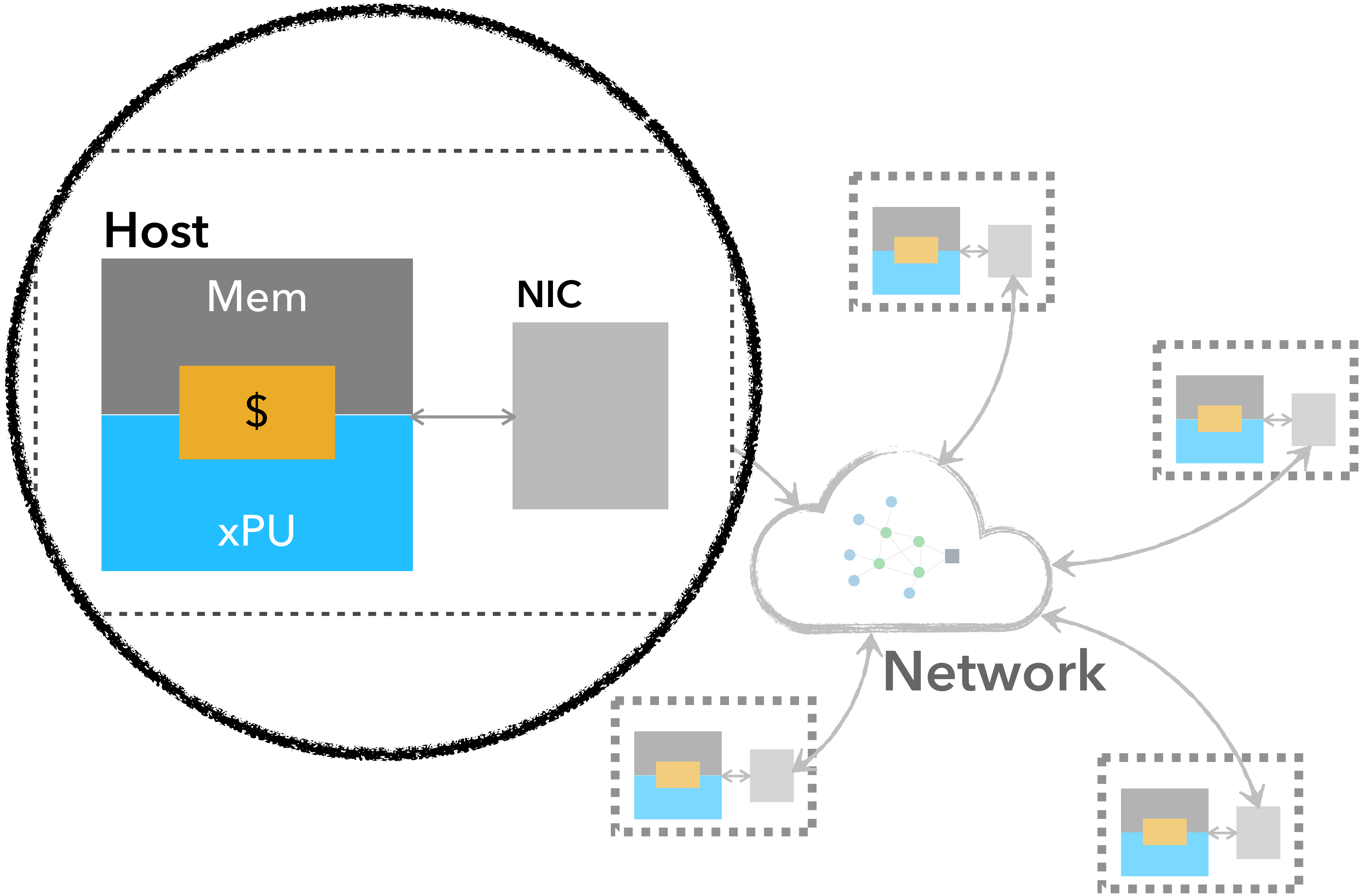
# Data-movement accelerators (DMXs)

**Richard (Rich) Vuduc** – November 29, 2023

These slides + links to papers:
hpcgarage.org/ase45



# Data-movement accelerators (DMXs)

**Richard (Rich) Vuduc** – November 29, 2023

These slides + links to papers:
hpcgarage.org/ase45

45TH ADVANCED SUPERCOMPUTING ENVIRONMENT SEMINAR

# Data-movement accelerators (DMXs)

**Richard (Rich) Vuduc** – November 29, 2023

These slides + links to papers:
hpcgarage.org/ase45

hpc garage

# "Smart" NICs are an old idea…

N.J. Boden et al. "**Myrinet**: a gigabit-per-second local area network." *IEEE Micro*, **15**(1):29–36, **1995**. doi:10.1109/40.342015
F. Petrini et al. "The **Quadrics** network: high-performance clustering technology." IEEE Micro, **22**(1):46–57, **2002**. doi:10.1109/40.988689
R. Brightwell et al. "**SeaStar** interconnect: balanced bandwidth for scalable performance." IEEE Micro, **26**(3):41–47, **2006**. doi:10.1109/MM.2006.65

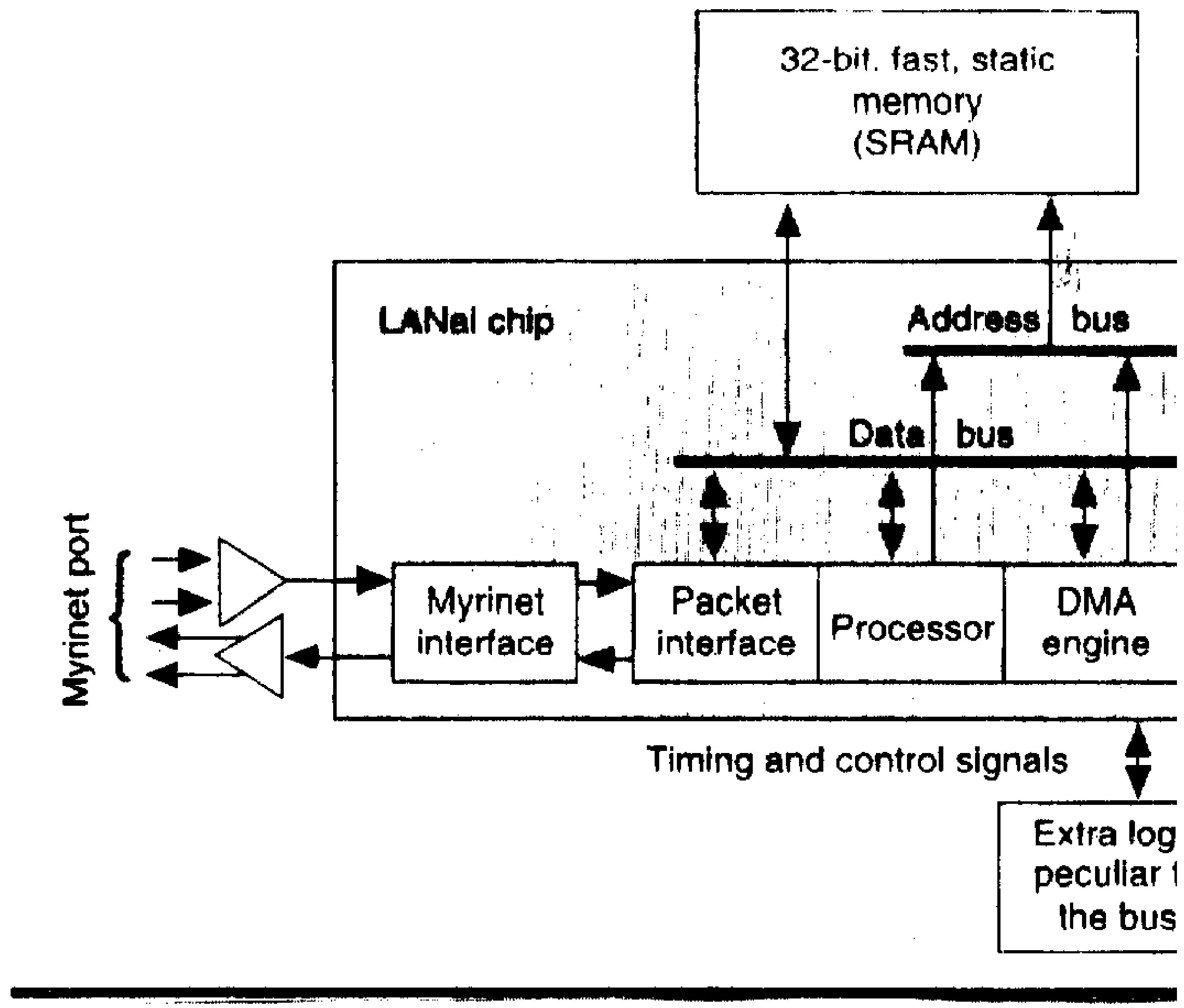



Figure 6. Host interface block diagram.

the switch itself. total pin count, introduce fewer internal conflicts

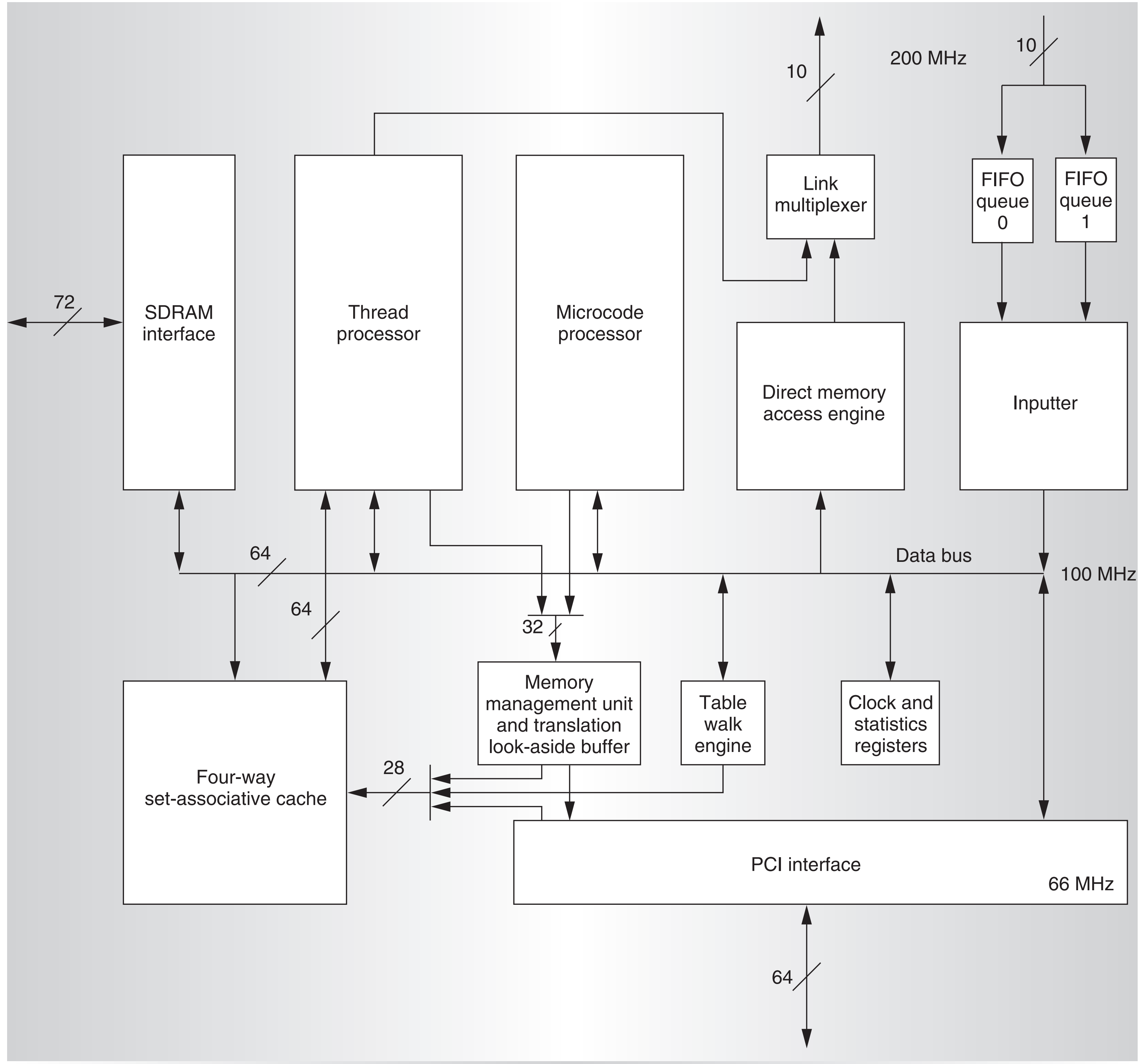



Figure 1. Elan functional units.

to-end protocols that detect faults and automatically retransmit packets.

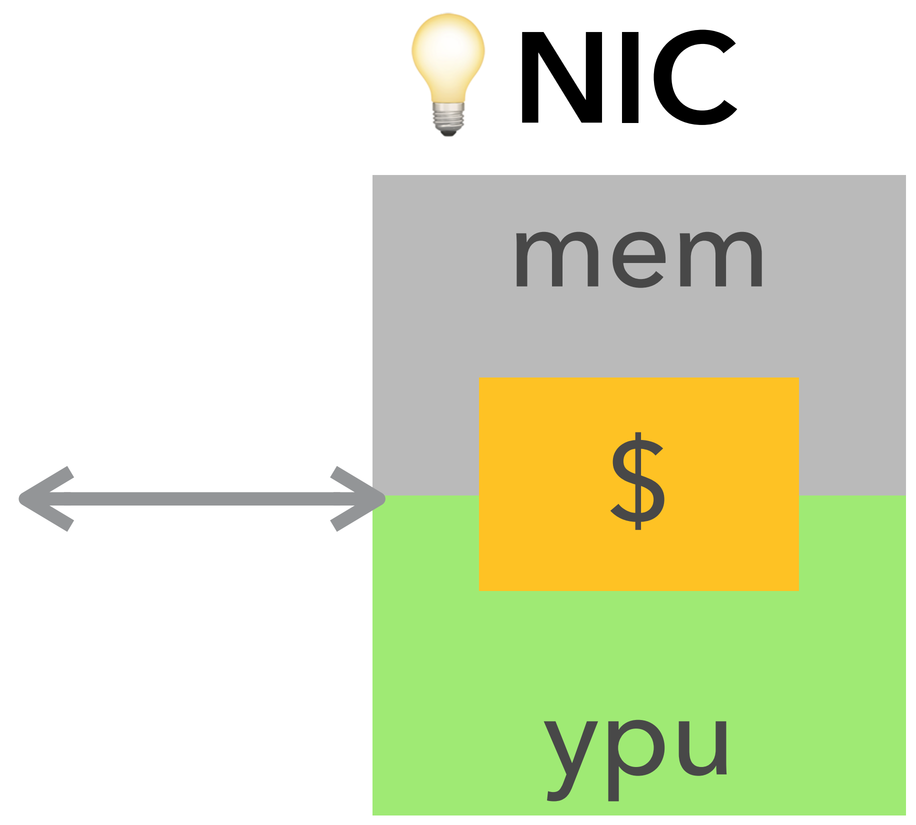
nal functional structure of Elan, shown in Figure 1, centers around two primary processing engines: the microcode processor and the

DATA-MOVEMENT ACCELERATORS (DMXS)

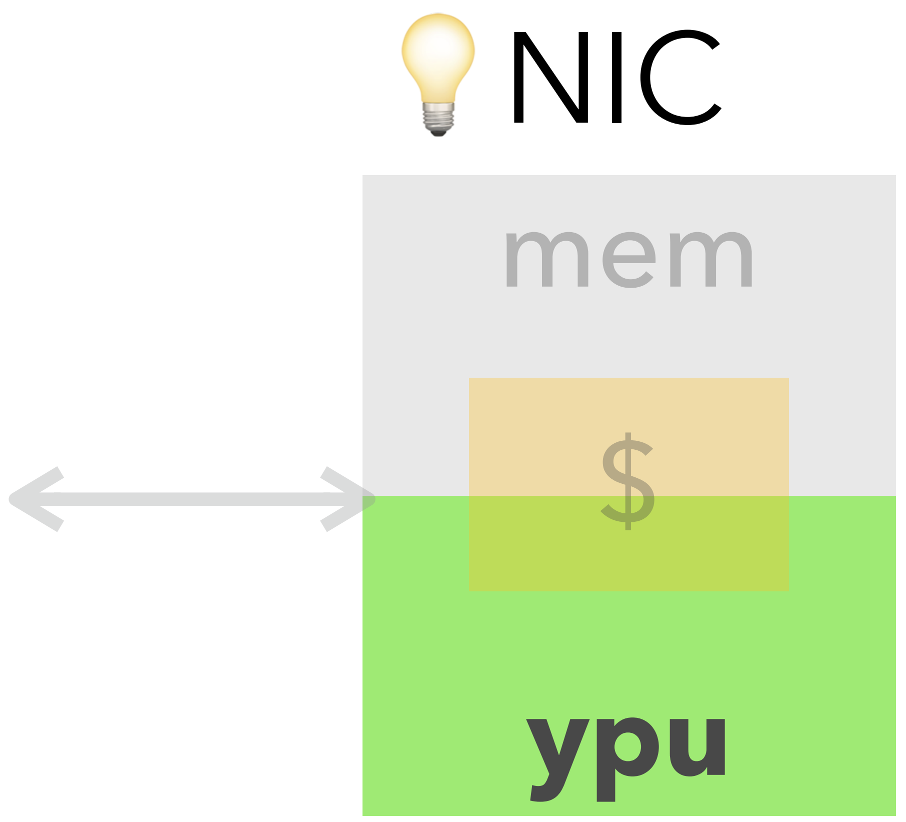
# … born again in a new context.

**Streaming data** and **latency-sensitive**, **in-transit processing** are the hallmarks of modern data center workloads.

# Today's smartNIC landscape: a panoply of designs

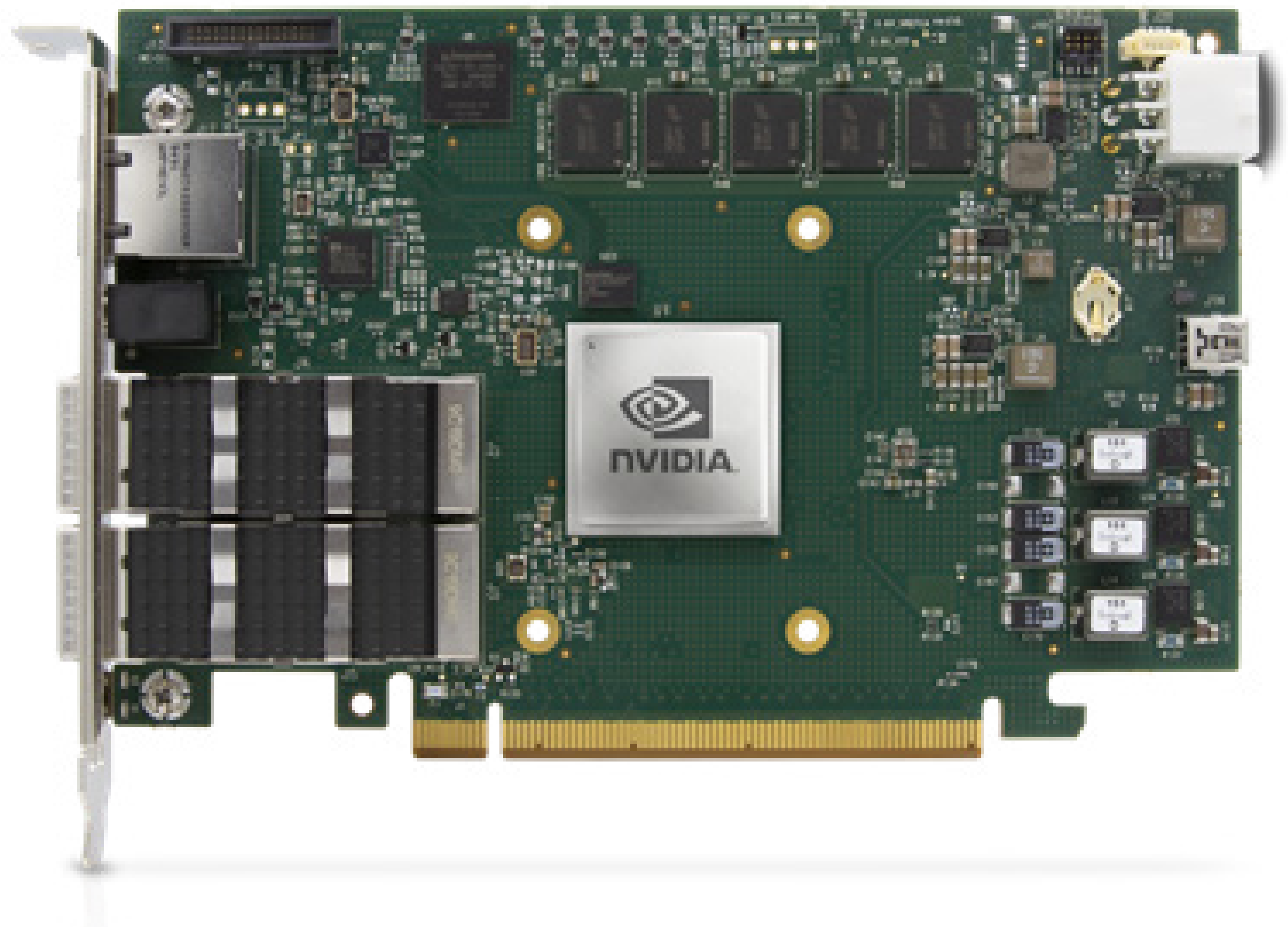


# Today's smartNIC landscape: a panoply of designs

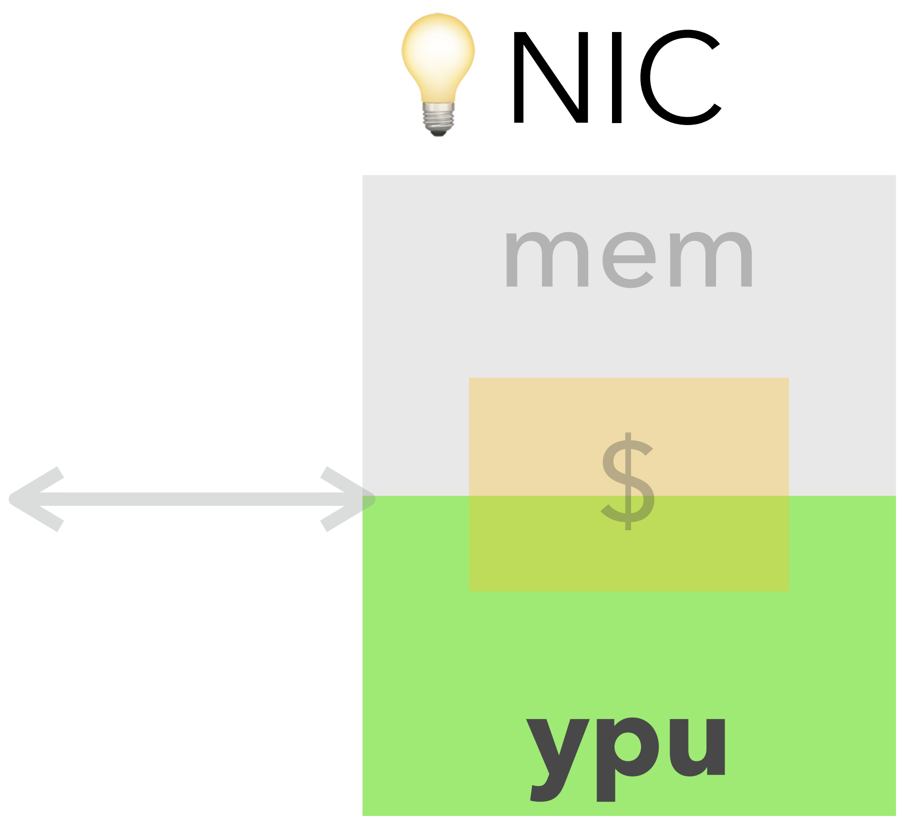



ASICs,
FPGAs,
multicore CPUs,
GPUs, …

# Today's smartNIC landscape: a panoply of designs

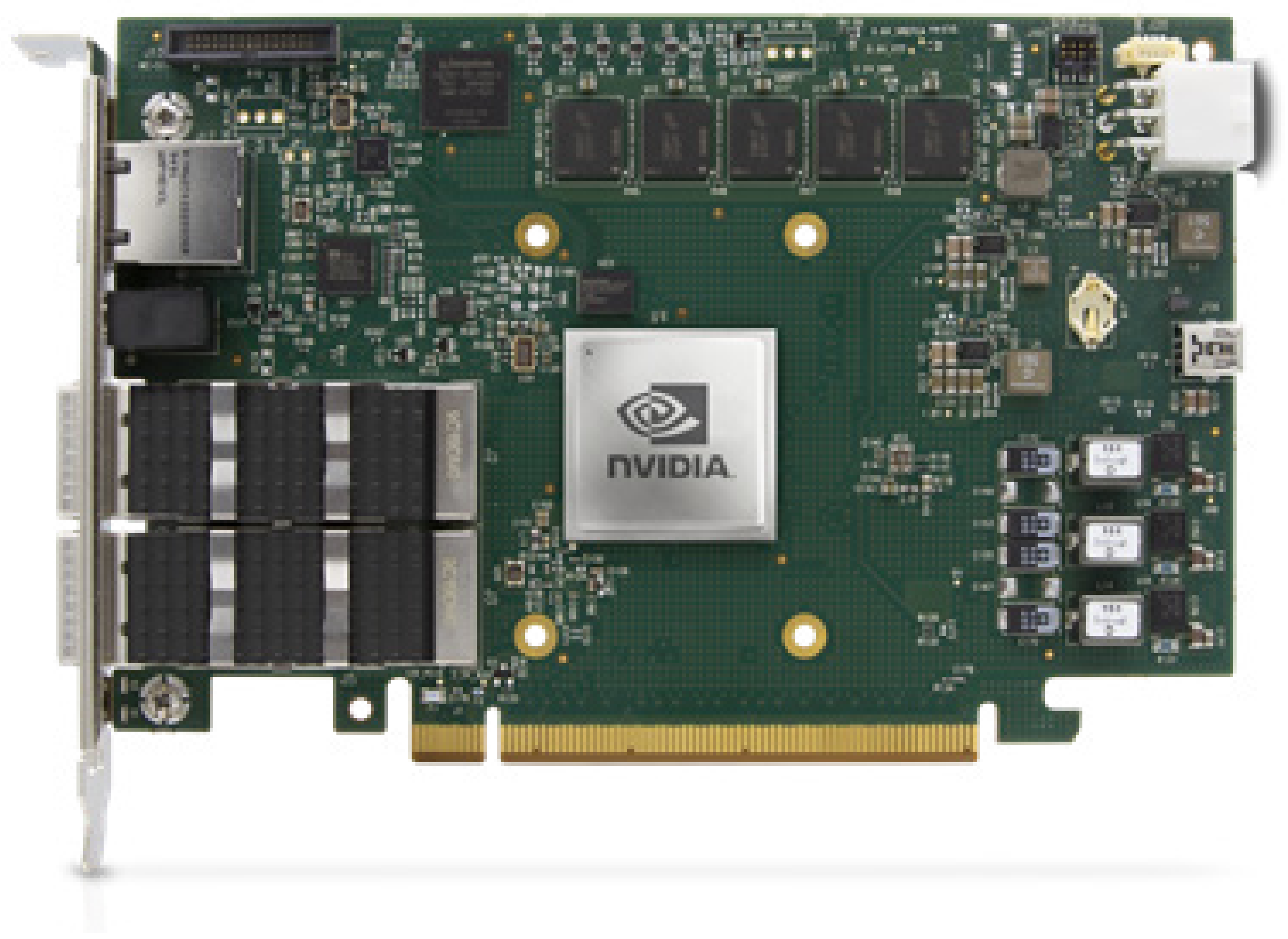



*BlueField-2 DPU - 2x 100Gb/s FHHL form factor*

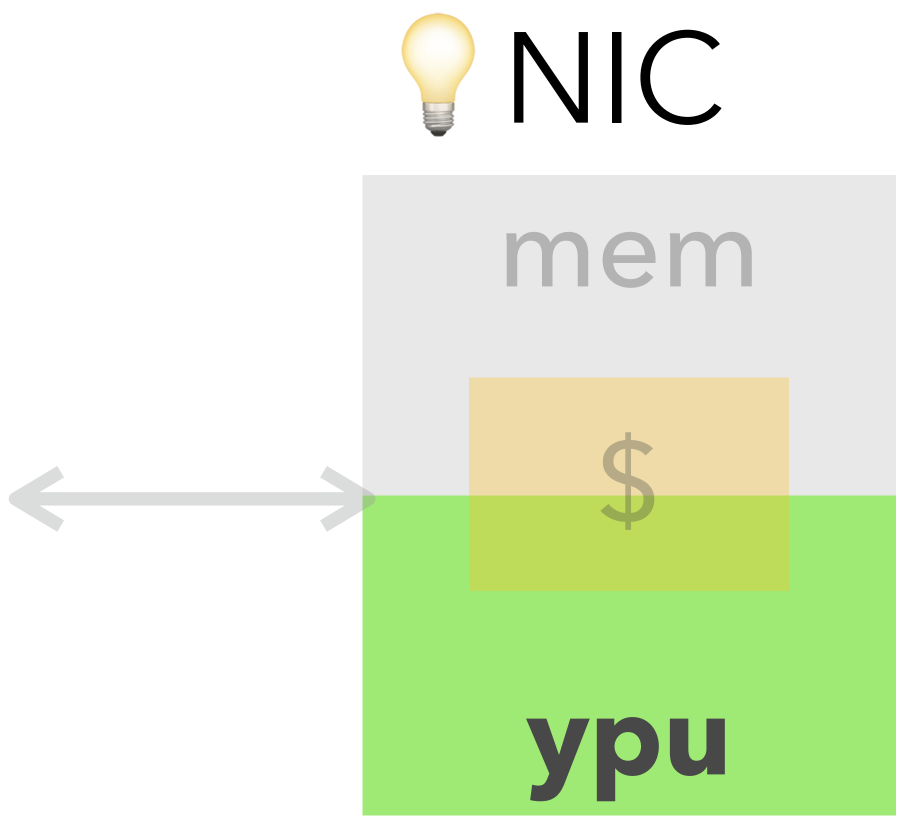



ASICs,
FPGAs,
**multicore CPUs,**
GPUs, ...

# Today's smartNIC landscape: a panoply of designs



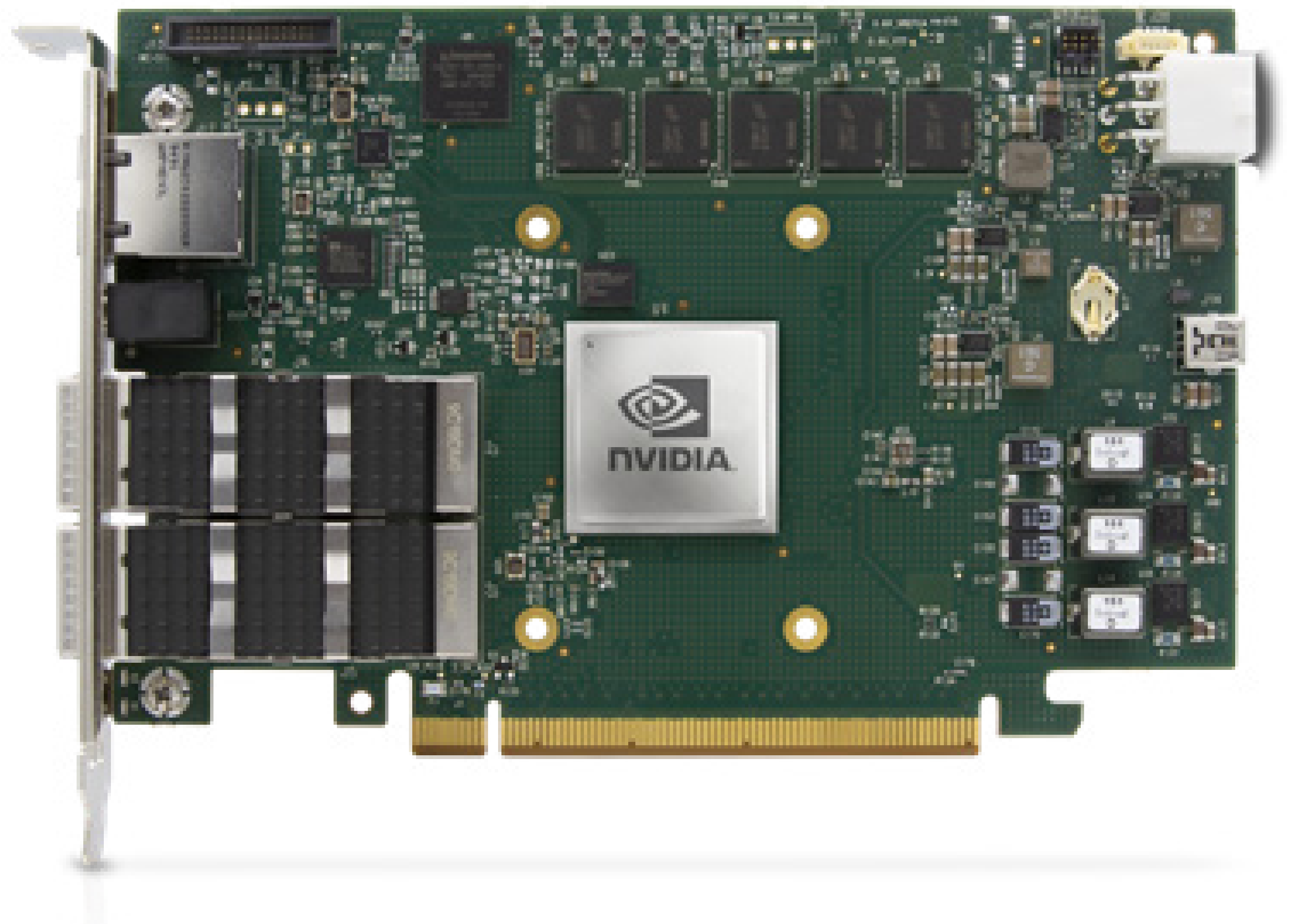

*BlueField-2 DPU - 2x 100Gb/s FHHL form factor*



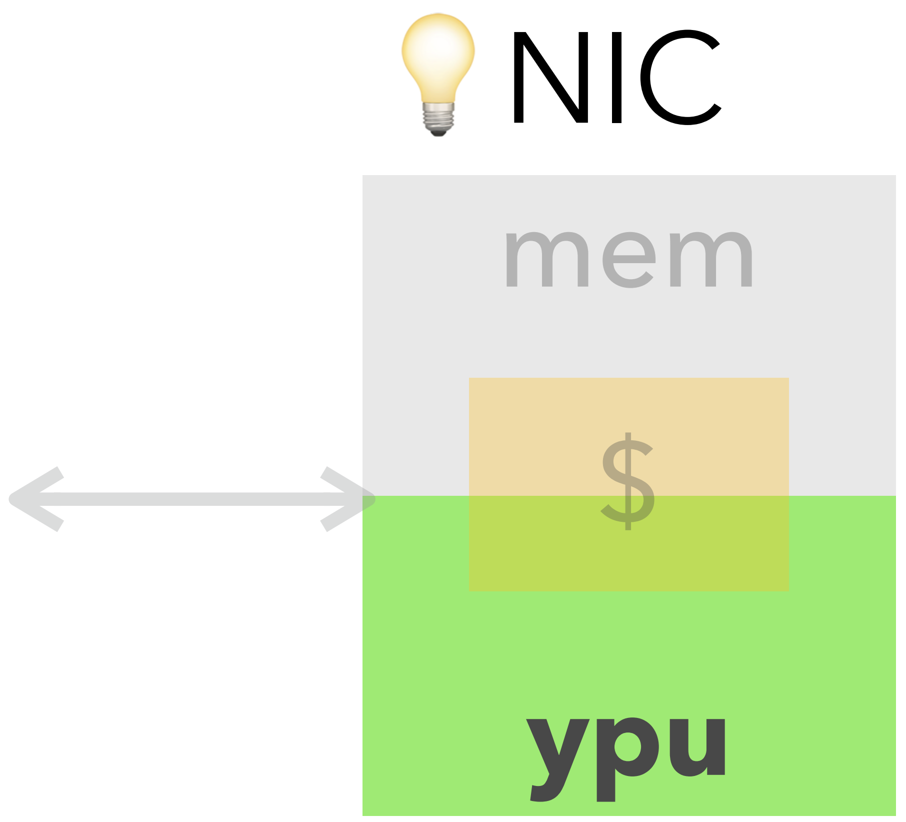

"DPU"

(data processing unit)

# Today's smartNIC landscape: a panoply of designs




*BlueField-2 DPU - 2x 100Gb/s FHHL form factor*




"DPU"

**BF-2**: 8-core Arm v8 A72 @ 2.6 GHz, DDR4 4800 MT/s, HDR100 @ 100 Gb/s

**BF-3**: 16-core Arm v8 A78 @ 2.25 GHz, DDR5 5600 MT/s, NDR200 @ 200 Gb/s

# NVIDIA DPU ROADMAP

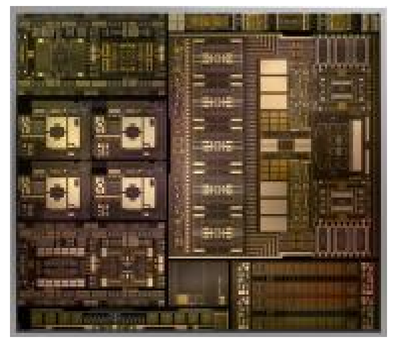
Exponential Growth in Data Center Infrastructure Processing

https://hc33.hotchips.org/assets/program/conference/day1/HC2021.NVIDIA.IdanBurstein.v08.norecording.pdf

100X

10X

1X

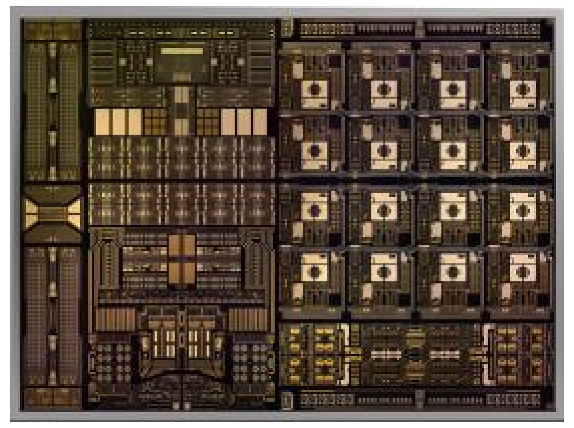


*BF-2*

7 B transistors

0.7 Top/s

200 Gb/s

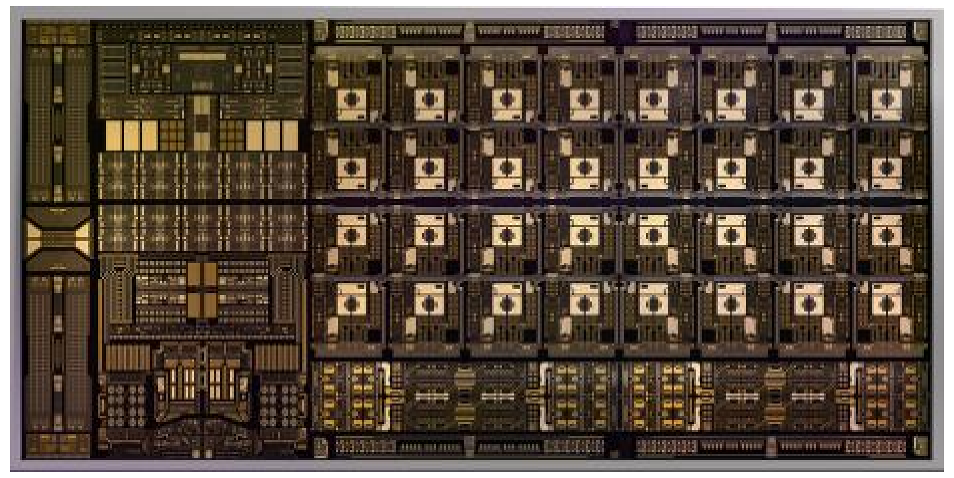


*BF-3*

22 B

1.5 Top/s

400 Gb/s



*BF-4*

64 B

400 Top/s

800 Gb/s

DOCA — ONE DEVELOPMENT ARCHITECTURE

2020 2022 2024

# Q: Are smartNICs for data centers relevant to HPC?

# Claim: Communication is an inevitable bottleneck

**Recall: “The” dominant paradigm of CS:**

$$\mathcal{O}\left(N^2\right) \longrightarrow \mathcal{O}(N)$$

Reduces **energy**: fewer (fl)ops, less storage

Recall: "The" dominant paradigm of CS:

$$\mathcal{O}(N^2) \longrightarrow \boxed{\mathcal{O}(N)}$$

**% time communicating increases**

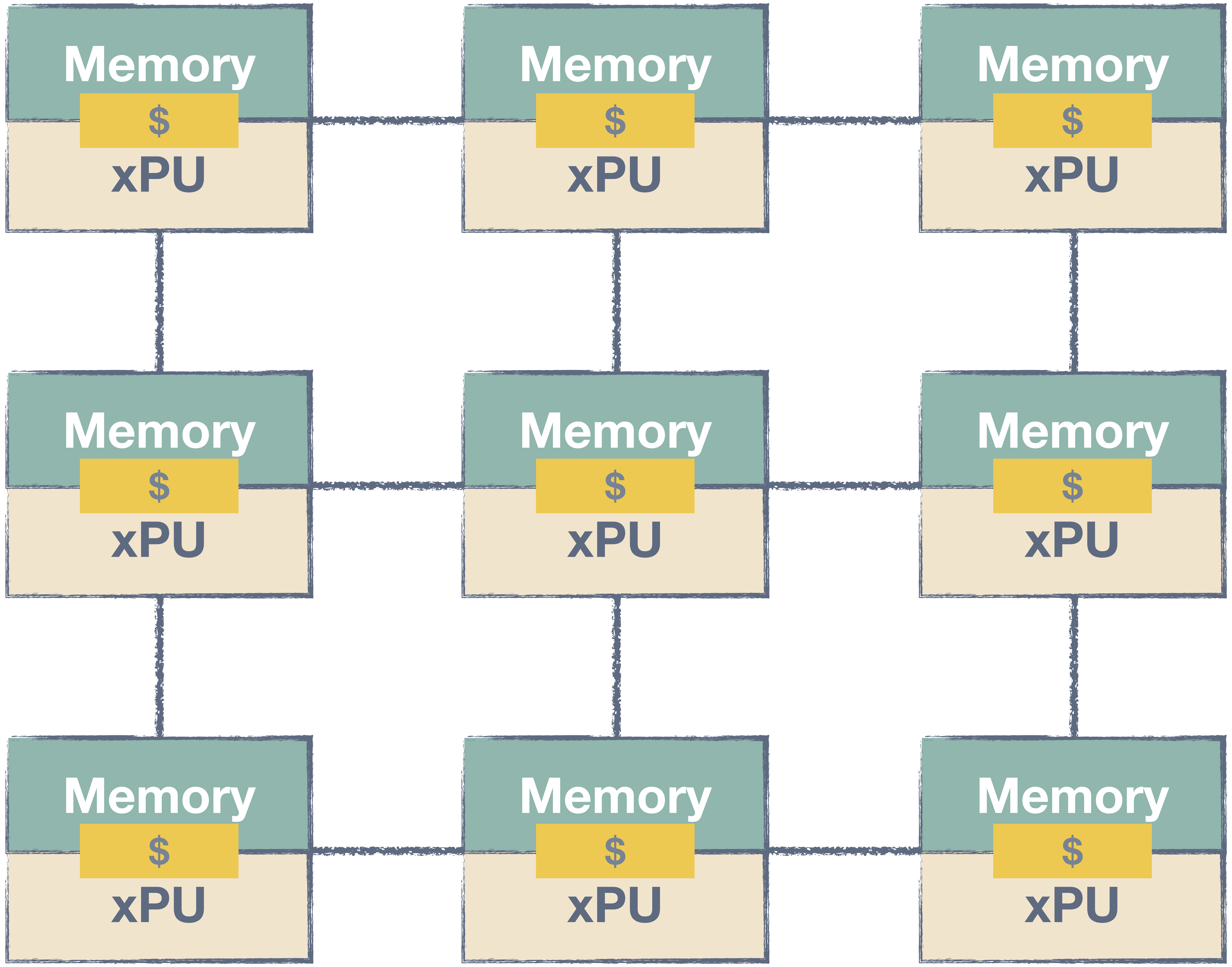
# An Iron Law of Parallel and Distributed Computation

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

**As a program executes on this system, it incurs two types of communication cost.**

"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.




Two costs: $T_{network} + T_{memory}$

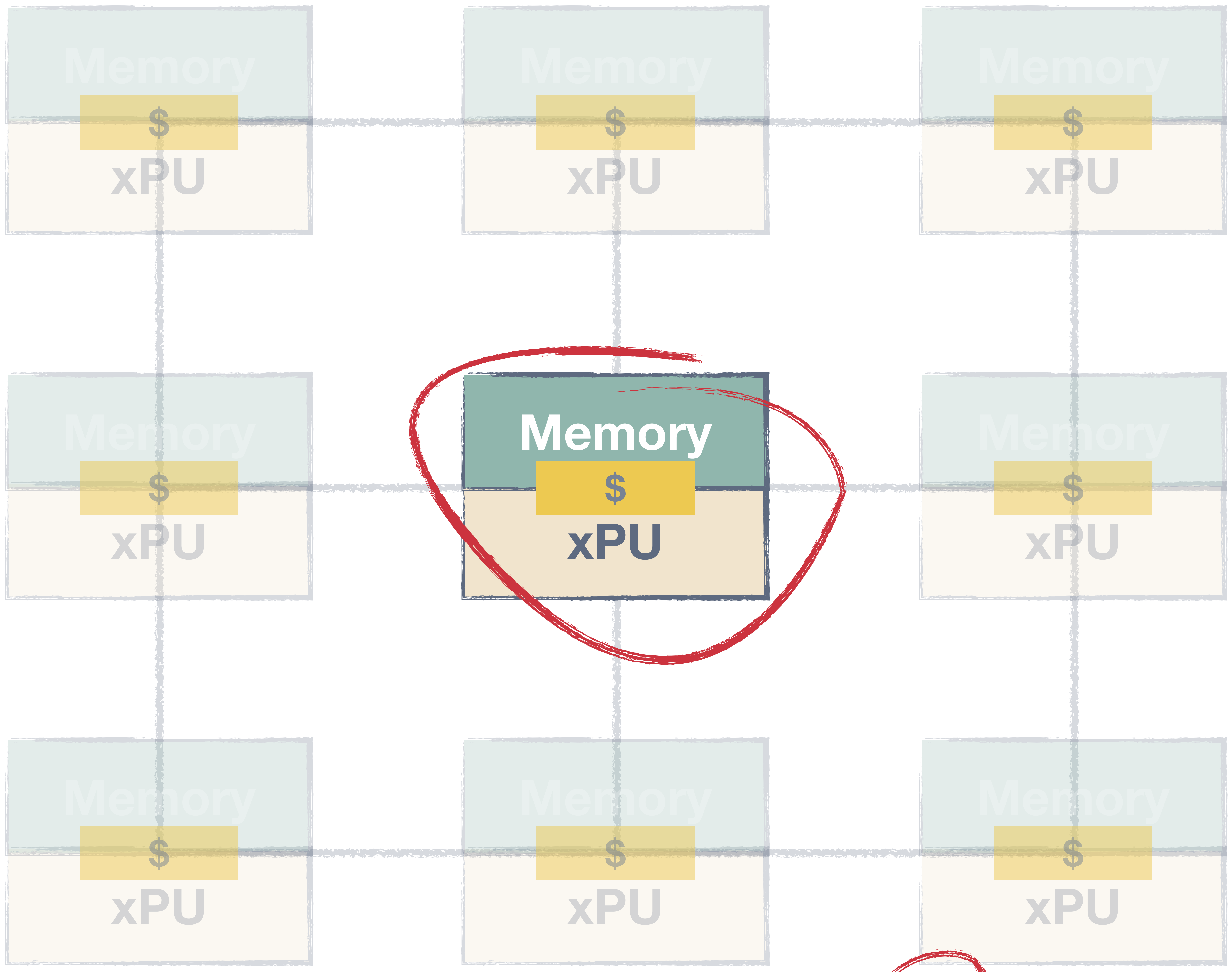
# An Iron Law of Parallel and Distributed Computation

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

**"Vertical" communication occurs in the memory system between, say, RAM and cache.**

"Horizontal" communication occurs between nodes across the network.




Two costs: $T_{\text{network}} + T_{\text{memory}}$

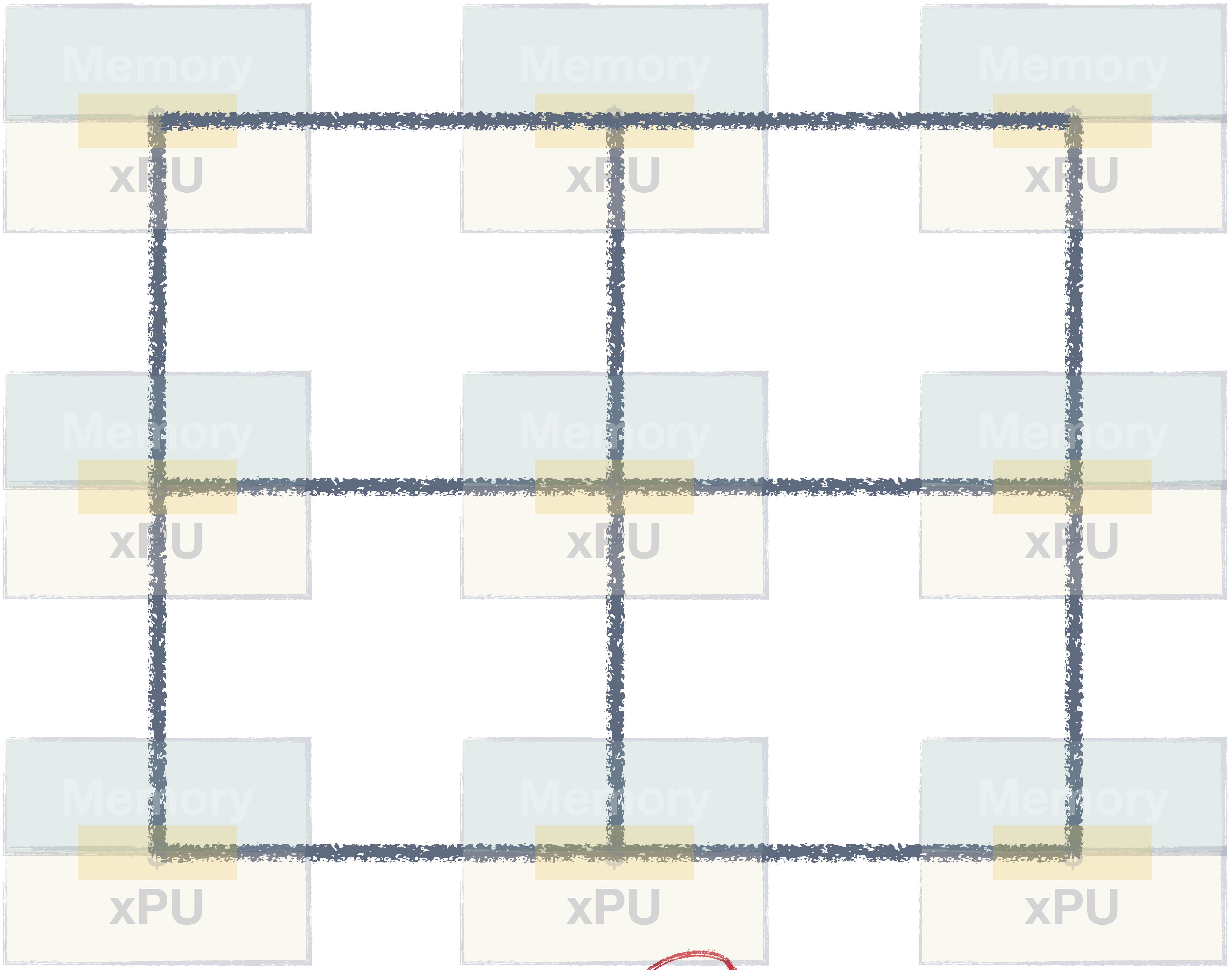
# An Iron Law of Parallel and Distributed Computation

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

**"Horizontal" communication occurs between nodes across the network.**




Two costs: $T_{\text{network}} + T_{\text{memory}}$

# An Iron Law of Parallel and Distributed Computation

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

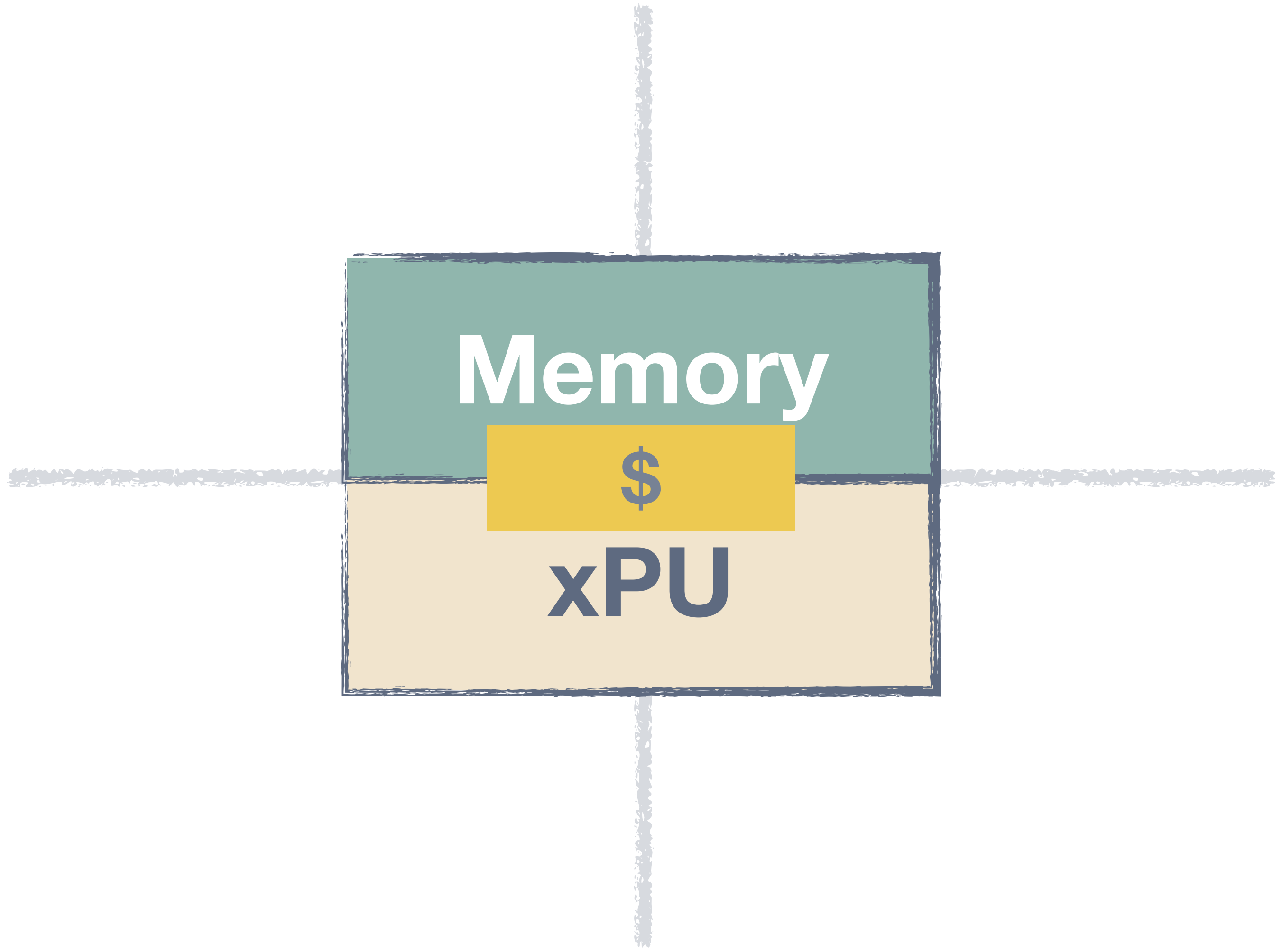
"Horizontal" communication occurs between nodes across the network.

