COL380

Introduction to
Parallel & Distributed Programming

Process Organization

- Communicator

  Topology

  - ➡ Groups of processes sharing a context
  - ➡ Intra and inter-communicator

  Predefined constant: MPI_COMM_WORLD

- Context
  - ➡ “communication universe”
  - ➡ Messages across context have no ‘interference’

- Groups
  - ➡ Collection of processes (can build hierarchy)
  - ➡ Ordered

  Use group-rank to address

Starting and Ending

- MPI_Init(&argc, &argv);

MPI_Init_thread

  - ➡ Needed before any other MPI call

```
int nump, id;
MPI_Comm_size (MPI_COMM_WORLD, &nump);
MPI_Comm_rank (MPI_COMM_WORLD, &id);
```

- MPI_Finalize();
  - ➡ Required

Blocking calls

```
int MPI_Send(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)
```

• message contents	blo
• count	nu
• message type	MI
• destination	ra
• tag	int
• communicator	

• message contents	memory buffer to store received message
• count	space in buffer, overflow error if too small
• message type	type of each item
• source	sender's rank (can be wild card)
• tag	message identifier (can be wild card)
• communicator	
• status	information about message received

Example

```
#include "mpi.h"     /* includes MPI library code specs */

#define MAXSIZE 100

int main(int argc, char* argv[])
{
  MPI_Init(&argc, &argv);                    // start MPI
  int nProcs, myRank, dat[2] = {5,6};
  MPI_Status status;
  MPI_Comm_size(MPI_COMM_W
  MPI_Comm_rank(MPI_COMM_W
  If(myRank == 0)
      MPI_Send(dat, 2, MPI
  If(myRank == nProcs-1)
      MPI_Recv(dat, 9, MPI_INT, 0, 11, MPI_COMM_WORLD, &status);
  MPI_Finalize();                           // stop MPI
}
```

```
status.MPI_TAG

status.MPI_SOURCE

MPI_Get_count(&status, MPI_INT,  &count);
```