Object-Based Cluster Storage Systems

Brent Welch & David Nagle
{welch,dnagle}@panasas.com
Panasas, Inc

New 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:

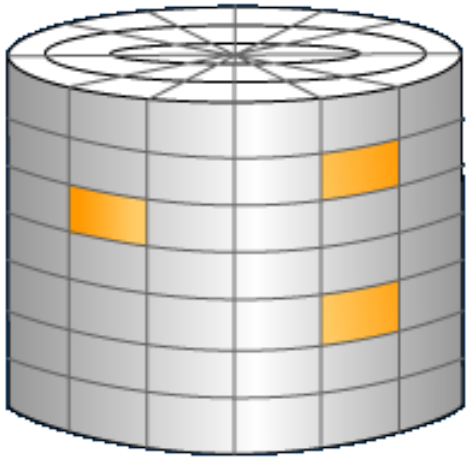
Read block
Write block

Addressing:

Block range

Allocation:

External



Object Based Disk

Operations:

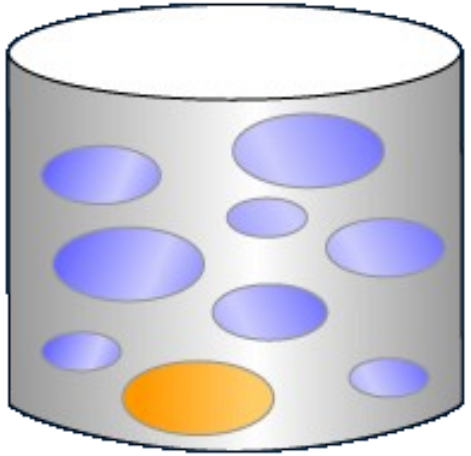
Create object
Delete object
Read object
Write object
Get Attribute
Set Attribute

Addressing:

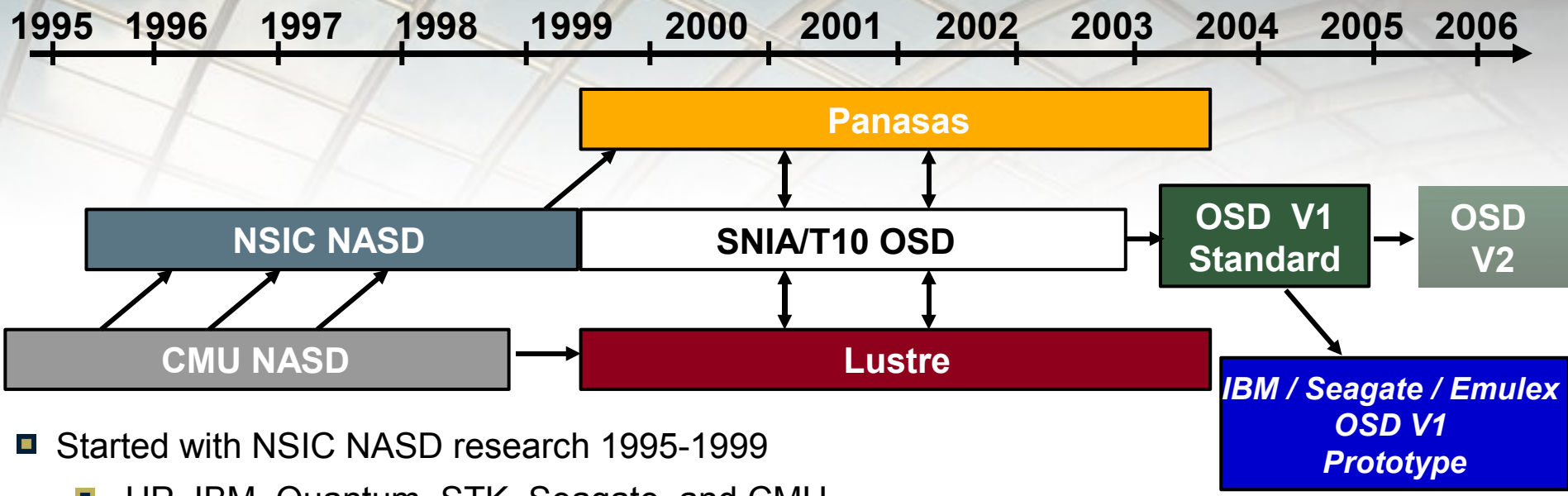
[object, byte range]

Allocation:

Internal



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

About Us

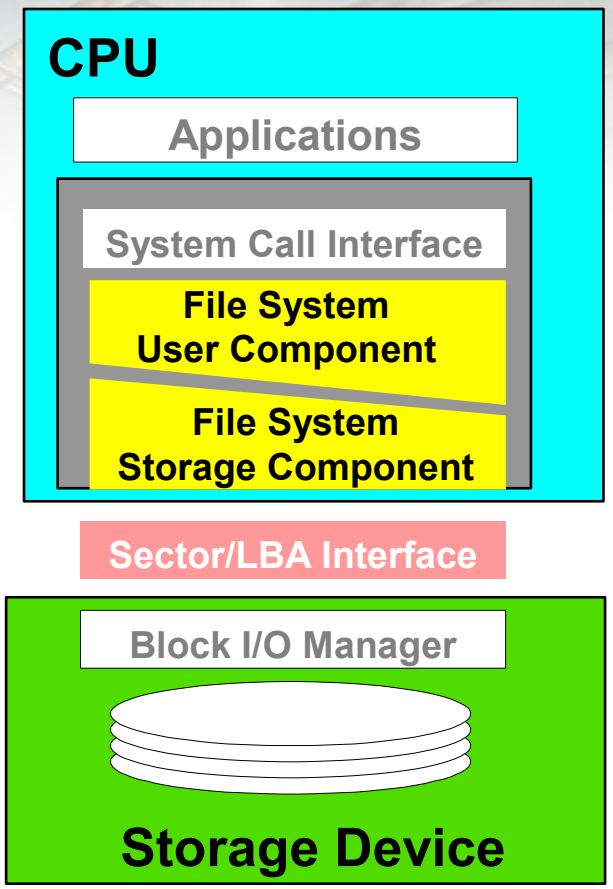
- Brent Welch (welch@panasas.com)
  - Director of Architecture, Panasas
  - Berkeley Sprite OS
  - Authored definitive TCL book
  - IETF pNFS
- David Nagle (dnagle@panasas.com)
  - Advanced Development Architect, Panasas
  - Parallel Data Lab, CMU
    - NASD, Active Storage Networks and MEMS-based storage projects
  - ANSI T10 OSD V1.0 and V2.0 standards

Agenda

- Intro
- Objects and the T10 OSD Standard Interface
- Scalable Storage System Architectures
  - Files over Objects
  - File Systems that support direct client-storage communcations
  - Performance
- OSD Security
- Example Systems
  - Lustre
  - IBRIX
  - EMC Centera
  - pNFS

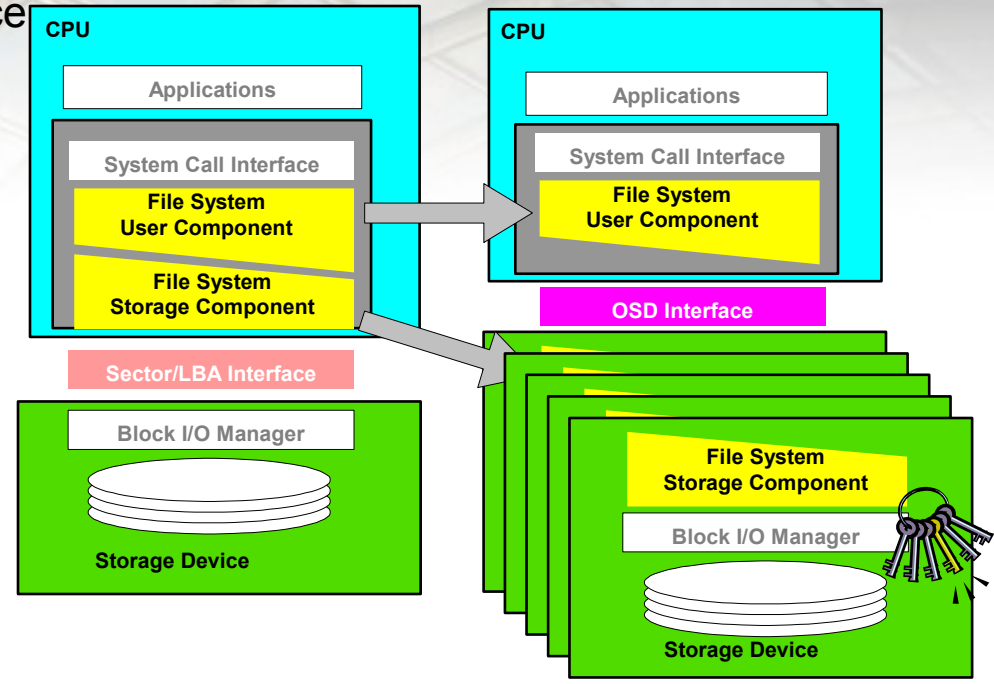
Review: Today's File System/Storage Protocol Stack

- OS / File System
  - OS performs file system policy checks
    - User authorization
    - Permission checks (read, write)
    - File attributes
      - Physical size, logical length, timestamps (last access time, last modified time), quota
  - Translates file to sector mapping
    - OS is responsible for performance of storage device
      - Layout, organization of storage requests
      - Prefetching to the clients
    - 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
    - Next best way, layout contiguous data on contiguous sectors/tracks



Object-based Storage (OSD)

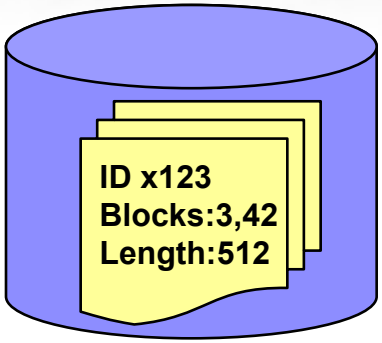
- At 10,000 feet, migrate the lowest part of the client file system into the storage device
  - Storage understands logical data layout and not just sectors
- Storage interface evolves from blocks to “objects”
  - Re-divides responsibility for managing the access to data
  - Assigns additional activities to the storage device (space management)
- Offloads inode processing to OSD
  - Management of block to file mapping distributed to storage device
- 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)



Objects

- An object is a logical unit of storage
  - Lives in (almost) flat name space – addressed via object ID
  - Contains application data AND **metadata**
    - 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

Object attributes

Table 76 — User Object Information attributes page contents

Attribute Number	Length (bytes)	Attribute	May Be Set	OSD Logical Unit Provided
0h	40	Page identification	No	Yes
1h	8	Partition_ID	No	Yes
2h	8	User_Object_ID	No	Yes
3h to 8h		Reserved	No	
9h	variable	Username	Yes	No
Ah to 80h		Reserved	No	
81h	8	Used capacity	No	Yes
82h	8	User object logical length	Yes	Yes
83h to FFFF FFFEh		Reserved	No	

size

length

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

Read – 0x8805

- What do commands look like?
  - SCSI extended CDB (cmd desc block of up to 250 Bytes)
  - First 7 bytes of CDB have special code to specify extended CDB

Table 58 — READ command

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8805h)				(LSB)
10				OPTIONS BYTE				
11	Reserved			GET/SET CDBFMT		Reserved		
12				TIMESTAMPS CONTROL				
16	(MSB)							
23				PARTITION_ID				(LSB)
24	(MSB)							
31				USER_OBJECT_ID				(LSB)
32								
35				Reserved				
36	(MSB)							
43				LENGTH				(LSB)
44	(MSB)							
51				STARTING BYTE ADDRESS				(LSB)

64 bits

64 bits

byte addressable

List – 0x8803

- List command allows retrieval of all object IDs w/in a partition
  - Necessary for recovery

Table 49 — LIST command

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8803h)				
9								(LSB)
10	Reserved							
11	Reserved		GET/SET CDBFMT			SORT ORDER		
12	TIMESTAMPS CONTROL							
16	(MSB)			PARTITION_ID				
23								
24	Reserved							
31								
32	(MSB)			LIST IDENTIFIER				
35								
36	(MSB)			ALLOCATION LENGTH				
43								
44	(MSB)			INITIAL OBJECT_ID				
51								(LSB)

only one option – ascending object id

continuation across LISTs

buffer size available

Commands Can Read/Write Attributes

- To minimize number of commands (messages), almost every command can read and write the objects attributes
  - Read (object=0x1234, offset=0, length=100bytes, readAttr=physicalSize)

Table 33 — Page oriented get and set attributes CDB parameters format

Bit / Byte	7	6	5	4	3	2	1	0
51	⋮ Other CDB fields							
52	(MSB)			GET ATTRIBUTES PAGE				
55								
56				GET ATTRIBUTES ALLOCATION LENGTH				
59								(LSB)
60				RETRIEVED ATTRIBUTES OFFSET				
63								
64				SET ATTRIBUTES PAGE				
67								(LSB)
68	(MSB)			SET ATTRIBUTE NUMBER				
71								(LSB)
72				SET ATTRIBUTE LENGTH				
75								(LSB)
76				SET ATTRIBUTES OFFSET				
79								
80	⋮ Other CDB fields							

which attrib

how much buffer host has available

which attrib

Num of attribs bytes being sent

T10 OSD V2.0 Work in Progress

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

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

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
  - Deleted every user object listed in the collection
- QUERY
  - Match against one or more specified <attribute, value> pairs, returning the list of a user objects that successfully matched
- Ordering of internal command processing is unspecified
  - Allows for efficient disk-directed processing
  - But how do you restart a command?

Error Handling – Multi-object Operations

- Multi-object operation may fail in the middle
  - e.g., REMOVE MEMBER OBJECTS dies ½ way through removing all of the objects
  - Recovery solutions
    - Collection's list of objects is ordered and client is informed of where in the list the command died (tricky if not impossible)
    - As member objects are removed from the collection, their entries are deleted from the collection
      - Restart is as simple as replaying the command again
  - Use same trick for other multi-object operations
    - Before performing multi-object operation, clone the collection object
    - Operate on the collection object, removing entries as objects are processed
      - Eliminates need for identifying position where failure occurred
      - Allows OSD to process objects in any order
        - Efficient head/resource scheduling opportunity

Error Handling – Damaged Objects

- Objects can be damaged for several of 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

Error Handling – Rebuilding Damaged Objects

- Damaged object may be repairable by OSD itself or may require outside intervention (e.g., damaged object is to be repaired by rewriting via RAID when RAID is done above the OSD interface)
  - Internal repairs done either automagically or with explicit FSCK command
- Reconstructing a file could be as simple as re-reading all surviving objects
  - But what about optimizations under the OSD interface (e.g., space efficient holes)
  - Requires outside entity to know some details of the object
    - Repair entity requests map of object's {holes, damaged regions}
    - Using the map, the outside entity can efficient repair the holes or selectively repair only the damaged regions
  - READDIFF and READMAP

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

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

Wide Variety of Object Storage Devices

- Disk array subsystem
- Ie. LLNL with Lustre

- “Smart” disk for objects
- 2 SATA disks, CPU and GE NIC

- Prototype Seagate OSD
- Highly integrated, single disk

Interfaces: Blocks, Files and Objects

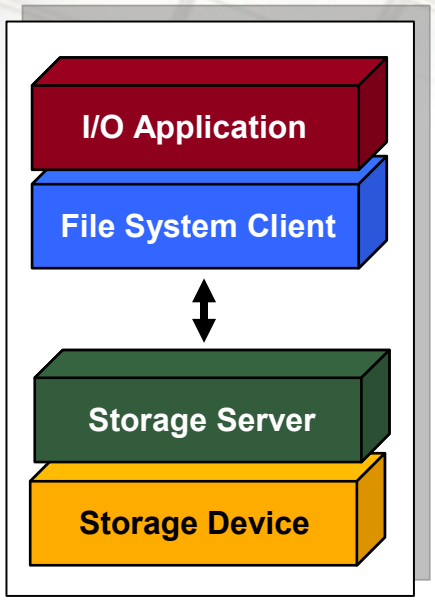
- Storage Interface
  - Block-based architecture: fast but private
    - Traditional SCSI and FC approaches
    - Expensive fabric, difficult to share between hosts
    - Lack of security
    - Large amount of metadata to describe layout of files
  - File-based architecture: sharable, but bottlenecked 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

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

Out-of-Band OSD System Architecture

Object-Based Architecture
Distributes **the System**



Clients

DirectorBlade
(aka Manager)

I/O Application
File System Client

Control

File System Policy
OSD Security Protocol

Data

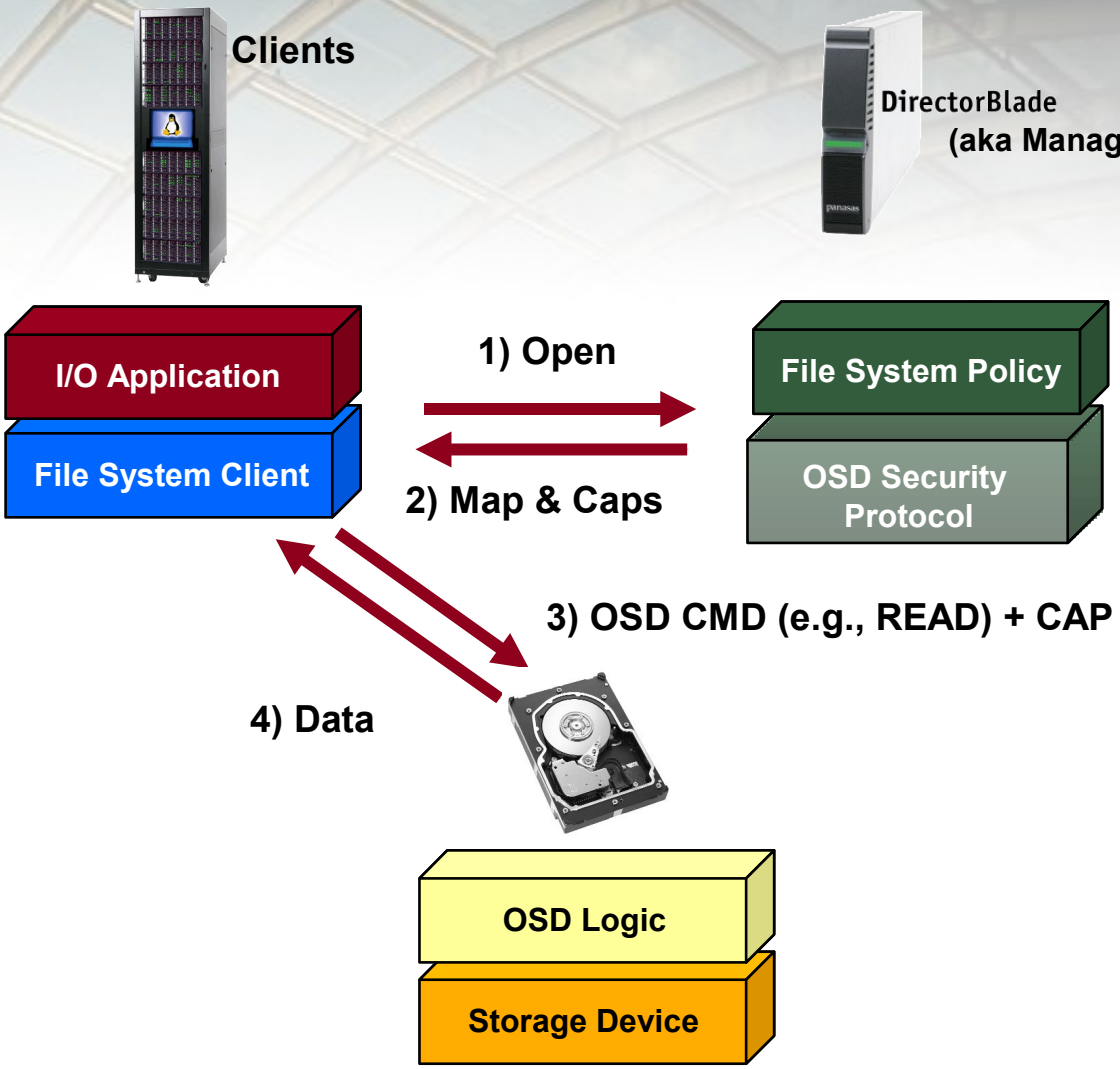
Configuration

OSD Logic
Storage Device

File manager is not in the data path

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



Objects as Building Blocks

Object

Comprised of:

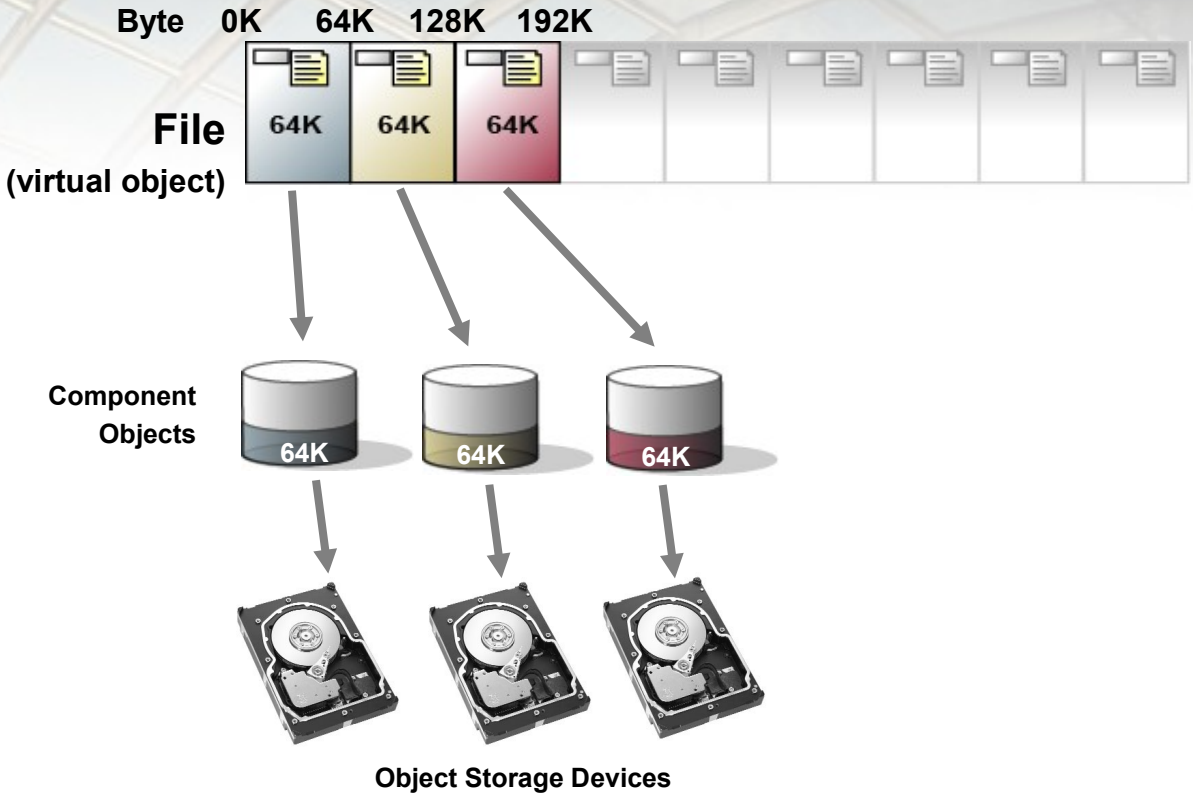
- User Data
- Attributes
- Layout

Interface:

- ID <dev,grp,obj>
- Read/Write
- Create/Delete
- Getattr/Setattr
- Capability-based

File Component:

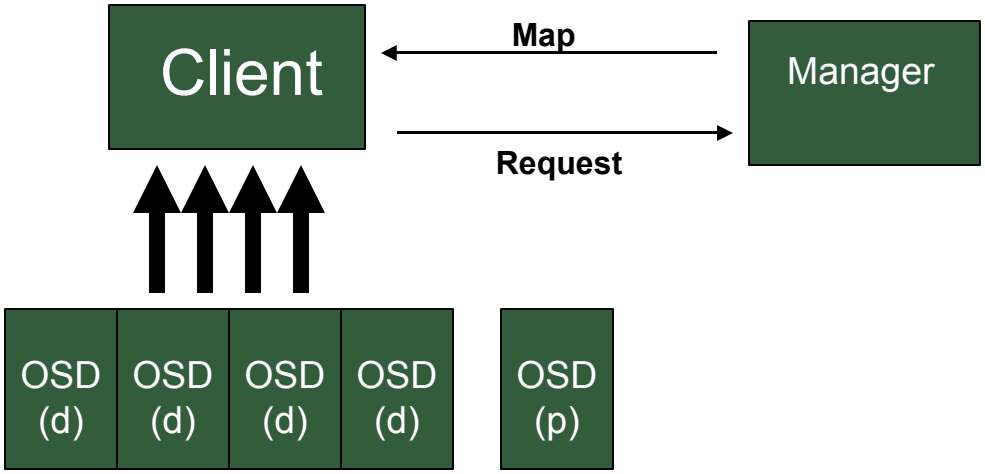
- Stripe files across storage nodes
- 1 File == 1 Virtual Object striped across 1 or more physical objects



By striping a file across multiple OSDs, the application(s) can realize the aggregate bandwidth of all OSDs in the system, including 1) Bandwidth, 2) Load balancing, 3) Reliability – RAID

Example: Client Read Operation

- Steps in a client read operation
  - Client determines file ID and responsible director from the directory entry
  - Client requests permission to read from the director (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 director

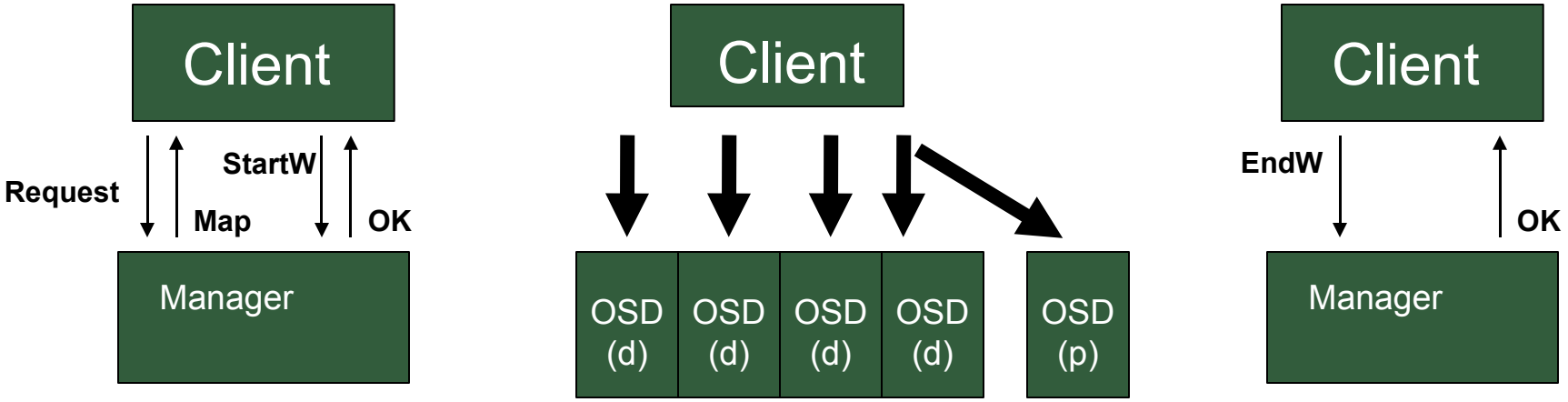


Client Read Operation

- Different files can use different sets of OSDs
- Bandwidth is achieved via parallel transfers from multiple OSDs
  - More OSDs ⇒ higher parallelism ⇒ 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
- Distinct from reads in that permission to read is long-lived



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)
    - Object and 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

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

Optimizing an object-based file system

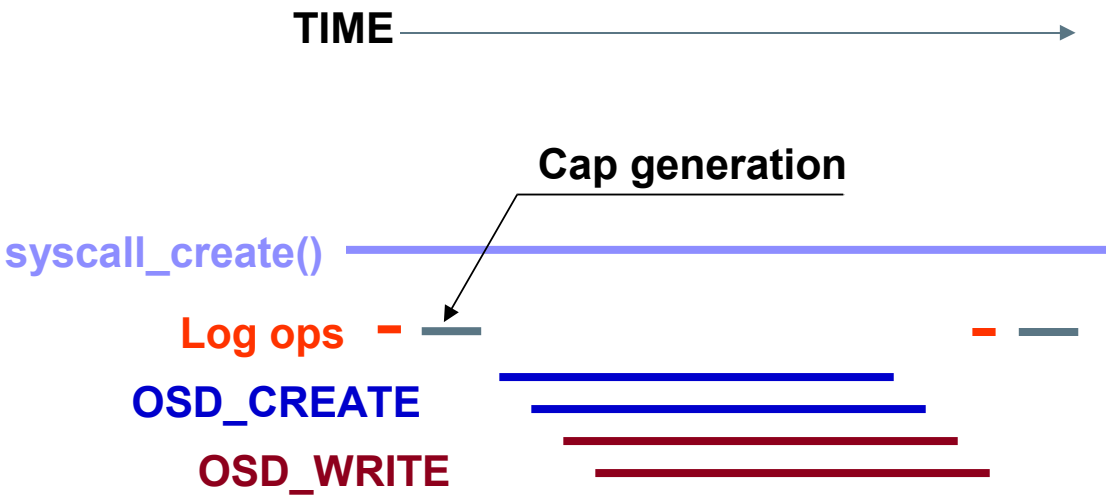
- Distributed cache consistency (not a new problem)
- Parallel create for efficiently creating redundant files
- High performance crash recoverability via logging
- Snapshots
- Reconstruction and object movement with snapshots
- OSD delayed block allocation and read ahead

Caching and Cache Consistency

- Caching information on client reduce network traffic
  - Clients cache file data, directory data, attributes, and capabilities
  - Clients cache both clean (read) and dirty (written) data
- Managers keep “callbacks”
  - Client indicates interest in file by registering callback
  - Director promises to notify the client if cached data or attributes become invalid, or other client wants to access file, via “callback break” message
  - 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

Example: Parallel Create

- Create a mirrored file (2 objects) and insert into directory
- 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



Metadata journaling

- Operations across multiple objects must be recoverable
  - OSD 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)
  - manager disk, object storage (.5 to 5 msec)

Snapshots

- Copy-on-write objects
  - CLONE_OBJECT and CREATE_SNAPSHOT OSD commands
  - No overwrite block allocation
- 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)

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
- 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

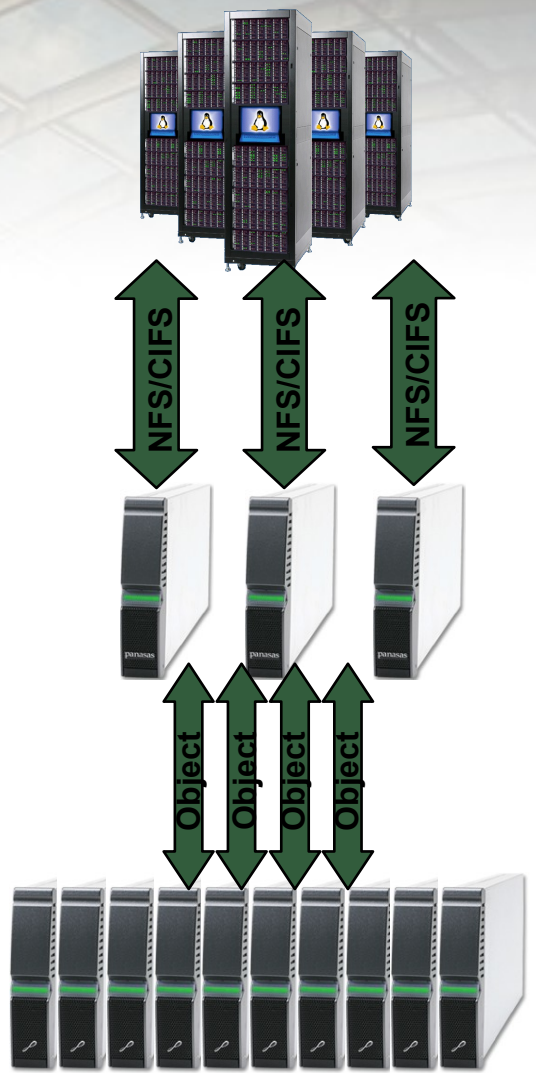
Proposed For OSDv2

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

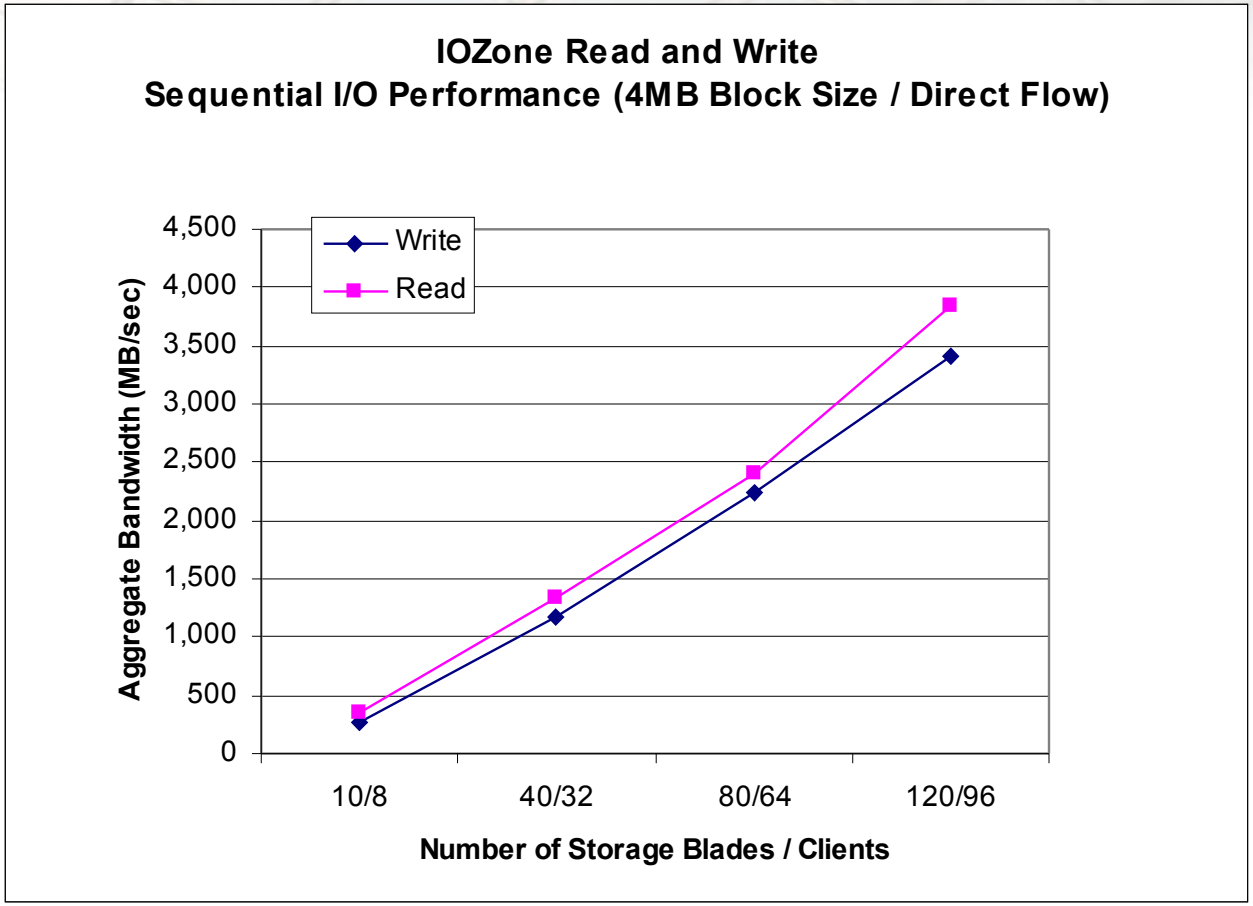
NFS/CIFS Access

- Gateway is a NFS/CIFS server on a Panasas FS client
  - Same Panasas FS client as Linux FS client
  - FreeBSD NFS server, Samba CIFS server
- All clients, including Gateways, have coherent view
  - All gateways provide identical view of PanFS, coherent with Panasas FS
  - Cache coherency provided by Panasas FS client
  - Each gateway accesses all data in system—one NFS mount or CIFS share for whole Panasas cluster
- Legacy systems does not need special software
  - NFS / CIFS are built in modern operating systems
- Managers translate file requests
  - From NFS / CIFS to object and exports data to realm
  - Maintaining permissions and other key attributes
- Scalable performance via additional Managers
- Same volumes available
  - Via Panasas FS, NFS and CIFS simultaneously



Scaling the system

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

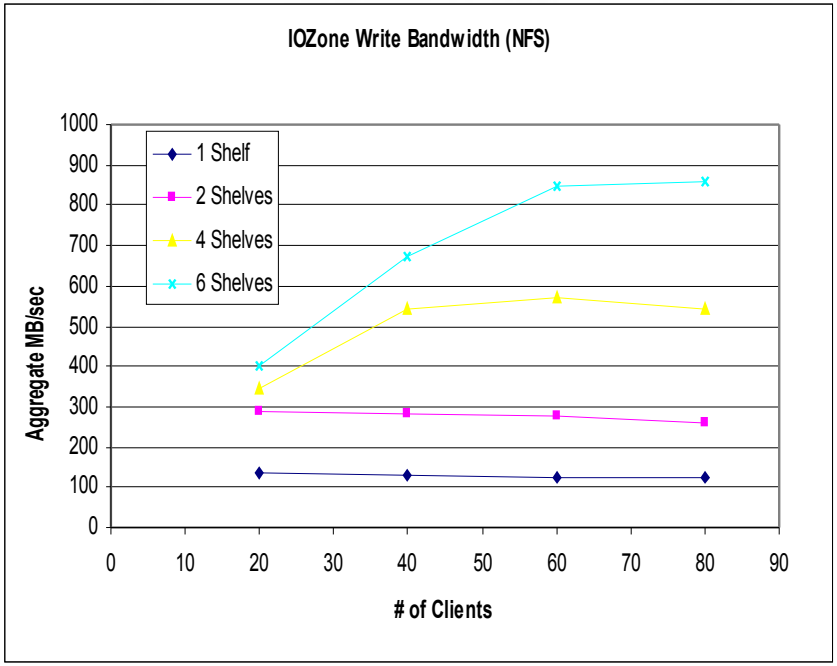


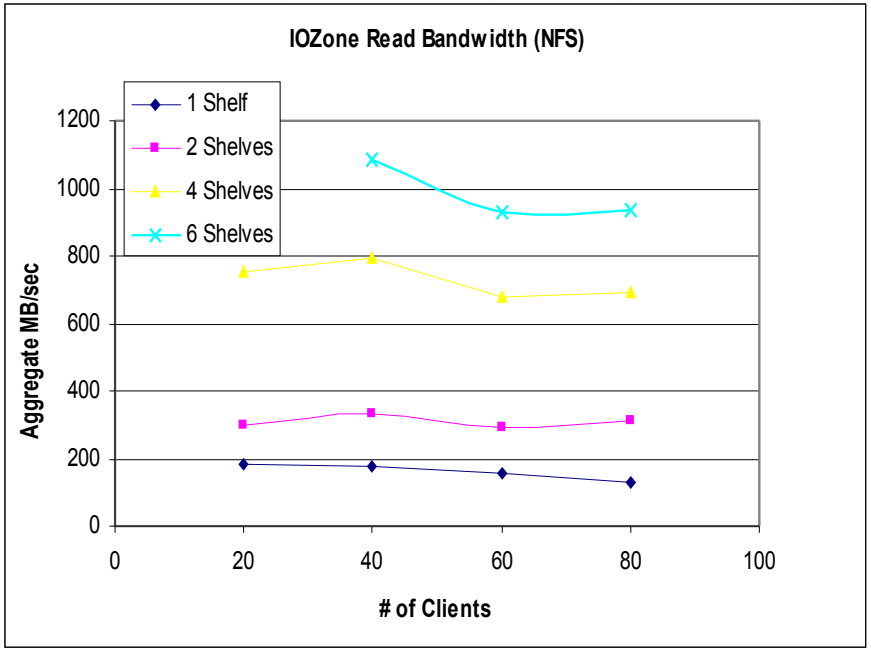
Scaling Clients

- Fixed system size, grow the number of clients (N-to-N over Direct Flow)

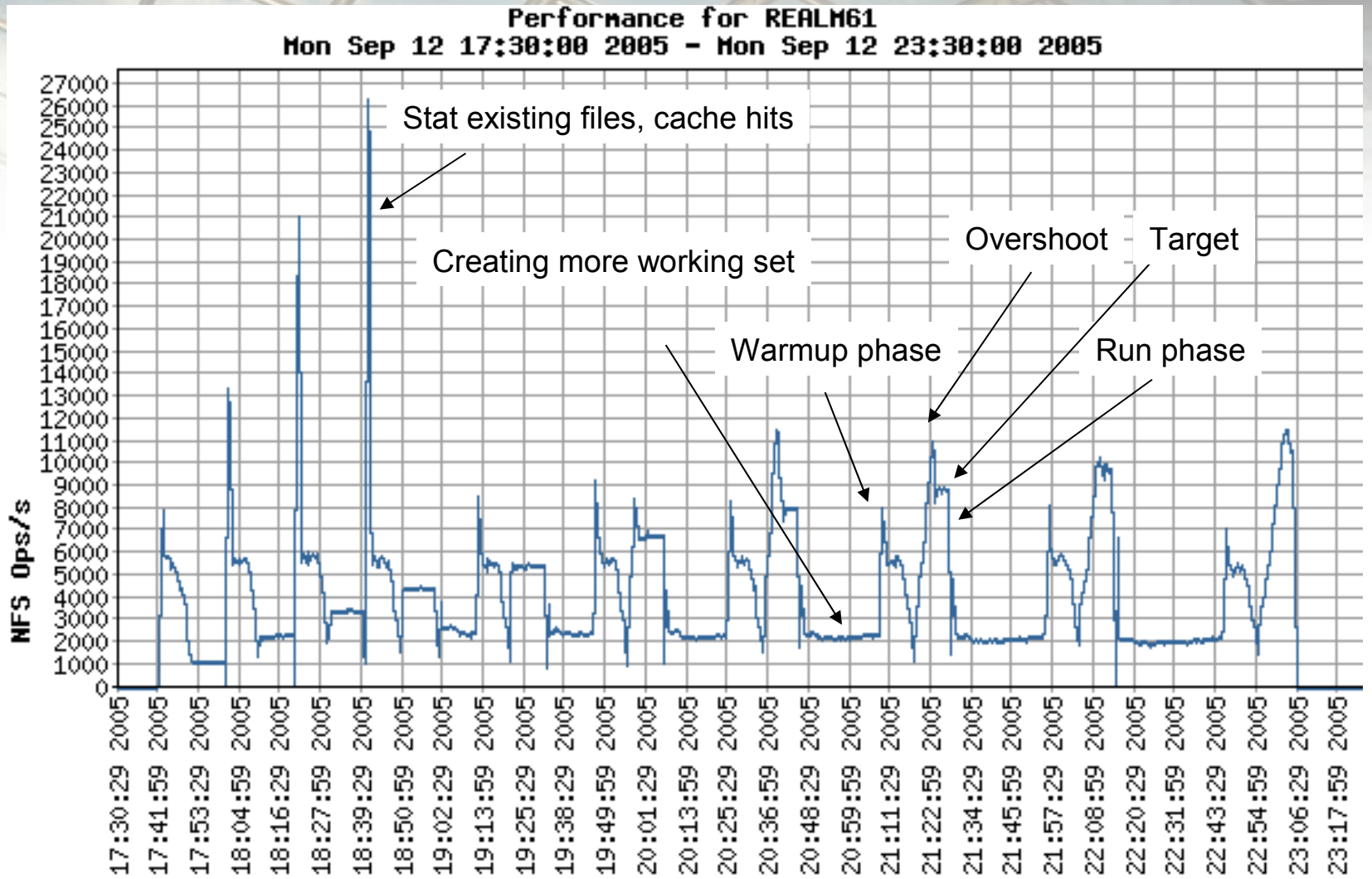
Scaling NFS

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





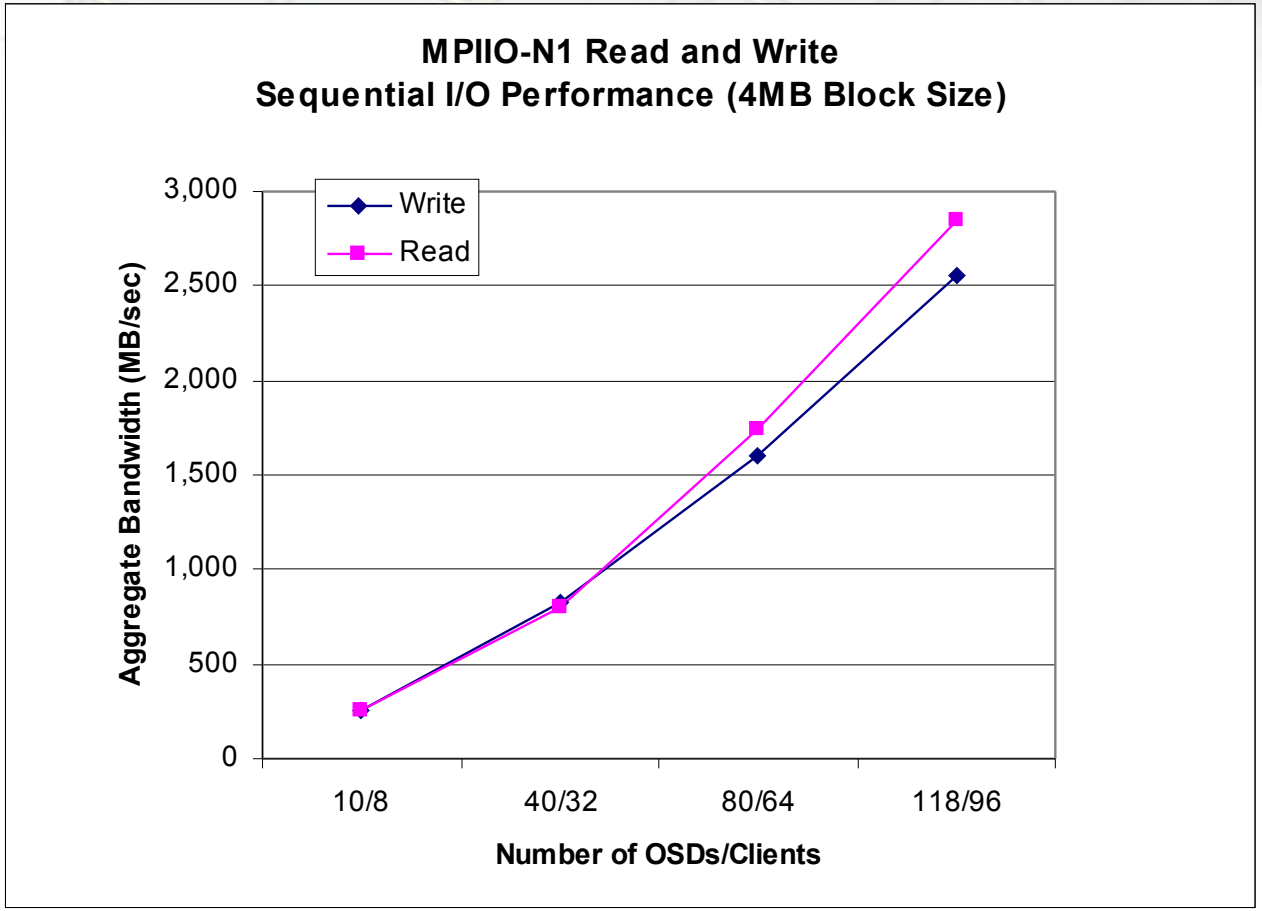
SFS Performance



SFS ORT Run. 10 Runs at successively higher target loads: 1000 to 10,000 ops/sec

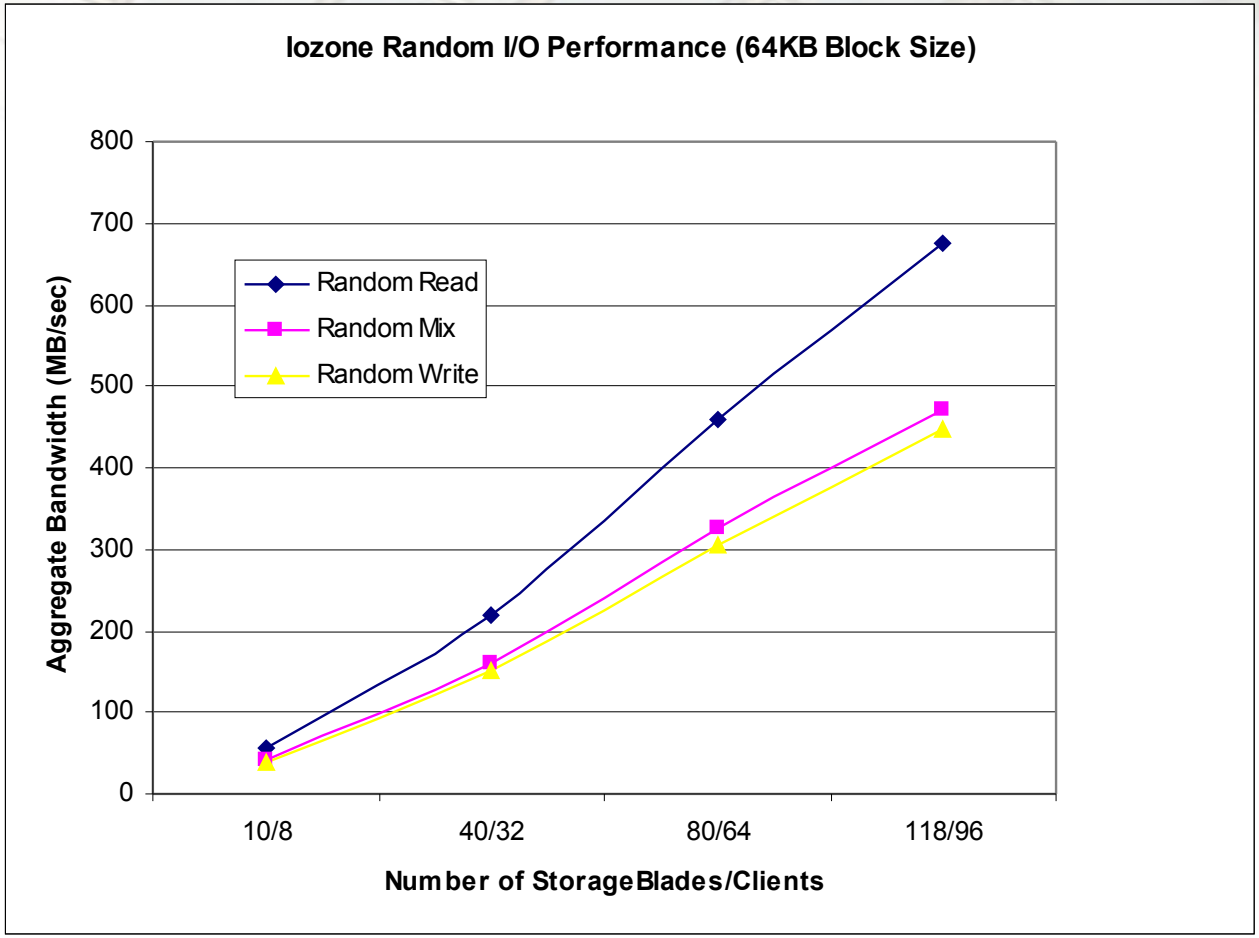
Concurrent Read/Write Performance

- N-to-1 read and write performance using MPI-IO benchmark (over Direct Flow)



Random IO

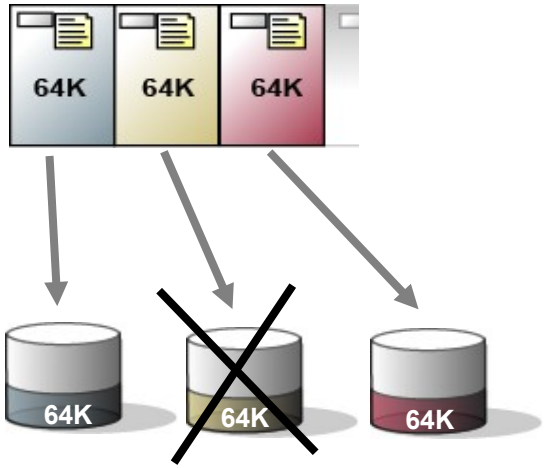
- IOZone over Direct Flow



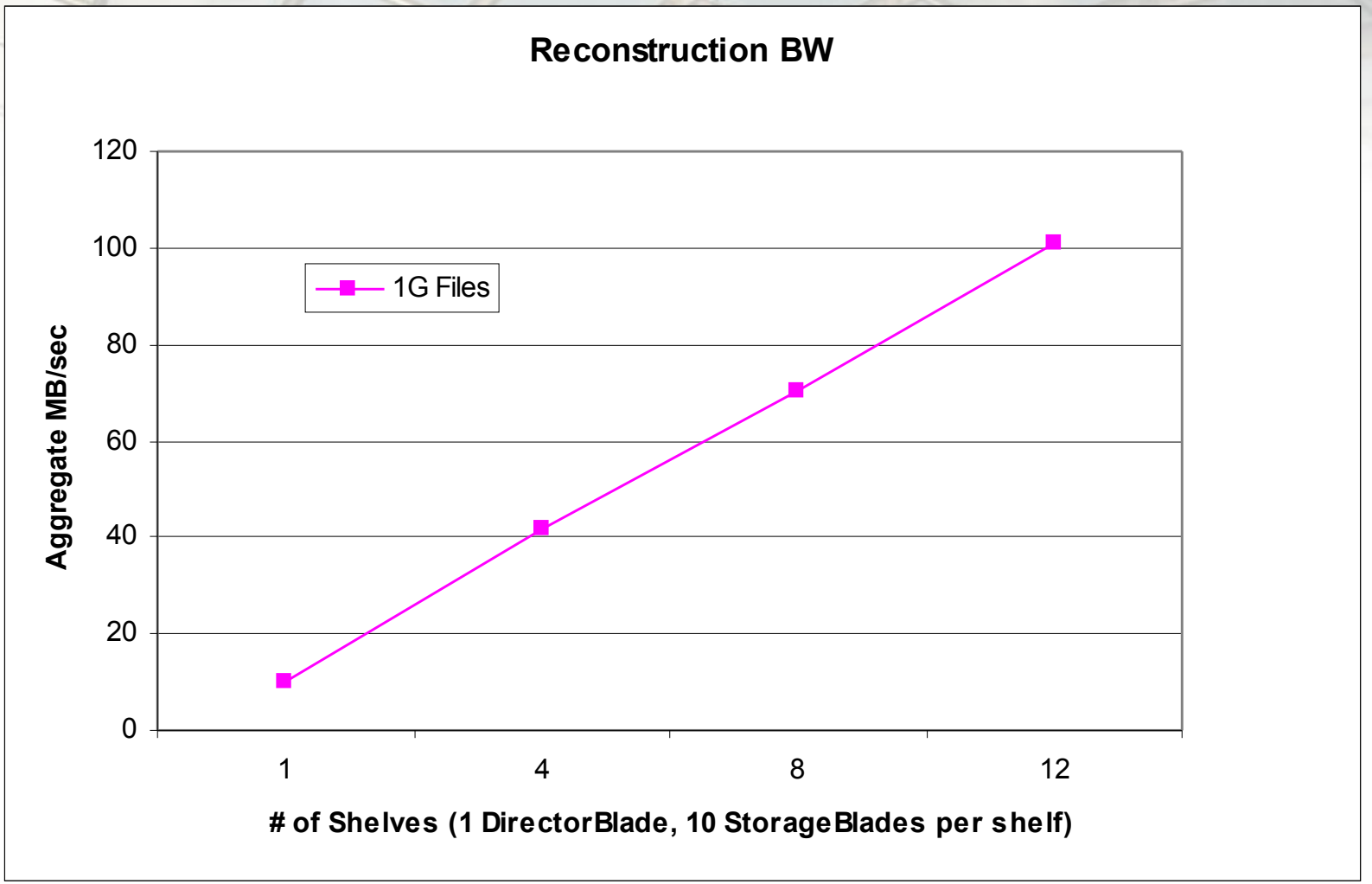
StorageBlade == OSD

Object-based Parallel Reconstruction

- Many systems manage a set of disks with a single RAID controller
  - Bottleneck for general I/O … but can be balanced with cache and limited number of spindles
  - Bottleneck for reconstruction
- By striping over objects and performing RAID at the clients, we eliminate these bottlenecks
  - And we only reconstruct used capacity (vs. the entire disk)
- Per-file Object reconstruction
  - Manager reads data from surviving objects and writes to new object
  - Achieves 10 MB/sec for large files
- Parallel reconstruction
  - Each manager given list of files to reconstruct
  - Each manager works concurrently

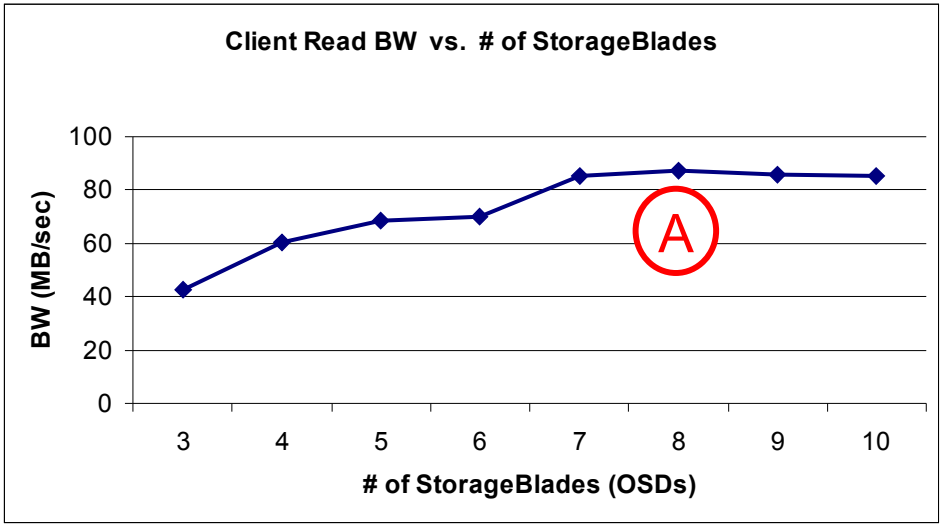


Reconstruction Performance



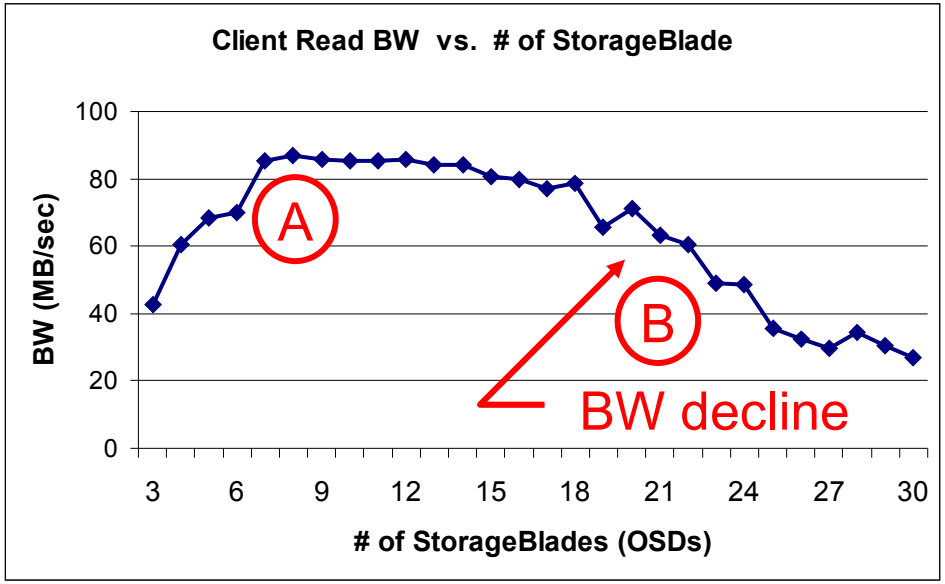
Achieving High Performance Storage

- 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: DirectFlow
    - Scalable NW: Ethernet/TCP
- Plot shows BW increasing as StorageBlades (OSDs) are added
- 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



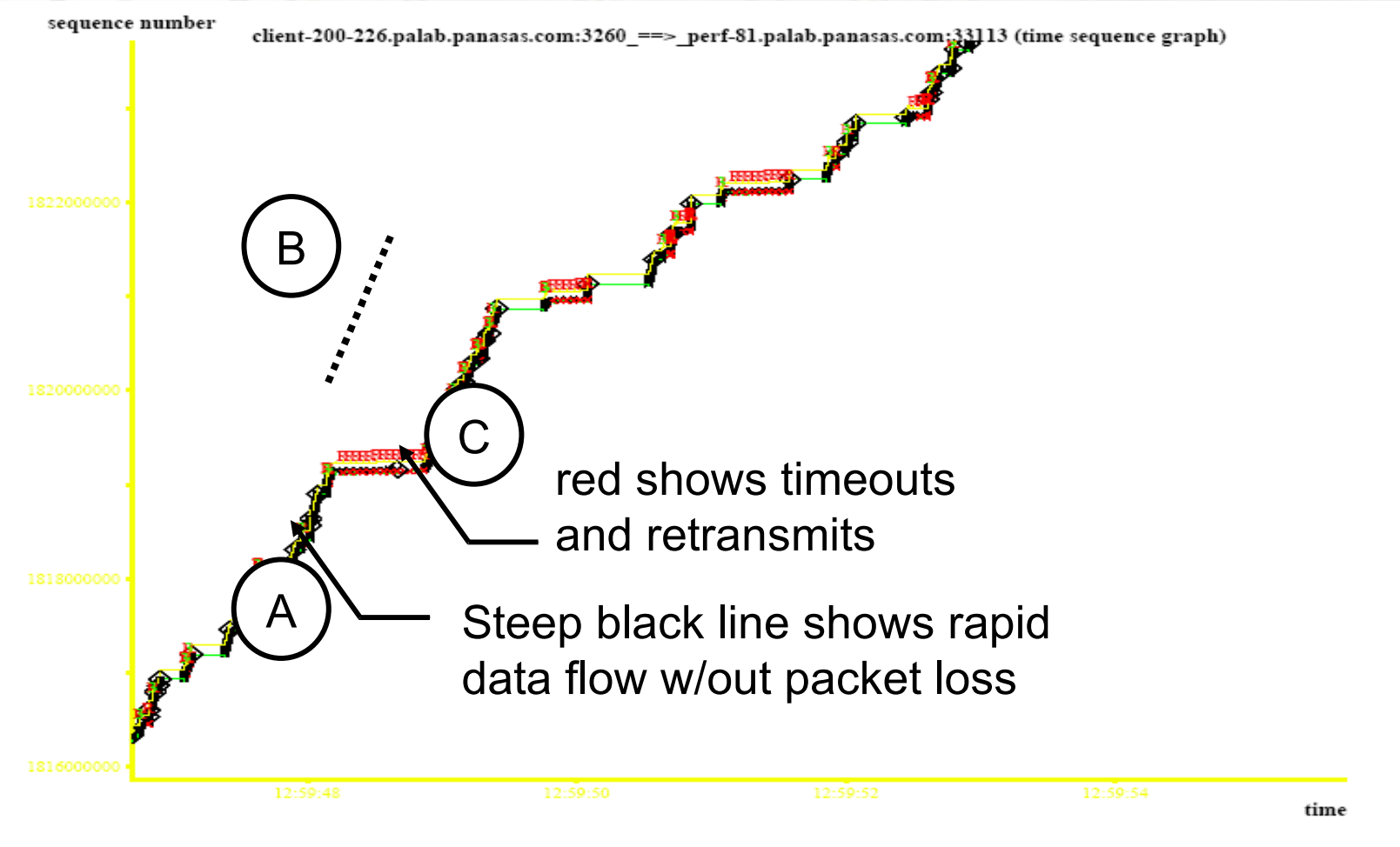
Scalability ?

- Greater than 15 StorageBlades, BW begins to decline
  - Why? Possible culprits include …
    - 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 solved ?



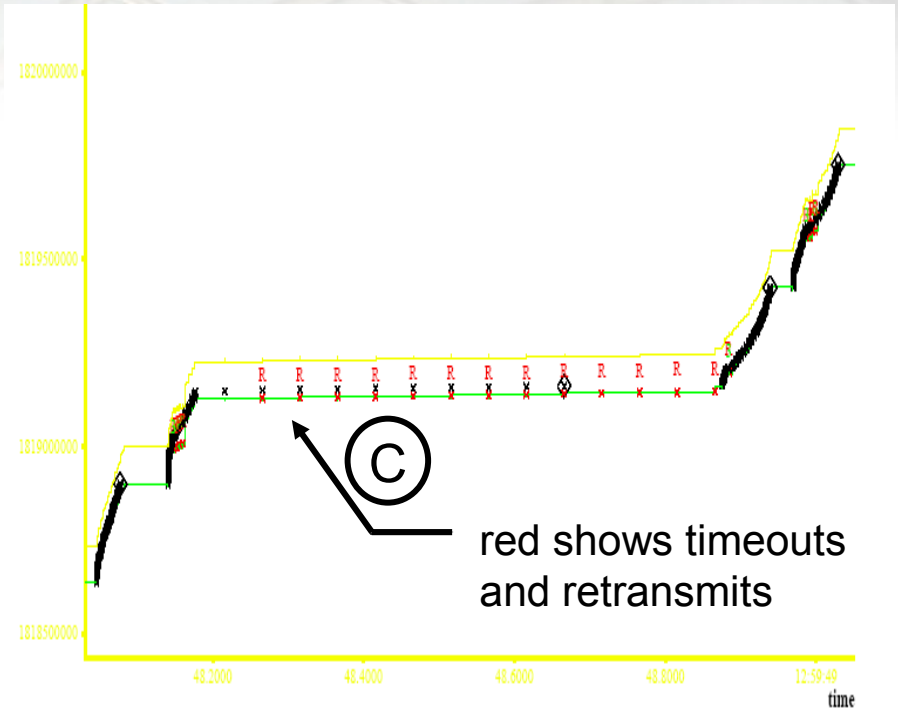
How Does TCP Perceive Incast?

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



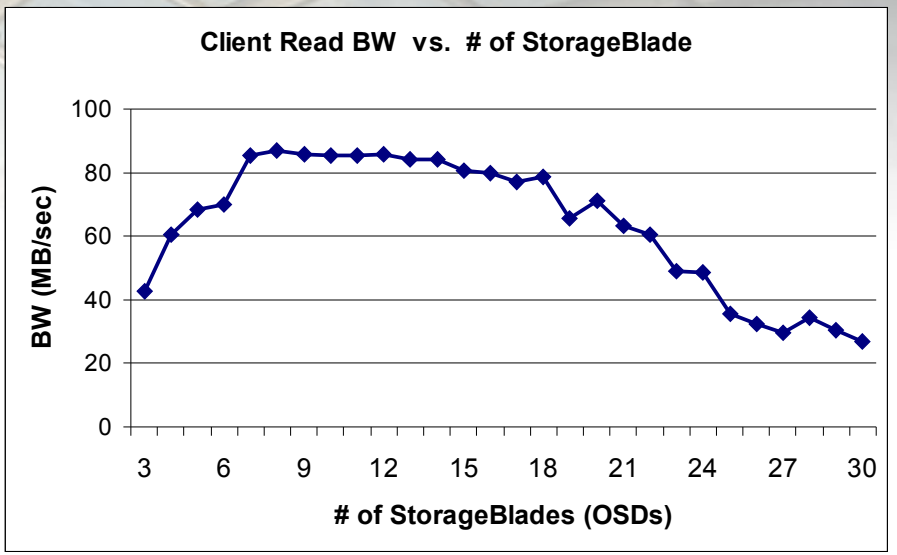
TCP Plot of Retransmissions

- 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 taildrop packet loss
    - SACK ineffective is 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



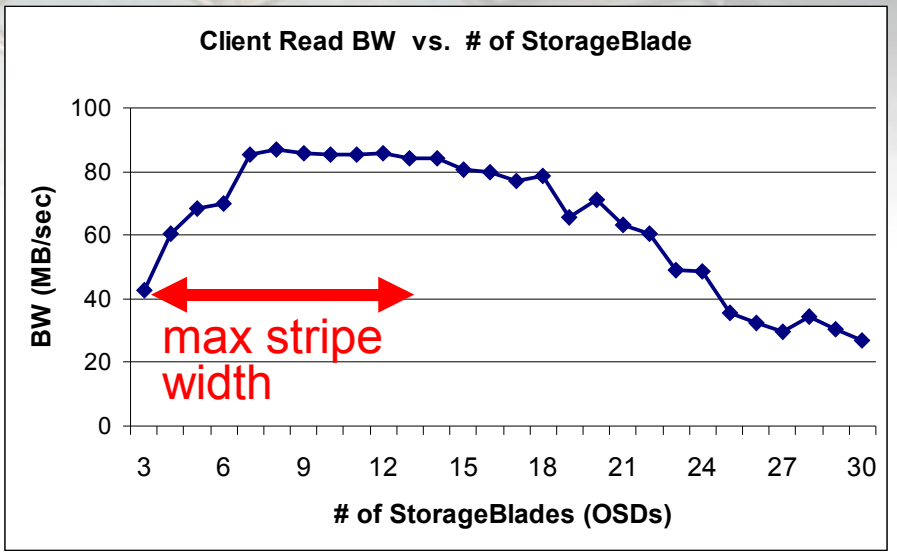
Overcoming Incast

- 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 …



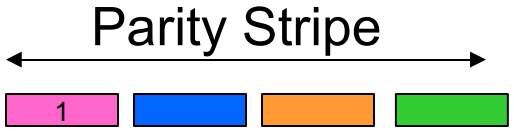
Overcoming Incast (2)

- 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



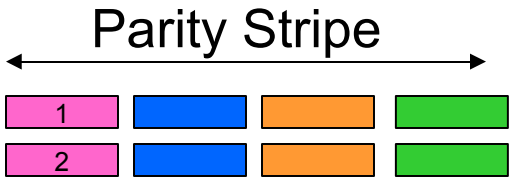
- What we really want is to: 1) limit stripe width to avoid INCAST while 2) striping across all of the storage blades

Option 3: 2-Level RAID Map



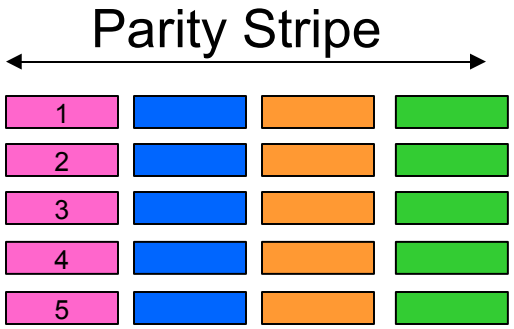
- Begin by striping the file across a limited number of OSDs

2-Level RAID Map



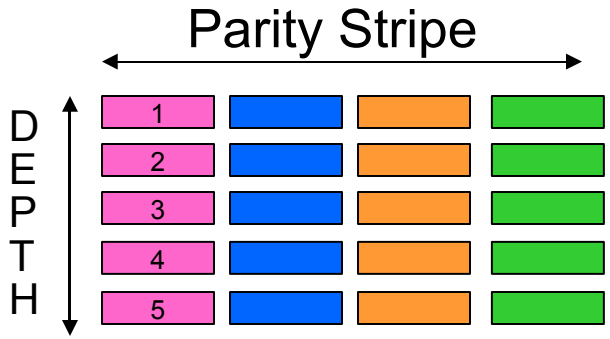
- Begin by striping the file across a set of OSDs
  - As the file grows, write more stripes

2-Level RAID Map



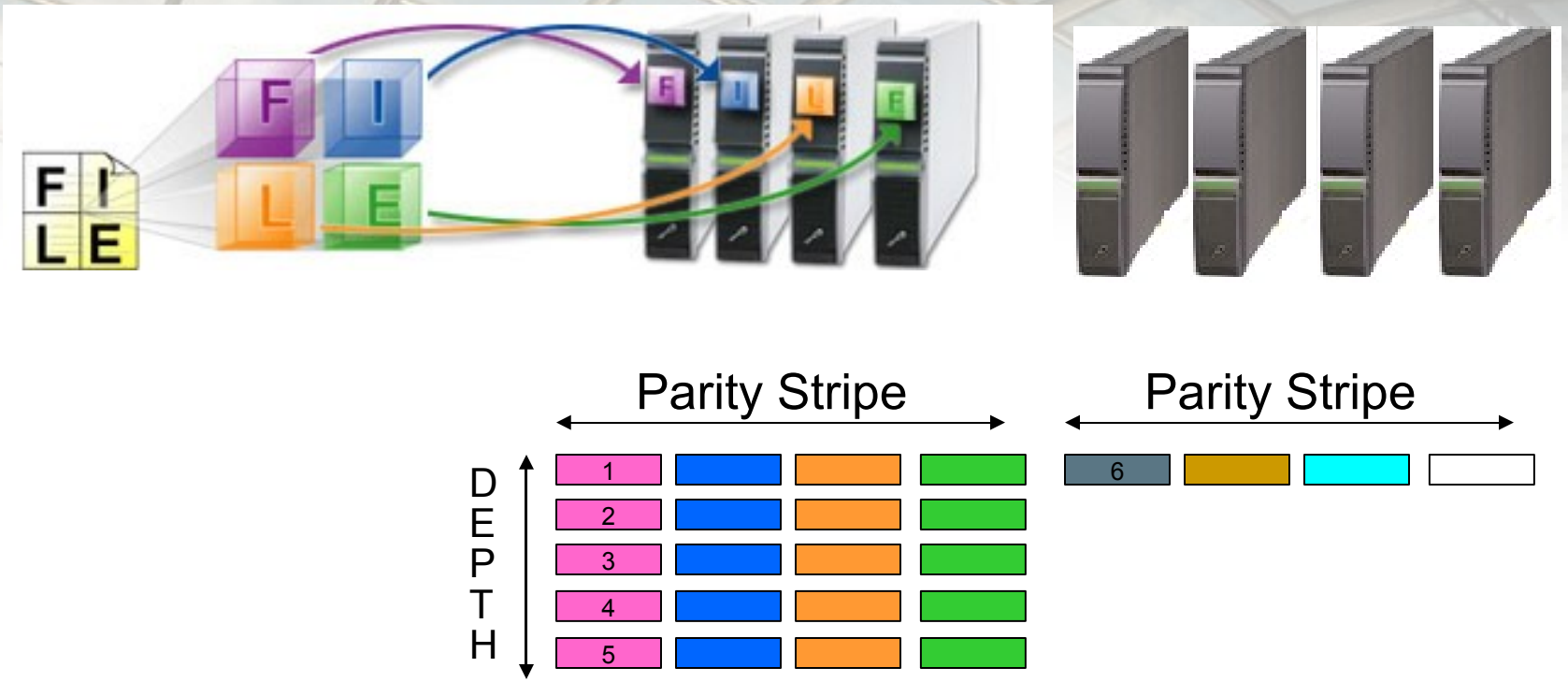
- Begin by striping the file across a set of OSDs
  - As the file grows, write more stripes

2-Level RAID Map



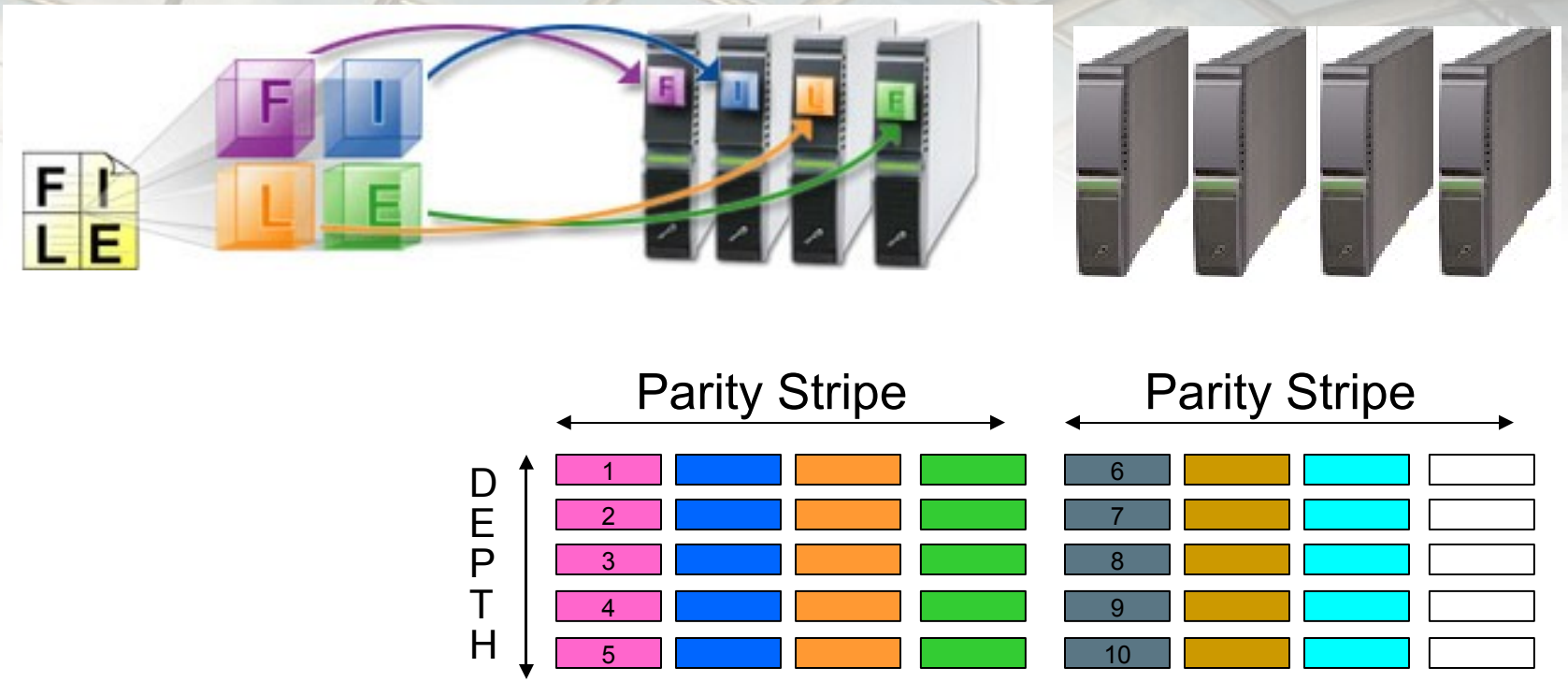
- Begin by striping the file across a set of OSDs
  - As the file grows, write more stripes
  - Once a “sufficient” number of stripes has been written (aka DEPTH),
    - stripe across a new set of OSDs

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

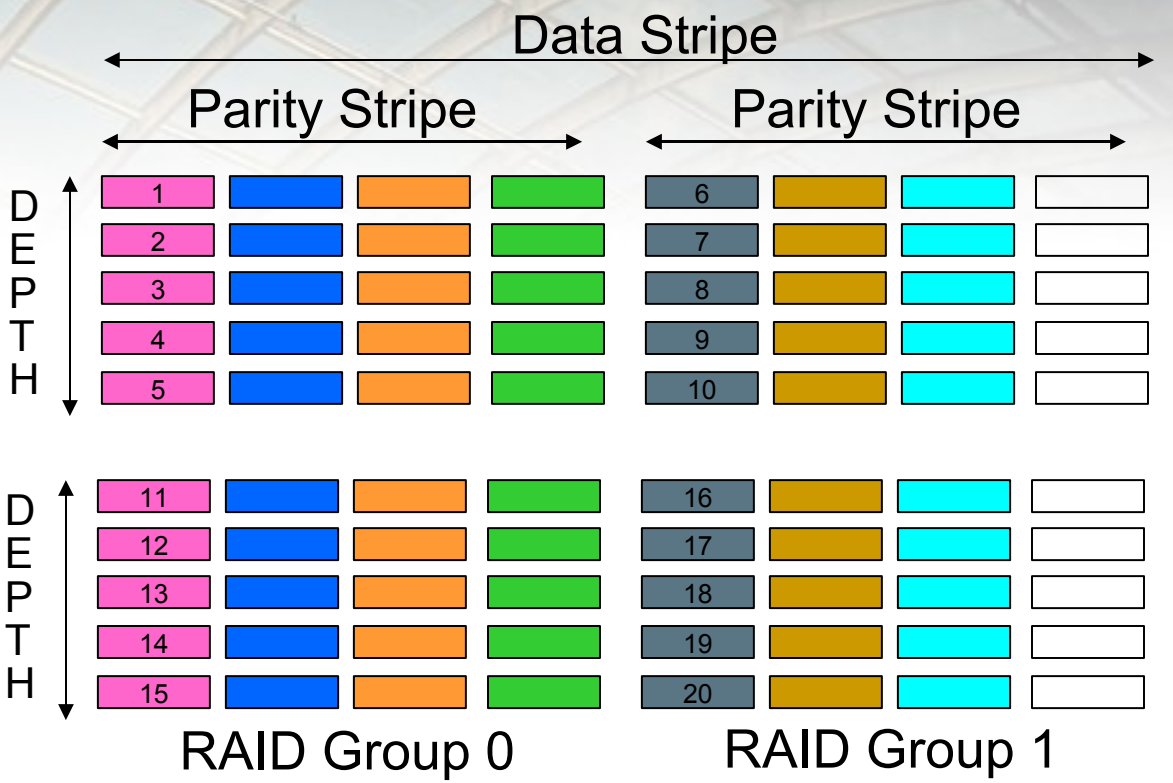
2-Level RAID Map



- 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

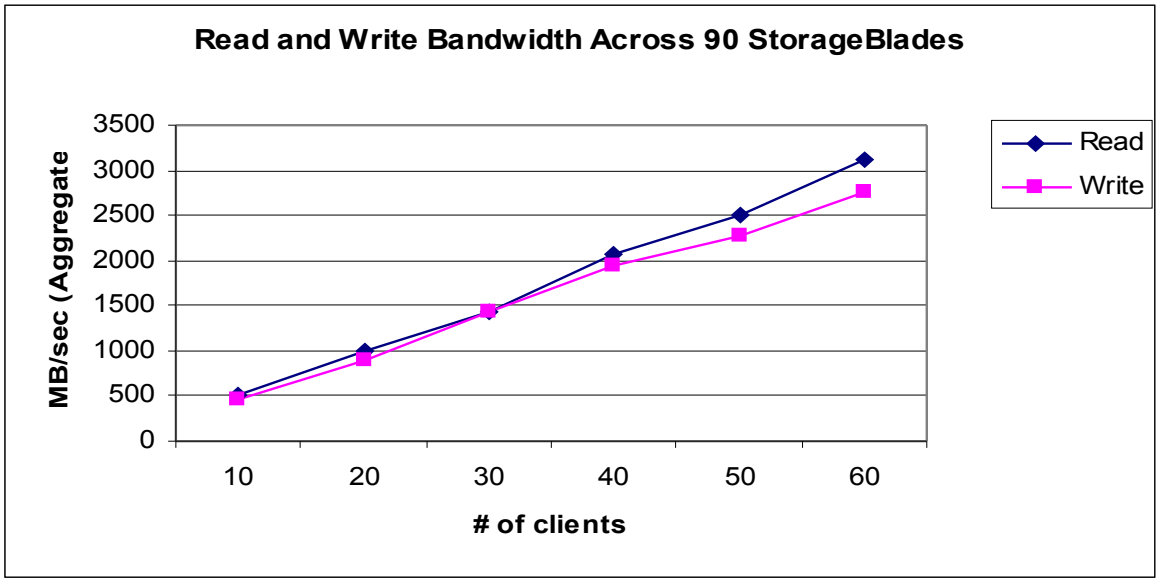
2-Level RAID Map

- 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



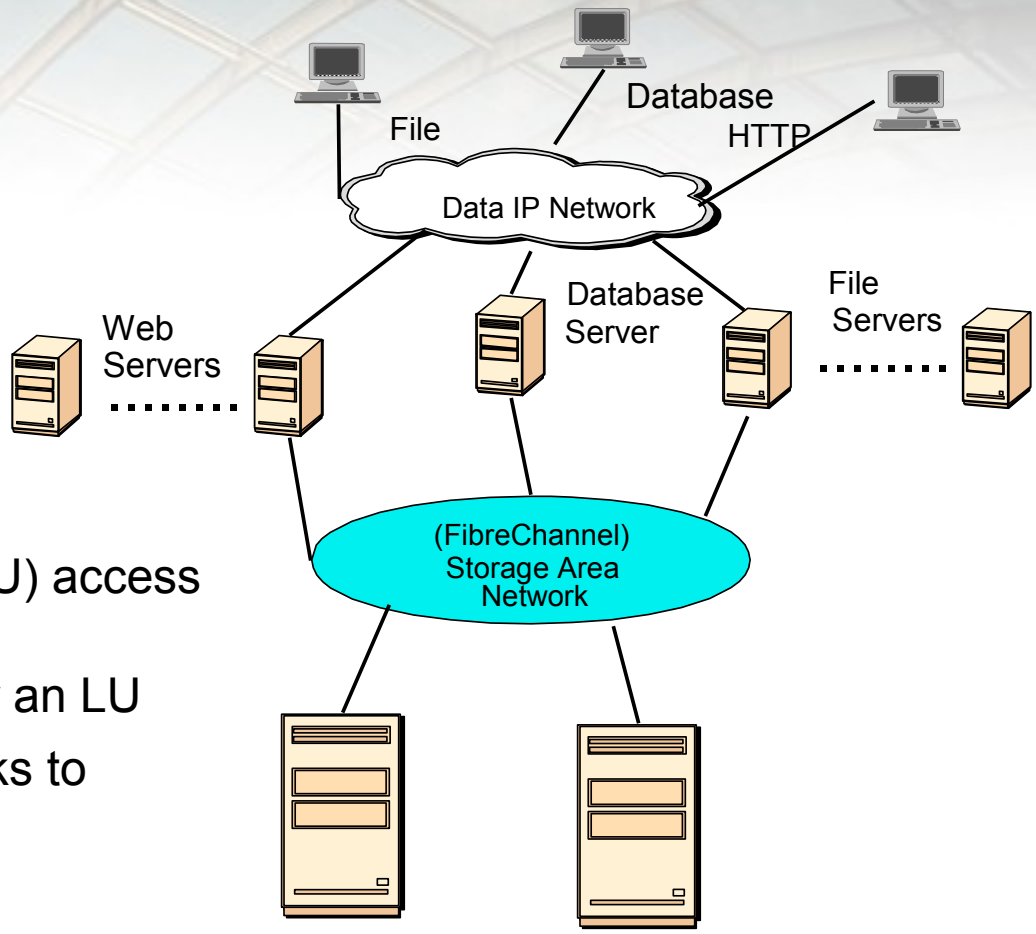
Scalable BW

- N-to-N Reads and Writes
  - Client-based RAID requires extra 10% parity overhead



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

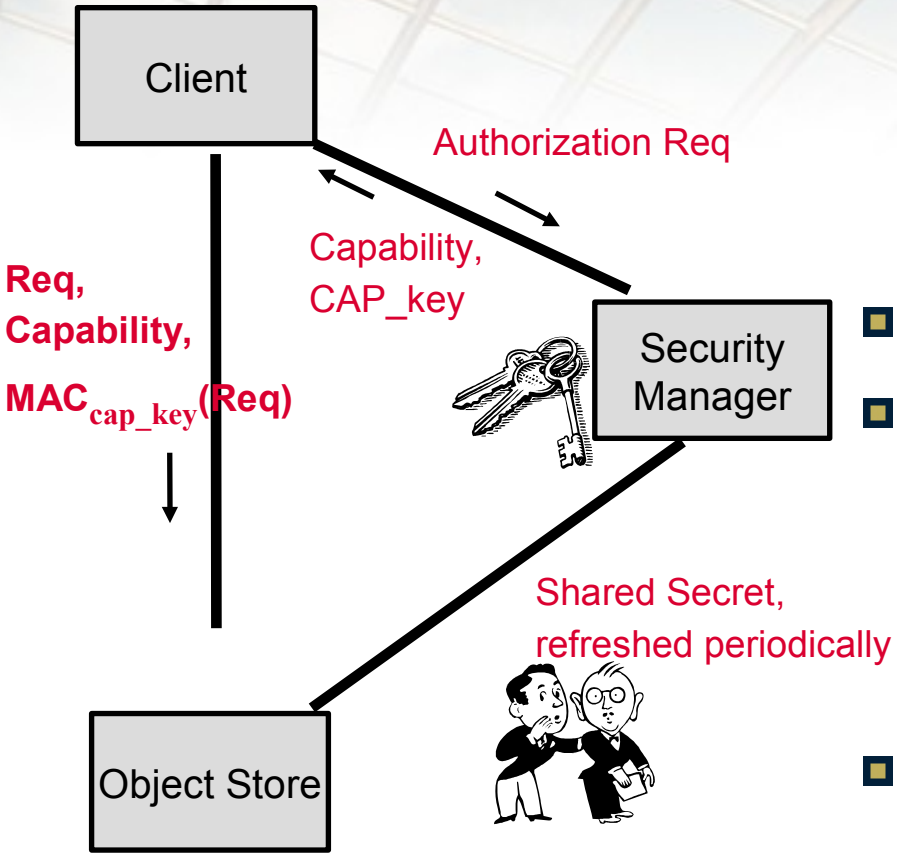
Object Store Security Goals

- 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.

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

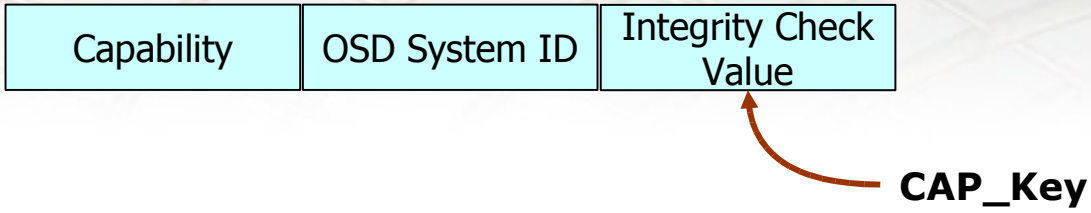
Threat	Threat thwarted by security method				
		CAPKEY			
		Over secure channel [a]			
	NOSEC	No	Yes	CMDRSP	ALLDATA
Forgery of credential	No	Yes	Yes	Yes	Yes
Alteration of capabilities	No	Yes	Yes	Yes	Yes
Use of credential by unauthorized application client	No	Yes [b]	Yes [c]	Yes	Yes
Replay of command 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

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 = MAC $_{\text{secret key}}$ (CAP_Args, 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

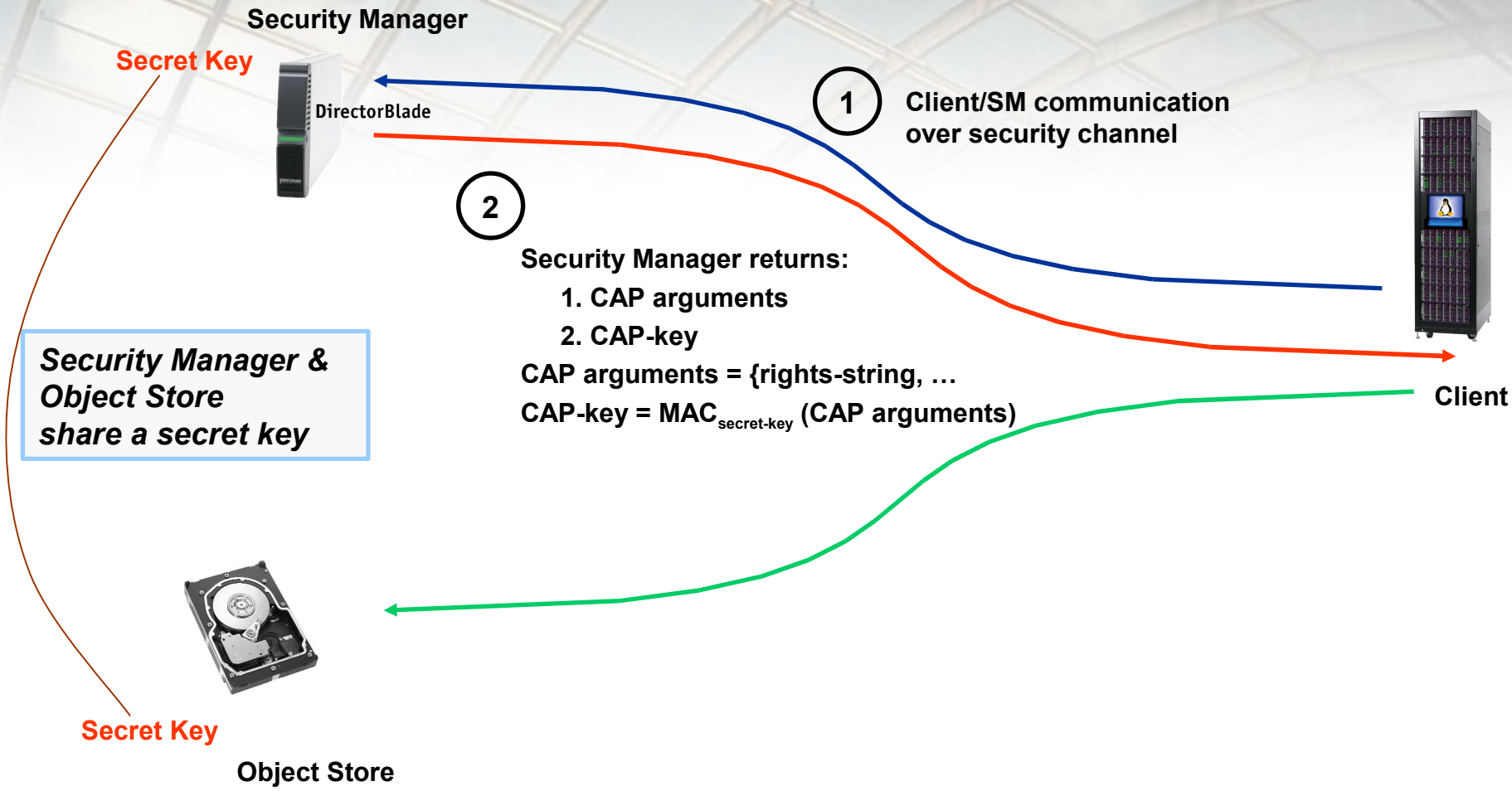
OSD Security Misc

- Capabilities are not bound to a specific client
  - Allow clients to delegate capability
- Capability
- Special commands to set keys for each object
  - Master and working keys
  - Root and partition objects

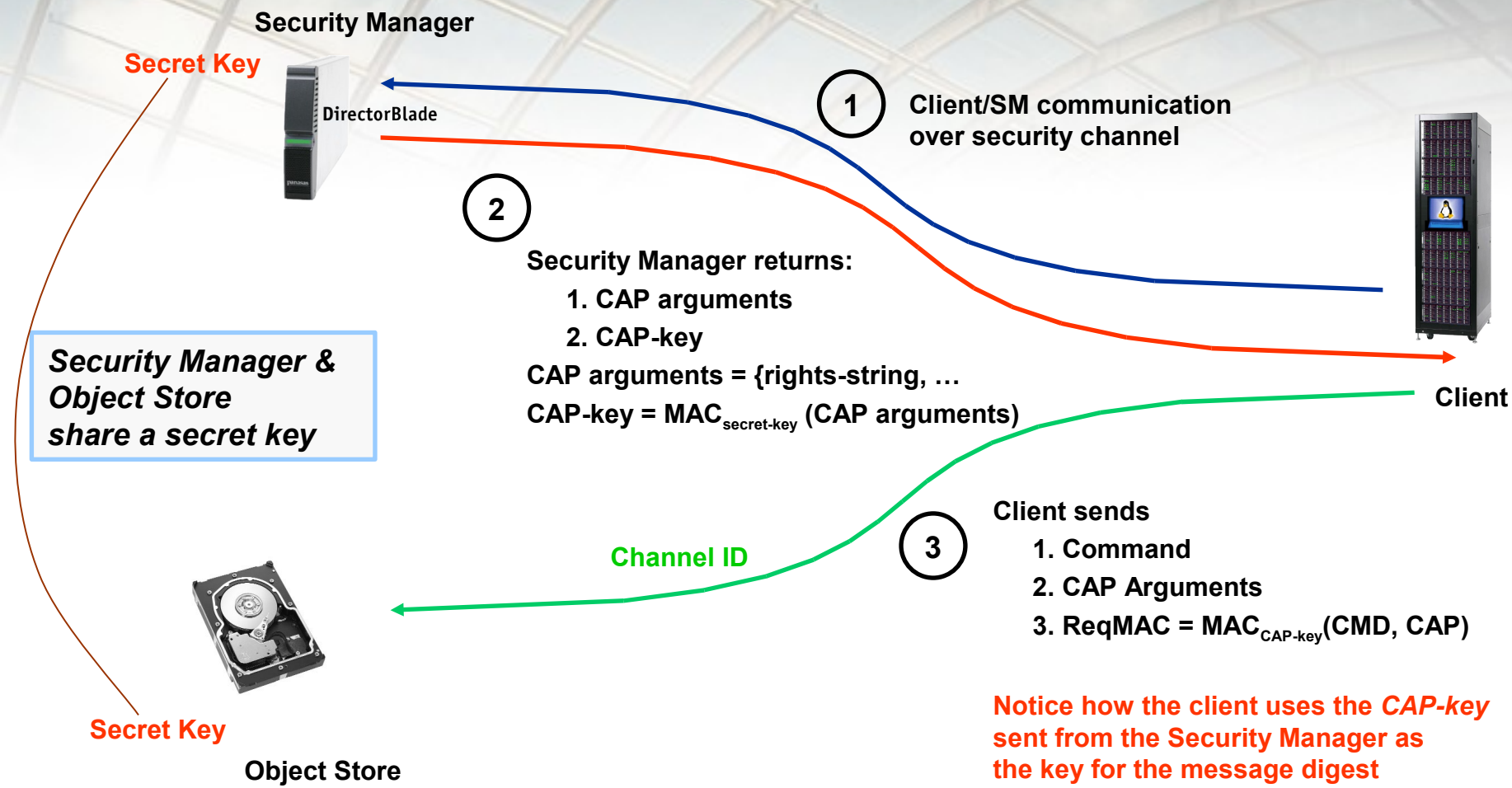
Table 6 — Capability format

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						
9								(LSB)
10		AUDIT						
29								
30	(MSB)	CAPABILITY DISCRIMINATOR						
41								(LSB)
42	(MSB)	OBJECT CREATED TIME						
47								(LSB)
48	OBJECT TYPE							
49		PERMISSIONS BIT MASK						
53								
54	Reserved							
55	OBJECT DESCRIPTOR TYPE				Reserved			
56		OBJECT DESCRIPTOR						
79								

The CAPKEY Mode



The CAPKEY Mode



The CAPKEY Mode

Security Manager

Secret Key

1 Client/SM communication over security channel

2

Security Manager returns:

1. CAP arguments
2. CAP-key

CAP arguments = {rights-string, …

CAP-key = $MAC_{secret\text{-}key}$ (CAP arguments)

Client

Security Manager & Object Store share a secret key

3 Client sends

1. Command
2. CAP Arguments
3. ReqMAC = $MAC_{CAP\text{-}key}$(0, Channel ID)

Channel ID

4 Object Store verifies request AND

1. X = $MAC_{secret\text{-}key}$ (CAP arguments)
2. Y = MAC_x(0, Channel ID)
3. Y =? ReqMAC

Calculated Once Per Credential

Secret Key

Object Store

5 Return (status, data)

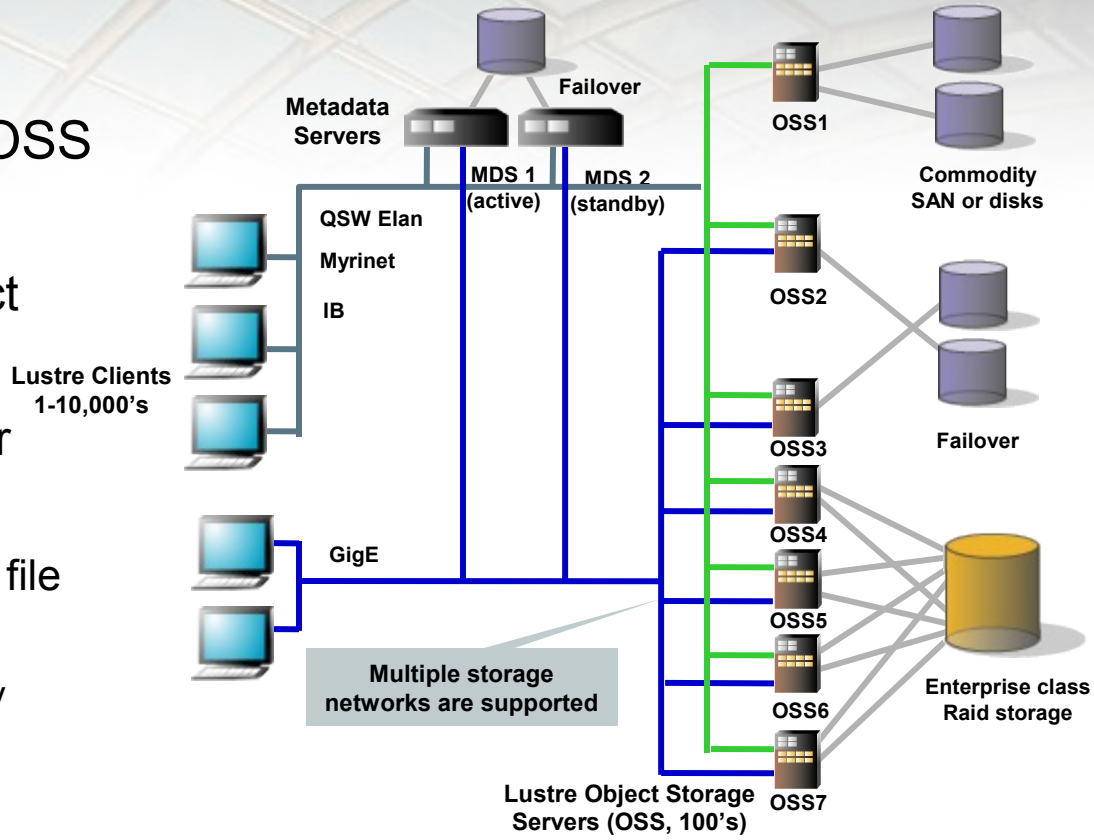
Object-based Storage Systems

Lustre

- Open source object-based storage system
  - Based on NASD architecture from CMU
  - Lots of file system ideas from Coda and InterMezzo
- Key design issues
  - OSD
  - Separation of data from metadata paths
  - Security

Lustre High Level Architecture

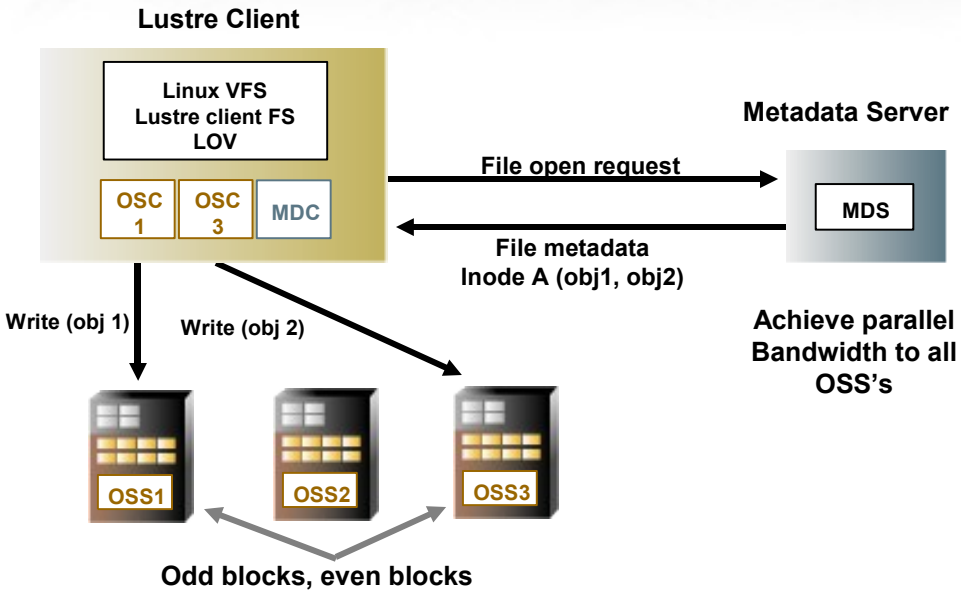
- Direct client-storage communication
- Clients communicate with OSS (Object Storage Servers)
  - Also known as OST (Object Storage Targets)
  - Data is striped RAID 0 over set of OSS's
    - Stripe width determined by file bandwidth requirements
      - Concurrent access typically requires wider stripes
  - 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 Metadata Server

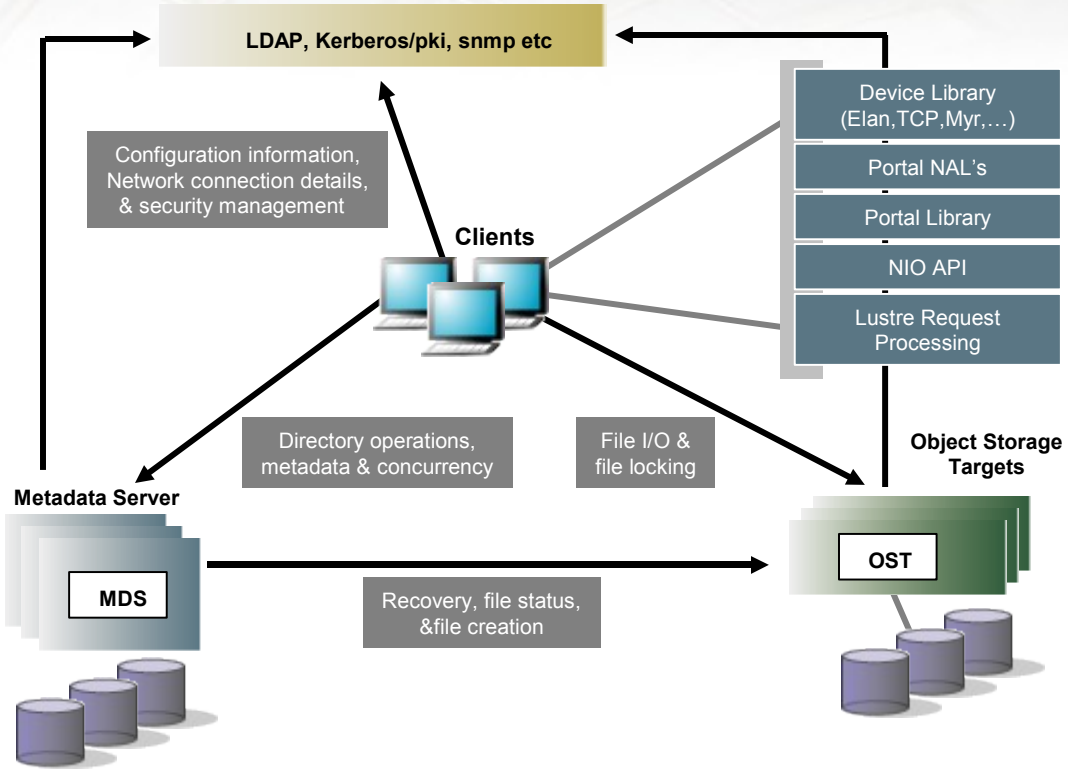
- Clients contact Metadata server to locate data and obtain capability
- File System Metadata
  - Supports lookup, file creation, file and directory attribute manipulation and mapping of file to OSS
  - File system metadata stored on metadata server (vs on OSSs)
  - Metadata server maintains transactional record of file system metadata changes
  - Supports failover in case of failure



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

Lustre Clients

- Client uses LDAP, Kerberos to obtain configuration information
- Uses Lustre RPC over Portals for network-agnostic communication
- OST provides locking



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

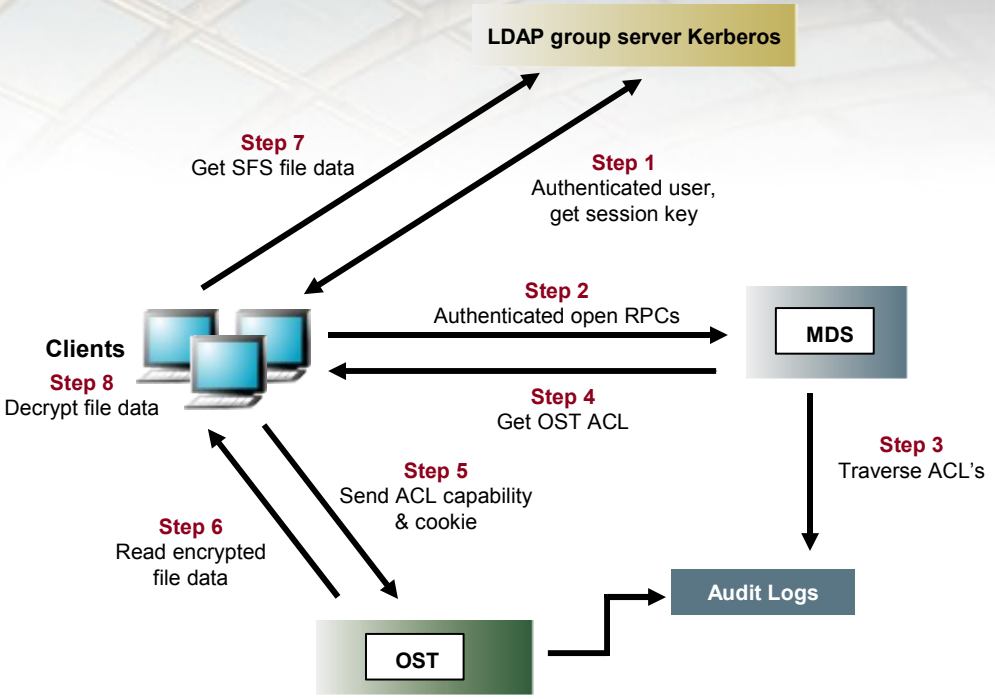
Lustre OSS

- 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 other 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 Security

- Similar to T10 Security
- Capability-based
- Capability on ACL instead of per-object



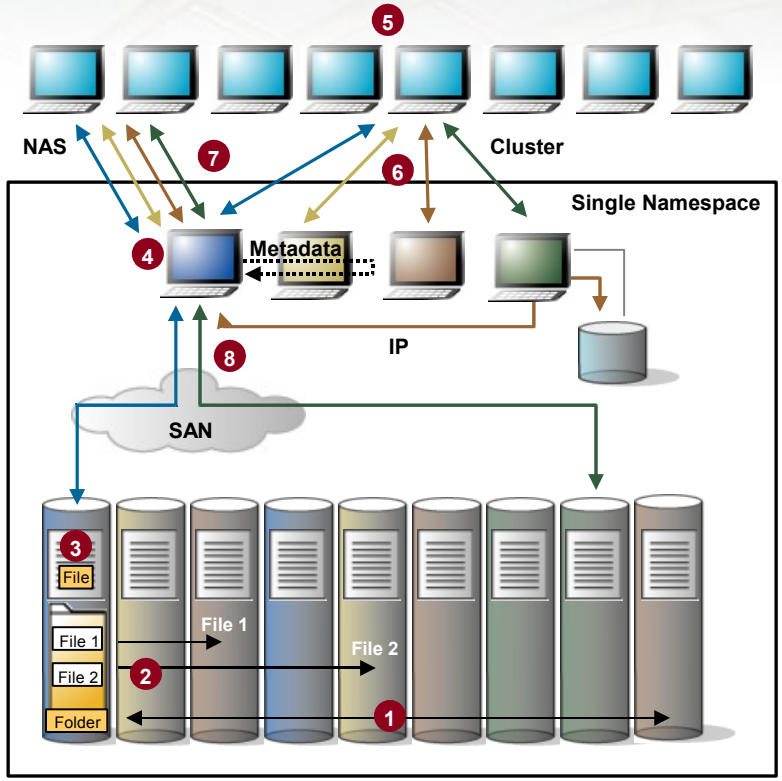
Lustre material from from www.lustre.org and various talks

Lustre Performance

- Testing across 64 OSTs fronting 8 racks of DataDirect S2A8500 storage (estimate about 500 drives)
- Peak at about 4.5GB/sec (network max is 6.4GB/sec ethernet)
  - Write file-per-process (FPP)
  - Read FPP is ~2.5 GB/sec
- Other performance measurements up to 11 GB/sec over specialized networks

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
- EMC CX7000 storage system
  - Eighty 10K RPM FC drives
- Read BW scales from 1 to 8 servers
- Write BW scales from 1 to 4 servers

EMC Centera Highlights

- Software enables Content-addressed Storage
  - Similar to OSD … present storage system with data, get back a single "handle" for future references
  - Different from SCSI or NFS/CIFS interface, requires application support
  - Designed for fixed content (write-once)
    - Any "object" presented to system is unchangable and authenticated (C-Clip technology)
- Physical Design (2 years old)
  - Storage: 4.8TB – 9.6TB RAID 1/RAIN (19.2TB Raw storage) per cabinet (up to 16 cabinets)
    - Up to 16 cabinets per cluster (150 TB) … and up to 7 clusters (1.2 PB)
  - ATA-based

What's this Content Stuff

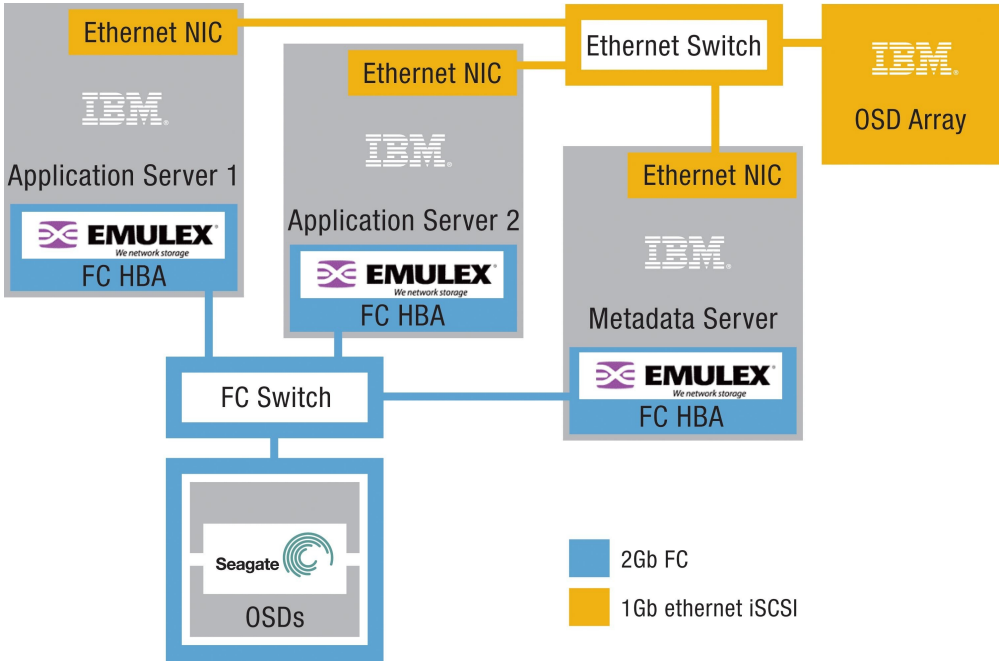
- Entire BLOB (Binary Large Object) is presented to the Centera system
- Application delivers data object to Centera API which generates claim check
  - Claim check is
    - RSA MD-5 generate a 256-bit signature
    - Inserted into XLM file called C-Clip Descriptor File (CDF)
  - Data blob and CDF are 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
  - 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

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

Out-of-Band Interoperability Issues

ADVANTAGES

- Capacity scales
- Bandwidth scales

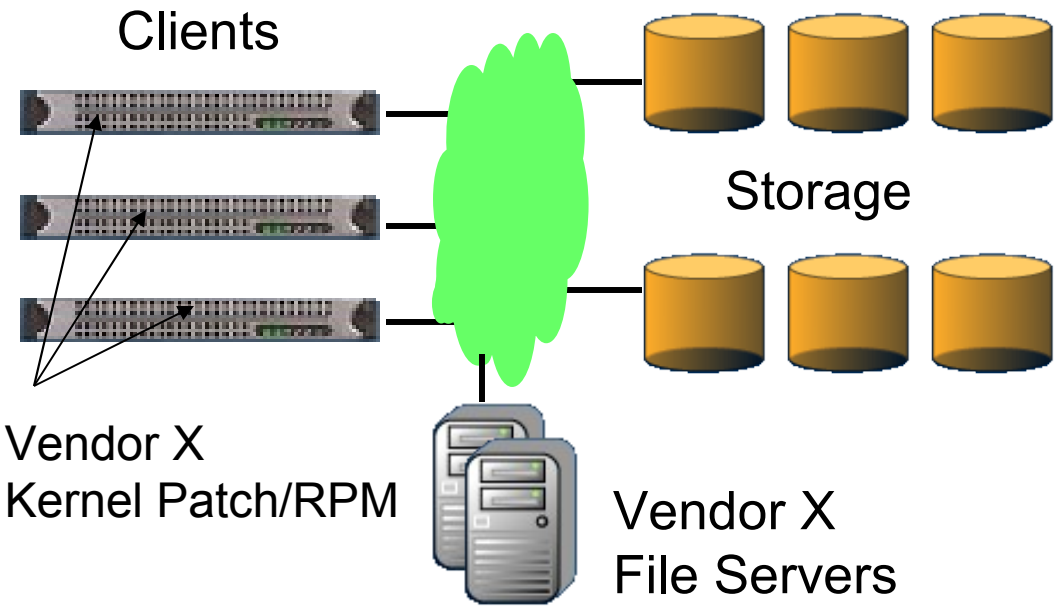
DISADVANTAGES

- Requires client kernel addition
- Many non-interoperable solutions
- (But, pNFS may change this)

EXAMPLE FEATURES

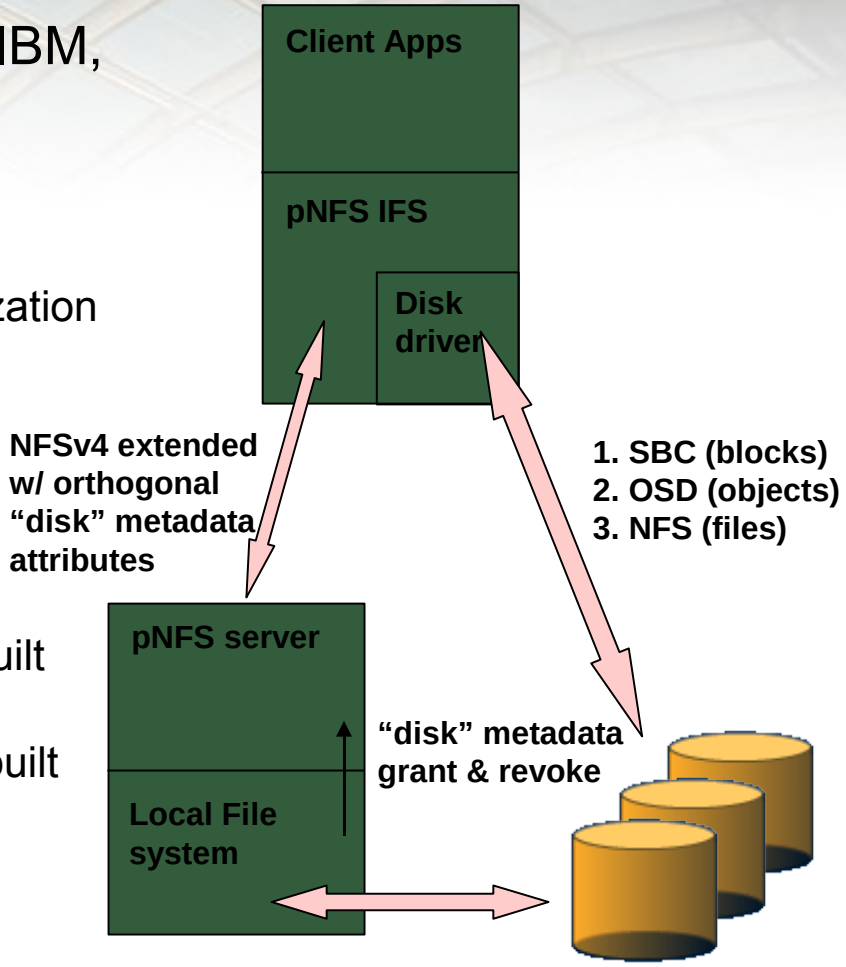
POSIX Plus & Minus

- Global mount point
- Fault tolerant cache coherence
- RAID 0, 1, 5 & snapshots
- Distributed metadata and online growth, upgrade



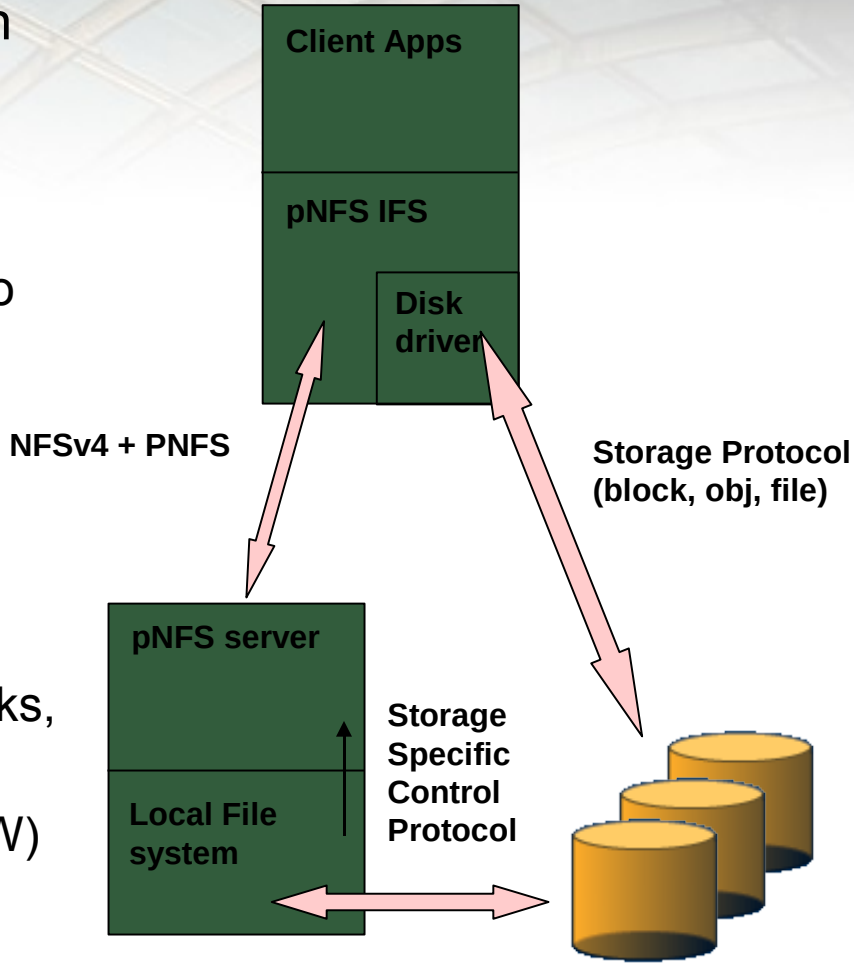
File Systems Standards: Parallel NFS

- IETF NFSv4 initiative
  - U. Michigan, NetApp, Sun, EMC, IBM, Panasas, ….
  - Enable parallel transfer in NFS
- Extension to NFSv4 for parallel I/O
  - Parallel NFS requests routing / file virtualization
- Provides asymmetric architecture (metadata and data servers) for NFS
- Framework supports
  - Blocks: SBC/FCP/FC or SBC/iSCSI for flies built on blocks
  - Objects: OSD/iSCSI/TCP/IP/GE for files built on objects
  - Files: NFS/ONCRPC/TCP/IP/GE for files built on subfiles
- Inode-level encapsulation in server & client code



pNFS Architecture

- NFSv4+pNFS enhances pNFS to enable direct client/storage communication on Read, Write and Flush
  - Client sends all metadata operations to server
  - Client decides on-the-fly to send data to server or storage
- Enabled by layout
  - Client requests layout information from pNFS server
- Layout includes:
  - TYPE - Storage protocol specific (blocks, files, objects)
  - IOMODE – permission type (READ, RW)
  - LAYOUT – list of devices and storage "location" identifiers



Proposed pNFS Layouts

- Block Storage
  - list of extents defined as
    struct {
        file_offset
        file_length
        VolumeID
        storage_offset
        storage_state {RW, READ, INVALID_DATA, NONE_DATA}
    }

Proposed pNFS Layouts (con't)

- Object Storage
  - 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
  }

Distributing and Managing Layouts

- Client requests layout with LAYOUTGET
  - Client specifies {file, offset+length, IOTYPE, layout-type}
  - Server may return one or more layouts
    - Layout describing a file could be vary large TB file stored in blocks)
- LAYOUTCOMMIT returns layout to client
  - For writes, LAYOUTCOMMIT informs server:
    - Which data segments were written (using a map)
    - Logical length of file
    - File's new capacity
- Server will also update file's attributes, including:
  - Change attribute – file serial number to detect/denote changes
  - Modify/access time
- LAYOUTCOMMIT informs server of changes while allowing the client to continue operating on the file

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

Consistency

- Consistency guarantees
  - Block and file storage cannot guarantee consistency of layout (objects have object-version number and capability)
  - Therefore, when become “stale” due to map changes (e.g., restriping of data) or file permissions change (e.g., RW layout type for a file that is chmod’d READONLY)
    - pNFS requires that server immediately recall stale layouts using LAYOUTRECALL
- Server recalls layout(s) from client(s) using CB_LAYOUTRECALL
  - Allows servers to maintain map consistency between clients (and server)
    - CB_LAYOUTRECALL specifies {fileID, offset, length, IOMODE} of layout recalled
  - Client replies immediately to CB_LAYOUTRECALL and will then begin to (async) return the maps requested
  - Client then issues LAYOUTRETURN to return each layout covered by CB_LAYOUTRECALL
    - LAYOUTRETURN removes client’s access to the map

pNFS Protocol Issues

- Race conditions
  - Recall traffic could create deadlocks (e.g., server recalling layout while client requests layout)
  - Handled by the server and client enforcing ordering
    - Server must reject new layout request to layout that is currently being recalled
    - Client must wait for server replied to any conflicting LAYOUTGET requests before returning layouts
- Leases
  - Client may only use layout who's lease has not expired
    - Lease on layout ensures that layout can be reclaimed by system

Crash Recovery

- Client recovery
  - Recovery similar to NFSv4 lock/delegation state
  - When client reboots, it throws away all layout information
  - Server can then reclaim layouts either with implicit expired lease or by explicit LAYOUTRECALL
    - Server determines client reboot when client contacts server on reboot (via SETCLIENTID protocol)
- Metadata Server Recovery
  - Client discovers server reboot with STALE_STATEID or STALE_CLIENTID reply
    - Informs client that current maps are invalid
  - Client will flush, if necessary, any dirty data
    - Flush can go directly to storage or through server
    - Requires new map request to go directly to storage

Crash Recovery (con't)

- Storage recovery
  - When storage crashes, client needs to discover this to recover
    - Failed read/write command
  - Client could write data via server if detects storage crash
  - Client could wait for device to be remapped (i.e., a new map)

Device Identification

- GETDEVICEINFO op conveys device indentification and addressing information from NFS server to pNFS client
- SCSI device identification
  - SCSI has a tradition of verifying volume labels before scribbling on disks
  - OSD Name attribute on root object serves as a label
- File server “device” identification
  - May be as simple as an IP address

pNFS

- Active in the NFSv4 working group
  - Panasas, Sun, NetApp, EMC, IBM, others …
- **IETF pNFS Documents:**
  - draft-gibson-pnfs-reqs-01.txt
  - draft-welch-pnfs-ops-03.txt (replaced by draft-ietf-nfsv4-pnfs doc)
  - draft-zelenka-pnfs-obj-01.txt (soon draft-ietf-nfsv4-pnfs-obj-00.txt)
  - draft-black-pnfs-block-01.txt (soon draft-ietf-nfsv4-pnfs-block-00.txt)
  - draft-ietf-nfsv4-pnfs-00.txt

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/labs/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-basedStorageDevi