Parallel Computing Overview

What is a parallel computer?

Laptop
4 processing cores

Medium scale cluster
Habanero Cluster at Columbia:
295 nodes x 24 cores/node
= 7080 cores

Supercomputer
IBM Bluegreen/P
164,000 cores

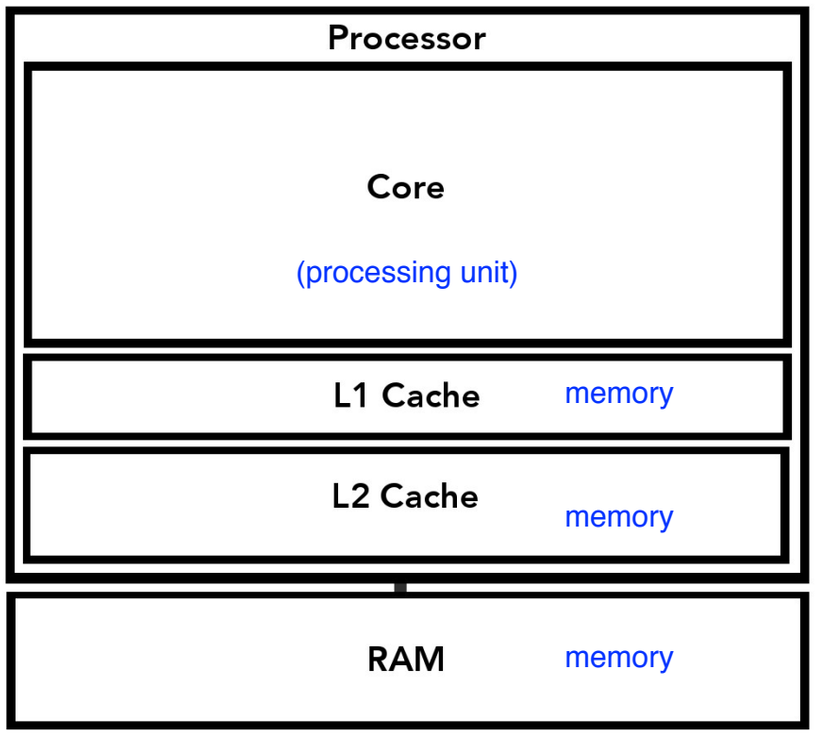
Graphics processing unit (GPU)

iPhone X
6 processing cores

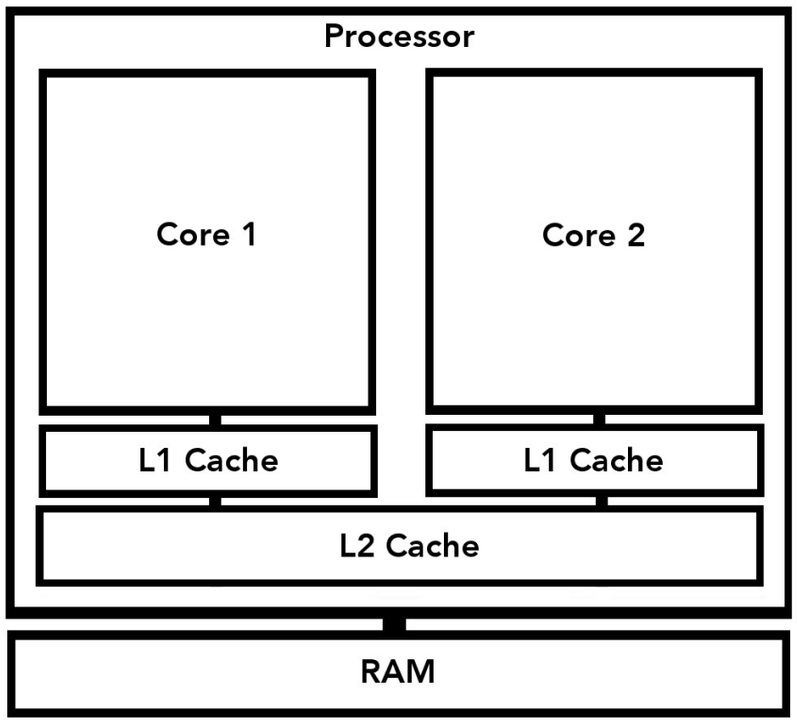
Samsung Galaxy S8
8 processing cores

Multicore Processors

Single core processor

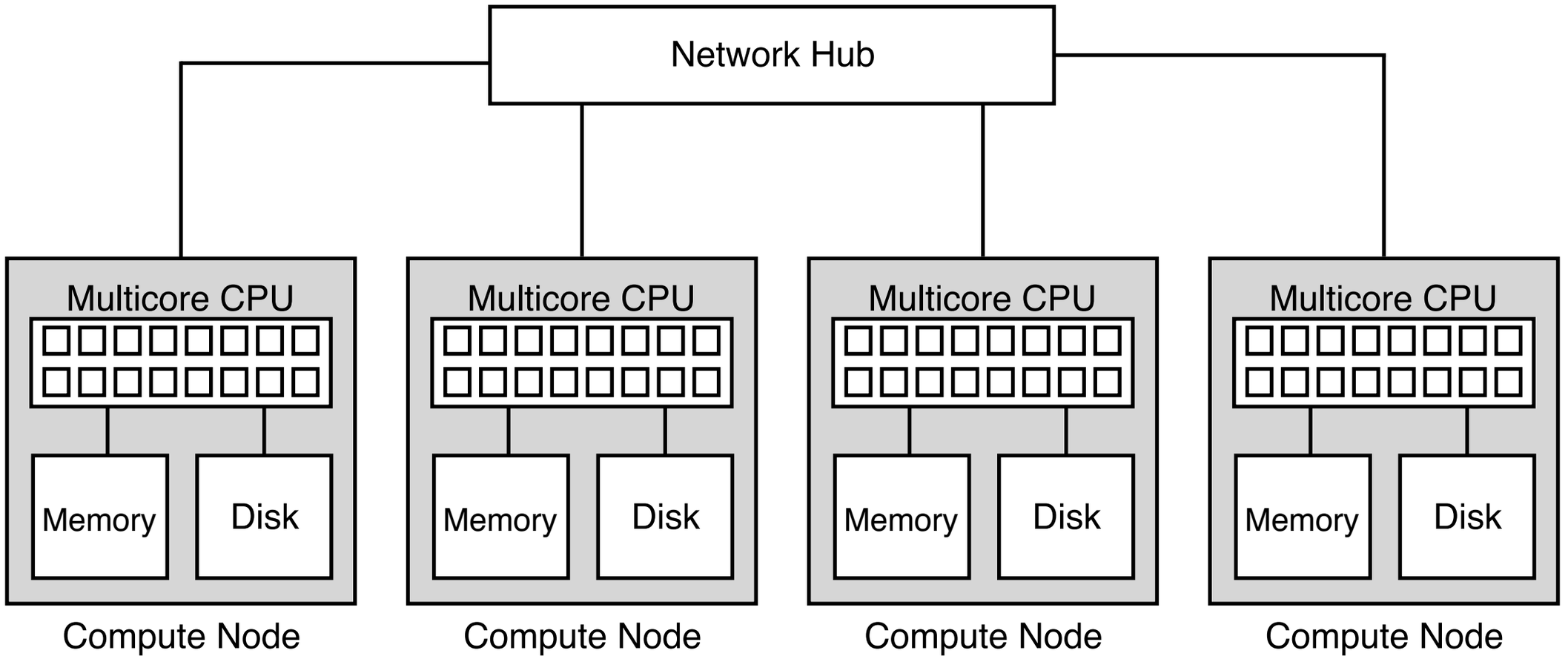


Multi-core processor



https://www.pcmech.com/article/all-about-multi-core-processors-what-they-are-how-they-work-and-where-they-came-from/

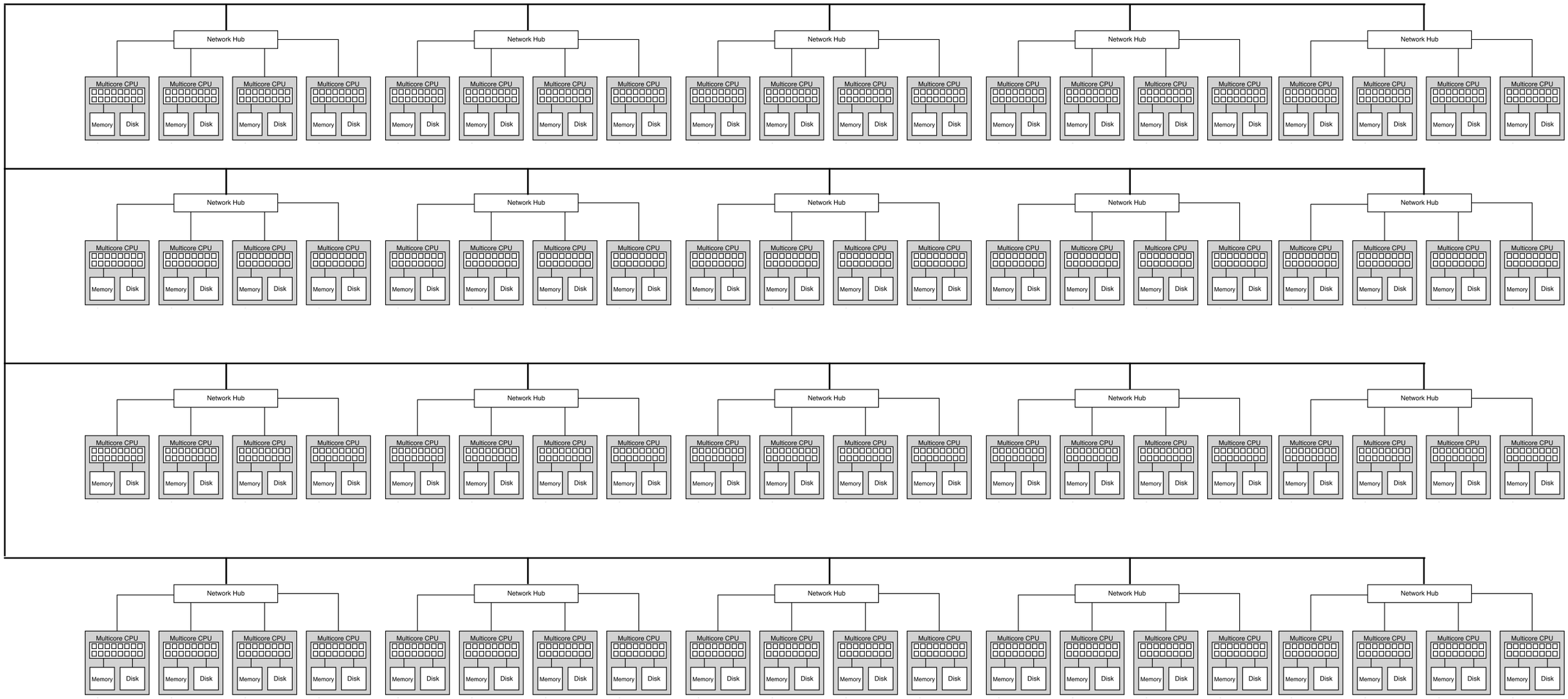
High performance computing:
Distributed multi-processor system (aka Cluster Computer)



- Current cluster compute nodes have two processors with 8–12 cores each
  Example: 10 node cluster with 24 cores per node = 240 processing cores

- Processor speed matters but so does the network speed
  - EDR InfiniBand network offers ~25 Gbit/s throughput

Large clusters have a similar layout that is repeated many times!



The Fastest High-Performance Computing Systems

Top500 List - November 2017

R_{max} and **R_{peak}** values are in TFlops. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

previous | 1 | 2 | 3 | 4 | 5 | next

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	361,760	19,590.0	25,326.3	2,272
4	Japan Agency for Marine-Earth Science and Technology Japan	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz ExaScaler	19,860,000	19,135.8	28,192.0	1,350
5	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
6	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
7	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	979,968	14,137.3	43,902.6	3,844

https://www.top500.org/list/2017/11/

Growth in computing power

https://www.top500.org/statistics/perfdevel/

Two main reasons to use HPC systems

1. Solve a fixed-size problem much faster than with a single processor

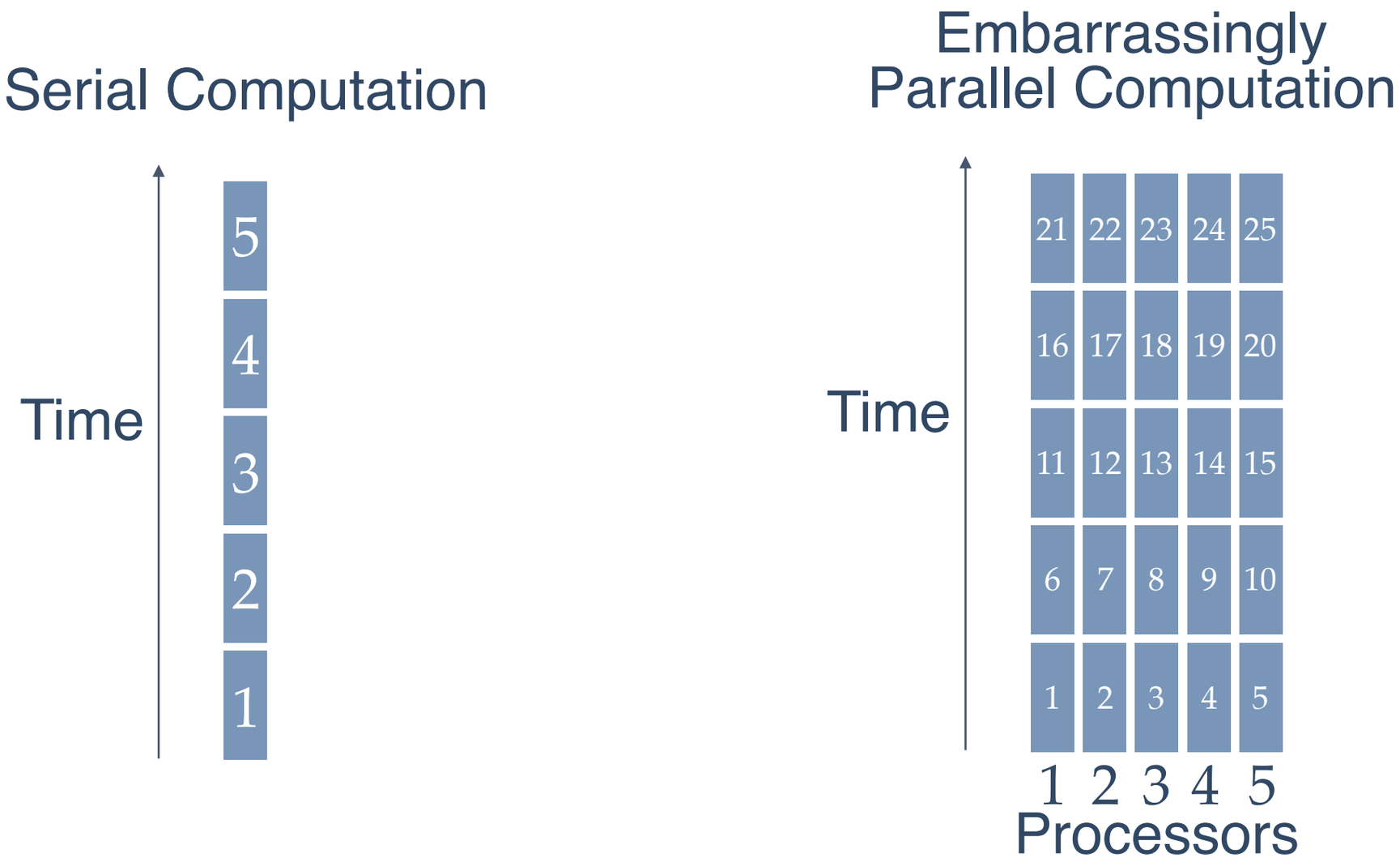
- Best case scenario: *n* times faster when run on *n* processors
  - Example: 365 days to run serial code vs 1 day to run parallel code on 365 processors

2. Solve a problem that requires more memory than possible on a single processor/node.

- Best case scenario (linear memory scaling): *n* times larger problem when run on *n* processors
- Example: Serial code can run models using 128GB ram on single CPU node. On 100 node system a parallel code has access to 12,800 GB ram.

Classification of Parallel Algorithms

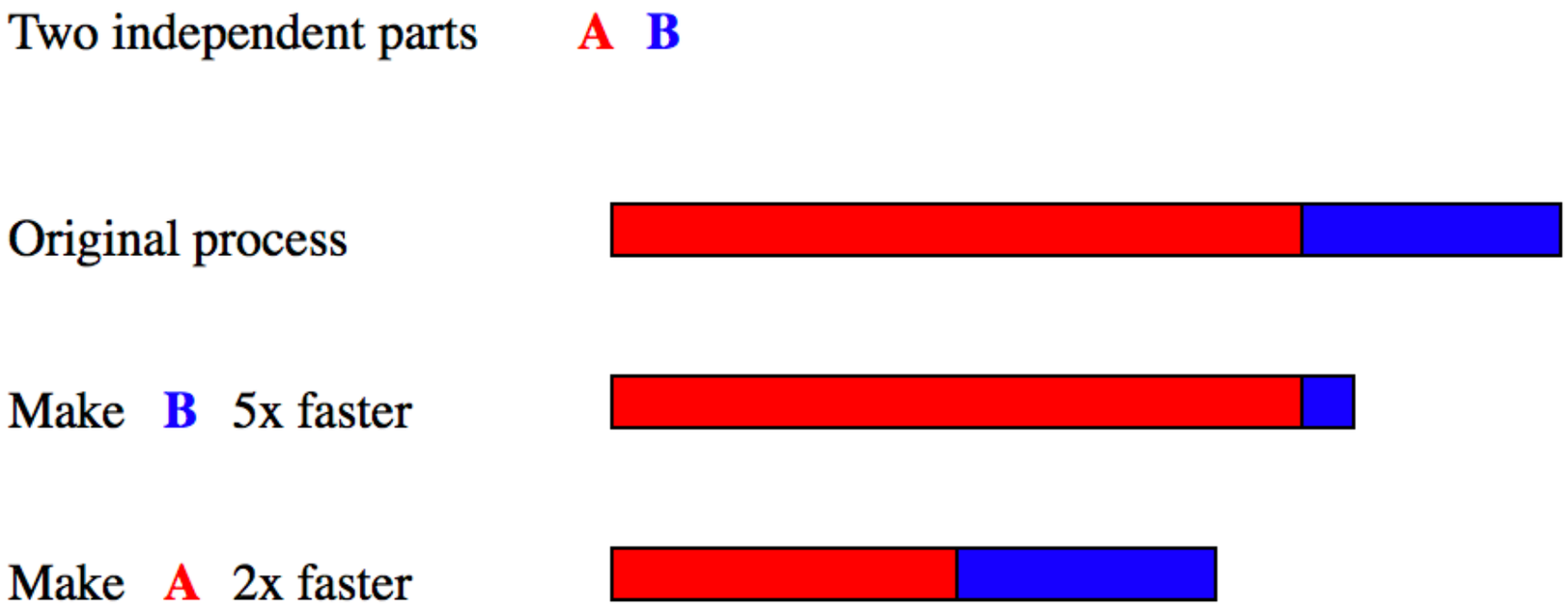
- **Embarrassingly parallel:**
  - requires little to no communication between processors
  - examples:
    - running same algorithm for a range of input parameters
    - rendering video frames in computer animation
    - proof-of-work systems used in cryptocurrency
- **Coarse-grained parallel:**
  - requires occasional communication between processors
- **Fine-grained parallel:**
  - requires frequent communication between processors
  - examples:
    - finite difference time-stepping on parallel grid
    - domain decomposition modeling for finite element method



- When run on *n* processors:
  - runs *n* times faster
  - or does *n* times as much work in same amount of time

When converting a serial code to parallel:

Focus on the slowest (longest) parts of your code



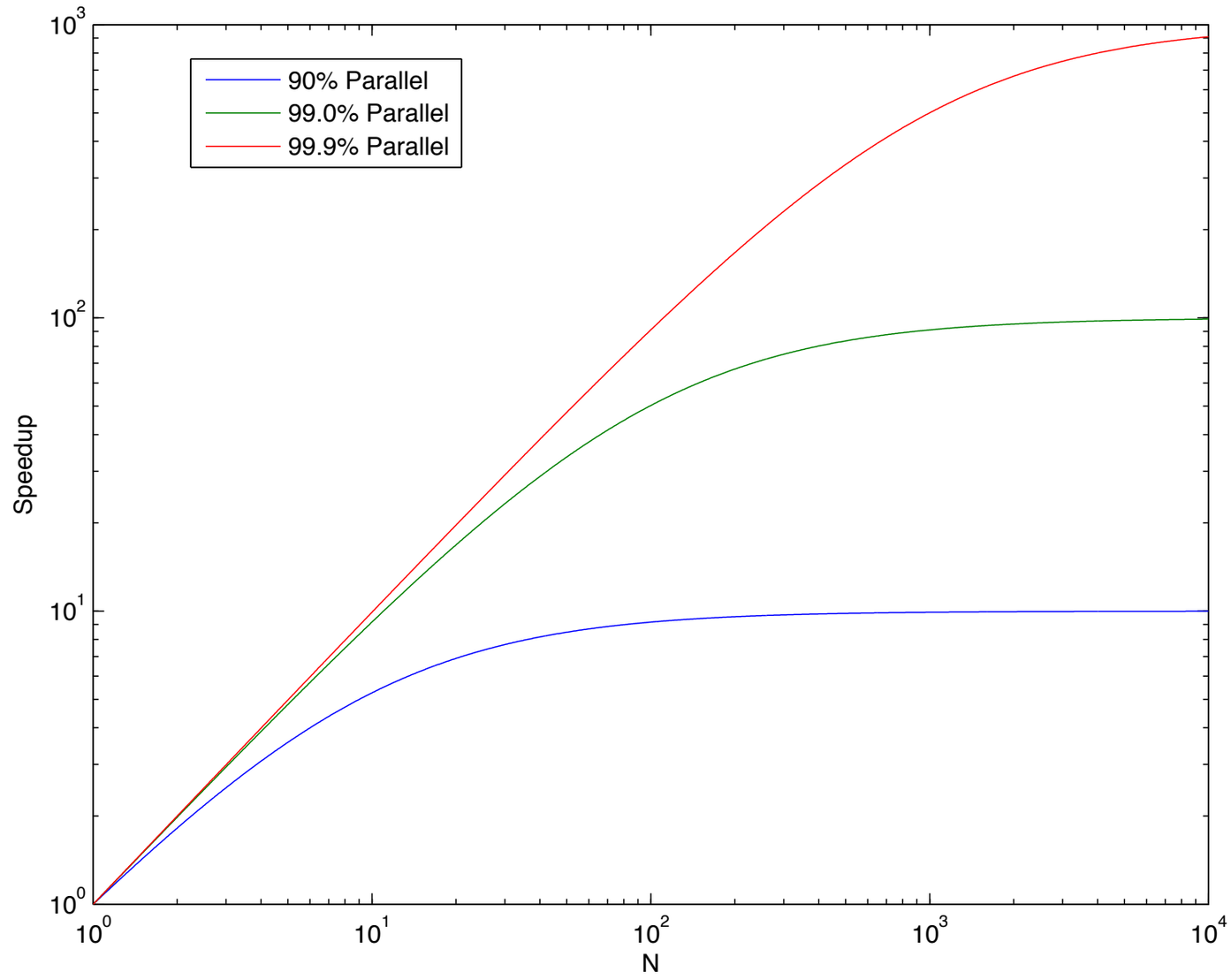
You might feel great about all your awesome coding to make B 5x faster, but speeding up A by only 2x had a bigger impact on the run time

https://en.wikipedia.org/wiki/Parallel_computing

Amdahl's Law

$$S_p \leq \frac{T_s}{T_p} = \frac{1}{f_s + \frac{f_p}{N}}$$

- Describes the potential speedup of a parallel program
- Speed of parallel program (S_p) depends on speed of parallel (T_p) and serial (T_s) portions of code
- f_s: serial fraction of the code
- f_p: parallel fraction of the code
- N: number of parallel processes



MPI: Message Passing Interface

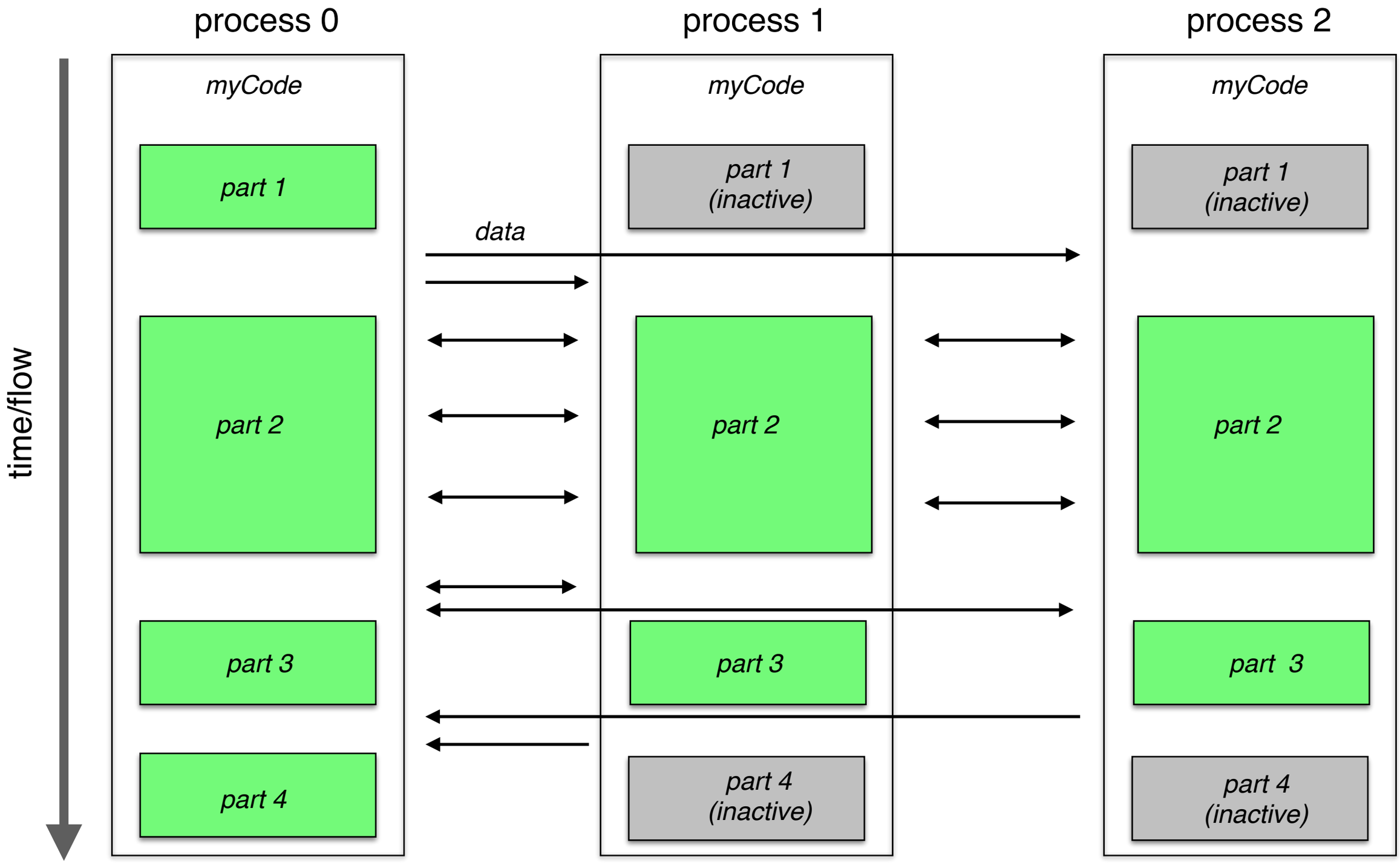
- **MPI** is a standardized and portable message-passing system designed to function on a wide variety of parallel computers. (http://mpi-forum.org)

- The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran.

- There are several well-tested and efficient implementations of MPI, many of which are open-source or in the public domain.
  - http://openmpi.org
  - http://mpich.org

- **mpi4py:** *MPI for Python* provides bindings of the MPI standard for the Python programming language, allowing any Python program to exploit multiple processors.

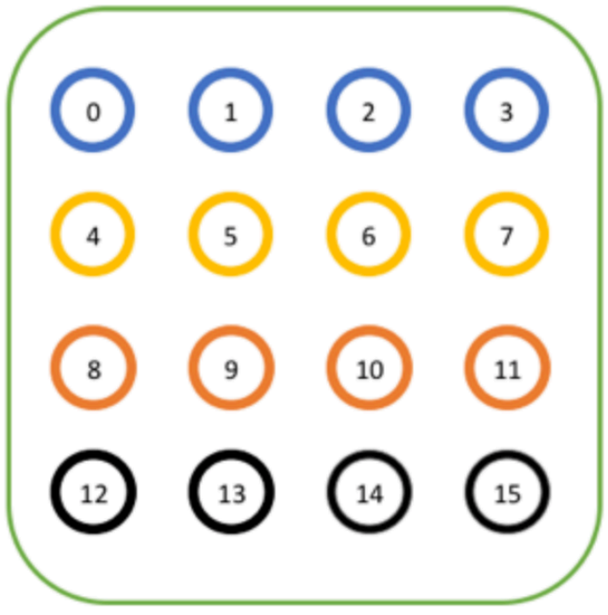
MPI program flow example



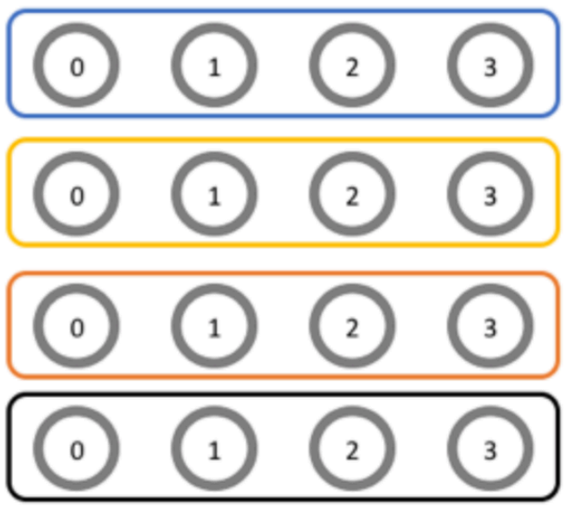
MPI: Communicators

- each parallel processor/core is known as a ***process***
- a group of ***processes*** is known as a ***communicator***
- within a ***communicator***, a process is known by its ***rank***
- ***communicators*** and ***ranks*** are assigned to integer variables
- a ***process*** can belong to more than one ***communicator***

Single ***communicator*** with 16 ***processes***



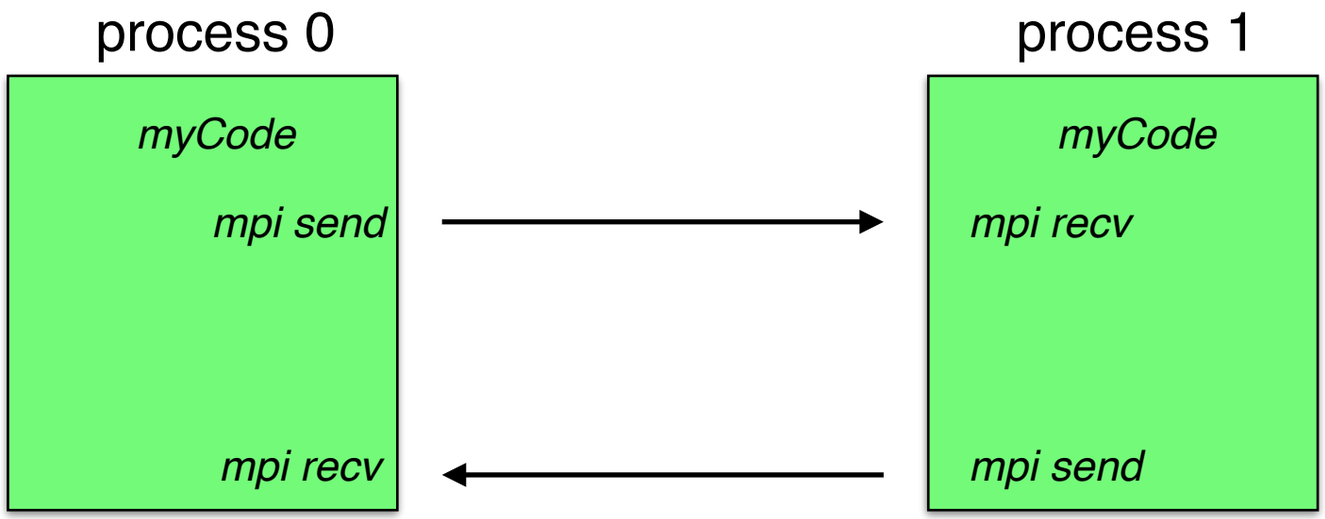
4 ***communicators*** with 4 ***processes*** each



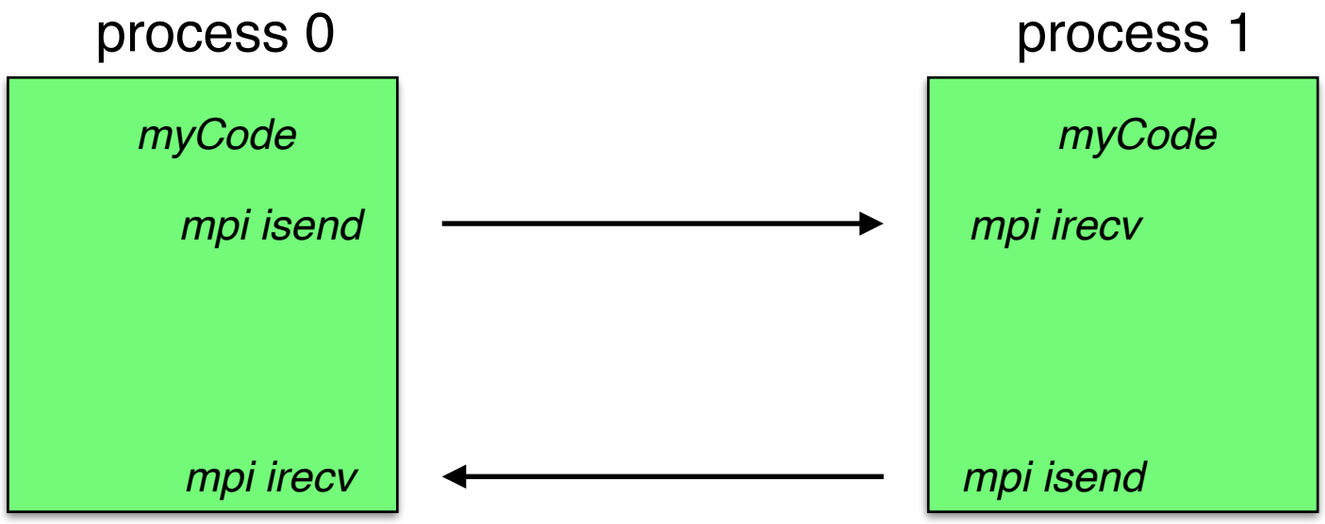
http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/

MPI: Point-to-point communications

Blocking (mpi **send** and **recv**):

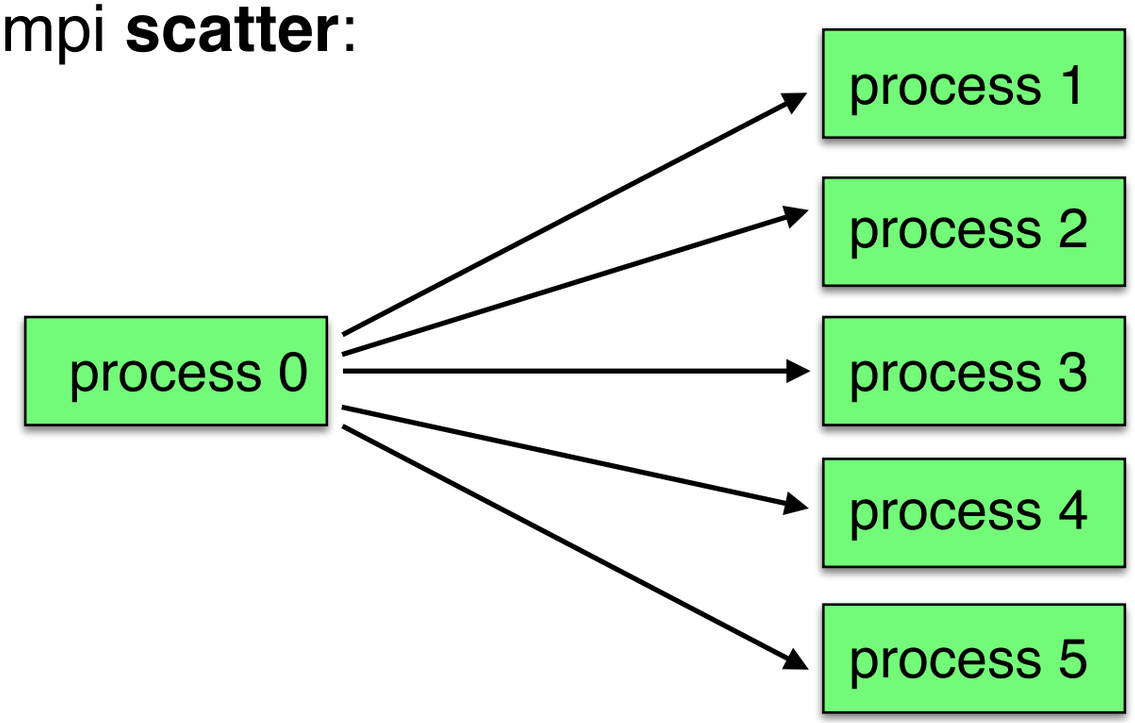


Non- Blocking (mpi **isend** and **irecv**):



MPI: Collective communications

mpi **scatter**:



mpi **gather**: