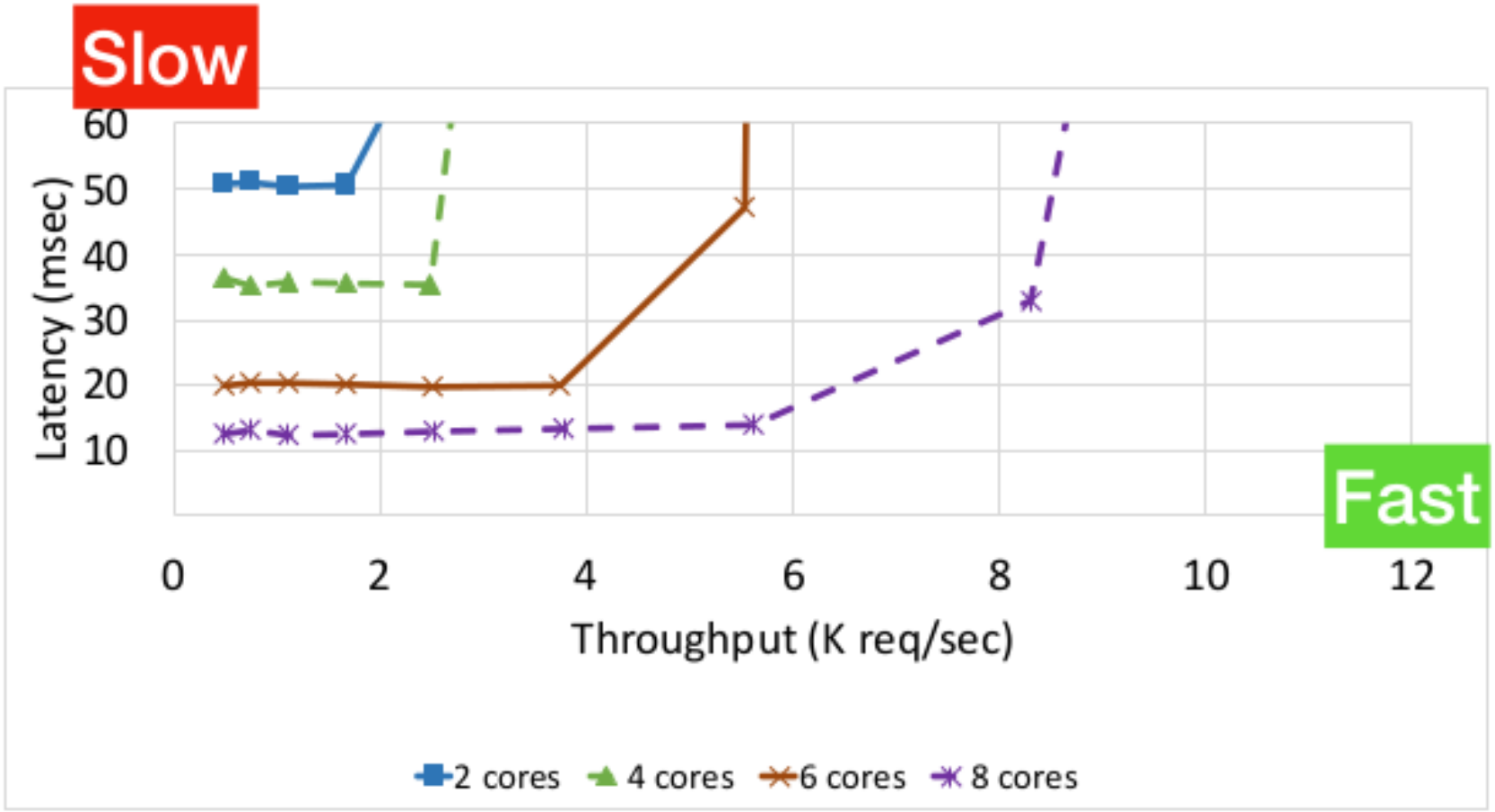


Traditional design: CPU controls GPU invocation and data movements

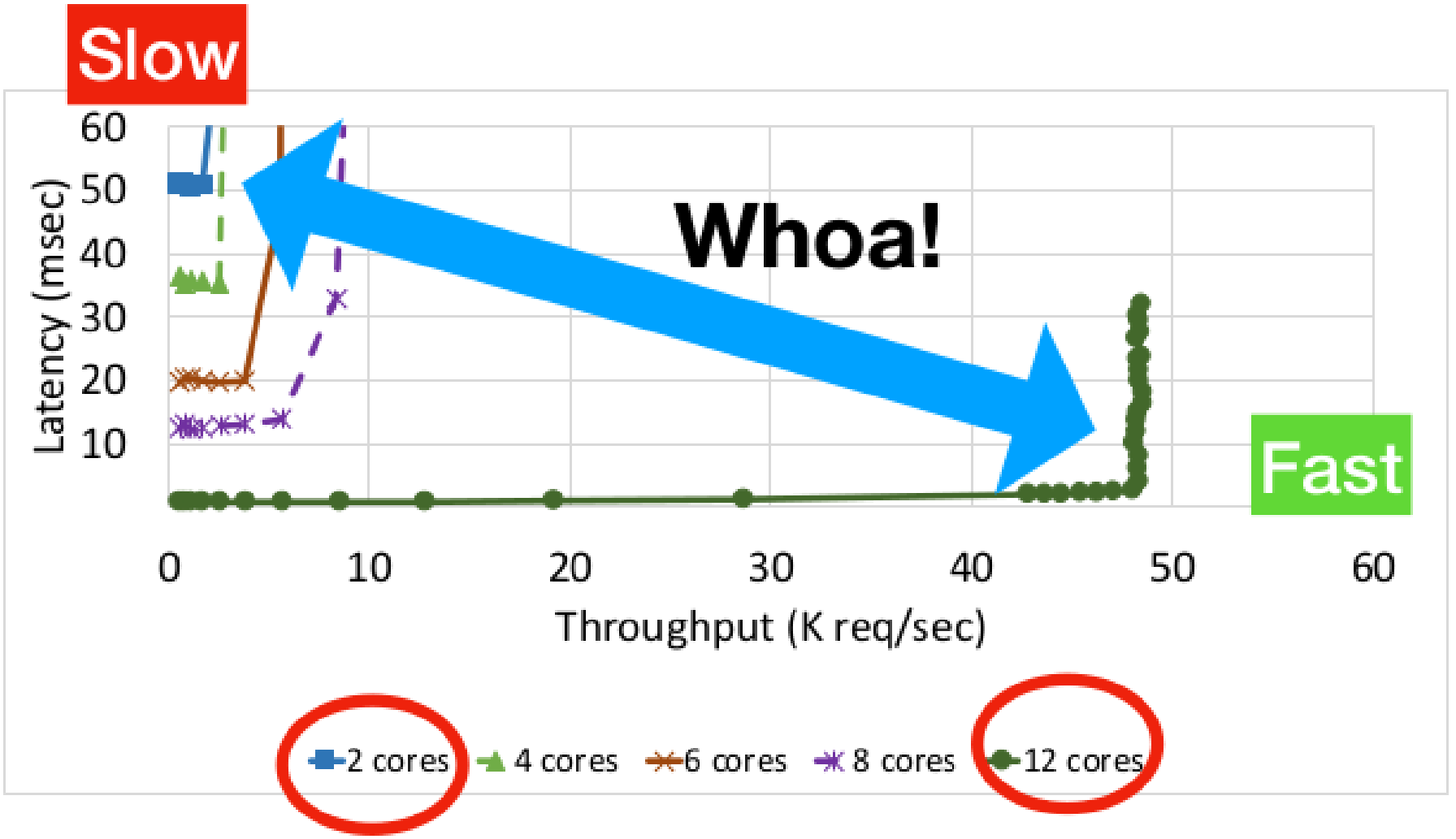


6 GPUs

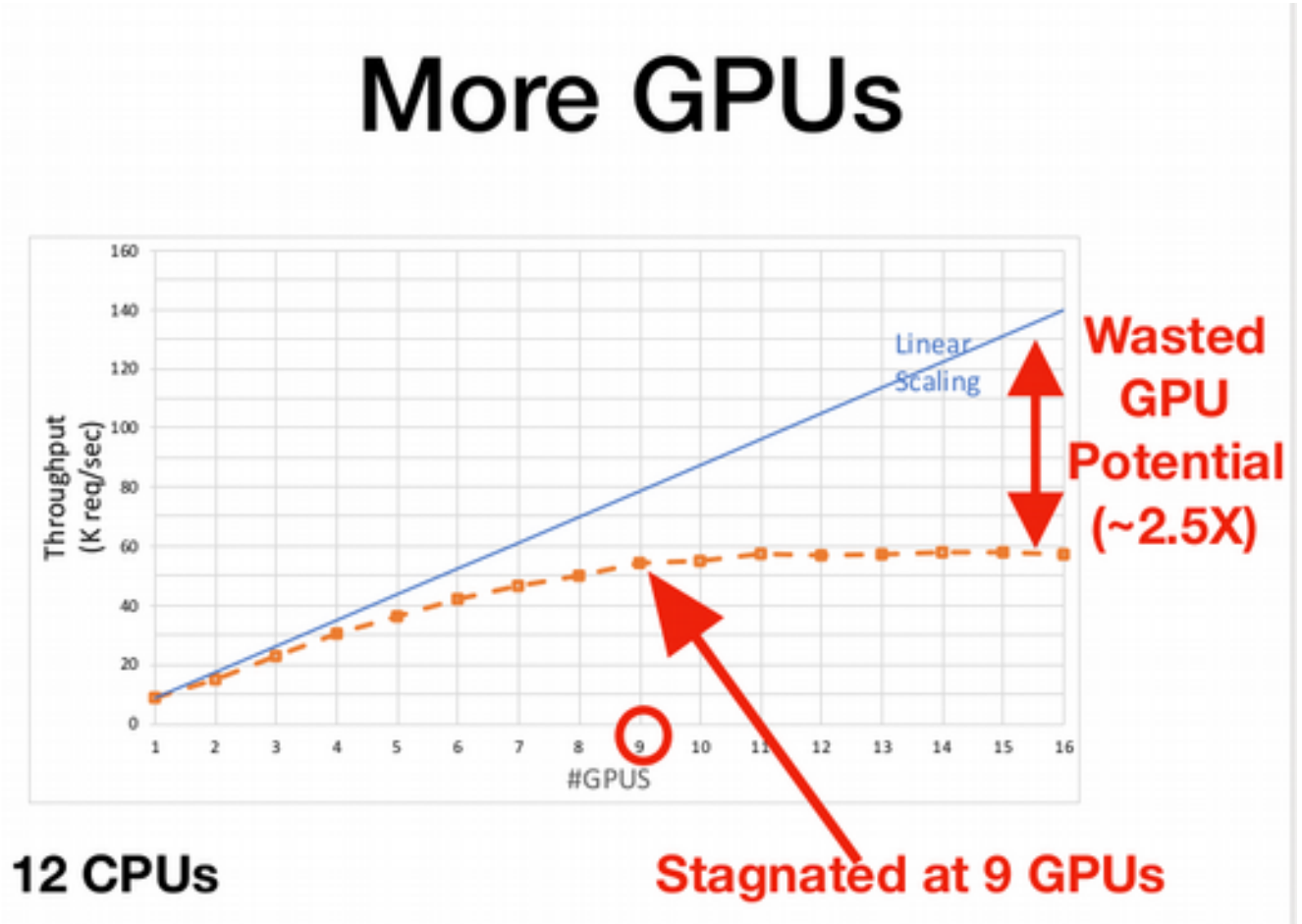
Lets add more CPU cores



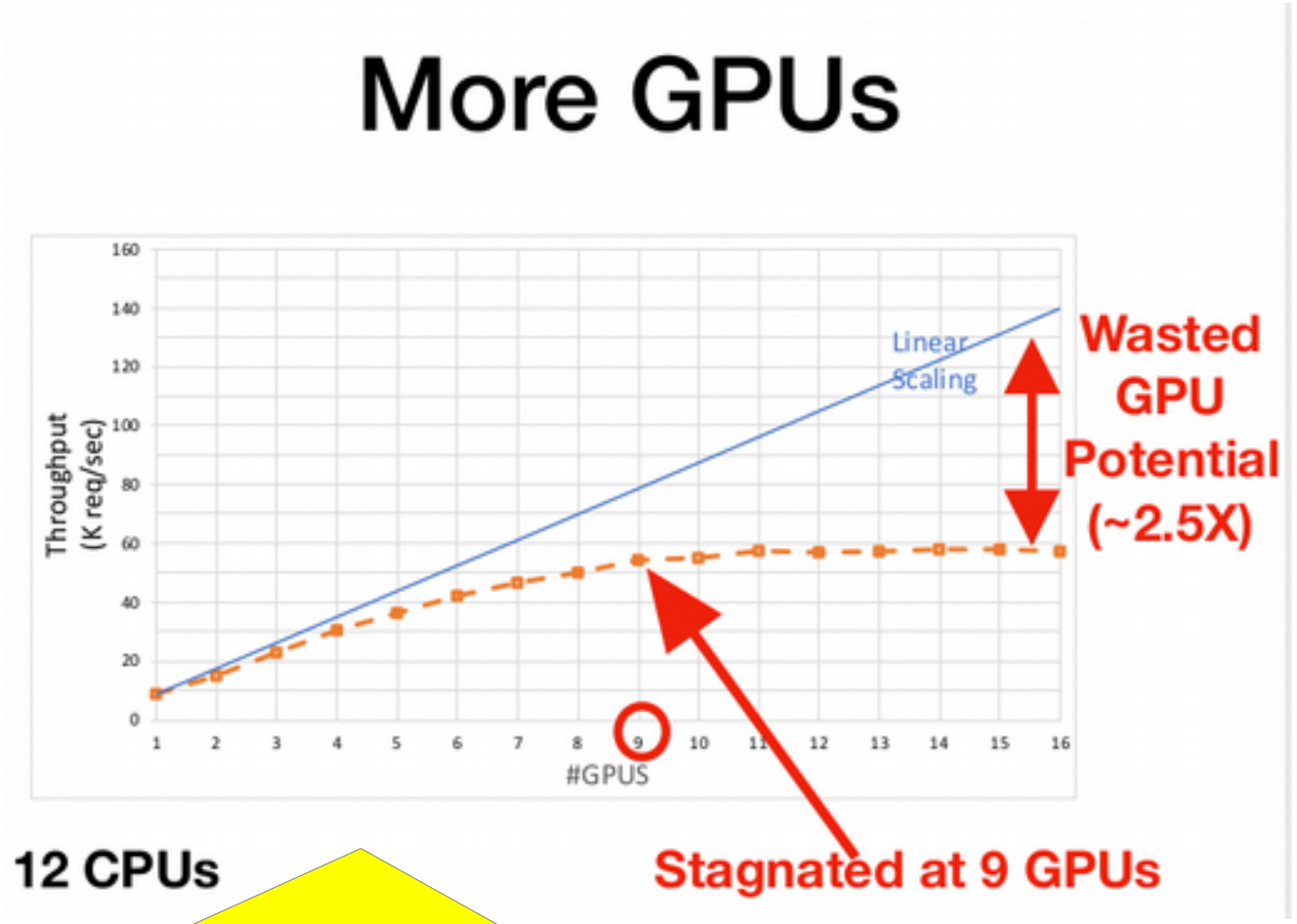
