COL380

Introduction to
Parallel & Distributed Programming

Programming Models

- Shared Memory model
- Distributed Memory model
- Task based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

Subodh Kumar

Example Models

```
void *threadFn
{
    // Do somet
}
…
{
    int arg;
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, threa
    // Continue doing other things
    pthread_join(thread_id, NULL);
}
```

```
int threadFn(int arg)
{
    // Do something with ‘arg’
}
…
{
    int x;

    x = spawn threadFn(1);
    // Continue doing other th
    sync;
}
```

```
func threadFn(id int, arg string, wg *sync.WaitGroup,
                                          ch chan int)
{
    defer wg.Done()
    // Do thing ‘id’ with ‘arg’
                   x <- ch
}
…
{
    var wg sync.WaitGroup
                   chA := make(chan int)
    wg.Add(1)
    go threadFn(1, “a string”, &wg, chA)
    // Continue doing other things
    wg.Wait()
                   chA <- result
}
```

Subodh Kumar

Message Passing

- Communication network (and infrastructure)
  - ➡ Ethernet, Infiniband, Custom-made
- Processor-local memory
- Access to other threads' data through explicit instructions
  - ➡ Implicit synchronization semantics
- Can double as inter-process synchronization

Shared Memory

<u>Multiple Threads of Execution</u>

OP operands	OP operands
OP operands	OP operands
OP operands	OP operands
OP operands	OP operands
OP operands	OP operands
OP operands	OP operands
OP operands	OP operands
OP operands	OP operands

Shared Variable Store

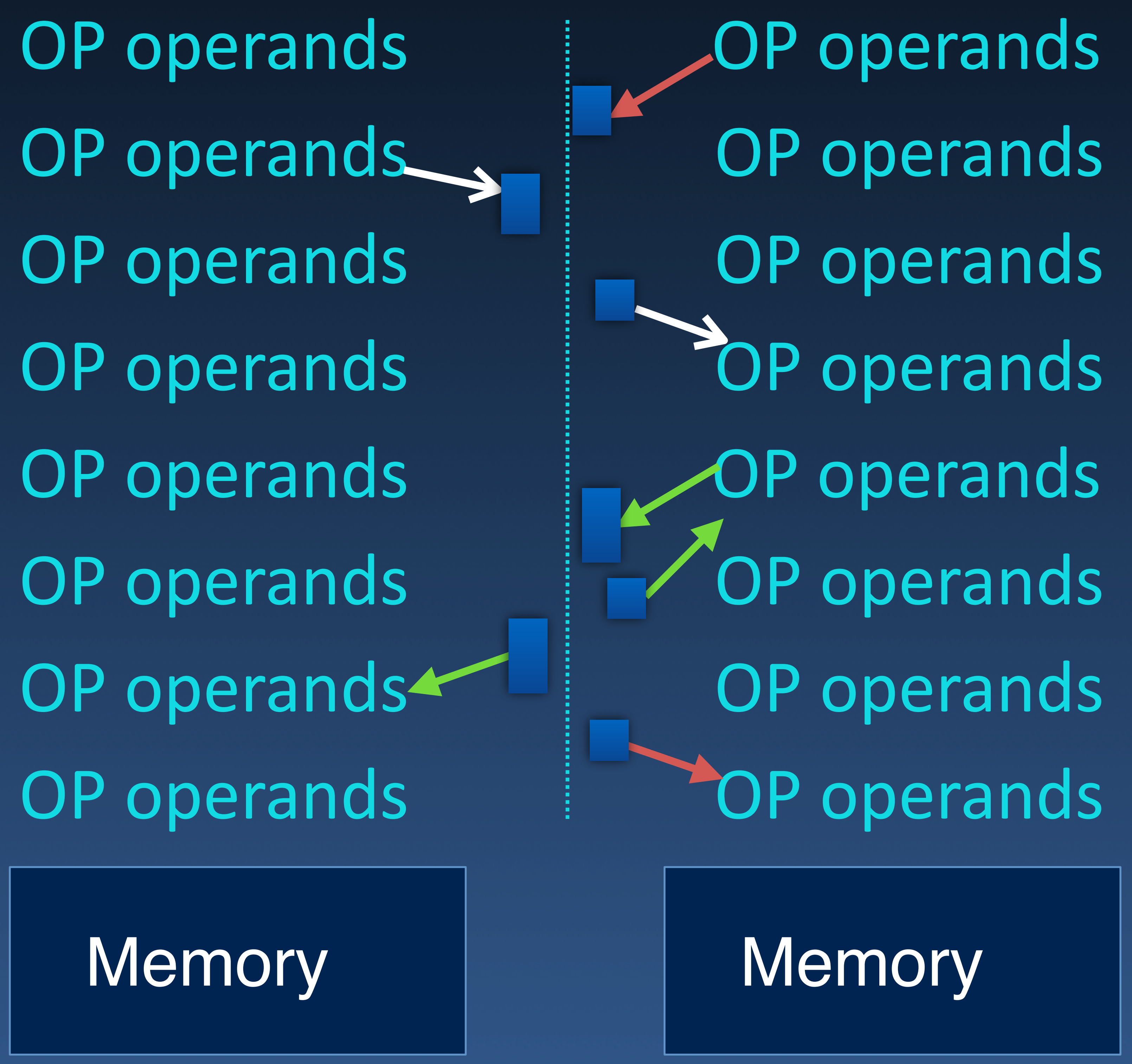
(May not share actual memory)

- How are they instantiated?
- Execution and memory model
- How do the interact?

