MPI+X: Hybrid Programming

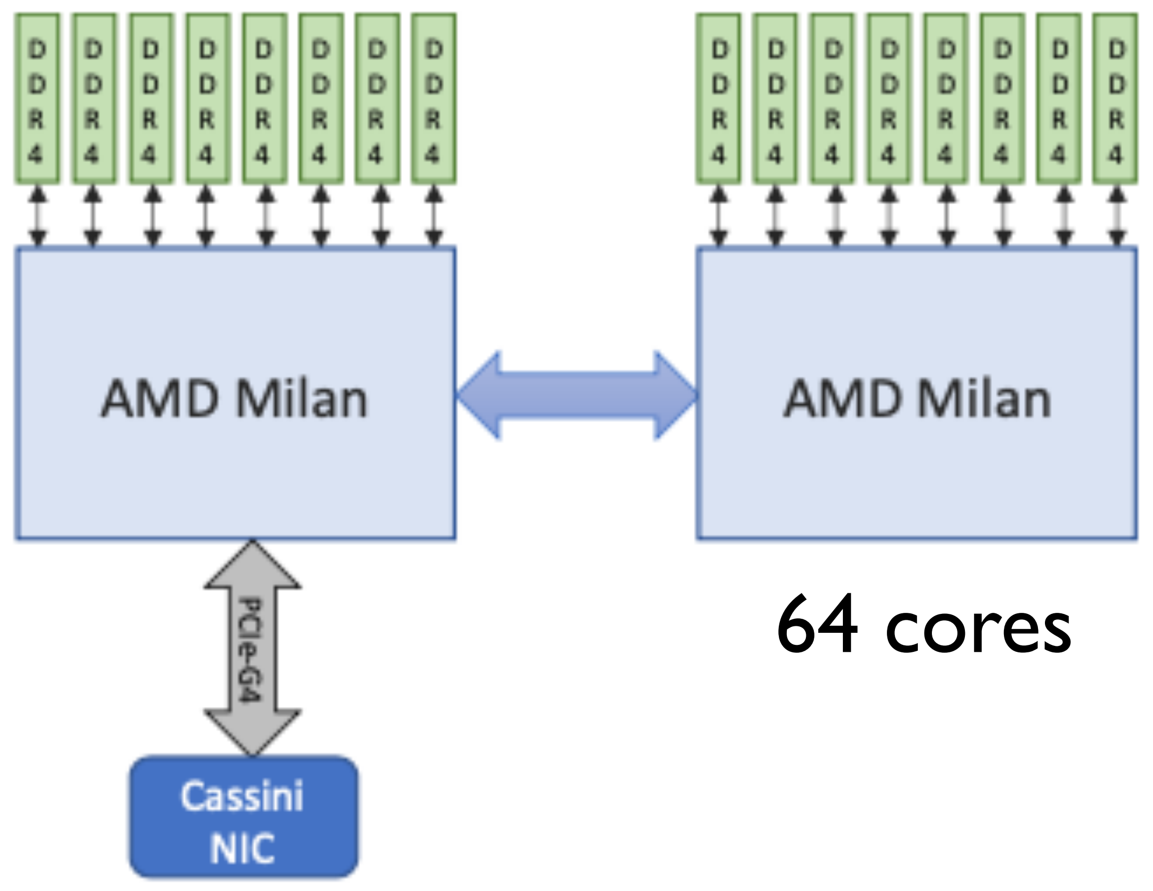
Abhinav Bhatele, Alan Sussman

UNIVERSITY OF MARYLAND

Announcements

- Assignment 1-3 grades have been released
- Assignment 4 has been posted, due on Nov 6 11:59 pm eastern time
- Extra credit assignments: same weight as other programming assignments for each section (10% of course grade for 416 students and 8% of course grade for 616 students)

