

Dynamic Processes: Spawn

Dynamic Processes

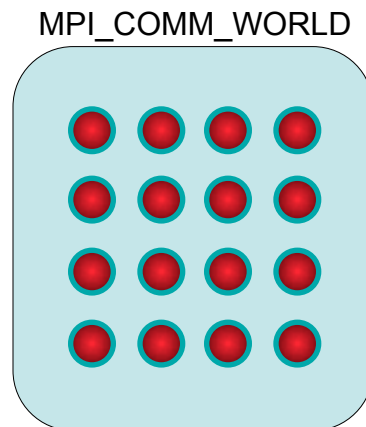
- Adding processes to a running job
 - As part of the algorithm i.e. branch and bound
 - When additional resources become available
 - Some master-slave codes where the master is started first and asks the environment how many processes it can create
- Joining separately started applications
 - Client-server or peer-to-peer
- Handling faults/failures

MPI-1 Processes

- All process groups are derived from the membership of the MPI_COMM_WORLD
 - No external processes
- Process membership static (vs. PVM)
 - Simplified consistency reasoning
 - Fast communication (fixed addressing) even across complex topologies
 - Interfaces well to many parallel run-time systems

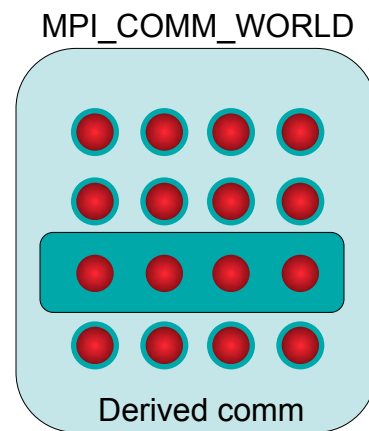
Static MPI-1 Job

- MPI_COMM_WORLD
- Contains 16 processes



Static MPI-1 Job

- MPI_COMM_WORLD
- Contains 16 processes
- Can only subset the original MPI_COMM_WORLD
 - No external processes



Disadvantages of Static Model

- Cannot add processes
 - Cannot remove processes
 - If a process fails or otherwise disappears, all communicators it belongs to become invalid
- ➔ Fault tolerance undefined

MPI-2

- Added support for dynamic processes
 - Creation of new processes on the fly
 - Connecting previously existing processes
- Does not standardize inter-implementation communication
 - Interoperable MPI (IMPI) created for this

Open Questions

How do you add more processes to an already-running MPI-1 job?

- How would you handle a process failure?
- How could you establish MPI communication between two independently initiated, simultaneously running MPI jobs?

MPI-2 Process Management

- MPI-2 provides “spawn” functionality
 - Launches a child MPI job from a parent MPI job
- Some MPI implementations support this
 - Open MPI
 - LAM/MPI
 - NEC MPI
 - Sun MPI
- High complexity: how to start the new MPI applications ?

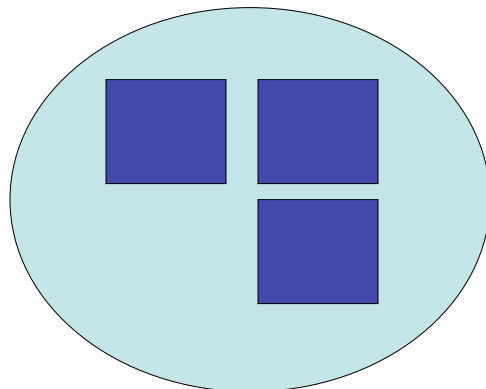
MPI-2 Spawn Functions

- MPI_COMM_SPAWN
 - Starts a set of new processes with the same command line
 - Single Process Multiple Data
- MPI_COMM_SPAWN_MULTIPLE
 - Starts a set of new processes with potentially different command lines
 - Different executables and / or different arguments
 - Multiple Processes Multiple Data

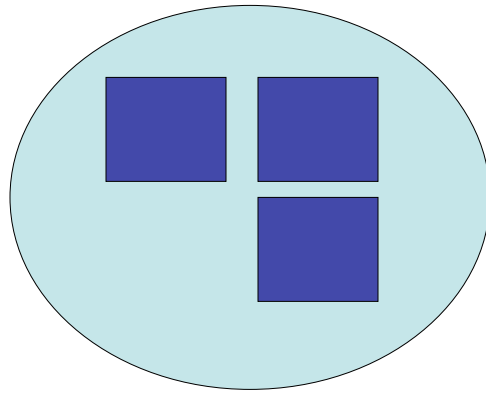
Spawn Semantics

- Group of parents collectively call spawn
 - Launches a new set of children processes
 - Children processes become an MPI job
 - An **inter**communicator is created between parents and children
- Parents and children can then use MPI functions to pass messages
- MPI_UNIVERSE_SIZE

