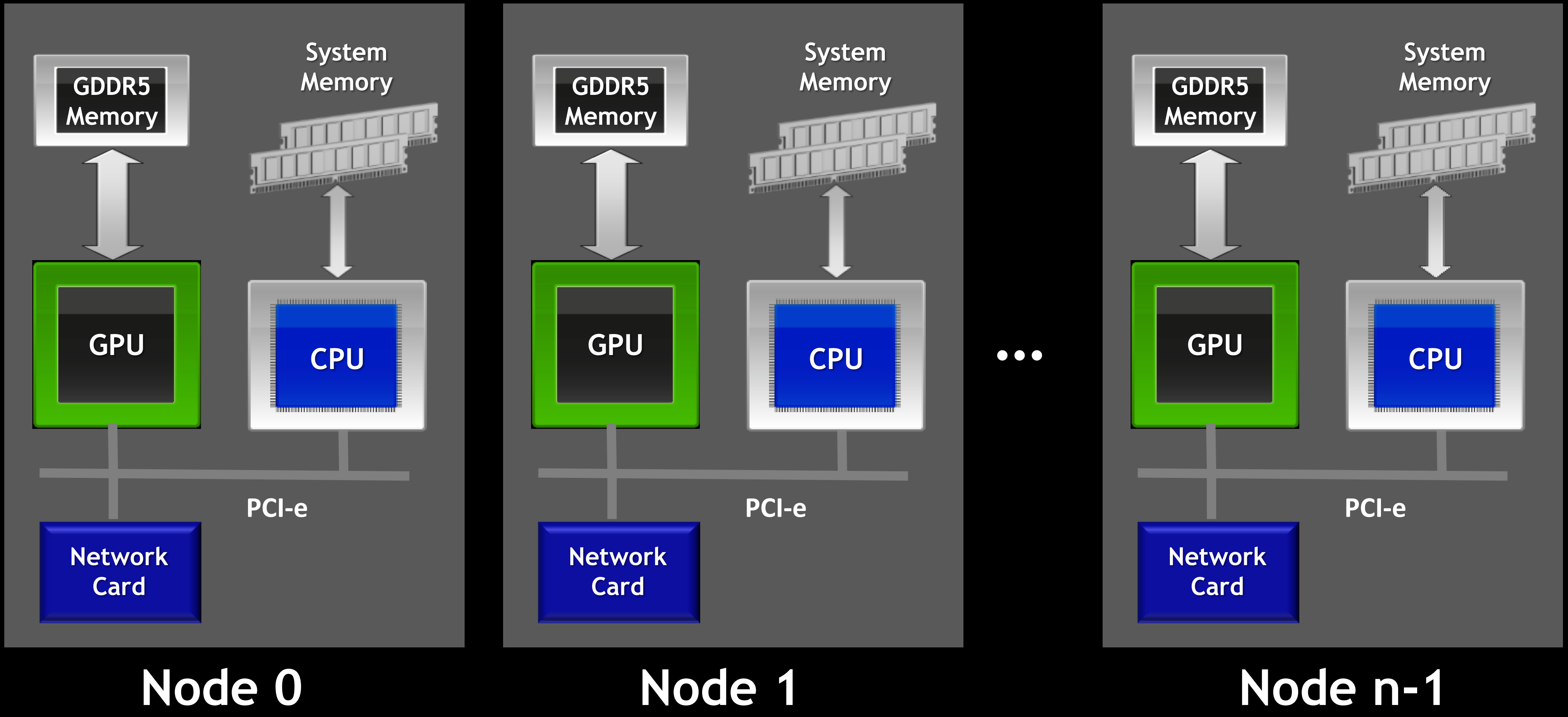
GPU TECHNOLOGY CONFERENCE

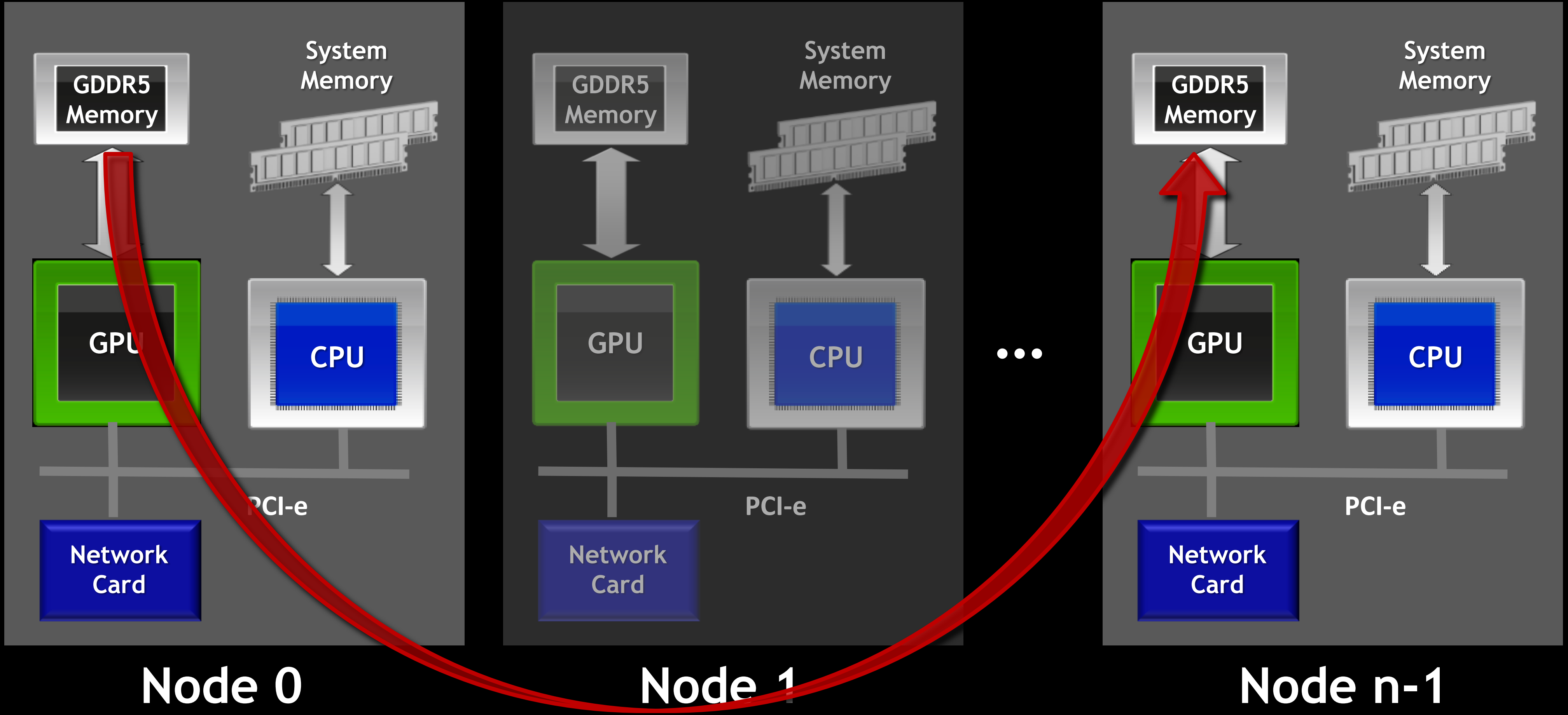
Introduction to CUDA-aware MPI and NVIDIA GPUDirect™

Jiri Kraus
Developer Technology HPC, NVIDIA

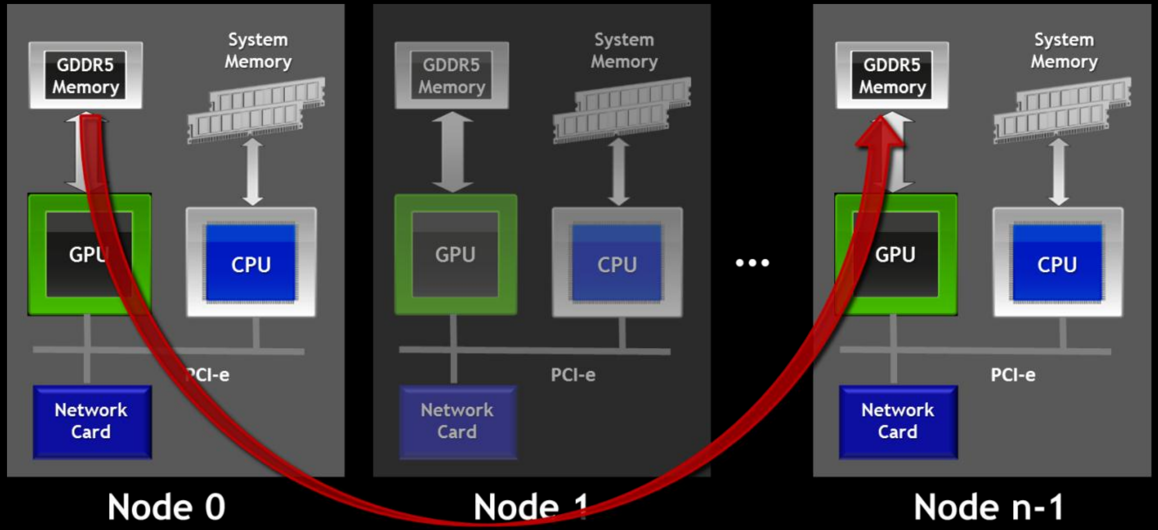
MPI+CUDA


GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 0
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 1
…
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node n-1

MPI+CUDA


GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 0
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 1
…
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node n-1

