Parallel File Systems

Anthony Kougkas
akougkas@iit.edu

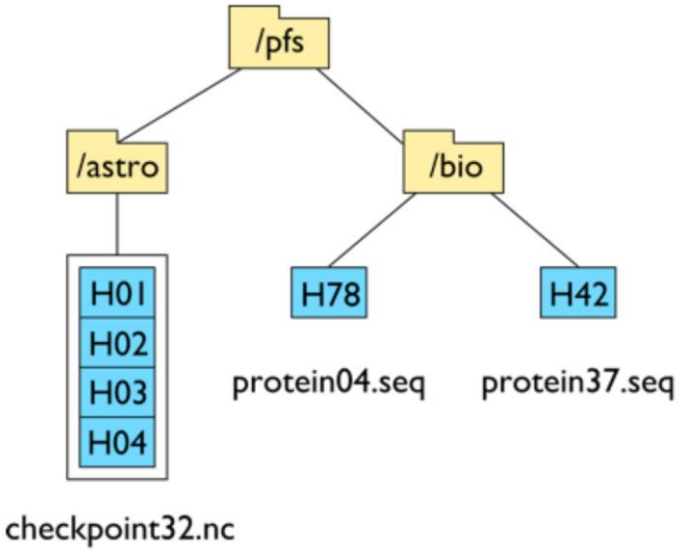
ILLINOIS INSTITUTE OF TECHNOLOGY

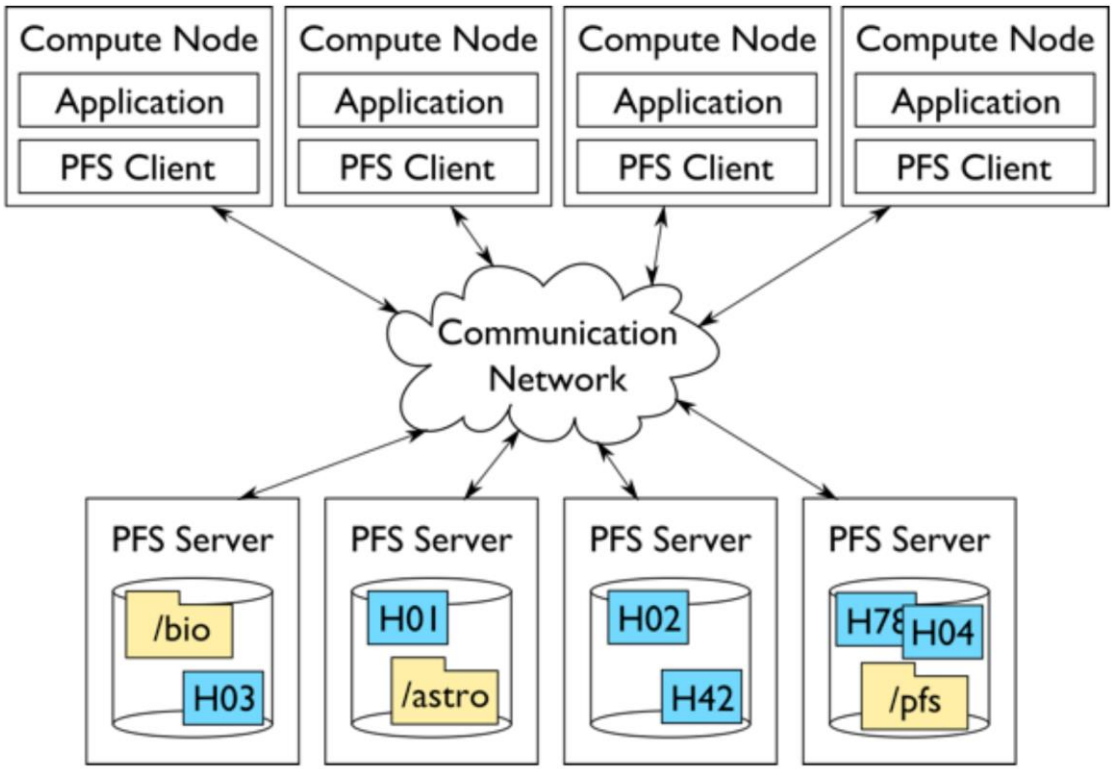
Parallel File Systems

- 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

10/28/2019

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)

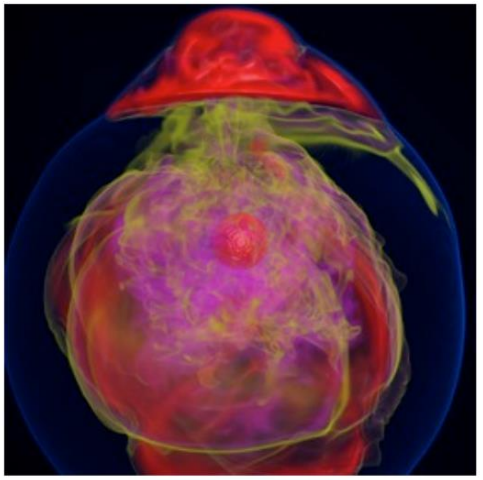
Outline

- What are Parallel File Systems?
- Who uses Parallel File Systems?
- Where are Parallel File Systems deployed?
- How are Parallel File Systems designed?

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

PI	Project	On-Line Data	Off-Line Data
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

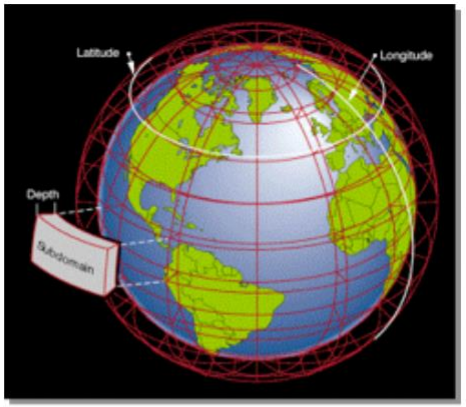
[1] S. Klasky, personal correspondence, June 19, 2008.

Selected Applications

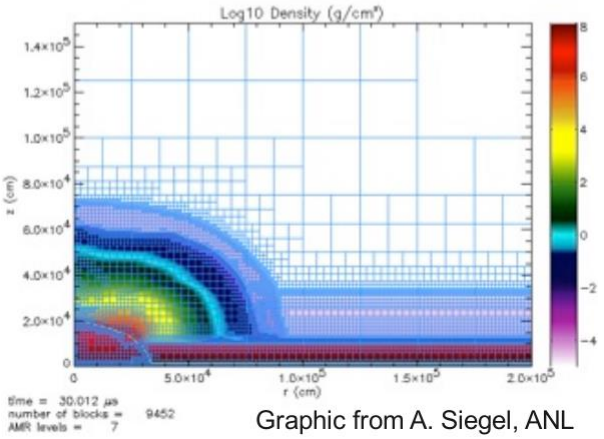
	Nuclear reactor simulation	Climate modeling	Astrophysics
Preliminary studies			
# of elements	4.5 million	3 million	70-500 million grid points
# of variables	7	100 (30 are vectors)	4 (1 is a vector)
# of timesteps	20,000	200,000-400,000	1,800
Total data size	2.5 Tbytes	30-120 Tbytes	80 Tbytes
Science runs			
# 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)
- High-level I/O libraries help map between these data models



Graphic from J. Tannahill, LLNL



Graphic from A. Siegel, ANL

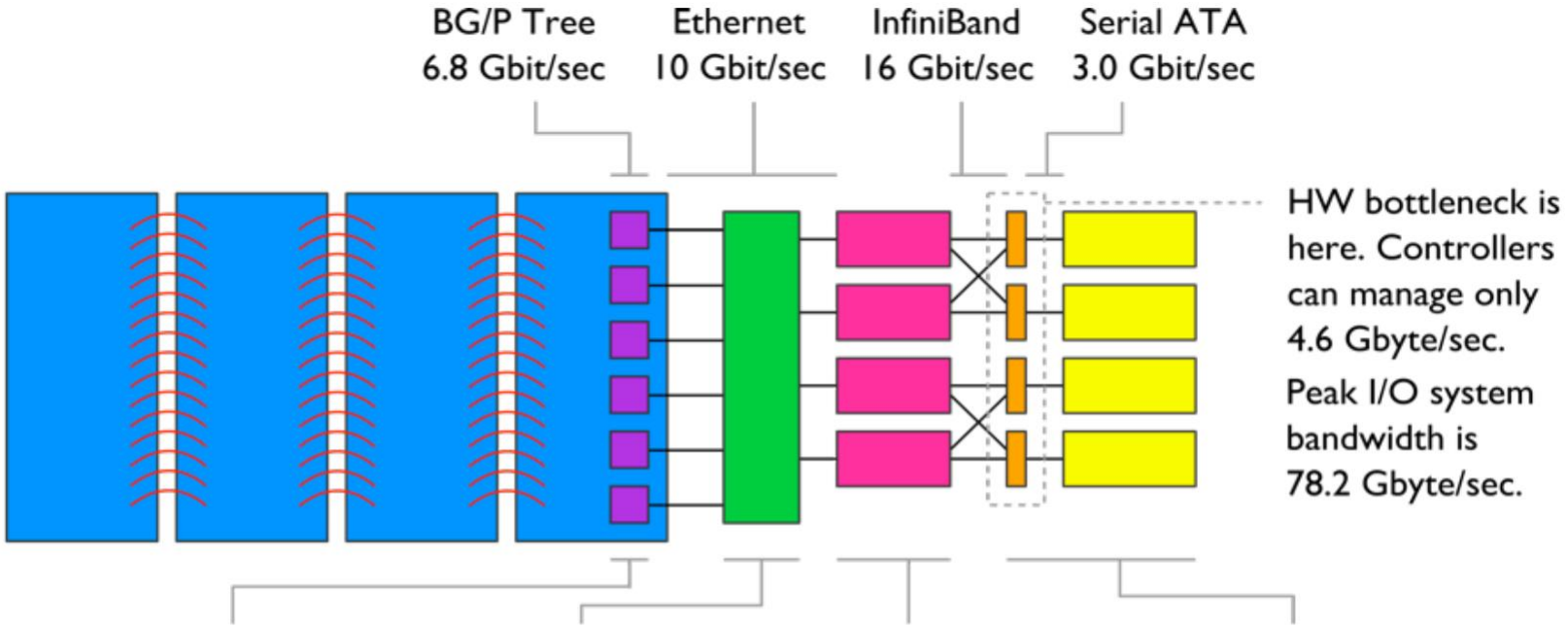
Shared-file vs. File-per-process

- Scientific applications perform I/O to PFS 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 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

Outline

- What are Parallel File Systems?
- Who uses Parallel File Systems?
- Where are Parallel File Systems deployed?
- How are Parallel File Systems designed?

Intrepid Parallel Storage System



Gateway nodes run parallel file system client software and forward I/O operations from HPC clients.

640 Quad core PowerPC 450 nodes with 2 Gbytes of RAM each

Commodity network primarily carries storage traffic.

900+ port 10 Gigabit Ethernet Myricom switch complex

Storage nodes run parallel file system software and manage incoming FS traffic from gateway nodes.

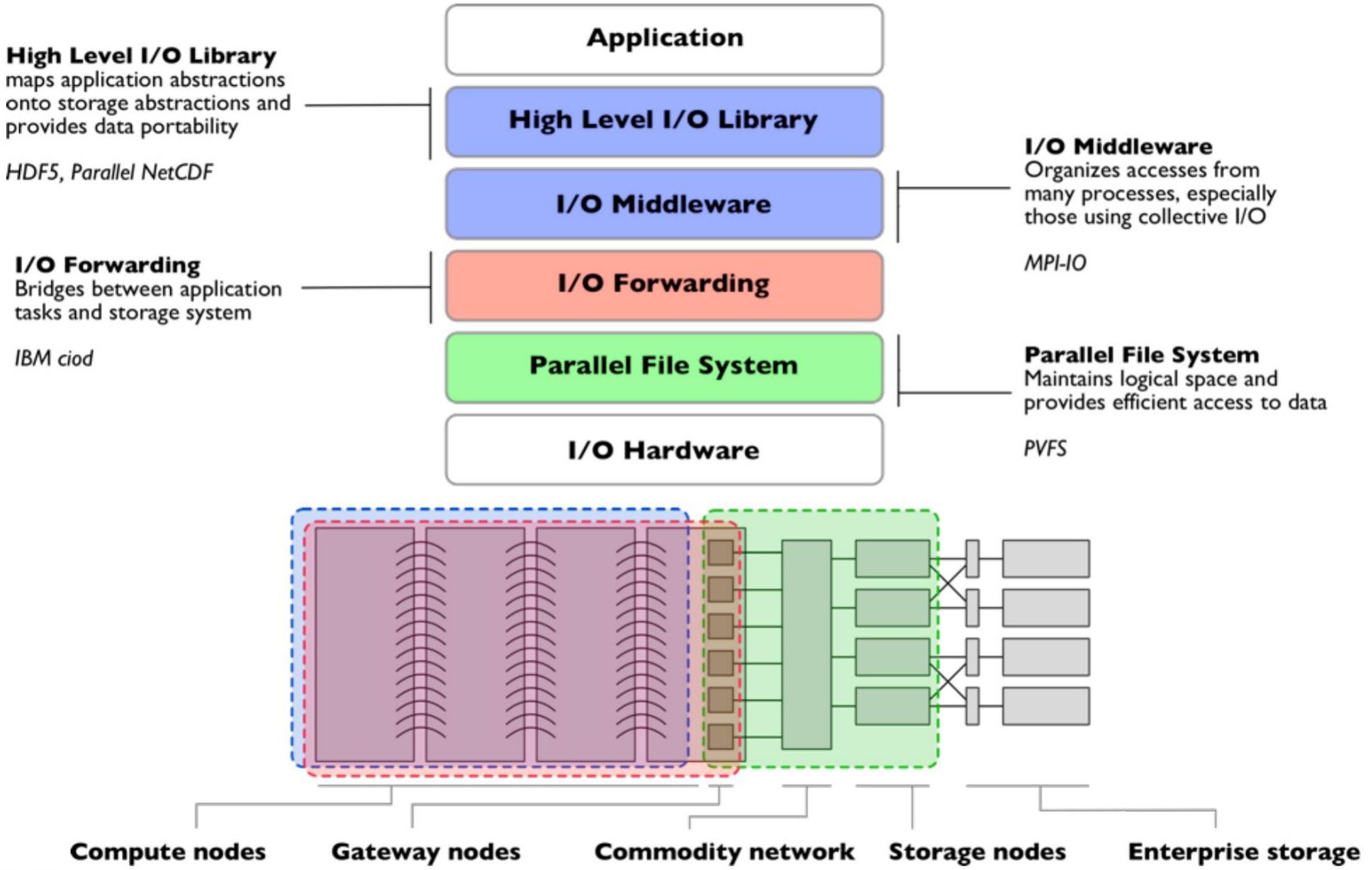
136 two dual core Opteron servers with 8 Gbytes of RAM each

Enterprise storage controllers and large racks of disks are connected via InfiniBand or Fibre Channel.

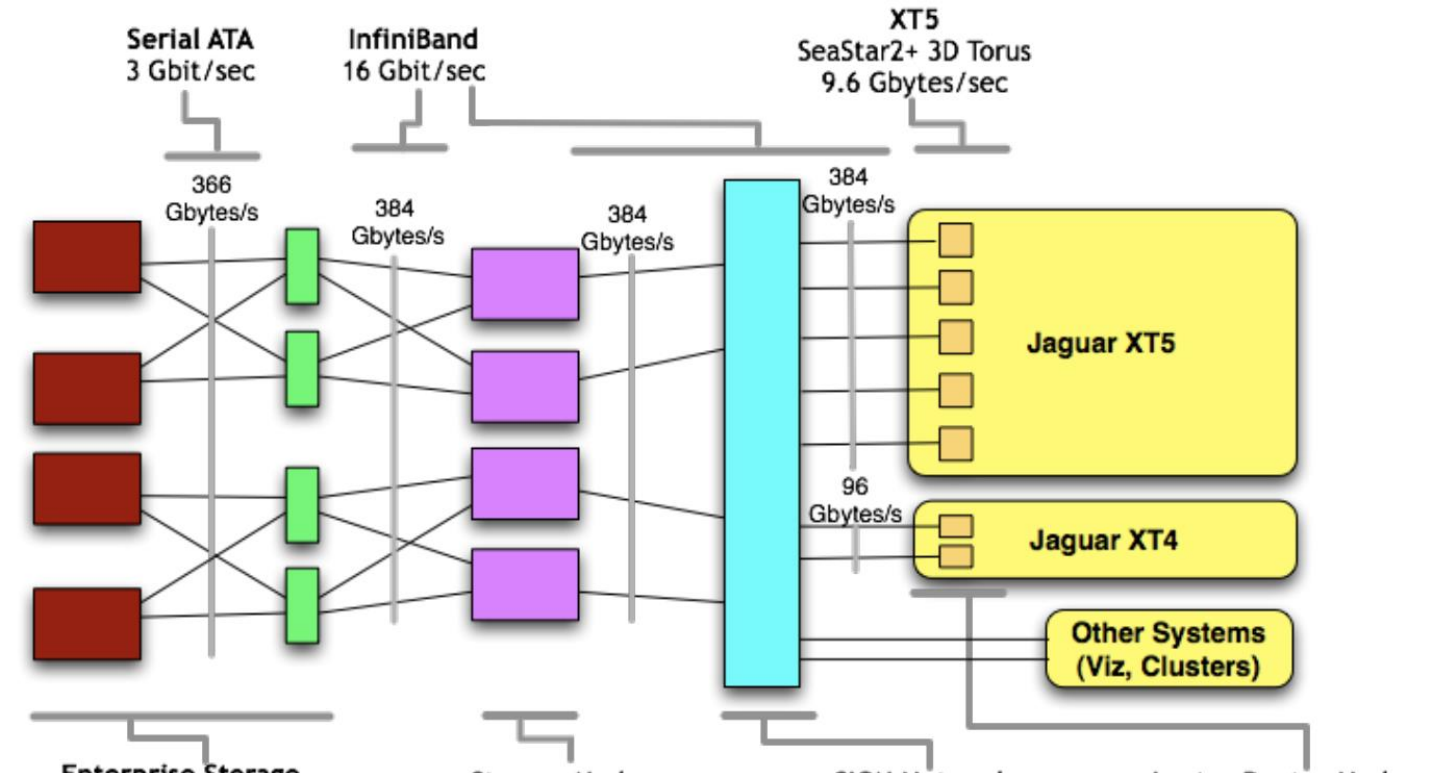
17 DataDirect S2A9900 controller pairs with 480 1 Tbyte drives and 8 InfiniBand ports per pair

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

I/O Software Stack on Intrepid



Jaguar Storage System



Enterprise Storage
controllers and large racks of disks are connected via InfiniBand.

48 DataDirect S2A9900 controller pairs with 1 Tbyte drives and 4 InifiniBand connections per pair

Storage Nodes
run parallel file system software and manage incoming FS traffic.

192 dual quad core Xeon servers with 16 Gbytes of RAM each

SION Network
provides connectivity between OLCF resources and primarily carries storage traffic.

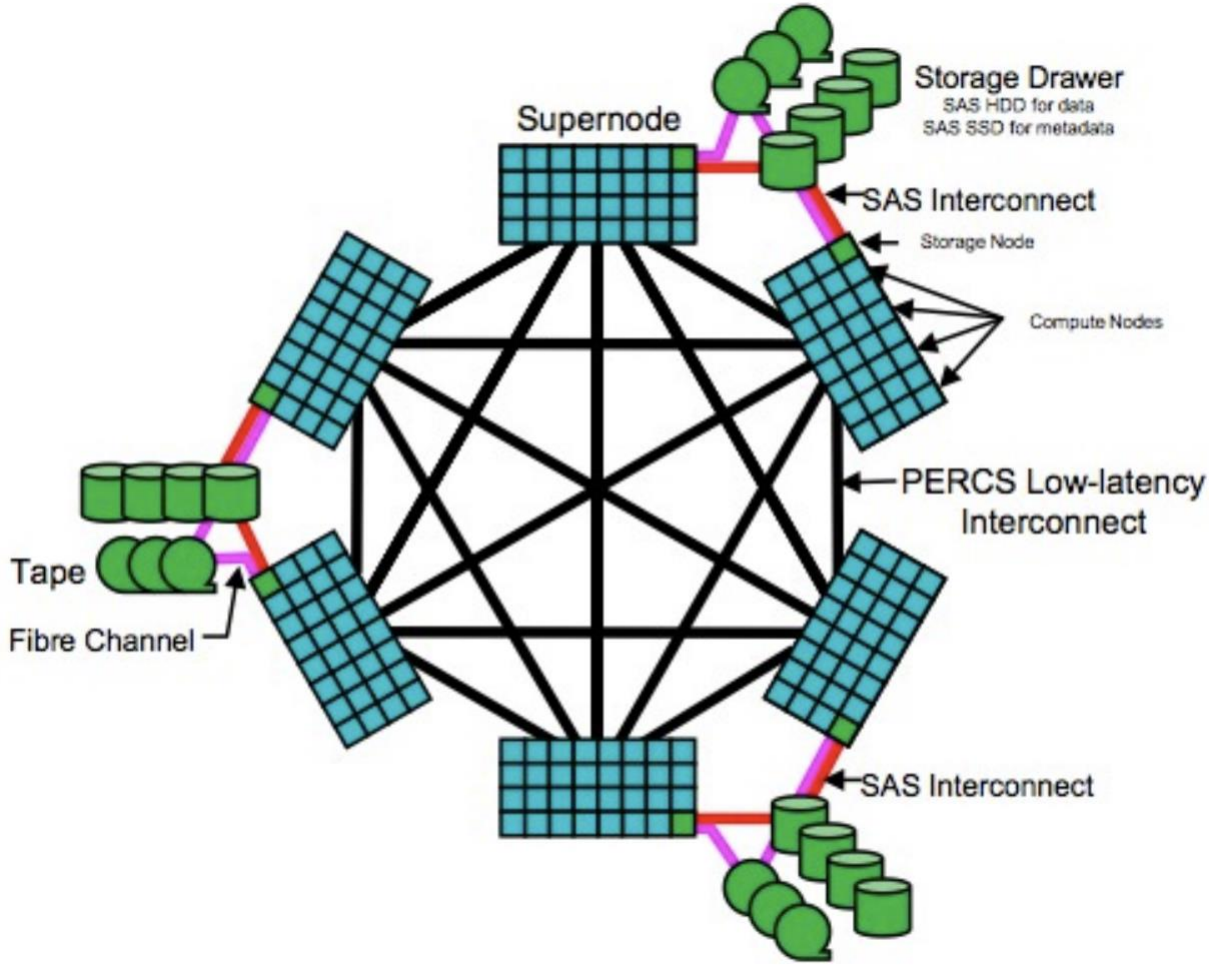
3000+ port 16 Gbit/sec InfiniBand switch complex

Lustre Router Nodes
run parallel file system client software and forward I/O operations from HPC clients.

192 (XT5) and 48 (XT4) one dual core Opteron nodes with 8 GB of RAM each

Bluewaters Storage System

- File System (GPFS) runs directly on compute nodes
- Storage nodes and physical storage embedded in compute racks
- 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



Outline

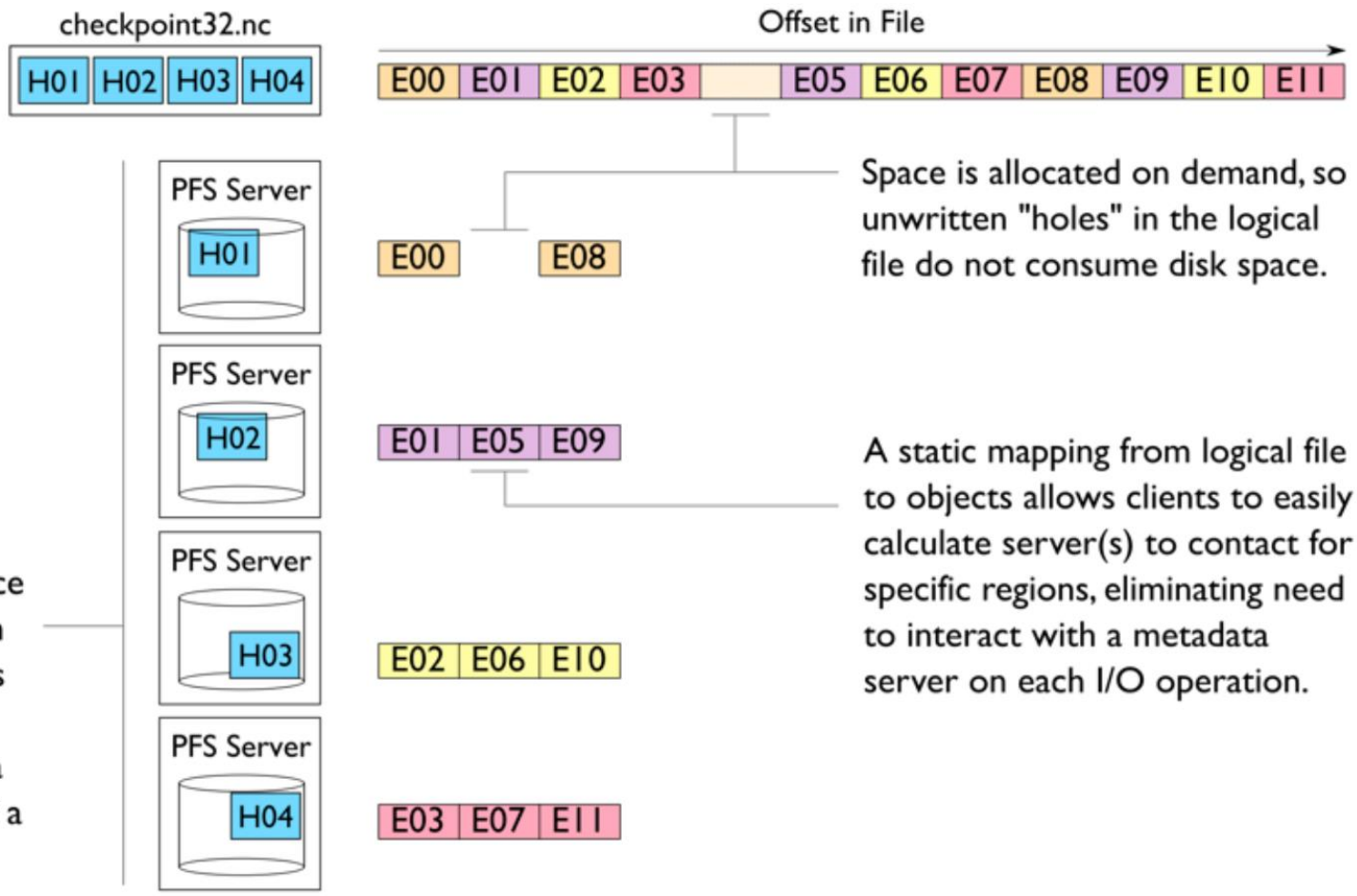
- What are Parallel File Systems?
- Who uses Parallel File Systems?
- Where are Parallel File Systems deployed?
- How are Parallel File Systems designed?

Data distribution in PFS

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.



Data Distribution

- Round-round is a reasonable default solution
  - Works consistently for a variety of workloads
  - Works well on most systems

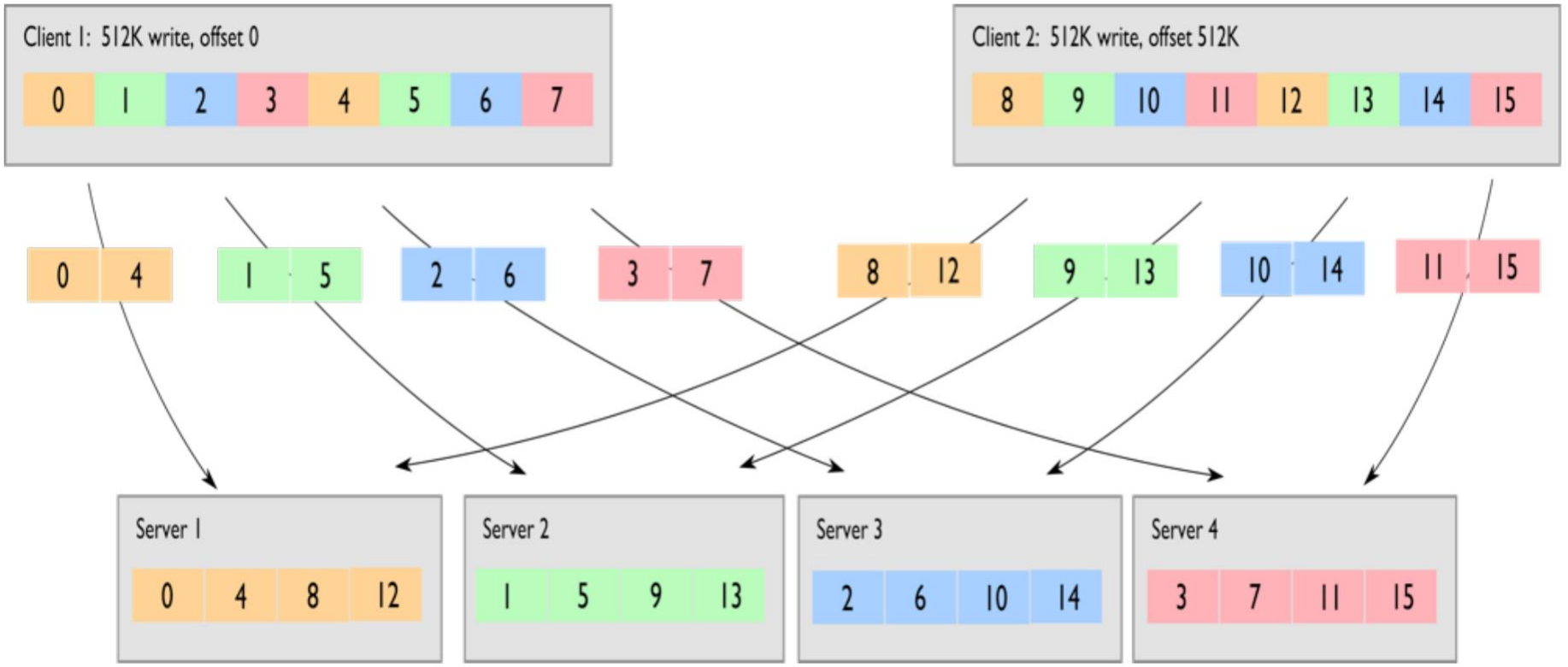
Data requirements for select 2008 INCITE applications at ALCF

PI	Project	On-Line Data	Off-Line Data
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.

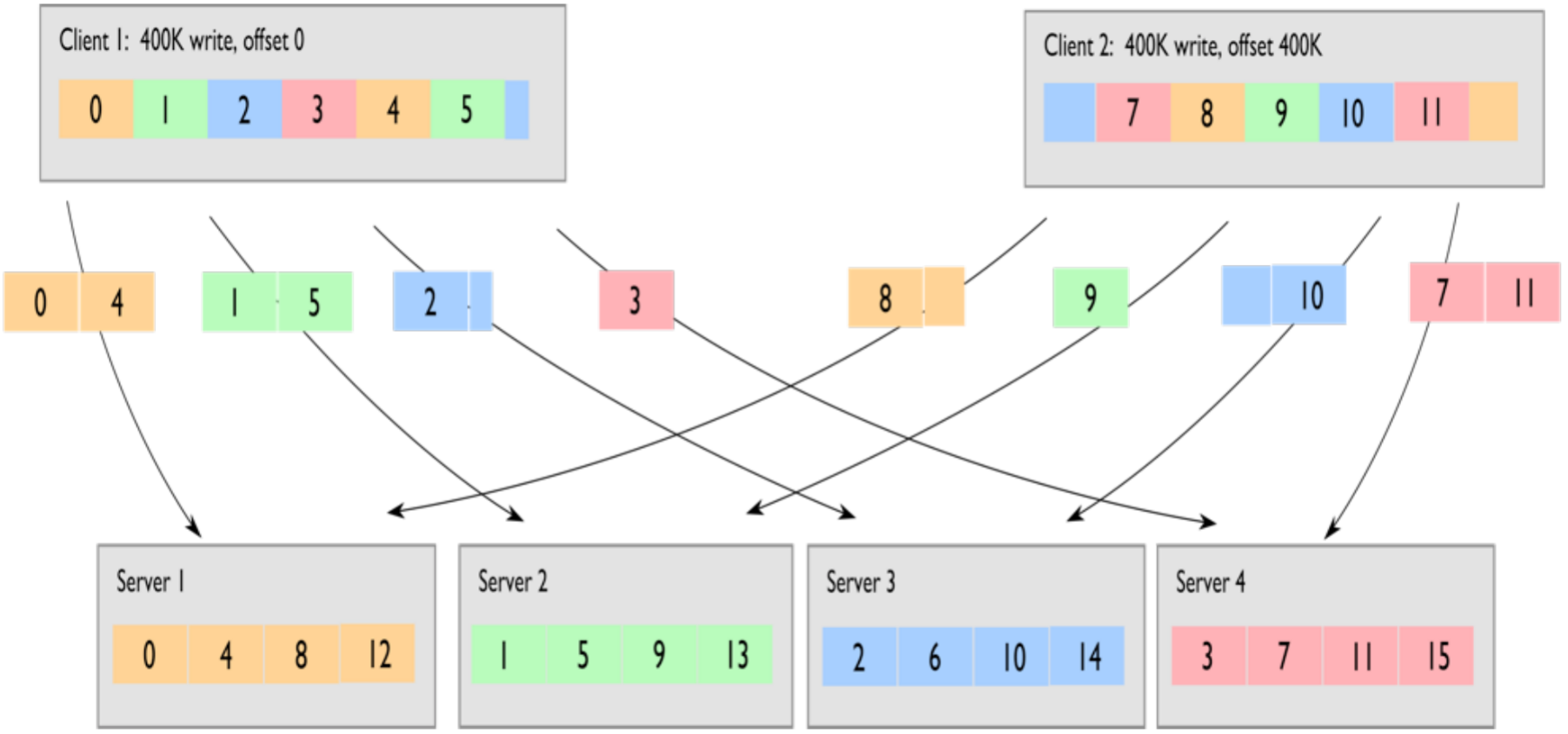
Data Distribution

Clients perform writes/reads of file at various regions



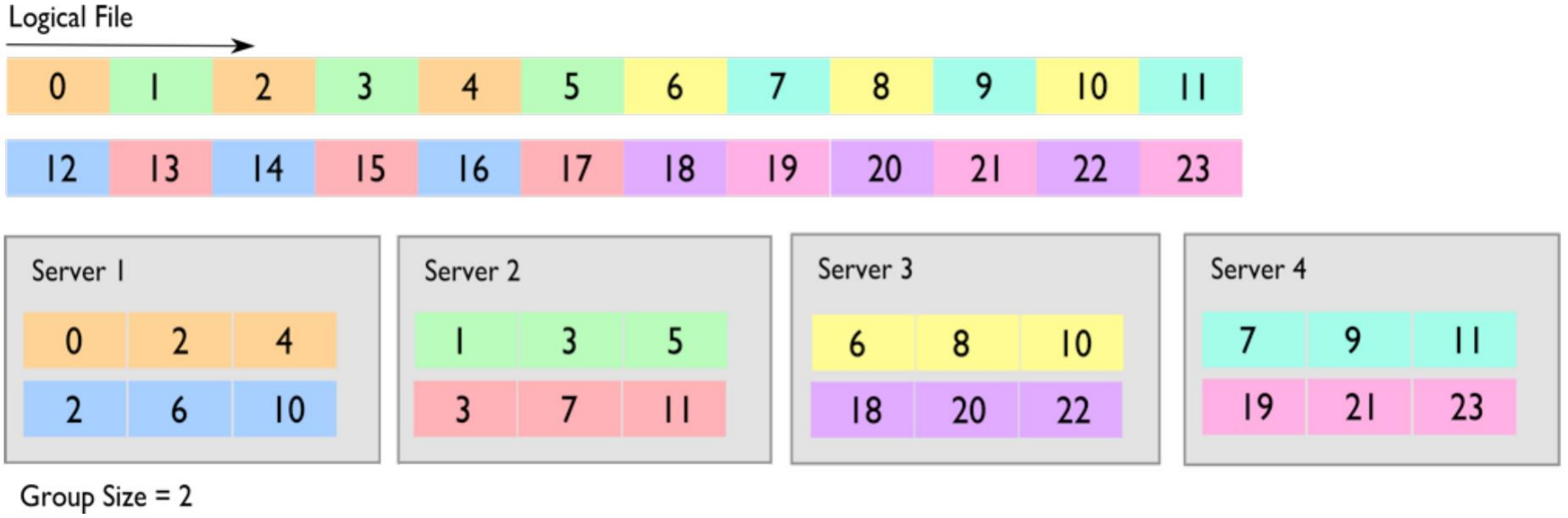
Data Distribution

- Sizes of requests, alignment to striping unit is important



Data Distribution

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

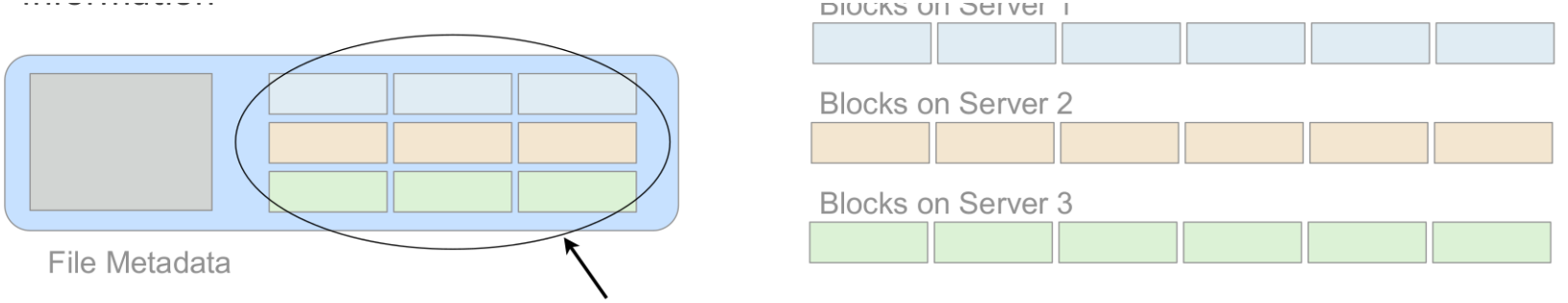


Classes of Parallel File Systems: Blocks vs. Objects

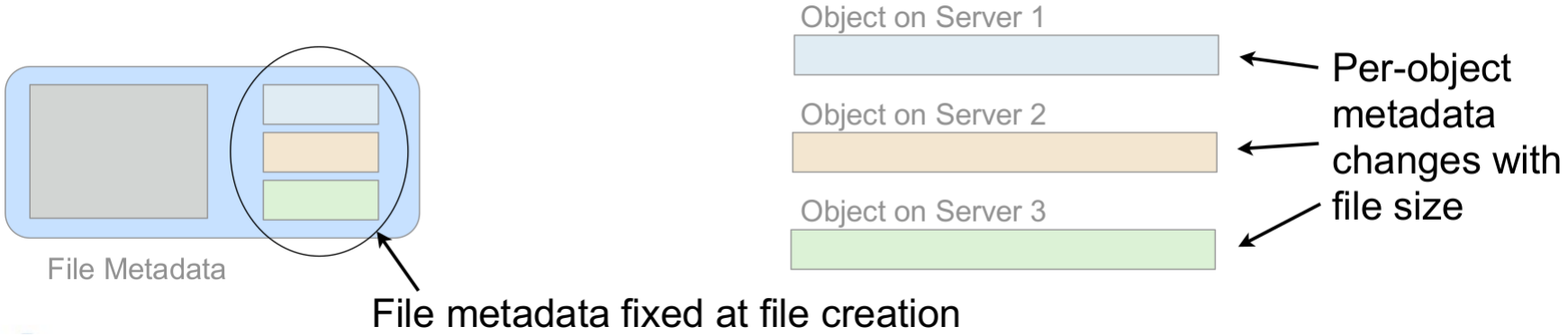
- Block-Based Parallel File Systems (i.e., “Shared--disk”)
  - Blocks are fixed-width
  - File growth requires more blocks
  - Blocks distributed over storage nodes
  - Suffer from block allocation issues, lock managers
  - Example: GPFS
- Object-based Parallel File Systems
  - Variable-length regions 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



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

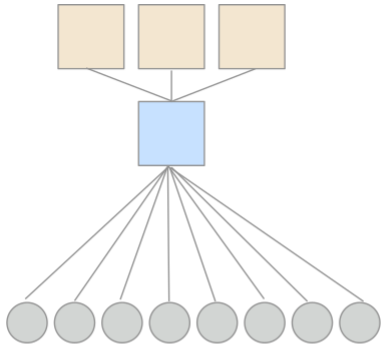


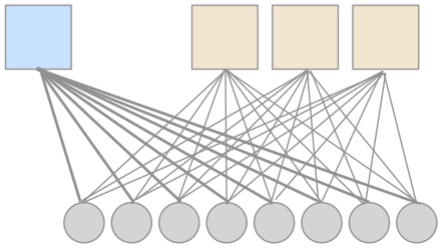
What is POSIX I/O?

- 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 (eventual consistency)
- Note: NFS uses relaxed close-to-open semantics, not POSIX

How do POSIX interfaces/semantics affect Parallel File Systems?

- Overlapping regions create a major problem
- How does the PFS 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.

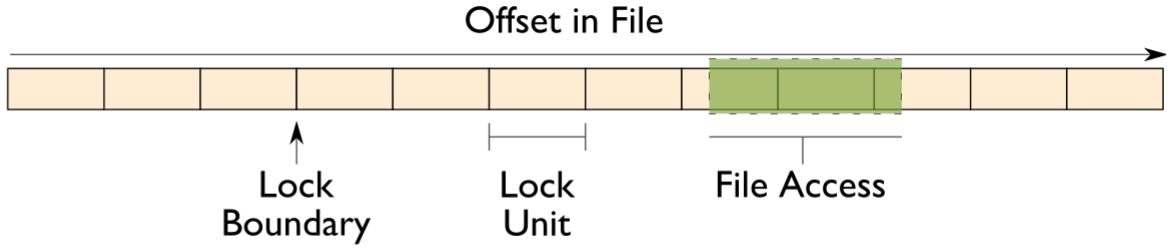




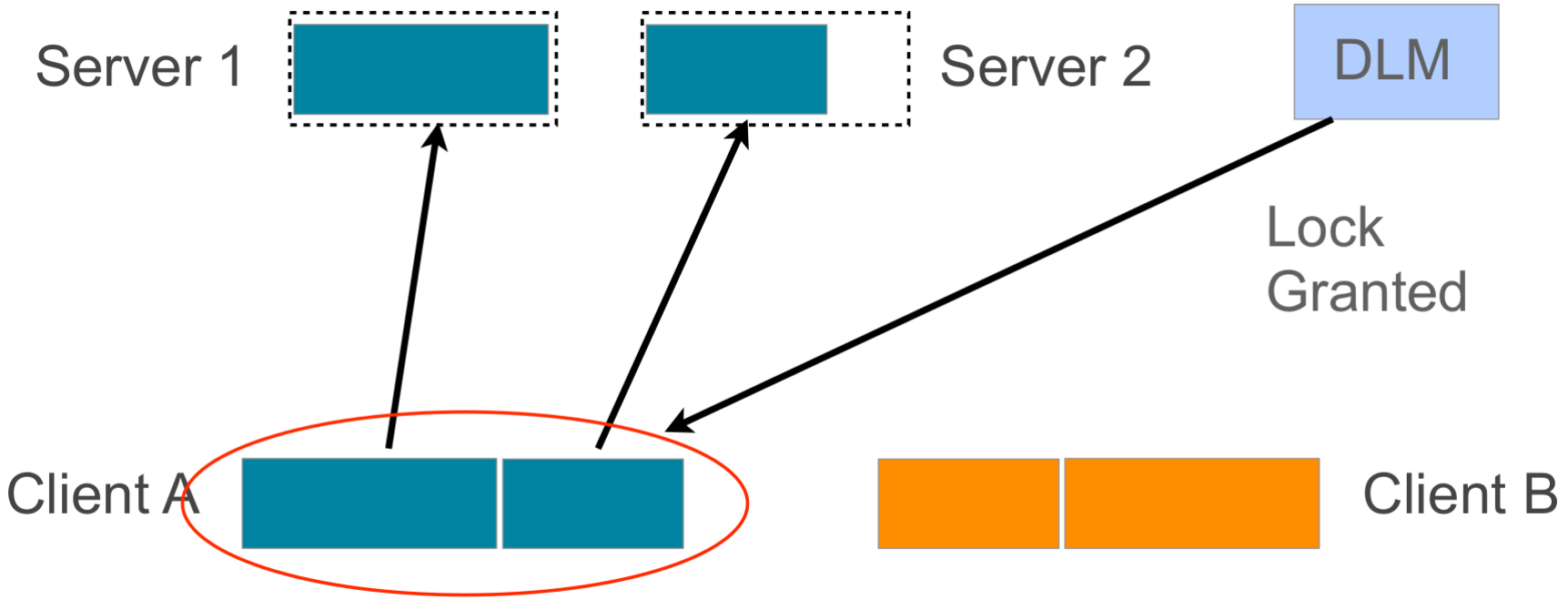
Locking in Parallel File Systems

- Most PFS 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

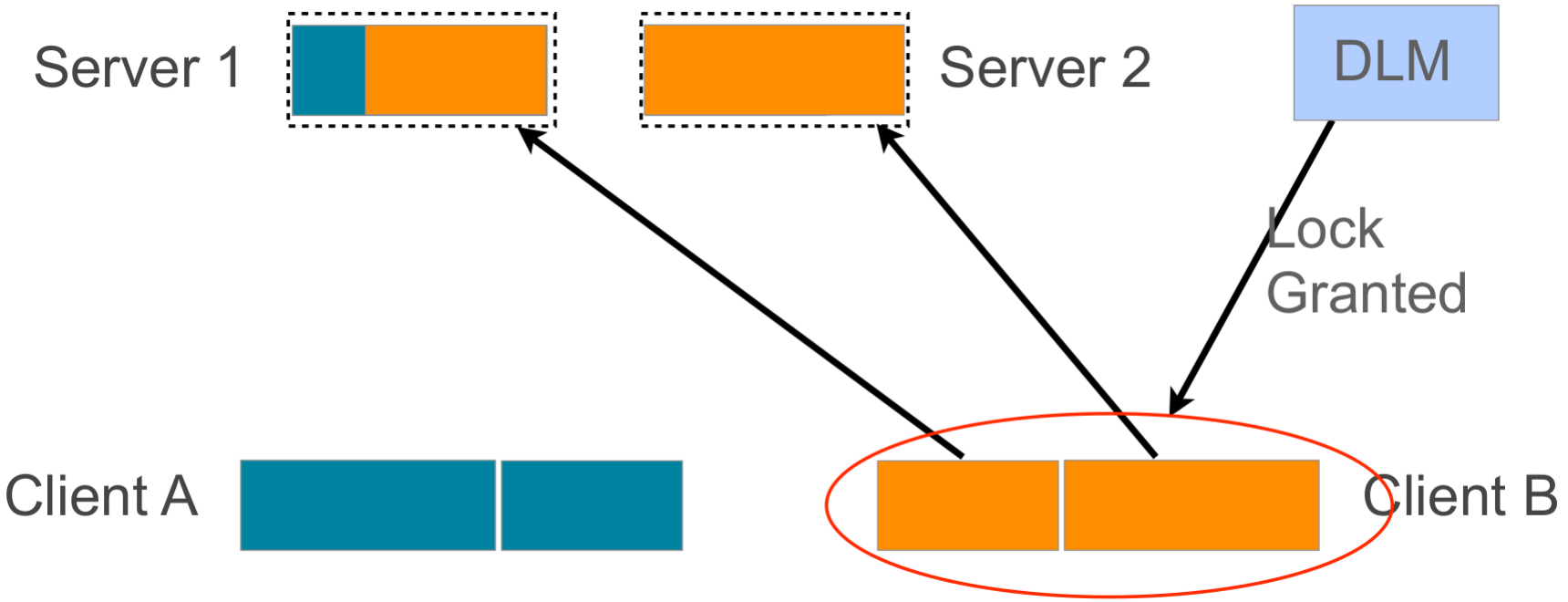
If an access touches any data in a lock unit, the lock for that region must be obtained before access occurs.



Locks and the DLM



Locks and the DLM

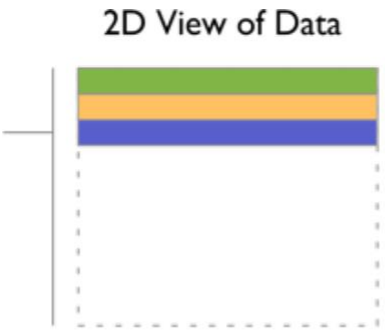


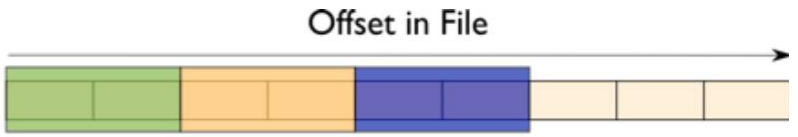
Distributed Lock Managers

- 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

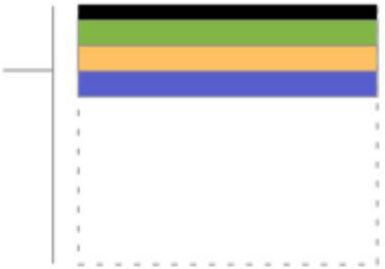
The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.

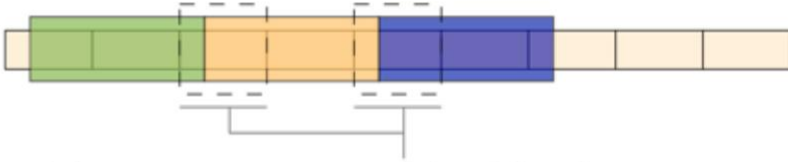




When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.

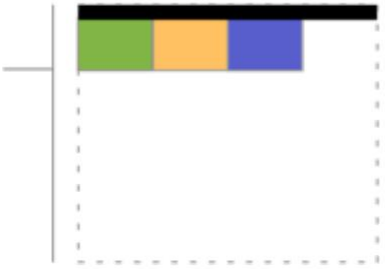
In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.

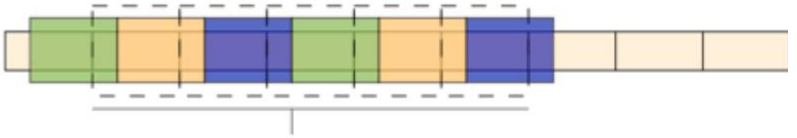




These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

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





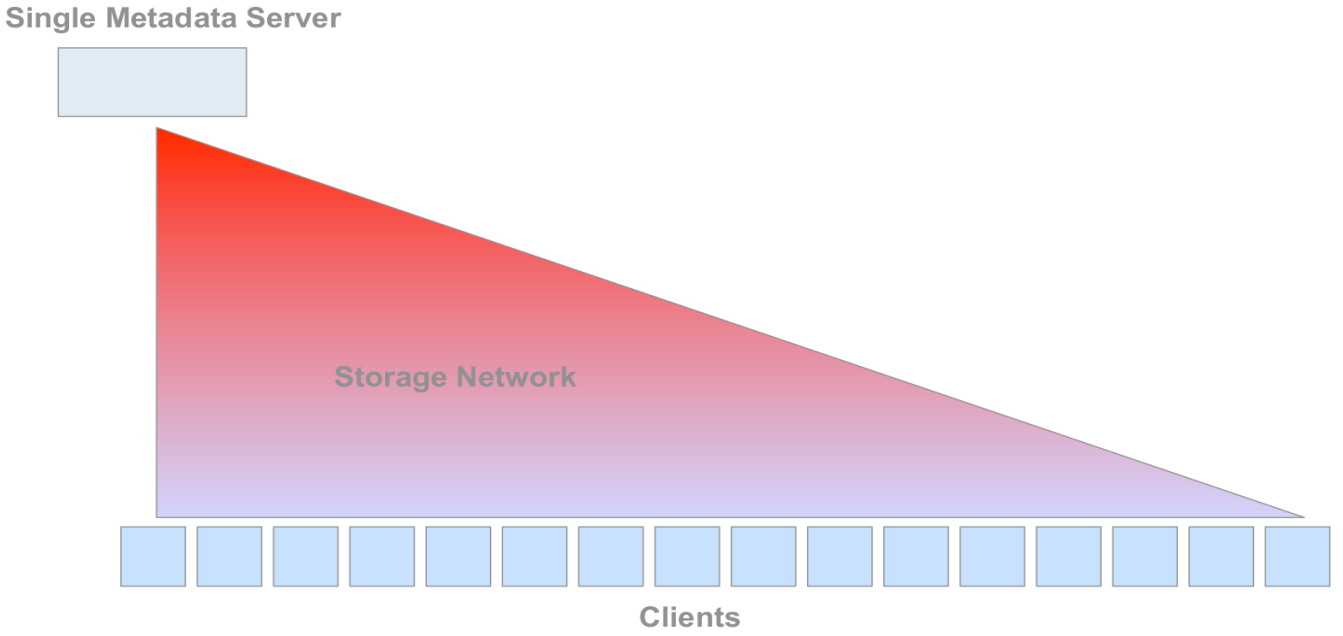
When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

What does PVFS do?

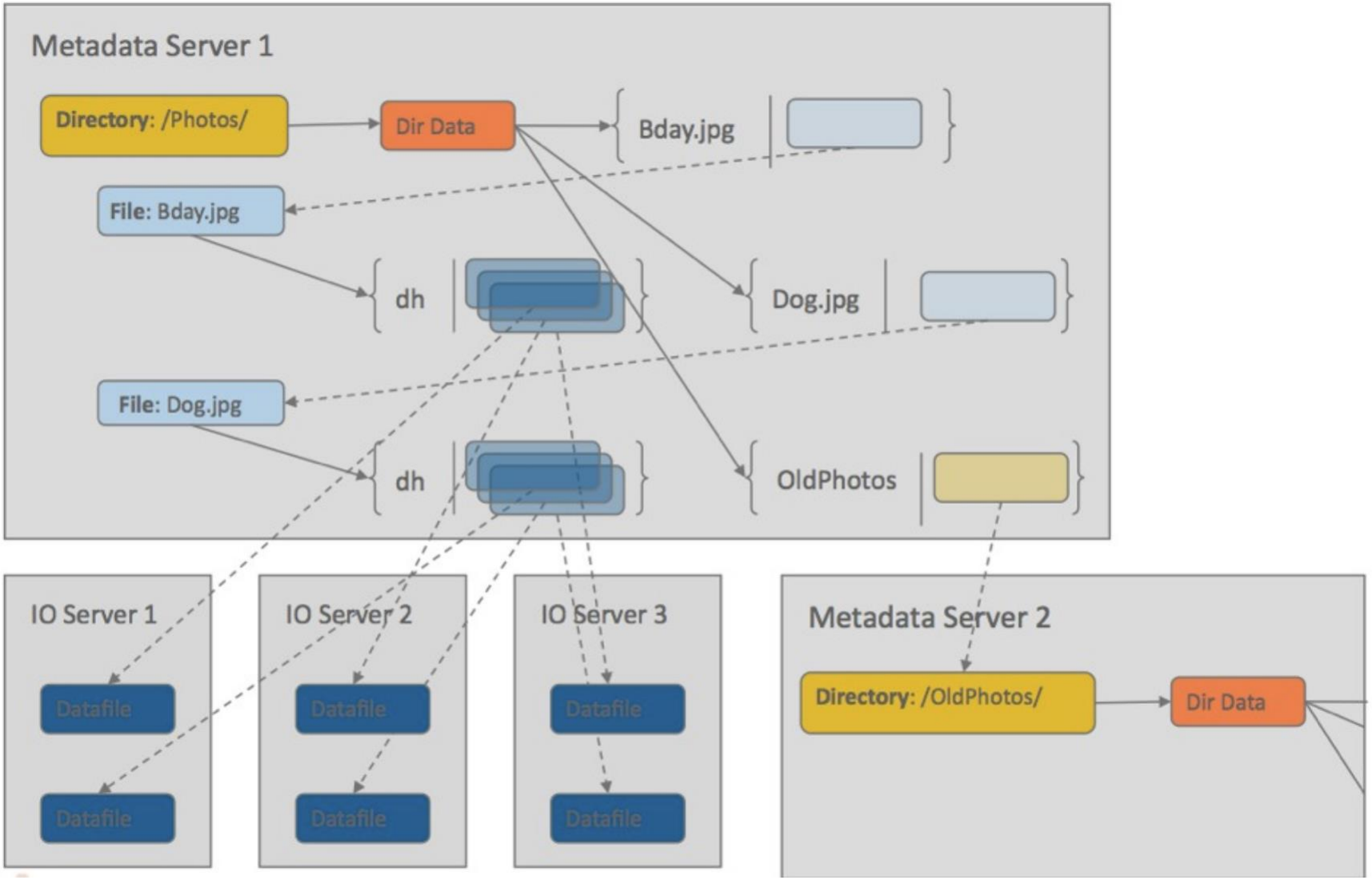
- Simply doesn't provide POSIX consistency guarantees
- If two clients write to overlapping regions concurrently, undefined results!
- Which applications perform this pattern of access (overwrite overlapping regions)?
  - Applications using the file system to update shared state, e.g. current value is 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?

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)



Distributing Directories in PVFS



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*

Parallel vs. Distributed

- Data distribution
  - Distributed file systems store files on a single storage node
  - Parallel file systems distribute data across multiple storage nodes
- Symmetry
  - DFS run on architectures where the storage is co-located with the application
  - PFS run on architectures where storage is physically separate
- Fault-tolerance
  - DFS take on fault-tolerance responsibilities
  - PFS run on enterprise shared storage
- Workloads
  - DFS are geared for loosely coupled, distributed applications (think data-intensive)
  - PFS target HPC applications, which tend to perform highly coordinated I/O accesses, and have massive bandwidth requirements