Cloud Computing
CS 15-319

Distributed File Systems and Cloud Storage – Part II

Lecture 13, Feb 27, 2012

Majd F. Sakr, Mohammad Hammoud and
Suhail Rehman

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

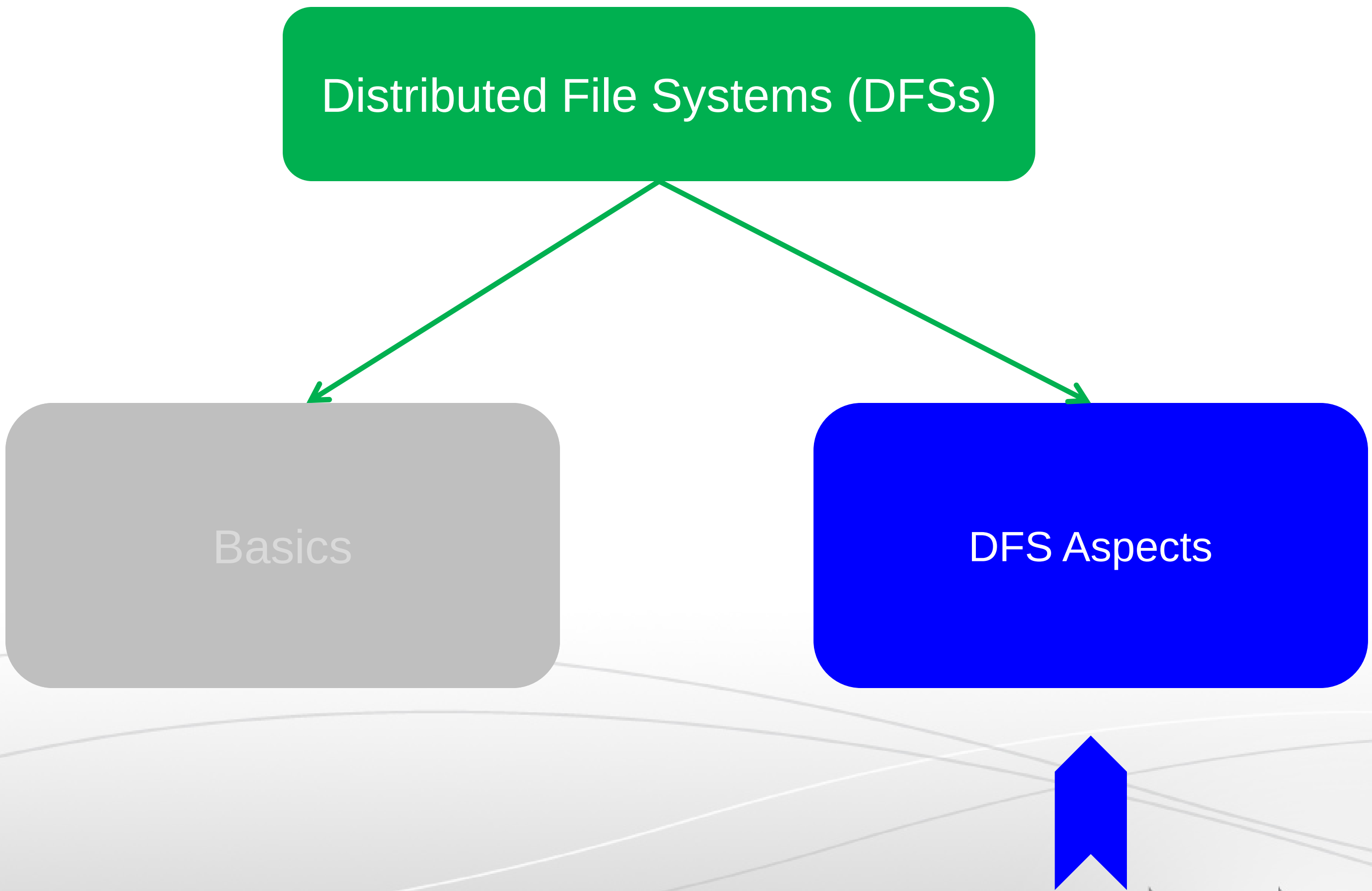
Today…

- Last session
  - Distributed File Systems and Cloud Storage- Part I

- Today's session
  - Distributed File Systems and Cloud Storage- Part II

- Announcement:
  - Project update is due next Wednesday, Feb 29

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?
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 typically 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)

- HDFS uses RPC for the communication between clients, DataNodes and the NameNode

- PVFS currently uses TCP for all its internal communication
  - The communication with I/O daemons and the manager is handled transparently within the API implementation

DFS Aspects

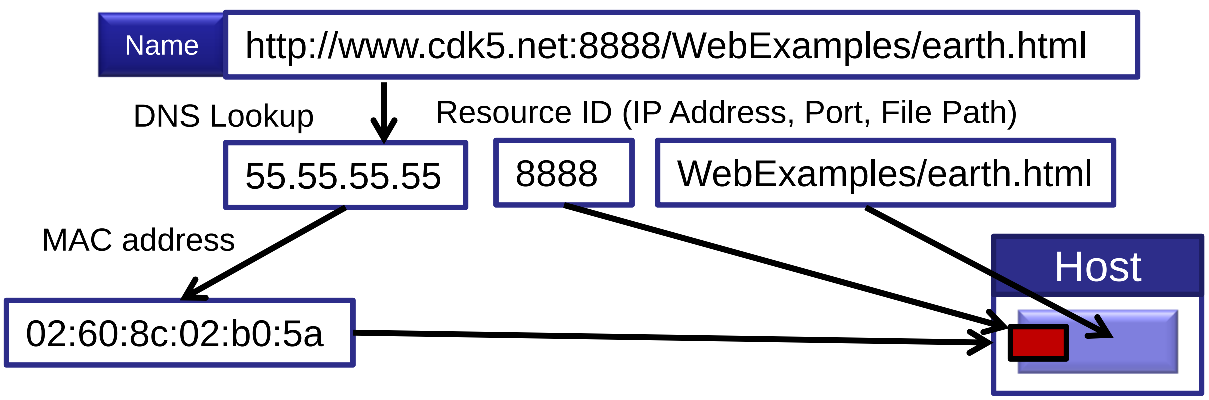
Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled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

Naming

- Names are used to uniquely identify entities in distributed systems
  - Entities may be processes, remote objects, newsgroups, …

- Names are mapped to an entity’s location using a *name resolution*

- An example of name resolution



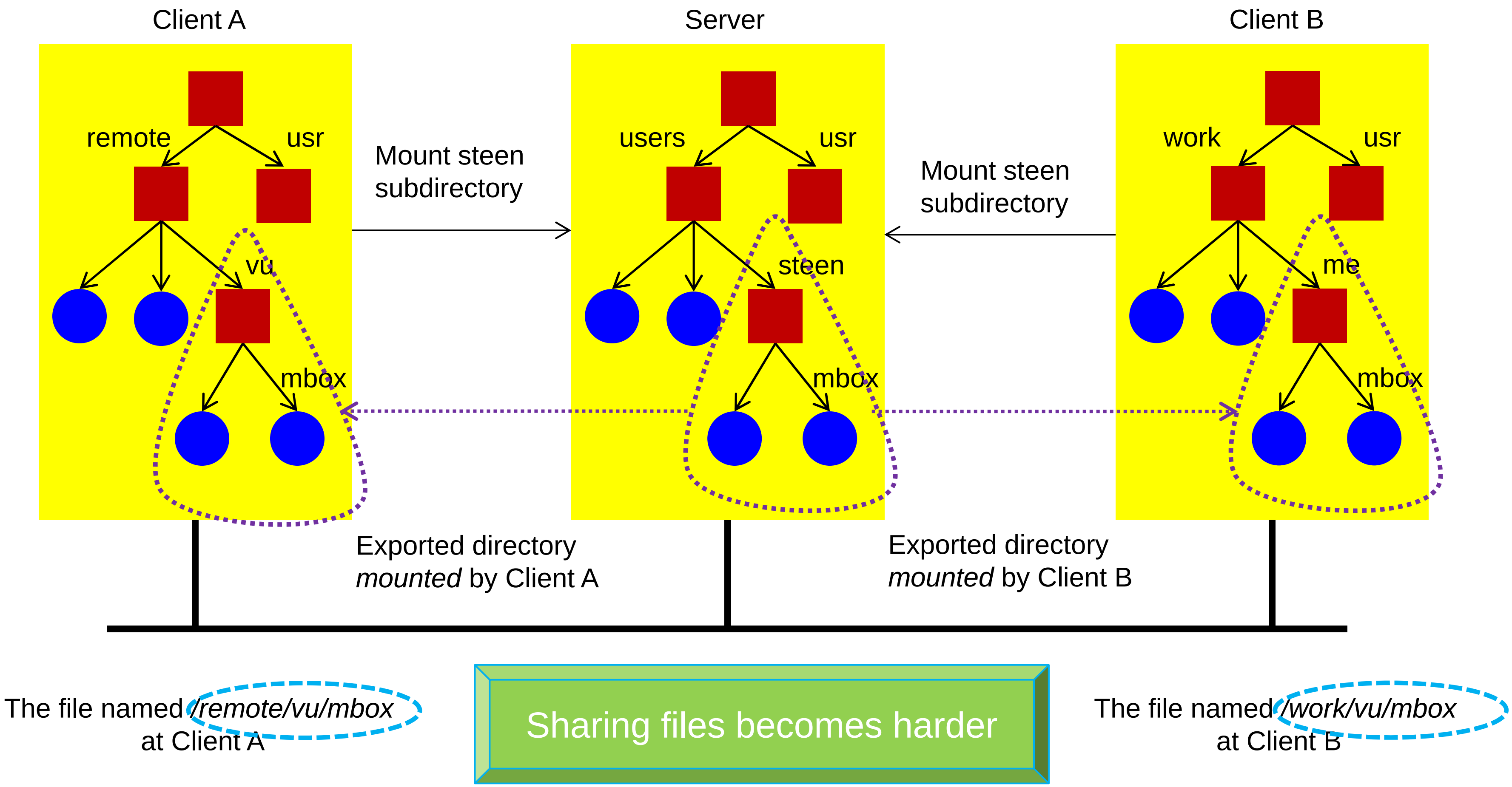
Naming In DFSs

NFS is considered as a representative of how naming is handled in DFSs

Naming In NFS

- The fundamental idea underlying the NFS naming model is to provide clients with complete transparency
- Transparency in NFS is achieved by allowing a client to *mount* a remote file system into its own local file system
- However, instead of mounting an entire file system, NFS allows clients to mount *only part* of a file system
- A server is said to export a directory to a client when a client mounts a directory, and its entries, into its own name space

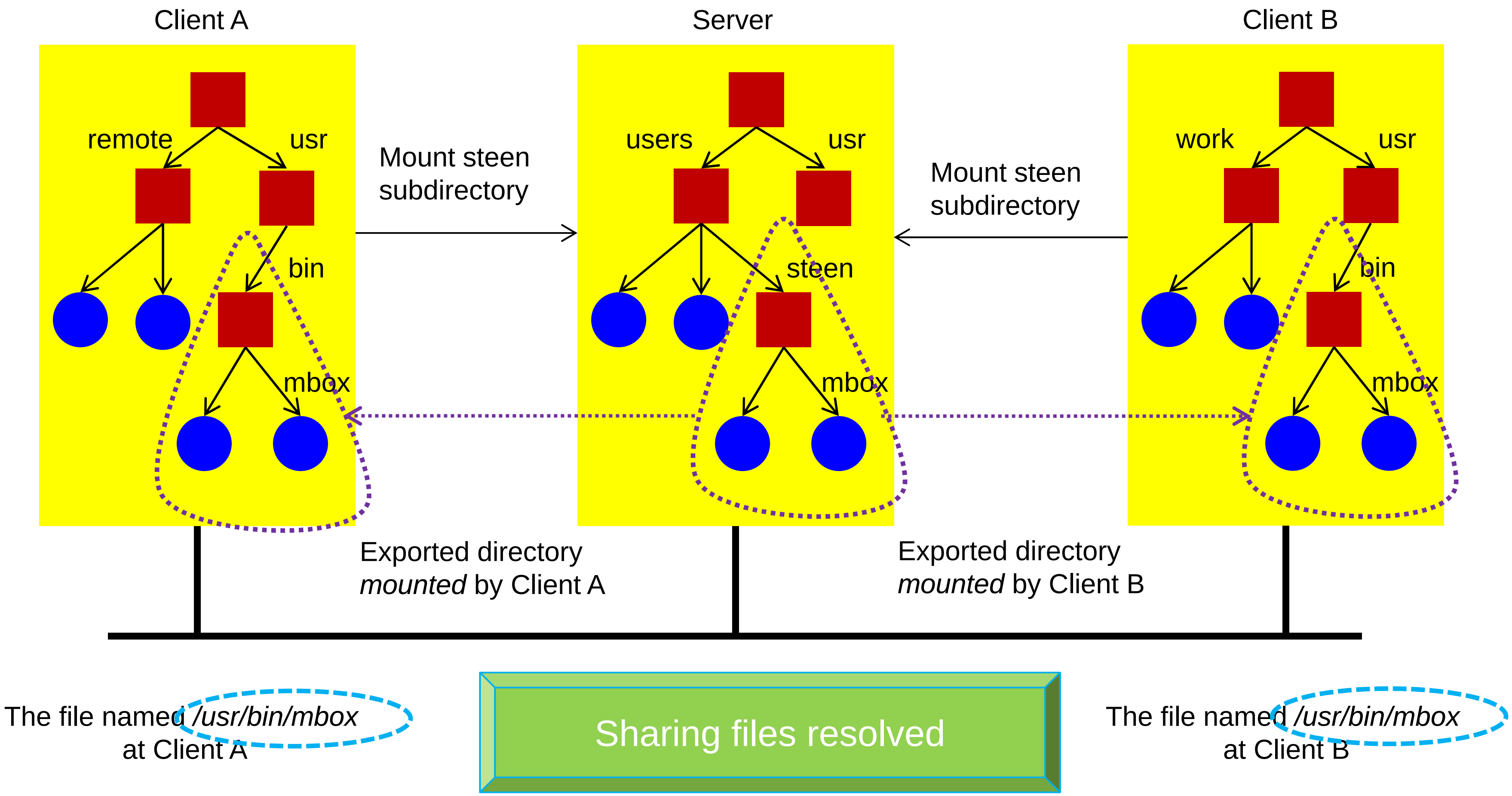
Mounting in NFS



Sharing Files In NFS

- A common solution for sharing files in NFS is to provide each client with a name space that is partly standardized
- For example, each client may by using the local directory /usr/bin to mount a file system
- A remote file system can then be mounted in the same manner for each user

Example


Client A
Server
Client B
remote
usr
bin
mbox
users
usr
steen
mbox
work
usr
bin
mbox
Mount steen subdirectory
Mount steen subdirectory
Exported directory *mounted* by Client A
Exported directory *mounted* by Client B
The file named */usr/bin/mbox* at Client A
Sharing files resolved
The file named */usr/bin/mbox* at Client B

Mounting Nested Directories In NFSv3

- An NFS server, ***S***, can itself mount directories, ***Ds***, that are exported by other servers

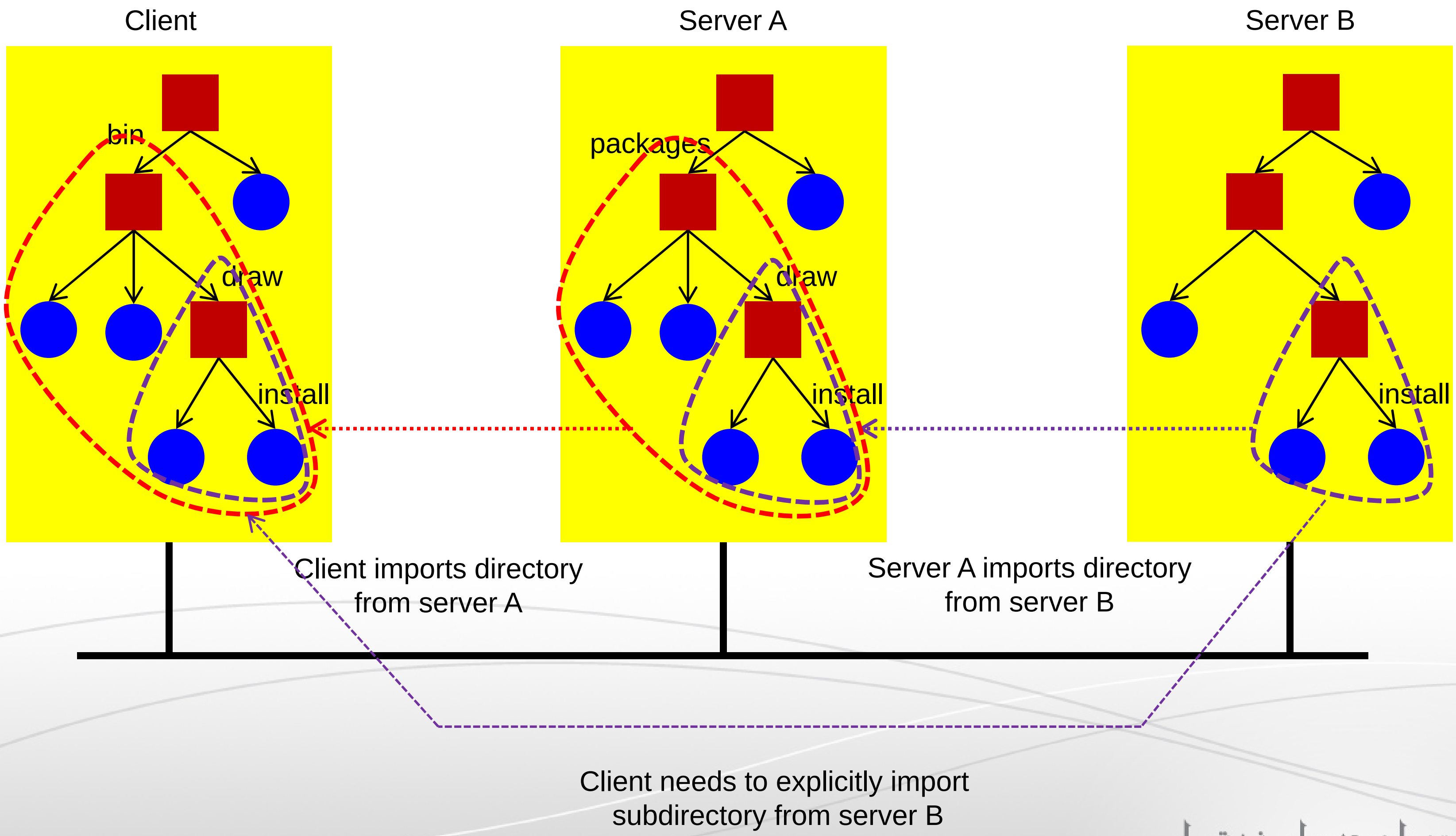
- However, in NFSv3, ***S*** is not allowed to export ***Ds*** to its own clients

