MULTI GPU PROGRAMMING WITH MPI

JIRI KRAUS, PRINCIPAL DEVTECH COMPUTE

MAGNUM IO STACK

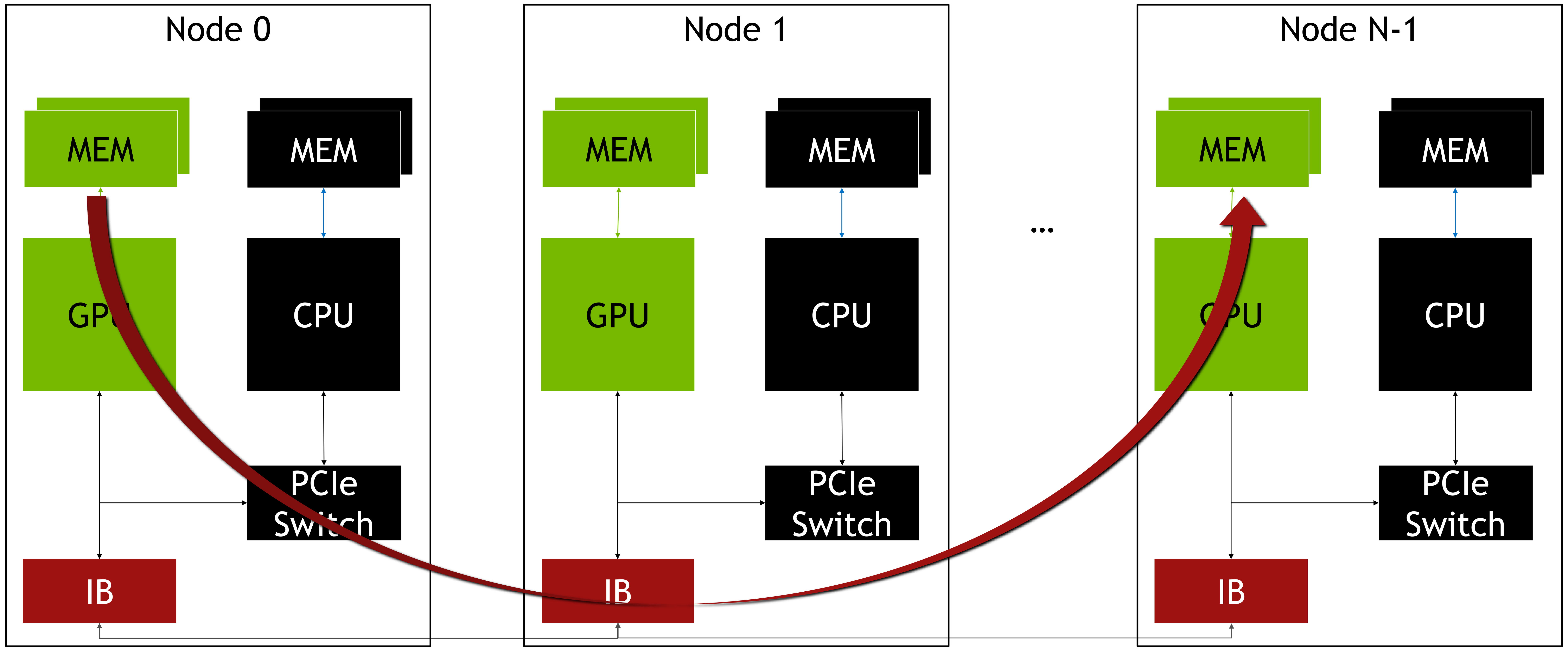
MAGNUM IO NETWORK IO TECHNOLOGIES AND LIBRARIES

Todays Topic

- HPC-X Toolkit
  - CUDA-aware MPI: OpenMPI based building on UCX
  - OpenSHMEM
  - UCX: CPU-centric GPU-aware low-level communications library
  - UCC and HCOLL: Accelerated collectives leveraging NCCL, SHARP and CORE-Direct. HCOLL should be eventually replaced by the UCF project UCC (**U**nified **C**ommunication **C**ollectives)
  - NCCL SHARP Plugins
- NCCL: **N**VIDIA **C**ollective **C**ommunication **L**ibrary (GPU-offloaded communications)
- NVSHMEM: OpenSHMEM implementation for NVIDIA GPUs supporting GPU initiated communications
- GPUDirect: P2P, RDMA and GDRCopy
- Accelerated Switch and Packet Processing ($ASAP^2$): accelerated virtual switching
- Data Plane Development Kit (DPDK): Libraries for fast packet processing in user space

Tomorrow: NCCL and NVSHMEM

CUDA-AWARE MPI




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

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

YOU WILL LEARN

- What MPI is
- How to use MPI for inter GPU communication with CUDA, Directives and standard language parallelism
- What CUDA-aware MPI is
- How to accelerated MPI collectives on GPU data
- What Multi Process Service is and how to use it
- How to use NVIDIA tools in an MPI environment
- How to hide MPI communication times

MESSAGE PASSING INTERFACE - MPI

- Standard to exchange data between processes via messages
  - Defines API to exchange messages
    - Point to Point: e.g. `MPI_Send`, `MPI_Recv`
    - Collectives: e.g. `MPI_Reduce`
- Multiple implementations (open source and commercial)
  - Bindings for C/C++, Fortran, Python, …
  - E.g. MPICH, OpenMPI, MVAPICH, IBM Spectrum MPI, Cray MPT, ParaStation MPI, …

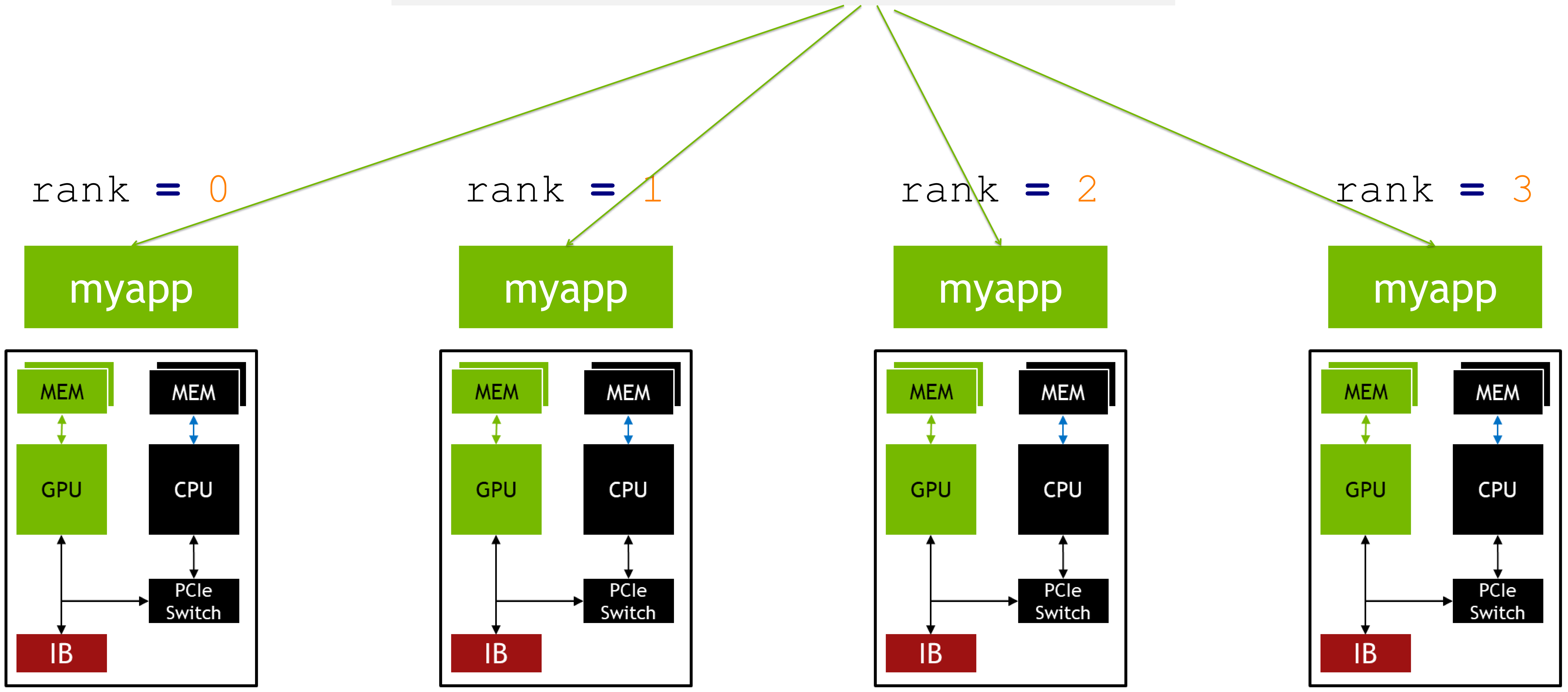
MPI - SKELETON

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

MPI

Compiling and Launching

```
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
```



EXAMPLE: JACOBI SOLVER

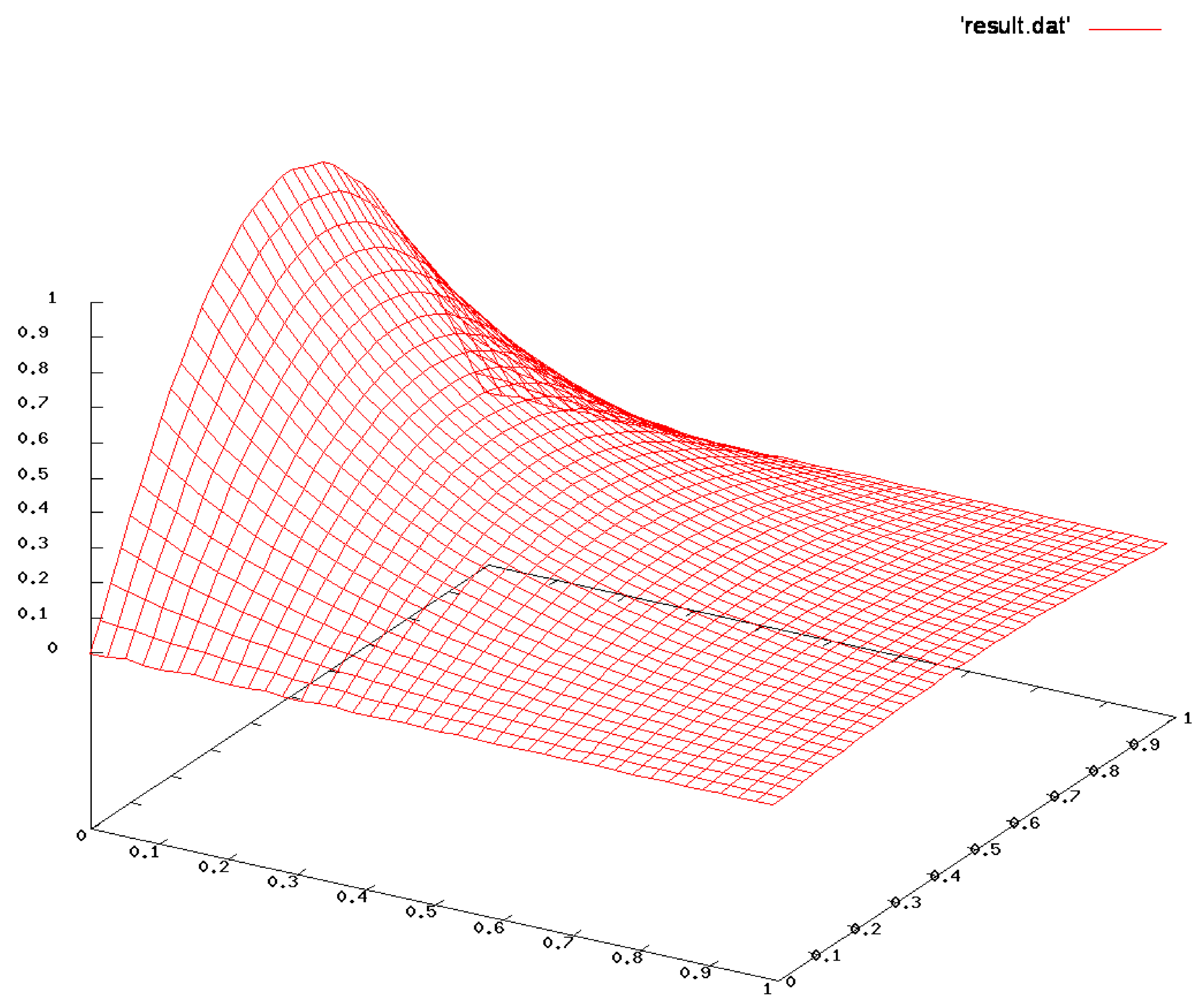
Solves the 2D-Laplace Equation on a rectangle

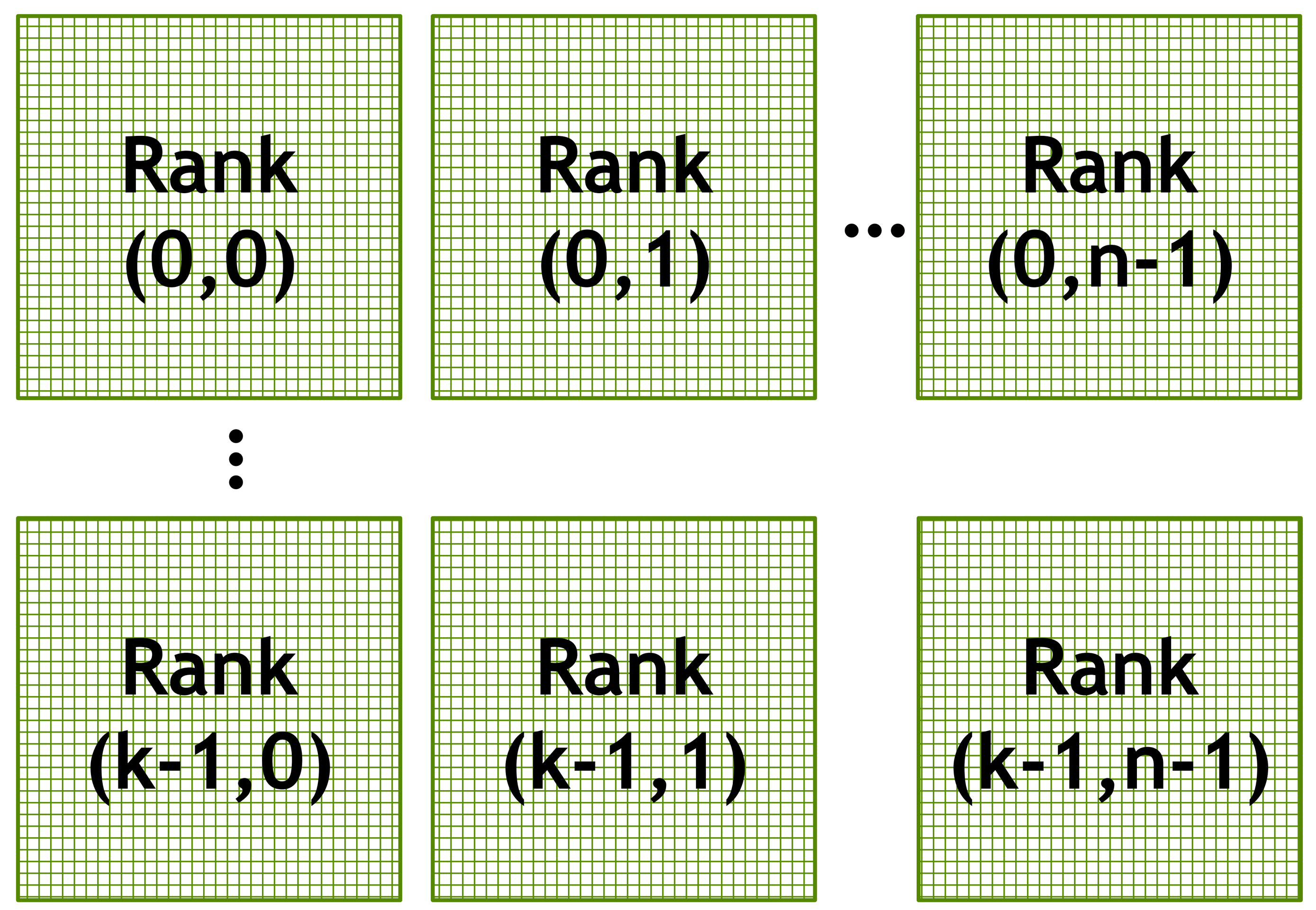
$$\Delta \boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) = \mathbf{0} \;\forall\; (\boldsymbol{x},\boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Dirichlet boundary conditions (constant values on boundaries)

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

2D domain decomposition with n x k domains





EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

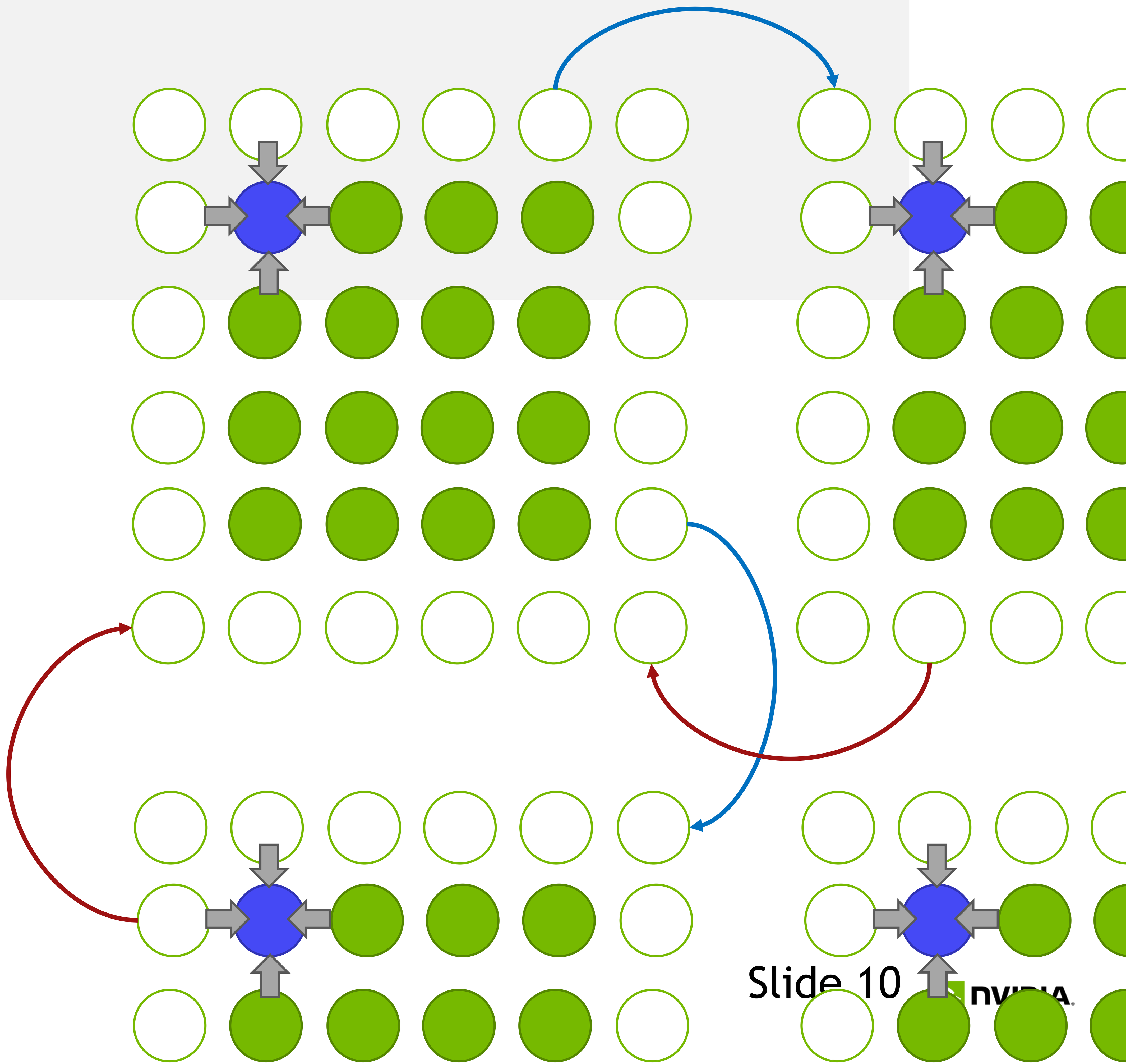
Do Jacobi step:

```
for (int iy=1; iy < ny-1; ++iy)
for (int ix=1; ix < nx-1; ++ix)
    u_new[ix][iy] = 0.0f - 0.25f*(  u[ix-1][iy] + u[ix+1][iy]
                                  + u[ix][iy-1] + u[ix][iy+1]);
```

