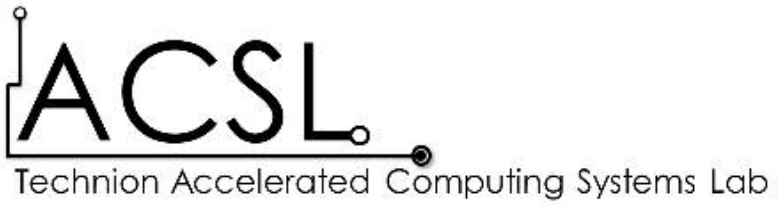
Omnix: an accelerator-centric OS for omni-programmable systems

Rethinking the role of CPUs in modern computers

Mark Silberstein

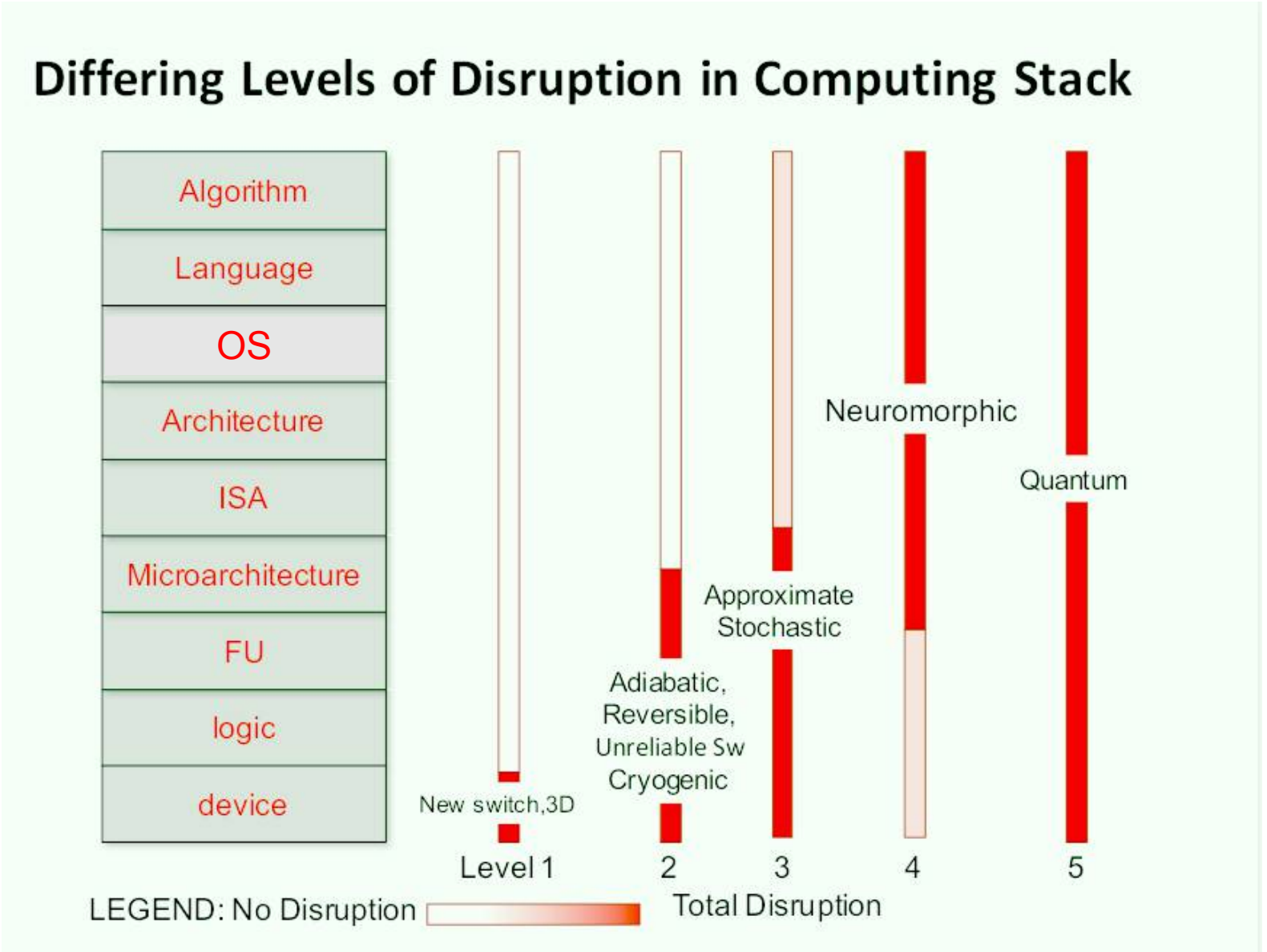


EE Technion

Beyond CMOS: from devices to systems
June 5-6, 2017

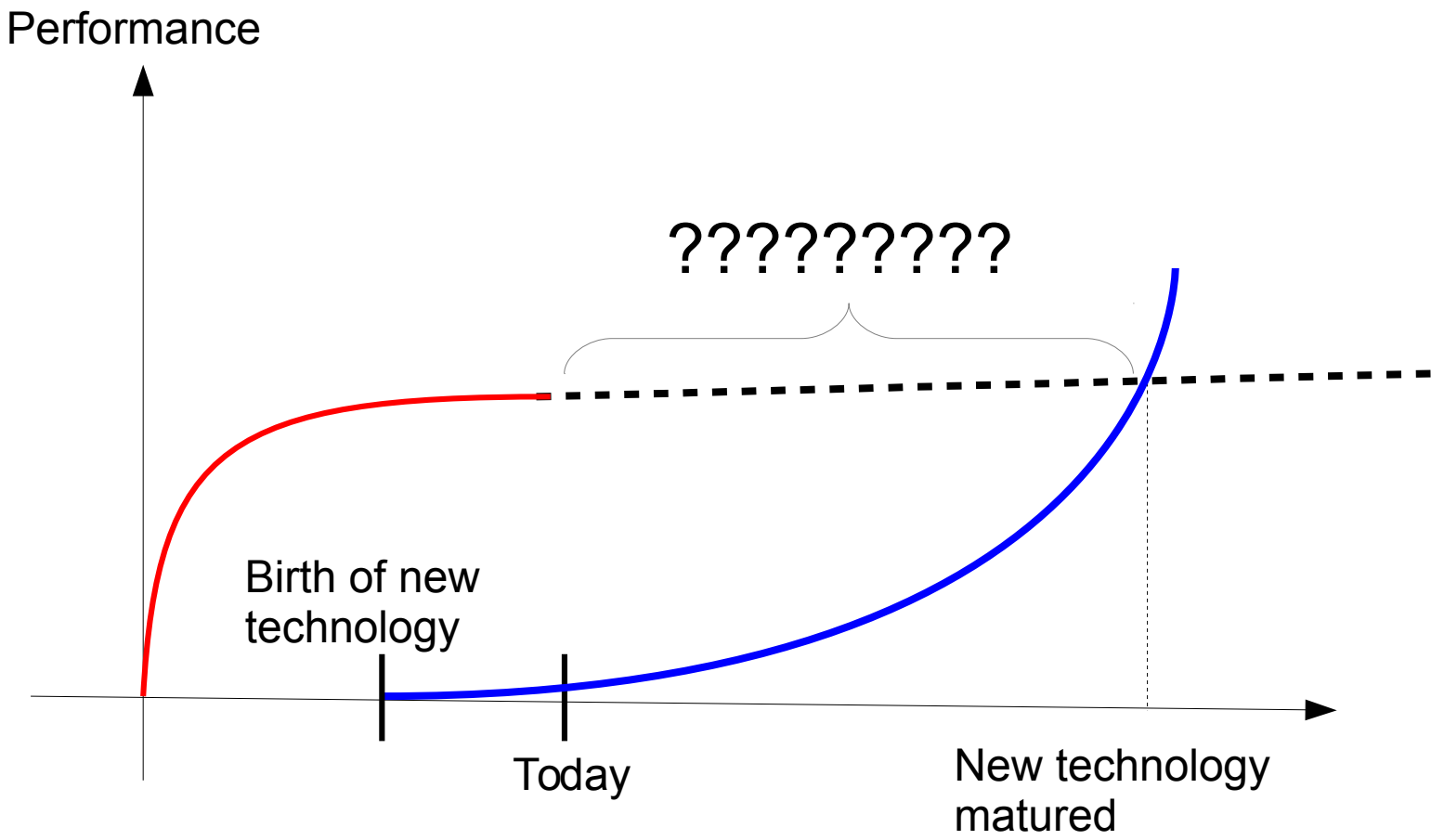
From devices to systems

Beyond CMOS: total disruption!

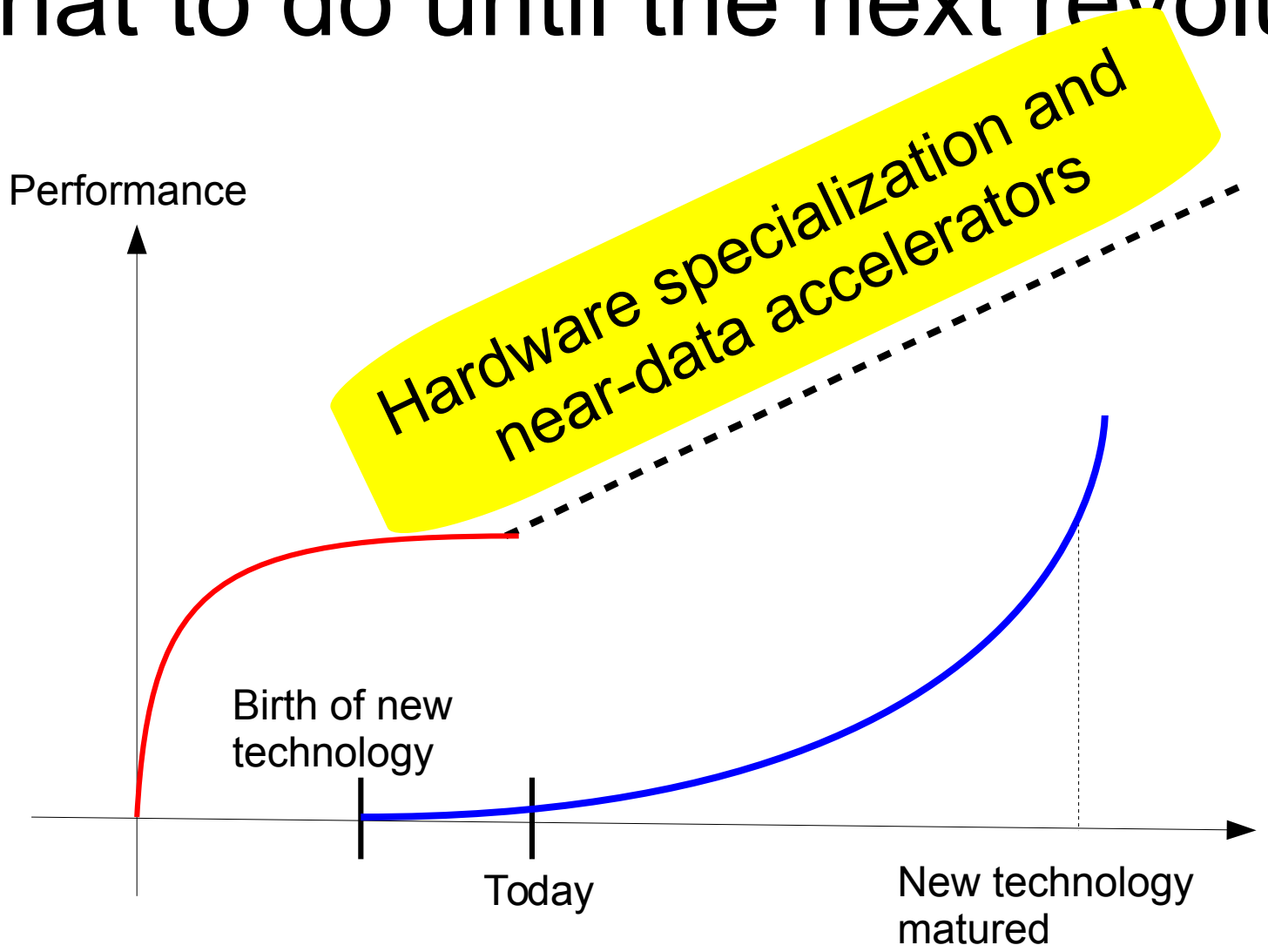


From «IEEE rebooting computing»

What to do until the next revolution?

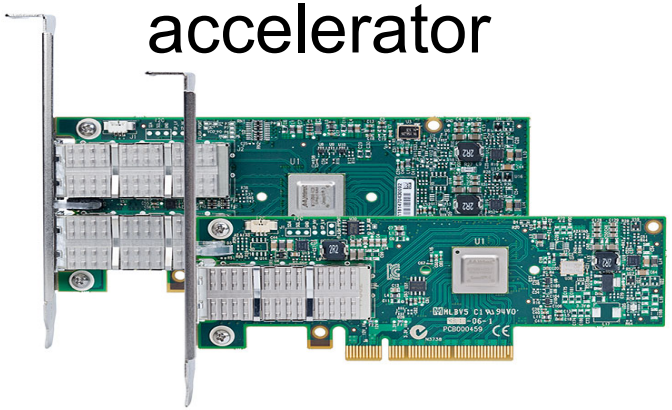


What to do until the next revolution?