12 CPU cores needed!



12 CPUs are not enough to scale!



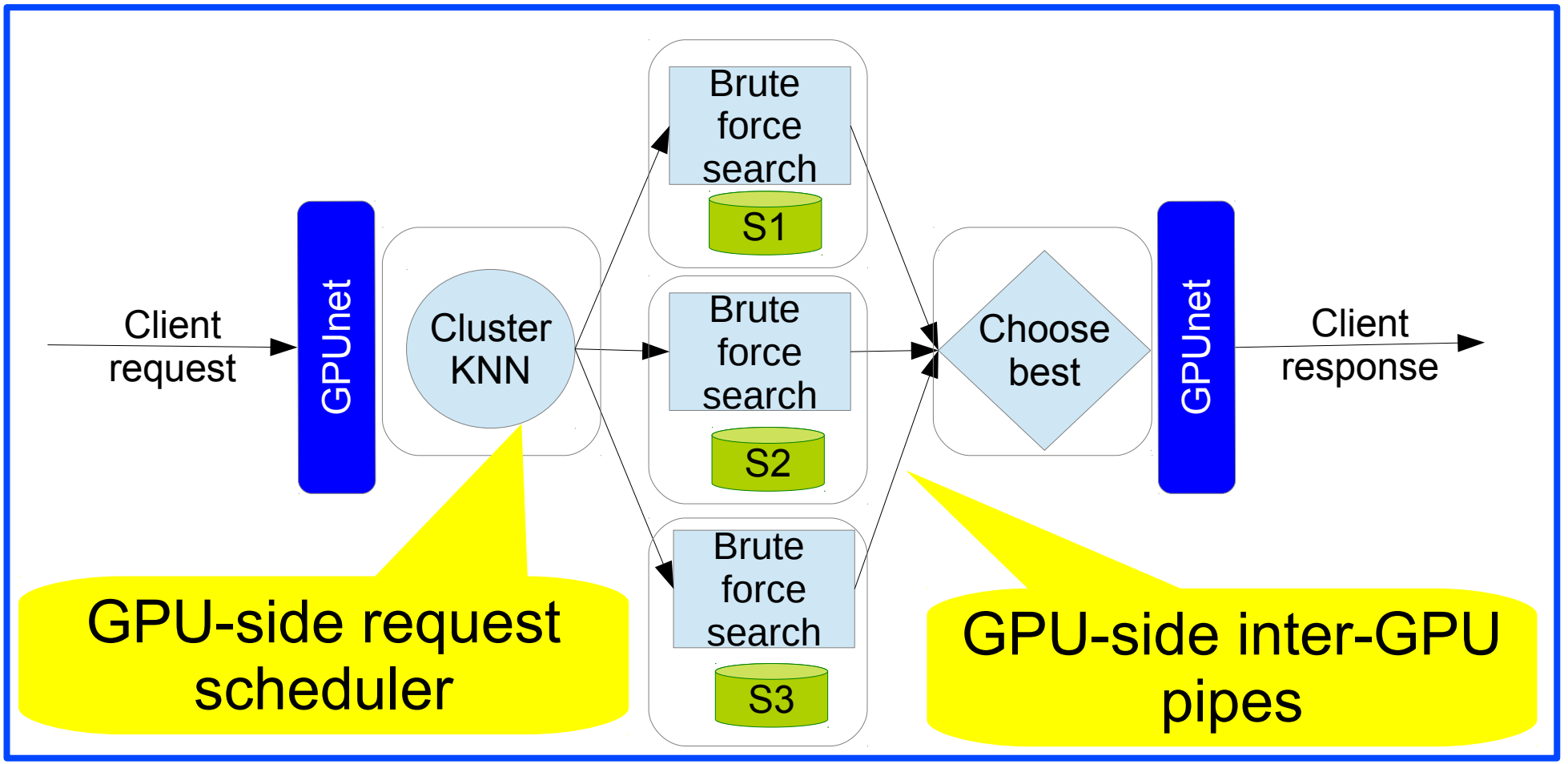
12 CPUs are not enough to scale!



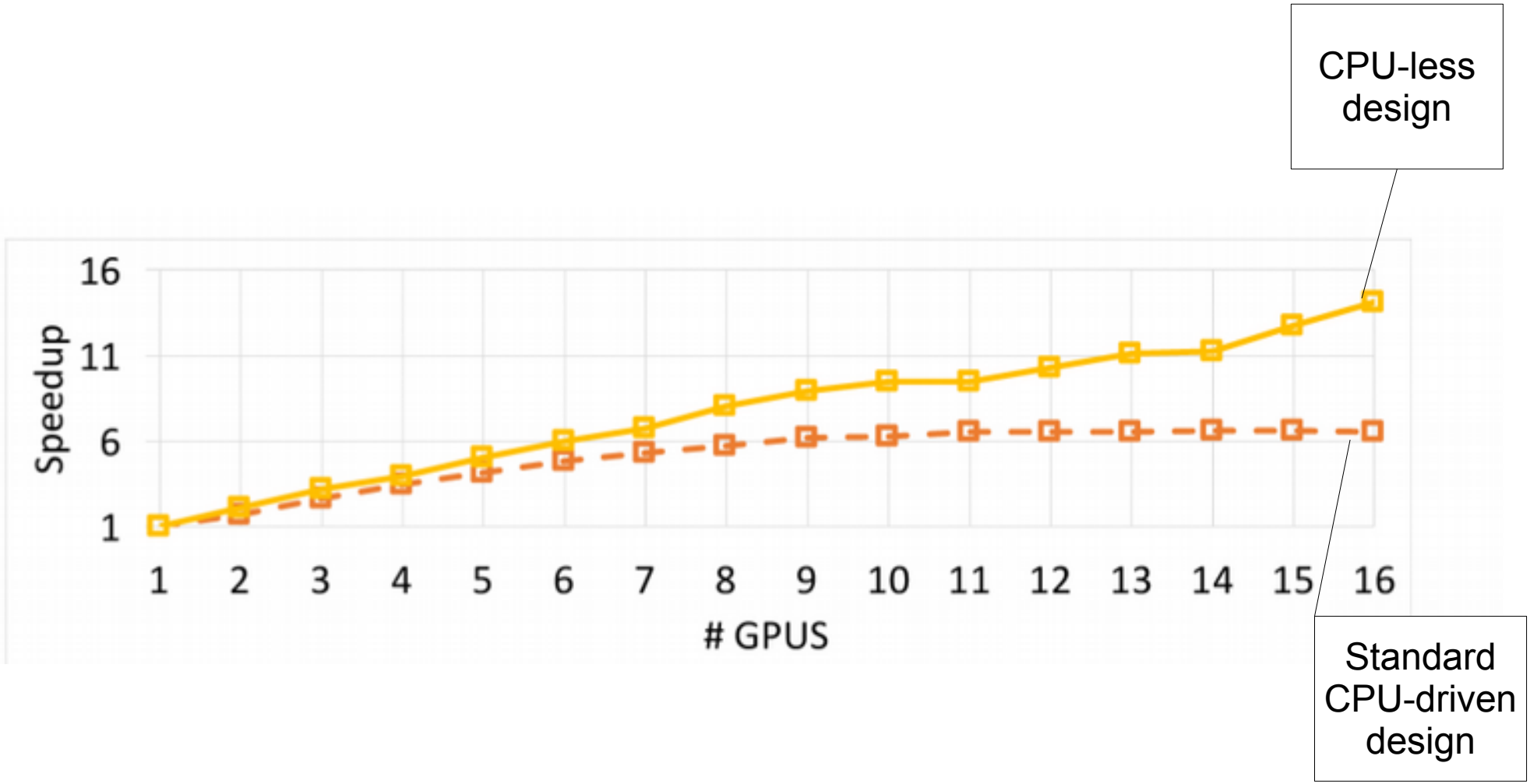
The problem is inherent in the CPU-driven design [PACT19]