MPI+CUDA



```
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,n-1,tag,MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

Outline

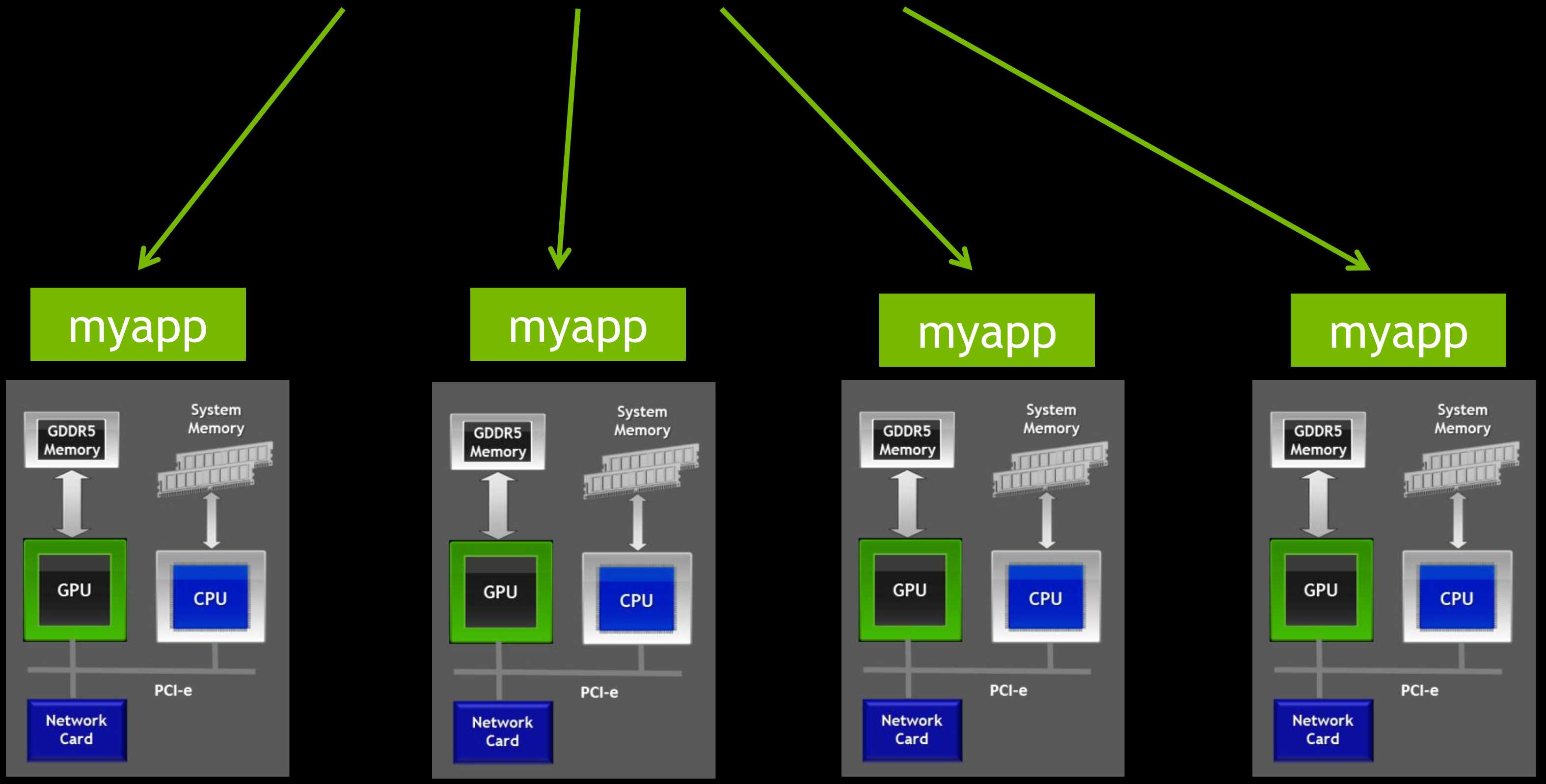
- Short Introduction to MPI
- Unified Virtual Addressing and GPUDirect
- How CUDA-aware MPI works
- Performance Results
- Wrap-up and conclusions

Message Passing Interface - MPI

- Standard to exchange data between processes via messages
  - Defines API to exchanges messages
    - Pt. 2 Pt.: e.g. MPI_Send, MPI_Recv
    - Collectives, e.g. MPI_Reduce
- Multiple implementations (open source and commercial)
  - Binding for C/C++, Fortran, Python, …

MPI - How to launch a MPI program

```
mpirun –np 4 ./myapp <args>
```



myapp

myapp

myapp

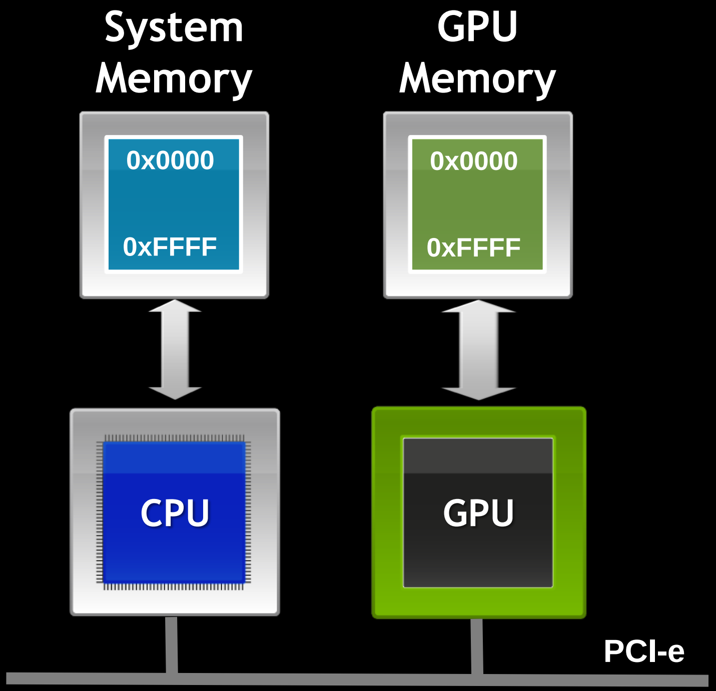
myapp

MPI - A minimal program

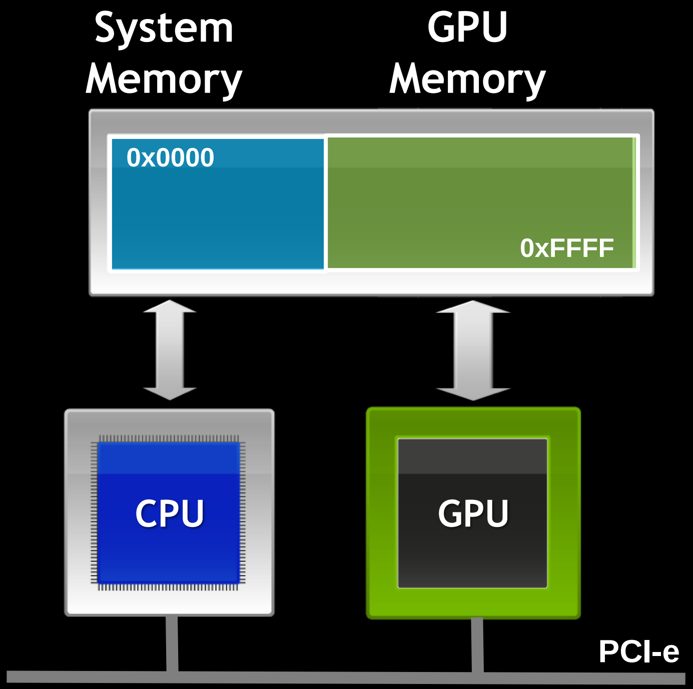
```
#include <mpi.h>
int main(int argc, char *argv[]) {
    int myrank;
    /* Initialize the MPI library */
    MPI_Init(&argc,&argv);
    /* Determine the calling process rank */
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    /* Call MPI routines like MPI_Send, MPI_Recv, ... */
    /* Shutdown MPI library */
    MPI_Finalize();
    return 0;
}
```

Unified Virtual Addressing

No UVA: Multiple Memory Spaces

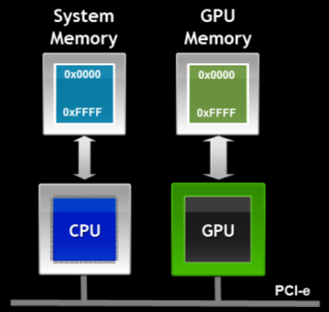


UVA : Single Address Space

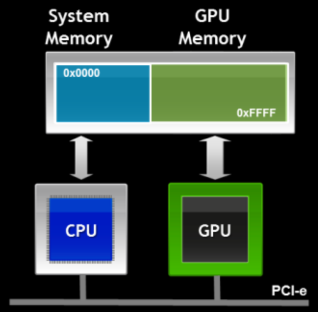


Unified Virtual Addressing

No UVA: Multiple Memory Spaces

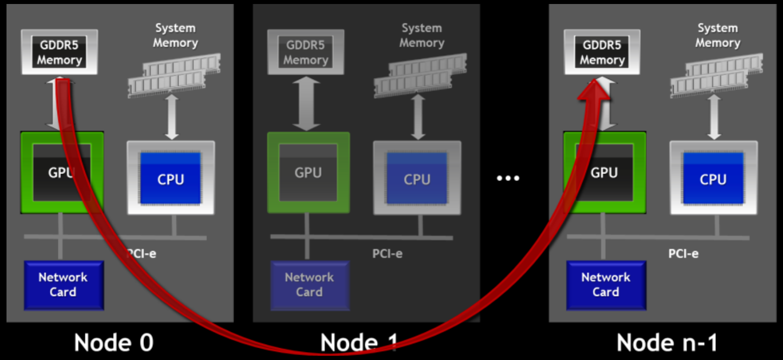


UVA : Single Address Space



- One address space for all CPU and GPU memory
  - Determine physical memory location from a pointer value
  - Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)
- Supported on devices with compute capability 2.0 for
  - 64-bit applications on Linux and on Windows also TCC mode

MPI+CUDA



With UVA and CUDA-aware MPI

```
//MPI rank 0
MPI_Send(s_buf_d,size,…);


//MPI rank n-1
MPI_Recv(r_buf_d,size,…);
```

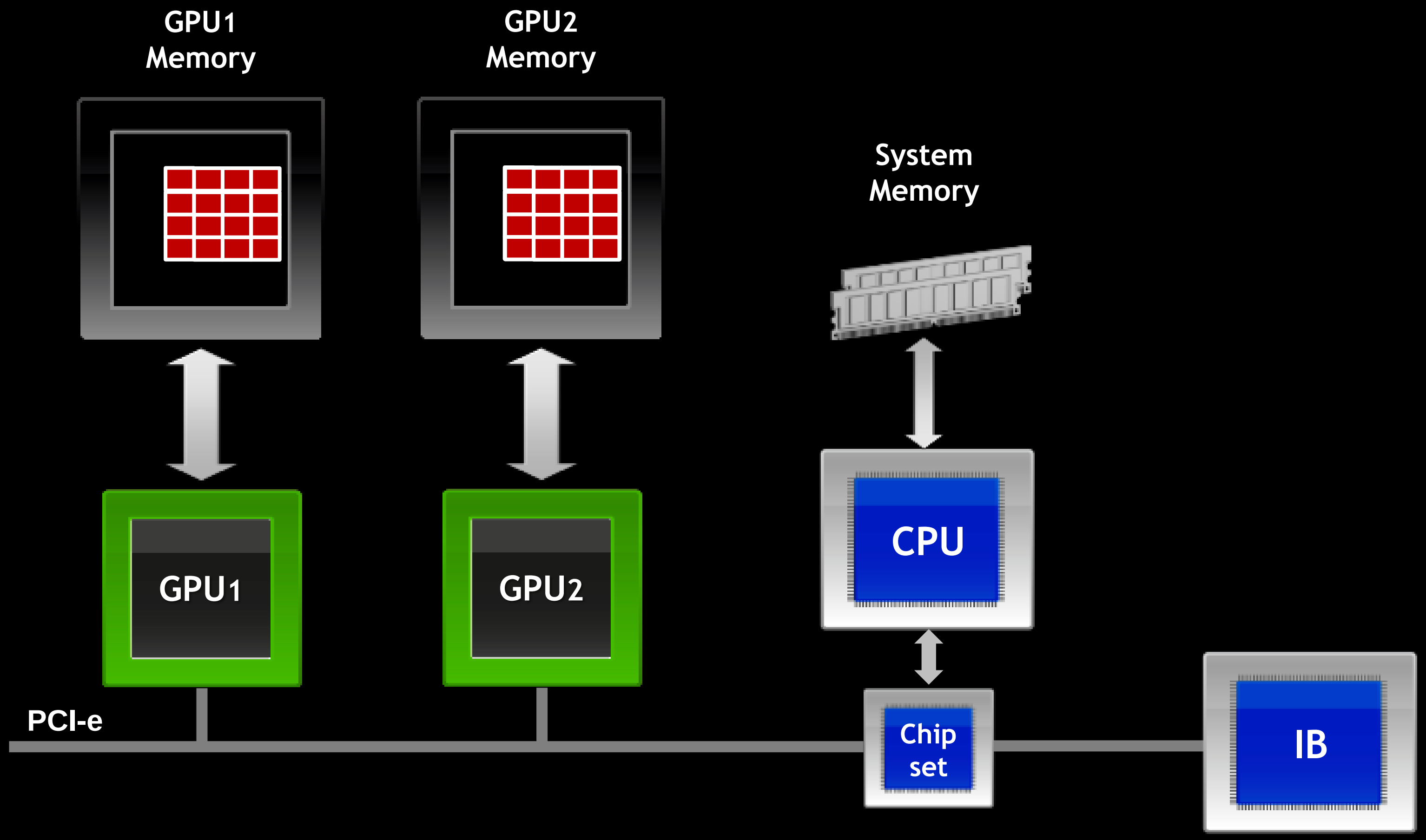
No UVA and regular MPI