Exchange Halo with 1 to 4 neighbors

Swap u_new and u

Next iteration



EXAMPLE JACOBI

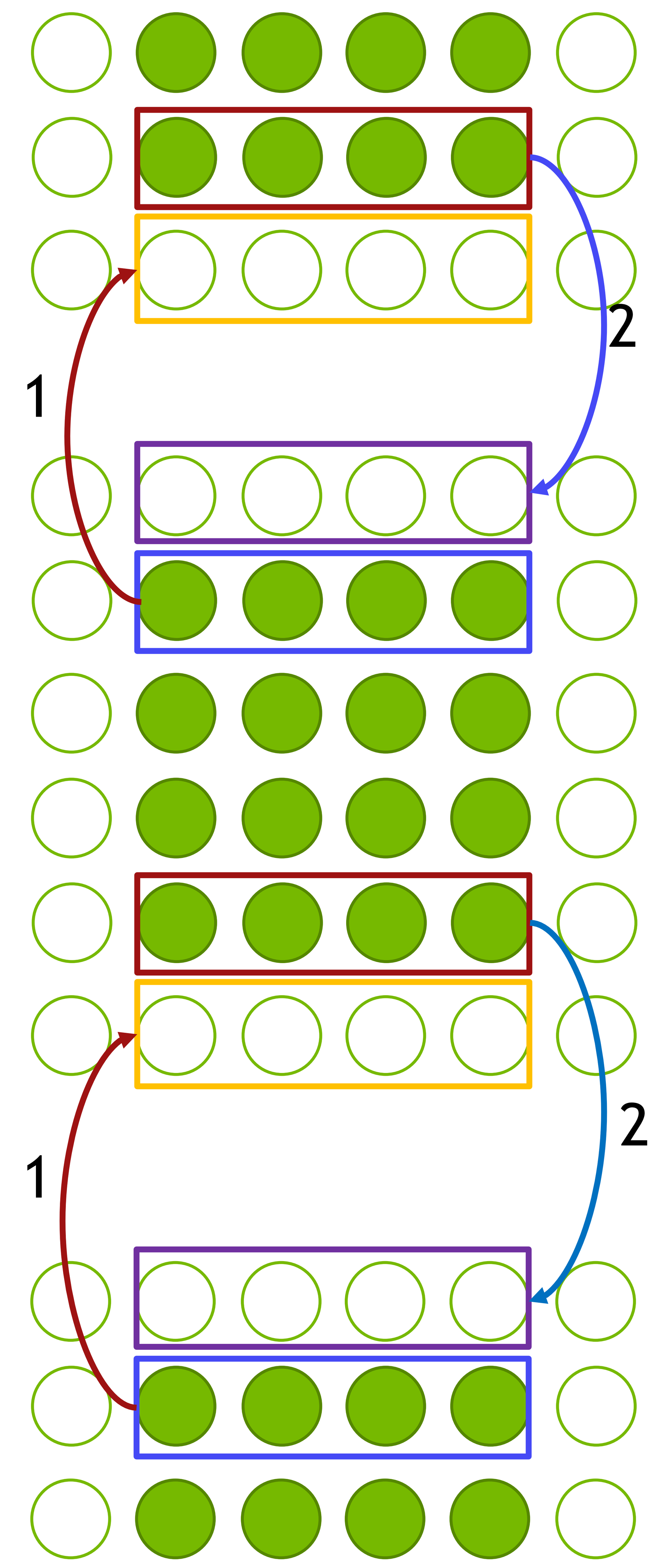
Top/Bottom Halo

OpenACC

```
#pragma acc host_data use_device ( u_new ) {
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

CUDA/ISO C++

```
MPI_Sendrecv(u_new_d+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new_d+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new_d+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new_d+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE JACOBI

Top/Bottom Halo

without CUDA-aware MPI

OpenACC

```
#pragma acc update host(u_new[offset_first_row:m-2],u_new[offset_last_row:m-2])
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma acc update device(u_new[offset_top_boundary:m-2],u_new[offset_bottom_boundary:m-2])
```

CUDA

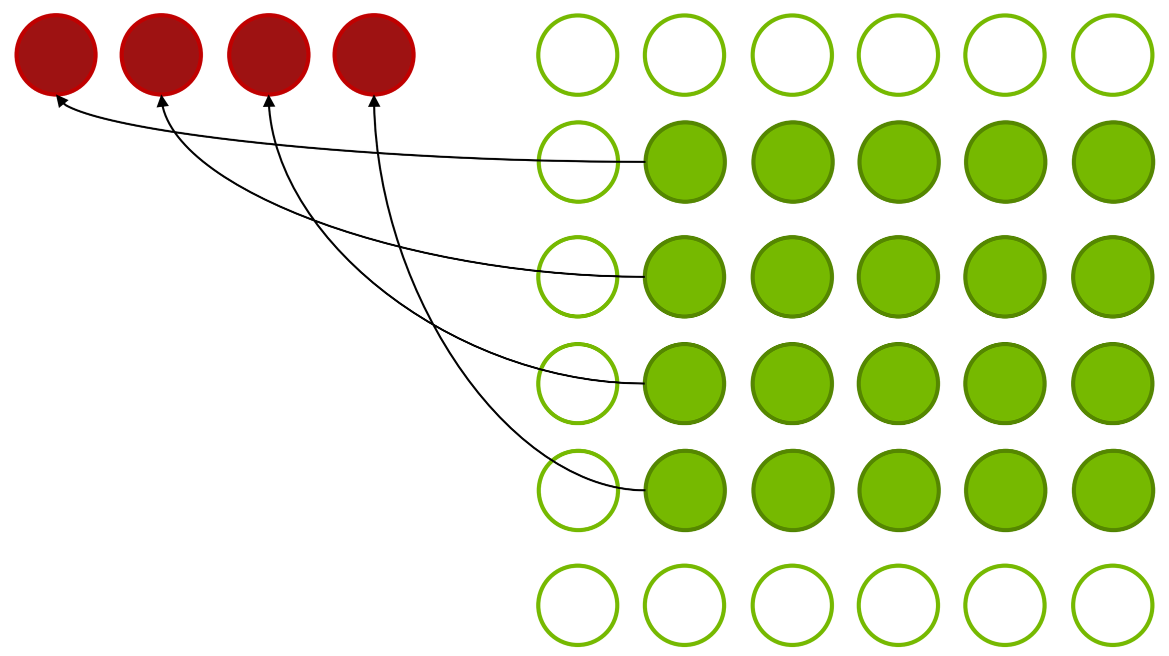
```
//send to bottom and receive from top top bottom omitted
cudaMemcpy(  u_new+offset_first_row,
             u_new_d+offset_first_row, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaMemcpy(  u_new_d+offset_bottom_boundary,
             u_new+offset_bottom_boundary, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
```

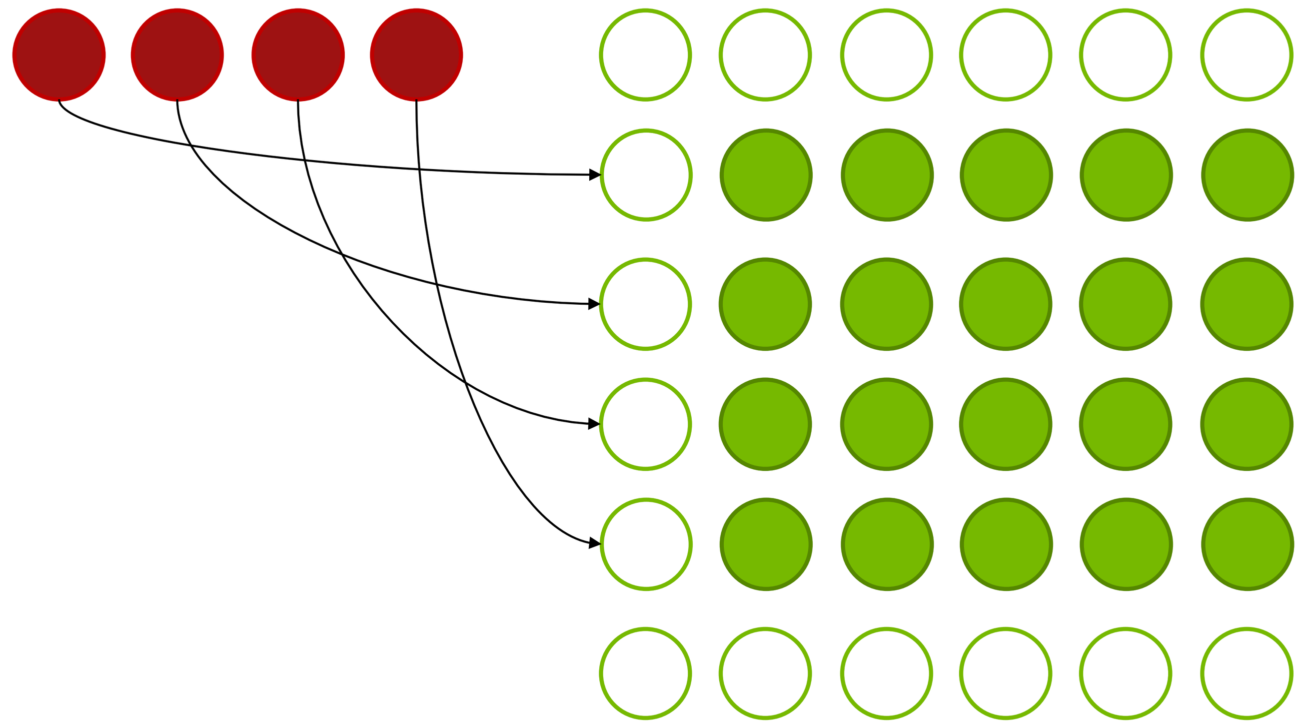
EXAMPLE: JACOBI

Left/Right Halo

ISO C++

```
//right neighbor omitted
std::for_each(std::execution::par_unseq,
              std::ranges::views::iota(0), n-2,
              [=](Index_t i) {
                  to_left[i] = u_new[(i+1)*m+1];
              });
MPI_Sendrecv( to_left, n-2, MPI_DOUBLE, l_nb, 0,
              from_right, n-2, MPI_DOUBLE, r_nb, 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE );
std::for_each(std::execution::par_unseq,
              std::ranges::views::iota(0), n-2,
              [=](Index_t i) {
                  u_new[(m-1)+(i+1)*m] = from_right[i];
              });
```



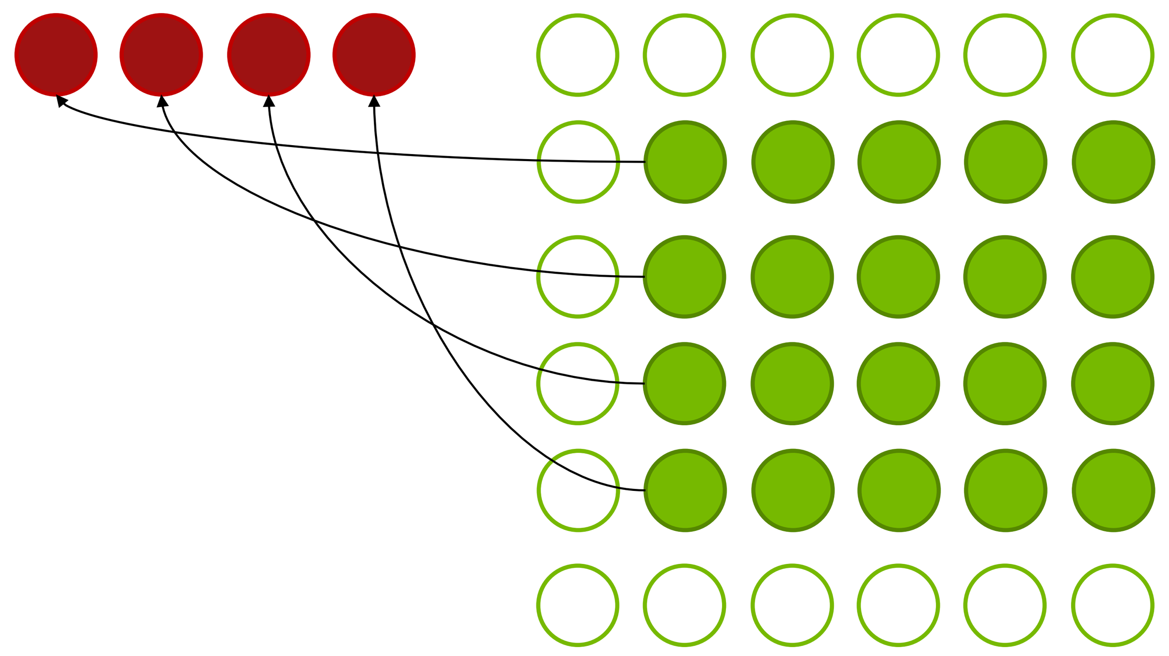


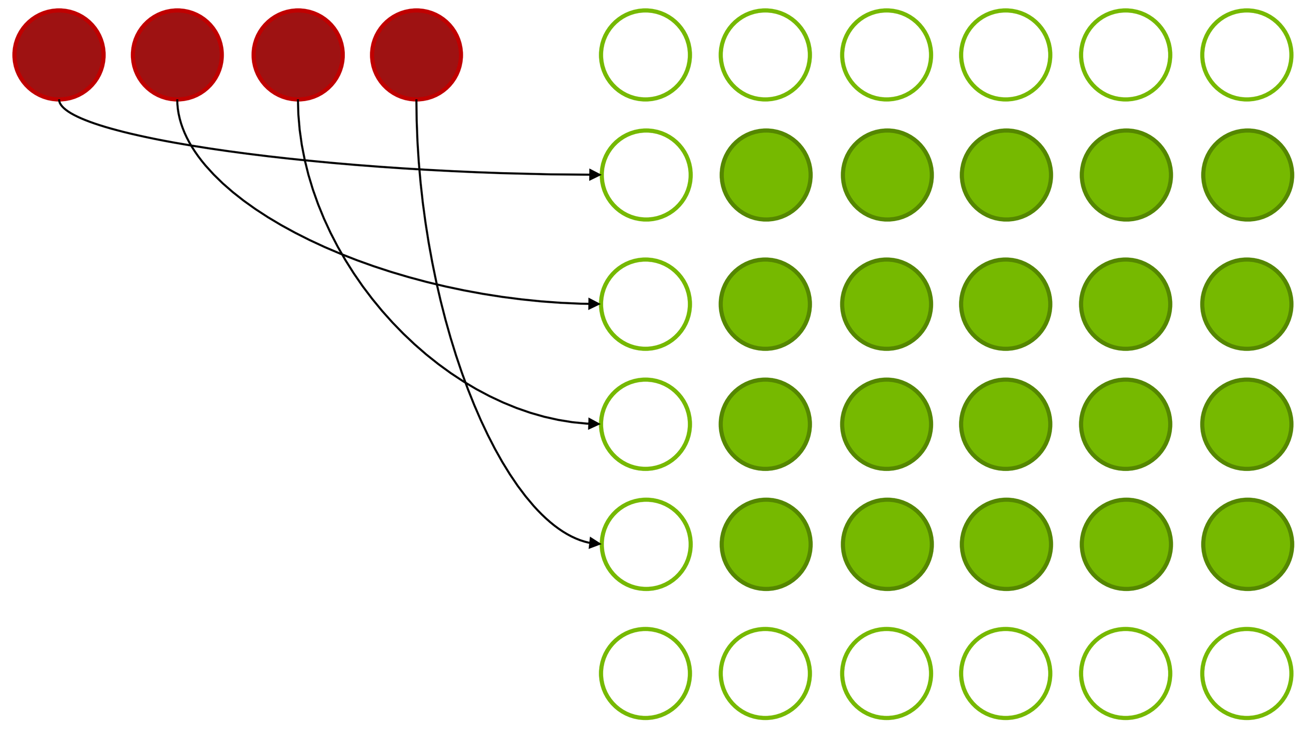
EXAMPLE: JACOBI

Left/Right Halo

OpenACC

```
//right neighbor omitted
#pragma acc parallel loop present ( u_new, to_left )
for ( int i=0; i<n-2; ++i )
    to_left[i] = u_new[(i+1)*m+1];

#pragma acc host_data use_device ( from_right, to_left ) {
  MPI_Sendrecv( to_left, n-2, MPI_DOUBLE, l_nb, 0,
                from_right, n-2, MPI_DOUBLE, r_nb, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}

#pragma acc parallel loop present ( u_new, from_right )
for ( int i=0; i<n-2; ++i )
    u_new[(m-1)+(i+1)*m] = from_right[i];
```



