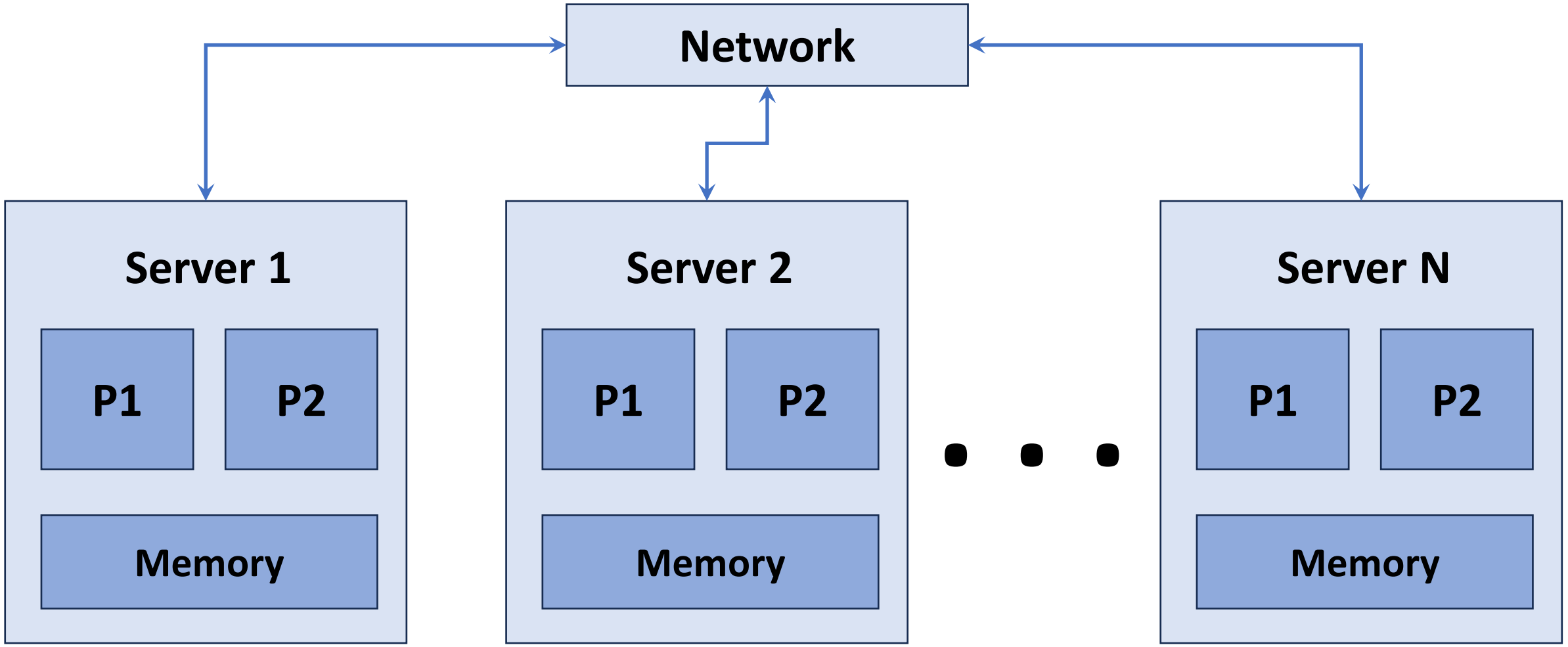
Introduction to HPC

Mandar Gurav

Outline

- Introduction to HPC
- Distributed memory programming using MPI
- Shared memory programming using OpenMP
- Shared memory programming using GP-GPU (CUDA)

General HPC Architecture



Pratyush Supercomputer at IITM, Pune

3315 nodes X 2 CPUs X 18 Cores = 119,340 cores

Reference: http://pratyush.tropmet.res.in/

Pratyush Supercomputer at IITM, Pune

3315 nodes X 2 CPUs X 18 Cores = 119,340 cores

119,340 X 2.1 Ghz = 250614 Gflops => ~250 Tflops

4 Pflops / 250 Tflops = 16?

Reference: http://pratyush.tropmet.res.in/

Pratyush Supercomputer at IITM, Pune

3315 nodes X 2 CPUs X 18 Cores = 119,340 cores

119,340 X 2.1 Ghz = 250614 Gflops => ~250 Tflops

4 Pflops / 250 Tflops = 16?

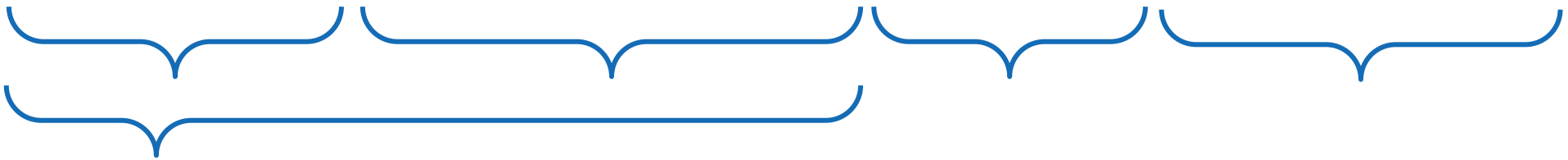
Each core has two 256 bit AVX units which can perform 16 flops per cycle

(256 bit / 64 bit) X (Two such units) X (Fused Multiply Add) = 16

Reference: http://pratyush.tropmet.res.in/

Pratyush Supercomputer at IITM, Pune (contd)

3315 nodes X 2 CPUs X 18 Cores X 2.1 Ghz X 16 Flops/Cycle



MPI OpenMP Serial Vectorization

So to fully utilize the system, you will need

- MPI + Vectorization
- MPI + OpenMP + Vectorization

Reference: http://pratyush.tropmet.res.in/

Pratyush Supercomputer at IITM, Pune (contd)

3315 nodes X 2 CPUs X 18 Cores X 2.1 Ghz X 16 Flops/Cycle

Distributed Memory Programming

Shared Memory Programming

Serial

SIMD / Vectorization

So to fully utilize the system, you will need

- MPI + Vectorization
- MPI + OpenMP + Vectorization

Reference: http://pratyush.tropmet.res.in/

Factors Affecting performance

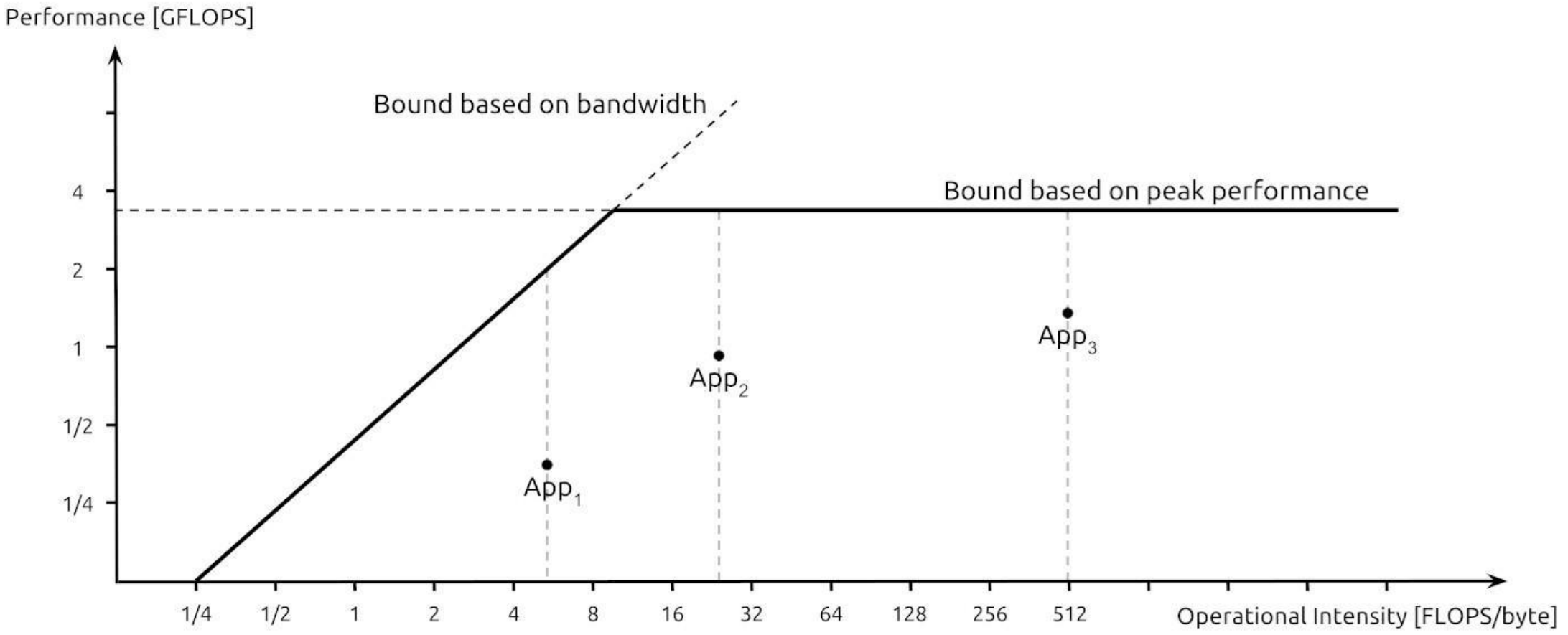
- Clock speed
- Core count
- SIMD width
- Memory bandwidth
- Network bandwidth
- Size of cache
- Instruction Level Parallelism
- Programming skills -
  - MPI - blocking, non-blocking, collective, one-sided
  - Shared Memory - false-sharing, NUMA

Speedup

$$\text{Speedup} = \frac{\text{Execution time for the Parallel Code}}{\text{Execution time for the Serial Code}}$$

- Speedup is a relative term. One can always achieve higher speedup by comparing parallel execution time with serial on slower reference hardware.
- While comparing speedups with other reference(s), Apples to Apples comparison can happen only when we satisfy all/most the following criterias
  - Hardware is same/similar
  - OS and compiler versions match.
  - Same compiler flags are used.
  - Same method/technique is used for measurements.
  - ….

