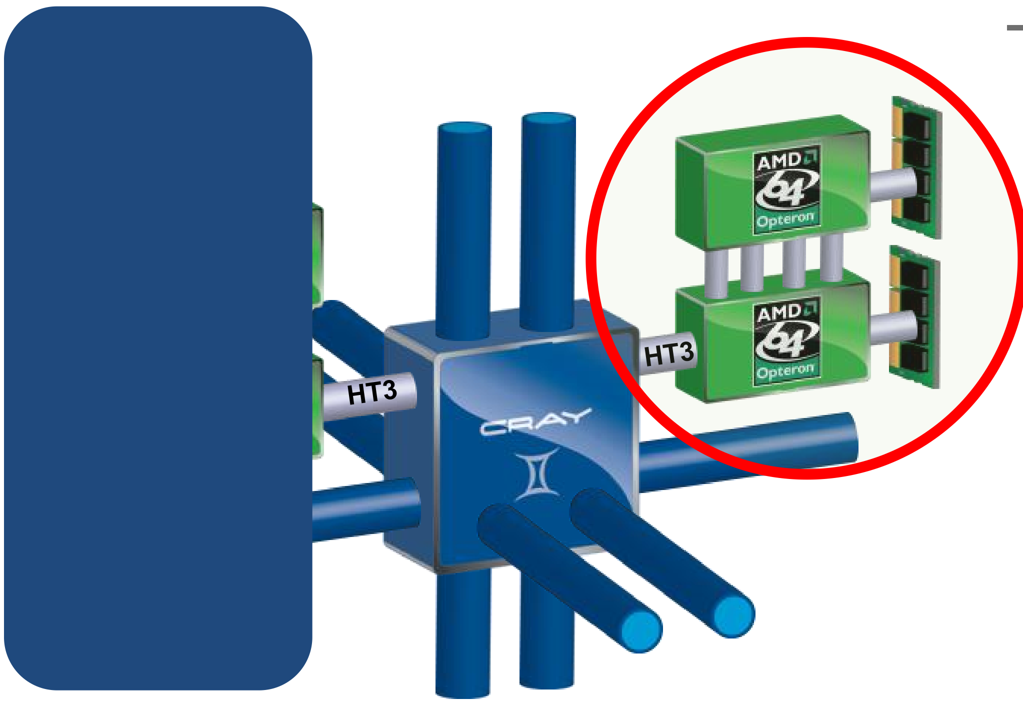
LECTURE 7: JOINT CUDA-MPI PROGRAMMING

Cray XE6 Nodes

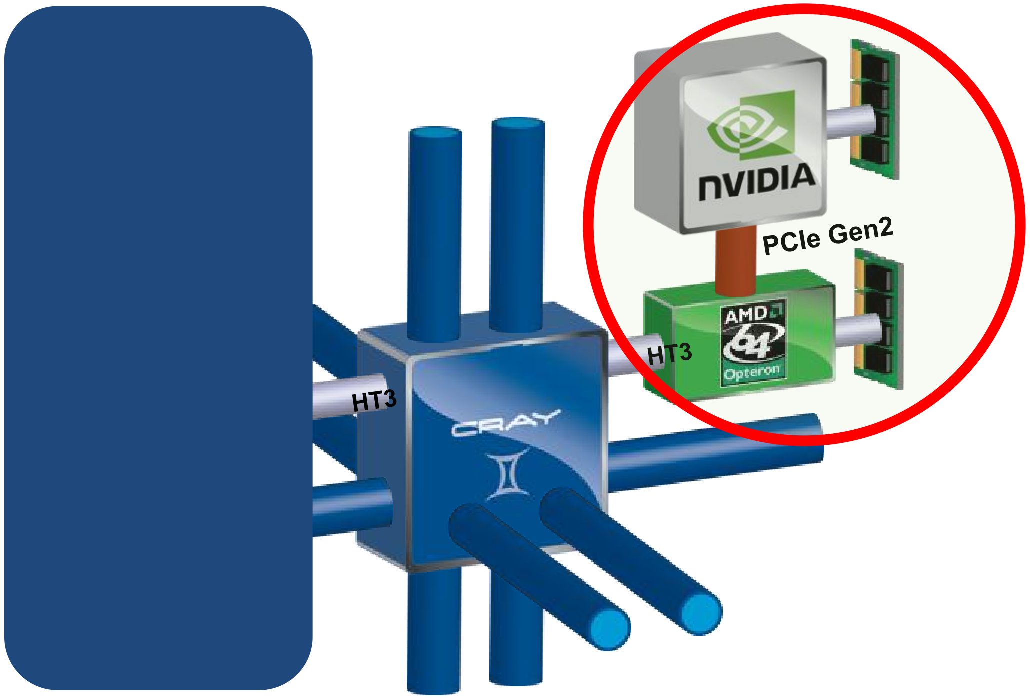


Blue Waters contains 22,640 Cray XE6 compute nodes.