```
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,…);
MPI_Send(s_buf_h,size,…);

//MPI rank n-1
MPI_Recv(r_buf_h,size,…);
cudaMemcpy(r_buf_d,r_buf_h,size,…);
```

CUDA-aware MPI makes MPI+CUDA easier.

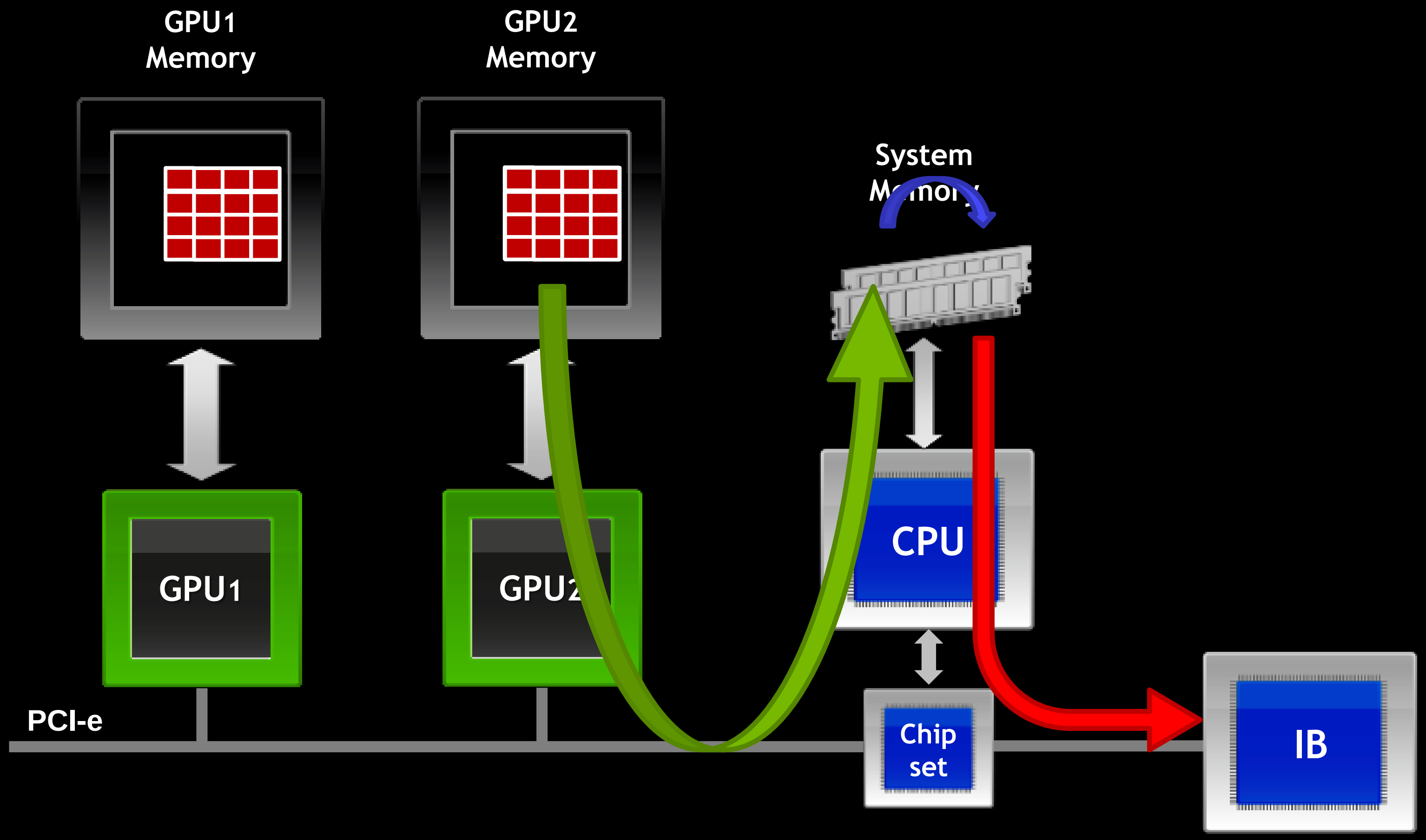
NVIDIA GPUDirect^{TM}

Accelerated communication with network & storage devices



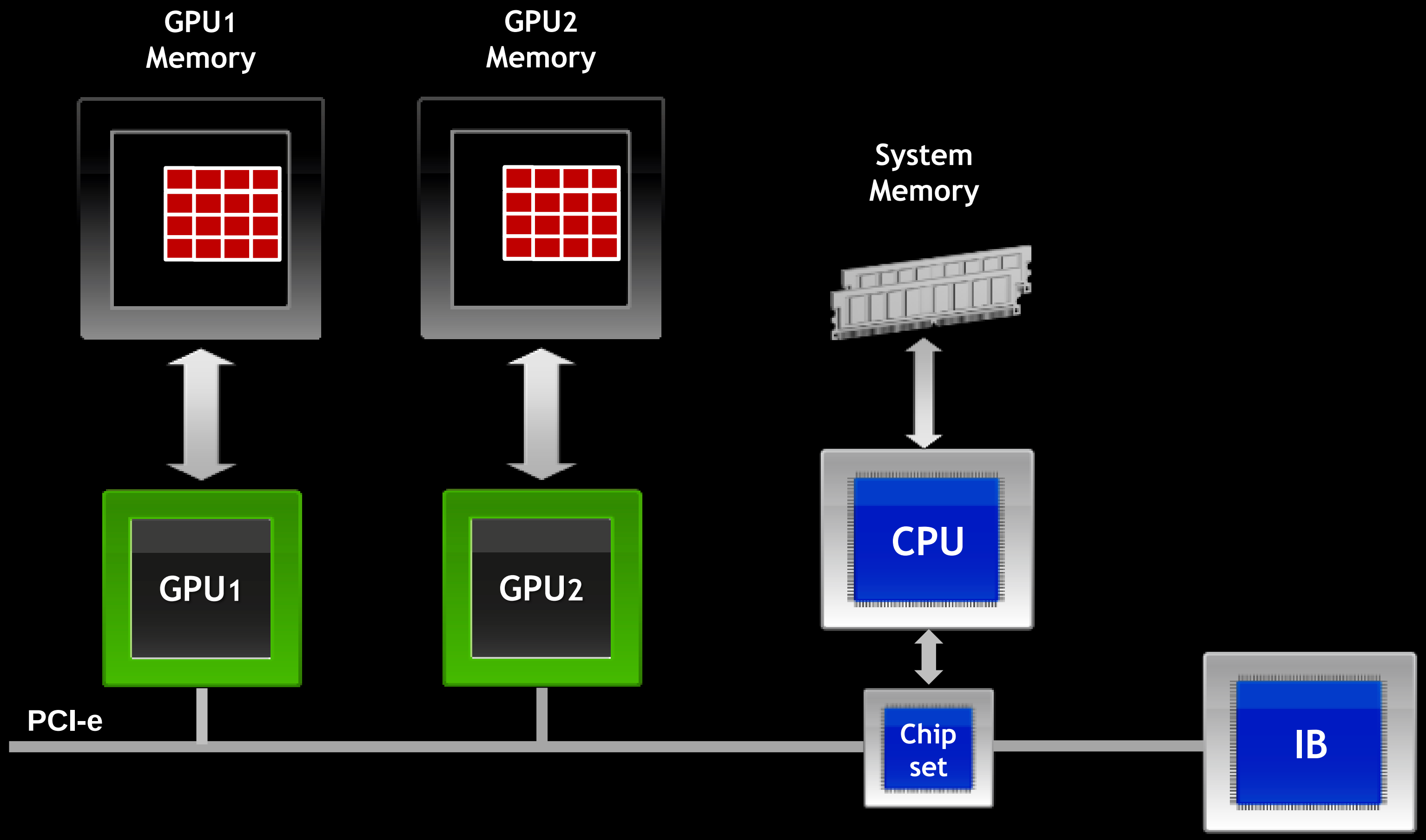
NVIDIA GPUDirect™

Accelerated communication with network & storage devices



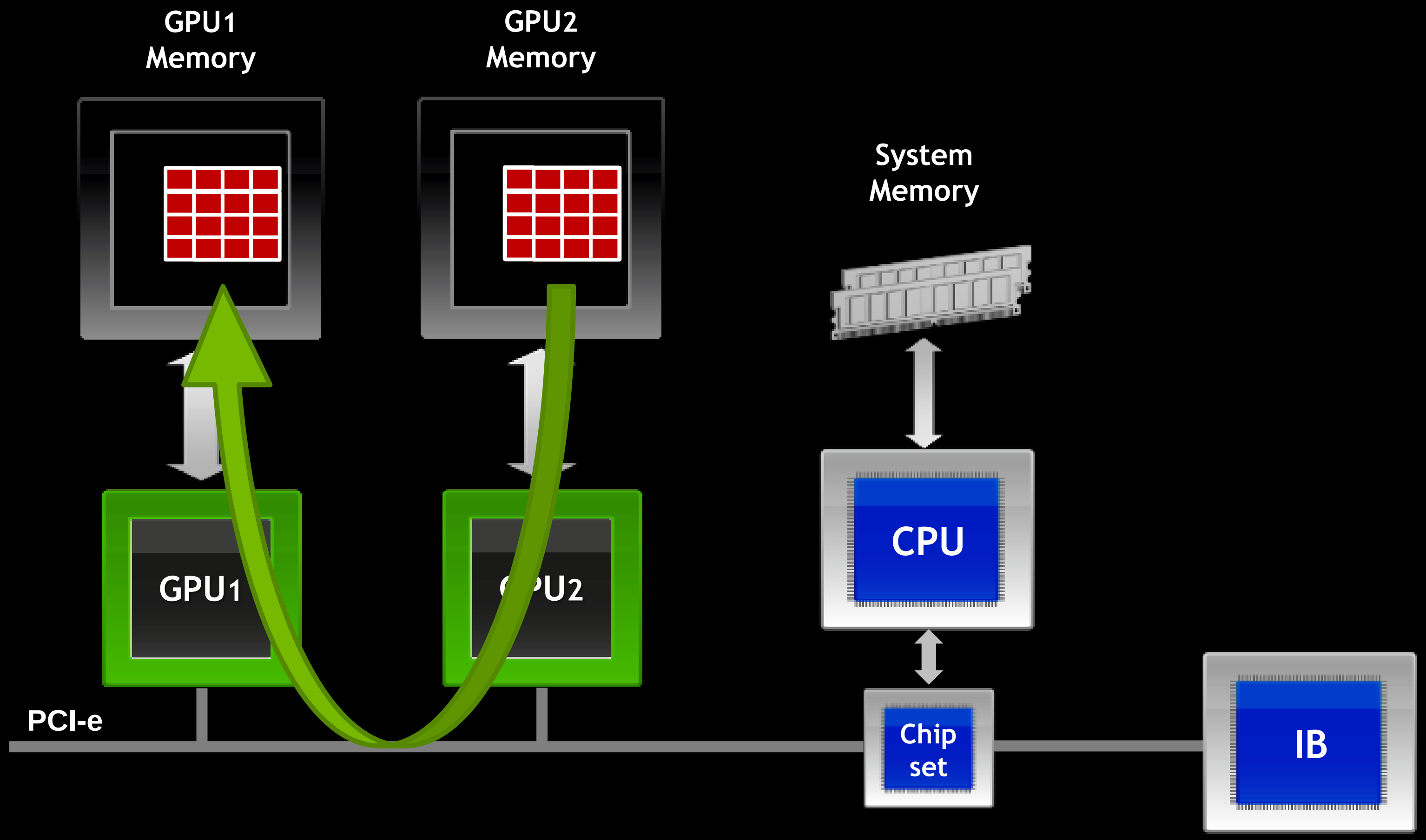
NVIDIA GPUDirect^{TM}

Peer to Peer Transfers



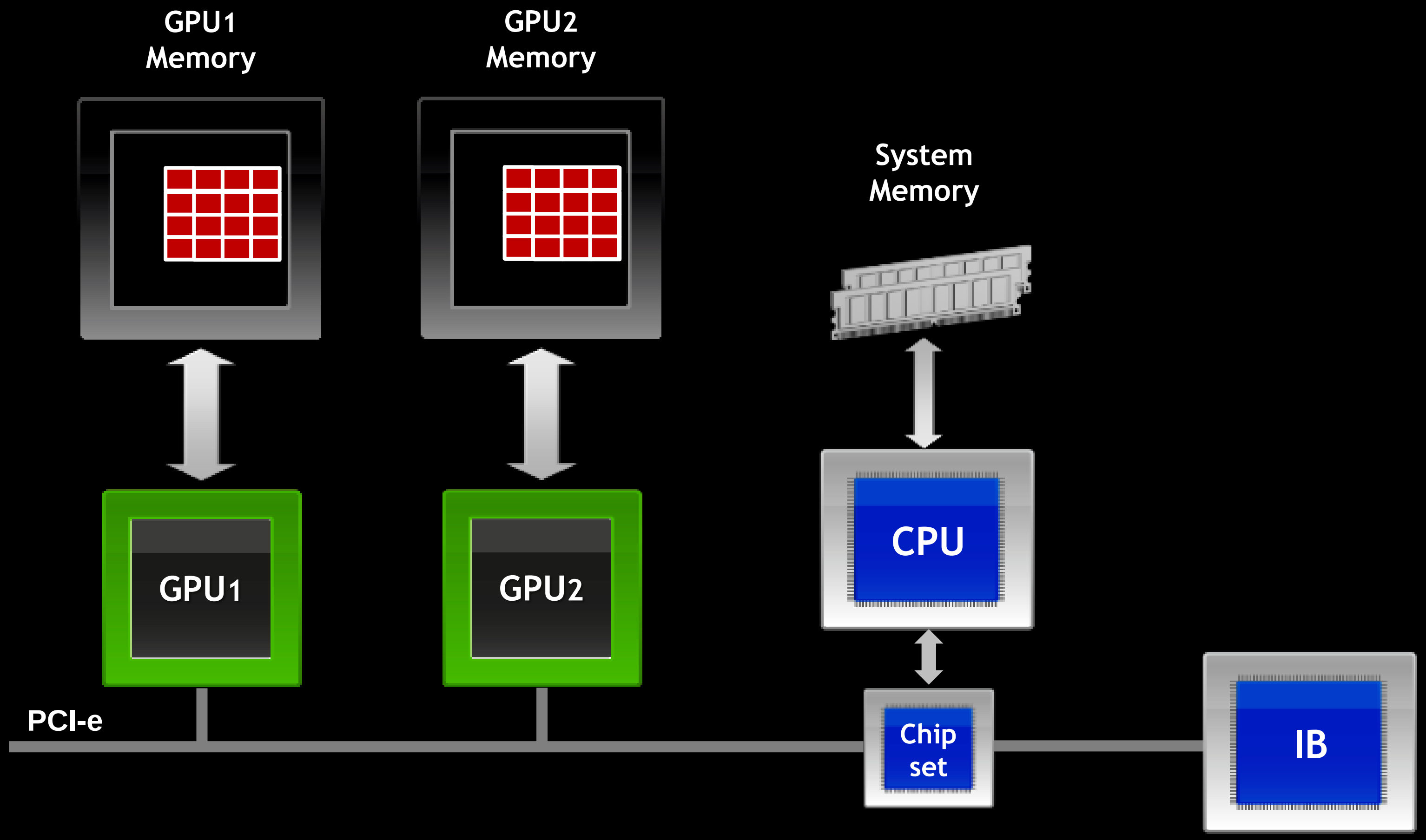
NVIDIA GPUDirect™

Peer to Peer Transfers



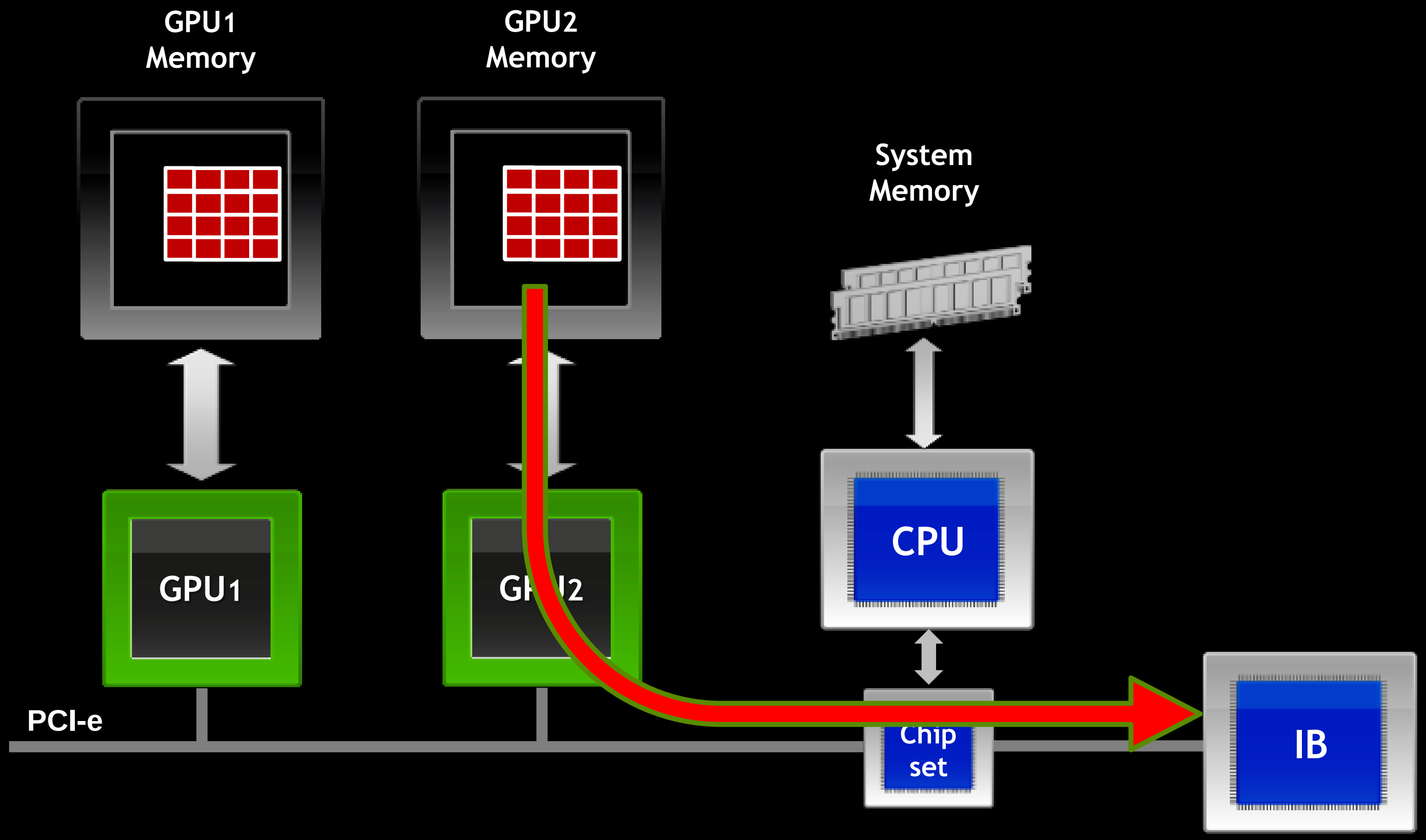
NVIDIA GPUDirect™

Support for RDMA



NVIDIA GPUDirect^{TM}

Support for RDMA



CUDA-Aware MPI

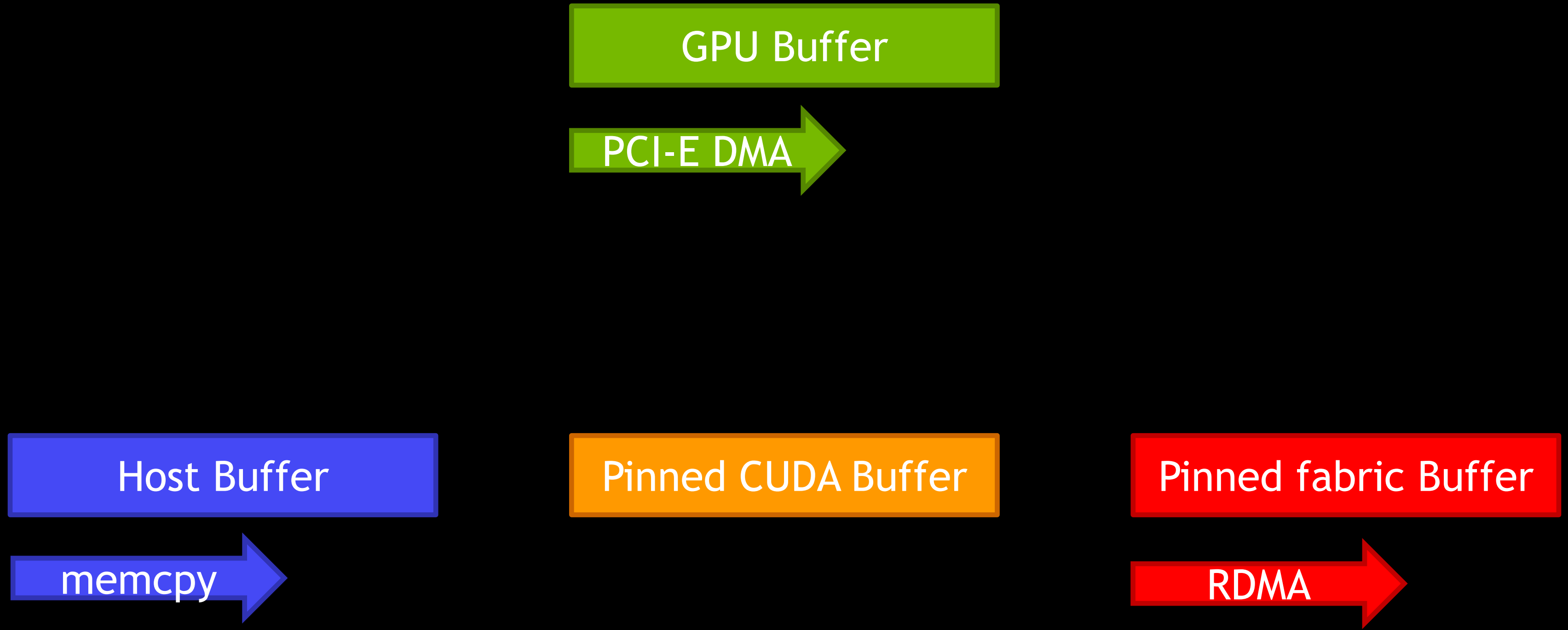
Example:

MPI Rank 0 MPI_Send from GPU Buffer

MPI Rank 1 MPI_Recv to GPU Buffer

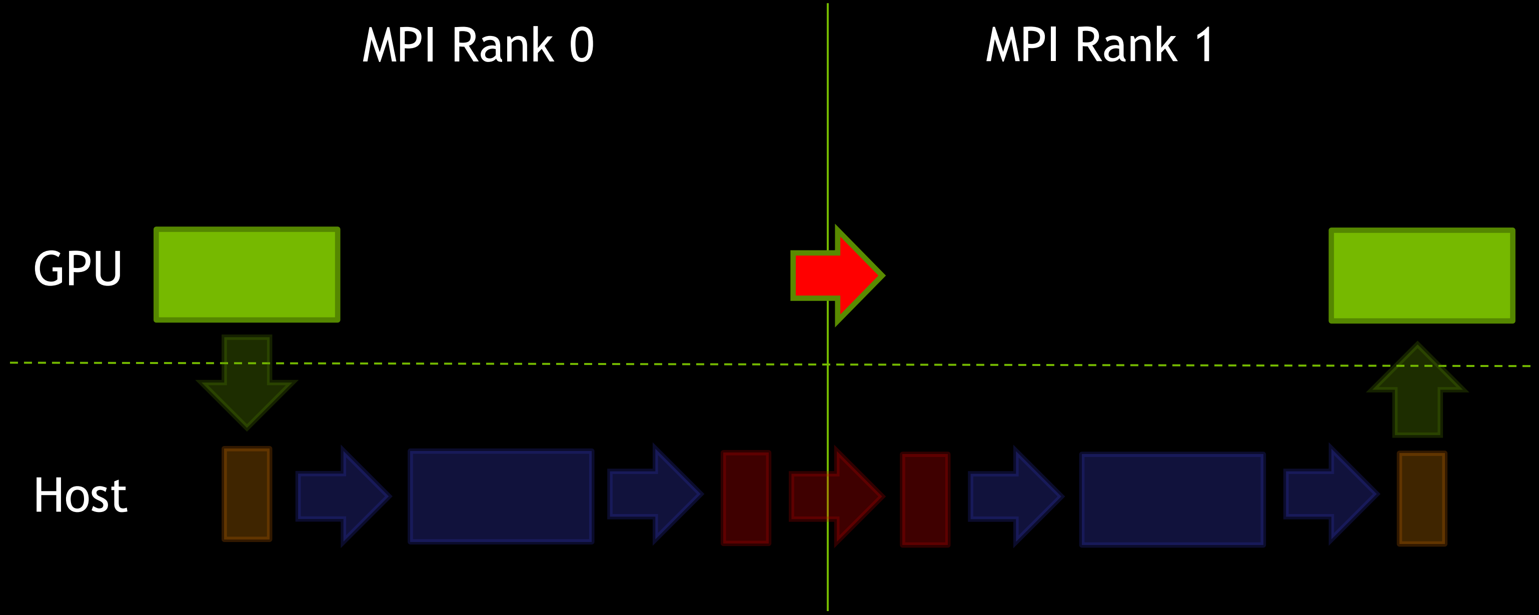
- Show how CUDA+MPI works in principle
  - Depending on the MPI implementation, message size, system setup, … situation might be different
- Two GPUs in two nodes

CUDA-Aware MPI



MPI GPU to Remote GPU

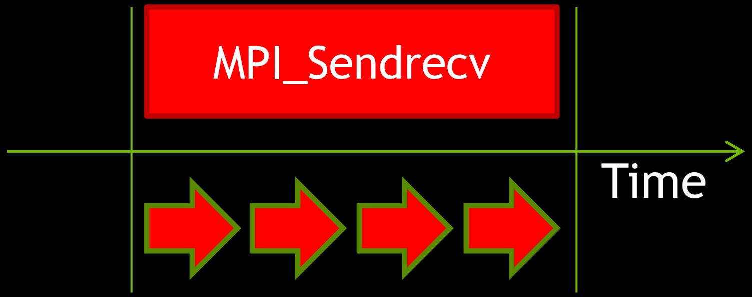
GPUDirect Support for RDMA



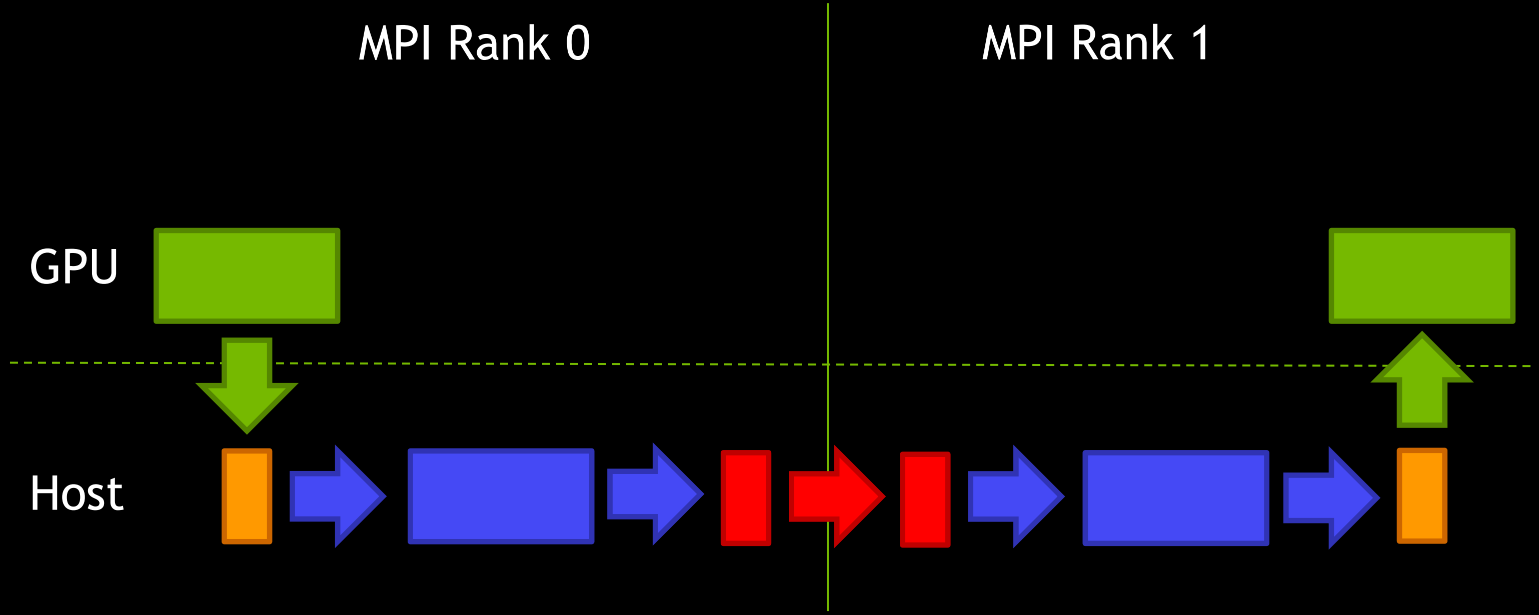
```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