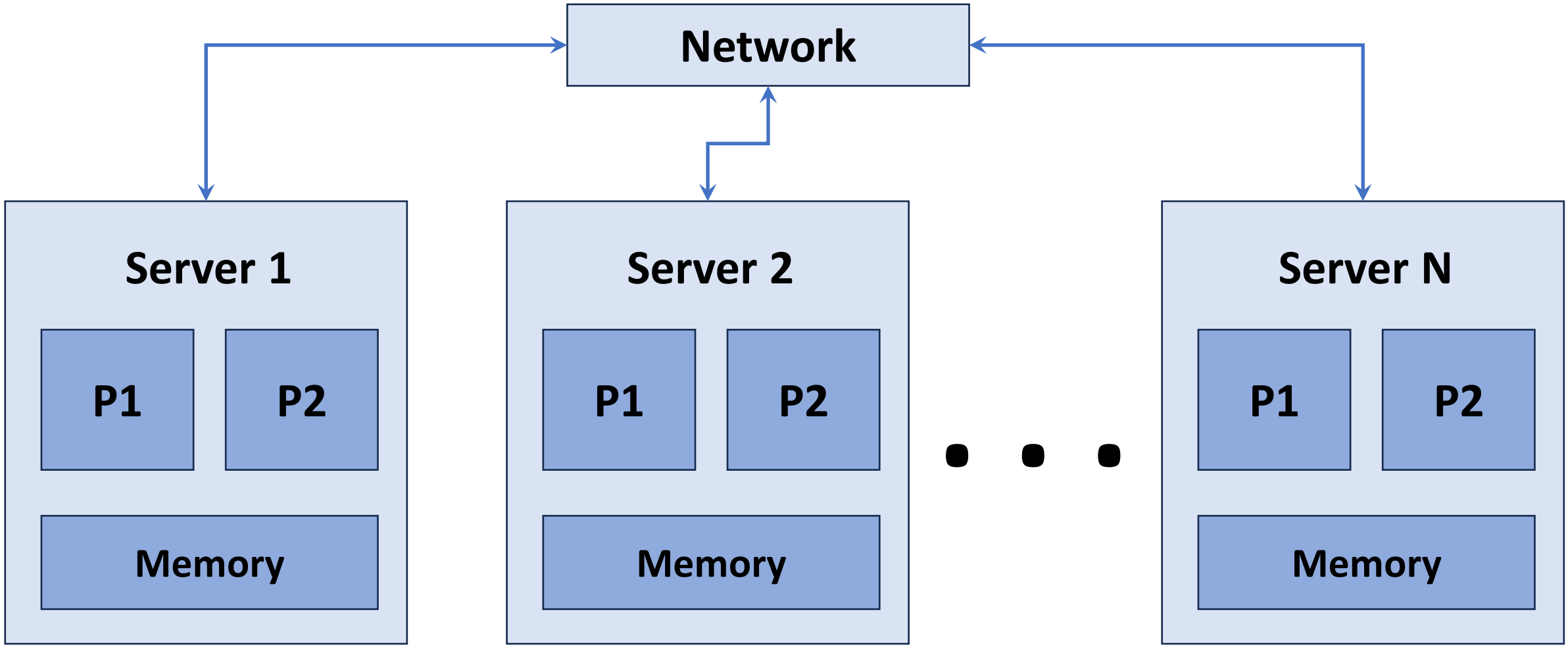
Roofline modeling



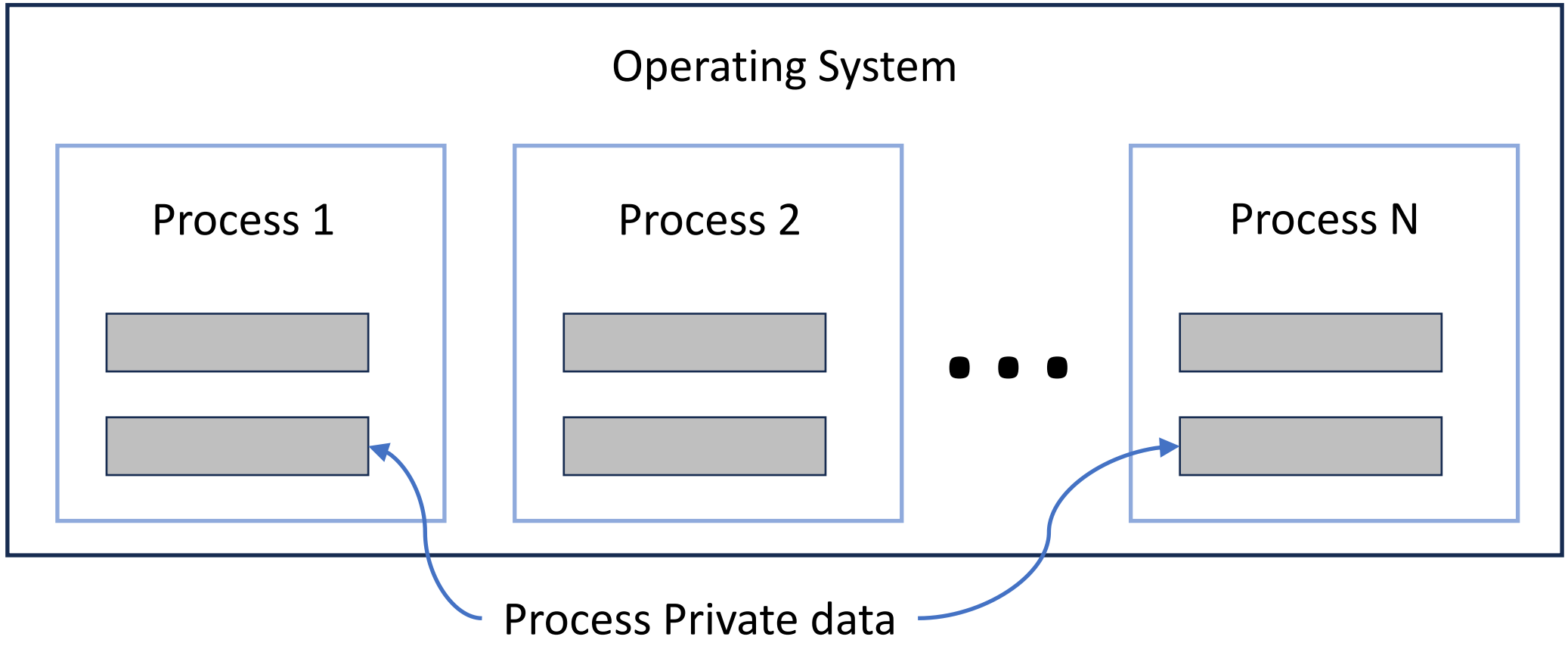
Ref: https://en.wikipedia.org/wiki/Roofline_model

Distributed Memory Programming

Why Distributed Memory Programming?



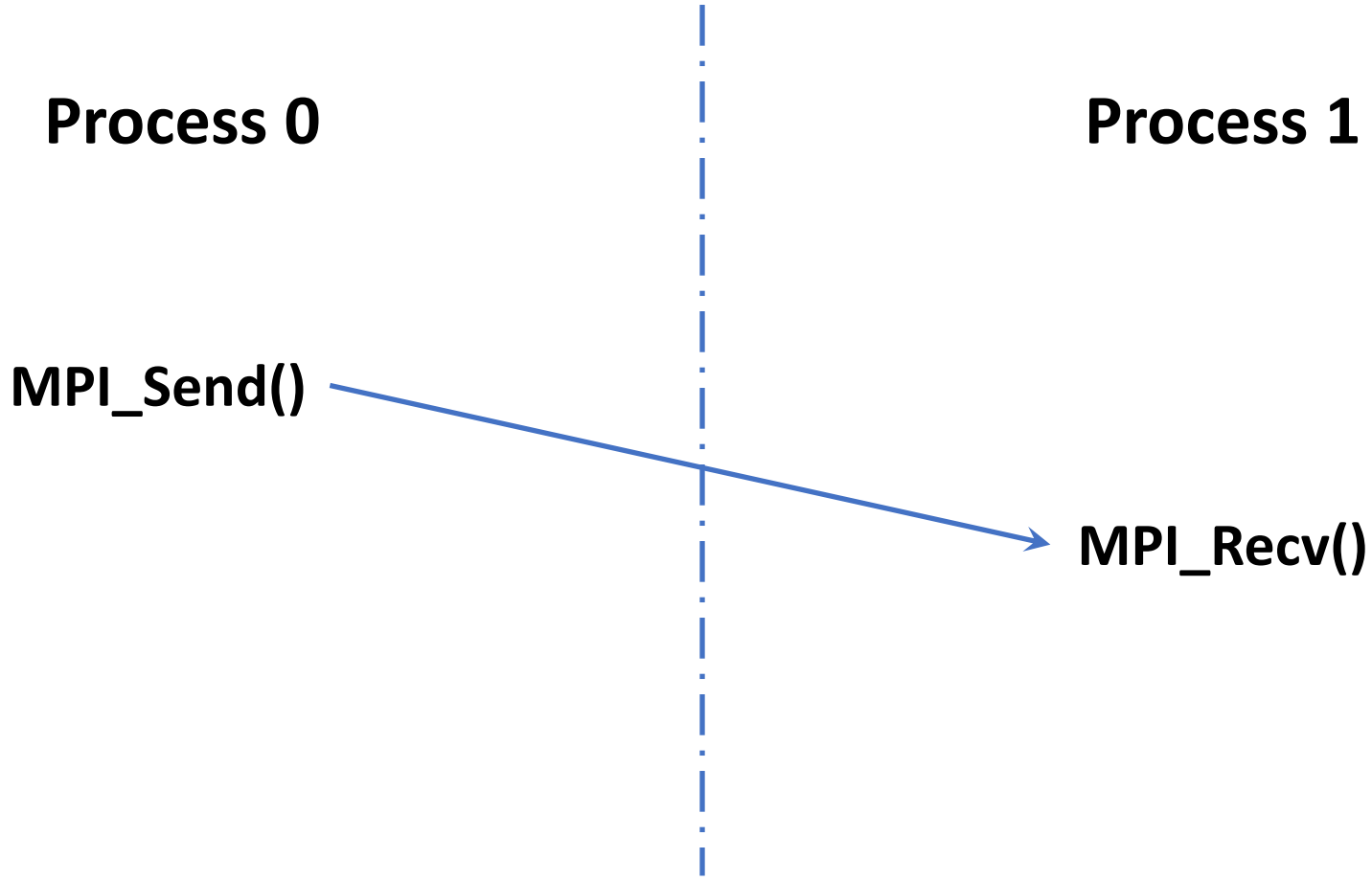
Why Distributed Memory Programming?



