

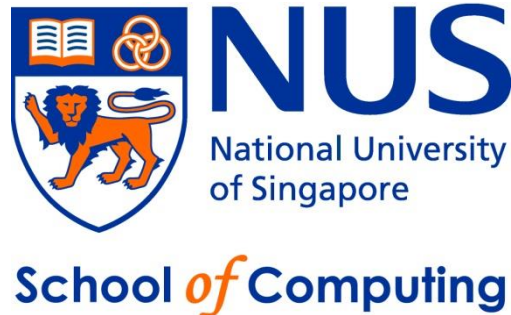
IT5002

Computer Systems and Applications

Lecture 17

File Systems

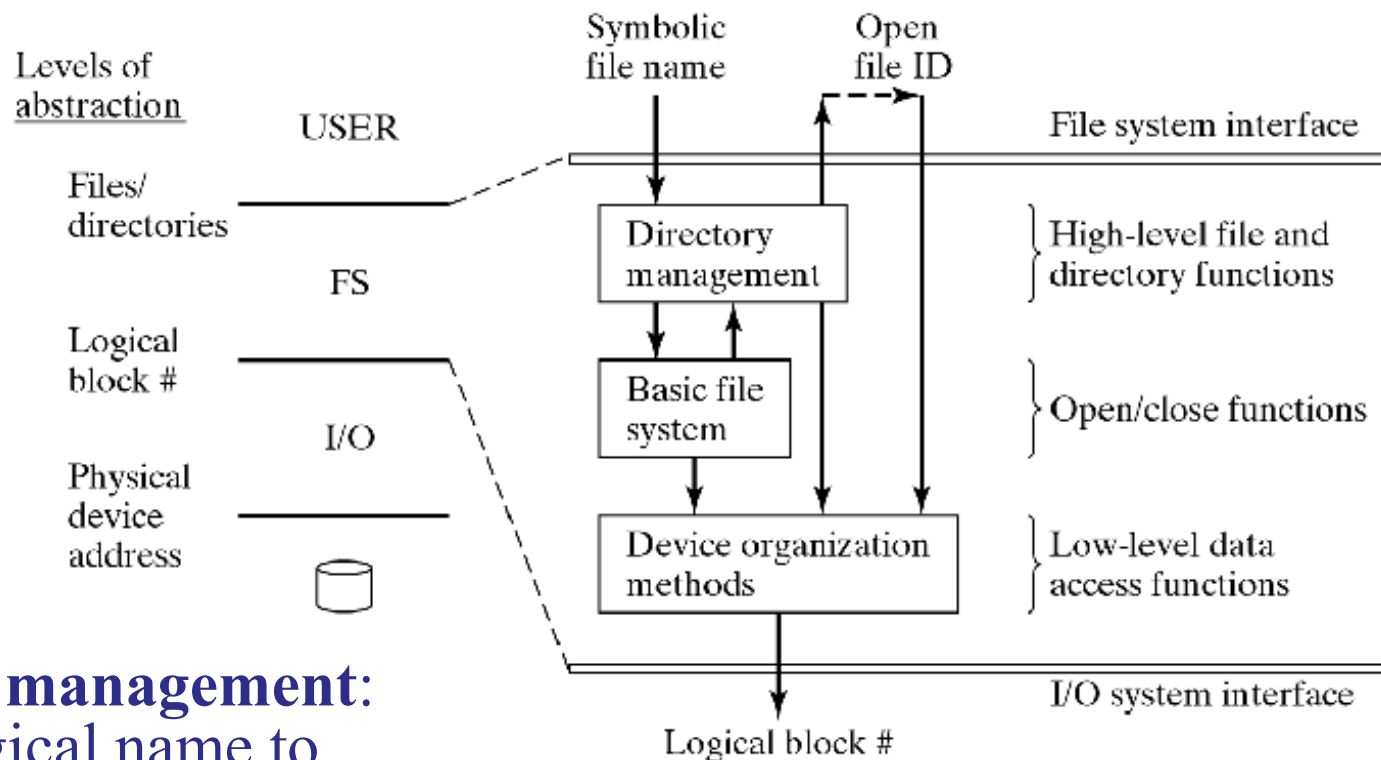
colintan@nus.edu.sg



What is a File System?

- **Present logical (abstract) view of files and directories**
 - Accessing a disk is very complicated:
 - ✓ **2D or 3D structure, track/surface/sector, seek, rotation, ...**
 - Hide complexity of hardware devices
- **Facilitate efficient use of storage devices**
 - Optimize access, e.g., to disk
- **Support sharing**
 - Files persist even when owner/creator is not currently active (unlike main memory)
 - Key issue: Provide protection (control access)

Hierarchical View of File Systems



Directory management:

- map logical name to unique Id, file descriptor

Basic file system:

- open/close files

Physical device organization:

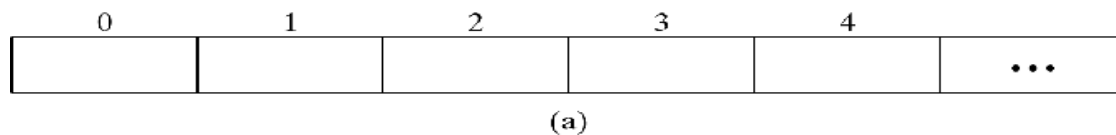
- map file data to disk blocks

User's View of a File

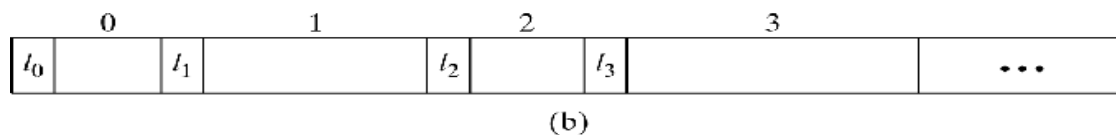
- **File name and type**
 - **Valid name**
 - ✓ Number of characters
 - ✓ Lower vs upper case, illegal characters
 - **Extension**
 - ✓ Tied to type of file
 - ✓ Used by applications
 - **File type** recorded in header
 - ✓ Cannot be changed (even when extension changes)
 - ✓ Basic types: text, object, load file, directory
 - ✓ Application-specific types, e.g., .doc, .ps, .html

User's View of a File

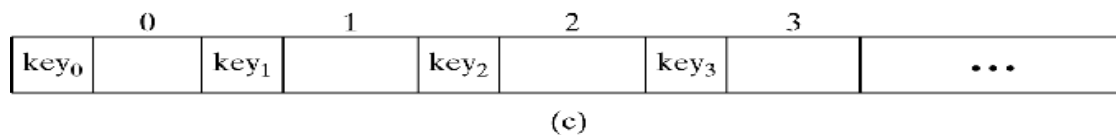
- **Logical file organization**
 - Most common: **byte stream**
 - Fixed-size or variable-size **records**
 - Addressed
 - ✓ Implicitly (sequential access to next record)
 - ✓ Explicitly by position (record#) or key



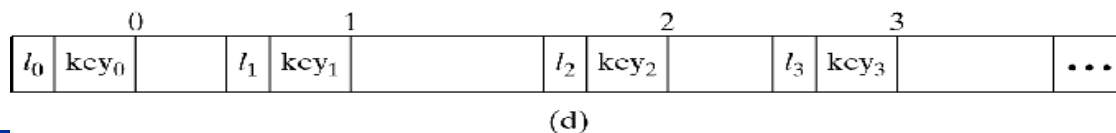
a) Fixed Length Record



b) Variable Length Record



c) Fixed Length with Key

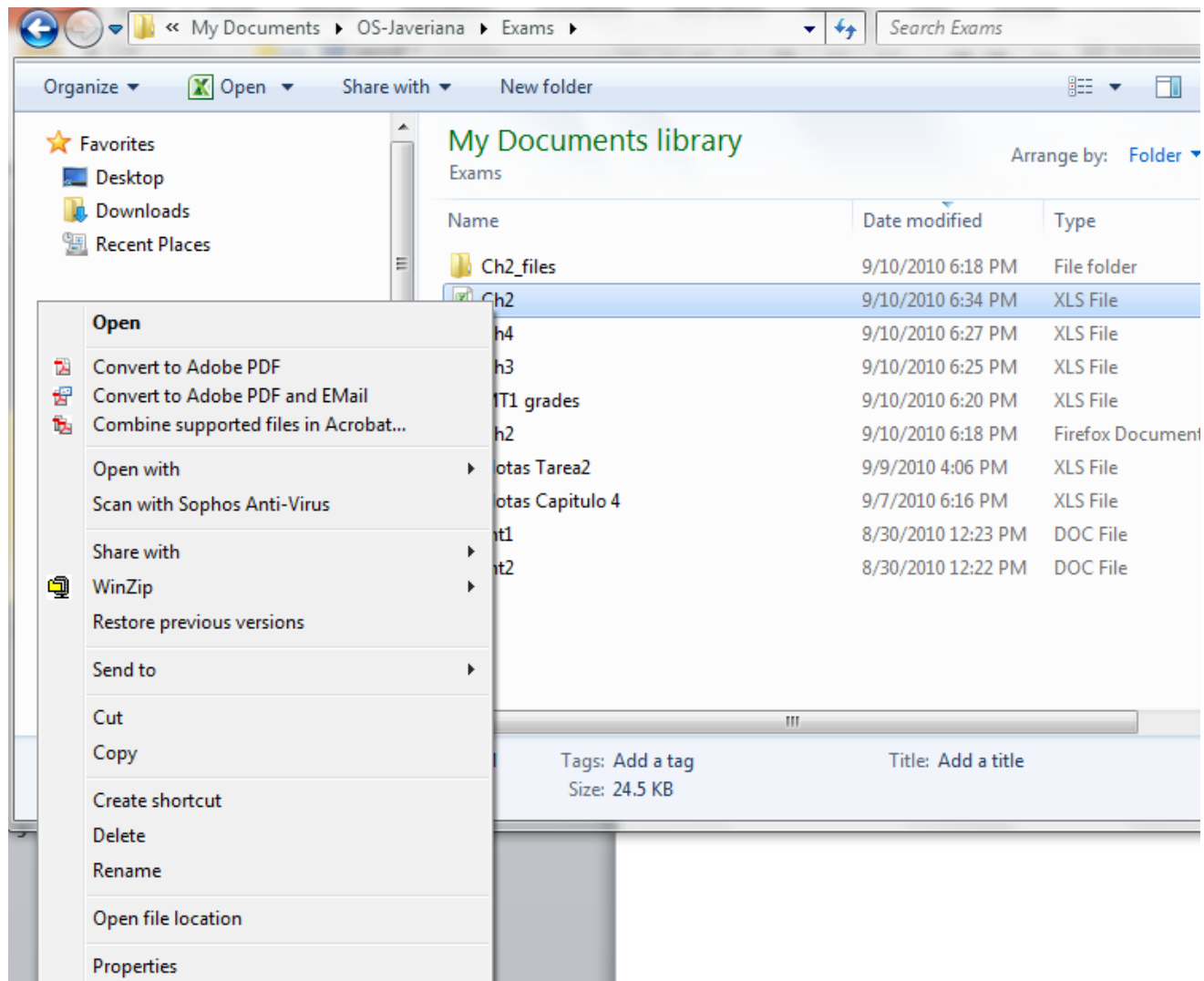


d) Variable Length with Key

Operations on Files

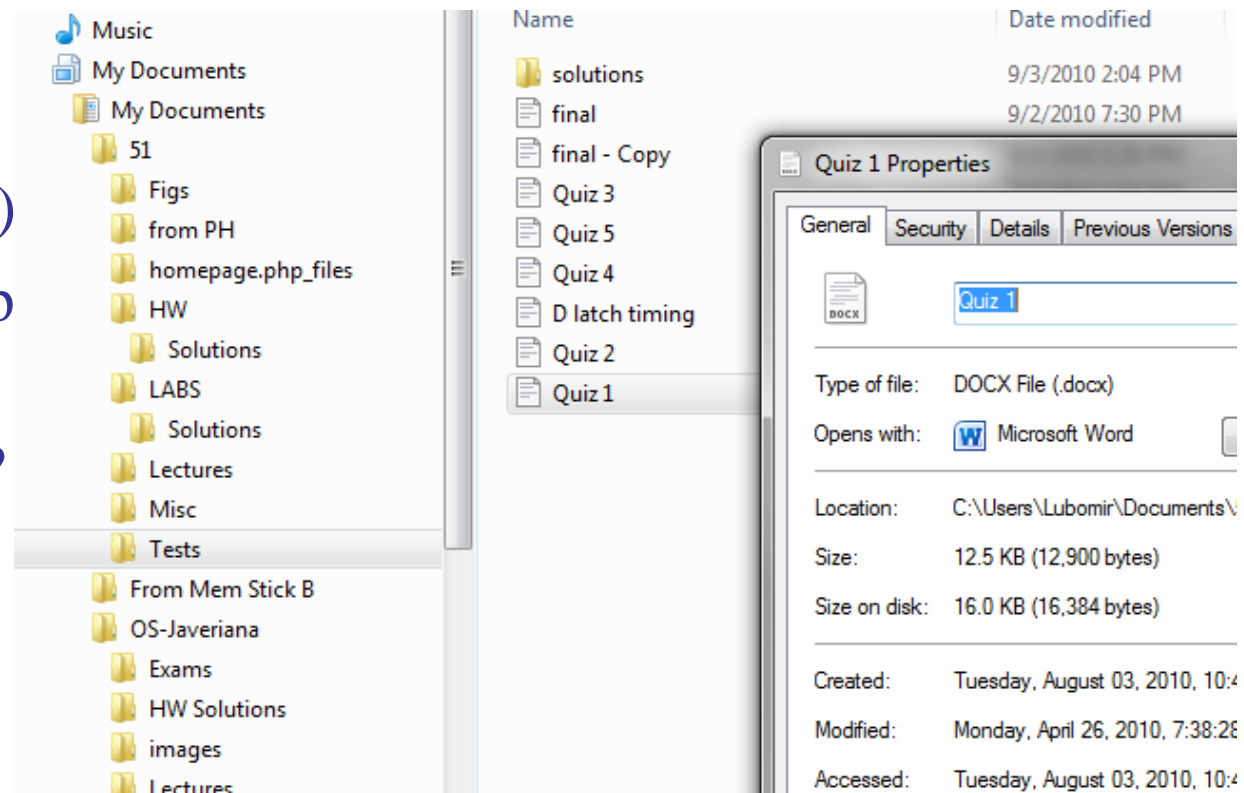
- **Create/Delete**
- **Open/Close**
- **Read/Write (sequential or direct)**
- **Seek/Rewind (sequential)**
- **Copy (physical or just link)**
- **Rename**
- **Change protection**
- **Get properties**
- **Each involves parts of Directory Management, BFS, or Device Organization**
- **GUI is built on top of these functions**

Operations on Files



Directory Management

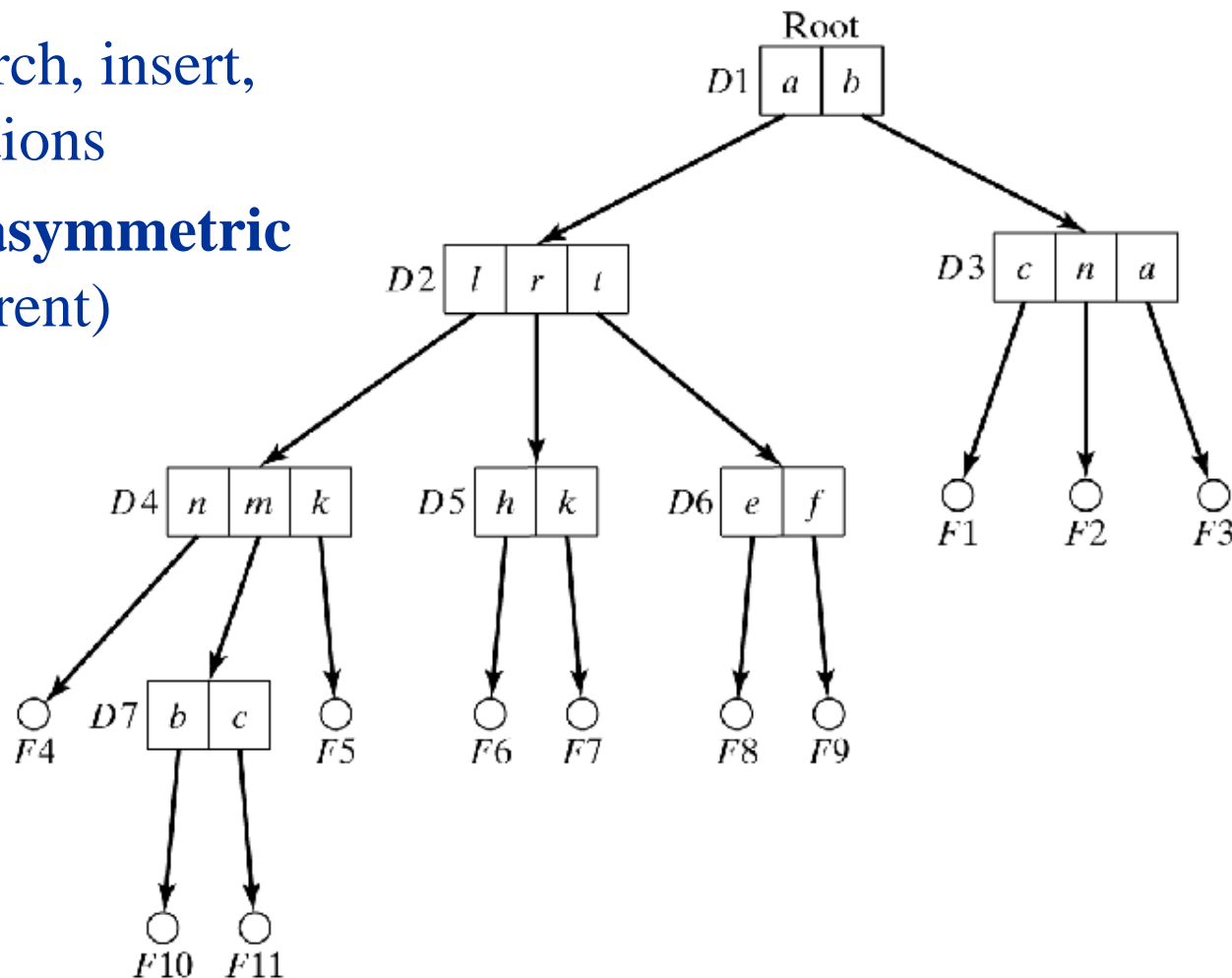
- Main issues:
 - **Shape** of data structure (e.g. tree, shortcuts, ...)
 - **What** info to keep about files
 - **Where** to keep it? (in directory?)
 - **How** to organize entries for efficiency?



File Directories

- **Tree-structured**

- **Simple** search, insert, delete operations
- Sharing is **asymmetric** (only one parent)



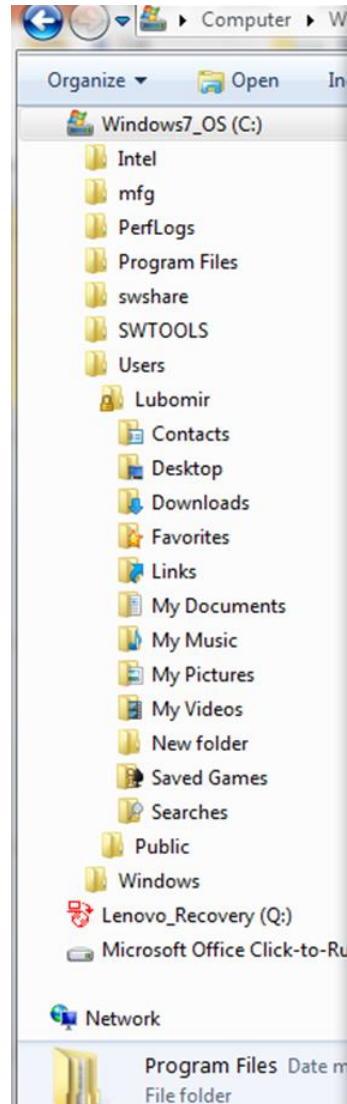
File Directories

- **How to uniquely name a file in the hierarchy?**
- **Path names**
 - Concatenated local names with delimiter:
(. or / or \)
 - **Absolute** path name: start with root
(/)
 - **Relative** path name: Start with current directory
(.)
 - Notation to move upward in hierarchy
(..)

Operations on File Directories

• GUI vs commands

- Create/delete
- List: sorting, wild cards, recursion, information shown
- Change (current, working) directory
- Move
- Rename
- Change protection
- Create/delete link (symbolic)
- Find/search routines



```

C:\Users\Lubomir>dir
Volume in drive C is Windows7_OS
Volume Serial Number is CE2C-14BC

Directory of C:\Users\Lubomir

09/24/2010  06:05 PM    <DIR>          .
09/24/2010  06:05 PM    <DIR>          ..
08/04/2010  06:38 AM    <DIR>          Contacts
09/27/2010  05:41 PM    <DIR>          Desktop
08/17/2010  06:53 PM    <DIR>          Documents
09/24/2010  06:04 PM    <DIR>          Downloads
08/04/2010  06:38 AM    <DIR>          Favorites
08/04/2010  06:38 AM    <DIR>          Links
08/04/2010  06:38 AM    <DIR>          Music
09/03/2010  02:13 PM    <DIR>          New folder
09/27/2010  12:41 PM    <DIR>          Pictures
08/04/2010  06:38 AM    <DIR>          Saved Games
08/04/2010  06:38 AM    <DIR>          Searches
08/04/2010  06:38 AM    <DIR>          Videos
               0 File(s)                0 bytes
               14 Dir(s)  260,614,795,264 bytes free

C:\Users\Lubomir>dir D*
Volume in drive C is Windows7_OS
Volume Serial Number is CE2C-14BC

Directory of C:\Users\Lubomir

09/27/2010  05:41 PM    <DIR>          Desktop
08/17/2010  06:53 PM    <DIR>          Documents
09/24/2010  06:04 PM    <DIR>          Downloads
               0 File(s)                0 bytes
               3 Dir(s)  260,614,795,264 bytes free

C:\Users\Lubomir>mkdir test

C:\Users\Lubomir>cd test

C:\Users\Lubomir\test>dir
Volume in drive C is Windows7_OS
Volume Serial Number is CE2C-14BC

Directory of C:\Users\Lubomir\test

09/28/2010  11:35 AM    <DIR>          .
09/28/2010  11:35 AM    <DIR>          ..
               0 File(s)                0 bytes
               2 Dir(s)  260,614,762,496 bytes free

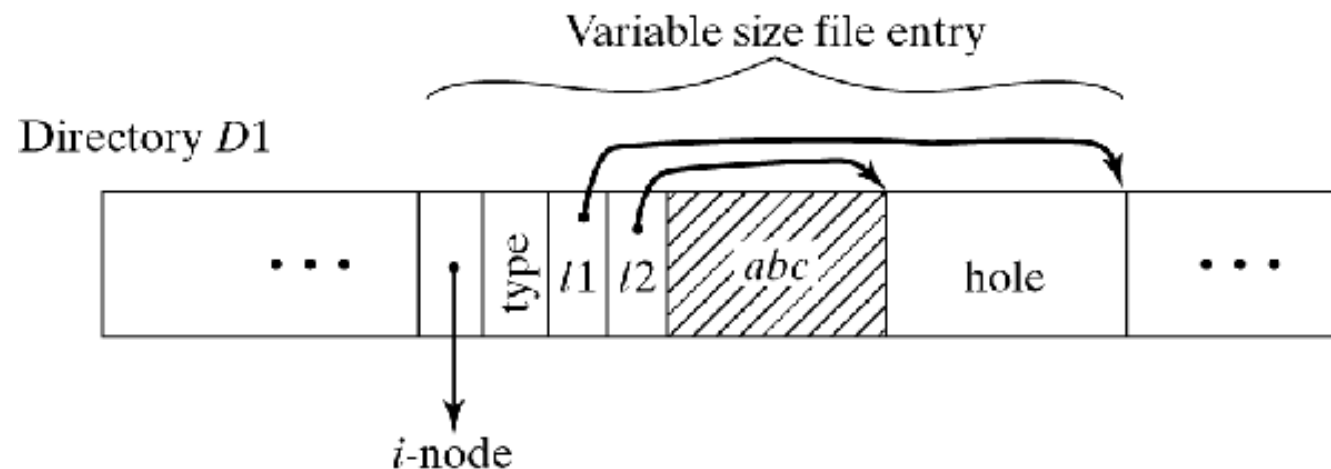
C:\Users\Lubomir\test>cd ..

C:\Users\Lubomir>rmdir test

C:\Users\Lubomir>
  
```

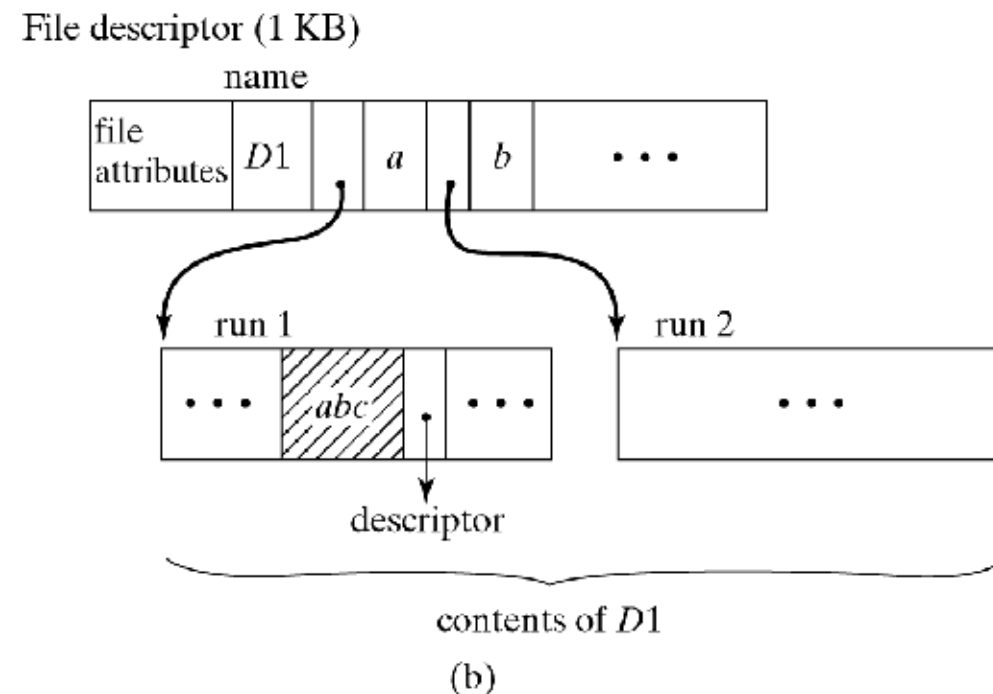
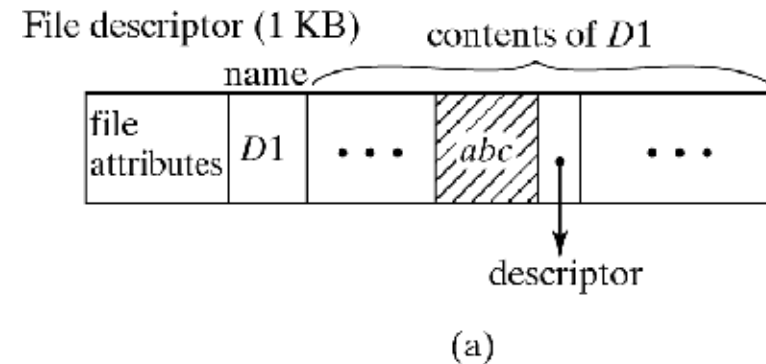
Implementation of Directories

- **What information to keep in each entry**
 - **All descriptive information**
 - ✓ **Directory can become very large**
 - ✓ **Searches are difficult/slow**
 - **Only symbolic name and pointer to descriptor**
 - ✓ **Needs an extra disk access to descriptor**
 - ✓ **Variable name length?**



Implementation of Directories

- **How to organize entries within directory**
 - **Fixed-size entries:**
use array of slots
 - **Variable-size entries:**
use linked list
 - **Size of directory:**
fixed or expanding
- **Example: Windows 2000**
 - when # of entries exceeds directory size,
expand using B⁺-trees



Revisit file operations

- **Assume:**
 - descriptors are in a dedicated area
 - directory entries have name and pointer only
- **create**
 - find free descriptor, enter attributes
 - find free slot in directory, enter name/pointer
- **rename**
 - search directory, change name
- **delete**
 - search directory, free entry, descriptor, and data blocks
- **copy**
 - like create, then find and copy contents of file
- **change protection**
 - search directory, change entry

Basic File System

- **Open/Close files**
 - Retrieve and set up information for **efficient** access:
 - ✓ get file descriptor
 - ✓ manage open file table
- **File descriptor (i-node in Unix)**
 - Owner id
 - File type
 - Protection information
 - Mapping to physical disk blocks
 - Time of creation, last use, last modification
 - Reference counter

Basic File System

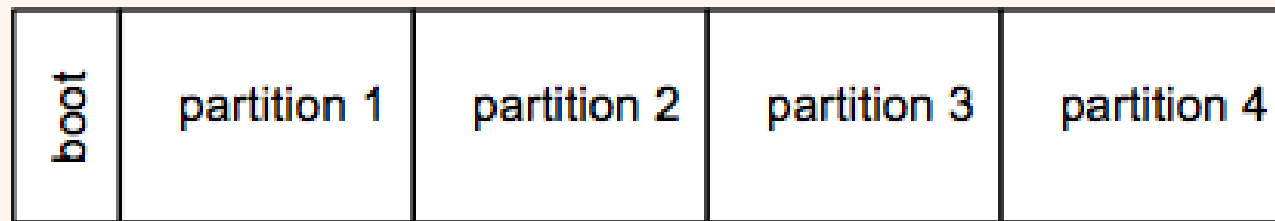
- **Open File Table (OFT) keeps track of currently open files**
- **open command:**
 - Search directory for given file
 - Verify access rights
 - Allocate OFT entry
 - Allocate read/write buffers
 - Fill in OFT entry
 - ✓ Initialization (e.g., current position)
 - ✓ Information from descriptor (e.g. file length, disk location)
 - ✓ Pointers to allocated buffers
 - Return OFT index

Basic File System

- **close command:**
 - Flush modified buffers to disk
 - Release buffers
 - Update file descriptor
 - ✓ file length, disk location, usage information
 - Free OFT entry

Organizing Data on a Disk

Disk Organization



Typical PC hard disk organization

Boot: usually start of disk which contains code to start loading the OS (booting the OS). For PC, boot block also called Master Boot Record (MBR) and contains also the partition table

Partitions: divide up disk into regions for filesystems, each filesystem in one partition. For PCs, only 4 primary partitions are allowed which can be booted

Note: details are PC specific, also can have logical partitions beyond 4 primary partitions, etc.

Organizing Files

- basic unit is a sector/block in a partition (compare with page in VM)
- logical view of file implementation – collection of logical blocks (possibly last block may be partial since file data may be \leq block size)
- logical blocks need not be same as physical sector size – may be some multiple of sectors (Why? consecutive I/O is the fastest)
- Simple schemes – minimum unit is 1 block – file size < 1 block has **internal fragmentation**, more complex schemes can share multiple small files in 1 block (we won't look at this)!

Notes: mostly we treat file as collection of bytes, MSDOS logical block is called cluster (which can be large!)

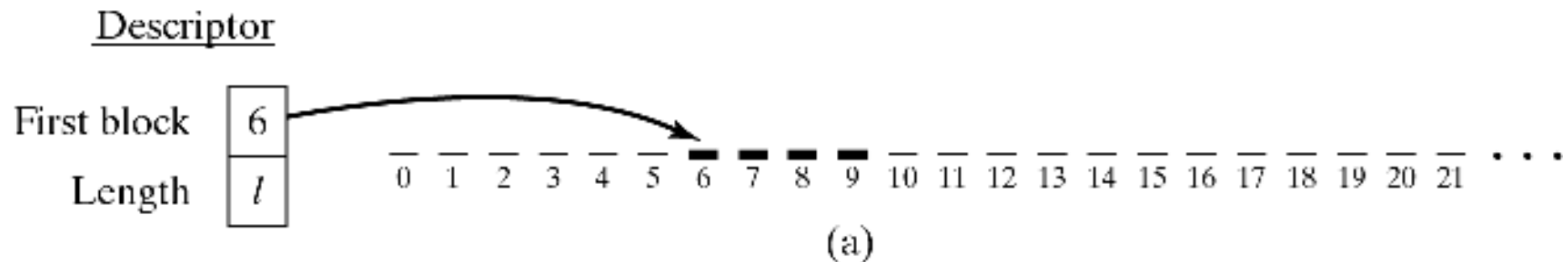
Data Structures on Disk

- need data structure to record which block belongs to which part of file, eg. data at positions 2020-4100 are in which blocks and which part of the block
- How does data in file change?
 - write more data at end of file
 - Unix: write data into holes! But basically means can write anywhere!
 - decreases with truncation operation (unlike mem no free op!)
- data structure must also be stored on disk (persistent)
- typical data structures: versions of list / trees / arrays

Notes: very similar to data structures for dynamic memory management and page tables

Physical Organization Methods

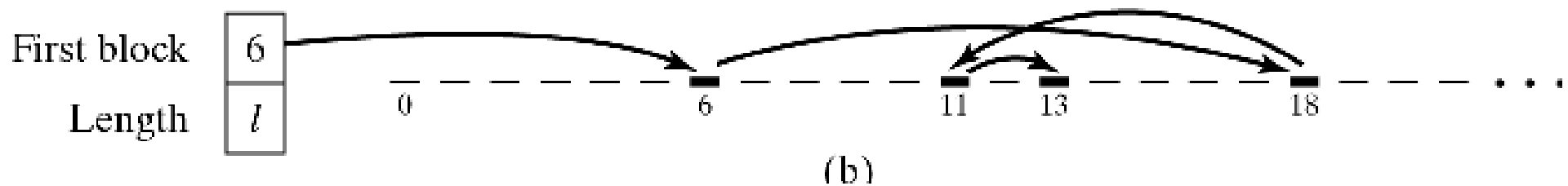
- **Contiguous organization**
 - Simple implementation
 - Fast sequential access (minimal arm movement)
 - Insert/delete is difficult
 - How much space to allocate initially
 - External fragmentation



Physical Organization Methods

- **Linked Organization**

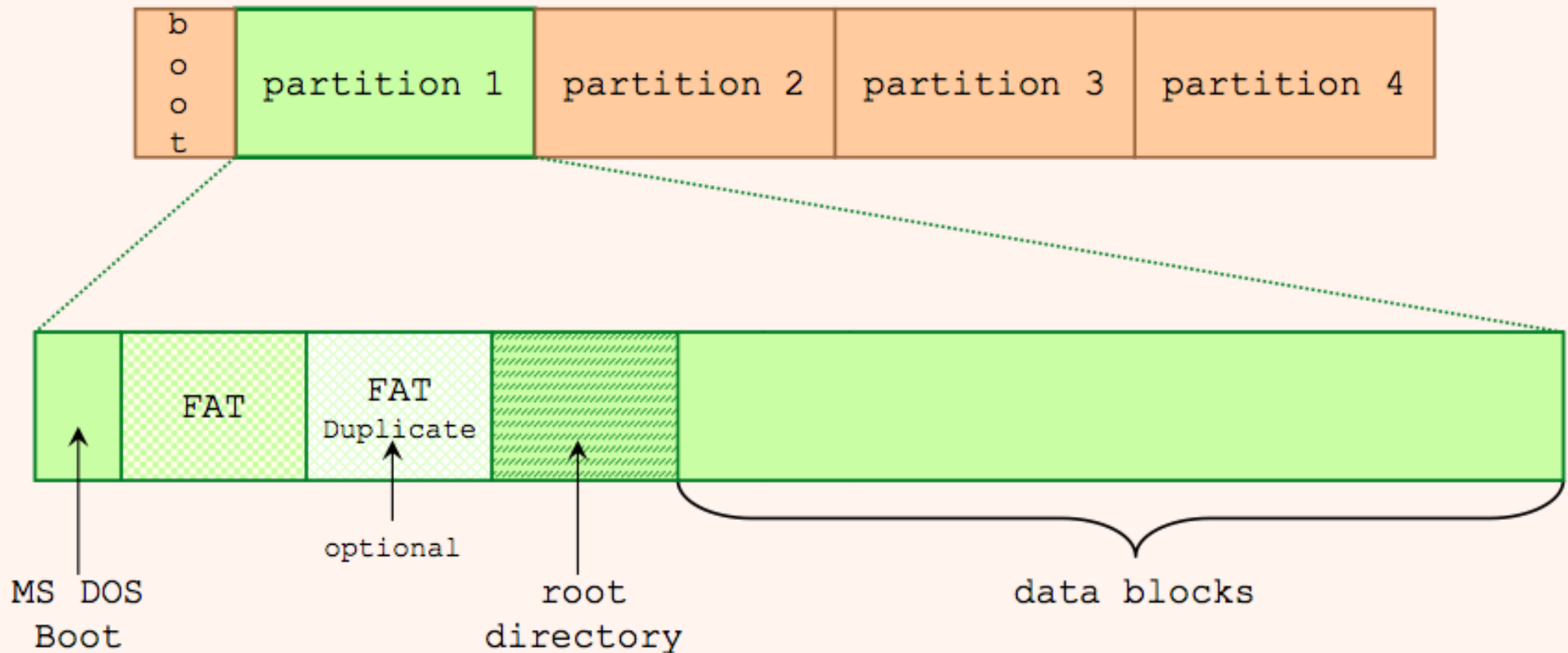
- Simple insert/delete, no external fragmentation
- Sequential access less efficient (seek latency)
- Direct access not possible
- Poor reliability (when chain breaks)



Linked List Allocation - FAT

- MSDOS uses File Allocation Table (FAT)
- Linked allocation but stored completely in FAT (after reading from disk)
- FAT kept in **RAM** (stored in disk but duplicated in RAM) – gives fast access to the pointers
- FAT table contains either: **block number** of next block, **EOF** code (corresponds to NULL pointer), **FREE** code (block is unused), **BAD** block (block is unusable, i.e. disk error) – combines bitmap for free blocks with linked allocation for list of blocks
- FAT table is 1 entry for every block. Space management becomes an array method (but bigger than bitmap)

MSDOS File System Layout

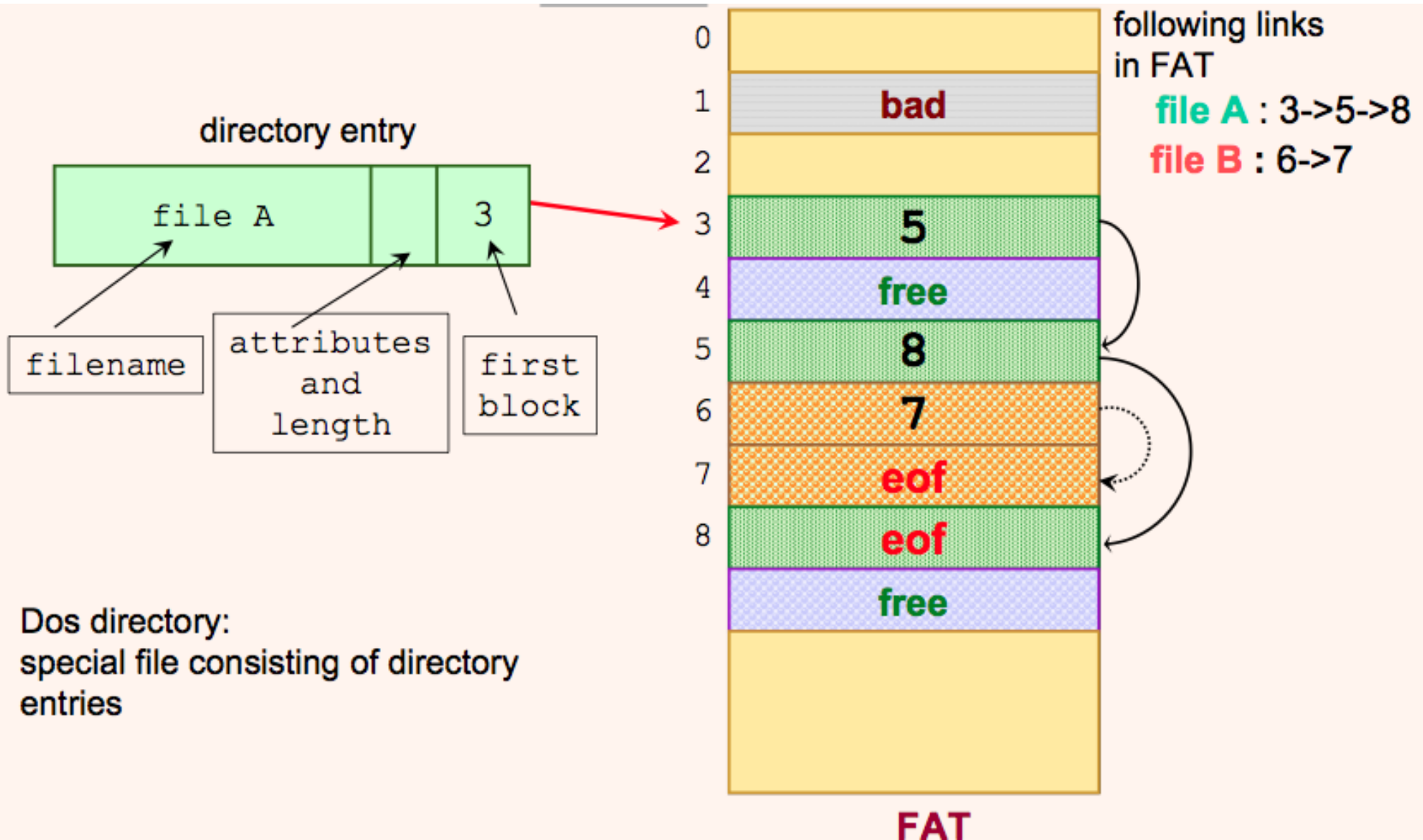


Notes: FS corresponds to 1 volume (logical drive, eg. C:), stored in 1 partition, eg: another drive (D:) in another partition etc. We don't cover extended partition

DOS Directory

- special file (type directory) containing directory entries (32 byte structures, little endian)
- **fat directory entry**: **filename + extension** (8+3 bytes), file **attributes** [1 byte: readonly, hidden, system, **directory flag** (distinguish directories from normal files), archive, volume label (disk volume name is dir entry)], time+date created, last access (read/write) date, time+date last write (creation is write), **first block (cluster) number**
- root directory is special (already known, FAT16 has limited root dir size, 512 entries), other directories distinguished by type

FAT Organization



FAT Organization

- Linked list implemented as array of FAT entries:
4 types: block number (part of a linked list), EOF type (end of linked list), special FREE type, special BAD type – some numbers are block numbers, some are reserved block numbers for EOF, FREE, BAD
- Blocks for file linked together in FAT entries, eg. file A:
3 -> 5 -> 8
- Free blocks indicated by FREE entry
- Bad blocks (unusable blocks due to disk error) marked in FAT entry: BAD cluster value

MSDOS: Deleting a File

How MSDOS deletes file/directory:

- Set first letter in filename to 0xE5 (destroys first byte of original filename)
- Free data blocks: set FAT entries in link list to FREE (tags all the data blocks in file free)

Can attempt to **undelete** – some information lost but can guess!

FAT16 Cluster Size

- FAT16: fat entry block # is **16 bits** (16 bit numbers in FAT entries), sector size=512, how to deal with disks bigger than 64K*512 (32M)
- Logical block size = multiple of sectors. MSDOS calls this the **cluster size**
- Maximum cluster size = 32K (normally)
- Maximum file system size for FAT16 = 64K*32K = 2G
- Maximum file size: slightly less than 2G
- large cluster size means large internal fragmentation!

Note: can use 64K cluster size, so limits become 4G

VFAT

- VFAT: adds long filenames (255 chars, introduced in Win95)
 - compatible with FAT16 by tricking it!
 - short FAT16 filename for FAT16 and long version, short filename created from long one (e.g. **longna~1.doc**)
 - for compatibility: long name stored as multiple directory entries using illegal attributes (not used by FAT16)
 - trick: have to manage 2 kinds of names, old SW uses short names, aliasing issue due to 2 names

FAT32

- Increase FAT size to 28 bits, cluster numbers 28-bits
- max filesystem size increased: Win98 ~127G limit, Win2K can only format up to 32G limit
- **decreases internal fragmentation** (cluster size can decrease)
- **FAT table size increases**
- FAT16 root directory size limit removed – normal directory
- Maximum file size $2^{32} - 1$

Notes: compare with direct paging, changing size of page and effect on page table given different sized virtual addresses

Disk Fragmentation

Fast disk access:

- contiguous blocks (from geometry/processing viewpoint)
- blocks in same cylinder

Suppose currently FS is optimally allocated (fresh install). Is this sustainable?

1. Delete files/blocks
2. Insert new files/blocks

After some operations – block ordering becomes more **random!**

Disk Fragmentation: logical contiguous blocks are “**far apart**” on disk (this is different from memory fragmentation)

Notes: internal fragmentation still exists from block size

Disk Fragmentation

FAT:

- affects FAT FS
- fragmentation effect less with large cluster size (but **large internal fragmentation!**)
- MSDOS solution: run defragmentation (like compaction) on entire FS – move all used blocks to be contiguous .. one big free space chunk after defragmentation. May take a long time to defrag! (Windows: Disk Defragmenter)

Unix S5FS:

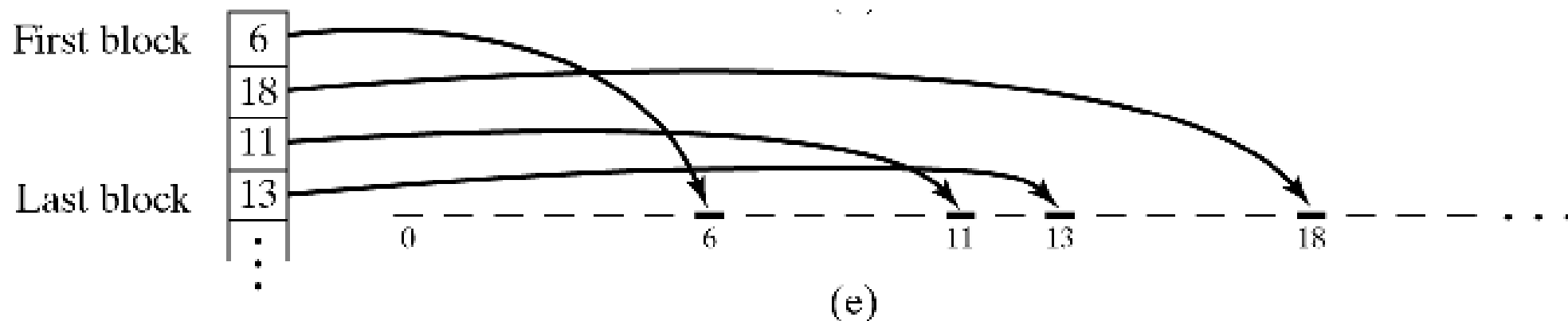
- also has disk fragmentation
- may be worse than DOS since smaller logical block size

Alternative (not covered): FS with **fragmentation resistance** (not necessarily optimal but don't need to defragment).

Physical Organization Methods

- **Indexed Organization**

- Index table: sequential list of records
- Simplest implementation: keep index list in descriptor



- Insert/delete is easy
- Sequential and direct access is efficient
- Drawback: file size limited by number of index entries

Physical Organization Methods

- **Variations of indexing**
 - **Multi-level index hierarchy**
 - ✓ **Primary index points to secondary indices**
 - ✓ **Problem: number of disk accesses increases with depth of hierarchy**
 - **Incremental indexing**
 - ✓ **Fixed number of entries at top-level index**
 - ✓ **When insufficient, allocate additional index levels**
 - ✓ **Example: Unix, 3-level expansion (see next slide)**

UNIX File Systems

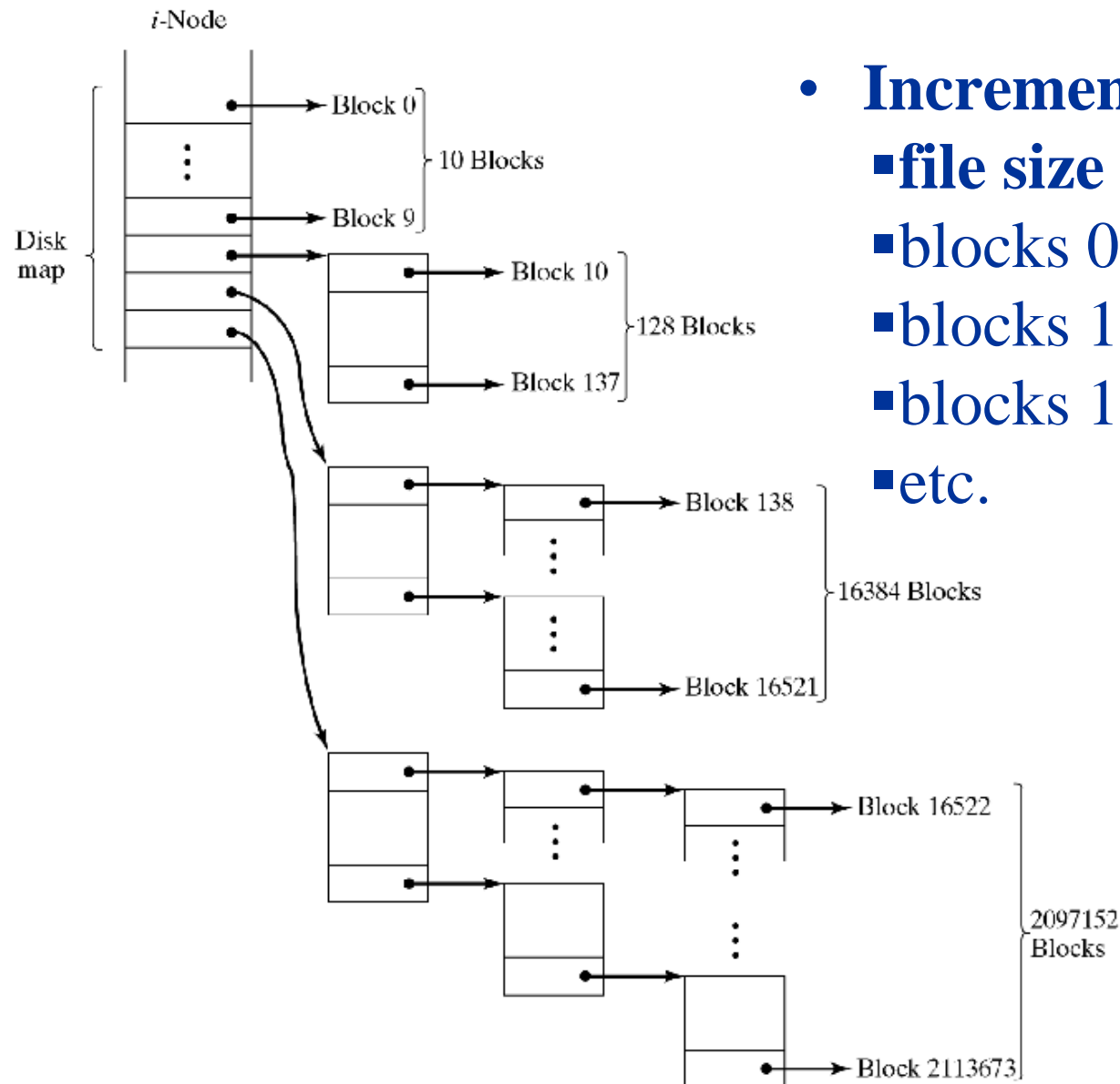
- Look at System V File System (s5fs) – simpler than modern FS implementations (rather old but some of the design remains in current implementations)
- inodes: represents every file
- directories: contain names of files (recall maps filename to eventual file (hard link) or pathname (symbolic link))
- file allocation using a multi-level tree index

Note: s5fs is the original Unix v7 FS. It's much simpler and less sophisticated than more modern Unix FS.

s5fs: Inodes

- actual file object
- every file has one inode (many to one mapping because of hard links)
- contains all meta-data about file **except filename**
- contains reference count i.e. # hard links, reference count = 0 means file can be deleted [subjected to no open files condition - all file descriptors to file object are closed]
- meta-data in inode includes Table of Contents (TOC) which gives mapping of file data to disk blocks (TOC is per file - contrast with MSDOS which has only **global** TOC (FAT))

Unix Table of Contents (TOC)



- **Incremental indexing in Unix**
 - **file size vs. speed of access**
 - **blocks 0-9: 1 access (direct)**
 - **blocks 10-137: 2 accesses**
 - **blocks 138-16521: 3 acc.**
 - **etc.**

s5fs: TOC Index Blocks

- Direct block pointers:
used for small files, no extra disk overhead, efficient direct access. VM analogy: TLB (however not a cache)
 - Single indirect block:
files bigger than direct blocks & smaller than double indirect. Disk overhead is 1 block. File access slightly slower than direct. VM analogy: direct mapped page table
 - double + triple indirect blocks:
files bigger than single indirect block (for sufficiently big block size, triple indirect usually not needed) . More disk overhead but is small fraction of file size. Random file access requires looking up the indirection blocks – slower than indirect. VM analog: 2-3 level page tables
- File sizes: direct << single indirect << double indirect << triple indirect

s5fs: File System Parameters

- # of direct blocks (eg. s5fs: 10, ext2: 12)
- # indirection levels (eg. max is 2 or 3)
- logical block size - determines efficiency of disk I/O, can be composed of several contiguous physical blocks, space wastage from internal fragmentation, determines # block pointers in index blocks and max indirection levels (eg. ext2 can select 1,2,4K when creating filesystem)
- block pointer size - affects indexing, determines max addressable disk block)

s5fs: File System Parameters

- small files only use the direct blocks in TOC, number of direct blocks can vary
- larger file requires first indirect block (this is like 1-level page table)
- even larger file uses double indirect block which points to indirect blocks (similar to 2-level page table). Even larger may have triple indirect (but the number of indirection levels may vary)
- like page tables, can allow blocks which do not exist (**logical zero filled holes** in the file) – set the block pointer to NULL in the TOC, return zeroes for the logical block when read

s5fs: Directories

- a file of type directory – accessing directory is like any other file
- only special directory operations allowed (read directory, link + unlink filenames)
- s5fs directory is array of 16 byte entries (inode: 16 bits, filename: 14 bytes), simplifies directory management (note don't need null pointer if filename length = 14)
- deleted file has inode 0
- note: most modern implementations allow long file names (ext2: 255-byte filenames), so space issues like in memory allocation

Inode Number	Filename
25	.
71	..
100	file1
200	file2

s5fs: Link + Unlink

- Create new file: new directory entry with new inode
- Hard link: new directory entry (in appropriate dir) with inode of the linked file: eg. `link(path1, path2)`
 - `i1` = inode for file with path1;*
 - let path2 = dir2/**file2**; // dont allow linking directories*
 - add new directory entry to dir2: [**i1**, **file2**]; // assume file2*
// not there
- Deleting (deleting is unlink since graph is DAG):
remove directory entry, decrement inode link count,
free file object when link count = 0 (plus open fd's
condition)

Free Storage Space Management

- **Similar to main memory management**
- **Linked list organization**
 - Linking **individual** blocks -- inefficient:
 - ✓ **no block clustering to minimize seek operations**
 - ✓ **groups of blocks are allocated/released one at a time**
 - Better: Link **groups** of consecutive blocks
- **Bit map organization**
 - Analogous to main memory
 - A single bit per block indicates if free or occupied