Spawn Example

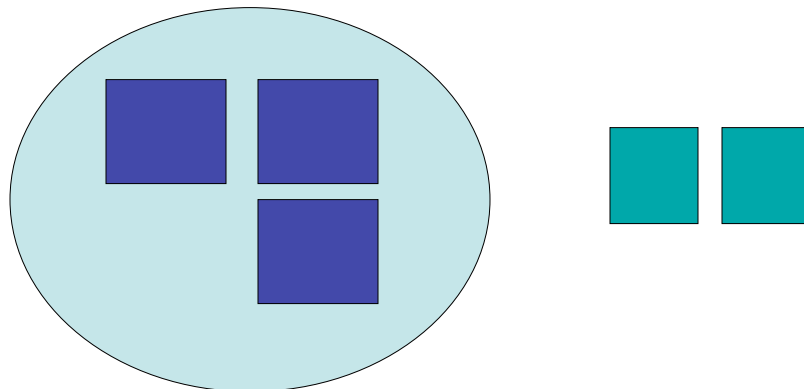


Spawn Example



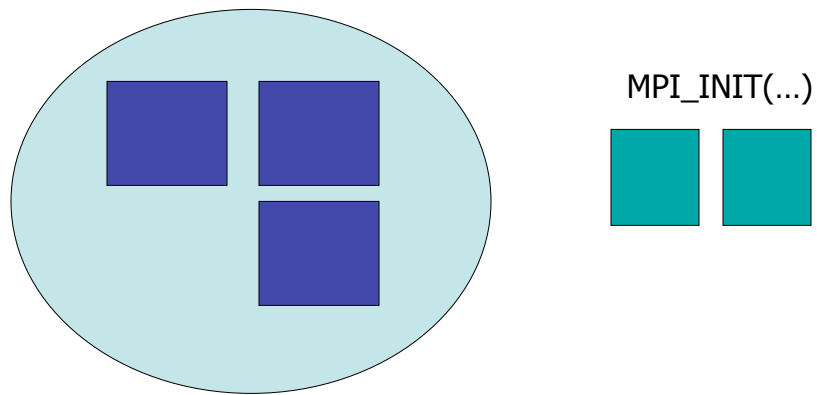
Parents call MPI_COMM_SPAWN

Spawn Example



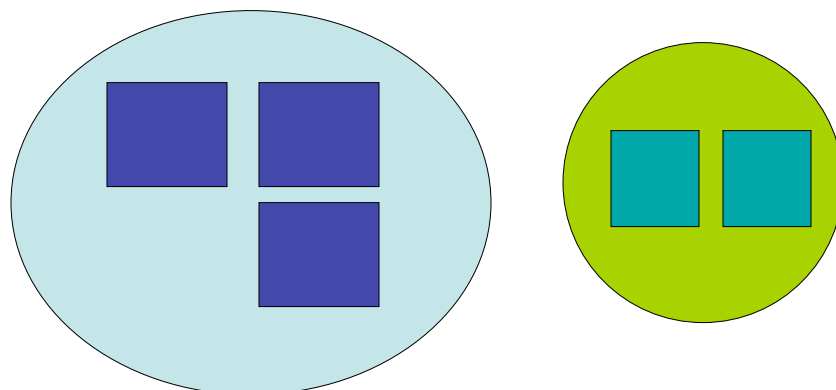
Two processes are launched

Spawn Example



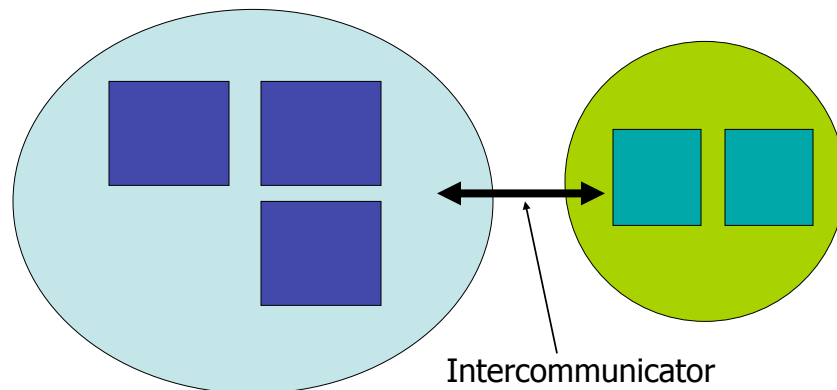
Children processes call MPI_INIT

Spawn Example



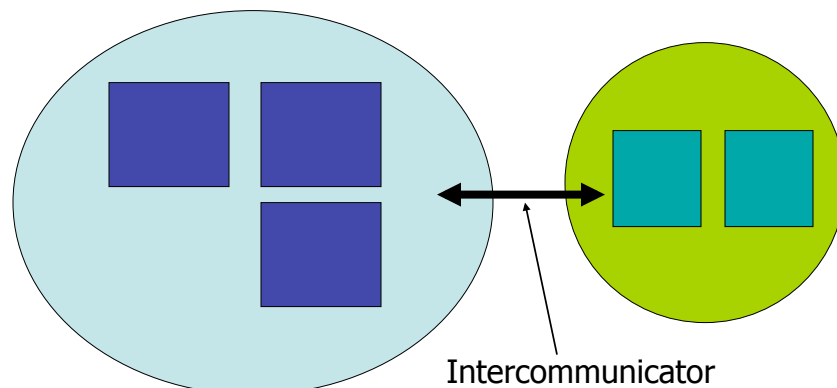
Children create their own MPI_COMM_WORLD

Spawn Example



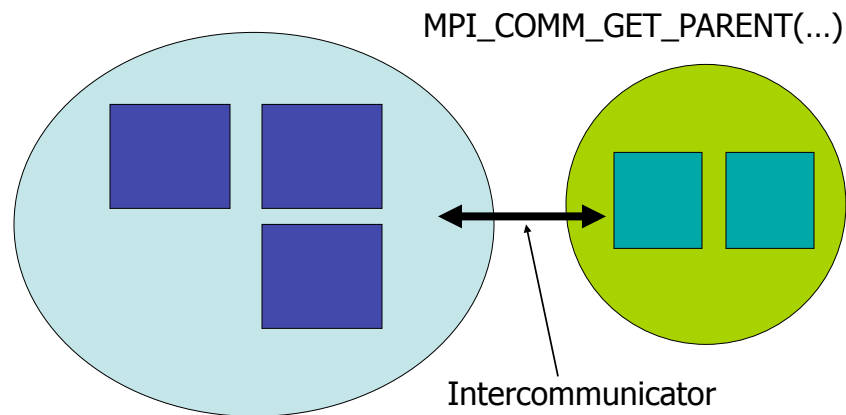
An intercommunicator is formed between parents and children

Spawn Example



Intercommunicator is returned from `MPI_COMM_SPAWN`

Spawn Example



Children call MPI_COMM_GET_PARENT to get intercommunicator

Master / Slave Demonstration

- Simple 'PVM' style example
 - User starts singleton master process
 - Master process spawns slaves
 - Master and slaves exchange data, do work
 - Master gathers results
 - Master displays results
 - All processes shut down

Master / Slave Demonstration

Master program

```

MPI_Init(...)
MPI_Spawn(..., slave, ...);

for (i=0; i < size; i++)
    MPI_Send(work, ..., i, ...);
for (i=0; i < size; i++)
    MPI_Recv(presults, ...);
calc_and_display_result(...)
MPI_Finalize()

```

Slave program

```

MPI_Init(...)
MPI_Comm_get_parent
(&intercomm)
MPI_Recv(work,...,
intercomm)
result = do_something
(work)
MPI_Send(result,...,
intercomm)
MPI_Finalize()

```

MPI “Connected”

- “Two processes are connected if there is a communication path directly or indirectly between them.”
 - E.g., belong to the same communicator
 - Parents and children from SPAWN are connected
- Connectivity is transitive
 - If A is connected to B, and B is connected to C
 - A is connected to C

