DISTRIBUTED AND CLOUD COMPUTING

LAB2 INTRODUCTION TO MPI COMMUNICATION MODELS

TROUBLESHOOTING

- Cannot open terminal (gnome-terminal) after installing Ubuntu in VirtualBox: possibly a problem in locale.
  - Press **Ctrl+Alt+F3**, and log in
  - Use admin permission to run the following commands:

```
locale-gen
localectl set-locale LANG="en_US.UTF-8"
reboot
```

  - Wait for the reboot to complete
- Don't simply copy and paste the code! Type by yourself.
  - These two symbols are different: – -
    - E.g., when you run `mpirun -np 4 ./mpihw`

The hello-world example

No communication between MPI processes

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
processor_name, world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

https://github.com/wesleykendall/mpitutorial/blob/gh-pages/tutorials/mpi-hello-world/code/mpi_hello_world.c

COMPILING AND RUNNING

- Use `mpicc` to compile an MPI application
  - Usage is the same as gcc: `mpicc foo.c –o foo`
- Use mpirun to launch the MPI application
  - `mpirun -np <num_process> <program>`
  - If you do not use mpirun, sub-processes can’t be spawned.

MPI BASIC INTERFACE

```
int MPI_Init(int *argc, char ***argv)
```

- The first function call of any MPI application
- Initialize MPI environment
- Start of parallel part of the application

MPI BASIC INTERFACE

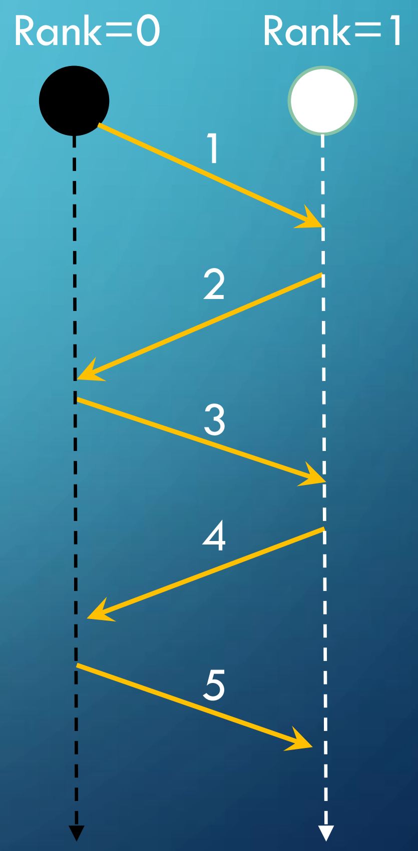
- `int MPI_Comm_rank(MPI_Comm comm, int *rank)`
  - Get the rank of the current process in a communicator.
- `int MPI_Comm_size(MPI_Comm comm, int *size)`
  - Get the size (num. of processes) in a communicator.
- `MPI_COMM_WORLD`
  - The default communicator, groups all processes at start. There's also a communicator MPI_COMM_SELF provided by MPI, to communicate with the process itself.

MPI BASIC INTERFACE

- `int MPI_Finalize()`
  - The last function call of MPI application.
  - Terminate the MPI application, also the sub-processes created by mpirun
  - End of parallel part of MPI application. Serial code can still be run in the main process.

AN EXAMPLE: PING-PONG

- The hello-world example does not involve communication between processes
- Here we consider an example that allows two MPI processes to play ping-pong
- MPI processes send messages to each other



Example program: ping-pong

It implements message passing between two MPI processes

```
const int PING_PONG_LIMIT = 10;
......

int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;

        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
               world_rank, ping_pong_count, partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
               world_rank, ping_pong_count, partner_rank);
    }
}
```

https://github.com/wesleykendall/mpitutorial/tree/gh-pages/tutorials/mpi-send-and-receive/code/ping_pong.c

Example program: ping-pong

It implements message passing between two MPI processes

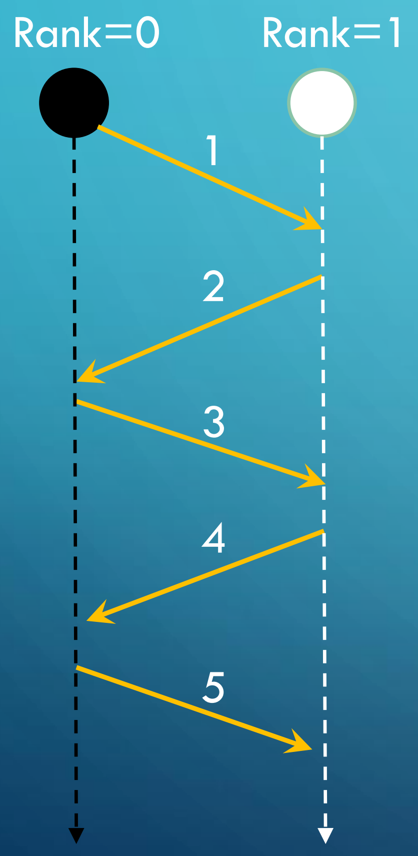
world_rank	partner_rank
1	0
0	1

```
const int PING_PONG_LIMIT = 10;
......

int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
                    Sender buffer    Num. of elements to be sent    Data type    Rank of destination process    Tag    Communicator
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
               world_rank, ping_pong_count, partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
               world_rank, ping_pong_count, partner_rank);
    }
}
```

https://github.com/wesleykendall/mpitutorial/tree/gh-pages/tutorials/mpi-send-and-receive/code/ping_pong.c

Two MPI processes playing ping-pong




```
~ mpirun -np 2 ./mpi-ping-pong
0 sent and incremented ping_pong_count 1 to 1
1 received ping_pong_count 1 from 0
1 sent and incremented ping_pong_count 2 to 0
0 received ping_pong_count 2 from 1
0 sent and incremented ping_pong_count 3 to 1
0 received ping_pong_count 4 from 1
0 sent and incremented ping_pong_count 5 to 1
0 received ping_pong_count 6 from 1
0 sent and incremented ping_pong_count 7 to 1
0 received ping_pong_count 8 from 1
0 sent and incremented ping_pong_count 9 to 1
0 received ping_pong_count 10 from 1
1 received ping_pong_count 3 from 0
1 sent and incremented ping_pong_count 4 to 0
1 received ping_pong_count 5 from 0
1 sent and incremented ping_pong_count 6 to 0
1 received ping_pong_count 7 from 0
1 sent and incremented ping_pong_count 8 to 0
1 received ping_pong_count 9 from 0
1 sent and incremented ping_pong_count 10 to 0
```

MPI SEND/RECEIVE

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int
tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
MPI_Comm comm, MPI_Status *status)

/**OUT buf                  Sender buffer

*IN count                   Num. of elements to be sent

*IN datatype                Data type

*IN source/dest             Rank of source/destination process

*IN tag Message             Tag

*IN comm                    Communicator

*OUT status                 Contains the actual received message */
```

MPI DATATYPE

- MPI supports various data types to be send among processes.
- In complex MPI applications, we typically use MPI_BYTE to communicate with custom protocols.

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_INT	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI COMMUNICATION MODE

- **Standard Mode, Buffered Mode, Synchronous Mode, Ready Mode**
- They have the same set of parameters
- Differences: The method of sending message and the state of receiver
- Locality of mode: If the mode requires communicating with other processes.
  - Local: Completion of procedure depends only on local process
  - Non-local: Completion of procedure needs to execute some MPI procedure on another process

MPI COMMUNICATION MODE

- Standard mode
  - In standard mode, MPI library itself determines if the data will be buffered.
  - Buffered: Copy the data into a buffer and return immediately. The sending will be done by MPI in background.
  - Non-buffered: Return when the data has completed sending.
  - Standard mode is **non-local**, since it may need to communicate with recipient to complete sending.

MPI COMMUNICATION MODE

- Buffered mode
  - In buffered mode, MPI copies the data to a provided buffer and return immediately. The sending is done by MPI in background.
  - Buffered mode is local.

MPI COMMUNICATION MODE

- Synchronous mode
  - Synchronous mode only returns when the recipient has started receiving message.
  - Synchronous mode is non-local.

MPI COMMUNICATION MODE

- Ready mode
  - Ready mode ensures the recipient is at ready state (waiting for receiving message), or it will raise an error
  - Ready mode is non-local.

MPI BLOCKING AND NON-BLOCKING COMMUNICATION

- Blocking communication: The function waits for operation to complete, or at least is securely copied by MPI library.
- Non-blocking communication: Always returns immediately, the actual operation is completed by MPI in background. User must ensure the operation is completed before doing the next, or there may be conflicts.

SEND/RECEIVE FUNCTIONS

SEND	Blocking	Nonblocking
Standard	mpi_send	mpi_isend
Ready	mpi_rsend	mpi_irsend
Synchronous	mpi_ssend	mpi_issend
Buffered	mpi_bsend	mpi_ibsend

RECEIVE	Blocking	Nonblocking
Standard	mpi_recv	mpi_irecv

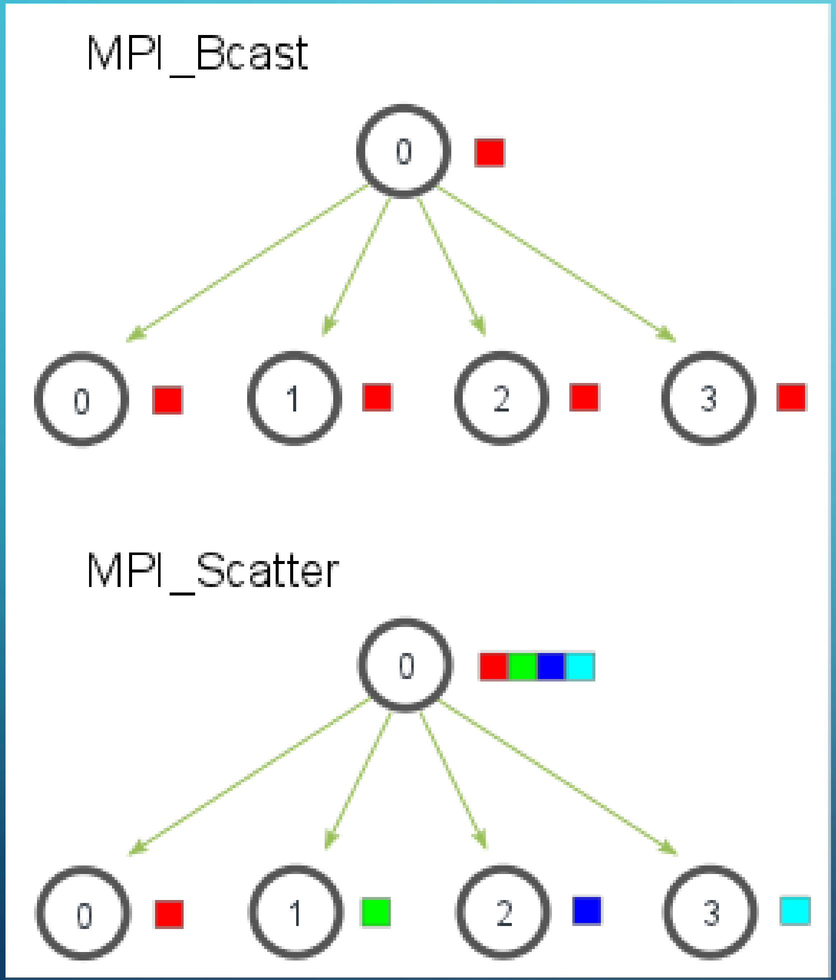
COLLECTIVE COMMUNICATION

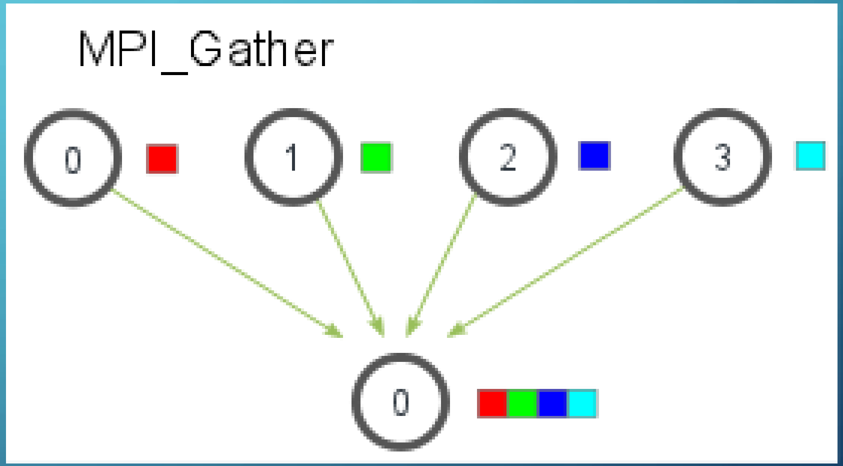
- Collective communication refers to the communication among multiple (3 or more) processes.
- One to many(1-N), many to one(N-1), many to many(N-N)
- In 1-N, N-1, the single process is called *root*.
- Functionalities: Communicate, synchronization, computation

COLLECTIVE COMMUNICATION

- `int MPI_Barrier(MPI_Comm comm)`
  - Synchronization
- `int MPI_Bcast(void* buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)`
  - Boradcast message to all processes (including self)
- `int MPI_Gather(void * sendbuf, int sendcount, MPI_Datatype sendtype, void * recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
- `int MPI_Scatter(void * sendbuf, int sendcount, MPI_Datatype sendtype, void * recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`

1-N AND N-1OPERATIONS





https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/

COLLECTIVE COMMUNICATION

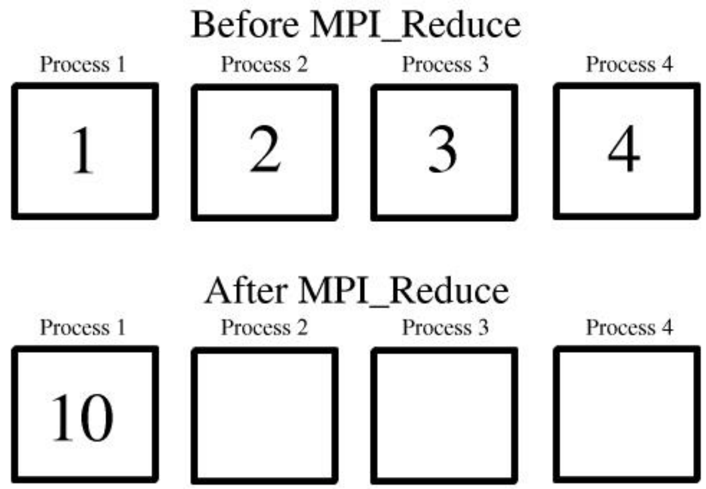
- int MPI_Reduce(void * sendbuf, void * recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

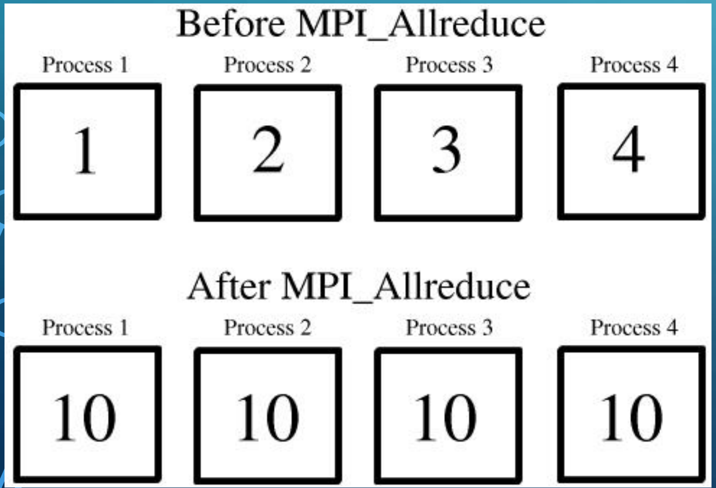
- int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

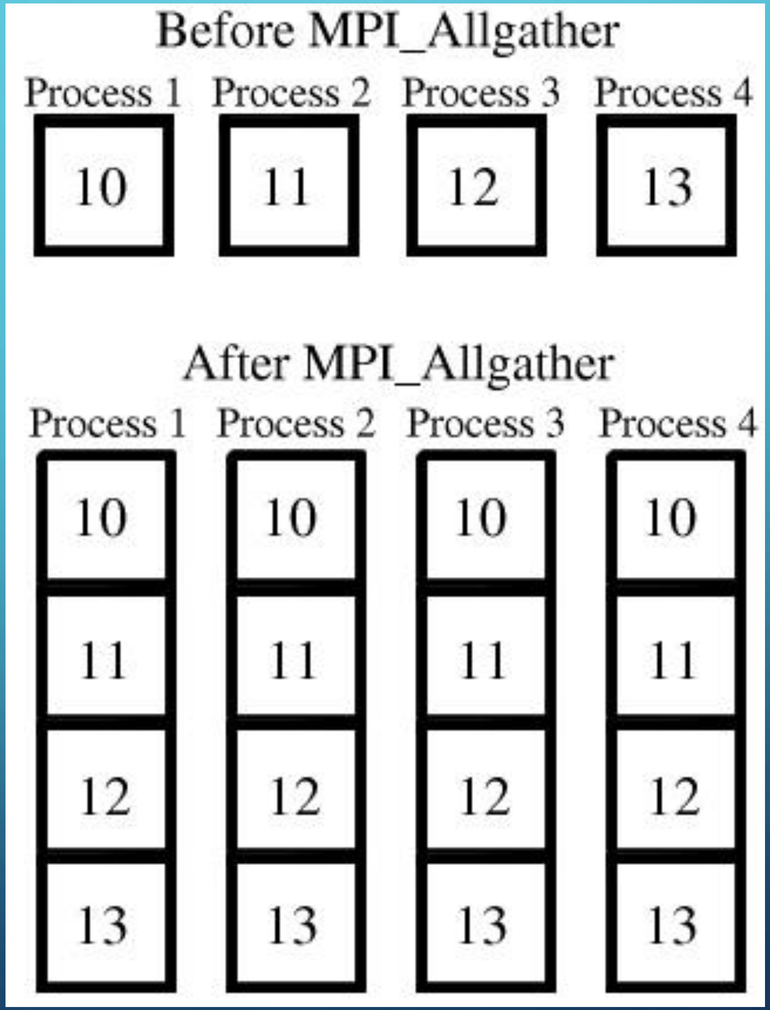
- int MPI_Allgather(void * sendbuf, int sendcount, MPI_Datatype sendtype, void * recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

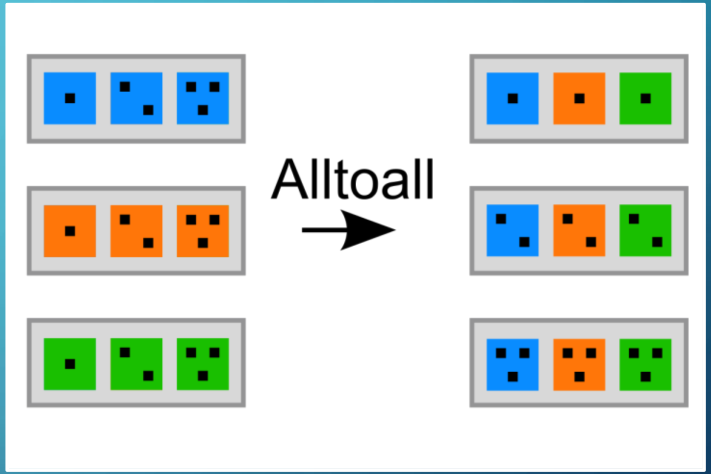
- int MPI_Alltoall(void * sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

N-N OPERATIONS





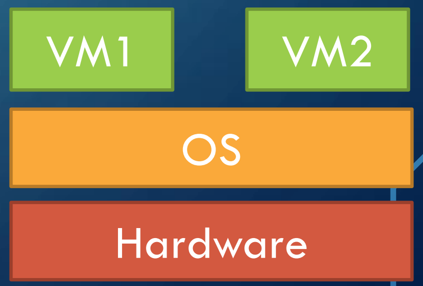




http://www.rc.usf.edu/tutorials/classes/tutorial/mpi/chapter8.html

RUNNING MPI PROGRAMS ON A VM CLUSTER

- MPI programs run on clusters that are composed of multiple machines
- We can emulate such a cluster with multiple VMs
- Our task: creating a cluster of VMs (here we use two VMs), and run an MPI program on this cluster
- Steps:
  - Set up the network connection between the two VMs
  - Enable password-less SSH connection between the two VMs
  - Compile the MPI program and deploy it on each VM
  - Run the MPI program



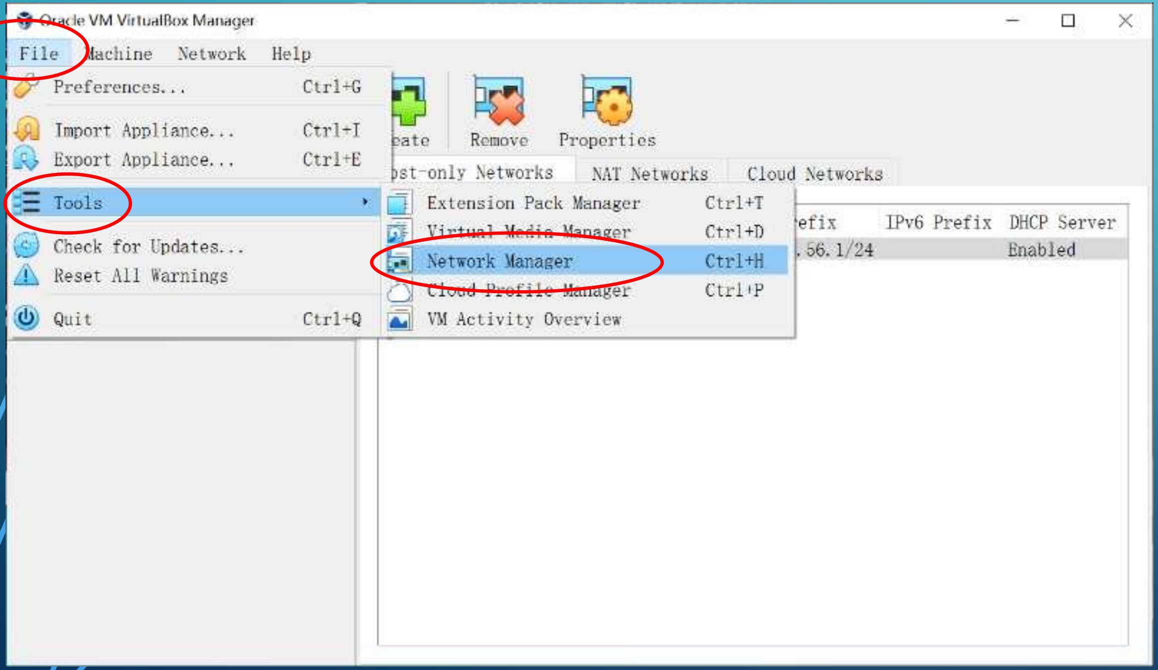
RUNNING MPI PROGRAMS ON A VM CLUSTER

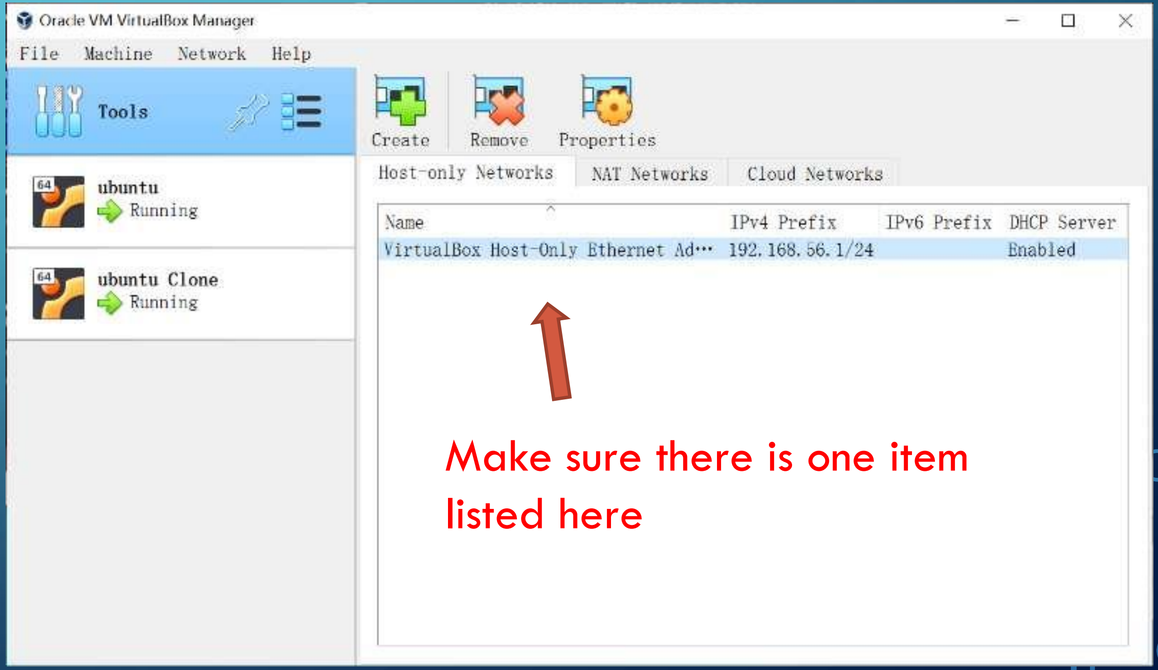
- Prepare two VMs. Each VM should have the followings installed:
  - SSH server
    - `sudo apt install openssh-server`
  - ifconfig
    - `sudo apt install net-tools`

How to install and configure OpenSSH
https://ubuntu.com/server/docs/service-openssh

RUNNING MPI PROGRAMS ON A VM CLUSTER

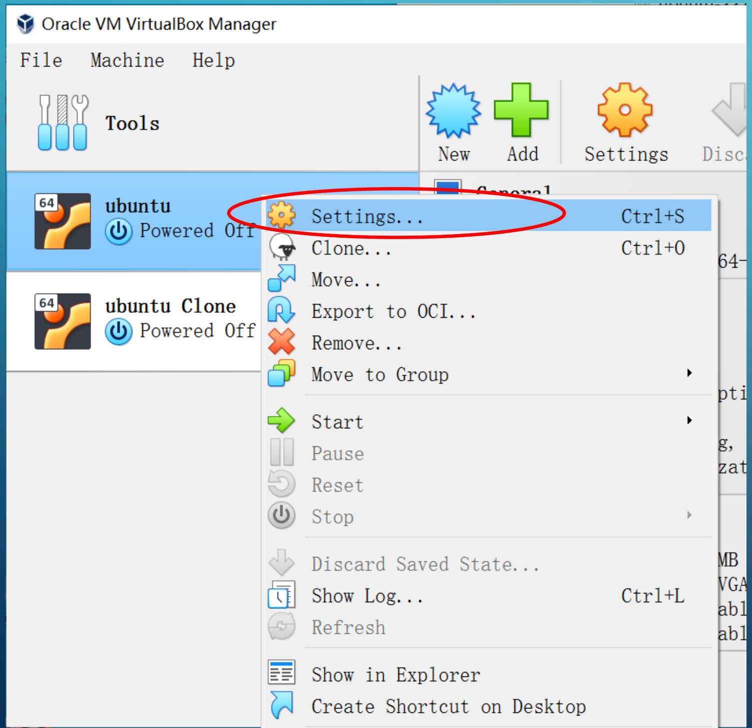
- Make sure there is an existing host-only network adapter
  - If not, create one.

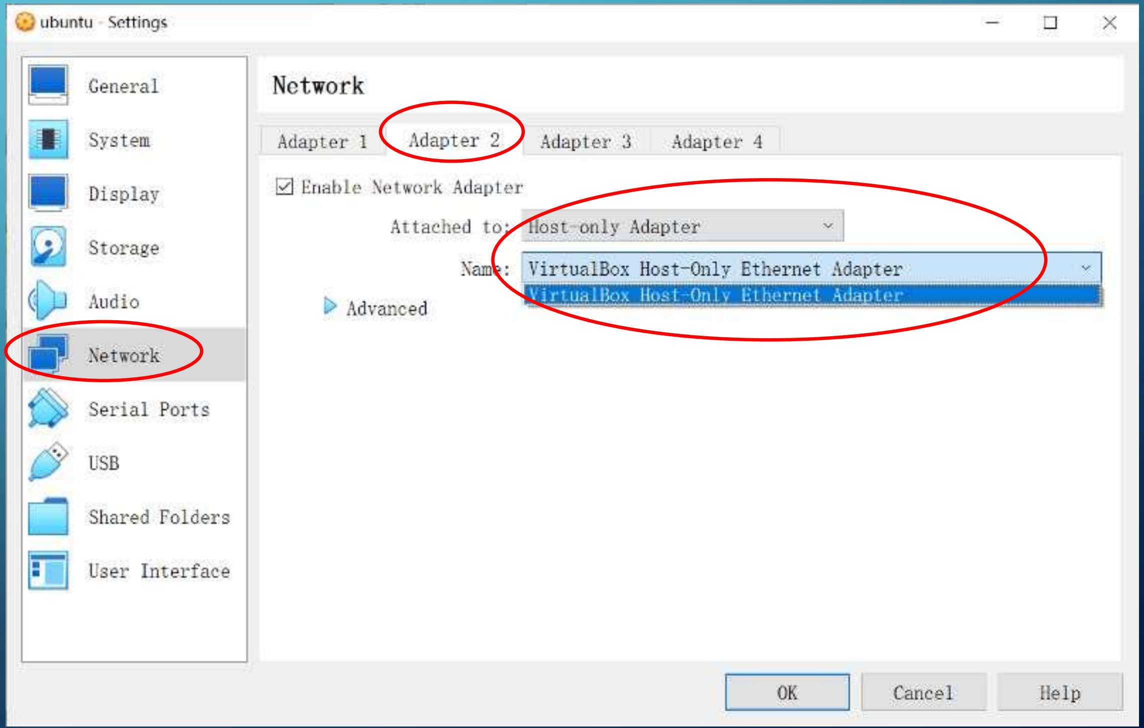




RUNNING MPI PROGRAMS ON A VM CLUSTER

- Shut down you VMs
- Set EACH of your VM to that host-only adapter (see the previous slide)





RUNNING MPI PROGRAMS ON A VM CLUSTER

- On each VM
  - Start the SSH server:
    - `sudo systemctl restart sshd.service`
  - Generate SSH keys
    - `ssh-keygen -t rsa`
  - Use `ifconfig` to check its own IP address
  - Copy the SSH key to the other VM
    - `ssh-copy-id username@remotehostIP`

How to install and configure OpenSSH
https://ubuntu.com/server/docs/service-openssh

RUNNING MPI PROGRAMS ON A VM CLUSTER

- The figures show how to enable password-less SSH from VM2 to VM1.
- You should also set password-less SSH from VM1 to VM2.

On VM1

```
nan@ubuntu:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::d625:cfc:c6ad:b91b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:02:b0:59  txqueuelen 1000  (Ethernet)
        RX packets 2778  bytes 3813532 (3.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1427  bytes 106400 (106.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.101  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::242a:9431:19fd:ed47  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:e0:71:3d  txqueuelen 1000  (Ethernet)
        RX packets 197  bytes 37397 (37.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 195  bytes 33715 (33.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 216  bytes 23876 (23.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 216  bytes 23876 (23.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

On VM2

```
nan@ubuntu:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/nan/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nan/.ssh/id_rsa
Your public key has been saved in /home/nan/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Oc5RN8H4z+8eyFfIawhHnOCX5A3Bm/BvVy715hqLRkg nan@ubuntu
The key's randomart image is:
+---[RSA 3072]----+
|          .+=.   |
|         .o=o*   |
```

```
nan@ubuntu:~$ ssh-copy-id nan@192.168.56.101
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter o
ut any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompte
d now it is to install the new keys
nan@192.168.56.101's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'nan@192.168.56.101'"
and check to make sure that only the key(s) you wanted were added.

nan@ubuntu:~$ ssh nan@192.168.56.101
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.19.0-32-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

267 updates can be applied immediately.
128 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Tue Feb 21 01:38:51 2023 from 192.168.56.102
nan@ubuntu:~$ exit
logout
Connection to 192.168.56.101 closed.
nan@ubuntu:~$
```

SSH without password

RUNNING MPI PROGRAMS ON A VM CLUSTER

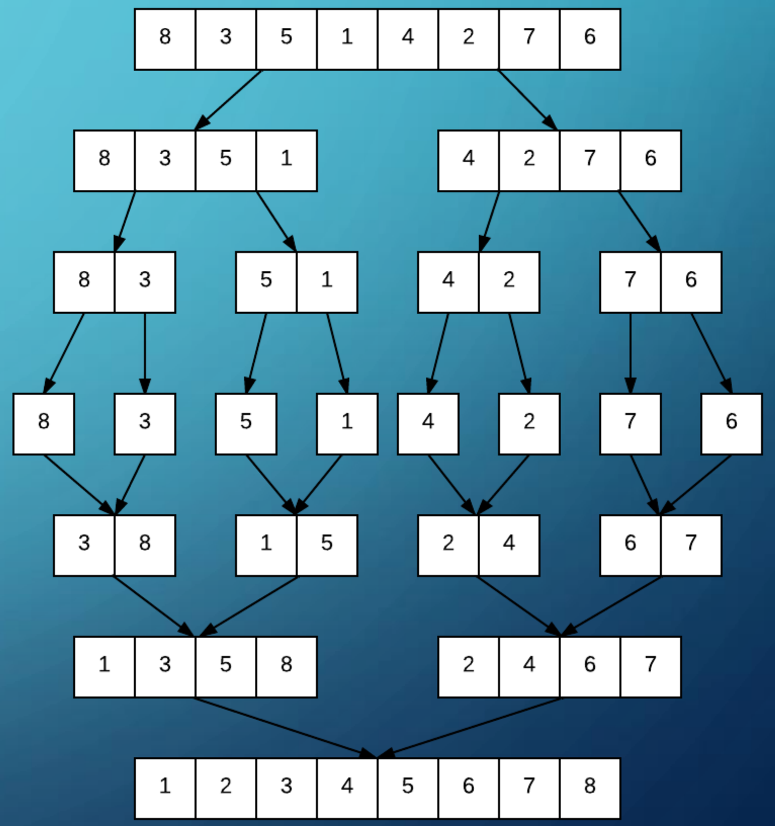
- Compile the MPI program, and copy it to the two VMs
- Run the program by specifying the hosts on either of the VM
  - `mpirun -np 2 --host <IP_of_VM1>,<IP_of_VM2> <your_program>`

https://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/

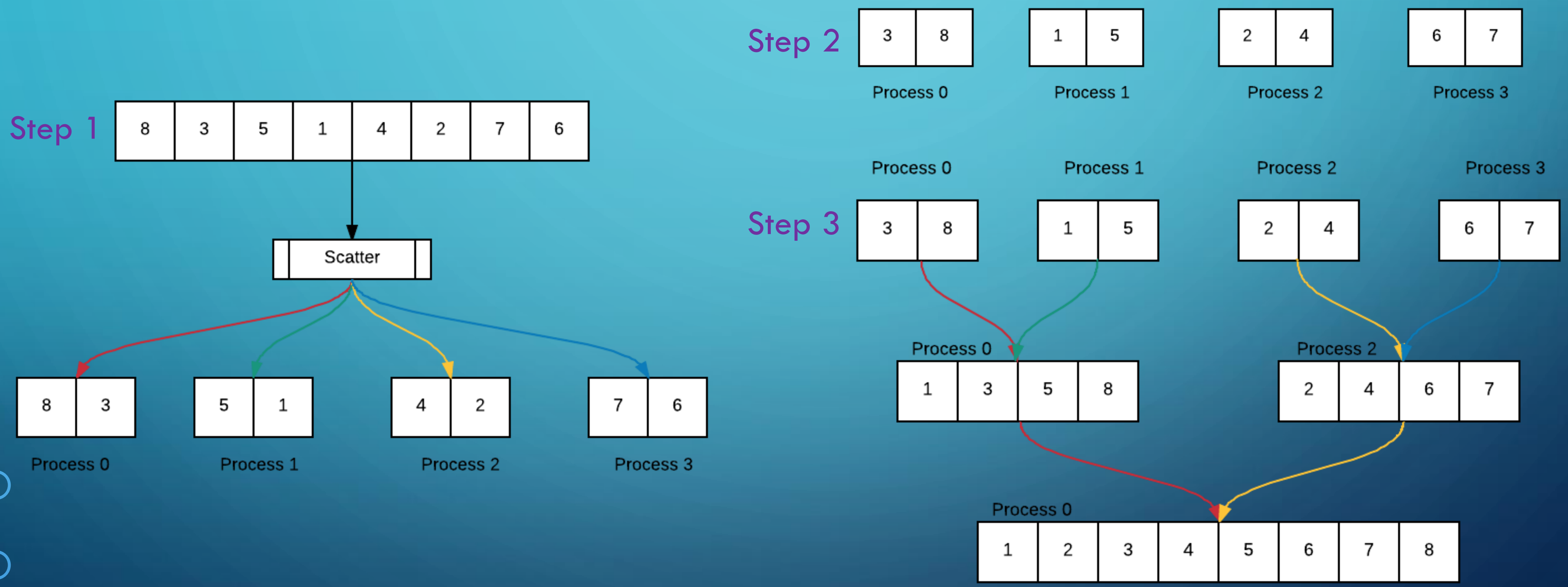
How to install and configure OpenSSH
https://ubuntu.com/server/docs/service-openssh

PARALLELIZE COMPUTATION – MERGE SORT

- Merge sort is a classic divide-and-conquer sorting algorithm
- Process: Divide list into unsorted sub-lists, then sort sub-lists, finally merge all sorted sub-lists.
- Good candidate for parallelize: Sorting of sub-lists are independent!



PARALLELIZE COMPUTATION – MERGE SORT



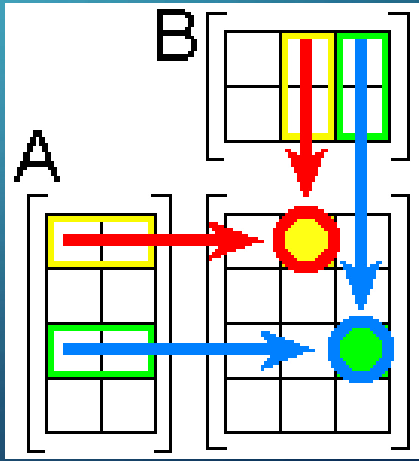
See more: http://selkie-macalester.org/csinparallel/modules/MPIProgramming/build/html/mergeSort/mergeSort.html

TASKS TODAY

- Set up the VM cluster
- Run the ping-pong program on the VM cluster

PRACTICE: MATRIX MULTIPLICATION

- Rule for matrix multiplication $(AB)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$
- Think about how to parallelize the computation process, and write an MPI application to do this.



- Can you parallelize the computation and also minimize the memory usage?
- Useful link: https://mpitutorial.com/tutorials/