Advanced Operating Systems

File Systems:

File Allocation Table,

Linux File System,

NTFS

Lecture 10

Case Studies of File Systems

- File Allocation Table (FAT)
- Unix File System
  - Berkeley Fast File System
- Linux File System
- Solaris File System
- Windows File System

File Allocation Table (FAT) Concepts

- Simple and common
- Primary file system for DOS and Windows
- Can be used with Windows NT, 2000, and XP
  - New Technologies File System (NTFS) is default for NT, 2000, XP, …
- Supported by all Windows and UNIX varieties

The FAT Family

- FAT 12, FAT16, FAT32
  - The number refers to the quantity of bits used in the FAT to refer to clusters

Disk Storage Review

- Data is stored on disks one entire sector at a time
  - A sector is usually 512 bytes
  - If you use only one byte, the system still provides the other 511 bytes for you
  - A sector is the minimum size read from, or written to, a disk
  - A sector is the minimum I/O unit

Disk Storage Review (cont.)

- Space is allocated to a file one cluster at a time
  - A cluster is a fixed number of sectors
    - Must be a power of 2 (1,2,…64)
  - Unused sectors retain the data that was on them prior to allocation
  - A cluster is the minimum file allocation unit

Slack

- Slack is the space allocated to a file, but unused
  - Space at the end of a cluster that remains unused by the file
  - Sectors allocated to the file that the file hasn’t yet used
- Slack space often contains useful evidence
  - Unused bytes in an allocated sector are less useful
  - Unused sectors in an allocated cluster retain their original contents and are very useful

Unallocated Clusters

- Many clusters on a modern hard drive are unallocated
- Unallocated clusters may have been allocated earlier though
  - These clusters retain their data until they are reallocated to a new file
  - Deleted files are still recoverable!

Cluster Allocation Algorithms

- First available
  - Always start at the beginning of the file system
  - Fragmented files common
  - Recovery of deleted content better at end of file system

Cluster Allocation Algorithms

- Best fit
  - Search for consecutive clusters that fit the size of file
  - Only works for files that don’t grow
- Next available
  - Start search with the cluster that was most recently allocated
  - More balanced for data recovery
  - Used by Windows 98 and XP

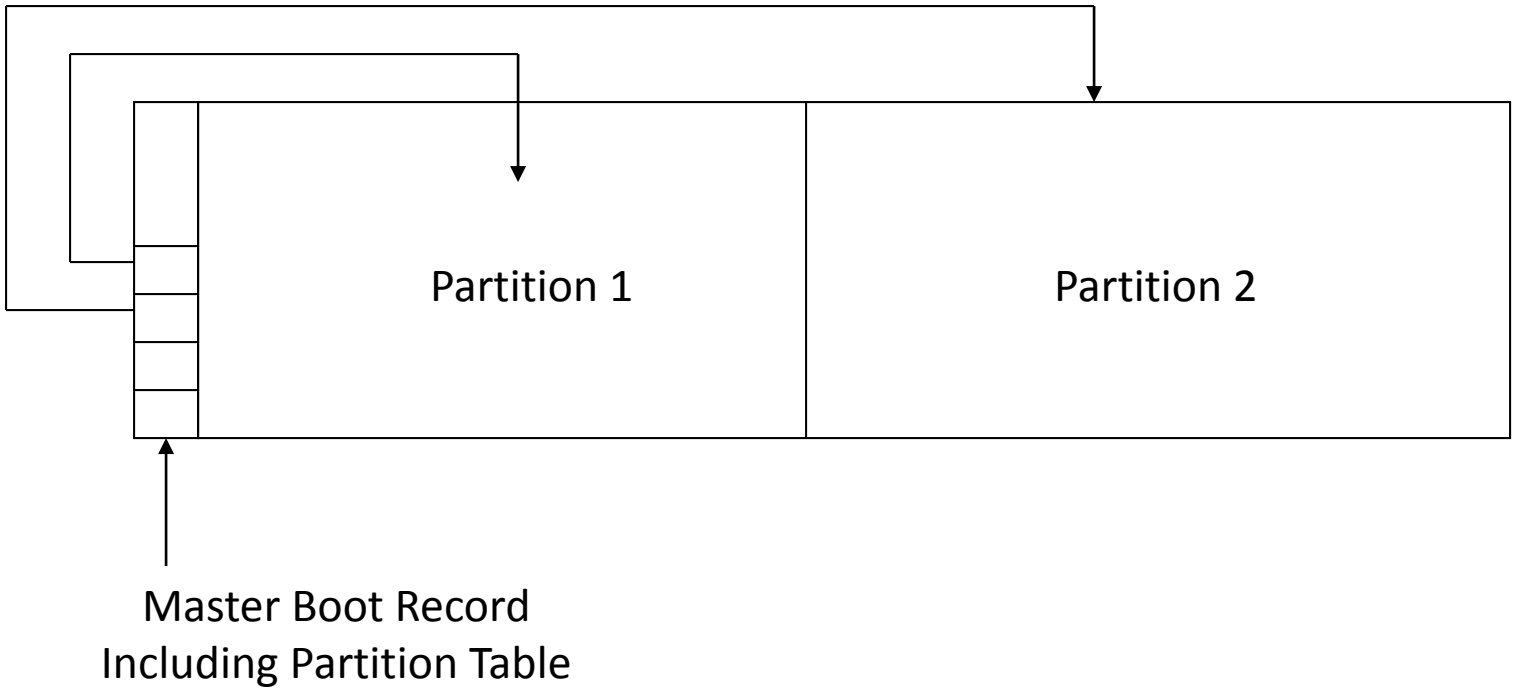
Disk Partitions

- The user creates partitions (logical drives or volumes)
  - Creates Master Boot Record with partition table
  - Each partition uses a file system
    - FAT16, FAT32, NTFS on Windows systems
    - EXT2, EXT3, UFS1, UFS2 on Linux and UNIX systems
- Recovery tools can often find data even if the disk was repartioned
  - FAT file systems have 0x55AA in bytes 510 and 511 of the partition, for example

DOS Partitions Review

- MBR in first 512-byte sector on disk
  - Boot code (Bytes 0-445)
  - Partition table (bytes 446-509)
- Partition table has four entries
  - Disk has four primary partitions
  - A primary partition may hold extended partitions