The case for **CPU-less** multi-GPU server design

PACT 19



CPU-less Multi-GPU network server



CPU-less design: better scaling

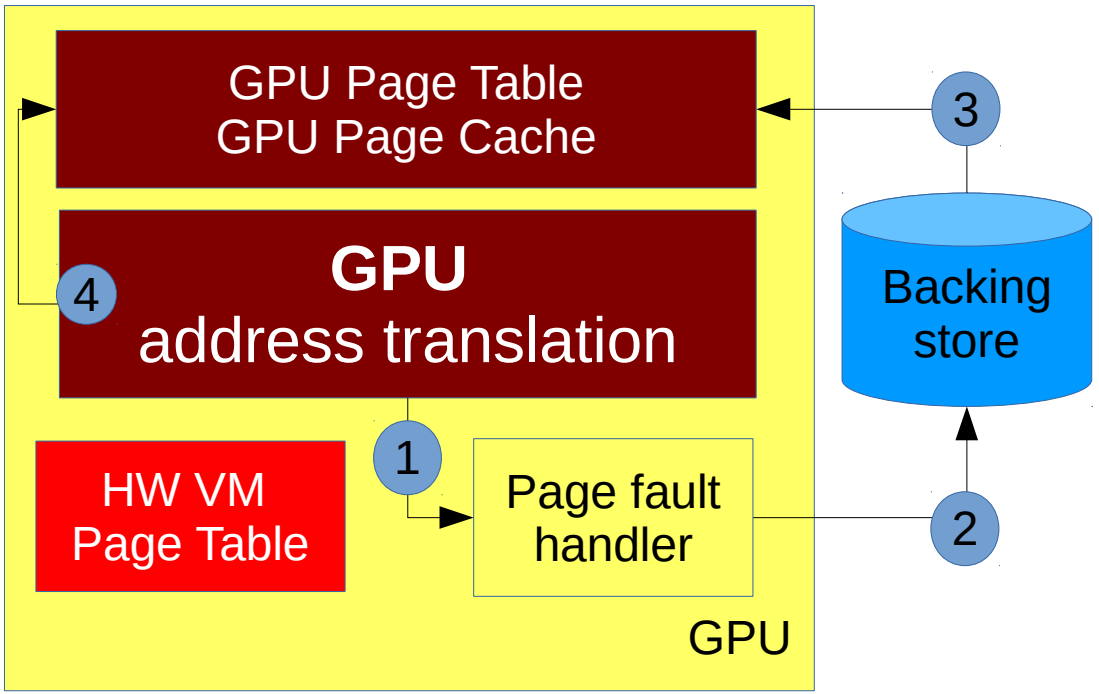
CPU's role

Do the setup
Then leave

GPU-centric Virtual Memory management

ISCA16

- GPU manages its own page tables
- GPU mmap



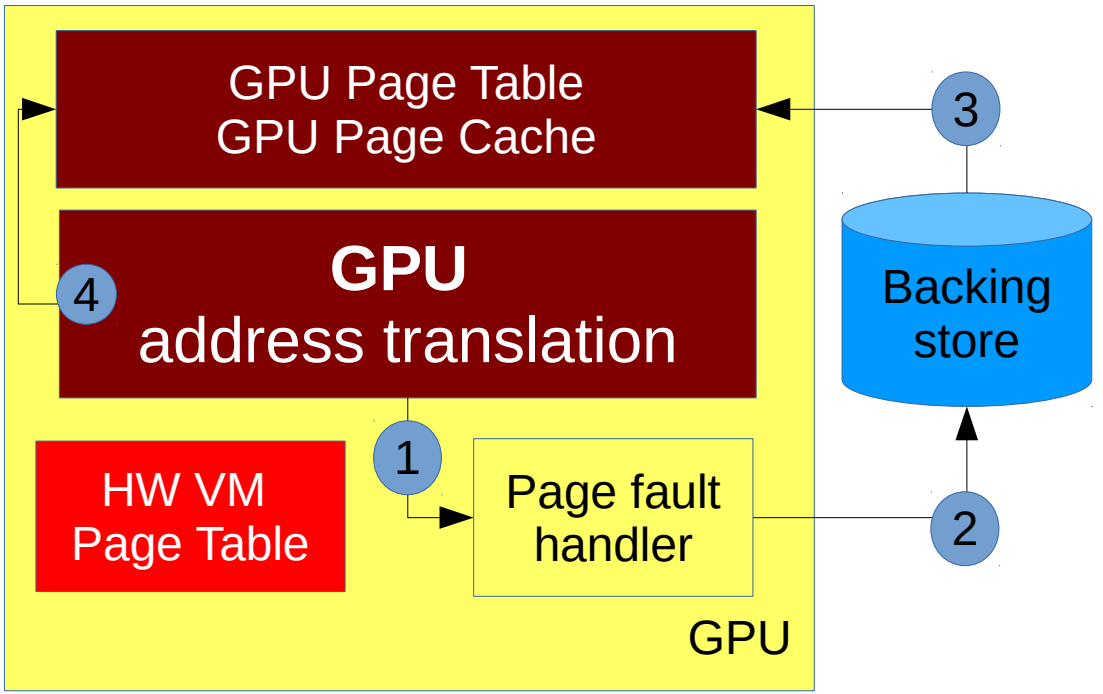
GPU-centric Virtual Memory management

ISCA16

- GPU manages its own page tables
- GPU mmap

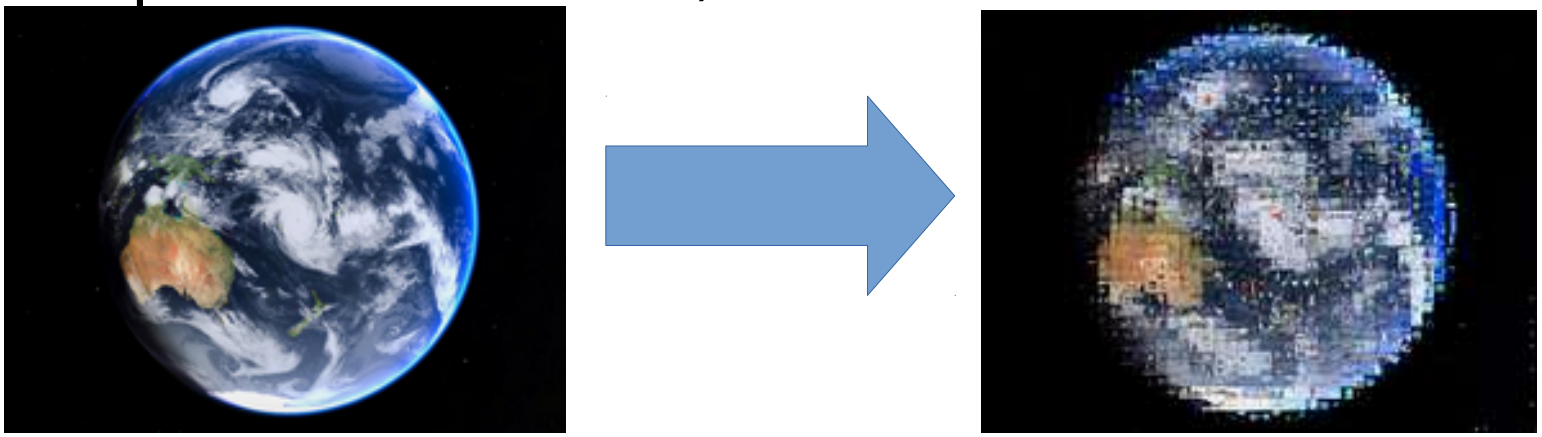
Implementation:

- **Software address translation**
- Overheads hidden thanks to HW multithreading



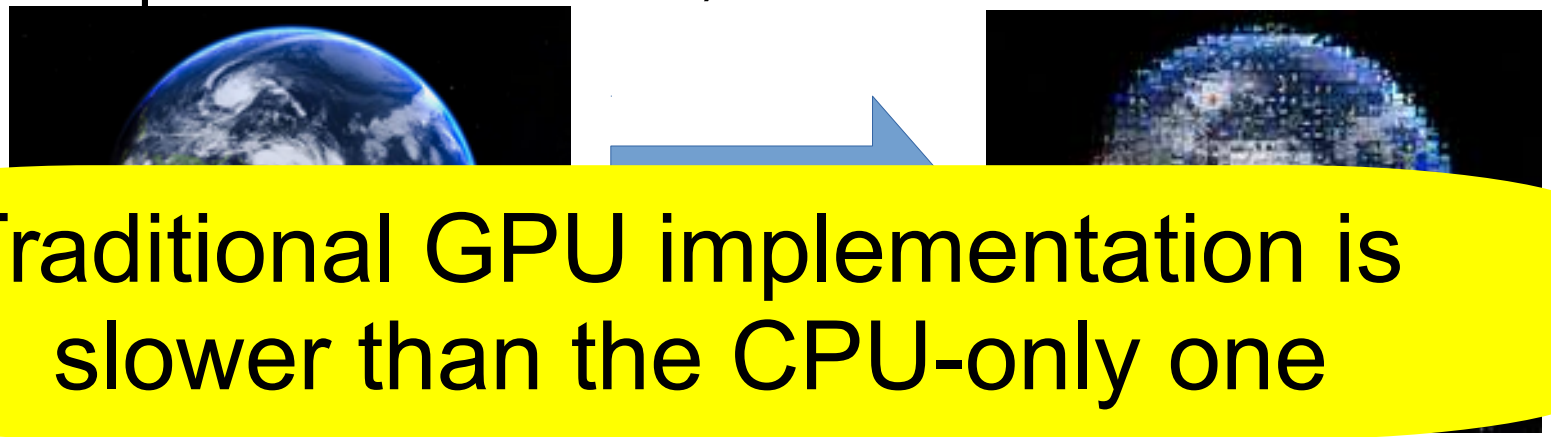
GPU-centric Virtual Memory management

- Application: Image collage
- GPU mmaps **a 40GB DB** file, data-driven accesses

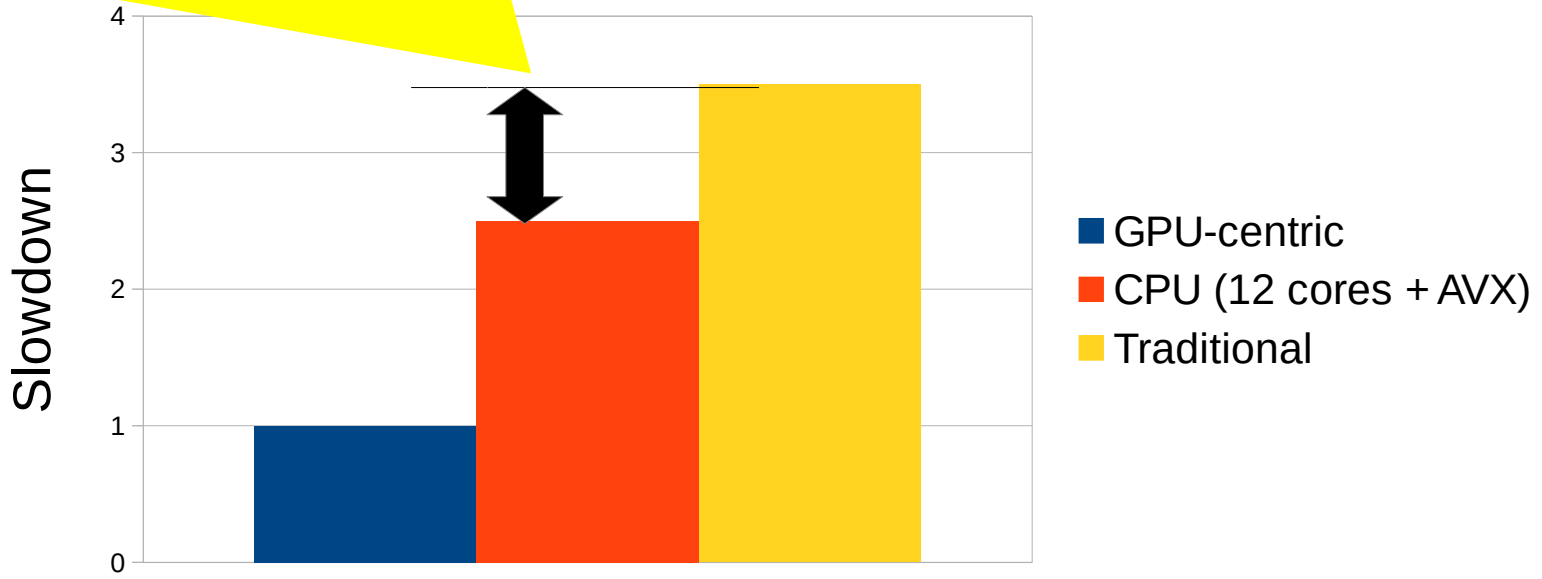


GPU-centric Virtual Memory management

- Application: Image collage
- GPU mmaps **a 40GB DB** file, data-driven accesses



Traditional GPU implementation is slower than the CPU-only one



Summary so far

- GPU-centric I/O services
  - Simplify code development
  - Enable transparent performance optimization
  - Leverage new hardware features
- Eliminate CPU from data path
- Eliminate CPU from control path

Summary so far

- GPU-centric I/O services
  - Simplify code development
  - Enable transparent performance optimization
  - Leverage new hardware features
- Eliminate CPU from data path
- Eliminate CPU from control path

Not everything is great…

Issues…

- GPUs are bad at system software
  - Function calls are extremely slow
  - Large code base causes slowdowns
  - No preemption, even not software-only
- GPU access to PCIe is ***slow***
  - even writes block multiple threads for a few usec
- No architecture support for persistent kernels

Issues…

- GPUs are bad at system software
  - Function calls are extremely slow
  - Large code base causes slowdowns
  - No preemption, even not software-only
- GPU access to PCIe is ***slow***
  - even writes block multiple threads for a few usec
- No architecture support for persistent kernels

Can we reap the benefits of CPU-less design while bypassing these issues?

GPU I/O V2.0

- Convenient access with *minimal* I/O stack on GPU
- Leverage new hardware features

GAIA: extending CPU page cache into GPU memory

[USENIX ATC19]

- Enables `mmap` for GPU by using GPU HW PF
- May cache/prefetch file data in GPU memory
- Insights:
  - Slim GPU driver API for enabling host page cache integration
  - Page cache consistency model
  - OS page cache and Linux kernel modifications for consistency support

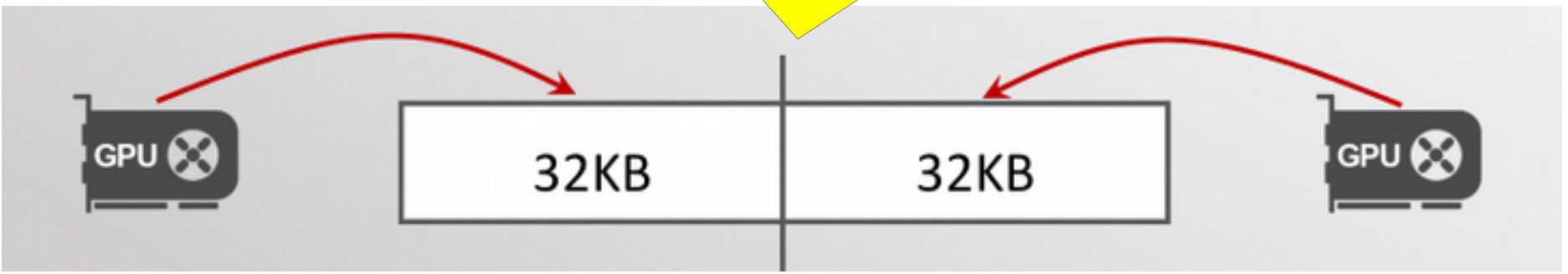
Question: can we use strong consistency in the page cache?

- Current practice in NVIDIA Unified Virtual Memory
- Single owner semantics: the page migrates to the requesting processor

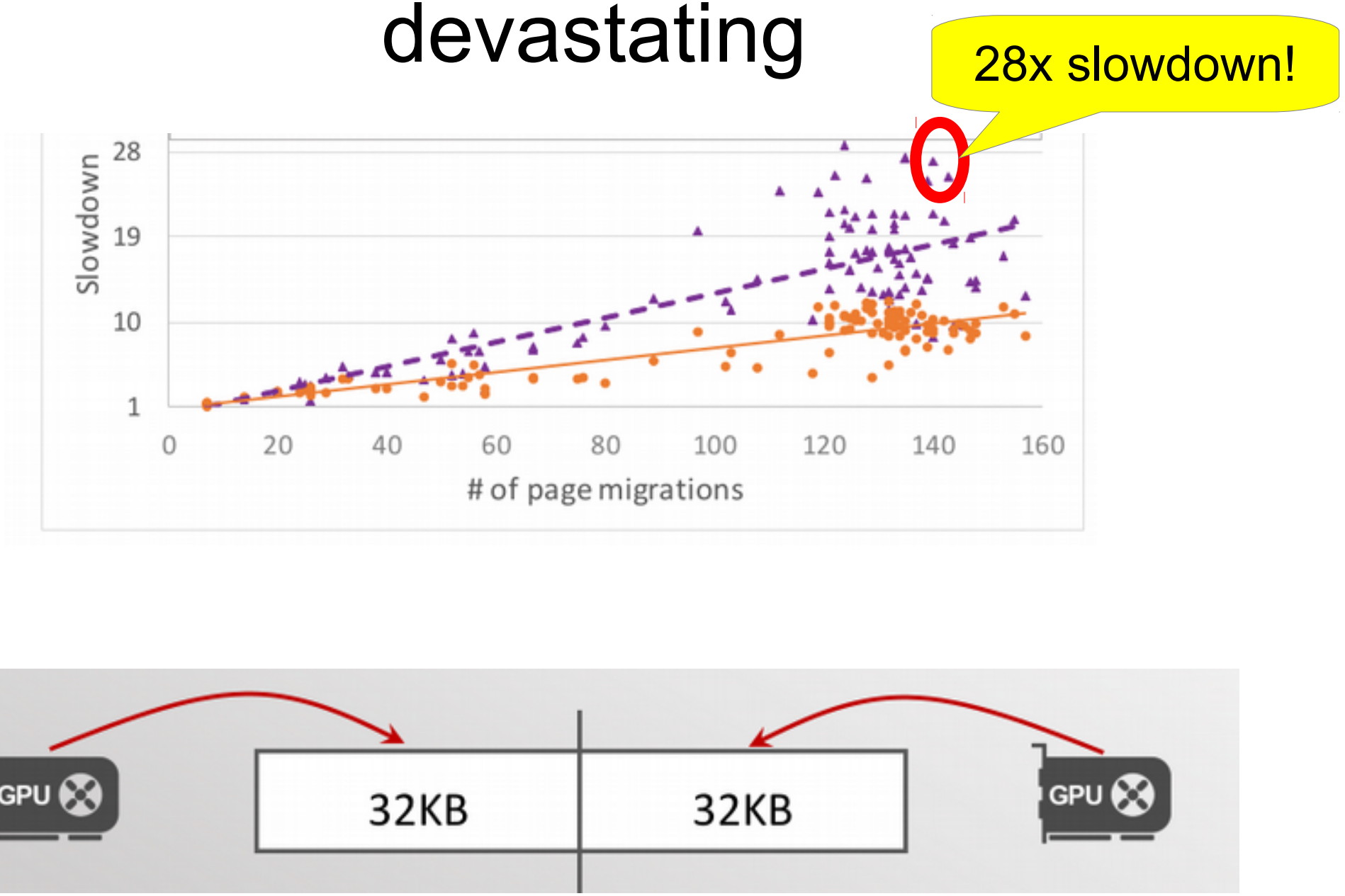
Question: can we use strong consistency in the page cache?

- Current practice in NVIDIA Unified Virtual Memory
- Single owner semantics: the page migrates to the requesting processor

But GPU page is 64KB!
False sharing inevitable
(also in real applications)

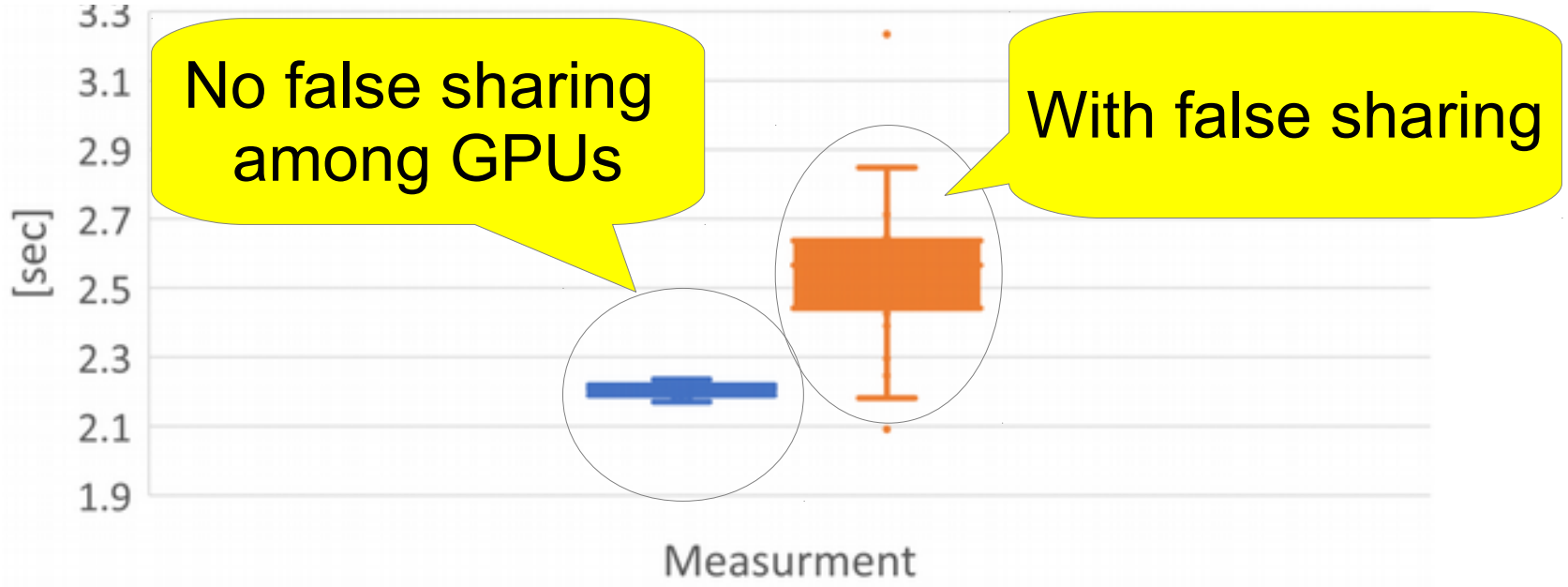


Extreme false sharing is devastating



It even affects CPU processes

CPU-only run of an HPC workload



Lazy Release Consistency to rescue

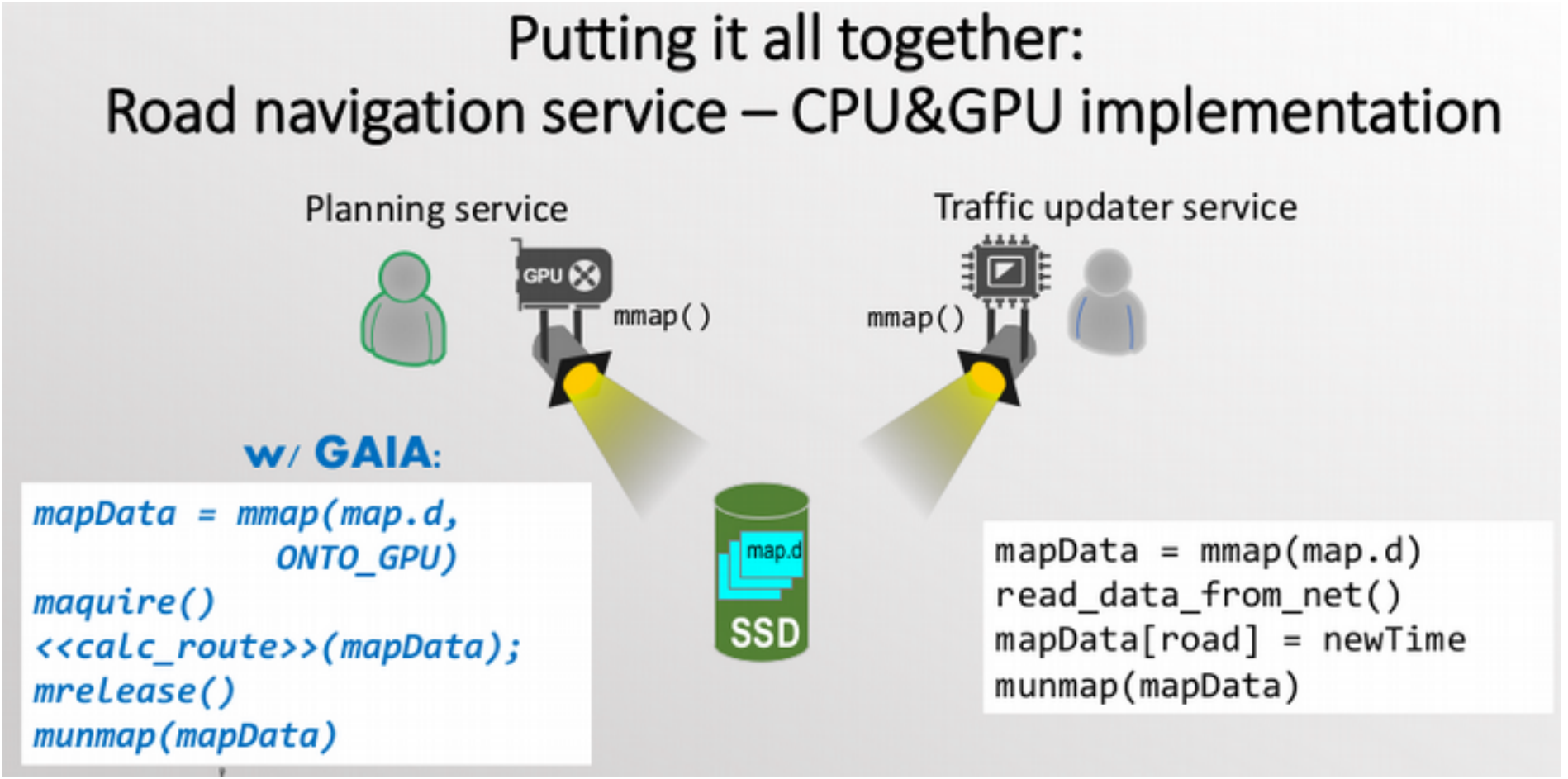
- Acquire-release to ensure update
- Version vectors for contention detection
- 3-way merge for conflict resolution
- *Transparent for legacy CPU processes*

