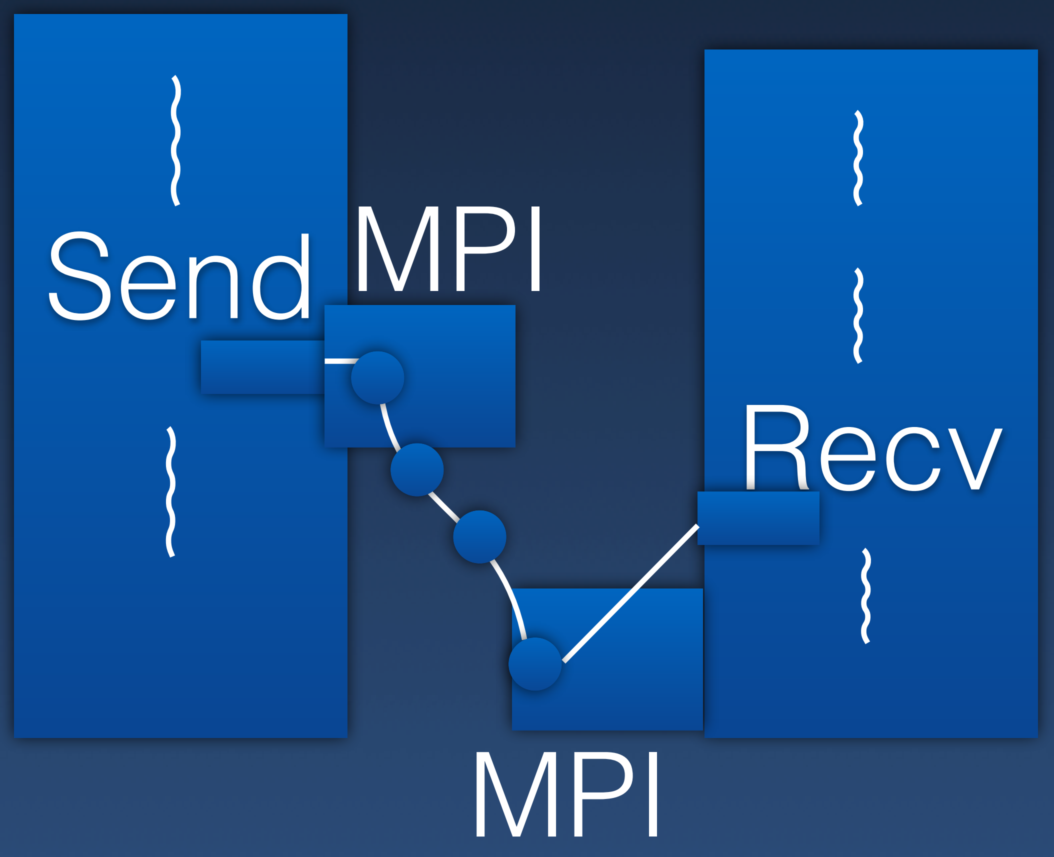
COL380

Introduction to
Parallel & Distributed Programming

Message Semantics

- Variables, Buffers, and Packets
  - ➡ Application to application
- Lossy?
  - ➡ Deal with loss
  - ➡ Acks
- FIFO?
- Point to Point vs Collective?
- Addressing?



- MPI is for inter-process communication
  - ➡ Process creation
  - ➡ Data communication (Buffering, Book-keeping ..)
  - ➡ Synchronization
- Allows
  - ➡ Synchronous communication
  - ➡ Asynchronous communication
    - compare to shared memory

MPI

- MPI is for inter-process communication
  - ➡ Process creation
  - ➡ Data communication (Buffering, Book-keeping ..)
  - ➡ Synchronization

Functions, Types, Constants

- Allows
  - ➡ Synchronous communication
  - ➡ Asynchronous communication
    - compare to shared memory

MPI

- MPI is for inter-process communication
  - ➡ Process creation
  - ➡ Data communication (Buffering, Book-keeping ..)
  - ➡ Synchronization

Functions, Types, Constants

- Allows
  - ➡ Synchronous communication
  - ➡ Asynchronous communication
    - compare to shared memory

- High-level constructs
  - broadcast, reduce, scatter/gather message
  - Collective functions
- Interoperable across architectures

- Compile: `mpiCC -o exec code.cpp`
    - script to compile and link
    - Automatically adds include, library flags
- Run:
    - mpirun -host host1,host2 exec args
    - Or, use hostfile
- Useful:
    - mpirun -mca <key> <value>

Running MPI Programs

- Compile: mpiCC -o exec code.cpp
  - script to compile and link
  - Automatically adds include, library flags
- Run:
  - mpirun -host host1,host2 exec args
  - Or, use hostfile
- Useful:
  - mpirun -mca <key> <value>

- mpirun -mca mpi_show_handle_leaks 1
- mpirun -mca btl openib,tcp
- mpirun -mca btl_tcp_min_rdma_size
- Check out "ompi_info"

Subodh Kumar

- Key based remote shell execution
- Use ssh-keygen to create public-private key pair
  - ➡ Private key stays in subdirectory ~/.ssh on your client
  - ➡ Public key on server in ~/.ssh/authorized_keys
  - ➡ Test: ‘ssh <server> ls’ works
  - ➡ On HPC, client and server share the same home directory
- PBS automatically creates appropriate host files
  - ➡ See also: -l select=2:ncpus=1:mpiprocs=1 -l place=scatter

- Communicator
  - ➡ Groups of processes sharing a context
  - ➡ Intra and inter-communicator
- Context
  - ➡ “communication universe”
  - ➡ Messages across context have no ‘interference’
- Groups
  - ➡ Collection of processes (can build hierarchy)
  - ➡ Ordered

Process Organization

- Communicator
  - ➡ Groups of processes sharing a context
  - ➡ Intra and inter-communicator

Predefined constant: MPI_COMM_WORLD

- Context
  - ➡ “communication universe”
  - ➡ Messages across context have no ‘interference’
- Groups
  - ➡ Collection of processes (can build hierarchy)
  - ➡ Ordered

Process Organization

- Communicator
  - ➡ Groups of processes sharing a context
  - ➡ Intra and inter-communicator

Predefined constant: MPI_COMM_WORLD

- Context
  - ➡ “communication universe”
  - ➡ Messages across context have no ‘interference’
- Groups
  - ➡ Collection of processes (can build hierarchy)
  - ➡ Ordered

Use group-rank to address

Process Organization

- Communicator Topology
  - ➡ Groups of processes sharing a context
  - ➡ Intra and inter-communicator Predefined constant: MPI_COMM_WORLD
- Context
  - ➡ “communication universe”
  - ➡ Messages across context have no ‘interference’
- Groups
  - ➡ Collection of processes (can build hierarchy)
  - ➡ Ordered Use group-rank to address

Example

```
#include "mpi.h"      /* includes MPI library code specs */

#define MAXSIZE 100

int main(int argc, char* argv[])
{
  MPI_Init(&argc, &argv);                       // start MPI
  int nProcs, myRank, dat[2] = {5,6};
  MPI_Status status;
  MPI_Comm_size(MPI_COMM_WORLD, &nProcs); // Group size
  MPI_Comm_rank(MPI_COMM_WORLD, &myRank); // get my rank
  If(myRank == 0)
      MPI_Send(dat, 2, MPI_INT, nProcs-1, 11, MPI_COMM_WORLD);
  If(myRank == nProcs-1)
      MPI_Recv(dat, 9, MPI_INT, 0, 11, MPI_COMM_WORLD, &status);
  MPI_Finalize();                              // stop MPI
}
```

Example

```
#include "mpi.h"      /* includes MPI library code specs */

#define MAXSIZE 100

int main(int argc, char* argv[])
{
  MPI_Init(&argc, &argv);                    // start MPI
  int nProcs, myRank, dat[2] = {5,6};
  MPI_Status status;
  MPI_Comm_size(MPI_COMM_W
  MPI_Comm_rank(MPI_COMM_W
  If(myRank == 0)
      MPI_Send(dat, 2, MPI                                LD);
  If(myRank == nProcs-1)
      MPI_Recv(dat, 9, MPI_INT, 0, 11, MPI_COMM_WORLD, &status);
  MPI_Finalize();                            // stop MPI
}
```

```
status.MPI_TAG

status.MPI_SOURCE

MPI_Get_count(&status, MPI_INT,  &count);
```

Starting and Ending

- MPI_Init(&argc, &argv);

MPI_Init_thread

  - ➡ Needed before any other MPI call

```
int nump, id;
MPI_Comm_size (MPI_COMM_WORLD, &nump);
MPI_Comm_rank (MPI_COMM_WORLD, &id);
```

- MPI_Finalize();
  - ➡ Required

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest,
    int tag, MPI_Comm comm)
```

• message contents	block of memory
• count	number of items in message
• message type	MPI_Datatype of each item
• destination	rank of recipient
• tag	integer “message identifier”
• communicator	

Blocking calls

Send/Receive

int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

MATCHING (Per context)

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

s block of memory
number of items in messag
MPI_Datatype of each iten
rank of recipient
integer “message identifier

message contents	memory buffer to store received message
count	space in buffer, overflow error if too small
message type	type of each item
source	sender’s rank (or MPI_ANY_SOURCE)
tag	message identifier (or MPI_ANY_TAG)
communicator	
status	information about message received

Eager vs Rendezvous

Eager

- Send-stub packetizes and transmits
  (May save a local message copy)
- Send-stub signals Done
- Recv-stub continuously accepts
- Delivered when Recv call matches

Rendezvous

- Send-stub transmits envelope info
  (May save local message copy)
- Recv-stub continuously accepts envelope info
- Recv-stub may signal OK (if it has space)
  Or, wait for matching Recv call to be made
- Recv-stub sets up “RDMA” with Send-stub
- Data transmitted
- Recv-stub signals Done
- Send-stub signals Done

Subodh Kumar

Send/Recv Synchronization

- Blocking
  - ➡ Send returns after some progress guarantee
    - Receive completed?
    - Synchronization (up to network delay)
- Immediate
  - ➡ Send returns with no progress guarantee
  - ➡ Receiver may also proceed immediately (message arrives later)

Send Semantics

- Standard mode:
  - ➡ implementation dependent
- Buffered mode
  - ➡ MPI saves a copy of message, Receiver can post later
  - ➡ User provided buffer
- Synchronous mode
  - ➡ Will complete only once a matching receive has started
- Ready mode
  - ➡ Send may start only if a matching receive has already been called
  - ➡ Helps performance

Subodh Kumar

Send Semantics

- MPI_Send/MPI_Recv are blocking
  - ➡ Recv blocks until output buffer is filled
  - ➡ Send blocks until some 'progress'

- Standard mode:
  - ➡ implementation dependent
- Buffered mode
  - ➡ MPI saves a copy of message, Receiver can post later
  - ➡ User provided buffer
- Synchronous mode
  - ➡ Will complete only once a matching receive has started
- Ready mode
  - ➡ Send may start only if a matching receive has already been called
  - ➡ Helps performance

Send Semantics

- Standard mode:
  - ➡ implementation dependent
- Buffered mode
  - ➡ MPI saves a copy of message, Receiver can post later
  - ➡ User provided buffer

See MPI_Buffer_attach

- Synchronous mode
  - ➡ Will complete only once a matching receive has started
- Ready mode
  - ➡ Send may start only if a matching receive has already been called
  - ➡ Helps performance

- MPI_Send/MPI_Recv are blocking
  - ➡ Recv blocks until output buffer is filled
  - ➡ Send blocks until some 'progress'

Send Semantics

- Standard mode: MPI_Send
  - ➡ implementation dependent
- Buffered mode MPI_Bsend
  - ➡ MPI saves a copy of message, Receiver can post later
  - ➡ User provided buffer See MPI_Buffer_attach
- Synchronous mode MPI_Ssend
  - ➡ Will complete only once a matching receive has started
- Ready mode MPI_Rsend
  - ➡ Send may start only if a matching receive has already been called
  - ➡ Helps performance

- MPI_Send/MPI_Recv are blocking
  - ➡ Recv blocks until output buffer is filled
  - ➡ Send blocks until some 'progress'

Subodh Kumar

Message Semantics

- In order (per pair and tag)
  - ➡ Multi-threaded applications need to be coordinate
- Progress
  - ➡ For a matching send/Recv pair, at least one of these two will complete
- Fairness not guaranteed
  - ➡ A Send or a Recv may starve because all matches are satisfied by others
- Resource limitation can cause deadlocks
- Ready/Synchronous sends requires the least resources
  - ➡ Also used for debugging

Example

If (rank == 0):

```
Send(sbuffer0, to 1);
Recv(rbuffer0, from 1);
```

else:

```
Send(sbuffer1, to 0);
Recv(rbuffer1, from  0);
```

match

match

Deadlock

if neither send can copy out its sbuffer

Non-blocking Call

int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Non-blocking calls

Returns even before buf copied out; caller must not use.

Example

If (rank == 0):

MPI_Isend(sbuffer0, to 1);

MPI_Irecv(rbuffer0, from 1);

Wait for earlier calls to finish

Will not deadlock

else:

MPI_Isend(sbuffer1, to 0);

MPI_Irecv(rbuffer1, from 0);

Wait for earlier calls to finish

Detecting Completion

- MPI_Wait(&request, &status)
  - status similar to MPI_recv
  - Blocks as per the blocking version's semantics
    - Send: message was copied out, Recv was started, etc.
    - Recv: Wait for data to fill
  - Request is freed as a side-effect
- MPI_Test(&request, &flag, &status)
  - Non-blocking poll
  - flag indicates whether operation is complete
  - Request is freed as a side-effect

Use MPI_Request_get_status to retain request

Later MPI_Request_free

Subodh Kumar

Detecting Completion

- MPI_Wait(&request, &status)
  - status similar to MPI_recv
  - Blocks as per the blocking version's semantics
    - Send: message was copied out, Recv was started, etc.
    - Recv: Wait for data to fill
  - Request is freed as a side-effect
- MPI_Test(&request, &flag, &status)
  - Non-blocking poll
  - flag indicates whether operation is complete
  - Request is freed as a side-effect

Also see:

MPI_Waitany, MPI_Waitall, MPI_Waitsome

MPI_Testany, MPI_Testall, MPI_Testsome

Subodh Kumar

Point to Point Basics

- Send - Recv is point-to-point
  - ➡ Call-to-call matching
  - ➡ Integer tag to control matching
  - ➡ Wildcard matching: MPI_ANY_SOURCE and MPI_ANY_TAG
- Recv buffer must contain enough space for message
  - ➡ Receiving fails otherwise
  - ➡ Can query the actual count received (MPI_Get_count)
    - Send determines the actual number sent
  - ➡ type parameters determines data structure ↔ message buffer copying

MPI Data types

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	

Objects of type MPI_Datatype

- MPI does not understand language's layout (struct, e.g.)
  - ➡ Too system architecture dependent

MPI_INT, MPI_FLOAT ..

- Typemap:
  - ➡ (type_0, disp_0), ..., (type_n, disp_n)
  - ➡ i^{th} entry is of type_i and starts at byte base + disp_i

```
MPI_Datatype newtype;
MPI_Type_contiguous(count, MPI_INT, &newtype);
```

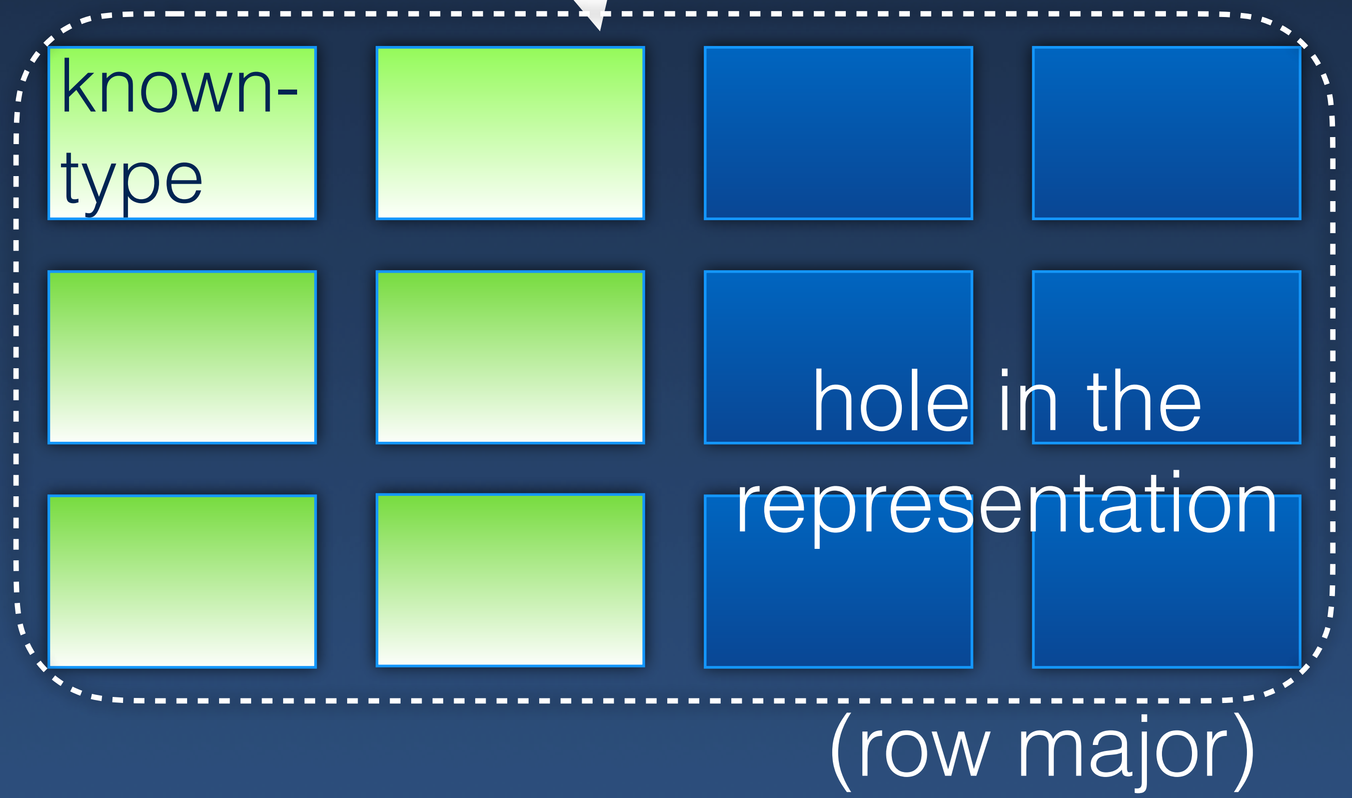
Blocks

- Equally-spaced blocks of the known datatype

➡ MPI_Type_vector(blockcount, blocklength, blockstride, knowntype, &newtype);

(with blockcount = 3, blocklength = 2, blockstride = 4)

- Assume contiguous copies of 'knowntype'
- Stride between blocks specified in units of knowntype
- All picked blocks are of the same length



➡ MPI_Type_create_hvector(blk_count, blk_length, bytestride, knowntype, &newtype);

Gap between blocks is in bytes

- **MPI_Type_indexed**(count, array_of_blocklengths, array_of_offsets, knowntype, &newtype);

  count: **5**; array_of_blocklengths: **2,1,1,1,2**; array_of_offsets: **0,4,6,8,10**

  - ➡ Blocks can contain different number of copies
  - ➡ And may have different strides
  - ➡ But the same data type

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26}; 
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26}; 
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

MPI_Type_get_contents(..)

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &beg, &extent);
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

Data Type Functions

- MPI_Type_commit(&datatype)
    - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
    - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &b
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

```
MPI_Datatype atype;
MPI_Type_contiguous(4, MPI_CHAR, &atype);
int asize;
MPI_Type_size(atype, &asize);
MPI_Type_commit(&atype);
MPI_Send(buf, nItems, atype, dest, ..);
MPI_Recv(...);
```

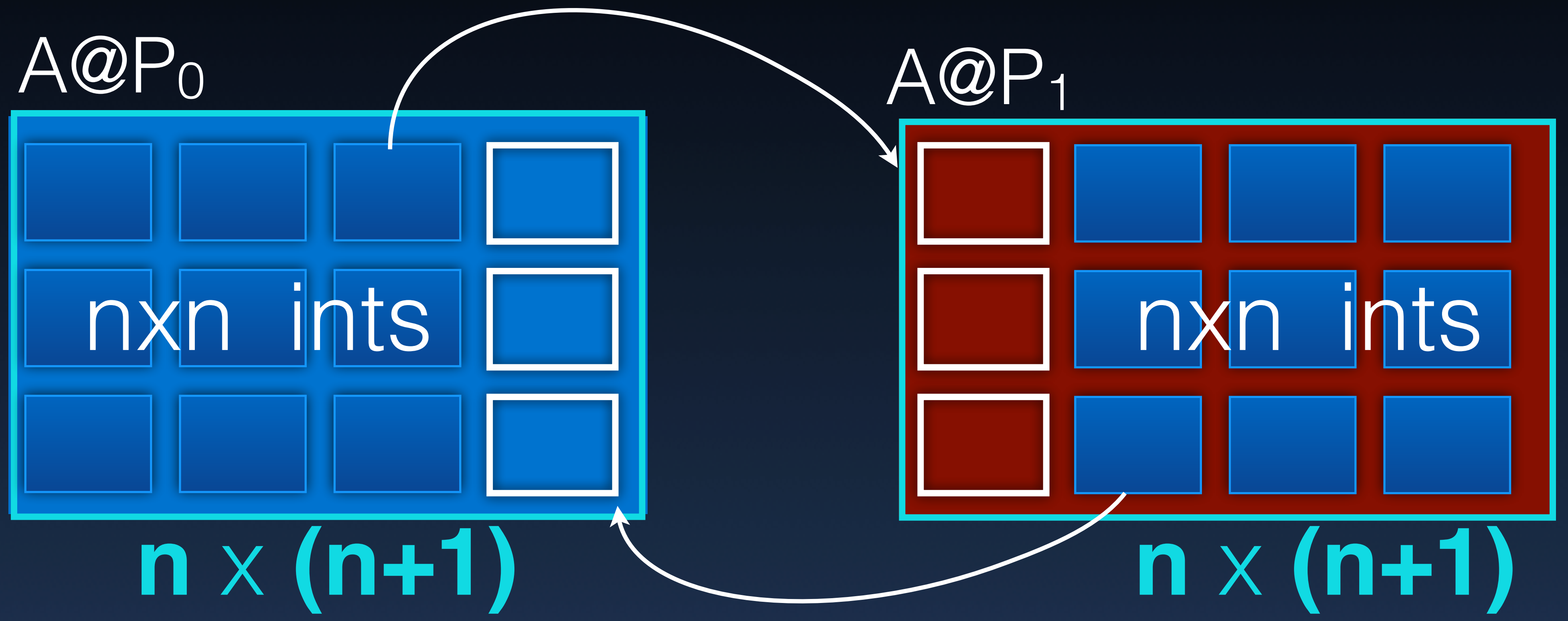
Derived Type Example

```
sendParticles(struct Particle particle[], int N):
    MPI_Datatype Particletype;
    MPI_Datatype types[3] = {MPI_INT, MPI_DOUBLE, MPI_CHAR};
    int blockcount[3] = {1, 6, 7};

    /* compute displacements of structure components */
    MPI_Aint disp[3];
    MPI_Address(particle, disp);
    MPI_Address(particle[0].d, disp+1);
    MPI_Address(particle[0].b, disp+2);
    for (int i=2; i >= 0; i--) disp[i] -= disp[0];

    MPI_Type_struct(3, blockcount, disp, types, &Particletype);
    MPI_Type_commit( &Particletype);
    MPI_Send(particle, N, Particletype, dest, tag, comm);
```

```
struct Particle
{
   int class;       // particle class
   double d[6]; // particle coordinates
   char b[7];     // some additional info
};
```

Subodh Kumar

Data Transfer




```
MPI_Status status;
MPI_Datatype column;
MPI_Type_vector(n, 1, n+1, MPI_INT, &column);
MPI_Type_commit(&column);
if(rank == 0) {
   MPI_Send(A+n-1, 1, column, 1, tag, MPI_COMM_WORLD);
   MPI_Recv(A+n, 1, column, 1, tag, MPI_COMM_WORLD, &status);
}
if(rank == 1) {
   MPI_Recv(A, 1, column, 0, tag, MPI_COMM_WORLD, &status);
   MPI_Send(A+1, 1, column, 0, tag, MPI_COMM_WORLD);
}
```

Subodh Kumar

- MPI_Barrier
  - Barrier synchronization across all members of a group
- MPI_Bcast
  - Broadcast from one member to all members of a group
- MPI_Scatter, MPI_Gather, MPI_Allgather
  - Gather data from all members of a group to one
- MPI_Alltoall
  - complete exchange or all-to-all
- MPI_Reduce, MPI_Allreduce,
  - Reduction operations
- MPI_Reduce_Scatter
  - Combined reduction and scatter operation
- MPI_Scan, MPI_Exscan
  - Prefix

Barrier

- Synchronization of the calling processes
  - – the call blocks until all of the processes have placed the call

```
MPI_Barrier(comm);
```

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

	Data index 0-2		
rank A	A0	A1	A2
rank B	?	?	?
rank C	?	?	?
rank D	?	?	?

→

	Data index 0-2		
A	A0	A1	A2
B	A0	A1	A2
C	A0	A1	A2
D	A0	A1	A2

Broadcast

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

pointer on all

number & type

identified sender

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

Broadcast

See: MPI_Reduce, MPI_Scan

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

pointer on all

number & type

identified sender

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

Subodh Kumar

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

	Data index..			→		Data index ..					
A	A0	A1	A2		A	A0	A1	A2	B0	B1	··
B	B0	B1	B2		B	B0	B1	B2			
C	C0	C1	C2		C	C0	C1	C2			
D	D0	D1	D2		D	D0	D1	D2			
:					:						

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, ...)

	Data index..		
A	A0	A1	A2
B	B0	B1	B2
C	C0	C1	C2
D	D0	D1	D2
:			

→

	Data index ..					
A	A0	A1	A2	B0	B1	··
B	A0	A1	A2	B0	B1	··
C	A0	A1	A2	B0	B1	··
D	A0	A1	A2	B0	B1	··
:						

(all gather)

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

Subodh Kumar

MPI_Scatter is opposite of MPI_Gather

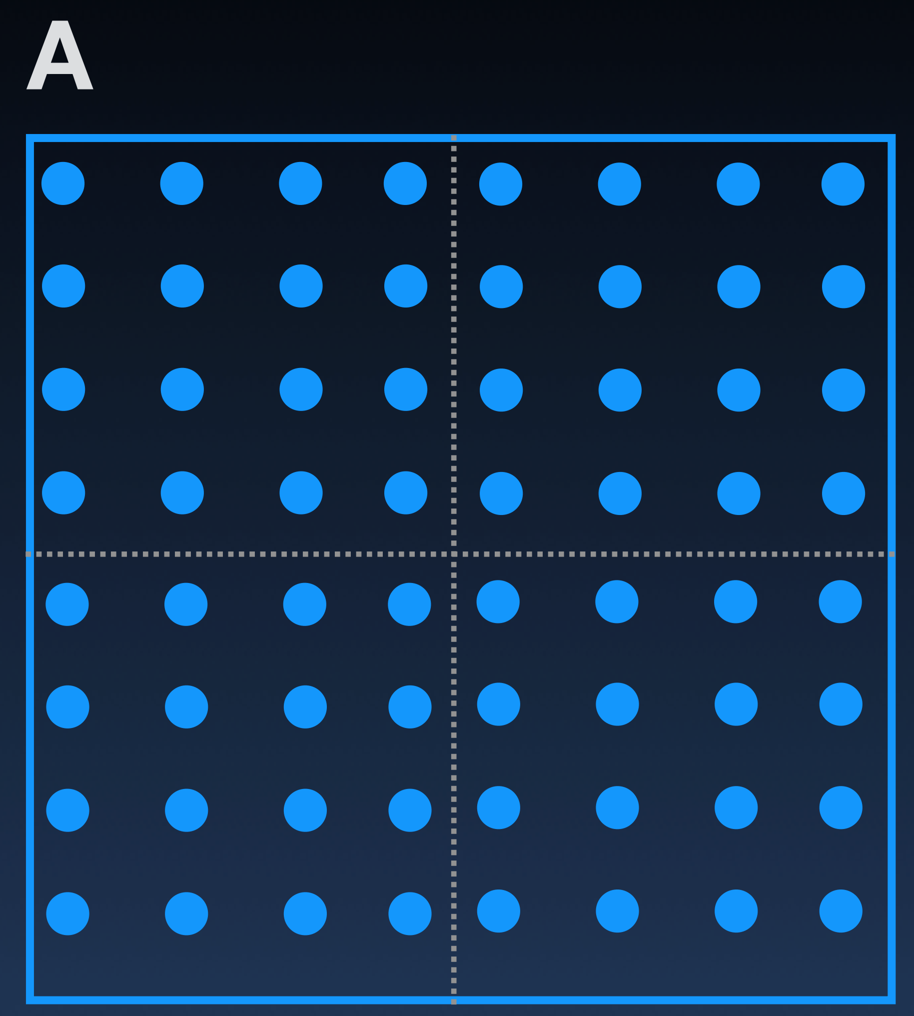
```
MPI_Gather(sendbuf, sendcount, sendtype,
           recvbuf, recvcount, recvtype, root, comm);
```

- 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

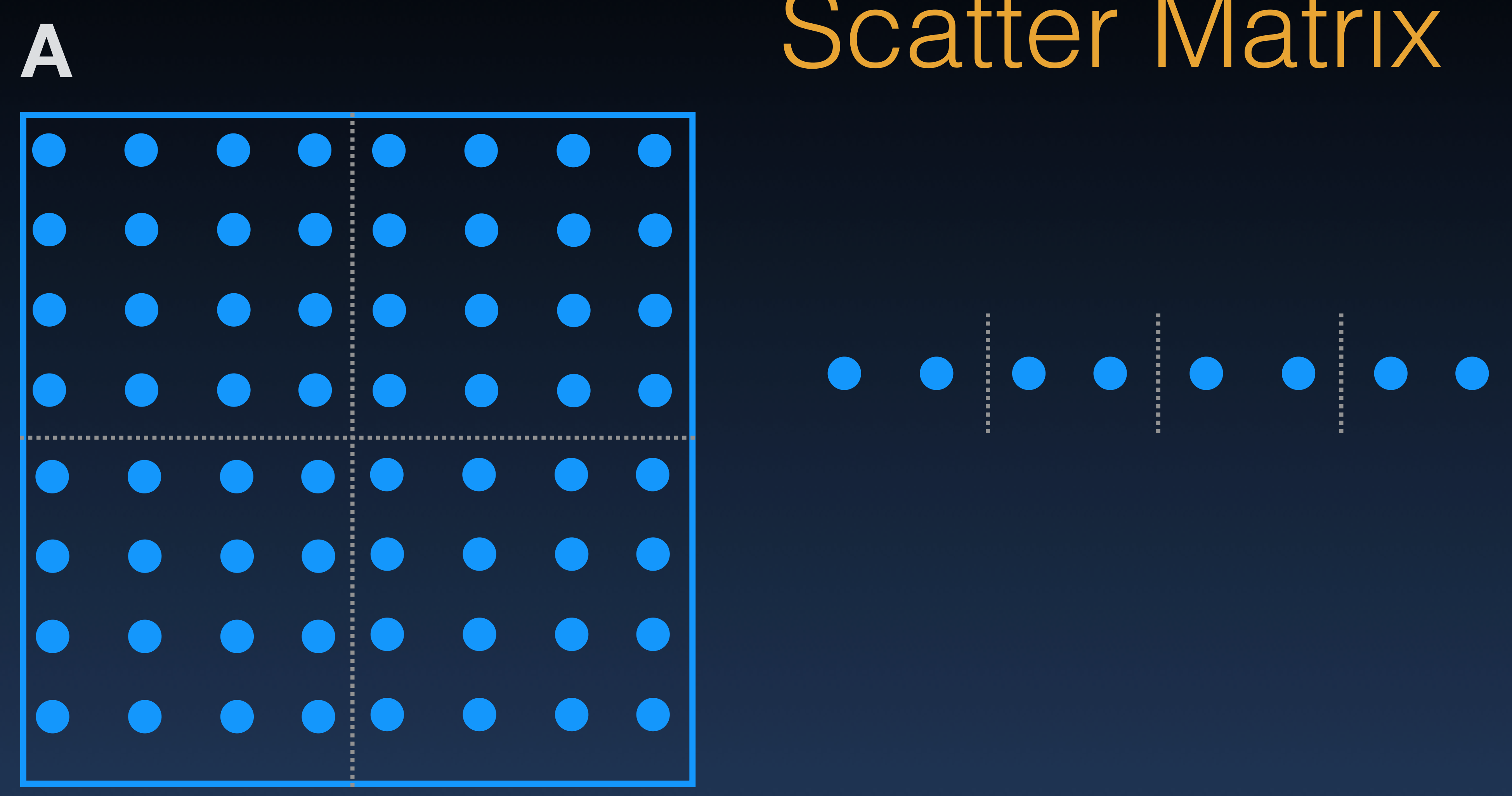
	Data index..			→		Data index ..					
A	A0	A1	A2		A	A0	A1	A2	B0	B1	··
B	B0	B1	B2		B	A0	A1	A2	B0	B1	··
C	C0	C1	C2		C	A0	A1	A2	B0	B1	··
D	D0	D1	D2		D	A0	A1	A2	B0	B1	··
:					:						

(all gather)

Scatter Matrix

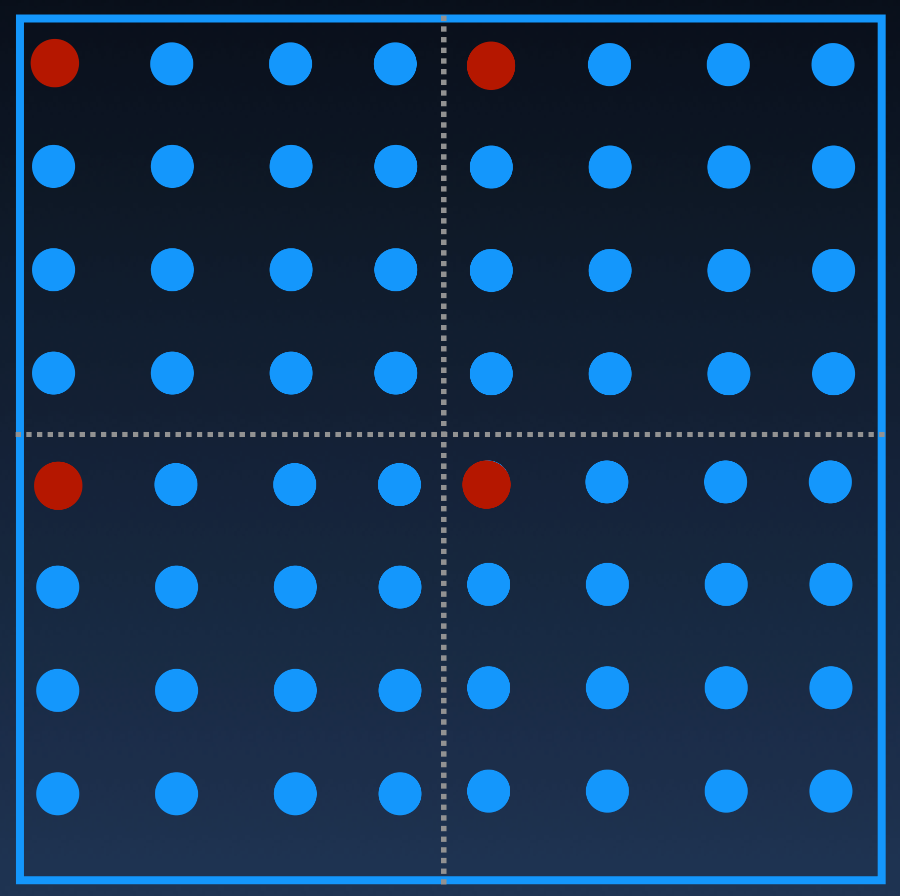


Scatter Matrix



Subodh Kumar

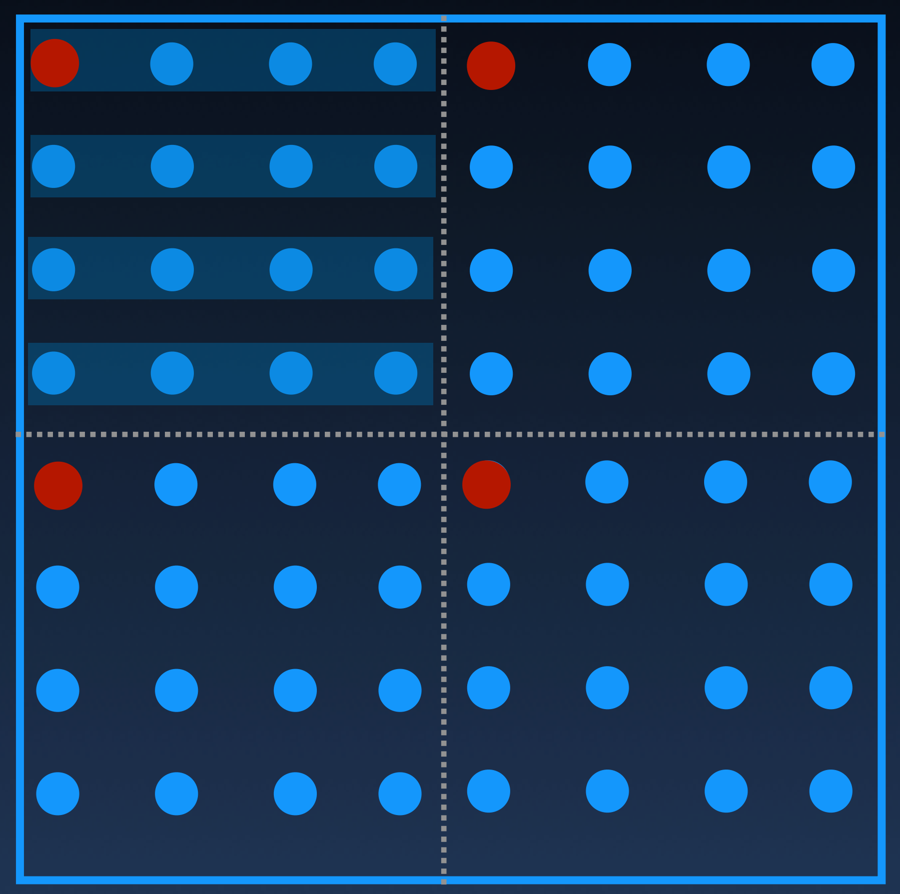
Scatter Matrix




```
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

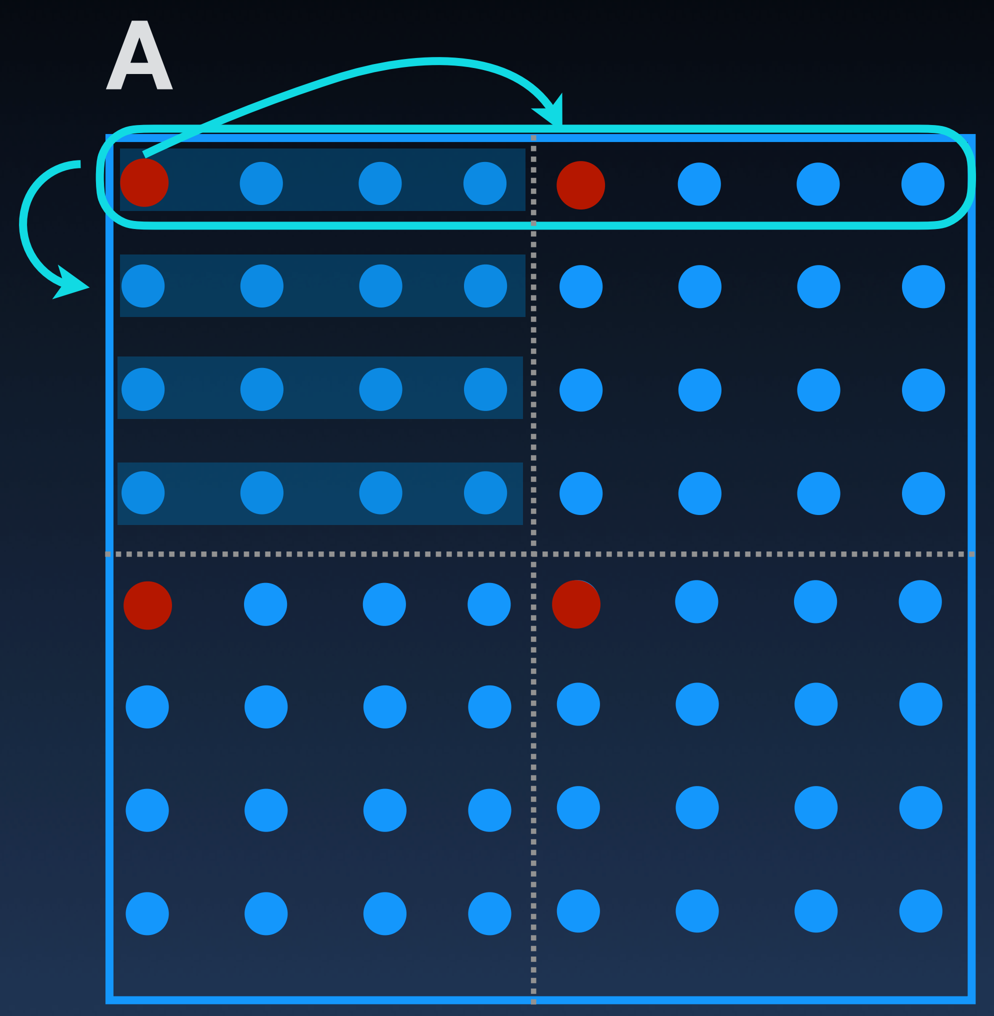
Scatter Matrix




```
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

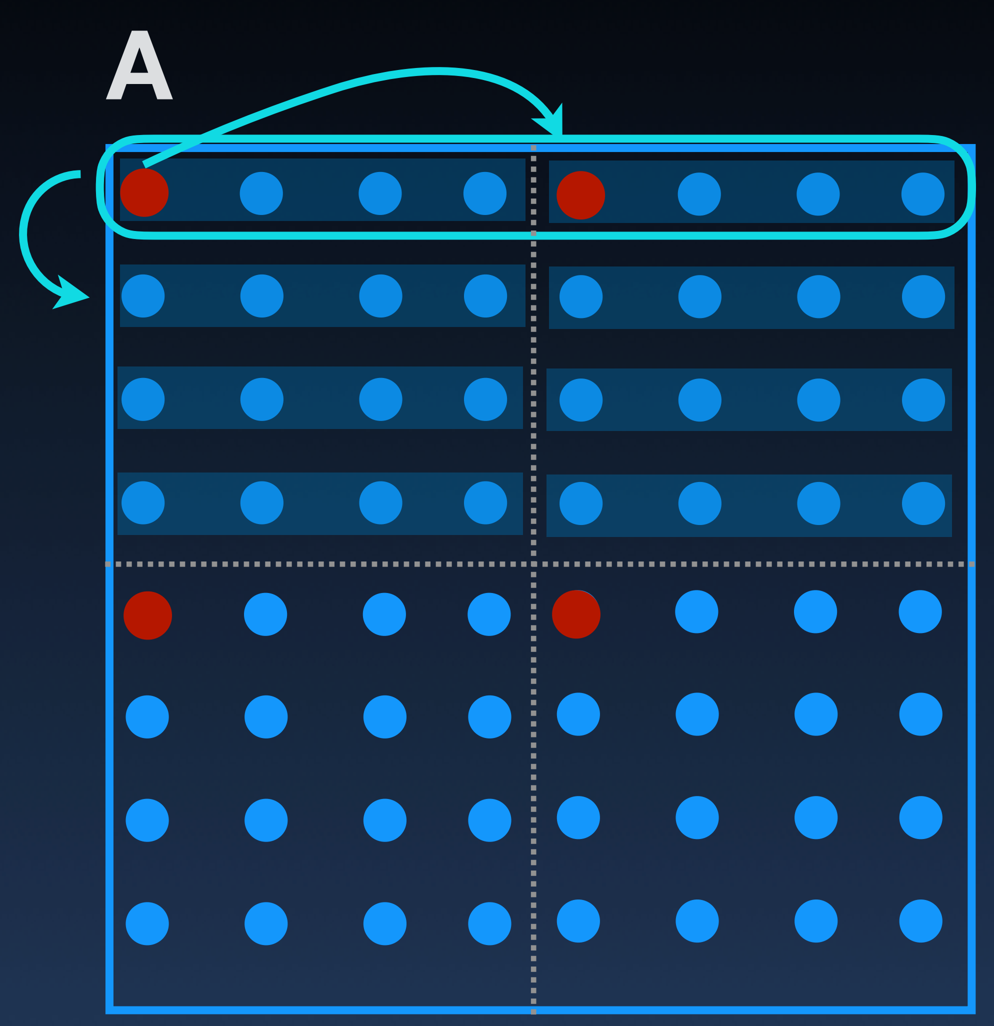



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

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 );
}
```

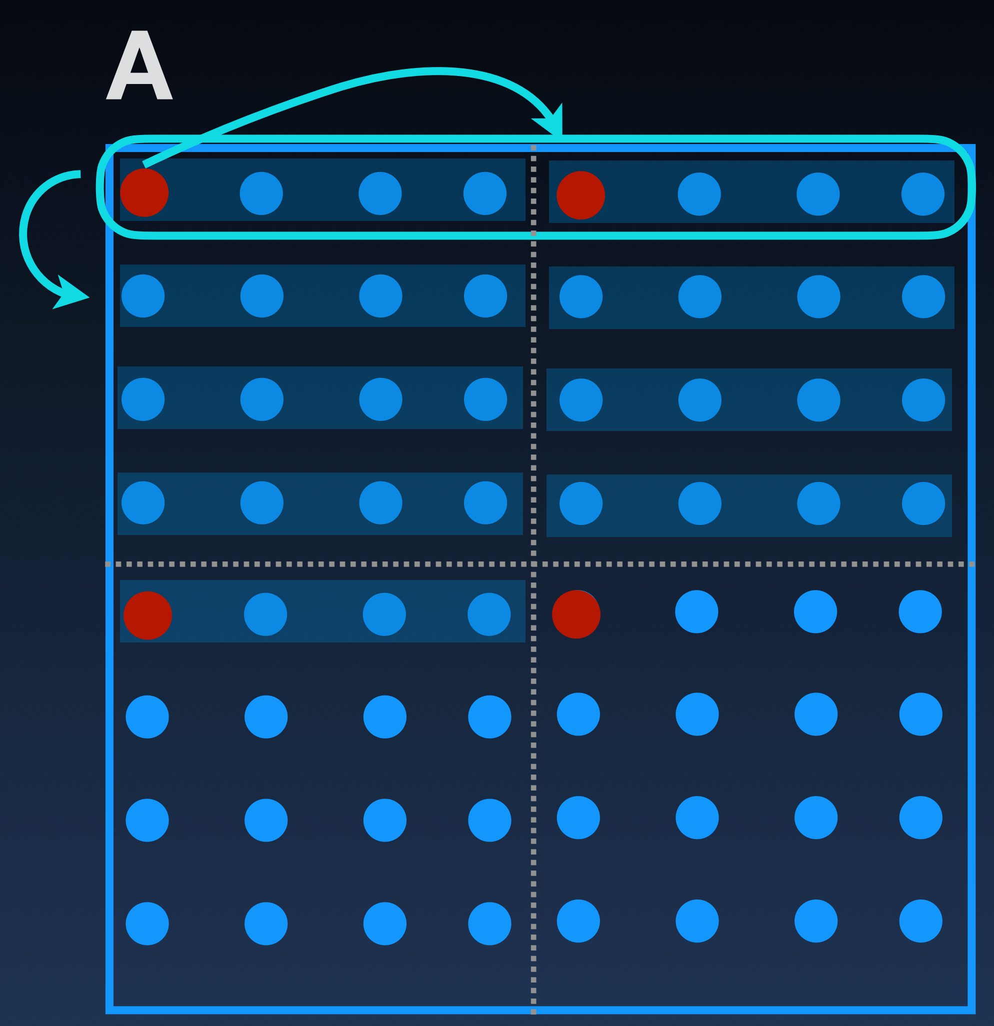


second stype element

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 );
}
```



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

All to All

	Data index..		
A	A0	A1	A2
B	B0	B1	B2
C	C0	C1	C2
:			

→

	Data index ..		
A	A0	B0	C0
B	A1	B1	C1
C	A2	B2	C2
:			

All to All

Data index..

A	A0	A1	A2
B	B0	B1	B2
C	C0	C1	C2
:			

→

Data index ..

A	A0	B0	C0
B	A1	B1	C1
C	A2	B2	C2
:			

Data index..

A	A0	A1	A2	A3	A4	A5
B	B0	B1	B2	B3	B4	B5
C	C0	C1	C2	C3	C4	C5
:						

Data index..

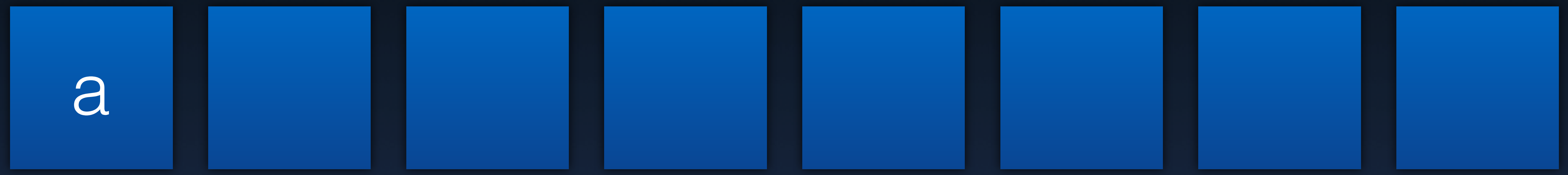
A	A0	A1	B0	B1	C0	C1
B	A2	A3	B2	B3	C2	C3
C	A4	A5	B4	B5	C4	C5
:						

Can all-to-all multiple data items

Subodh Kumar

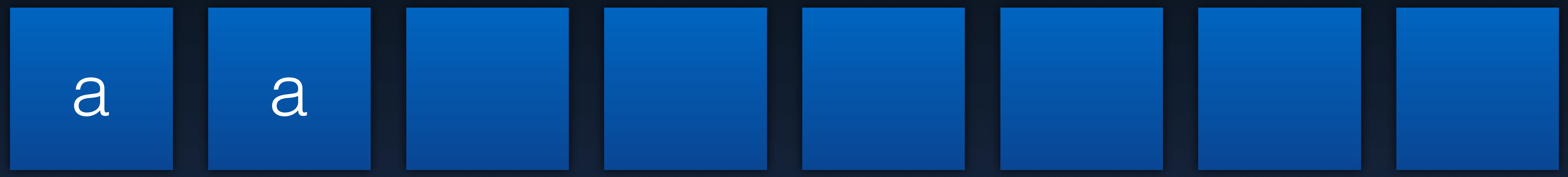
Collective Implementation

Bcast



Collective Implementation

Bcast



Subodh Kumar

Collective Implementation

Bcast	a	a	a	a				

Subodh Kumar

Collective Implementation

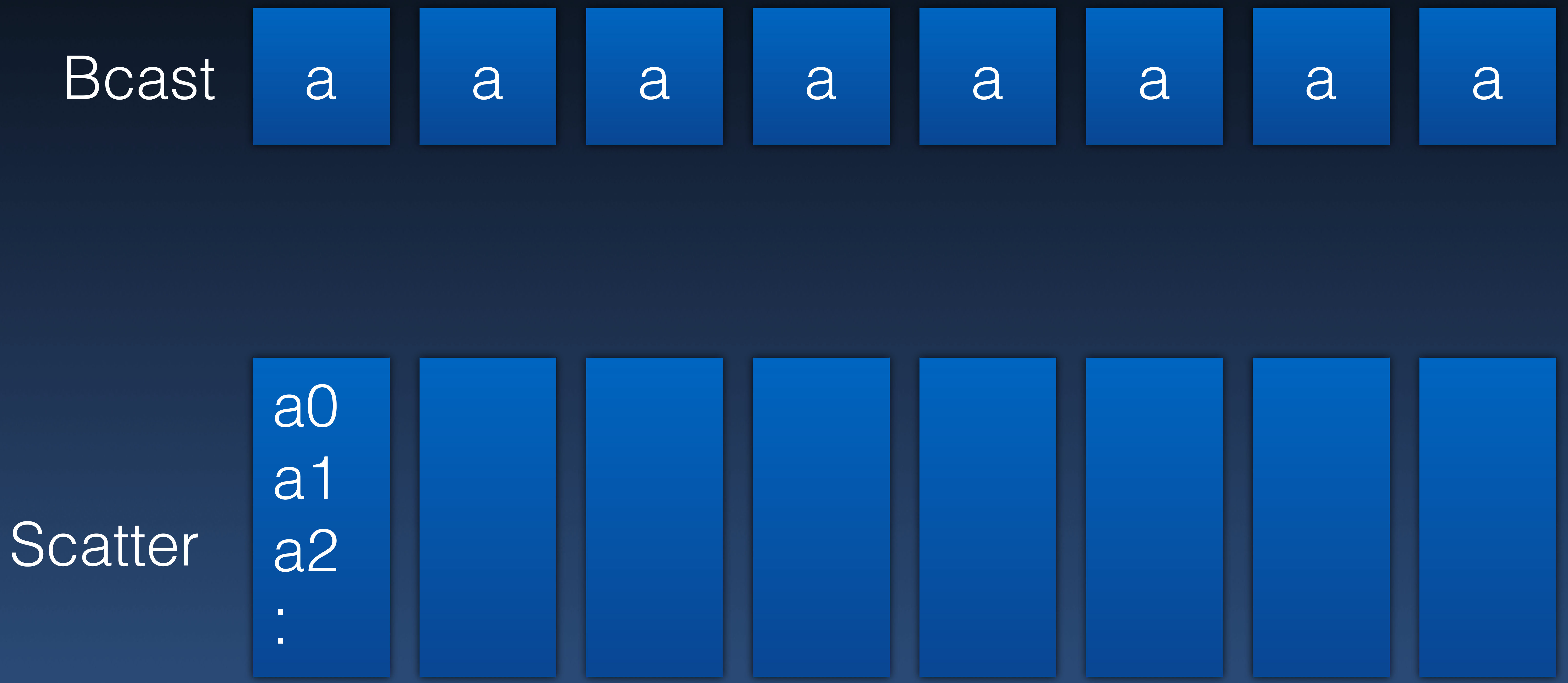
Bcast	a	a	a	a	a	a	a	a

Collective Implementation

Bcast	a	a	a	a	a	a	a	a

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

Collective Implementation

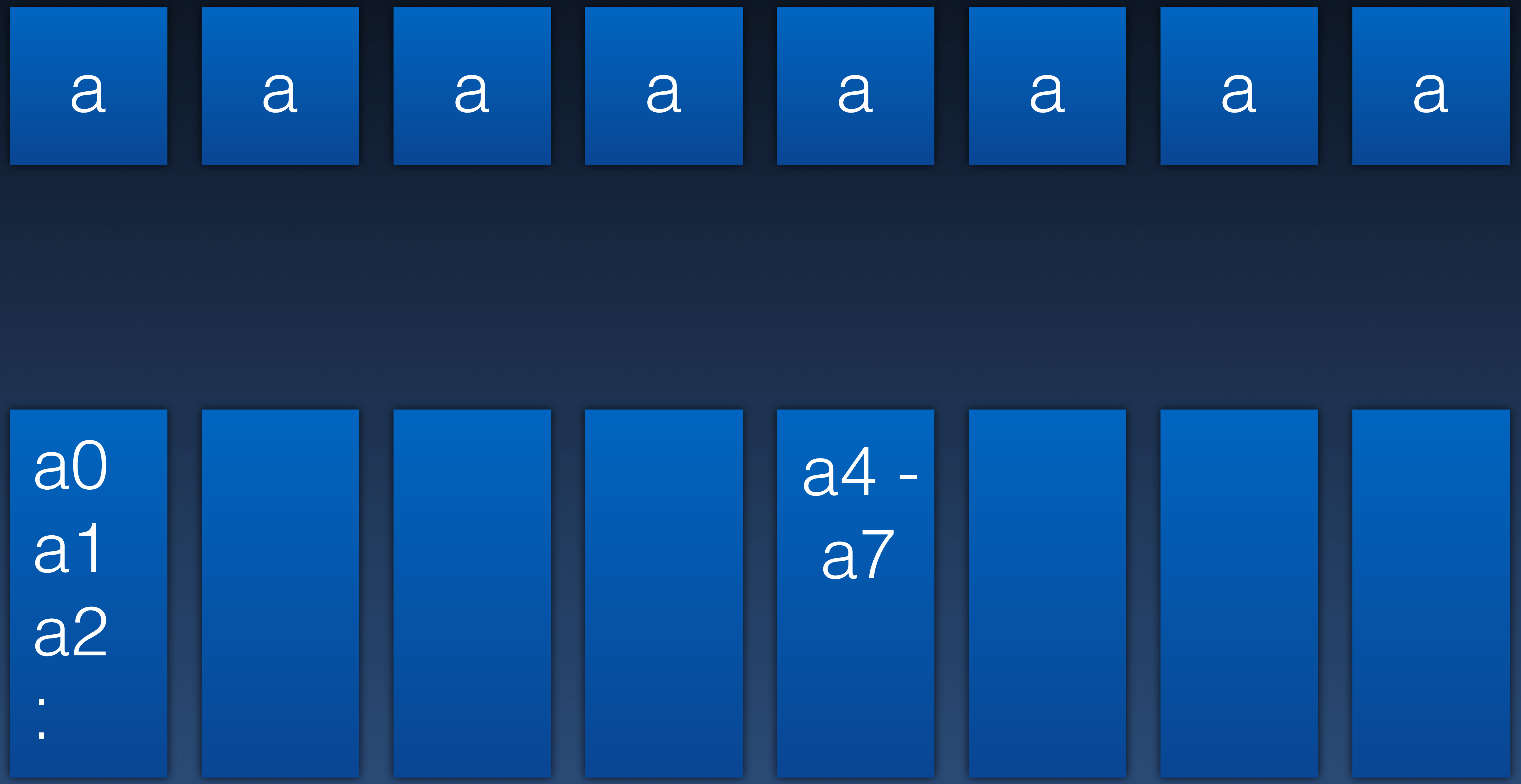


log P rounds
1 message/round/pair
of ‘unit’ size

Collective Implementation

Bcast

Scatter

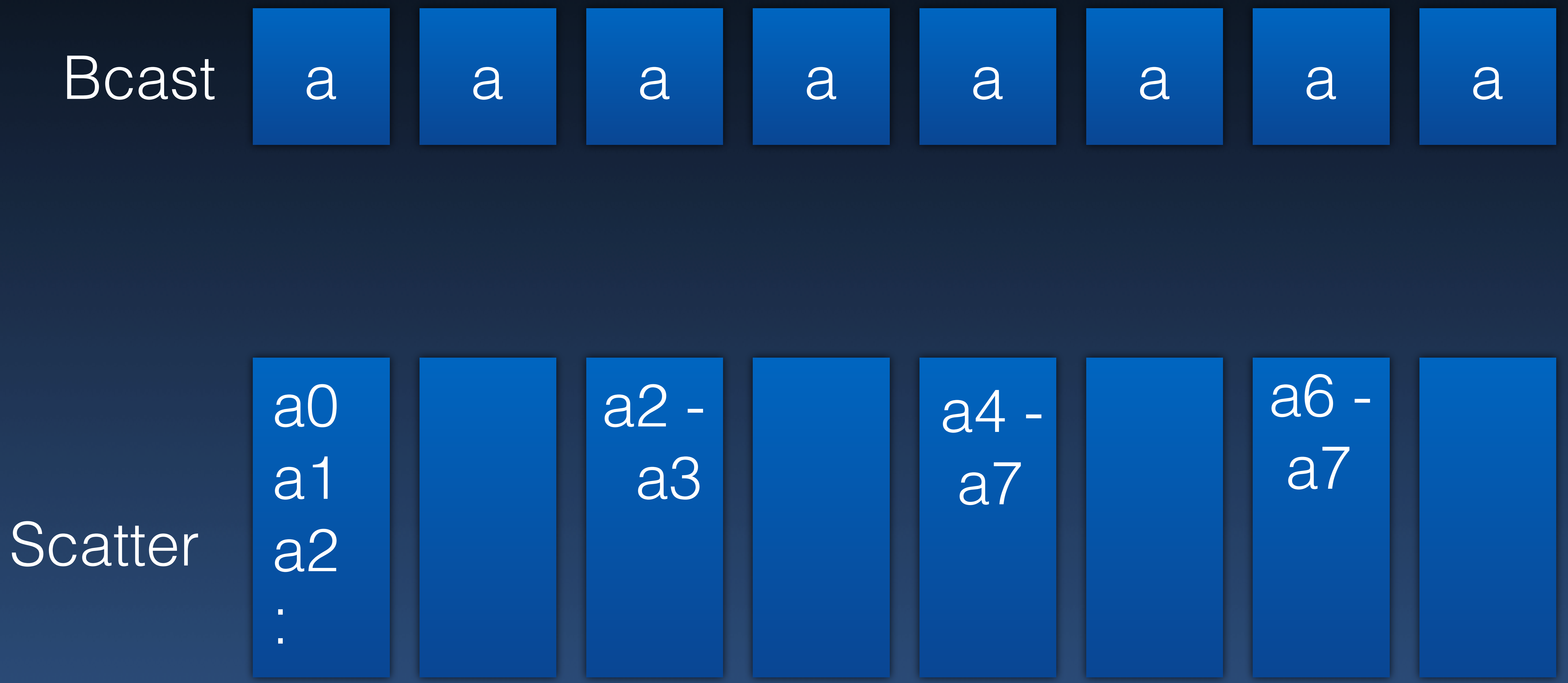


log P rounds

1 message/round/pair

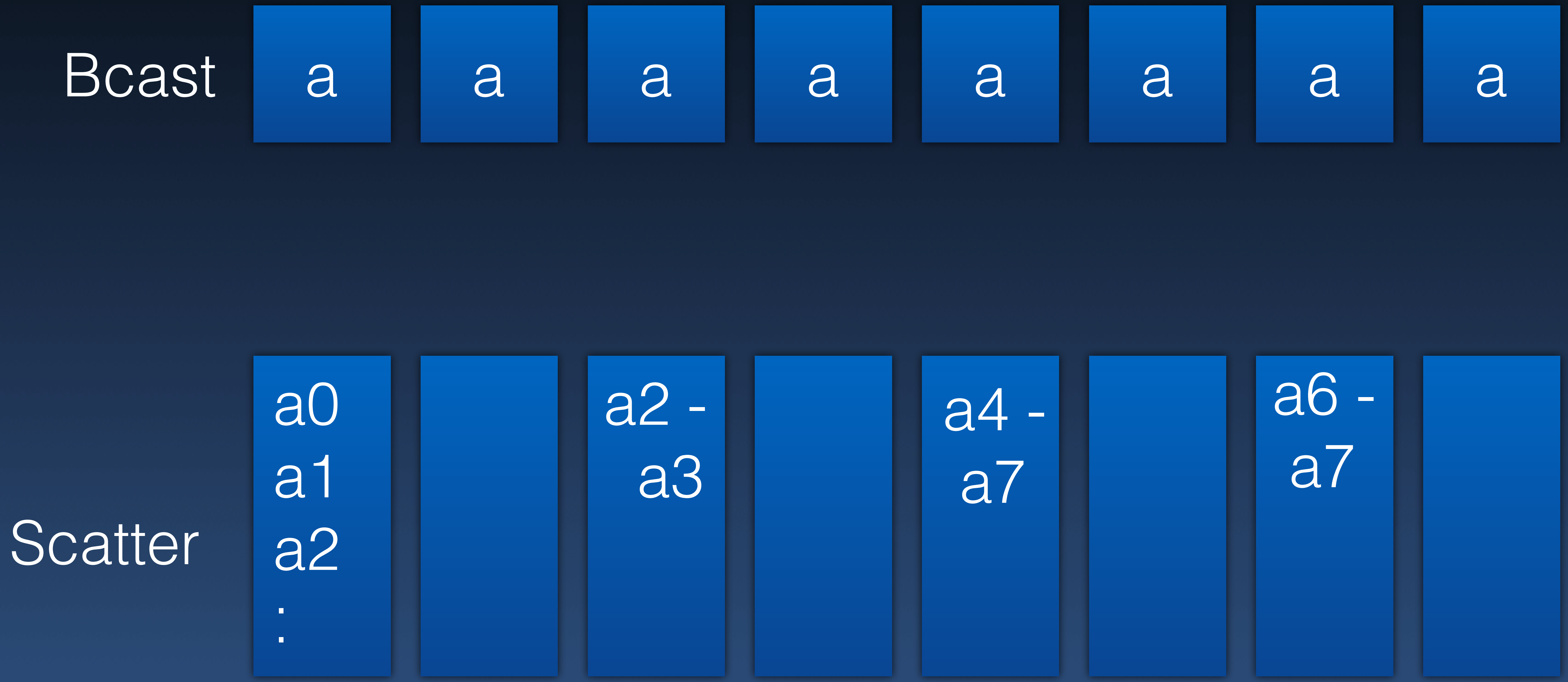
of ‘unit’ size

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

Bcast	a	a	a	a	a	a	a	a

log P rounds
1 message/round/pair
of 'unit' size

Scatter	a0 a1 a2 :		a2 - a3		a4 - a7		a6 - a7	

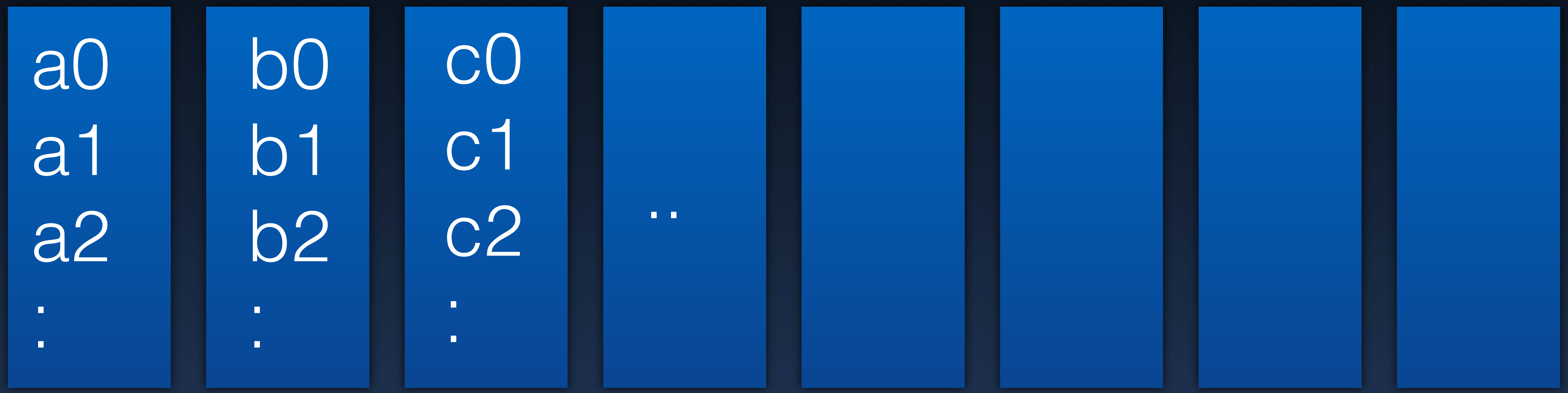
log P rounds
1 message/round/pair
of P/2, P/4, P/8 .. units

$r = 2^{\lceil \log n \rceil}$

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

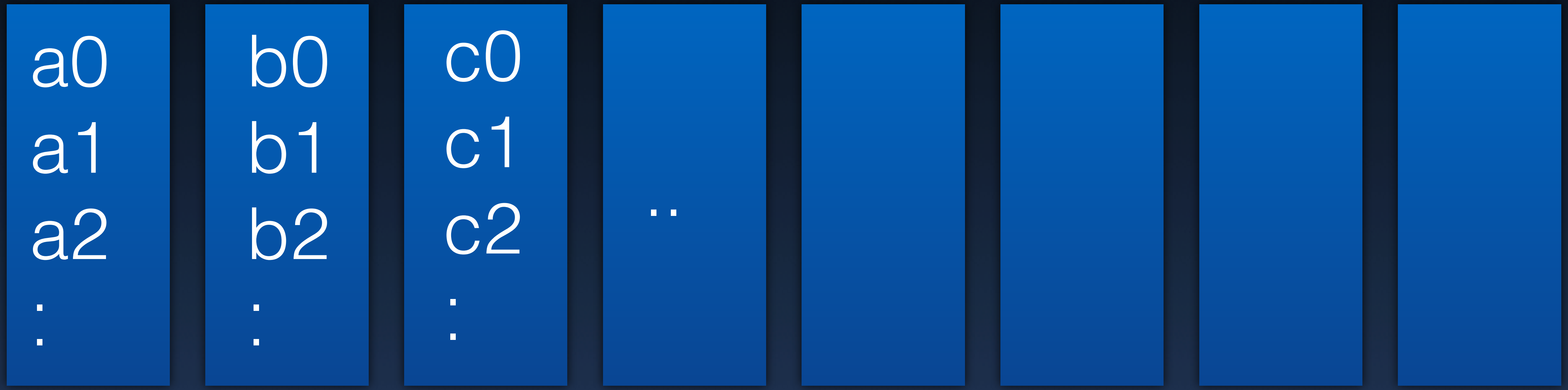
Collective Communication

All to All



Collective Communication

All to All



P sequential Scatters?

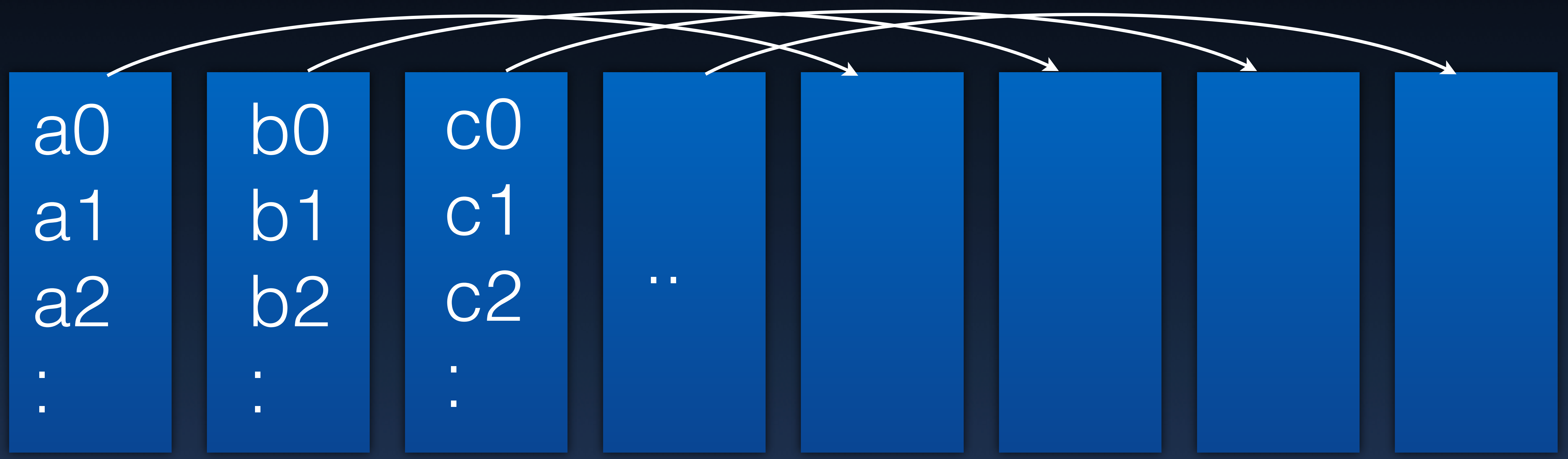
Collective Communication

All to All

P sequential Scatters?

Collective Communication

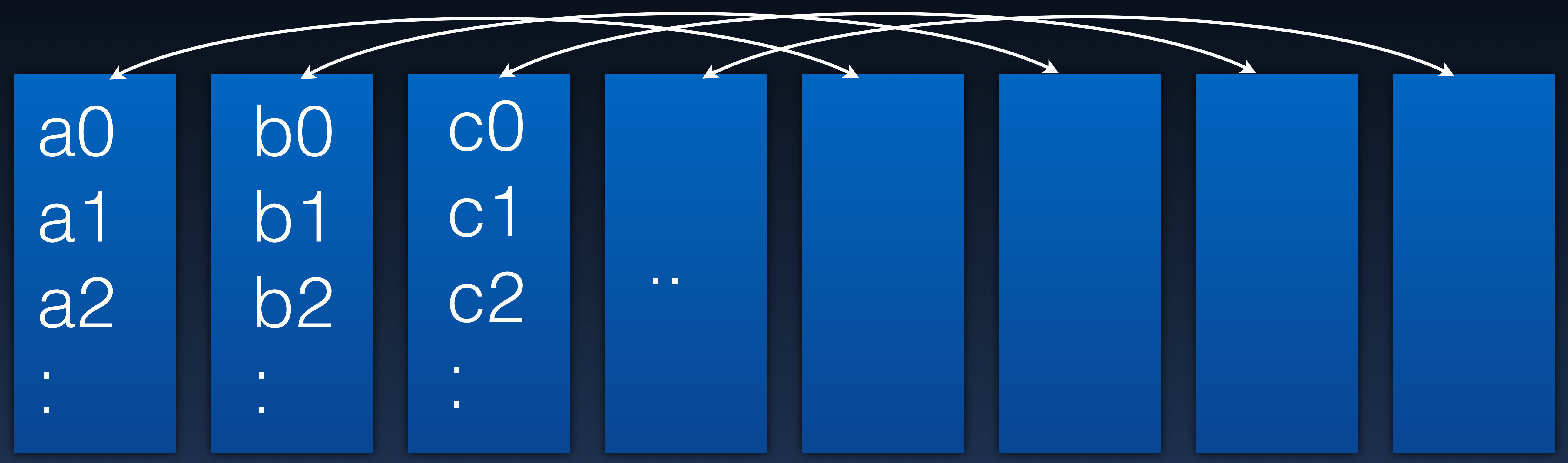
All to All



P sequential Scatters?

Collective Communication

All to All

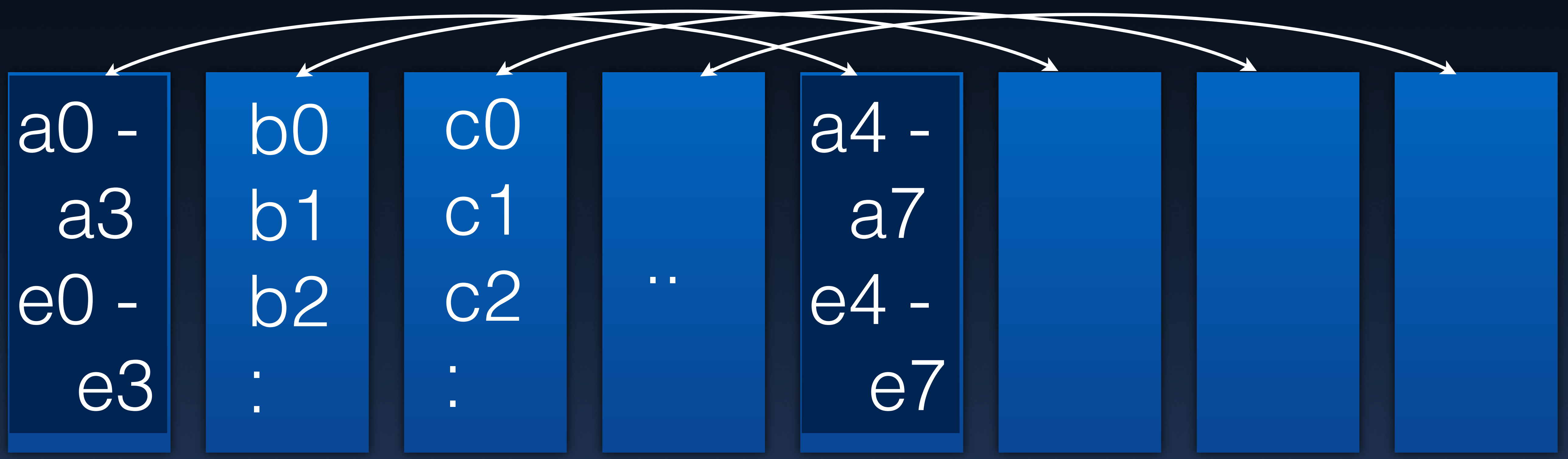


P sequential Scatters?

Subodh Kumar

Collective Communication

All to All



P sequential Scatters?

Collective Communication

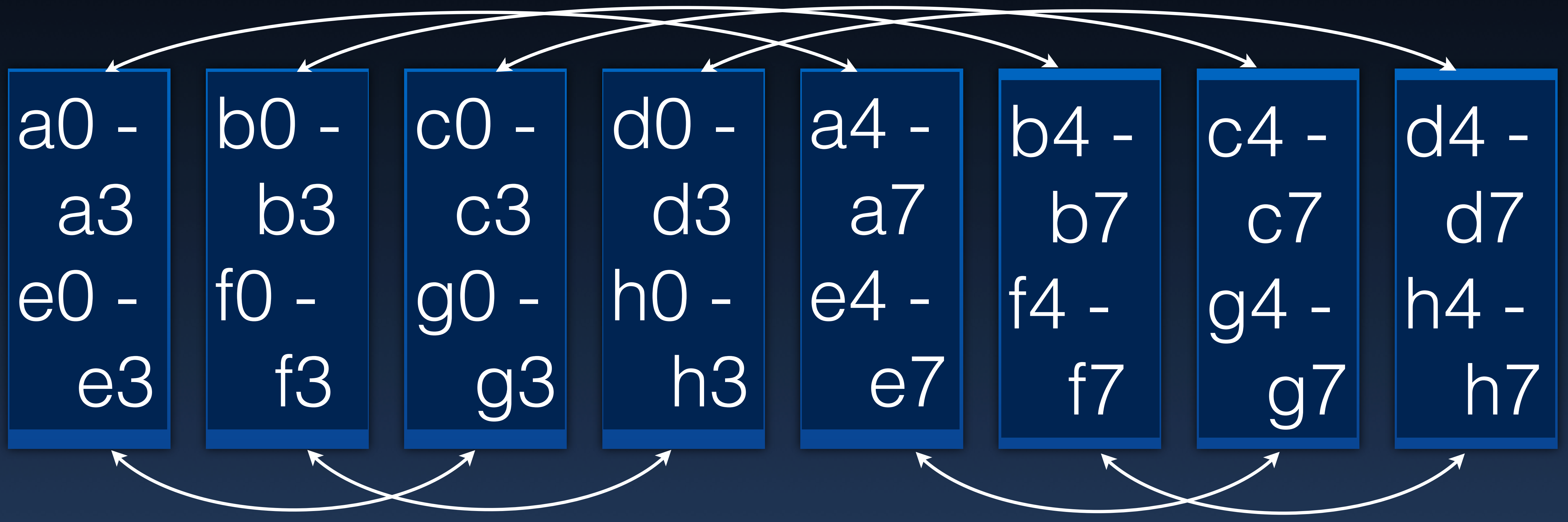
All to All

P sequential Scatters?

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

Collective Communication

All to All

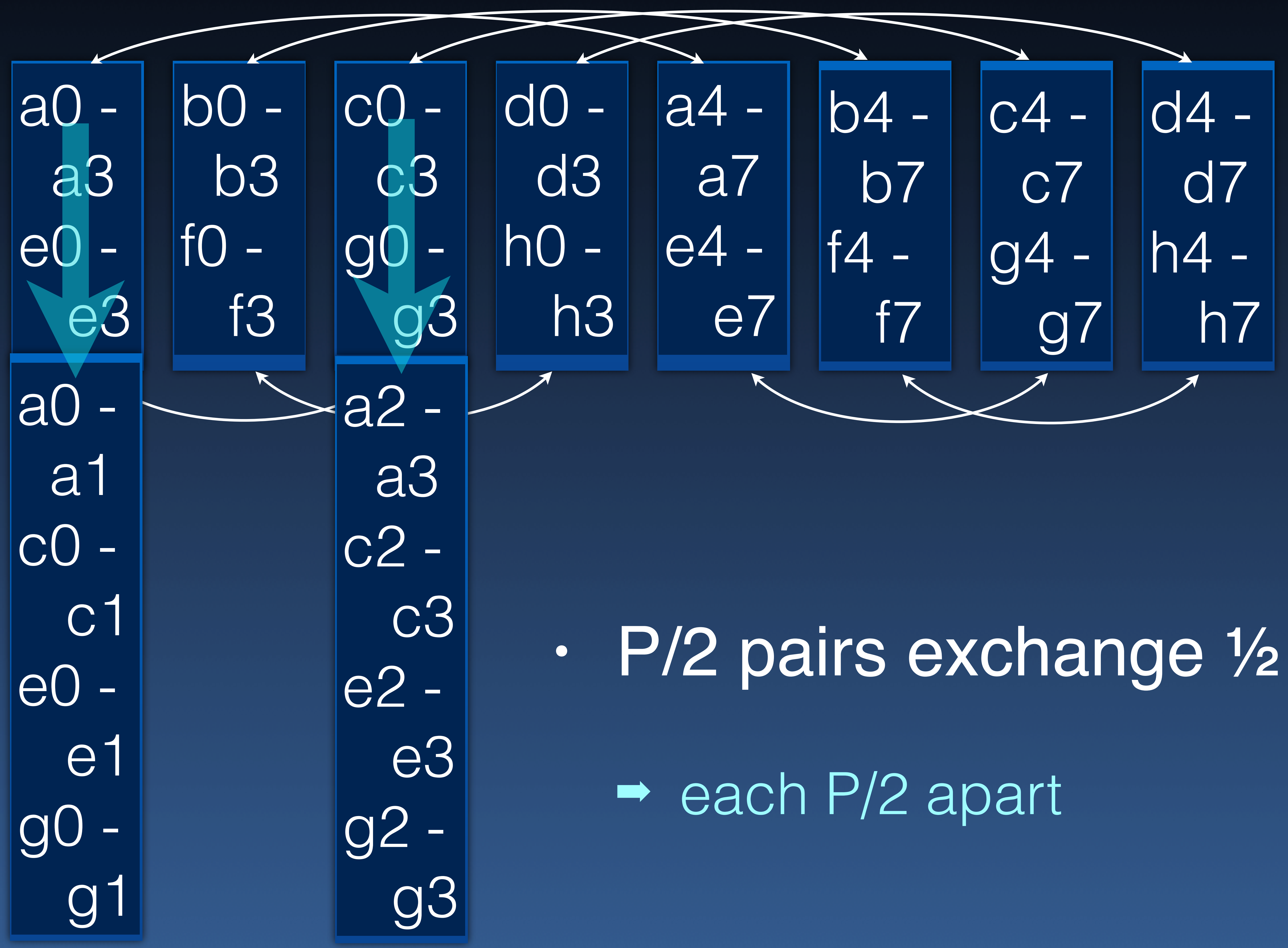


P sequential Scatters?

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

Collective Communication

All to All

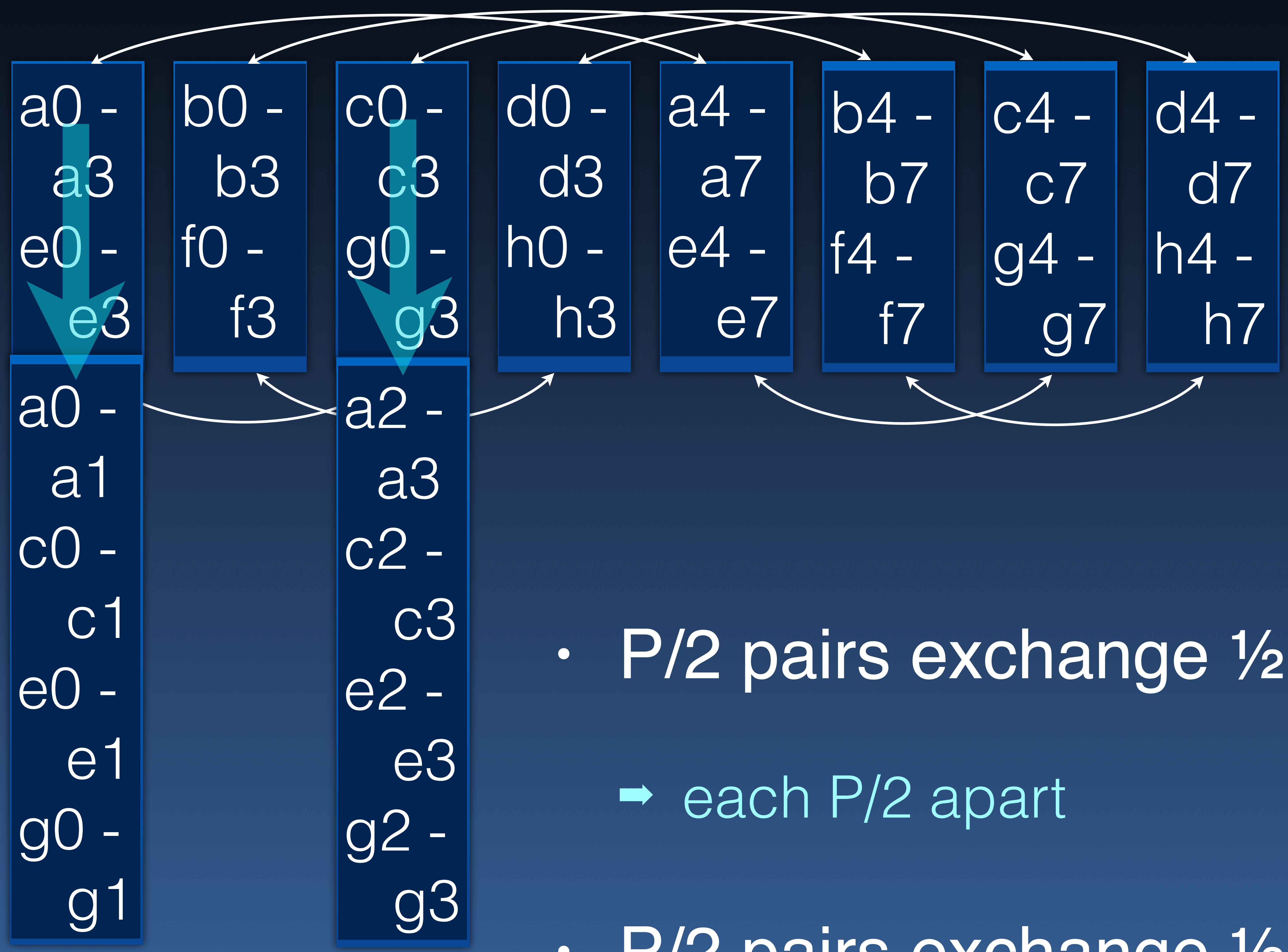


P sequential Scatters?

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

Collective Communication

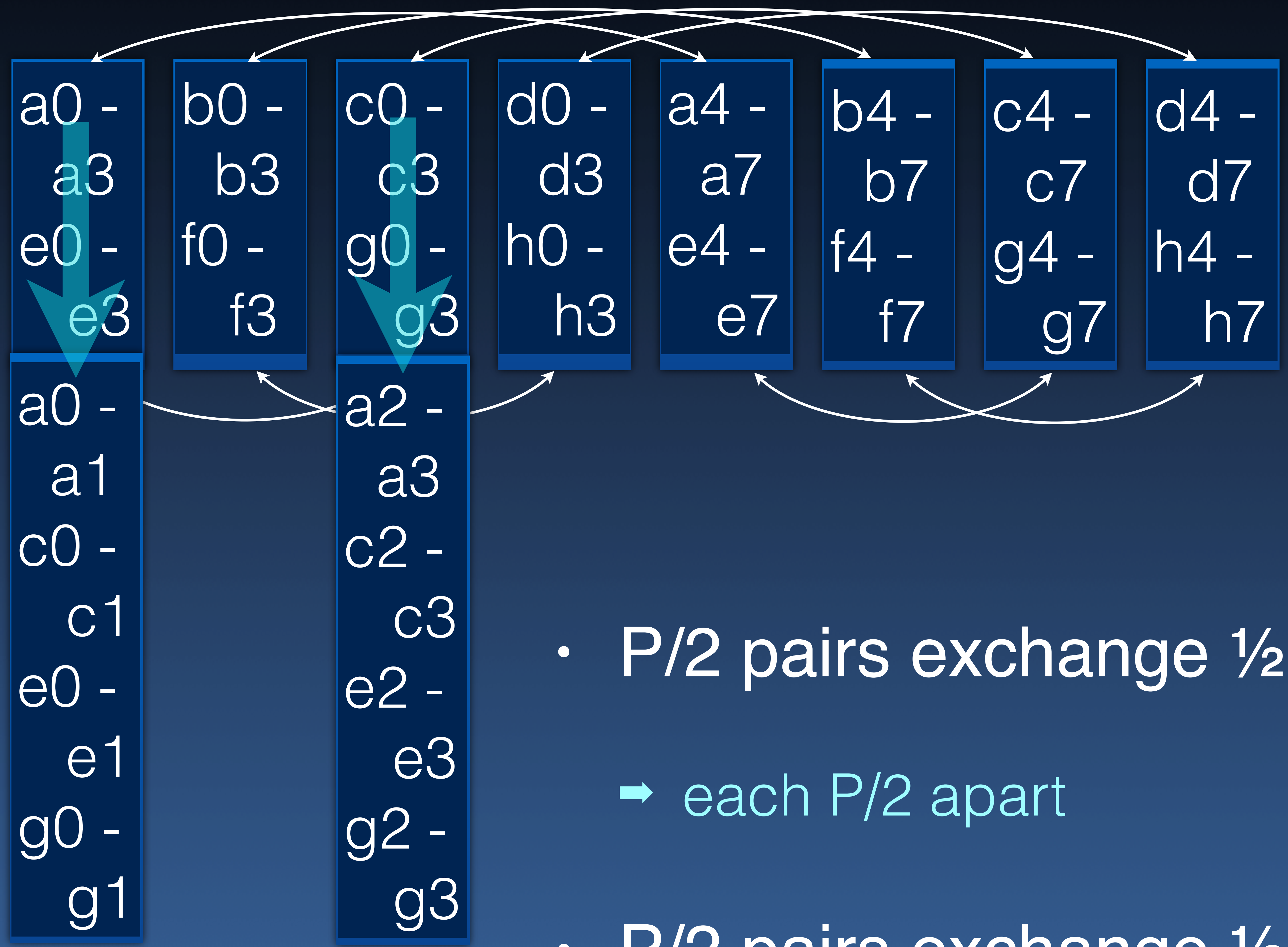
All to All



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

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

Remote Memory

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

MPI_Info   MPI_Win

…

MPI_Win_free(&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)

Subodh Kumar

- 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

Look these up

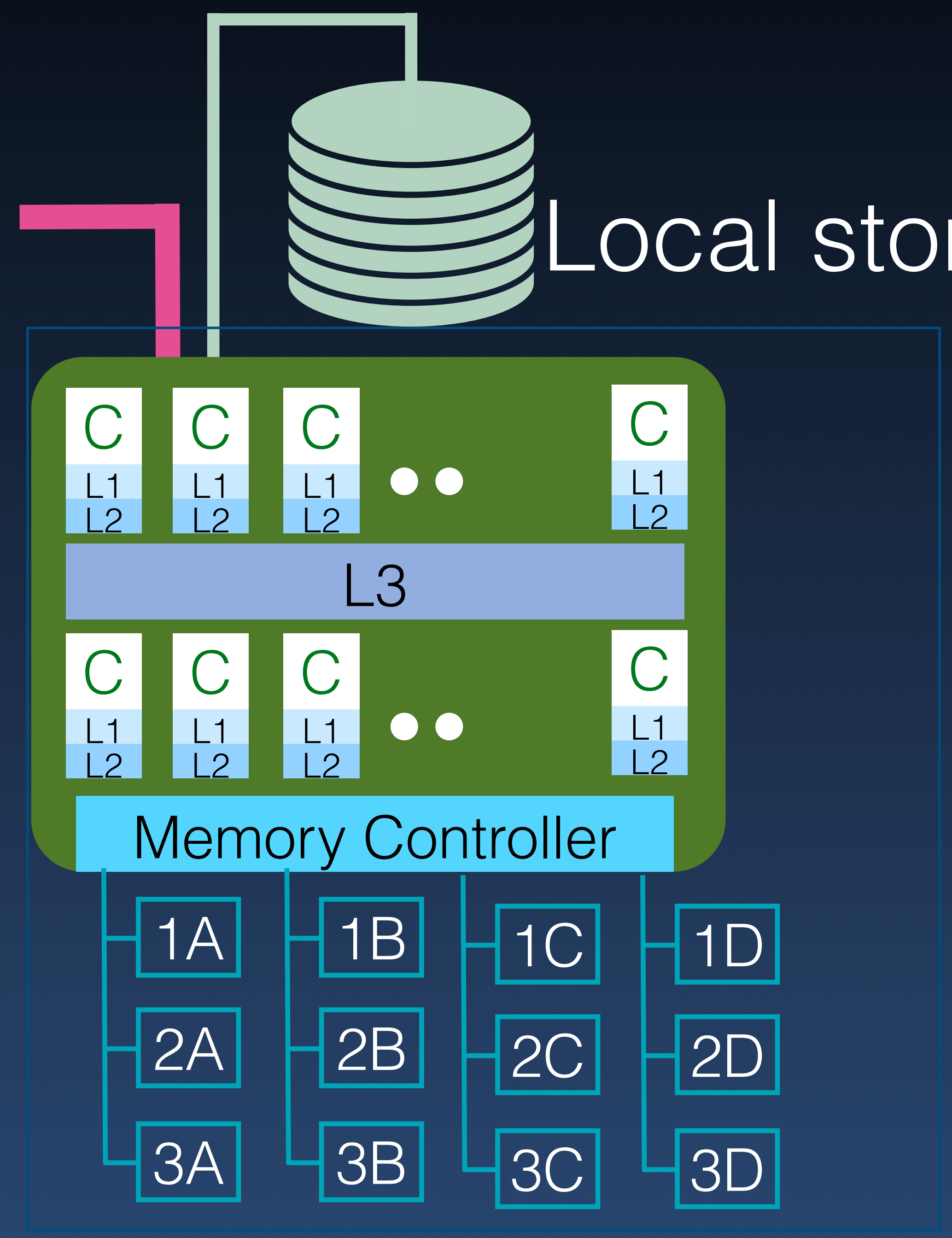
```
int winbuf[1024];
MPI_Win windo;
MPI_Win_create(winbuf, 1024*sizeof(int), sizeof(int),
               MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

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

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access

Parallel File Systems

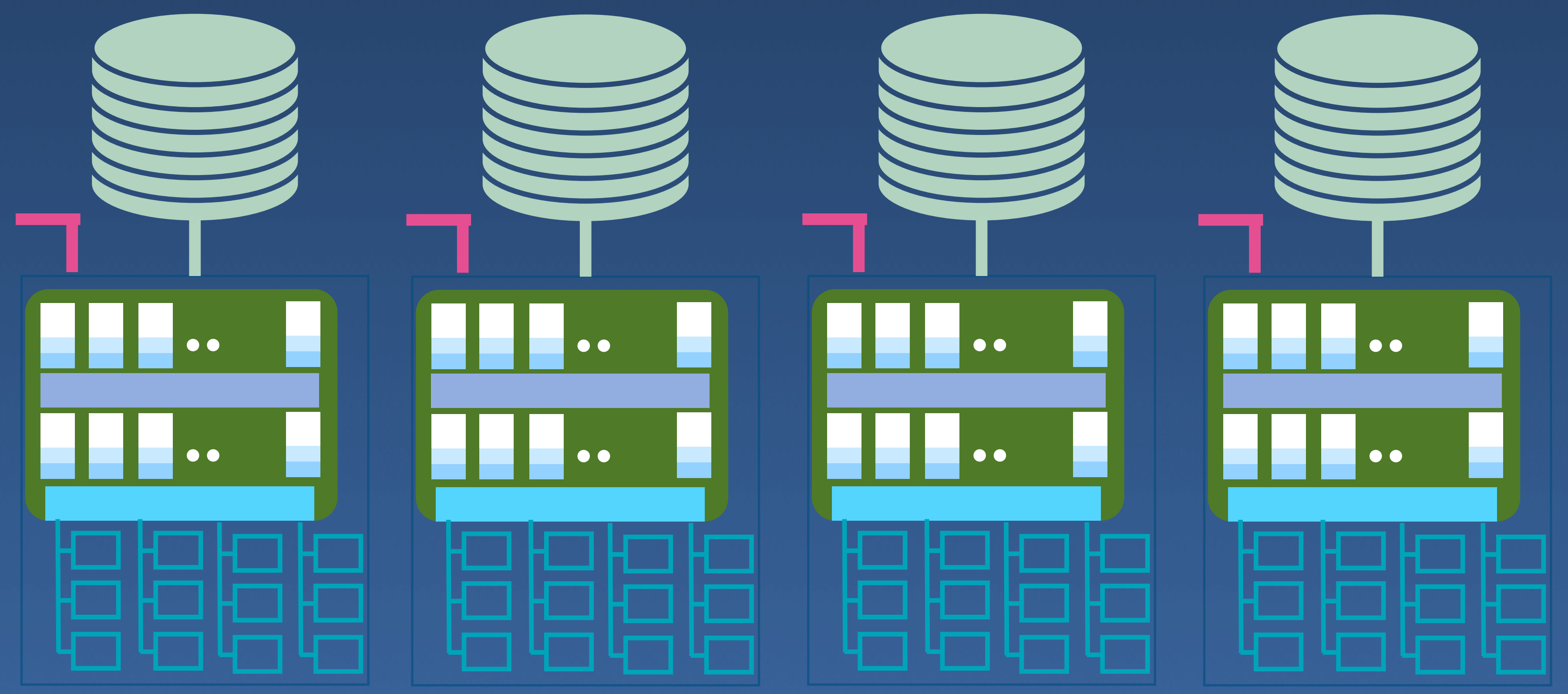
- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Subodh Kumar

Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access

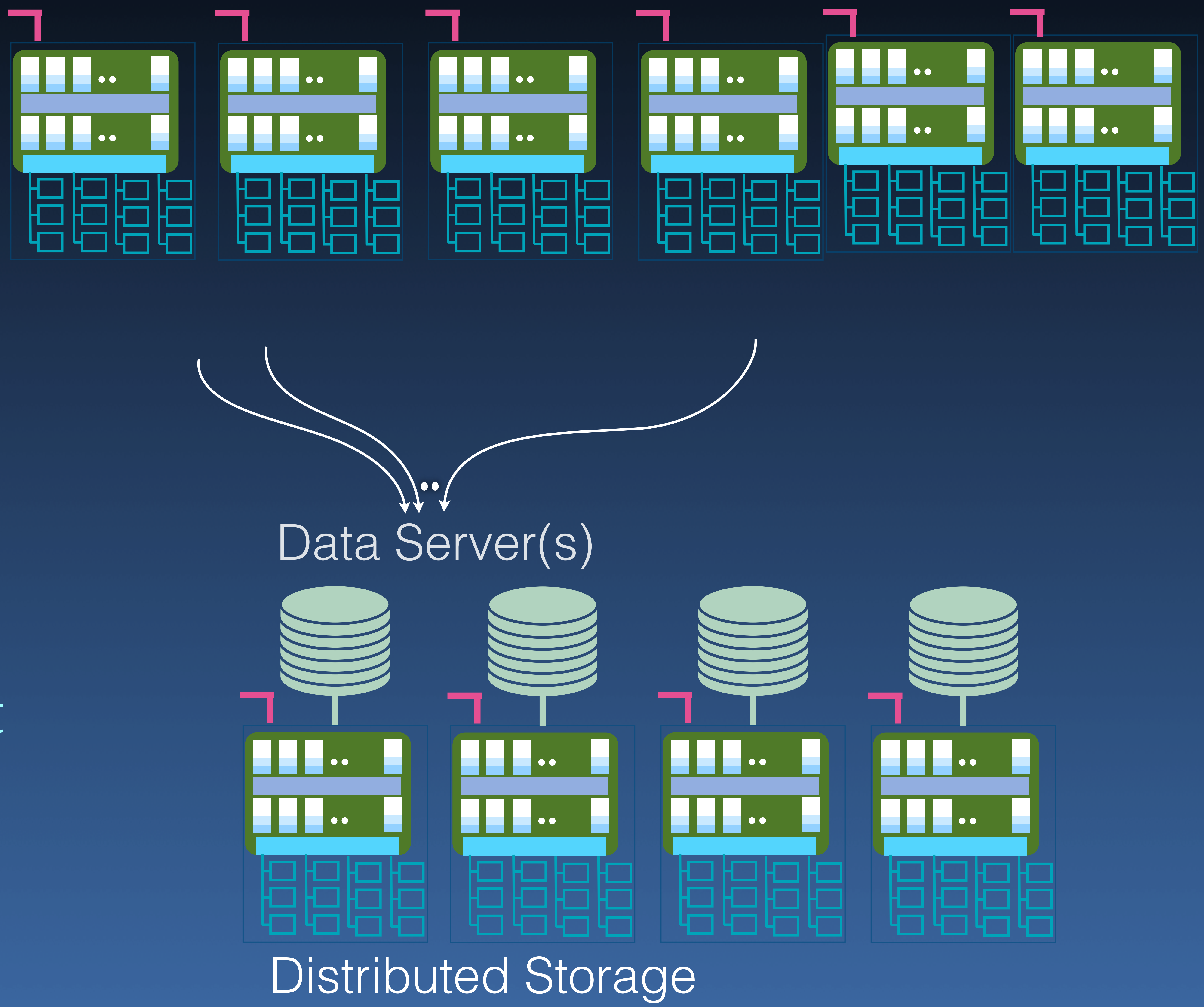


Distributed Storage

Subodh Kumar

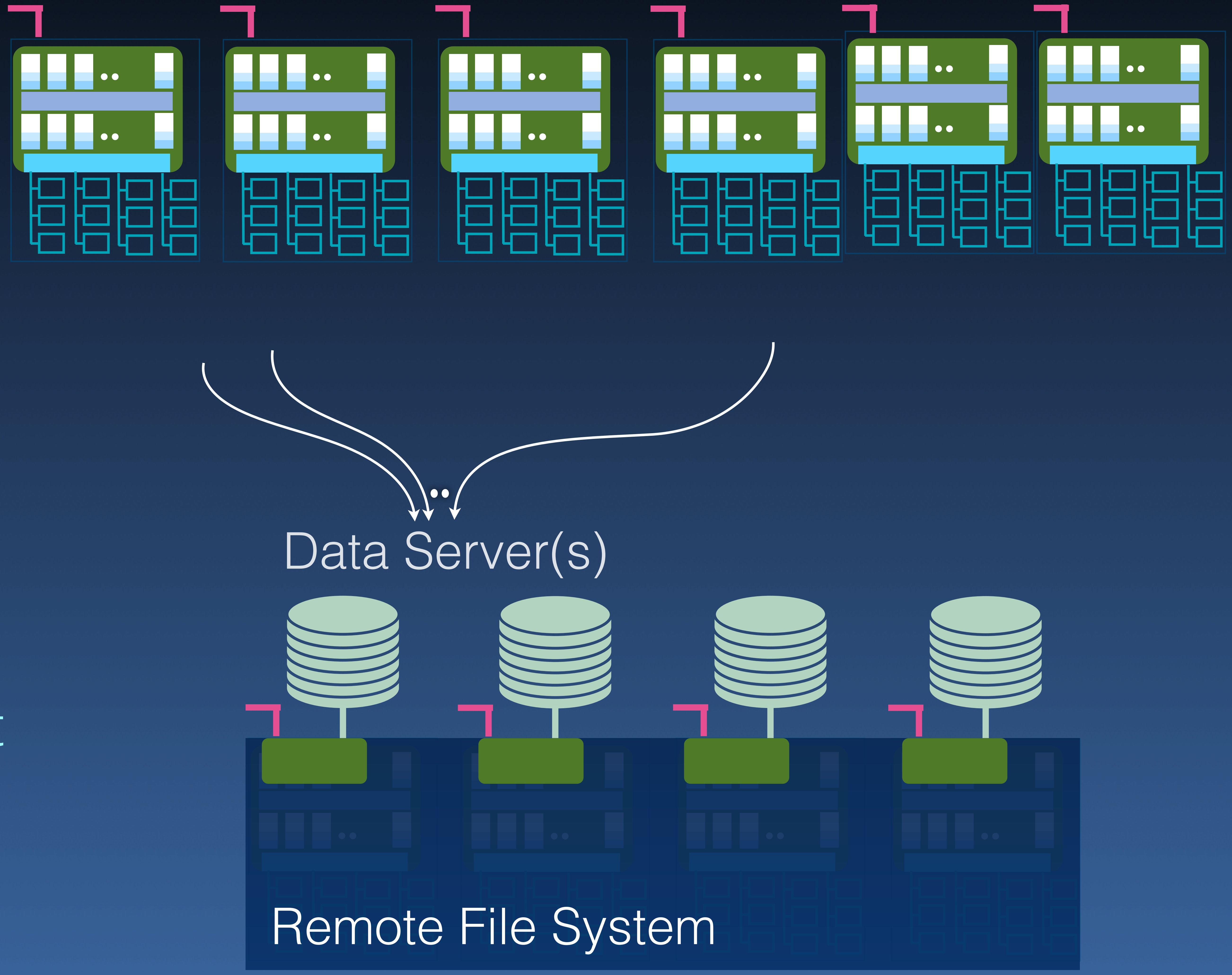
Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



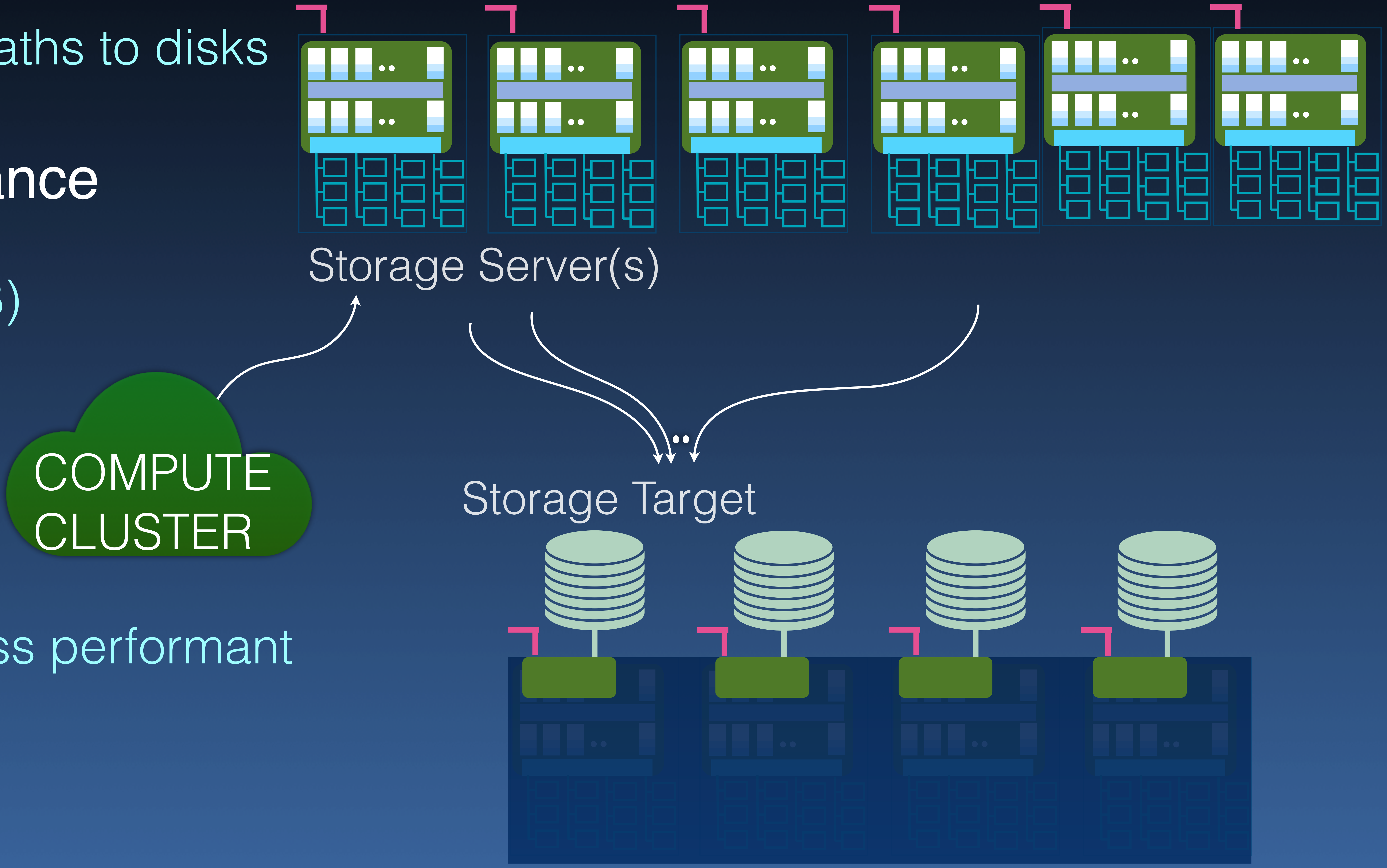
Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



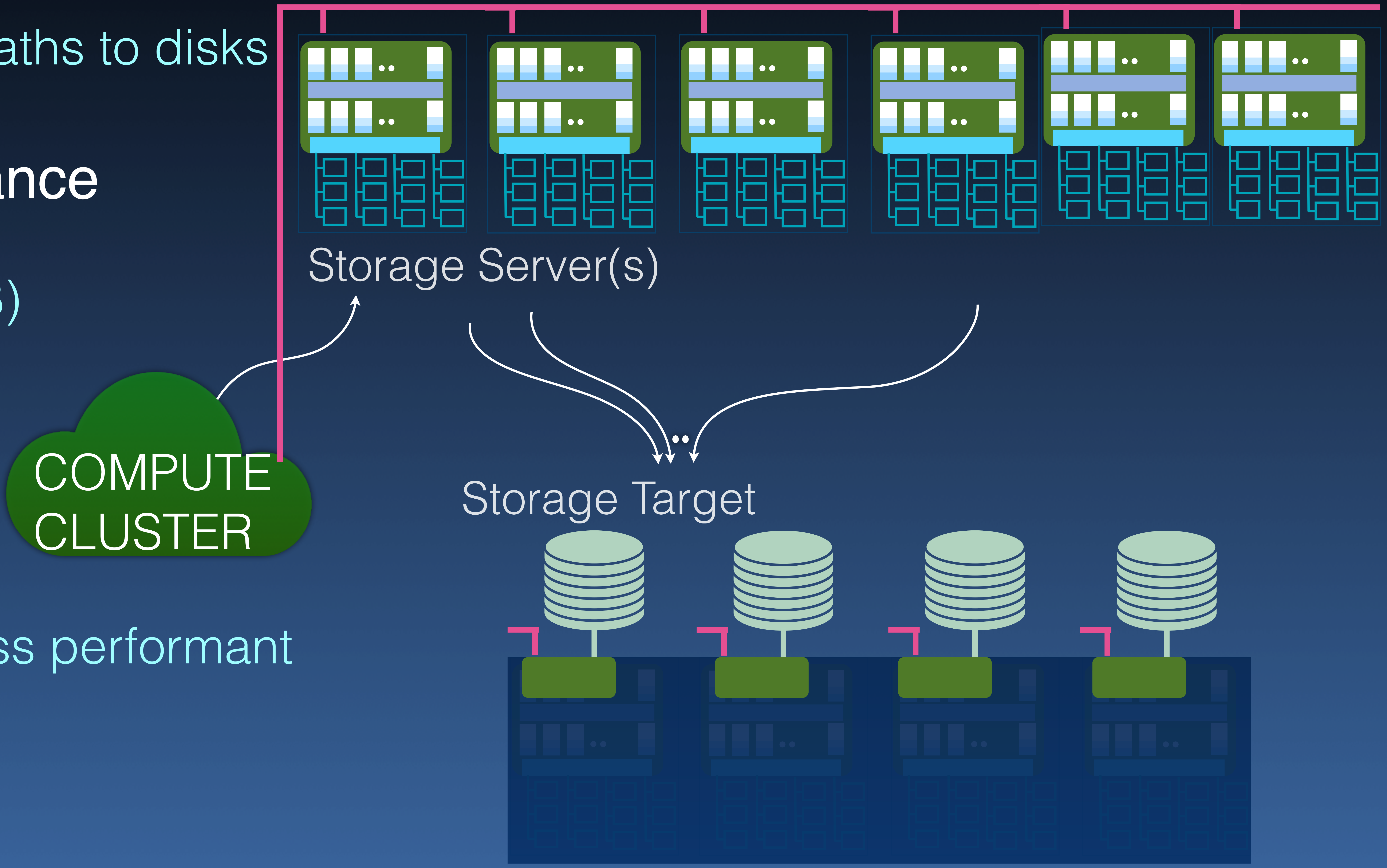
Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access

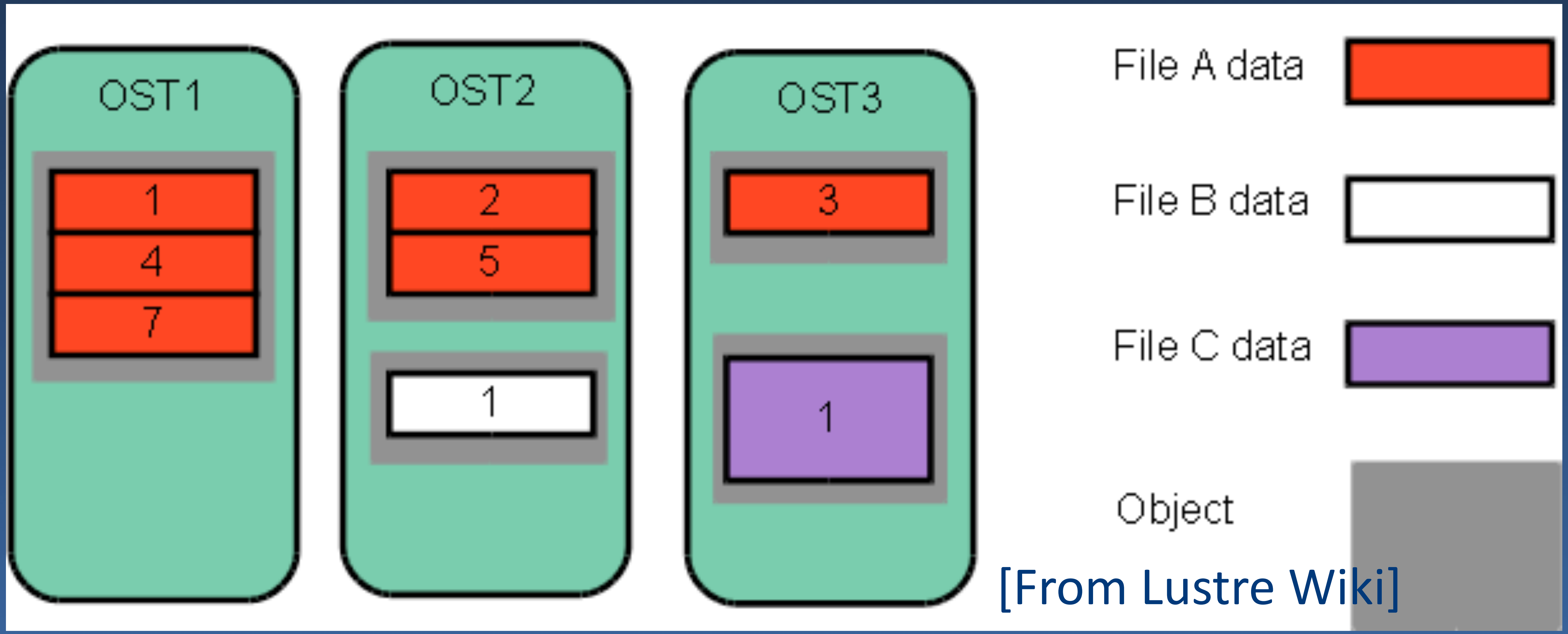


PFS Striping

- Configuration per file
  - ➡ number of stripes, stripe size, and OSTs to use

Stripe counts
File A: 3
File B: 1
File C: 1

Stripe size of File C is larger



[From Lustre Wiki]

```
> lfs getstripe <filename>
> lfs setstripe <dirname>
```

Example: Collective IO

```
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_File_open(MPI_COMM_WORLD, "file", MPI_MODE_RDWR|MPI_MODE_CREATE,
                    MPI_INFO_NULL, &fh ); // Collective, Blocking

MPI_File_write_ordered( fh, buf, 1, MPI_INT, &status );// Collective write in order of ranks
MPI_Barrier(MPI_COMM_WORLD);                            // Let all writes complete

MPI_File_seek( fh, 0, MPI_SEEK_SET );                   // Each separately ‘rewinds’ to the top
MPI_File_read_all( fh, buf, size, MPI_INT, &status );   // All read size ints from their fh

MPI_File_seek_shared(fh, 0, MPI_SEEK_SET );             // Collective rewind of shared fh
MPI_File_read_ordered(fh, buf, 1, MPI_INT, &status ); // Collective read in order of ranks

MPI_File_close( &fh );
```

IO Variants

- Location

  MPI_File_read_**at**(fh, offset, buffer, count, datatype, &status)

- Non-blocking

  MPI_File_**i**read(fh, buffer, count, datatype, &request)

- Collective

  MPI_File_read_**all**(fh, buffer, count, datatype, &status)

- Shared File pointer (Common data IO)

  MPI_File_read_**shared**(fh, buffer, count, datatype, &status) // Not collective

  MPI_File_read_**ordered** (fh, buffer, count, datatype, &status) // Collective

See:
  MPI_File_set_atomicity
  MPI_File_sync

File View

- 3-tuple: <displacement, etype, filetype>
  - ➡ byte displacement from the start of the file
  - ➡ etype: data unit type
  - ➡ filetype: portion of the file visible to the process
- MPI_File_set_view

File View

Map data structures with file data

Subodh Kumar

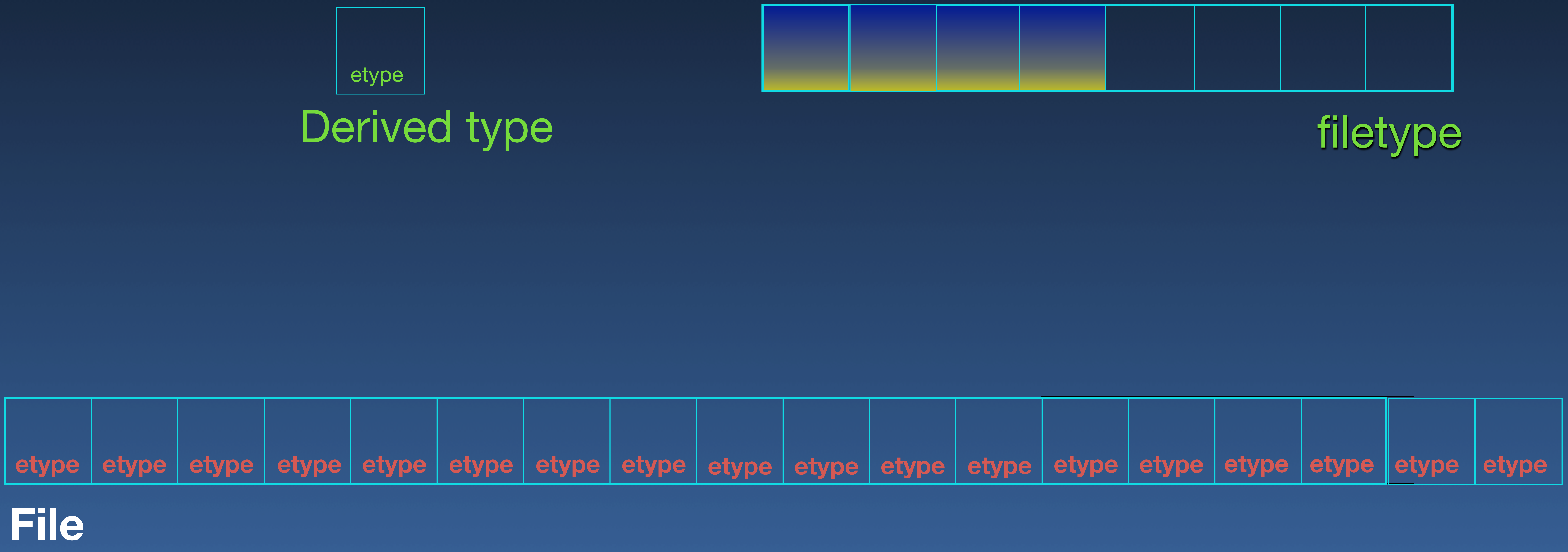
File View

Map data structures with file data

Subodh Kumar

File View

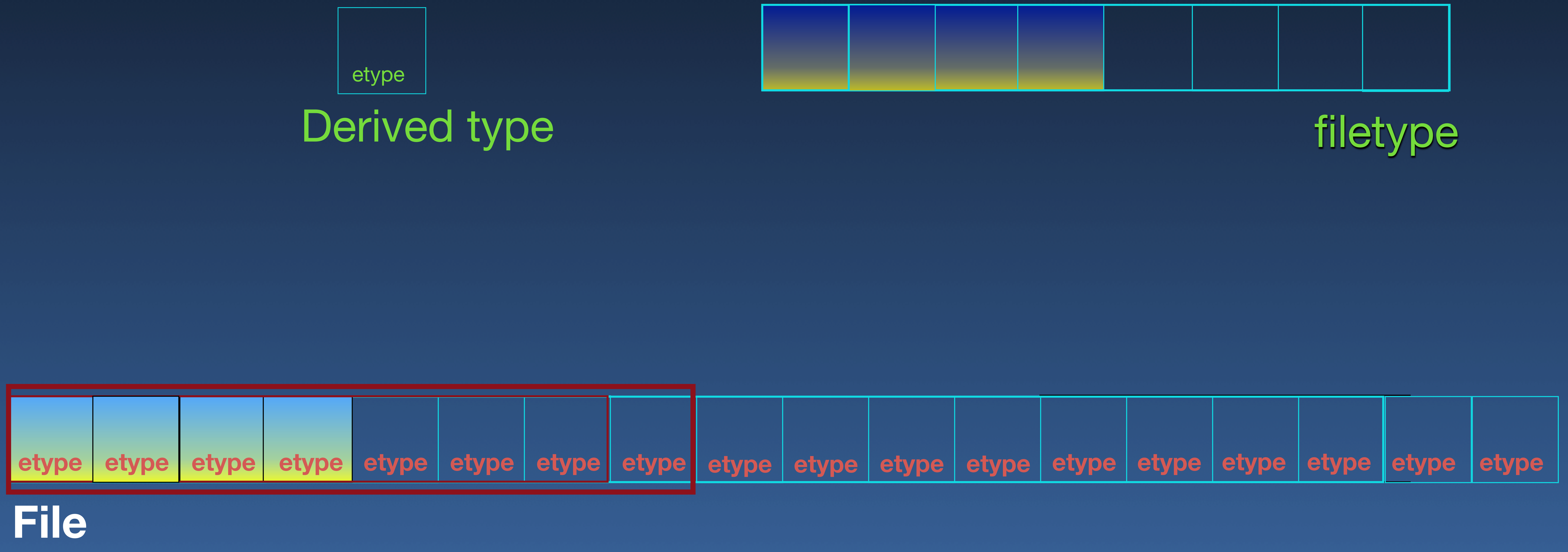
Map data structures with file data



Subodh Kumar

File View

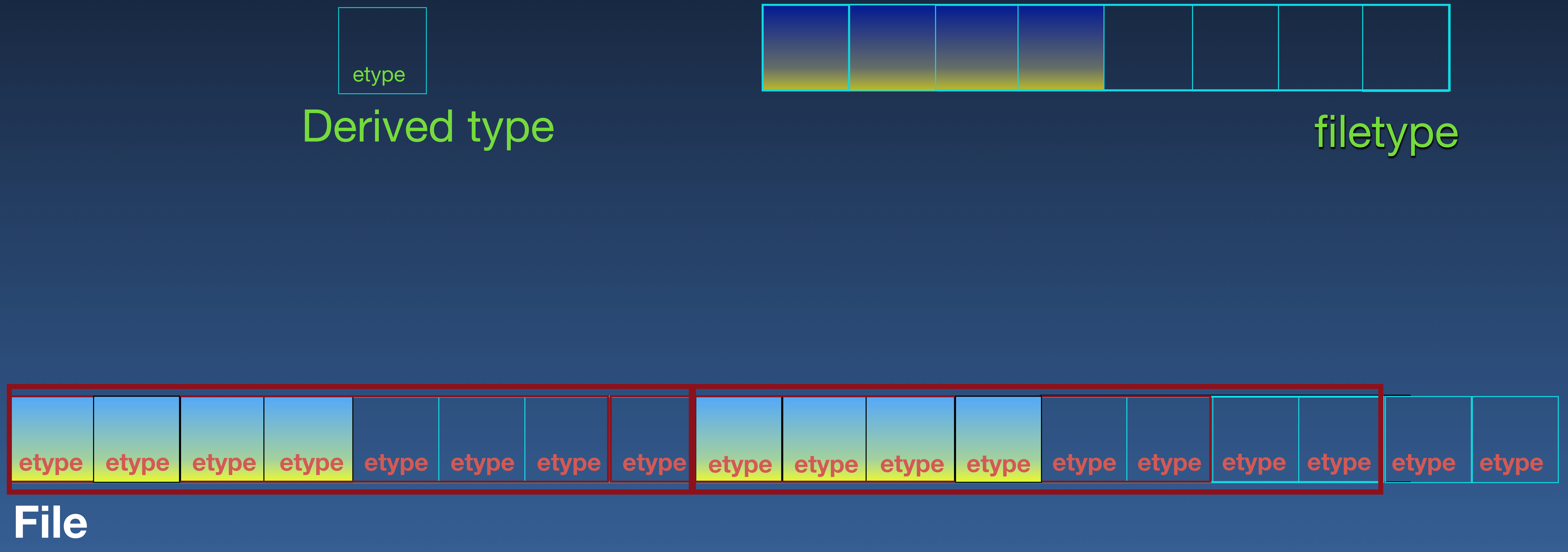
Map data structures with file data



Subodh Kumar

File View

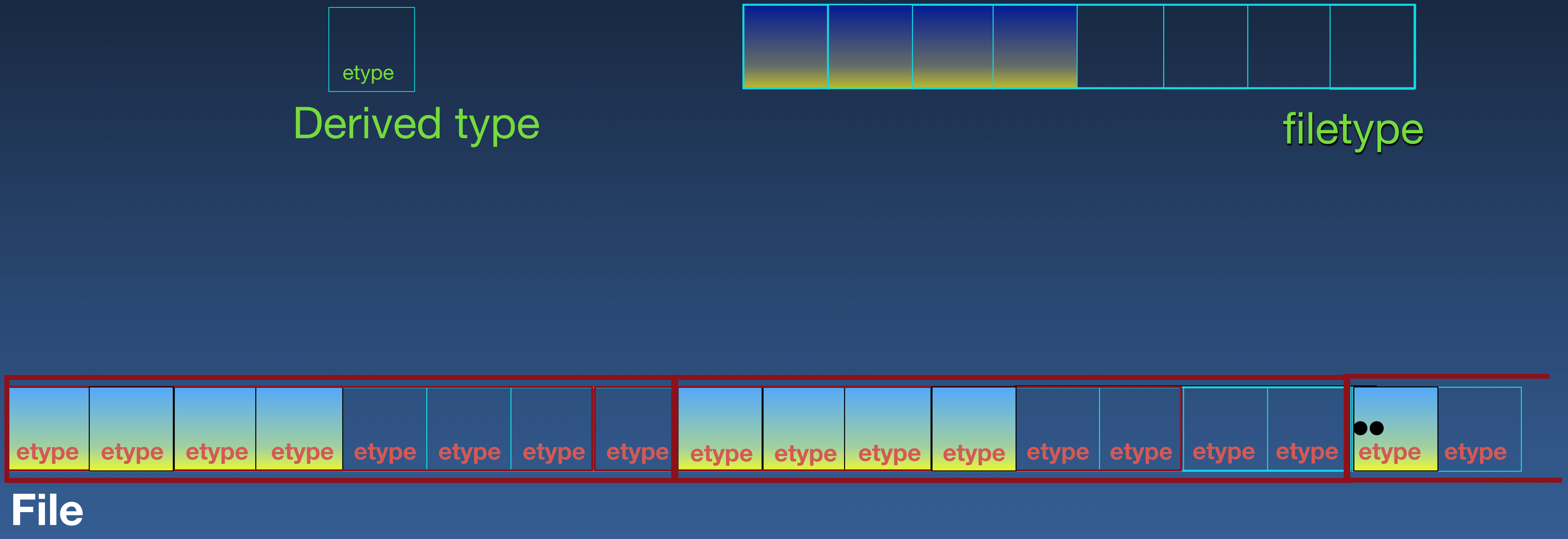
Map data structures with file data



Subodh Kumar

File View

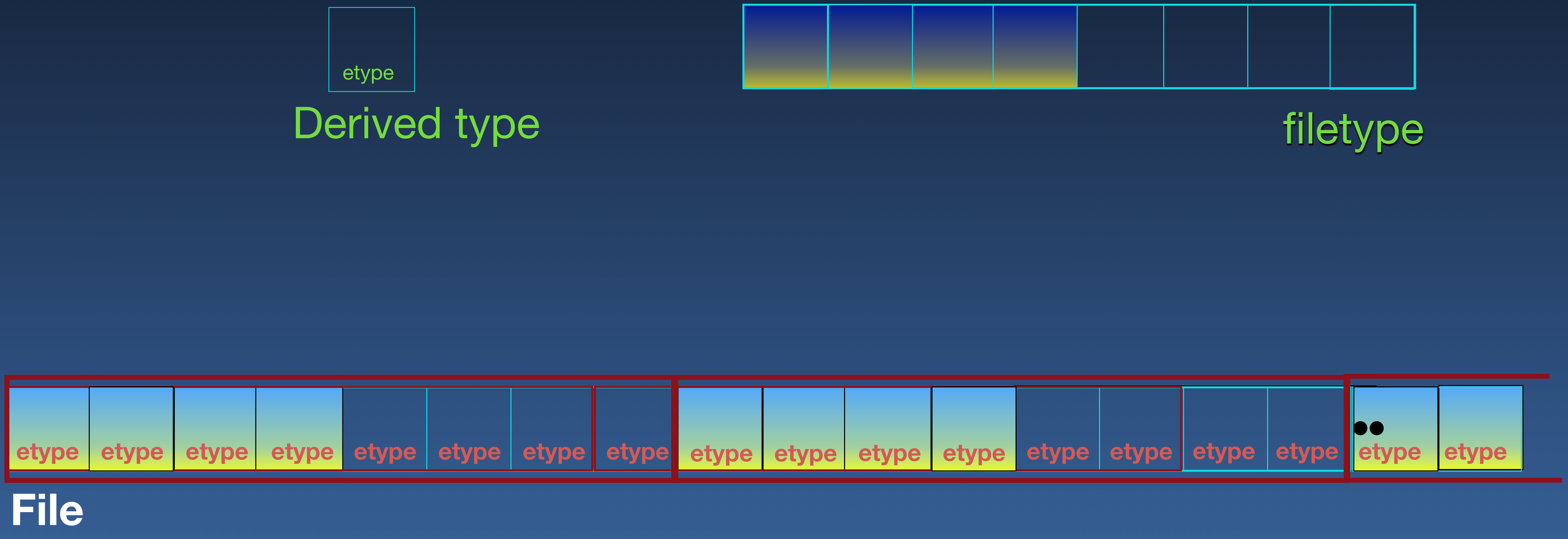
Map data structures with file data



Subodh Kumar

File View

Map data structures with file data



Example: Views in IO

block-column distribution

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &nproc );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
    displ[i] = rank + i * nproc;
    blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,"file", MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, "native", MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close ( &fh );
```

file

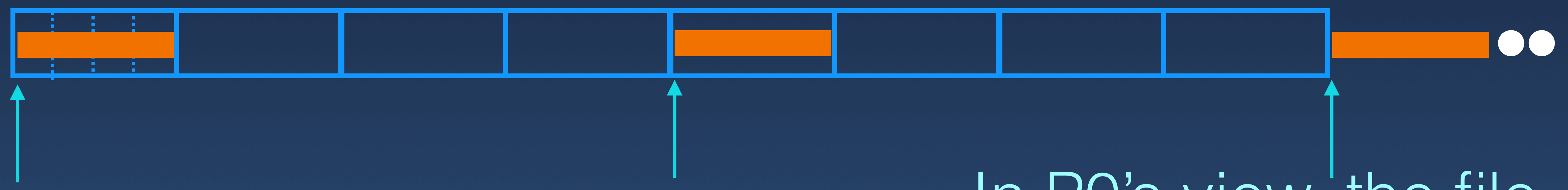
In P0's view, the file consists of only its data

Subodh Kumar

Example: Views in IO

block-column distribution

file



In P0's view, the file consists of only its data

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &nproc );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
      displ[i] = rank + i * nproc;
      blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,"file", MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, "native", MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close ( &fh );
```

Subodh Kumar

Example: Views in IO

block-column distribution

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &nproc );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
      displ[i] = rank + i * nproc;
      blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,"file", MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, "native", MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close ( &fh );
```

file

In P0's view, the file consists of only its data