Separate from HW
(Who does the execution?)

Message Passing

Multiple Threads of Execution



(Could share actual memory)

- How are they instantiated?
- Execution and memory model
- How do the interact?

Separate from HW
(Who does the execution?)

Subodh Kumar

- Multiple "threads" of execution
  - Do not share address space (Processes)
  - Each process may further have multiple threads of control that share memory with each other

Shared Memory Model	Message Passing Model
Read Input Create Sharing threads: Process(sharedInput, myID)	Read Input Create Remote Processes Loop: Send data to each process Wait and collect results

: Recv data
Process(data)
Send results

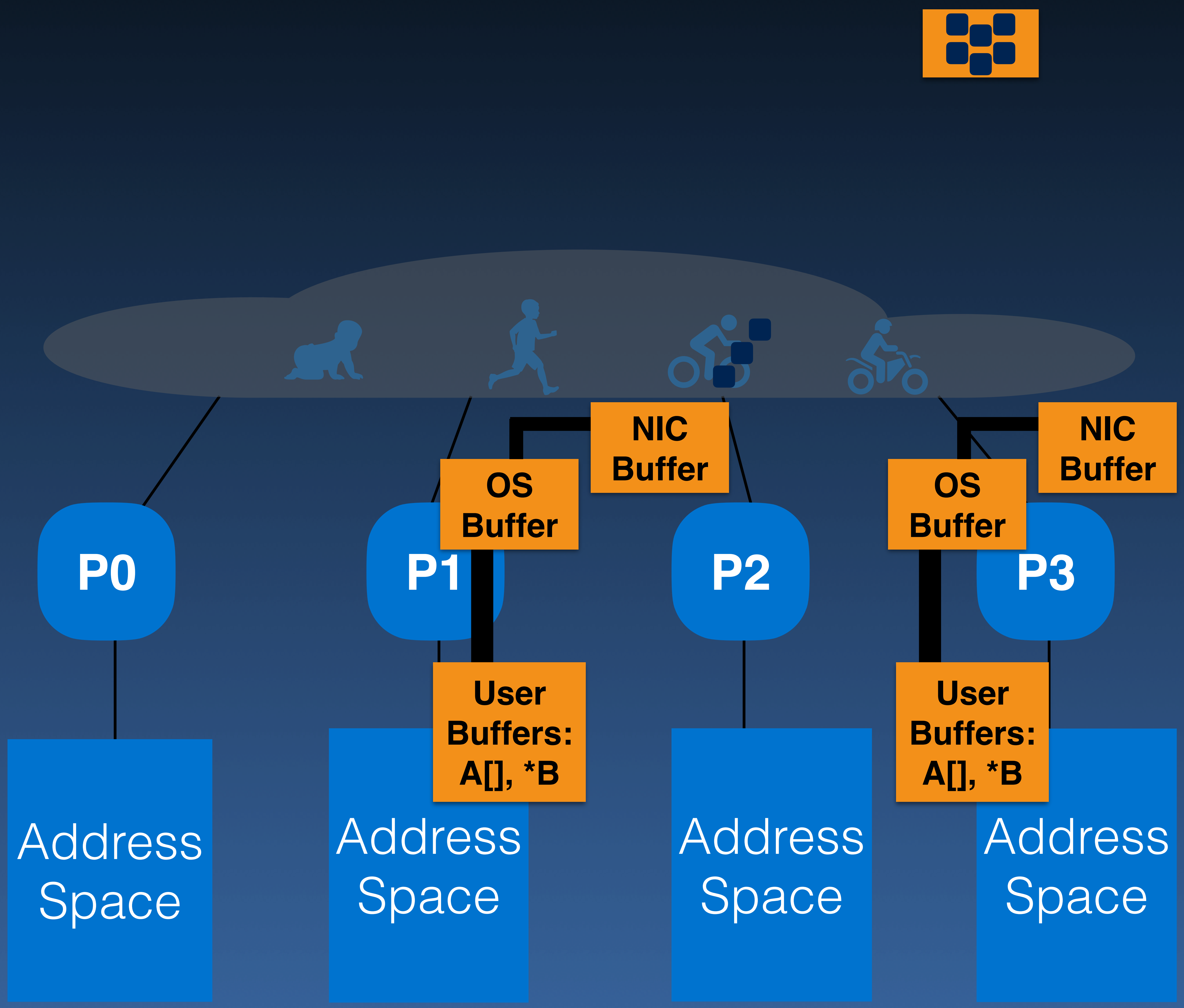
+/- of Shared Memory

+ Easier to program with global address space
+ Typically fast memory access
    - ➡ (when hardware supported)
- Hard to scale
    - Adding CPUs (geometrically) increases traffic
- Programmer initiated synchronization of memory accesses

+/- of Message Passing

- \+ Memory is scalable with number of processors
- \+ Local access is fast (no cache coherency overhead)
- \+ Cost effective, with off-the-shelf processor/ network
- \- Programs often looks more complex
- \- Data communication needs to be managed

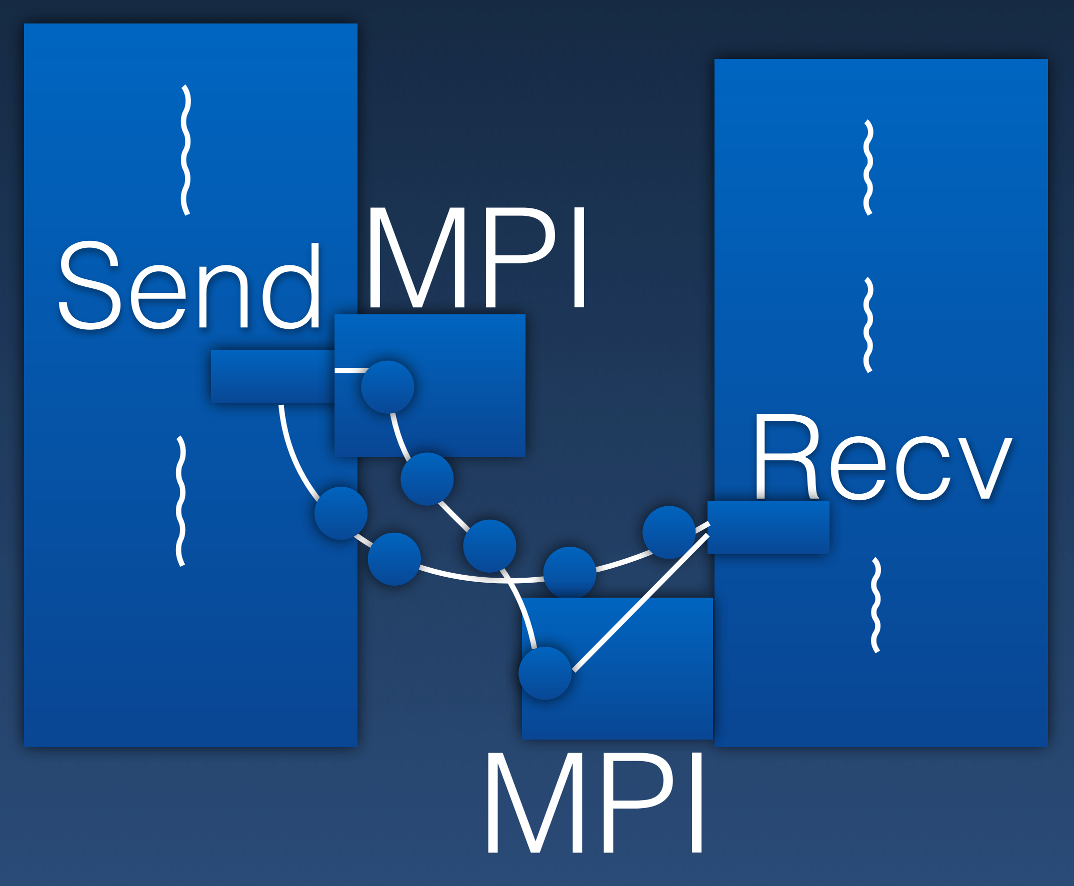
Distributed



Subodh Kumar

Message Semantics

- Variables, Buffers, and Packets
  - ➡ Application to application
- Lossy?
  - ➡ Deal with loss
  - ➡ Acks
- FIFO?
- Point to Point vs Collective?
- Addressing?



MPI

- MPI is for inter-process communication
  - ➡ Process creation
  - ➡ Data communication (Buffering, Book-keeping ..)
  - ➡ Synchronization
- Allows
  - ➡ Synchronous communication
  - ➡ Asynchronous communication
    - compare to shared memory

Functions, Types, Constants

- High-level constructs
  - broadcast, reduce, scatter/gather message
  - Collective functions
- Interoperable across architectures

Running MPI Programs

- **Compile:** mpiCC -o exec code.cpp
  - script to compile and link
  - Automatically adds include, library flags
- **Run:**
  - mpirun -host host1,host2 exec args
  - Or, use hostfile
- **Useful:**
  - mpirun -mca <key> <value>

- mpirun -mca mpi_show_handle_leaks 1
- mpirun -mca btl openib,tcp
- mpirun -mca btl_tcp_min_rdma_size
- Check out “ompi_info”

- Key based remote shell execution
- Use ssh-keygen to create public-private key pair
  - ➡ Private key stays in subdirectory ~/.ssh on your client
  - ➡ Public key on server in ~/.ssh/authorized_keys
  - ➡ Test: ‘ssh <server> ls’ works
  - ➡ On HPC, client and server share the same home directory
- PBS automatically creates appropriate host files
  - ➡ See also: -l select=2:ncpus=1:mpiprocs=1 -l place=scatter