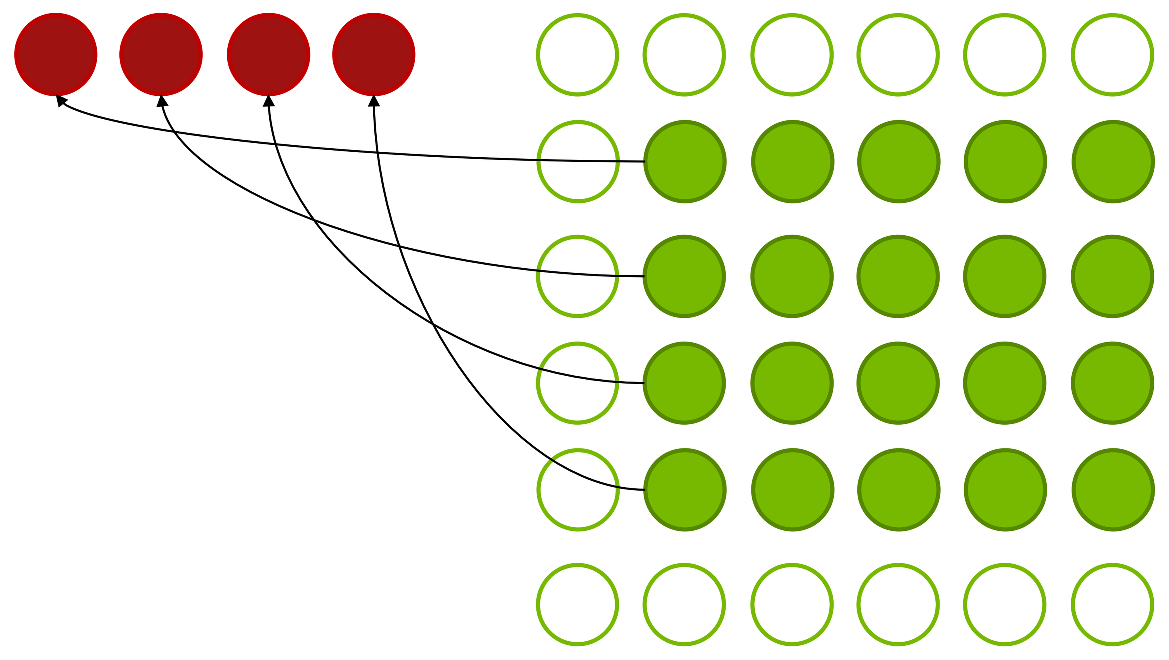


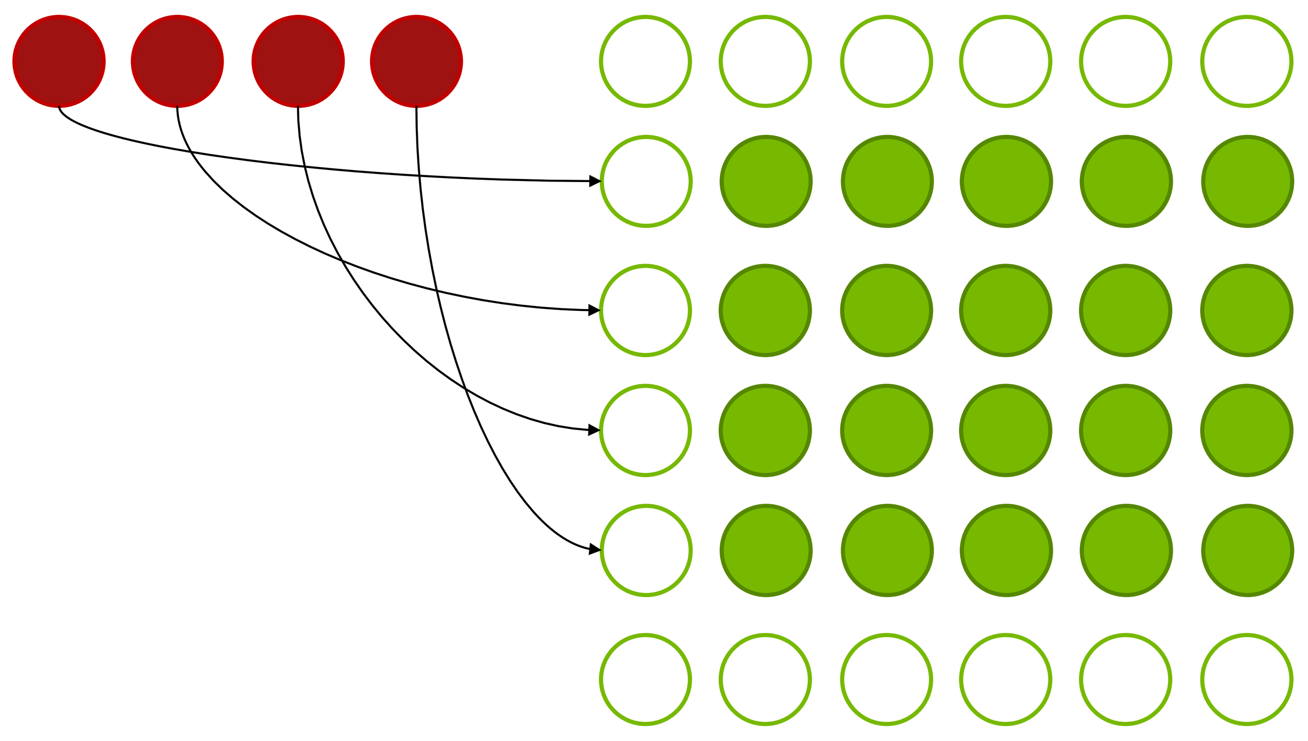
EXAMPLE: JACOBI

Left/Right Halo

CUDA

```
//right neighbor omitted
pack<<<gs,bs,0,s>>>(to_left_d, u_new_d, n, m);
cudaStreamSynchronize(s);


  MPI_Sendrecv( to_left_d, n-2, MPI_DOUBLE, l_nb, 0,
                from_right_d, n-2, MPI_DOUBLE, r_nb, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );


unpack<<<gs,bs,0,s>>>(u_new_d, from_right_d, n, m);
```



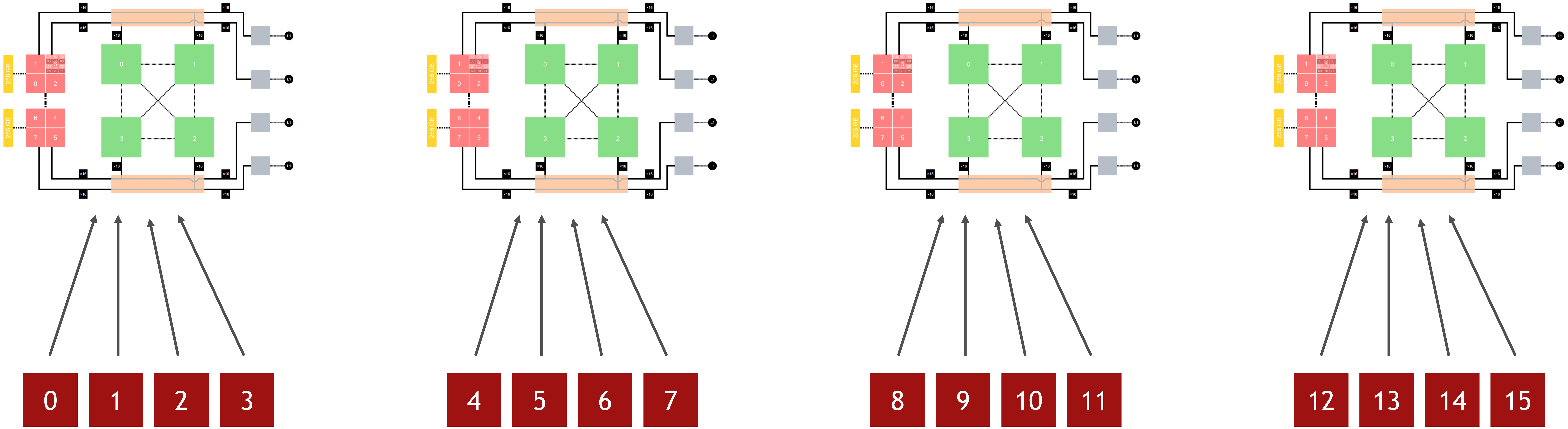


LAUNCH MPI+CUDA/OPENACC PROGRAMS

Launch one process per GPU

- MVAPICH: `$ MV2_USE_CUDA=1 mpirun -np ${np} ./myapp <args>`
- Open MPI: CUDA-aware features are enabled per default
- Cray: `MPICH_RDMA_ENABLED_CUDA`
- IBM Spectrum MPI: `$ mpirun -gpu -np ${np} ./myapp <args>`
- ParaStation MPI: `$ PSP_CUDA=1 mpirun -np ${np} ./myapp <args>`

HANDLING MULTI GPU NODES

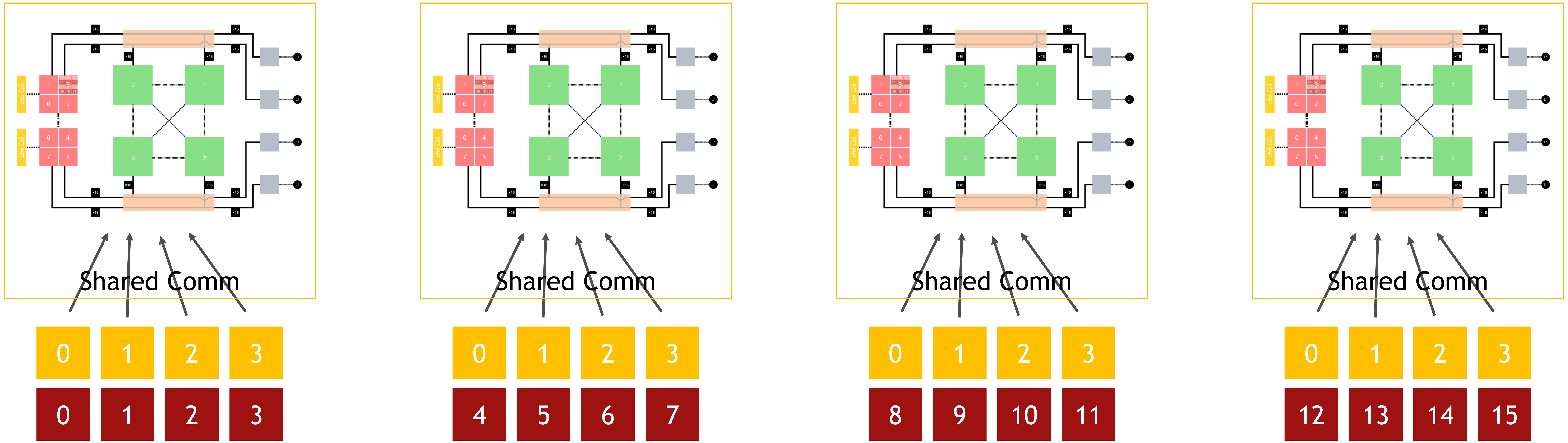


HANDLING MULTI GPU NODES

How to determine the local rank? – MPI-3

```
MPI_Comm local_comm;
MPI_CALL(MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, MPI_INFO_NULL, &local_comm));

MPI_CALL(MPI_Comm_rank(local_comm, &local_rank));

MPI_CALL(MPI_Comm_free(&local_comm));
```

HANDLING MULTI GPU NODES



HANDLING MULTI GPU NODES

GPU-affinity

Use local rank:

```
int local_rank = //determine local rank
int num_devs = 0;
cudaGetDeviceCount(&num_devs);
cudaSetDevice(local_rank%num_devs);
```

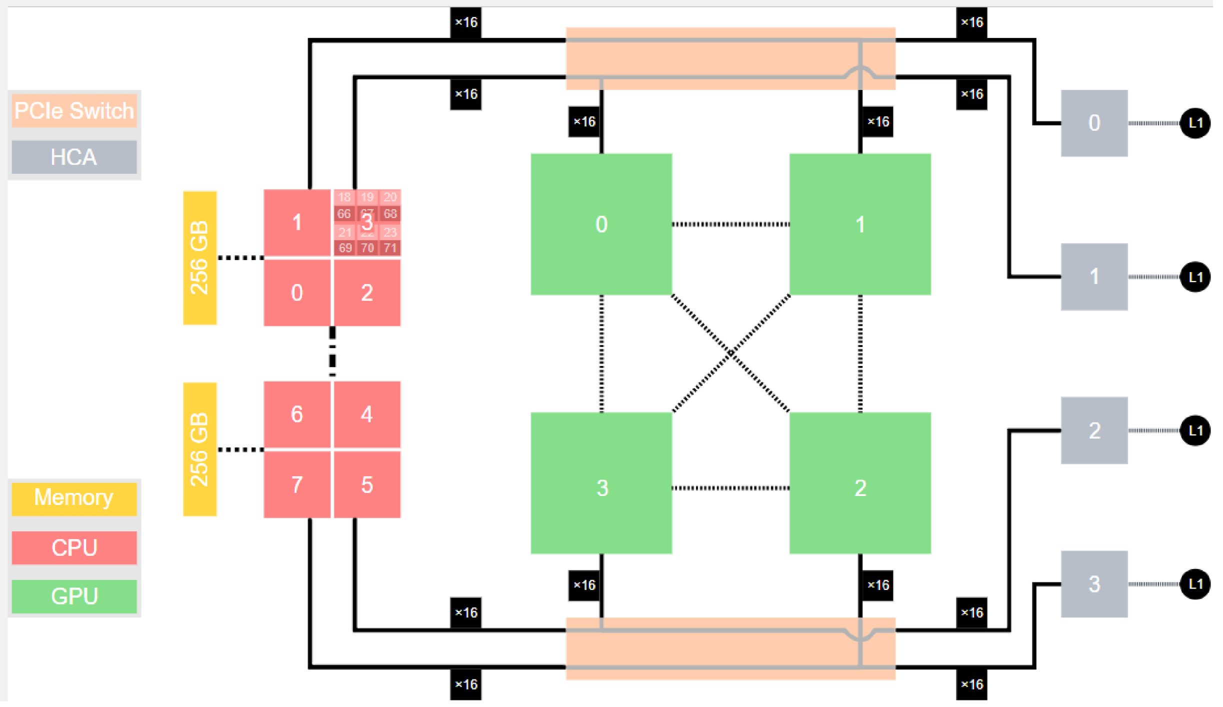
Needed if resource manager handles GPU affinity

UCX TIPS AND TRICKS

Example binding script

```
case ${SLURM_LOCALID} in
  0)
       export CUDA_VISIBLE_DEVICES=0
       export UCX_NET_DEVICES=mlx5_1:1
       CPU_BIND=18-23
       ;;
   1)
       export CUDA_VISIBLE_DEVICES=1
       export UCX_NET_DEVICES=mlx5_0:1
       CPU_BIND=6-11
       ;;
   2)
       export CUDA_VISIBLE_DEVICES=2
       export UCX_NET_DEVICES=mlx5_3:1
       CPU_BIND=42-47
       ;;
   3)
       export CUDA_VISIBLE_DEVICES=3
       export UCX_NET_DEVICES=mlx5_2:1
       CPU_BIND=30-35
       ;;
esac
numactl --physcpubind=${CPU_BIND} $*
```

```
[kraus1@jwb0007]$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity    NUMA Affinity
GPU0     X      NV4     NV4     NV4     SYS     PIX     SYS     SYS     18-23,66-71     3
GPU1    NV4      X      NV4     NV4     PIX     SYS     SYS     SYS     6-11,54-59      1
GPU2    NV4     NV4      X      NV4     SYS     SYS     SYS     PIX     42-47,90-95     7
GPU3    NV4     NV4     NV4      X      SYS     SYS     PIX     SYS     30-35,78-83     5
mlx5_0  SYS     PIX     SYS     SYS      X      SYS     SYS     SYS
mlx5_1  PIX     SYS     SYS     SYS     SYS      X      SYS     SYS
mlx5_2  SYS     SYS     SYS     PIX     SYS     SYS      X      SYS
mlx5_3  SYS     SYS     PIX     SYS     SYS     SYS     SYS      X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```



https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

HANDLING MULTI GPU NODES

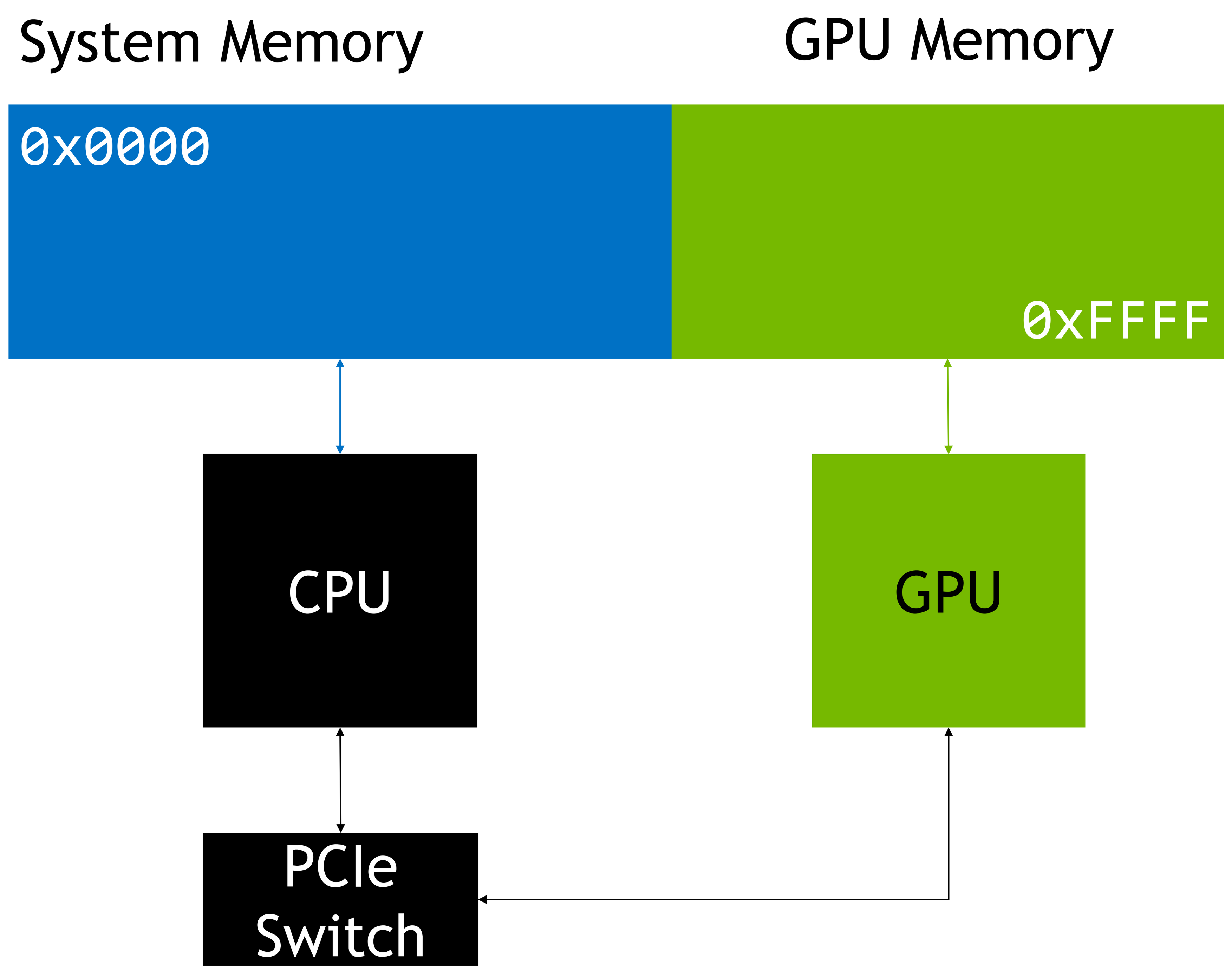
Hands On

See `01-MPI/01-GPU_affinity/exercises/Instructions.md`:

- Experiment with transparent GPU affinity handling on JURECA (set `CUDA_VISIBLE_DEVICES=0,1,2,3` to disable)
  - Compare Nsight Systems traces and reported runtimes
- Add GPU affinity handling code (follow TODOs in `task/jacobi.cpp`)

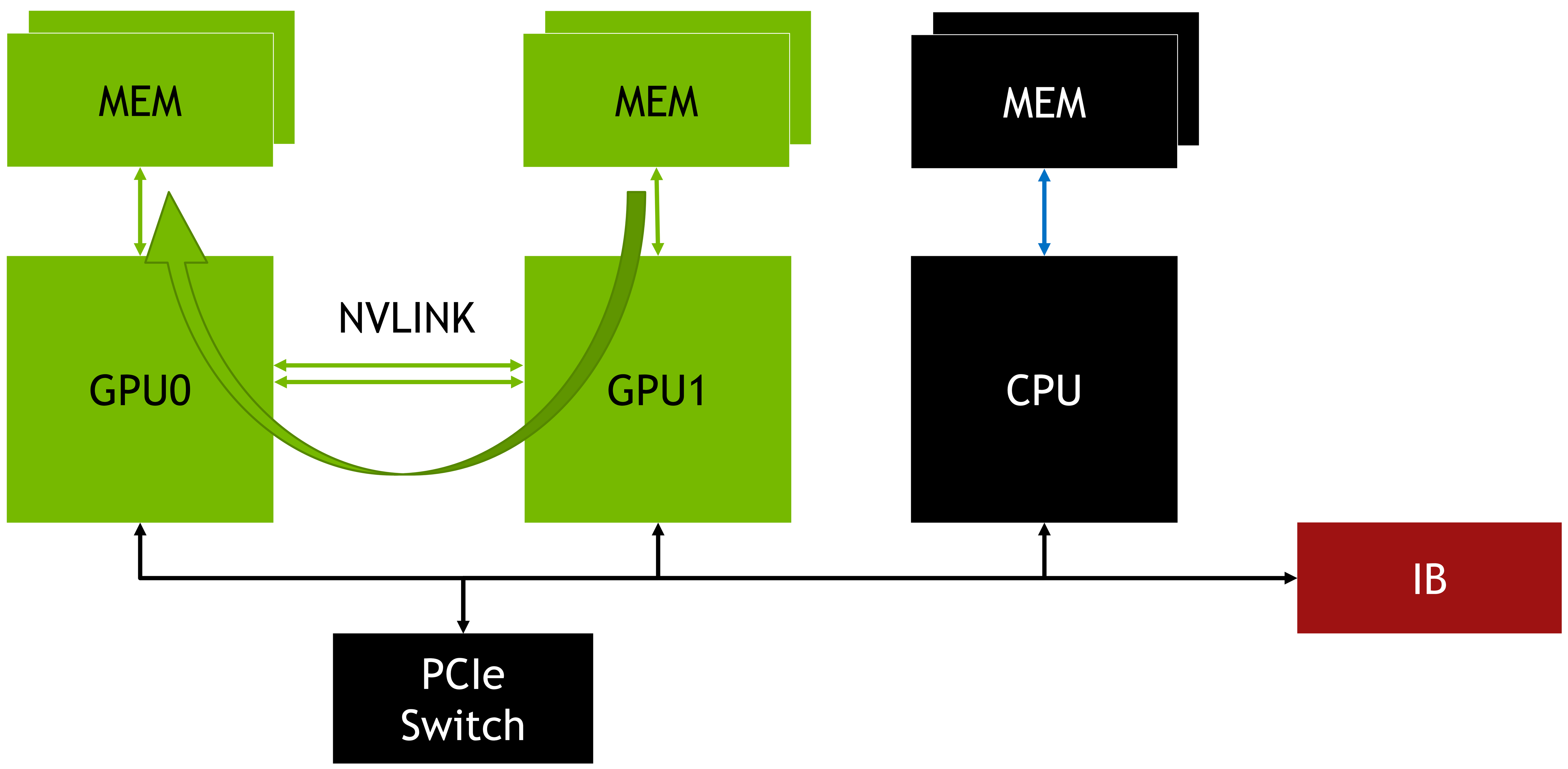
UNIFIED VIRTUAL ADDRESSING

- 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 Windows (+TCC)



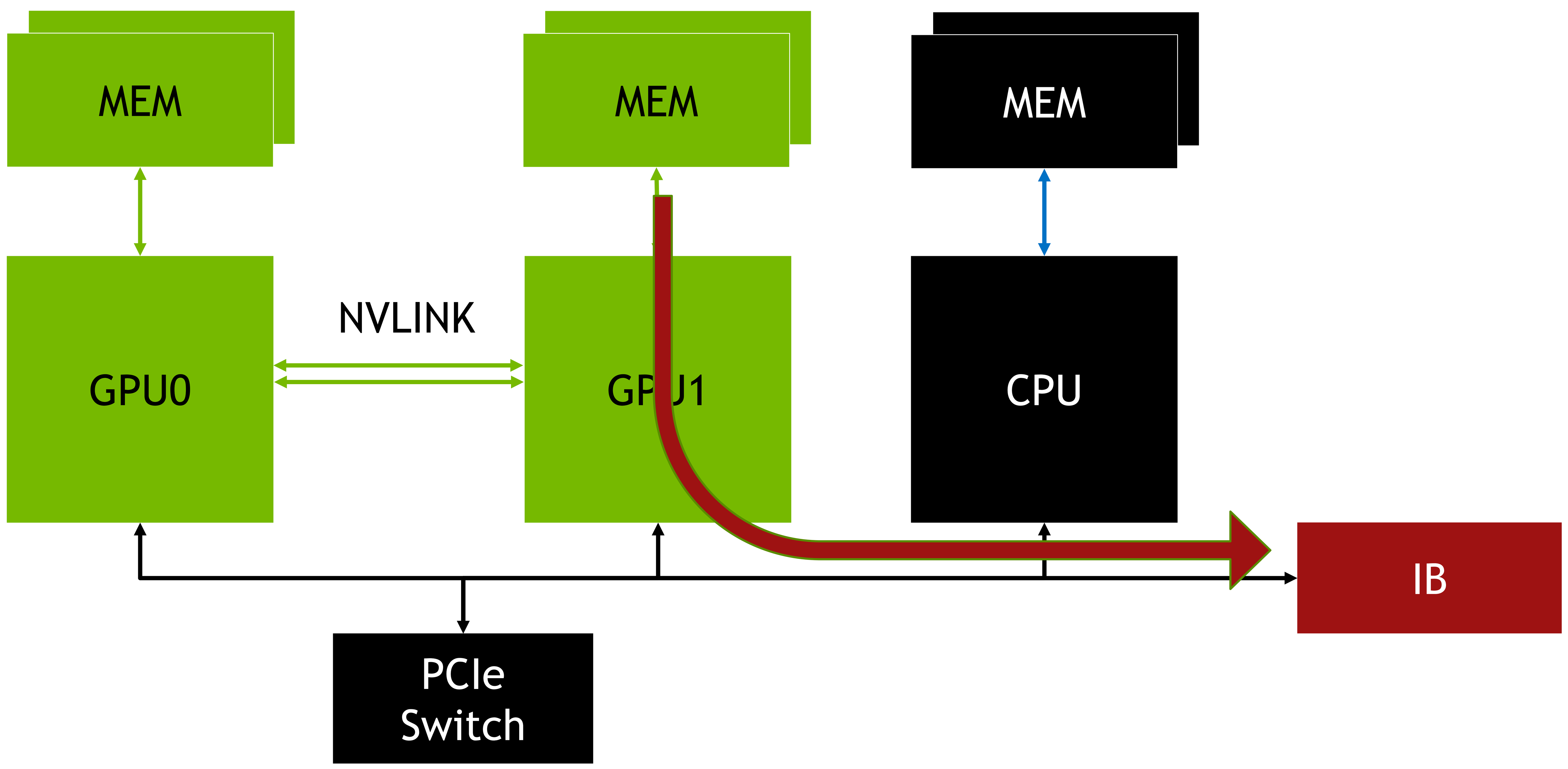
NVIDIA GPUDIRECT

Peer to Peer Transfers



NVIDIA GPUDIRECT

Support for RDMA



CUDA-AWARE MPI

Example:

MPI Rank 0 `MPI_Send` from GPU Buffer

MPI Rank 1 `MPI_Recv` to GPU Buffer

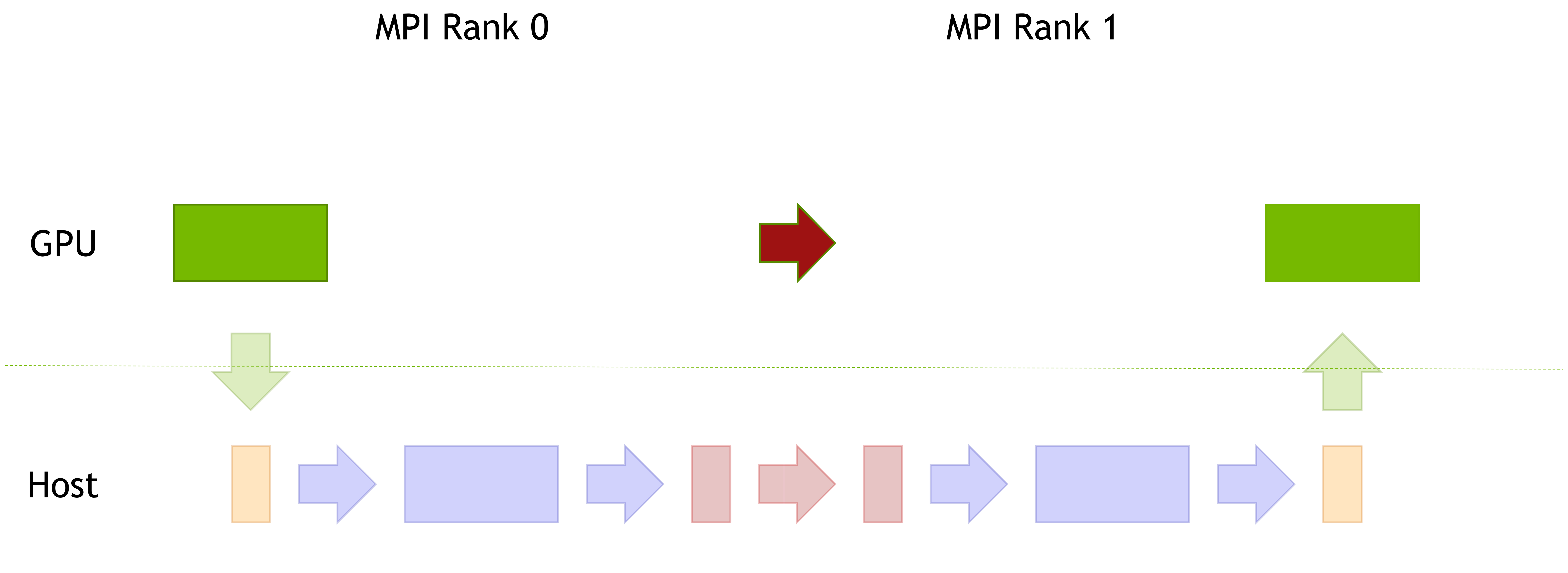
Show how CUDA+MPI works in principle

Depending on the MPI implementation, message size, system setup, … situation might be different

Two GPUs in two nodes

GPU TO REMOTE GPU

CUDA-aware MPI with support for GPUDirect RDMA




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

MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

OSU_BW NSIGHT SYSTEMS TIMELINE

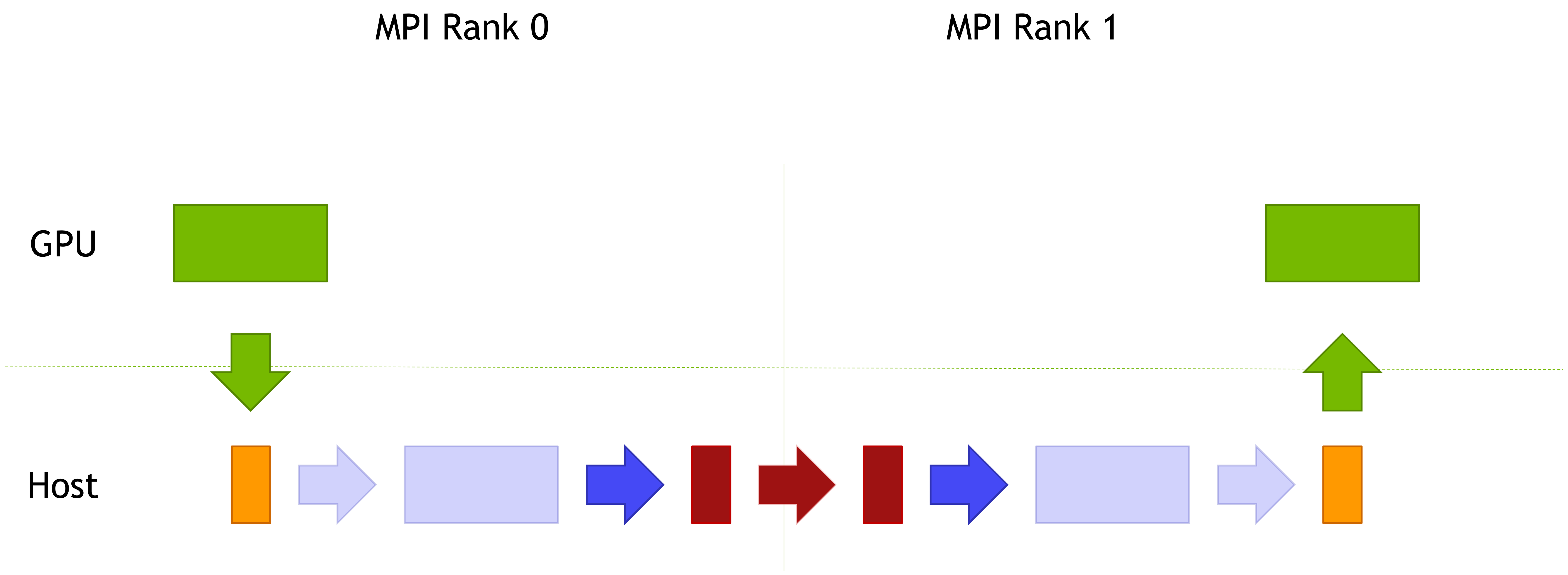
Internode with GPUDirect RDMA on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.GDR.%q{SLURM_PROCID}
```

#	Name	Start	Duration	TID	Category
1	ucp_tag_send_nb	0.787995s	1.350 µs	22042	UCP transfer s...
2	ucp_tag_send_nb	0.787996s	1.900 µs	22042	UCP transfer s...
3	ucp_tag_send_nb	0.787998s	182.481 µs	22042	UCP transfer ...
4	ucp_tag_probe_nb	0.787998s	130 ns	22042	UCP [1]

GPU TO REMOTE GPU

CUDA-aware MPI without support for GPUDirect




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

MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

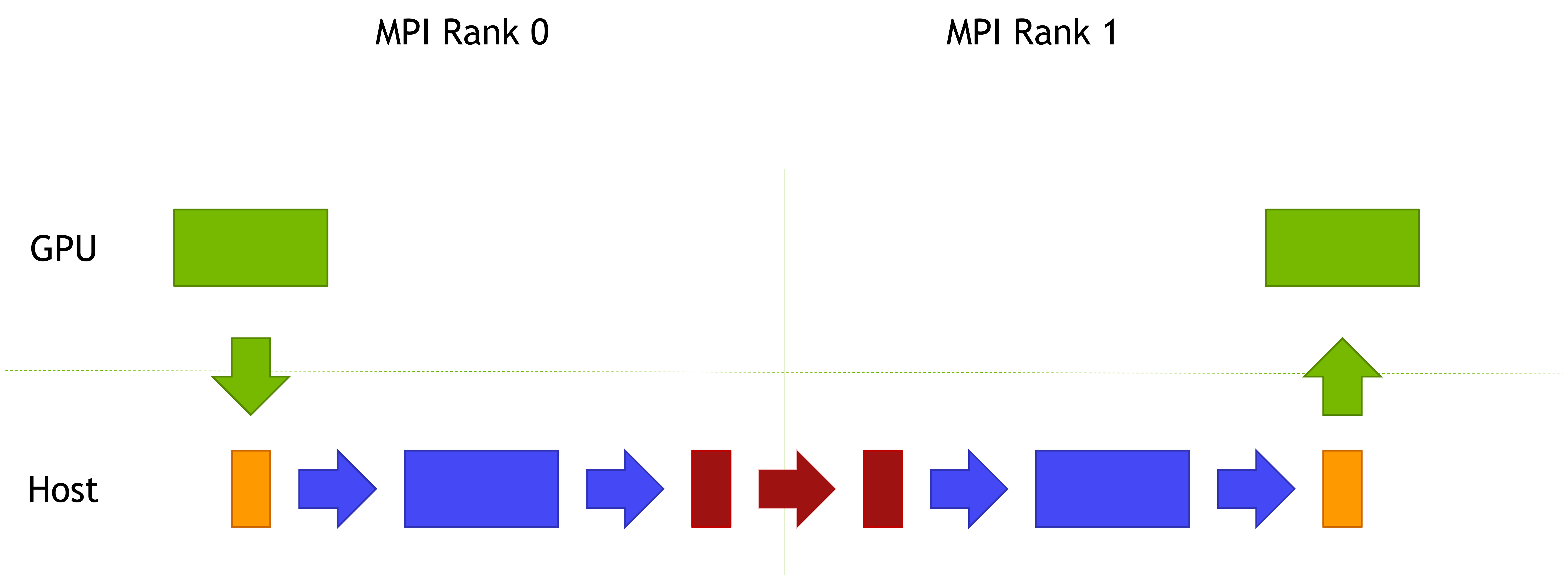
OSU_BW NSIGHT SYSTEMS TIMELINE

Internode without GPUDirect on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.noGDR.%q{SLURM_PROCID}
```

GPU TO REMOTE GPU

MPI without CUDA support




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

MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaMemcpy(r_buf_d, r_buf_h, size, cudaMemcpyHostToDevice);
```

OSU_BW NSIGHT SYSTEMS TIMELINE

Internode MPI without CUDA support on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.noCUDAMPI.%q{SLURM_PROCID}
```

PERFORMANCE RESULTS GPUDIRECT RDMA

OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS Booster

Latency (1 byte) 4.27 us 24.56 us 25.64 us

JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

PERFORMANCE RESULTS GPUDIRECT P2P

OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS Booster

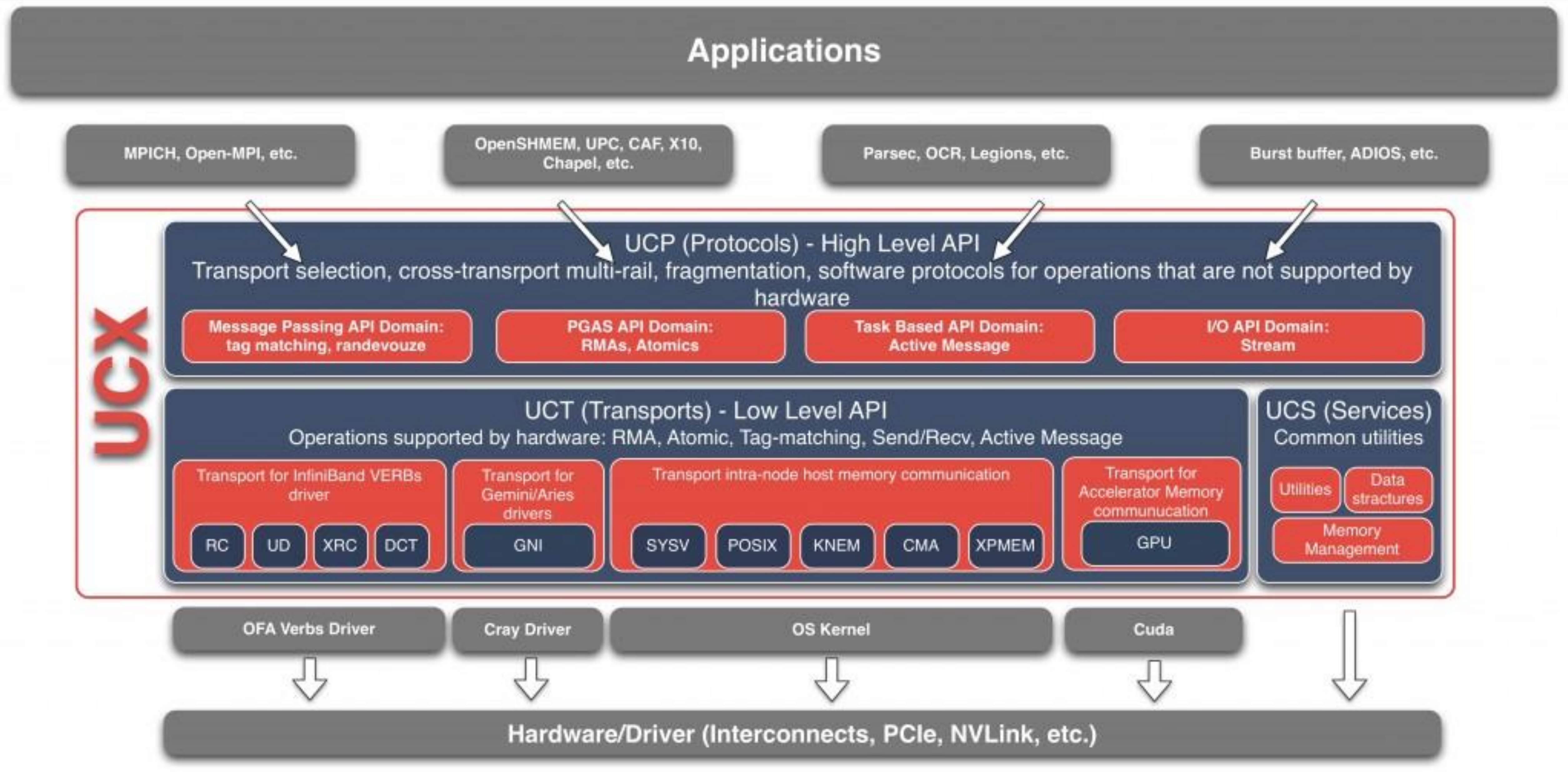
Latency (1 byte) 2.45 us 22.01 us 23.50 us

JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

UCX TIPS AND TRICKS

Check setting and knobs with `ucx_info`