GPU management code on the CPU

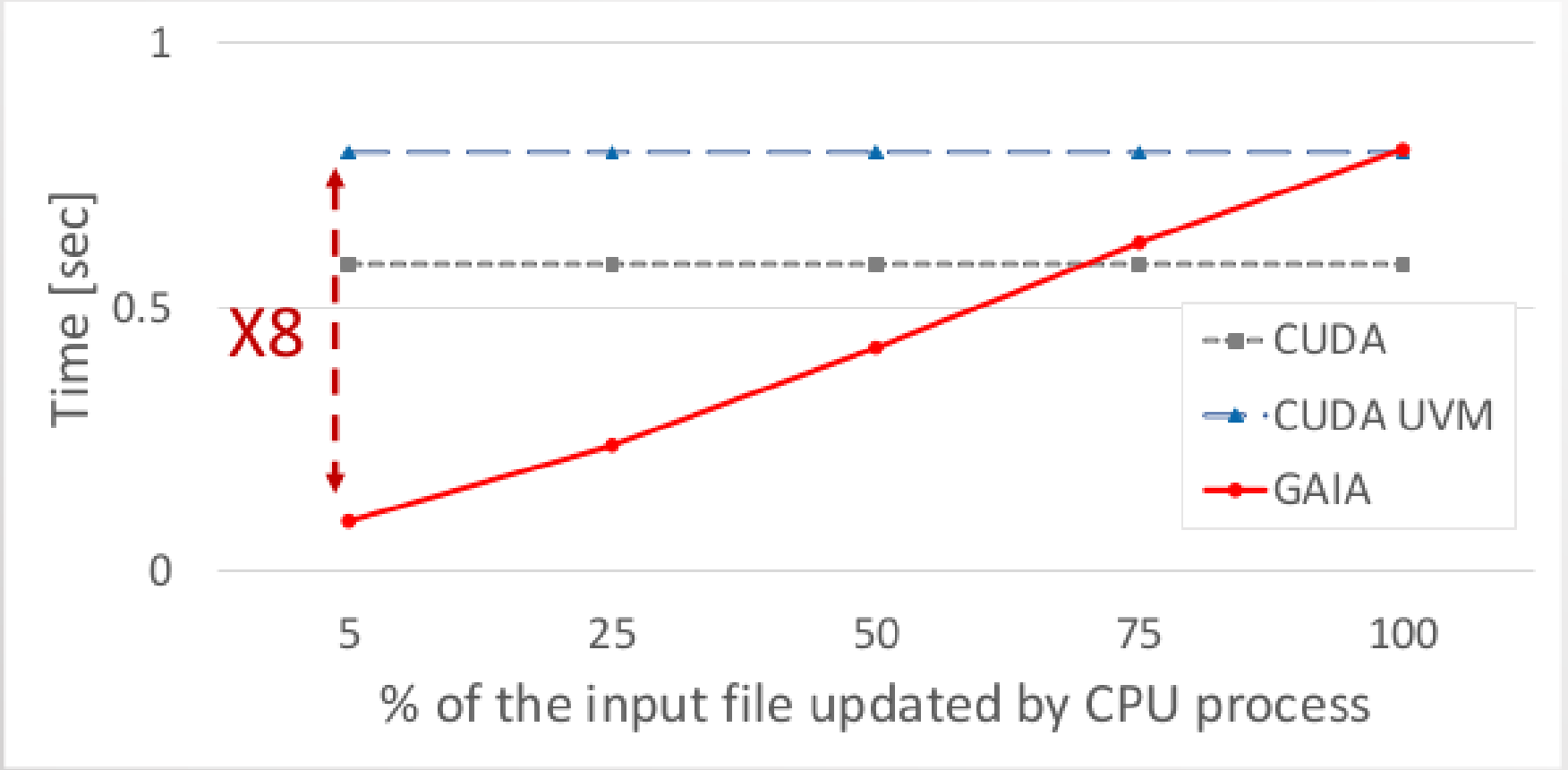
```
int fd=open(«shared_file»);
void* ptr=mmap(…,ON_GPU,fd);
macquire(ptr);
gpu_kernel<<<>>>(ptr);
mrelease(ptr);
```

40% app improvement over strong consistency

Road navigation service

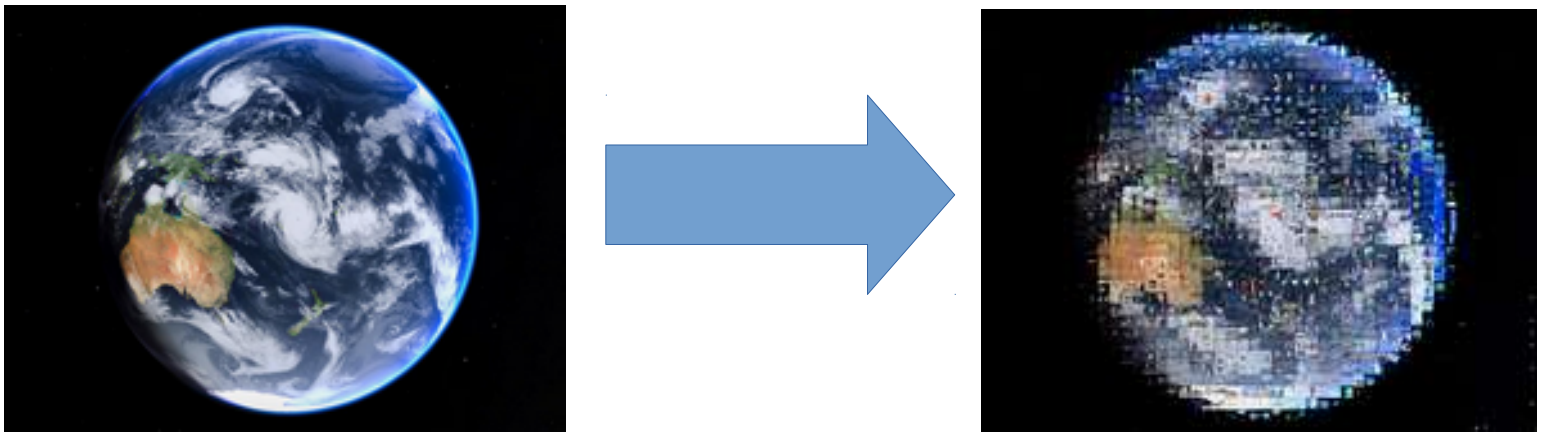


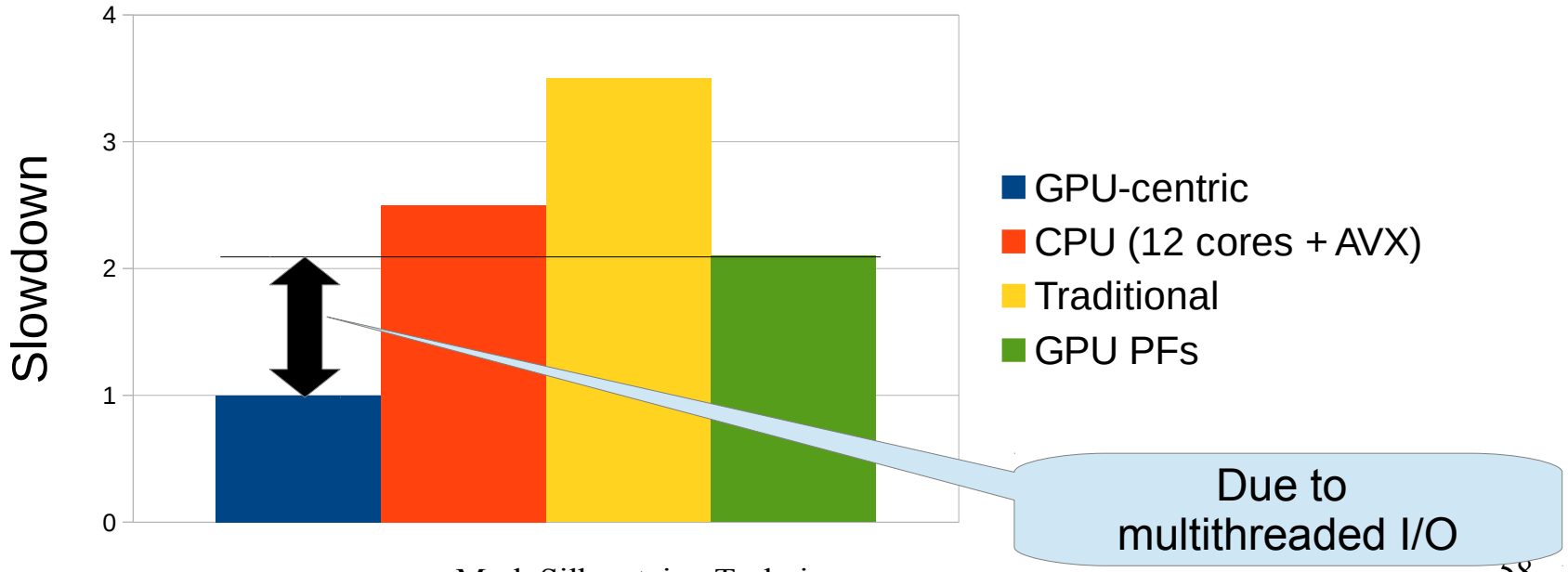
Performance improvement due to fine-grain consistency



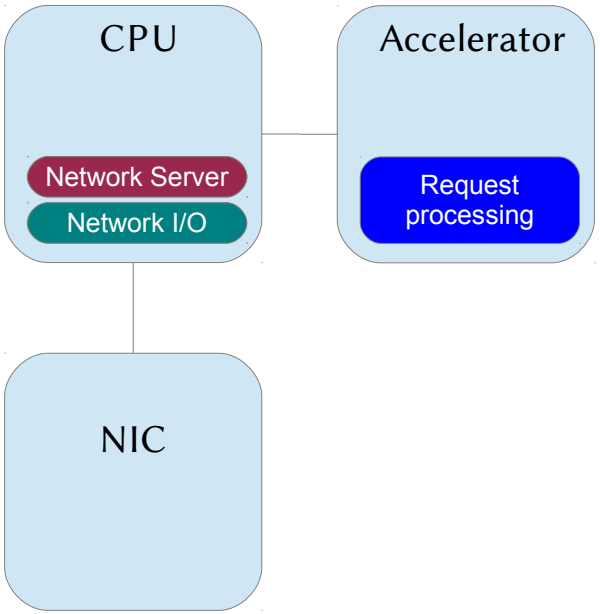
Comparison with software-only PF

- Application: Image collage
- GPU mmaps **a 40GB DB** file, data-driven accesses

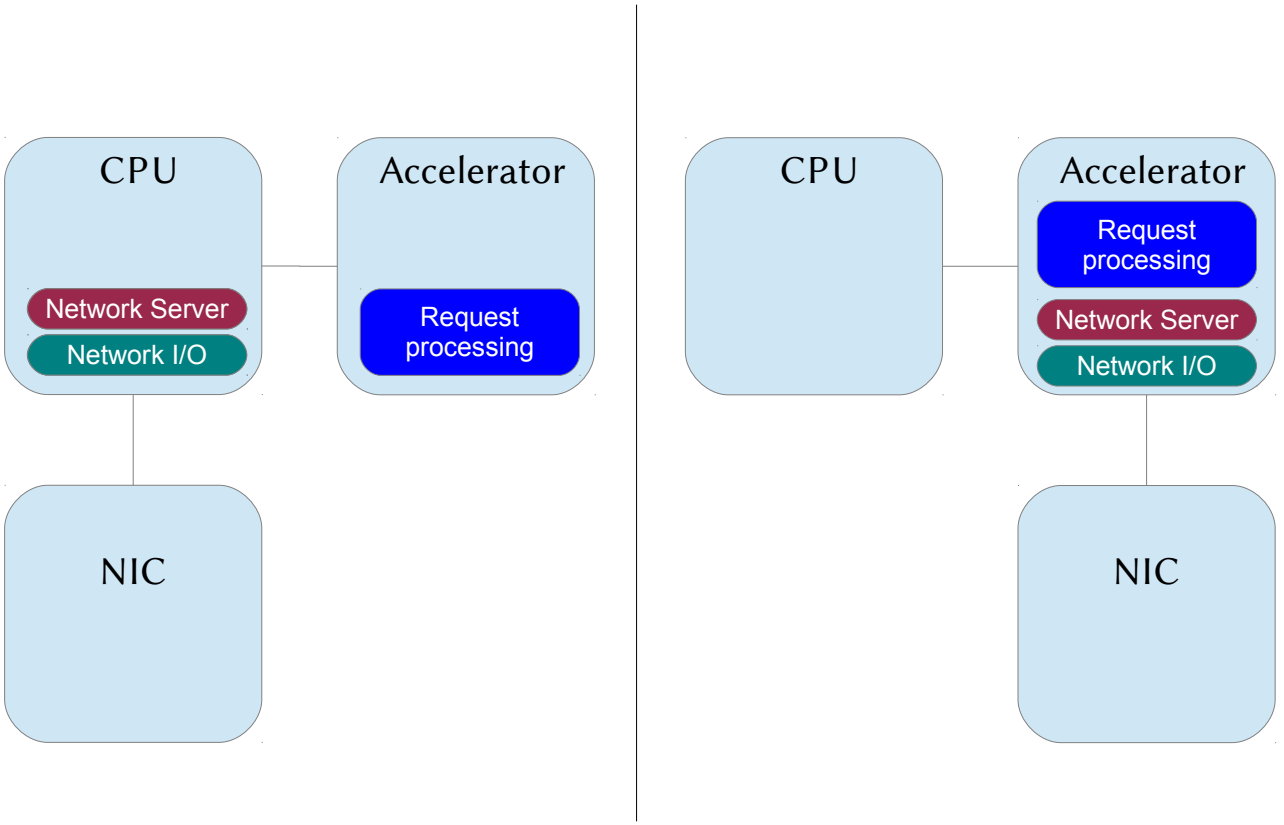




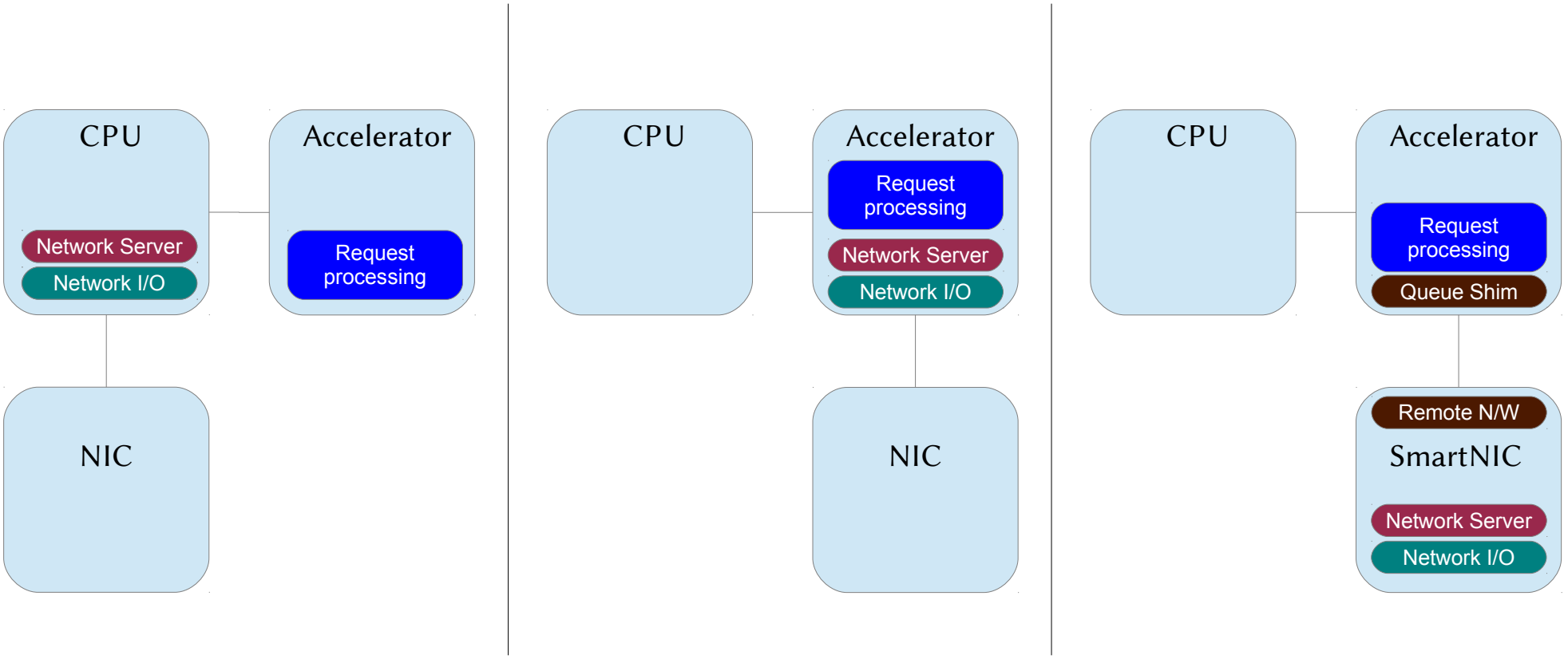
GPU-accelerated server V0.0
Request processing offload



GPU-accelerated server V1.0
GPU-centric, GPU-driven

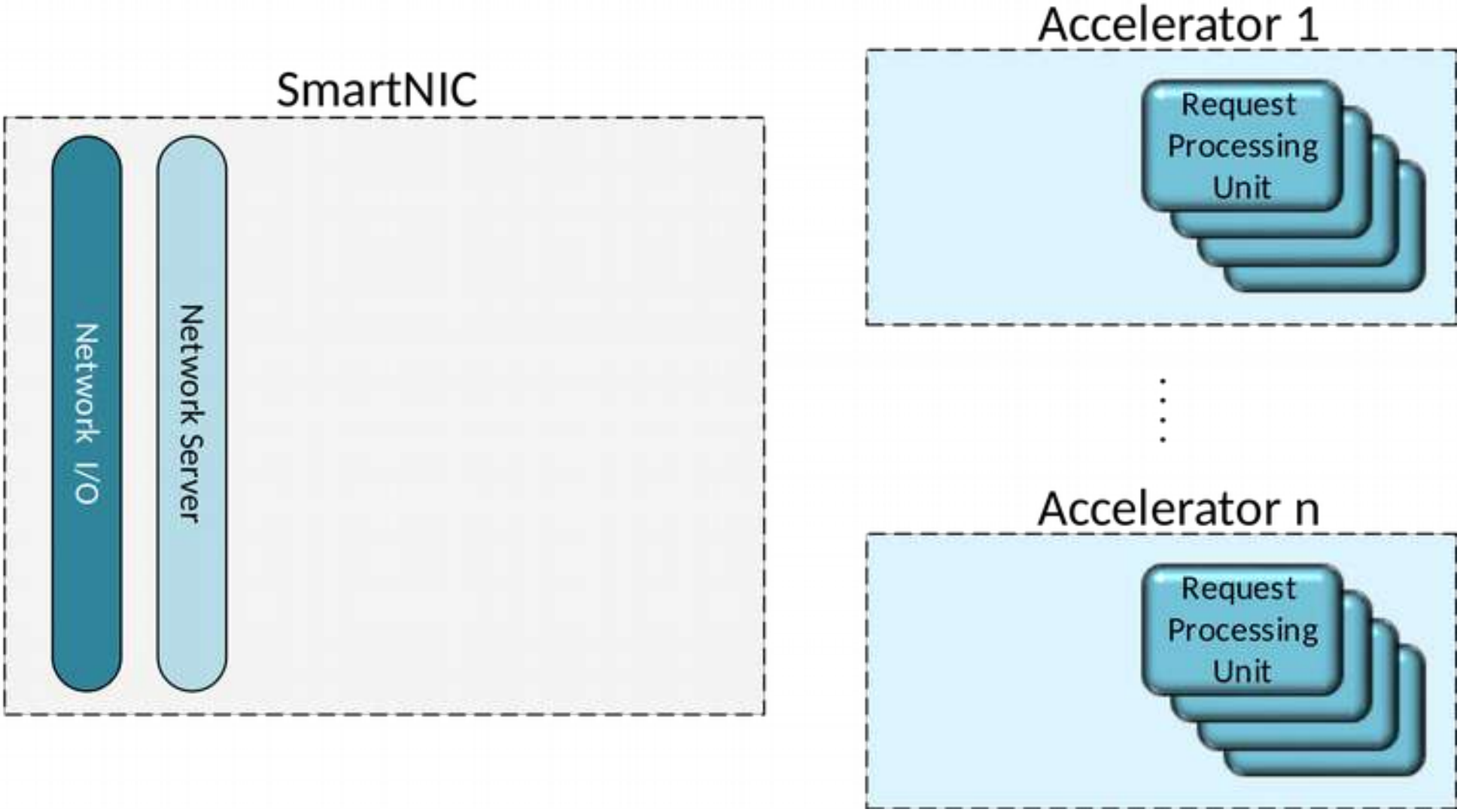


GPU accelerated server V2.0
GPU-centric, SmartNIC-driven

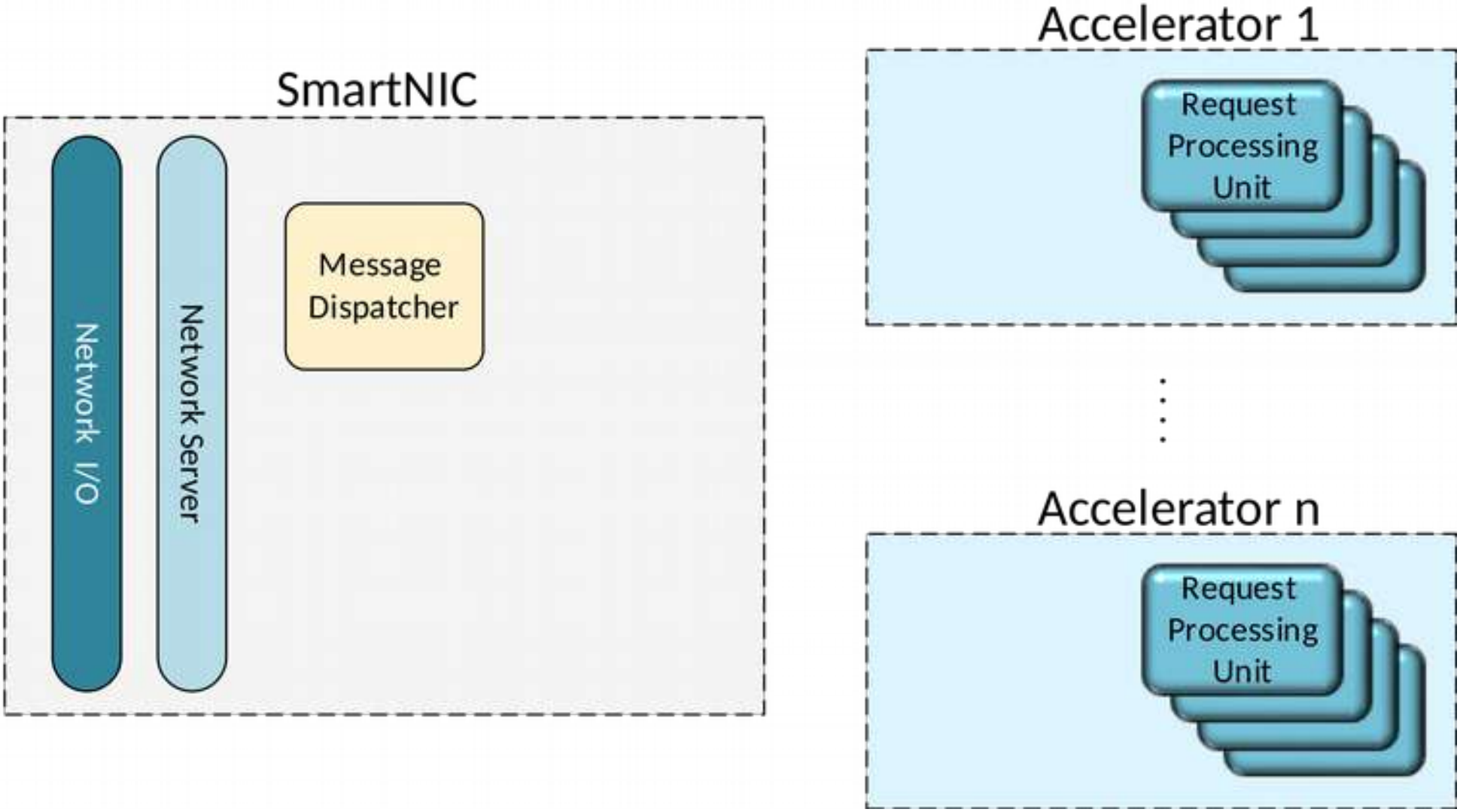


No GPU I/O stack, only simple shim

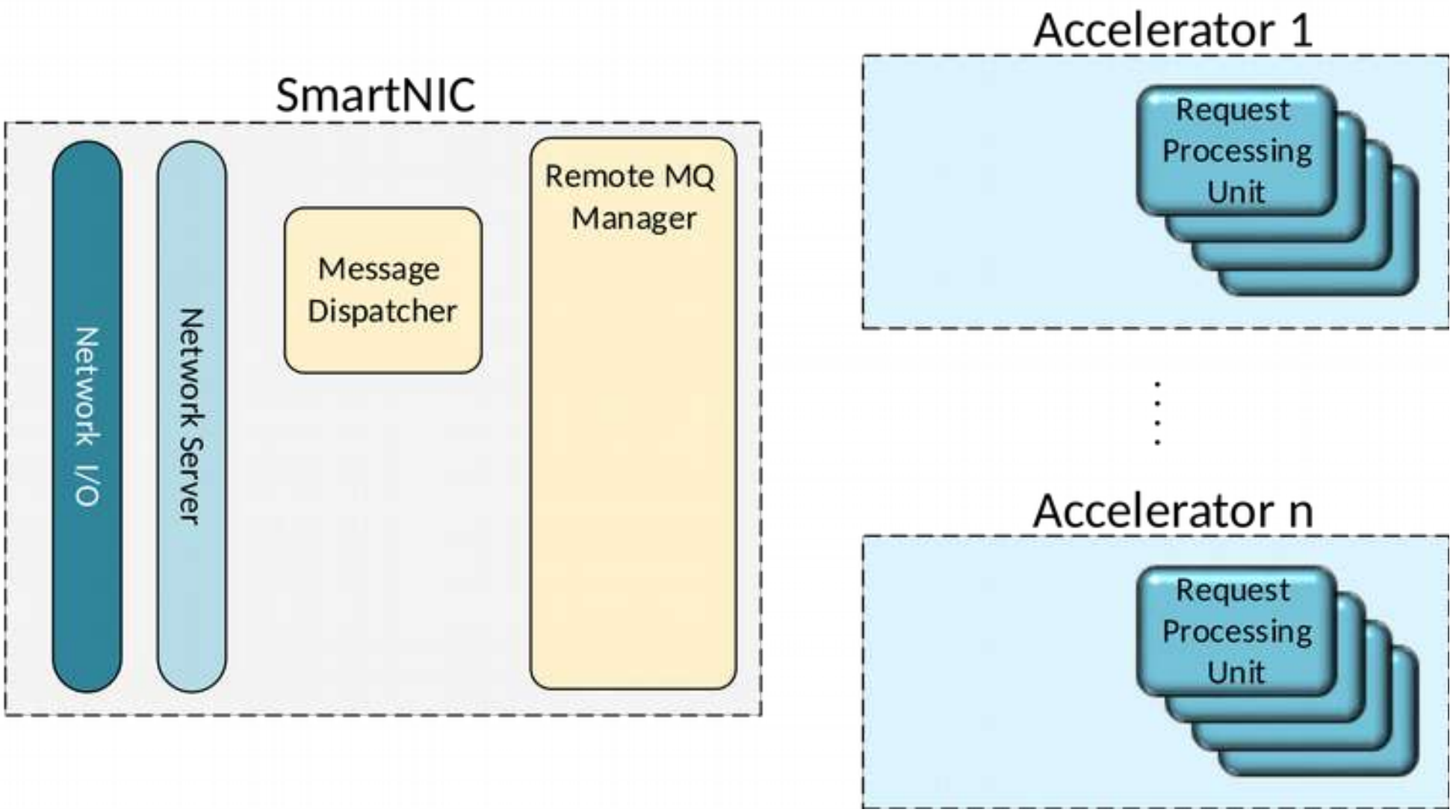
SmartNIC-driven accelerator-centric server



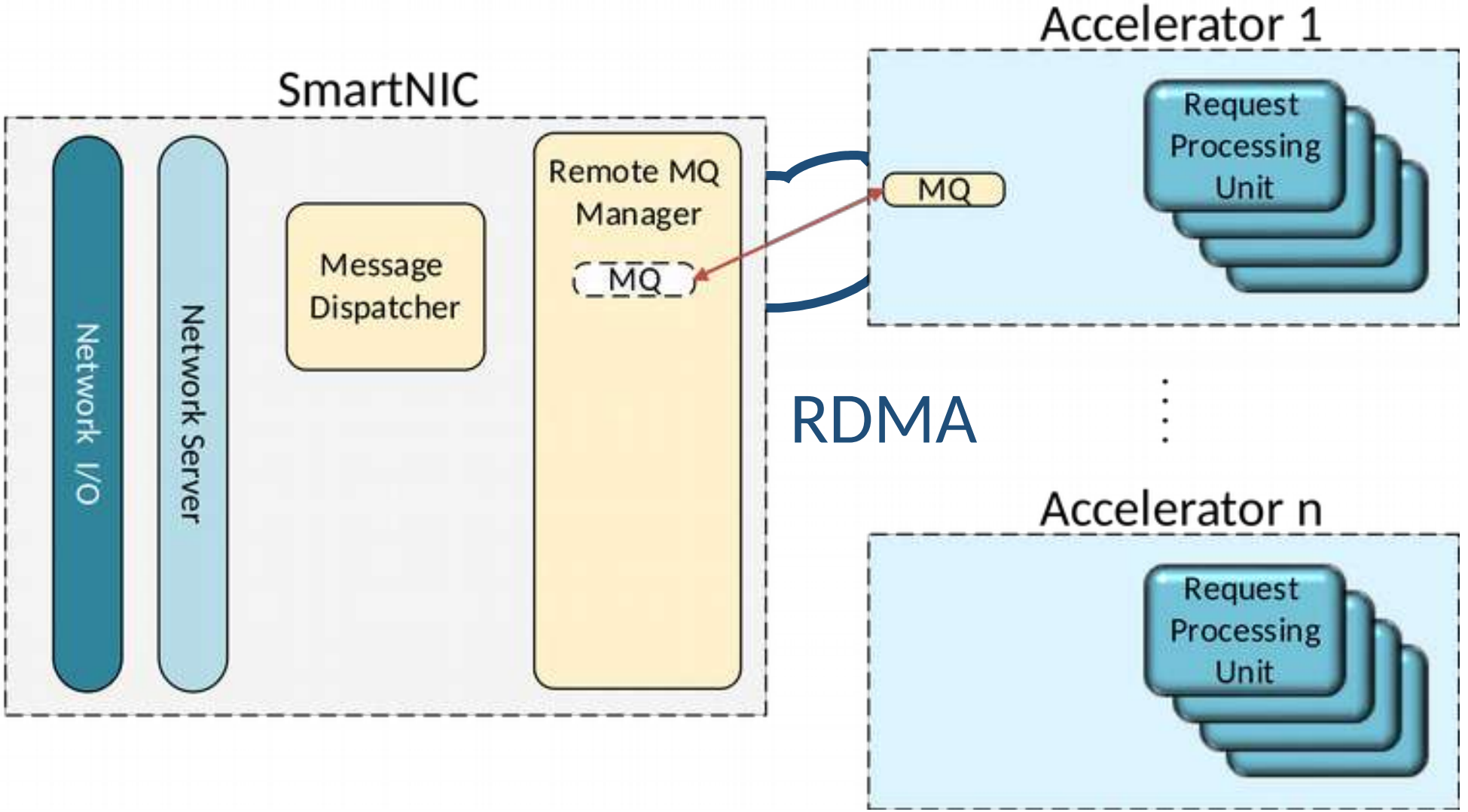
SmartNIC-driven accelerator-centric server



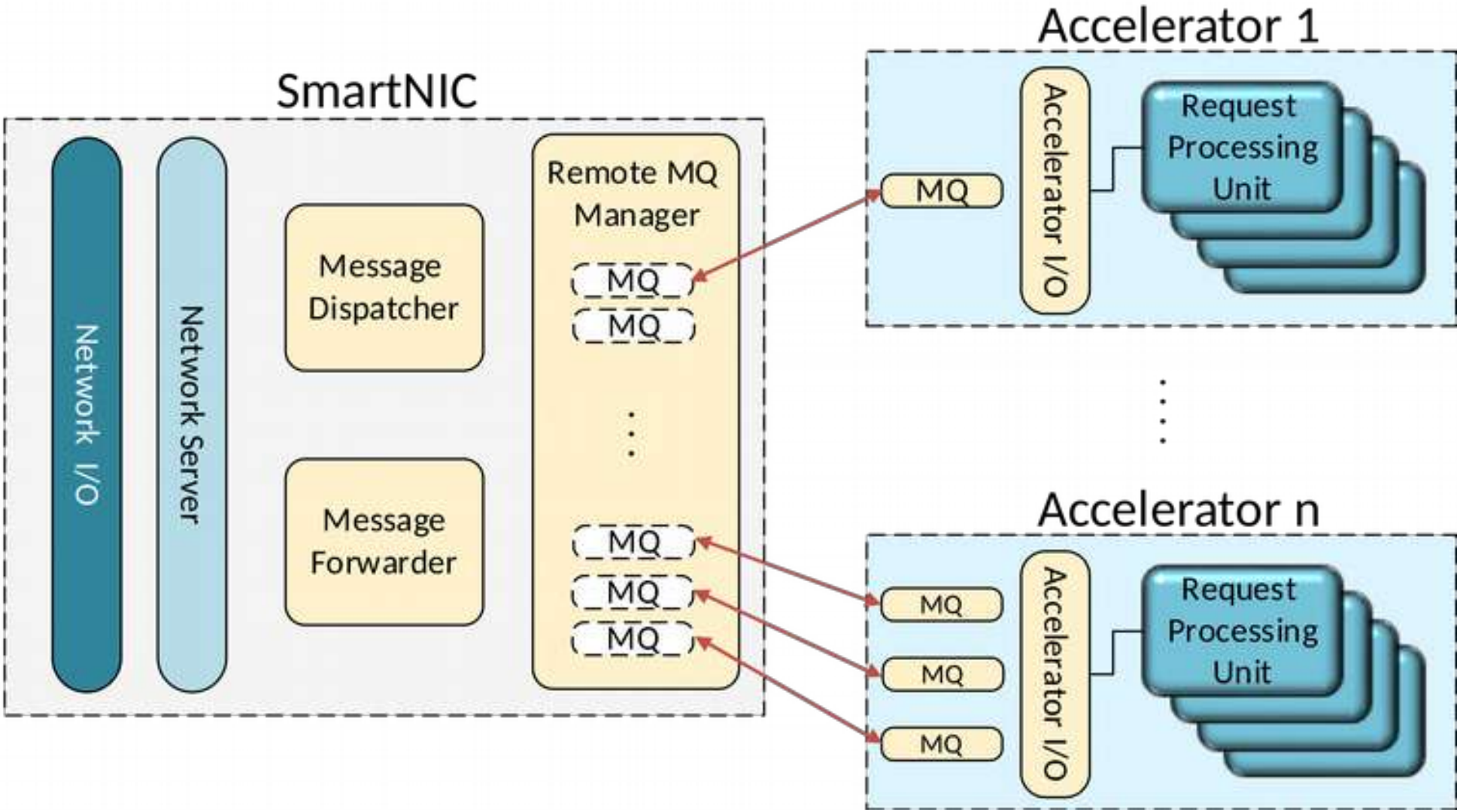
SmartNIC-driven accelerator-centric server



SmartNIC-driven accelerator-centric server



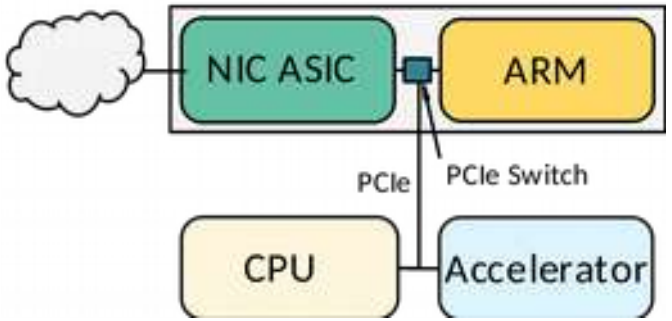
SmartNIC-driven accelerator-centric server



SmartNICs

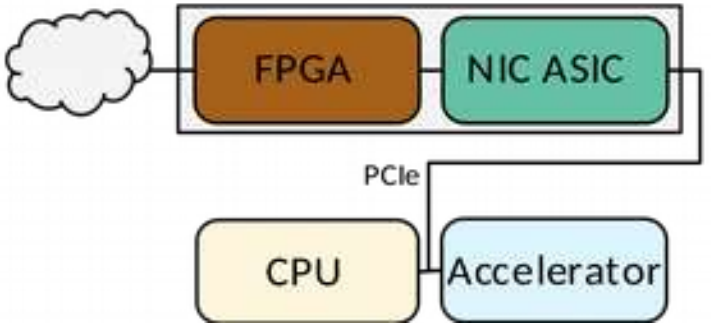
Mellanox Bluefiled

- V1: ARMv8 A72 (8x800MHz)
- BlueOS (Linux)
- OFED (RDMA)
- VMA (User-level N/W stack)



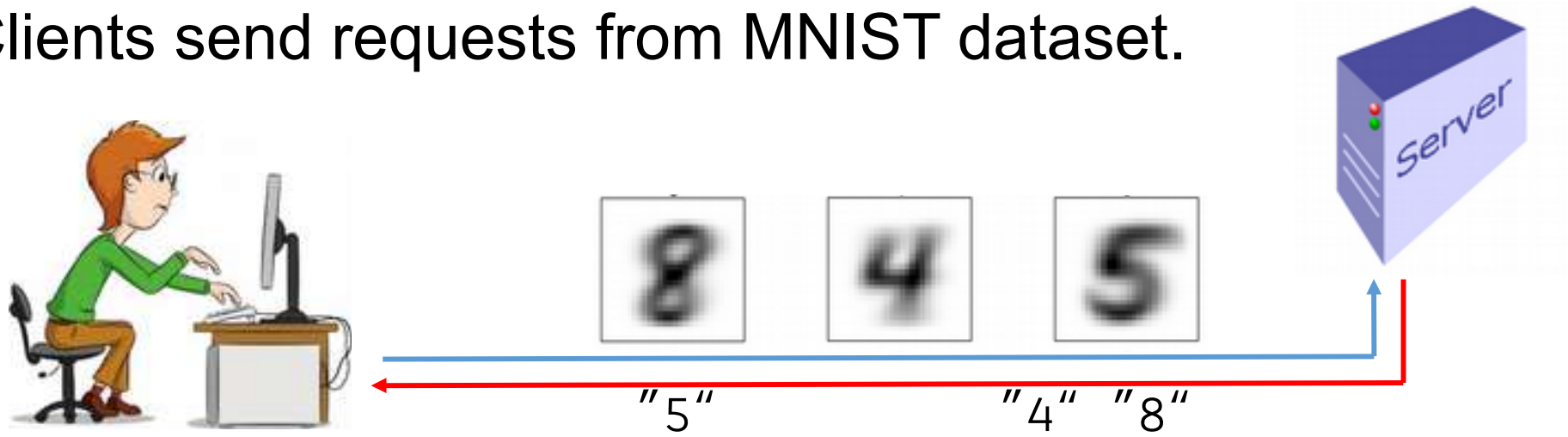
Mellanox Innova

- Xilinx FPGA XCKU060
- Bump-in-the-wire
- Software support: NICA (USENIX ATC19)



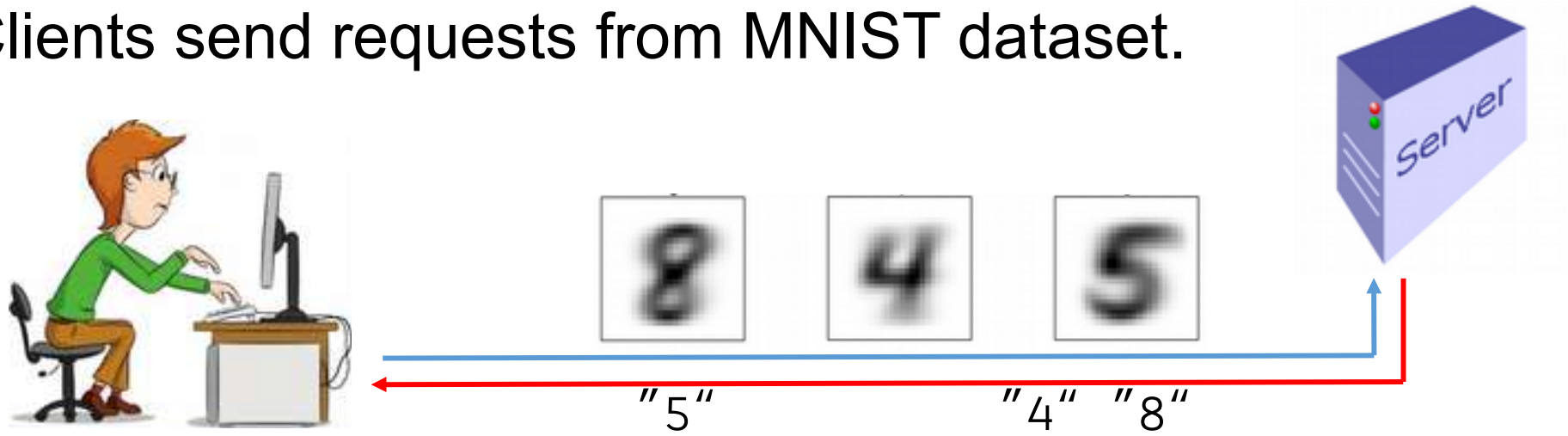
LeNet CNN inference Server

- LeNet developed using **TensorFlow** and optimized using the **TVM compiler**.
- Runs as a **persistent kernel** (kernel invocations via nested parallelism)
- Clients send requests from MNIST dataset.



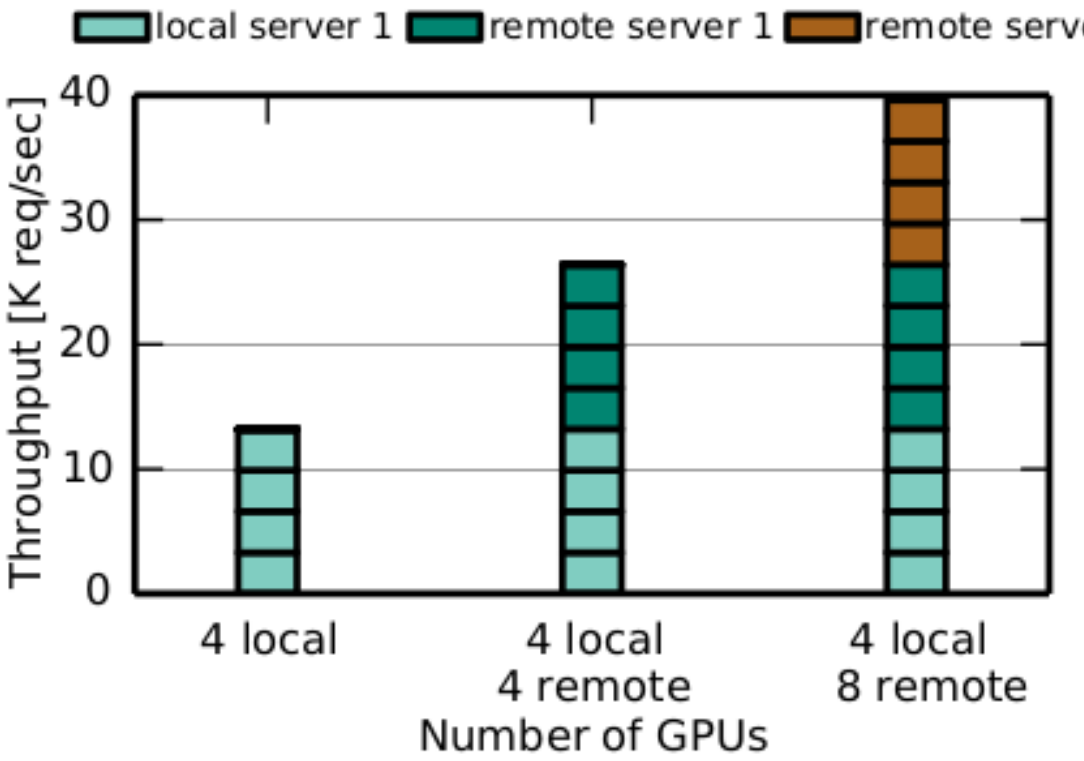
LeNet CNN inference Server

- LeNet developed using **TensorFlow** and optimized using the **TVM compiler**.
- Runs as a **persistent kernel** (kernel invocations via nested parallelism)
- Clients send requests from MNIST dataset.



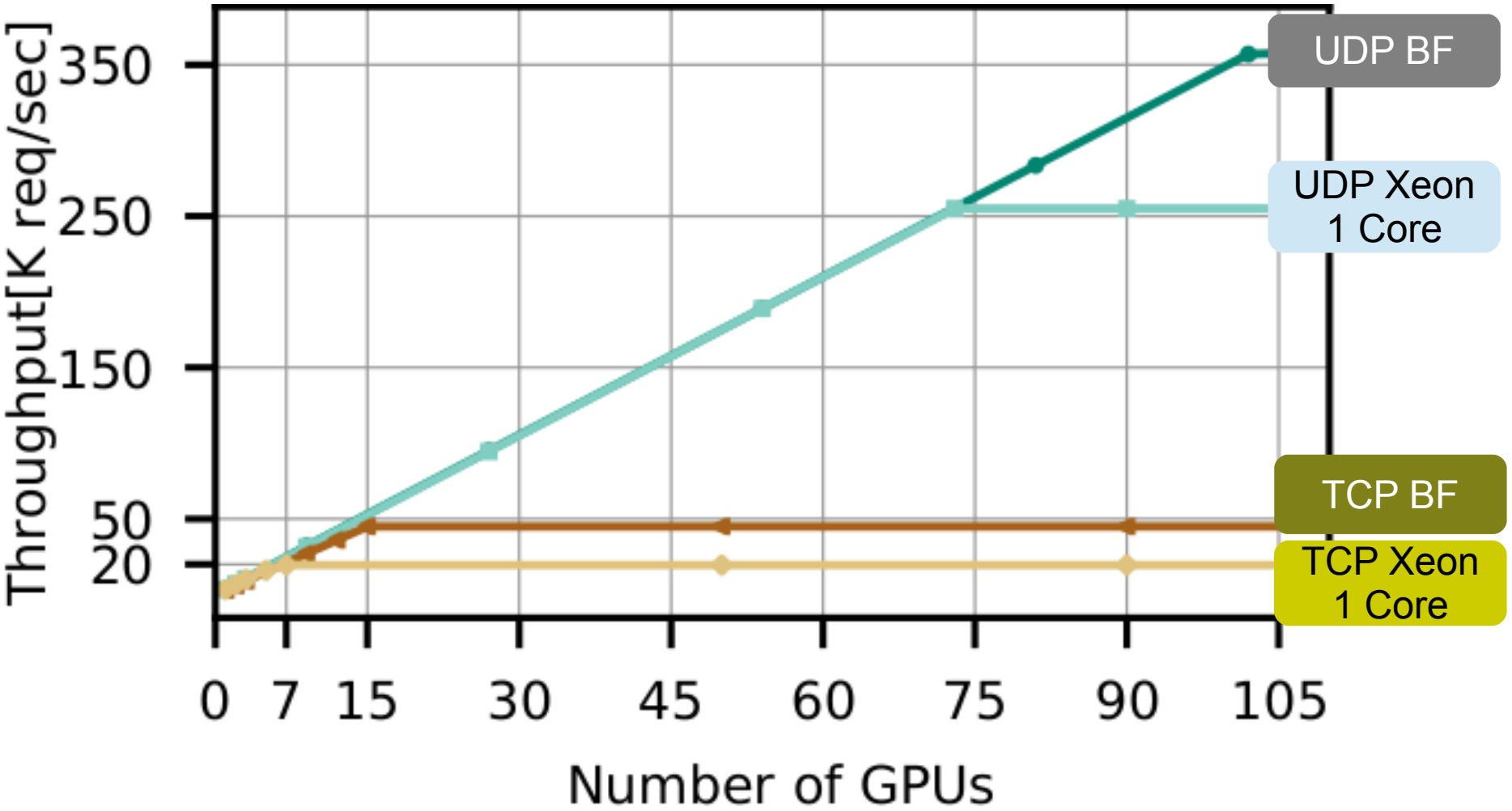
Latency=300us (25% lower than CPU-centric)
Thpt = 100% of theoretical max (GPU bottleneck)

Scale out to remote GPUs



Use of RDMA for queue management enables seamless scale out

Theoretical scalability for LeNet



What do GPUs need to enable GPU I/O V2.0?

- Proper PCIe consistency mechanisms
- Well-defined GPU memory management interface
- Hardware support for persistent/lightweight kernels
  - More efficient GPU-driven kernel invocation
  - Network-triggered kernel invocation (similar to cudaAsync)

Summary

- GPU-side I/O is effective and convenient
- Purely GPU-driven (GPU I/O V1.0) involves «fat» software stack on the GPU – costly
- GPU I/O V2.0 leverages new hardware to eliminate the software bloat

Same principles are useful for SGX [Eurosys17,USENIX ATC19] and disaggregated data centers [SFMA19, ongoing]

Code available @ https://github.com/acsl-technion

OmniX is an ongoing work in

Thank you!

mark@ee.technion.ac.il
https://sites.google.com/site/silbersteinmark

Haggai Eran, Amir Watad, Shai Bergman, Tanya Brokhman, Lior Zeno, Maroun Tork, Meni Orenbach, Lev Rosenblit, Alon Rashelbach, Pavel Lifshits, Gabi Malka, Lina Maudlej