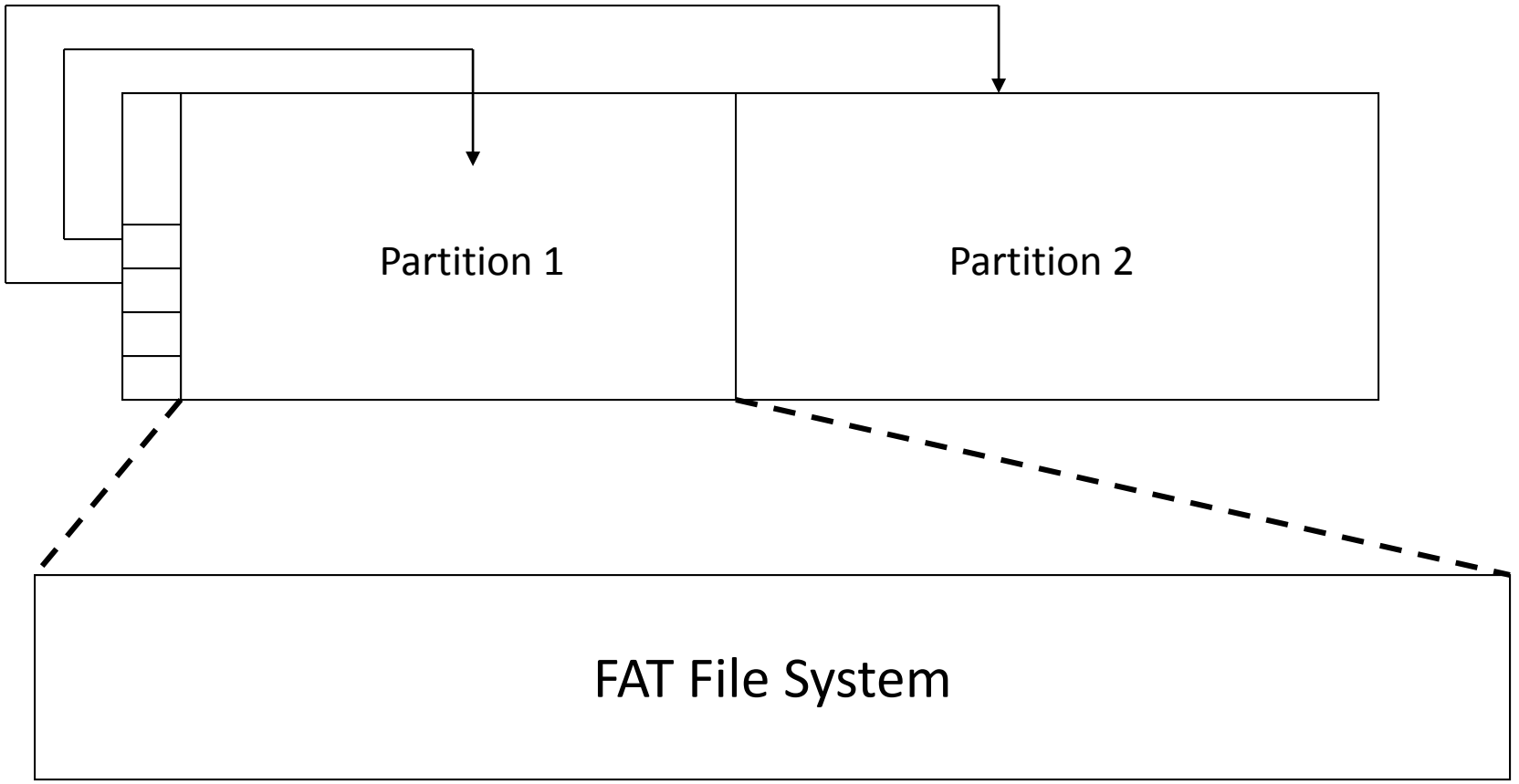
DOS Disk


Partition 1
Partition 2
Master Boot Record
Including Partition Table

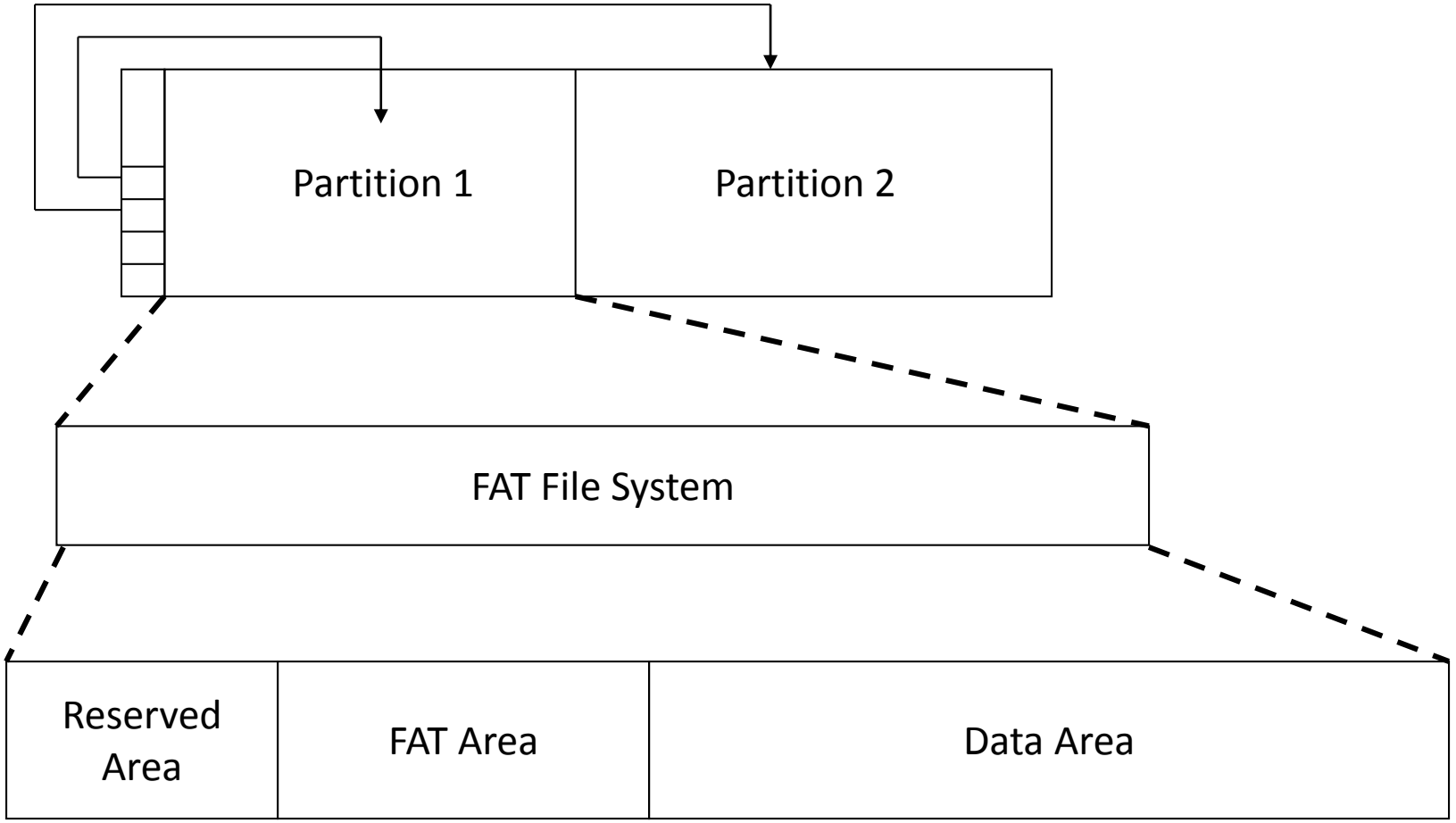
File Systems

- High-level formatting creates file system data structures
  - Root directory
  - Data that tracks which clusters are unused, allowing the OS to find available clusters quickly
    - File Allocation Table (FAT) on older Windows systems
    - Bitmap in the Master File Table (MFT) on newer Windows
  - Exact details depend on operating system

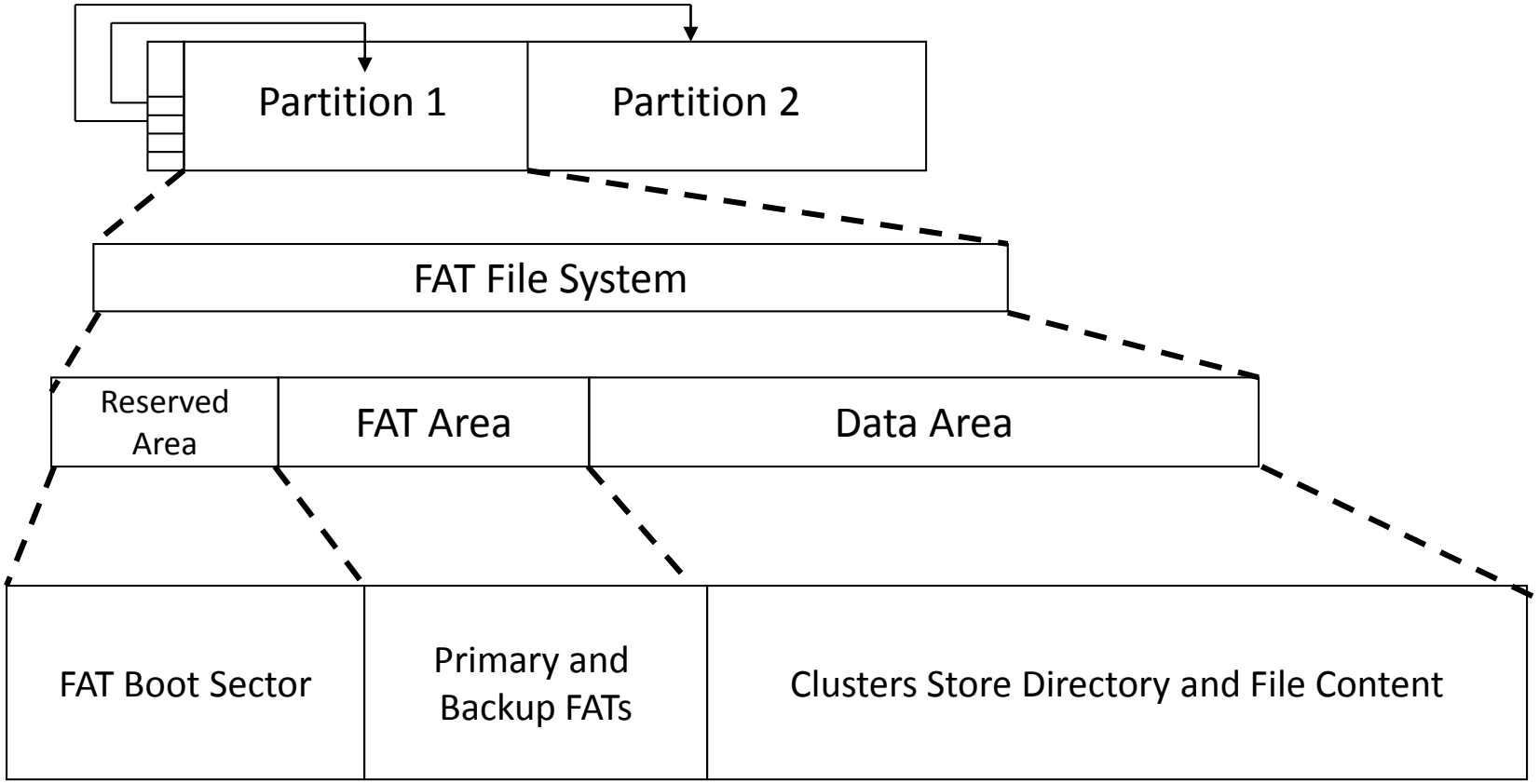
Partition Holds a File System from the FAT Family



