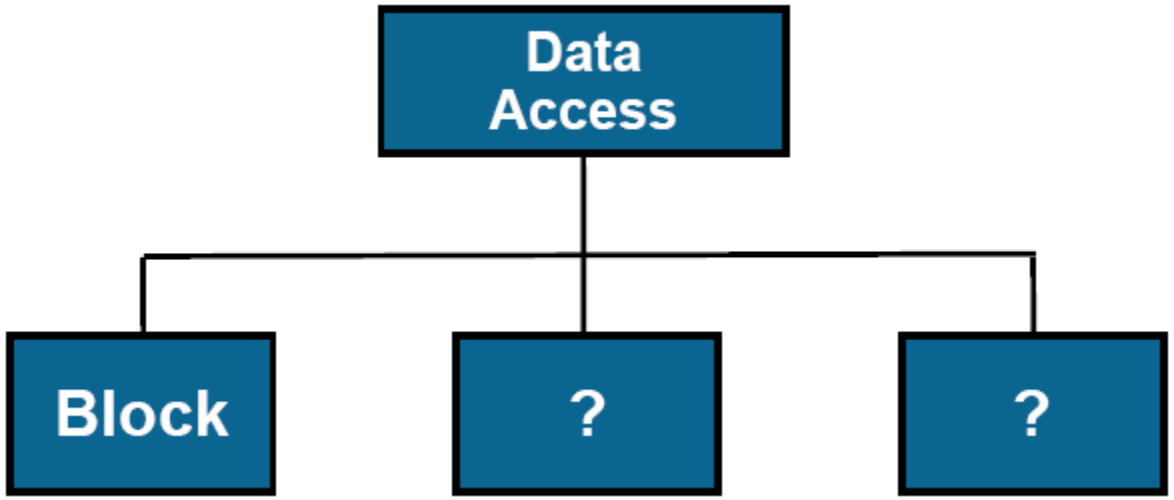
对象存储
Object-Based Storage

History of Object Storage

The Data Access Taxonomy

- The Block Paradigm

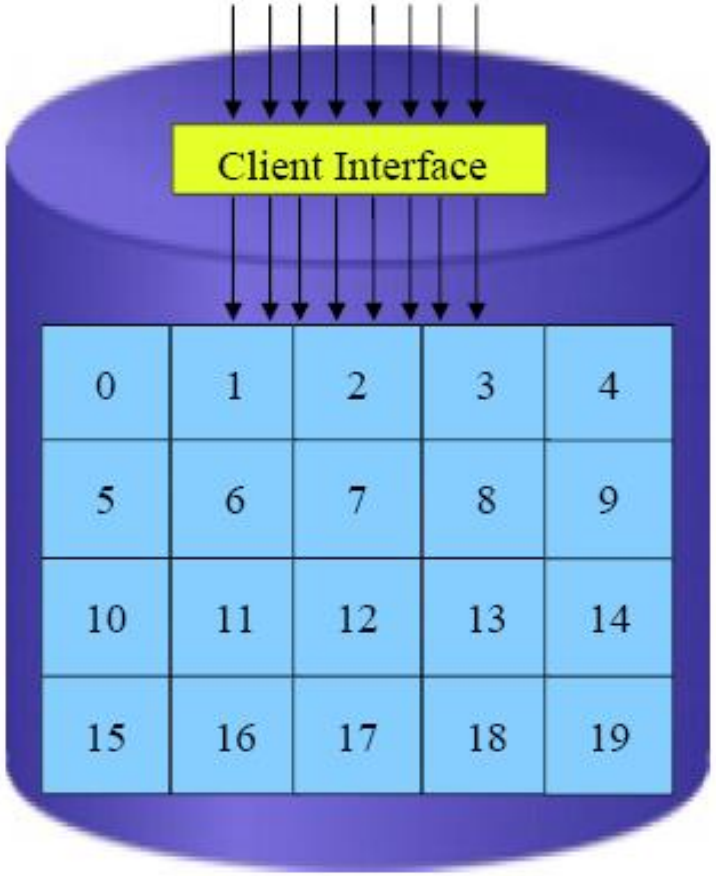


SCSI，ATA
SAS，SATA
FCP，iSCSI
SRP，NVMe…

DAS, SAN

The Block Paradigm

SCSI，ATA，SAS，SATA，FCP，iSCSI，SRP，NVMe…

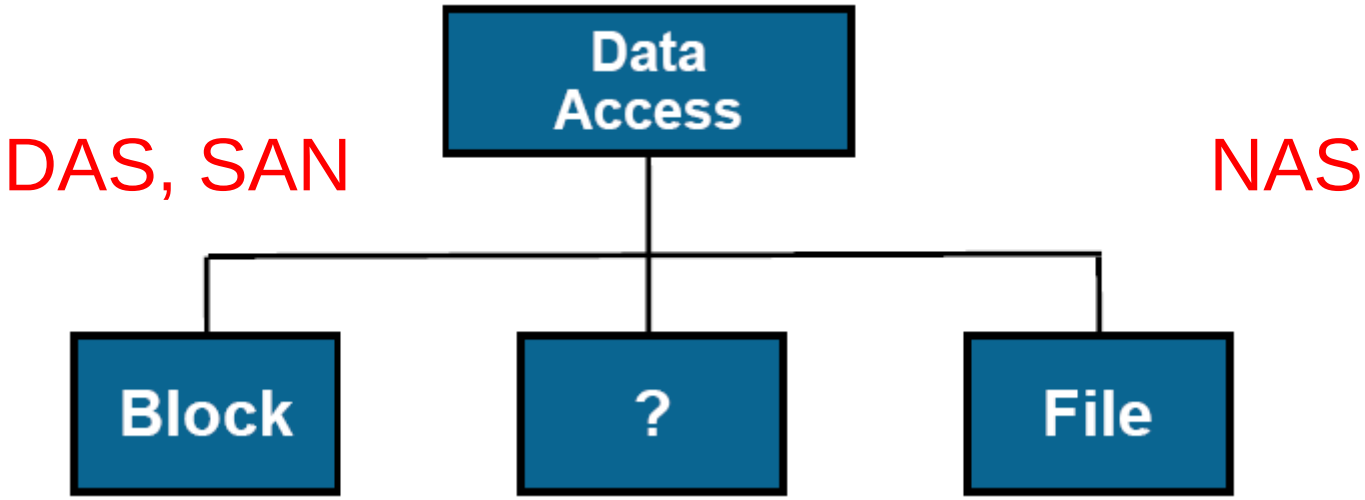


LBA：Logical Block Address

Physical Blocks:
e.g. 512 bytes
4KB

The Data Access Taxonomy

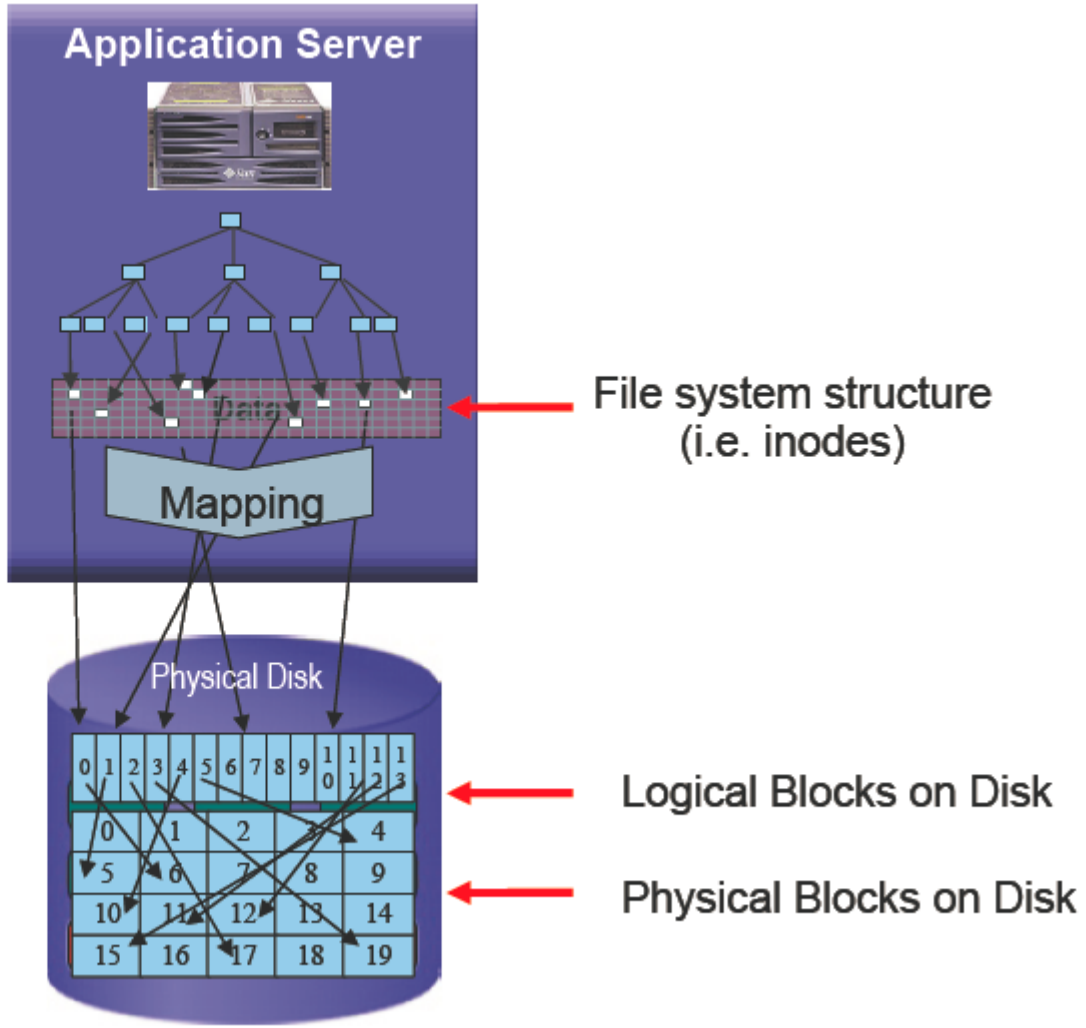
- The File Paradigm



Local File Systems

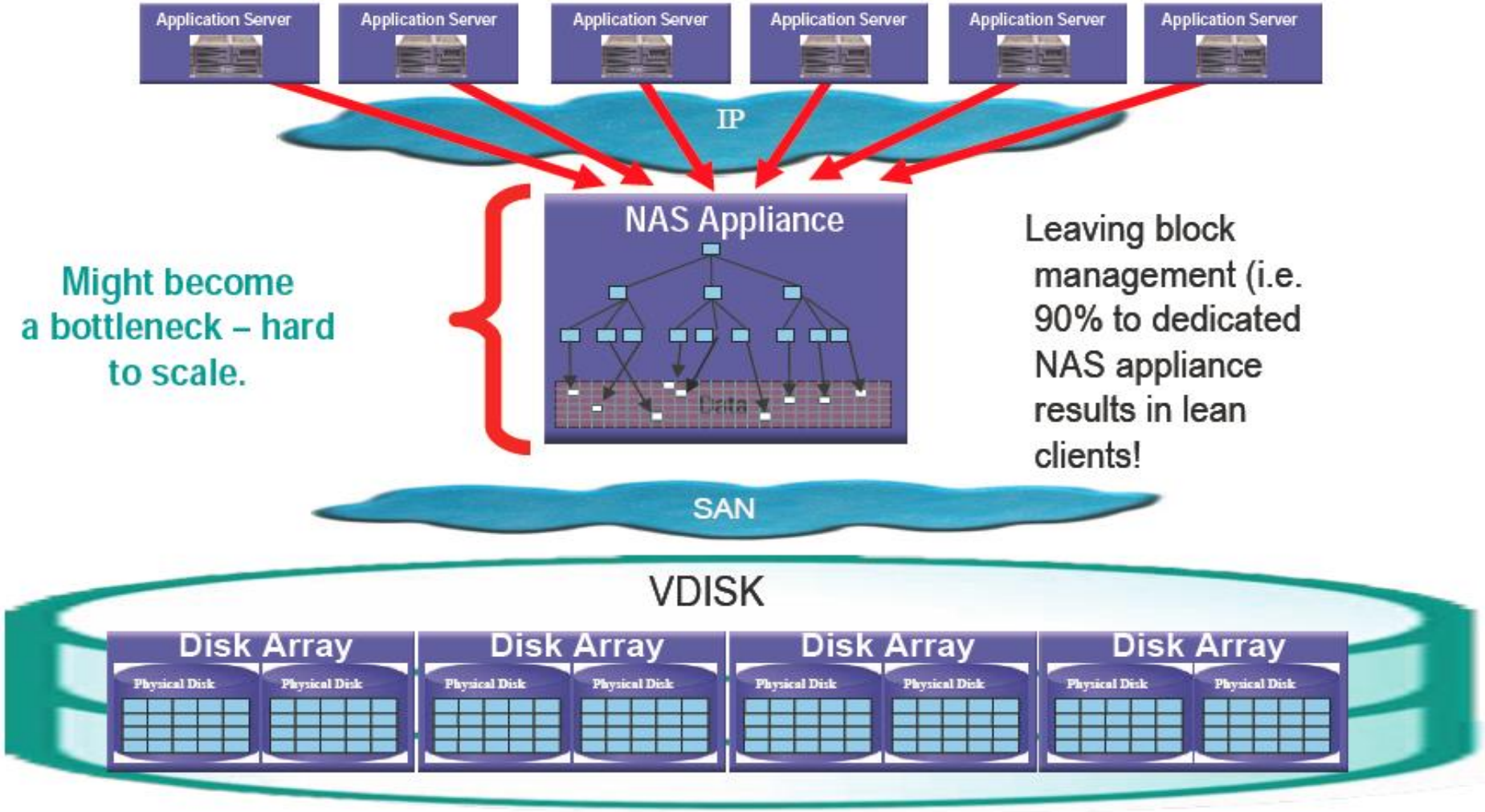
- file/directory management(~10% of workload)
- block/sector management (~90% of workload)

