Cloud Computing
CS 15-319

Distributed File Systems and Cloud Storage – Part I

Lecture 12, Feb 22, 2012

Majd F. Sakr, Mohammad Hammoud and
Suhail Rehman

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

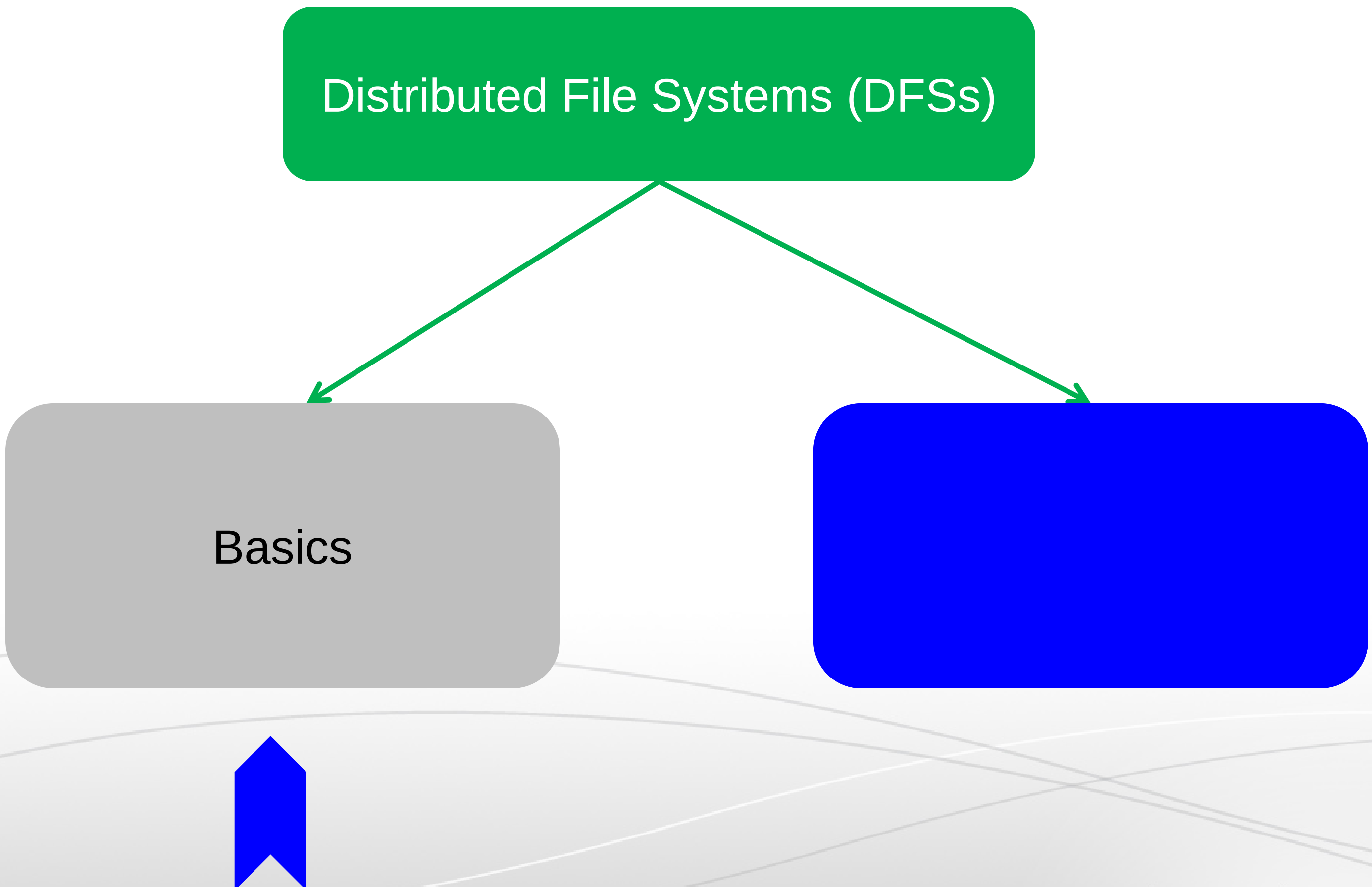
Today…

- Last two sessions
  - Pregel, Dryad and GraphLab

- Today's session
  - Distributed File Systems- Part I

- Announcement:
  - Project update is due today

Discussion on Distributed File Systems



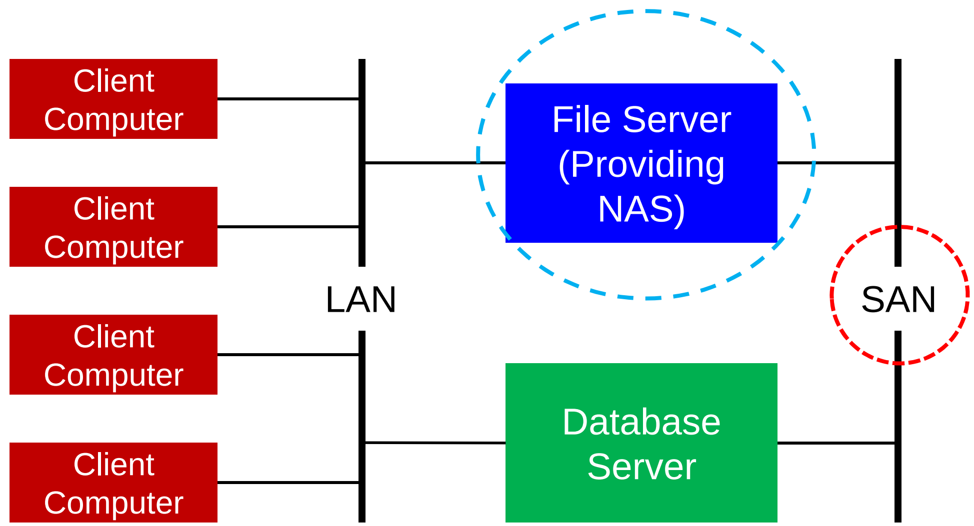
Distributed File Systems

- *Why File Systems?*
  - To organize data (as files)
  - To provide a means for applications to *store, access, and modify* data
- *Why Distributed File Systems?*
  - Big data continues to grow
  - In contrary to a local file system, a distributed file system (DFS) can hold big data and provide access to this data to many clients distributed across a network

NAS versus SAN

- Another term for DFS is *network attached storage* (NAS), referring to attaching storage to network servers that provide file systems

- A similar sounding term that refers to a very different approach is *storage area network* (SAN)
  - SAN makes storage devices (not file systems) available over a network



Benefits of DFSs

- DFSs provide:

  1. File sharing over a network: without a DFS, we would have to exchange files by e-mail or use applications such as the Internet's FTP
  2. Transparent files accesses: A user's programs can access remote files as if they are local. The remote files have no special APIs; they are accessed just like local ones
  3. Easy file management: managing a DFS is easier than managing multiple local file systems

DFS Components

- DFS information can be typically categorized as follows:

  1. The data state: This is the contents of files
  2. The attribute state (meta data): This is the information about each file (e.g., file's size and access control list)
  3. The open-file state: This includes which files are open or otherwise in use, as well as describing how files are locked