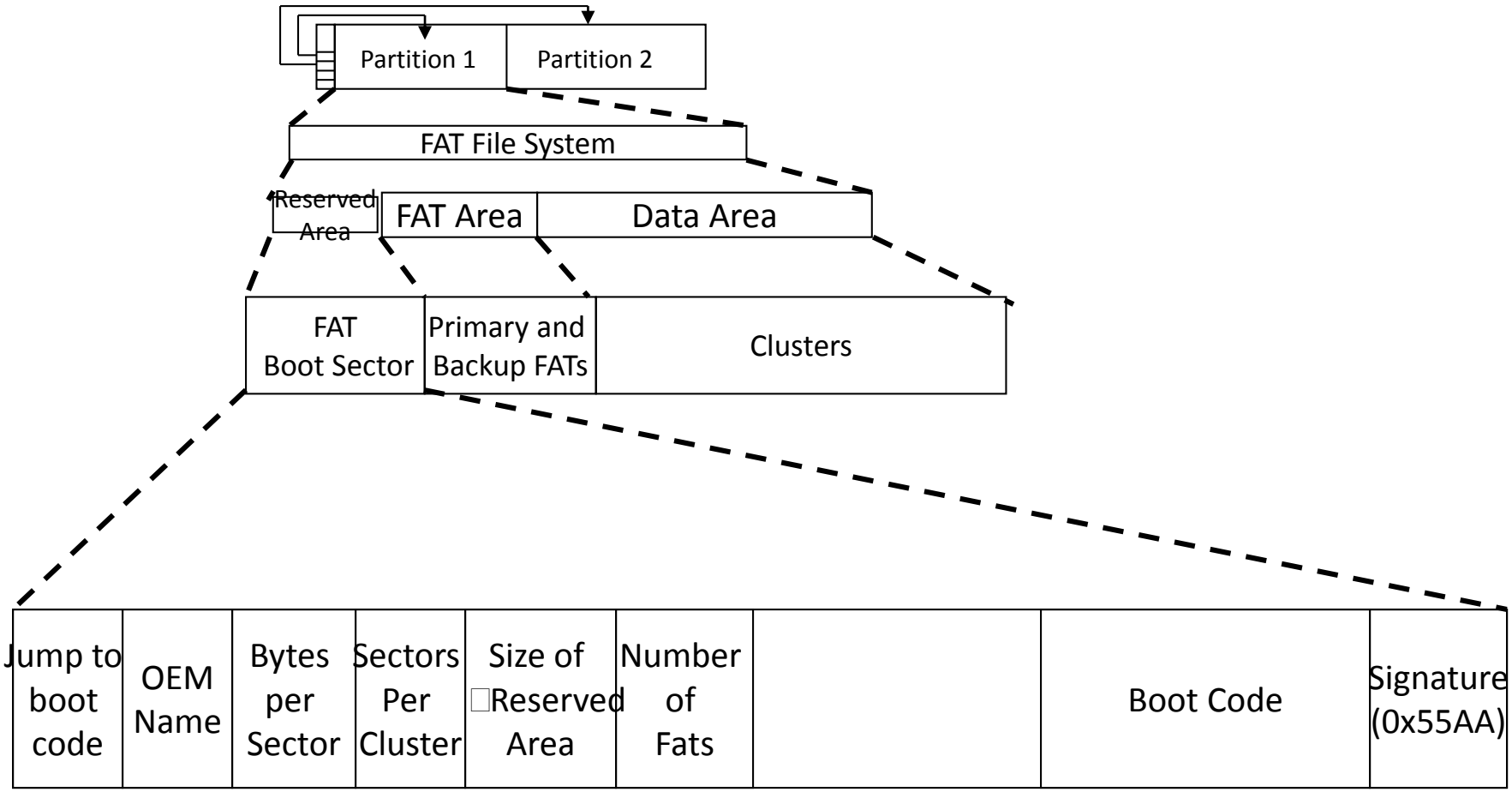
FAT Family File System



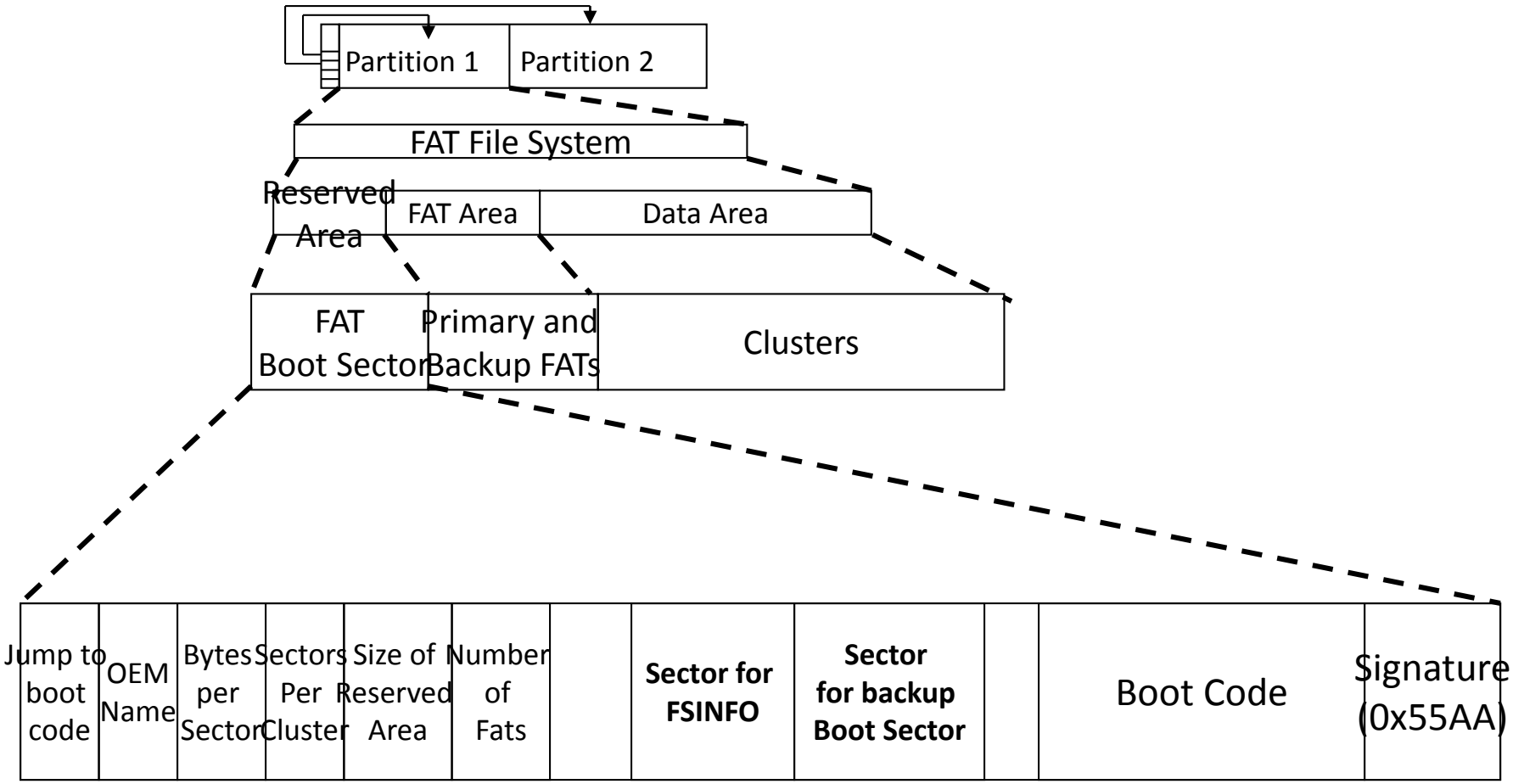
FAT File System Layout



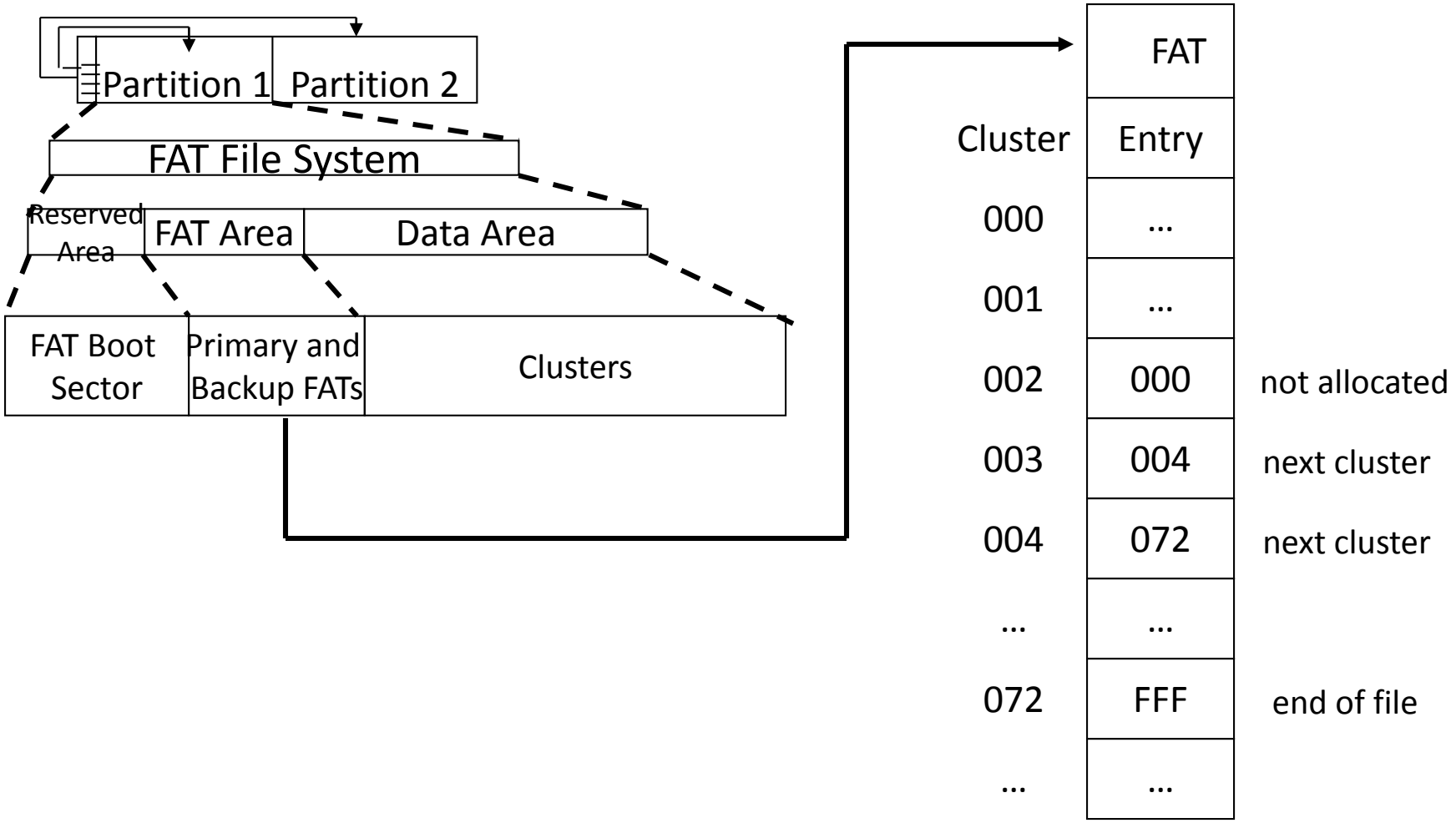
FAT File System Boot Sector



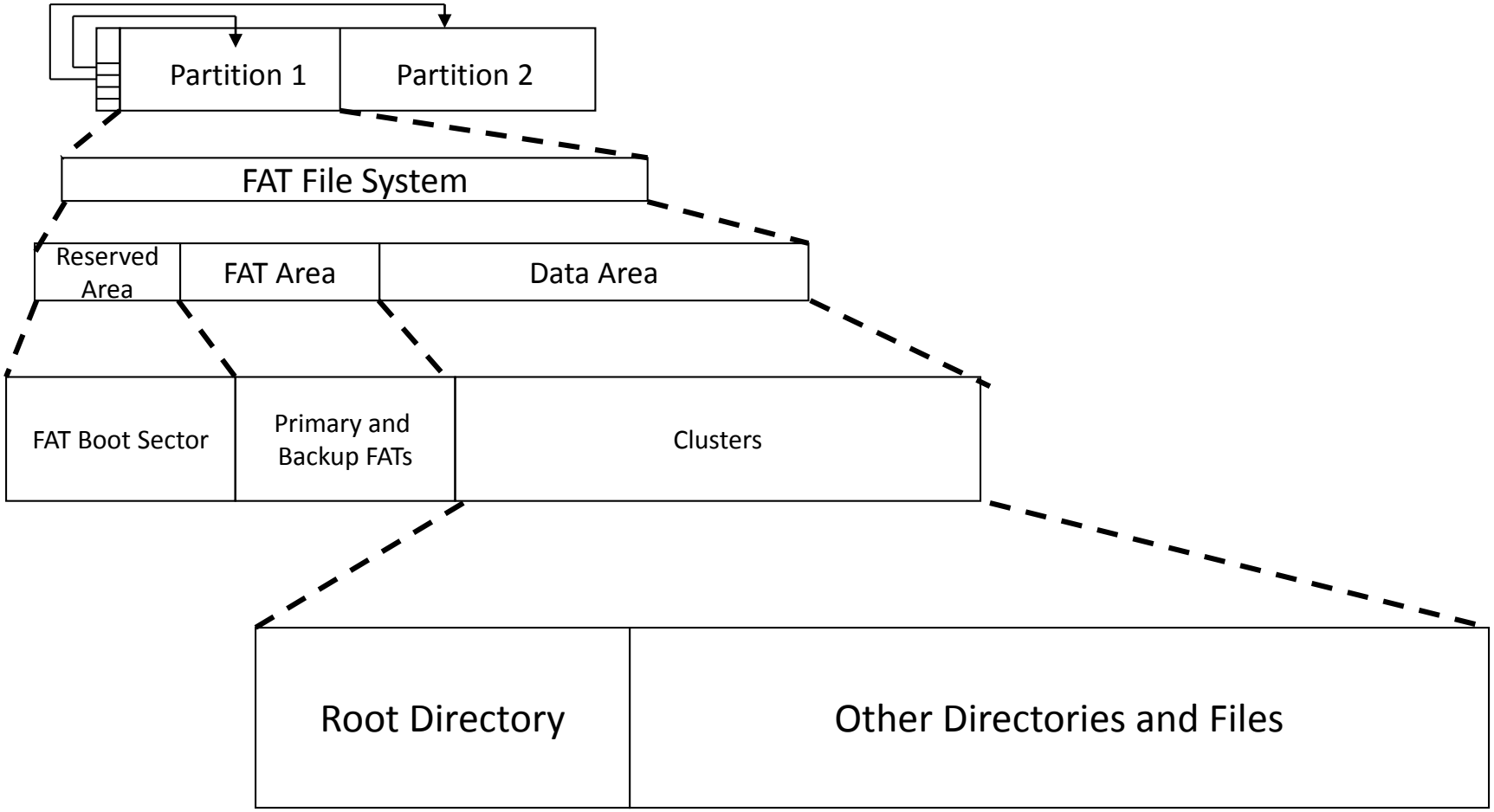
FAT32 Boot Sector



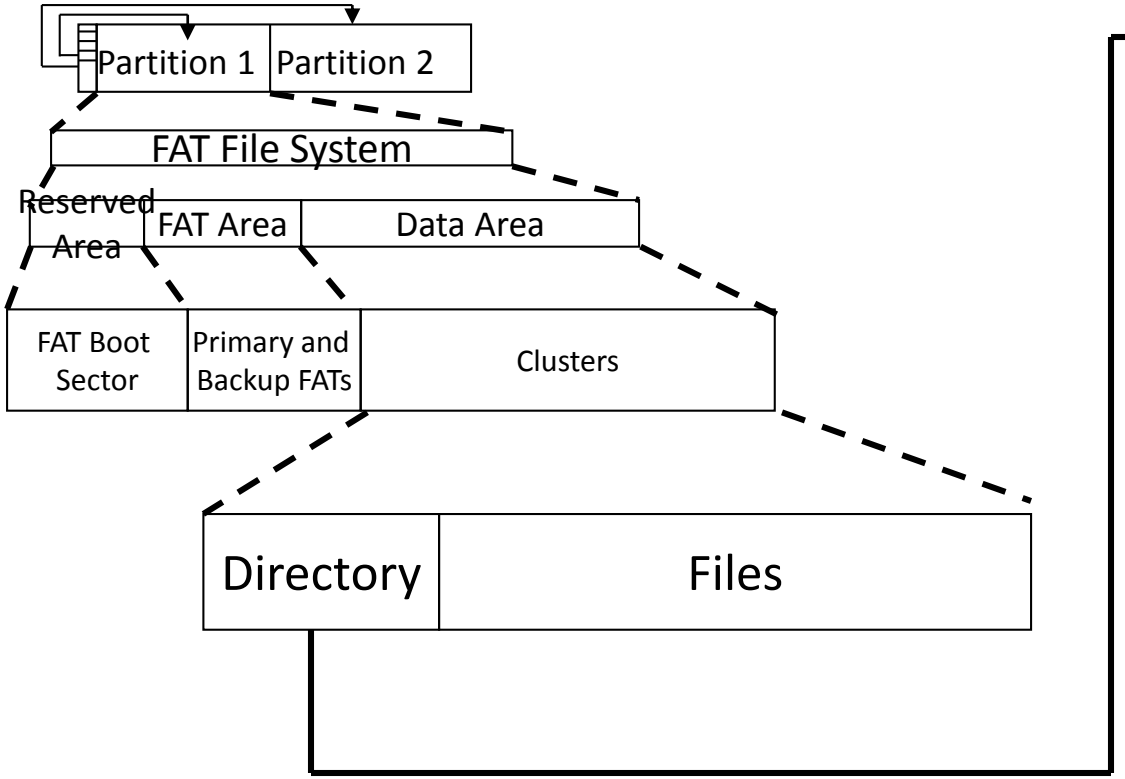
File Allocation Table Concepts



Data Area Concepts



FAT Directories



Dir Attributes

Long File Name
8.3 Filename
File attributes (read only, hidden, system, archive, etc.)
Created time/day
Accessed day
Modified time/day
First cluster
address Size of file (0 for directory)

FAT Advantages/Disadvantages

- One block cannot be assigned to two files
- Fast random access since FAT is in main memory
- Takes a lot of space from Main memory
- Max 2GB for blocks of size 32k

Linux File System

Linux File System

- Linux provides a virtual file system (VFS)
  - Supports a common file model that resembles the Unix file model
- Standard file system is ext2
  - Variety of file locks for process synchronization
    - *Advisory locks*, *mandatory locks*, *leases*
  - Uses notion of a *block group*
- ext3 incorporates journaling

System vs file system

- The file system resides on a single logical disk or partition
- A partition can be viewed as a linear array of blocks
  - block represents the granularity of space allocation for files
  - a disk block is 512 bytes * some power of 2
  - physical block number identifies a block on a given disk partition
  - physical block number can be translated into physical location on a partition

Disk partition

- Boot area
  - Code required to bootstrap the operating system
- Superblock
  - Attributes and metadata of the file system itself
- i-node list
  - a linear array of i-nodes
- data blocks
  - data blocks for files and directories, and indirect blocks

Superblock

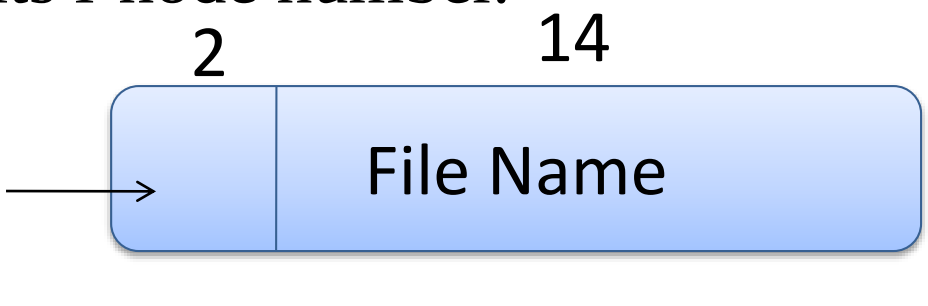
- UNIX/Linux must **mount** a file system first
- The kernel reads the **superblock** and stores it in **memory** when mounting the file system
- A **superblock** contains information about block layout on a specific partition
  - Size in blocks of the file system
  - Size in blocks of the i-node list
  - Number of free blocks and i-nodes
  - Free block list

i-node

- Each file has an i-node associated with it
- i-node contains metadata for file
- on-disk i-node refers to i-node stored in disk within the i-node list
- in-core i-node refers to i-node stored in memory when a file is open

