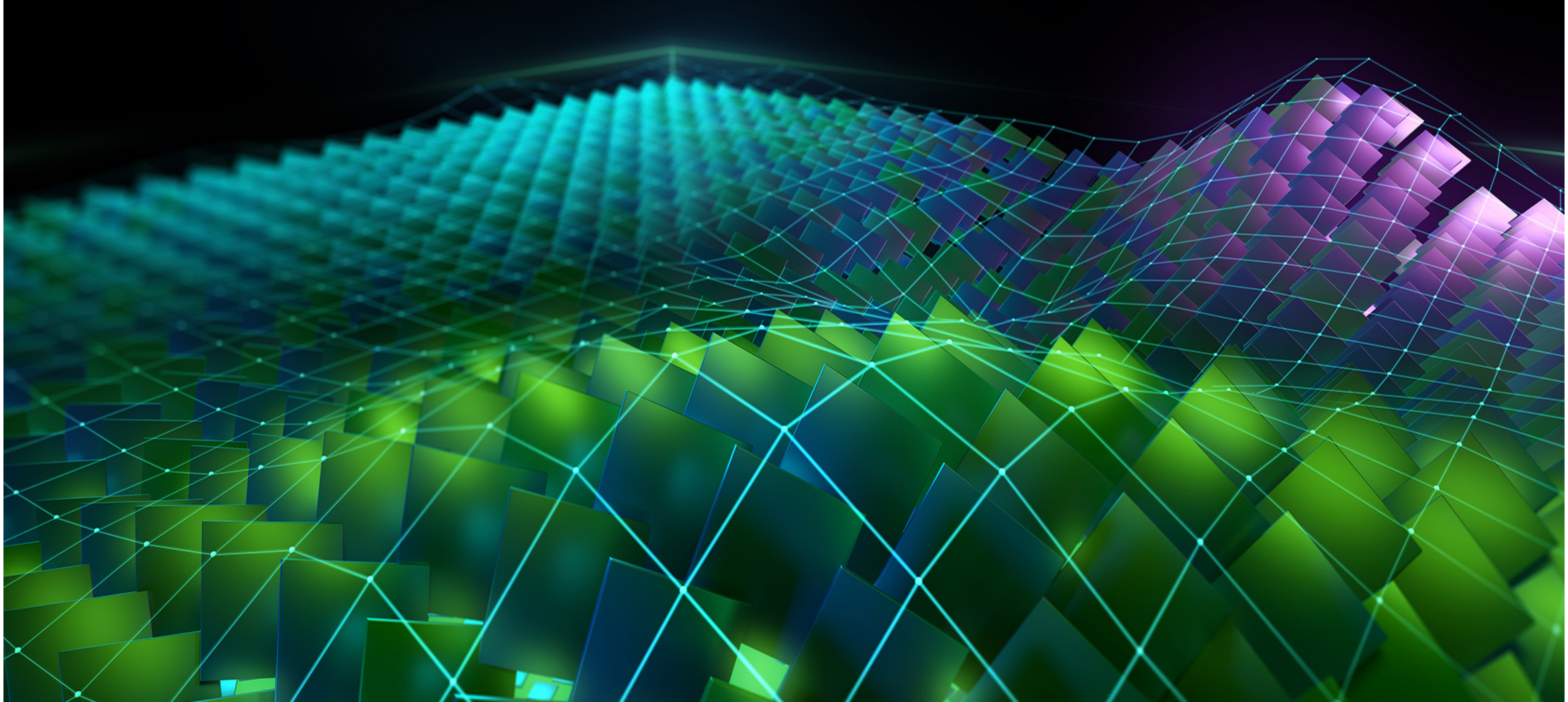
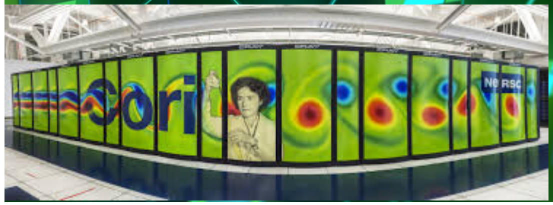
GPU Computing with CUDA (and beyond)
Part 4: Programing Multiple GPU Nodes



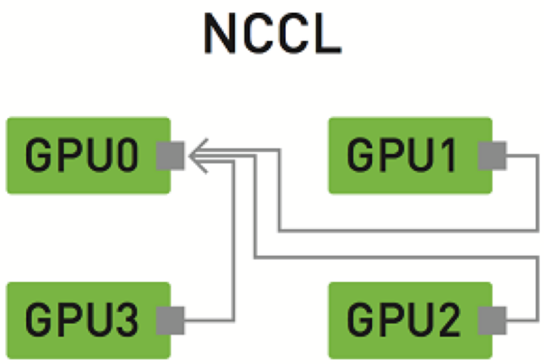


Johannes Langguth

Simula Research Laboratory

Collective Communications

NCCL does not have the collective communications we need, but there is a system which does: MPI



MPI: Message Passing Interface

MPI has been the standard for supercomputer communication since the 1990. It is a highly mature system.

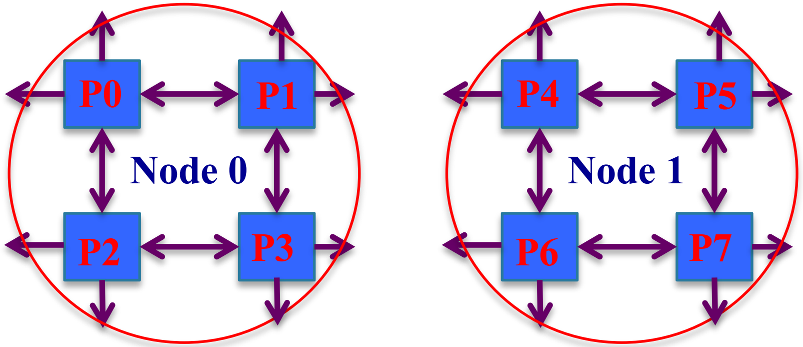
MPI is a standard. Several implementations exist.
Current version: MPI 3.0, 4.0 is under discussion.

Message passing programming model

- One way to program distributed memory computers is to use *message passing*, e.g. MPI
- Processes send and receive messages and have direct access to local memory only
- Processes share the interconnect
- Dominant control model: all ranks execute the same program (SPMD) and the number of ranks is fixed

Programming with Message Passing

- Programs execute as a set of P processes (user specifies P)
- Each process assumed to run on a different core
  - Usually initialized with the same code, but has private state SPMD = "Same Program Multiple Data"
  - Communicates with other processes by passing messages
  - Executes instructions at its own rate according to its *rank* (0:P-1) and the messages it sends and receives



Programming with Message Passing

- There are two kinds of communication patterns:
- ***Point-to-point*** communication:
  a single pair of communicating processes copy data between address space
- ***Collective communication***: all the processors participate, possibly exchanging information

A Hello World in MPI

```
#include "mpi.h”
int main(int argc, char **argv ){
  MPI_Init(&argc, &argv);
  int rank, size;
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf("Hello, world! I am process %d of %d.\n”,
  rank, size);
  MPI_Finalize();
  return(0);
}
```

A Hello World in MPI

```
mpicc hello.c -o hello
mpirun -np 4 ./hello
```

```
Hello, world! I am process 2 of 4.
Hello, world! I am process 0 of 4.
Hello, world! I am process 3 of 4.
Hello, world! I am process 1 of 4.
```

MPI processes (called ranks) are OS processes. They do not share memory. They can run on separate computers over a network, but we will stay on the DGX-2 for now.

Messaging in MPI

Basic MPI communication is 2-sided. Sender and receiver must do something to move the data.

OpenMP:

```
cudaMemcpyAsynch(V[sep[i][j]],V[sep[i][j]],
      sepsize, cudaMemcpyDeviceToDevice);
```

MPI:

```
if (rank == i)
      MPI_Send();
if (rank == j)
      MPI_Recv();
```

Send and Recv

```
const int Tag=99;
int msg[2] = { rank, rank * rank};
if (rank == 0) {
        MPI_Status status;
        MPI_Recv(msg, 2,
                 MPI_INT, 1,
                 Tag, MPI_COMM_WORLD, &status);
}
else  MPI_Send(msg, 2,
                 MPI_INT, 0,
                 Tag, MPI_COMM_WORLD);
```

Message length

SOURCE Process ID

Message Buffer

Message Tag

Communicator

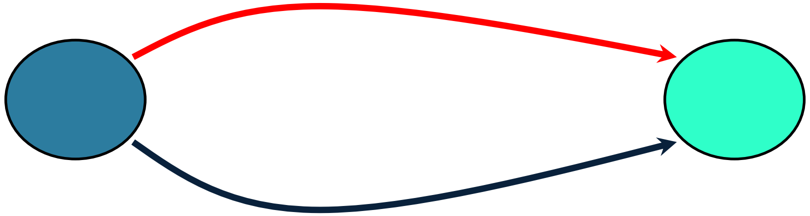
Destination Process ID

Communicators

- A communicator is a name-space (or a context) describing a set of processes that may communicate
- MPI defines a default communicator **MPI_COMM_WORLD** containing all processes
- MPI provides the means of generating uniquely named subsets
- A mechanism for screening or filtering messages

MPI Tags

- Tags enable processes to organize or screen messages
- Each sent message is accompanied by a user-defined integer *tag*:
  - Receiving process can use this information to organize or *filter* messages
  - **MPI_ANY_TAG** inhibits tag filtering



MPI Datatypes

- MPI messages have a specified length
- The unit depends on the type of the data
    - The length in bytes is sizeof(type) × # elements
    - We don’t specify length as the # byte
- MPI specifies a set of built-in types for each of the primitive types of the language
- In C: **MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR, MPI_LONG, MPI_UNSIGNED, MPI_BYTE,…**
- Also defined types, e.g. structs

Messaging for our Application

```
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
if (rank == j)
      MPI_Recv(&V[sep[i][j]], sepsize, MPI_DOUBLE,
            i, tag, MPI_COMM_WORLD, &status);

if (rank == i)
      MPI_Send(&V[sep[i][j]], sepsize, MPI_DOUBLE,
            j, tag, MPI_COMM_WORLD);
```

Asynchronous versions of Send and Recv are available.
With these, we can replicate the OpenMP version
It could even run on multiple nodes.
But we can do better!

Collective Communications in MPI

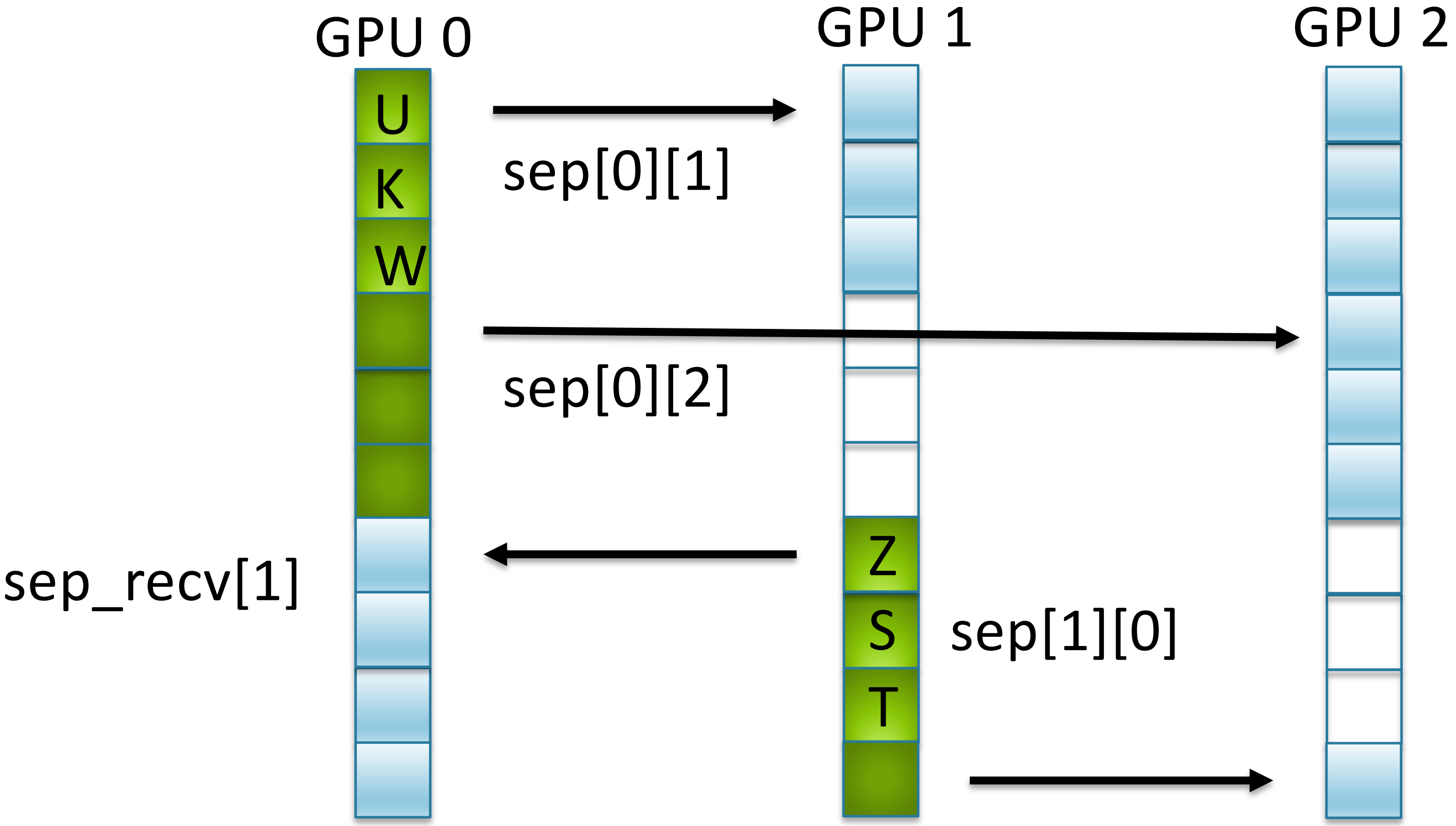
We can replace communication by:

```
int MPI_Alltoallv(V, sepsize, sep[myrank], MPI_DOUBLE,
 Vdest, sepsize, sep_recv[myrank], MPI_ DOUBLE,
  MPI_COMM_WORLD);
```

Collective Communications in MPI

MPI collective communication:

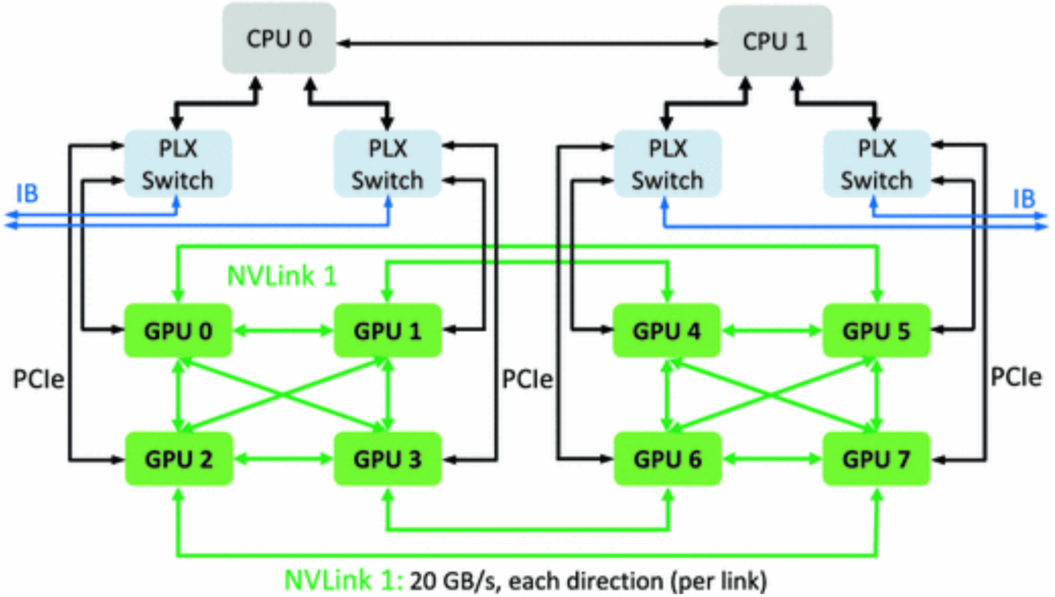
```
int MPI_Alltoallv(V, sepsize, sep[myrank], MPI_DOUBLE,
 Vdest, sepsize, sep_recv[myrank], MPI_ DOUBLE,
   MPI_COMM_WORLD);
```



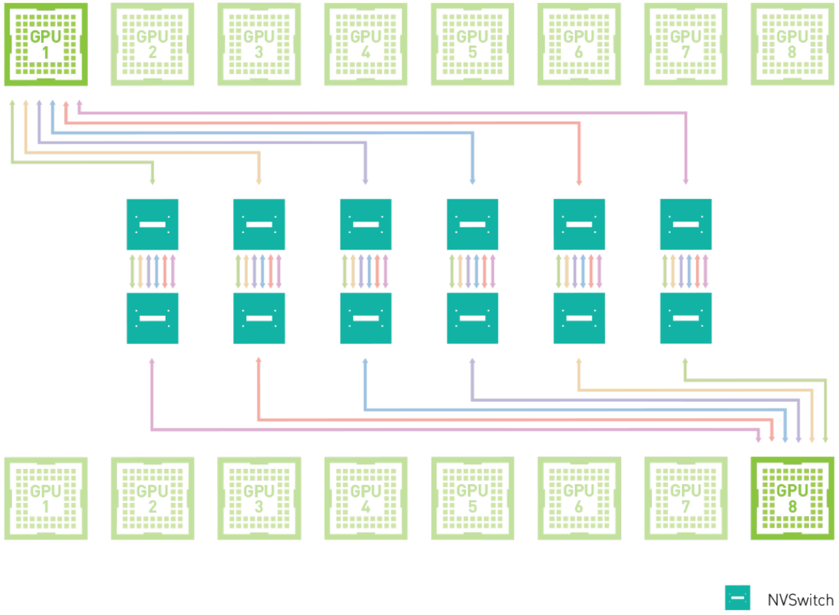
Collective Communications in MPI

Main advantage of collective communications: the system decides.

DGX-1



DGX-2



- Match communication patterns to network topology
- Avoid hand-optimizing communication

More Collective Communications in MPI

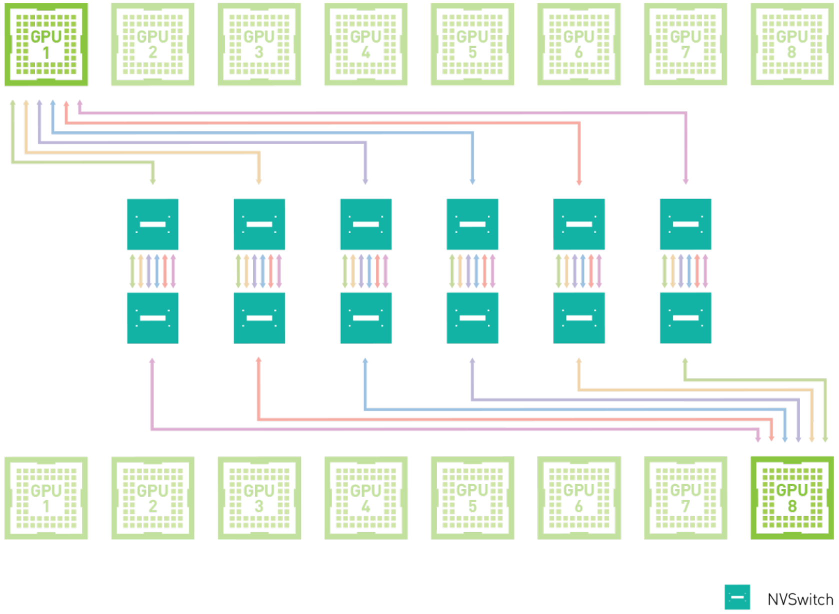
Broadcast: distribute data one process to all the others
Reduce: combine data from all processes on the root process
Scatter: spread array among all other ranks
Gather: collect elements from each rank in one array on root
Allgather: each rank collects the array

Scatter, Gather, and Allgather have variable length (v) versions

All collectives have nonblocking versions.

CUDA-aware MPI

Problem: MPI collectives are nice, but they happen on the CPU...



Remember: copying data between CPU and GPU is costly.
(unless your application is communication-light)

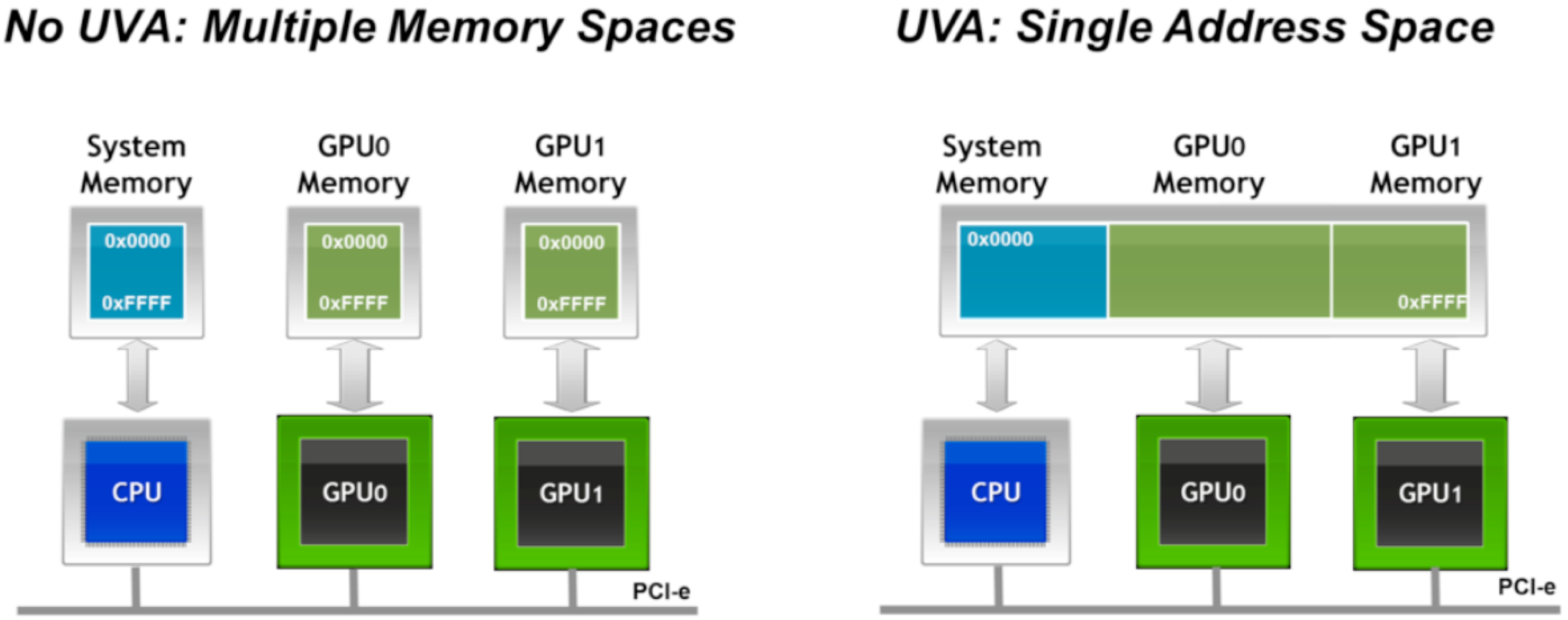
Can we use MPI to move data through the NVSwitch ?

CUDA-aware MPI

- CUDA-aware MPI communication between GPUs
- use Nvlink etc. within a node.
- Unified Virtual Adressing to specify location.

```
int MPI_Alltoallv(V, sepsize, sep[myrank], MPI_DOUBLE,
Vdest, sepsize, sep_recv[myrank], MPI_ DOUBLE,
MPI_COMM_WORLD);
```

Remember Unified Virtual Adressing



- Unified Virtual Adressing to points MPI to GPU memory

```
int MPI_Alltoallv(V, sepsize, sep[rank], MPI_DOUBLE,
Vdest, sepsize, sep_recv, MPI_ DOUBLE,
MPI_COMM_WORLD);
```

CUDA-aware MPI

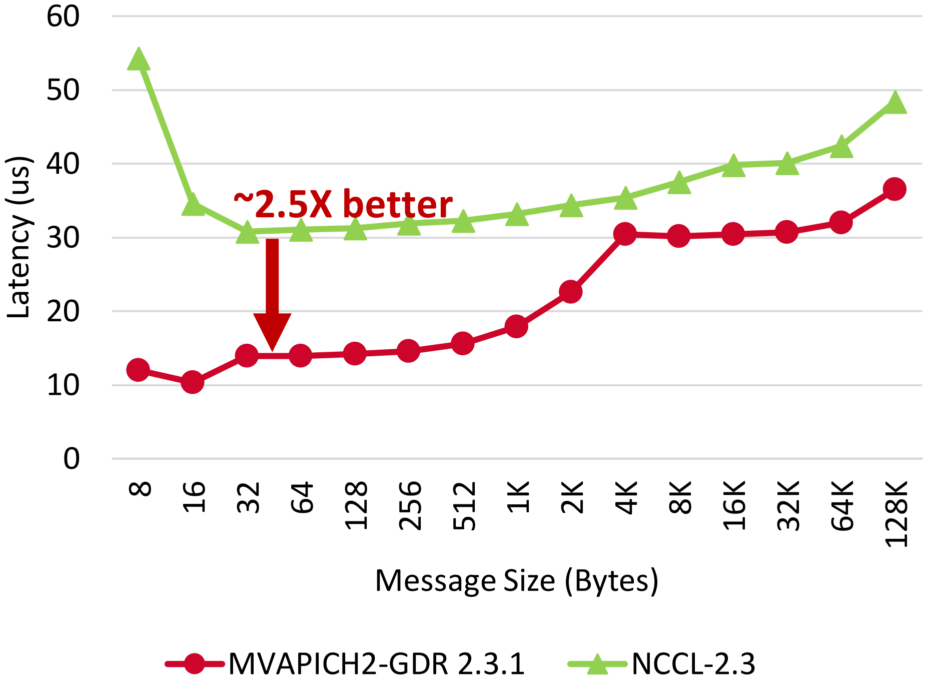
CUDA-aware MPI implementations:

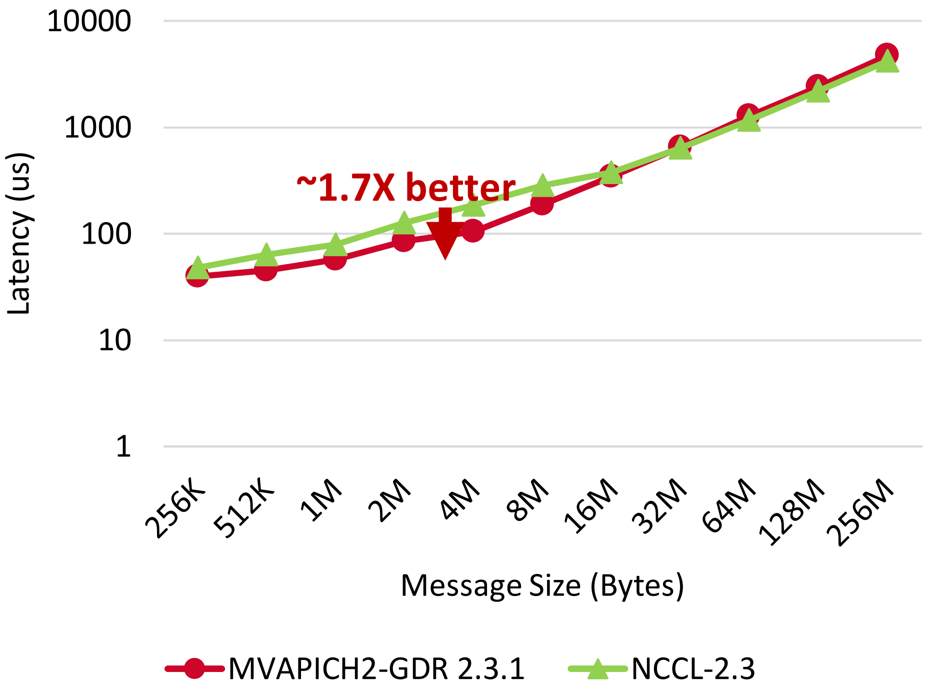
MVAPICH2	1.8/1.9b
OpenMPI	1.7 (beta)
CRAY MPI	(MPT 5.6.2)
IBM Platform MPI	(8.3)
SGI MPI	(1.08)

CUDA-aware MPI: MVAPICH2-GDR

MVAPICH2-GDR vs. NCCL2 – Allreduce Operation (DGX-2)

- **Optimized designs in MVAPICH2-GDR 2.3.1 offer better/comparable performance for most cases**
- **MPI_Allreduce (MVAPICH2-GDR) vs. ncclAllreduce (NCCL2) on 1 DGX-2 node (16 Volta GPUs)**



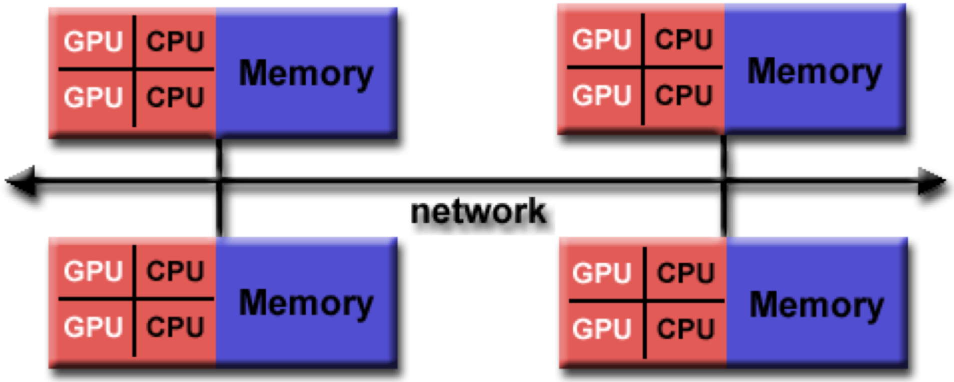


Platform: Nvidia DGX-2 system (16 Nvidia Volta GPUs connected with NVSwitch), CUDA 9.2

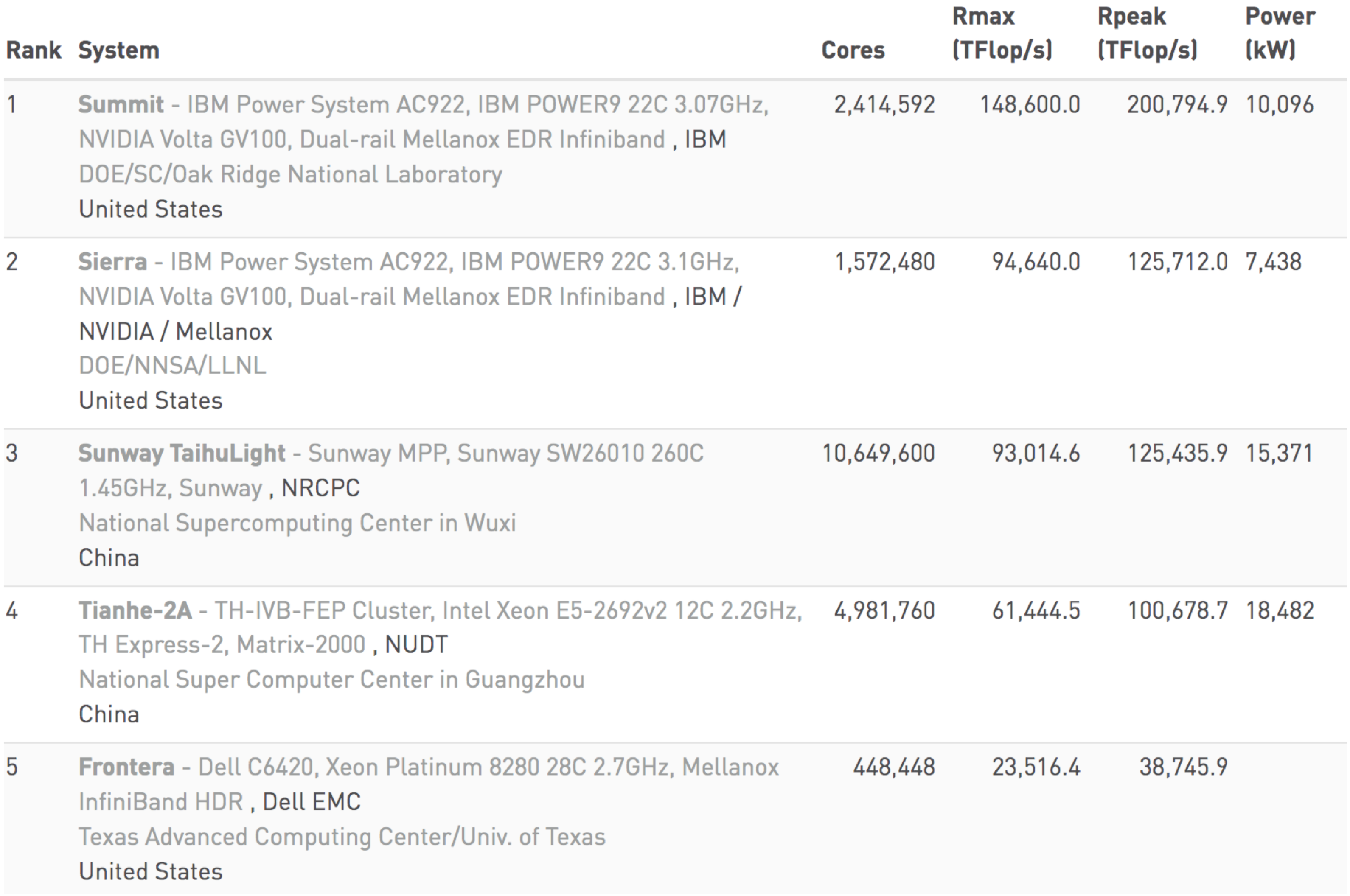
Next Step: CUDA on Supercomputers

- DGX-2 is powerful, but cannot be extended
- Need to connect multiple machines
- Each machine is an independent **compute node**
- Multiple nodes + highspeed interconnect

  =**Supercomputer**

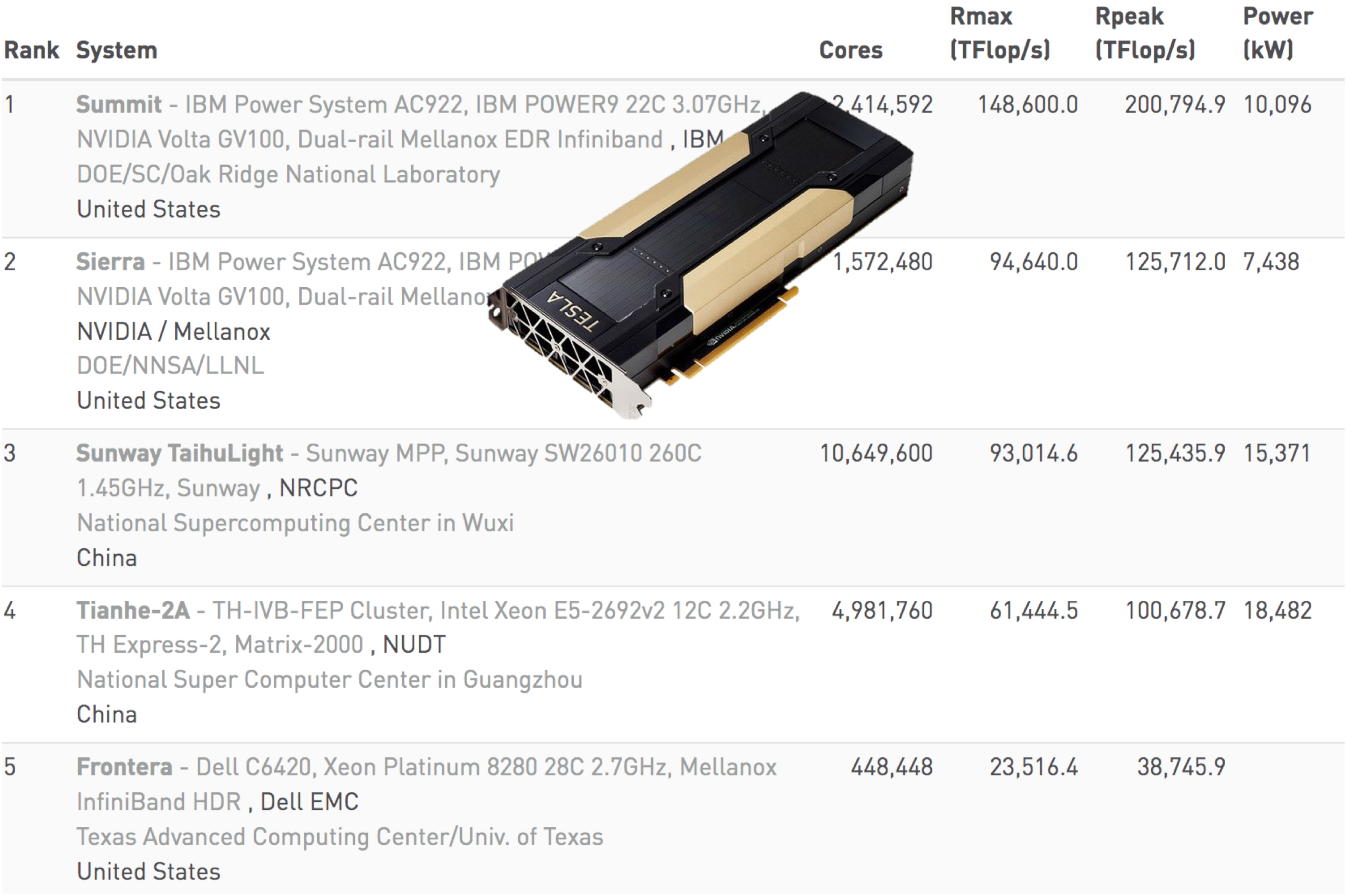


Top 500: The List of Supercomputers



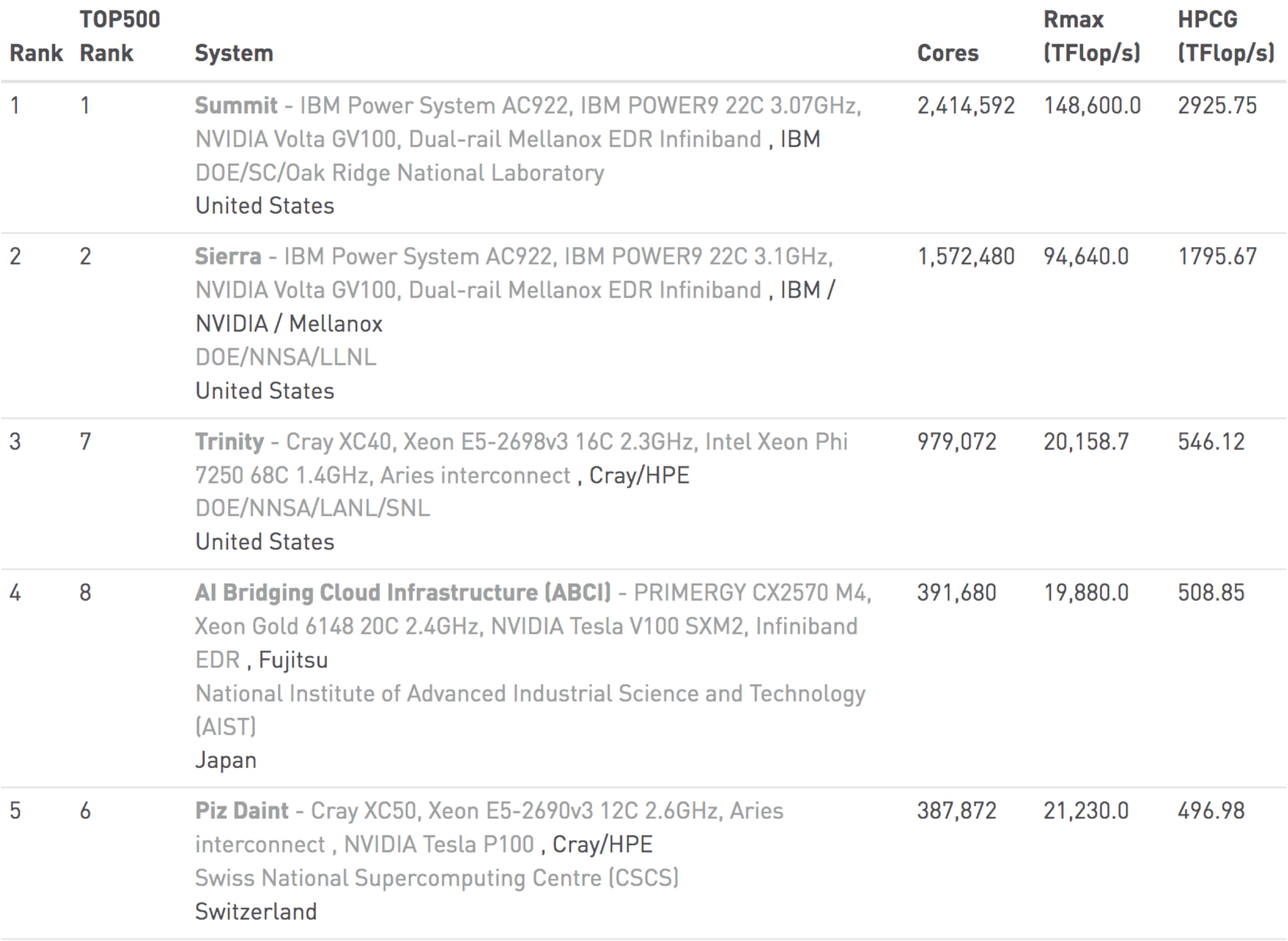
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

Top 500: The List of Supercomputers



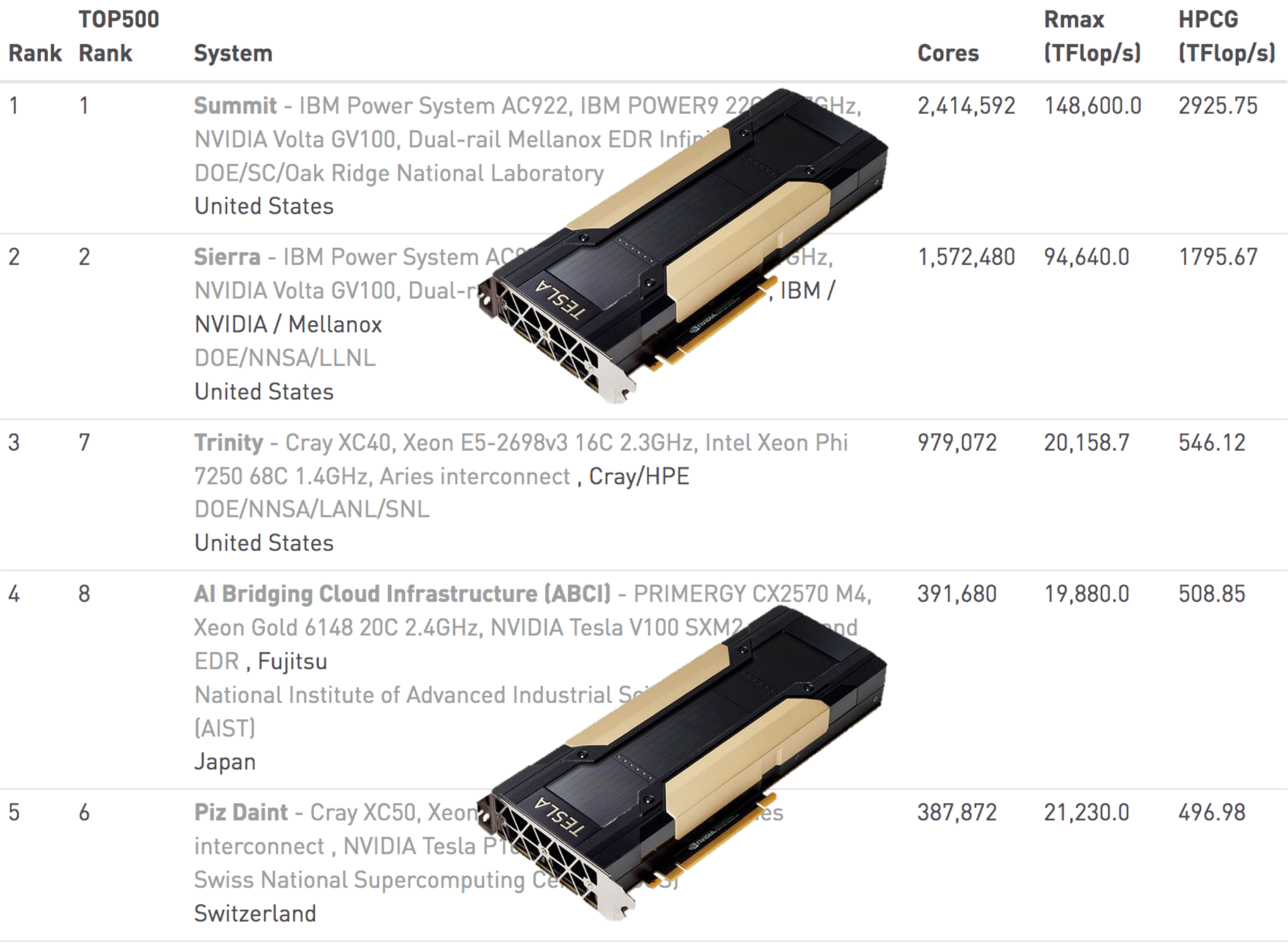
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System AC922, IBM PO NVIDIA Volta GV100, Dual-rail Mellano NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

HPCG 500: The List that Matters



Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	HPCG (TFlop/s)
1	1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	2925.75
2	2	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	1795.67
3	7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray/HPE DOE/NNSA/LANL/SNL United States	979,072	20,158.7	546.12
4	8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	508.85
5	6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	496.98

HPCG 500: The List that Matters

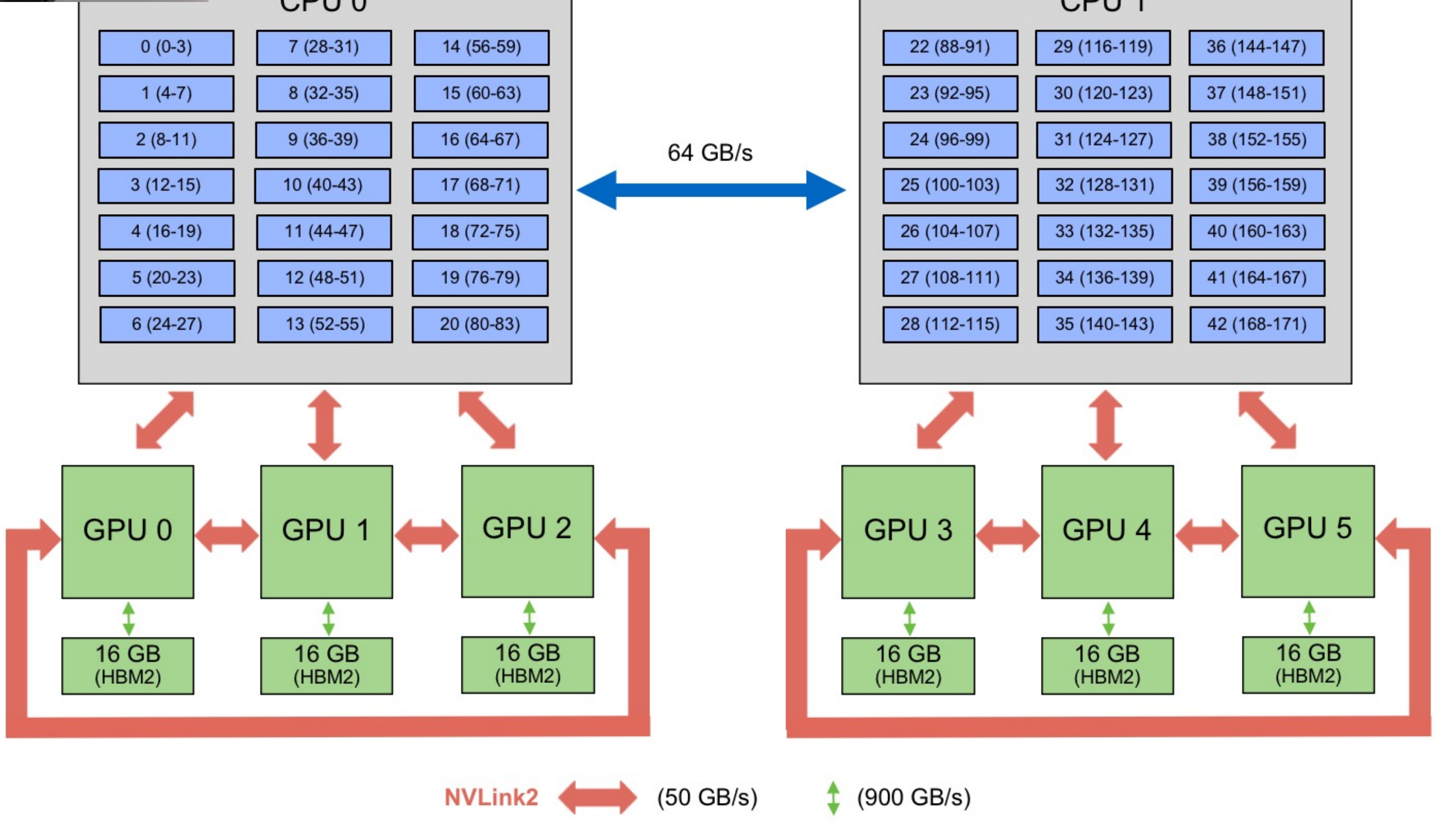


Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	HPCG (TFlop/s)
1	1	**Summit** - IBM Power System AC922, IBM POWER9 22 GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infi DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	2925.75
2	2	**Sierra** - IBM Power System AC GHz, NVIDIA Volta GV100, Dual-r , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	1795.67
3	7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray/HPE DOE/NNSA/LANL/SNL United States	979,072	20,158.7	546.12
4	8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2 nd EDR , Fujitsu National Institute of Advanced Industrial Sc (AIST) Japan	391,680	19,880.0	508.85
5	6	**Piz Daint** - Cray XC50, Xeon es interconnect , NVIDIA Tesla P1 Swiss National Supercomputing Ce S) Switzerland	387,872	21,230.0	496.98

Summit: the Top of Top 500

Summit Node

(2) IBM Power9 + (6) NVIDIA Volta V100



CPU 0

0 (0-3)	7 (28-31)	14 (56-59)
1 (4-7)	8 (32-35)	15 (60-63)
2 (8-11)	9 (36-39)	16 (64-67)
3 (12-15)	10 (40-43)	17 (68-71)
4 (16-19)	11 (44-47)	18 (72-75)
5 (20-23)	12 (48-51)	19 (76-79)
6 (24-27)	13 (52-55)	20 (80-83)

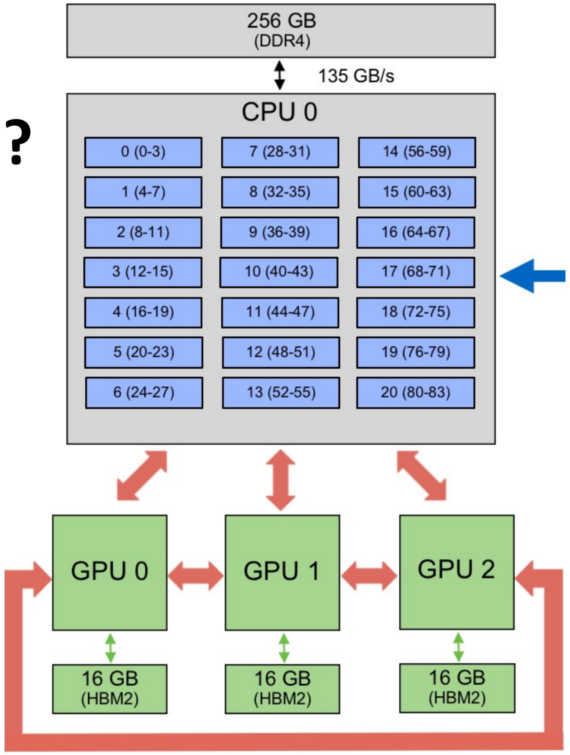
CPU 1

22 (88-91)	29 (116-119)	36 (144-147)
23 (92-95)	30 (120-123)	37 (148-151)
24 (96-99)	31 (124-127)	38 (152-155)
25 (100-103)	32 (128-131)	39 (156-159)
26 (104-107)	33 (132-135)	40 (160-163)
27 (108-111)	34 (136-139)	41 (164-167)
28 (112-115)	35 (140-143)	42 (168-171)

Summit: the Top of Top 500

- Many Supercomputers follow the Summit design
- 2 groups with 1 CPU and 3 GPUs per node
- Unlike DGX-2, CPU-GPU and GPU-GPU is equal
- GPUs not connected to other group

How to distribute computation among GPUs ?



256 GB
(DDR4)

135 GB/s

CPU 0

0 (0-3)	7 (28-31)	14 (56-59)
1 (4-7)	8 (32-35)	15 (60-63)
2 (8-11)	9 (36-39)	16 (64-67)
3 (12-15)	10 (40-43)	17 (68-71)
4 (16-19)	11 (44-47)	18 (72-75)
5 (20-23)	12 (48-51)	19 (76-79)
6 (24-27)	13 (52-55)	20 (80-83)