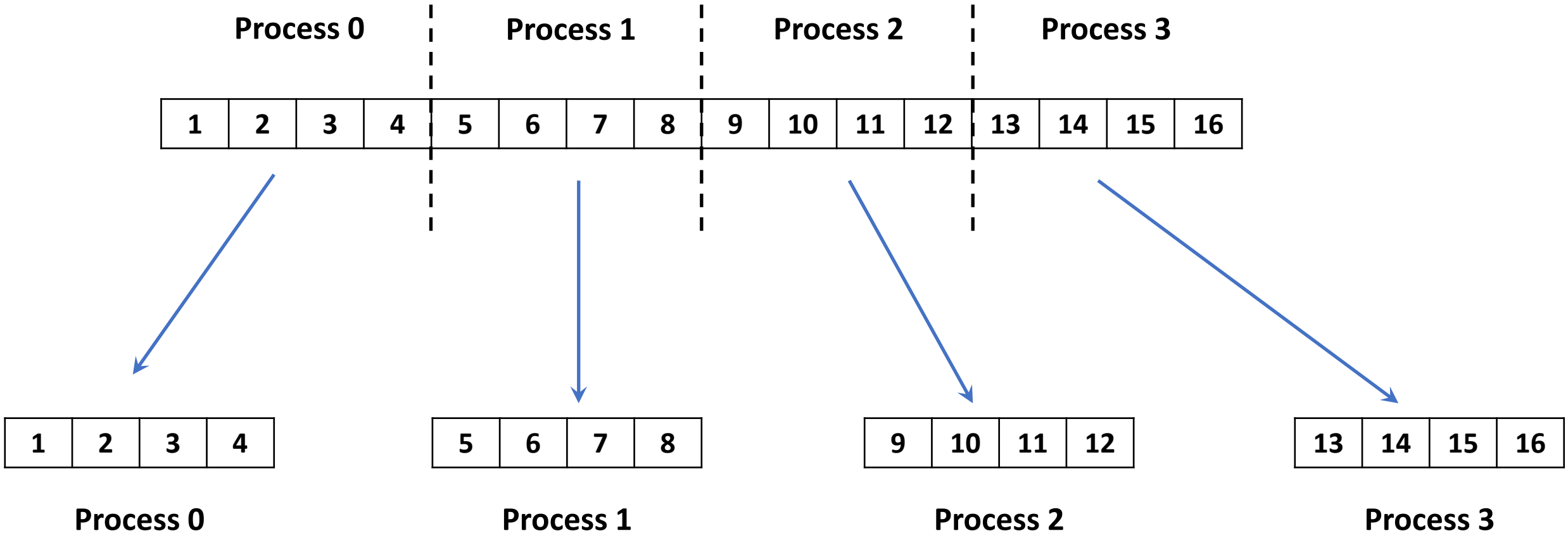
Introduction to Message Passing Interface (MPI)

- Stable and matured message-passing standard
- Defines the syntax and semantics of library routines
- Allow us to exchange/send-receive/share the data over distributed memory nodes
- Supports C, C++ And Fortran. Other languages are also enabled (e.g. Python etc)
- Several open-source MPI implementations available along with the vendor specific optimized versions.

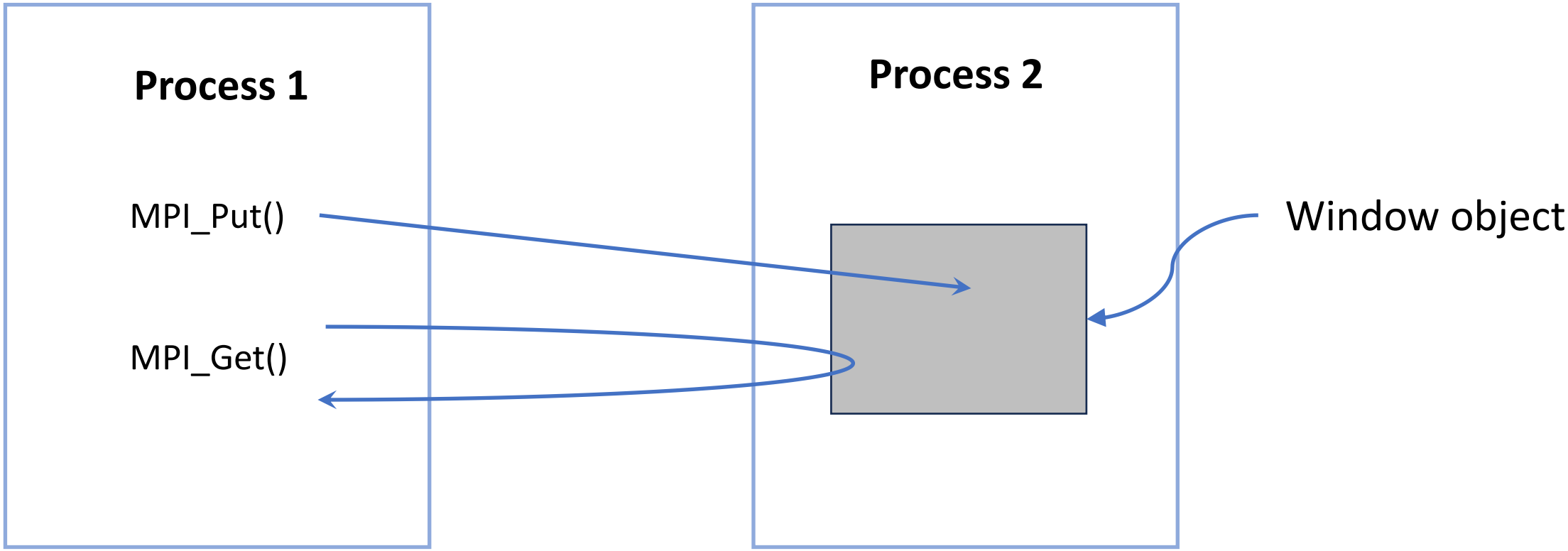
Point-to-Point Communication



Collective Communication (MPI_Scatter)

