



Course Overview

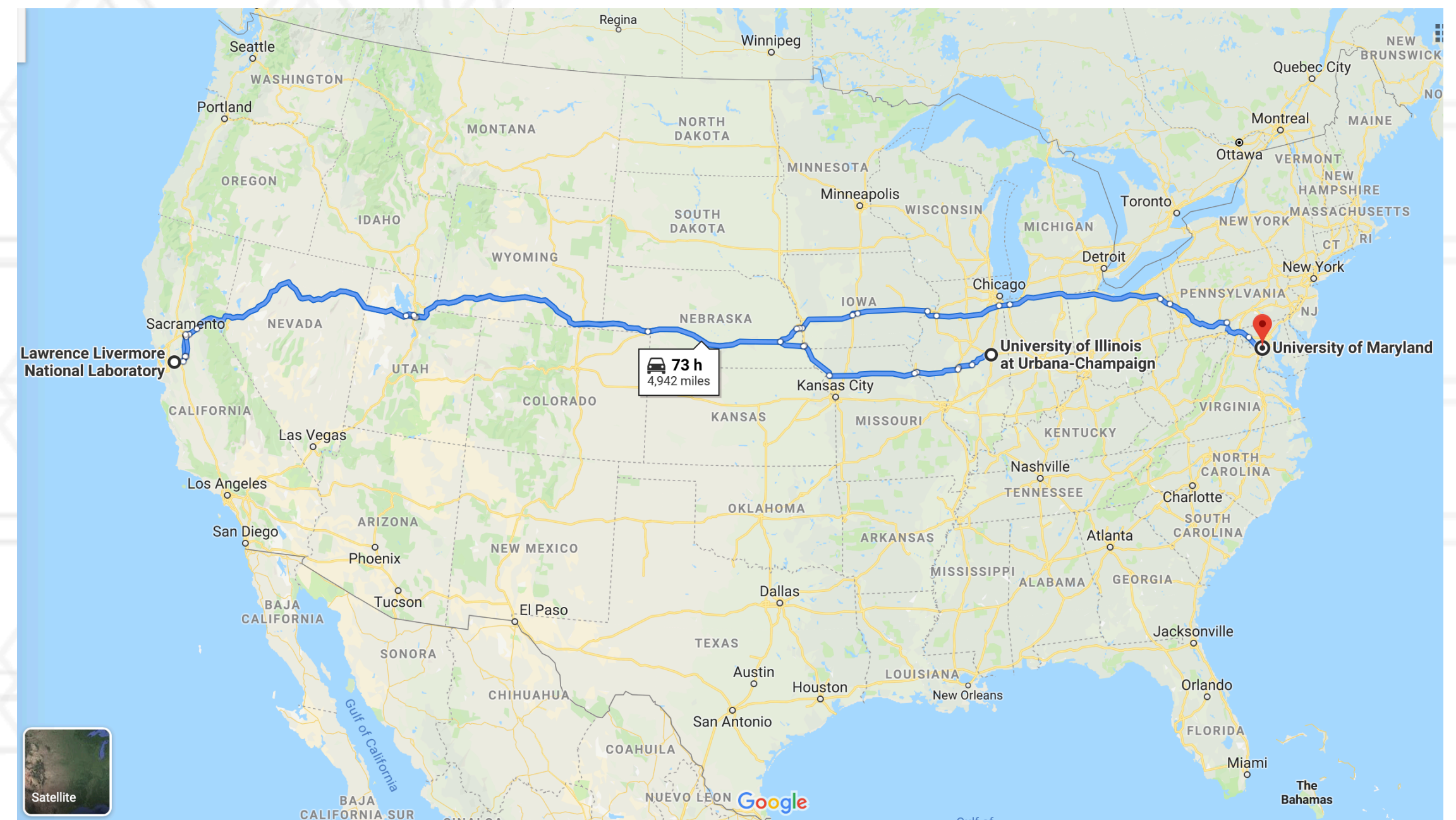
Abhinav Bhatele, Department of Computer Science



UNIVERSITY OF
MARYLAND

About the instructor — Dr. Bhatele

- Ph.D. from the University of Illinois at Urbana-Champaign (midwest)
- Spent eight years at Lawrence Livermore National Laboratory (SF bay area)
- 8th year at the University of Maryland
- Research areas:
 - High performance computing
 - Distributed AI



Getting to know you all ...

- Name
- Junior/Senior/MS/PhD student
- Something interesting/unique about yourself
- **(optional)** *Why this course?*

This course is

- An introduction to parallel computing
- 416: Upper Level CS Coursework / General Track / Area I: Systems
- 616: Qualifying course for MS/PhD —
 - Before Fall 2026: Area — Computer Systems
 - Starting Fall 2026: Area — Systems, SE, PL, Architecture (SSPA)
- Work expected:
 - Four to five programming assignments
 - Three to five quizzes (no advance notice)
 - Two midterm exams, and final exam

Course topics

- Introduction to parallel computing (1 week)
- Shared-memory parallel programming (1 week)
- Parallel algorithm design (2 weeks)
- Distributed memory parallel programming (3 weeks)
- Performance analysis and tools (2 weeks)
- GPU programming (1 week)
- Parallel architectures and networks (1 week)
- Parallel simulation codes (2 weeks)

Tools we will use for the class

- Syllabus, lecture slides, programming assignment descriptions on course website:
 - <https://www.cs.umd.edu/class/fall2026/cmsc416>
- Programming assignment submissions on gradescope
 - <https://www.gradescope.com/courses/1367315>
- Quizzes in class or on ELMS
- Discussions: Piazza
 - <https://piazza.com/umd/fall2026/cmsc416cmsc616>
- If you want to contact the course staff outside of piazza, send an email to:
cmsc416@cs.umd.edu

Zaratan accounts

- Zaratan is the UMD DIT cluster we'll use for the programming assignments
- You should have received an email about account creation
- Helpful resources:
 - <https://hpcc.umd.edu/getting-started>
 - <https://missing.csail.mit.edu>
 - <https://www.cs.umd.edu/~mmarsh/books/cmdline/cmdline.html>
 - <https://www.cs.umd.edu/~mmarsh/books/tools/tools.html>

Programming assignments

- You can write and debug most of your assignment locally:
 - Use a virtual linux box
 - Docker containers
 - MacOS: use macports or homebrew
- On zaratan:
 - Use command line editors such as vim or emacs

Excused absence

Any student who needs to be excused for an absence from a single lecture, due to a medically necessitated absence shall make a reasonable attempt to inform the instructor of his/her illness prior to the class. Upon returning to the class, present the instructor with a self-signed note attesting to the date of their illness. Each note must contain an acknowledgment by the student that the information provided is true and correct. Providing false information to University officials is prohibited under Part 9(i) of the Code of Student Conduct (V-I.00(B) University of Maryland Code of Student Conduct) and may result in disciplinary action.

Self-documentation may **not be used for Major Scheduled Grading Events (midterm and final exams)** and it may only be used for one class meeting during the semester. Any student who needs to be excused for a prolonged absence (two or more consecutive class meetings), or for a Major Scheduled Grading Event, must provide written documentation of the illness from the Health Center or from an outside health care provider. This documentation must verify dates of treatment and indicate the timeframe that the student was unable to meet academic responsibilities. In addition, it must contain the name and phone number of the medical service provider to be used if verification is needed. No diagnostic information will ever be requested.

Use of AI tools for programming assignments

AI tool disclosure: If a student chooses to use an AI tool to assist in any course work (e.g. assignments, programs, reports, etc), they must disclose this information to the instructor. This disclosure should include the name of the AI tool and explain how it was used.

Consequences for non-compliance: Failure to adhere to this policy may result in a zero on the particular course work where the AI tool is used. In addition the university honor code is applicable here: violation of the honor code and appropriate action will be enforced.

Academic dishonesty

- In the last few years, we have discovered that 10-15% of students have been involved in academic dishonesty each semester
- We use software to detect plagiarism in programming assignments
- If we suspect something, we must report it to the Office of Student Conduct (OSC)
- Sanctions:
 - Undergrads: grade of XF
 - Graduate student: grade of XF and **separation from the university**

<https://studentconduct.umd.edu/conduct-process/academic-misconduct-process>

Announcements

- Zaratan accounts have been created for everyone
- Quiz 0 has been posted on ELMS
- Assignment 0.1 has been posted

What is parallel computing?

- Serial or sequential computing: doing a task in sequence on a single processor
- Parallel computing: breaking up a task into sub-tasks and doing them in parallel (concurrently) on a set of processors (often connected by a network)
- Some tasks do not need any communication: embarrassingly parallel

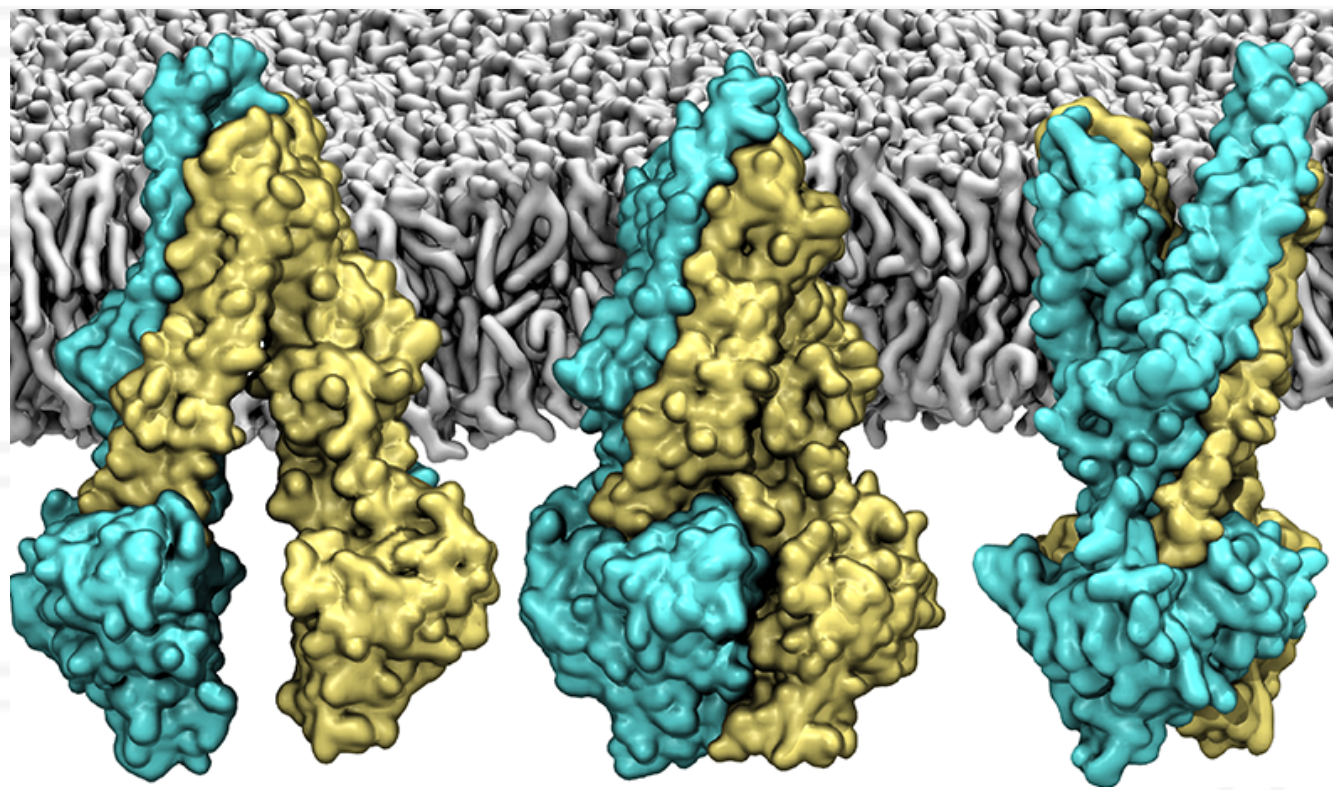
What is parallel computing?

- Does it include:
 - Grid computing: processors are dispersed geographically
 - Distributed computing: processors connected by a network
 - Cloud computing: on-demand availability, typically pay-as-you-go model
- Does it include:
 - Superscalar processors
 - Vector processors
 - Accelerators (GPUs, FPGAs)