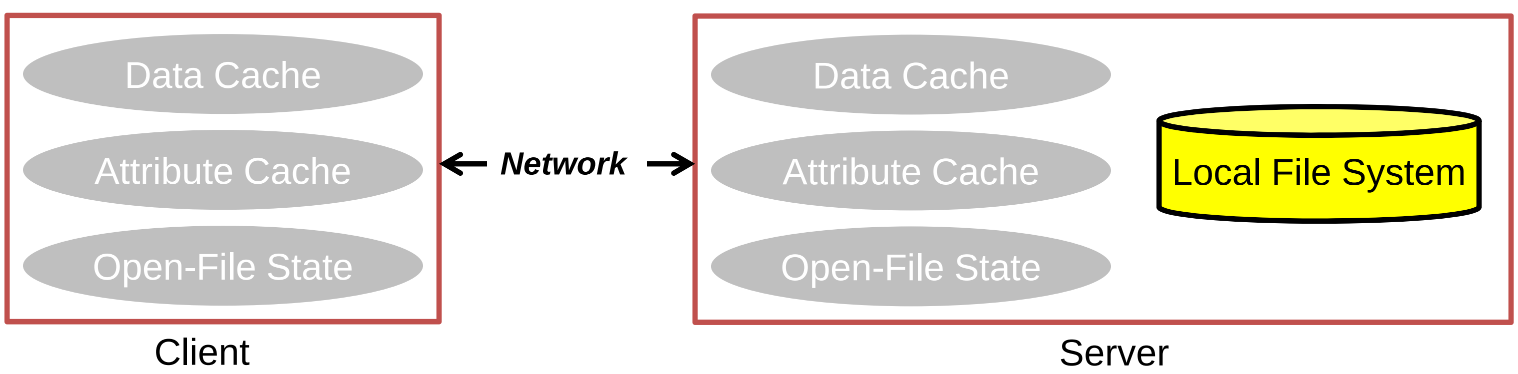
- Designing a DFS entails determining how its various components are placed. Specifically, by component placement we indicate:

  - What resides on the servers
  - What resides on the clients

DFS Component Placement (1)

- The data and the attribute states permanently reside on the server's local file system, but recently accessed or modified information *might* reside in server and/or client caches

- The open-file state is *transitory*; it changes as processes open and close files



DFS Component Placement (2)

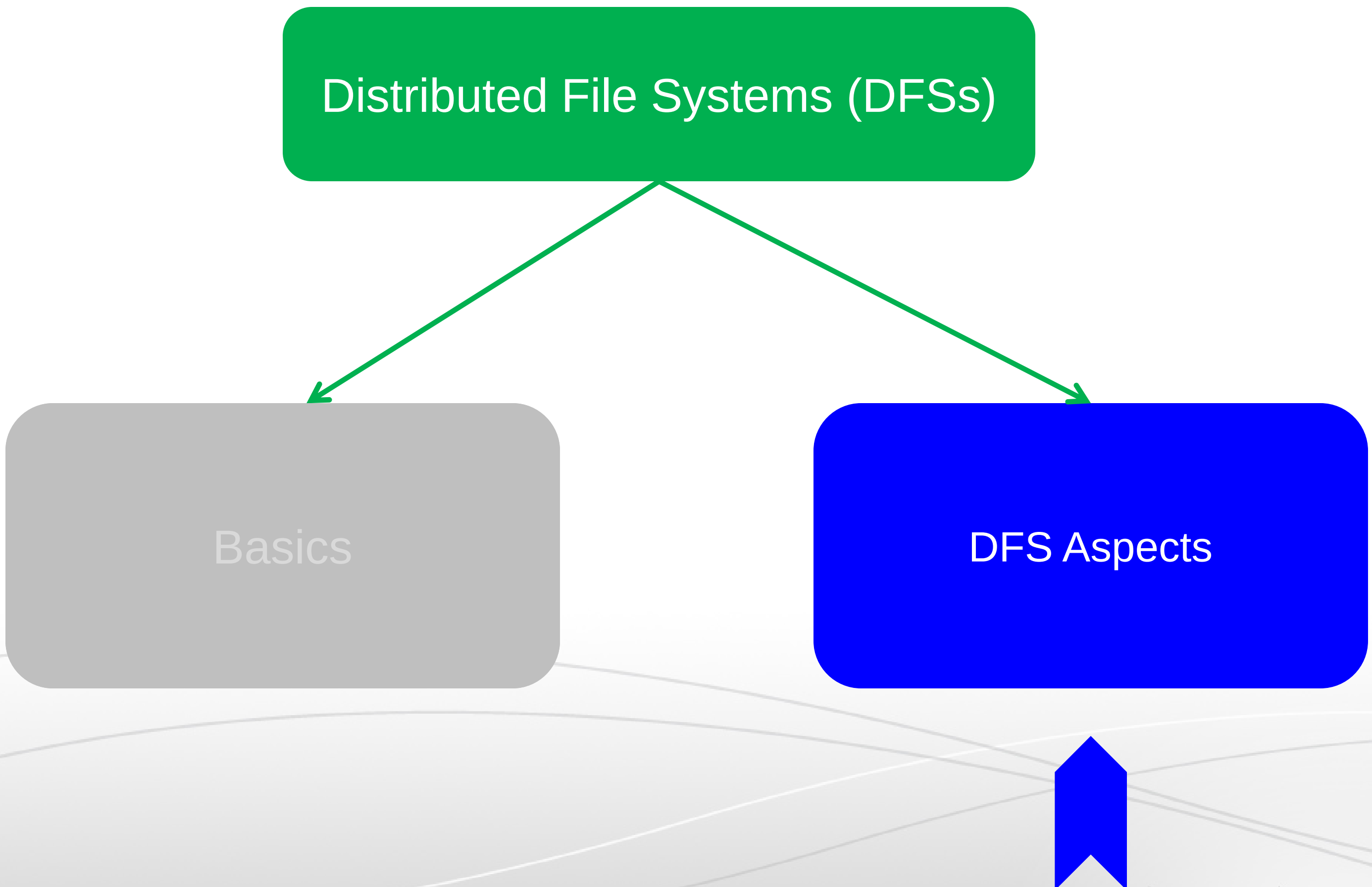
- Three *basic* concerns govern the DFS components placement strategy:

1. Access speed: Caching information on clients improves performance considerably

2. Consistency: If clients cache information, do all parties share the same view of it?

3. Recovery: If one or more computers crash, to what extent are the others affected? How much information is lost?

Discussion on Distributed File Systems



DFS Aspects

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?
Fault Tolerance	How is fault tolerance handled in DFSs?

DFS Aspects

Aspect	Description
Architecture	How are DFSs generally organized?

Architectures

1. Client-Server Distributed File Systems
2. Cluster-Based Distributed File Systems
3. Symmetric Distributed File Systems

Network File System

- Many distributed file systems are organized along the lines of client-server architectures

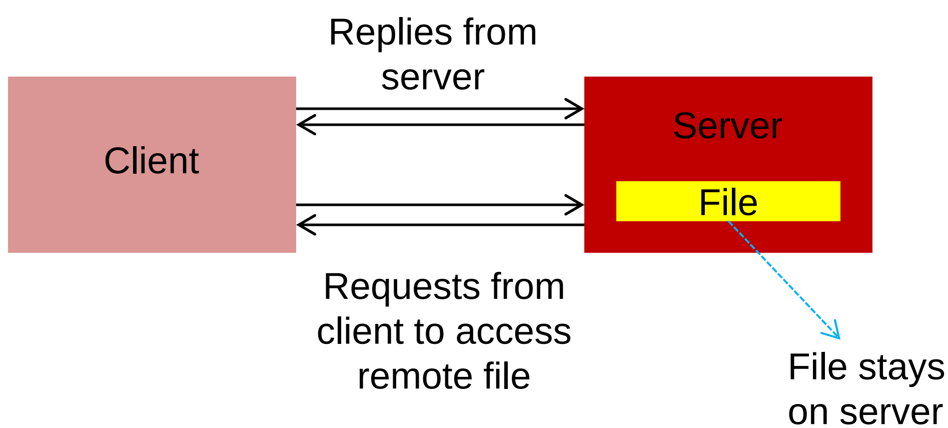
- Sun Microsystem's Network File System (NFS) is one of the most widely-deployed DFSs for Unix-based systems

- NFS comes with a protocol that describes precisely how a client can access a file stored on a (remote) NFS file server

- NFS allows a heterogeneous collection of processes, possibly running on different OSs and machines, to share a common file system

Remote Access Model

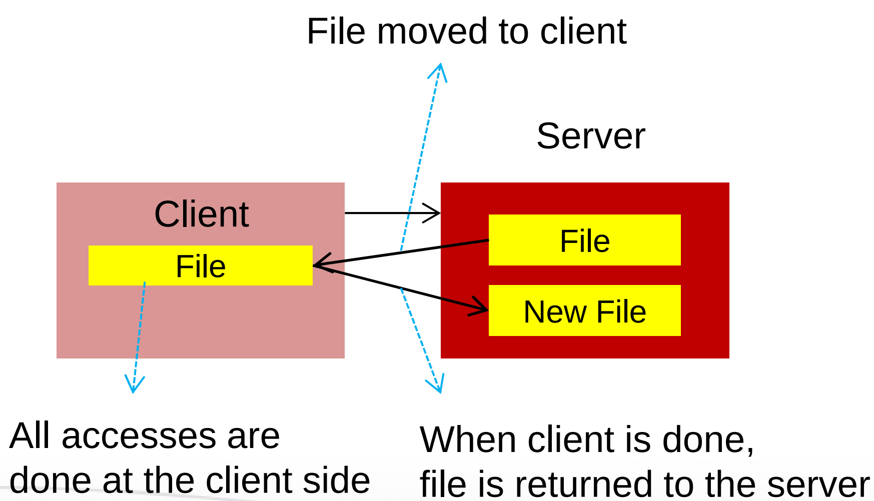
- The model underlying NFS and similar systems is that of remote access model



- In this model, clients:
  - Are offered transparent access to a file system that is managed by a remote server
  - Are normally unaware of the actual location of files
  - Are offered an interface to a file system similar to the interface offered by a conventional local file system

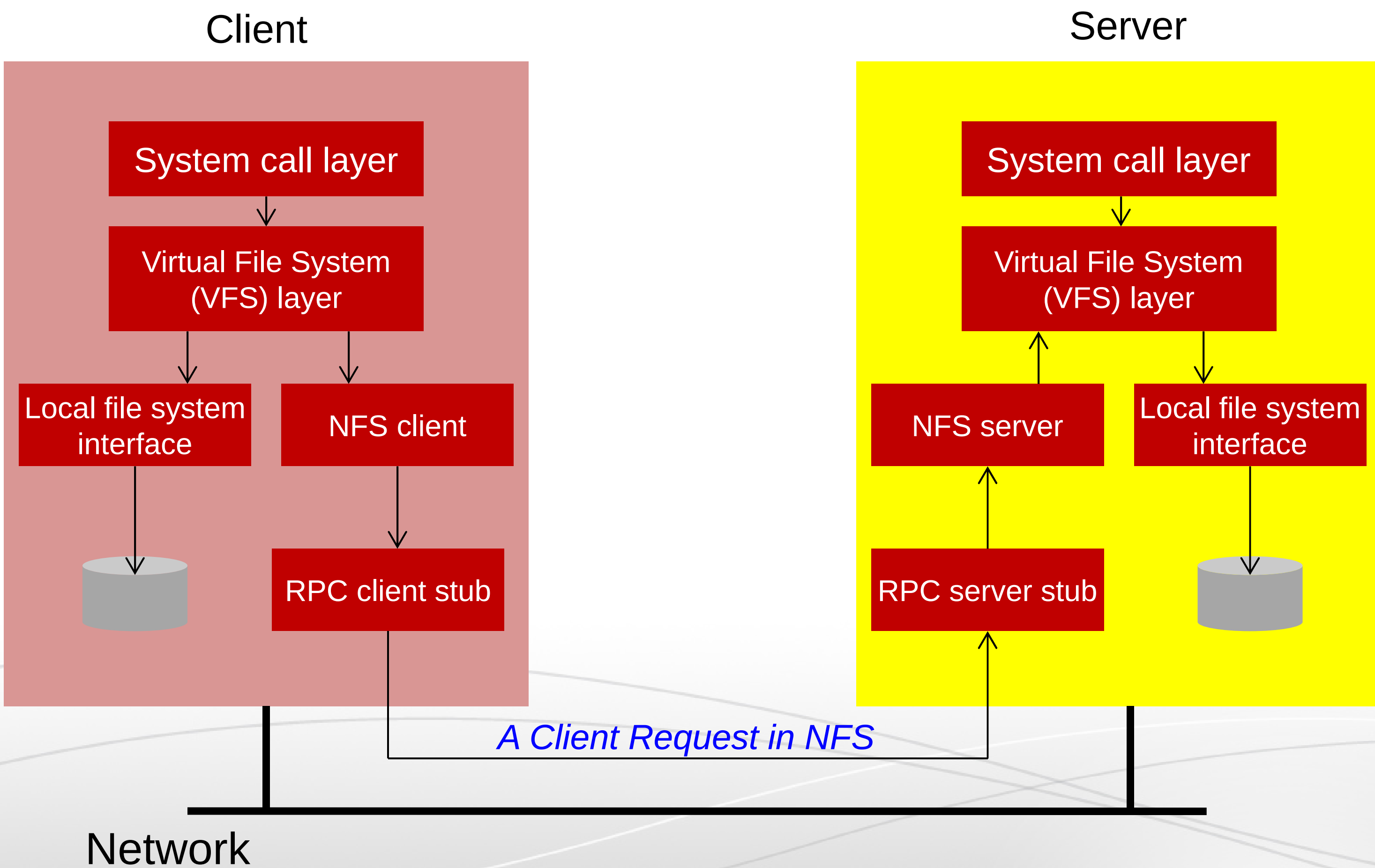
Upload/Download Model

- A contrary model, referred to as upload/download model, allows a client to access a file *locally* after having downloaded it from the server



- The Internet's FTP service can be used this way when a client downloads a complete file, modifies it, and then puts it back

The Basic NFS Architecture

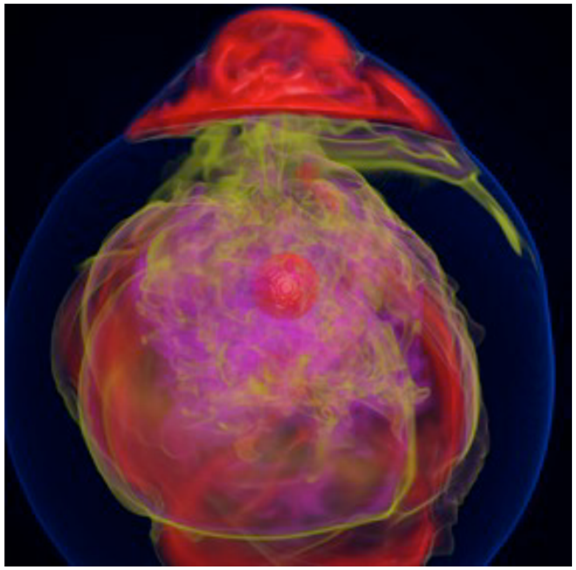


Architectures

1. Client-Server Distributed File Systems
2. Cluster-Based Distributed File Systems
3. Symmetric Distributed File Systems

Data-Intensive Applications

- Today there is a deluge of large data-intensive applications
- Most data-intensive applications fall into one of two styles of computing:
  - Internet services (or cloud computing)
  - High-performance computing (HPC)
- Cloud computing and HPC applications run typically on thousands of compute nodes and handle big data



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Cluster-Based Distributed File Systems

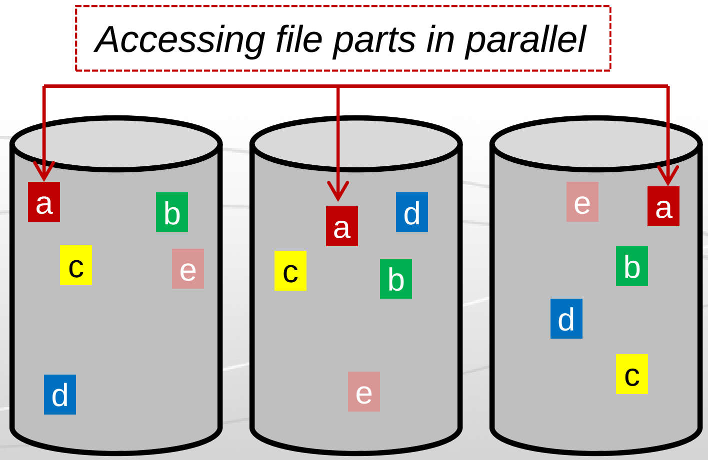
- The underlying cluster-based file system is a key component for providing scalable data-intensive application performance

- The cluster-based file system divides and distributes big data, using file striping techniques, for allowing concurrent data accesses

- The cluster-based file system could be either a cloud computing or an HPC oriented distributed file system
  - Google File System (GFS) and S3 are examples of cloud computing DFSs
  - Parallel Virtual File System (PVFS) and IBM's General Parallel File System (GPFS) are examples of HPC DFSs

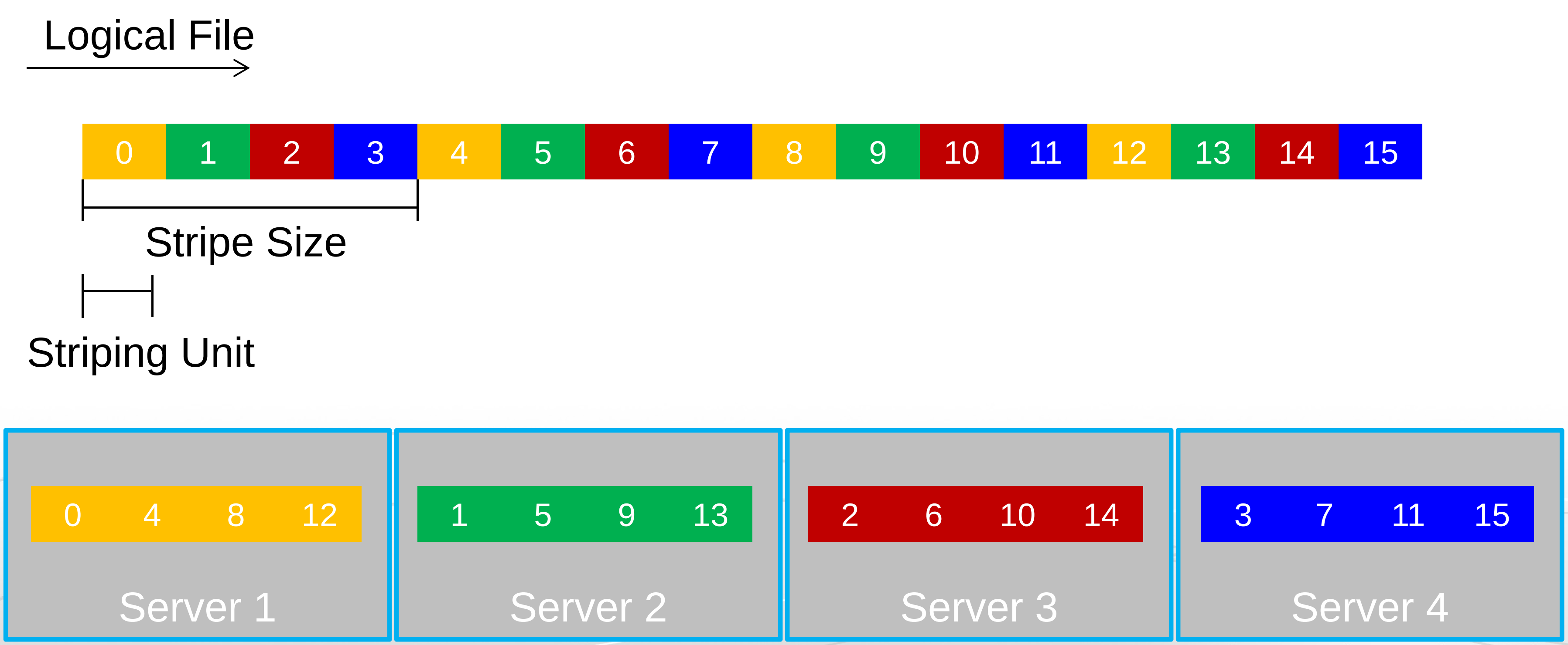
File Striping Techniques

- Server clusters are often used for parallel applications and their associated file systems are adjusted to satisfy their requirements
- One well-known technique is to deploy file-striping techniques, by which a single file is distributed across multiple servers
- Hence, it becomes possible to fetch different parts in parallel



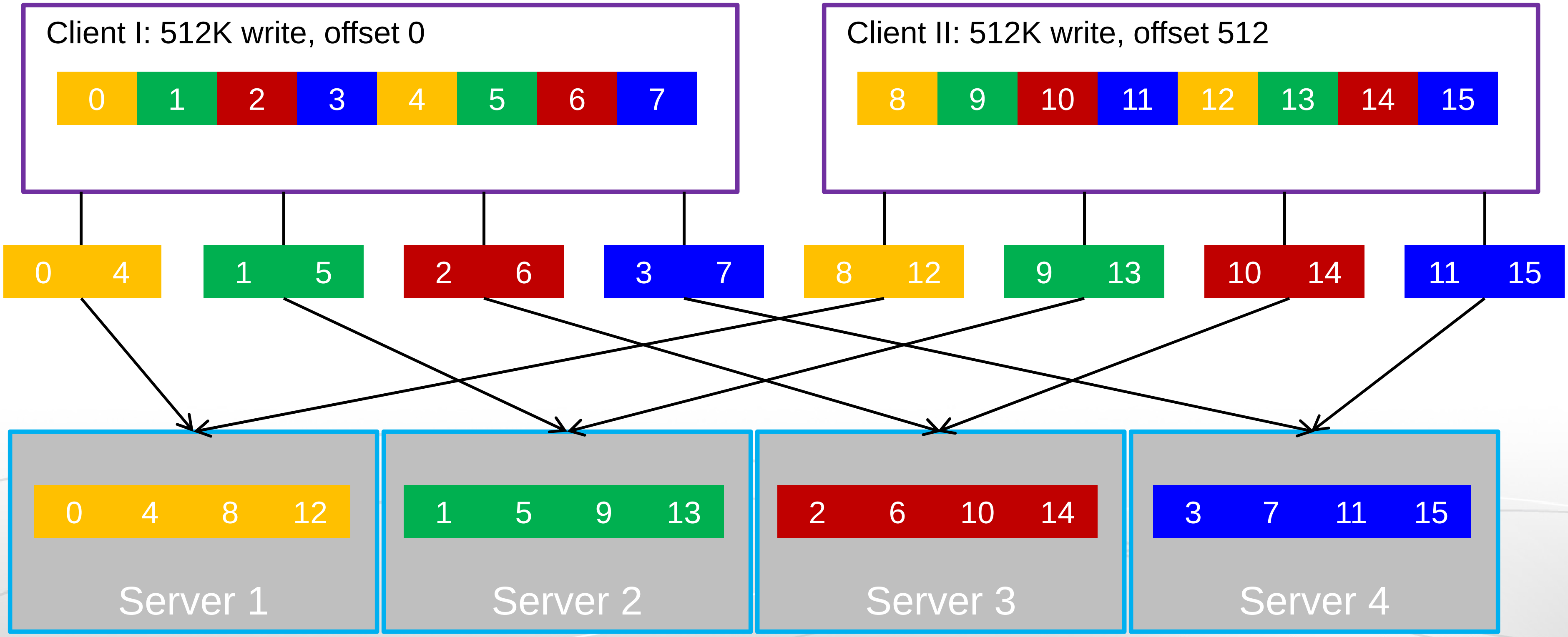
Round-Robin Distribution (1)

- How to stripe a file over multiple machines?
  - Round-Robin is typically a reasonable default solution



Round-Robin Distribution (2)

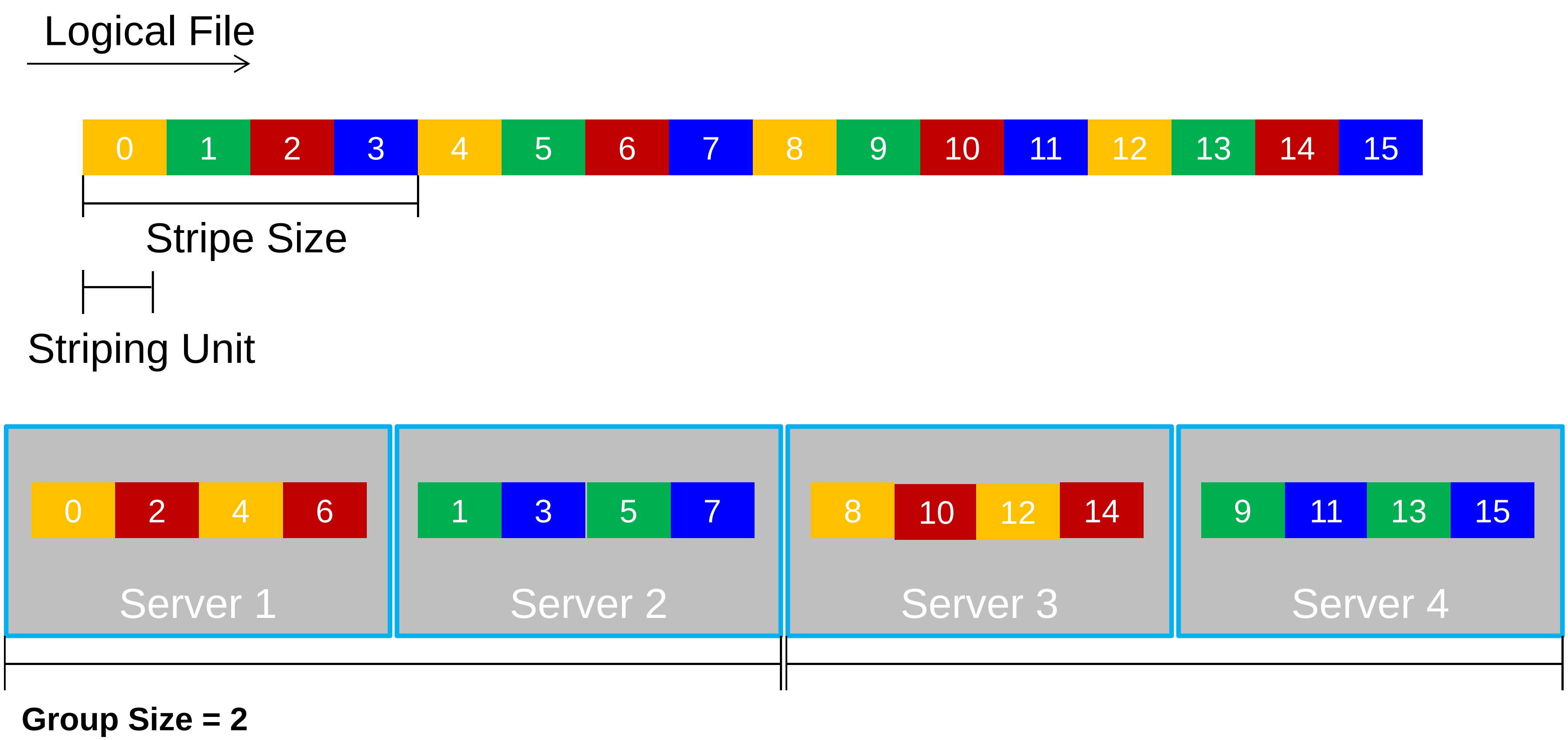
- Clients perform writes/reads of file at various regions



جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

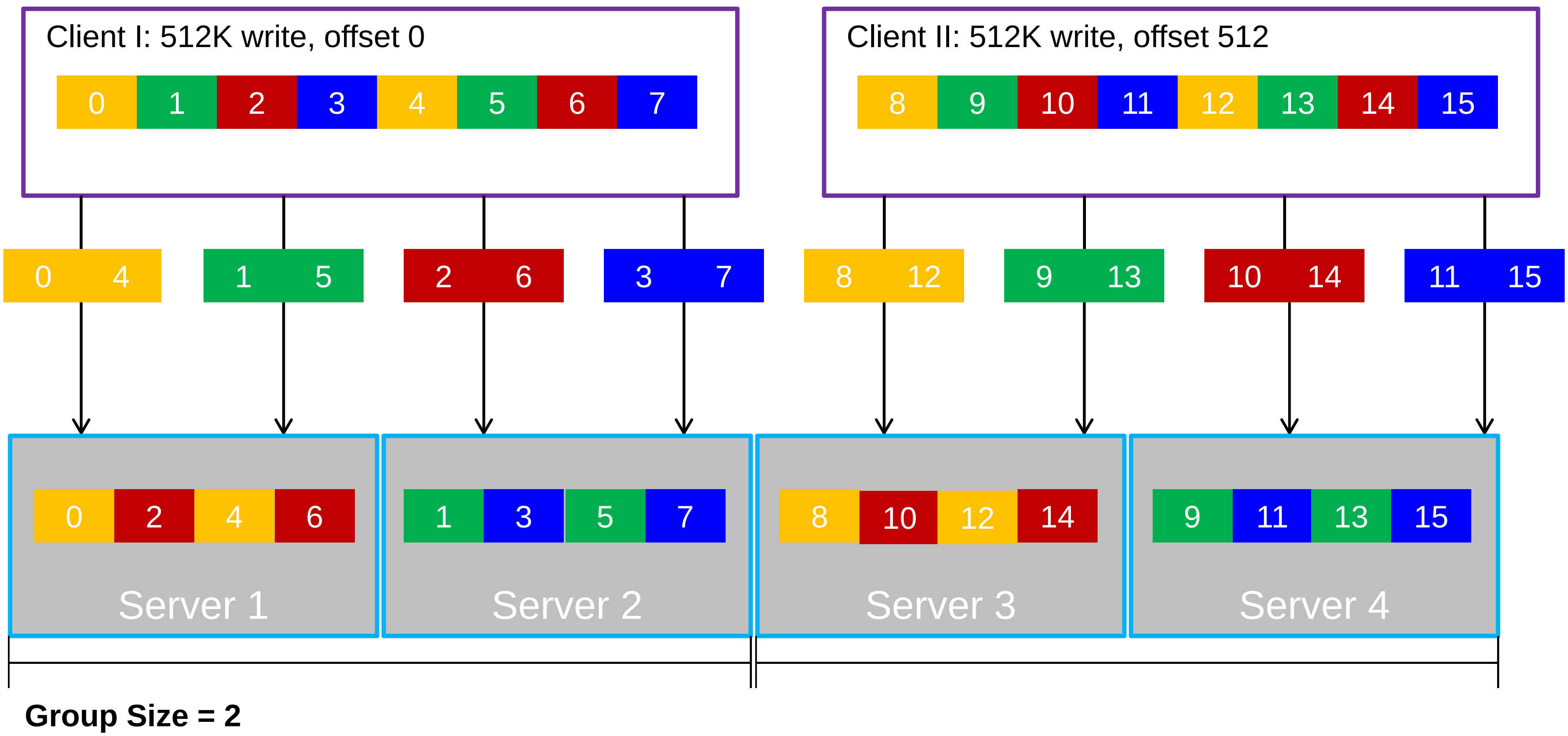
2D Round-Robin Distribution (1)

- What happens when we have many servers (say 100s)?
  - 2D distribution can help



2D Round-Robin Distribution (2)

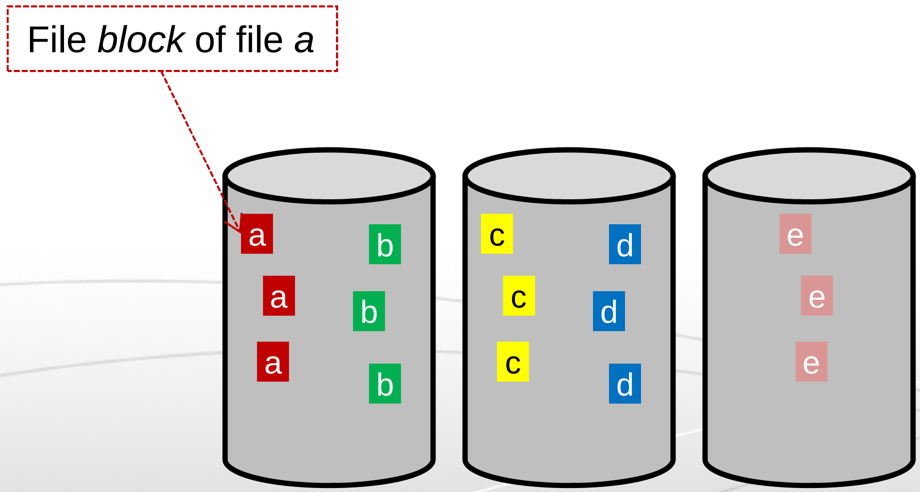
- 2D distribution can limit the number of servers per client



General Purpose Applications

- For general-purpose data-intensive applications, or those with irregular or many different types of data structures, file striping may not be effective

- In those cases, it is often more convenient to partition the file system as a *whole* and simply store files on different servers



GFS Data Distribution Policy

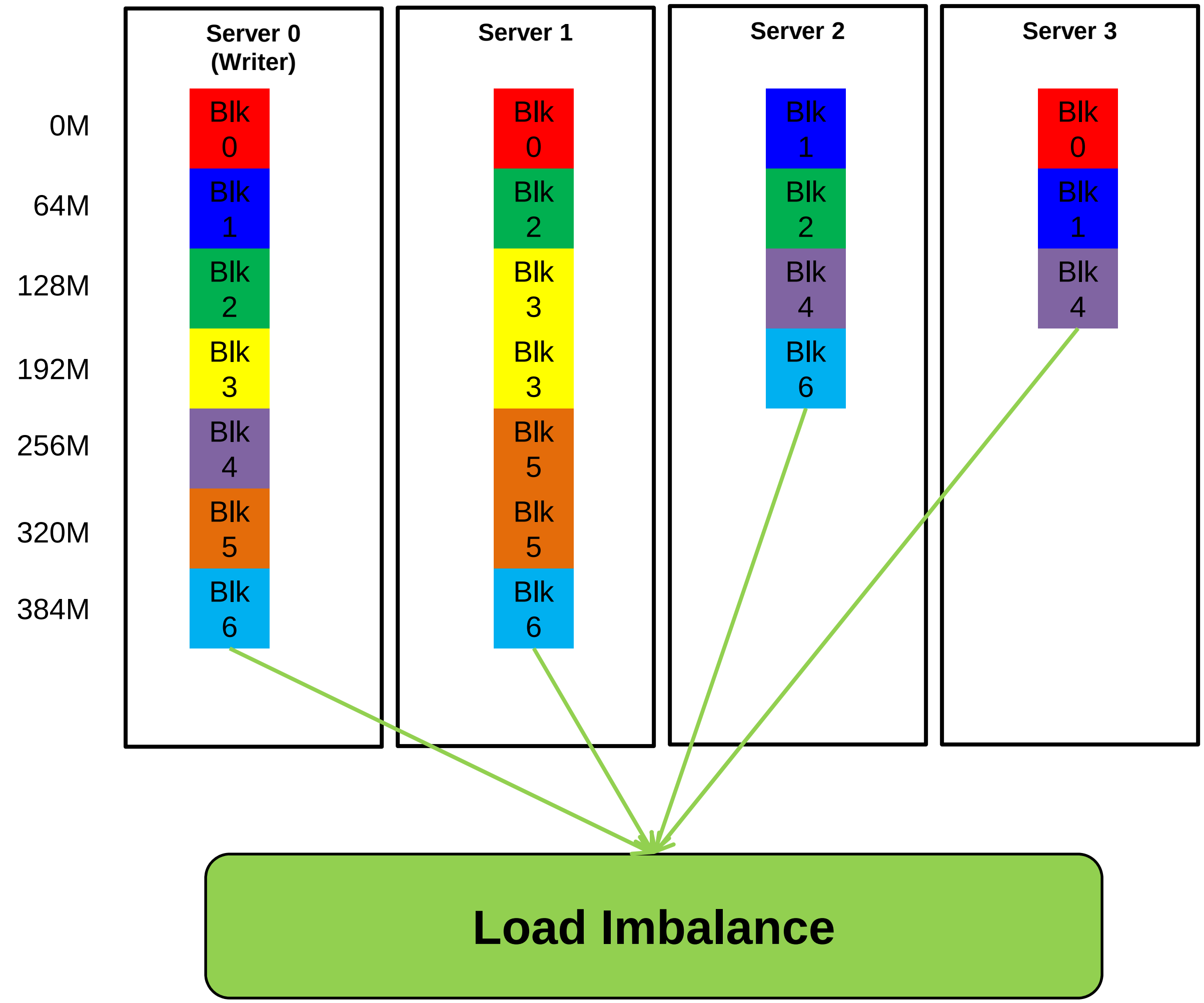
- The Google File System (GFS) is a cloud-computing-based scalable DFS for large distributed data-intensive applications

- GFS divides large files into multiple pieces called chunks or blocks (by default 64MB) and stores them on different data servers
  - This design is referred to as block-based design

- Each GFS chunk has a unique 64-bit identifier and is stored as a file in the lower-layer local file system on the data server

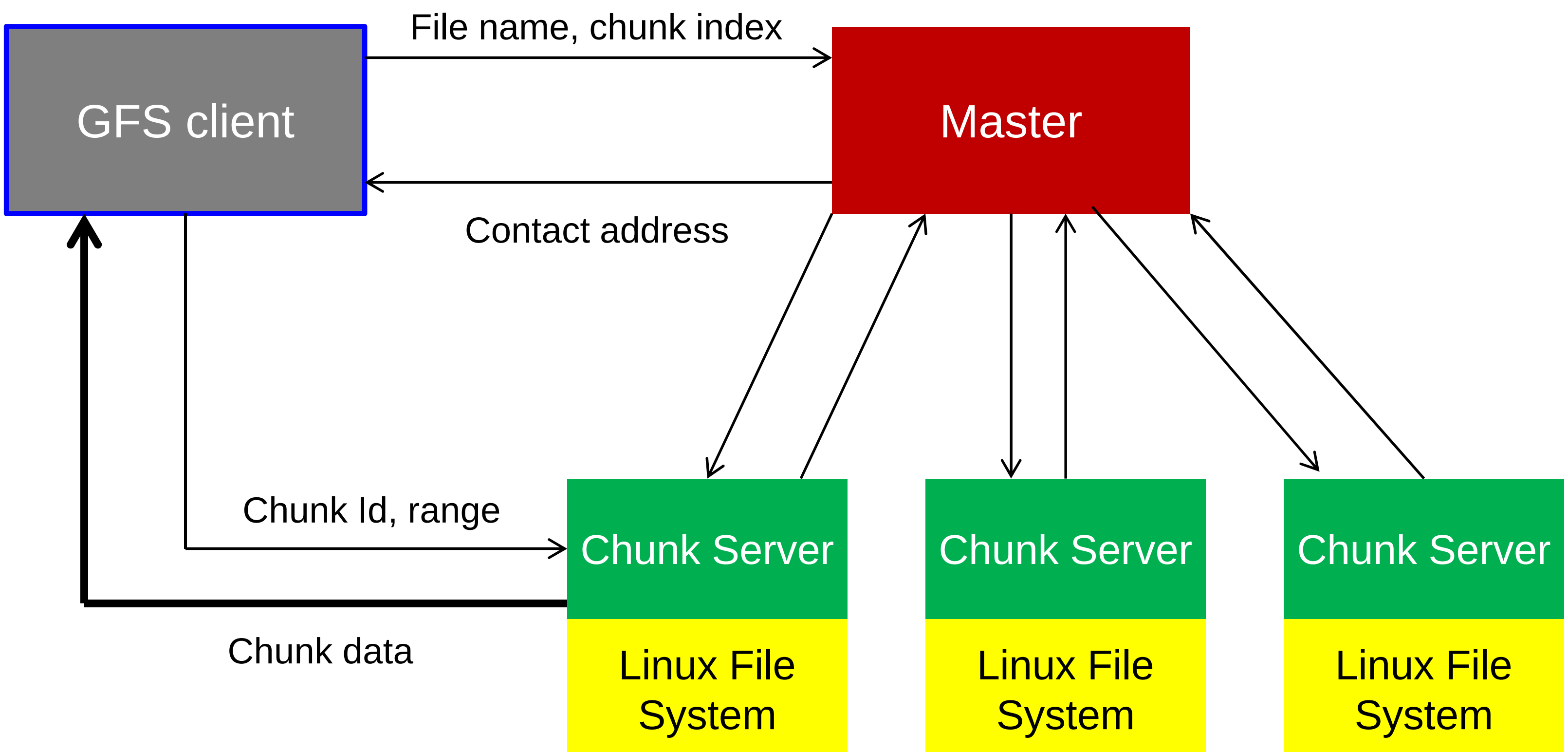
- GFS distributes chunks across cluster data servers using a random distribution policy

GFS Random Distribution Policy



GFS Architecture

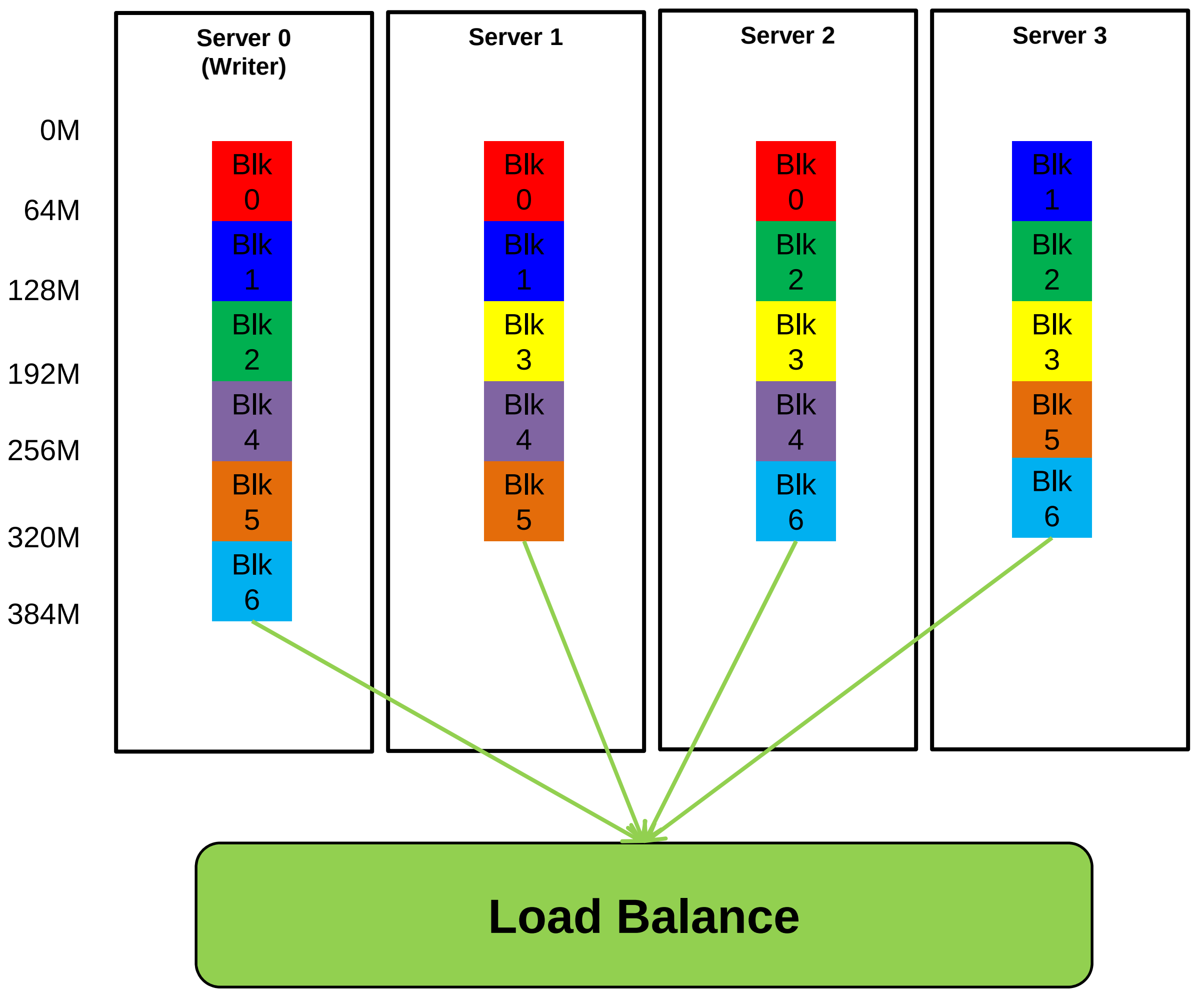
- The storage and compute capabilities of a cluster are organized in two ways:
    1. Co-locate storage and compute in the same node **GFS**
    2. Separate storage nodes from compute nodes



PVFS Data Distribution Policy

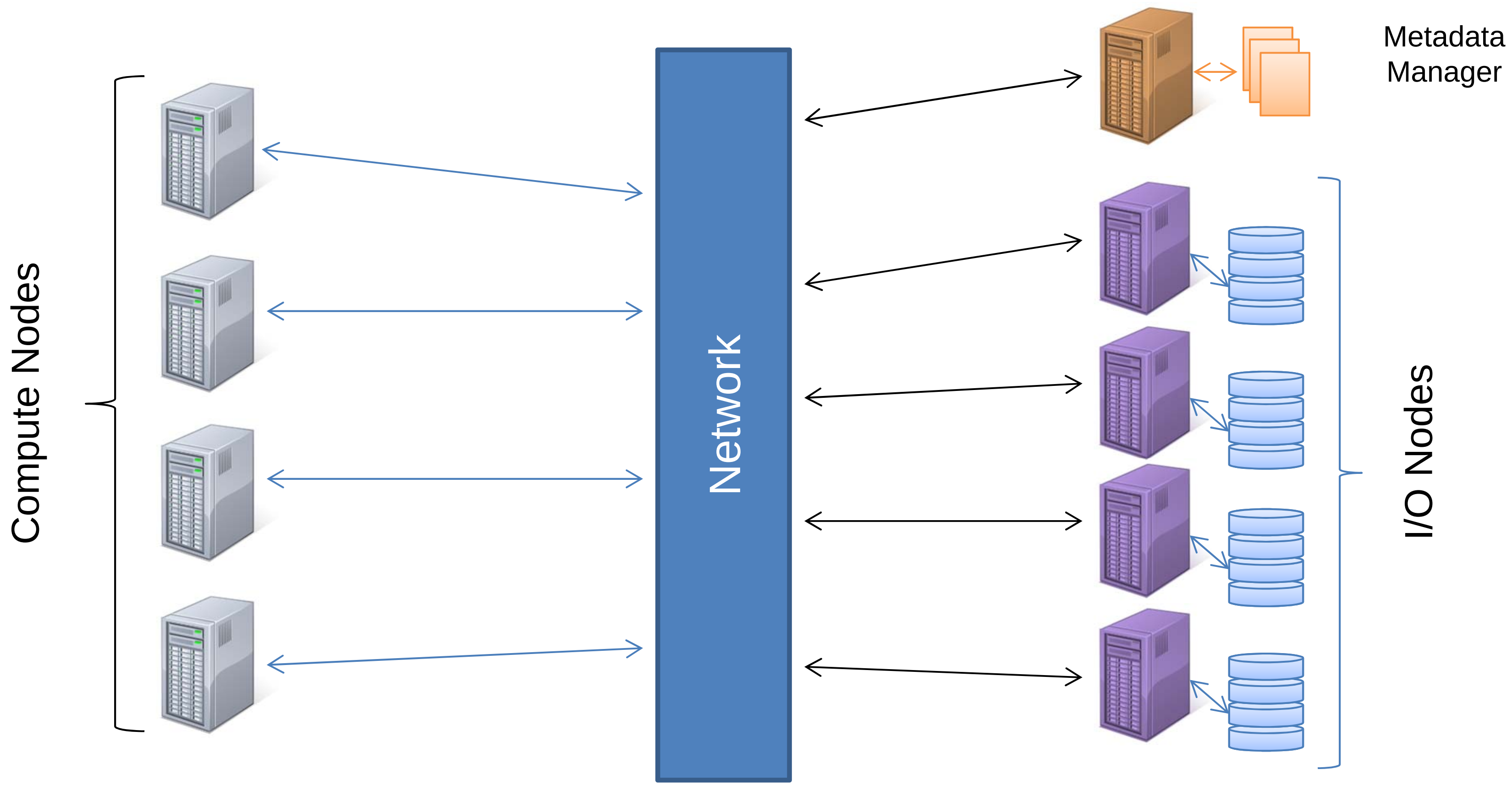
- Parallel Virtual File System (PVFS) is an HPC-based scalable DFS for large distributed data-intensive applications
- PVFS divides large files into multiple pieces called stripe units (by default 64KB) and stores them on different data servers
  - This design is referred to as object-based design
- Unlike the block-based design of GFS, PVFS stores an object (or a handle) as a file that includes all the stripe units at *a* data server
- PVFS distributes stripe units across cluster data servers using a round robin policy

PVFS Round-Robin Distribution Policy



PVFS Architecture

- The storage and compute capabilities of a cluster are organized in two ways:
  1. Co-locate storage and compute in the same node
  2. Separate storage nodes **PVFS** compute nodes



Architectures

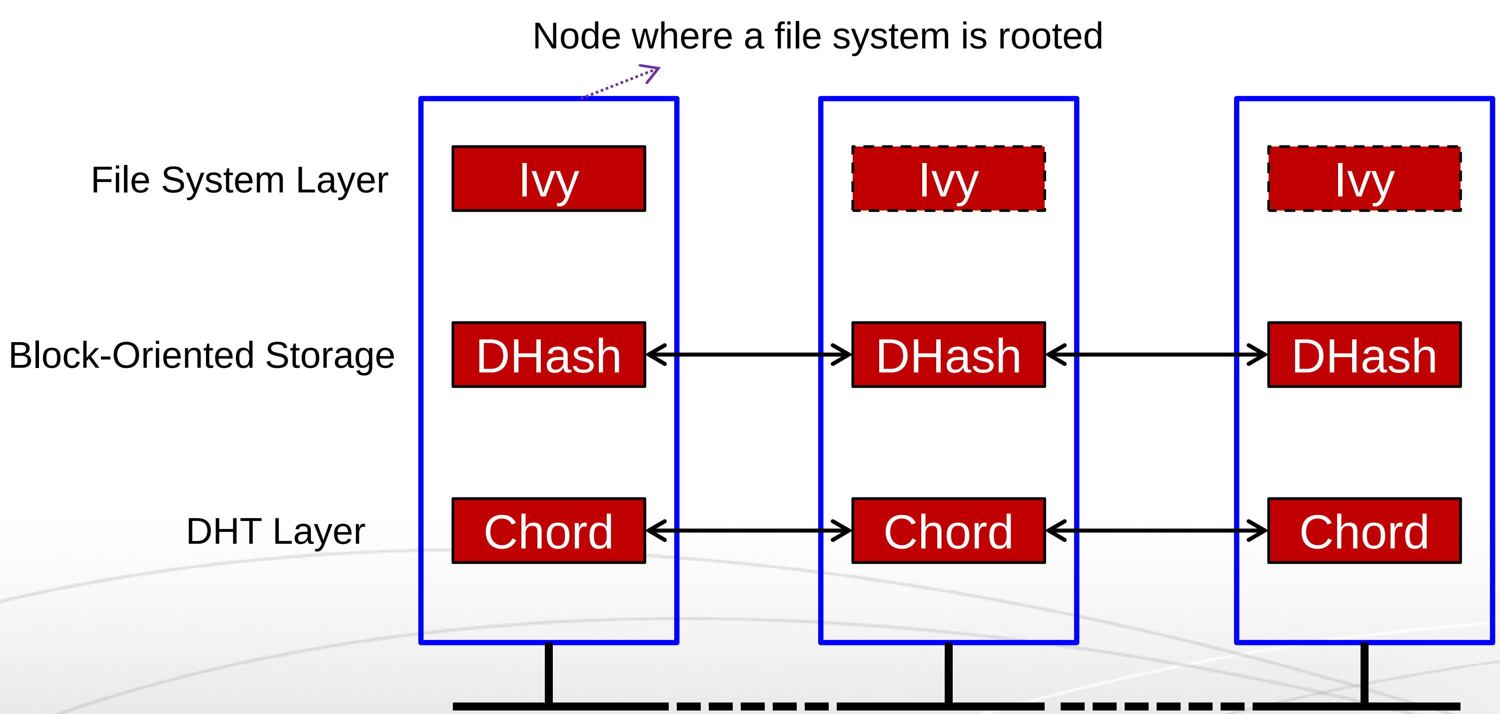
- Client-Server Distributed File Systems
- Cluster-Based Distributed File Systems
- Symmetric Distributed File Systems

Ivy

- Fully symmetric organizations that are based on peer-to-peer technology also exist
- All current proposals use a DHT-based system for distributing data, combined with a key-based lookup mechanism
- As an example, Ivy is a distributed file system that is built using a Chord DHT-based system
- Data storage in Ivy is realized by a block-oriented (i.e., blocks are distributed over storage nodes) distributed storage called DHash

Ivy Architecture

- Ivy consists of 3 separate layers:



DFS Aspects

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?

Processes (1)

- Cooperating processes in DFSs are usually the storage servers and file manager(s)

- The most important aspect concerning DFS processes is whether they should be stateless or stateful

1. Stateless Approach:

    - Does not require that servers maintain any client state
    - When a server crashes, there is no need to enter a recovery phase to bring the server to a previous state
    - Locking a file cannot be easily done
    - E.g., NFSv3 and PVFS (no client-side caching)

Processes (2)

2. Stateful Approach:

   - Requires that a server maintains some client state

   - Clients can make effective use of caches but this would entail an efficient underlying cache consistency protocol

   - Provides a server with the ability to support callbacks (i.e., the ability to do RPC to a client) in order to keep track of its clients

   - E.g., NFSv4 and HDFS

DFS Aspects

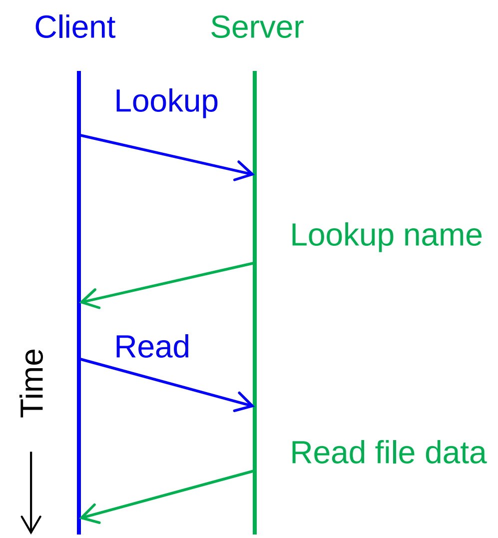
Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?

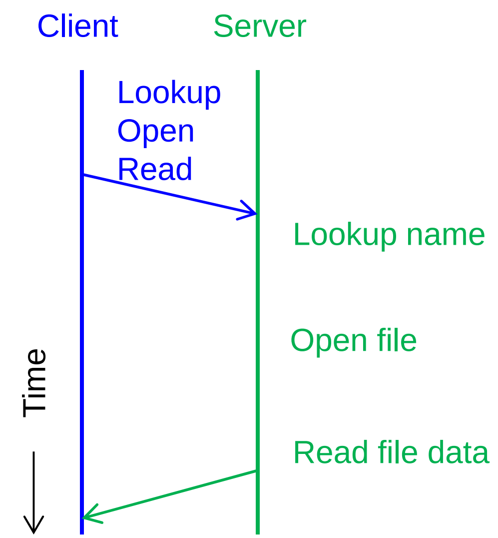
Communication

- Communication in DFSs is mainly based on remote procedure calls (RPCs)
- The main reason for choosing RPC is to make the system independent from underlying OSs, networks, and transport protocols
- In NFS, all communication between a client and server proceeds along the Open Network Computing RPC (ONC RPC)
- GFS uses RPC and may break a read into multiple RPCs to increase parallelism
- PVFS currently uses TCP for all its internal communication

RPCs in NFS

- Up until NFSv4, the client was made responsible for making the server's life as easy as possible by keeping requests simple
- The drawback becomes apparent when considering the use of NFS in a wide-area system
- In that case, the extra latency of a second RPC leads to performance degradation
- To circumvent such a problem, NFSv4 supports compound procedures





Next Class: DFS Aspects

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?
Fault Tolerance	How is fault tolerance handled in DFSs?