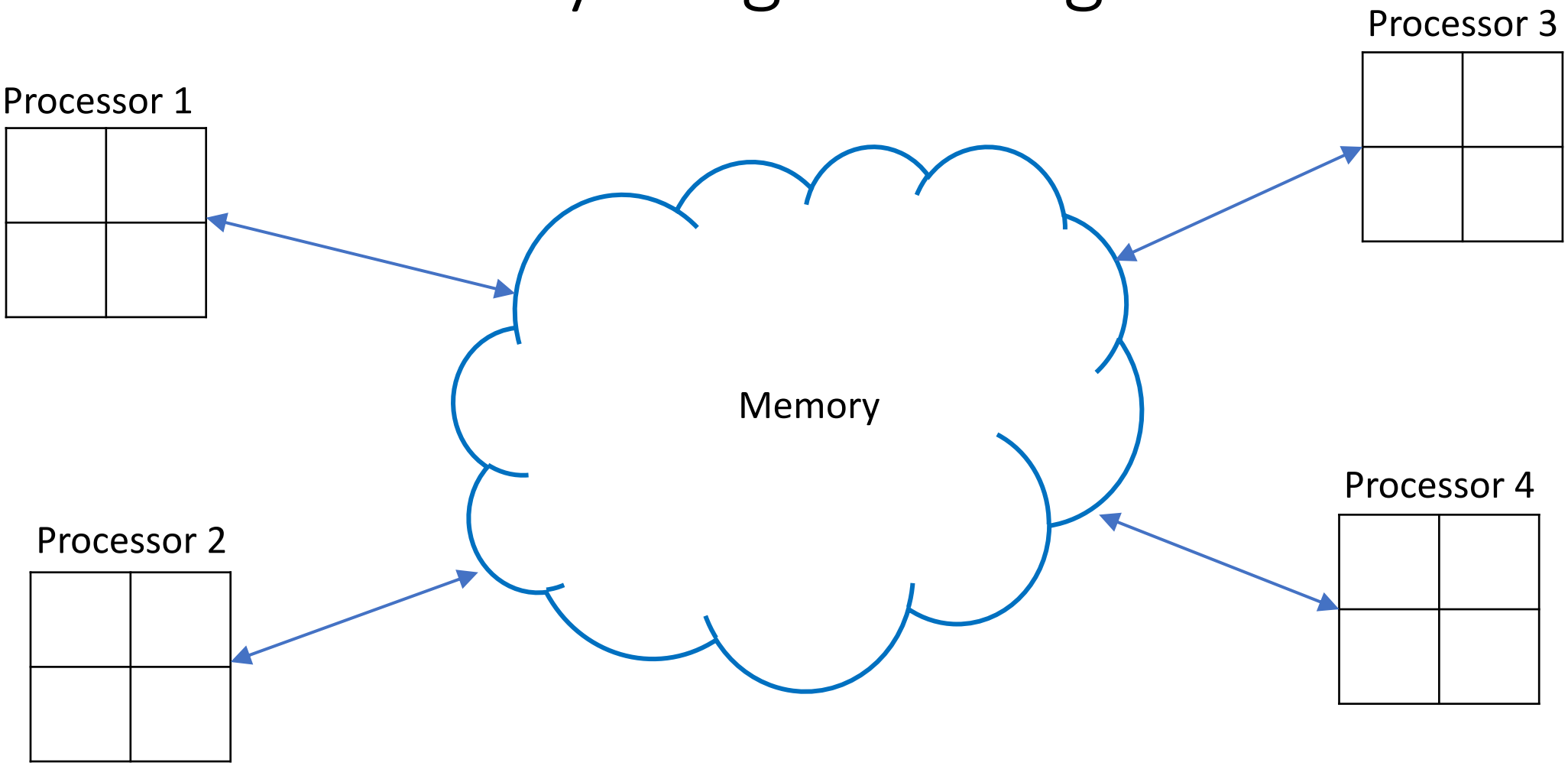


One Sided communication

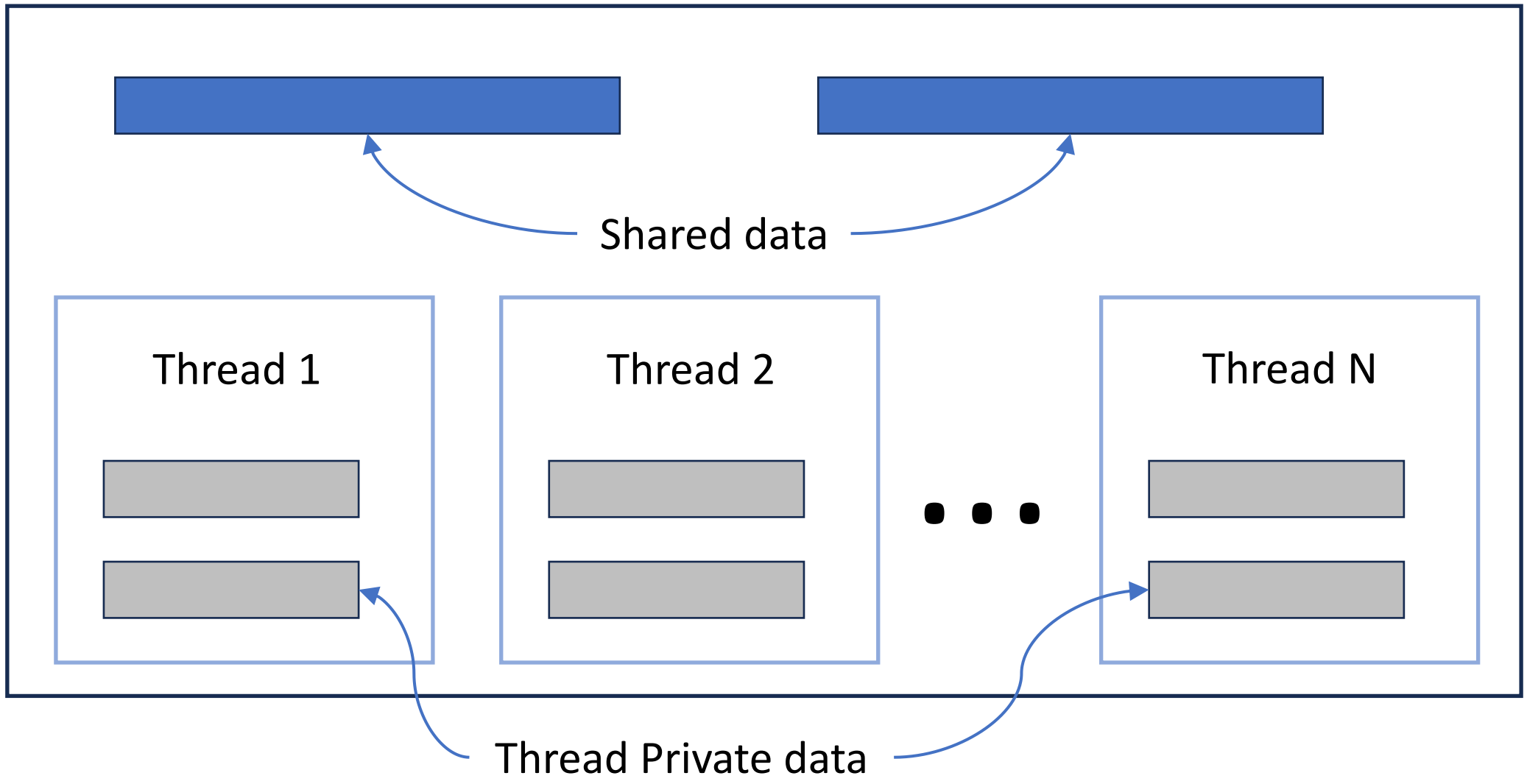


Shared Memory Programming

Shared Memory Programming



Shared Memory Programming

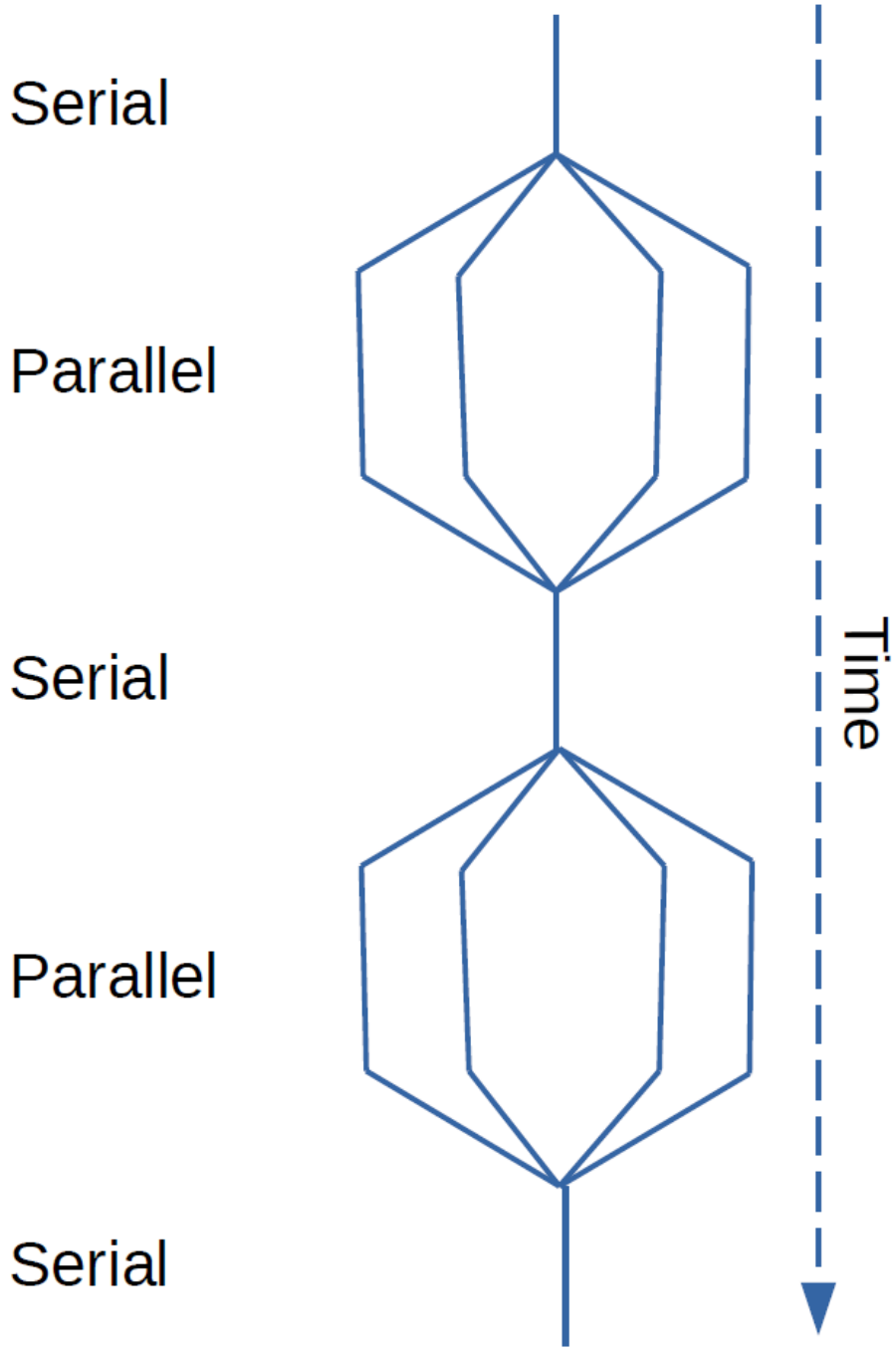


Shared memory programming using OpenMP

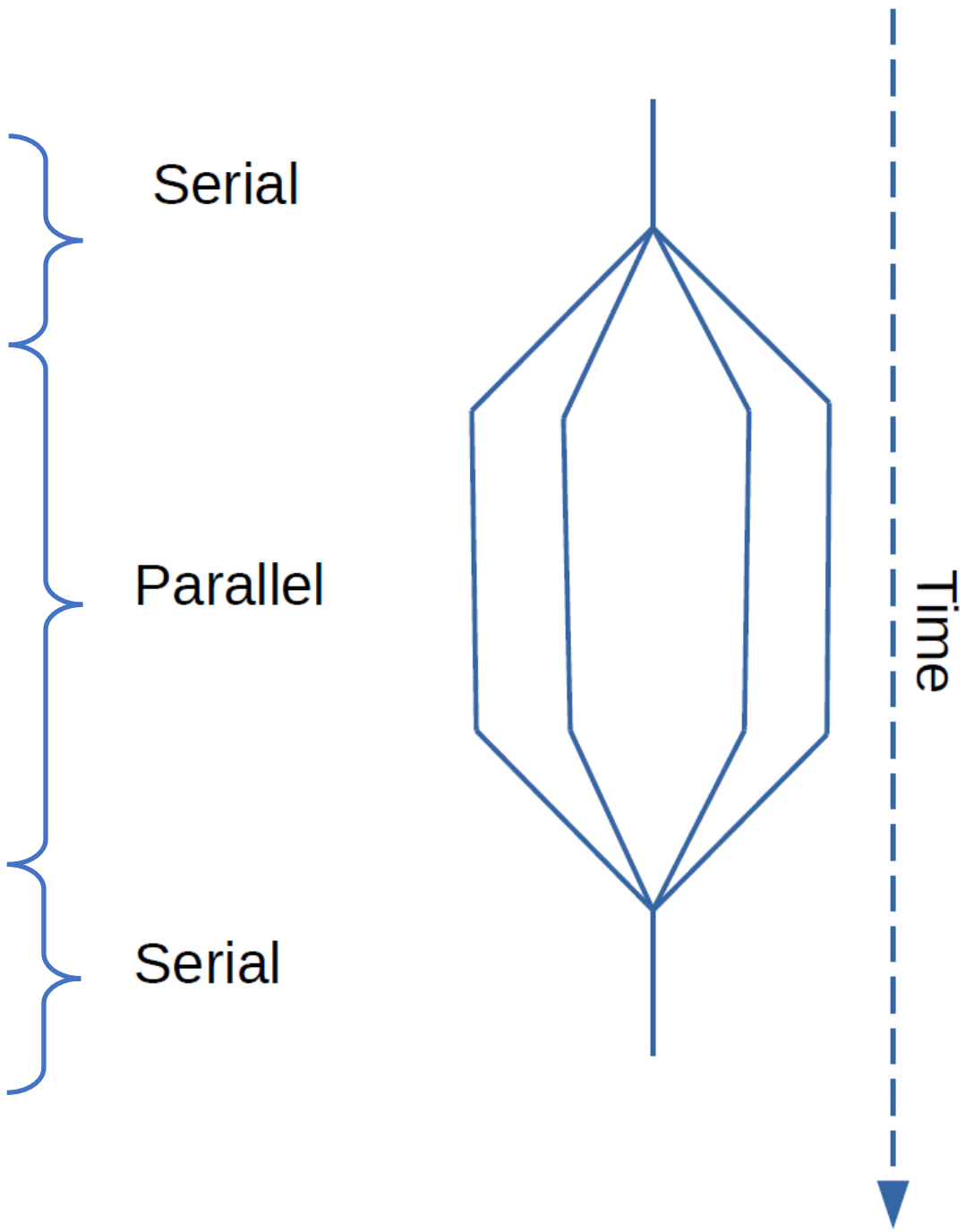
OpenMP

- Application Program Interface (API) for shared memory programming
- Managed by OpenMP Architecture Review Board (or OpenMP ARB)
- Consists of compiler directives, library routines, and environment variables
- Portable and scalable
- Supports C/C++, Fortran

Fork-Join Model

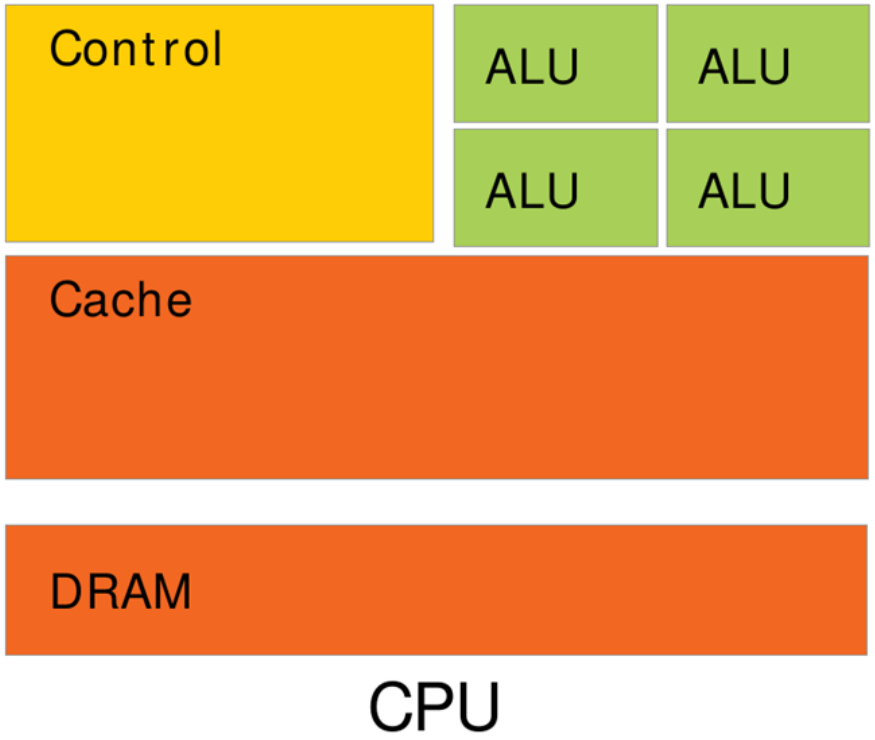


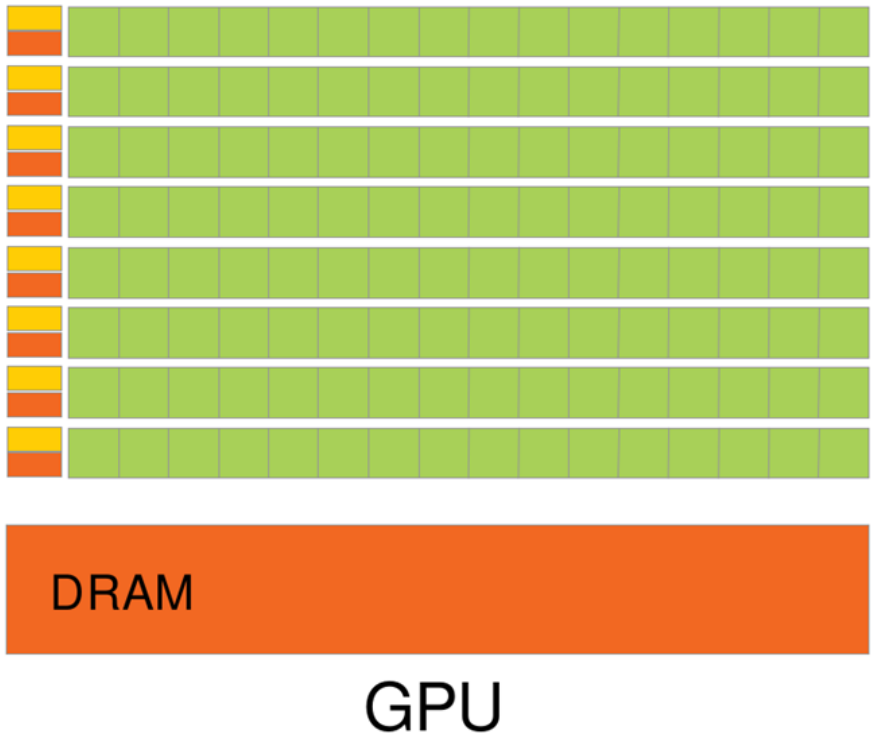
```
int i = 10000;
…
#pragma omp parallel for
for(j=0;i<i;j++)
{
        C[j] = A[j] + B[j];
}
…
printf("\nProgram Exit!\n");
```



Shared Memory Programming using GP-GPUs

CPU vs GPU





Source: CUDA C Programming Guide (NVIDIA)

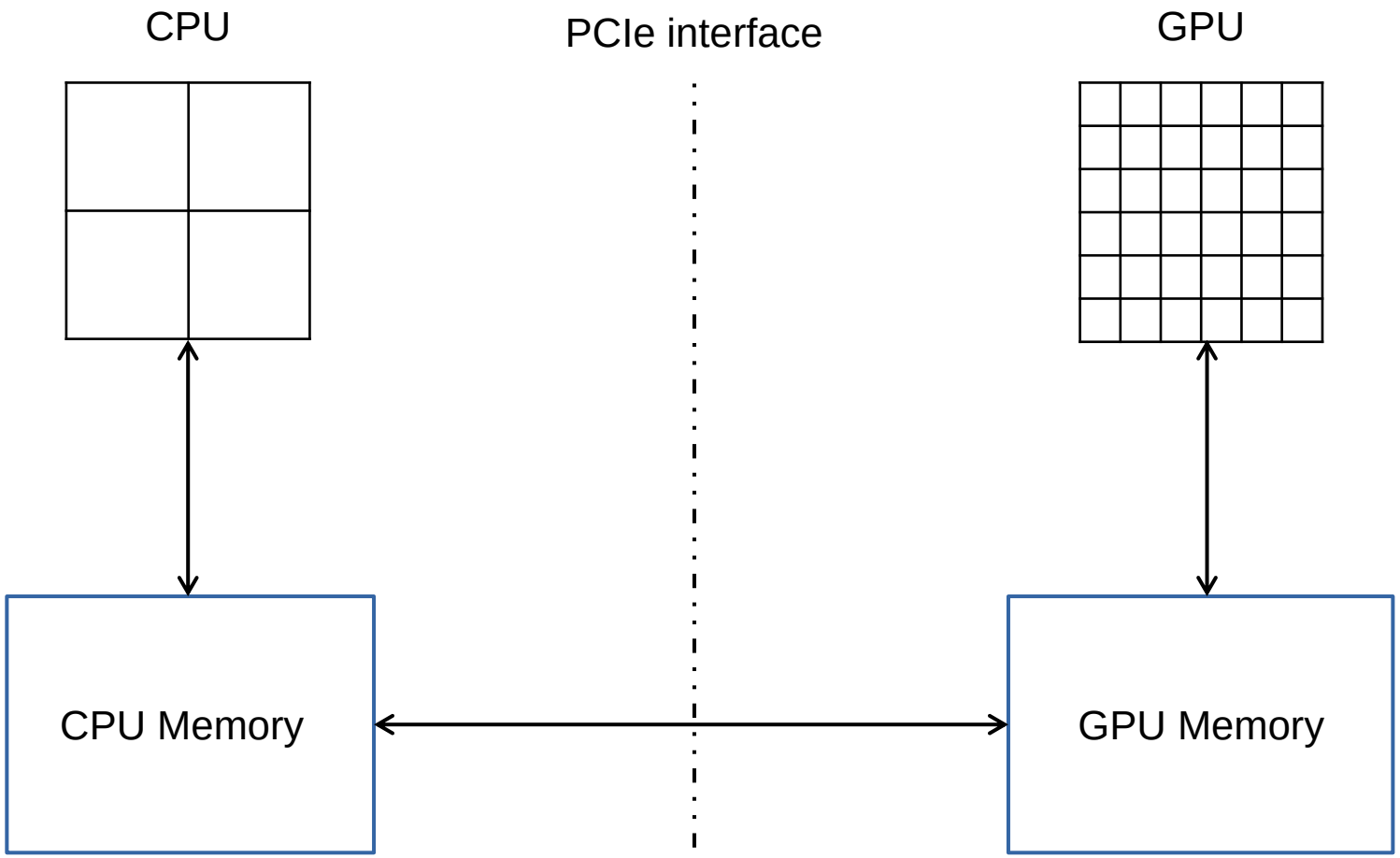
CPU vs GPU (contd)

- CPU
  - Optimized for serial thread performance
  - Good for complex tasks
  - Few, large, complex cores
  - Large number of transistors are allocated for Caches, Instruction Level Parallelism
- GPU
  - Optimized for data parallel, throughput computations
  - Large number of very small, simple cores
  - More number of transistors are allocated for computations

CUDA

- Parallel computing platform and application programming interface (API)
- Nvidia Corp.- General Purpose Graphics Processing Unit (GPGPU)
- Supports most of the today's Nvidia's gaming cards.
- Platforms
  - GeForce : Desktop
  - Quadro : Workstation
  - Tesla : Datacentre
  - Tegra, Jetson, Drive : Embedded

