Tut 149s1 @ SC2017

Application Porting & Optimization on GPU-accelerated POWER Architectures

Best practices for porting scientific applications

Christoph Hagleitner, hle@zurich.ibm.com

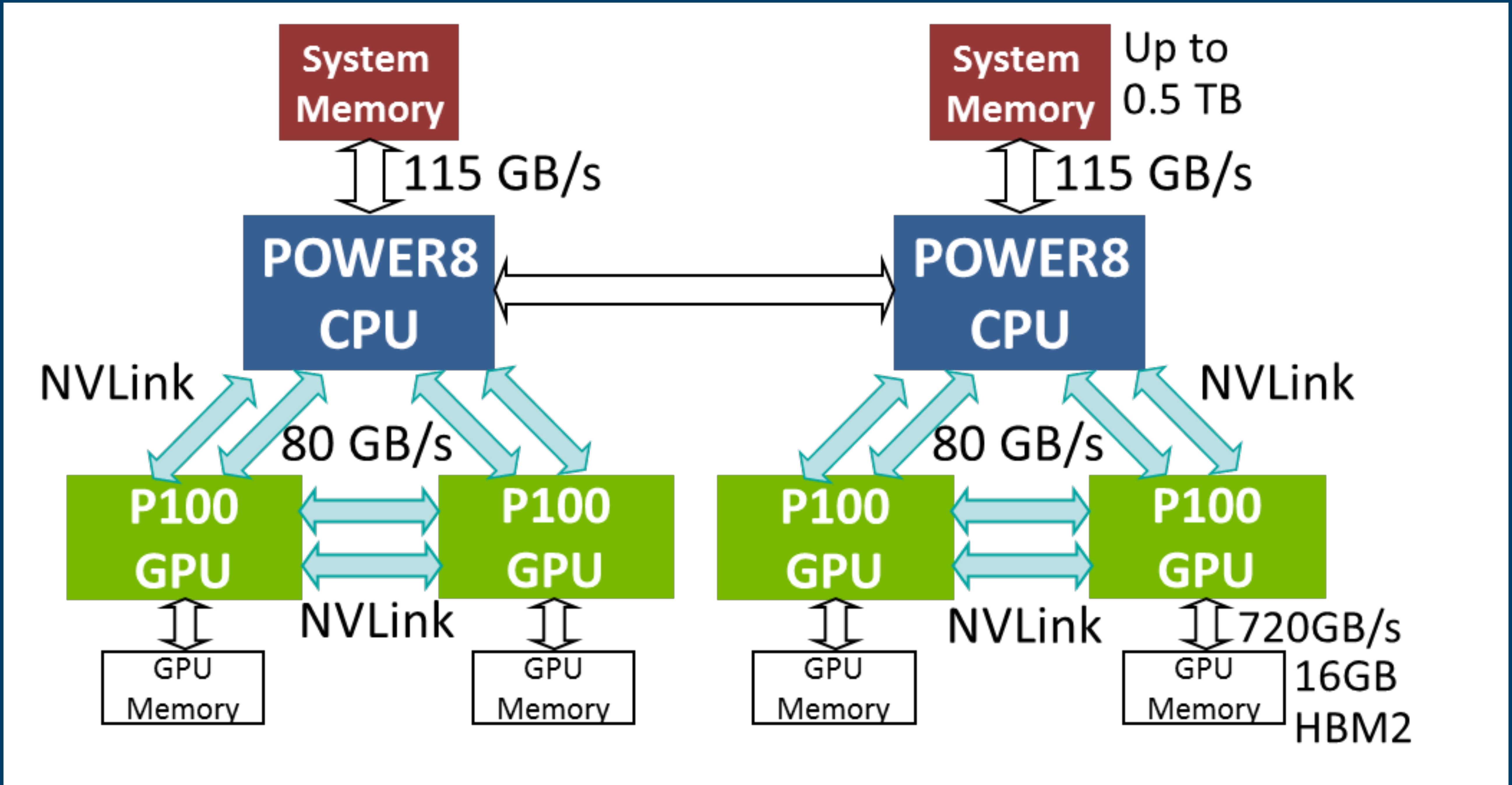
Evaluation

https://submissions.supercomputing.org/eval.html

Agenda

- (open)POWER for HPC: differentiating features
- Porting a complex application: CPMD
- Large-scale AI / Machine Learning
- Dense Storage
- Conclusion

S822LC: IBM POWER8+ for HPC

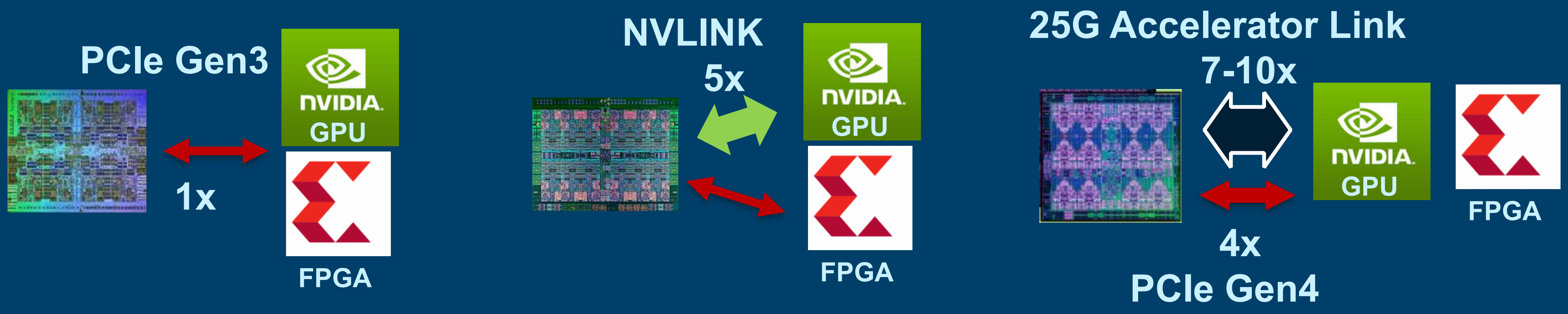


OpenPOWER Core Technology Roadmap

IBM

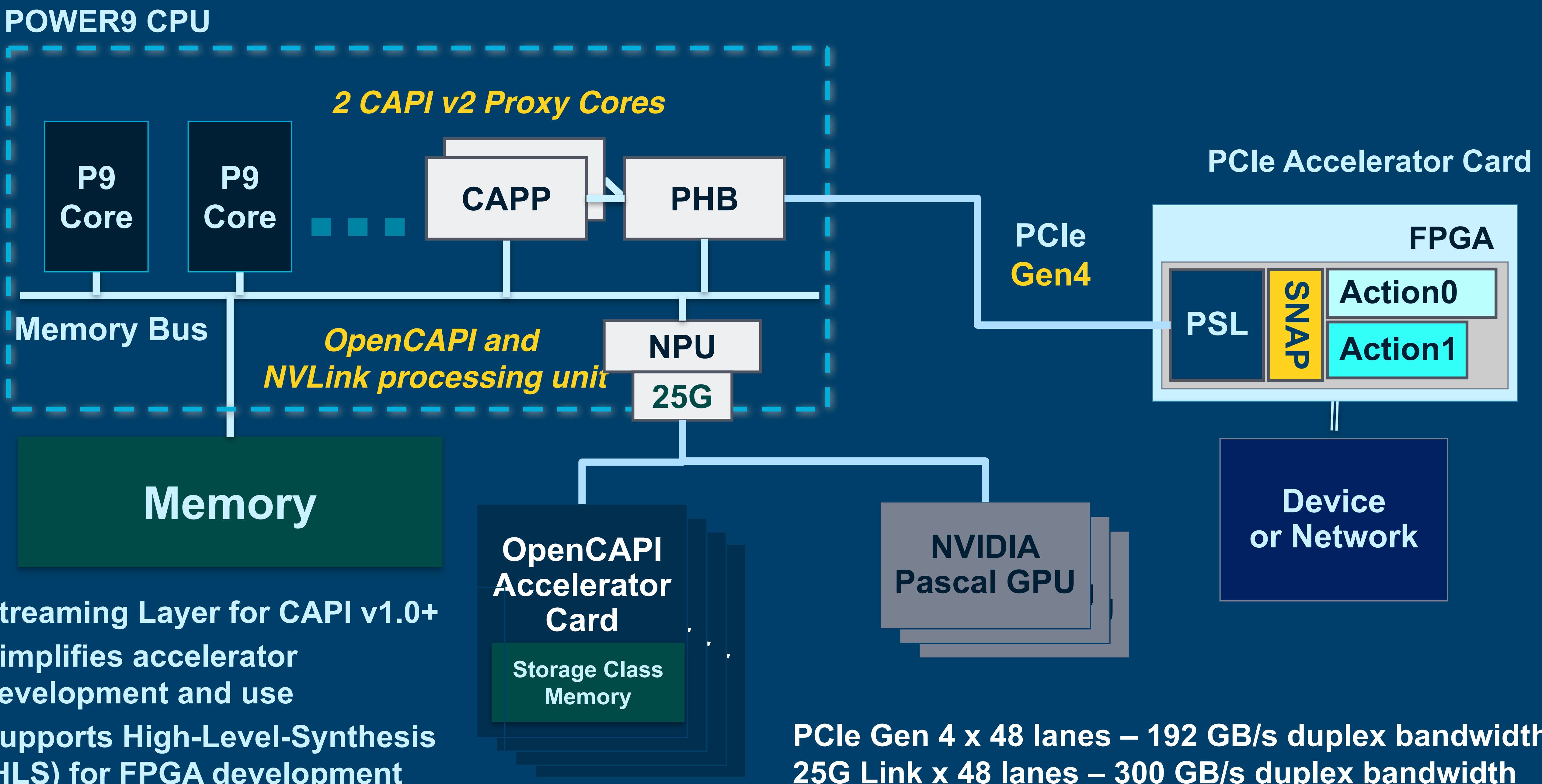
	2015	2016	2017
Mellanox Interconnect	Connect-IB FDR Infiniband PCIe Gen3	ConnectX-4 EDR Infiniband CAPI over PCIe Gen3	ConnectX-6 HDR Infiniband Enhanced CAPI over PCIe Gen4
NVIDIA GPUs	Kepler PCIe Gen3	Pascal NVLink	Volta Enhanced NVLink
IBM CPUs	**POWER8** PCIe Gen3 & CAPI Interface	**POWER8’** NVLink & CAPI	**POWER9** Enhanced NVLink, OpenCAPI & PCIe Gen4

Accelerator Links



2015 2016 2017

OpenCAPI v3.0 and NVLINK 2.0 with POWER9



CAPI SNAP

- Streaming Layer for CAPI v1.0+
- Simplifies accelerator development and use
- Supports High-Level-Synthesis (HLS) for FPGA development
- Available as open source

PCIe Gen 4 x 48 lanes – 192 GB/s duplex bandwidth
25G Link x 48 lanes – 300 GB/s duplex bandwidth

Looking Ahead: POWER9 Accelerator Interfaces

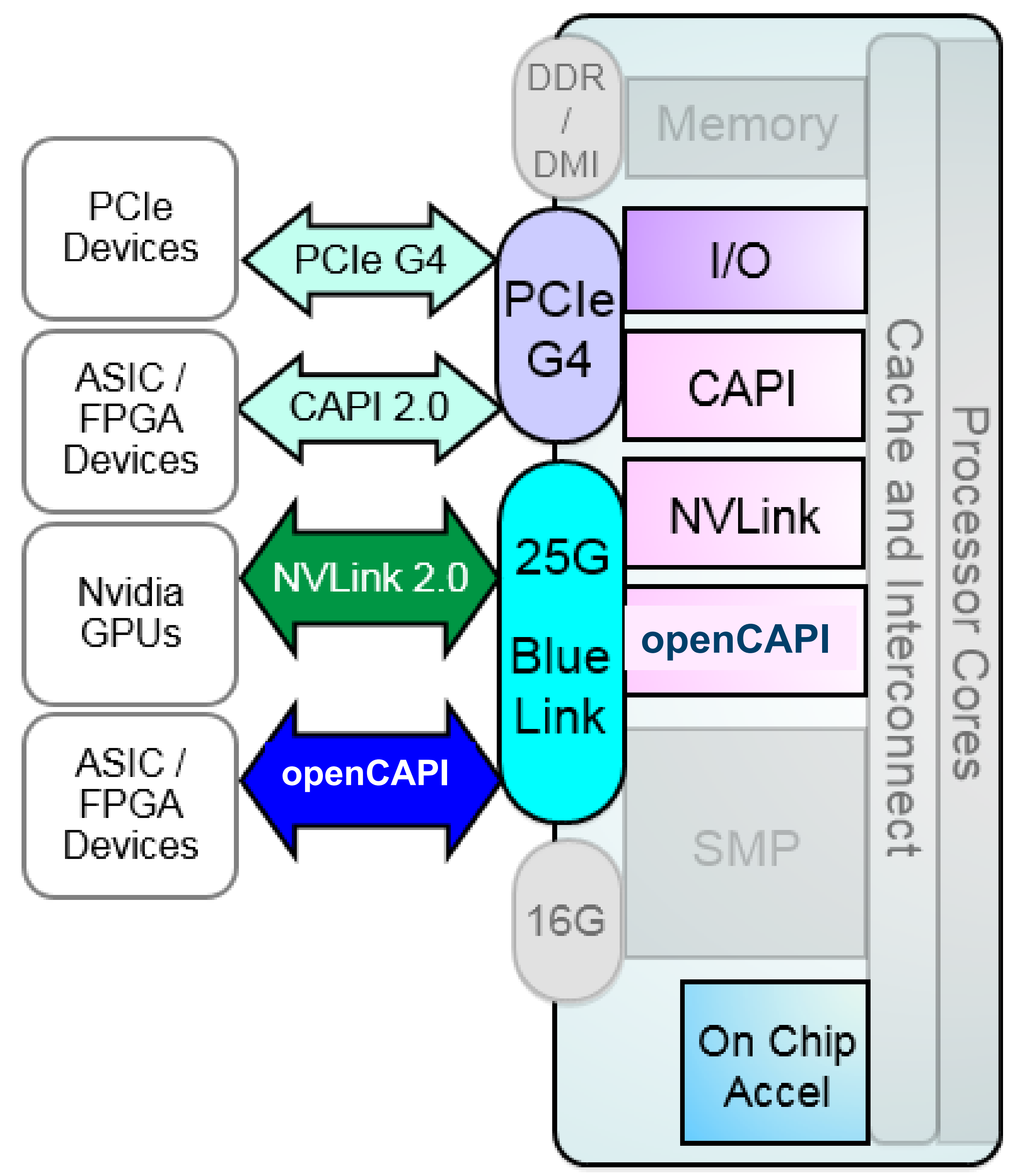
IBM

Extreme Accelerator Bandwidth and Reduced Latency

- PCIe Gen 4 x 48 lanes – 192 GB/s peak bandwidth (duplex)
- IBM BlueLink 25Gb/s x 48 lanes – 300 GB/s peak bandwidth (duplex)

Coherent Memory and Virtual Addressing Capability for all Accelerators

- CAPI 2.0 - 4x bandwidth of POWER8 using PCIe Gen 4
- NVLink 2.0 – Next generation of GPU/CPU bandwidth and integration using BlueLink
- OpenCAPI – High bandwidth, low latency and open interface using BlueLink



Porting a Complex HPC Application to POWER + GPUs

- Heterogeneous systems (eg, CPU/GPU) are key to reach exascale

- OpenPOWER systems combining CPUs and GPUs address key issues on the road to scalable acceleration
  - Compute density
  - Data transfer density/BW
  - Coherent memory space

- Thus there is a need to port computational science codes to heterogeneous systems. This requires algorithm rethinking and code reengineering in order to fully exploit next generation of heterogeneous architectures.

- Today's showcase: electronic structure code CPMD

OpenPOWER EcoSystem

- POWER-optimized libraries & compilers
  - Advanced toolchain
    https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/IBM%20Advance%20Toolchain%20for%20PowerLinux%20Documentation
  - XL-compilers
    https://www.ibm.com/developerworks/community/groups/community/xlpower/
  - ESSL
    https://www-03.ibm.com/systems/power/software/essl/

- GPU optimization
  - CUDA
  - CUDNN
  - openGL

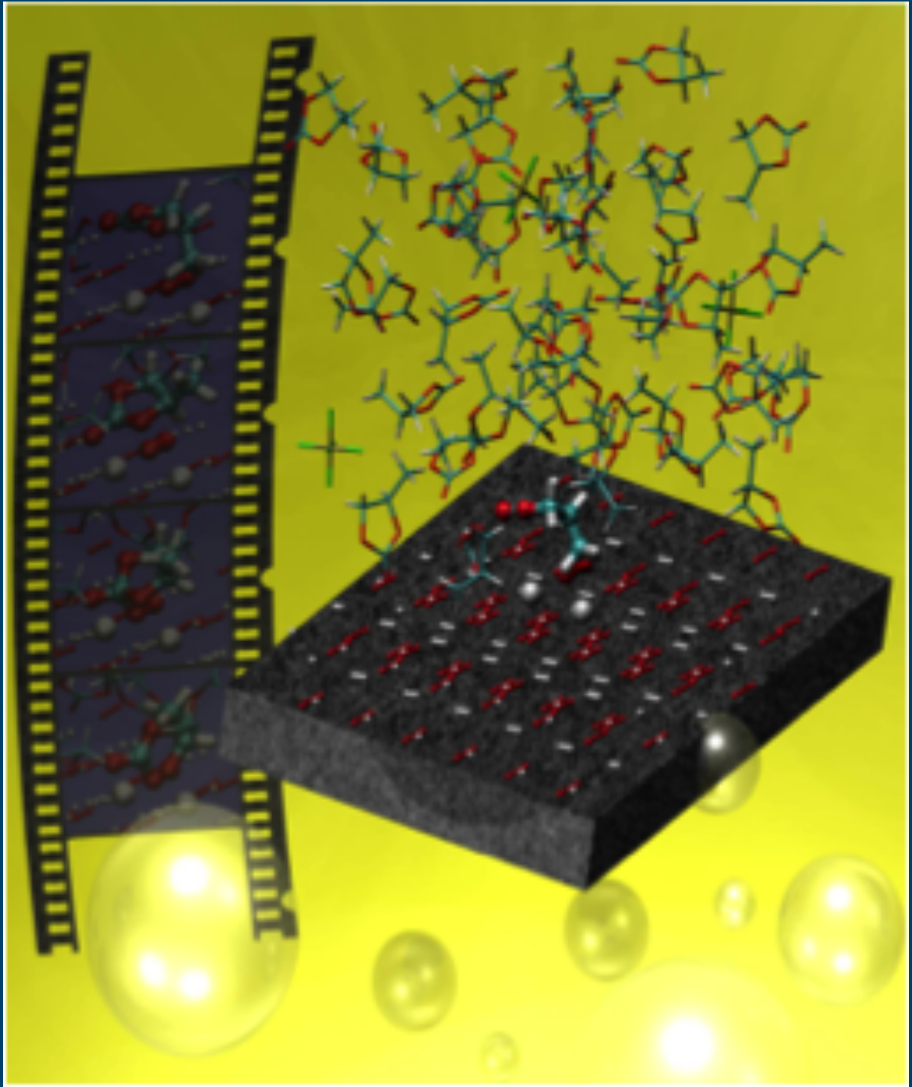
- PowerAI

Agenda

- (open)POWER for HPC: differentiating features
- Porting a complex application: CPMD
- AI / Machine Learning
- Dense Storage
- Conclusion

Car–Parrinello Molecular Dynamics: CPMD

- Shown to scale to very large systems
- Numerous showcases, eg, Li-Air batteries



Simulations of Li_2O_2 in Propylenecarbonate, T. Laino, A. Curioni, A New Piece in the Puzzle of Lithium/Air Batteries, Chemistry, DOI 10.1002/chem.201103057 (22 February 2012)

Introduction: Kohn–Sham equations

Observation:

- Each iteration step require at least N x 3D FFT (inverse/forward)

We focused on:

- Construction of the electronic density

$$\rho(\mathbf{r}) = \sum_i^N |\phi_i(\mathbf{r})|^2$$

- Applying the potential to the wavefunctions

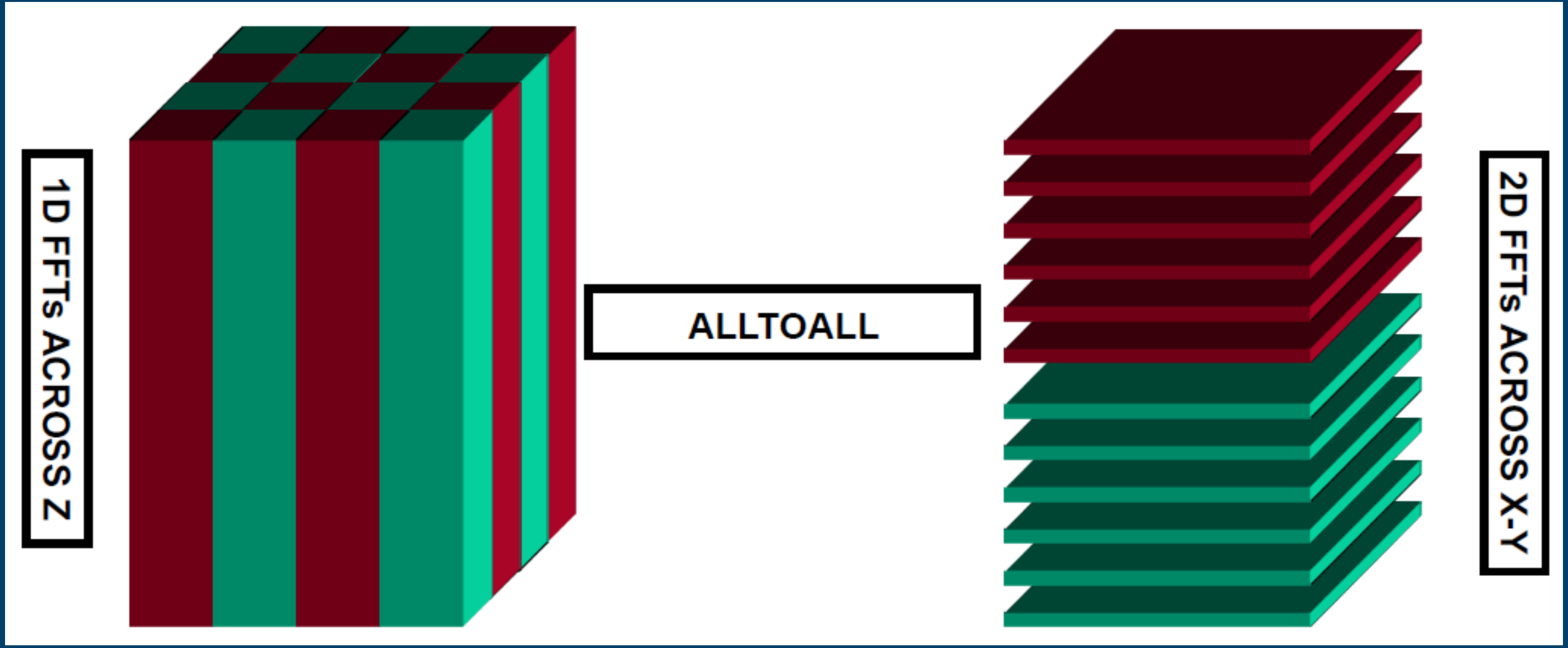
$$\left[-\frac{1}{2}\nabla_i^2 + V_{\text{eff}}[\rho]\right]\phi_i(\mathbf{r}) = \epsilon_i\phi_i(\mathbf{r}),$$

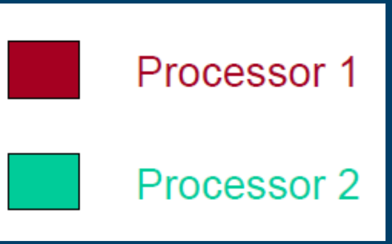
- Orthogonalization of the wavefunctions

$$\int \phi_i(\mathbf{r})\phi_j(\mathbf{r})d^3r = \delta_{ij}$$

Parallelization: Distributed Memory and 3D FFT

$$\tilde{\phi}_i(\mathbf{G}) \underset{\text{FFT}}{\overset{\text{invFFT}}{\Longleftrightarrow}} \phi_i(\mathbf{r})$$





Limited Scalability in Standard 3D FFT

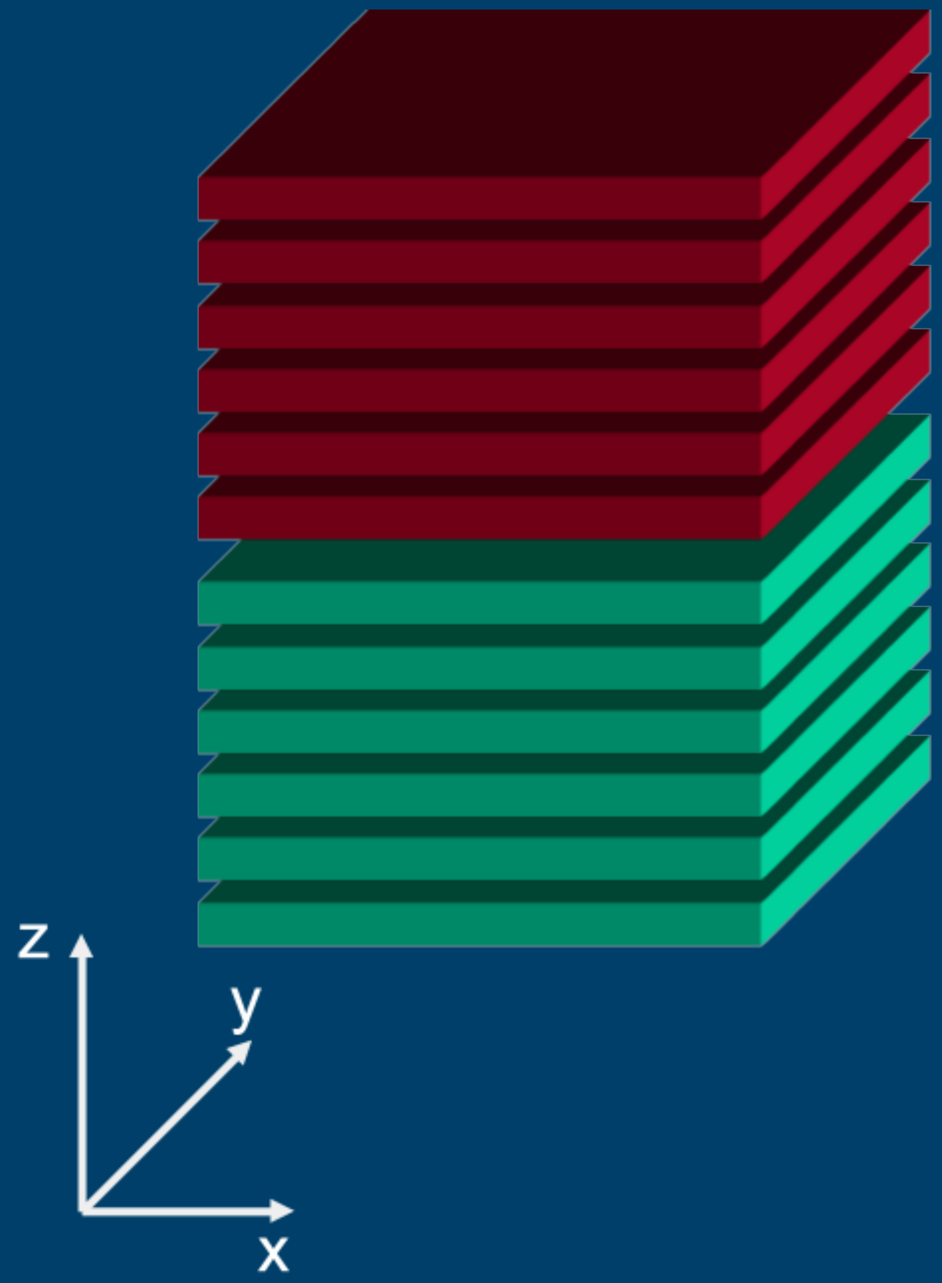
Each processor takes a number of whole planes …

… very good scheme for small – medium sized computational platforms

… but observe that scalability is limited by the number of planes across the Z-direction!

... which is in the order of a few hundred

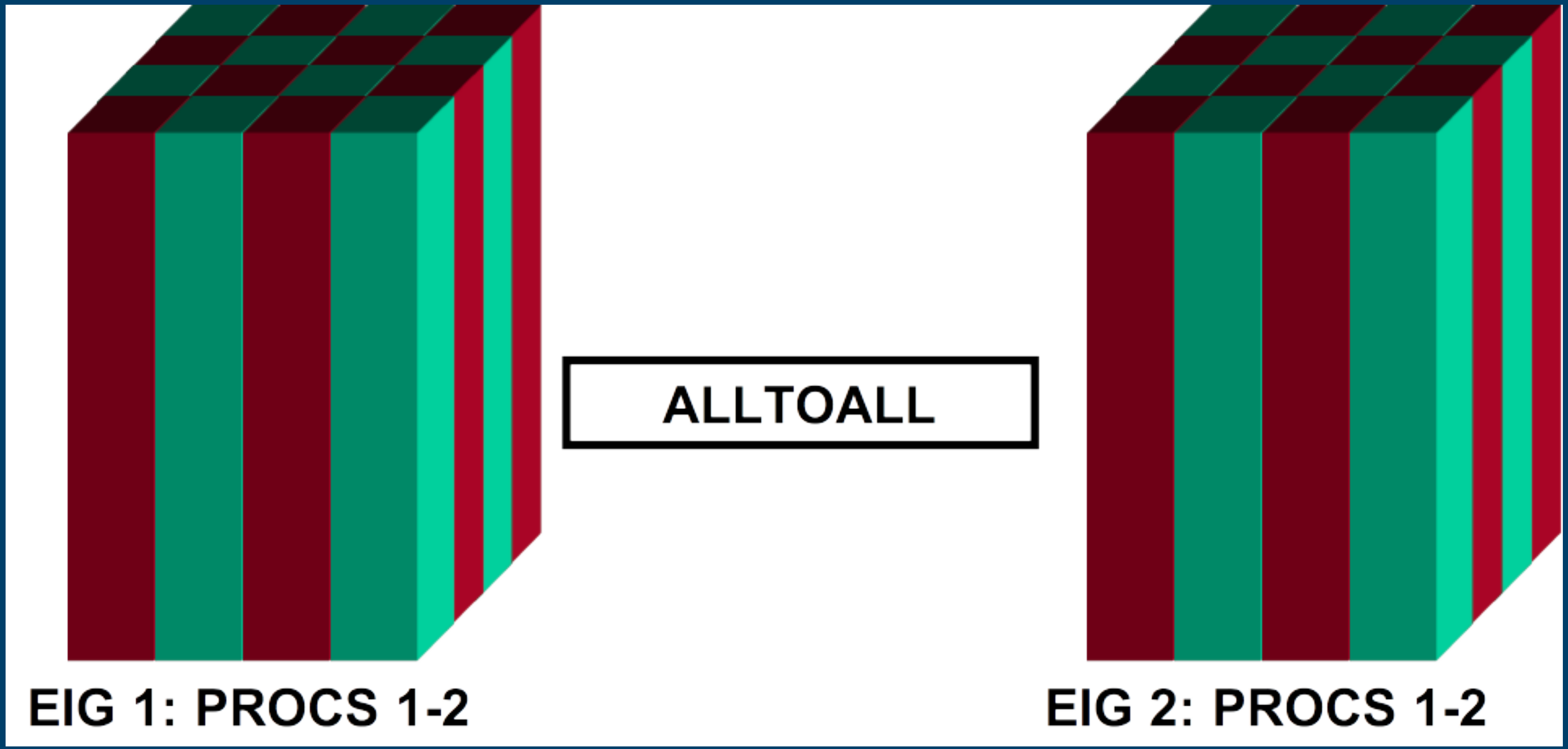
Thus: not appropriate for a massively parallel system



3D FFTs Using Task Groups

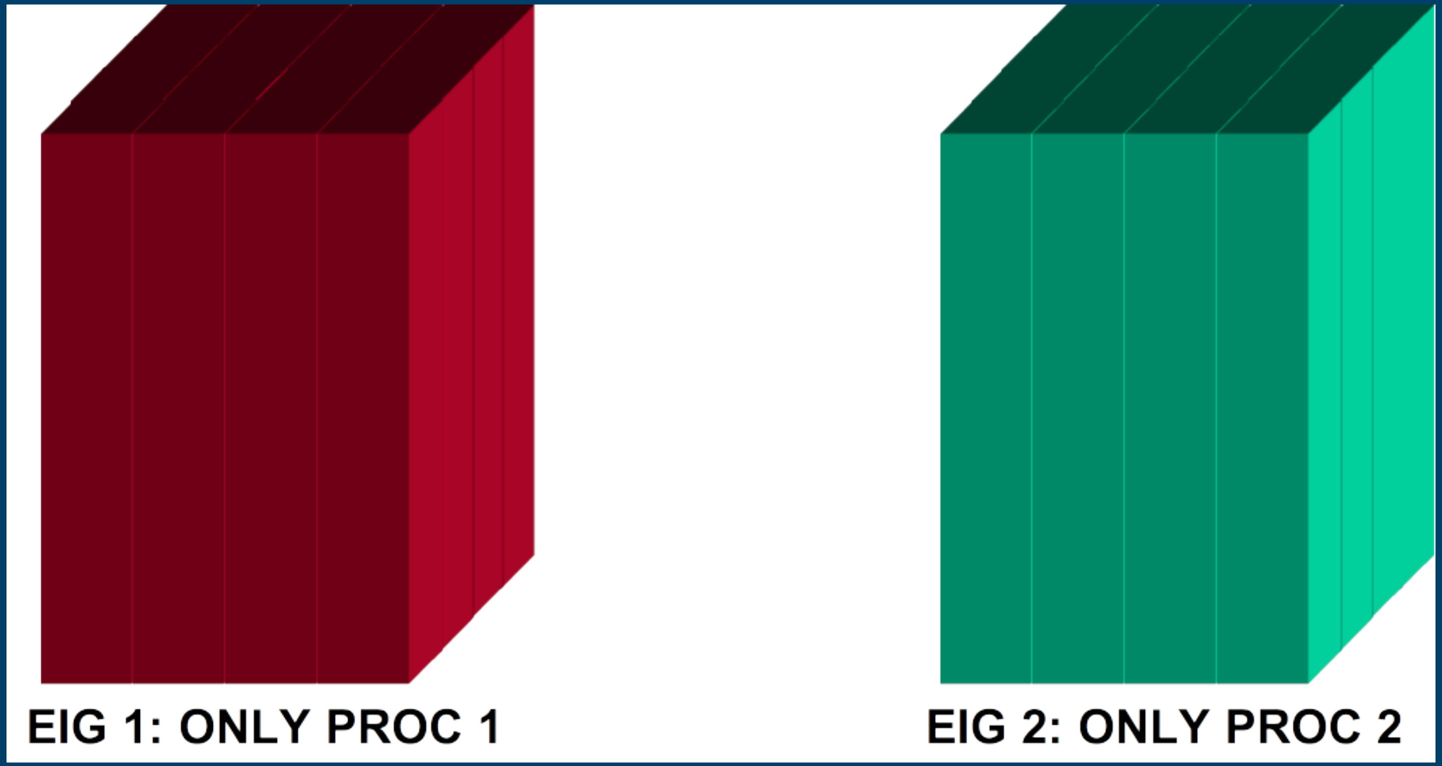
$$\rho(r) = \sum_{occ} |\psi_i(r)|^2$$

- Loop across the number of electrons. Each iteration requires 1 3D FFT.
- Hierarchical parallelism*: Assign to each Task Group a number of iterations



3D FFTs Using Task Groups

- task groups of processors will work on different eigenstates concurrently
- number of processors per group: Ideally the one that achieves the best scalability for the original parallel 3D FFT scheme

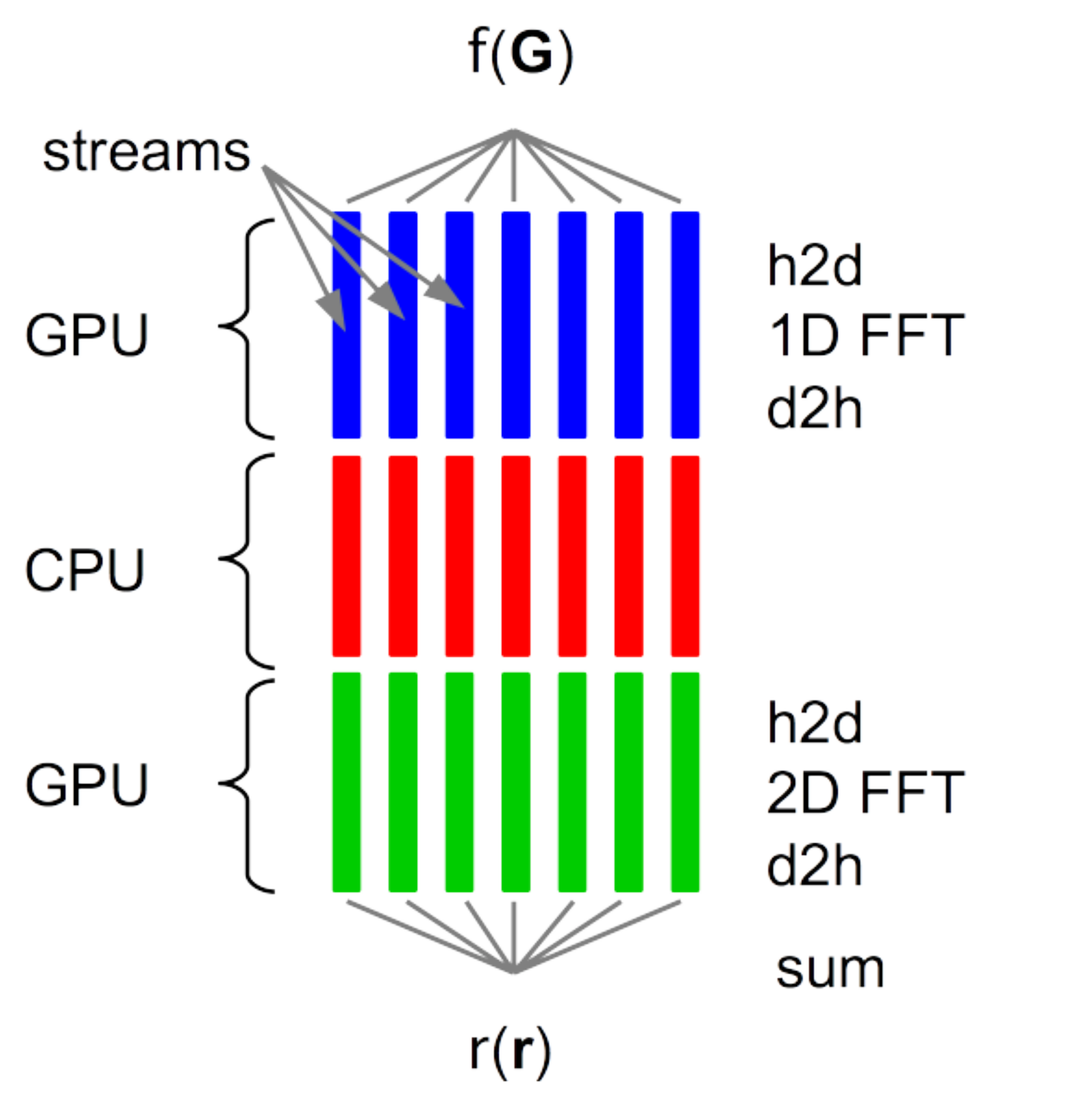


GPU Porting: Construction of the electronic density

$$\phi_i(\mathbf{r}) = \mathrm{invFFT}(\tilde{\phi}_i(\mathbf{G}))$$

$$\rho(\mathbf{r}) = \sum_{i}^{N} |\phi_i(\mathbf{r})|^2$$

- The reverse Fourier transform of the N states φ(G) is distributed over the NS streams that work concurrently.
- Each stream is assigned to a CPU thread.
- Each stream transforms a state φ(G) to the corresponding density (1D FFT – all2all – 2D FFT)