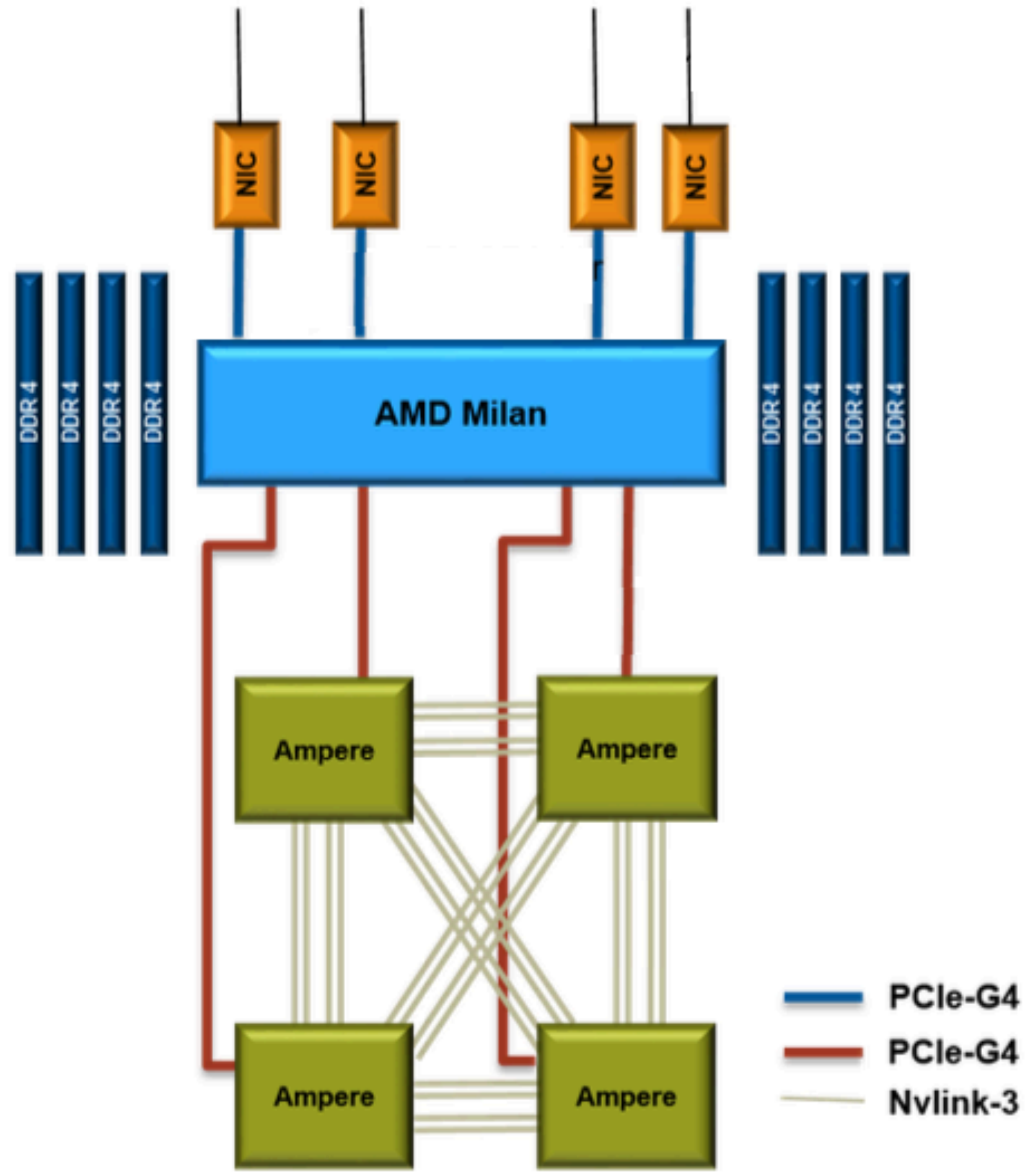
Complex node architectures



64 cores

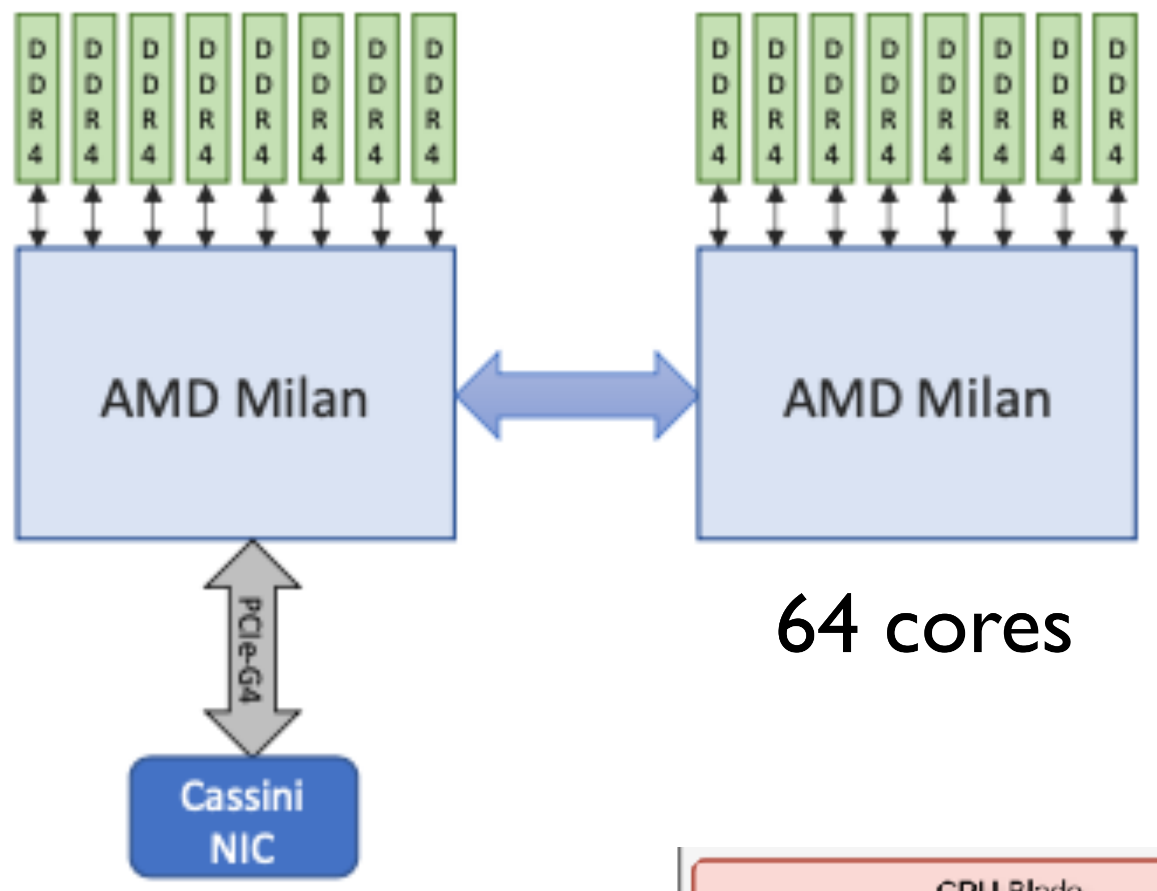
CPU Node

Perlmutter @ NERSC



GPU Node

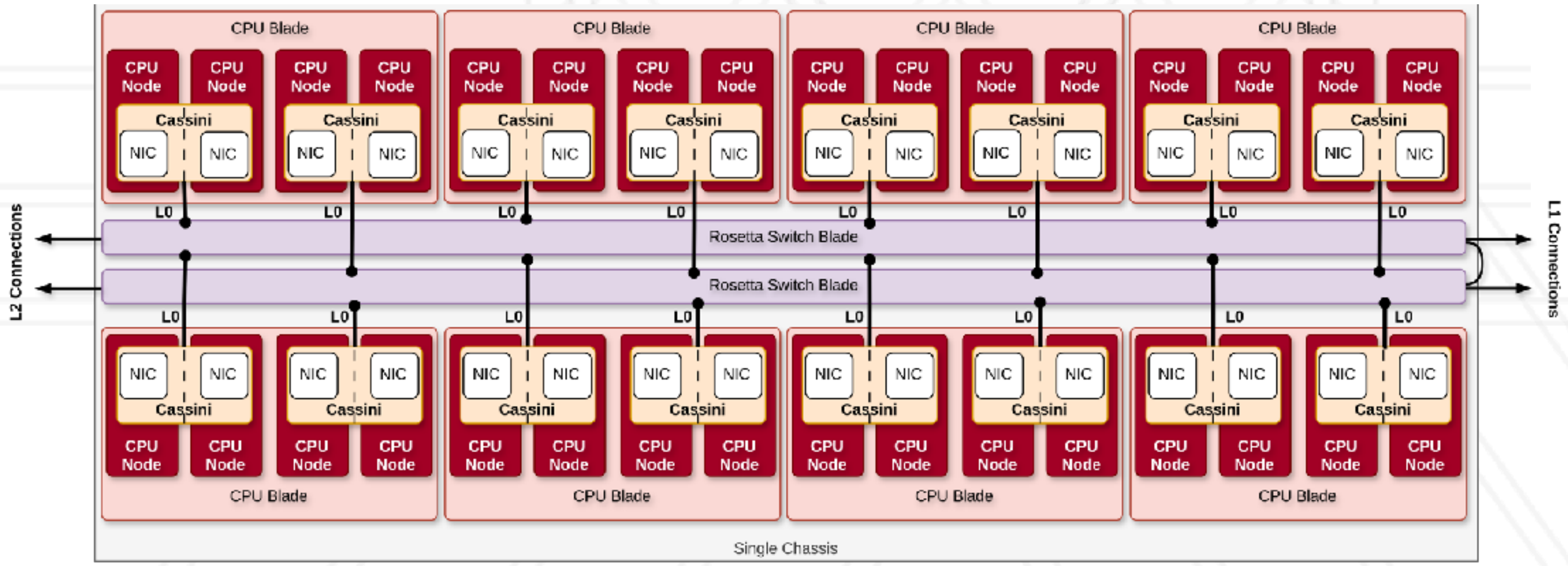
Complex node architectures

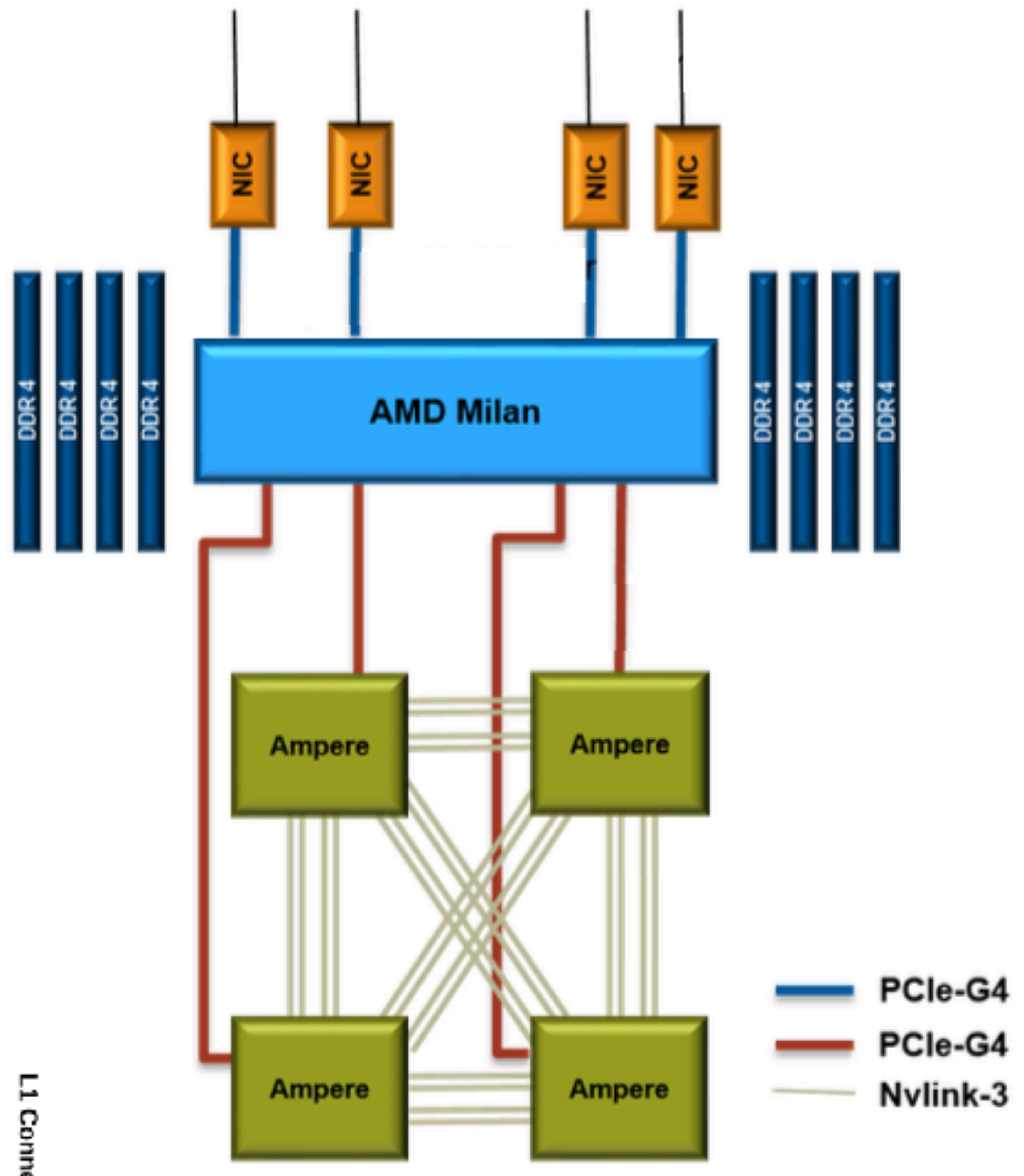


64 cores

CPU Node

Perlmutter @ NERSC





GPU Node

Several possible approaches

- Use MPI everywhere
  - Lets say you are running on 2 nodes with 128 cores each — create 256 MPI processes
- Use MPI+X where X handles within node parallelization
  - MPI handles inter-node communication
  - Also referred to as hybrid programming
- X could be OpenMP for CPUs and CUDA for GPUs
  - CPU nodes: Create 1 MPI process and 128 threads per node
  - GPU nodes: Create 1 MPI process per GPU and use CUDA for launching GPU kernels

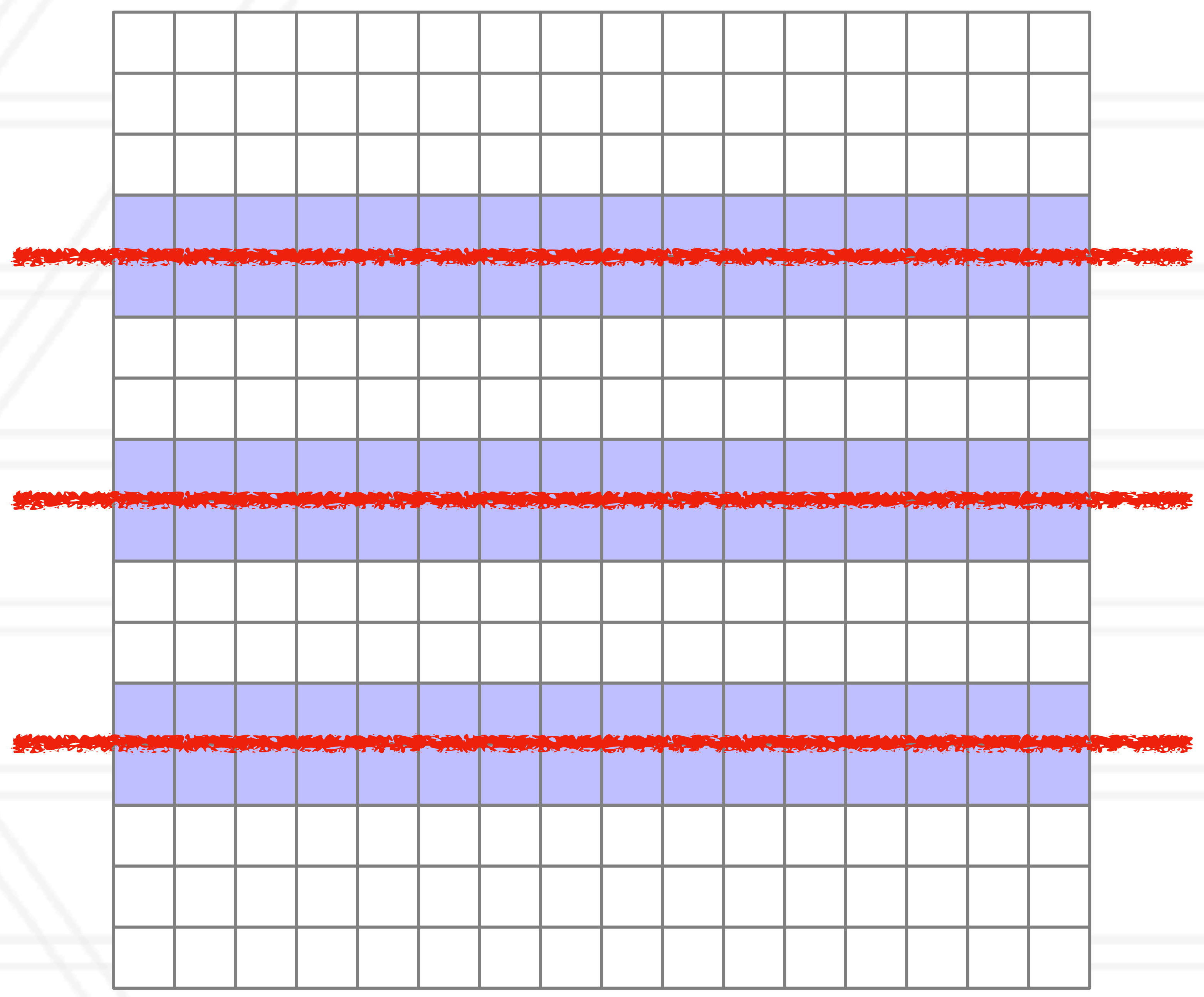
Why use hybrid programming?