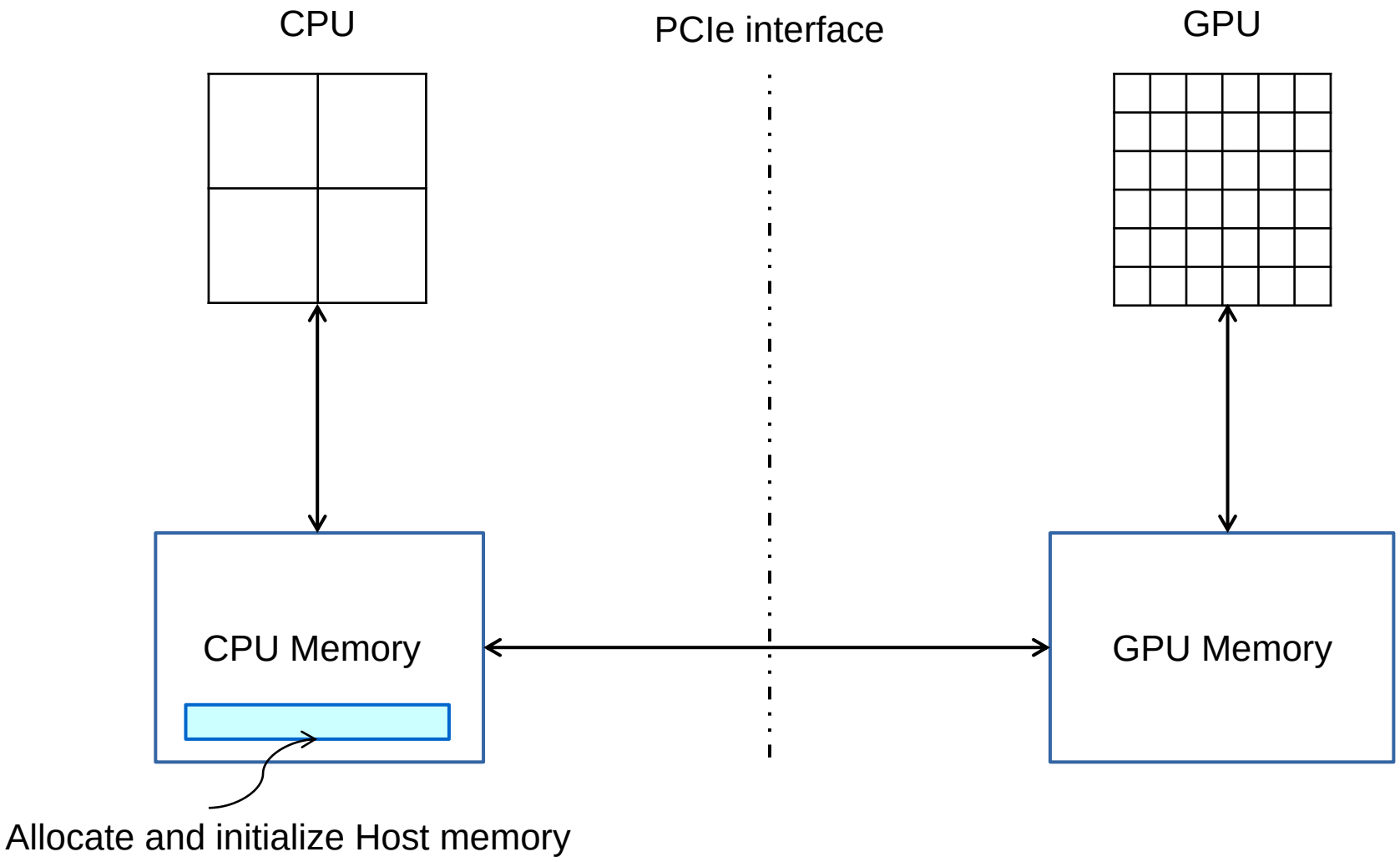
Programmer's view



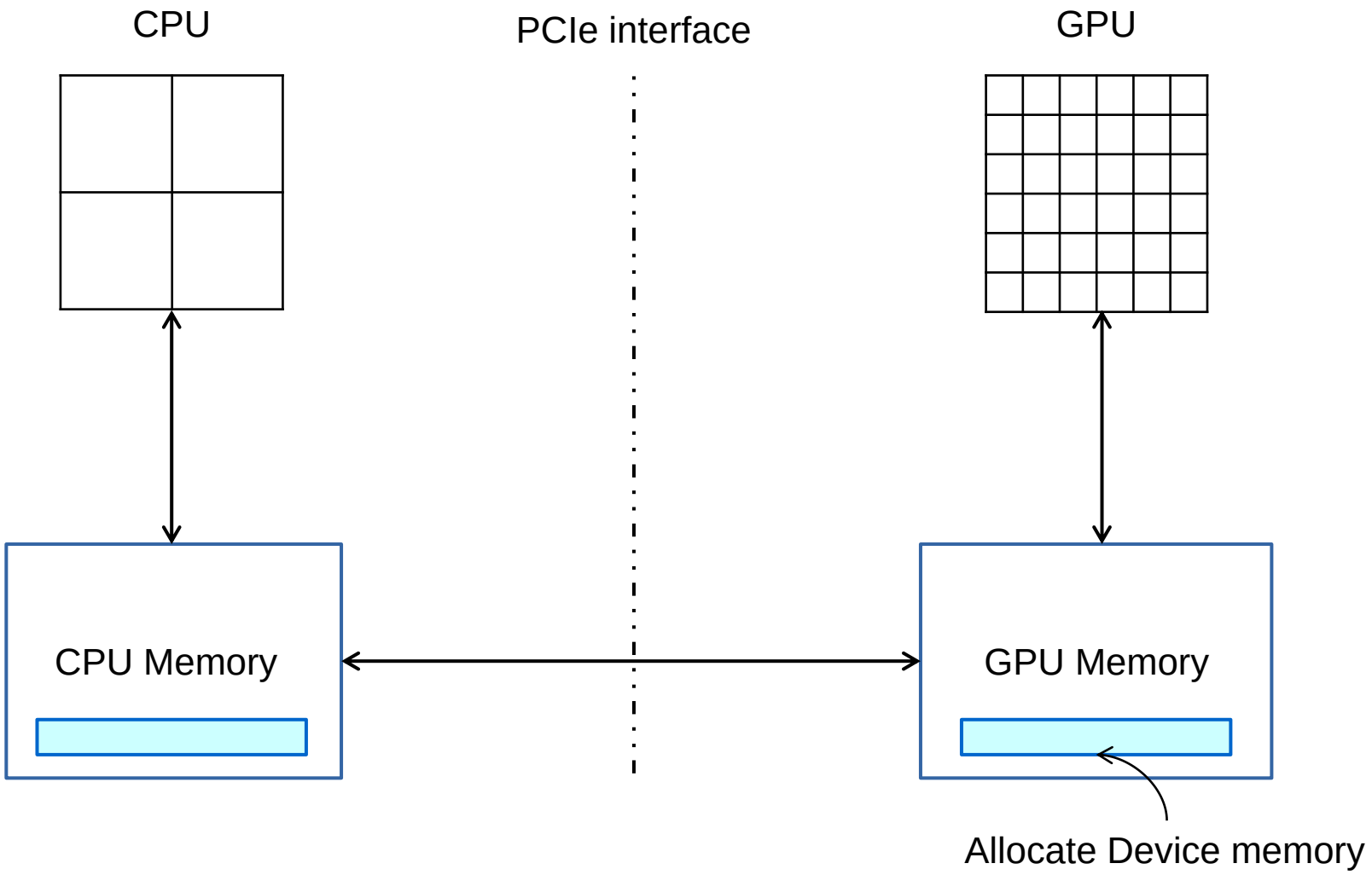
Pseudo Code

1. Allocate and initialize memory on Host (CPU)
2. Allocate memory on Device (GPU)
3. Transfer data from Host memory to Device memory
4. Launch “kernel” on the Device – large number of threads execute in parallel on Device
5. Transfer results from Device memory to Host memory
6. De-allocate all the memory and terminate the program

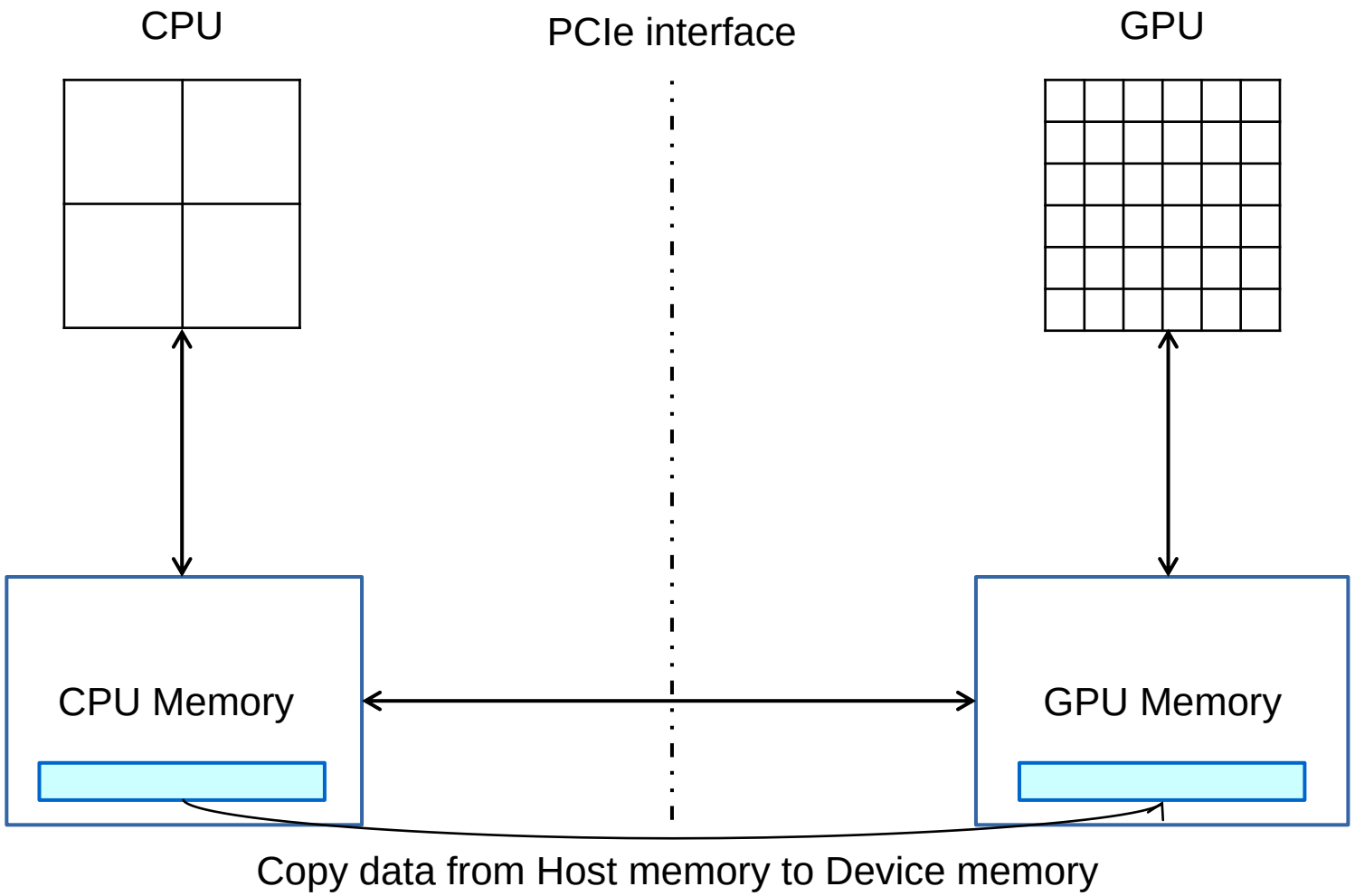
Step 1


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Allocate and initialize Host memory