One more level of indirection



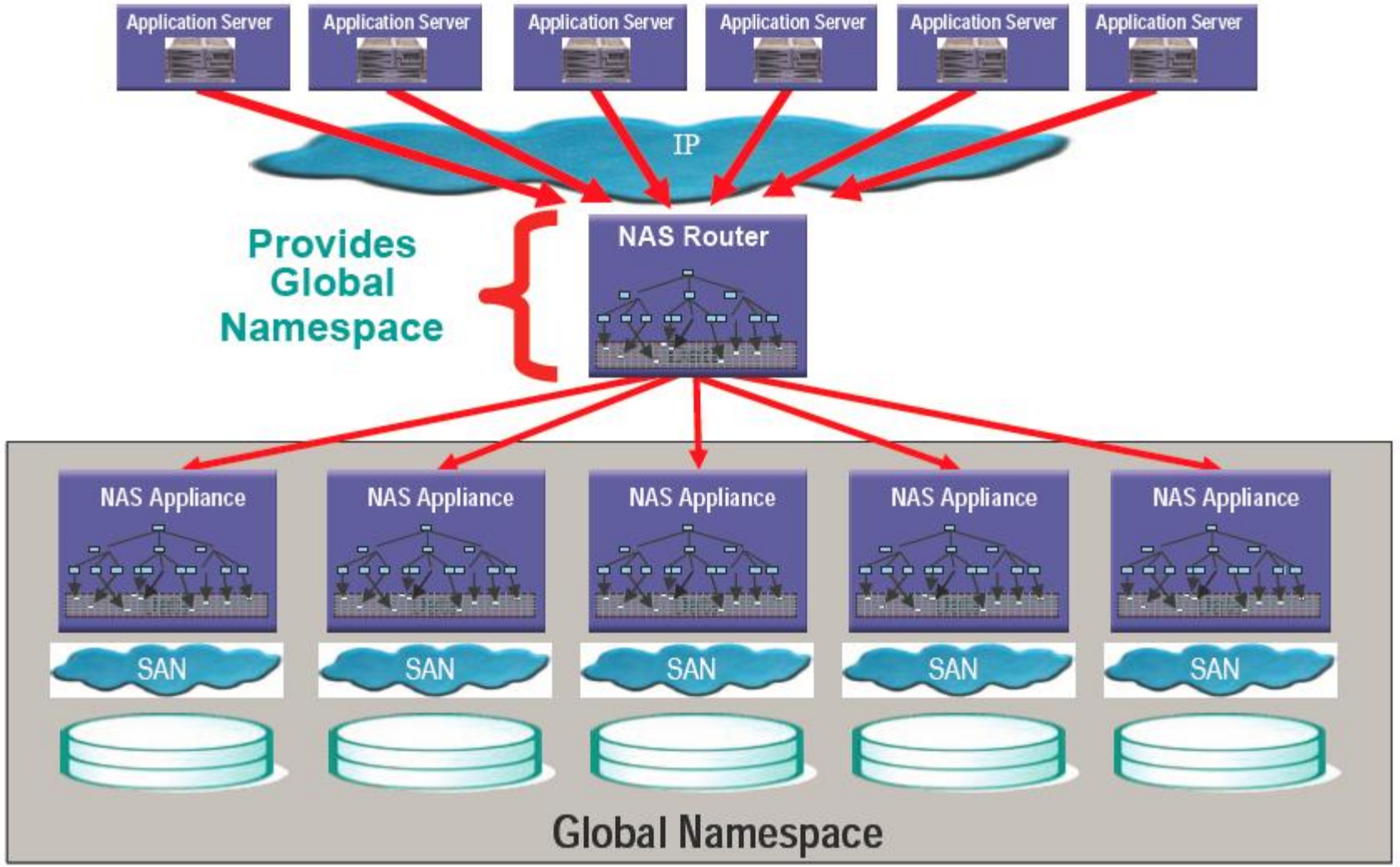
Distributed File Systems

e.g. NAS with NFS,CIFS Protocol



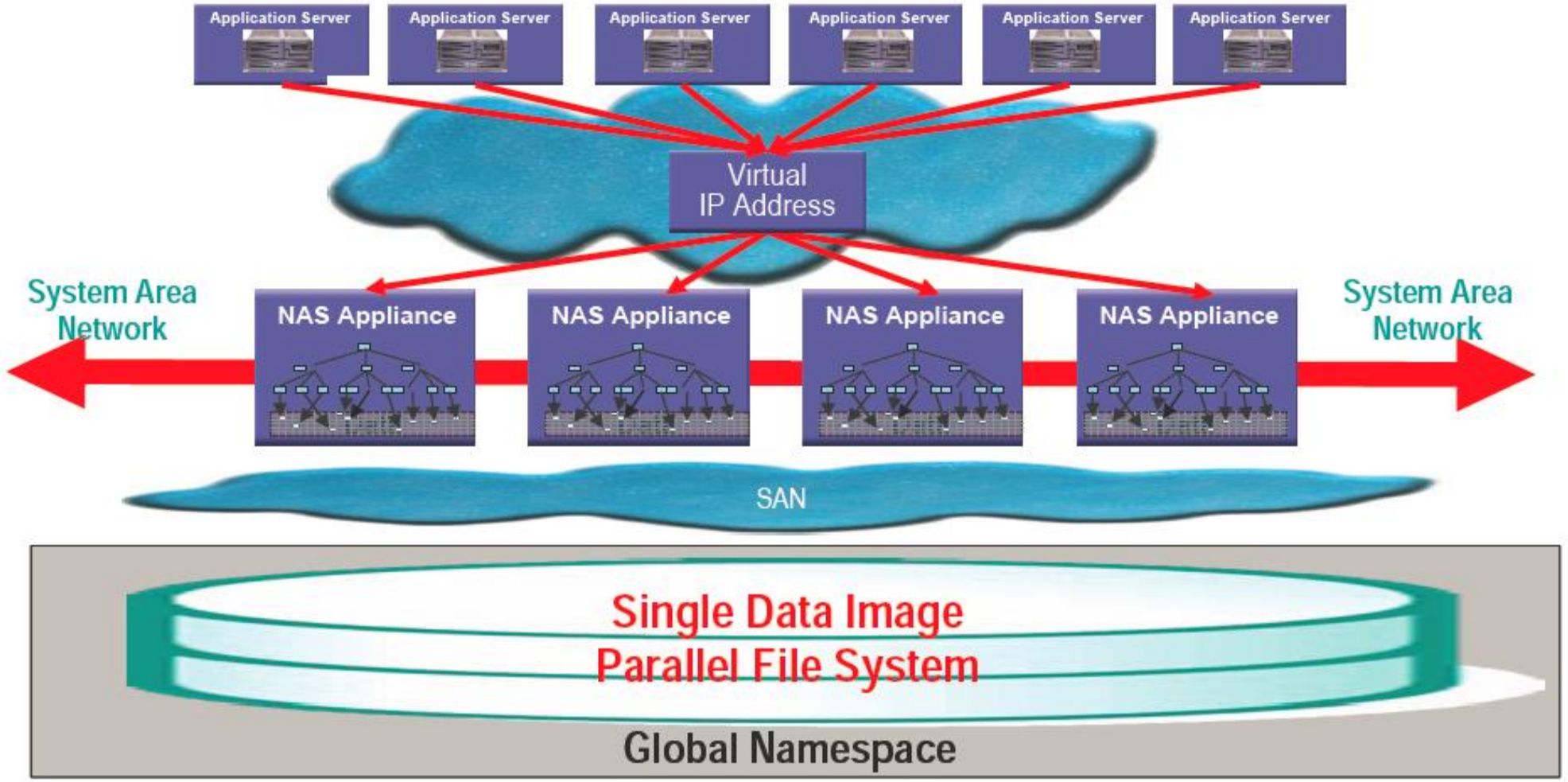
NAS Aggregation/Virtualization

Global Namespace



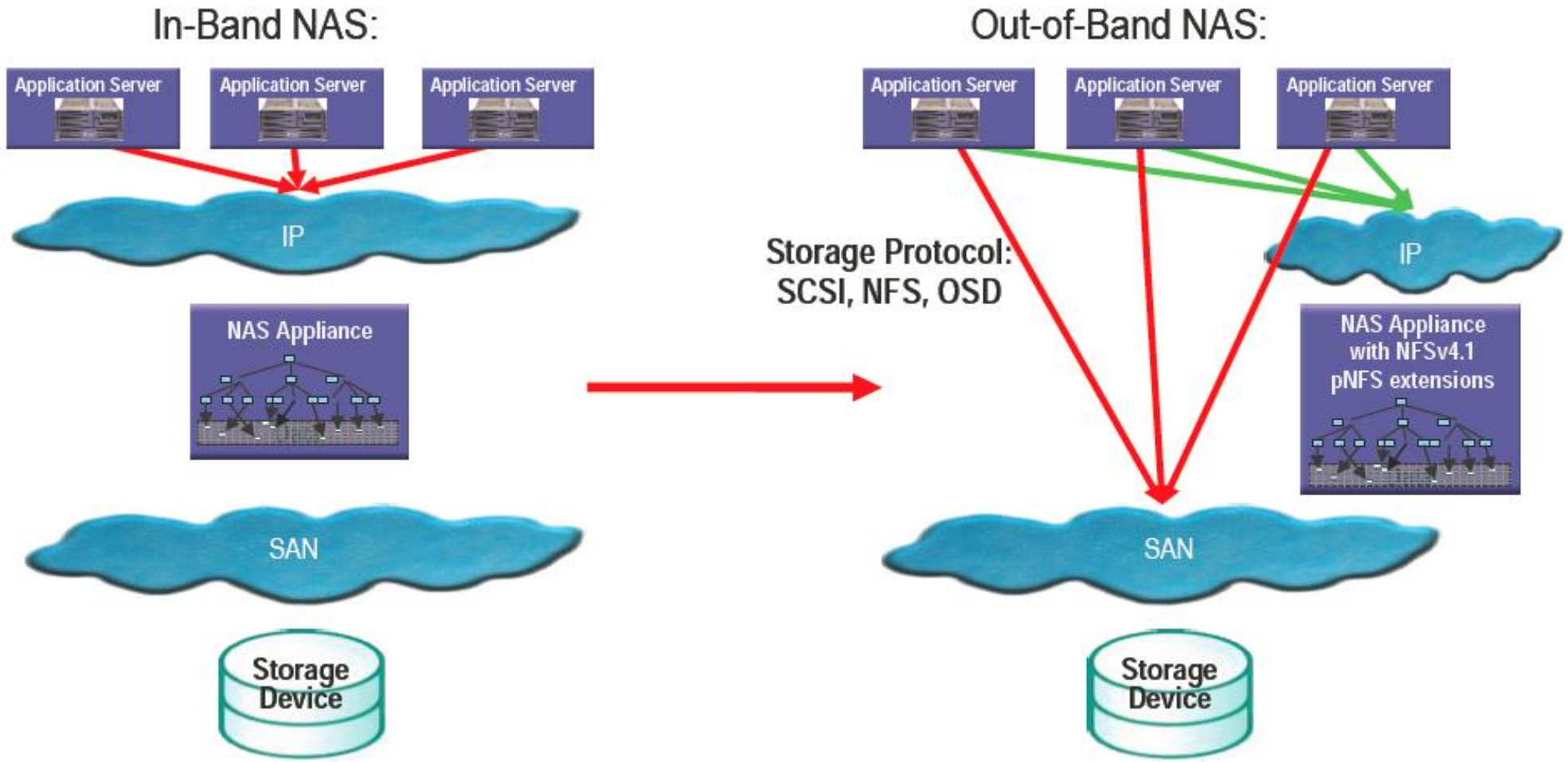
NAS Cluster

Tightly Coupled NAS



NAS Cluster

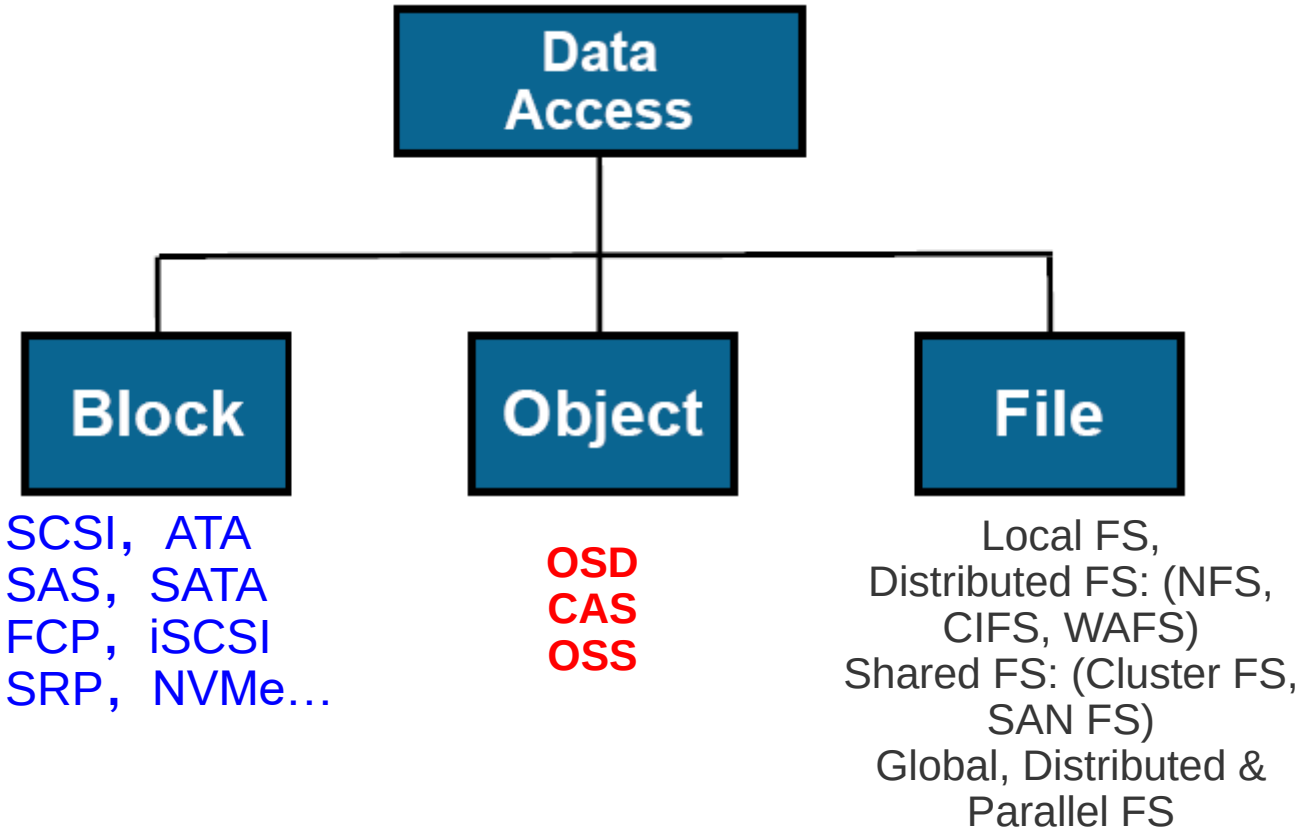
Loosely Coupled NAS: Global Namespace with NFSv4.1 and pNFS



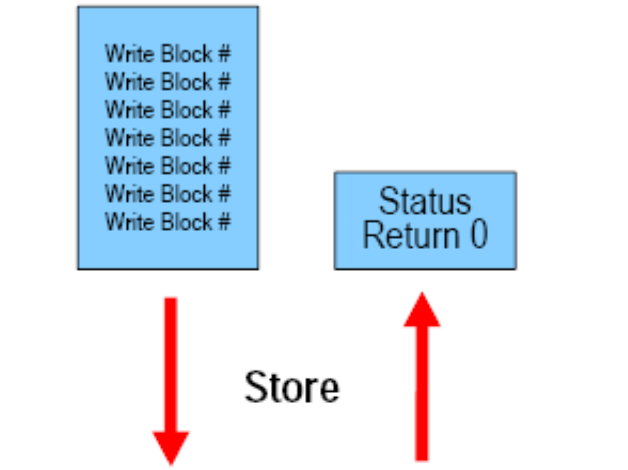
Scalable NAS
Loosely Coupled NAS Cluster

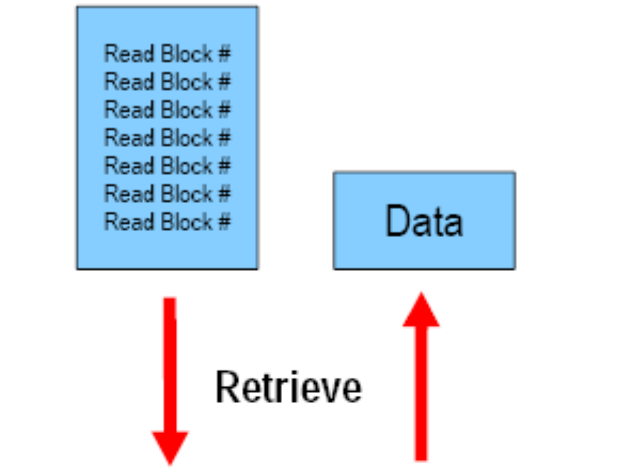
The Data Access Taxonomy

- The Object Paradigm

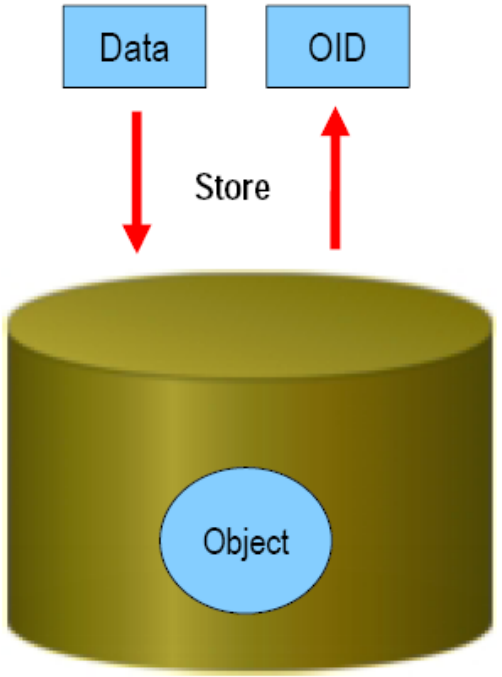


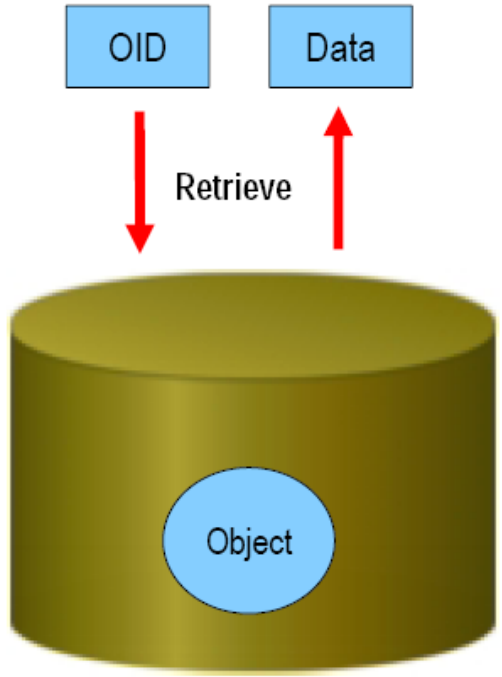
The Old Block Paradigm



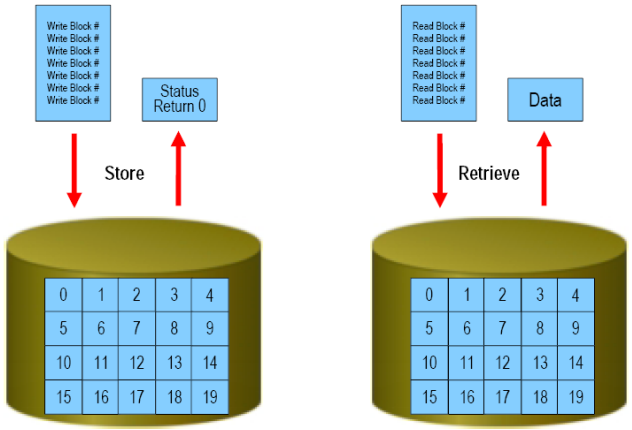


The New Object Paradigm



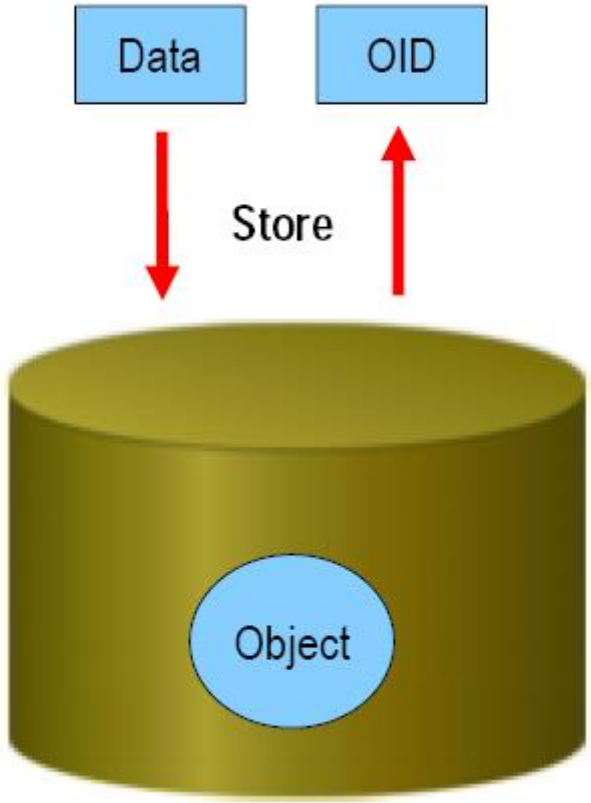


Block Paradigm



The New Object Paradigm

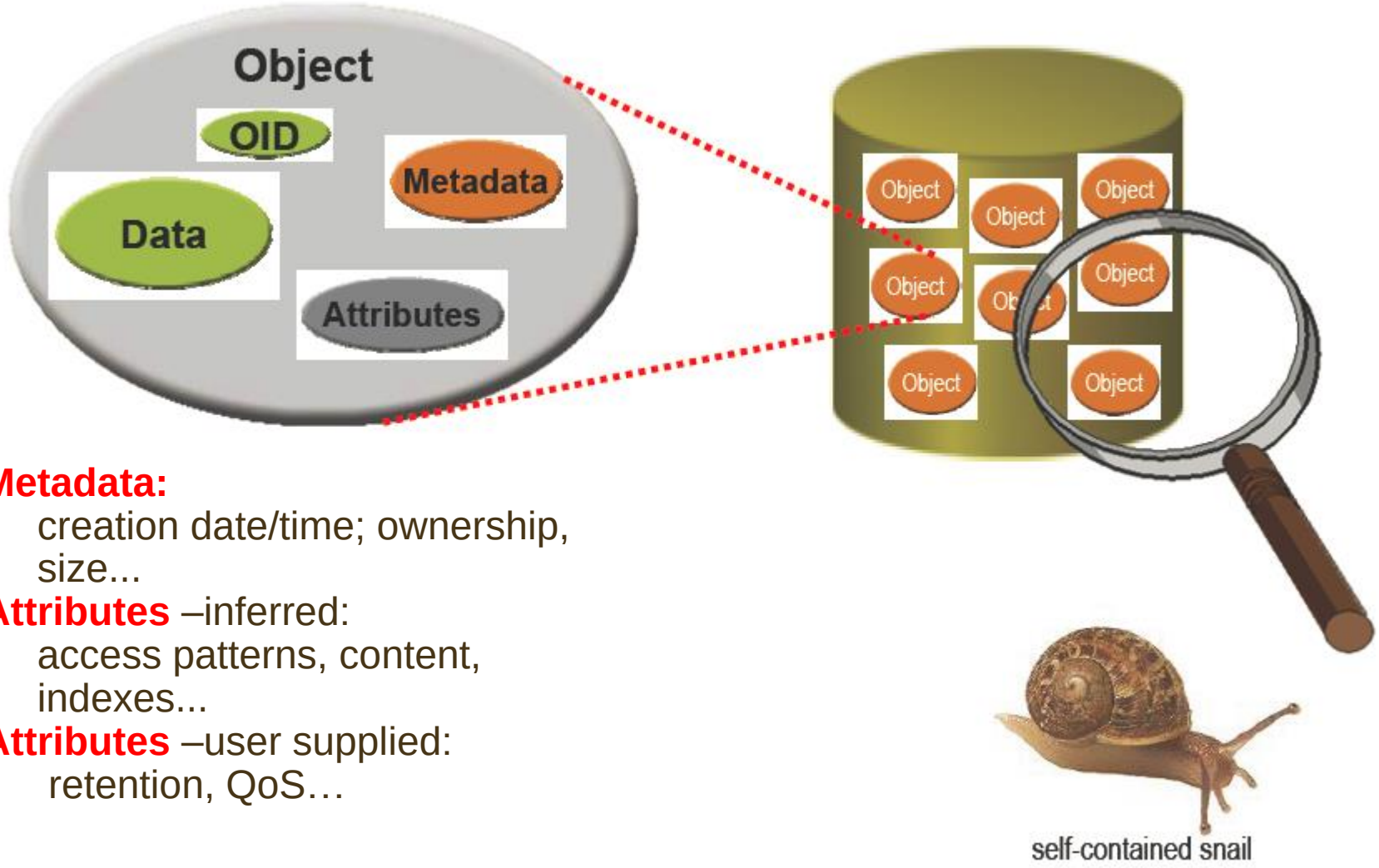
- WRITE 26,763 Bytes
- QoS= High
- Description = “X-Ray”
- Retention = 50 years
- Access Key = *&^%#
- Data Payload.....



Object Storage Responsibilities:

- Space Management
- Access Control (Identity Mgmt)
- QoS Management
- Cache, Backup
- Policy Migration, Retention

Self-Contained Objects



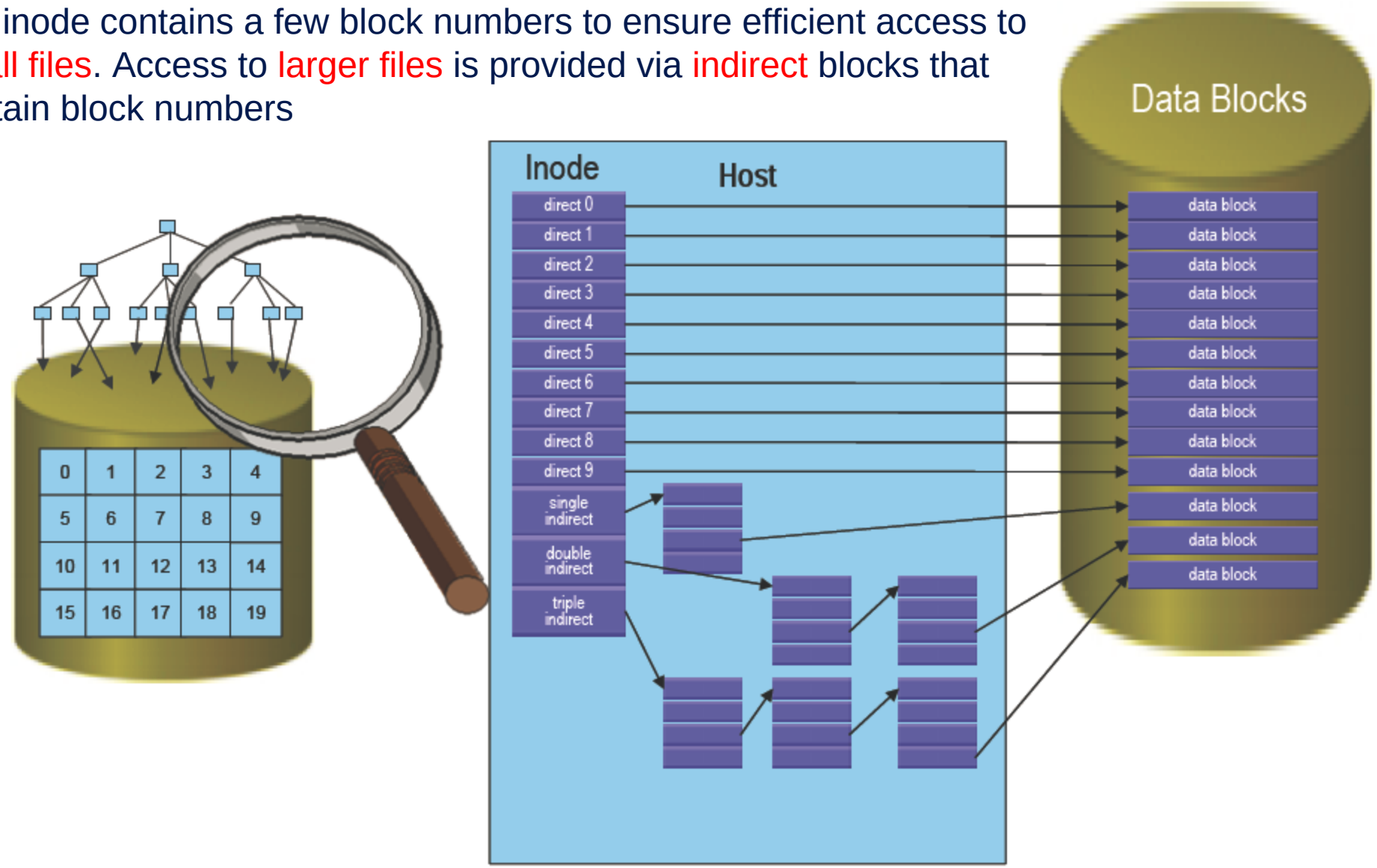
Metadata:
creation date/time; ownership, size...

Attributes –inferred:
access patterns, content, indexes...

Attributes –user supplied:
retention, QoS…

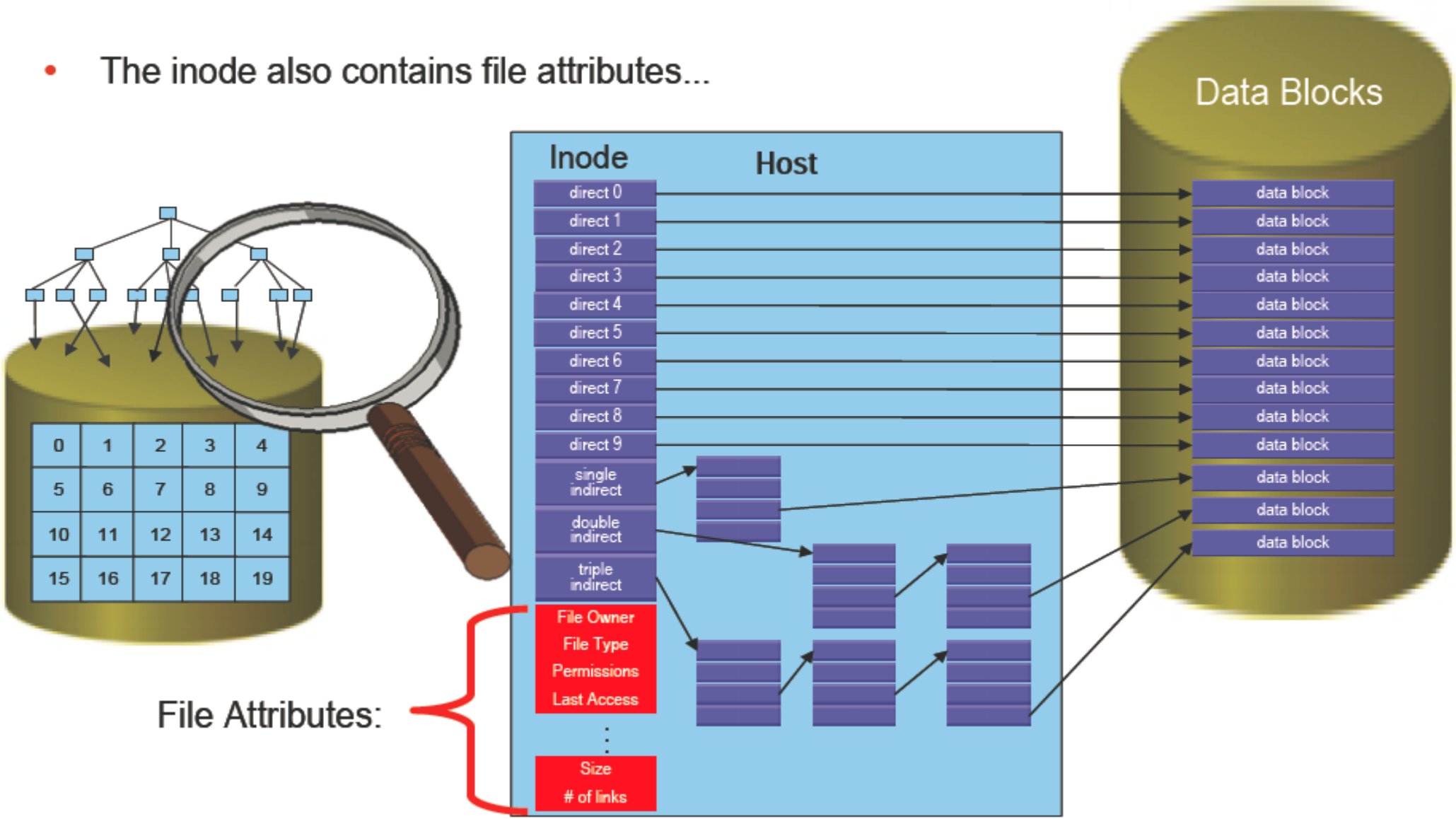
Block Access - Inodes

The inode contains a few block numbers to ensure efficient access to small files. Access to larger files is provided via indirect blocks that contain block numbers

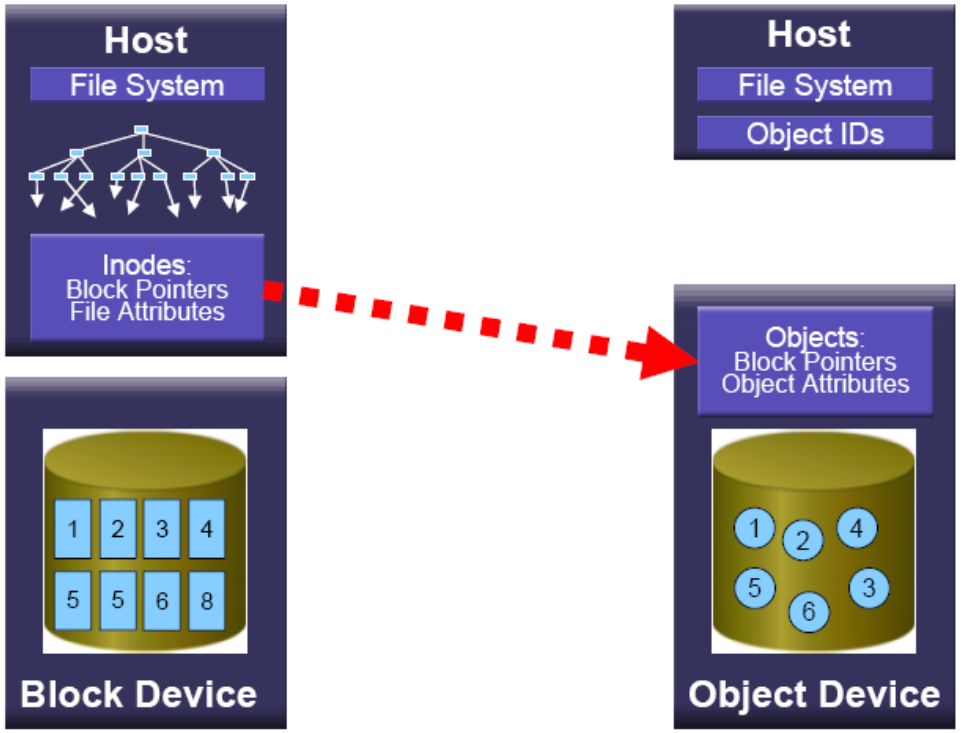


Block Access - Inodes

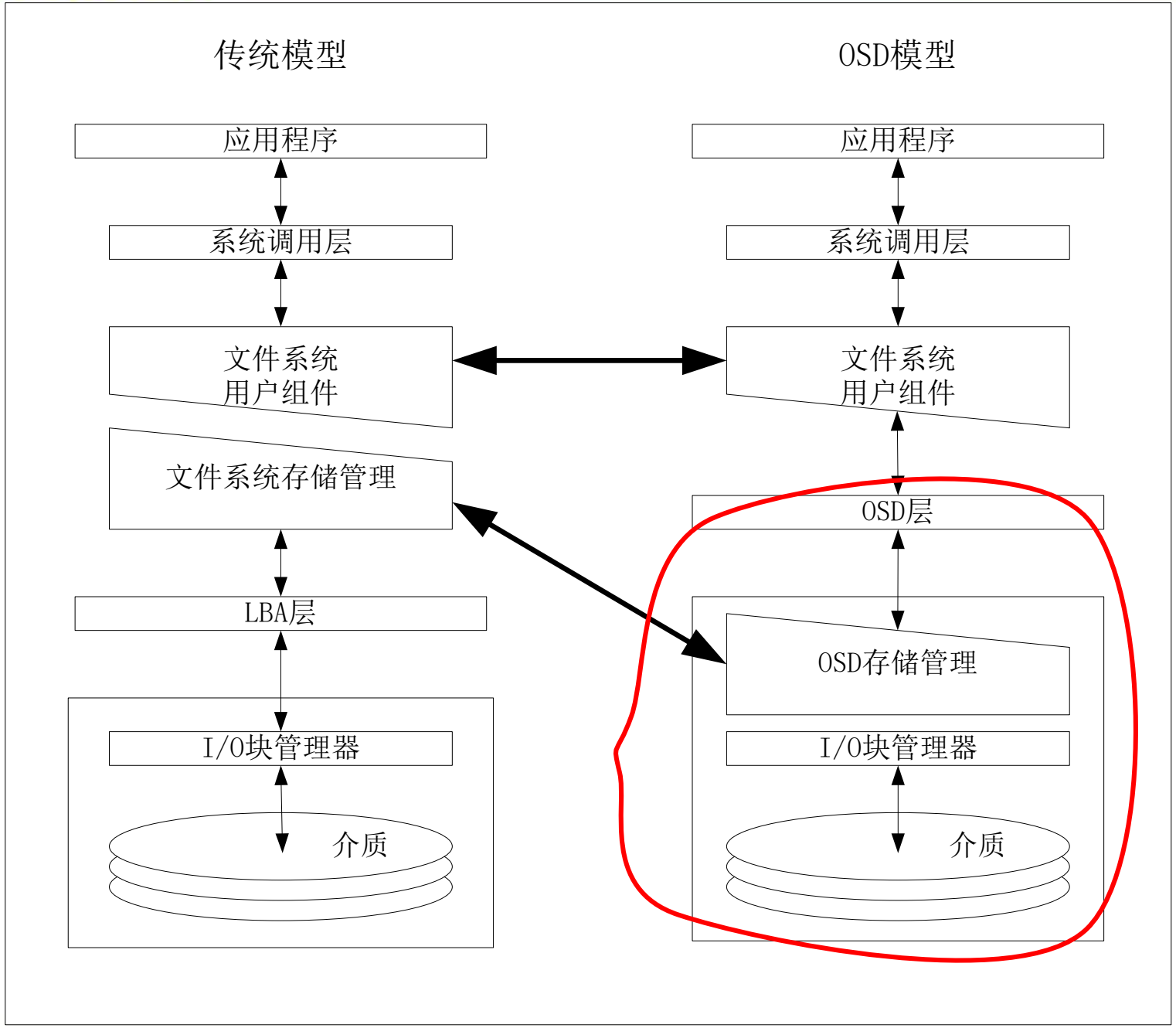
- The inode also contains file attributes...



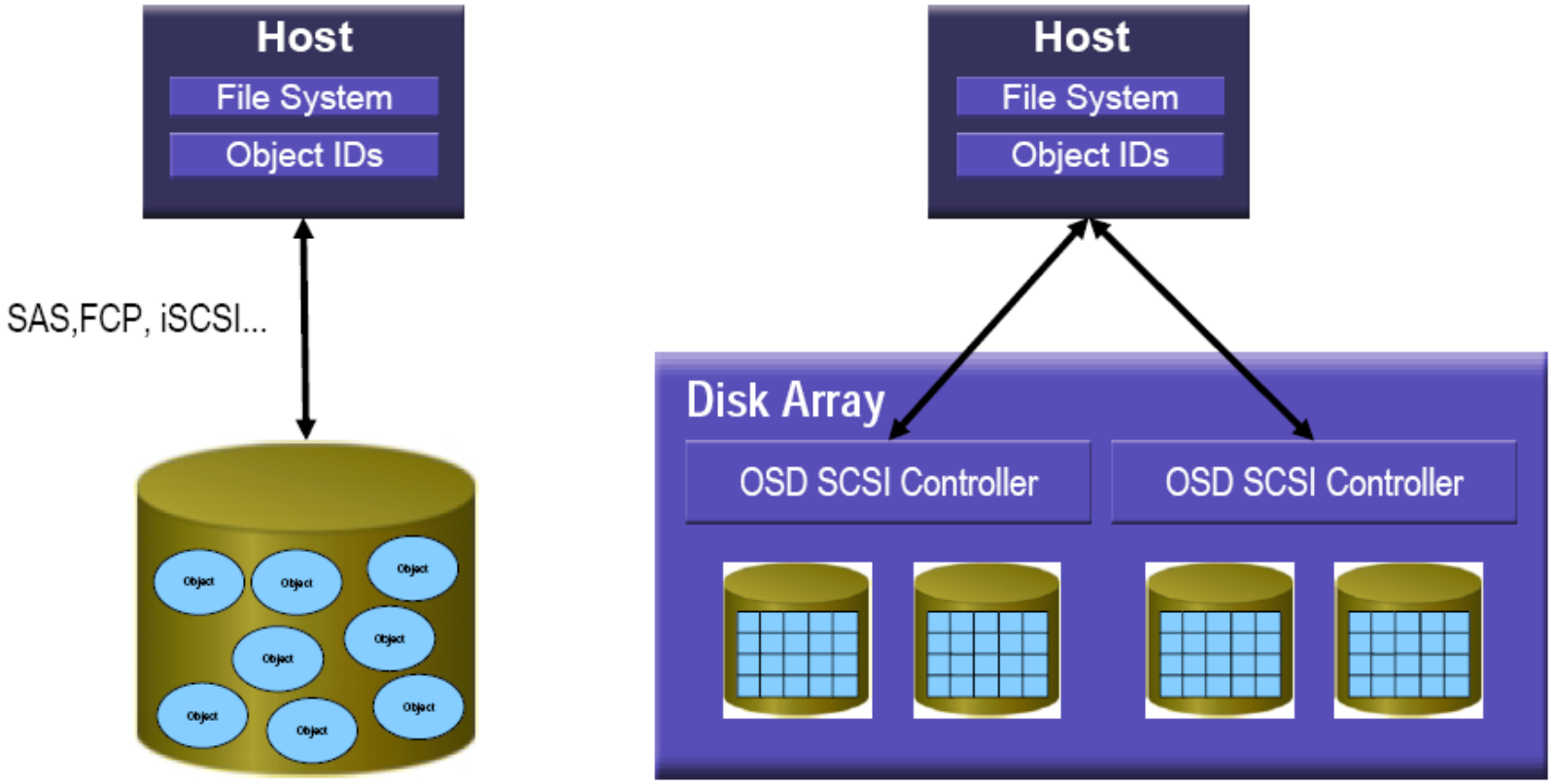
Inodes vs. Objects



- Abstract some of the lower layers of storage away from the administrators and applications.
- Inclusion of rich custom metadata **within** the Object.
  - ✓ specific information(from user or app.) for better indexing purposes
  - ✓ Support data-management policies
  - ✓ Centralize management of storage across many individual nodes and clusters
  - ✓ Optimize metadata storage and caching/indexing independently from the data storage


传统模型
OSD模型
应用程序
系统调用层
文件系统
用户组件
文件系统存储管理
LBA层
I/O块管理器
介质
应用程序
系统调用层
文件系统
用户组件
OSD层
OSD存储管理
I/O块管理器
介质

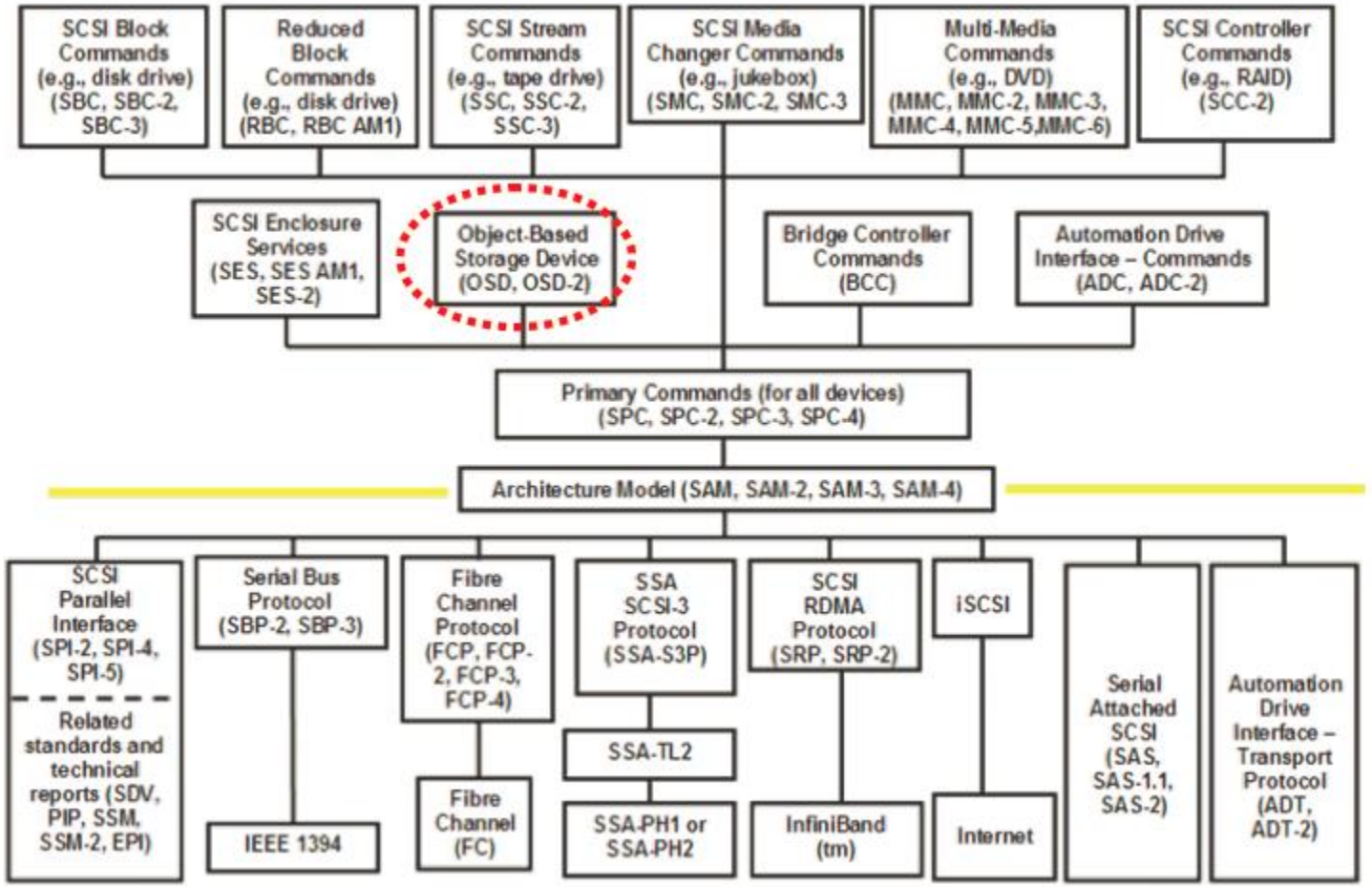
ANSI T10 OSD SCSI Targets



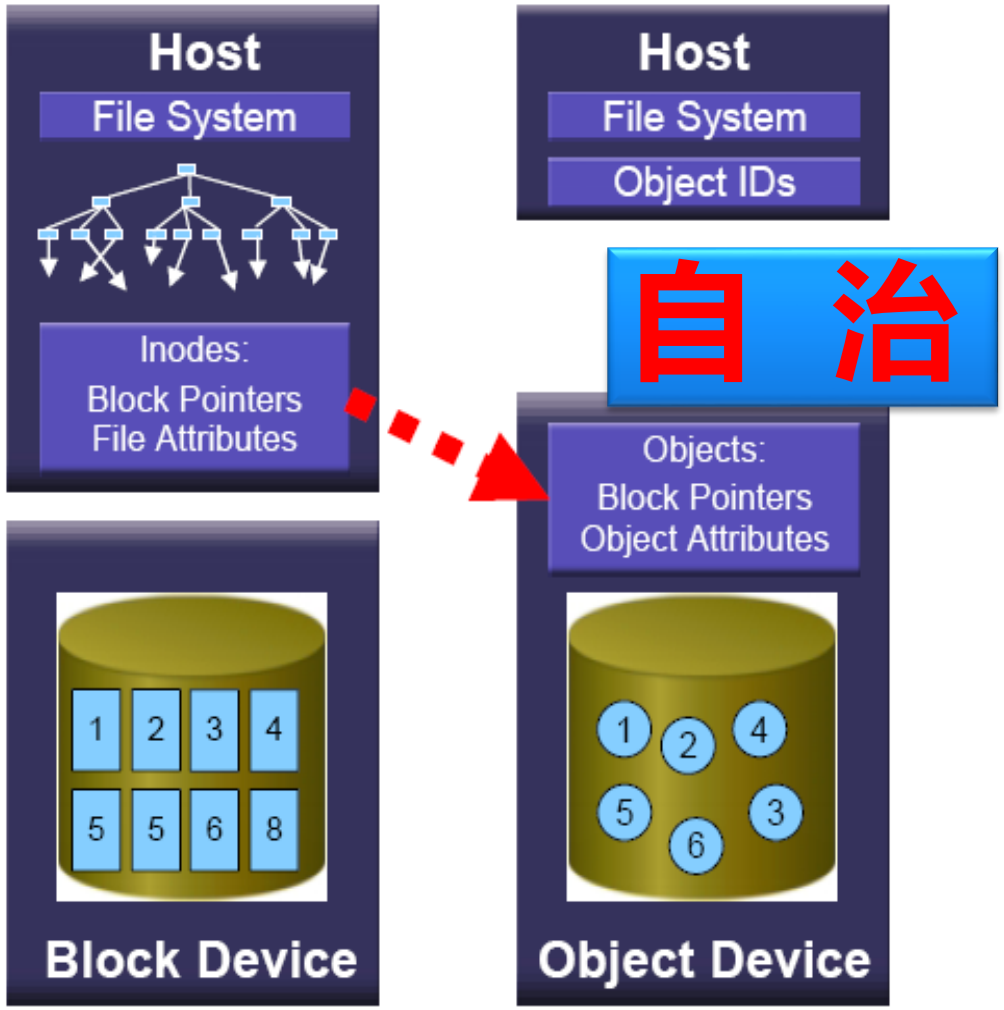
对象存储是一种将数据作为对象进行管理的计算机数据存储体系结构。与文件存储和块存储不同，每个对象通常包括数据本身，数量可变的元数据和全局唯一标识符。对象存储可以在多个级别上实现，包括设备级别，系统级别和接口级别。

SCSI Standards Architecture

设备级
对象存储



Object Autonomy



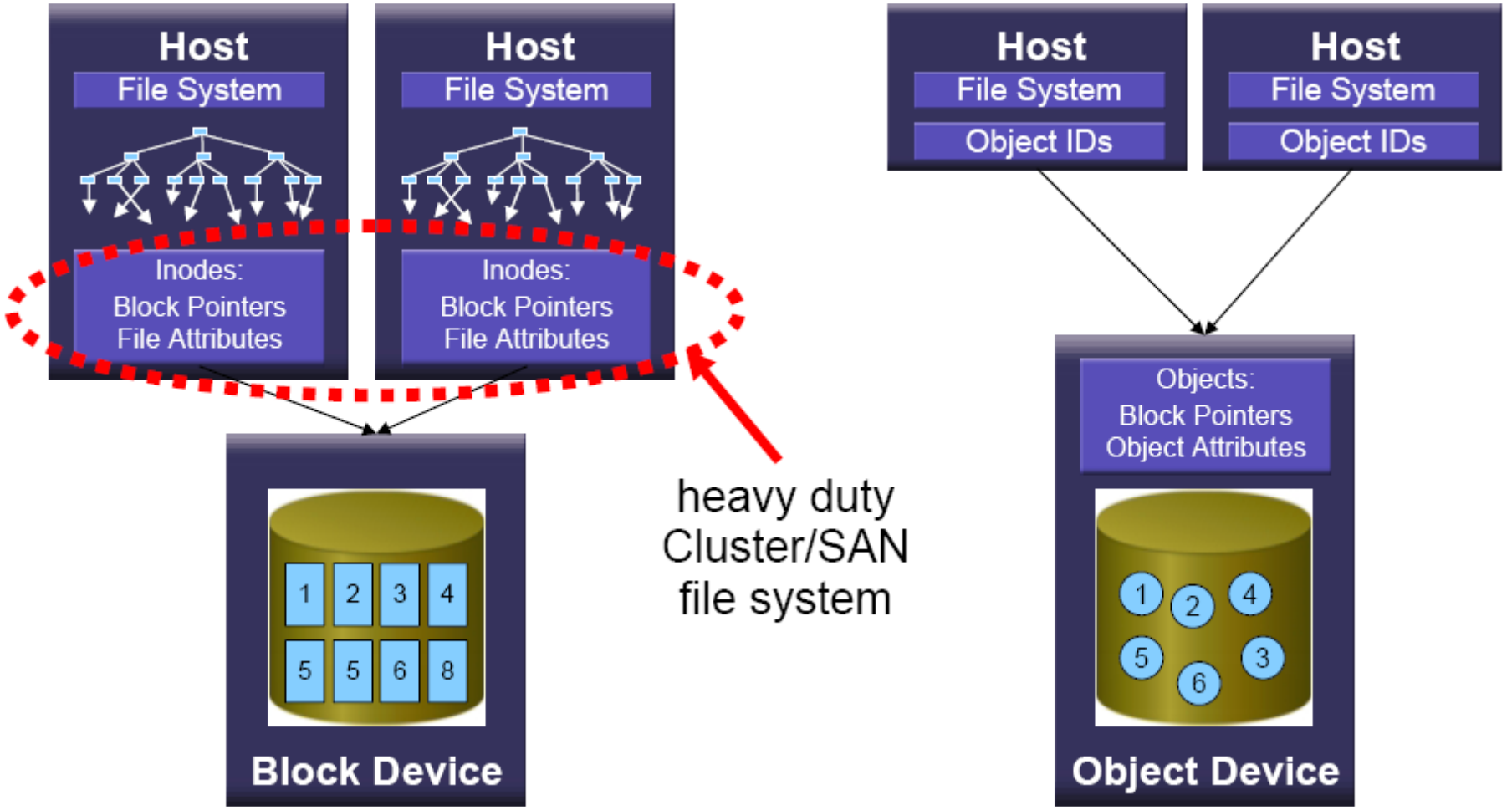
Storage becomes autonomous:

- –capacity planning，thin provisioning
- –load balancing
- –backup
- –QoS, SLAs
- –understand data/object grouping
- –aggressive pre-fetching
- –search
- –compression/de-duplication/encryption
- –strong security
- –compliance/retention/secure delete
- –availability/replication
- –audit

…

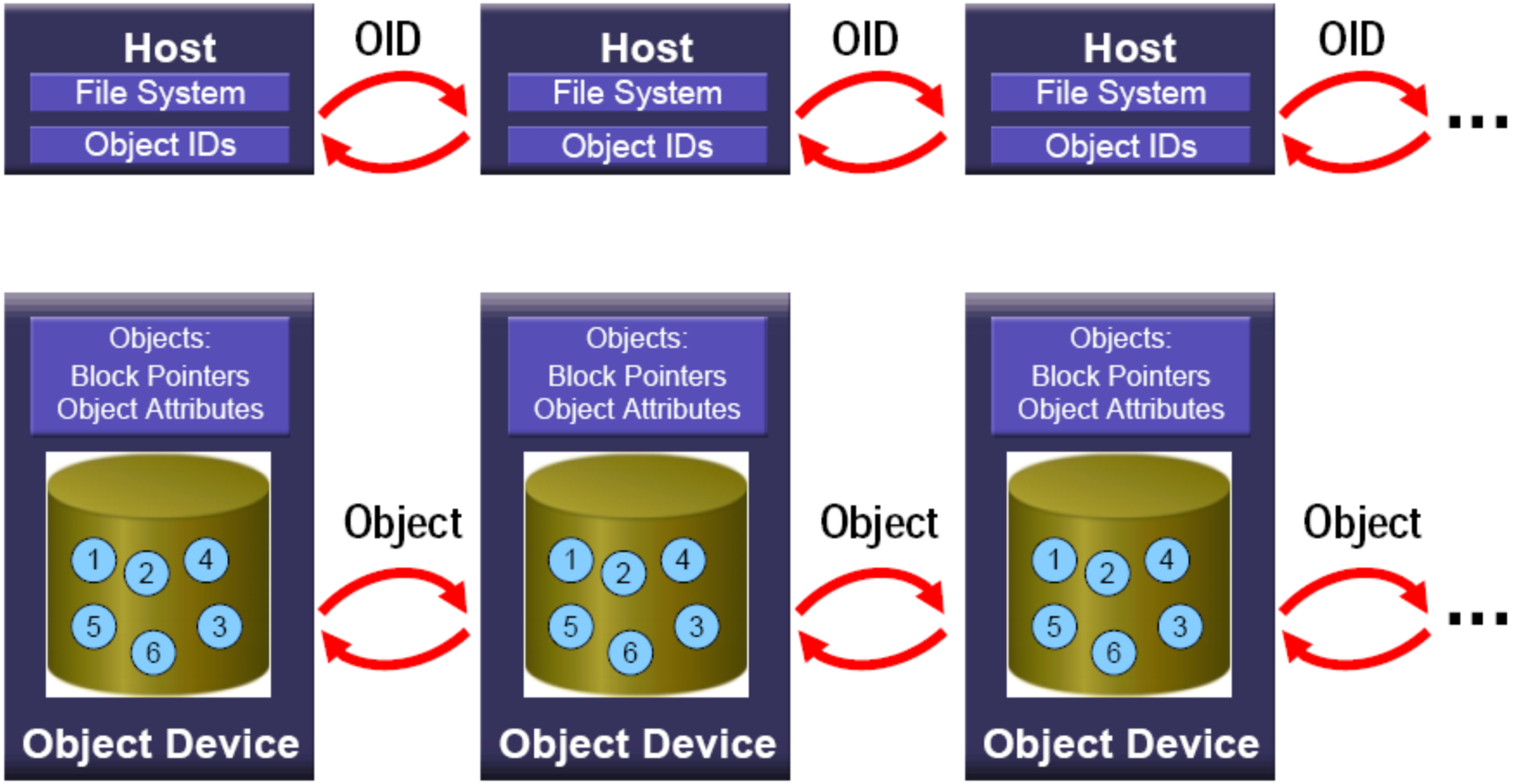
Data Sharing

Homogeneous/Heterogeneous

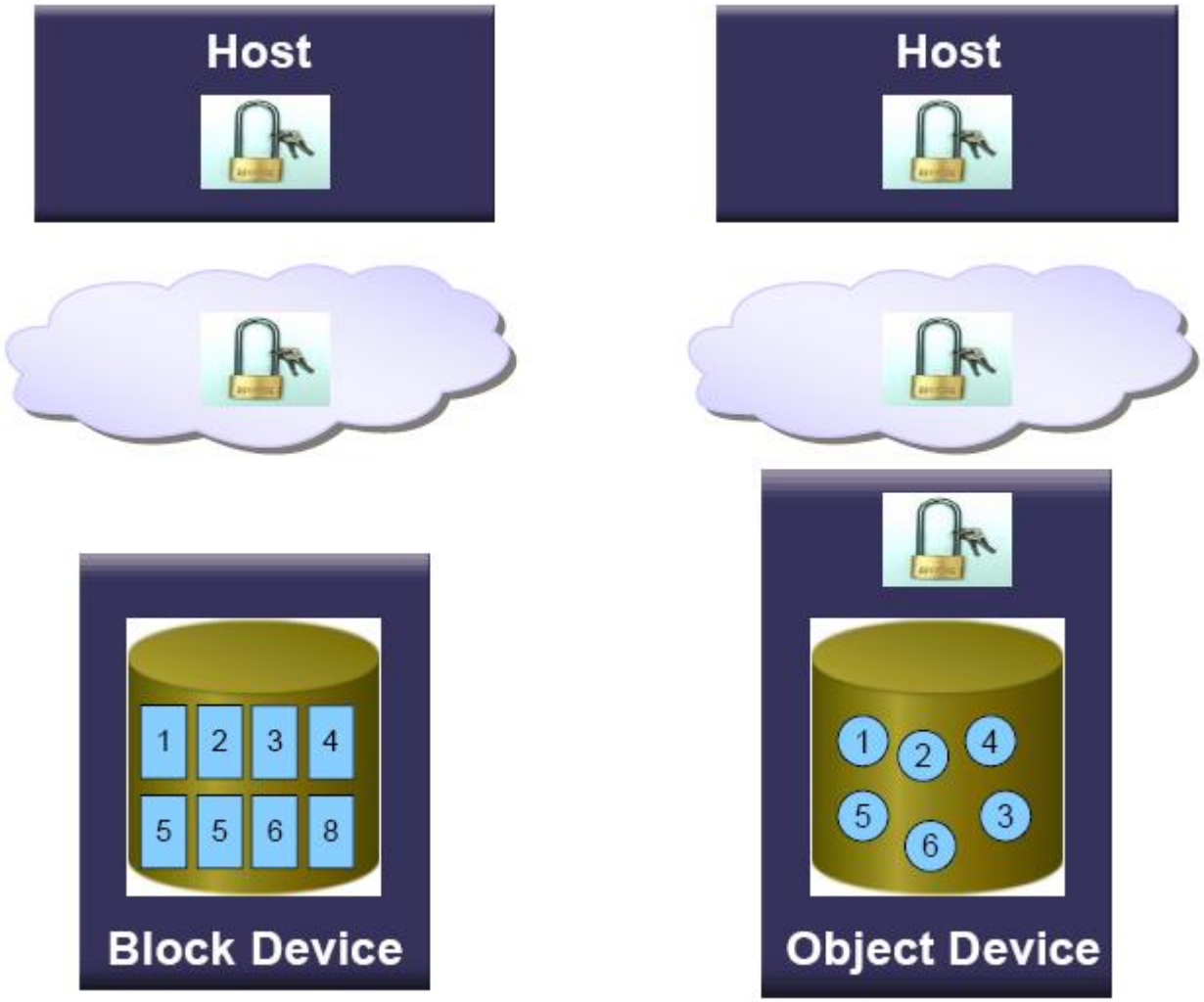


Data Migration - ILM

Homogeneous/Heterogeneous



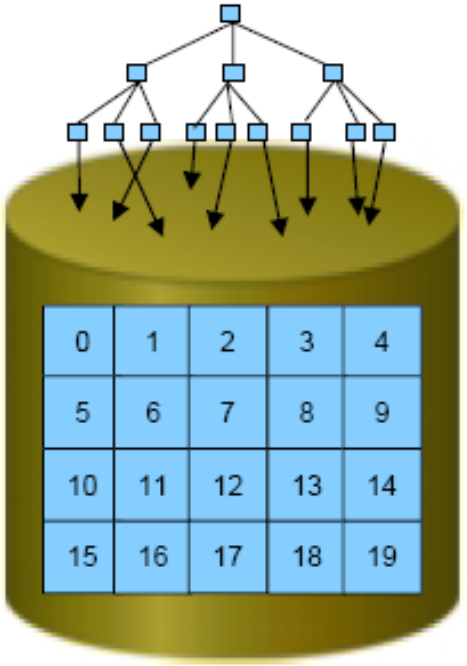
Additional Layer of Security



- strong security via external service
  - authentication
  - authorization
  - ....
- fine granularity
  - per object

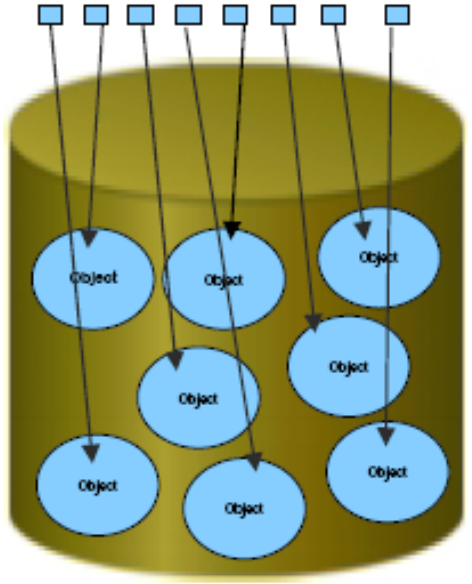
Living in a Flat Namespace

File names / inodes



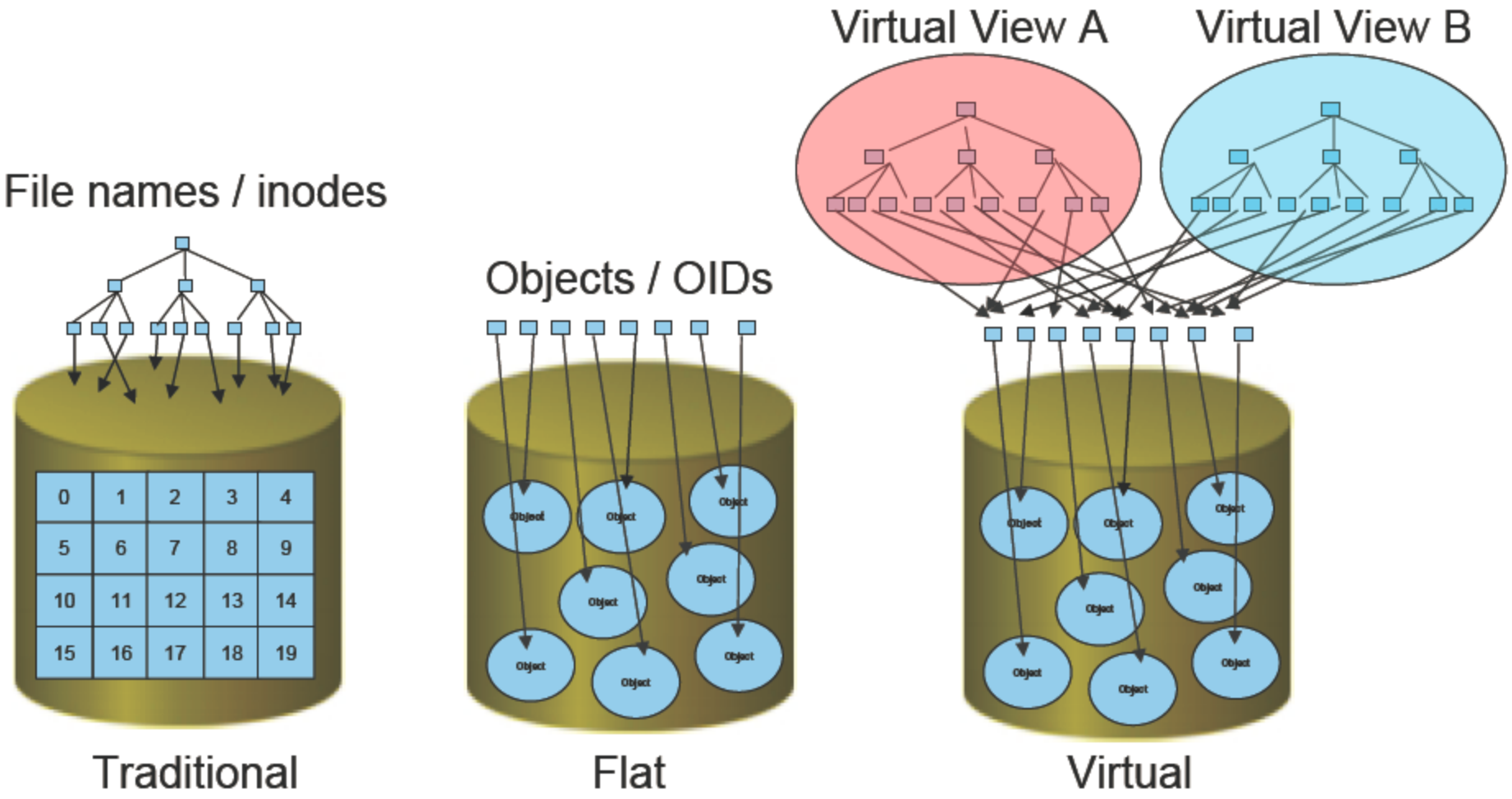
Traditional
Hierarchical

Objects / OIDs

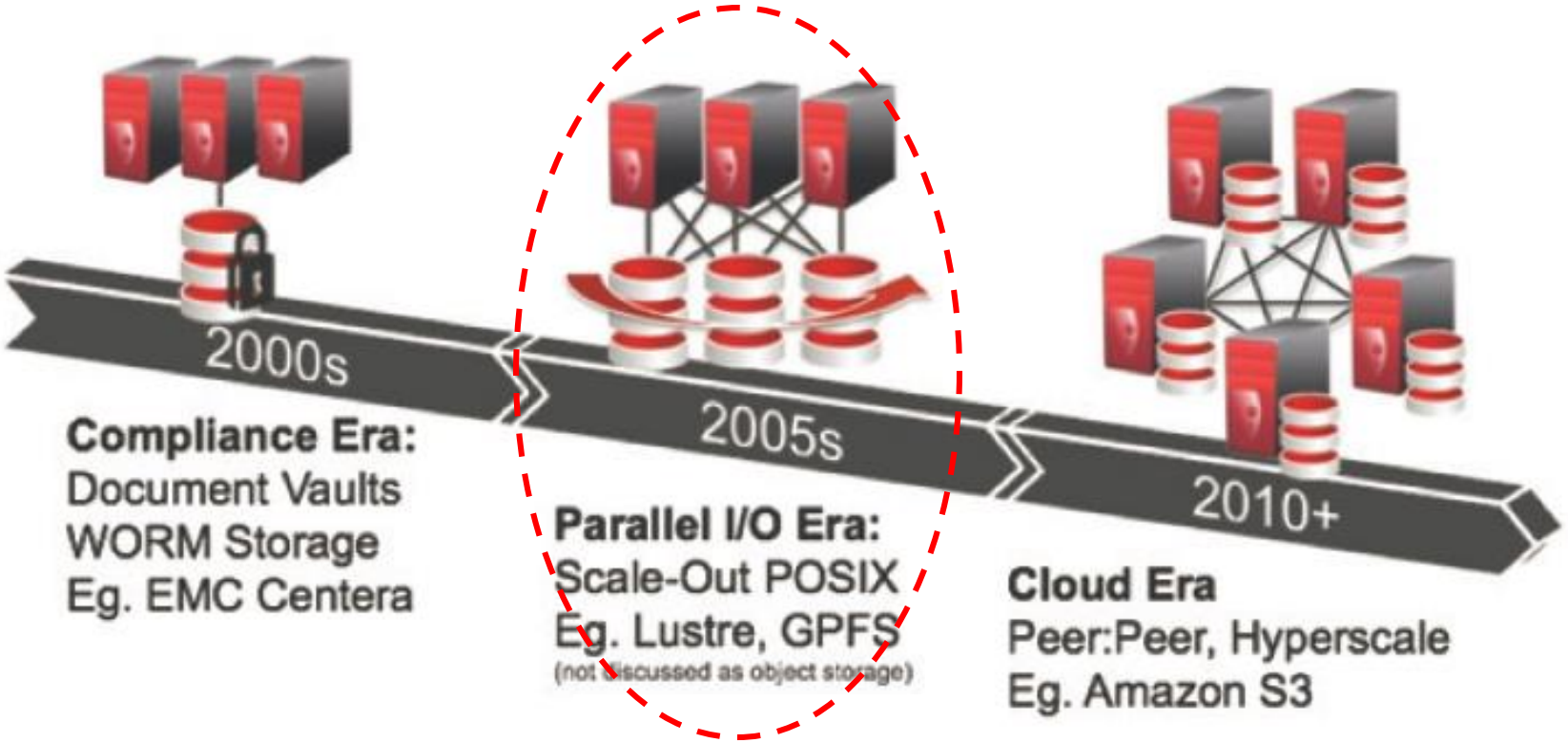


Flat

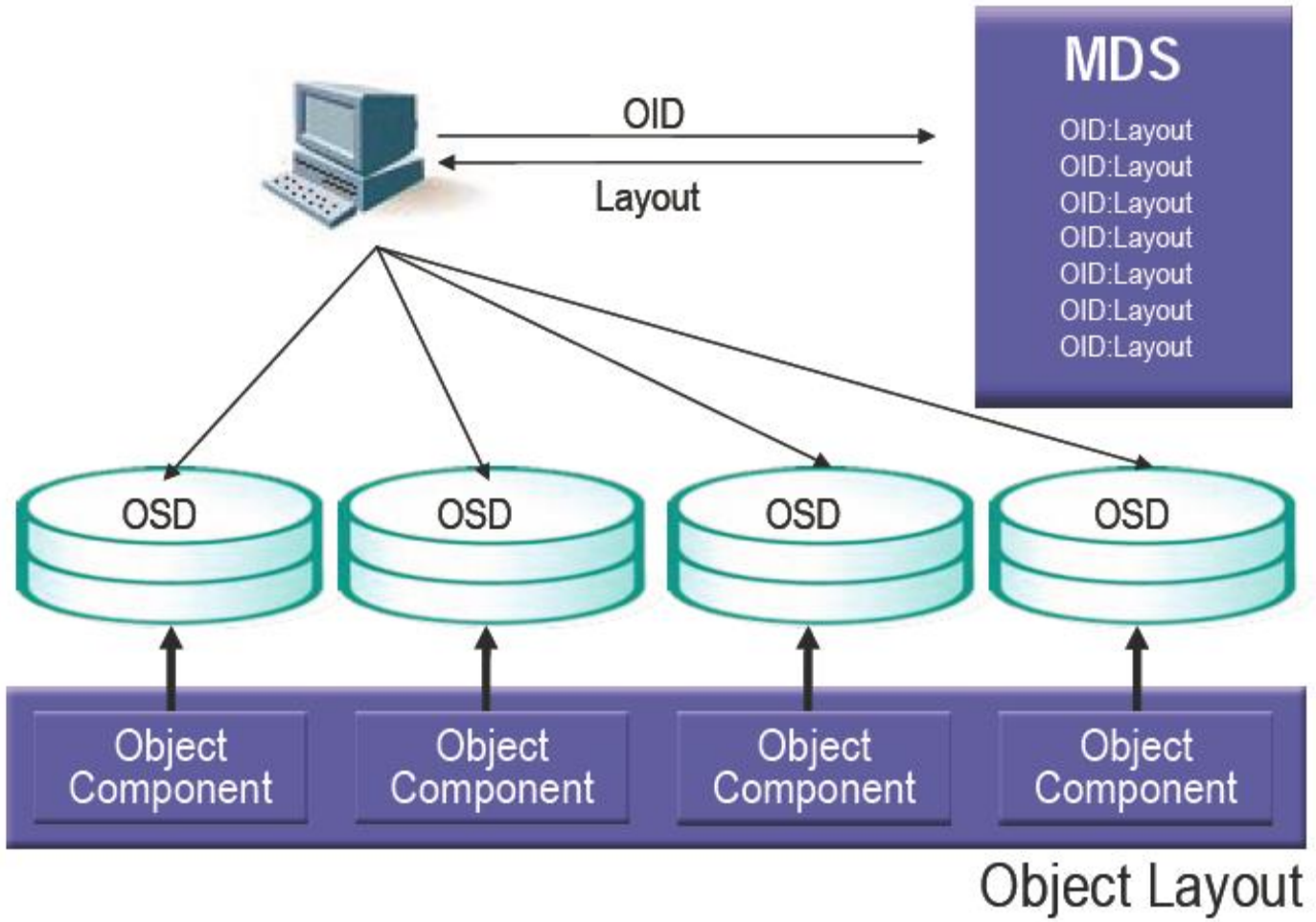
Virtual View / Virtual File Systems



The First Generation of Object Storage（Device-level）



Object Decomposition



对象存储与传统存储的对比

	存储接口	存储系统	优点	缺点
块级存储	块	块存储设备	提供高性能的随机I/O和数据吞吐率，如: SAN	可扩展性和可管理性较差、价格较高、不能满足成千上万CPU 规模的系统
文件储存	文件	块存储设备＋文件系统	扩展性好、易于管理、价格便宜，如:NAS	开销高、带宽低、延迟大，不利于高性能集群中应用
对象存储	对象	块存储设备＋文件系统＋定位逻辑＋应用程序	支持高并行、可伸缩的数据访问，管理性好、安全性高、适合高性能集群使用	兼容性,相应的硬件、软件支持有待进一步完善

对象存储的特性（总结）

- 性能优势
- 存储设备的智能化
- 数据的共享更容易
- 管理更方便
- 更好的安全性

是大数据、AI时代存储组织架构的理想选择

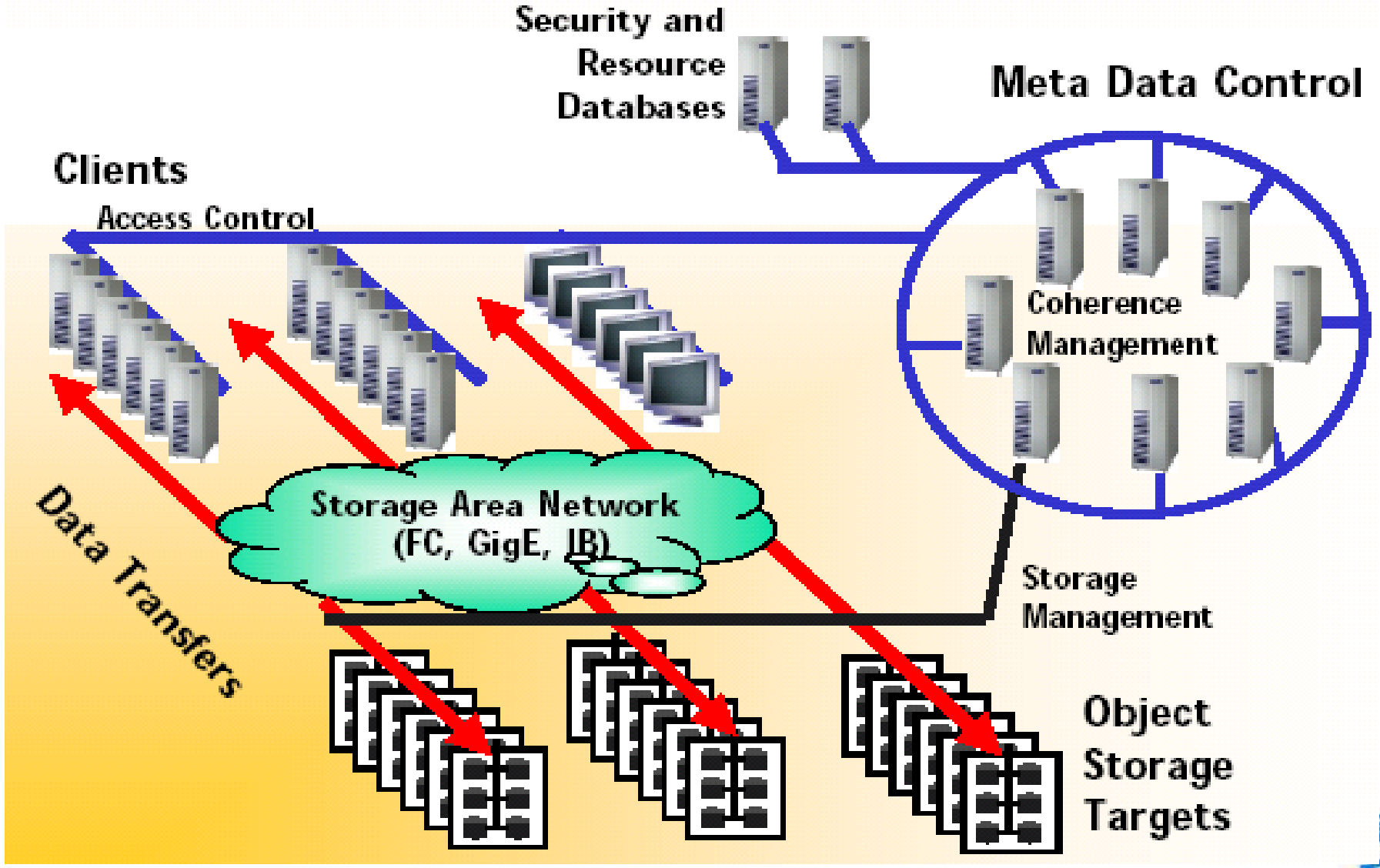
对象存储系统实例：Lustre

- A shared file system for HPC clusters
  - Open Source software (GPL)
  - Linux Cluster
- Very high metadata and I/O performance
  - 5,000 file creations/sec in 1 dir, 1,000 nodes
  - Single clients up to 290MB/sec.
  - Aggregate up to 11GB/sec
- Scalable to 1,000's of nodes
- In production now on such clusters

Lustre Retrospective

- 1999 Initial ideas @CMU
- Seagate: management aspects, prototypes
  - Much survives today
- 2000 National Labs
  - Can Lustre be next generation FS? 100 GB/sec, trillion files, 10,000's clients, secure, PBs
- 2002 – 2003
  - Many partners: Dell, HP, Cray, DDN others. Production use, 1.0 released
- 2004 – 2010 Enterprise-led Phase （Sun, Oracle）
- 2011 – 2016 Open-source community (Intel, TOP500)
- 2017 -- ： DDN Era
  - April 3rd, 2018, Lustre 2.11.0 released

A Lustre Cluster



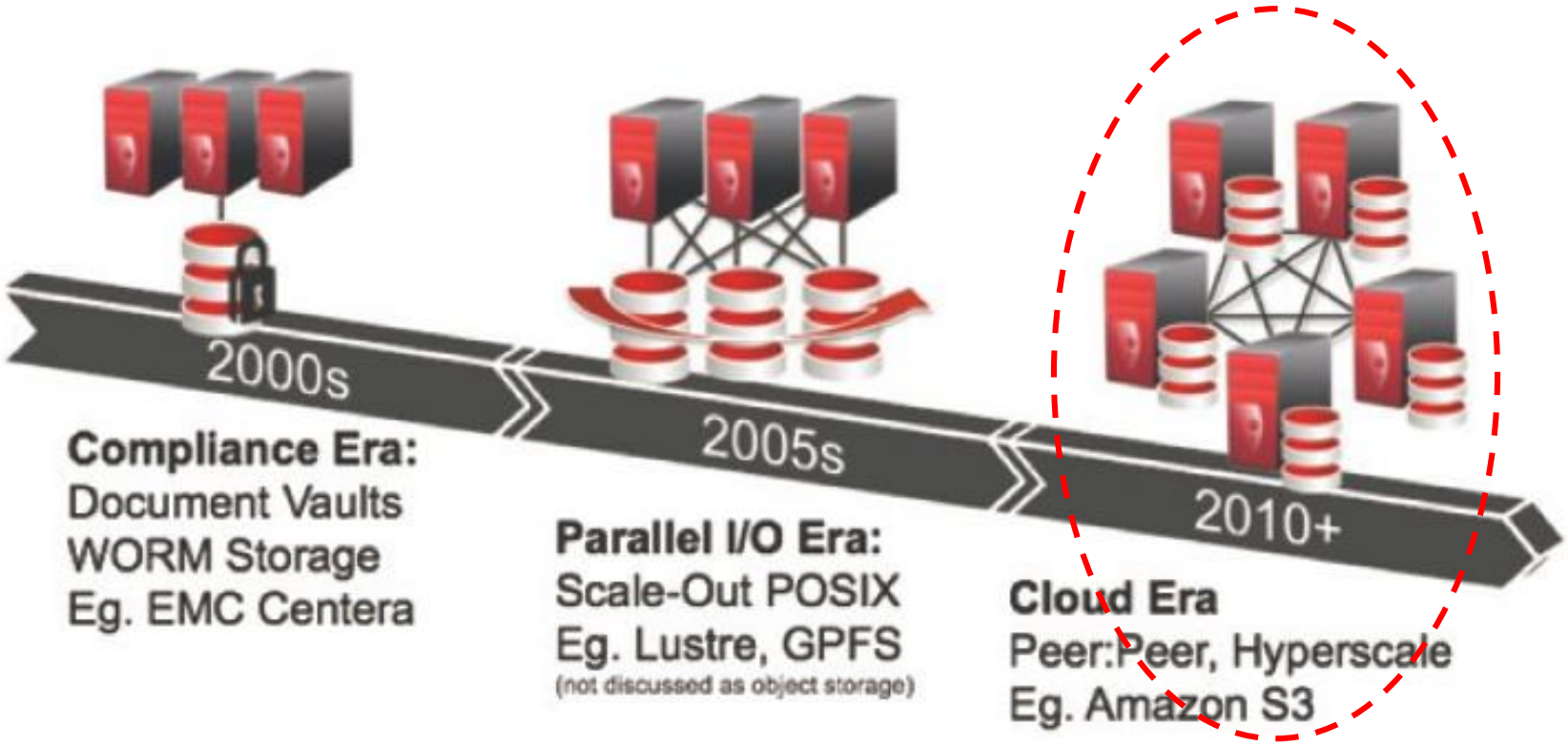
Key Design Issue : Scalability

- I/O throughput
  - How to avoid bottlenecks
- Metadata scalability
  - How can 10,000's of clients work on files in same folder
- Cluster Recovery
  - If sth fails, how can transparent recovery happen
- Management
  - Adding, removing, replacing, systems; data migration & backup

Reference

- Lustre: A SAN File System for Linux
  - http://lustre.org/documentation/
- Several presentation materials from Dr. Peter J. Braam

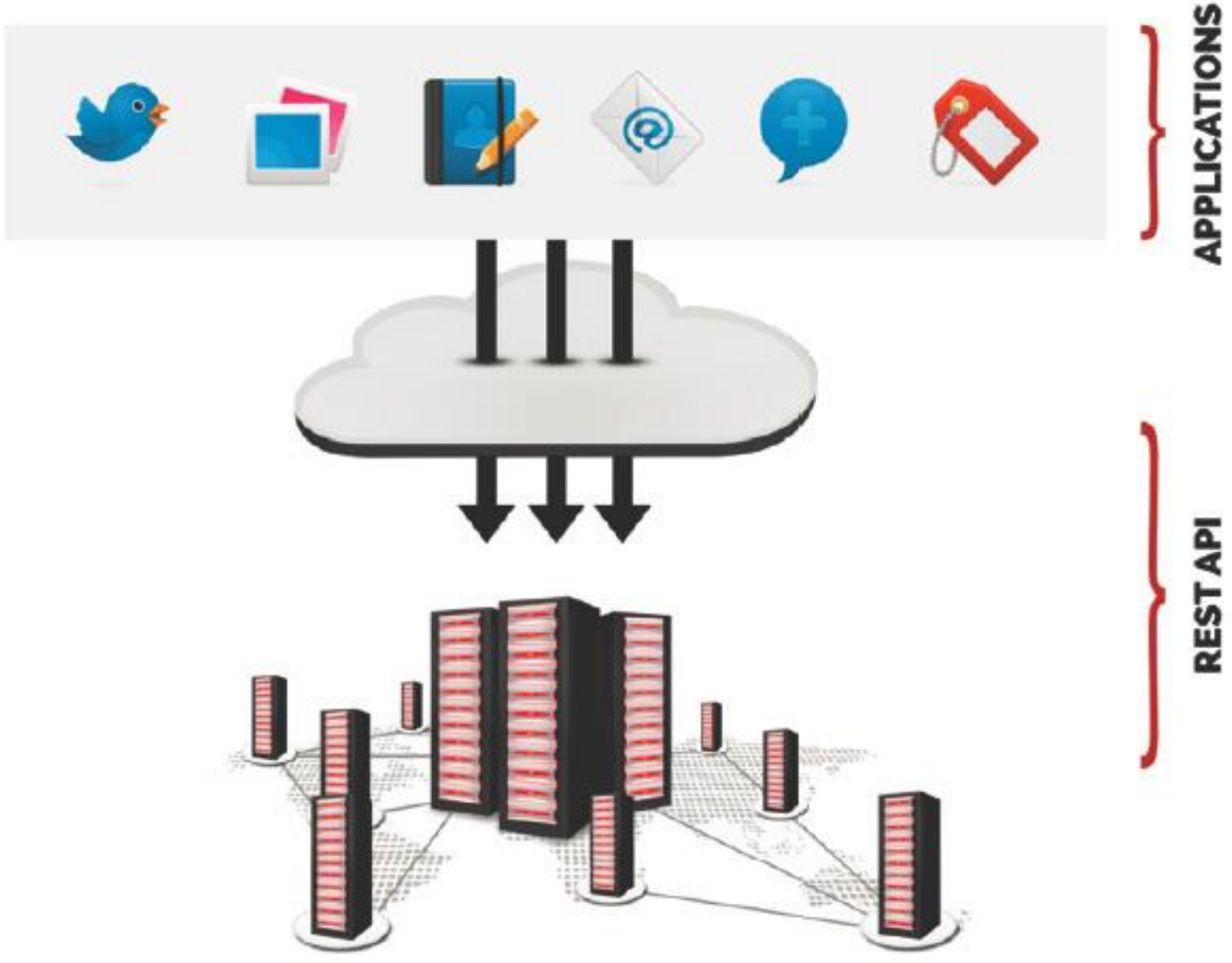
The Current Generation of Object Storage



Cloud Storage, Storage Clouds, Object Storage

- **Cloud Storage** is the storage used for Compute Cloud infrastructures
  - Compute Clouds are very IOPS intensive and usually **block storage** is used in these applications
- **Storage Clouds** are “storage in the cloud”, whether public or private
  - Storage Clouds are simply storage capacity that is made available through the Internet
  - Most of today’s storage clouds use **object storage** technologies

Scale out object storage with simple REST API



REST API's

- REST stands for Representational State Transfer
- It is a software architecture that is used for distributed application environments
- REST API's have become the predominant interface for cloud applications to connect to the cloud
- For storage-centric cloud applications, a REST API is the interface between the application and the object storage platform
- PUT GET DELETE…

Current Object Storage Summary

- Data is stored as objects in one large, scalable pool of storage
- Objects are stored with metadata – information about the object
- An Object ID is stored, to locate the data
- REST is the **standard interface**, simple commands used by applications
- Objects are immutable; edits are saved as a new object

对象存储与键值存储

◆ 相同点

- 对象ID 类似于Key（任意字符串）
- 数据可以具有任意大小

◆ 不同点

- 对象存储还允许将一组有限的属性（元数据）与每个(数据)对象相关联
- 对象存储针对大量数据（数百兆甚至千兆字节）进行了优化，而对于键值存储，其值则相对较小（千字节）
- 对象存储通常提供较弱的一致性保证，例如最终的一致性，而键值存储则提供较强的一致性。

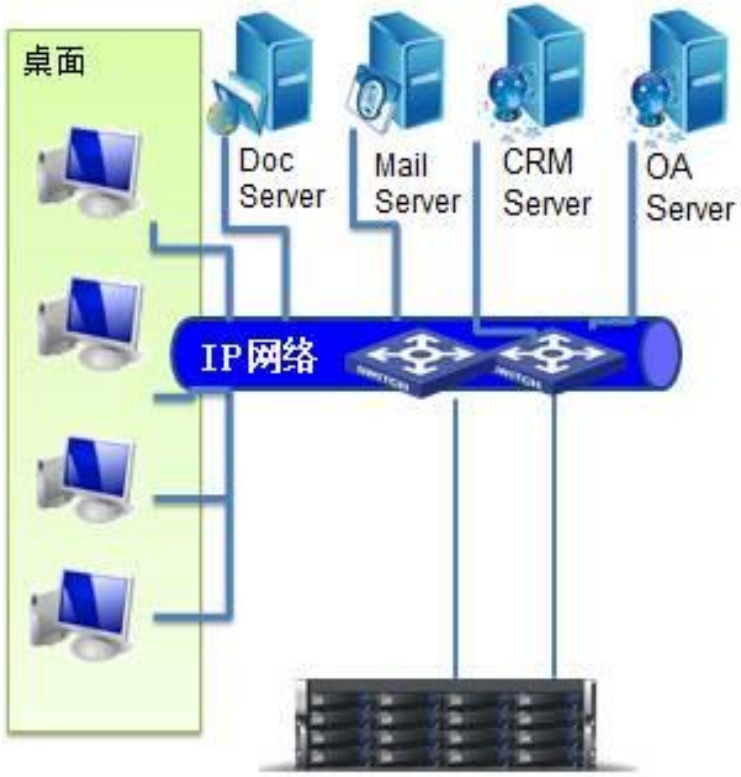
块、文件和对象

- **块存储**（DAS/SAN）
  - FC、iSCSI协议、NVMe-oF
  - 专用系统中，高读写性能和高可靠性
  - 一套存储只服务一个应用系统，例如如交易系统，计费系统。典型行业如金融、制造、能源、电信等

块、文件和对象

口 文件存储（NAS）

- ✓NFS/CIFS协议，IP网络
- ✓兼顾多个应用和更多用户访问，同时提供方便的数据共享手段
- ✓中小企业市场，CRM系统、SCM系统、OA系统、HPC、AIGC等



块、文件和对象

- **对象存储**（Object）
  - OSD，HTTP协议
  - 互联网或者公网
  - 海量数据，高并发访问
  - 常见的适配应用如网盘、媒体娱乐、医疗PACS、气象、归档等数据量超大而又相对“冷数据”和非在线处理的应用类型

块、文件和对象

块存储　文件存储　对象存储

Throughput easy, latency hard

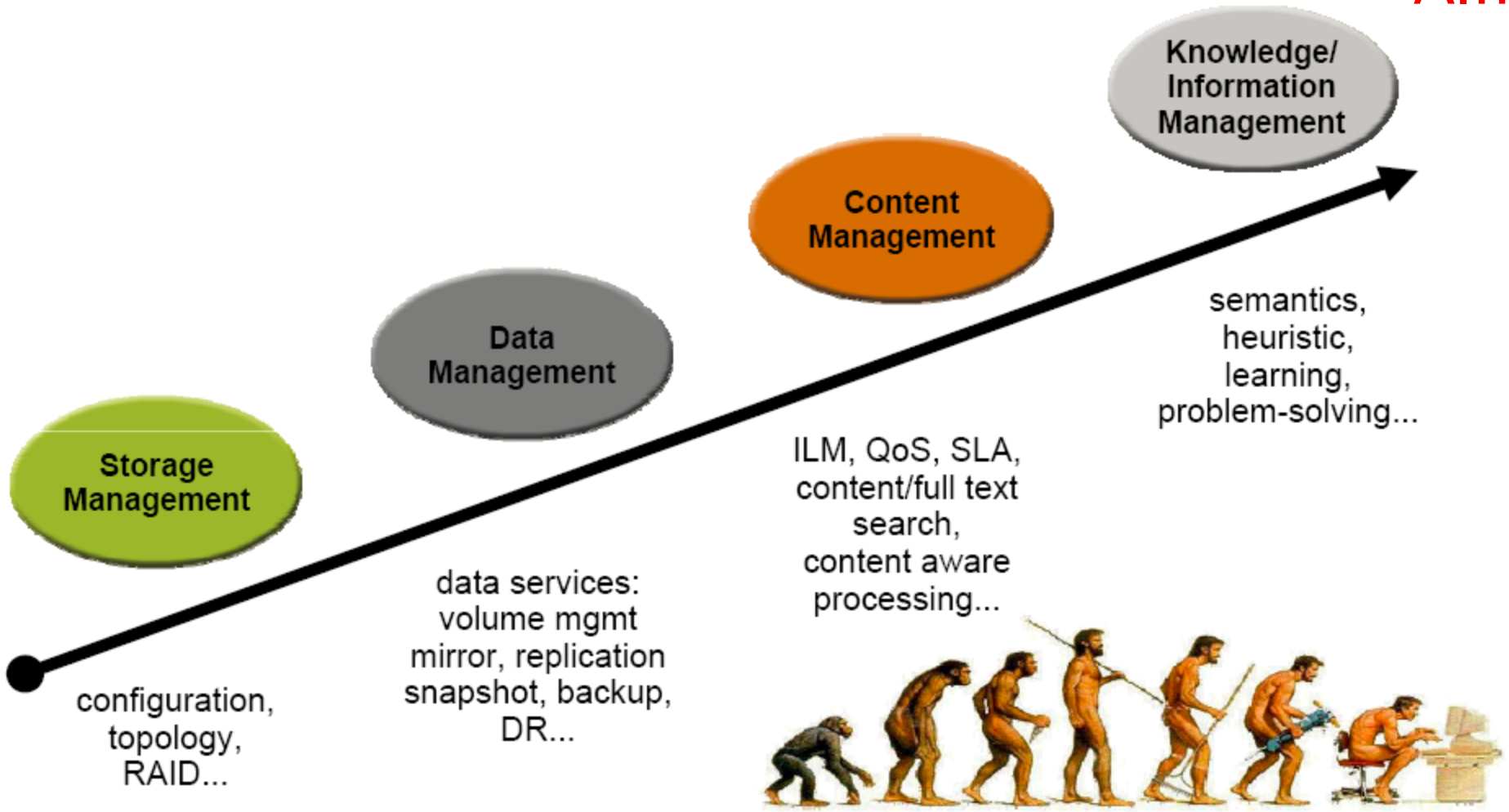
Throughput is easy

Latency is hard

Throughput is an engineering problem, latency is a physics problem!

The Evolution of Data Processing

AI…



Moving Data to the Processor is Costly

One floating-point calculation

17 picojoules

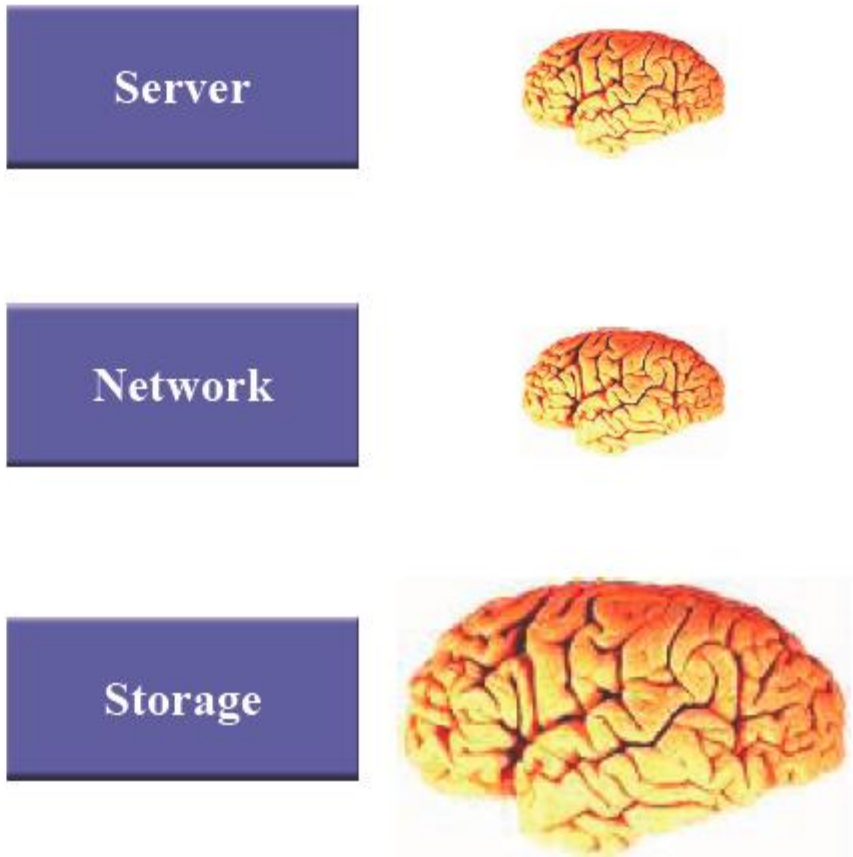
Moving data from DRAM to CPU

17,000 picojoules

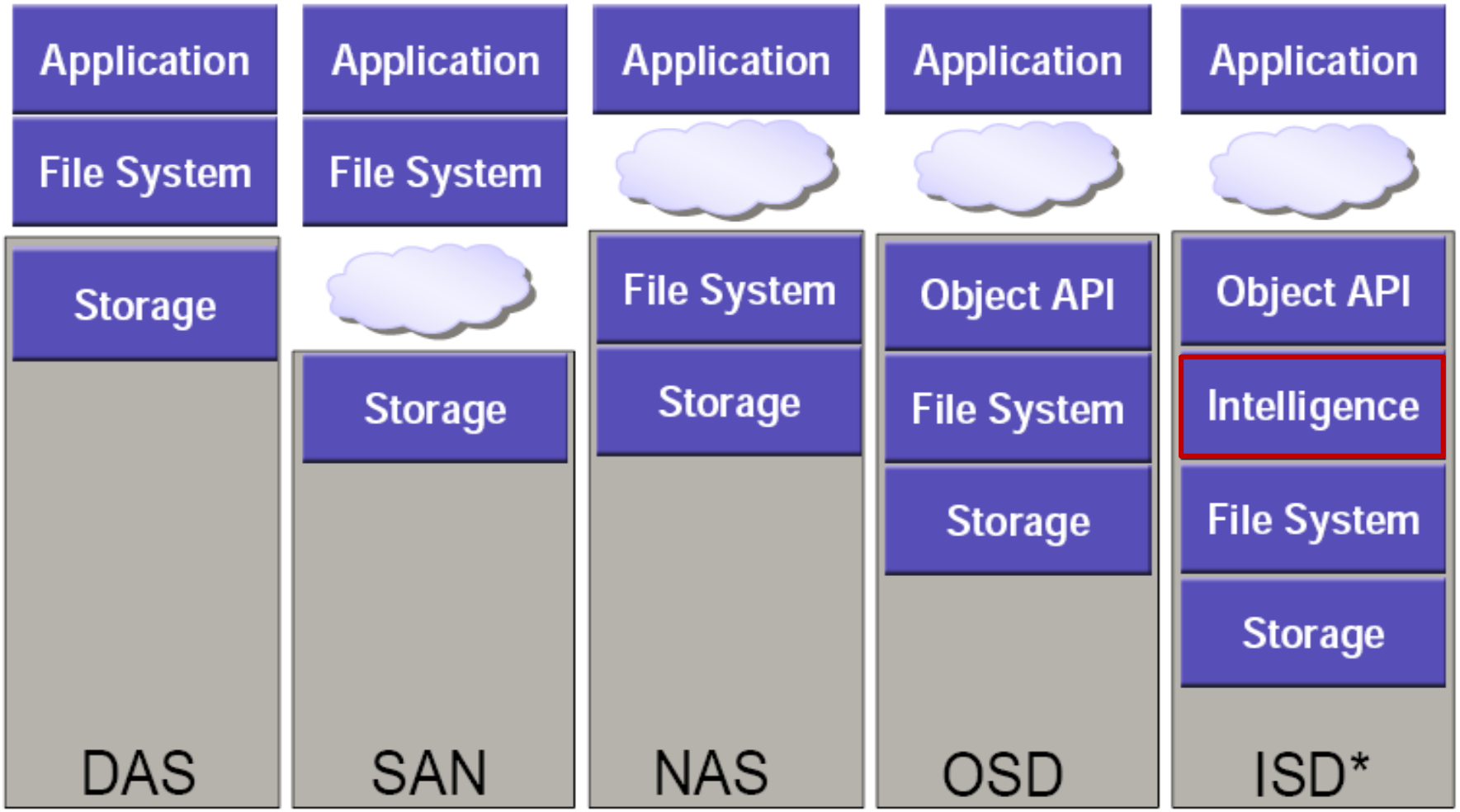
Opportunities for **1000x improvement** are increasingly rare

Migration of Storage Application

- Process the data where it lives...

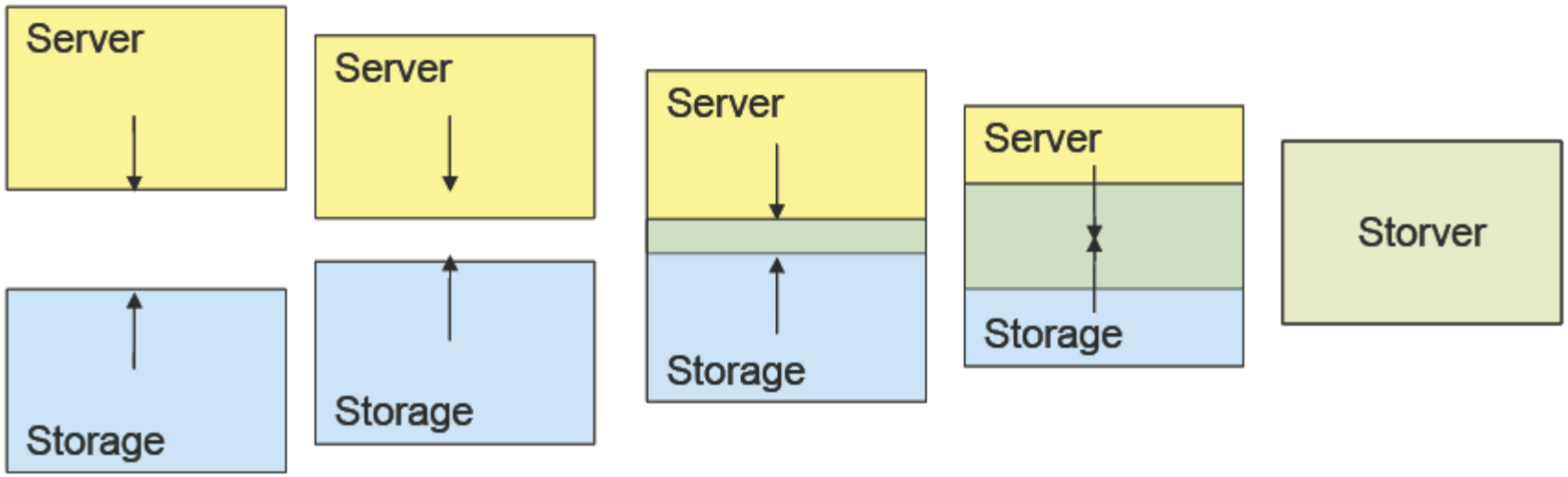


The Evolution of Storage



The Vertical Consolidation

- Storage and server
- Migration of data processing applications
- **No I/O is best I/O**



近数据处理
(Near Data Processing)

近数据处理

- 近数据处理技术作为一种存算一体思想的新型算力，有望解决冯·诺依曼架构下存储墙等问题

根据计算与存储的距离远近分类：

近存计算
Processing Near Memory, PNM

存内处理
Processing In Memory, PIM

存内计算
Computing In Memory, CIM

NDP-近存计算

- 近存计算是一种通过将计算逻辑靠近存储设备实现高效数据处理的计算机架构技术，旨在突破传统冯·诺依曼架构的瓶颈。其核心是通过减少数据传输距离和频率来降低延迟并提升能效。

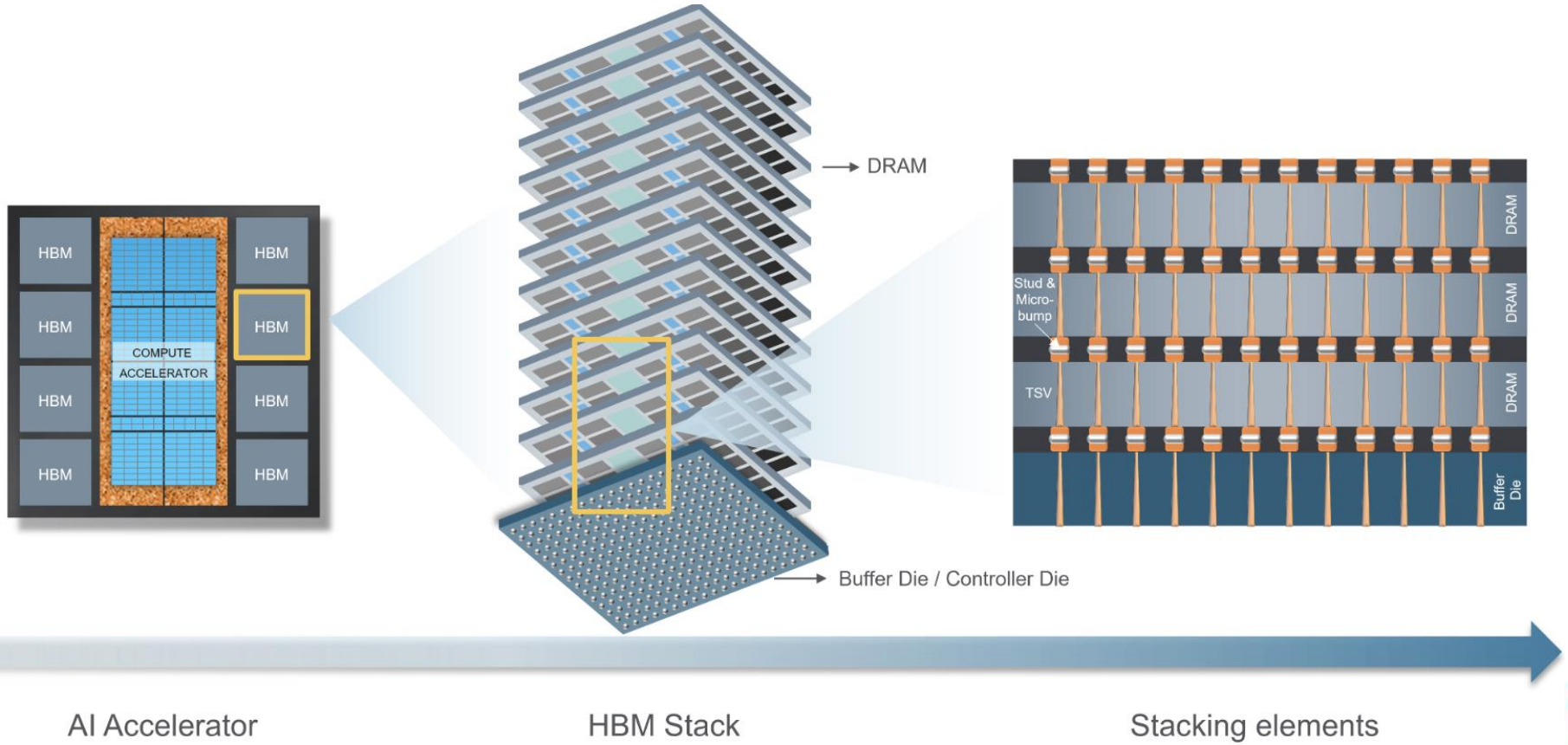
  --通过芯片封装和板卡组装等方式，在**内存或存储设备附近集成计算单元**，增加访存带宽、减少数据搬移

- 近存计算仍是**存算分离架构**，计算操作由位于存储外部、**独立的计算单元**完成，其技术成熟度较高，主要包括**存储上移、计算下移**两种方式

NDP-存储上移

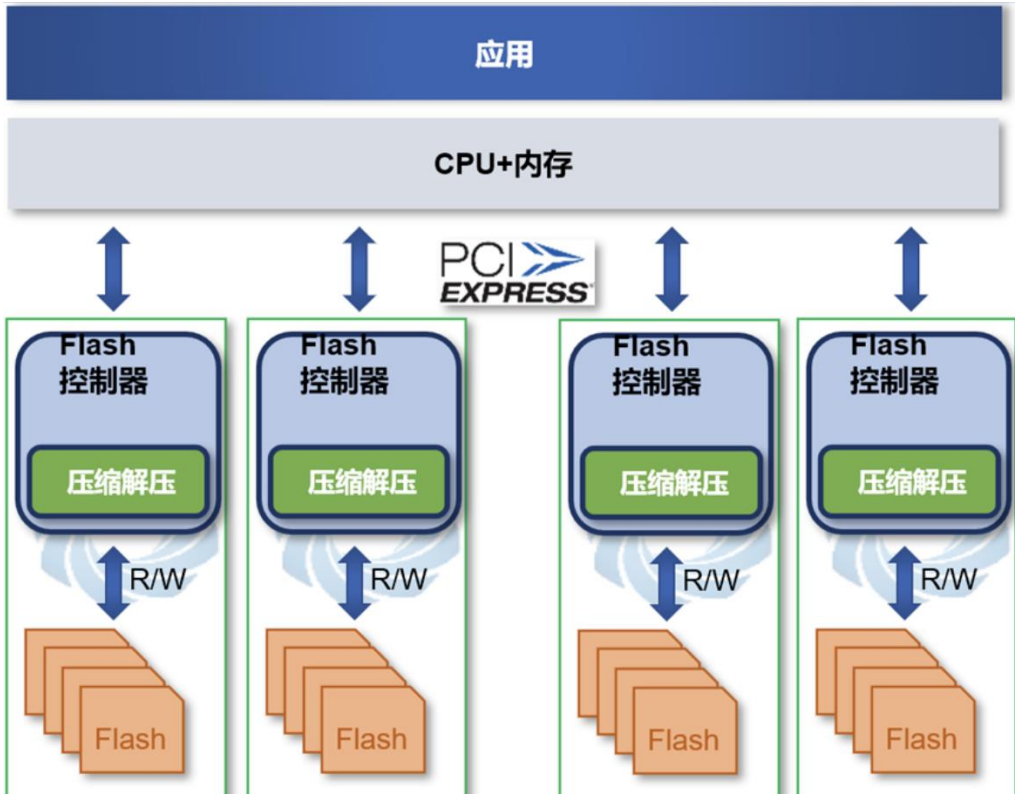
- **存储上移**

用先进封装技术将存储器向处理器(XPU)靠近，增加计算和存储间的链路数量，提供更高访存带宽。
典型代表：**HBM** (**H**igh **B**andwidth **M**emory)，将内存颗粒通过硅通孔多层堆叠实现存储容量提升，同时基于硅中介板的高速接口与计算单元互联提供**高带宽**存储服务



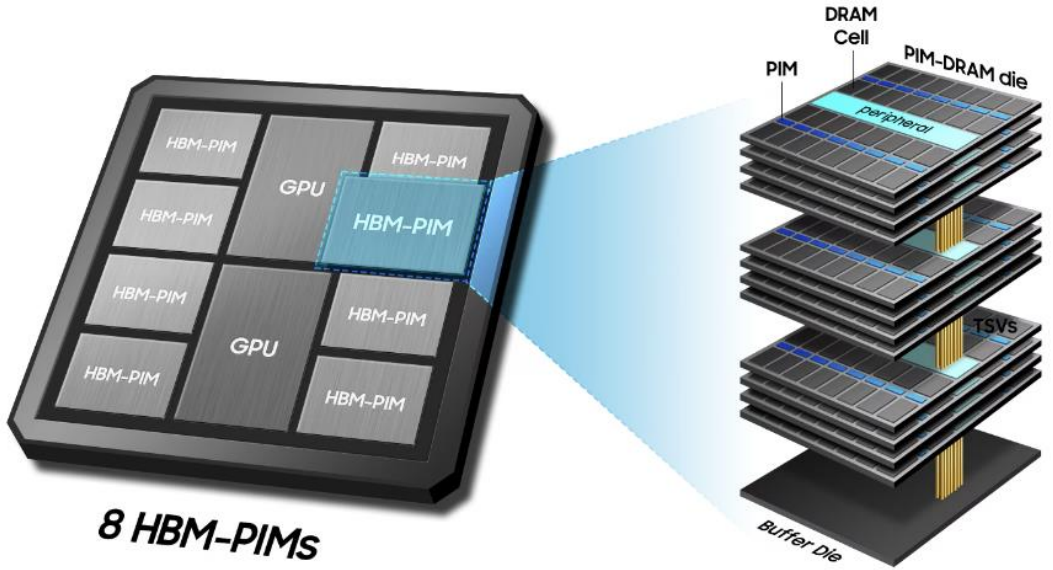
NDP-计算下移

- **计算下移**
  - 采用集成技术将数据处理能力卸载到存储器，由近端处理器进行数据处理，有效减少存储器与远端处理器的数据搬移开销。典型的方案为可计算存储**CSD** (**C**omputational **S**torage **D**rives)
  - CSD在SSD内部引入的计算引擎功能，主要功能包括：压缩和解压缩、加密和解密等。e.g., 压缩和解压缩，会在主控将数据写到后端NAND颗粒之前对要写入的数据进行压缩；反过来读取的时候从NAND颗粒读出的数据再进行解压缩，节省空间，降低损耗



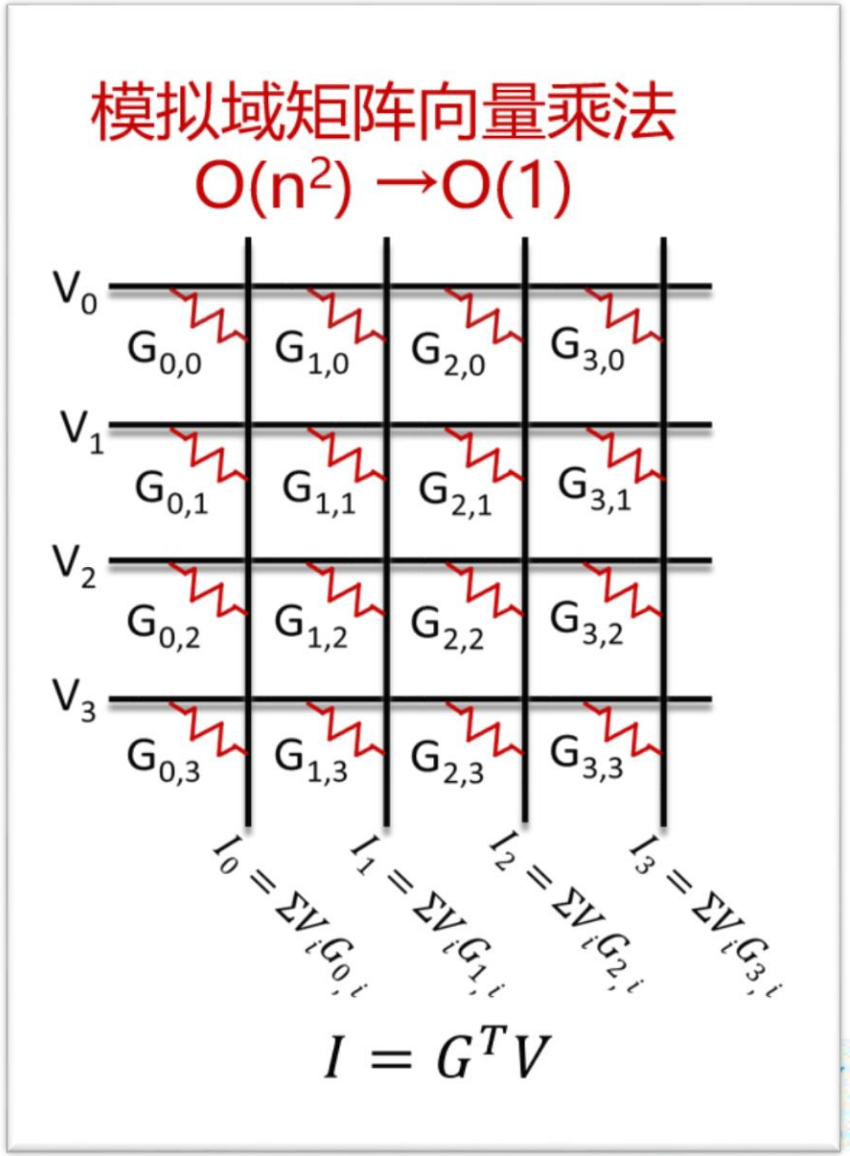
NDP-存内处理

- 存内处理是在芯片制造的过程中，将**存和算集成在同一个晶粒中**，使存储器本身具备了一定算的能力。存内处理**本质上仍是存算分离**，相比于近存计算，“存”与“算”**距离更近**
- 当前存内处理方案大多在内存芯片中实现部分数据处理，较为典型的产品形态为HBM-PIM和PIM-DIMM，在DRAM Die中内置处理单元，提供大吞吐低延迟片上处理能力，可应用于语音识别、数据库索引搜索、基因匹配等场景



NDP-存内计算

- 存内计算即**狭义的存算一体**，在芯片设计过程中，不再区分存储单元和计算单元，真正实现 **存算融合**
- 存内计算是计算新范式的研究热点，其本质是利用不同存储介质的物理特性，对存储电路进行重新设计使其同时具备计算和存储能力，直接消除“存”“算”界限，使计算能效达到数量级的提升
- 典型应用场景是AI提供向量矩阵乘法的算子加速，例如卷积神经网络CNN，循环神经网络RNN....



NDP总结

- 目标：将计算资源放置在数据存储的更近处
- 优势：减少数据移动开销、降低延迟、提升系统性能等

主动对象存储

- 传统存储系统被动响应服务请求
- 对象具有智能性

智能的系统能够提供主动服务

存储组织结构对比

传统模式

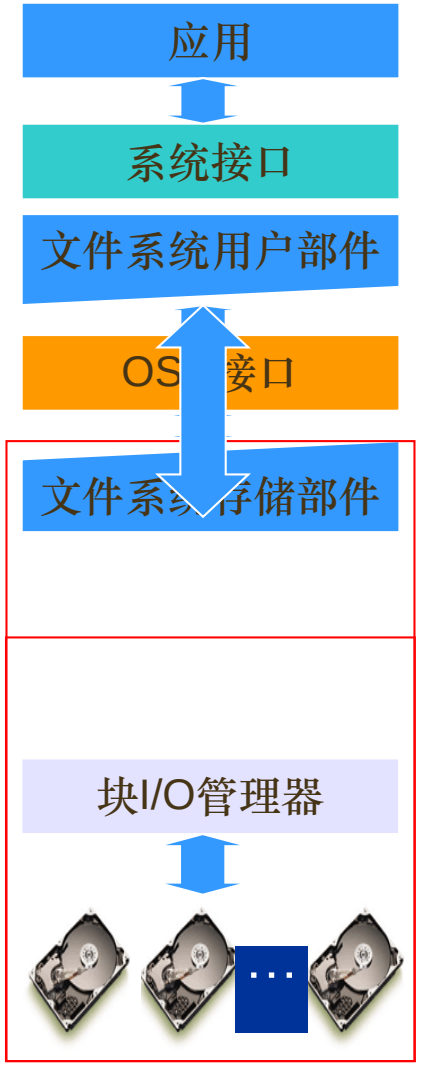
主机

存储系统

效率低
管理复杂

- 只含数据
- 主机组织和管理
- 被动响应

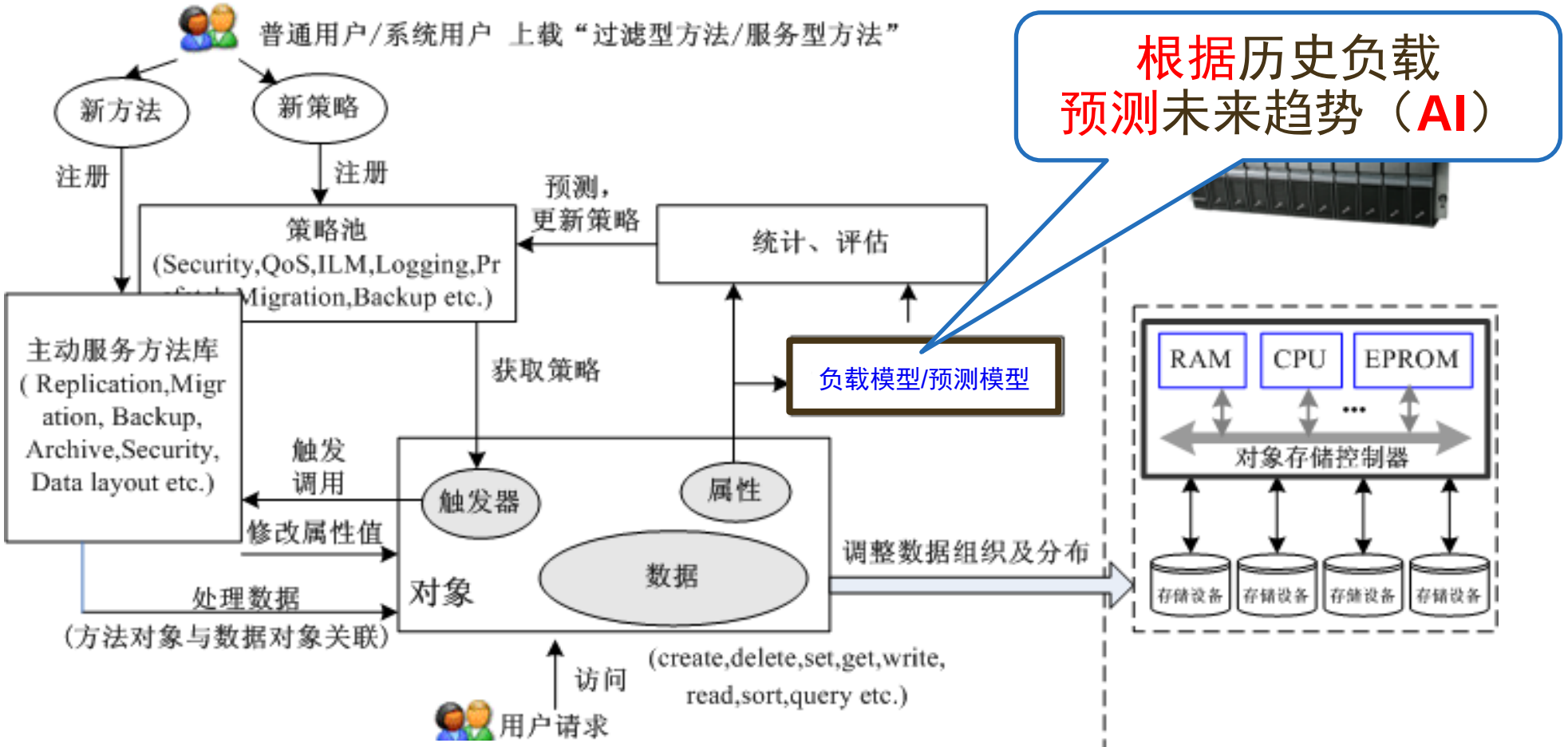
主动对象模式



性能高
管理简单

- 含“数据、属性”和“**操作**”
- 自组织和自管理
- **主动服务**

主动对象存储服务机制



存储主动服务

存储节点

主动对象特点

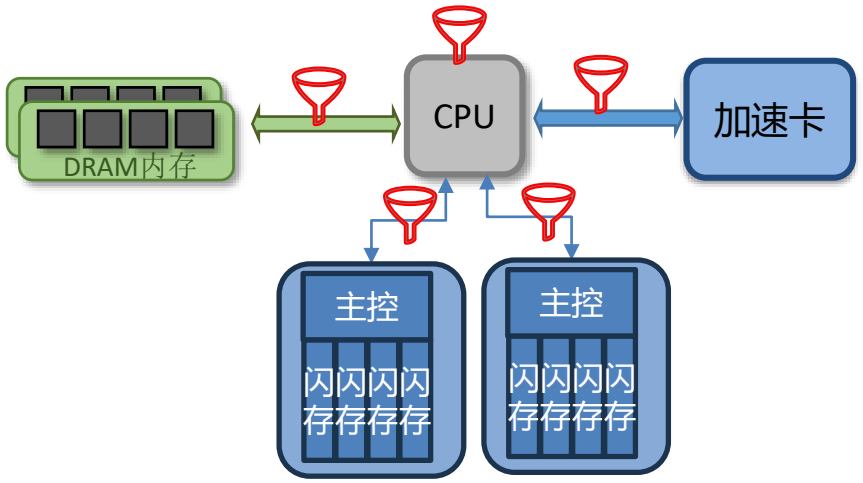
- 自我学习和策略触发机制，实现存储主动服务
- 自我组织与管理，自我优化调节，使系统整体性能最佳
- 利用对象“封闭”特性，使系统具有安全性
- 提高系统可靠性，在故障出现时实现快速恢复

自组织、自优化、自愈

主动对象实例：近数据处理

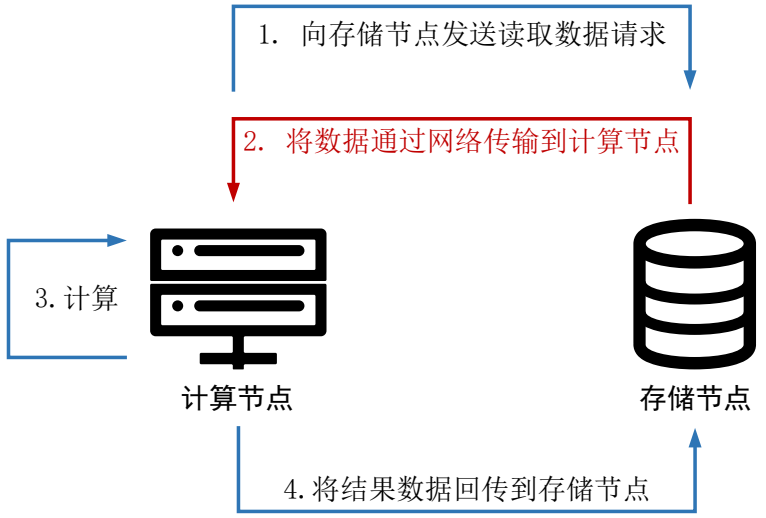
- 数据总量爆炸式增长，数据搬运成为瓶颈

节点内



- 数据先从存储设备搬运至内存，再由CPU、加速卡处理
- 数据处理集中于CPU、加速卡，导致高占用率和性能瓶颈
- 数据搬运造成大量延迟和能耗开销

节点间

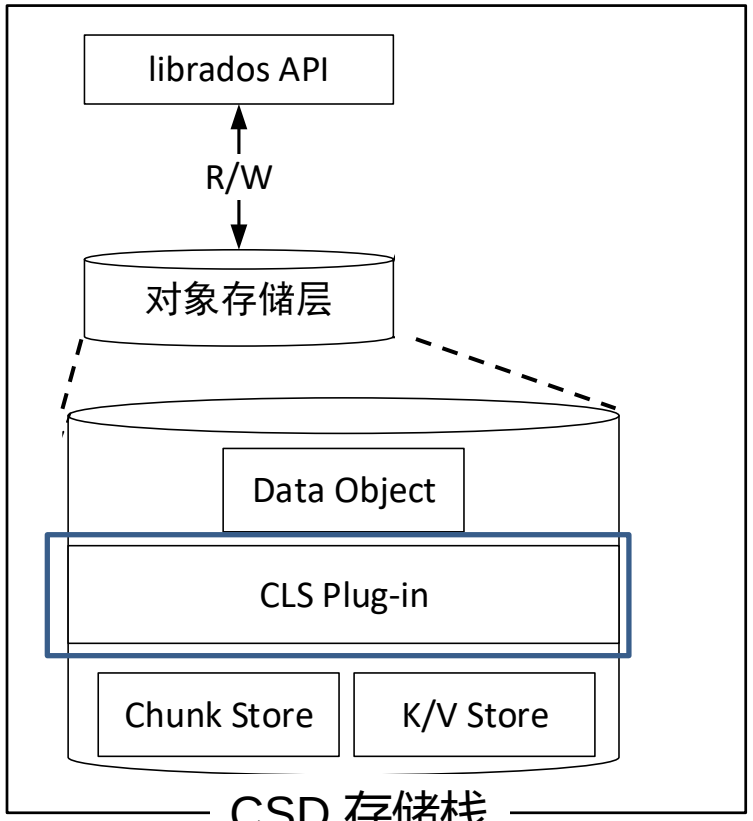


- 数据先从存储节点搬运至计算节点，再由计算节点处理
- 数据处理集中于计算节点，无法利用存储结点的计算能力
- 数据搬运占用大量网络带宽资源

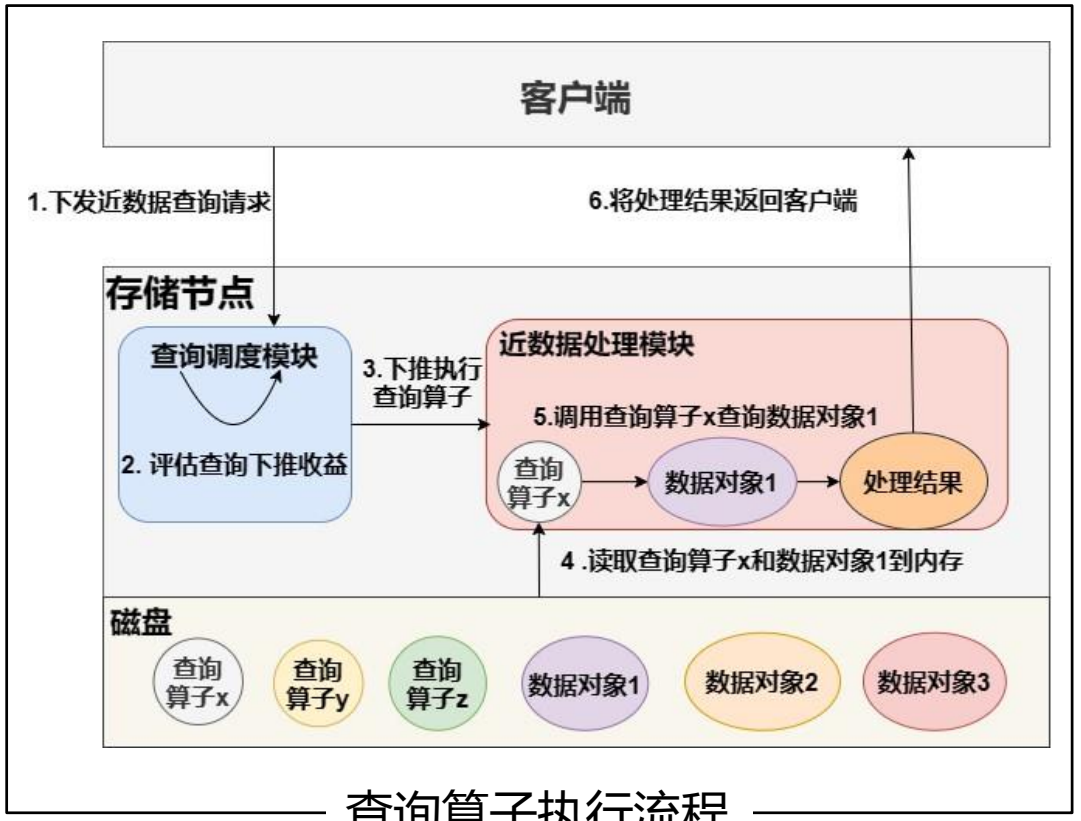
亟需近数据处理，减少数据搬运，提高处理效率

基于收益评估的算子动态卸载方法

- 问题：如何在缓解集群网络压力同时提高查询算子性能?
- 方法：建立基于系统运行信息的算子收益评估模型，实现算子动态卸载至存储节点的方法



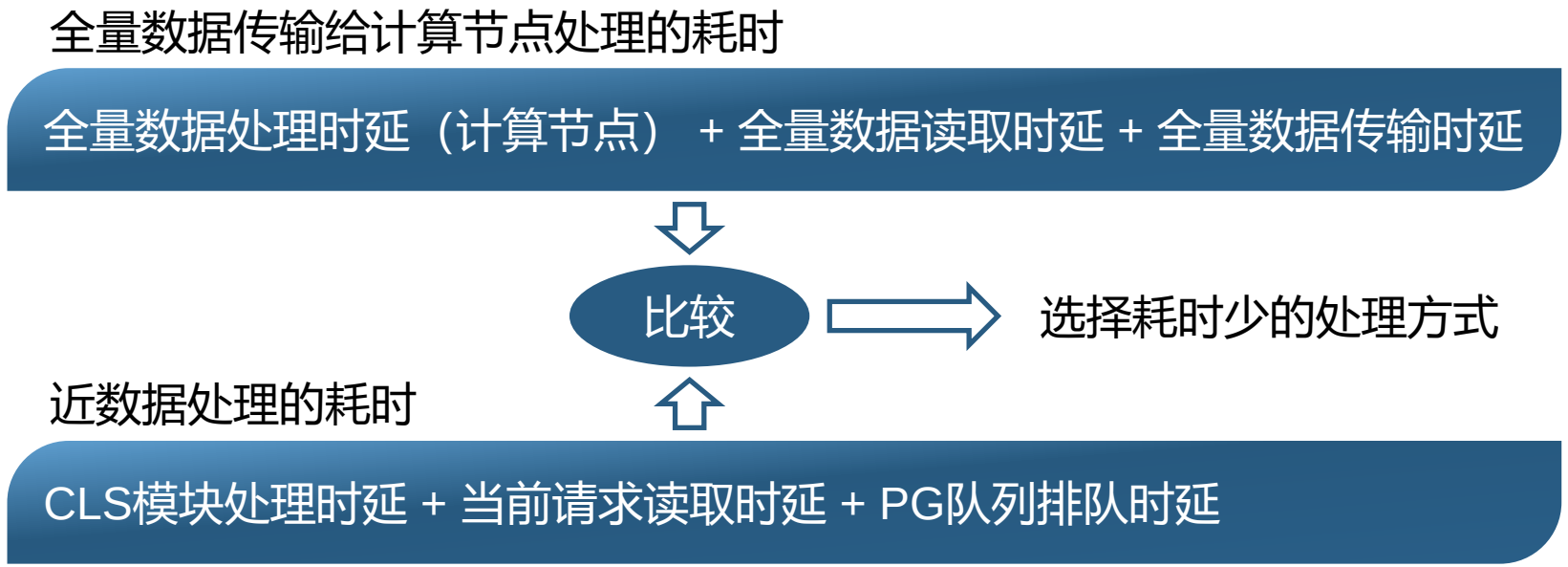
CSD 存储栈



查询算子执行流程

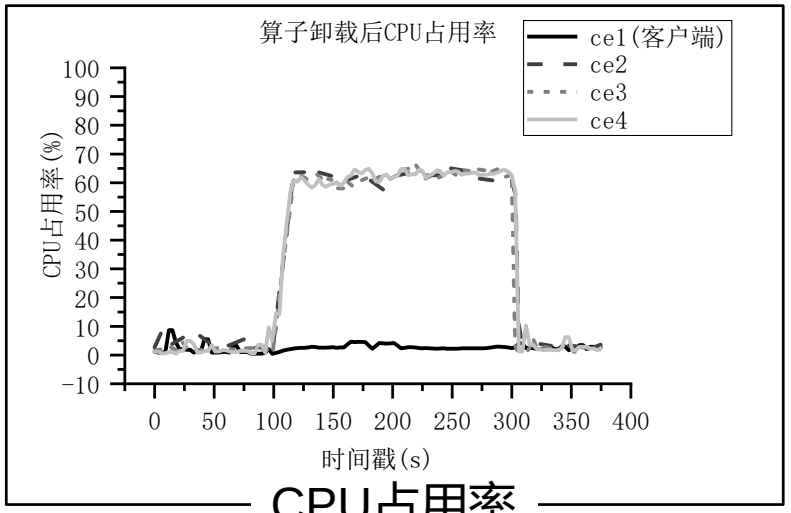
基于收益评估的算子动态卸载方法

- 算子卸载收益评估
  - 根据系统运行时信息动态判断算子卸载收益
  - 判断方式：预估算子交给计算节点处理和近数据处理耗时
  - 选择耗时更少的处理方式

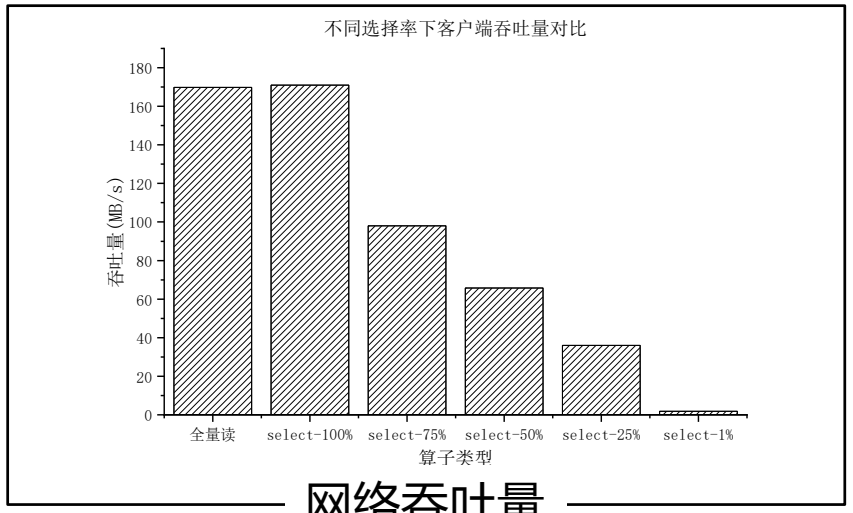


基于收益评估的算子动态卸载方法

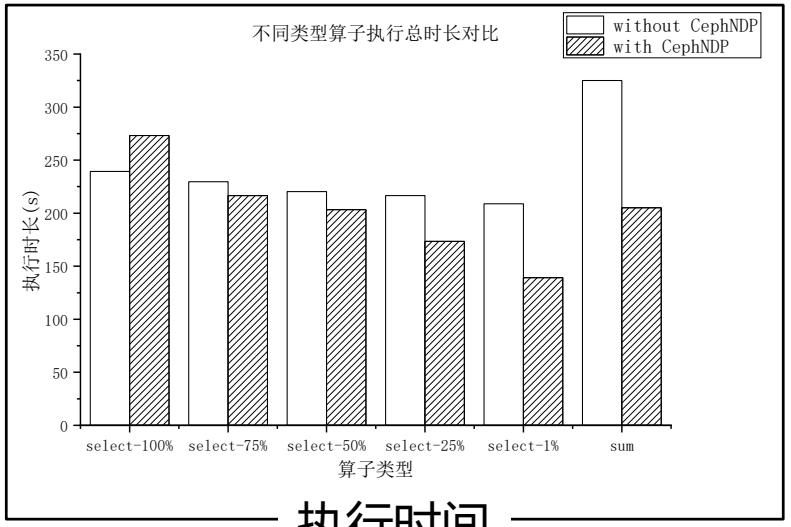
- 测试结果



CPU占用率



网络吞吐量



执行时间

相比未卸载算子的情况，查询操作时集群网络流量降低99%，计算节点CPU消耗减少99%，执行时间减少33%

谢 谢!