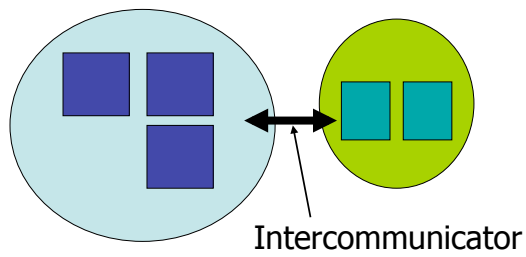
MPI “Connected”

- Why does “connected” matter?
 - MPI_FINALIZE is collective over set of connected processes
 - MPI_ABORT *may* abort all connected processes
- How to disconnect?
 - ...stay tuned

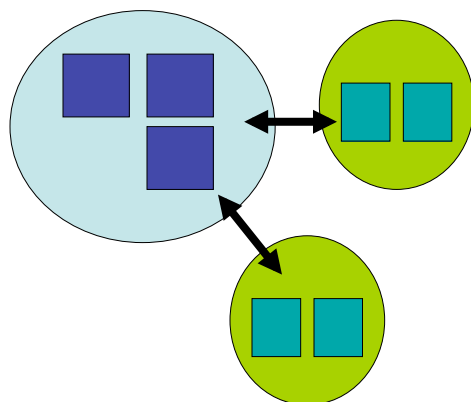
Multi-Stage Spawning

- What about multiple spawns?
 - Can sibling children jobs communicate directly?
 - Or do they have to communicate through a common parent?
- ➔ Is all MPI dynamic process communication hierarchical in nature?

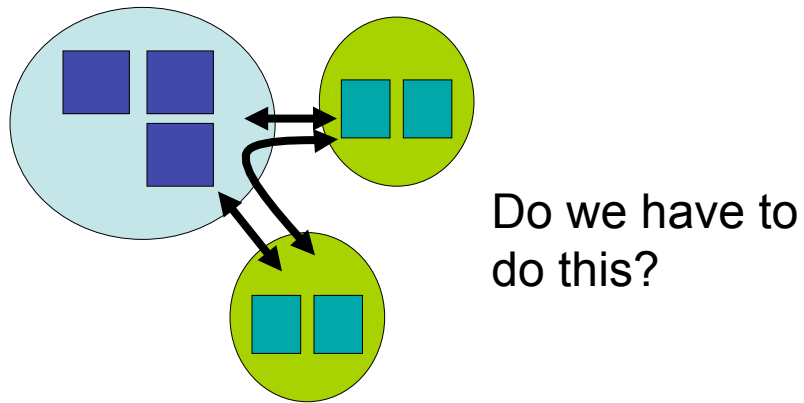
Multi-Stage Spawning



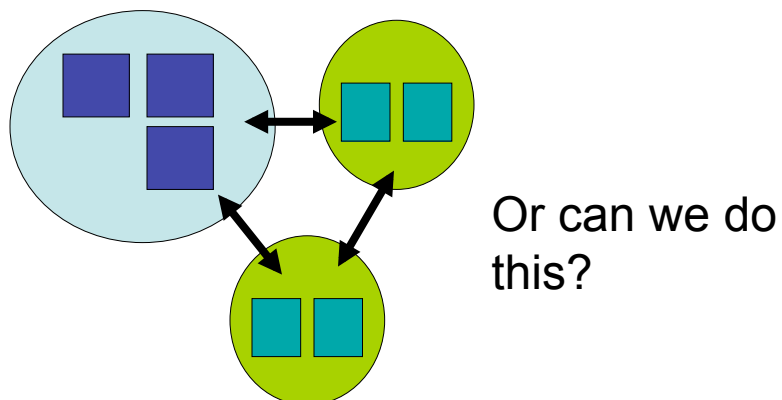
Multi-Stage Spawning



Multi-Stage Spawning



Multi-Stage Spawning



Dynamic Processes: Connect / Accept

Establishing Communications

- MPI-2 has a TCP socket style abstraction
 - Process can accept and connect connections from other processes
 - Client-server interface
- MPI_COMM_CONNECT
- MPI_COMM_ACCEPT

Establishing Communications

- How does the client find the server?
 - With TCP sockets, use IP address and port
 - What to use with MPI?
- Use the MPI name service
 - Server opens an MPI “port”
 - Server assigns a public “name” to that port
 - Client looks up the public name
 - Client gets port from the public name
 - Client connects to the port

