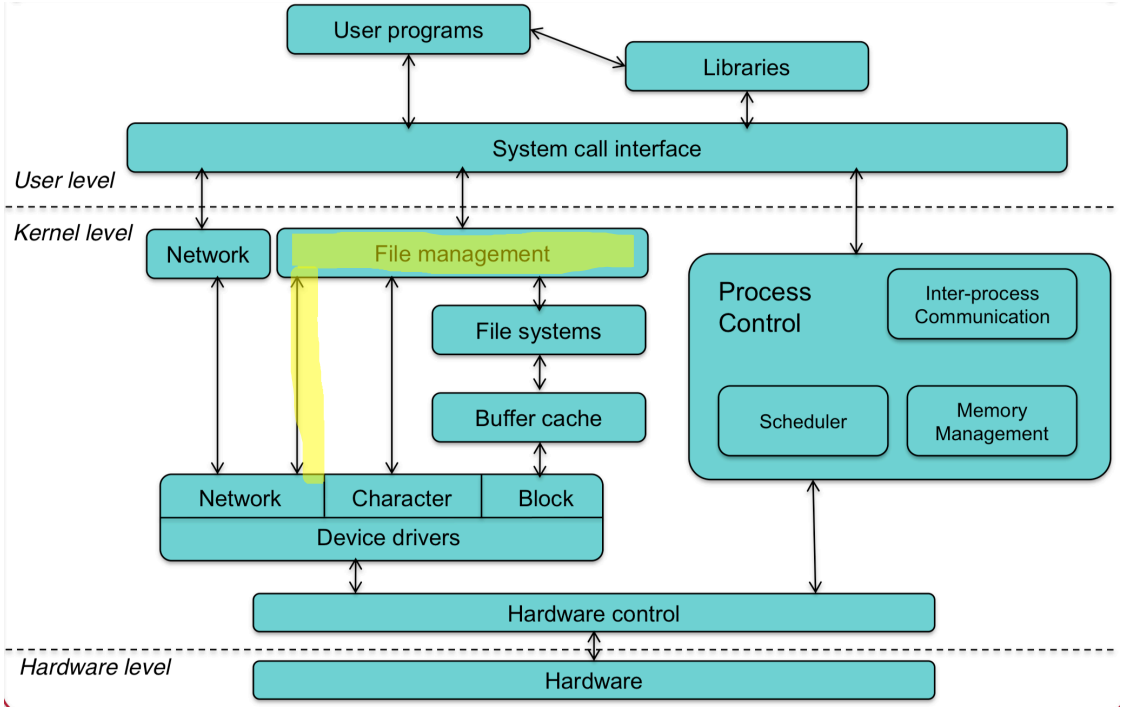
# LS-13:
# DISTRIBUTED FILE SYSTEMS



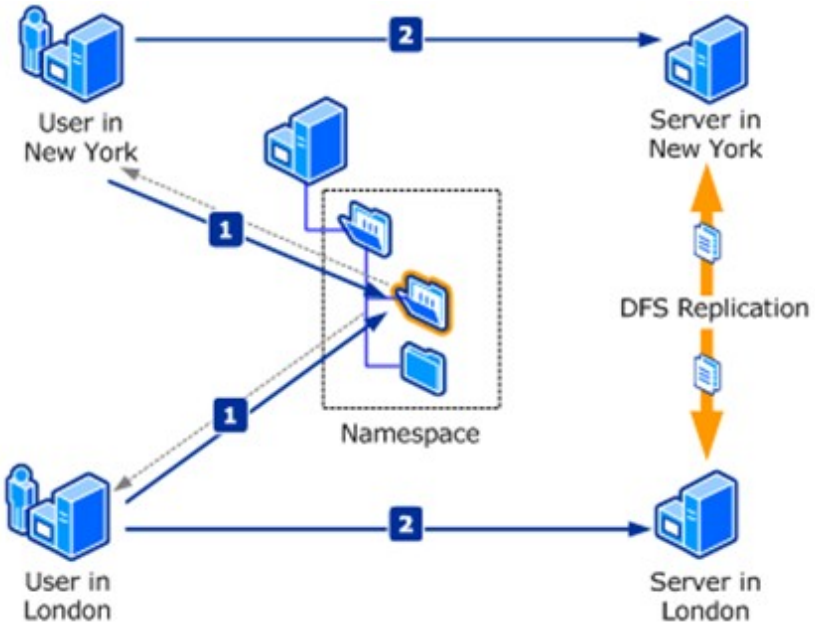
# Distributed (or Networked) File Systems