I-node…

- **Kernel** memory contains a copy of the i-node for every open file.
- A directory entry holds the filename and a reference to its associated i-node or its i-node number.
- A directory entry



- Number of inodes limited

On-disk i-node

- The size of on-disk i-node is **64** bytes

Field	Size	Description
mode	2	File type, permissions
uid	2	Owner UID
gid	2	Owner GID
size	4	Size in bytes
addr	39	Array of block addresses-13 items
:	:	:

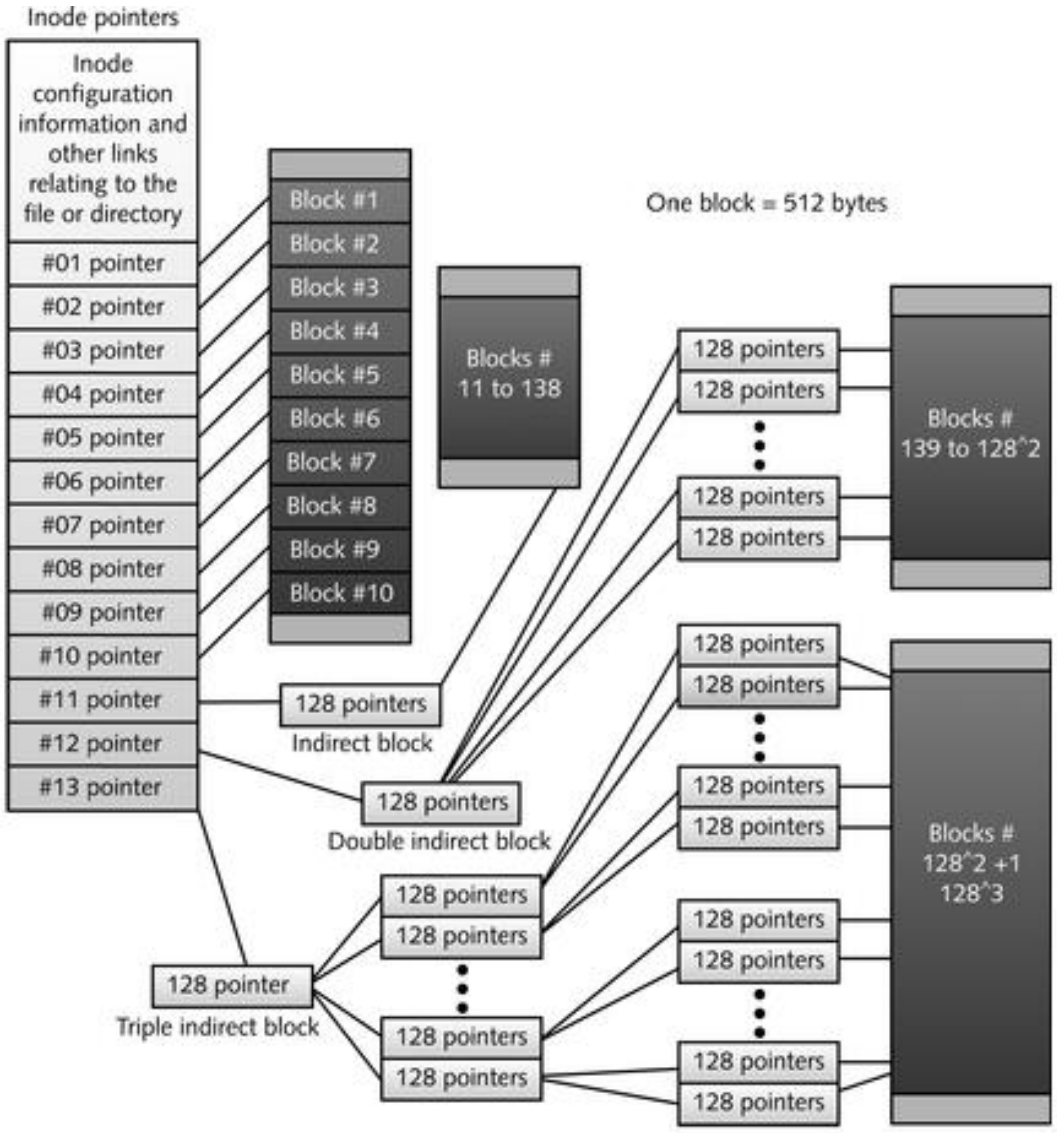
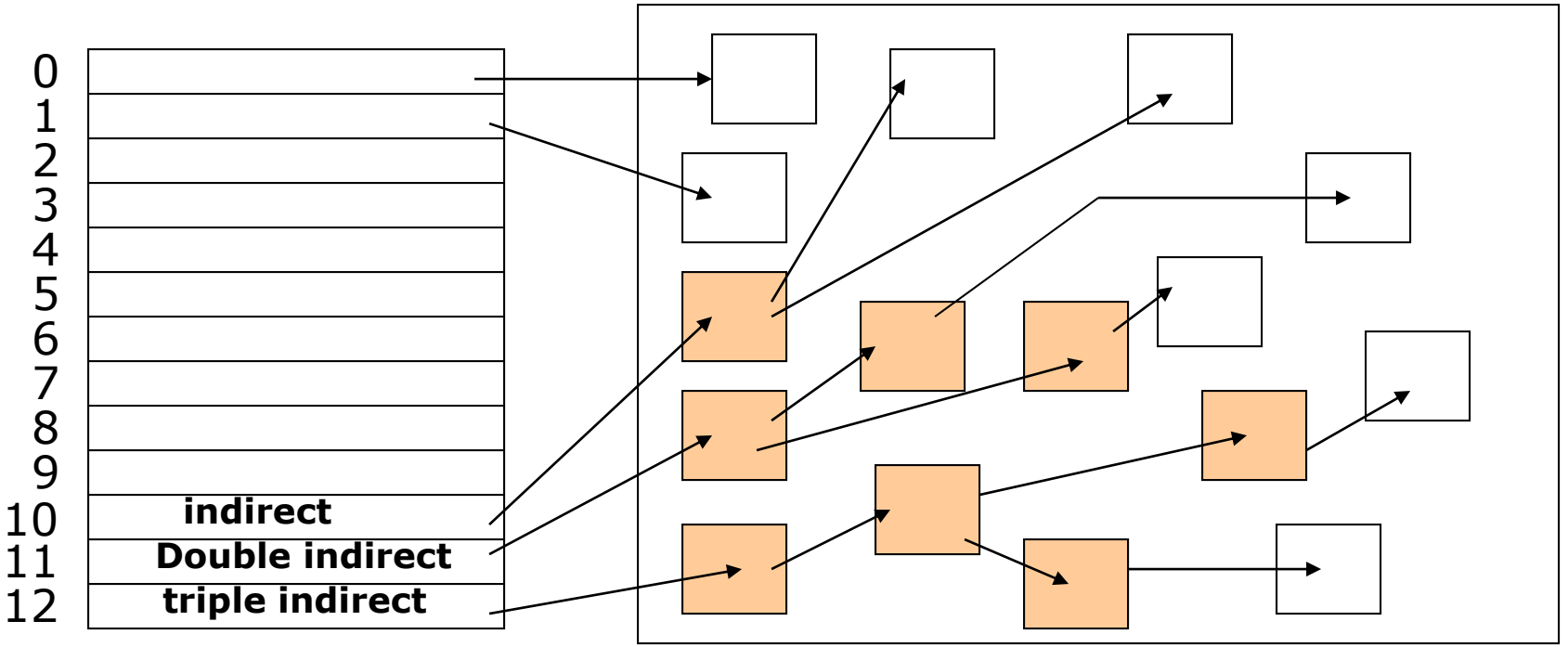


Figure 4-8 Linux file system inode pointers

On-disk i-node

- Unix files are not contiguous on disk
- File system need to maintain a map of the disk location of every block of the file



How large can a file be?

- Depends on
  - block sizes
  - Size of disk addresses used
- Example : assuming maximum block size = 4KB
  - Direct block – (10) direct blocks) = 40KB
  - Indirect block – assuming each level of indirection can access 256 blocks
    - Level 1 : 256blocks
    - Level 2 : 256*256 blocks
    - Level 3 : 256*256*256 Blocks
    - Total = (10+256 +256*256 + 256 * 256 * 256) * 4kB

File size

- Very large files possible
  - Can try with other block sizes
- However file access time will be slow due to multiple level indexes
- Most unix variants do not use > 2 level index
  - Due to incompatibility with storage hardware

i-nodes (1)

An assigned i-node contains the following information about a file or directory:

-The mode and type of the file or directory.

-The number of links to a file or directory.

-The UID and GID of the file's or directory's owner.

-The number of bytes contained in the file or directory.

-The file's or directory's last access time and last modified time.

-The i-node's last status change time.

i-nodes (2)

-Ten block address for the file data.

-The indirect, double indirect, and triple indirect block addresses for the file data.

-Current usage status of the i-node.

-The number of actual blocks assigned to the file.

-File generation number and version number

i-node operation

- i-node lookup:
  - translates a pathname and returns a pointer to the vnode of the desired file
- allocate i-node:
  - read an i-node from disk into memory by i-node number or initialize an empty i-node if not found
- release i-node:
  - kernal writes the i-node to disk if the in-core copy differs from the disk copy

I-node Operation

- ✓i-node lookup:
  translates a pathname and returns a pointer to the vnode of the desired file
- ✓allocate i-node:
  read an i-node from disk into memory by i-node number or initialize an empty i-node if not found
- ✓release i-node:
  kernal writes the i-node to disk if the in-core copy differs from the disk copy

Directories

2 bytes	14 bytes
1	.
1	..
4	Bin
7	Dev
14	Lib
9	Etc
6	user

- UNIX/Linux single **dot (.)** : the current directory
- Two **dots (..) :**the parent directory
- **cd ..** moves you up a level in the directory structure

Example

Root directory

1	.
1	..
4	Bin
7	Dev
14	Lib
9	Etc
6	user

Looking for /user

i-node 6 /user

Attributes
132

/user is in block 132

Block 132 /user directory

6	.
1	..
19	Dick
30	Erik
51	Jim
26	Ast
45	bal

/user/ast is i-node 26

Cont…

I-node 26 is for /user/ast

Attributes
406

Block 406 is /user/ast directory

26	.
6	..
64	Grants
92	Books
60	Mbox
81	Minix
17	src

i-node 26 says /user/ast is in 406

/user/ast/mbox is In i-node 60

Advantage/Disadvantage

- Can be shared while in main memory
- Fast processing
- Updates are not instant

NTFS
(New Technology File System)

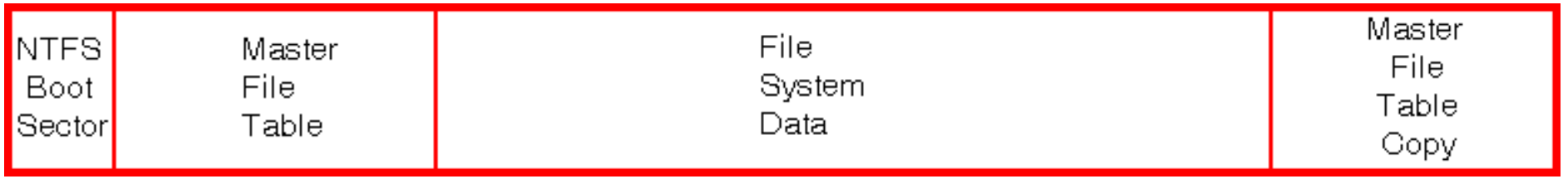
Windows File System

- NTFS is designed for servers and workstations
  - Key feature: recoverability of the file system
- Notion of *partition* and *volumes (*single *and spanned);* volumes have a *master file table* (MFT)
- Directory organized as a B+ tree
- Hard links and symbolic links (called *junctions*)
- Special techniques for sparse files and data compression
- Metadata modifications are atomic transactions
- *Write behind* capabilities of journaling file systems
- Vista has many new features for recovery

NTFS Block size is dynamic

- Different cluster size for diff volume size
  - <= 512 MB 512B
  - 512< <=1G 1K
  - 1G< <=2G 2K
  - 2G< <=4G 4K
  - 4G< <=8G 8K
  - 8< <=16G 16K
  - 16< <=32G 32K
  - >= 32g 64K

NTFS Architecture



* Boot sector signature is 55AA

NTFS Boot Sector

- 0x00 3B Jump Instruction
- 0x03 8B OEM-ID (original equipment manufacturer-id)
- 0x0B 25B BPB (BIOS Parameter Block)
- 0x24 48B Extended BPB
- 0x54 426B Bootstrap Code.
- 0x1FE 2B End of Sector Marker

NTFS Boot Sector

- Many fields are not important, but:
  - 0x0B, Bytes per sector.
  - 0x0D Sectors per Cluster
  - 0x15 Media descriptor. F8: HD; F0: HD Floppy
  - 0x28 Total sectors.
  - 0x30 Logical cluster number for the MFT
  - 0x38 Logical cluster number copy of the MFT
  - 0x40 Clusters per MFT Record.
  - 0x48 Volume serial

NTFS Master File Table

- First four entries are replicated, so that MFT can be repaired
- First 16 records are reserved for metadata files, their name begins with a dollar sign ($)

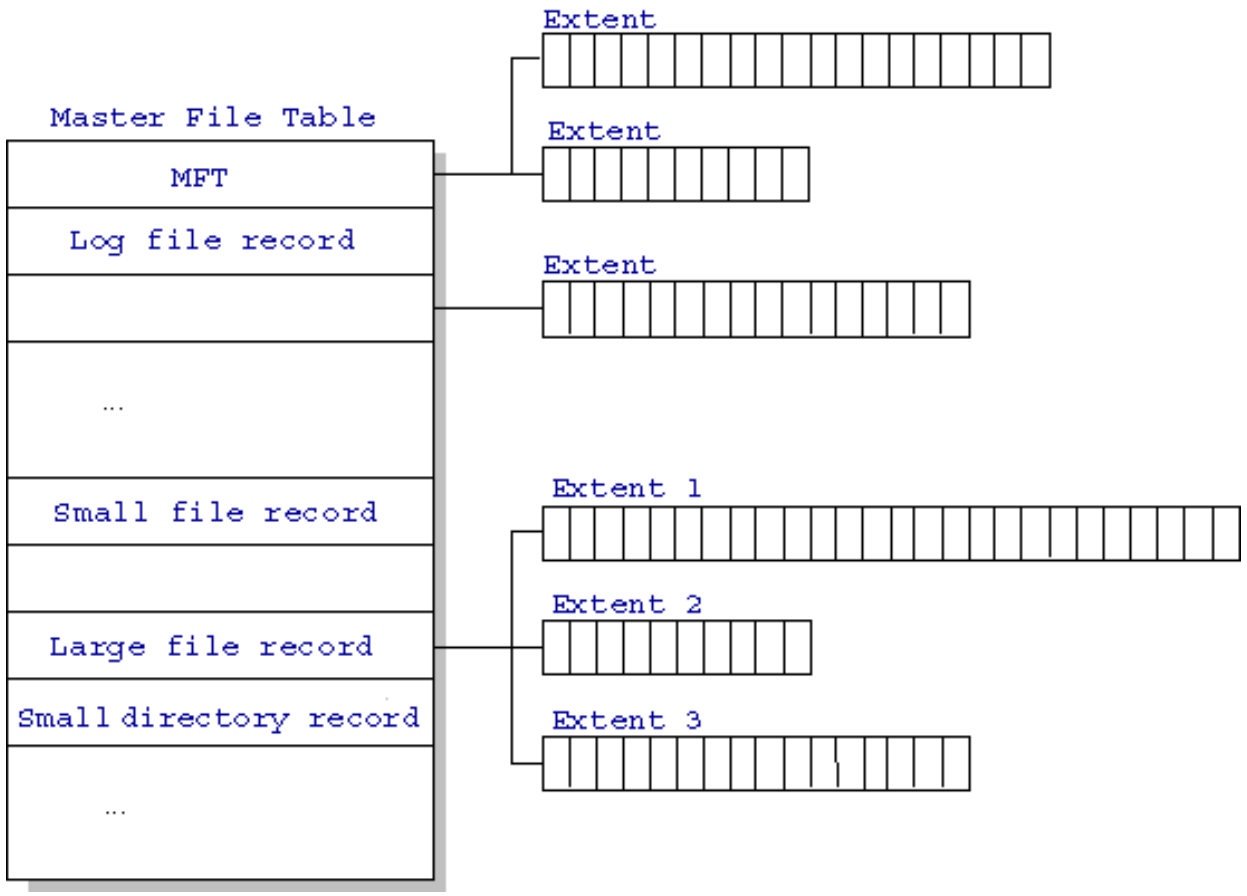
NTFS Master File Table

- Main data structure
- Sequence of 1KB
- Each row for one file (a file can use more than one record)

Attributes, data block addresses

1. Master file table $MFT.
2. Master file table mirror $MftMirr.
3. Log file $LogFile.
4. Volume $Volume Attribute definitions $AttrDef.
5. The root folder “.”
6. Cluster bitmap $Bitmap
7. Boot sector $Boot (located at the beginning of partition)
8. Bad cluster file $BadClus
9. Security file $Secure (list of all the Security Descriptors on the volume)
10. Upcase table $Upcase (Table used for converting uppercase and lowercase characters to the matching uppercase Unicode characters)
11. NTFS extension file $Extend, that is used for future use.

