Process Groups

- Ordered group of processes
- Scope communication for collective and point to point operations
- Defined dynamically (at runtime)

https://www.msi.umn.edu/content/mpi-group-management-communicator
Collective Operations

- Collective = uses all processes to accomplish the task
  - i.e. the whole communicator
- Collective allows the runtime to optimize the communication pattern

Operations:

- **MPI_Broadcast**
  - `MPI_Broadcast ( sendbuf, recvbuf, count, datatype, op, root, comm )`

- **MPI_Gather**
  - `MPI_Gather(sendbuf,sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)`

http://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/
Collective Operations II

- Operations
  - All-to-all communication
    - Provide unified global view
  - Scatter
    - Disseminate from a single process

http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/
MPI Reductions

MPI_Reduce ( sendbuf, recvbuf, count, datatype, op, root, comm )

- Using the specified operation
  - Aggregates: mean, sum
  - Extrema: min, max
  - User defined functions via MPI_OP_CREATE
    - Required property of function: Associativity
Reduce Functions

- All operations are “algebraic” in that they can be applied in any order.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>Maximum</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>Minimum</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>Sum</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>Product</td>
</tr>
<tr>
<td>MPI_BAND</td>
<td>Bit-wise and</td>
</tr>
<tr>
<td>MPI_LAND</td>
<td>Logical and</td>
</tr>
<tr>
<td>MPI_BOR</td>
<td>Bit-wise or</td>
</tr>
<tr>
<td>MPI_LXOR</td>
<td>Logical exclusive or</td>
</tr>
<tr>
<td>MPI_BXOR</td>
<td>Bit-wise exclusive or</td>
</tr>
<tr>
<td>MPI_MAXLOC</td>
<td>Maximum value and corresponding index</td>
</tr>
<tr>
<td>MPI_MINLOC</td>
<td>Minimum value and corresponding index</td>
</tr>
</tbody>
</table>
Reduce Example

- Using MPI_Reduce

```c
h = 1.0 / (double) n;
sum = 0.0;
for (i = rank + 1; i <= n; i += size) {
    x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
mypi = h * sum;

MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI::DOUBLE, 
                       MPI::SUM, 0);
if (rank == 0)
    cout << "pi is approximately " << pi 
    << ", Error is " << fabs(pi - PI25DT) 
    << endl;
```