



Cluster Storage and File Systems Technology

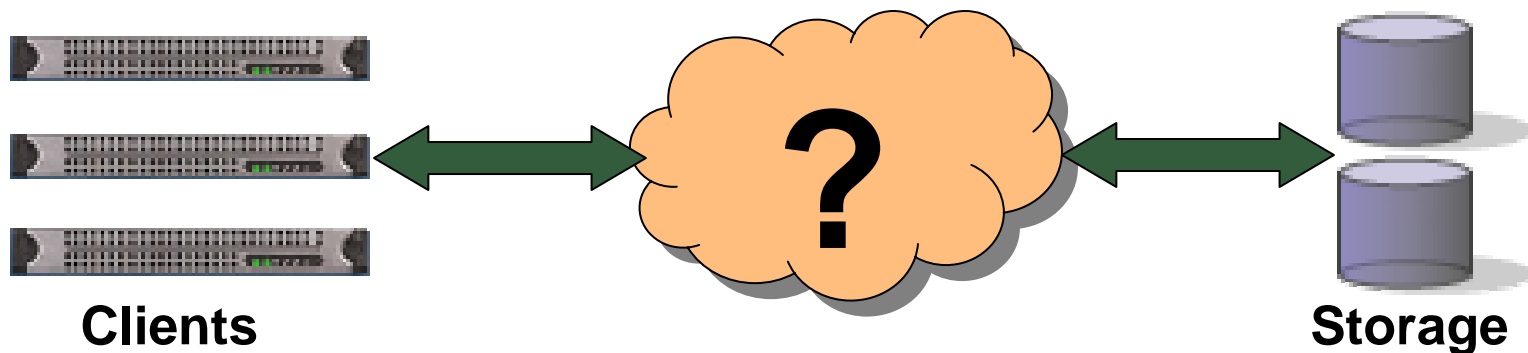
SC06

Brent Welch , Marc Unangst
{welch,mju}@panasas.com
Panasas, Inc



Cluster Storage Problem Statement

- Compute clusters are growing larger in size (8, 128, 1024, 4096...)
 - Scientific codes, seismic data, digital animation, biotech, EDA ...
- Each host in the cluster needs uniform access to any stored data
- Demand for storage capacity and bandwidth is growing (GBs/sec)
- Apply clustering techniques to the storage system itself
- Maintain simplicity of storage management even at large scale



Outline

- Distributed file system architectures
 - NAS, SAN, Clustered NAS, Object Storage
 - Parallel NFS (pNFS in NFSv4.1)
- Details
 - Object Storage Devices (OSD)
 - RAID taxonomy, block-based, file-based, declustering
 - Metadata management
 - Scaling and Performance
 - Security
- Along the way we'll classify existing products
 - ADIC, CXFS, GFS, GPFS, TerraScale, Exanet, PolyServe, Isilon, IBRIX, Centera, HighRoad, Lustre, Panasas, and others

About Us

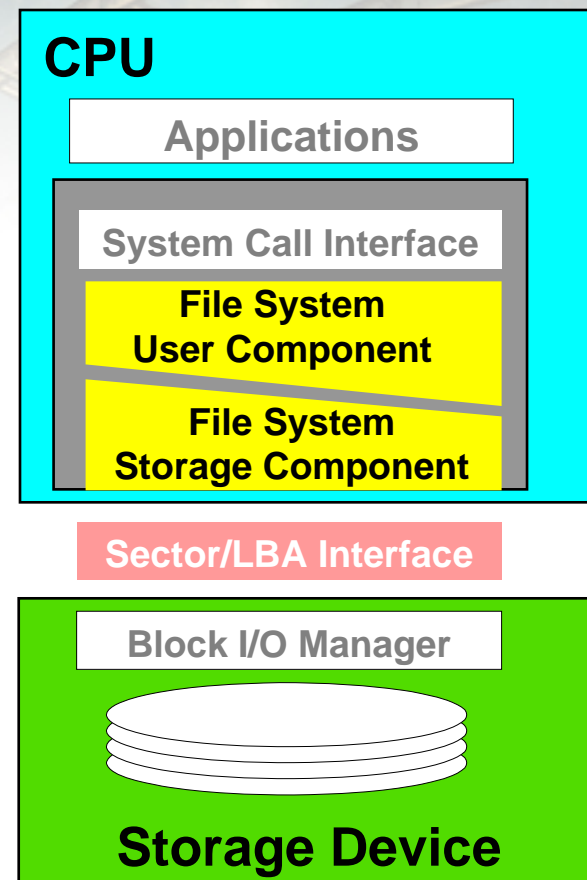
- Brent Welch (welch@panasas.com)
 - Director of Architecture, Panasas
 - Berkeley Sprite OS Distributed File System
 - Panasas Distributed File System
 - IETF pNFS
- Marc Unangst (mju@panasas.com)
 - Software Architect, Panasas
 - Object Storage, Fault Tolerance, File System Protocol design
- Panasas
 - Storage Cluster Vendor
 - Top500, Oil & Gas, Digital Animation, Commercial HPC (EDA, Fluid Dynamics)
 - Founder is Garth Gibson: RAID, NASD, OSD pioneer

Local File System

- Private storage for a host operating system
- Backend storage can be DAS (SCSI, SAS) or SAN (FC, iSCSI)
- Feature differences
 - Performance
 - Snapshot support
 - Fast recovery
 - Scalability
 - Capacity, files/directory, files/file system
- Examples: ext3, reiserFS, NTFS, ufs, ffs, Sun zfs, NetApp WAFL
- This tutorial is mostly about distributed file systems
 - Problems of scale require lots of storage computers working together

File System/Storage Protocol Stack

- OS performs file system policy checks
 - User authorization
 - Permission checks (read, write, quota)
 - Maintains file-level attributes
 - Physical size, logical length, timestamps (last access time, last modified time)
- File System translates to blocks
 - OS is responsible for performance of storage device
 - Layout, organization of storage requests
 - Prefetching and buffer caching
 - Disk has primitive caching and prefetching algorithms
 - Based on physical layout
 - No knowledge of application behavior
- Best way to improve IO performance
 - Avoid touching the platter at all cost
 - Layout contiguous data on contiguous sectors/tracks



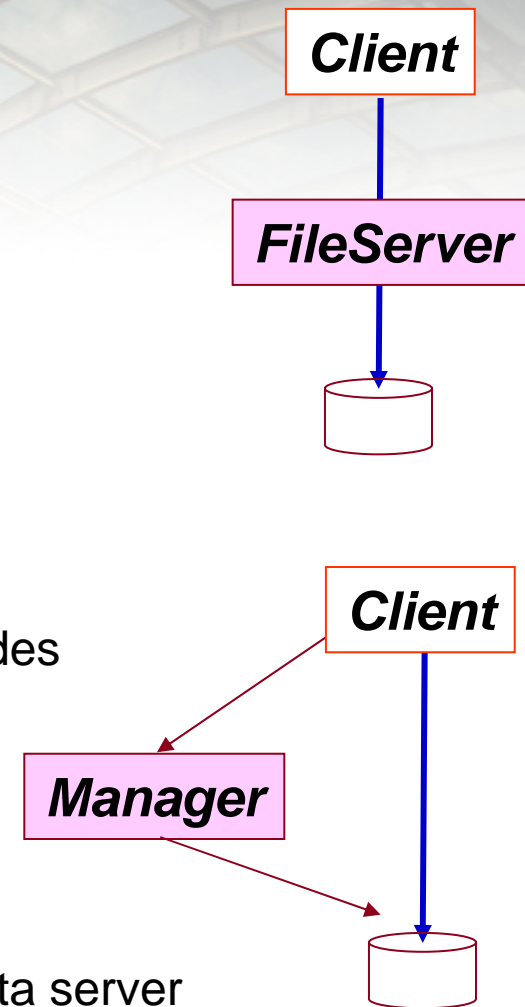
One host, one wire, one disk

Distributed File System Terminology

- SAN (Storage Area Network)
 - FiberChannel, block-based, file system is private or shared among hosts
- NAS (Network Attached Storage)
 - NFS, CIFS, File server architecture
- Symmetric File System
 - All nodes provide all services (client and server simultaneously)
- Asymmetric File System
 - Specialized metadata, data, and client nodes
- Global File System
 - Uniform naming, especially in a WAN or multi-organizational application
- Cluster File System
 - Many servers working together to provide a scalable architecture
- Parallel File System
 - Parallel I/O from cluster applications to, e.g., a single file

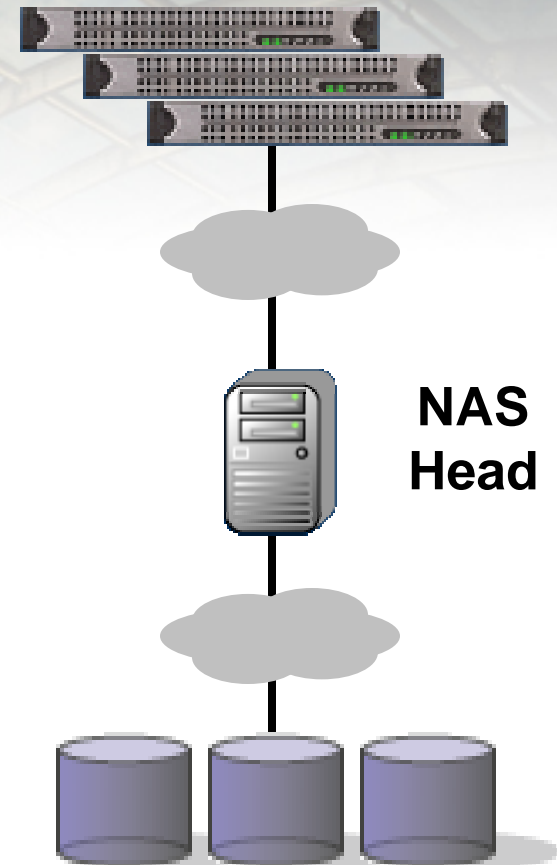
Storage Architectures: In- & Out-of-Band

- In-band
 - Data flows through server or cluster of servers
 - Server touches/manages all data and metadata
 - Easy to build and maintain at limited scale
 - Limited load balancing across head-node(s)
 - Narrow, well-defined failure modes
 - Security is function of server node
- Out-of-band
 - Data flows directly from client to storage
 - Get rid of central bandwidth bottlenecks
 - Metadata is managed off the datapath
 - Performance proportional to number of storage nodes
 - Load-balance workload across system
 - Performance and capacity
 - Security is function of storage protocol
- Symmetric
 - Many nodes that all run client, metadata server, data server



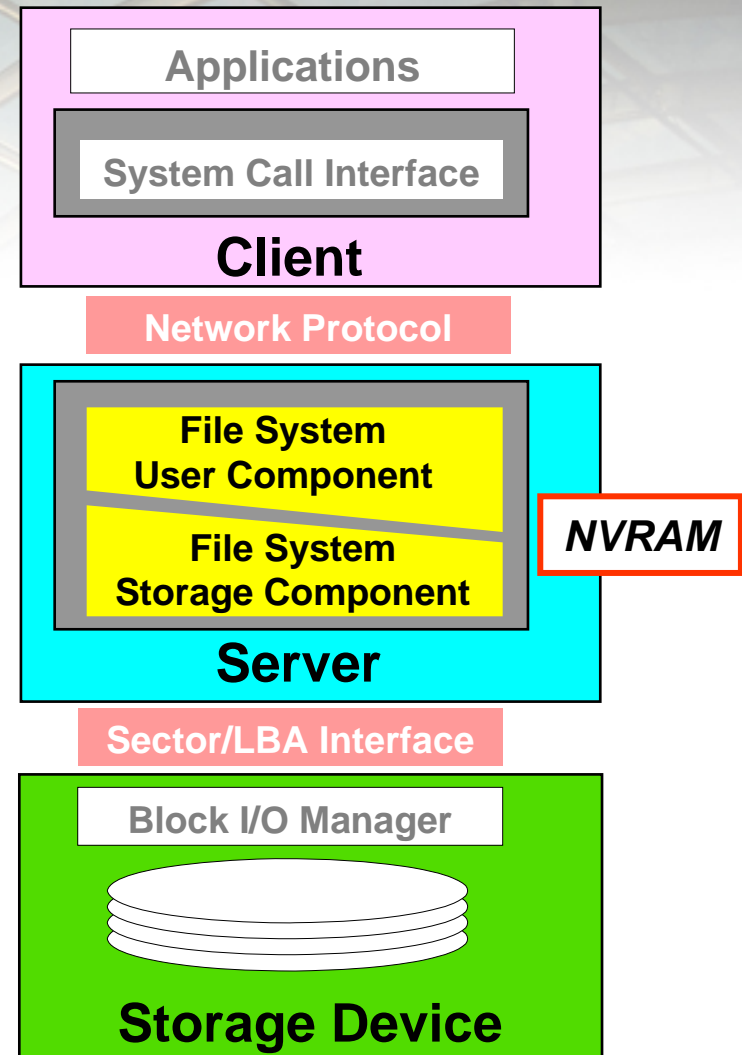
Network Attached Storage

- File Server exports storage at the file level
 - NFS/CIFS are widely deployed
 - NFS is the only official file system standard
- Scalability limited by server hardware
 - Moderate number of clients (10's to 100's)
 - Moderate amount of storage (few TB)
- A nice model until it runs out of steam
 - “Islands of storage”
 - Bandwidth to a file limited by its server
- NetApp (ONTAP 7.x), Sun, HP, SnapServer, EMC Celerra, StorEdge NAS, IBM TotalStorage NAS, whitebox Linux



File System/Storage Protocol Stack (NAS)

- NAS File system
 - High-level interface insulates the file system from change
 - Generic layer of Client modified by introduction of VFS (vnode) interface
 - Any server file system can be exported
 - Non-volatile write buffer (NVRAM) needed for good performance and crash resilience
- In-Band architecture moves data and metadata operations through the NAS server to the storage device



File Locking Protocol (NAS)

- File Locking
 - Application-level locks for serialization
 - Advisory UNIX locks (POSIX)
 - Mandatory byte-range locks (CIFS)
 - Share mode locks (CIFS)
- NAS Support
 - NFS v2 and v3 require separate network daemons for advisory POSIX locks
 - `rpc.lockd` maintains lock state
 - `rpc.statd` does heartbeat and lock recovery
 - CIFS has mandatory locks and “oplocks” for delegating locks
 - Delegated lock lets client service lock/unlock requests locally
 - NFS v4 includes advisory, mandatory, and share locking with delegation
- Various proprietary schemes layered on SAN and cluster file systems

- Not to be confused with “Distributed Lock Managers” that are internal mechanisms used in various ownership and metadata protocols, including the implementation of application-level locks

Content-Addressed Storage (CAS)

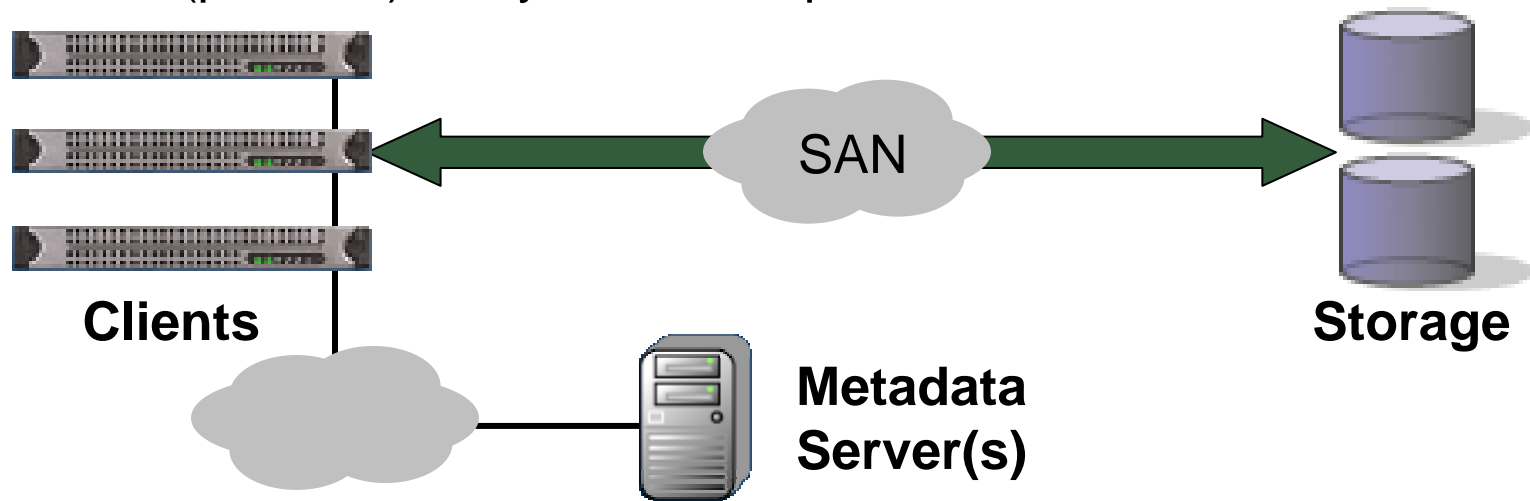
- NAS optimized for write-once, archival storage
- Metadata manager maintains cryptographic finger print of each block
 - API is “Store block, get back its unique signature”
 - Duplicate blocks/extents are not duplicated on storage
 - Metadata database assembles higher level files from lists of block signatures
- EMC Centera, Archivas, Nexsan Assureon
 - Bell Labs’ “File Motel” – Blocks check in, but they never check out
- EMC recently opened up its programming API to achieve broader acceptance of this storage model

What's this Content Stuff

- Entire BLOB (Binary Large Object) is presented to the Centera system
- Application delivers data object to Centera API which generates claim check
 - Claim check is
 - RSA MD-5 generated 256-bit signature
 - Inserted into XLM file called C-Clip Descriptor File (CDF)
 - Data blob and CDF are written (mirrored) to disk
 - CDF is also returned to application
 - Application maps CDF to actual file name
 - CF included {date, time, system, creator, project, content address, size, and file name}
 - If data changes, Claim Check will detect change
- API
 - Applications are written to new Centera Object-based Storage API
 - Standard API being proposed
 - Basic API is
 - Store, Retrieve, Exists, Delete, Query
 - Delete is controlled by protection that disallows deletion until specified time
 - Query can retrieve list of objects stored over time interval

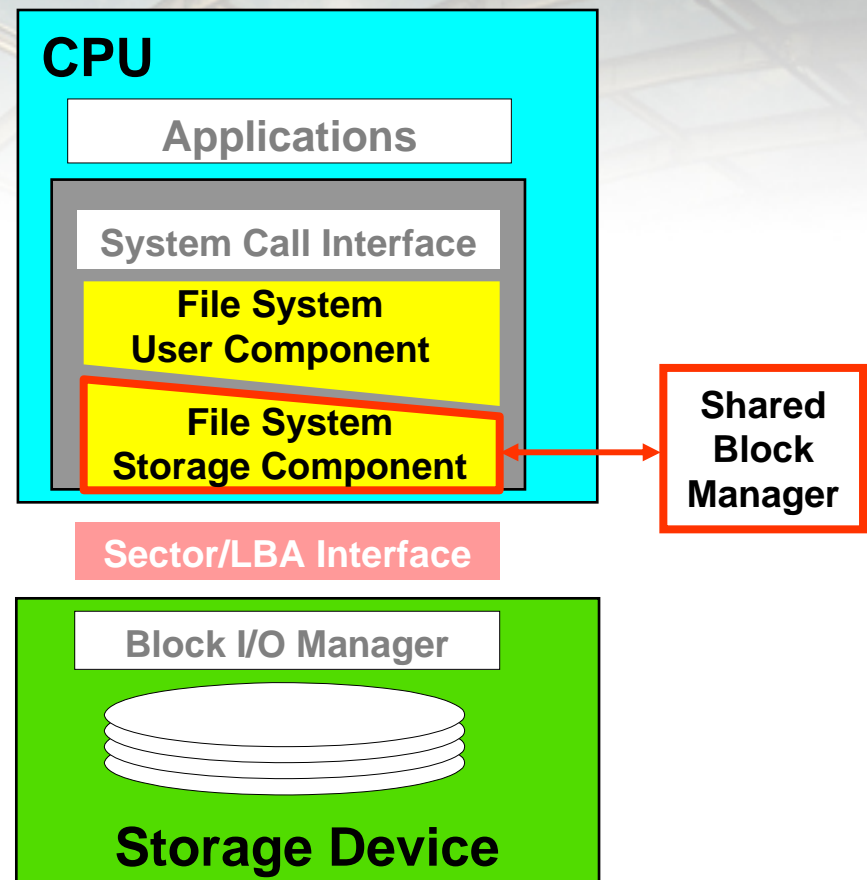
SAN File Systems

- Common management and provisioning for host storage
 - Block Devices (JBOD or RAID) accessible via iSCSI or FC network
 - Wire-speed/RAID-speed performance potential
- Proprietary solutions for shared file systems
 - Scalability limited by block management on metadata server (e.g., 32 nodes)
 - NAS access provided by “file head” that re-exports the SAN file system
- Asymmetric (pictured) or Symmetric implementations



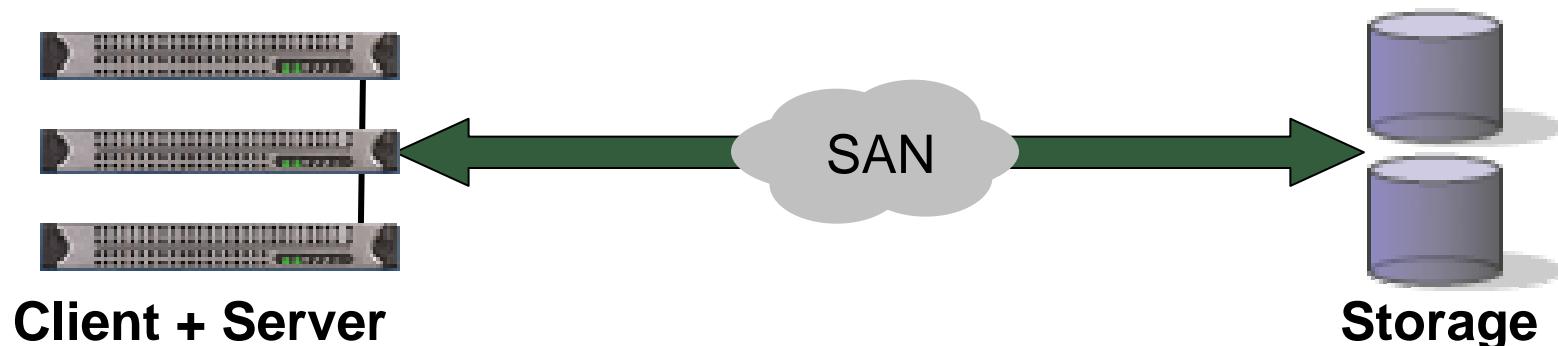
File System/Storage Protocol Stack (SAN)

- Shared SAN File system
 - Storage Layer must synchronize block-level access among hosts that share the file system
 - Narrow interface around block ownership and allocation allows “easy” modification of existing file systems
 - e.g., XFS -> cXFS, Terrascale
 - Low-level interface imposes more overhead in distributed system
 - Write() system call involves several block-level IO ops
 - Different kinds of blocks (inodes, indirect blocks, data, allocation bitmaps)
 - Billions of blocks to manage
 - 1 billion 512-byte sectors on 500 GB device. 100-billion for 50TB.



SAN File System Architectures

- Asymmetric metadata server
 - Dedicated metadata node or nodes (cluster, or active/passive)
 - Potential bottleneck
 - EMC HighRoad, IBM SAN-FS, Sun QFS, ADIC StorNext, SGI CXFS, Apple Xsan
- Symmetric metadata service
 - Every node runs client and metadata server software (cluster)
 - Inter-node communication can limit scaling
 - Red Hat GFS/Sistina, IBM GPFS, IBRIX, Terrascale/TerraGrid, Polyserve



Ownership Protocols (SAN)

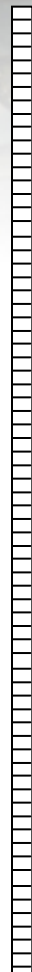
- Ownership of:
 - Free blocks
 - Inodes (i.e., files, file locks)
 - Data blocks or byte ranges
 - Directory <name,inode> maps
- Central server (Asymmetric)
 - Serializes conflicting accesses
 - May use database for metadata and ownership state
 - Scale the central server box
 - CXFS, ADIC
- Symmetric Systems
 - Nodes negotiate for ownership through a master lock owner
 - Ownership delegated and recorded persistently
 - Conflicts serialized through master
 - GPFS
- Segment Servers
 - Storage (files and blocks) owned by dedicated nodes
 - Requests forwarded to owning node
 - IBRIX

Directory

File (inode)

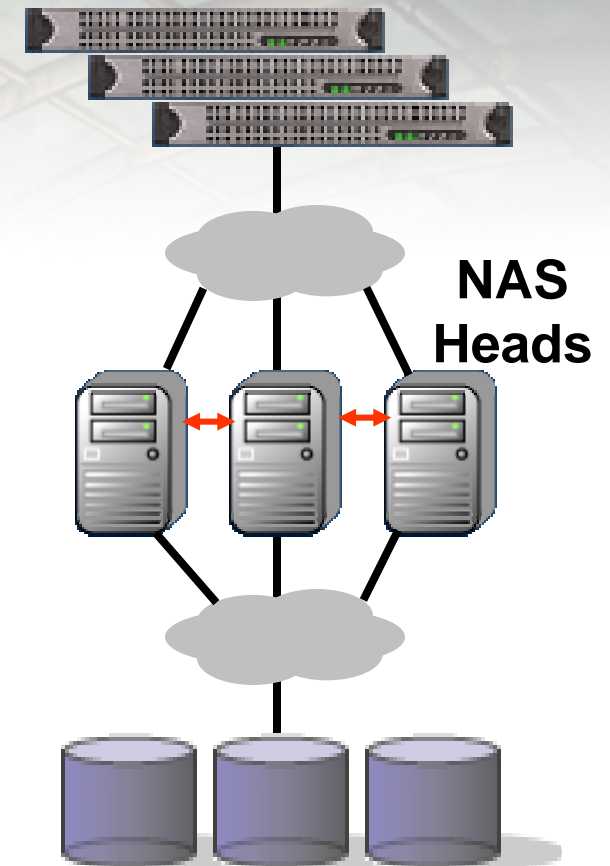
Data Blocks

Free Block Map



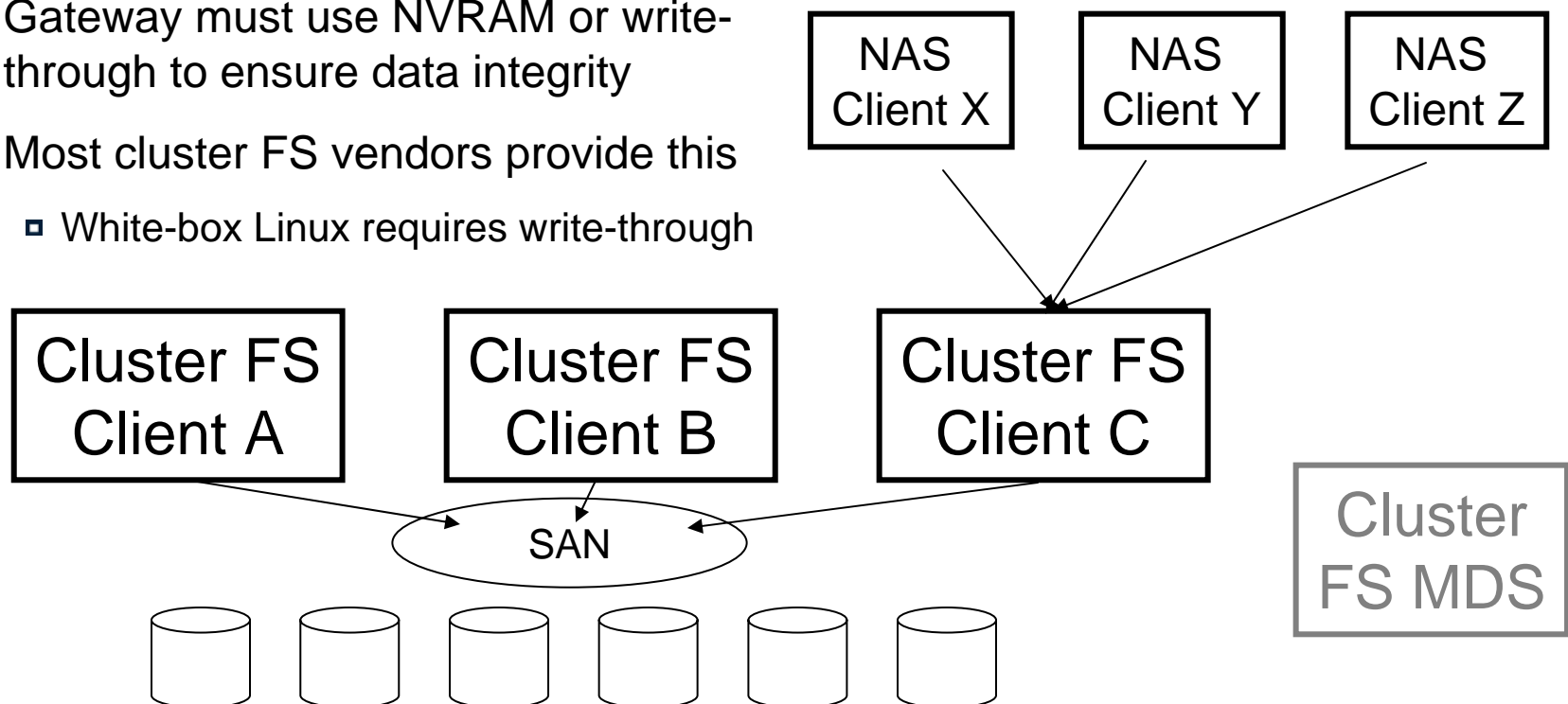
Clustered NAS

- More scalable than single-headed NAS
 - Multiple NAS heads share back-end storage
 - “In-band” NAS head still limits performance and drives up cost
- Two primary architectures
 - Forward requests to “owner Head”
 - Export NAS from clustered SAN file system
- NFS does not provide a good mechanism for dynamic load balancing
 - Clients permanently mount a particular Head
- GPFS, Isilon OneFS, IBRIX, Polyserve, NetApp-GX, BlueArc, Exanet ExaStore, ONStor, Pillar Data, IBM/Transarc AFS, IBM DFS



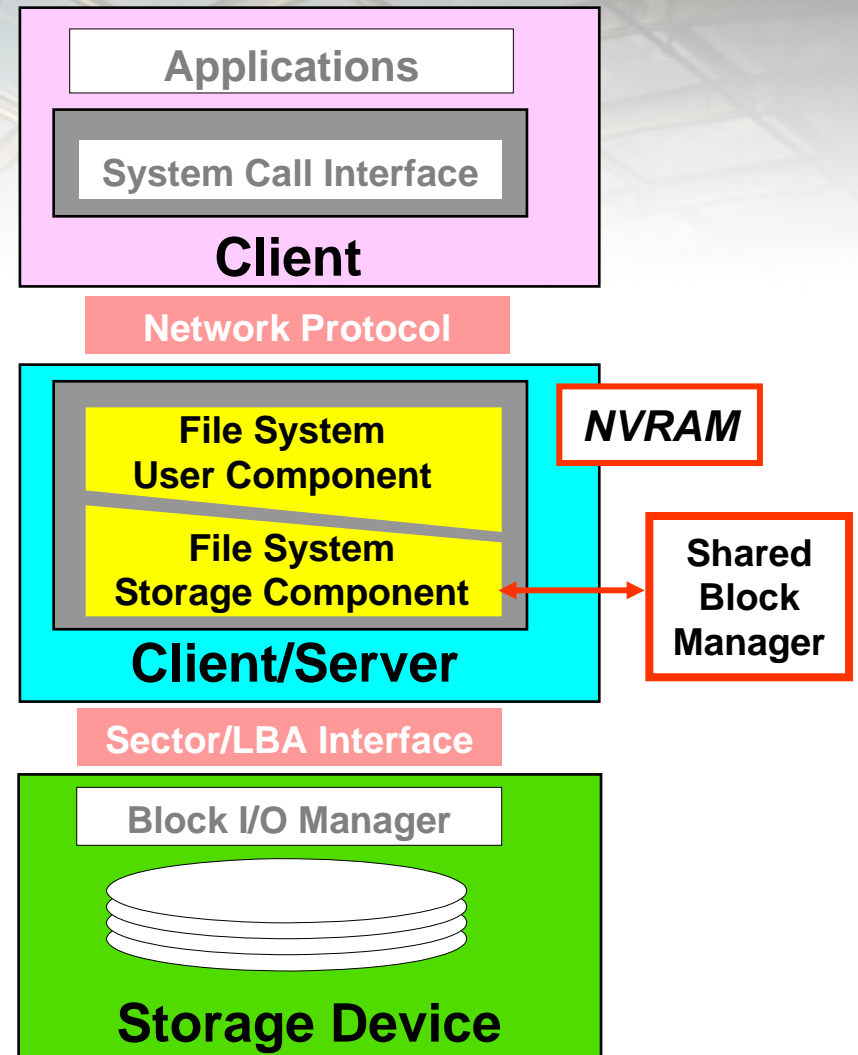
Exporting NAS from Cluster File Systems

- Export NFS from proprietary file system clients
 - Underlying cluster (Panasas, GPFS) or SAN (ADIC) file system
 - NFS/CIFS “gateway” exports NAS protocol
 - Gateway must use NVRAM or write-through to ensure data integrity
 - Most cluster FS vendors provide this
 - White-box Linux requires write-through



File System/Storage Stack (Clustered NAS)

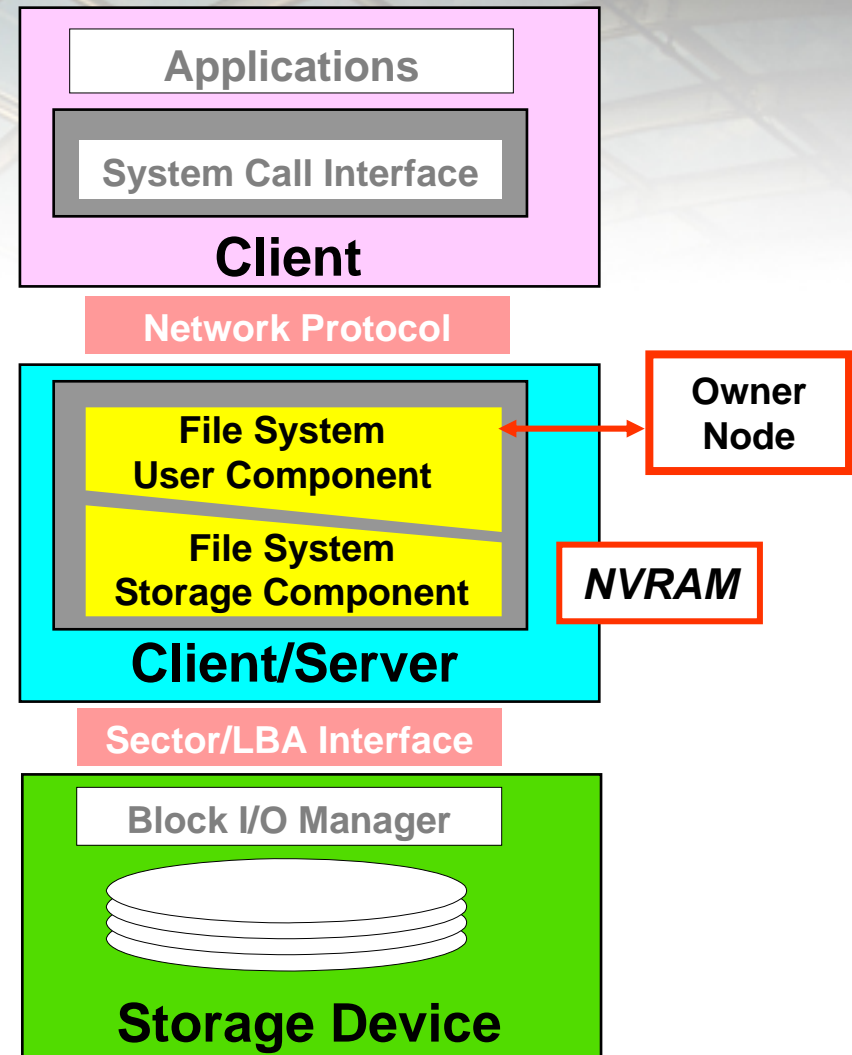
- Hybrid model
 - SAN file system clients re-export file system via network protocol
 - Same scalability limitations of SAN file system for block allocation and write traffic
 - GPFS has this model, with non-standard network protocol



(Forwarding model next slide)

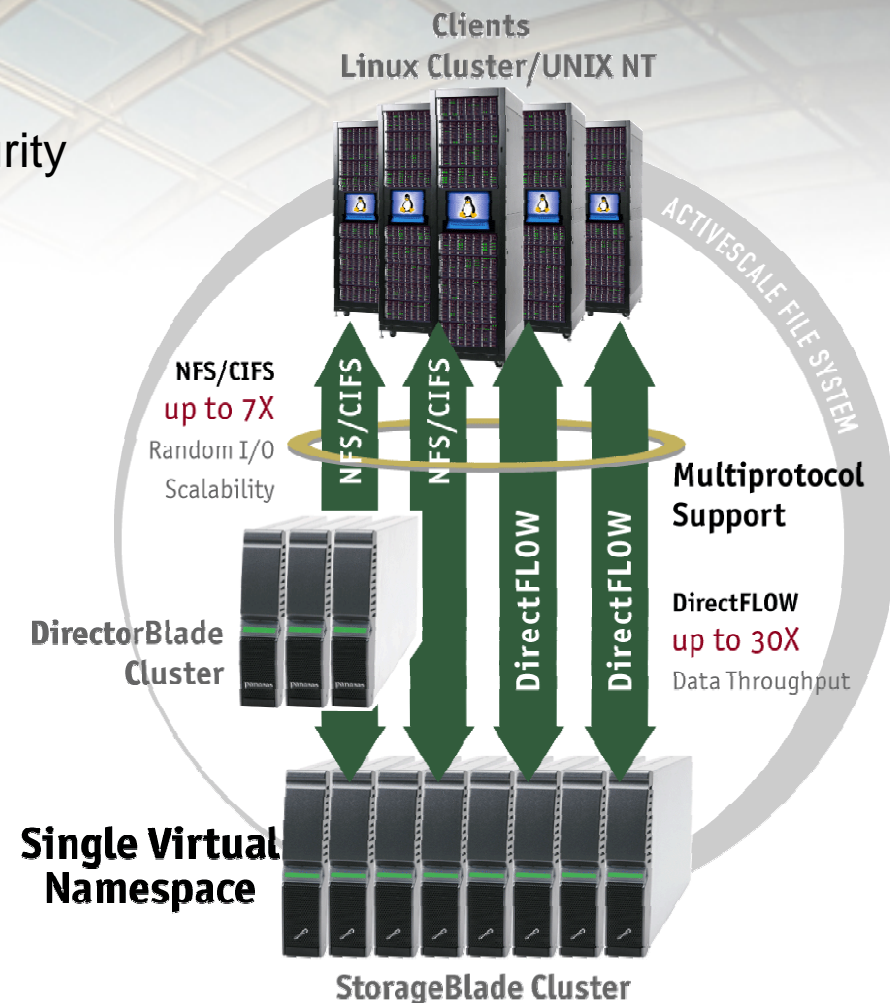
File System/Storage Stack (Clustered NAS)

- Forwarding model
 - Nodes own subsets of storage (e.g., for block allocation)
 - NAS heads have big cache, and forward operations that miss the cache to owner
 - Non-cached operations have same bottleneck as a traditional file server
 - NetApp-GX, Isilon, IBRIX



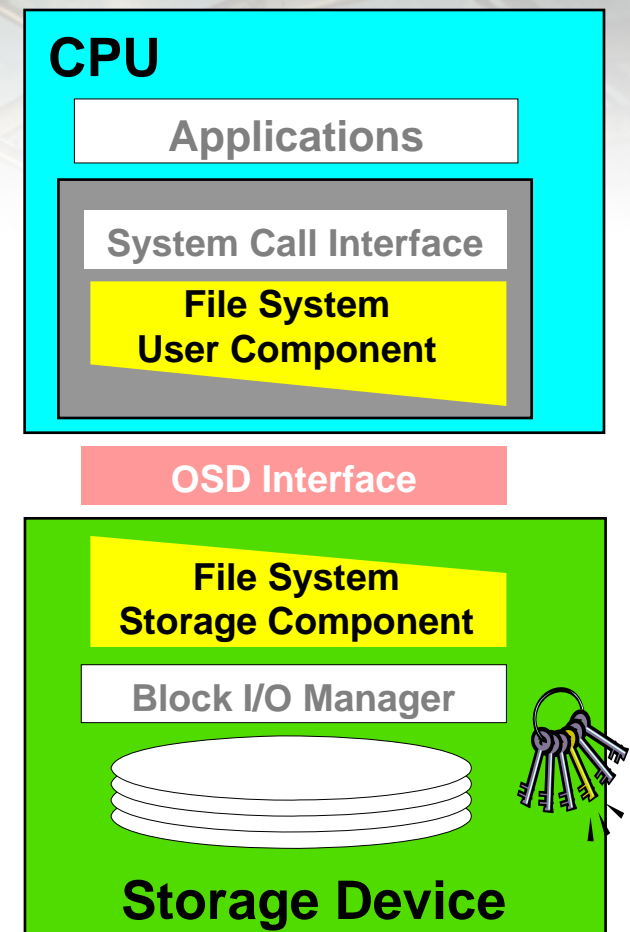
Object-based Storage Clusters

- Object Storage Devices
 - High-level interface that includes security
 - Block management inside the device
 - OSD standard (T10)
- File system layered over objects
 - National labs
 - Seismic data processing
 - Digital animation
- High performance through clustering
 - Scalable to thousands of clients
 - 10 GB/sec demonstrated
- Lustre, Panasas

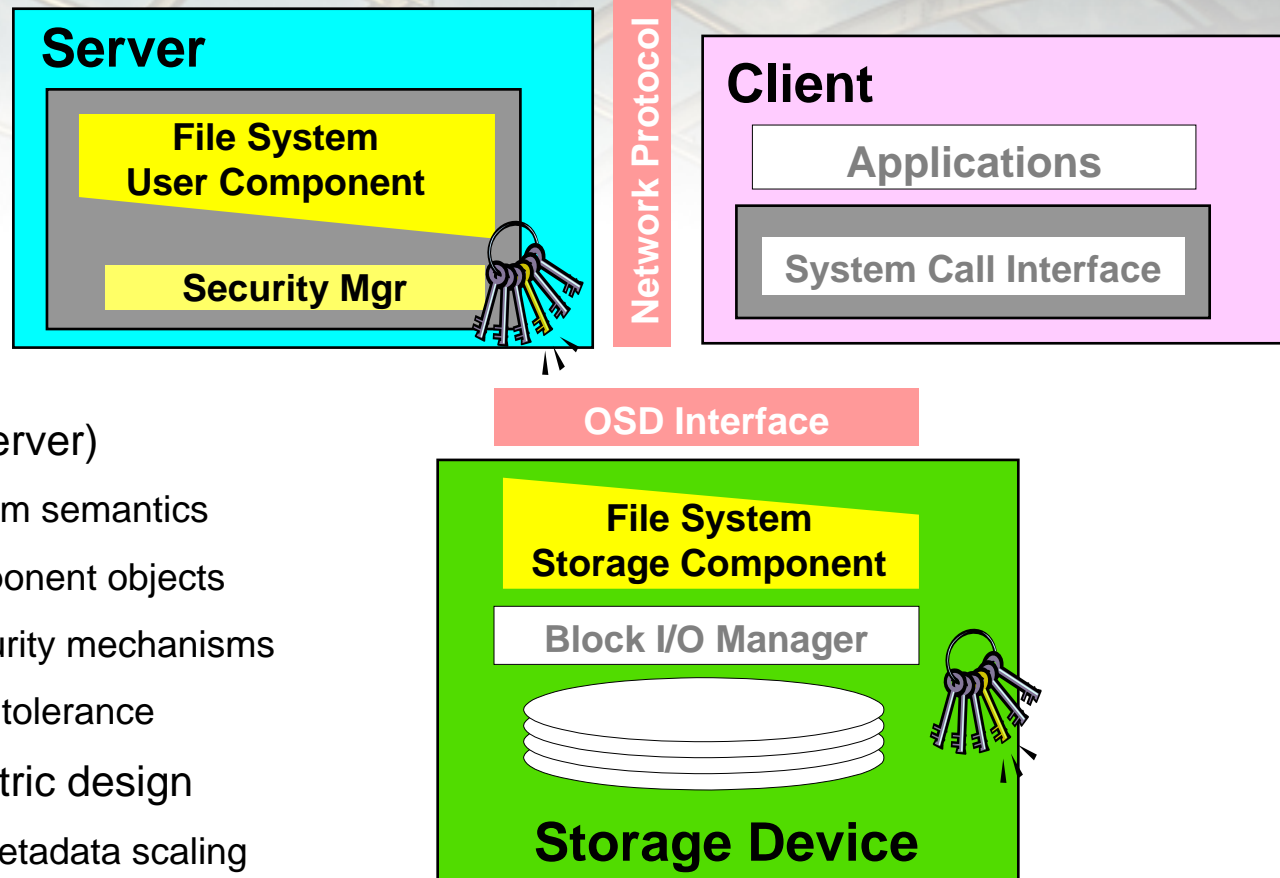


Object-based Storage (OSD)

- Object is container for data and attributes
 - Similar to inode abstraction
 - Storage understands logical data layout, not just sectors
- Provides secure access control on every command
 - A client sends an OSD command + capability (secure permission slip) that the OSD uses to decide if it's ok to process the command
 - Example, Read cmd w/CAP(R + W + Create)
- Higher-level storage systems built on objects
 - POSIX file system is just one possibility



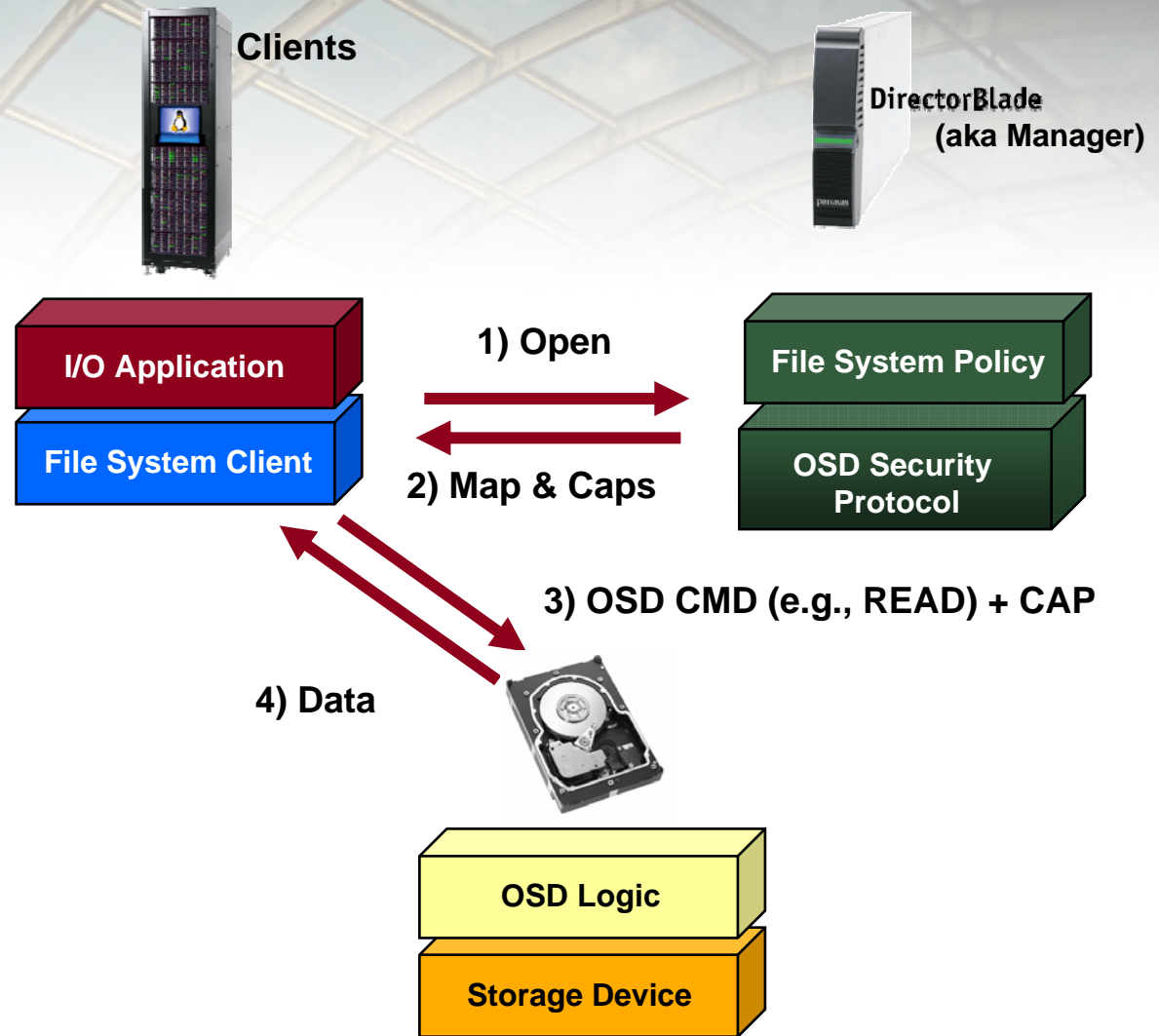
Distributed Object-based Storage (OSD)