- Dual-socket Node
  - Two AMD Interlagos chips
    - 16 core modules, 64 threads
    - 313 GFs peak performance
    - 64 GBs memory
      - 102 GB/sec memory bandwidth
  - Gemini Interconnect
    - Router chip & network interface
    - Injection Bandwidth (peak)
      - 9.6 GB/sec per direction

University of Electronic Science and Technology of China

Cray XK7 Nodes

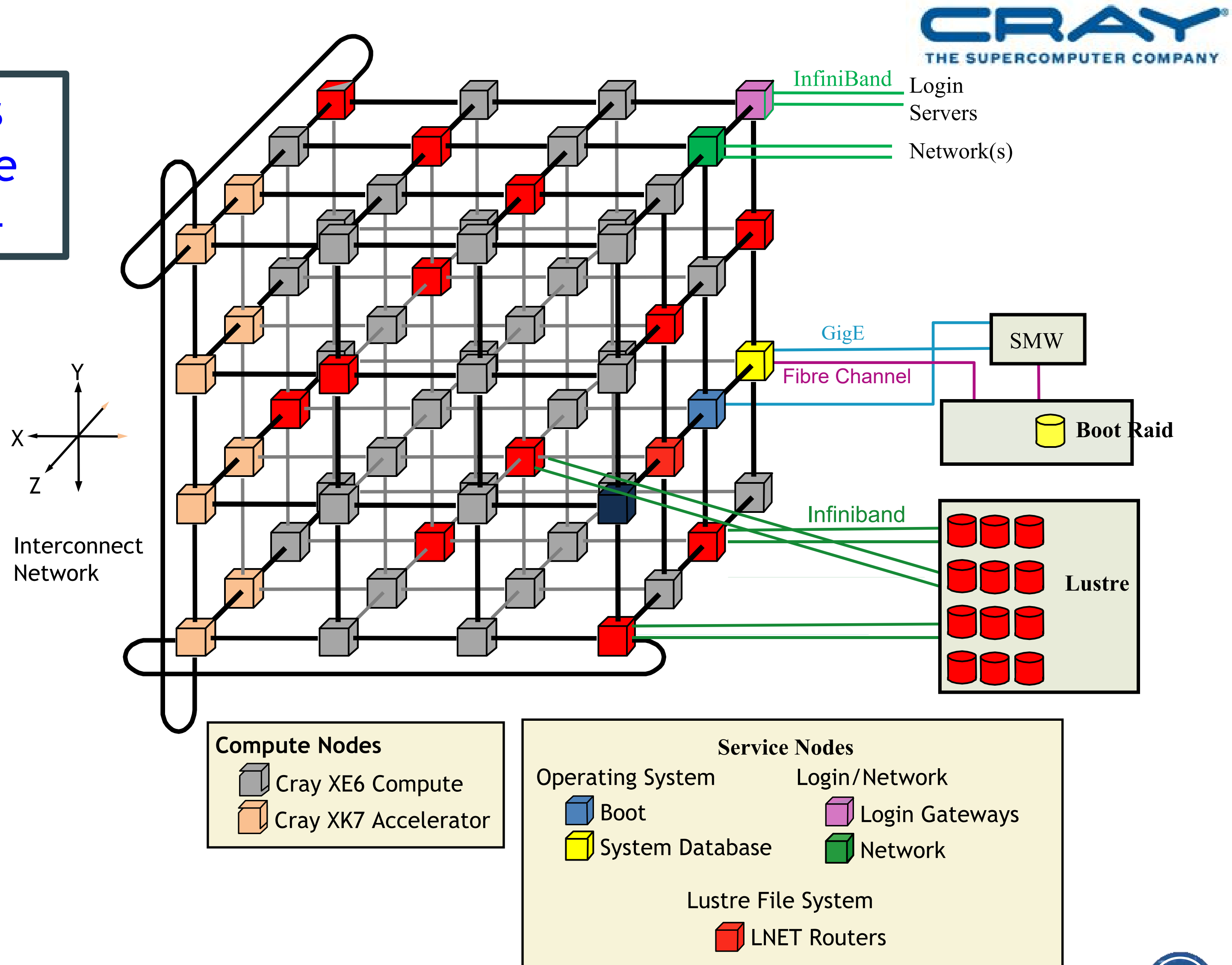


Blue Waters contains 3,072 Cray XK7 compute nodes.

- Dual-socket Node
