COL380

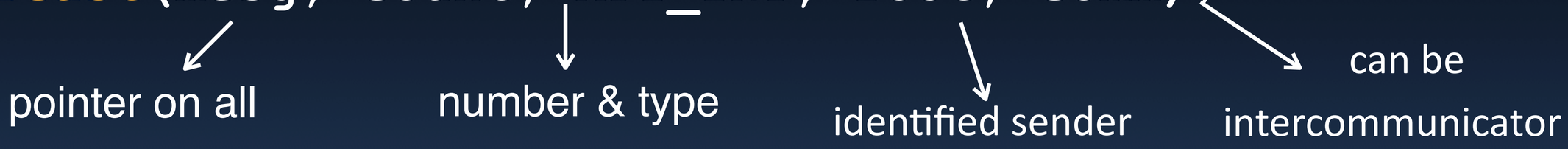
Introduction to
Parallel & Distributed Programming

Agenda

- MPI Collectives and their implementation
- One-sided communication
- MPI supported file IO

Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```



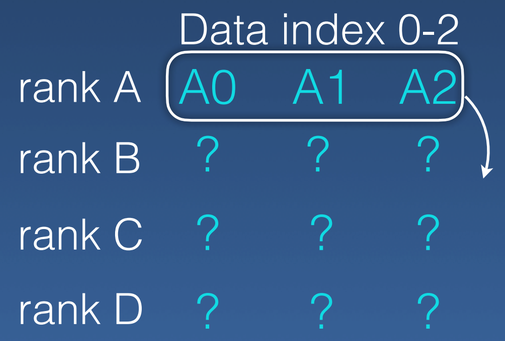
- All participants must call, match by comm and root
- No implicit synchronization

Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```

- mesg: pointer on all
- count, MPI_INT: number & type
- root: identified sender
- comm: can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization

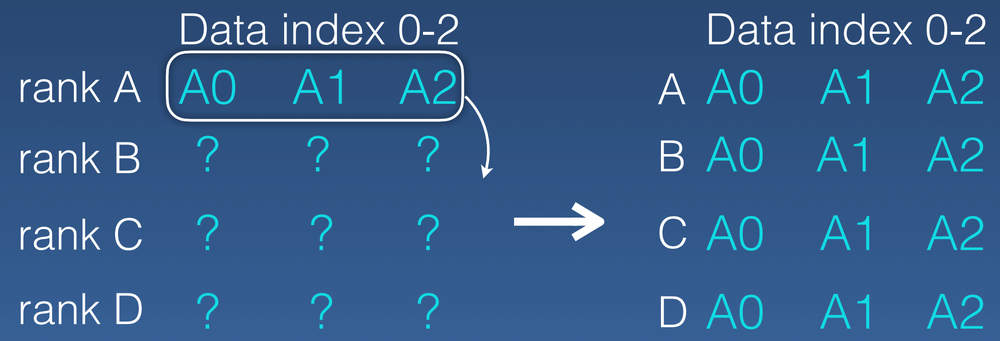


Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```

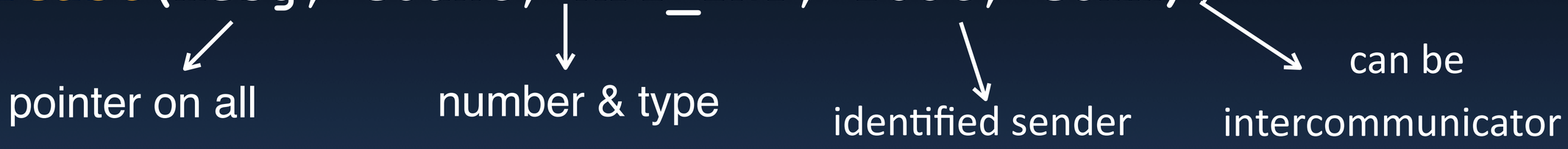
- mesg: pointer on all
- count: number & type
- root: identified sender
- comm: can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization



Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```



- All participants must call, match by comm and root
- No implicit synchronization

Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```

- mesg: pointer on all
- count, MPI_INT: number & type
- root: identified sender
- comm: can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization

Rank 0
```
MPI_Bcast(buf1, count, type, 0, comm);
MPI_Bcast(buf2, count, type, 1, comm);
```

Rank 1
```
MPI_Bcast(buf1, count, type, 1, comm);
MPI_Bcast(buf2, count, type, 0, comm);
```

Deadlock

Broadcast

See: MPI_Reduce, MPI_Scan

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```

- mesg: pointer on all
- count, MPI_INT: number & type
- root: identified sender
- comm: can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization

see `MPI_Ibcast`

Rank 0

```
MPI_Bcast(buf1, count, type, 0, comm);
MPI_Bcast(buf2, count, type, 1, comm);
```

Rank 1

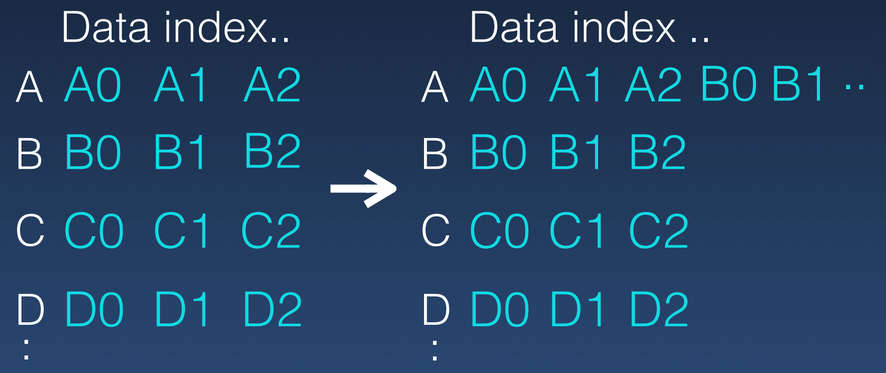
```
MPI_Bcast(buf1, count, type, 1, comm);
MPI_Bcast(buf2, count, type, 0, comm);
```

Deadlock

MPI_Gather

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

- Similar to non-roots sending:
  - MPI_Send(sendbuf, sendcount, sendtype, root, ...),
- and the root receiving n times:
  - MPI_Recv(recvbuf + i * recvcount *extent(recvtype), recvcount, recvtype, i, ...)

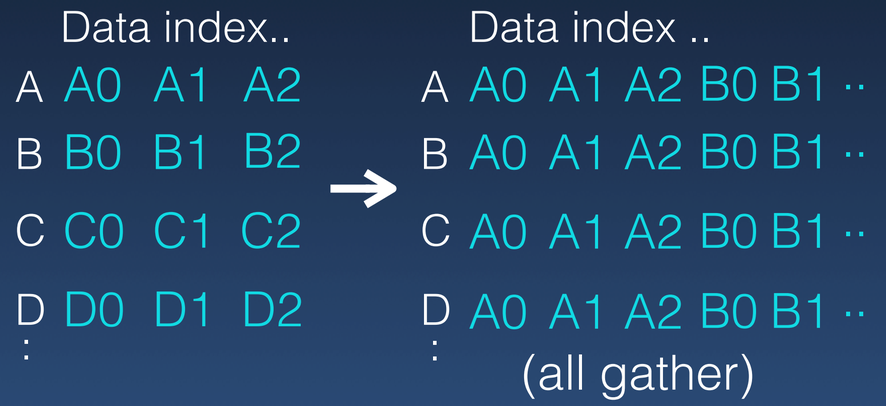


- **MPI_Gatherv** allows different size data to be gathered
- **MPI_Allgather** has no *root*; all nodes receive similarly

MPI_Gather

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

- Similar to non-roots sending:
  - MPI_Send(sendbuf, sendcount, sendtype, root, ...),
- and the root receiving n times:
  - MPI_Recv(recvbuf + i * recvcount *extent(recvtype), recvcount, recvtype, i, ...)

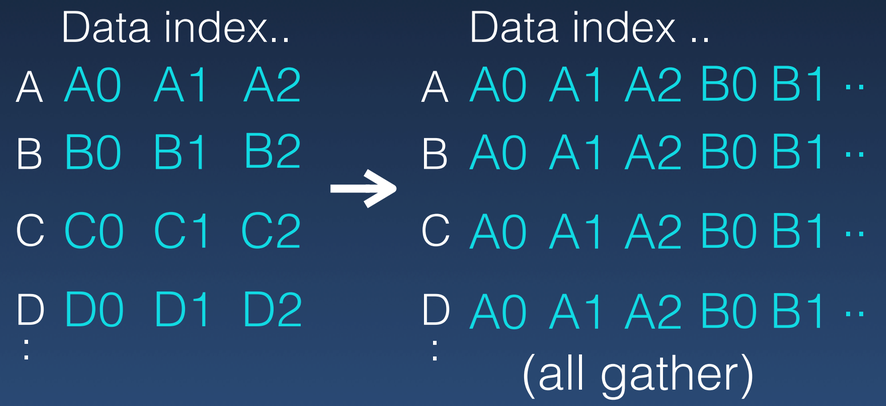


- **MPI_Gatherv** allows different size data to be gathered
- **MPI_Allgather** has no *root*; all nodes receive similarly

MPI_Scatter is opposite of MPI_Gather

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

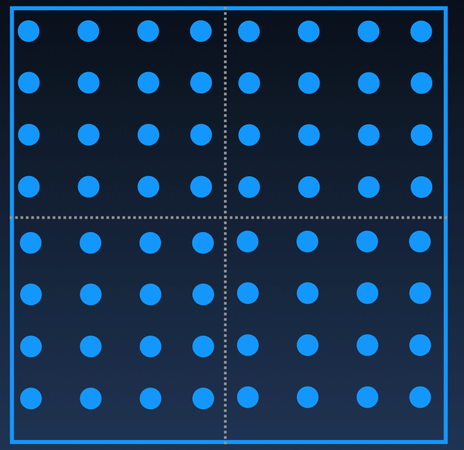
- Similar to non-roots sending:
  - MPI_Send(sendbuf, sendcount, sendtype, root, ...),
- and the root receiving n times:
  - MPI_Recv(recvbuf + i * recvcount *extent(recvtype), recvcount, recvtype, i, ...)



- **MPI_Gatherv** allows different size data to be gathered
- **MPI_Allgather** has no *root*; all nodes receive similarly

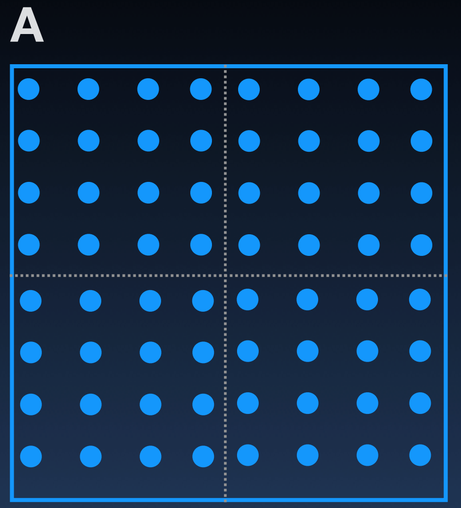
Scatter Matrix

A



Subodh Kumar

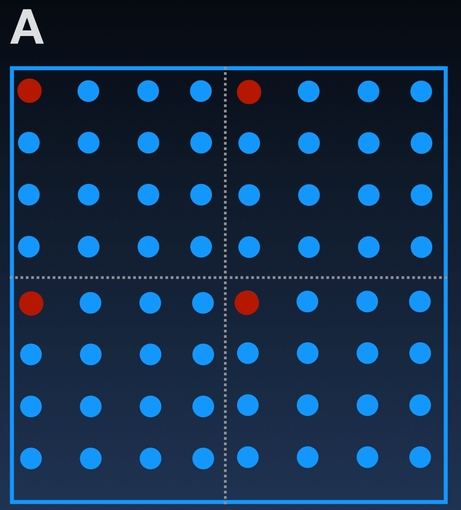
Scatter Matrix




```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

Scatter Matrix

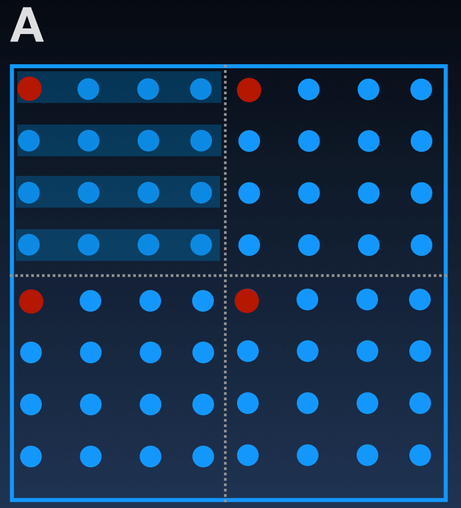



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

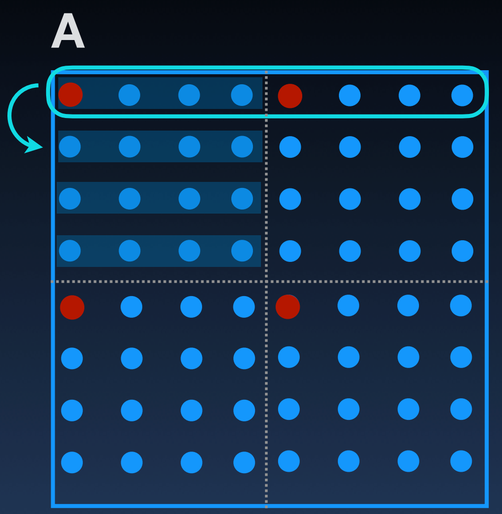
Scatter Matrix

```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```



Subodh Kumar

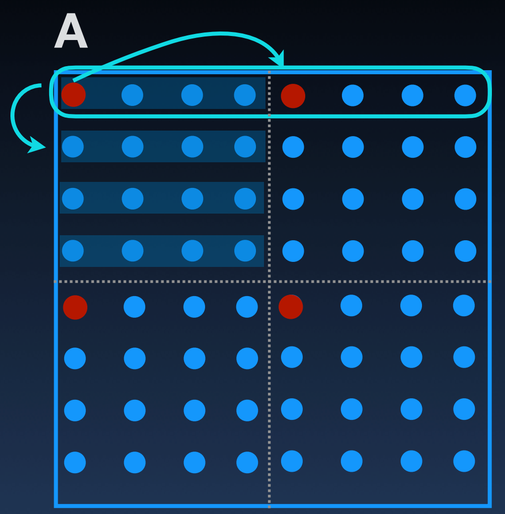
Scatter Matrix




```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

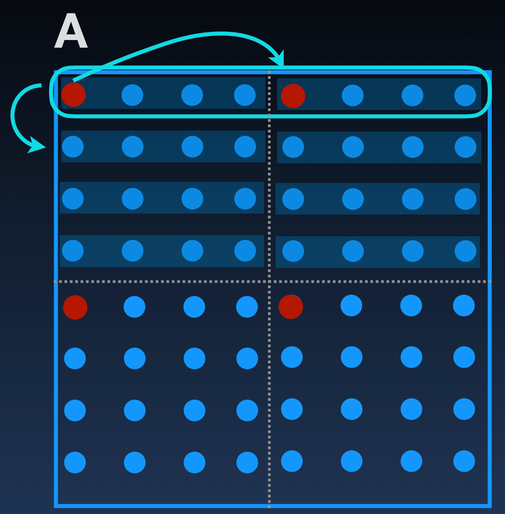
Scatter Matrix




```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

Scatter Matrix

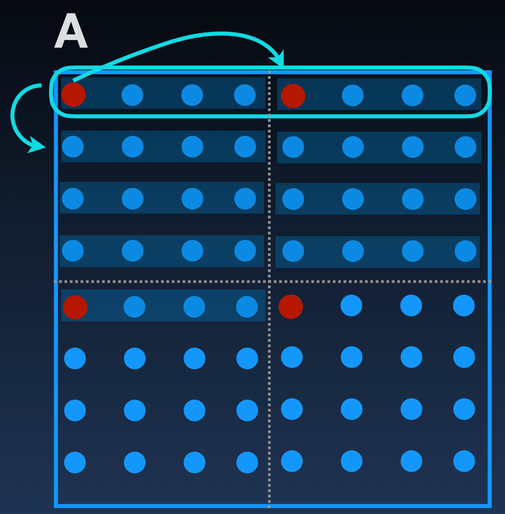


second stype element

```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Scatter Matrix

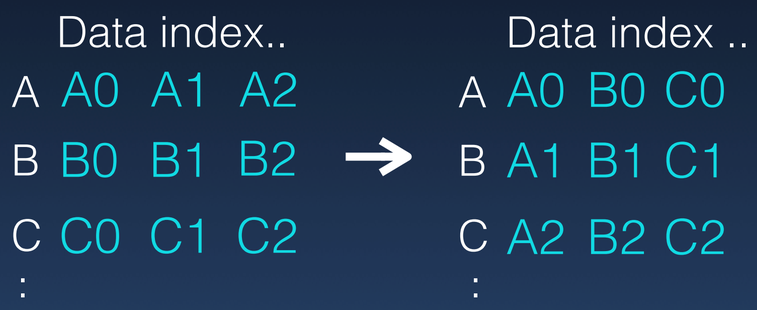
```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9; / Send from offsets 0,1,8,9
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                          alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

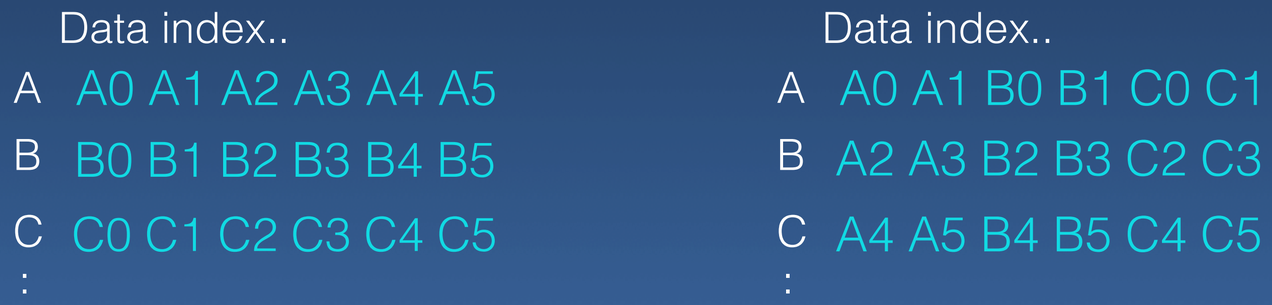


Red marks start of stype elements 0, 1, 8, 9 resp.

Subodh Kumar

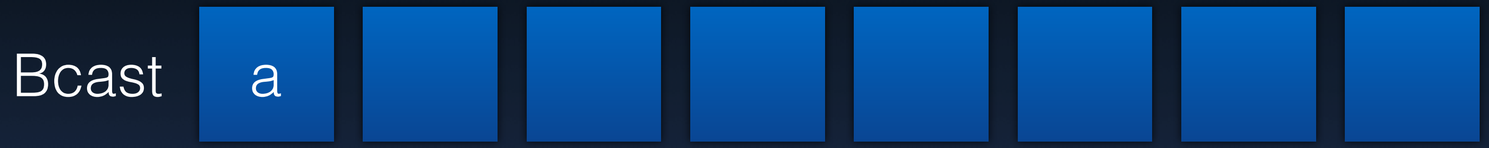
All to All



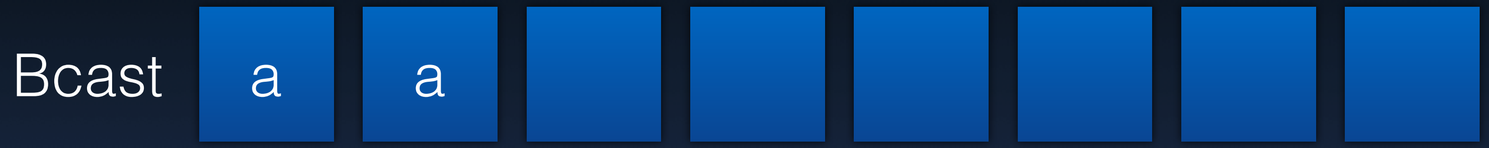


Can all-to-all multiple data items

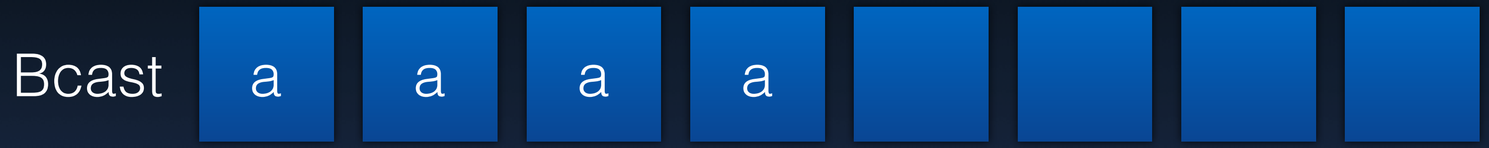
Collective Implementation



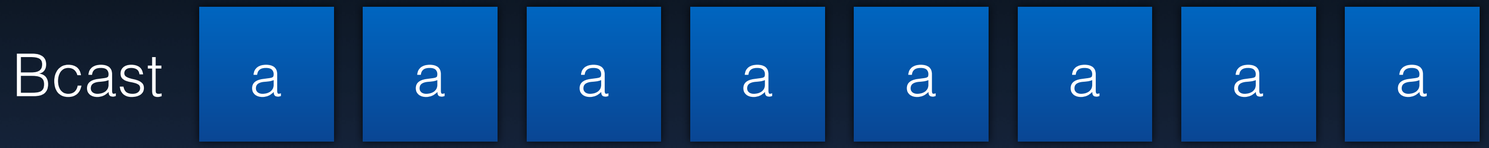
Collective Implementation



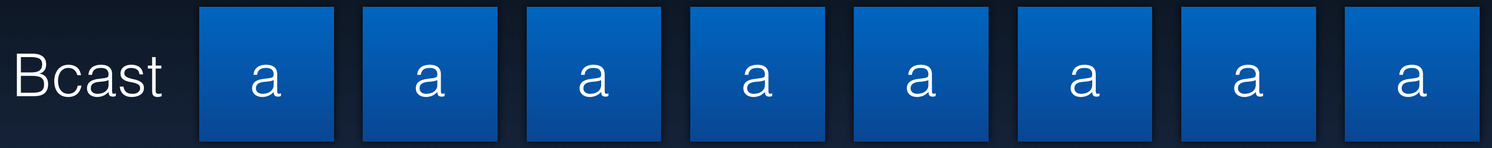
Collective Implementation



Collective Implementation

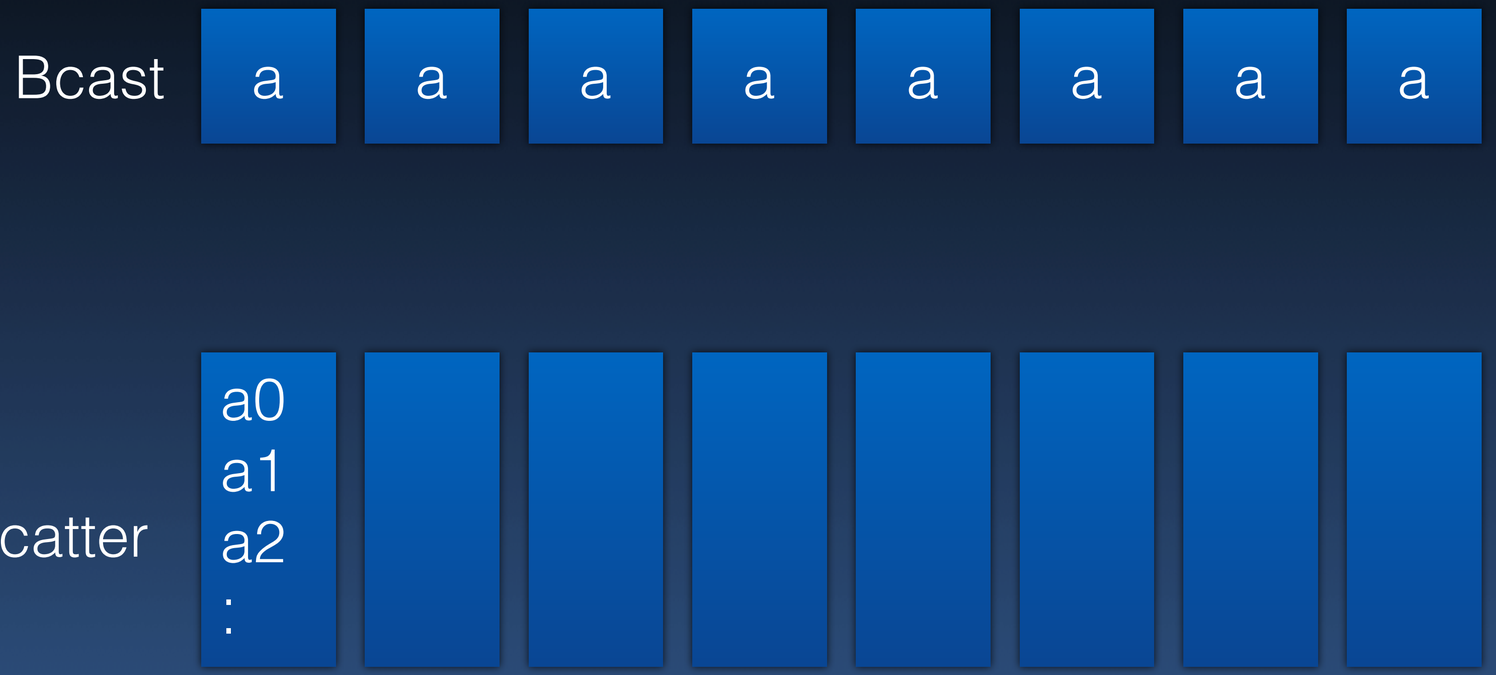


Collective Implementation



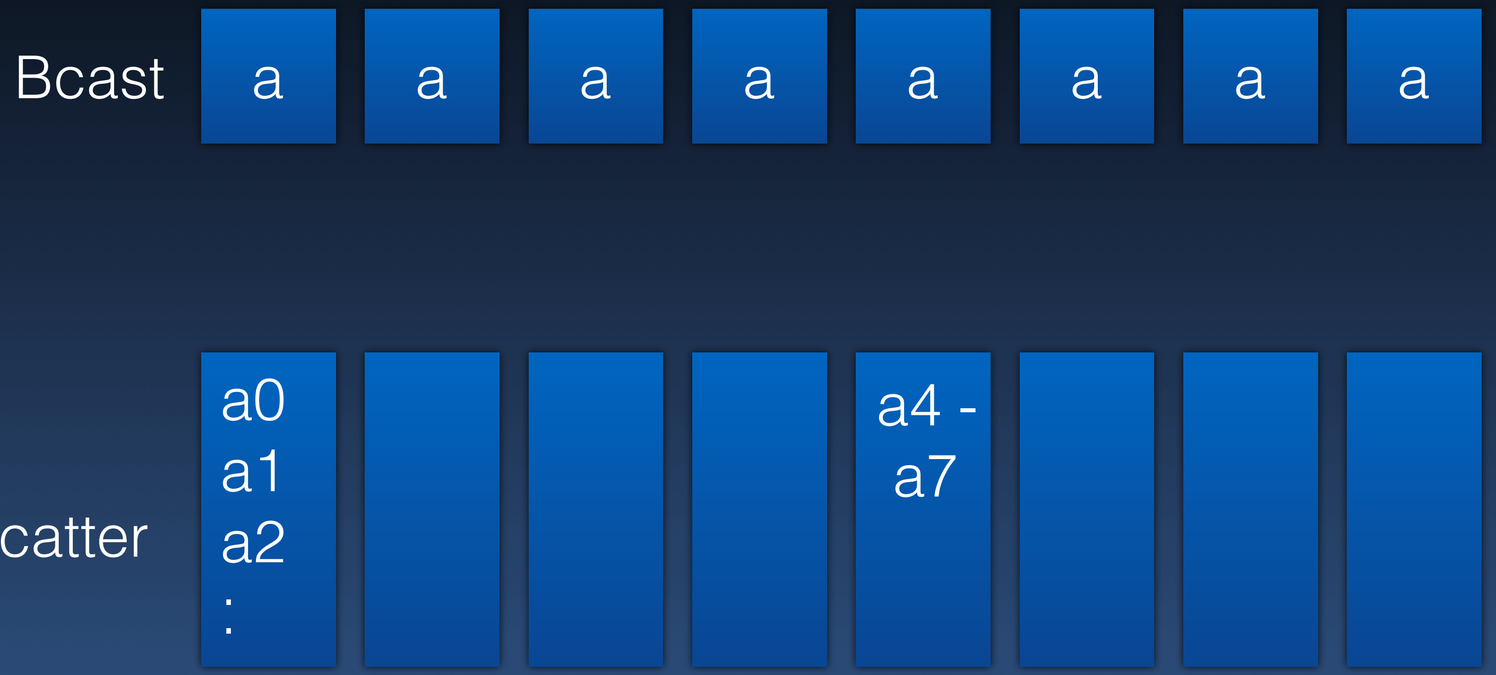
log P rounds
1 message/round/pair
of ‘unit’ size

Collective Implementation



log P rounds
1 message/round/pair
of ‘unit’ size

Collective Implementation



log P rounds
1 message/round/pair
of ‘unit’ size

Collective Implementation

log P rounds
1 message/round/pair
of ‘unit’ size

Collective Implementation

log P rounds
1 message/round/pair
of ‘unit’ size

log P rounds
1 message/round/pair
of P/2, P/4, P/8 .. units

Collective Implementation

log P rounds
1 message/round/pair
of ‘unit’ size

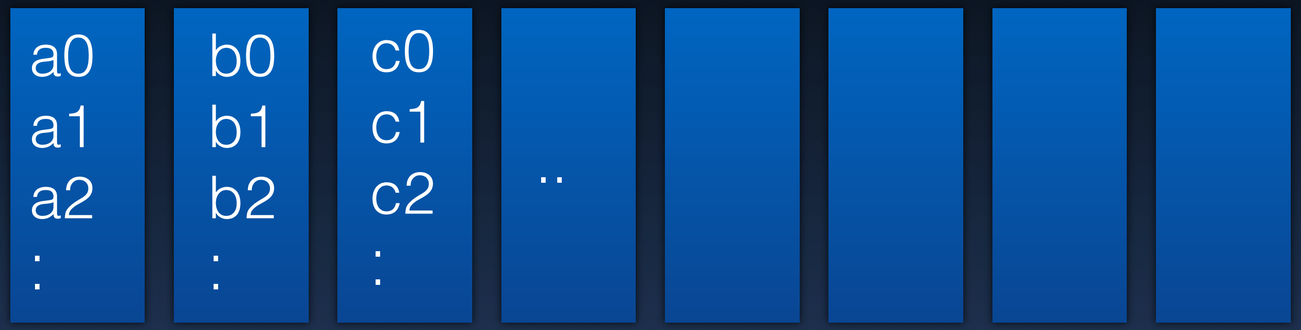
log P rounds
1 message/round/pair
of P/2, P/4, P/8 .. units

```
r = 2⌈log n⌉
while(r > 1):
    if( PID & (r-1) == 0)
        Send items[r/2:end] to PID+r/2 (match recv)
    r  /= 2;
```

Subodh Kumar

Collective Communication

All to All



P sequential Scatters?

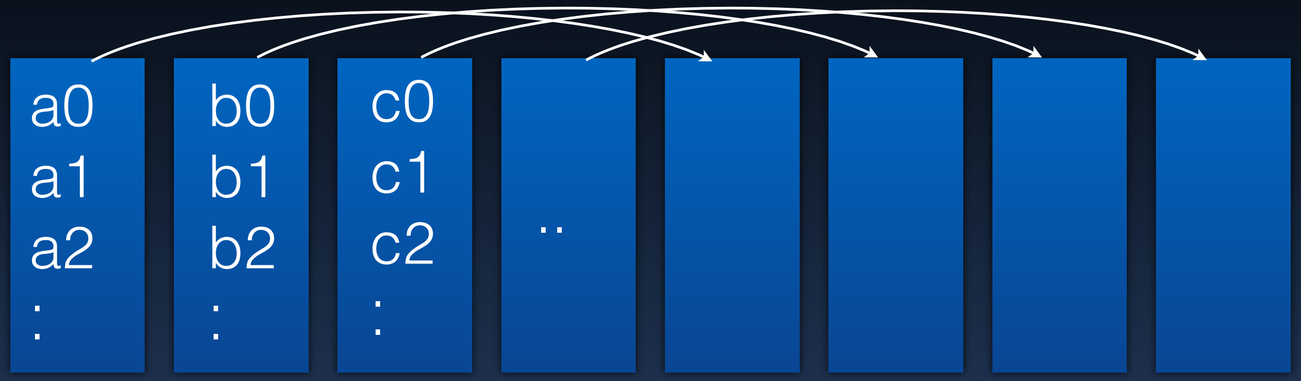
Collective Communication

All to All

P sequential Scatters?

Collective Communication

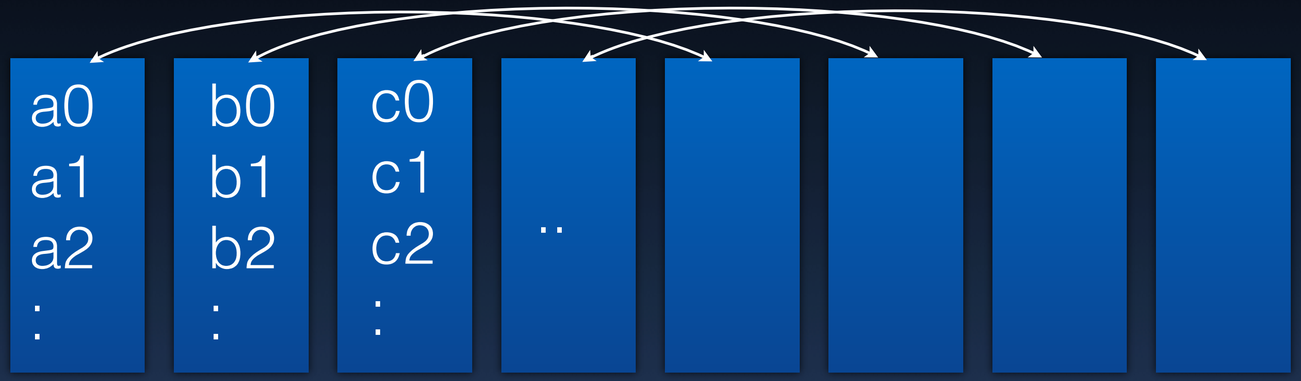
All to All



P sequential Scatters?

Collective Communication

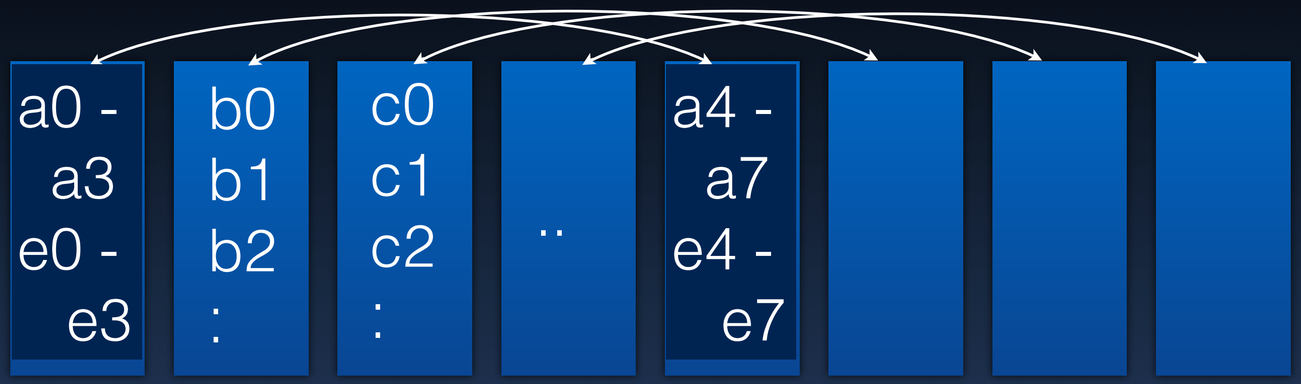
All to All



P sequential Scatters?

Collective Communication

All to All



P sequential Scatters?

Collective Communication

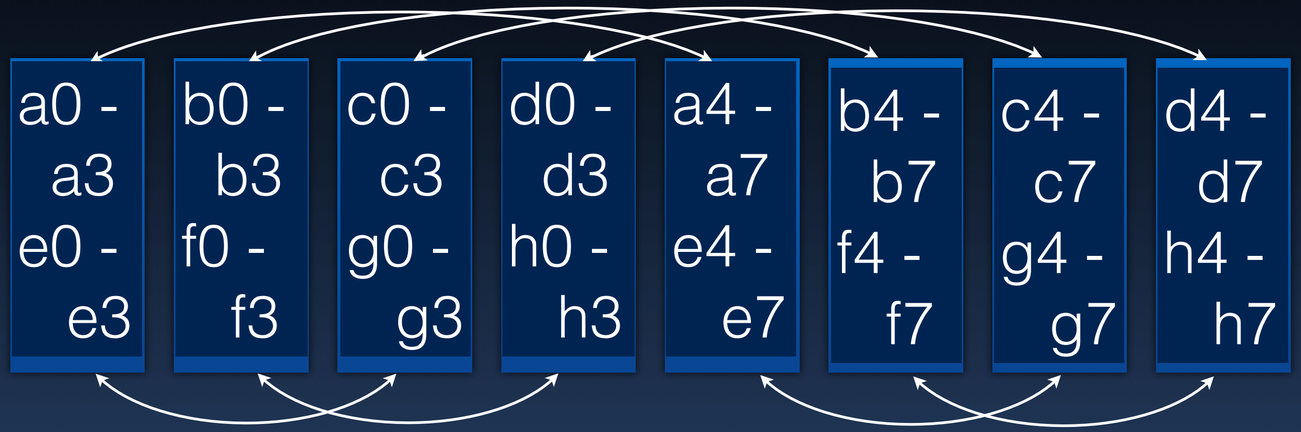
All to All

P sequential Scatters?

- P/2 pairs exchange ½ their data (first/second half)
  - ➡ each P/2 apart

Collective Communication

All to All



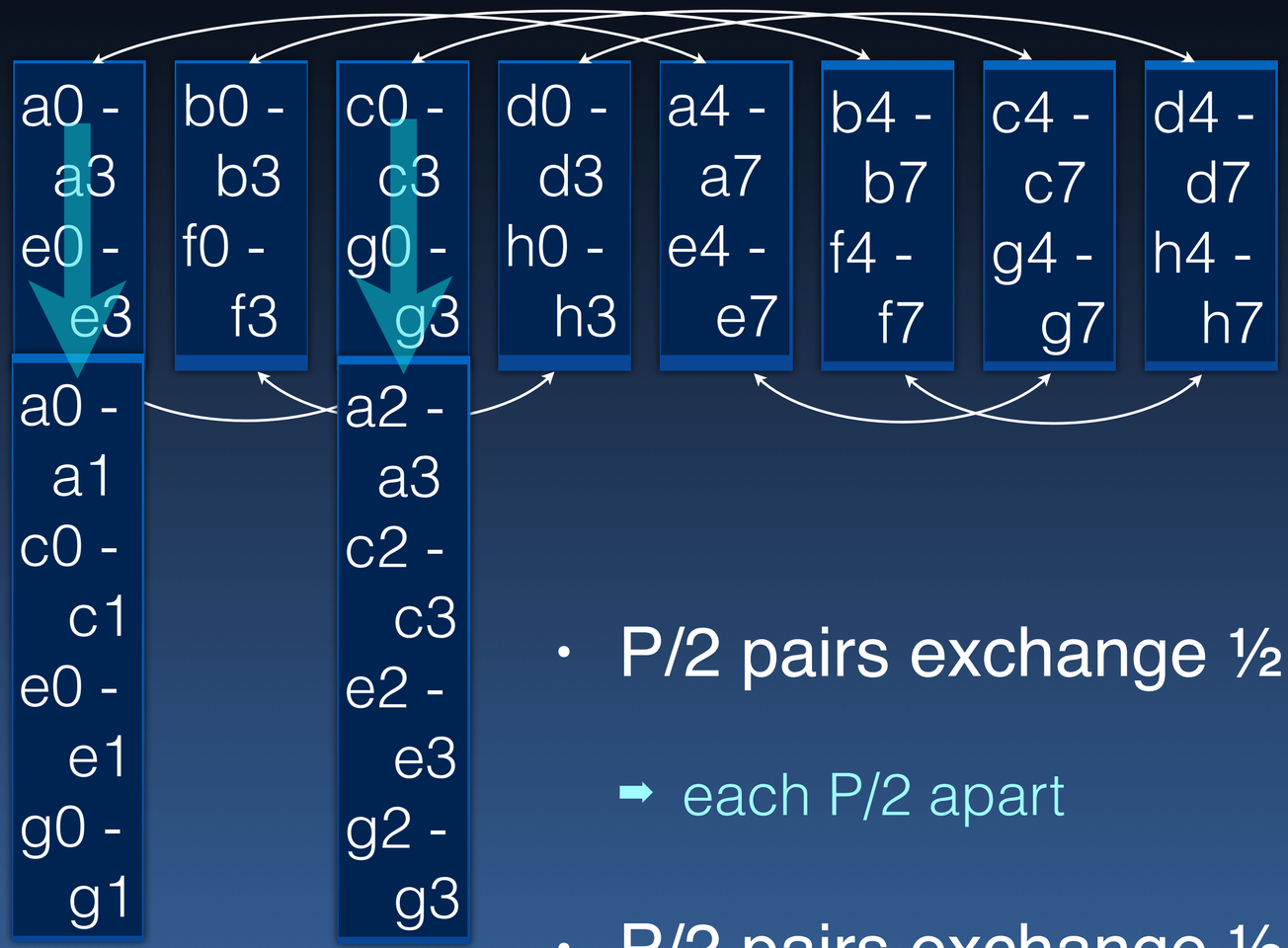
P sequential Scatters?

- P/2 pairs exchange ½ their data (first/second half)

  ➡ each P/2 apart

Collective Communication

All to All

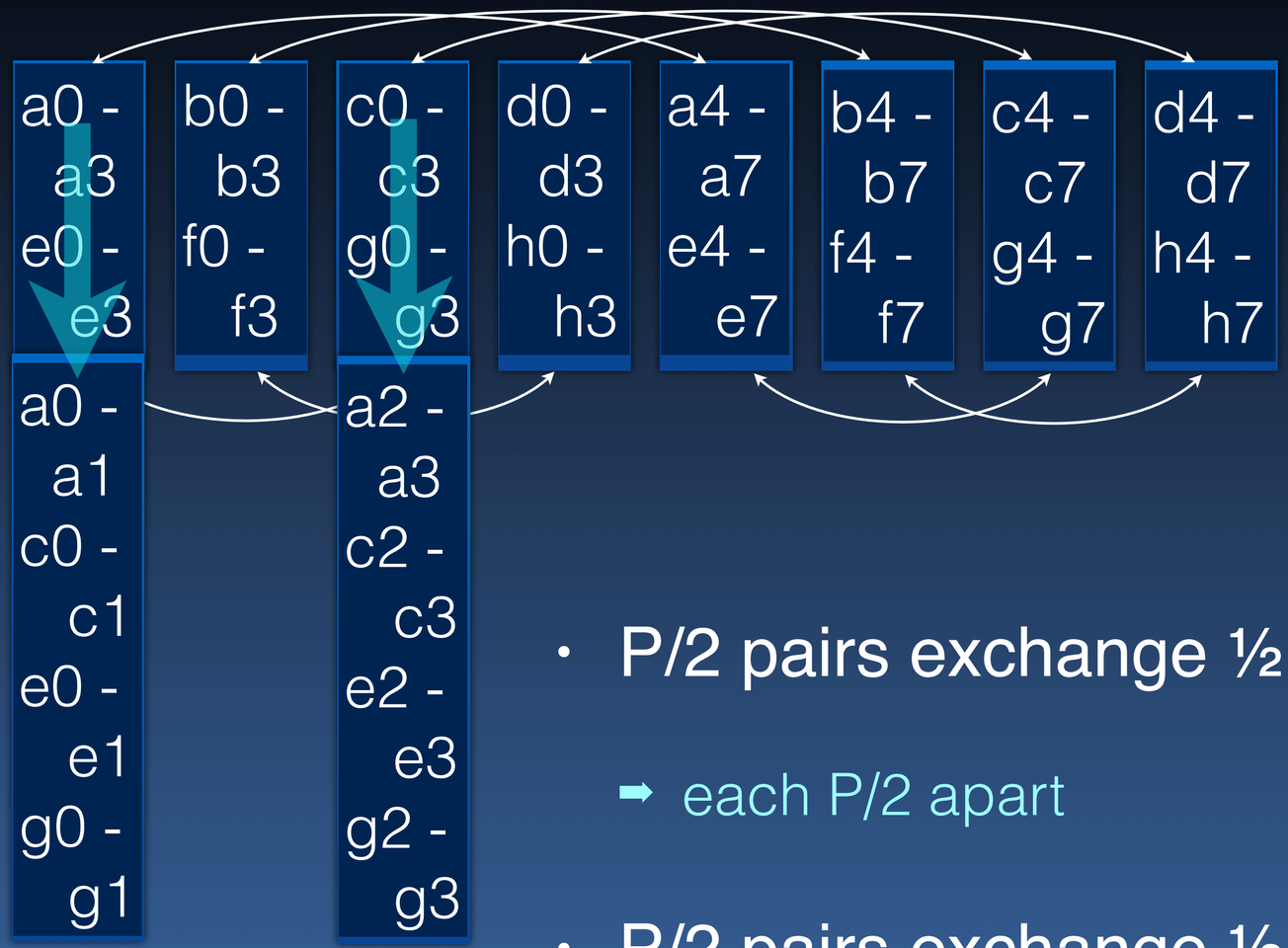


P sequential Scatters?

- P/2 pairs exchange ½ their data (first/second half)
  - ➡ each P/2 apart
- P/2 pairs exchange ½ of each ½ assembled earlier
  - ➡ each P/4 apart

Subodh Kumar

All to All



~~P sequential Scatters?~~

log P rounds
1 message/round/pair

of P units

- P/2 pairs exchange ½ their data (first/second half)
  - ➡ each P/2 apart
- P/2 pairs exchange ½ of each ½ assembled earlier
  - ➡ each P/4 apart

```
MPI_Win_create(addr, size, displ_unit, info, MPI_COMM_WORLD, &win);
…
MPI_Win_free(&win);
```

MPI_Info

MPI_Win

- Weak synchronization
- Collective call
- Info specifies system-specific information (e.g., memory locking)
  - ➡ Designed to optimize performance
- Also see MPI_Alloc_mem/MPI_Win_allocate for <addr> allocation (RMA friendly)

MPI_Put, MPI_Get

- MPI_Put(my_addr, my_count, my_datatype,
  there_rank, there_disp, there_count, there_datatype, win);
  - ➡ Written in the dest window-buffer at address
    - ▸ window_base + disp×disp_unit
  - ➡ Must fit in the target buffer
  - ➡ there_datatype defined on the “putter”
    - ▸ But refers to memory “there”
    - ▸ Usually defined on both sides

MPI_Put, MPI_Get

- MPI_Put(my_addr, my_count, my_datatype,
  there_rank, there_disp, there_count, there_datatype, win);
  - ➡ Written in the dest window-buffer at address
    - ▸ window_base + disp×disp_unit
  - ➡ Must fit in the target buffer
  - ➡ there_datatype defined on the “putter”
    - ▸ But refers to memory “there”
    - ▸ Usually defined on both sides

MPI_Get does the reverse: there ➜ my

Also see:
MPI_Accumulate
performs an “op” at destination

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

```
int winbuf[1024];
MPI_Win windo;
MPI_Win_create(winbuf, 1024*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

if(rank == 1) {
        int lbuf[5];
        initialize(lbuf);
        MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo); // Wait for MPI_Put complete

if(my_rank == 0)
        use(winbuf+5);
```

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

```
int winbuf[1024];
MPI_Win windo;
MPI_Win_create(winbuf, 1024*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

if(rank == 1) {
        int lbuf[5];
        initialize(lbuf);
        MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo); // Wait for MPI_Put complete

if(my_rank == 0)
        use(winbuf+5);
```

"Assertion" by program

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

Look these up

```
int winbuf[1024];
MPI_Win windo;
MPI_Win_create(winbuf, 1024*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

if(rank == 1) {
        int lbuf[5];
        initialize(lbuf);
        MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo); // Wait for MPI_Put complete

if(my_rank == 0)
        use(winbuf+5);
```

"Assertion" by program

Review

- Semantics of collective calls (and their synchronization)
- Using Types (and resized types) to control data scatter/gather
- Shared memory semantics using one-sided communication