- Instead, a client of ***S*** will have to explicitly mount ***Ds***

- If ***S*** will be allowed to export ***Ds***, it would have to return to its clients file handles that include identifiers for the exporting servers

- NFSv4 solves this problem

Mounting Nested Directories in NFS



جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

NFS: Mounting Upon Logging In (1)

- Another problem with the NFS naming model has to do with deciding *when* a remote file system should be mounted

- Example: Let us assume a large system with 1000s of users and that each user has a local directory /home that is used to mount the home directories of other users

  - Alice's (a user) home directory is made locally available to her as /home/alice

  - This directory can be automatically mounted when Alice logs into her workstation

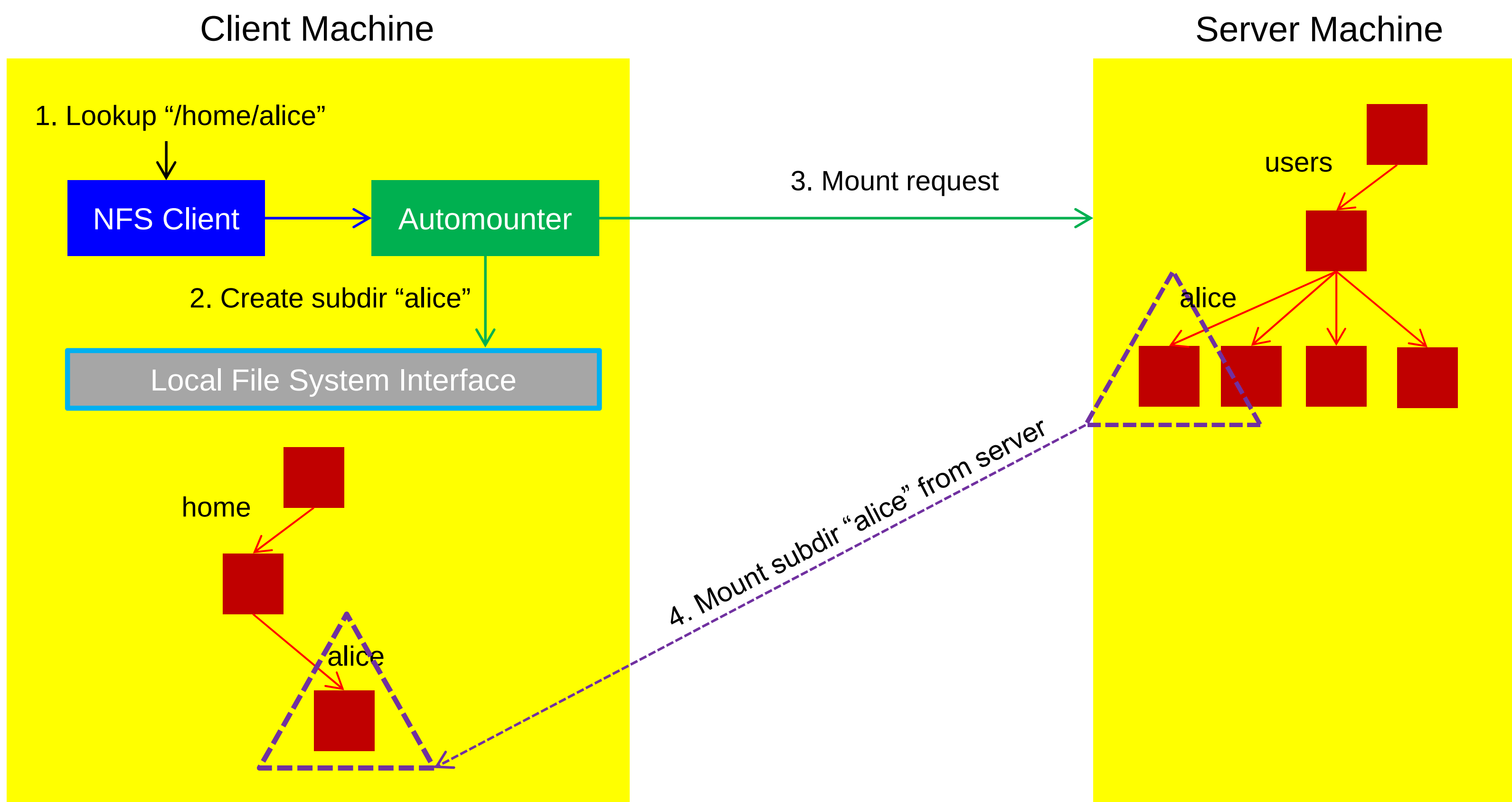
  - In addition, Alice may have access to Bob's (another user) public files by accessing Bob's directory through /home/bob

NFS: Mounting Upon Logging In (2)

- Example (*Cont'd*):
  - The question, however, is whether Bob's home directory should also be mounted *automatically* when Alice logs in
- If automatic mounting is followed for each user:
  - Logging in could incur a lot of communication and administrative overhead
  - All users should be known in advance
- A better approach is to transparently mount another user's home directory on-demand

On-Demand Mounting In NFS

- On-demand mounting of a remote file system is handled in NFS by an automounter, which runs as a separate process on the client's machine



Naming in HDFS

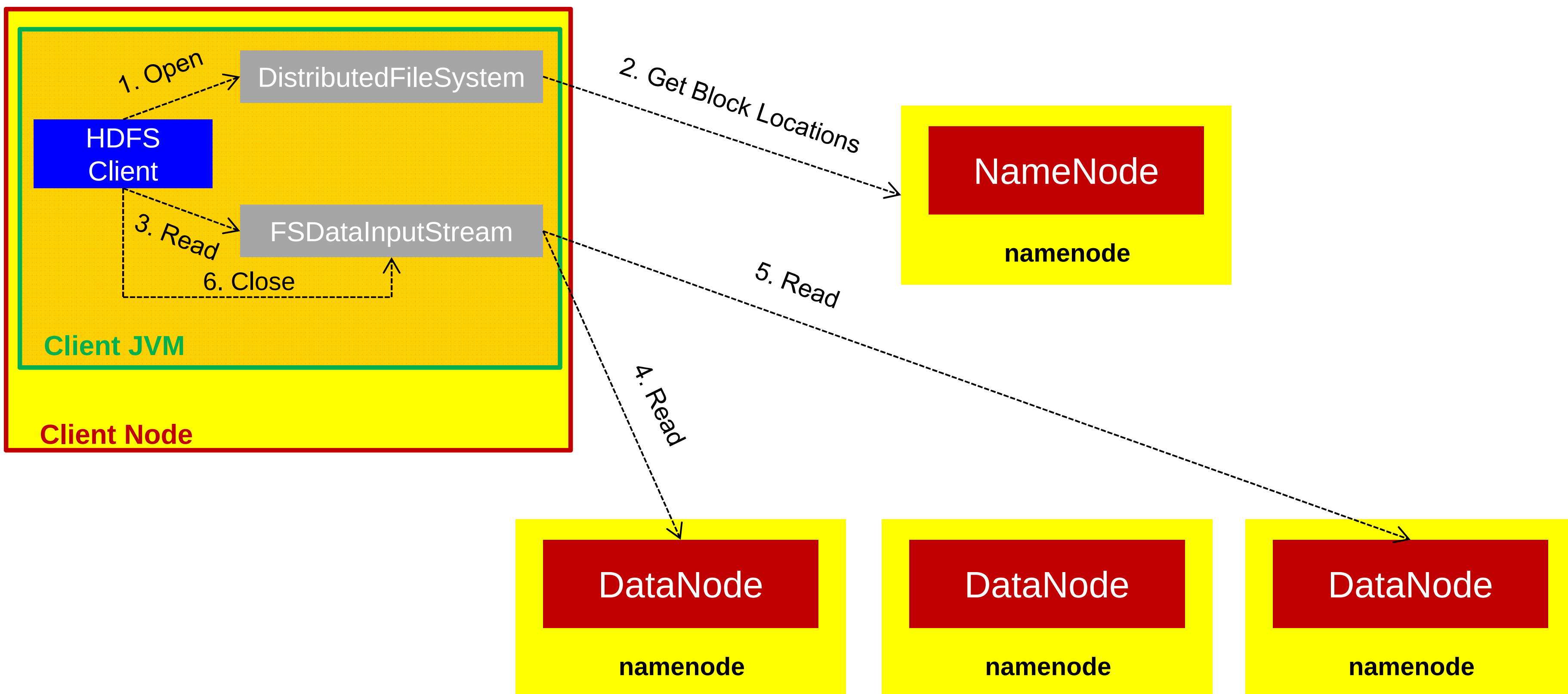
- An HDFS cluster consists of a single NameNode (the master) and multiple DataNodes (the slaves)

- The NameNode manages HDFS namespace and regulates accesses to files by clients
  - It executes file system namespace operations (e.g., opening, closing, and renaming files and directories)
  - It is an arbitrator and repository for all HDFS metadata
  - It determines the mapping of blocks to DataNodes

- The DataNodes manage storage attached to the nodes that they run on
  - They are responsible for serving read and write requests from clients
  - They perform block creation, deletion, and replication upon instructions from the NameNode

A Client Reading Data from HDFS

- Here is the main sequence of events when reading a file in HDFS

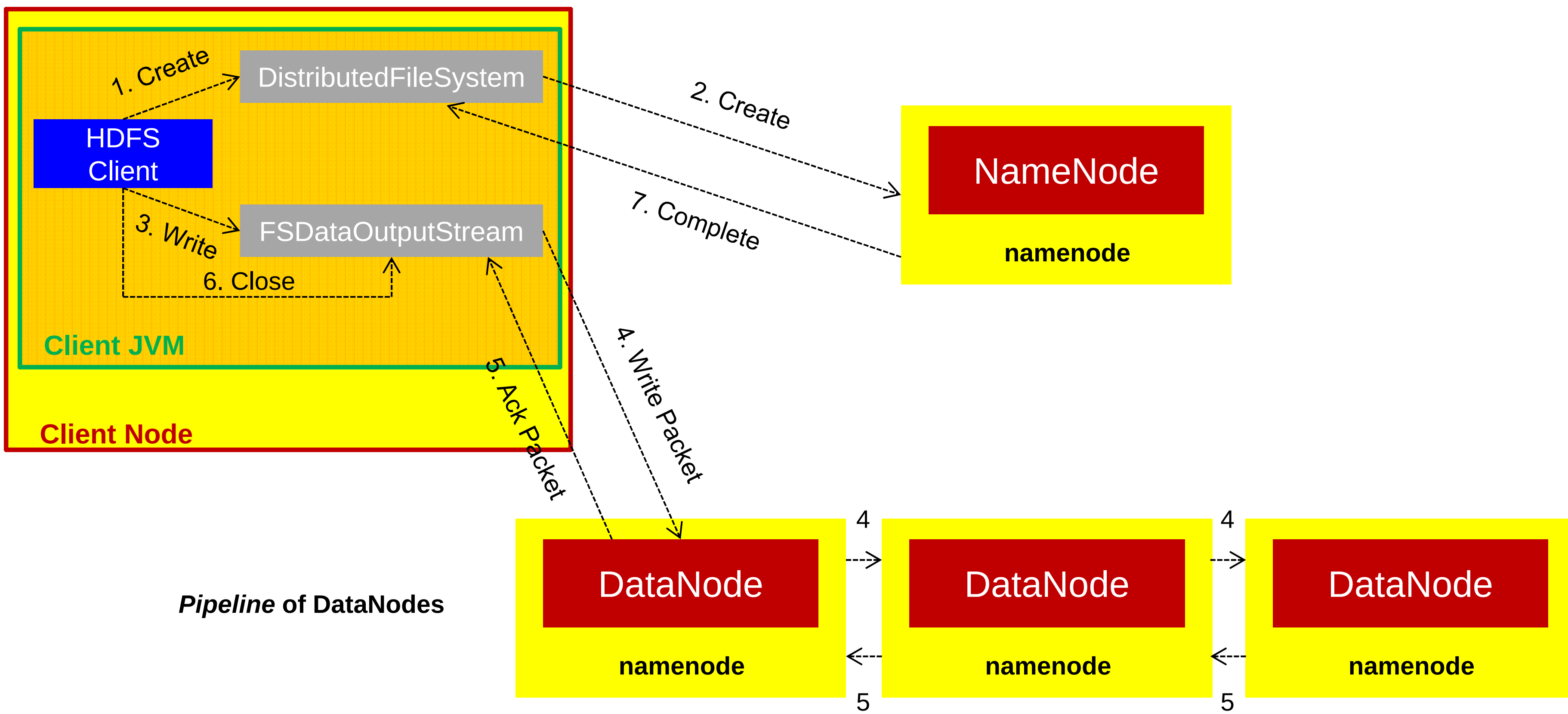


Data Reads

- DistributedFileSystem calls the NameNode, using RPC, to determine the locations of the blocks for the first few blocks in the file
- For each block, the NameNode returns the addresses of the DataNodes that have a copy of that block
- During the read process, DFSInputStream calls the NameNode to retrieve the DataNode locations for the next batch of blocks needed
- The DataNodes are sorted according to their proximity to the client in order to exploit data locality
- An important aspect of this design is that the client contacts DataNodes directly to retrieve data and is guided by the NameNode to the best DataNode for each block

A Client Writing Data to HDFS

- Here is the main sequence of events when writing a file to HDFS (the case assumes creating a new file, writing data to it, then closing the file)



Data Pipelining

- When a client is writing data to an HDFS file, its data is first written to a local file
- When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode
- The client then flushes the block to the first DataNode
- The first DataNode:
  - Starts receiving the data in small portions (4KB)
  - Writes each portion to its local repository
  - Transfers that portion to the subsequent DataNode in the list
- A subsequent DataNode follows the same steps as the previous DataNode
  - Thus, the data is pipelined from one DataNode to the next

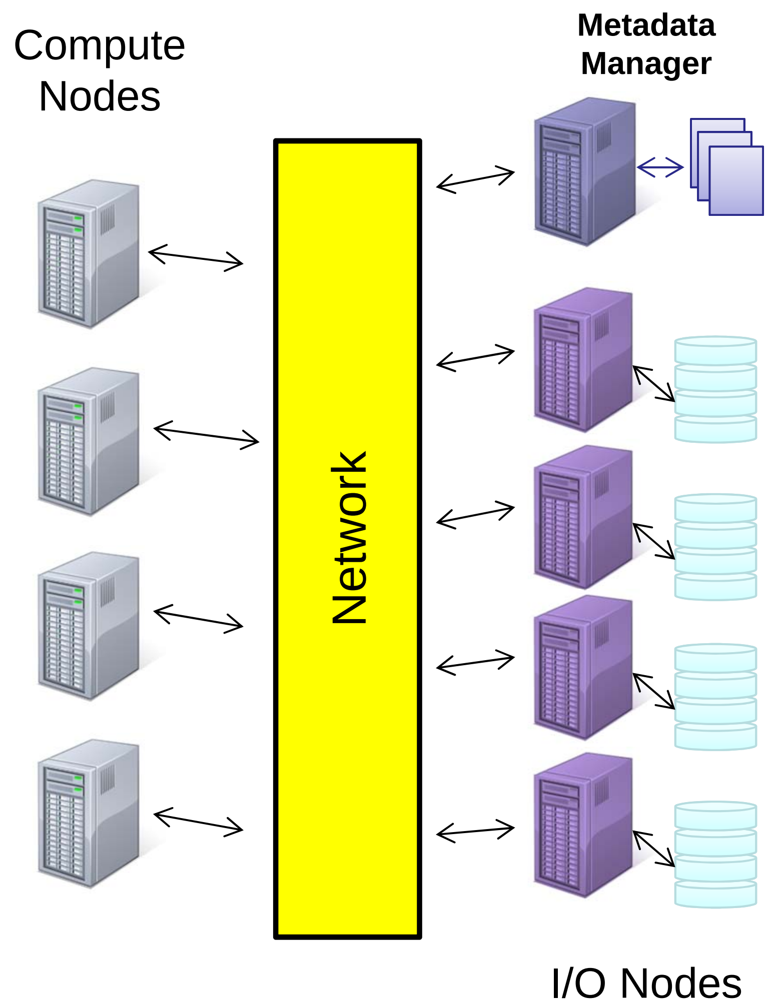
PVFS System View

- Some major components of the PVFS system:
  - Metadata server (mgr)
  - I/O server (iod)
  - PVFS native API (libpvfs)

- The mgr manages all file metadata for PVFS files

- The iods handle storing and retrieving file data stored on local disks connected to the node

- Libpvfs:
  - provides user-space access to the PVFS servers
  - handles the scatter/gather operations necessary to move data between user buffers and PVFS servers

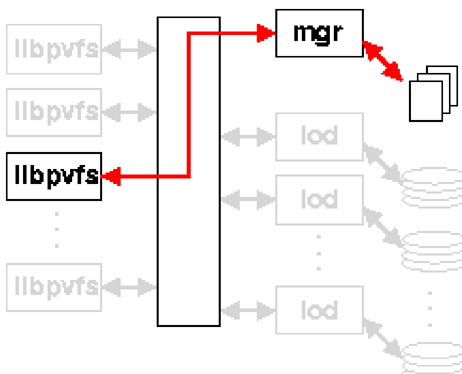


Naming in PVFS

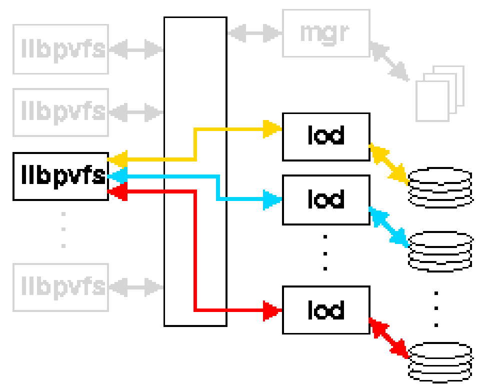
- PVFS file systems may be mounted on all nodes in the same directory simultaneously
- This allows all nodes to see and access all files on the PVFS file system through the same directory scheme
- Once mounted, PVFS files and directories can be operated on with all the familiar tools, such as ls, cp, and rm
- With PVFS, clients can avoid making requests to the file system through the kernel by linking to the PVFS native API
  - This library implements a subset of the UNIX operations which directly contact PVFS servers rather than passing through the local kernel

Metadata and Data Accesses

- For metadata operations, applications communicate through the library with the metadata server
- For data access, the metadata server is eliminated from the access path and instead I/O servers are contacted directly



Metadata Access



Data Access

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?