The need for parallel computing or HPC

HPC stands for High Performance Computing

Drug discovery

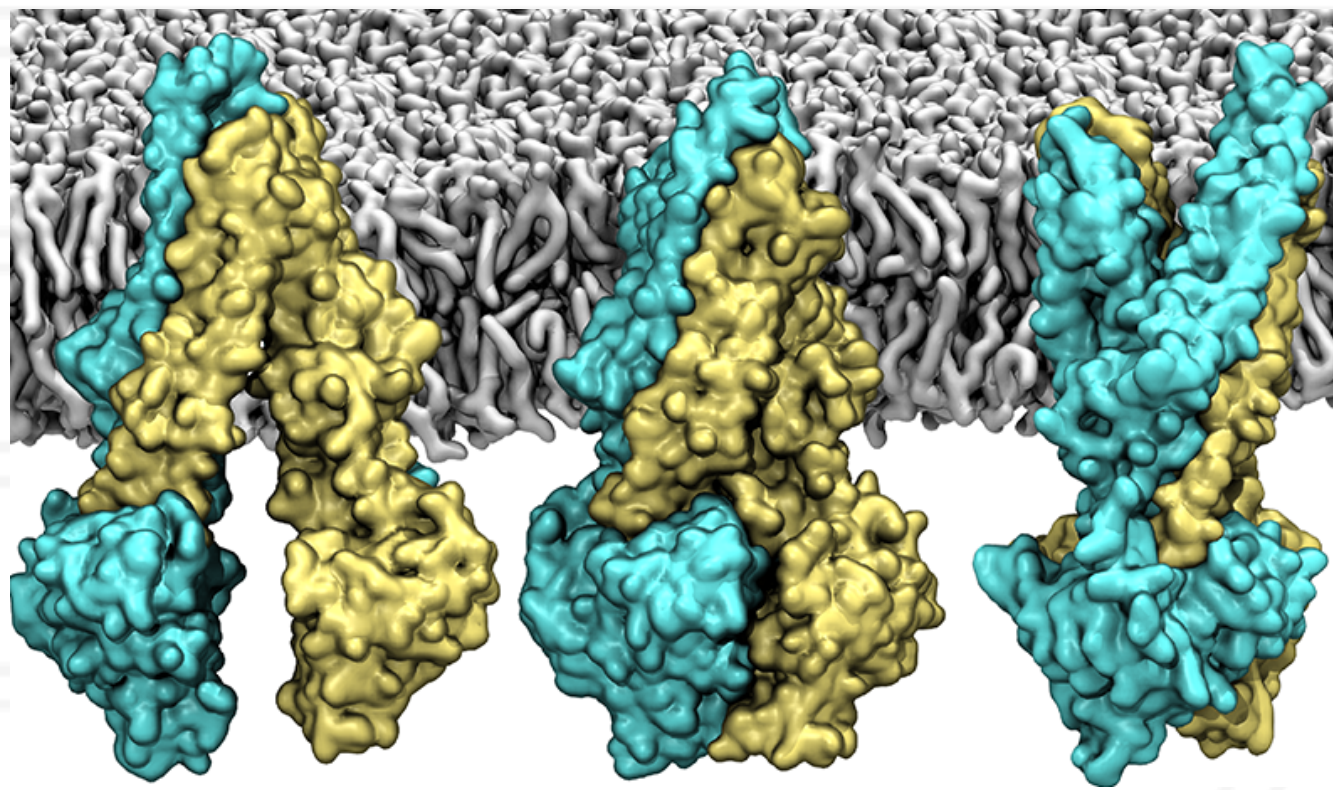


<https://www.nature.com/articles/nature21414>

The need for parallel computing or HPC

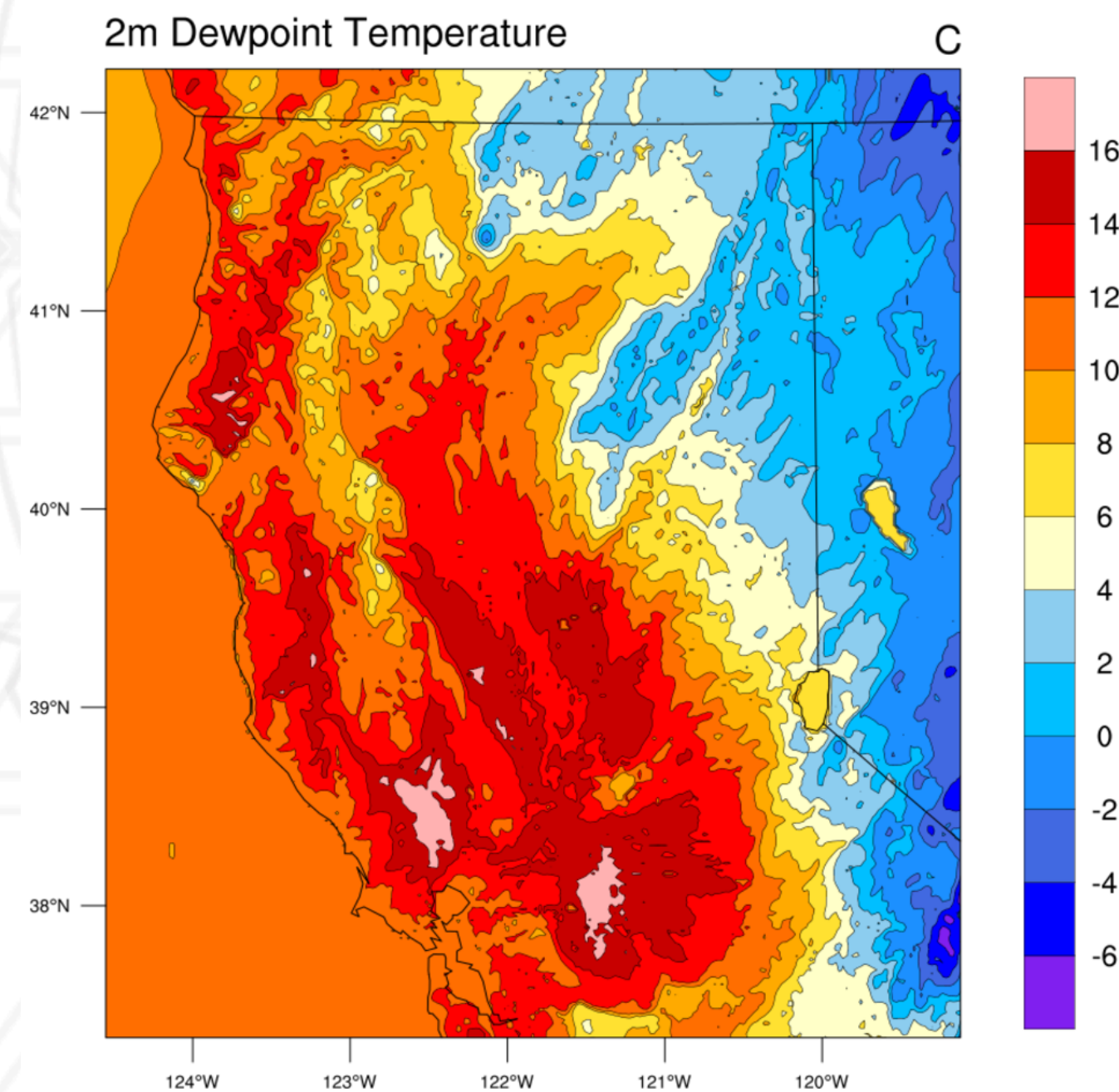
HPC stands for High Performance Computing

Drug discovery



<https://www.nature.com/articles/nature21414>

Weather forecasting

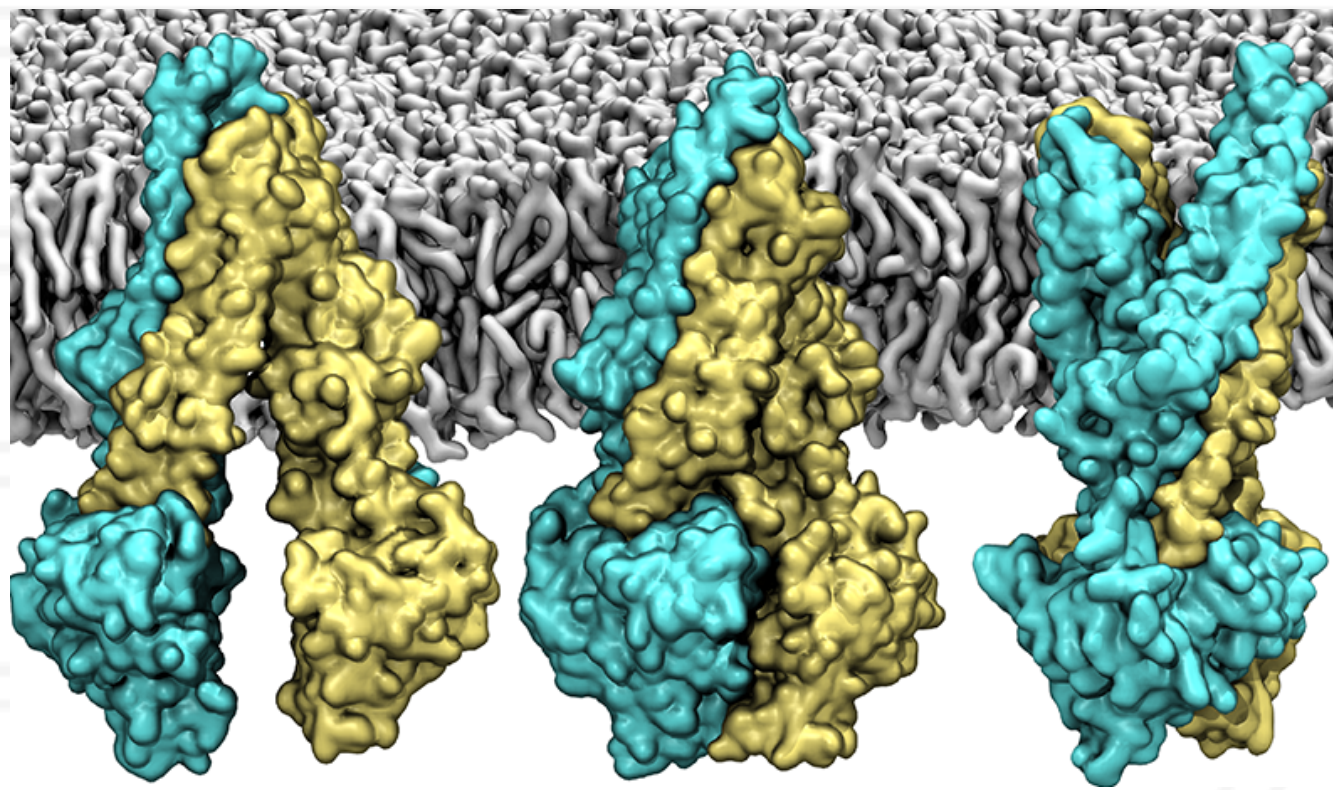


<https://www.ncl.ucar.edu/Applications/wrf.shtml>

The need for parallel computing or HPC

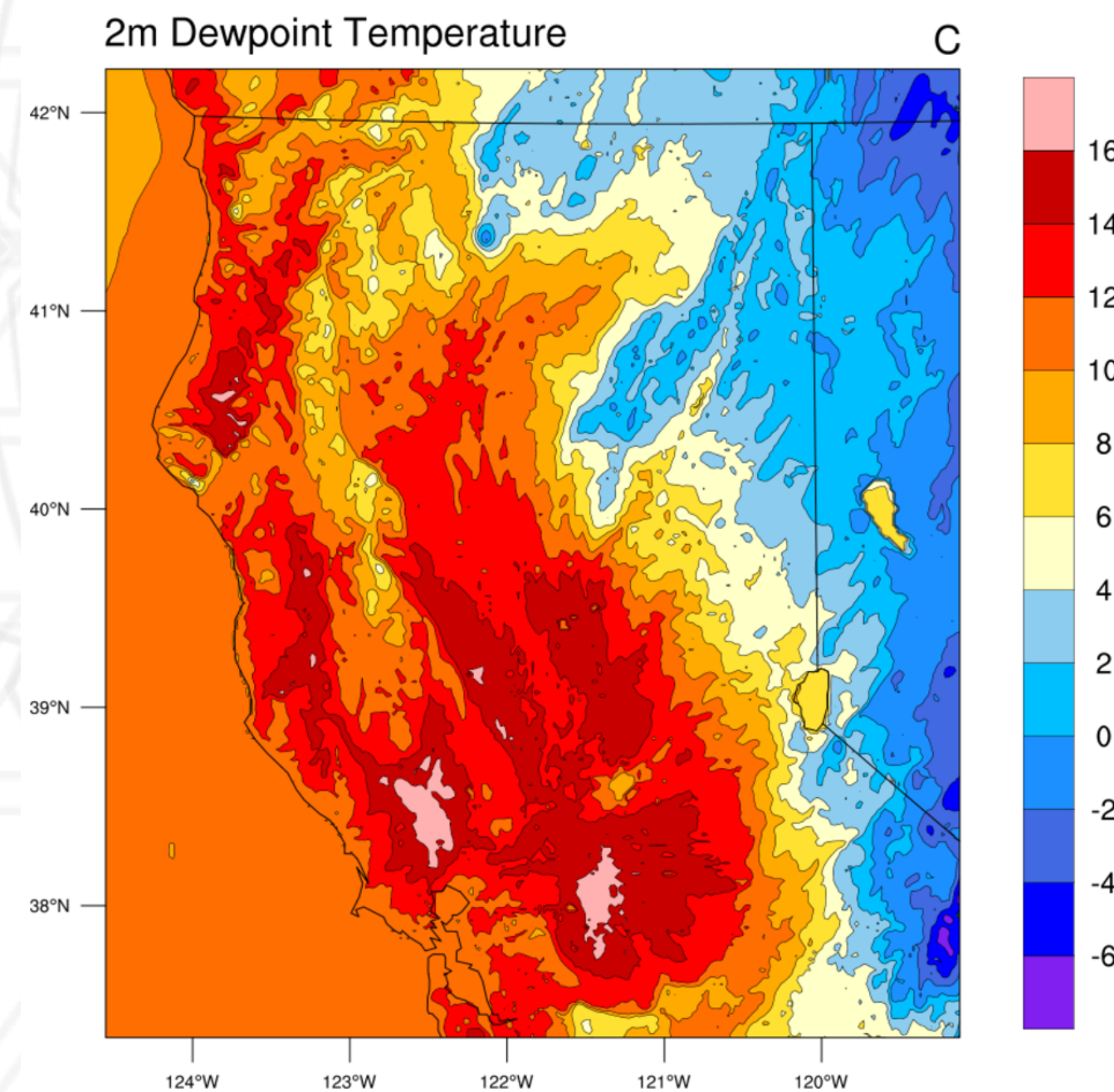
HPC stands for High Performance Computing

Drug discovery



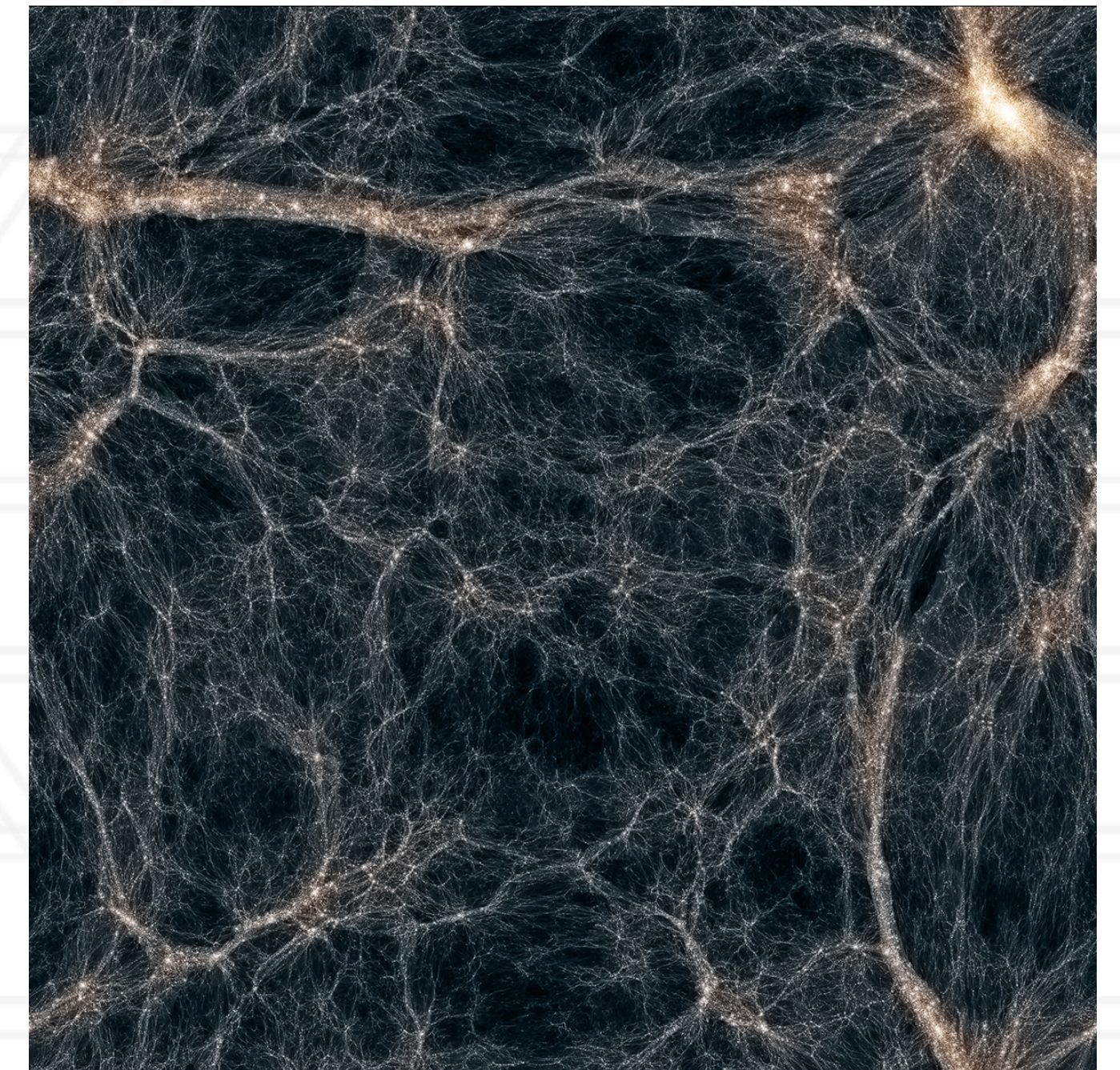
<https://www.nature.com/articles/nature21414>

Weather forecasting



<https://www.ncl.ucar.edu/Applications/wrf.shtml>

Study of the universe



<https://www.nasa.gov/SC14/demos/demo27.html>

Why do we need parallelism?

- Typical constraints are speed or memory requirements
 - Either it would take too long to do the simulations
 - Or the simulation data would not fit in the memory of a single processor
- **Speedup**: solve the same problem in less time
- **Sizeup**: solve a larger problem in the same time
- Make some science simulations feasible in the lifetime of humans
- Provide answers in realtime or near realtime

Large supercomputers

- Top500 list: <https://www.top500.org/lists/top500/2025/06>

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	JUPITER Booster - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN EuroHPC/FZJ Germany	4,801,344	793.40	930.00	13,088
5	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure United States	2,073,600	561.20	846.84	

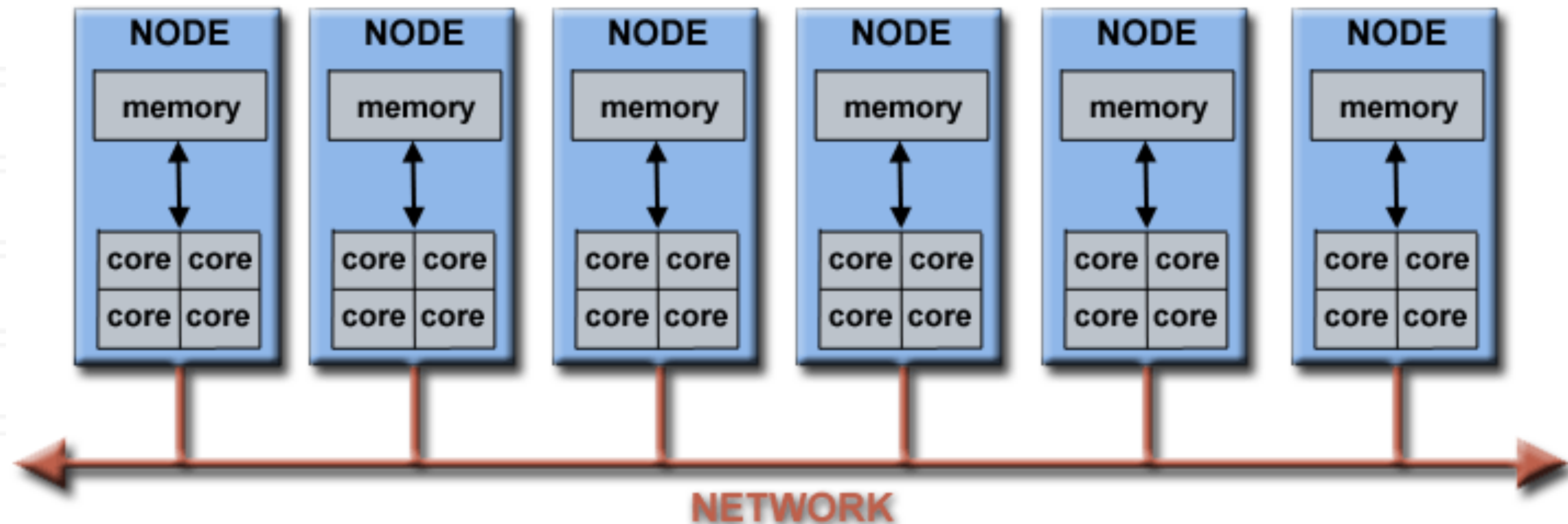


El Capitan @ LLNL

[https://en.wikipedia.org/wiki/El_Capitan_\(supercomputer\)](https://en.wikipedia.org/wiki/El_Capitan_(supercomputer))

Parallel architecture

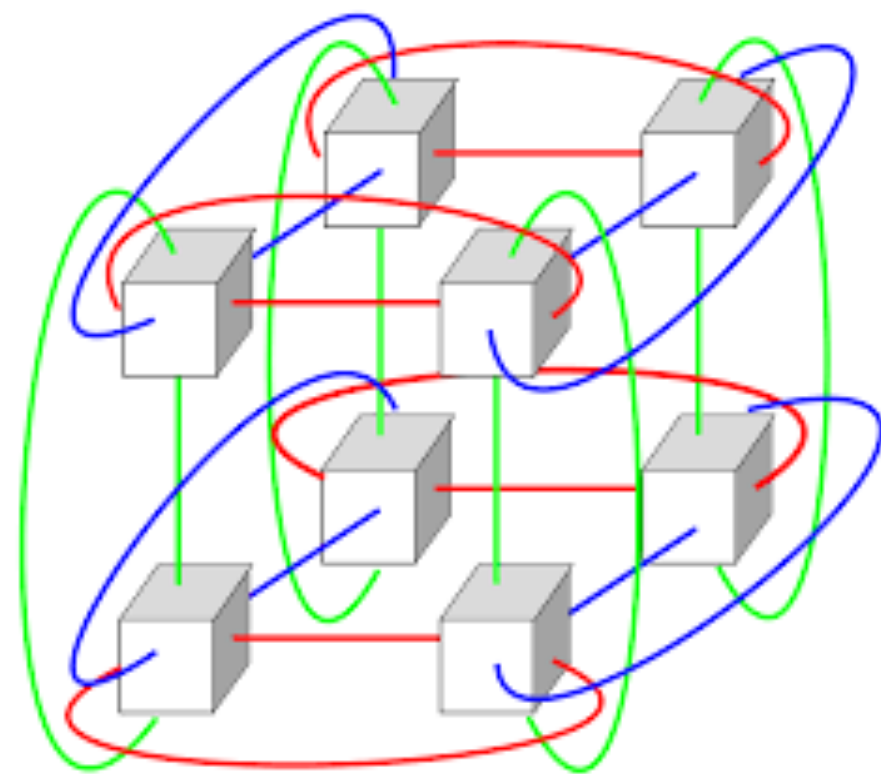
- A set of nodes or processing elements connected by a network.



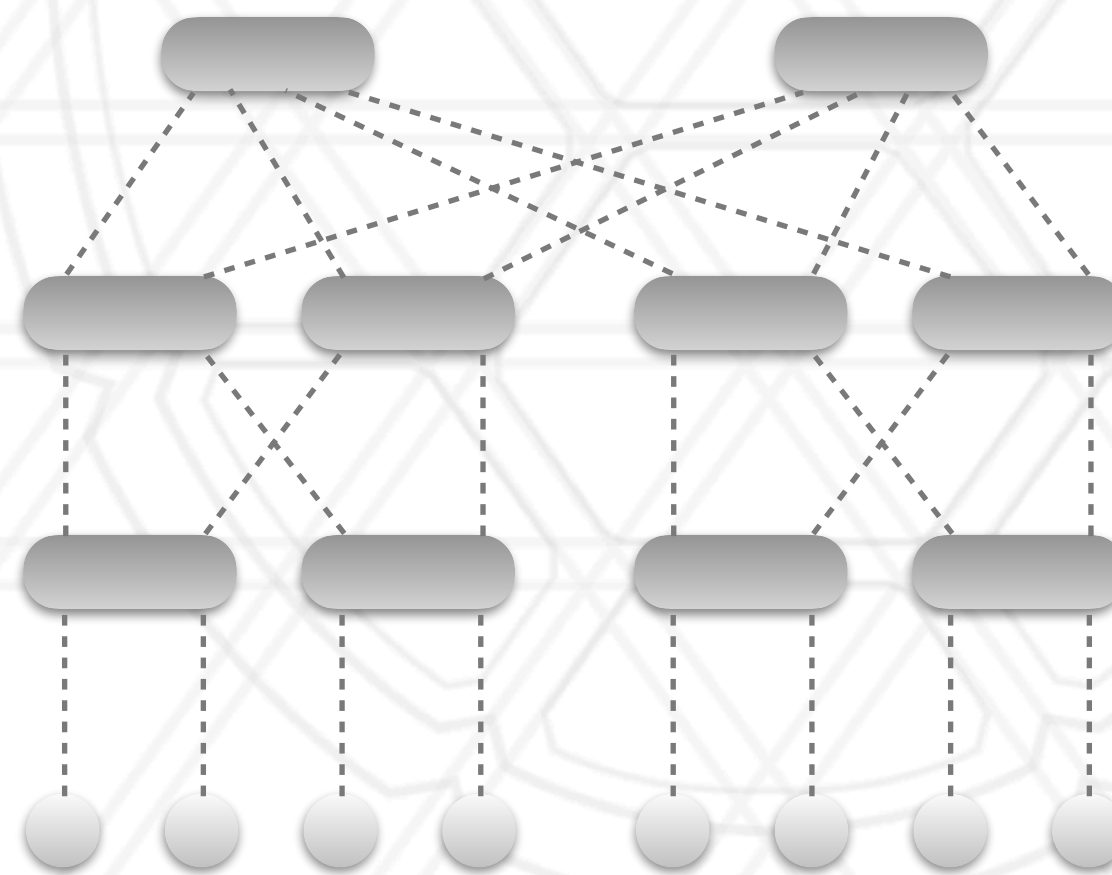
https://computing.llnl.gov/tutorials/parallel_comp

Interconnection networks

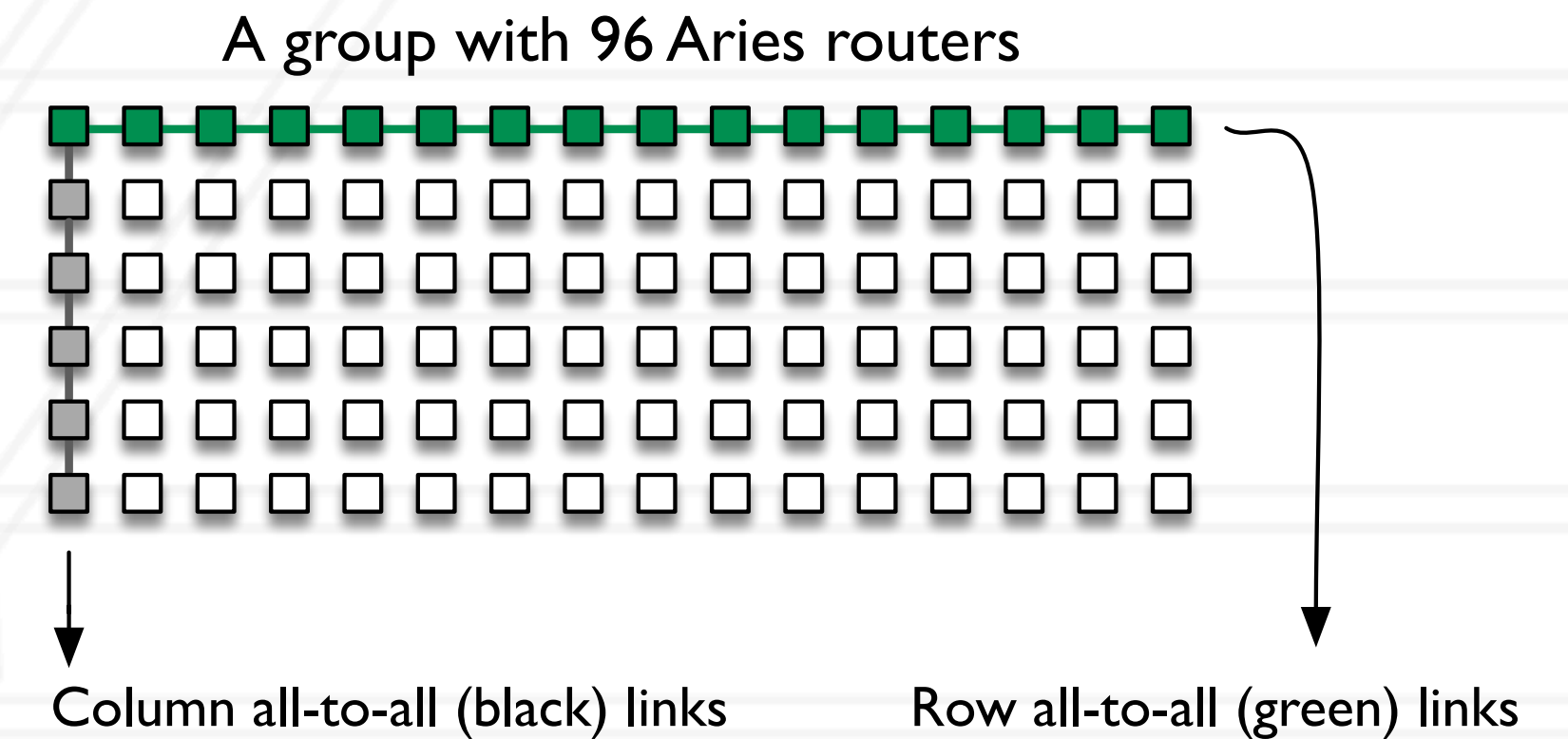
- Different topologies for connecting nodes together
- Used in the past: torus, hypercube
- More popular currently: fat-tree, dragonfly



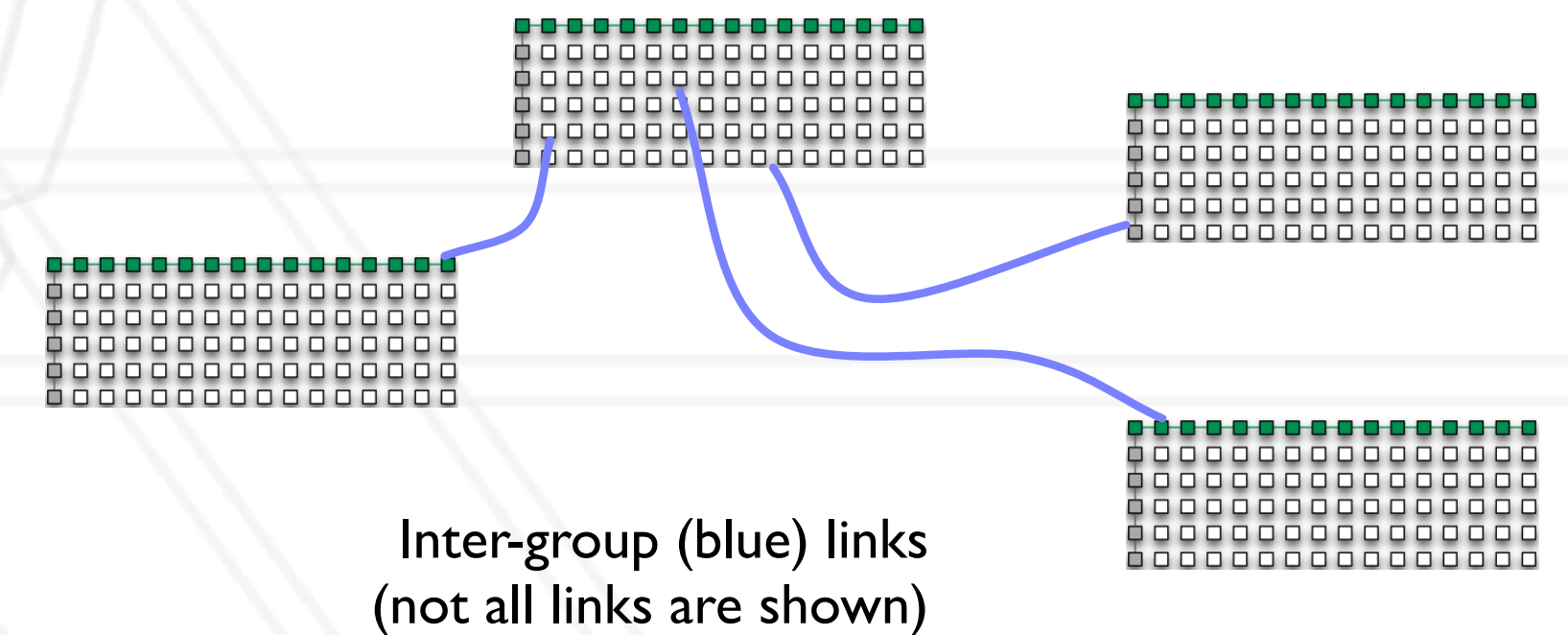
Torus



Fat-tree



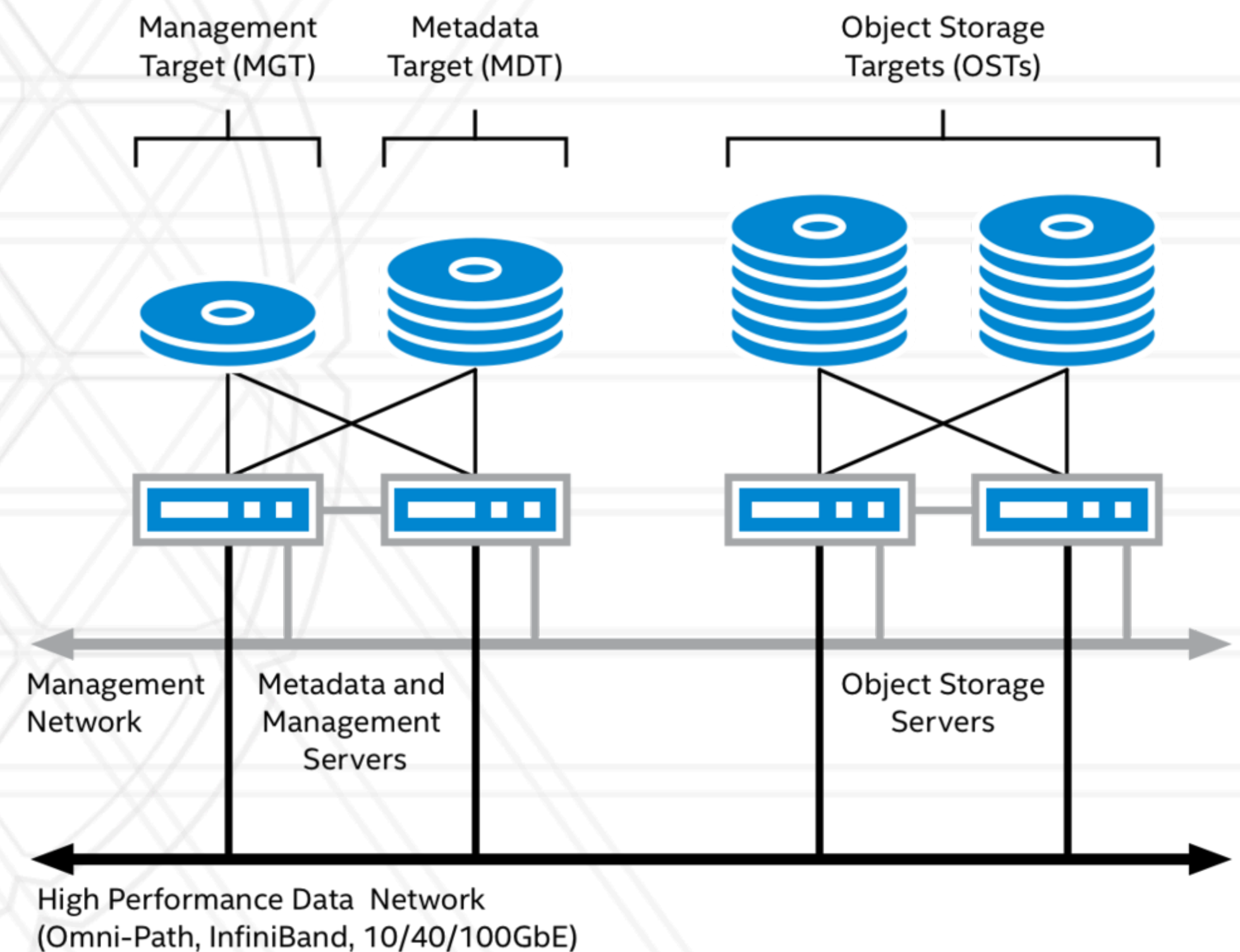
Two-level dragonfly with multiple groups



Dragonfly

I/O sub-system / Parallel file system

- Home directories and scratch space on clusters are typically on a parallel file system
- Compute nodes do not have local disks
- Parallel filesystem is mounted on all login and compute nodes



http://wiki.lustre.org/Introduction_to_Lustre

System software: models and runtimes

- Parallel programming model

- Parallelism is achieved through language constructs or by making calls to a library and the execution model depends on the model used.

User code

- Parallel runtime [system]:

- Implements the parallel execution model

Parallel runtime

- Shared memory/address-space

- Pthreads, OpenMP, Chapel

Communication library

- Distributed memory

- MPI, Charm

Operating system



Terminology and Definitions

Abhinav Bhatele, Department of Computer Science



UNIVERSITY OF
MARYLAND

Announcements

- Zaratan accounts have been created for everyone
- AI policy for programming assignments:
 - Use of any AI tools for coding is allowed — you should mention in your report how and to what extent did you use them?
- Collaborating on programming assignments is not allowed
- Please send accommodation letters

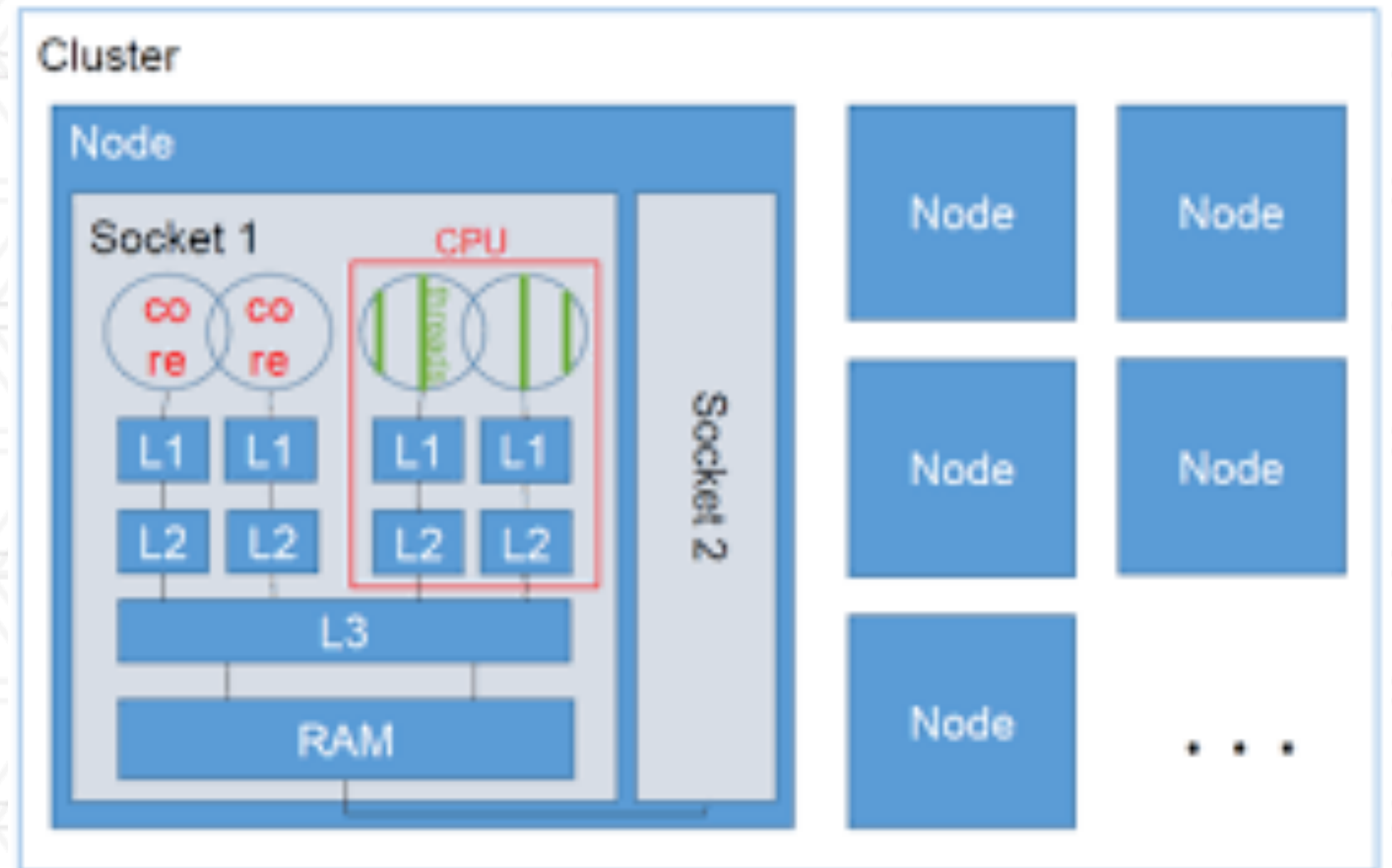
Getting started with zaratan

- Over 360 nodes with AMD Milan processors (128 cores/node, 512 GB memory/node)
- 20 nodes with four NVIDIA A100 GPUs (40 GB per GPU)

```
ssh username@login.zaratan.umd.edu
```

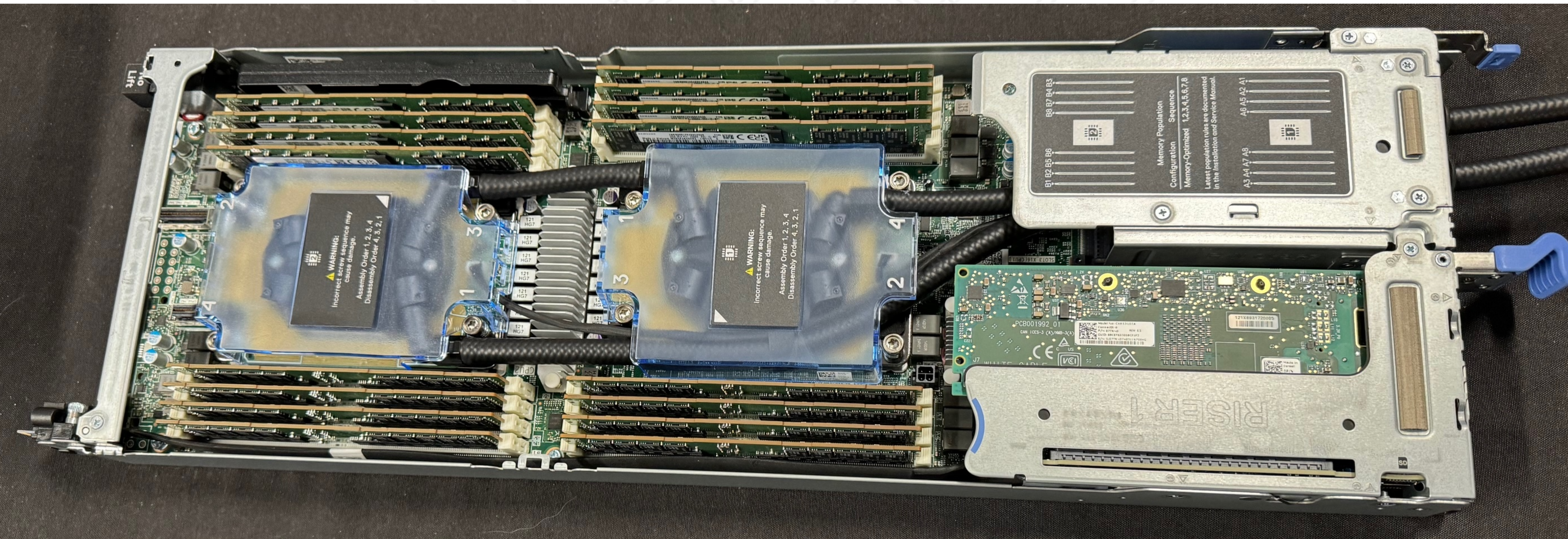
Cores, sockets, nodes

- Core: a single execution unit that has a private ALU, registers, L1 cache, etc. and can execute instructions independently
- Processor: several cores on a single Integrated Circuit (IC) or chip are called a multi-core processor
 - Cores share higher-level caches and main memory
- Socket: physical connector into which an IC/chip or processor is inserted.
- Node: a packaging of sockets — motherboard or printed circuit board (PCB) that has multiple sockets



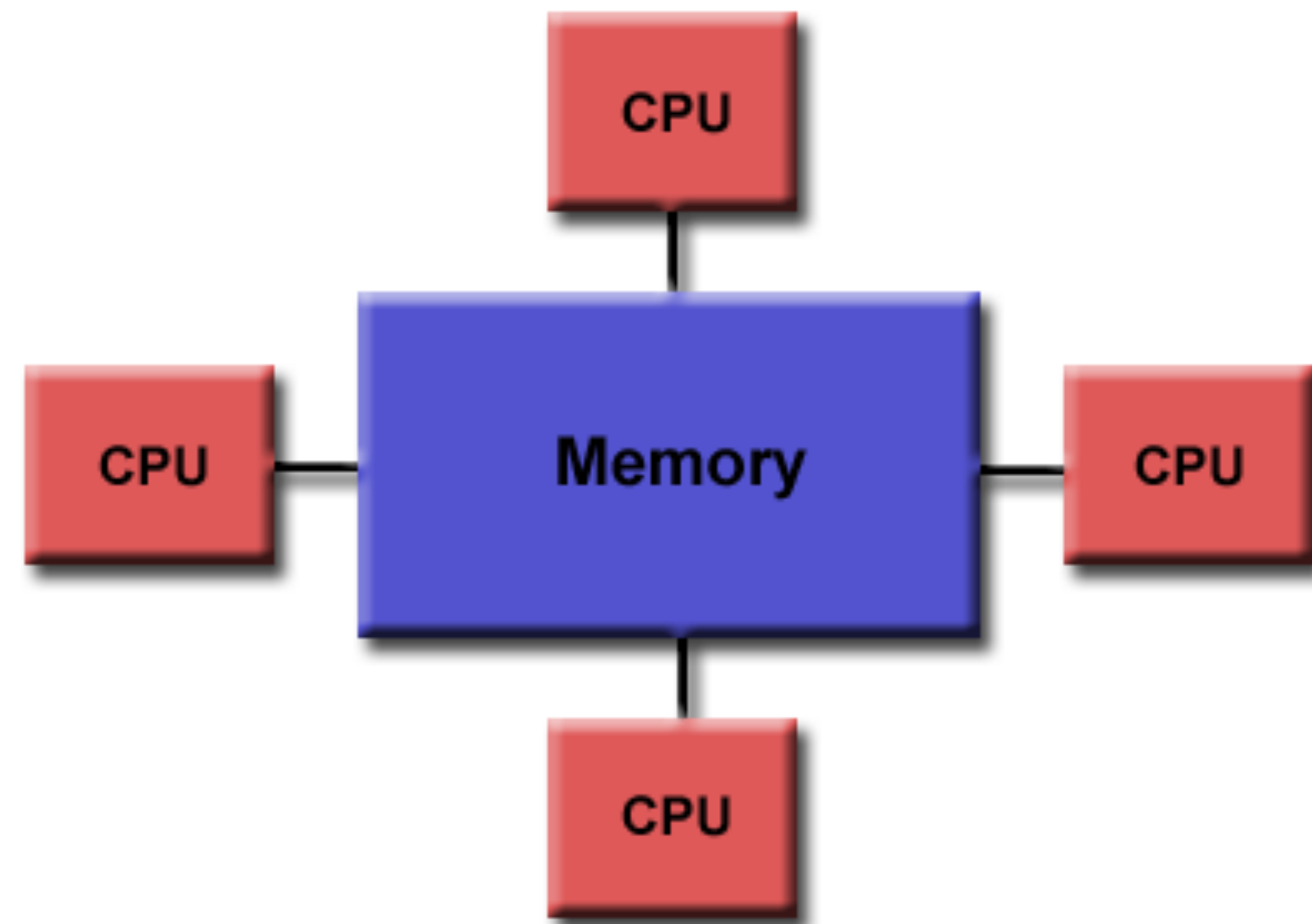
<https://hpc-wiki.info/hpc/HPC-Dictionary>

Zaratan CPU compute node



Shared memory architecture

- All processors/cores can access all memory as a single address space

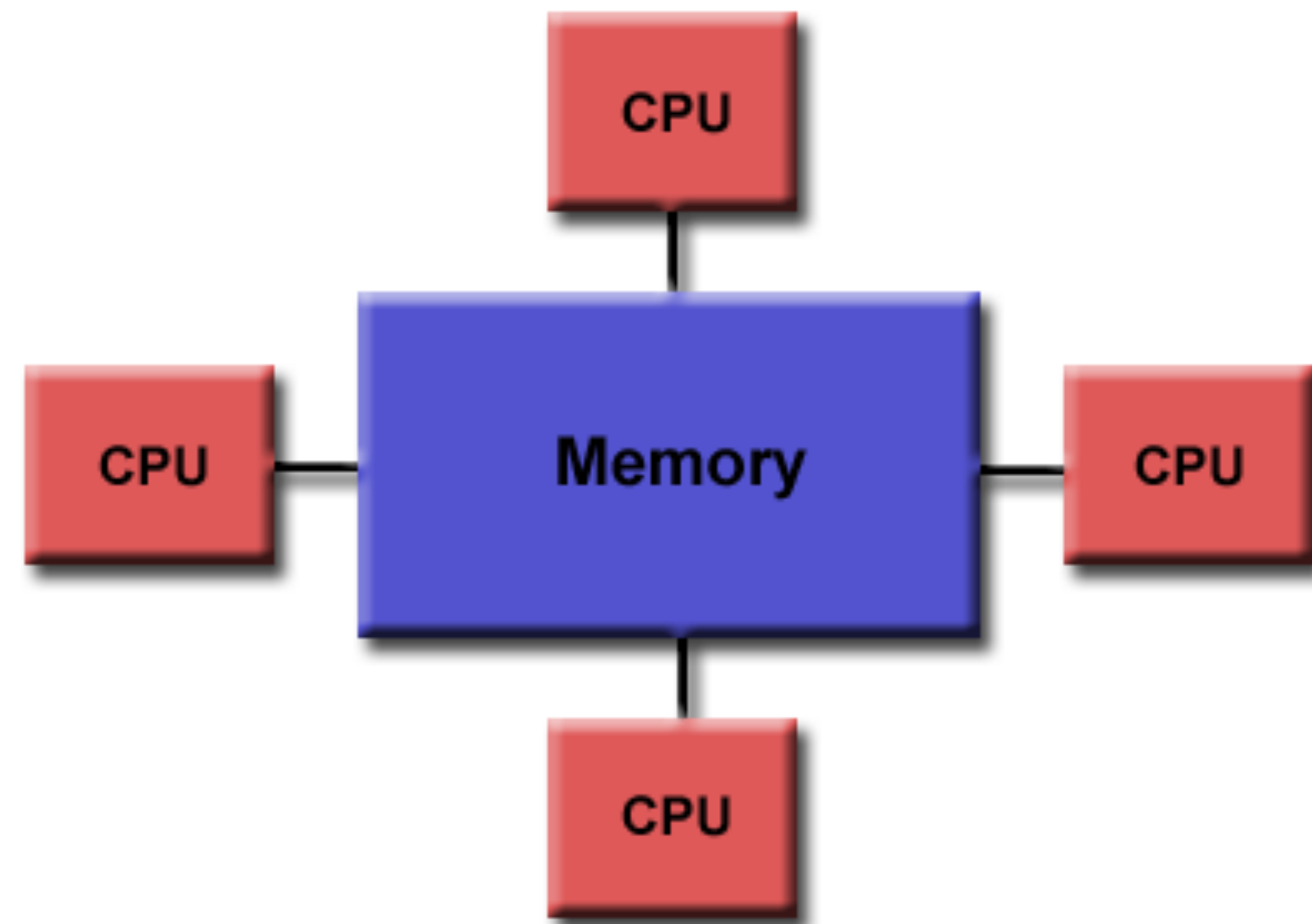


Uniform Memory Access

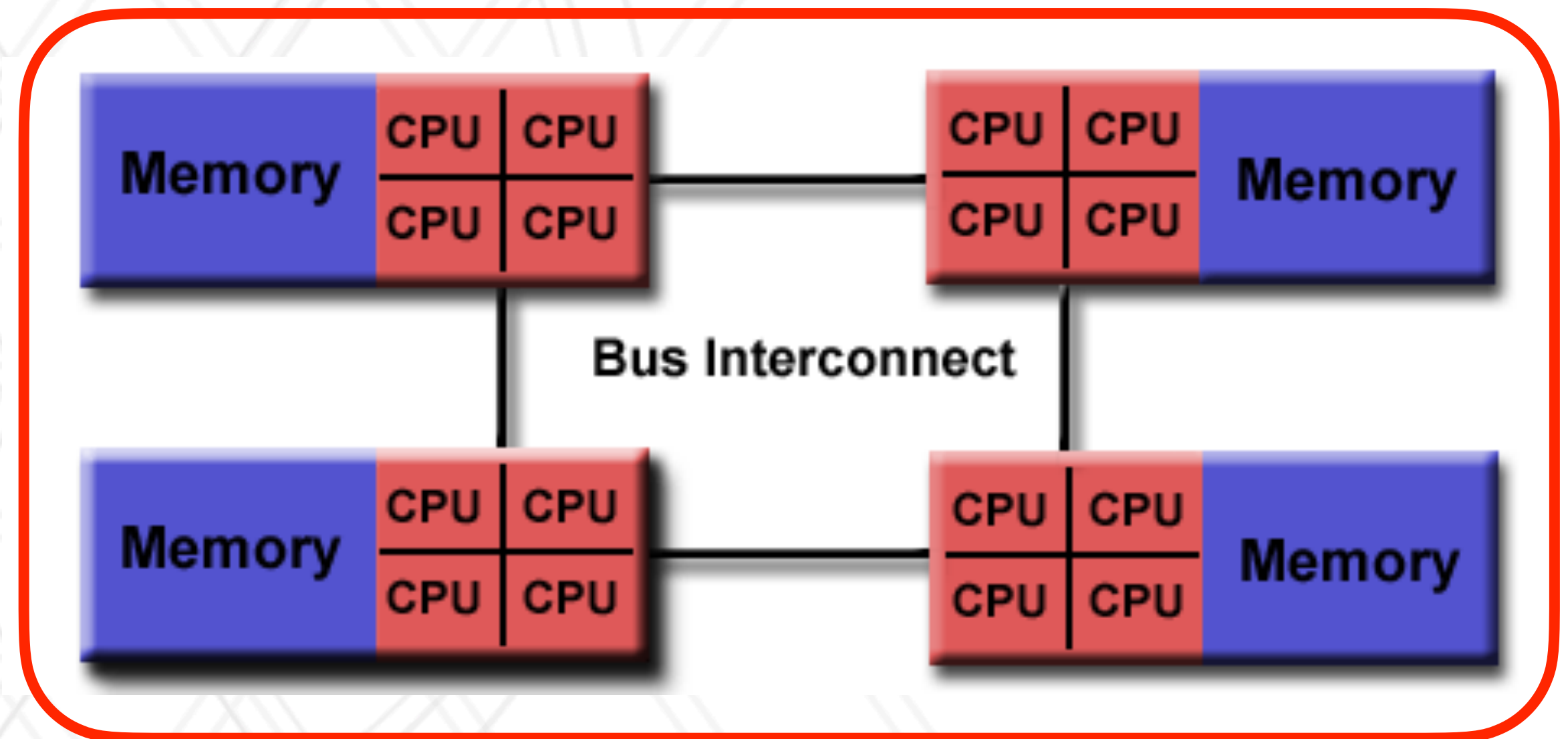
https://computing.llnl.gov/tutorials/parallel_comp/#SharedMemory

Shared memory architecture

- All processors/cores can access all memory as a single address space



Uniform Memory Access

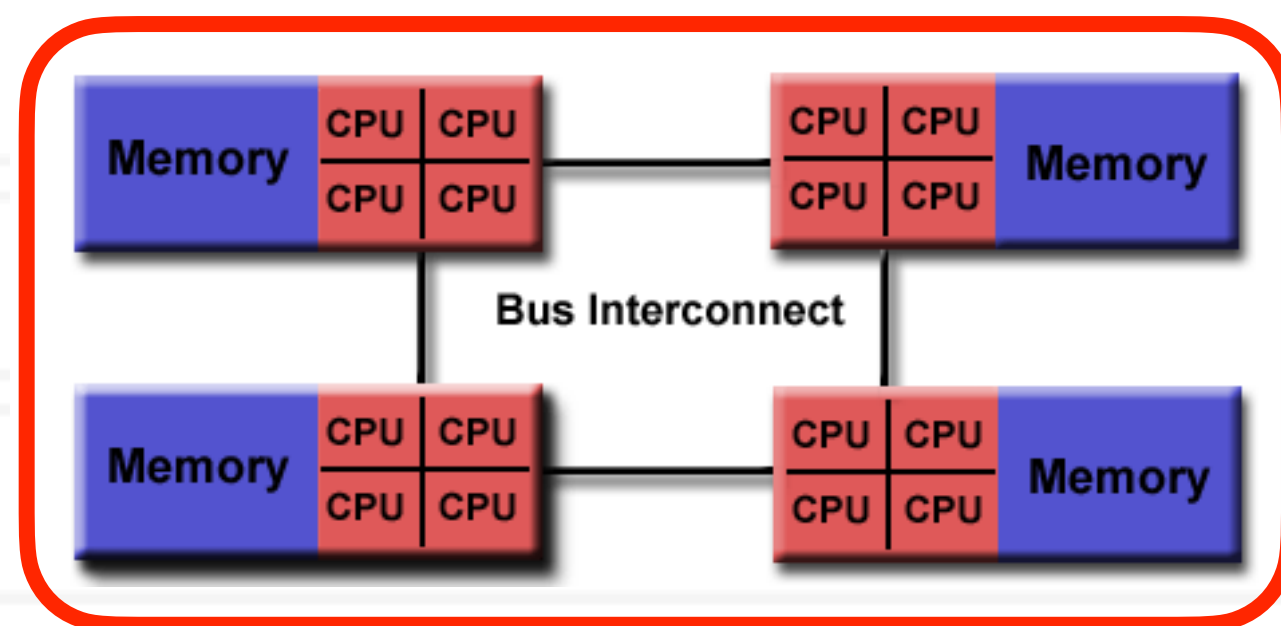


Non-uniform Memory Access (NUMA)

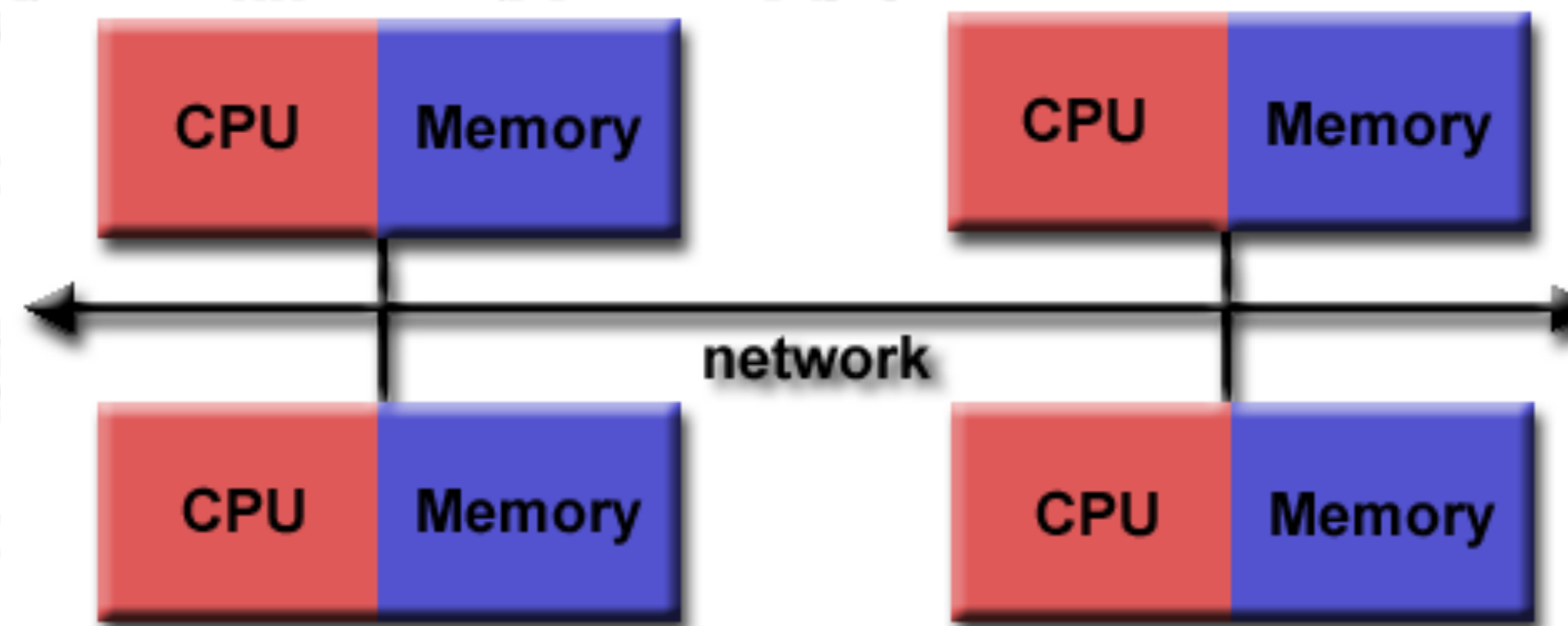
https://computing.llnl.gov/tutorials/parallel_comp/#SharedMemory

Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



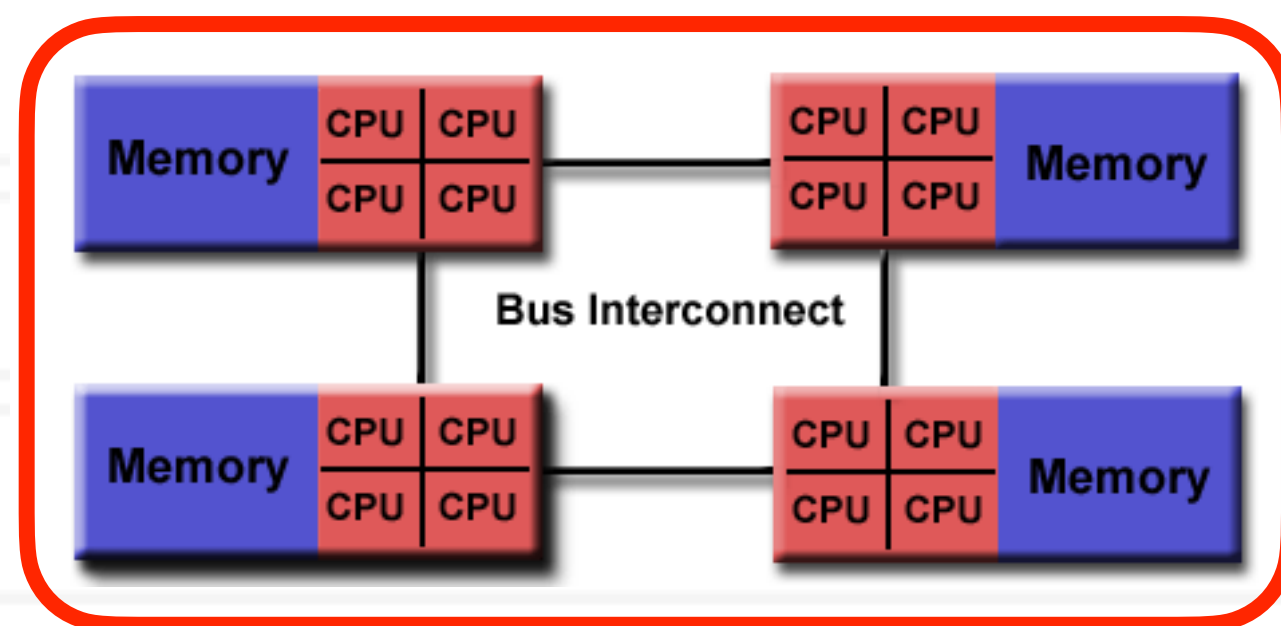
Shared memory (NUMA)



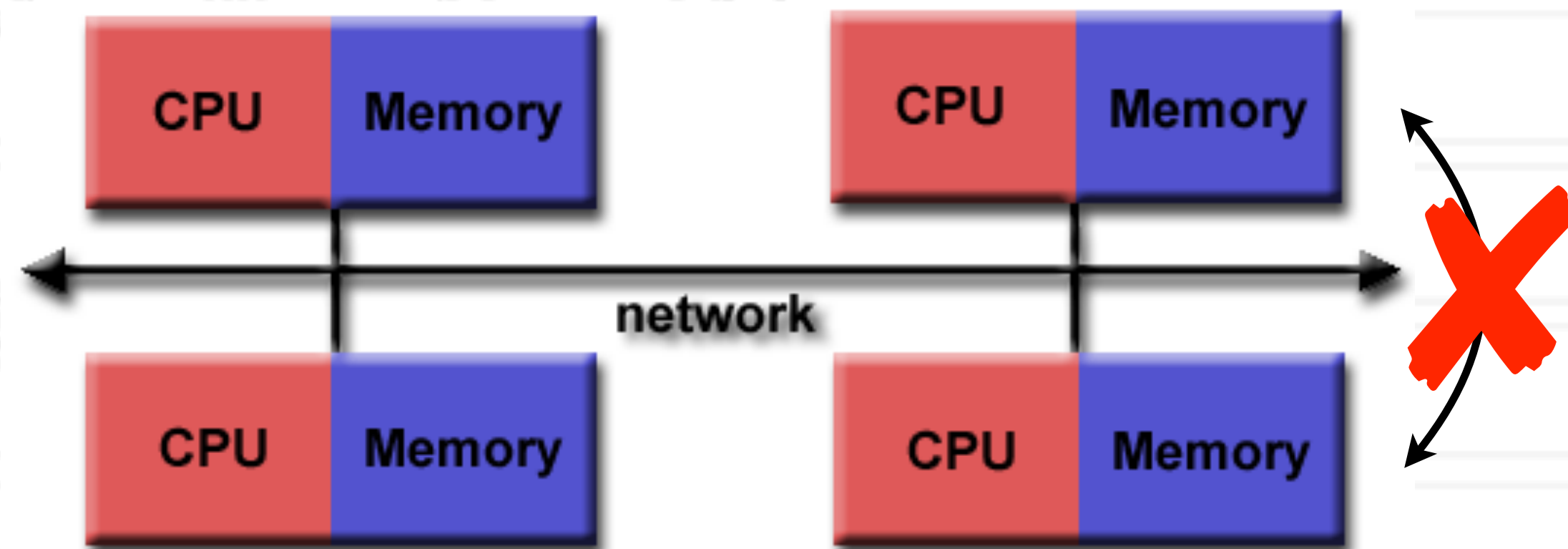
Distributed memory

Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



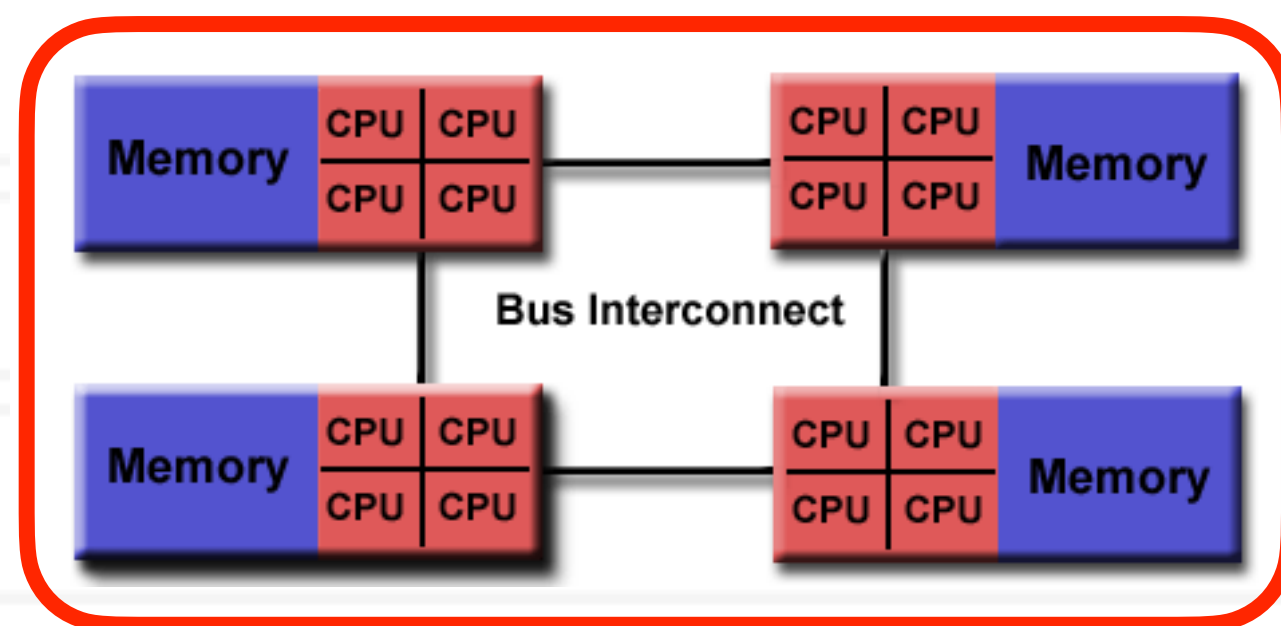
Shared memory (NUMA)



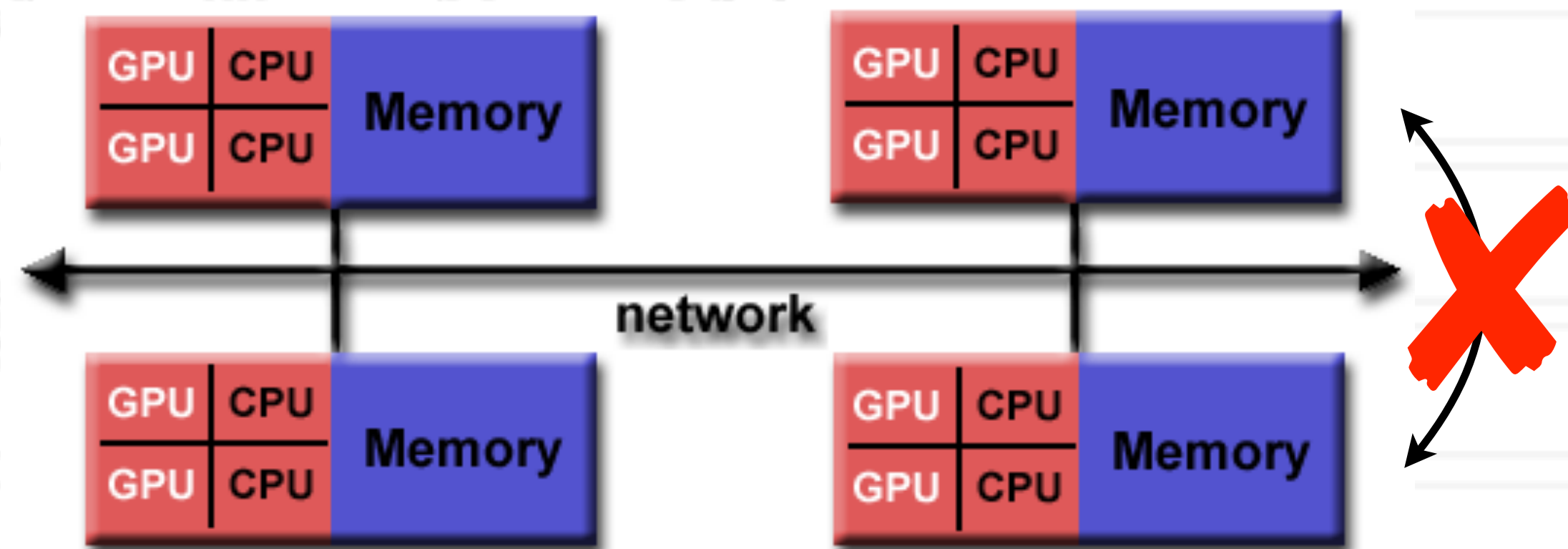
Distributed memory

Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



Shared memory (NUMA)



Distributed memory

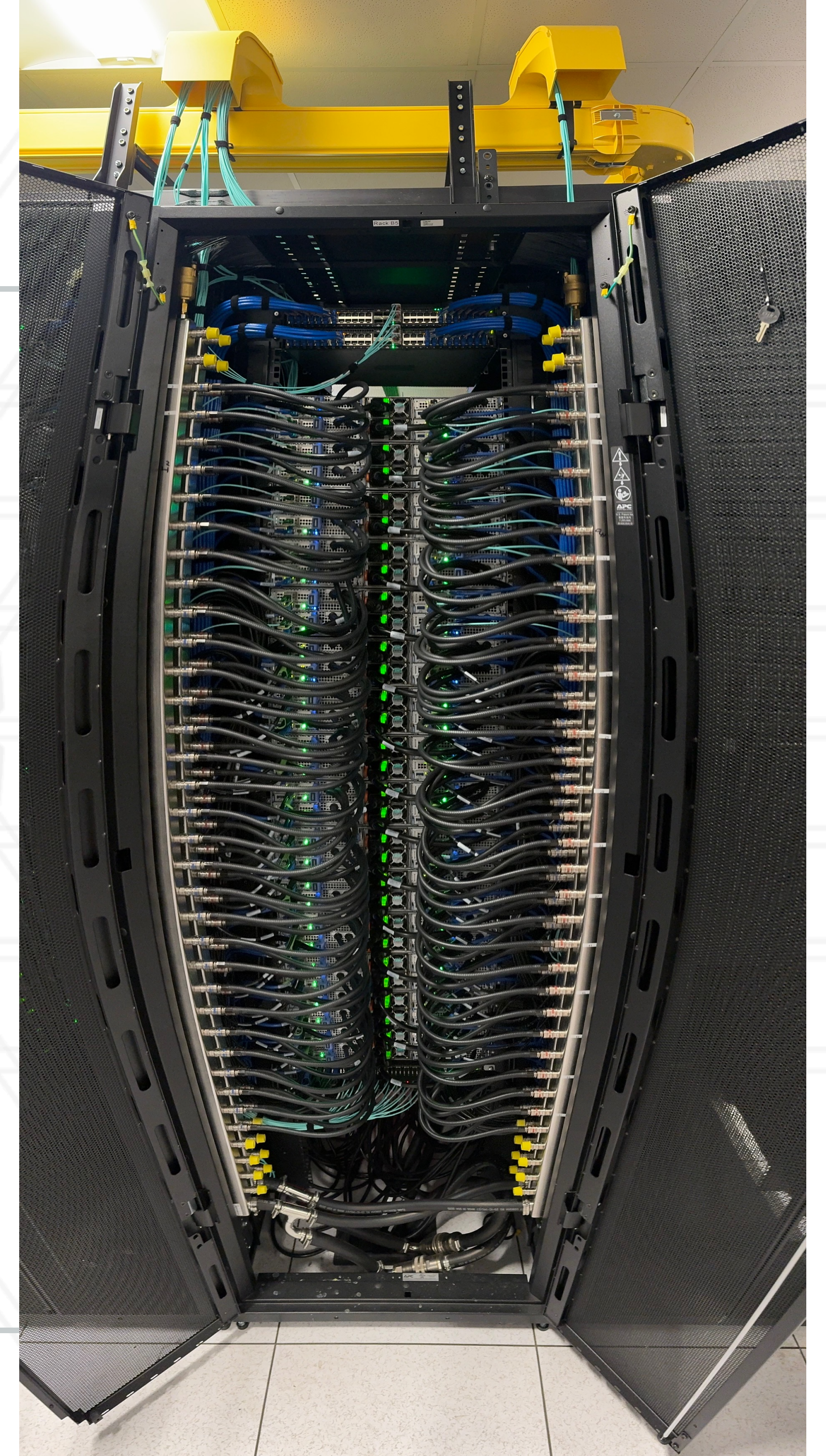
Rackmount servers



Rackmount servers



Zaratan racks / cabinets



Job scheduling



Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler

Job Queue

		#Nodes Requested	Time Requested
1		128	30 mins
2		64	24 hours
3		56	6 hours
4		192	12 hours
5		...	...
6		...	...

Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler
- The scheduler decides:
 - what job to schedule next (based on an algorithm: FCFS, priority-based,)
 - what resources (compute nodes) to allocate to the ready job

Job Queue

		#Nodes Requested	Time Requested
1		128	30 mins
2		64	24 hours
3		56	6 hours
4		192	12 hours
5		...	...
6		...	...

Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler
- The scheduler decides:
 - what job to schedule next (based on an algorithm: FCFS, priority-based,)
 - what resources (compute nodes) to allocate to the ready job

- Compute nodes: dedicated to each job
- Network, filesystem: shared by all jobs

Job Queue

		#Nodes Requested	Time Requested
1		128	30 mins
2		64	24 hours
3		56	6 hours
4		192	12 hours
5		...	...
6		...	...

Compute nodes vs. login nodes

- Compute nodes: dedicated nodes for running jobs
 - Can only be accessed when they have been allocated to a user by the job scheduler
- Login nodes: nodes shared by all users to compile their programs, submit jobs etc.
- Service/managements nodes: I/O nodes, etc.

Supercomputers vs. commodity clusters

- Supercomputer refers to a large expensive installation, typically using custom hardware
 - High-speed interconnect
 - IBM Blue Gene, Cray XT, Cray XC
- Cluster refers to a cluster of nodes, typically put together using commodity (off-the-shelf) hardware

Serial vs. parallel code

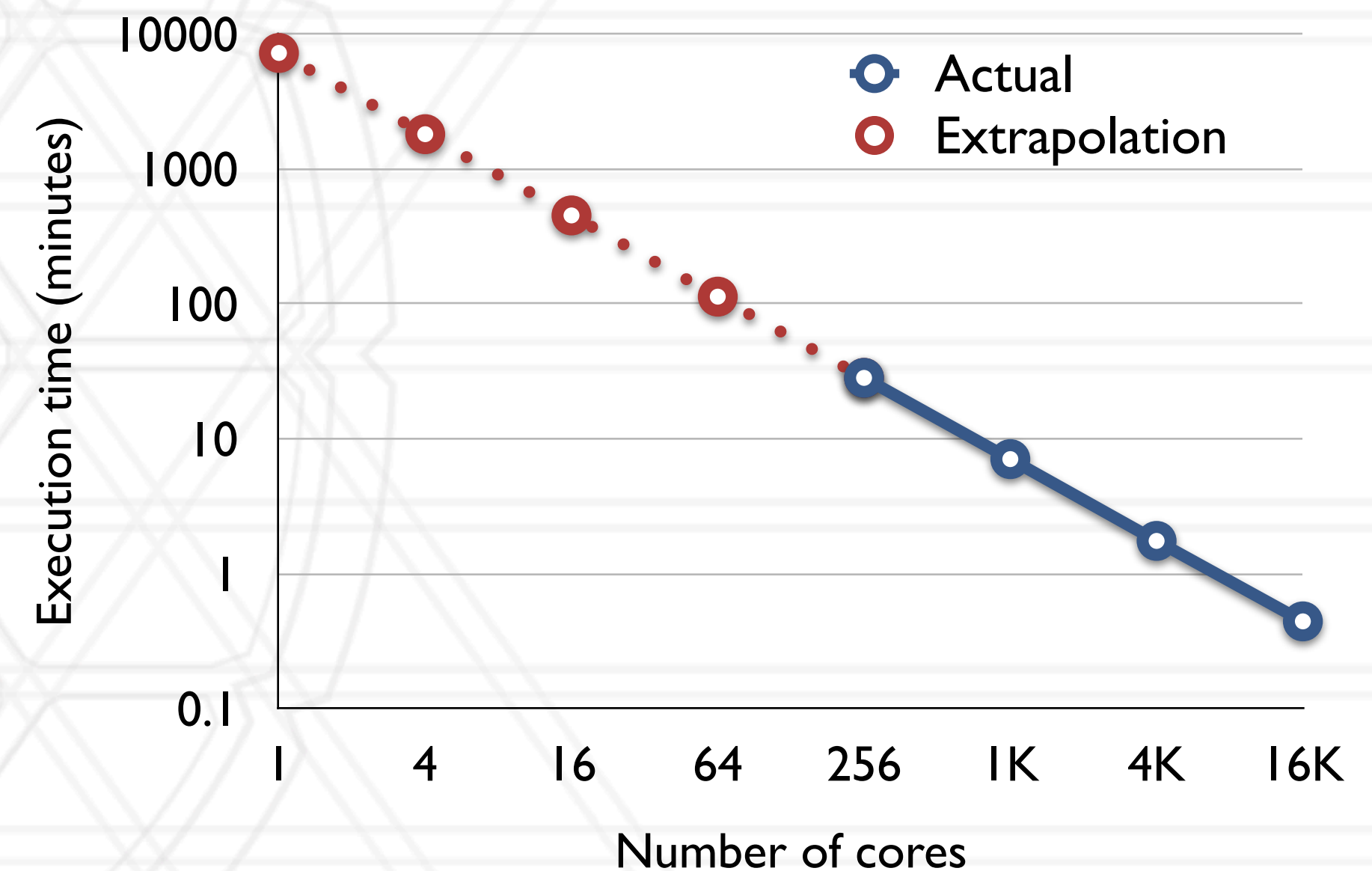
- Thread: a thread or path of execution managed by the operating system (OS)
 - Threads share the same memory address space
- Process: heavy-weight, processes do not share resources such as memory, file descriptors etc.
- Serial or sequential code: can only run on a single thread or process
- Parallel code: can be run on one or more threads or processes

Scaling and scalable

- Scaling: the action of running a parallel program on 1 to n processes
 - 1, 2, 3, ..., n
 - 1, 2, 4, 8, ..., n
- Scalable: A program is *scalable* if its performance improves when using more resources

Scaling and scalable

- Scaling: the action of running a parallel program on 1 to n processes
 - 1, 2, 3, ... , n
 - 1, 2, 4, 8, ..., n
- Scalable: A program is *scalable* if its performance improves when using more resources



Weak versus strong scaling

- Strong scaling: **Fixed total** problem size as we run on more processes
 - Sorting n numbers on 1 process, 2 processes, 4 processes, ...
 - Problem size per process decreases with increase in number of processes
- Weak scaling: **Fixed** problem size **per process** but *increasing total* problem size as we run on more processes
 - Sorting n numbers on 1 process
 - $2n$ numbers on 2 processes
 - $4n$ numbers on 4 processes

Speedup and efficiency

- (Parallel) Speedup: Ratio of execution time on one process to that on p processes

$$\text{Speedup} = \frac{t_1}{t_p}$$

- (Parallel) efficiency: Speedup per process

$$\text{Efficiency} = \frac{t_1}{t_p \times p}$$

Amdahl's law

- Speedup is limited by the serial portion of the code
 - Often referred to as the serial “bottleneck” — the portion that cannot be parallelized
- Lets say only a fraction f of the program (in terms of execution time) can be parallelized on p processes

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$

Amdahl's law

- Speedup is limited by the serial portion of the code
 - Often referred to as the serial “bottleneck” — the portion that cannot be parallelized
- Lets say only a fraction f of the program (in terms of execution time) can be parallelized on p processes

$$\text{Speedup} = \frac{1}{(1 - f) + \boxed{f/p}}$$

Amdahl's law

- Speedup is limited by the serial portion of the code
 - Often referred to as the serial “bottleneck” — the portion that cannot be parallelized
- Lets say only a fraction f of the program (in terms of execution time) can be parallelized on p processes

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$

Amdahl's law

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$

```
fprintf(stdout, "Process %d of %d is on %s\n",
    myid, numprocs, processor_name);
fflush(stdout);

n = 10000;          /* default # of rectangles */
if (myid == 0)
    startwtime = MPI_Wtime();
```

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs)
{
    x = h * ((double)i - 0.5);
    sum += f(x);
}
mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Total time on 1 process = 100s

Serial portion = 40s

Portion that can be parallelized = 60s

Amdahl's law

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$

```
fprintf(stdout, "Process %d of %d is on %s\n",
    myid, numprocs, processor_name);
fflush(stdout);

n = 10000;          /* default # of rectangles */
if (myid == 0)
    startwtime = MPI_Wtime();
```

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs)
{
    x = h * ((double)i - 0.5);
    sum += f(x);
}
mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Total time on 1 process = 100s

Serial portion = 40s

Portion that can be parallelized = 60s

$$f = \frac{60}{100} = 0.6$$

Amdahl's law

$$\text{Speedup} = \frac{1}{(1 - f) + f/p}$$

```
fprintf(stdout, "Process %d of %d is on %s\n",
    myid, numprocs, processor_name);
fflush(stdout);

n = 10000;          /* default # of rectangles */
if (myid == 0)
    startwtime = MPI_Wtime();
```

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs)
{
    x = h * ((double)i - 0.5);
    sum += f(x);
}
mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Total time on 1 process = 100s

Serial portion = 40s

Portion that can be parallelized = 60s

$$f = \frac{60}{100} = 0.6$$

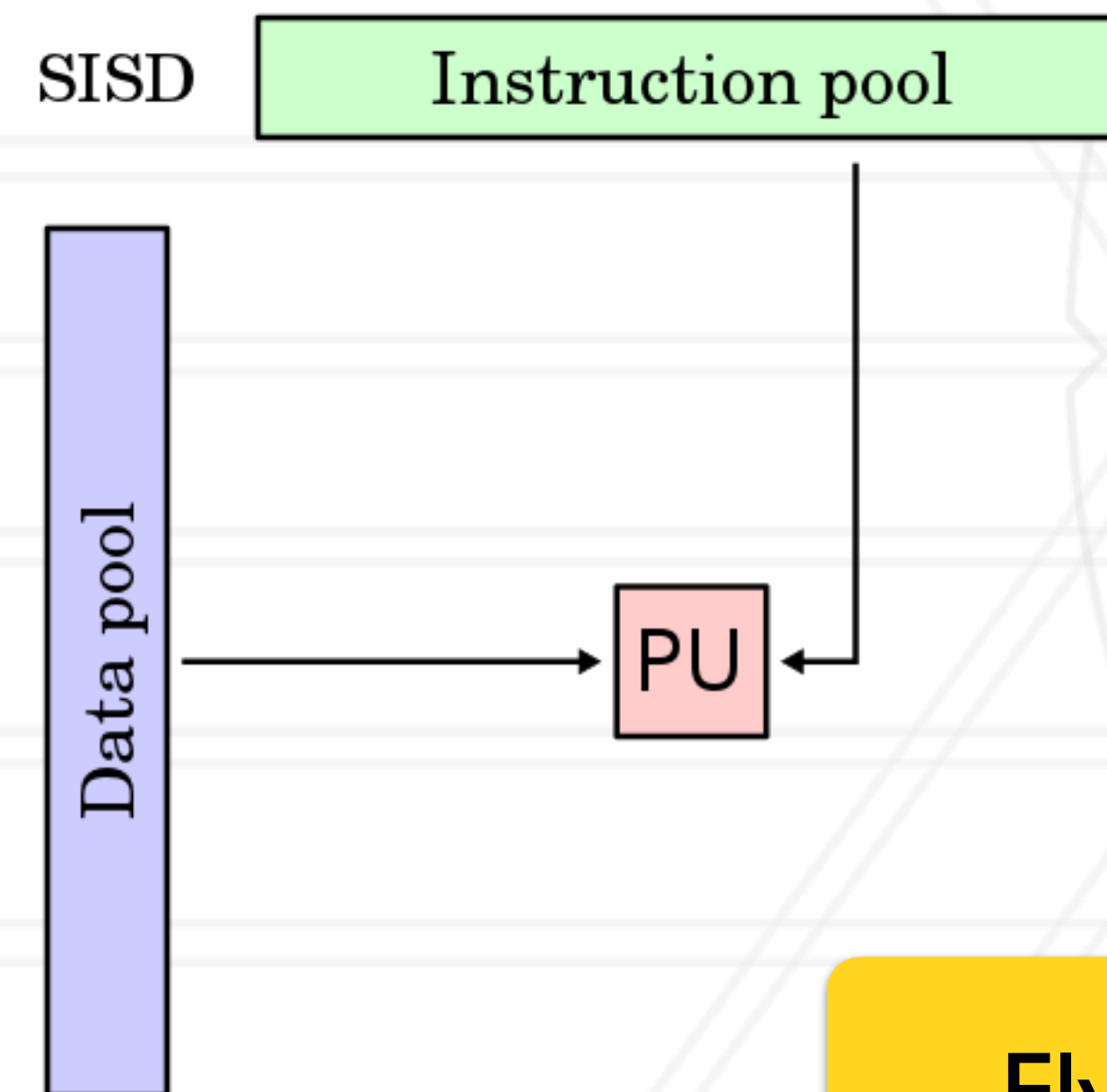
$$\text{Speedup} = \frac{1}{(1 - 0.6) + 0.6/p}$$

Communication and synchronization

- Each process may execute serial code independently for a while
- When data is needed from other (remote) processes, messaging is required
 - Referred to as communication or synchronization (or MPI messages)
- Intra-node communication: among cores within a node
- Inter-node communication: among cores on different nodes connected by a network
- Bulk synchronous programs: All processes compute simultaneously, then synchronize (communicate) together

Different models of parallel computation

SISD: Single Instruction Single Data



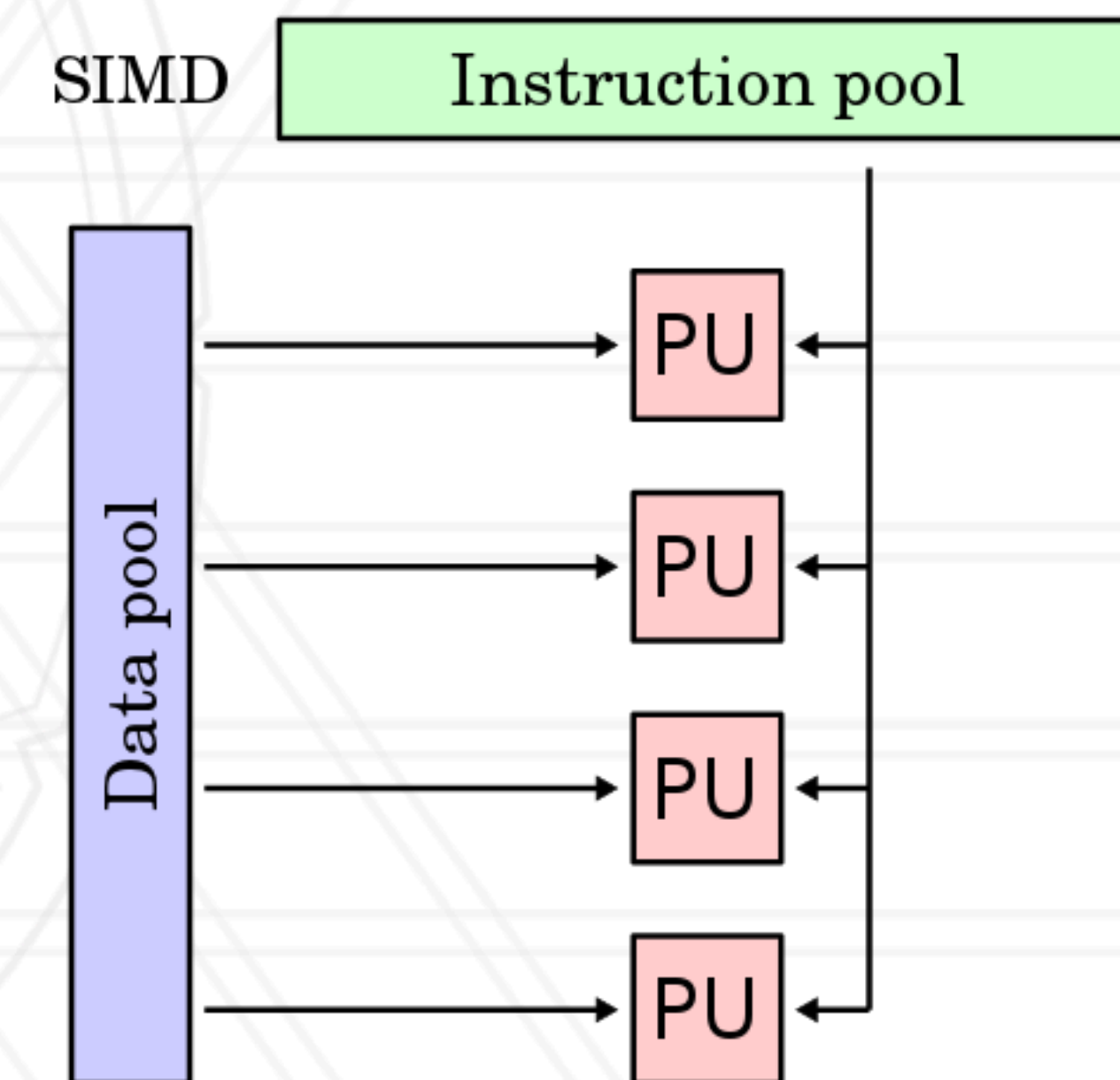
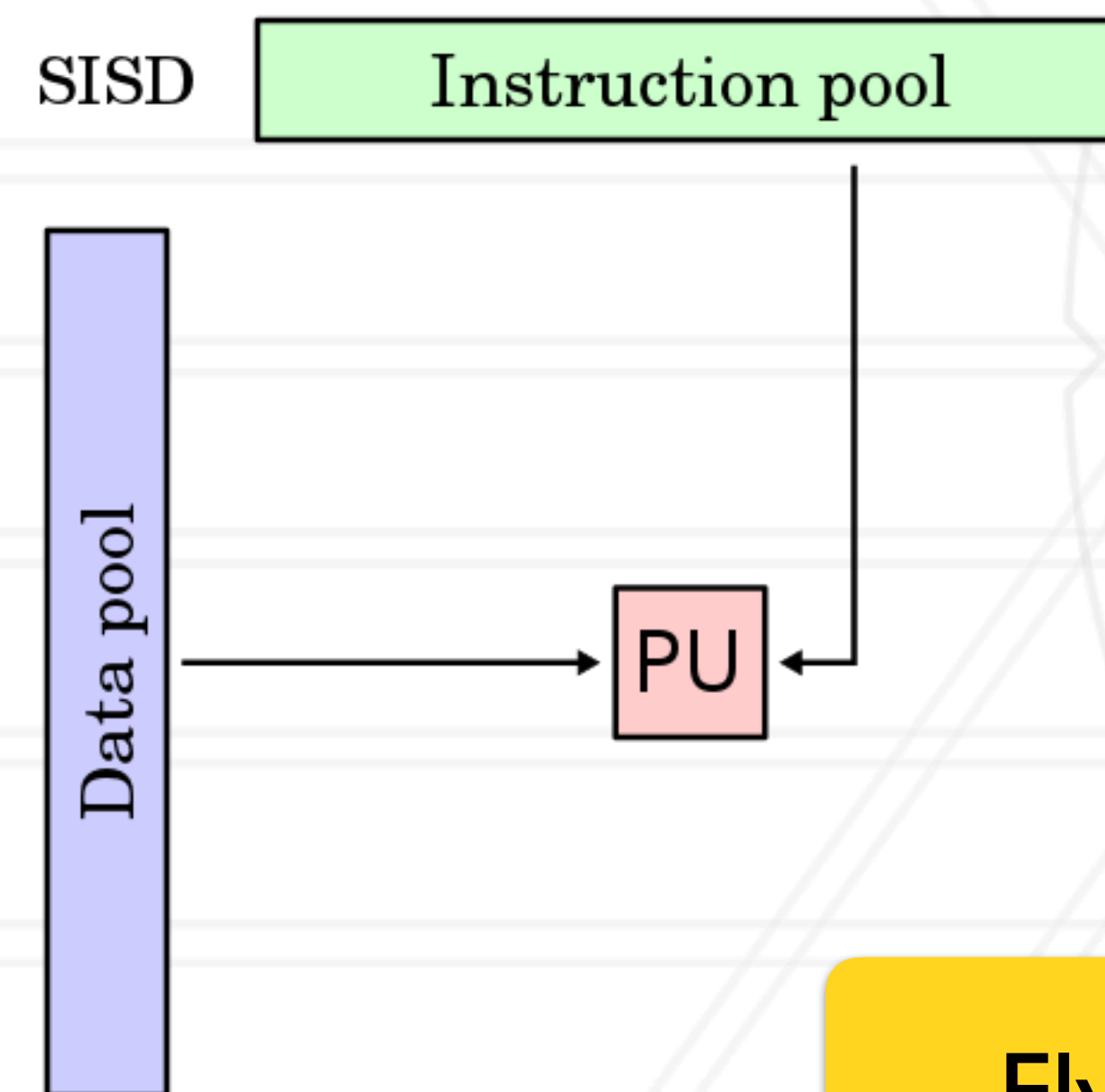
Flynn's Taxonomy

https://en.wikipedia.org/wiki/Flynn's_taxonomy

Different models of parallel computation

SISD: Single Instruction Single Data

SIMD: Single Instruction Multiple Data



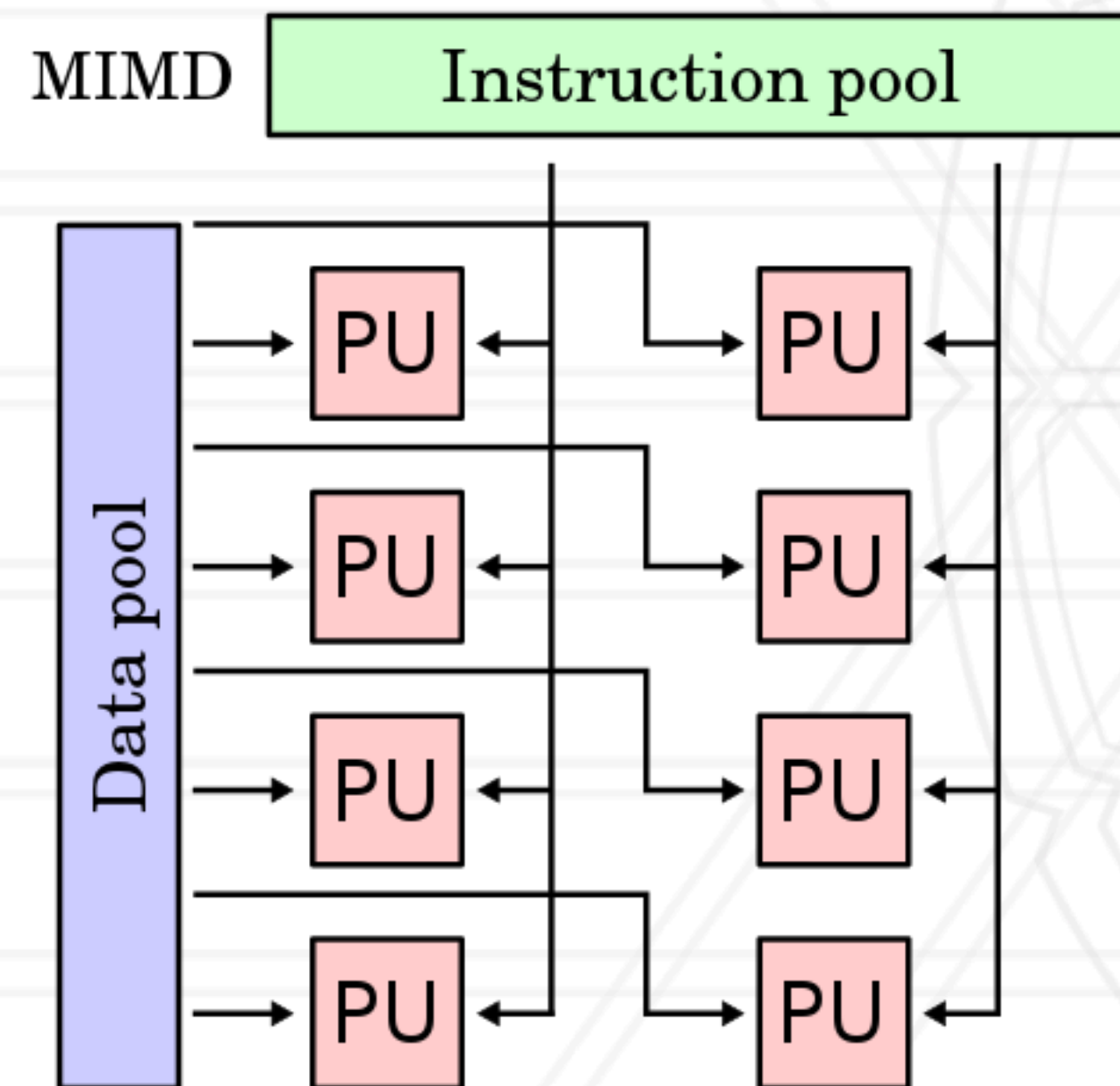
Flynn's Taxonomy

Example: Vector / array processors

https://en.wikipedia.org/wiki/Flynn's_taxonomy

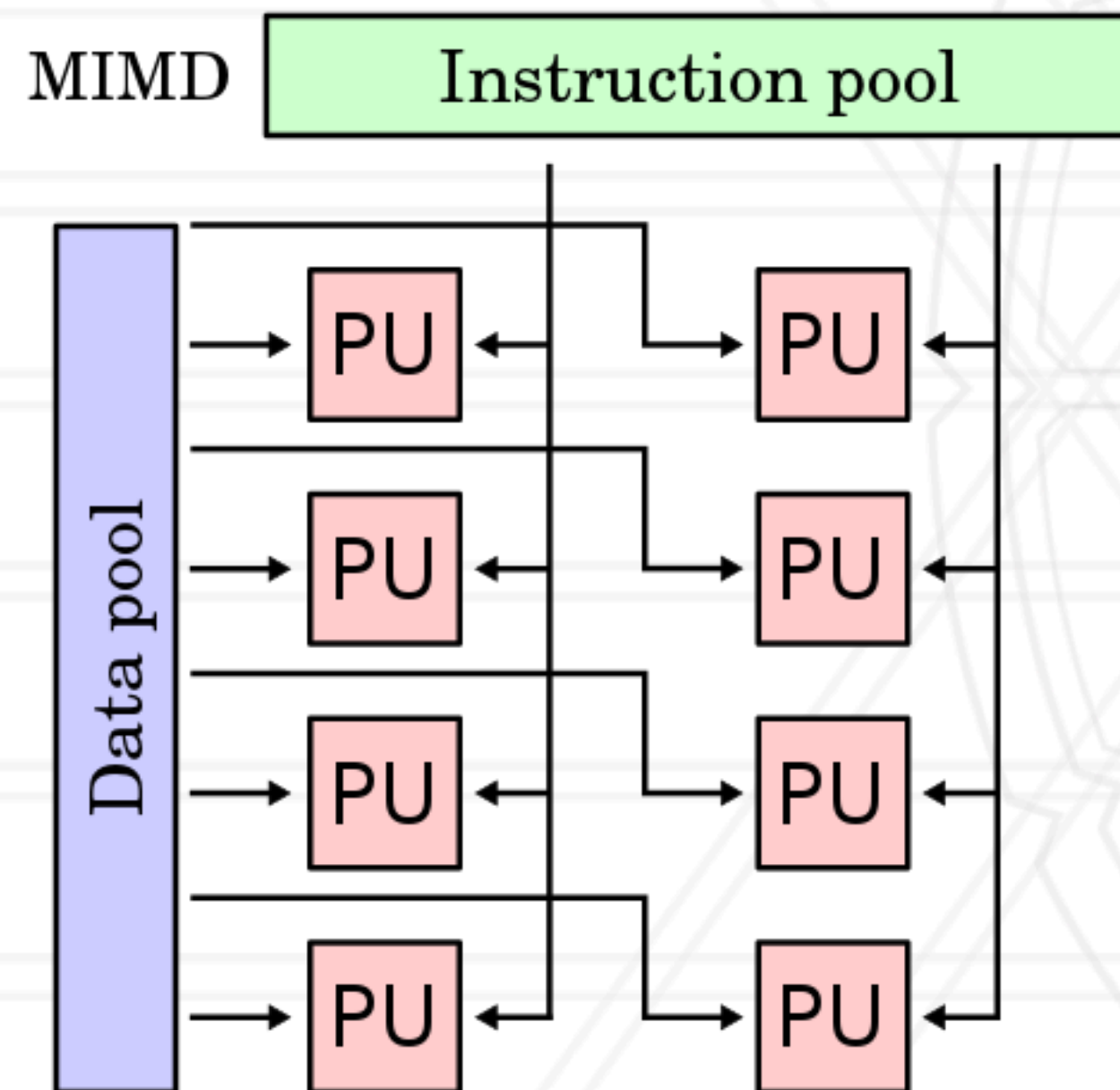
Different models of parallel computation

MIMD: Multiple Instruction Multiple Data



Different models of parallel computation

MIMD: Multiple Instruction Multiple Data



- Two other variations
- SIMT: Single Instruction Multiple Threads
 - Threads execute in lock-step
 - Example: GPUs
- SPMD: Single program Multiple Data
 - All processes execute the same program but act on different data
 - Enables MIMD parallelization

Parallel programming models

- Shared memory model: All threads have access to all of the memory
 - pthreads, OpenMP, CUDA
- Distributed memory model: Each process has access to its own local memory
 - Also sometimes referred to as message passing
 - MPI, Charm++
- Hybrid models: Use of both shared and distributed memory models together in the same program
 - MPI+OpenMP, Charm++ (SMP mode)



UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu