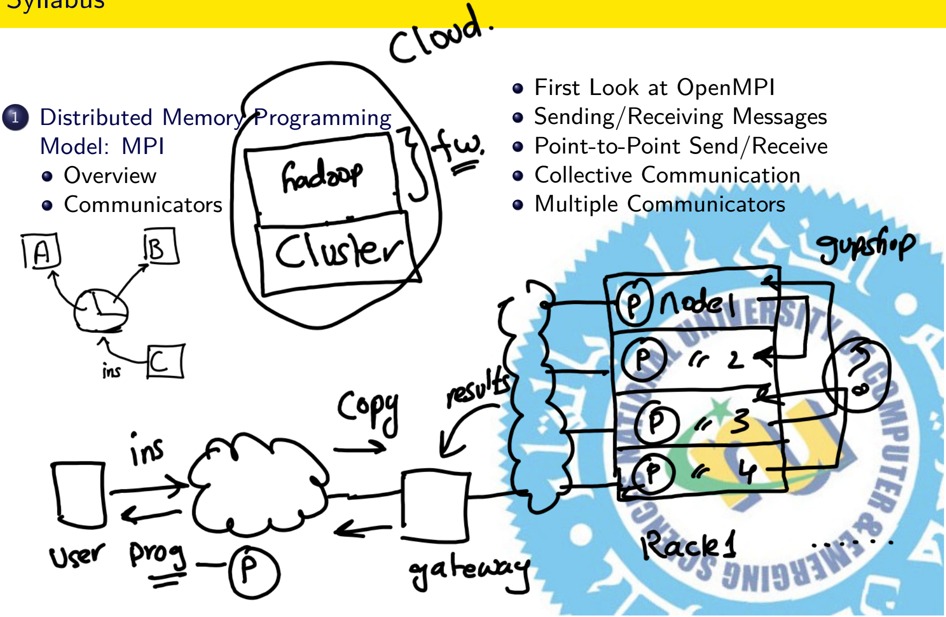
Syllabus

1. Distributed Memory Programming Model: MPI
   - Overview
   - Communicators
   - First Look at OpenMPI
   - Sending/Receiving Messages
   - Point-to-Point Send/Receive
   - Collective Communication
   - Multiple Communicators

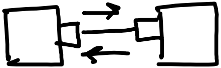


Single cpu ✓

multiple cpus (cluster) ✓

Socket programming.

Berkeley → ZeroMQ

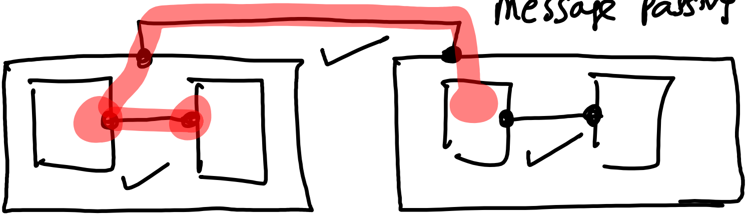


hidden

1. Distributed Memory Programming Model: MPI
   - Overview
   - Communicators
   - First Look at OpenMPI
   - Sending/Receiving Messages
   - Point-to-Point Send/Receive
   - Collective Communication
   - Multiple Communicators

prog. model.

message passing.



single process → copies/duplicates

1. Open MPI ✓

different processes

2. Socket Programming ✓

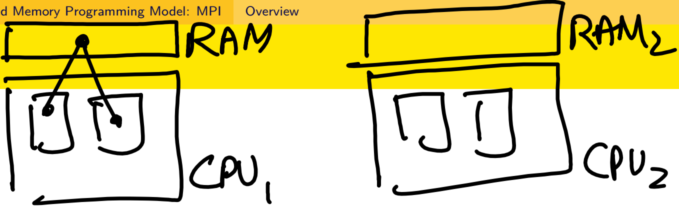
3. Remote Procedure Calls

single process

" RPC ✗

Method Invocation ✓

MPI Overview

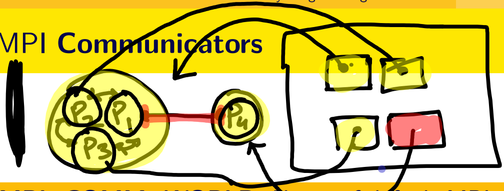


- **M**essage **P**assing **I**nterface
- MPI Can be used for Shared Memory, as well as Distributed Memory architectures (Hybrid, if requiredi)
- Supported by Fortran, C, C++ (but modules also available for python, & Java)
- Hides hardware details of underlying system (so portable)
- Many high performance libraries have MPI versions of API calls
- MPI version 3.0 specification has 400+ commands (function calls). Knowledge of only 11-12 of them can help you do the job in more than 90% of cases.

OpenMP.

OpenMPI

MPI Communicators



$P_2 \longleftrightarrow P_3$

$P_1 \longleftrightarrow P_2$

$P_1 \longleftrightarrow P_3$

MPI_COMM_WORLD: Name of default MPI Communicator

- A communication universe (communication domain, communication group) for a group of processes
- Stored in variables of type **MPI_COMM**
- Communicators are used as arguments to all message transfer MPI routines
- Each process within communicator has a **rank**; a unique integer identifier ranging between $[0, \#processors - 1]$
- Multiple communicators can be established in a single MPI program
- **Intra-Communicator**: Used for communication within a single group
- **Inter-Communicator**: Used for communication between two disjoint groups

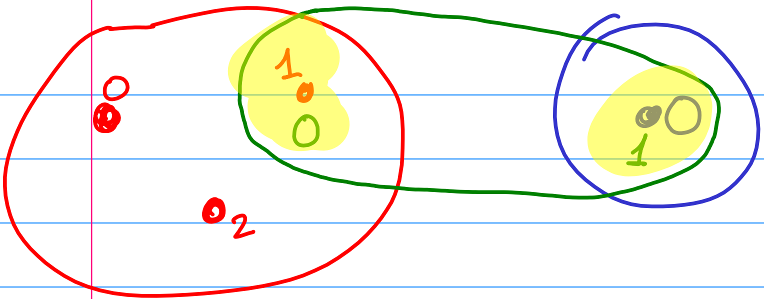
$P_1 \longleftrightarrow P_4$

global Communicator.

OpenMP → { { { for each thread omp_get_thread_id()

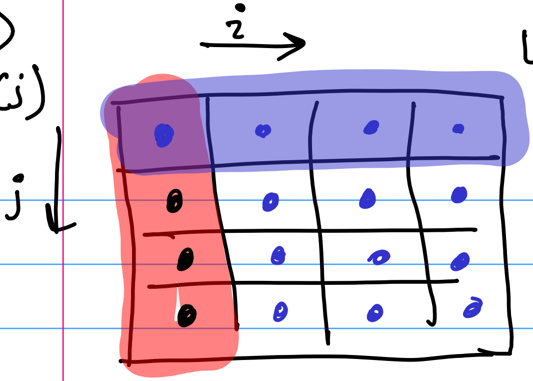
0 2 1

id



MPI id → Rank

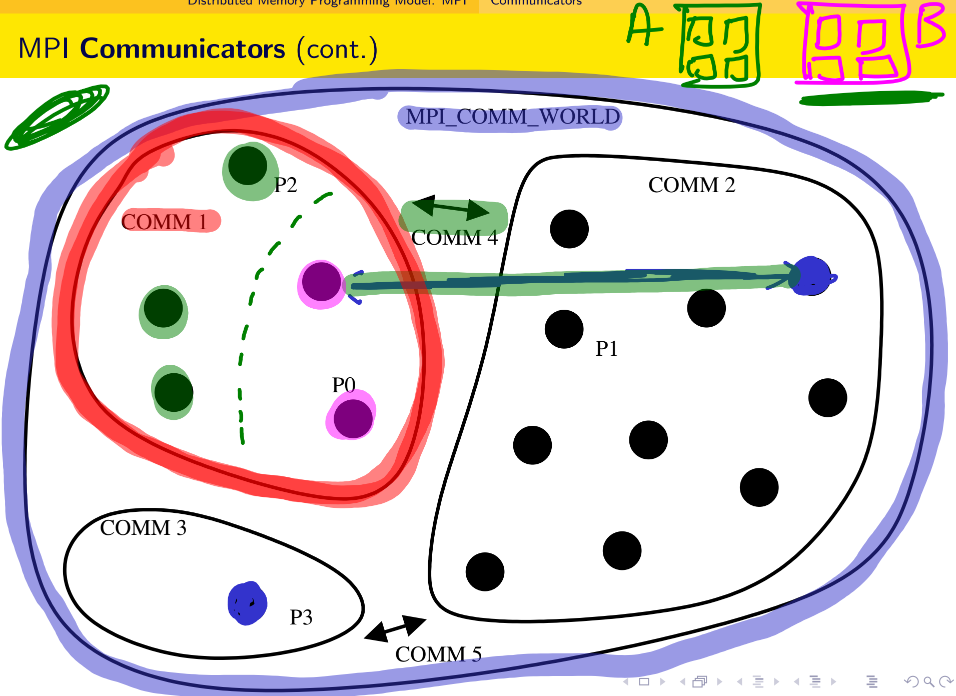
```
for(i)
    for(j)
```

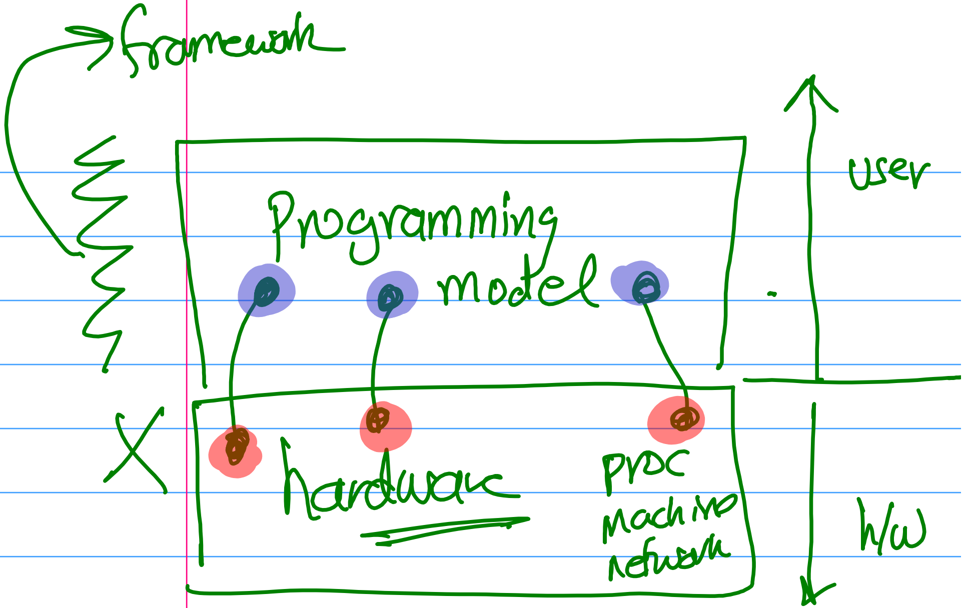


4 x 4

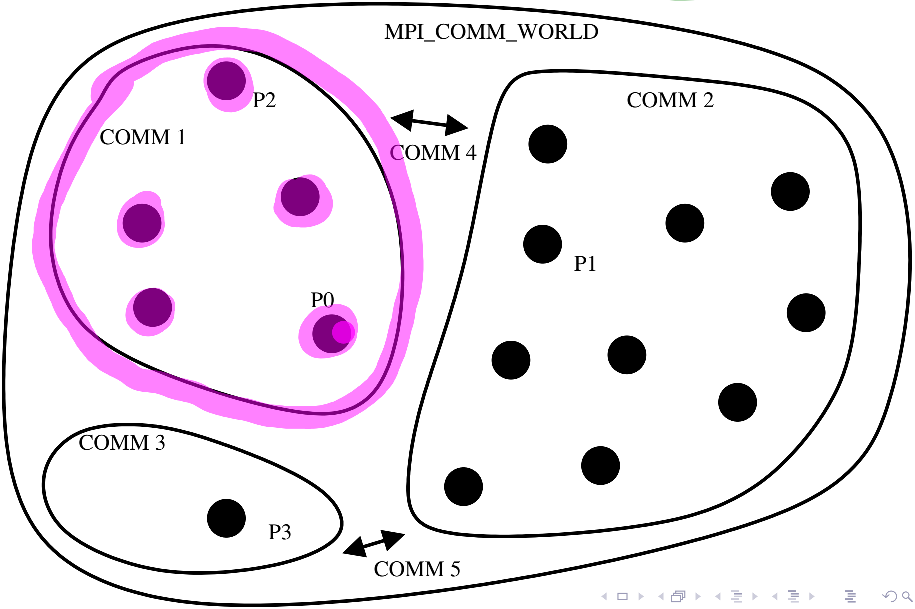
16 distributed threads

MPI **Communicators** (cont.)




framework
Programming model
user
X
hardware
proc
machine
network
h/w

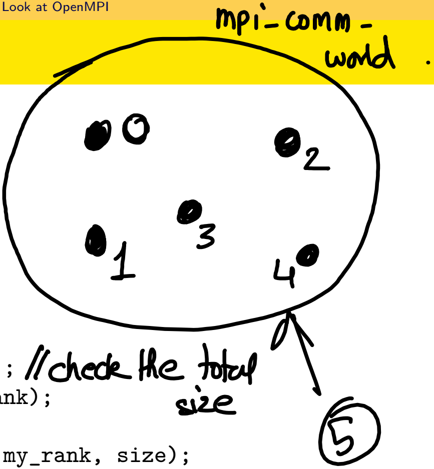
MPI **Communicators** (cont.)



First Look (hellompi.c)

~~Set the total size~~

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

int main(int argc, char **argv)
{
    int size, my_rank;

1   MPI_Init(&argc, &argv);
2   MPI_Comm_size(MPI_COMM_WORLD, &size); //check the total size
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    printf("Hello from %d out of %d\n", my_rank, size);

    MPI_Finalize();

    return 0;
}
mpicc hellompi.c # Compilation (mpiCC for C++, also gcc hellompi.c -lmpi)
mpirun -np 4 -hostfile filename a.out     # Execution
```



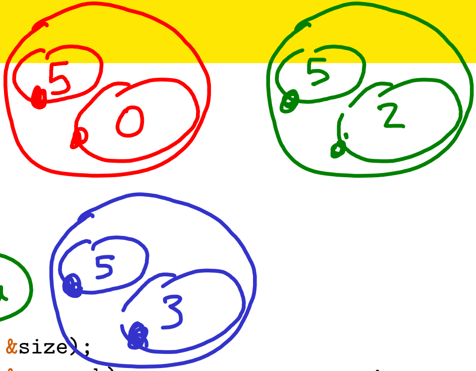
First Look (hellompi.c)

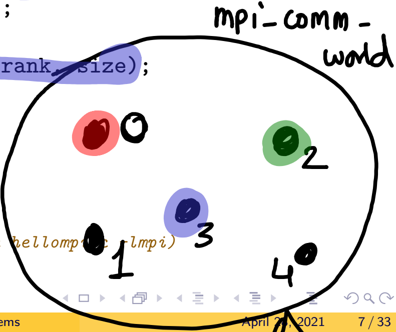
```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
    int size, my_rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    printf("Hello from %d out of %d\n", my_rank, size);

    MPI_Finalize();

    return 0;
}
```

initia

destroy

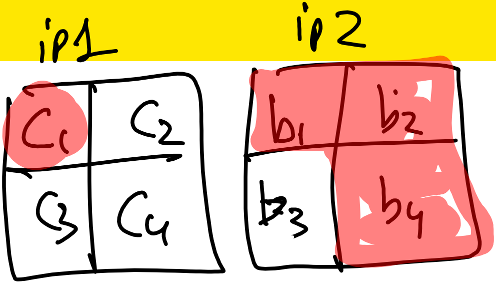
```
mpicc hellompi.c # Compilation (mpiCC for C++, also gcc hellompi.c -lmpi)
mpirun -np 4 -hostfile filename a.out      # Execution
```





First Look (hellompi.c)

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

int main(int argc, char **argv)
{
    int size, my_rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    printf("Hello from %d out of %d\n", my_rank, size);

    MPI_Finalize();

    return 0;
}
```

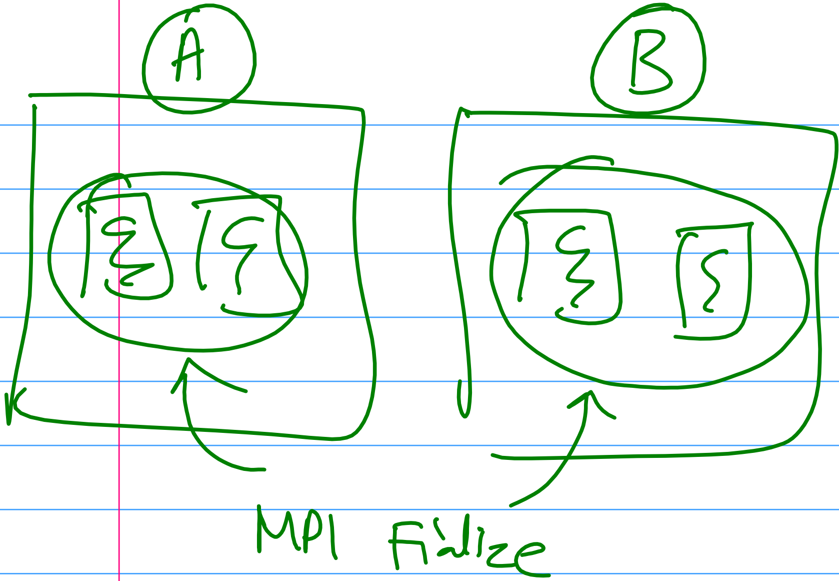


ip1 1
ip2 3

other 4 activities.

C2, C3, C4, b3

```
mpicc hellompi.c # Compilation (mpiCC for C++, also gcc hellompi.c -lmpi)
mpirun -np 4 -hostfile filename a.out      # Execution
```

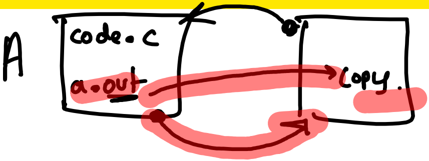


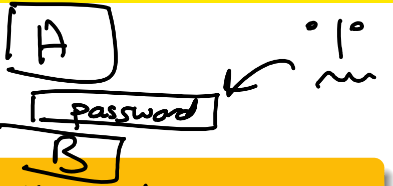
A
B
MPI Finalize

First Look (hellompi.c)

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

int main(int argc, char **argv)
{
    int size, my_rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    printf("Hello from %d out of %d\n", my_rank, size);

    MPI_Finalize();

    return 0;
}
mpicc hellompi.c # Compilation (mpiCC for C++, also gcc hellompi.c -lmpi)
mpirun -np 4 -hostfile filename a.out      # Execution
```

Configuring a Simple MPI based Distributed Computing Cluster





Requirements

- SSH Server

```
apt-get install openssh-server
```

- OpenMPI Library

```
apt-get install openmpi-bin openmpi-doc libopenmpi-dev
```

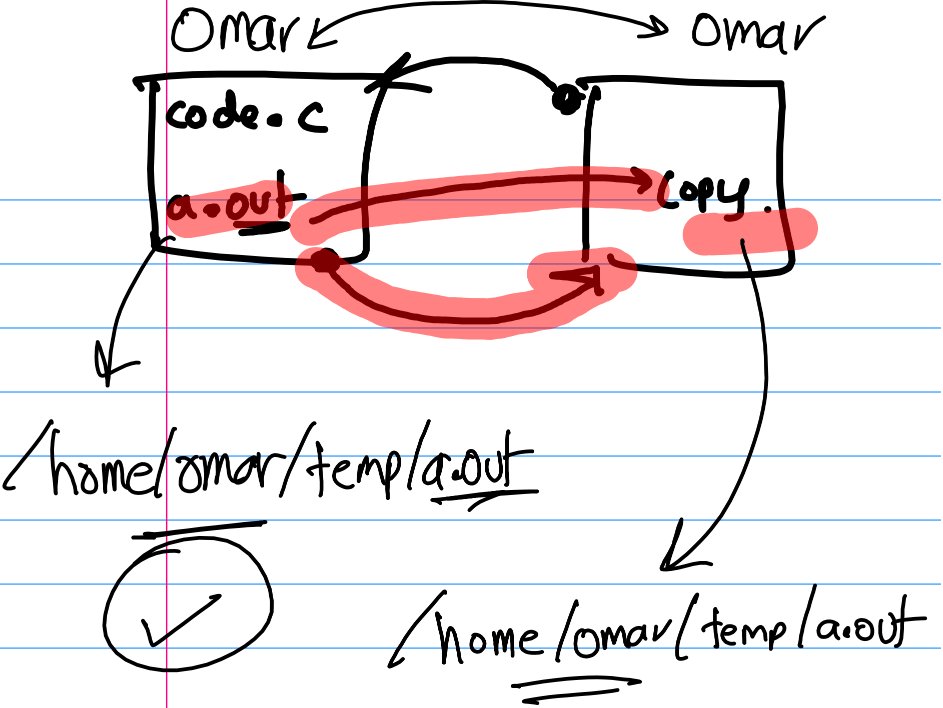
- NFS Network File System

```
apt-get install nfs-server nfs-client
```

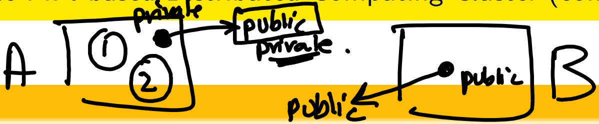
authentication SSH
password less

OpenMPI

mpichv2


Omar
omar
code.c
a.out
Copy.
/home/omar/temp/a.out
/home/omar/temp/a.out
✓

Configuring a Simple MPI based Distributed Computing Cluster (cont.)



Passwordless Login

- Generate a public-private key pair for your user name. Do not specify a pass-phrase when asked

  ```
  ssh-keygen
  ```
  ①

- Copy the public part of your key to the remote server

  ```
  ssh-copy-id <ip of remote computer>
  ```
  ②

- If the above command does not work, continue with these commands:
  - Copy the public key to the remote computer. For e.g., if the username is omar, the remote ip address is 1.2.3.4, then:

    ```
    scp /home/omar/.ssh/id_rsa.pub omar@1.2.3.4:/home/omar
    ```
    ☆

  - Login to the remote computer using your username

    ```
    ssh 1.2.3.4 -l omar
    ```

  - Add the public key to authorized keys

    ```
    cat /home/omar/id_rsa.pub >> /home/omar/.ssh/authorized_keys
    ```
    ☆

Configuring a Simple MPI based Distributed Computing Cluster (cont.)

NFS

Dropbox

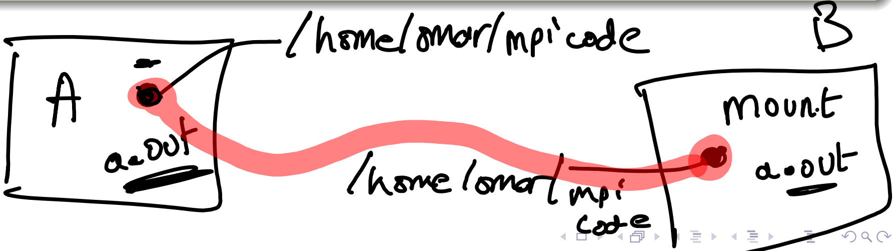
Manual copy

Transfering Files

- There are many ways to transfer files. You can setup an **NFS** mountpoint, share files using dropbox, or send files using scp. The scp method is given below:

```
scp /location/of/a.out username@ipaddress:/home/username/a.out
```

- **Note: All cluster nodes must be able to find the executable file at the same location as any other cluster node**


/home/omar/mpi'code
B
A
a.out
mount
a.out
/home/omar/mpi code