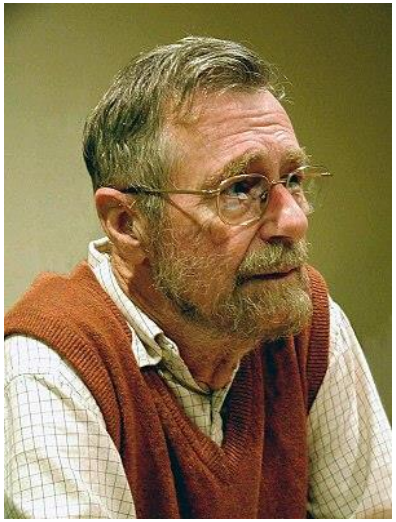
"as long as there were no machines, programming was no problem at all;

when we had a few weak computers, programming became a mild problem, and

now we have gigantic computers, programming has become an equally gigantic problem."



pic: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

- Edgar Dijkstra, 1972 Turing Award Lecture

A Gigantic Computer

- System 360 / Model 91

Source: https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2091.html

Cluster 101

NSM Nodal Center for Training in HPC and AI,
IIT Madras

Nikhil Hegde, IIT Dharwad

March 20, 2021

What is a Cluster ?

- Gigantic computer
  - from interconnecting several smaller computers

VIRGO Super Cluster, IIT Madras. Source: https://cc.iitm.ac.in/node/184

What is a Cluster ?

- Gigantic computer
  - from interconnecting several smaller computers
- Compute power in the order of 10^{15} floating point operations per second (Peta* FLOPS)
  - Your `i7`-based personal computer – few Giga FLOPs (10^9)

** 10^{12} is tera, 10^{18} is exa.*

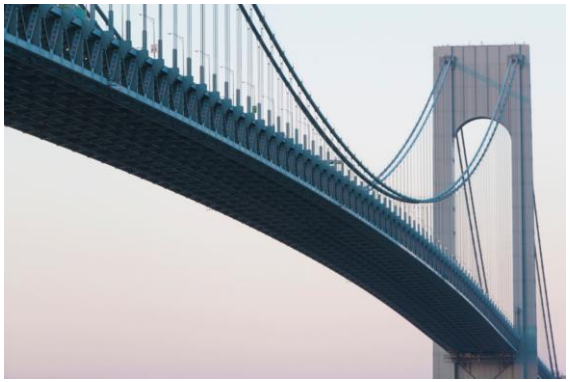
What is a Cluster ?

- Gigantic computer
  - from interconnecting several smaller computers
- Compute power in the order of 10^{15} floating point operations per second (Peta* FLOPS)
  - Your `i7`-based personal computer – few Giga FLOPs (10^9)
- E.g.
  - Chandra (IIT Palakkad), Virgo (IIT Madras), AnantGanak (IIT Dharwad) etc.

** 10^{12} is tera, 10^{18} is exa.*

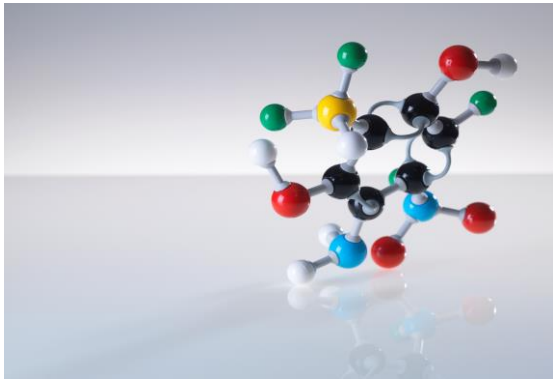
Why Clusters?

Financial Analysis



Design Simulation

Oil Exploration

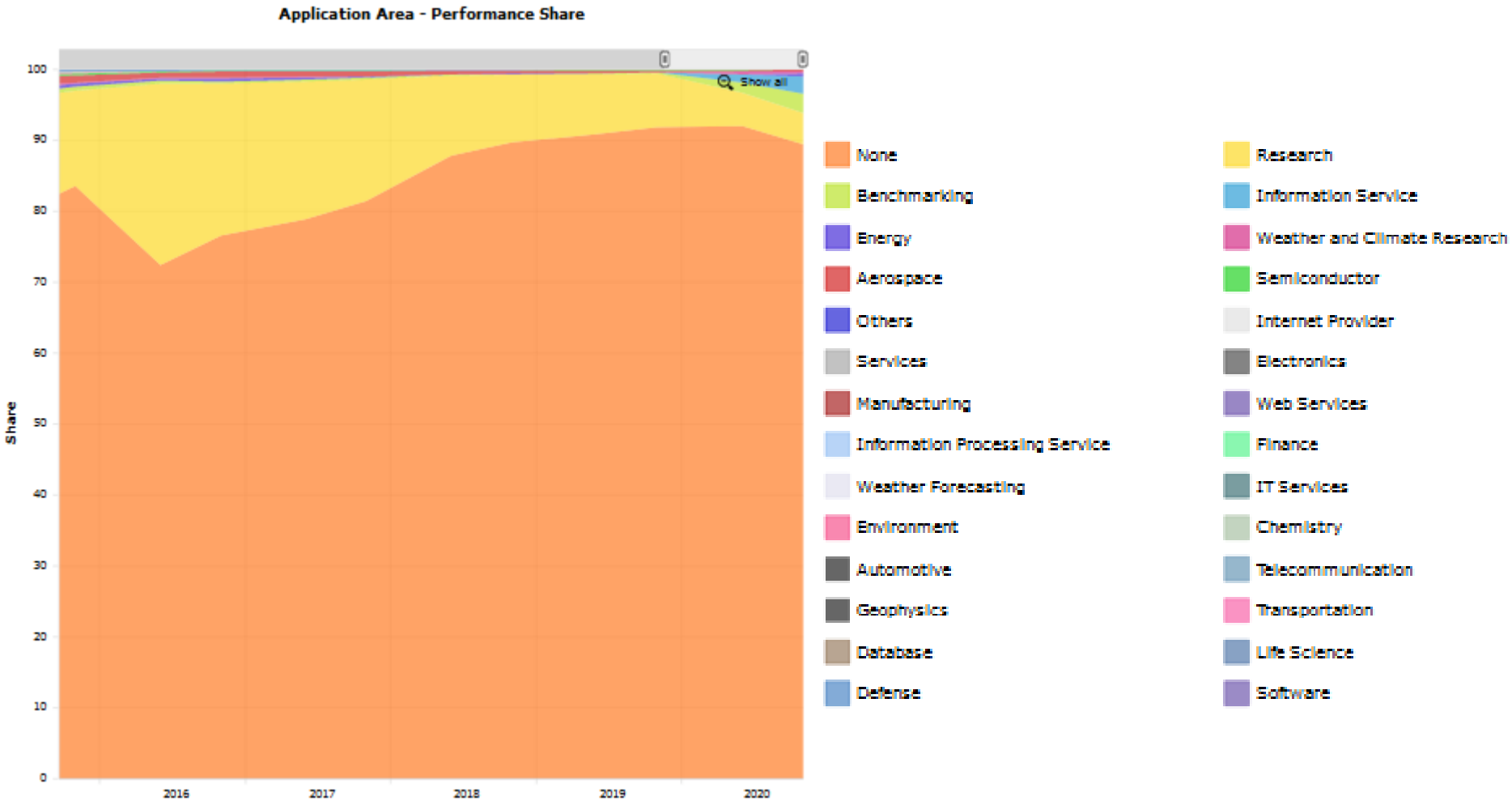


Genomics

Weather Prediction

pic source: stock images

Why Clusters? Application Areas



source: top500.org

Terminology - Cluster Elements

- Processing Element
  - Core, CPU, Node, GPU, Virtual CPUs
- Storage
- Interconnect

Hardware

- Partition
- Job and Job Scheduler
- Operating System (OS), Software Development Tools, Applications

Software

- *Infrastructure* – power, cooling,

Processing Element

- Node
  - Standalone computer
  - Comprised of multiple CPUs/Processors/Cores, memory, network interfaces.

Each Green dot is a Node

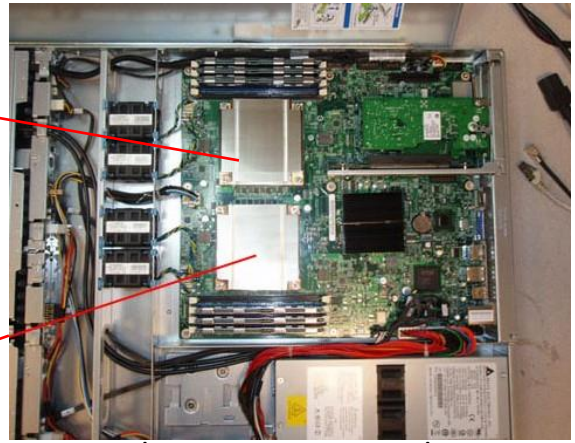
  - `Master / Log-in` node: is what end user interacts with (think: operator's console)

Processing Element

- CPU/Processor and Socket
  - No consensus on terminology. Some vendors call multi-core CPUs as sockets.

CPU

CPU

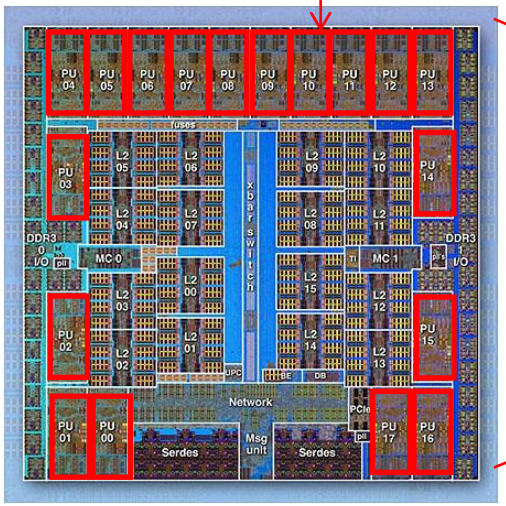


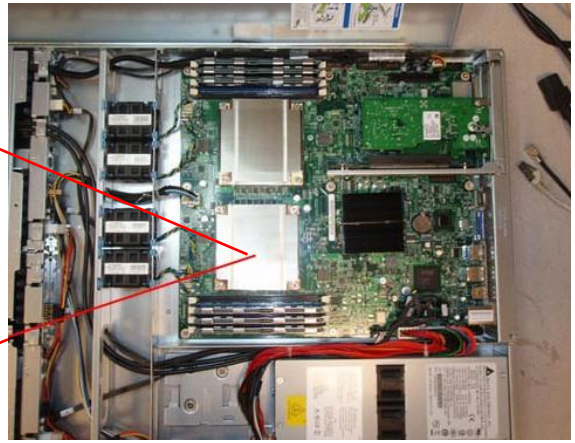
source: Blaise Barney, Introduction to Parallel Computing, https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

  - Socket can also be a place to plug a CPU. E.g. dual-socket motherboard in pic.

Processing Element

- Core
  - Each PU shown is a core. Pic: IBM BG/Q with 18 Cores



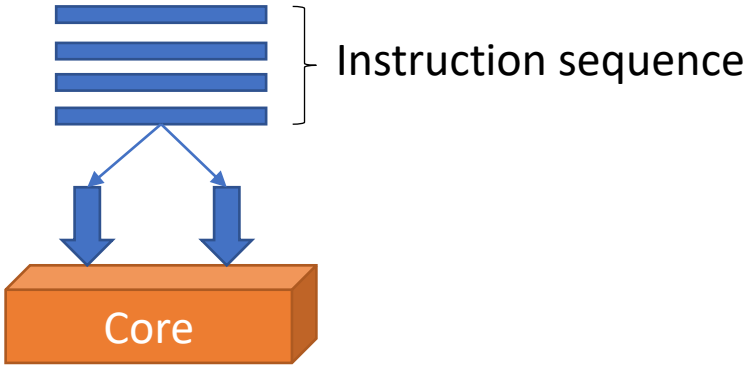


source: Blaise Barney, Introduction to Parallel Computing, https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

  - In the past, each CPU (with just one core) was a single execution unit. Now, each core is an independent execution unit.

Processing Element

- Thread (hardware)
  - Pathway for flow of instruction within a core
  - When exposed to the OS, the OS gives an illusion to the programmer that multiple cores exist ("HyperThreading")

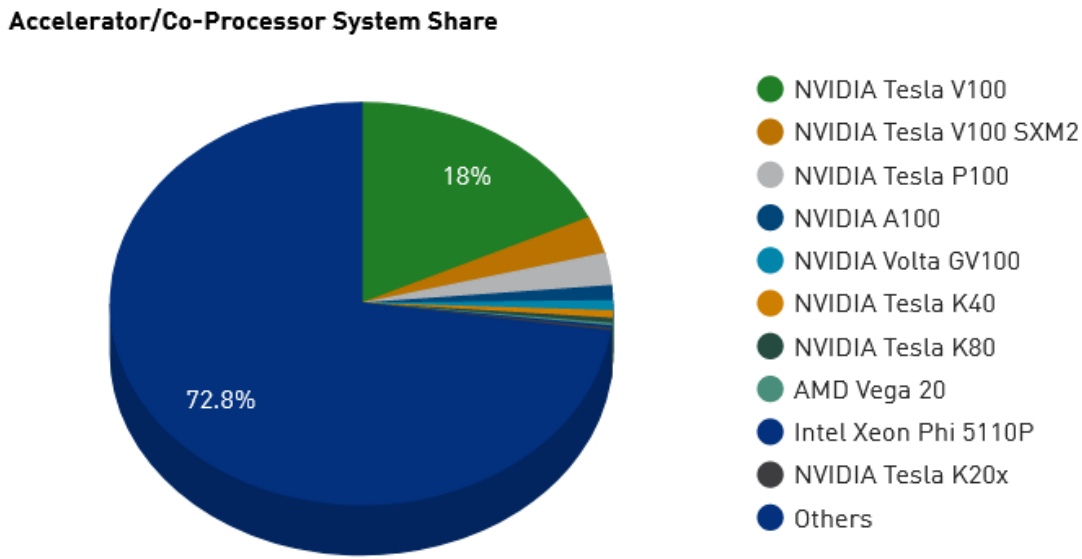


Processing Element

- Virtual CPUs
  - A term that you often get to hear when working with clusters hosted on ‘Cloud’
  - Virtual Machine (VM) assigned to a single physical core
    - A VM is an abstraction/emulation of a computer

Processing Element

- GPU (Graphics Processing Unit) – Add-on devices
  - Traditionally: accelerate image creation
  - Now: GPGPUs for large-scale modeling, genetic programming



source:top500.org

Cluster Elements

- Interconnect
  - Nvidia-Mellanox Switch

source: https://www.nvidia.com/en-in/networking/infiniband-switching/

  - Cabling (determines topology)

https://www.rcac.purdue.edu/training/clusters101/

Cluster Elements

- Storage
  - $HOME
    - Landing directory when you log in to the master node
  - $SCRATCH and/or Parallel File System
    - A fast memory where you should keep all the data needed for executing the task
    - E.g. Lustre, BeeGFS, etc.

Cluster Elements - Software

- Job
  - A task performed by the cluster
  - Set of commands to the cluster, captured in a script, to precisely tell how to execute the task
  - Usually, the set of commands do not require your intervention i.e. non-interactive
    - You issue the commands (read: “submit a job”) and go for coffee..

Cluster Elements - Software

- Batch System - Job Scheduler and Resource manager
  - Provide a user interface to submit, monitor, and run jobs
  - Manage the computational resources mentioned previously
  - Implement the usage policies set by HPC Admin
- E.g. SLURM (Simple Linux Utility for Resource Management), PBS (Portable Batch System) – Torque, Moab.

Cluster Elements - Software

- Partitions/Queues
  - A logical grouping of (a subset of) nodes in the cluster
  - Single node can belong to multiple partitions (not done in practice)
  - Have predefined attributes (and limits) set for a job
    - A job submitted to a specific partition runs with low priority
    - A job can request a maximum of 4 cores
    - etc.

Cluster Elements - Software

- Operating System (OS), Software Development Tools, Applications
  - Linux-based OS in 100% of the supercomputing clusters in top500.org (2015 onwards)
  - Compiler tool chains, Runtime systems, Profilers
  - E.g. GCC, ICC, MPICH, JDK, Docker, Matlab, Intel Parallel Studio, NVProf, Tau etc.

Cluster Elements

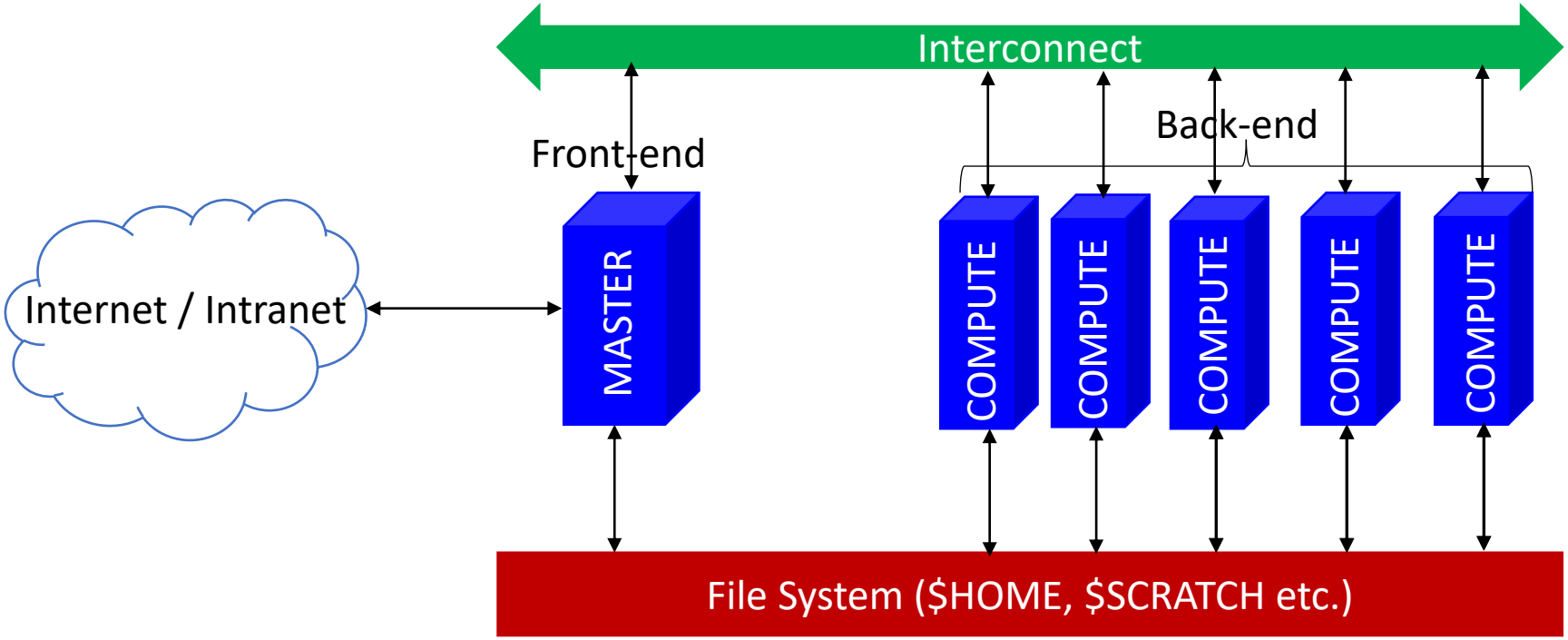
Threads and Processes

- Abstraction provided by OS, are units of execution

- Process
  - Self-contained i.e. has its own private resources to execute/run programs. E.g. of a resource: memory. Is an instance of a running program.
  - Have an illusion that *entire computer* is for itself.
- Thread
  - Belongs to a process. Share memory and other resources among threads of the same process.
  - Have an illusion that *entire processor* is for itself.

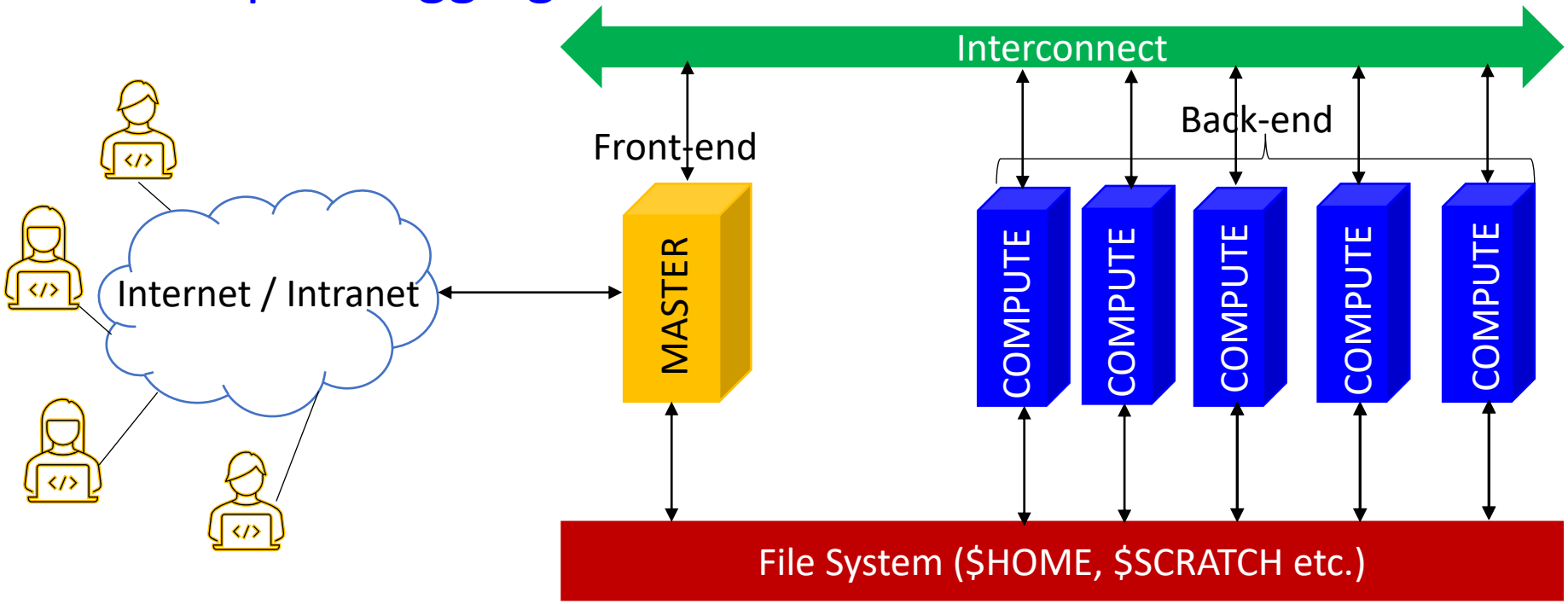
Recap

- Clusters



Clusters

- Step 1: Logging-in



Logging-in

- Logging into remote Linux system (master node) requires you to use SSH ("Secure Shell Protocol")
  - Login credentials are encrypted
  - `SSH server` must be running on the system that you are logging into; Happens on most Linux systems by default.
  - `SSH client`, another piece of software, is used to authenticate and connect to the `SSH server`
    - Client software is available for all platforms (OSs)

Logging-in Windows

- Powershell on Windows 10
  - Press (Windows + R) -> Type “powershell”
  - Type “ssh <username>@<masternode_IP_address>
  - Type ‘Yes’ when prompted (only first time)
  - Provide log-in credentials

Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ndheg> ssh nikhilh@10.250.101.100_
```

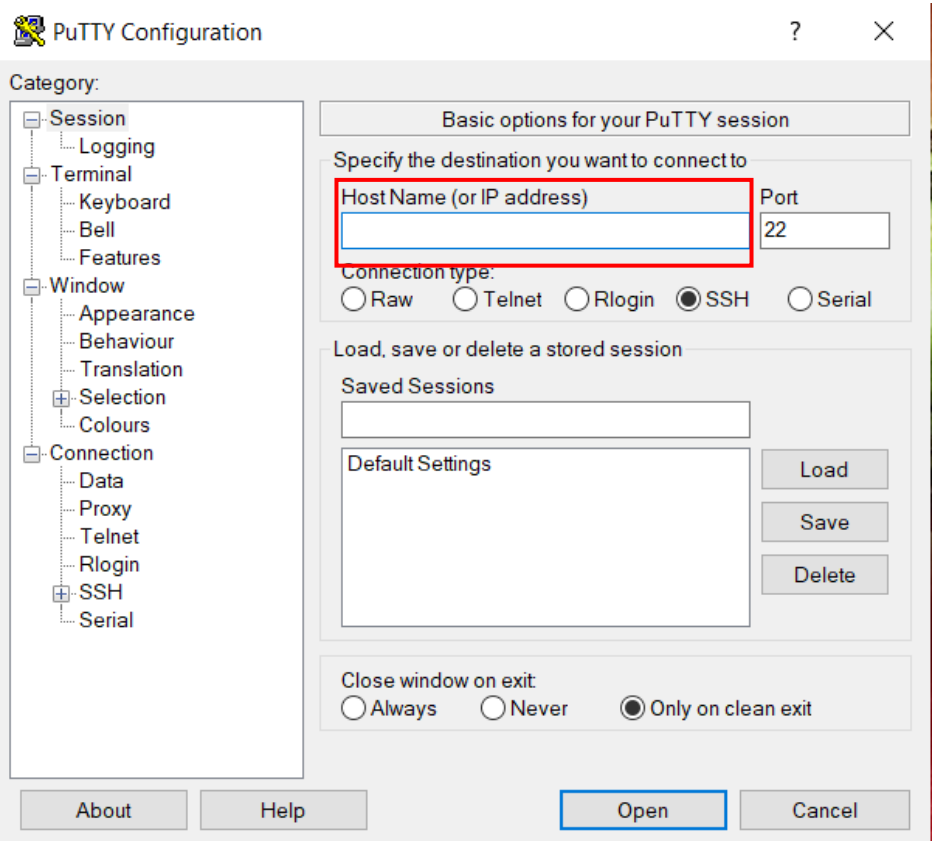
Logging-in Windows

- PuTTY SSH client Windows
  - Download PuTTY from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html (64-bit .exe)
  - Double click on the  icon after downloading

putty

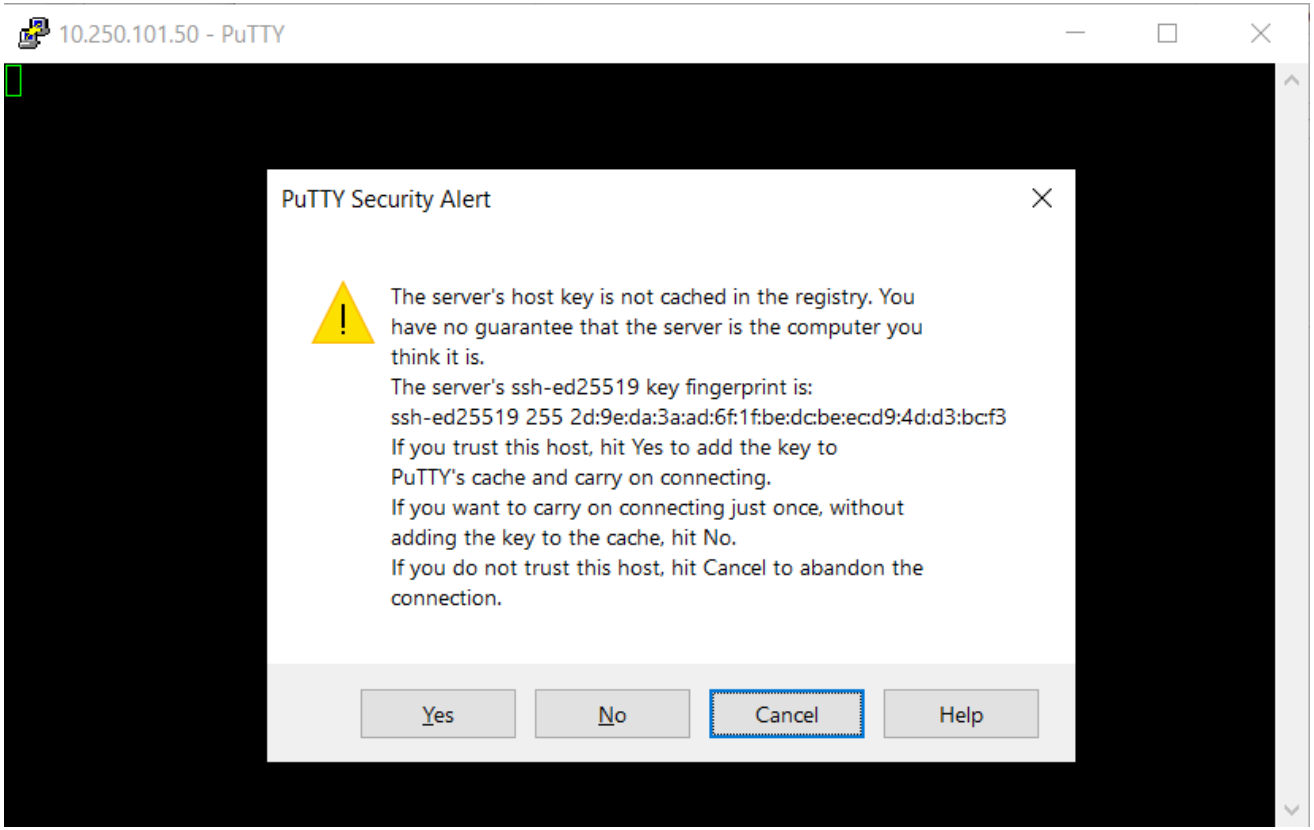
Logging-in Windows

- Type in the Host Name / IP address and click ‘Open’



Logging-in Windows

- Click on ‘Yes’ (you are accepting the server host key)



Logging-in Windows

- Enter log-in credentials

nikhilh@iitdhmaster:~

```
login as: nikhilh
nikhilh@10.250.101.100's password:
Last login: Wed Mar 17 09:53:33 2021 from 10.196.7.237
Intel(R) Parallel Studio XE 2020 Update 2 for Linux*
Copyright 2009-2020 Intel Corporation.
[nikhilh@iitdhmaster ~]$
```

Logging-in MAC

- Open the ‘Terminal’ program on MAC (Go -> Applications -> Terminal)

```
Last login: Sun Mar  7 11:35:13 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
apples-MacBook-Pro:~ apple$
```

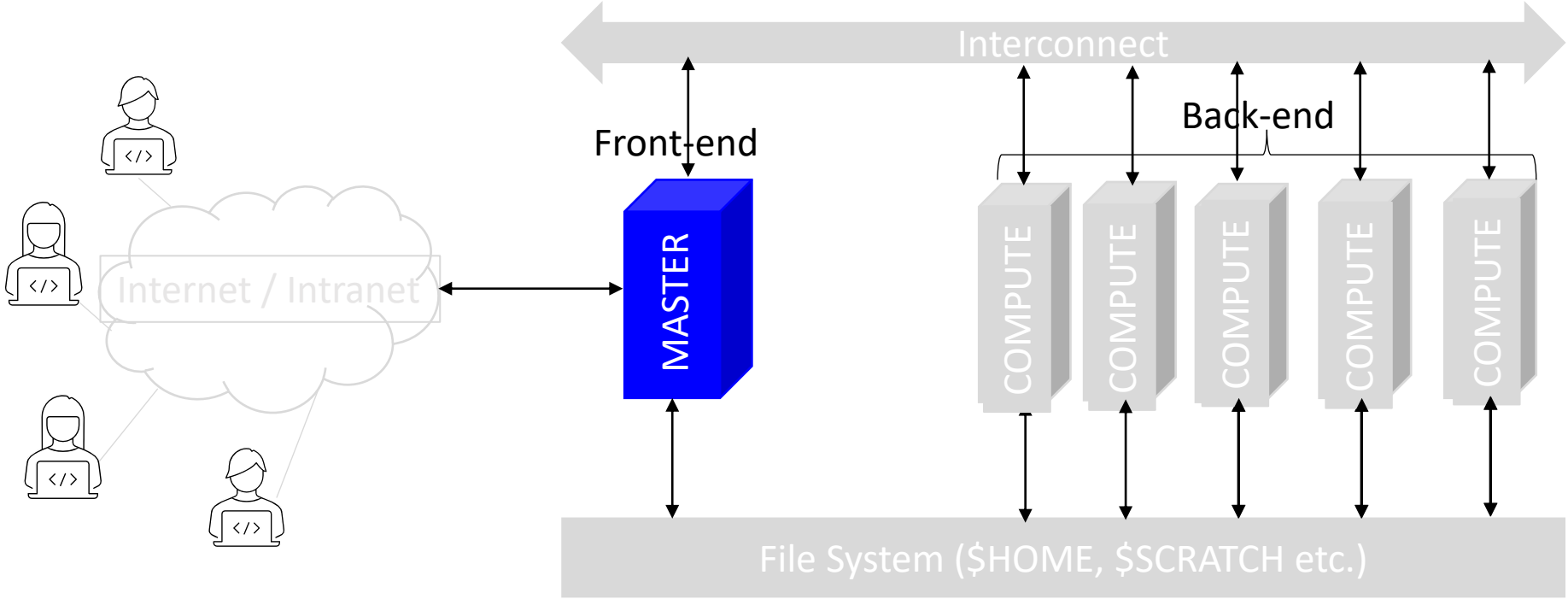
- Type “ssh <username>@<masternode_IP_address>
- Type ‘Yes’ when prompted (only first time)
- Provide log-in credentials

Logging-in Linux

- If you are a Linux user, you know what a ‘Terminal’ is ☺
- Type “ssh <username>@<masternode_IP_address>
- Type ‘Yes’ when prompted (only first time)
- Provide log-in credentials

Clusters

- Step 2: Activities on the Master Node



Useful Linux Commands

```
ls, ls -l
man
mkdir
cd
pwd
cp
mv
scp
rm  //use with caution!
```

```
cat
less
head, tail
vi, vim, emacs, nano, pico
gzip, tar, zip
who
```

Type "`man <command_name_here>`" *on the Linux terminal to get help info*

Useful Linux Commands

`scp`

To move files back and forth between master node and your local system

```
scp file1 <user_name>@<master_node_ip>
```

from your system to Master node

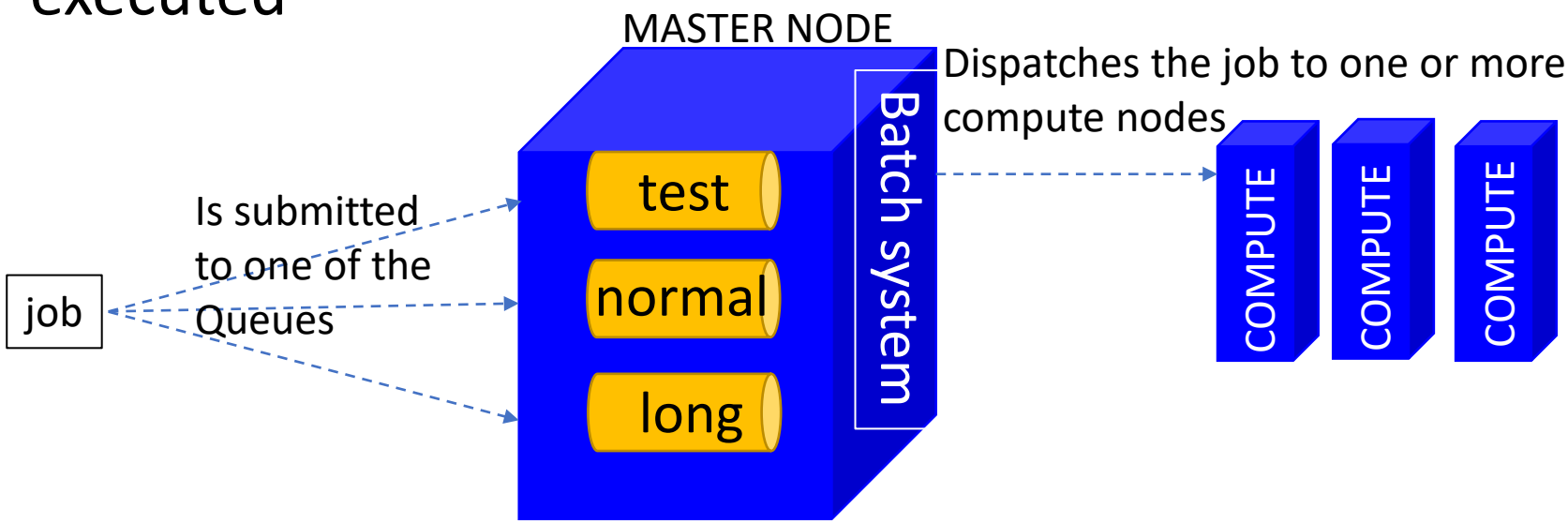
```
Windows PowerShell
PS C:\Temp> scp README.txt nikhilh@10.250.101.100:
nikhilh@10.250.101.100's password:
README.txt                                    100%  137    19.4KB/s   00:00
PS C:\Temp> scp nikhilh@10.250.101.100:pscp.exe .
nikhilh@10.250.101.100's password:
pscp.exe                                      100%  669KB   3.9MB/s   00:00
PS C:\Temp>
```

from Master node to your system

```
scp <user_name>@<master_node_ip>:file1 .
```

Scheduler

- Runs continuously on the Master node
- Scans the jobs submitted
  - user jobs are submitted to queues
- Determines when and where the jobs are to be executed



Clusters – How are they programmed?

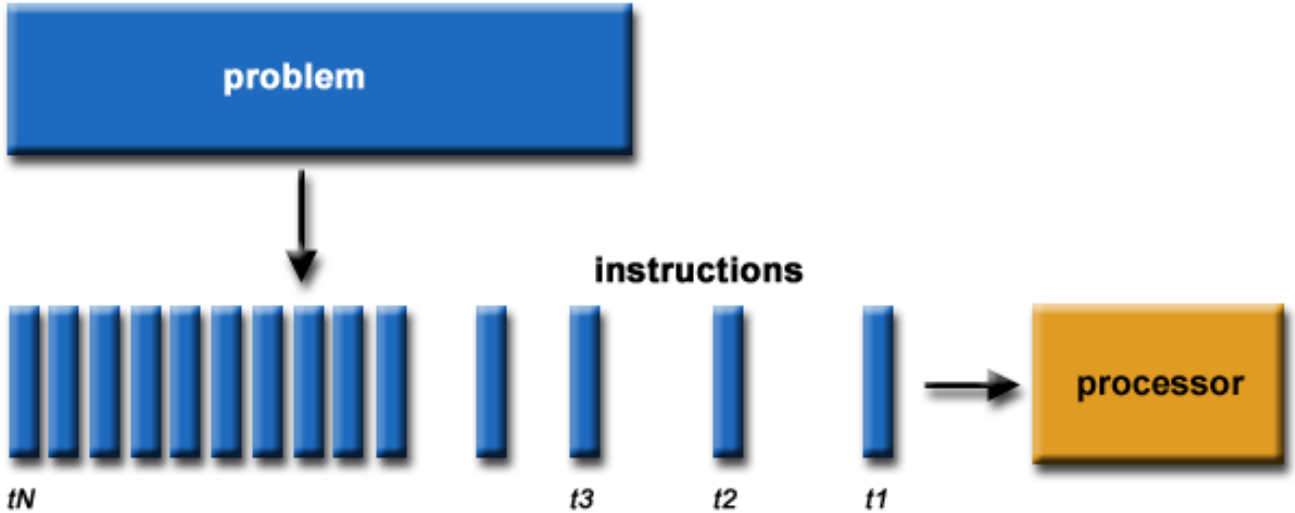
- It’s (all) about parallelism!

Clusters – How are they programmed?

- It’s (all) about parallelism!
  - exploiting parallelism is crucial for improved performance on multicore systems

Clusters – How are they programmed?

- It's (all) about parallelism!
- Sequential Program - *single sequence of instructions - single-threaded.*



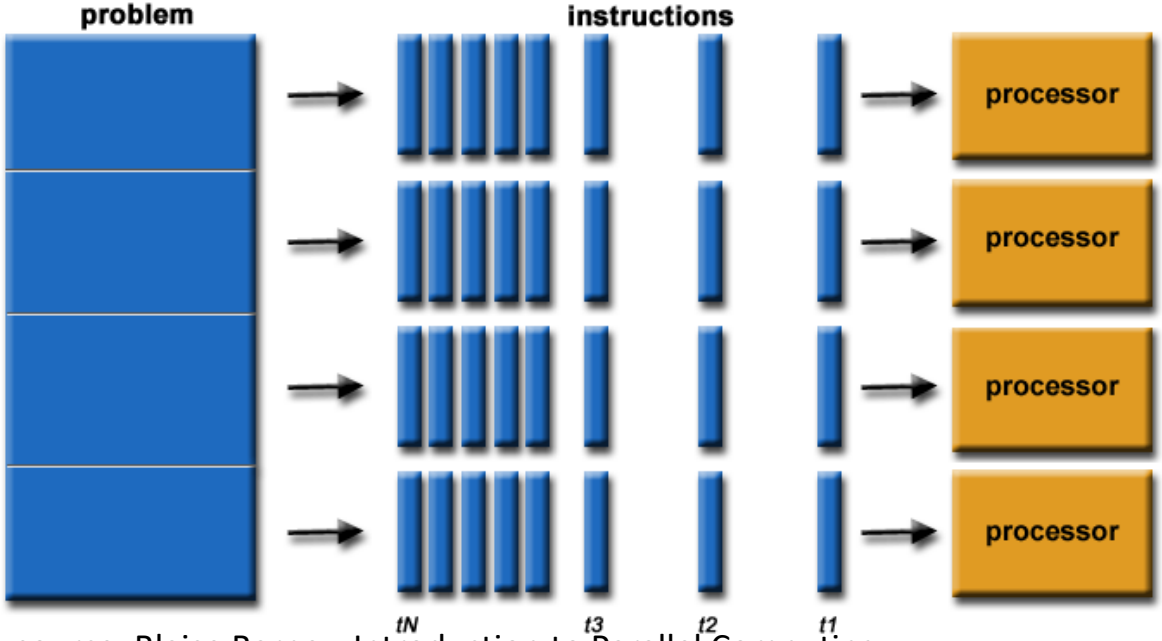
source: Blaise Barney, Introduction to Parallel Computing, https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Clusters – How are they programmed?

- It's (all) about parallelism!
- Sequential Program - *single sequence of instructions - single-threaded.*
- Concurrent Program - *Multiple sequence of instructions executing concurrently.* Instructions from one sequence *may* communicate and interfere with other.
  - How are they (multi- threads) executing?
    - Multiprogramming – threads multiplexing their execution on **one processor**
    - Multiprocessing – threads multiplexing their execution on multiprocessor or **multicores**
    - Distributed Processing – processes multiplexing their executions on **multiple nodes**

Clusters – How are they programmed?

- Parallel Program – a *concurrent program* designed to execute on ***parallel hardware***
  - Multiple processors in a computer (multiprocessing),
  - Multiple computers in a network (distributed processing)



source: Blaise Barney, Introduction to Parallel Computing, https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Parallel Hardware

- Flynn's taxonomy - categories of computing systems
  - Based on how processing elements see *instruction* and *data*

	Single Data (SD)	**Multiple Data (MD)**
Single Instruction (SI)	SISD	SIMD
Multiple Instruction (MI)	MISD	MIMD

Parallel Hardware

- Flynn's taxonomy - categories of computing systems
  - Based on how processing elements see *instruction* and *data*

	Single Data (SD)	**Multiple Data (MD)**
Single Instruction (SI)	SISD	SIMD
Multiple Instruction (MI)	MISD	MIMD

Clusters belong to this category

Parallel Hardware

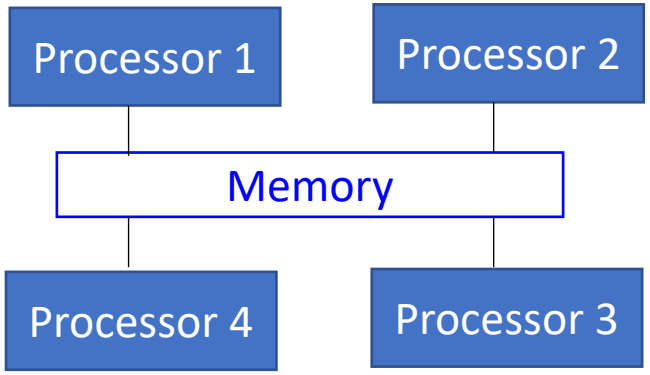
- Categorization based on how processing elements see *system memory*
  - Shared Memory
  - Distributed Memory
  - Distributed-Shared Memory

Parallel Hardware

- Categorization based on how processing elements see *system memory*
  - Shared Memory
  - Distributed Memory
  - Distributed-Shared Memory

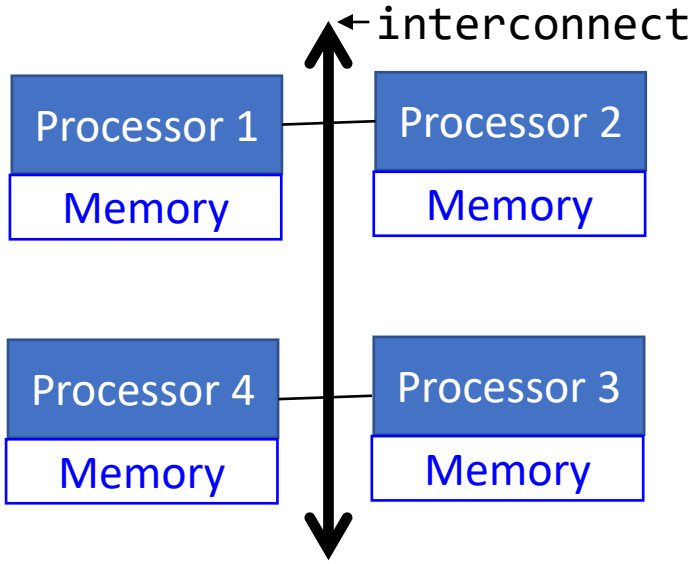
Most clusters belong to this category

Parallel Hardware



Shared Memory Architecture

Distributed Memory Architecture



Data and Control Parallelism

- Threads executing the same function but with different data – ***data parallelism***
  - E.g. two construction workers laying bricks to build walls of different parts of a house
- Threads executing different functions – ***control parallelism***.
  - E.g. A carpenter getting a window frame ready while a mason is laying bricks in the wall

Need for Open MP

- Multithreaded programs using `std::thread` or `pthreads :`
  - do not *scale automatically* - add more processors and you will have to rewrite the program to utilize available processors
  - (pthreads) programs are not portable

- Open MP (multiprocessing) / OMP provides a scalable and portable alternative to data-parallel computing on *shared-memory architectures*

What is OpenMP

- An open standard for shared memory programming in C/C++ and Fortran
- Supported by IBM, Intel, GNU and others
- Same program running on multiple threads each operating on different data (single program multiple data – SPMD)

Programming in OpenMP

Compiler directives

```
#pragma omp parallel
{
    #pragma omp for
    for(i=0;i<N;i++)
        fruits[i].Energy();
}
```

- `#pragma parallel`

  executes as many threads as there are processors

- `#pragma omp for`

  divides the whole work among available threads

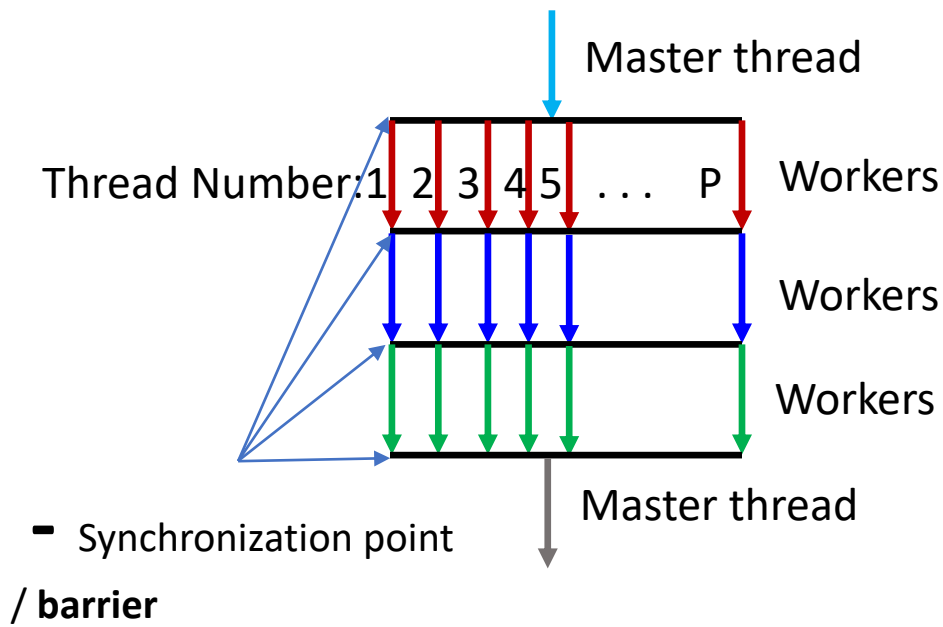
Programming in OpenMP

```
#pragma omp parallel
{
   #pragma omp for
   for(i=0;i<N;i++)
      fruits[i].Energy();
}
```

- Example of loop parallelism
  - Common in scientific codes
- Programmer is still responsible for handling data races.

Programming in OpenMP

```
//code region 0
#pragma omp parallel
{
    //code region 1
    #pragma omp for
    for(i=0;i<N;i++) {
        //code region 2
    }
    //code region 3
}
//code region 4
```



- Execution begins with a master thread (executes code region 0 and 4)
- Master thread creates / forks worker threads (execute code region 1,2, and 3)
- Worker threads join master thread

fork / join parallelism

Programming in OpenMP

- How many workers? / threads?
  - = number of processors by default. Can also be set with `omp_set_num_threads(P)`
  - Can query the number of processors available on a machine with `omp_get_num_procs()`
  - Each thread has an ID returned by `omp_get_thread_num()`

Programming in OpenMP

- Example – *reductions (sum of array elements)*

```
#include<omp.h>
int main() {
    int total=0, a[NUM_SAMPLES];
    for(int i=0;i<NUM_SAMPLES;i++)
        a[i] = i;
    #pragma omp parallel
    {
        //#pragma omp for reduction(+: total)
        for(i=0;i<NUM_SAMPLES;i++)
            total = total + a[i];
    }
}
```

Programming in OpenMP

- Other operations supported in reductions:
  - +: addition
  - *: multiplication
  - |: bitwise OR
  - &: bitwise AND
  - ^: bitwise exclusive OR
  - ||: logical OR
  - &&: logical AND

  Note the *commutative nature* of these operations

Programming in OpenMP - Summary

- Open MP provides a way to specify what parts of program execute in parallel with one another
- How the work is distributed across different cores
- Whether to serialize (atomic) accesses to memory
- What order memory is read and written (barriers – `nowait` clause)

All while providing *portable performance*

Distributed Memory Programming

- Program executes as a collection of processes

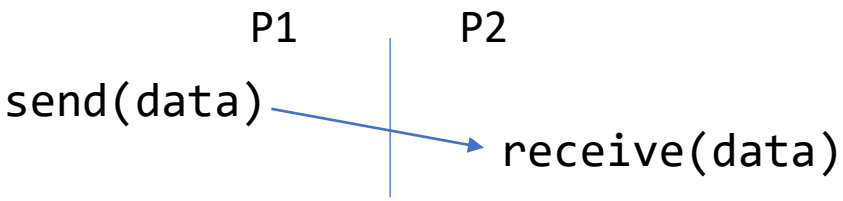
  Distributed Processing – processes multiplex their executions on multiple machines

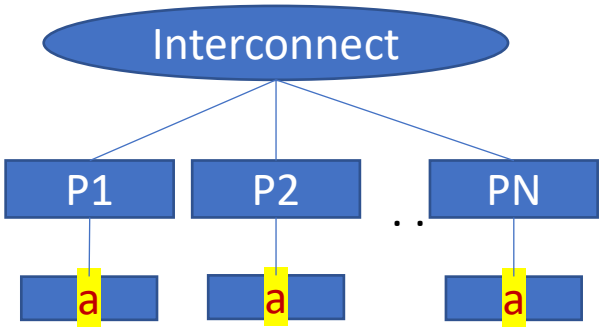
- Each process / processor has its own memory
  - Total memory available for an MPI program is the combined memory space of all processors
  - Exchanging data requires cooperation between two processors

Distributed Memory Programming

- Data exchange requires explicit communication:

Programmer must set up communication channels and exchange data





Value of a in P1 may be different from that in P2

1) P1 sends a copy of a to P2,

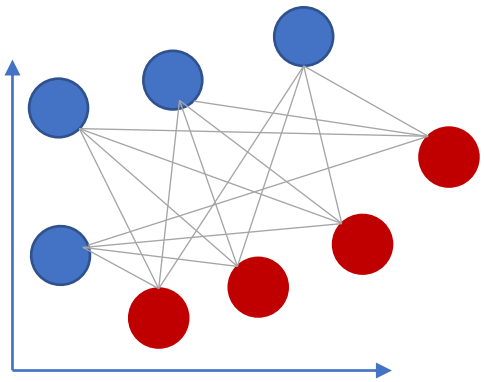
2) P2 receives the copy stores it in its data region.

- Every data element must belong to one of the memory spaces Programmer must decide where to place data

Distributed Memory Programming

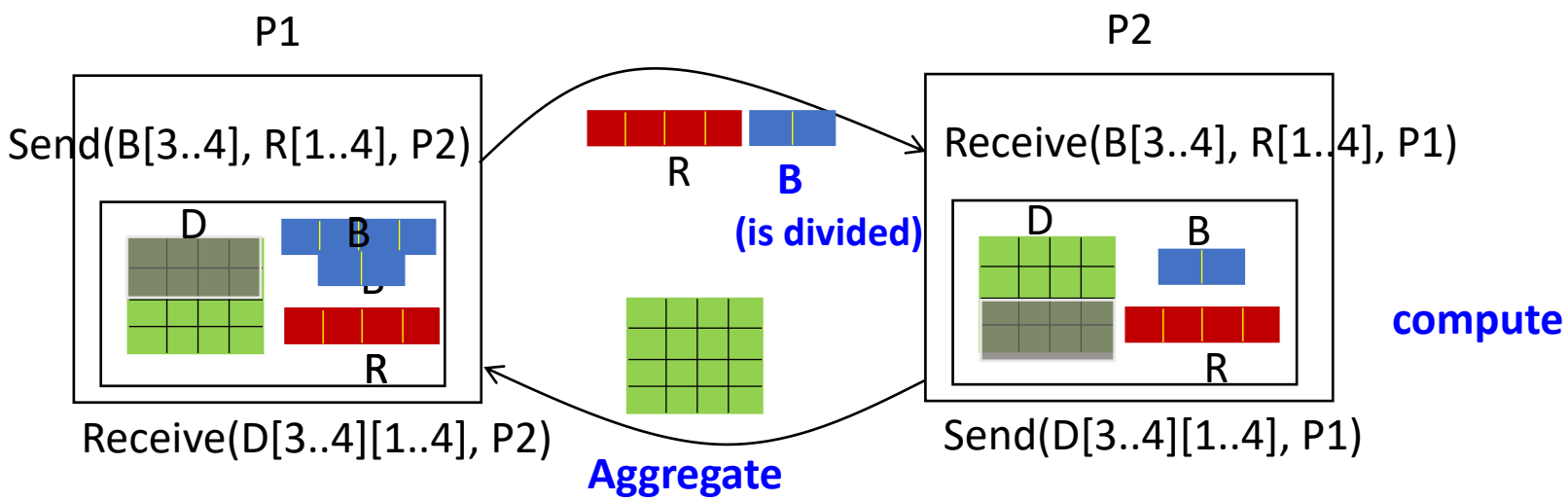
- More complex than shared-memory programming model
- Most programs are written in *Single Program Multiple Data* (SPMD) model
- Computing power and cost scaling is better than with shared memory – e.g. rack mounted blades
- E.g. weather forecasting, simulating dynamics of gases and fluids - where to put exhaust fans in a basement parking? can an ATV topple while wading through body of water?

Distributed Memory Programming - Example



Calculate the distance from each point in set Blue (B) to that in set Red (R) and store the result in set D

```
for(i=1 to 4)
    for(j=1 to 4)
        D[i][j] = distance(B[i],R[j])
```



Distributed Memory Programming - Example

```
        Processor P1
//initialize B, R, and D
Send(B[3..4], R[1..4], P2)
for(i=1 to 2)
    for(j=1 to 3)
        D[i][j] = distance(B[i], R[j])
Receive(D[3..4][1..4], P2)
```

```
        Processor P2
//initialize B, R, and D
Receive(B[3..4], R[1..4], P1)
for(i=3 to 4)
    for(j=1 to 3)
        D[i][j] = distance(B[i], R[j])
Send(D[3..4][1..4], P1)
```

- How is work divided among processors?
- What does it mean for send and receive to complete?
- How does a receiver interpret data that a sender sends?

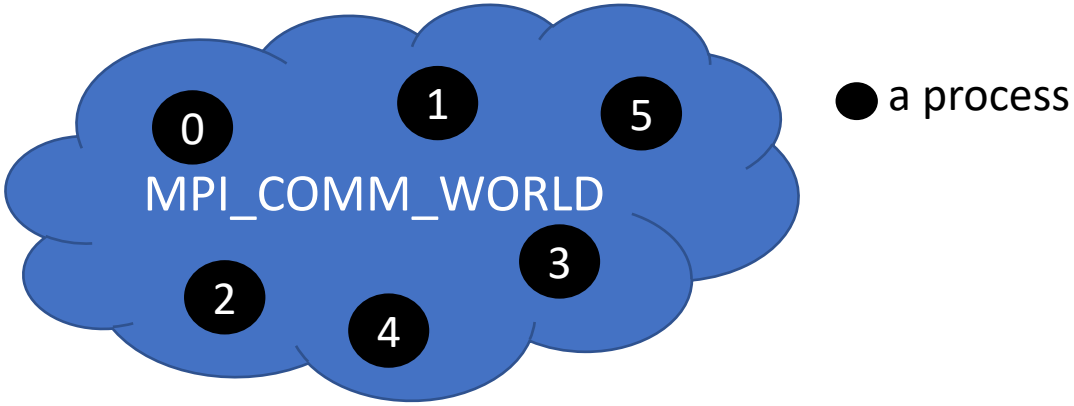
Converting to MPI Program

- Initialization and Termination

```
#include<mpi.h>

int main(int argc, char* argv[]) {
    MPI_Init(&argc,&argv); //initializes MPI Environment

    //all other code here

    MPI_Finalize(); //releases system resources
}
```

Converting to MPI Program

- Environment
  - 0,1,. . 5 – ranks / process numbers
  - MPI_COMM_WORLD – communicator / group of processes that are allowed to exchange messages



  - MPI_Init initializes the communicator.

Converting to MPI Program

- Rank and Size: obtaining process number and number of processes in the execution environment

```
#include<mpi.h>
int main(int argc, char* argv[]) {
    int rank, size;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Finalize();
}
```

Converting to MPI Program

- Divide the work and compute

```
#include<mpi.h>
int main(int argc, char* argv[]) {
    int rank, size;
    float d[NUMPOINTS][NUMPOINTS], b[NUMPOINTS], r[NUMPOINTS];
    //initialize b and r arrays from input
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    for(i=rank*NUMPOINTS/size; i<(rank*NUMPOINTS/size)+NUMPOINTS/size; i++)
        for(j=0;j<NUMPOINTS;j++)
            d[i][j] = distance(b[i], r[j]);
    MPI_Finalize();
}
```

Converting to MPI Program

- Aggregate the results at master – MPI_Send, MPI_Recv

```
#include<mpi.h>
int main(int argc, char* argv[]) {
    int rank, size;
    float d[NUMROWS][NUMROWS], b[NUMROWS], r[NUMROWS];
    //initialize b and r arrays from input
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //compute d[i][j] as before
    if(rank !=0){
     for(int i=rank*NUMPOINTS/size;i<(rank*NUMPOINTS/size + NUMPOINTS/size);i++
        MPI_Send(d[i], NUMPOINTS, MPI_FLOAT, 0, MY_MESSAGE_TAG, MPI_COMM_WORLD)
    else
     for(int i=NUMPOINTS/size;i<NUMPOINTS;i++)
        MPI_Recv(d[i], NUMPOINTS, MPI_FLOAT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
    MPI_Finalize();
}
```

Converting to MPI Program

- Aggregate the results at master – MPI_Send, MPI_Recv

```
#include<mpi.h>
int main(int argc, char* argv[]) {
    int rank, size;
    float d[NUMROWS][NUMROWS], b[NUMROWS], r[NUMROWS];
    //initialize b and r arrays from input
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //compute d[i][j] as before
    if(rank !=0){
     for(int i=rank*NUMPOINTS/size;i<(rank*NUMPOINTS/size + NUMPOINTS/size);i++
        MPI_Send(d[i], NUMPOINTS, MPI_FLOAT, 0, MY_MESSAGE_TAG, MPI_COMM_WORLD)
    else
     for(int i=NUMPOINTS/size;i<NUMPOINTS;i++)
        MPI_Recv(d[i], NUMPOINTS, MPI_FLOAT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
    MPI_Finalize();
}
```

Send buffer

Count of sent items

Data type of sent items

Destination process ID

Message ID

Recv buffer

Recv count

any Message ID

From any Source Process

Converting to MPI Program

- Aggregate the results at master (collectives) – MPI_Gather

```
#include<mpi.h>
int main(int argc, char* argv[]) {
    int rank, size;
    float d[NUMROWS][NUMROWS], tmpd[NUMROWS], b[NUMROWS], r[NUMROWS];
    //initialize b and r arrays from input
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //compute distance into tmpd[NUMROWS];

    MPI_Gather(tmpd, NUMROWS, MPI_FLOAT, d, NUMROWS, MPI_FLOAT, 0,
MPI_COMM_WORLD)
    MPI_Finalize();
}
```

Send buffer

Count of sent items

Data type of sent items

Source Process ID

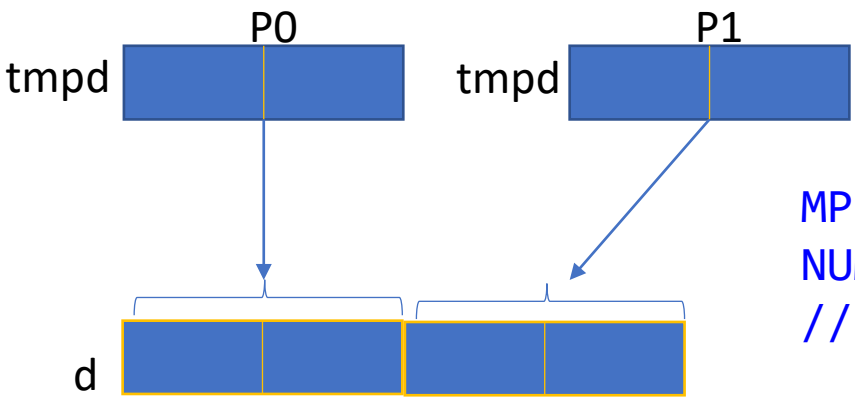
Recv buffer

Recv count

Converting to MPI Program

- collectives – MPI_Gather

```
MPI_Gather(sendbuf, sendcnt, send_type, recvbuf, recvcnt, recv_type,
           source_proc, MPI_COMM_WORLD)
```




```
MPI_Gather(tmpd, NUMROWS, MPI_FLOAT, d,
NUMROWS, MPI_FLOAT, 0, MPI_COMM_WORLD)
//NUMROWS = 2, 2 processes, 2x2 matrix
```

MPI Programming - Collectives

- MPI_Barrier – all processes wait at that line of code – a synchronization point
- MPI_Bcast – Broadcasting data from one process
- MPI_Scatter – Distribute data from master to all processes
- MPI_Gather – Collect data from all processes at master
- MPI_Allgather – Same as gather but all processes collect results
- MPI_Reduce – Aggregate results at master (recall `reduction` in OMP)
- MPI_Allreduce – Aggregate results at all processes

refer https://computing.llnl.gov/tutorials/mpi/ for API details

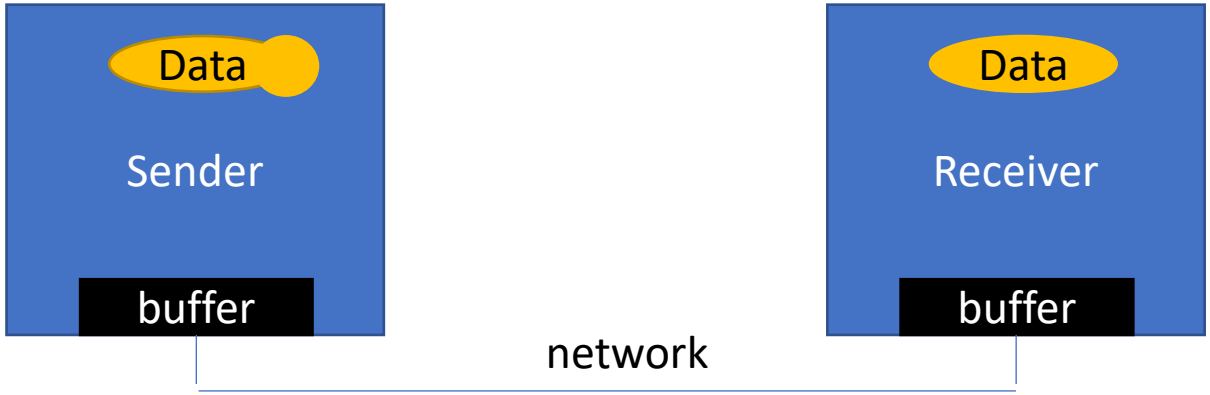
MPI Programming – Point-to-Point

- In most MPI programs, communication is between a pair of processors.
  - Think other types of communication: Broadcast (one-to-all), Reduce(All-to-one), Scatter (one-to-several), Gather(several to one), All-to-All

- When is `send/receive` complete?
  - Synchronous / Asynchronous
  - Blocking / non-blocking
  - Buffered

MPI Programming – Point-to-Point

- Synchronous vs. Asynchronous
  - Synchronous: sender notified when message is received
  - Asynchronous: sender only knows that the message is sent



MPI Programming – Point-to-Point

- Blocking vs. Non-blocking
  - Blocking:
    - Sender waits until message is transmitted – buffer is empty
    - Receiver waits until message is received – buffer is full
  - Non-blocking
    - sender continues execution immediately after calling send