  - One AMD Interlagos chip
    - 8 core modules, 32 threads
    - 156.5 GFs peak performance
    - 32 GBs memory
      - 51 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.3 TFs peak performance
    - 6 GBs GDDR5 memory
      - 250 GB/sec bandwidth
  - Gemini Interconnect
    - Same as XE6 nodes

Gemini Interconnect Network

Blue Waters
3D Torus Size
23 x 24 x 24



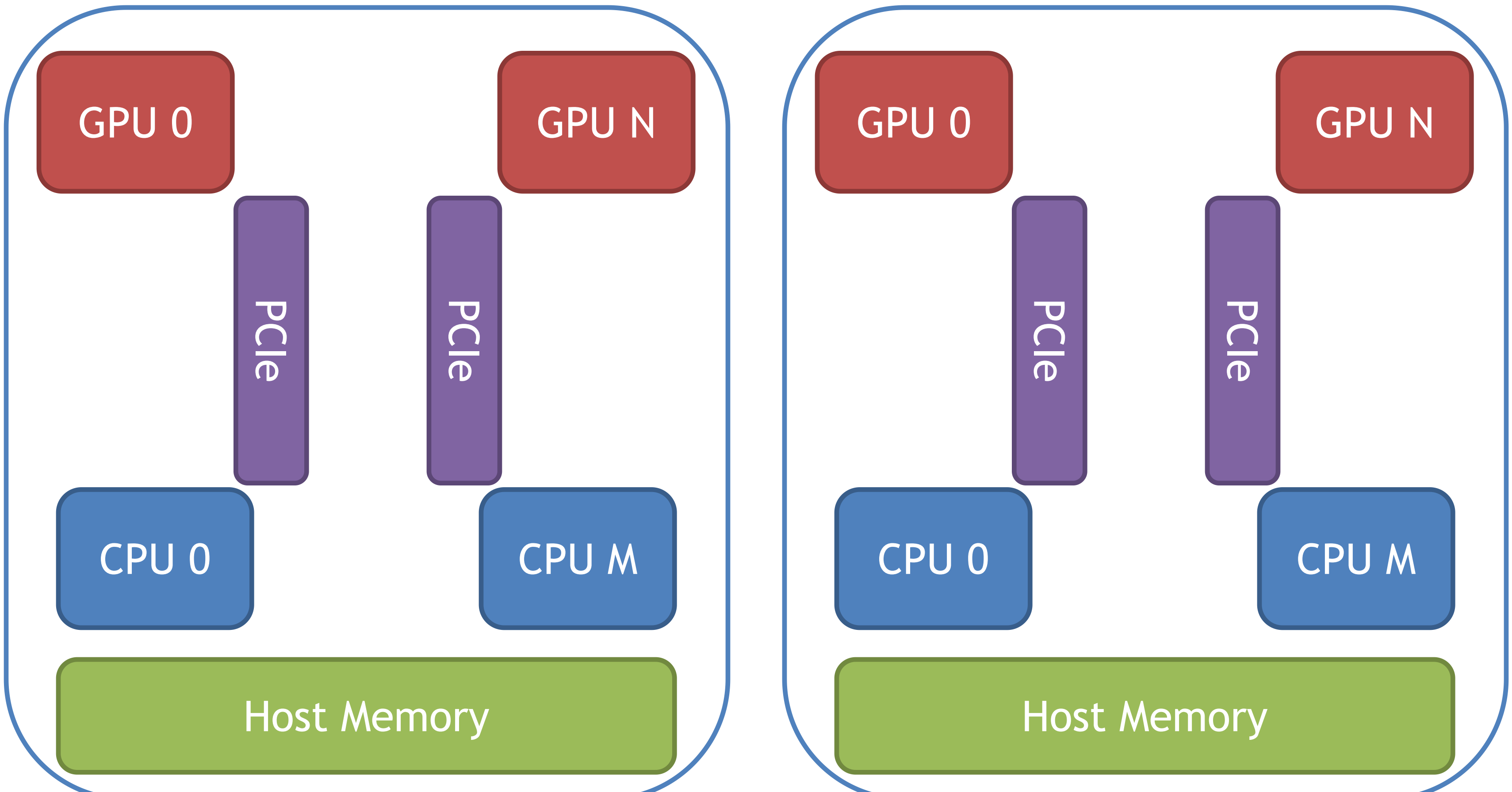
University of Electronic Science and Technology of China

Blue Waters and Titan Computing Systems

System Attribute	NCSA Blue Waters	ORNL Titan
Vendors	Cray/AMD/NVIDIA	Cray/AMD/NVIDIA
Processors	Interlagos/Kepler	Interlagos/Kepler
Total Peak Performance (PF)	11.1	27.1
Total Peak Performance (CPU/GPU)	7.1/4	2.6/24.5
Number of CPU Chips	48,352	18,688
Number of GPU Chips	3,072	18,688
Amount of CPU Memory (TB)	1511	584
Interconnect	3D Torus	3D Torus
Amount of On-line Disk Storage (PB)	26	13.6
Sustained Disk Transfer (TB/sec)	>1	0.4-0.7
Amount of Archival Storage	300	15-30
Sustained Tape Transfer (GB/sec)	100	7

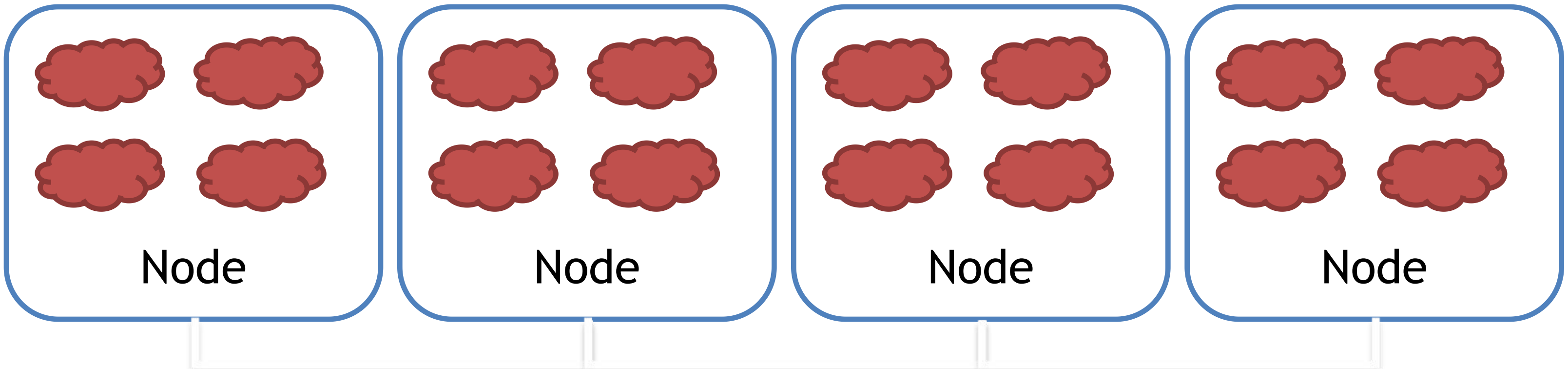
CUDA-based cluster

– Each node contains *N* GPUs



MPI Model

- Many processes distributed in a cluster



- Each process computes part of the output
- Processes communicate with each other
- Processes can synchronize

MPI Initialization, Info and Sync

- `int MPI_Init(int *argc, char ***argv)`
  - Initialize MPI
- `MPI_COMM_WORLD`
  - MPI group with all allocated nodes
- `int MPI_Comm_rank (MPI_Comm comm, int *rank)`
  - Rank of the calling process in group of comm
- `int MPI_Comm_size (MPI_Comm comm, int *size)`
  - Number of processes in the group of comm