Server Side

- Open and close a port
 - `MPI_OPEN_PORT(info, port_name)`
 - `MPI_CLOSE_PORT(port_name)`
- Publish the port name
 - `MPI_PUBLISH_NAME(service_name, info, port_name)`
 - `MPI_UNPUBLISH_NAME(service_name, info, port_name)`

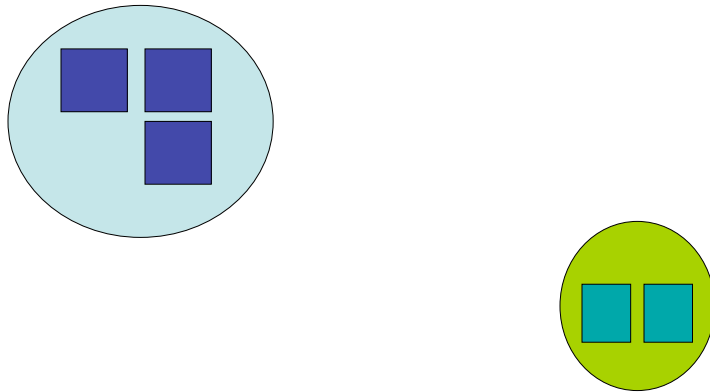
Server Side

- Accept an incoming connection
 - MPI_COMM_ACCEPT(port_name, info, root, comm, newcomm)
 - comm is a **intra**communicator; local group
 - newcomm is an **inter**communicator; both groups

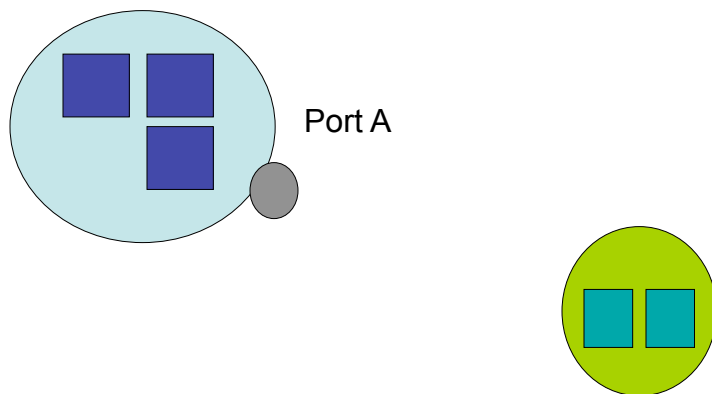
Client Side

- Lookup port name
 - MPI_LOOKUP_NAME(service_name, info, port_name)
- Connect to the port
 - MPI_COMM_CONNECT(port_name, info, root, comm, newcomm)
 - comm is a **intra**communicator; local group
 - newcomm is an **inter**communicator; both groups

Connect / Accept Example

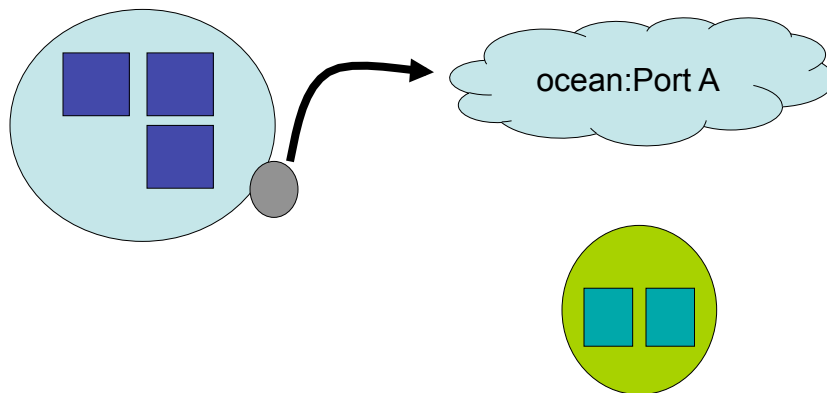


Connect / Accept Example



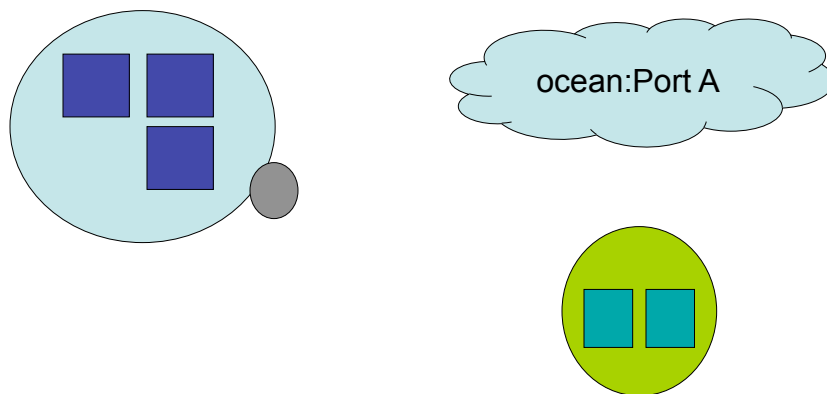
Server calls MPI_OPEN_PORT

Connect / Accept Example



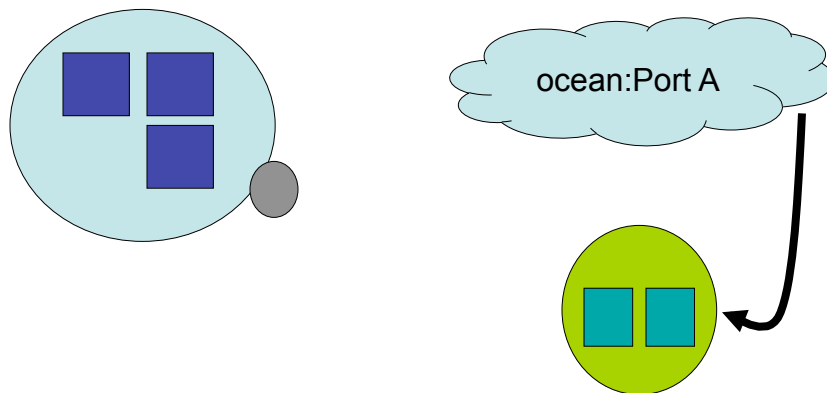
Server calls `MPI_PUBLISH_NAME("ocean", info, port_name)`

Connect / Accept Example



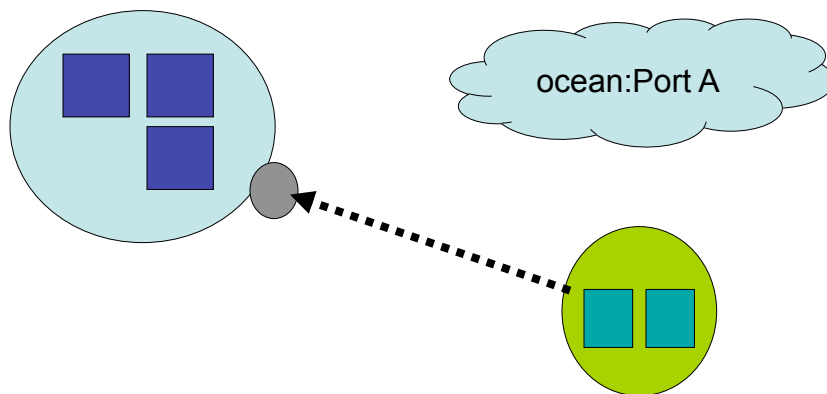
Server blocks in `MPI_COMM_ACCEPT("Port A", ...)`

Connect / Accept Example



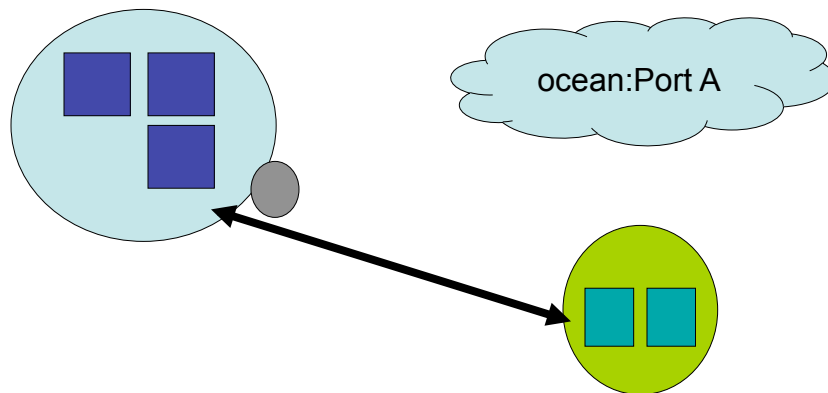
Client calls `MPI_LOOKUP_NAME("ocean", ...)`, gets "Port A"

Connect / Accept Example



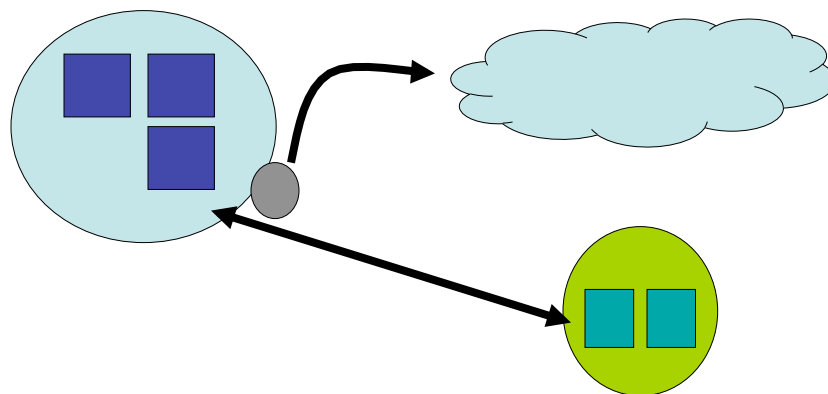
Client calls `MPI_COMM_CONNECT("Port A", ...)`

Connect / Accept Example



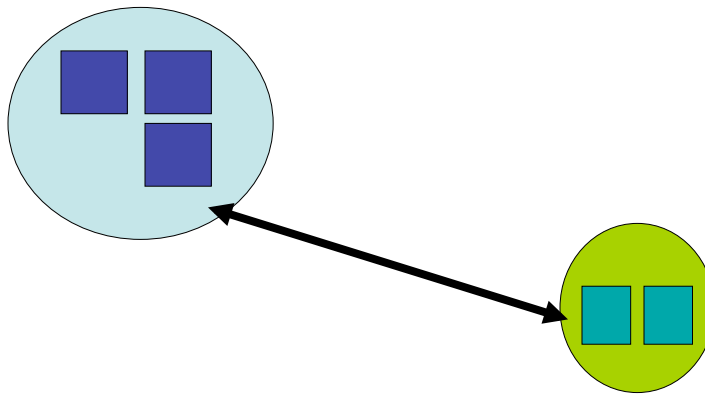
Intercommunicator formed; returned to both sides

Connect / Accept Example



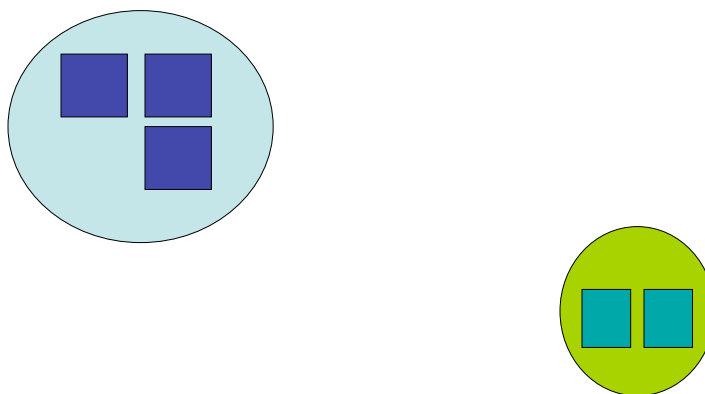
Server calls `MPI_UNPUBLISH_NAME("ocean", ...)`

Connect / Accept Example



Server calls MPI_CLOSE_PORT

Connect / Accept Example



Both sides call MPI_COMM_DISCONNECT

Summary

- Summary
 - Server opens a port
 - Server publishes public “name”
 - Client looks up public name
 - Client connects to port
 - Server unpublishes name
 - Server closes port
 - Both sides disconnect
- ➔ Similar to TCP sockets / DNS lookups

MPI_COMM_JOIN

- A third way to connect MPI processes
 - User provides a socket between two MPI processes
 - MPI creates an intercommunicator between the two processes
- ➔ Will not be covered in detail here

Disconnecting

- Once communication is no longer required
 - MPI_COMM_DISCONNECT
 - Waits for all pending communication to complete
 - Then formally disconnects groups of processes -- no longer “connected”
- Cannot disconnect MPI_COMM_WORLD