Compute time

Network time

$$T \geq \max\{T_{\text{op}}, T_{\text{mem}}, T_{\text{net}}\}$$

Memory time



# An Iron Law of Parallel and Distributed Computation

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

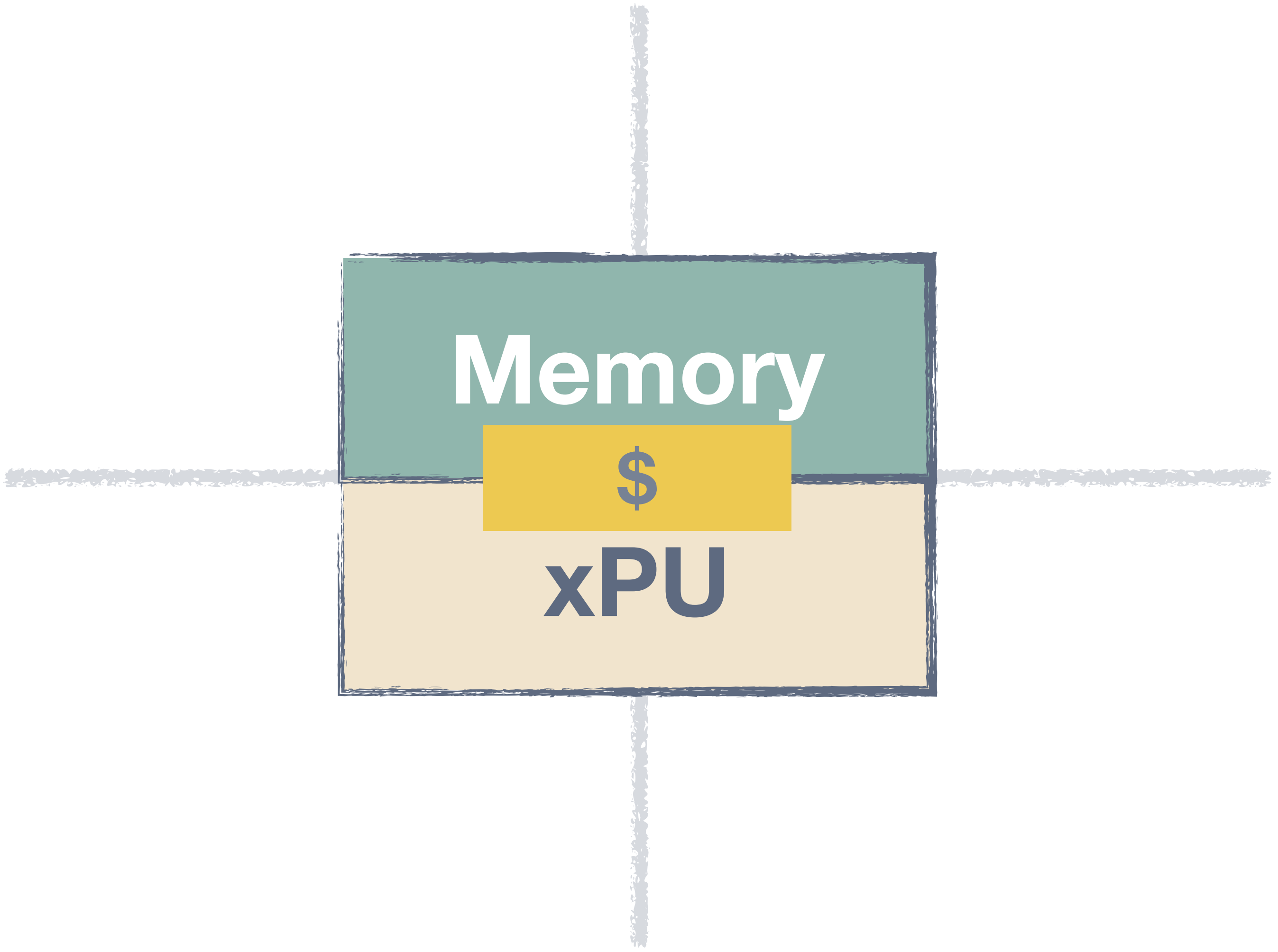
"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.

network penalty

$$T \geq T_{\text{op}} \max\left\{1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}}\right\}$$

memory penalty



# An Iron Law of Parallel and Distributed Computation

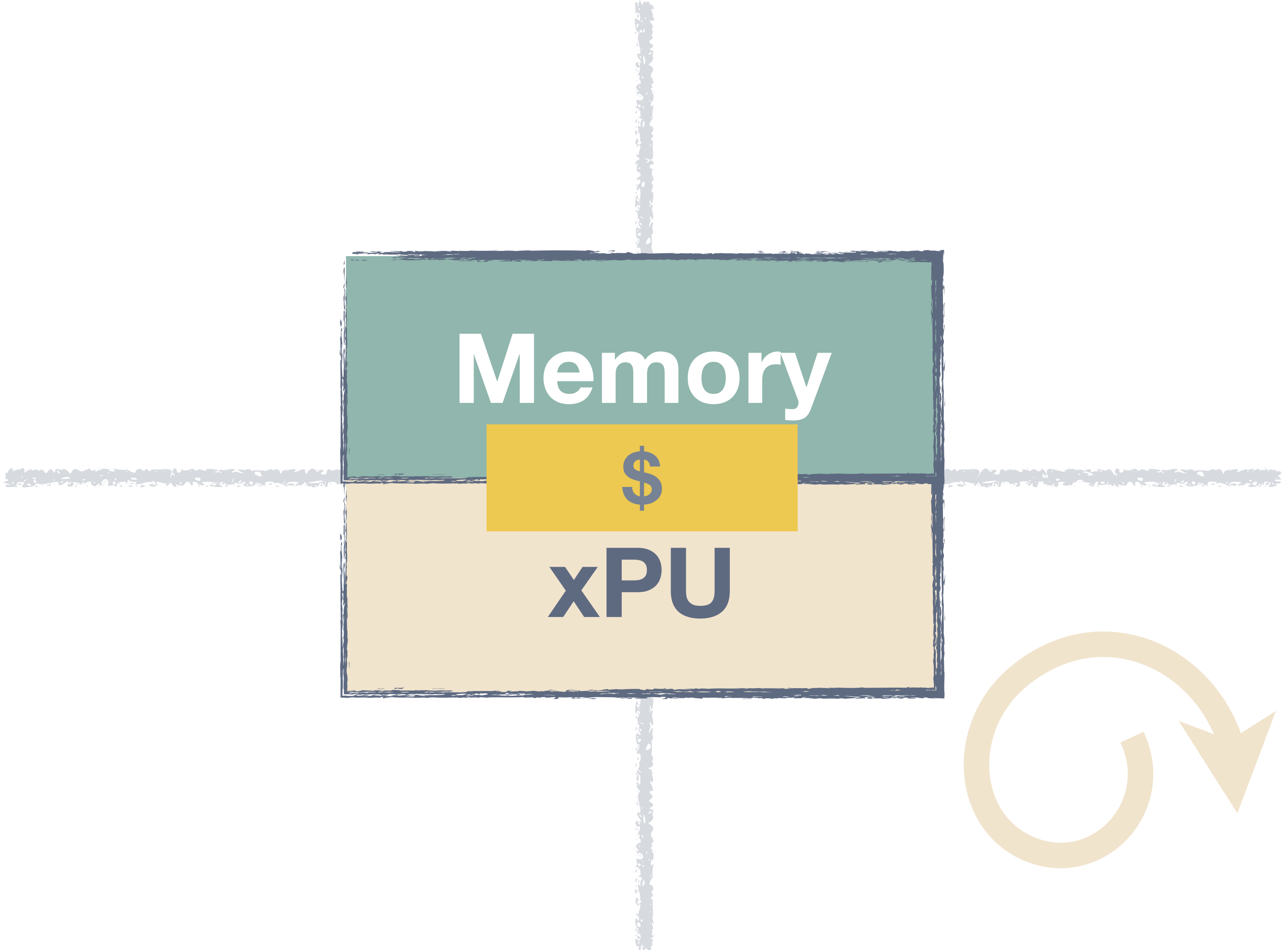
A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.

$$T \geq T_{\text{op}} \max \left\{ 1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}} \right\}$$




$R_0$ $\frac{\text{ops}}{\text{time}}$

# An Iron Law of Parallel and Distributed Computation

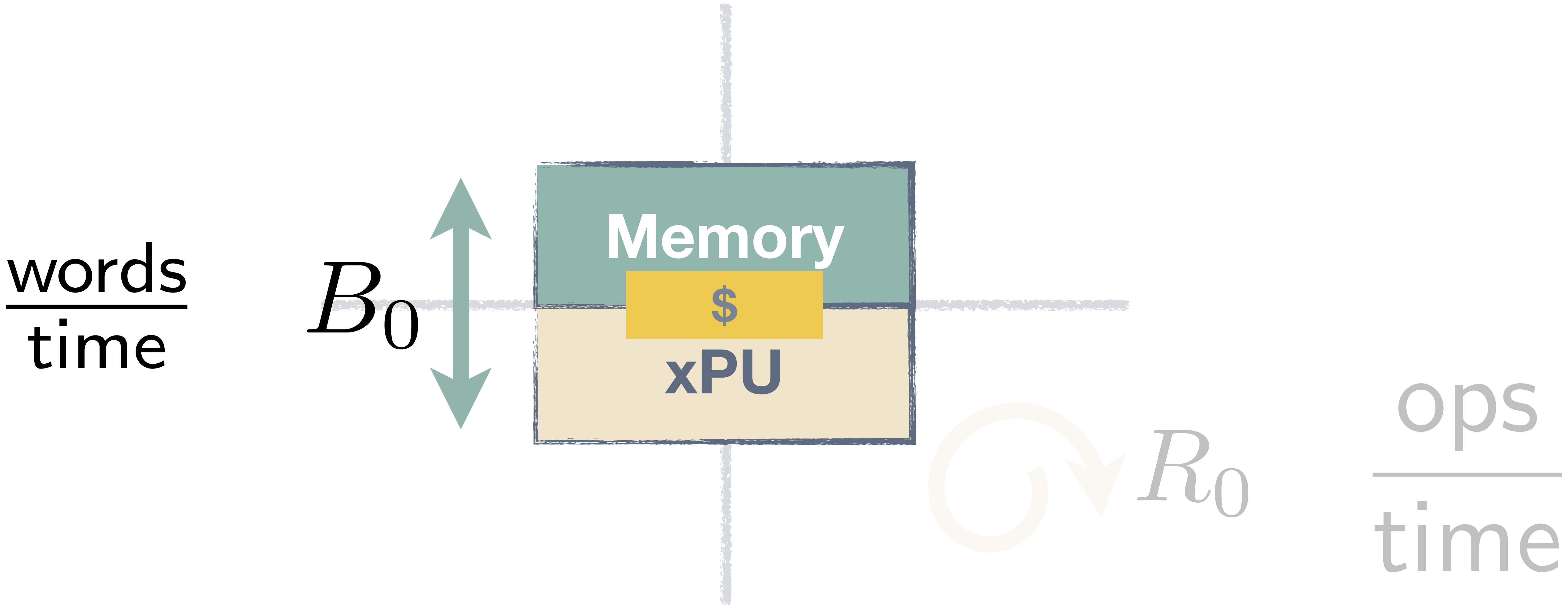
$$T \geq T_{\text{op}} \max\left\{1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}}\right\}$$

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.



# An Iron Law of Parallel and Distributed Computation

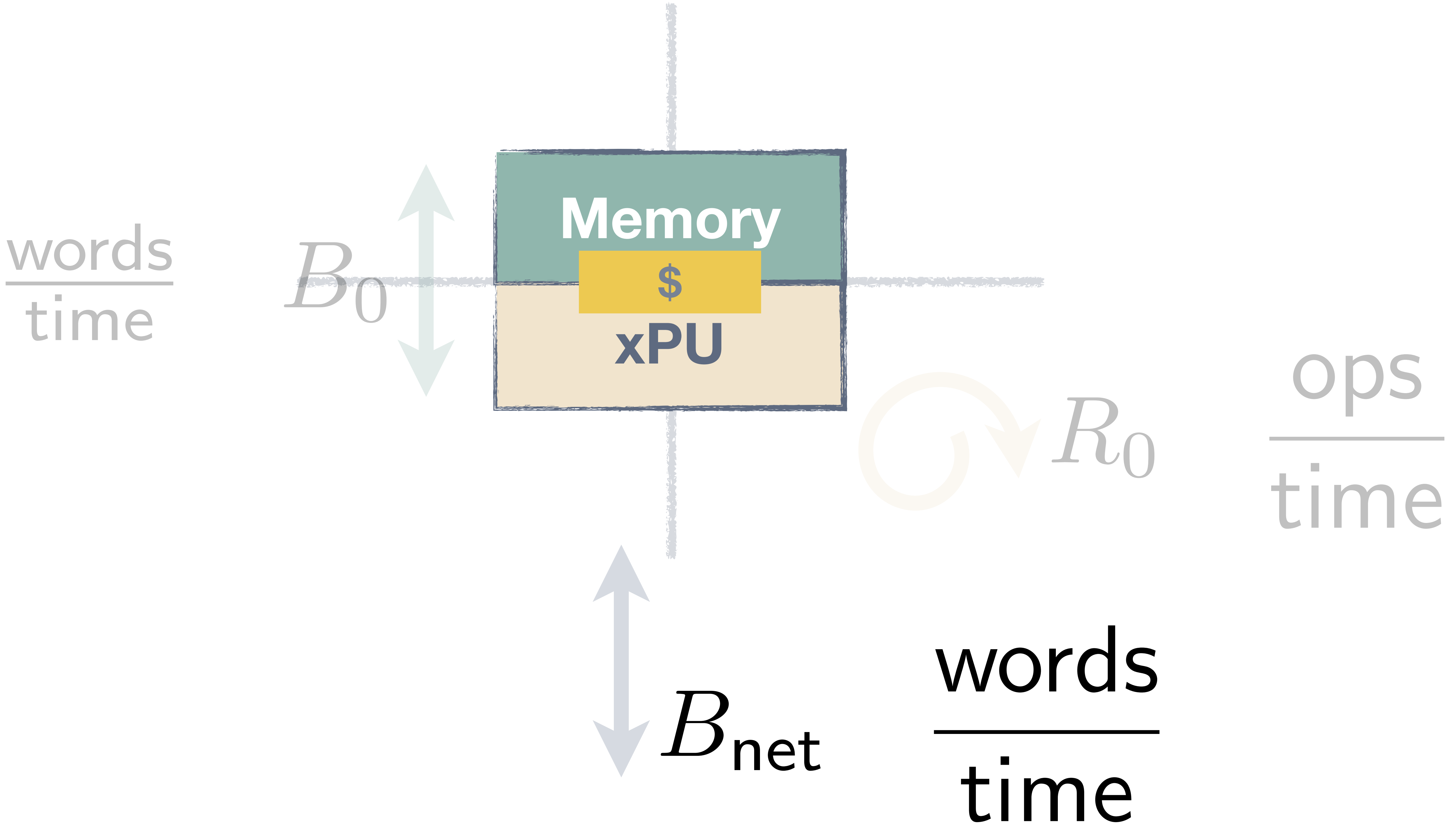
$$T \geq T_{\text{op}} \max \left\{ 1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}} \right\}$$

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.



# An Iron Law of Parallel and Distributed Computation

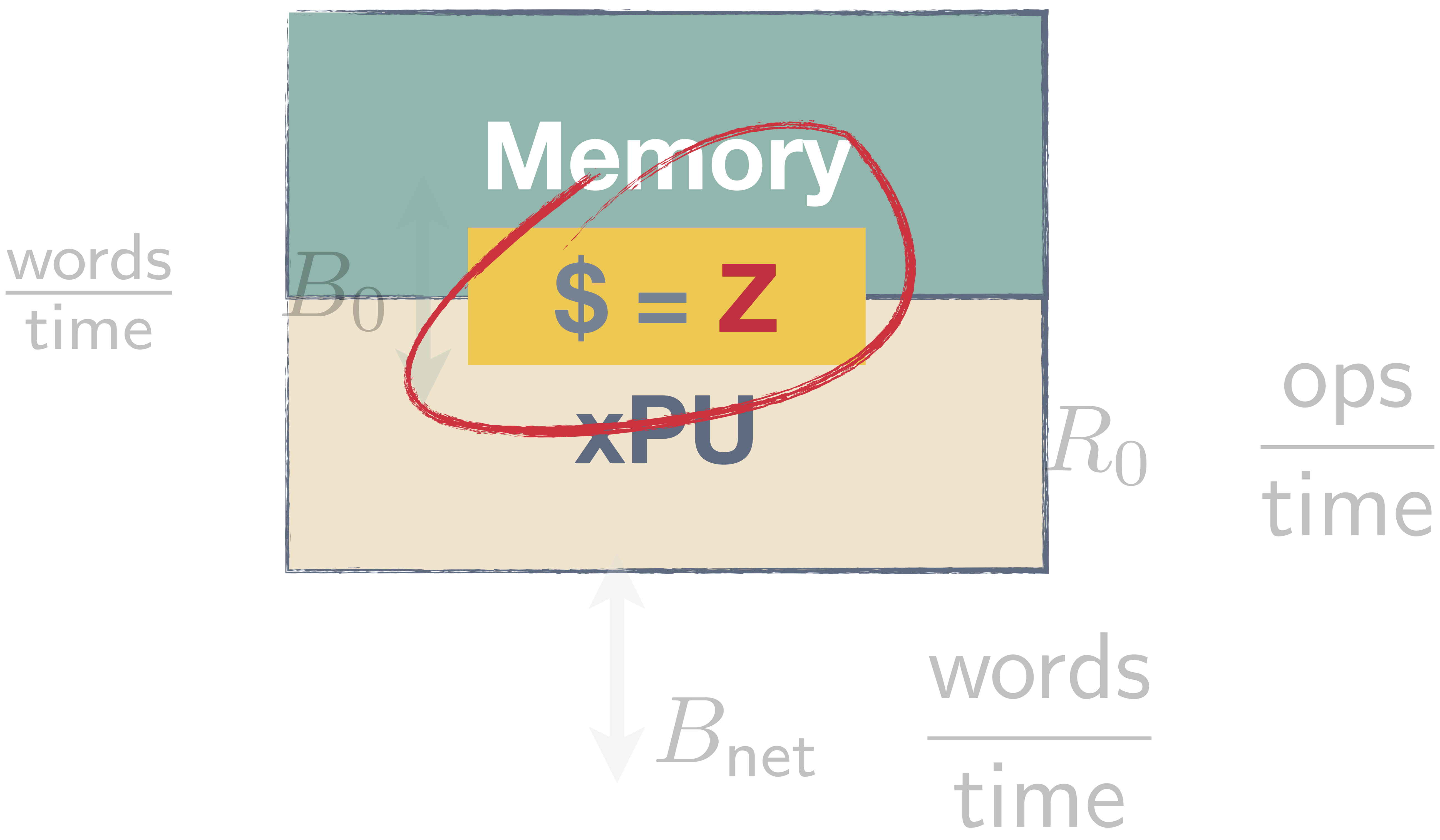
$$T \geq T_{\text{op}} \max\left\{1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}}\right\}$$

A modern cluster or supercomputer is, to first order, a collection of processing nodes. Each node has a processor ("xPU") and a two-level memory hierarchy. Nodes are connected by a network.

As a program executes on this system, it incurs two types of communication cost.

"Vertical" communication occurs in the memory system between, say, RAM and cache.

"Horizontal" communication occurs between nodes across the network.



# An Iron Law of Parallel and Distributed Computation

**memory penalty**

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

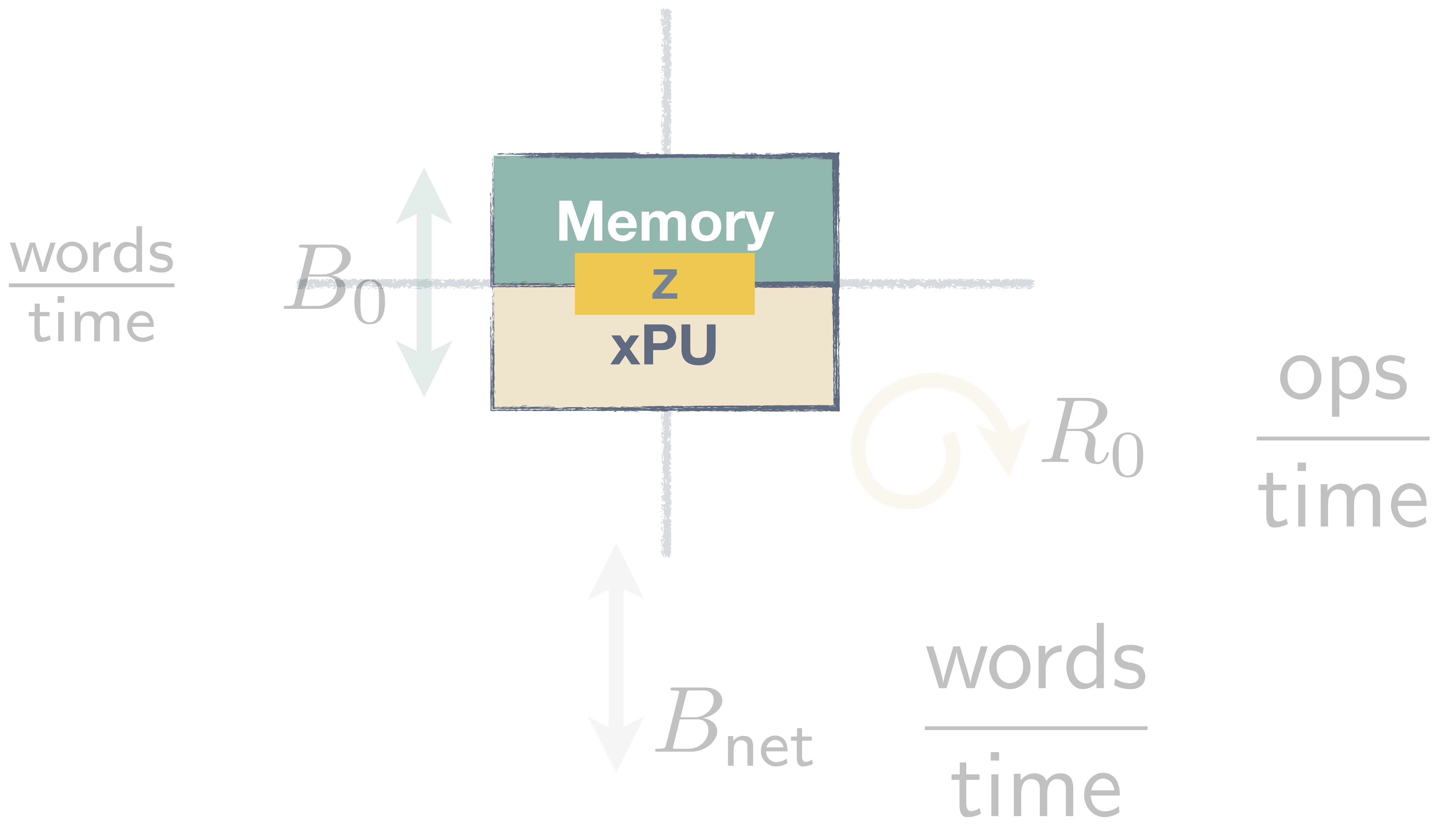
$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

**network penalty**

network penalty

$$T \geq T_{\text{op}} \max\left\{1, \frac{T_{\text{mem}}}{T_{\text{op}}}, \frac{T_{\text{net}}}{T_{\text{op}}}\right\}$$

memory penalty




*Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."*

# An Iron Law of Parallel and Distributed Computation

## Lower is better

(⇒ "smaller" processors)

memory penalty

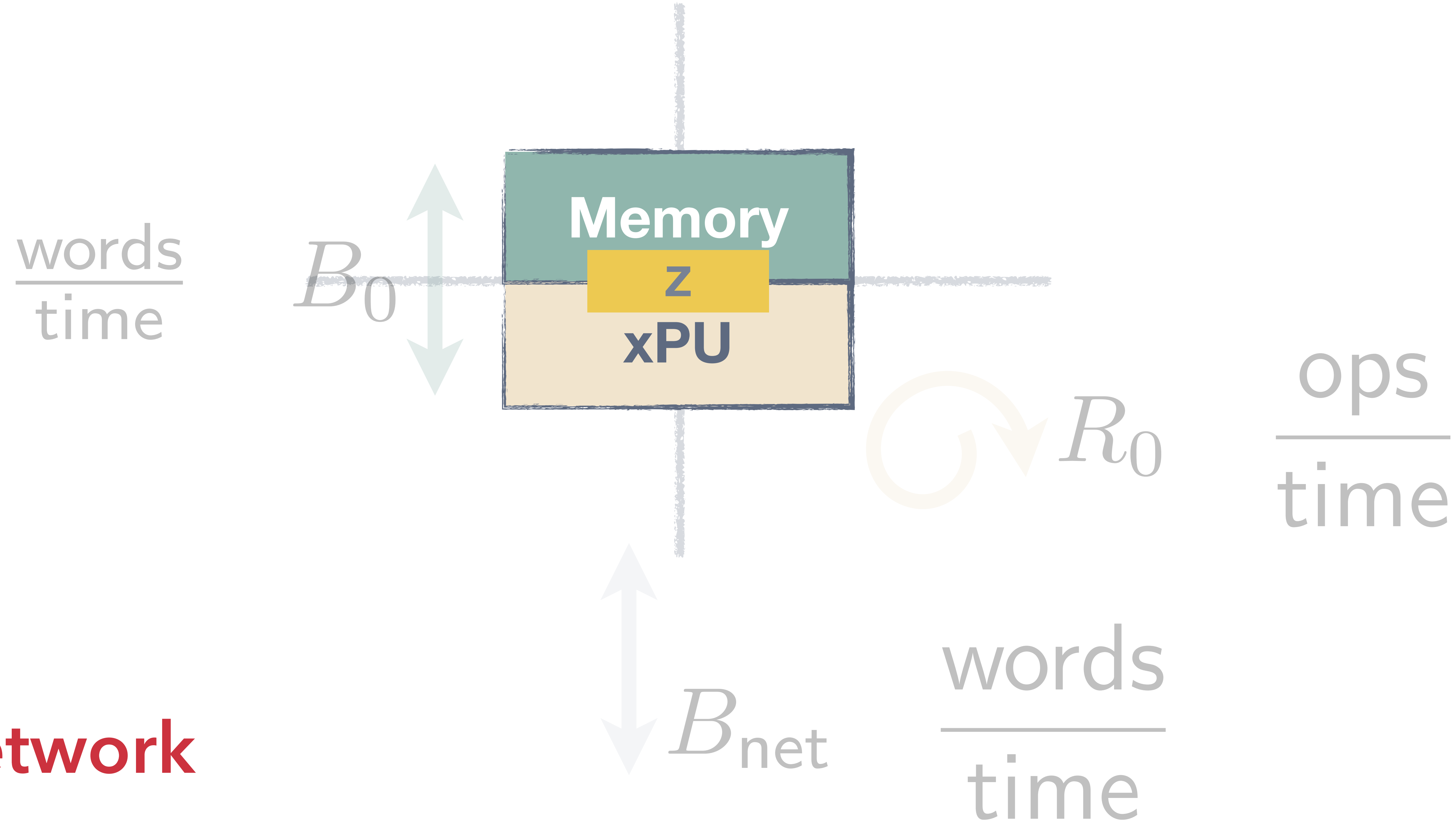
Processor-memory op:byte

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

network penalty

Processor-network op:byte




Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."

# An Iron Law of Parallel and Distributed Computation

memory penalty

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

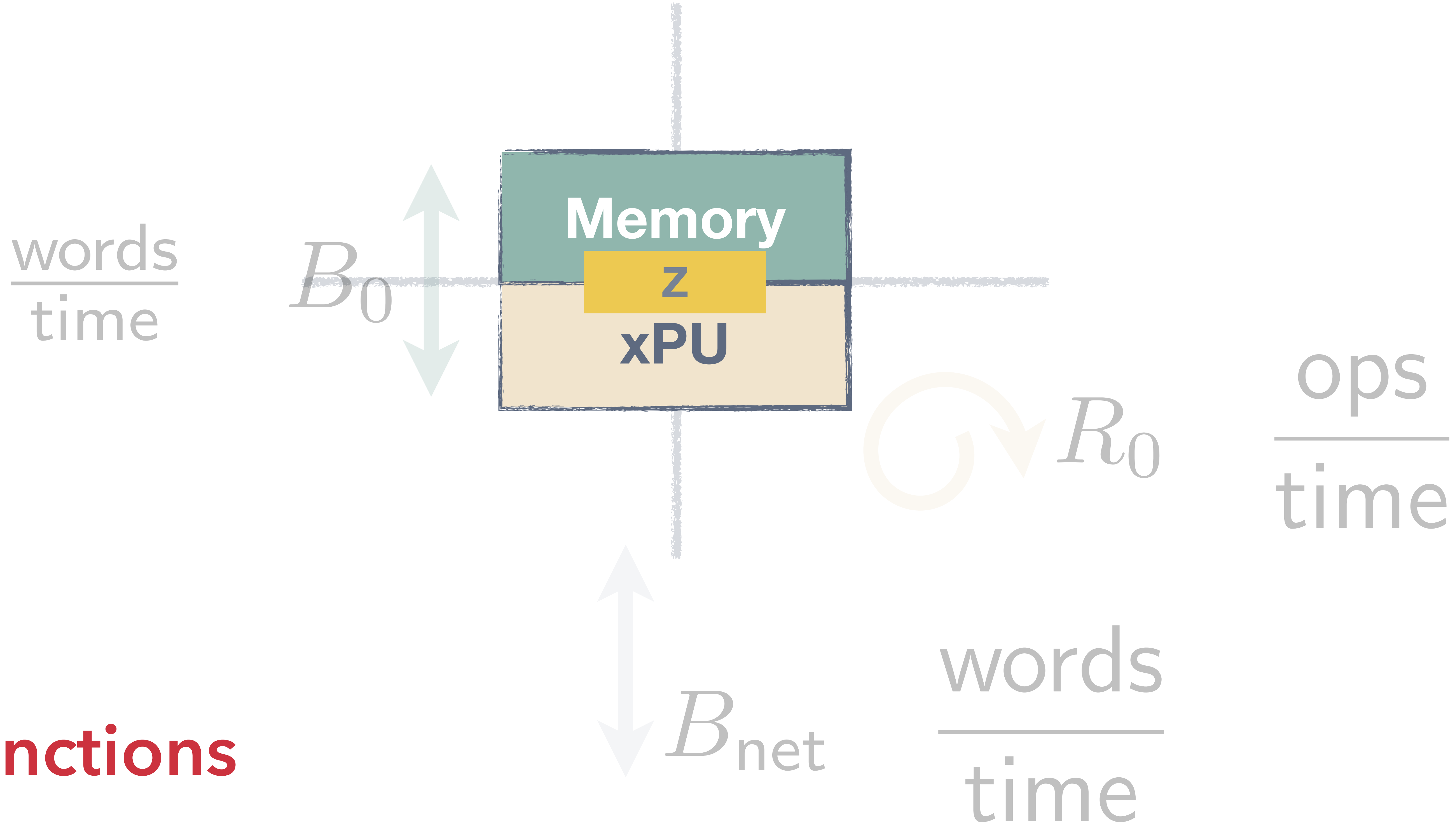
$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

network penalty

Increasing functions

## Lower is better

(⇒ "**bigger**" processors)






*Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."*

# An Iron Law of Parallel and Distributed Computation

memory penalty

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

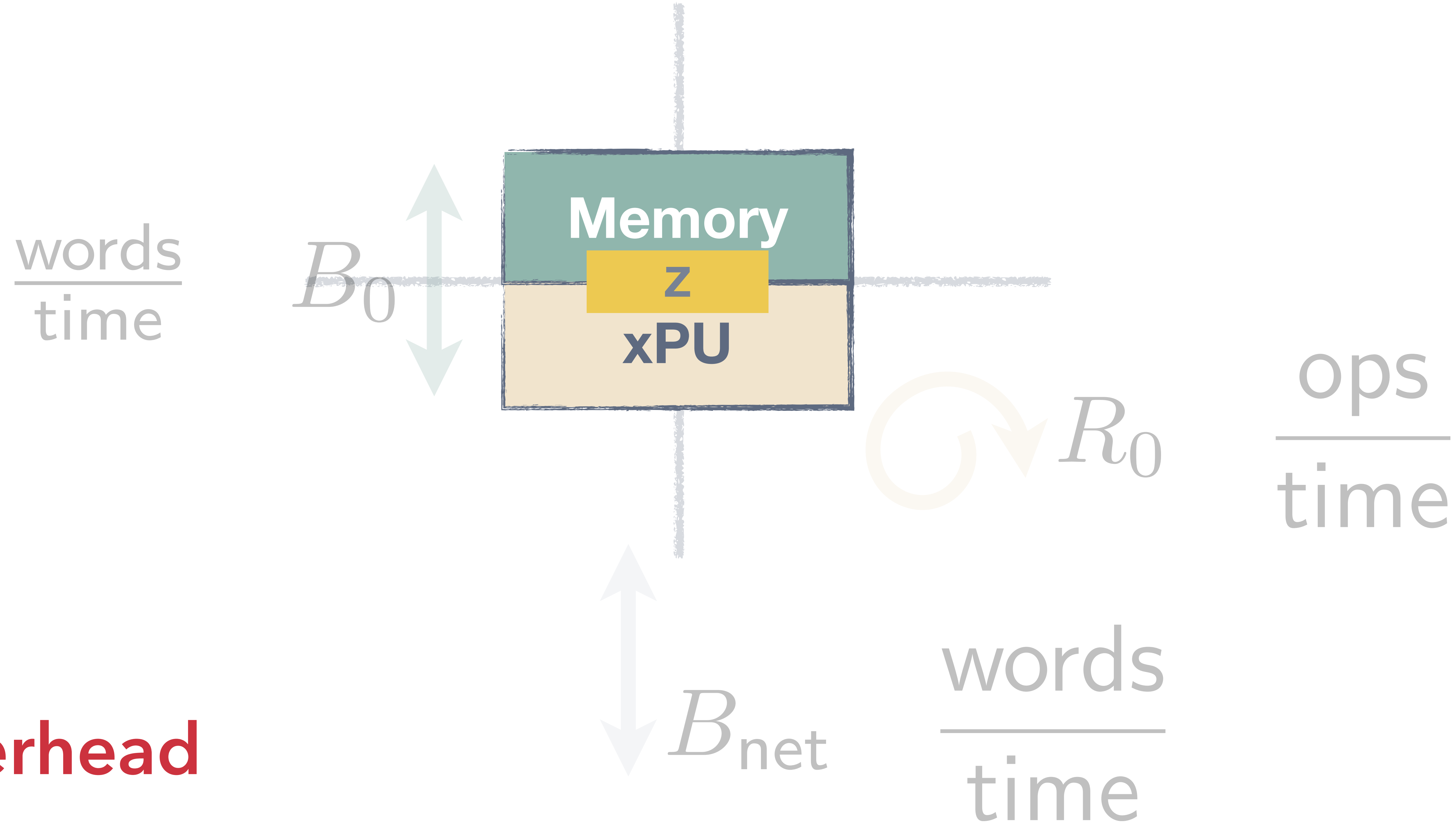
$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

network penalty

Network overhead

## Lower is better

(⇒ "**bigger**" processors)






*Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."*

# An Iron Law of Parallel and Distributed Computation

memory penalty

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

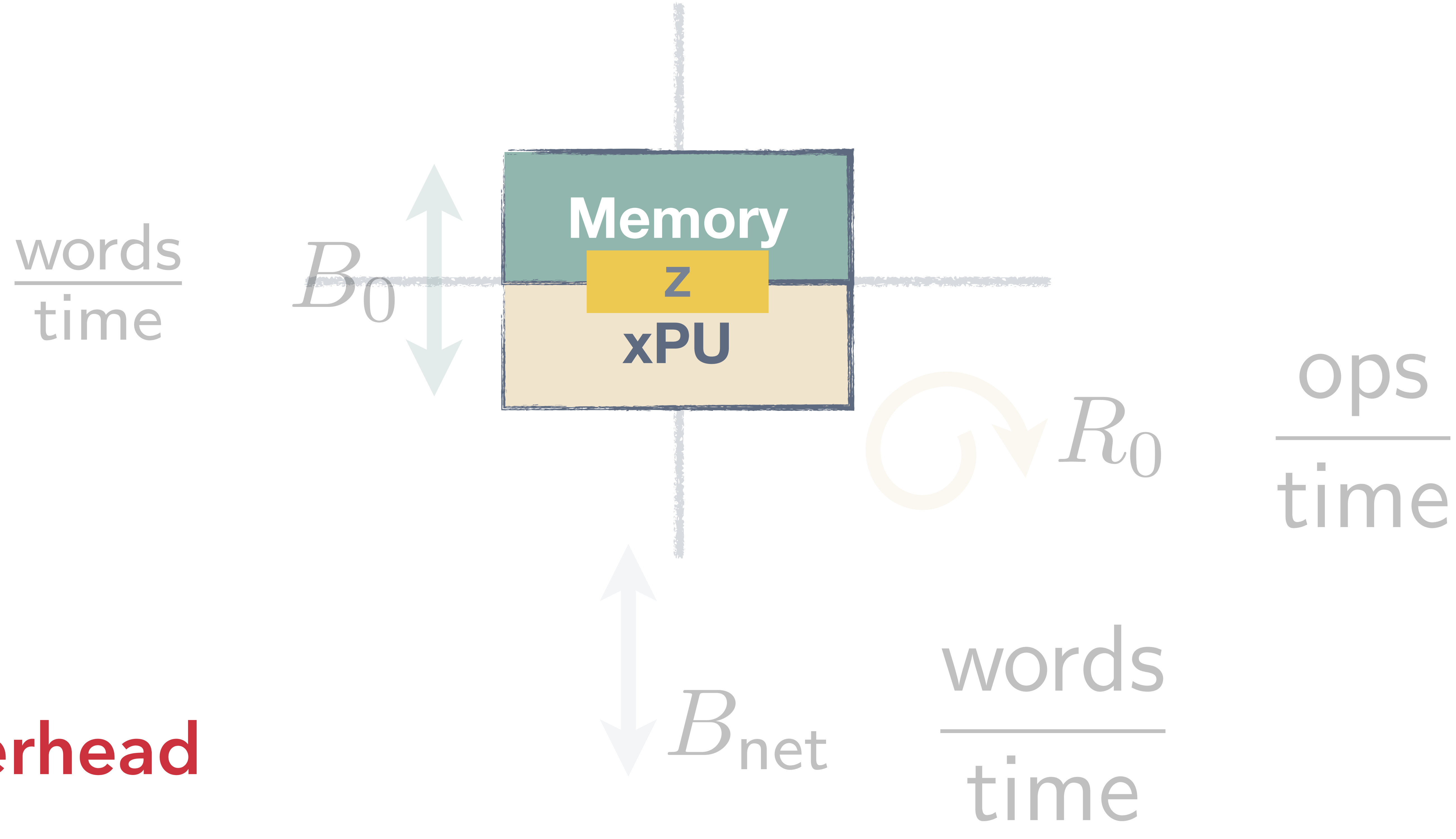
$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

network penalty

Network overhead

## Memory and network communication trade off!

(under strong scaling)




Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."

# An Iron Law of Parallel and Distributed Computation

# Higher is better

($\Rightarrow$ **weak** scaling)

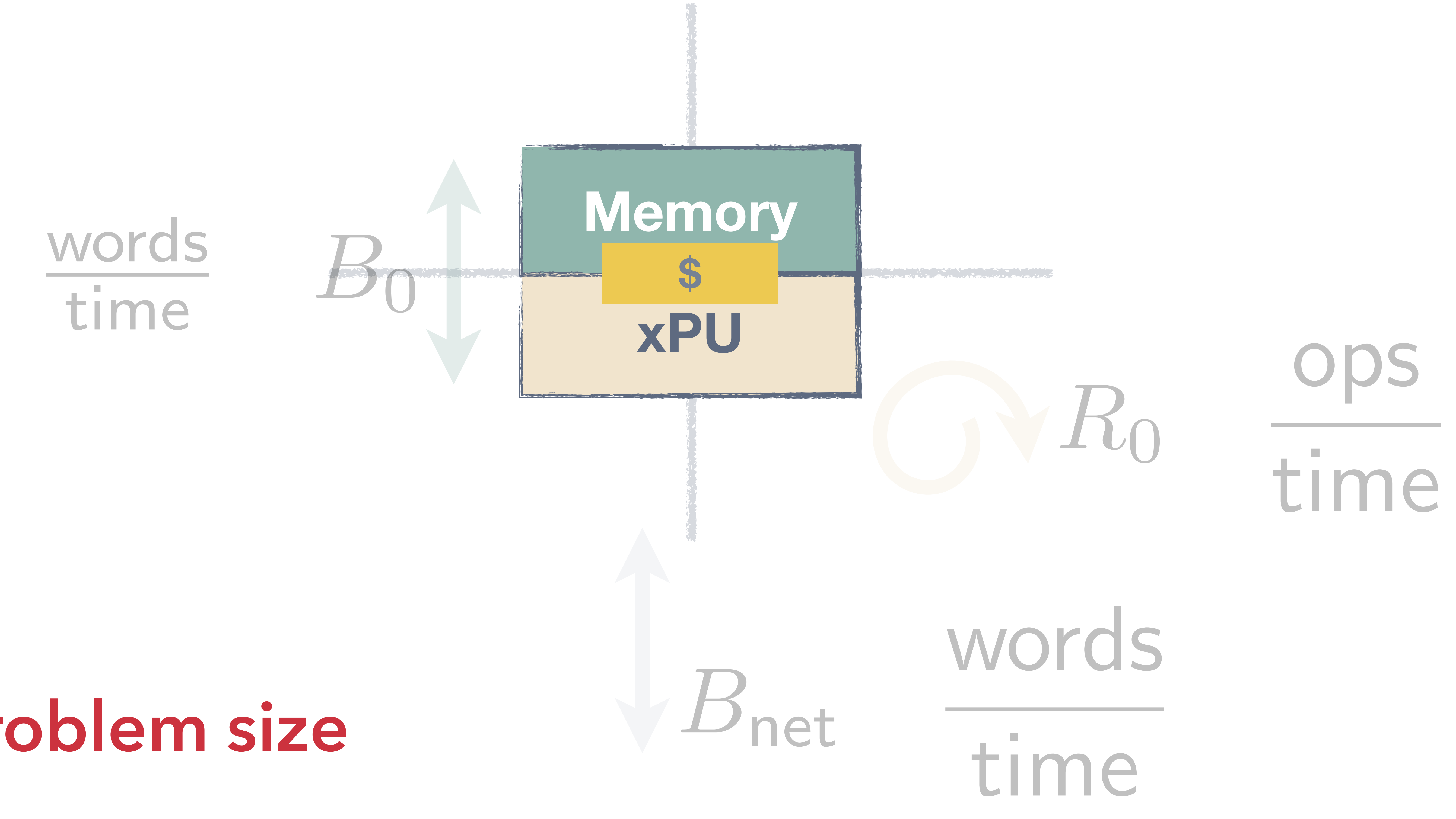
memory penalty

$$\frac{T_{\text{mem}}}{T_{\text{op}}} \approx \frac{R_0}{B_0} \cdot \frac{1}{g(Z)}$$

$$\frac{T_{\text{net}}}{T_{\text{op}}} \approx \frac{R_0}{B_{\text{net}}} \cdot \frac{h_1(P)}{h_2(n)}$$

network penalty

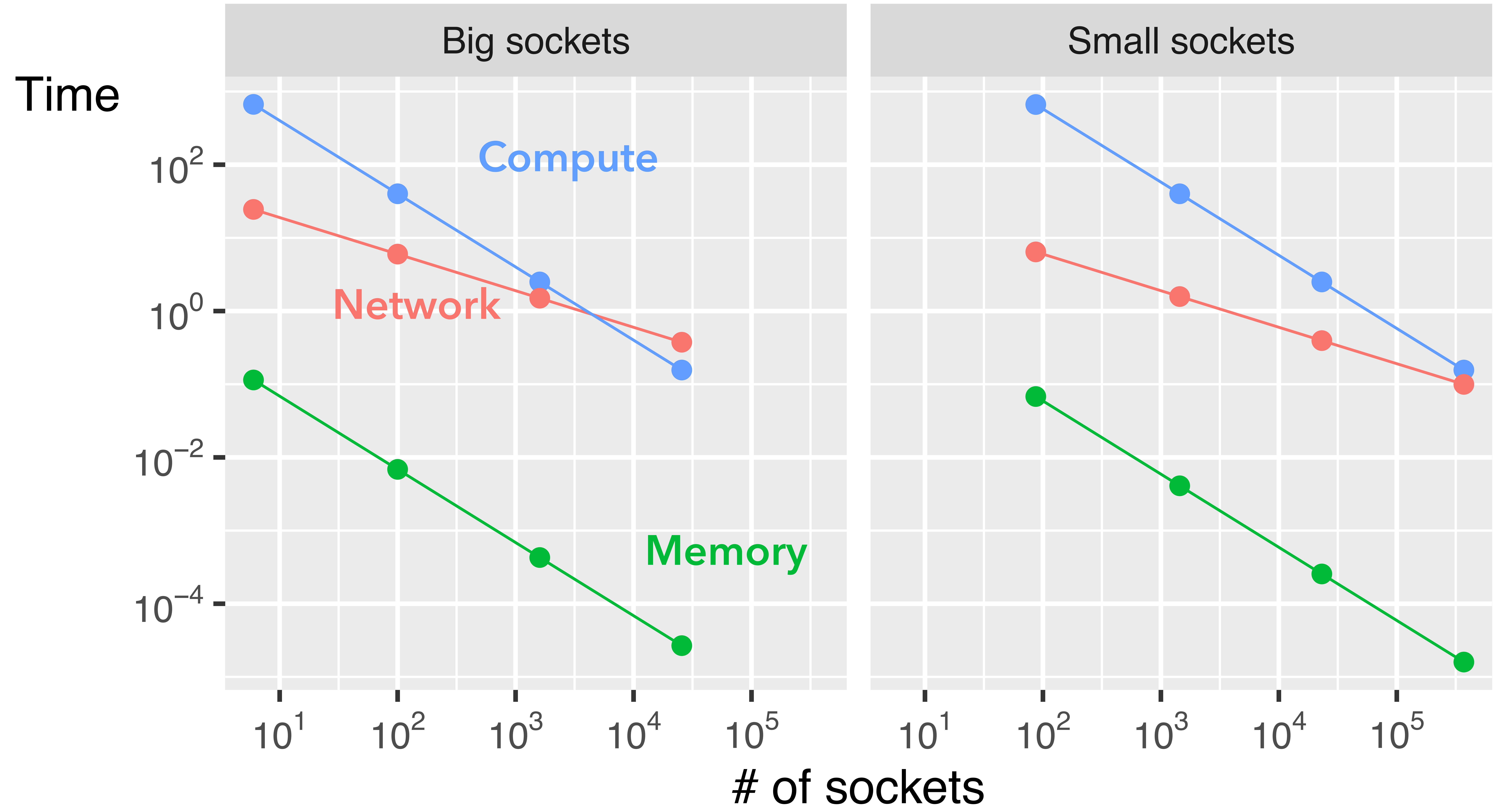
Per-node problem size



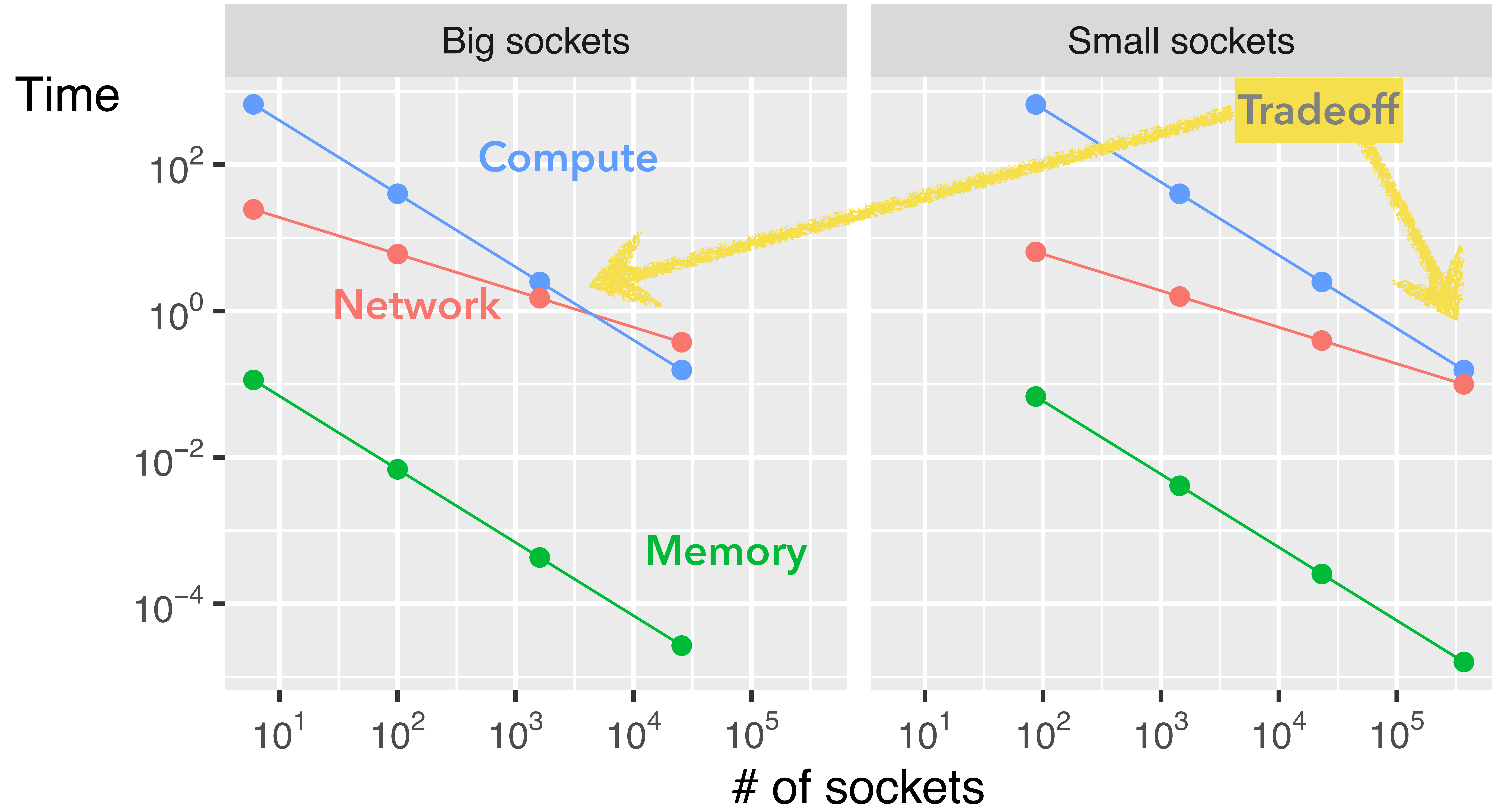



*Young & Vuduc (2016). "Finding balance in the post-Moore's Law era."*

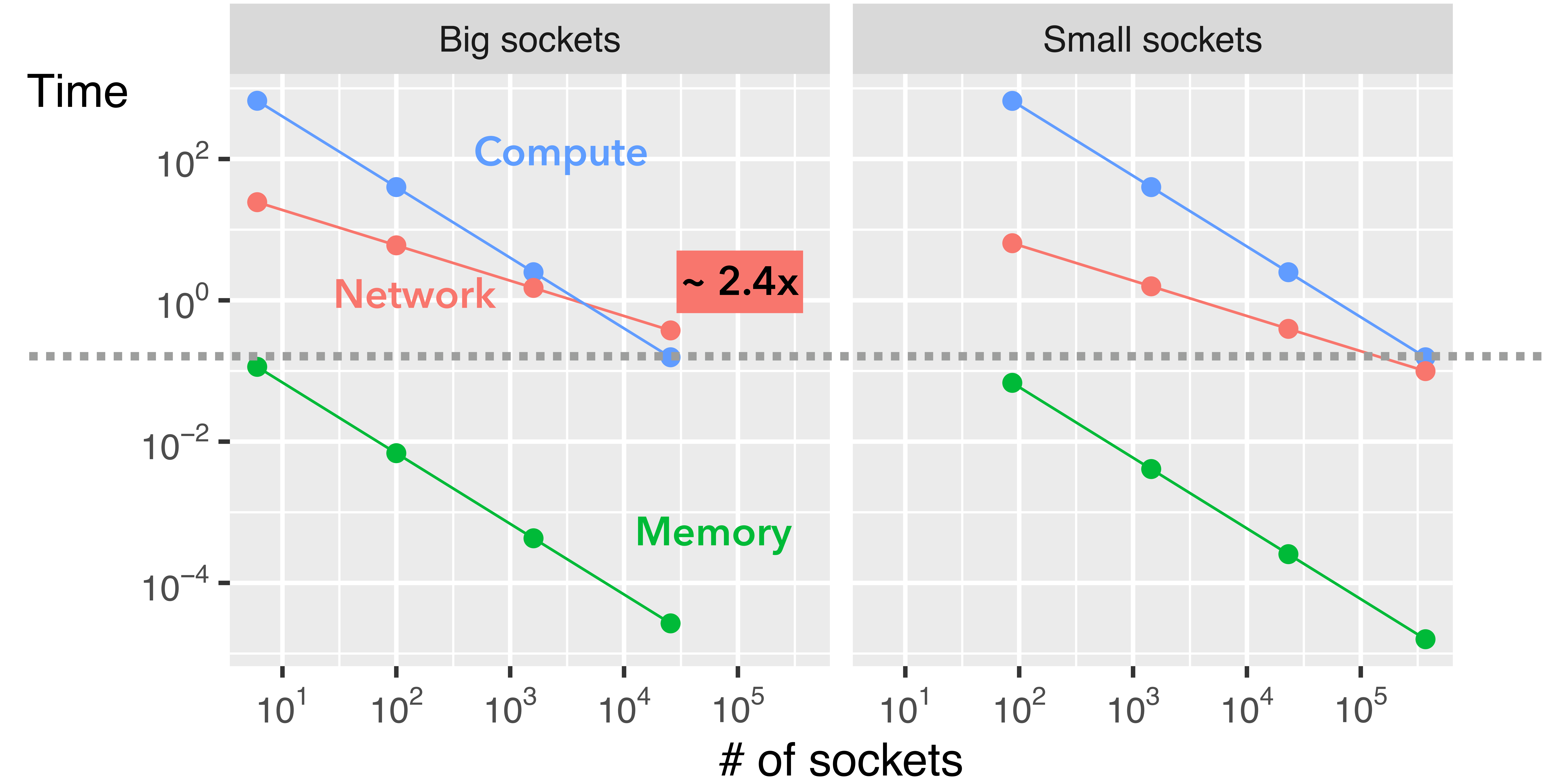
## **Matrix multiplication** (same peak op/s, strong scaling, ORNL Summit-like)



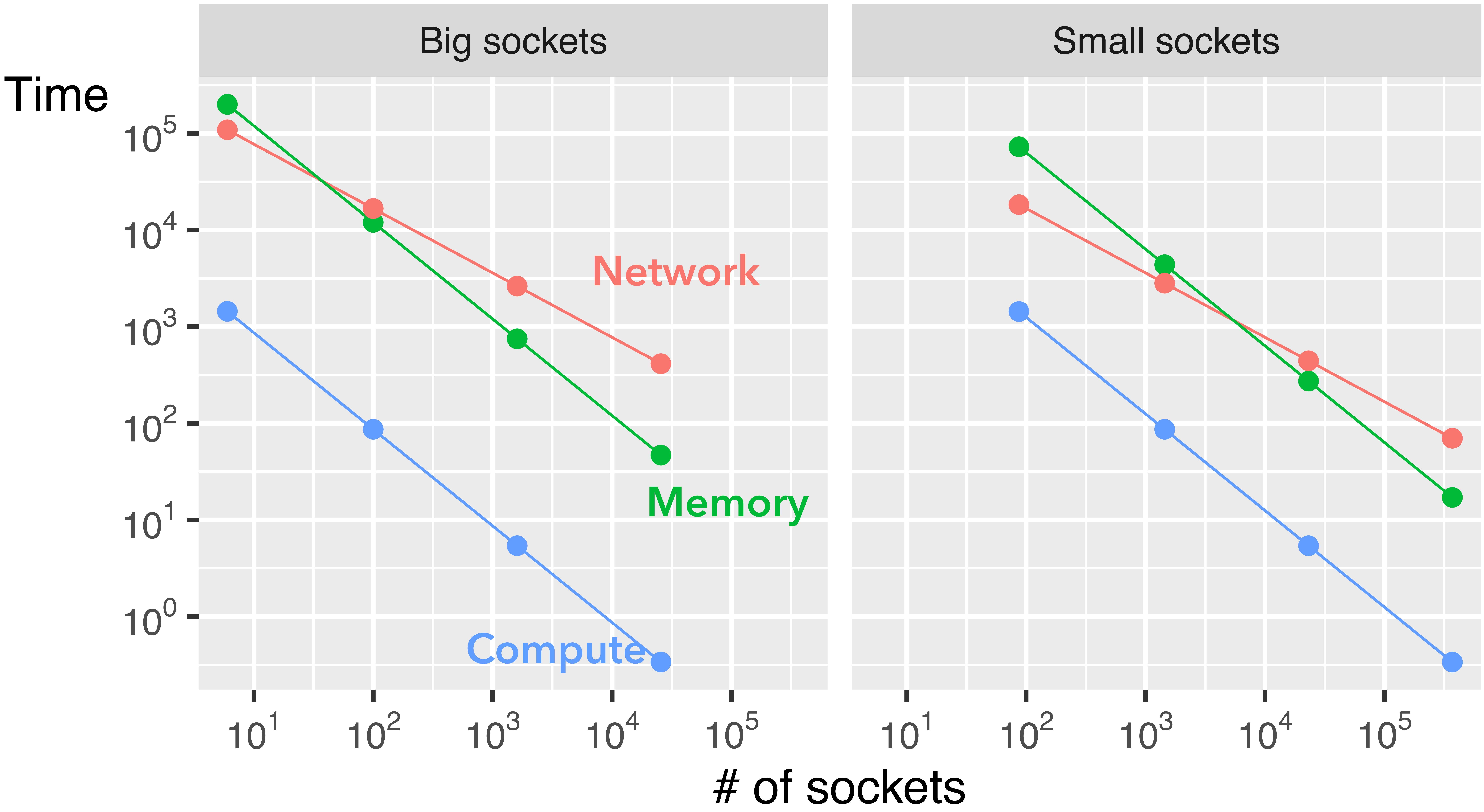
**Matrix multiplication** (same peak op/s, strong scaling, ORNL Summit-like)



**Matrix multiplication** (same peak op/s, strong scaling, ORNL Summit-like)

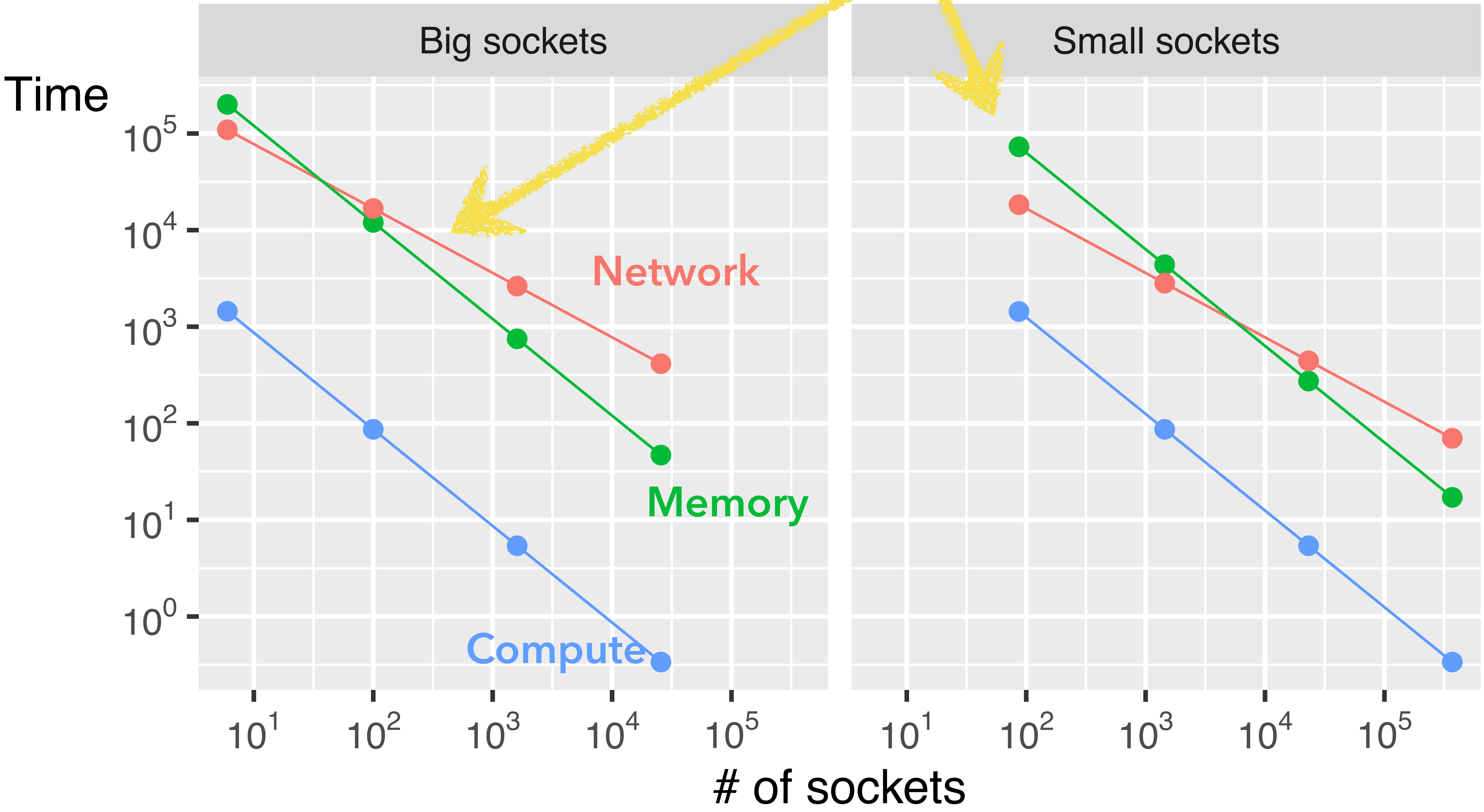


**3D FFT** (same peak op/s, strong scaling, ORNL Summit-like sockets)

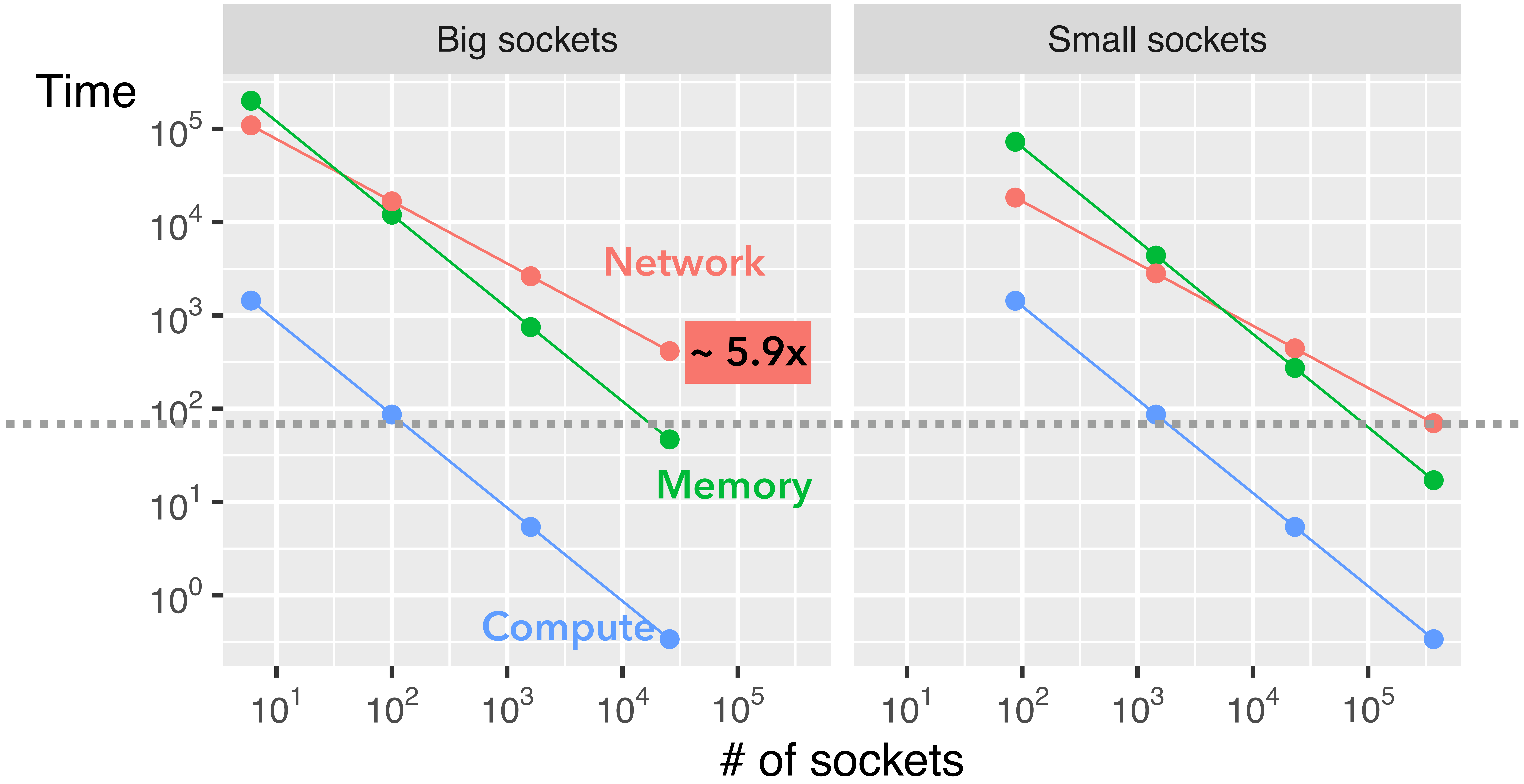


**3D FFT** (same peak op/s, strong scaling, ORNL Sum

Communication tradeoffs



**3D FFT** (same peak op/s, strong scaling, ORNL Summit-like sockets)



# “Communication is inevitable & system components help manage scalability tradeoffs.

CONCLUSION SO FAR?

# SmartNICs (DPUs) as DMXs (data-movement accelerators)

# DPU-DMXs in modern clusters

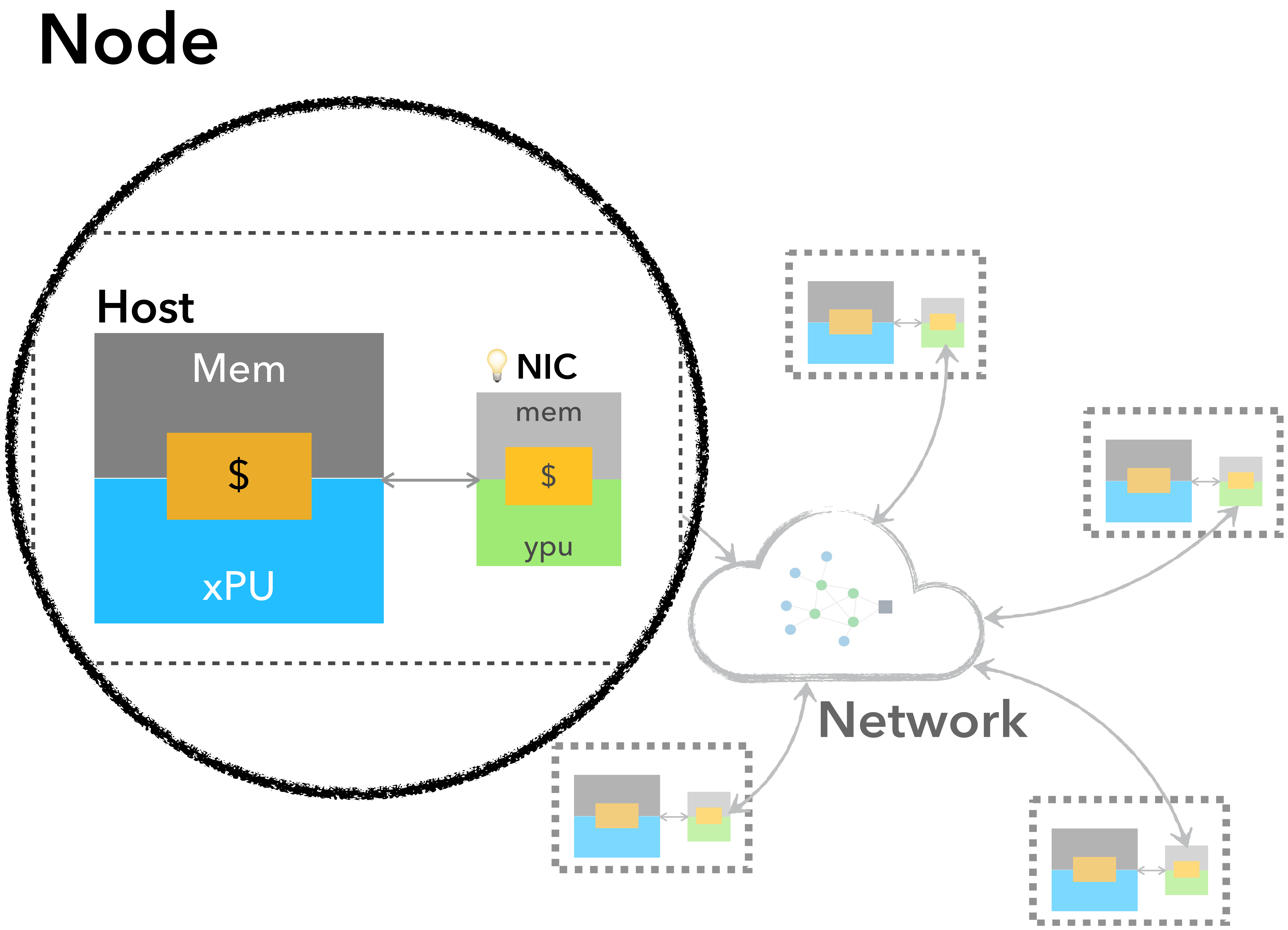
The basic building block of a distributed-memory cluster or supercomputer is a node.

Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.

In a **DPU**, the NIC becomes "**host-like**" via the addition of processing (ypu) and memory.



# Uses of DPUs, actual and envisioned

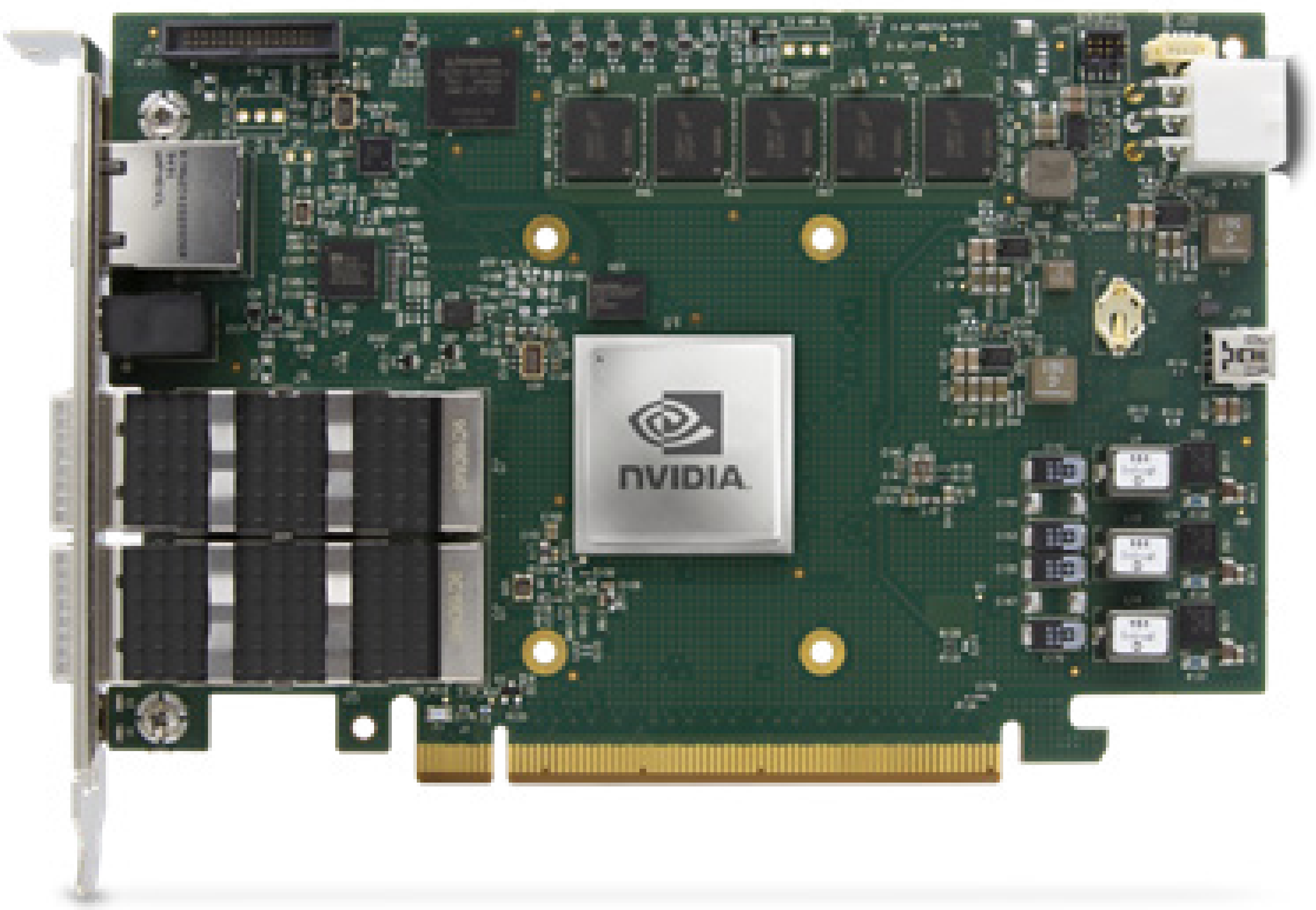
OPPORTUNITIES AND PITFALLS: SEE **GRANT ET AL. IPDRM'20 SURVEY**, "RADD RUNTIMES"

- **Accelerating network applications**: packet classification, traffic shaping, multicasting
- **Network, storage, and sensor algorithms**, e.g., distributed key-value stores, consensus algorithms, computer vision
- **Advanced runtimes**: e.g., distributed resource and I/O management, fault tolerance (See Ryan et al. IPDRM'20)
- **HPC algorithms?**

# DPU algorithms

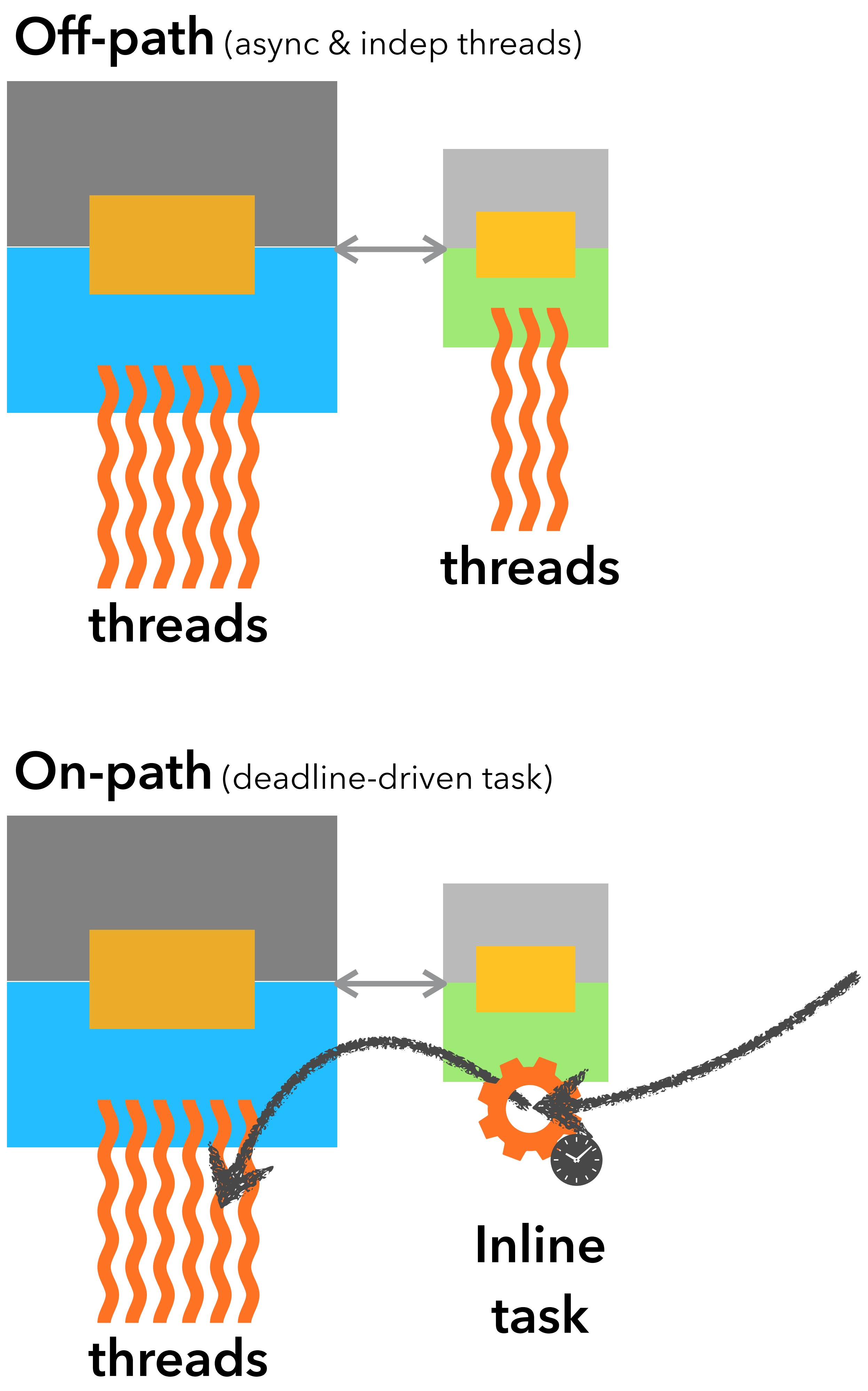
- MiniMD ( S. Karamati et al., "“Smarter” NICs for faster molecular dynamics: a case study," IPDPS, 2022 )
- Maxwell’s equation (Current work)

# Our focus




- **Platform**: **DPUs** like BF2 & BF3, which are based on general-purpose multicore CPUs
- Usage model: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our focus




- Platform: DPUs like BF2 & BF3, which are based on general-purpose multicore CPUs
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our focus

**OpenSNAPI**

- OpenSNAPI is a project of the UCF Consortium
- Straight from the source:
  - *"OpenSNAPI is a collaboration between industry, laboratories and academia with the goal to create a standard application programming interface (API) for accessing the compute engines on the network, and specifically on the smart network adapter. OpenSNAPI allows application developers to leverage the network compute cores in parallel to the host compute cores for accelerating application runtime, and to perform operations and processing closer to the data."*

- Platform: DPUs like BF2 & BF3, which are based on general-purpose multicore CPUs
- Usage model: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation)
- **Programming model**: **Multiprogram MPI** with DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

Sara Karamati

# A MiniMD case study

*"In theory, theory and practice are the same. In practice, they are not."*

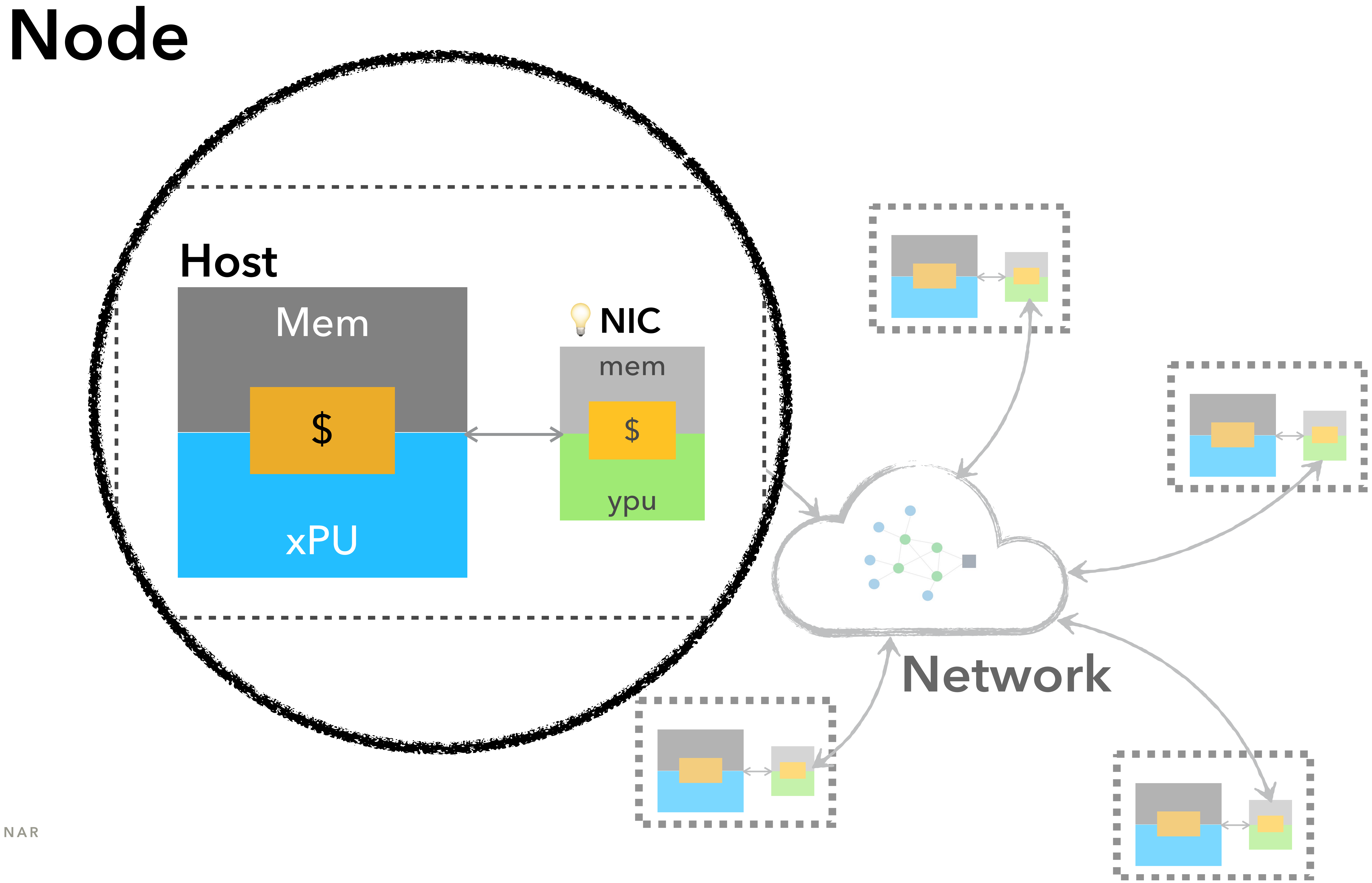


**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati



# Target platform: NVIDIA (née Mellanox) BF2



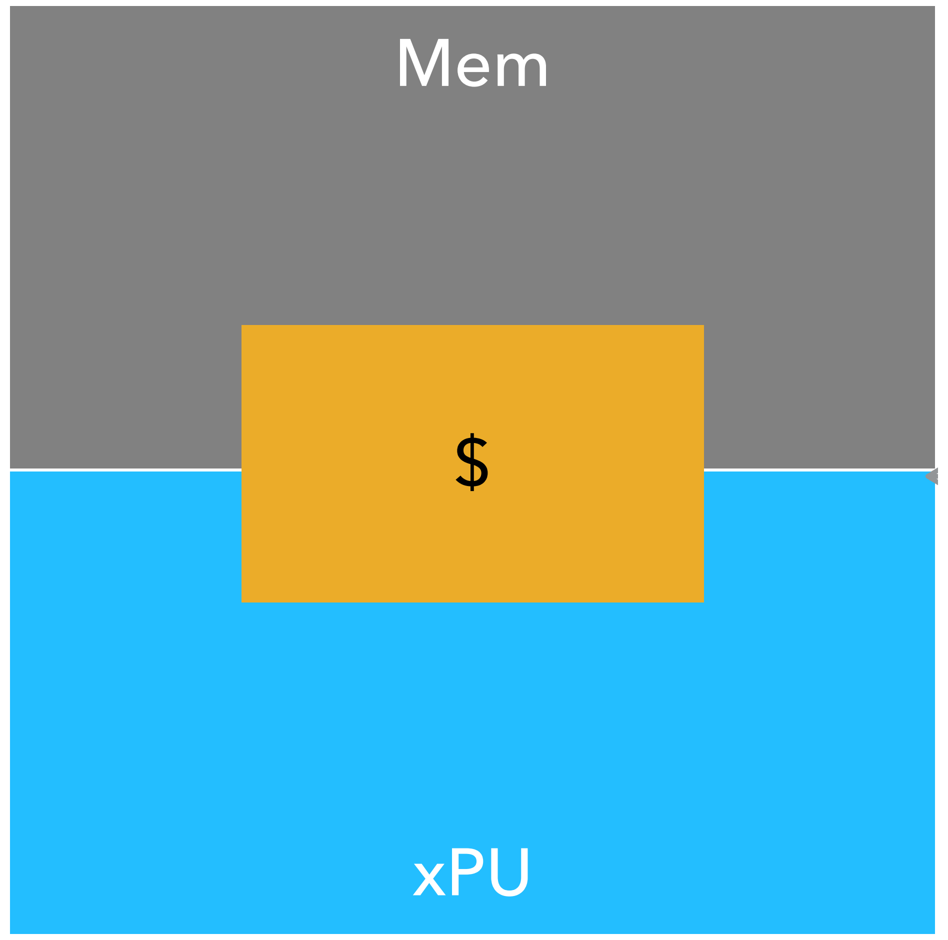



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati

# Target platform: NVIDIA (née Mellanox) BF2

## One host xPU (16 cores)



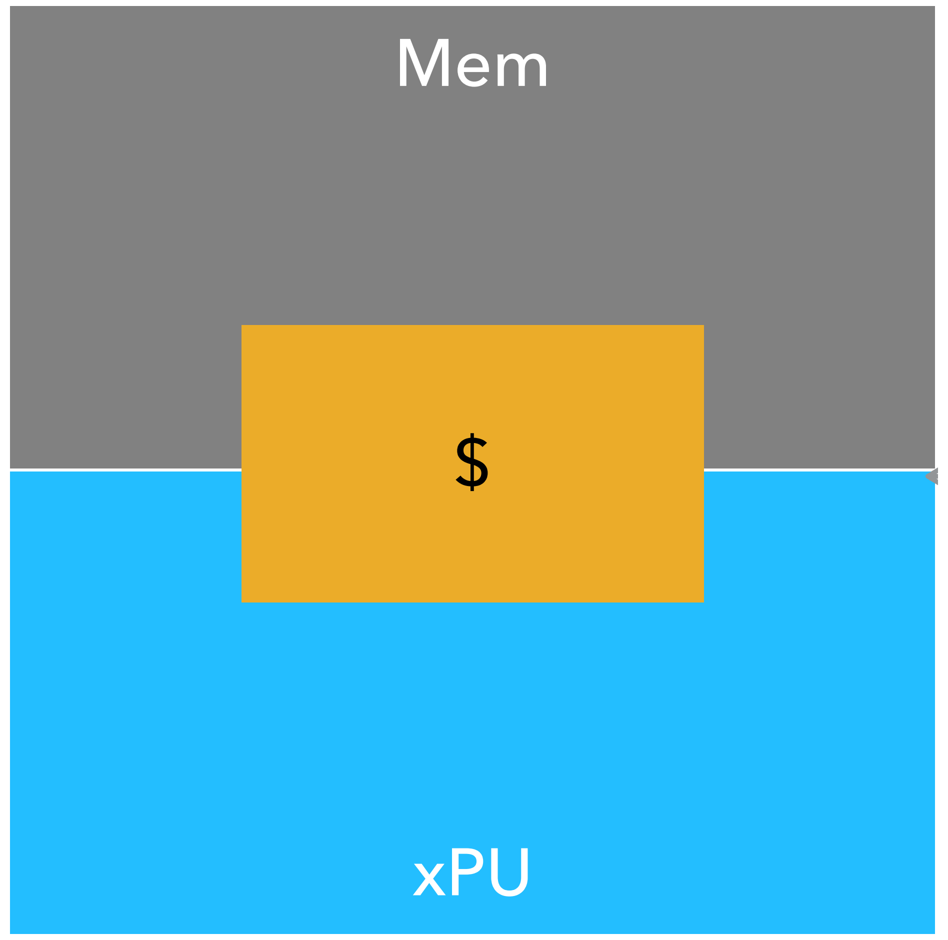



**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati

# Target platform: NVIDIA (née Mellanox) BF2

## One host xPU (16 cores)




**657 GF/s** (fp64)



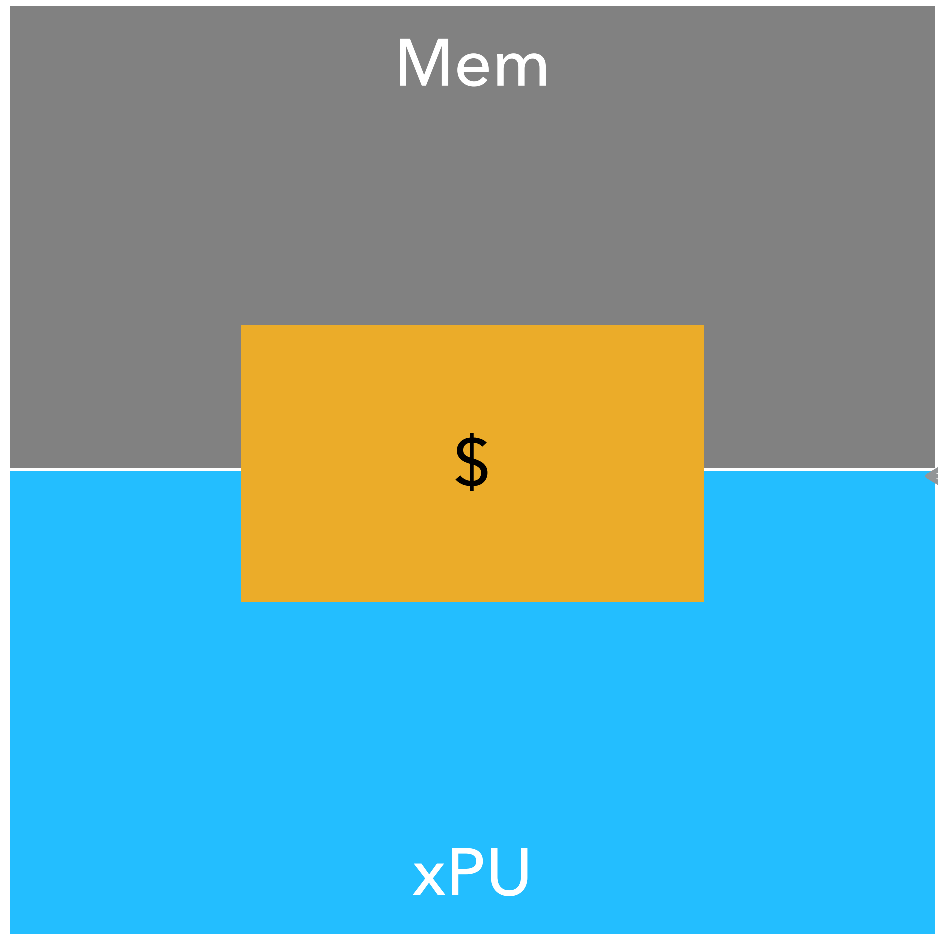
**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati



# Target platform: NVIDIA (née Mellanox) BF2

## One host xPU (16 cores)




**657 GF/s** (fp64)

**76.8 GB/s**

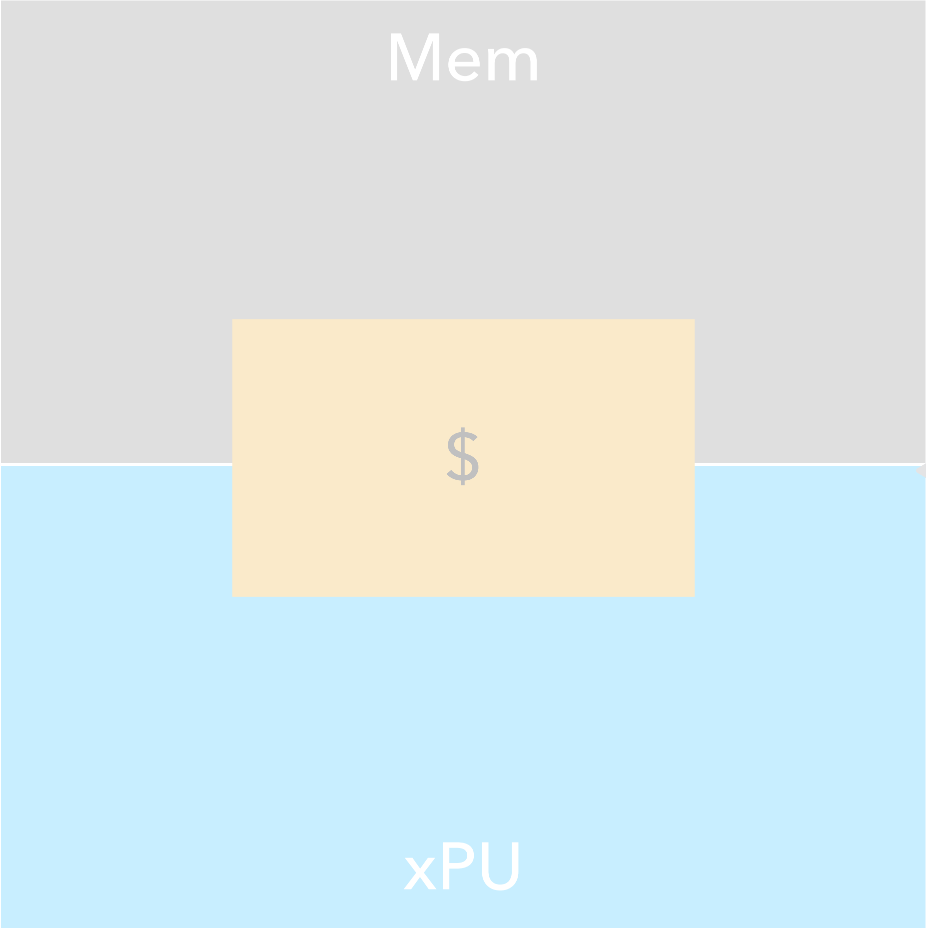


S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati

# Target platform: NVIDIA (née Mellanox) BF2

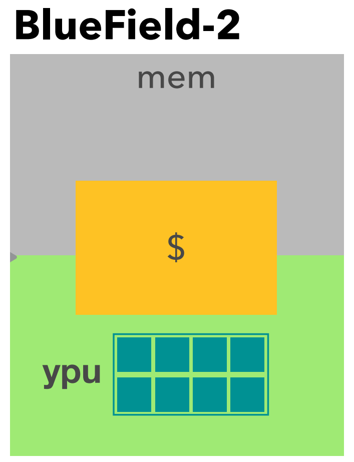
One host xPU (16 cores)




657 GF/s (fp64)

76.8 GB/s

BF-2 yPUs (no host)





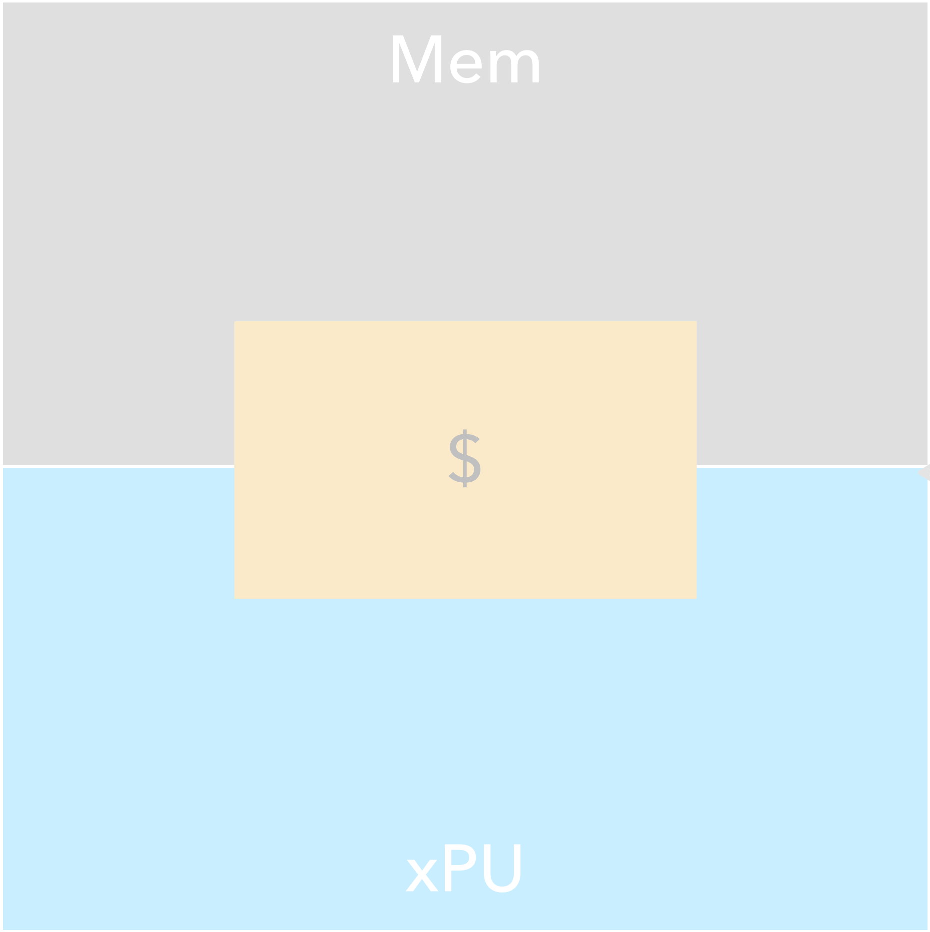

S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Sara Karamati



# Target platform: NVIDIA (née Mellanox) BF2

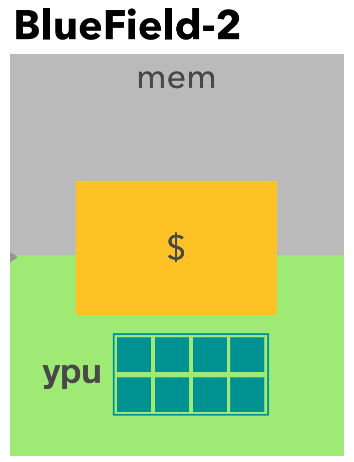
## One host xPU (16 cores)




657 GF/s (fp64)

76.8 GB/s

## BF-2 yPUs (no host)

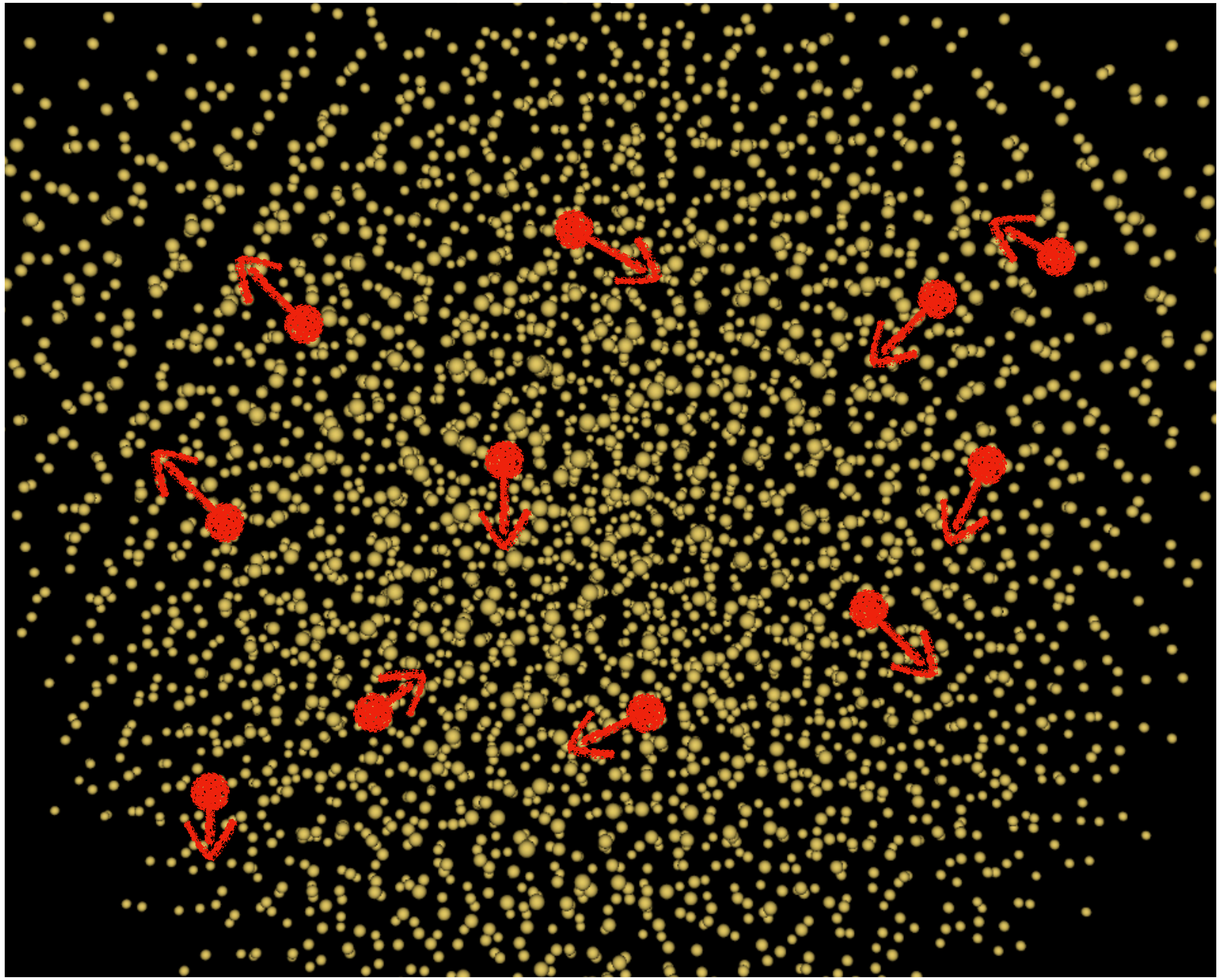



80 GF/s

25.6 GB/s



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

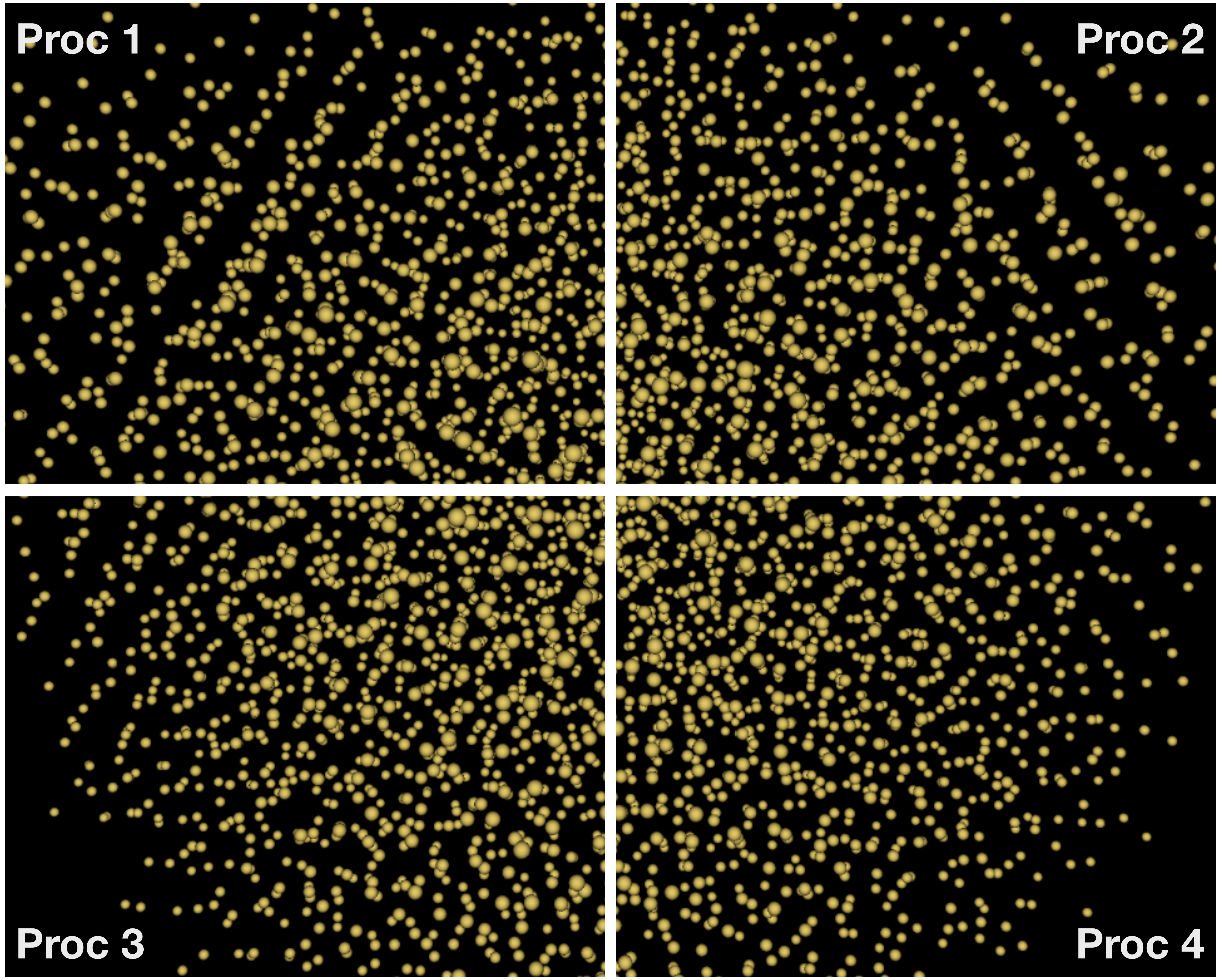


# Baseline MiniMD

MiniMD is a molecular dynamics proxy-app. It calculates the position and velocity of a set of interacting particles in discrete time steps (iterations).



**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063




# Baseline MiniMD

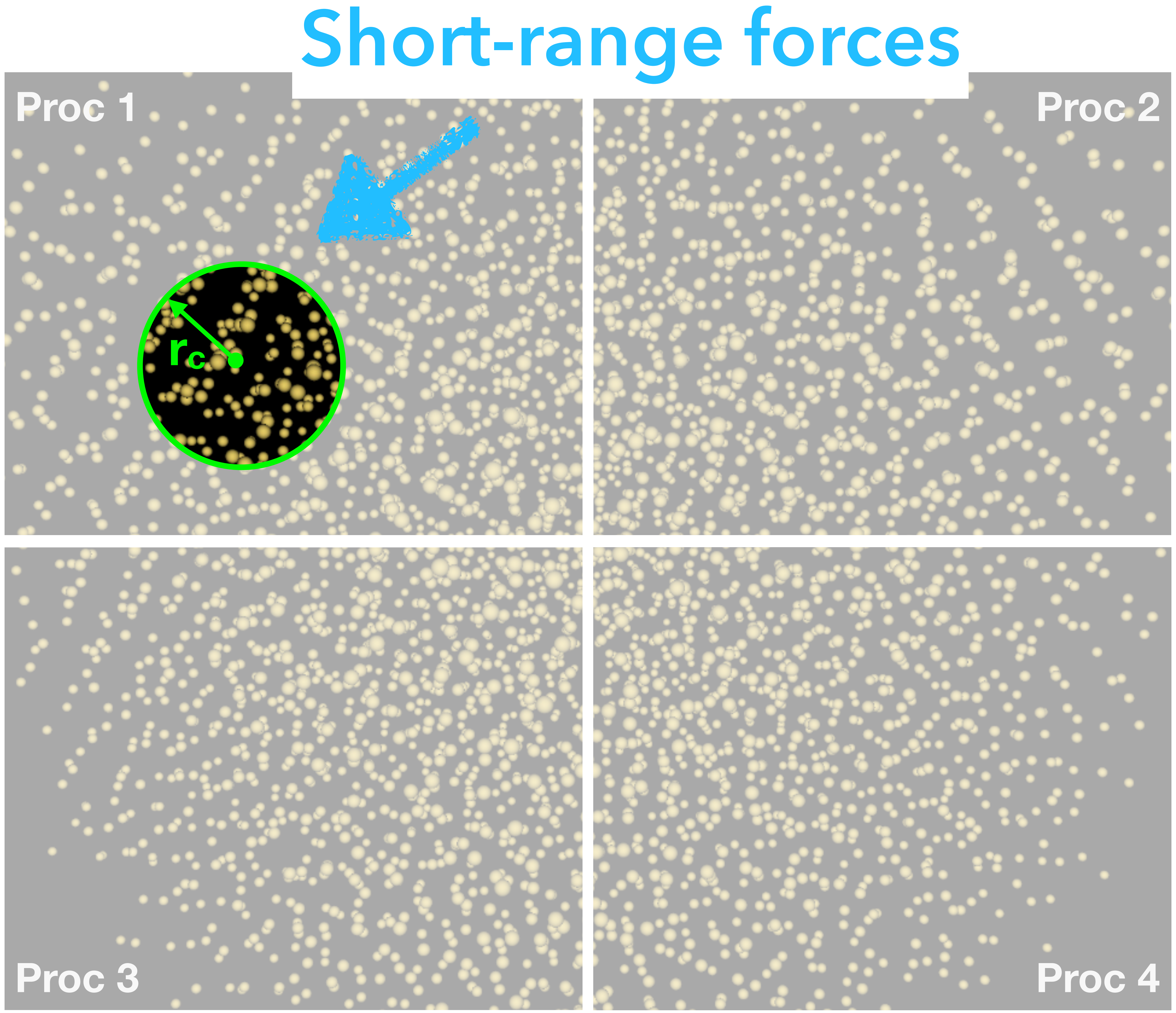
MiniMD is a molecular dynamics proxy-app. It calculates the position and velocity of a set of interacting particles in discrete time steps (iterations).

In the distributed-memory setting, the simulation domain is divided spatially among MPI processes.

Every process owns its particles, computes force on these particles and then updates the position and velocity of these particles.



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063




# Baseline MiniMD

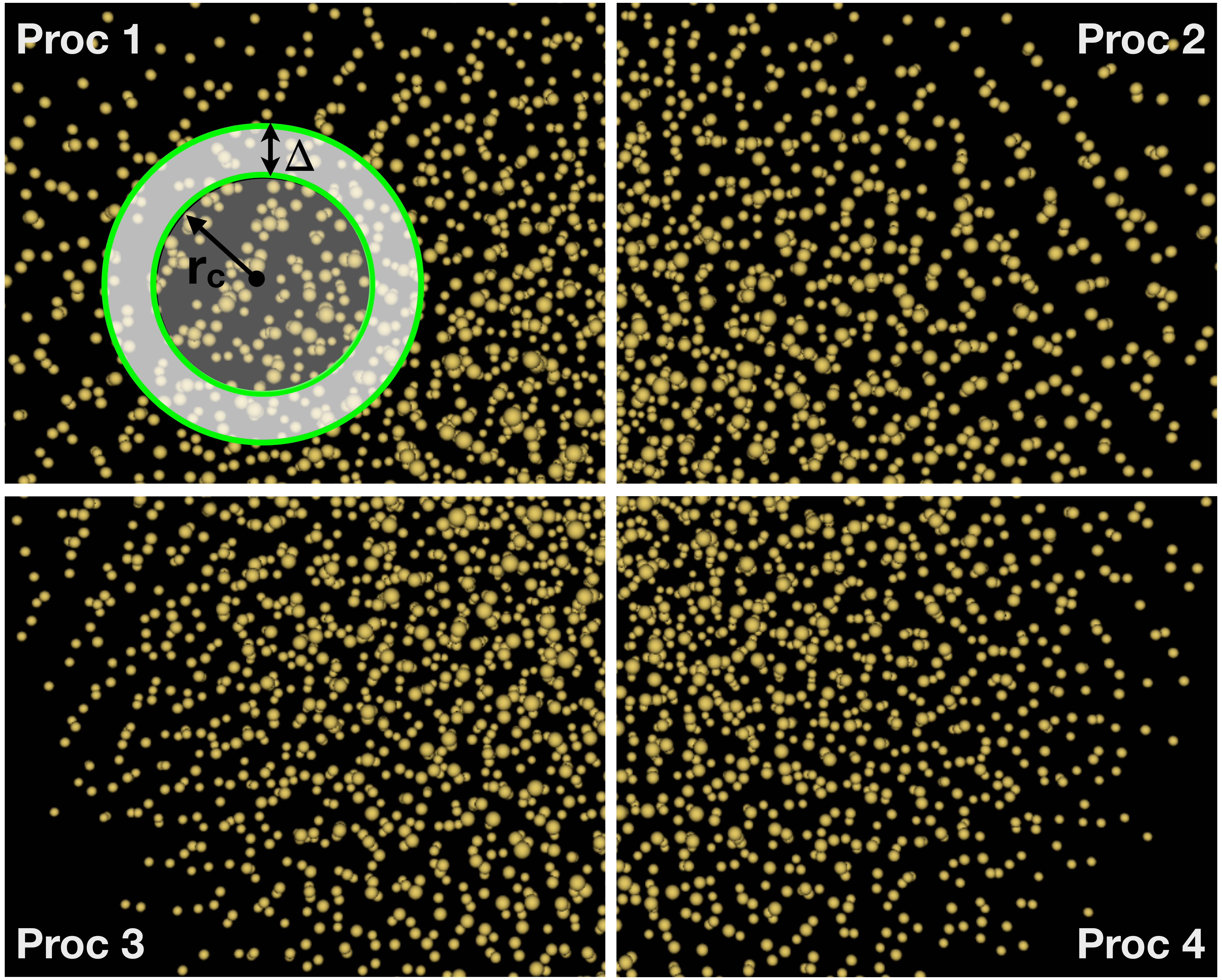
In each iteration, every particle interacts with others that lie within a some **cutoff distance** ($r_c$). A particle's **neighbor list** stores references to them.

***Recall:***

$$\mathcal{O}(N^2) \longrightarrow \mathcal{O}(N)$$



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063



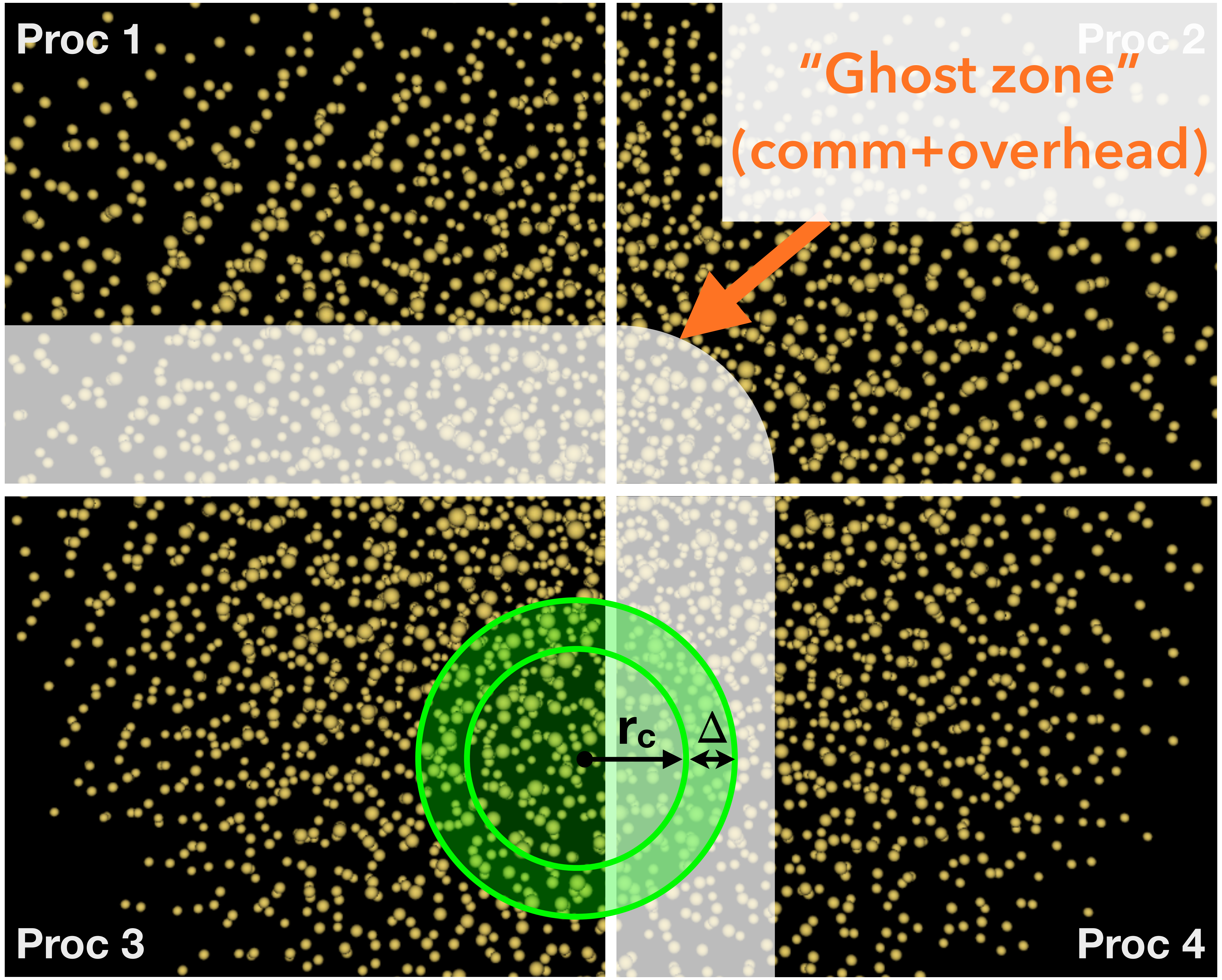

# Baseline MiniMD

In each iteration, every particle interacts with others that lie within a some **cutoff distance** ($r_c$). A particle's **neighbor list** stores references to them.

The neighbor list must be updated as particles move. But such **updates are expensive**! So every list includes a buffer of "extra" particles that lie within a surrounding annulus, or "**skin**," parameterized by its thickness (Δ).



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063




# Baseline MiniMD

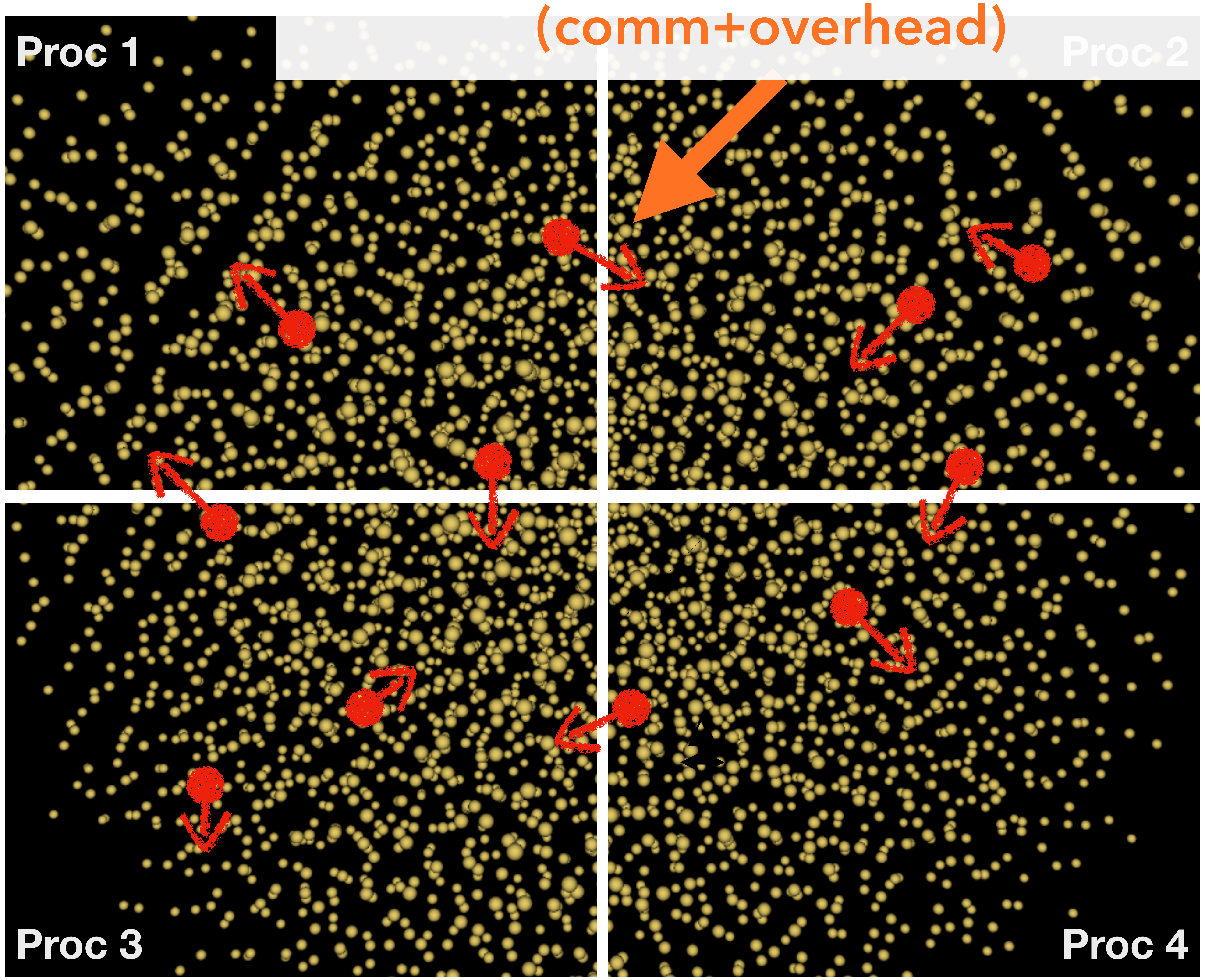
The cutoff distance ($r_c$) and skin thickness (Δ) imply the size of the interaction region just outside the boundaries of each process.

Each process keeps a copy of particles in that region.



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

Particles move through subdomains (comm+overhead)




# Baseline MiniMD

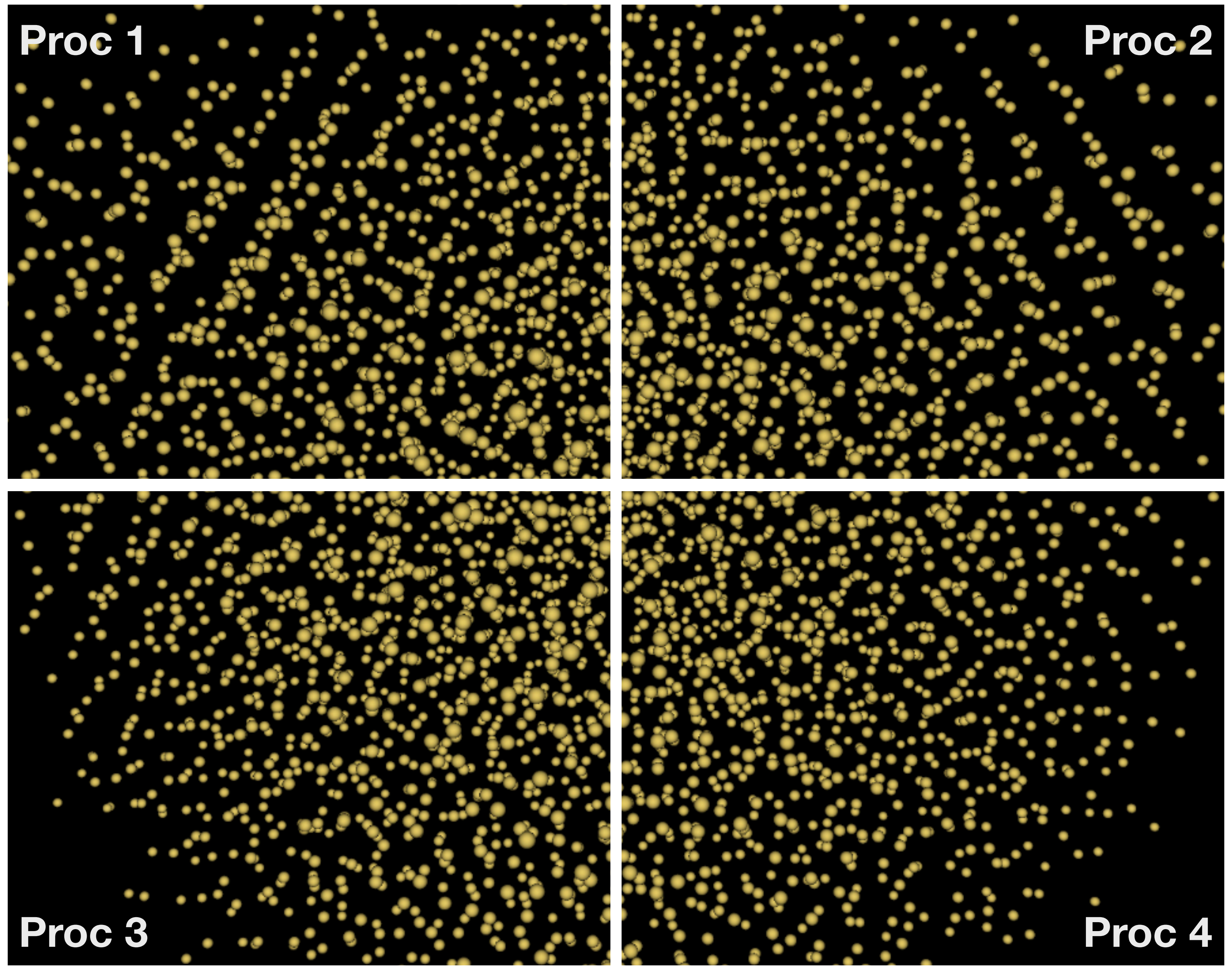
Particles are reassigned to new processes as they move through the spatial domain.

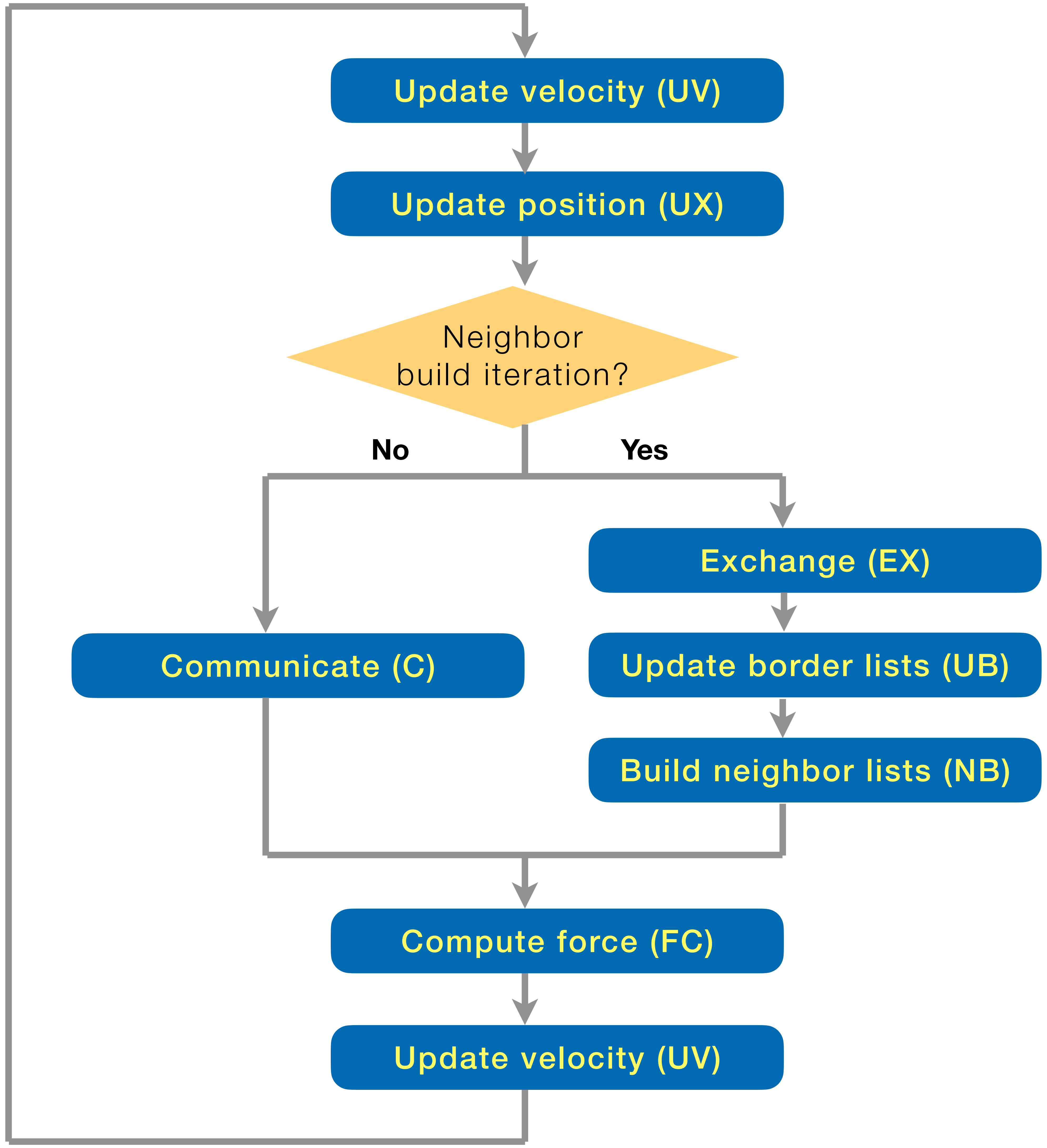
Neighbor list updates, boundary region exchanges, and particles reassignment to processes are triggered every so often via a user-selected parameter (e.g., every k iterations).



**S. Karamati (GT) et al. (Sandia, Queens U.; 2022)** – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Baseline MiniMD



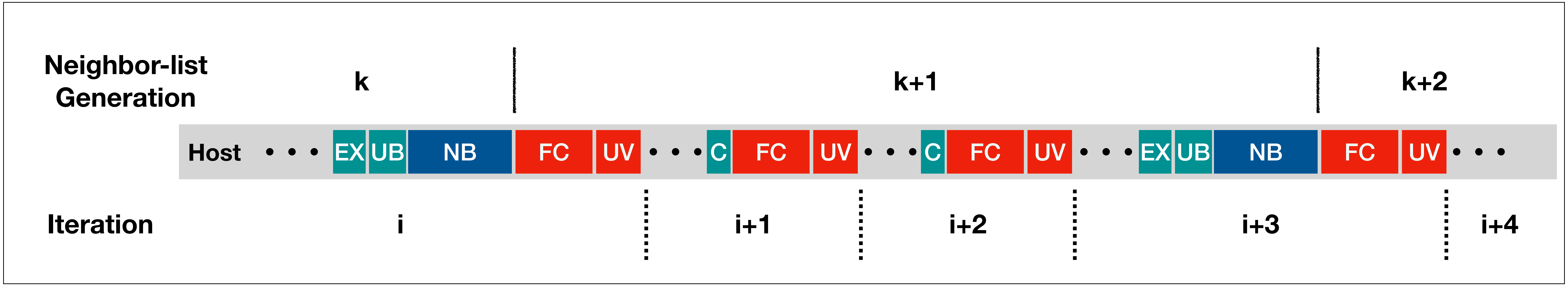




Each task is **parallelizable** but the sequence is **sequential** as shown by edges



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

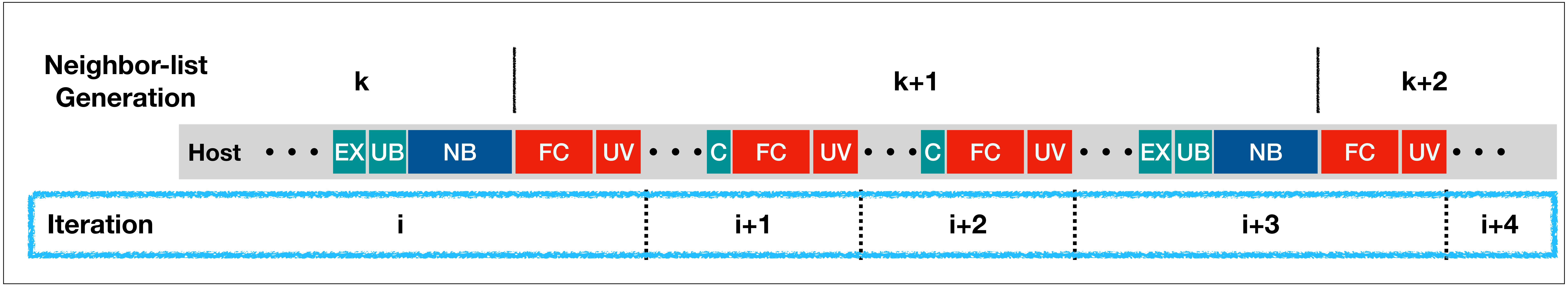
# Serial dependencies






S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

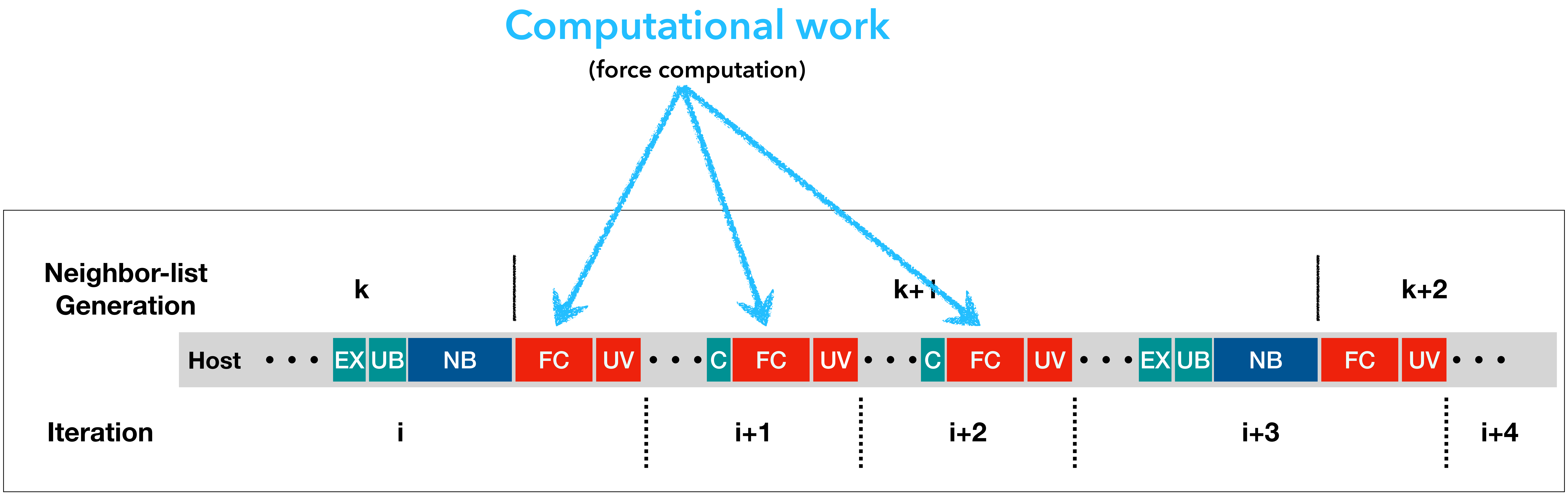
# Serial dependencies






**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Serial dependencies

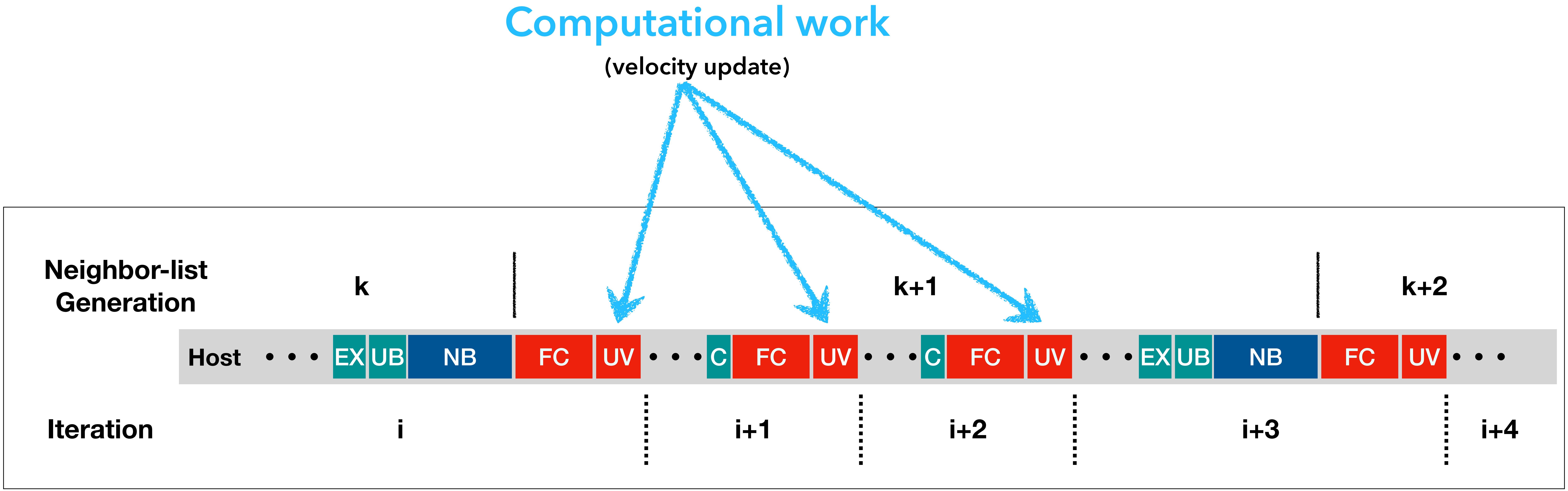





S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

FC work: force computation

# Serial dependencies



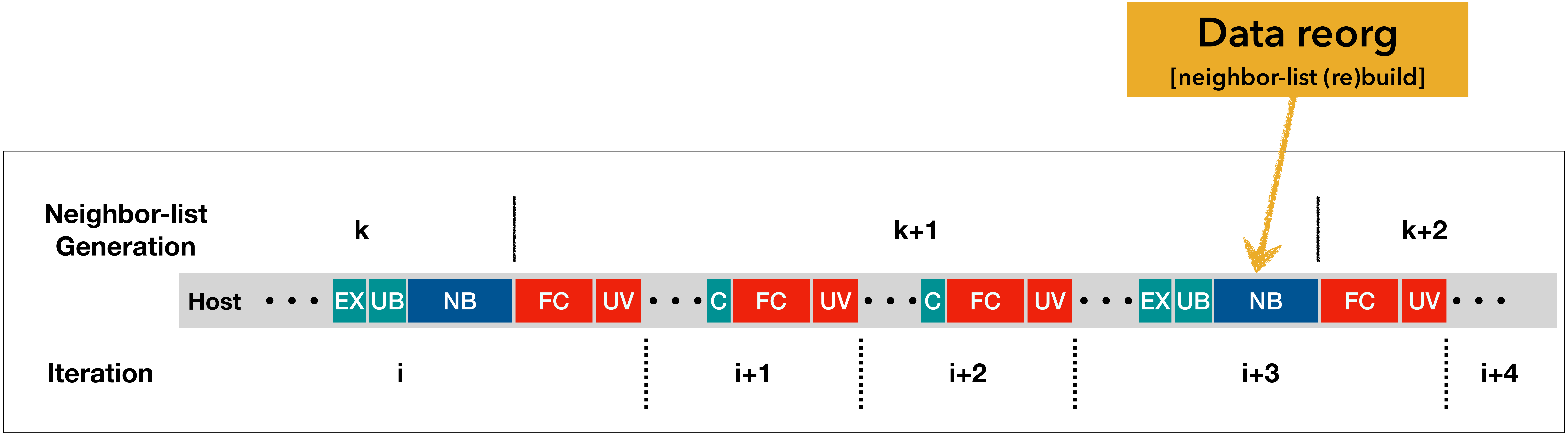



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

FC work: force computation

UV work: update velocity

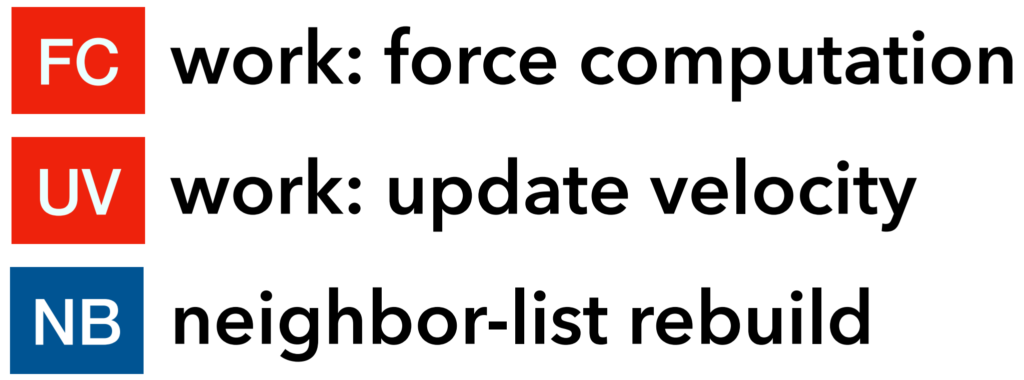
# Serial dependencies

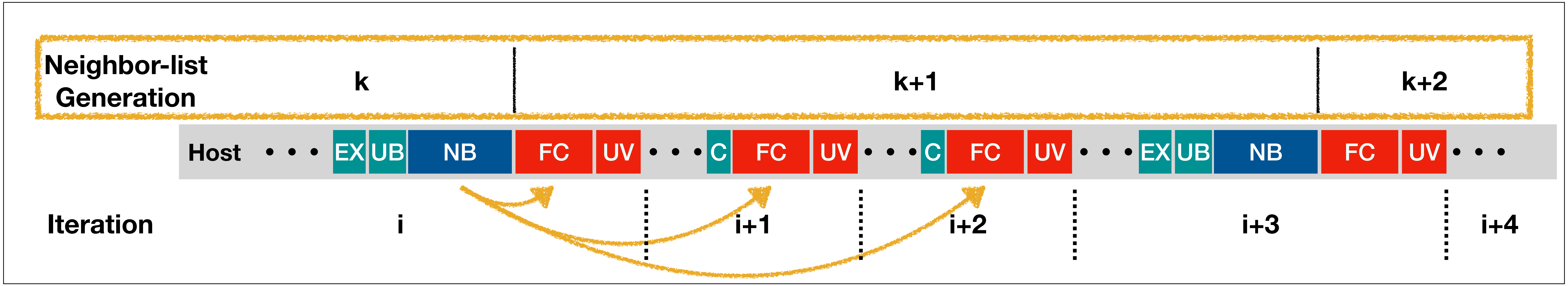





S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Serial dependencies

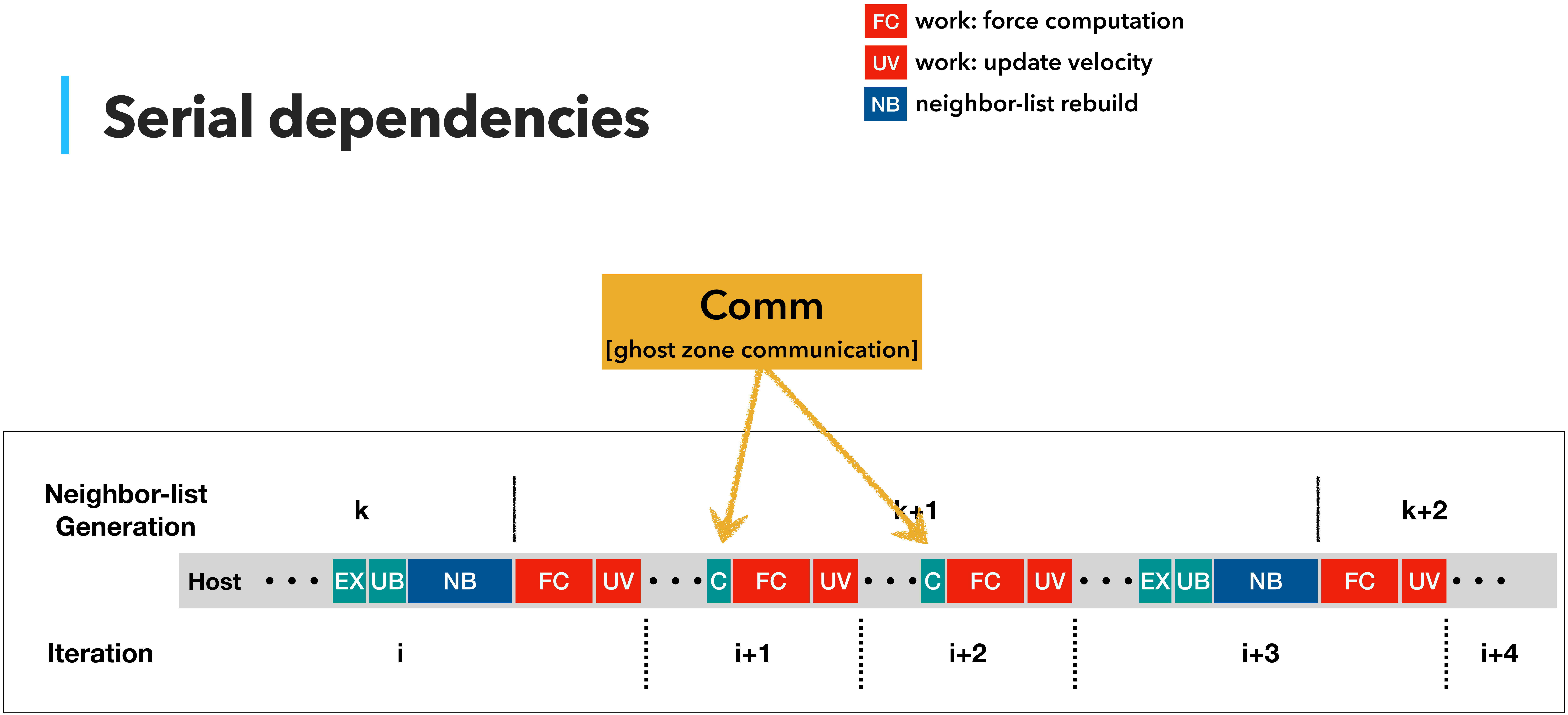





S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

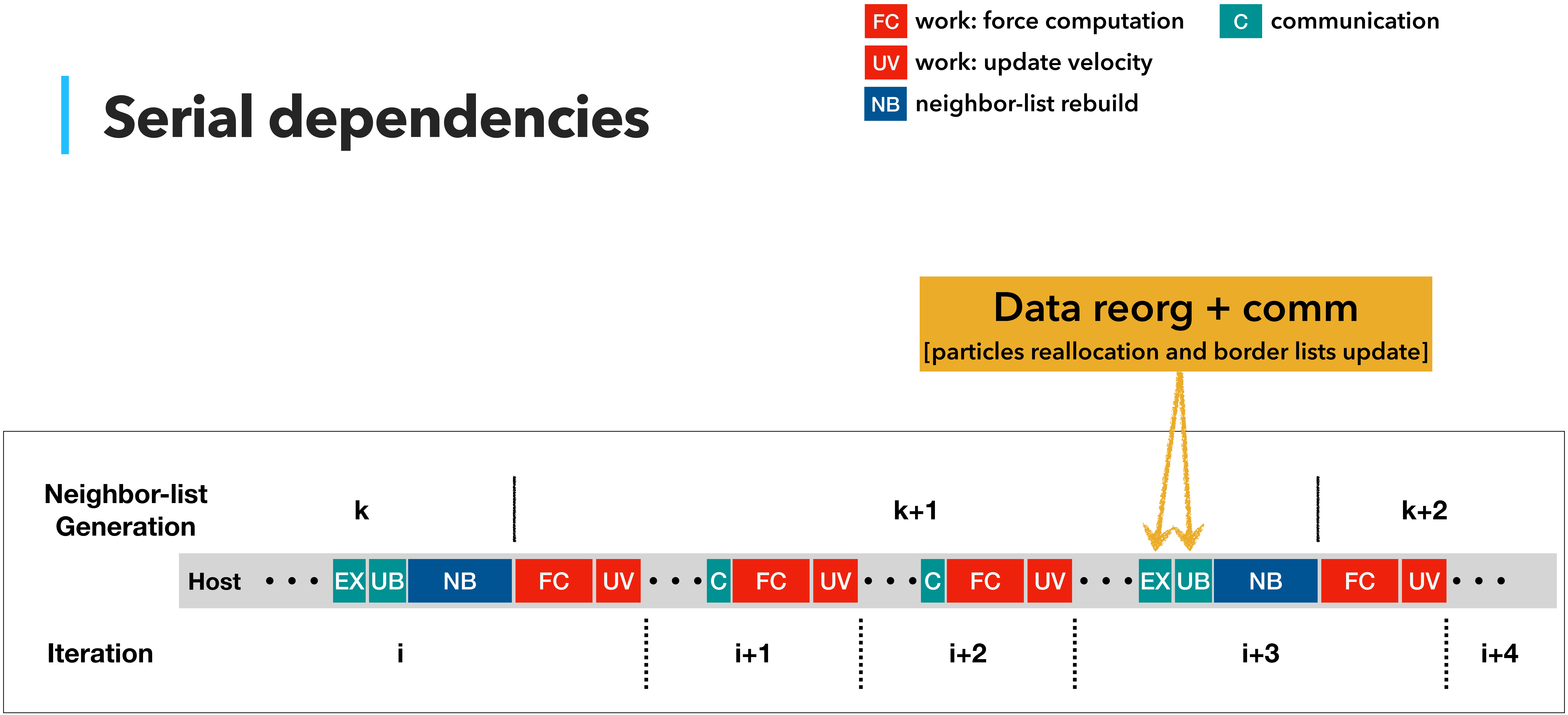
# Serial dependencies






S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Serial dependencies

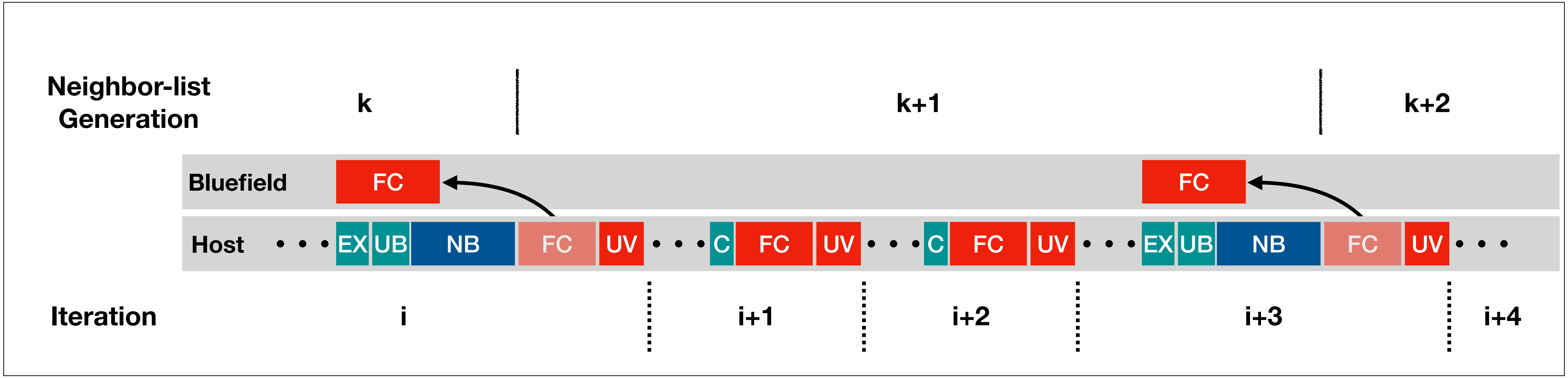





S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Breaking the dependencies
("Off-path" algorithm)

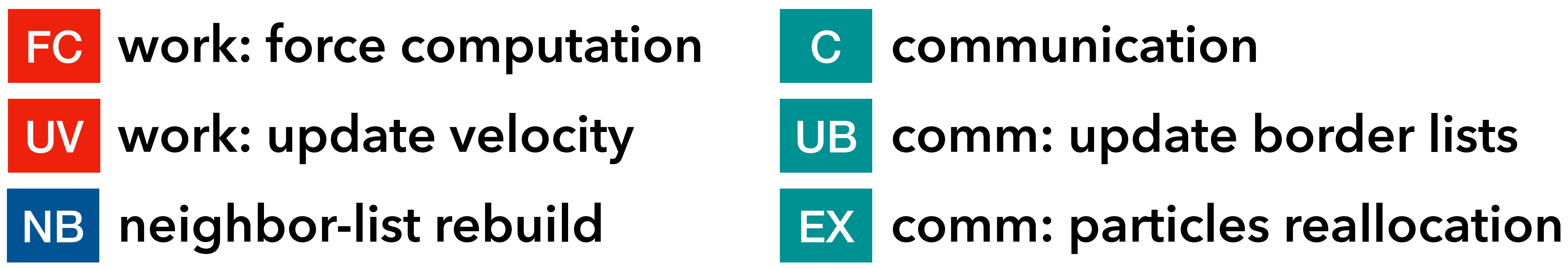
FC work: force computation
UV work: update velocity
NB neighbor-list rebuild
C communication
UB comm: update border lists
EX comm: particles reallocation

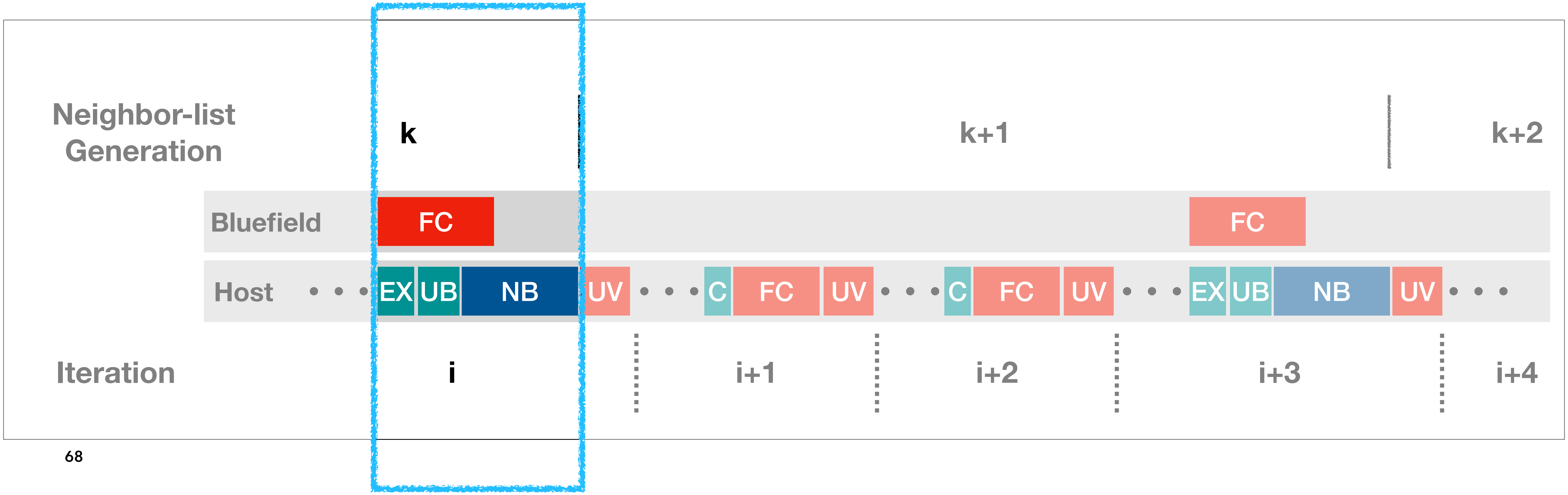





S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Breaking the dependencies
("Off-path" algorithm)

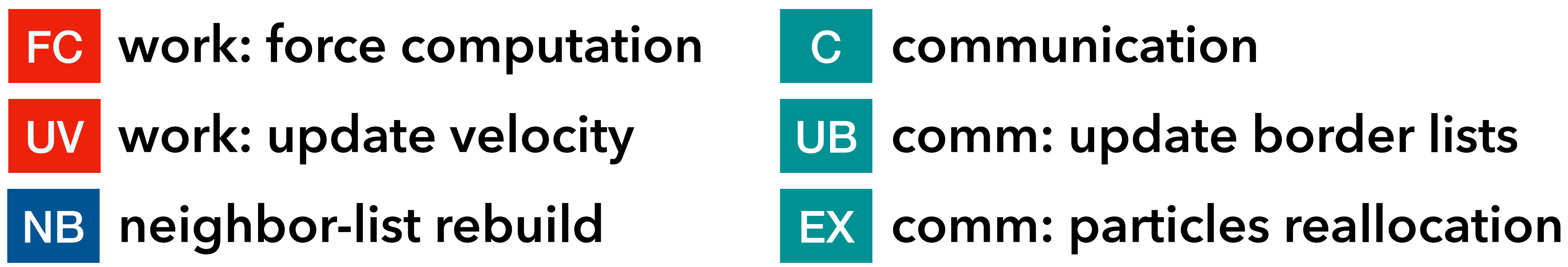


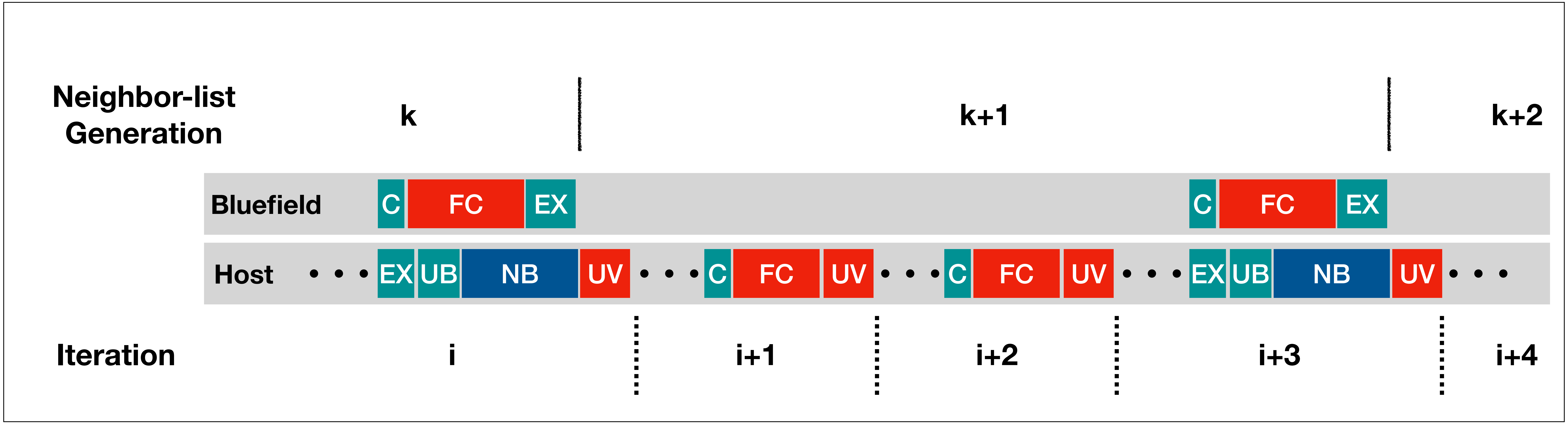






S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Breaking the dependencies
("Off-path" algorithm)






S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Baseline experiments

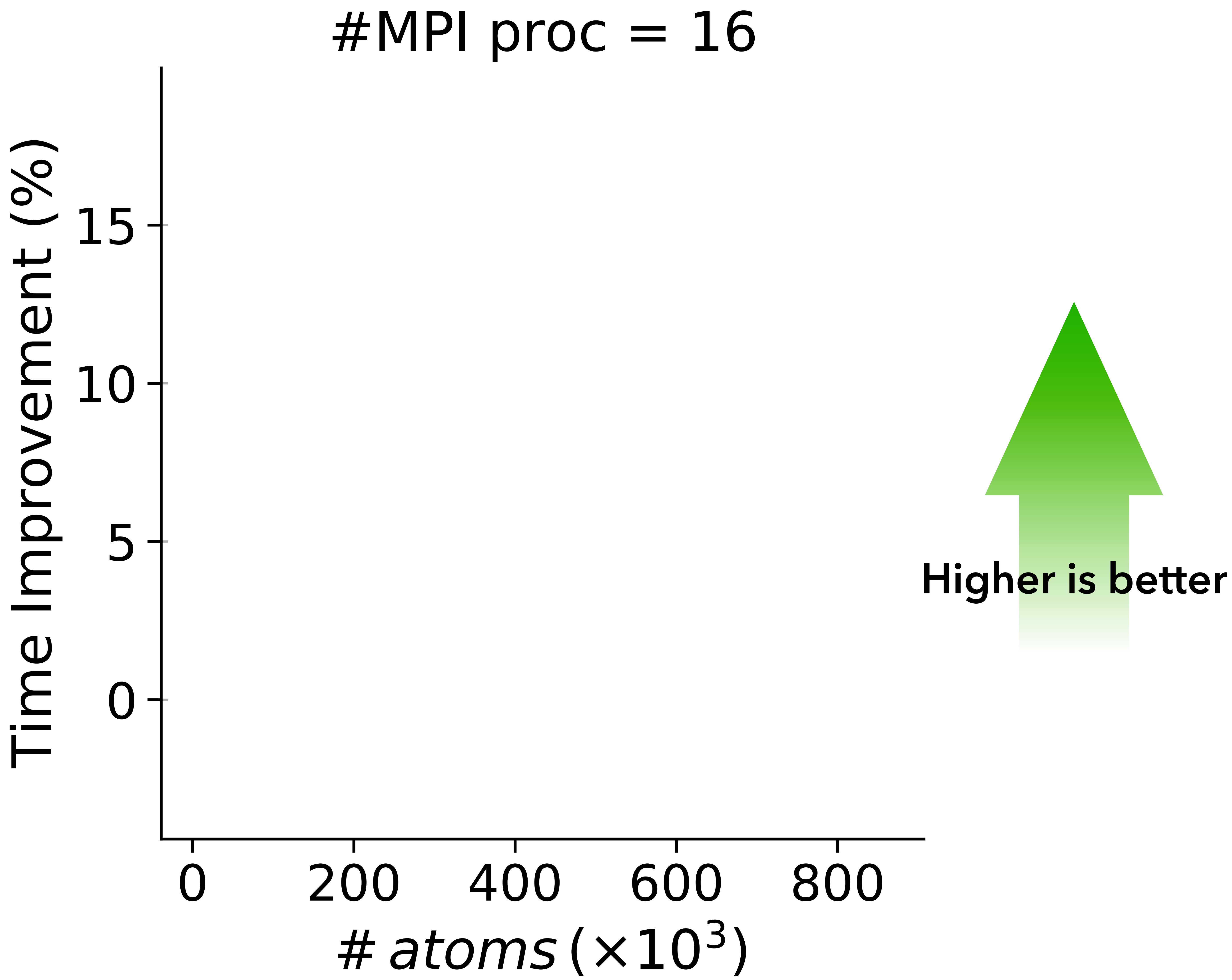
"THOR" CLUSTER, MAINTAINED BY THE HPC·AI ADVISORY COUNCIL [LINK]

- **System**: 16 nodes, Infiniband HDR (100 Gbps)
- **Hosts**: (2-socket) x (16-core Intel Broadwell E5-2697A, 2.6 GHz) + (256 GiB DDR4 RAM, 2400 MHz)
- **NICs per node**
  - 1 x NVIDIA **ConnectX-6 HDR100** (100 Gbps) InfiniBand/VPI adapters
  - 1 x NVIDIA **BlueField-2 SoC** – (8-core ARMv8 A72, 2.5 GHz) + (16 GiB DDR4 RAM) + (HDR100)



**S. Karamati (GT) et al. (Sandia, Queens U.; 2022) –** "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Restructured method …



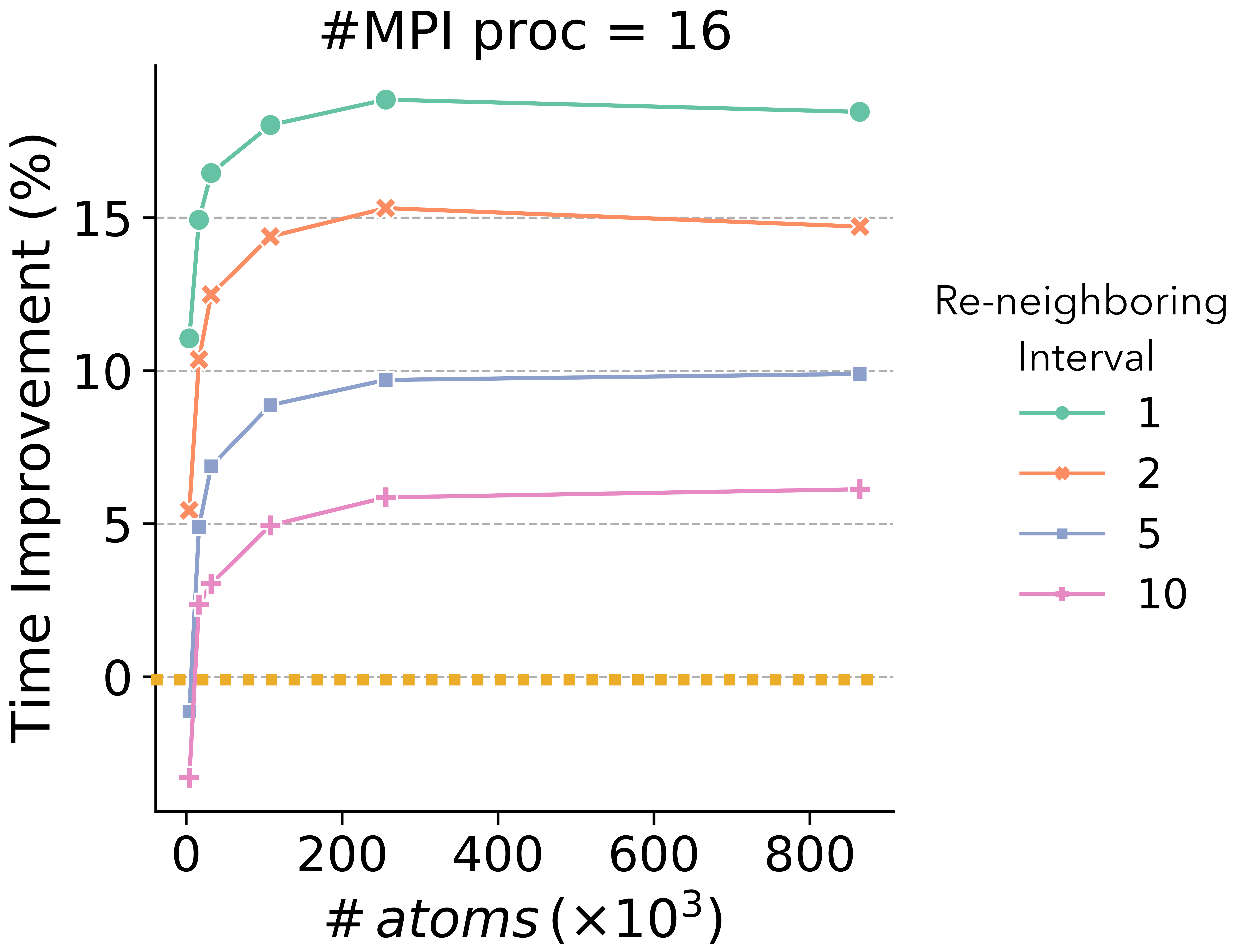



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Restructured method is faster

We observe small, but largely uniform, **speedups of up to 20%** compared to host-only execution with conventional NICs.