```
$ ucx_info -caf | grep -B9 UCX_RNDV_SCHEME

#

# Communication scheme in RNDV protocol.

#  get_zcopy - use get_zcopy scheme in RNDV protocol.

#  put_zcopy - use put_zcopy scheme in RNDV protocol.

#  auto      - runtime automatically chooses optimal
scheme to use.

#

# syntax:    [get_zcopy|put_zcopy|auto]

#

UCX_RNDV_SCHEME=auto
```



UCX TIPS AND TRICKS

Enable logging to see what is going on

`UCX_LOG_LEVEL=data UCX_LOG_FILE=log-%h-%p` helpful to check for used protocols and selected HCAs:

```
[1605706306.970537] [jwb1238:7263 :0]     ucp_worker.c:1627 UCX  INFO  ep_cfg[0]: tag(cuda_copy/cuda); rma(gdr_copy/cuda);
[1605706306.972721] [jwb1238:7263 :0]     ucp_worker.c:1627 UCX  INFO  ep_cfg[1]: tag(self/memory rc_mlx5/mlx5_1:1 cma/memory cuda_copy/cuda);
[1605706306.997849] [jwb1238:7263 :1]     ucp_worker.c:1627 UCX  INFO  ep_cfg[2]: tag(rc_mlx5/mlx5_1:1);
```

UCX TIPS AND TRICKS

https://github.com/openucx/ucx/wiki/UCX-environment-parameters

UCX_NET_DEVICES: To select HCA for optimal GPU-HCA affinity, should not be necessary with UCX 1.9 or newer

UCX_TLS: Select transports to use, default: `all`

- `cuda` is an alias for: `cuda_copy`, `cuda_ipc`, `gdr_copy`
- To run without any GPUDirect flavor set `UCX_TLS` to only include `cuda_copy`, e.g. `UCX_TLS=rc,sm,cuda_copy` and `UCX_IB_GPU_DIRECT_RDMA=no` (`rc` transport uses GPUDirect RDMA otherwise).
- Parastation MPI also has `PSP_CUDA_ENFORCE_STAGING=1`.

UCX_MEMTYPE_CACHE: Set to n to disable mem type cache. Sometimes necessary if the CUDA runtime is linked statically!

GPUDIRECT EXPERIMENTS

Hands On

See `01-MPI/02-GPUDirect/Instructions.md`: for experiments:

- Run and profile with default settings
- Run and profile with GPUDirect P2P disabled (`UCX_TLS=rc_x,self,sm,cuda_copy`)
- Run and profile with GPUDirect P2P disabled (`UCX_TLS=rc_x,self,sm,cuda_copy`) and GPUDirect RDMA disabled (`UCX_IB_GPU_DIRECT_RDMA=no`)
- Compare what happens during MPI on the Nsight Systems timeline for all variants

HIERARCHICAL COMMUNICATION ALGORITHMS LIBRARY

HCOLL

Library for software

- **H**ierarchical **CO**mmunication a**L**gorithms (HCOL) – CPU and GPU data
- NCCL – GPU data*

and hardware

- **S**calable **H**ierarchical **A**ggregation and **R**eduction **P**rotocol (SHARP – in Network/Switch)
- **C**ollectives **O**ffload **R**esource **E**ngine (CORE-Direct – HCA offloading)

accelerated collectives.

To be replaced by the UCF project Unified Collective Communications (UCC): https://github.com/openucx/ucc

Used by HPC-X, other MPI implementations can link `libhcoll.so`, e.g. OpenMPI

HCOLL predecessor was **F**abric **C**ollective **A**ccelerator (FCA)

*With MPI depending on the used thread mode and if blocking or non-blocking collectives are used limitations apply.

ACCELERATED MPI COLLECTIVES

HPC-X 2.10 + UCX 1.12.0 on Selene

Selene: https://www.top500.org/system/179842/

ACCELERATED MPI COLLECTIVES

HPC-X 2.10 + UCX 1.12.0 + NCCL 2.11.4 on Selene

Selene: https://www.top500.org/system/179842/

NCCL ACCELERATED MPI COLLECTIVES

Limitations

NCCL accelerated collective do not work with MPS!

Thread Mode	Blocking Collectives	Non-Blocking Collectives
thread-single	Any communicator	COMM_WORLD only
thread-funneled	Any communicator	COMM_WORLD only
thread-serialized	Any communicator	COMM_WORLD only
thread-multiple	COMM_WORLD only	COMM_WORLD only

Limitations apply to HCOLL and similar limitations will apply to UCC!

HCOLL NCCL backend requires opt-in by setting: `HCOLL_CUDA_BCOL=nccl`

CAVEAT: HCOLL does not enforce these restrictions so apps my run into deadlocks if they opt into the NCCL backend and use an unsupported combination!

UNIFIED COLLECTIVE COMMUNICATION

UCC

“UCC is a collective communication operations API and library that is flexible, complete, and feature-rich for current and emerging programming models and runtimes.” from https://github.com/openucx/ucc

UCC can leverage multiple Team Layers (TLs) for optimized collectives. Relevant TLs for for GPU data are:

`cuda`: TL supporting CUDA device memory exploiting NVLINK connections between GPUs.

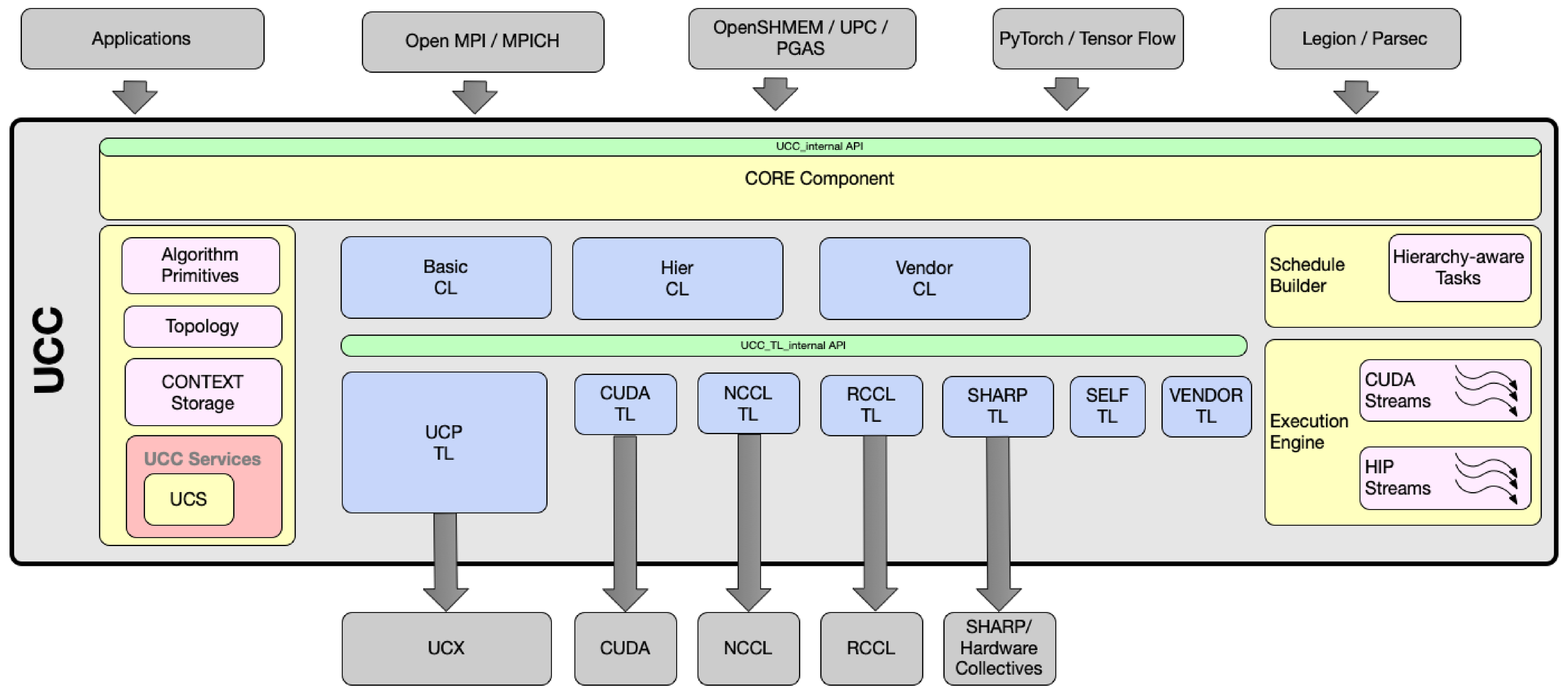
`nccl`: TL leveraging NCCL for collectives on CUDA device memory. In many cases, UCC collectives are directly mapped to NCCL collectives. If that is not possible, a combination of NCCL collectives might be used.

`self`: TL to support collectives with only 1 participant.

`ucp`: TL building on UCP point to point communication routines from UCX. This is the most general TL which supports all memory types. If required computation happens local to the memory, e.g. for CUDA device memory CUDA kernels are used for computation.

`sharp`: TL leveraging the NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)™ in-network computing features to accelerate inter-node collectives.

From : https://github.com/openucx/ucc/blob/master/docs/user_guide.md



ACCELERATED MPI COLLECTIVES

Hands On

See `01-MPI/03-Collectives/Instructions.md`: for experiments:

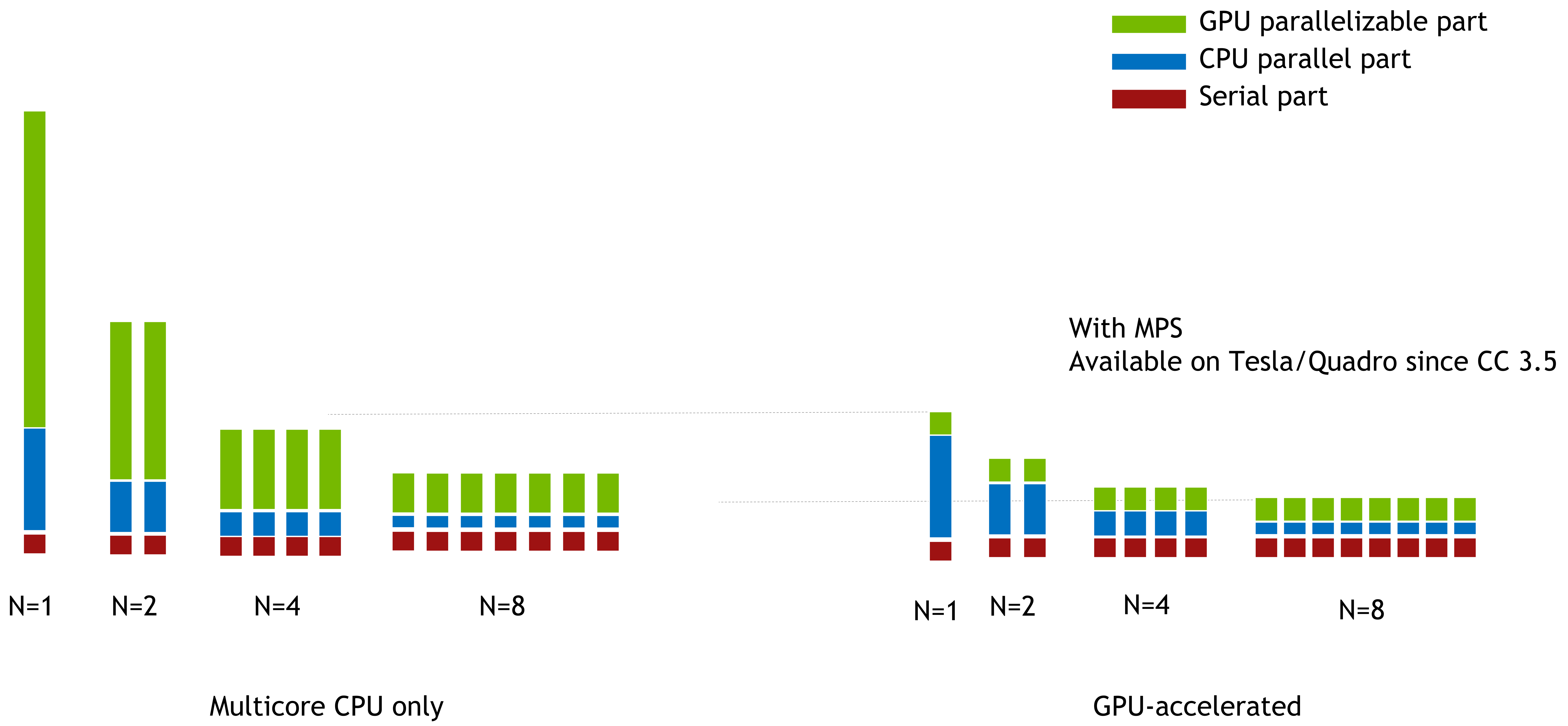
- Run with and without UCC (`OMPI_MCA_coll_ucc_enable=0|1`) and compare performance
- Prioritize or disable UCC Team Layers (TLs)
  - `UCC_TL_NCCL_TUNE=allreduce:cuda:inf`: prioritize NCCL TL for allreduce on CUDA device memory buffers
  - `UCC_TL_CUDA_TUNE=0`: generally deprioritize CUDA TL

GPU ACCELERATION OF LEGACY MPI APPS

- Typical legacy application
  - MPI parallel
  - Single or few threads per MPI rank (e.g. OpenMP)
- Running with multiple MPI ranks per node
- GPU acceleration in phases
  - Proof of concept prototype, …
  - Great speedup at kernel level
- Application performance misses expectations

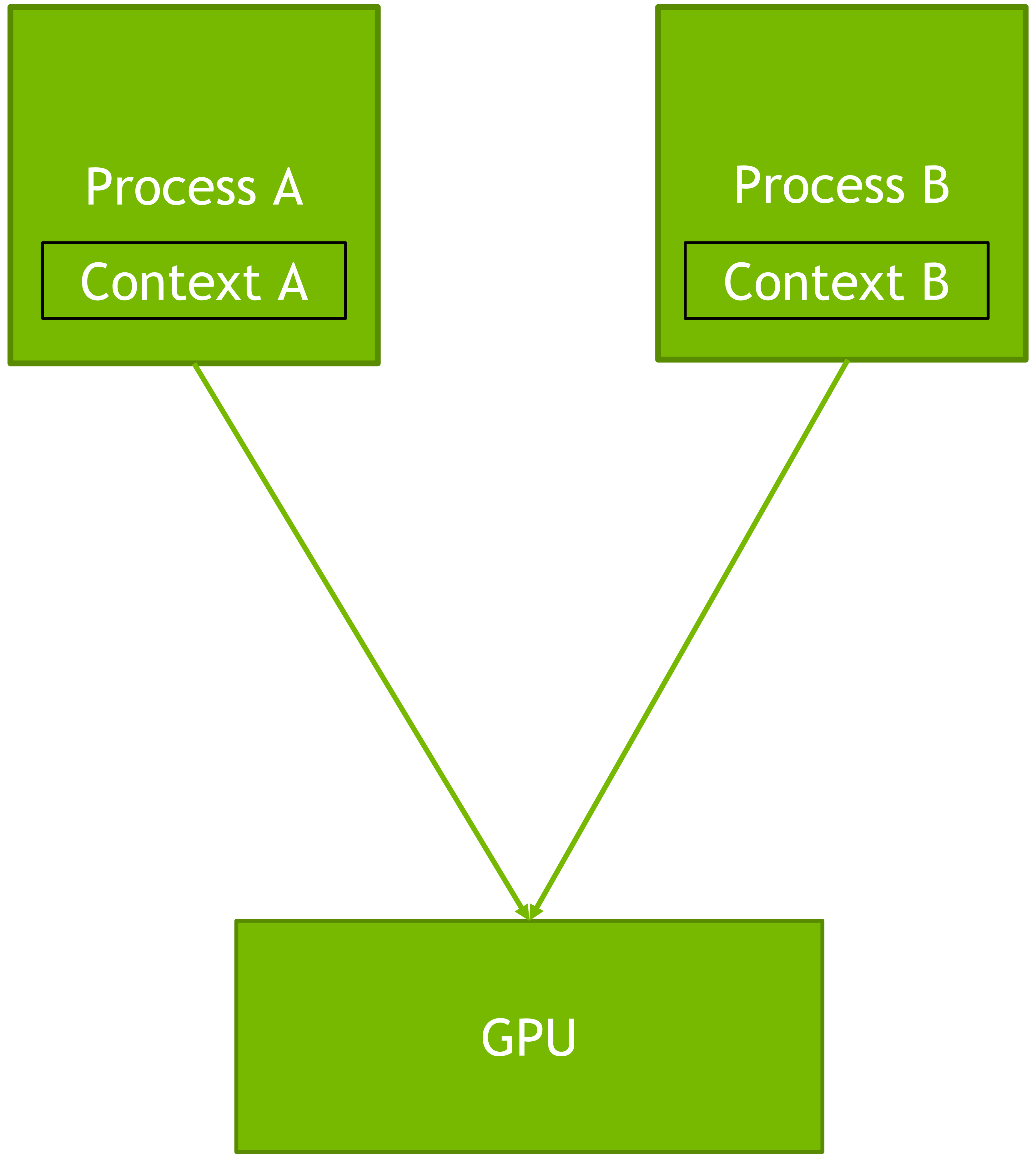
MULTI PROCESS SERVICE (MPS)

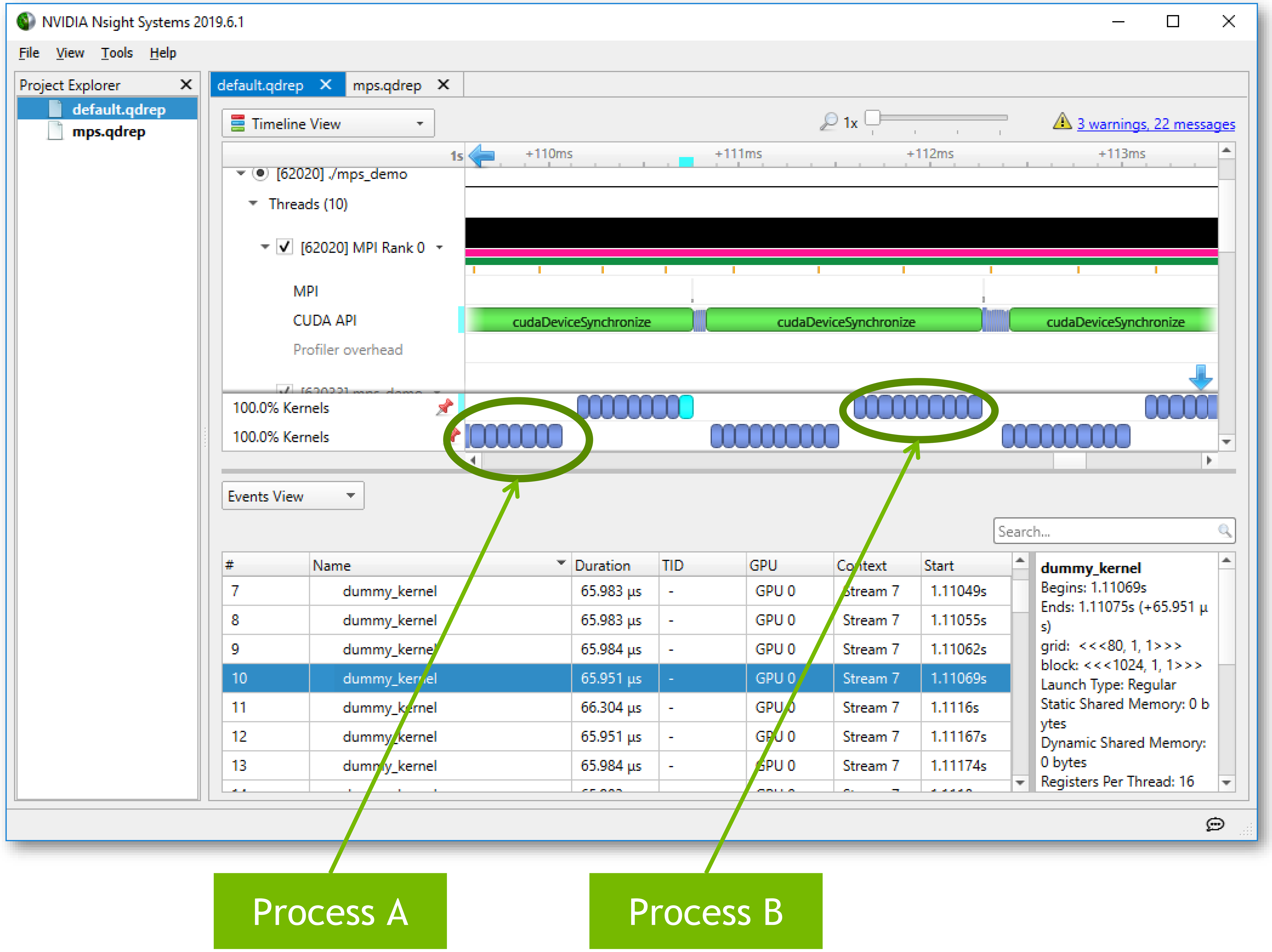
For Legacy MPI Applications



PROCESS SHARING GPU WITHOUT MPS

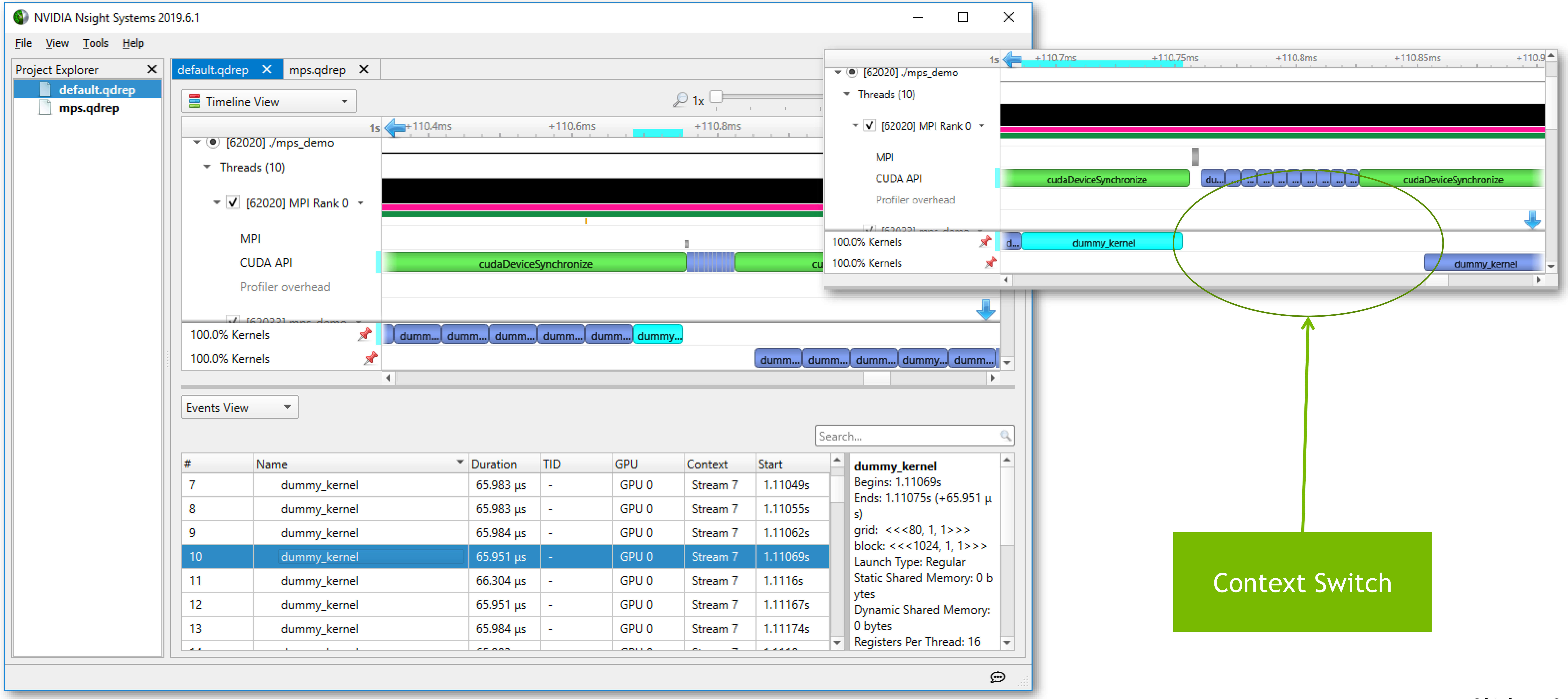
No Overlap





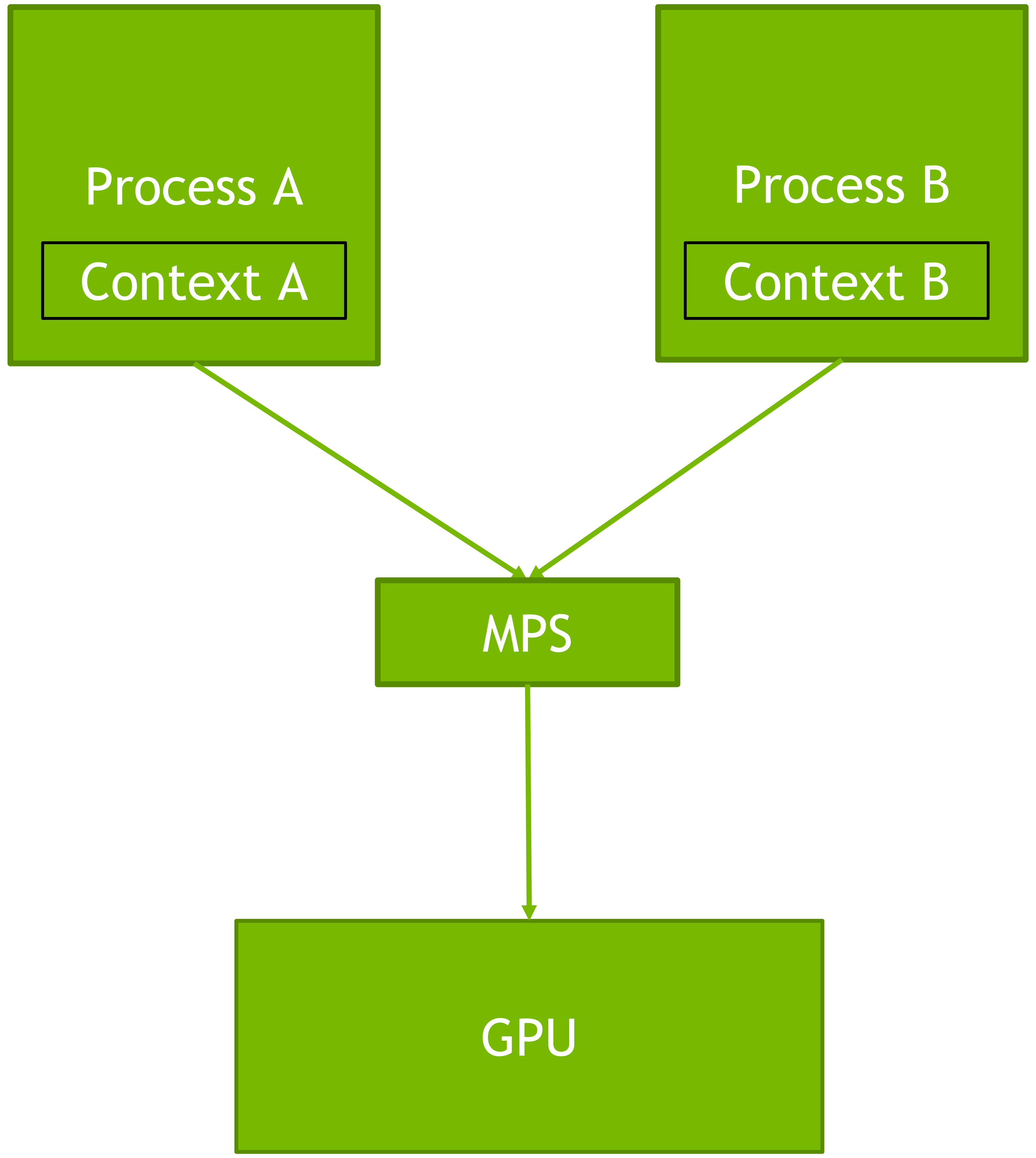
PROCESSES SHARING GPU WITHOUT MPS

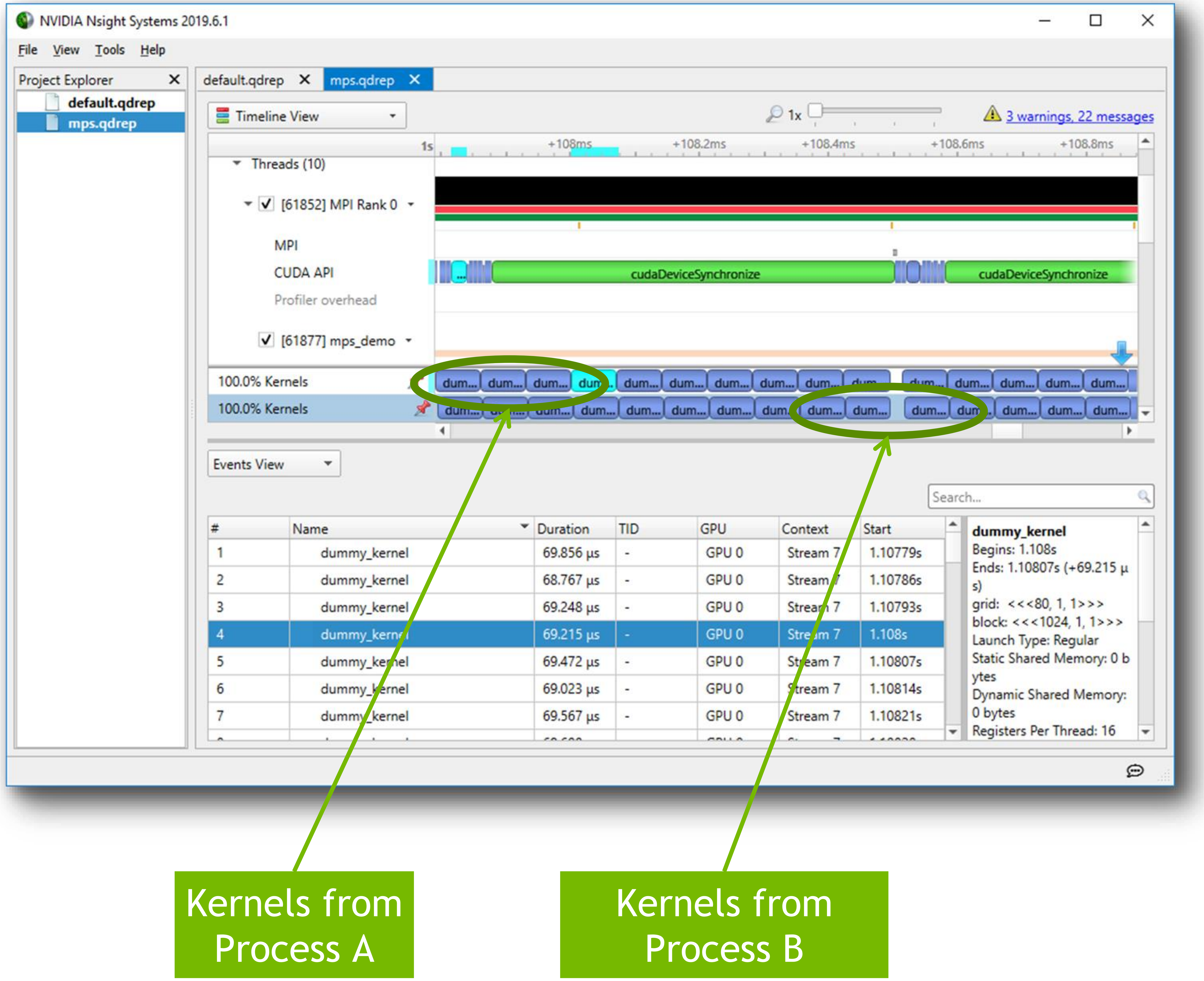
Context Switch Overhead



PROCESS SHARING GPU WITH MPS

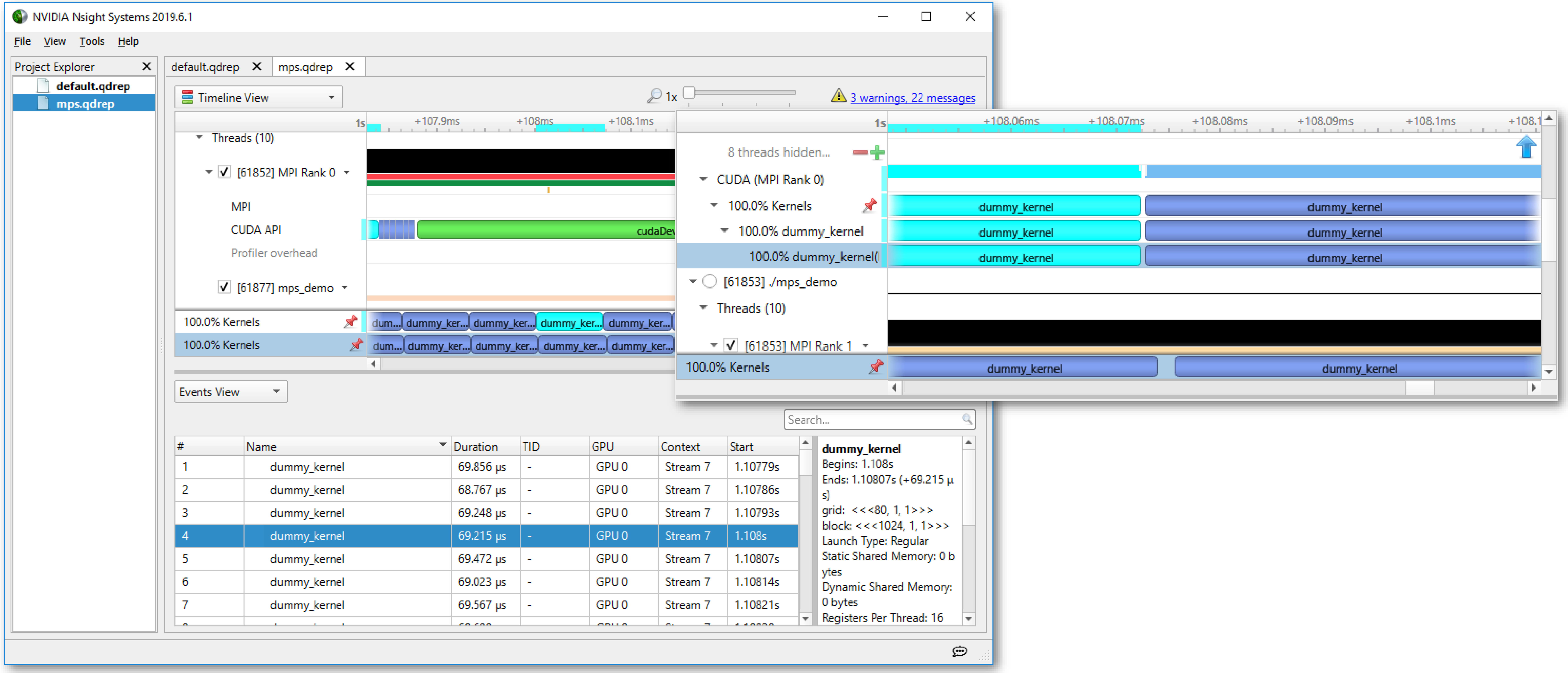
Maximum Overlap



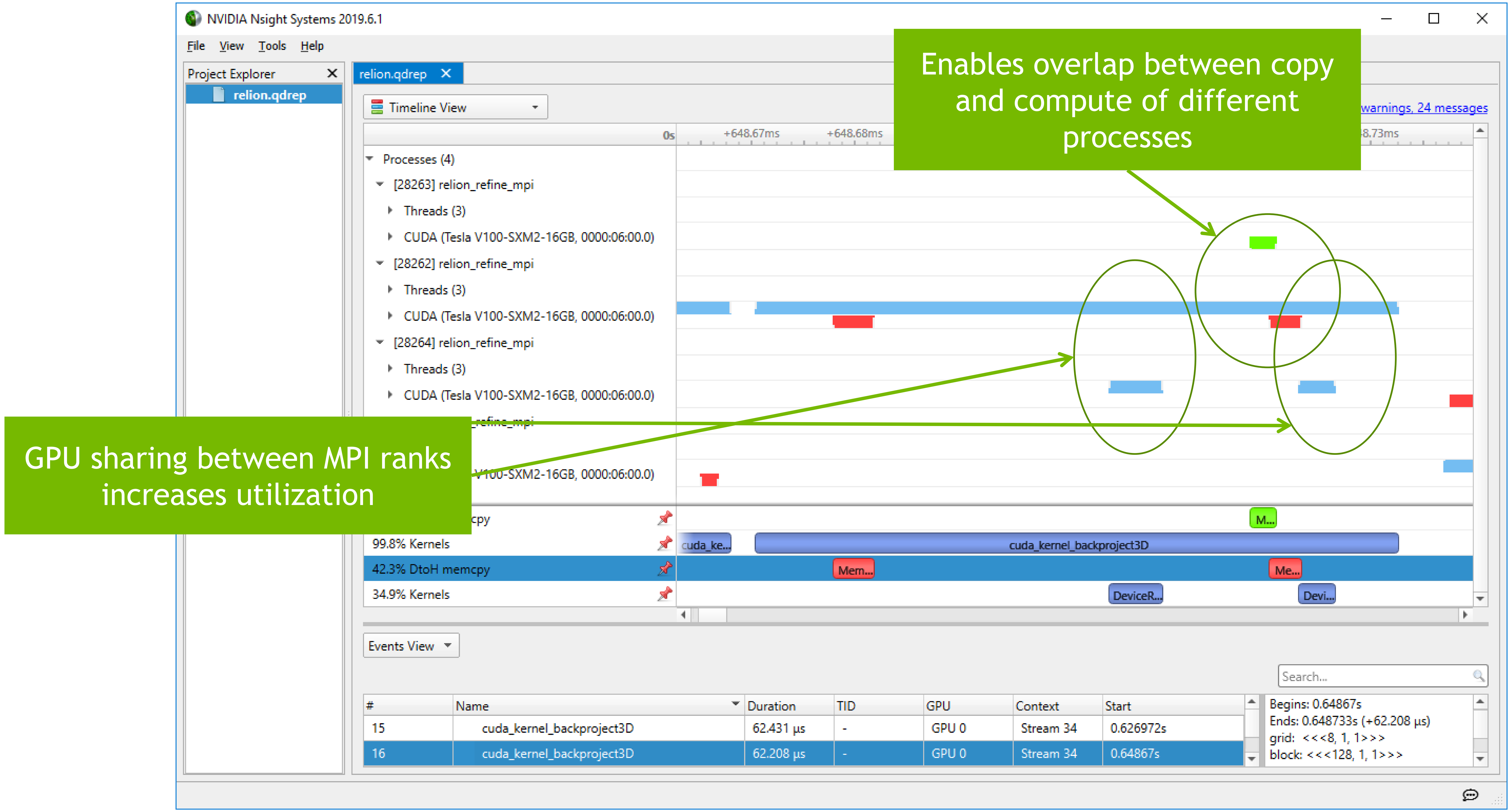


PROCESSES SHARING GPU WITH MPS

No Context Switch Overhead



MPS CASE STUDY: RELION



USING MPS

No application modifications necessary

Not limited to MPI applications

MPS control daemon

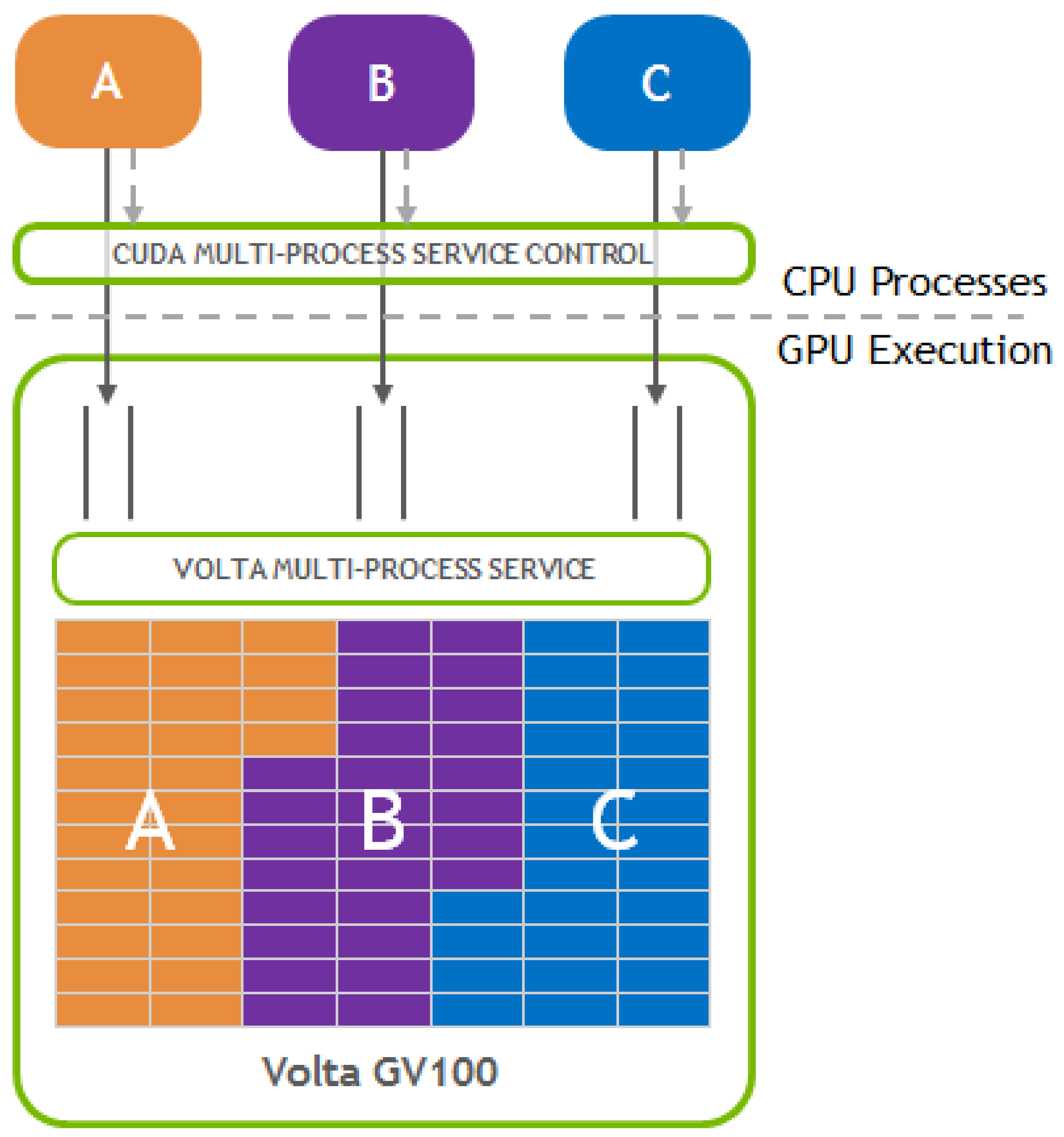
Spawn MPS server upon CUDA application startup

On CC 7.0+ supports limited execution resource provisioning for Quality of Service (QoS). -> `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE`

```
nvidia-smi -c EXCLUSIVE_PROCESS
nvidia-cuda-mps-control -d
```

Slurm option on JURECA/JUWELS-Booster:

```
--cuda-mps              Start the CUDA Multi-Process server
```



TOOLS FOR CUDA APPLICATIONS

Memory checking: `compute-sanitizer`

Debugging: `cuda-gdb`

Profiling: NVIDIA Nsight Systems

APPROACHES FOR MULTI-PROCESS TOOLS

- Tools usually run on a single process – adapt for highly distributed applications?
  - Bugs in parallel programs are often serial bugs in disguise
- Common MPI paradigm: Workload distributed; bug classes/performance similar for all processes
  - Not: Load imbalance, parallel race conditions; require parallel tools
- Ergo: Run tool N times in parallel, have N output files, only look at 1 (or 2, ...)
- `%q{ENV_VAR}` supported by all the NVIDIA tools discussed here, embed environment variable in file name
  - `ENV_VAR` should be one set by the process launcher, unique ID
  - Evaluated only once tool starts running (on compute node) – not when launching job
- Other tools: Use a launcher script, for late evaluation

OpenMPI:
`OMPI_COMM_WORLD_RANK`
`OMPI_COMM_WORLD_LOCAL_RANK`

MVAPICH2:
`MV2_COMM_WORLD_RANK`
`MV2_COMM_WORLD_LOCAL_RANK`

Slurm:
`SLURM_PROCID`
`SLURM_LOCALID`

https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.2-userguide.html#x1-32100013
https://slurm.schedmd.com/srun.html#SECTION_OUTPUT-ENVIRONMENT-VARIABLES

Credit Markus Hrywniak

COMPUTE-SANITIZER

Functional correctness checking suite for GPU

- `compute-sanitizer` is a collection of tools
- `memcheck` (default) tool comparable to Valgrind's memcheck.
- Other tools include
  - `racecheck`: shared memory data access hazard detector
  - `initcheck`: uninitialized device global memory access detector
  - `synccheck`: identify whether a CUDA application is correctly using synchronization primitives

- Example run:

```
srun -n 4 compute-sanitizer \
--log-file jacobi.%q{SLURM_PROCID}.log \
--save jacobi.%q{SLURM_PROCID}.compute-sanitizer \
./jacobi -niter 10
```

- Stores (potentially very long) text output in `*.log` file, raw data separately, once per process.
- Compile with `–lineinfo` to get generate line correlation for device code

Credit Markus Hrywniak

COMPUTE-SANITIZER

Anatomy of an error

- Look into log file, or use `compute-sanitizer --read <save file>`
- Actual output can be very long, if many GPU threads produce (similar) errors.

```
========= COMPUTE-SANITIZER
[...]
========= Invalid __global__ write of size 4 bytes
=========     at 0x6d0 in mpi/jacobi_kernels.cu:60:initialize_boundaries(float*, float*, float,
                                                          int, int, int, int)
=========     by thread (1,0,0) in block (32,0,0)
=========     Address 0x14fb88020000 is out of bounds
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame: [0x20d6ea]
=========           in libcuda.so.1
=========     Host Frame: [0x115ab]
[...]
=========
========= ERROR SUMMARY: 10 errors
```

- We introduced an off-by-one error in line 60 ourselves:

```
a_new[iy * nx + (nx - 1) + 1] = y0;
```

Credit Markus Hrywniak

NVIDIA

USING CUDA-GDB WITH MPI

- Compile with host `-g` and device debug symbols `-G` (slow!)
- Launcher (`mpirun/srun/...`) complicates starting process inside debugger
- Workaround: Attach later

```
#include <unistd.h>
if (rank == 0) {
    char name[255]; gethostname(name, sizeof(name)); bool attached;
    printf("rank %d: pid %d on %s ready to attach\n", rank, getpid(), name);
    while (!attached) { sleep(5); }
}
```

- Launch process, sleep on particular rank

```
$ srun -n 4 ./jacobi -niter 10
rank 0: pid 28920 on jwb0001.juwels ready to attach
```

- Then attach from another terminal (may need more flags)

```
[jwlogin]$ srun -n 1 --jobid ${JOBID} --pty bash -i
[jwb0001]$ cuda-gdb --attach 28920
```

- Wake up sleeping process and continue debugging normally

```
(cuda-gdb) set var attached=true
```

Credit Markus Hrywniak

USING CUDA-GDB WITH MPI

Environment variables for easier debugging

- Automatically wait for attach on exception without code changes:

```
$ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 srun ./jacobi -niter 10
Single GPU jacobi relaxation: 10 iterations on 16384 x 16384 mesh with norm check every 1 iterations
jwb0129.juwels: The application encountered a device error and CUDA_DEVICE_WAITS_ON_EXCEPTION is set. You
can now attach a debugger to the application (PID 31562) for inspection.
```

- Same as before, go to node and attach `cuda-gdb`:

```
$ cuda-gdb --pid 31562
CUDA Exception: Warp Illegal Address
The exception was triggered at PC 0x508ca70 (jacobi_kernels.cu:88)

Thread 1 "jacobi" received signal CUDA_EXCEPTION_14, Warp Illegal Address.
[Switching focus to CUDA kernel 0, grid 4, block (0,0,0), thread (0,20,0), device 0, sm 0, warp 21, lane 0]
0x000000000508ca80 in jacobi_kernel<32, 32><<<(512,512,1),(32,32,1)>>> (/*...*/) at jacobi_kernels.cu:88

88                real foo = *((real*)nullptr);
```

Credit Markus Hrywniak

DEBUGGING MPI+CUDA APPLICATIONS

More environment variables for offline debugging

- With `CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1` core dumps are generated in case of an exception
  - `CUDA_ENABLE_LIGHTWEIGHT_COREDUMP=1` does not dump application memory - faster
  - Can be used for post-mortem debugging
  - Helpful if live debugging is not possible
- Enable/Disable CPU part of core dump (enabled by default)
  - `CUDA_ENABLE_CPU_COREDUMP_ON_EXCEPTION`
- Specify name of core dump file with `CUDA_COREDUMP_FILE`
- Open GPU

```
(cuda-gdb) target cudacore core.cuda
```

- Open CPU+GPU

```
(cuda-gdb) target core core.cpu core.cuda
```

https://docs.nvidia.com/cuda/cuda-gdb/index.html#gpu-coredump

Credit Markus Hrywniak

EXAMPLE: OPENING A CORE DUMP

- Running and generating the core file

```
$ CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1 CUDA_ENABLE_LIGHTWEIGHT_COREDUMP=1 srun ./jacobi -niter 10
Single GPU jacobi relaxation: 10 iterations on 16384 x 16384 mesh with norm check every 1 iterations
srun: error: jwb0021: tasks 0-3: Aborted (core dumped)

$ ls core*
core.jwb0021.juwels.23959  core_1633801834_jwb0021.juwels_23959.nvcudmp ...
```

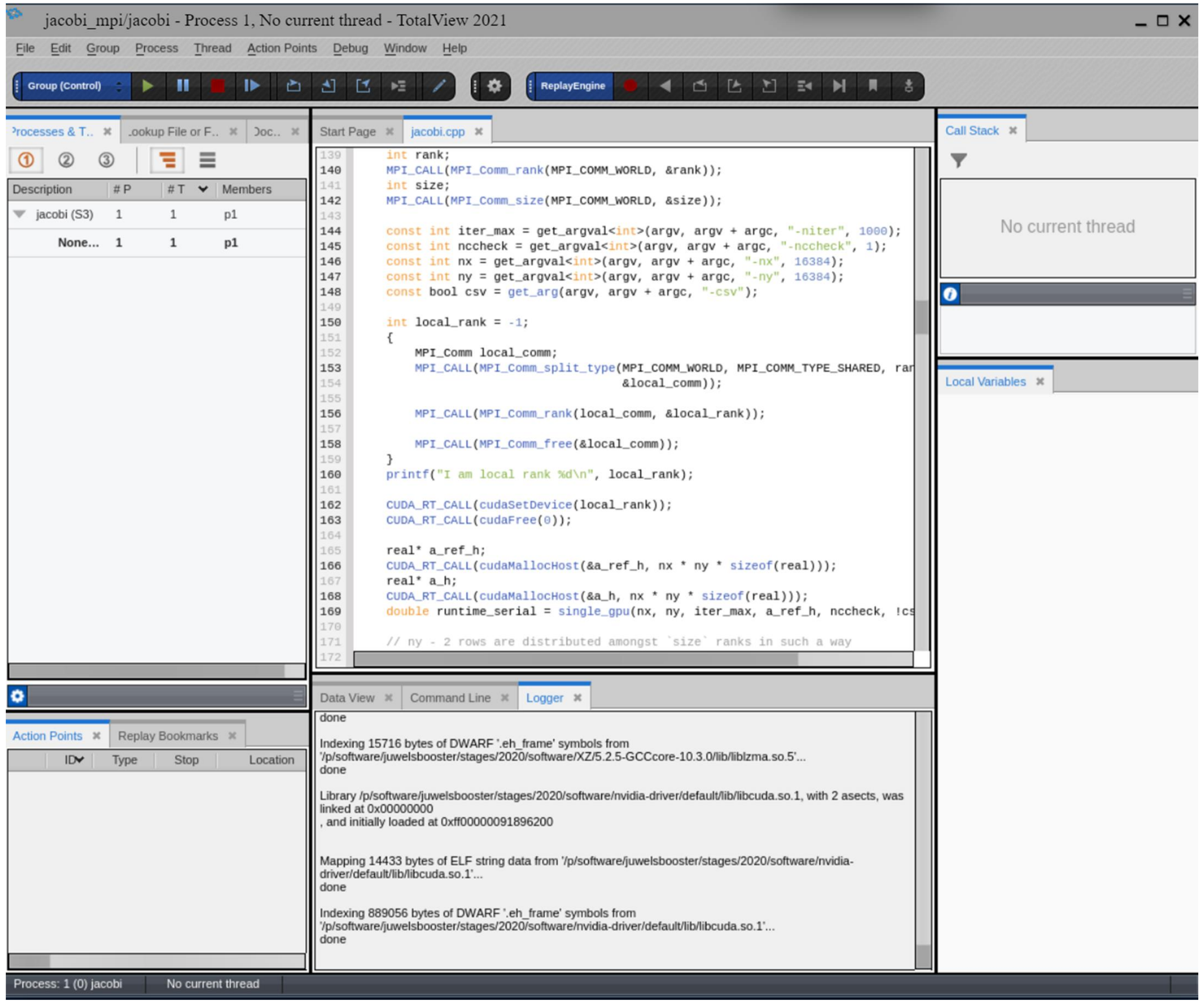
- And opening the core dump in `cuda-gdb`

```
(cuda-gdb) target cudacore core_1633801834_jwb0021.juwels_23959.nvcudmp
Opening GPU coredump: core_1633801834_jwb0021.juwels_23959.nvcudmp

[New Thread 23979]

warning: No exception was found on the device

[Current focus set to CUDA kernel 0, grid 4, block (0,0,0), thread (0,2,0), device 0, sm 0, warp 0, lane 0]

#0  0x00000000057e1ae0 in void jacobi_kernel<32, 32>(float*, float const*, float*, int, int, int, bool)

   <<<(512,512,1),(32,32,1)>>> ()

    at multi-gpu-programming-models/mpi/jacobi_kernels.cu:87

87              real foo = *((real*)nullptr);

(cuda-gdb)
```

Credit Markus Hrywniak

SPECIALIZED PARALLEL DEBUGGERS

with CUDA Support

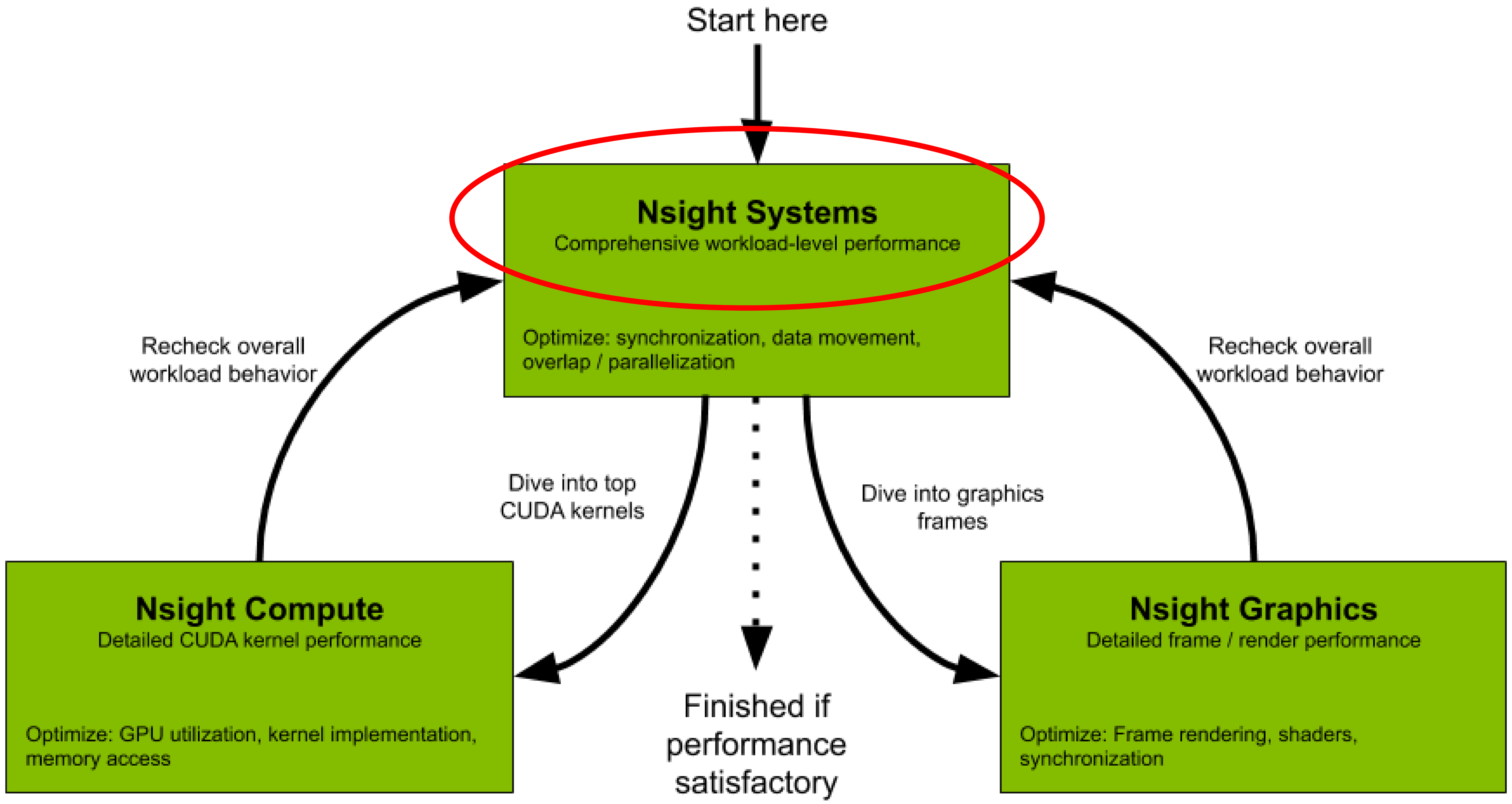


- `cuda-gdb` can debug multiple processes (`add-inferior`), although...
- For truly parallel bugs (e.g. multi-node, multi-process race conditions), third-party tools offer more convenience
  - Or enable „live“ analysis in the first place
- ARM DDT
- Perforce TotalView (screenshot)

THE NSIGHT SUITE COMPONENTS

How the pieces fit together

- Nsight **Systems**: Coarse-grained, whole-application
- Nsight **Compute**: Fine-grained, kernel-level
- NVTX: Support and structure across tools
- Main purpose: Performance optimization
  - But at their core, advanced *measurement* tools



Credit Markus Hrywniak

USING NSIGHT SYSTEMS

Recording with the CLI

- Use the command line

```
srun nsys profile --trace=cuda,nvtx,mpi --output=my_report.%q{SLURM_PROCID} ./jacobi -niter 10
```

- Inspect results: Open the report file in the GUI
  - Also possible to get details on command line
  - Either add `--stats` to profile command line, or: `nsys stats --help`

- Runs set of reports on command line, customizable (sqlite + `Python`):
  - Useful to check validity of profile, identify important kernels

```
Running [.../reports/gpukernsum.py jacobi_metrics_more-nvtx.0.sqlite]...

Time(%)  Total Time (ns)  Instances  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)          Name

-------  ---------------  ---------  ---------  ---------  --------  --------  -----------  ---------------------
   99.9         36750359         20  1837518.0  1838466.5    622945   3055044    1245121.7  void jacobi_kernel
    0.1            22816          2    11408.0    11408.0      7520     15296       5498.5  initialize_boundaries
```

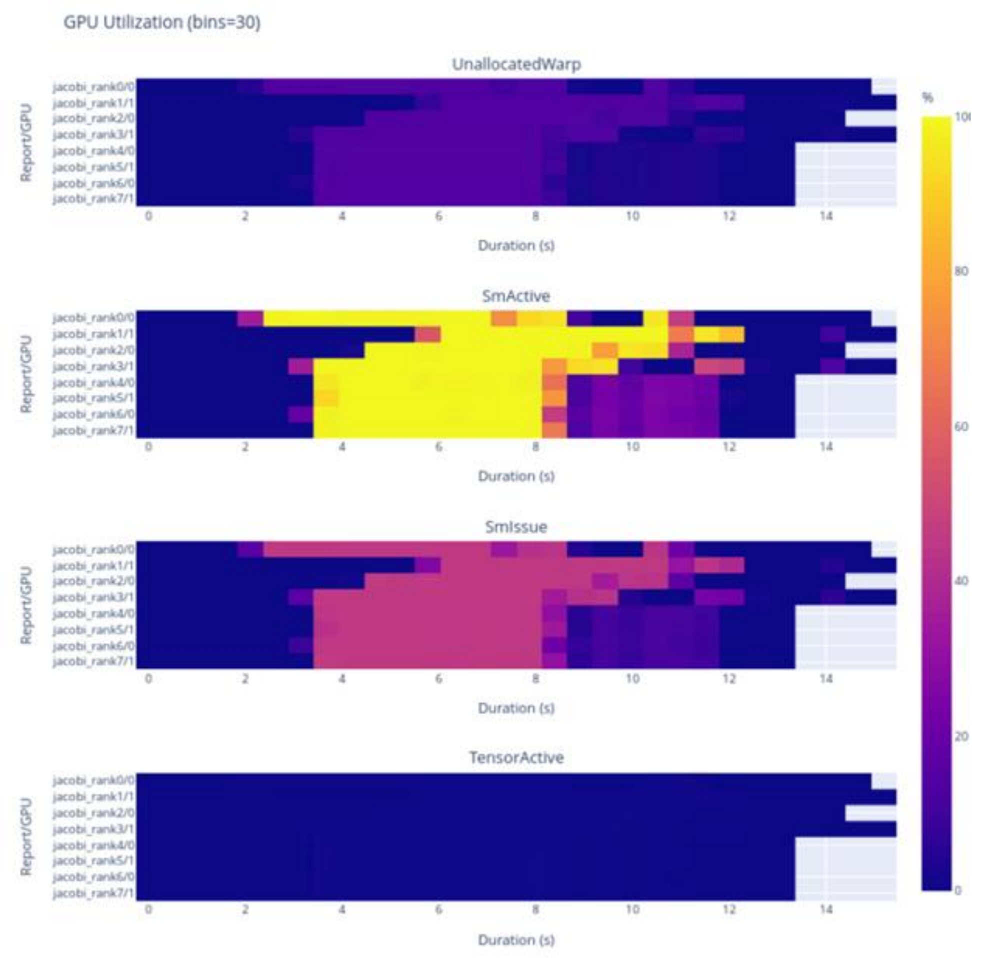
Credit Markus Hrywniak

USING NSIGHT SYSTEMS

visualize reports

PROFILING AT SCALE

- Detecting issues at scale of thousands of GPUs (and processes)
  - Need to slice and dice data, too much to make sense of raw data
- Nsight Systems 2023.2.1 introduced Multi-Report Analysis as a preview feature:
  - Multi-Node Analysis - When you run Nsight Systems across a cluster, it typically generates one result file per rank on the cluster. While you can load multiple result files into the GUI for visualization, this analysis system allows you to run statistical analysis across all of the result files.
  - Multi-Pass Analysis - Some features in Nsight Systems cannot be run together due to overhead or hardware considerations. For example, there are frequently more CPU performance counters available than your CPU has registers. Using this analysis, you could run multiple runs with different sets of counters and then analyze the results together.
  - Multi-Run Analysis - Sometimes you want to compare two runs that were not taken at the same time together. Perhaps you ran the tool on two different hardware configurations and want to see what changed. Perhaps you are doing regression testing or performance improvement analysis and want to check your status. Comparing those result files statistically can show patterns.
- See “Optimizing at Scale: Investigating Hidden Bottlenecks for Multi-Node Workloads” GTC 2023 Talk for more information. Available on NVIDIA On-Demand: https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s51421/



COMMUNITY PROFILING TOOLS

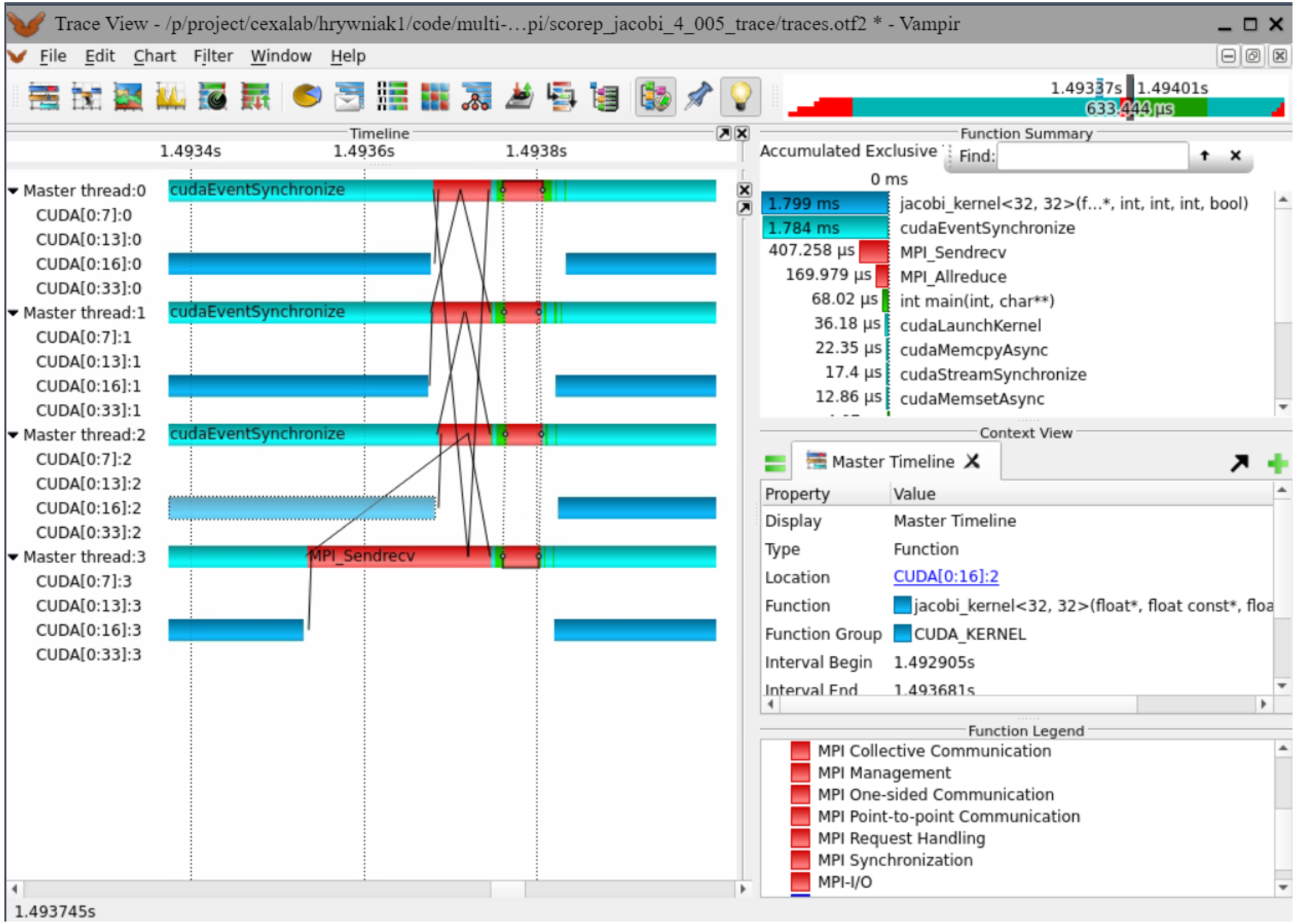
Specialized for large-scale distributed analysis

- Common measurement/instrumentation infrastructure: *Score-P*
  - Prefix all compilation/linker commands with `scorep –cuda`
- GPU data integration
  - CUDA profiling tools interface (CUPTI)
- Run the application to collect...
  - profiling data
  - tracing data
- ... and analyze with
  - TAU
  - Vampir
  - (selection of tools not exhaustive)
- Tracing in particular: Careful tuning to keep overhead low (filtering)

Credit Markus Hrywniak

VAMPIR

- Analyze multi-process patterns
- What you can see in screenshot
  - Main timeline
  - Function summary
- Example analysis: Pinpoint MPI message relationships
  - e.g. late sender issues
- https://vampir.eu/



Credit Markus Hrywniak

COMMUNICATION + COMPUTATION OVERLAP

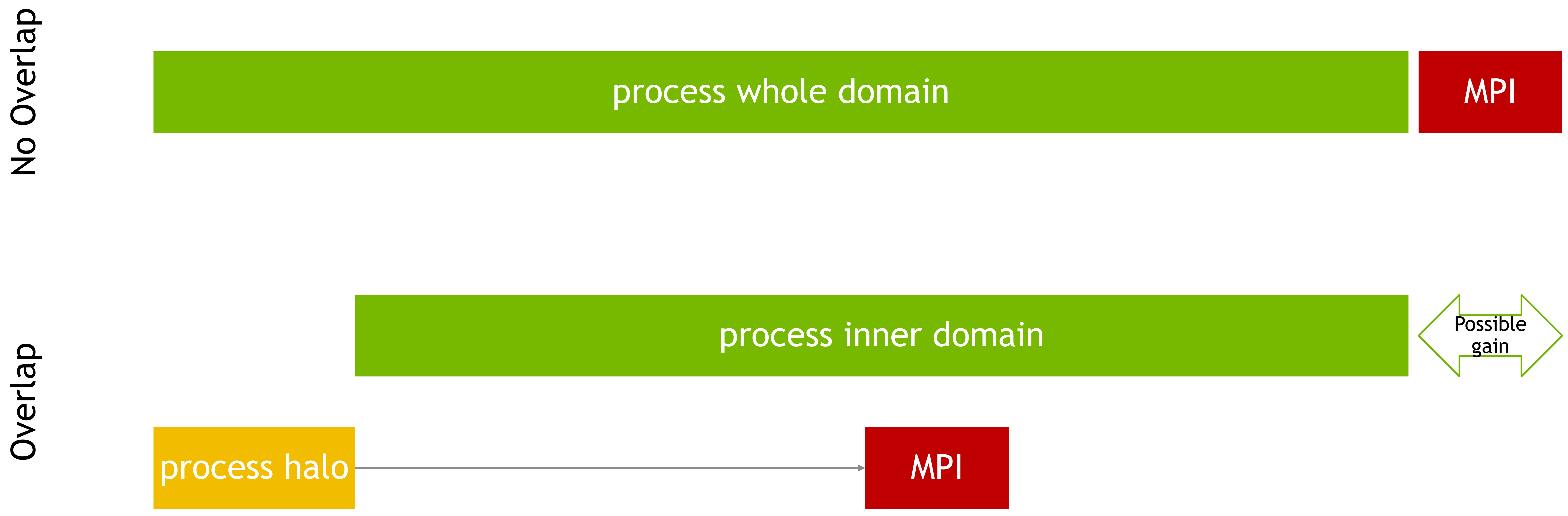
ParaStationMPI 5.4.10-1 – JUWELS Booster – NVIDIA A100 40 GB – Jacobi on 17408x17408

Source: *https://github.com/NVIDIA/multi-gpu-programming-models*
JUWELS Booster: *https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html*

MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI 8 NVIDIA A100 40GB on JUWELS Booster

COMMUNICATION + COMPUTATION OVERLAP



MPI COMMUNICATION + COMPUTATION OVERLAP

CUDA

```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);


const int top = rank > 0 ? rank - 1 : (size - 1);
const int bottom = (rank + 1) % size;

cudaStreamSynchronize(push_top_stream);
MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0,
             a_new + (iy_end * nx), nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaStreamSynchronize(push_bottom_stream);
MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0,
             a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

MPI COMMUNICATION + COMPUTATION OVERLAP

OpenACC

```
#pragma acc parallel loop present ( a_new, a ) async(1)
for ( ... )
    //Process top boundary
#pragma acc parallel loop present ( a_new, a ) async(2)
for ( ... )
    //Process bottom boundary
#pragma acc parallel loop present ( a_new, a ) async(3)
for ( ... )
    //Process inner domain
#pragma acc wait(1)     //wait for top boundary
#pragma acc host_data use_device ( a_new ) {
    MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0,
                 a_new + (iy_end * nx), nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

}
#pragma acc wait(2)     //wait for bottom boundary
#pragma acc host_data use_device ( a_new ) {
    MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0,
                 a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
#pragma acc wait            //wait for iteration to finish
```

MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster

COMMUNICATION + COMPUTATION OVERLAP

ParaStationMPI 5.4.10-1 – JUWELS Booster – NVIDIA A100 40 GB – Jacobi on 17408x17408

Source: https://github.com/NVIDIA/multi-gpu-programming-models
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

MPI COMMUNICATION + COMPUTATION OVERLAP

```
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);

const int top = rank > 0 ? rank - 1 : (size - 1);
const int bottom = (rank + 1) % size;


cudaStreamSynchronize(push_top_stream);
MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0,
             a_new + (iy_end * nx), nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaStreamSynchronize(push_bottom_stream);
MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0,
             a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster

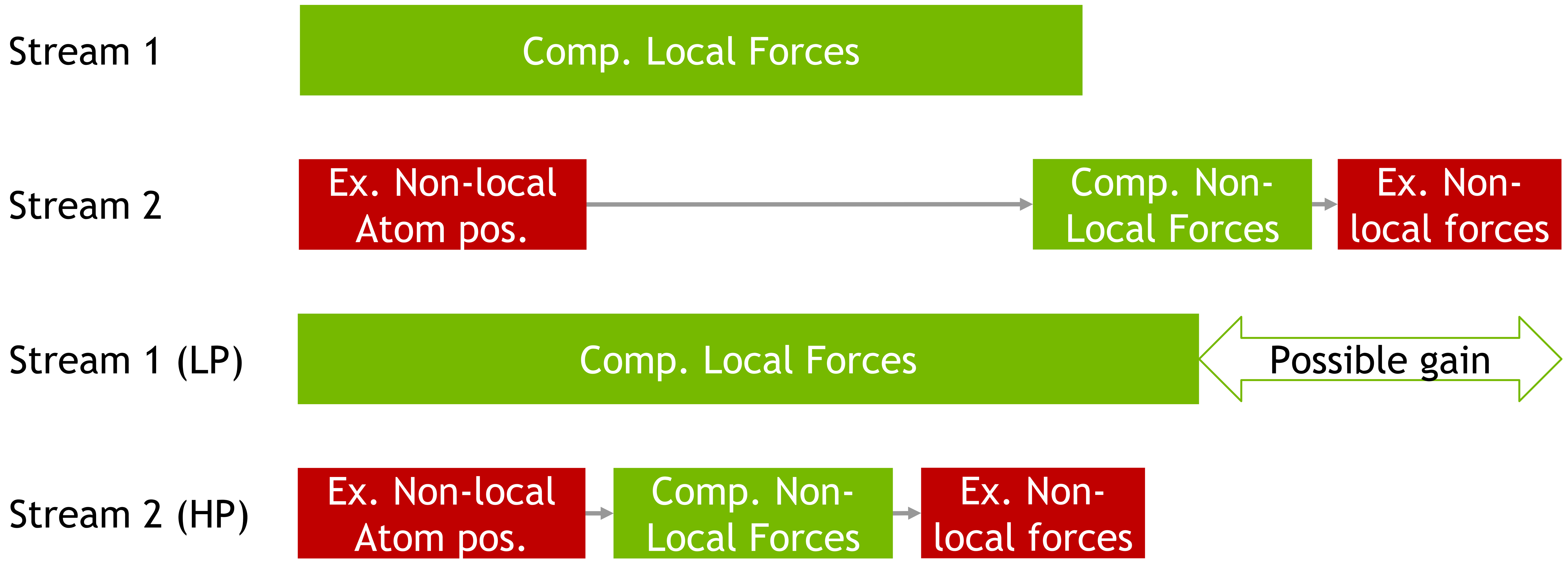
NVIDIA

HIGH PRIORITY STREAMS

Improve scalability with high priority streams (available on CC 3.5+)

```
cudaStreamCreateWithPriority ( cudaStream_t* pStream, unsigned int  flags, int  priority )
```

Use-case: MD-Simulations



MPI COMMUNICATION + COMPUTATION OVERLAP

with high priority streams

```
int leastPriority = 0;
int greatestPriority = leastPriority;
cudaDeviceGetStreamPriorityRange(&leastPriority, &greatestPriority);

cudaStream_t compute_stream;
cudaStream_t push_top_stream;
cudaStream_t push_bottom_stream;

cudaStreamCreateWithPriority(&compute_stream, cudaStreamDefault, leastPriority);
cudaStreamCreateWithPriority(&push_top_stream, cudaStreamDefault, greatestPriority);
cudaStreamCreateWithPriority(&push_bottom_stream, cudaStreamDefault, greatestPriority);
```

MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster

NVIDIA

COMMUNICATION + COMPUTATION OVERLAP

ParaStationMPI 5.4.10-1 – JUWELS Booster – NVIDIA A100 40 GB – Jacobi on 17408x17408

Source: https://github.com/NVIDIA/multi-gpu-programming-models
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

HANDS ON COMMUNICATION COMPUTATION OVERLAP

Hands On

See `01-MPI/04-Communication_Computation_Overlap/Instructions.md`:

- Profiles with srun nsys profile …
  - Import multiple profiles into Nsight Systems
- Implement communication computation overlap
- Optionally use high priority streams

DETECTING CUDA-AWARENESS

ParaStation MPI and OpenMPI (since 2.0.0) via `mpi-ext.h`

Macro:

```
MPIX_CUDA_AWARE_SUPPORT
```

Function for runtime decisions

```
MPIX_Query_cuda_support()
```

See http://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support

ParaStation MPI: `MPI_INFO_ENV`

```
MPI_Info_get(MPI_INFO_ENV, “cuda_aware”,
             sizeof(is_cuda_aware)-1,is_cuda_aware,
             &api_available);
```

THANK YOU FOR YOUR ATTENTION

Questions?

- SC/ISC Tutorial(s): Efficient Distributed GPU Programming for Exascale https://github.com/FZJ-JSC/tutorial-multi-gpu/
- NVIDIA On-Demand Content:
  - GTC 2022 Multi-GPU Programming with MPI (a Magnum IO Session) https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41018/
  - GTC 2024 Multi GPU Programming Models for HPC and AI https://www.nvidia.com/en-us/on-demand/session/gtc24-s61339/
- GitHub projects:
  - Multi GPU Programming Models: https://github.com/NVIDIA/multi-gpu-programming-models