Distributed (Networked) File Systems

- Allow a centralized file server to export a file-system to multiple clients.
- Redirecting user to the right copy of data.
- Provide file level access, and raw blocks access.
- Clustered file-systems also exist, where multiple servers work in together.

DFS Architectures

- as Client-Server Architecture (Centralized)
  - NFS (Network File System)
  - CIFS/SMB
    (Windows Common Internet FS/Samba protocol)
  - Andrew FS
- as Cluster-Based Arch (Less Centralized)
  - GFS (Global File System, Google FS)
- as Symmetric Arch (Fully Distributed)
  - DHT-based (Distributed Hash Table)




- Comparison of distributed file systems

https://en.wikipedia.org/wiki/Comparison_of_distributed_file_systems

# Distributed FS: Client-Server

- A Client-Server distributed file system enables clients to access files stored on one or more remote file servers.
- Two Models:

1. Upload/Download Model (entire files)
2. Remote Access Model (remote file operations – RPC)

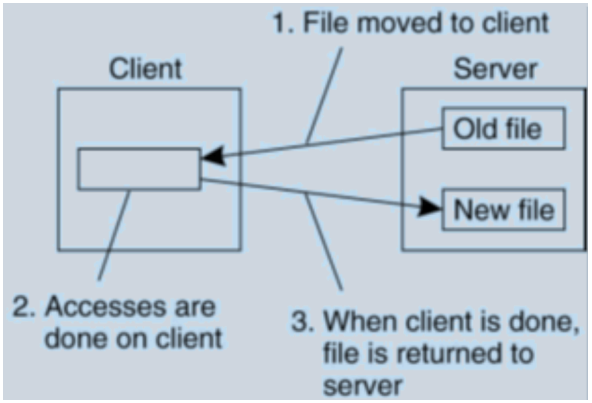
NFS /2, /3, /4

- Developed by Sun in the 1984.
- NFS is OS-independent FS: client and server.
- Client requests are implemented as remote procedure calls (RPCs).
- Stateless. Means server and client can reboot without the other noticing.
- A server, nfsd, exports filesystems as described in /etc/exports.
- The server can be in user-space or in the kernel-space.

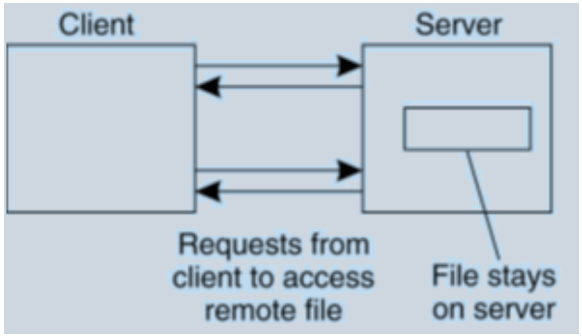
CIFS/SMB

- CIFS – Windows Common Internet File System.
- Poorly documented.
- SMB (or Samba) reimplements it, originally reverse-engineered.

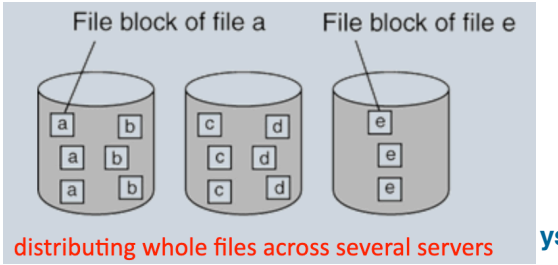
- For speeding file access use Simple Distributing files across servers →




1. The Upload/Download Model

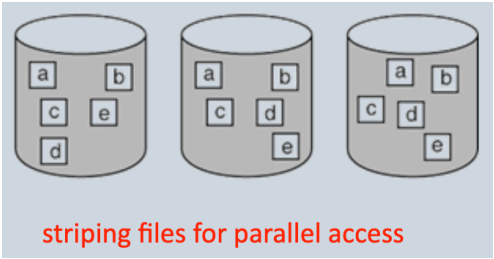



2. The Remote Access Model



# Distributed FS: Cluster based

- With very large data collections, a simple client-server approach is not going to work.
- For speeding up file accesses, apply striping techniques by files can be fetched in parallel.

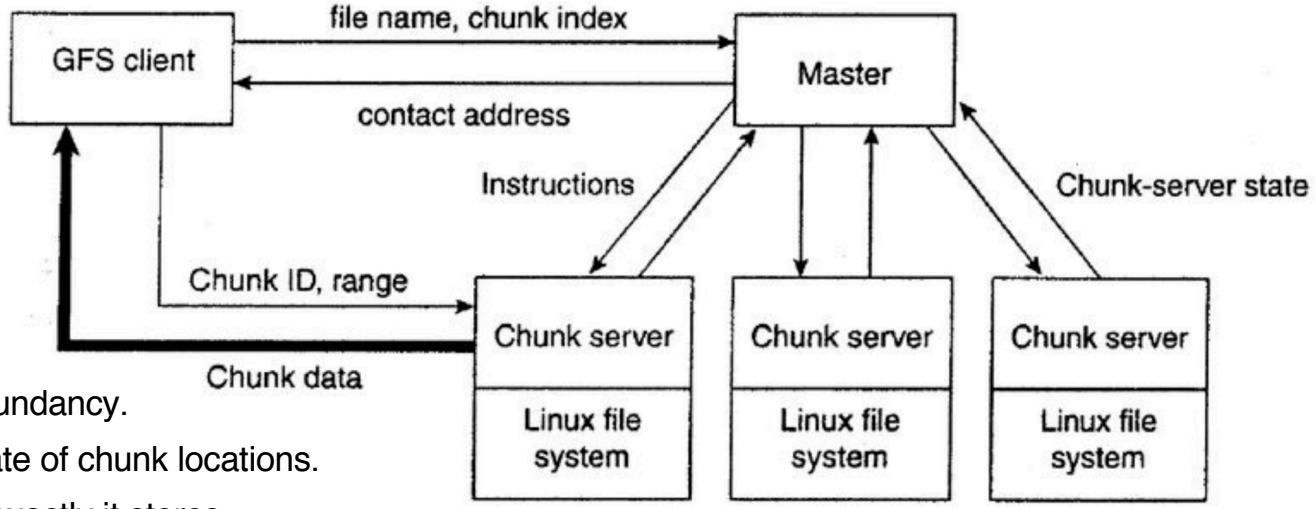


striping files for parallel access

Example of GFS: Google FS

How does it work?

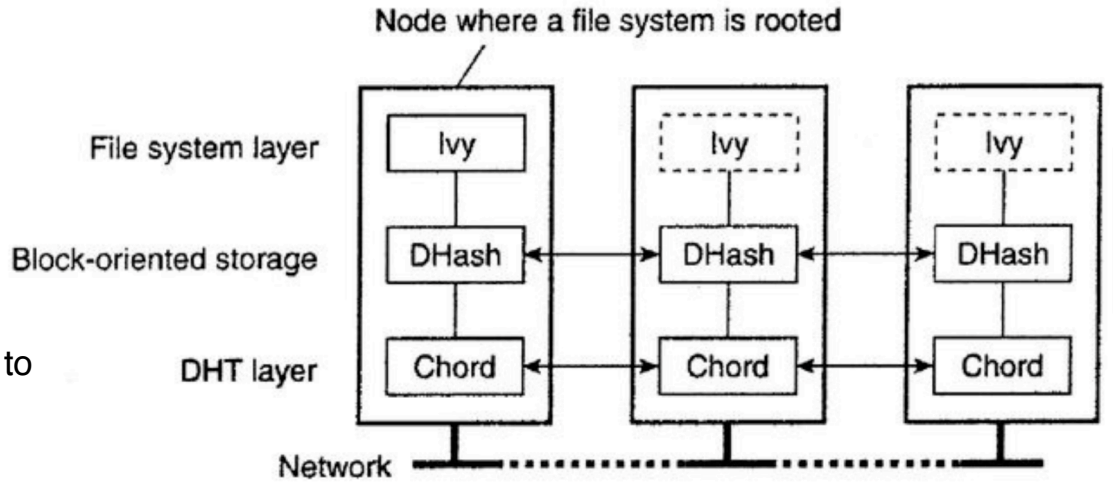
- A cluster has a master node, which ONLY keeps meta information of files
- A big file is splited into CHUNKS, a CHUNK of size 64Mbs.
- Chunks are spread on many chunk servers
- More details on GFS:
  - Chunks are replicated for Redundancy.
  - Master does not keep up-to-date of chunk locations.
  - A Chunks server knows what exactly it stores.
  - If client retrieval failed(low probability), ask Master again, and master update latest info from chunk servers.
  - File update. Client pushes back updated file chunk to corresponding chunk server.
  - Chunk server conducts the backup/replication.
  - Master node is kept out of this loop, bottle neck problem is solved.
  - I/O Performance of a GFS is pretty good and Scalability is good as well.



# Distributed FS: Symmetric

Example of DHS: Ivy

- Peer-to-Peer. Multi user Read/Write.
- NoClient, NoServer, NoMaster, NoChunk.
- First realization is Ivy.

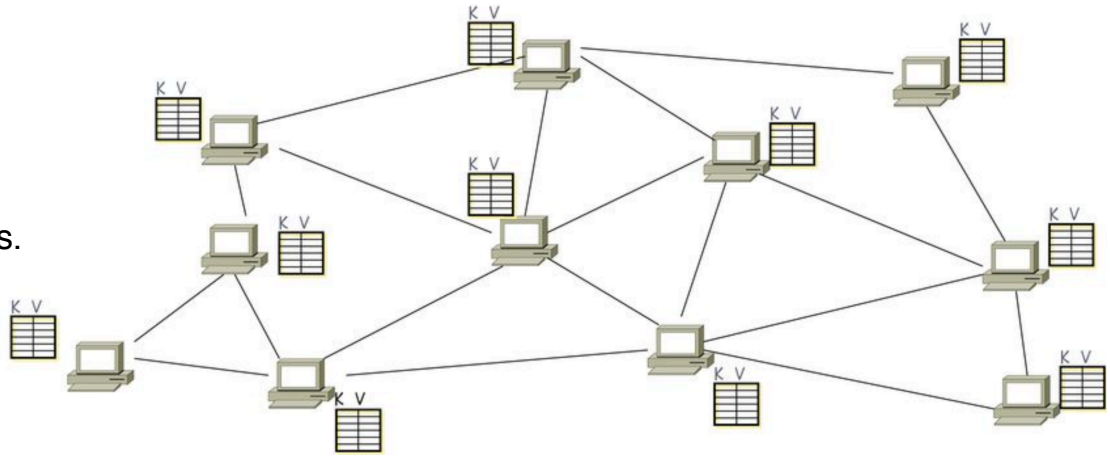



**What is DHT?**

- Hash Table - data structure that maps "keys" to "values".
- Distributed Hash Table (DHT) - similar, but spread across many hosts.
- Interface
  - insert(key, value)
  - lookup(key)

**Ivy details**

- Data storage. File composed of 8kb data blocks.
  - Content-hash data blocks
  - Public-key based blocks
- Replication
  - Every block B is stored on K immediate successors, better availability



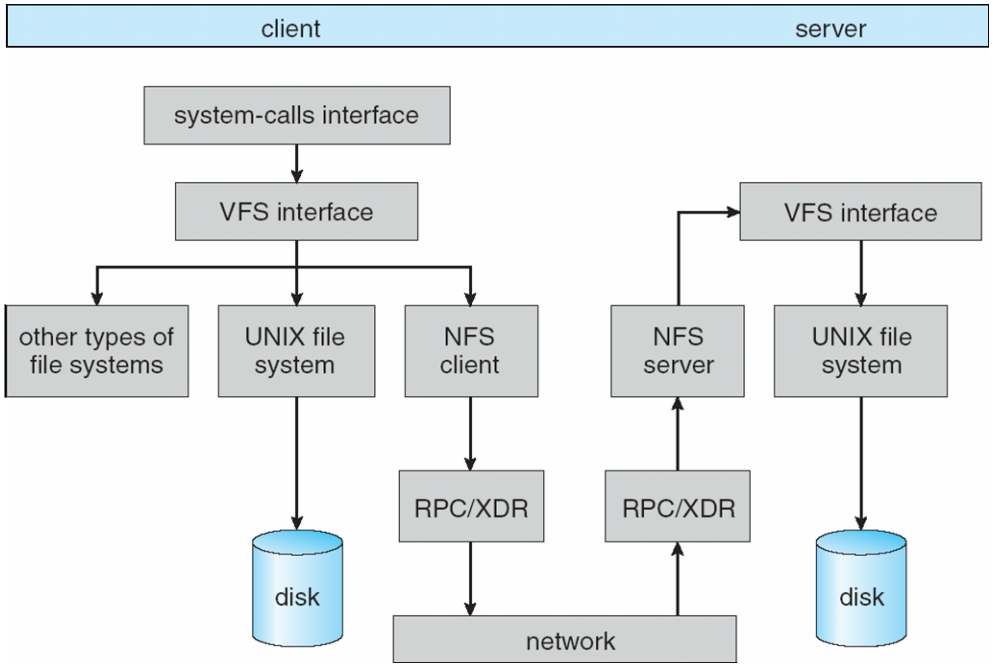
Basic Idea. Operation: take key as input: route messages to node holding key

# Distributed FS: Ex.1. NFS Architecture

- NFS - Network File System, first implemented on SUN Solaris.
- NFS is the predominant DFS implementation on Unix System, used for accessing remote files across LANs (or WANs) with using an unreliable datagram protocol (UDP/IP protocol and Ethernet).
- This independence is achieved through the use of **RPC** primitives built on top of an External Data Representation (**XDR**) protocol used between two implementation-independent interfaces.

**Three Major Layers of NFS Architecture**

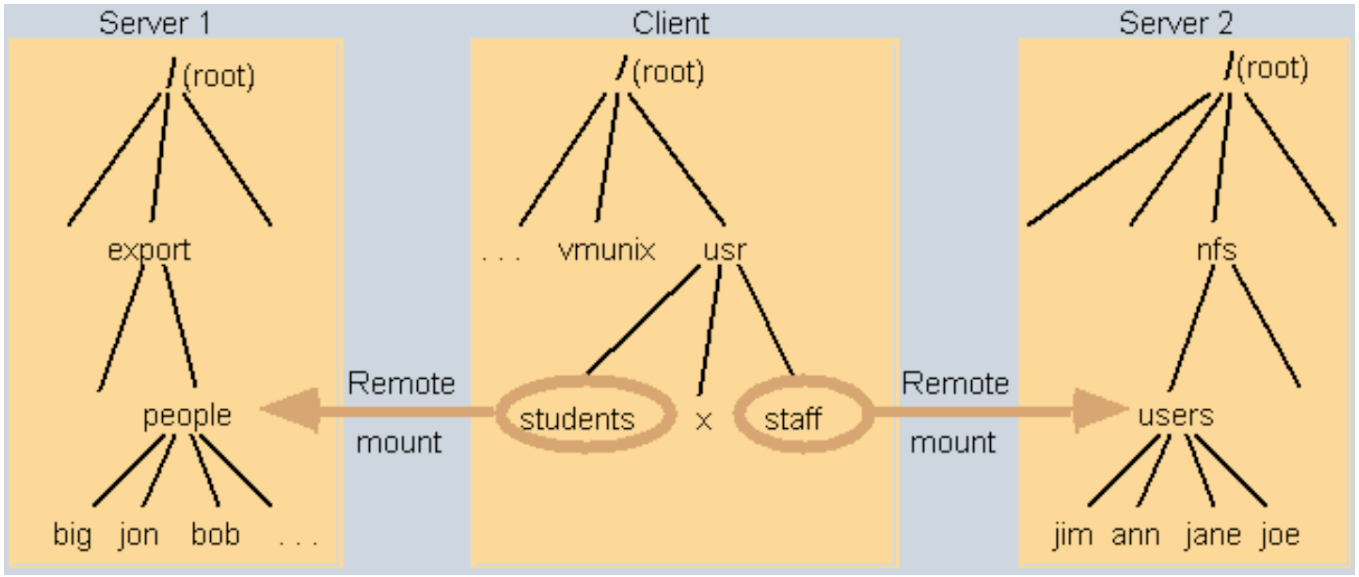
1. Unix FS Interface layer based on the **open, read, write**, and **close** calls, and **file descriptors.**
2. VFS Layer
   - VFS layer – distinguishes local and remote files.
   - VFS provides a standard file system interface, hides difference between accessing local and remote file systems.
     - The VFS activates file-system-specific operations to local requests.
     - Calls the NFS protocol procedures for remote requests.
3. NFS service layer – implements the NFS protocols.
     - NFS have mount mechanism services and the actual remote-file-access services.



# Distributed FS: Ex.1. NFS Mount Protocol

- NFS establishes initial logical connection between server and client
- Mount operation includes local mount point, name of remote directory to be mounted and name of server machine storing it.
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

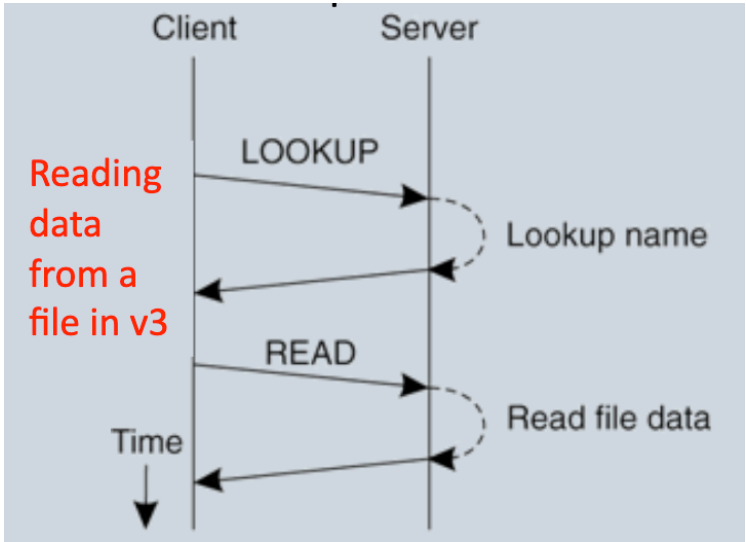
**mount –t nfs Server1:/export/people /usr/students**
**mount –t nfs Server2:/nfs/users /usr/staff**



# Distributed FS: Ex.1. NFS Remote Operations

RPC - remote procedure calls.

- NFS provides a set of RPC's for remote file operations.
- NFS servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- NFS use to the remote-service paradigm, but use buffering and caching techniques for the sake of performance
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server




- NFS RPC's operation:
  - read(fh, offset, count)
  - write(fh, offset, count, data)
  - create(dirfh, name, attr)
  - remove(dirfh, name)
  - getattr(fh)
  - setattr(fh, attr)
  - lookup(dirfh, name)
  - rename(dirfh, name, todirfh, toname)
  - link(newdirfh, newname, dirfh, name)
  - readdir(dirfh, cookie, count)
  - symlink(newdirfh, newname, string)
  - readlink(fh)
  - mkdir(dirfh, name, attr)
  - rmdir(dirfh, name)
  - statfs(fh)

# END