Vector Addition: Main Process

```
int main(int argc, char *argv[]) {
    int vector_size = 1024 * 1024 * 1024;
    int pid=-1, np=-1;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    if(np < 3) {
        if(0 == pid) printf("Nedded 3 or more processes.\n");
        MPI_Abort( MPI_COMM_WORLD, 1 ); return 1;
    }
    if(pid < np - 1)
        compute_node(vector_size / (np - 1));
    else
        data_server(vector_size);

    MPI_Finalize();
    return 0;
}
```

Vector Addition: Server Process (I)

```
void data_server(unsigned int vector_size) {
    int np, num_nodes = np – 1, first_node = 0, last_node = np - 2;
    unsigned int num_bytes  = vector_size * sizeof(float);
    float *input_a = 0, *input_b = 0, *output = 0;

    /* Set MPI Communication Size */
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    /* Allocate input data */
    input_a = (float *)malloc(num_bytes);
    input_b = (float *)malloc(num_bytes);
    output = (float *)malloc(num_bytes);
    if(input_a == NULL || input_b == NULL || output == NULL) {
        printf(“Server couldn't allocate memory\n");
        MPI_Abort( MPI_COMM_WORLD, 1 );
    }
    /* Initialize input data */
    random_data(input_a, vector_size , 1, 10);
    random_data(input_b, vector_size , 1, 10);
```

Vector Addition: Server Process (II)

```
/* Send data to compute nodes */
float *ptr_a = input_a;
float *ptr_b = input_b;

for(int process = 1; process < last_node; process++) {
    MPI_Send(ptr_a, vector_size / num_nodes, MPI_FLOAT,
             process, DATA_DISTRIBUTE, MPI_COMM_WORLD);
    ptr_a += vector_size / num_nodes;

    MPI_Send(ptr_b, vector_size / num_nodes, MPI_FLOAT,
             process, DATA_DISTRIBUTE, MPI_COMM_WORLD);
    ptr_b += vector_size / num_nodes;
}

/* Wait for nodes to compute */
MPI_Barrier(MPI_COMM_WORLD);
```

Vector Addition: Server Process (III)

```
    /* Wait for previous communications */   MPI_Barrier(MPI_COMM_WORLD);

    /* Collect output data */
    MPI_Status status;
    for(int process = 0; process < num_nodes; process++) {
        MPI_Recv(output + process * num_points / num_nodes,
            num_points / num_comp_nodes, MPI_REAL, process,
            DATA_COLLECT, MPI_COMM_WORLD, &status );
    }

    /* Store output data */
    store_output(output, dimx, dimy, dimz);

    /* Release resources */
    free(input);
    free(output);
}
```

Vector Addition: Compute Process (I)

```
void compute_node(unsigned int vector_size ) {
    int np;

    MPI_Comm_size(MPI_COMM_WORLD, &np);

    unsigned int num_bytes = vector_size * sizeof(float);
    float *input_a, *input_b, *output;
    MPI_Status status;

    int server_process = np - 1;

    /* Alloc host memory */
    input_a = (float *)malloc(num_bytes);
    input_b = (float *)malloc(num_bytes);
    output = (float *)malloc(num_bytes);

    /* Get the input data from server process */
    MPI_Recv(input_a, vector_size, MPI_FLOAT, server_process,
            DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
    MPI_Recv(input_b, vector_size, MPI_FLOAT, server_process,
            DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
```

Vector Addition: Compute Process (II)

```
    /* Compute the partial vector addition */
    for(int i = 0; i < vector_size; ++i) {
        output[i] = input_a[i] + input_b[i];
    }

    /* Send the output */
    MPI_Send(output, vector_size, MPI_FLOAT,
             server_process, DATA_COLLECT, MPI_COMM_WORLD);

    /* Release memory */
    free(input_a);
    free(input_b);
    free(output);
}
```

MPI Barriers

- `int MPI_Barrier (MPI_Comm comm)`
  - Comm: Communicator (handle)

- Blocks the caller until all group members have called it; the call returns at any process only after all group members have entered the call.

MPI Barriers

– Wait until all other processes in the MPI group reach the same barrier

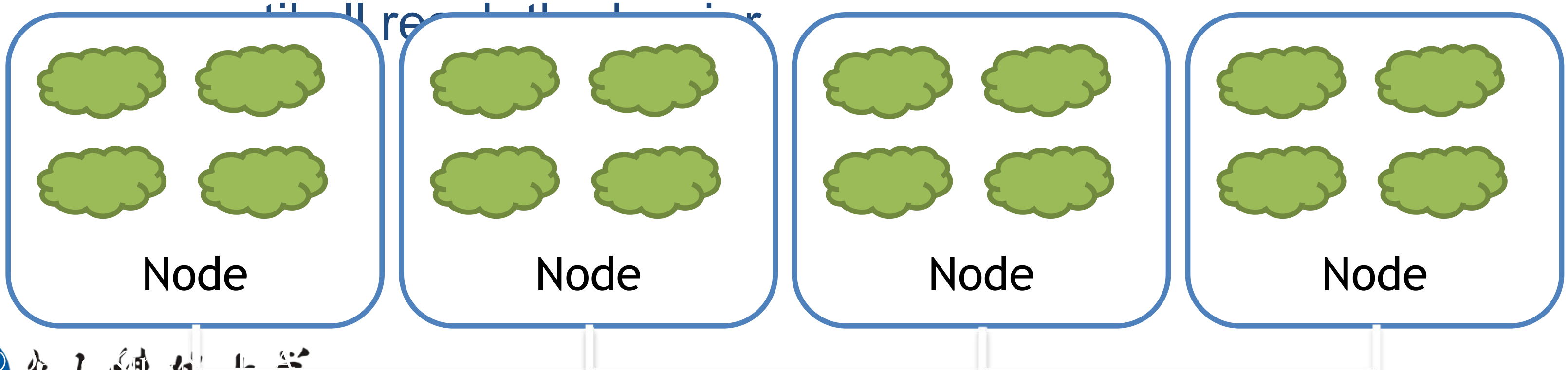
1. All processes are executing Do_Stuff()
2. Some processes reach the barrier
   and the wait in the barrier

Example Code

```
Do_stuff();

MPI_Barrier();

Do_more_stuff();
```



Vector Addition: Compute Process (II)

```
    /* Compute the partial vector addition */
    for(int i = 0; i < vector_size; ++i) {
        output[i] = input_a[i] + input_b[i];
    }

    /* Report to barrier after computation is done*/
    MPI_Barrier(MPI_COMM_WORLD);


    /* Send the output */
    MPI_Send(output, vector_size, MPI_FLOAT,
            server_process, DATA_COLLECT, MPI_COMM_WORLD);

    /* Release memory */
    free(input_a);
    free(input_b);
    free(output);
}
```

ADDING CUDA TO MPI

Vector Addition: CUDA Process (I)

```
void compute_node(unsigned int vector_size ) {
    int np;
    unsigned int num_bytes = vector_size * sizeof(float);
    float *input_a, *input_b, *output;
    MPI_Status status;

    MPI_Comm_size(MPI_COMM_WORLD, &np);
    int server_process = np - 1;

    /* Allocate memory */
    cudaHostAlloc((void **)&h_a, num_bytes, cudaHostAllocDefault);
    cudaHostAlloc((void **)&h_b, num_bytes, cudaHostAllocDefault);
    cudaHostAlloc((void **)&h_output, num_bytes, cudaHostAllocDefault);

    /* Get the input data from server process */
    MPI_Recv(h_a, vector_size, MPI_FLOAT, server_process,
     DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
    MPI_Recv(h_b, vector_size, MPI_FLOAT, server_process,
     DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
```

Vector Addition: CUDA Process (II)

```
/* Transfer data to CUDA device */
cudaMalloc((void **) &d_A, size);
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMalloc((void **) &d_B, size);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
cudaMalloc((void **) &d_output, size);

 /* Compute the partial vector addition */
dim3 Db(BLOCK_SIZE);
dim3 Dg((vector_size + BLOCK_SIZE – 1)/BLOCK_SIZE);
vector_add_kernel<<<Dg, Db>>>(d_output, d_a, d_b, vector_size);
cudaMemcpy(h_output, d_output, size, cudaMemcpyDeviceToHost);

/* Send the output */
MPI_Barrier(d_output);

/* Send the output */
MPI_Send(output, vector_size, MPI_FLOAT, server_process,
   DATA_COLLECT, MPI_COMM_WORLD);
```

Vector Addition: CUDA Process (III)

```
/* Release device memory */
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_output);
cudaFreeHost(h_a);
cudaFreeHost(h_b);
```

QUESTIONS?