- Metadata manager (server)
 - Implements file system semantics
 - Maps files onto component objects
 - Leverages OSD security mechanisms
 - Responsible for fault tolerance
- Out-of-band, Asymmetric design
 - Cluster servers for metadata scaling
 - Motivated by having many more clients (>1000) than metadata servers (>10)

Out-of-Band OSD System Architecture

- (1) Client initially contacts File Manager
 - File Manager checks file system policy
- (2a) File Manager returns list of objects
<OSD 987, OID 23>
- (2b) File Manager also returns a security CAPABILITY
 - Capability is security token used by client to access OSD - CAPABILITY authorizes client requests
- (3) Client sends requests (read, write) directly to OSD along with signature based on CAPABILITY
 - OSD checks signature, CAPABILITY, performs request
- (4) Direct-data transfer between client and OSD

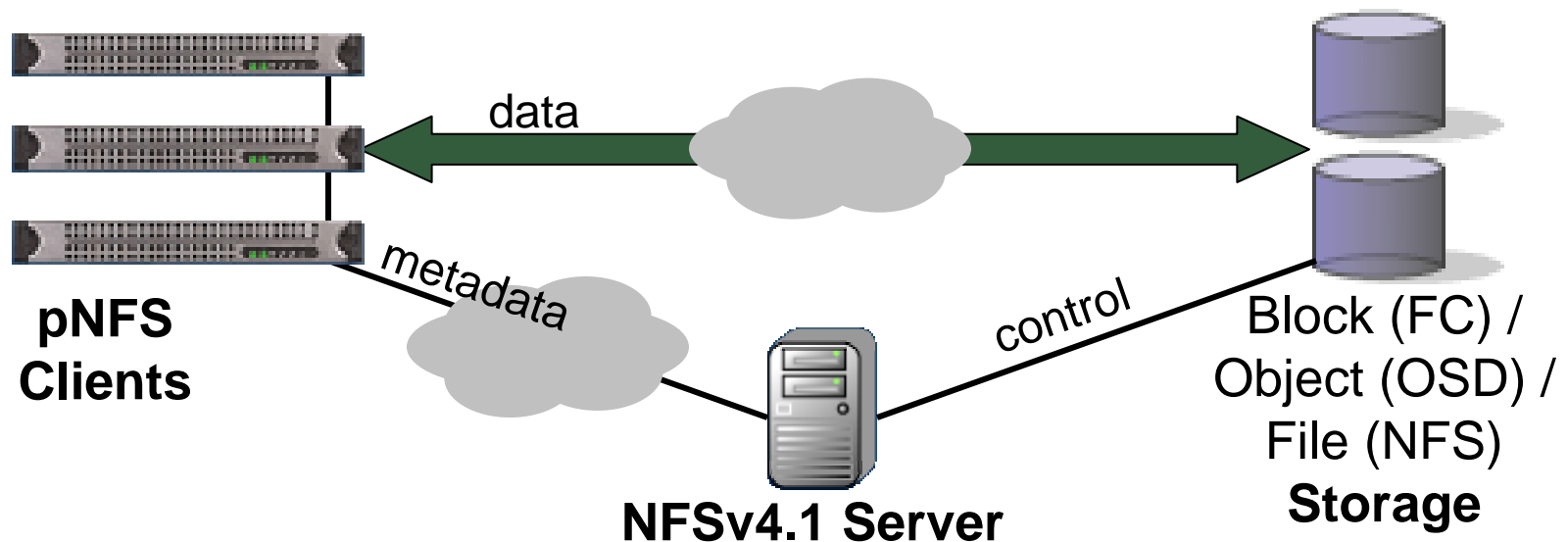


Interfaces: Blocks, Files and Objects

- Block-based architecture: **fast but private**
 - Traditional SCSI and FC approaches
 - Expensive fabric, difficult to share between hosts
 - Lack of security
 - Large amount of metadata to describe layout of files
- File-based architecture: **sharable, but bottlenecked performance**
 - NAS storage (NFS, CIFS, AFS and DFS)
 - Optimized for centralized server architecture
 - Coherence and security models vary widely
- Object-based: **fast, shared, secure device**
 - Storage device exports objects (collection of related data) instead of blocks
 - Building block for higher-level file systems

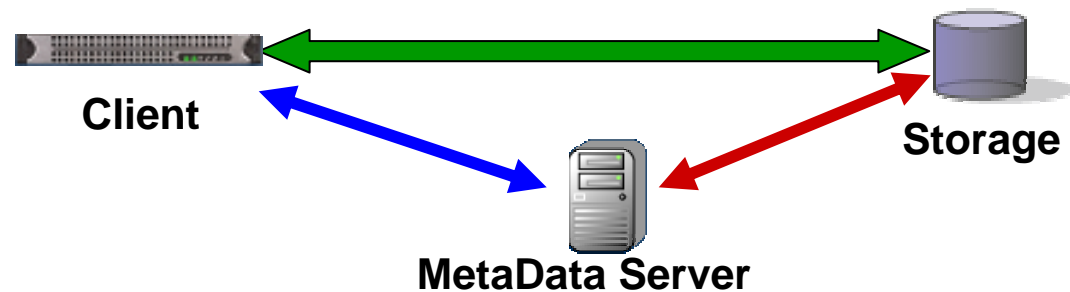
pNFS: Standard Storage Clusters

- pNFS is an extension to the Network File System v4 protocol standard
- Allows for parallel and direct access
 - From Parallel Network File System clients
 - To Storage Devices over multiple storage protocols
 - Moves the Network File System server out of the data path



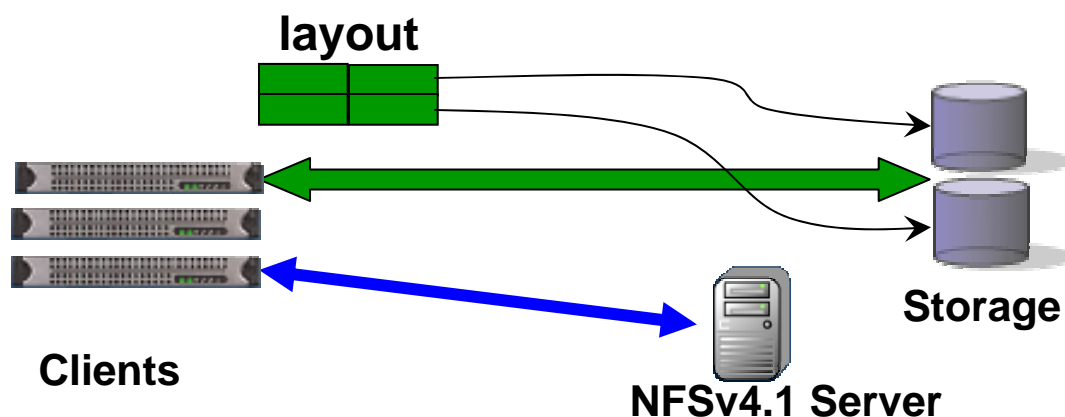
The pNFS Standard

- The **pNFS** standard defines the NFSv4.1 protocol extensions between the **server and client**
- The **I/O** protocol between the **client and storage** is specified elsewhere, for example:
 - SCSI **Block** Commands (**SBC**) over Fibre Channel (**FC**)
 - SCSI **Object**-based Storage Device (**OSD**) over iSCSI
 - Network **File** System (**NFS**)
- The **control** protocol between the **server and storage** devices is also specified elsewhere, for example:
 - SCSI **Object**-based Storage Device (**OSD**) over iSCSI



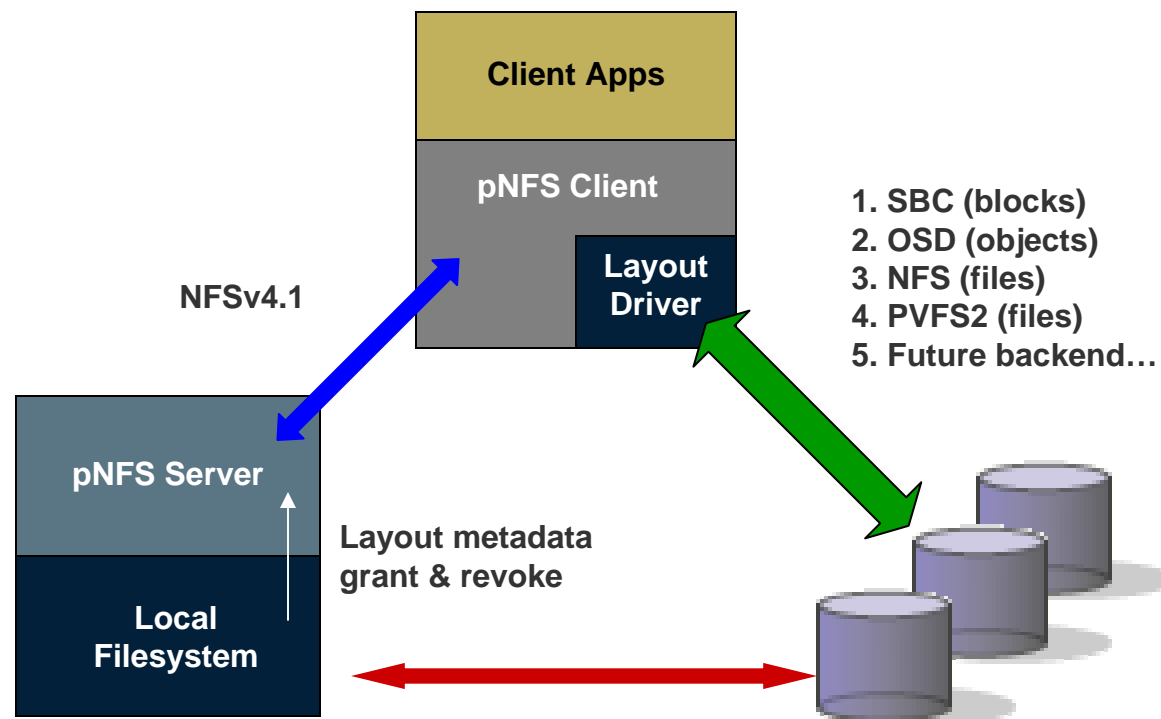
pNFS Layouts

- Client gets a *layout* from the NFS Server
- The layout maps the file onto storage devices and addresses
- The client uses the layout to perform direct I/O to storage
- At any time the server can recall the layout
- Client commits changes and returns the layout when it's done
- pNFS is optional, the client can always use regular NFSv4 I/O



pNFS Client

- Common client for different storage back ends
- Wider availability across operating systems
- Fewer support issues for storage vendors



pNFS is not...

- Improved cache consistency
 - NFS has open-to-close consistency enforced by client polling of attributes
 - NFSv4.1 directory delegations can reduce polling overhead
- Perfect POSIX semantics in a distributed file system
 - NFS semantics are good enough (or, all we'll give you)
 - But note also the POSIX High End Computing Extensions Working Group
 - <http://www.opengroup.org/platform/hecewg/>
- Clustered metadata
 - Not a server-to-server protocol for scaling metadata
 - But, it doesn't preclude such a mechanism

pNFS Protocol Operations

- LAYOUTGET
 - (filehandle, io_type, byte range) -> type-specific layout
- LAYOUTRETURN
 - (filehandle, byte range) -> server can release state about the client
- LAYOUTCOMMIT
 - (filehandle, byte range, updated attributes, layout-specific info) -> server ensures that data is visible to other clients
 - Timestamps and end-of-file attributes are updated
- CB_LAYOUTRECALL
 - Server tells the client to stop using a layout
- GETDEVICEINFO
 - Map deviceID in layout to type-specific addressing information

Layouts and Access Control

- Layouts provide a mechanism for clients to access storage
- But layouts do not provide permission
- Access to files uses standard OPEN, LOCK, and ACCESS operations
- Client and server must check access rights before using layout
- If layout or access rights are modified behind the clients back, pNFS has consistency model to recall layouts

pNFS Layouts: File

- File layout is an array of file handles with striping parameters

```
struct nfsv4_file_layout {                /* Per data stripe */
    pnfs_deviceid4    dev_id<>;
    nfs_fh4           fh;
};
```

```
struct nfsv4_file_layouttype4 {          /* Per file */
    stripetype4      stripe_type;
    bool             commit_through_mds;
    length4          stripe_unit;
    length4          file_size;
    nfsv4_file_layout dev_list<>;
};
```

pNFS Layouts: Block

- Block layout is a list of extents, each defined as:

```
struct {  
    file_offset  
    file_length  
    volumeID  
    storage_offset  
    storage_state {RW, READ, INVALID_DATA, NONE_DATA}  
}
```

pNFS Layouts: Object

- Object layout is an array of object IDs, capabilities, and striping parameters

```
objectid {  
    device_id  
    partition_id  
    object_id  
}
```

```
object_cred {  
    object_id  
    osd_version  
    opaque credential  
}
```

```
osd_layout {  
    file_size  
    list of components  
    data map  
}
```

```
data map {  
    stripe_unit  
    group_width  
    group_depth  
    mirror_cnt  
    raid_algorithm  
}
```

Is pNFS Enough?

- Standard for out-of-band metadata
 - Great start to avoid classic server bottle neck
 - NFS has already relaxed some semantics to favor performance
 - But there are certainly some workloads that will still hurt
- Standard framework for clients of different storage backends
 - Files
 - Objects
 - Blocks
 - PVFS2
 - Your project...

pNFS Status

- pNFS is part of the IETF NFSv4 minor version 1 standard draft
 - Bi-weekly conference calls to drive 4.1 draft to closure
- Reference open source client done in CITI
 - CITI owns NFSv4 Linux client and server
 - Weekly pNFS implementers' conference call
- Participants:
 - CITI (Files over PVFS2, Files over NFSv4)
 - Netapp (Files over NFSv4)
 - IBM (Files, based on GPFS)
 - EMC (Blocks, based on HighRoad)
 - Sun (Files over NFSv4, Objects based on OSDv1)
 - Panasas (Objects, based on Panasas ActiveScale Storage Cluster OSDs)
 - Carnegie-Melon University, performance and correctness testing

More pNFS Status

- Working out code structure for internal Linux APIs
 - Client: Merged PVFS prototype and NetApp NFSv4 files client
 - Client: should layout cache be generic or layout-specific
 - Server: standard API to export file system layouts
 - No recalls, yet (i.e., callbacks)
- Prototype interoperability began in 2006
 - San Jose Connect-a-thon March '06
 - Ann Arbor NFS Bake-a-thon September '06
- Availability
 - TBD – gated behind NFSv4 and working implementations of pNFS

Object Storage Architecture

- Raises storage's level of abstraction
 - From logical blocks to objects (object is a container for data and attributes)
 - Allows storage to understand how different blocks of a object are related
 - Provides storage with necessary info to optimize storage resources
- An evolutionary improvement to standard (SCSI) storage interface

Block Based Disk

Operations:

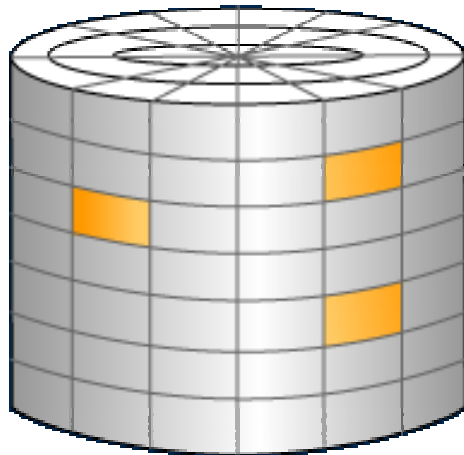
Read block
Write block

Addressing:

Block range

Allocation:

External



Operations:

Create object
Delete object
Read object
Write object
Get Attribute
Set Attribute

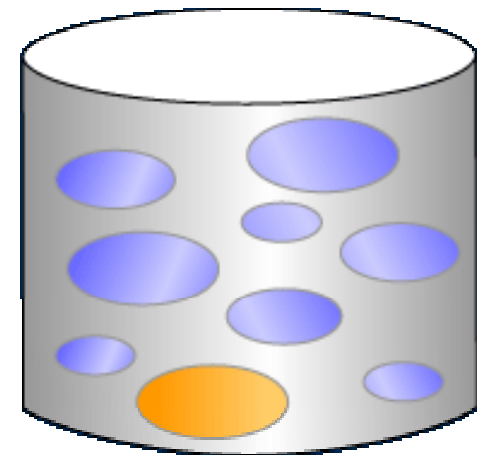
Addressing:

[object, byte range]

Allocation:

Internal

Object Based Disk



Read ahead and write behind

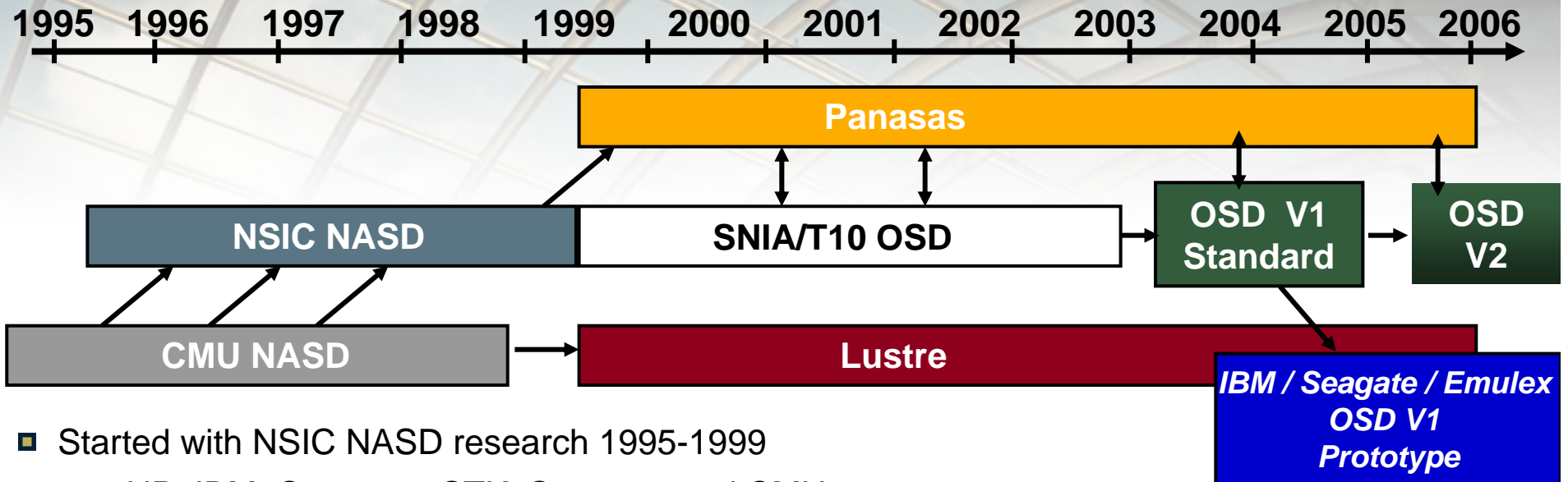
- OSD maintains many read-ahead contexts
 - SATA drive firmware implements 10 block-based read ahead contexts
 - OSD can implement 100's of object-aware read ahead contexts
 - Read ahead data in response to initial GETATTR
 - Read ahead other object descriptors in response to GETATTR
 - Requires higher level hints to relate objects
- Delayed block allocation for optimal layout
 - Buffer IO (in NVRAM)
 - Onodes, block pointers, refmap updates, and data blocks
 - Allocate blocks at the last moment before IO
- These optimizations are examples from the Panasas OSD

Wide Variety of Object Storage Devices

- Disk array subsystem
 - Ie. LLNL with Lustre
- Panasas StorageBlade
 - 2 SATA disks, CPU and GE NIC
- Prototype Seagate OSD
 - Highly integrated, single disk



Object Storage Timeline



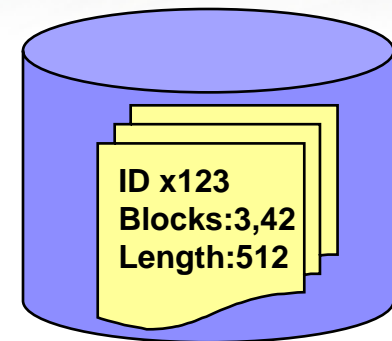
- Started with NSIC NASD research 1995-1999
 - HP, IBM, Quantum, STK, Seagate, and CMU
 - Eventually became SNIA Technology working group in '99
 - 45 participating companies
- 1999 moves to SNIA/T10 working group
- 1/2005: ANSI ratifies V1 T10 OSD standard (ANSI/INCITS 400-2004)
 - SNIA TWG already working on OSD V2 features
 - Snapshots, import/export, multi-object capabilities and extended attributes



Objects

- An object is a logical unit of storage
 - Lives in (almost) flat name space – addressed via object ID
 - Contains application data AND **metadata**
 - Data: variable size container, byte addressing
 - Metadata: block allocation, physical length (similar to inode)
 - And contains user accessible **attributes**
 - OSD interpreted: QoS requirements, capacity quota, etc.
 - Opaque to OSD: file system and app metadata
- Object access via file-like methods
 - read, write, create, delete, getattr, setattr

An Object



Type of Objects (con't)

■ Object Hierarchy and Types

■ User-object

- Up to 2^{64} per partition
- Basic storage unit for user data + attributes

■ Partition-object

- Up to 2^{64} per device
- Container for user-objects that share security & space mgmt characteristics

■ Root-object

- One per device, defines device characteristics (e.g., device capacity)
- Container for partition objects

■ Collection-object

- Up to 2^{64} per partition ... share namespace with user-objects
- Collection holds list of user-objects
- Fast index that applications can use to create arbitrary set of user-objects
 - Audio collection that lists all of my MP3s
- Useful support for fault-tolerance/recovery
- V2 support for multi-object operations (e.g., delete, query, list)

T10 OSD Object Attributes

- **Every object has a set of associated attributes**
 - Stores various information (e.g., capacity used, last-access time, object_version_number)
 - Some attributes defined by standard ... others available for higher-level software
- **Attribute Pages**
 - 2^{32} attribute pages with 2^{32} attributes per page
 - All attributes virtually exist on object create
 - Attributes written with explicit setattr()
 - Attributes read with explicit getattr()
- **Attribute page definition/contents differ by object type and categories**
 - Defined by T10 OSD Standard, other standards, manufacturers, etc
- **Most commands embed set- or getattr()**
 - Piggy-backed operations minimize message traffic

OSD Commands (from ANSI T10 Spec)

Basic Protocol

- READ

- WRITE

- CREATE

- REMOVE

- GET ATTR

- SET ATTR

}

very basic

}

space mgmt

}

attributes

- timestamps
- vendor-specific
- opaque
- shared

Specialized

- FORMAT OSD

- APPEND – write w/o offset

- CREATE & WRITE – save msg

- FLUSH – force to media

- FLUSH OSD – device-wide

- LIST – recovery of objects

Security

- Authorization – each request

- Integrity – for args & data

- SET KEY

- SET MASTER KEY

}

shared secrets

Groups

- CREATE COLLECTION

- REMOVE COLLECTION

- LIST COLLECTION

- FLUSH COLLECTION

Management

- CREATE PARTITION

- REMOVE PARTITION

- FLUSH PARTITION

- PERFORM SCSI COMMAND

- PERFORM TASK MGMT

OSD impact on iSCSI stack

- Large CDB
 - OSD commands are big (256 bytes), using the extended CDB format
 - E.g., Linux normally has small in-line struct, now keeps a pointer to large CDB
- Bi-Directional Data Transfer
 - Three buffers involved in a command:
 - Data payload
 - Piggy-back set attributes
 - Piggy-back get attributes
- Open Solaris, Linux support (patches available)

T10 OSD V2.0 Work in Progress

- Collections and Multi-object collection operations
 - Single command instructs OSD to operate on set of objects
- Snapshot support
 - Copy-on-write for objects, collections of objects
- Efficient error handling
 - Fast error detection and recovery
 - Error handling for multi-object operations

Collections

- OSD Collection Object is an object used to store a list of user object IDs
- User objects are added to or removed from a collection by performing a SETATTR on the user object's collection page
 - Enables collection manipulation as side effect of other command (e.g., WRITE)
- Use case 1: Fast Index
 - Transaction needs to record all objects it touches
 - Using piggyback SETATTR(into collection X) to add each object into the collection as the object is dirtied
 - If client fails (e.g., reboots), it can discover which objects are dirty by listing the collection
- Use case 2: List of related objects
 - EX: Pseudo directory of all MP3 objects
- Basic collection commands
 - CREATE COLLECTION, REMOVE COLLECTION

OSD Snapshot

- OSD V2.0 defines snapshots to be point-in-time copies of partitions
 - Used partition as basis for snapshot because partitions are the basic unit of space management
- Snapshots may be implemented as
 - Efficient copy-on-write
 - Sync byte-by-byte copy
 - Async byte-by-byte copy
- OSD keeps list of snapshots (parent / child relationships in snapshot attribute page)
- Set of snapshot commands
 - CREATE SNAPSHOT
 - RESTORE SNAPSHOT
 - REFRESH PARTITION
 - DELETE SNAPSHOT

Multi-object Operations

- GET MEMBER ATTRIBUTES
 - Returns the specified attribute(s) from every object listed in the collection
- SET MEMBER ATTRIBUTES
 - Sets the specified attribute(s) on every object listed in the collection
- REMOVE MEMBER OBJECTS
 - Deletes every user object listed in the collection
- QUERY
 - Match against one or more specified <attribute, value> pairs, returning the list of user objects that successfully matched
- Ordering of internal command processing is unspecified
 - Allows for efficient disk-directed processing

Error Handling – Damaged Objects

- Objects can be damaged for several reasons including media defects and software bugs
- Management should be alerted when a damaged object is detected
 - Proactively: OSD sends message to manager
 - Discovered: Object damage is recorded inside OSD and may be queried by manager
- OSD marks damaged objects
 - Specific object is fenced (topmost bit of object version # is set)
 - Prevents client access to object until manager can examine object
 - Partition and root attribute set to timestamp of latest discovered damage
 - Manager can poll timestamps to discover new damage has been detected by OSD

Strengths of Object Storage

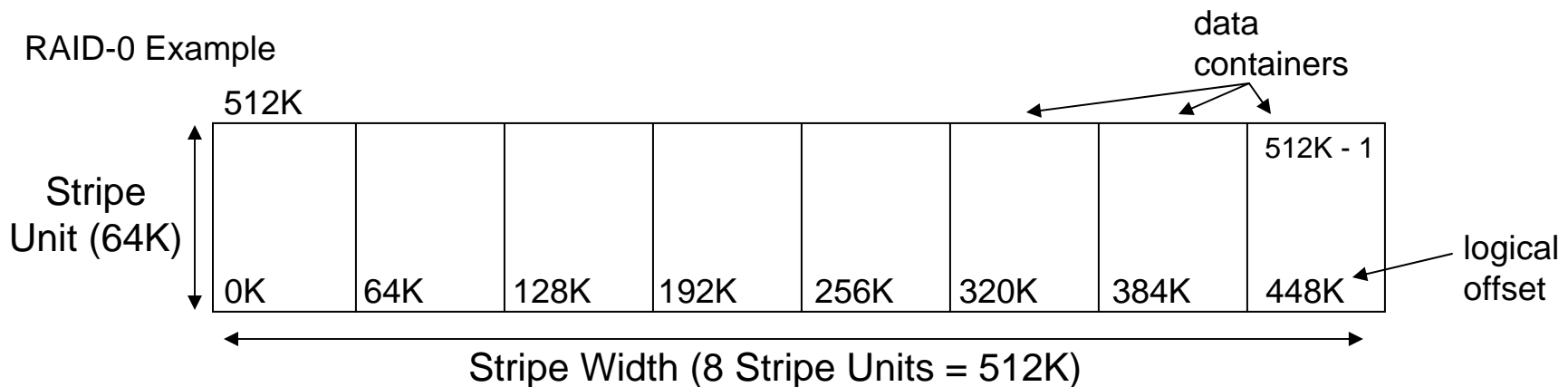
- **Object maintains data relationship within OSD**
 - Decisions on data layout can be optimized based on object size and usage
 - OSD can be self-organizing, self-optimizing
- **Extensible attributes**
 - E.g. size, timestamps, ACLs, etc.
- **Access Control decisions are signed, cached at clients, enforced at device**
 - Clients can be untrusted (bugs & attacks expose only authorized object data)
- **Command set works with SCSI architecture model (SAM)**
 - Encourages cost-effective implementation by storage device vendors

More to come...

- Scaling RAID to deal with high-capacity media
 - Block- and file-based RAID
- Metadata
 - Recoverability and Scalability
- OSD Security
- Performance Studies
 - Experience with Panasas Object Storage

RAID Taxonomy

RAID-0	Striping, no redundancy
RAID-1	Mirroring
RAID-10	Striped mirrors
RAID-3	Striping, parity, synchronized disks. <i>DataDirect Networks</i>
RAID-4	Striping, dedicated parity disk. <i>NetApp</i>
RAID-5	Striping, rotated parity.
RAID-6	Striping, parity, 2 nd "parity" drive. Rotated parity an option. <i>NetApp Diagonal Parity. Reed-Solomon p+q.</i>

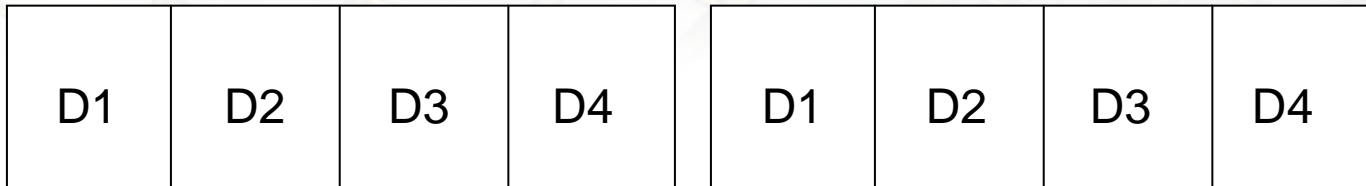


RAID Taxonomy Examples

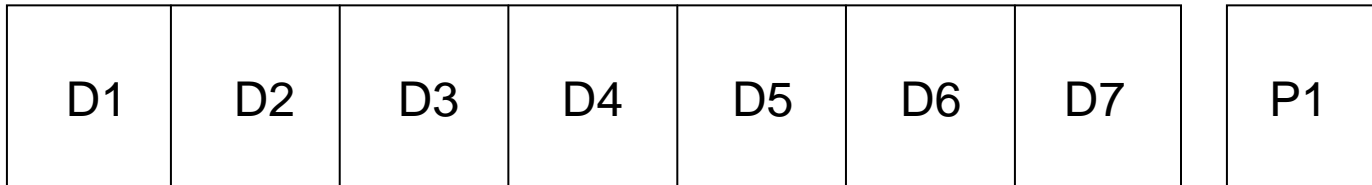
RAID-1



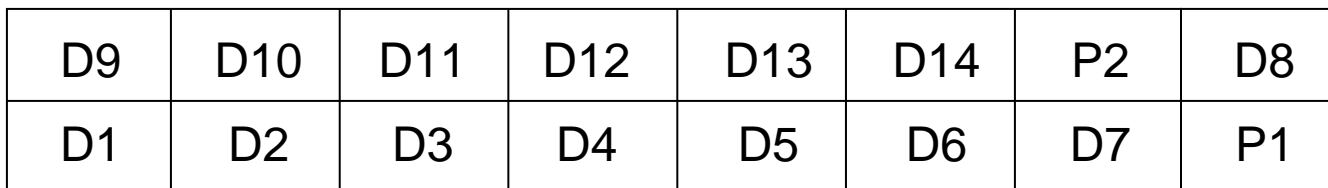
RAID-10



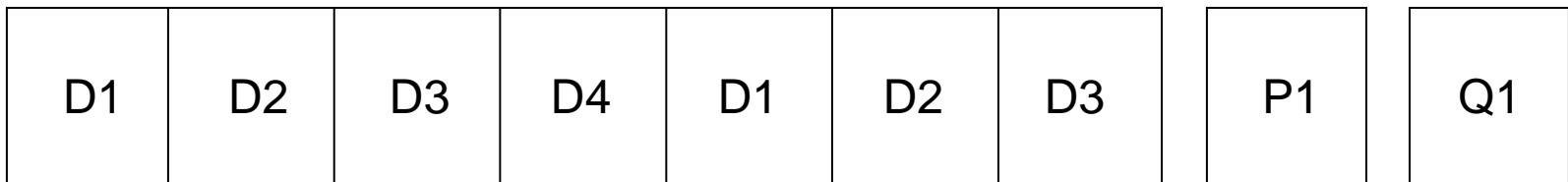
RAID-4



RAID-5



RAID-6



RAID Issues

- RAID Small Write Problem
 - Writes less than full stripe require 4-cycle IO to read old parity, read old data, compute new parity block(s), write new data, write new parity
 - RAID-1 and RAID-10 avoid this problem, but with 100% capacity overhead
- Dedicated parity drives (RAID-4) are hot spots
- Synchronized heads (RAID-3)
 - “free” parity, but many fewer random read IO operations
 - RAID-4 with 1-byte stripe unit size – always reading and verifying parity

RAID Reconstruction

- Consumes controller bandwidth, interfering with on-line workload
 - Fixed bandwidth encourages smaller stripes (e.g., 4 not 8)
 - Example controller with 100 MB/sec throughput
 - 4+1 rebuild at 20 MB/sec
 - 6+1 rebuilds at 14 MB/sec
 - 8+1 rebuilds at 11 MB/sec
- Vulnerable to media errors, which are addressed by per-file RAID or RAID-6
 - Note that vendors often use wide RAID-6 stripes to reduce parity overhead, but that could double the time for a single-failure rebuild
- Complex encoding require multiple passes over the data to recover from more than one failure

Block and File RAID

- Traditional RAID devices encode at the block level
 - Parity drive is a function (XOR) of N data drives
 - Read every sector of surviving drives during reconstruction
 - Free space is reconstructed
 - Active data may or may not be prioritized
 - Capacity grows faster than bandwidth => increasing rebuild times
- File-level RAID uses replication or other encoding on a per-file basis
 - Parity equations applied to chunks of a file, not to disks
 - Different files can have different encoding
 - Free space not reconstructed
 - Performance scalability through *declustering*
 - Unrecoverable failures quarantined to a file (not a LUN)
 - Panasas, Isilon, Google

Scalable Panasas RAID

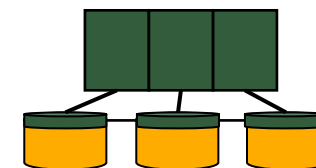
- System assigns RAID level based on file size
 - ≤ 64 KB RAID 1 for efficient space allocation
 - > 64 KB RAID 5 for optimum system performance
 - > 1 GB two-level RAID-5 for scalable performance
 - RAID1 and RAID-10 for optimized small writes
- Automatic transition from RAID 1 to 5 without re-striping
- Programmatic control for application-specific layout optimizations
 - Create with layout hint
 - Inherit layout from parent directory

Small File



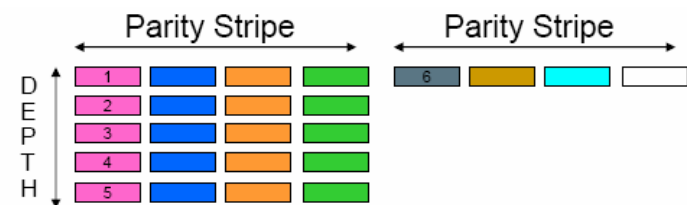
RAID 1 Mirroring

Large File



RAID 5 Striping

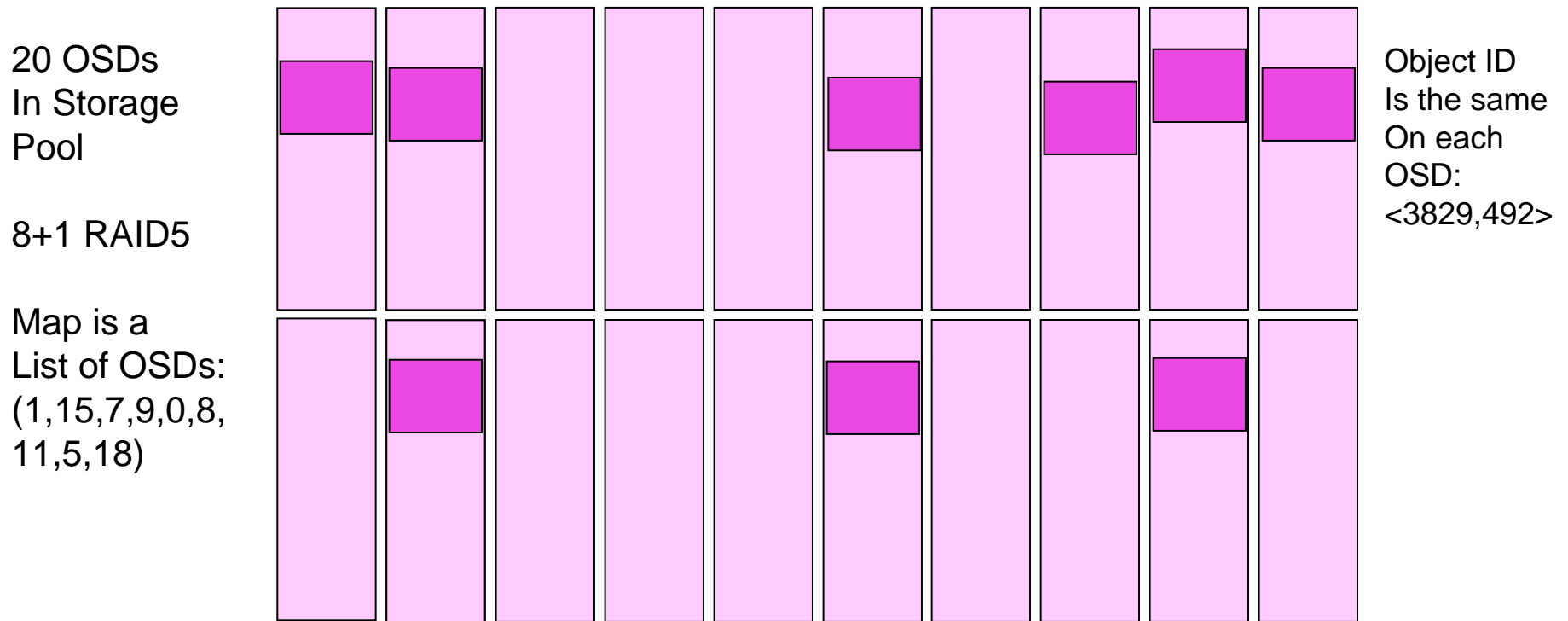
Very Large File



2-level RAID 5 Striping

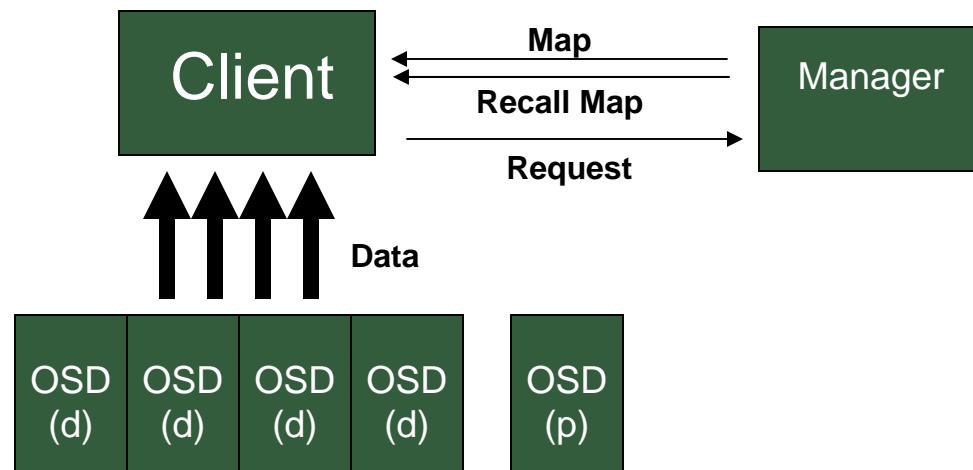
Per-File RAID

- Files are striped across component objects on different StorageBlades (OSD)
- Component objects include file data and file parity for reconstruction
 - Think of them as virtual disks (i.e., containers) that stripe one file
- File attributes are replicated with two component objects



Example: Client Read Operation

- Steps in a client read operation
 - Client determines file ID and responsible manager from the directory entry
 - Client requests permission to read from the manager (read cap)
 - Director returns permission + file map identifying components
 - Client determines byte ranges within components and initiates a network transfer for each, all in parallel, ignoring the parity blocks
 - Client can re-use permission and map until told otherwise by manager

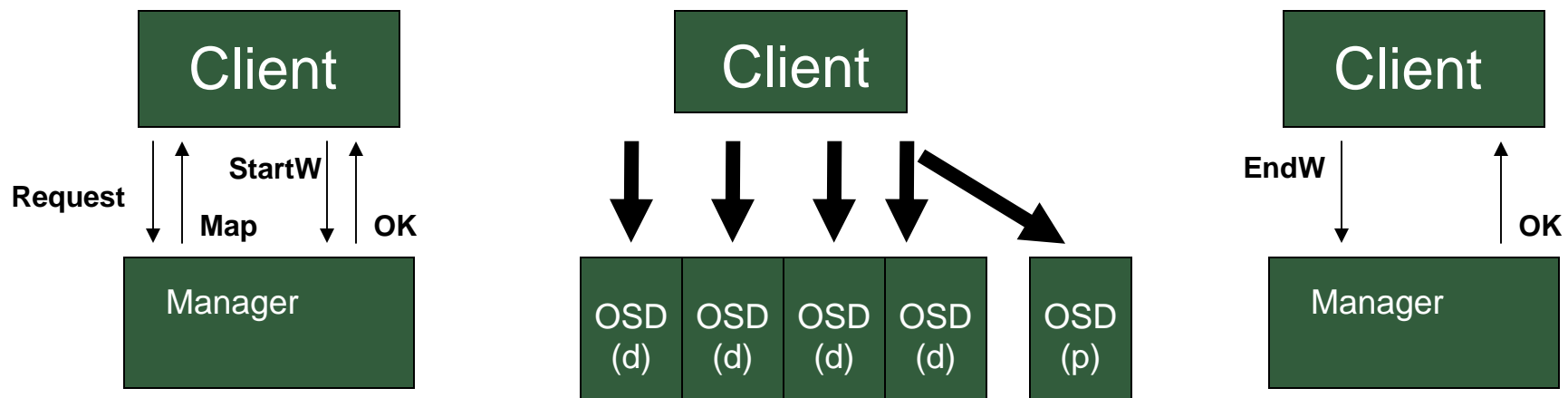


Client Read Operation

- Different files can use different sets of OSDs
- Bandwidth is achieved via parallel transfers from multiple OSDs
 - More OSDs \Rightarrow higher parallelism \Rightarrow higher bandwidth
- No serialization at a server machine, no bottleneck at a RAID controller
- Many OSDs and ample switch infrastructure together allow many cluster nodes to simultaneously achieve high bandwidth from storage
- Client need not keep track of sector mapping: it requests a single byte range from a single named object
- OSD is free to optimize: cache, read-ahead, write-behind, relocate

Example: Client Write Operation

- Director must assure that concurrent writes do not corrupt parity, and that clients see coherent data in storage
- Therefore follow “start-write / end-write” policy (`LAYOUTGET/LAYOUTCOMMIT`)
- Client accumulates “enough” dirty data to get high bandwidth transfer
- Client requests permission to write, writes all dirty data and updates parity, releases write permission back to the director
- Write permission is relatively short term to bound recovery commitments

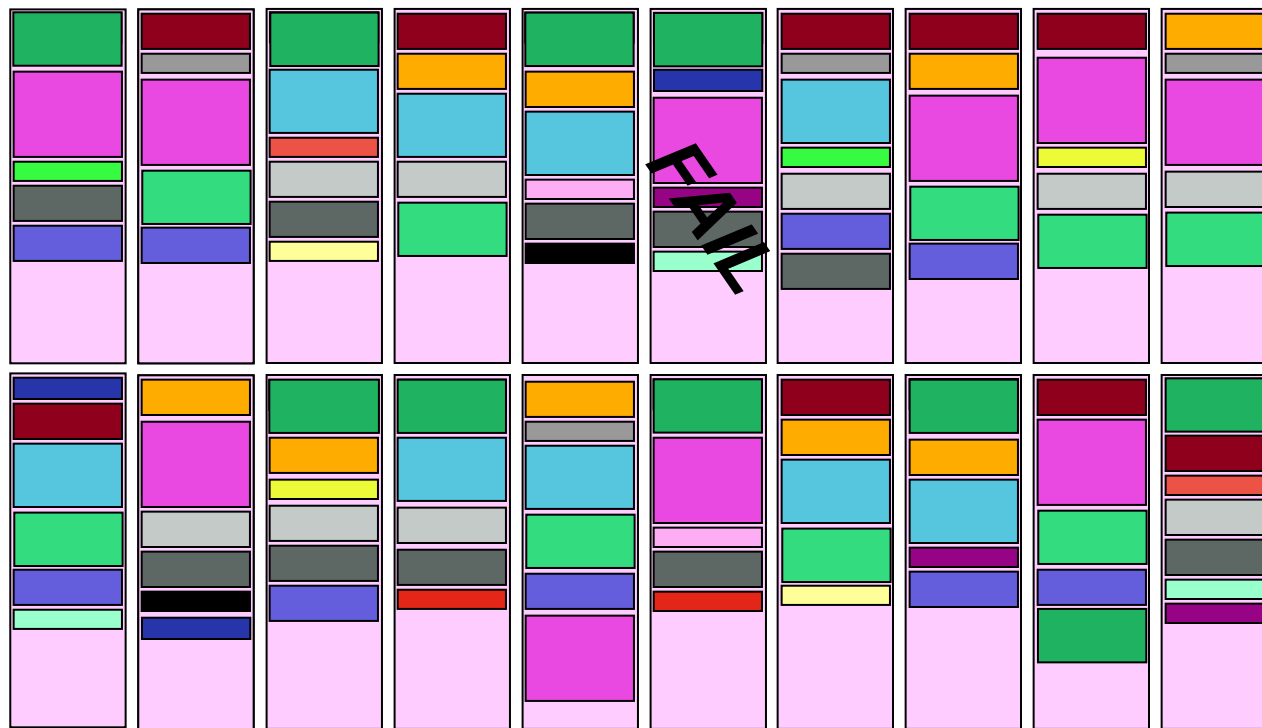


Declassified RAID

- Files are striped across component objects on different StorageBlades
- Component objects include file data and file parity for reconstruction
- File attributes are replicated with two component objects
- Declassified, randomized placement distributes RAID workload

20 OSD
Storage Pool

Mirrored
or 9-OSD
Parity
Stripes



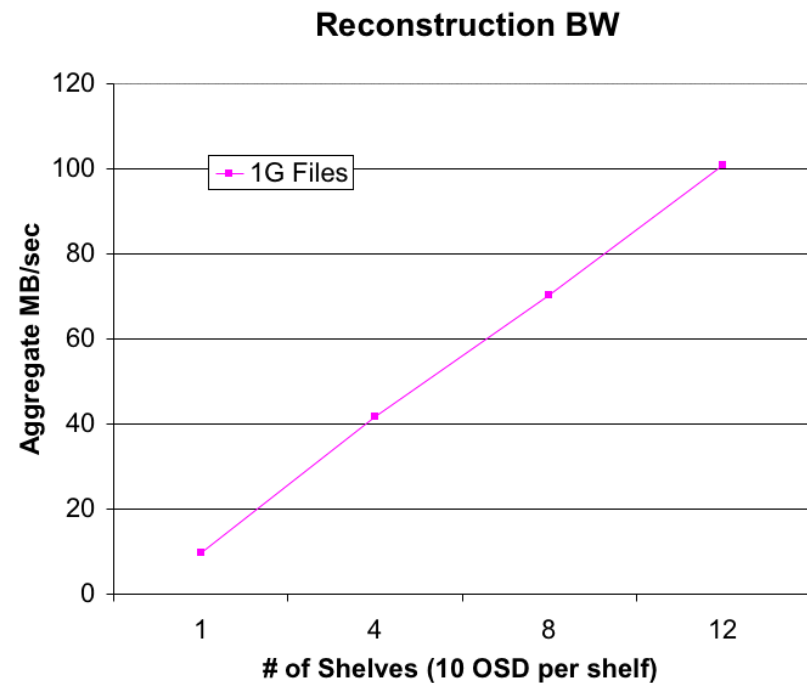
Read about half of each surviving OSD

Write a little to each OSD

Scales up in larger Storage Pools

Scalable RAID Rebuild

- Two main causes of RAID failures
 - 1) **2nd drive failure in same RAID set during reconstruction of 1st failed drive**
 - Risk of two failures depends on time-to-repair
 - 2) Media failure in same RAID set during reconstruction of 1st failed drive
- Shorter repair time in larger storage pools
 - From 12 hours to 1.5 hours for 500GB
- Four techniques to reduce MTTR
 - Use multiple “RAID engines” (DirectorBlades) in parallel
 - Spread disk I/O over more disk arms (StorageBlades)
 - Reconstruct data blocks only – not unused space
 - Blade Drain to remove SMART failing blades



Scalable rebuild is mandatory

- ↑ Having more drives increases risk, just like having more light bulbs increases the odds one will be burnt out at any given time
- ↓ Larger storage pools must mitigate their risk by decreasing repair times
 - The math says
 - if (e.g.) 100 drives are in 10 RAID sets of 10 drives each and
 - each RAID set has a rebuild time of N hours
 - The risk is the same if you have a single RAID set of 100 drives
 - and the rebuild time is N/10
 - Block-based RAID scales the wrong direction for this to work
 - Bigger RAID sets repair more slowly because more data must be read
 - Only declustering provides scalable rebuild rates

$$MTTF_{RAID} = \frac{(MTTF_{Disk})^2}{(D+C*n_G)*(G+C-1)*MTTR}$$

Total number of drives → $(D+C*n_G)$ $(G+C-1)$ ↑ ↓ $MTTR$ → *Repair time*

Drives per RAID set → n_G

Snapshots and object movement

- Reconstruction and data migration must be snapshot aware
 - Reconstruction is the generation of a lost object from redundant data
 - Migration is movement of an object for load balancing
- Naïve implementation expands shared blocks
 - 10 GB object with 20 clones => 200 GB when copied or reconstructed
 - 10 GB object with 9 GB hole, 1 GB => 10 GB when copied or reconstructed
- Object interface needs to expose layout information
 - Optimizations always cause layering violations

OSD Commands for Snapshot Rebuild

- OSD commands support recreating snapshot chains
 - READDIFF – what ranges differ between two objects
 - READMAP – what ranges of object are really holes?
- Reconstruction
 - Read oldest object in chain. Create and write oldest copy
 - READDIFF next object
 - CLONE then READ and WRITE ranges that changed

*Proposed
For OSDv2*

Handling media defects

- Two main causes of RAID failures
 - 1) 2nd drive failure in same RAID set during reconstruction of 1st failed drive
 - Risk of two failures depends on time-to-repair
 - 2) **Media failure in same RAID set during reconstruction of 1st failed drive**
- Storage System has to monitor media for defects
 - Catch-22 – writes are blind, errors are discovered on read
 - Must suffer read error, then write new data, before drive remaps failed sector
 - Parity scrubbing – background task to verify RAID parity equation
- What to do if a media error occurs during reconstruction?
 - Block RAID – write zeros to failed stripe
 - Block RAID – fail the entire LUN (and by implication, its file system)
 - File RAID – fence file affected by the double error

Managing Metadata

- Out-of-band architecture manages metadata off the datapath
 - All clients contact metadata manager(s) that manage:
 - File to storage mapping
 - File system security policy
 - File system sharing and cache coherence policy
 - Clients obtain the following from the metadata manager
 - Layout that maps file to object(s) or blocks
 - File attributes (e.g., access time)
 - Capability that grants access to objects (and specific ranges w/in objects)
 - Callbacks to provide file coherence when shared
- Symmetric architectures distribute work to software entities co-resident on the file system clients
 - File access results in local or remote metadata operation
 - Same metadata functions as those listed above

Managing Metadata (con't)

■ Issues

- High-level file system metadata management
 - Where is this information?
 - How does it scale in size and performance?
- File system data and metadata consistency
 - Ensure consistency among sharing clients
 - Ensure file system metadata consistency
- Protocol support for efficient metadata management
 - Minimize network traffic between client and server
 - Minimize network traffic between server and storage
- Scaling metadata workload
 - Partition workload among cluster of metadata managers

Locating a File via a Database

- Option 1: Database stores map and other attributes for each file
 - Clients contact MDS who does the file lookup for them
 - Requires scalable metadata database
 - Loss of database leaves data meaningless
 - Isolate database storage
 - Dual-redundant MDS servers
 - Back it up!
 - ADIC (and many block-based systems), Lustre, Google
 - Lustre uses an Ext3 file system on the MDS that stores pointers to objects
 - Content-addressible systems (EMC Centera)

Locating a File via Directories

- Option 2: Extend traditional Directory structure to handle distribution
 - Directory contains file name, low-level ID, and (in some systems) location hint
 - Client reads and parses directories
 - Block-based systems retain existing formats (XFS -> cXFS)
 - Clients obtain locks to perform directory updates, allocate inodes, etc.
 - File attributes stored in an inode structure on disk
 - Panasas FS stores file system attributes on a pair of objects
 - Complete storage map as list of OSDs and striping parameters
 - File length, capacity, timestamps
 - Owner and ACL information
 - Parent pointer(s)
 - Can recover file system after complete loss of metadata manager
 - Panasas (object), GPFS (block), CXFS (block)

Locating a File, Trade-offs

- A: Database stores map for each virtual object
 - As fast (or slow) as the database
 - Can be optimized for important queries
 - Decoupled from the data (independent failure domains)
 - Data could be meaningless with loss of metadata database
- B: Distribute metadata with objects
 - Standard file system design
 - Distributed workload to storage devices allows scalability
 - Recoverable data and metadata together with fsck
 - Rummaging through TB of data is expensive – online algorithms attractive
- Hybrid models
 - Incur costs of both, but could allow optimized, recoverable systems

Caching and Cache Consistency

- Caching information on client reduces network traffic
 - Clients cache file data, directory data, attributes, and capabilities
 - Clients cache both clean (read) and dirty (written) data
- Stateless servers (NFS) require client to poll for consistency
- Stateful servers lease “callbacks” to clients
 - Server promises to notify the client if cached data or attributes become invalid, or other client wants to access file
 - Callback type indicates sharing state, similar to CIFS opportunistic lock (oplock)
 - Exclusive callback = no other clients using file; we can cache reads and writes
 - Read-only shared callback = other clients reading, but no writers; we can cache reads
 - Read/write shared callback = other clients reading or writing; don't cache anything
 - Concurrent write callback = special sharing mode for cooperating apps
 - Callbacks are leased & expire after ~8 hrs unless renewed
 - AFS, DFS, Panasas, NFSv4 (file lock delegation only)

Unified Locking and Caching Protocols

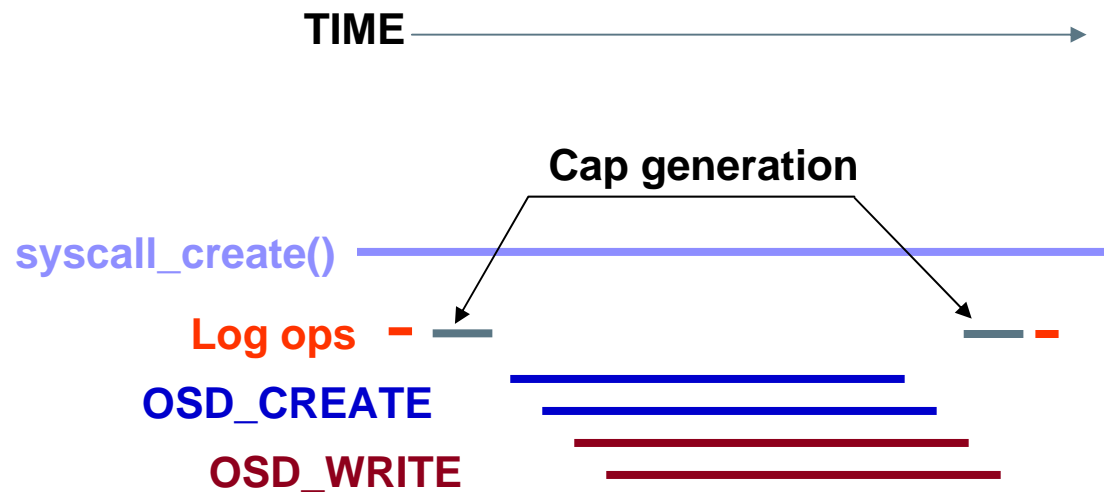
- Caching and locking both imply ownership of something
 - Directory, File data, attributes, application-level file lock
- Lock manager can be used to implement cache consistency and serialization
 - Caching data, attributes takes read-only lock
 - Setting an attribute, creating a file, modifying a file, takes read-write lock
- Distributed lock manager delegates lock control to different participants
 - Optimizing locking traffic can result in complex protocols
- Lock recovery protocols required to handle crashed participants
 - Leases imply grace periods during which no locks are granted
 - Special recovery operations rebuild lock state during grace period
 - Long leases reduce traffic, but cause long hangs after crashes
- GPFS, Exanet, Lustre

Metadata journaling

- Operations across multiple objects must be recoverable
 - Storage operation could fail
 - Metadata manager could crash
 - Storage must remain consistent or be returned to consistent state
- Types of logging
 - Ledger: add record, do operation, delete record
 - State log: longer term memory of FS state
 - **write cap, client participants, repair log**
 - Reply cache for non-idempotent operations
- Where to log
 - NVRAM (5 usec), remote memory (120 usec over GE)
 - local disk, remote object storage (.5 to 5 msec)

Example: Parallel Create

- Create a mirrored file (2 objects) and insert into directory (Panasas)
- Timeline
 - client:syscall_create()
 - mgr: log operation
 - mgr: generate capability for manager
 - mgr-to-osd: OSD_CREATE & OSD_CREATE for redundant user object
 - mgr-to-osd:OSD_WRITE & OSD_WRITE for redundant directory update
 - mgr: generate capability for client
 - Mgr: clean up create log entry, remember write capability



Generating pictures like this from real trace data is enlightening

Snapshots

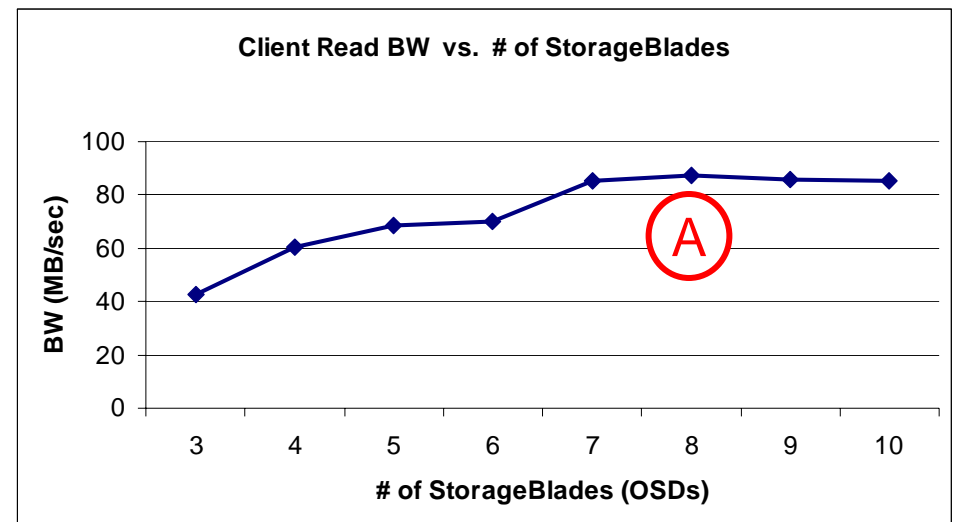
- Metadata managers coordinate snapshots
 - Pause client activity
 - Clone partitions that store objects
 - Resume client activity
 - Provide namespace for snapshot access (e.g., `.snapshot` sub directories)
- Copy-on-write objects
 - `CLONE_OBJECT` and `CREATE_SNAPSHOT` OSD commands
 - No overwrite block allocation
- Quiescing write activity can be expensive
 - Large clusters can have lots of IO in flight
 - Global quiesce is slow
 - Rolling quiesce (e.g., to each LUN) is not consistent

Capacity Management

- Can you grow your storage cluster transparently
 - Or, do you add a new storage island?
- Blocks world: Concatenate a new LUN onto your volume
 - Is new LUN a hot spot?
 - Can the file system migrate cold blocks to new storage?
 - Can the file system migrate data off an ancient LUN transparently?
 - Some volume managers can do this. Big pain point for customers.
- Objects world (anything with per-file mappings)
 - Transparently shuffle components around and update the map

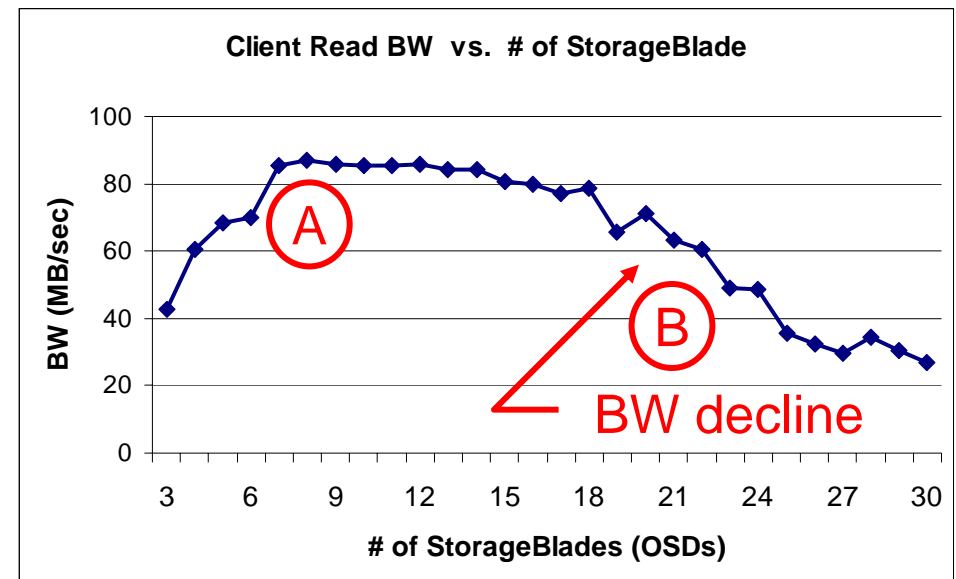
Achieving High Performance Storage

- Build a System that delivers linear scalability of BW
 - Adding disks increases both capacity & storage bandwidth all the way to client apps
 - Requires elimination of all bottlenecks between storage & clients
 - Remove server bottleneck: Direct client-storage transfers
 - Scalable storage devices: Object-based storage
 - Scalable file system: Panasas DirectFlow
 - Scalable NW: Ethernet/TCP
- Plot shows BW increasing as StorageBlades (OSDs) are added
 - All numbers are from disk, not cache
- Details of graph
 - Single client reading from N StorageBlades
 - File data is striped across StorageBlades, with each data point restriping the file
 - Network iSCSI, TCP/IP, Gigabit ethernet
 - Storage: Object-based cmds T10 OSD
 - Panasas File System – Direct Flow



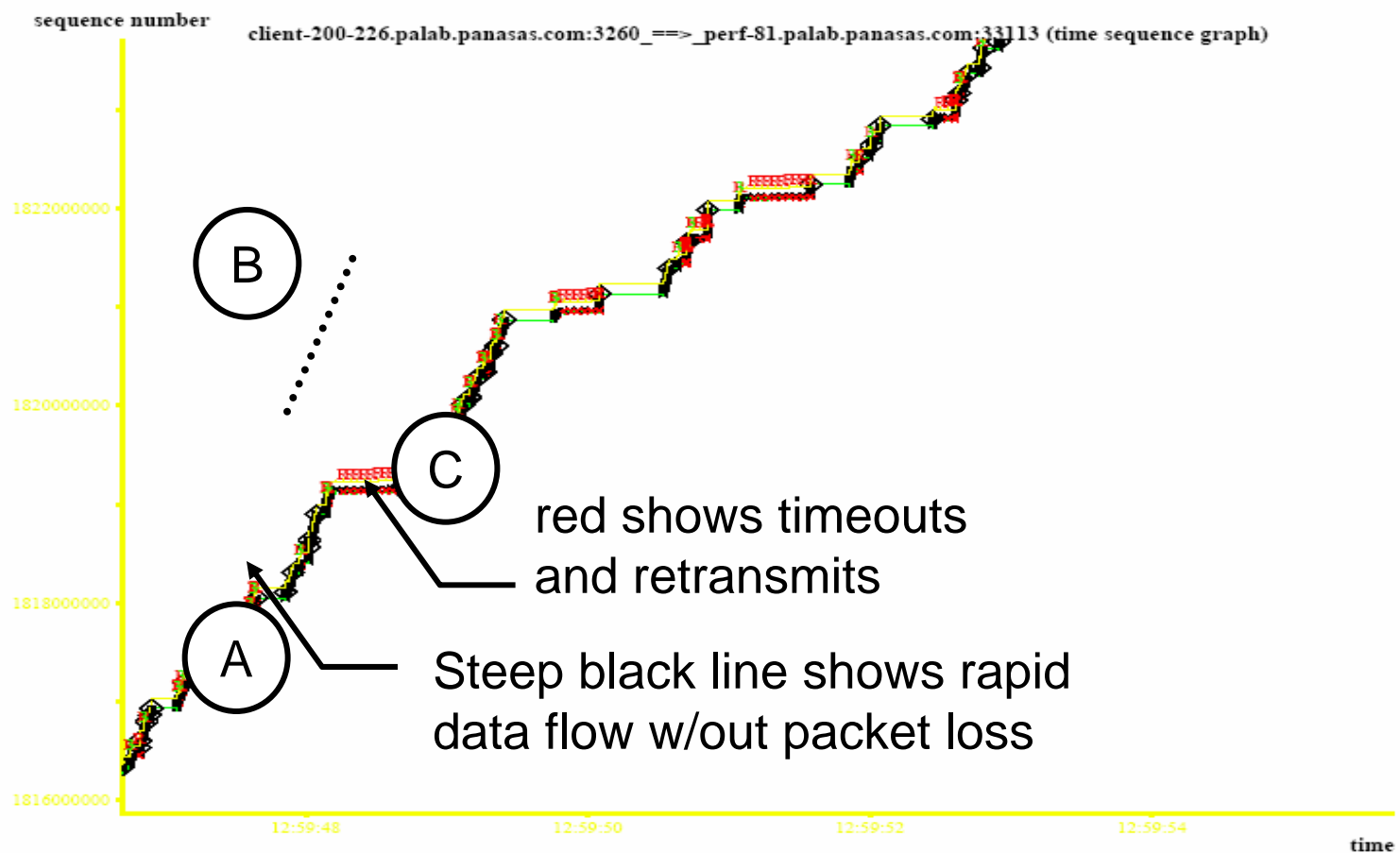
Scalability (wide stripes)?

- Greater than 15 StorageBlades, BW begins to decline
 - Why? Possible culprits included ...
 - Longer storage access times - striping wider decreases amount of data fetched per disk
 - Interference with other requests ... but data was collected on unloaded system
 - Client can't receive data fast enough
 - Each StorageBlade can transmit at 1Gbit/sec (peak), but client only receive at 1Gbit/sec max
- Real culprit ... the network
 - We call this problem INCAST (opposite of multicast)
 - Can effect any application that receives data from multiple sources
 - Network congestion between sources and destination cause packet loss
- Common data access pattern for striped storage
 - Object-based Storage, iSCSI, NFS Aggregation
- Can Incast be avoided ?
- Can Incast be fixed ?



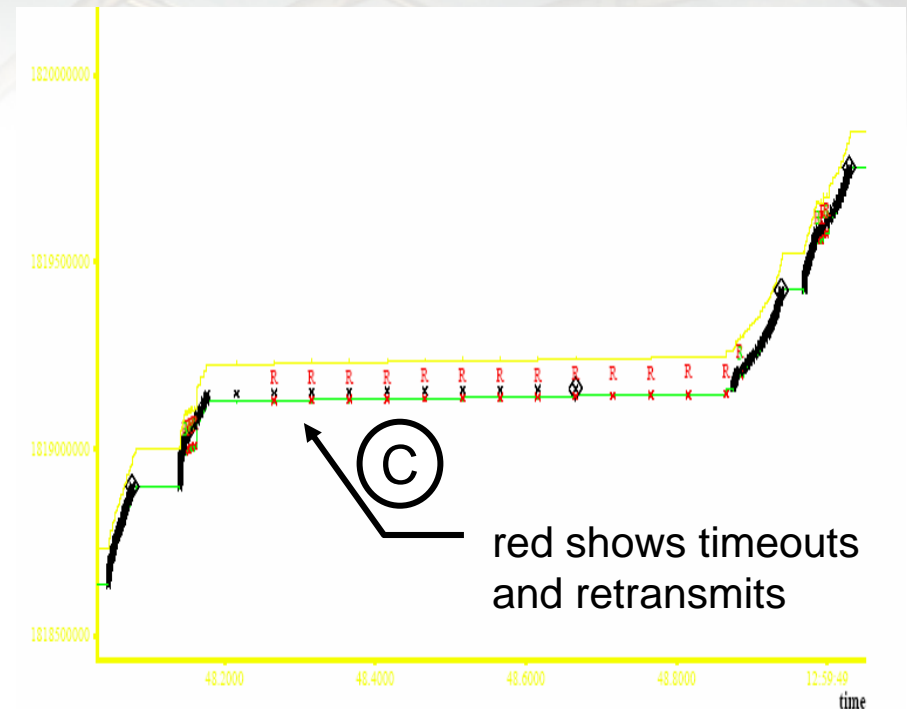
How Does TCP Perceive Incast?

- TCP Sequence Number Plot (Time vs. Sequence Number)
 - Many ODSs sending to single client (picture of one 1 OSD/Client flow)



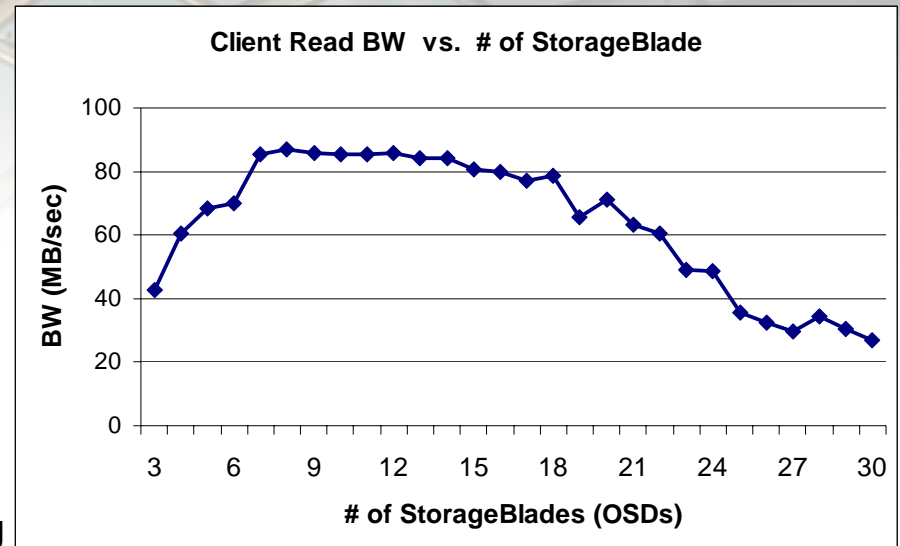
TCP Plot of Retransmissions

- Network buffers overflow and packets are dropped
 - TCP retransmit will eventually kick in
 - TCP timeouts on scale 100's of msec, SAN operates on usec scale
 - TCP fast retransmit and SACK defeated by significant tailedrop packet loss
 - SACK ineffective if receiver doesn't know packets were dropped
 - Important difference between storage and traditional network flows
 - Storage is much more bursty
 - More difficult for TCP flow control to adapt to storage flows
 - By striping across devices, storage requires ALL flows to respond before the app can proceed
 - Any interrupted flow hurts total throughput



Overcoming Incast

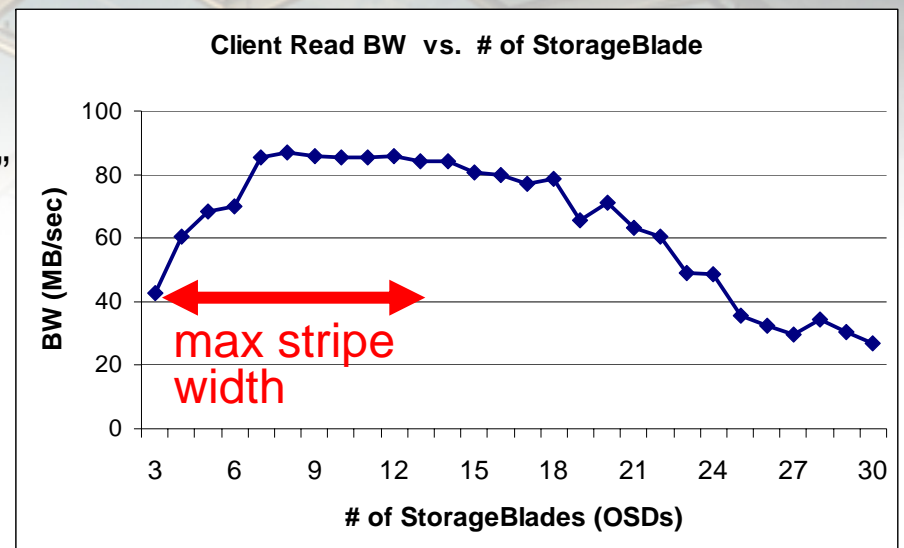
- Option 1: Fix the network
 - Link-level flow control
 - Can penalize good senders
 - Large network buffers
 - Work with vendors
 - Reduce socket buffers
 - Use small socket buffers (< 64KB)
 - Since receiver has N TCP streams working in concert ... ideally buffer sizing would consider aggregate of all flows
 - VERY fast retransmit ... TCP timeouts not designed for SAN latencies
 - Modified TCP to reduce timeout by factor of 4X
 - Tail-drop is still a problem
 - SAN-optimized Ethernet and TCP would be excellent research topic
 - Still want to stay on commodity curve
- All fixes above were not sufficient ...



Changing TCP behavior is hard

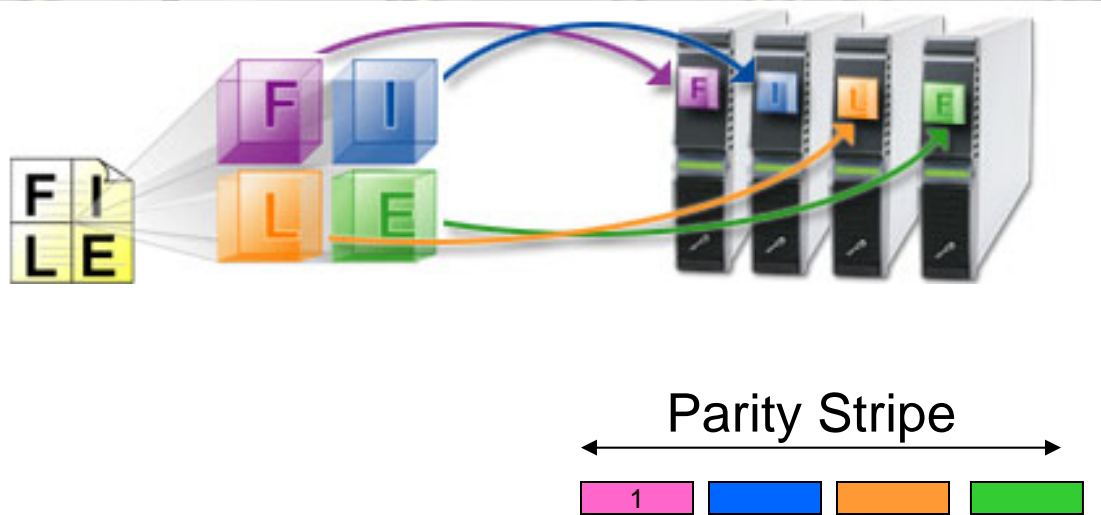
Overcoming Incast (2)

- Option 2: Reduce number of senders (aka StorageBlades)
 - Stripe only wide enough to achieve “peak” bandwidth
 - Con: Reduces aggregate bandwidth because bandwidth is limited to number of StorageBlades file is striped across
 - Con: Restricts maximum file size
 - Con: Potential hot spots



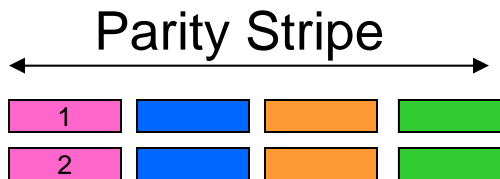
- What we really want is to: 1) limit stripe width to avoid INCAST while 2) striping across all of the storage blades

Option 3: 2-Level RAID Map



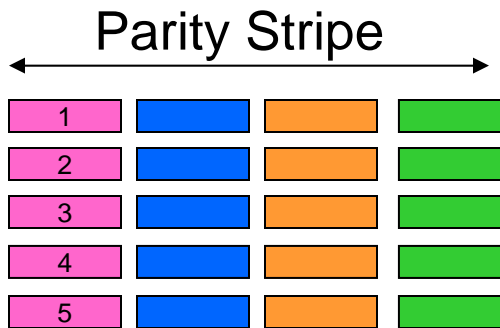
- Begin by striping the file across a limited number of OSDs

2-Level RAID Map



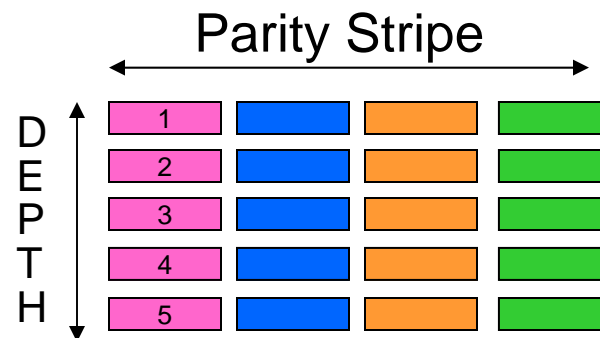
- Begin by striping the file across a set of OSDs
 - As the file grows, write more stripes

2-Level RAID Map



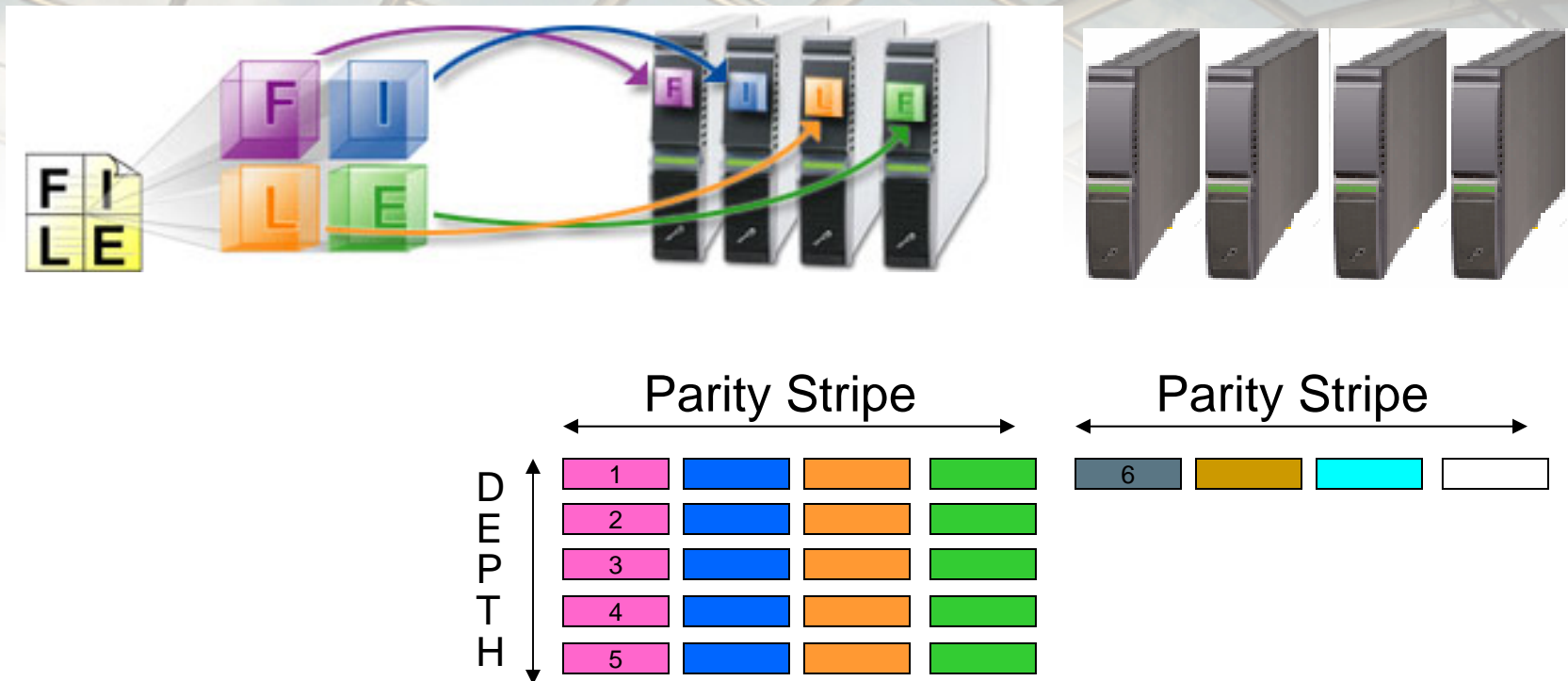
- Begin by striping the file across a set of OSDs
 - As the file grows, write more stripes

2-Level RAID Map



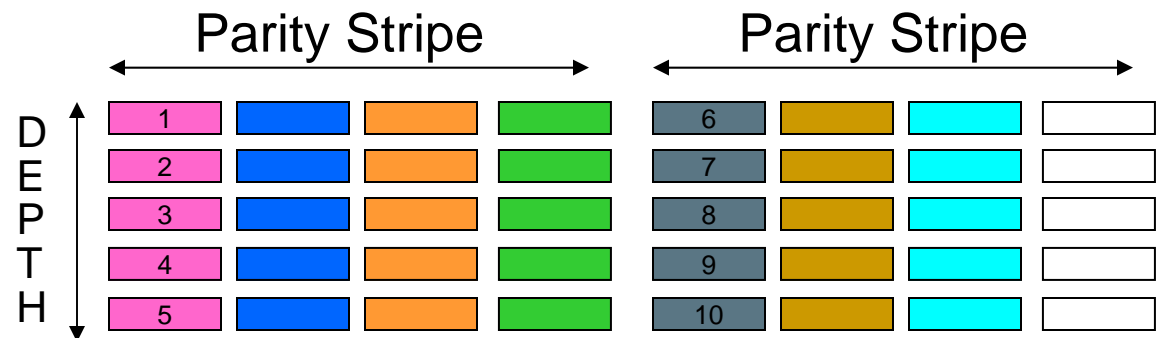
- Begin by striping the file across a set of OSDs
 - As the file grows, write more stripes
 - Once a “sufficient” number of stripes has been written (aka DEPTH),
 - stripe across a new set of OSDs

2-Level RAID Map



- Begin by striping the file across a set of OSDs
 - As the file grows, write more stripes
 - Once a “sufficient” number of stripes has been written (aka DEPTH),
 - stripe across a new set of OSDs

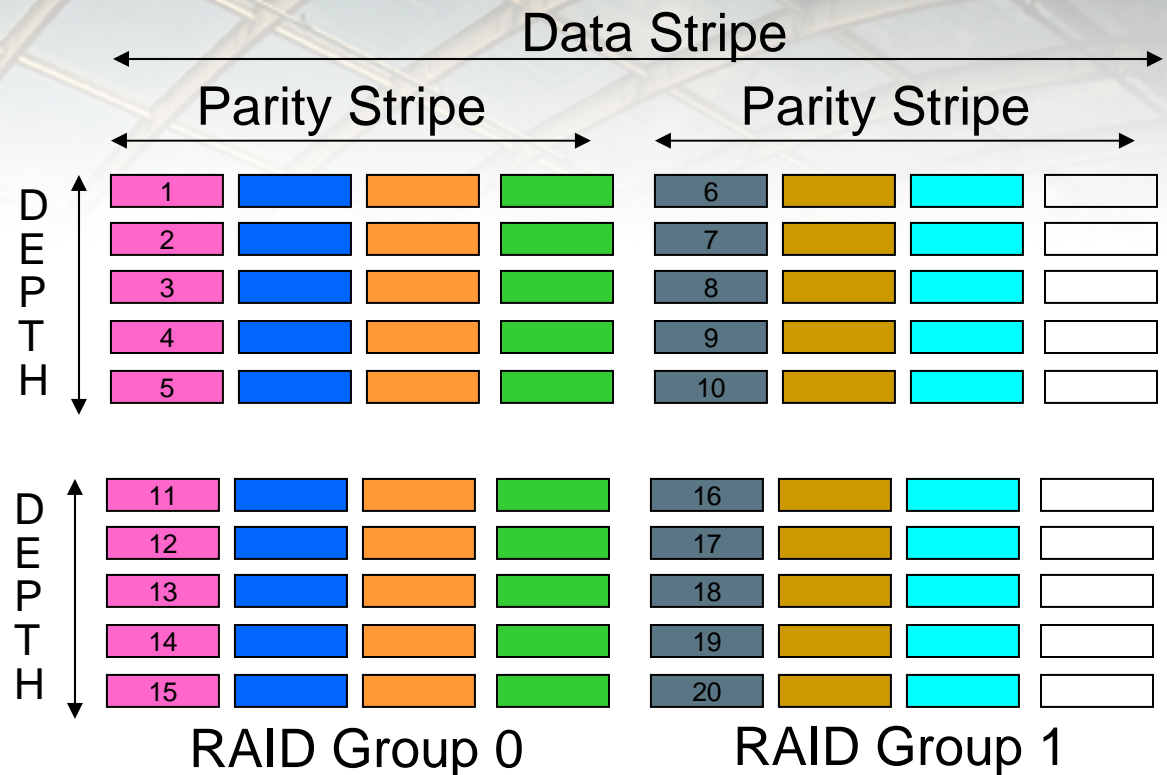
2-Level RAID Map



- Begin by striping the file across a set of OSDs
 - As the file grows, write more stripes
 - Once a “sufficient” number of stripes has been written (aka DEPTH),
 - stripe across a new set of OSDs

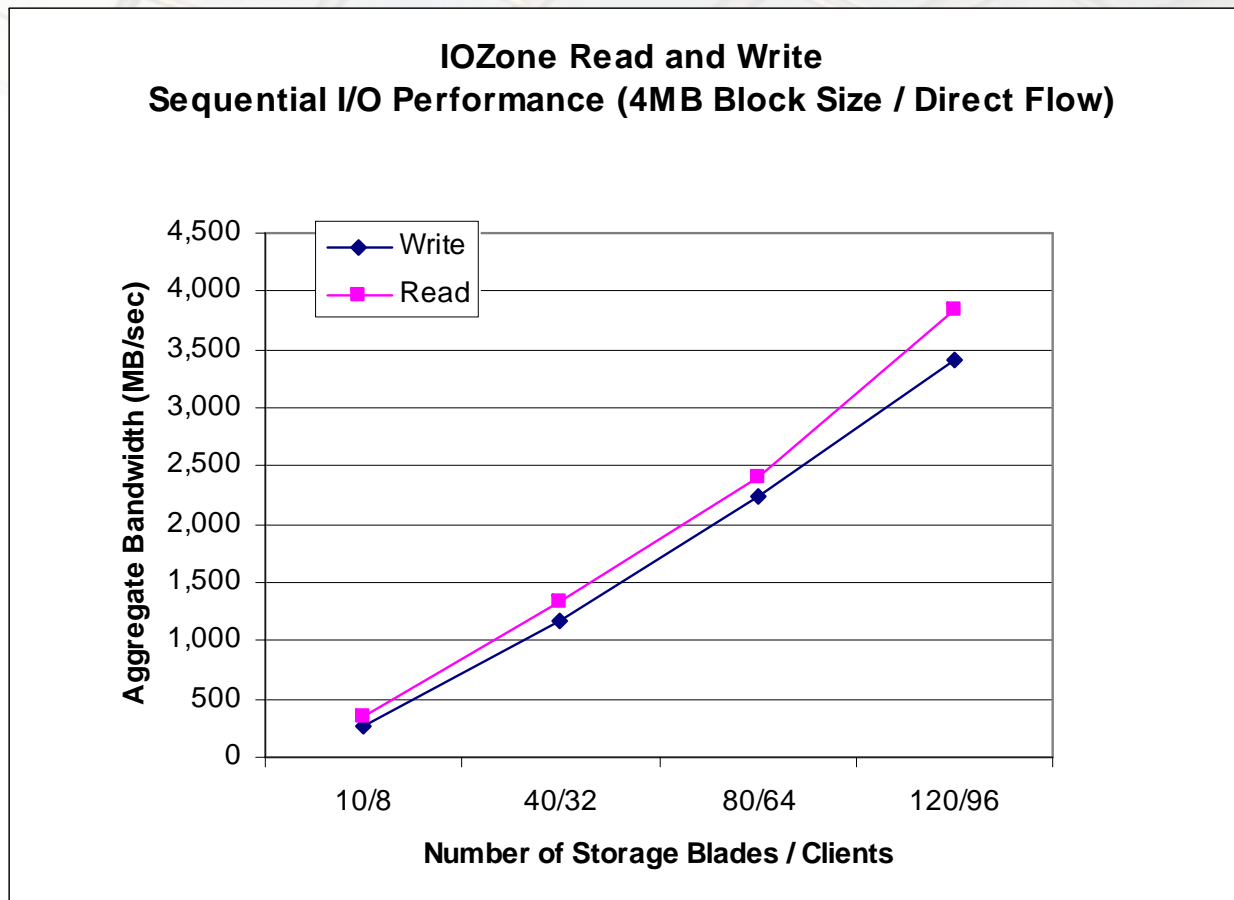
2-Level RAID Map

- Maximize bandwidth by striping across ALL StorageBlades in the system
- Minimize incast by only communicating with one RAID group at a time
- Optimize disk utilization by making stripe depth large enough to read sequentially from multiple tracks
- Load balance by distributing clients across all Storage Blades
 - Each file is striped using a different set of RAID Groupings
 - Multi-client/single file accesses fan out across RAID groups



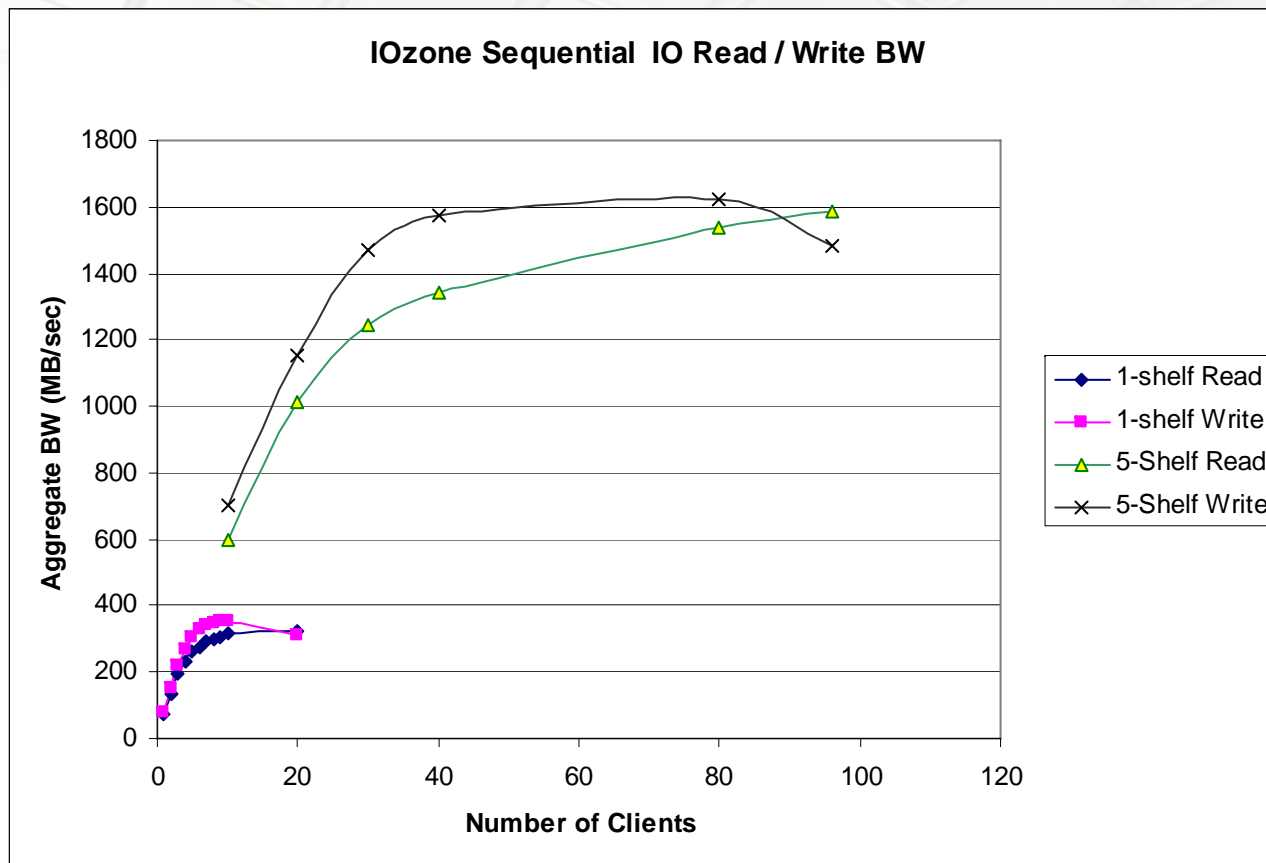
Scaling the system

- Scale the system and clients at the same time (N-to-N IOzone, Panasas)



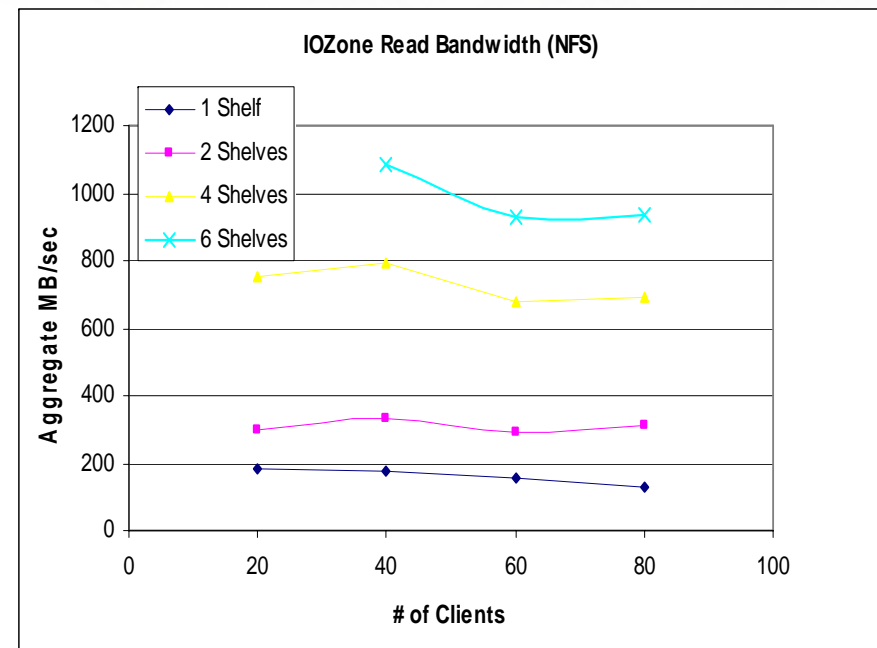
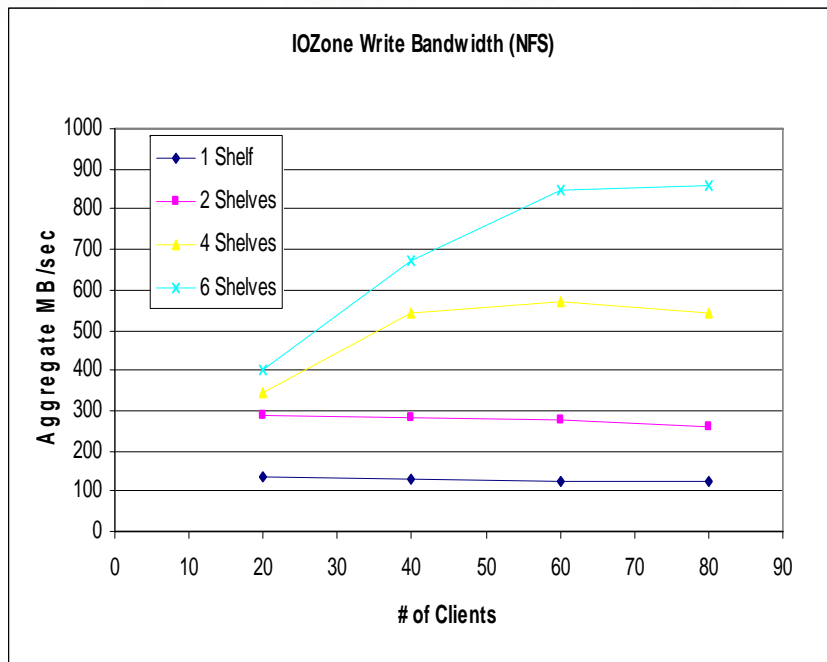
Scaling Clients

- Fixed system size, grow the number of clients (N-to-N, Panasas DirectFlow)



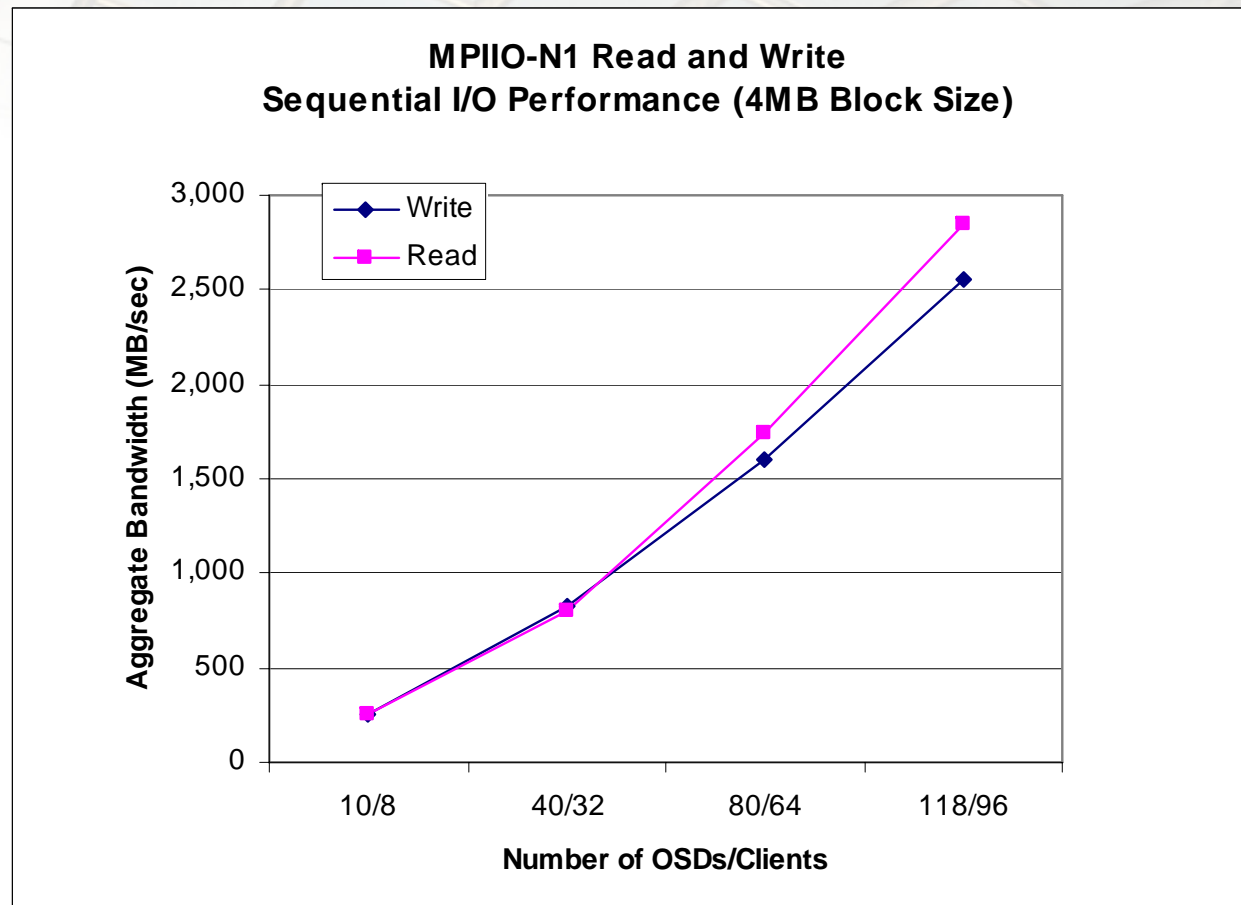
Scaling NFS

IOZone over Panasas NFS (each shelf is 3 Directors + 8 Storage Blades)



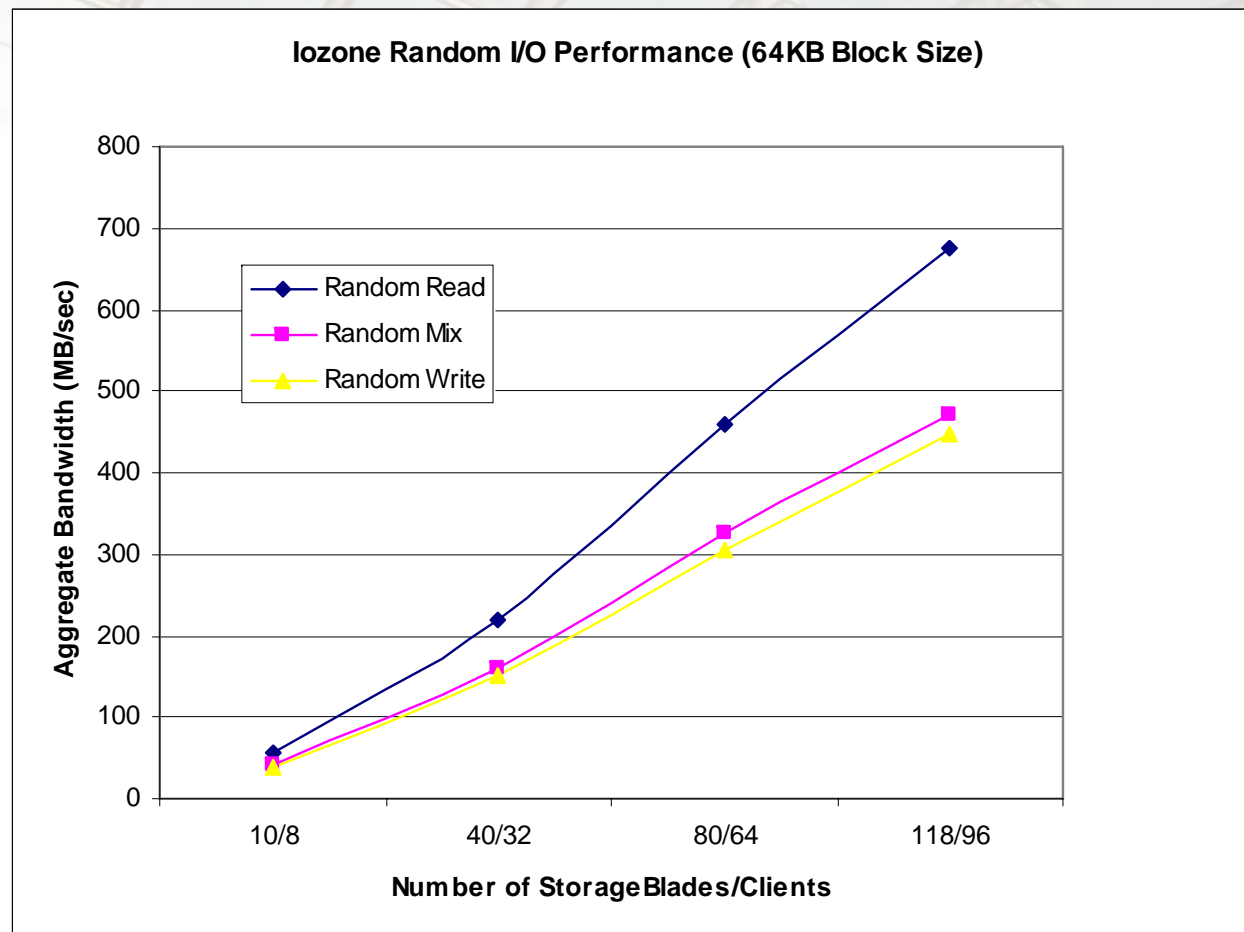
Concurrent Read/Write Performance

- N-to-1 read and write performance (MPI-IO Panasas Direct Flow)



Random IO

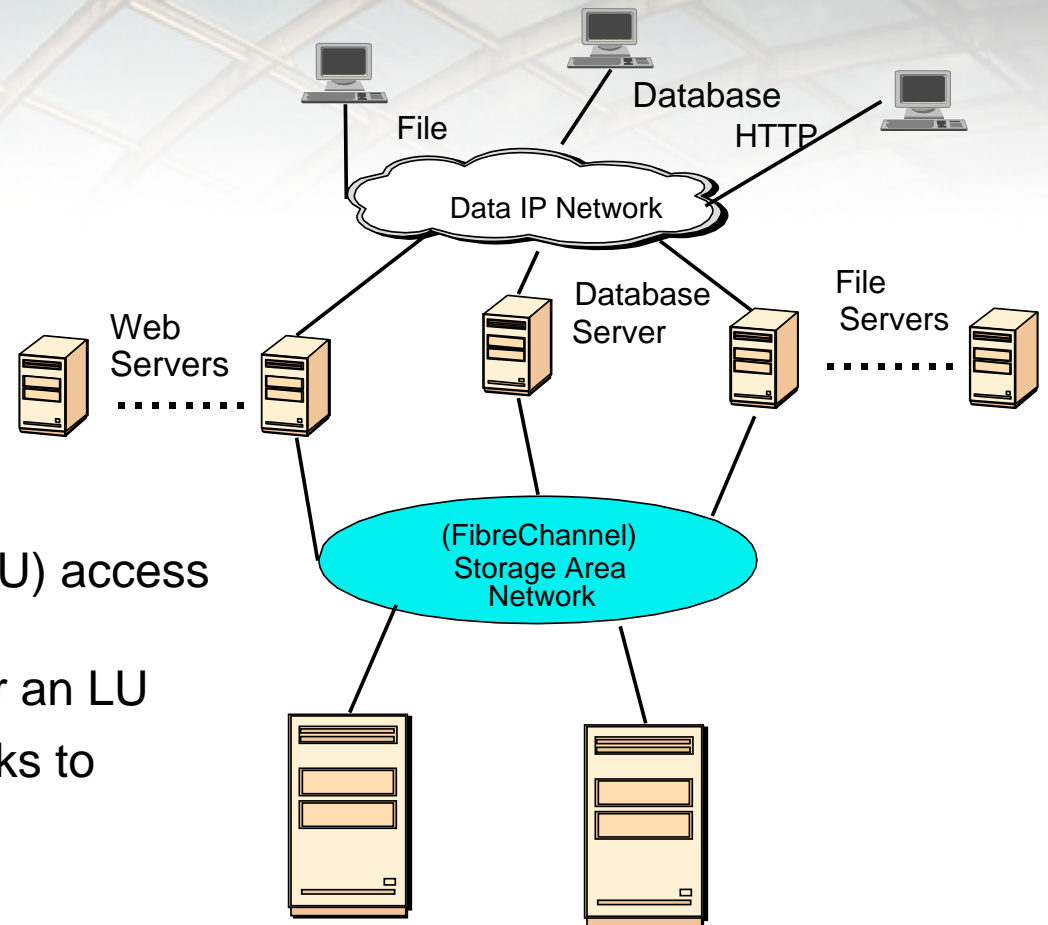
IOZone over Panasas Direct Flow



StorageBlade = OSD

The Need for Storage Security

- SAN Security Today:
 - Essentially doesn't exist
 - Assume only trusted clients
- Work arounds
 - Zoning/Fencing
 - Hard to use
 - Physical level
 - LUN Masking/Logical Unit (LU) access controls
- At best, provide all or nothing for an LU
 - Too many actively used blocks to provide block-level security



Security Slides courtesy of the SNIA/T10 OSD Working Group, www.snia.org

Object Store Security Goals

- Increased protection/security
 - Fine-grained protection on objects rather than LU
 - Hosts do not access/modify metadata directly, hidden w/in OSD interface
- Allow non-trusted clients to sit on SAN
- Allow shared access to storage without giving clients access to all data on volume
- From the OSD ver1.0 Roadmap value proposition:
 - Robust, shared access by many clients, OS's
 - Strong, fine-grain, end-to-end security
 - Scalable performance via offloaded data path
- Prevent attacks on individual objects
- Prevent network attacks
- Must not duplicate the cost of security – layered approach
 - Provide a stand-alone solution that works without a network security infrastructure
 - Provide a solution that leverages standard network security infrastructures.
- Allow low cost implementation of the critical path
 - Allow efficient implementation on existing network transports

Concepts

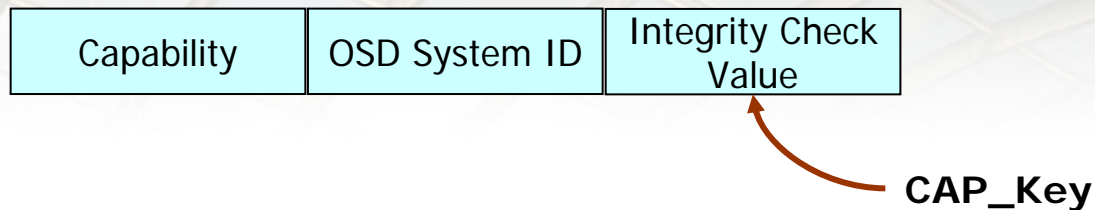
- **capability:** The fields in a CDB that specify what command functions the command may request and on what objects. A capability (CAP) is included in every request to an OSD.
- **credential:** A capability that is protected by an integrity check value that is sent to an application client in order to grant defined access to an OSD
- **security manager:** The component of an OSD configuration that manages secret keys and prepares secure credentials containing capabilities thus granting application clients specified access to a specified OSD logical unit.
- **integrity check value:** A value computed using a security algorithm (e.g., HMAC-SHA1), a secret key and an array of bytes.
- **capability key:** The integrity check value of the capability that is used by an application client to compute integrity check values for a single OSD command.

Capability Structure

Key version	Alg.	Sec. Method	Expiry	Audit	Discriminator	Obj Creation time	Obj desc	Permissions
-------------	------	-------------	--------	-------	---------------	-------------------	----------	-------------

- Key Version – identifies secret key for integrity check
- Algorithm – algorithm used for integrity check
- Security Method - Verified that security method is allowed is at the device
- Expiry – expires a credential, allowing different lifetimes for credentials
- Audit – allows Manager to associate the capability with a specific client
- Capability Discriminator – ensures all credentials are unique (aka nonce)
- Creation time – creation time of the object
 - Distinguished between objects with the same Object ID
- Permission Bit Mask – operations the client is entitled to perform
 - Multiple operations can be set {read, write, create, delete, get-, set-attributes, ...}
- Object Descriptor – Partition ID, Object ID and Policy tag
 - Policy Tag
 - Version Tag - Used to invalidate all credentials for an object
 - Fence – allows OSD to prevent access to an object

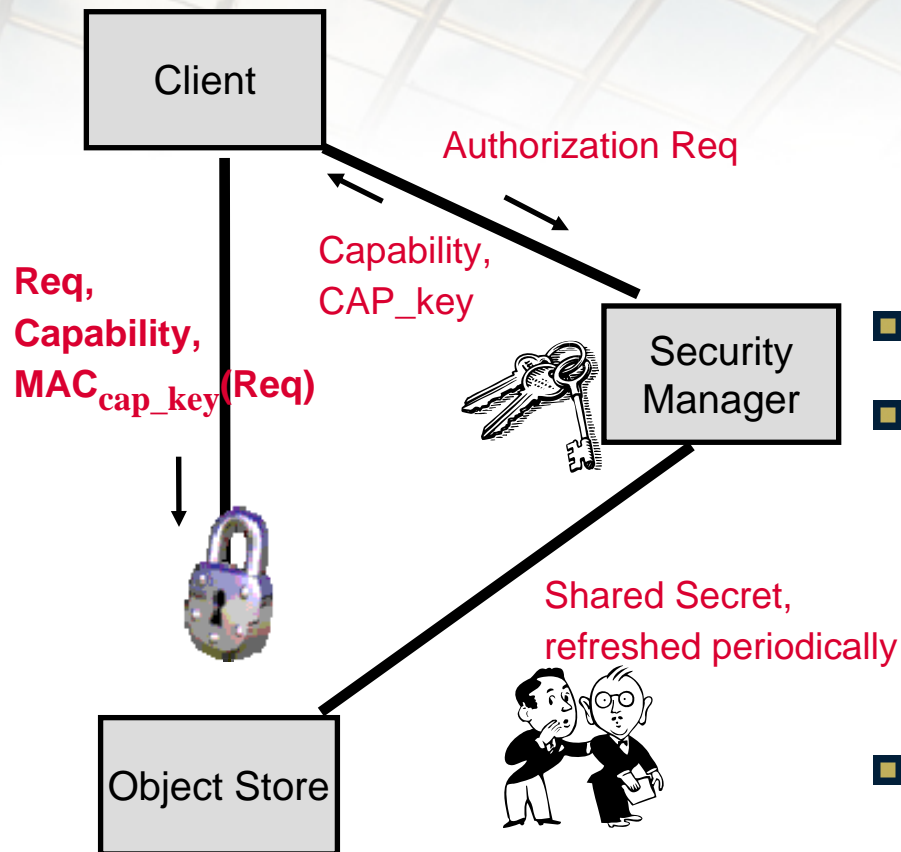
Credential Structure



- **Secret Key** is the key shared between the Manager and Client
- A **Credential** is derived from the capability arguments.
 - $CAP_Key = \text{MAC}_{\text{secret key}}(\text{CAP_Args}, \text{OSD System ID})$
 - It holds a signature over the capability using the device secret key
 - That signature is used as another signing key over OSD commands
 - OSD can unwrap the two levels of signatures to verify commands

*Audience:
Wake up
Here*

Basic Security Model



- All operations are secured by a capability
 - Is the command valid?
 - Is the command allowed to access the specified object ?
- Manager and OSD are trusted
- Security achieved by cooperation of:
 - Manager – authenticates/authorizes clients and generates credentials.
 - OSD -- validates credential that a client presents.
- Credential is cryptographically hardened
 - OSD and Manager share a secret

Security Methods

- NOSEC
 - No security but capability must allow required commands
 - Can prevent some mistakes, e.g., accessing wrong object
- CAPKEY
 - Secure (i.e., signed) capability
 - When used with a secure network provides security for entire exchange
- CMDRSP
 - Secure capability and command but not data
- ALLDATA
 - Secure capability, command and data
 - Provides security for entire exchange when network is not secure
 - Duplicate work if the network is secure

Security Methods

Threat	Threat thwarted by security method				
	NOSEC	CAPKEY		CMDRSP	ALLDATA
		Over secure channel ^a			
		No	Yes		
Forgery of credential	No	Yes	Yes	Yes	Yes
Alteration of capabilities	No	Yes	Yes	Yes	Yes
Use of credential by unauthorized application client	No	Yes ^b	Yes ^c	Yes	Yes
Replay of command or status	No	No	Yes ^c	Yes	Yes
Alteration of command or status	No	No	Yes ^c	Yes	Yes
Replay of data	No	No	Yes ^c	No	Yes
Alteration of data	No	No	Yes ^c	No	Yes
Inspection of command, status or data	No	No	Yes/No ^d	No	No

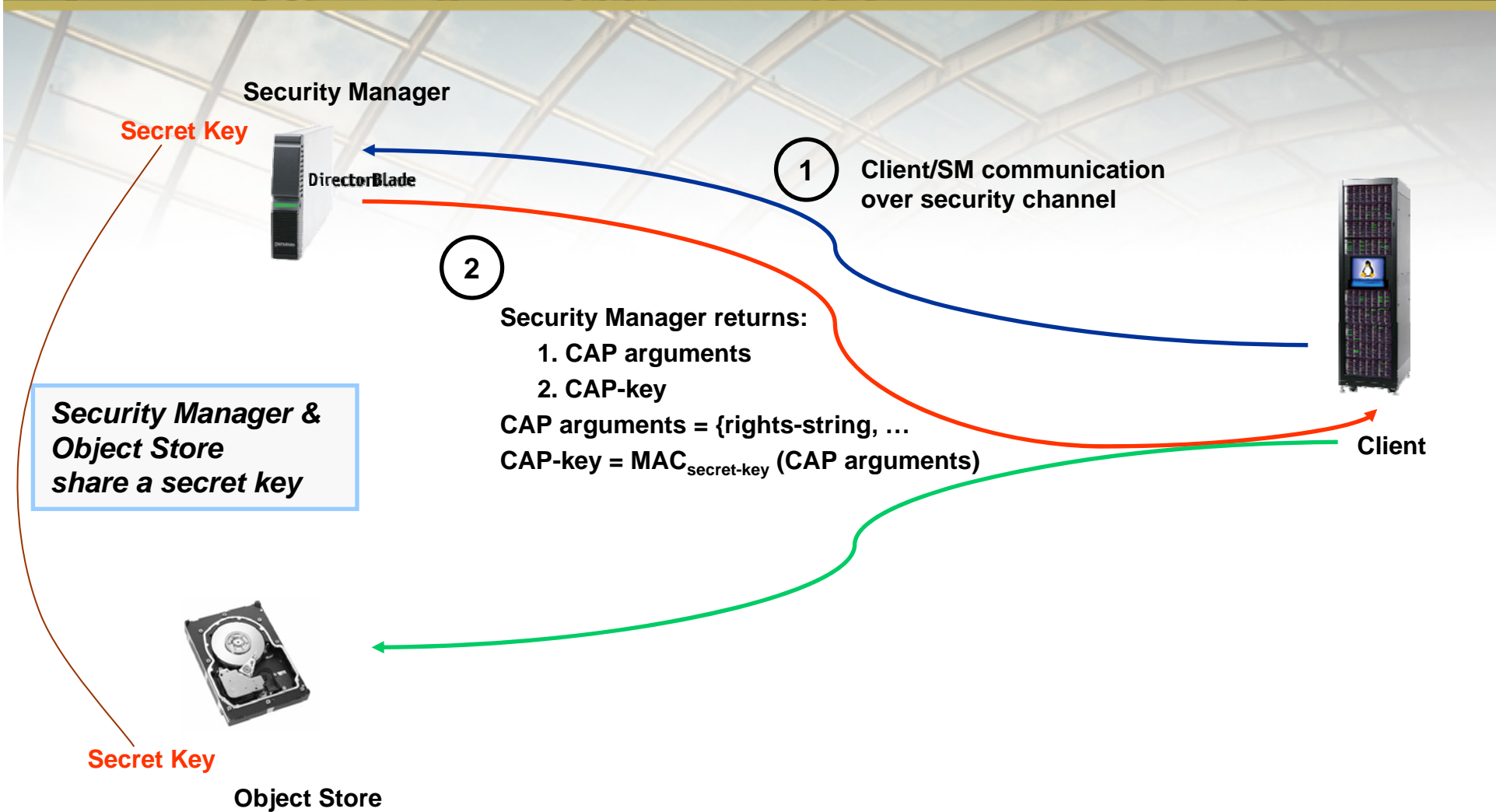
OSD Security Misc

- Capabilities are not bound to a specific client
 - Allow clients to delegate capability
- Key Management
 - SET_MASTER_KEY
 - DH key exchange
 - SET_KEY
 - Create working key N as a function of the master key

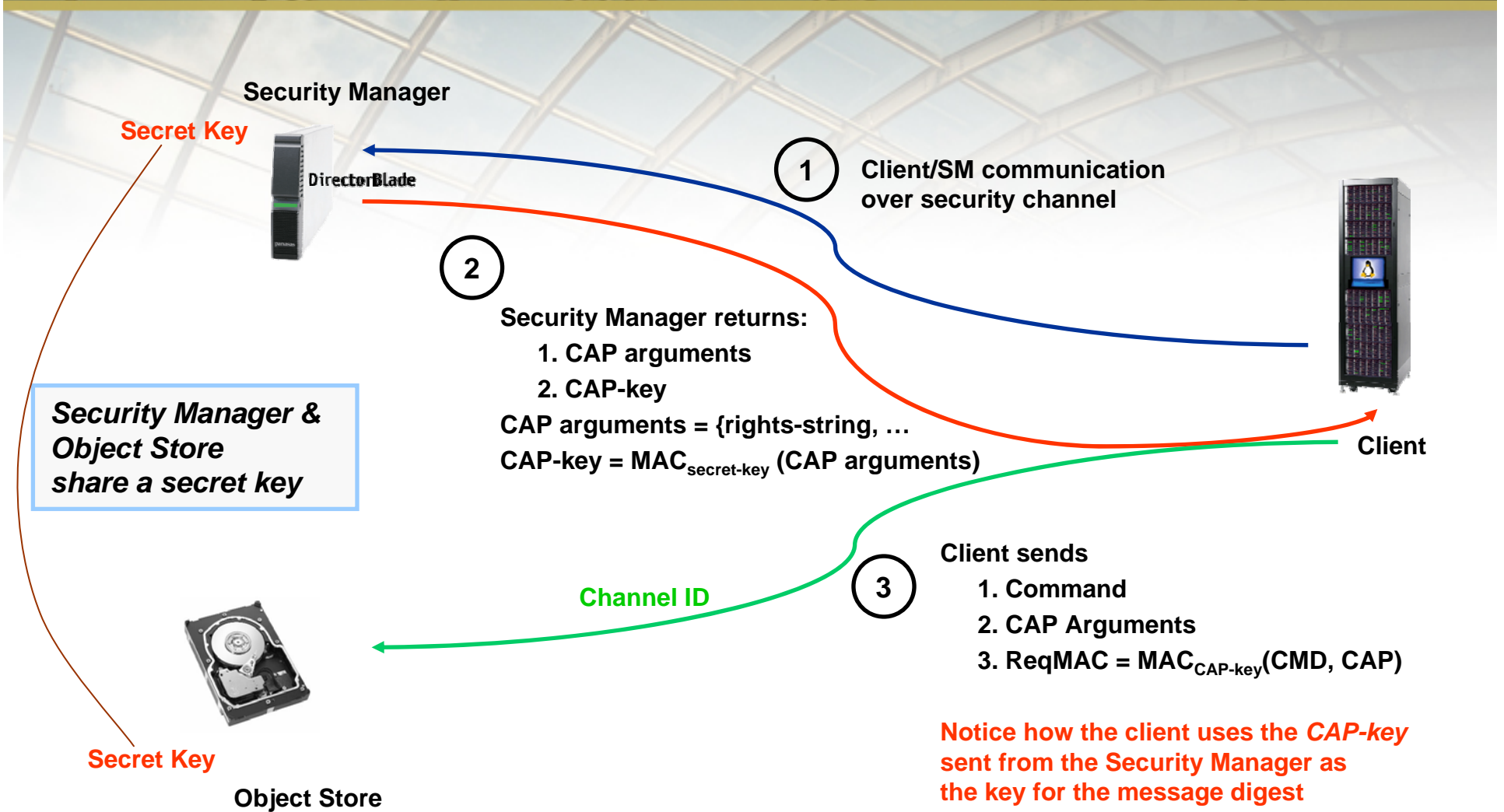
Table 6 — Capability format

Bit Byte	7	6	5	4	3	2	1	0
0	Reserved				CAPABILITY FORMAT (1h)			
1	KEY VERSION				INTEGRITY CHECK VALUE ALGORITHM			
2	Reserved				SECURITY METHOD			
3	Reserved							
4	(MSB)	CAPABILITY EXPIRATION TIME				(LSB)		
9								
10	AUDIT							
29								
30	(MSB)	CAPABILITY DISCRIMINATOR				(LSB)		
41								
42	(MSB)	OBJECT CREATED TIME				(LSB)		
47								
48	OBJECT TYPE							
49								
53	PERMISSIONS BIT MASK							
54	Reserved							
55	OBJECT DESCRIPTOR TYPE				Reserved			
56								
79	OBJECT DESCRIPTOR							

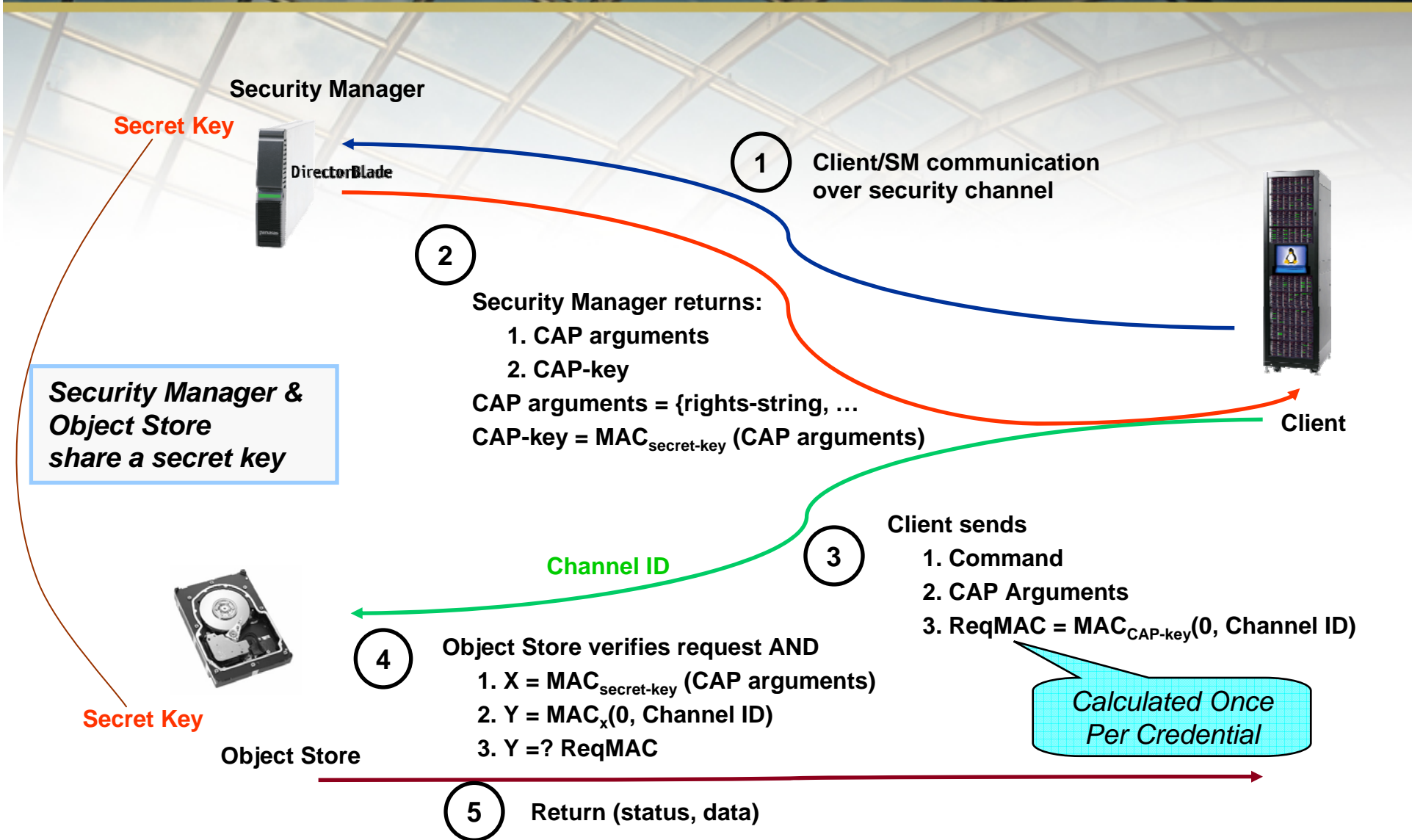
The CAPKEY Mode



The CAPKEY Mode



The CAPKEY Mode



Storage System Examples

- Panasas
- Lustre
- IBRIX
- GPFS
- Isilon
- NetApp-GX
- Seagate OSD

Panasas

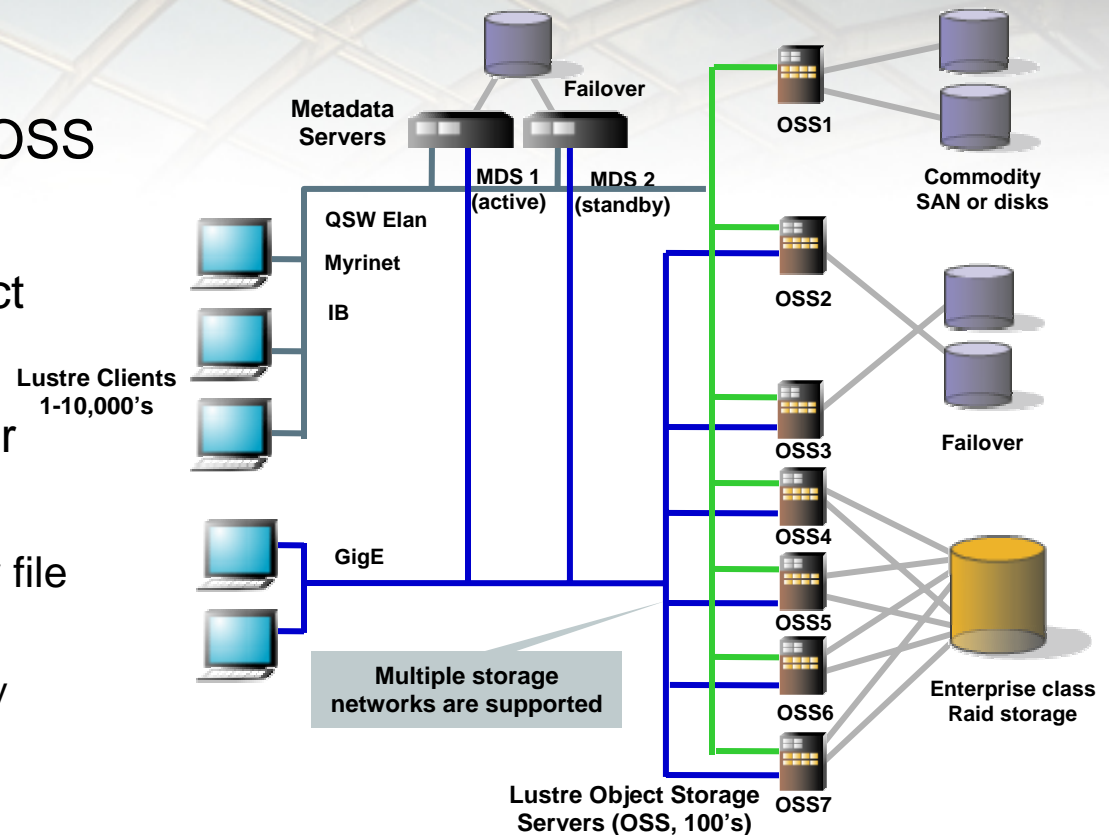
- Distributed file system layered over object storage
 - Linux RPM file system client
 - `mount -t panfs panfs://server:global /panfs`
- Scalable, per-file RAID with tunable parameters
- Metadata clustering (coarse grain), client cache consistency
- Large, high-performance storage clusters
 - 10 to 500 OSDs, >5000 clients, >10 GB/sec demonstrated
- Blade hardware architecture
 - StorageBlade is an OSD
 - DirectorBlade is metadata server and NAS head
 - Shelf enclosure has integrated network switch, power, battery backup

Lustre

- Open source object-based storage system
 - Based on NASD architecture from CMU
 - Lots of file system ideas from Coda and InterMezzo
- Key design features
 - OSD (not T10). OSS (server) hosts multiple OST (targets)
 - Exotic network support (enhanced Sandia Portals stack)
 - RAID from block storage behind OST
 - RAID-0 striping across OST
 - OSS fail over if OST is dual-ported
 - Single metadata server with manual fail over to backup
 - Distributed lock protocol among clients, OSS, and MDS

Lustre High Level Architecture

- Direct client-storage communication
- Clients communicate with OSS (Object Storage Servers)
 - Also known as OST (Object Storage Targets)
 - Data is striped RAID 0 over set of OSS's
 - Stripe width determined by file bandwidth requirements
 - Concurrent access typically requires wider stripes
 - Data reliability (RAID 1,5) is managed by RAID controllers in OSS or RAID array



Lustre material from from www.lustre.org and various talks

Lustre OSS

- Does not follow T10 OSD standard
- OST is transactional – two-phase reply to enable recovery
 - OST is built on top of other file systems (e.g., ext3)
- Objects can belong to several groups
 - Allows OST to log information about object (e.g., marked for deletion)
- Command differences
 - Open and close – Lustre maintains ref counts
 - Preallocate – creates N objects, used for efficient create
 - Read and write – support scatter-gather lists
 - Sync – can specify range of object to sync
 - Migrate – move a data range from one object to another on the same OST
 - Iterate – OST applies function to all objects within a group
 - Lock_{enqueue, convert, cancel}

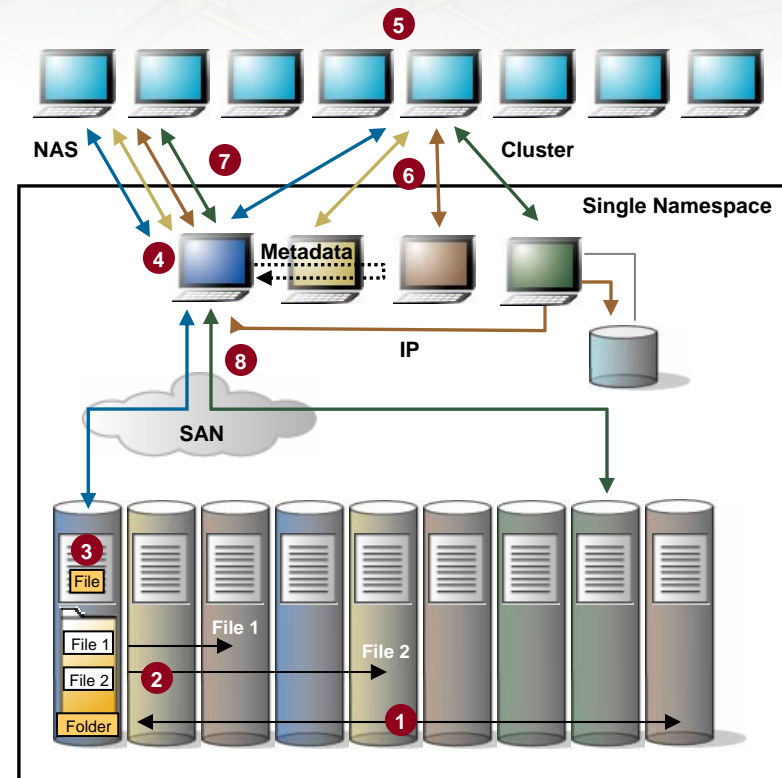
Lustre material from www.lustre.org and various talks

Lustre Performance

- OSS
 - single GE link, 110 MB/sec
 - dual GE link, 220 MB/sec
 - 10 GE link, 550 MB/sec
- Testing across 64 OSTs fronting 8 racks of DataDirect S2A8500 storage (estimate about 500 drives)
 - network max is 6.4GB/sec Ethernet)
 - Write file-per-process (FPP) 4.5 GB/sec (72 MB/sec OSS)
 - Read FPP is ~2.5 GB/sec (40 MB/sec OSS)
- 11 GB/sec over 104 single-GE OSS
 - www.clusterfs.com/faq.html#io-3
 - 108 MB/sec per OSS

IBRIX

- Not an object-based storage system, but draws on decoupled data/metadata architecture
- Segments instead of objects
 - File can span one or more segments (storage devices)
- Hardware RAID / SAN
 - RAID under the segments
- NFS and CIFS servers running IBRIX software
- Client can run IBRIX driver for direct storage access
- Metadata owning Segment Servers can take ownership of metadata or proxy requests



IBRIX Performance

- Dell PowerEdge 1750 Xeon processor (3 GHz) as segment servers
 - Dual GE
- EMC CX7000 storage system
 - Eighty 10K RPM FC drives, 16 5-drive RAID-3 LUNs
- Write scaling a function of RAID controller configuration
- www.ibrix.com/dell_saify.pdf

	READ	WRITE
1 server 2 clients	162 MB/s	131 MB/s
2 servers 4 clients	319 MB/s	250 MB/s
4 servers 8 clients	626 MB/s	445 MB/s
8 servers 16 clients	1123 MB/s	487 MB/s

GPFS

■ IBM SAN File System

- Symmetric design originally, extended to have storage-owning nodes
 - Built into the compute cluster
- Originally specific to SP3 architecture, now has Linux version
- Lock manager allows locks on data ranges, individual attributes, parts of directories.
 - One central token manager per file system. (Recent versions address this bottleneck.)
- RAID a property of back-end SAN storage
- Large block sizes (e.g., 256K or even 4MB) alleviates metadata overhead

Isilon

- OneFS file system
 - Per-file RAID, scalable reconstruction
 - NFS export, or proprietary file system driver
 - Storage nodes with approx 12 drives
 - Cache nodes with file service and cache, no drives
 - 70 MB/sec throughput per node (1 GE?)
 - Scaled to 1 GB/sec with 20TB system
 - At least 10 nodes based on 1.9 TB limit, probably 14 nodes
 - www.isilon.com/pdfs/white_papers/Clustered_Storage_Revolution.pdf

NetApp-GX

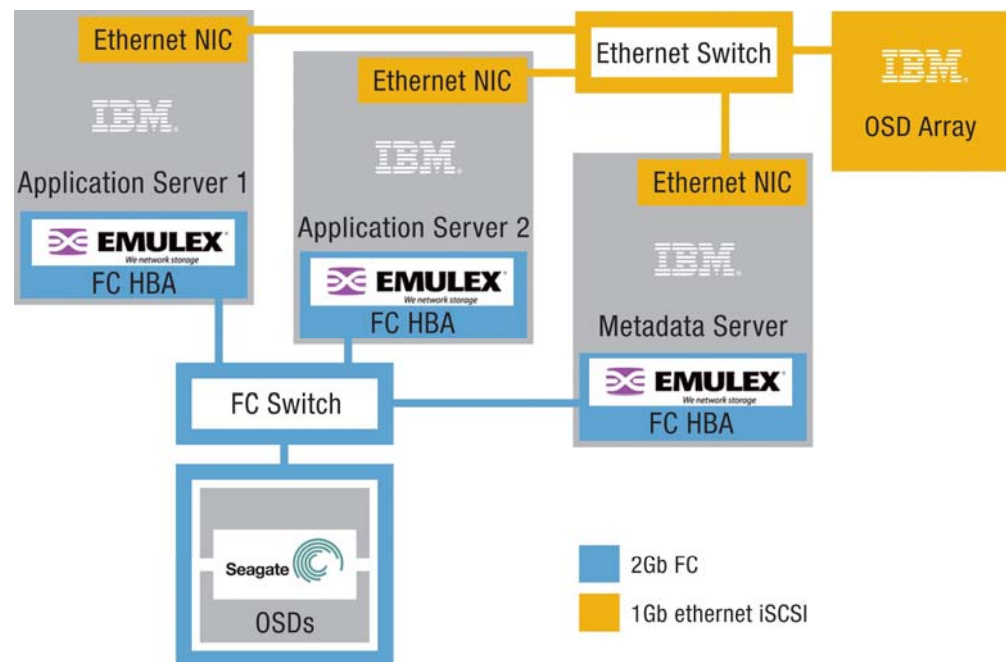
- NAS Cluster with forwarding model
 - Nodes own virtual volumes (i.e., quota tree)
 - Cache miss forwarded to owner
 - Ownership can be remapped manually to change load
 - Spinnaker design married to NetApp RAID engine

Seagate, IBM and Emulex OSD V1

Emulex, IBM & Seagate Team Up to Demonstrate Industry's First Standards-Compliant Object-Based Storage Devices

- OSD Technology Demonstration Signals Move Toward More Powerful, Intelligent and Simplified Storage

- Demonstration: video files written into file system & played
 - Files were written to explicit devices by OSD system
 - All in a common file system
 - File system did not know or care where the files were located



Seagate/IBM OSD: Lessons

- First prototype – not high performance
 - Block architecture based HDD
 - First iteration of OSD implementation
- Can be made very competitive
 - Some small changes to HDD required
 - Working set > OOM smaller than host based file system
 - Big savings in I/O communication time
- HDD OSD order of magnitude < Host FS working set
 - Huge difference in path lengths
 - Huge difference in memory requirements
- Similar to drive-based IO management
 - Host consumes MB's to manage I/O stream
 - Drive manages same stream in 10's of KBs (excluding buffer)
 - Constrained, tailored environment

References

- <http://www.pdl.cmu.edu/NASD>
- <http://www.t10.org/scsi-3.htm>
- <http://www.t10.org/ftp/t10/drafts/osd>
- <http://www.intel.com/technology/computing/storage/osd/>
- <http://www.panasas.com>
- <http://www.lustre.org/>
- http://www.seagate.com/docs/pdf/whitepaper/tp_536.pdf
- [http://www.snia.org/education/tutorials/spr2005/storage/Object-basedStorageDevice\(OSD\)Basics.pdf](http://www.snia.org/education/tutorials/spr2005/storage/Object-basedStorageDevice(OSD)Basics.pdf)

References

- **“NFS Version 4 Minor Version 1”**
draft-ietf-nfsv4-minorversion1-05.txt
- **“pNFS Problem Statement”**
Garth Gibson (Panasas), Peter Corbett (Netapp), Internet-draft, July 2004,
<http://www.pdl.cmu.edu/pNFS/archive/gibson-pnfs-problem-statement.html>
- **“NFSv4 pNFS Extensions”**
G. Goodson (Netapp), B. Welch, B. Halevy (Panasas), D. Black (EMC), A. Adamson (CITI), Internet-draft, October 2005,
<http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-00.txt>
- **“Linux pNFS Kernel Development”**
CITI,
<http://www.citi.umich.edu/projects/asci/pnfs/linux/>