- Processes are heavy-weight
- Using MPI everywhere can lead to a large number of messages
- Using threads can enable better sharing of data on symmetric multi-processing (SMP) and multi-core nodes
- Larger grain size (per MPI process) can help with fewer overheads
- Required when you have GPUs attached to a node

What are our choices for X

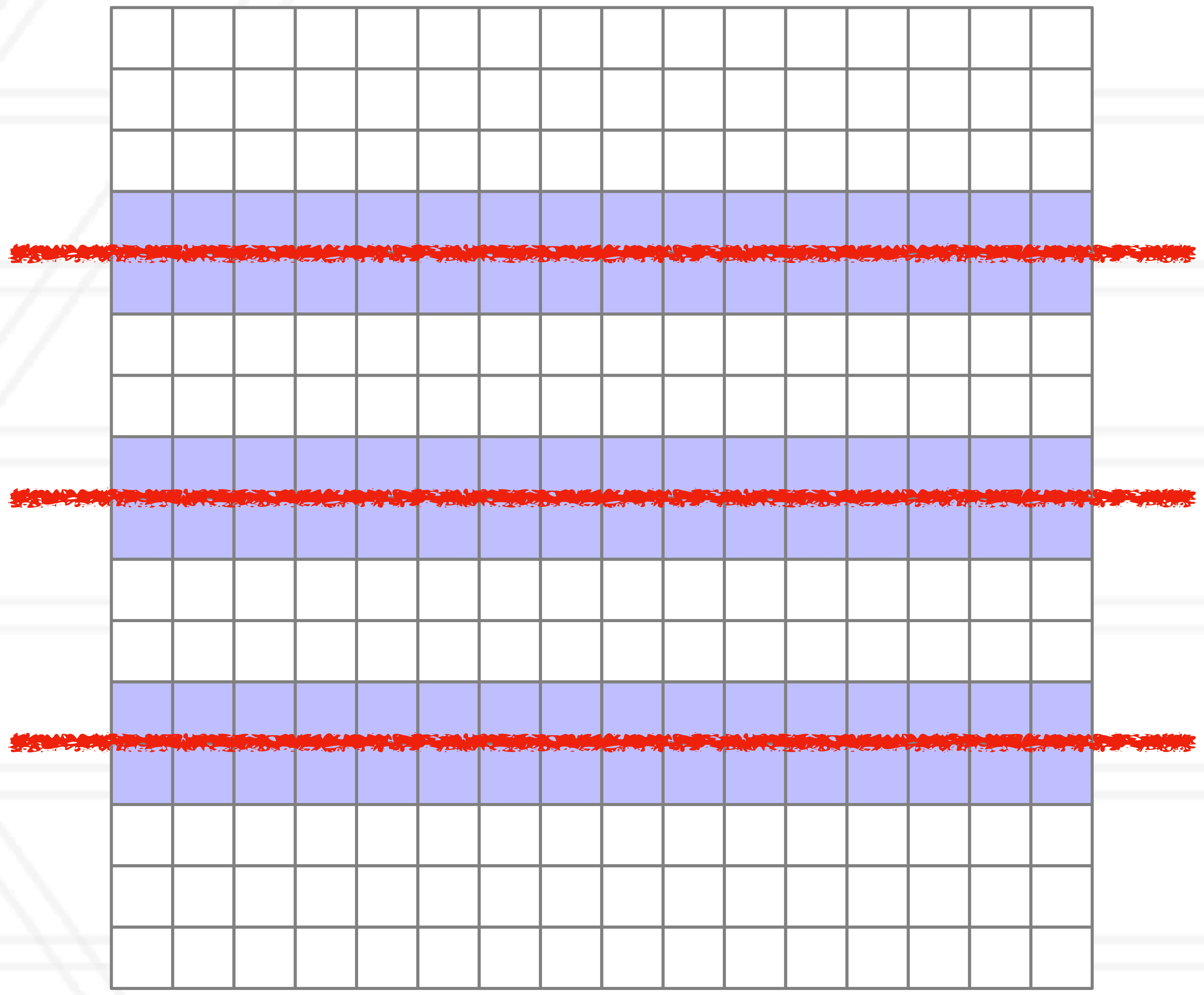
- CPUs: OpenMP, pthreads, RAJA, Kokkos, …
- GPUs: CUDA, HIP, OpenMP offload, RAJA, Kokkos, …
- Notice that some models can be used on both CPUs and GPUs
  - Referred to as "portable" programming models
  - Allow use to right a single code that can run on the CPU or GPU

