Collective Communication

- Collective Communication
- Barrier Synchronization
- Broadcast*
- Scatter*
- Gather
- Gather/Scatter Variations
- Summary Illustration
- Global Reduction Operations
- Predefined Reduction Operations
- MPI_Reduce
- Minloc and Maxloc*
- User-defined Reduction Operators
- Reduction Operator Functions
- Registering a User-defined Reduction Operator*
- Variants of MPI_Reduce
- Class Exercise: Last Ring

*includes sample C and Fortran programs

Collective Communication

- Communications involving a group of processes
- Called by *all* processes in a communicator
- Examples:
  - Broadcast, scatter, gather (Data Distribution)
  - Global sum, global maximum, etc. (Collective Operations)
  - Barrier synchronization

Characteristics of Collective Communication

- Collective communication will not interfere with point-to-point communication and vice-versa
- All processes must call the collective routine
- Synchronization not guaranteed (except for barrier)
- No non-blocking collective communication
- No tags
- Receive buffers must be exactly the right size

Barrier Synchronization

- Red light for each processor: turns green when all processors have arrived
- Slower than hardware barriers (example: SGI/Cray T3E)

C:

```
int MPI_Barrier (MPI_Comm comm)
```

Fortran:

```
CALL MPI_BARRIER (COMM,IERROR)
INTEGER COMM,IERROR
```

Broadcast

- One-to-all communication: same data sent from root process to all the others in the communicator

- C:

```
int MPI_Bcast (void *buffer, int, count,
  MPI_Datatype datatype,int root, MPI_Comm comm)
```

- Fortran:

```
MPI_BCAST(BUFFER, COUNT, DATATYPE, ROOT, COMM IERROR)

<type> BUFFER (*)
INTEGER COUNT, DATATYPE, ROOT, COMM, IERROR
```

- All processes must specify same root rank and communicator

Sample Program #5 - Fortran

```
PROGRAM broadcast
INCLUDE 'mpif.h'
INTEGER err, rank, size
real param
CALL MPI_INIT(err)
CALL MPI_COMM_RANK(MPI_WORLD_COMM,rank,err)
CALL MPI_COMM_SIZE(MPI_WORLD_COMM,size,err)
if(rank.eq.5) param=23.0
call MPI_BCAST(param,1,MPI_REAL,5,MPI_COMM_WORLD,err)
print *,"P:",rank," after broadcast param is ",param
CALL MPI_FINALIZE(err)
END
```

```
            Program Output
P:1 after broadcast parameter is 23.
P:3 after broadcast parameter is 23.
P:4 after broadcast parameter is 23
P:0 after broadcast parameter is 23
P:5 after broadcast parameter is 23.
P:6 after broadcast parameter is 23.
P:7 after broadcast parameter is 23.
P:2 after broadcast parameter is 23.
```

Scatter

- One-to-all communication: different data sent to each process in the communicator (in rank order)

C:

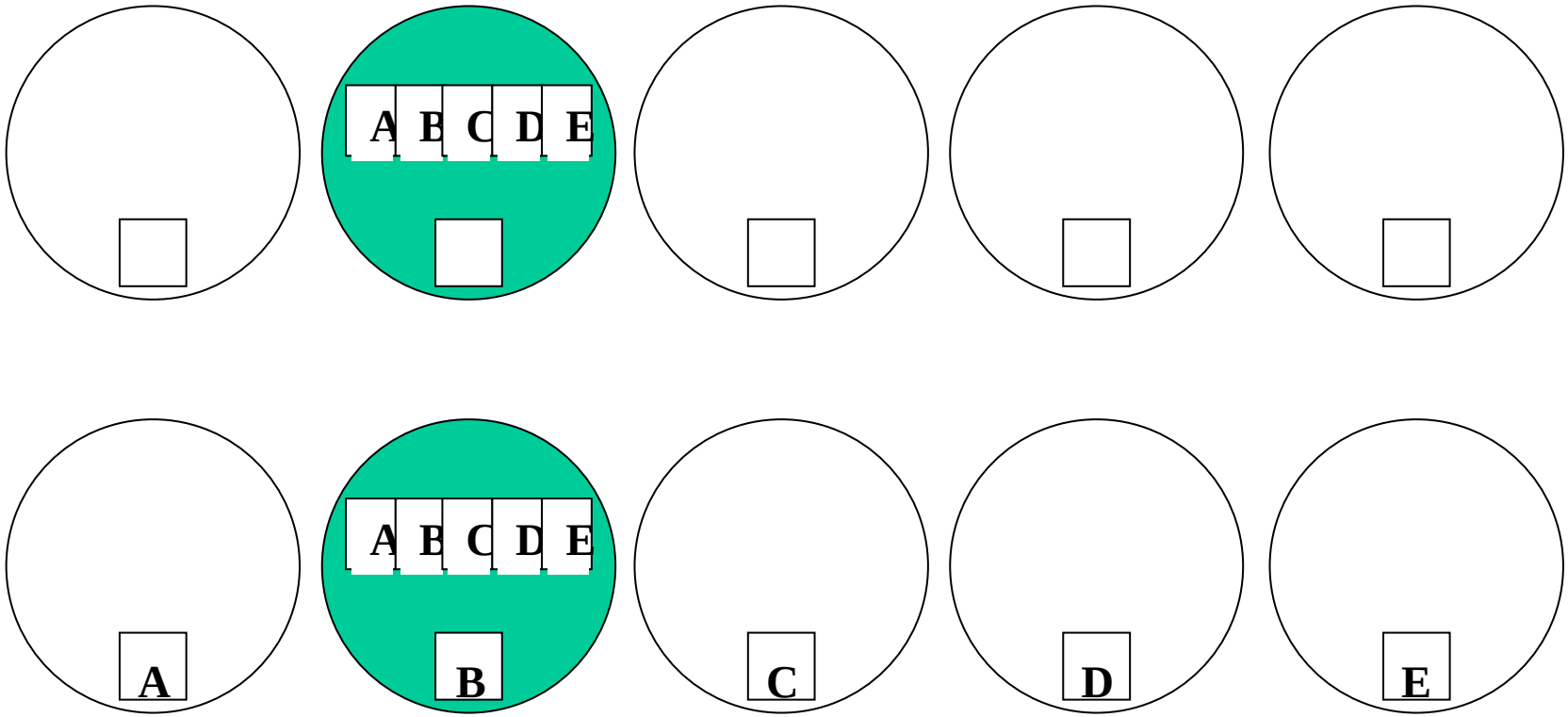
```
int MPI_Scatter(void* sendbuf, int sendcount,
        MPI_Datatype sendtype, void* recvbuf, int recvcount,
        MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Fortran:

```
CALL MPI_SCATTER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF,
        RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)
<type> SENDBUF(*), RECVBUF(*)
```

- `sendcount` is the number of elements sent to each process, not the "total" number sent
  - send arguments are significant only at the root process

Scatter Example



Sample Program #6 - Fortran

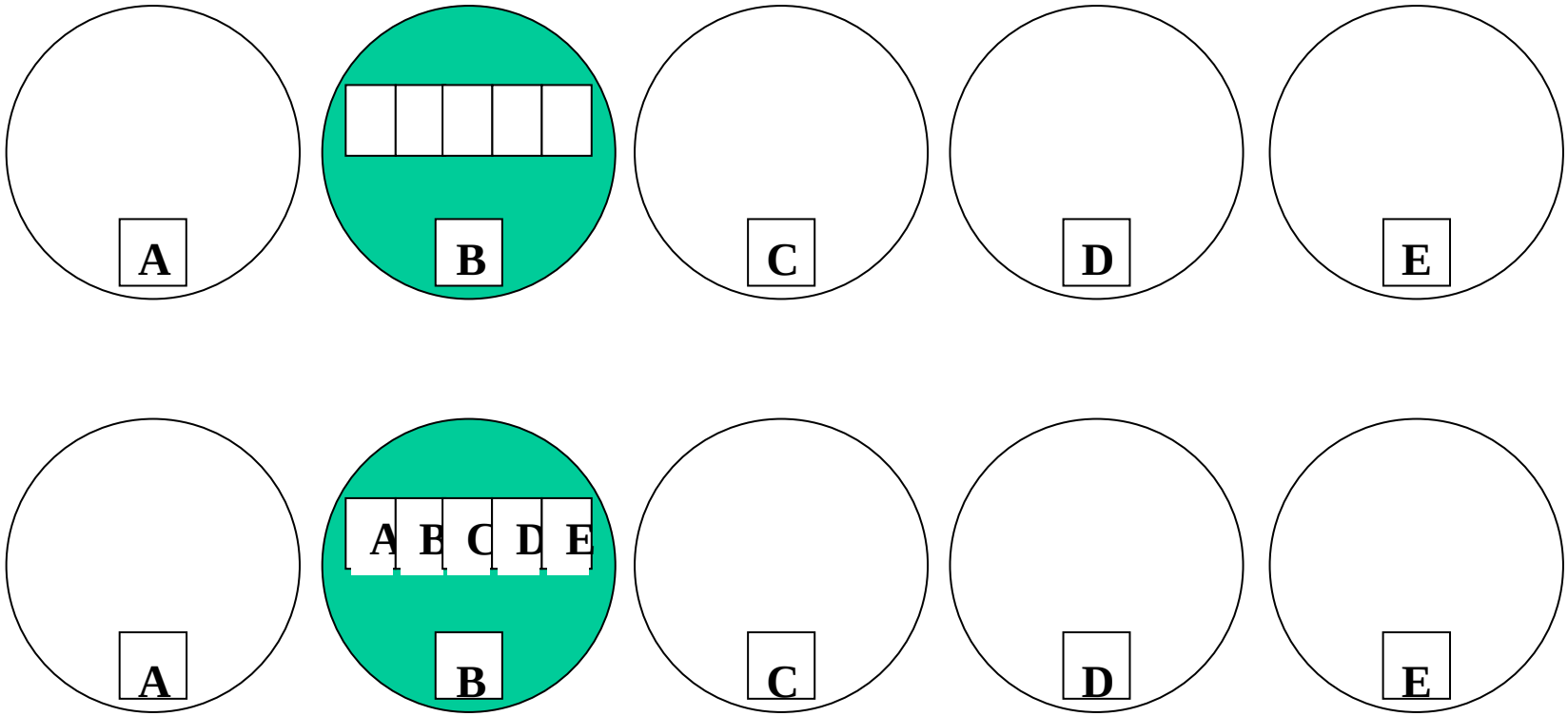
```
 PROGRAM scatter
 INCLUDE 'mpif.h'
 INTEGER err, rank, size
 real param(4), mine
 integer sndcnt,rcvcnt
 CALL MPI_INIT(err)
 CALL MPI_COMM_RANK(MPI_WORLD_COMM,rank,err)
 CALL MPI_COMM_SIZE(MPI_WORLD_COMM,size,err)
 rcvcnt=1
 if(rank.eq.3) then
    do i=1,4
       param(i)=23.0+i
    end do
    sndcnt=1
 end if
 call MPI_SCATTER(param,sndcnt,MPI_REAL,mine,rcvcnt,MPI_REAL,
&            3,MPI_COMM_WORLD,err)
 print *,"P:",rank," mine is ",mine
 CALL MPI_FINALIZE(err)
 END
```

```
   Program Output
P:1 mine is 25.
P:3 mine is 27.
P:0 mine is 24.
P:2 mine is 26.
```

Gather

- All-to-one communication: different data collected by root process
  - Collection done in rank order
- `MPI_GATHER` & `MPI_Gather` have same arguments as matching scatter routines
- Receive arguments only meaningful at the root process

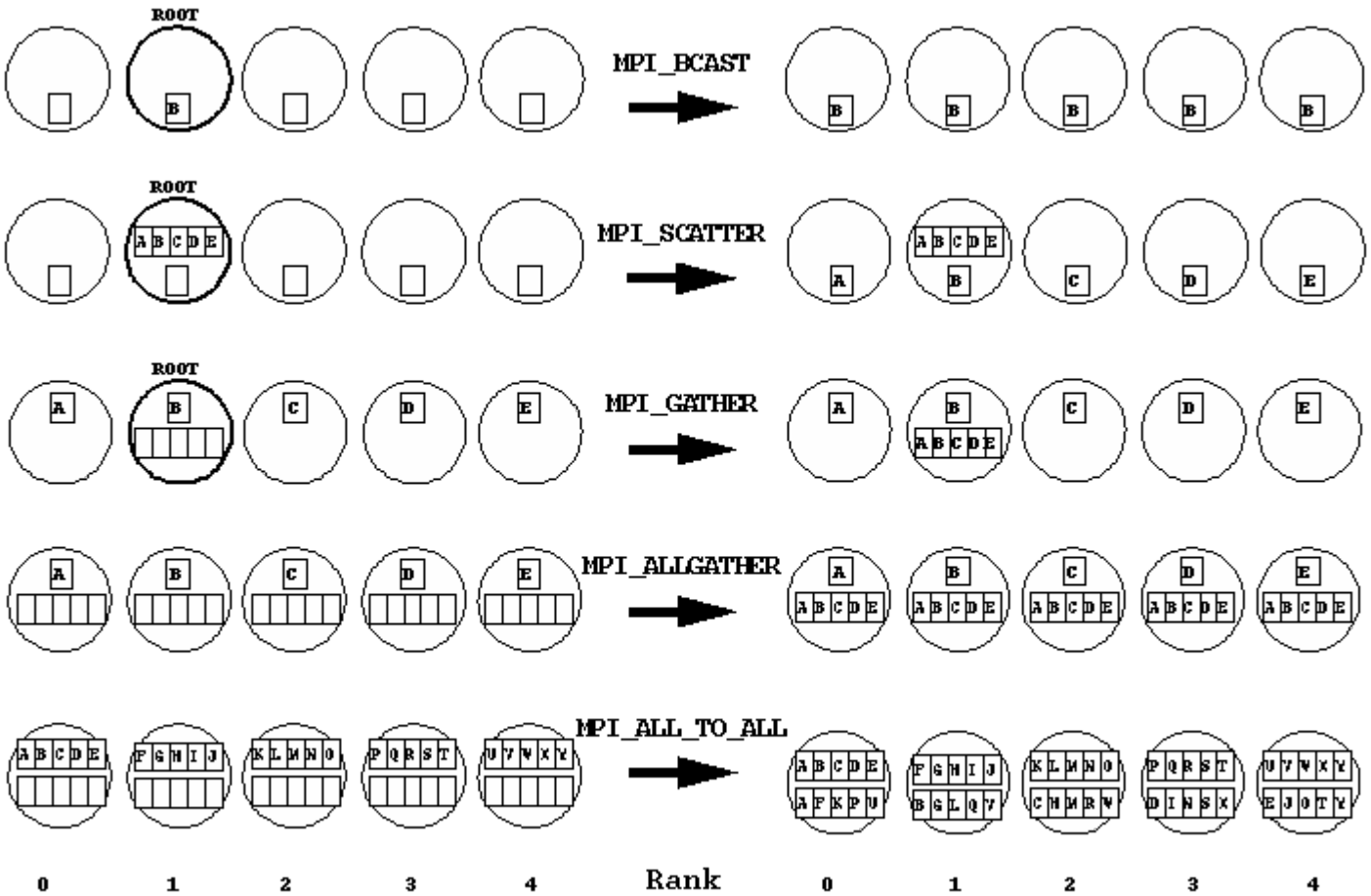
Gather Example



Gather/Scatter Variations

- `MPI_Allgather`
- `MPI_Alltoall`
- No root process specified: all processes get gathered or scattered data
- Send and receive arguments significant for all processes

Summary


ROOT
MPI_BCAST
ROOT
MPI_SCATTER
ROOT
MPI_GATHER
MPI_ALLGATHER
MPI_ALL_TO_ALL
Rank
0
1
2
3
4

Global Reduction Operations

- Used to compute a result involving data distributed over a group of processes
- Examples:
  - Global sum or product
  - Global maximum or minimum
  - Global user-defined operation

Example of Global Reduction

- Sum of all the `x` values is placed in `result` only on processor 0

C:

```
MPI_Reduce (&x, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD)
```

Fortran:

```
CALL MPI_REDUCE (x,result,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,IERROR)
INTEGER COMM,IERROR
```

Predefined Reduction Operations

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

General Form

- `count` is the number of "*ops*" done on consecutive elements of `sendbuf` (it is also size of `recvbuf`)

- `op` is an associative operator that takes two operands of type `datatype` and returns a result of the same type

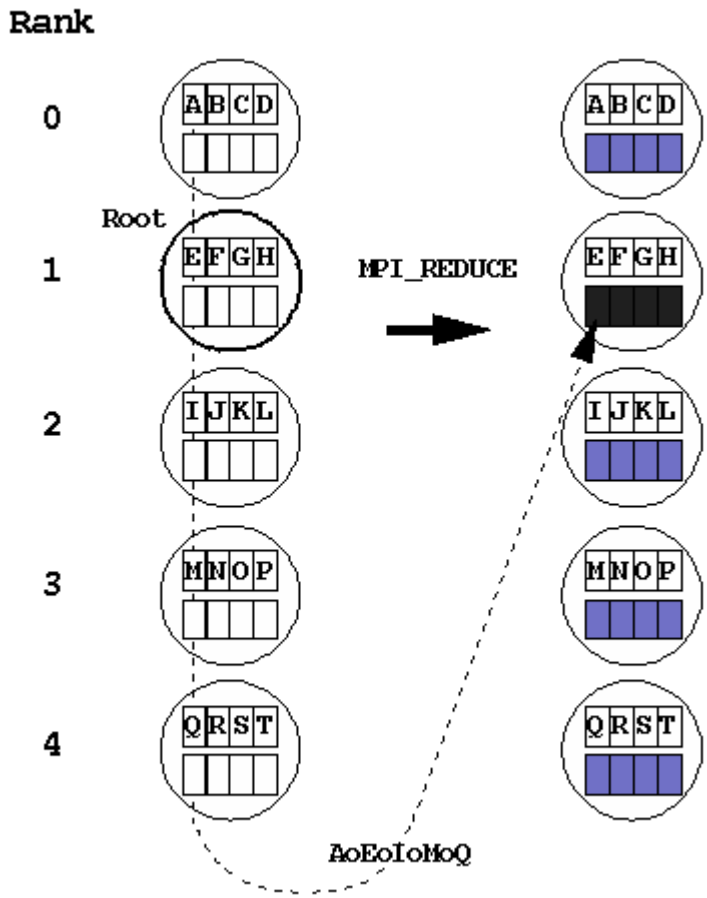
C:

```
int MPI_Reduce(void* sendbuf, void* recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm)
```

Fortran:

```
CALL MPI_REDUCE(SENDBUF,RECVBUF,COUNT,DATATYPE,OP,ROOT,COMM,IERROR)
<type> SENDBUF(*), RECVBUF(*)
```

MPI_Reduce


Rank
0
1
2
3
4
Root
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_REDUCE
A B C D
E F G H
I J K L
M N O P
Q R S T
AoEoIoMoQ

Minloc and Maxloc

- Designed to compute a global minimum/maximum and and index associated with the extreme value
  - Common application: index is the processor rank (see sample program)
- If more than one extreme, get the first
- Designed to work on operands that consist of a value and index pair
- `MPI_Datatypes` include:

C:

```
MPI_FLOAT_INT, MPI_DOUBLE_INT, MPI_LONG_INT, MPI_2INT,
   MPI_SHORT_INT, MPI_LONG_DOUBLE_INT
```

Fortran:

```
MPI_2REAL, MPI_2DOUBLEPRECISION, MPI_2INTEGER
```

Sample Program #7 - Fortran

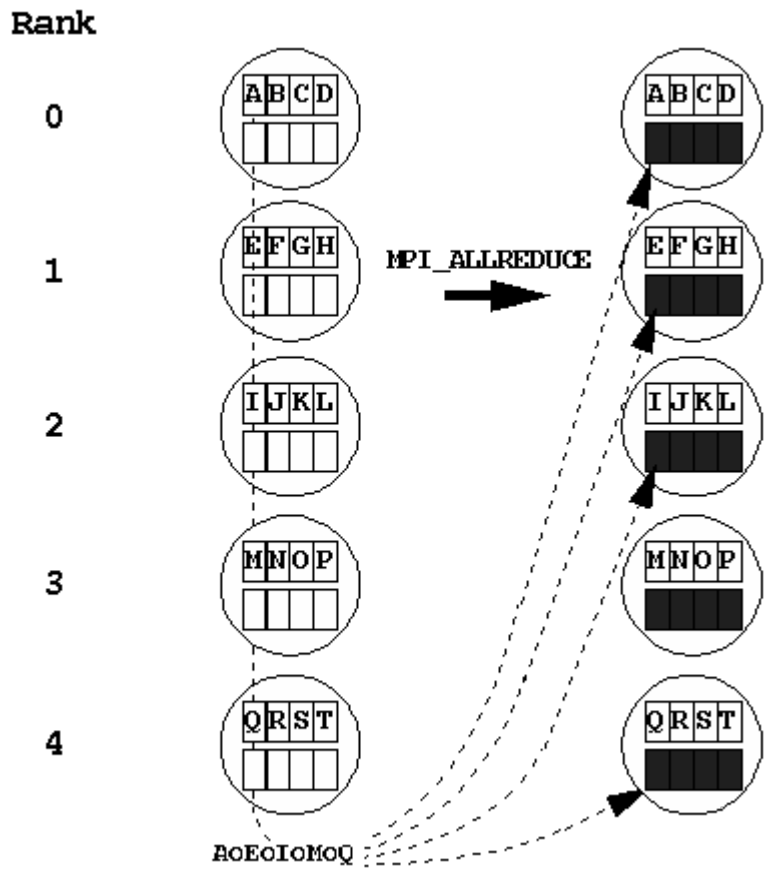
```
      PROGRAM MaxMin
C
C Run with 8 processes
C
      INCLUDE 'mpif.h'
      INTEGER err, rank, size
      integer in(2),out(2)
      CALL MPI_INIT(err)
      CALL MPI_COMM_RANK(MPI_WORLD_COMM,rank,err)
      CALL MPI_COMM_SIZE(MPI_WORLD_COMM,size,err)
      in(1)=rank+1
      in(2)=rank
      call MPI_REDUCE(in,out,1,MPI_2INTEGER,MPI_MAXLOC,
     &                7,MPI_COMM_WORLD,err)

      if(rank.eq.7) print *,"P:",rank," max=",out(1)," at rank ",out(2)

      call MPI_REDUCE(in,out,1,MPI_2INTEGER,MPI_MINLOC,
     &                2,MPI_COMM_WORLD,err)

      if(rank.eq.2) print *,"P:",rank," min=",out(1)," at rank ",out(2)

      CALL MPI_FINALIZE(err)
      END
```

```
    Program Output
P:2 min=1 at rank 0
P:7 max=8 at rank 7
```

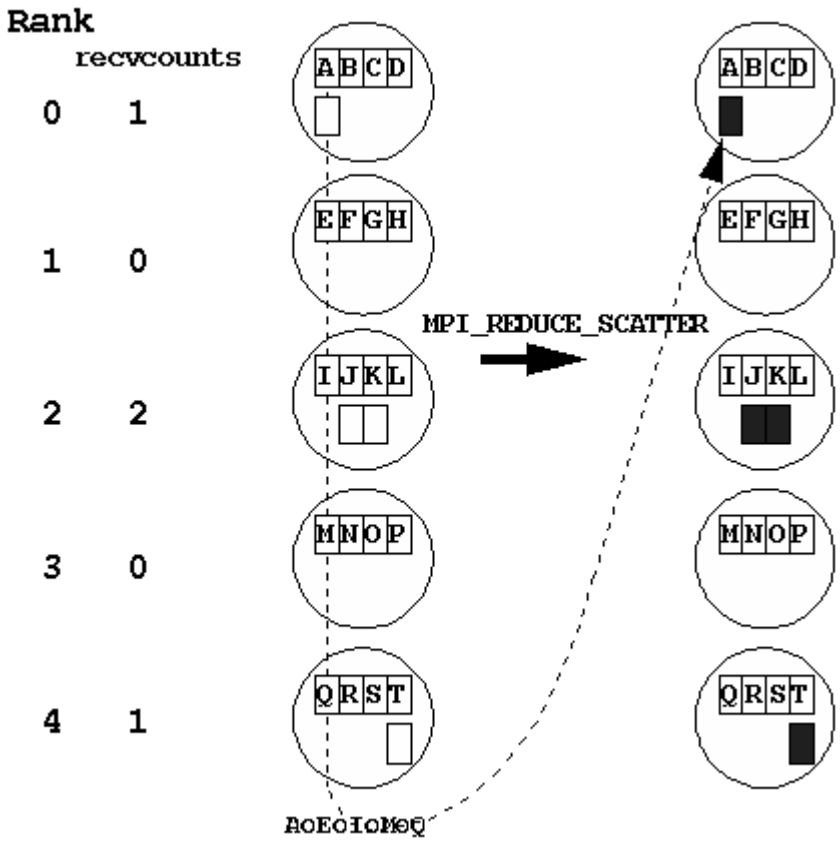
Variants of MPI_REDUCE

- `MPI_ALLREDUCE` -- no root process (all get results)
- `MPI_REDUCE_SCATTER` -- multiple results are scattered
- `MPI_SCAN` -- “parallel prefix”

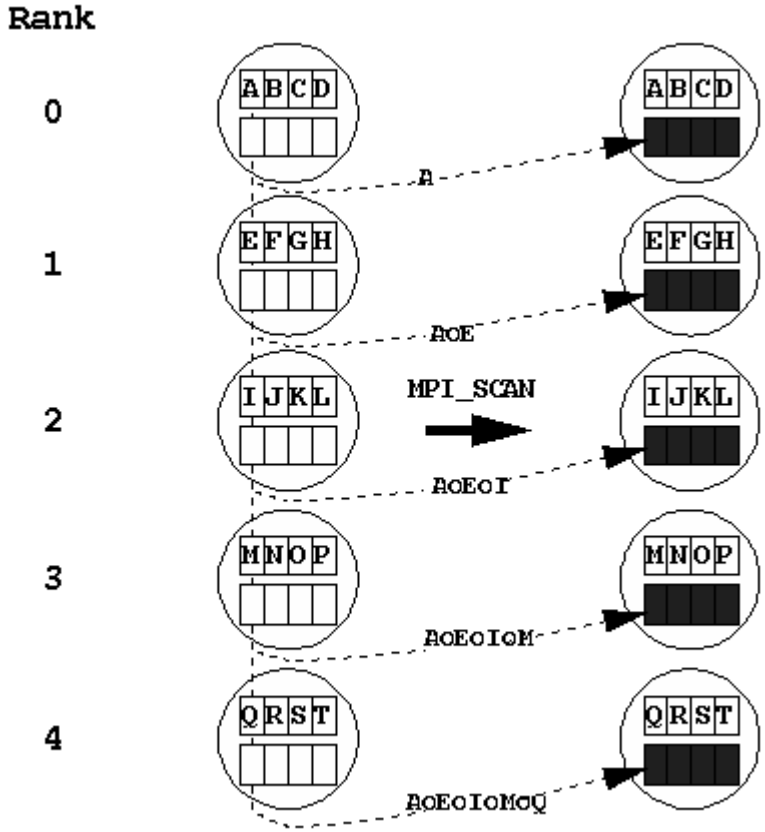
MPI_ALLREDUCE



MPI_REDUCE_SCATTER



MPI_SCAN


Rank
0
1
2
3
4
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_SCAN
A
AoE
AoEoI
AoEoIoM
AoEoIoMoQ