**This improvement compares favorably** with the **power increase** on each node due to BF2, which we estimate from sensors to be **as little as 6%**.



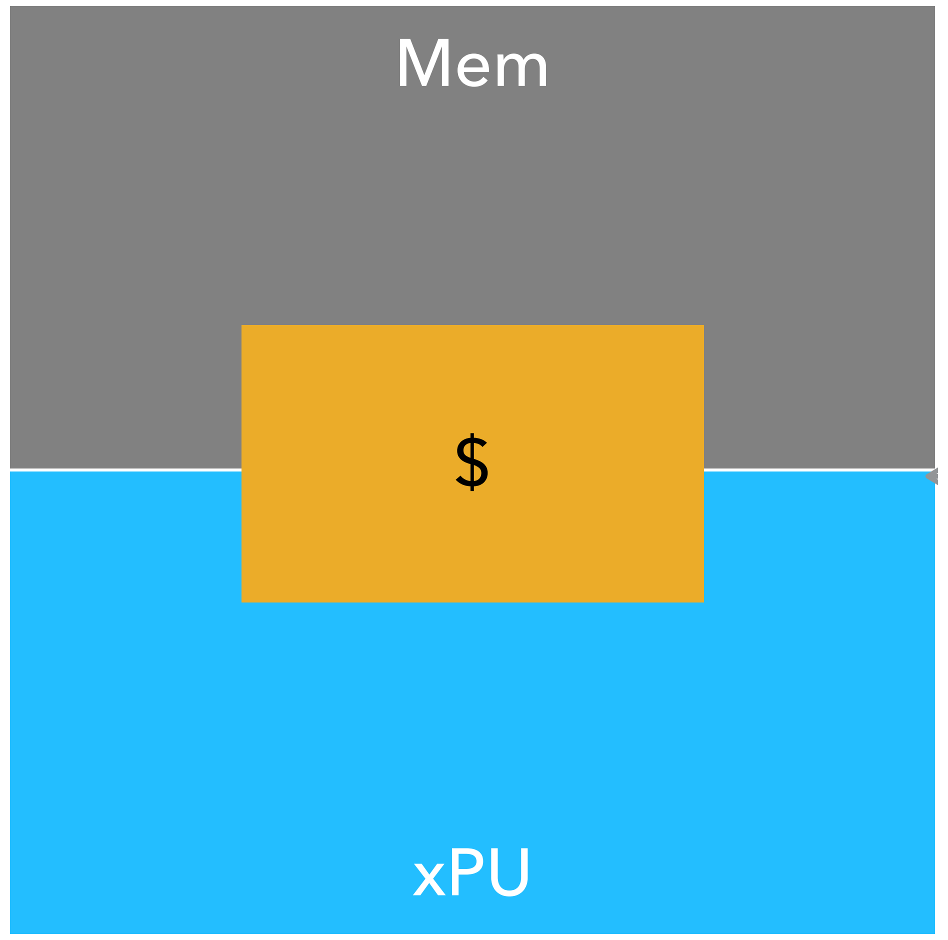



S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Hypothetical: Multi-SmartNIC

a.k.a., revisiting the "iron law"

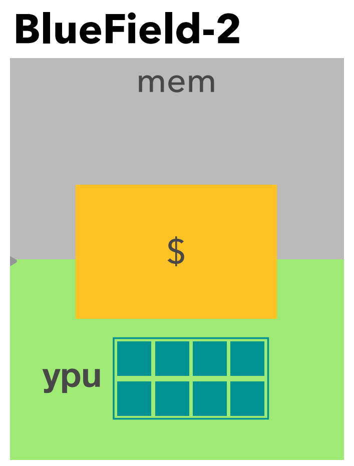
## One host xPU (16 cores)




**657 GF/s** (fp64)

**76.8 GB/s**

## BF-2 yPUs (no host)




**80 GF/s**

**25.6 GB/s**

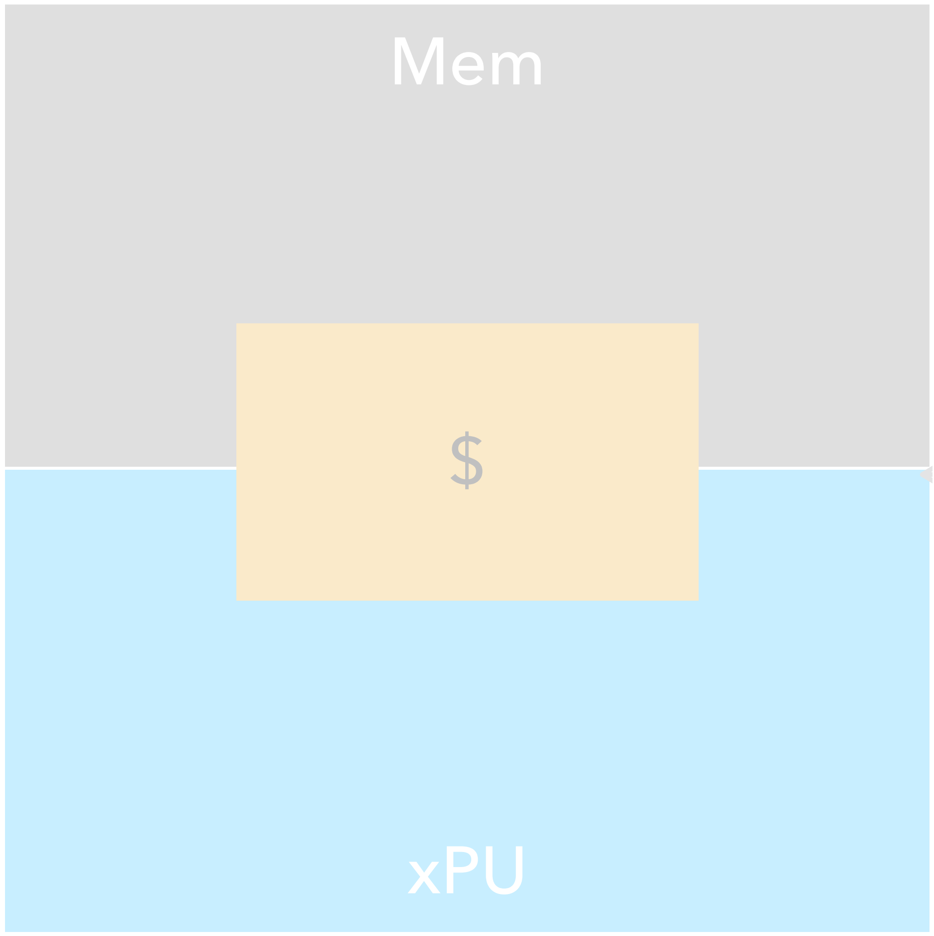


S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Hypothetical: Multi-SmartNIC

a.k.a., revisiting the "iron law"

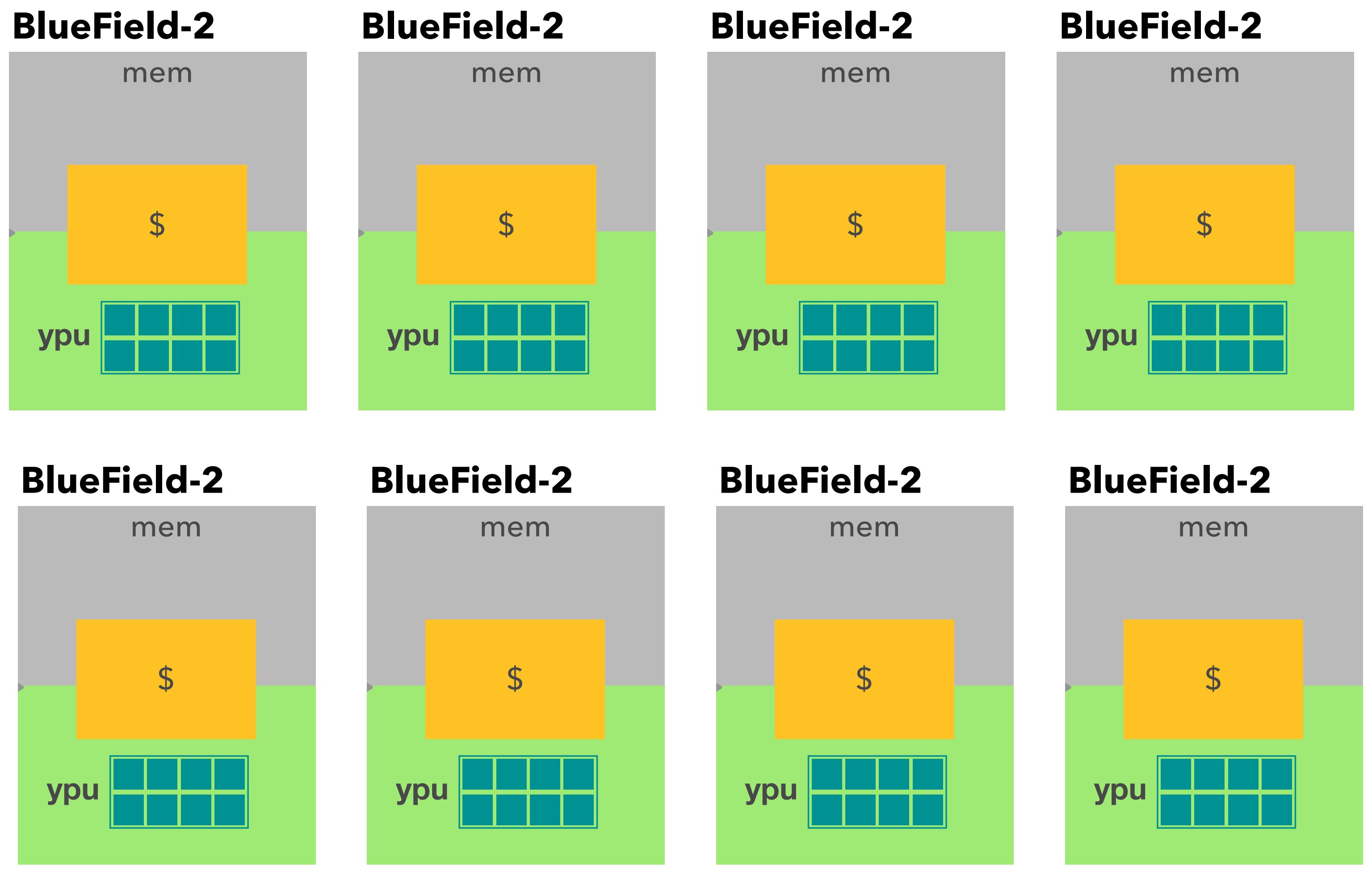
One host xPU (16 cores)




657 GF/s (fp64)

76.8 GB/s

8 x BF-2 yPUs (no host)




640 GF/s

204 GB/s

(aggregate)

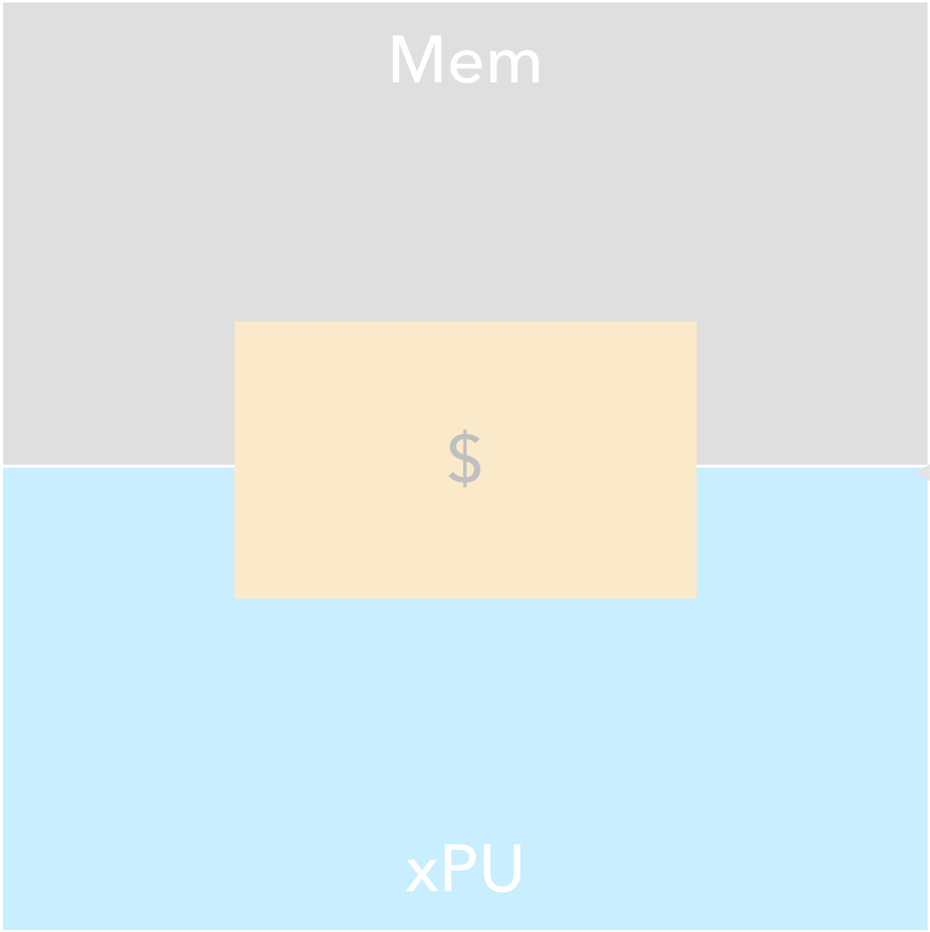


S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Hypothetical: Multi-SmartNIC

a.k.a., revisiting the "iron law"

## One host xPU (16 cores)




~ 8.5 F:B

## 8 x BF-2 yPUs (no host)


~ 3.1 F:B

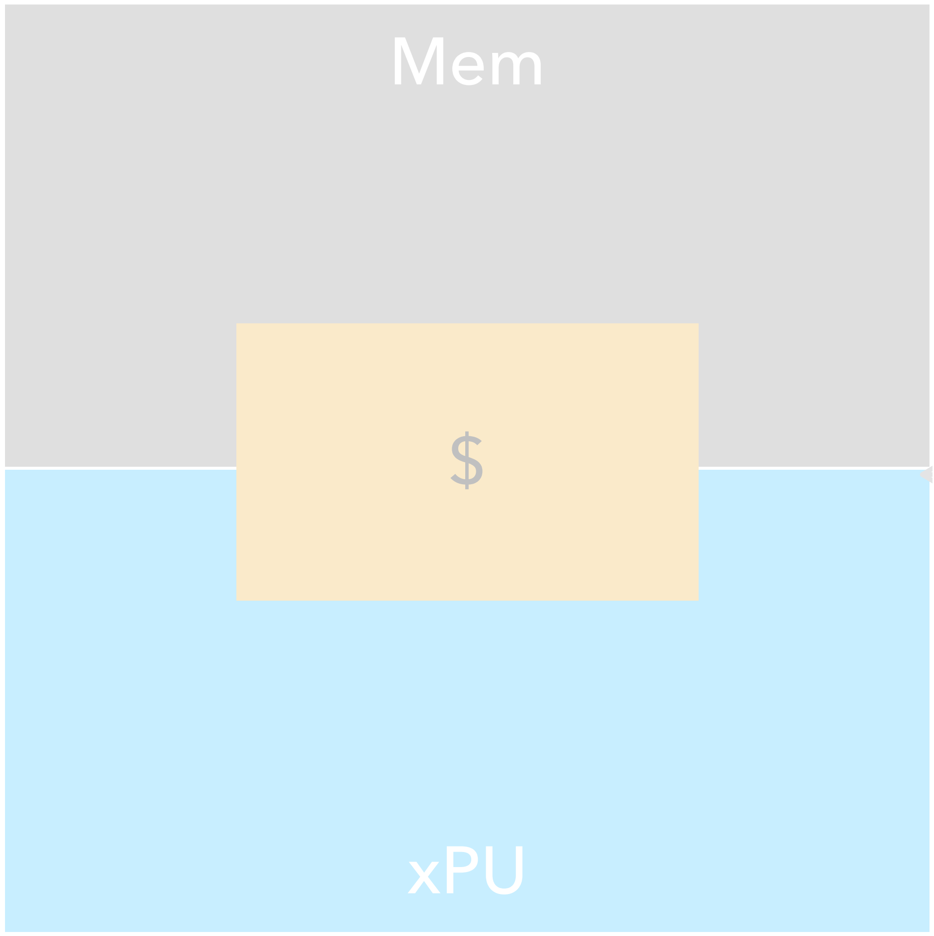


S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# Hypothetical: Multi-SmartNIC

a.k.a., revisiting the "iron law"

## One host xPU (16 cores)




**Time = "1"**

using all cores

## 8 x BF-2 yPUs (no host)


**Speedup ~ 1.7x**

**Real measurement on MiniMD!**

*(Similar for P3DFFT, SuperLU_DIST)*



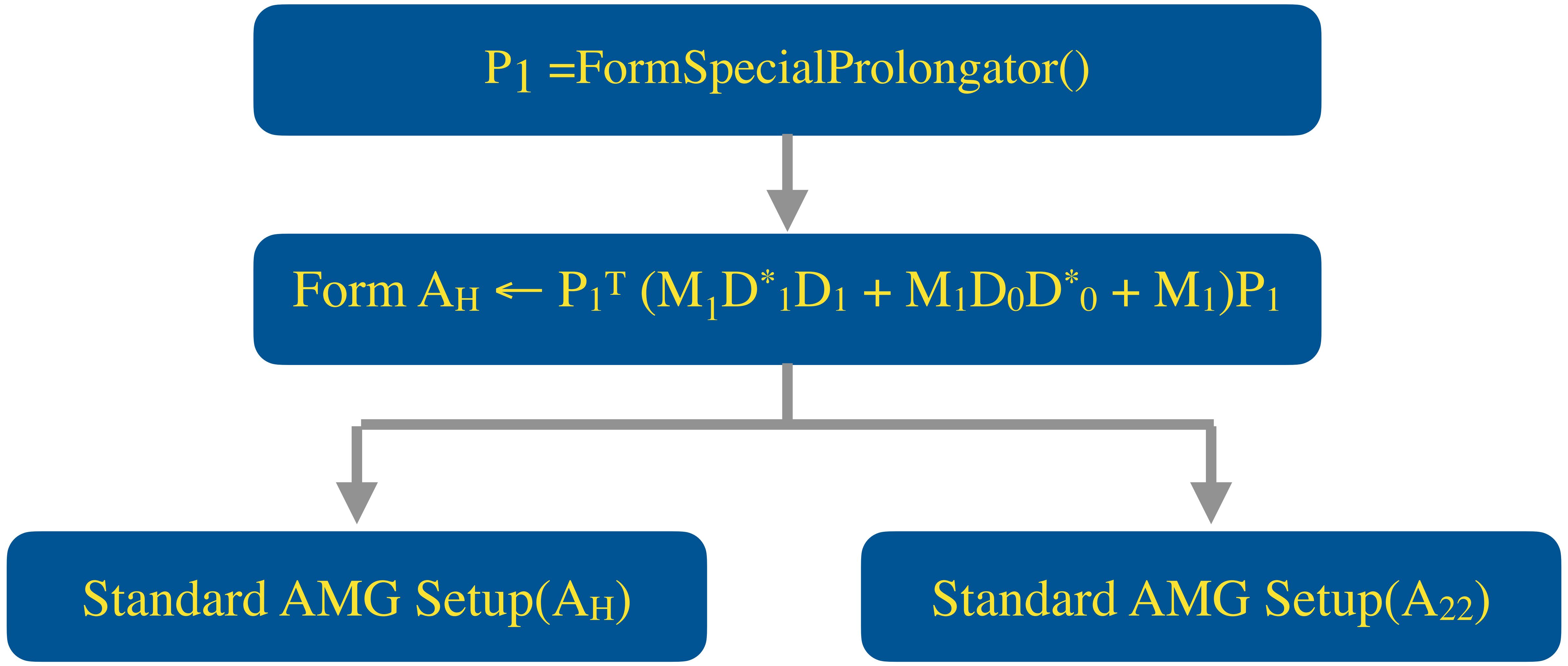
S. Karamati (GT) et al. (Sandia, Queens U.; 2022) – "Smarter NICs for faster molecular dynamics: a case study." doi:10.1109/IPDPS53621.2022.00063

# miniEM (MueLu) – Maxwell solver case study

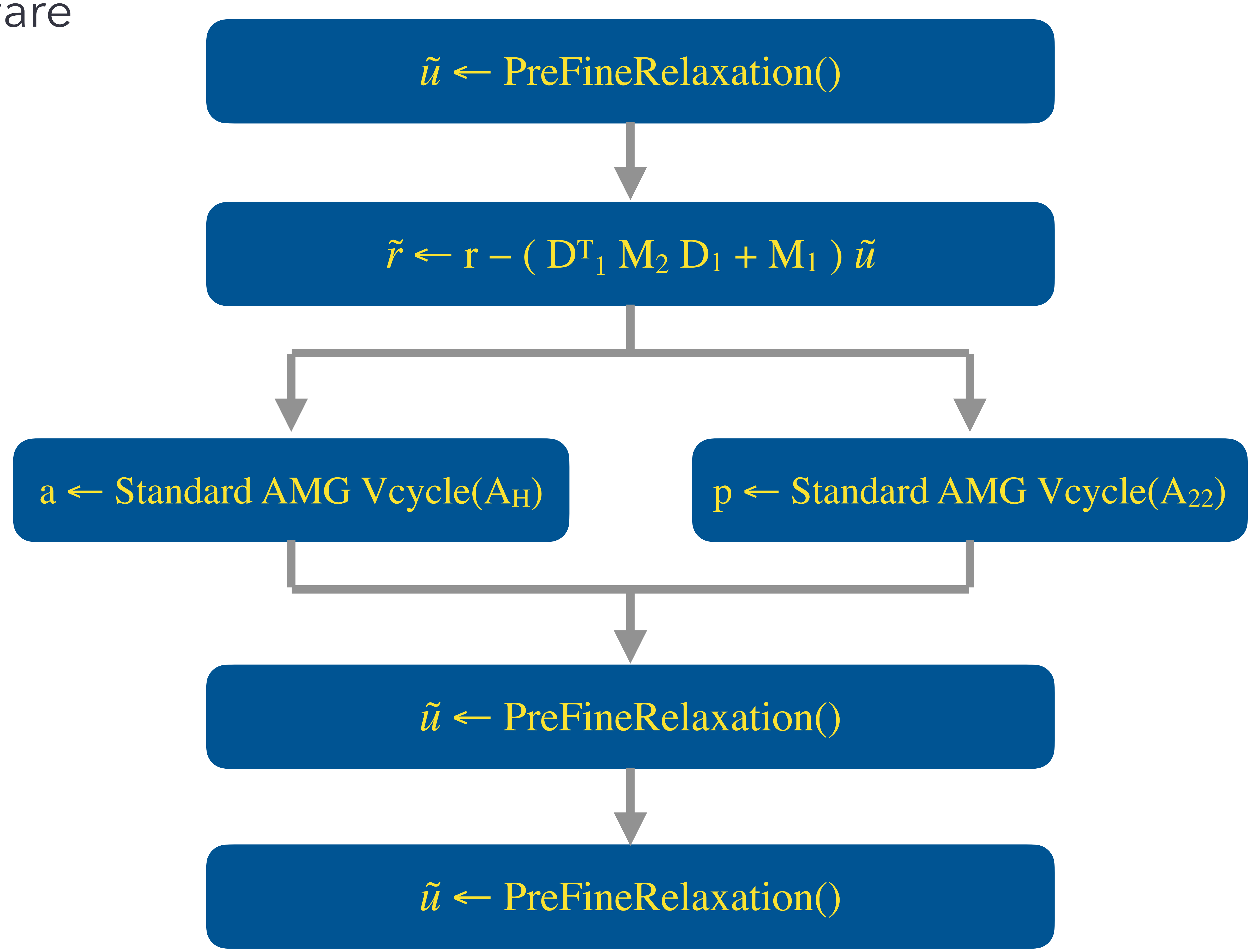
# Baseline Maxwell Solver

Maxwell multigrid solver from MueLu, an open-source software library within the Trilinos project

## Setup Phase




## Solve Phase

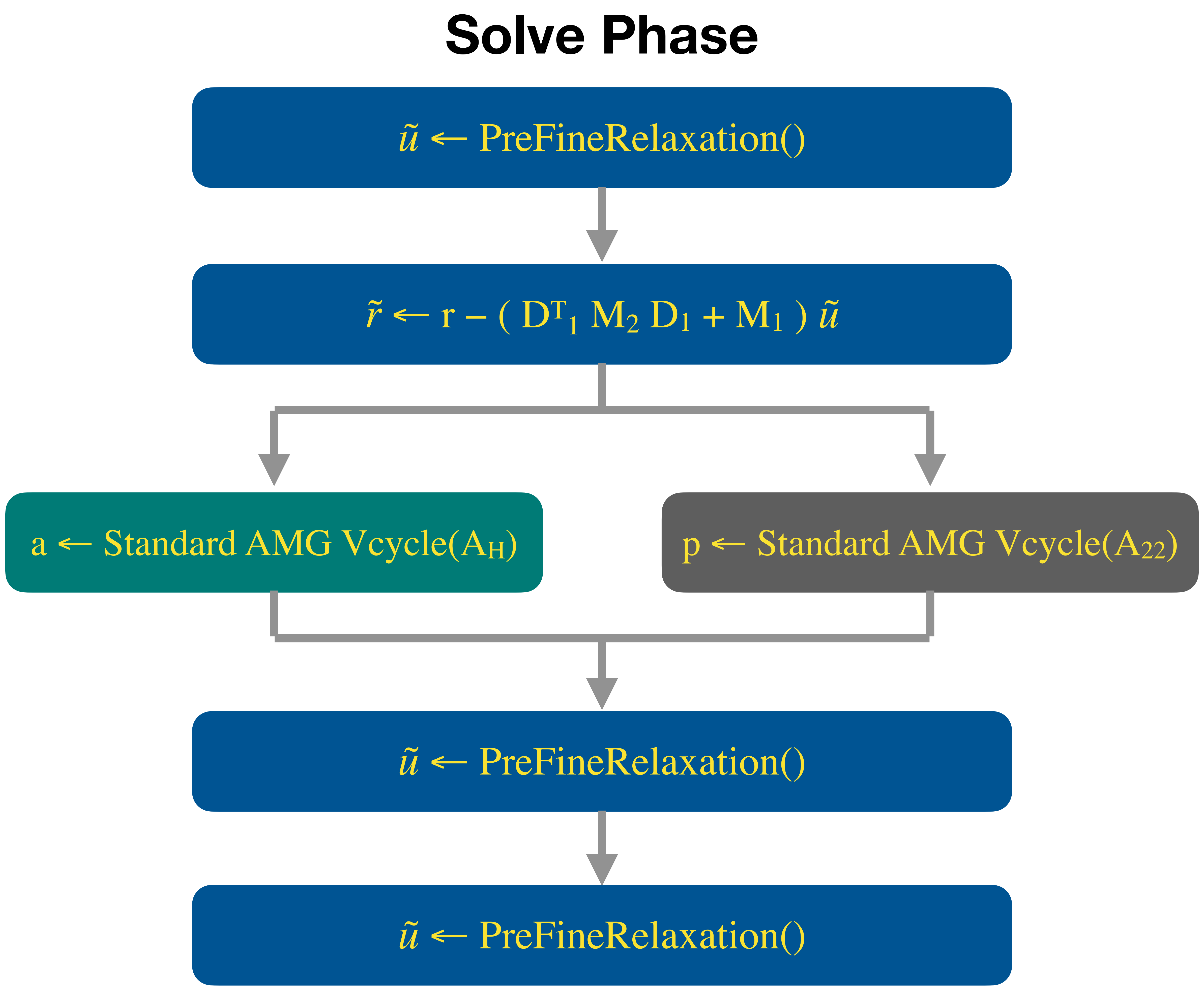




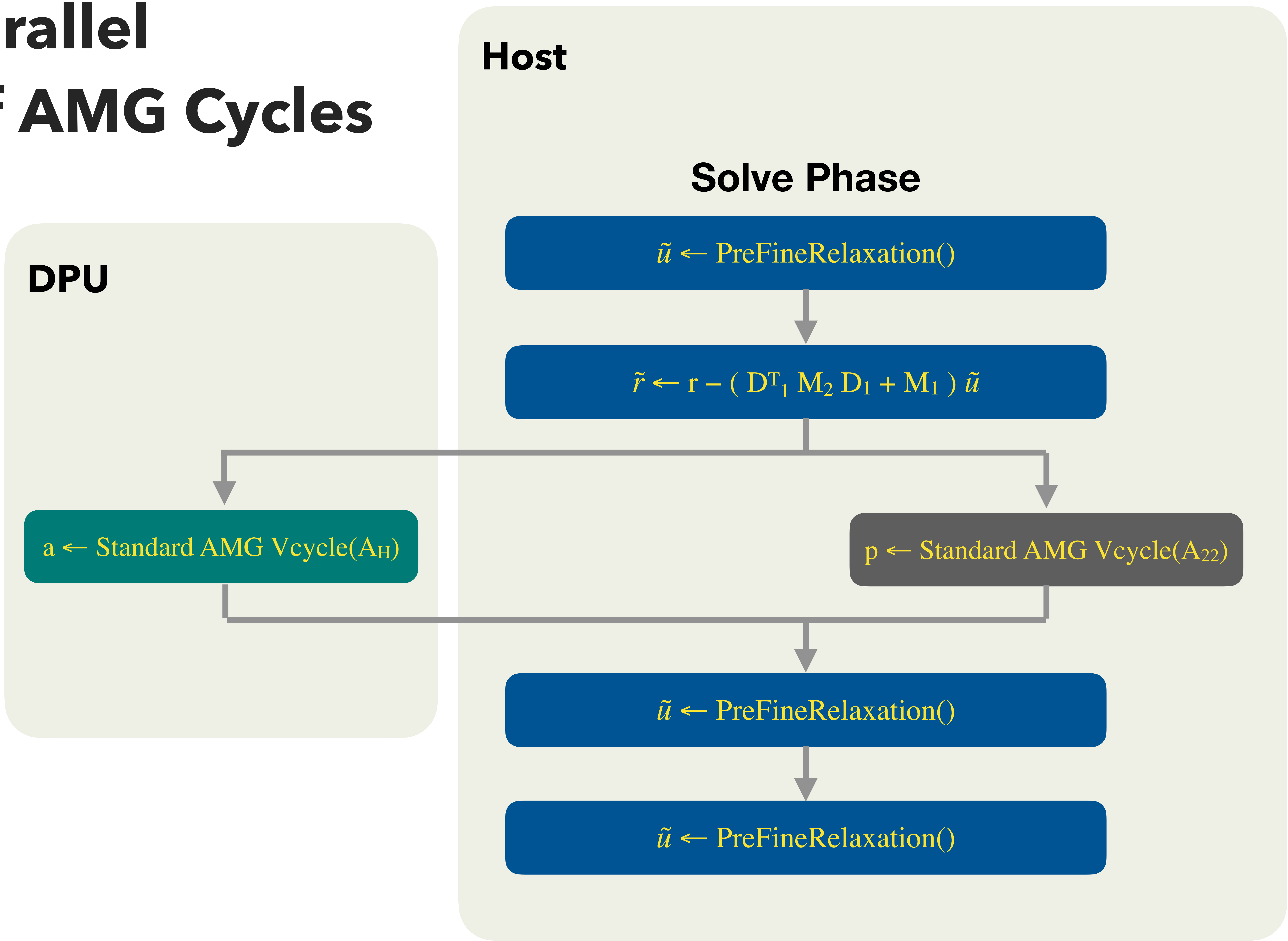
Bochev, Pavel B., et al. "An algebraic multigrid approach based on a compatible gauge reformulation of Maxwell's equations." *SIAM Journal on Scientific Computing* 31.1 (2008)

# Baseline Maxwell Solver

Standard AMG Vcycle(AH) and Standard AMG Vcycle(A22) operate **independently** and can be executed in **parallel** for optimized performance.



# Proposed Parallel Execution of AMG Cycles



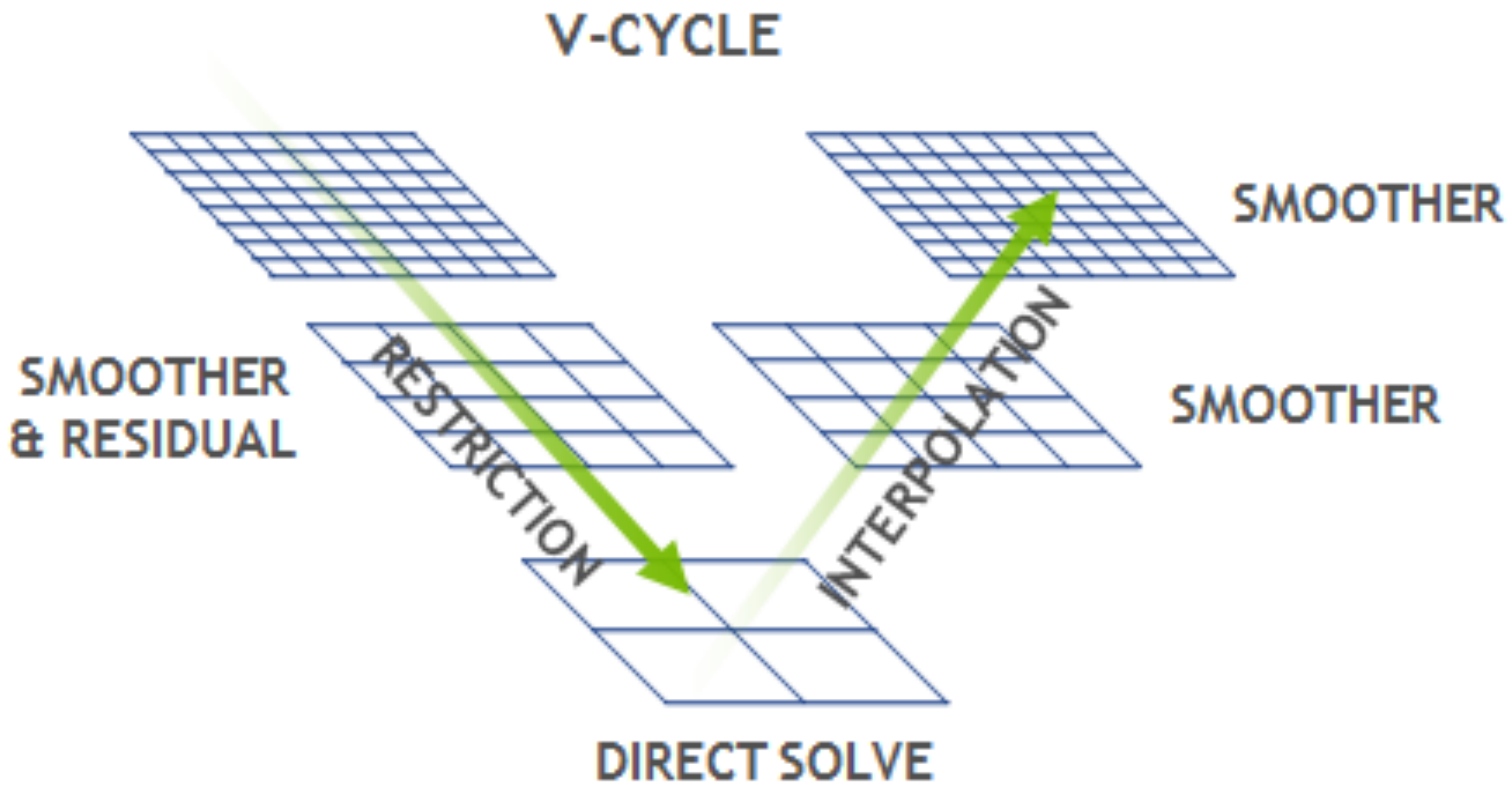
# AMG V-Cycles

**Multigrid Methods Involve:**

1. **Smoothing:**
   - Utilizes simple iterative methods like Gauss-Seidel.
   - Reduces oscillatory high-frequency error.

2. **Coarse-grid Correction:**
   - Transfers information to a coarser grid through restriction.
   - Solves the coarse-grid system of equations.
   - Eliminates low-frequency error.

3. **Interpolation:**
   - Transfers the solution back to the fine grid.




Source: Multi grid V-cycle

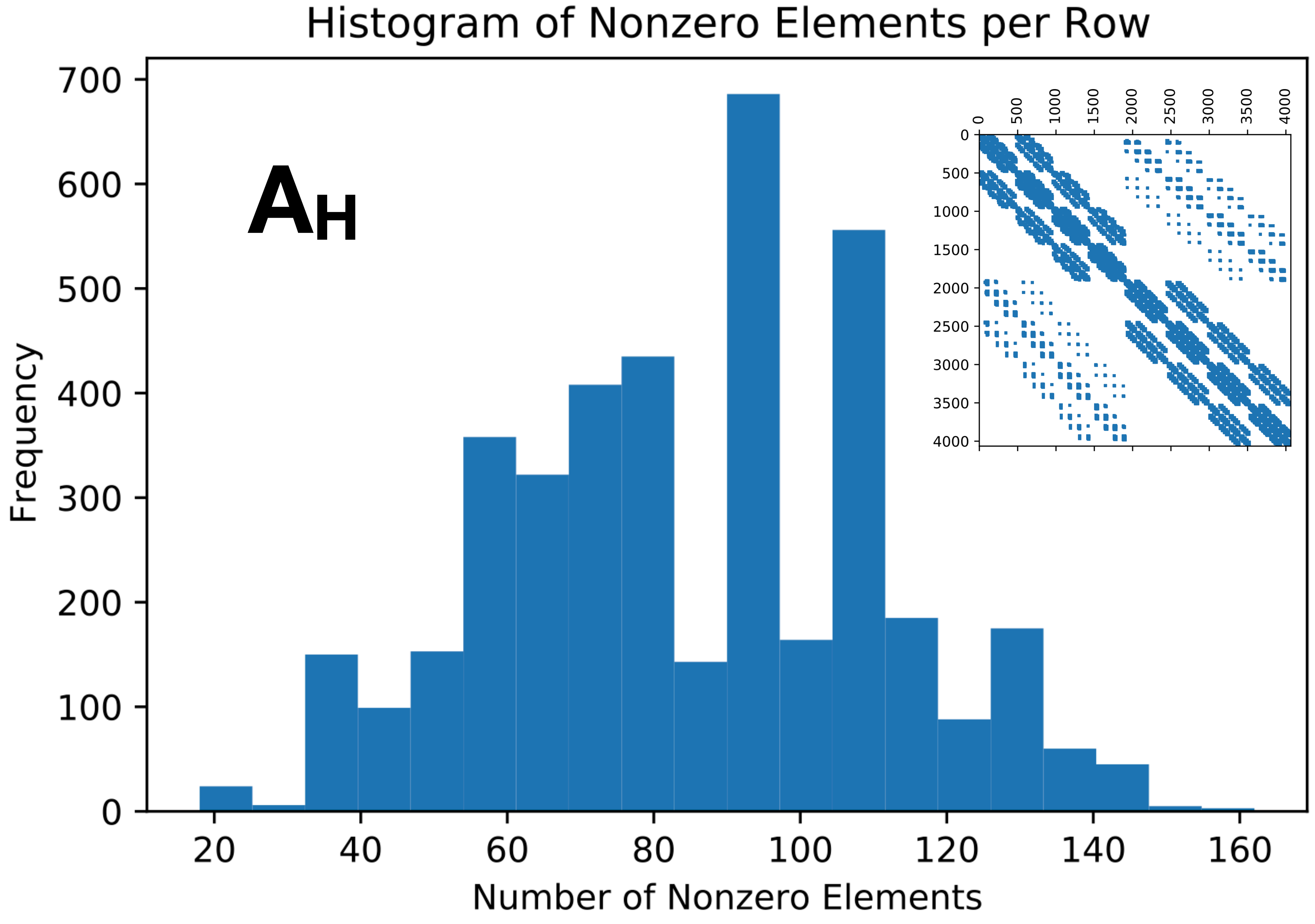
Computational costs are primarily governed by **sparse** matrix operations

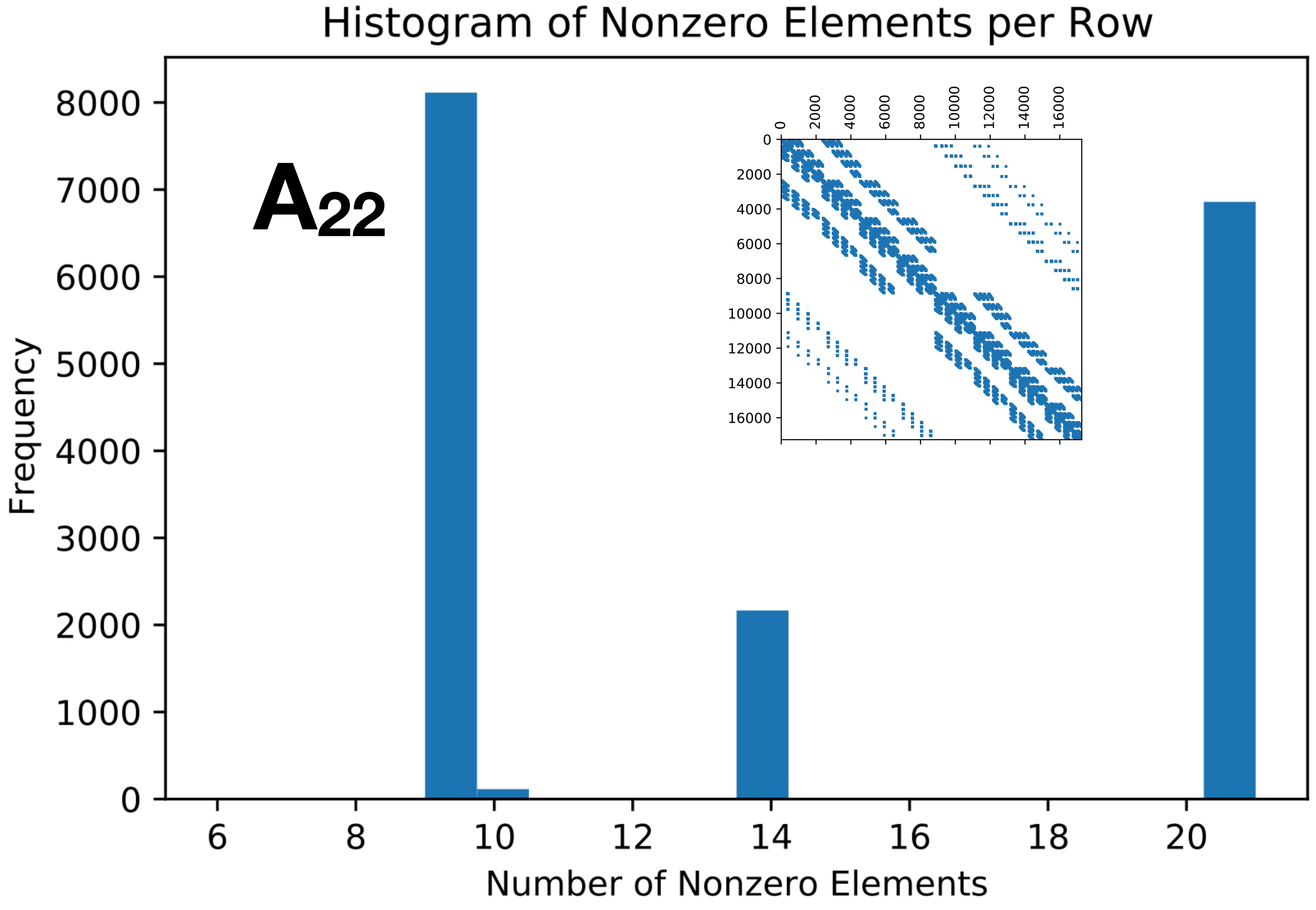
# Experimental Testbed

| | Host | Bluefield-3 |
|---|---|---|
| **Core** | Intel Broadwell E5-2697A | Arm Cortex A-78 |
| **# Sockets** | 2 | 1 |
| **Cores/Socket** | 16 | 16 |
| **Clock (GHz)** | 2.6 | 2.25 |
| **Private L1 DCache** (per core) | 32 KB | 64KB |
| **Private L2 Dcache** (per core) | 256 KB | 512 KB |
| **Shared L3 Cache** (per node) | 80 MB | 16 MB |
| **DRAM** | DDR4 (4800 MT/s) | DDR5 (5600 MT/s) |
| **Peak flop/s per socket (FP64)** | 656.6 Gflop/s | 288 Gflop/s |
| **Peak GB/s per socket** | 76.8 GB/s | 69.21 GB/s |

# Matrix Representation and Non-Zero Element Distribution

| Matrix | rows | nnz |
|---|---|---|
| $A_H$ | 4,065 | 346,665 |
| $A_{22}$ | 17,261 | 248,581 |

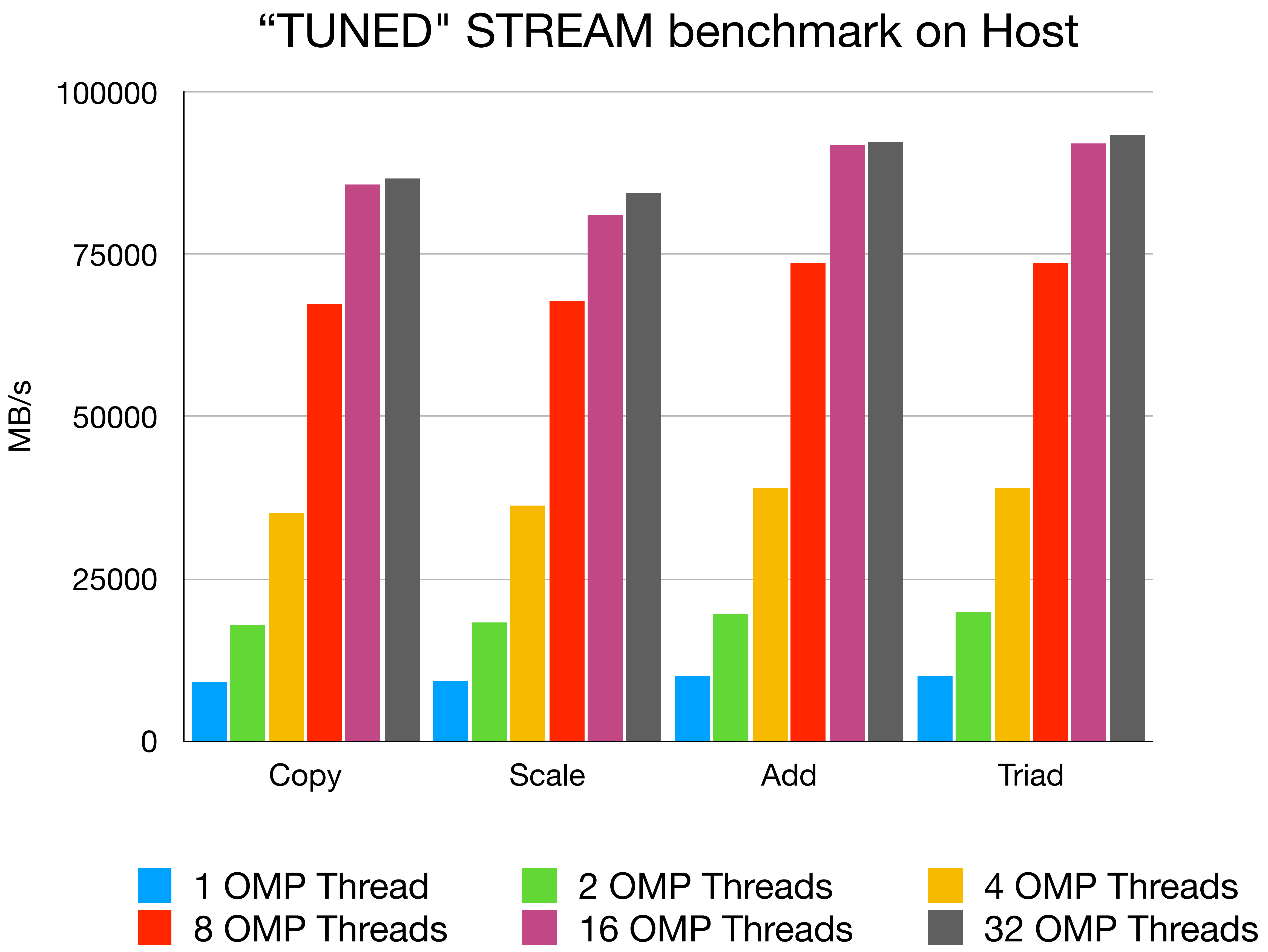


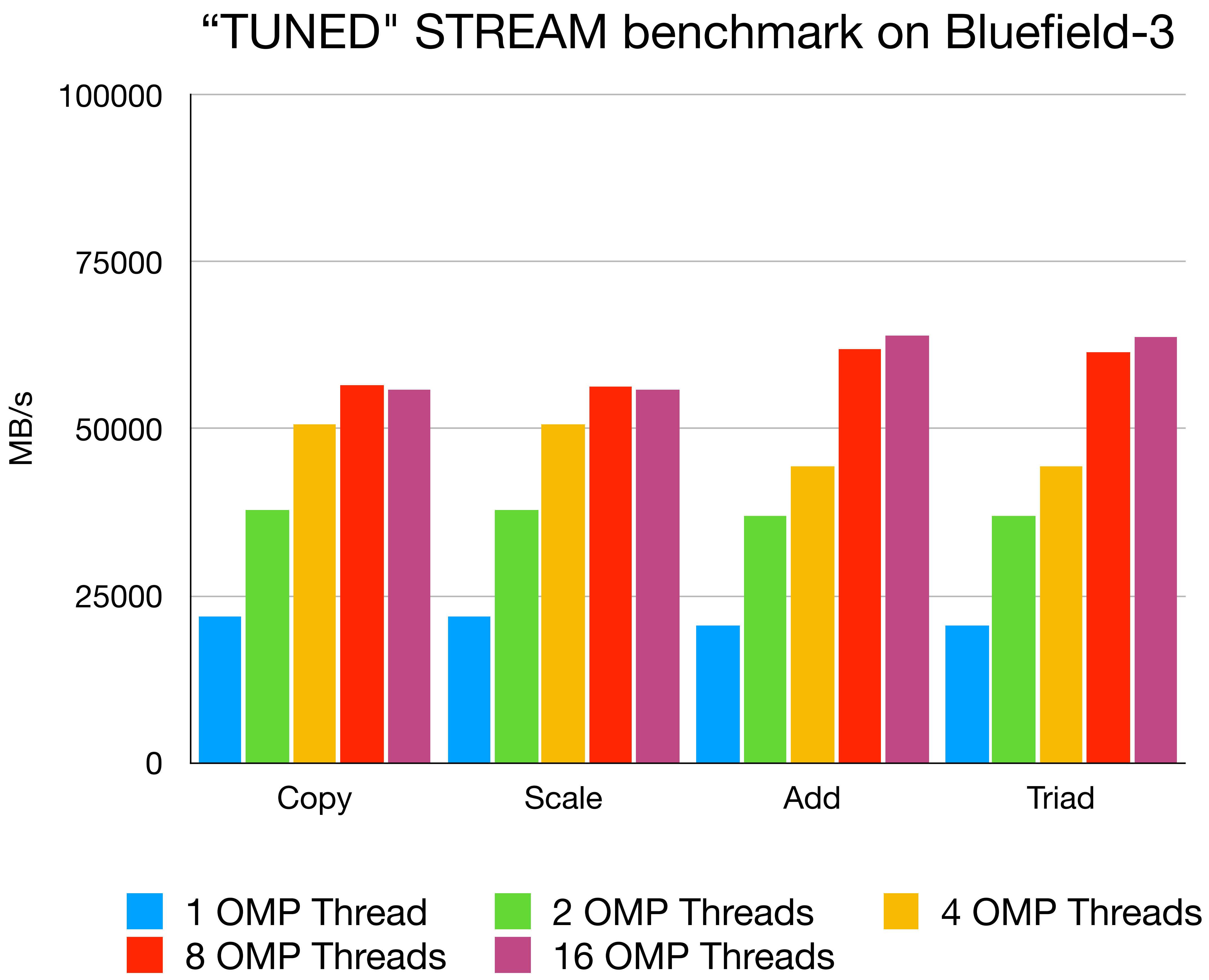




Observation: **$A_{22}$ is smaller but $A_H$ stresses cache more, so it's not clear *a priori* which to offload**

# Comparative Benchmarking:
# “TUNED" STREAM benchmark on Host vs. Bluefield

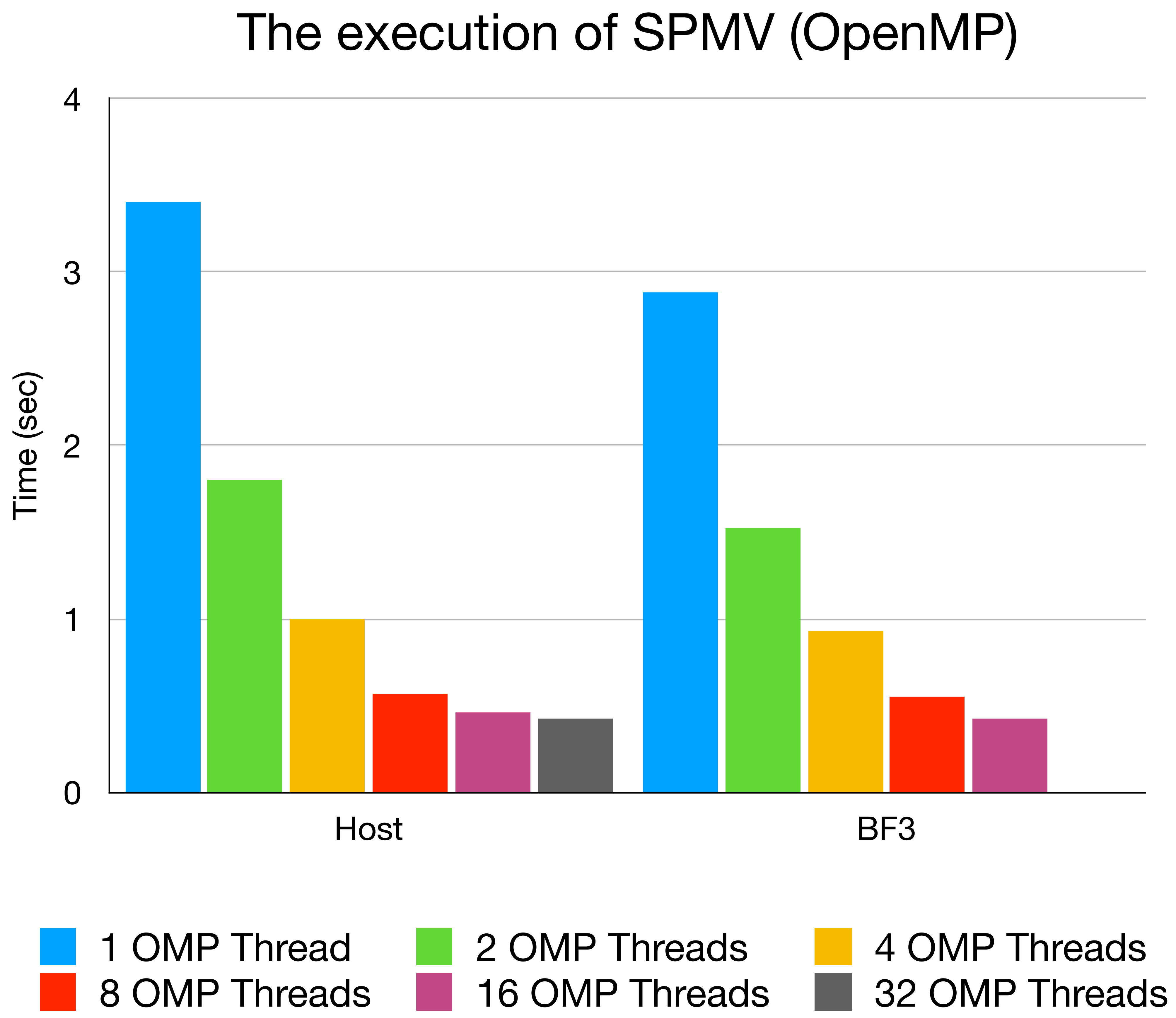
Evaluating with Array of 20,000,000 elements, running each test 100 times.

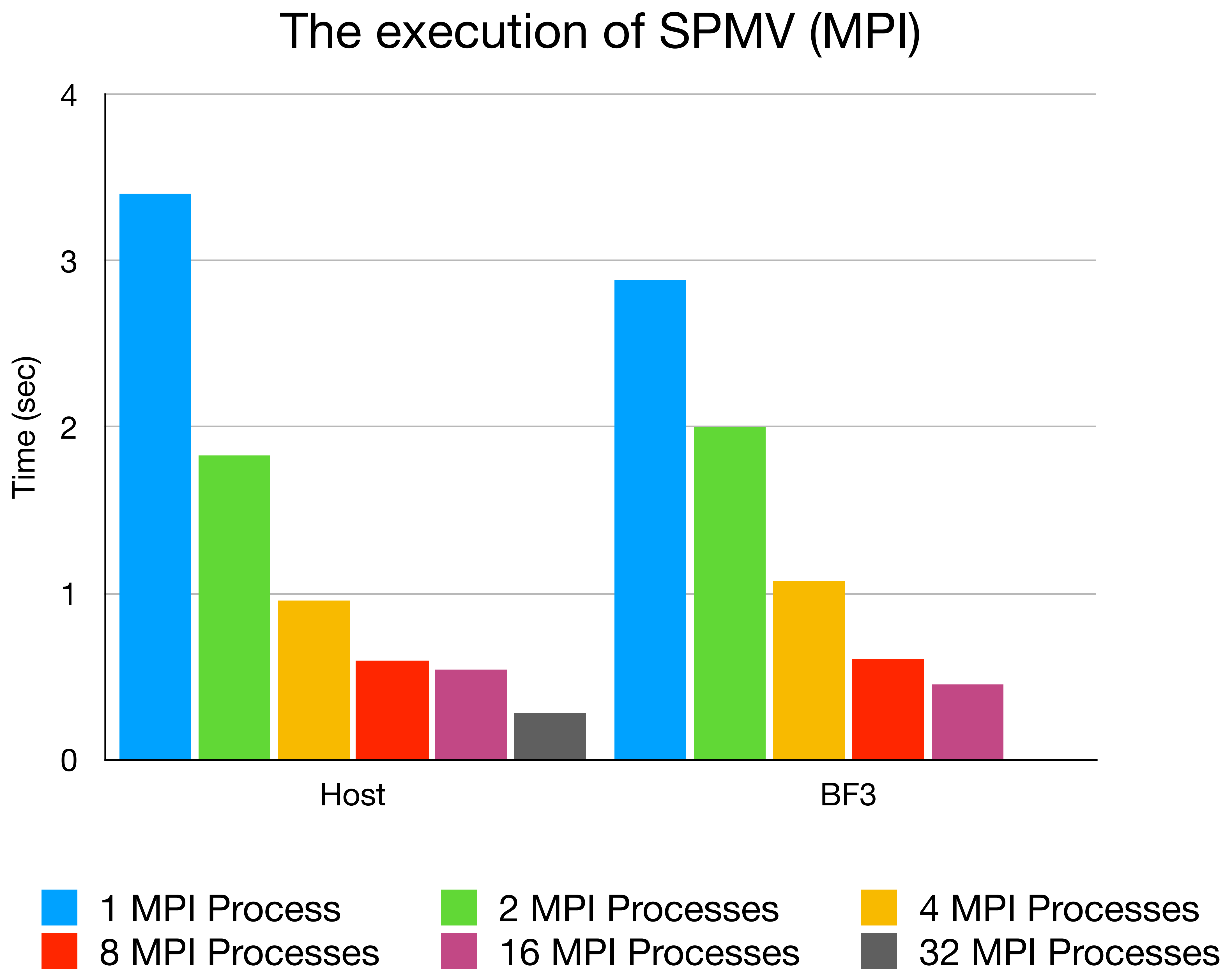


# Comparative Benchmarking:
# Sparse Matrix Vector Performance on Host vs. Bluefield

Evaluating with Matrix of 163,617 Rows and 13,662,045 nnz Elements



# Preliminary results: End-to-End Solve-Phase Times for miniEM: Host vs. Host + BlueField3

Times shown are end-to-end solver times.

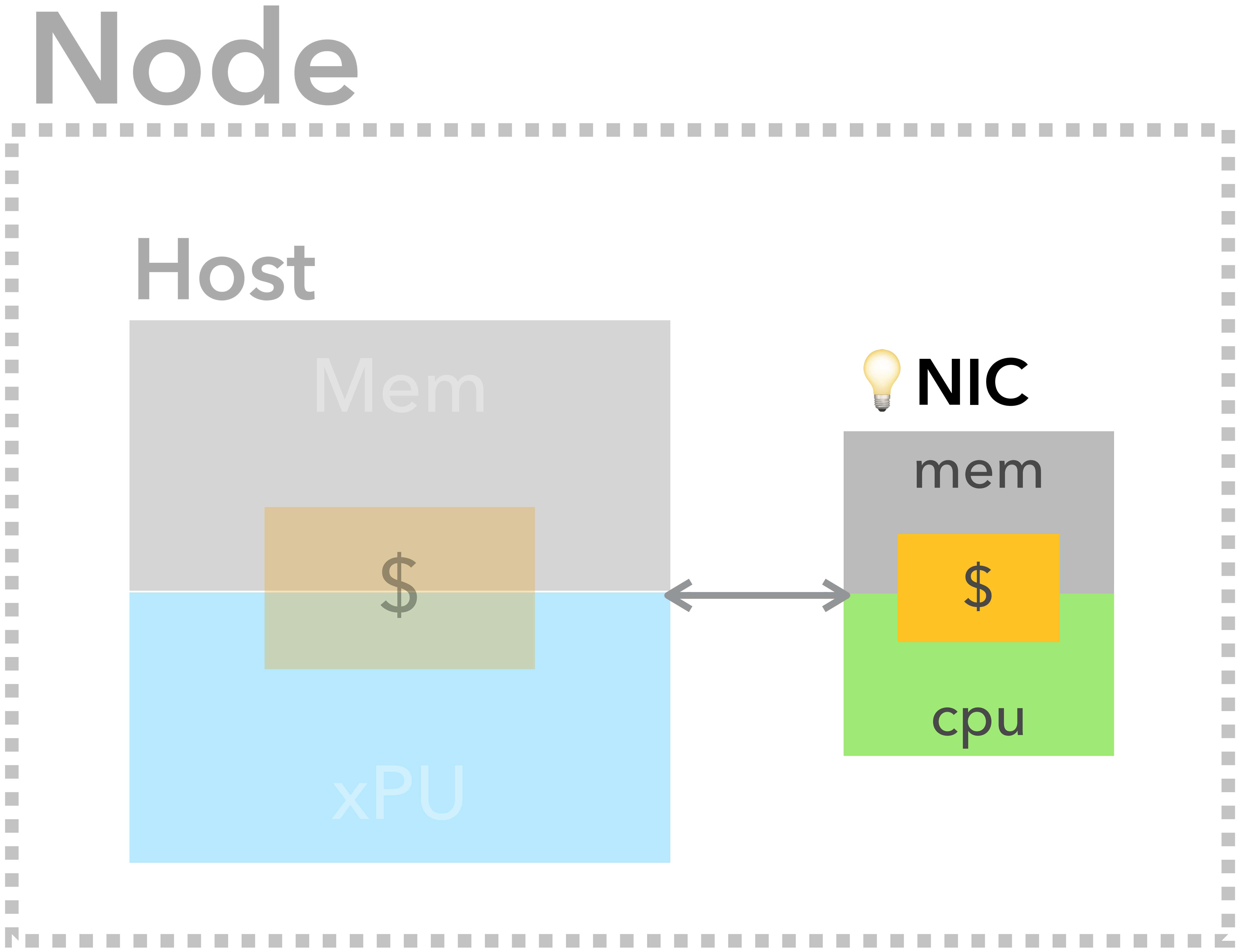
| Problem Size | #Host Cores / #BF3 Cores | Host-only Time (seconds) | H+BF Time (seconds) | Relative Scaling |
|---|---|---|---|---|
| **120** | 256 / 128 | 85.358 | 76.1103 | 1.12 |
| **120** | 128 / 64 | 137.062 | 129.928 | 1.05 |
| **120** | 64 / 32 | 220.767 | 213.919 | 1.03 |
| **80** | 256 / 128 | 59.862 | 50.9477 | 1.17 |
| **80** | 128 / 64 | 72.0195 | 63.5482 | 1.13 |
| **80** | 64 / 32 | 95.3597 | 87.6009 | 1.09 |
| **60** | 256 / 128 | 38.5208 | 31.2498 | 1.23 |
| **60** | 128 / 64 | 42.619 | 37.5846 | 1.13 |
| **60** | 64 /32 | 51.4106 | 46.4185 | 1.11 |

# Summary

**Communication is fundamental and inevitable**, so anything that addresses it should be pursued vigorously.

**Restructuring algorithms**, especially increasing asynchrony, can exploit smartNICs in HPC. We are pursuing a variety of candidates, including distributed time-tiled stencils, AMR, novel collectives, among others.

Many open questions remain, regarding other techniques, programming, runtimes, and performance modeling.



# Asynchronous I/O in  BlueField Target using OpenMP

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
#pragma omp target nowait
{
    for (i = 0; i < 5; ++i)
        printf("hola - %05d\n", i);
} // end omp target
#pragma omp task
{
    for (j = 0; j < 5; ++j)
        printf("adios - %06d\n", j);
} // end omp task
} // end omp single
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# Asynchronous I/O in  BlueField Target using OpenMP

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
#pragma omp target nowait
{
    for (i = 0; i < 5; ++i)
        printf("hola - %05d\n", i);
} // end omp target
#pragma omp task
{
    for (j = 0; j < 5; ++j)
        printf("adios - %06d\n", j);
} // end omp task
} // end omp single
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# Asynchronous I/O in BlueField Target using OpenMP

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
#pragma omp target nowait
{
    for (i = 0; i < 5; ++i)
        printf("hola - %05d\n", i);
} // end omp target
#pragma omp task
{
    for (j = 0; j < 5; ++j)
        printf("adio  - %06d\n", j);
} // end omp task
} // end omp single
```

```
uthmanhere@jupiter030:~/test_openmp/build$ ./async
adio - 000000
adio - 000001
adio - 000002
adio - 000003
adio - 000004
uthmanhere@jupiter030:~/test_openmp/build$
```

```
uthmanhere@jupiterbf030:~$ hola - 00000
hola - 00001
hola - 00002
hola - 00003
hola - 00004
```

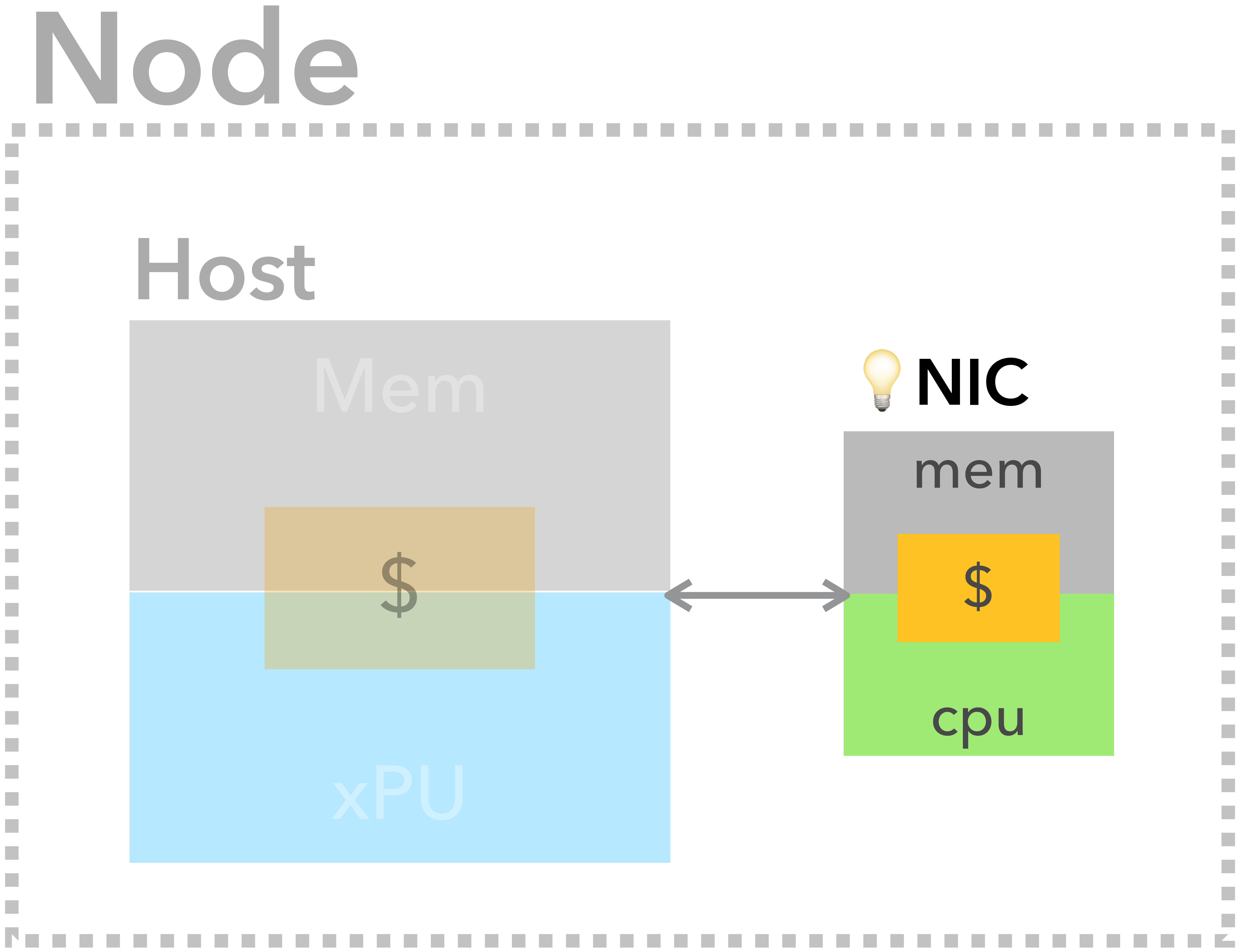
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# Summary

**Communication is fundamental and inevitable**, so anything that addresses it should be pursued vigorously.

**Restructuring algorithms**, especially increasing asynchrony, can exploit smartNICs in HPC. We are pursuing a variety of candidates, including distributed time-tiled stencils, AMR, novel collectives, among others.

Many open questions remain, regarding other techniques, programming, runtimes, and performance modeling.



# Director’s cut

# Four "generations" of computing

**Gregory Abowd (2016)**. "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

| Generation | Time frame | Human–computer ratio | Canonical device | Application | |
|---|---|---|---|---|---|
| | | | | Initial | Follow-on |
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# Four "generations" of computing

**Gregory Abowd (2016)**. "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

| Generation | Time frame | Human–computer ratio | Canonical device | Application: Initial | Application: Follow-on |
|---|---|---|---|---|---|
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# Four "generations" of computing

**Gregory Abowd (2016)**. "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

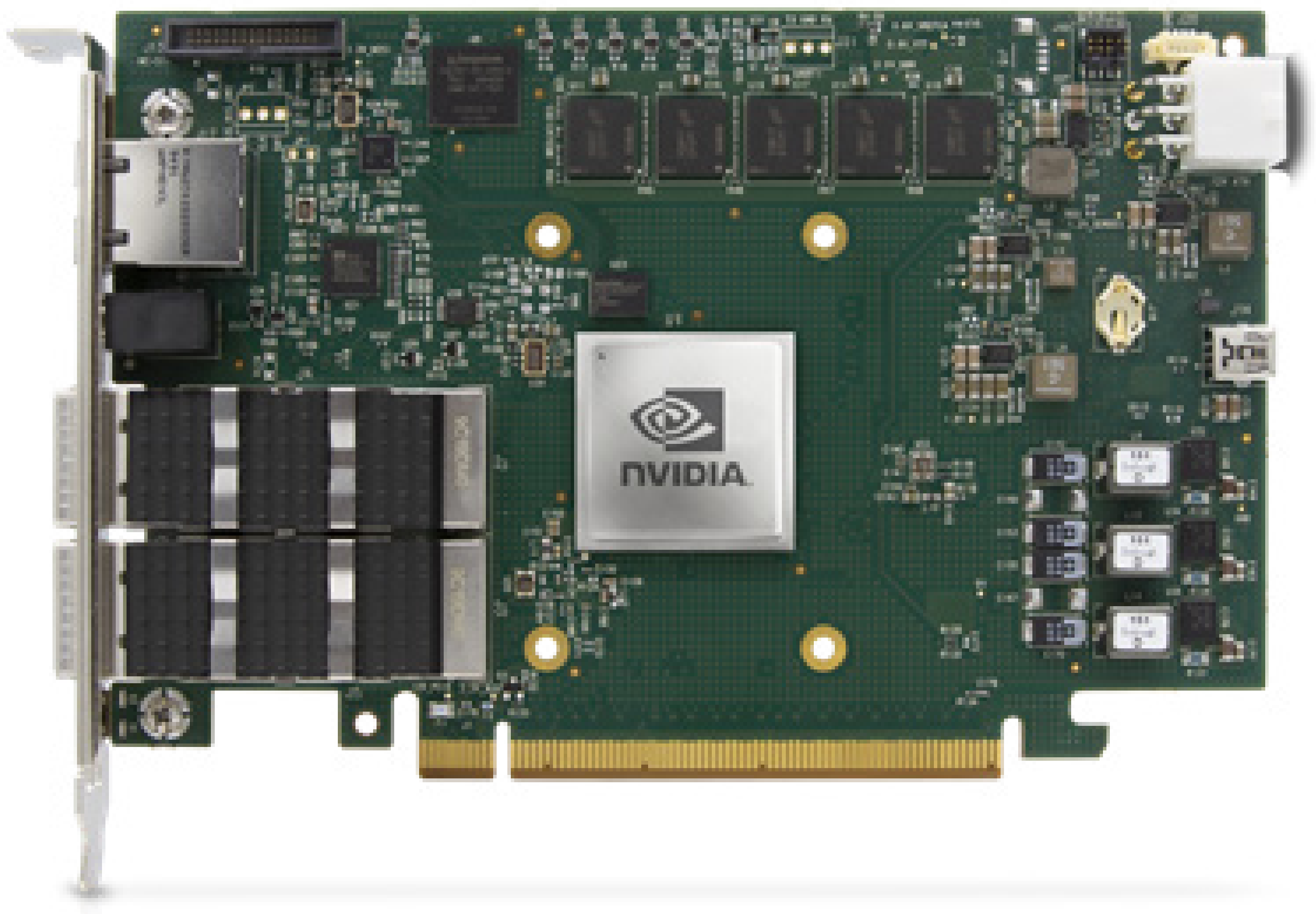
| Generation | Time frame | Human–computer ratio | Canonical device | Application | |
|---|---|---|---|---|---|
| | | | | Initial | Follow-on |
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# Our foci (gaps)

SEE PART 3 OF THIS TALK

- **Platform**: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- **Applications**: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- **Programming model**: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)




SEE PART 3 OF THIS TALK

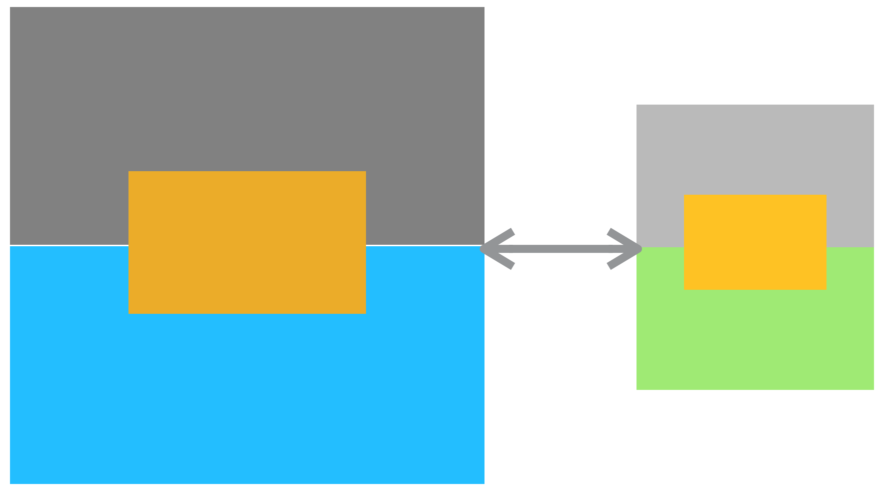
- **Platform**: **DPUs** like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- Usage model: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)

- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

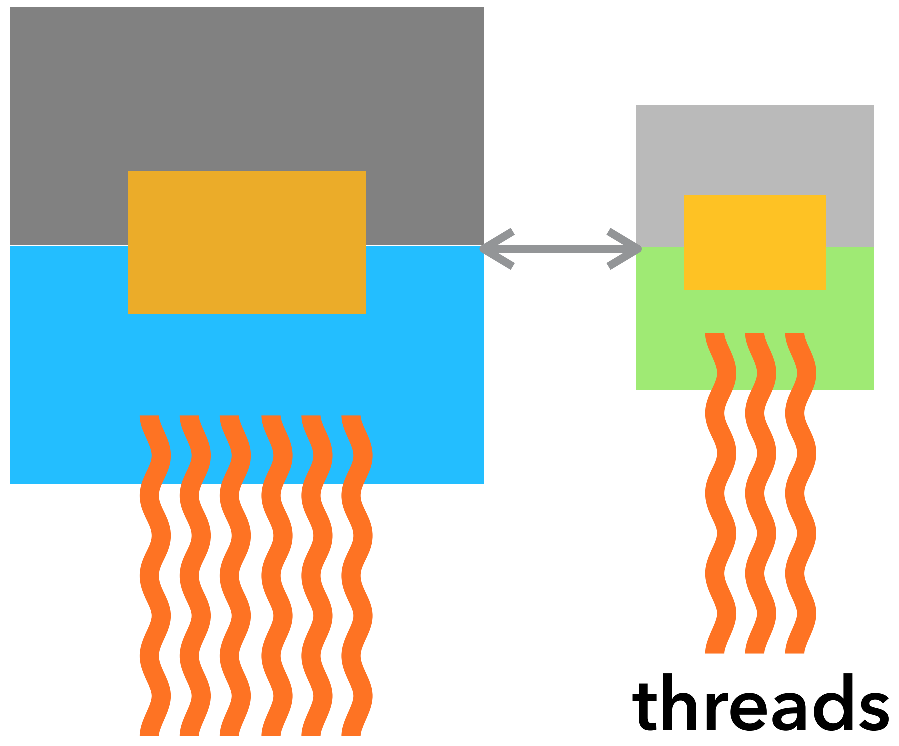
# Our foci (gaps)

**Off-path** (async & indep threads)



- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

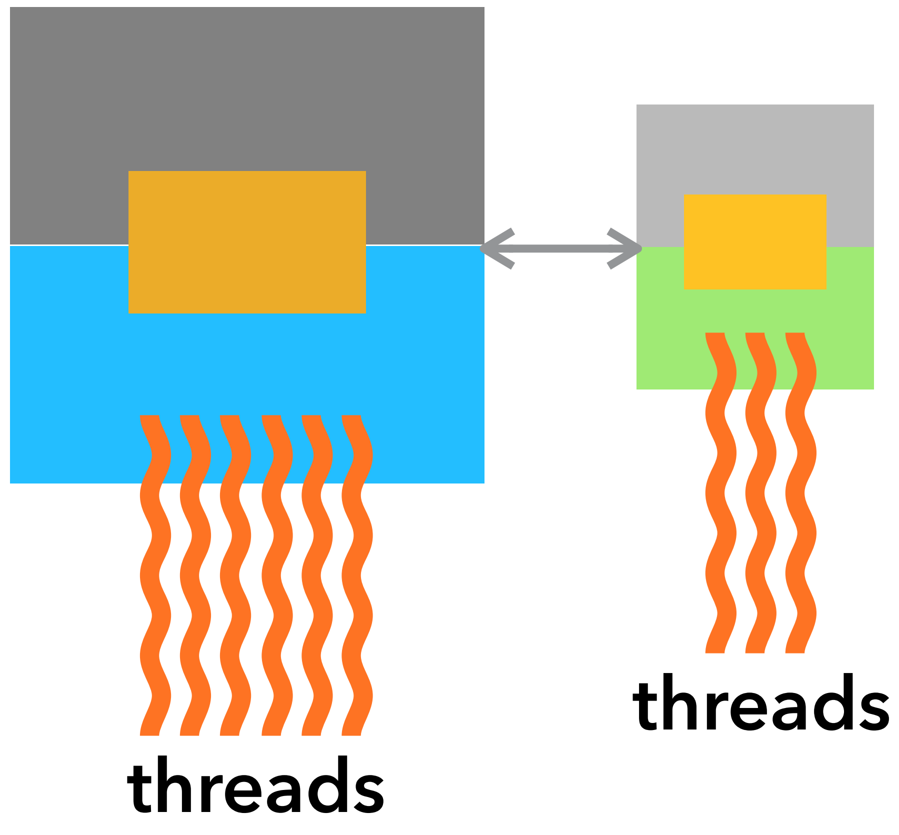
# Our foci (gaps)



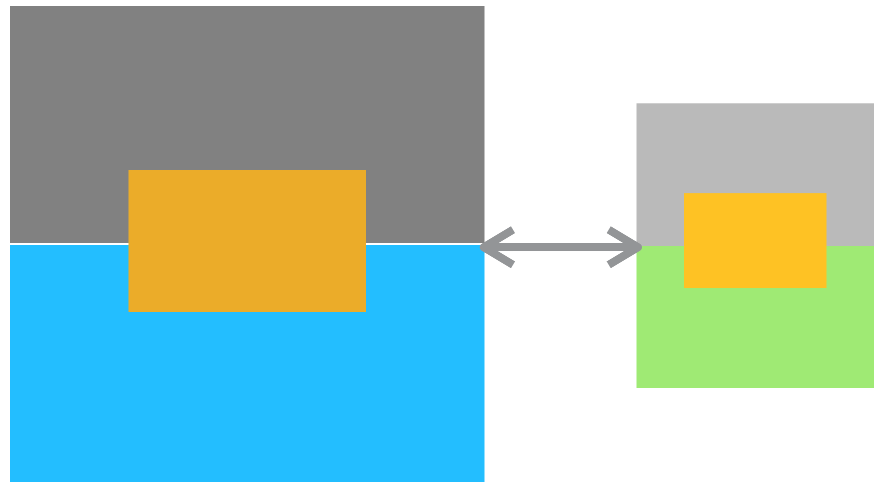

- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)

**Off-path** (async & indep threads)

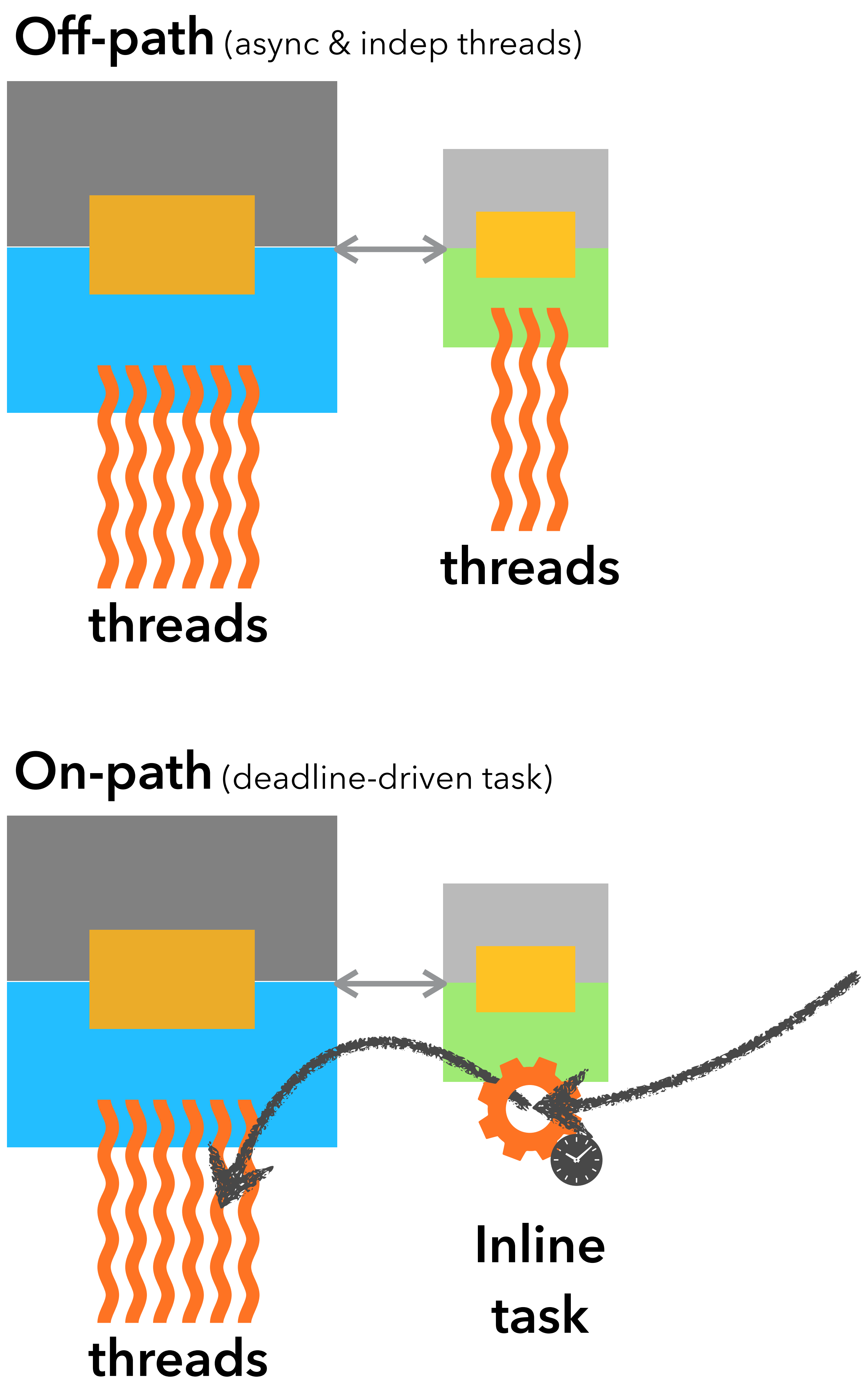



**On-path** (deadline-driven task)



- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)




- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- **Usage model**: **Off-path** computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)

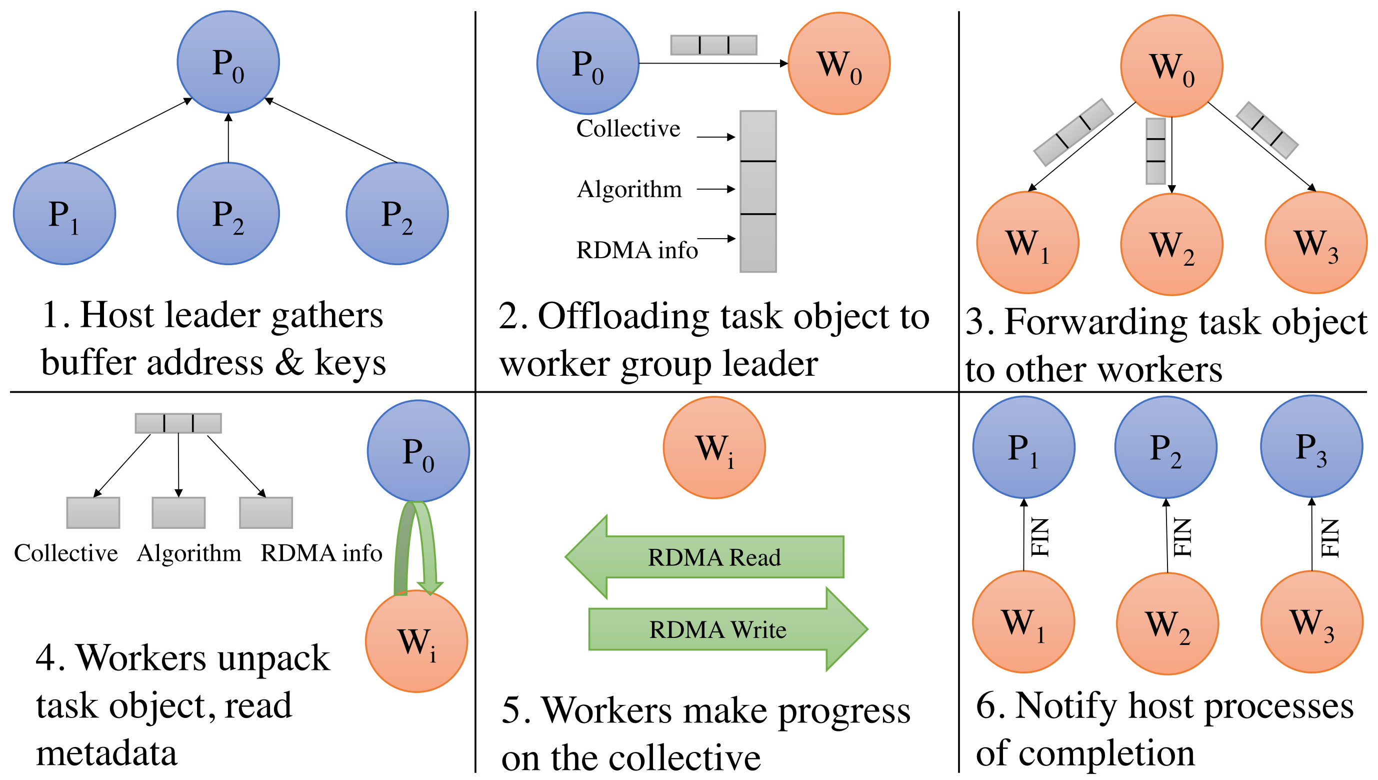



**Fig. 4.** BluesMPI procedure to offload non-blocking Alltoall collective operation to the worker processes on the Smart NIC. Step 0 is not included in this figure.

SEE PART 3 OF THIS TALK

- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- Usage model: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- **Applications**: **HPC algorithms and proxy-apps** with **aggressive restructuring** rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- Programming model: Multiprogram MPI with the DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

# Our foci (gaps)

## OpenSNAPI

- OpenSNAPI is a project of the UCF Consortium
- Straight from the source:
  - *"OpenSNAPI is a collaboration between industry, laboratories and academia with the goal to create a standard application programming interface (API) for accessing the compute engines on the network, and specifically on the smart network adapter. OpenSNAPI allows application developers to leverage the network compute cores in parallel to the host compute cores for accelerating application runtime, and to perform operations and processing closer to the data."*

SEE PART 3 OF THIS TALK

- Platform: DPUs like BF2, which are based on general-purpose multicore CPUs (e.g., generalizing INCA)
- Usage model: Off-path computation (i.e., asynchronous, independent progress) rather than on-path (i.e., "on-the-wire" computation, e.g., sPIN)
- Applications: HPC algorithms and proxy-apps with aggressive restructuring rather than relying on middleware, "basic" porting, or simple offload schemes (e.g., BluesMPI, Williams et al. PENNANT study, which found no speedup)
- **Programming model**: **Multiprogram MPI** with DPU in "host mode" rather than any vendor-specific model or lower-level communication library (e.g., OpenSNAPI)

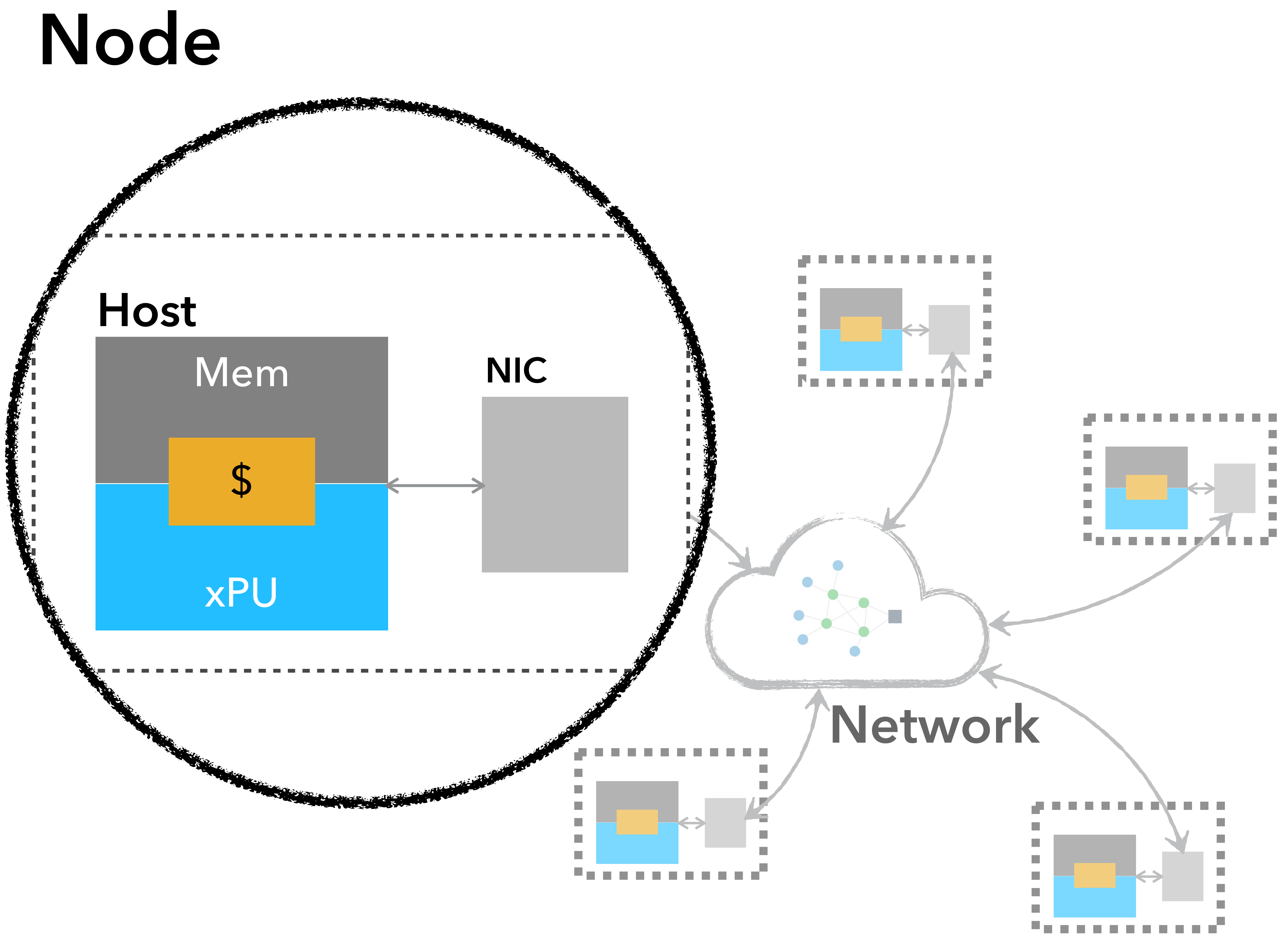
# Anatomy of a supercomputer

The basic building block of a distributed-memory cluster or supercomputer is a node.

Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A **network** connects the nodes. The nodes may be arranged in some topology, which determines the network’s carrying capacity and cost.



# DPUs in modern clusters

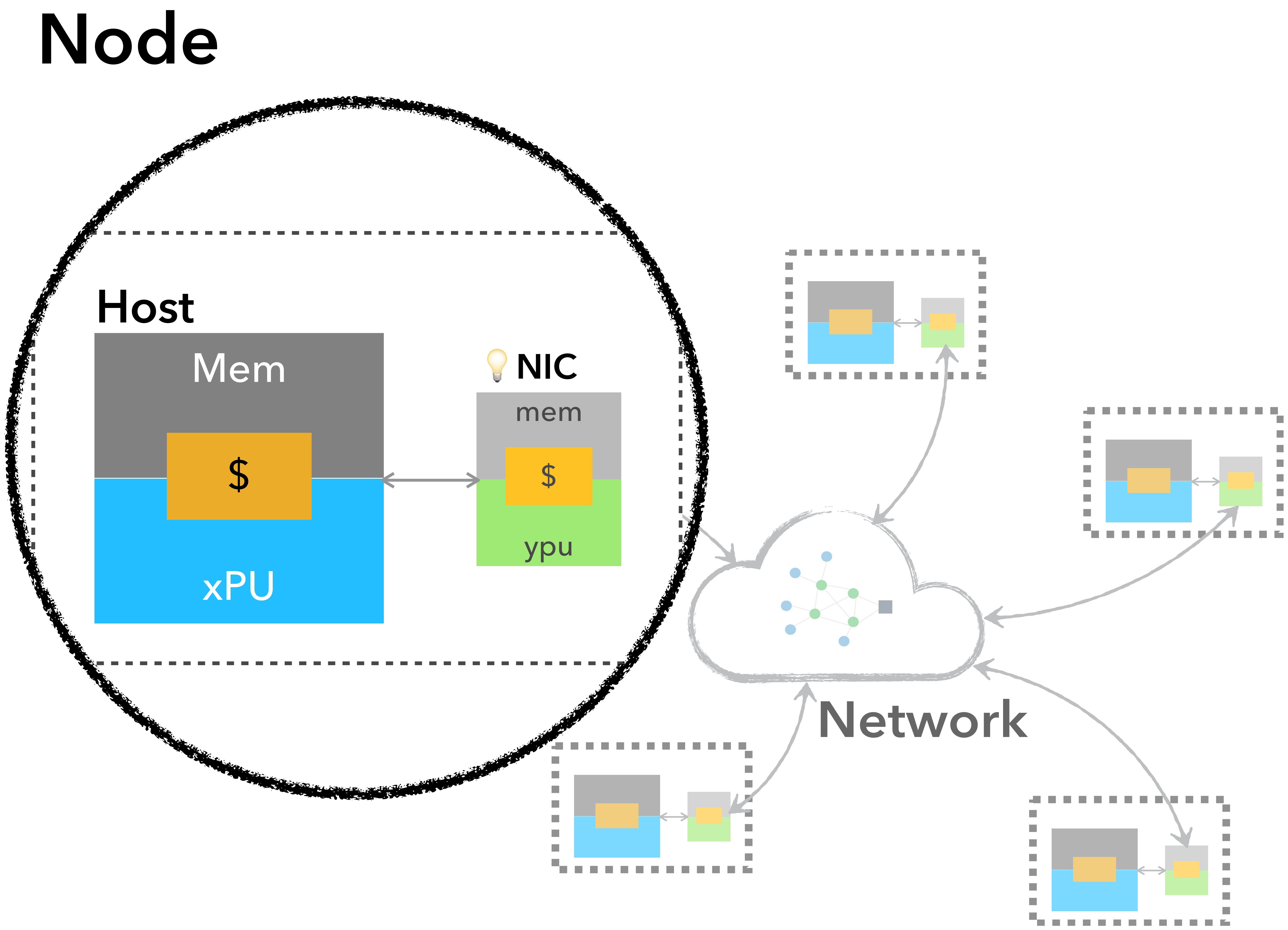
The basic building block of a distributed-memory cluster or supercomputer is a node.

Each node includes a host, which is a processor (xPU) + memory hierarchy.

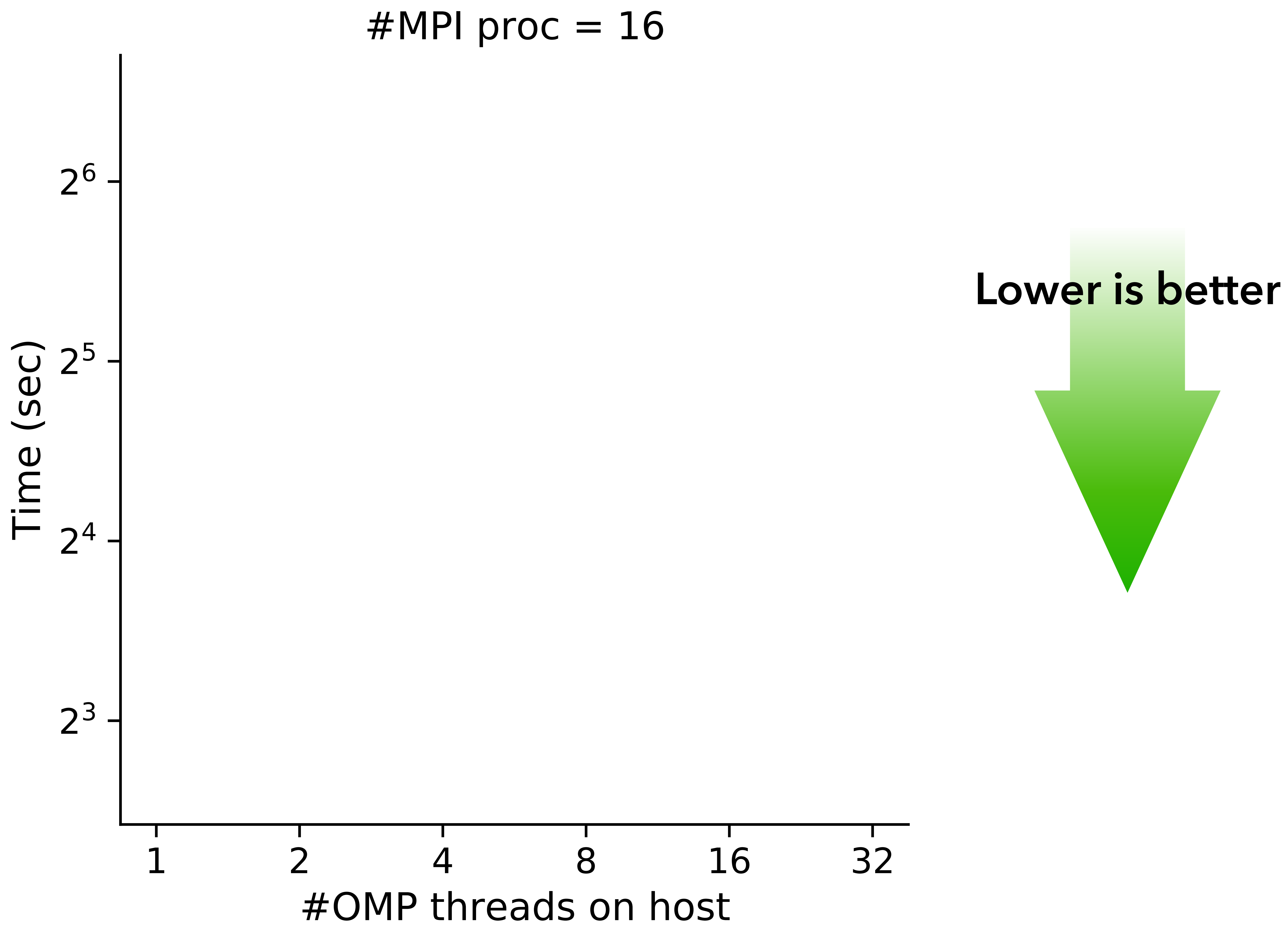
The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.

In a **smartNIC**, the NIC becomes "**host-like**" via the addition of processing (ypu) and memory.



# Hybrid MPI/OpenMP performance results



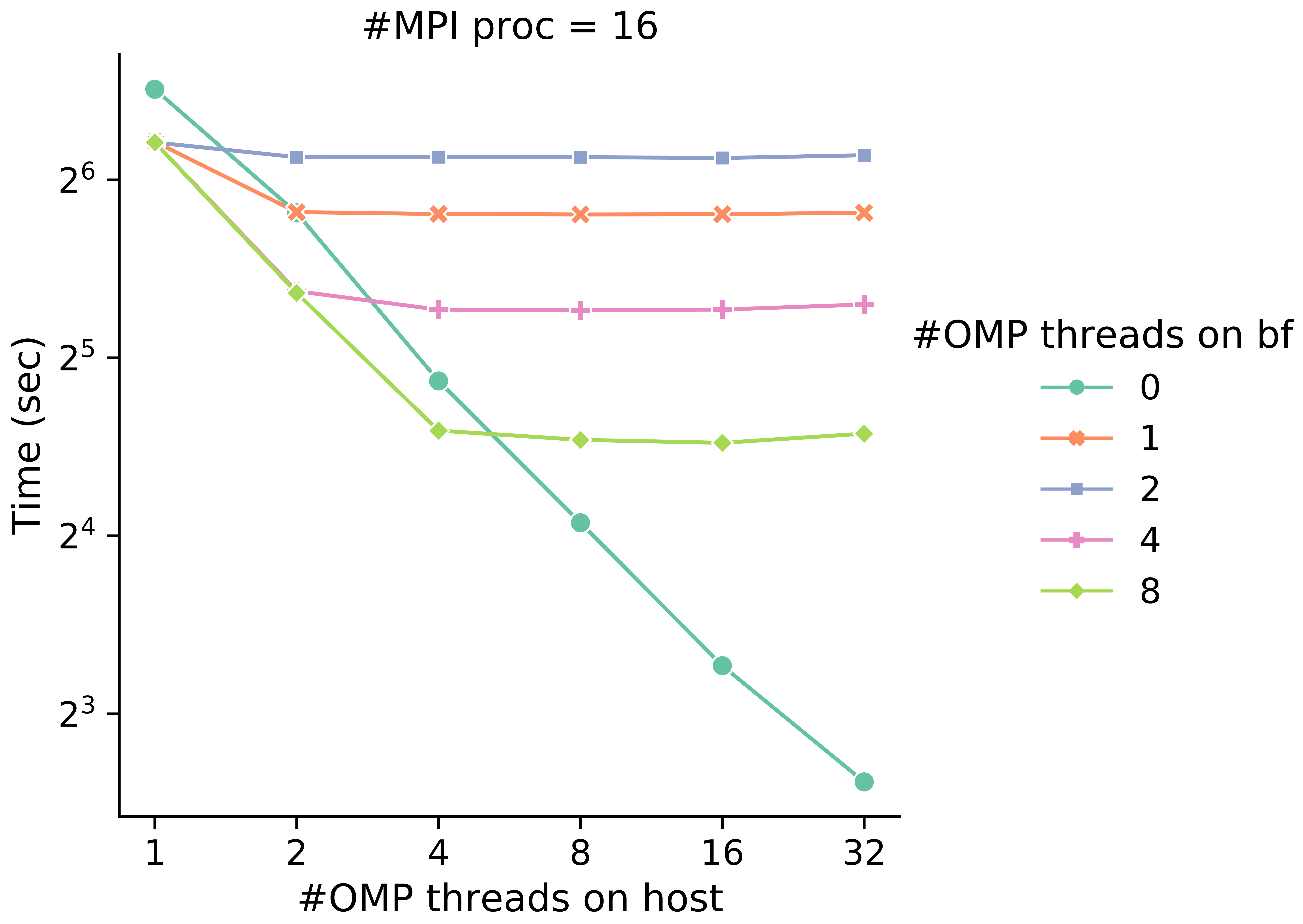
# Hybrid MPI/OpenMP performance results

Our algorithm works best when it can completely hide the force computation time on BlueField.

The degree of achievable overlap depends on the relative computational power of the host and BlueField.

The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

Thread synchronization overhead in the force computation routine causes the performance not to scale proportionally to the number of threads.



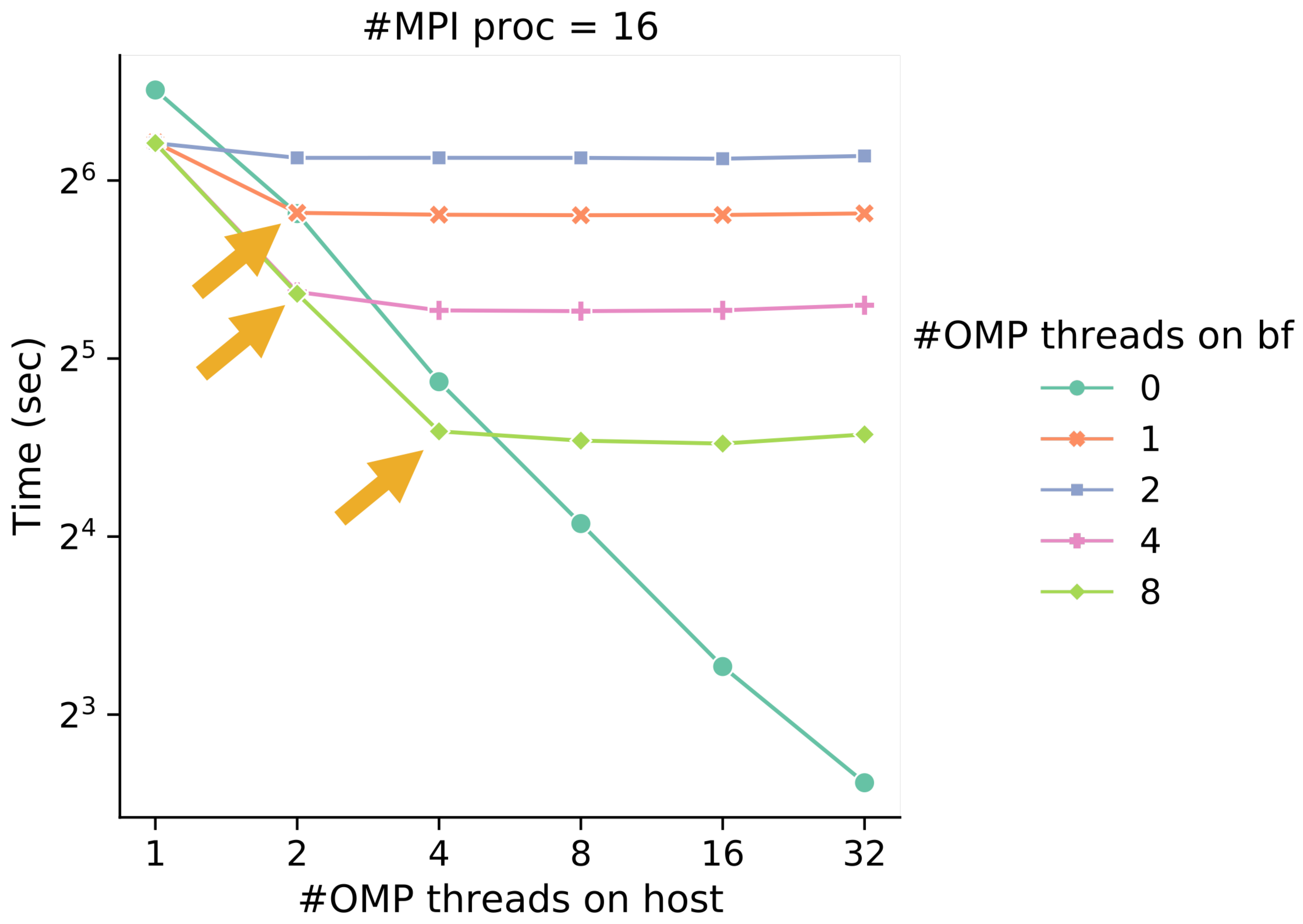
# Hybrid MPI/OpenMP performance results

Our algorithm works best when it can completely hide the force computation time on BlueField.

The degree of achievable overlap depends on the relative computational power of the host and BlueField.

The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

Thread synchronization overhead in the force computation routine causes the performance not to scale proportionally to the number of threads.



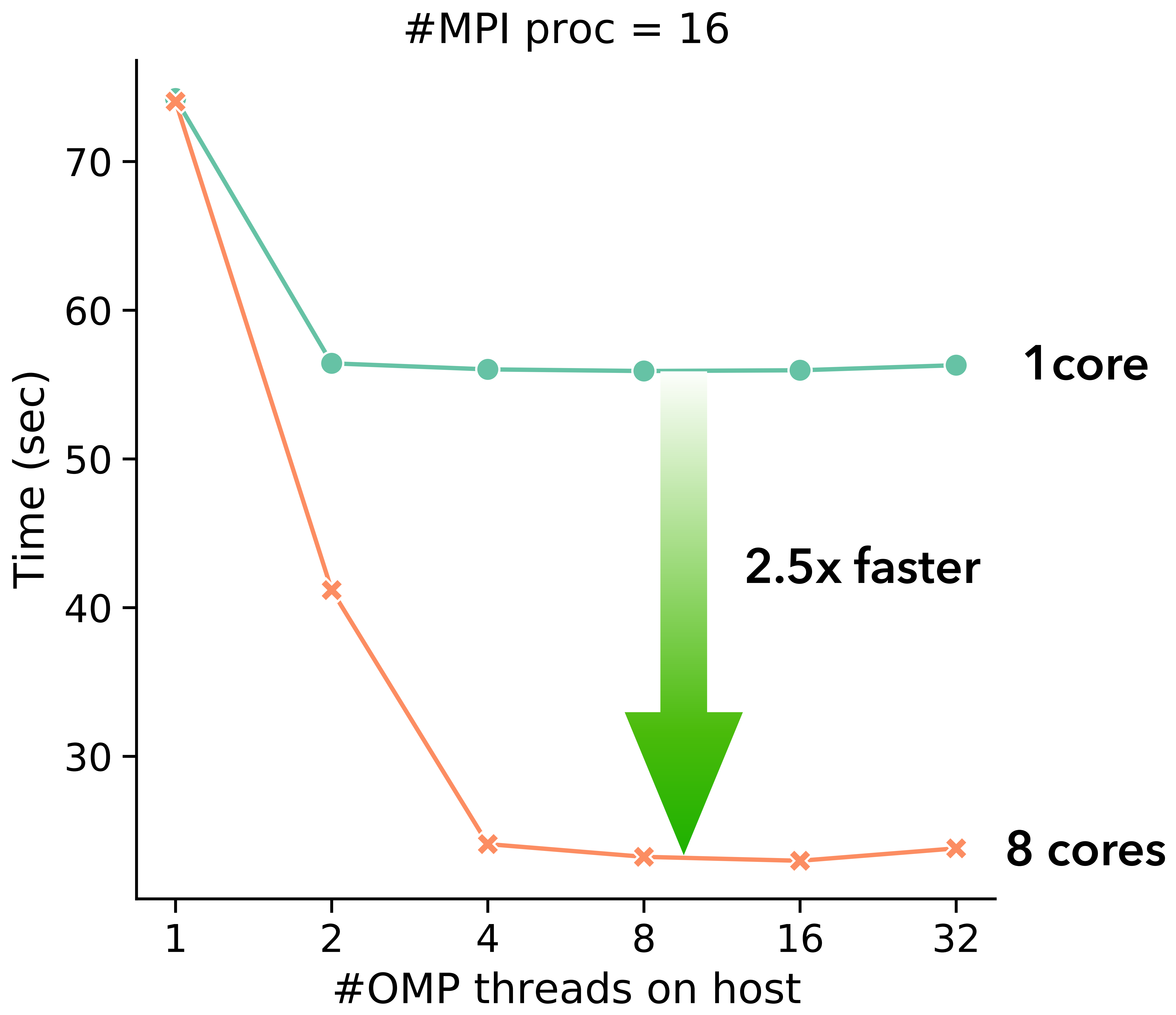
# Hybrid MPI/OpenMP performance results

Our algorithm works best when it can completely hide the force computation time on BlueField.

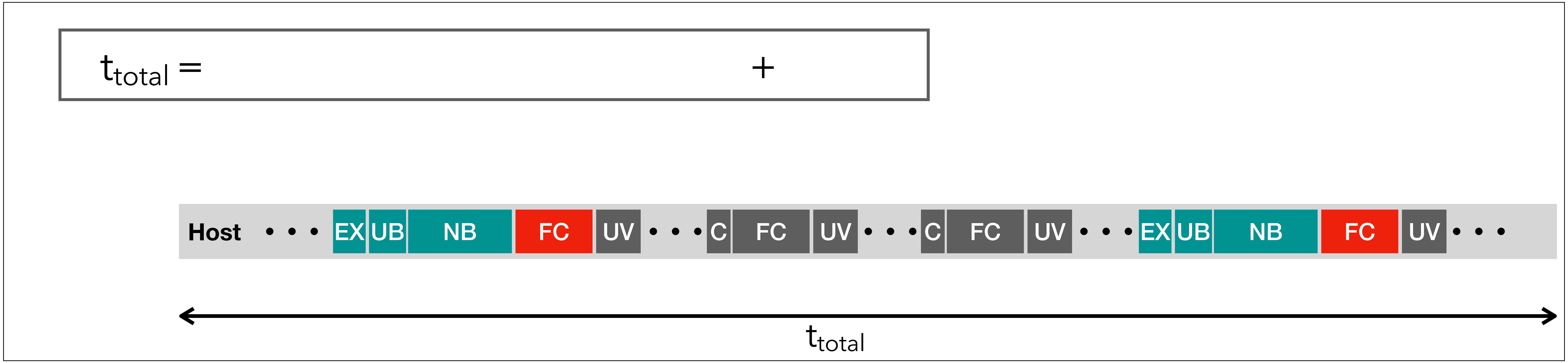
The degree of achievable overlap depends on the relative computational power of the host and BlueField.

The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

Thread synchronization overhead in the force computation routine causes the performance not to scale proportionally to the number of threads.

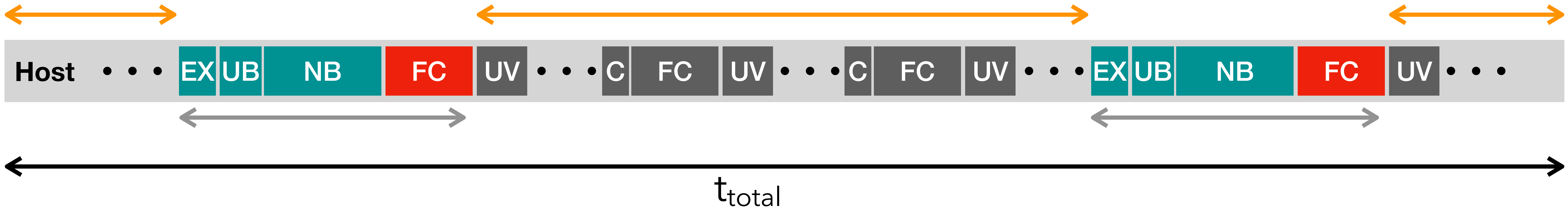


# An explanatory performance model



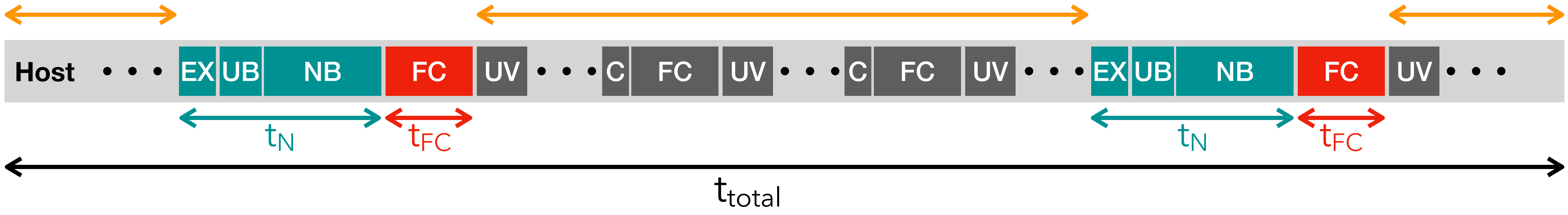
# An explanatory performance model

$$t_{total} = (\qquad\qquad) \times \#\text{re-neighboring} + t_{remain}$$



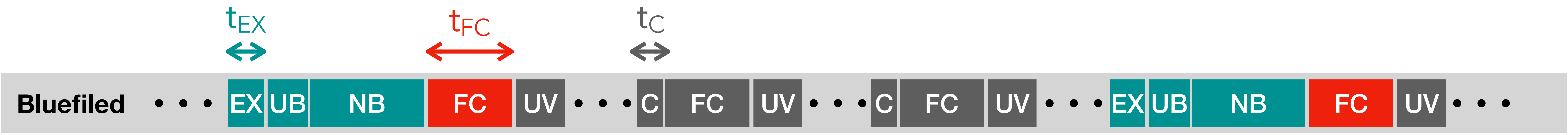
# An explanatory performance model

$$t_{total} = ( t_N + t_{FC} ) \times \#\text{re-neighboring} + t_{remain}$$



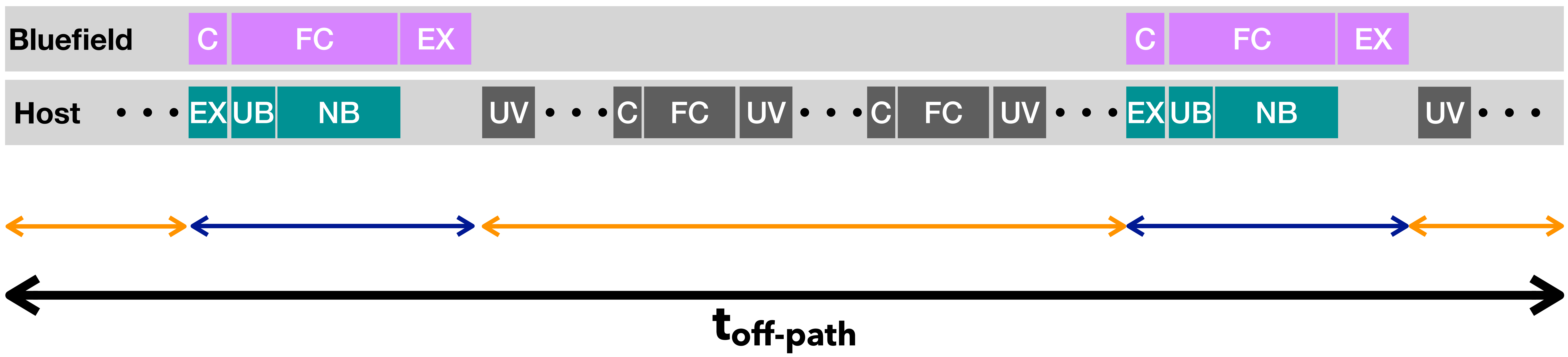
# An explanatory performance model

$$t_F = t_{EX} + t_{FC} + t_C$$



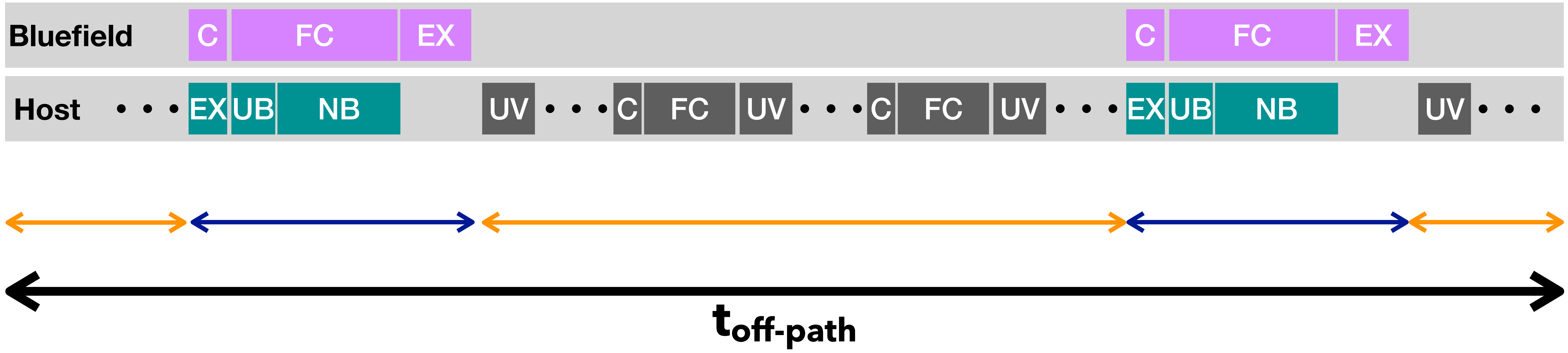
# An explanatory performance model

$t_{off\text{-}path}$ = +



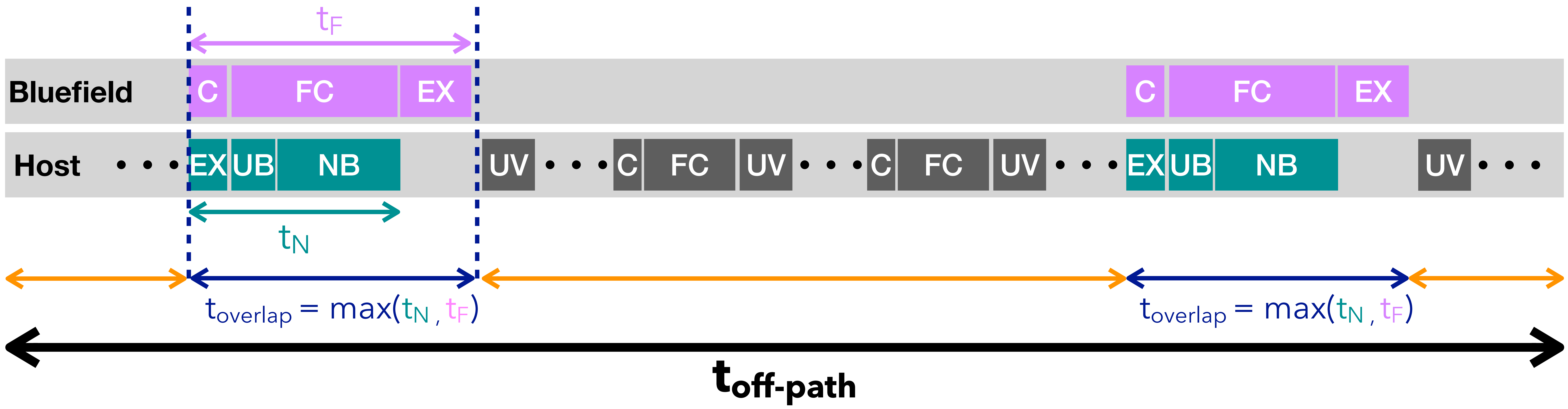
# An explanatory performance model

$t_{off\text{-}path} = t_{remain} + (\qquad) \times$ #re-neighboring



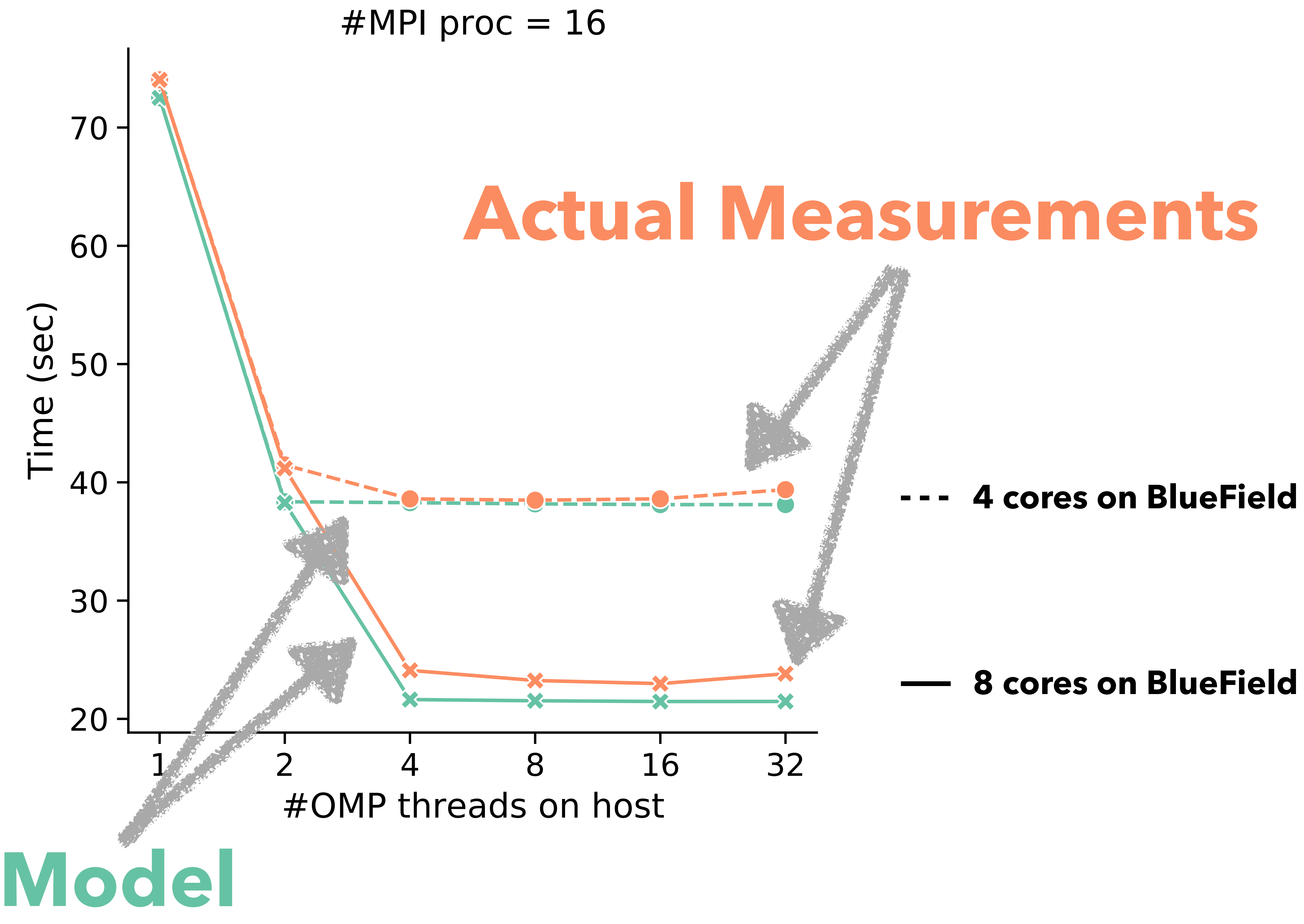
# An explanatory performance model

$$t_{\text{off-path}} = t_{\text{remain}} + \max(t_N, t_F) \times \#\text{re-neighboring}$$

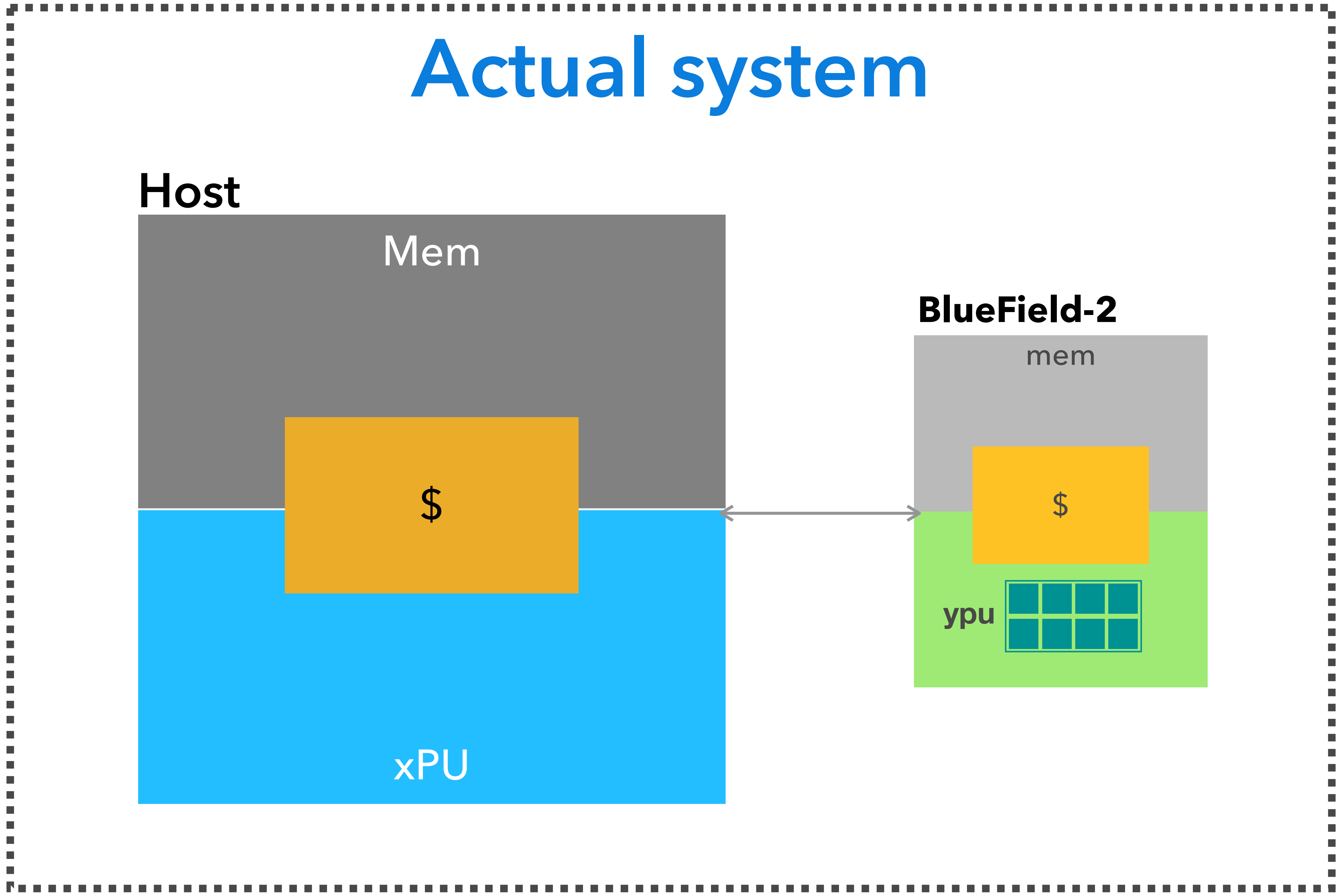


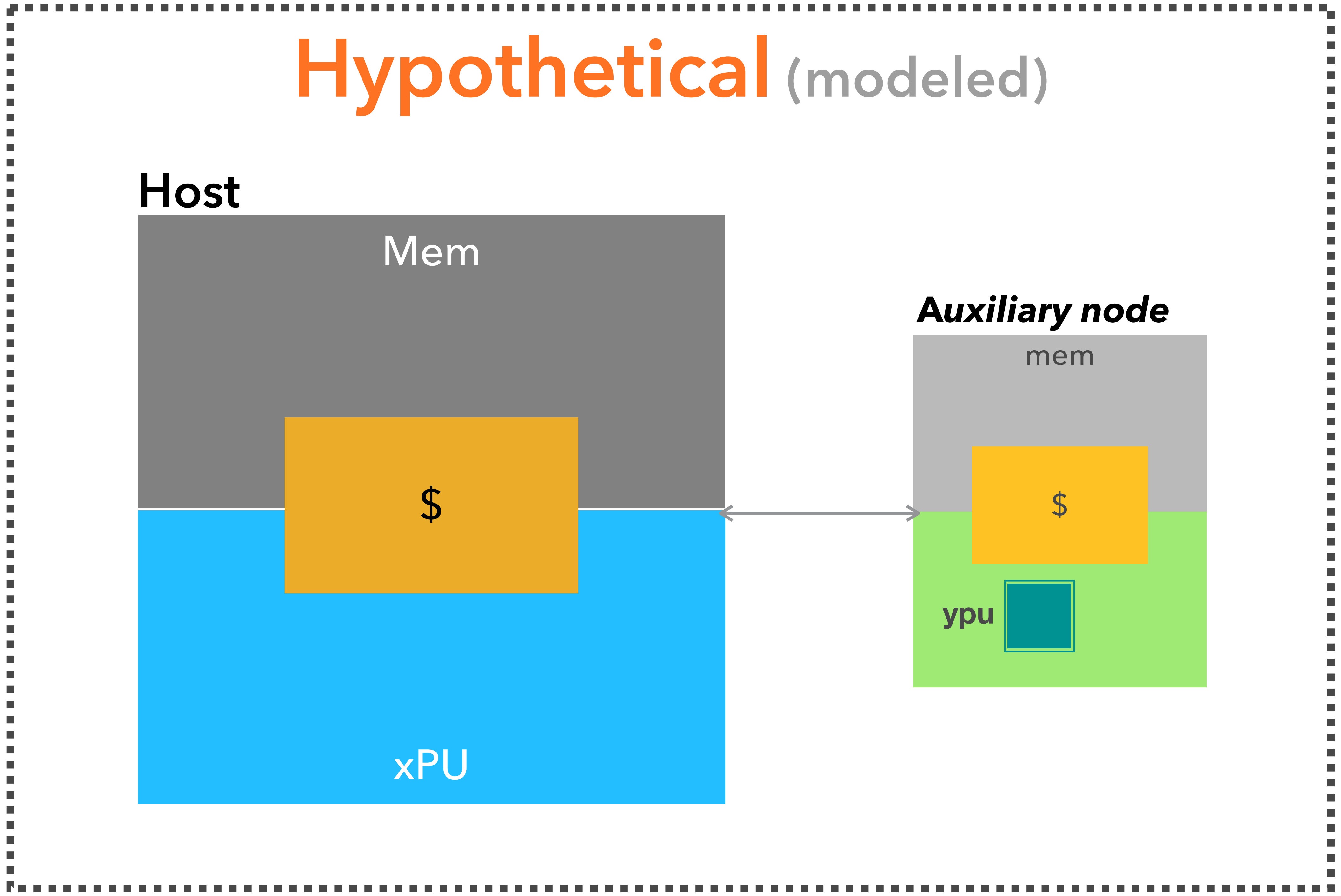
# Predictive power of our performance model

The model can closely predict the algorithm runtime.

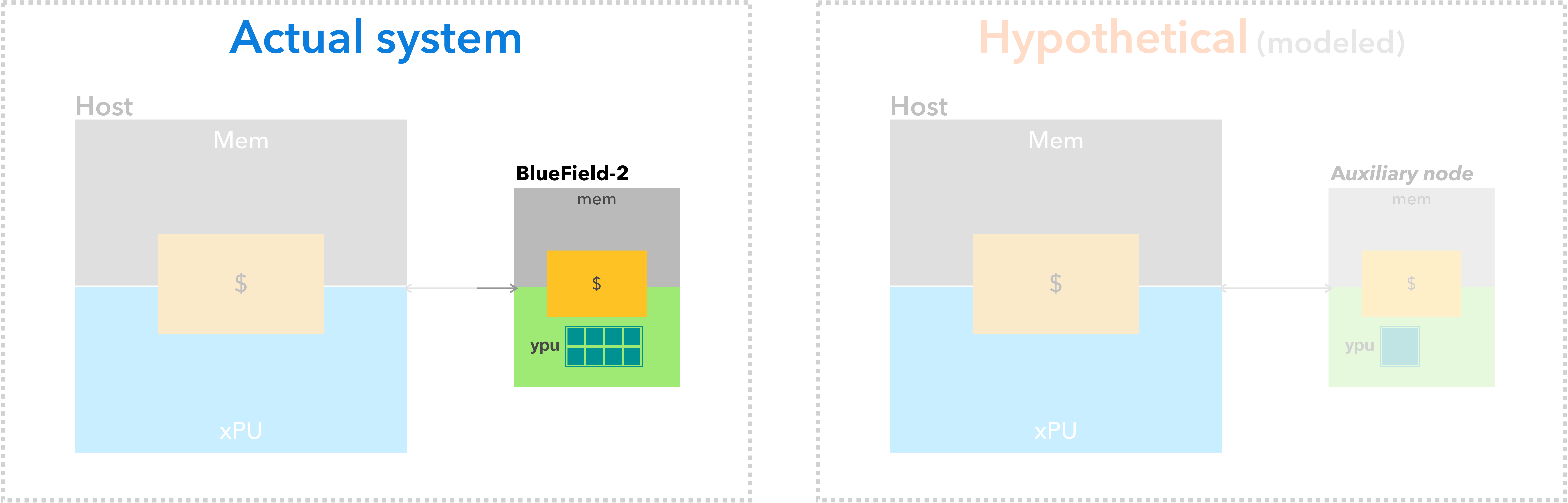


# How about a DPU with more performant core?

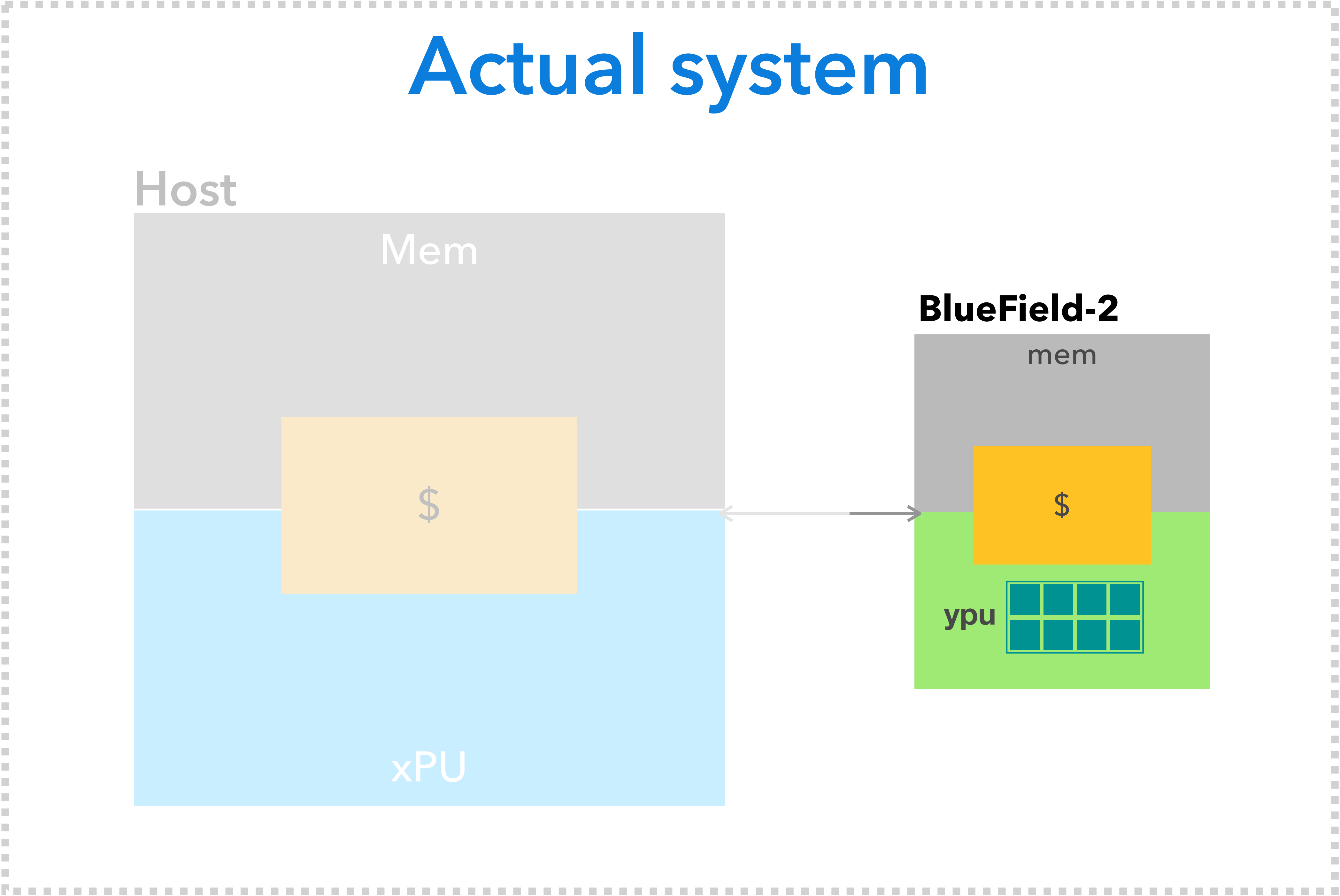


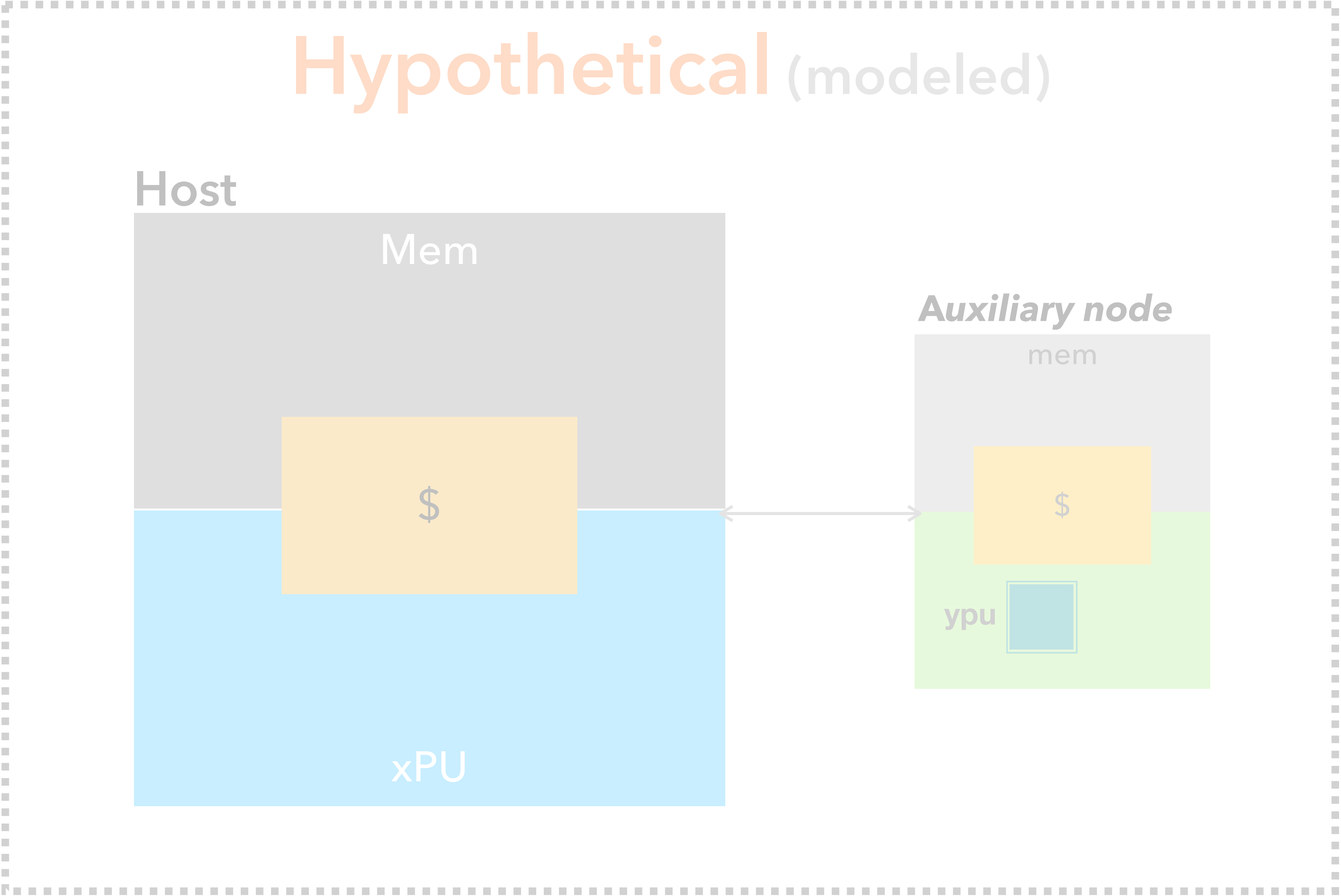



# How about a DPU with more performant core?



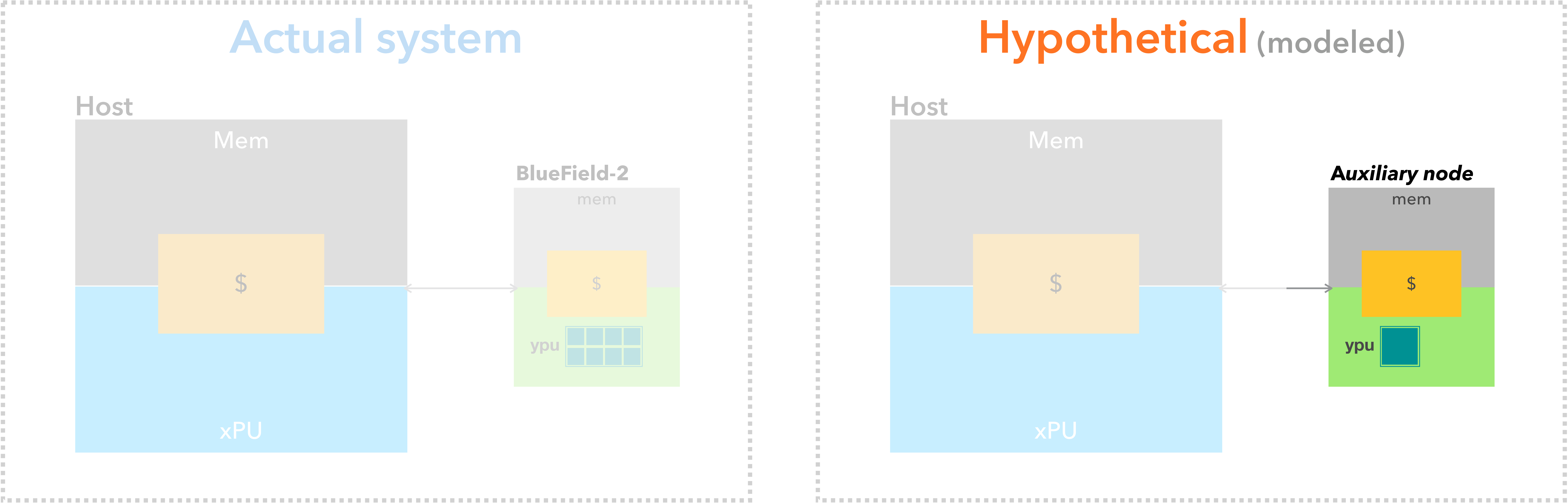
# How about a DPU with more performant core?




## 80 GF/s peak (8 cores)

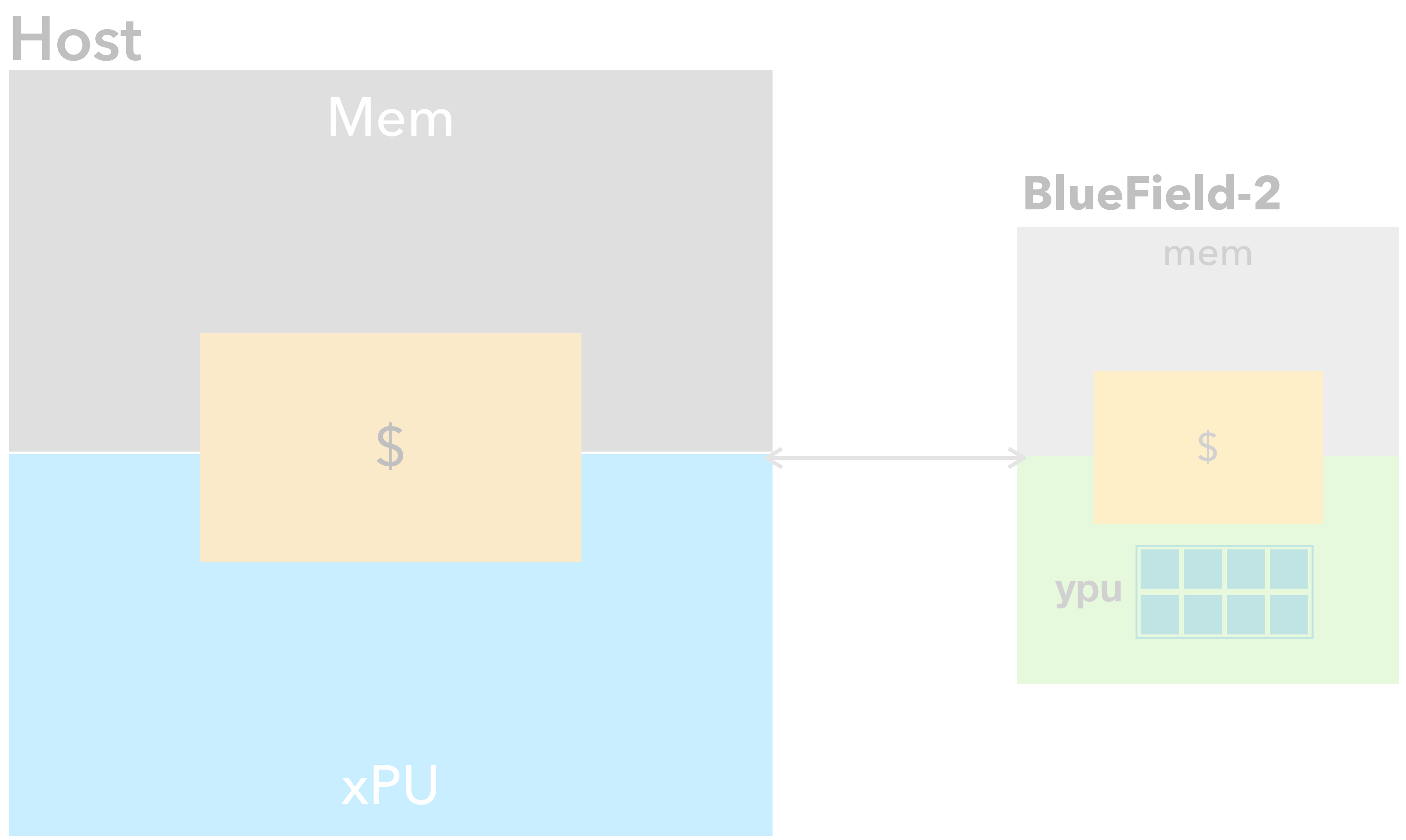
# How about a DPU with more performant core?




80 GF/s peak (8 cores)

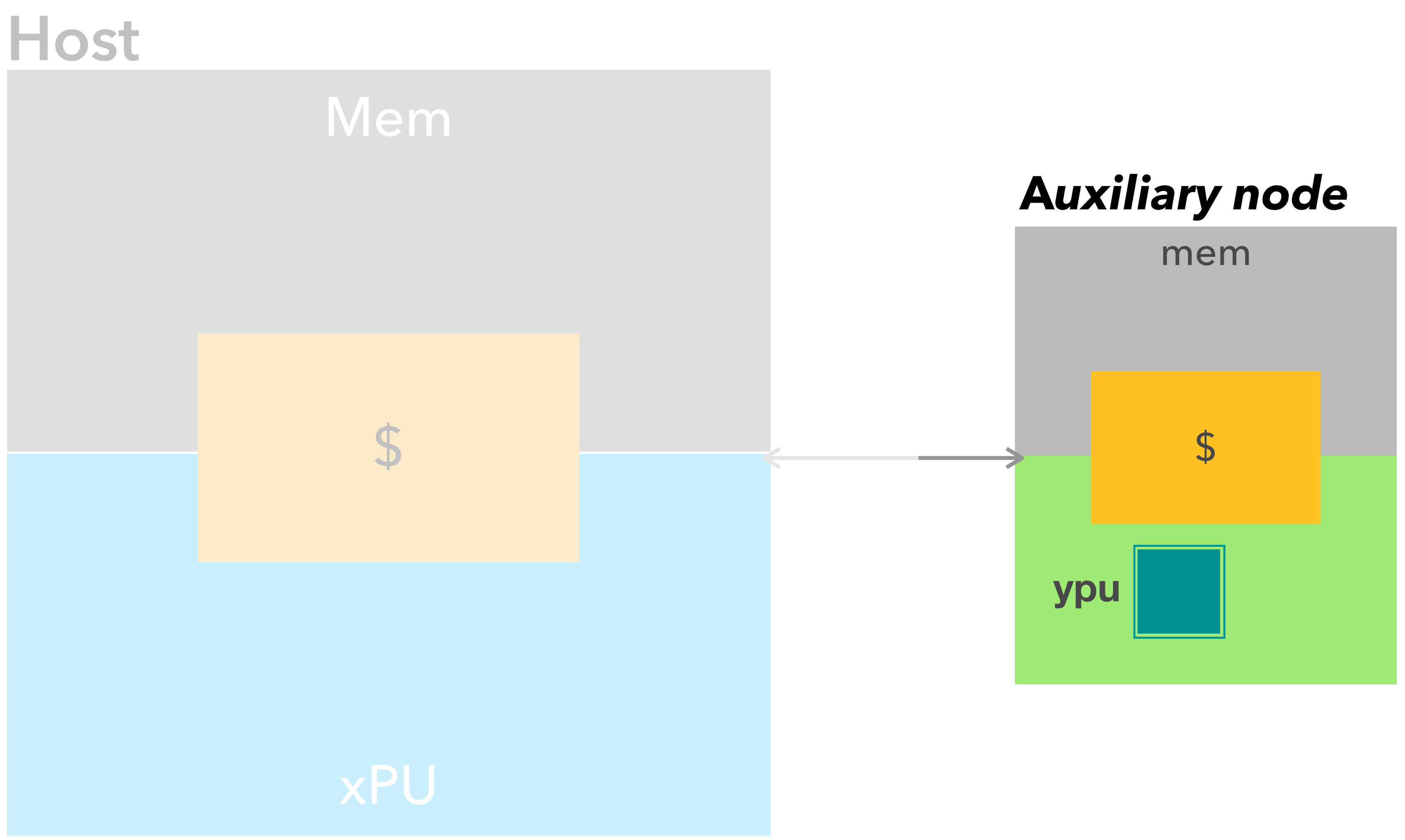
# How about a DPU with more performant core?

## Actual system



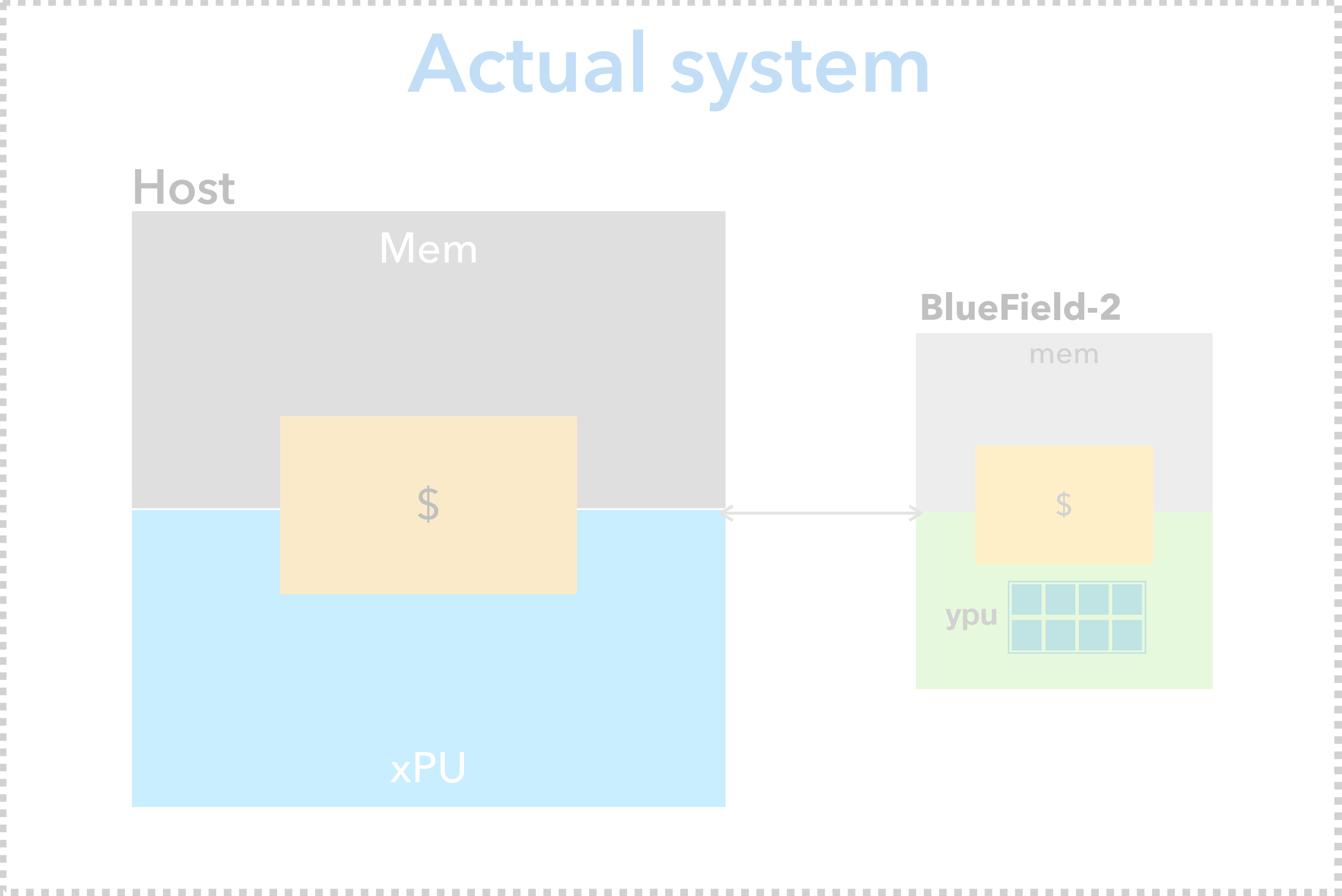

**80 GF/s peak (8 cores)**

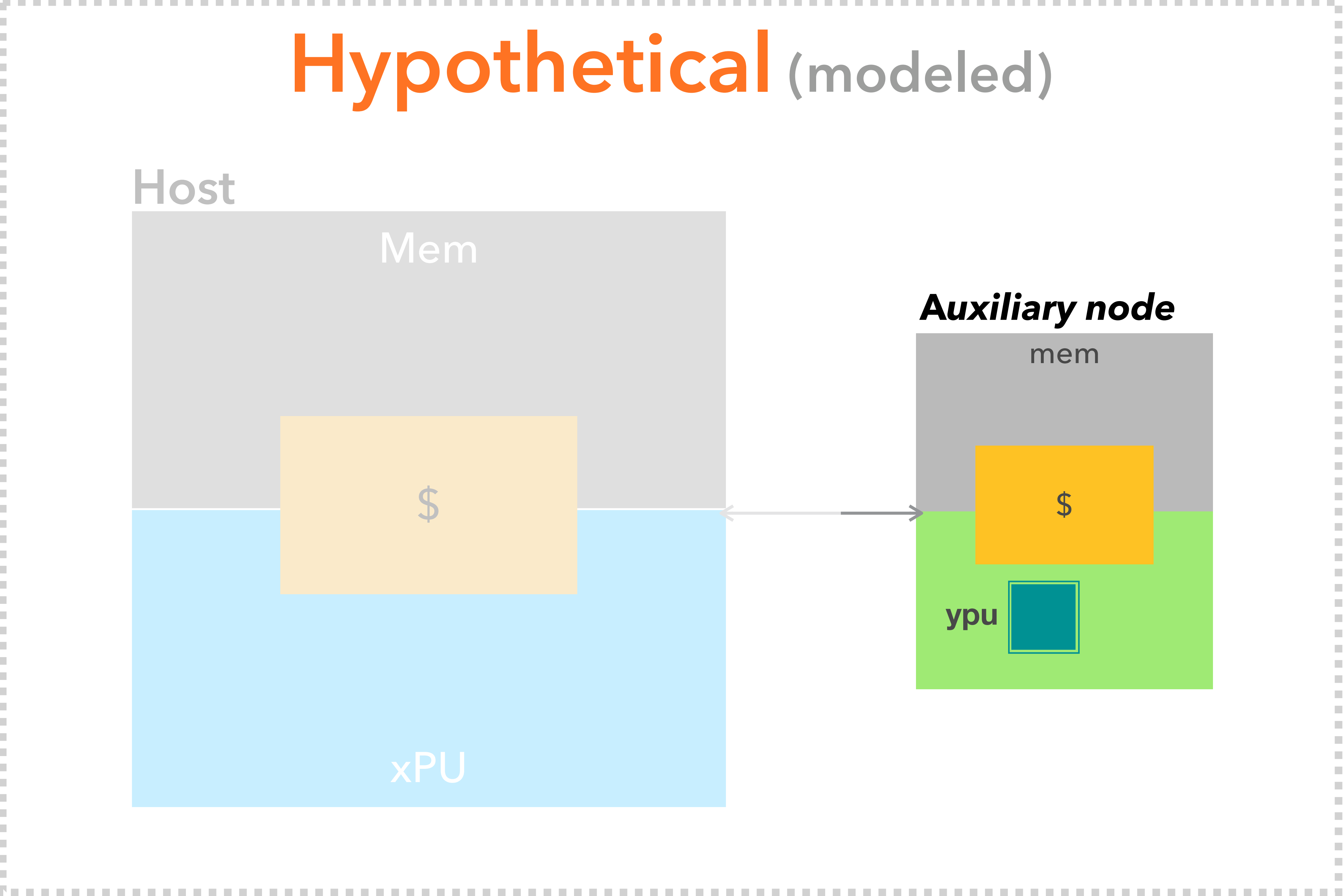
## Hypothetical (modeled)




**40 GF/s peak (1 core)**

# How about a DPU with more performant core?



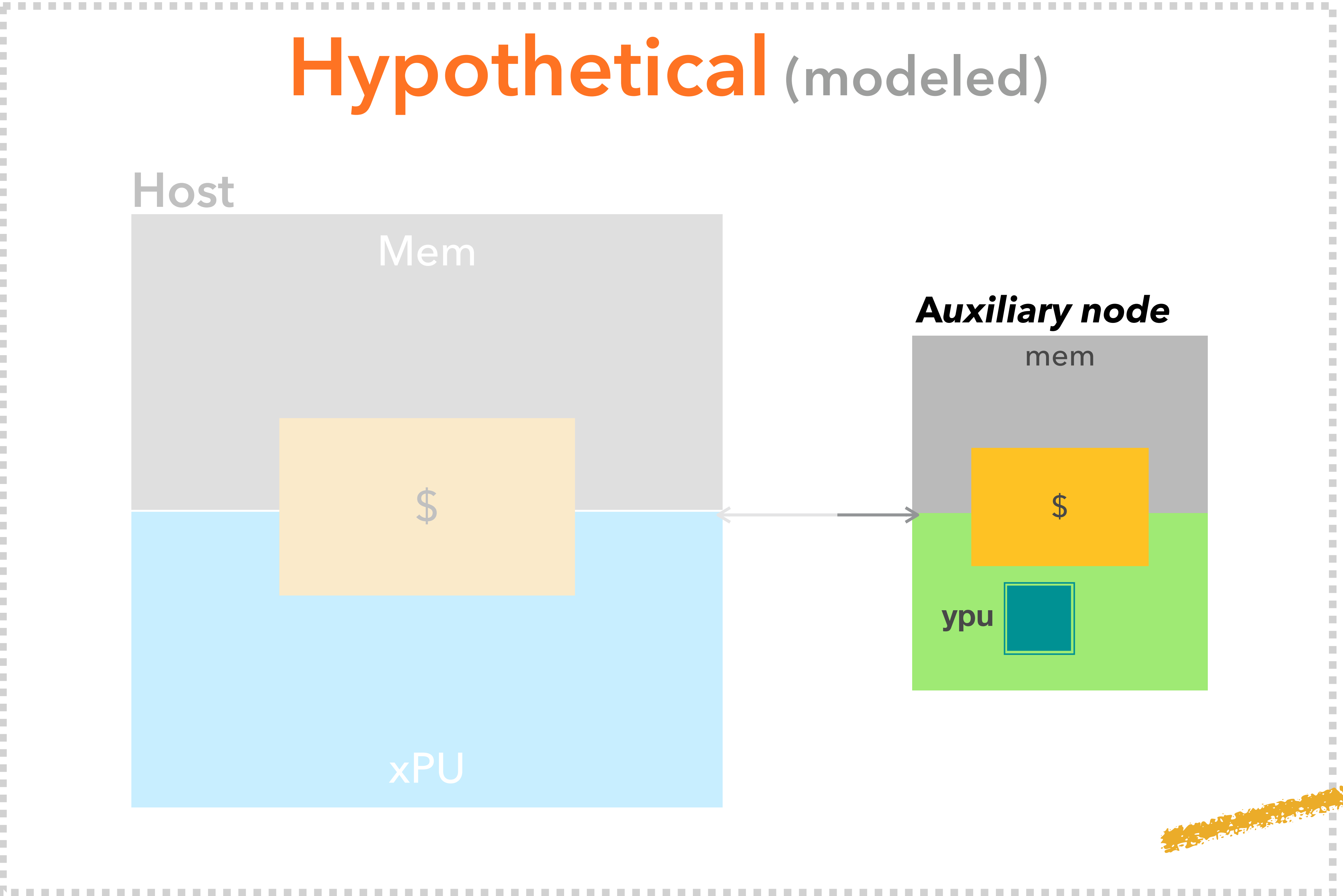




**80 GF/s peak (8 cores)**

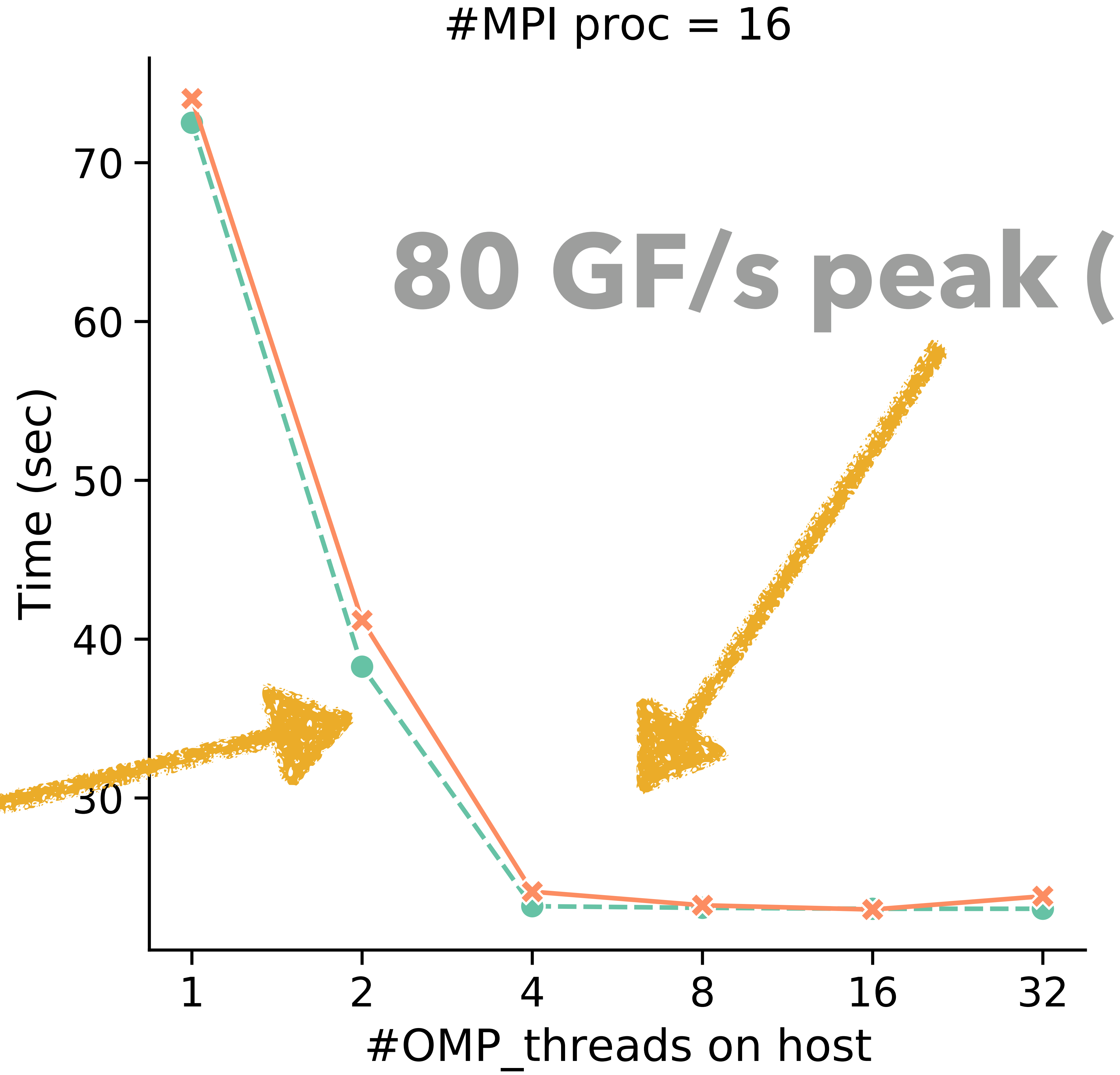
**40 GF/s peak (1 core)**

# Same predicted performance!

(due to no sync overhead on aux node)




40 GF/s peak (1 core)




80 GF/s peak (8 cores)

# Other highlights from the paper

Conducted performance analysis using the **OSU Microbenchmarks suite** and the "off-the-shelf" version of MiniMD to understand the opportunities and limitations presented by the BlueField for potential HPC applications.

Verified that the computed simulation results for the restructured method are still within an **acceptable level of accuracy**, in terms of calculated physical quantities.

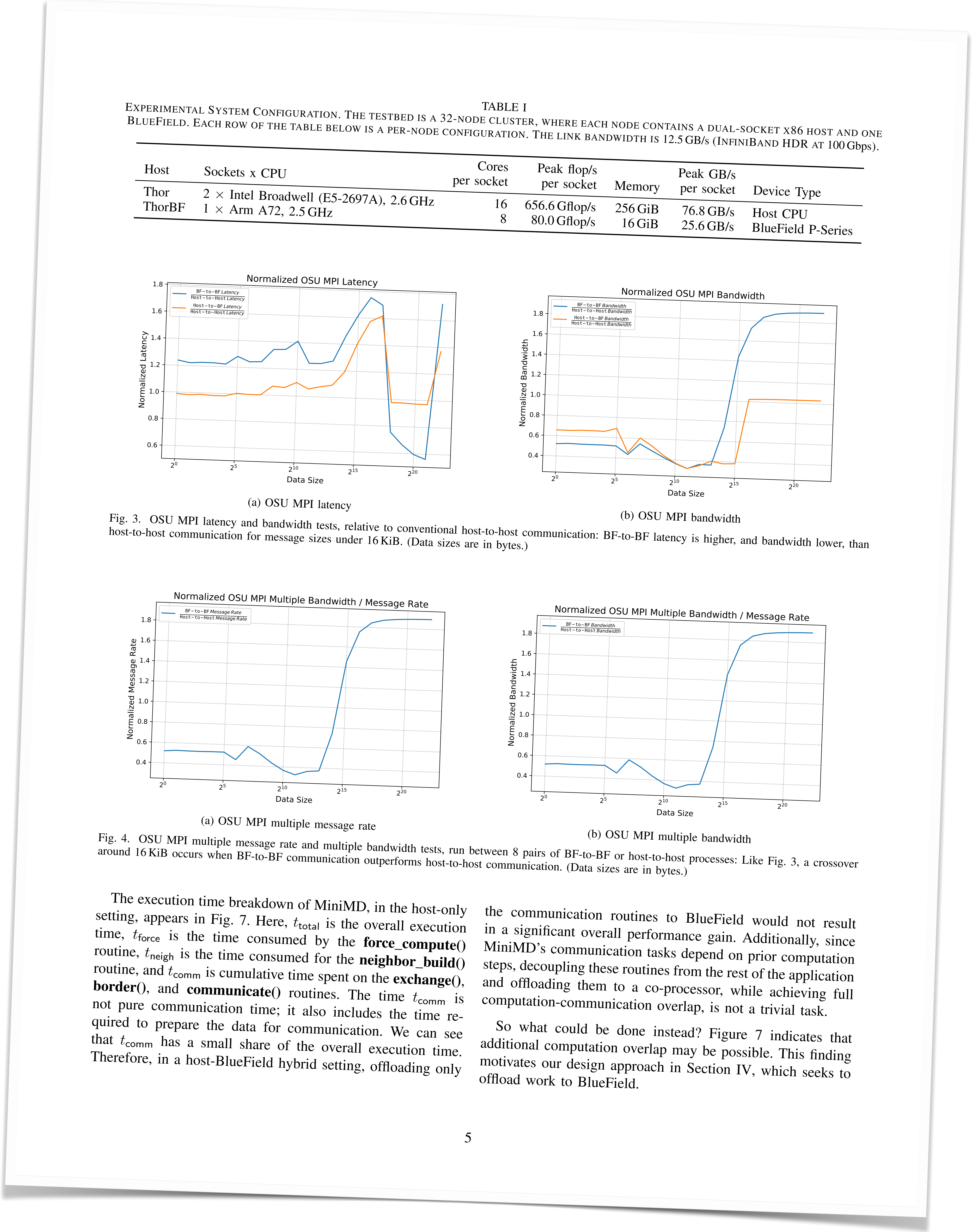


TABLE I

EXPERIMENTAL SYSTEM CONFIGURATION. THE TESTBED IS A 32-NODE CLUSTER, WHERE EACH NODE CONTAINS A DUAL-SOCKET X86 HOST AND ONE BLUEFIELD. EACH ROW OF THE TABLE BELOW IS A PER-NODE CONFIGURATION. THE LINK BANDWIDTH IS 12.5 GB/s (INFINIBAND HDR AT 100 Gbps).

| Host | Sockets x CPU | Cores per socket | Peak flop/s per socket | Memory | Peak GB/s per socket | Device Type |
|---|---|---|---|---|---|---|
| Thor | 2 × Intel Broadwell (E5-2697A), 2.6 GHz | 16 | 656.6 Gflop/s | 256 GiB | 76.8 GB/s | Host CPU |
| ThorBF | 1 × Arm A72, 2.5 GHz | 8 | 80.0 Gflop/s | 16 GiB | 25.6 GB/s | BlueField P-Series |



(a) OSU MPI latency

(b) OSU MPI bandwidth

Fig. 3. OSU MPI latency and bandwidth tests, relative to conventional host-to-host communication: BF-to-BF latency is higher, and bandwidth lower, than host-to-host communication for message sizes under 16 KiB. (Data sizes are in bytes.)



(a) OSU MPI multiple message rate

(b) OSU MPI multiple bandwidth

Fig. 4. OSU MPI multiple message rate and multiple bandwidth tests, run between 8 pairs of BF-to-BF or host-to-host processes: Like Fig. 3, a crossover around 16 KiB occurs when BF-to-BF communication outperforms host-to-host communication. (Data sizes are in bytes.)

The execution time breakdown of MiniMD, in the host-only setting, appears in Fig. 7. Here, $t_{total}$ is the overall execution time, $t_{force}$ is the time consumed by the **force_compute()** routine, $t_{neigh}$ is the time consumed for the **neighbor_build()** routine, and $t_{comm}$ is cumulative time spent on the **exchange()**, **border()**, and **communicate()** routines. The time $t_{comm}$ is not pure communication time; it also includes the time required to prepare the data for communication. We can see that $t_{comm}$ has a small share of the overall execution time. Therefore, in a host-BlueField hybrid setting, offloading only the communication routines to BlueField would not result in a significant overall performance gain. Additionally, since MiniMD's communication tasks depend on prior computation steps, decoupling these routines from the rest of the application and offloading them to a co-processor, while achieving full computation-communication overlap, is not a trivial task.

So what could be done instead? Figure 7 indicates that additional computation overlap may be possible. This finding motivates our design approach in Section IV, which seeks to offload work to BlueField.

5

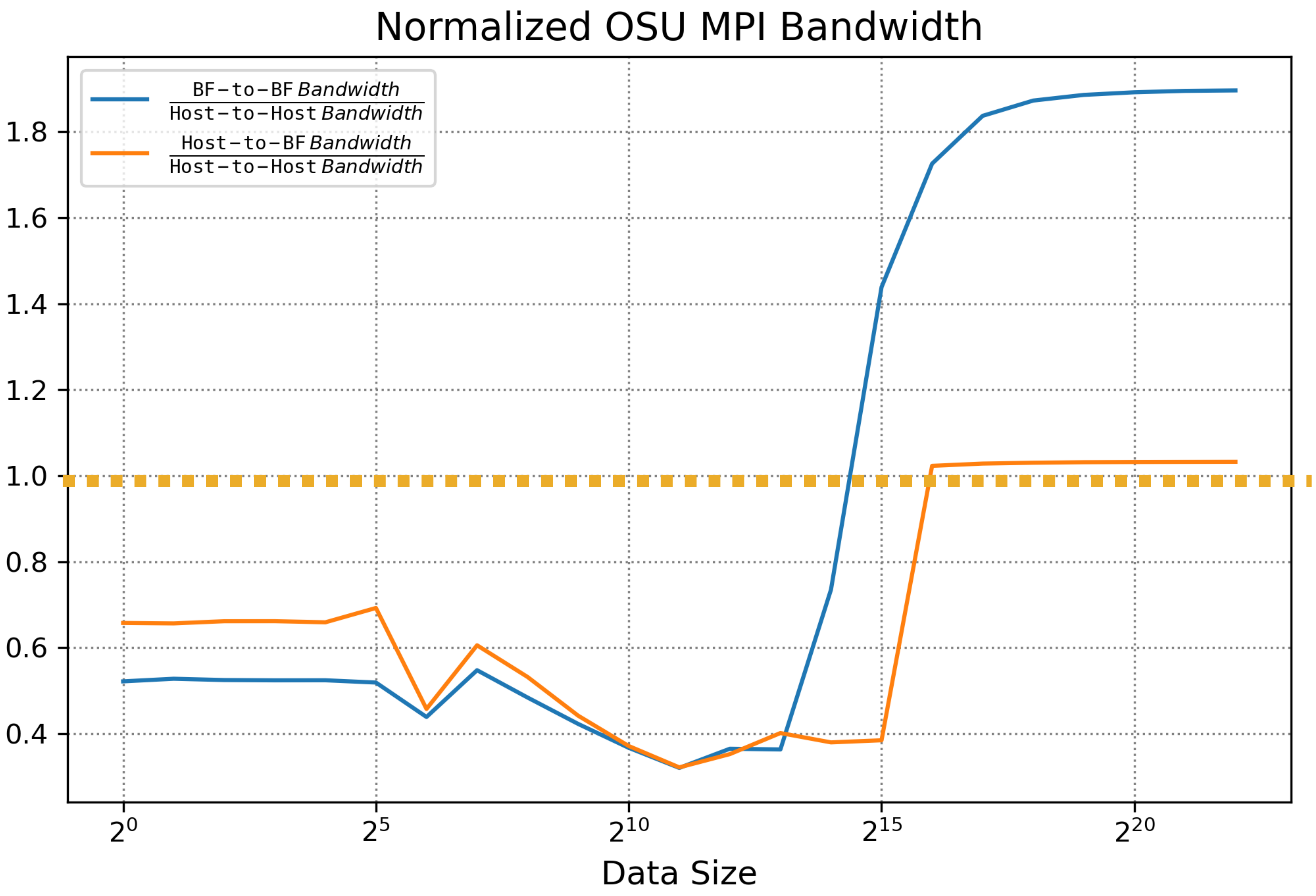
# OSU benchmarks imply off-path mode

(Blue line) For message sizes under 16 KiB, BF2-to-BF2 communication is **slower** than host-to-host communication using conventional Infiniband NICs.

(Orange line) For messages under 64 KiB, it is even **slower** to exchange messages between the host and the BF2 **on the same node**!

Similar findings hold for multi-pair communication and all-gather operations.

Thus, our best bet for getting any performance improvement will be via **off-path** execution.



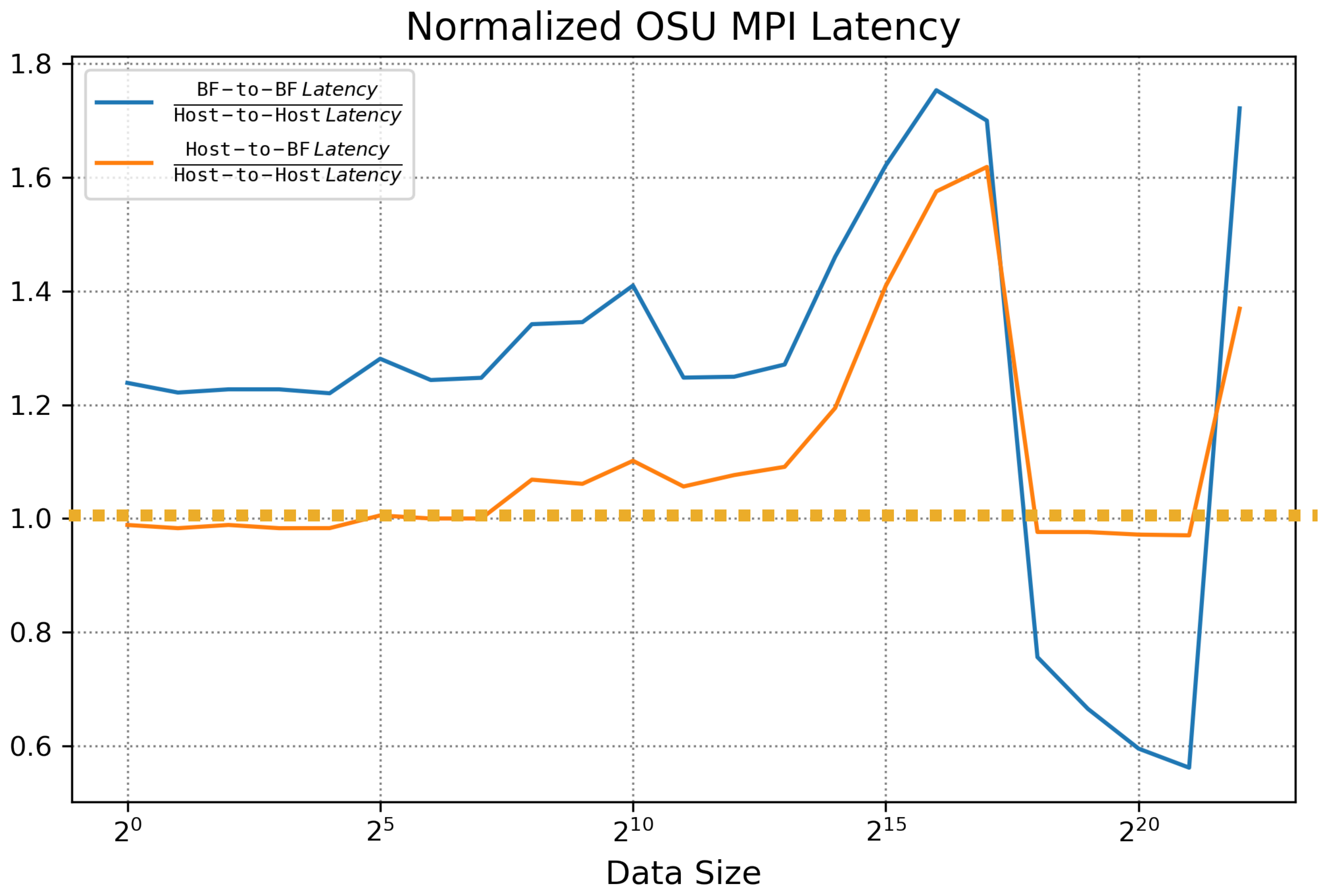
# OSU benchmarks imply off-path mode

(Blue line) For message sizes under 128 KiB, the latency of BF2-to-BF2 communication is **higher** than host-to-host communication using conventional Infiniband NICs.

(Orange line) For messages under 256 KiB, it is even **slower** to exchange messages between the host and the BF2 **on the same node**!

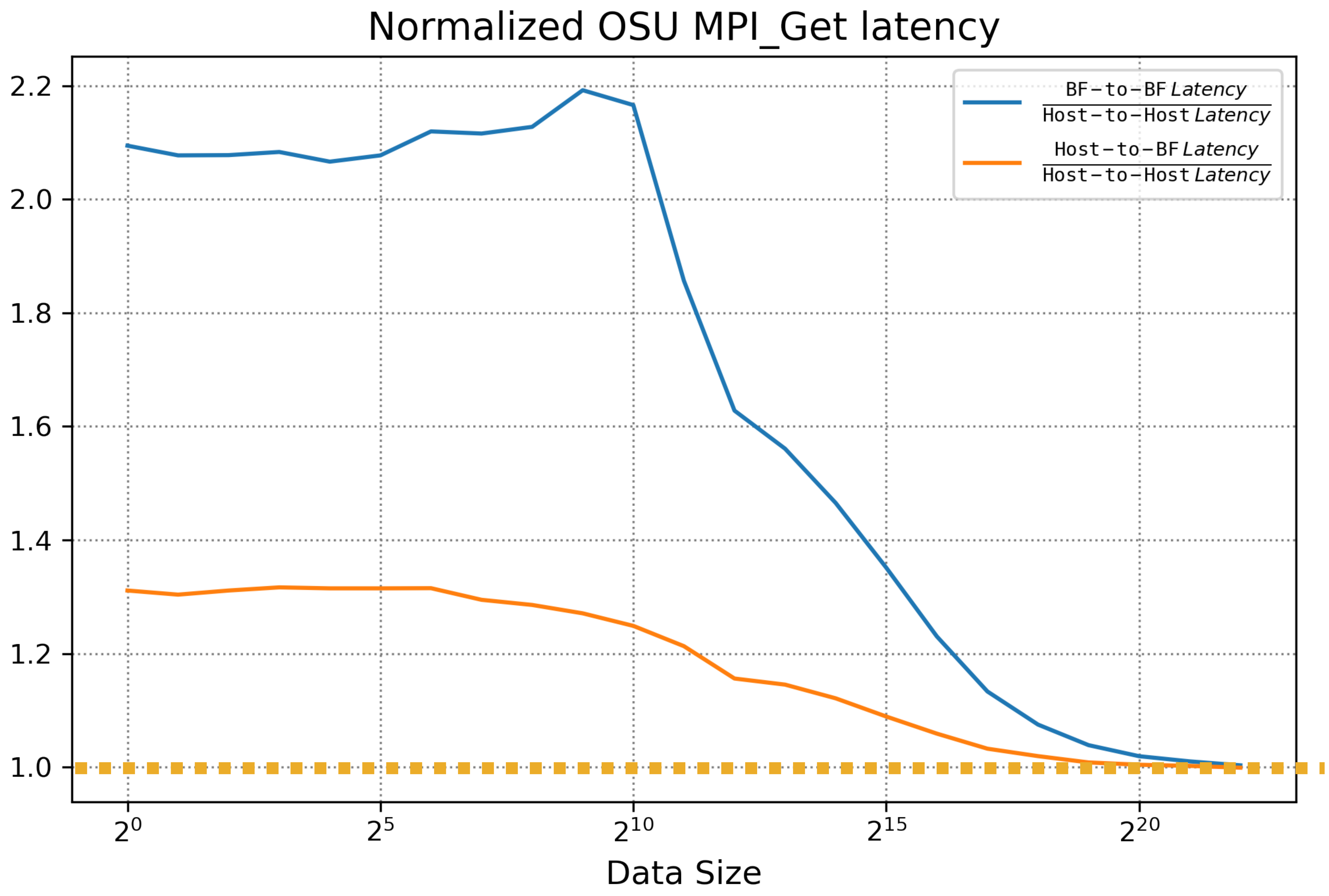
Similar findings hold for multi-pair communication and all-gather operations.

Thus, our best bet for getting any performance improvement will be via **off-path** execution.



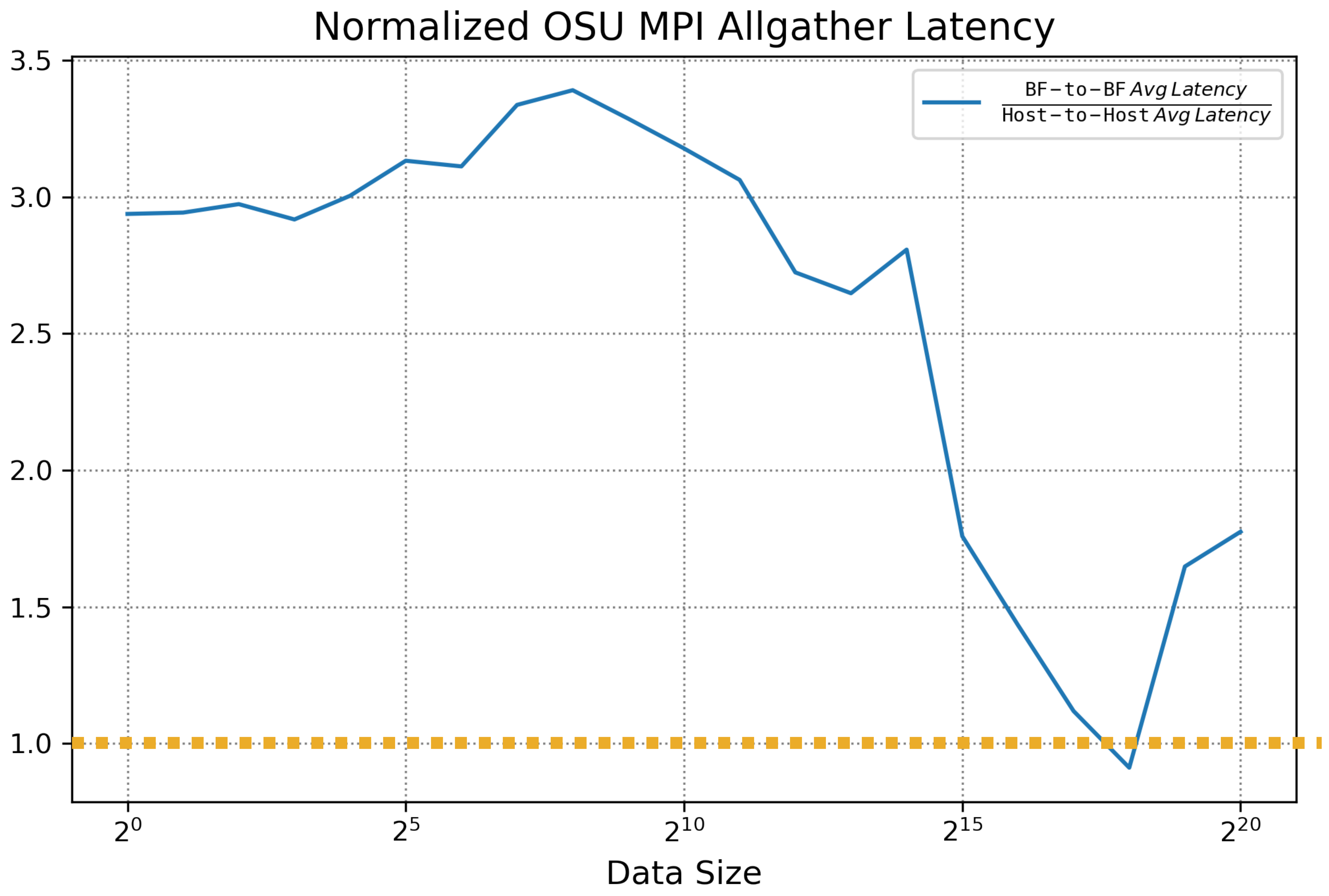
# OSU benchmarks imply off-path mode

Even BF2 **one-sided communication** is **slower** than conventional Infiniband.



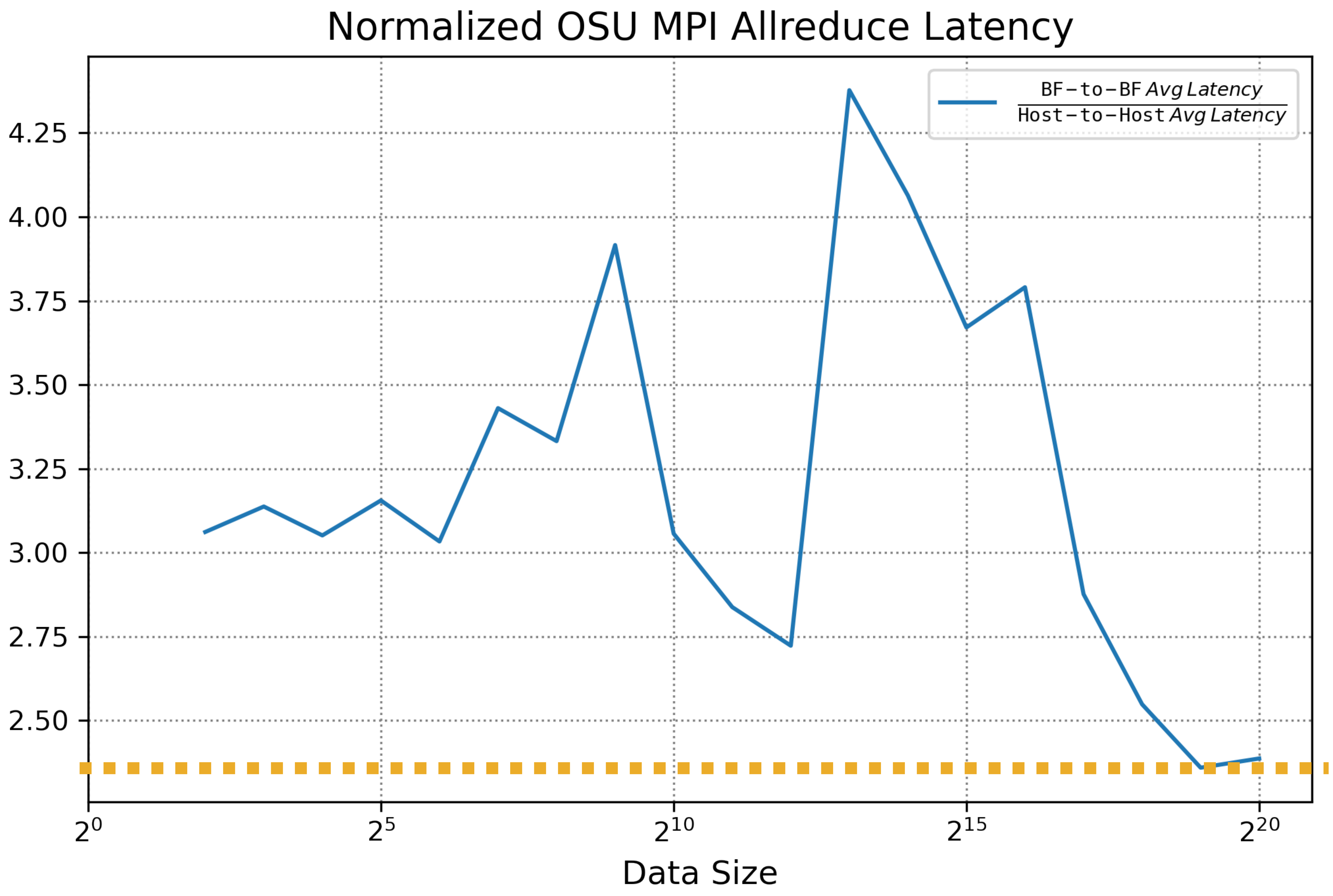
# OSU benchmarks imply off-path mode

The time to complete a BlueField-to-BlueField **all-gather is always worse**—up to 3x—than via conventional NICs.



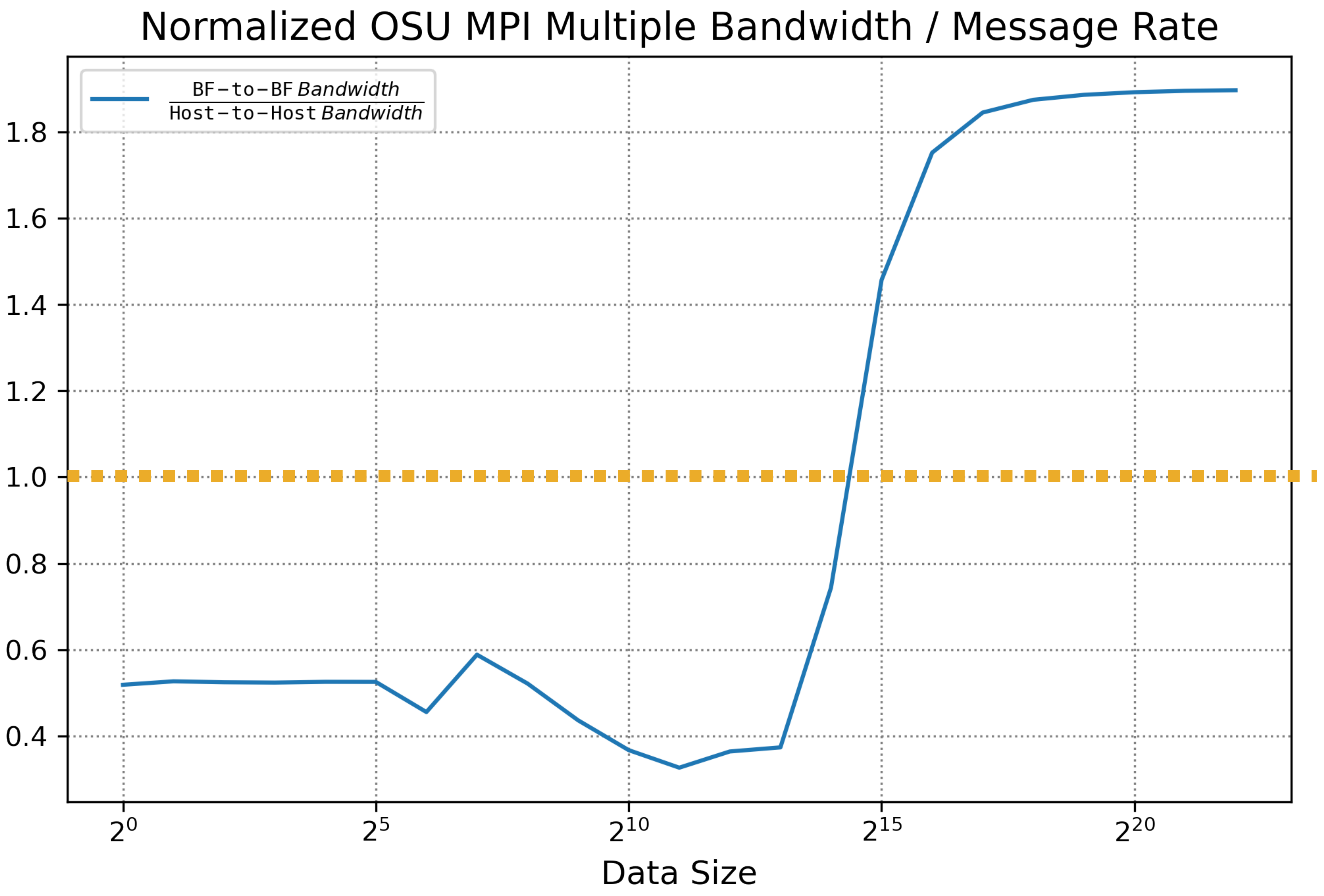
# OSU benchmarks imply off-path mode

The time to complete a BlueField-to-BlueField **all-reduce is always worse**–up to more than 4x–than via conventional NICs.



# OSU benchmarks imply off-path mode

For message sizes at 16 KiB or smaller, BF-to-BF communication is **slower** than conventional Infiniband.

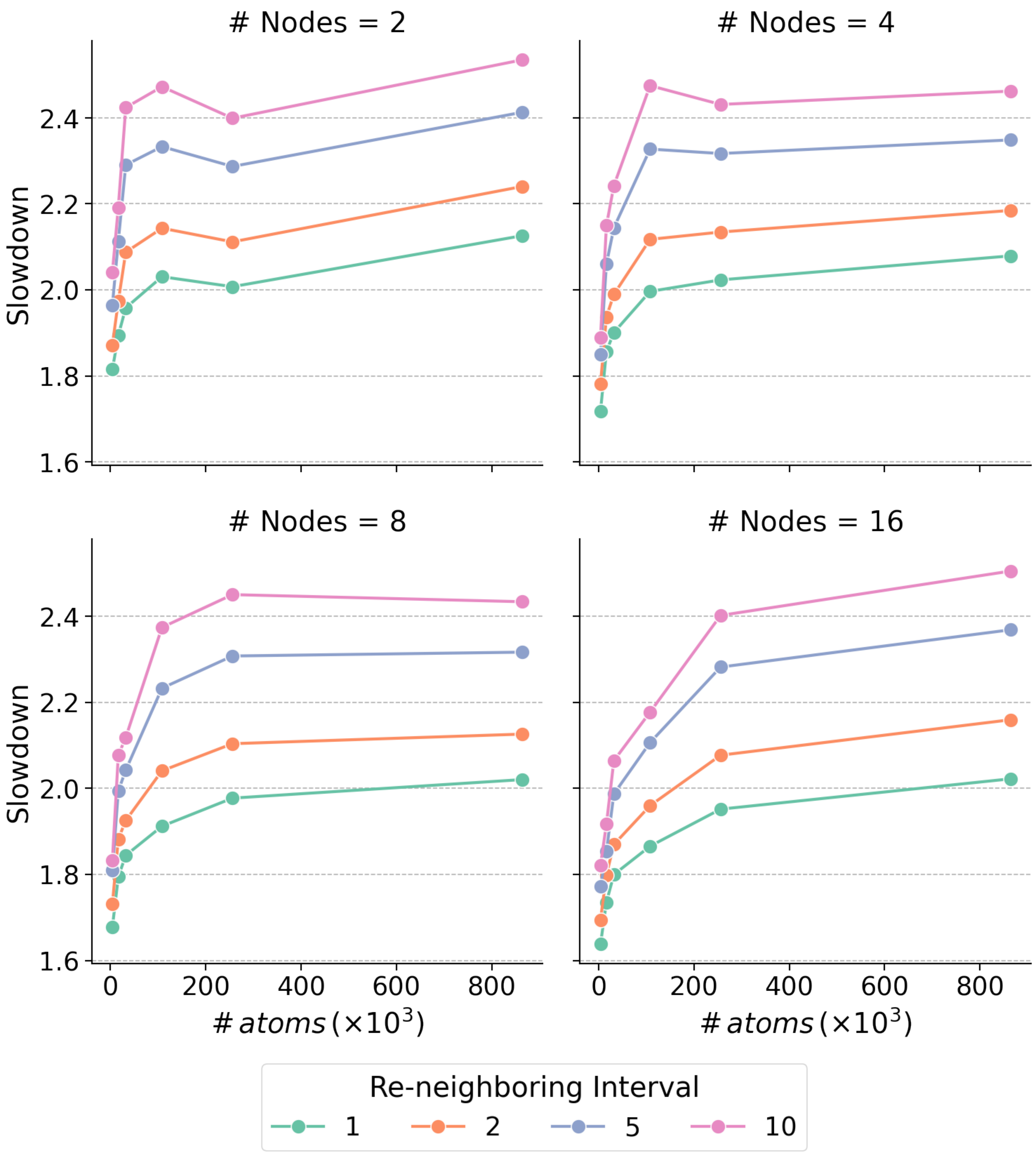


# MiniMD "as-is" does not benefit from BF2

Each BF2 is a "mini-host." Therefore, consider an experiment in which we run MiniMD using only the BF2 cards (i.e., no node-host processing).

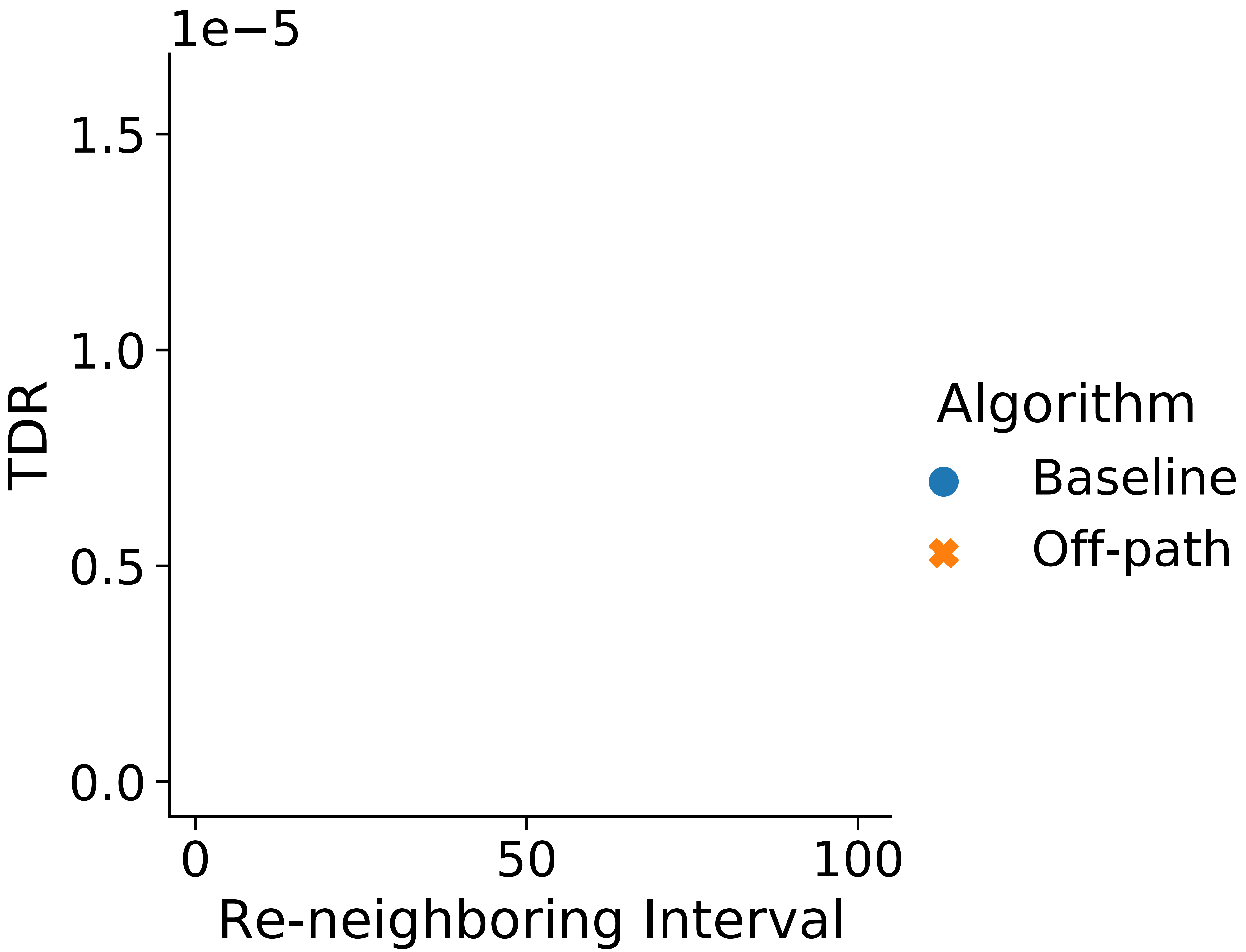
Because of its slower cores and worse communication properties, **MiniMD is always slower than running it without BF2**, at all problem sizes (x-axis), any re-neighboring interval (lines), and any node configuration (subplot).

**This type of result is characteristic** of the PENNANT study of Williams et al. mentioned previously: without any algorithmic or code restructuring, we should not expect any benefits.



# Restructured method is a viable simulation heuristic

Temperature divergence rate (**TDR**): a proxy metric to assess the accuracy of our algorithm



# Restructured method is a viable simulation heuristic

Temperature divergence rate (**TDR**): a proxy metric to assess the accuracy of our algorithm

We also verified that the computed results of the restructured method are still within an acceptable level of accuracy.