GPU Porting: Applying the potential to the wavefunctions

- The reverse and forward Fourier transforms as well as the application of the potential V to the N states are distributed over NS streams that work concurrently.
- Each stream is assigned to a CPU thread.
- Each stream transforms a state φ(G) to φ(r) (1D FFT – all2all – 2D FFT). The potential is applied and the result back transformed (2D FFT – all2all – 1D FFT).

$$\phi_i(\mathbf{r}) = \mathrm{invFFT}(\tilde{\phi}_i(\mathbf{G}))$$

$$V(\mathbf{r})\phi_i(\mathbf{r})$$

$$(\widetilde{V\phi_i})(\mathbf{G}) = \mathrm{FFT}((V\phi_i)(\mathbf{r}))$$

GPU Porting: Orthogonalization via block Gram-Schmidt

- we seek the orthogonalized coefficient matrix

$$\tilde{C} = \mathrm{ortho}(C)$$

- the coefficients of the expansion of φ(G) on the plane-wave basis is block-partitioned column-wise into n blocks of size b.
- the block Gram–Schmidt scheme loops over the n blocks Ci and orthogonalizes them one after the other

$$C = [C_1, C_2, \ldots, C_n]$$

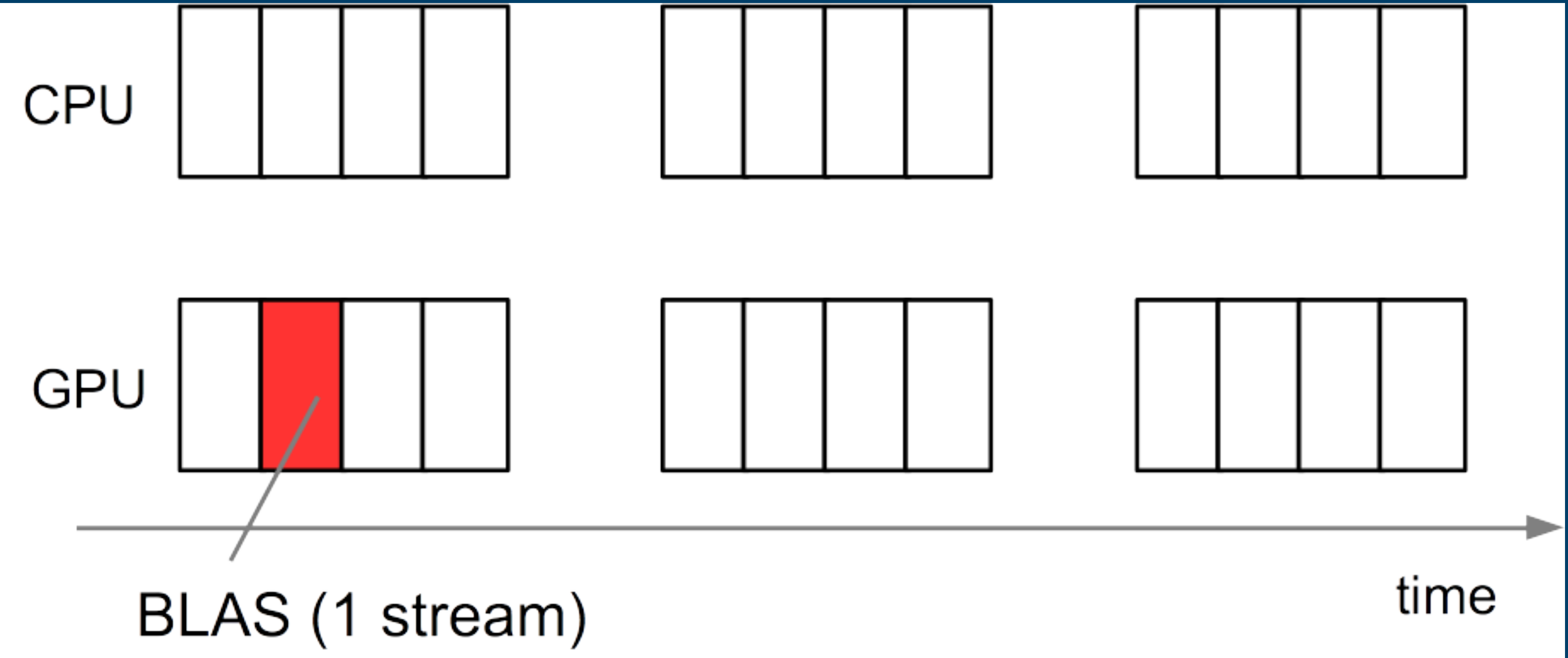
$$[\tilde{C}_1, \ldots, \tilde{C}_{i-1}, C_i, \ldots, C_n]$$

$$\tilde{C}_i = \mathrm{ortho}((I - \textstyle\sum_{j=1}^{i-1} \tilde{C}_j \tilde{C}_j^T) C_i)$$

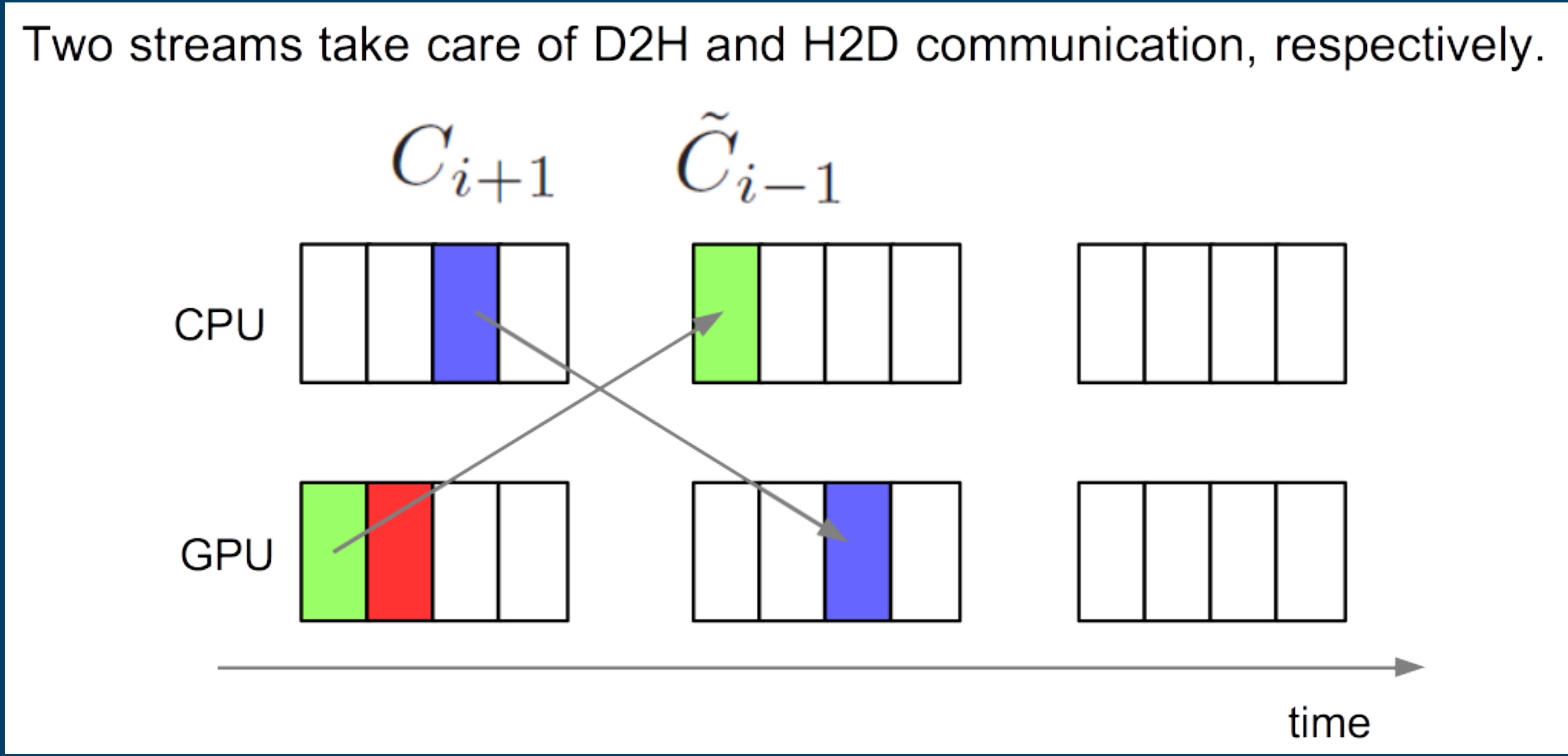
GPU Porting: Orthogonalization via block Gram-Schmidt

$$[\tilde{C}_1, \ldots, \tilde{C}_{i-1}, C_i, \ldots, C_n]$$

$$\tilde{C}_i = \mathrm{ortho}((I - \textstyle\sum_{j=1}^{i-1} \tilde{C}_j \tilde{C}_j^T) C_i)$$

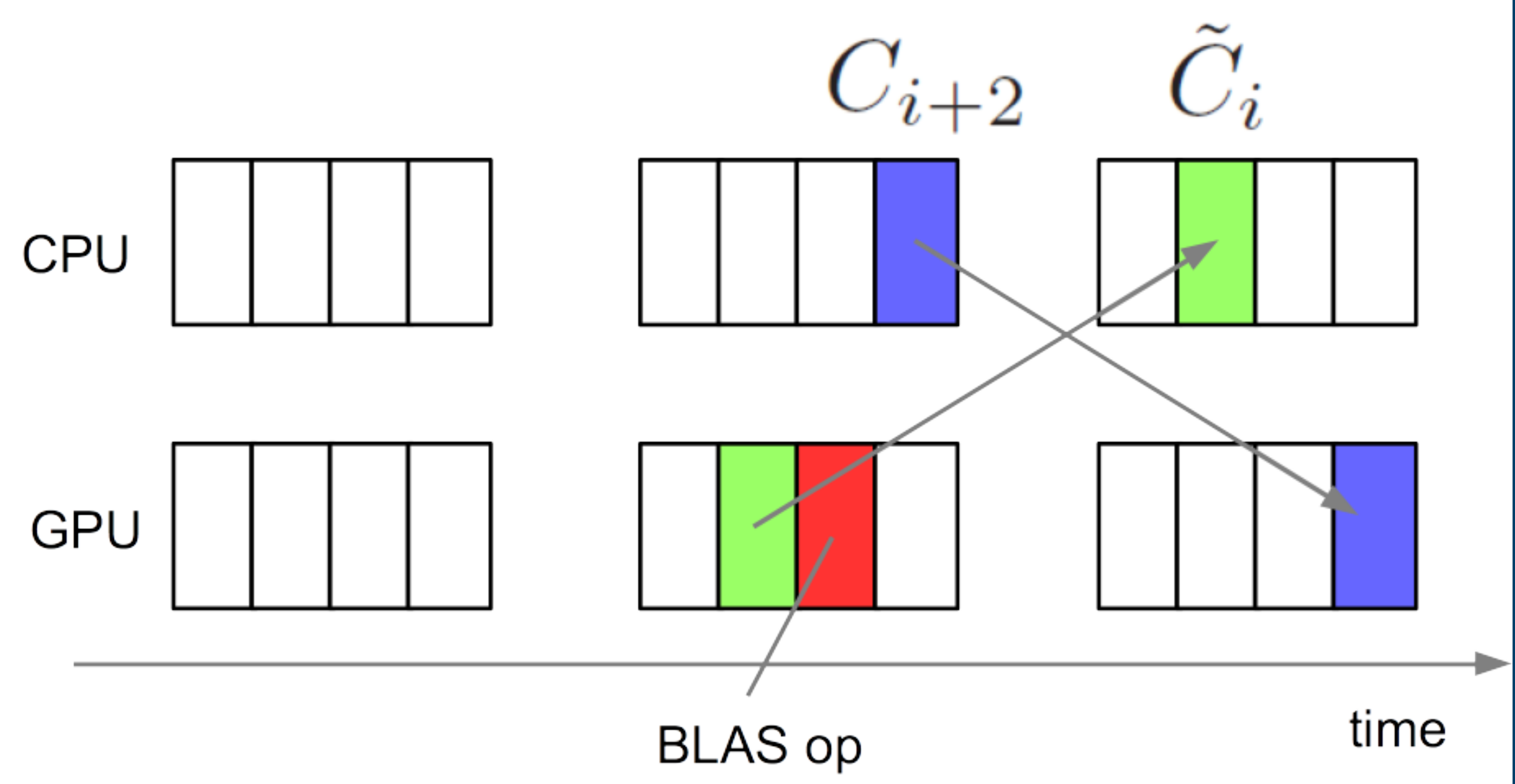


GPU Porting: Orthogonalization via block Gram-Schmidt



GPU Porting: Orthogonalization via block Gram-Schmidt

$$\tilde{C}_{i+1} = \mathrm{ortho}((I - \textstyle\sum_{j=1}^{i} \tilde{C}_j \tilde{C}_j^T)C_{i+1})$$



Data Transfer (Initialization)

- 2 socket Power8 (Minsky and Firestone)
  - PCIe attached K80 GPU on Firestone
  - NVlink attached P100 GPU on Minsky
- Data transfer volume ~100 GB (HtoD & DtoH)
- Overall results
  - Data transfer overhead too high on Firestone (slower than CPU)
  - Crossover on Minsky (faster than CPU)

	Firestone	Minsky.
	Total Time (seconds)	Total Time (Seconds)
Host to Device	15.801	6.862
Device to Host	12.891	6.638

Benchmark w/ Minsky

- Minsky (20-cores / P100)
- 128 water box, 100Ry, GTH
- 20 MPI / 1 Threads

Stream	CPU	1 GPU	2 GPU	4 GPU
	203			
1	-	211 S	154 S	137
vpsi	91.57	87.68	54.24	46.61
fwfftn	46.08	54.53	31.28	24.69
invfftn	80.27	81.34	47.01	34.68
rhoofr	48.03	55.34	32.96	26.07
ortho	12.59	5.37	4.70	4.62

Agenda

- (open)POWER for HPC: differentiating features
- Porting a complex application: CPMD
- AI / Machine Learning
- Dense Storage
- Conclusion

Deep Learning on openPOWER: PowerAI

Package of Pre-Compiled Major Deep Learning Frameworks

Easy to install & get started with Deep Learning with Enterprise-Class Support

Optimized for Performance To Take Advantage of NVLink

Enabled by High Performance Computing Infrastructure

PowerAI Deep Learning Software Distribution

IBM

Deep Learning Frameworks

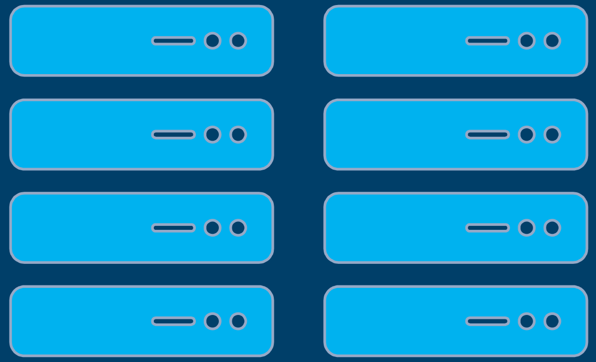
- Caffe
- NVCaffe
- IBMCaffe
- Torch
- TensorFlow
- Distributed TensorFlow
- Theano
- Chainer

Supporting Libraries

- OpenBLAS
- Bazel
- Distributed Communications
- NCCL
- DIGITS

Accelerated Servers and Infrastructure for Scaling

Cluster of NVLink Servers



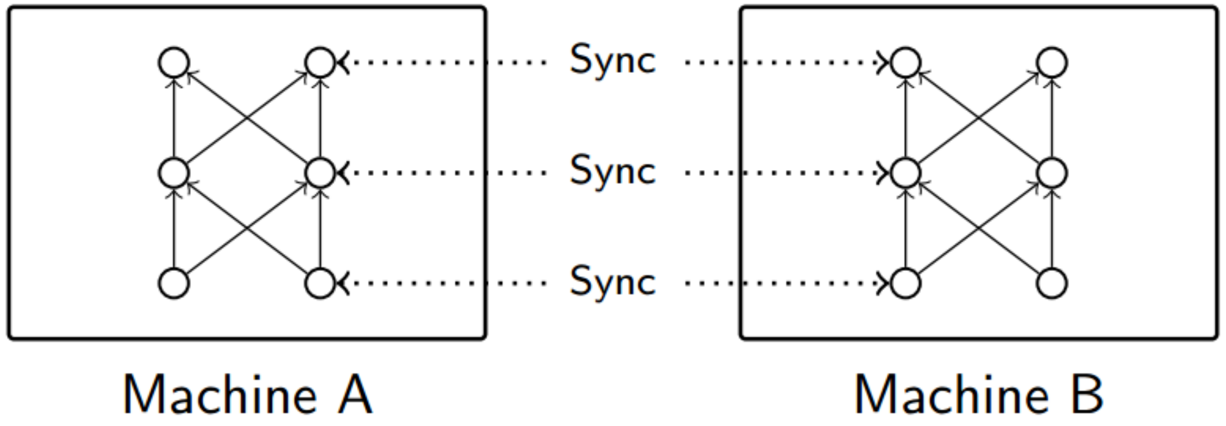
Spectrum Scale: High-Speed Parallel File System

Scale to Cloud

Distributed Deep Learning

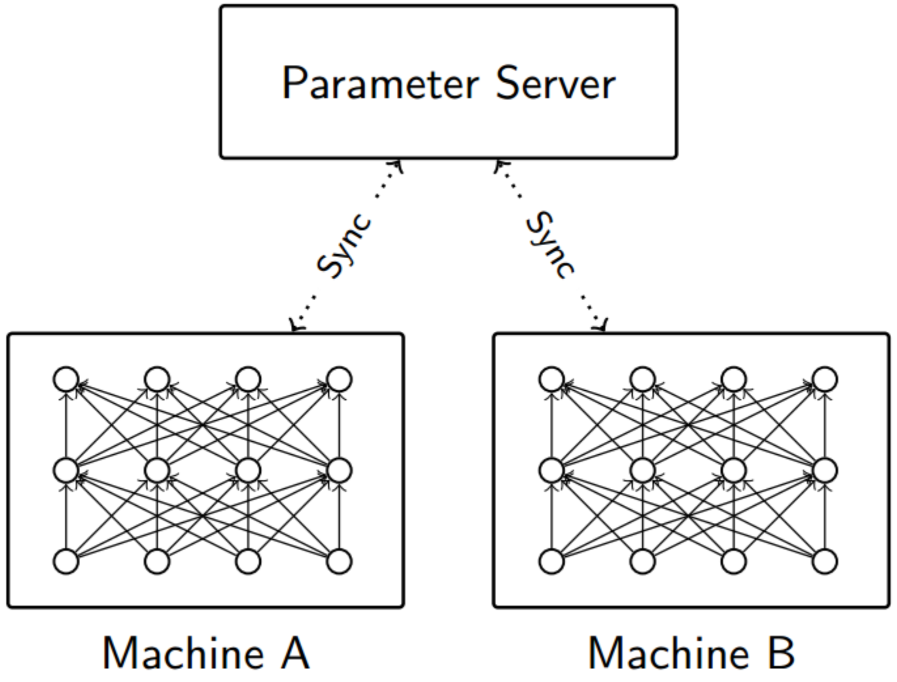
[P. Goldsborough]

Anything

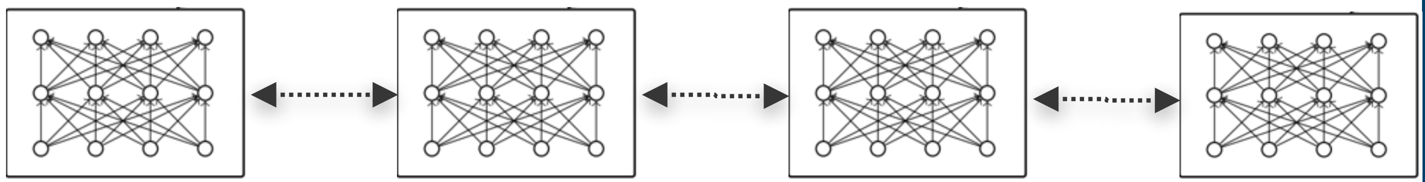


Model parallelism (complex partitioning)

Gradient/weight (10MB-1GB)

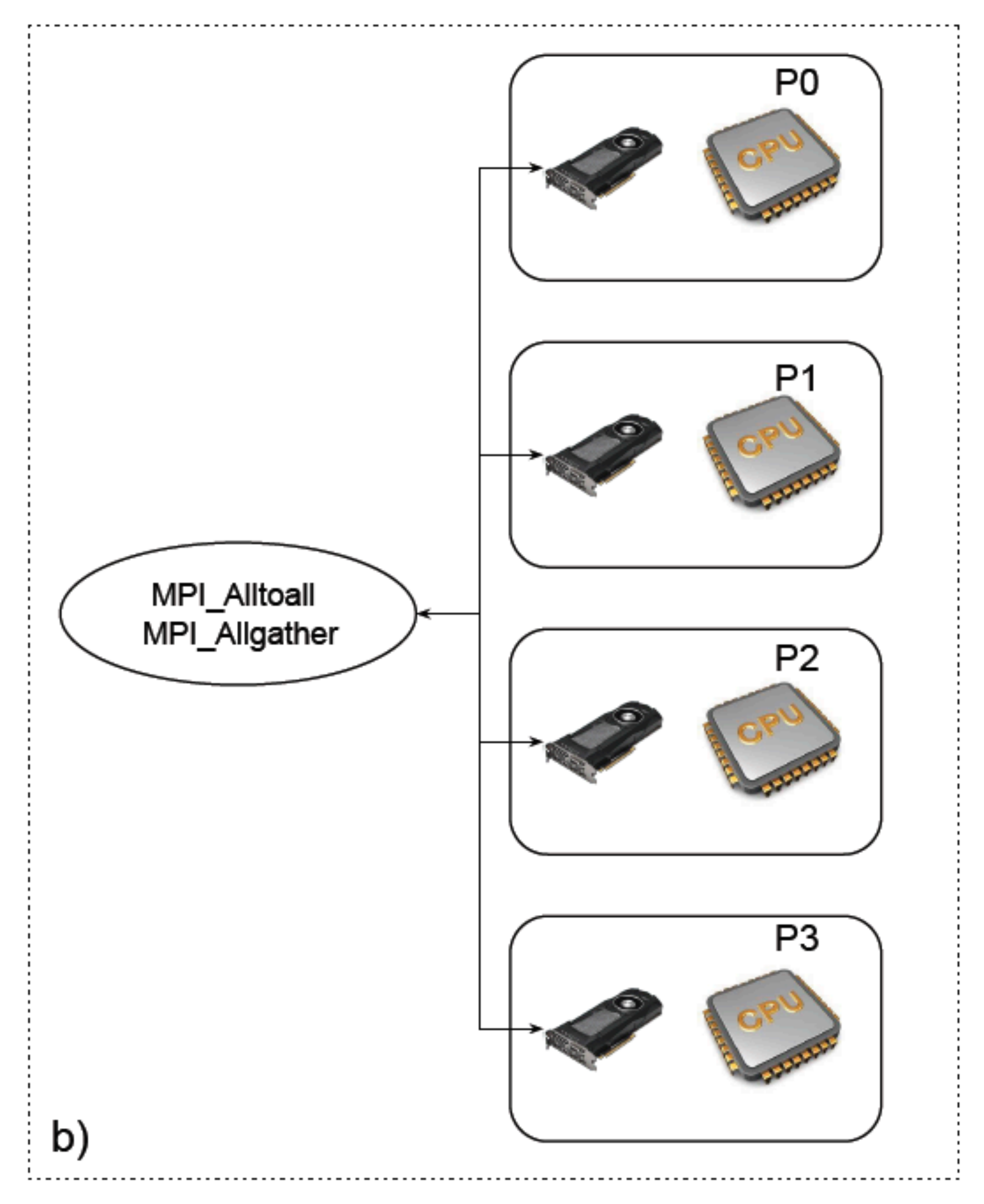


Data parallelism : Parm-Server



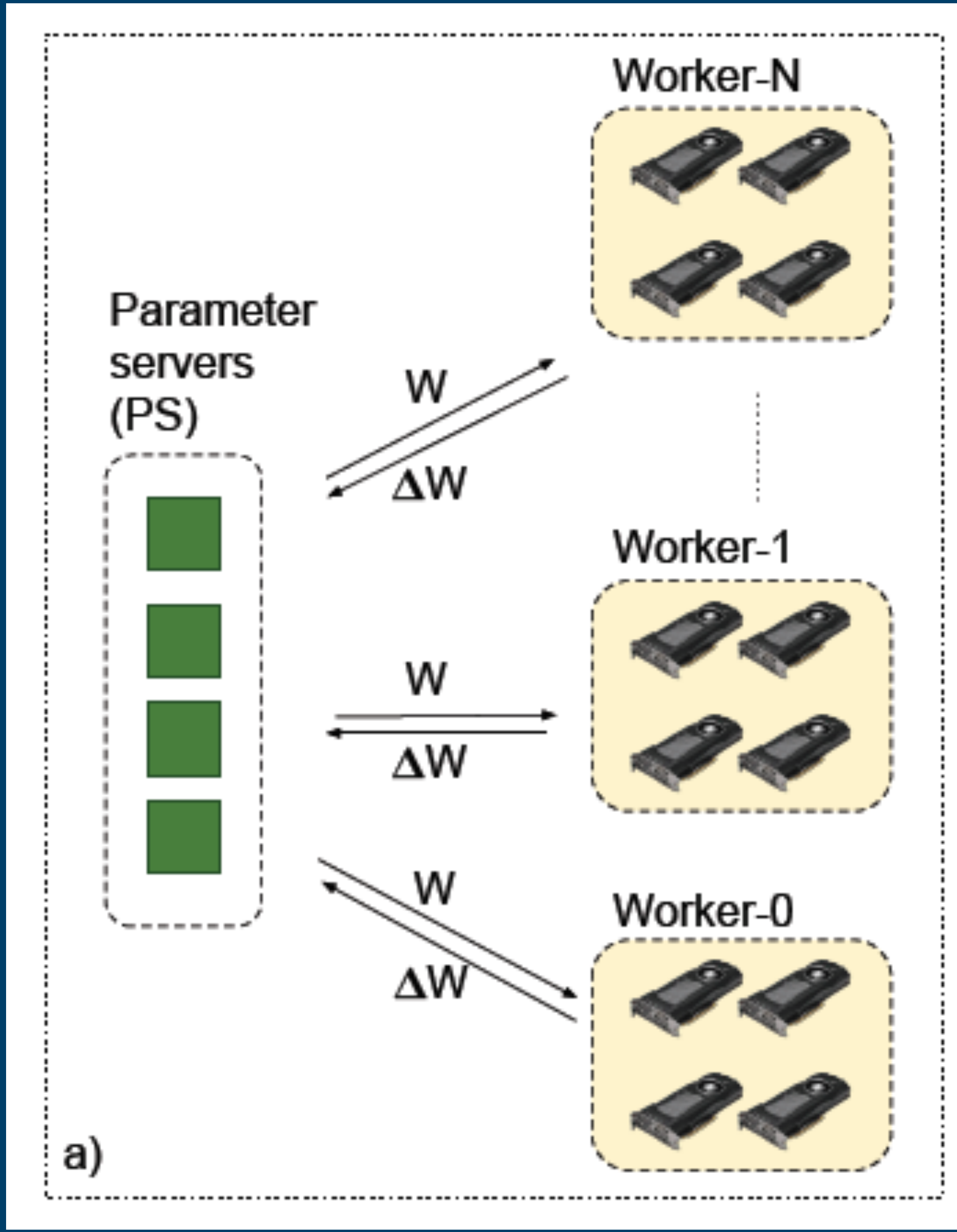
Data parallelism : Allreduce

Multinode Scaling: Theano-MPI



- multi-node, multi-GPU training
- relies on CUDA-aware MPI (fast inter-GPU memory exchanges)
- Half precision parameter transfers support to further reduce parameters overhead, while summing them at full precision

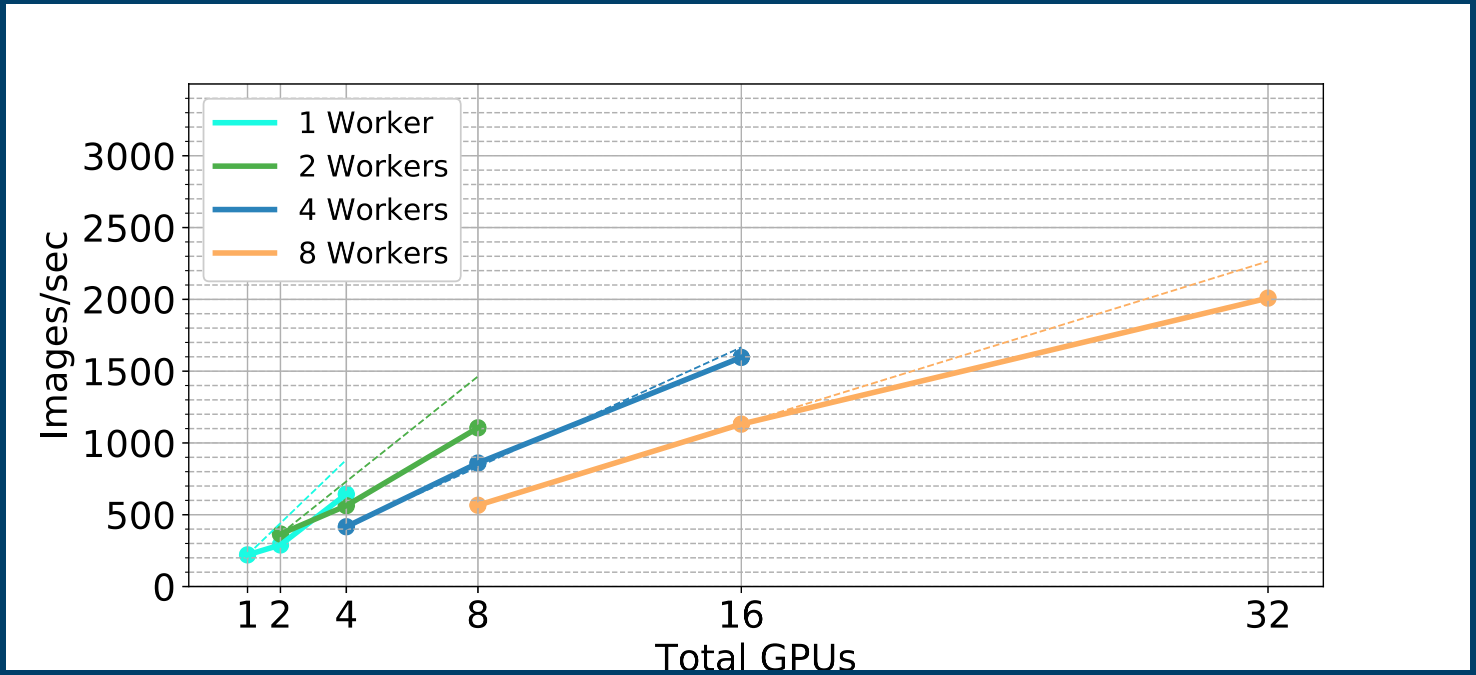
Multinode Scaling: Tensorflow

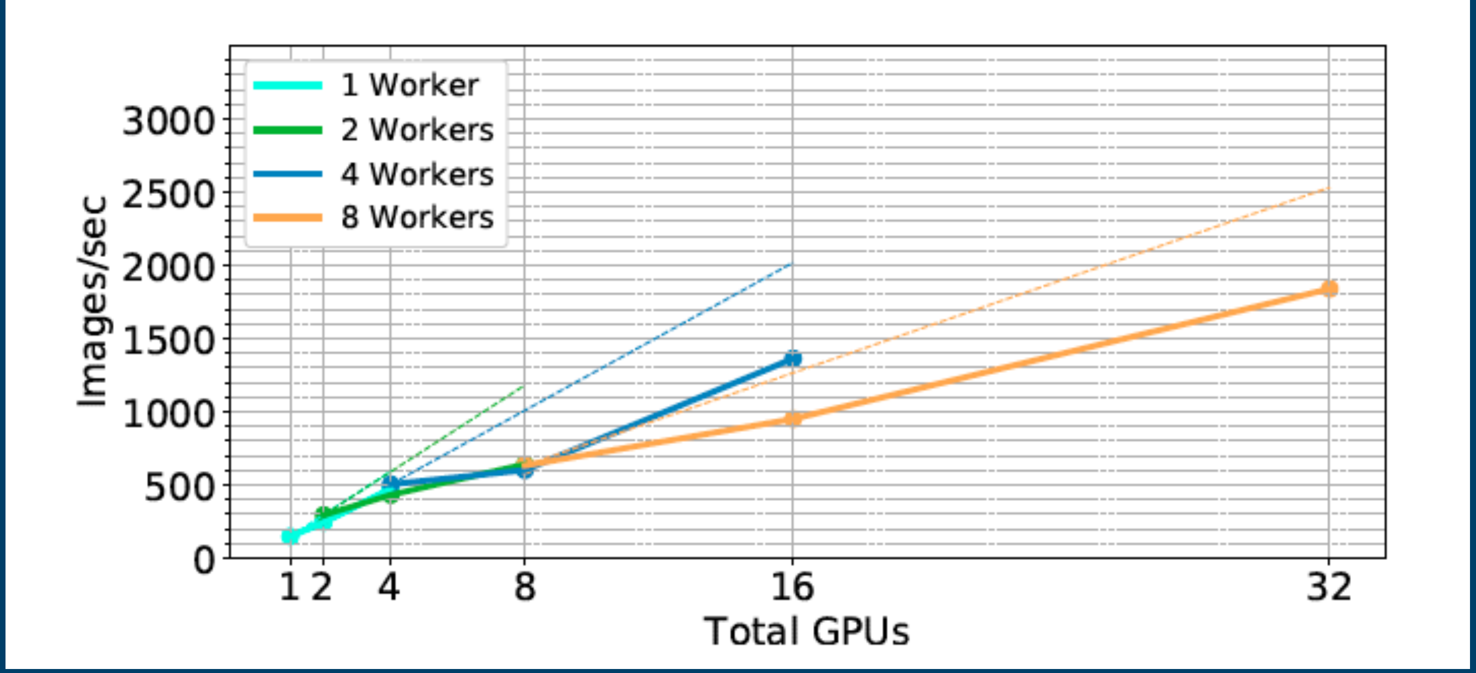


- two types of jobs: ps & worker
- each worker owns a network replica and performs inference and backpropagation
- each ps stores part of the graph, updates local parameters, and sends updated parameters to workers
- used TensorFlow 1.2.1 for experiments

Tensorflow Multinode Performance for ResNet and Inception

- Multi node performance
  - ResNet-50 (upper)
  - InceptionV3 (lower)
- trained with different cluster configurations.
- Every line connects three experiments.
- All three experiments are run with the same number of workers (1, 2, 4 or 8), each worker having 1, 2 or 4 GPUs.
- In multi-worker experiments, each worker is hosted on a different node.
- Dashed lines show ideal scalability.

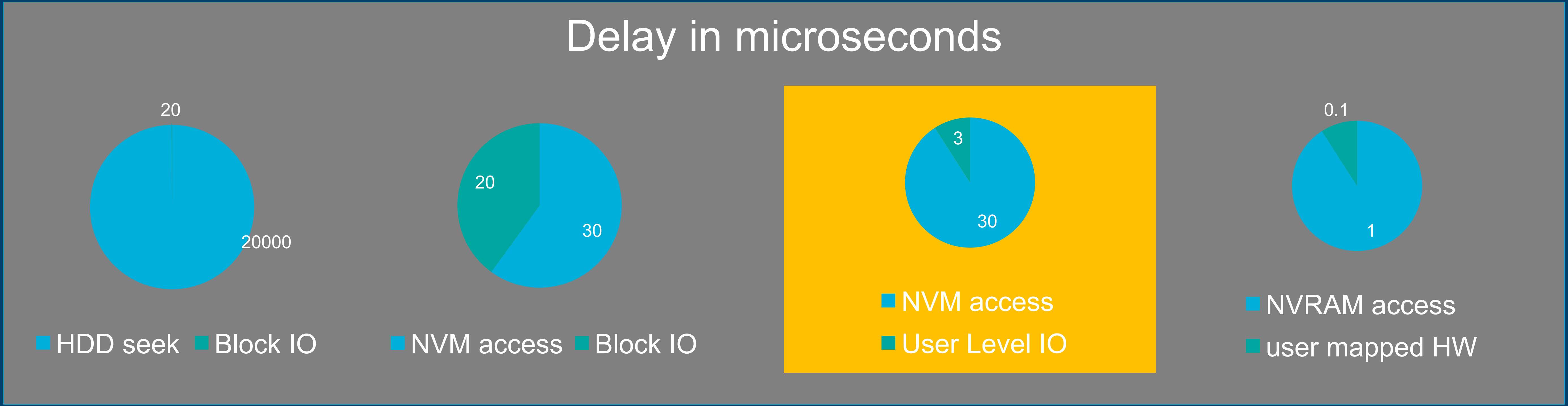




Agenda

- (open)POWER for HPC: differentiating features
- Porting a complex application: CPMD
- AI / Machine Learning
- Dense Storage
- Conclusion

Storage



- HD: Block IO stack efficient
- NVM: Block IO stack becomes bottleneck
- NVRAM etc: User mapped hardware, lib integration

+ …byte granular access

DSS: High Performance Distributed Shared Storage Stack

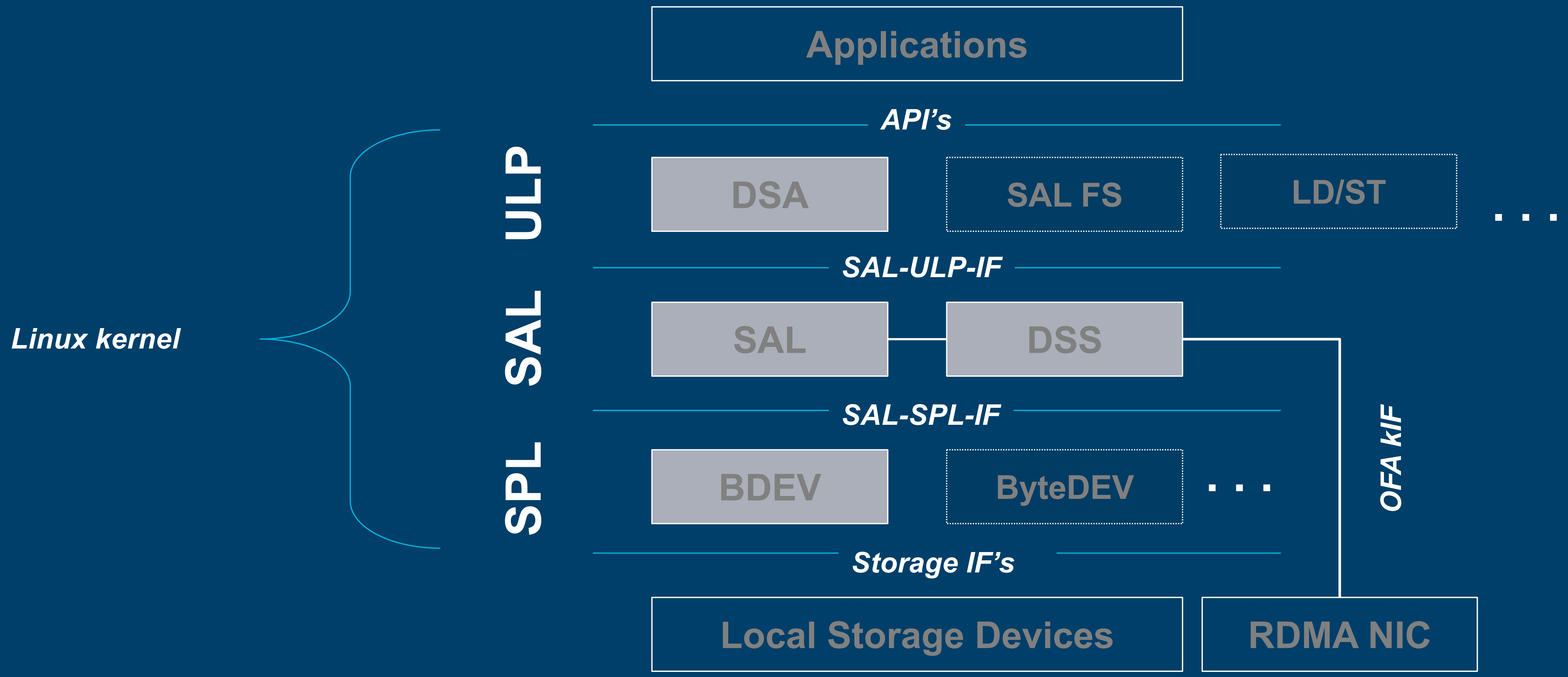
- Cover all types of SCM with one industry standard interface (OpenFabrics Verbs) and enable global sharing.

- Integrate Storage Class Memory access with industry standard RDMA communication stack.

- Use RDMA for global sharing

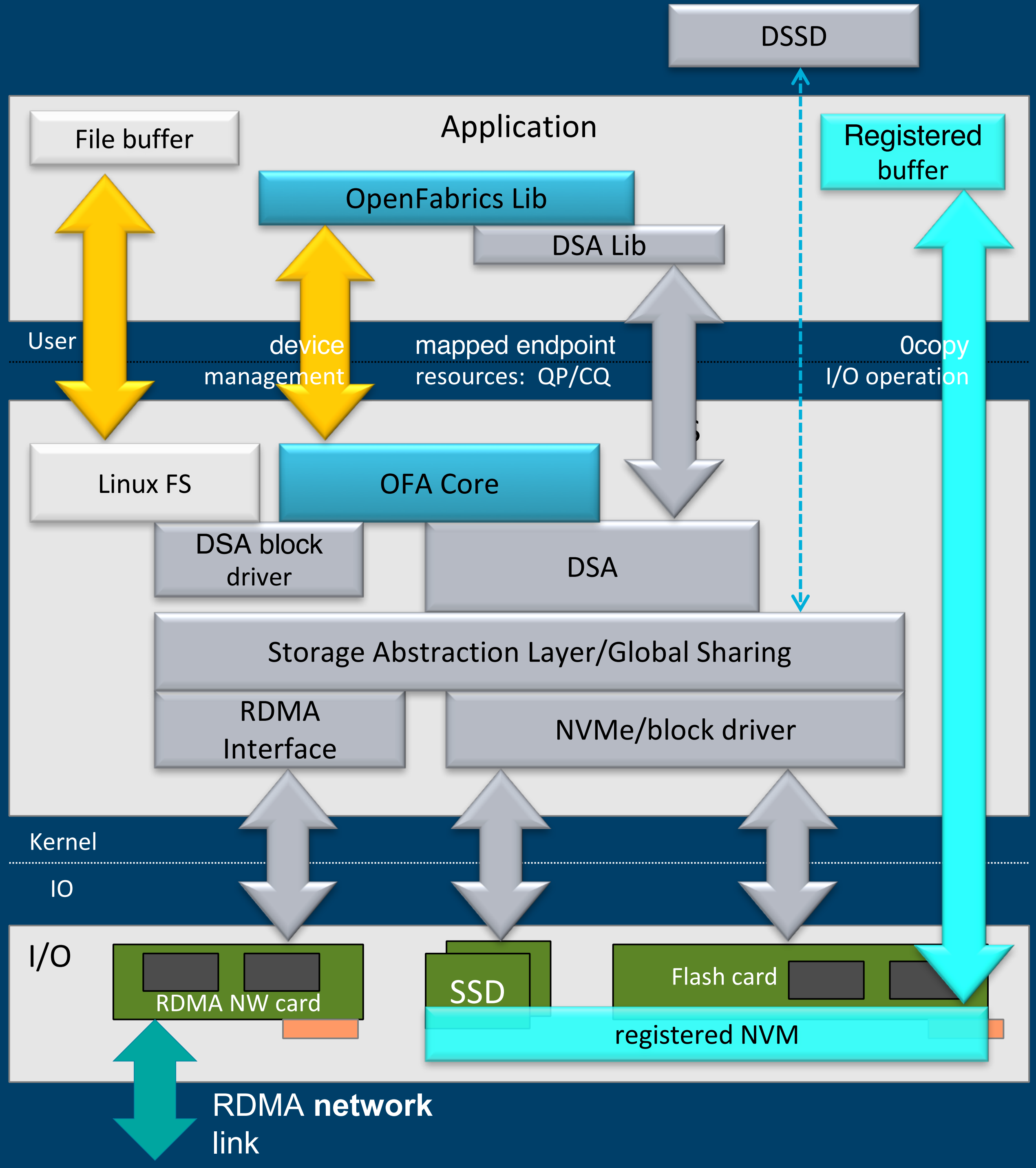
- to be open sourced

DSS Host Software Layering

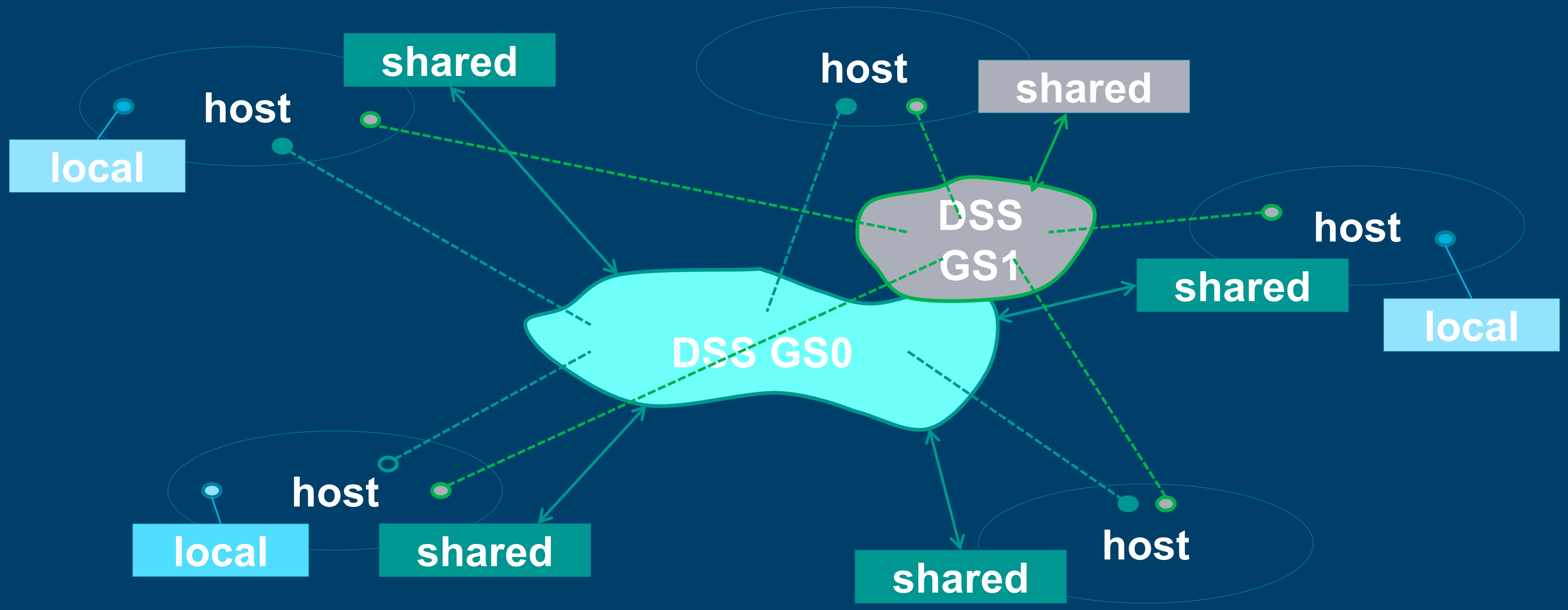


DSS Host Software Stack: Details

- OFA RDMA interface
- Local NVMe attachment
  - Generic device driver
  - Also works for any block dev
  - Byte granular access
    - Read-Modify-Write for block dev
- Global sharing
  - Global access space
  - IB/RDMA integration
  - Flexibly configurable
  - User level dssd process
  - dsssh command shell

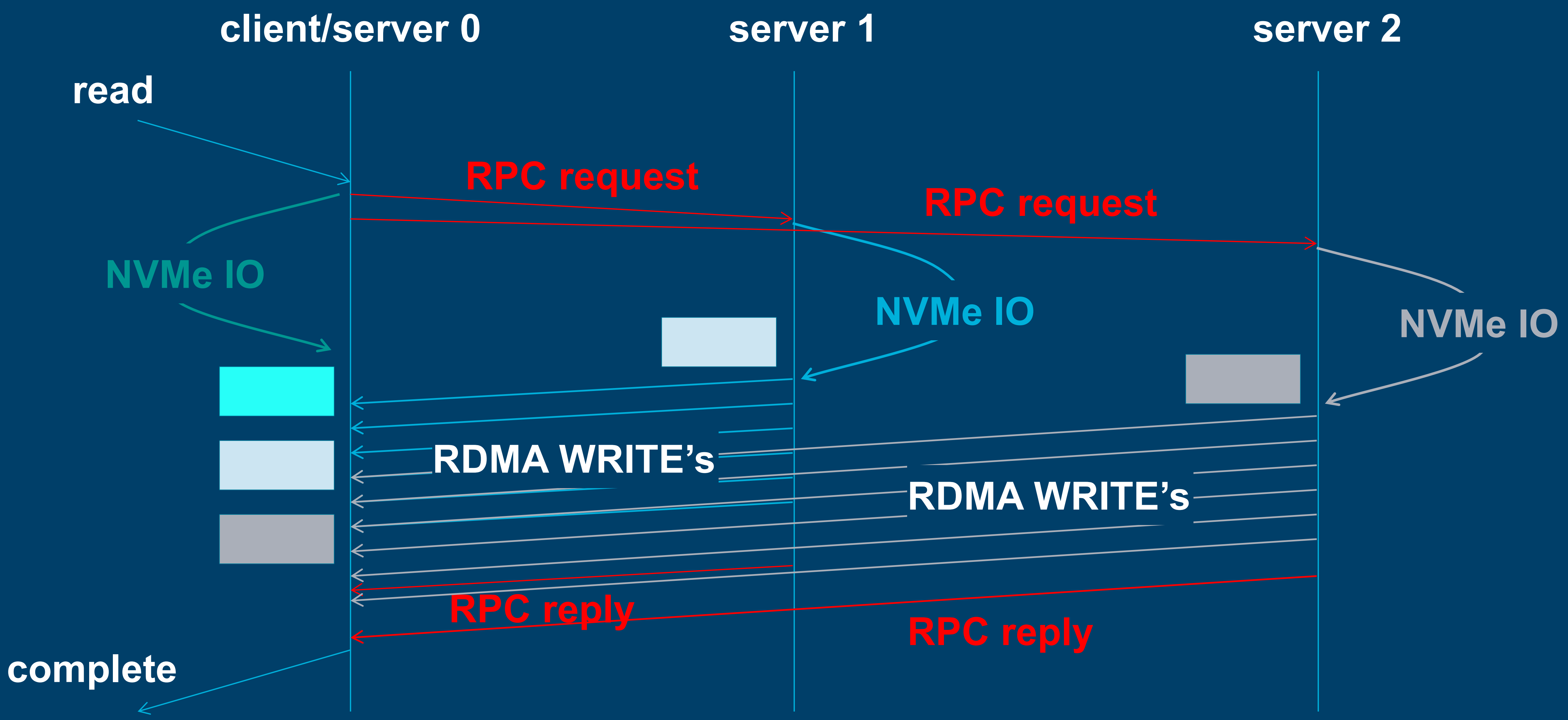


Example DSS Resource Configuration

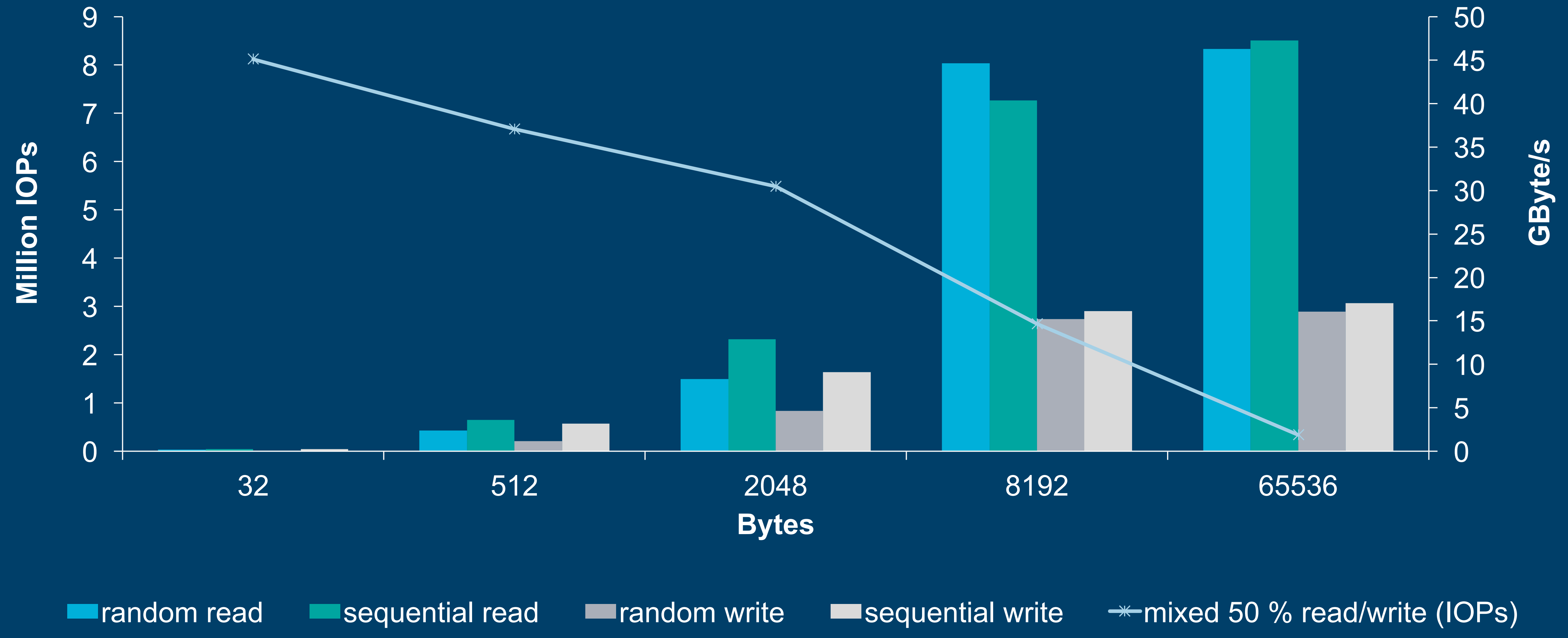


- Mix of local and shared DM resources
- Multiple shared DM partitions possible
- Partitions may overlap

RPC Protocol for distributed Access



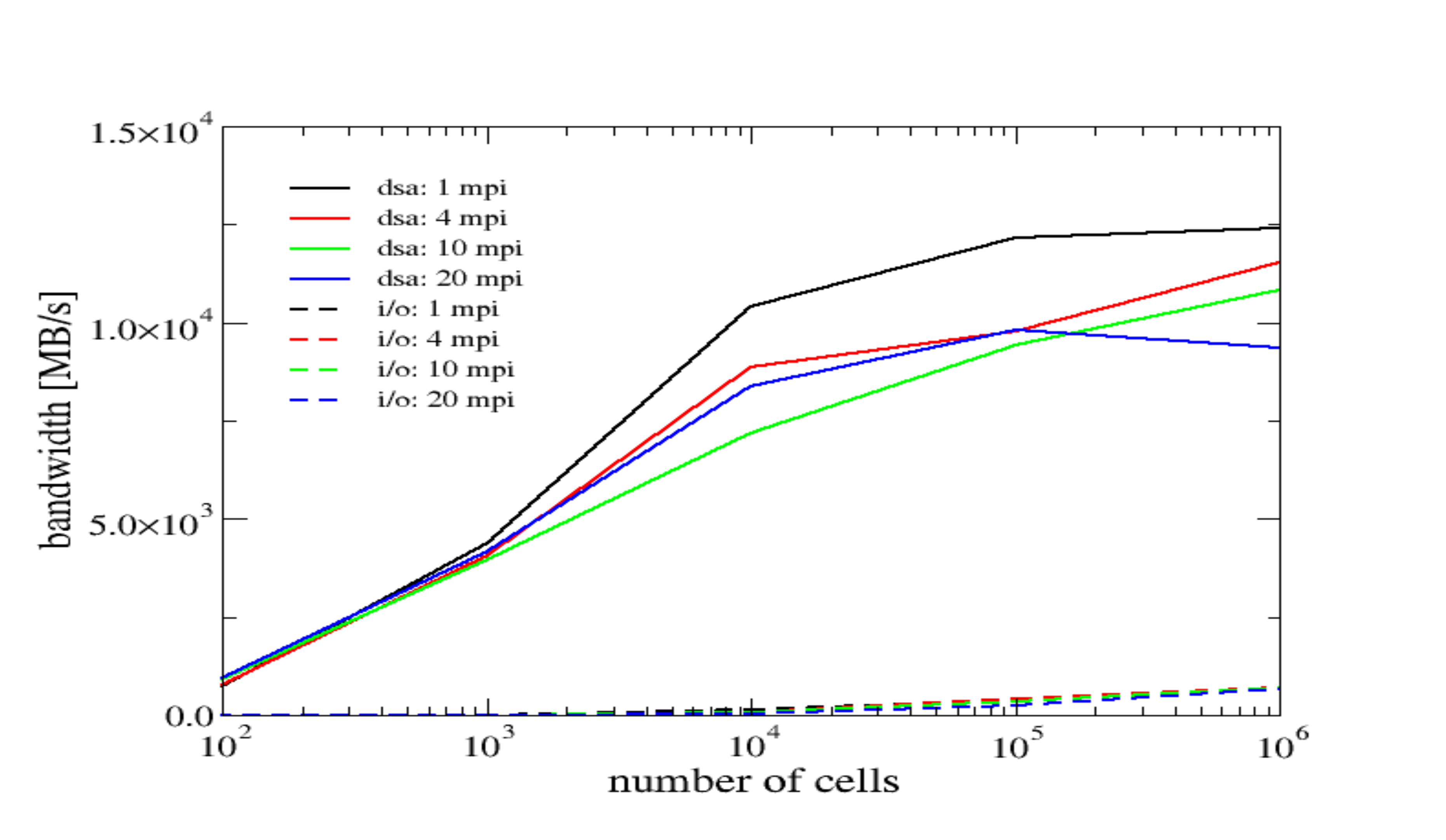
IOPs and BW on 16 Nodes: 16 Clients



16 clients, 16 servers, queue depth 1024, average from 2 x 10 seconds runs

Core-Neuron benchmark: RepLib mini-app ported to DSS

- Bandwidth using distributed DSS vs standard I/O
- Significant bandwidth gains!

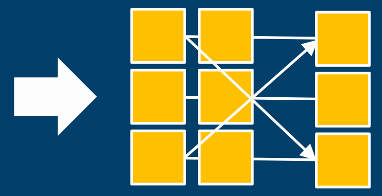


FPGA Technology: Parallel, pipelined

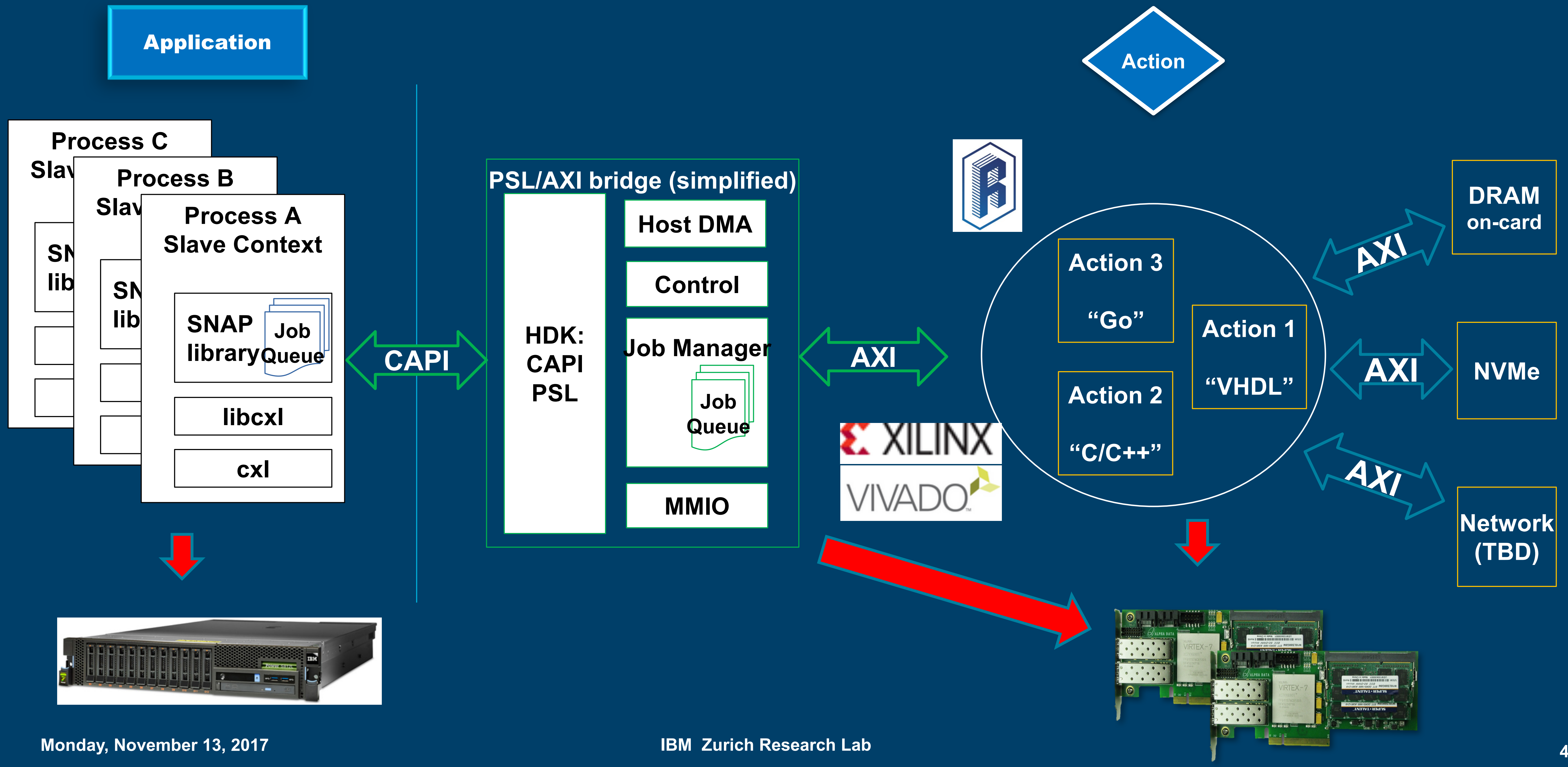
- Highly parallel designs
  - e.g. >100 AES or image scaling cores

- Deep pipeline 

- FPGA advantage comes from
  instruction complexity * parallelism * pipeline stages
  - Often >> 10x, makes up for lower frequency (~250MHz vs. ~3GHz)
  - Limit: available FPGA resources
- Control data flow dynamically, e.g. select min of 4 values



CAPI SNAP Framework : the whole picture



Summary

- 300+ members working on open innovation (… many for HPC)

Implementation, HPC & Research

Software

System Integration

I/O, Storage & Acceleration

Boards & Systems

Chips & SoCs

THINK

BIG