2D stencil: MPI+OpenMP



2D stencil: MPI+OpenMP

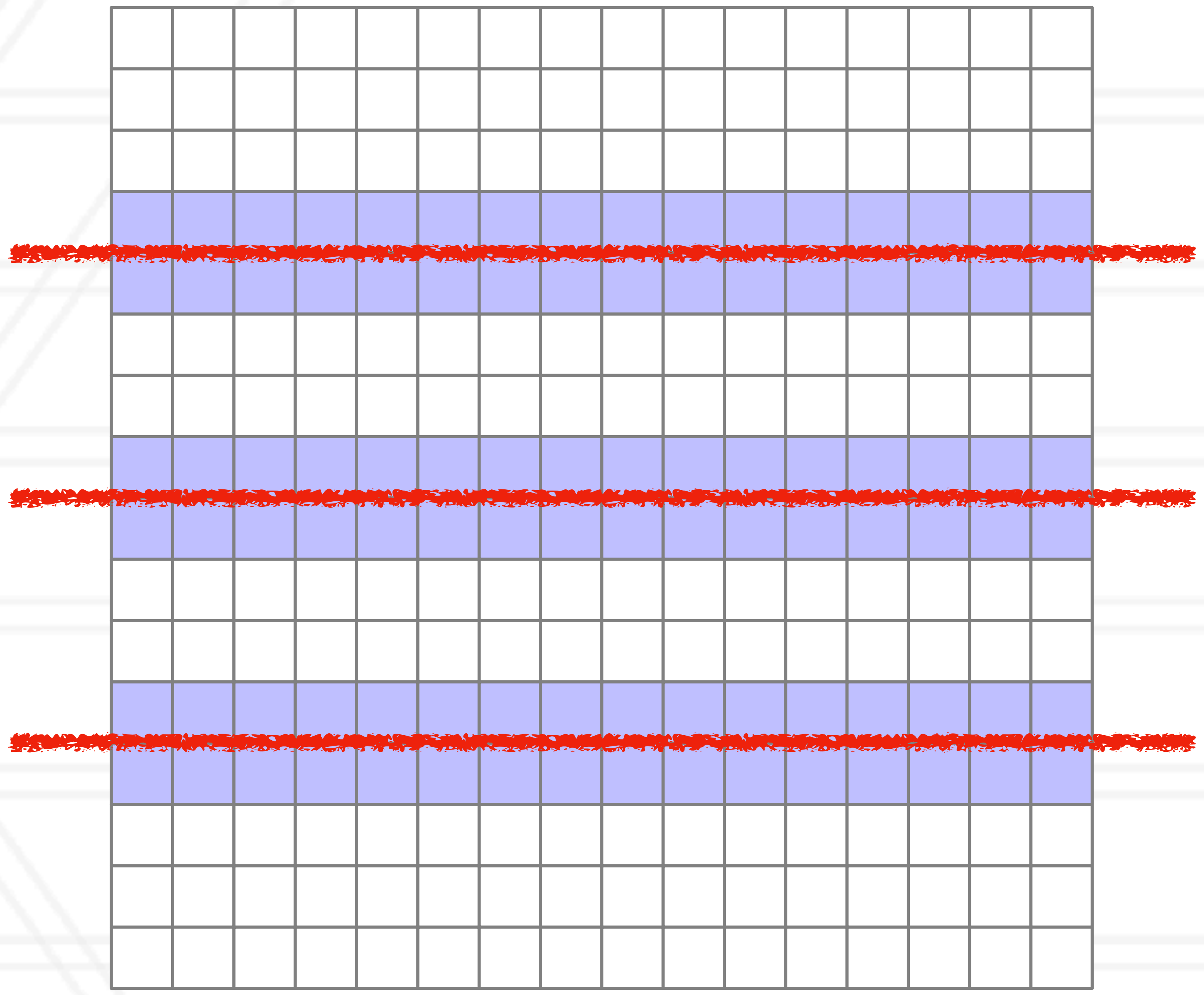
```
int main(int argc, char *argv) {
  ...

  for(int t=0; t<num_steps; t++) {

    MPI_Irecv(&data1, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Irecv(&data2, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);

    MPI_Isend(&data3, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Isend(&data4, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);


    MPI_Waitall(…);

    compute();
  }
  ...
}
```