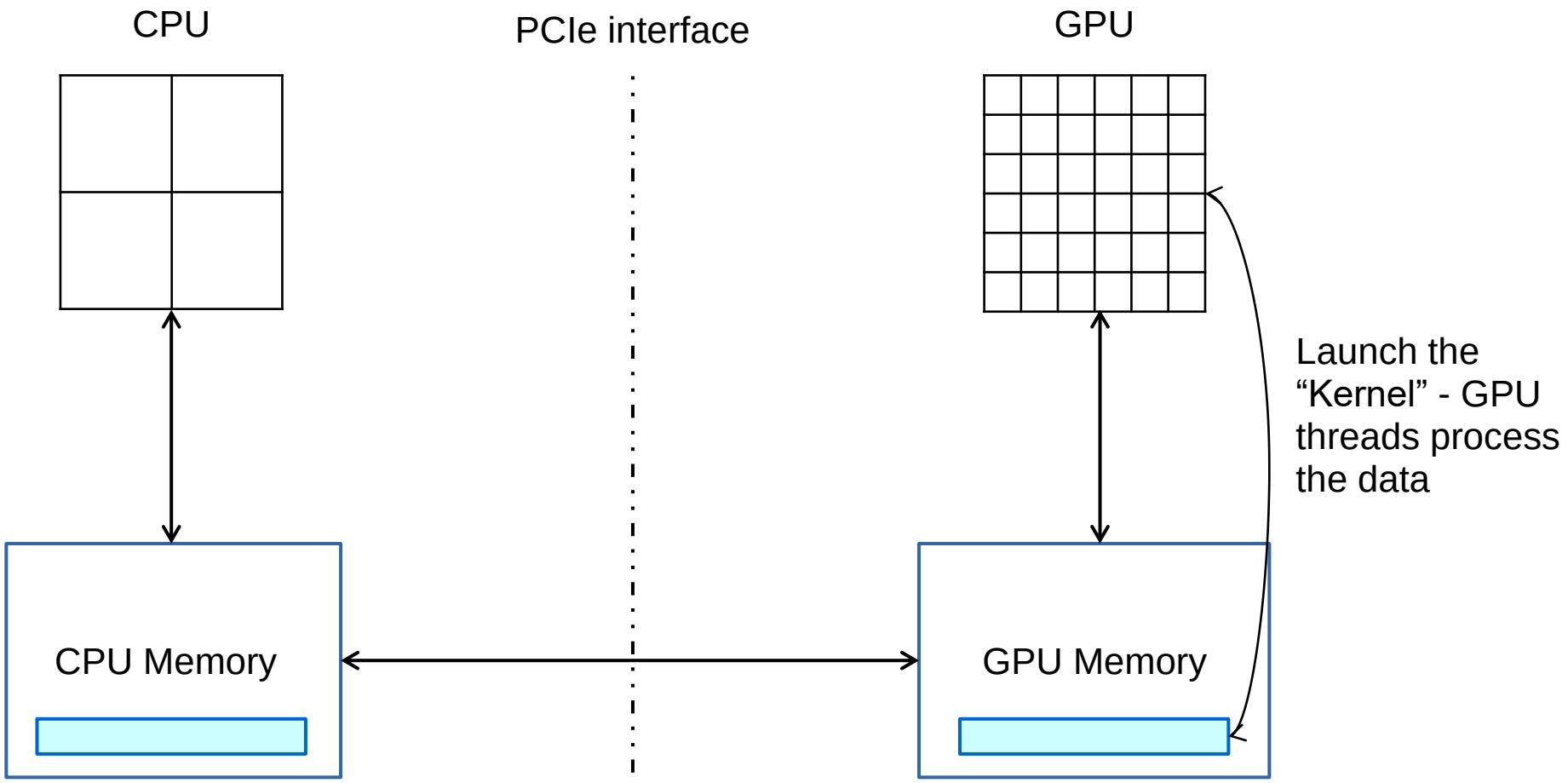
Step 2


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Allocate Device memory

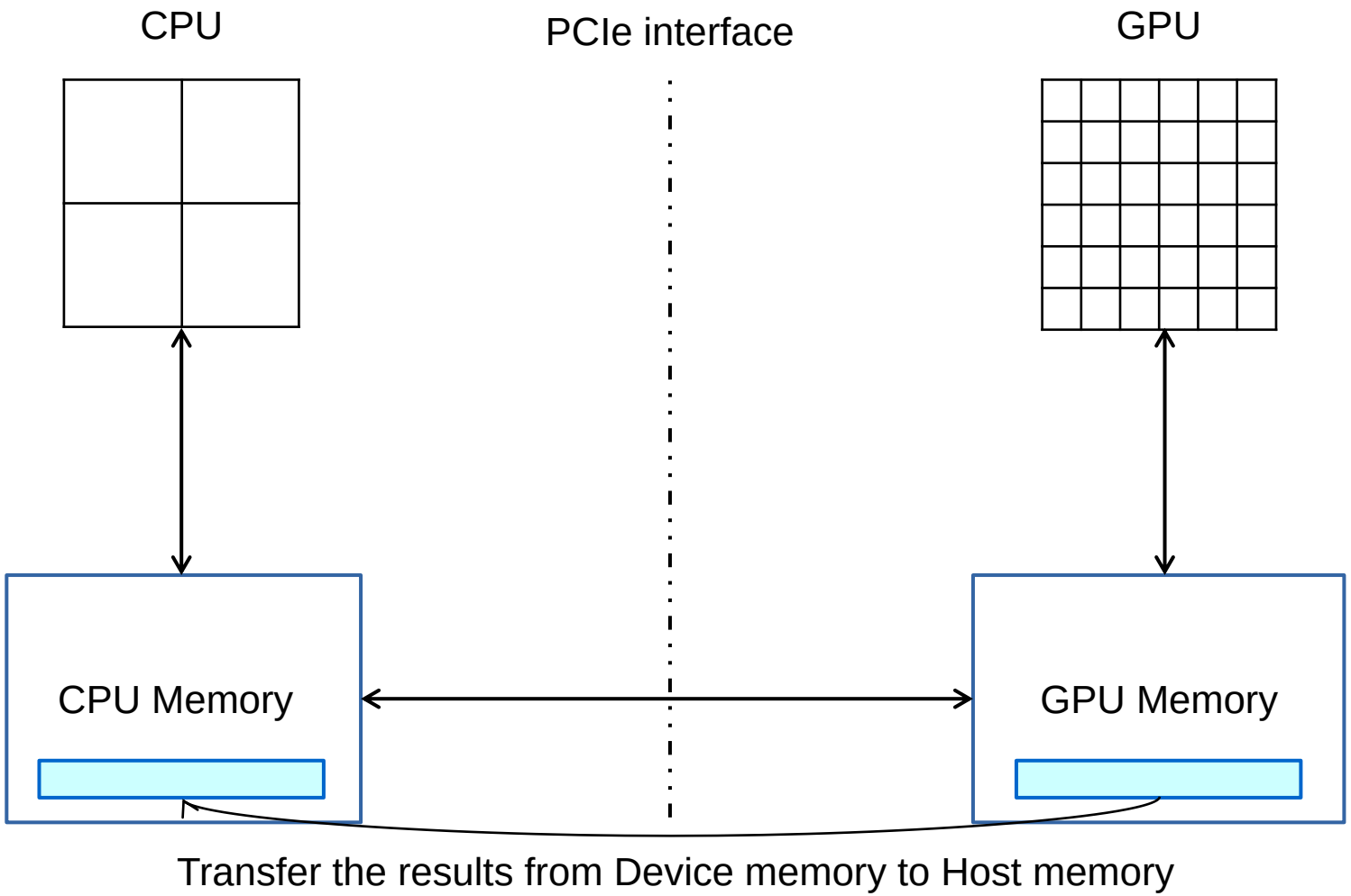
Step 3


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Copy data from Host memory to Device memory

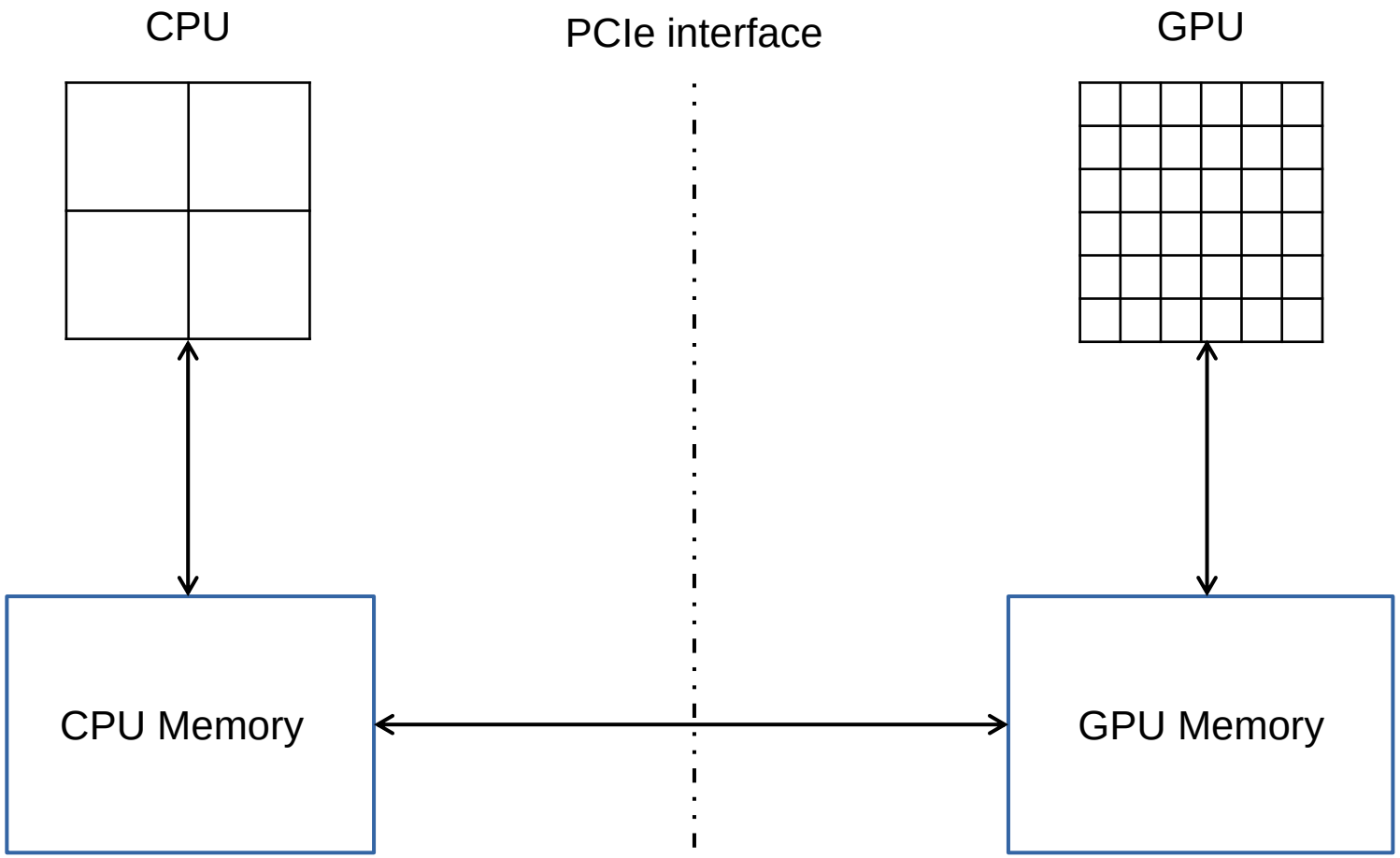
Step 4


CPU
PCIe interface
GPU
Launch the "Kernel" - GPU threads process the data
CPU Memory
GPU Memory

Step 5


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Transfer the results from Device memory to Host memory

Step 6



De-allocate the memories and terminate the program

Thank you!

mandar.hpsc@gmail.com