COL380

Introduction to
Parallel & Distributed Programming

Agenda

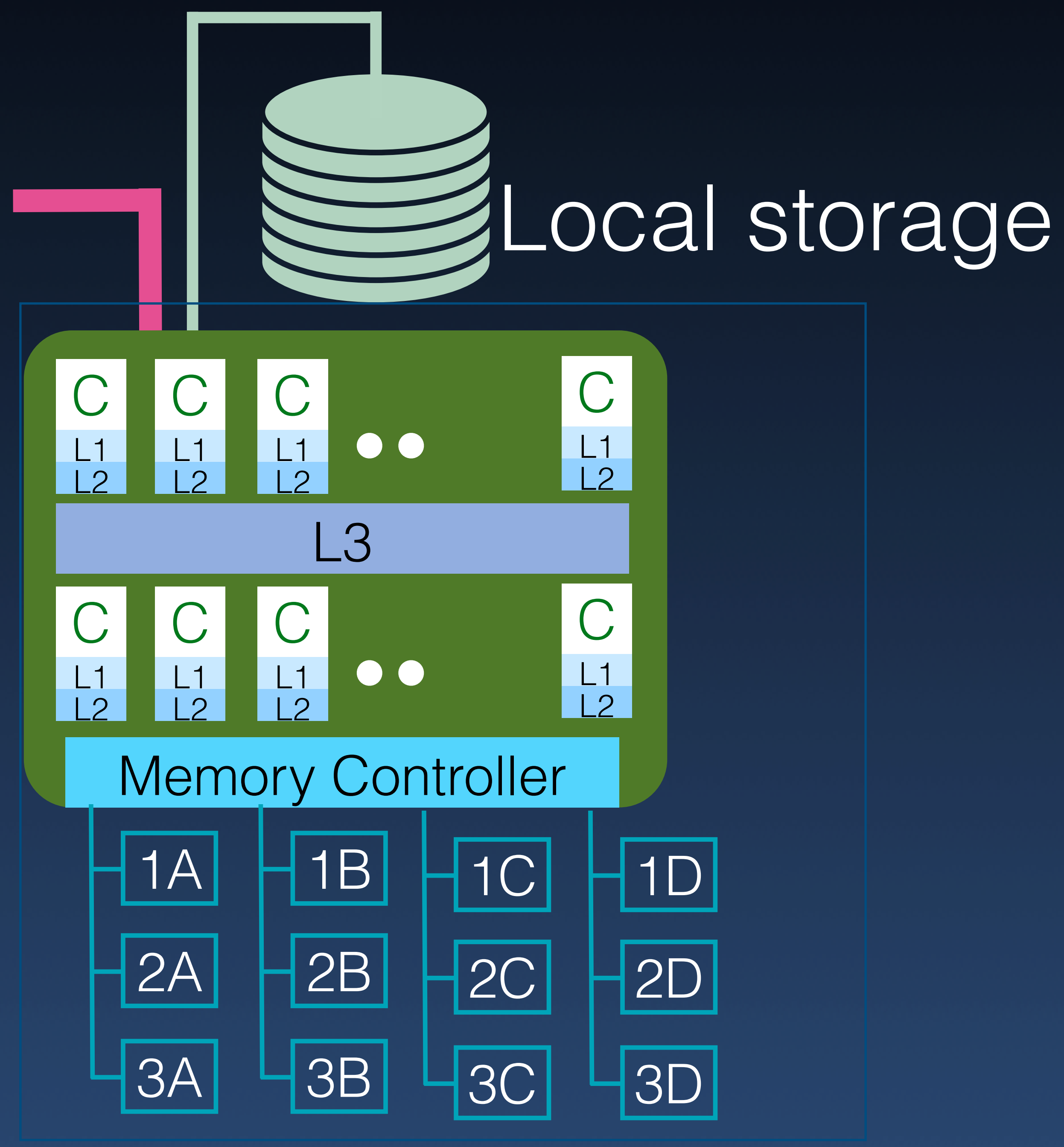
- Parallel IO
- Parallel/distributed programming frameworks

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

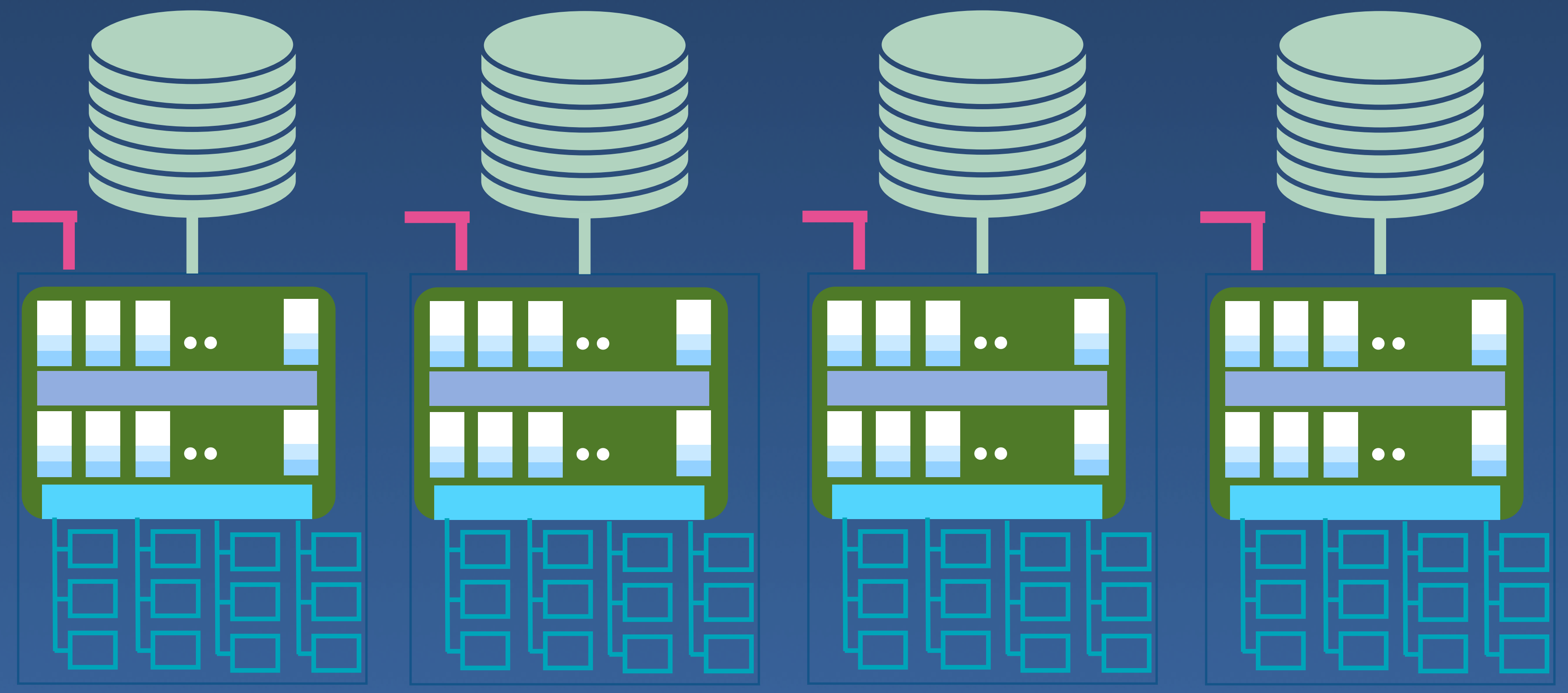
Parallel File Systems

- 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

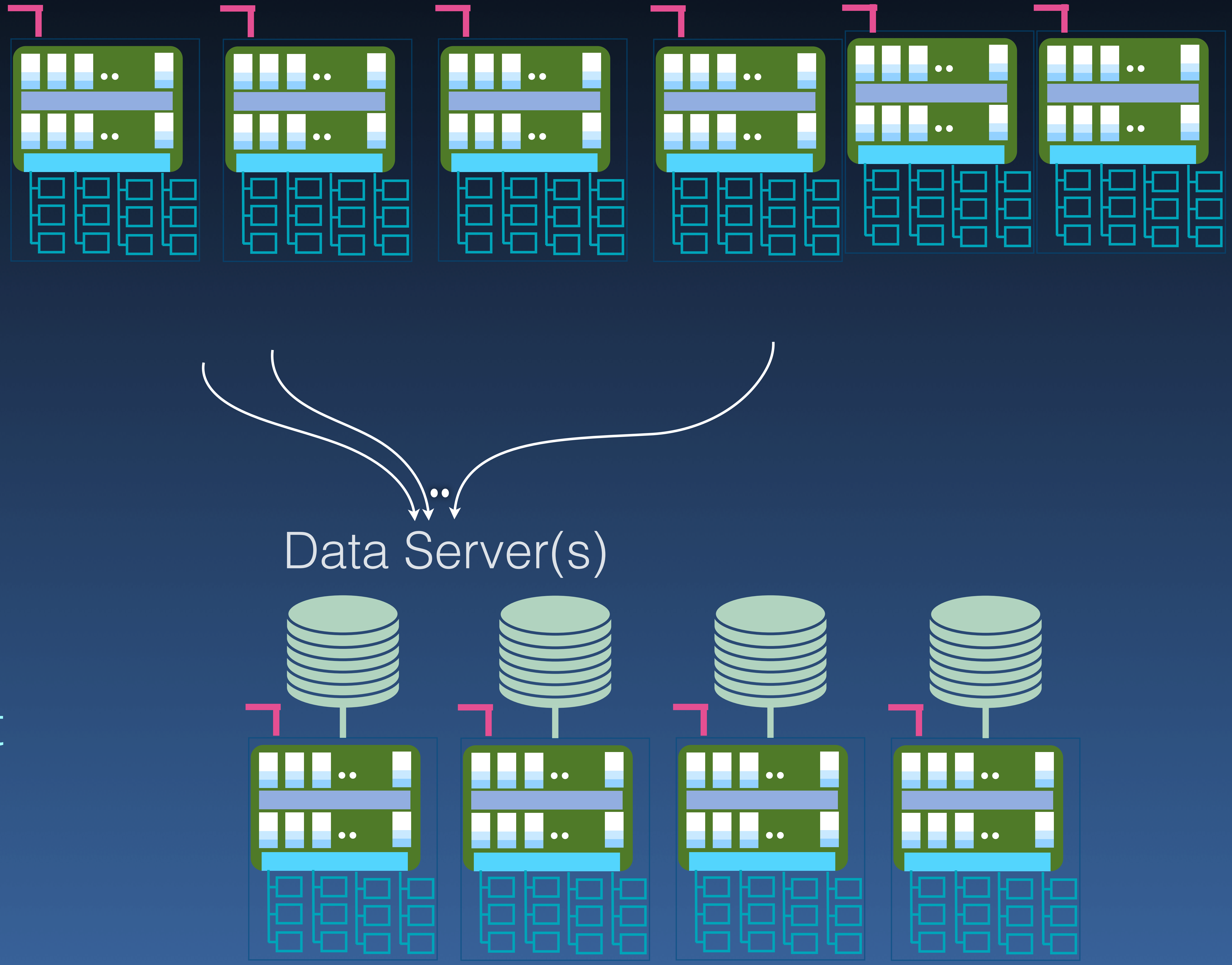


Distributed Storage

Subodh Kumar

Parallel File Systems

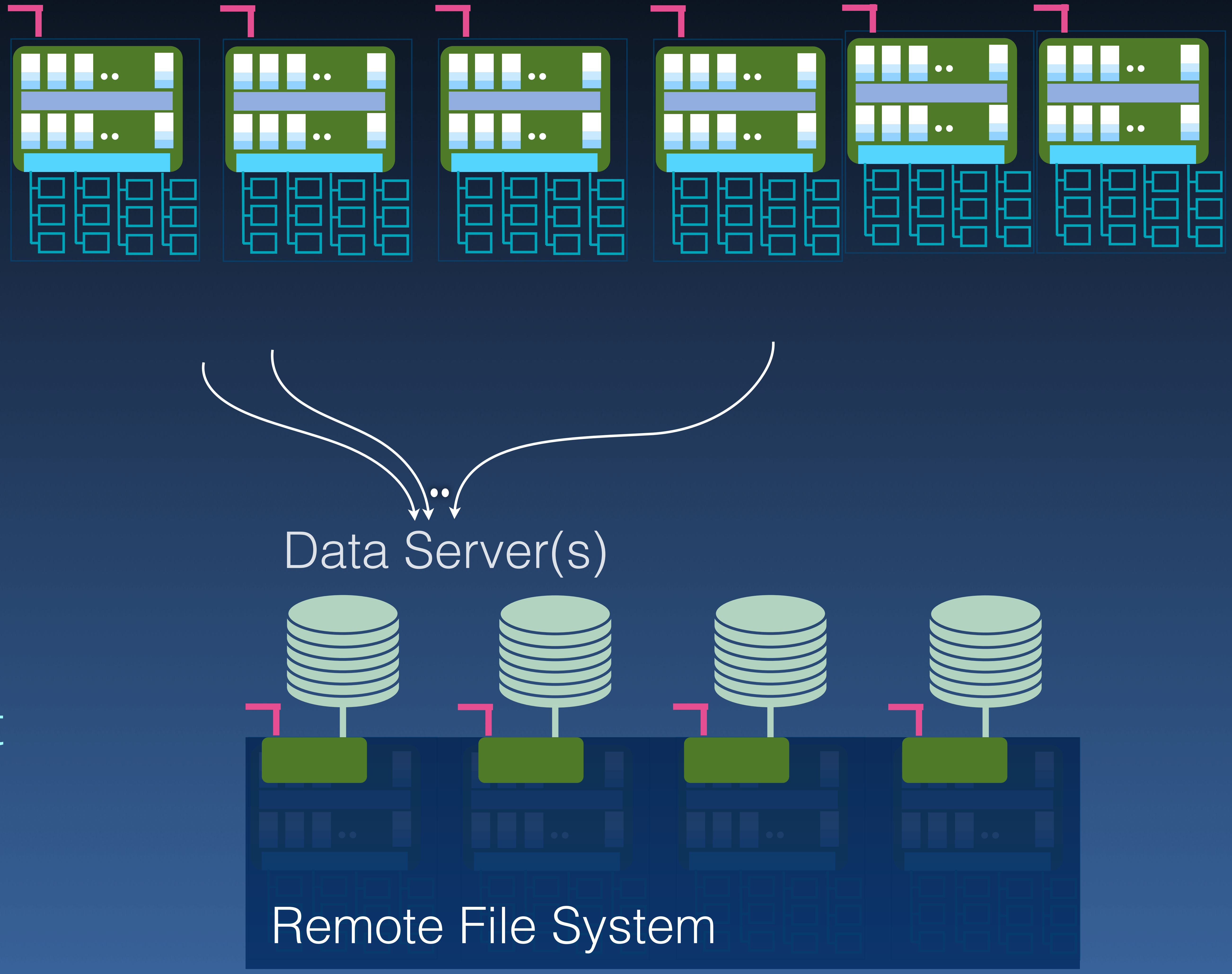
- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Subodh Kumar

Parallel File Systems

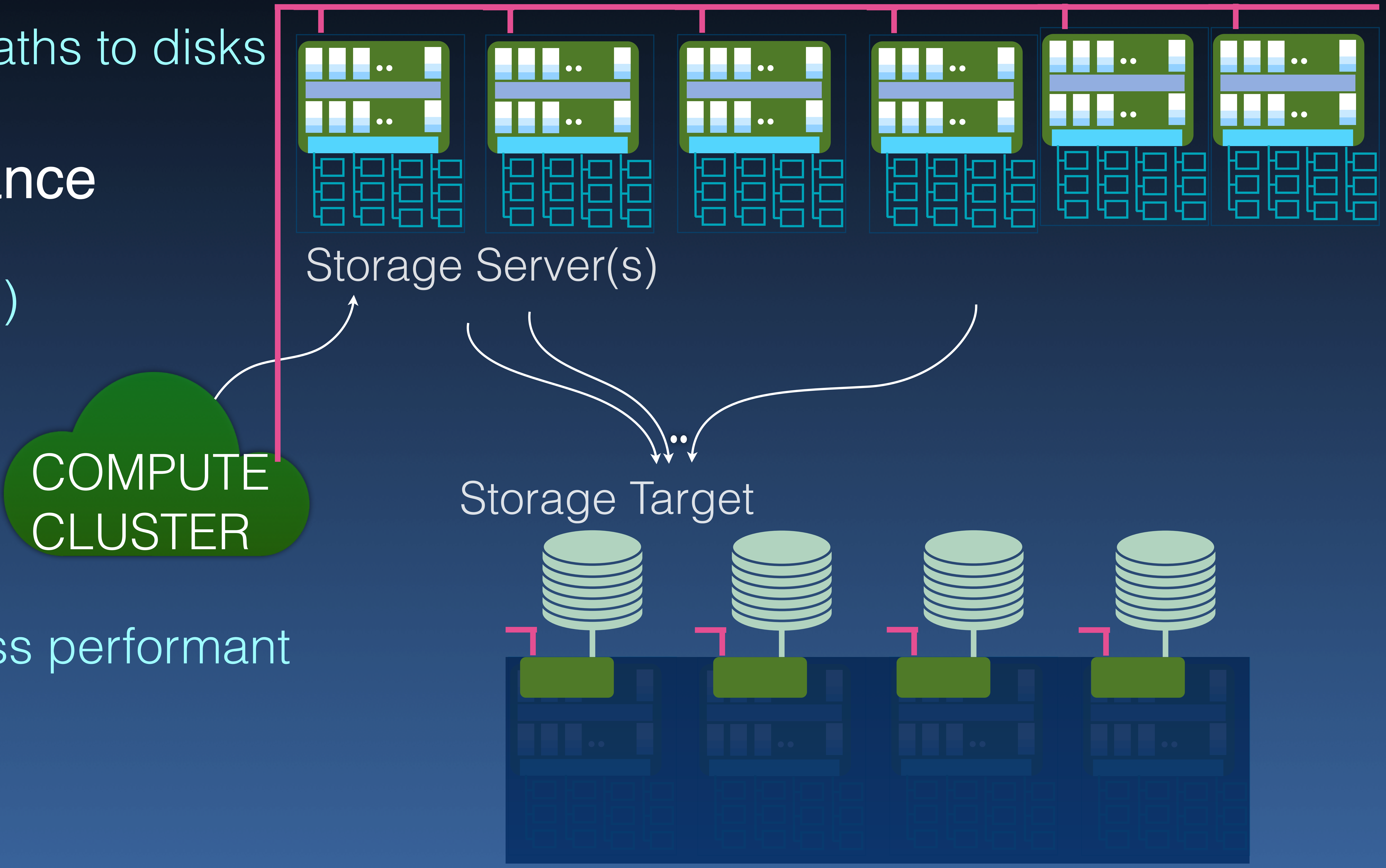
- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



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



Subodh Kumar

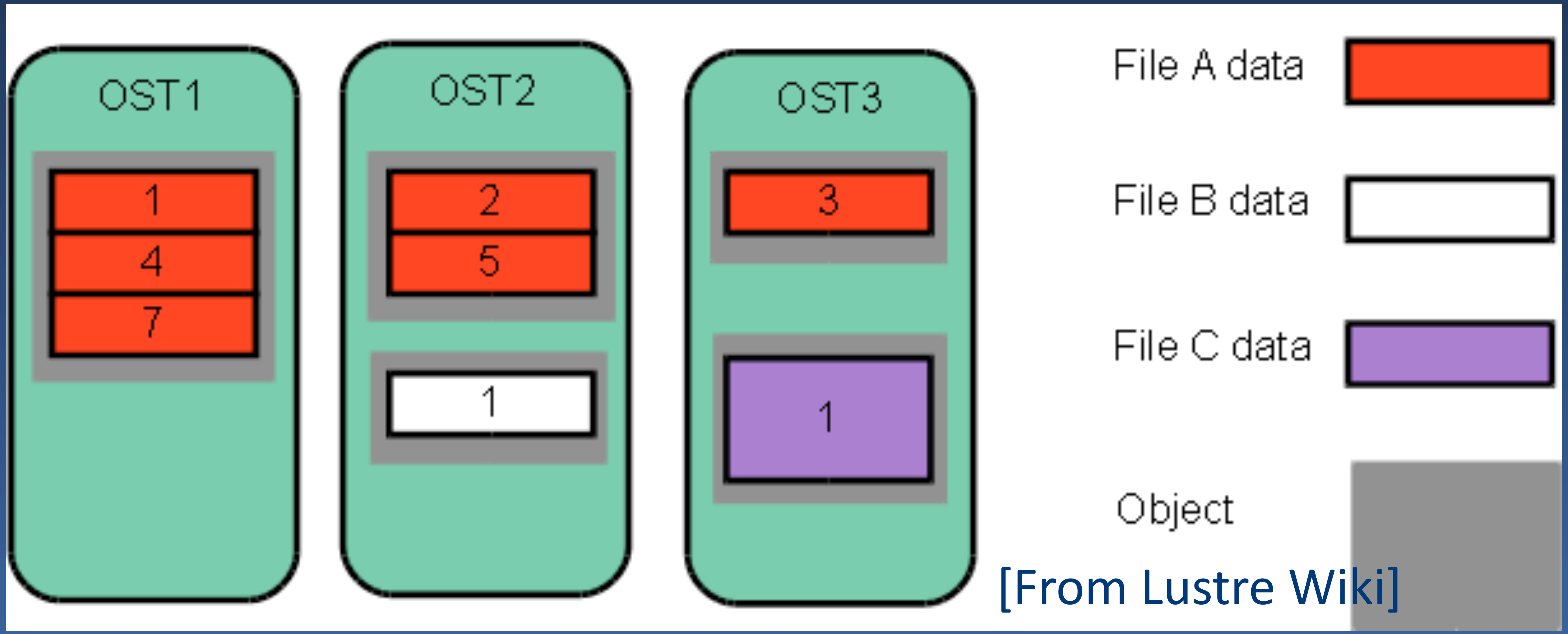
PFS Striping

- 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]

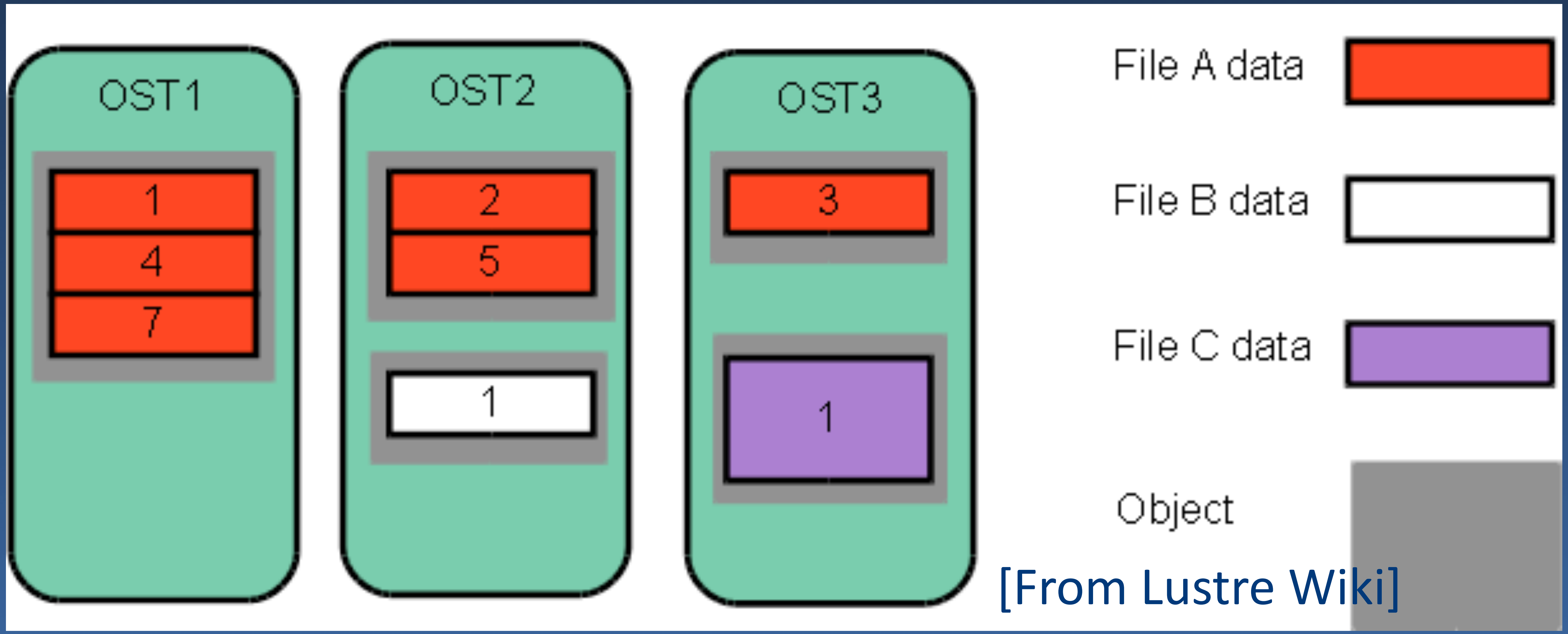
PFS Striping

- 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

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

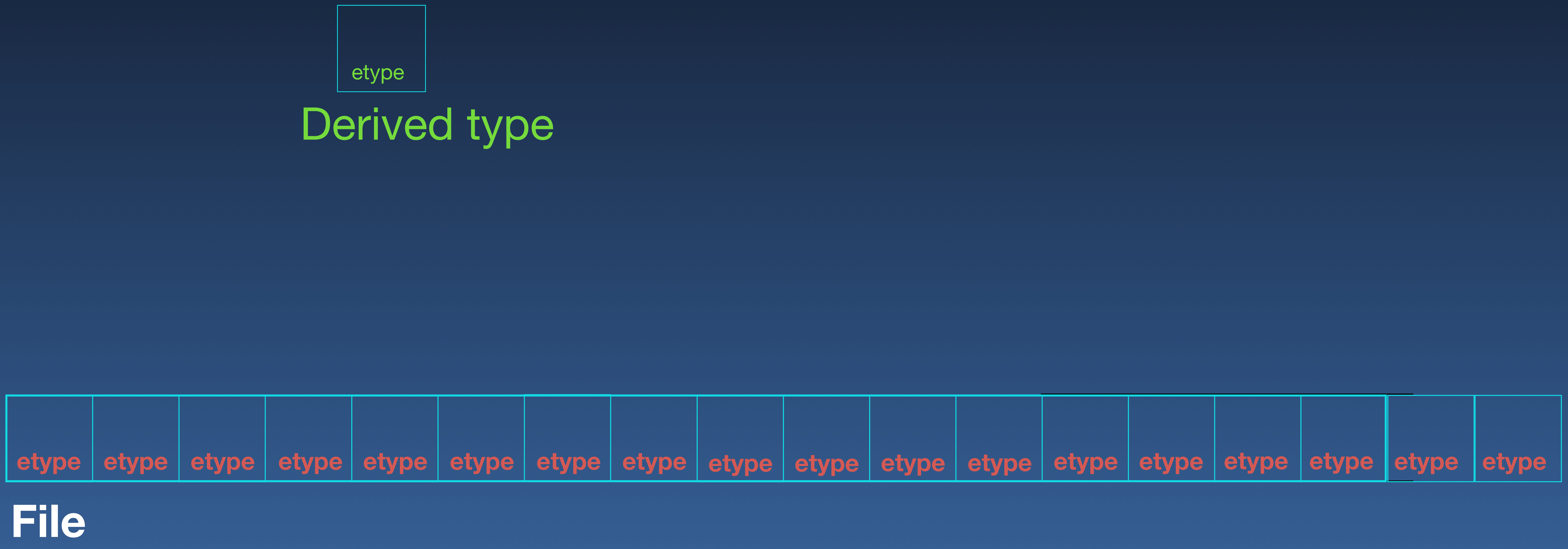
File View

Map data structures with file data

Derived type

File View

Map data structures with file data



etype

Derived type

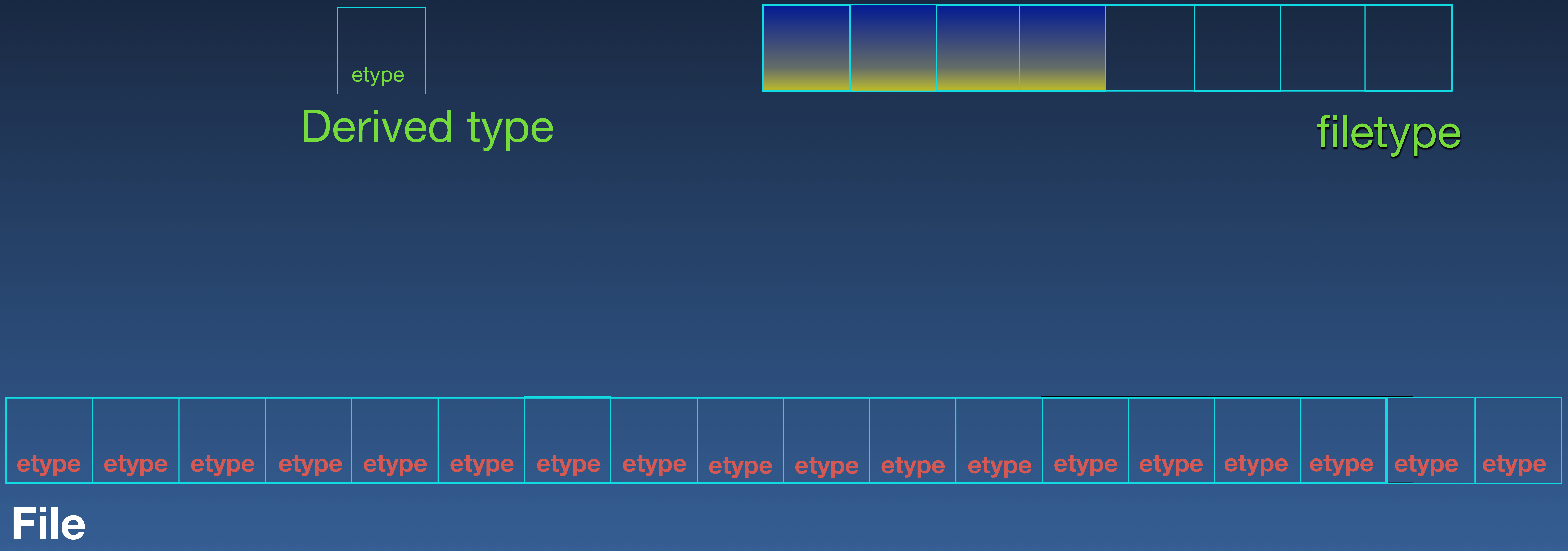
etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

Subodh Kumar

File View

Map data structures with file data



etype

Derived type

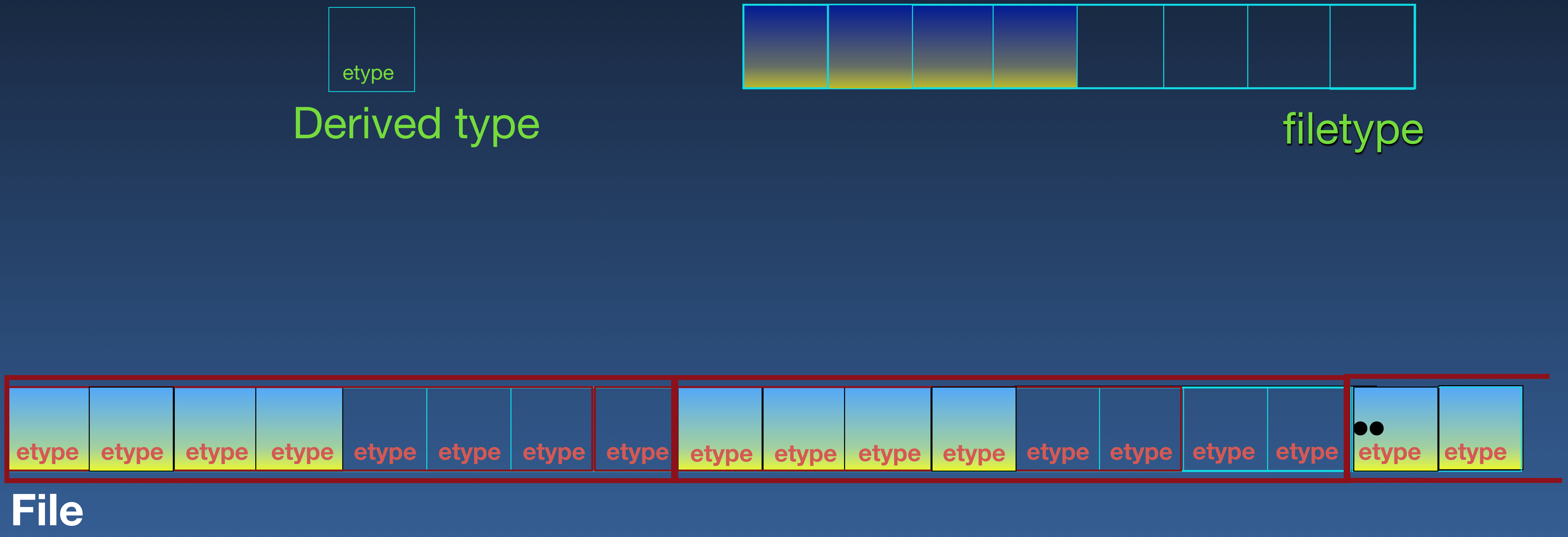
filetype

etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

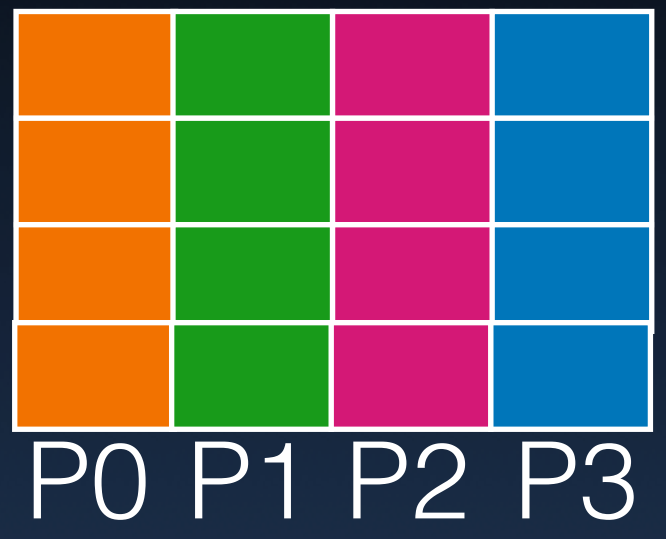
File View

Map data structures with file data



Subodh Kumar

Example: Views in IO

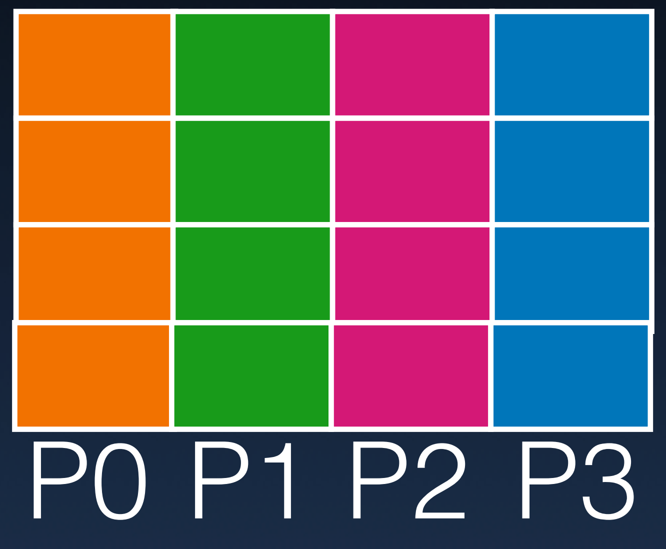


block-column distribution

```
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 );
```

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

Example: Views in IO

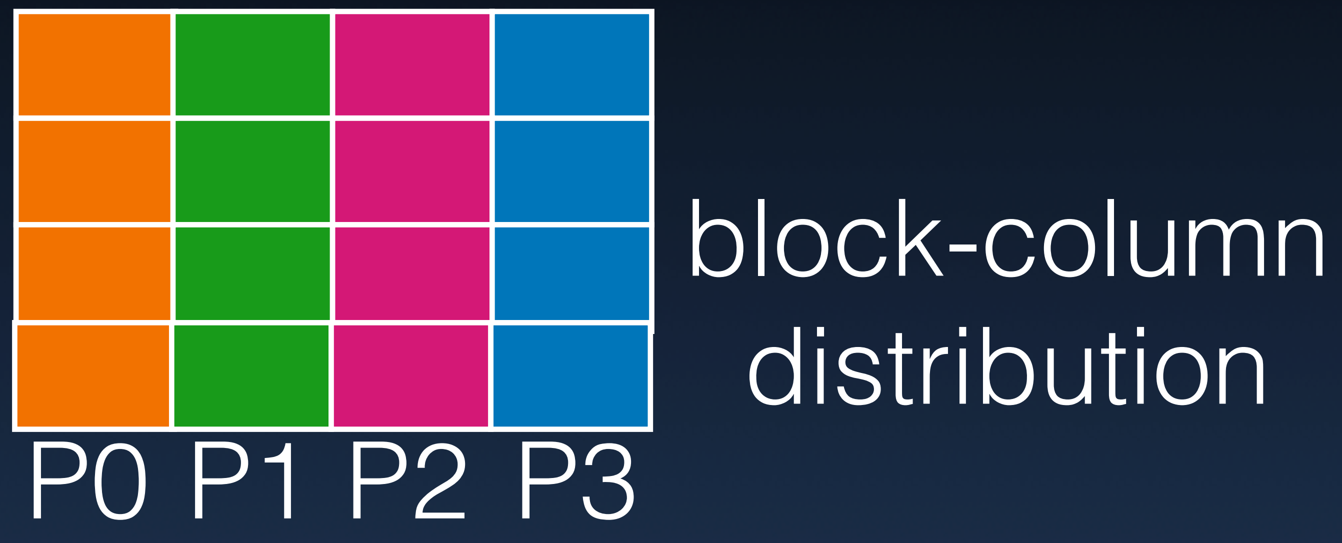


block-column distribution

```
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 );
```

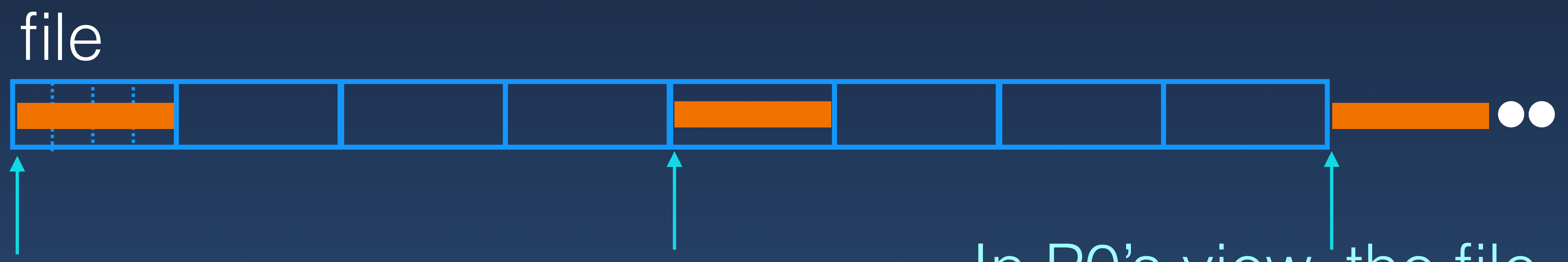
In P0’s view, the file consists of only its data

Example: Views in IO



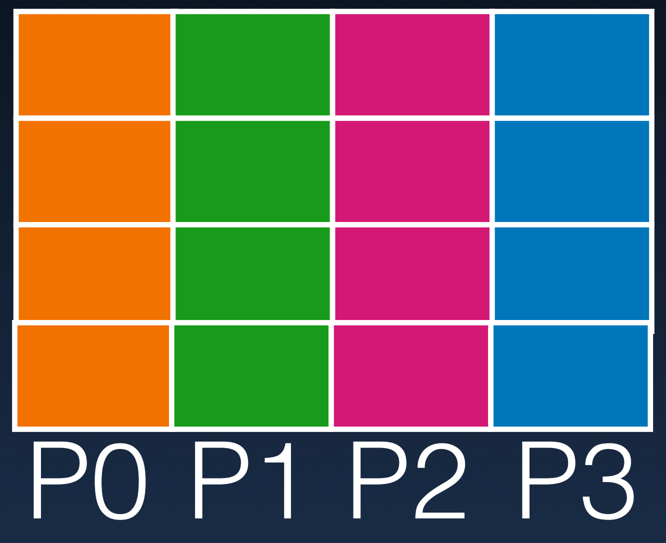
block-column distribution

```
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 );
```



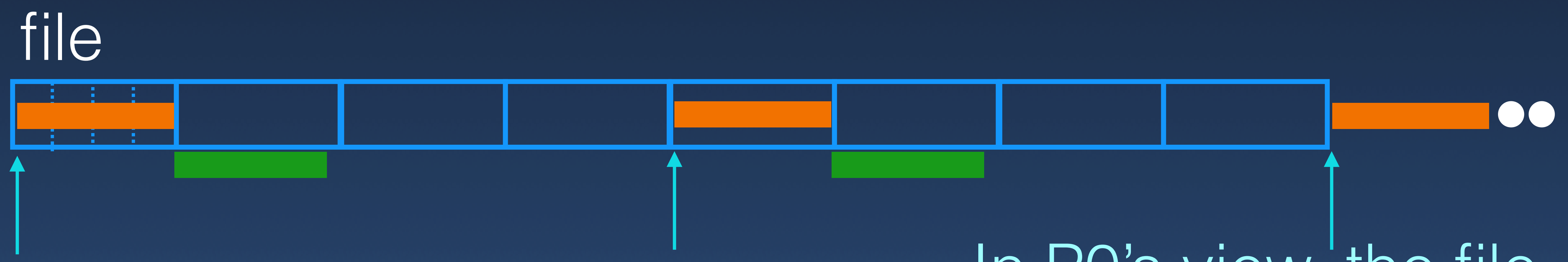
In P0’s view, the file consists of only its data

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 );
```



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

Subodh Kumar

Review

- Basic structure of parallel file system
- MPI File IO
  - ➡ Collective, individual, shared
  - ➡ Data type and file views