GPU 0

GPU 1

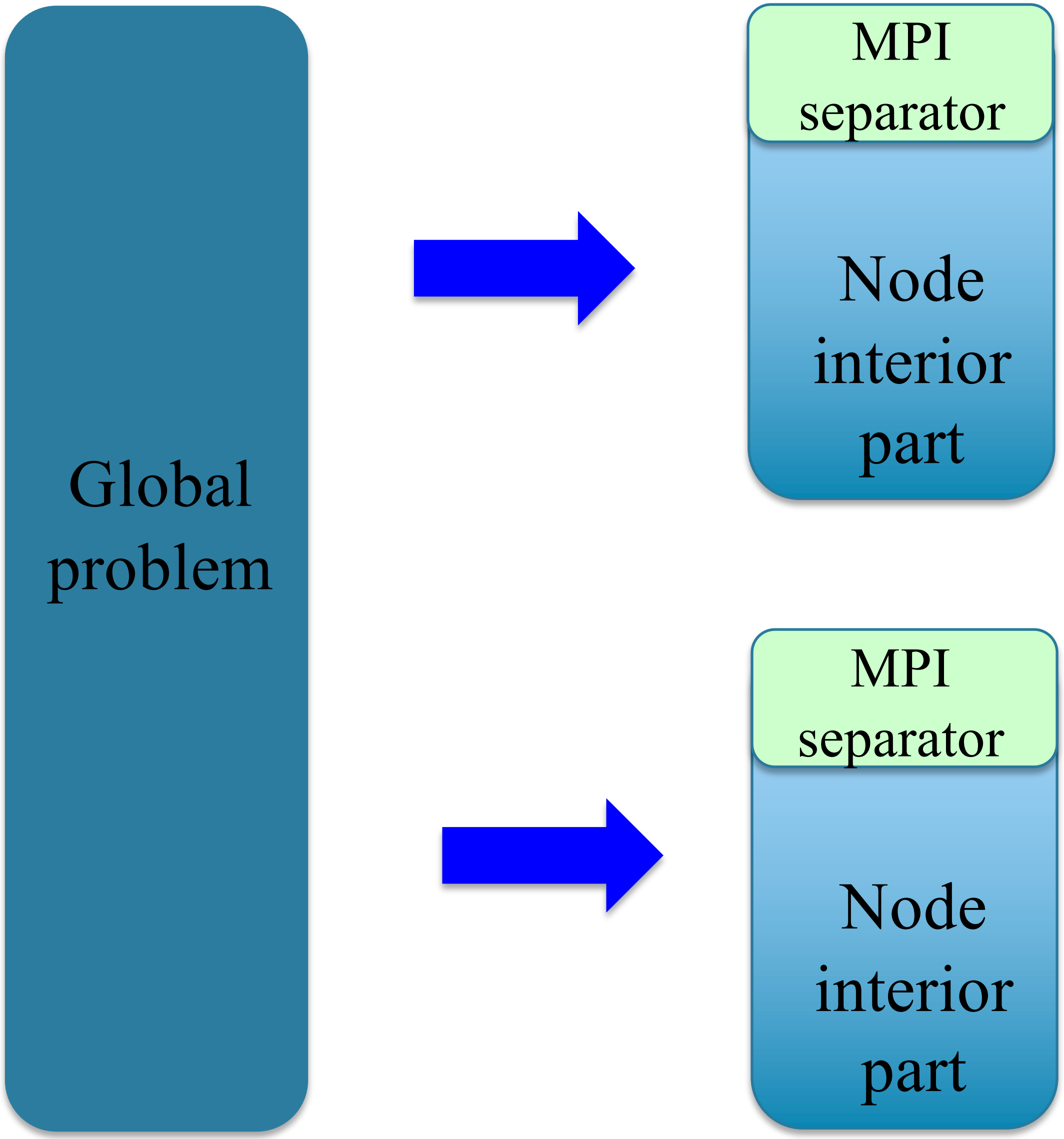
GPU 2

16 GB
(HBM2)

16 GB
(HBM2)

16 GB
(HBM2)

Traditional Partitioning for Multiple Nodes



Global problem

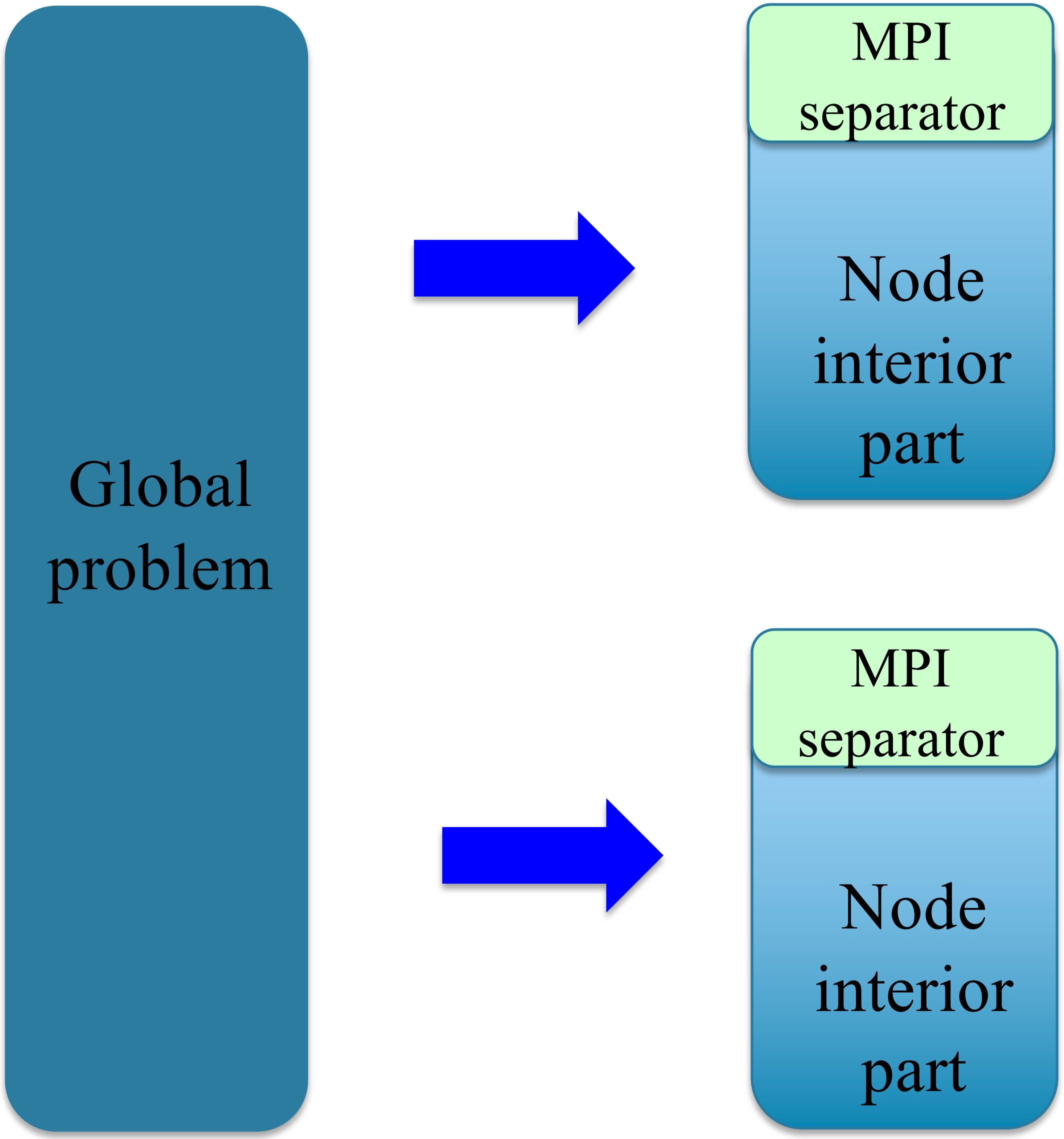
MPI separator

Node interior part

MPI separator

Node interior part

Traditional Partitioning for Multiple Nodes



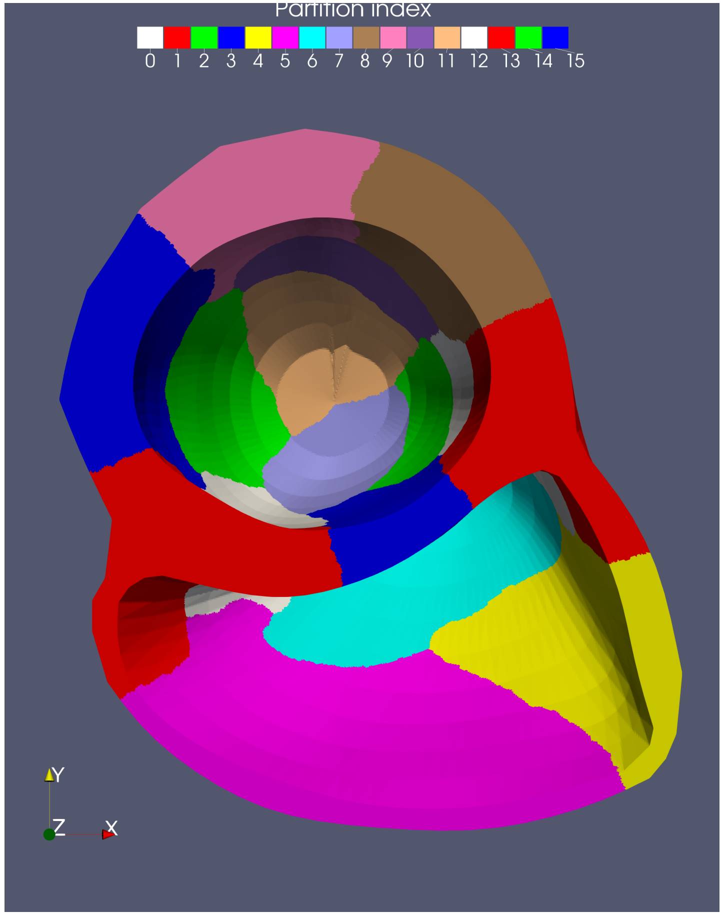
Global problem

MPI separator

Node interior part

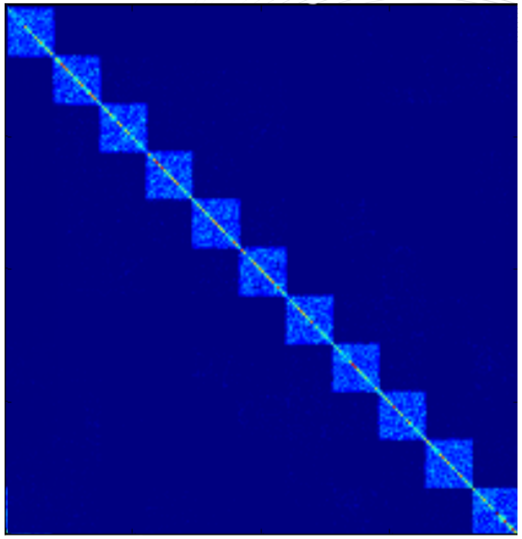
MPI separator

Node interior part



Separators

$$\left(\begin{array}{cccccc} * & * & * & & & \\ * & * & * & & & \\ \hline * & & * & * & * & \\ & * & * & & * & * \\ \hline & & & * & * & * \\ & & & * & * & * \end{array}\right) \qquad \begin{array}{c} \text{Block} \\ \hline \text{Separator} \\ \hline \text{Block} \end{array}$$

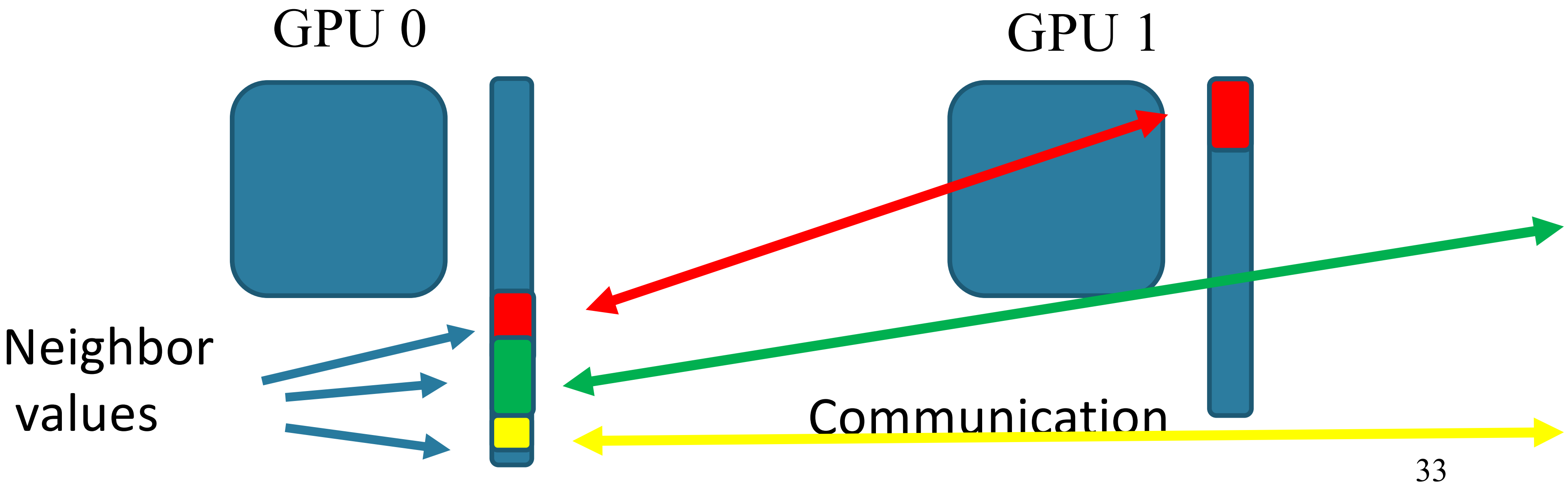


Separator: set of vertices whose removal makes graph disconnected

Graph communicators in MPI-3

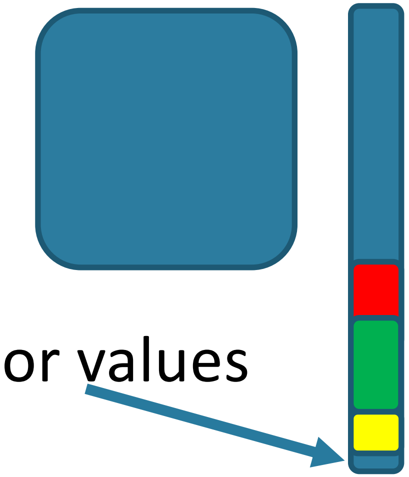
```
MPI_Dist_graph_create_adjacent(MPI_COMM_WORLD,
neighbourcount, graphneighbours, (int *)MPI_UNWEIGHTED,
neighbourcount, graphneighbours, (int *)MPI_UNWEIGHTED,
      MPI_INFO_NULL, 0, graphcomm_cl);

MPI_Neighbor_alltoallv(V, sendsizes, senddisps,
MPI_DOUBLE, V+mysize, recvsizes, recvdisps, MPI_DOUBLE,
      *(graphcomm_cl));
```



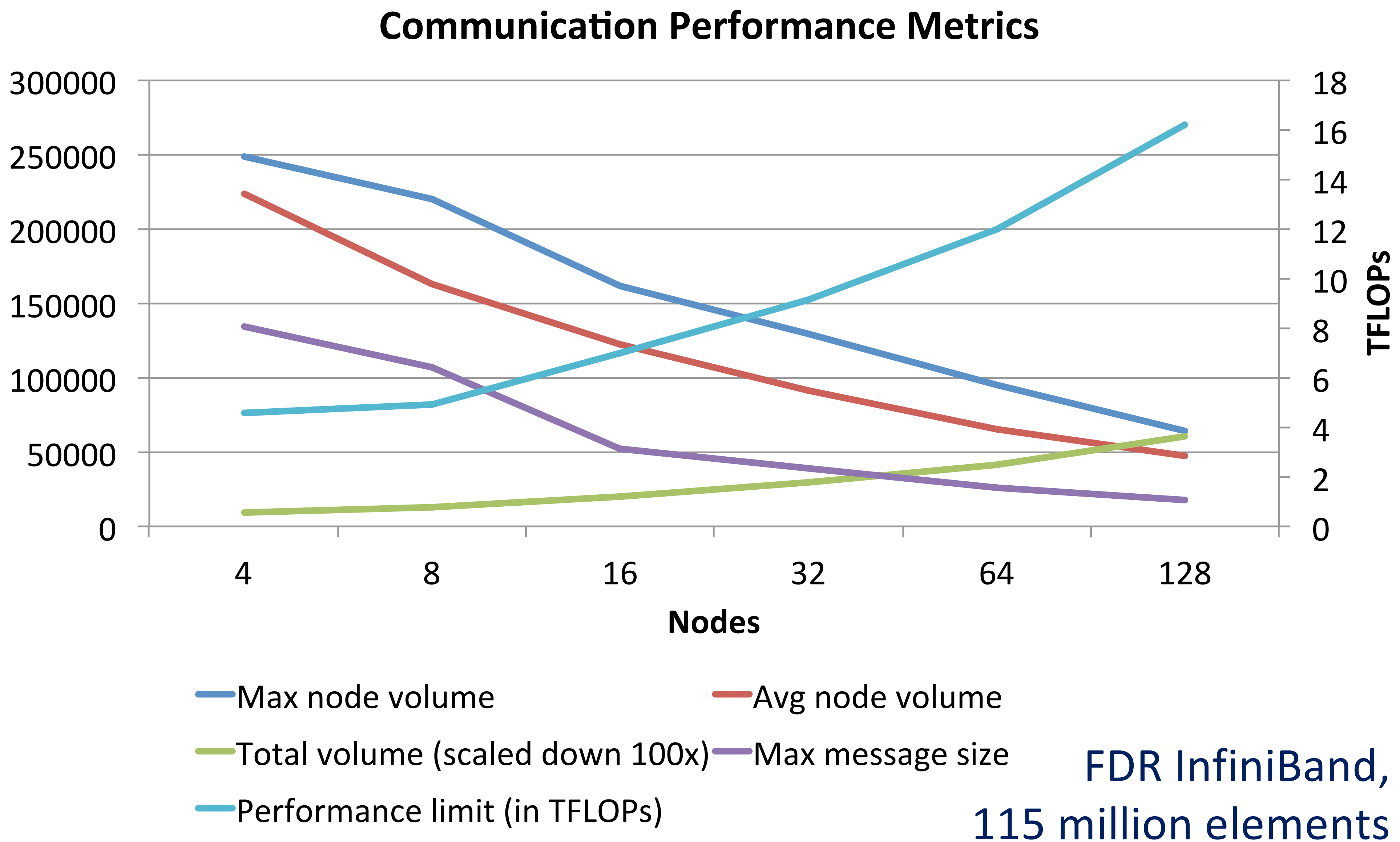
Problem: Mixed adjacency elements

```
for (int i = 0; i < sendcount_mixed; i ++) {
      sendbuffer[i] =   newV[sendidx_mixed[i]];
}
MPI_Neighbor_alltoallv(sendbuffer, sendsizes_mixed,
senddisps_mixed,  MPI_DOUBLE,
newV+mysize+remoteVcount_clean, recvsizes_mixed,
      recvdisps_mixed, MPI_DOUBLE, *graphcomm_mixed);}
```



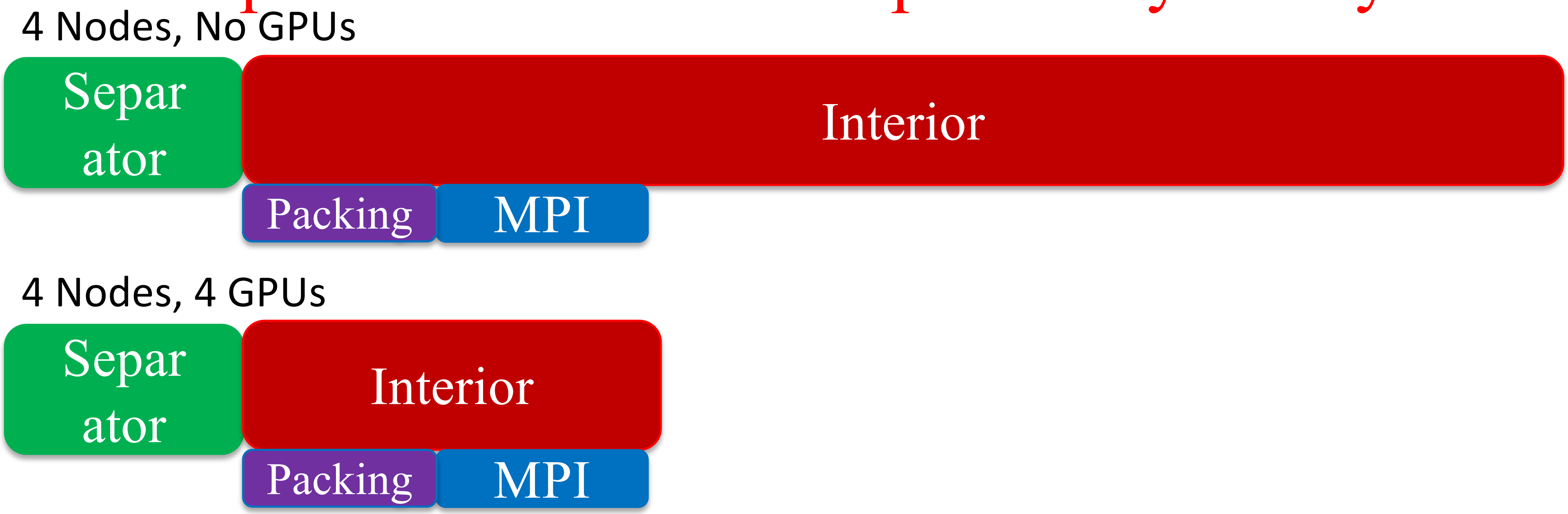
Mixed neighbor values

Homogeneous Communication



FDR InfiniBand,
115 million elements

Separators Become Comparatively Heavy



4 Nodes, No GPUs

Separator

Interior

Packing

MPI

4 Nodes, 4 GPUs

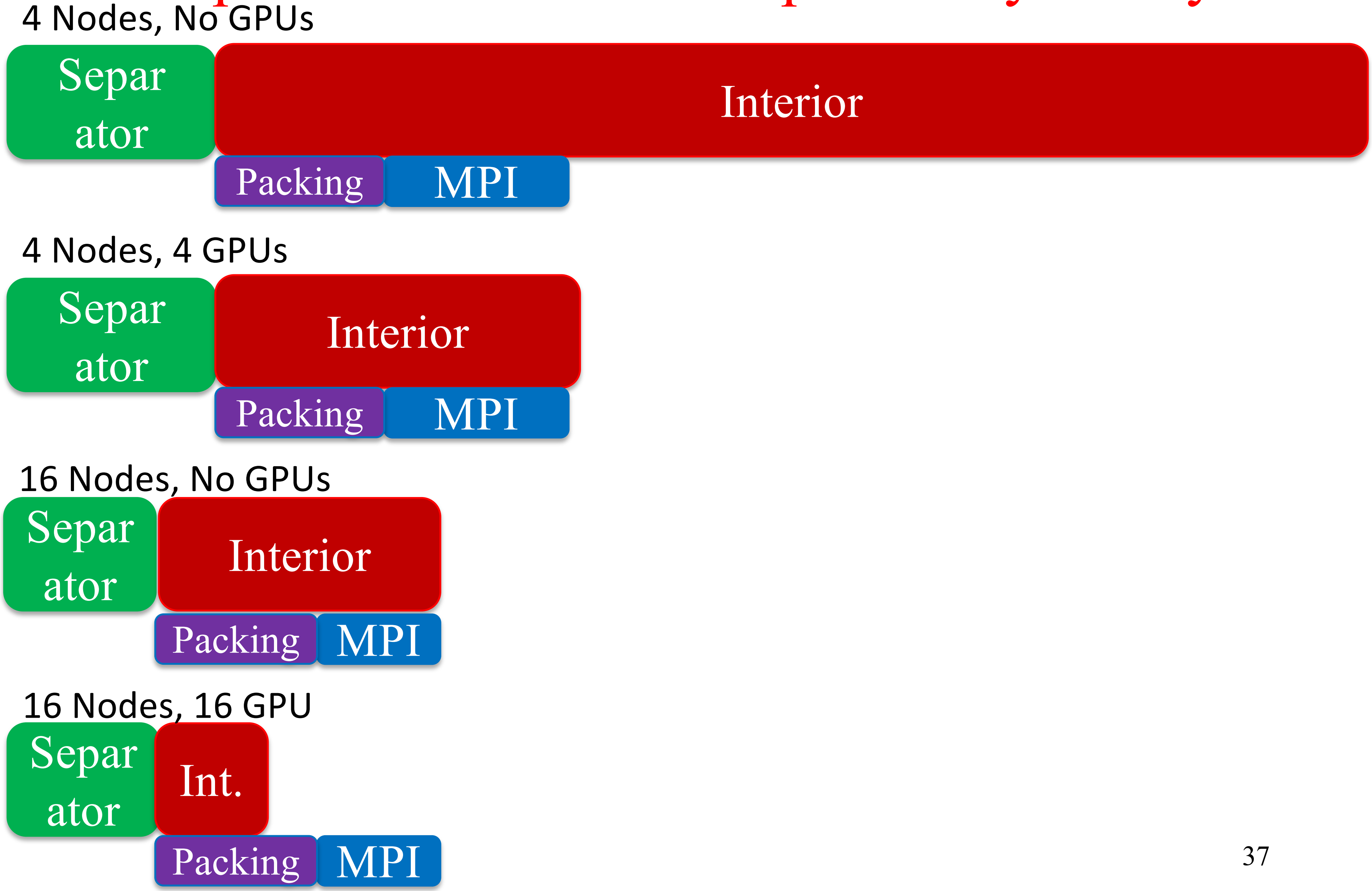
Separator

Interior

Packing

MPI

Separators Become Comparatively Heavy



4 Nodes, No GPUs

Separ ator

Interior

Packing

MPI

4 Nodes, 4 GPUs

Separ ator

Interior

Packing

MPI

16 Nodes, No GPUs

Separ ator

Interior

Packing

MPI

16 Nodes, 16 GPU

Separ ator

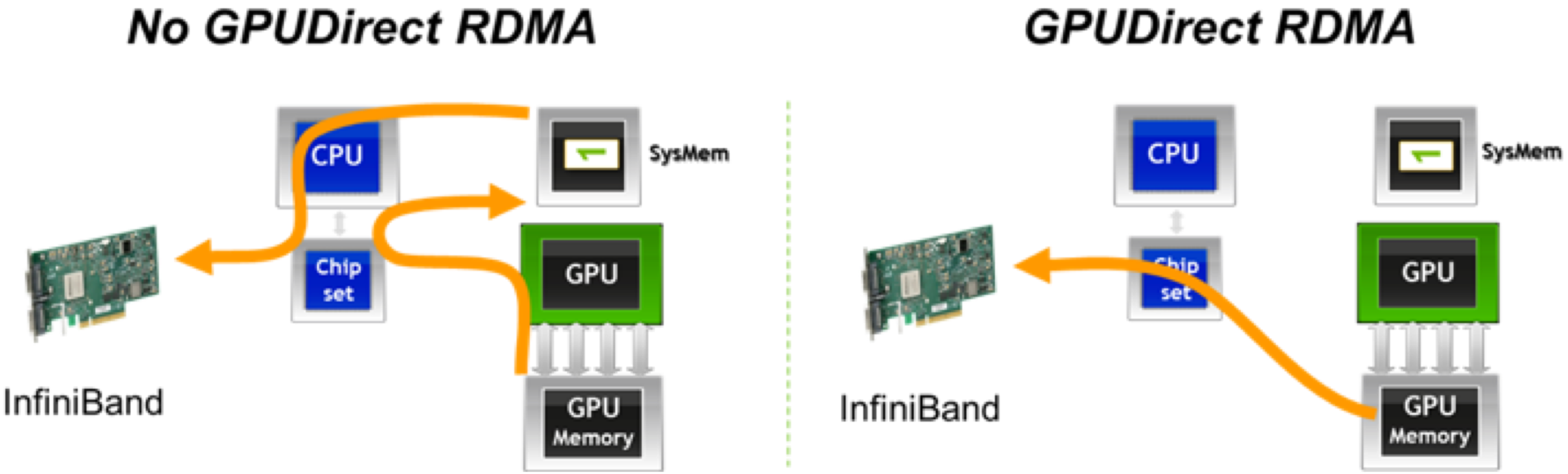
Int.

Packing

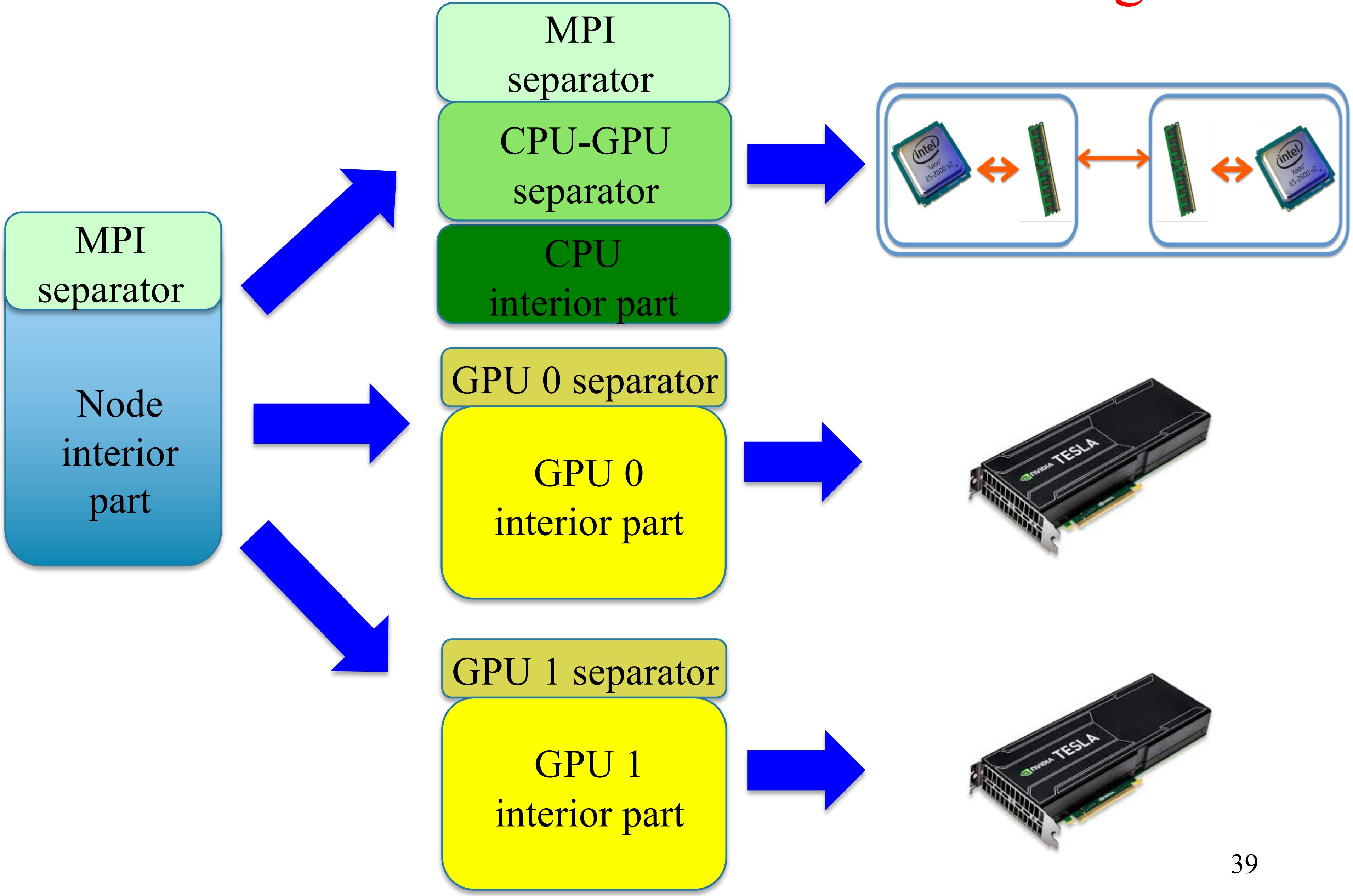
MPI

CUDA-aware MPI: RDMA Transfers

- GPUDirect RDMA allows send/recv directly from the GPU
- Subject to limitations due to GPU memory



Alternative: Hierarchical Partitioning



MPI separator

Node interior part

MPI separator

CPU-GPU separator

CPU interior part

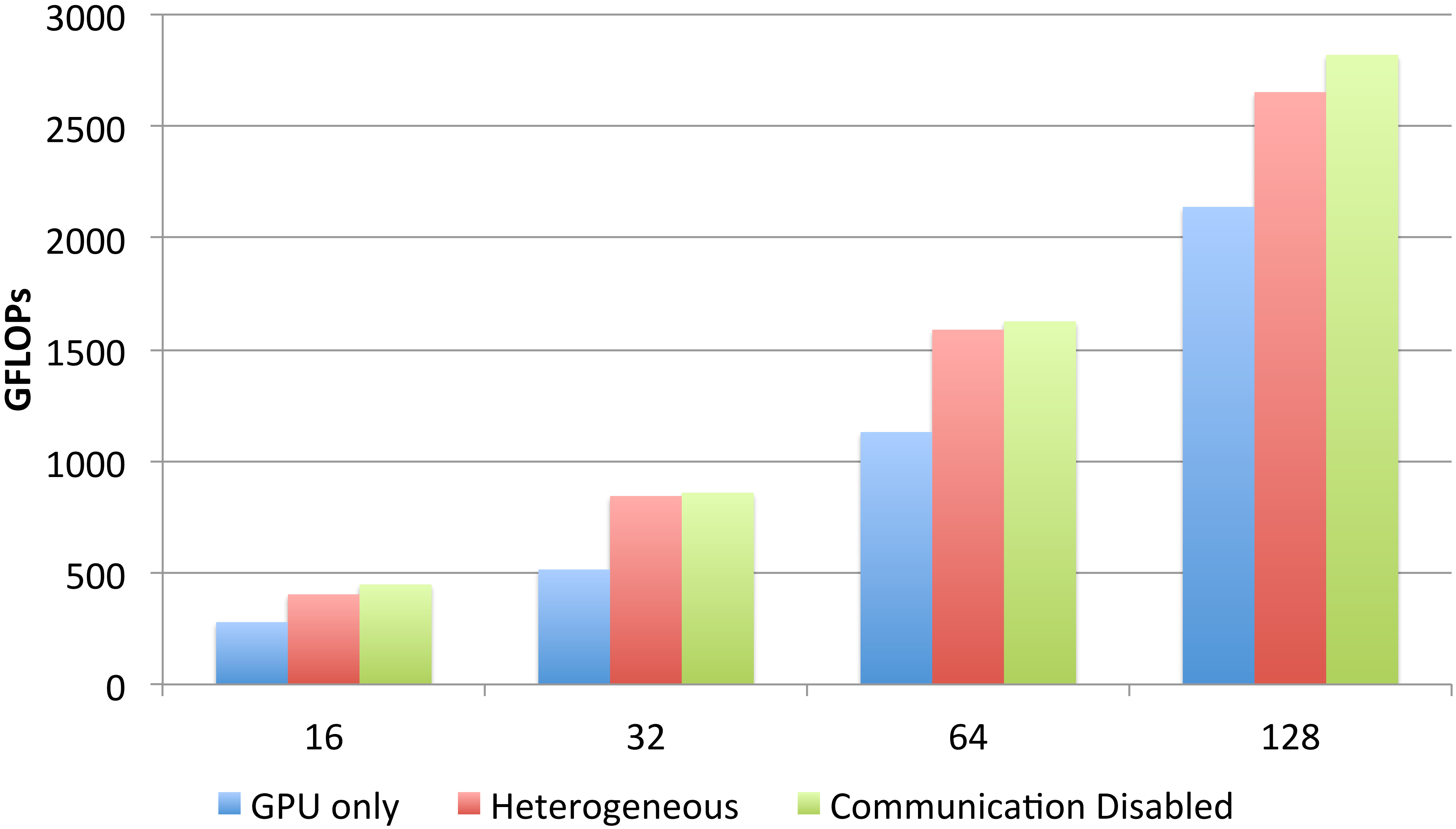
GPU 0 separator

GPU 0 interior part

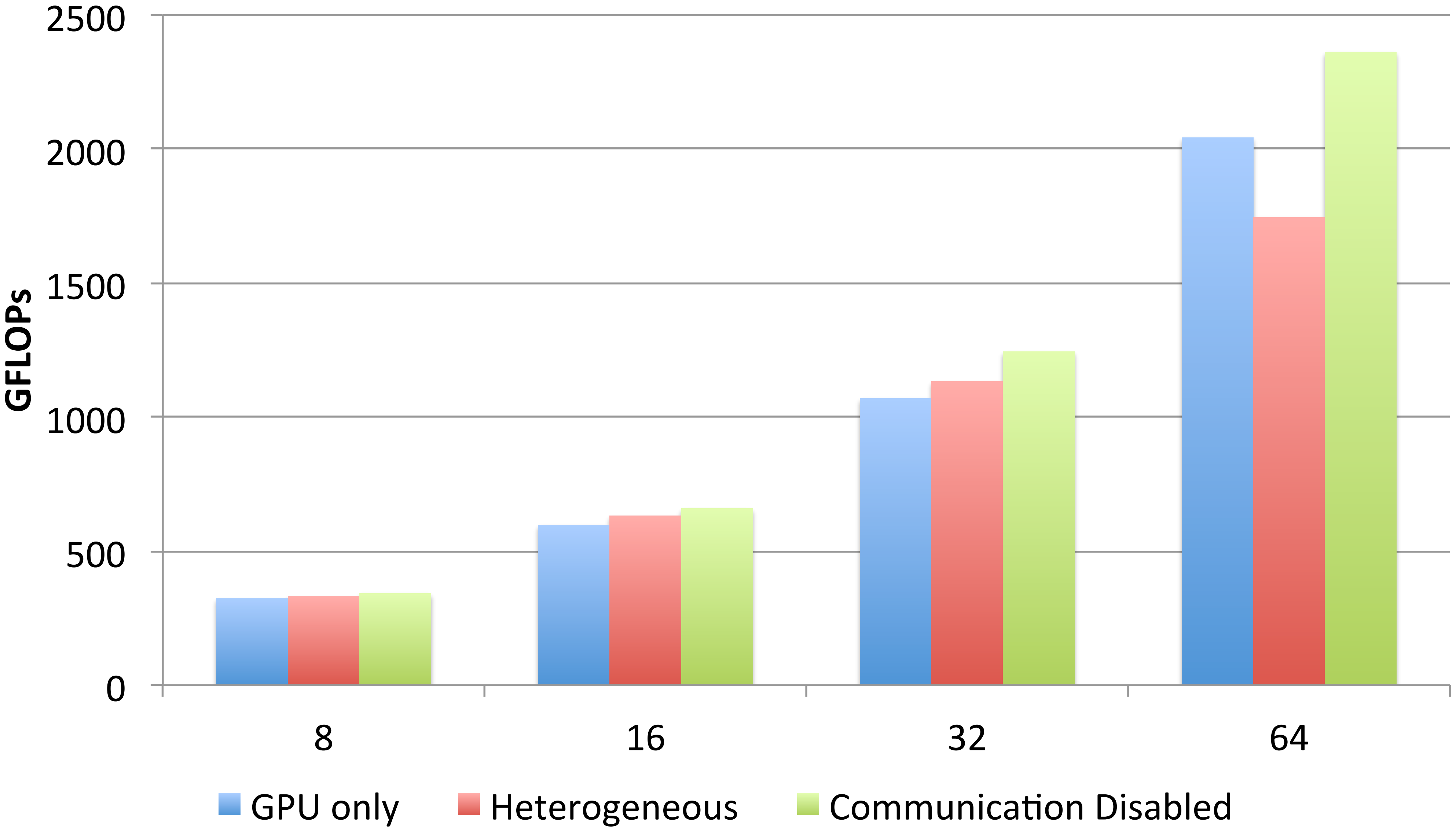
GPU 1 separator

GPU 1 interior part

Stampede, 1 GPU, strong CPUs



Wilkes 2 GPUs, weak CPUs



Summary on Multi Node Multi GPU

- Multiple GPU is comparatively easy
- GPU-heavy machines give lots of power easily
- NCCL collectives still missing
- MPI can be used for single and multiple node
- Collectives are a good way of organizing communication
- Scalable multi-node codes are hard
- Distributing irregular problems on Supercomputers is even harder

References

Langguth, J., Sourouri, M., Lines, G. T., Baden, S. B., & Cai, X. (2015). Scalable heterogeneous CPU-GPU computations for unstructured tetrahedral meshes. *IEEE Micro*, *35*(4), 6-15.

Credit: Lecture contains NVIDIA material available at https://developer.nvidia.com/cuda-zone
Image source: wikipedia.org, top500.org, ornl.gov, mvapich.cse..ohio-state.edu
Contains material from ACACES 2018 summer school, originally designed by Scott Baden