Omnix: an accelerator-centric OS architecture for omni-programmable systems

Rethinking the role of CPUs in modern computers

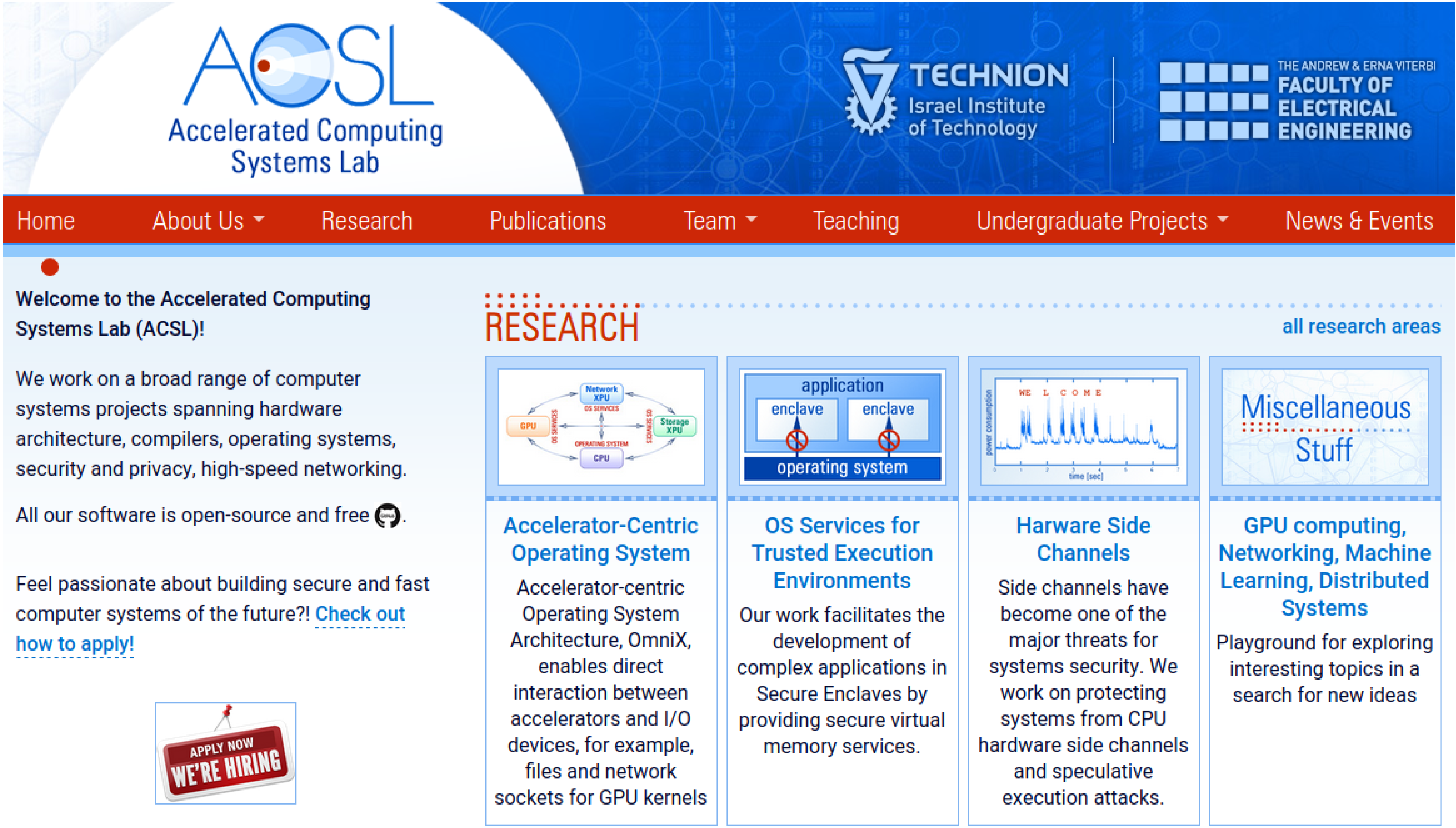
Mark Silberstein

Accelerated Computing Systems Lab

Technion

May 2021
ICL

https://acsl.group



ACSL
Accelerated Computing Systems Lab
TECHNION
Israel Institute of Technology
THE ANDREW & ERNA VITERBI
FACULTY OF ELECTRICAL ENGINEERING
Home
About Us
Research
Publications
Team
Teaching
Undergraduate Projects
News & Events

Welcome to the Accelerated Computing Systems Lab (ACSL)!

We work on a broad range of computer systems projects spanning hardware architecture, compilers, operating systems, security and privacy, high-speed networking.

All our software is open-source and free.

Feel passionate about building secure and fast computer systems of the future?! Check out how to apply!

RESEARCH

all research areas

Accelerator-Centric Operating System

Accelerator-centric Operating System Architecture, OmniX, enables direct interaction between accelerators and I/O devices, for example, files and network sockets for GPU kernels

OS Services for Trusted Execution Environments

Our work facilitates the development of complex applications in Secure Enclaves by providing secure virtual memory services.

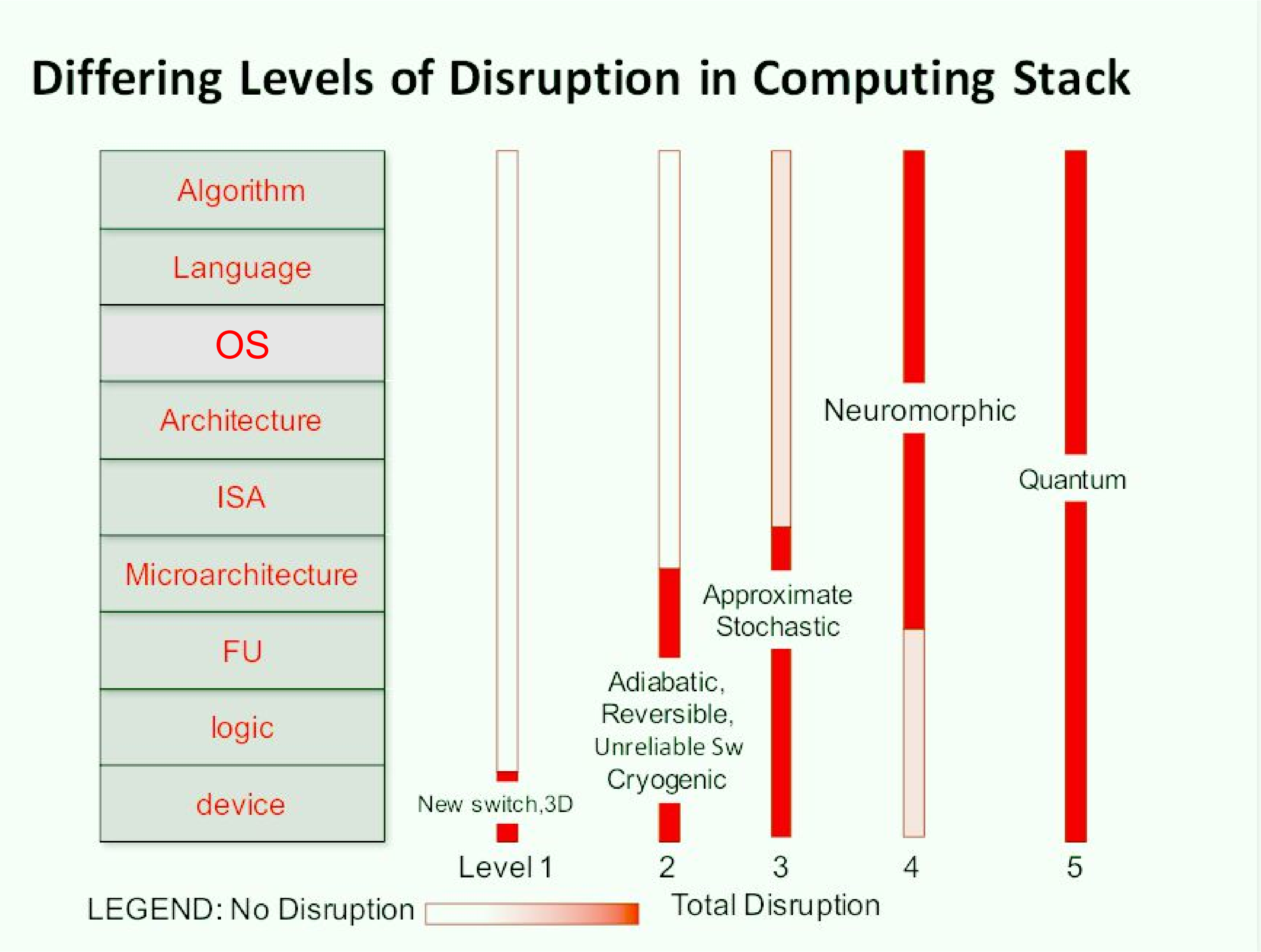
Harware Side Channels

Side channels have become one of the major threats for systems security. We work on protecting systems from CPU hardware side channels and speculative execution attacks.

GPU computing, Networking, Machine Learning, Distributed Systems

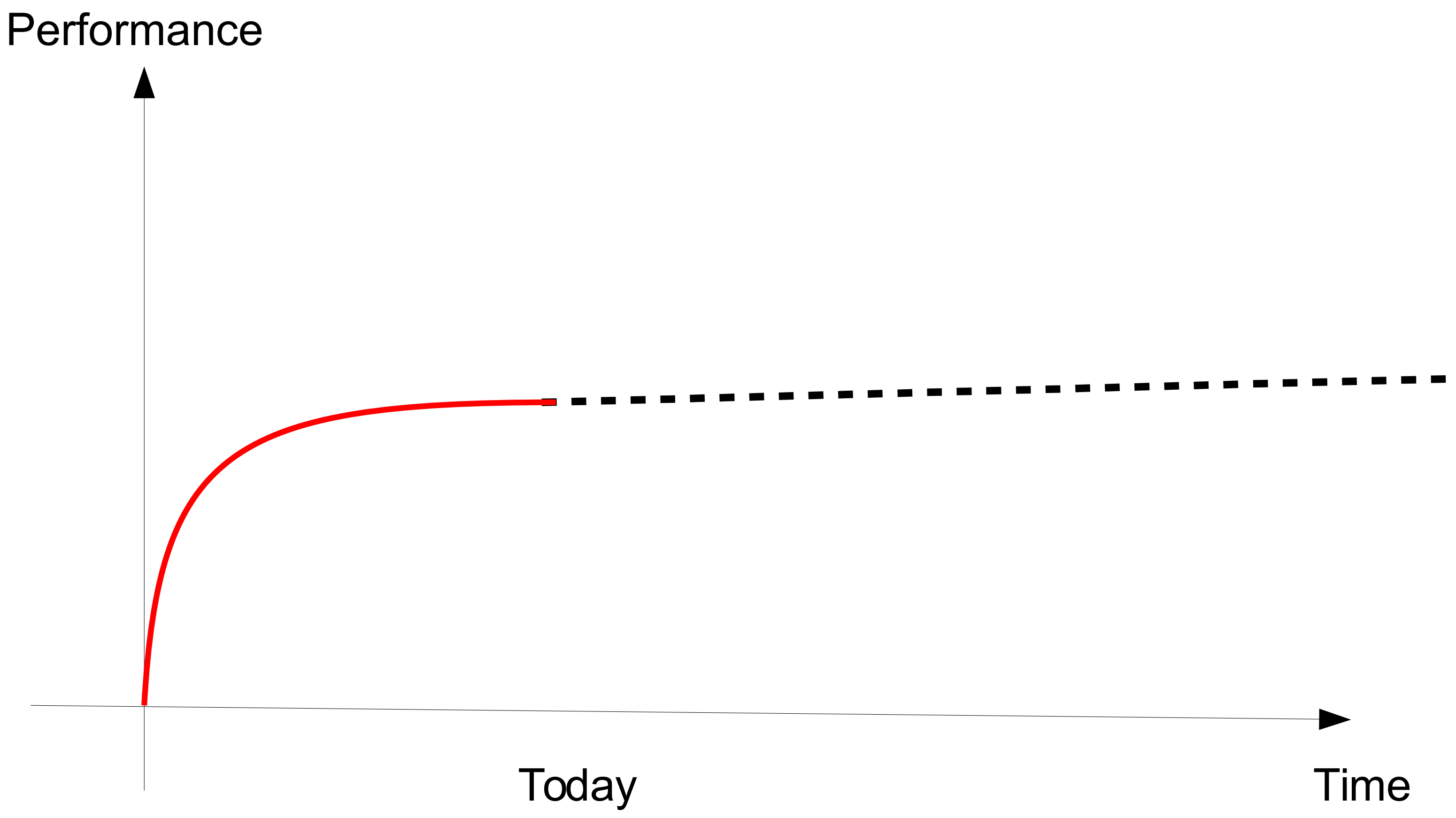
Playground for exploring interesting topics in a search for new ideas

Beyond CMOS: total disruption!

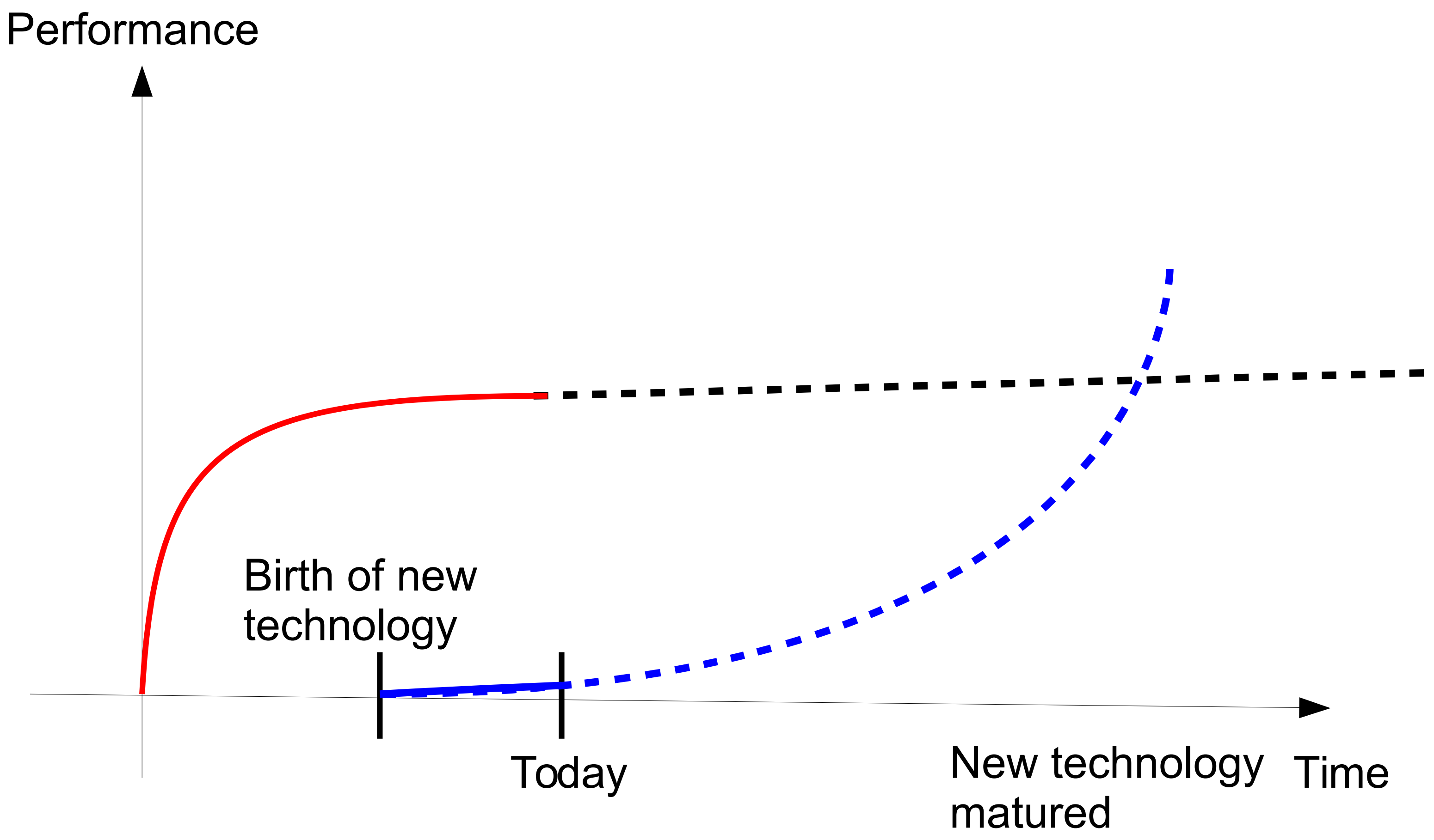


From «IEEE rebooting computing»

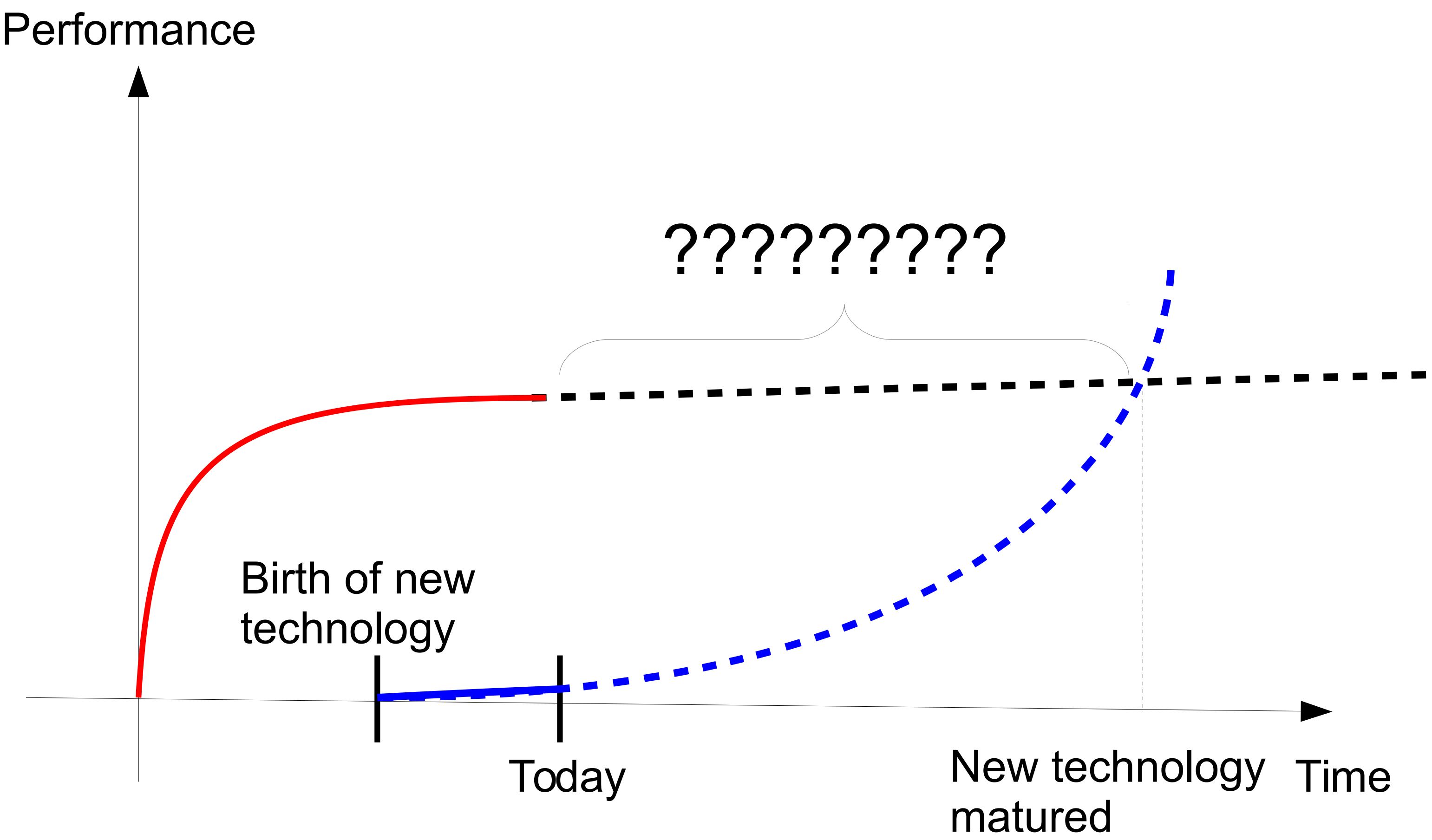
Stagnation of the current processing technology



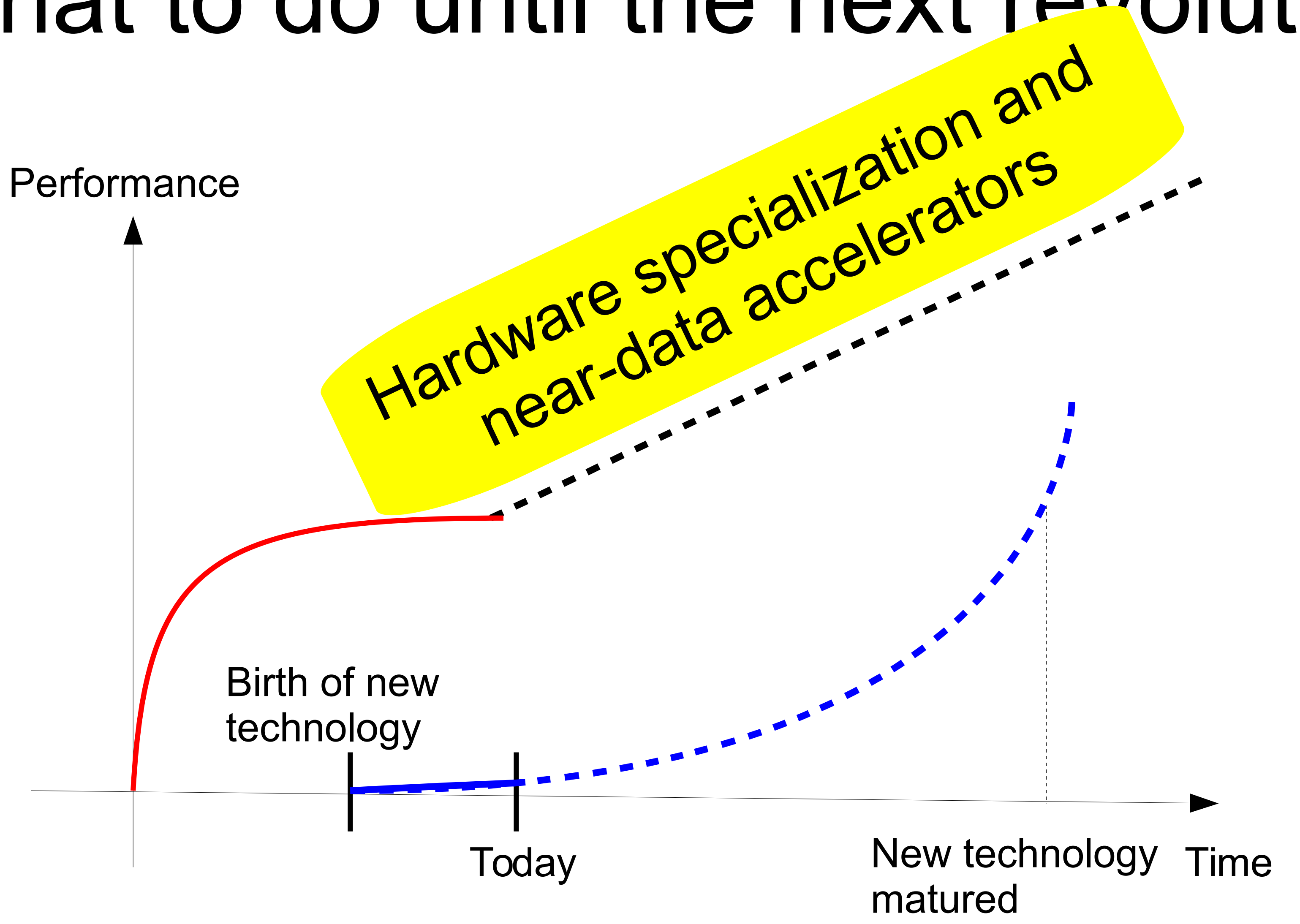
Next generation is coming soon...



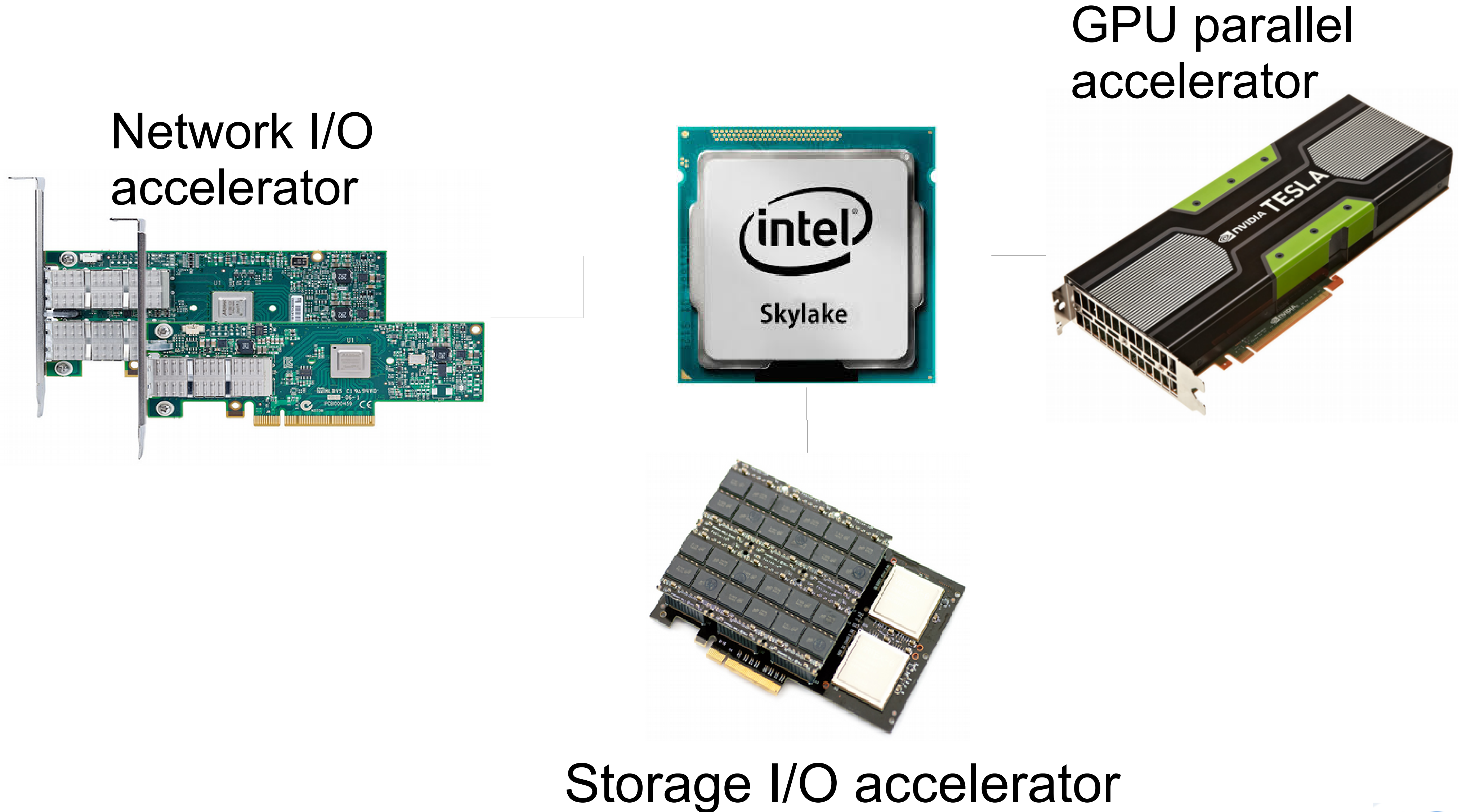
What to do until the next revolution?



What to do until the next revolution?



Computer hardware: circa ~2021

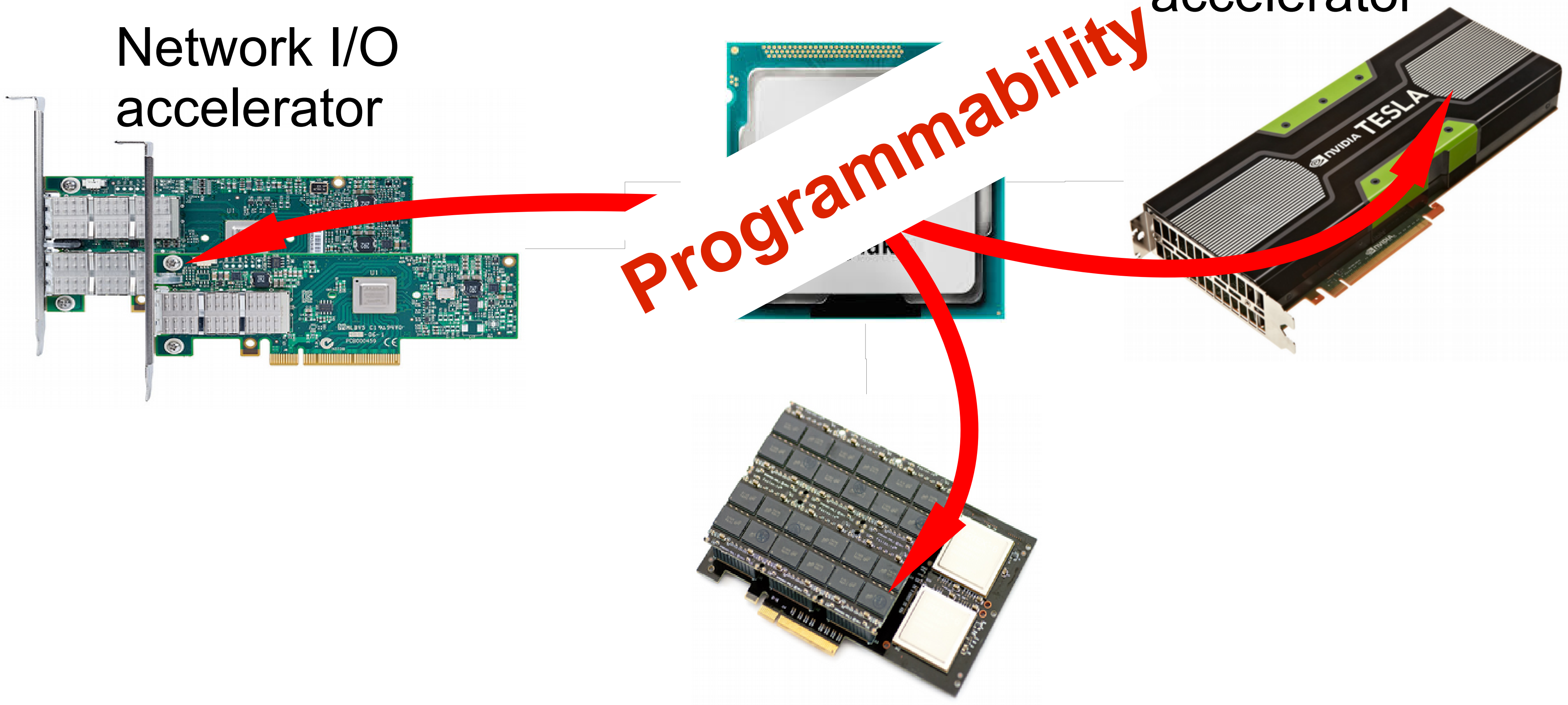


Network I/O accelerator

GPU parallel accelerator

Storage I/O accelerator

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



Network I/O accelerator

GPU parallel accelerator

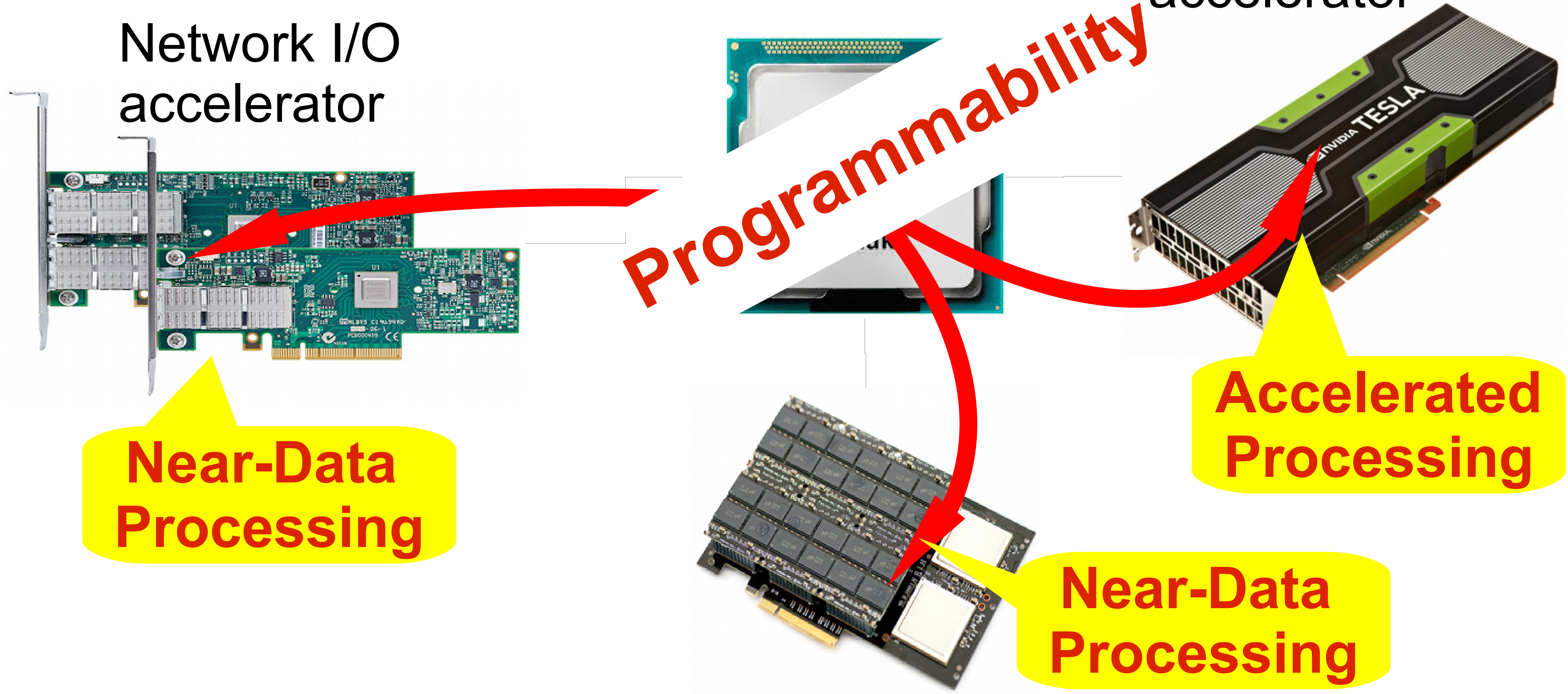
Programmability

Storage I/O accelerator

Omni-programmable system
X- Processing Units: xPUs

Network I/O accelerator

GPU parallel accelerator



Programmability

Near-Data Processing

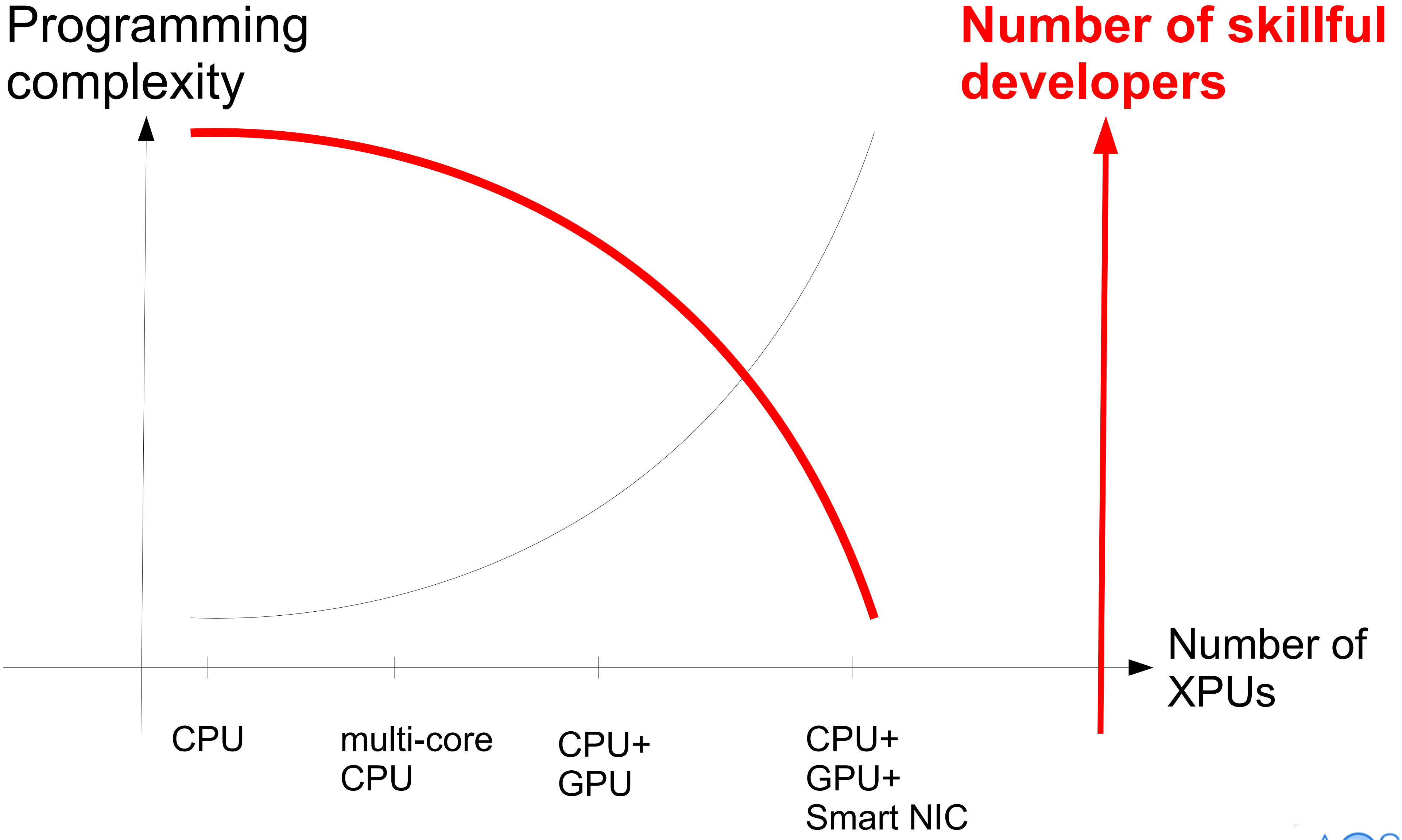
Accelerated Processing

Near-Data Processing

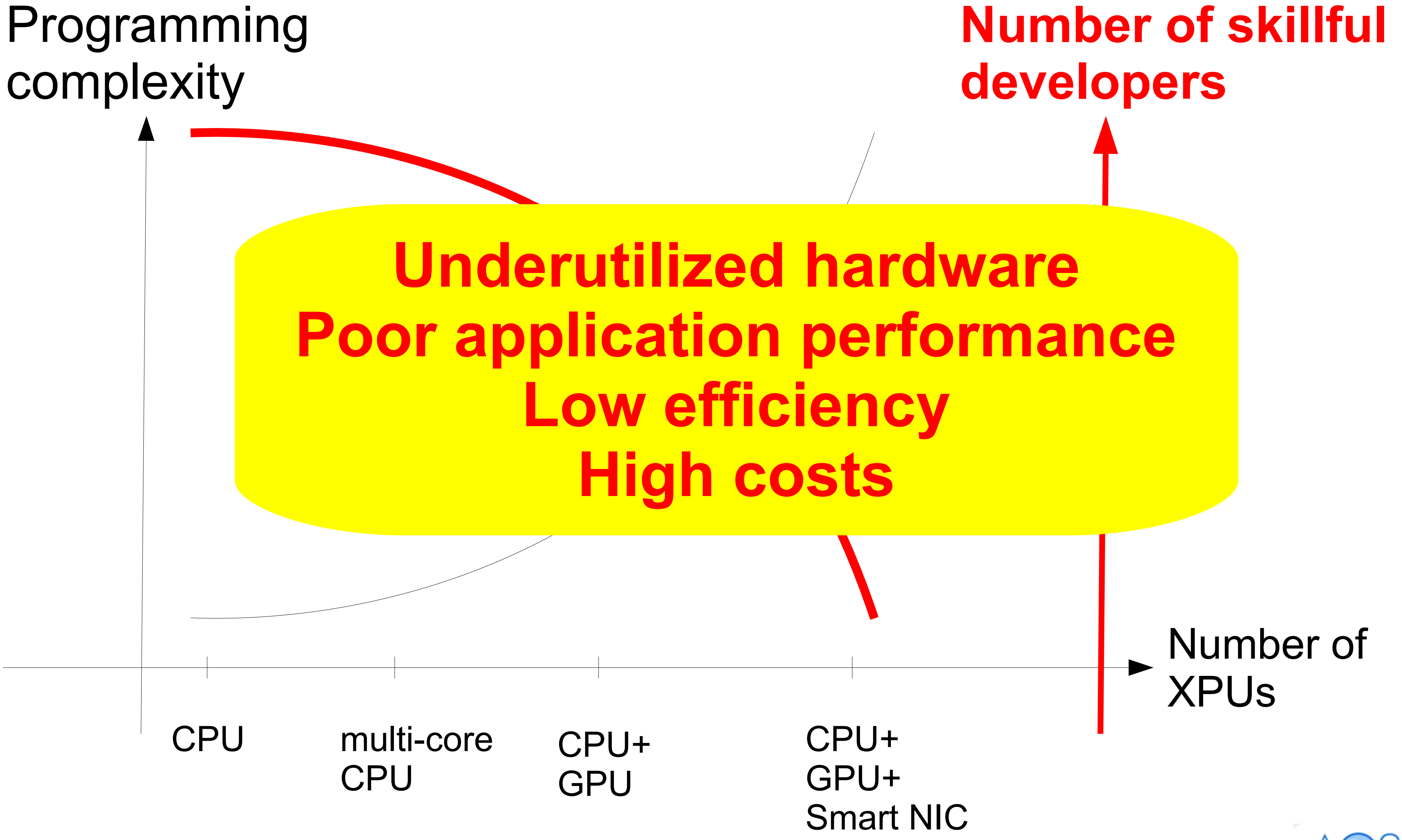
Storage I/O accelerator

But XPUs create...

Hard to maintain whole-application efficiency



Hard to maintain whole-application efficiency



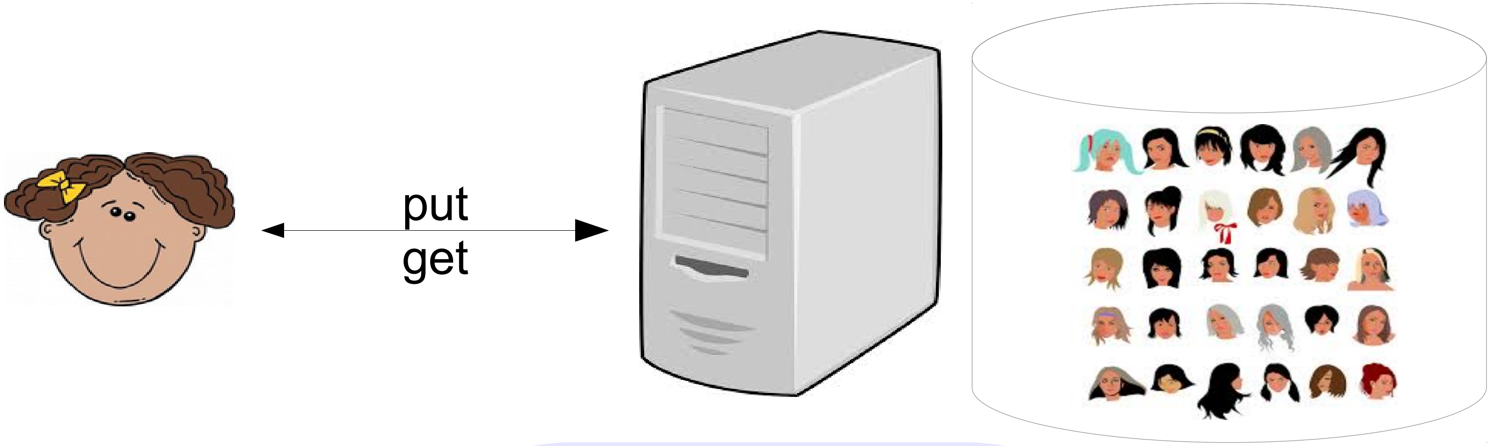
Underutilized hardware
Poor application performance
Low efficiency
High costs

Agenda

- The root cause of the programmability wall
- OmniX: accelerator-centric OS design
  - Principles
  - Examples
- Future-proof: OmniX and disaggregated systems

Example: image server

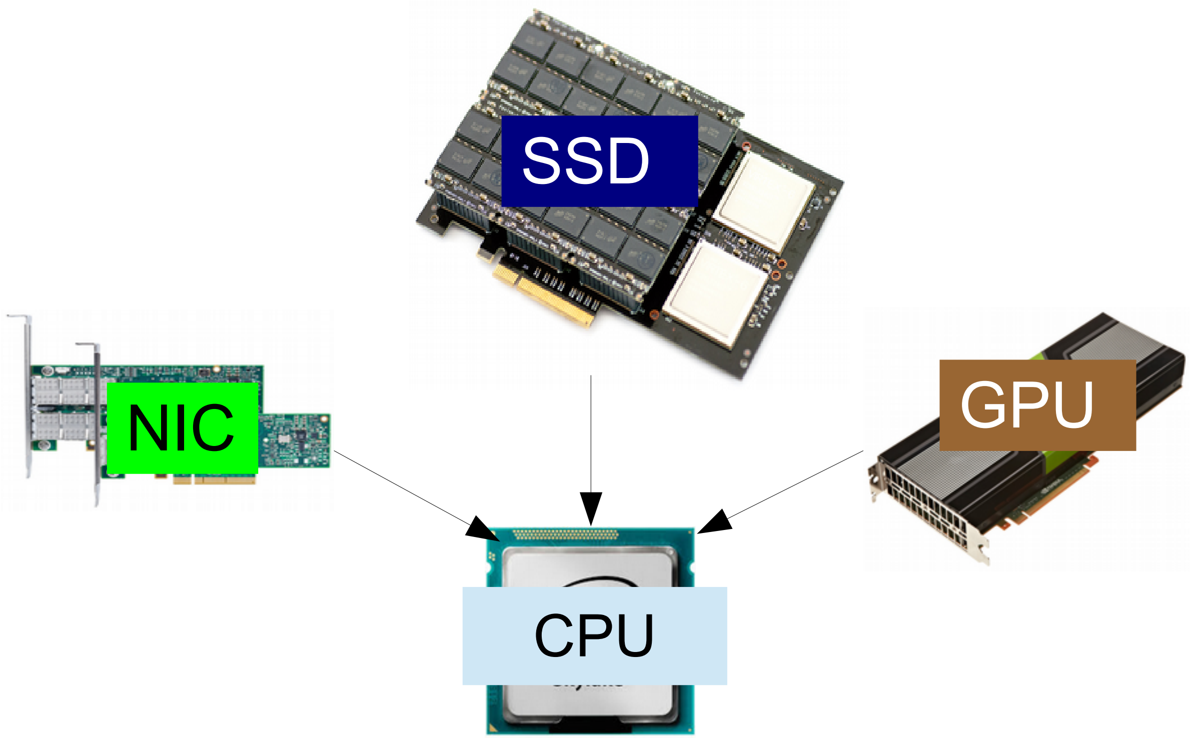
1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal



Similar architecture used in Flickr

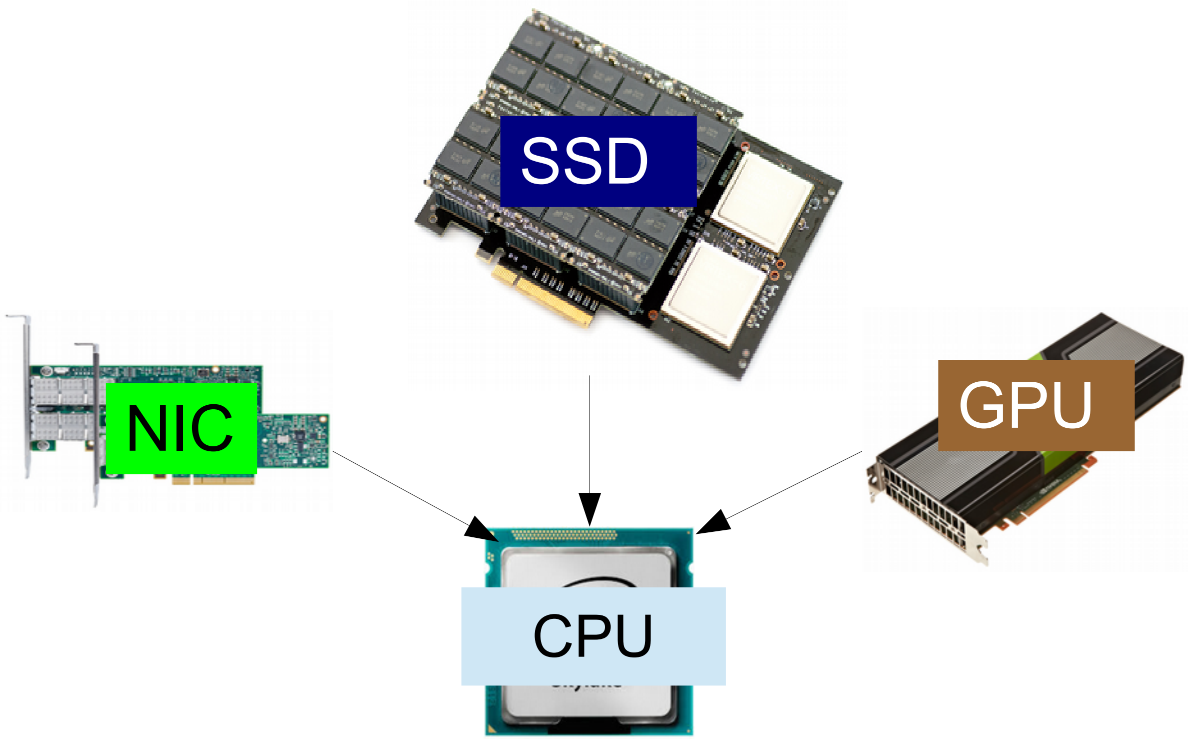
Example: image server

1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal



Accelerating with XPUs

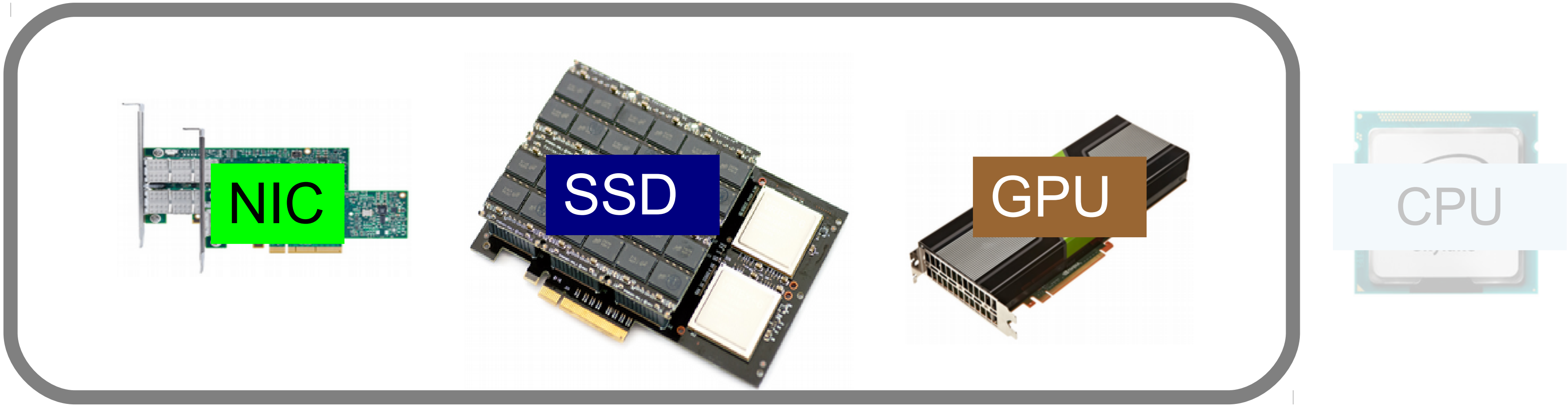
1. put: parse → **contrast-enhance** → **store**
2. get: parse → **resize** → **store** → marshal



May 2021

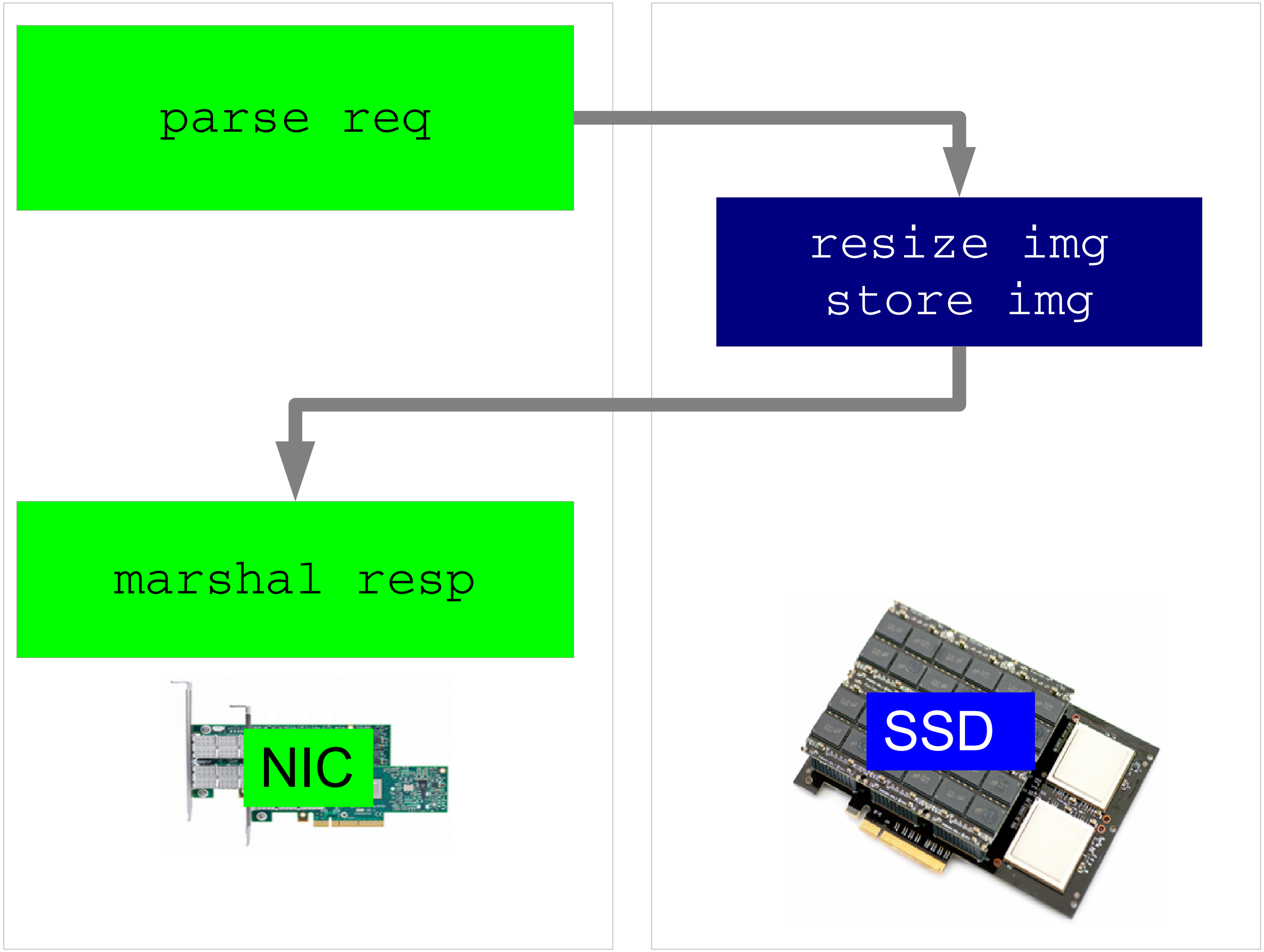
Accelerating with XPUs

1. put: parse → **contrast-enhance** → **store**
2. get: parse → **resize** → **store** → marshal



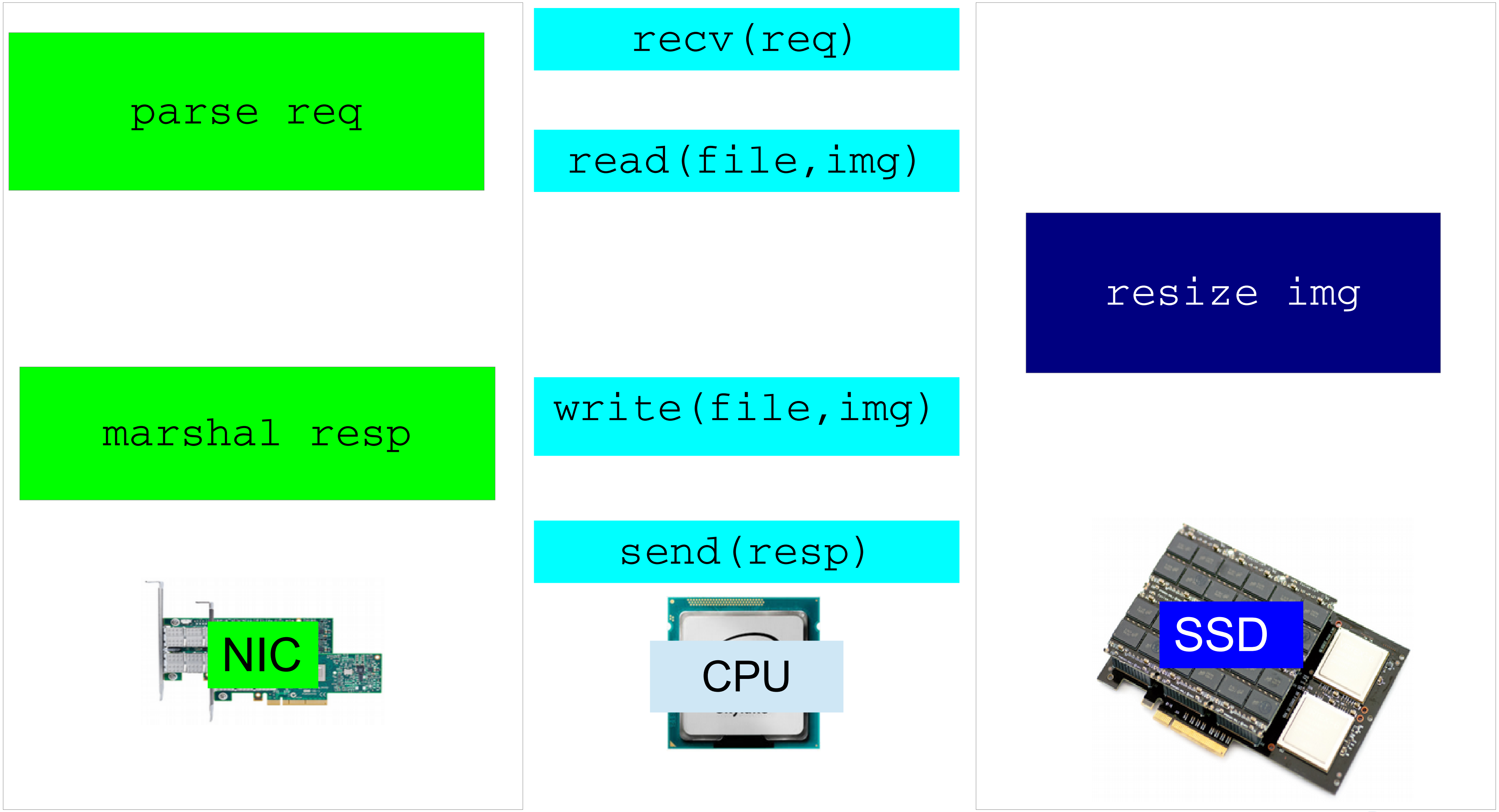
Closer look at *`get`*

parse → **resize** → **store** → marshal



OS services run on CPUs

get: parse → **resize** → **store** → marshal



parse req

marshal resp

recv(req)

read(file,img)

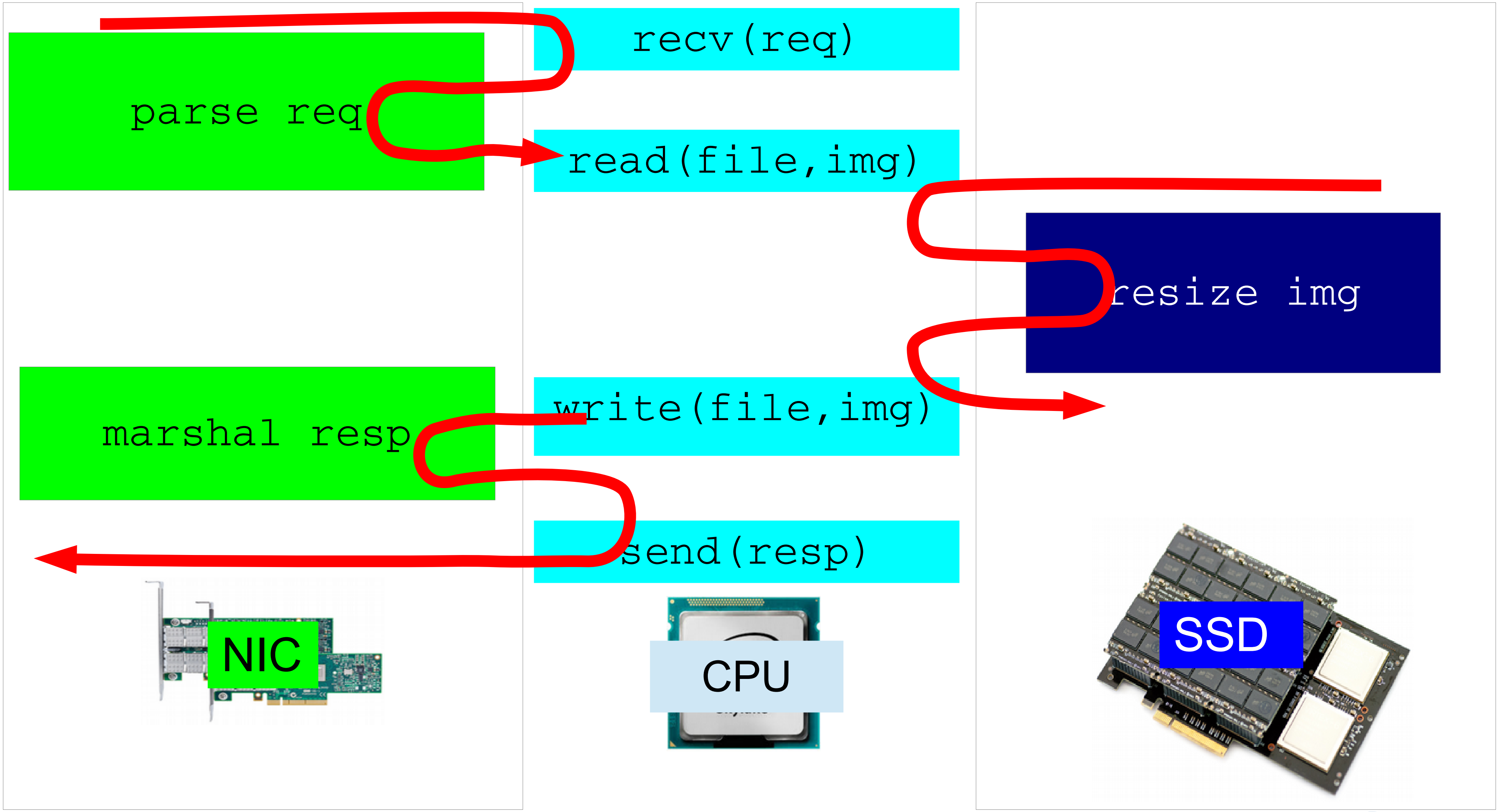
write(file,img)

send(resp)

resize img

Result: offloading overheads dominate

get: parse → **resize** → **store** → marshal



parse req

recv(req)

read(file,img)

resize img

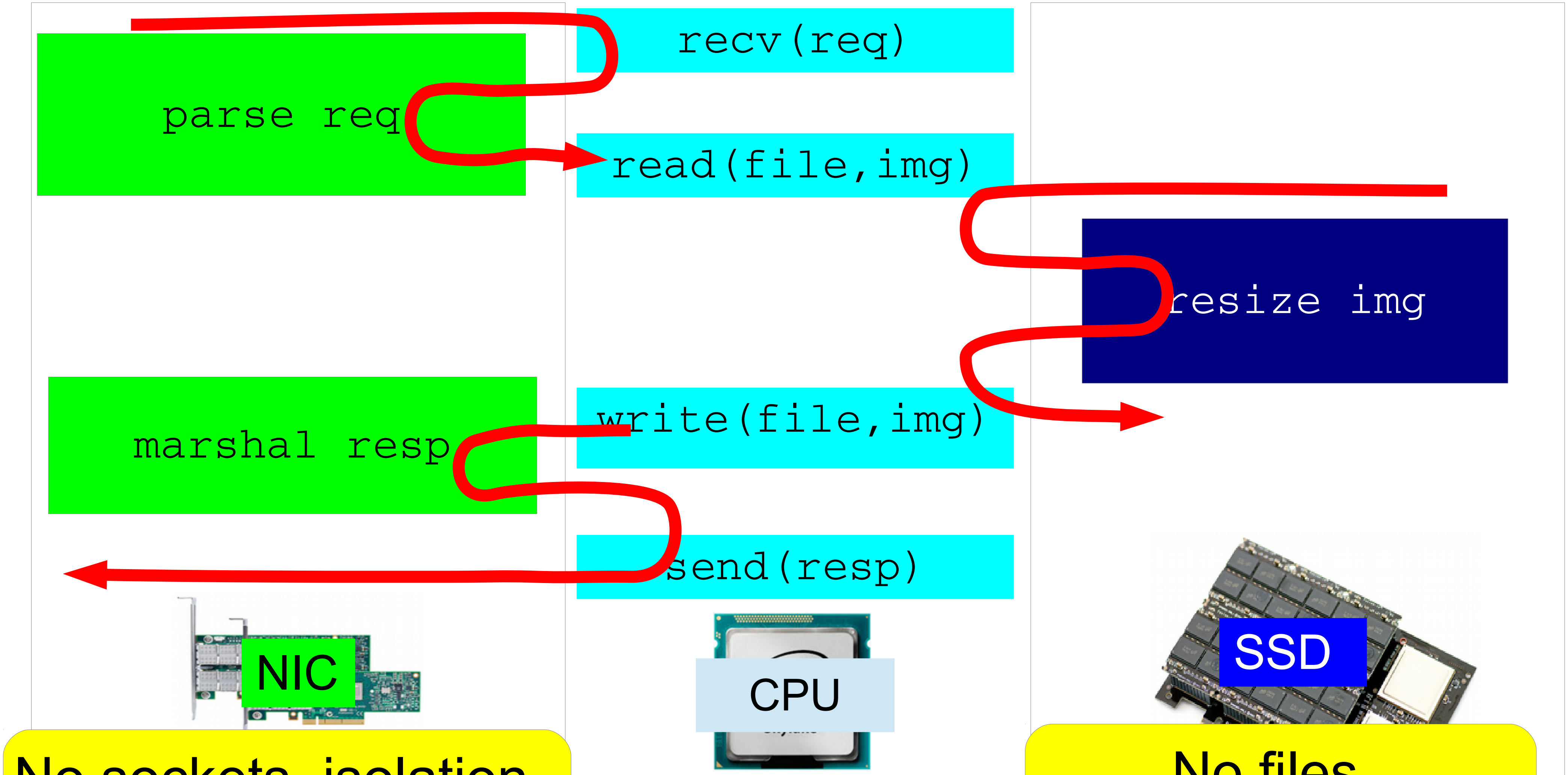
marshal resp

write(file,img)

send(resp)

Result: offloading overheads dominate

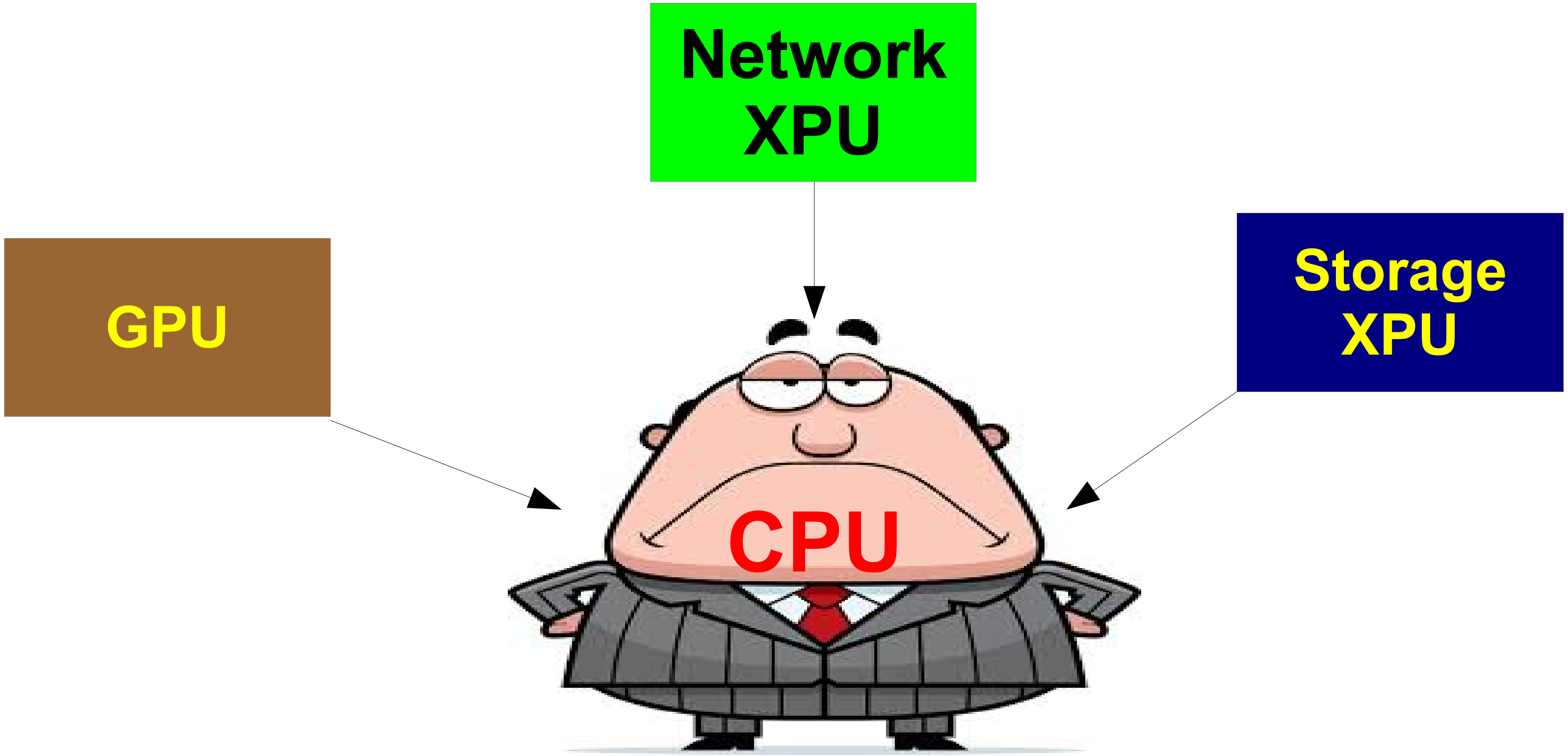
get: parse → **resize** → **store** → marshal



No sockets, isolation, transport layer …

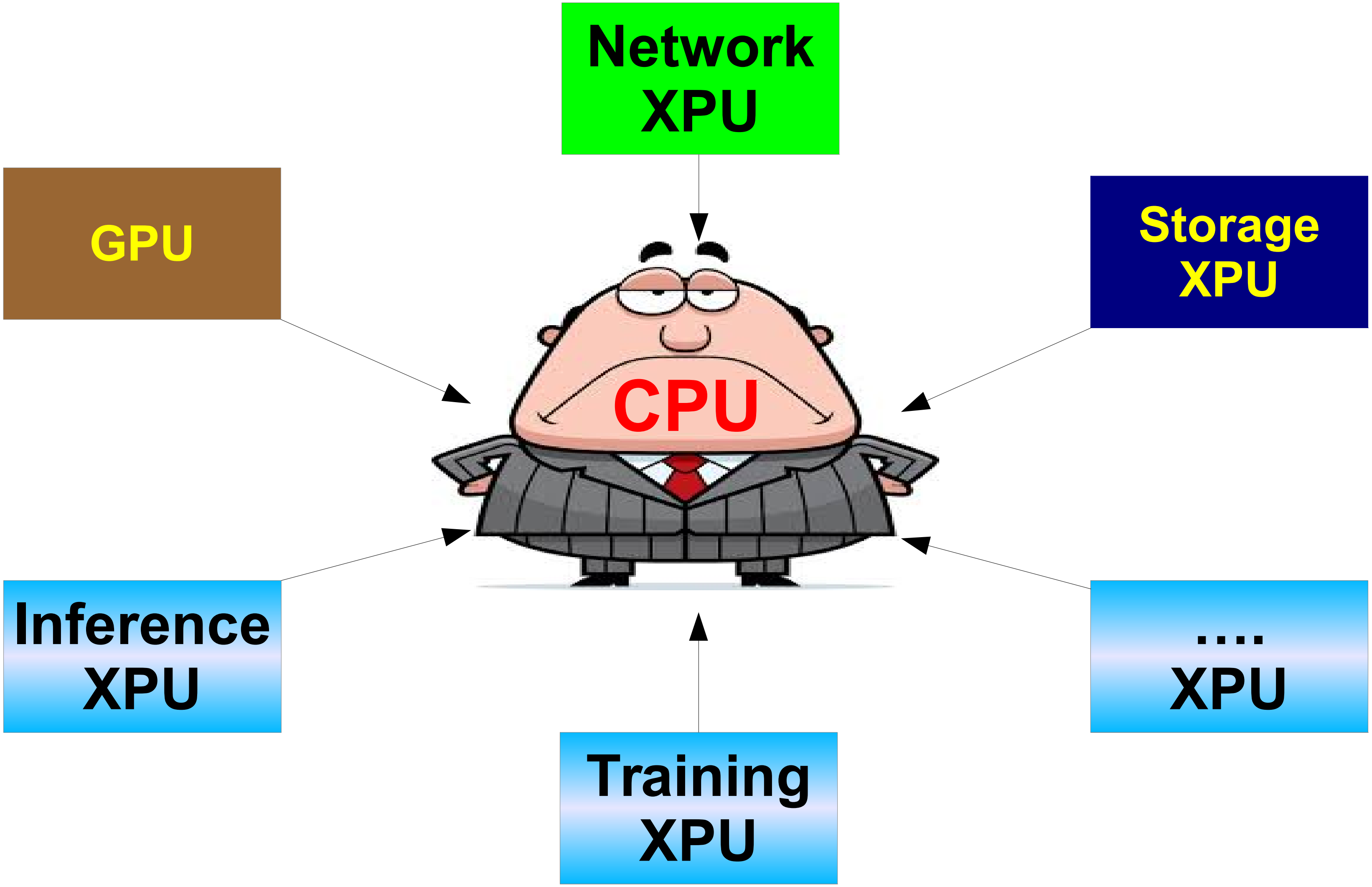
No files, protection...

THE problem:
OS architecture is CPU - centric



May 2021

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



Network XPU

GPU

Storage XPU

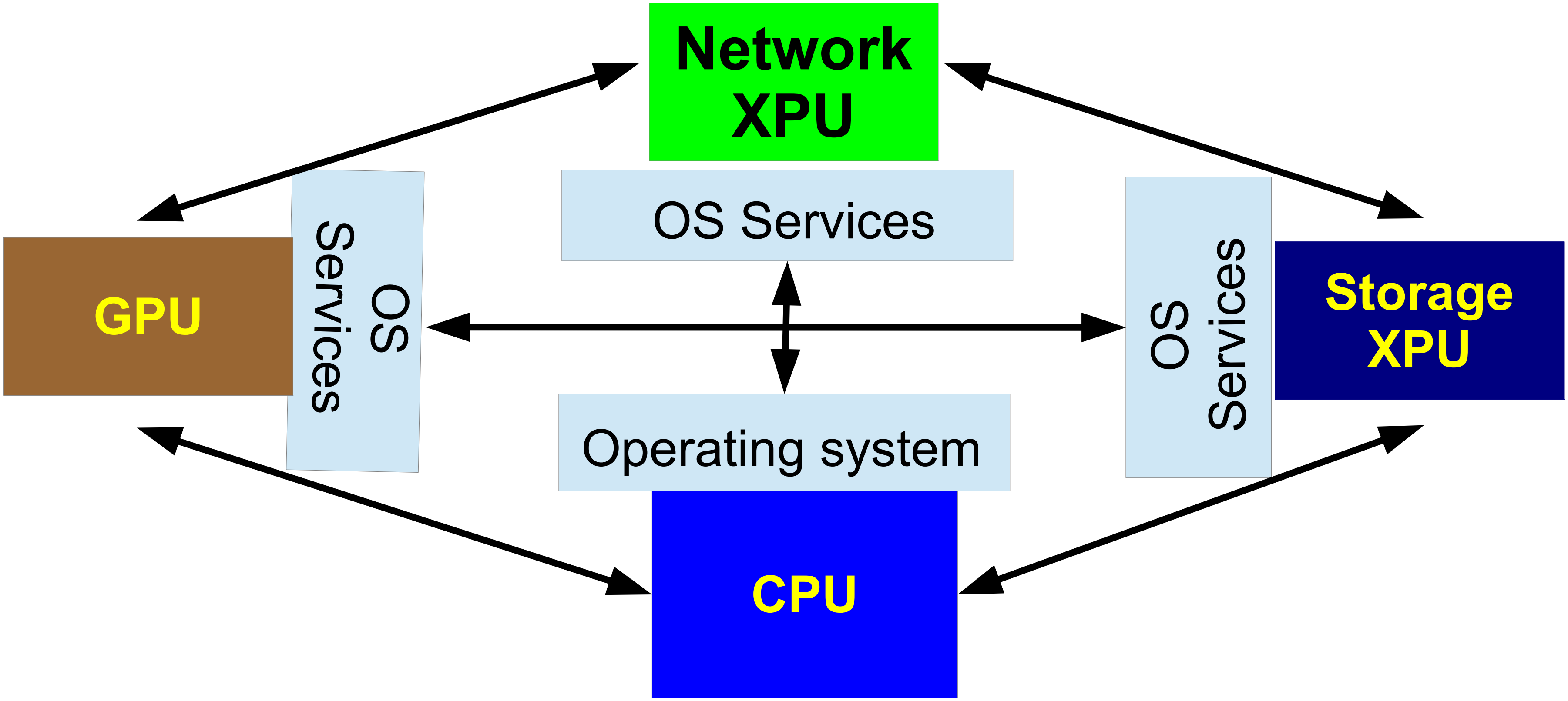
CPU

Inference XPU

…. XPU

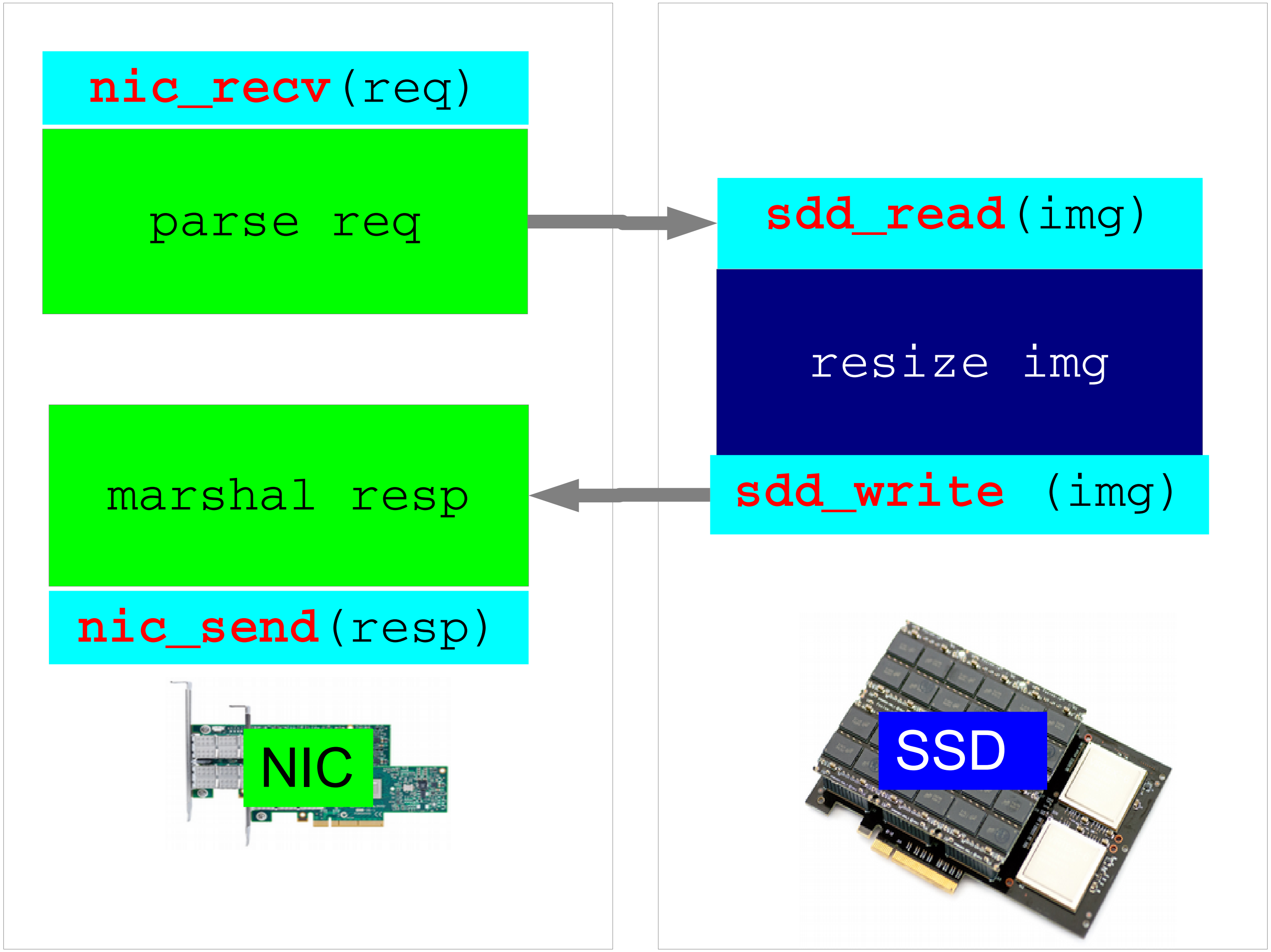
Training XPU

OmniX: accelerator-centric OS architecture



Execution in OmniX

get: parse → **resize** → **store** → marshal



nic_recv(req)

parse req

sdd_read(img)

resize img

sdd_write (img)

marshal resp

nic_send(resp)

Accelerator-centric OS architecture

Types of OS abstractions for accelerators

Accelerator-centric: *on-accelerator* services

Accelerator-friendly: *accelerator-aware* host OS changes

Data-centric: CPU-less inline near-data processing

Types of OS abstractions for accelerators

Accelerator-centric: *on-accelerator* services

- Networking: GPUnet, GPUrdma, Centaur, LYNX
- Files: GPUfs, ActivePointers

Accelerator-friendly: *accelerator-aware* host OS changes

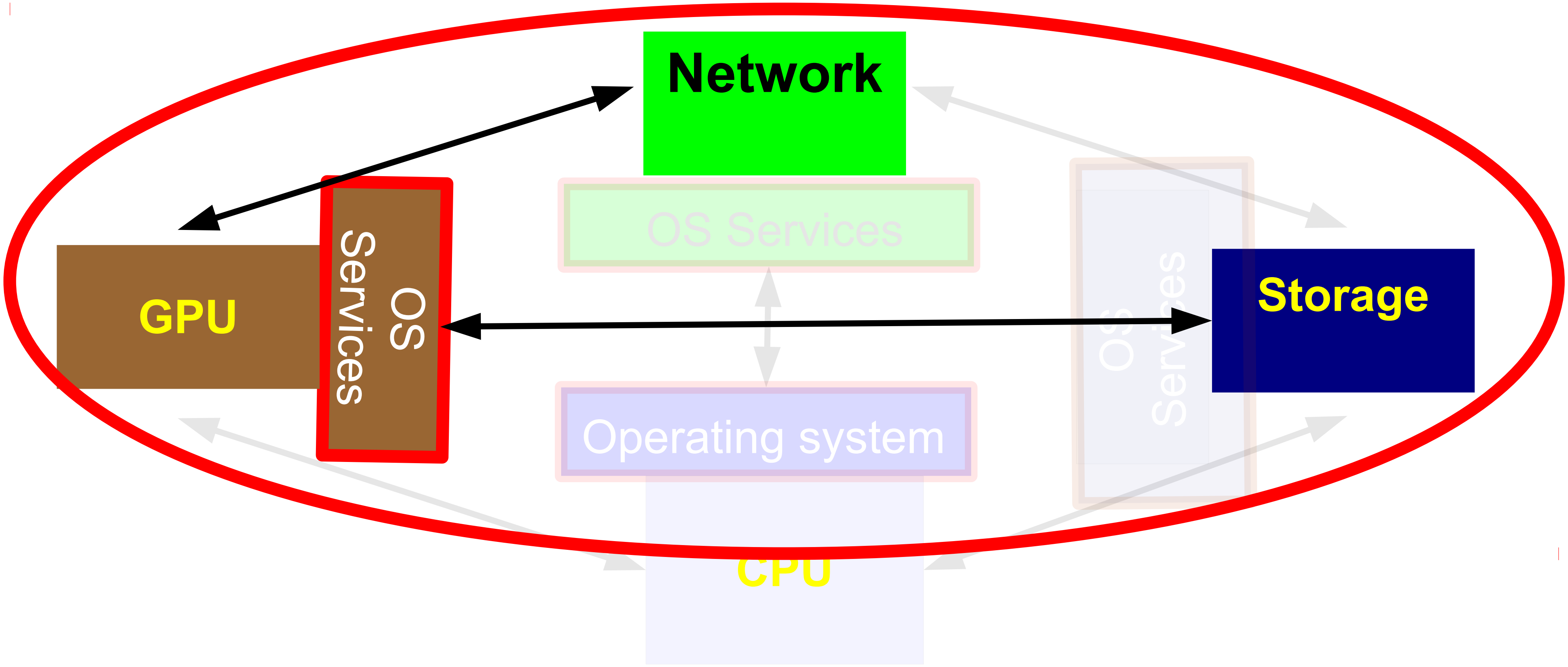
- SPIN, GAIA – host-accelerator file sharing

Data-centric: CPU-less inline near-data processing

- NICA – Server acceleration on FPGA-based SmartNICs

ASPLOS13,TOCS14,OSDI14,TOCS15,ISCA16,SYSTOR16, ROSS16, ATC17,HotOS17,ATC19, ATC19-2, TOCS19, PACT19, ASPLOS20

On-GPU I/O services

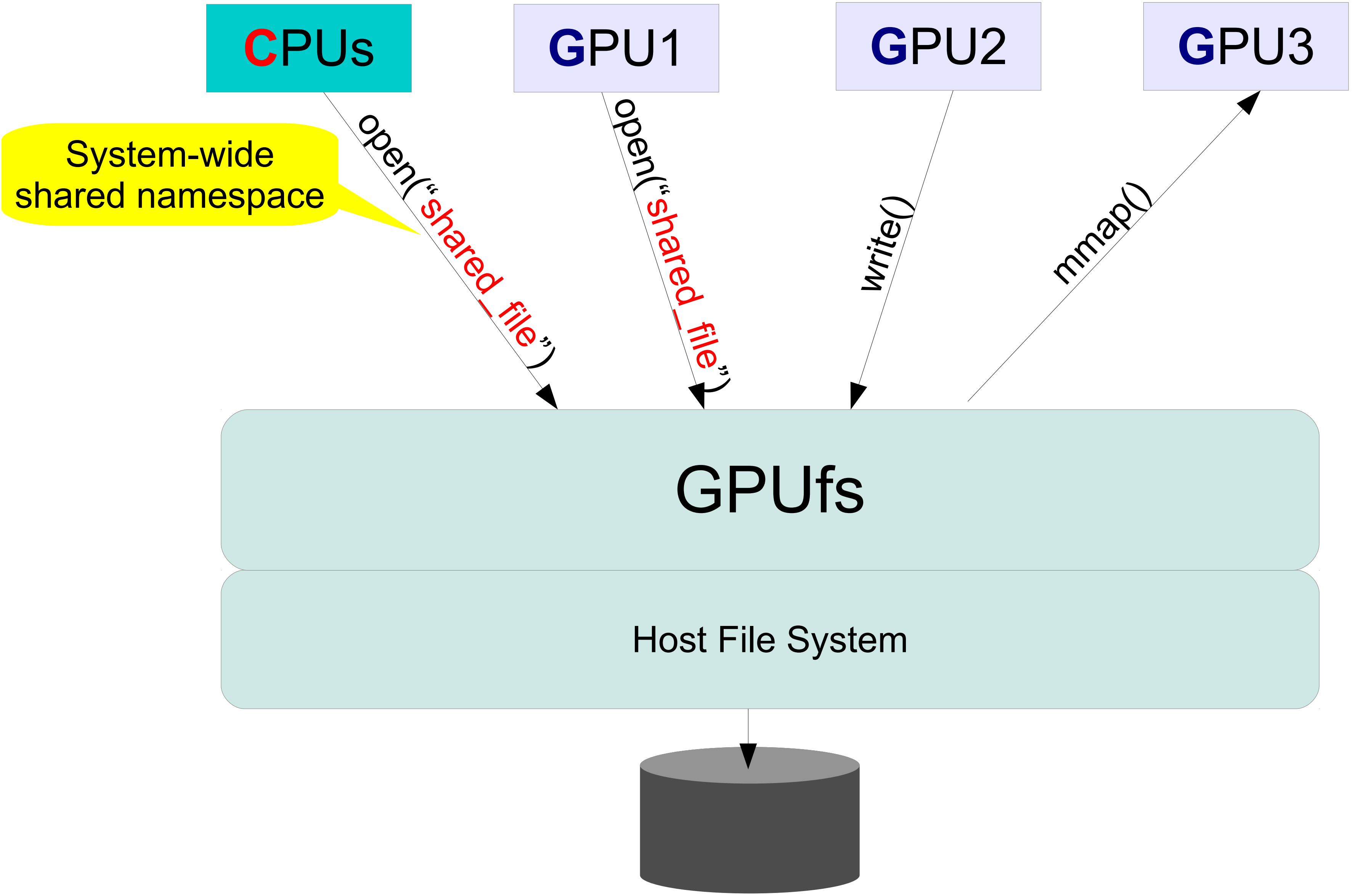


May 2021

Accelerated Computing Systems Lab

GPUfs: File system library for GPUs

ASPLOS13: S., Keidar, Ford, Witchel



CPUs

GPU1

GPU2

GPU3

System-wide shared namespace

open("shared_file")

open("shared_file")

write()

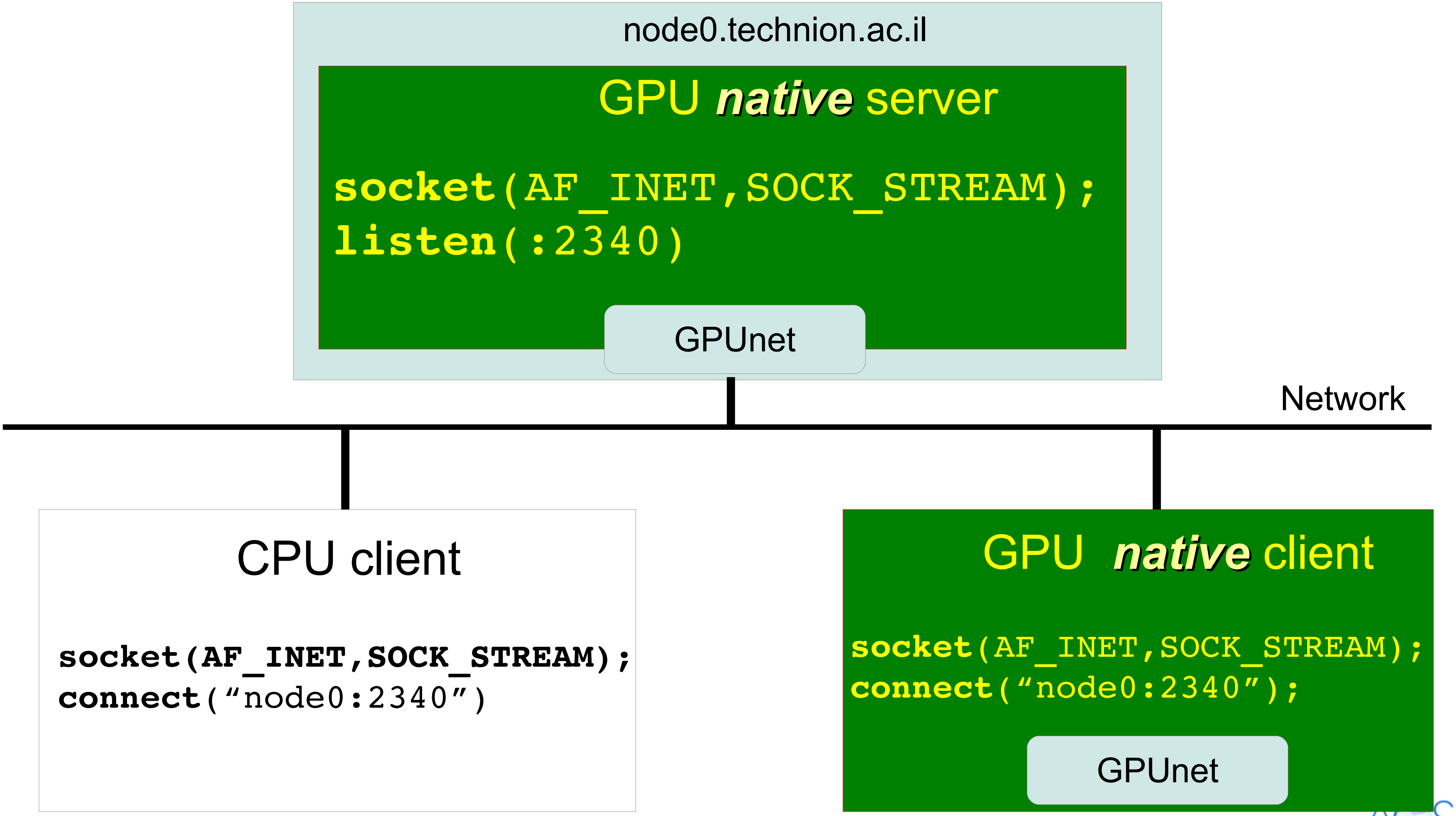
mmap()

GPUfs

Host File System

GPUnet: Network library for GPUs

OSDI14, S, Kim, Witchel



node0.technion.ac.il

GPU ***native*** server

```
socket(AF_INET,SOCK_STREAM);
listen(:2340)
```

GPUnet

Network

CPU client

```
socket(AF_INET,SOCK_STREAM);
connect(“node0:2340”)
```

GPU ***native*** client

```
socket(AF_INET,SOCK_STREAM);
connect(“node0:2340”);
```

GPUnet

Accelerator in full control over its I/O

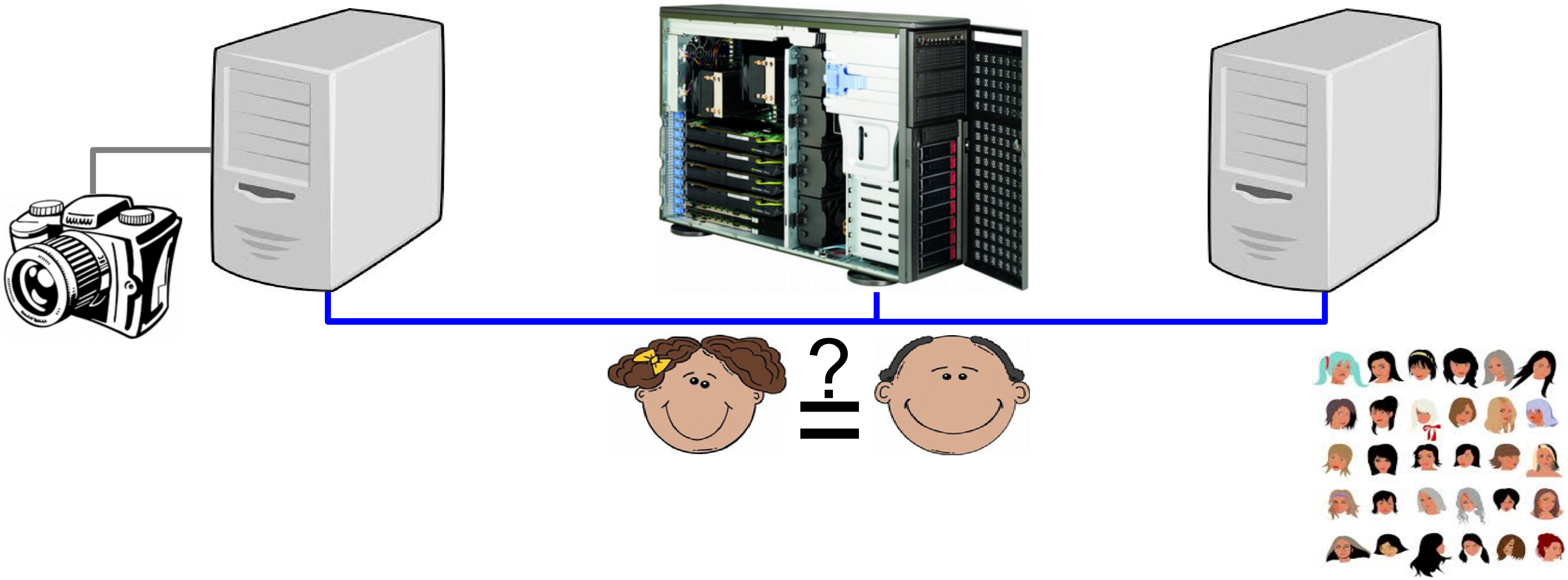
- I/O without «leaving» the GPU kernel
    - Data-driven access to huge DBs
    - Full-blown multi-tier GPU network servers
    - Multi-GPU Map/Reduce (no user CPU code)
- POSIX-like APIs with slightly modified semantics
- Transparency for the rest of the system
- Reduced code complexity
- Unleashed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

Example: face verification server

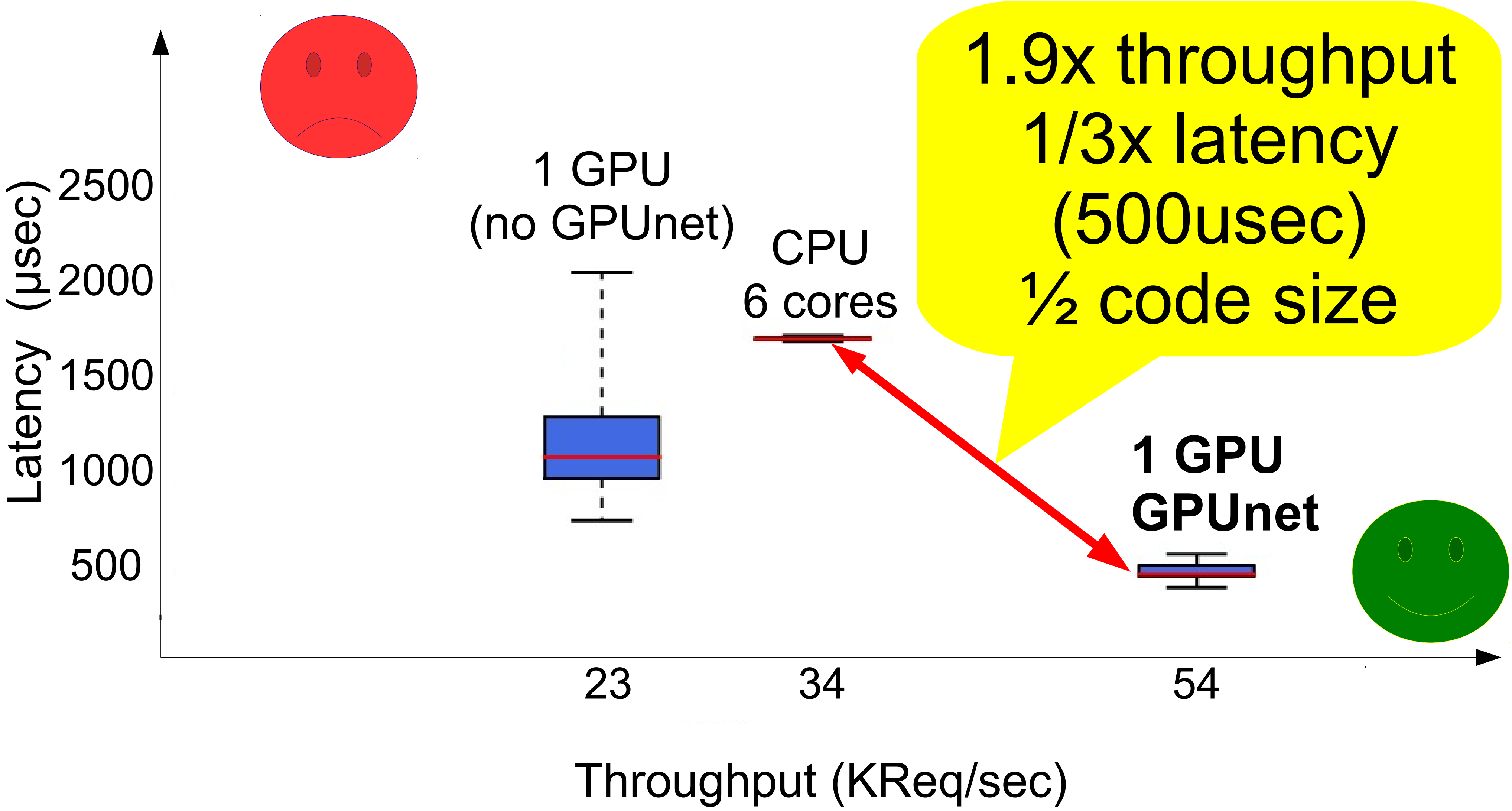
CPU client
(unmodified)

GPU server
(GPUnet)

memcached
(unmodified)



Face verification:
Different implementations



1.9x throughput
1/3x latency
(500usec)
½ code size

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

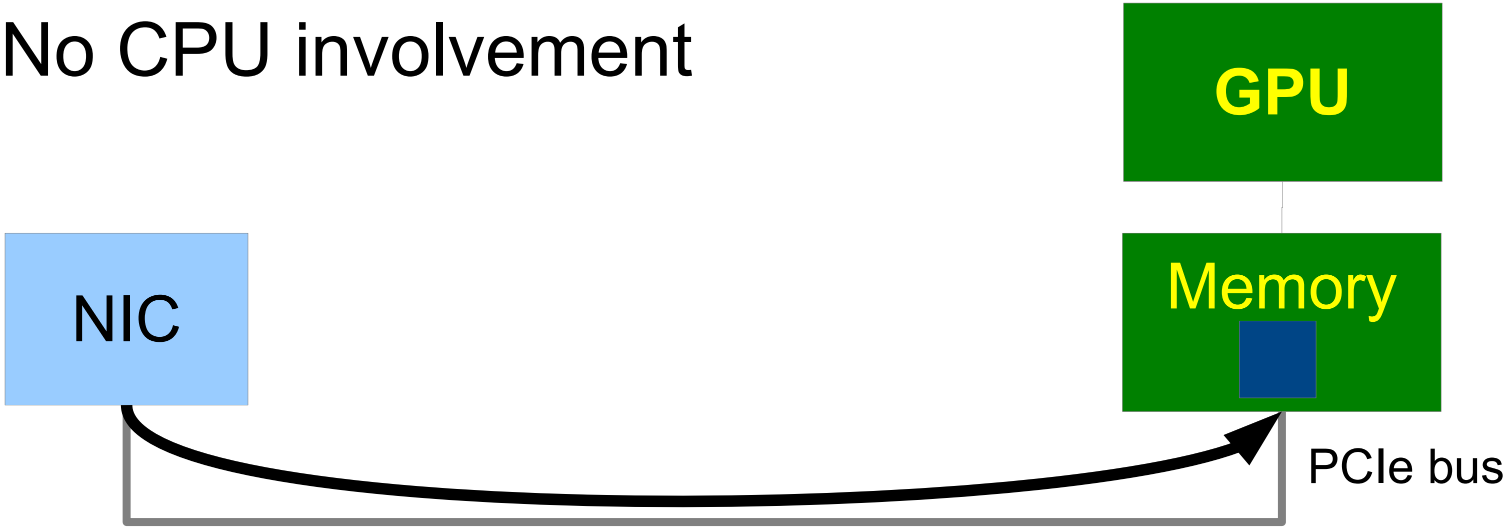
- Micro-kernel design
  - RPC to File/Network services on the CPU
  - User-land abstraction implementation (libOSes)
- Single name space with the CPU OS
  - Same socket space, same file name space
- Extensive SW layer on the GPU
  - Handles massive API parallelism
  - Implements consistency model (FS)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

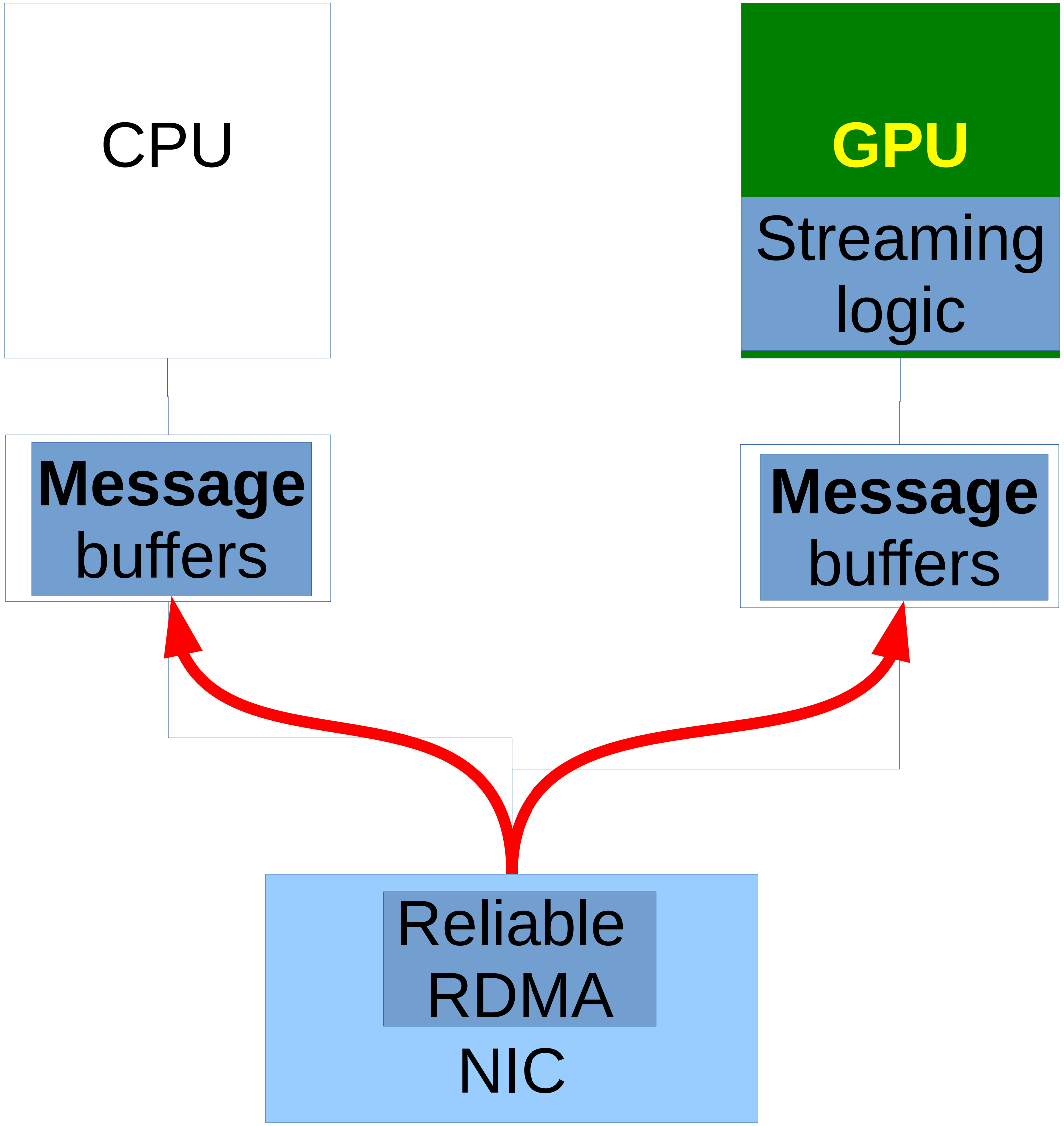
Challenge: the OS is on the CPU!

I/O device sharing, multiplexing, transport layer

Examples:

- GPU and CPU both need to access the network
- TCP on GPU?

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



Summary so far...

- Accelerator-centric OS services
  - Simplify code development
  - Enable transparent performance optimization
- But what if we **cannot add code** to an accelerator?
  - Accelerators are inefficient when running OS logic
  - Some systems use close-source accelerated libs

Summary so far...

- Accelerator-centric OS services
  - Simplify code development
  - Enable transparent performance optimization
- But what if we **cannot add code** to an accelerator?
  - Accelerators are inefficient when running OS logic
  - Some systems use close-source accelerated libs

Make host OS accelerator-aware

Storage: OS integration of P2P DMA between SSD and GPUs

SPIN: USENIX ATC17, partially adopted by NVIDIA

- Accelerator-aware modification to host FS API
- Allows using **GPU** memory buffers in `read/write`
  - Transparently selects page cache or P2P DMA
  - Maintains POSIX FS consistency
  - Integrates with OS prefetcher
  - Compatible with OS block layer (i.e., software RAID)
- Results:
  - 5.2GB/s from SSDs to GPU
  - 2-3x speedup in applications

Storage: Extending CPU page cache into GPU memory

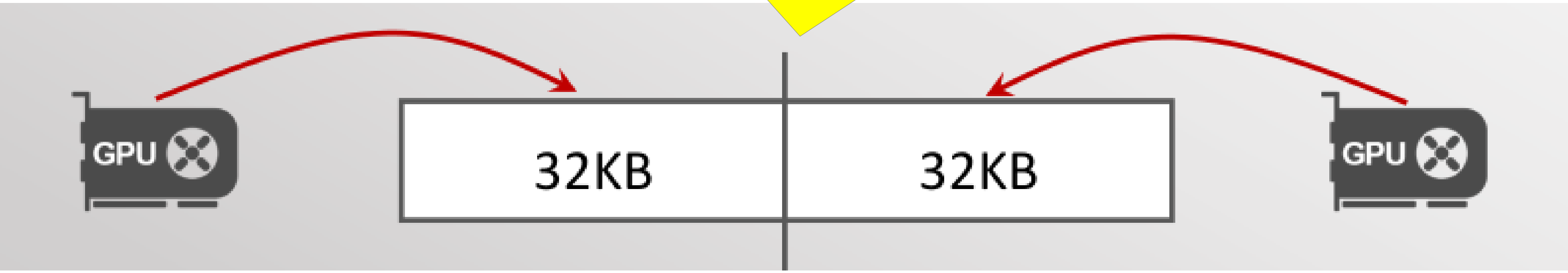
GAIA: USENIX ATC19

- Accelerator-aware modification to host page cache to use GPU page faults
- Enables `mmap` for GPU
- Enables CPU-GPU file sharing
- May cache/prefetch file data in GPU memory
- Insights:
  - Slim GPU driver API for enabling host page cache integration
  - Page cache release consistency model **for high performance**
  - OS page cache and Linux kernel modifications for consistency support

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

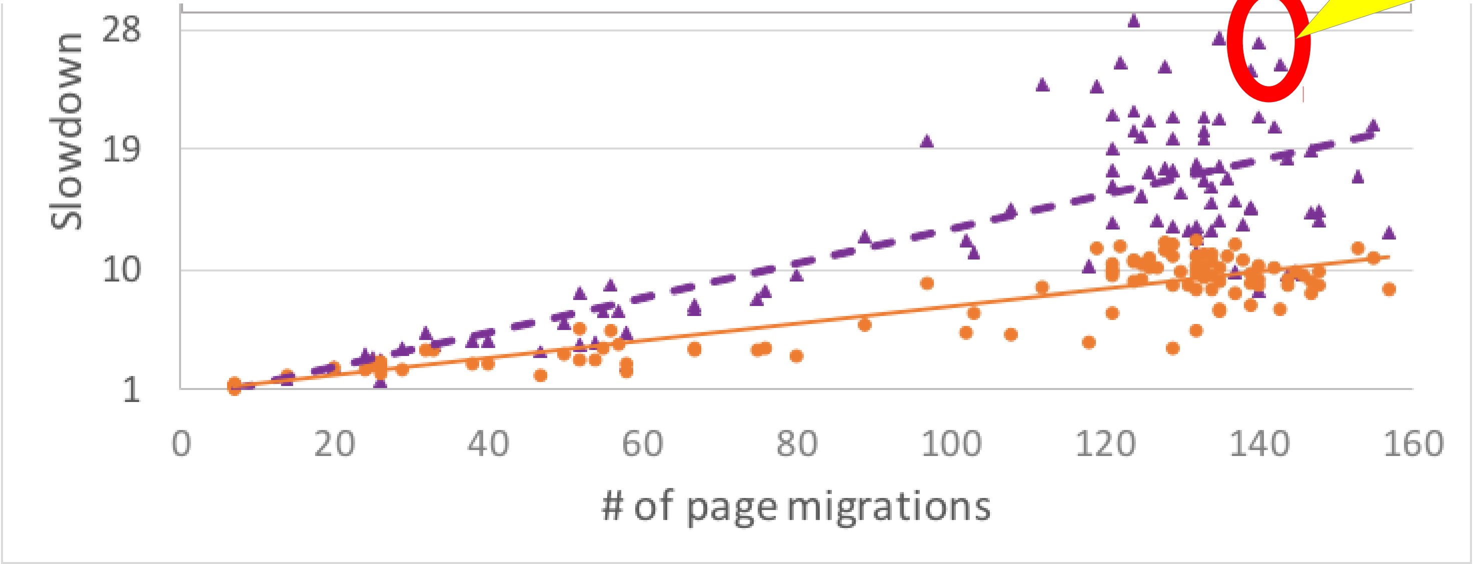
- Current practice in NVIDIA Unified Virtual Memory
- Single owner semantics: the page migrates to the requesting processor

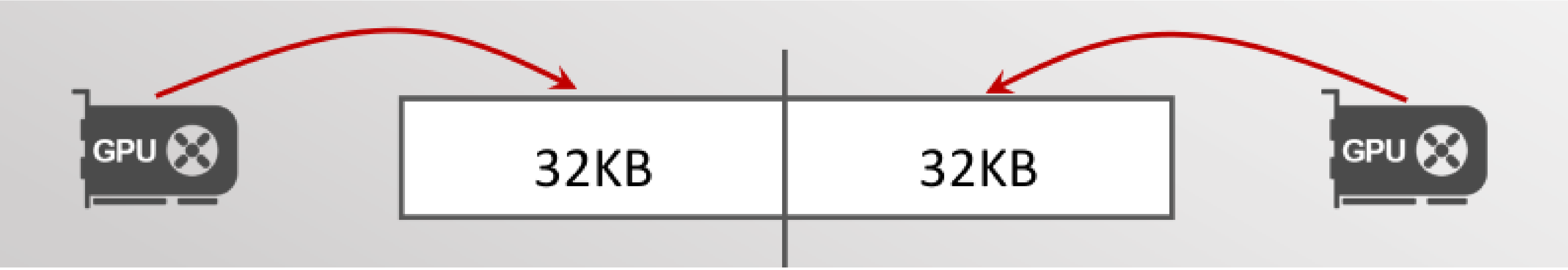
But GPU page is 64KB!
False sharing inevitable
(also in real applications)



Extreme false sharing is devastating

28x slowdown!





March 2021

Lazy Release Consistency to rescue

GPU management code on the CPU

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

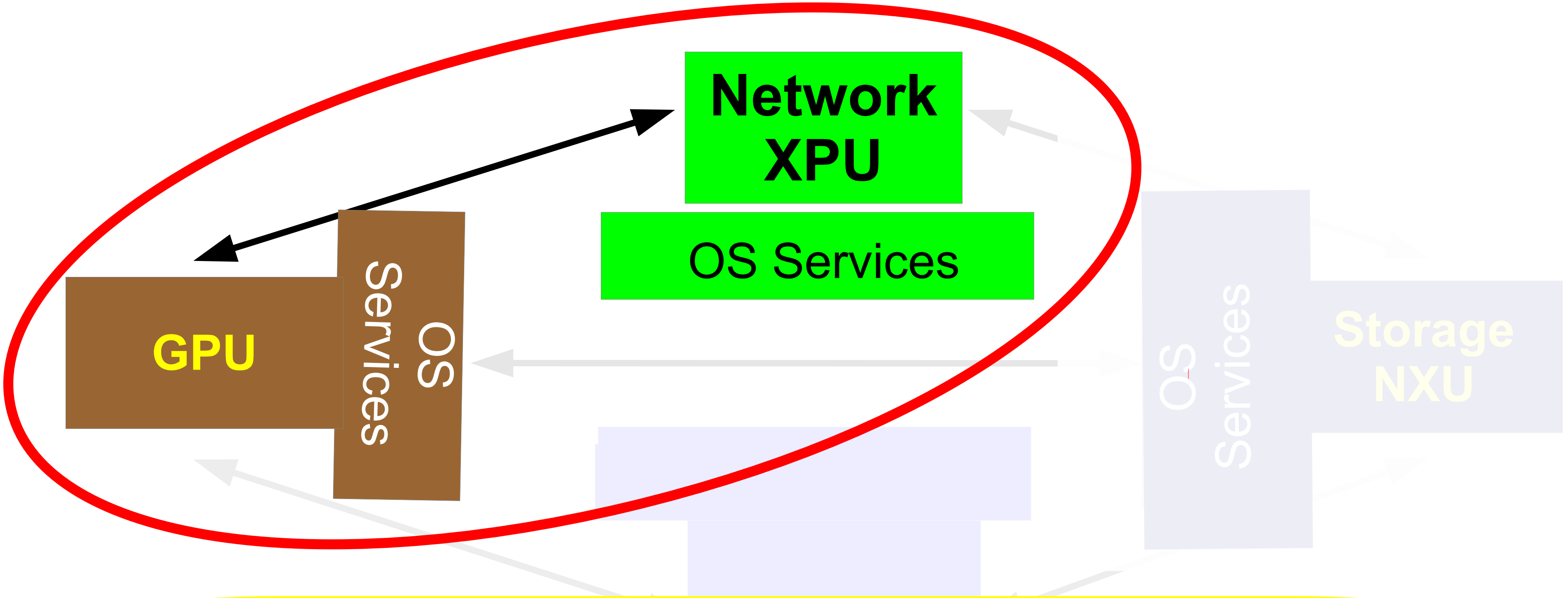
40% app improvement over strong consistency

- *Transparent for legacy CPU processes*
- *Transparent for legacy GPU kernels*

Summary so far...

- Accelerator-centric OS services
  - Simplify code development for accelerators
  - Enable transparent performance optimization
- Accelerator-aware host OS services
  - Optimize I/O for unmodified accelerators
  - Coordinate sharing with the host OS
- But can we ***remove*** *host* CPUs altogether?

CPU-less design: no CPU in control and data path



Lower latency (no CPU roundtrip)
Better scalability (no CPU load)
Lower costs (wimpy CPUs)

CPU's role

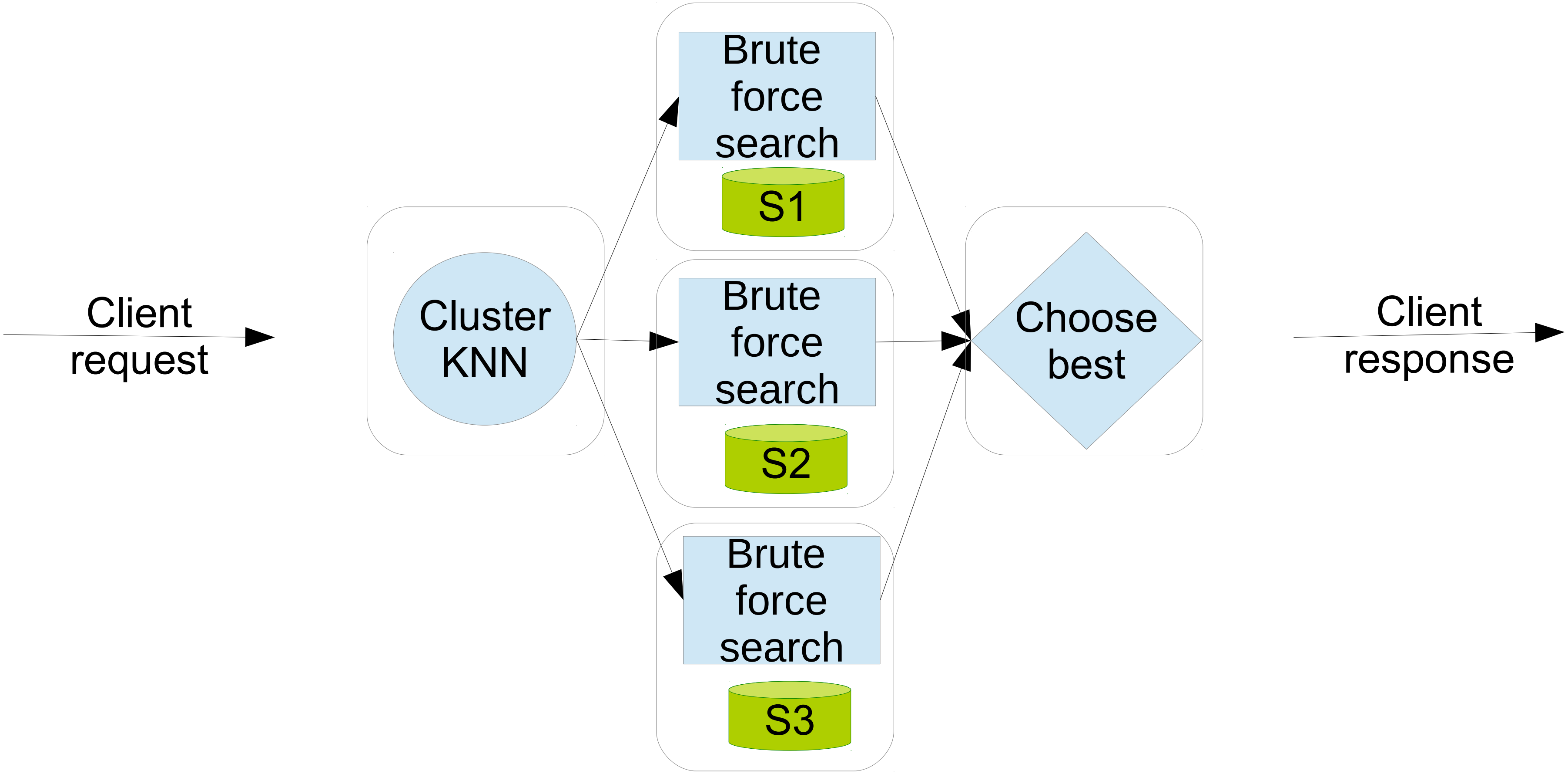
Do the setup
Then leave

CPU-less systems

- GPUrdma [ROSS'16]
  - RDMA VERBs from GPUs
  - Achieves 2-3 usec latency and high throughput
- Centaur [PACT'19]
  - Multi-GPU UNIX sockets and data flow runtime
  - Multi-GPU scaling with zero CPU utilization
- NICA [ATC'19]
  - Inline server acceleration on FPGA-based SmartNICs
- LYNX [ASPLOS'20]
  - Accelerator-centric server architecture on SmartNICs

The case for CPU-less multi-GPU server design

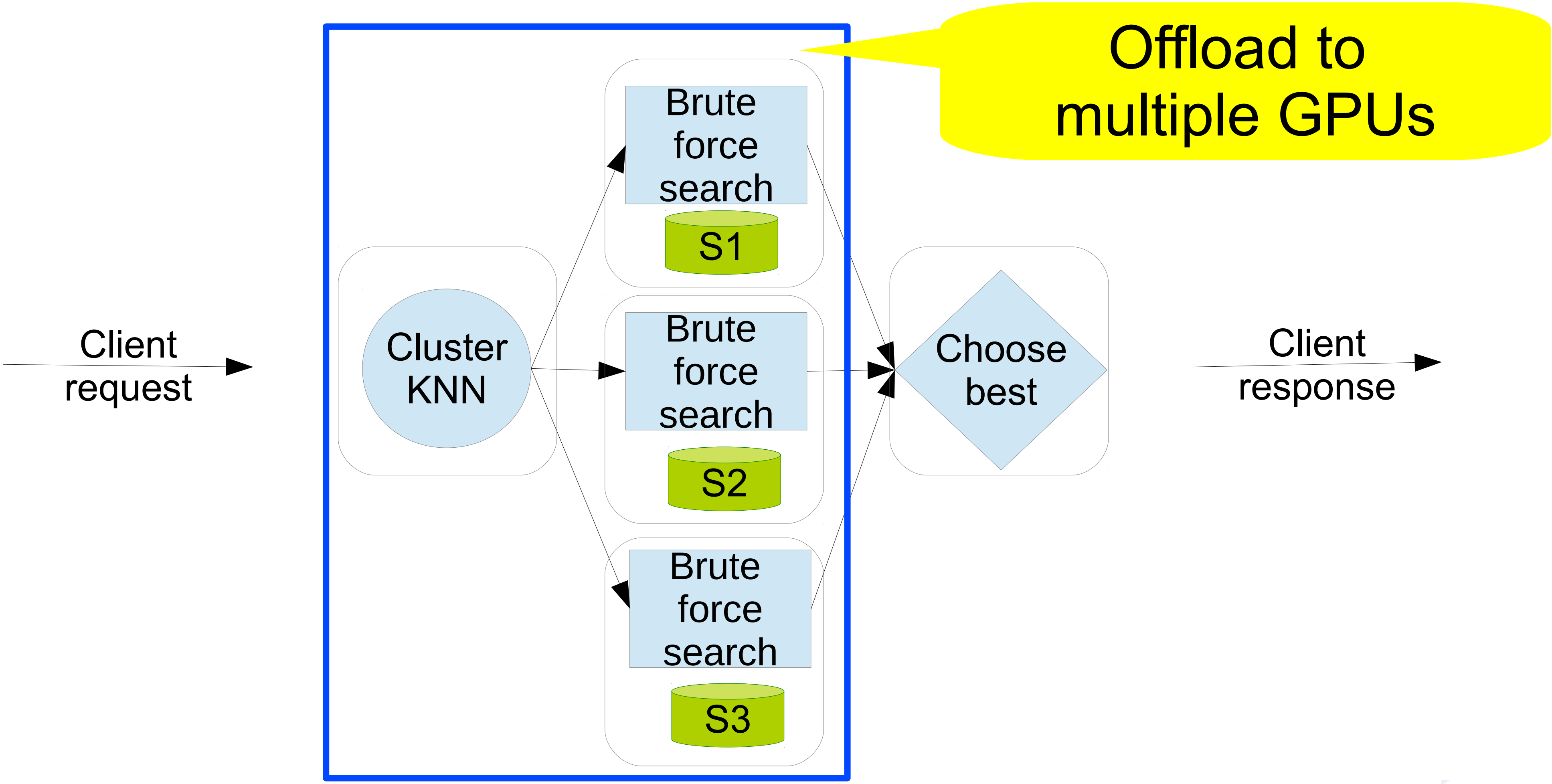
Image Similarity Search



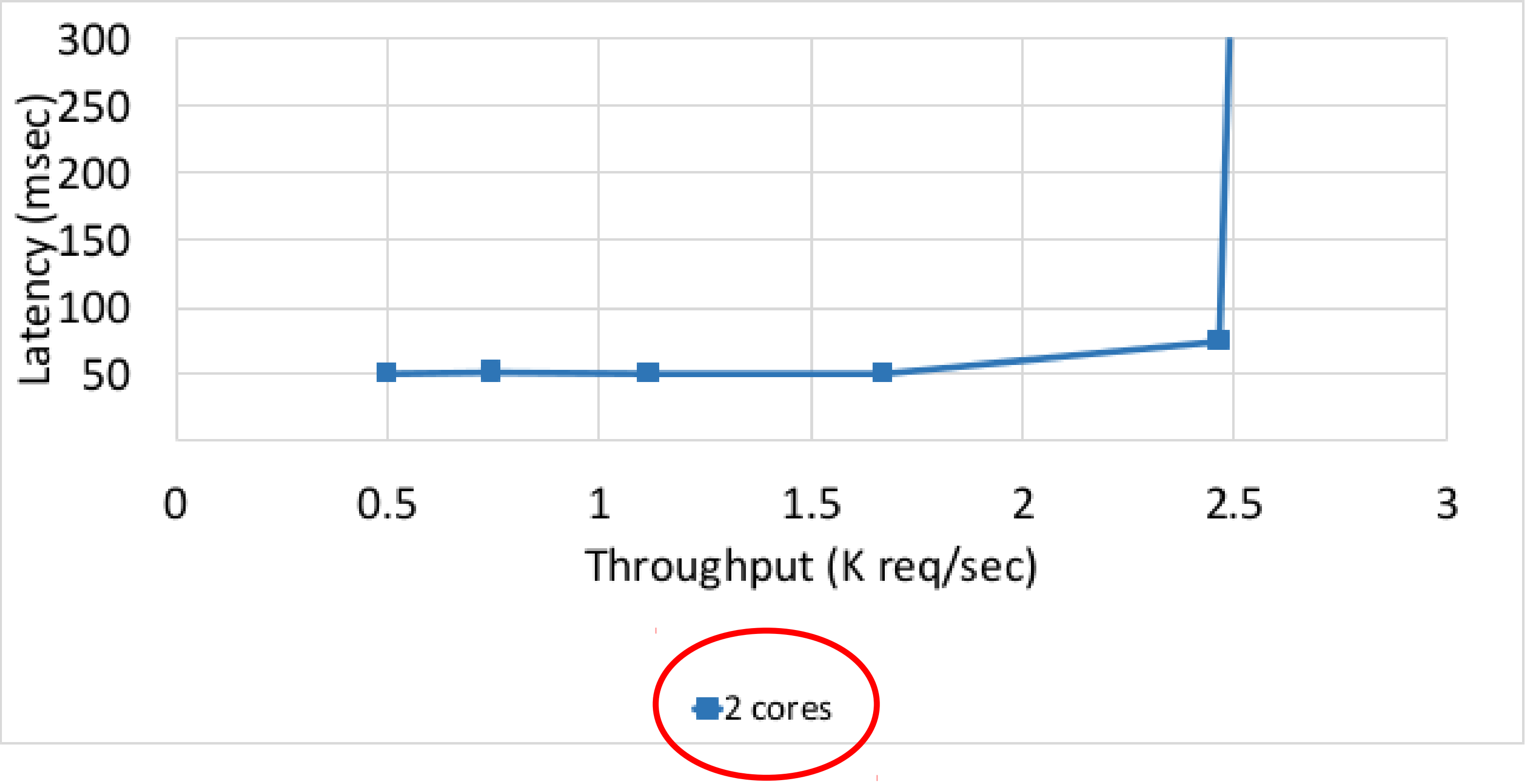
March 2021

Accelerated Computing Systems Lab

Traditional design: CPU controls GPU invocation and data movements

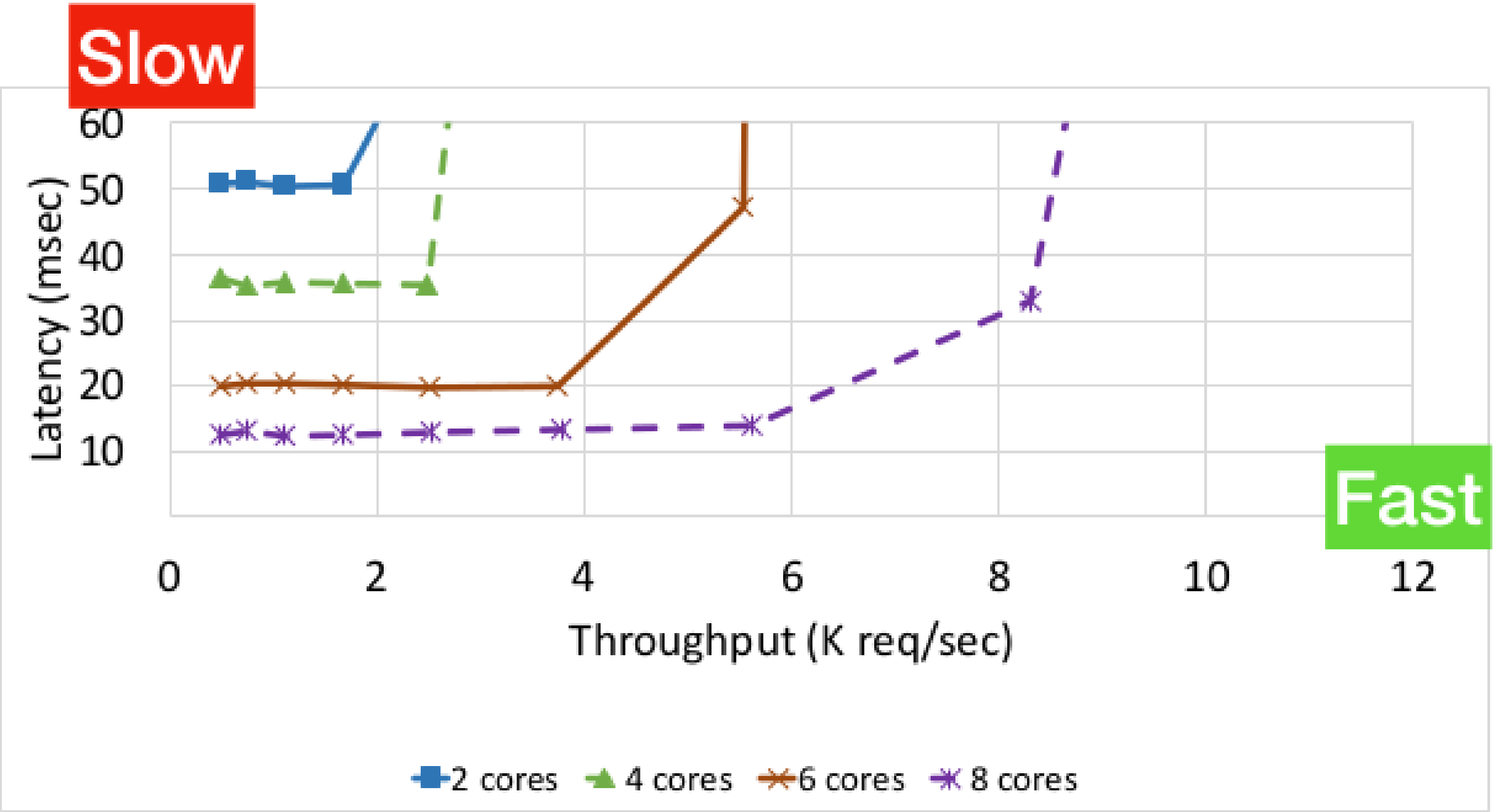


Traditional design: CPU controls GPU invocation and data movements

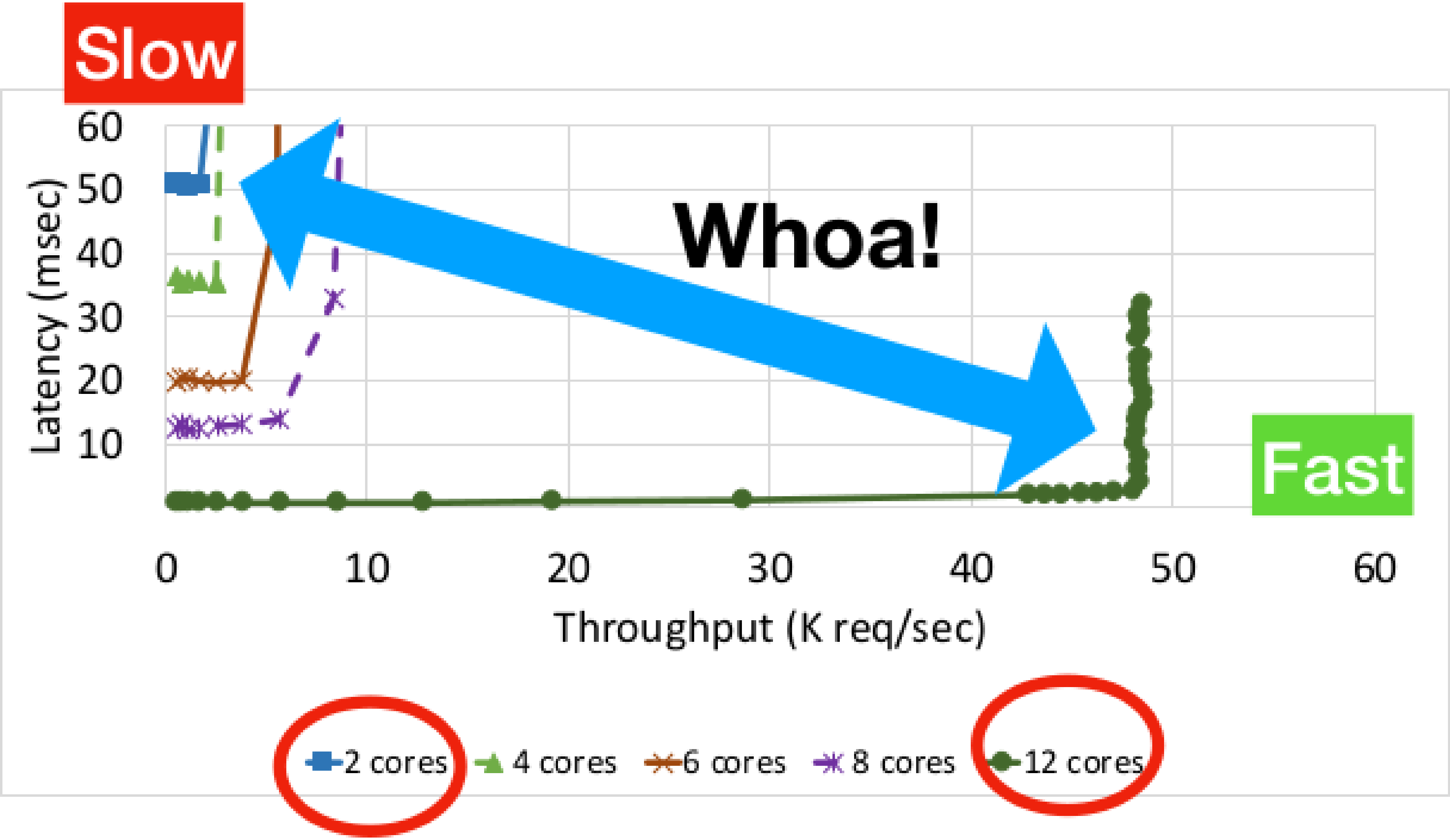


6 GPUs

Lets add more CPU cores

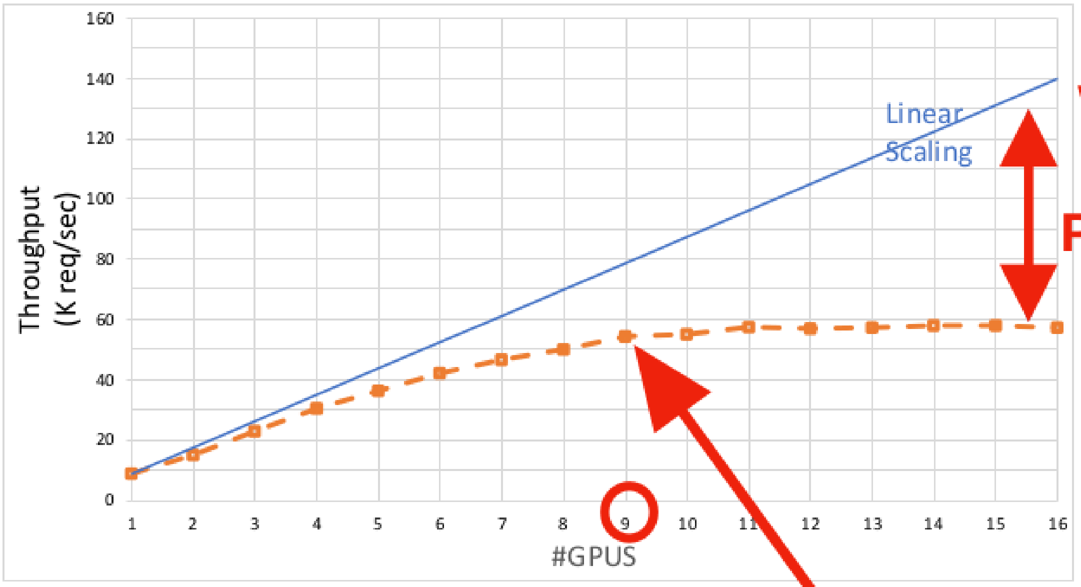


12 CPU cores needed!



12 CPUs are not enough to scale!

More GPUs



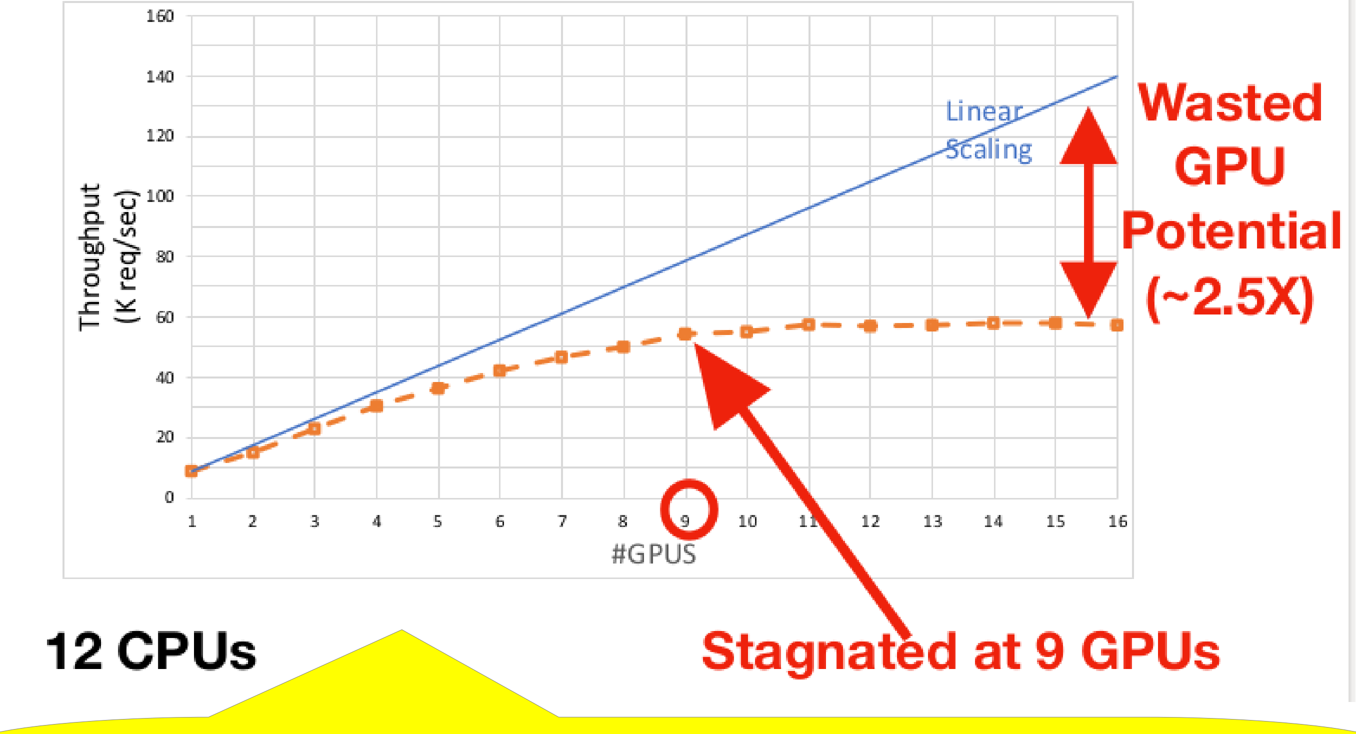
Wasted GPU Potential (~2.5X)

12 CPUs

Stagnated at 9 GPUs

12 CPUs are not enough to scale!

More GPUs



Wasted GPU Potential (~2.5X)

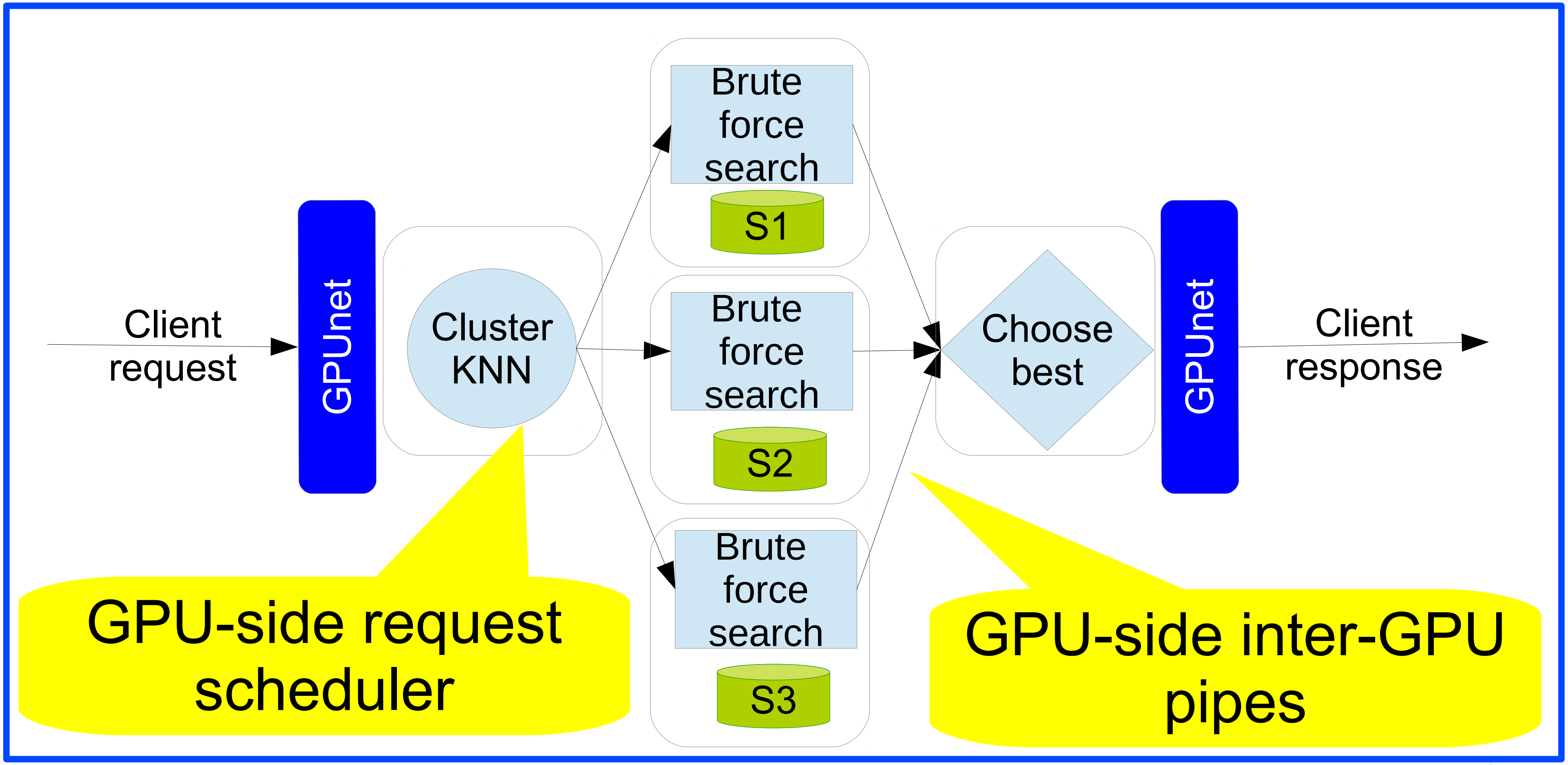
12 CPUs

Stagnated at 9 GPUs

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

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

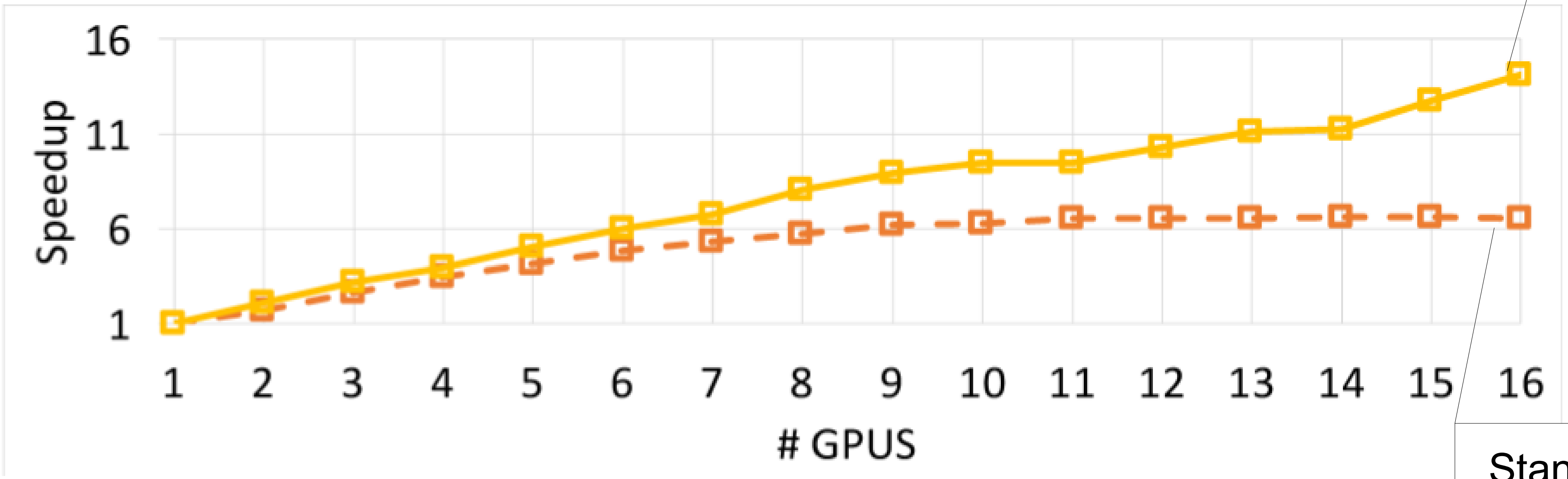
PACT 19



GPU-side request scheduler

GPU-side inter-GPU pipes

CPU-less Multi-GPU network server



CPU-less design

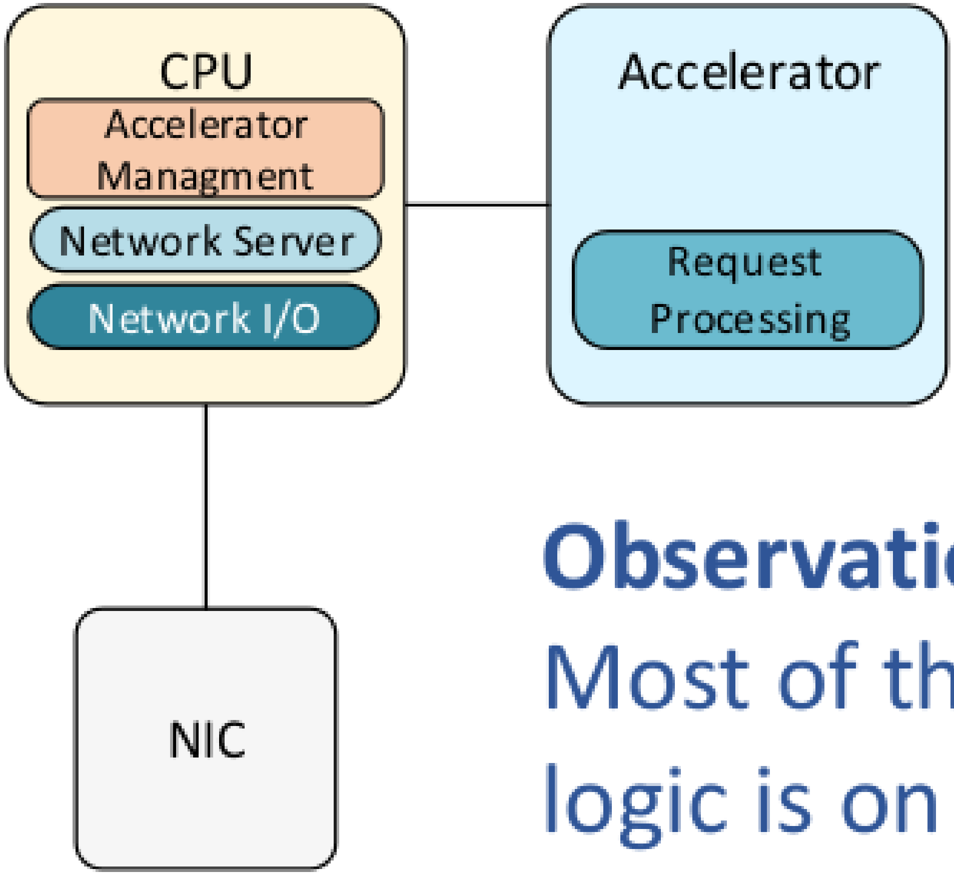
Standard CPU-driven design

CPU-less design: better scaling

CPU-less systems

- GPUrdma [ROSS'16]
  - RDMA VERBs from GPUs
  - Achieves 2-3 usec latency and high throughput
- Centaur [PACT'19]
  - Multi-GPU UNIX sockets and data flow runtime
  - Multi-GPU scaling with zero CPU utilization
- NICA [ATC'19]
  - Inline server acceleration on FPGA-based SmartNICs
- LYNX [ASPLOS'20]
  - Accelerator-centric server architecture on SmartNICs

Traditional host-centric



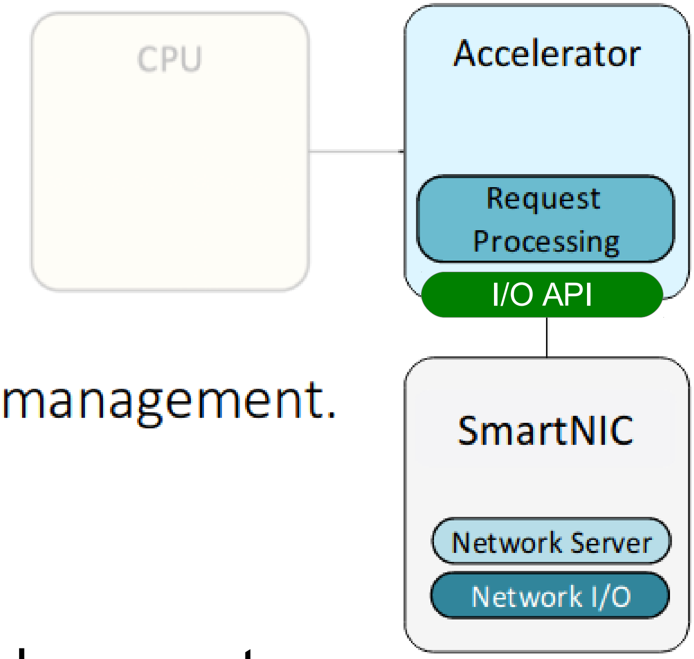
Observation

Most of the application-specific logic is on accelerator!

Lynx - Vision

Goal

Demonstrate and build a general accelerated-centric server.



How?

Use SmartNIC for network processing and accelerator management.

- **Full CPU offload**
- **No application code on SmartNIC**

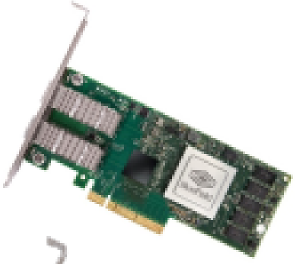
Thin on-accelerator abstractions for serving network requests

Transport processing offloaded to the SmartNIC

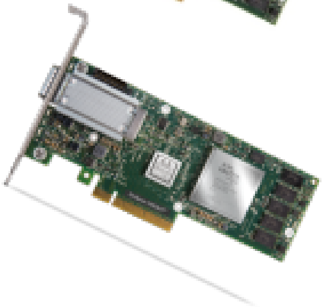
Implementation

SmartNICs

- ARM-based (Bluefield)



- FPGA-based (Innova)

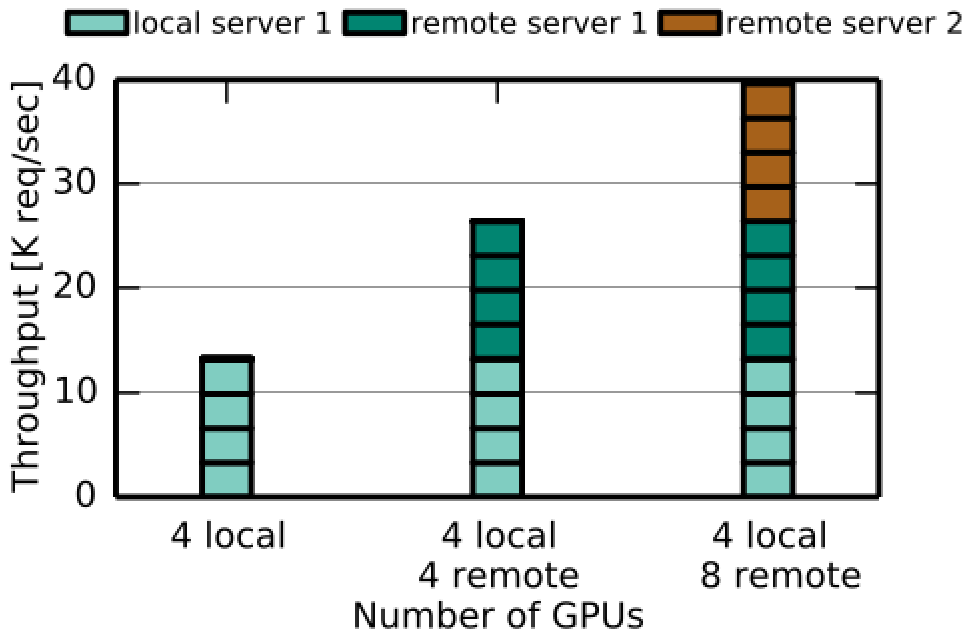


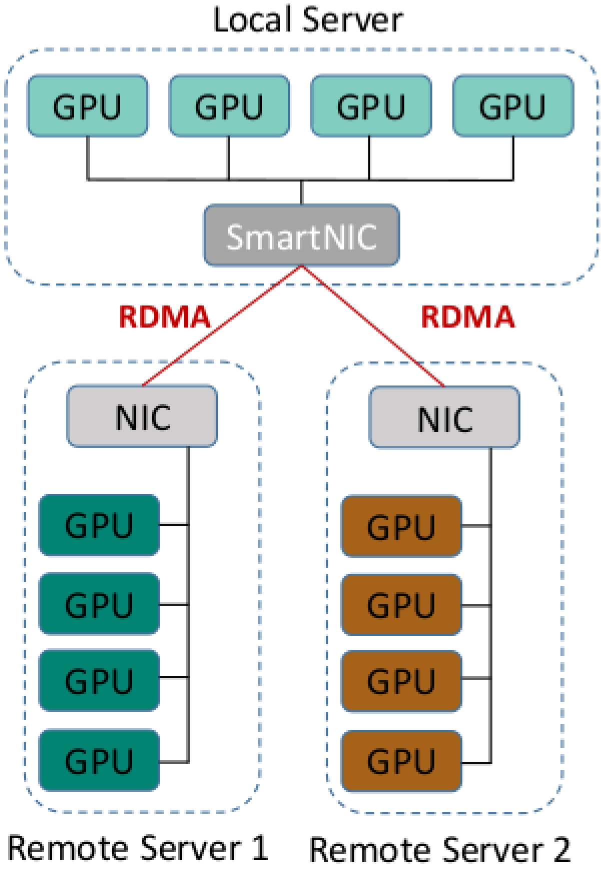
Accelerators

- NVIDIA GPU

- Intel Visual Computer Accelerator – VCA

Inference server: Scalability with disaggregated GPUs





1 SmartNIC can support up to 100 accelerators performing neural net inference

Productized in Toga Networks [Huawei] as we speak

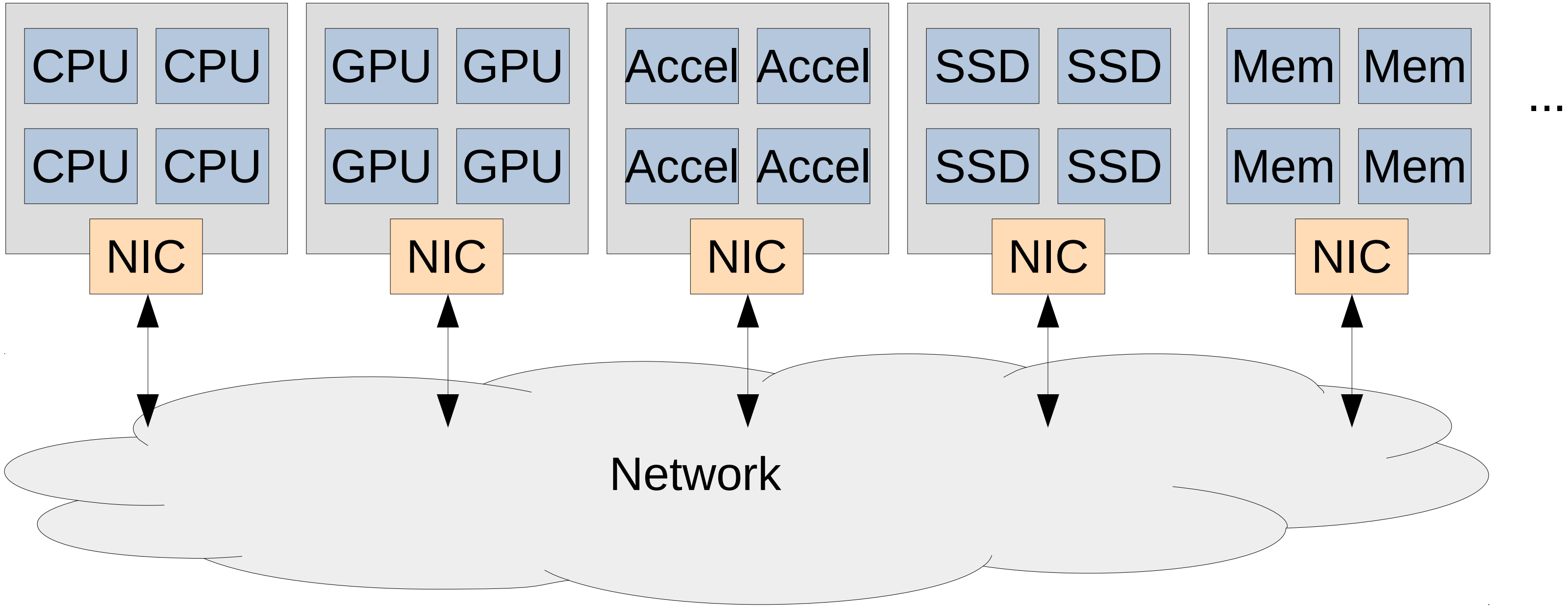
Summary so far..

- Accelerator-centric OS architecture is feasible today
- Advantageous for high performance, resource efficiency, code simplicity
- On-accelerator libraryOS approach with the CPU used for privileged operations

- **But will it apply to future *disaggregated systems*?**

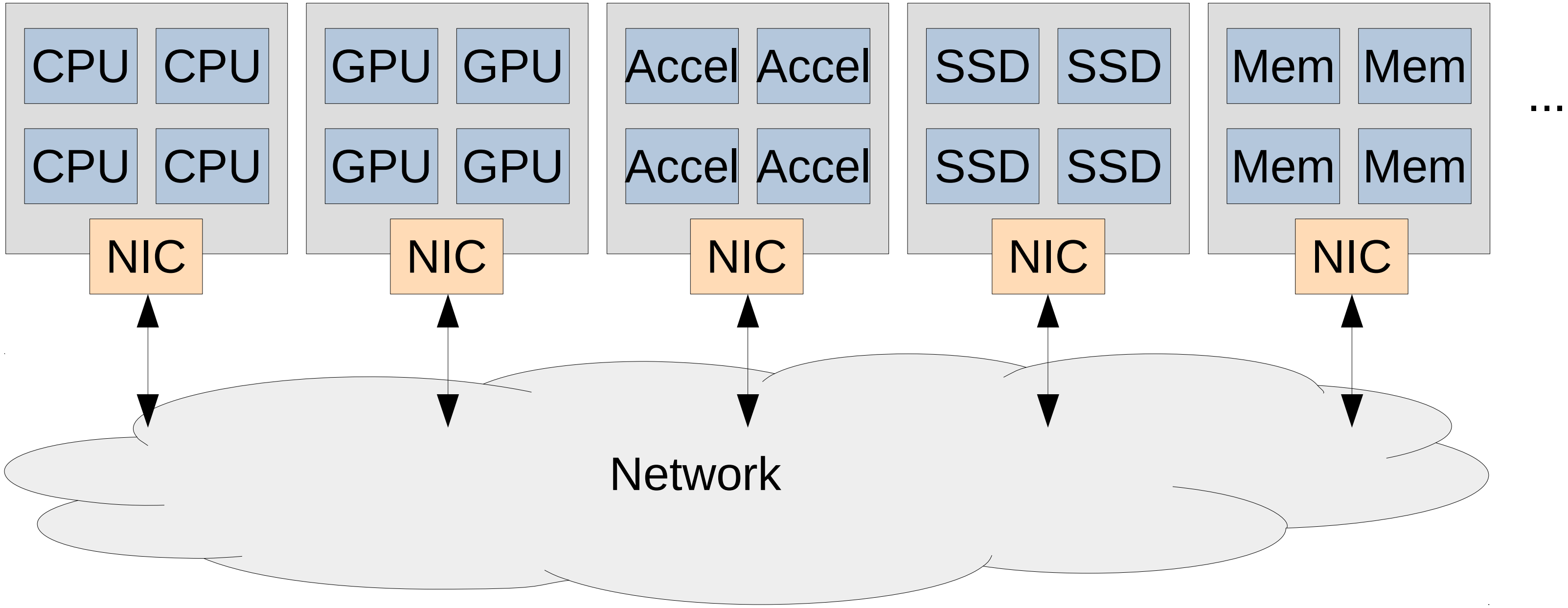
Data Center Trends

- Hardware: Resource disaggregation
- High benefits in TCO and utilization



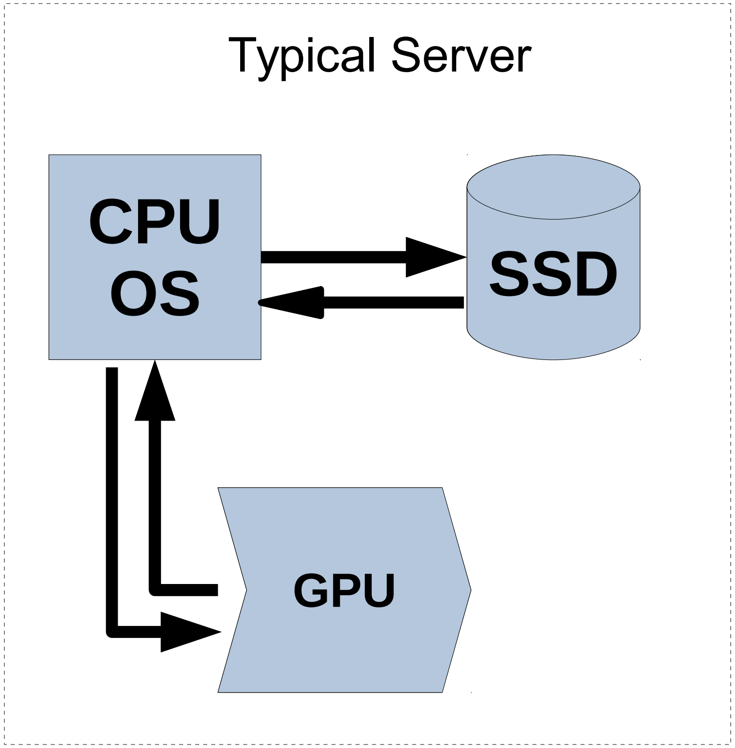
Data Center Trends

- Hardware: Resource disaggregation
- High benefits in TCO and utilization

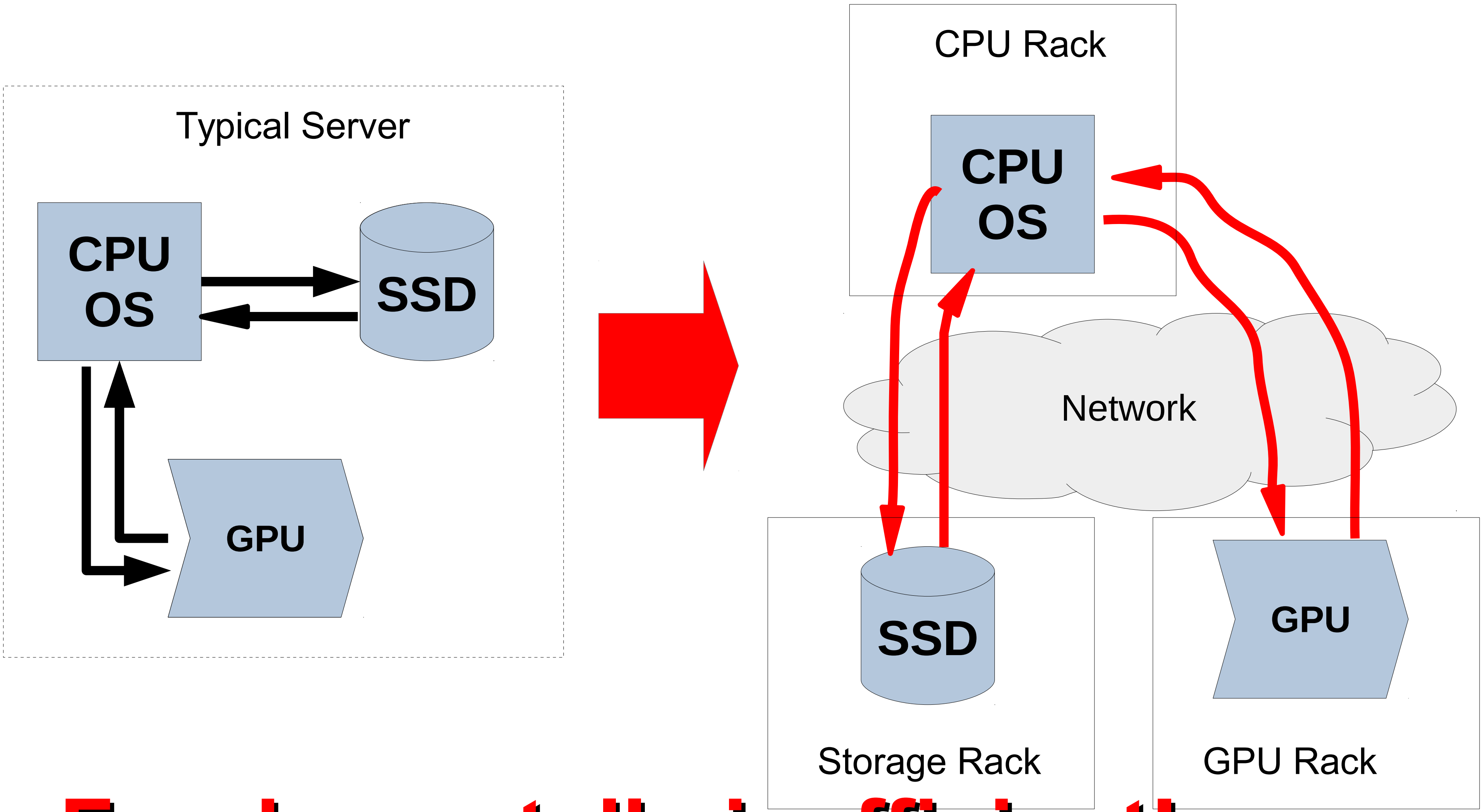


But what about performance?

Not with the traditional ***server***-centric design



Not with the traditional ***server***-centric design

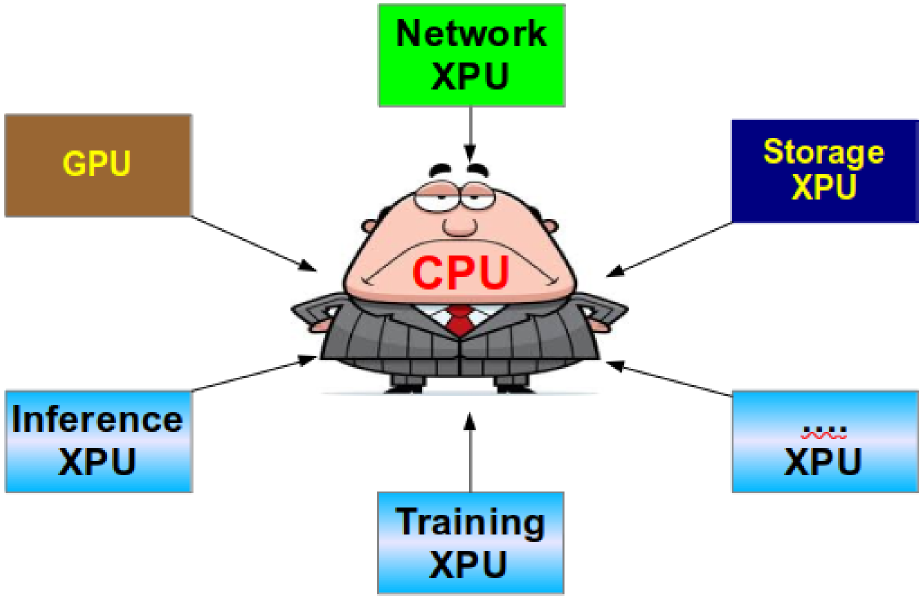


Fundamentally inefficient!

What's wrong with the server-centric design ?

- A centralized OS is a control/data bottleneck
- I/O devices and accelerators are *slaves*
- Application control and data planes are centralized

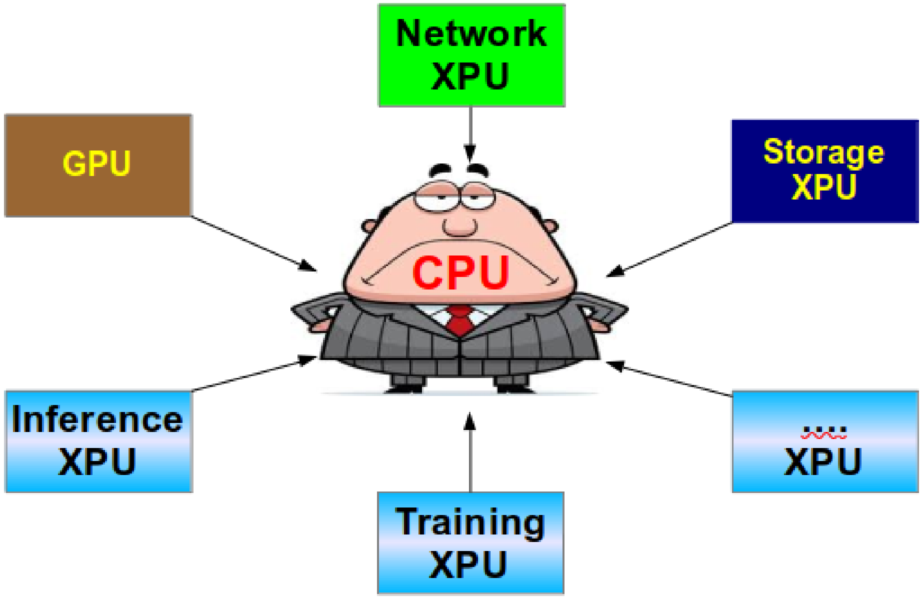
OS architecture is CPU - centric



What's wrong with the server-centric design ?

- A centralized OS is a control/data bottleneck
- I/O devices and accelerators are *slaves*
- Application control and data planes are centralized

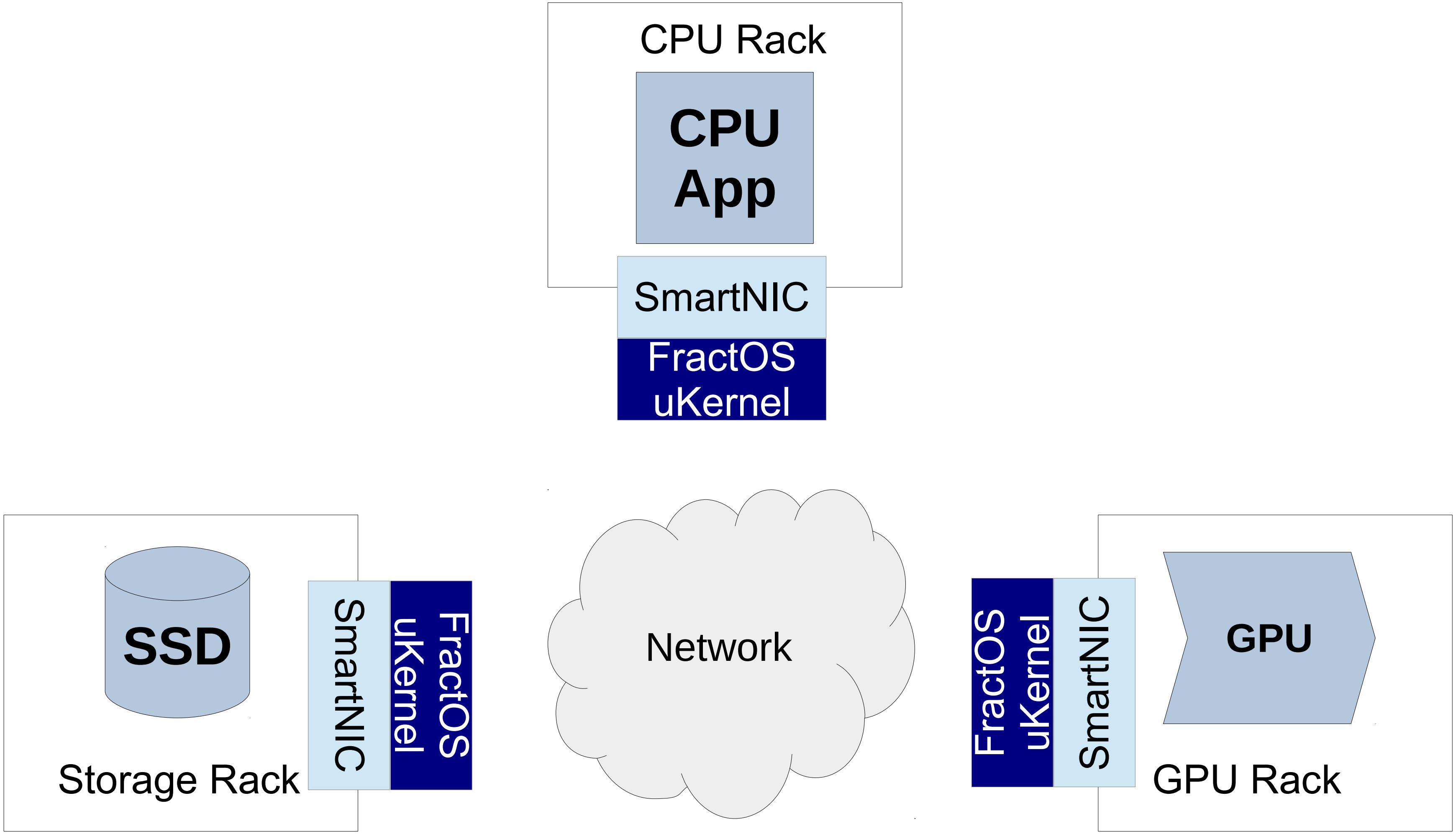
OS architecture is CPU - centric



Needed disaggregation-native OS!

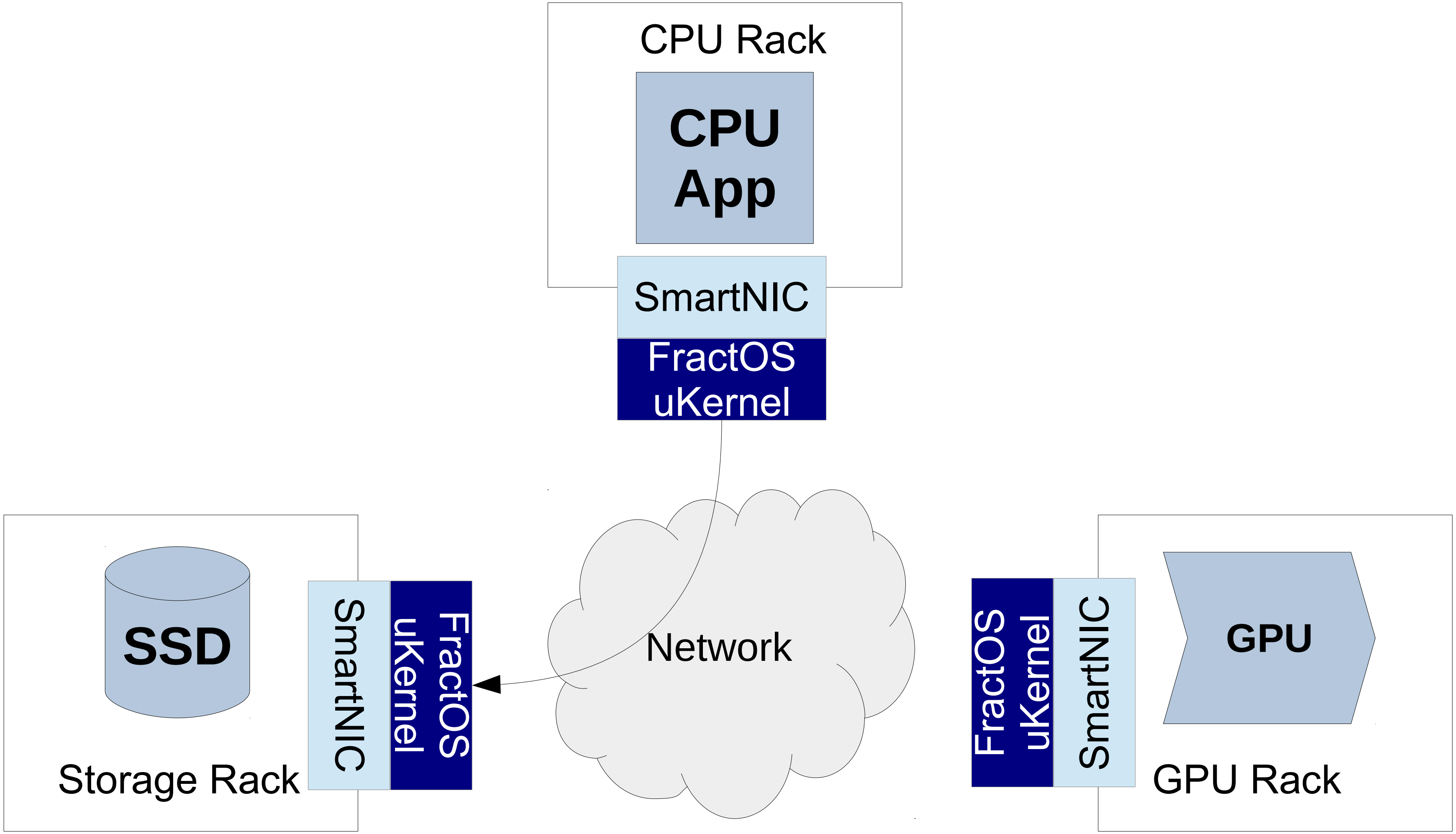
FractOS: decentralized disaggregation-native OS

Joint work with L Vilanova (Imperial), H. Haertig and his team (TU Dresden & Bakhausen)



FractOS: decentralized disaggregation-native OS

Joint work with L Vilanova (Imperial), H. Haertig and his team (TU Dresden & Bakhausen)

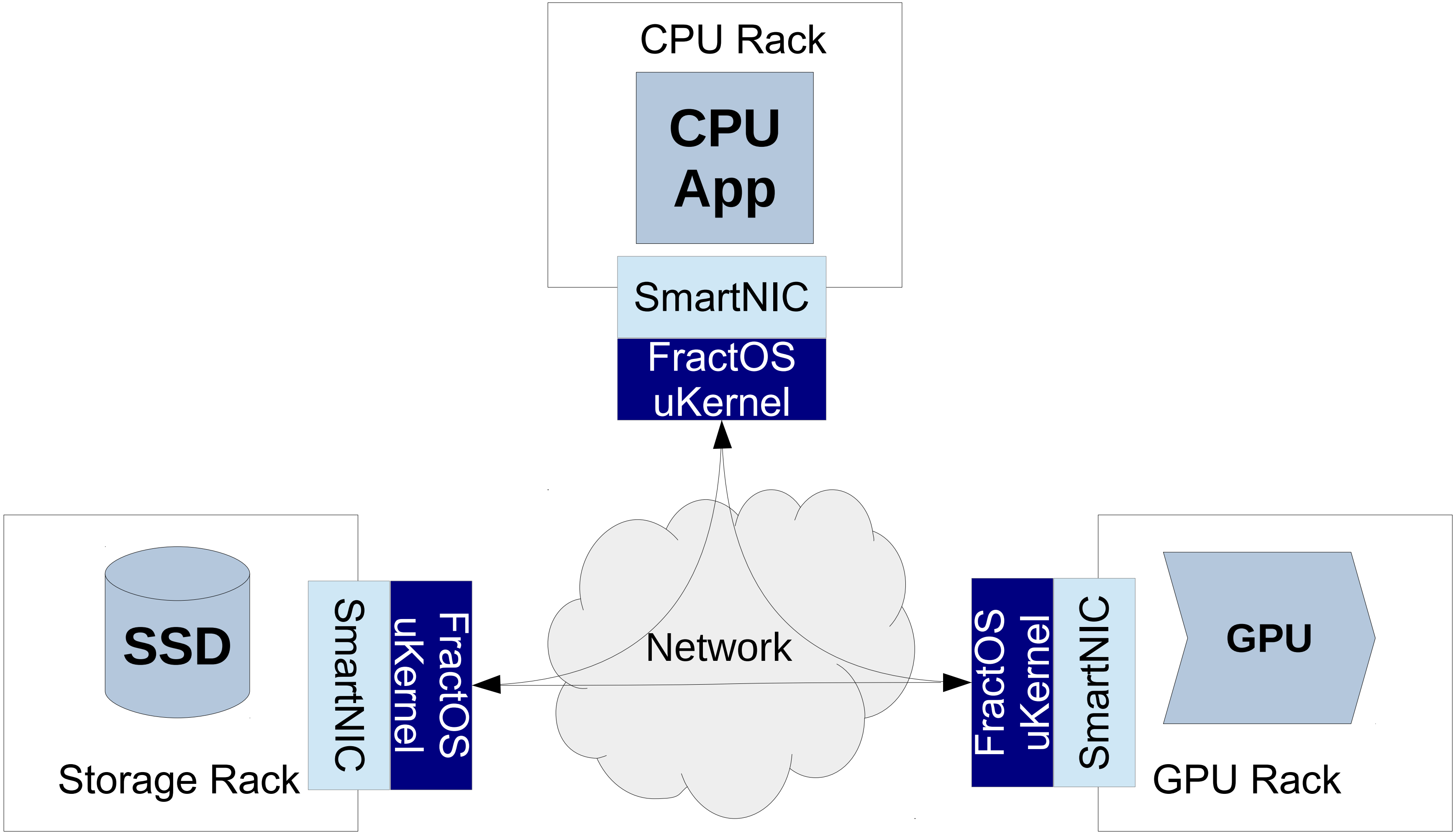


May 2021

Accelerated Computing Systems Lab

FractOS: decentralized disaggregation-native OS

Joint work with L Vilanova (Imperial), H. Haertig and his team (TU Dresden & Bakhausen)



May 2021

FractOS vs. OmniX

- Avoid CPU in data/control path
- Devices as first-class citizens
- Direct interaction among devices
- Transparent data-path optimizations

OmniX

- Decentralized capability management
- Decentralized task graph execution
- Unified software/hardware interfaces

Summary

- Future omni-programmable systems face **programmability wall**
- **Accelerator-centric OS architecture** simplifies programming and improves performance
- It exposes **OS abstractions on accelerators**
- Tightly integrates new abstractions **with the host OS**

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

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

AOSL

Accelerated Computing Systems Lab

Thank you!

mark@ee.technion.ac.il

https://marksilberstein.com