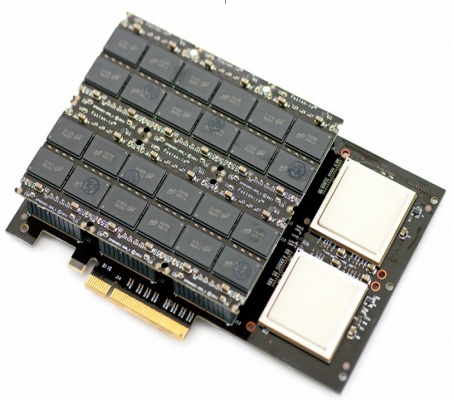


Computer hardware: circa ~2017

Network I/O accelerator

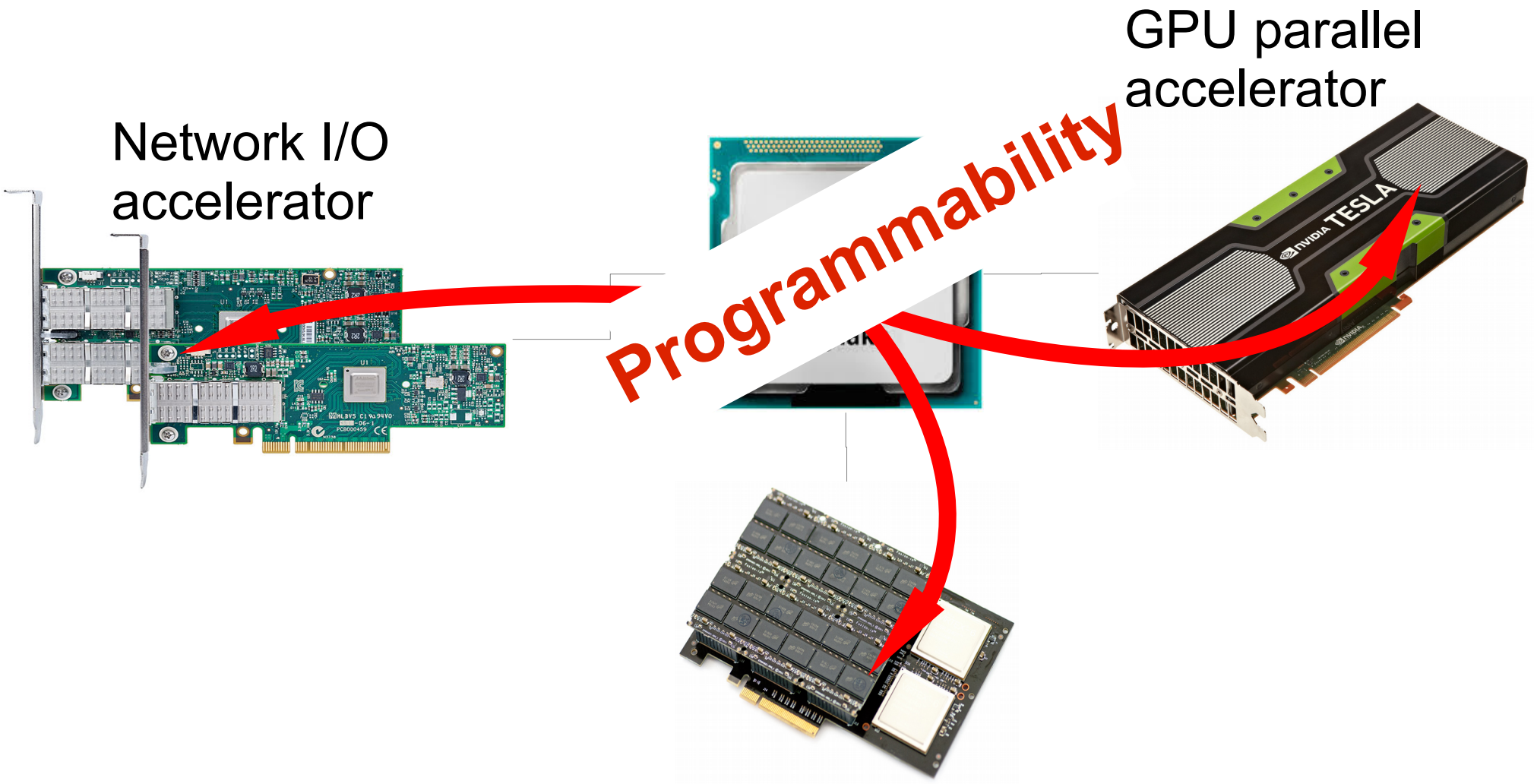


GPU parallel accelerator

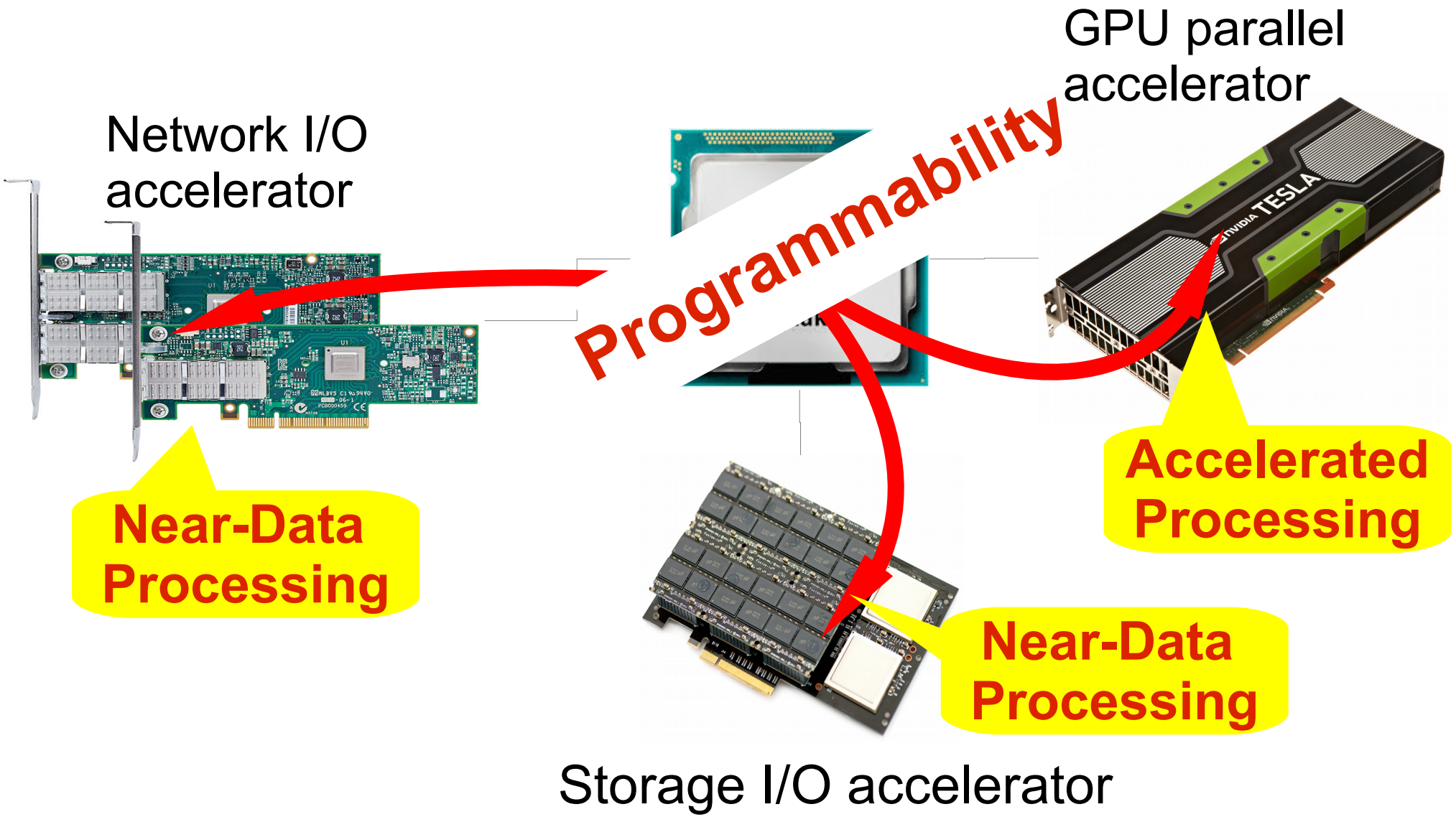


Storage I/O accelerator

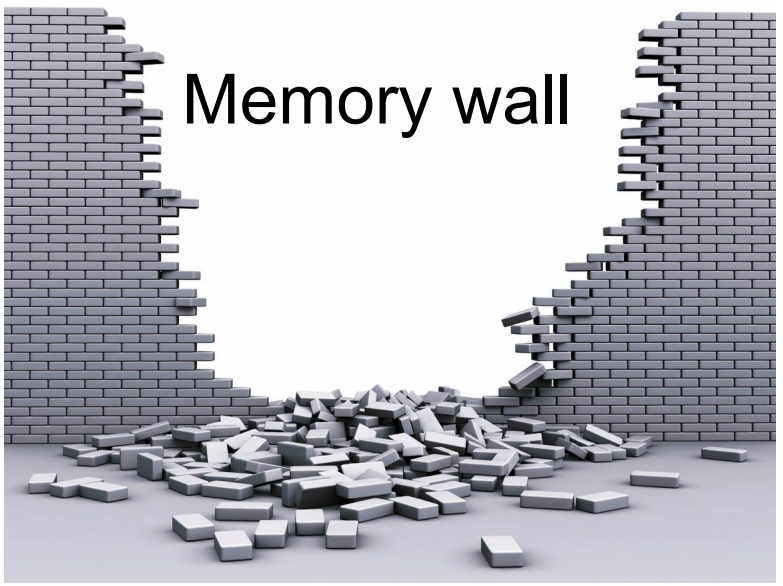
Central Processing Units (CPUs) are no longer **Central**



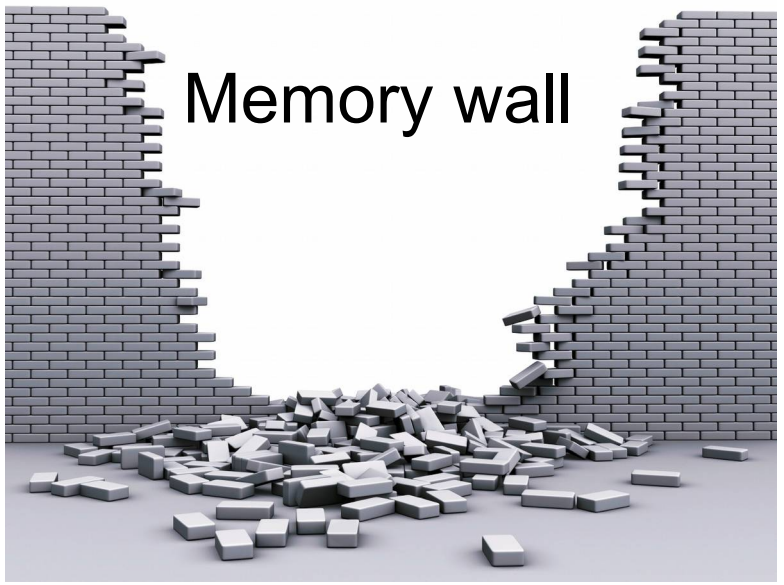
Omni-programmable system
Near-*X*-execution Units: NXUs

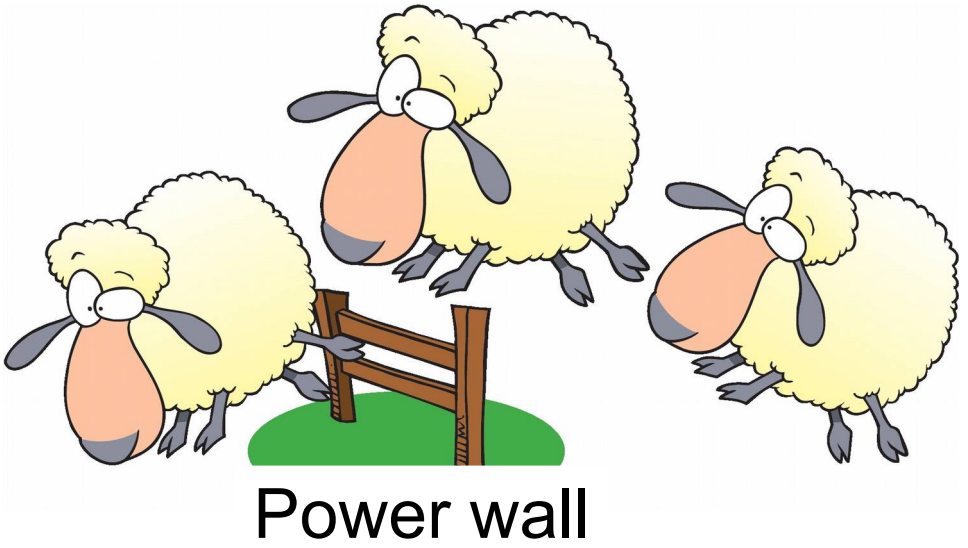


NXUs break walls

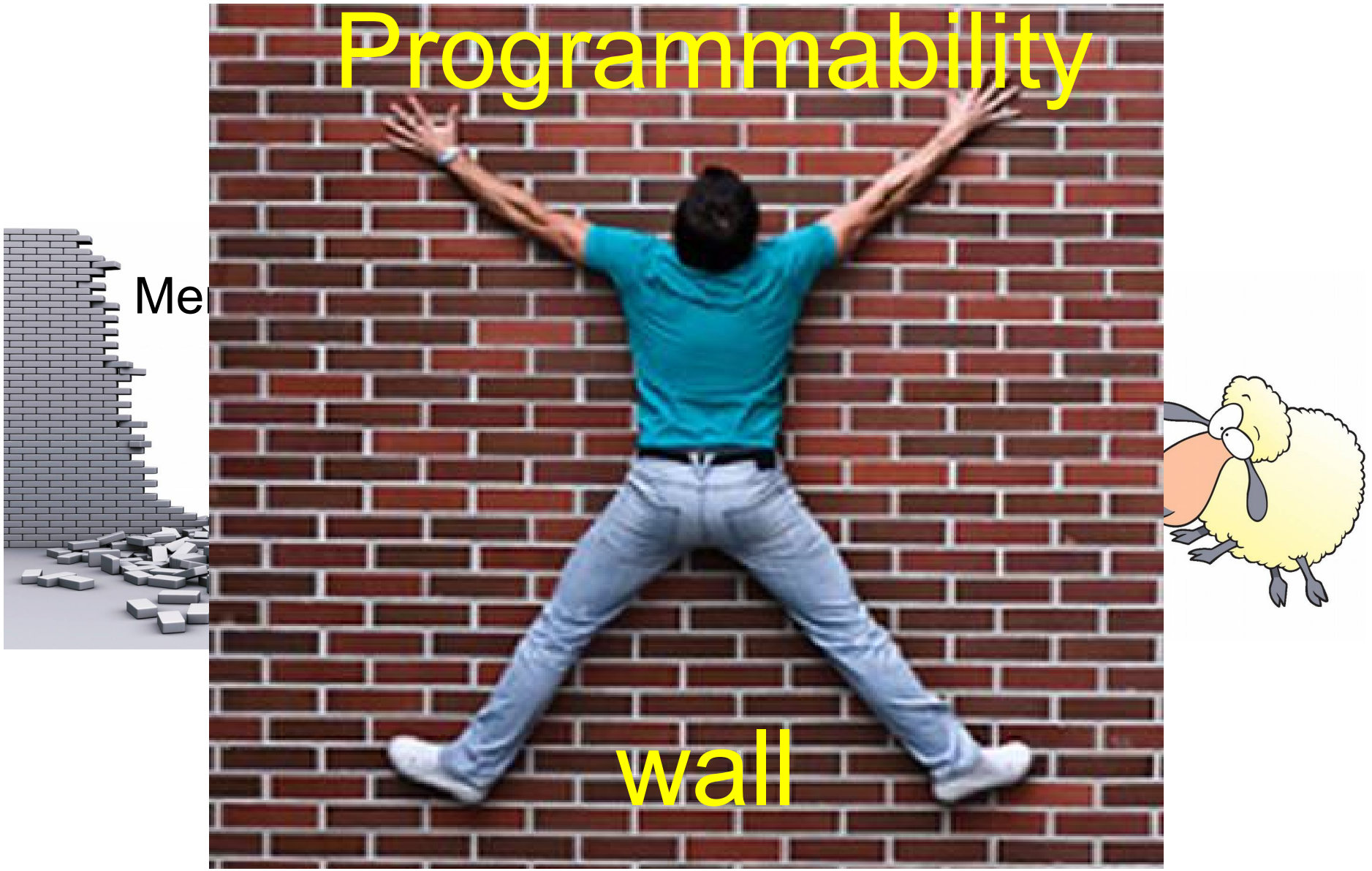


NXUs break walls
jump over walls





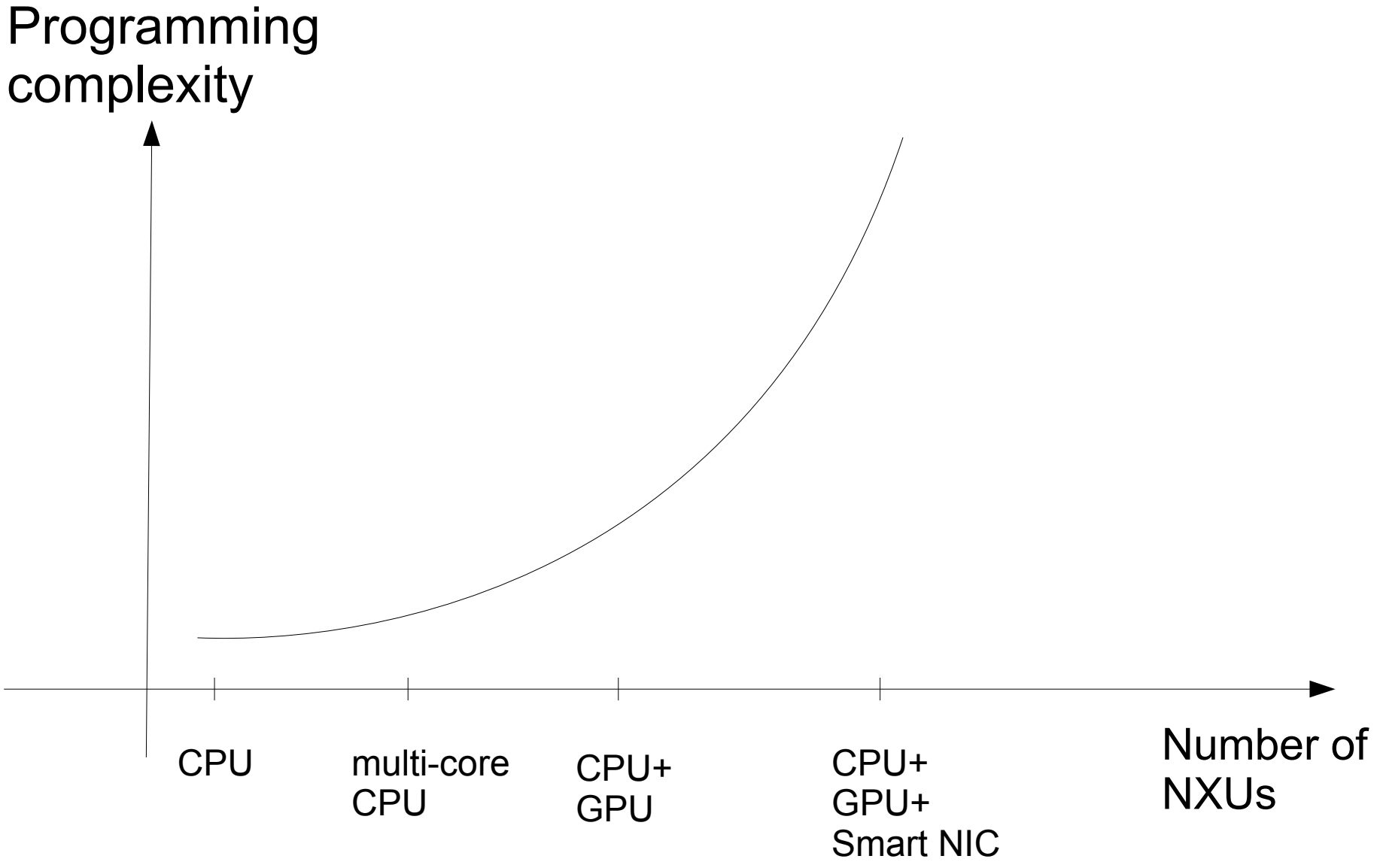
But NXUs also build new walls!



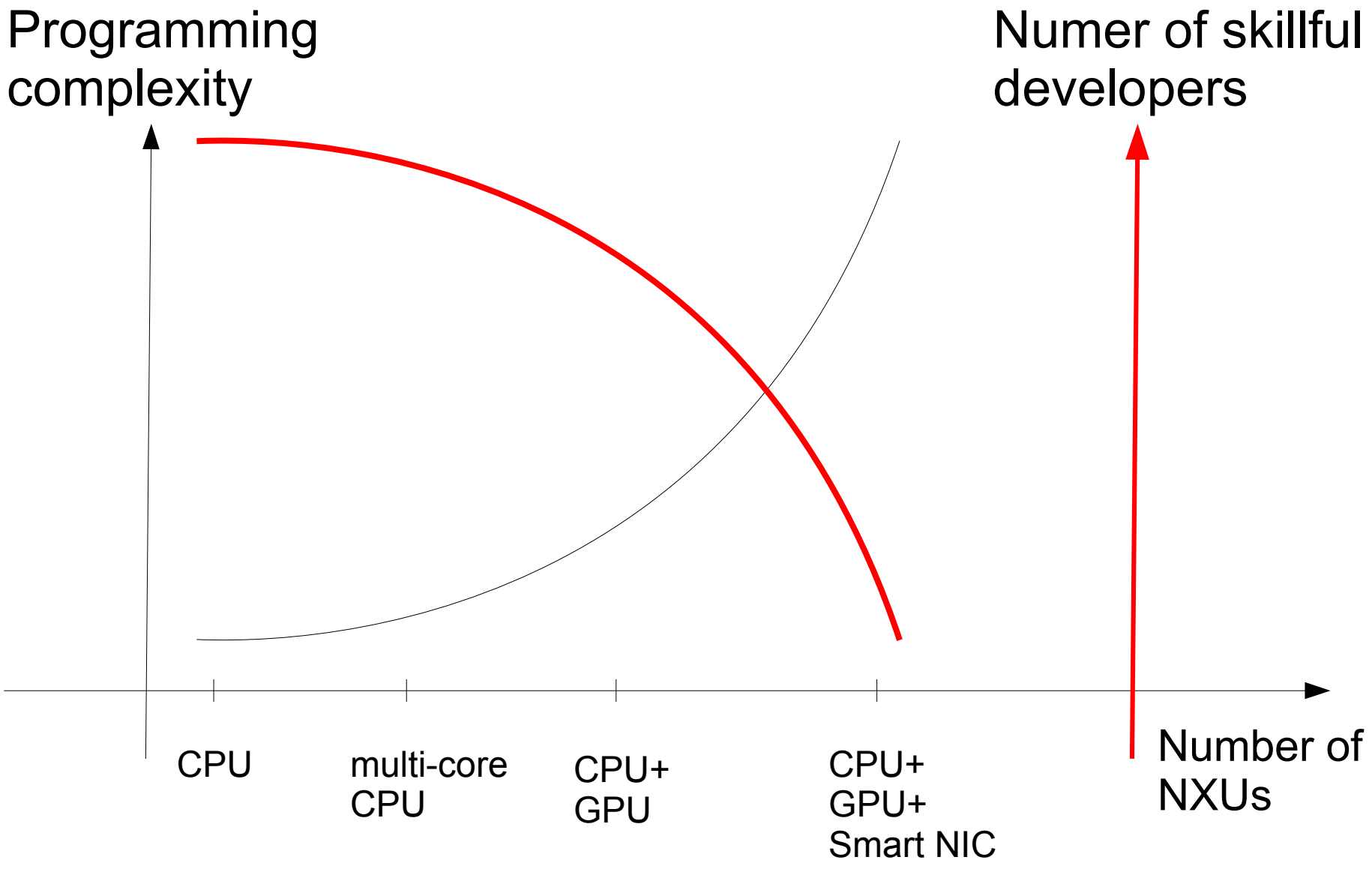
Agenda

- The root cause of the programmability wall
- OmniX: accelerator-centric OS design
- Example: I/O OS services for GPUs
- Example: Near-data computing in network adapters
- Conclusions

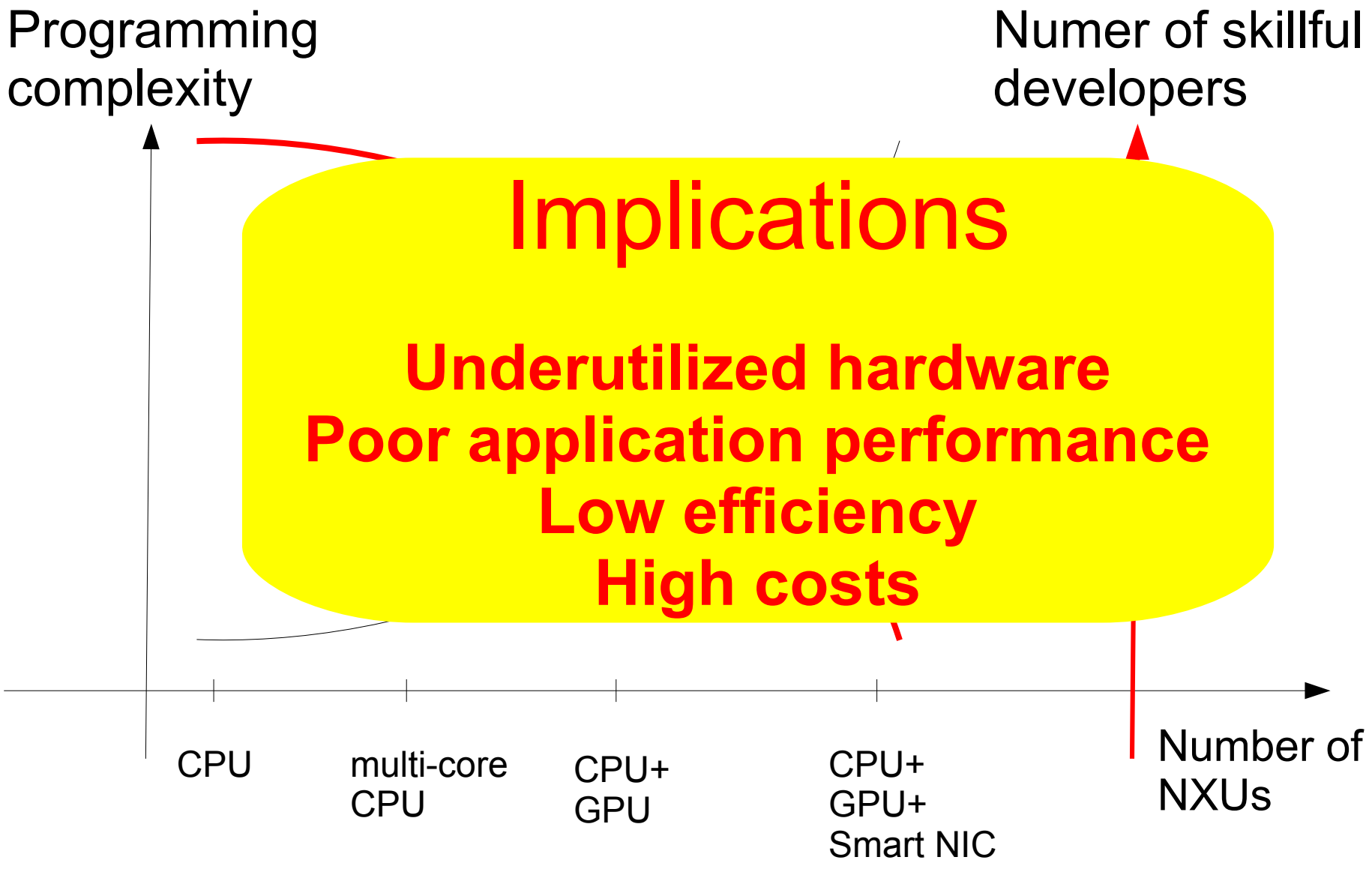
Hard to maintain whole-application efficiency



Hard to maintain whole-application efficiency

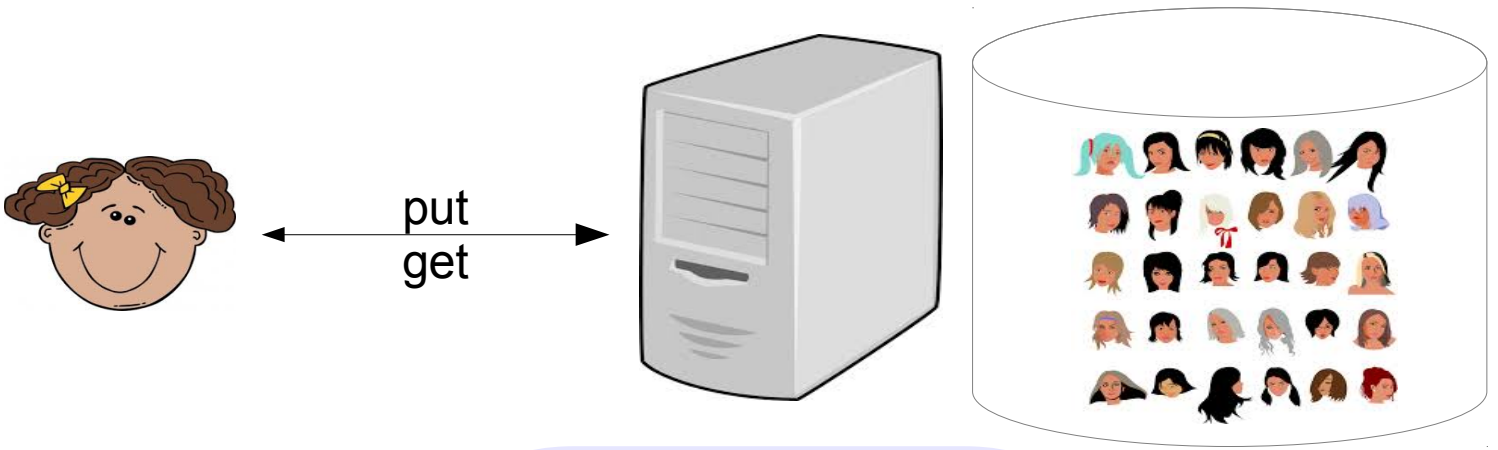


Hard to maintain whole-application efficiency



Example: image server

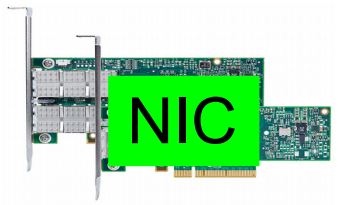
1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal

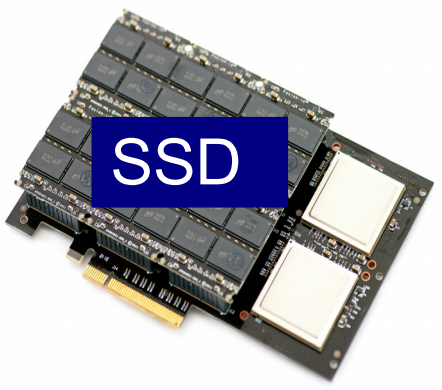


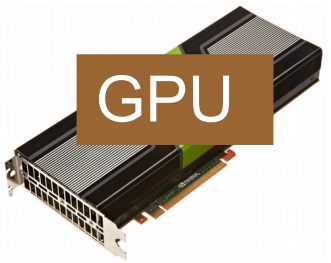
Similar architecture used in Filckr

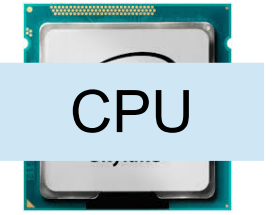
Example: image server

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



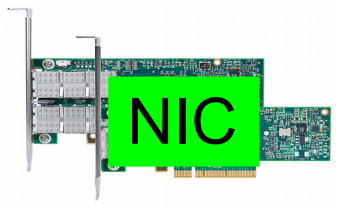


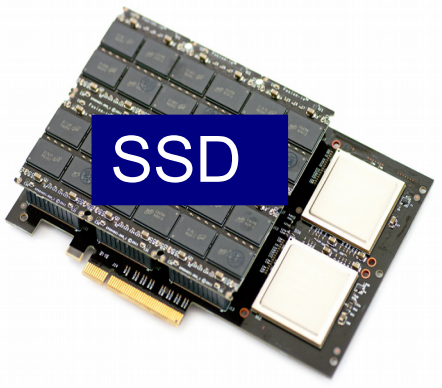


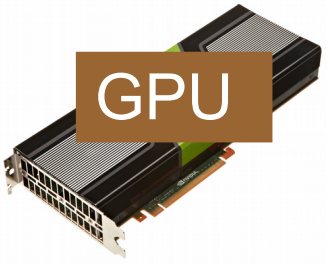


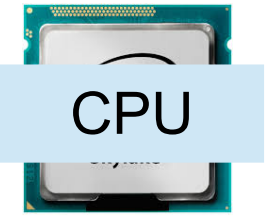
Accelerating with NXUs

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



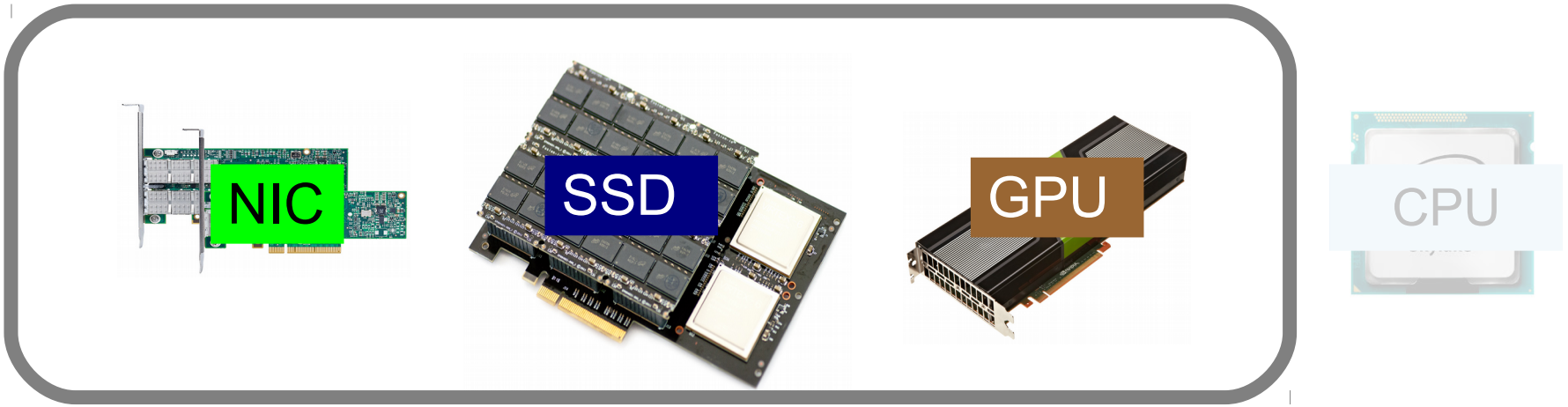






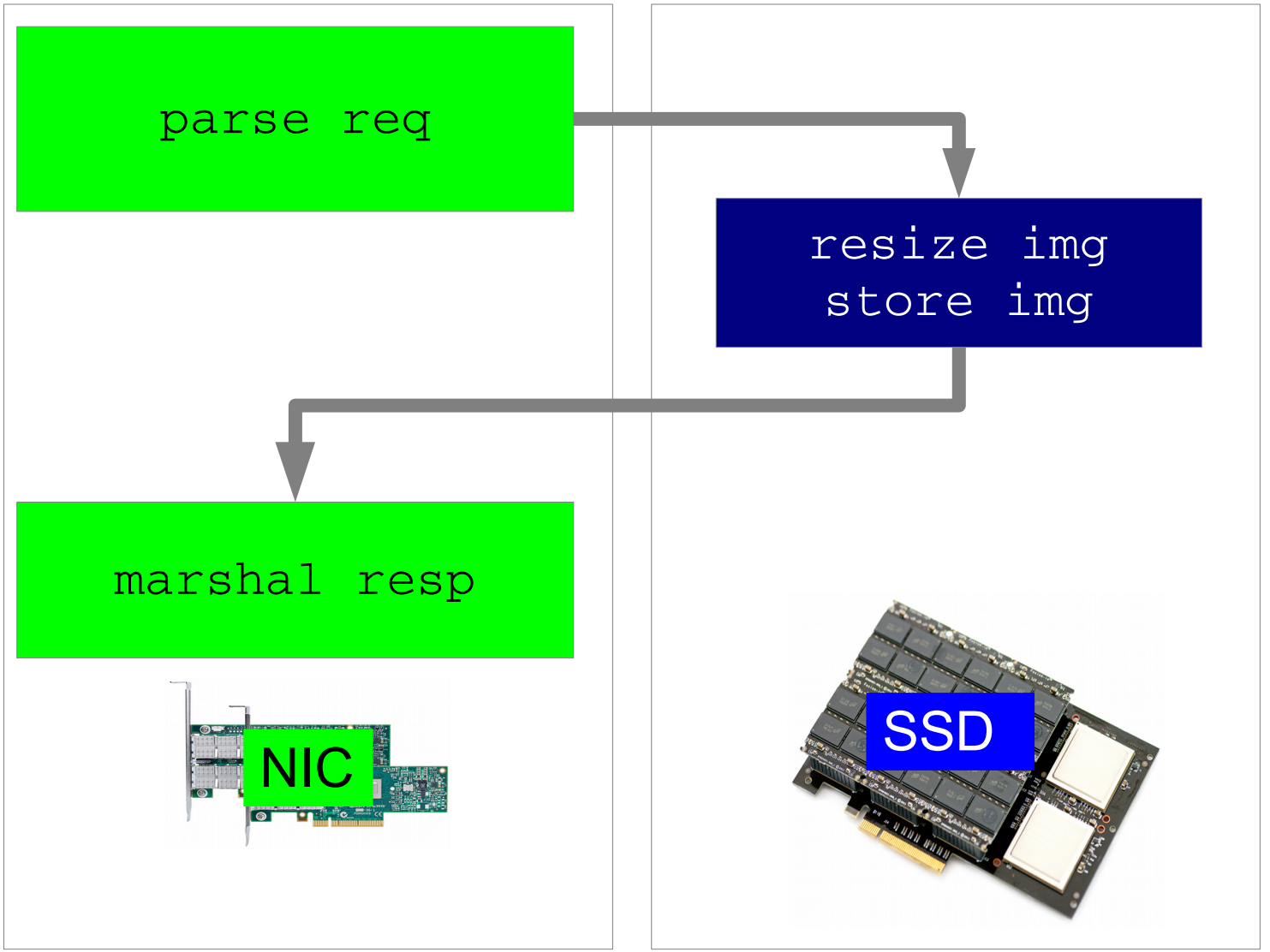
Accelerating with NXUs

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`

`recv(req)`

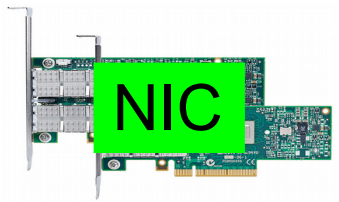
`read(file,img)`

`resize img`

`marshal resp`

`write(file,img)`

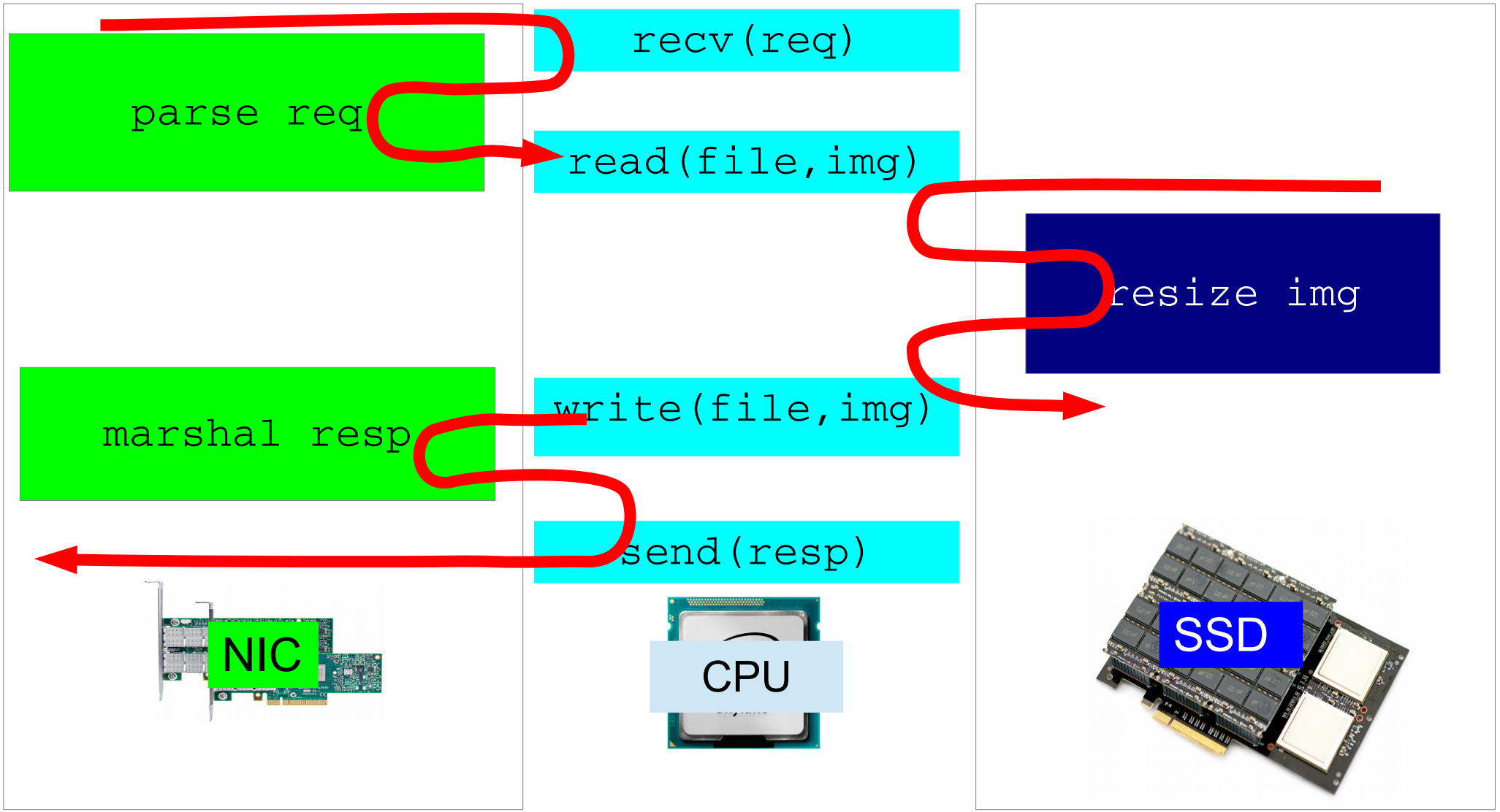
`send(resp)`



CPU

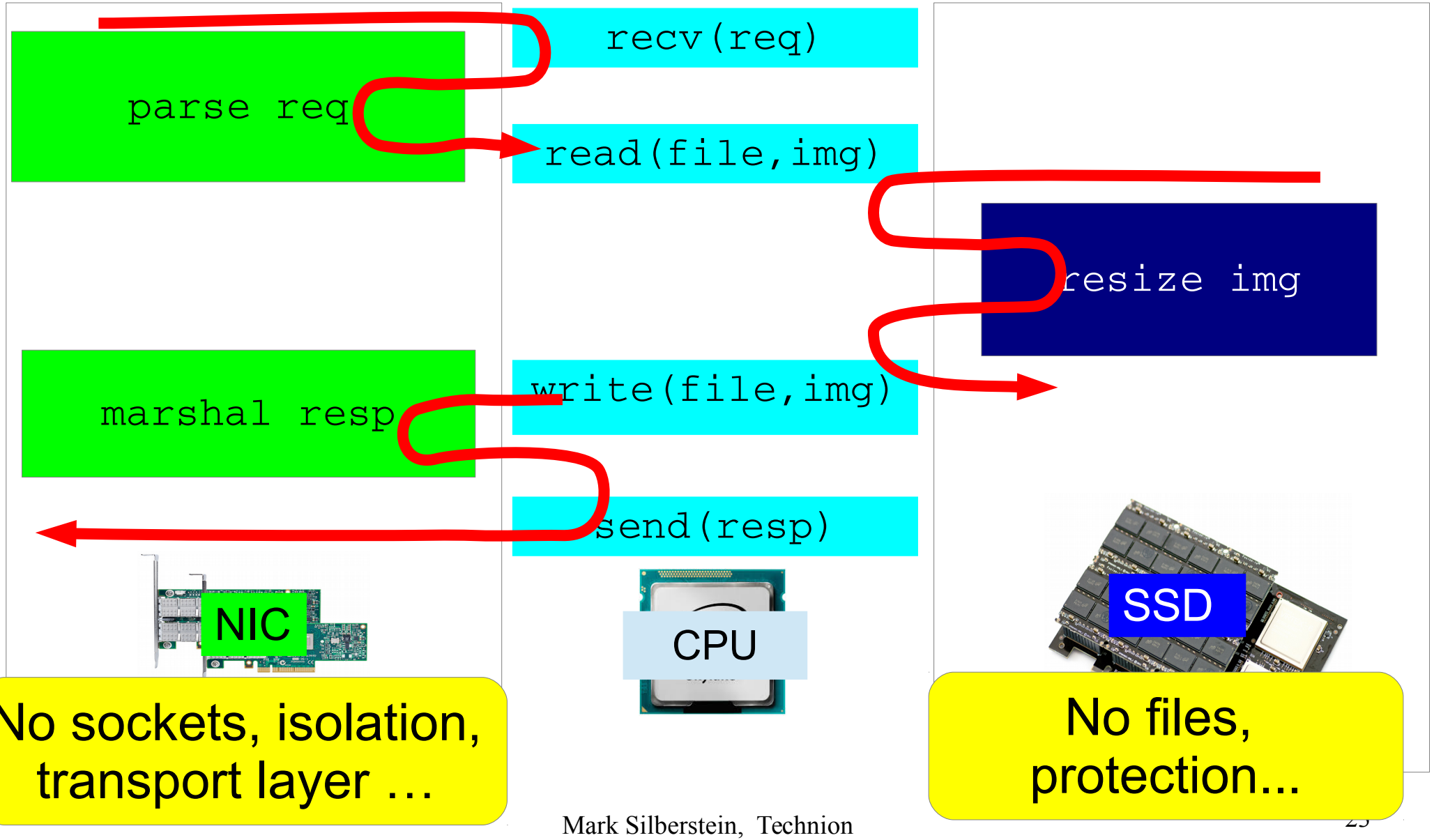
Result: offloading overheads dominate

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

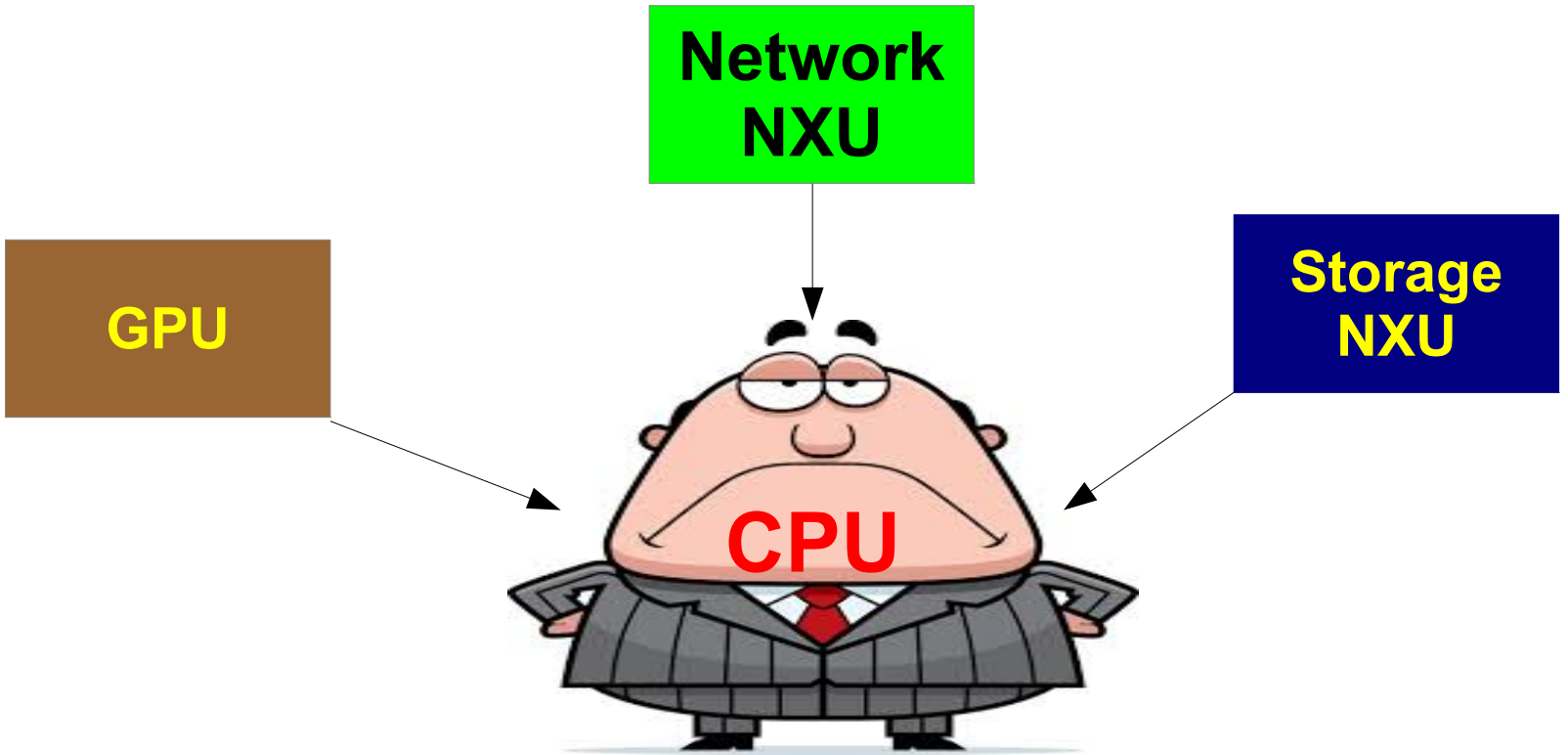


Result: offloading overheads dominate

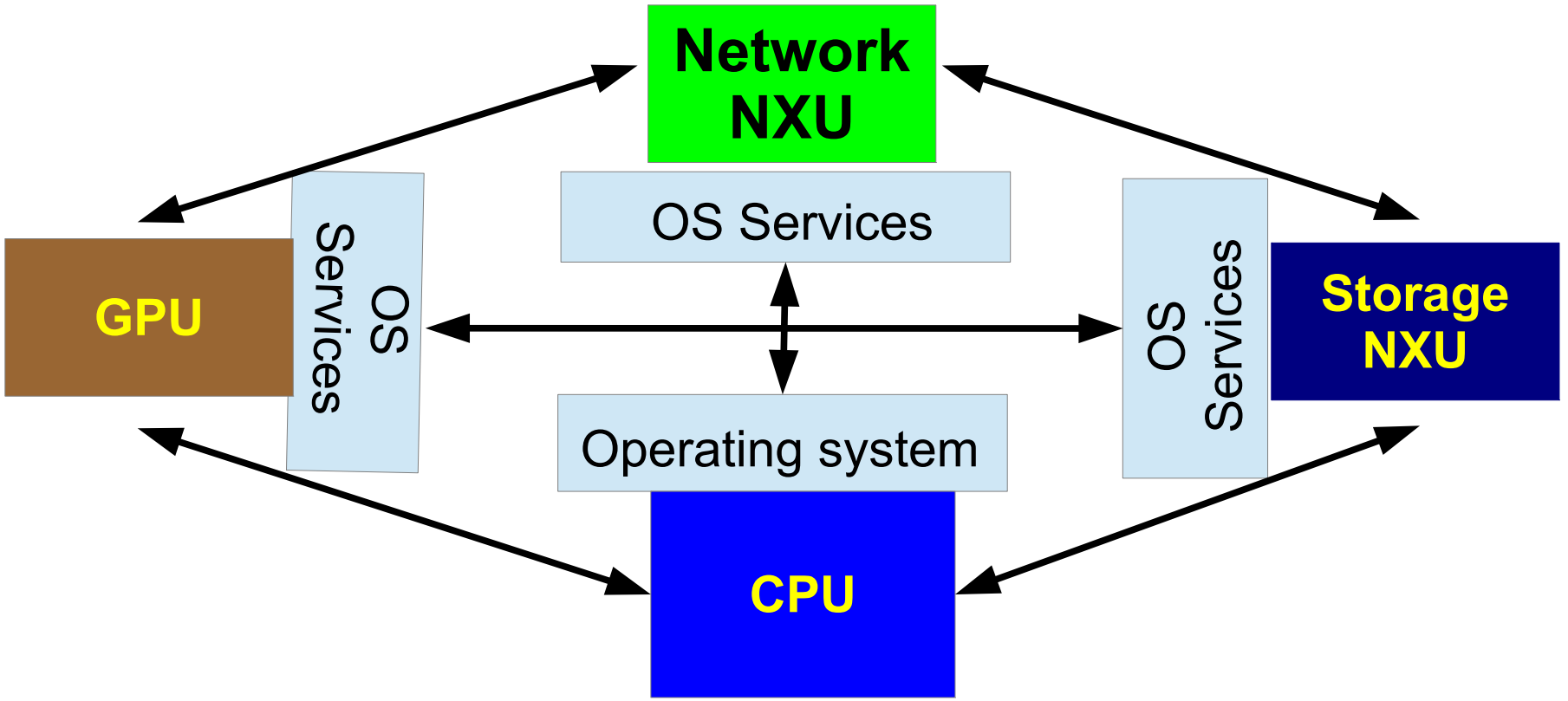
get: parse → **resize** → **store** → marshal



THE problem:
OS architecture is CPU - centric

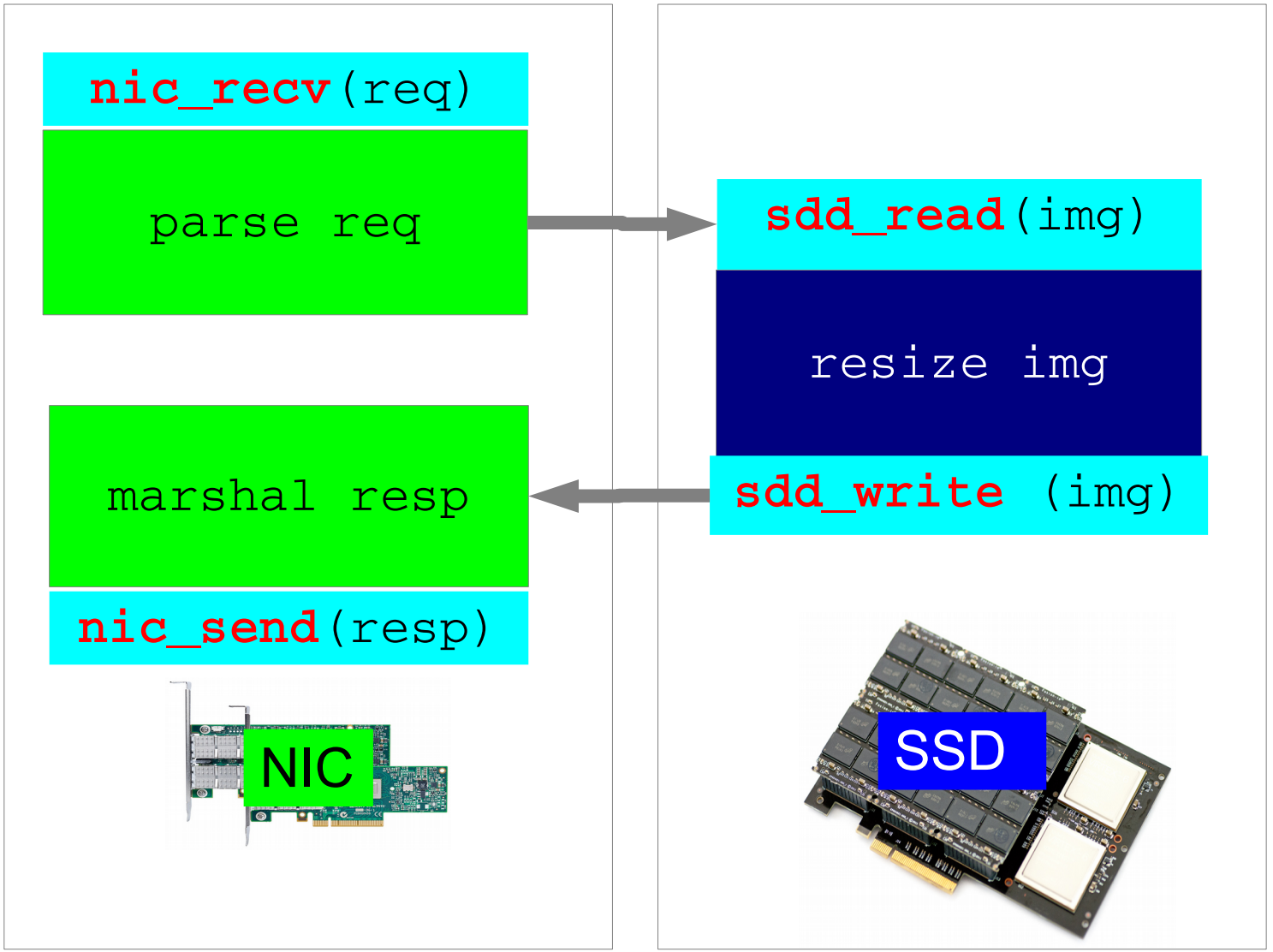


OmniX: accelerator-centric OS architecture



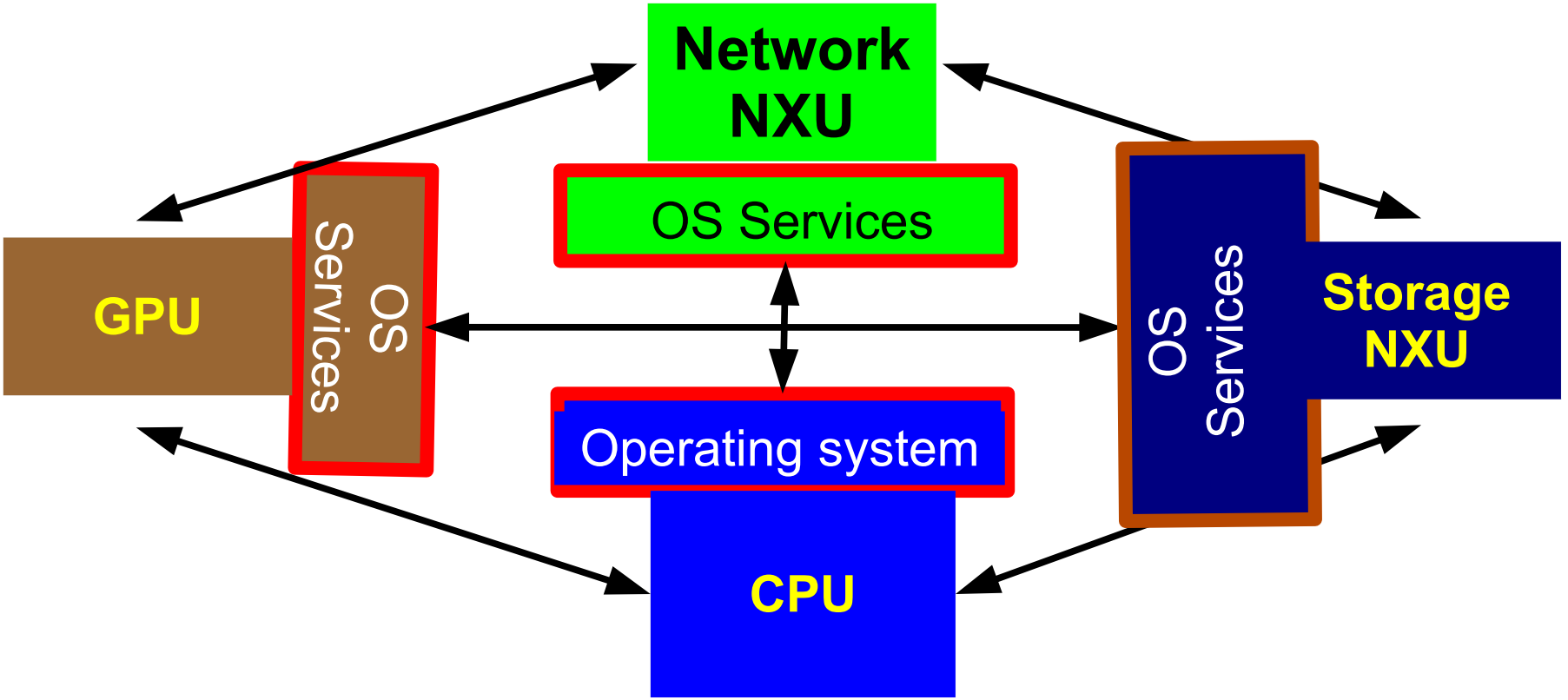
Wouldn't it be lovely?

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



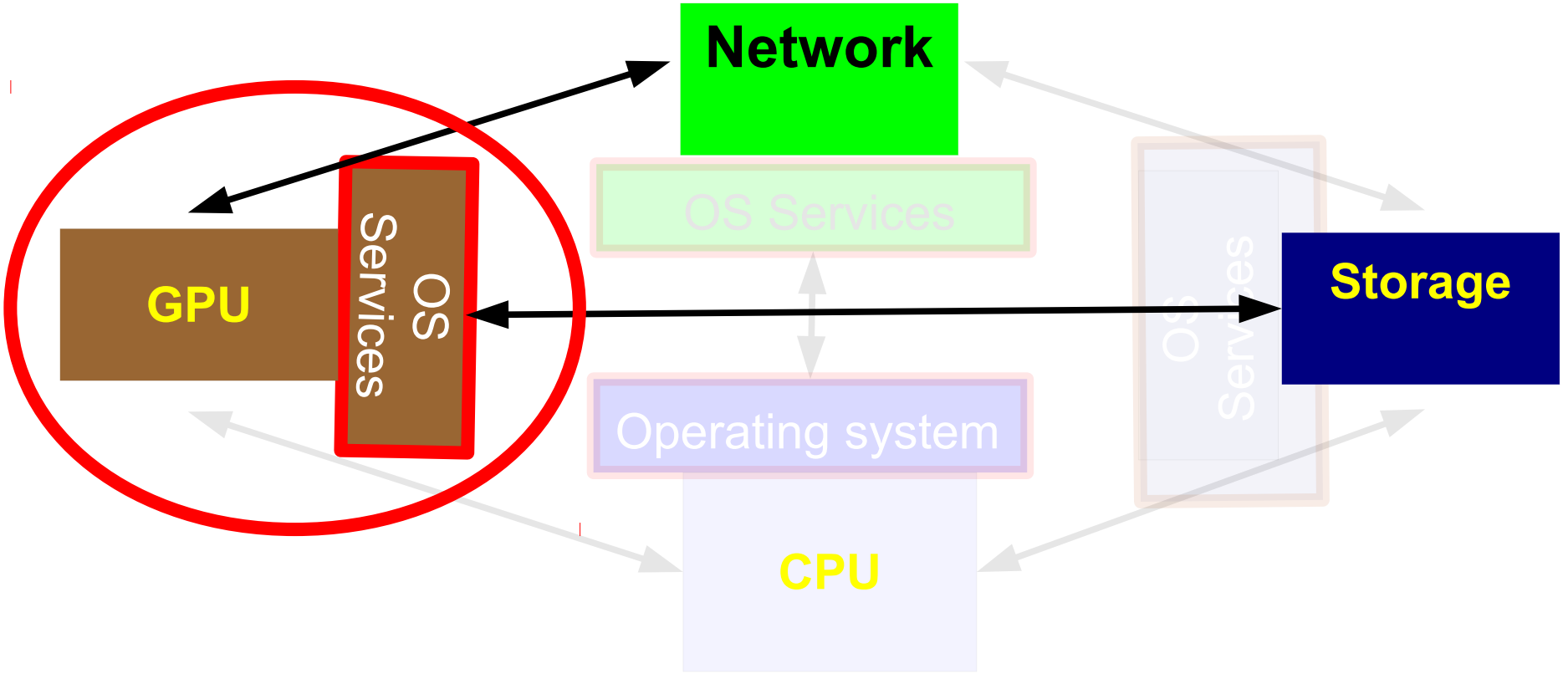
OmniX OS design principles

1. A single application OS



Each processor runs an optimized library of OS services

1. A single application OS



Each processor runs an optimized library of OS services

Challenges

- GPUs run hundreds of thousands of threads!

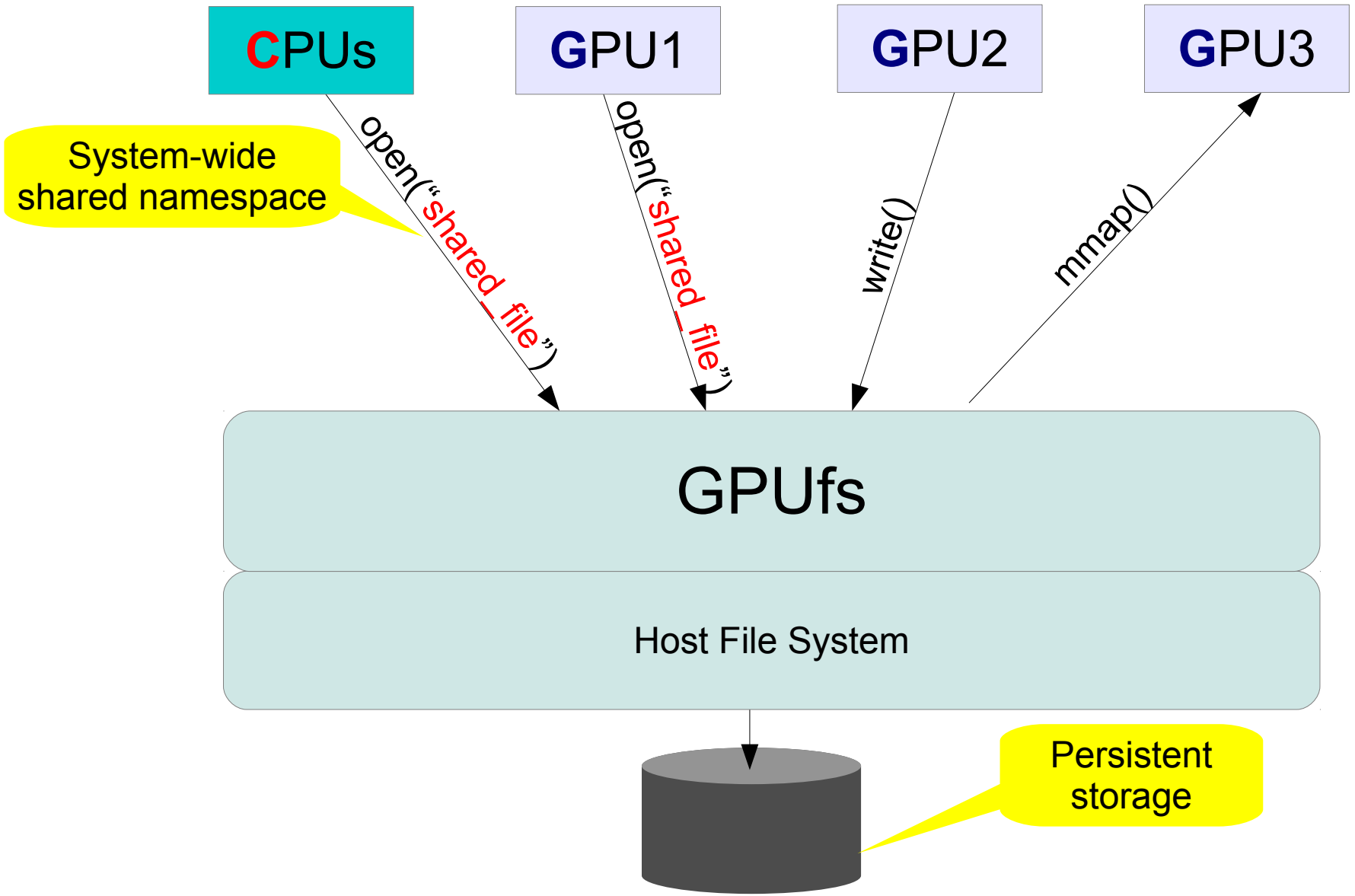
  → Require new OS Interface semantics
- GPUs have distinct NUMA properties

  → Require new locality-optimized OS design
- GPUs cannot run privileged code

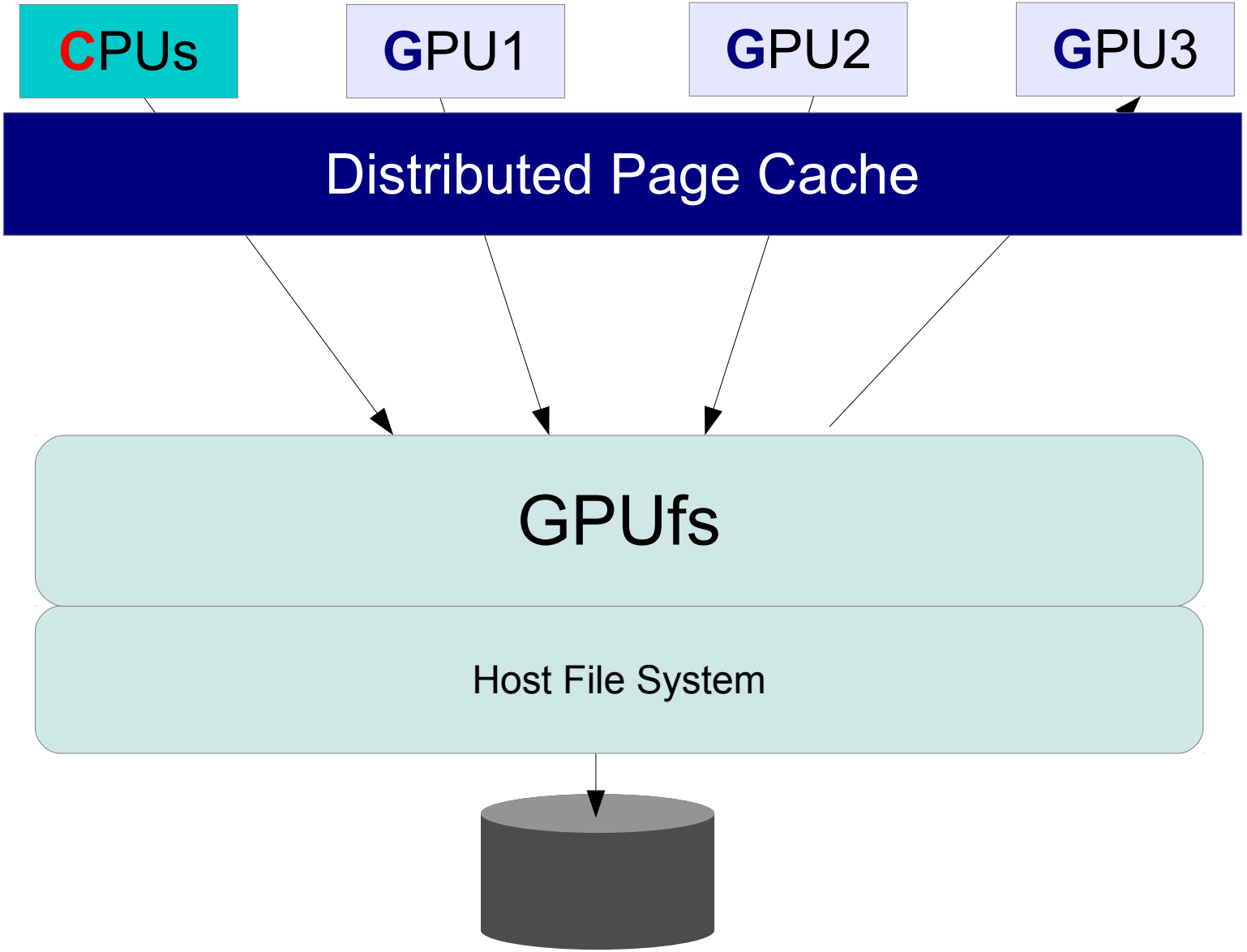
  → Require CPU assistance / peripheral support

GPUfs: File system library for GPUs

ASPLOS13, TOCS14, CACM15, ISCA16, SYSTOR16

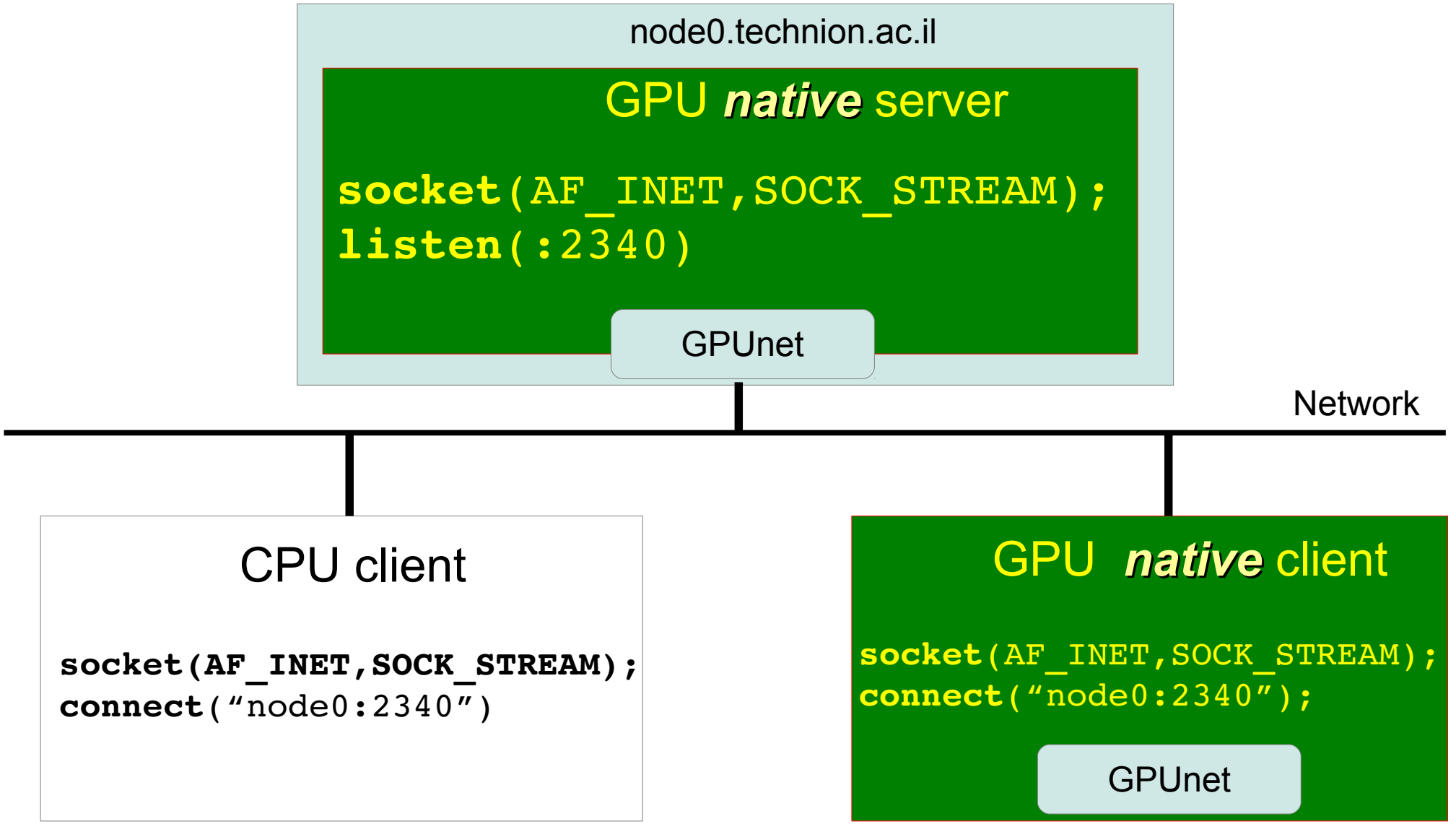


Transparent locality optimization

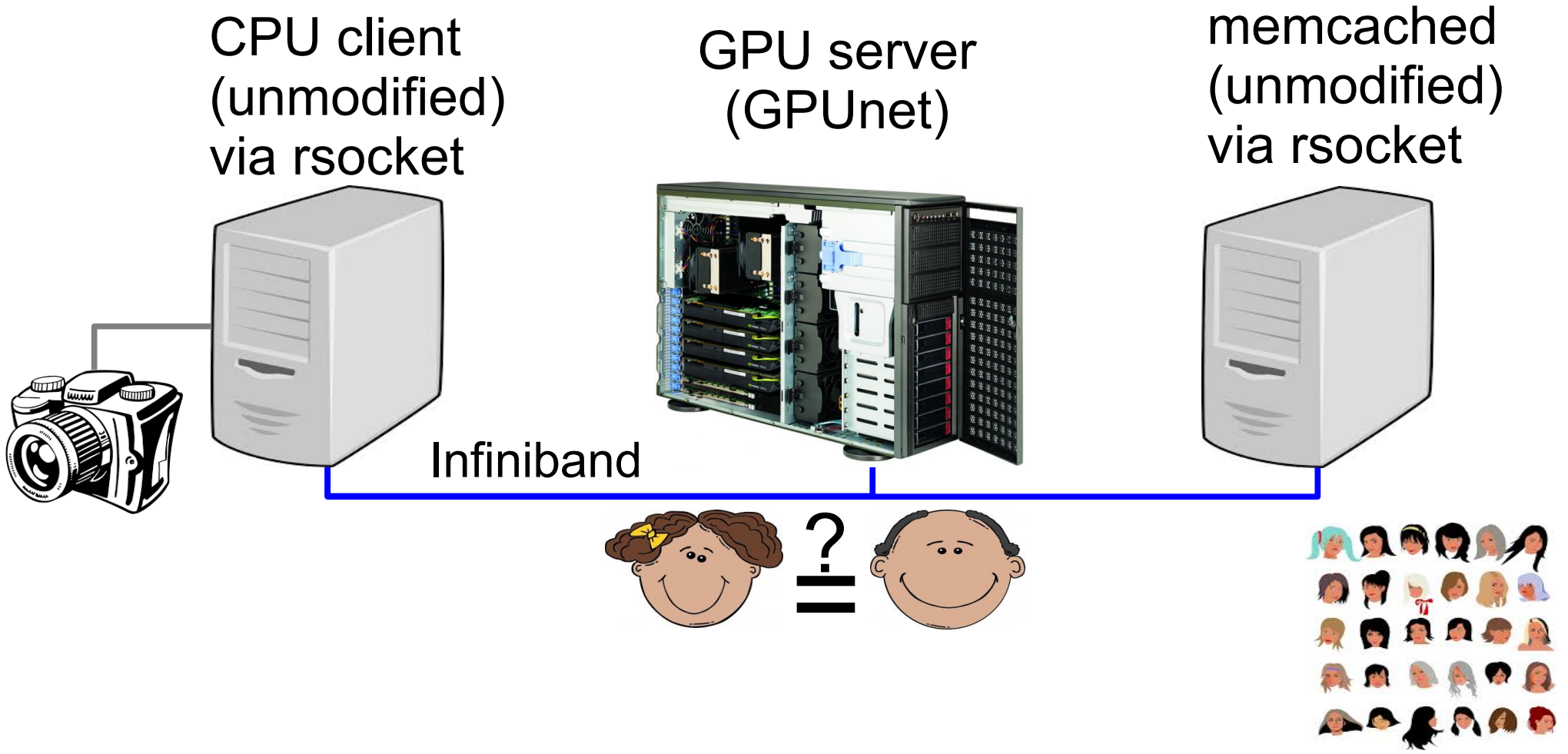


GPUnet: Network library for GPUs

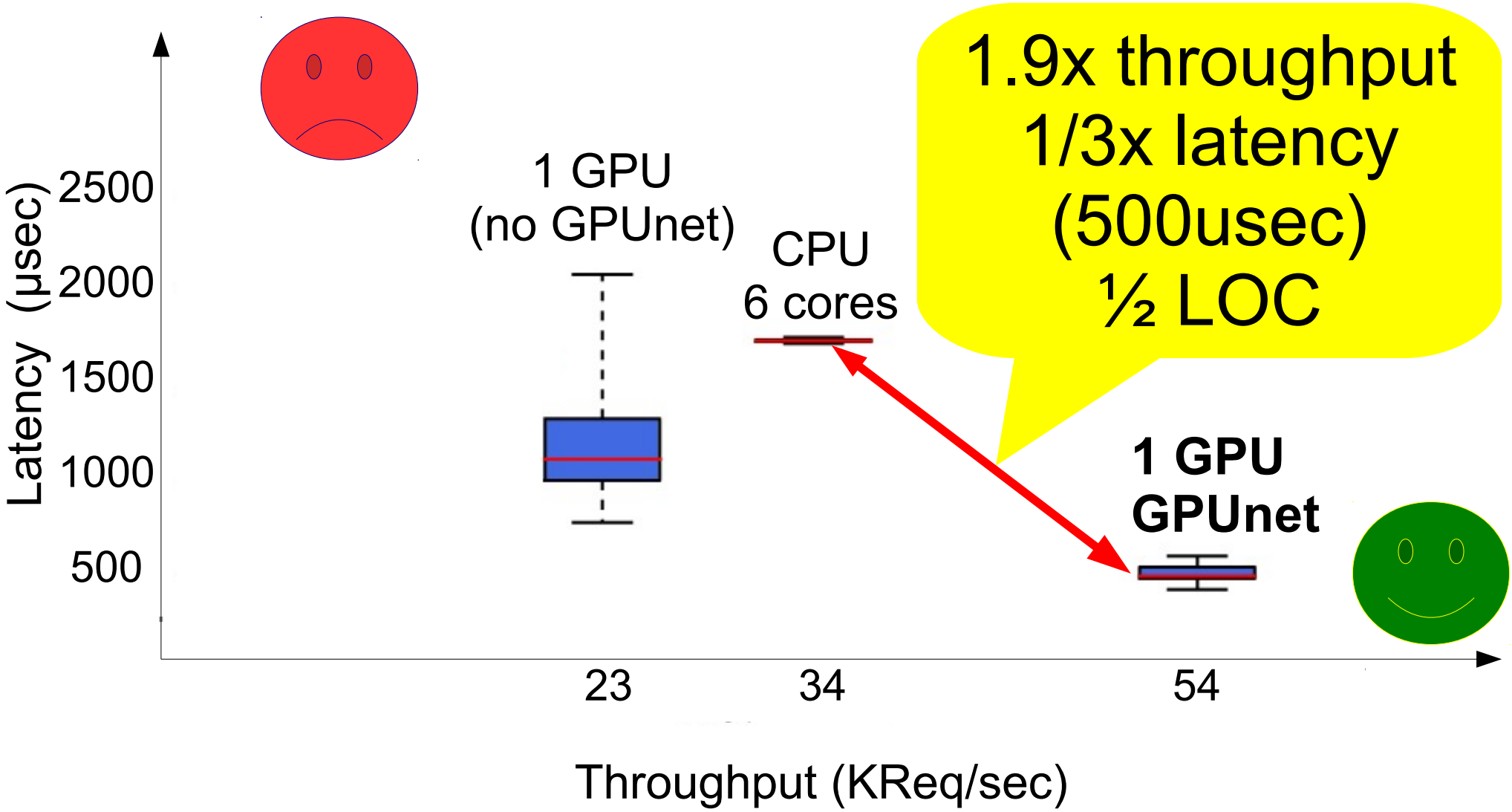
OSDI14, TOCS16



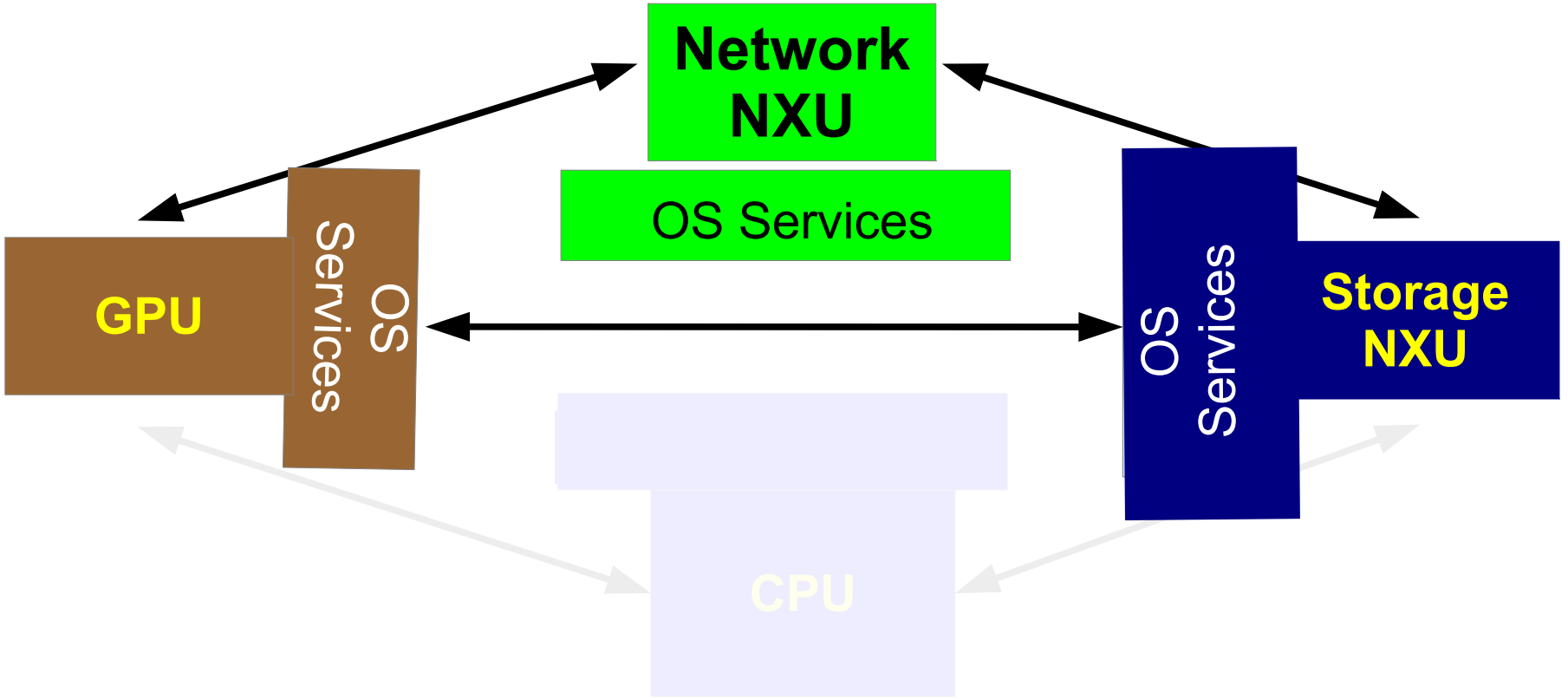
Face verification server



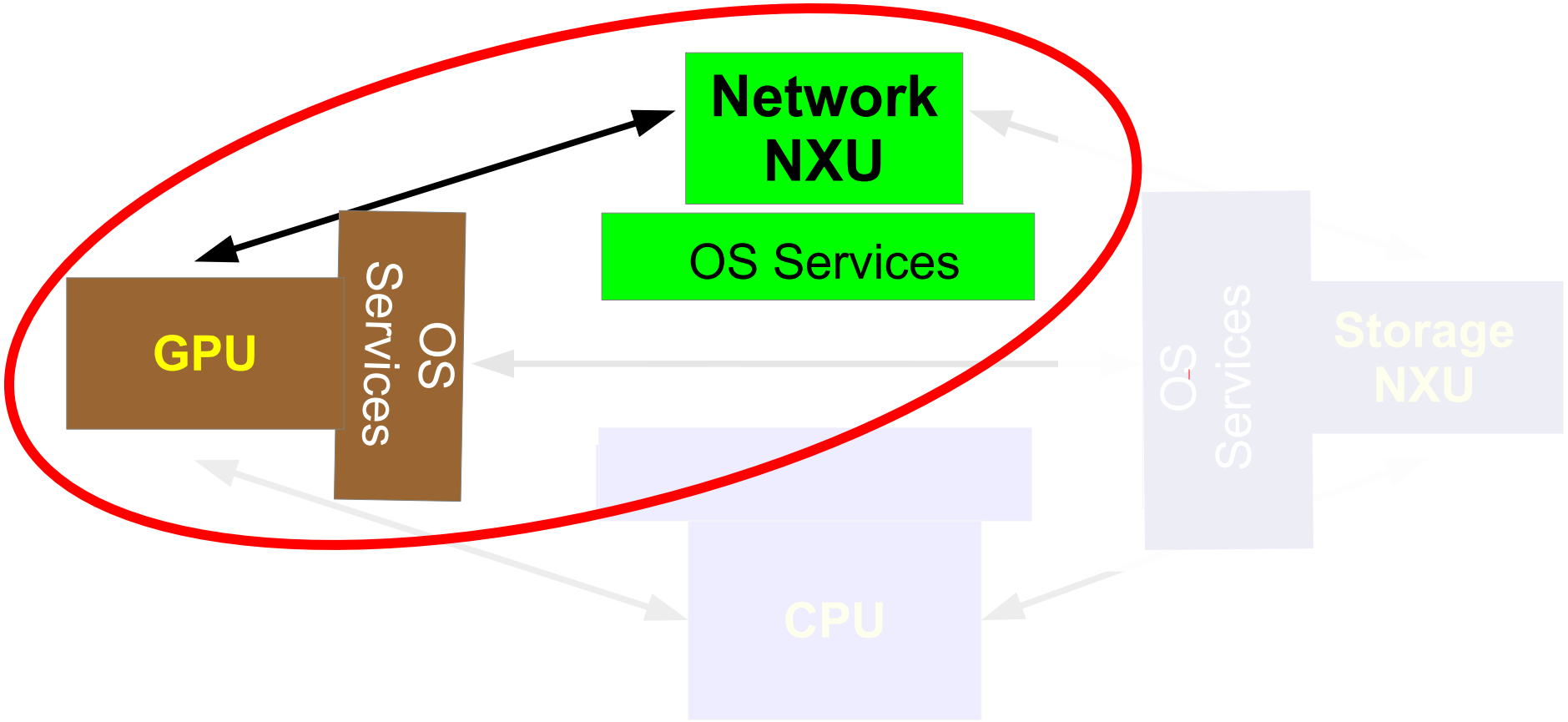
Face verification: Different implementations



2. Exclude the **CPU** from control and data path



2. Exclude the CPU from control and data path



GPUrdma: direct control of network adapter from GPUs

ROSS16

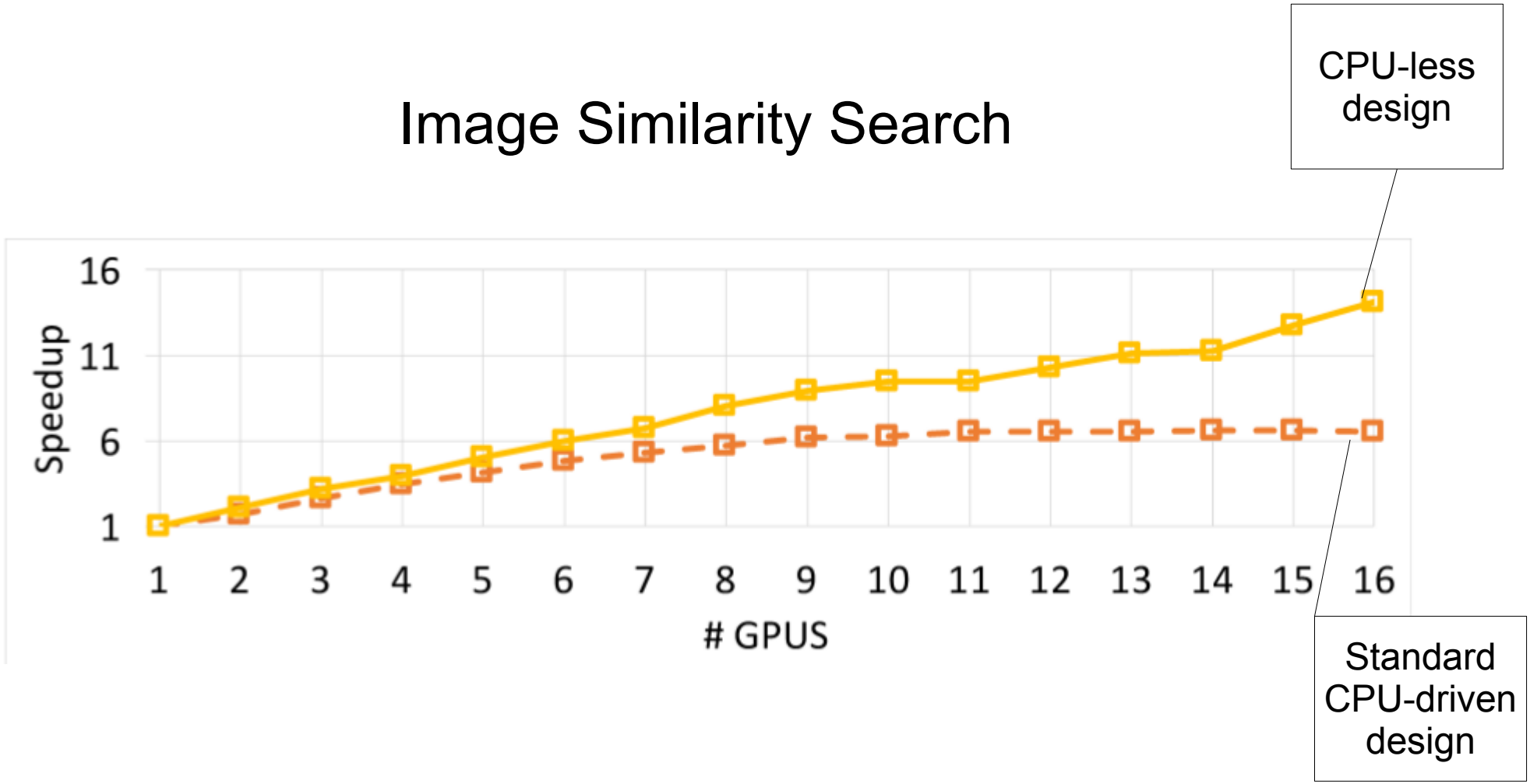
GPU-to-GPU roundtrip latency via Infiniband

GPUnet (CPU-mediated): 50 usec

GPUrdma (NIC controled by GPU): 10 usec

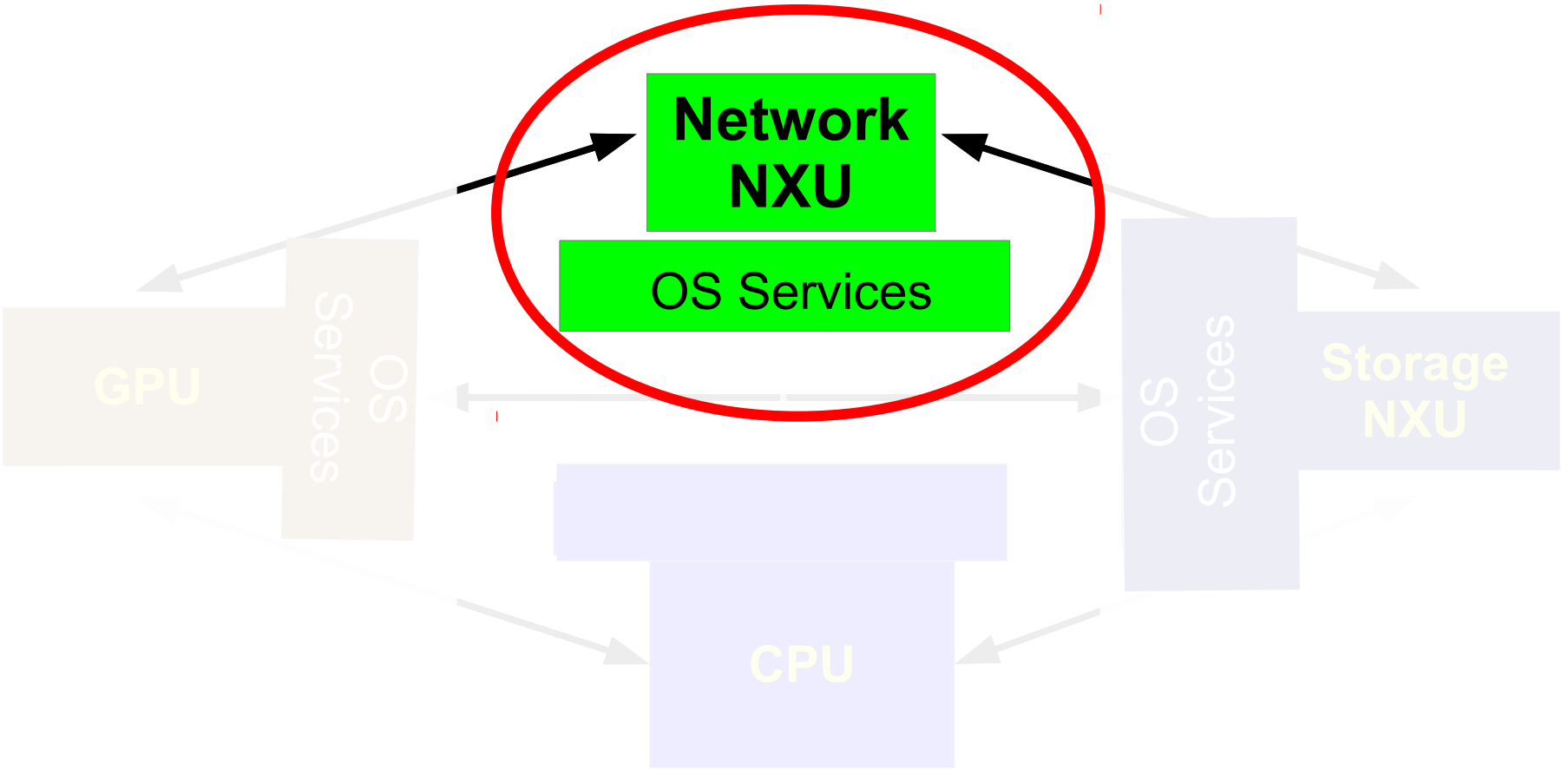
CPU-less design: lower latency

GPUpipe: CPU-less network server



CPU-less design: much better scaling

3. Support for near-data processing

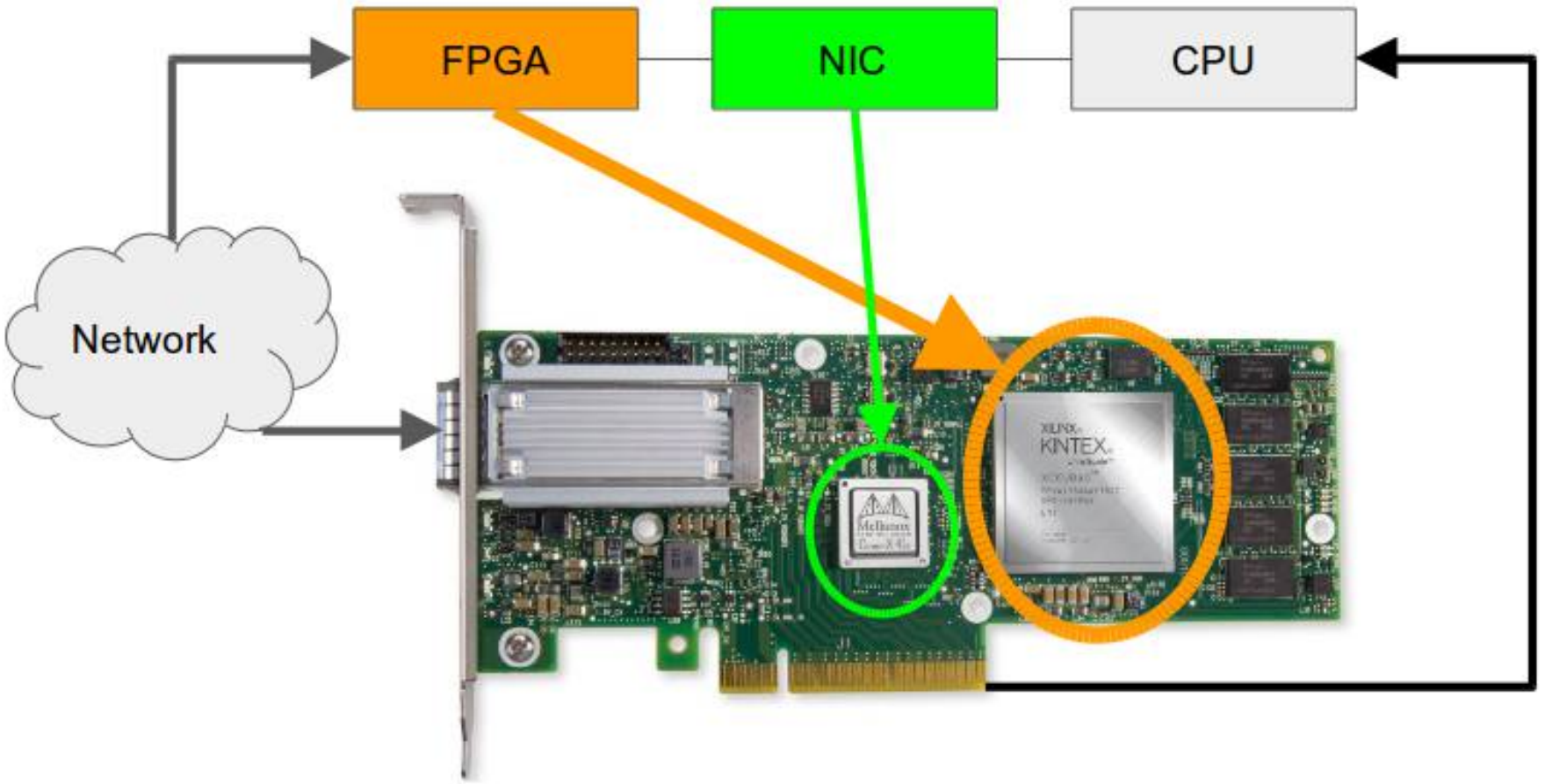


Goals

- Low, predictable latency
- Alleviate memory bandwidth bottlenecks
- High power efficiency
- Free CPU (and its caches) for other tasks

These goals are essential for data center applications

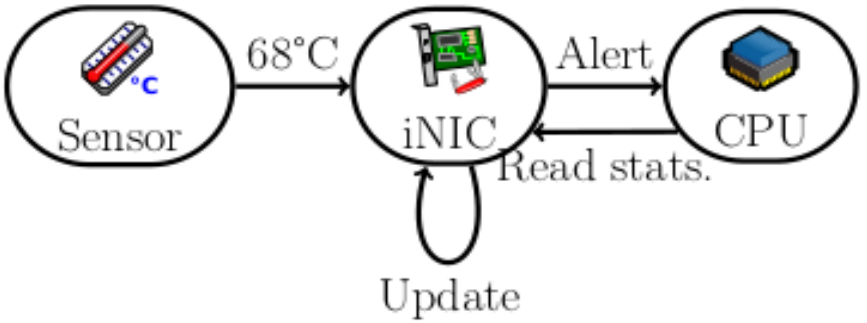
Smart Network Adapters



On-the-fly processing at wire speed!

Typical application scenario: sensor monitoring

- Receive measurements
- Update statistics
- Alert on **critical** events
- Retrieve statistics

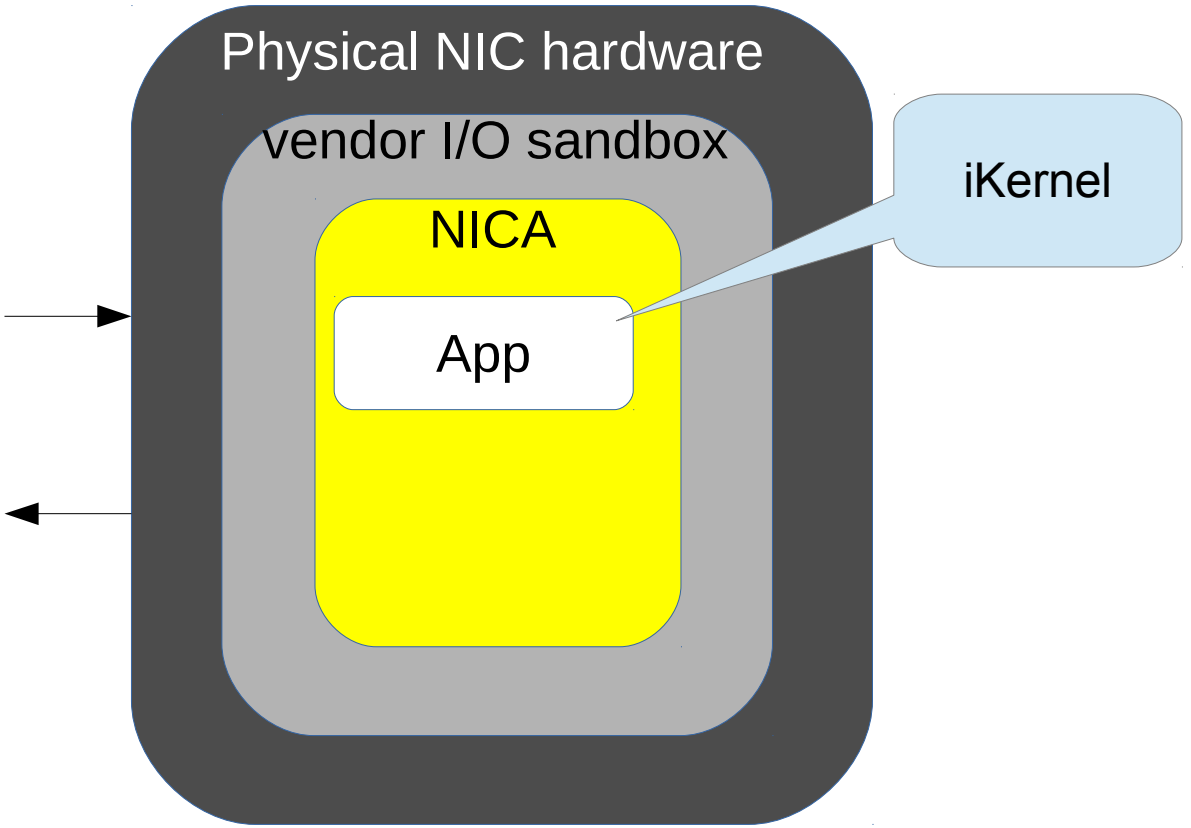


Uses CPU only for rare events

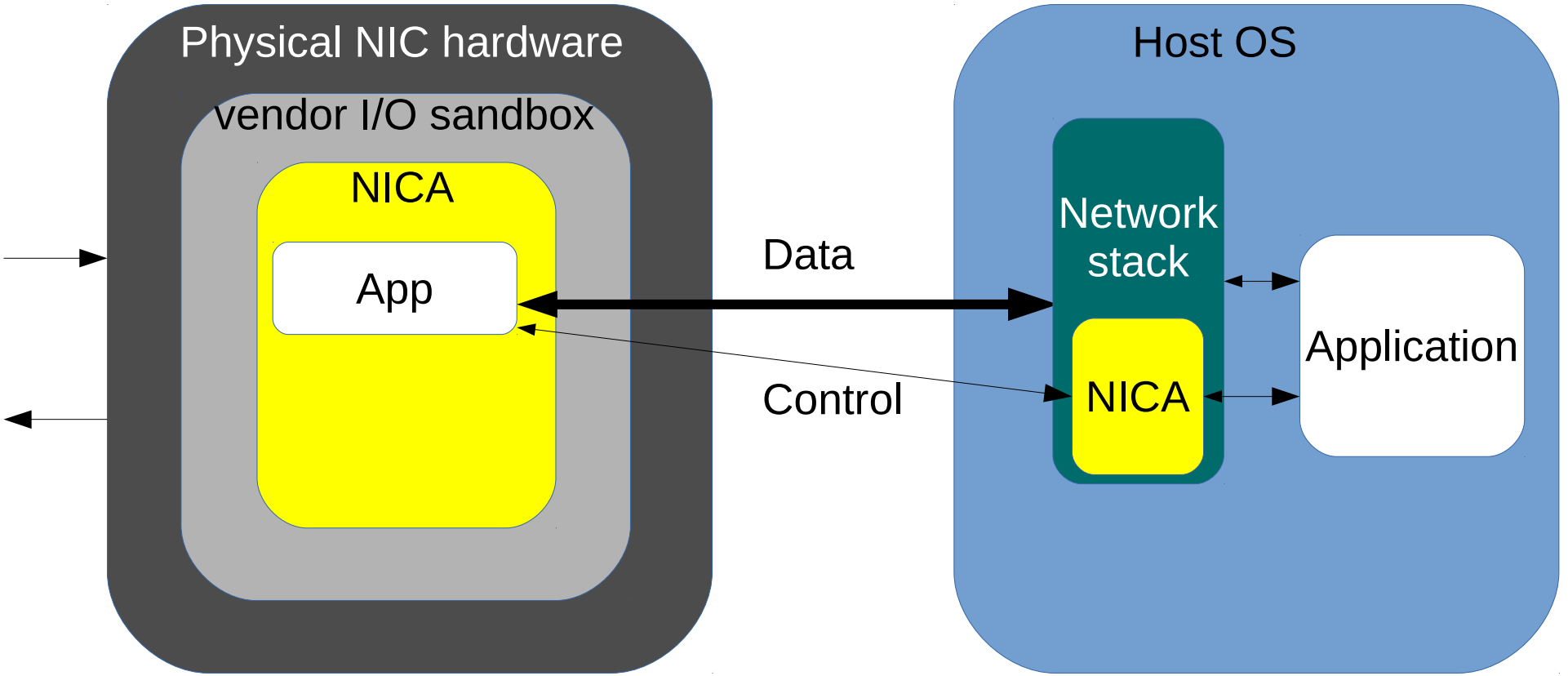
NICA
Network applICation Acceleration

- Programming model
- In-NIC programming and development
- OS integration

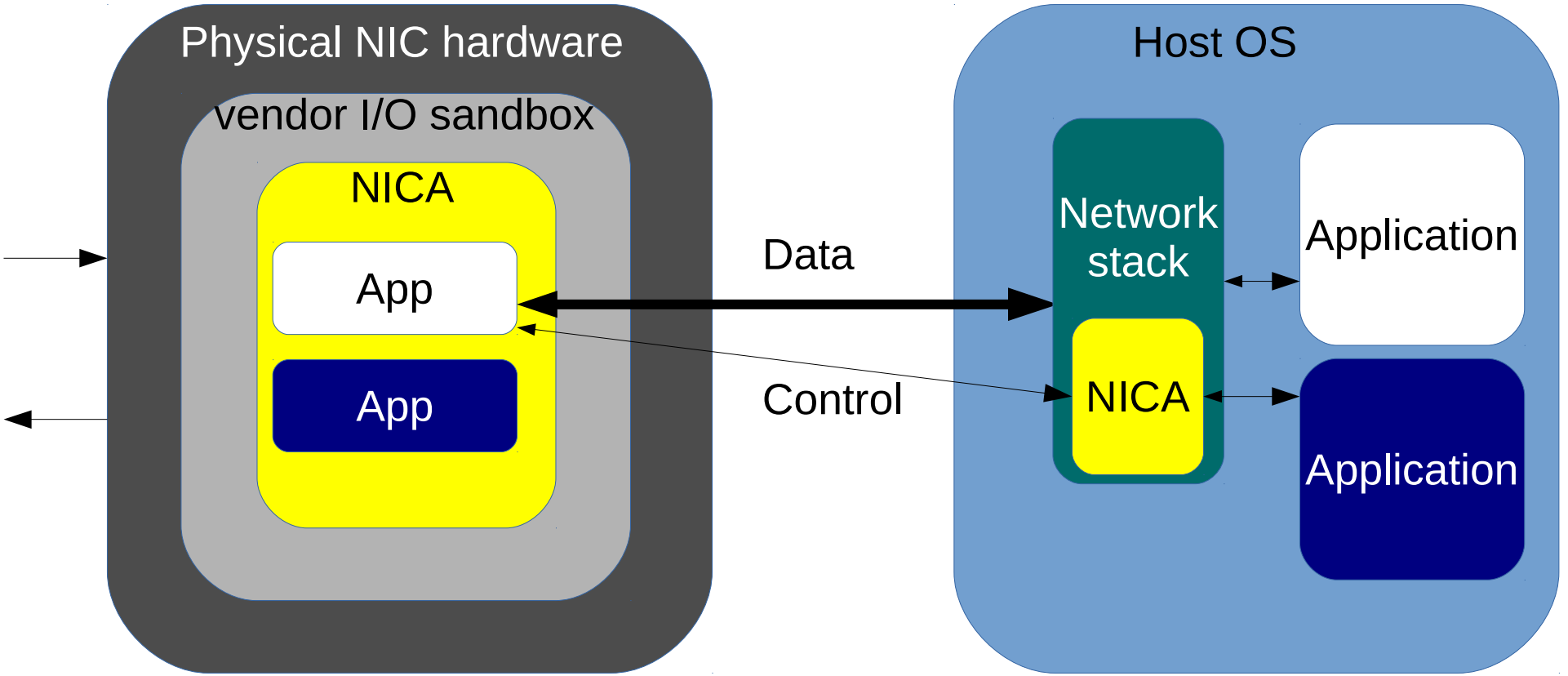
High-level design



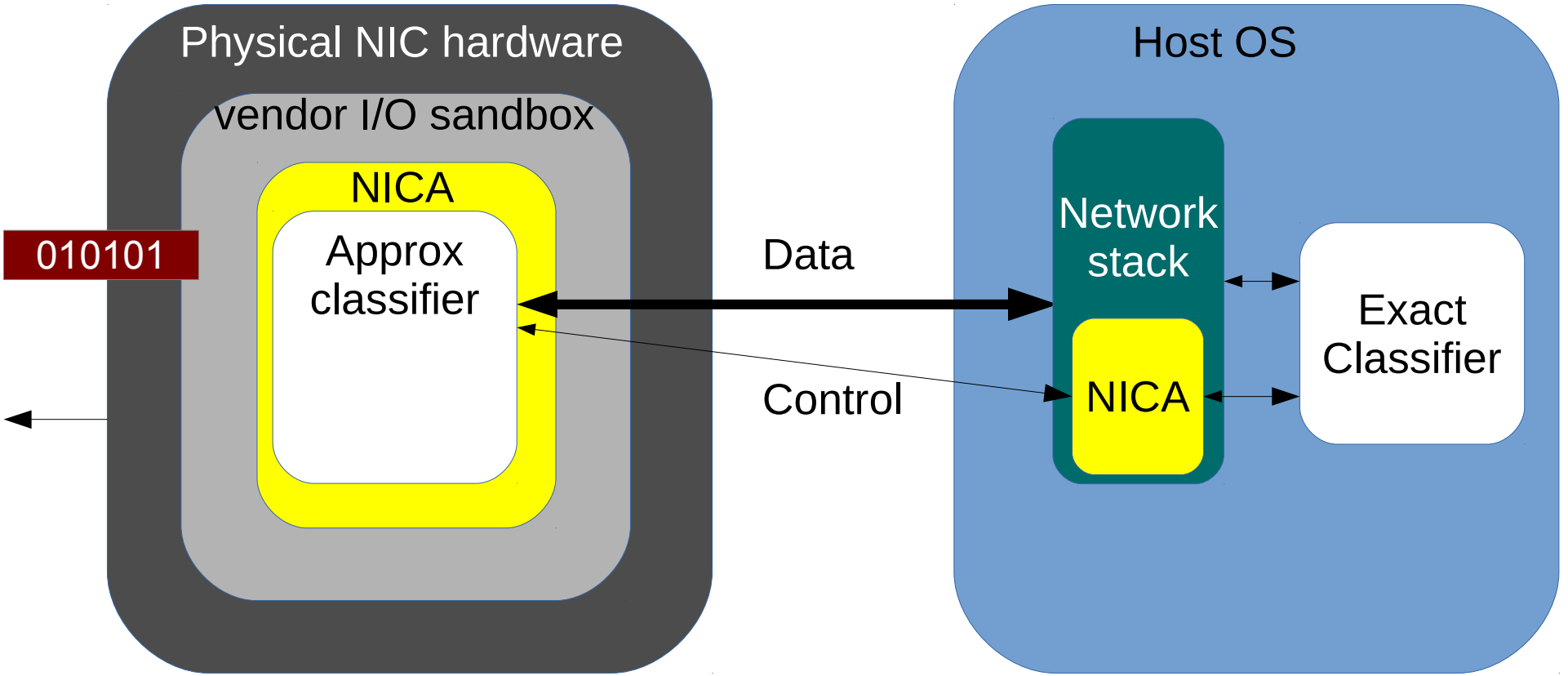
High-level design



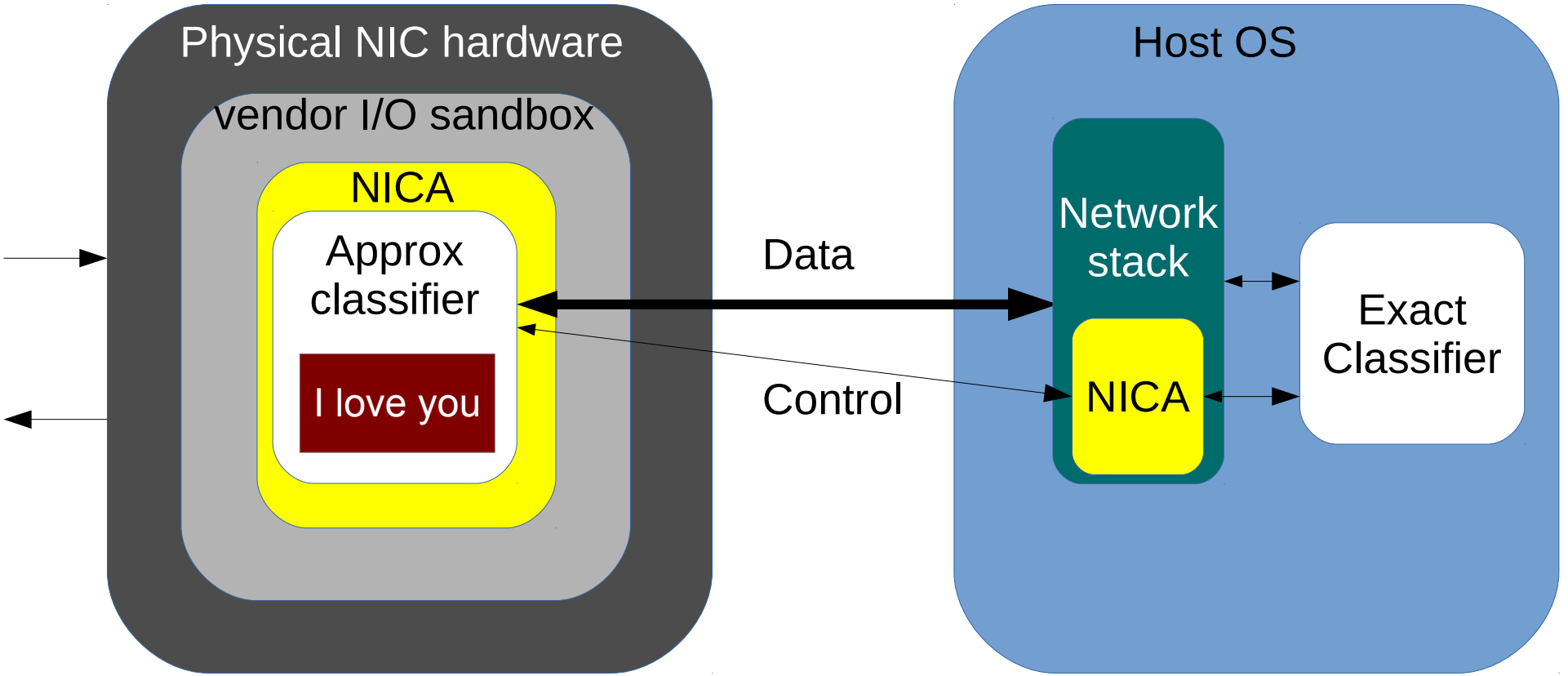
High-level design



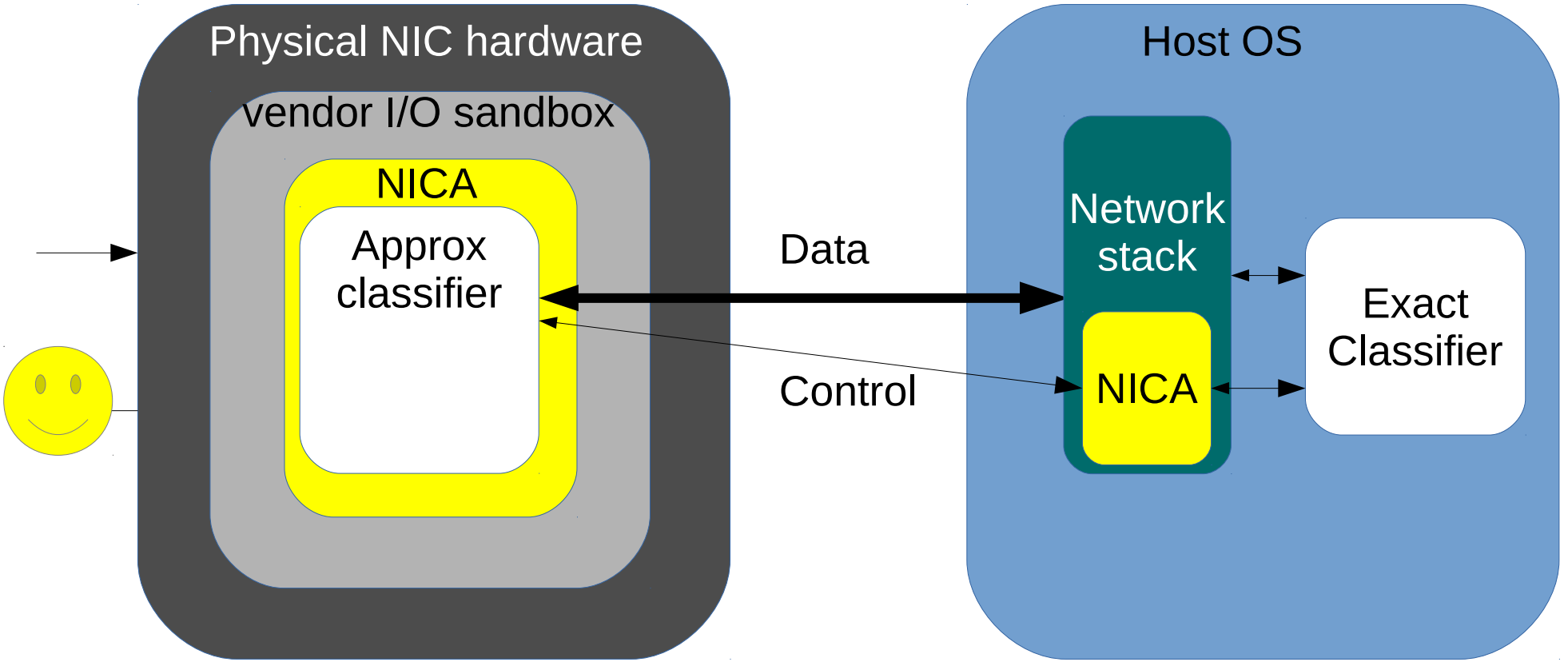
Example: Tweet sentiment analysis



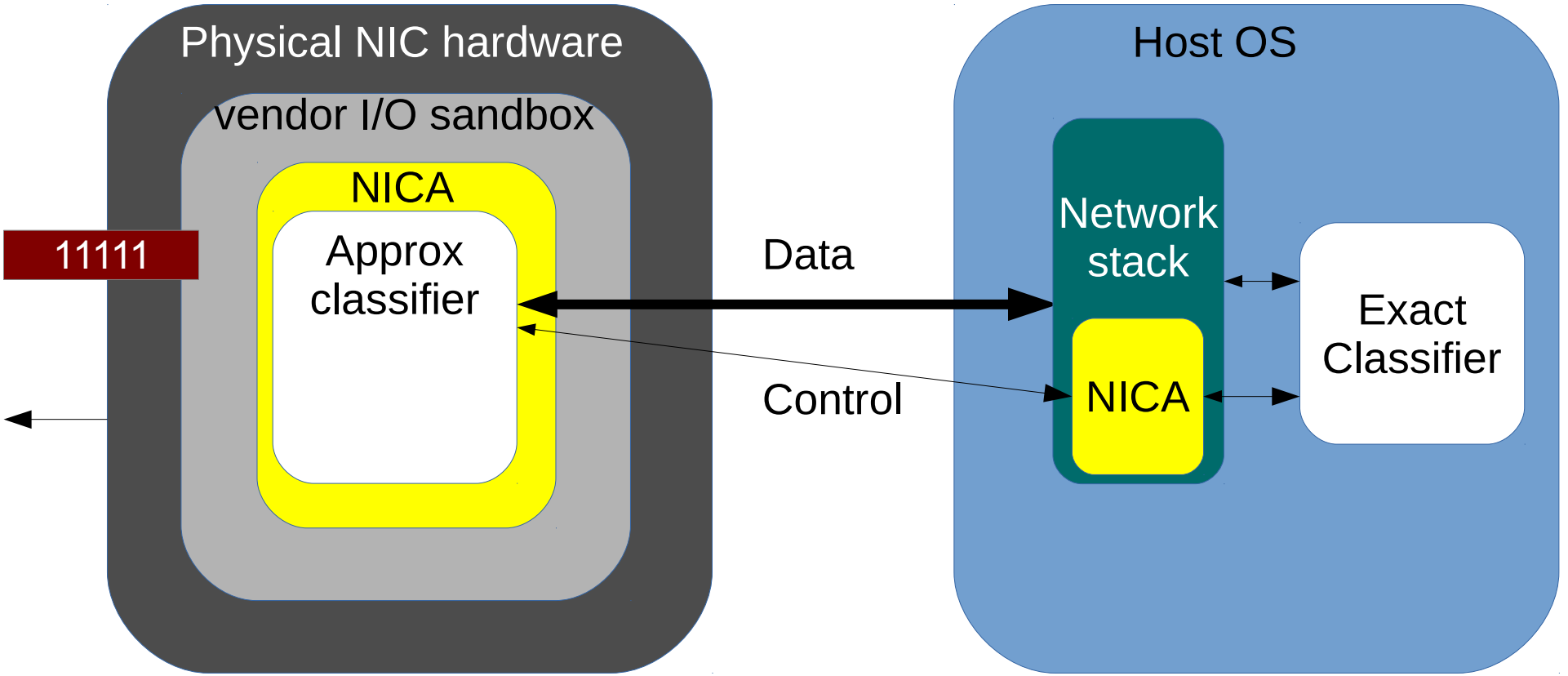
Example:
Tweeter sentiment analysis



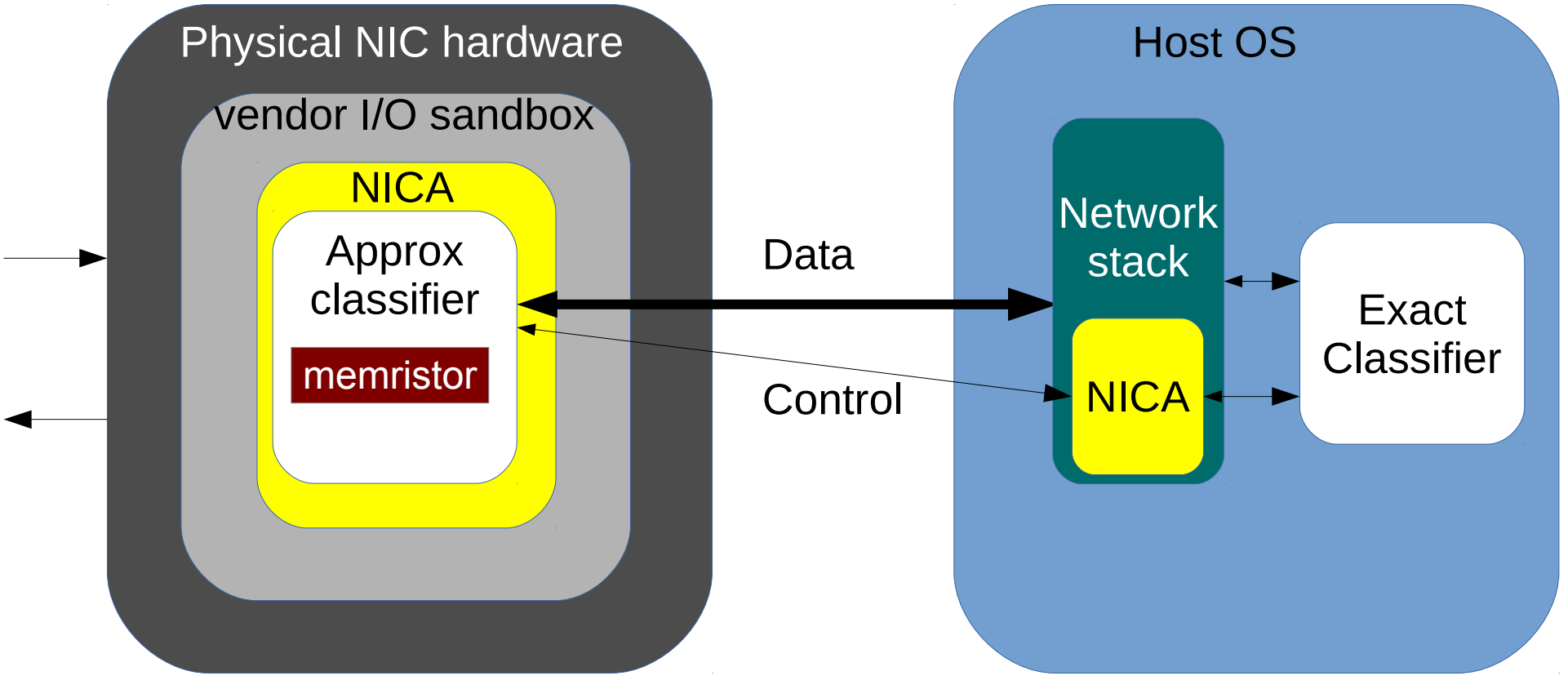
Example:
Tweeter sentiment analysis



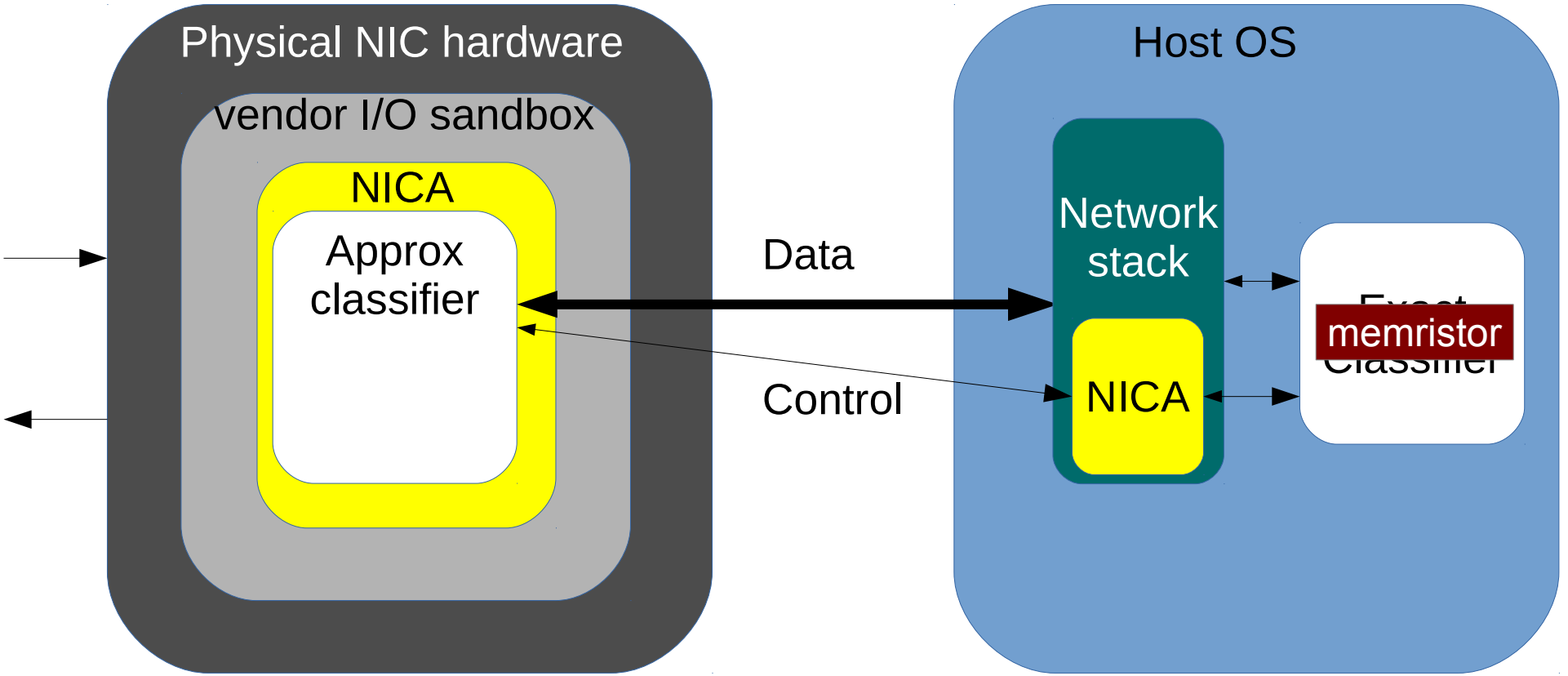
Example:
Tweeter sentiment analysis



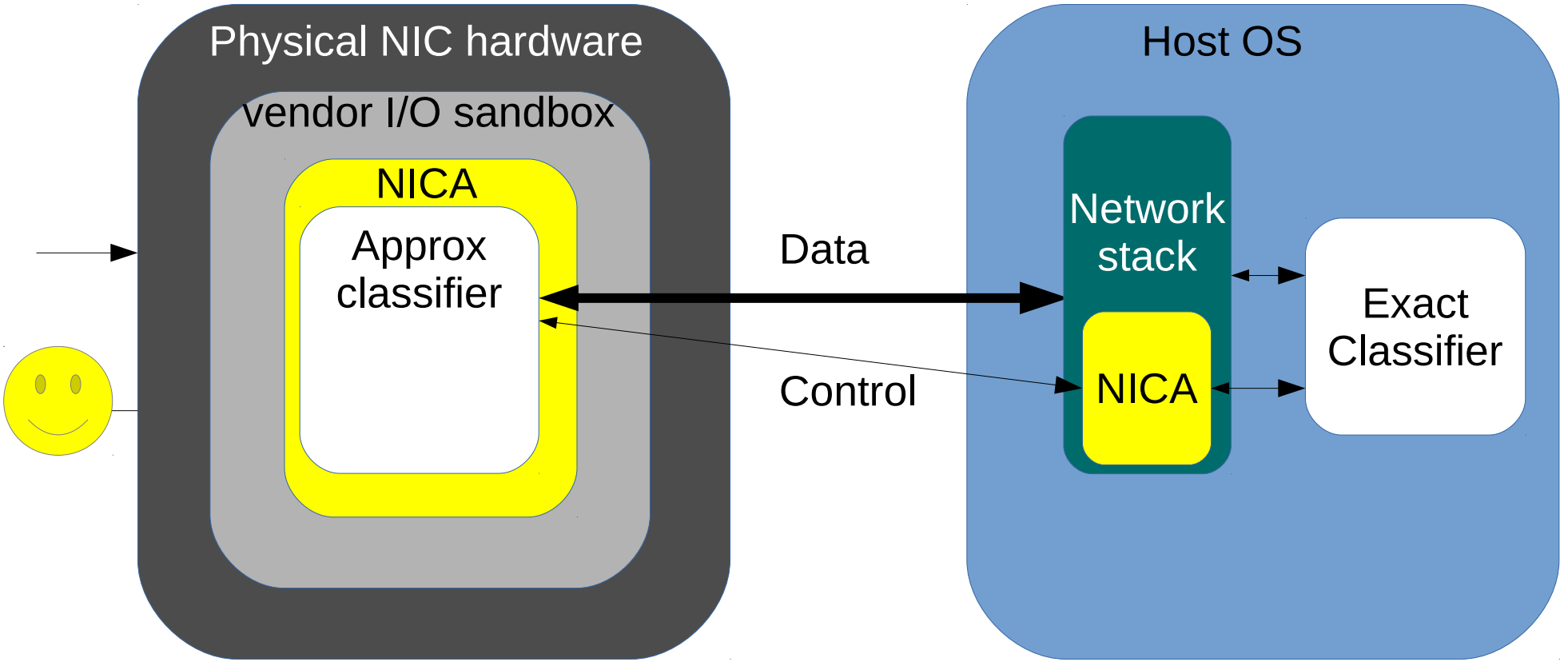
Example:
Tweeter sentiment analysis

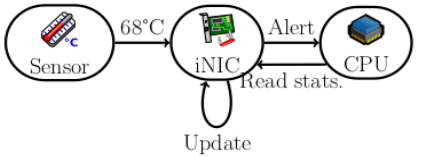


Example:
Tweeter sentiment analysis

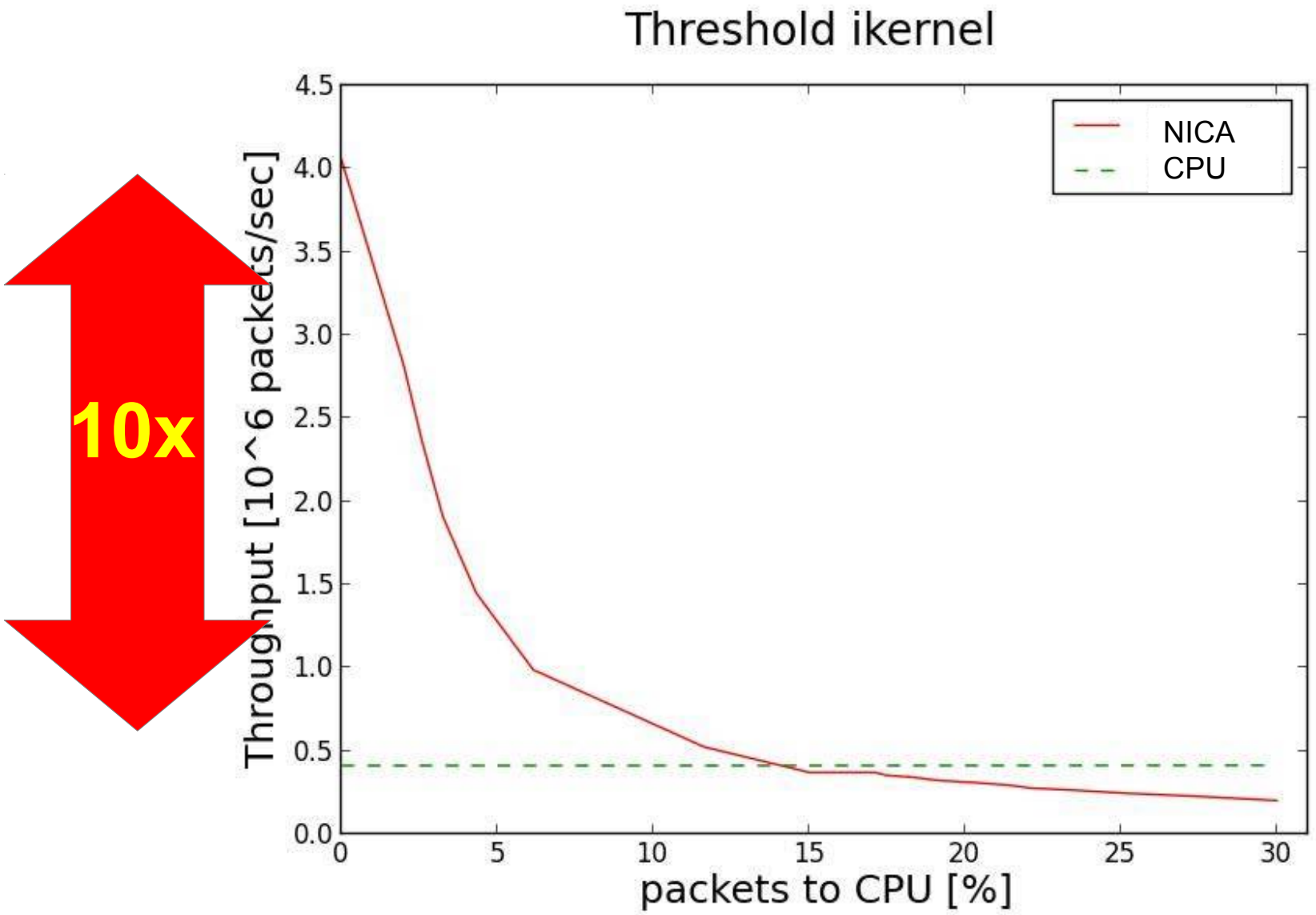


Example:
Tweeter sentiment analysis





Preliminary results



CPU's role

Do the setup
Then leave

CPU's role

Do the setup
Then leave

- Not really:
  - Runs the main program
  - Performs privileged operations, handle exceptions

OmniX is an ongoing work in Accelerator Computer Systems Lab

- Haggai Eran, Amir Watad, Shai Bergman, Tanya Brokhman, Vasilis Dimistas, Lior Zeno, Maroun Tork, Meni Orenbach, Shai Vakhnin, Lev Rosenblit, Marina Minkin, Pavel Lifshits and Gabi Malka

Summary

- Future omni-programmable systems face **programmability wall**
- OmniX: Accelerator-centric Operating System design
- Operating System Services for on-NXU programs
- Omnix removes CPU from control and data pathes

More technical details in [HotOS17]

Summary

- Future omni-programmable systems face **programmability wall**
- OmniX: Accelerator-centric Operating System design
- Operating System Services for on-NXU programs
- Omnix removes CPU from control and data pathes

More technical details in [HotOS17]

Thank you!

mark@ee.technion.ac.il