Big Data Processing Technologies

Chentao Wu
Associate Professor
Dept. of Computer Science and Engineering
wuct@cs.sjtu.edu.cn

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Schedule

- lec1: Introduction on big data and cloud computing
- lec2: Introduction on data storage
- lec3: Data reliability (Replication/Archive/EC)
- lec4: Data consistency problem
- lec5: Block storage and file storage
- lec6: Object-based storage
- lec7: Distributed file system
- lec8: Metadata management

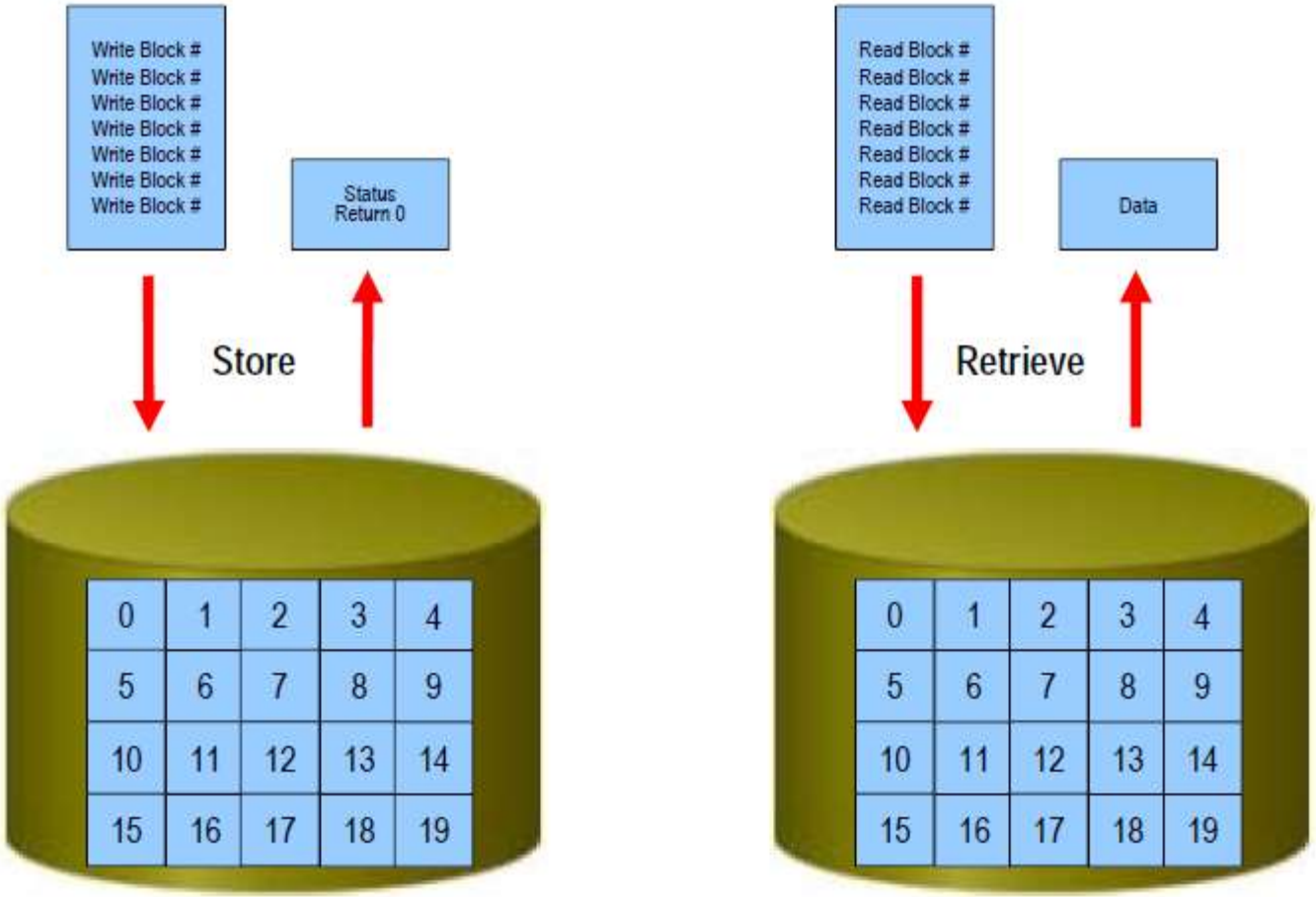
Collaborators

Google 谷歌

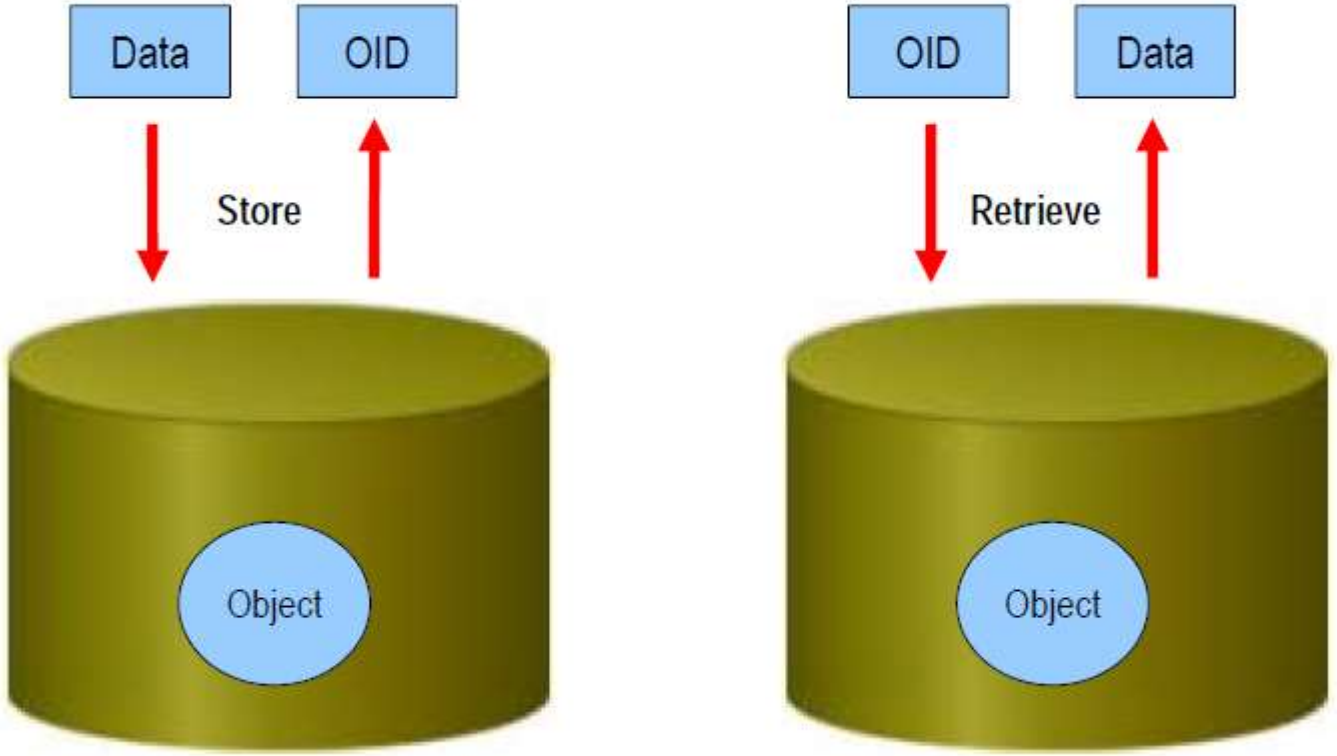
阿里云 aliyun.com

DELL EMC

The Block Paradigm

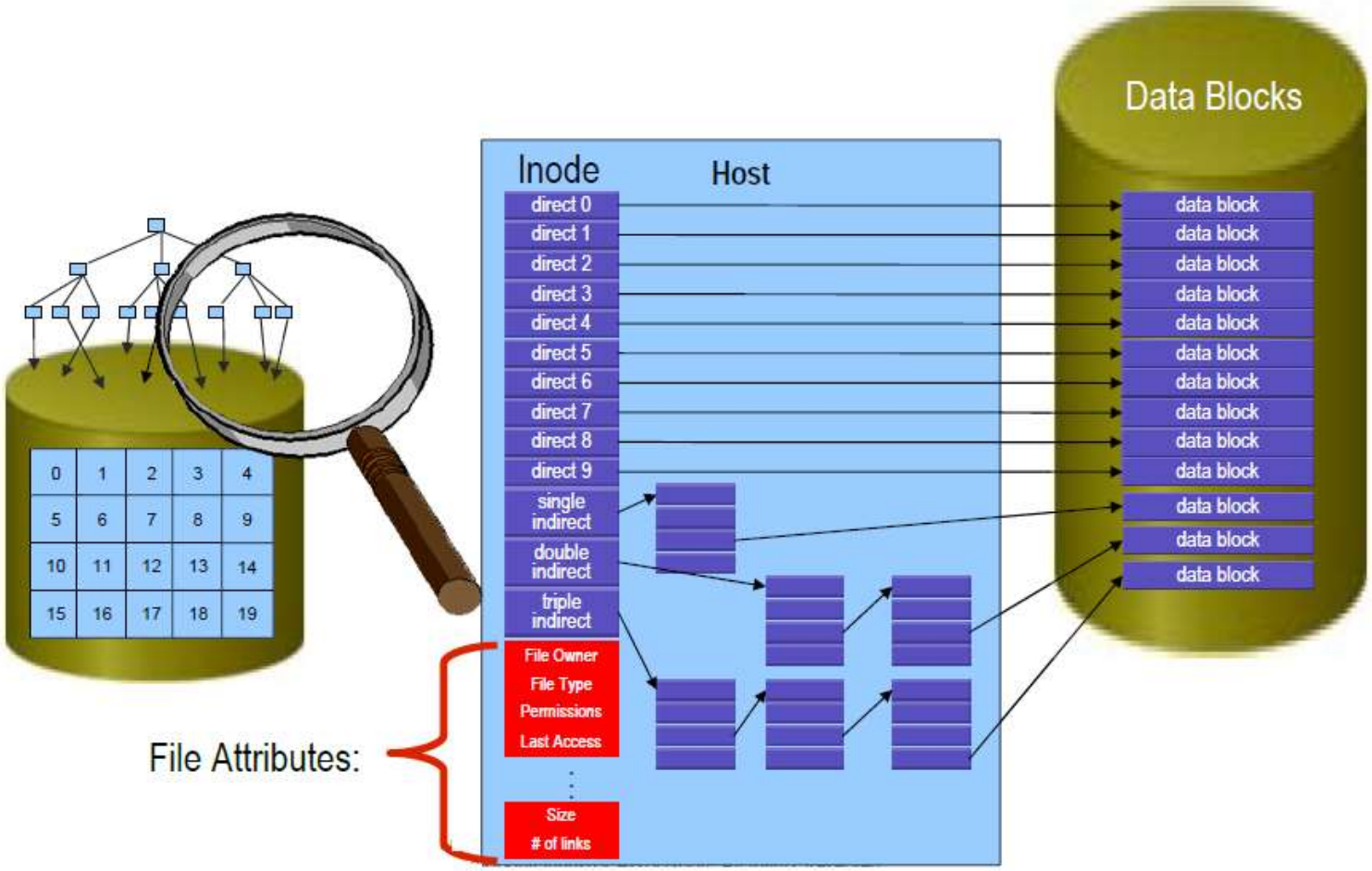


The Object Paradigm

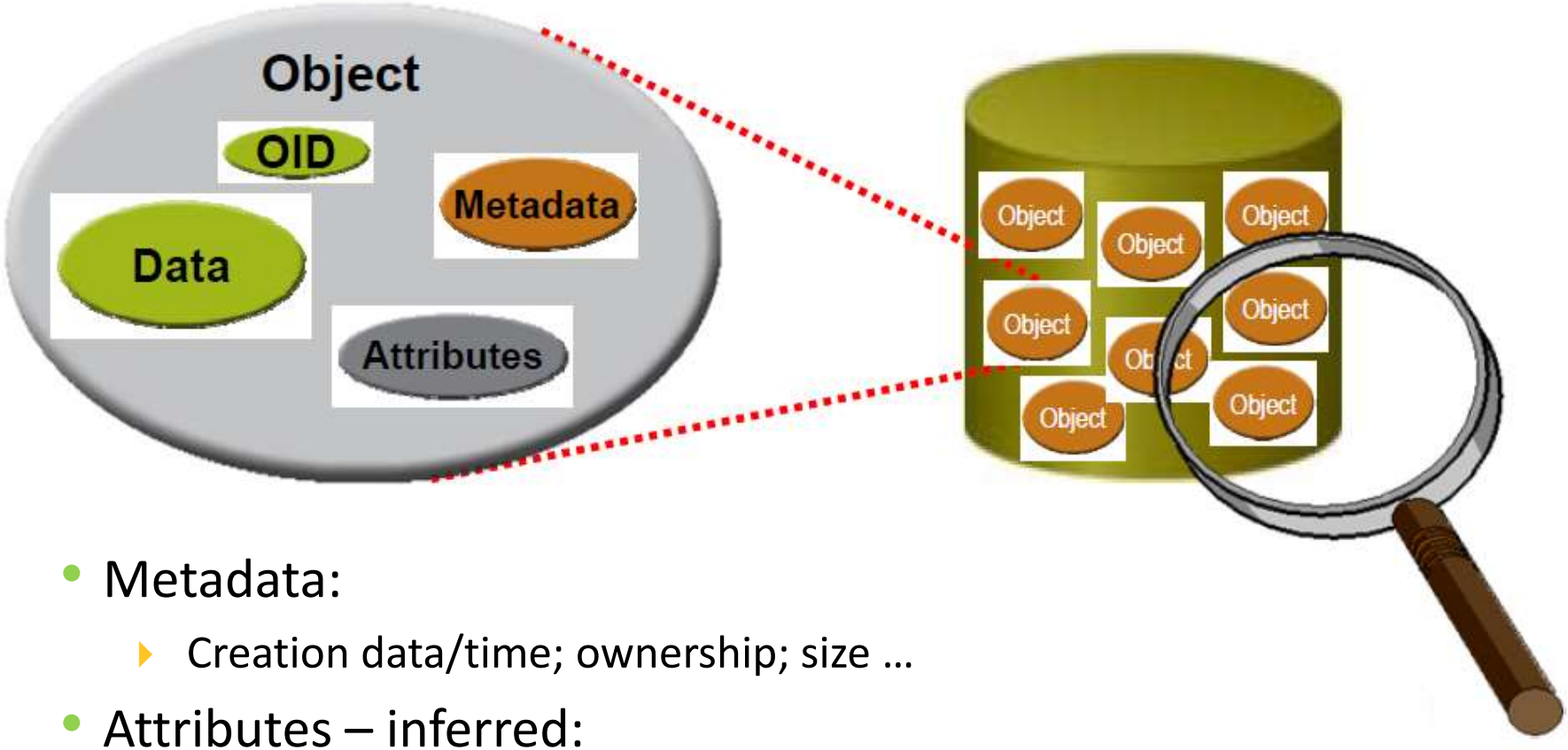


File Access via Inodes

- Inodes contain file attributes

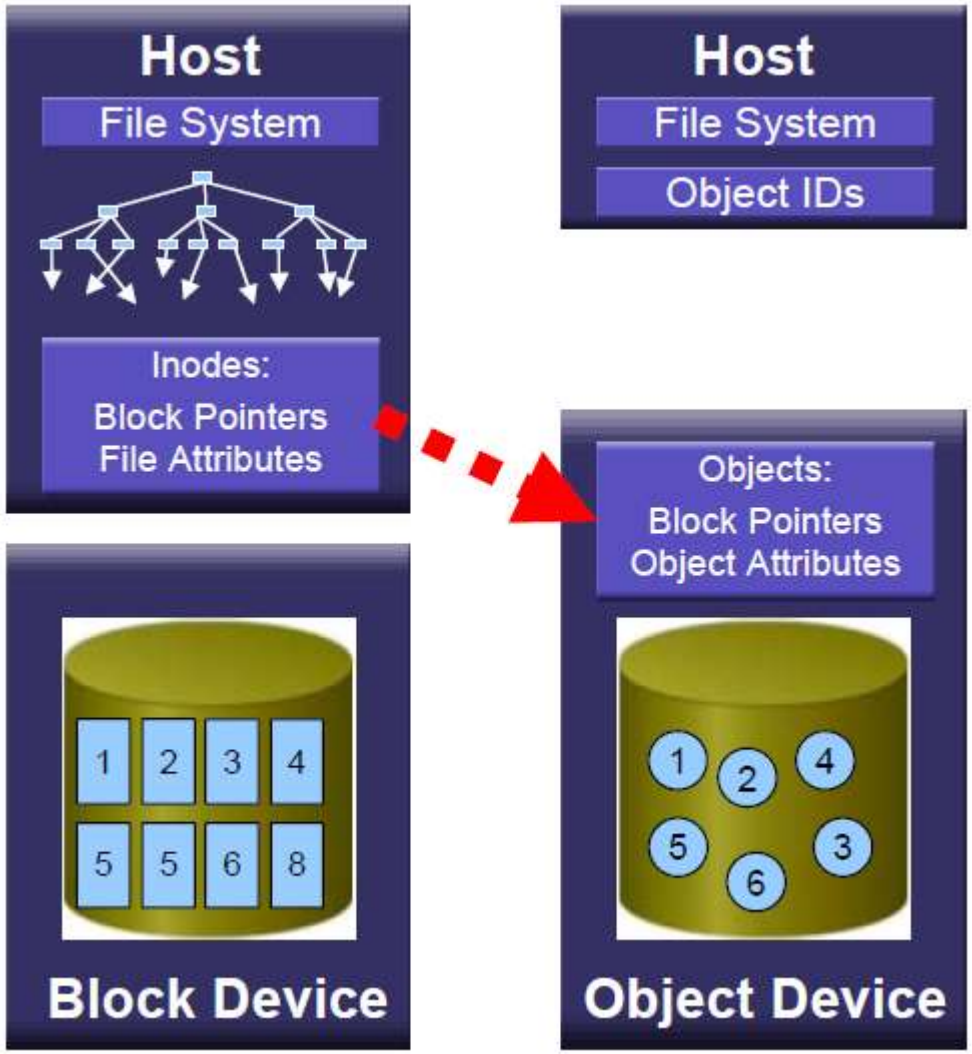


Object Access



- Metadata:
  - Creation data/time; ownership; size …
- Attributes – inferred:
  - Access patterns; content; indexes …
- Attributes – user supplied:
  - Retention; QoS …

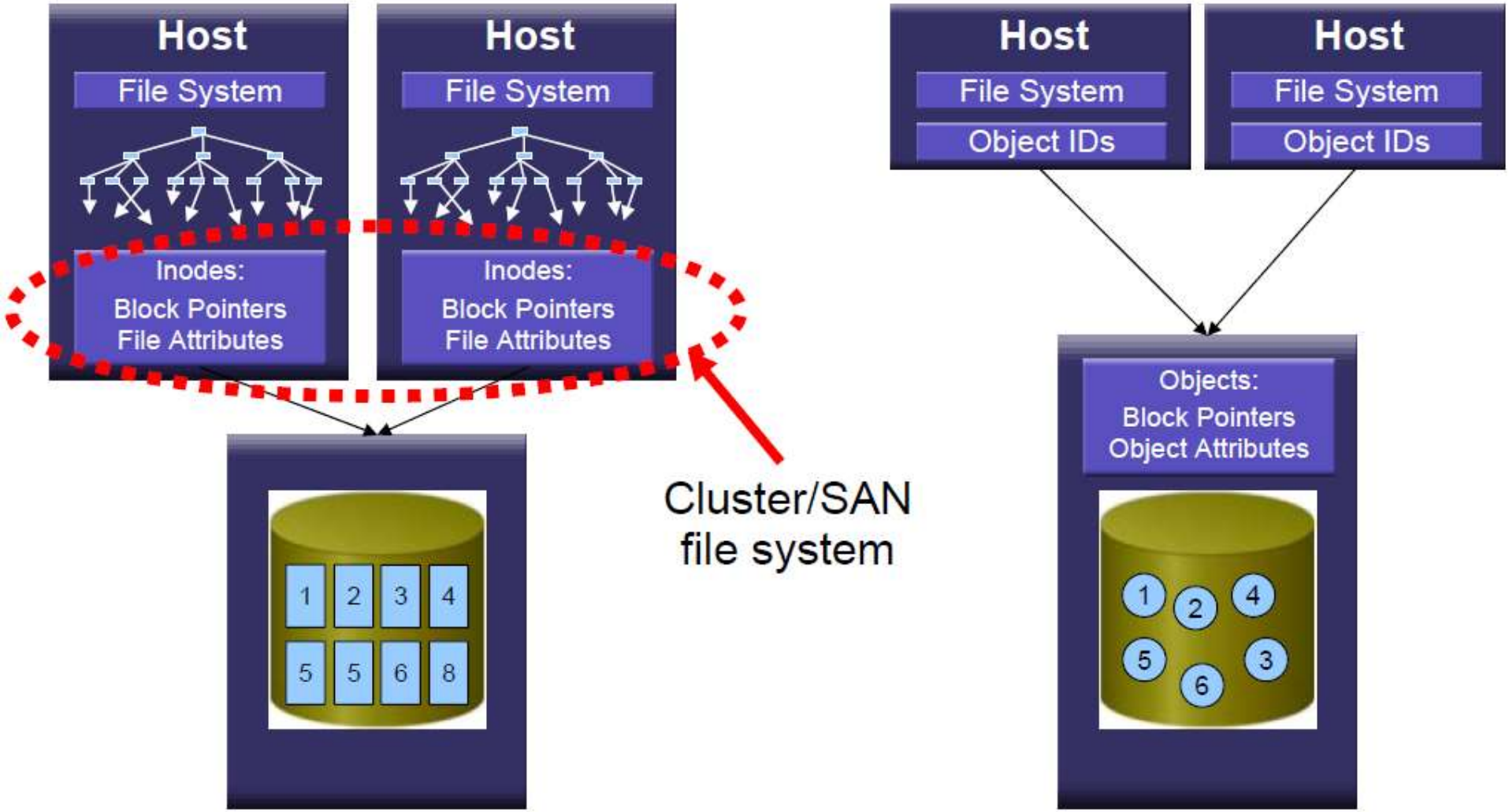
Object Autonomy



- Storage becomes autonomous
  - Capacity planning
  - Load balancing
  - Backup
  - QoS, SLAs
  - Understand data/object grouping
  - Aggressive prefetching
  - Thin provisioning
  - Search
  - Compression/Deduplication
  - Strong security, encryption
  - Compliance/retention
  - Availability/replication
  - Audit
  - Self healing

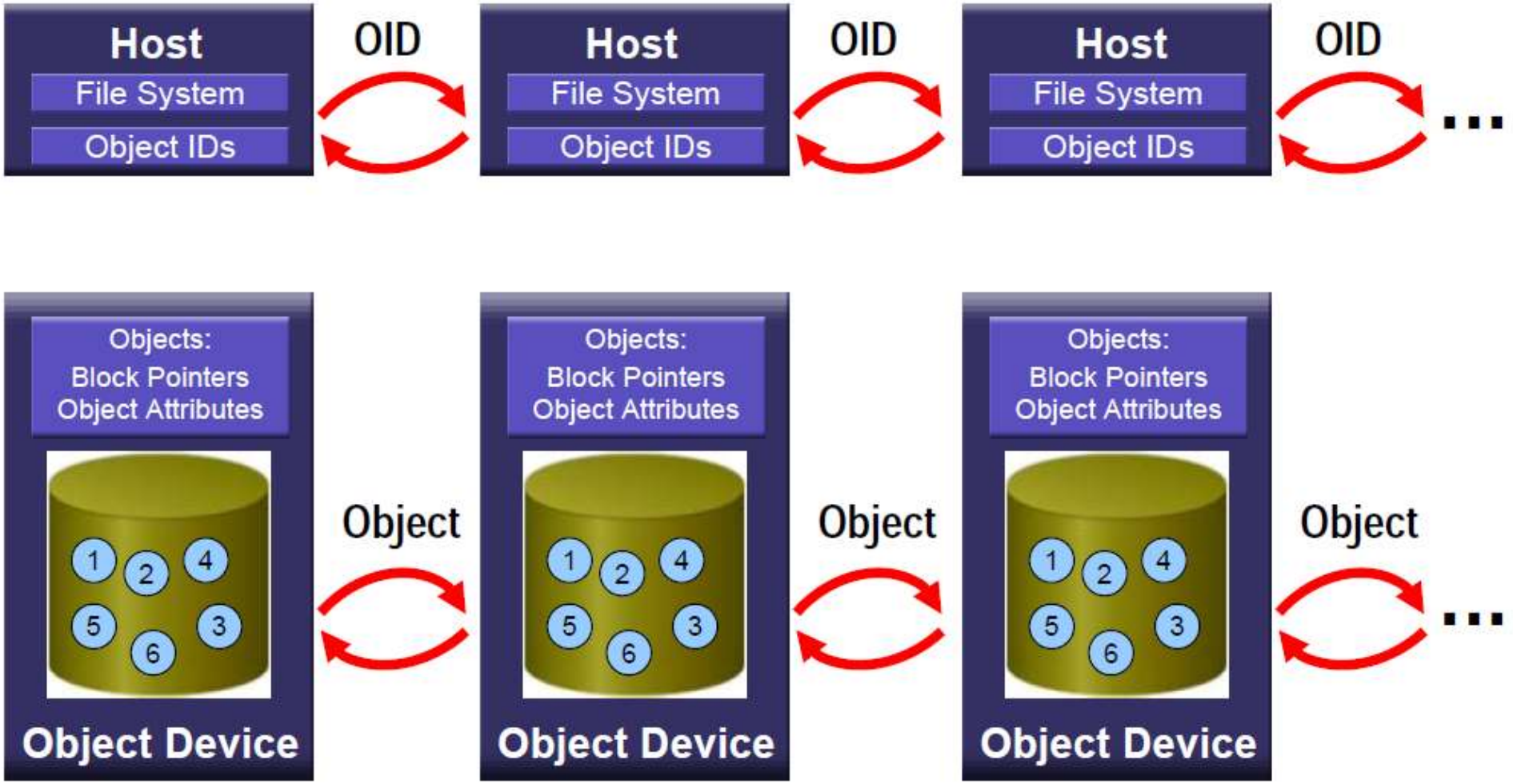
Data Sharing

homogeneous/heterogeneous

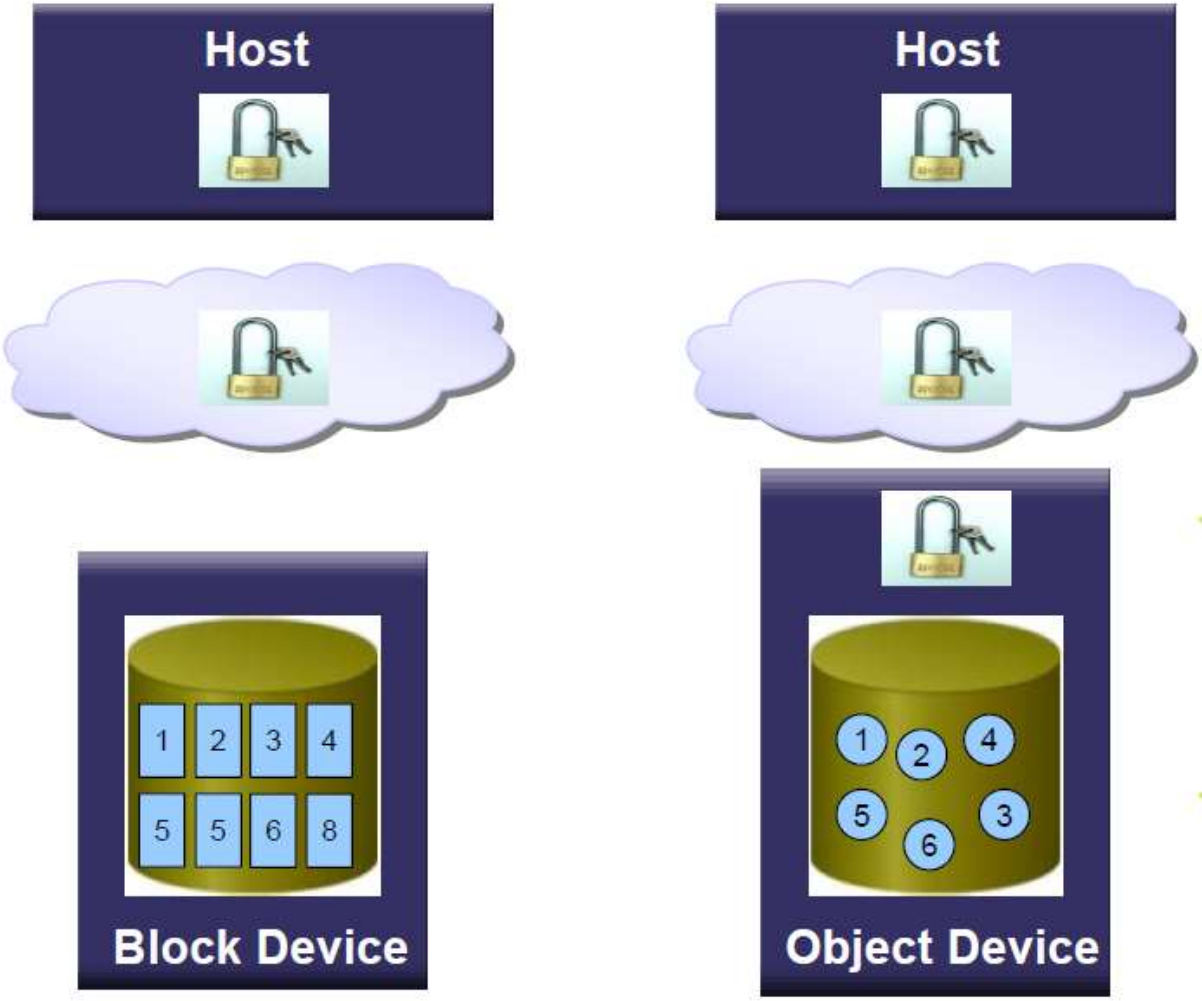


Data Migration

homogeneous/heterogeneous



Strong Security Additional layer

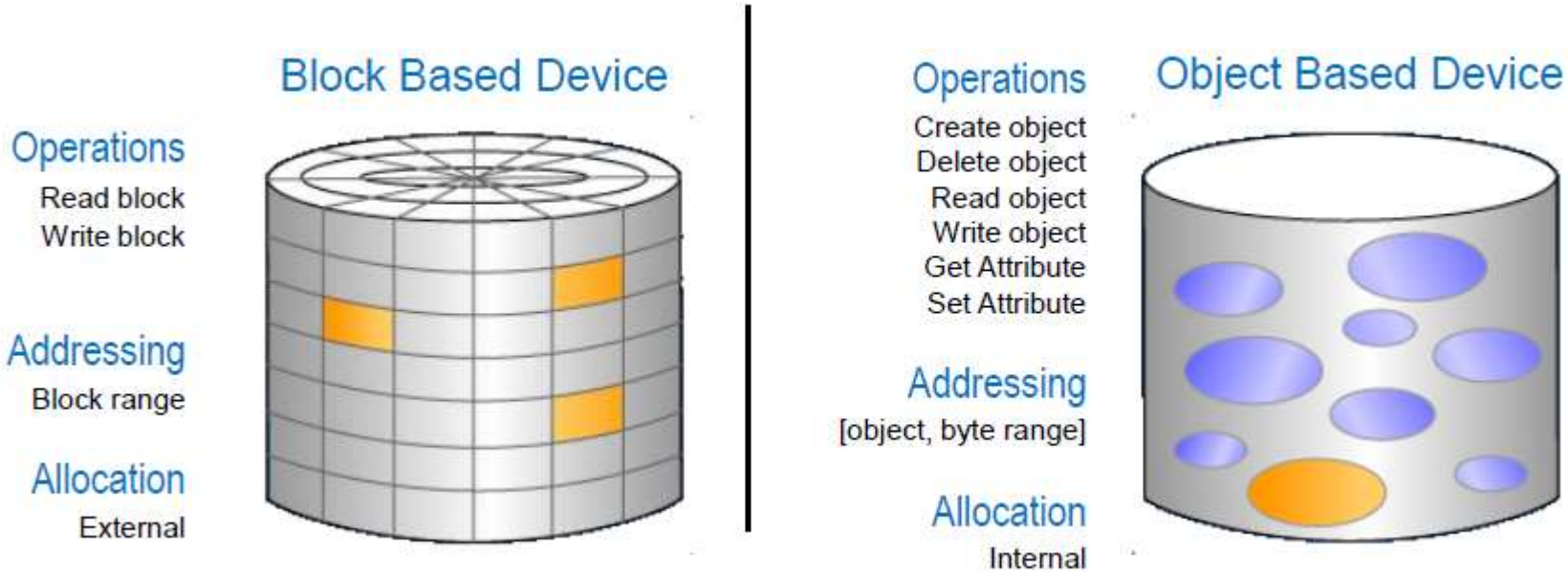


- Strong security via external service
  - Authentication
  - Authorization
  - …
- Fine granularity
  - Per object

2 Object-based Storage Devices

Data Access (Block-based vs. Object-based Device)

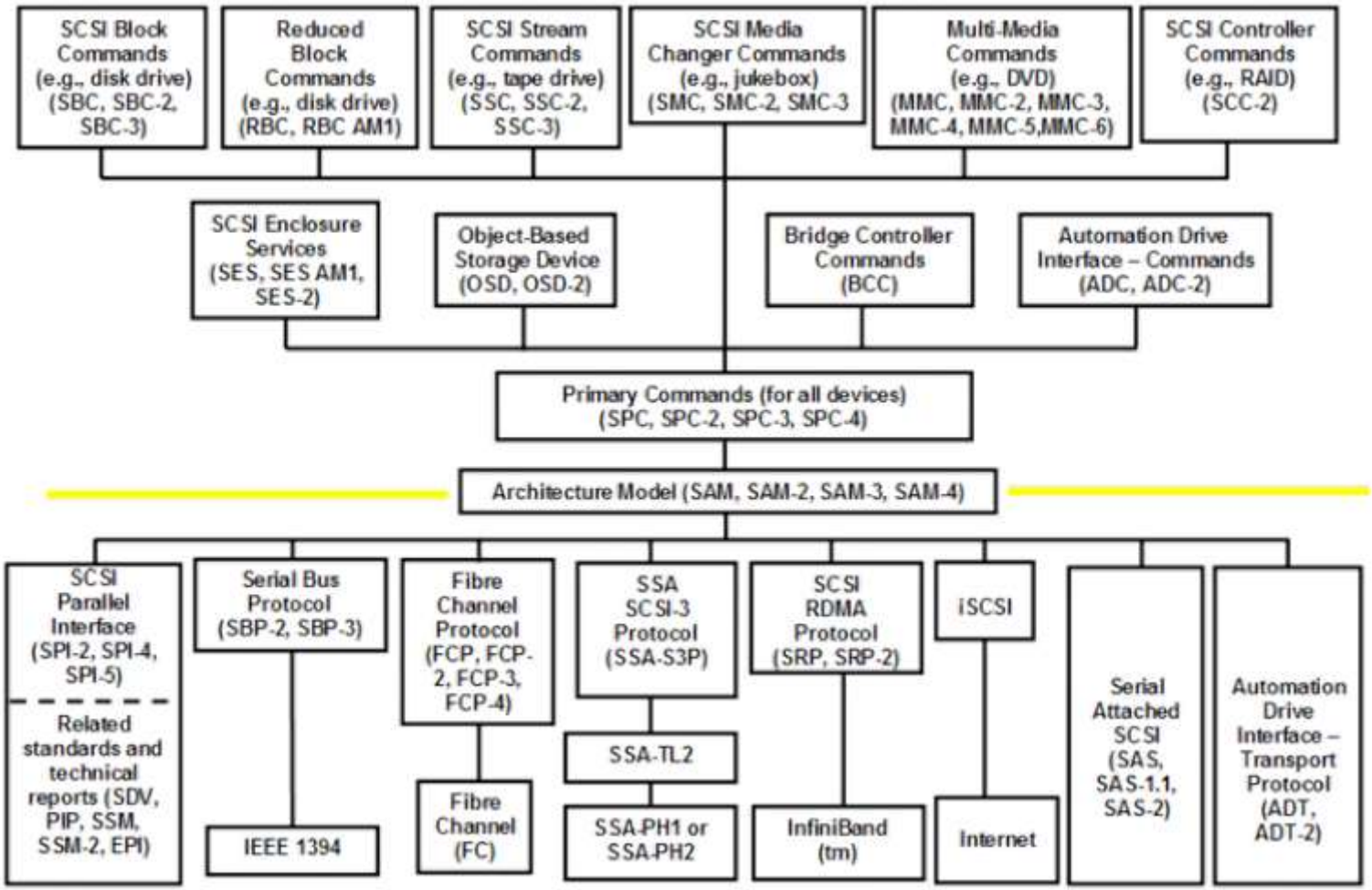
- Objects contain both data and attributes
  - Operations: create/delete/read/write objects, get/set attributes



OSD Standards (1)

- ANSI INCITS T10 for OSD (the SCSI Specification, www.t10.org)
  - ANSI INCITS 458
  - OSD-1 is basic functionality
    - Read, write, create objects and partitions
    - Security model, Capabilities, manage shared secrets and working keys
  - OSD-2 adds
    - Snapshots
    - Collections of objects
    - Extended exception handling and recovery
  - OSD-3 adds
    - Device to device communication
    - RAID-[1,5,6] implementation between/among devices

OSD Standards (2)



OSD Forms

- Disk array/server subsystem
  - Example: custom-built HPC systems predominantly deployed in national labs
- Storage bricks for objects
  - Example: commercial supercomputing offering
- Object Layer Integrated in Disk Drive

OSDs: like disks, only different

	Disk	OSD
Model	Array of blocks • Number never changes • Size never changes	Objects • Created and deleted • Grow and shrink
Operations	Read/write disk blocks	Create/delete object Read/write object blocks
Security	Zoning, LUN masking •Applies to entire device	Capability-based •Applies to each object and op
Typical transports	Fibre Channel, SCSI, iSCSI	iSCSI, **ONC-RPC over TCP/IP**

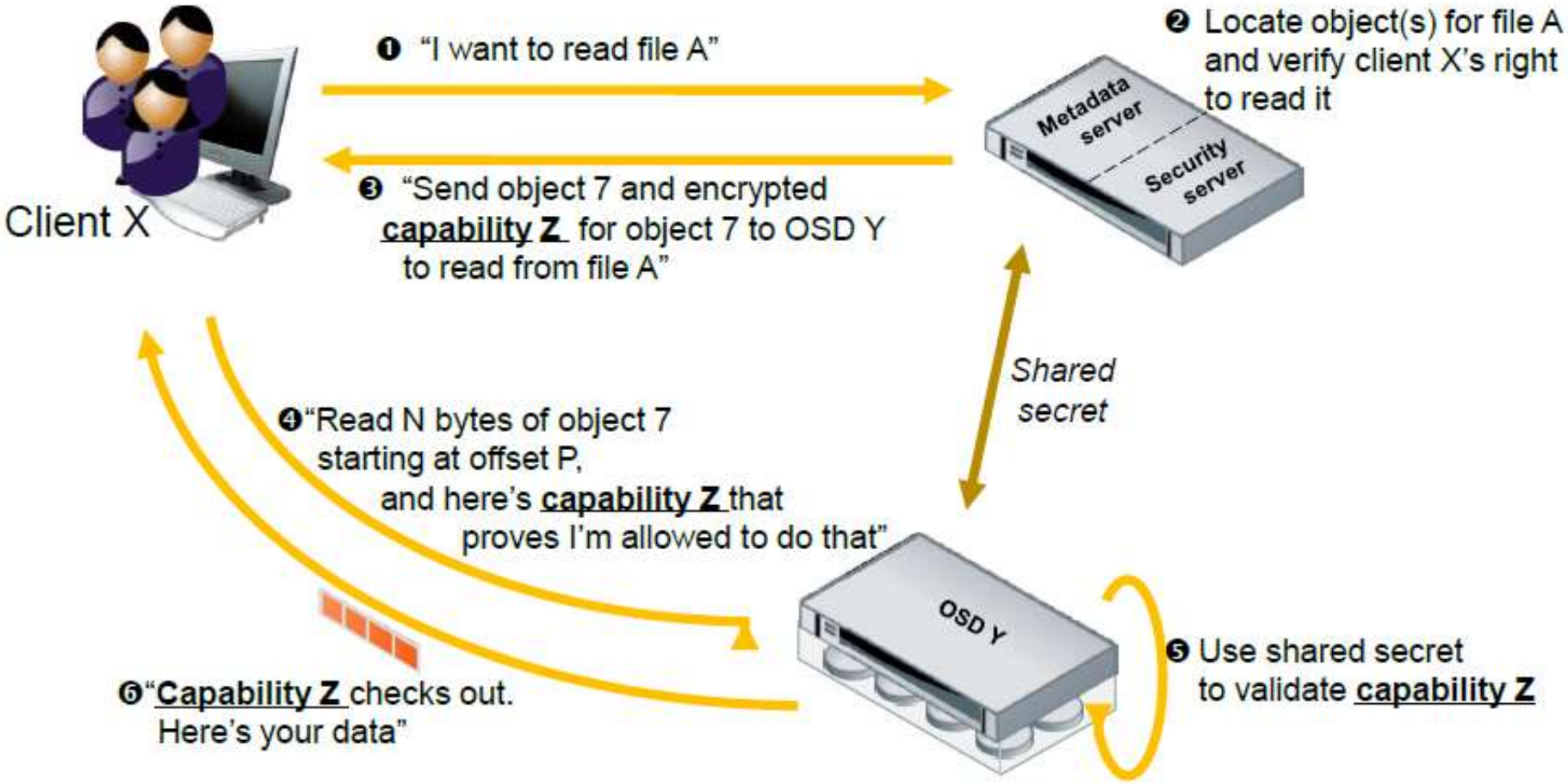
OSDs: like a file server, only different

	File server	OSD
Model	Files	Objects
Naming	Human-readable names in a hierarchical directory tree	Two level name space: 64 bit object "name" in a 64 bit partition "name"
Operations	File: create, delete, rename File byte range: read, write, append, truncate	Object: create, delete, Object block range: read, write, append, truncate
Security	User | group | world × `rwx` or access control lists • Checked at initial file access	Digitally signed ***capabilities*** • Checked for every I/O request

OSD Capabilities (1)

- Unlike disks, where access is granted on an all or nothing basis, OSDs grant or deny access to individual objects based on **Capabilities**
- A Capability must accompany each request to read or write an object
  - Capabilities are cryptographically signed by the Security Manager and verified (and enforced) by the OSD
  - A Capability to access an object is created by the Security Manager, and given to the client (application server) accessing the object
  - Capabilities can be revoked by changing an attribute on the object

OSD Capabilities (2)



OSD Security Model

- OSD and File Server know a secret key
  - Working keys are periodically generated from a master key
- File server authenticates clients and makes access control policy decisions
  - Access decision is captured in a capability that is signed with the secret key
  - Capability identifies object, expire time, allowed operations, etc.
- Client signs requests using the capability signature as a signing key
  - OSD verifies the signature before allowing access
  - OSD doesn't know about the users, Access Control Lists (ACLs), or whatever policy mechanism the File Server is using

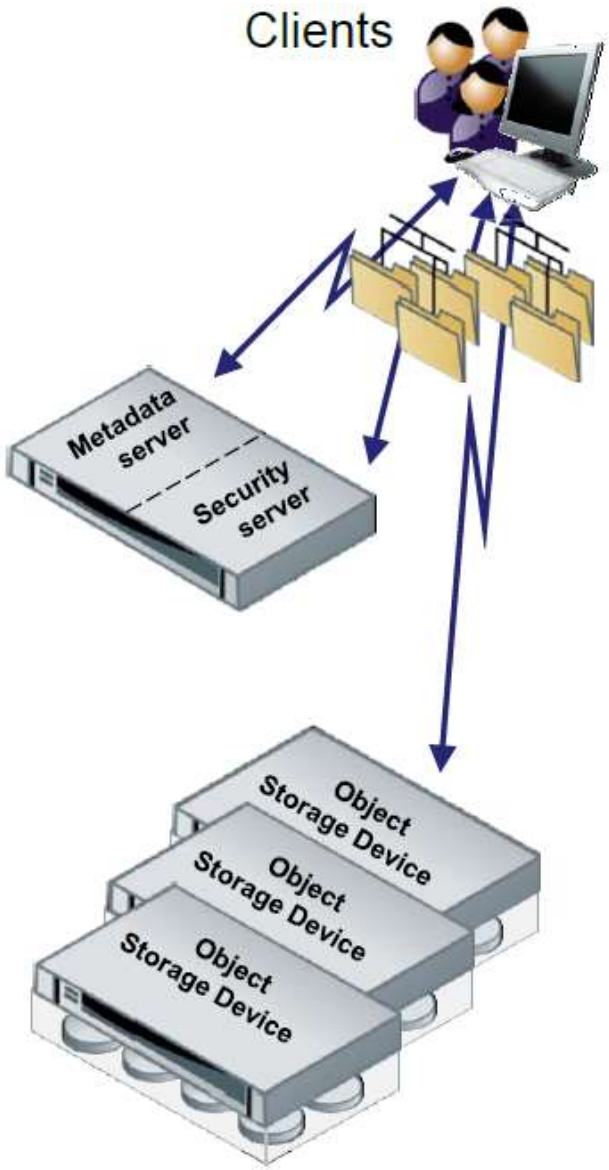
3 Object-based File Systems

Why not just OSD = file system?

- Scaling
  - What if there's more data than the biggest OSD can hold?
  - What if too many clients access an OSD at the same time?
  - What if there's a file bigger than the biggest OSD can hold?
- Robustness
  - What happens to data if an OSD fails?
  - What happens to data if a Metadata Server fails?
- Performance
  - What if thousands of objects are access concurrently?
  - What if big objects have to be transferred really fast?

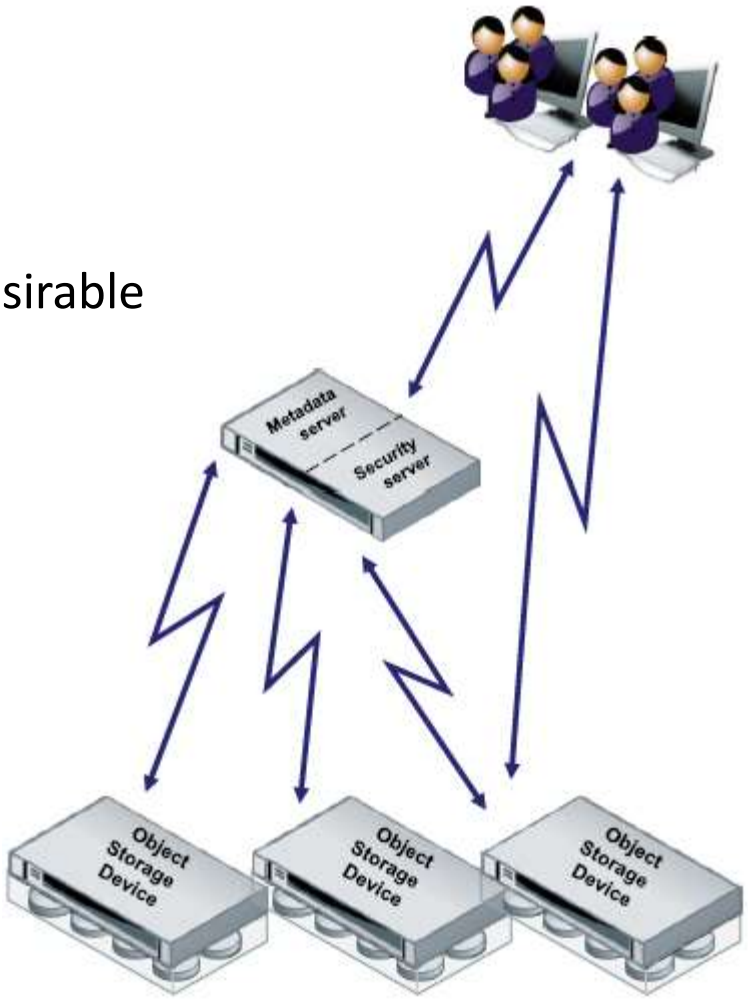
General Principle

- Architecture
  - File = one or more groups of objects
    - Usually on different OSDs
  - Clients access Metadata Servers to locate data
  - Clients transfer data directly to/from OSDs
- Address
  - Capacity
  - Robustness
  - Performance



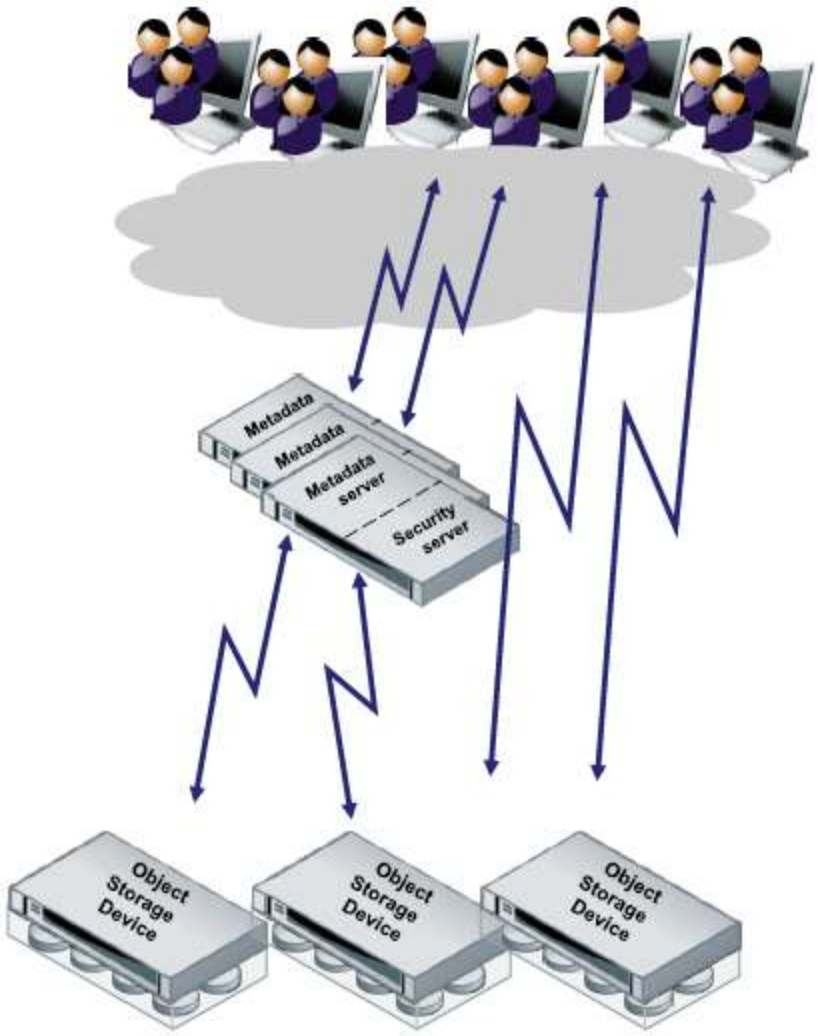
Capacity

- Add OSDs
  - Increase total system capacity
  - Support bigger files
    - Files can span OSDs if necessary or desirable



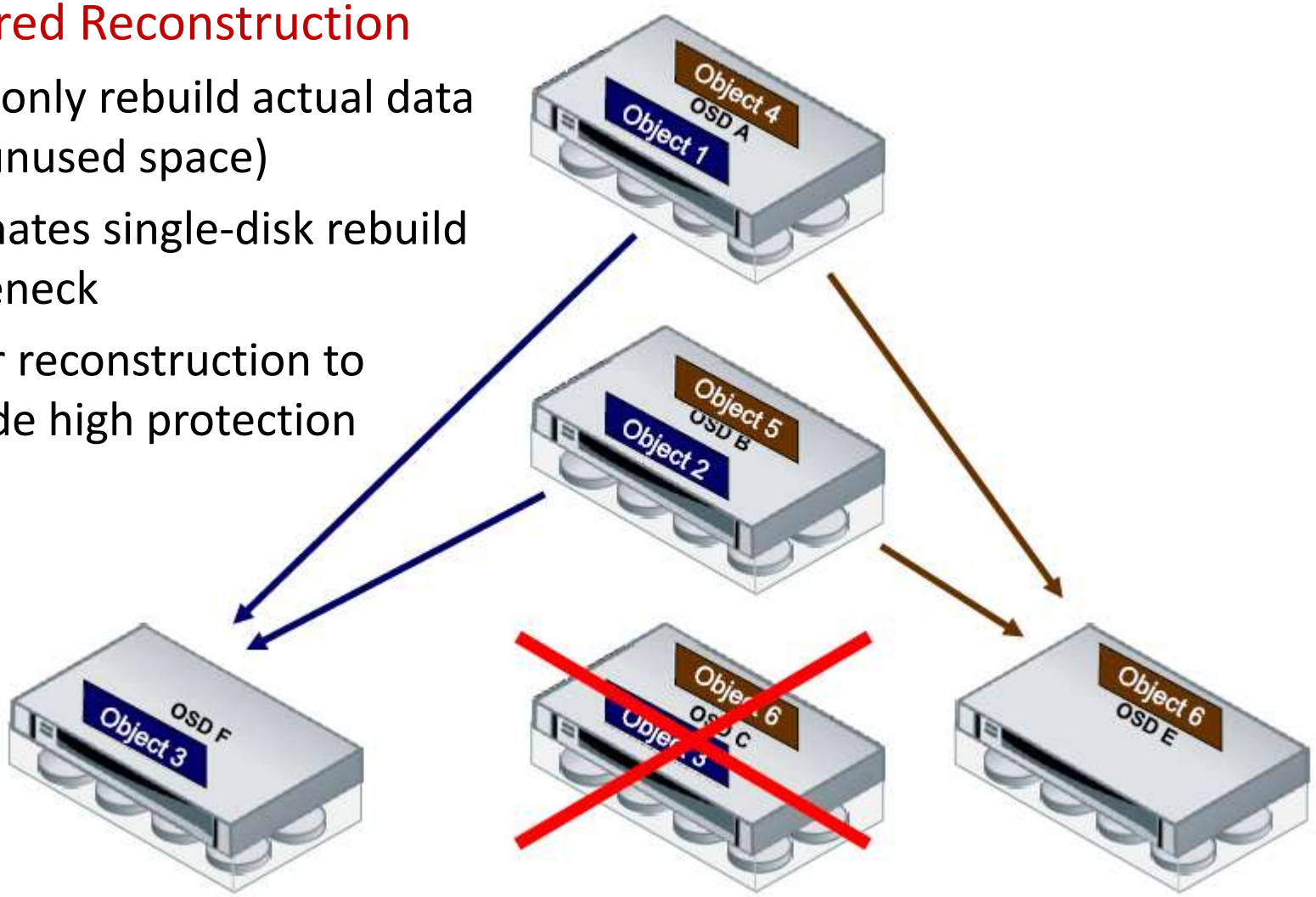
Robustness

- Add metadata servers
  - Resilient metadata services
  - Resilient security services
- Add OSDs
  - Failed OSD affects small percentage of system resources
  - Inter-OSD mirroring and RAID
  - Near-online file system checking



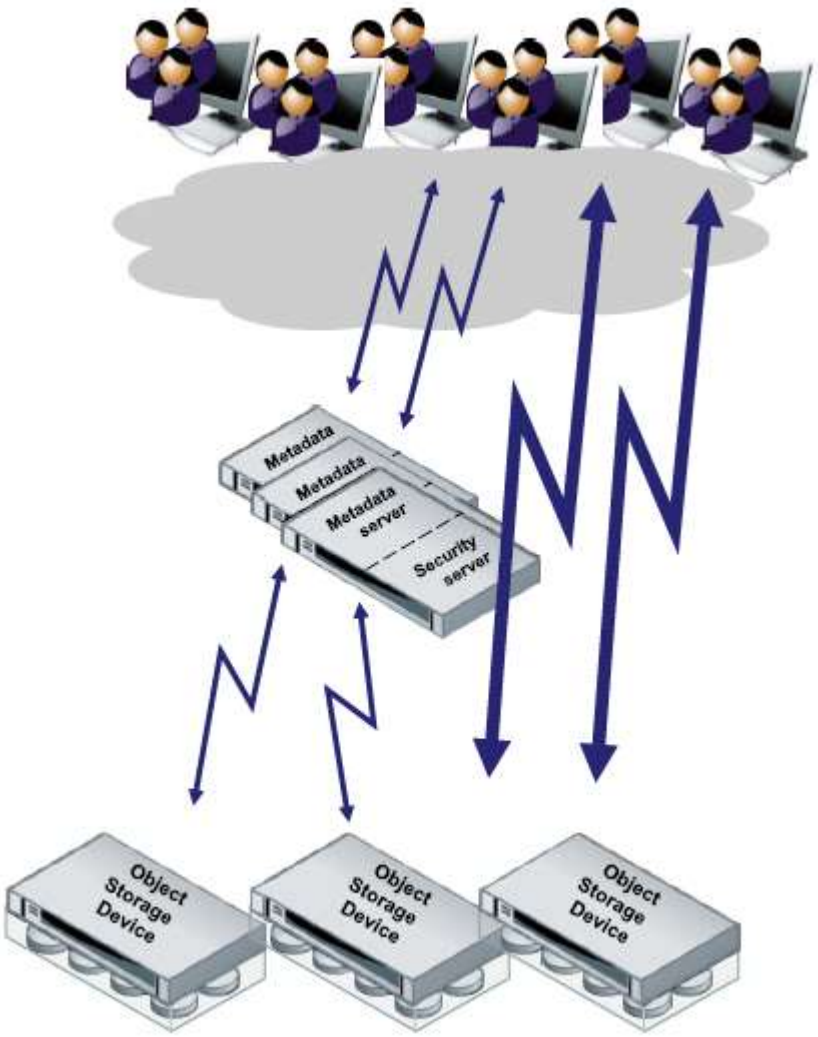
Advantage of Reliability

- Declustered Reconstruction
  - OSDs only rebuild actual data (not unused space)
  - Eliminates single-disk rebuild bottleneck
  - Faster reconstruction to provide high protection



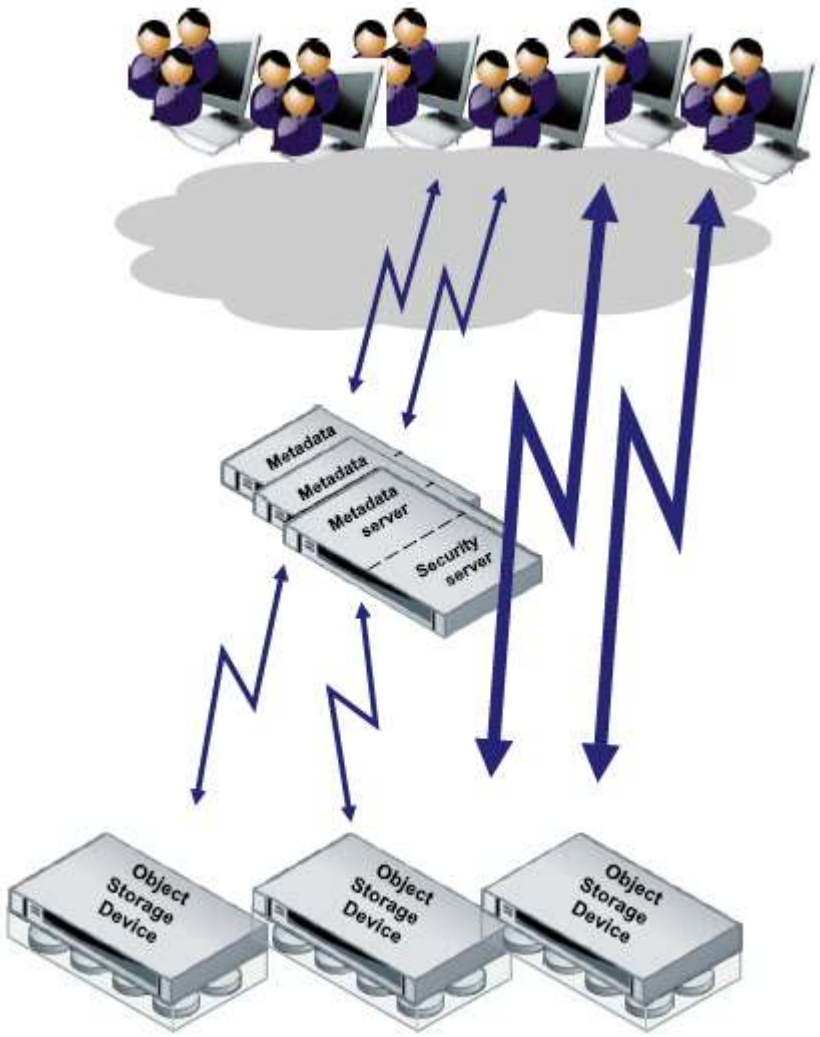
Performance

- Add metadata servers
  - More concurrent metadata operations
    - Getattr, Readdir, Create, Open, …
- Add OSDs
  - More concurrent I/O operations
  - More bandwidth directly between clients and data

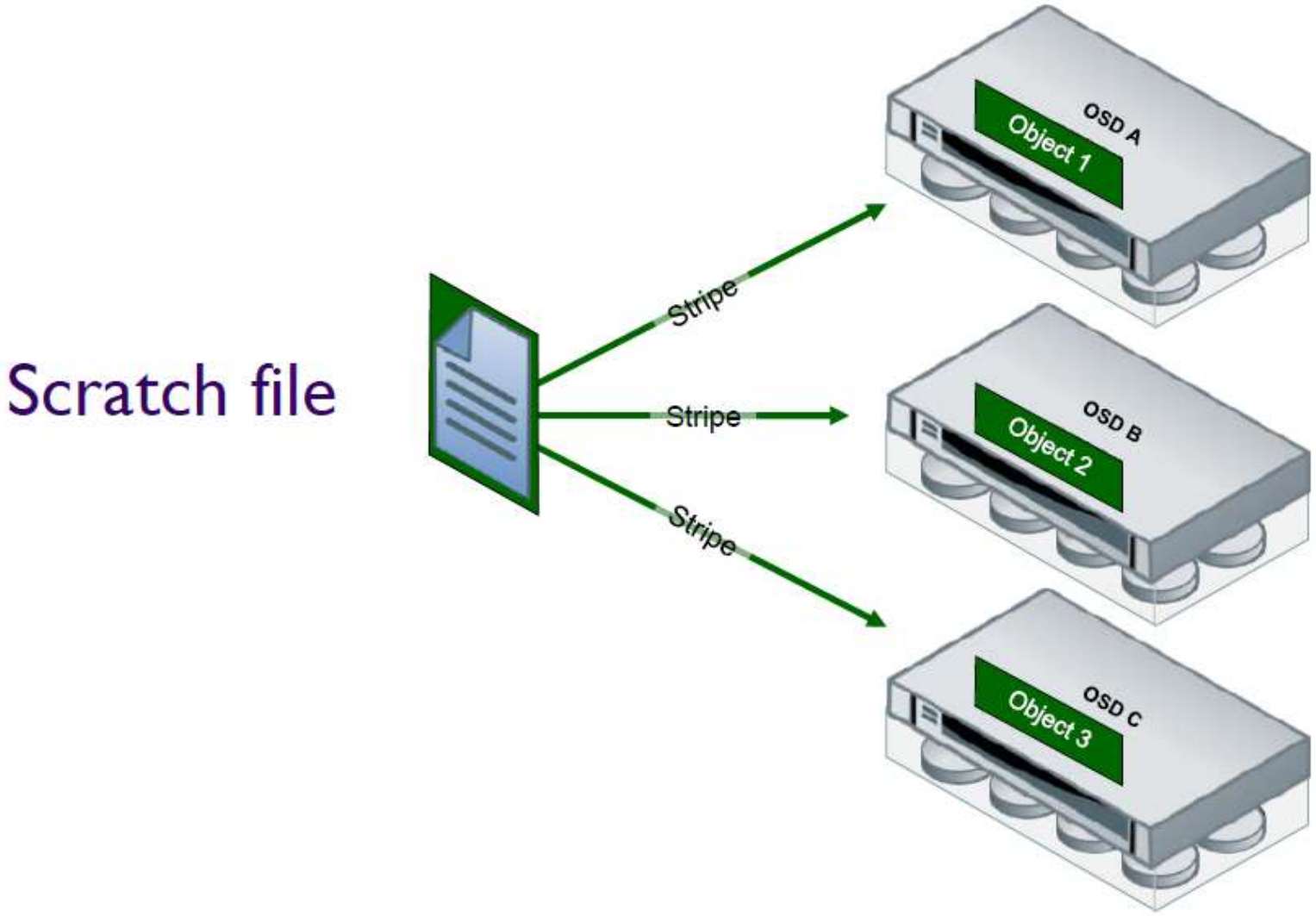


Additional Advantages

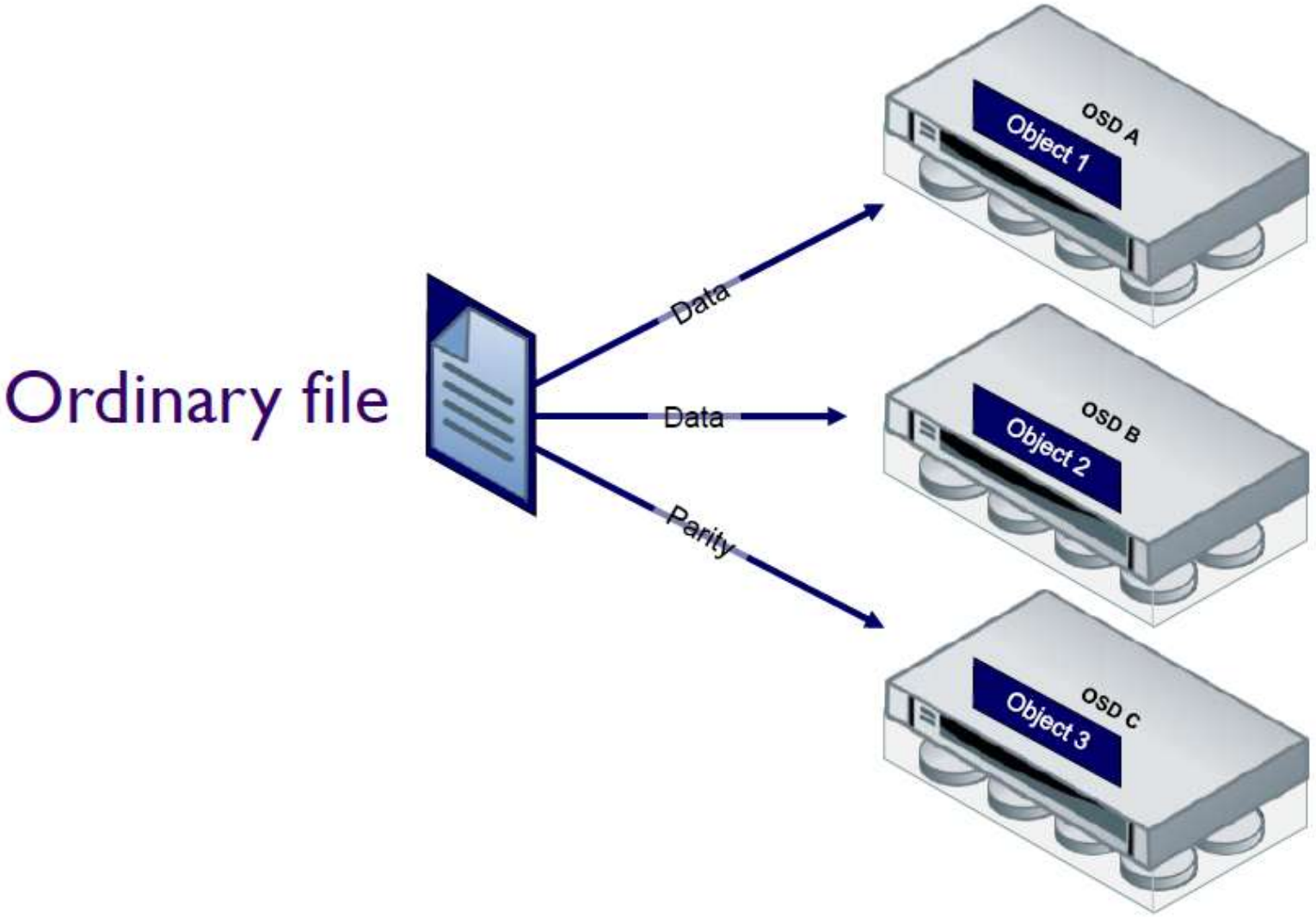
- Optimal data placement
  - Within OSD: proximity of related data
  - Load balancing across OSDs
- System-wide storage pooling
  - Across multiple file systems
- Storage tiering
  - Per-file control over performance and resiliency



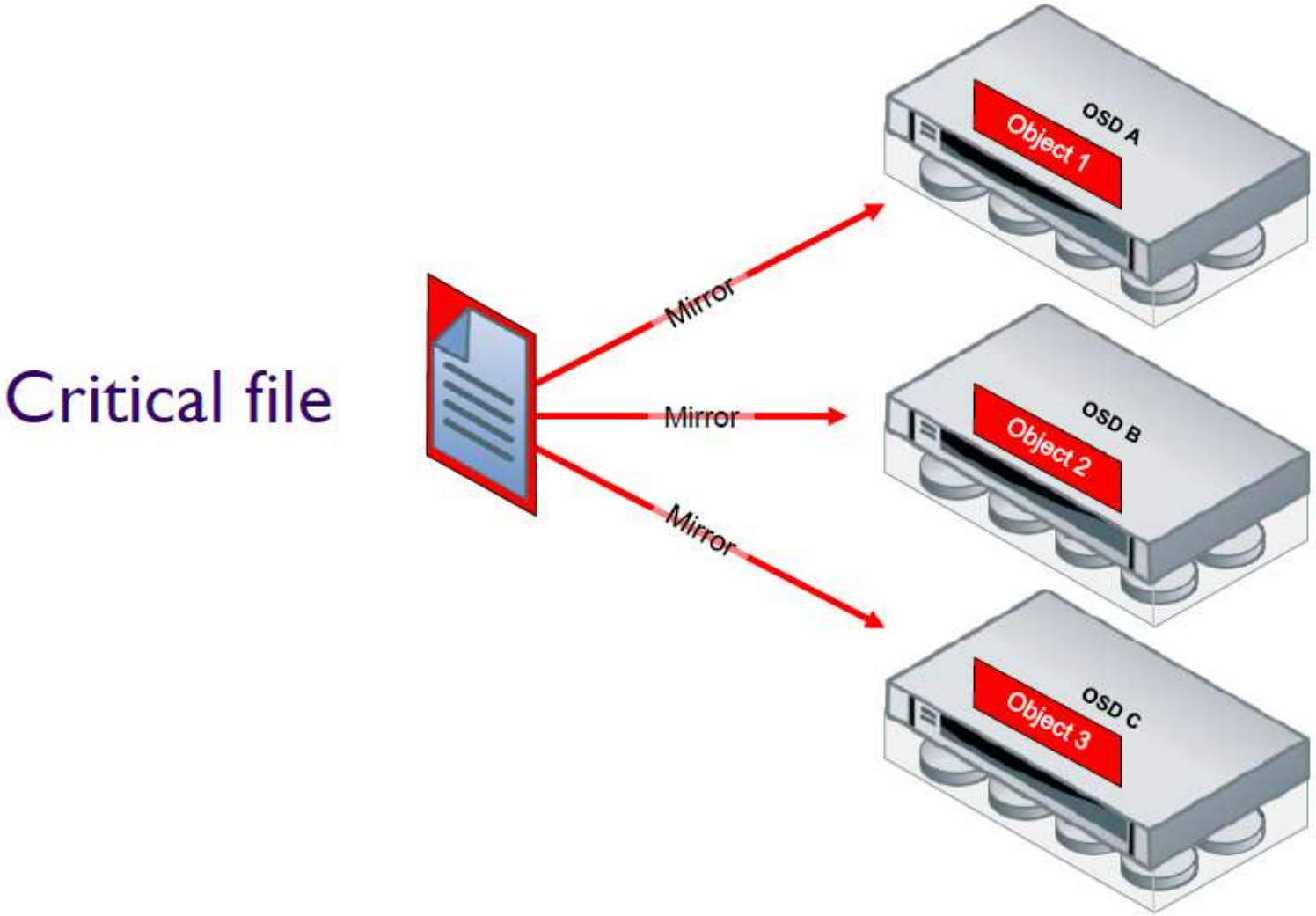
Per-file tiering in OSDs: striping



Per-file tiering in OSDs: RAID-4/5/6

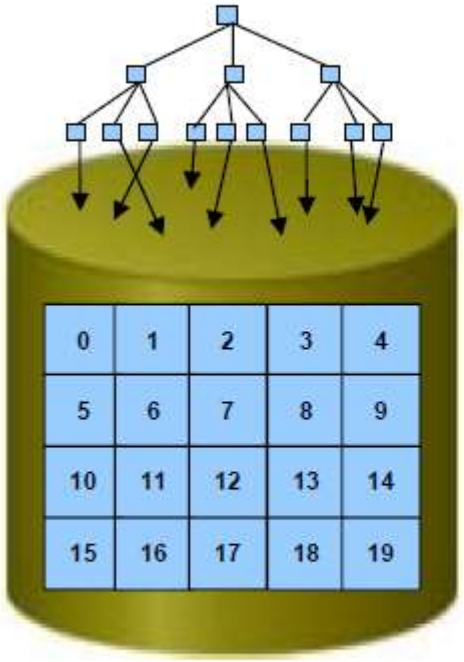


Per-file tiering in OSDs: mirroring(RAID-1)



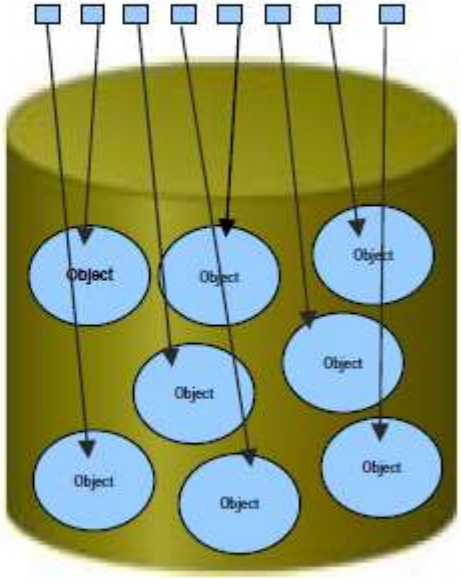
Flat namespace

File names / inodes



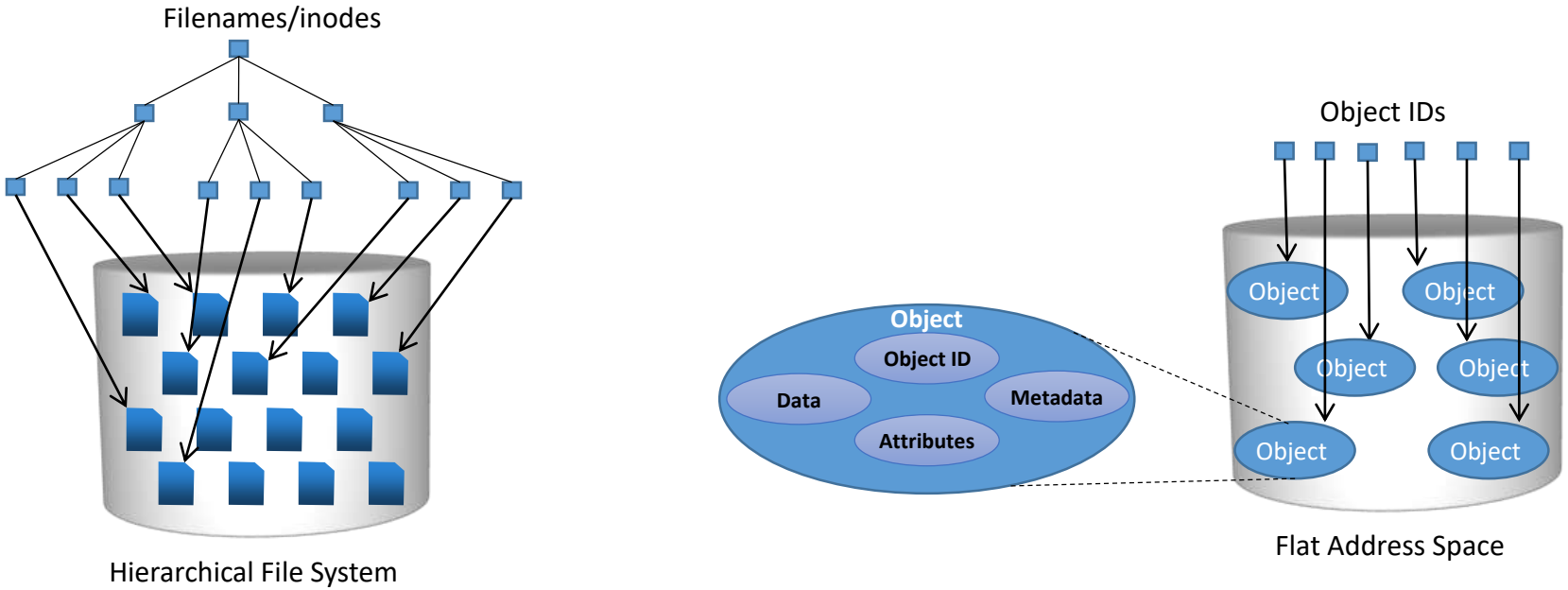
Traditional
Hierarchical

Objects / OIDs



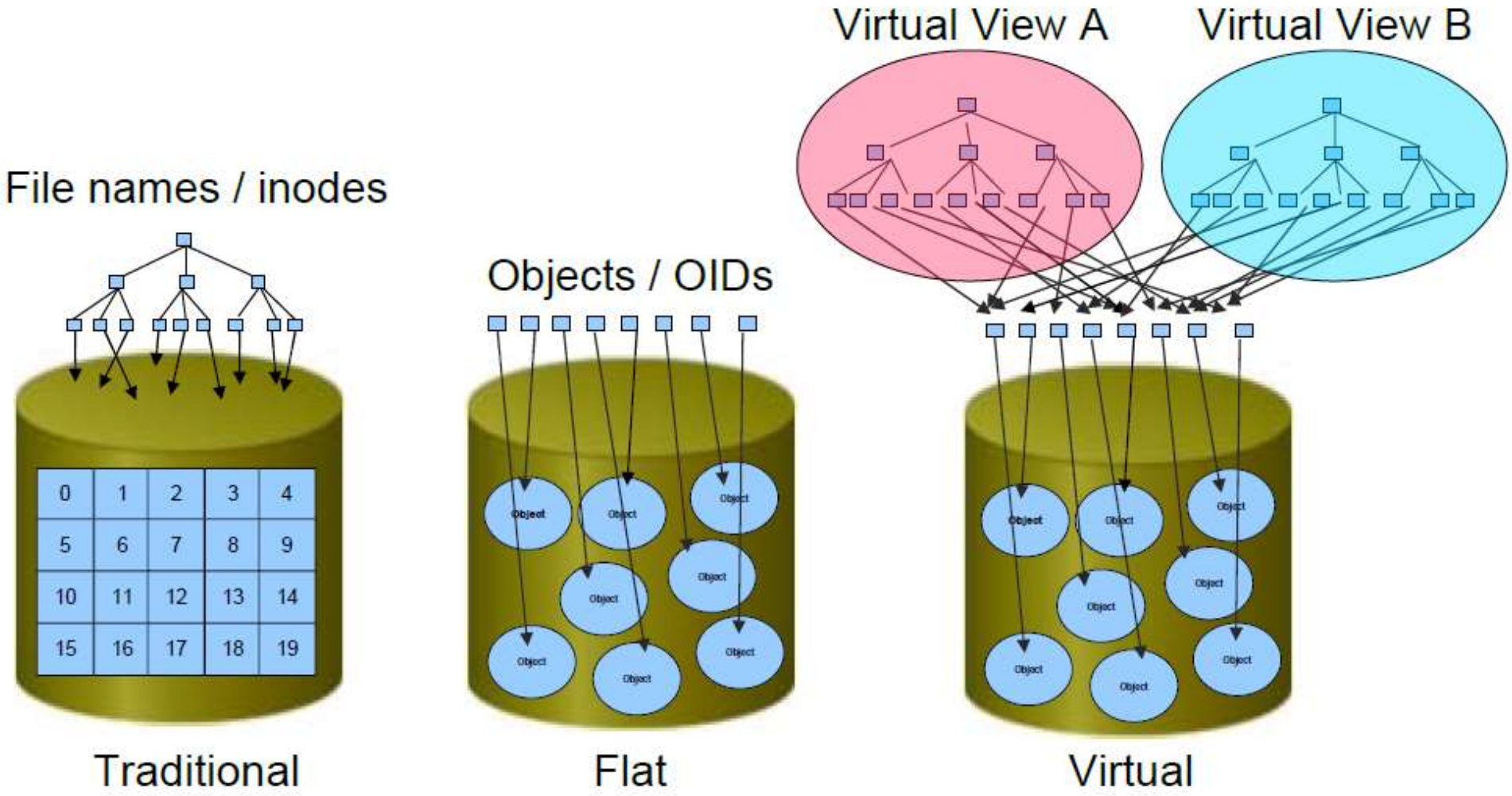
Flat

Hierarchical File System Vs. Flat Address Space

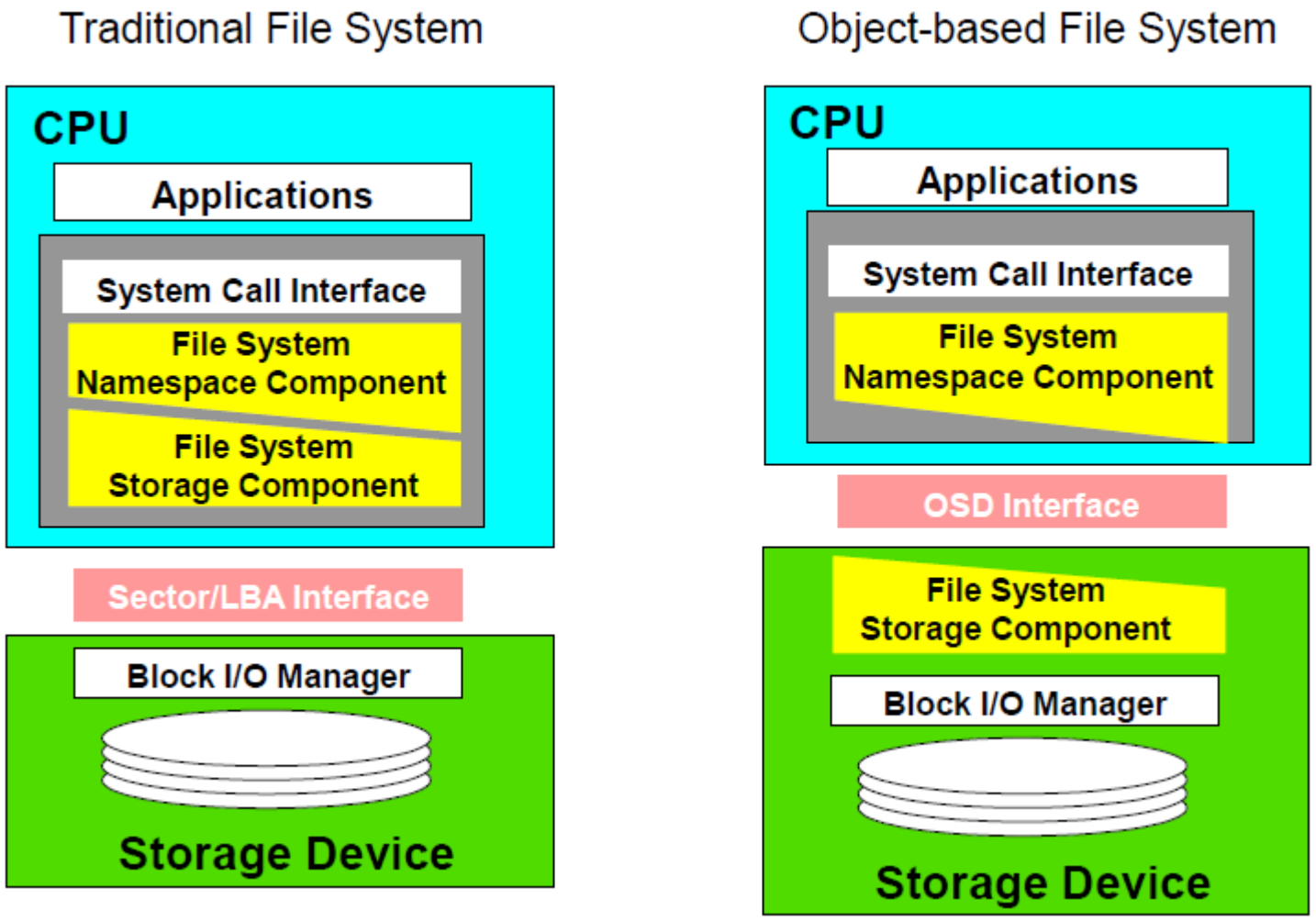


- Hierarchical file system organizes data in the form of files and directories
- Object-based storage devices store the data in the form of objects
  - It uses flat address space that enables storage of large number of objects
  - An object contains user data, related metadata, and other attributes
  - Each object has a unique object ID, generated using specialized algorithm

Virtual View / Virtual File Systems



Traditional FS Vs. Object-based FS (1)

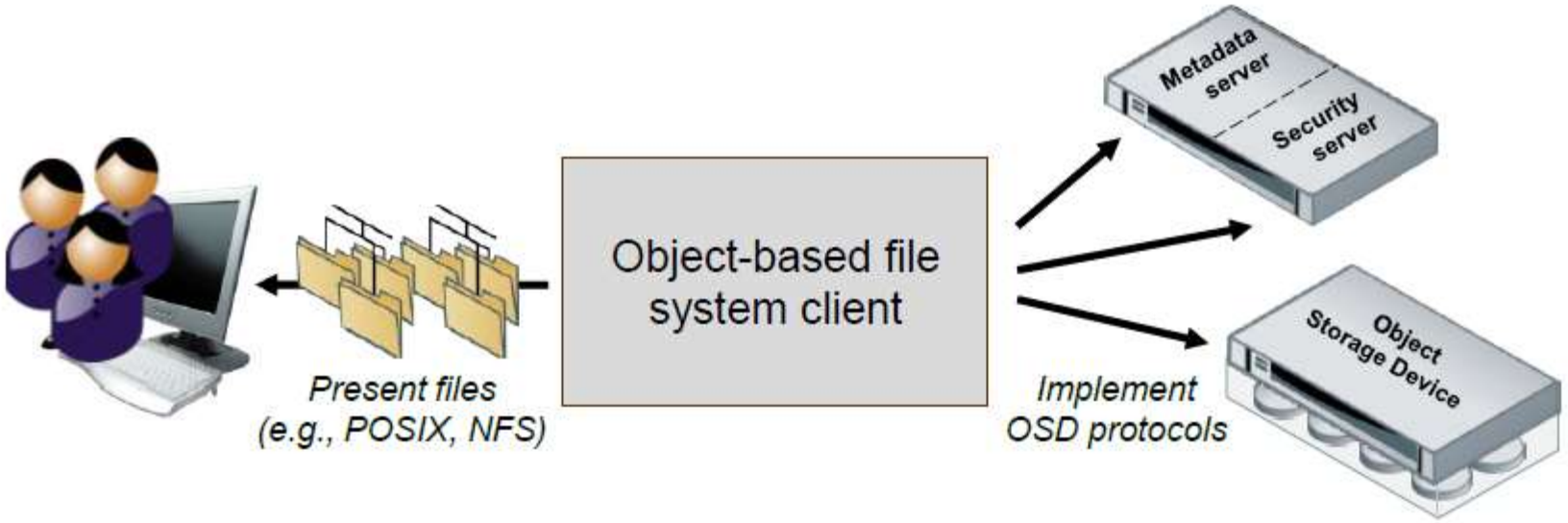


Traditional FS Vs. Object-based FS (2)

- File system layer in host manages
  - Human readable namespace
  - User authentication, permission checking, Access Control Lists (ACLs)
  - OS interface
- Object Layer in OSD manages
  - Block allocation and placement
  - OSD has better knowledge of disk geometry and characteristic so it can do a better job of file placement/optimization than a host-based file system

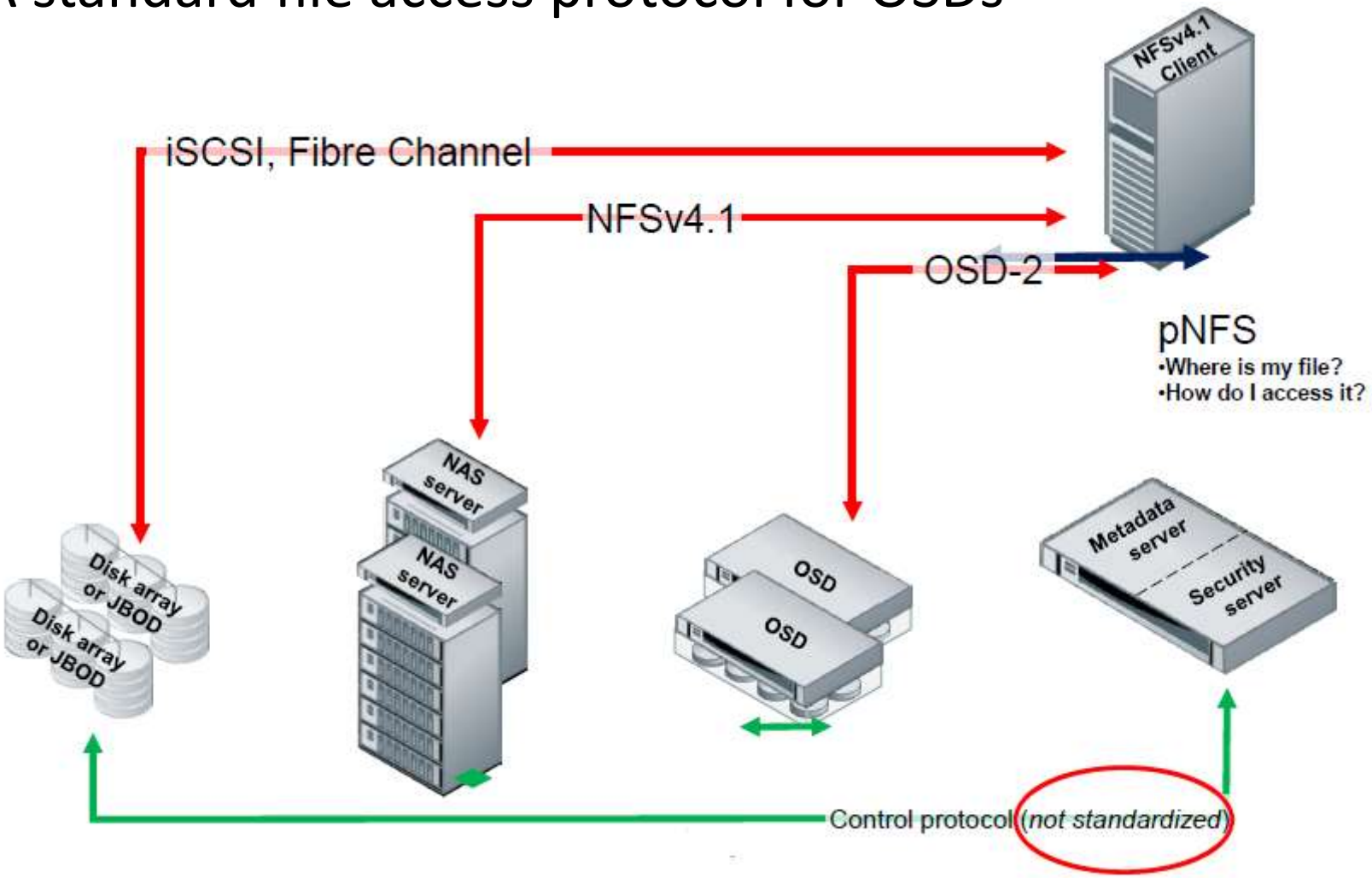
Accessing Object-based FS

- Typical Access
  - SCSI (block), NFS/CIFS (file)
- Needs a client component
  - Proprietary
  - Standard

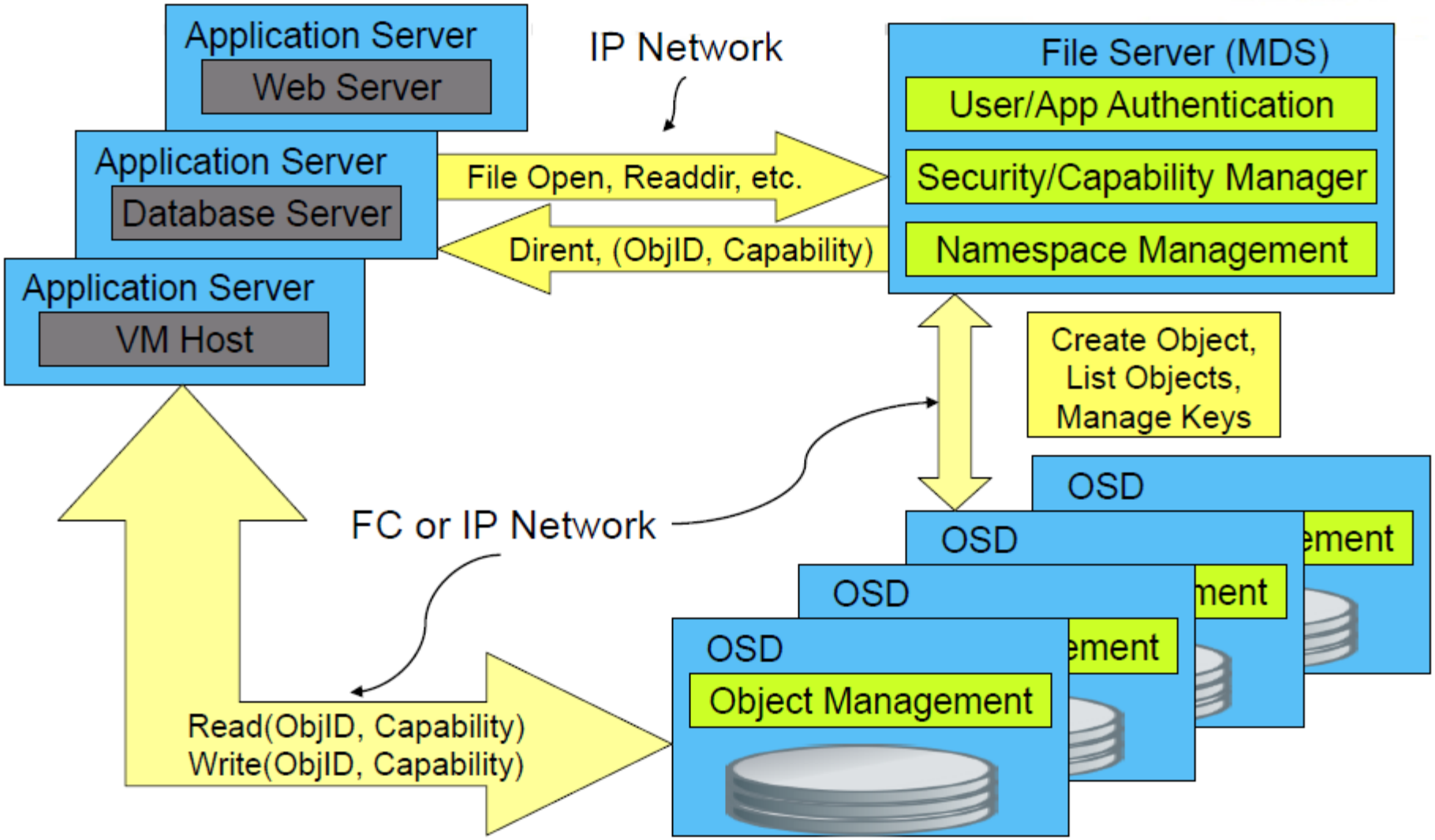


Standard→ NFS v4.1

- A standard file access protocol for OSDs



Scaling Object-based FS (1)



Scaling Object-based FS (2)

- App servers (clients) have direct access to storage to read/write file data securely
  - Contrast with SAN where security is lacking
  - Contrast with NAS where server is a bottleneck
- File system includes multiple OSDs
  - Grow the file system by adding an OSD
  - Increase bandwidth at the same time
  - Can include OSDs with different performance characteristics (SSD, SATA, SAS)
- Multiple File Systems share the same OSDs
  - Real storage pooling

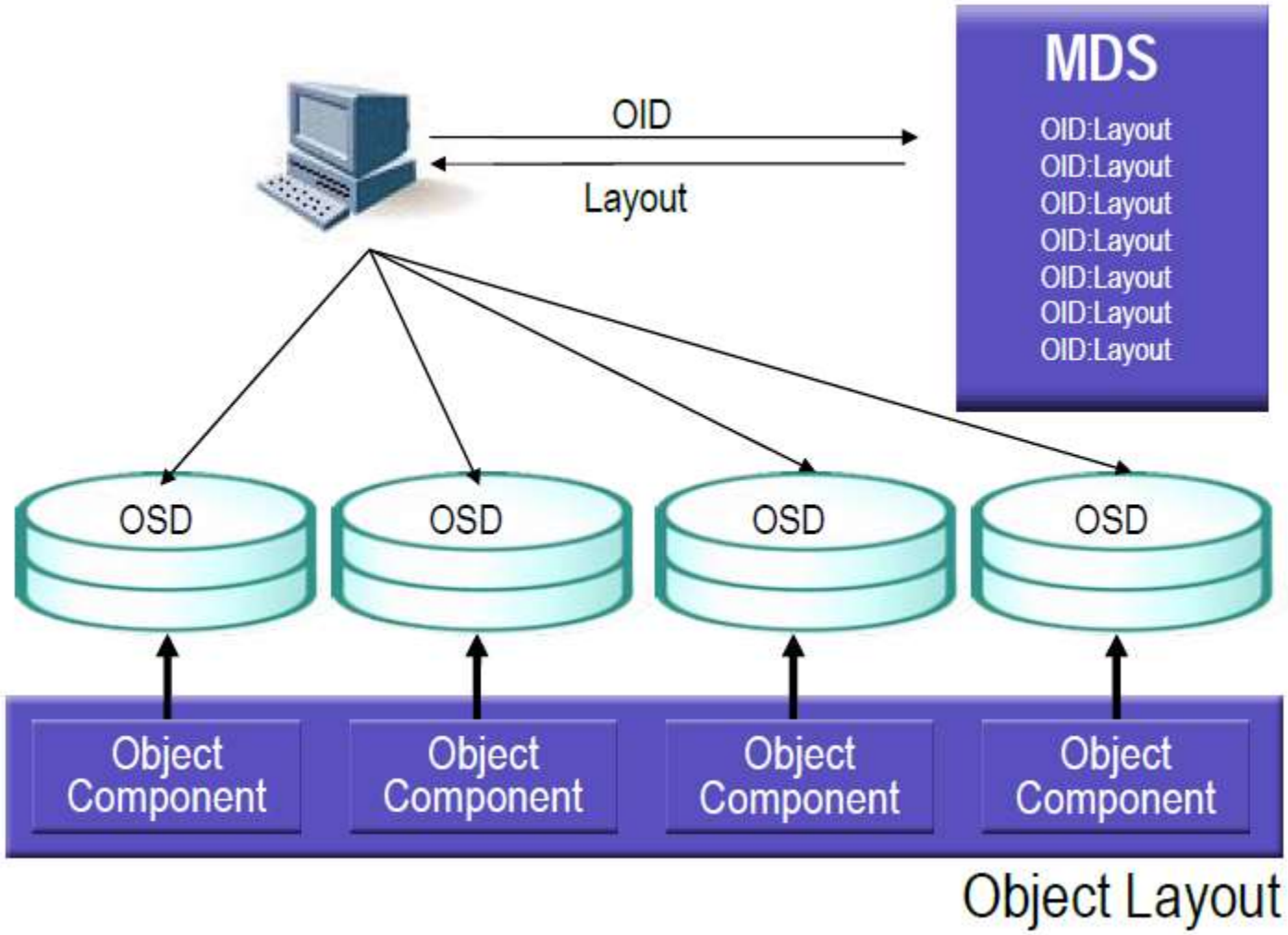
Scaling Object-based FS (3)

- Allocation of blocks to Objects handled within OSDs
  - Partitioning improves scalability
  - Compartmentalized managements improves reliability through isolated failure domains
- The File Server piece is called the MDS
  - Meta-Data Server
  - Can be clustered for scalability

Why Objects helps Scaling

- 90% of File System cycles are in the read/write path
  - Block allocation is expensive
  - Data transfer is expensive
  - OSD offloads both of these from the file server
  - Security model allows direct access from clients
- High level interfaces allow optimization
  - The more function behind an API, the less often you have to use the API to get your work done
- Higher level interfaces provide more semantics
  - User authentication and access control
  - Namespace and indexing

Object Decomposition



Object-based File Systems

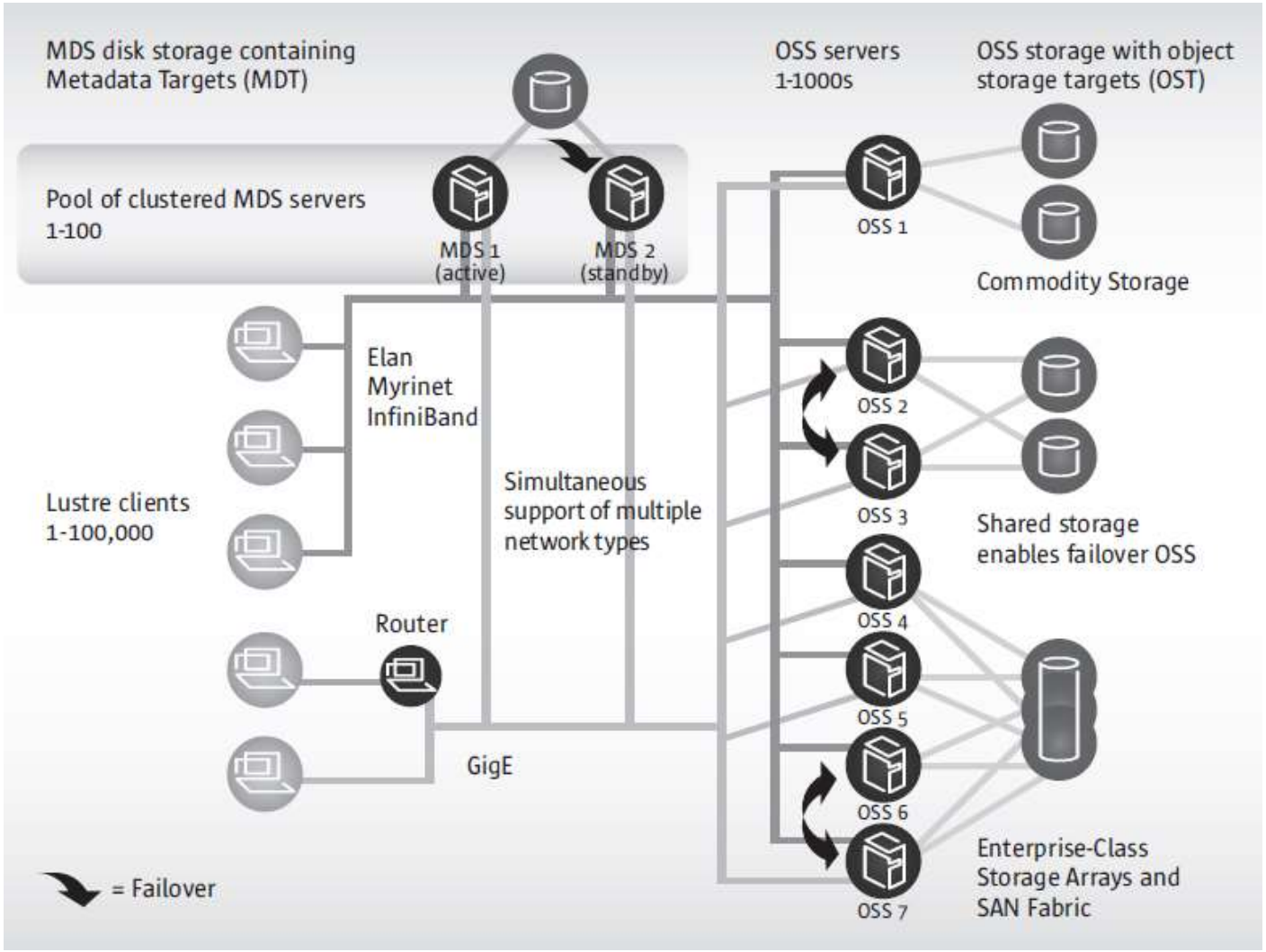
- Lustre
  - Custom OSS/OST model
  - Single metadata server
- PanFS
  - ANSI T10 OSD model
  - Multiple metadata servers
- Ceph
  - Custom OSD model
  - CRUSH metadata distribution
- pNFS
  - Out-of-band metadata service for NFSv4.1
  - T10 Objects, Files, Blocks as data services

- These systems scale
  - 1000's of disks (i.e., PB's)
  - 1000's of clients
  - 100's GB/sec
  - All in one file system

Lustre (1)

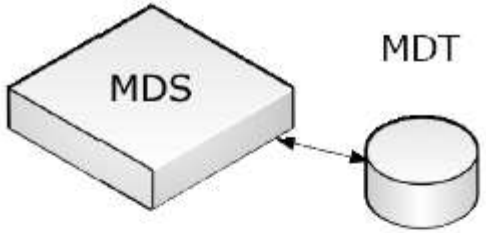
- Supercomputing focus emphasizing
  - High I/O throughput
  - Scalability in the Pbytes of data and billions of files
- OSDs called OSTs (Object Storage Targets)
- Only RAID-0 supported across Objects
  - Redundancy inside OSTs
- Runs over many transports
  - IP over ethernet
  - Infiniband
- OSD and MDS are Linux based & Client Software supports Linux
  - Other platforms under consideration
- Used in Telecom/Supercomputing Center/Aerospace/National Lab

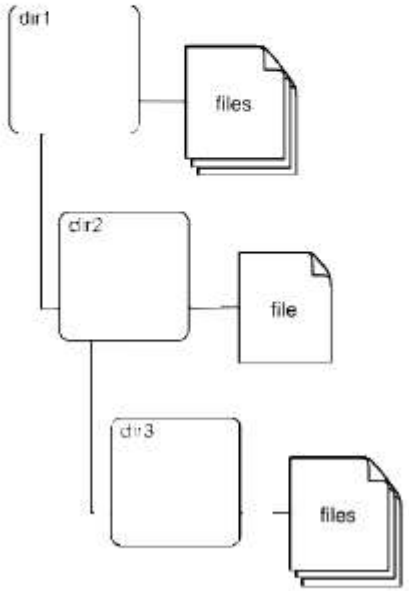
Lustre (2) Architecture



Lustre (3) Architecture-MDS

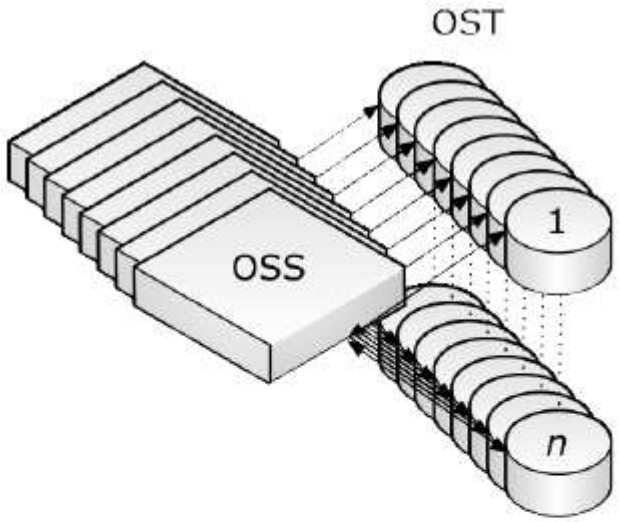
- Metadata Server (MDS)
  - Node(s) that manage namespace, file creation and layout, and locking. Directory operations
    - File open/close
    - File status
    - File creation
    - Map of file object location
  - Relatively expensive serial atomic transactions to maintain consistency
- •**Metadata Target (MDT)**
  - Block device that stores metadata

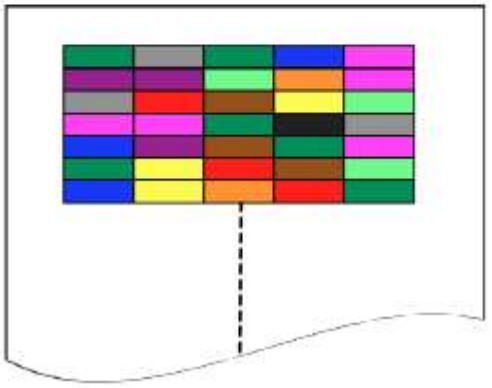




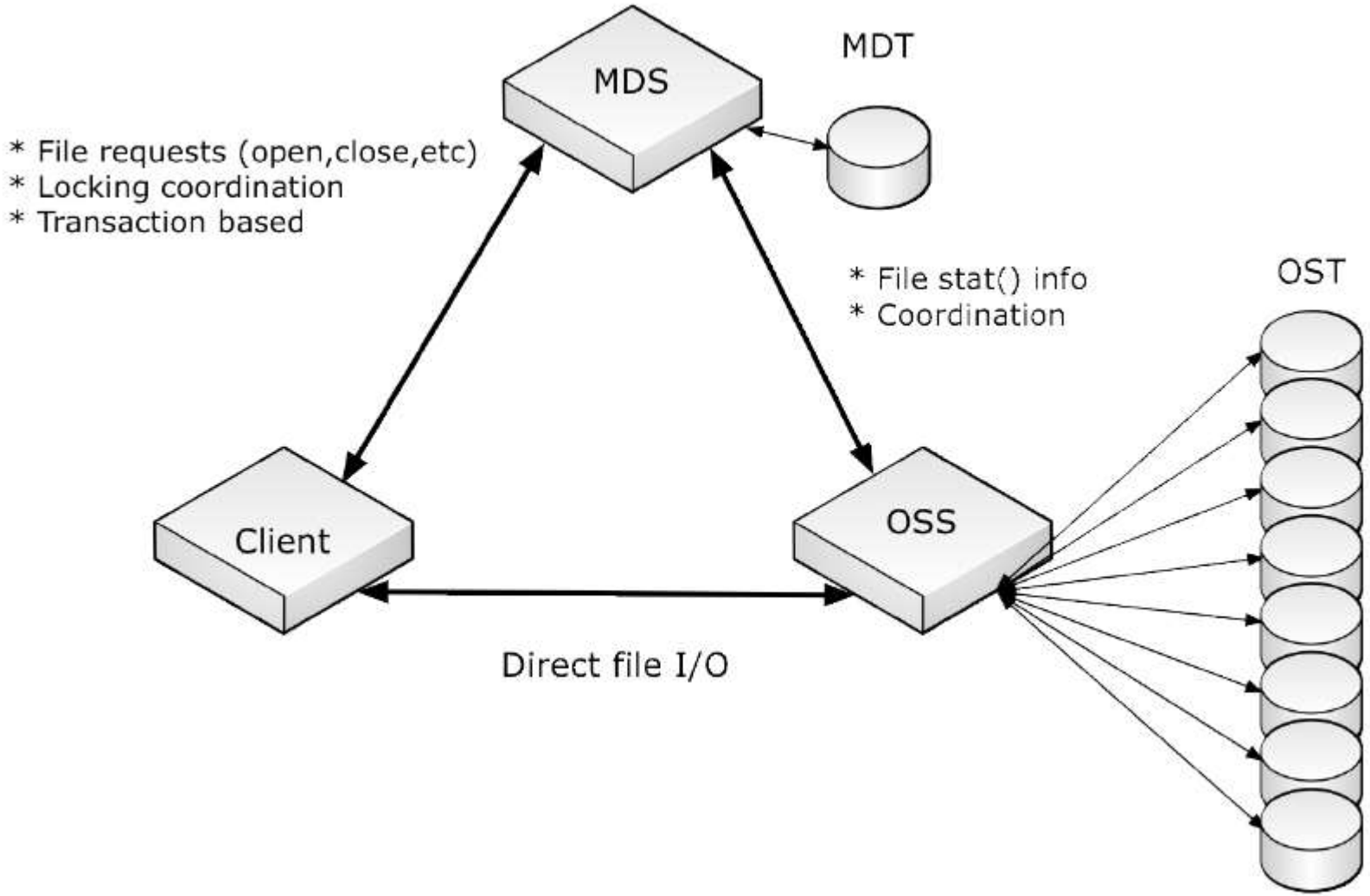
Lustre (3) Architecture-OSS

- **Object Storage Server (OSS)**
  - Multiple nodes that manage network requests for file objects on disk.
- **Object Storage Target (OST)**
  - Block device that stores file objects



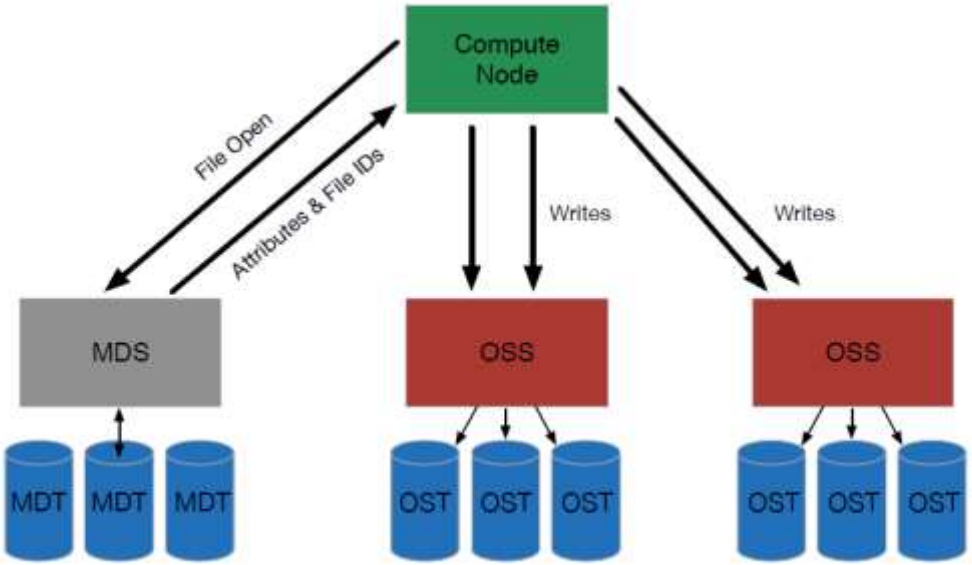


Lustre (4) Simplest Lustre File System



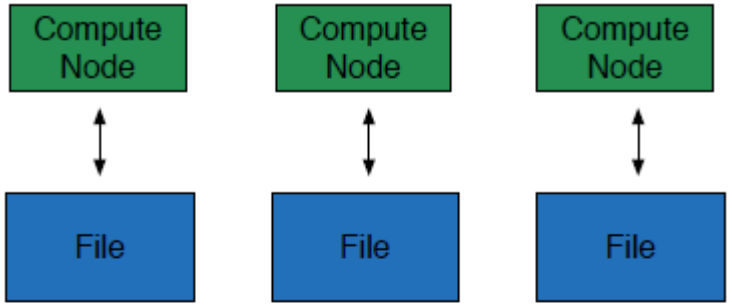
Lustre (5) File Operation

- When a compute node needs to create or access a file, it requests the associated storage locations from the MDS and the associated MDT.
- I/O operations then occur directly with the OSSs and OSTs associated with the file bypassing the MDS.
- For read operations, file data flows from the OSTs to the compute node.

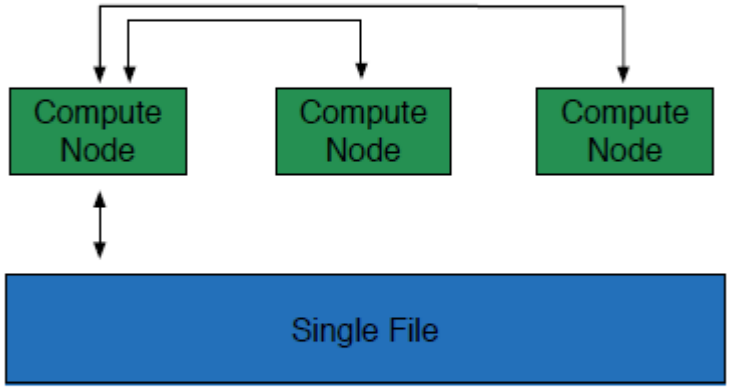


Lustre (6) File I/Os

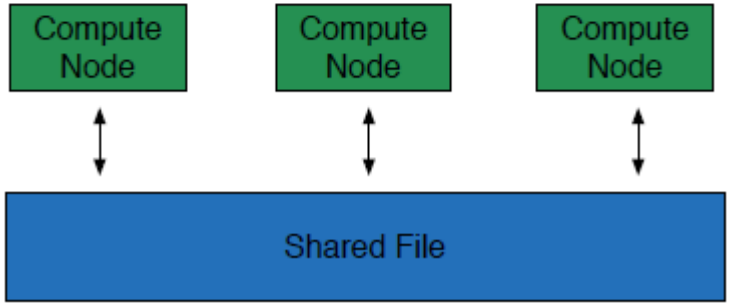
- Single stream



- Single stream through a master

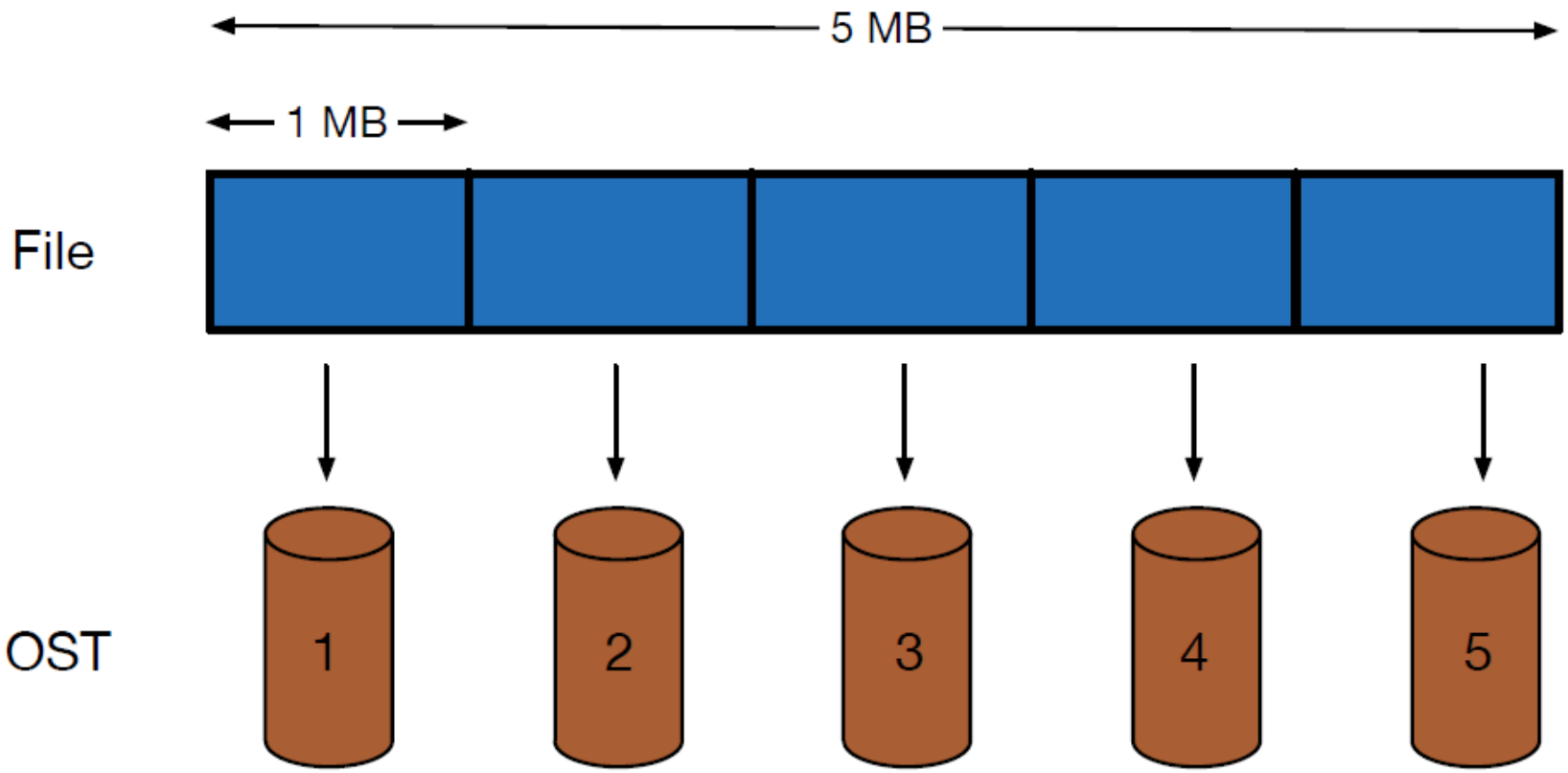


- Parallel



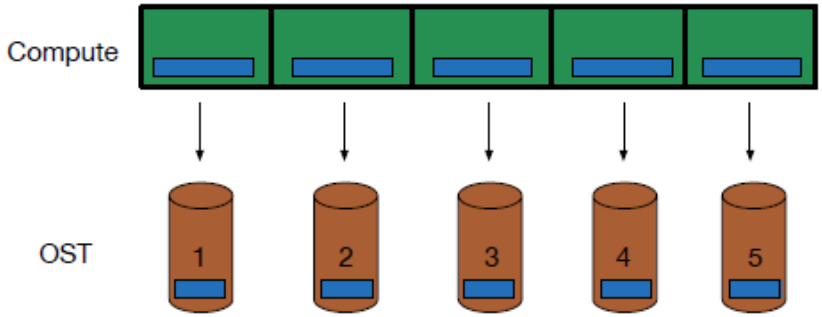
Lustre (7) File Striping

- A file is split into segments and consecutive segments are stored on different physical storage devices (OSTs).

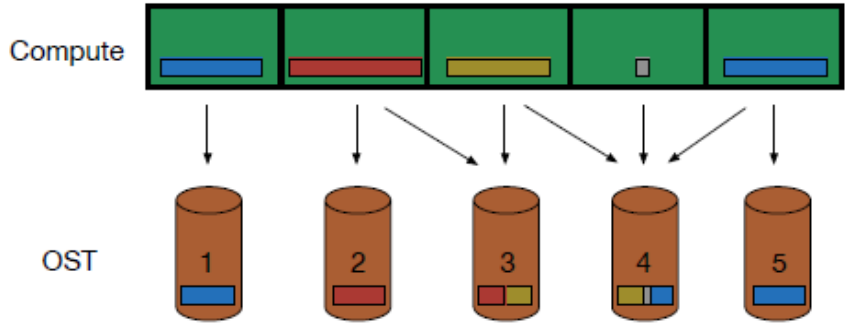


Lustre (8) Aligned and Unaligned Stripes

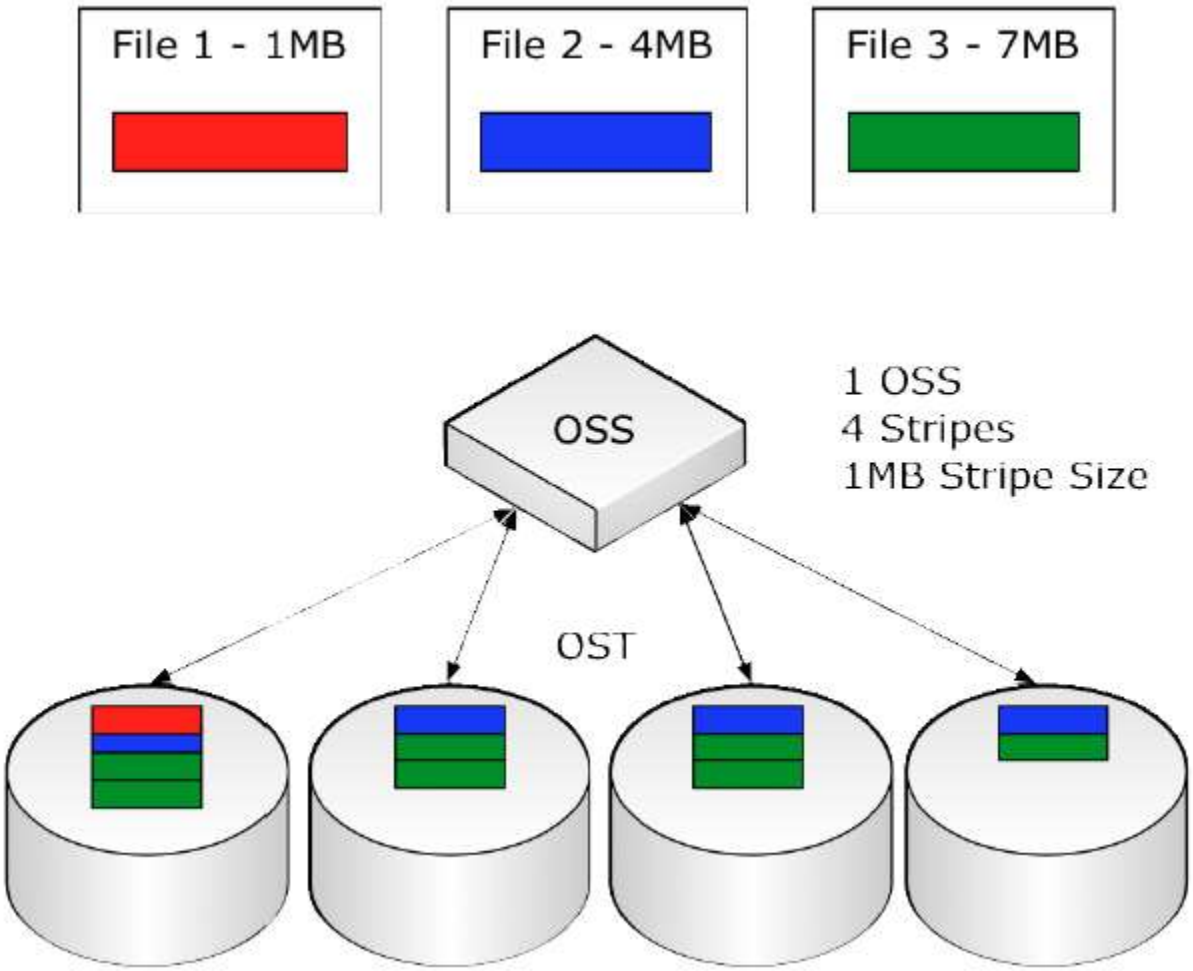
- Aligned stripes is where each segment fits fully onto a single OST. Processes accessing the file do so at corresponding stripe boundaries.



- Unaligned stripes means some file segments are split across OSTs.



Lustre (9) Striping Example



Lustre (10) Advantages/Disadvantages

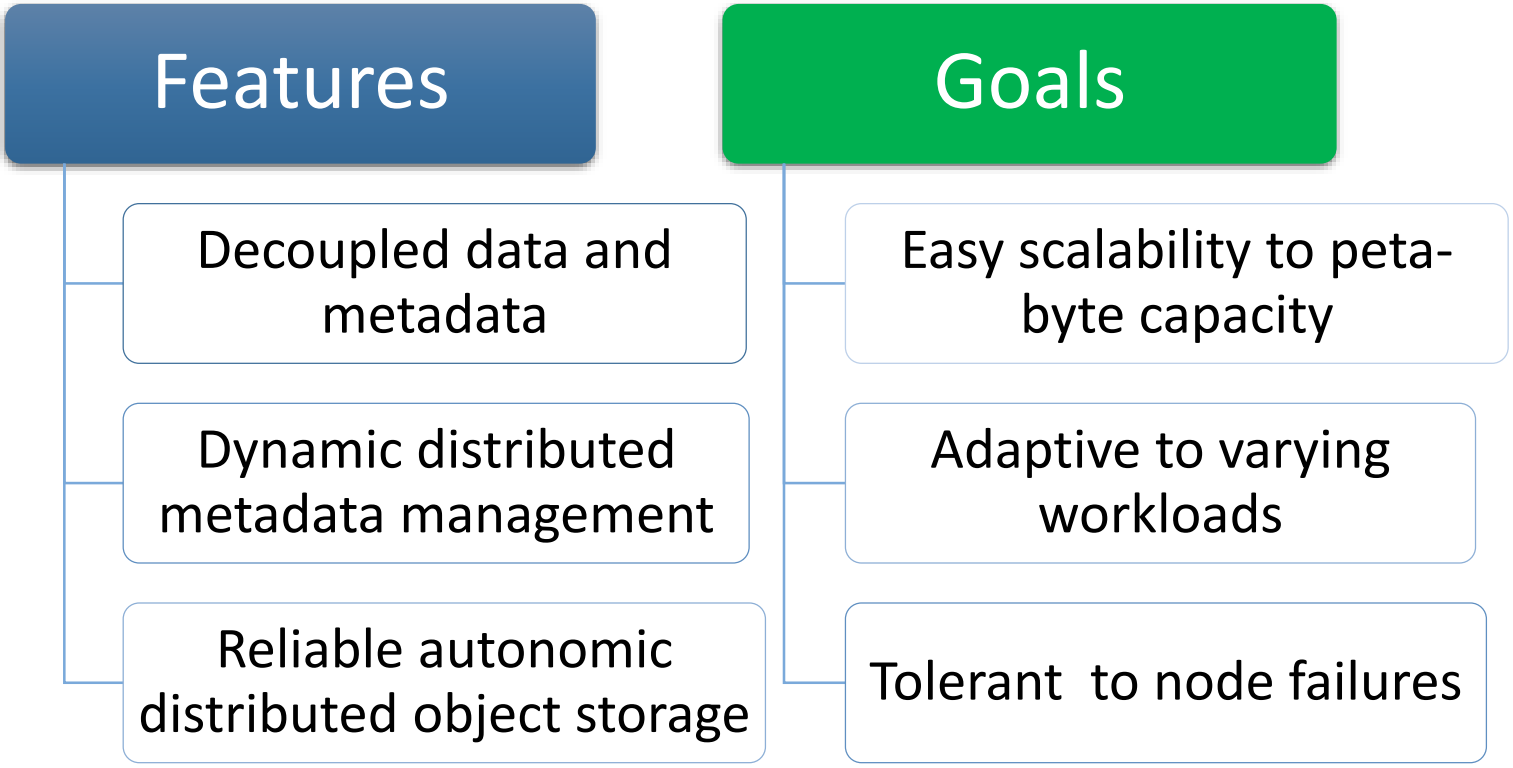
- Striping will not benefit *ALL* applications

Advantages	Disadvantages
Bandwidth – file objects are striped across OSTs, so bandwidth is the aggregate I/O rate	**User Overhead** – Time and thought required to understand your I/O patterns and create stripe layout for directories and files
File Size – file objects striped across OST can have a total size larger than available space on any single OST	**System Overhead** – Additional stripes means more OST lookups to determine the size of the file (more time)

Ceph (1)

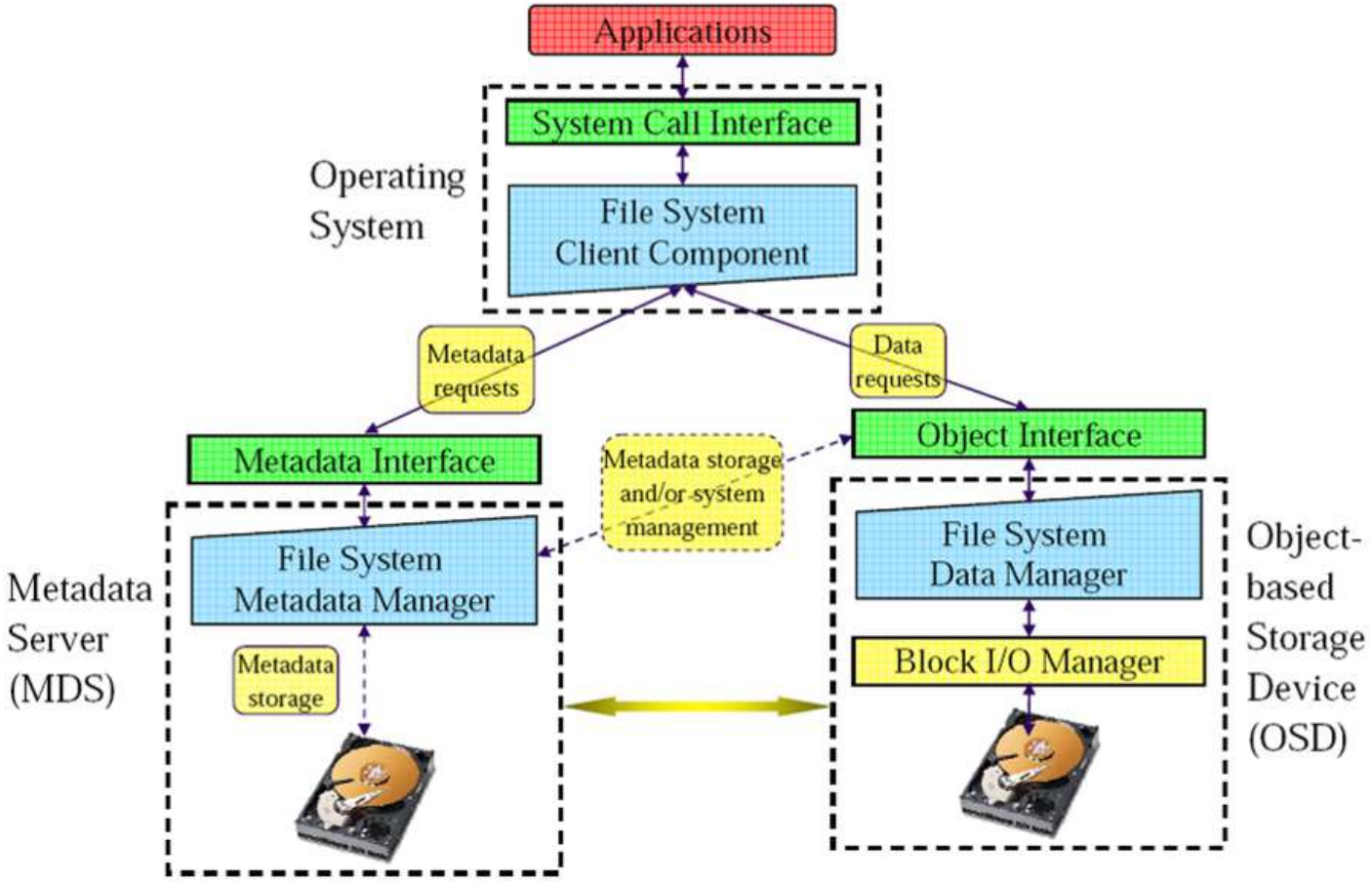
- What is Ceph?

Ceph is a distributed file system that provides excellent performance, scalability and reliability.

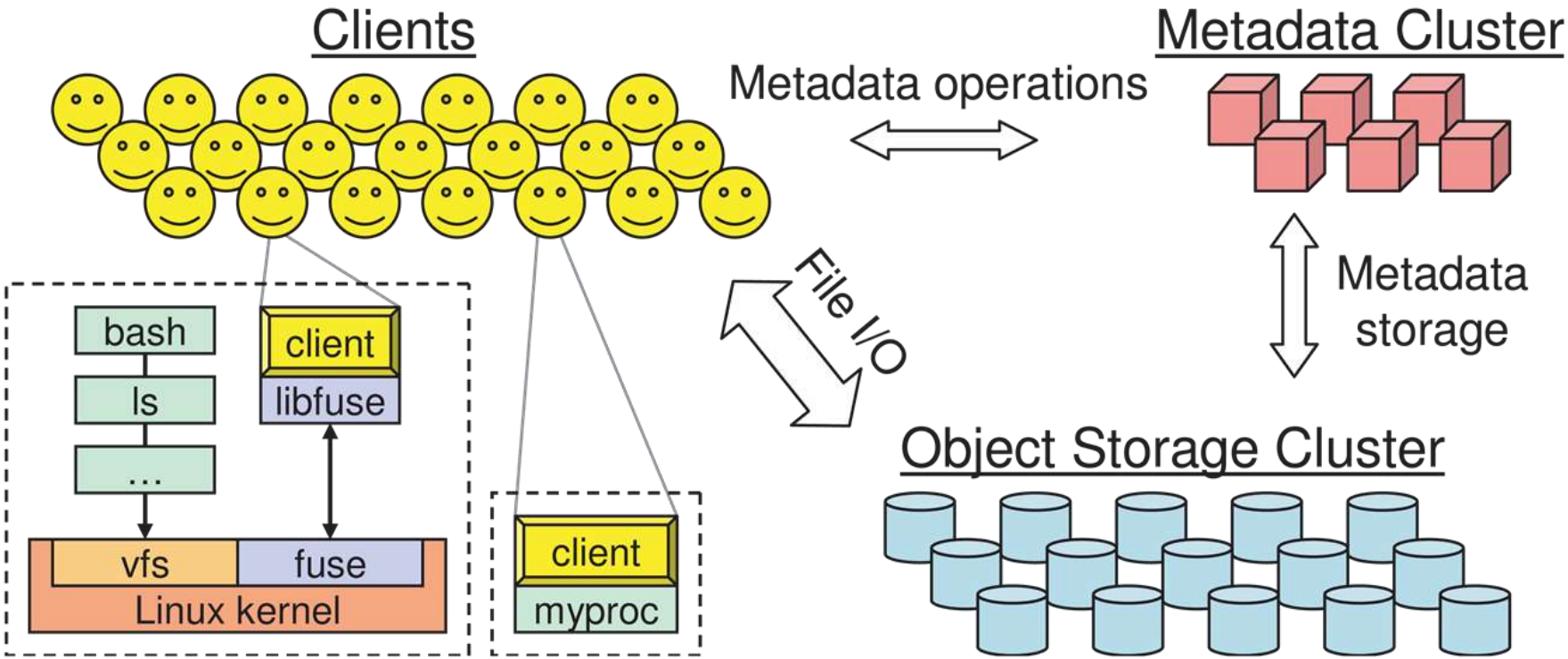


Ceph (2) – Architecture

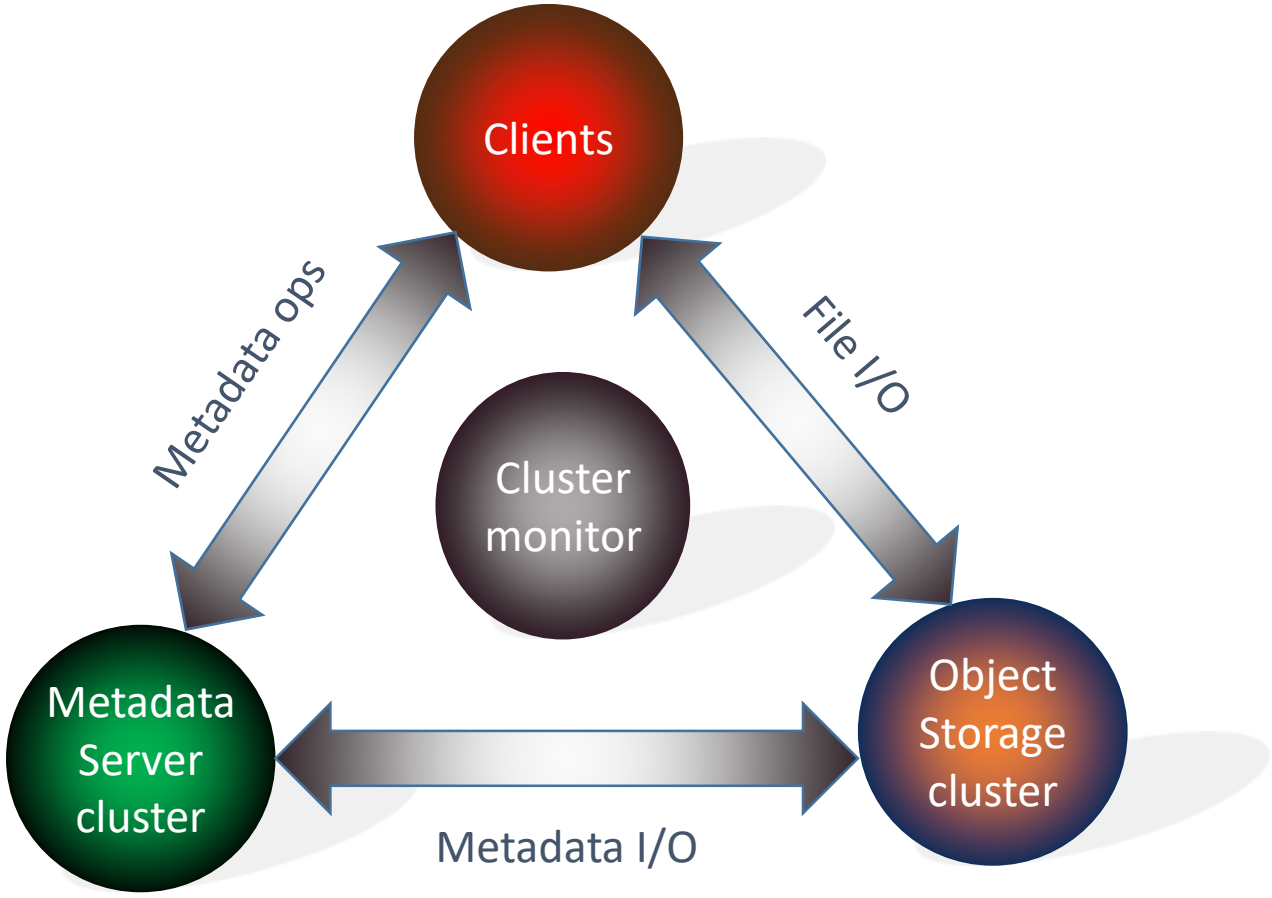
- Decoupled Data and Metadata



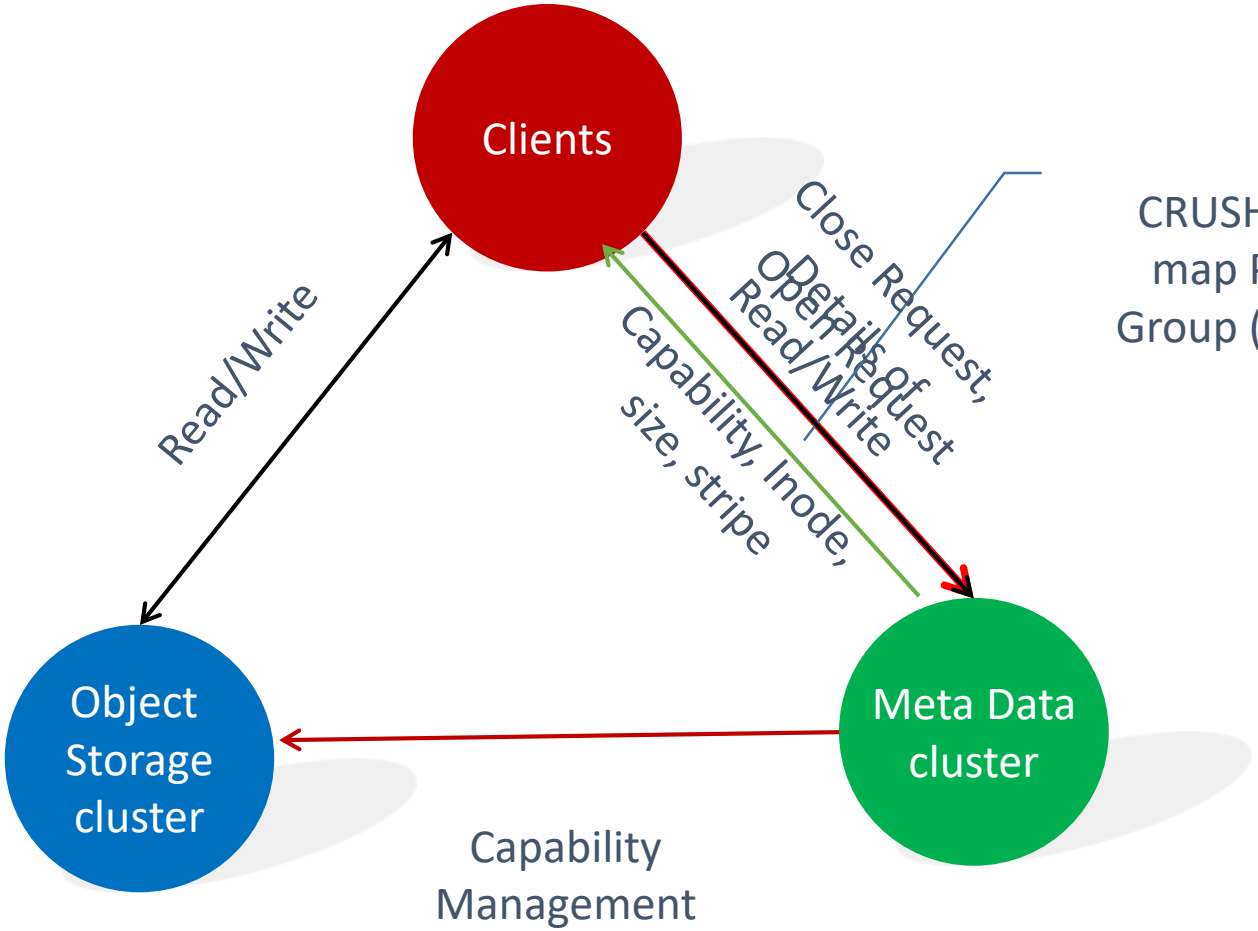
Ceph (3) – Architecture



Ceph (4) – Components

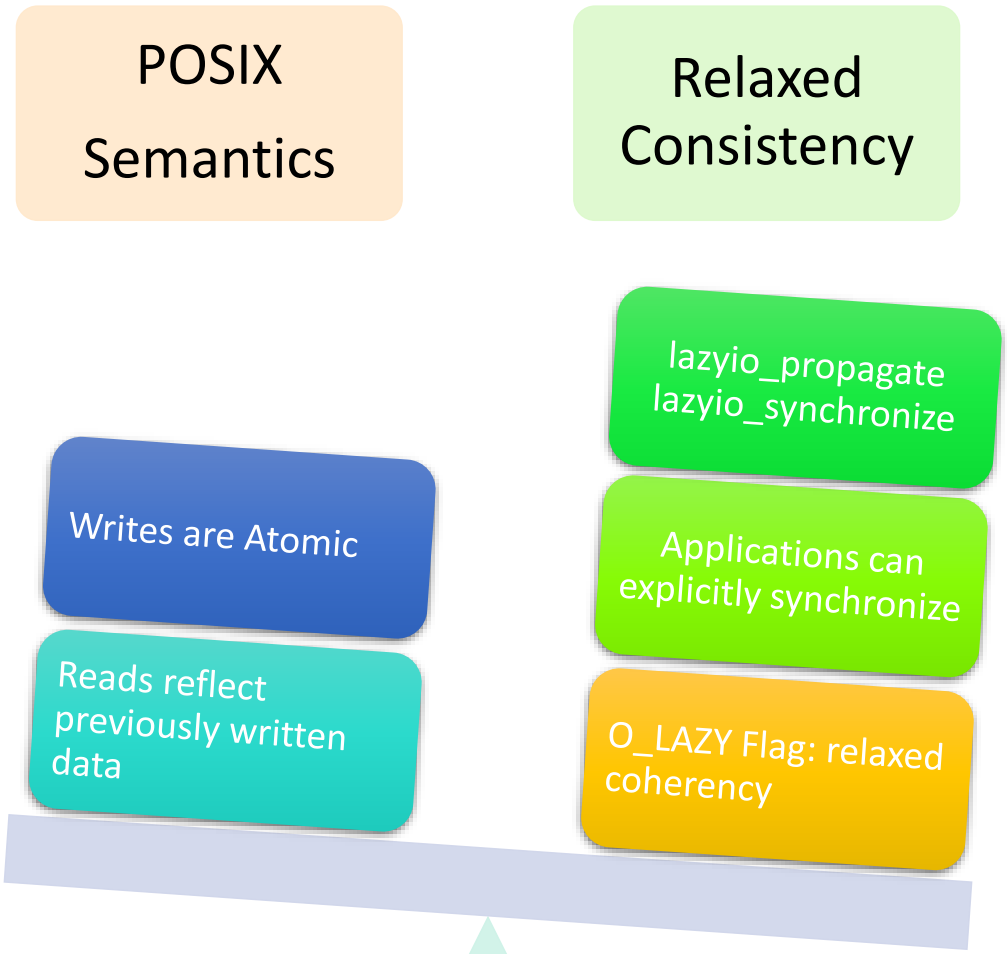


Ceph (5) - Components

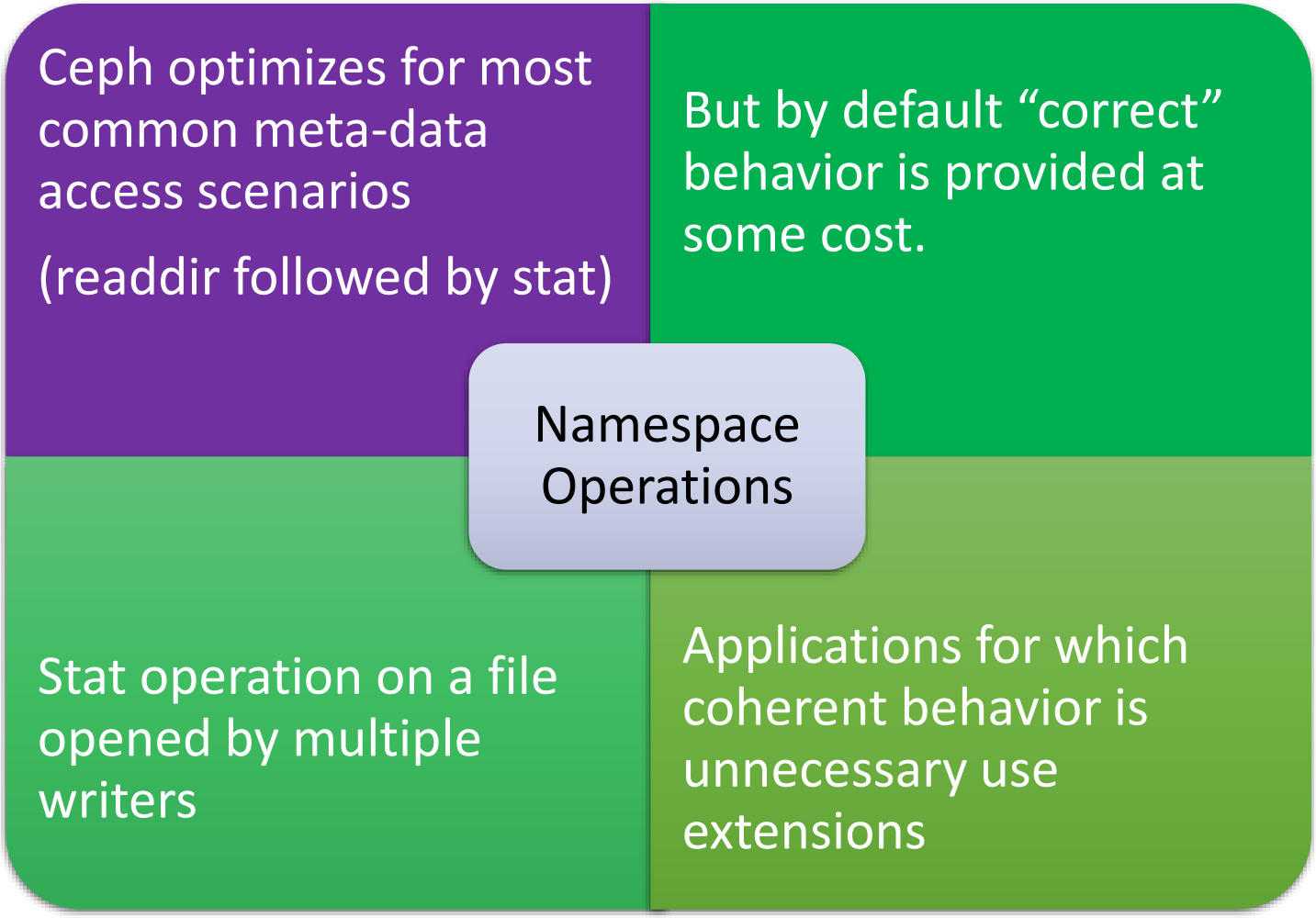


Ceph (6) – Components

- Client Synchronization
  - Synchronous I/O. performance killer
  - Solution: HPC extensions to POSIX
    - Default: Consistency / correctness
    - Optionally relax
  - Extensions for both data and metadata

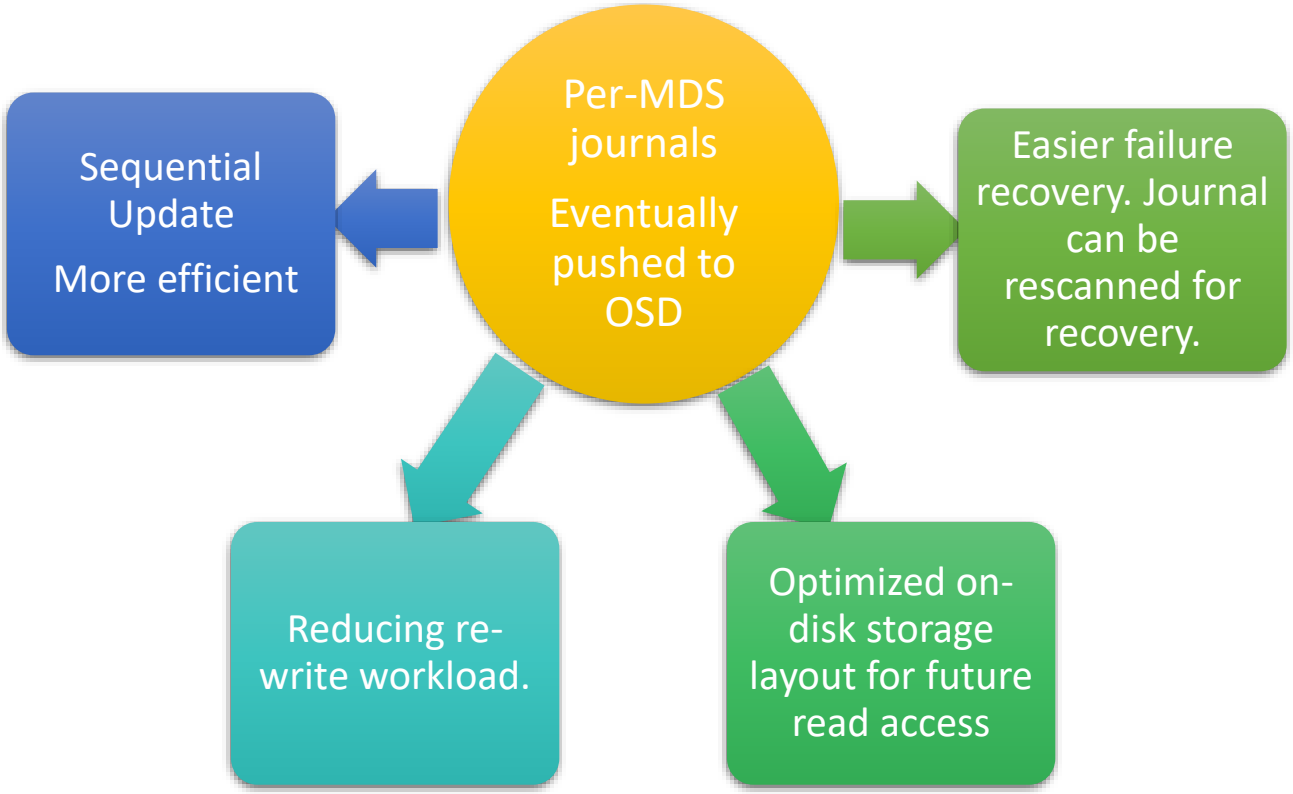


Ceph (7) – Namespace Operations



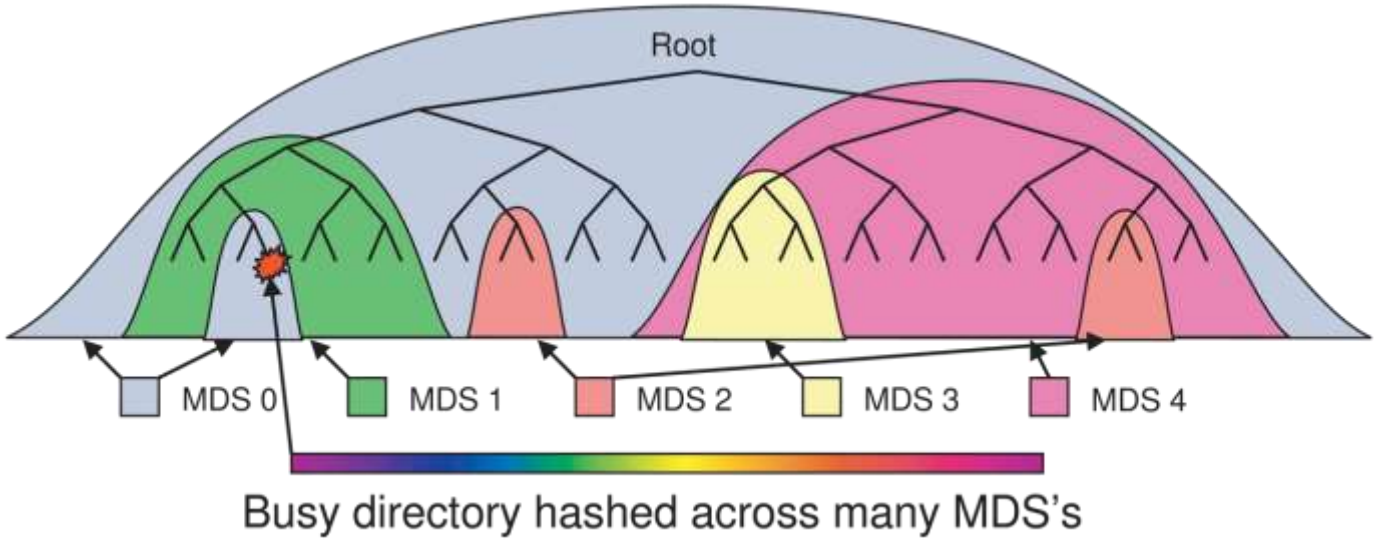
Ceph (8) – Metadata

- Metadata Storage
  - Advantages



Ceph (9) – Metadata

- Dynamic Sub-tree Partitioning

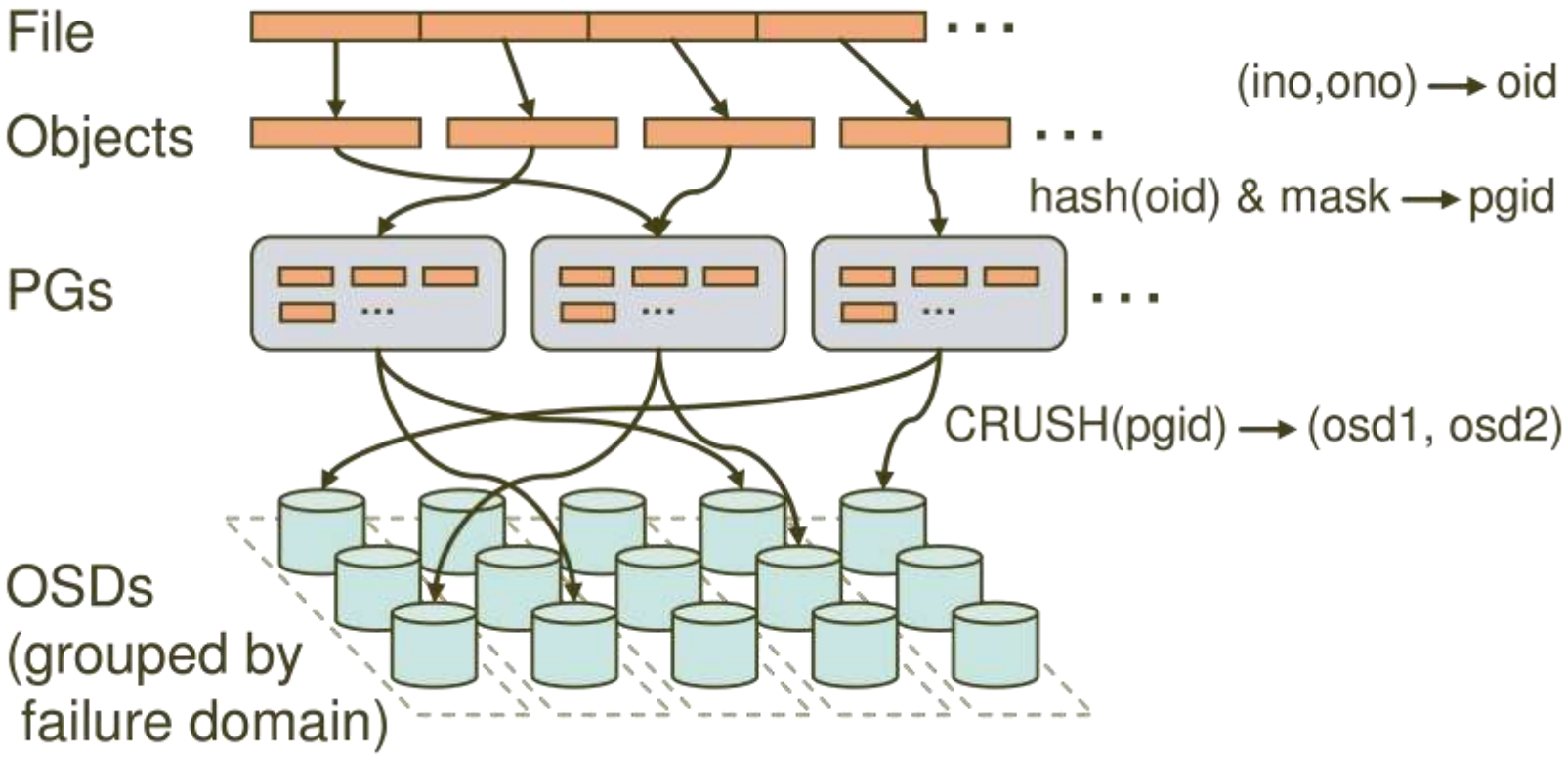


- Adaptively distribute cached metadata hierarchically across a set of nodes.
- Migration preserves locality.
- MDS measures popularity of metadata.

Ceph (10) – Metadata

- Traffic Control for metadata access
  - Challenge
    - Partitioning can balance workload but can't deal with hot spots or flash crowds
  - Ceph Solution
    - ✓ Heavily read directories are selectively replicated across multiple nodes to distribute load
    - ✓ Directories that are extra large or experiencing heavy write workload have their contents hashed by file name across the cluster

Ceph (11) – Distributed Object Storage

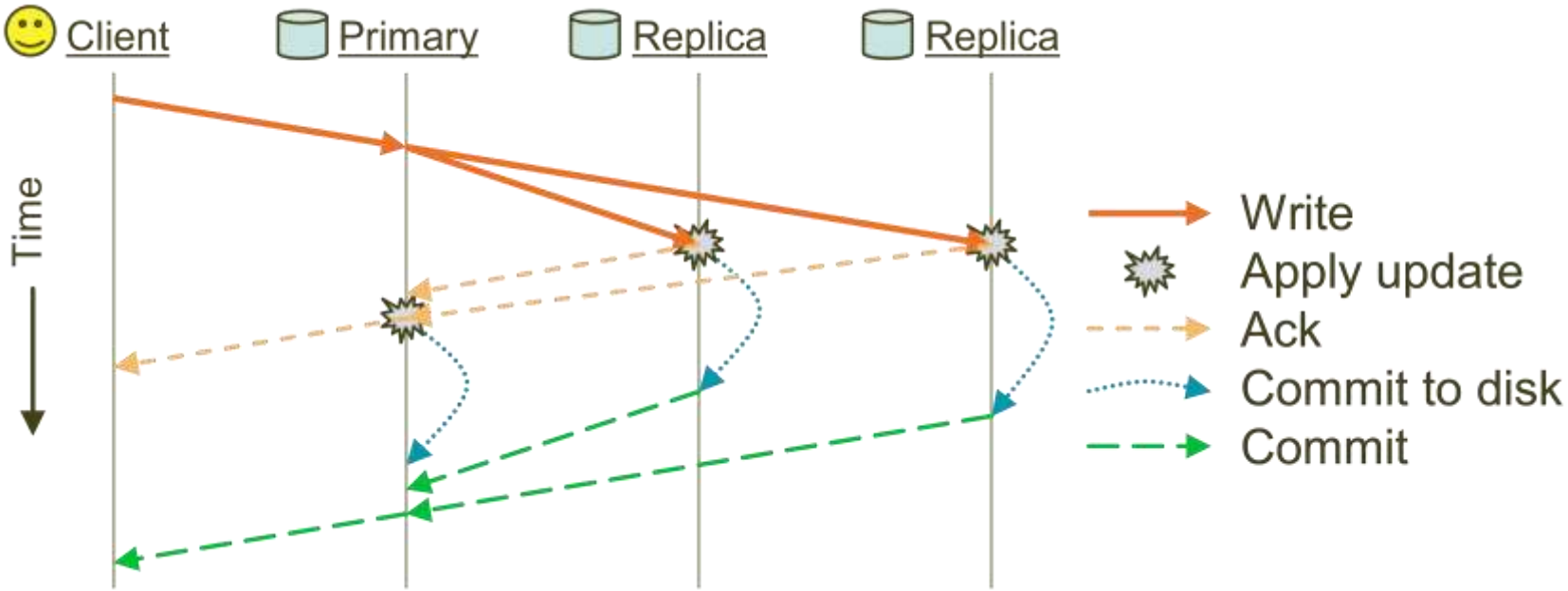


Ceph (11) – CRUSH

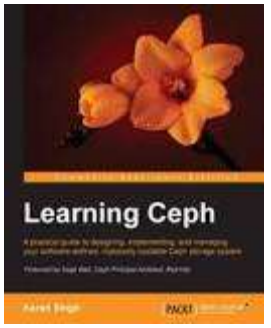
- CRUSH(x) → (osd_{n1}, osd_{n2}, osd_{n3})
  - Inputs
    - x is the placement group
    - Hierarchical cluster map
    - Placement rules
  - Outputs a list of OSDs
- Advantages
  - Anyone can calculate object location
  - Cluster map infrequently updated

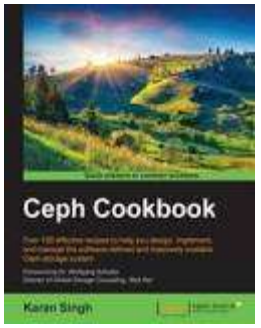
Ceph (12) – Replication

- Objects are replicated on OSDs within same PG
  - Client is oblivious to replication



Ceph (13) – Conclusion





- Strengths:
  - Easy scalability to peta-byte capacity
  - High performance for varying work loads
  - Strong reliability
- Weaknesses:
  - MDS and OSD Implemented in user-space
  - The primary replicas may become bottleneck to heavy write operation
  - N-way replication lacks storage efficiency
- References
  - Ceph: A Scalable, High Performance Distributed File System. In Proc. of OSDI'06

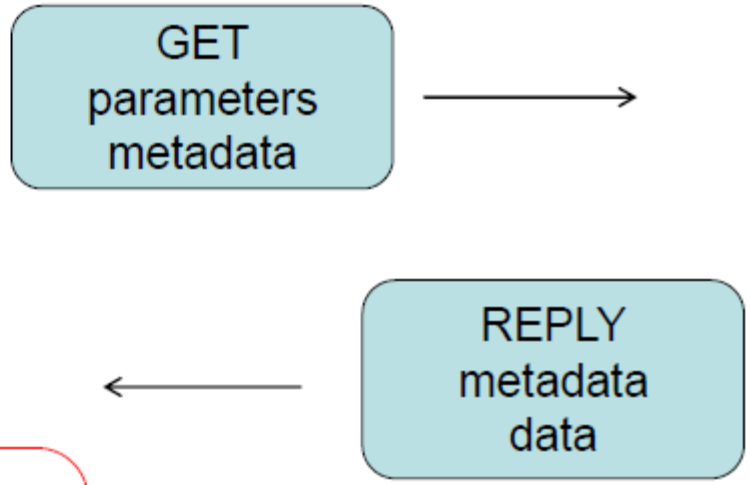
4 Object-based Storage in Cloud

Web Object Features

- RESTful API (i.e., web-based)
- Security/Authentication tied to Billing
- Metadata capabilities
- Highly available
- Loosely consistent
- Data Storage
  - Blobs
  - Tables
  - Queues
- Other related APIs (compute, search, etc.)
  - Storage API is relatively simple in comparison

Simple HTTP example

```
% telnet www.google.com 80
GET /index.html HTTP/1.0
(blank line)
HTTP/1.0 200 OK
Date: Wed, 13 Feb 2013 07:24:07 GMT
Content-Type: text/html; charset=ISO-8859-1
<html>
<head><title>Google</title></head>
<body><img src= /images/srpr/logo3w.png>
<form><input type=text name=q>
<input type=submit value="Google Search" name="search">
<input type=submit value="I'm Feeling Lucky" name="lucky">
</form></body></html>
```



HTTP and objects

- Request specifies method and object:
  - Operation: GET, POST, PUT, HEAD, COPY
  - Object ID (/index.html)
- Parameters use MIME format borrowed from email
  - Content-type: utf8;
  - Set-Cookie: tracking=1234567;
- Add a data payload
  - Optional
  - Separated from parameters with a blank line (like email)
- Response has identical structure
  - Status line, key-value parameters, optional data payload

This is a method call on an object

These are parameters

This is data

OpenStack REST API for Storage

- GET v1/account HTTP/1.1
  - Login to your account
- HEAD v1/account HTTP/1.1
  - List account metadata
- PUT v1/account/container HTTP/1.1
  - Create container
- PUT v1/account/container/object HTTP/1.1
  - Create object
- GET v1/account/container/object HTTP/1.1
  - Read object
- HEAD v1/account/container/object HTTP/1.1
  - Read object metadata

Create an object

```
PUT /v1/<account>/<container>/<object> HTTP/1.1
Host: storage.swiftdrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
ETag: 8a964ee2a5e88be344f36c22562a6486
Content-Length: 512000
X-Delete-At: 1339429105
Content-Disposition: attachment; filename=platmap.mp4
Content-Type: video/mp4
Content-Encoding: gzip
X-Object-Meta-PIN: 1234
[ ...object content... ]
```

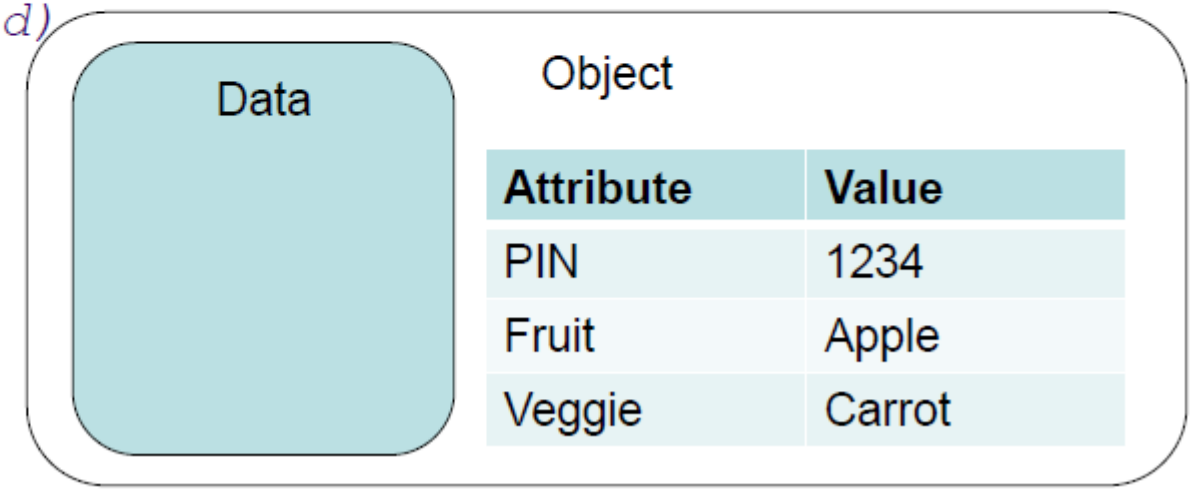
MD5 checksum

Mon Jun 11 08:38:25 PDT 2012

User defined metadata

Update metadata

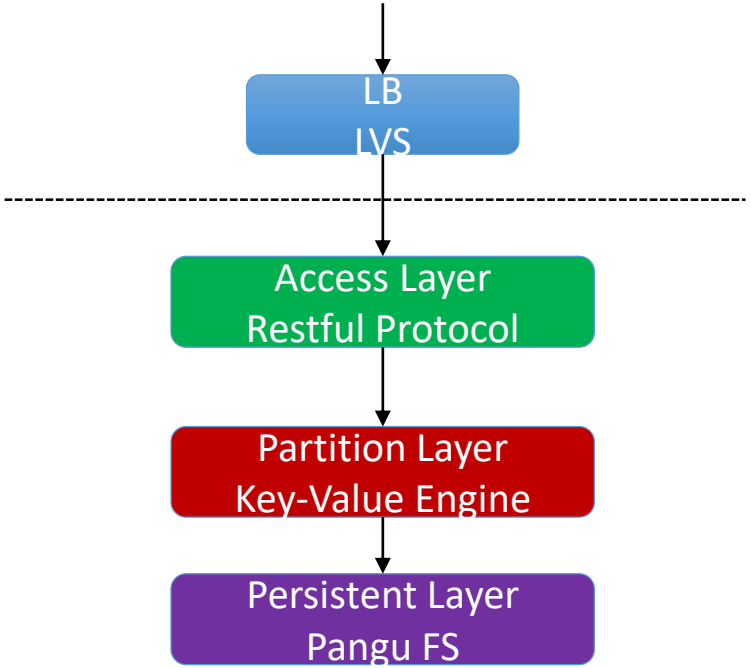
```
POST /v1/<account>/<container>/<object> HTTP/1.1
Host: storage.swiftdrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
X-Object-Meta-Fruit: Apple
X-Object-Meta-Veggie: Carrot

(no data payload)
```



Ali OSS (1)

阿里云 aliyun.com

- Access URL: http://<bucket>.oss-cn-beijing.aliyuncs.com/<object>



Load Balancing

Protocol Manager &
Access Control

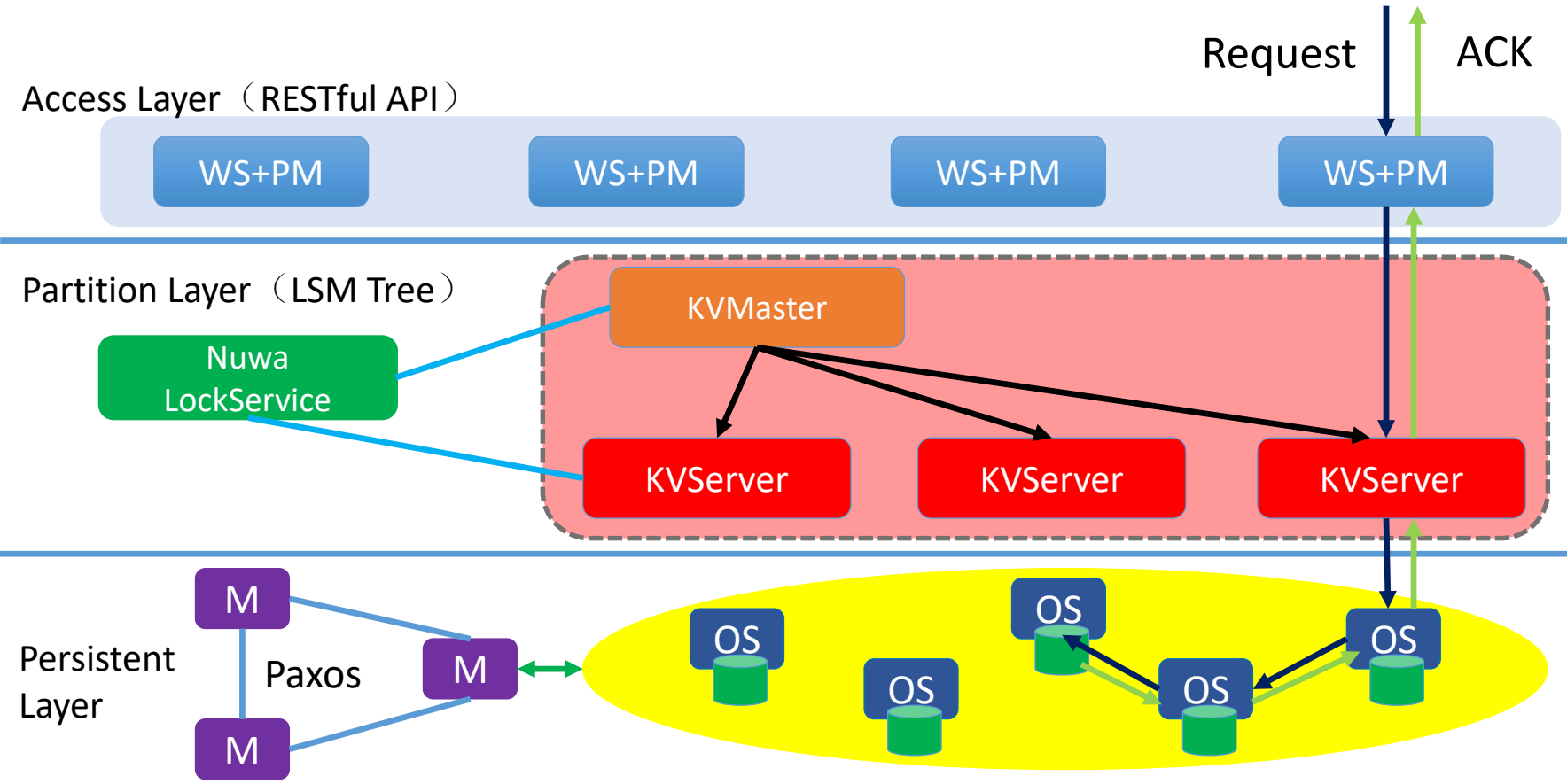
Partition & Index

Persistent, Redundancy
& Fault-Tolerance

Ali OSS (2) Architecture

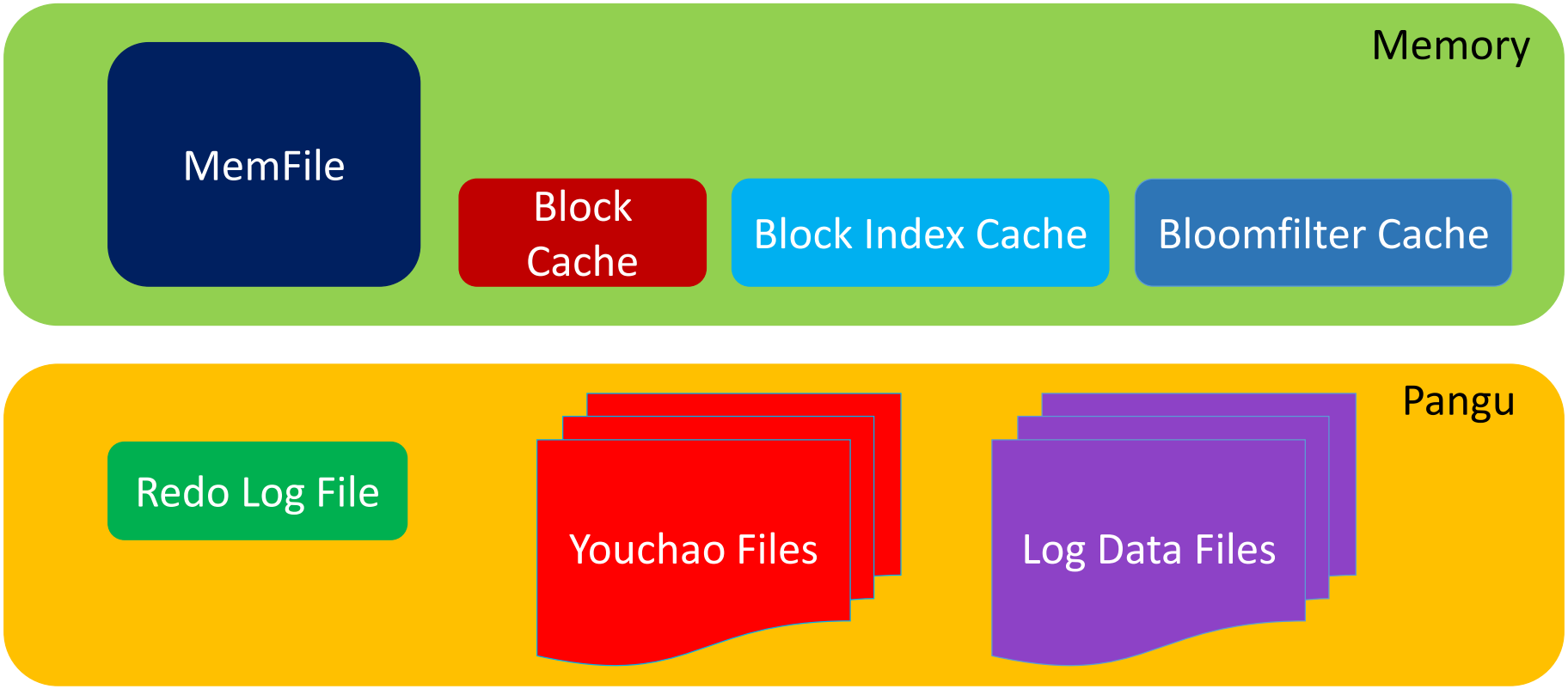
阿里云 aliyun.com

- WS: Web Server PM: Protocol Manager



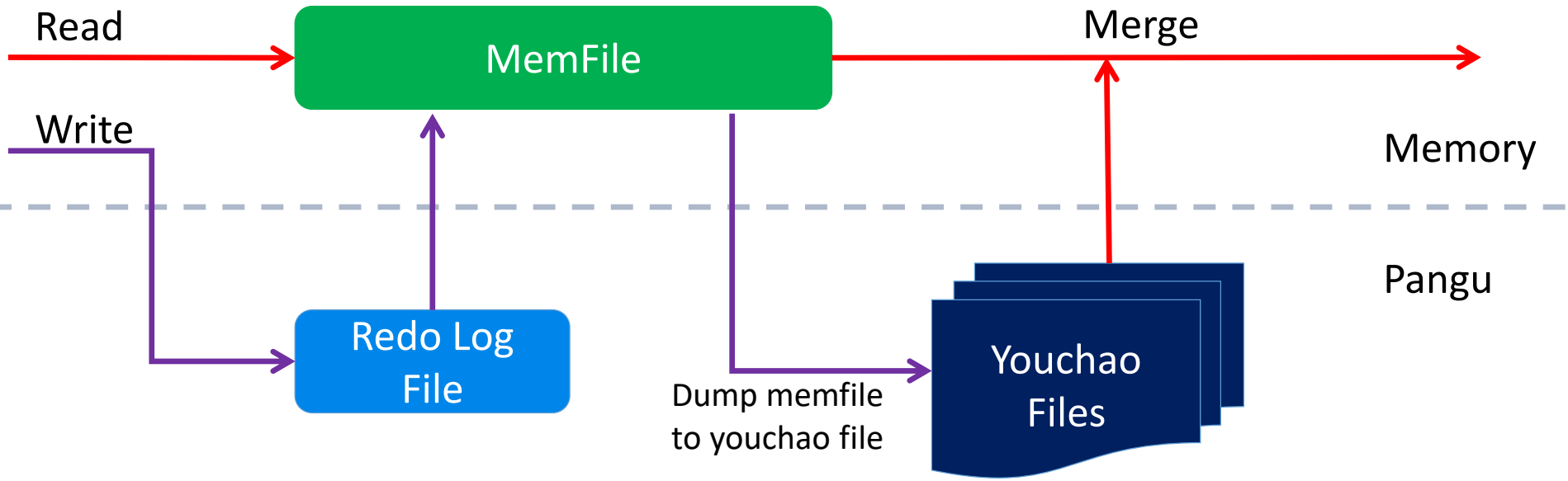
Ali OSS (3) Partition Layer

- Append/Dump/Merge



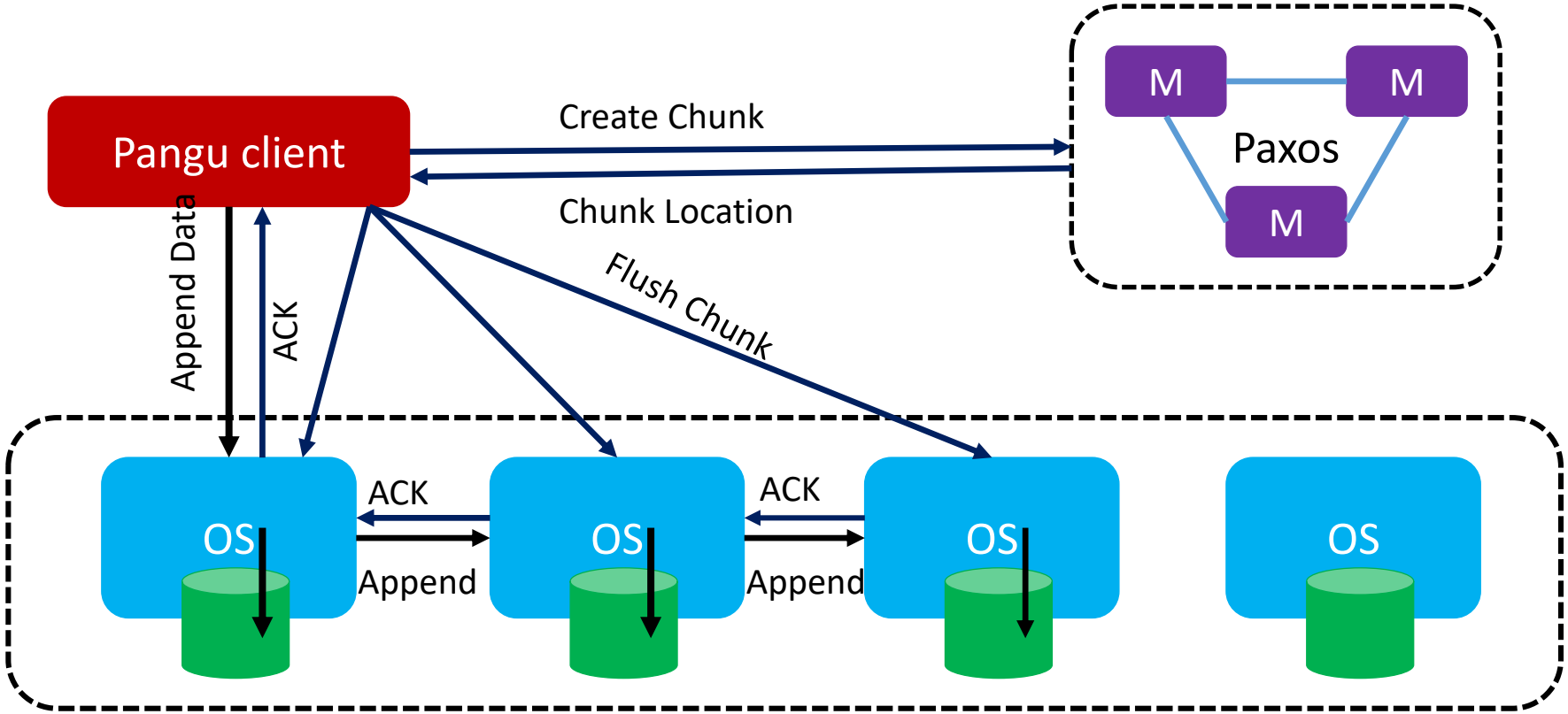
Ali OSS (4) Partition Layer

- Read/Write Process



Ali OSS (5) Persistent Layer

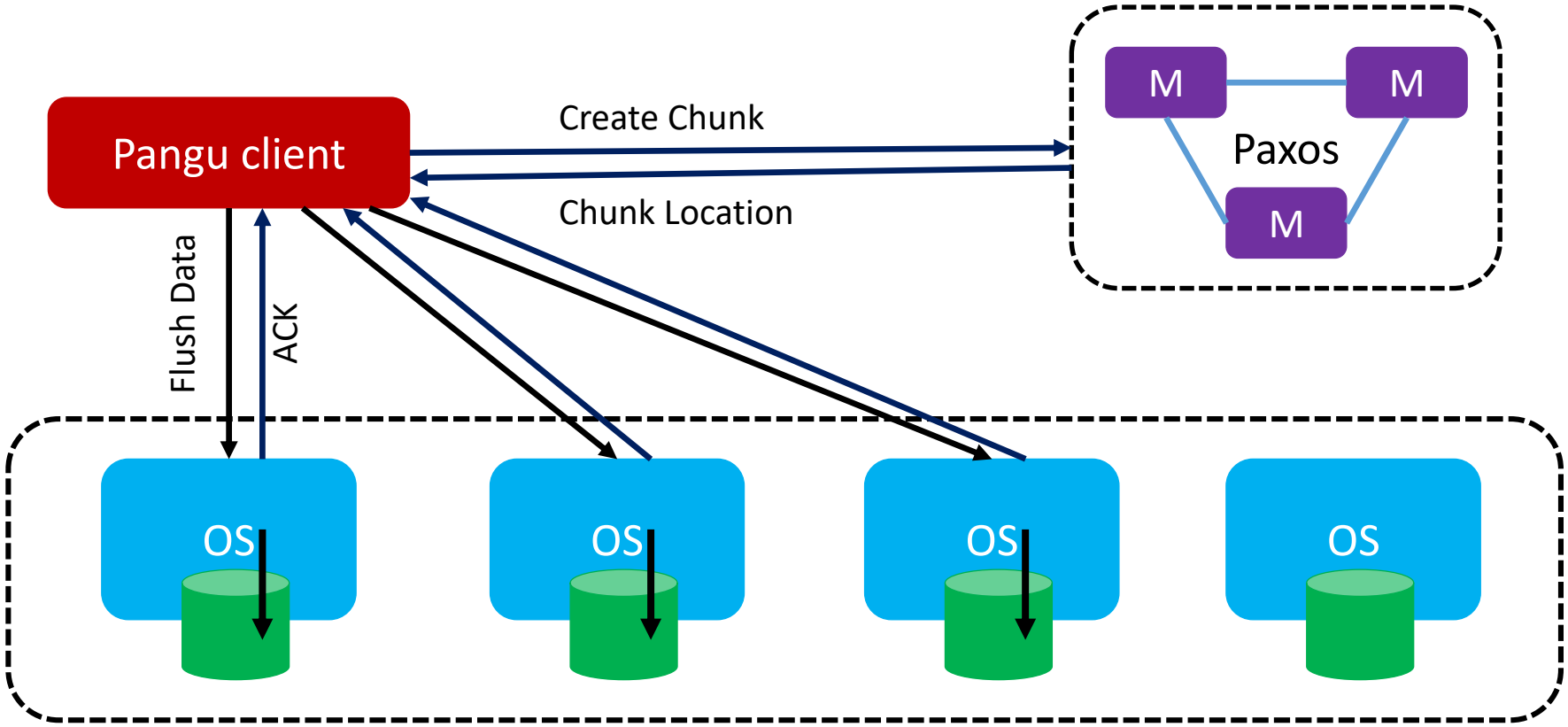
- Write Pangu Normal File



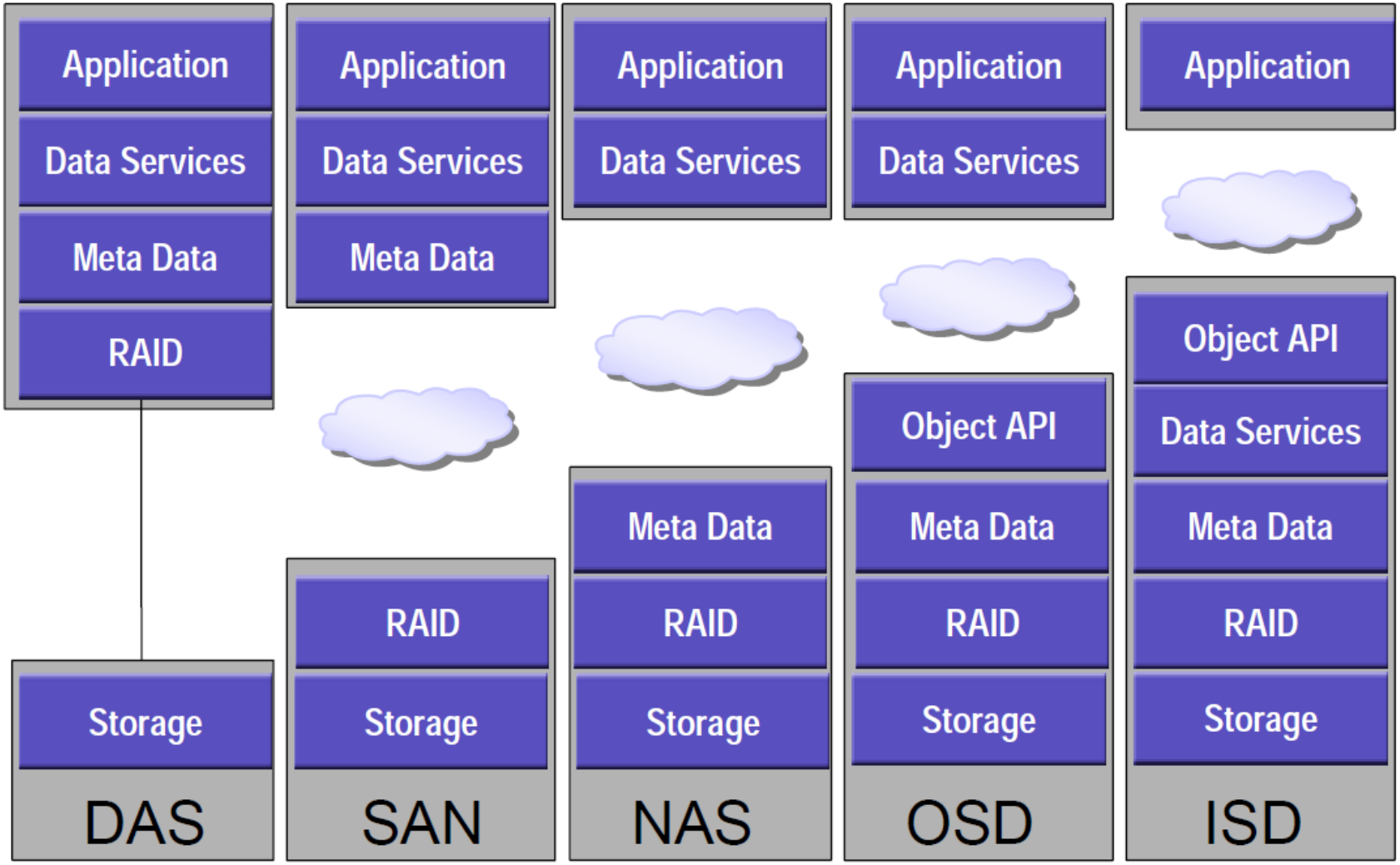
Ali OSS (6) Persistent Layer

阿里云 aliyun.com

- Write Pangu Log File



The Evolution of Data Storage



Thank you!