NTFS Master File Table

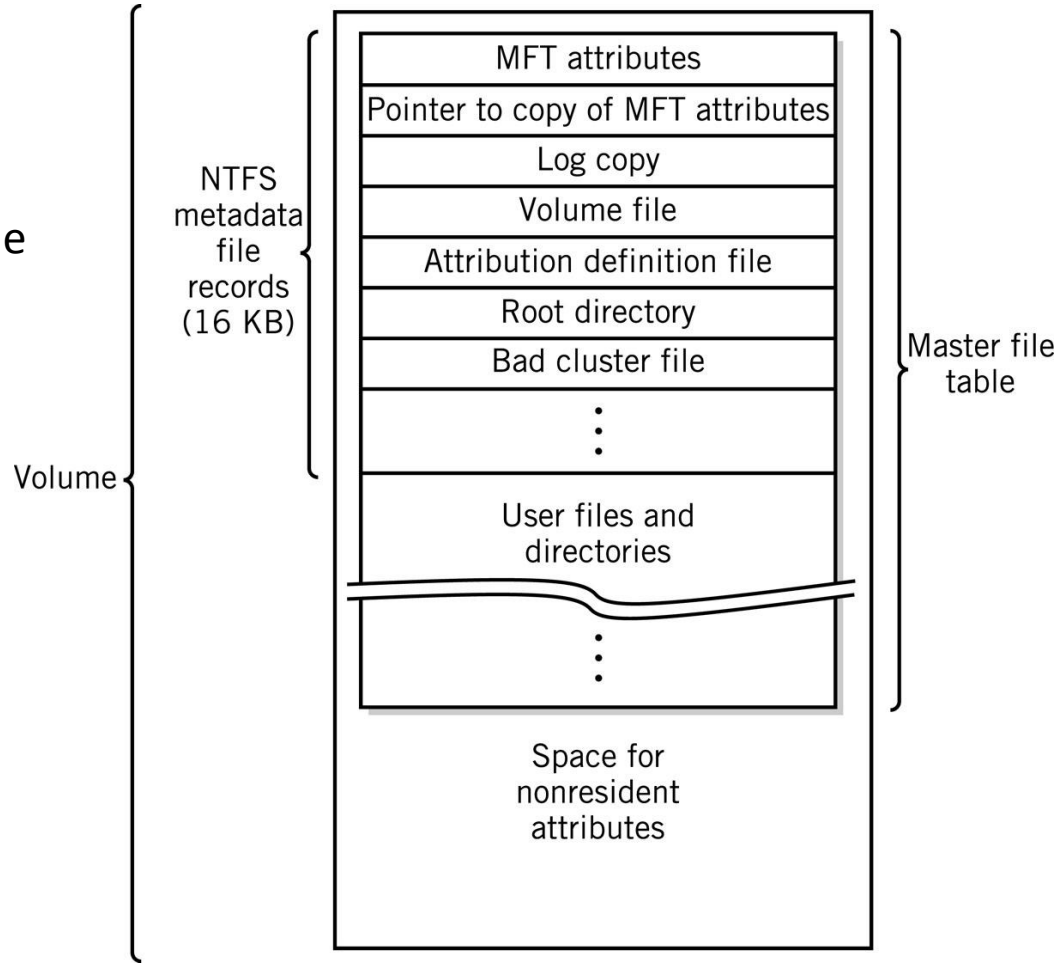


Master File Table (MFT)

-Each record is 1K long

-16 rows are special

-The rest, one record per file



MFT Record Structure

- Entries are 1KB each
- Entries for large files and directories contain
  - File Attributes
  - Location Data
- Entries for small files or directories contain
    - File attributes
    - Data

MFT Records

- Small Files or directories (<900B) are contained completely in the MFT entry.

Standard information
Directory and file names for directories, or data for files
Unused space

MFT Records

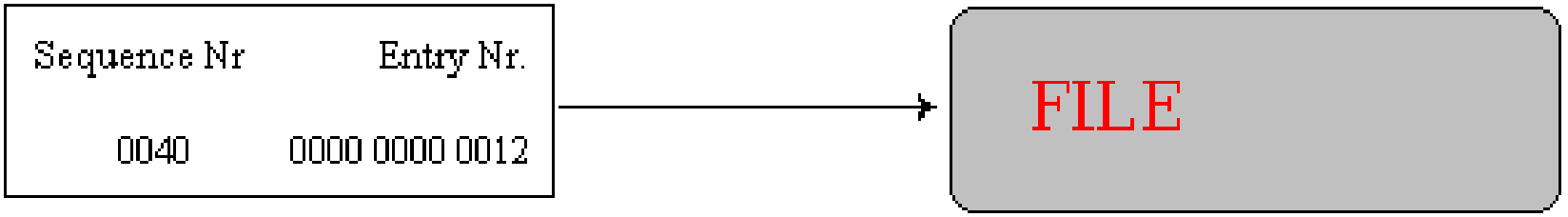
- Folders contain index data to represent B+tree.
- Small folders reside within the MFT record
- Larger folders have an index structure to other data blocks. They use a B+tree structure.

MFT Record

- Each MFT record is addressed by a 48 bit MFT entry value.
  - First entry has address 0.
- Each MFT entry has a 16 bit sequence number that is incremented when the entry is allocated.
- MFT entry value and sequence number combined yield 64b file reference address.

MFT Record

- NTFS uses the file reference address to refer to MTF entries.
  - When the system crashes during allocation, then the sequence number describes whether the MTF entry belonged to the previous file or to the current one.



MFT Record

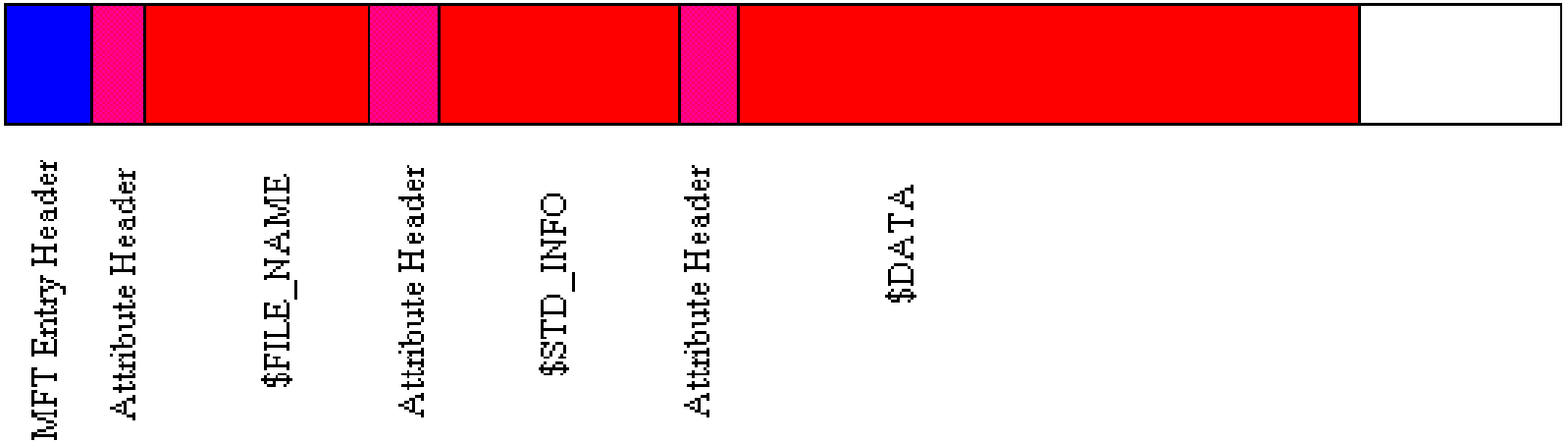
- Each attribute is preceded by the attribute header.
- The attribute header identifies
  - Type of attribute.
  - Size.
  - Name.

MFT Record Structure

- The attribute header gives basic information about the attribute.
- A resident attribute is stored in the MFT entry.
- A non-resident entry is stored in other clusters
  - Cluster run consists of consecutive clusters and are identified by starting cluster and run length.
  - NTFS distinguishes between Virtual Cluster Numbers and Logical Cluster Numbers.
    - LCN * (#sectors in cluster) = sector number
    - LCN 0 is first cluster in the volume (boot sector).
    - VCN 0 refers to the first cluster in a cluster run.

MFT Record Structure

- MFT entry header has a fixed structure



MFT Record Structure

0x00 - 0x03: Magic Number: "FILE"

0x04-0x05: Offset to the update sequence.

0x06-0x07: Number of entries in fixup array

0x08-0x0f: $LogFile Sequence Number (LSN)

0x10-0x11: Sequence number

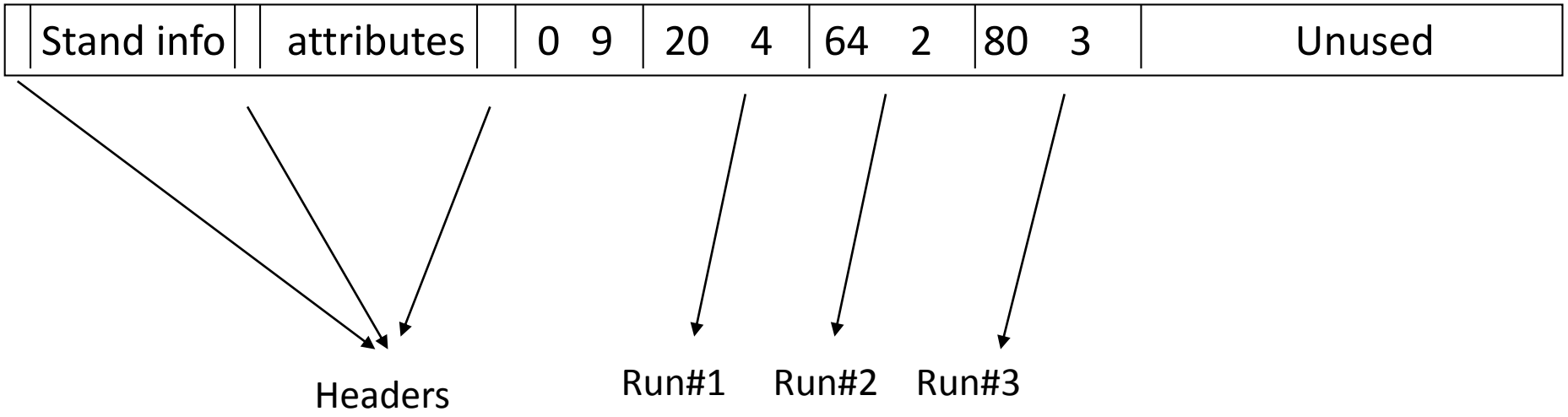
0x12 - 0x13: Hard link count

0x14-0x15: Offset to first attribute

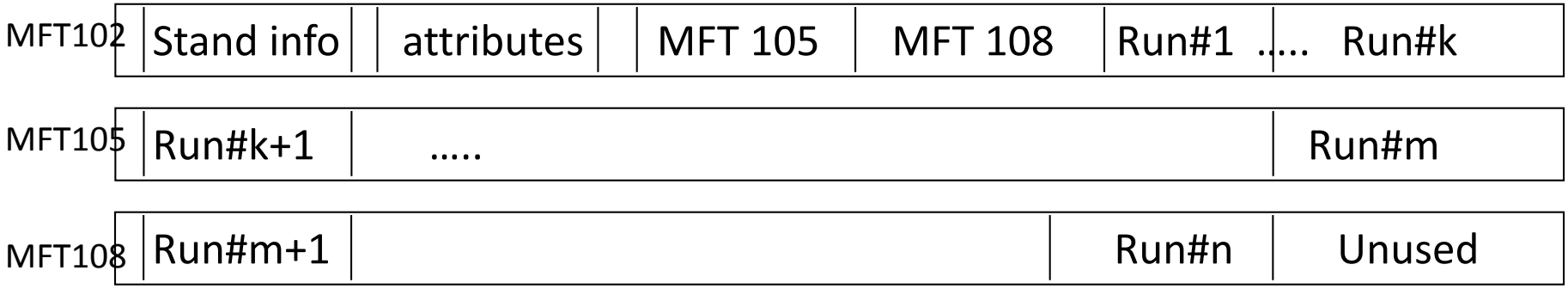
MFT Record Structure

0x16 - 0x17: Flags: 0x01: record in use, 0x02 directory.
0x18-0x1b: Used size of MFT entry
0x1c-0x1f: Allocated size of MFT entry.
0x20-0x27: File reference to the base FILE record
0x28-0x29: Next attribute ID
0x2a-0x2b: (XP) Align to 4B boundary
0x2c-ox2f: (XP) Number of this MFT record
0x30-0x100: Attributes and fixup value

MFT record example



- For long files, corresponding MFT record is continued in another record of MFT
- In this case, the first record will provide the supplementary records number
- Example:

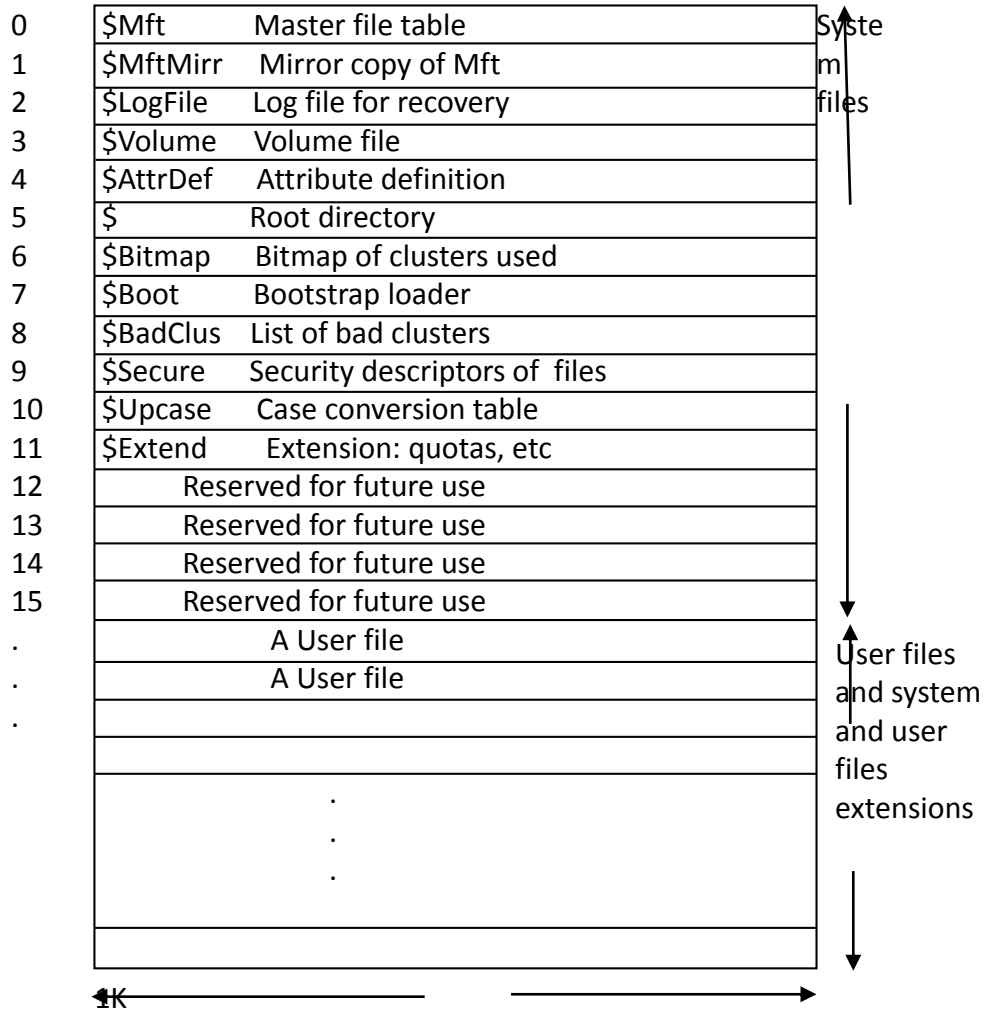


* Records 105 and 108 are first and second extension records

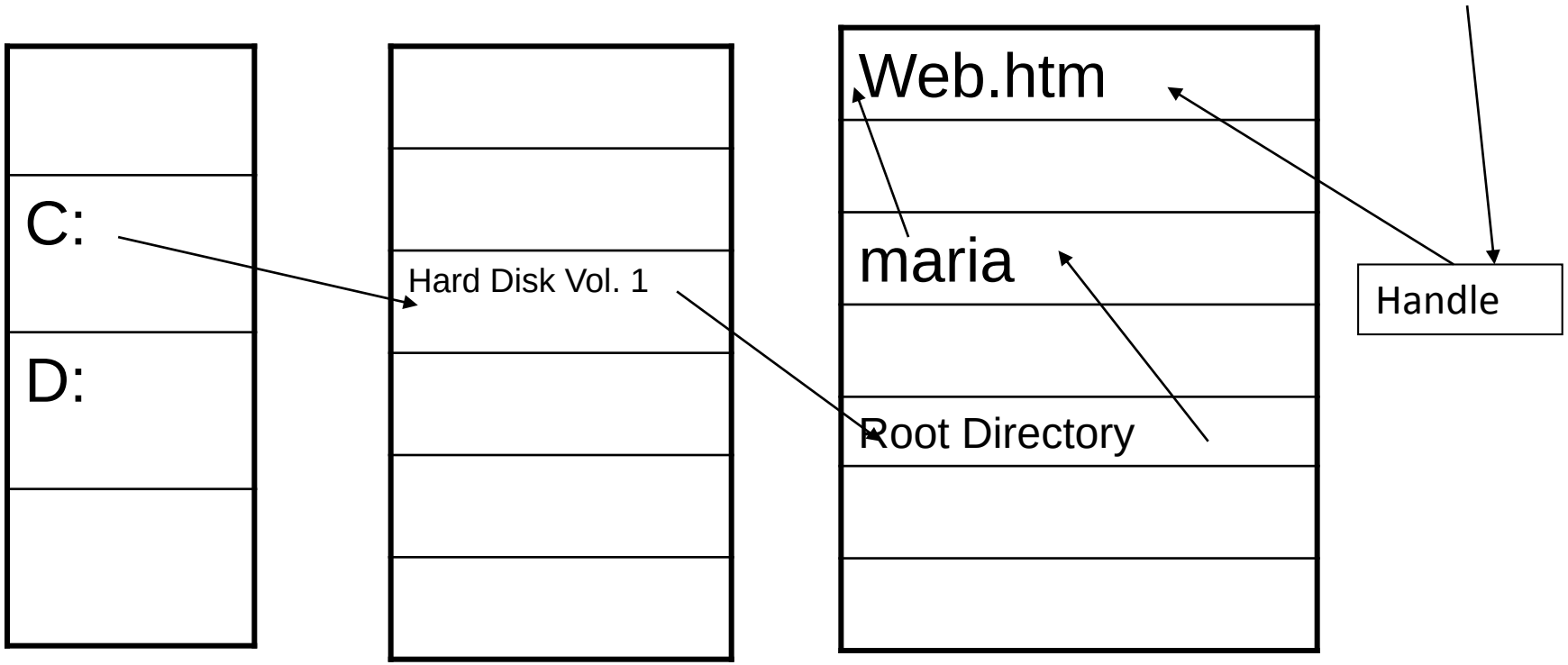
Directories

- Names in directories are stores using either
  - A linear list, for small directories or
  - a B+tree, for large directories

First 16 files



Name Lookup: Ex: C:\maria\web.htm



NTFS ads/disads

- Better cluster assignment
- Recoverability: physical changes are transaction-based
- File-level security: is definable directly using the OS
- File-level compression
- File-level encryption: directly done by the OS

NTFS Summary

- Each file on an NTFS volume is represented by a record in a special file called the master file table (MFT).
- NTFS reserves the first 16 records of the table for special information.
- The first record of this table describes the master file table itself, followed by a MFT *mirror record*.
- If the first MFT record is corrupted, NTFS reads the second record to find the MFT mirror file, whose first record is identical to the first record of the MFT.
- The locations of the data segments for both the MFT and MFT mirror file are recorded in the boot sector.
- A duplicate of the boot sector is located at the logical center of the disk.
- The third record of the MFT is the log file, used for file recovery.
- Ddirectory (also viewed as a file by NTFS) on the volume.
- Figure provides a simplified illustration of the MFT structure: