Distributed File System

Chandan Mazumdar

Jadavpur University

Agenda

- File Systems – Quick Recapitulation
- Distributed File Systems
- Hadoop DFS

File Concept

- Contiguous logical address space
  - OS abstracts from the physical properties of its storage device to define a logical storage unit called file.
  - Persistent
  - OS maps files to physical devices.

- Types
  - Data
    - numeric, character, binary
  - Program
    - source, object (load image)
  - Documents

File Structure

- None - sequence of words/bytes
- Simple record structure
  - Lines
  - Fixed Length
  - Variable Length
- Complex Structures
  - Formatted document
  - Relocatable Load File
- Can simulate last two with first method by inserting appropriate control characters
- Who decides
  - Operating System
  - Program

File Attributes

- Name
  - symbolic file-name, only information in human-readable form
- Identifier
  - Unique tag that identifies file within filesystem; non-human readable name
- Type -
  - for systems that support multiple types
- Location -
  - pointer to a device and to file location on device
- Size -
  - current file size, maximal possible size
- Protection -
  - controls who can read, write, execute
- Time, Date and user identification
  - data for protection, security and usage monitoring
- Information about files are kept in the directory structure, maintained on disk

File Operations

- A file is an abstract data type. It can be defined by operations:
  - Create a file
  - Write a file
  - Read a file
  - Reposition within file - file seek
  - Delete a file
  - Truncate a file
  - Open(Fi)
    - search the directory structure on disk for entry Fi, and move the content of entry to memory.
  - Close(Fi)
    - move the content of entry Fi in memory to directory structure on disk.

Directory Structure

- Number of files on a system can be extensive
  - Break file systems into partitions (treated as a separate storage device)
  - Hold information about files within partitions.
- Device Directory: A collection of nodes containing information about all files on a partition.
- Both the directory structure and files reside on disk.

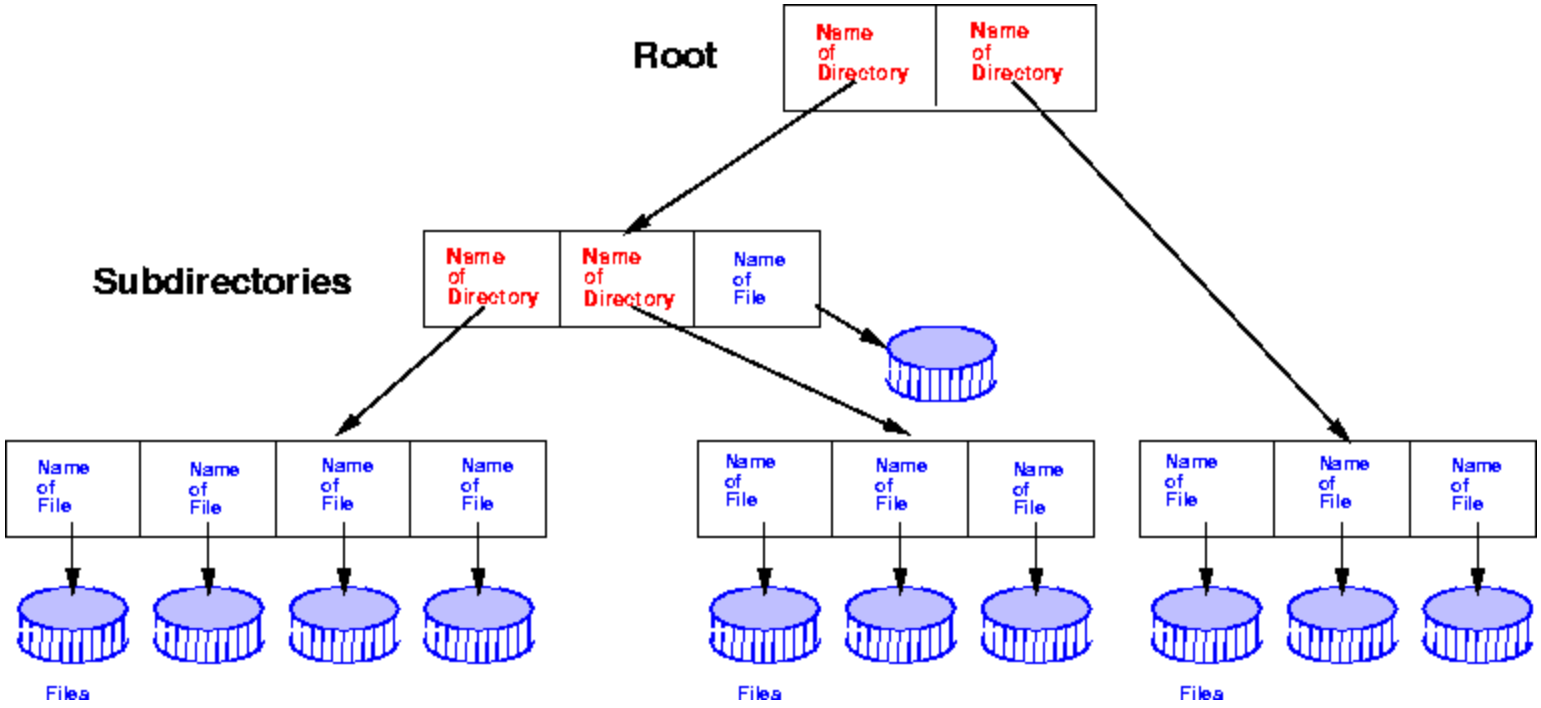
Information in a Device Directory

- File Name
- File Type
- Address or Location
- Current Length
- Maximum Length
- Date created, Date last accessed (for archival), Date last updated (for dump)
- Owner ID (who pays), Protection information
  - Also on a per file, per process basis
    - Current position - read/write position
    - usage count

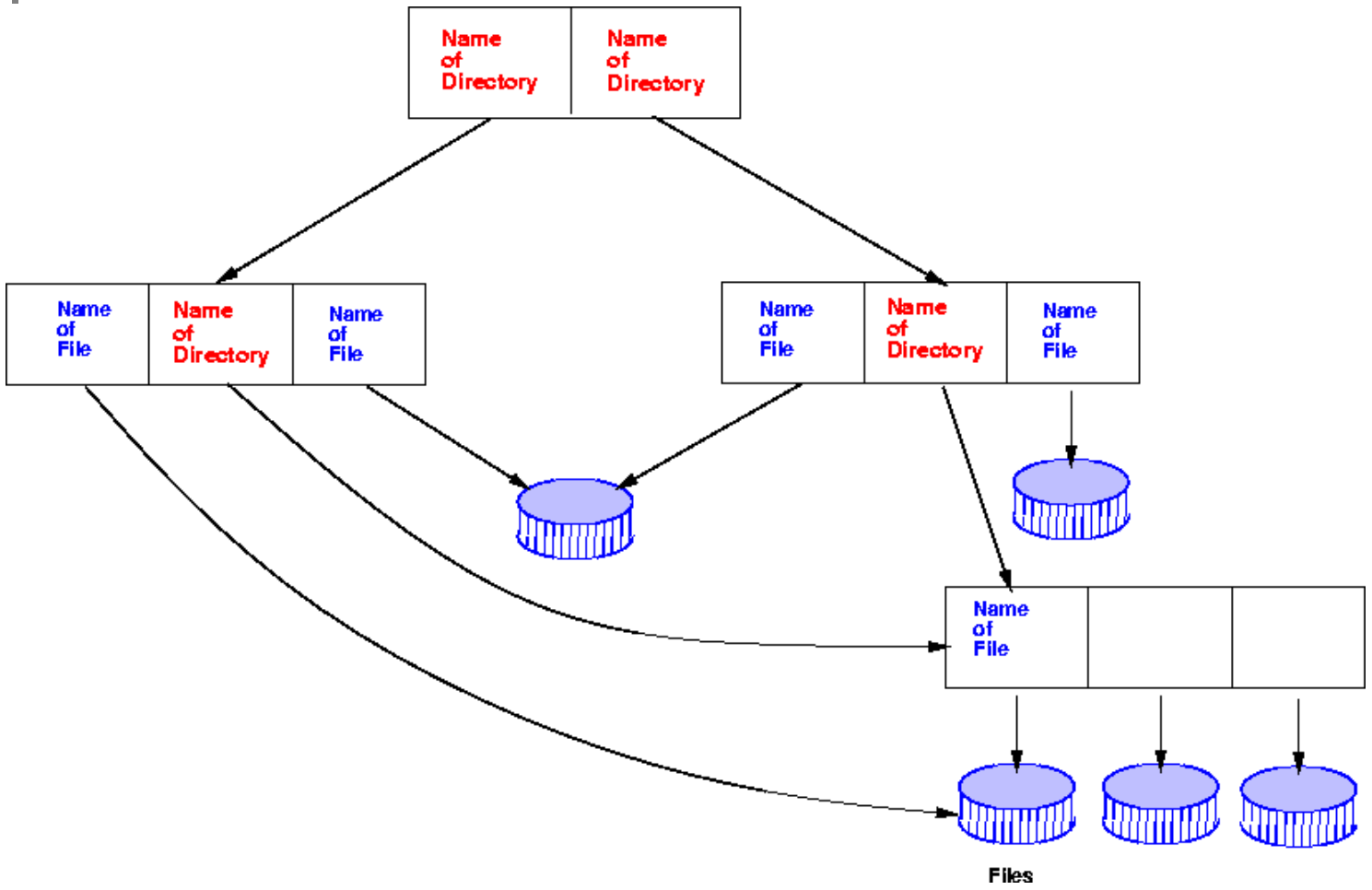
Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the filesystem

Tree structured Directories



Acyclic Graph Directories



Access Methods

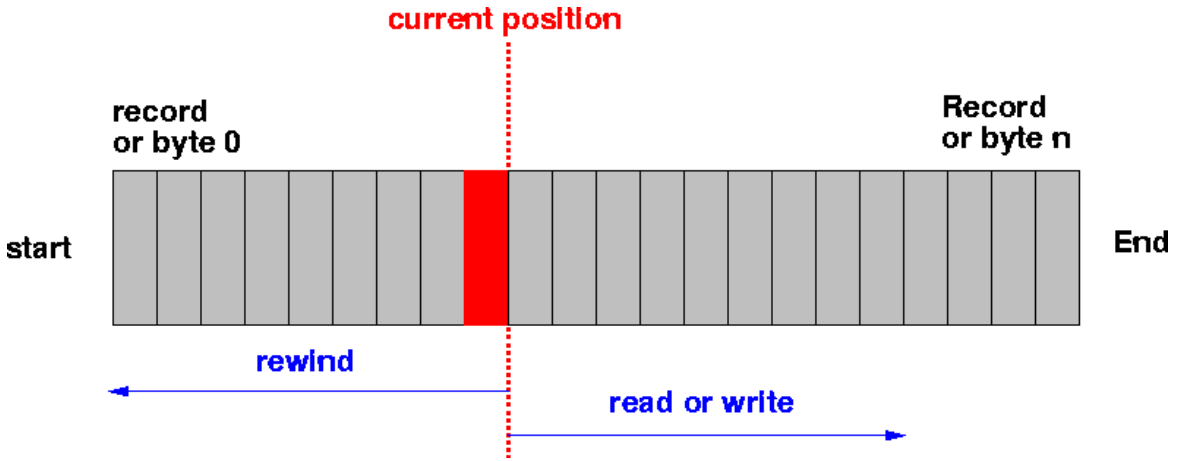
- Sequential Access

  read next
  write next
  reset
  no read after last write (rewrite)

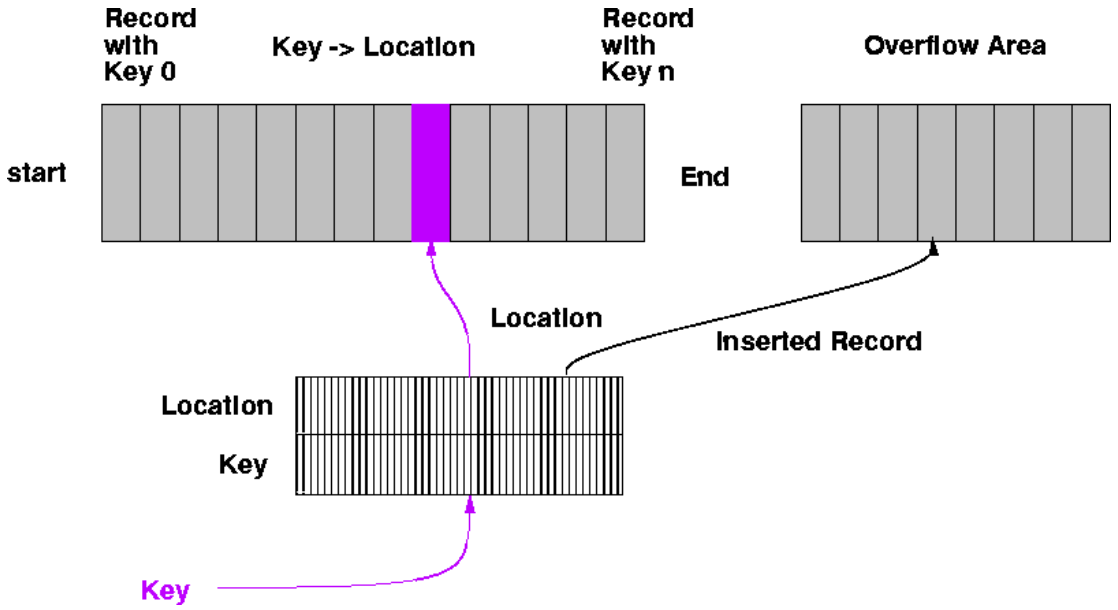
- Direct Access (n = relative block number)

  read n
  write n
  position to n
    read next
    write next
  rewrite n

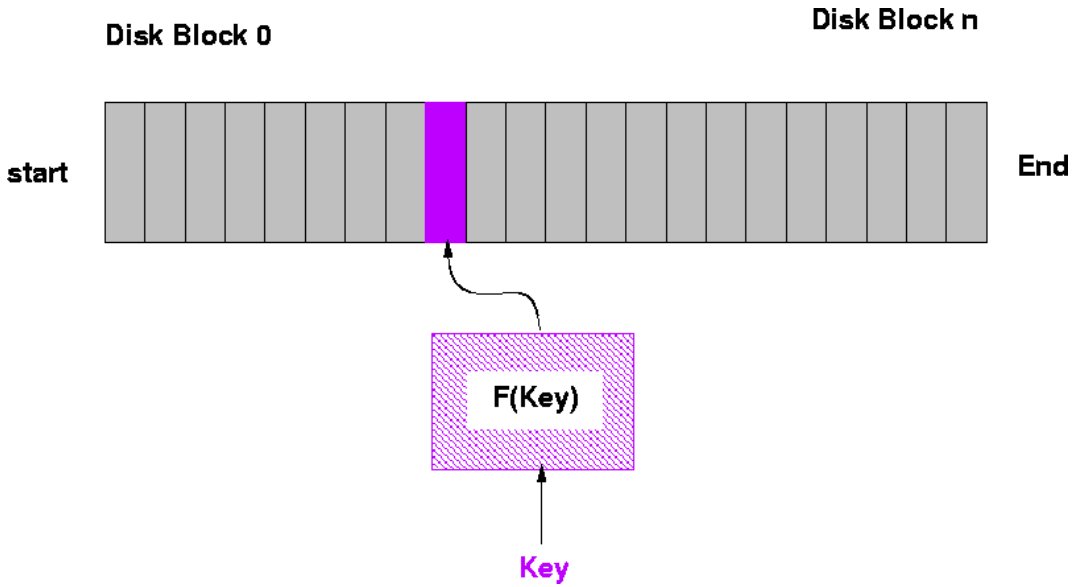
Sequential File Organization



Indexed Sequential or Indexed File Organization



Direct Access File Organization



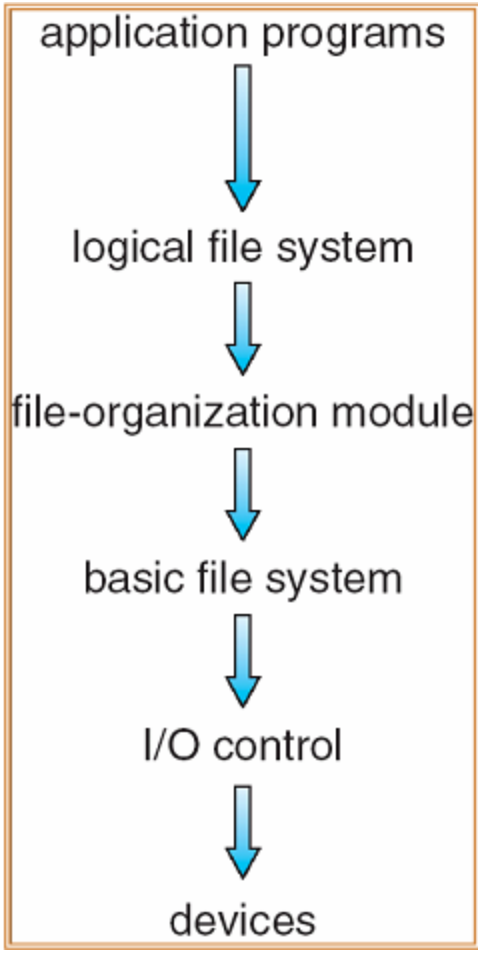
Protection

- File owner/creator should be able to control
  - what can be done
  - by whom

- Types of access
  - read
  - write
  - execute
  - append
  - delete
  - list

File-System Implementation

Layered File System



Handles the CONTENT of the file. Knows the file's internal structure.

Handles the OPEN, etc. system calls. Understands paths, directory structure, etc.

Uses directory information to figure out blocks, etc. Implements the READ. POSITION calls.

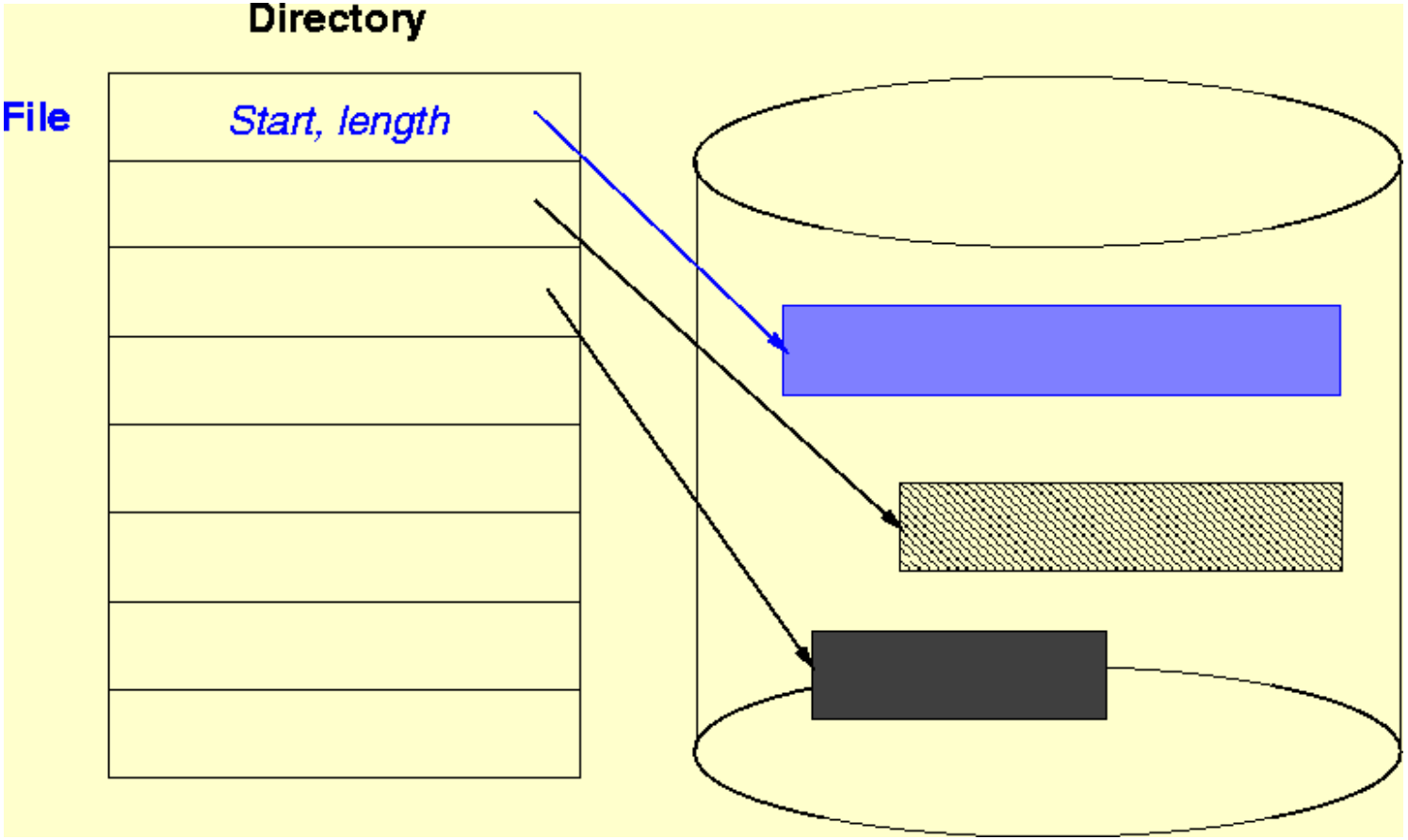
Determines where on the disk blocks are located.

Interfaces with the devices – handles interrupts.

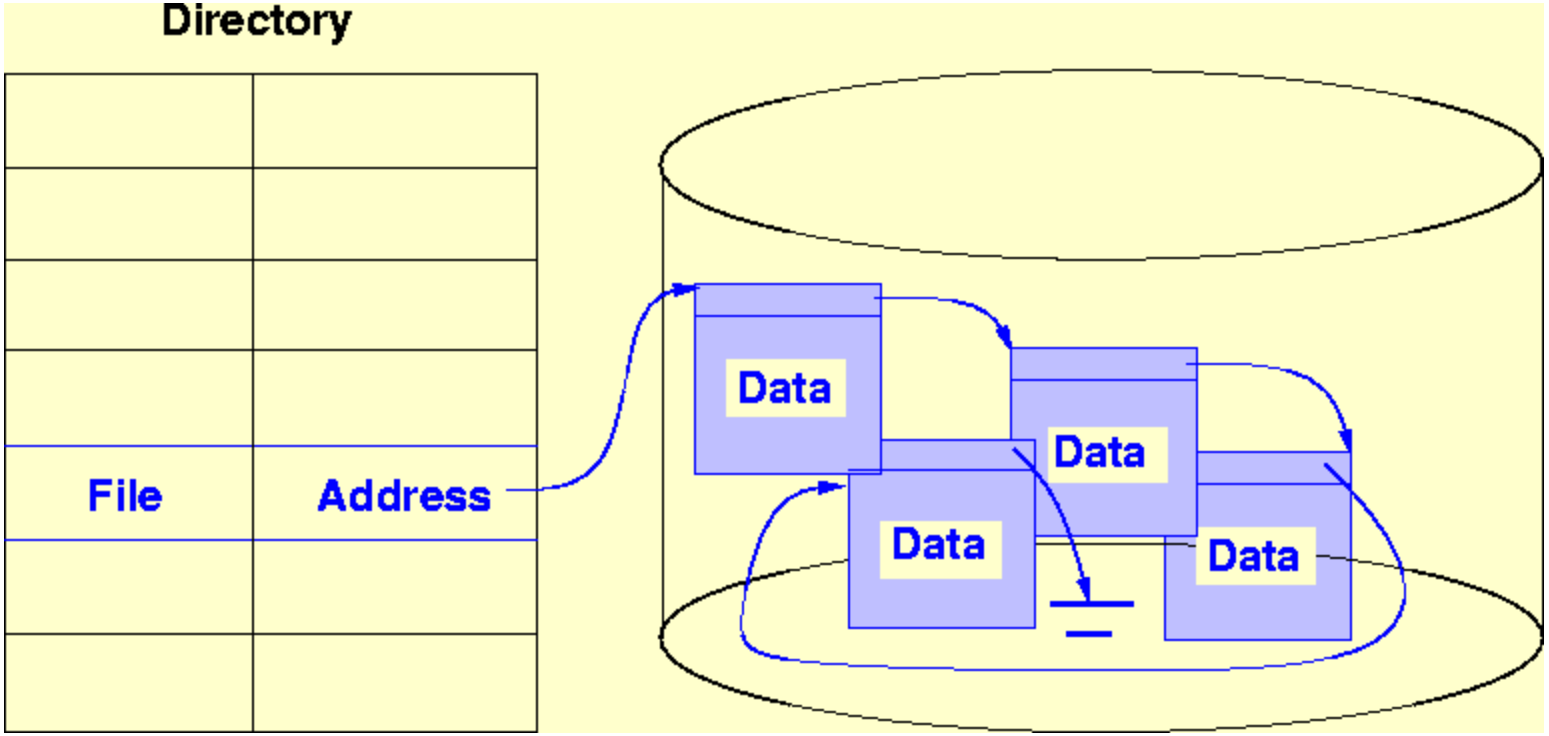
Allocation of Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous Allocation
  - Linked List Allocation
  - Block Allocation

Contiguous Allocation (Contd.)

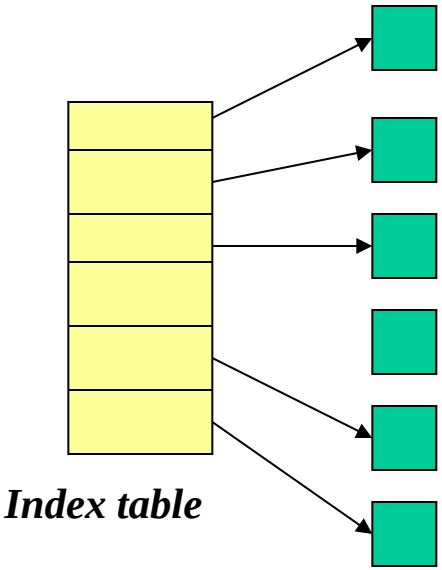


Linked Allocation (Contd.)

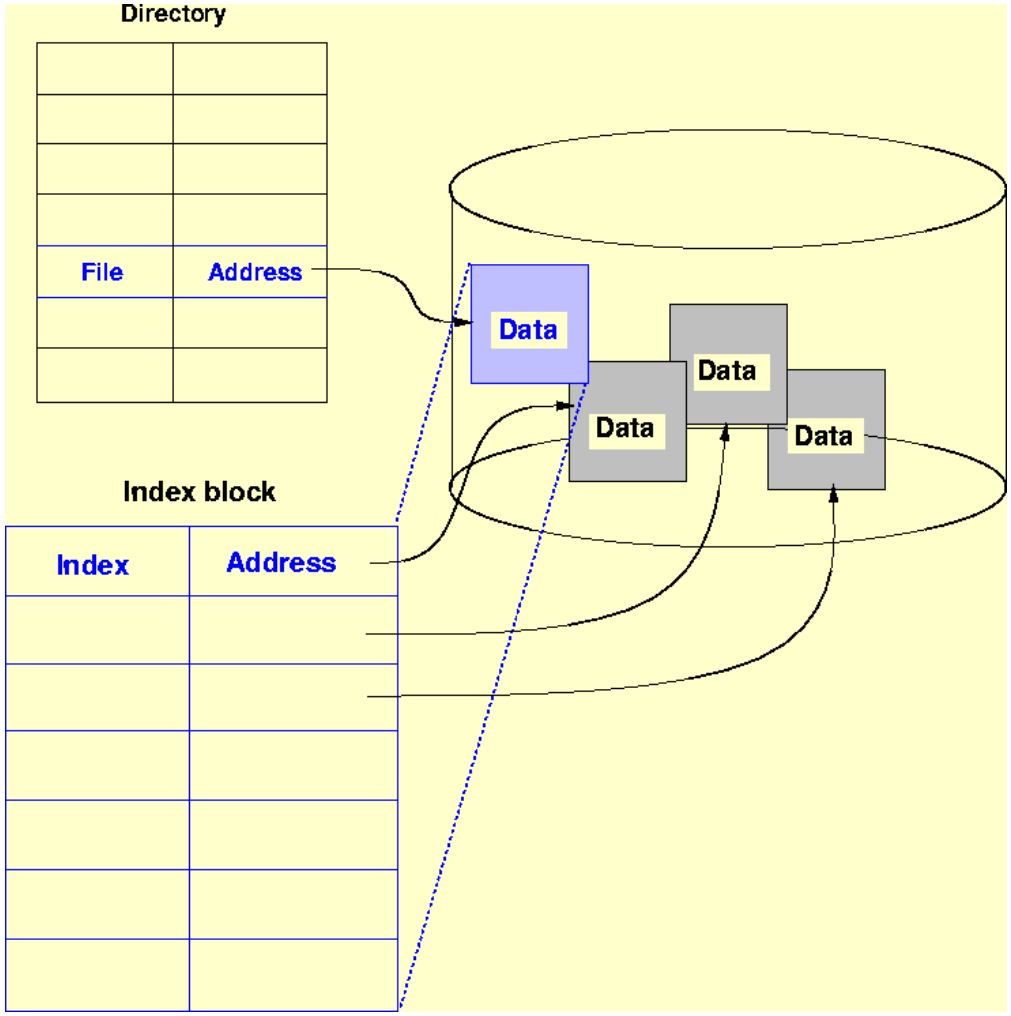


Indexed Allocation

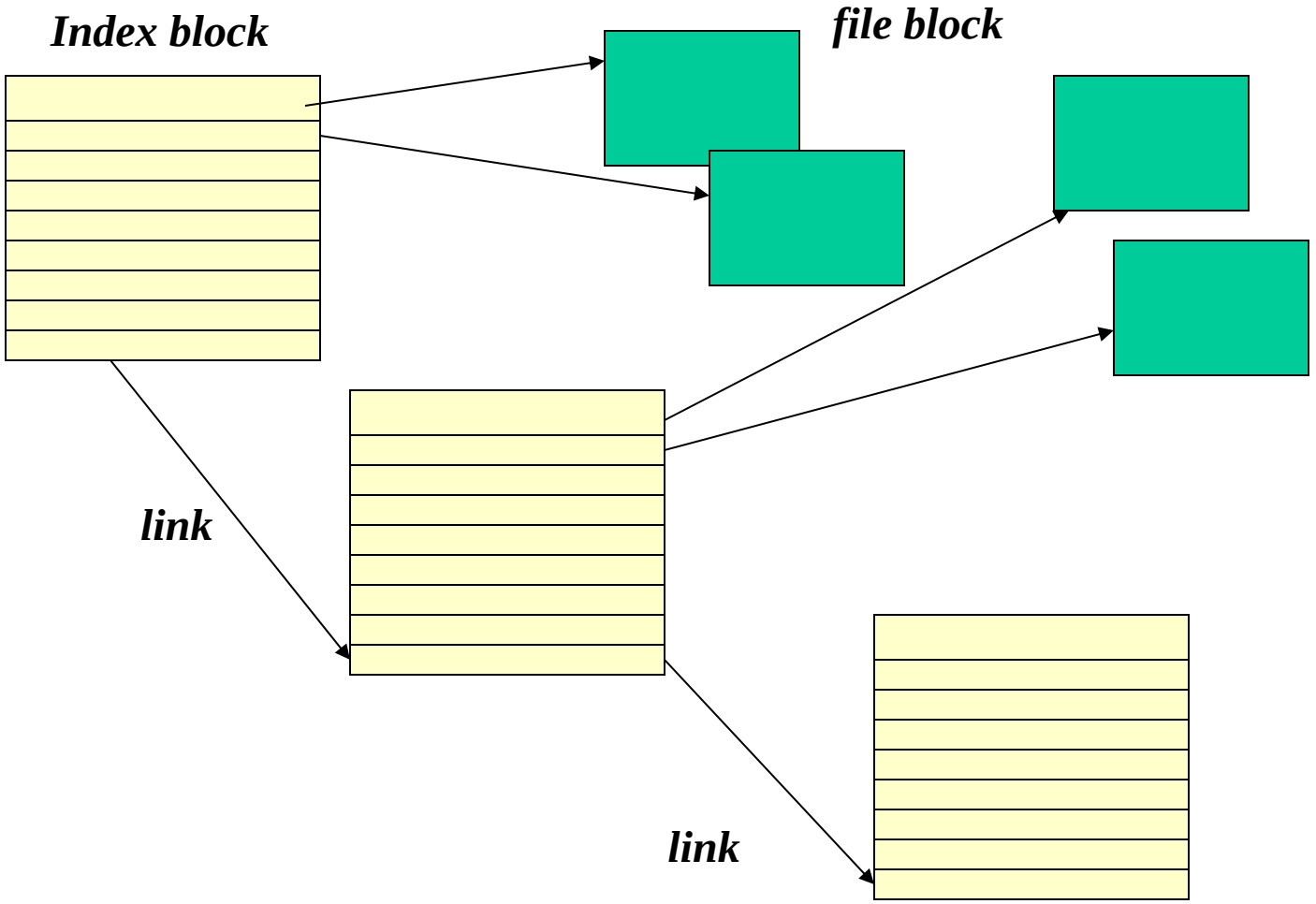
- Brings all pointers together into the index block.
- Logical view



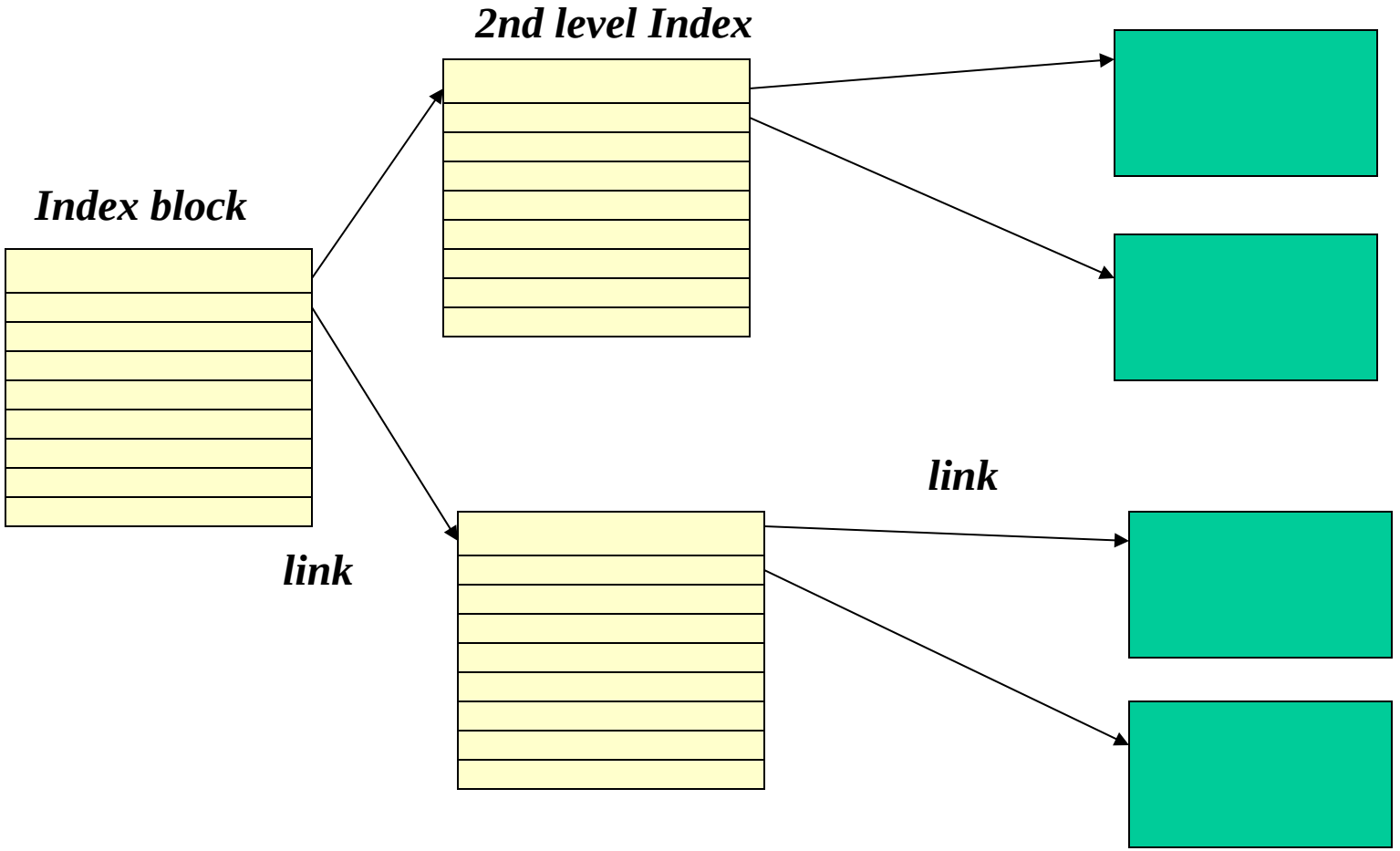
Indexed Allocation (Contd.)



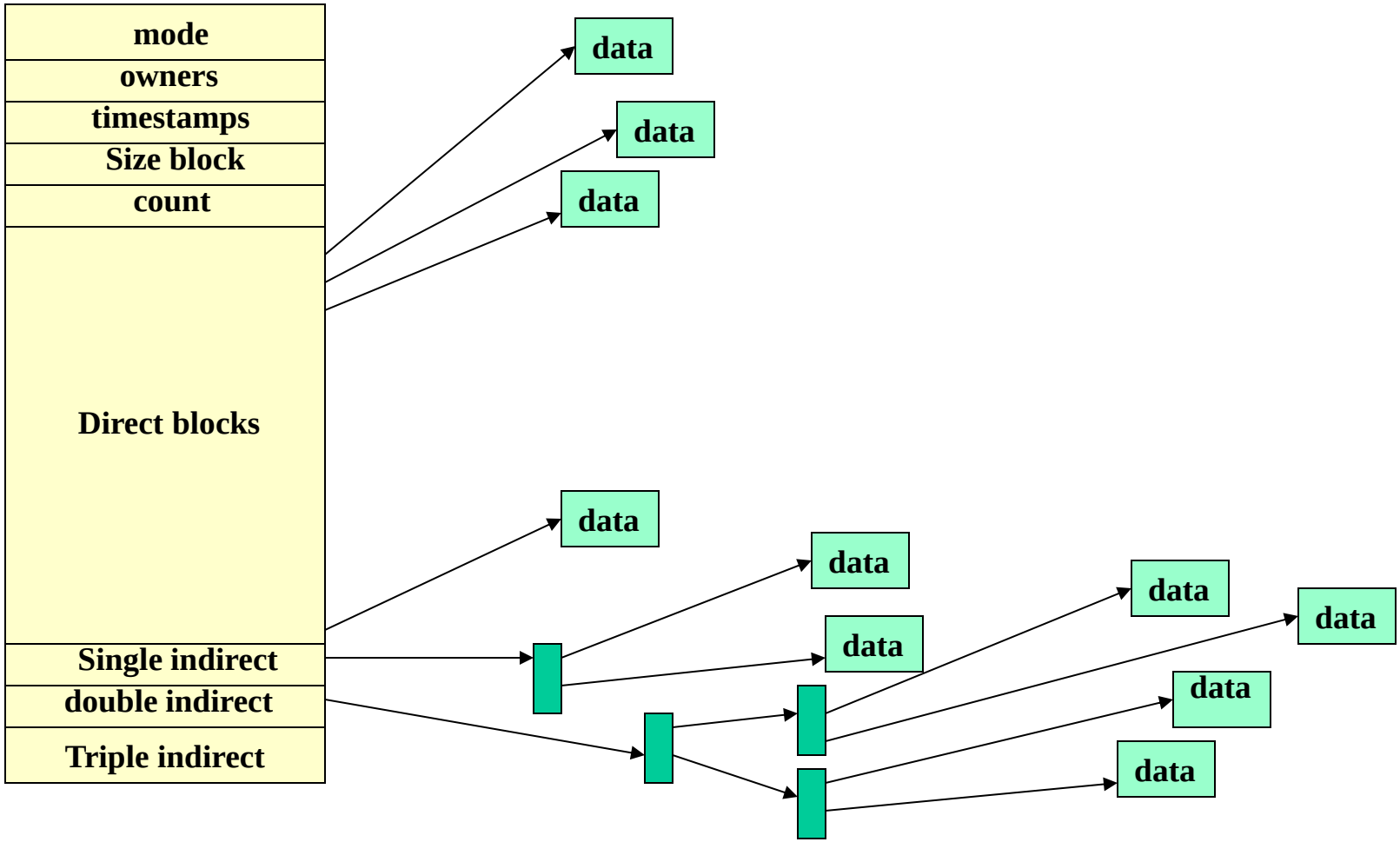
Indexed File - Linked Scheme



Indexed Allocation - Multilevel index



Combined Scheme: UNIX Inode



What is an inode?

- An inode (index node) is a control structure that contains key information needed by the OS to access a particular file. Several file names may be associated with a single inode, but each file is controlled by exactly ONE inode.

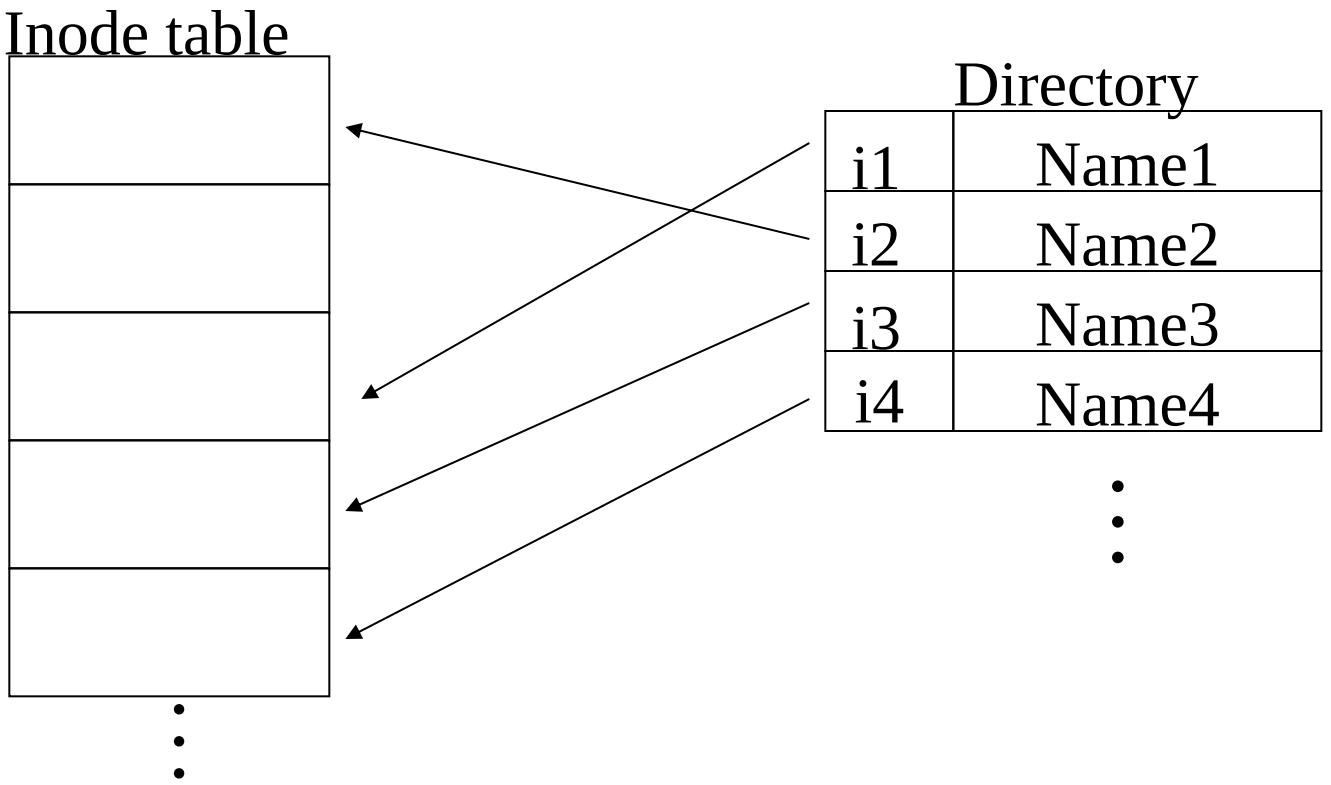
- On the disk, there is an inode table that contains the inodes of all the files in the filesystem. When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.

Information in the inode

File Mode	16-bit flag that stores access and execution permissions associated with the file. 12–14 File type (regular, directory, character or block special, FIFO pipe 9–11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

Directories

- In Unix a directory is simply a file that contains a list of file names plus pointers to associated inodes



Directory Implementation

- Linear list of file names with pointers to the data blocks
  - simple to program
  - time-consuming to execute - linear search to find entry.
  - Sorted list helps - allows binary search and decreases search time.

- Hash Table - linear list with hash data structure
  - decreases directory search time
  - collisions - situations where two file names hash to the same location.
  - Each hash entry can be a linked list - resolve collisions by adding new entry to linked list.

Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in the files directory entry
  - Dynamic allocation of kernel structures

- Performance improved by:
  - *On-board cache* - for disk controllers
  - *Disk Cache* - separate section of main memory for frequently used blocks. Block replacement mechanisms
    - LRU
    - *Free-behind* - removes block from buffer as soon as next block is requested.
    - *Read-ahead* - request block and several subsequent blocks are read and cached.
  - Improve PC performance by dedicating section of memory as *virtual disk* or RAM *disk*.

Recovery

- Ensure that system failure does not result in loss of data or data inconsistency.
- Consistency checker
  - compares data in directory structure with data blocks on disk and tries to fix inconsistencies.
- Backup
  - Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape).
- Restore
  - Recover lost file or disk by restoring data from backup.

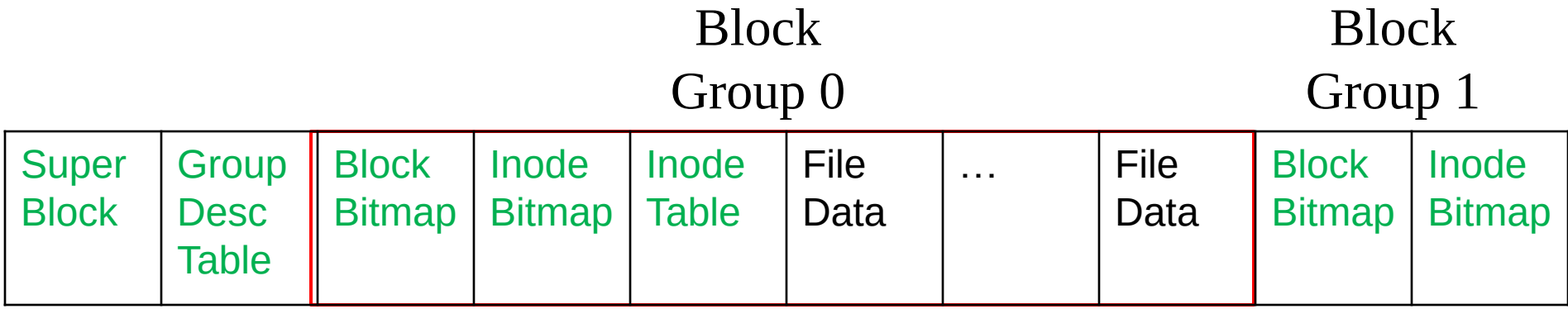
Example File System Ext2 and Ext3

Introduction

- Ext2 and Ext3 are the default Linux file system.
- Ext3 is the new version of Ext2 and adds journaling mechanism, but the basic structures are the same.
- The metadata is stored throughout the file system, and the metadata which is associated with a file are stored “near” it.

File System Layout

- The whole area is divided into several block groups, and block groups contains several blocks.
- A block group is used to store file metadata and file content



Metadata Concepts

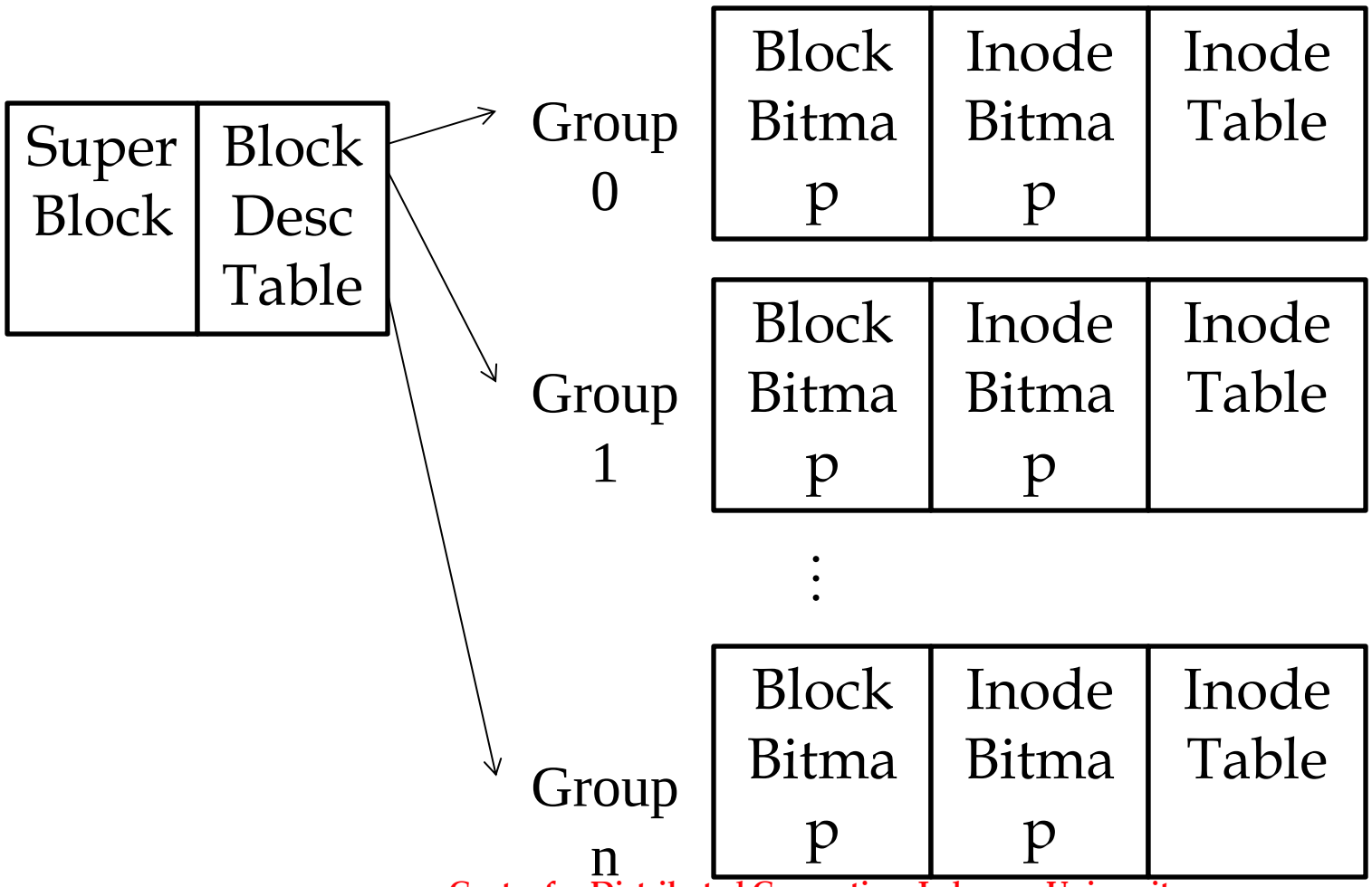
- Superblock:
  - The Ext2 superblock is located 1024 bytes from the start of the file system and is 1024 bytes in size. (The first two sectors are used to store boot code if necessary)
  - Back up copies are typically stored in the first file data block of each block group
  - It contains basic information of the file system, such as the block size, the total number of blocks, etc.

Metadata Concepts (Contd.)

- Block Group Descriptor Table:
  - It contains a group descriptor data structure for every block group.
  - The group descriptor stores the address of block bitmap and inode bitmap for the block group.

- Bitmaps:
  - The block bitmap manages the allocation status of the blocks in the group.
  - The inode bitmap manages the allocation status of the inodes in the group

Metadata Concepts (Contd.)

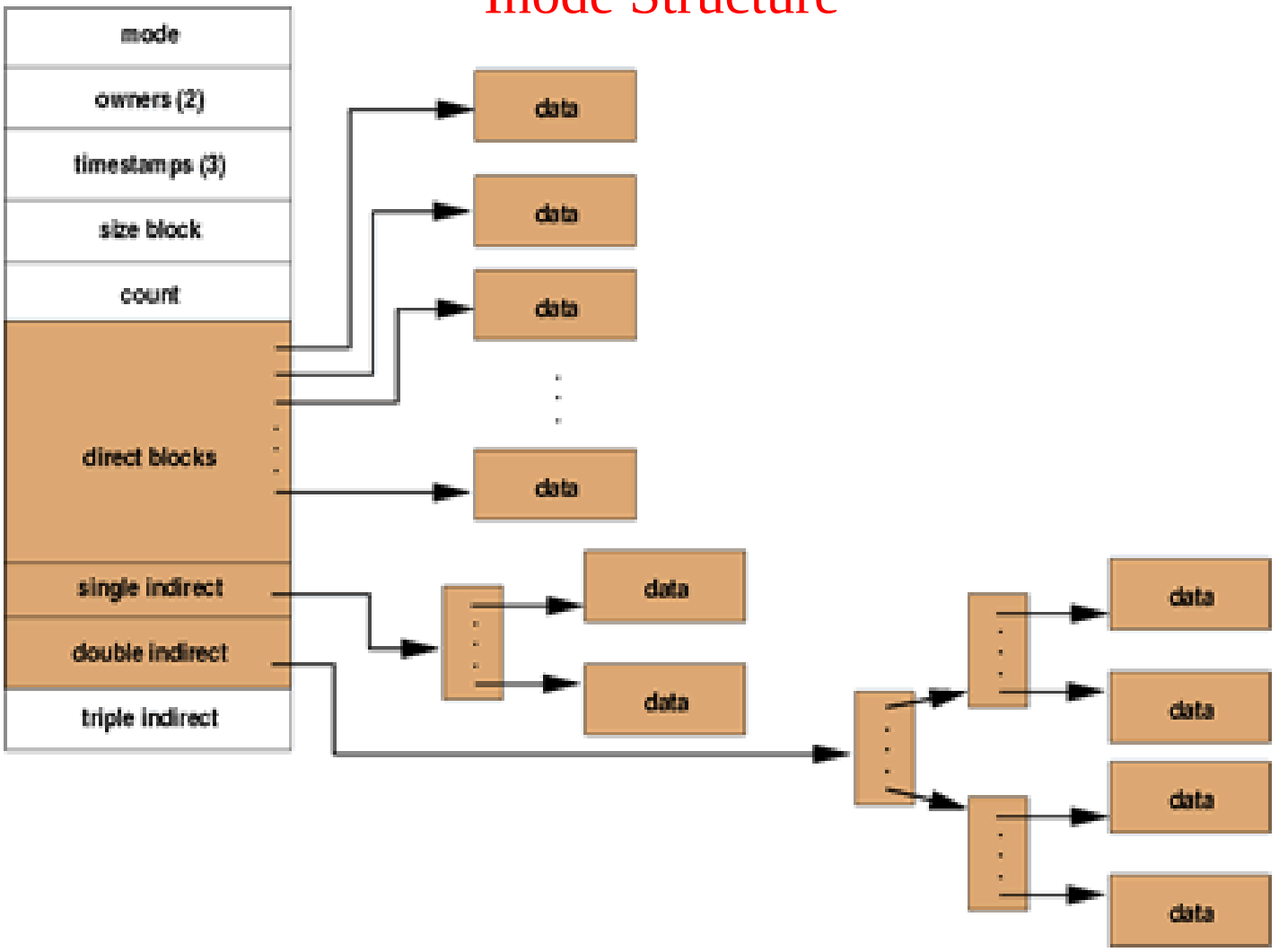


Metadata Concepts (Contd.)

- Inode Tables:
  - Inode table contains the inodes that describes the files in the group
- Inodes:
  - Each inode corresponds to one file, and it stores file's primary metadata, such as file's size, ownership, and temporal information.
  - Inode is typically 128 bytes in size and is allocated to each file and directory

Metadata Concepts (Contd.)

Inode Structure



Metadata Concepts (Contd.)

- Inode Allocation:
  - If a new inode is for a non-directory file, Ext2 allocates an inode in the same block group as the parent directory.
  - If that group has no free inode or block, Ext2 uses a quadratic search (add powers of 2 to the current group)
  - If quadratic search fails, Ext2 uses linear search.

Metadata Concepts (Contd.)

- Inode Allocation:
  - If a new inode is for a directory, Ext2 tries to place it in a group that has not been used much.
  - Using total number of free inodes and blocks in the superblock, Ext2 calculates the average free inodes and blocks per group.
  - Ext2 searches each of the group and uses the first one whose free inodes and blocks are less than the average.
  - If the pervious search fails, the group with the smallest number of directories is used.

Metadata Concepts (Contd.)

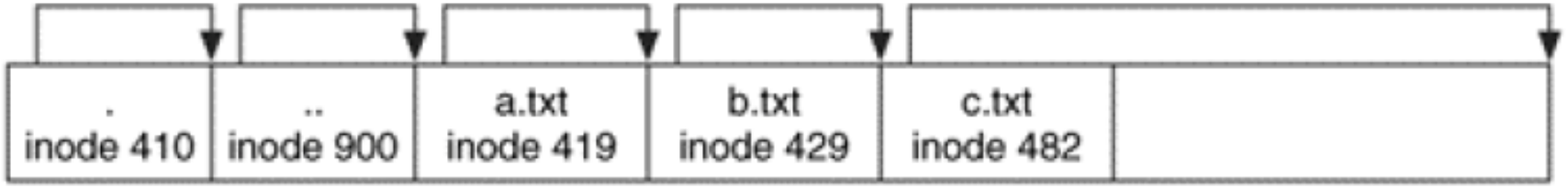
- Ext2 uses blocks as its data unit and is similar to clusters in FAT and NTFS.
- The default size of blocks is 4KB, and the size is given in the superblock.
- When a block is allocated to an inode, Ext2 will try to allocate in the same group as the inode and use first-available strategy.

Indexing and Directories

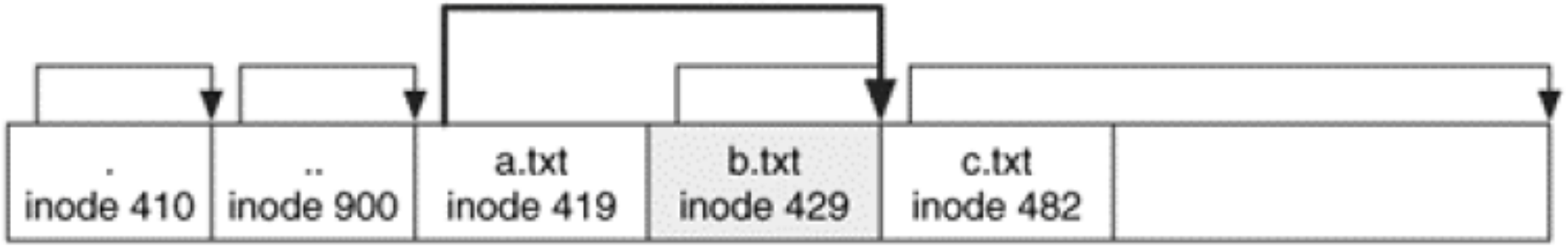
- An Ext2 is just like a regular file except it has a special type value.
- The content of directories is a list of directory entry data structure, which describes file name and inode address.
- The length of directory entry varies from 1 to 255 bytes.
- There are two fields in the directory entry:
  - Name length: the length of the file name
  - Record length: the length of this directory entry

Indexing and Directories

A)



B)



When Ext2 wants to delete a directory entry, it just increase the record length of the previous entry to the end to deleted entry.

Indexing and Directories

- Directory entry allocation:
  - Ext2 starts at the beginning of the directory and examines each directory entry.
  - Calculate the actual record length and compare with the record length field.
  - If they are different, Ext2 can insert the new directory entry at the end of the current entry.
  - Else, Ext2 appends the new entry to the end of the entry list.

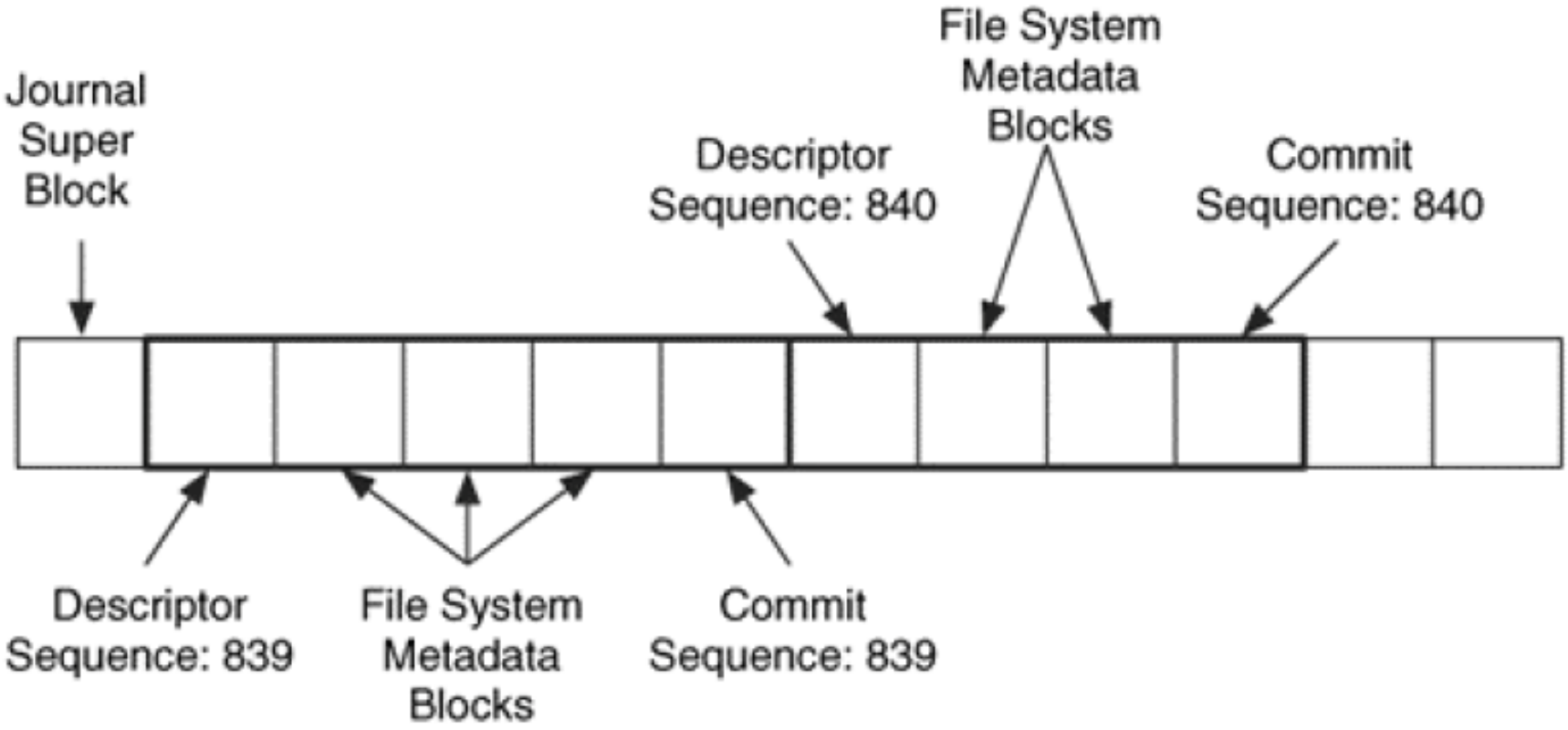
Journaling

- A file system journaling records updates to the file system so that it can be recovered after a crash.
- There are two modes of journaling:
  - Only metadata updates are recorded
  - All updates are recorded
- Journaling in Ext3 is done at block level
- The first block in the journal is journal superblock, and it contains the first logging data address and its sequence number.

Journaling (Contd.)

- Updates are done in transactions, and each transaction has a sequence number.
- Each transaction starts with a descriptor block that contains the transaction sequence number and a list of what blocks are being updated.
- Following the descriptor block are the updated blocks.
- When the updates have been written to disk, a commit block is written with the same sequence number.

Journaling (Contd.)



Distributed File System

Introduction

- Distributed file system (DFS) – a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources.

- A DFS manages set of dispersed storage devices

- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces.

Introduction (Contd.)

- Distributed File Systems' main objective is to make remote files to look and feel just like local ones.

- Transparency can be given in different ways like,
  - Access transparency
    - Clients are unaware that files are distributed and can access them in the same way as local files are accessed.
  - Location transparency
    - A consistent name space exists encompassing local as well as remote files. The name of a file does not give it's location.

Introduction (Contd.)

- Concurrency transparency
  - All clients have the same view of the state of the file system.
  - If one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
- Failure transparency
  - The client and client programs should operate correctly after a server failure.
- Heterogeneity
  - File service should be provided across different hardware and operating system platforms.

Introduction (Contd.)

- Scalability
  - The file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).
- Replication transparency
  - To support scalability, we may wish to replicate files across multiple servers.
  - Clients should be unaware of this replication
- Migration transparency
  - Files should be able to move around without the client's knowledge.

Introduction (Contd.)

- Support fine-grained distribution of data
  - To optimize performance it is better to locate individual objects near the processes that use them.
- Tolerance for network partitioning
  - The entire network or certain segments of it may be unavailable to a client during certain periods (e.g. disconnected operation of a laptop).
  - The file system should be tolerant of this.

Distributed File Service

- A file service is a specification of what the file system offers to clients.
- A file server is the implementation of a file service and runs on one or more machines.
- File Service Types:
  - Upload/Download model
  - Remote Access model

Distributed File Service Types

- Upload/Download model
  - fundamental operations
    - Read file
      - transfers an entire file from the server to the requesting client
    - write file
      - copies the file back to the server
  - Pros
    - Simple model
    - Efficient in that it provides local access to the file when it is being used
  - Cons
    - It can be wasteful if the client needs access to only a small amount of the file data.
    - It can be problematic if the client doesn't have enough space to cache the entire file.
    - Problem if others need to modify the same file.

Distributed File Service Types (Contd.)

- Remote Access model
  - The file service provides remote operations such as open, close, read bytes, write bytes, get attributes, etc.
  - The file system itself runs on servers.
  - Cons
    - The servers are accessed for the duration of file access.

Naming issues

- In considering our goals in name resolution, we must distinguish between location transparency and location independence.

- location transparency
  - we mean that the path name of a file gives no hint to where the file is located

- location independence
  - the files can be moved without their names changing.
  - If machine or server names are embedded into path names we do not achieve location independence.

Naming issues (Contd.)

- Access transparency is required
  - The remote file system name space should be syntactically consistent with the local name space
  - Solutions
    - Files named by combination of their host name and local name; guarantees a unique system wide name
      - This can cause legacy applications to fail so not a good option
    - Mount remote file systems onto the local directory hierarchy (merging the two namespaces)
    - Provide a single name space which looks the same on all machines.

Should Server Maintain State?

- A stateless system is one in which the client sends a request to a server, the server carries it out, and returns the result. Between these requests, no client-specific information is stored on the server.

- A stateful system is one where information about client connections is maintained on the server.

Stateless system

- Each request must be complete – the file has to be fully identified and any offsets specified.
-
- Fault tolerance: if a server crashes and then recovers, no state was lost about client connections because there was no state to maintain.
- No remote open/close calls are needed (they only serve to establish state).
- No wasted server space per client.
- No limit on the number of open files on the server; they aren't "open" – the server maintains no per-client state.
- No problems if the client crashes. The server does not have any state to clean up.

Stateful system

- Requests are shorter (less info to send).
- Better performance in processing the requests.
- Cache coherence is possible.
- File locking is possible; the server can keep state that a certain client is locking a file (or portion thereof)

Caching

- Caching improves system performance. There are four places in a distributed system where we can hold data:
  - On the server's disk
  - In a cache in the server's memory
  - In the client's memory
  - On the client's disk
- The first two places are not an issue
  - any interface to the server can check the centralized cache.
- It is in the last two places that problems arise
  - Several approaches may be taken for cache consistency

Caching Mechanisms

- Write-through
  - Write data through to disk as soon as they are placed on any cache. Reliable, but poor performance.

- Delayed writes
  - Modifications written to the cache and then written through to the server later. Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.
    - Poor reliability; unwritten data will be lost whenever a user machine crashes.
    - Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan.
    - Variation – write-on-close, writes data back to the server when the file is closed. Best for files that are open for long periods and frequently modified.

Caching Mechanisms (Contd.)

- centralized control
  - Server keeps track of who has what open in which mode. A stateful system may be used and deal with signaling traffic.

File Replication

- Replicas of the same file reside on failure-independent machines.
- Improves availability and can shorten service time.
- Naming scheme maps a replicated file name to a particular replica.
  - Existence of replicas should be invisible to higher levels.
  - Replicas must be distinguished from one another by different lower-level names.
- Updates
  - Replicas of a file denote the same logical entity, and thus an update to any replica must be reflected on all other replicas.
- Demand replication
  - Reading a nonlocal replica causes it to be cached locally, thereby generating a new non primary replica.

Hadoop DFS

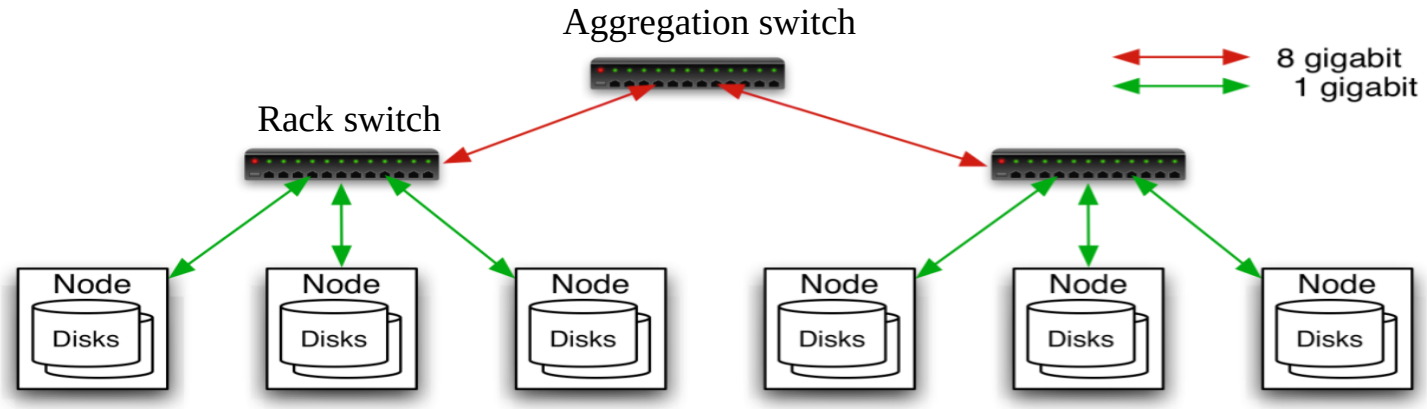
Hadoop - Why ?

- Need to process huge datasets on large clusters of computers
- Very expensive to build reliability into each application
- Nodes fail every day
  - Failure is expected, rather than exceptional
  - The number of nodes in a cluster is not constant
- Need a common infrastructure
  - Efficient, reliable, easy to use
  - Open Source, Apache Licence

Who uses Hadoop?

- Amazon/A9
- Facebook
- Google
- New York Times
- Veoh
- Yahoo!
- …. many more

Commodity Hardware



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 3-4 gigabit
  - Rack-internal is 1 gigabit

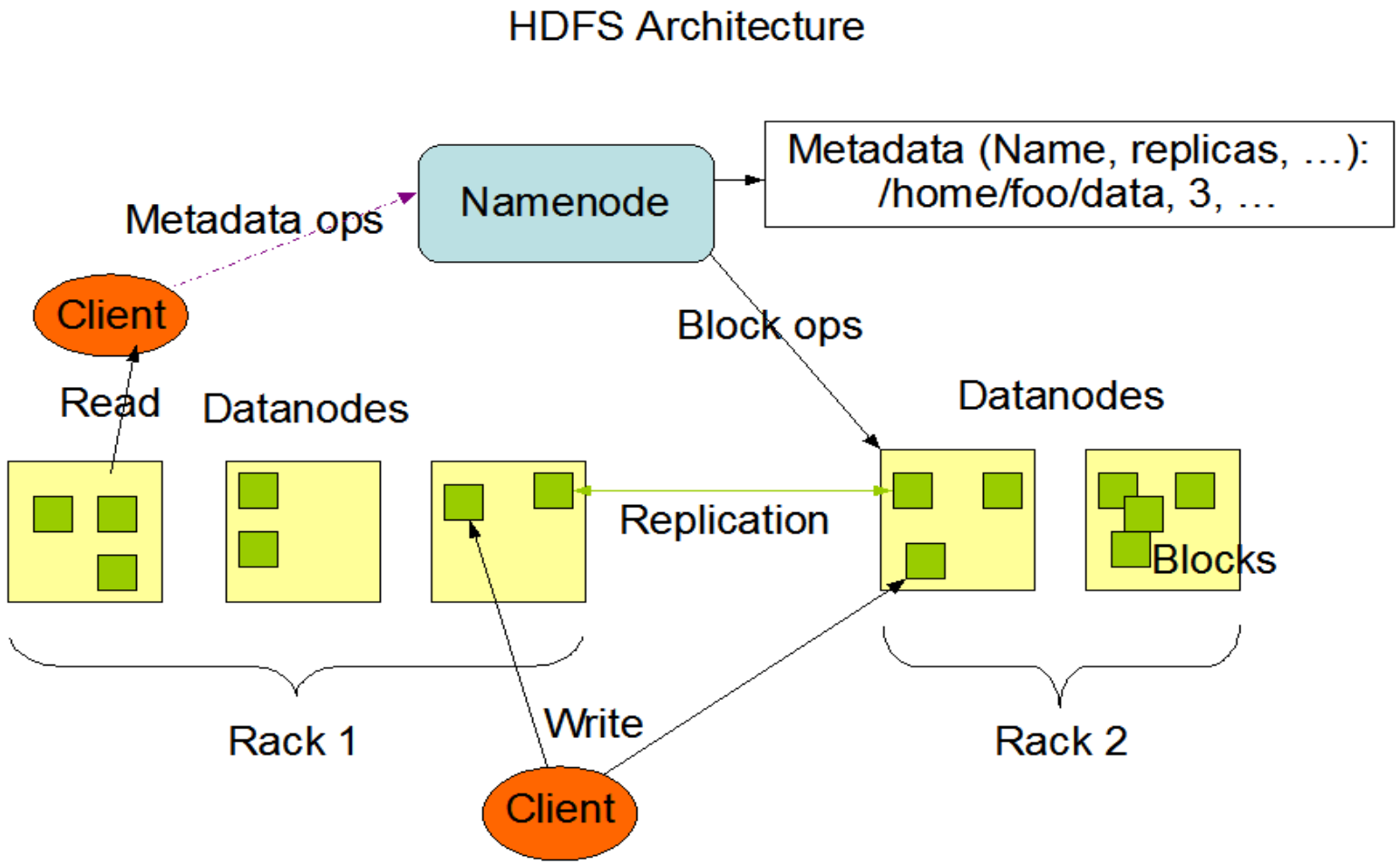
Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

Distributed File System

- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

HDFS Architecture



Functions of a NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

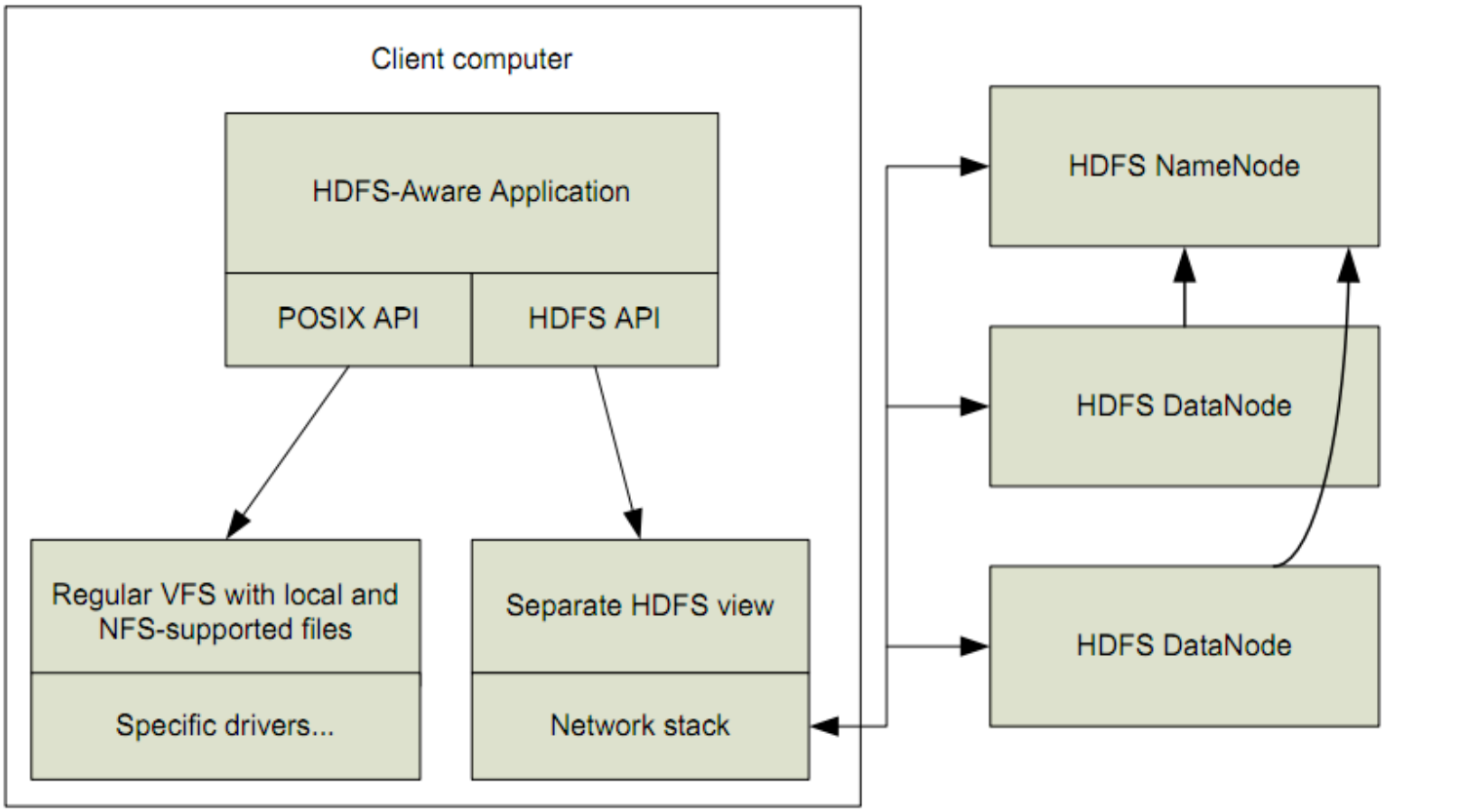
Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable

Heartbeats

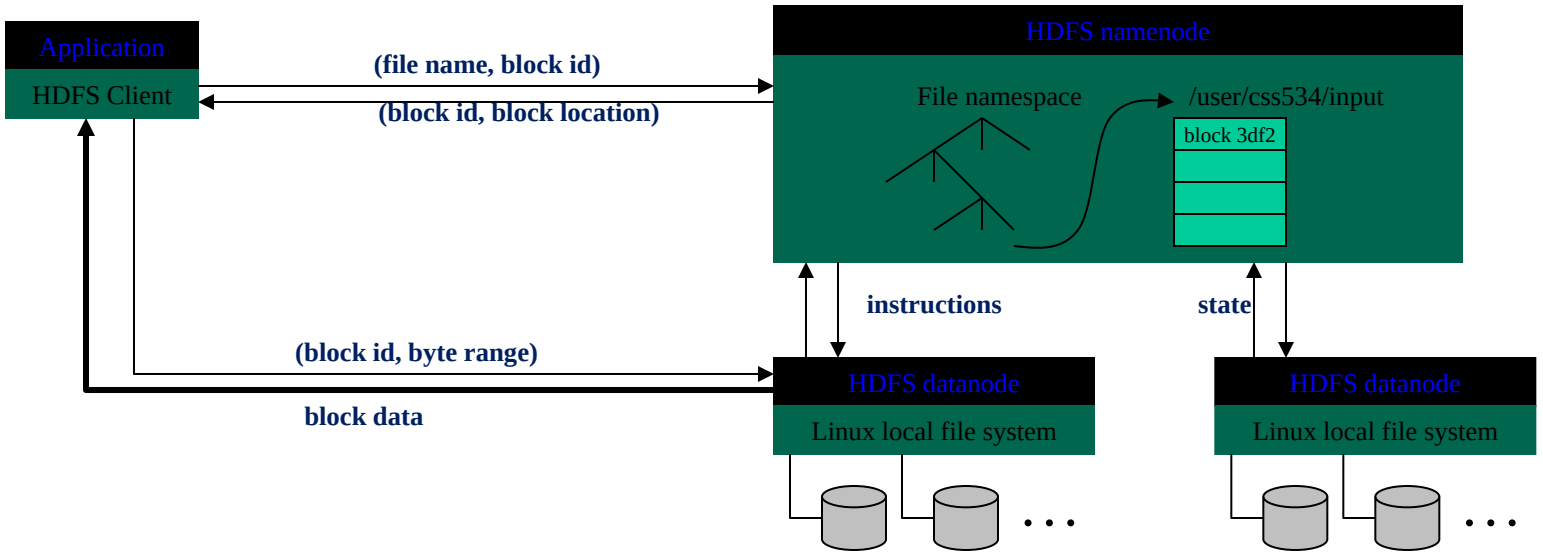
- DataNodes send heartbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

HDFS Client Block Diagram



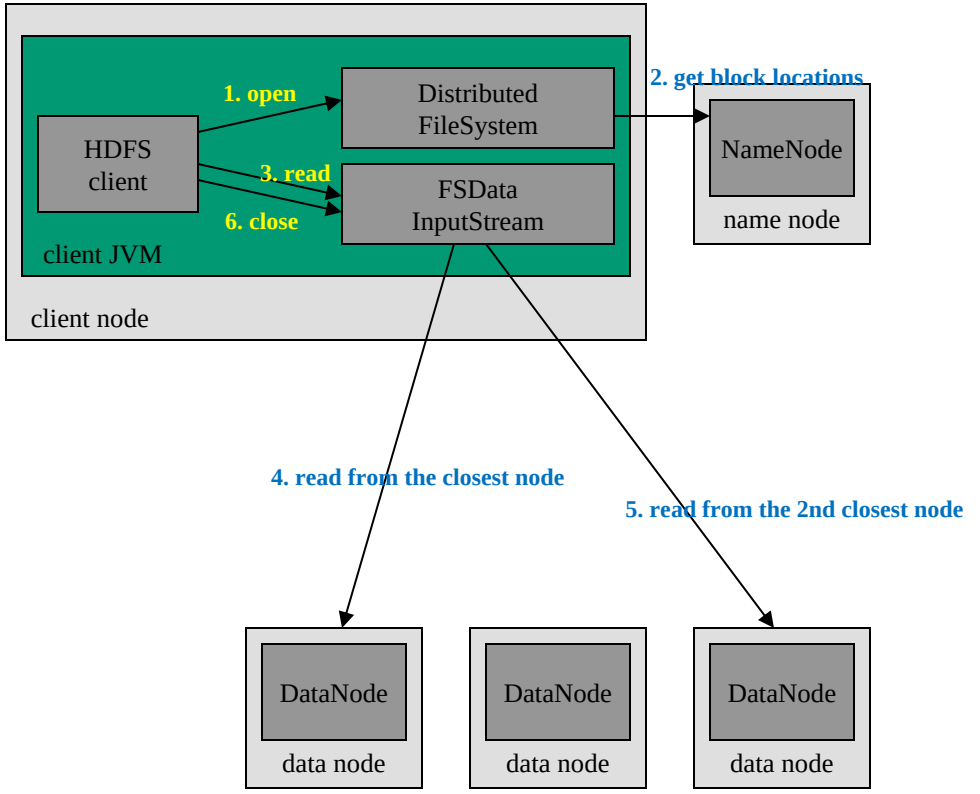
HDFS

- Client requests meta data about a file from namenode
- Data is served directly from datanode

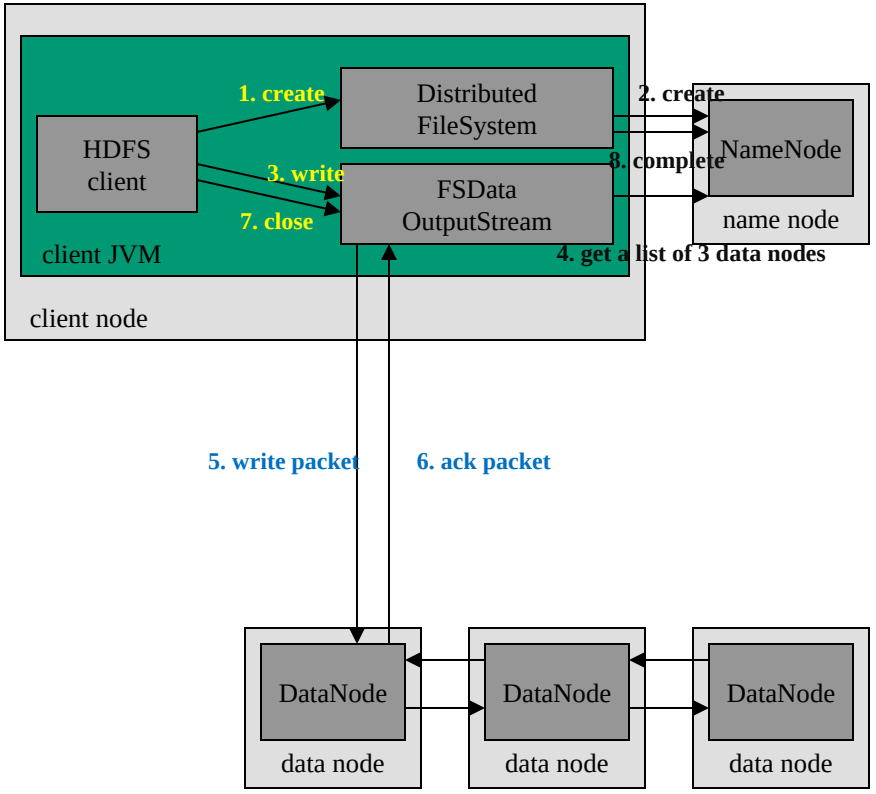


File Read/Write in HDFS

- File Read



- File Write



If a data node crashed, the crashed node is removed, current block receives a newer id so as to delete the partial data from the crashed node later, and Namenode allocates an another node.

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

Data Correctness

- Use Checksums to validate data
  - Use CRC32
- File Creation
  - Client computes checksum per 512 bytes
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution

Data Pieplining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool

Secondary NameNode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Transaction Log on NameNode is purged

User Interface

- Commads for HDFS User:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/myfile.txt
  - hadoop dfs -rm /foodir/myfile.txt
- Commands for HDFS Administrator
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommision datanodename
- Web Interface
  - http://host:port/dfshealth.jsp

Native FS to HDFS

- *copyFromLocal (src, dst)*
- *copyFromHdfs (src, dst)*
- using package
- *import org.apache.hadoop.fs.Filesystem;*

File system	GFS	KFS	Hadoop	Lustre	Panasas	PVFS2	RGFS
Architecture	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, symmetric, parallel, aggregation-based	Clustered-based, symmetric, parallel, block-based
Processes	Stateful	Stateful	Stateful	Stateful	Stateful	Stateless	Stateful
Communication	RPC/TCP	RPC/TCP	RPC/TCP&UDP	Network Independence	RPC/TCP	RPC/TCP	RPC/TCP
Naming	Central metadata server	Central metadata server	Central metadata server	Central metadata server	Central metadata server	Metadata distributed in all nodes	Metadata distributed in all nodes
Synchronization	Write-once-read-many, Multiple-producer/single-consumer, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases	Hybrid locking mechanism, using leases	Give locks on objects to clients	No locking method, no leases	Give locks on objects to clients
Consistency and Replication	Server side replication, Asynchronous replication, checksum, relax consistency among replications of data objects	Server side replication, Asynchronous replication, checksum	Server side replication, Asynchronous replication, checksum	Server side replication – Only metadata replication, Client side caching, checksum	Server side replication – Only metadata replication	No replication, relaxed semantic for consistency	No replication
Fault tolerance	Failure as norm	Failure as norm	Failure as norm	Failure as exception	Failure as exception	Failure as exception	Failure as exception
Security	No dedicated security mechanism	No dedicated security mechanism	No dedicated security mechanism	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy

THANK YOU !