Distributed file systems

Viet Trung Tran
Department of Information Systems
School of Information and Communication Technology
Hanoi university of science and technology
trungtv@soict.hust.edu.vn

File systems

NTFS

hadoop HDFS

Ext4 File System

NFS NETWORK FILE SYSTEM

PVFS

The Google File System

By Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
(Presented at SOSP 2003)

Definitions

- **Filenames**
  - File Identity
- **Directories (folders)**
  - Group of files in separate collections
- **Metadata**
  - Creation time, last access time, last modification time
  - Security information (Owner, Group owner)
  - **Mapping file to its physical location of file (e.g. location in storage devices)**

File system overview

- Computer file
  - A resource for storing information
  - Durable, remained available for access
  - Data: sequences of bits
- File system
  - Control how **computer file** are stored and retrieved
  - Main operators: READ, WRITE (offset, size), CREATE, DELETE

Local vs. distributed file systems

Local file systems

NTFS

Ext4
File System

hadoop
HDFS

NFS
NETWORK FILE SYSTEM

The Google File System

By Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
(Presented at SOSP 2003)

PVFS

Distributed file system

- File system
  - Abstraction of storage devices
- Distributed file system
  - Available to **remote processes in distributed systems**
- Benefits
  - File sharing
  - Uniform view of system from different clients
  - Centralized administration

DFS@wikipedia

- In computing, a **distributed file system** or **network file system** is any file system that allows access to files from multiple hosts sharing via a computer network.[1] This makes it possible for multiple users on multiple machines to share files and storage resources.
- The client nodes do not have direct access to the underlying block storage but interact over the network using a protocol. This makes it possible to restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.

Goals

- **Network (Access)Transparency**
  - Users should be able to access files over a network as easily as if the files were stored locally.
  - Users should not have to know the physical location of a file to access it.
- Transparency can be addressed through naming and file mounting mechanisms

Components of Access Transparency

- Location Transparency: file name doesn't specify physical location (Ch. 1)
- Location Independence: files can be moved to new physical location, no need to change references to them. (A name is independent of its addresses – see Ch. 5)
- Location independence → location transparency, but the reverse is not necessarily true.

Goals

- Availability: files should be easily and quickly accessible.
- The number of users, system failures, or other consequences of distribution shouldn't compromise the availability.
- Addressed mainly through replication.

Architectures

- Client-Server
  - Traditional; e.g. Sun Microsystem Network File System (NFS)
  - Cluster-Based Client-Server; e.g., Google File System (GFS)
- Symmetric
  - Fully decentralized; based on peer-to-peer technology
  - e.g., Ivy (uses a Chord DHT approach)

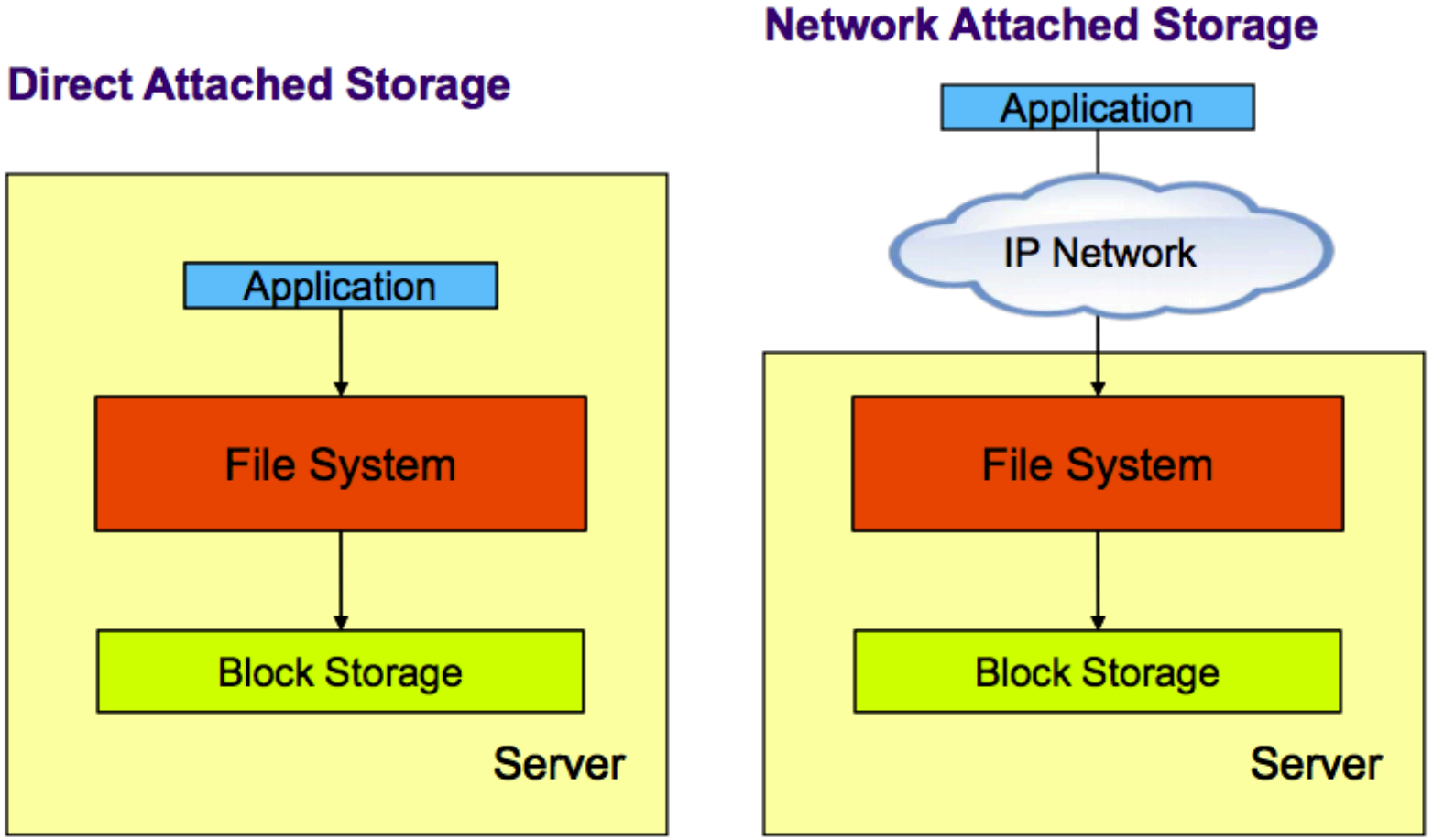
Client-Server Architecture

- One or more machines (file servers) manage the file system.
- Files are stored on disks at the servers
- Requests for file operations are made from clients to the servers.
- Client-server systems centralize storage and management; P2P systems decentralize it.

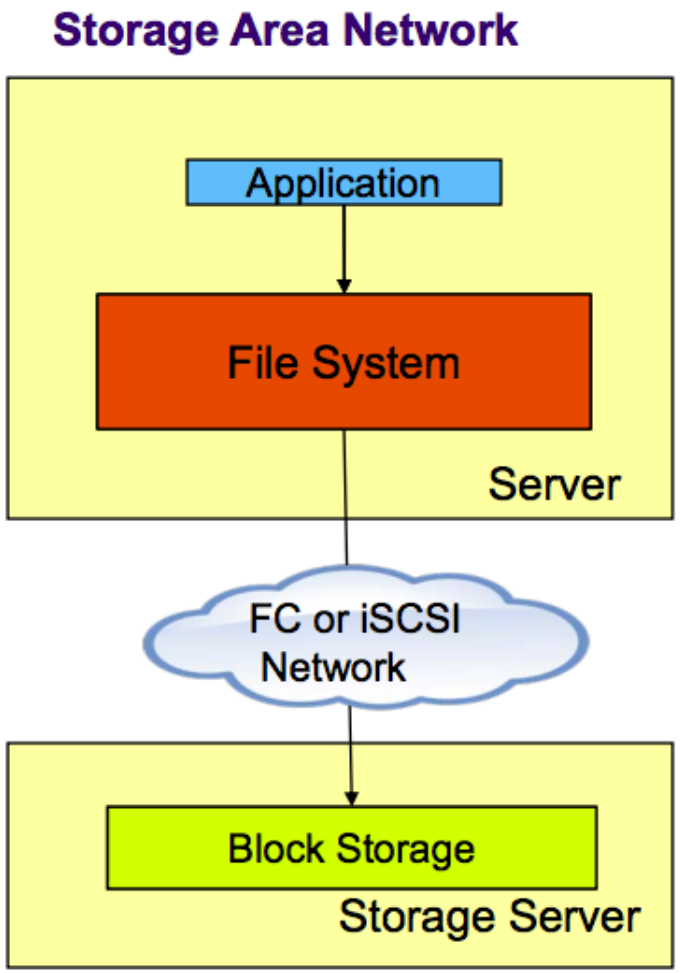
The Evolution of Storage

- Direct attached storage (DAS)
- Network attached storage (NAS)
- Storage area network (SAN)
- Combined technologies

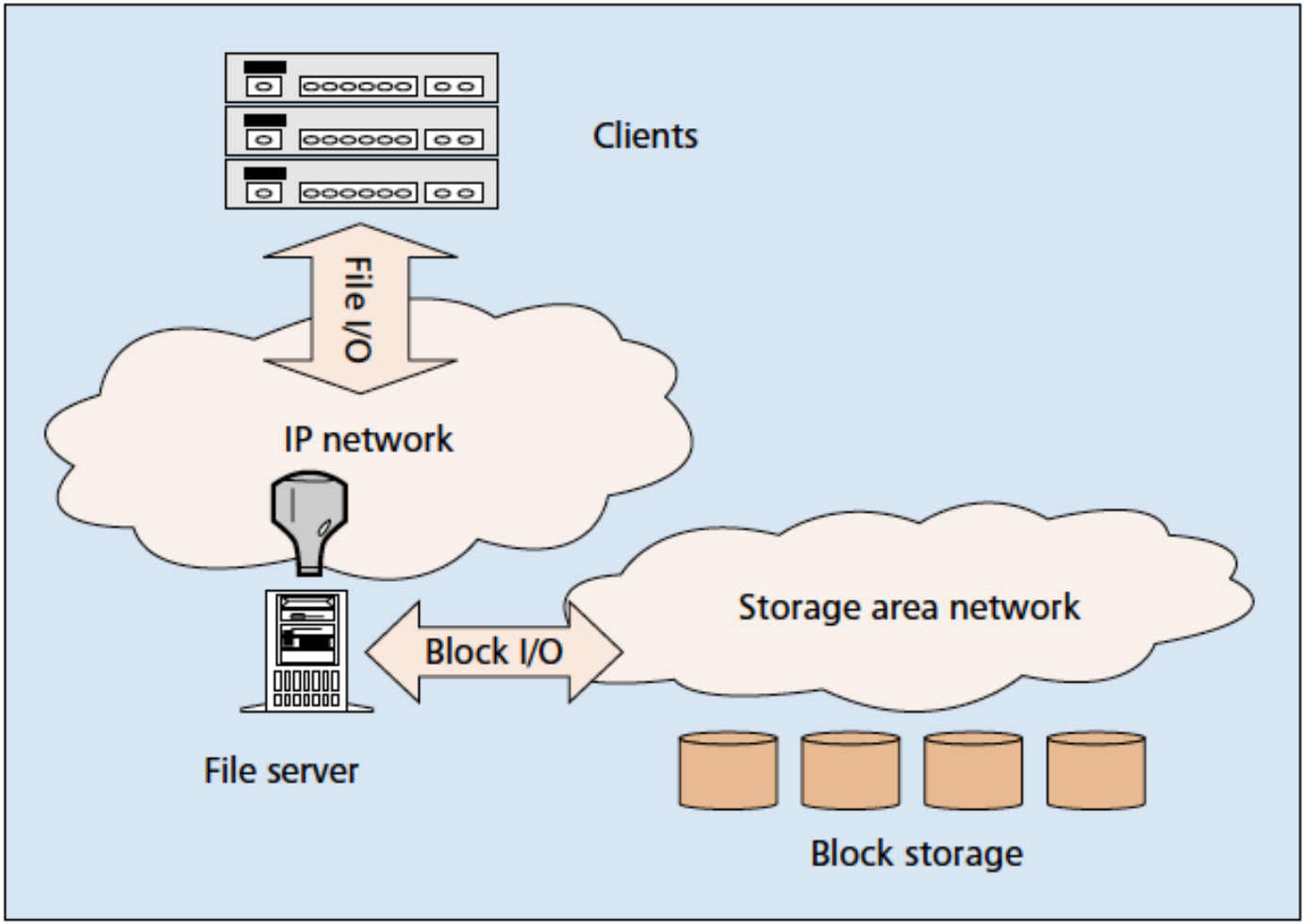
From DAS to NAS



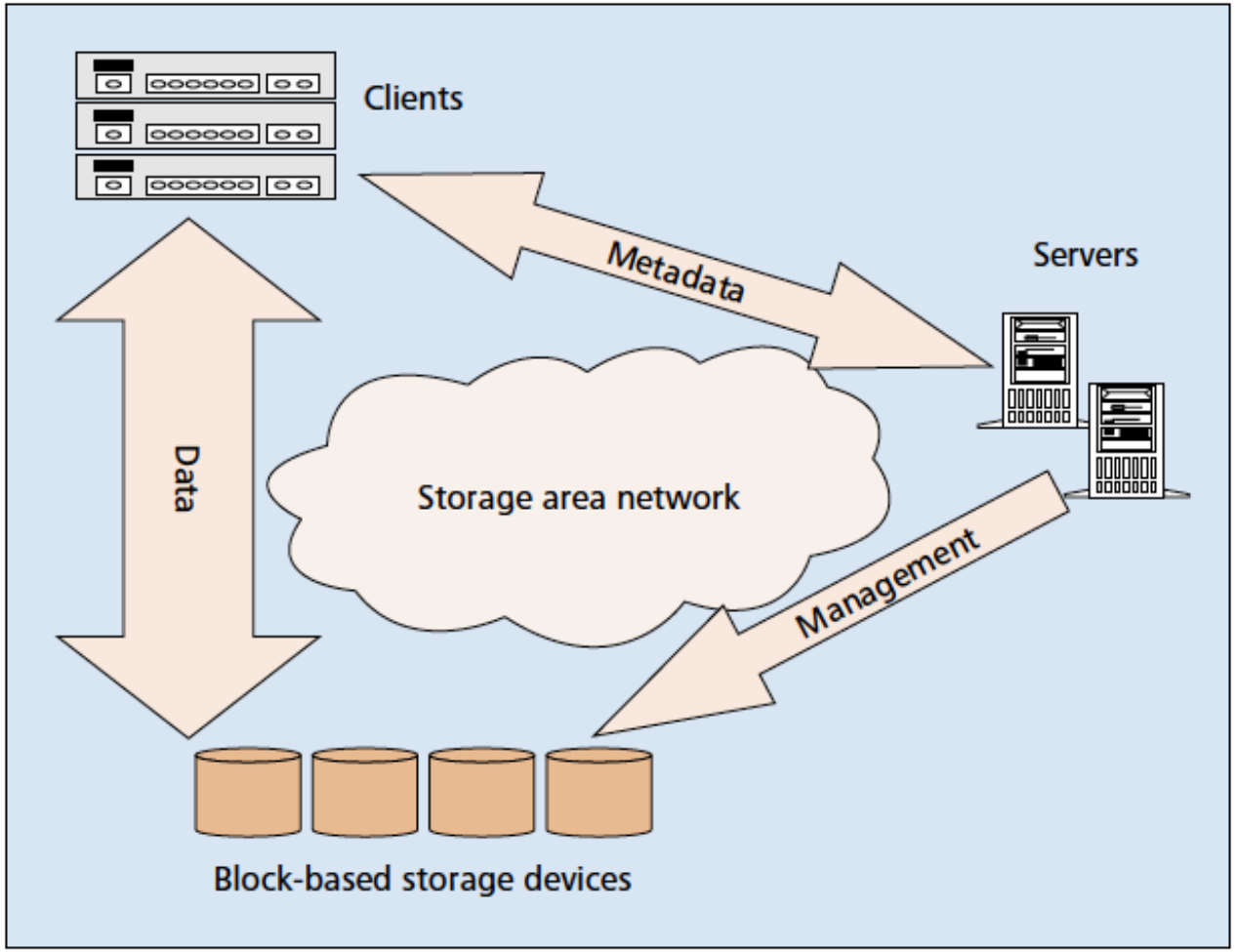
To SAN


Storage Area Network
Application
File System
Server
FC or iSCSI
Network
Block Storage
Storage Server

NAS file system



SAN file system



Object storage device (OSD)

- OSDs hold objects rather than blocks, which are like files in a simple file system
  - Objects are identified by a 64 bit Object ID (OID)
  - Objects are dynamically created and freed
  - Objects are variable length
- OSDS manage space allocation of objects
  - **OSD are not dump as conventional storage disks**

OSD vs. disk

- Disk storage
  - Fixed array of blocks
- Disk operations
  - Read block, write block, format
- Disk security
  - Zoning and LUN Masking of entire disk
- Transport
  - FC SCSI and iSCSI

- OSD storage
  - Many objects of variable size
- OSD operations
  - Read object, write object, create object, list objects, etc.
- OSD security
  - On a per object basis using Capabilities
- Transport
  - FC SCSI, iSCSI, RPC

OSD advantages

- Data sharing
- Security
- Intelligence

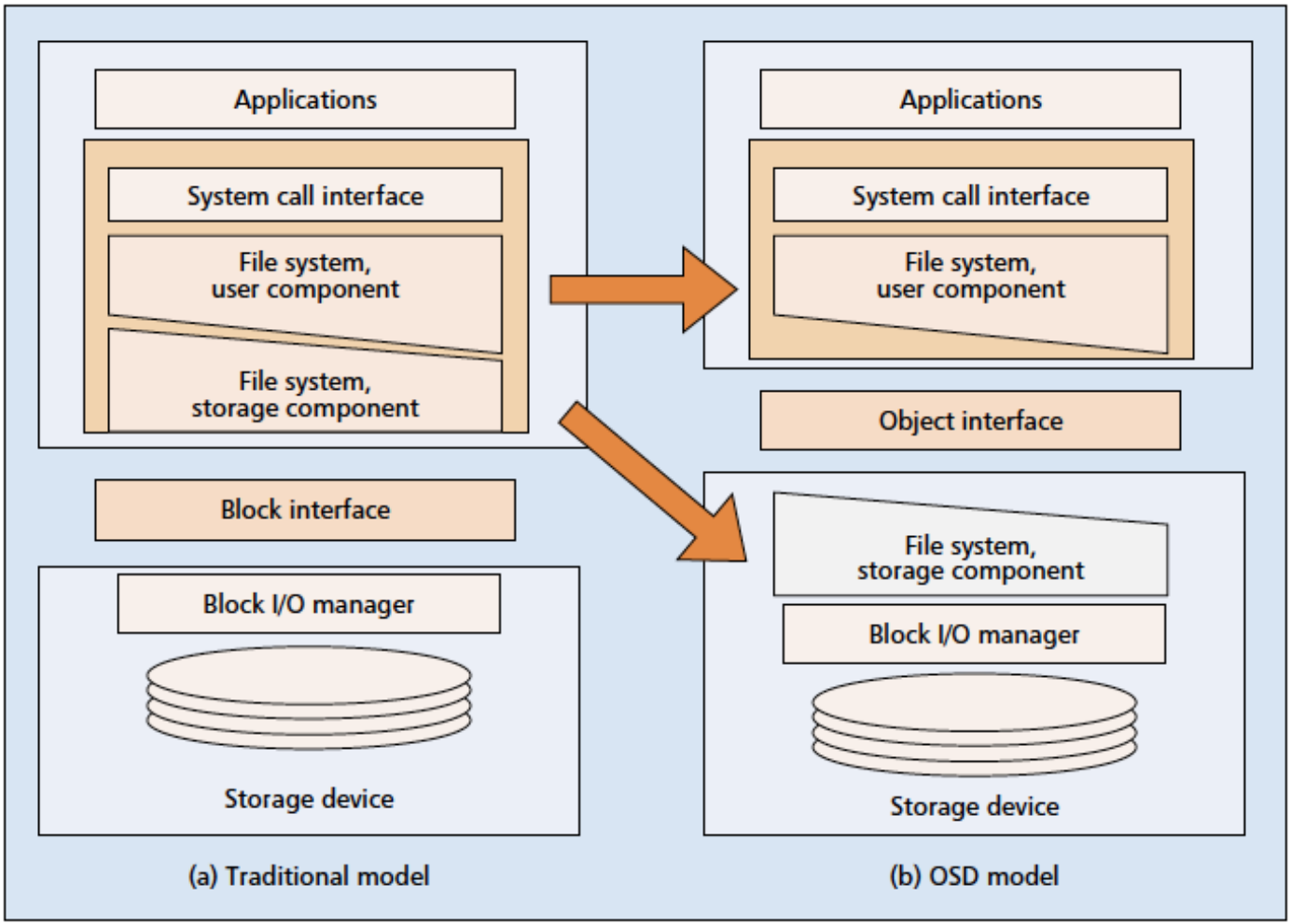
OSD form factors

Disk array/server subsystem
Example: custom-built HPC systems predominantly deployed in national labs

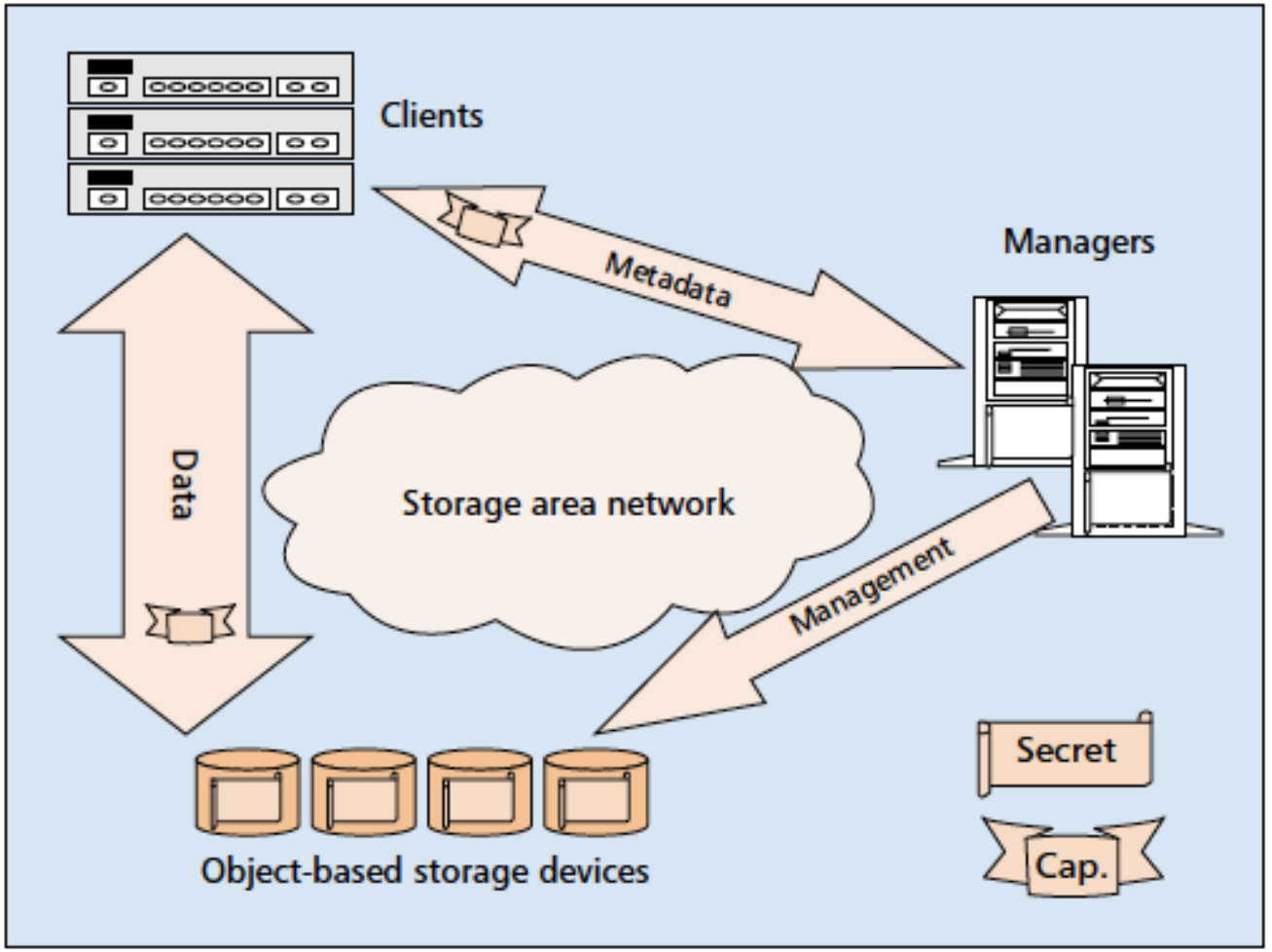
Storage bricks for objects
Example: commercial supercomputing offerings

Object Layer Integrated in Disk Drive
Example: only in prototypes

Offloading storage management from the file system

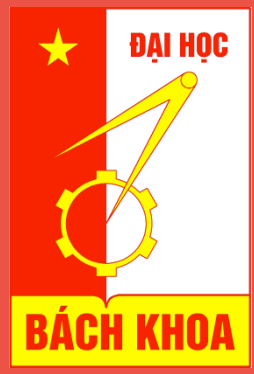


Object-based file system



Object-based file system

- Clients have direct access to storage to read and write file data securely
  - Contrast with SAN?
  - Contrast with NAS?
- File system includes multiple OSDs
  - Better scalability
- Multiple file systems share the same OSDs
  - Real storage pooling
- **File server is called the Metadata server (MDS)**



Design issues in distributed file systems

Design issues

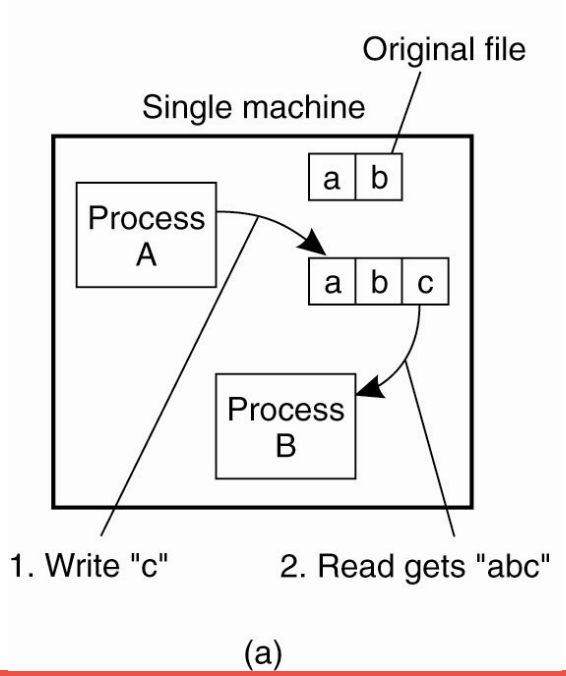
- Naming and Name Resolution
- Semantics of file sharing
- Caching
- Replication

Naming and Name Resolution

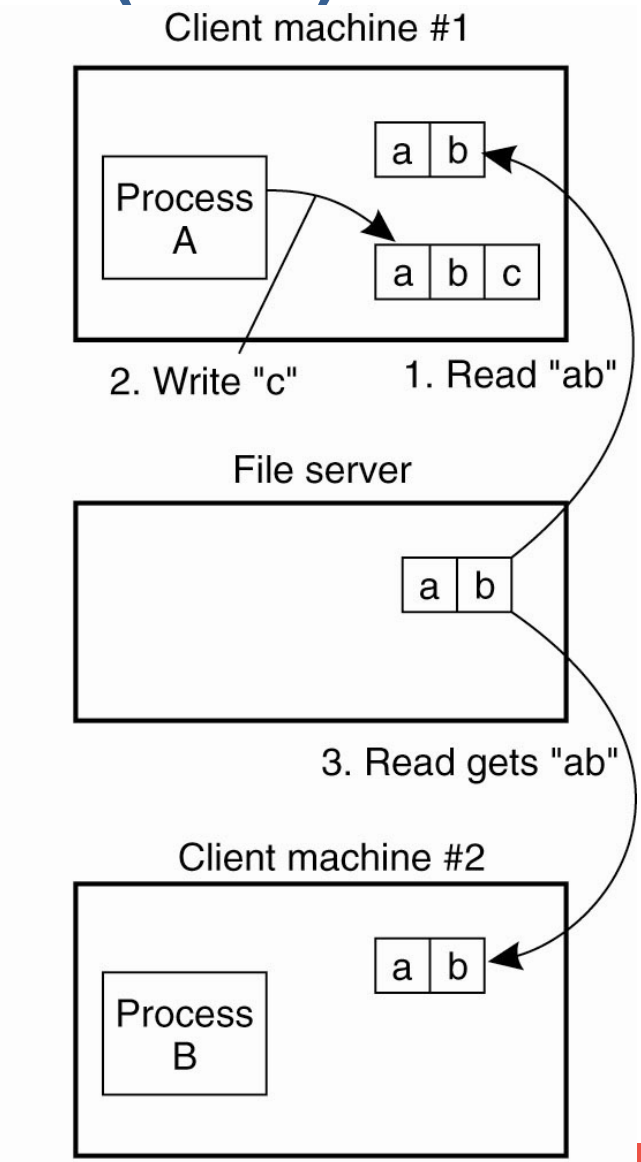
- A name space -- collection of names
- Name resolution -- mapping a name to an object
- 3 traditional ways
  - Concatenate the host name to the names of files stored on that host
  - Mount remote directories onto local directories
  - Provide a single global directory

File Sharing Semantics (1/2)

- **Problem:** When dealing with distributed file systems, we need to take into account the ordering of concurrent read/write operations, and expected semantics (=consistency).



(a)



(b)

File Sharing Semantics (2/2)

- Assume open; reads/writes; close
  1. **UNIX semantics:** value read is the value stored by last write
     Writes to an open file are visible immediately to others that have this file opened at the same time. Easy to implement if one server and no cache.
  2. **Session semantics:**
     Writes to an open file by a user is not visible immediately by other users that have files opened already.
     Once a file is closed, the changes made by it are visible by sessions started later.
  3. **Immutable-Shared-Files semantics:**
     A sharable file cannot be modified.
     File names cannot be reused and its contents may not be altered.
     Simple to implement.
  4. **Transactions:** All changes have all-or-nothing property. W1,R1,R2,W2 not allowed where P1 = W1;W2 and P2 = R1;R2

Caching

- **Server caching:** in main memory
  - cache management issue, how much to cache, replacement strategy
  - still slow due to network delay
  - Used in high-performance web-search engine servers
- **Client caching in main memory**
  - can be used by diskless workstation
  - faster to access from main memory than disk
  - compete with the virtual memory system for physical memory space
- **Client-cache on a local disk**
  - large files can be cached
  - the virtual memory management is simpler
  - a workstation can function even when it is disconnected from the network

Caching tradeoffs

- Reduces remote accesses ⇒ reduces network traffic and server load
- Total network overhead is lower for big chunks of data (caching) than a series of responses to specific requests.
- Disk access can be optimized better for large requests than random disk blocks
- Cache-consistency problem is the major drawback. If there are frequent writes, overhead due to the consistency problem is significant.

Replication

- File data is replicated to multiple storage servers
- Goals
  - Increase reliability
  - improve availability
  - balance the servers workload
- How to make replication transparent?
- How to keep the replicas consistent?
  - a replica is not updated due to its server failure
  - network partitioned

NFS & AFS file systems

Network File System (NFS)

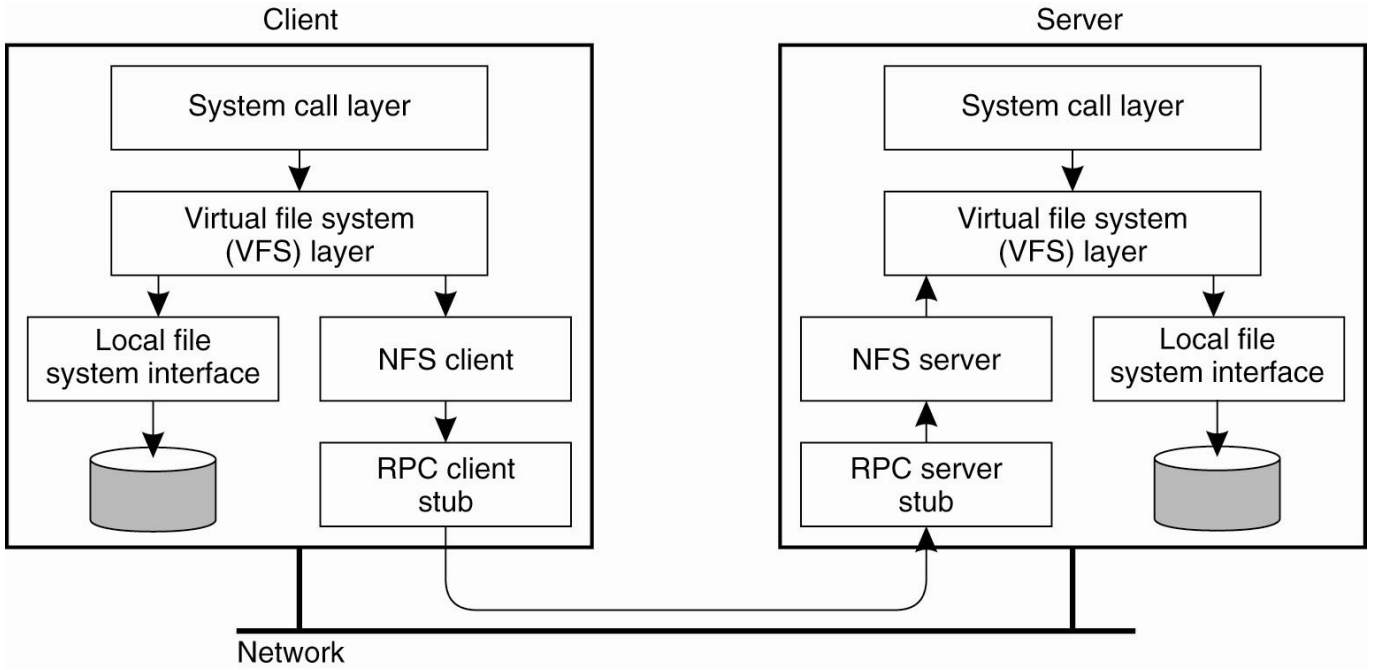
- First developed in 1980s by Sun
- Presented with standard UNIX FS interface
- Network drives are *mounted* into local directory hierarchy
  - Type ‘man mount’, 'mount' some time at the prompt if curious

NFS Protocol

- Initially completely stateless
  - Operated over UDP; did not use TCP streams
  - File locking, etc, implemented in higher-level protocols
- Modern implementations use TCP/IP & stateful protocols

NFS Architecture for UNIX systems

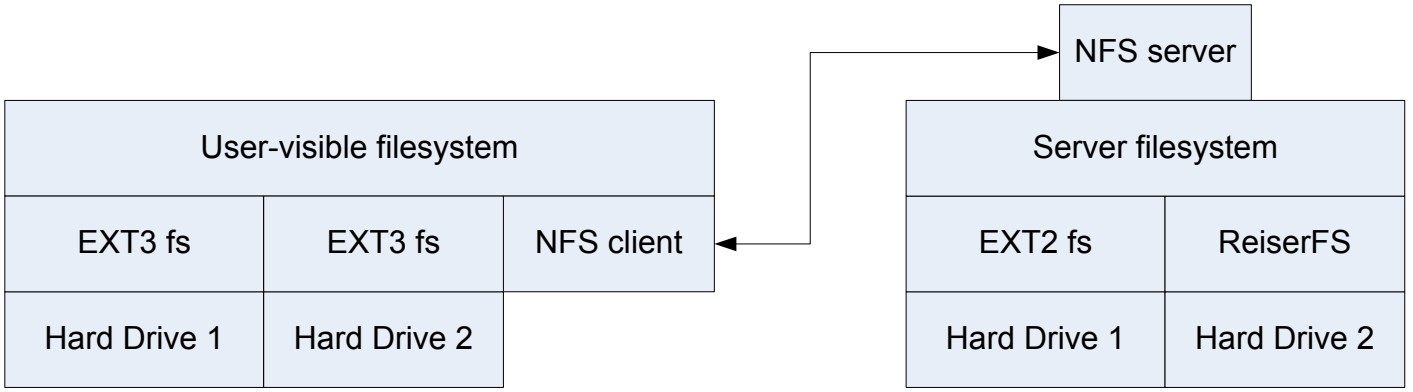
- NFS is implemented using the Virtual File System abstraction, which is now used for lots of different operating systems:



- Essence: VFS provides standard file system interface, and allows to hide difference between accessing local or remote file system.

Server-side Implementation

- NFS defines a *virtual file system*
  - Does not actually manage local disk layout on server
- Server instantiates NFS volume on top of local file system
  - Local hard drives managed by concrete file systems (EXT, ReiserFS, ...)



Typical implementation

- Assuming a Unix-style scenario in which one machine requires access to data stored on another machine:
- The server implements NFS daemon processes in order to make its data generically available to clients.
- The server administrator determines what to make available, exporting the names and parameters of directories.
- The server security-administration ensures that it can recognize and approve validated clients.
- The server network configuration ensures that appropriate clients can negotiate with it through any firewall system.
- The client machine requests access to exported data, typically by issuing a **mount** command.
- If all goes well, users on the client machine can then view and interact with mounted filesystems on the server within the parameters permitted.

NFS Locking

- NFS v4 supports stateful locking of files
  - Clients inform server of intent to lock
  - Server can notify clients of outstanding lock requests
  - Locking is lease-based: clients must continually renew locks before a timeout
  - Loss of contact with server abandons locks

NFS Client Caching

- NFS Clients are allowed to cache copies of remote files for subsequent accesses
- Supports *close-to-open* cache consistency
  - When client A closes a file, its contents are synchronized with the master, and timestamp is changed
  - When client B opens the file, it checks that local timestamp agrees with server timestamp. If not, it discards local copy.
  - Concurrent reader/writers must use flags to disable caching

NFS: Tradeoffs

- NFS Volume managed by single server
  - Higher load on central server
  - Simplifies coherency protocols
- Full POSIX system means it “drops in” very easily, but isn’t “great” for any specific need

Distributed FS Security

- Security is a concern at several levels throughout DFS stack
  - Authentication
  - Data transfer
  - Privilege escalation

- How are these applied in NFS?

AFS (The Andrew File System)

- Developed at Carnegie Mellon
- Strong security, high scalability
  - Supports 50,000+ clients at enterprise level
- AFS heavily influenced Version 4 of NFS.

Local Caching

- File reads/writes operate on locally cached copy
- Local copy sent back to master when file is closed
- Open local copies are notified of external updates through *callbacks*

Symmetric File Systems

Peer-to-Peer

Symmetric Architectures

- Fully distributed (decentralized) file systems do not distinguish between client machines and servers.
- Most proposed systems are based on a distributed hash table (DHT) approach for data distribution across nodes.
- The Ivy system is typical. It has a 3-layer structure.

Ivy System Structure

- The DHT layer implements a Chord scheme for mapping keys (which represent objects to be stored) to nodes.
- The DHash layer is a block-storage layer
  - Blocks = logical file blocks
  - Different blocks are stored in different locations
- The top, or file system layer, implements an NFS-like file system.

Characteristics

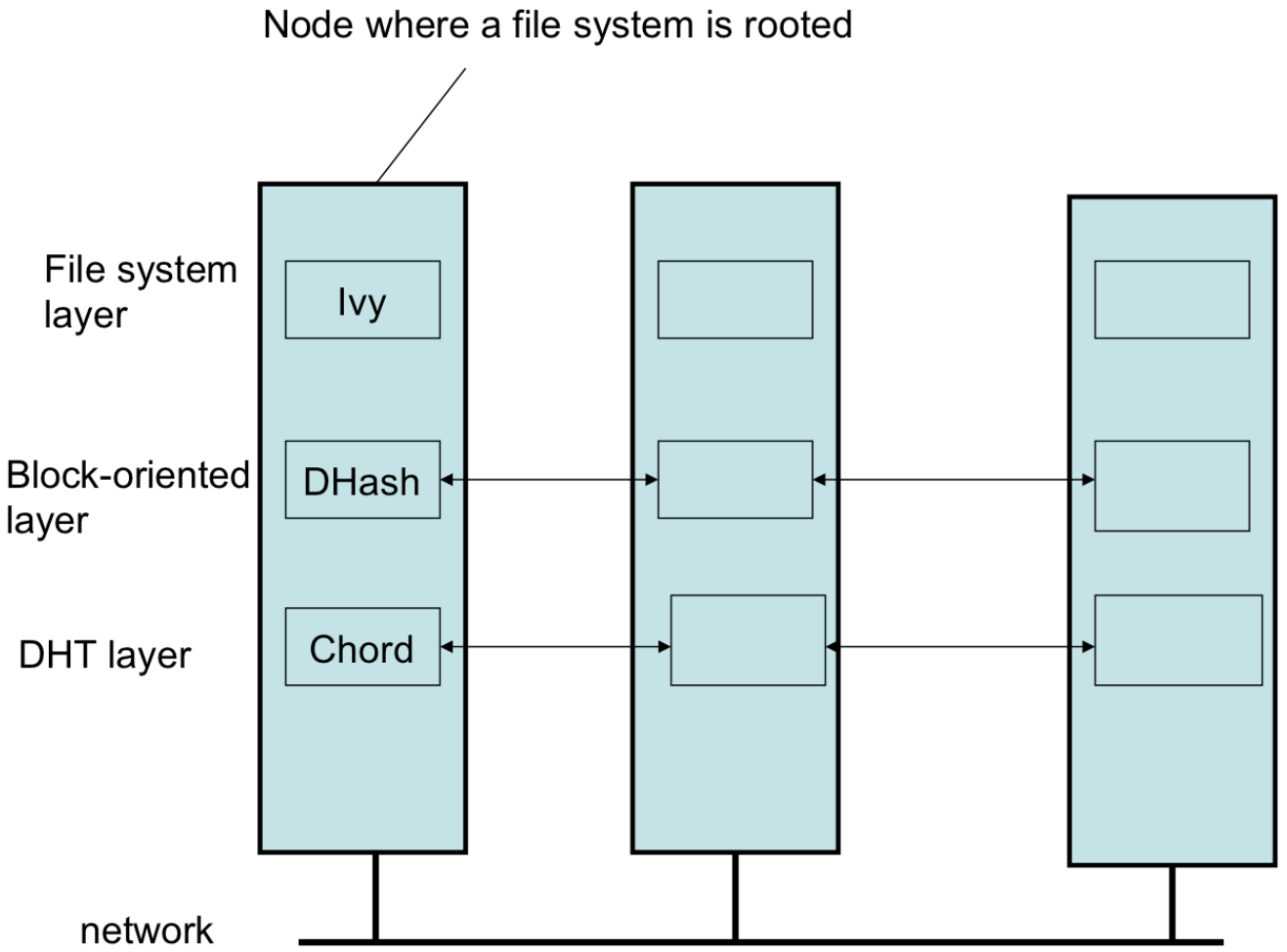
- File data and meta-data stored as blocks in a DHash P2P system
- Blocks are distributed and replicated across multiple sites – increased availability.
- Ivy is a read-write file system. Writing introduces consistency issues (of data and meta-data)

Characteristics

- Presents an NFS-like interface and semantics
- Performance: 2X-3X slower than NFS
- Potentially more available because of distributed nature

Logs

- Like most P2P systems, trustworthiness of users is not guaranteed
  - Must be able to undo modifications
- Network partitioning means the possibility of conflicting updates – how to manage?
- Solution: logs – one per user
  - Used to record changes made locally to file data and metadata
  - Contrast to shared data structures
  - Avoids use of locks for updating metadata


Node where a file system is rooted
File system layer
Ivy
Block-oriented layer
DHash
DHT layer
Chord
network

Dhash layer

- Manages data blocks (of a file)
- Stored as content-hash block or public-key block
- Content-hash blocks
  - Compute the secure hash of this block to get the key
  - Clients must know the key to look up a block
  - When the block is returned to a client, compute its hash to verify that this is the correct (uncorrupted) block.

DHash Layer – Public Key Blocks

- "A public key block requires the block's key to be a public key, and the value to be signed using the private key."
- Users can look up a block without the private key, but cannot change data unless they have the private key.
- Ivy layer verifies all the data DHash returns and is able to protect against malicious or corrupted data.

DHash Layer

- The DHash layer replicates each file block B to the next k successors of the server that stores B.
  - (remember how Chord maps keys to nodes)
- This layer has no concept of files or file systems. It merely knows about blocks

Ivy – the File System Layer

- A file is represented by a log of operations
- The log is a linked list of immutable (can't be changed) records.
  - Contains all of the additions made by a single user (to data and metadata)
- Each record records a file system operation (open, write, etc.) as a DHash content-hash block.
- A *log-head* node is a pointer to the most recent log entry

Using Logs

- A user must consult all logs to read file data, (find records that represent writes) but makes changes only by adding records to its own log.
- Logs contain data and metadata
- Start scan with most recent entry
- Keep local snapshot of file to avoid having to scan entire logs

- Update: Each participant maintains a log of its changes to the file system
- Lookup: Each participant scans all logs
- The view-block has pointers to all log-heads

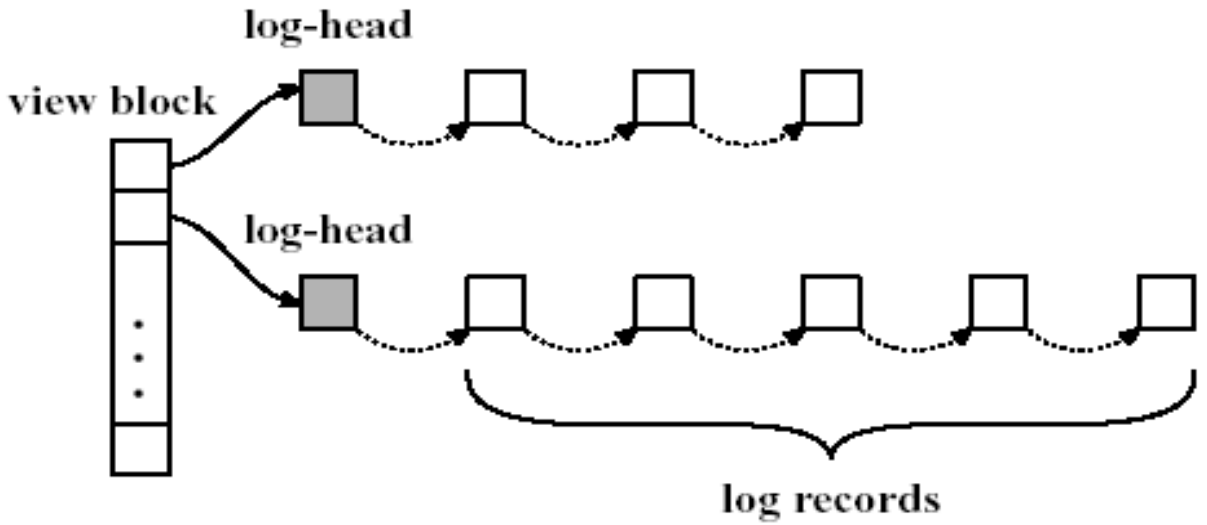


Figure 1: Example Ivy view and logs. White boxes are DHash content-hash blocks; gray boxes are public-key blocks.

Combining Logs

- Block order should reflect causality
- All users should see same order
- For each new log record assign
  - A sequence # (orders blocks in a single log)
  - A tuple with an entry for each log showing the most recent info about that log (from current user's viewpoint)
    - Tuples are compared somewhat like vector timestamps; either u < v or v < u or v = u or no relation (v and u are concurrent)
    - Concurrency is the result of simultaneous updates