

Parallel File Systems

IIT Course: Data-Intensive Computing

Guest Lecture

Samuel Lang

September 20, 2010

What are Parallel File Systems?



Parallel File Systems

- ▶ Store application data *persistently*
 - usually extremely large datasets that can't fit in memory
- ▶ Provide global *shared* namespace (files, directories)
- ▶ Designed for *parallelism*
 - *Concurrent* (often coordinated) access from many clients
- ▶ Designed for *high-performance*
 - Operate over high-speed networks (IB, Myrinet, Portals)
 - Optimized I/O path for maximum bandwidth

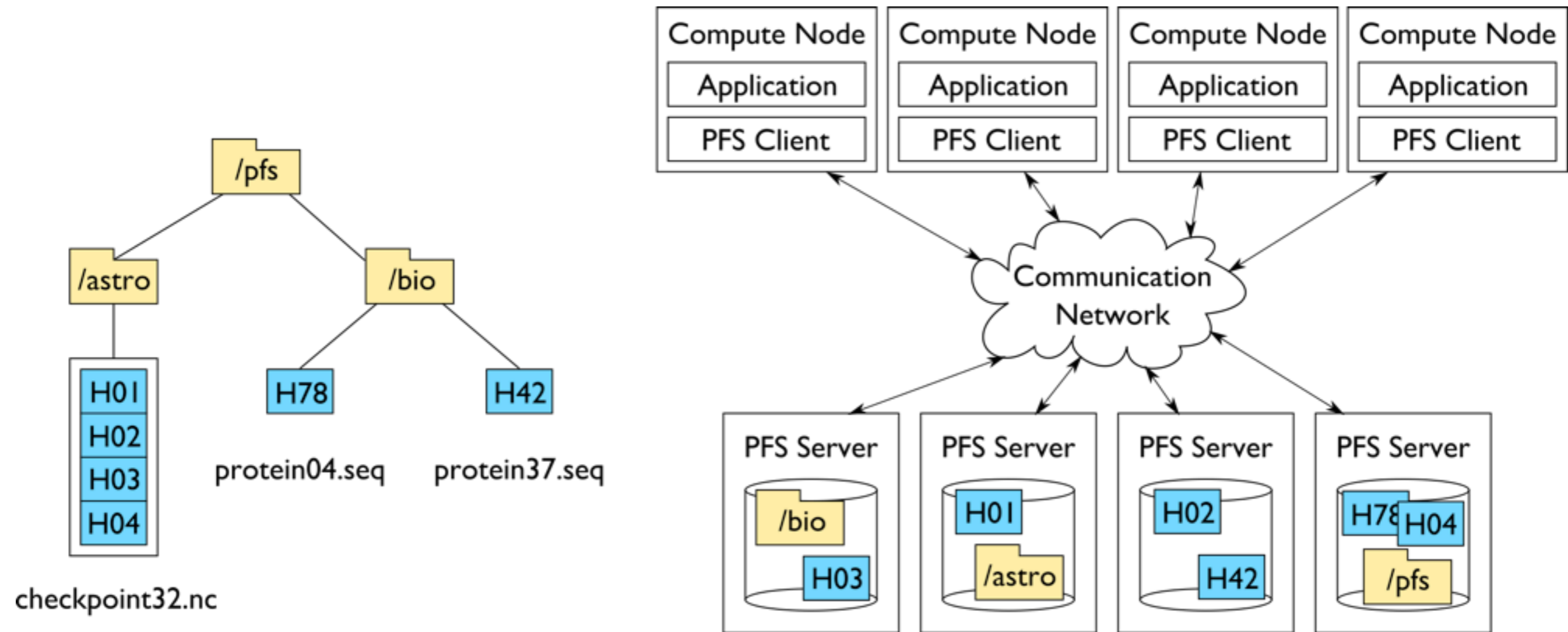


Parallel vs. Distributed

- ▶ How are Parallel File Systems different from Distributed File Systems?
- ▶ Data distribution
 - Distributed file systems often store *entire objects (files)* on a single storage node
 - Parallel file systems distribute data of a single object across multiple storage nodes
- ▶ Symmetry
 - Distributed file systems often run on architectures where the storage is co-located with the application (not always, e.g. GoogleFS, Ceph)
 - Parallel file systems are often run on architectures storage is physically separate from the compute system (not always true here either)
- ▶ Fault-tolerance
 - Distributed file systems take on fault-tolerance responsibilities
 - Parallel file systems run on enterprise shared storage
- ▶ Workloads
 - Distributed file systems are geared for loosely coupled, distributed applications (think *data-intensive*)
 - Parallel file systems target HPC applications, which tend to perform highly coordinated I/O accesses, and have massive bandwidth requirements
- ▶ Overloaded terms!
 - GlusterFS, Ceph claim to be both
 - PVFS is often run in symmetric environments



Parallel File Systems



- Provide a directory tree all nodes can see (the global name space)
- Map data across many servers and drives (parallelism of access)
- Coordinate access to data so certain access rules are followed (useful semantics)

Who uses Parallel File Systems?

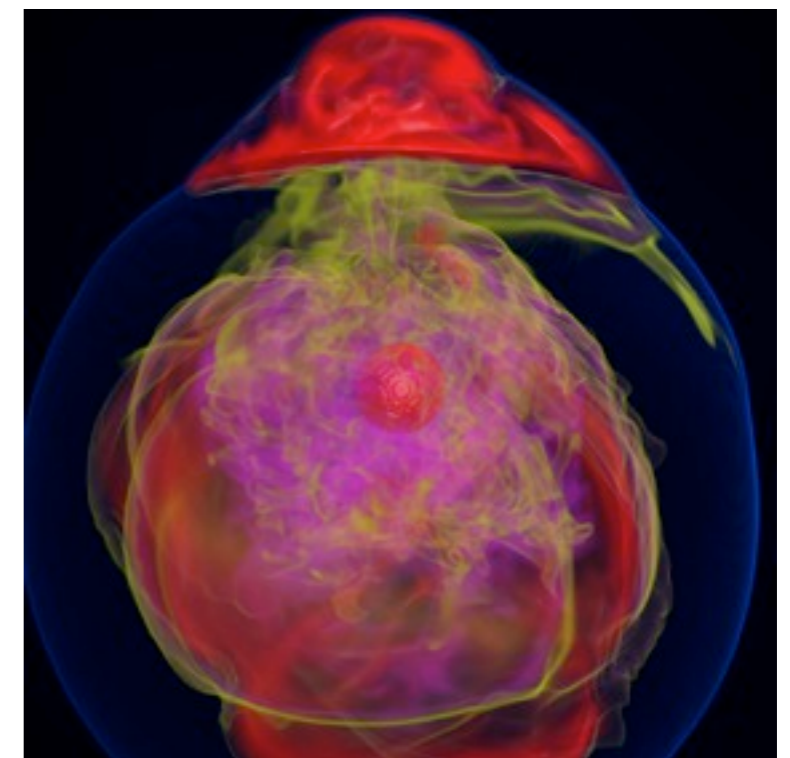


Computational Science

- Use of computer simulation as a tool for greater understanding of the real world
 - Complements experimentation and theory
- Problems are increasingly computationally challenging
 - Large parallel machines needed to perform calculations
 - Critical to leverage parallelism in all phases
- Data access is a huge challenge
 - Using parallelism to obtain performance
 - Finding usable, efficient, portable interfaces
 - Understanding and tuning I/O



IBM Blue Gene/P system at Argonne National Laboratory.



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis.

Large-Scale Data Sets

Application teams are beginning to generate 10s of Tbytes of data in a single simulation. For example, a recent run on 29K processors on the XT4 generated over 54 Tbytes of data in a 24 hour period [1].

Data requirements for select 2008 INCITE applications at ALCF

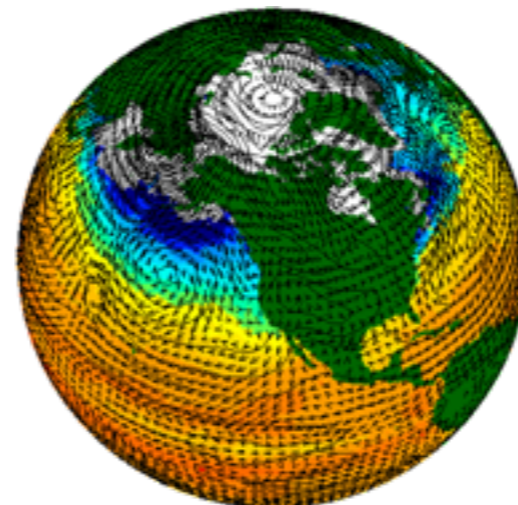
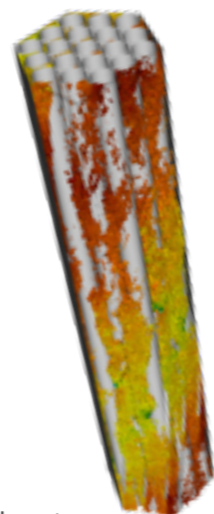
<u>PI</u>	<u>Project</u>	<u>On-Line Data</u>	<u>Off-Line Data</u>
Lamb, Don	FLASH: Buoyancy-Driven Turbulent Nuclear Burning	75TB	300TB
Fischer, Paul	Reactor Core Hydrodynamics	2TB	5TB
Dean, David	Computational Nuclear Structure	4TB	40TB
Baker, David	Computational Protein Structure	1TB	2TB
Worley, Patrick H.	Performance Evaluation and Analysis	1TB	1TB
Wolverton, Christopher	Kinetics and Thermodynamics of Metal and Complex Hydride Nanoparticles	5TB	100TB
Washington, Warren	Climate Science	10TB	345TB
Tsigelny, Igor	Parkinson's Disease	2.5TB	50TB
Tang, William	Plasma Microturbulence	2TB	10TB
Sugar, Robert	Lattice QCD	1TB	44TB
Siegel, Andrew	Thermal Striping in Sodium Cooled Reactors	4TB	8TB
Roux, Benoit	Gating Mechanisms of Membrane Proteins	10TB	10TB

[1] S. Klasky, personal correspondence, June 19, 2008.



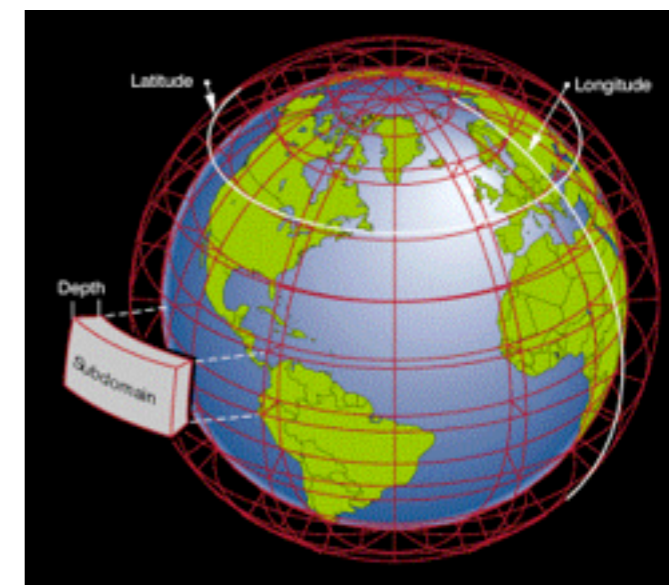
Drilling Down on Selected Applications

	Nuclear reactor simulation	Climate modeling	Astrophysics
<u>Preliminary studies</u>			
# of elements	4.5 million	3 million	70-500 million grid points
# of variables	7	100 (30 are vectors)	4 (1 is a vector)
# of timesteps	20,000	200,000-400,000	1,800
Total data size	2.5 Tbytes	30-120 Tbytes	80 Tbytes
<u>Science runs</u>			
# of elements	120 million	6 million	4.3 billion grid points
# of timesteps	90,000	4 million	1,800
Total data size	900-1200 Tbytes	1.2 Pbytes	50 Tbytes

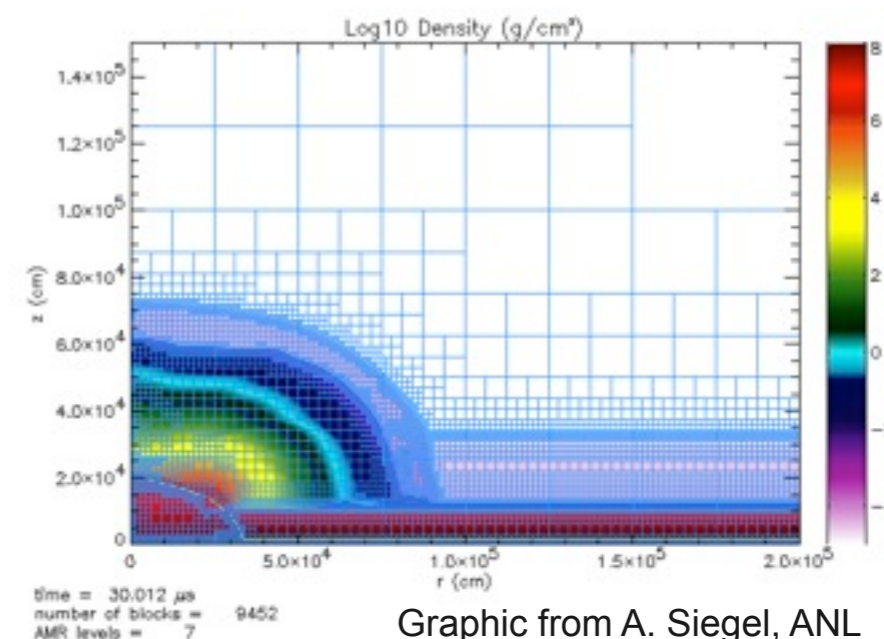


Application and Storage Data Models

- Applications have data models appropriate to domain
 - Multidimensional typed arrays, images composed of scan lines, variable length records
 - Headers, attributes on data
- I/O systems have very simple data models
 - Tree-based hierarchy of containers
 - Some containers have streams of bytes (files)
 - Others hold collections of other containers (directories or folders)
- High-level I/O libraries help map between these data models



Graphic from J. Tannahill, LLNL



Graphic from A. Siegel, ANL

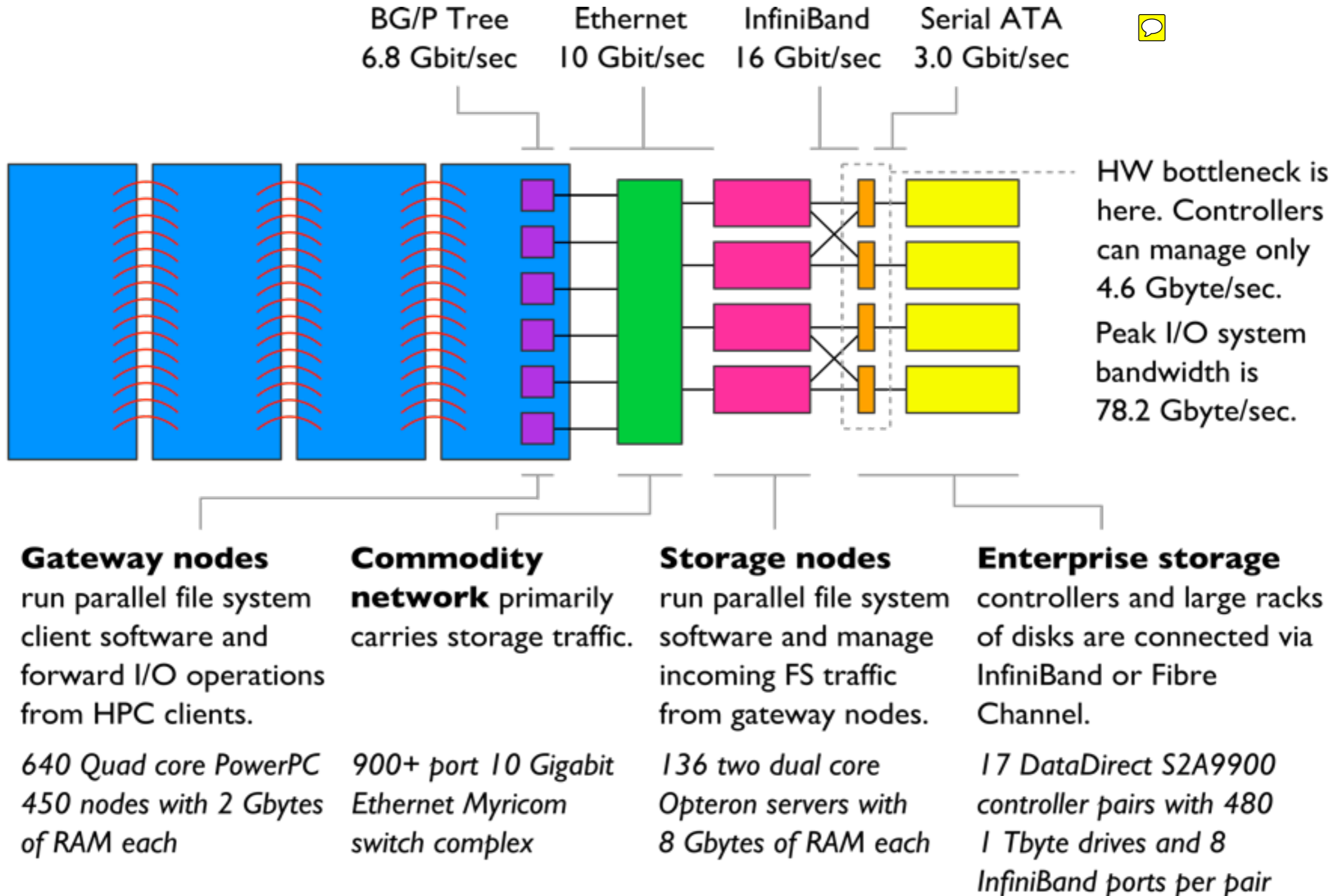
Shared-file vs. File-per-process

- Scientific applications perform I/O to parallel file system in primarily one of two ways:
 - **Shared-file (N-to-1):** A single file is created, and all application tasks write to that file (usually to completely disjoint regions)
 - Increases usability: only one file to keep of by application
 - Can create lock contention and hinder performance on some systems
 - **File-per-process (N-to-N):** Each application task creates a separate file, and writes to that only that file.
 - Avoids lock contention on file systems that use locks to maintain POSIX consistency
 - Applications running today create as many as 100,000 tasks
 - Impossible to restart application with different number of tasks



Where are Parallel File Systems deployed?

Intrepid Parallel Storage System



Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.

I/O Software Stack on Intrepid

High Level I/O Library
maps application abstractions onto storage abstractions and provides data portability

HDF5, Parallel NetCDF

I/O Forwarding
Bridges between application tasks and storage system

IBM ciod

Application

High Level I/O Library

I/O Middleware

I/O Forwarding

Parallel File System

I/O Hardware

I/O Middleware

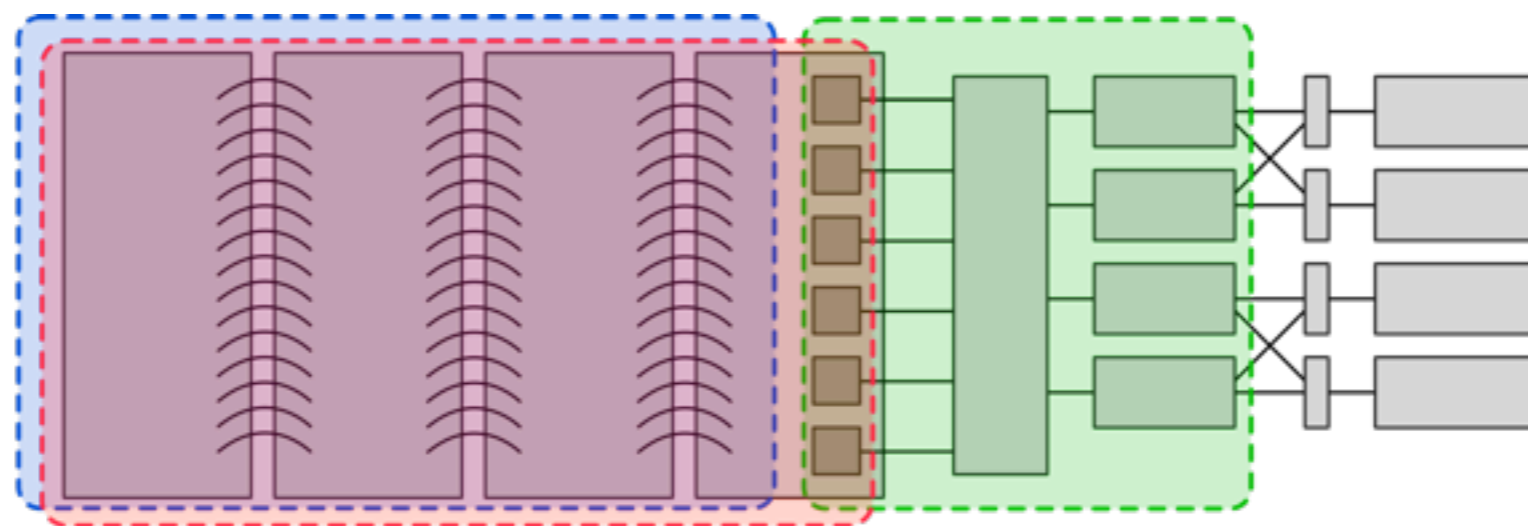
Organizes accesses from many processes, especially those using collective I/O

MPI-IO

Parallel File System

Maintains logical space and provides efficient access to data

PVFS



Compute nodes

Gateway nodes

Commodity network

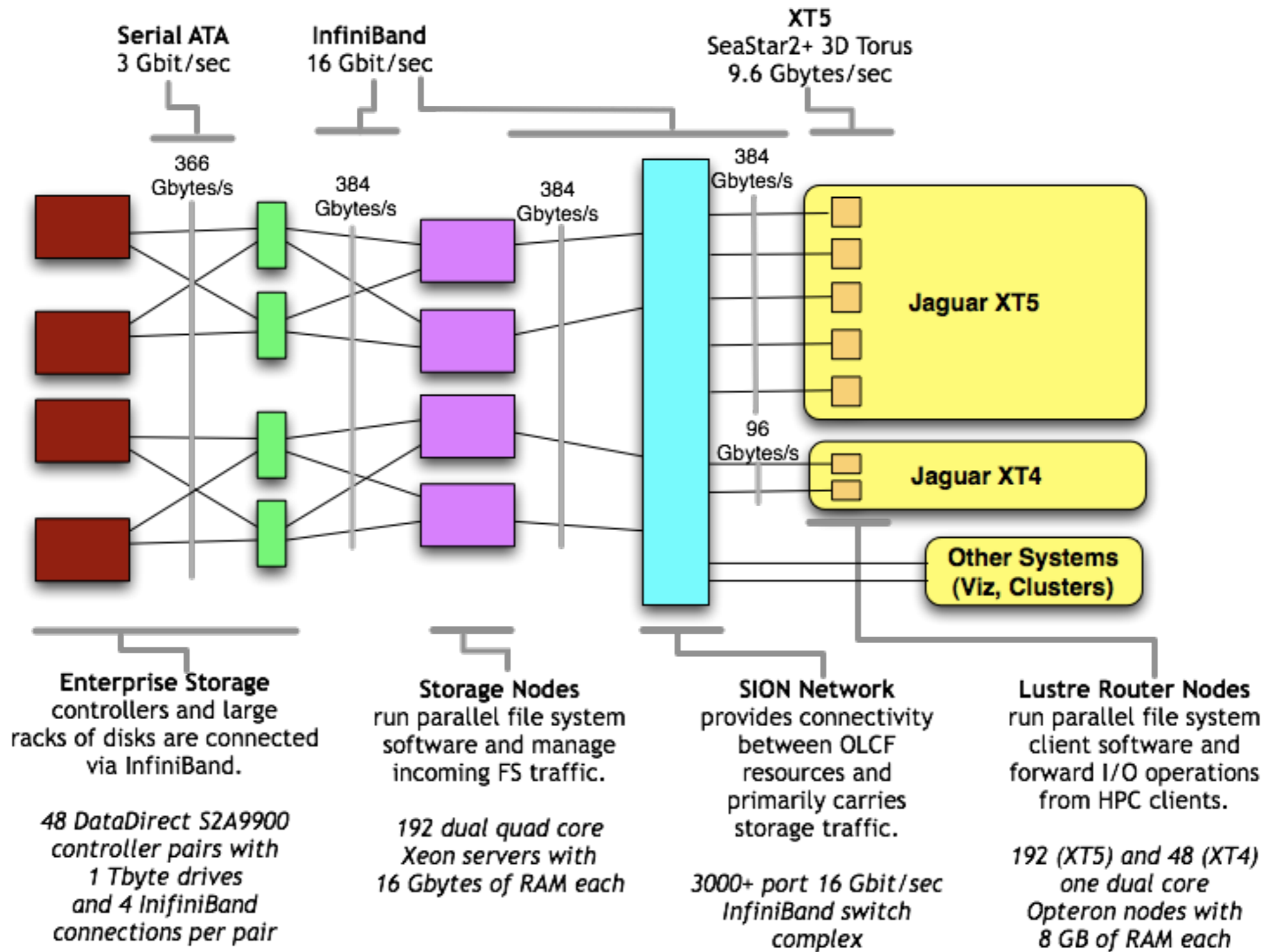
Storage nodes

Enterprise storage

Computer Science Division / Argonne National Laboratory

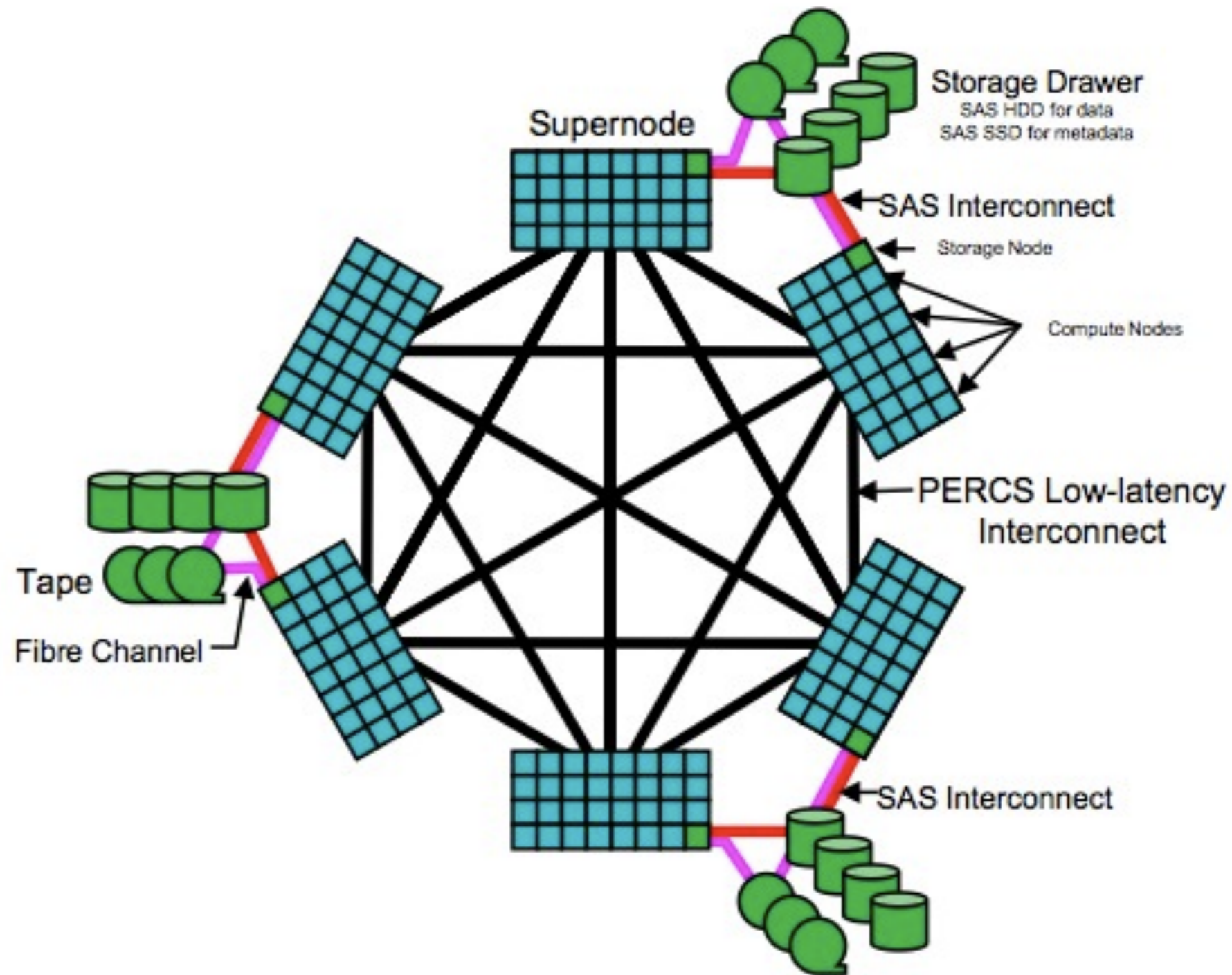


Jaguar Storage System



Blue Waters Storage System

- ▶ File System (GPFS) runs directly on compute nodes
- ▶ Storage nodes and physical storage embedded in compute racks
 - JBOD with software RAID
 - Metadata embedded as well
- ▶ All I/O messages use internal fabric
 - lower latency to storage
 - reduced cost
 - may cause contention between I/O heavy and communication heavy applications
- ▶ **1.5 TB/s** peak bandwidth
- ▶ **18 Petabytes** of Storage



How are Parallel File Systems designed?

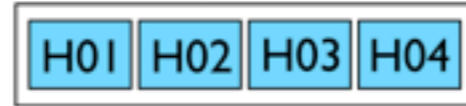
Data distribution in parallel file systems

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

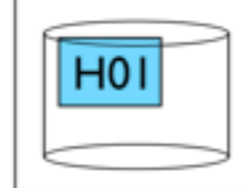
Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.

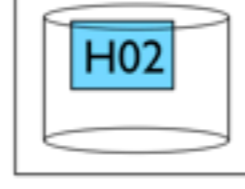
checkpoint32.nc



PFS Server



PFS Server



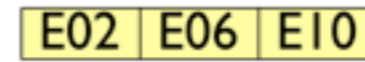
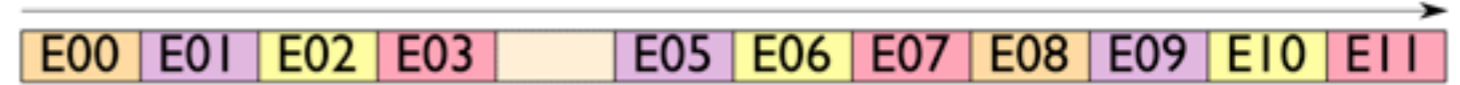
PFS Server



PFS Server



Offset in File

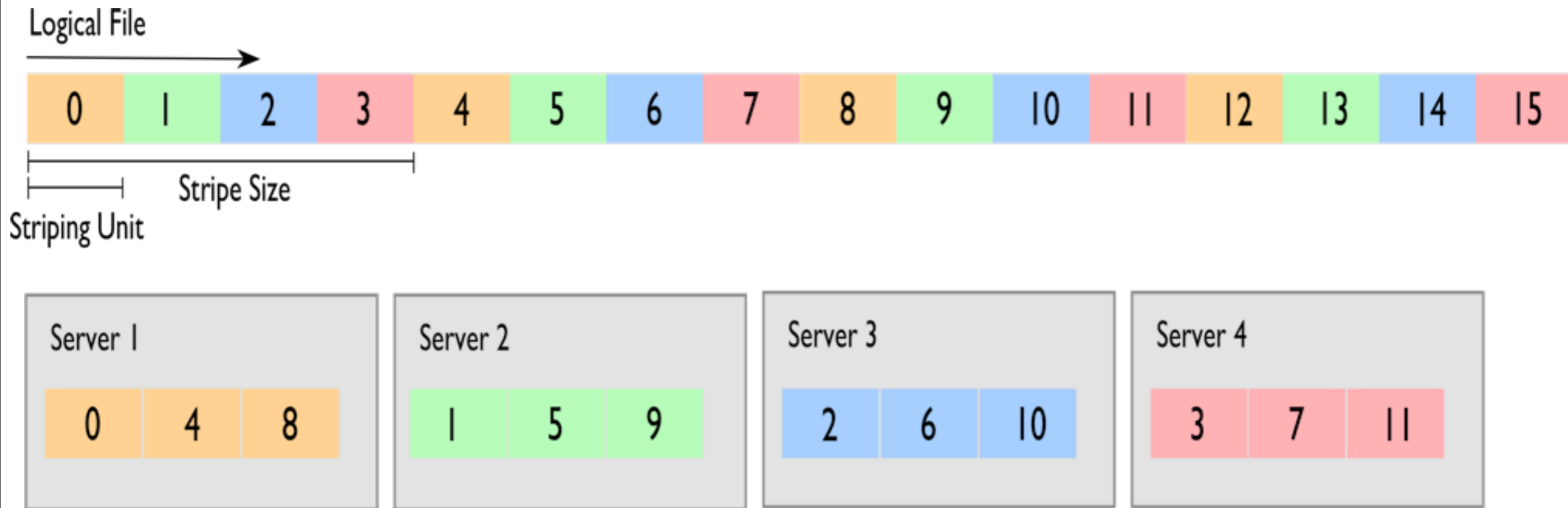


Space is allocated on demand, so unwritten "holes" in the logical file do not consume disk space.

A static mapping from logical file to objects allows clients to easily calculate server(s) to contact for specific regions, eliminating need to interact with a metadata server on each I/O operation.

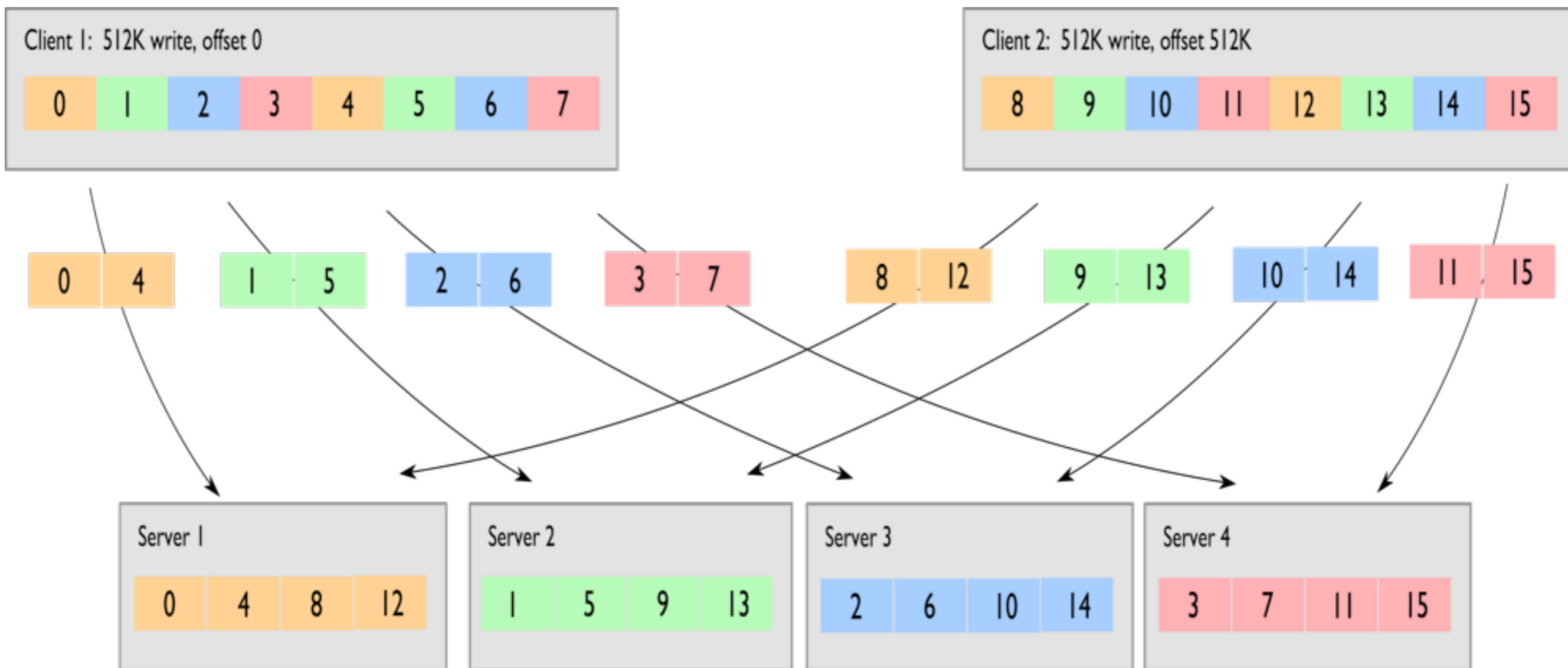
Data Distribution

- Round-round is a reasonable default solution
 - Works consistently for a variety of workloads
 - Works well on most systems
 - Who uses it? GPFS, Lustre, PVFS...
 - Can you think of a system where this might not work so well?
 - What other distributions could be used?



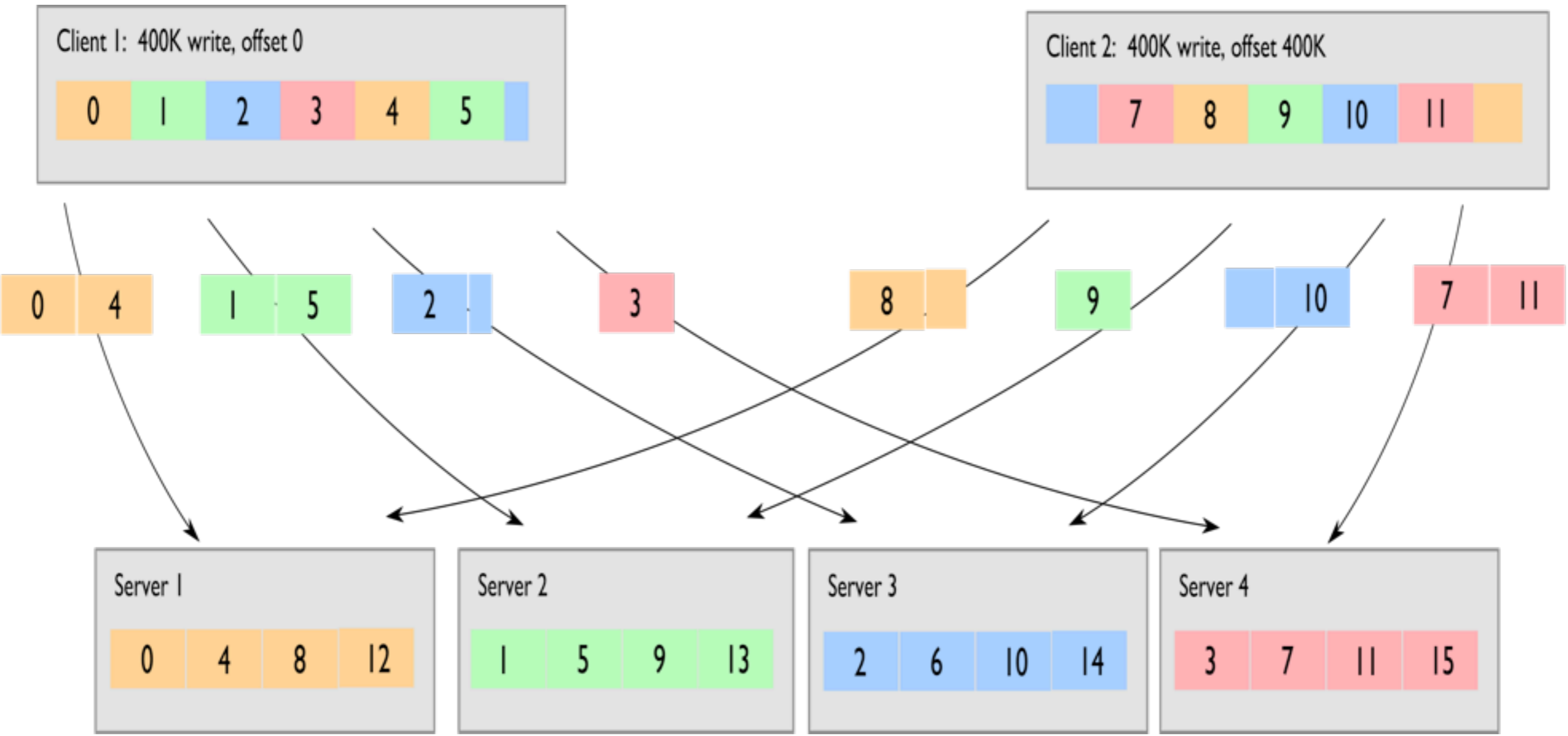
Data Distribution

- Clients perform writes/reads of file at various regions
 - Usually depends on application workload and number of tasks



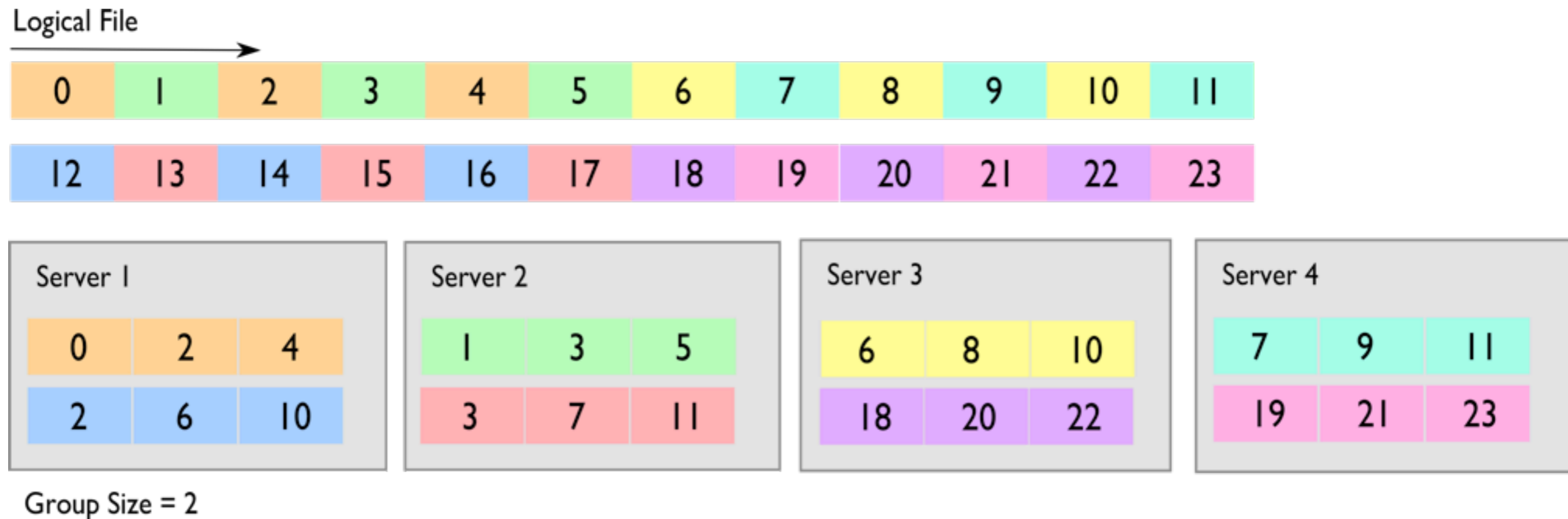
Data Distribution

- Sizes of requests, alignment to striping unit is important



Data Distribution continued...

- What happens when we have many servers (hundreds)?
- Two-dimensional distributions help
- Can also limit number of servers per file



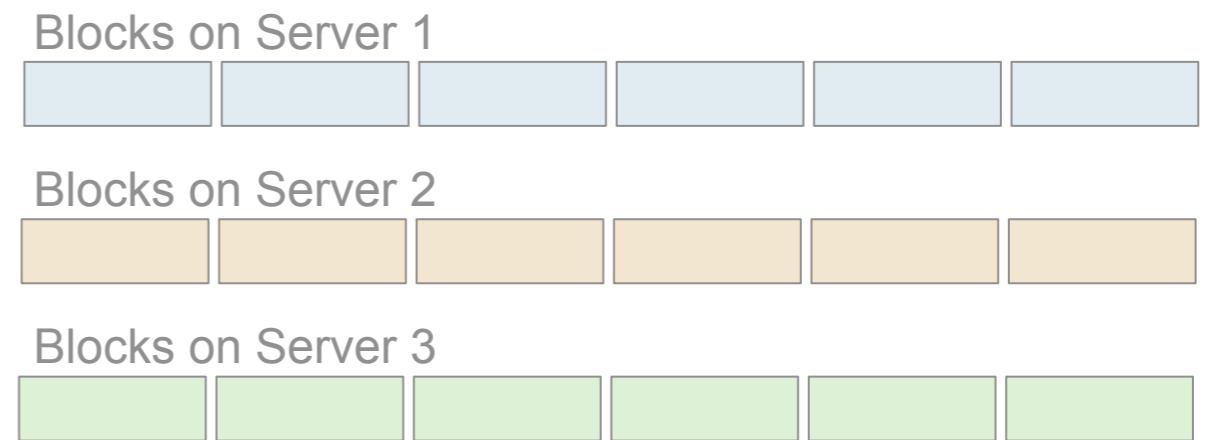
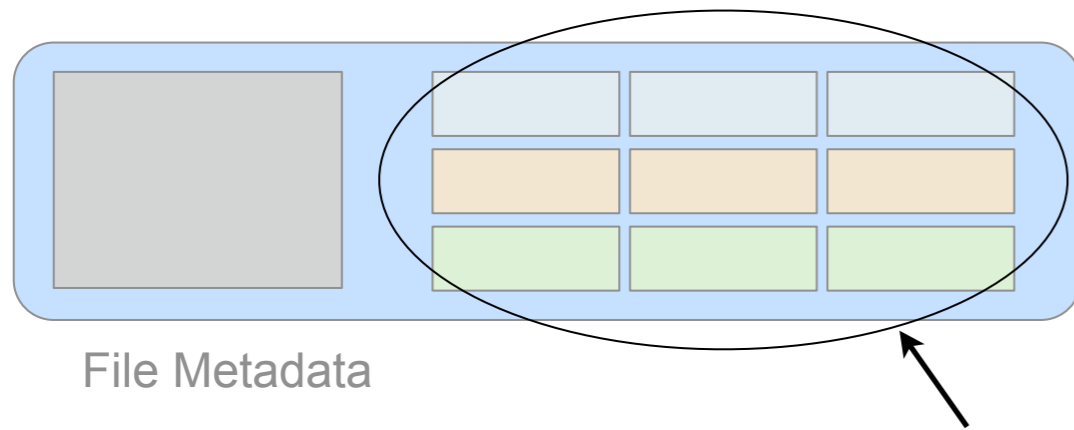
Classes of Parallel File Systems: Blocks vs. Objects

- Block-Based Parallel File Systems (AKA “Shared-disk”)
 - Blocks are *fixed-width*
 - File growth requires more blocks
 - Blocks distributed over storage nodes
 - Suffer from block allocation issues, lock managers
 - Example: GPFS

- Object-based Parallel File Systems
 - *Variable-length* regions of the file
 - A file has a constant number of objects
 - Objects are given global identifiers (object-ids, handles, etc.)
 - File growth increases the size of object(s)
 - Objects are easier to manage and distribute
 - Space allocation is managed locally on a per-object basis
 - Examples: Lustre, PVFS

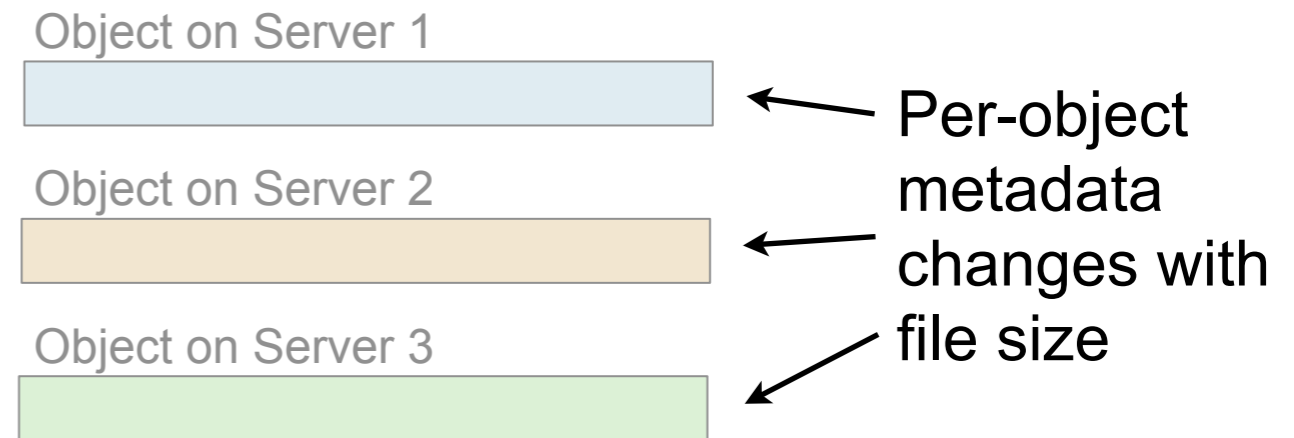
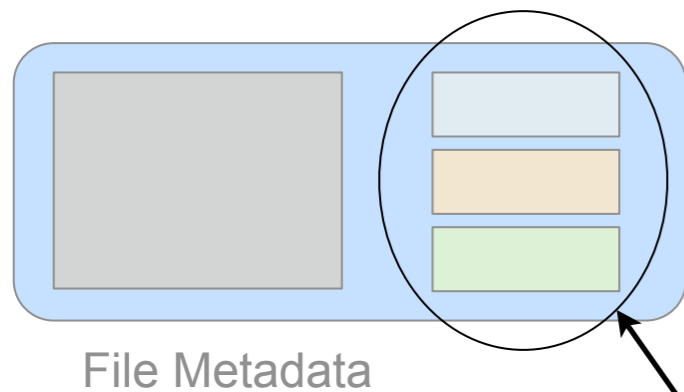
Blocks vs. Objects

- ▶ Metadata for a file includes distribution information
- ▶ Block-based file systems (Shared-disk) require *dynamic* metadata for distribution information



File metadata changes with file size

- ▶ Object-based file systems only need *static* metadata for distribution information



File metadata fixed at file creation

What is POSIX I/O?

- A set of interfaces defined in 1970s:
 - `fd = open(filename, mode);`
 - `read(fd, buffer, size);`
 - `write(fd, buffer, size);`
- Specification also defines rules for maintaining consistency
 - Two processes writing to *overlapping regions* must get consistent results from I/O system
 - Easy on local file systems
 - Distributed/Parallel file systems must manage consistency via locks
 - Other alternatives exist
- Note: NFS uses relaxed close-to-open semantics, not POSIX

Overlapping Regions: Scenario 1

Server 1



Server 2

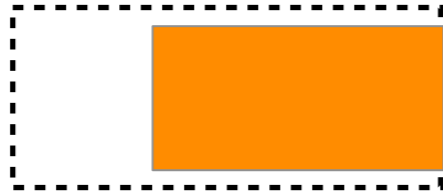
Client A



Client B

Overlapping Regions: Scenario 1

Server 1



Server 2



Client A



Client B



Overlapping Regions: Scenario 1

Server 1



Server 2

Consistent!

Client A



Client B

Overlapping Regions: Scenario 2

Server 1



Server 2

Client A



Client B



Overlapping Regions: Scenario 2

Server 1



Server 2



Client A



Client B



Overlapping Regions: Scenario 2

Server 1



Server 2



Consistent!

Client A



Client B



Overlapping Regions: Scenario 3

Server 1



Server 2



Client A

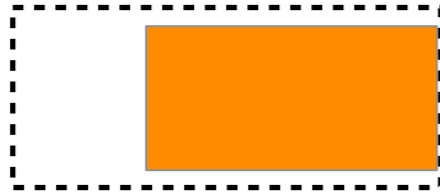


Client B



Overlapping Regions: Scenario 3

Server 1



Server 2



Client A



Client B



Overlapping Regions: Scenario 3

Server 1



Server 2



Inconsistent!

Client A



Client B



Overlapping Regions: Scenario 4

Server 1



Server 2

Client A



Client B



Overlapping Regions: Scenario 4

Server 1



Server 2



Client A



Client B



Overlapping Regions: Scenario 4

Server 1



Server 2



Inconsistent!

Client A

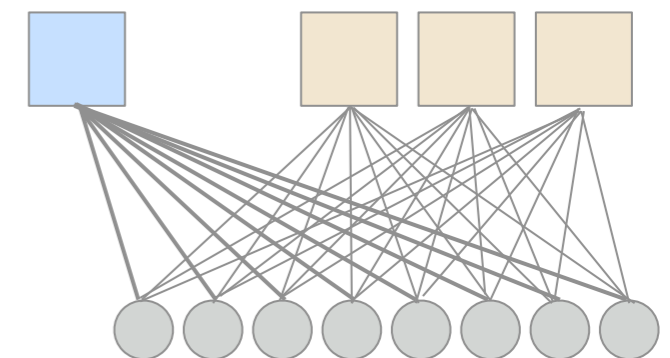
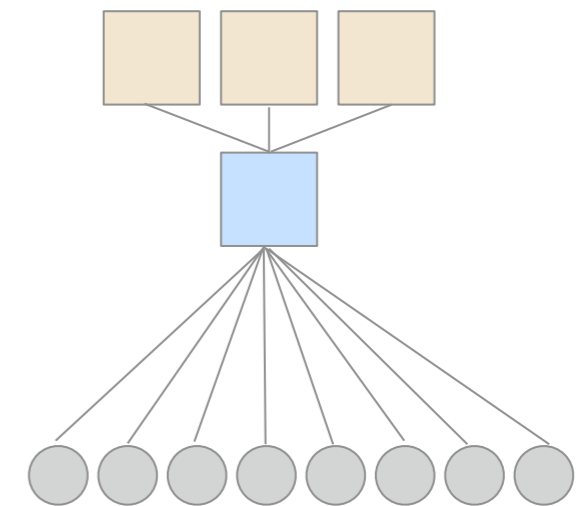


Client B



How do POSIX interfaces/semantics affect Parallel File Systems?

- ▶ Overlapping regions create a major problem
- ▶ How does the Parallel File System provide POSIX consistency semantics?
- ▶ Two Choices:
 - *Centralized Management* - All client requests are made to a broker server, which can serialize the requests to overlapping regions of a file and perform them in isolation where necessary
 - *Distributed Locking* - Clients request a lock from a *lock manager* for the region of data they wish to access. Once a lock has been granted, clients can write exclusively to the region. This requires a *Distributed Lock Manager (DLM)*: a server that hands out locks to clients as they request them.

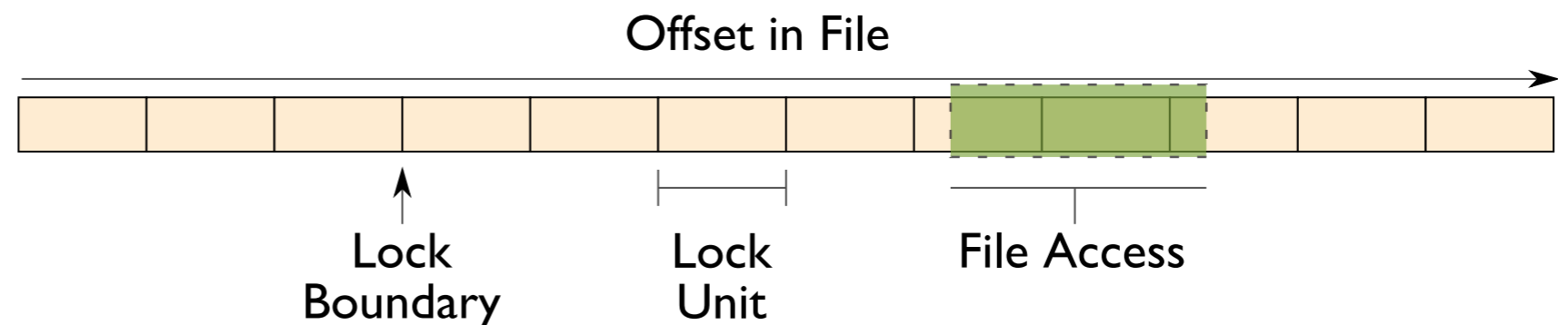


Locking in Parallel File Systems

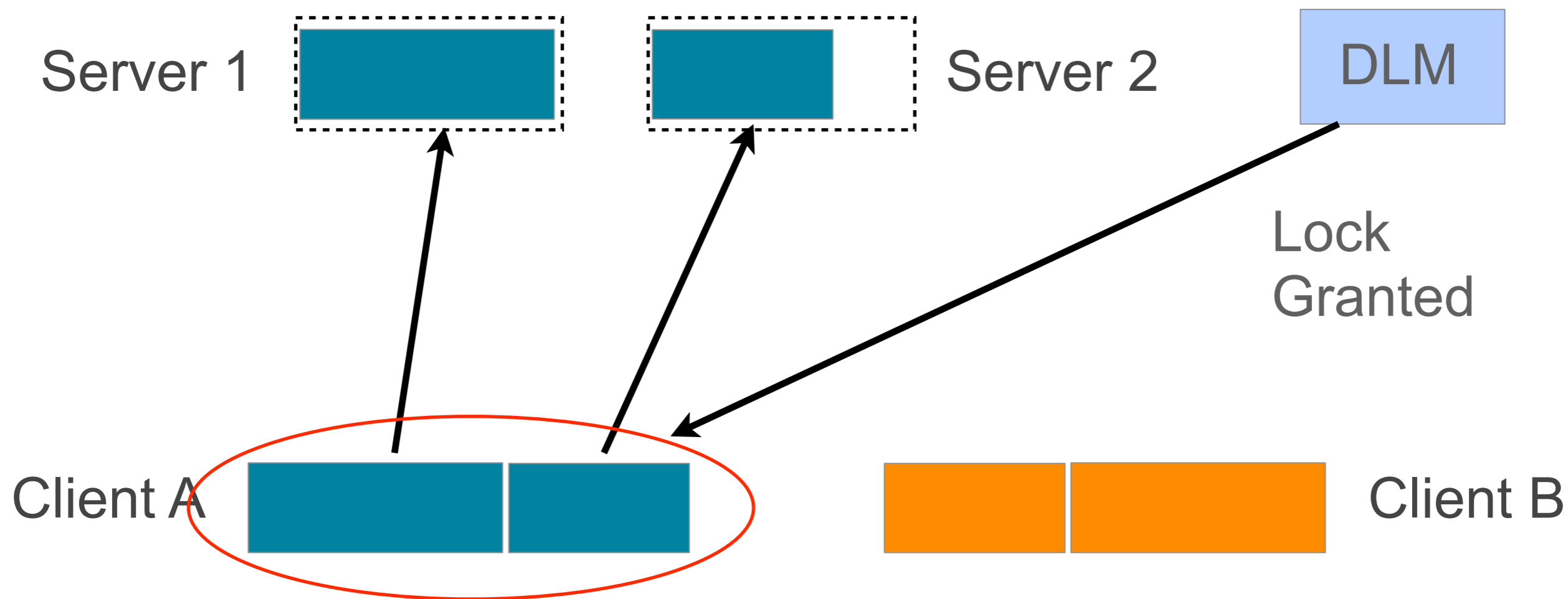
Most parallel file systems use **locks** to manage concurrent access to files

- ▶ Files are broken up into lock units
- ▶ Clients obtain locks on units that they will access before I/O occurs
- ▶ Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
 - Client can optimize small I/O with readahead
- ▶ Locks are reclaimed from clients when others desire access
- ▶ Locks are delegated and revoked through distributed lock managers

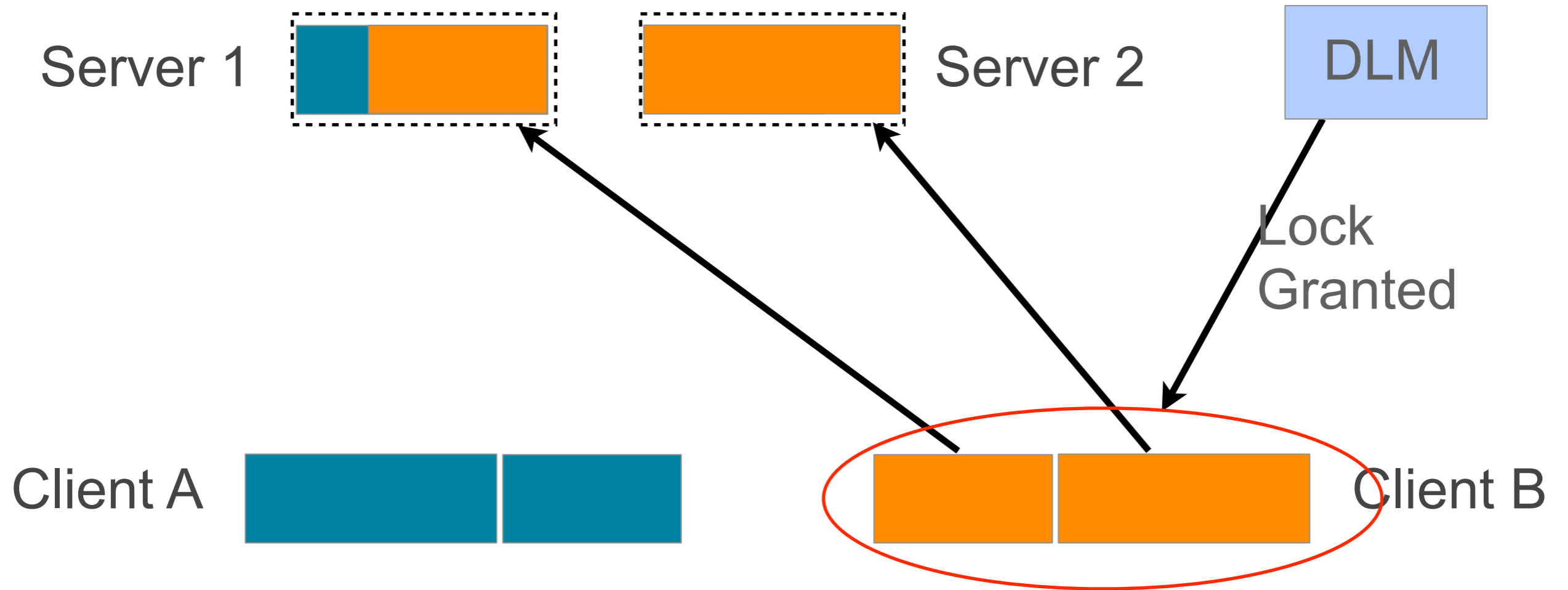
If an access touches any data in a lock unit, the lock for that region must be obtained before access occurs.



Locks and the DLM



Locks and the DLM



Distributed Lock Managers

- ▶ Implementation burden
 - DLMs add complexity to file system
 - What if the DLM node fails?
- ▶ Locks are expensive!
 - Round-trip latencies between clients and DLM
 - What happens on client failure?
- ▶ Idea: Lets just not write to overlapping regions!
 - Most applications don't write to overlapping regions concurrently anyway



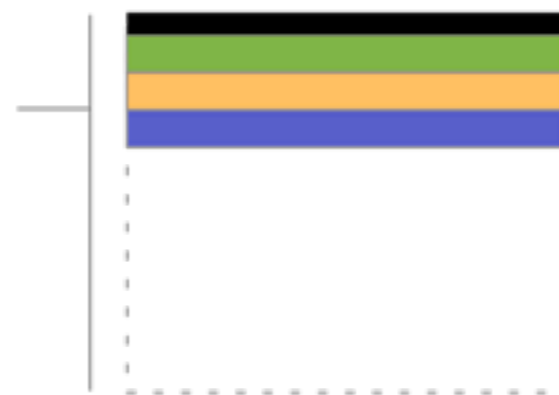
Locking and Concurrent Access

2D View of Data

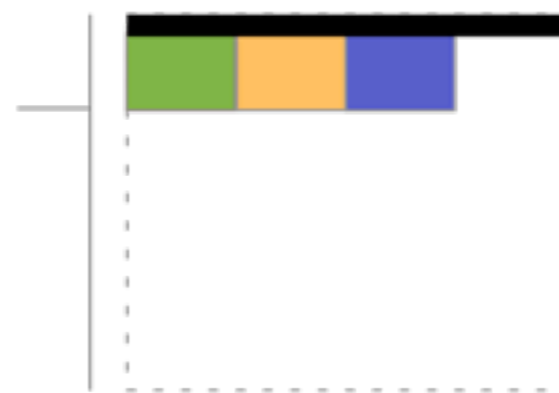


The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.

In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.



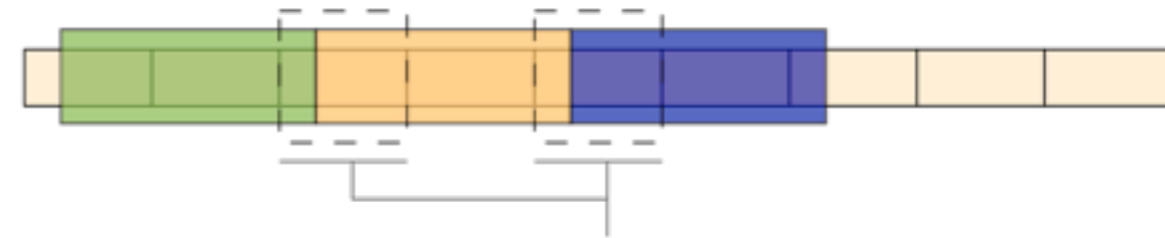
In this example, processes exhibit a block-block access pattern (e.g. accessing a subarray). This results in many interleaved accesses in the file.



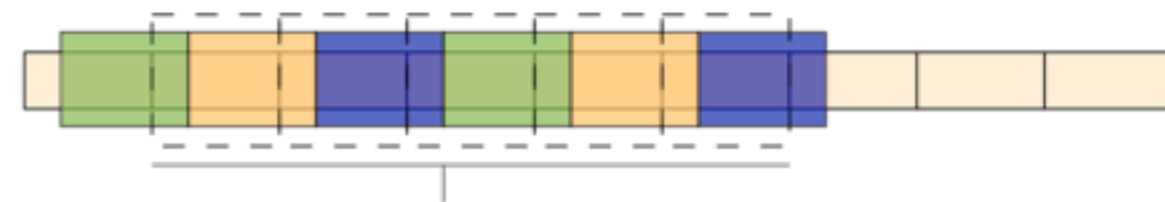
Offset in File



When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.



These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.



When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

What does PVFS do?

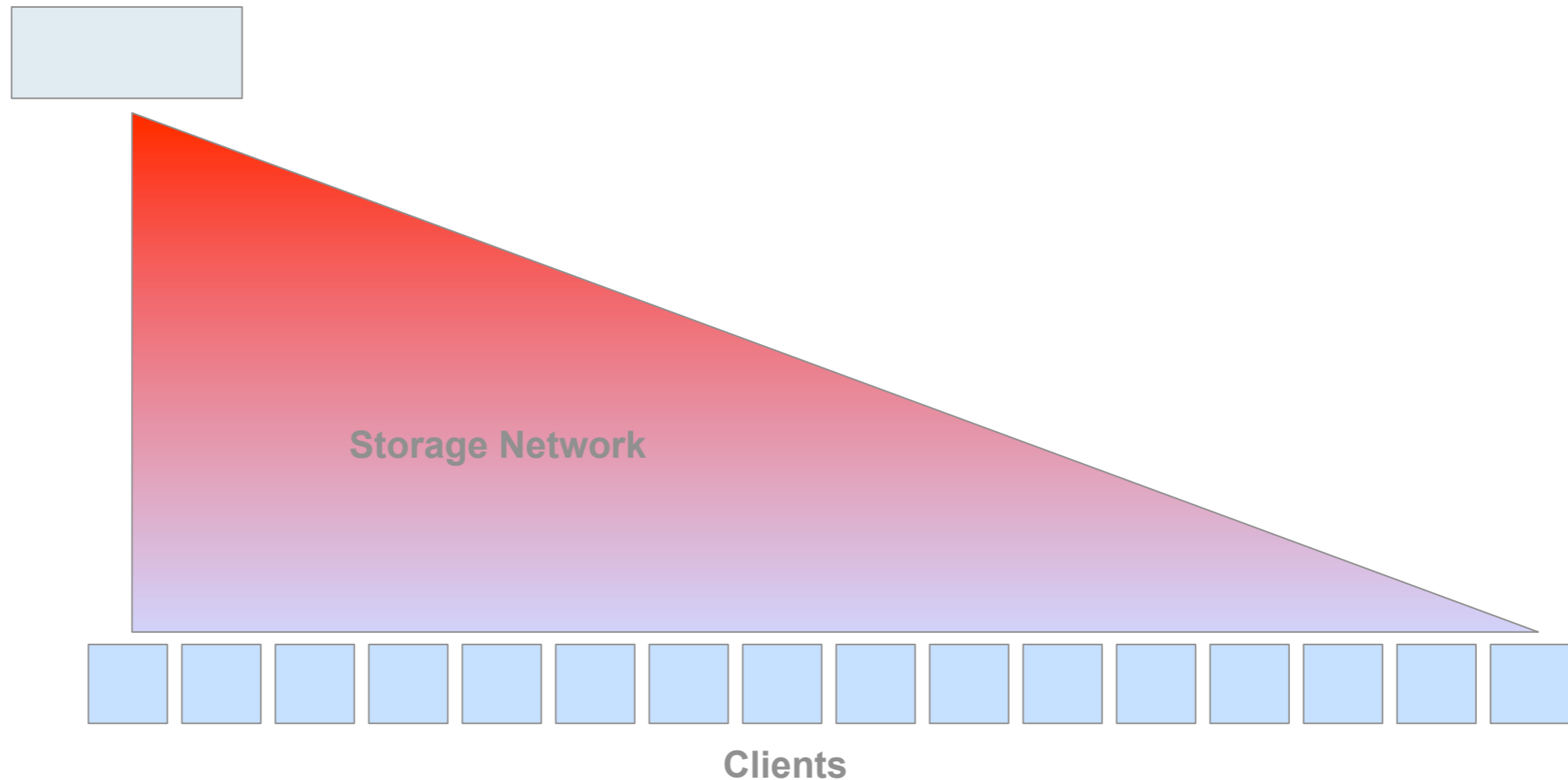
- ▶ Simply doesn't provide POSIX consistency guarantees
- ▶ If two clients write to overlapping regions concurrently, **undefined results!**
- ▶ **Oh no! Not POSIX!**
- ▶ Which applications perform this pattern of access (overwrite overlapping regions)?
 - Applications using the file system to update shared state, e.g. current value is 42
 - Event notification
 - Changes to a file: new configuration, editing, etc.
- ▶ Computational Science doesn't often access overlapping regions
 - Better to use MPI to communicate shared state in a distributed fashion
 - Event notification shouldn't be implemented in the file system
 - Editing files is done interactively, not by clients accessing regions concurrently
- ▶ What about appending records to a file?
 - Don't care about offset, just want to append
 - Requires (*atomically*) updated file size information
 - Shared file pointers?
 - Google FS uses *record append*



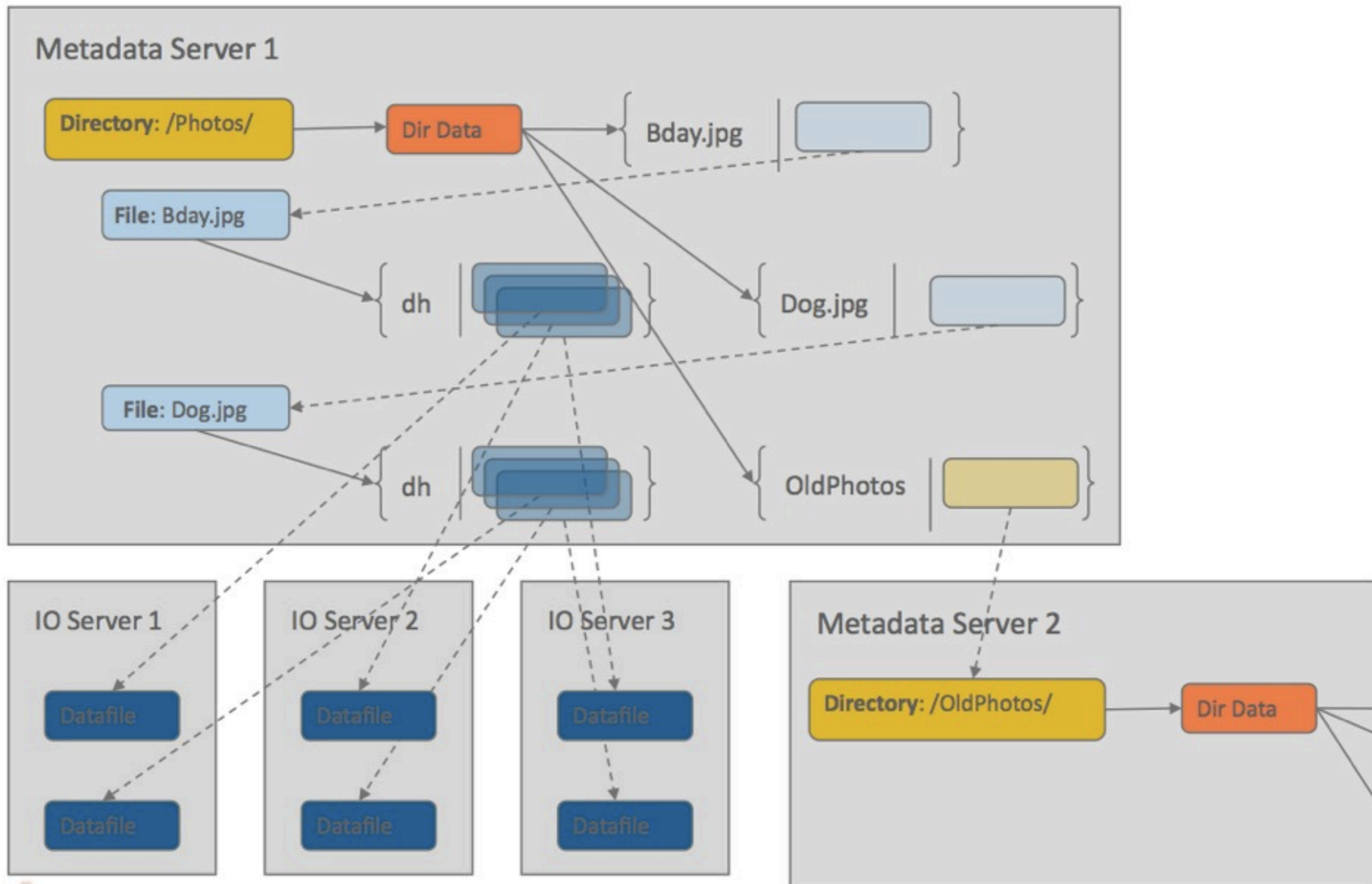
Metadata in Parallel File Systems

- ▶ A single metadata server creates a single point of contention (hotspot)
 - Many clients try to open the same file at the same time: Creates an N-to-1 pattern of lookup requests
 - Many clients try to create new files at once: Creates an N-to-1 pattern of create requests (requires disk access too!)
- ▶ How can metadata be distributed across metadata servers?
 - Depends on underlying design (blocks vs. objects)

Single Metadata Server

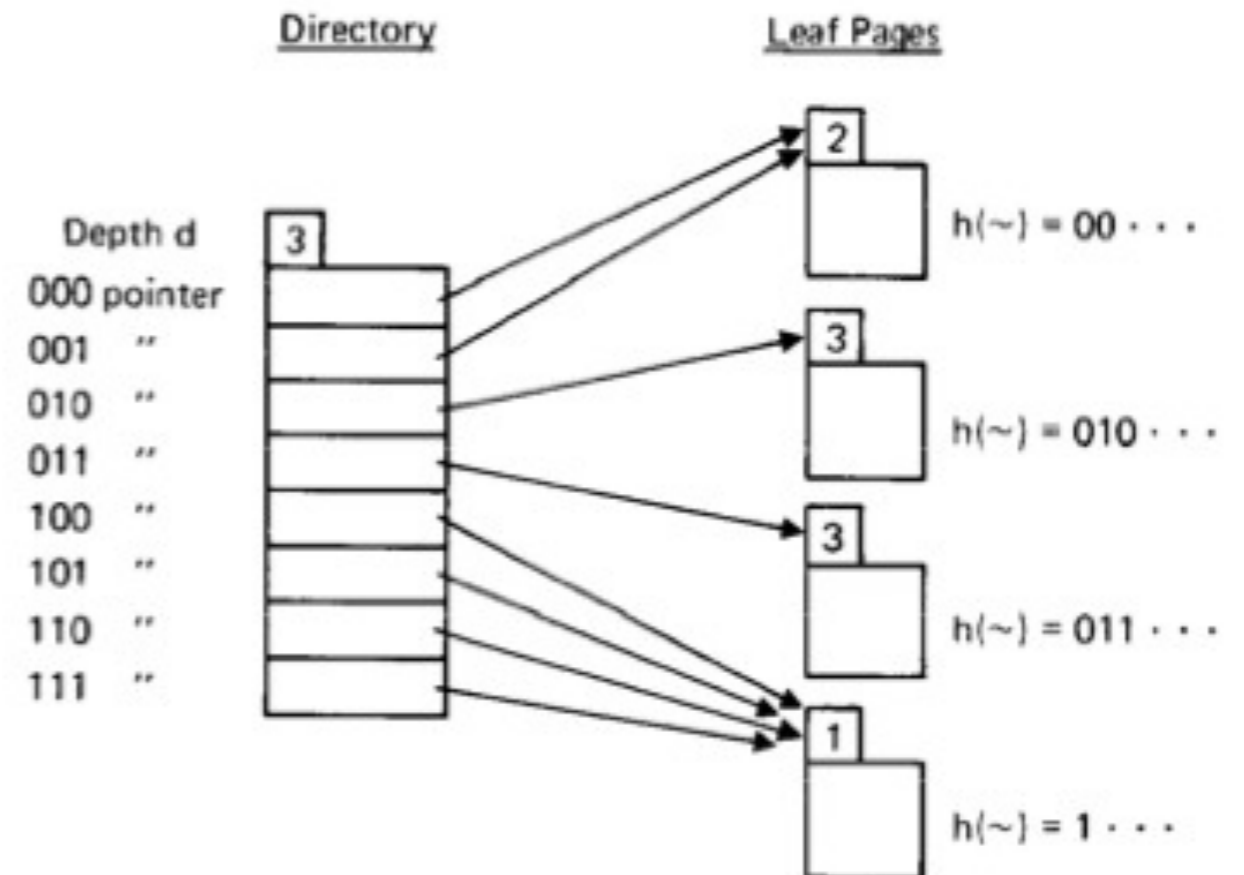


Distributing Directories in PVFS



Distributing Metadata in GPFS

- ▶ PVFS distributes metadata by placing directories on different metadata servers
- ▶ What about a single directory with millions (or billions!) of entries?
- ▶ GPFS stores directory entries in essentially a *sparse file*
- ▶ The sparse file (directory entries) is distributed across blocks on different storage servers
- ▶ GPFS determines block accesses using



Parallel File System Comparisons

	PVFS	GPFS	Lustre	Ceph	Ursa Minor
Production Ready	yes	yes	yes	no	no
Noncontiguous I/O	yes	no	yes*	no	no
Stateless Clients	yes	no	no	no	no
High Performance Shared File Writes	yes	yes	no*	no	no
Fully Decentralized Metadata	yes	yes	no	no	no
Software Redundancy	no	yes*	no	yes	yes
Conserves Client BW when Replicating	no	no	no	yes	no
App.-Level Object Abstraction	no	no	no	yes	yes
Scalable Failure Detection	no	no	no	no	no
Reads from Multiple Replicas	no	no*	no	no	no*

Thanks!