Distributed File Systems

Thoai Nam

High Performance Computing Lab (HPC Lab)
Faculty of Computer Science and Engineering
HCMC University of Technology

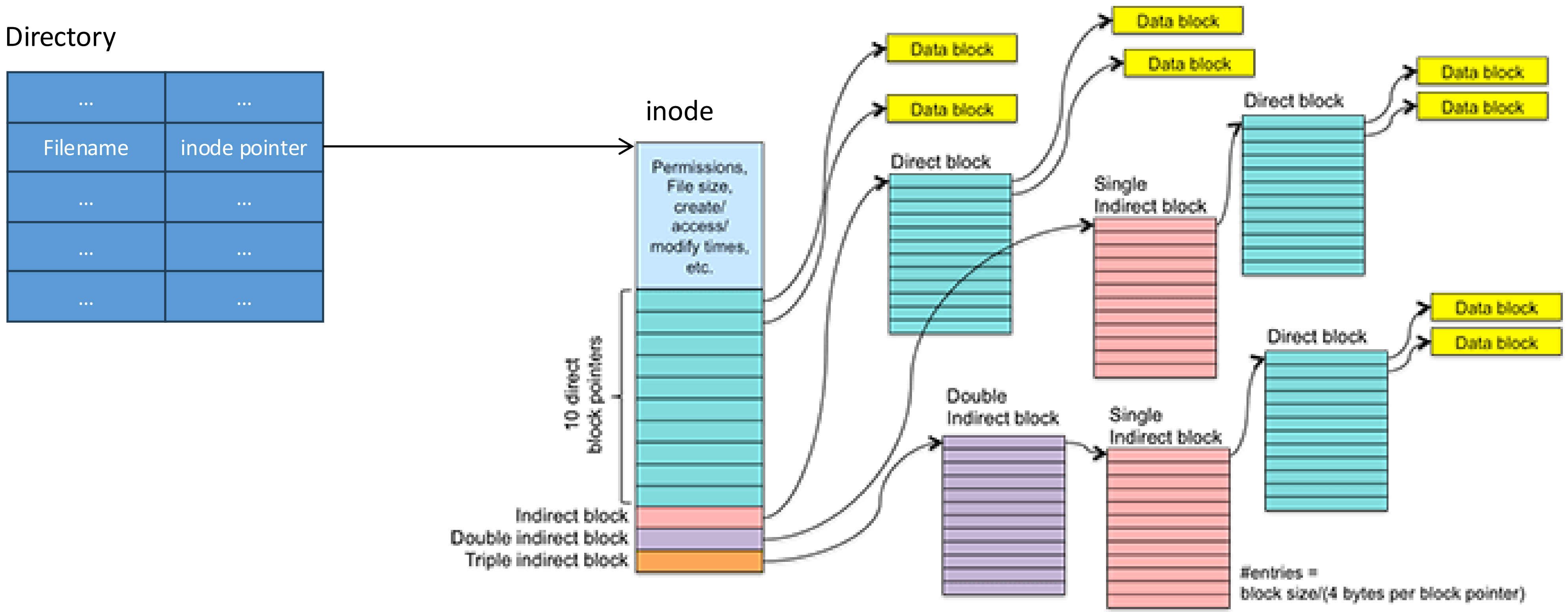
Contents

- Distributed File System architecture
- NFS, HDFS
- ...

What is a file system?

- Persistent stored data sets
- Hierarchical name space visible to all processes
- API with the following characteristics:
  - access and update operations on persistently stored data sets
  - Sequential access model (with additional random facilities)
- Sharing of data between users, with access control
- Concurrent access:
  - certainly for read-only access
  - what about updates?
- Other features:
  - mountable file stores
  - more? ...

Unix file system?



File attribute record structure

updated by system:

- File length
- Creation timestamp
- Read timestamp
- Write timestamp
- Attribute timestamp
- Reference count

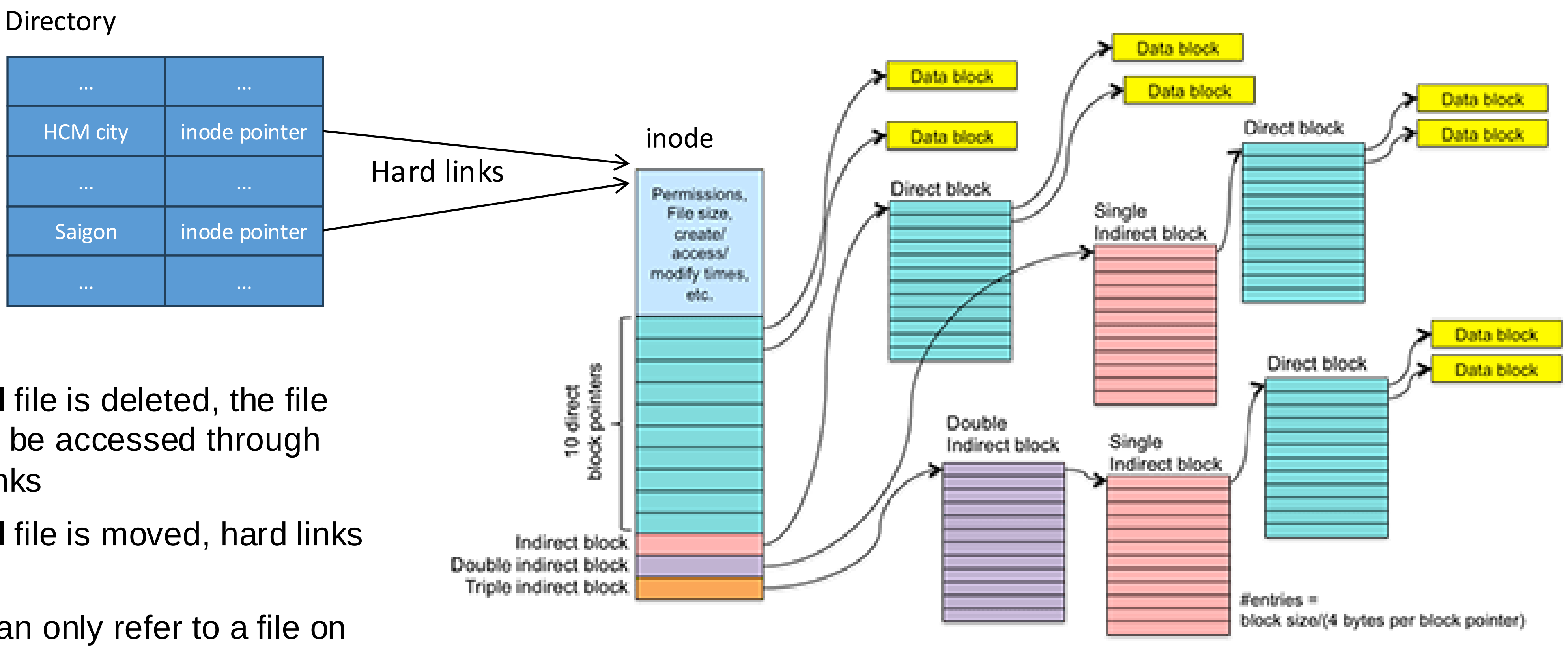
updated by owner:

- Owner
- File type
- Access control list

  E.g. for UNIX: `rw-rw-r--`

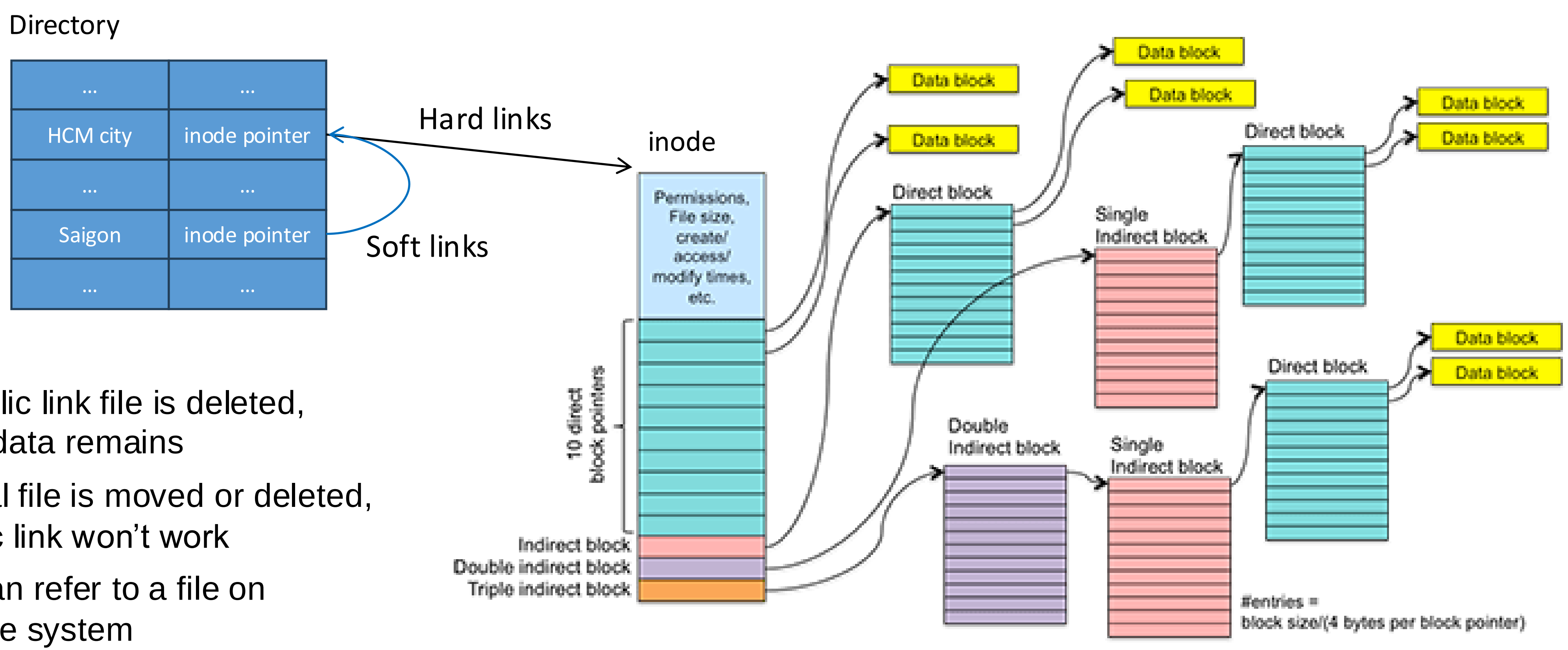
Link count

Hard links



- If the original file is deleted, the file data can still be accessed through other hard links
- If the original file is moved, hard links still work
- A hard link can only refer to a file on the same file system
- The inode and file data are permanently deleted when the number of hard links is zero.

Soft/Symbolic links (symlinks)



- If the symbolic link file is deleted, the original data remains
- If the original file is moved or deleted, the symbolic link won't work
- A soft link can refer to a file on a different file system
- Soft links are often used to quickly access a frequently-used file without typing the whole location.

Hard & Soft links

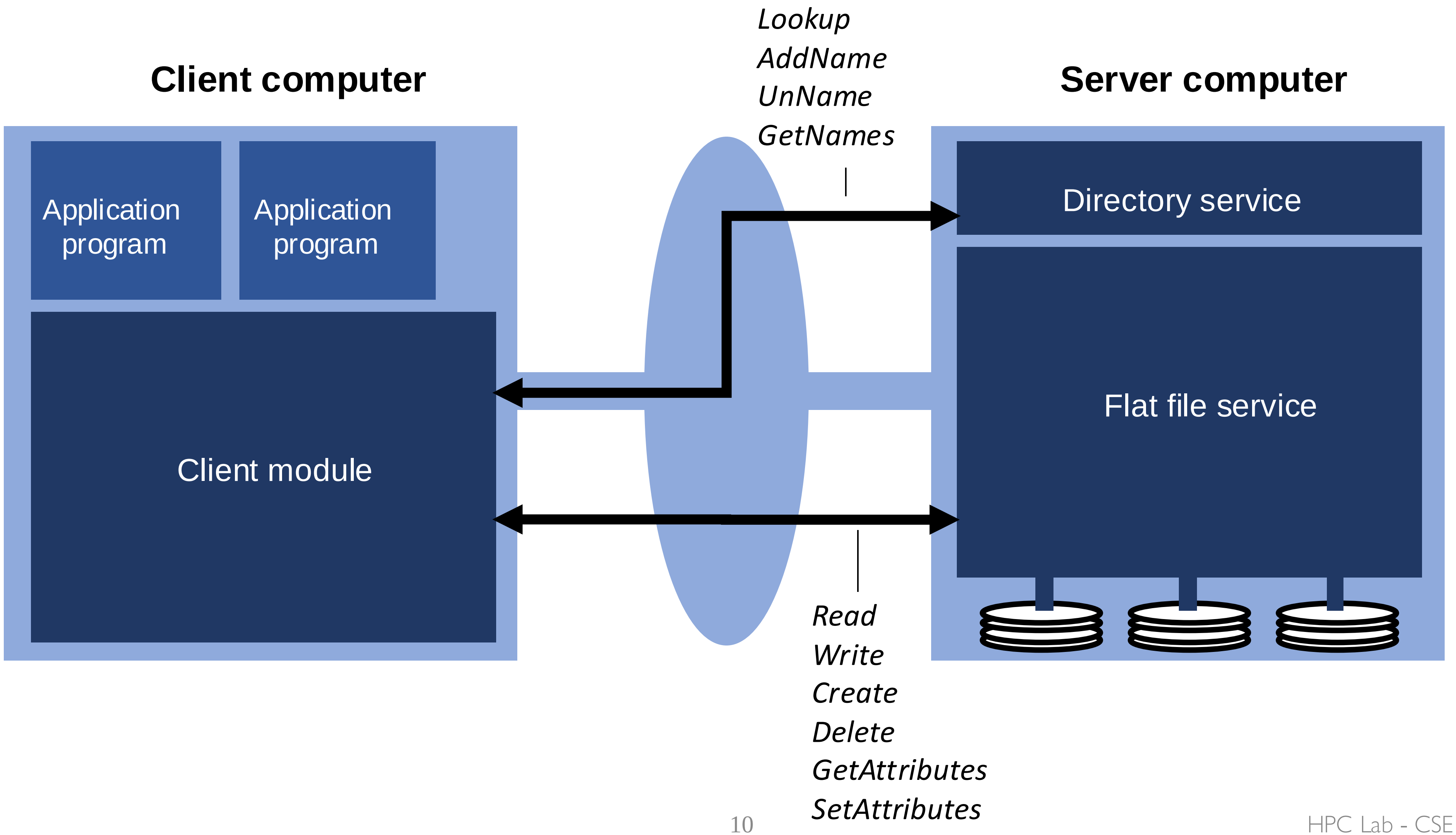
- A soft link does not contain the data in the target file
- A soft link points to another entry somewhere in the file system
- A soft link has the ability to link to directories, or to files on remote computers networked through NFS
- Deleting a target file for a symbolic link makes that link useless
- A hard link preserves the contents of the file
- A hard link cannot be created for directories, and they cannot cross filesystem boundaries or span across partitions
- In a hard link, you can use any of the hard link names created to execute a program or script in the same manner as the original name given.

Operations

UNIX file system operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

Model file service architecture



Server operations for the model file service

Flat file service

Position of first byte

Read(FileId, i, n) -> Data

Write(FileId, i, Data)

Create() -> FileId

Delete(FileId)

GetAttributes(FileId) -> Attr

SetAttributes(FileId, Attr)

Directory service

Lookup(Dir, Name) -> FileId

FileId

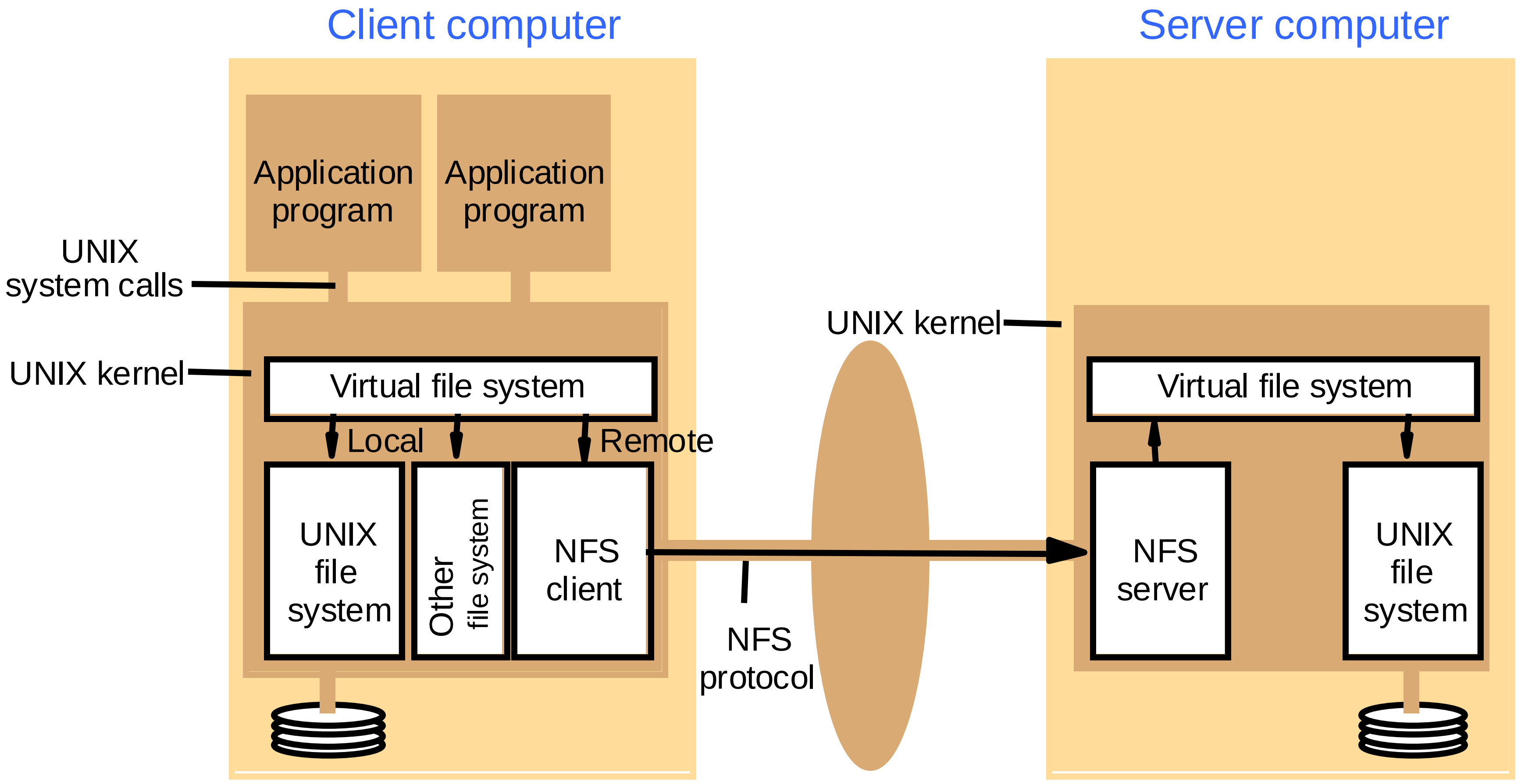
AddName(Dir, Name, ~~File~~)

UnName(Dir, Name)

GetNames(Dir, Pattern) -> NameSeq

Network File System - NFS

NFS



- The Network File System (NFS) was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive
- This allows for fast, seamless sharing of files across a network.

NFS server operations (1)

lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirth*
create(dirfh, name, attr) -> newfh, attr	Creates a new file *name* in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) -> status	Removes file *name* from directory *dirfh*.
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) -> attr	Writes *count* bytes of *data* to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

NFS server operations (2)

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

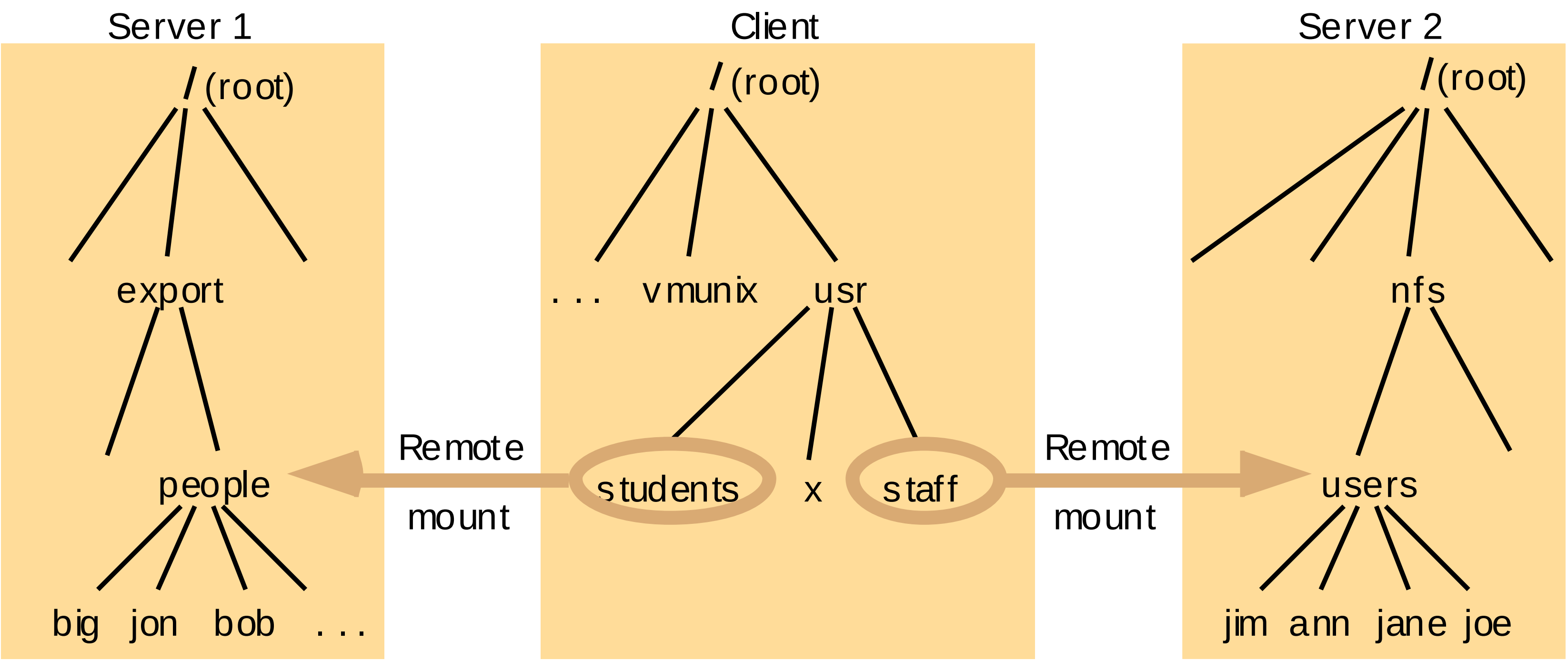
NFS overview

- Remote Procedure Calls (RPC) for communication between client and server
- Client Implementation
  - Provides transparent access to NFS file system
    - UNIX contains Virtual File system layer (VFS)
    - Vnode: interface for procedures on an individual file
  - Translates Vnode operations to NFS RPCs
- Server Implementation
  - Stateless: Must not have anything only in memory
  - Implication: All modified data written to stable storage before return control to client
    - Servers often add NVRAM to improve performance

Mapping Unix system calls NFS operations

- Unix system call: `fd = open("/dir/foo")`
  - Traverse pathname to get filehandle for foo
    - `dirfh = lookup(rootdirfh, "dir");`
    - `fh = lookup(dirfh, "foo");`
  - Record mapping from `fd` file descriptor to `fh` NFS file handle
  - Set initial file offset to 0 for `fd`
  - Return `fd` file descriptor
- Unix system call: `read(fd,buffer,bytes)`
  - Get current file offset for `fd`
  - Map `fd` to `fh` NFS filehandle
  - Call data = `read(fh, offset, bytes)` and copy data into buffer
  - Increment file offset by bytes
- Unix system call: `close(fd)`
  - Free resources associated with `fd`

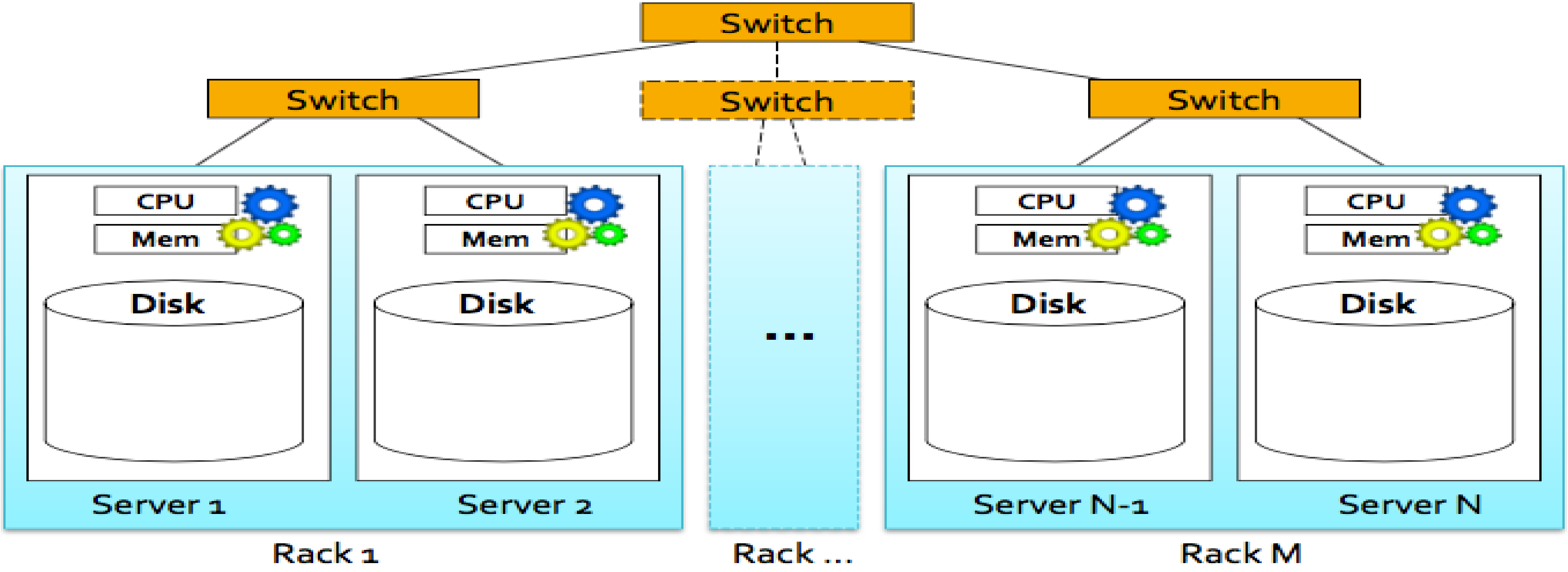
Local and remote file systems accessible on an NFS client



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

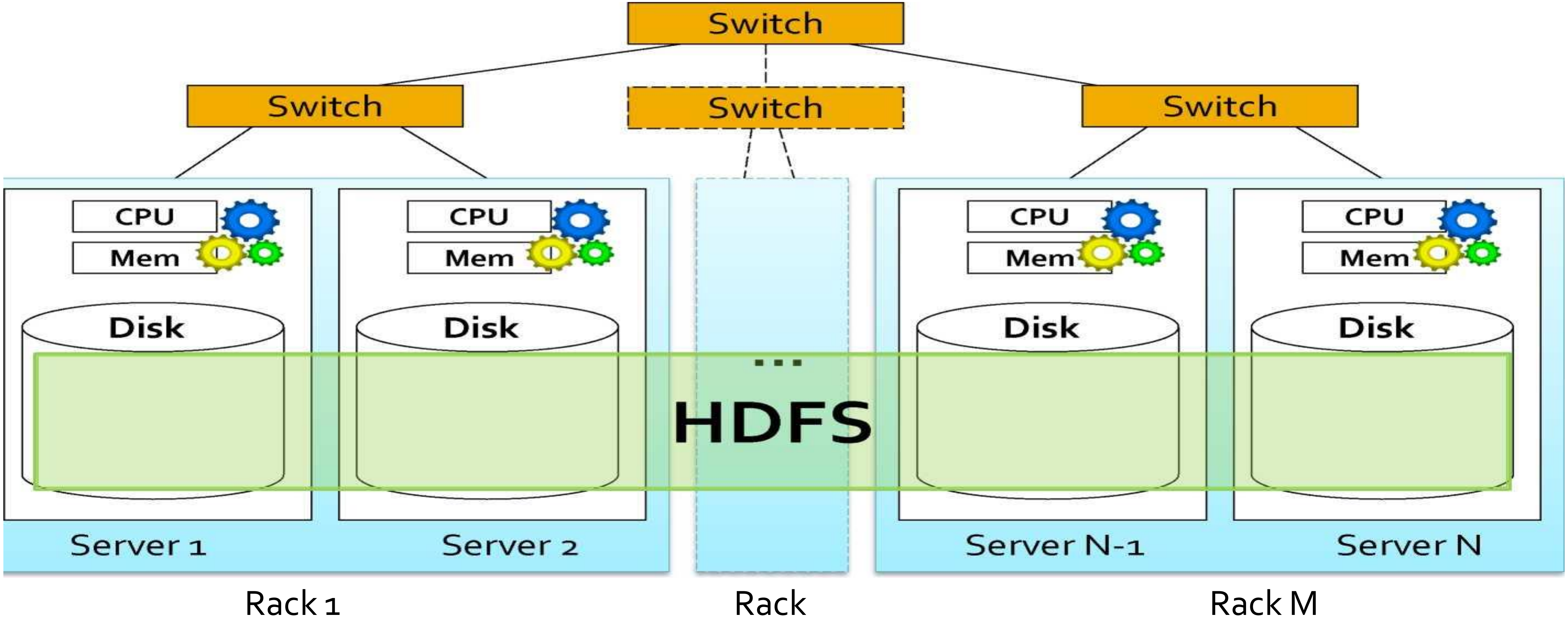
Hadoop

Hadoop: main components



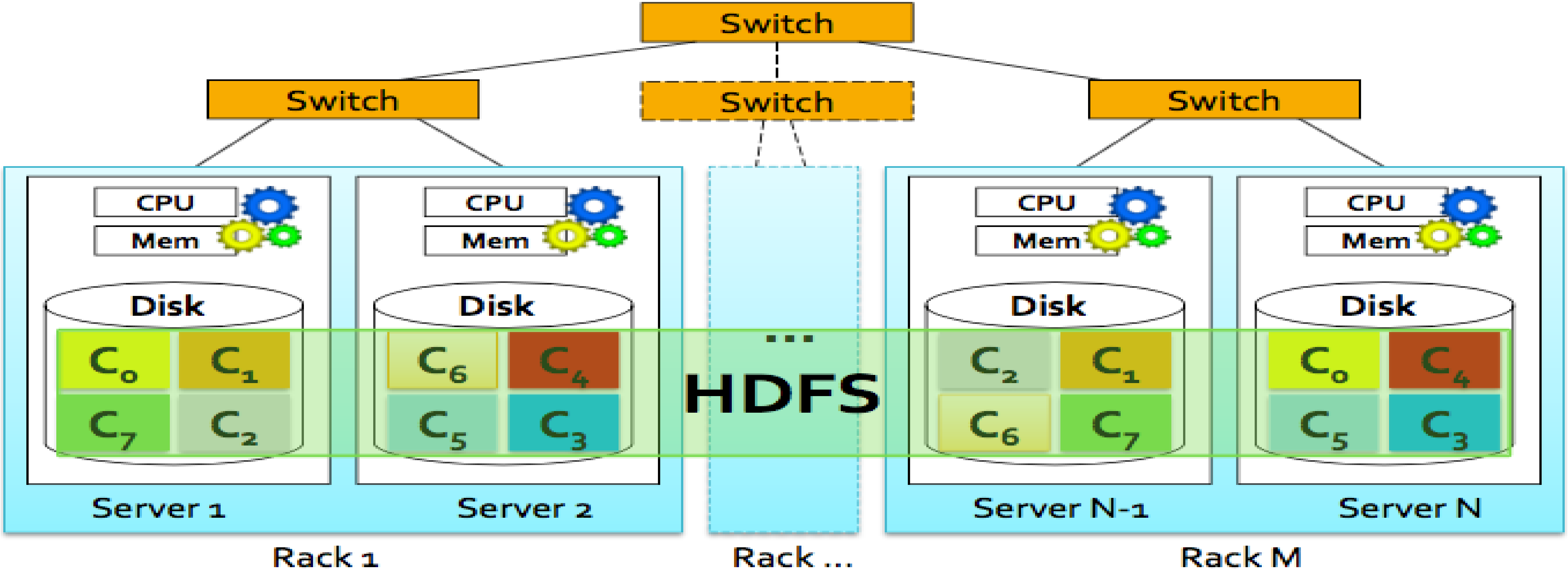
Example with number of replicas per chunk = 2

Hadoop: main components



Example with number of replicas per chunk = 2

Hadoop: main components



Example with number of replicas per chunk = 2

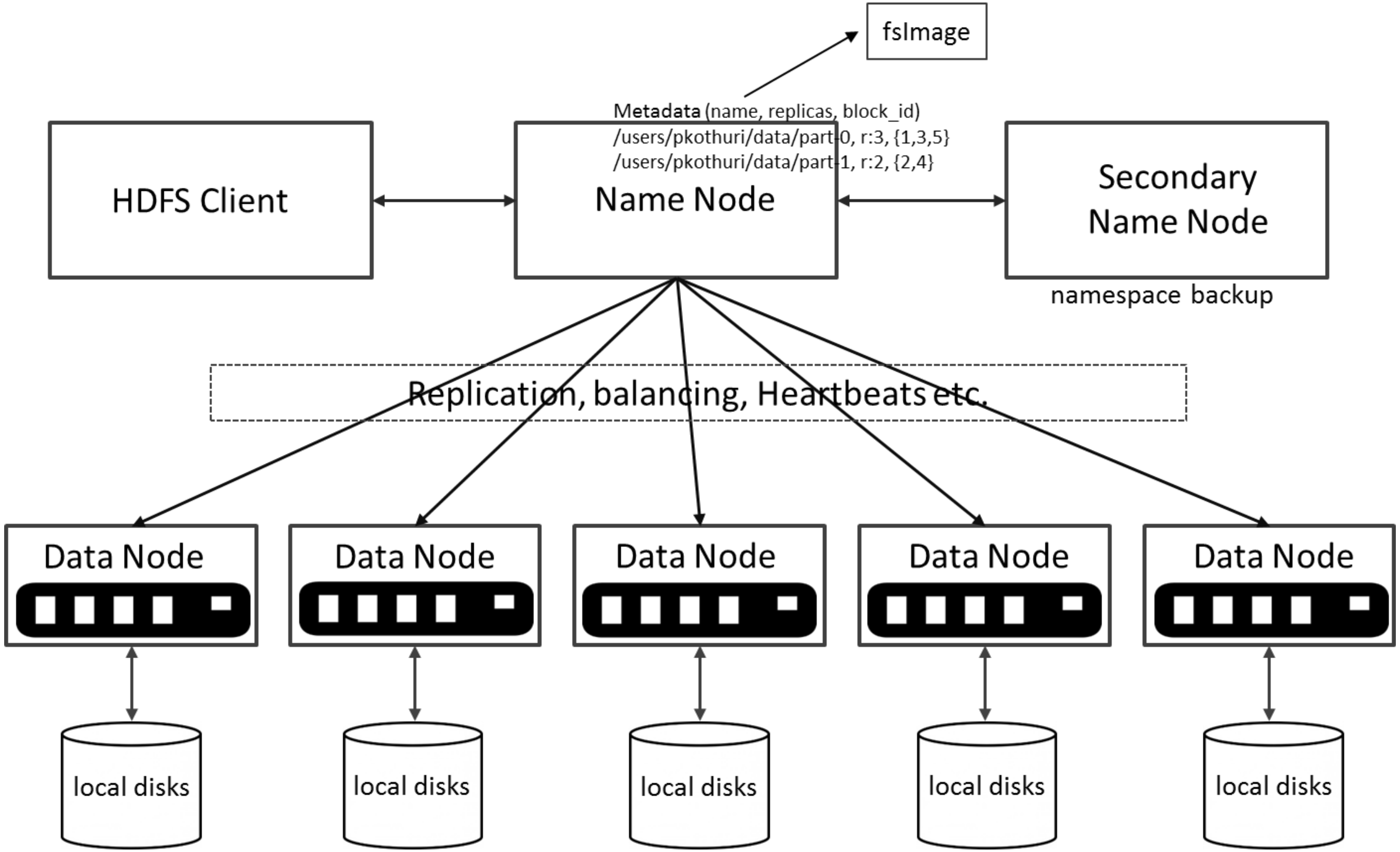
What is HDFS

- HDFS is a distributed file system that is fault tolerant, scalable and extremely easy to expand
- HDFS is the primary distributed storage for Hadoop applications
- HDFS provides interfaces for applications to move themselves closer to data
- HDFS is designed to ‘just work’, however a working knowledge helps in diagnostics and improvements

Components of HDFS

- There are two (*and a half*) types of machines in a HDFS cluster
- NameNode is the heart of an HDFS filesystem, it maintains and manages the file system metadata. E.g; what blocks make up a file, and on which datanodes those blocks are stored
- DataNode where HDFS stores the actual data, there are usually quite a few of these

HDFS Architecture



Unique features of HDFS

HDFS also has a bunch of unique features that make it ideal for distributed systems:

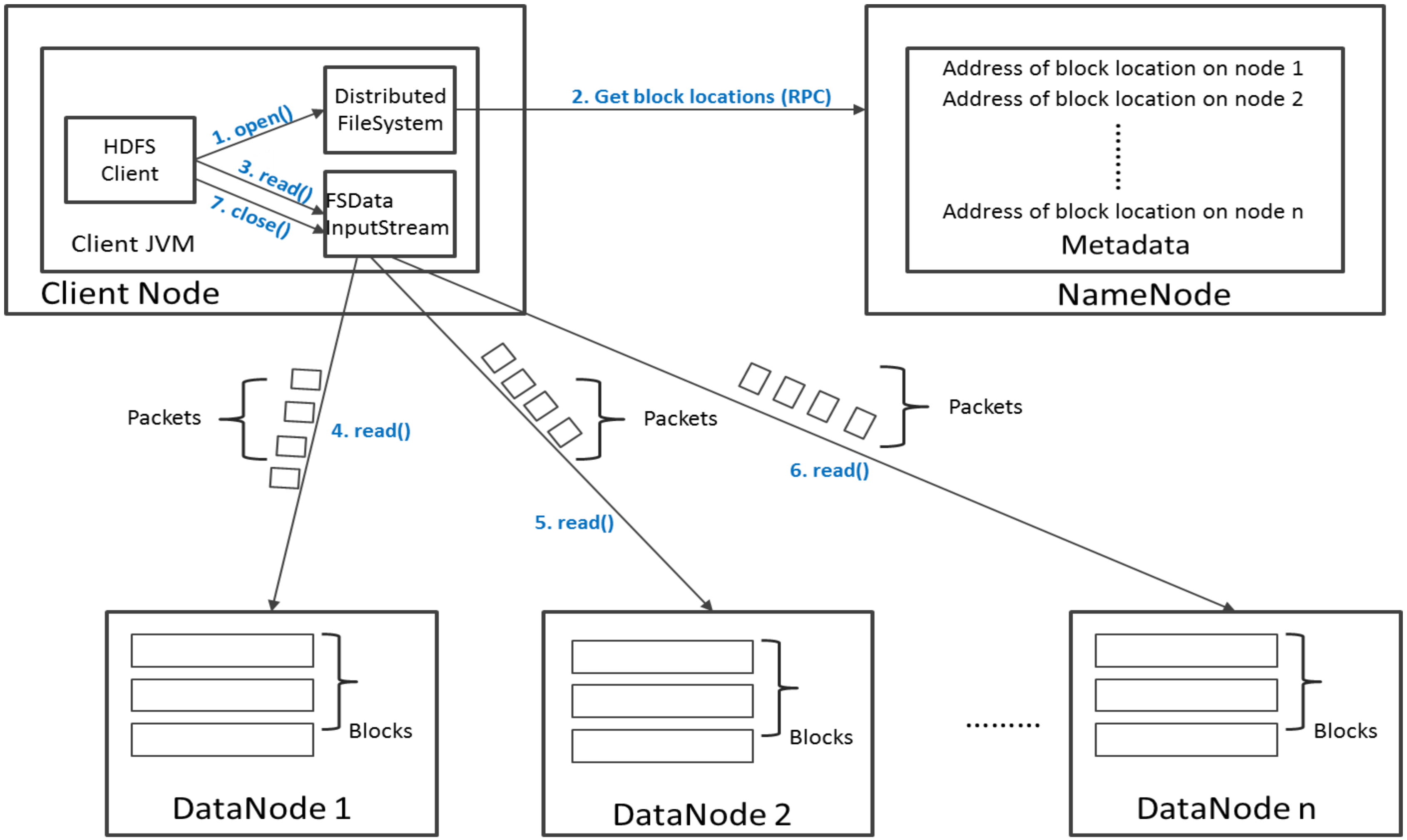
- Failure tolerant - data is duplicated across multiple DataNodes to protect against machine failures. The default is a replication factor of 3 (every block is stored on three machines).
- Scalability - data transfers happen directly with the DataNodes so your read/write capacity scales fairly well with the number of DataNodes
- Space - need more disk space? Just add more DataNodes and re-balance
- Industry standard - Other distributed applications are built on top of HDFS (HBase, Map-Reduce)

HDFS is designed to process large data sets with write-once-read-many, it is not for low latency access

HDFS – Data organization

- Each file written into HDFS is split into data blocks
- Each block is stored on one or more nodes
- Each copy of the block is called replica
- Block placement policy
    - First replica is placed on the local node
    - Second replica is placed in a different rack
    - Third replica is placed in the same rack as the second replica

Read Operation in HDFS



Write Operation in HDFS