MPI GPU to Remote GPU

GPUDirect Support for RDMA

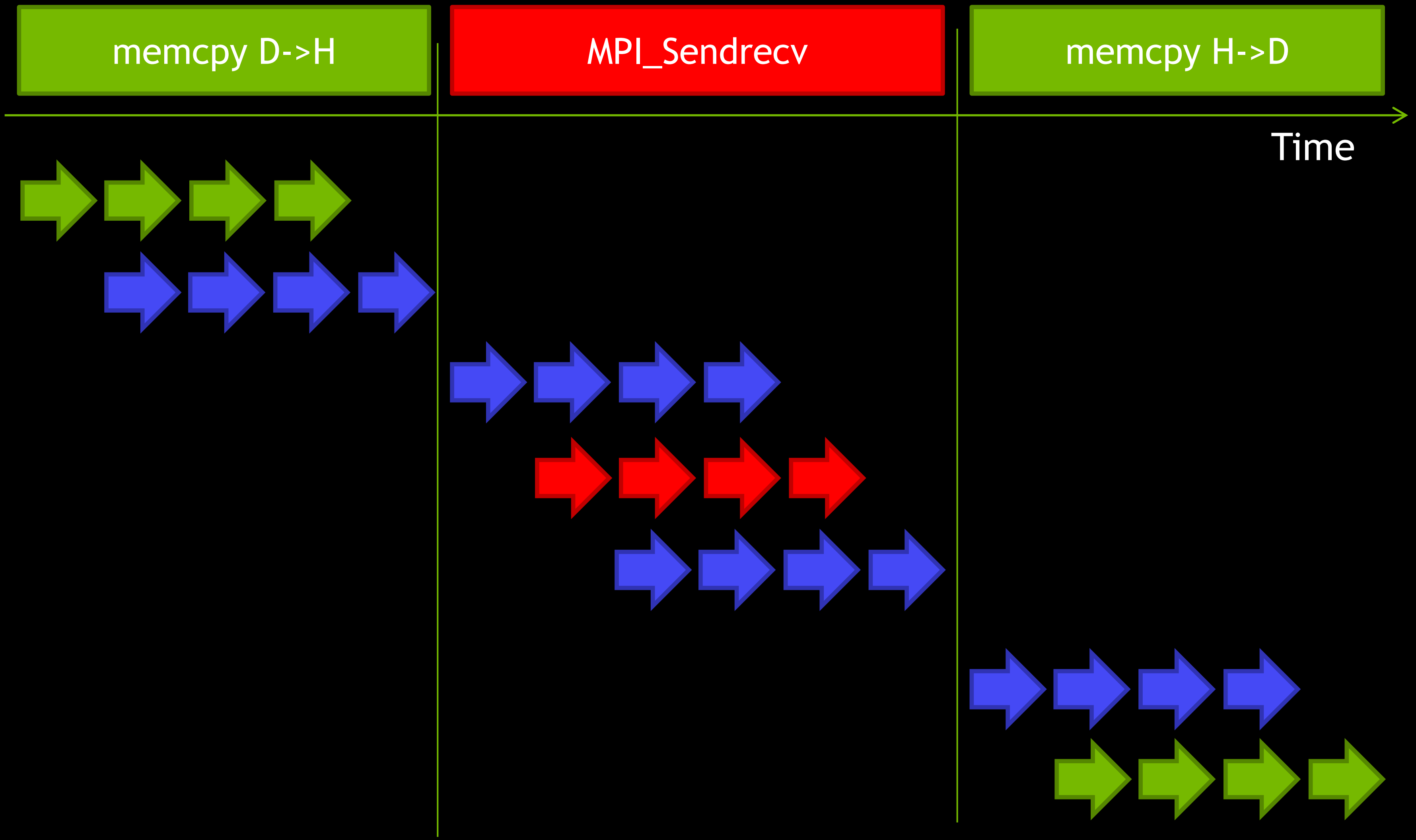


Regular MPI GPU to Remote GPU



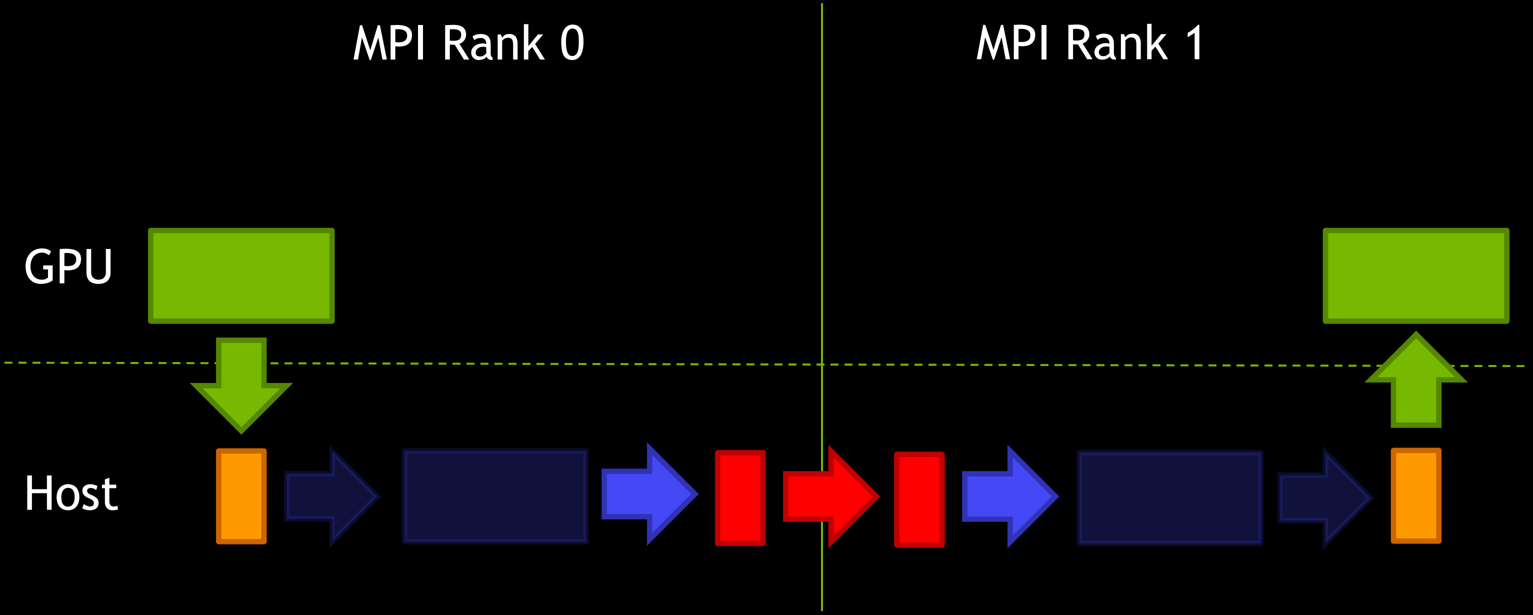
```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_h,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

Regular MPI GPU to Remote GPU



MPI GPU to Remote GPU

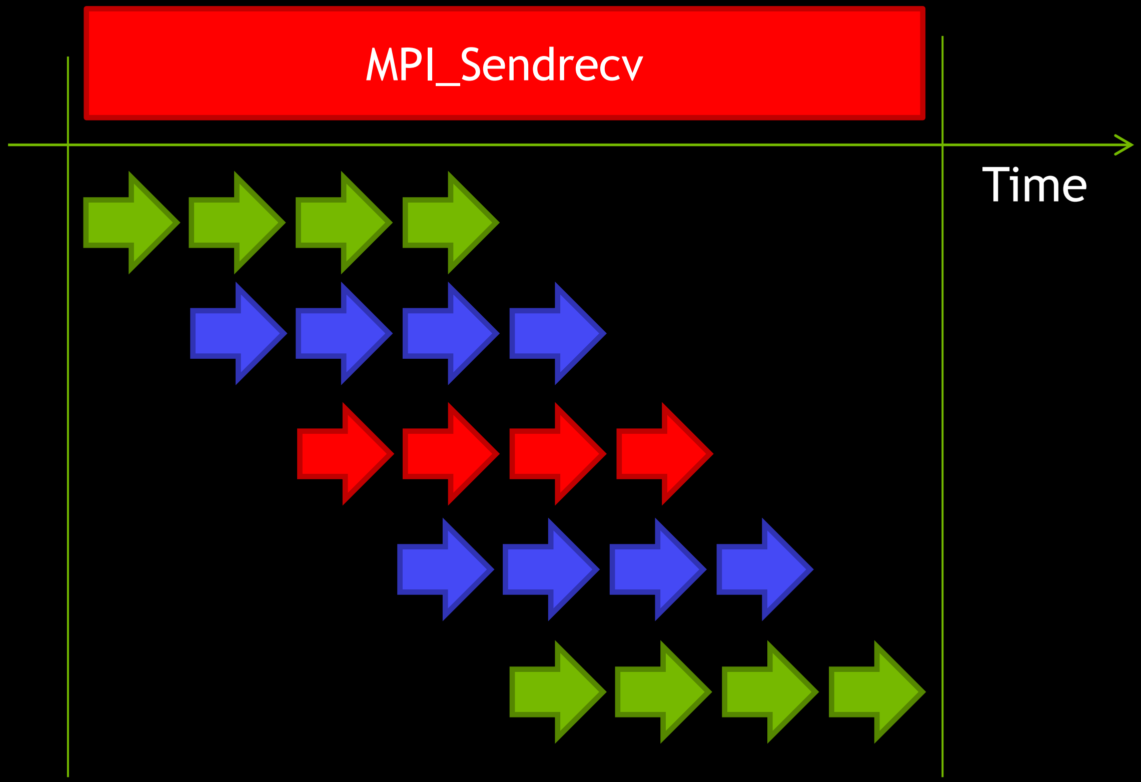
Without GPUDirect



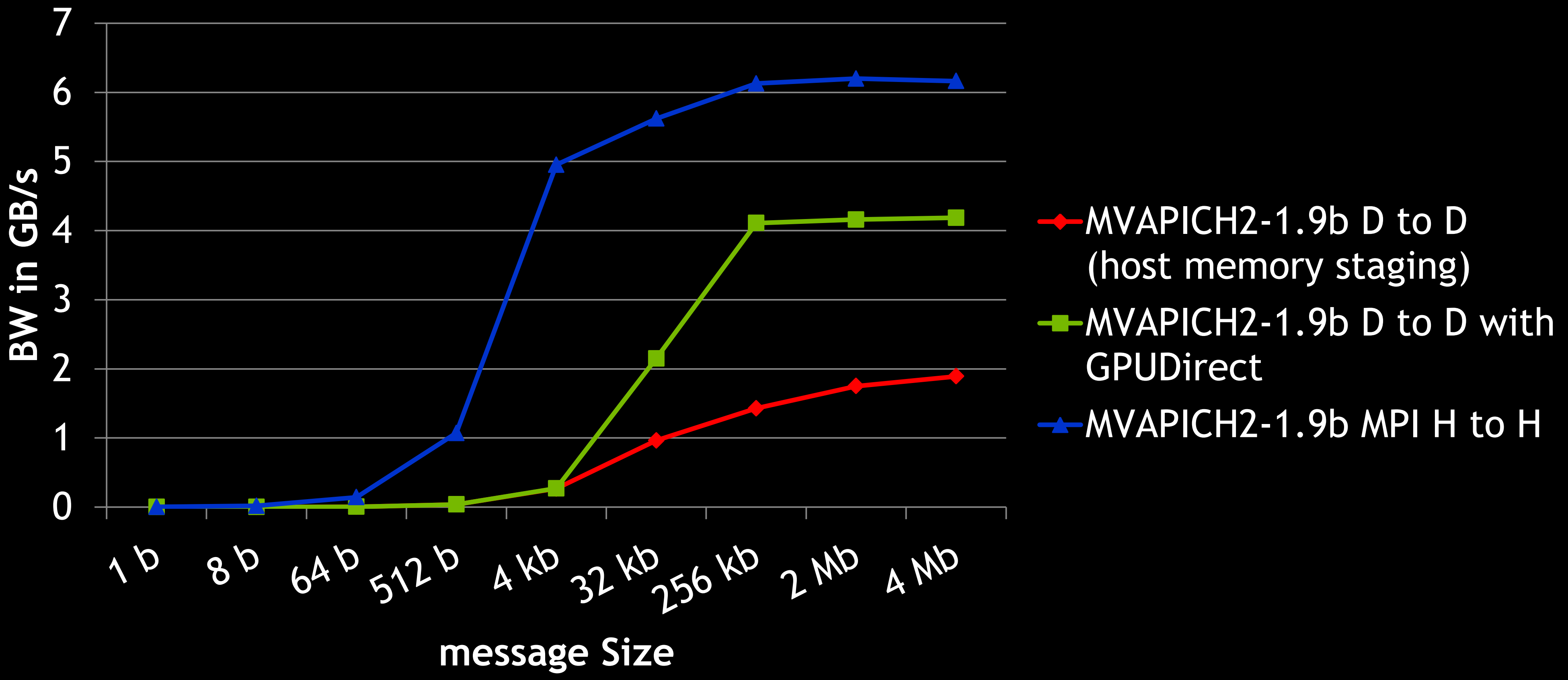
```
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_h,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

MPI GPU to Remote GPU

Without GPUDirect



Performance Results two Nodes



Latency (1 byte) 19.00 µs 18.34 µs 1.11 µs

Example: Jacobi

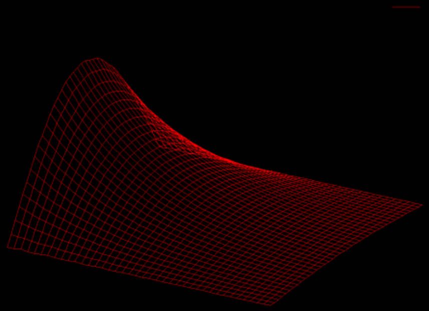
- Solves the 2D-Poisson equation on a rectangle

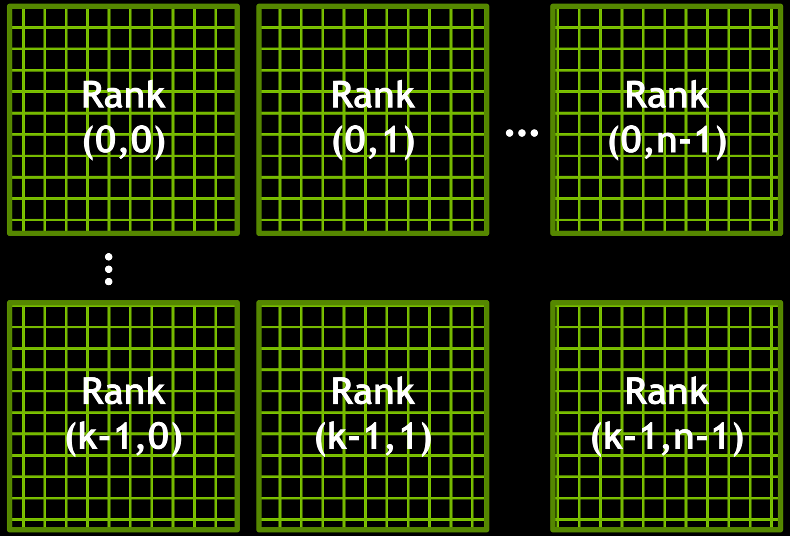
$$\Delta \boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{0}\ \forall\ (\boldsymbol{x},\boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

  - Dirichlet boundary conditions

$$\boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) \in \boldsymbol{\delta}\Omega$$

- 2D domain decomposition with n x k domains





Rank (0,0)

Rank (0,1)

...

Rank (0,n-1)

⋮

Rank (k-1,0)

Rank (k-1,1)

Rank (k-1,n-1)

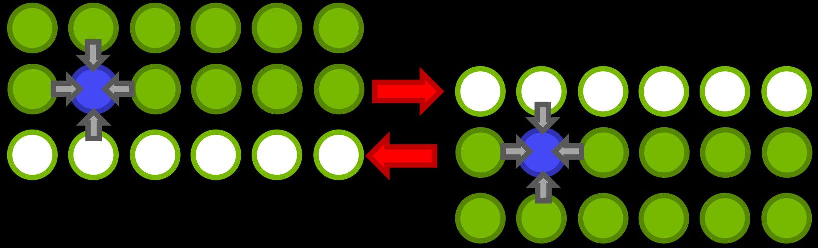
Example: Jacobi

While not converged

- Do Jacobi step:

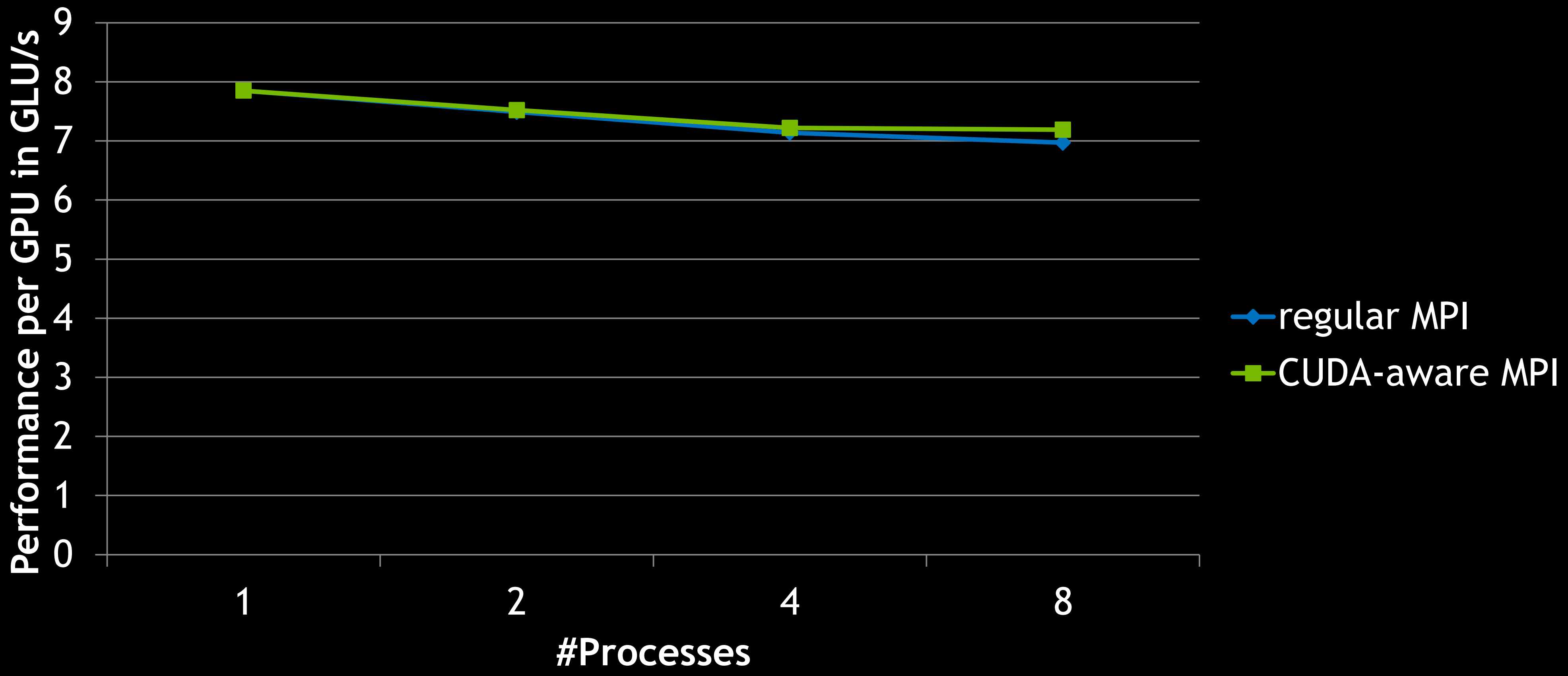
```
for (int i=1; i < n-1; i++) for (int j=1; j < m-1; j++)
    u_new[i][j] = 0.0f - 0.25f*(u[i-1][j] + u[i+1][j]
                              + u[i][j-1] + u[i][j+1])
```

- Exchange halo with 2 – 4 neigbours
- Swap `u_new` and `u`
- Next iteration



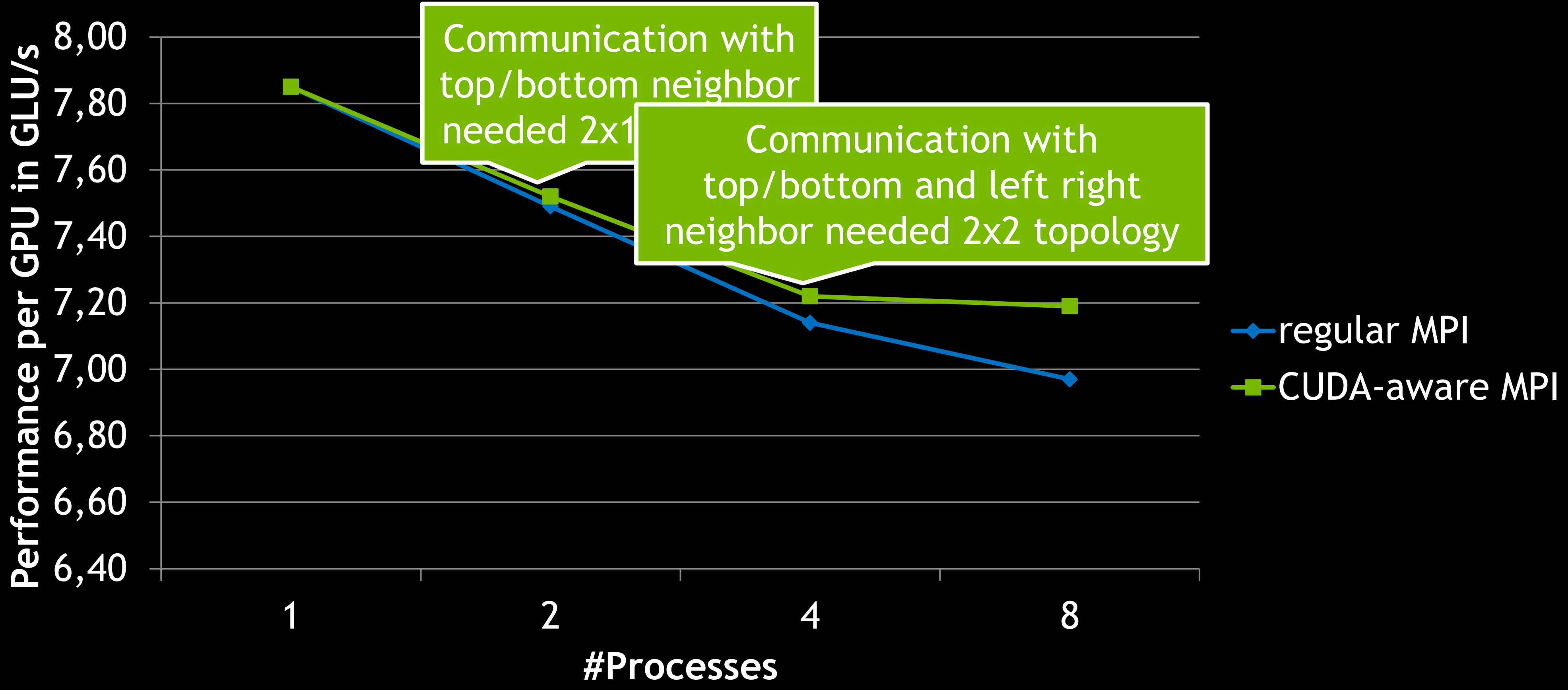
Jacobi Results (1000 steps)

weak scaling 4k x 4k per process



Jacobi Results (1000 steps)

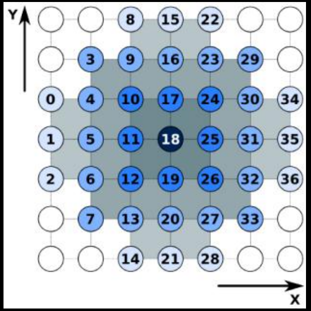
weak scaling 4k x 4k per process

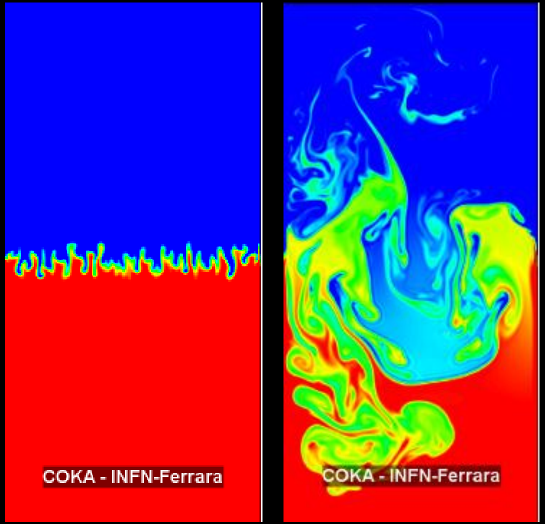


Communication with top/bottom neighbor needed 2x1

Communication with top/bottom and left right neighbor needed 2x2 topology

LBM D2Q37





Lattice Boltzmann Method (LBM)

- D2Q37 Model
- Application developed at U Rome Tore Vergata/INFN, U Ferrara/INFN, TU Eindhoven
- Reproduce dynamics of fluid by simulating virtual particles which collide and propagate, e.g. solve the Rayleigh-Taylor instability
- Simulation of large problems requires double precision and many GPUs

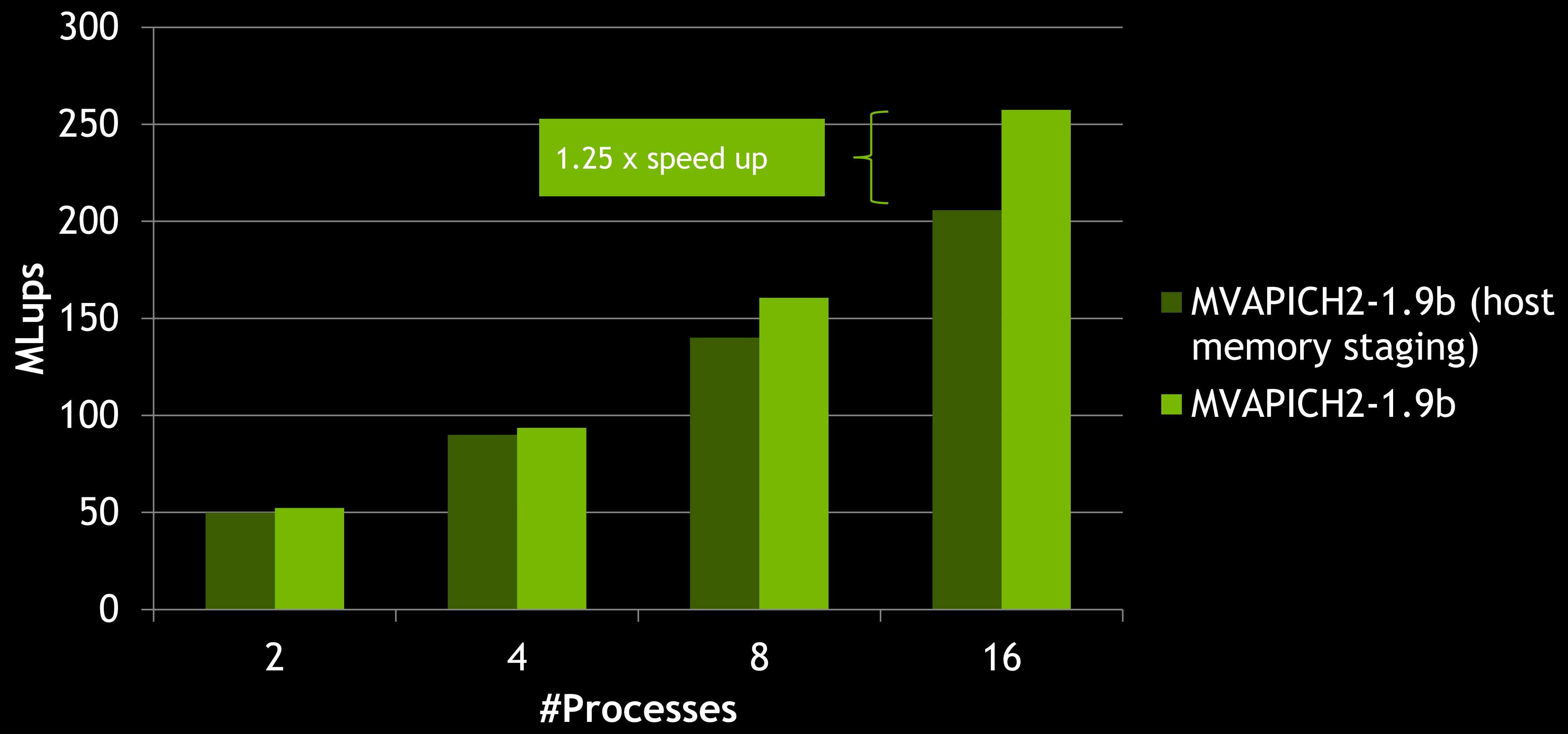
Implementation and Benchmarks:
F. Schifano (U Ferrara)

UNIFE

JÜLICH
APPLICATION LAB
NVIDIA

LBM D2Q37 Results

strong scaling on 8192x1024 cells



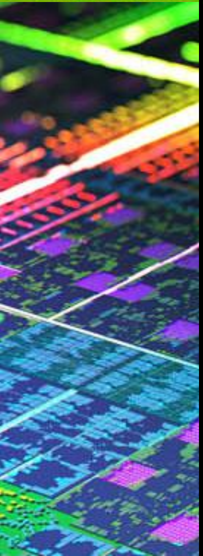
CUDA-Aware MPI Implementations

Integrated Support for GPU Computing

- MVAPICH2 1.8/1.9b
  - http://mvapich.cse.ohio-state.edu/overview/mvapich2/
- OpenMPI 1.7 (beta)
  - http://www.open-mpi.org/
- CRAY MPI (MPT 5.6.2)
- IBM Platform MPI (8.3)

CUDA-Aware Caveats

- cudaSetDevice needs to be called before MPI_Init
- MPI Environment vars. can be used to set GPU affinity
  - MVAPICH2: MV2_COMM_WORLD_LOCAL_RANK
  - OpenMPI: OMPI_COMM_WORLD_LOCAL_RANK
- MV2_USE_CUDA needs to be set for MVAPICH
- MPICH_RDMA_ENABLED_CUDA for MPT on Cray
- PMPI_GPU_AWARE for Platform MPI
- Lib needs to be build with CUDA-awarenes enabled



CUDA-Aware MPI + OpenACC

- To use CUDA-aware MPI with OpenACC

```
#pragma acc host_data use_device(s_buf)
MPI_Send(s_buf,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

- To use MPI with OpenACC

```
#pragma acc update host(s_buf[0:size] )
MPI_Send(s_buf,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

Profiling MPI+CUDA applications

- Use nvprof:

  ```
  mpirun –np n nvprof --output-profile out.$MV2_COMM_WORLD_RANK ./app
  ```

  see docs.nvidia.com for details

- Use CUDA-aware tracing libraries like score-p or VampirTrace and tools like Vampir

- Watch the recording of Rolf van de Vaart’s Session: S3045 - Tips & Tricks for Getting the Most Out of GPU-accelerated Clusters

Conclusions

- Use CUDA-aware MPI when possible
- Depending on CUDA version, hardware setup, … a CUDA-aware MPI gives you
  - Ease of programming
  - Pipelined data transfer which automatically provides optimizations when available
    - Overlap CUDA copy and RDMA transfer
  - Utilization of the best GPUDirect technology available
- Examples are available for download at github:

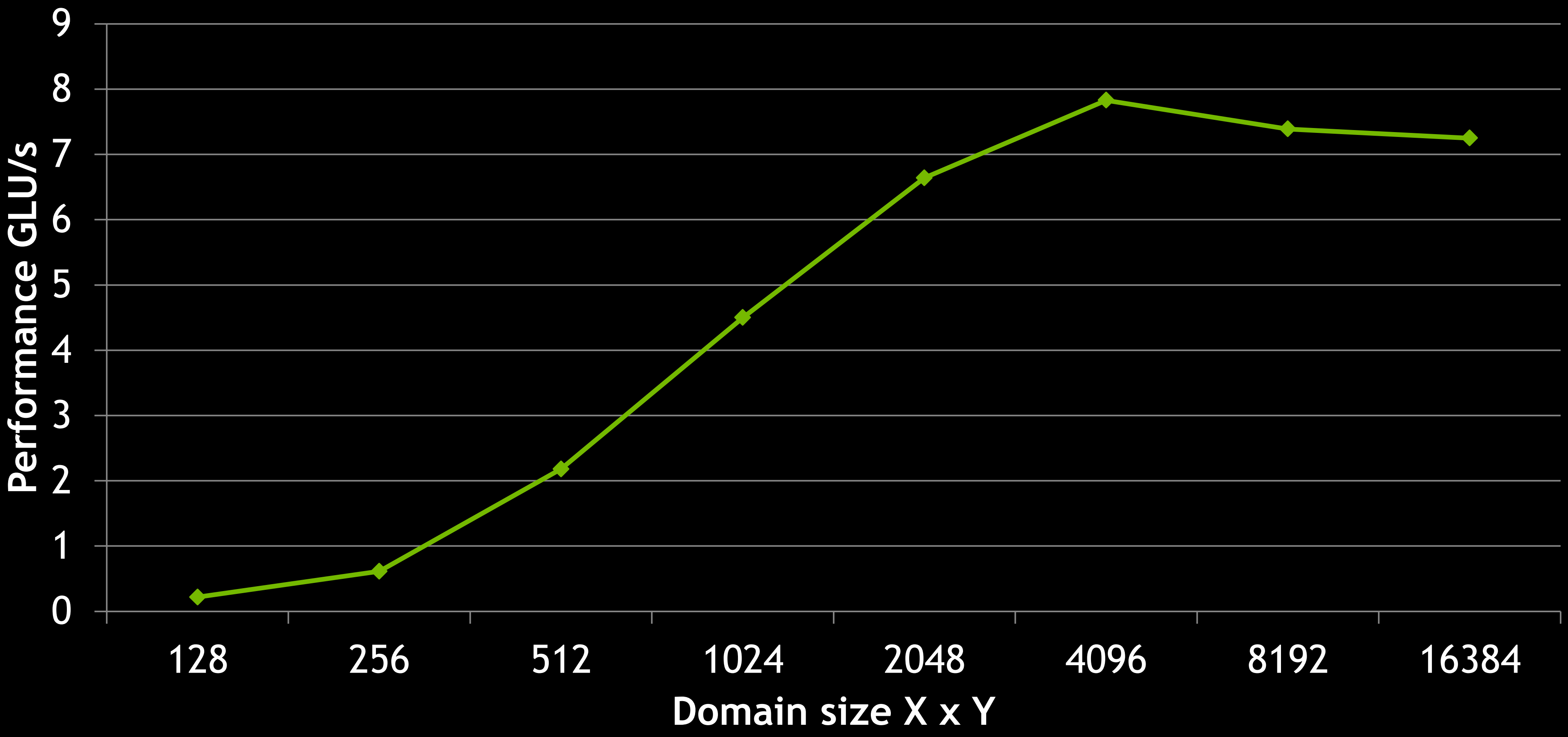
https://github.com/parallel-forall/code-samples/tree/master/posts/cuda-aware-mpi-example

Thank you

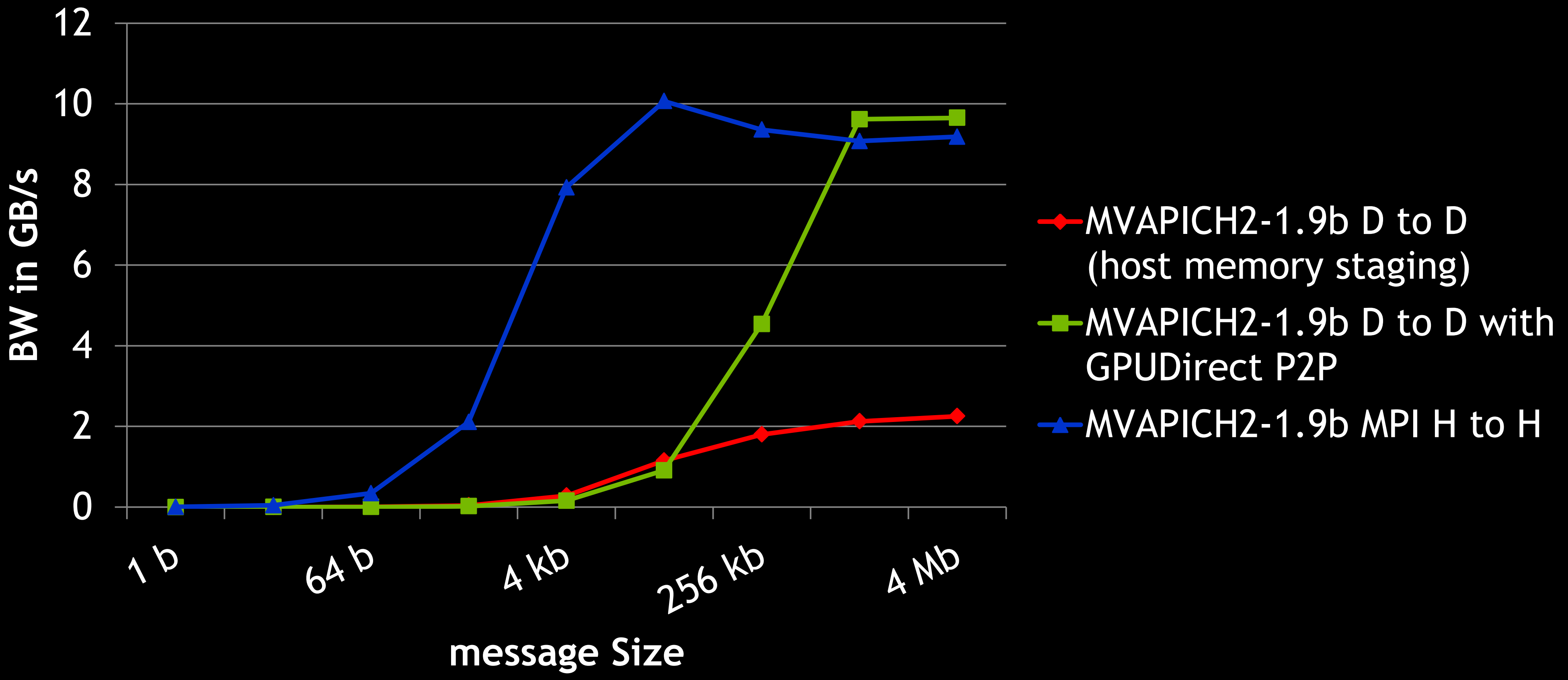
Examples with source in my Parallel Forall blog posts:

http://developer.nvidia.com/content/introduction-cuda-aware-mpi

CUDA Jacobi – Single GPU Performance



Performance Results single Node



Latency (1 byte) 15.87 µs 19.70 µs 0.24 µs