COMP 364 / 464
High Performance Computing

Collection Commumcation:

Message Passing Interface (MPI)

MPI Collective Communications

- Every process *within the communicator* MUST call the routine
  - All calls are blocking
  - A process may return when participation is complete
  - May or may not synchronize (implementation dependent)
- You can build your own using p2p methods but MPI has done most of this for us (send/recv are the building blocks)
- Send and Receive sizes must match
  - mapping may vary
- Basic calls have a *root (or destination)* — ”all” versions don’t

MPI Collective Communications

- Involves a group of processes.
- Basic Routines

  Broadcast— **`MPI_Bcast()`**

  Reduce— **`MPI_Reduce()`**

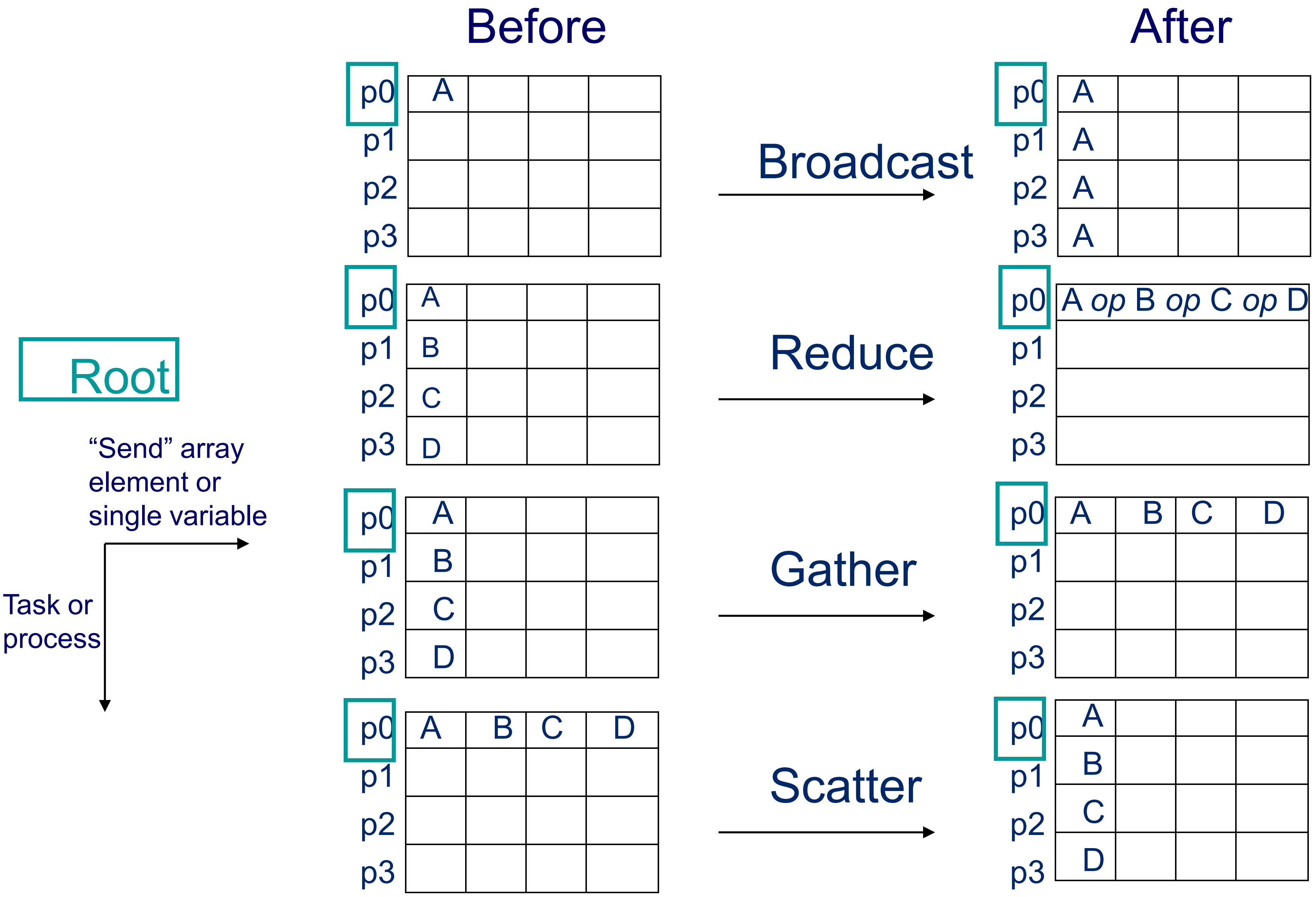
  Gather/Scatter— **`MPI_Gather()MPI_Scatter()...`**
- “All” versions

  `MPI_Allreduce()`

  `MPI_Allgather()`

  `MPI_Alltoall()...`
- Others

  `MPI_Barrier ...`



Root

"Send" array element or single variable

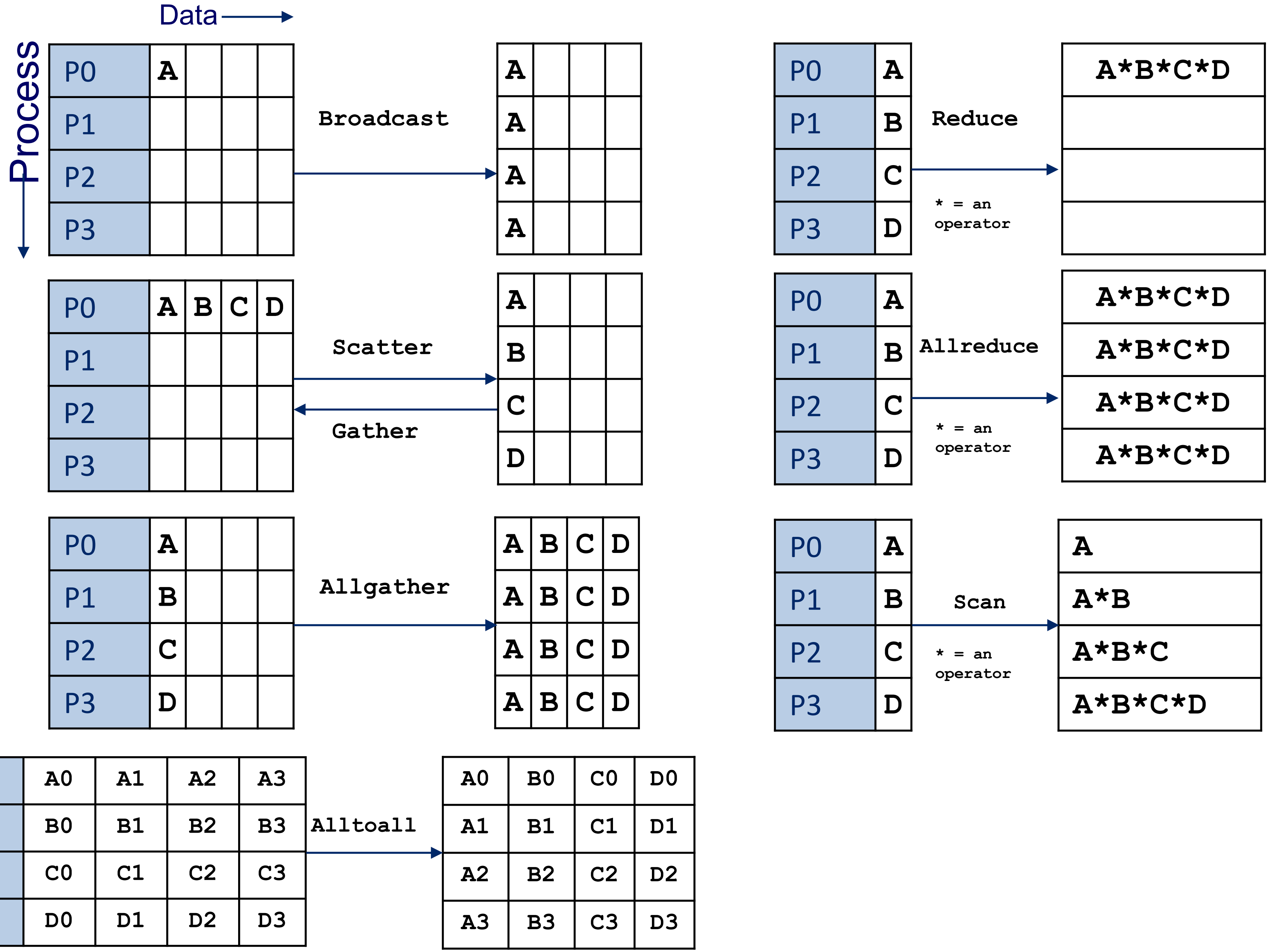
Task or process

	Before						After			
p0	A					p0	A			
p1					Broadcast	p1	A			
p2						p2	A			
p3						p3	A			

	Before						After
p0	A					p0	A *op* B *op* C *op* D
p1	B				Reduce	p1	
p2	C					p2	
p3	D					p3	

	Before						After			
p0	A					p0	A	B	C	D
p1	B				Gather	p1				
p2	C					p2				
p3	D					p3				

	Before						After			
p0	A	B	C	D		p0	A			
p1					Scatter	p1	B			
p2						p2	C			
p3						p3	D			



Broadcast

P0	A			
P1				
P2				
P3				

A			
A			
A			
A			

Scatter / Gather

P0	A	B	C	D
P1				
P2				
P3				

A			
B			
C			
D			

Allgather

P0	A			
P1	B			
P2	C			
P3	D			

A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Alltoall

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3

Reduce

* = an operator

P0	A
P1	B
P2	C
P3	D

A*B*C*D

Allreduce

* = an operator

P0	A
P1	B
P2	C
P3	D

A*B*C*D
A*B*C*D
A*B*C*D
A*B*C*D

Scan

* = an operator

P0	A
P1	B
P2	C
P3	D

A
A*B
A*B*C
A*B*C*D

Broadcast Operation: **MPI_Bcast**

- All nodes call **`MPI_Bcast`**
- One node (**`root`**) sends a message to all
  - all others receive the message
- All MPI functions have error handling. MPI_SUCCESS = all good.

```
int ierr = MPI_Bcast(&dat, cnt, datatype, root,
   comm);
```

Reduction Operations

- Used to combine (reduce) partial results from all processors
- Result returned to root processor
- pre-defined or user-defined operations
  - Predefined: associative & commutative (com)
    Order may not be canonical (i.e., rank order)
  - User defined: Must be associative. com or non-com
    “Canonical” evaluation
- Works on a scalar variable or arrays (elemental)

MPI_Reduce

```
ierr = MPI_Reduce(&sbuf[0], &rbuf[0],
  count, datatype, operator, root, comm)
```

- Parameters
  - like **`MPI_Bcast`**, a root is specified
  - operation is a type of mathematical operation
- Applies the operator to each element globally
  - send and receive buffers are the same size
- Use **`MPI_Op_create`** for user-defined operation.

Operations for MPI_Reduce

MPI_PROD	Product
MPI_SUM	Sum
MPI_LAND	**L**ogical and
MPI_LOR	**L**ogical or
MPI_LXOR	**L**ogical exclusive or
MPI_BAND	**B**itwise and
MPI_BOR	**B**itwise or
MPI_BXOR	**B**itwise exclusive or
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_MAXLOC	Maximum value and location
MPI_MINLOC	Minimum value and location

Dot Product of Two Vectors

```
double a[N], b[N];
…
double localSum=0.0;
for (int i = 0; i < N; ++i)
   localSum += a[i]*b[i];

double globalSum;
const int root = 0;
MPI_Reduce(&localSum, &globalSum, 1, MPI_DOUBLE,
   MPI_SUM, root, MPI_COMM_WORLD);

printf(“rank= %d local= %f global= %f\n”,
   rank, localSum, globalSum);
```

MPI_Scatter Syntax

```
ierr = MPI_Scatter(&sbuf[0], scnt, stype, &rbuf[0],
rcnt, rtype, root, comm);
```

- Parameters
  - **sbuf** = array of size np*scnt (np = # of ranks)
  - **scnt** = number of elements sent to each processor
  - **rcnt** = number of element(s) obtained from the root processor
  - **rbuf** = element(s) obtained from the root processor (rcnt in size)

e.g. **MPI_Scatter(**S, 1, **stype**, R, 1, **rtype**, **root**, **comm**)

Array Scalar

We gratefully acknowledge the sponsorship of Chevron Corporation, whose generous support of TACC has made possible this Scientific Computing Curriculum and other student-focused initiatives.