2D stencil: MPI+OpenMP

```
int main(int argc, char *argv) {
  ...

  for(int t=0; t<num_steps; t++) {

    MPI_Irecv(&data1, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Irecv(&data2, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);

    MPI_Isend(&data3, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Isend(&data4, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);


    MPI_Waitall(…);

    compute();
  }
  ...
}
```

```
# pragma omp parallel for
for(i ...)
  for(j ...)
    A_new[i, j] = (A[i, j] + A[i-1, j] + …
```

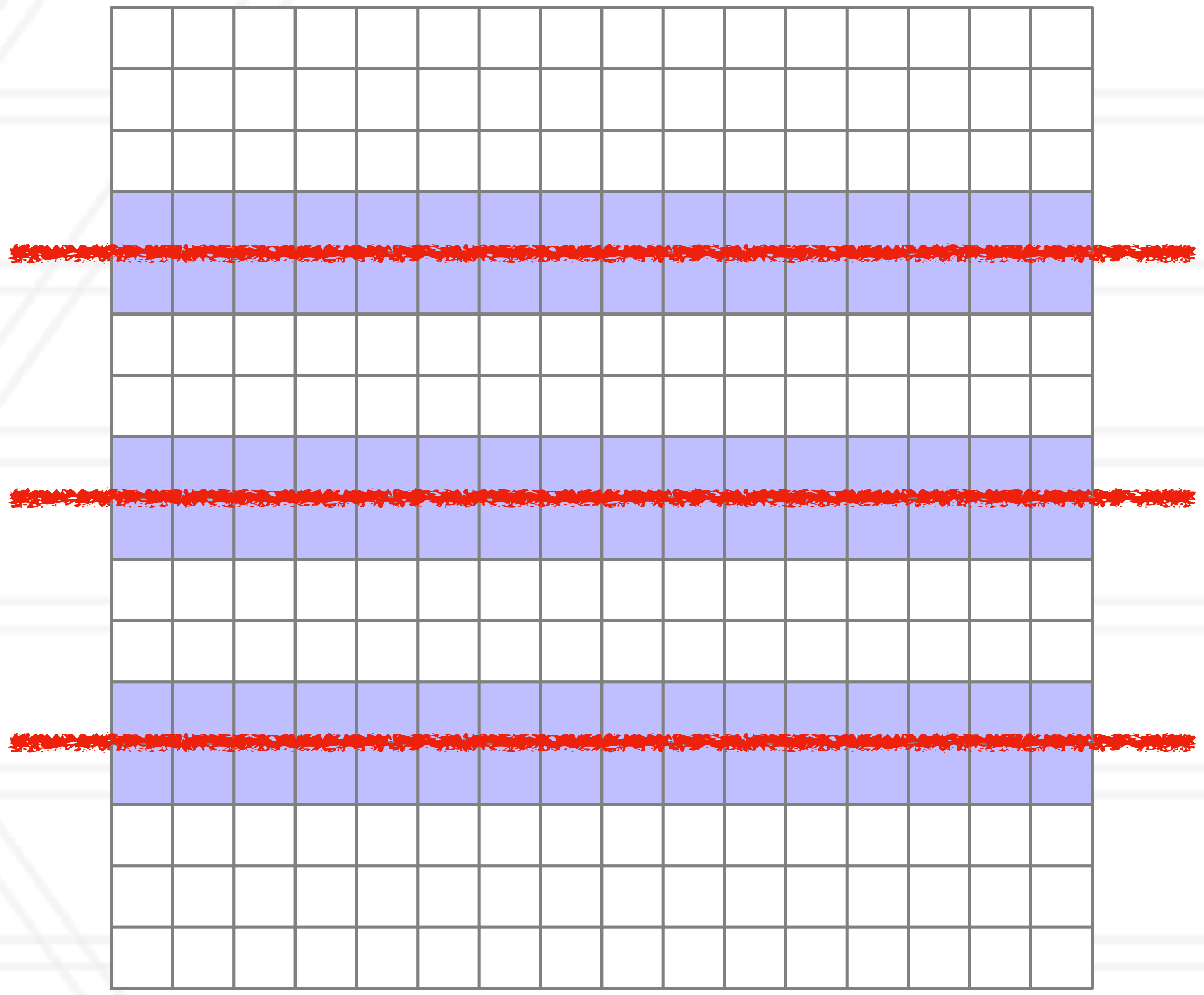


2D stencil: MPI+OpenMP

```
int main(int argc, char *argv) {
  ...

  for(int t=0; t<num_steps; t++) {

    MPI_Irecv(&data1, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Irecv(&data2, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);

    MPI_Isend(&data3, 16, MPI_DOUBLE, (myrank-1)%4, 0, ...);
    MPI_Isend(&data4, 16, MPI_DOUBLE, (myrank+1)%4, 0, ...);


    MPI_Waitall(…);

    compute();
  }
  ...
}
```

