

# Parallel Computing for Science & Engineering Spring 2013: MPI collectives 1

Instructors:

Victor Eijkhout, Research Scientist, TACC

Kent Milfeld, Research Associate, TACC



# MPI Collective Communications

- Involves a group of processes.

- Basic Routines

Broadcast— `MPI_Bcast()`

Reduce— `MPI_Reduce()`

Gather/Scatter— `MPI_Gather()/MPI_Scatter()...`

- “All” versions

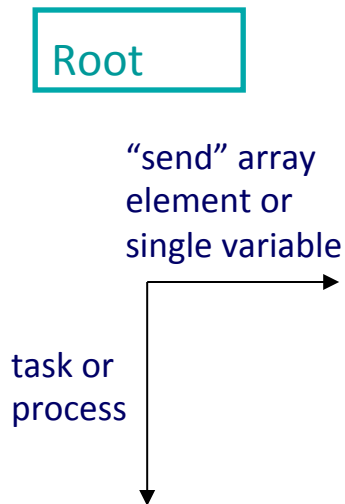
`MPI_Allreduce()`

`MPI_Allgather()`

`MPI_Alltoall()...`

- Others

`MPI_Barrier ...`



before

p0	A			
p1				
p2				
p3				

p0	A			
p1	B			
p2	C			
p3	D			

p0	A	B	C	D
p1				
p2				
p3				

broadcast

reduce

gather

scatter

after

p0	A			
p1	A			
p2	A			
p3	A			

p0	$A \text{ op } B \text{ op } C \text{ op } D$			
p1				
p2				
p3				

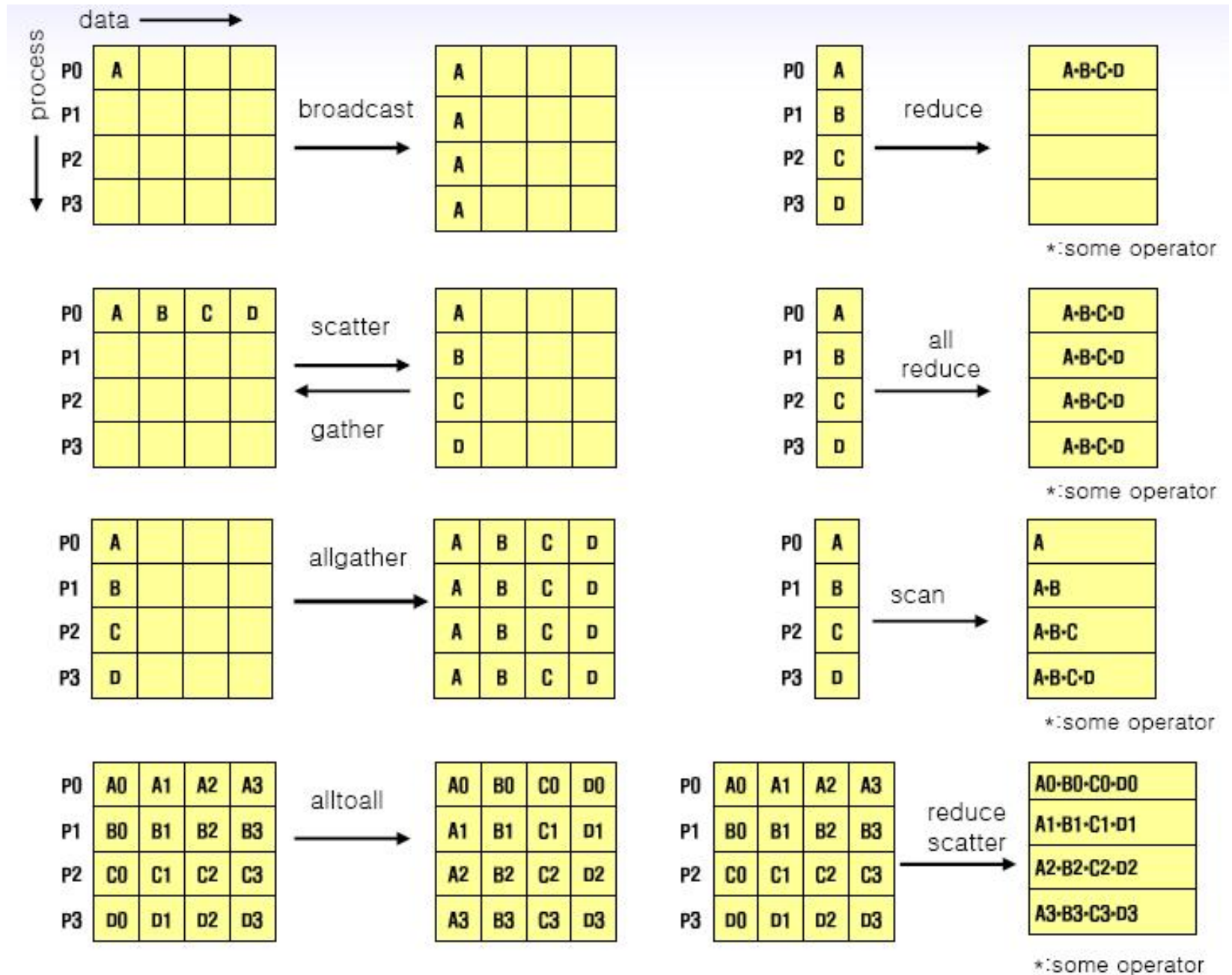
  

p0	A	B	C	D
p1				
p2				
p3				

p0	A			
p1	B			
p2	C			
p3	D			

# Collective Communication: Summary



# MPI Collective Communications

- Every process **MUST** call the routine
  - All calls are blocking
  - A task may return when participation is complete
  - May or may not synchronize (implementation dependent)
- Must have “matching” arguments
  - no status
  - no tags
- Send and Receive sizes must match
  - mapping may vary
- Basic calls have a *root*—“all” versions don’t

# Broadcast Operation: MPI\_Bcast

- All nodes call MPI\_Bcast
- One node (root) sends a message to all
  - all others receive the message

- C

```
ierr=MPI_Bcast(&dat[0], cnt, datatype, root, comm);
```

- Fortran

```
call MPI_Bcast( dat, cnt, datatype, root, comm, ierr)
```

# 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 (rank order)
  - User defined: Must be associative. com or non-com  
“Canonical” evaluation
- Works on a scalar variable or arrays (elemental)

# MPI\_Reduce

- C

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

- Fortran

```
call MPI_Reduce( sbuf,      rbuf,      count,  
               datatype, operator, root,  
               comm, ierr)
```

- 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	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or
MPI_BAND	Bitwise and
MPI_BOR	Bitwise or
MPI_BXOR	Bitwise 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    globalSum=0.0, localSum=0.0;  
...  
for (i=0; i<N; ++i) localSum+=a[i]*b[i];  
  
MPI_Reduce(&localSum, &globalSum, 1, MPI_DOUBLE  
           MPI_SUM, root, MPI_COMM_WORLD);
```

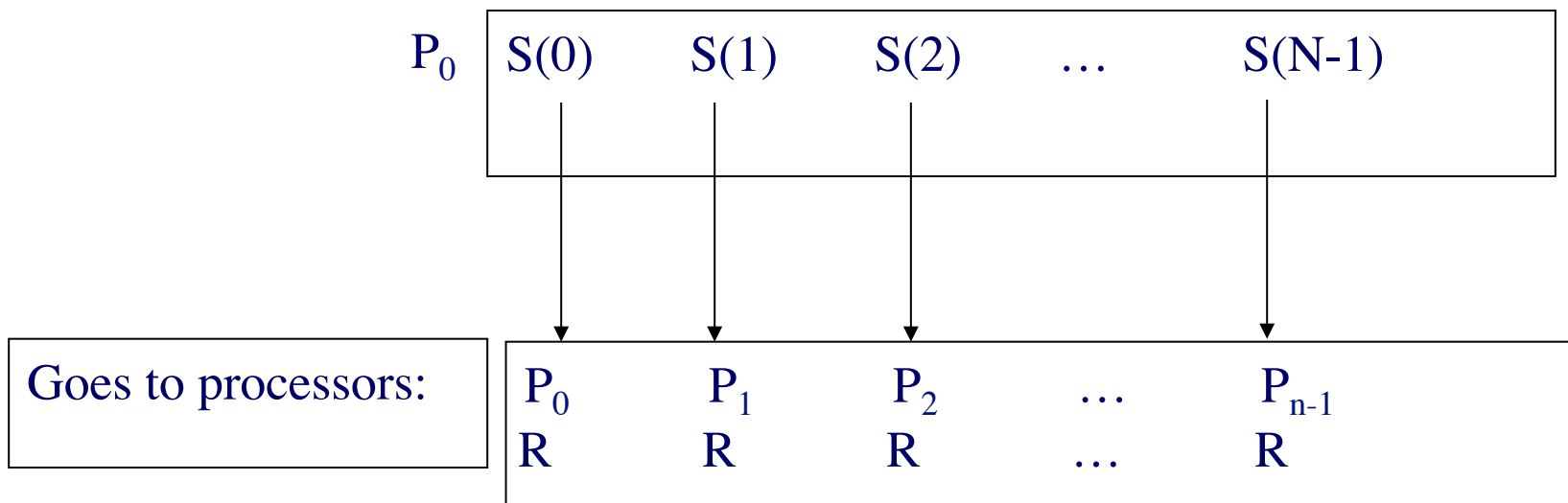
# Dot Product of Two Vectors

```
real*8 :: a(N), b(N);  
real*8 :: globalSum=0.0, localSum=0.0;  
...  
localSum = dot_product(a,b);  
  
MPI_Reduce( localSum, globalSum, 1, MPI_REAL8 &  
            MPI_SUM, root, MPI_COMM_WORLD, ierr);
```

# Scatter Operation using MPI\_Scatter

- Similar to Broadcast but sends a section of an array to each processor

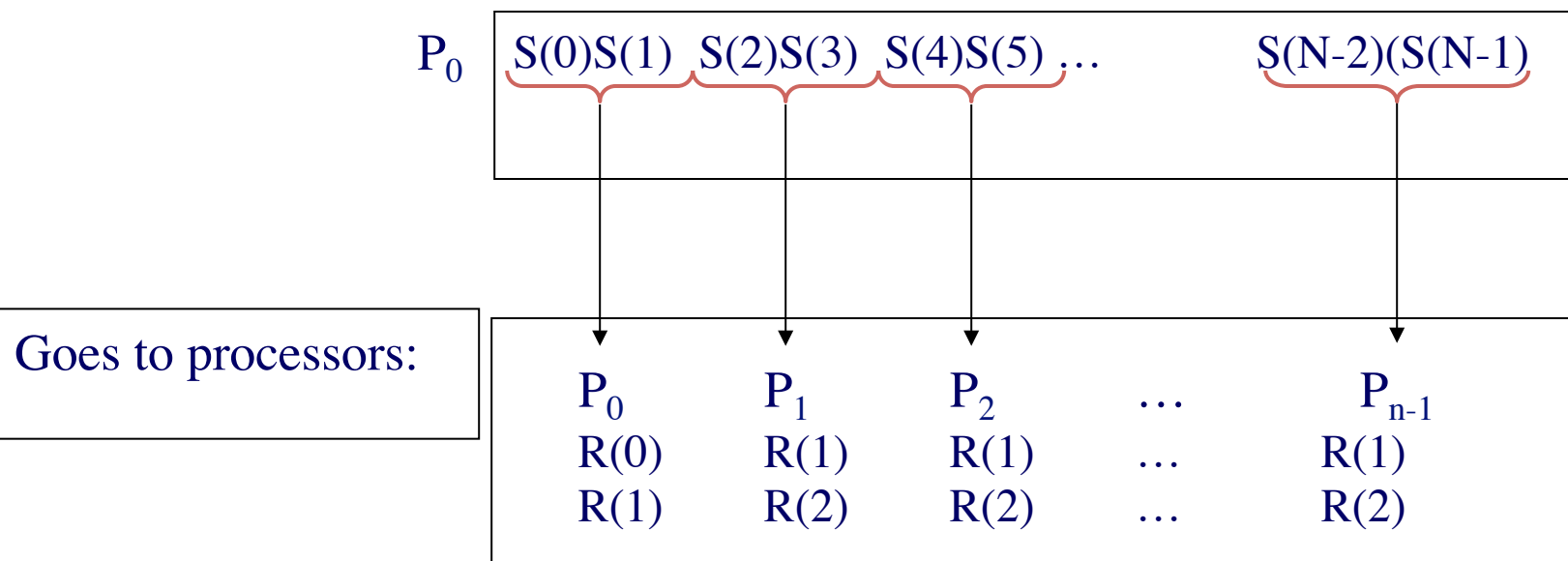
Data in an array on root node,  $P_0$ , sending 1 element to each task:



# Scatter Operation using MPI\_Scatter

- 2-elements per processor

Data in an array on root node,  $P_0$ , sending 2 elements:



# MPI\_Scatter Syntax

- C

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

- Fortran

```
call MPI_Scatter( sbuf, scnt, stype, rbuf, rcnt, rtype, root, comm, ierr)
```

- 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