(Some) Types of File Systems

- (Local) Disk File System
  - Controls how data is stored and retrieved
  - ext4, NTFS, FAT
- Network File System
  - Any file system that is not local.
  - NFS is being served over a network, with the physical storage units and their management hosted by a different entity.
  - Clients will see a one server (or appearance of one server)
  - NFS, SMB
- Distributes File System
  - A file system that has its components spread across multiple systems.
  - Multiple servers with a “Shared Nothing” model
  - So a network file system is inherently a distributed file system as well.
  - Multiple servers coordinate to distribute the load across them in a way that is transparent to the client
  - A file may be located fully or partially in a specific node
  - HDFS, AFS, Ceph

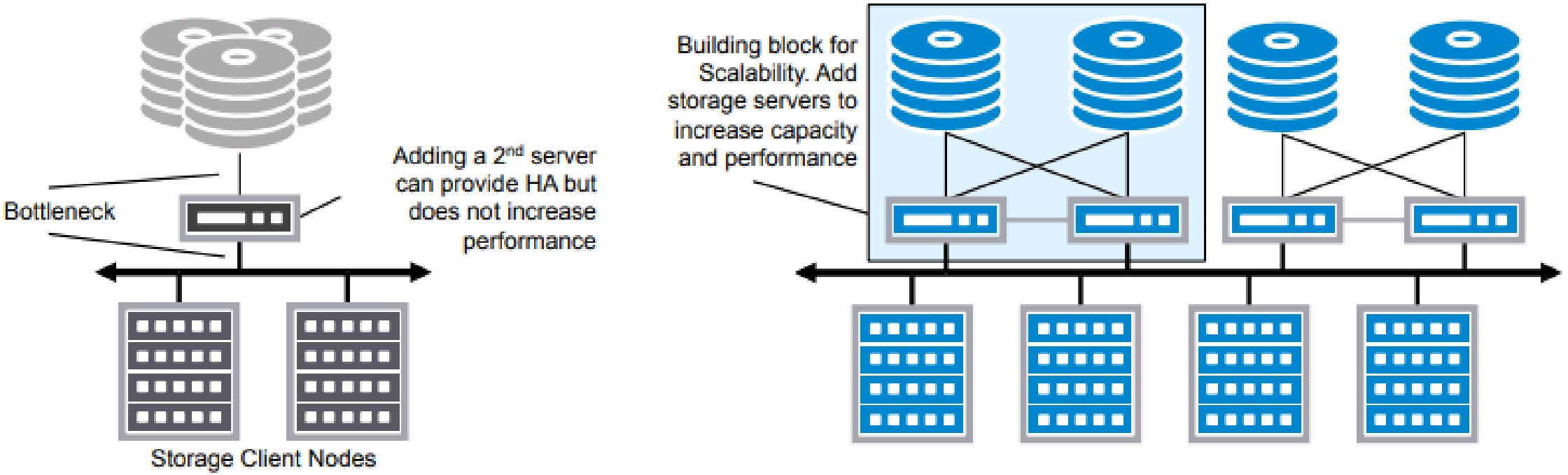
(Some) Types of File Systems

- Cluster File Systems
  - A clustered file system is a file system which is shared by being simultaneously mounted on multiple servers.
  - There are several approaches to clustering, most of which do not employ a clustered file system (only direct attached storage for each node).
  - Clustered file systems can provide features like location-independent addressing and redundancy which improve reliability or reduce the complexity of the other parts of the cluster.
  - Parallel file systems are a type of clustered file system that spread data across multiple storage nodes, usually for redundancy or performance.
  - Multiple serves with a “shared-disk” model (SAN with failover)
  - Designed with the assumption that data is stored on SAN
  - Data is accessible to multiple hosts over a shared SCSI connection
  - Can simultaneously be mounted by multiple hosts at the same time
  - CFS drivers take special care not to step on each other
  - It is also a Distributed File System which is served over a network
  - GPFS, Vertias CFS, Oracle CFS,

(Some) Types of File Systems

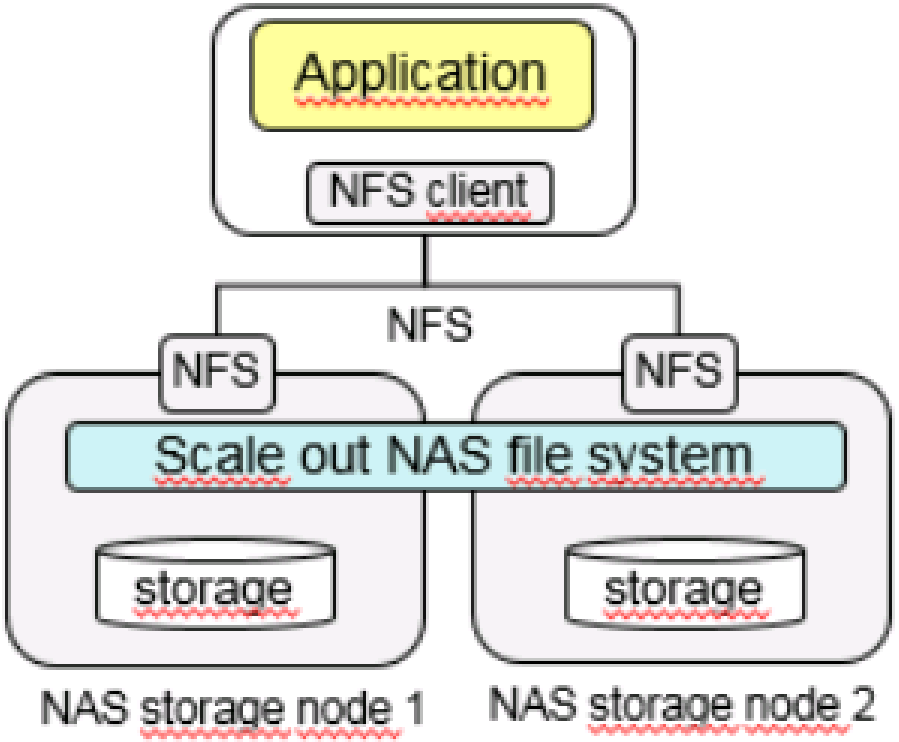
- Parallel File System
  - A natural next step from the distributed file systems
  - Designed with the assumption that “multiple processes writing to the same file simultaneously (parallel access)” is a common practice.
  - Unlike distributed file system, a file is striped in objects which are all placed in different server nodes
  - Allow multiple clients to read and write concurrently from the same file in a more efficient way
  - Support for parallel IO is essential for the performance of many applications
  - Uses an appropriate strategy to write a single file on different storage devices to retrieve performance
  - Lustre is a parallel distributed file system

Scale-Up vs Scale-Out



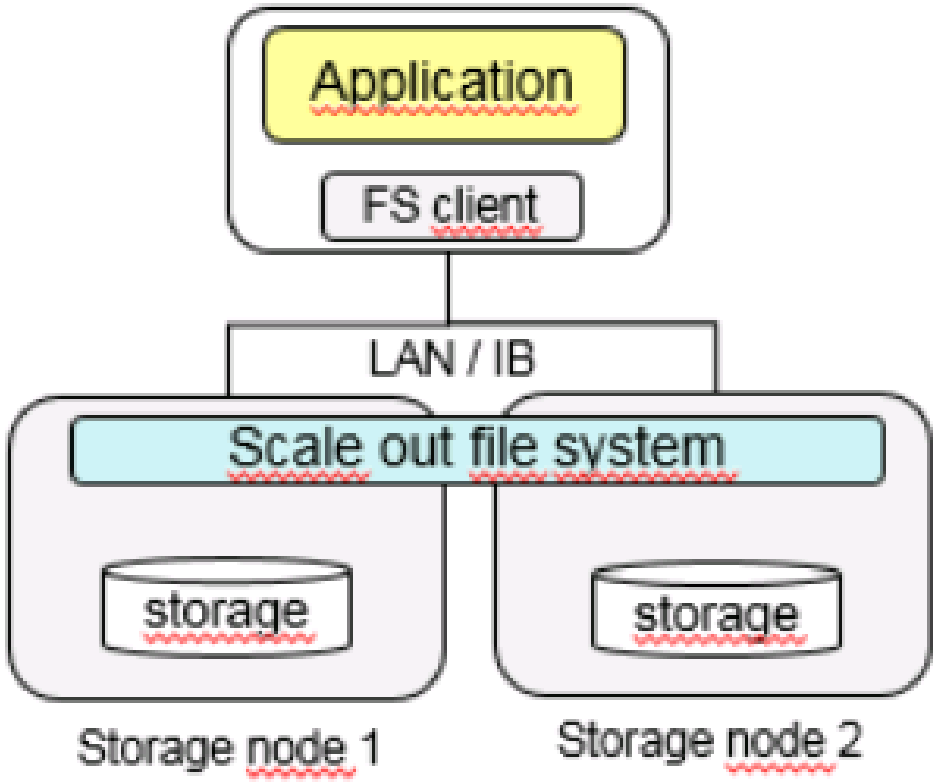
Scale-Out NAS system architecture with NFS access

- A scale out NAS system is comprised of multiple NAS storage nodes with internal or external storage (it does not really matter)
- Each NAS storage node represents the same file systems via file or object protocols such as NFS, SMB or OpenStack® Swift to the applications (global name space)
- Data is stored across all NAS storage nodes.
- Scalability can be achieved by adding more NAS storage nodes.



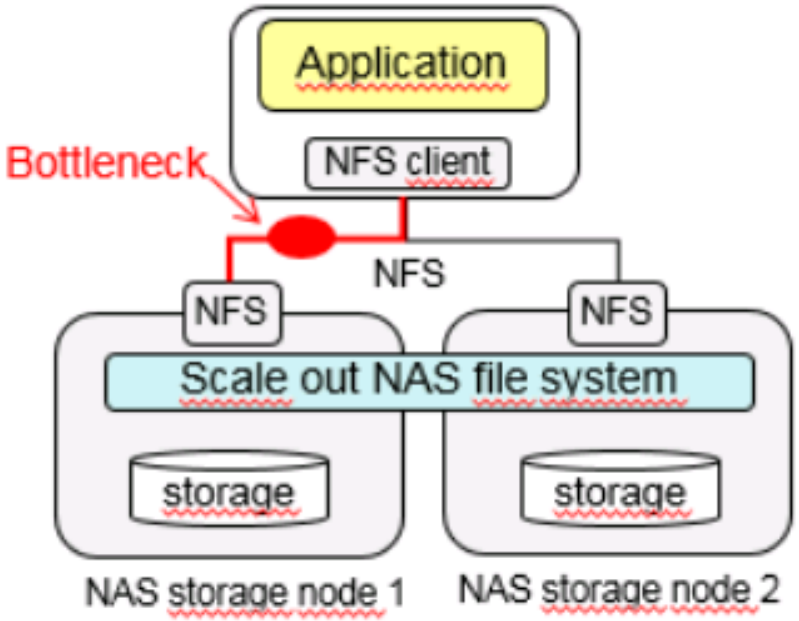
Scale-Out File System

- A scale out file system has a similar architecture as a scale out NAS system and is also comprised of multiple storage nodes with internal or external storage with subtle difference
- Unlike a scale out NAS system access to the file system is granted through the file system interface of the scale out file system and not through NAS protocols



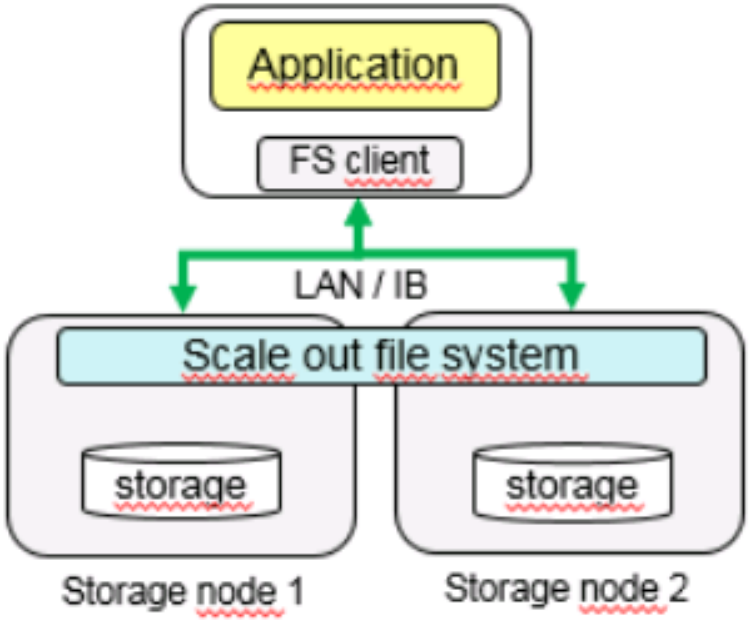
NFS performance bottleneck

Scale out NAS system bottleneck



With NFS a single data stream is limited to one NAS storage node

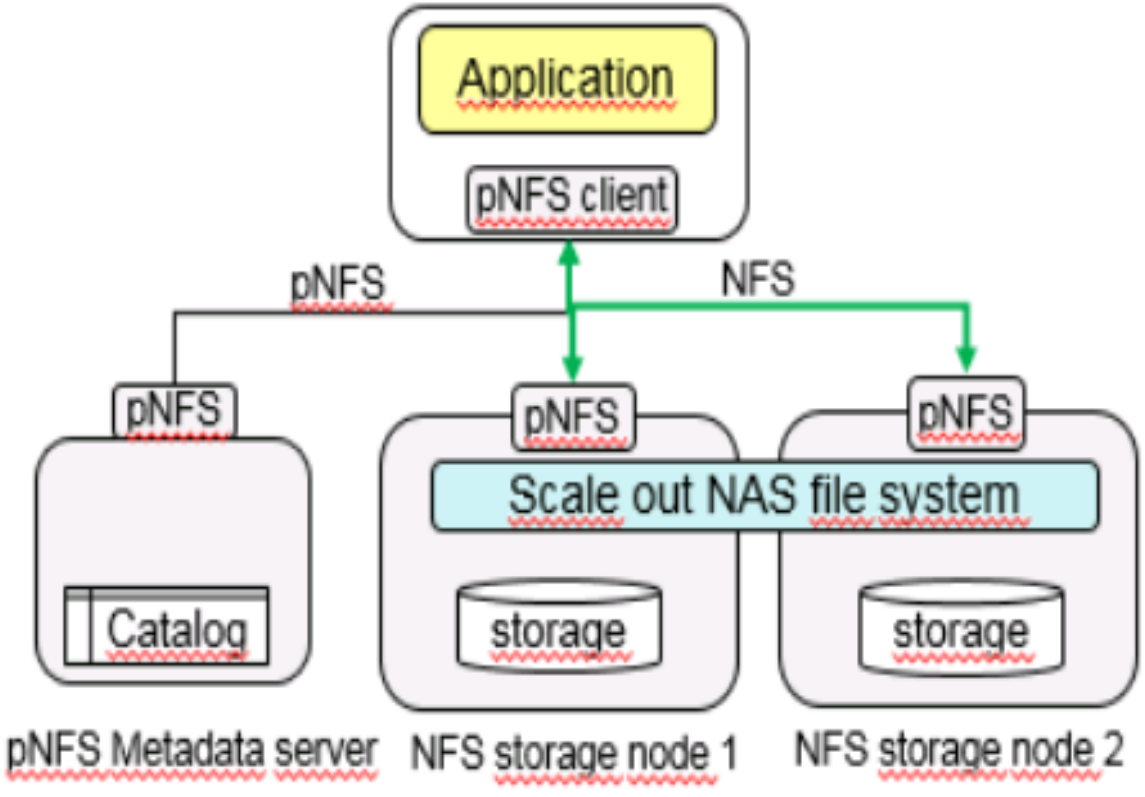
Scale out file system NO bottleneck



With scale out file system a single data stream is scales across all storage nodes

pNFS

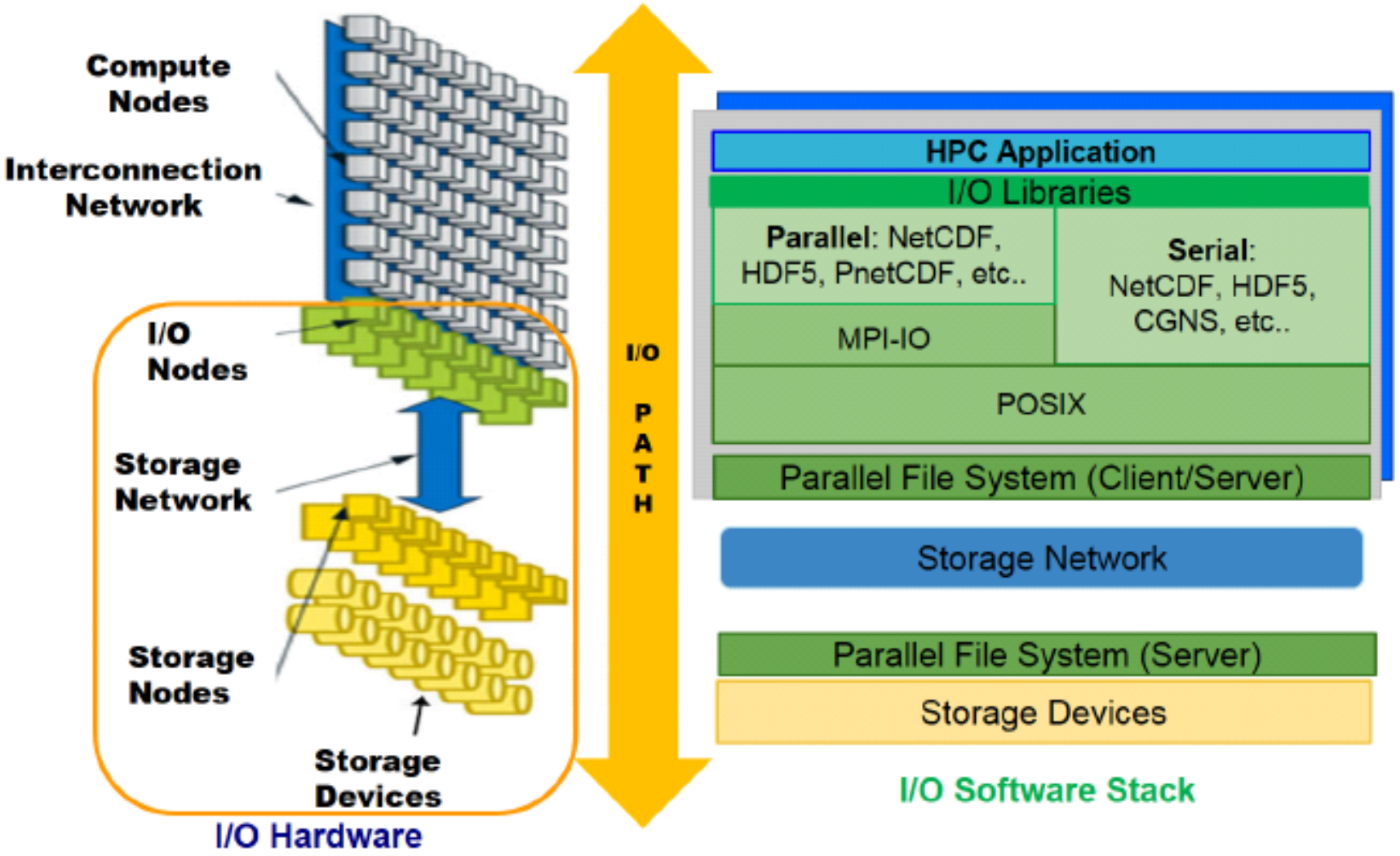
- Parallel NFS (pNFS) provides the ability to stripe single file I/O operations across multiple NAS storage nodes
- With pNFS there is a metadata server that controls the striping across multiple NAS storage nodes by maintaining information about location and layout of files.
- Based on this location information the pNFS client stripes a file across multiple NAS storage nodes. This reduces the NFS performance bottleneck to a certain level when streaming a single file, but still does not resolve the underlying point-to-point connection problem with NFS.
- Parallel NFS is not a mandatory feature within NFS version 4.1, this may be one reason it has not found broad acceptance in the market. In fact, most scale out NAS systems do **NOT** support pNFS today.
-



What does parallel I/O mean?

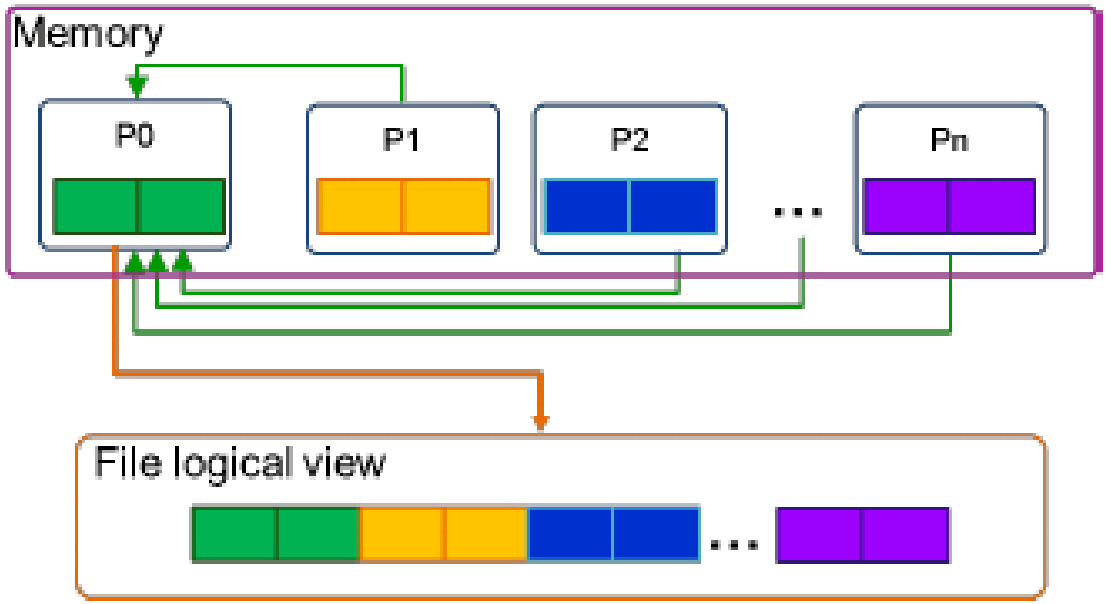
- At the program level
  - Concurrent reads or writes from multiple processes to a common file
- At the system level
  - A parallel file system and hardware that supports such concurrent access

Typical high performance IO system



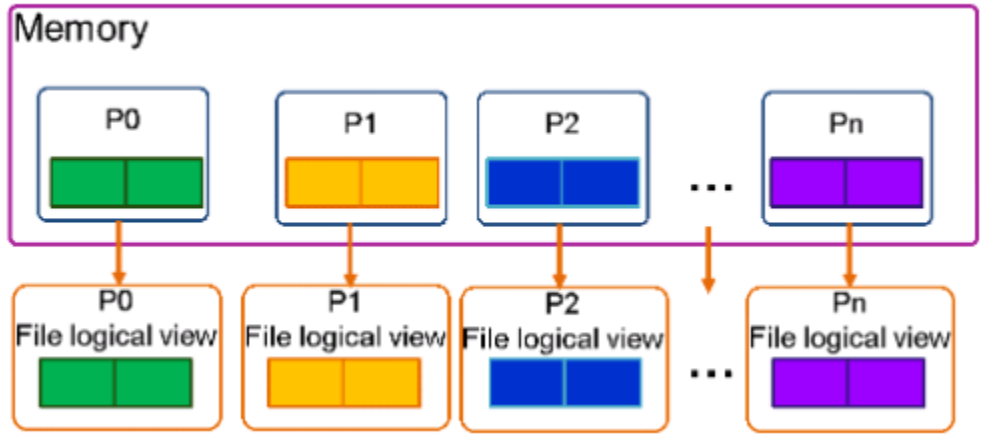
I/O strategies in parallel applications

- Single file, single writer (Serial I/O)
  - A common approach is to funnel all the I/O through a single master process. Although this has the advantage of producing a single file, the fact that only a single client is doing all the I/O means that it gains little benefit from the parallel file system.
  - A single master task coordinates the data rearrangement, e.g. receiving rows from many tasks and reconstructing the global data set prior to writing it out.
  - This pattern is also called Master I/O.
  - Normally a single process cannot benefit from the total available bandwidth and such an access scheme is also limited by the memory capabilities of the single master process.
  - Larger chunks of data might be transferred step by step which serialises the write process even more.



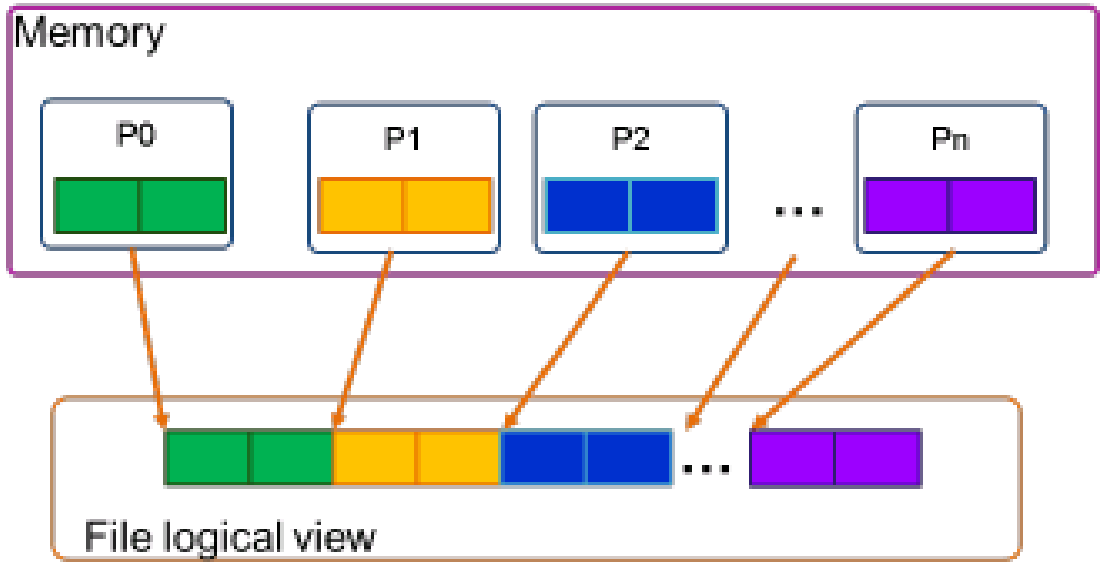
I/O strategies in parallel applications

- Multiple files, multiple writers (File per process approach)
  - Each task writing its data to a different file.
  - In practice this does not avoid the issue, but simply delays it to a later time
    - subsequent post-processing or analysis programs will almost certainly have to access multiple files to read the data they require.
  - For very large core counts (>10,000) this scheme starts to run up against technological limits in parallel file systems (particularly in metadata operations) and this limits the scaling of this approach at this scale.



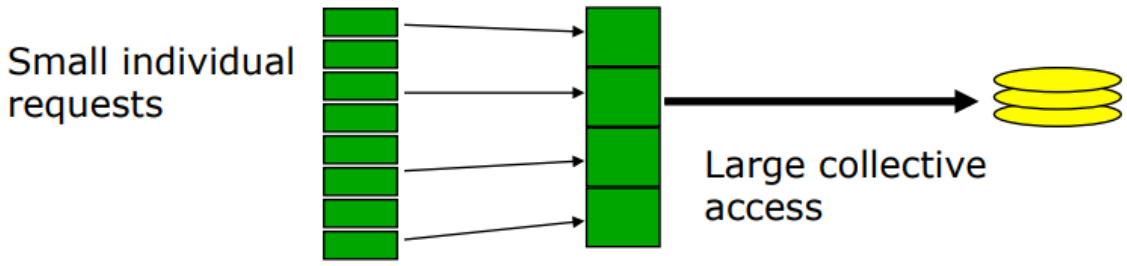
I/O strategies in parallel applications

- Single file, multiple writers without collective operations
  - There are a number of ways to achieve this. For example, many processes can open the same file but access different parts by skipping some initial offset; parallel I/O libraries such as MPI-IO, HDF5 and NetCDF also enable this.
  - Shared-file I/O has the advantage that all the data is organised correctly in a single file making analysis or restart more straightforward.
  - The problem is that, with many clients all accessing the same file, there can be a lot of contention for file system resources.



I/O strategies in parallel applications

- Single shared file, multiple writes with collective operations
  - if I/O is done collectively where the library knows that all clients are doing I/O at the same time, then reads and writes can be explicitly coordinated to avoid clashes. It is only through collective I/O that the full bandwidth of the file system can be realised while accessing a single file.
  - Basic idea is to build large blocks, so that reads/writes in I/O system will be large
    - Requests from different processes may be merged together
    - Particularly effective when the accesses of different processes are noncontiguous and interleaved



What is Lustre?

- Open-source object-based, distributed, parallel clustered file system
- Storage architecture for clusters
  - Power the largest HPC systems in the world
- POSIX standard compliant UNIX file system interface
- Kernel Open Source File System
- Can achieve performance of TB/s and manage Petabytes of storage
- Scalable design
  - Client #
  - Server #
  - Storage system #

What Lustre is not?

- A non Linux FS
- A user space file-system
- A work-station file system for Campus Area Network
- A distributed file-system for Wide Area Network

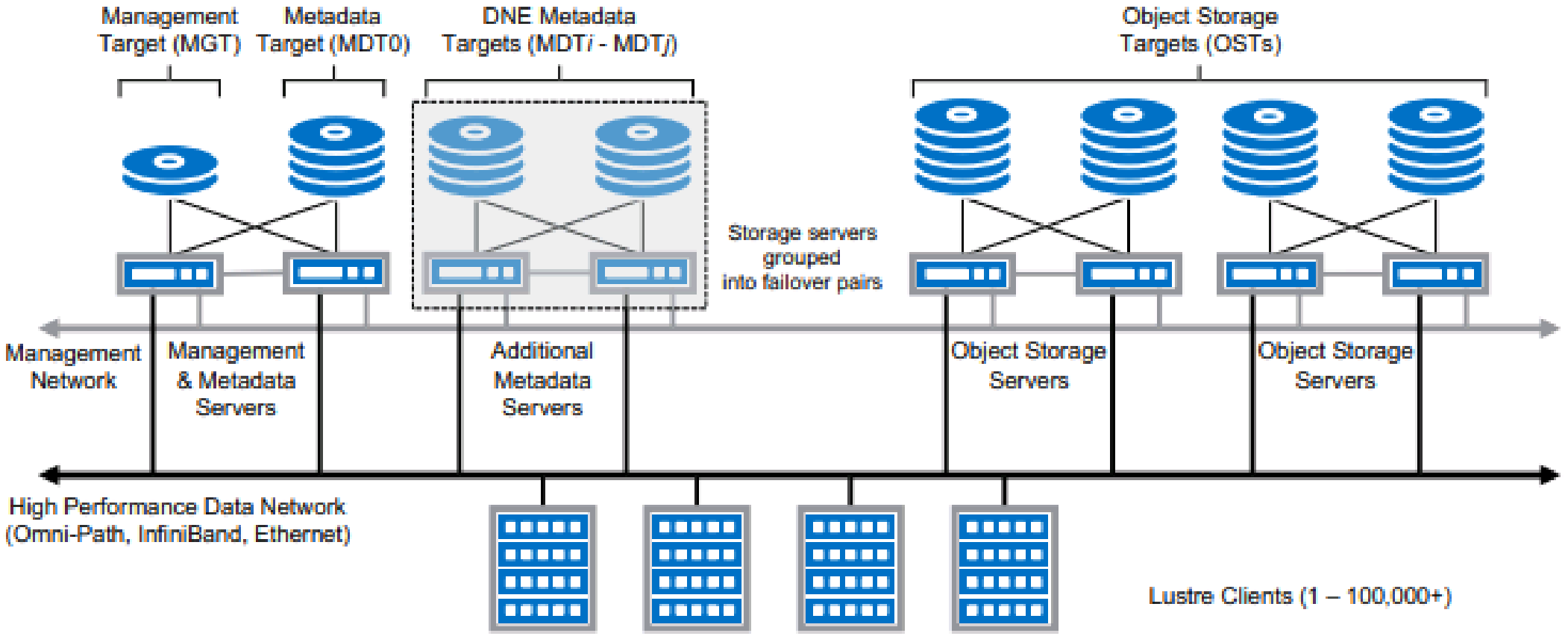
Lustre Server Components

- MGS + MGT
  - Management service, provides a registry of all active Lustre servers and clients
  - Stores Lustre configuration information.
  - MGT is the management service storage target used to store configuration data
  - Each target register to MGS
  - Each client contact MGS to retrieve information
- MDS + MDT
  - Metadata service, provides file system namespace (the file system index), storing the inodes for a file system.
  - MDT is the metadata storage target, the storage device used to hold metadata information persistently. Multiple MDS and MDTs can be added to provide metadata scaling.
  - Maintains MD which includes information seen via stat()
- OSS + OST
  - Object Storage service, provides bulk storage of data.
  - Files can be written in stripes across multiple object storage targets (OSTs).
  - Striping delivers scalable performance and capacity for files.

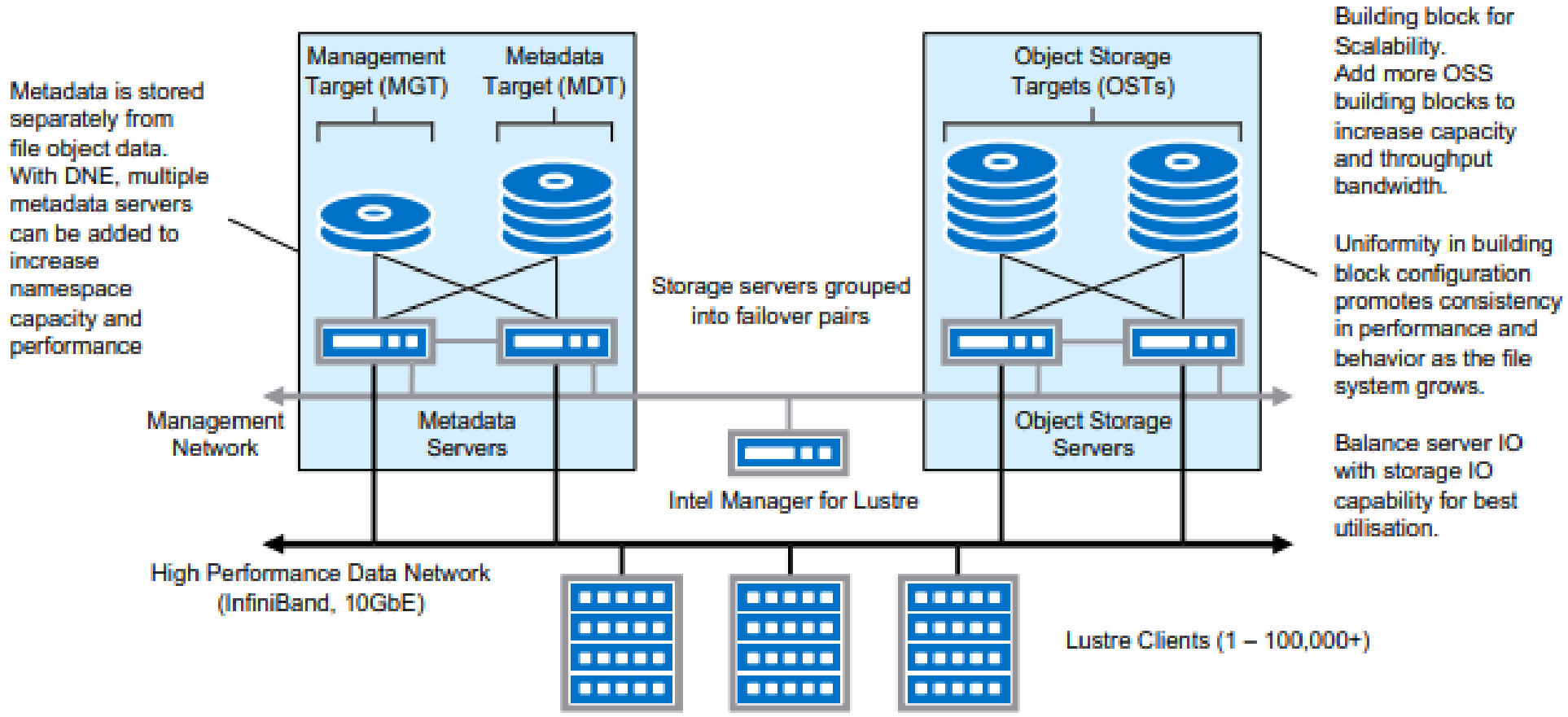
Lustre Server Components

- Clients
  - Lustre clients mount each Lustre file system instance using the Lustre Network protocol (LNet).
  - Presents a POSIX-compliant file system to the OS.
  - Applications use standard POSIX system calls for Lustre IO, and do not need to be written specifically for Lustre.
- Network
  - Lustre is a network-based file system, all IO transactions are sent using network RPCs.
  - Clients have no local persistent storage and are often diskless.
  - Supports many different network technologies, including OPA, IB, Ethernet.

Lustre Scalable Storage



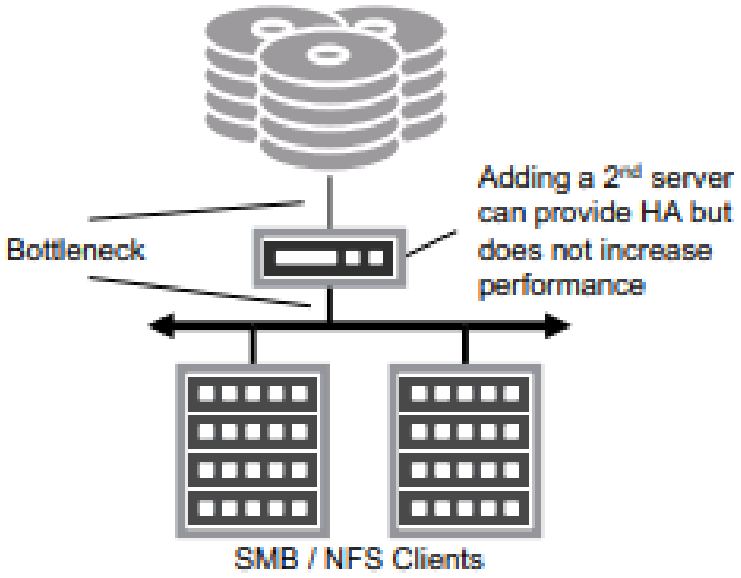
Lustre Building Blocks



Lustre Storage Scalability

	Value using LDISKFS backend	Value using ZFS backend	Notes
Maximum stripe count	2000	2000	Limit is 160 for ldiskfs if "ea_inode" feature is not enabled on MDT
Maximum stripe size	< 4GB	< 4GB	
Minimum stripe size	64KB	64KB	
Maximum object size	16TB	256TB	
Maximum file size	31.25PB	512PB*	
Maximum file system size	512PB	8EB*	
Maximum number of files or subdirectories per directory	10M for 48-byte filenames. 5M for 128-byte filenames.	2^{48}	
Maximum number of files in the file system	4 billion per MDT	256 trillion per MDT	
Maximum filename length	255 bytes	255 bytes	
Maximum pathname length	4096 bytes	4096 bytes	Limited by Linux VFS

Traditional Network File System vs Lustre



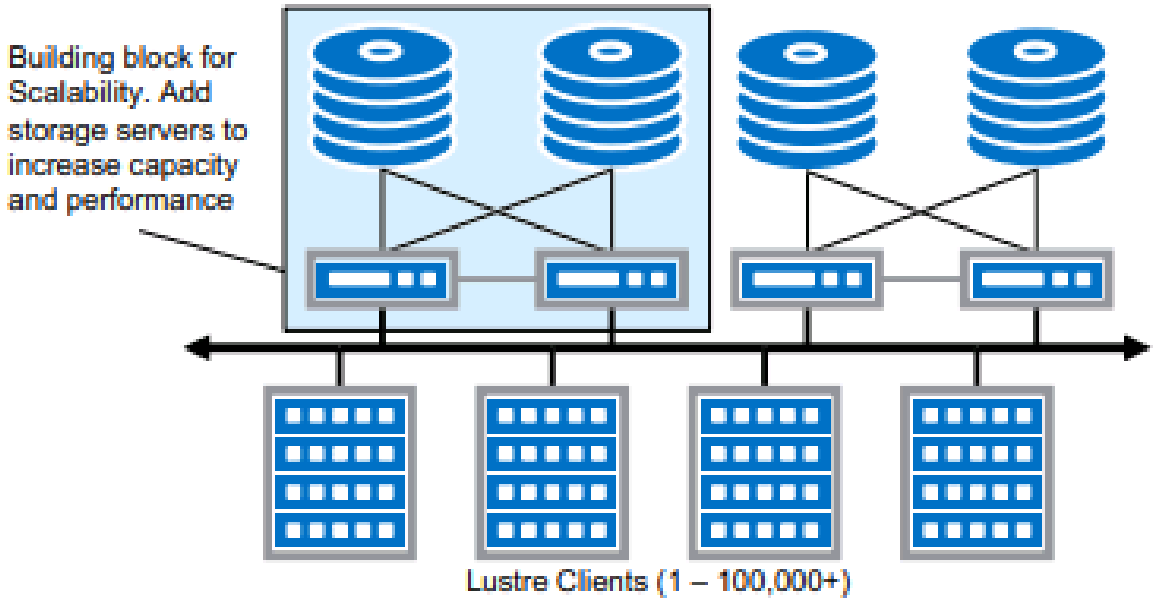
Distributed client, central server, e.g. NFS, SMB

Performance and scalability is limited by single server bandwidth

All clients use a single network path to one server

Adding a new storage server creates a separate file system

Storage becomes fragmented into silos



Distributed client, distributed servers (Lustre)

Scalable Metadata and Object Storage services

Scale performance and capacity linearly as client population grows

Client network scalability is not limited by the file system

All clients can access all storage servers simultaneously and in parallel

Single coherent name space across all servers. Can have hundreds of network storage servers in a single file system

Lustre Installation (http://wiki.lustre.org/Installing_the_Lustre_Software)

- Server
  - e2fsprogs
  - epel repo
  - lustre-patched kernel package
  - reboot
  - lustre and ldiskfs kmod packages
  - Verify that software is installed correctly (Load lustre kernel module)
  - After verification unload lustre module from kernel
- Client
  - kernel packages
  - reboot
  - epel repo
  - Lustre client user-space tools and DKMS kernel module package
  - Verify that software is installed correctly (Load lustre kernel module)
  - After verification unload lustre module from kernel

Installation

- Server (MGS/MDS)
  - (iptables –F)
  - Create /etc/yum.repos.d/CentOS-Lustre.repo as mentioned at http://wiki.lustre.org/Installing_the_Lustre_Software
  - yum update
  - yum -y install epel-release
  - yum --nogpgcheck --disablerepo=* --enablerepo=e2fsprogs-wc install e2fsprogs
  - yum --nogpgcheck --disablerepo=base,extras,updates --enablerepo=lustre-server install kernel kernel-devel kernel-headers kernel-tools kernel-tools-libs kernel-tools-libs-devel --skip-broken
  - reboot
  - yum --nogpgcheck --enablerepo=lustre-server install kmod-lustre kmod-lustre-osd-ldiskfs lustre-osd-ldiskfs-mount lustre lustre-resource-agents
  - modprobe -v lustre
  - lustre_rmmod

Installation

- Which packages are installed on server
  - # rpm -qa | grep lustre
  - lustre-osd-ldiskfs-mount-2.12.3-1.el7.x86_64
  - kmod-lustre-2.12.3-1.el7.x86_64
  - kernel-3.10.0-1062.1.1.el7_lustre.x86_64
  - lustre-2.12.3-1.el7.x86_64
  - lustre-resource-agents-2.12.3-1.el7.x86_64
  - kmod-lustre-osd-ldiskfs-2.12.3-1.el7.x86_64
  - kernel-devel-3.10.0-1062.1.1.el7_lustre.x86_64

Installation

- Client Installation
  - iptables –F
  - create /etc/yum.repos.d/CentOS-Lustre.repo (as mentioned in URL http://wiki.lustre.org/Installing_the_Lustre_Software)
  - yum update
  - yum -y install epel-release
  - yum install kernel kernel-devel kernel-headers kernel-abi-whitelists kernel-tools kernel-tools-libs kernel-tools-libs-devel
  - reboot
  - yum install epel-release
  - yum --nogpgcheck --enablerepo=lustre-client install lustre-client-dkms lustre-client
  - modprobe -v lustre
  - lustre_rmmod
  - uname -a

Installation

- Which packages are installed on client
  - # rpm -qa | grep lustre
  - lustre-client-2.12.3-1.el7.x86_64
  - lustre-client-dkms-2.12.3-1.el7.noarch

Lustre Networking

- A Lustre network is comprised of clients and servers running the Lustre software.
- It need not be confined to one LNet subnet but can span several networks provided routing is possible between the networks.
- In a similar manner, a single network can have multiple LNet subnets
- The Lustre networking stack is comprised of two layers
  - LNet and
  - LND (Lustre Network Driver)
- LNet layer operates above the LND layer in a manner similar to the way the network layer operates above the data link layer.
- LNet layer is connectionless, asynchronous and does not verify that data has been transmitted while the LND layer is connection oriented and typically does verify data transmission

LNet

- Originally derived from a project called Portals
- Lightweight, efficient and versatile, capable of operating over different network fabrics
- Implemented as a Linux kernel module
- The LND provides an interface abstraction between the upper level LNet protocol and the kernel device driver for the network interface
- Multiple LNDs can be active on a host simultaneously
- LNets are uniquely identified by a label comprised of a string corresponding to an LND and a number that uniquely identifies each LNet
  - tcp0, o2ib0, or o2ib1
- Each node on an LNet has at least one network identifier (NID).
- A NID is a combination of the address of the network interface and the LNet label in the form: address@LNet_label
  - 192.168.1.2@tcp0
  - 10.13.24.90@o2ib1

Supported networks Types

- The LNet code module includes LNDs to support many network types
  - InfiniBand: OpenFabrics OFED (o2iblnd)
  - TCP (any network carrying TCP traffic, including GigE, 10GigE, and IPoIB) (socklnd)
  - Atos BXI (Bull eXtreme Interconnect) (ptlflnd)
  - Cray Aries (gnilnd)
  - RapidArray
  - Quadrics
  - Intel Omni-path(OPA) : o2iblnd
- (o2iblnd is OpenFabrics Enterprise Distribution (OFED) Lnet Driver A “standard” RDMA interface)

LNet Configuration

- # echo "options lnet networks=tcp0(ens192)" > /etc/modprobe.d/lustre.conf
- # modprobe lnet
- # lctl network configure (lnetctl lnet configure)
- # lctl list_nids

  192.168.20.1@tcp
- # lnetctl global show

```
global:
    numa_range: 0
    max_intf: 200
    discovery: 1
    drop_asym_route: 0
    retry_count: 3
    transaction_timeout: 10
    health_sensitivity: 100
```

LNet Configuration

- “lctl ping” to ensure that nodes are connected to each other
  - MGS/MDS
    - [root@ntro-team01 ~]# cat /etc/modprobe.d/lustre.conf
      options lnet networks="tcp0(ens192)“
    - [root@ntro-team01-node2 ~]# lctl list_nids
      192.168.20.2@tcp
    - [root@ntro-team01-node2 ~]# lctl ping 192.168.20.1@tcp0
      12345-0@lo
      12345-192.168.20.1@tcp
  - OSS
    - [root@ntro-team01-node2 ~]# cat /etc/modprobe.d/lustre.conf
      options lnet networks="tcp0(ens192)“
    - [root@ntro-team01 ~]# lctl list_nids
      192.168.20.1@tcp
    - [root@ntro-team01 ~]# lctl ping 192.168.20.2@tcp0
      12345-0@lo

LNet Configuration

- “lctl ping” to ensure that nodes are connected to each other
  - Client
    - [root@ntro-team01-node3-1 ~]# cat /etc/modprobe.d/lustre.conf
      options lnet networks="tcp0(ens192)“
    - [root@ntro-team01-node3-1 ~]# lctl list_nids
      192.168.20.3@tcp

Create MGS, MDS and mount Lustre FS on client

- MDS/MGS (192.168.20.2)
  - mkfs.lustre --fsname lustre --mgs --mdt --index=0 /dev/sdb
  - mkdir /mnt/MDS
  - mount -t lustre /dev/sdb /mnt/MDS/

- OSS (192.168.20.1)
  - mkfs.lustre --fsname lustre --ost --mgsnode=192.168.20.2@tcp0 --index=0 /dev/sdb
  - mkdir /mnt/ost0
  - mount -t lustre /dev/sdb /mnt/ost0

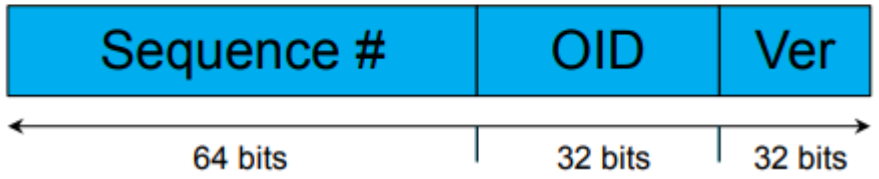
Create MGS, MDS and mount Lustre FS on client

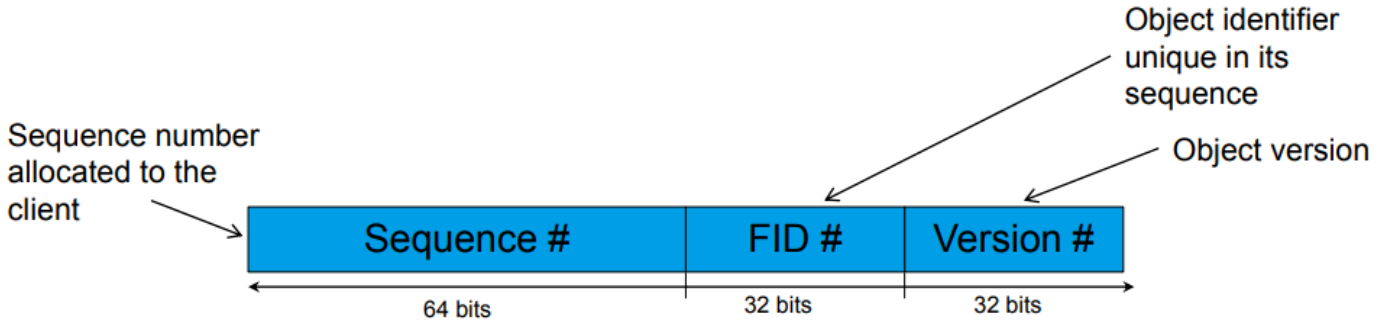
- Client (192.168.20.3)
  - modprobe lustre
  - mkdir /mnt/lustre
  - mount -t lustre 192.168.20.2@tcp0:/lustre /mnt/lustre
  - lfs df –h # what is the capacity
- OSS (192.168.20.2)
  - mkfs.lustre --fsname lustre --ost --mgsnode=192.168.20.2@tcp0 --index=1 /dev/sdc
  - mkdir /mnt/ost1
  - mount -t lustre /dev/sdc /mnt/ost1
  - df –h # Now look at the capacity in client again!!

File Identifiers (FID)

- Independent of backend filesystem
- Although not used at this time, FID added support for object versioning
- 128-bit identifier
  - 64 bit sequence number (new sequence number for each client)
  - 32 bit object id (sequential within each sequence)
  - 32 bit version (for snapshots, datasets, currently unused)
  - Unique across entire Lustre filesystem
  - All files have 64 bit inode numbers

```
[client]# lfs path2fid foobar
[0x200000400:0x123:0x0]
```





FID

- Sequences
  - Sequences are granted to clients by servers
  - When a client connects, a new FID sequence is allocated
    - upon disconnect, new sequence is allocated to client when it comes back
  - Each sequence has a limited number of objects which may be created in it
    - on exhaustion, a new sequence should be started
  - Sequences are cluster-wide and prevent FID collision

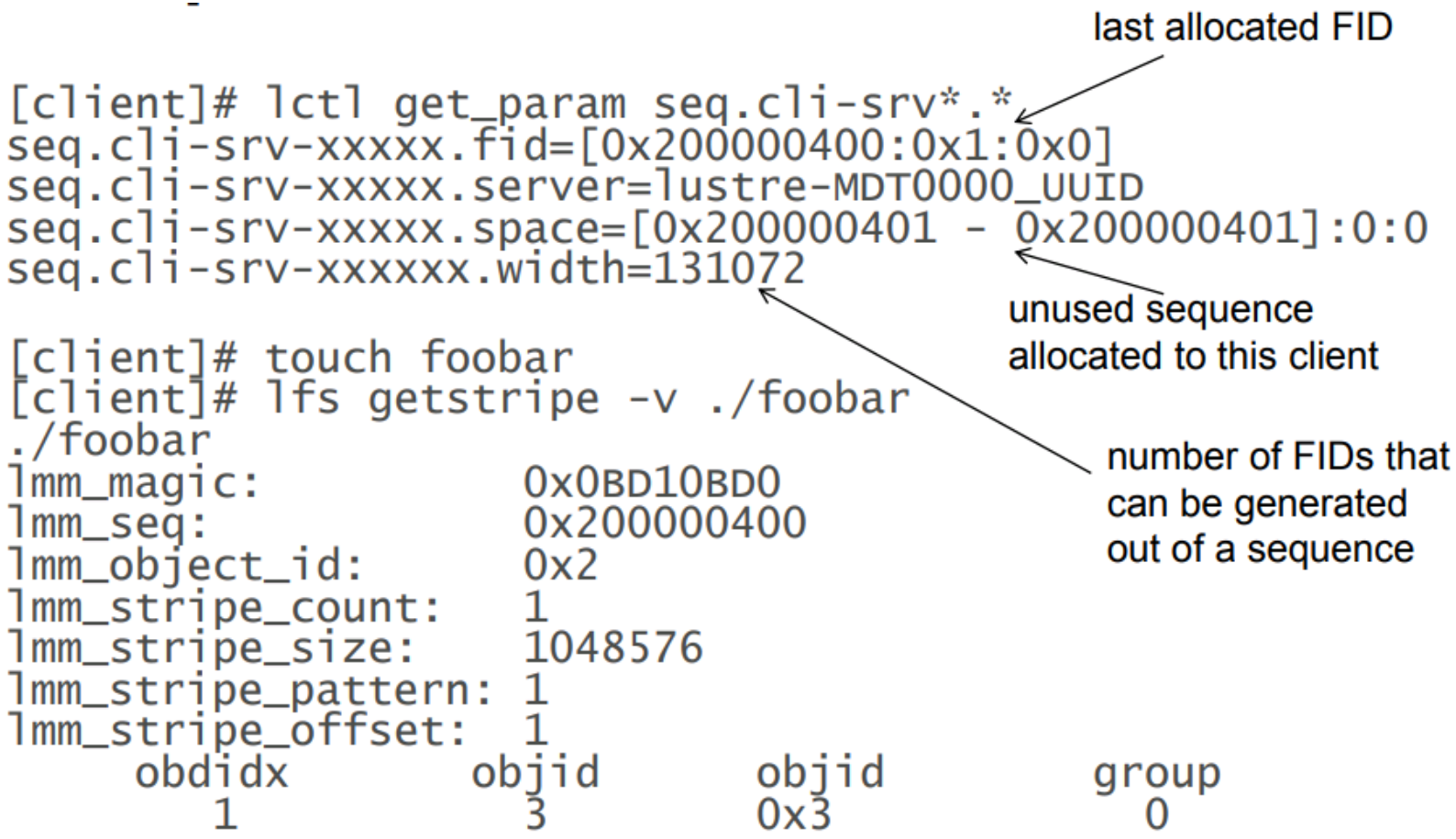
FID

- Each lustre object is associated with
  - A path name
  - An FID
- To get object FID
  - $ lfs path2fid object
- To get path from FID
  - lfs fid2path FSNAME FID

```
[client]# touch foobar
[client]# lfs path2fid foobar
[0x200000400:0x123:0x0]

[client]# lfs getstripe -v foobar
lmm_seq:            0x200000400
lmm_object_id:      0x123
  :                  :

[client]# ls -i foobar
1441152052557253 47 foobar
[client]# printf "%#x\n" $(stat -c %i /mnt/lustre/etc/hosts)
0x200000400000123

[client]# ln foo bar; mkdir tmp; ln foo tmp/baz
[client]# lfs fid2path /mnt/lustre [0x200000400:0x123:0x0]
/mnt/lustre/foo
/mnt/lustre/bar
/mnt/lustre/tmp/baz
```

FID

- Sequence in practice



FID

- Where are FIDs stored?
  - The underlying filesystem still operates on inodes
  - An object index is stored on disk to handle FID/on-disk inode mapping
  - For ldiskfs, the object index is an IAM lookup table (namely oi.16)
    - Object Index (OI) stores FID->ino translation
  - FID is also stored in an extended attribute called LMA (Lustre Metadata Attribute)

```
debugfs:  ls
    2(12)   .            2(12) ..           11(20) lost+found
   12(16)   CONFIGS  25001(16) OBJECTS      19(20) lov_objid
   22(16)   oi.16       23(12) fld          24(16) seq_srv
   25(16)   seq_ctl     26(20) capa_keys 25002(16) PENDING
25003(12)   ROOT        27(20) last_rcvd 25004(20) REM_OBJ_DIR
   31(3852)CATALOGS
```

MGC, MDC, OSC

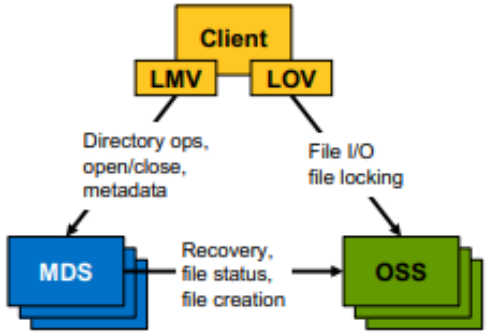
- Lustre client combines the metadata and object storage into a single, coherent POSIX file system
  - Presented to the client OS as a file system mount point
  - Applications access data as for any POSIX file system
  - Applications do not therefore need to be re-written to run with Lustre
- All Lustre client I/O is sent via RPC over a network connection
  - Clients do not make use of any node-local storage, can be diskless
- The Lustre client software provides an interface between the Linux virtual file system and the Lustre servers.
- The client software is composed of several different services, each one corresponding to the type of Lustre service it interfaces to.
- A Lustre client instance will include
  - one management client (MGC)
  - one or more metadata clients (MDC), and
  - multiple object storage clients (OSCs), one corresponding to each OST in the file system.

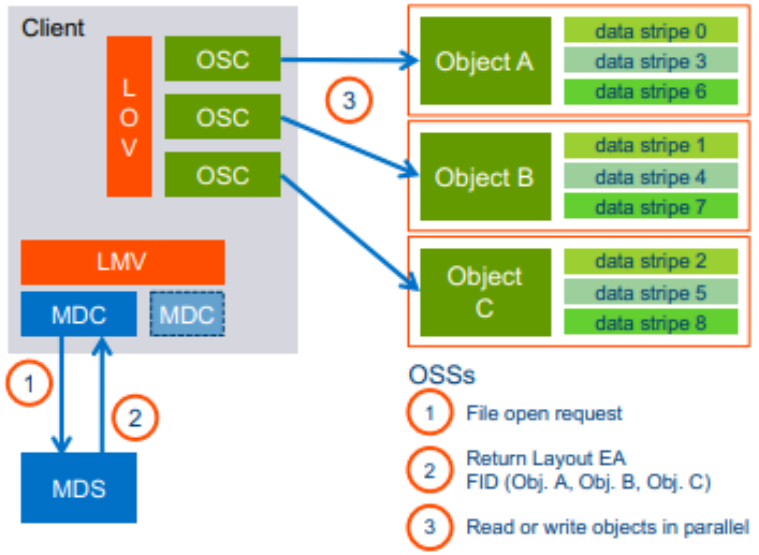
LMV, LOV

- Logical Metadata Volume (LMV)
  - Aggregates the MDCs and presents a single logical metadata namespace to clients
  - Provides transparent access across all the MDTs
  - This allows the client to see the directory tree stored on multiple MDTs as a single coherent namespace, and aggregates the MDCs and presents a single logical metadata namespace to clients, providing transparent access across all the MDTs.
- Logical Object Volume (LOV)
  - Aggregates the OSCs to provide transparent access across all the OSTs.
  - Client with the Lustre file system mounted sees a single, coherent synchronized namespace and all files (even striped ones) are presented within that namespace as a single addressable data object

Overview of Lustre IO Operations

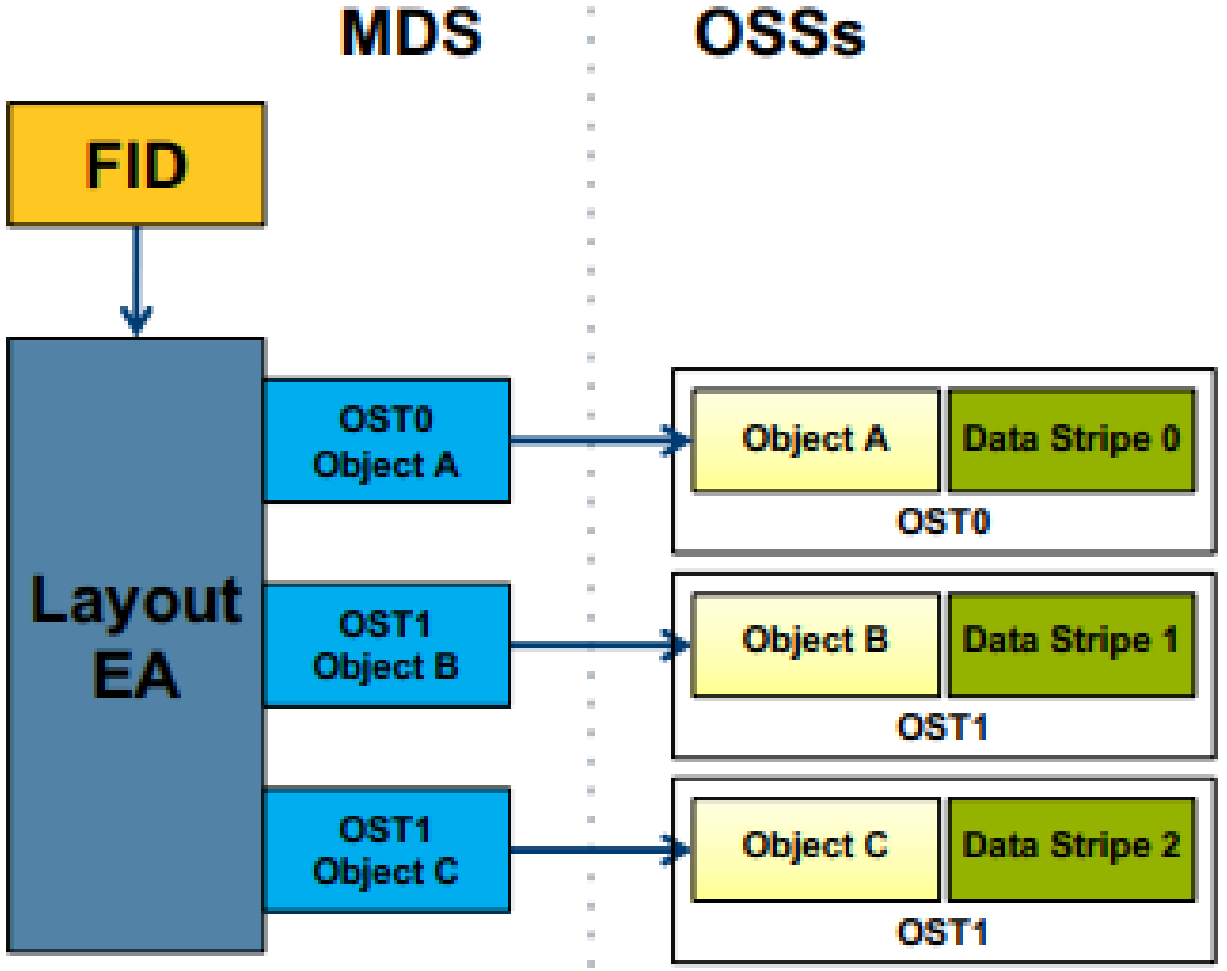
- read/write
  - Client sends an RPC to MDS to get a lock (either read lock for lookup or write lock for create)
  - MDS returns a lock and all the metadata attributes and file layout extended attributes to the client.
    - File layout contains list of OSTs containing file's data and the layout access pattern describing how the data has been distributed across the set of objects
    - Layout information allows the client to access data directly from the OSTs.
  - Client sends RPCs to each of the necessary OSSs to read/write the file data





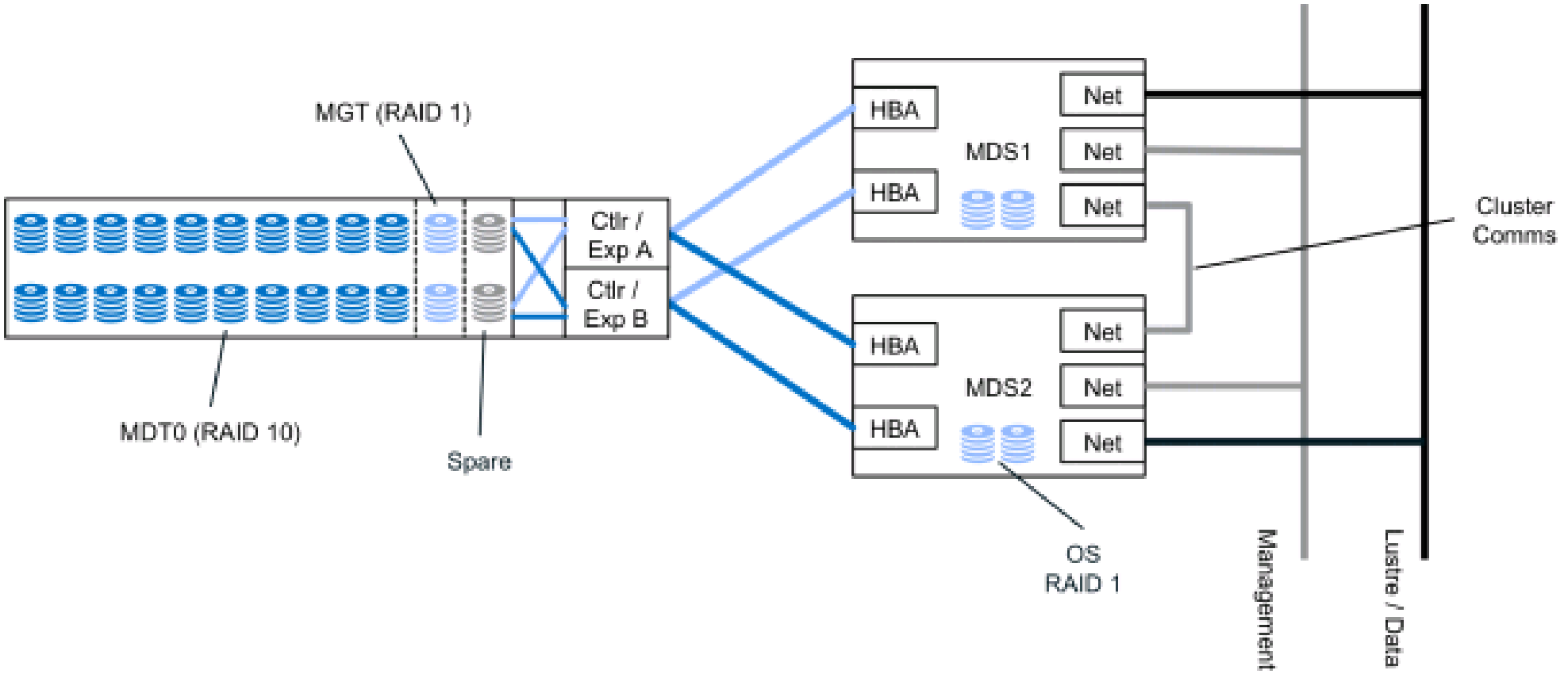
Layout EA

- Layout EA is an extended attribute stored as part of a file's metadata on the MDT
  - A list of FIDs, used to locate the file data objects on the OST(s)
  - The layout EA points to 1-to-N OST object(s) on the OST(s) that contain the file data
  - If the layout EA points to one object, all the file data is stored entirely in that object.
  - If the layout EA points to more than one object, the file data is striped across the objects using RAID 0, and each object is stored on a different OST



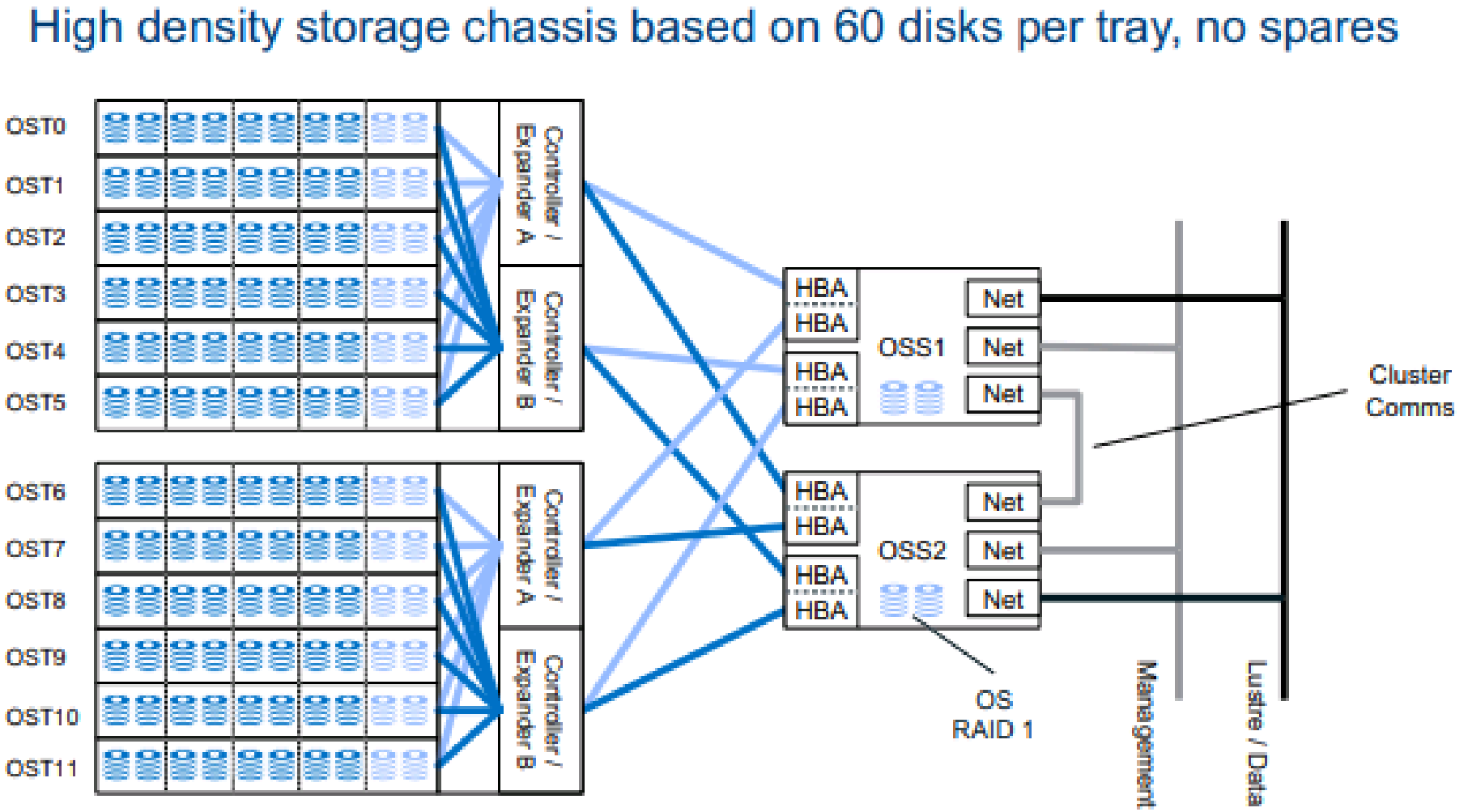
MGS and MDS Reference Design

- Active/Passive failover configuration



OSS Reference Design

- Active/Active failover configuration



Striping

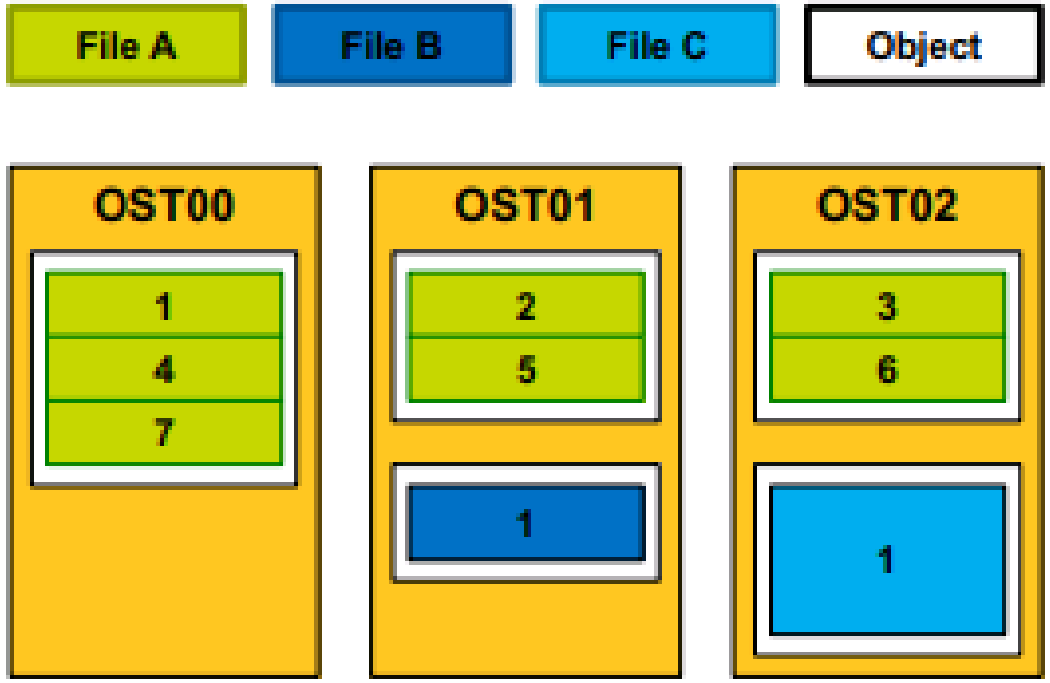
- Reasons to use striping
  - High bandwidth access
    - Many applications require high-bandwidth access to a single file, which may be more bandwidth than can be provided by a single OSS
  - Improving performance when OSS bandwidth is exceeded
    - Striping across many OSSs can improve performance if the aggregate client bandwidth exceeds the server bandwidth and the application reads and writes data fast enough to take advantage of the additional OSS bandwidth
  - Providing space for very large files
- Reasons to avoid striping
  - Increased overhead
    - Striping results in more locks and extra network operations during common operations such as stat and unlink.
  - Increased risk
    - When files are striped across all servers and one of the servers breaks down, a small part of each striped file is lost

Choosing a stripe size

- The stripe size must be a multiple of the page size
  - Lustre software tools enforce a multiple of 64 KB (the maximum page size on ia64 and PPC64 nodes)
- The smallest recommended stripe size is 512 KB
  - Although you can create files with a stripe size of 64 KB, the smallest practical stripe size is 512 KB because the Lustre file system sends 1MB chunks over the network
  - Choosing a smaller stripe size may result in inefficient I/O to the disks and reduced performance
- A good stripe size for sequential I/O using high-speed networks is between 1 MB and 4 MB
  - In most situations, stripe sizes larger than 4 MB may result in longer lock hold times and contention during shared file access
- The maximum stripe size is 4 GB
- Choose a stripe pattern that takes into account the write patterns of your application
  - Writes that cross an object boundary are slightly less efficient than writes that go entirely to one server

File Layout: Striping

- Each file in Lustre has its own unique file layout, comprised of 1 or more objects in a stripe equivalent to RAID 0
- File layout is allocated by the MDS
- Layout is selected by the client, either
  - by policy (inherited from parent directory)
  - by the user or application
- Layout of a file is fixed once created



Striping

- When objects of a file are created?
  - Objects for a file are normally allocated when the file is created.
  - Objects are initially empty when created and all objects for a file (or the first component of a PFL file) are created when the file itself is created. This means that if a file is created with a stripe count of 4 objects, all 4 of the objects will be allocated to the file, even if the file is initially empty, or has less than one stripe's worth of data.)

Administrative vs User Commands

- Administrative commands use “lctl”
- Non-Administrative commands use “lfs”

  - Example: Get information about the OST pools

```
$ lfs pool_list /myfs
Pools from myfs:
myfs.mypool
```

... and then follow up getting a list of the OSTs in a particular pool

```
$ lfs pool_list myfs.mypool
Pool: myfs.mypool
myfs-OST0001_UUID
myfs-OST0002_UUID
```

Striping: Examples

- Setting stripe size
  - [client]# lfs setstripe –S 4M /mnt/lustre/new_file
- Setting stripe count
  - [client]# lfs setstripe -c -1 /mnt/lustre/full_stripe
- Creating a file on a specific OST
  - $ lfs setstripe -c 1 -o 0 file1
  - $ lfs setstripe -c 1 -o 1 file2
- Display stripe information
  - $ lfs getstripe /mnt/lustre/file1
  - $ lfs getstripe /mnt/lustre/file2

Striping: Examples

- A file on multiple OSTs
  - [root@ntro-team01-node3-1 lustre]# lfs setstripe –c -1 /mnt/lustre/zero.dat
  - [root@ntro-team01-node3-1 lustre]# dd if=/dev/zero of=/mnt/lustre/zero.dat bs=4M count=2
    2+0 records in
    2+0 records out
    8388608 bytes (8.4 MB) copied, 0.697943 s, 12.0 MB/s
  - [root@ntro-team01-node3-1 lustre]# lfs getstripe /mnt/lustre/zero.dat

Striping: Examples

- Which OST a file is located on?
  - [root@ntro-team01-node3-1 lustre]# lfs getstripe zero.dat

    zero.dat

    lmm_stripe_count: 1

    lmm_stripe_size: 1048576

    lmm_pattern: raid0

    lmm_layout_gen: 0

    lmm_stripe_offset: 1

obdidx	objid	objid	group
1	2	0x2	0

  - The file is located at OST with index 1. To find which node is serving this OST,
    - [root@ntro-team01-node3-1 lustre]# lctl get_param osc.lustre-OST0001-osc*.ost_conn_uuid

      osc.lustre-OST0001-osc-ffff88f2f031a000.ost_conn_uuid=192.168.20.2@tcp

Some more commands

- List MDTs and OSTs
  - Client# lfs mdts /mnt/lustre
  - Client # lfs osts /mnt/lustre
- Search directory tree
  - Client# lfs find
- Check disk space usage
  - Client# lfs df –h
- Fid
  - Client# lfs getname /mnt/lustre/
    - lustre-ffff88f2f031a000 /mnt/lustre/
  - Client# lfs path2fid /mnt/lustre/zero.dat
    - [0x2000013a1:0x1:0x0]

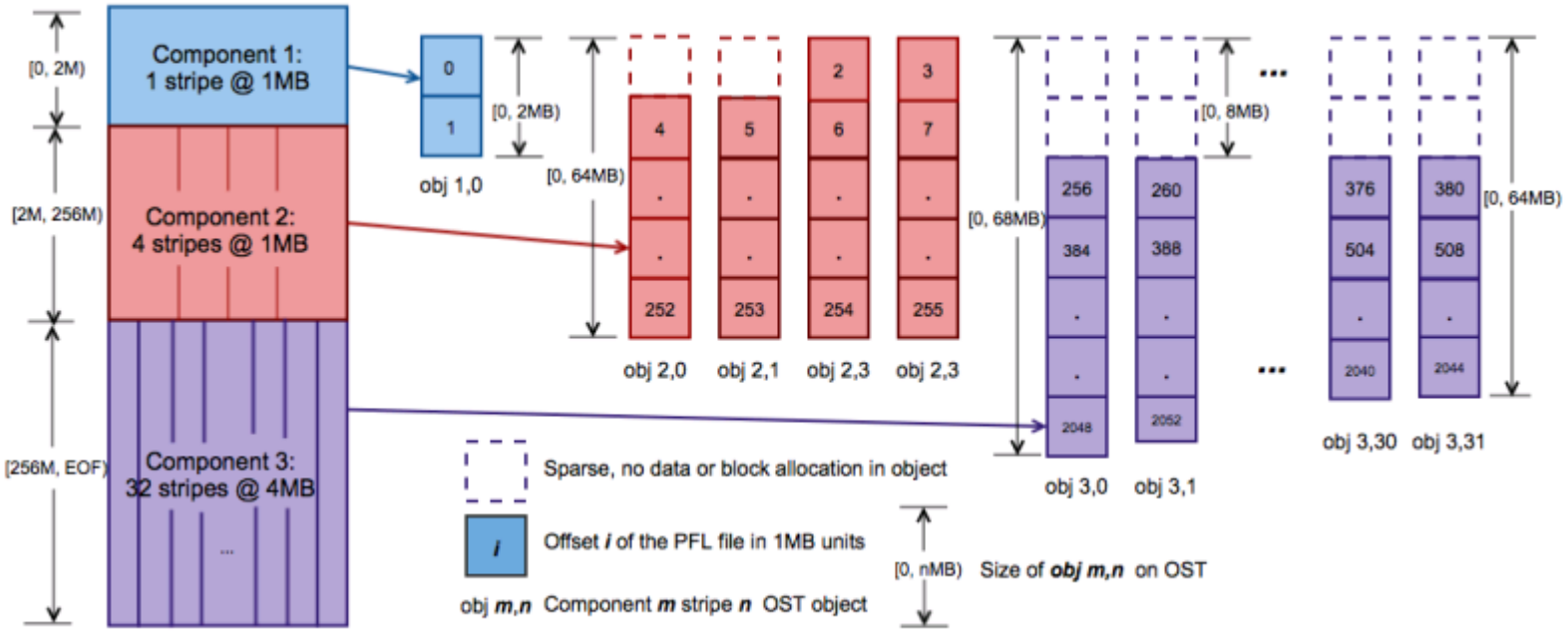
Some more commands

- Create a file with 1M stripe size and -1 (use all OSTs) stripe count
- [root@ntro-team01-node3-1 lustre]# lfs setstripe -S 1M -c -1 /mnt/lustre/file_ss1M_sc2
- [root@ntro-team01-node3-1 lustre]# lfs getstripe /mnt/lustre/file_ss1M_sc2
  - /mnt/lustre/file_ss1M_sc2
  - lmm_stripe_count: 2
  - lmm_stripe_size: 1048576
  - lmm_pattern: raid0
  - lmm_layout_gen: 0
  - lmm_stripe_offset: 1
  - obdidx objid objid group
  - 1 4 0x4 0
  - 0 36 0x24 0

Progressive File Layout (PFL)

- The layout of a PFL file is stored on disk a composite layout. A PFL file is essentially an array of sub-layout components, with each sub-layout component being a plai layout covering different and non-overlapped extents of the file.
- Example
  - 3 components
  - stripe size for the first two components is 1MB,
  - stripe size for the third componen is 4MB.
  - first component only has two 1ME blocks and the single object has a size of 2MB.
  - The second component holds the next 254MB of the file spread ove 4 separate OST objects in RAID-0, each one will have a size of 256MB / 4 objects = 64MB per object.
  - The final component holds the next 1800MB spread over 32 OST objects

Figure 19.1. PFL object mapping diagram

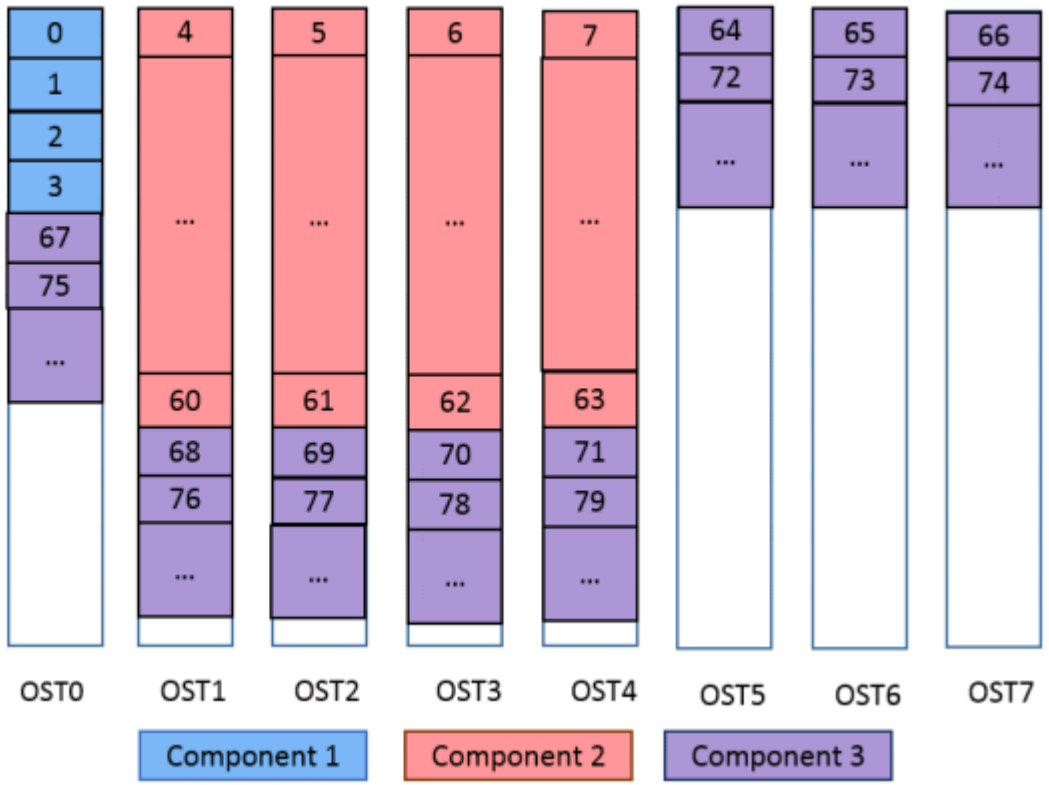


Mapping from 2055MB PFL file data blocks to OST objects of three components

PFL

```
$ lfs setstripe -E 4M -c 1 -E 64M -c 4 -E -1 -c -1 -i 4 \
/mnt/testfs/create_comp
```

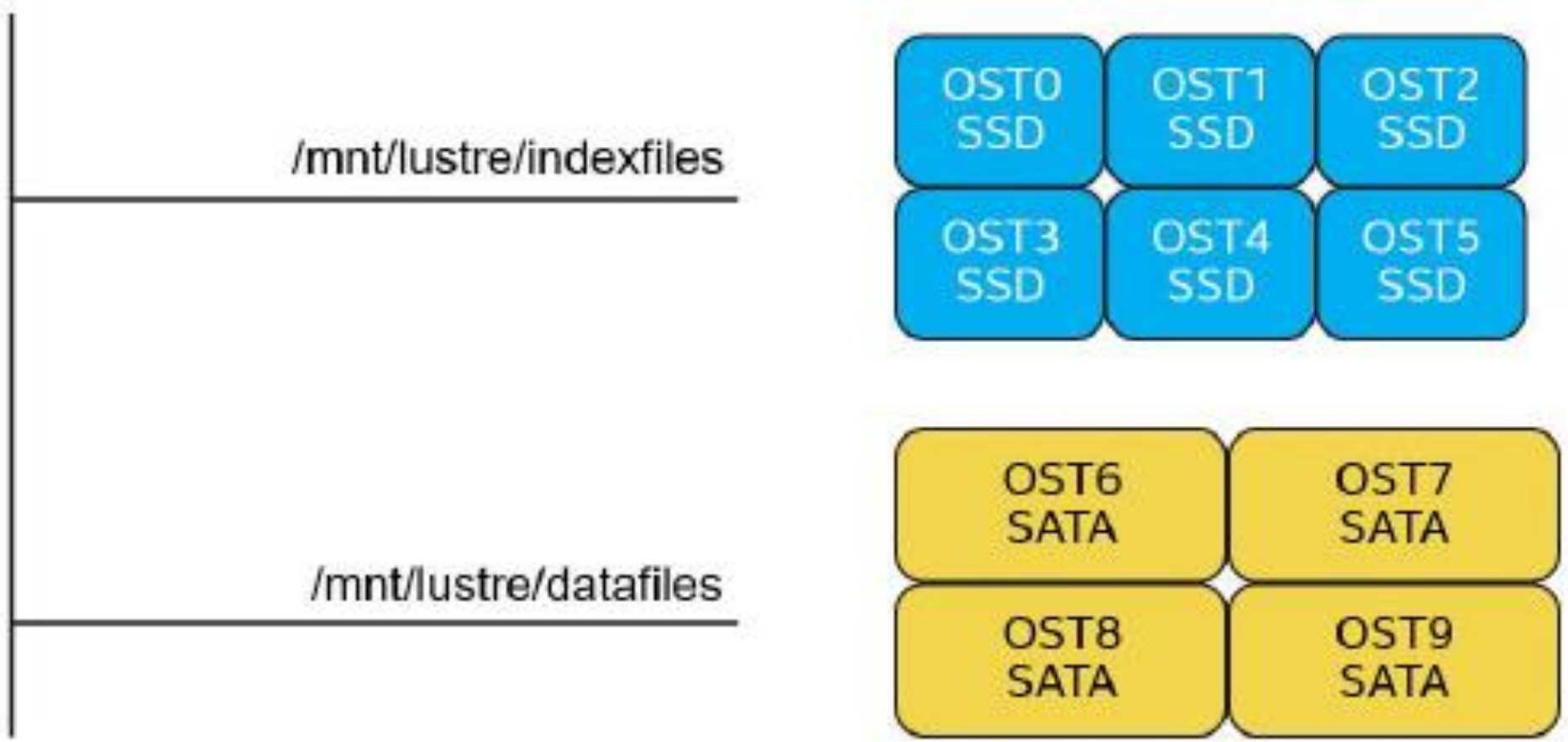
Figure 19.2. Example: create a composite file



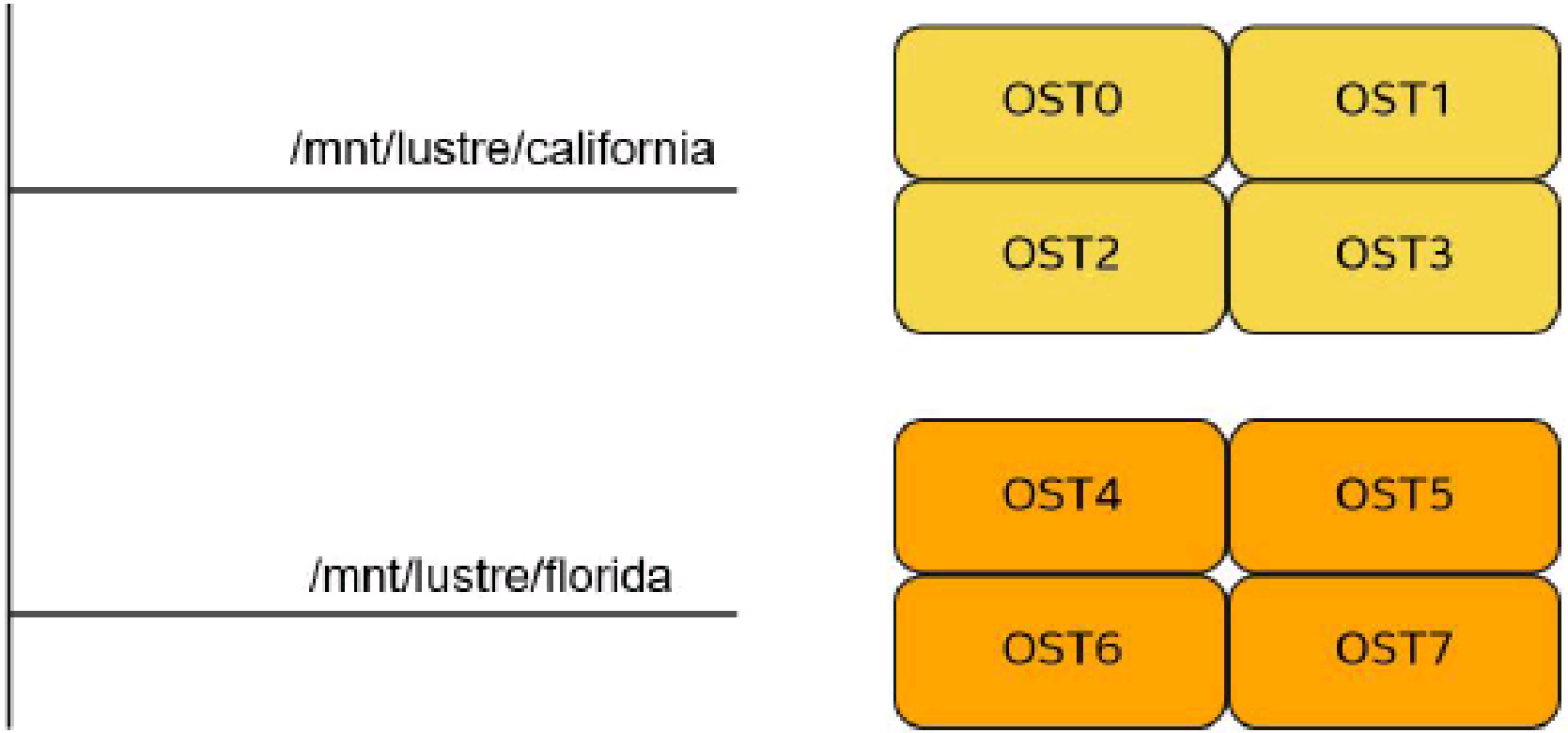
OST Pools

- Pooling OSTs provides powerful functionality
  - Restrict end users to a subset of OSTs with proper configuration/directory permissions
  - Provides different SLAs to different parts of the file system
  - Subdivide a Lustre file system into virtual Lustre file systems
- OST pools are not commonly used
  - Have existed for years but less frequently implemented
- Why not?
  - Lustre most frequently used for HPC deployments – not Enterprise
  - Pooling OSTs not needed – Goal is large, shared storage
- How might this change in near future?
  - Traditional HPC vs emerging markets

Storage Pools Example – File Types



Storage Pools Example - Locations



Storage Pools – Using OST Pools to Enforce Isolation

Example 1: File system with two new OST pools added

```
root group0 rwxrwxr-- /mnt/lustre
root group1 rwxrwxr-- /mnt/lustre/dir-for-pool-1
root group2 rwxrwxr-- /mnt/lustre/dir-for-pool-2
```

- Will *NOT* work as might be expected - group0 can write to all OSTs

Example 2: File system with two new OST pools added

```
root group0 rwxrwxr-- /mnt/lustre/dir-for-pool-0
root group1 rwxrwxr-- /mnt/lustre/dir-for-pool-1
root group2 rwxrwxr-- /mnt/lustre/dir-for-pool-2
```

Therefore, to enforce FULL isolation, must FULLY use OST Pools

New files created in subdirectories will use the pool by default

Creating an OST Pool

Create a pool with: `lctl pool_new`

Example:

```
# lctl pool_new
add a new pool
usage pool_new <fsname>.<poolname>

# lctl pool_new myfs.mypool
Pool myfs.mypool created

# lctl pool_list myfs
Pools from myfs:
myfs.mypool
```

Adding OSTs to a Pool

Add OSTs to a pool with: `lctl pool_add`

Examples:

```
# lctl pool_add myfs.mypool myfs-OST0000
OST myfs-OST0000_UUID added to pool myfs.mypool

# lctl pool_list myfs.mypool
Pool: myfs.mypool
myfs-OST0000_UUID

# lctl pool_add myfs.mypool myfs-OST0001 myfs-OST0002
OST myfs-OST0001_UUID added to pool myfs.mypool
OST myfs-OST0002_UUID added to pool myfs.mypool

# lctl pool_list myfs.mypool
Pool: myfs.mypool
myfs-OST0000_UUID
myfs-OST0001_UUID
myfs-OST0002_UUID
```

Removing OSTs from a Pool

Remove OSTs with: `lctl pool_remove`

Example:

```
# lctl pool_remove myfs.mypool myfs-OST0001
OST myfs-OST0001_UUID removed from pool myfs.mypool

# lctl pool_list myfs.mypool
Pool: myfs.mypool
myfs-OST0000_UUID
myfs-OST0002_UUID
```

Removing an OST Pool

Remove a pool with: `lctl pool_destroy`

Note that the pool must not contain any OSTs

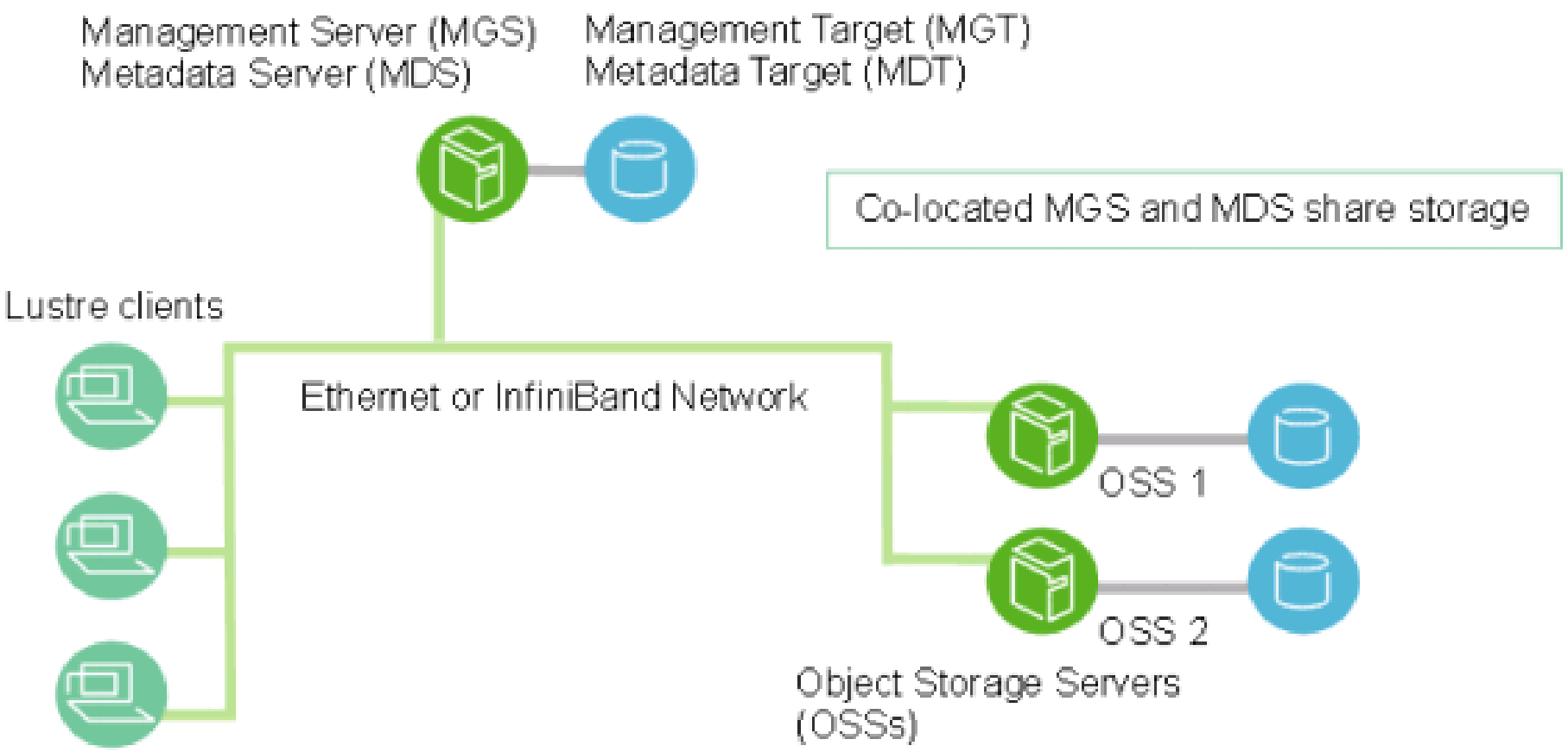
Example:

```
# lctl pool_destroy myfs.mypool
Pool myfs.mypool not empty, please remove all members
pool_destroy: Directory not empty

# lctl pool_remove myfs.mypool myfs-OST0000 myfs-OST0002
OST myfs-OST0000_UUID removed from pool myfs.mypool
OST myfs-OST0002_UUID removed from pool myfs.mypool

# lctl pool_destroy myfs.mypool
Pool myfs.mypool destroyed

# lctl pool_list myfs
Pools from myfs:
```

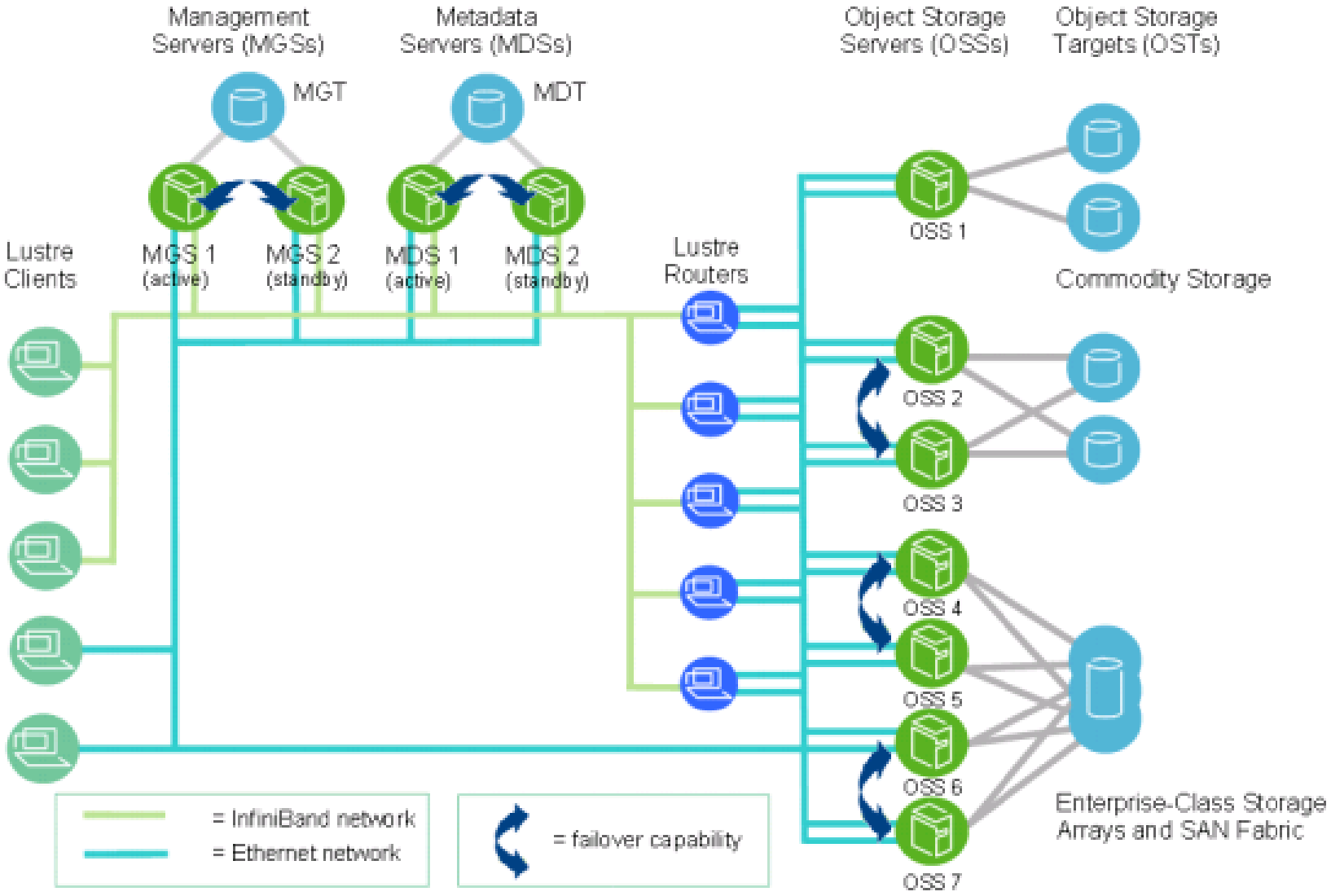
lfs migrate

- Lfs migrate commands are used to re-layout the data in the existing files with the new layout parameter by copying the data from the existing OST(s) to the new OST(s)
- The difference between migrate and setstripe is that migrate is to re-layout the data in the existing files, while setstripe is to create new files with the specified layout

Simple Configuration



Larger Configuration



HA and Failover

- High-Availability (HA) system
  - Reduce unscheduled downtime by using redundant hardware and software components that automate recovery when a failure occurs.
- Failover
  - Availability is accomplished by replicating hardware and/or software so that when a primary server fails or is unavailable, a standby server can be switched into its place to run applications and associated resources. This process, called failover, is automatic in an HA system and, in most cases, completely application transparent.

HA and Failover

- Failover capabilities
  - Resource fencing
    - Protects physical storage from simultaneous access by two nodes.
  - Resource management
    - Starts and stops the Lustre resources as a part of failover, maintains the cluster state, and carries out other resource management tasks.
  - Health monitoring
    - Verifies the availability
- HA Software
  - HA software is responsible for detecting failure of the primary Lustre server node and controlling the failover. The Lustre software works with any HA software that includes resource (I/O) fencing.

HA and Failover

- Types of Failover Configurations
  - Active/Passive pair
    - The active node provides resources and serves data, while the passive node is usually standing by idle.
    - If the active node fails, the passive node takes over and becomes active.
  - Active/Active pair
    - Both nodes are active, each providing a subset of resources.
    - In case of a failure, the second node takes over resources from the failed node

HA and Failover

- MDT Failover Configuration
  - Active/Passive
  - Active/Active

Figure 3.1. Lustre failover configuration for a active/passive MDT

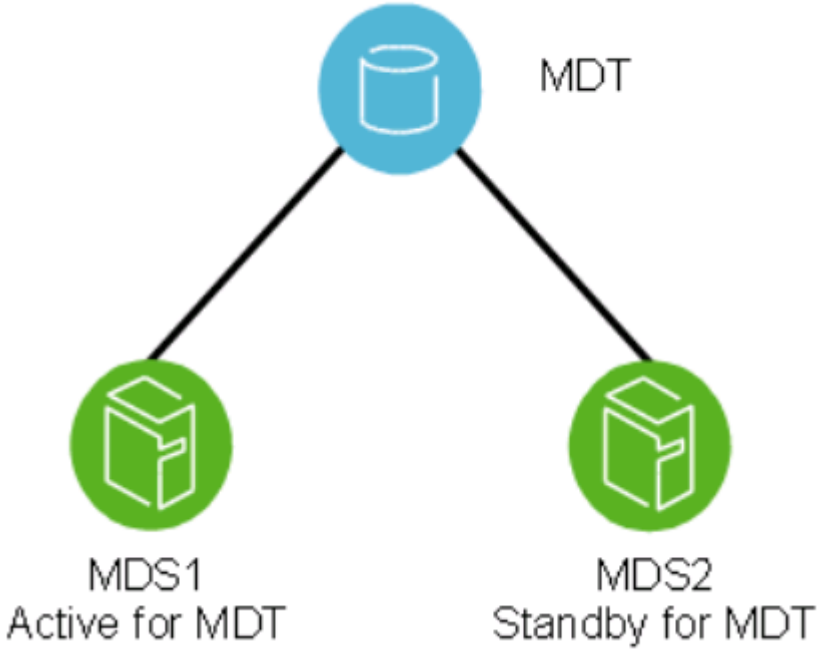
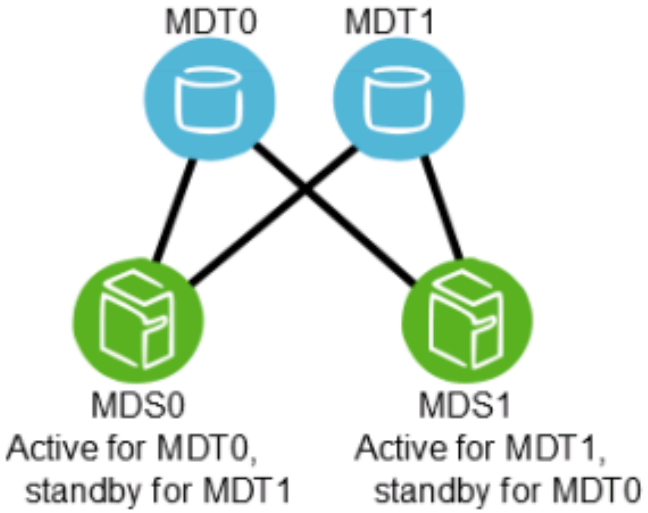


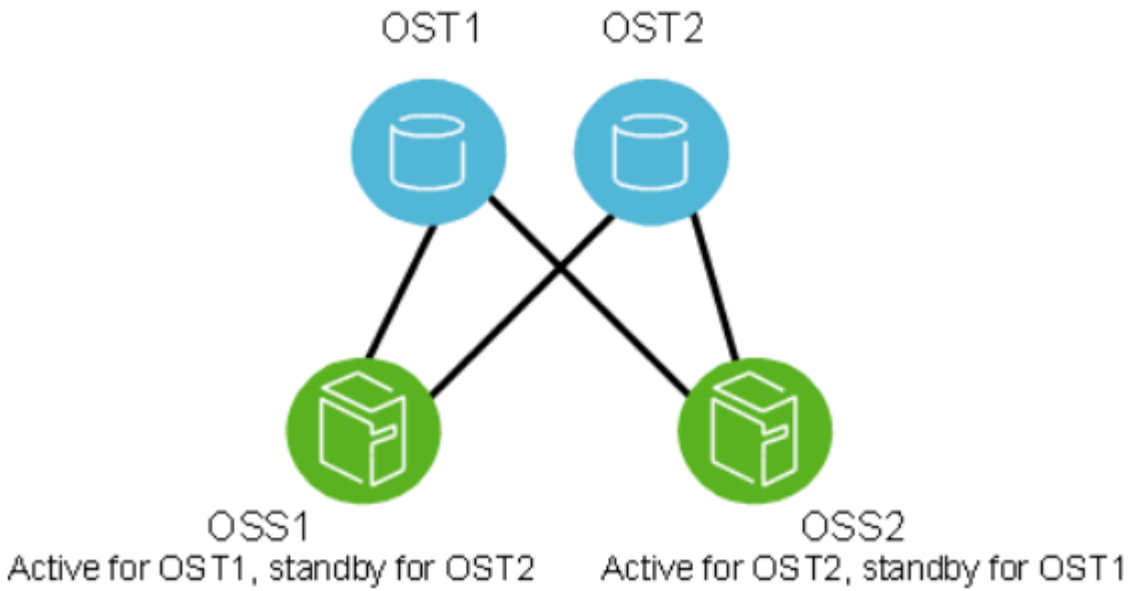
Figure 3.2. Lustre failover configuration for a active/active MDTs



HA and Failover

- OST Failover Configuration
  - Active/Active

Figure 3.3. Lustre failover configuration for an OSTs



HA and Failover

- Preparing Lustre file system for Failover using “—servicenode” option
  - mkfs.lustre with “—servicenode=“ option
    - mkfs.lustre --reformat --ost --fsname testfs --mgsnode=192.168.10.1@o3ib --index=0 --servicenode=192.168.10.7@o2ib --servicenode=192.168.10.8@o2ib /dev/sdb
  - (More than two servicenodes can be specified)
  - Whichever servicenode mounts the target first becomes the primary node for the target.
  - Client is informed about the servicenodes (via NIDs of the servicenodes) at the time of mount
    - mount -t lustre 10.10.120.1@tcp1:10.10.120.2@tcp1:/testfs /lustre/testfs

HA and Failover

- Preparing Lustre file system for Failover using “—failnode” option
  - Previous to Lustre 2.0, the –failnode option to mkfs.lustre was used to designate a failover service node for a primary server for a target
  - With “—failnode” option, certain restrictions apply
    - The target must be initially mounted on the primary service node, not the failover node designated by the --failnode option.
    - If a --failnode option is added to a target to designate a failover server for the target, the target must be re-mounted on the primary node before the --failnode option takes effect

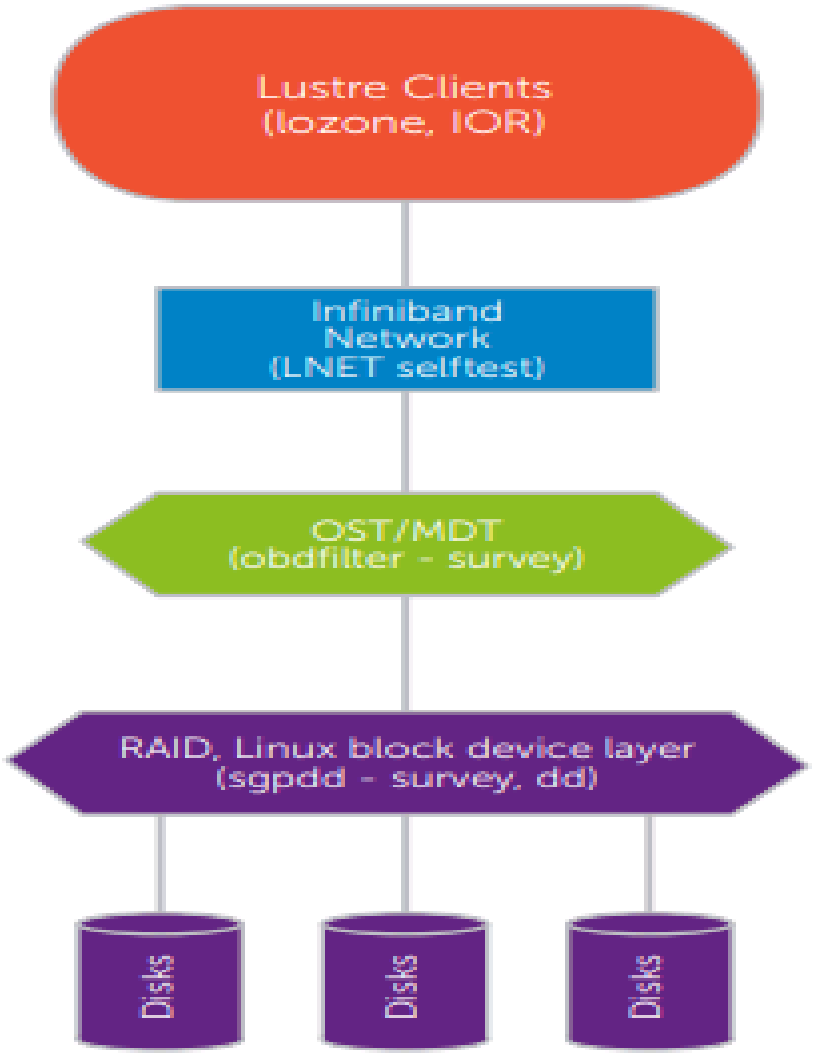
Lustre Benchmarking

- Lustre I/O Kit
- Benchmarking Stack
- sgpdd_survey
- obdfilter_survey
- ost_survey
- mds_survey
- IOR

Lustre I/O Kit

- Lustre I/O kit contains three tests, each of which tests a progressively higher layer in the Lustre software stack
  - sgpdd-survey
    - Measure basic 'bare metal' performance of devices while bypassing the kernel block device layers, buffer cache, and file system.
  - obdfilter-survey
    - Measure the performance of one or more OSTs directly on the OSS node or alternately over the network from a Lustre client.
  - ost-survey
    - Performs I/O against OSTs individually to allow performance comparisons to detect if an OST is performing sub-optimally due to hardware issues.

Benchmarking Stack

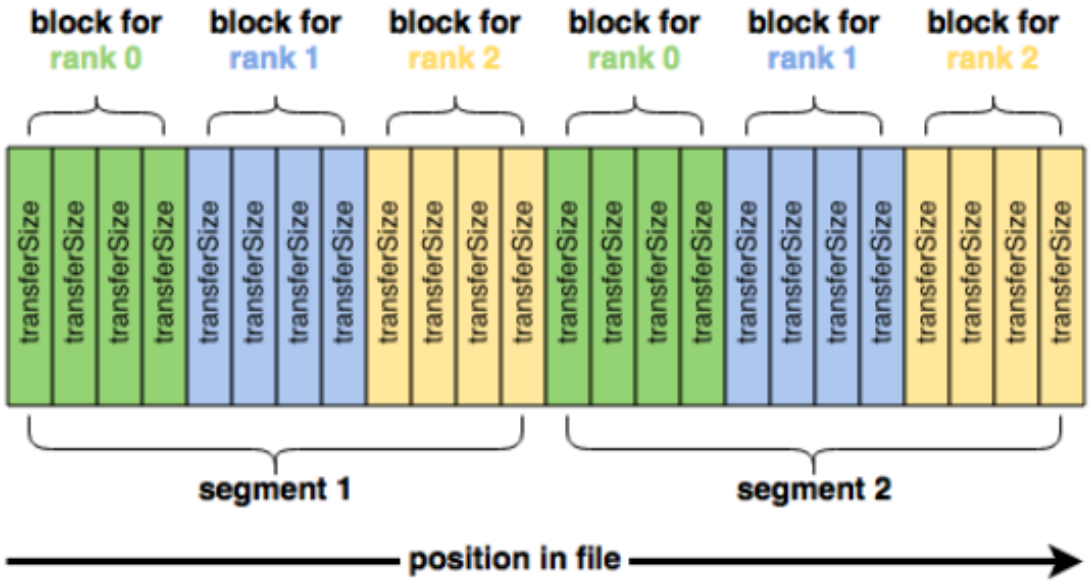


IOR

- Within IOR, one can configure the benchmark for File-Per-Process, and Single-Shared-File
  - File-Per-Process: Creates a unique file per task and most common way to measure peak throughput of a Lustre Parallel Filesystem
  - Single-Shared-File: Creates a Single File across all tasks running on all clients
- Two primary modes for IOR
  - Buffered: This is default and takes advantage of Linux page caches on the Client
    DirectIO: Bypasses Linux page caching and writes directly to the filesystem

IOR

- IOR writes data sequentially with the following parameters:
  - blockSize (-b)
  - transferSize (-t)
  - segmentCount (-s)
  - numTasks (-n)
- These four parameters are all you need to get started with IOR.
- If we run a four-node IOR test that writes a total of 16 GiB
  - Writing to a single shared-file
    - $ ior –t 1m –b 16m –s 16
  - Writing to one file per processes
    - $ ior –t 1m –b 16m –s 16 –F
  - ./src/ior -v -a MPIIO -w -r -i 4 -o ior-test.file -t 262144 -b 1m -O "lustreStripeCount=-1"



SAN

- A SAN is a computer network which provides access to consolidated, block-level data storage.
- SANs are primarily used to enhance accessibility of storage devices, such as disk arrays and tape libraries, to servers so that the devices appear to the operating system as locally-attached devices.
- A SAN typically is a dedicated network of storage devices not accessible through the local area network (LAN) by other devices, thereby preventing interference of LAN traffic in data transfer.
- A SAN does not provide file abstraction, only block-level operations. However, file systems built on top of SANs do provide file-level access, and are known as shared-disk file systems.

Shared Disk File System

- A shared-disk file system uses a SAN to allow multiple computers to gain direct disk access at the block level.
- Access control and translation from file-level operations that applications use to block-level operations used by the SAN must take place on the client node.
- Shared-disk file system provides a consistent and serializable view of the file system (by adding mechanisms for concurrency control) , avoiding corruption and unintended data loss even when multiple clients try to access the same files at the same time.
- Shared-disk file-systems commonly employ some sort of fencing mechanism to prevent data corruption in case of node failures, because an unfenced device can cause data corruption if it loses communication with its sister nodes and tries to access the same information other nodes are accessing.
- The underlying storage area network may use any of a number of block-level protocols, including SCSI, iSCSI, HyperSCSI, ATA over Ethernet (AoE), Fibre Channel, network block device, and InfiniBand.

GPFS

- IBM Spectrum Scale is a cluster file system that provides concurrent access to a single file system
- The nodes can be
  - SAN attached,
  - network attached,
  - a mixture of SAN attached and network attached, or
  - in a shared nothing cluster configuration.
- This enables high performance access to this common set of data to support a scale-out solution or to provide a high availability platform.

Simplified Storage Management

- IBM Spectrum Scale provides storage management based on the definition and use of
  - Storage pools
    - A storage pool is a collection of disks or RAIDs with similar properties that are managed together as a group.
    - Storage pools provide a method to partition storage within a file system
  - Policies
    - Files are assigned to a storage pool based on defined policies.
      - Placing files in a specific storage pool when the files are created (Placement policies)
      - Migrating files from one storage pool to another (File management policies)
      - Deleting files based on file characteristics (File management policies)
  - Filesets
    - Filesets provide a method for partitioning a file system and allow administrative operations at a finer granularity than the entire file system. For example, filesets allow you to:
      - Define data block and inode quotas at the fileset level
      - Apply policy rules to specific filesets
      - Create snapshots at the fileset level

Basic Spectrum Scale Structure

- IBM Spectrum Scale is a clustered file system defined over one or more nodes.
- On each node in the cluster, IBM Spectrum Scale consists of three basic components
  - administration commands
  - a kernel extension, and
  - a multithreaded daemon.

GPFS Administrative Commands

- GPFS administration commands are scripts and programs that control the operation and configuration of GPFS.
- By default, GPFS commands can be executed from any node in the cluster.
- If tasks need to be done on another node in the cluster, the command automatically redirects the request to the appropriate node for execution.
- For administration commands to be able to operate, passwordless remote shell communications between the nodes is required

GPFS Kernel Extension

- The GPFS kernel extension provides the interfaces to the operating system vnode and virtual file system (VFS) layer to register GPFS as a native file system.

GPFS Daemon

- The GPFS daemon performs all I/O operations and buffer management for GPFS.
- This includes read-ahead for sequential reads and write-behind for all writes not specified as synchronous.
- I/O operations are protected by GPFS token management, which ensures consistency of data across al
- The daemon is a multithreaded process with some threads dedicated to specific functions.
- Dedicated threads for services requiring priority attention are not used for or blocked by routine work.
- In addition to managing local I/O, the daemon also communicates with instances of the daemon on other nodes to coordinate configuration changes, recovery, and parallel updates of the same data structures.

GPFS Network Shared Disk (NSD)

- The GPFS Network Shared Disk (NSD) component provides a method for cluster-wide disk naming and high-speed access to data for applications running on nodes that do not have direct access to the disks.
- The NSDs in your cluster may be physically attached to all nodes or serve their data through an NSD server that provides a virtual connection.
- You are allowed to specify up to eight NSD servers for each NSD.
- If one server fails, the next server on the list takes control from the failed node.
- For a given NSD, each of its NSD servers must have physical access to the same NSD.
- However, different servers can serve I/O to different non-intersecting sets of clients.
- The existing subnet functions in GPFS determine which NSD server should serve a particular GPFS client.
- NSD Discovery
  - GPFS determines if a node has physical or virtual connectivity to an underlying NSD through a sequence of commands that are invoked from the GPFS daemon. This determination, which is called NSD discovery, occurs at initial GPFS startup and whenever a file system is mounted.

GPFS Cluster Configurations

- An IBM Spectrum Scale cluster can be configured in a variety of ways. The cluster can be a heterogeneous mix of hardware platforms and operating systems.
- Four basic IBM Spectrum Scale Configurations
  - All nodes attached to a common set of LUNS
  - Some nodes are NSD clients
  - A cluster is spread across multiple sites
  - Data is shared between clusters

PVFS

- Parallel Virtual File System
- Grew out of research project from 1993 (by Walt Ligon and Eric Blumer)
- Since 2008, the main development branch is called OrangeFS
- In 2016, it became part of Linux kernel

Features

- Object-based design
  - All PVFS server requests involved objects called dataspaces
  - A dataspace can be used to hold:
    - File data
    - File metadata
    - Directory metadata
    - Directory entries
    - Symbolic links
  - Every dataspace in the file system has a unique handle
  - A dataspace has two components:
    - Bytestream, typically used to hold file data
    - Set of key/value pairs, typically used to hold metadata

Features

- Separation of data and metadata
  - Client can access a server for metadata once
  - Then access data server without further interaction with the metadata server
  - Removes a critical bottleneck from the system
- MPI-based requests
  - Client program requests data from PVFS it can supply a description of the data that is based on MPI_Datatypes
- Multiple network support
  - PVFS uses a networking layer named Buffer Message Interface (BMI) which provides a non-blocking message interface designed specifically for file systems.
  - BMI has multiple implementation modules for a number of different networks used in high performance computing including TCP/IP, Myrinet, Infiniband, and Portals.
- Stateless
  - Server does not share any state with each other or with clients. If a server crashes, another can be started in its place.
  - Update are performed without using locks.

Features

- User-level implementation
  - Clients and servers are running at user level
  - It has an optional kernel module to allow a PVFS to be mounted like any other file system
  - Or, programs can be linked directly to user interface like MPI-IO and Posix-like interface
  - Makes it easy to install and less prone to causing system crashes
- System-level interface
  - PVFS interface is designed to integrate at the system level
  - It exposes many of the features of the underlying file system so that interfaces can take advantage of them if desired
  - It is similar to the Linux VFS, making it easy to implement as a mountable file system.

Architecture

- There are four major components to the PVFS system
  - Metadata server (MGR)
  - I/O server (ION or Iod)
  - PVFS native API (libpvfs on CN)
  - PVFS Linux kernel support