Wraparound

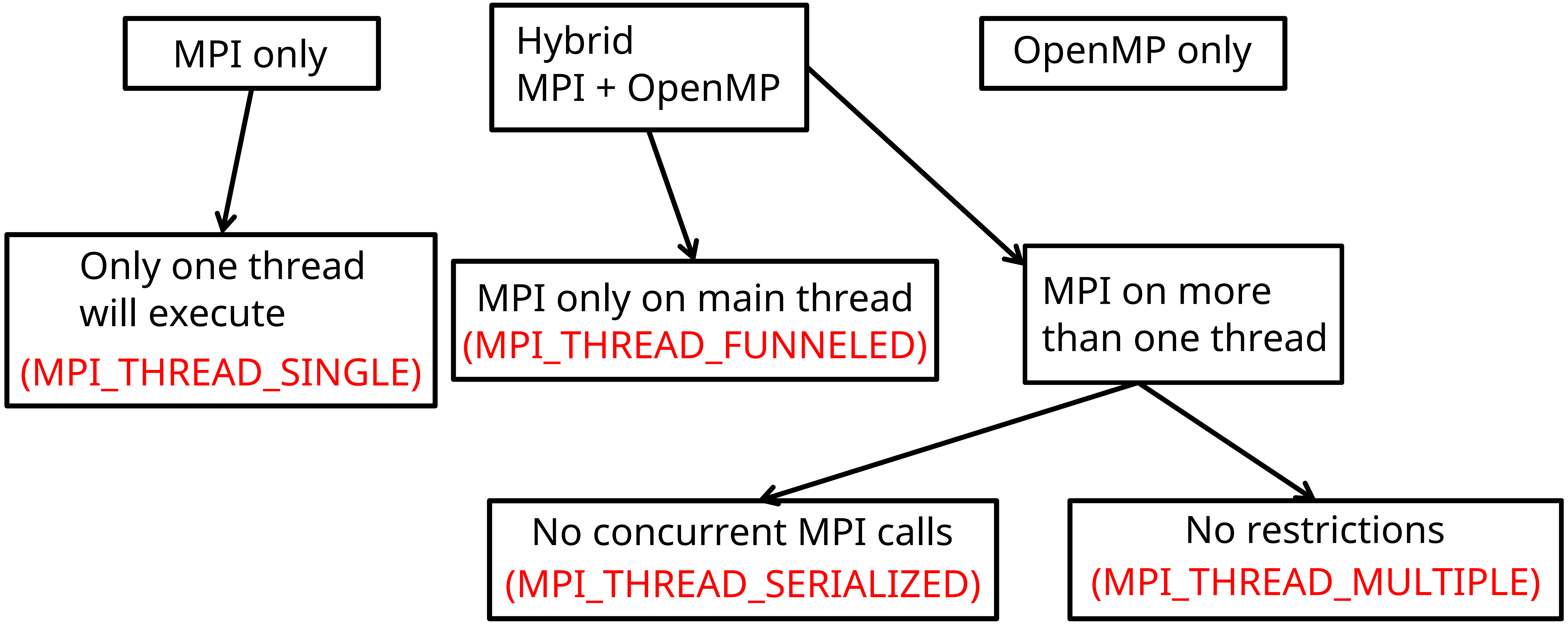
```
# pragma omp parallel for
for(i ...)
  for(j ...)
    A_new[i, j] = (A[i, j] + A[i-1, j] + …
```



Different methods for MPI communication

- MPI_THREAD_SINGLE: all MPI communication is done by the main OpenMP thread outside of OpenMP regions
- MPI_THREAD_FUNNELED: all MPI communication is done by the main OpenMP thread inside OpenMP regions
- MPI_THREAD_SERIALIZED: multiple threads call MPI routines but one thread at a time
- MPI_THREAD_MULTIPLE: multiple threads call MPI routines, potentially simultaneously

Thread support in MPI


MPI only
Hybrid MPI + OpenMP
OpenMP only
Only one thread will execute (MPI_THREAD_SINGLE)
MPI only on main thread (MPI_THREAD_FUNNELED)
MPI on more than one thread
No concurrent MPI calls (MPI_THREAD_SERIALIZED)
No restrictions (MPI_THREAD_MULTIPLE)

https://events.prace-ri.eu/event/1225/attachments/1632/3145/Lecture slides_Hybrid CPU programming with OpenMP and MPI @ CSC (PTC | ONLINE), 4.10-5.10.2021.pdf

Number of threads vs. processes

- It depends!

Process and thread affinity

- Normally, the OS can run processes and threads on any core, and even move them around
- For performance, it's best to pin processes/threads to specific cores
- Use slurm options such as --tasks-per-node and --cpus-per-task to spread tasks apart
- Pinning: --cpu-bind, OMP_PROC_BIND

MPI+CUDA

- Typically let one MPI process manage each GPU

```
MPI_Comm_rank(icomm, &myrank);                  // my MPI rank

int deviceCount;
cudaGetDeviceCount(&deviceCount);               // How many GPUs?

int device_id = myrank % deviceCount;
cudaSetDevice(device_id);                       // Map MPI process to a GPU
```

- Send data to other nodes using the MPI processes on each node

Sending messages to other GPUs/nodes

- Copy data from device to host and then send messages between MPI processes
- GPU-aware MPI: You can provide GPU memory pointers in the MPI_Isend/MPI_Irecv calls
  - Avoids the device to host memcpy in user code
  - The runtime might still do a copy
- MPI built with GPUDirect: When enabled, it avoids an extra copy and directly sends data between GPUs on different nodes

UNIVERSITY OF
MARYLAND