File Structures

File Structures, Sequential Files, Indexed Files, Inverted Files and Hashed Files

File Structures

- A File Structure is representations of file data o facilitate operations for accessing the data.
- A File Structure allows applications to read, write and modify data.
- It supports effective searching and reading of data in some particular order.

Files

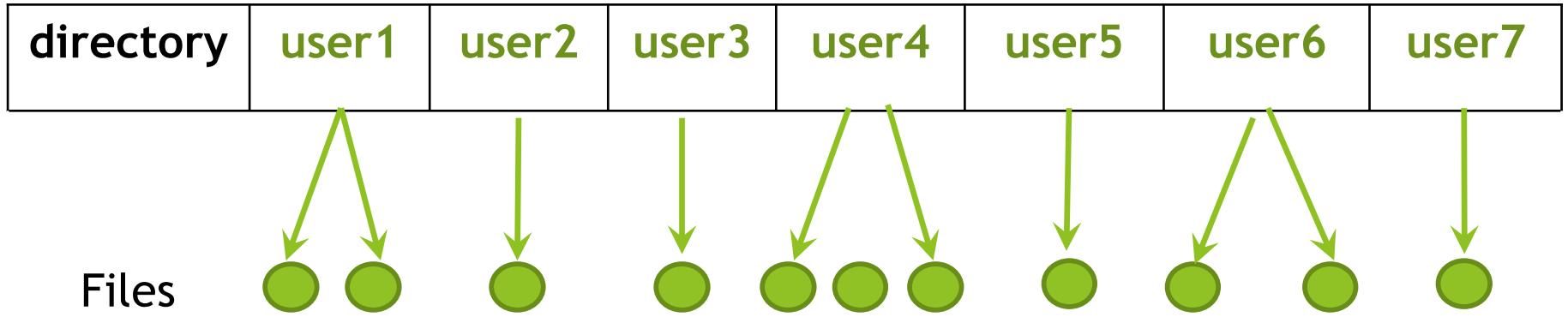
- A file is collection of bytes
- OS needs to ensure that data does not undergo accidental changes while being stored on persistent storage devices such as hard disk, pen drive etc.
- Filename to differentiate between different files,

Directory

- Assists in better organization of files
- Contains information on name and attributes of all files present in the directory. One entry per file.
- A directory can contain other directories. Also . And .. are used for current and parent directories.
- Tree-like directory structure is most common.
- Root of this structure is represented as / (unix based systems) and \ (windows based systems).

Directory Structure

- Single-level directory structure: A single directory for all users (called root directory)

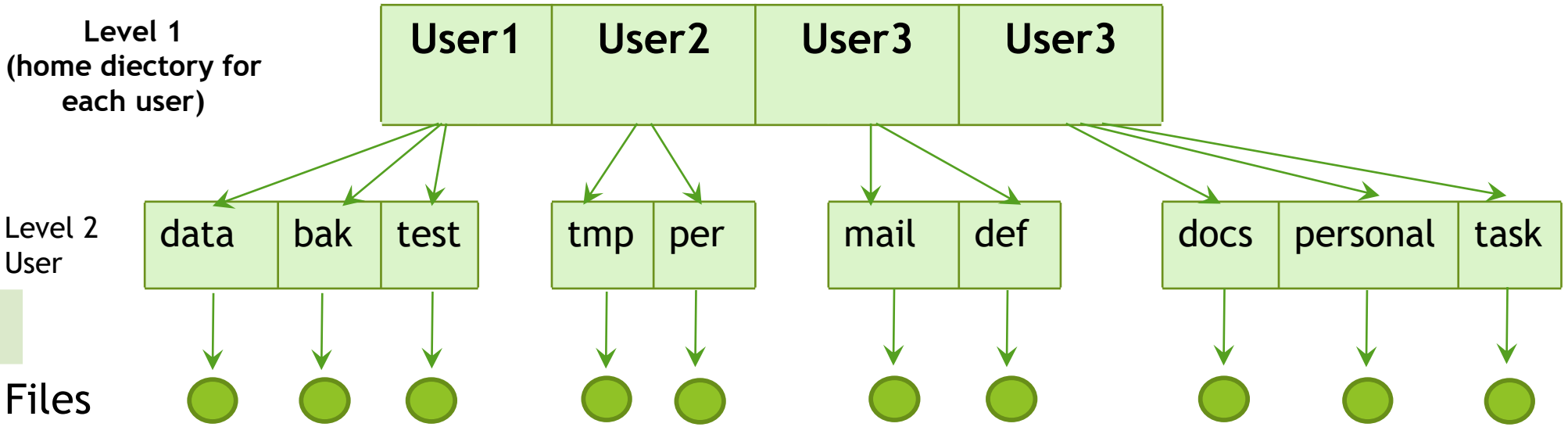


Pros: Simple, easy to quickly locate files

Cons: Inconvenient naming (uniqueness), no grouping

Directory Structure

- Two-level structure Separate directory for each users

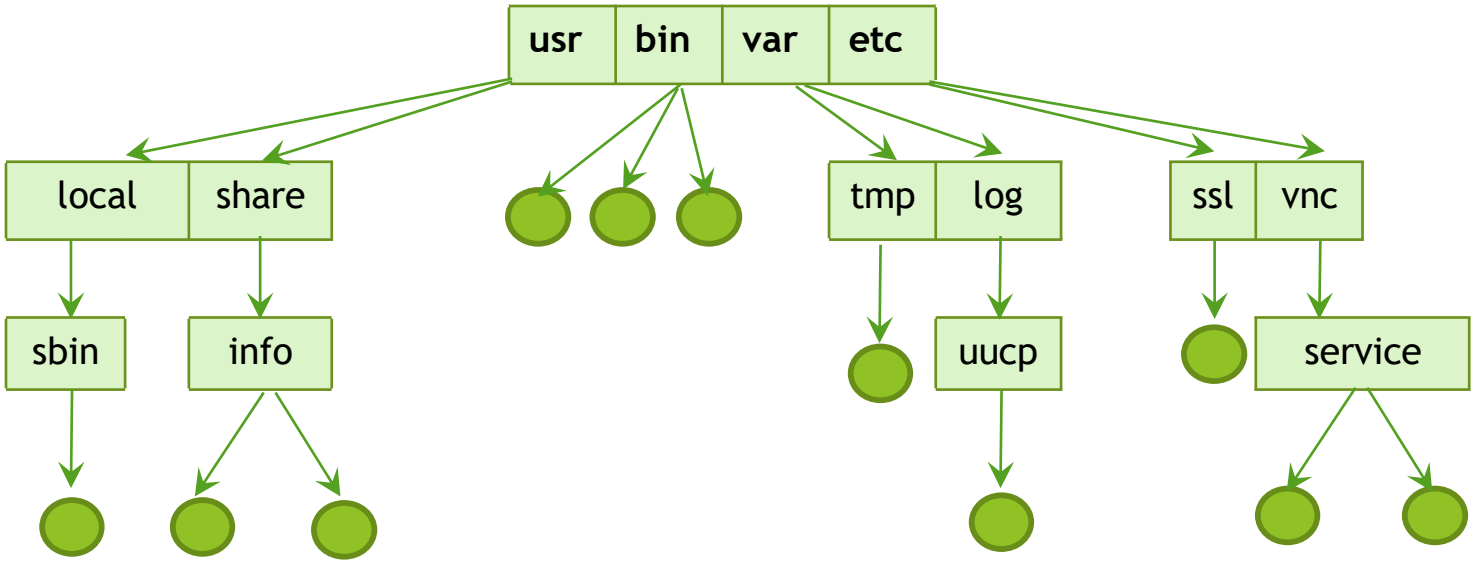


Pros: Have notion of path, can have same file name for different user, efficient searching

Cons: No grouping capability

Directory Structure

- Tree-structured directory Efficient searching & allows grouping



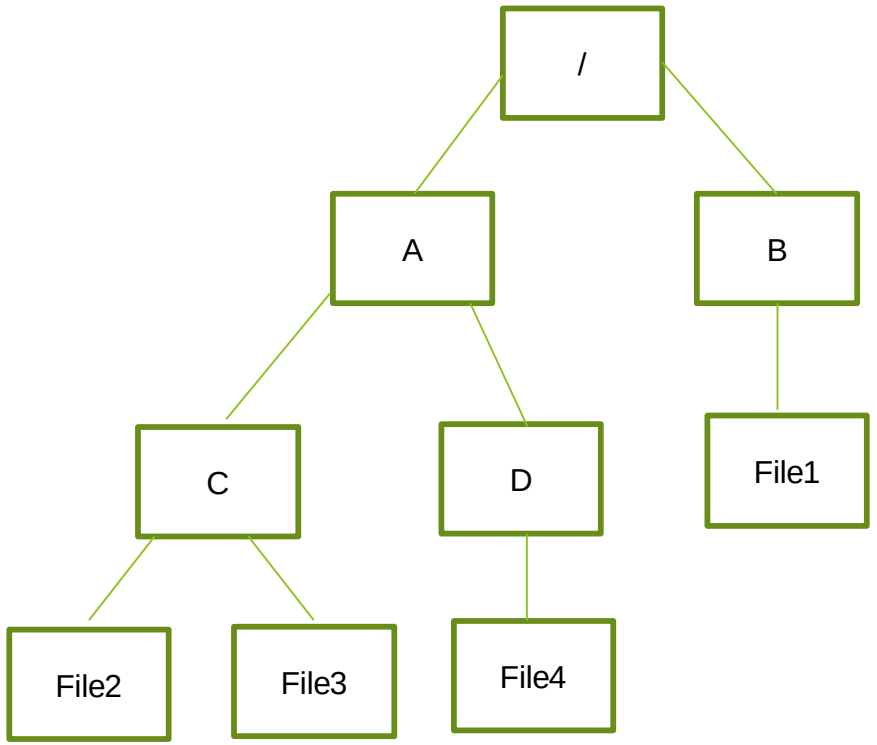
Filename

- / is also used as a separator between directory names in the name of the file.
- Notations . and .. are used for current and parent directory respectively.
- OS resolves the correct path (i.e. directory in which it lies) of a file through
  - Absolute name: with respect to root directory.
  - Relative name : with respect to any node in the directory structure.

Filename contd.

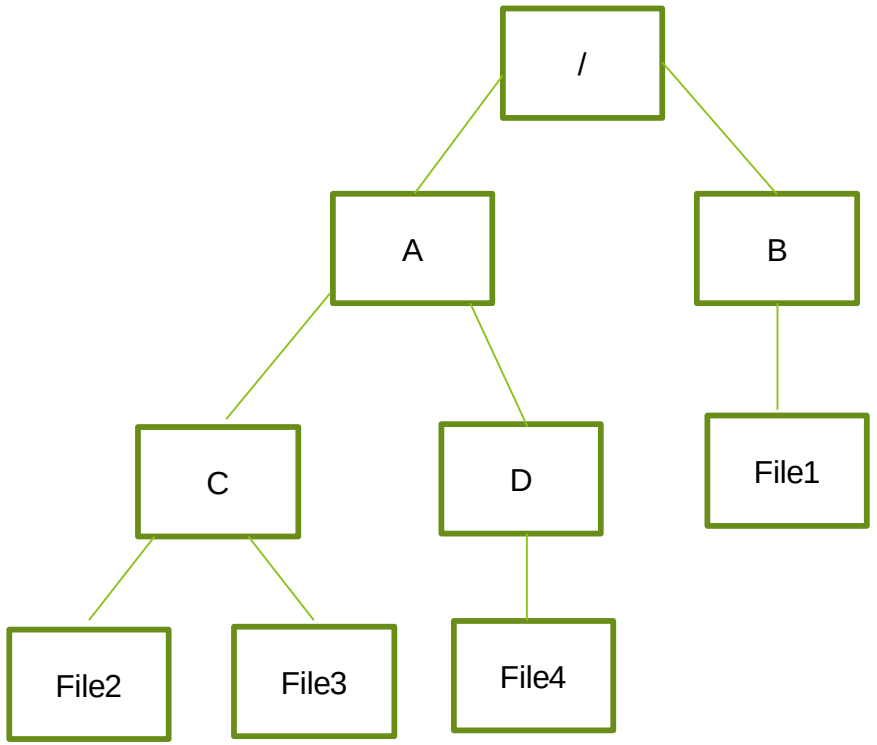
- Consists of two parts
  - base name
  - extension
- Separated by a dot
- An extension specifies the type of file.
- Type is associated with application needed for reading from/writing to file.
- Examples: program.exe, report.pdf, input.txt, etc.

Absolute Filename



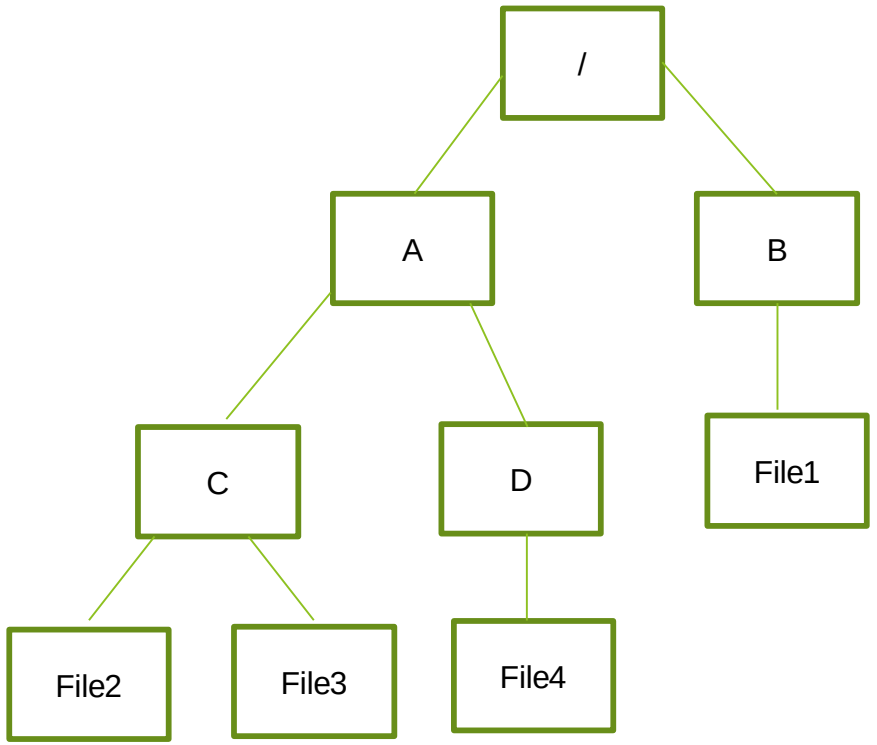
- All four files
  - /B/File1
  - /A/C/File2
  - /A/C/File3
  - /A/D/File4
- And directories
  - /A, /B
  - /A/C
  - /A/D

Relative Filename



- With respect to B
- All four files
  - File1
  - ../A/C/File2
  - ../A/C/File3
  - ../A/D/File4
- And directories
  - . (for B itself)
  - ../A
  - ../A/C
  - ../A/D

Quiz 1: Relative Filename



- Relative names with respect to C (current working directory is C.
- All four files
- And directories

File Attributes

- **Name** – only information kept in human-readable form
- **Owner**
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk

File Operations

- Create a new file
- Open a file: Loads file contents to main memory
- Read from a file; Write to a file
- Rename a file
- Close a file; Delete a file
- Reposition in a file – lseek() system call.
- Change owner/Group of a file
- Change file permissions
- Creation of link of a file

File Operations

- Create a file
  - **fd = open(filename, flags, pmode);**
    - fd = file descriptor
    - filename = physical file name
    - flags = O_CREAT
    - pmode = owner, group, public

File Operations

- Open a file: **fd = open(filename, flags, pmode);**
  - fd = file descriptor
  - filename = physical file name
  - flags = O_APPEND, O_RDONLY, O_RDWR, O_TRUNC, O_WRONLY.
  - pmode = permissions for owner, group, public
- OS keeps open_file table for information of all open files.
- OS identifies a file through an index in this table.
- On closing a file, OS removes its entry from this table.
- create(), delete() work with closed rather than open files.

Filenames contd.

- OS does not rely on extension name.
- Uses first byte (called magic number) of the file to identify a file's type.
- Home assignment: Open different files of different types through a binary/hex editor. What are magic numbers of most popular file types? Check against https://en.wikipedia.org/wiki/List_of_file_signatures

File Attributes

- Home assignment: identify what fields related to a file are stored in a directory entry. Write a small program to read and list a directory entry related to each file.
- What are linux commands related to identifying/modifying (where applicable)
  - Name of the file
  - Type of the file
  - Size of the file
  - Owner of the file
  - Date and time when a file was created
  - Date and time when a file was last accessed/modified
  - Permissions of the file

Open() File

- OS uses two open_file tables per-process table and a system-wide table.
- Per process table keeps information on process's use of the file (i.e current file pointer), access rights and accounting information plus pointer to entry of a system-wide open-file table.
- System-wide table contains information on file size, file location on disk, access dates, plus count field.
- count in system-wide table
  - is incremented when a file is opened by another process,
  - is decremented when a file is closed by another process,
- A file's entry from system-wide is removed only when count becomes zero.

File Operations

- Close a file
  - **Files are usually closed automatically by the operating system.**
- Read from file
  - **Read(Source_file, Destination_addr, Size)**
    - Source_file = location the program reads from, i.e., its logical file name
    - Destination_addr = first address of the memory block where we want to store the data.
    - Size = how much information is being brought in from the file (byte count).

File Operations

- Write to file
  - **Write(Destination_file, Source_addr, Size)**
    - Destination_file = the logical file name where the data will be written.
    - Source_addr = first address of the memory block where the data to be written is stored.
    - Size = the number of bytes to be written.
- Seek into file
  - **Seeking is the action of moving directly to a certain position in a file.**
  - **lseek(Source_file, Offset)**

File operations (contd.)

- Open a file: In linux, open() system call is used for creation of a new file as well as opening an existing file. C library also has a function fopen() that makes use of open() system call. Opening a file loads it into main memory and returns a file descriptor.
- Home Assignments:
  - Identify linux command/system call/ utility for file and directoty operations
  - Map C library function and respective linux sytem calls.

File Permissions

- What access permitted to a user or a group
- Types of access
  - Read
  - Write
  - Execute
- Categories of users
  - Owner
  - Group
  - Others
- 9 bits used for permissions for three categories of users (owner, group, others)
- 3 bits per category - one bit each for read, write, execute

File Permissions (Linux)

- Respective bit is set to allow access
- Respective bit is reset to deny access
- Example: Owner has read, write and execute access; group has read and execute access only; other can only read.

Type	Values	Bit pattern	Octal code
owner	R=1,W=1,X = 1	111	7
group	R=1,W=0,X = 1	101	5
others	R=1,W=0,X = 0	100	4

- In case of directory, X stands for search.
- To set above permissions chmod 754 <name of file>

File Permissions (contd.)

- Home assignment: what shall be chmod command for setting following permissions
  - owner has read and execute access; group has read access only; other have no access.
  - owner and group have all permissions; other have read and execute permissions only.
  - Current permissions are: owner has read, write and execute access; group has read and execute access only; other can only read. How to drop write permission for owner also.

File Permissions (contd.)

- **Permission bits**
  - Pro: Compact enough to fit in just a few bytes
  - Con: Coarse granularity of control
- **Access Control Matrix: Stores what type of access is permitted for a user - One row per user (or group) and one column per object**
- **Storage requirement is prohibitory. Implemented as**
  - **Access Control List**: This is a per object list that tells users that can access file [Column of Access Control Matrix]
  - **Capability:** Stores access permissions allowed to a user [Row of Access Control Matrix]
  - **Shall be discussing these in OS security**

File Structures

- Types of Data Storage
  - Primary Storage (Computer Memory) (Limited)
  - Secondary Storage (Disk/Tape and CD-ROM accessible to the computer)
- Secondary storage can pack huge amount of data, but their access is too slow.
- Eg. A slow RAM takes 120 nanoseconds whereas a disk takes 30 milliseconds.
- By improving the file structure, secondary storage access time can be improved.

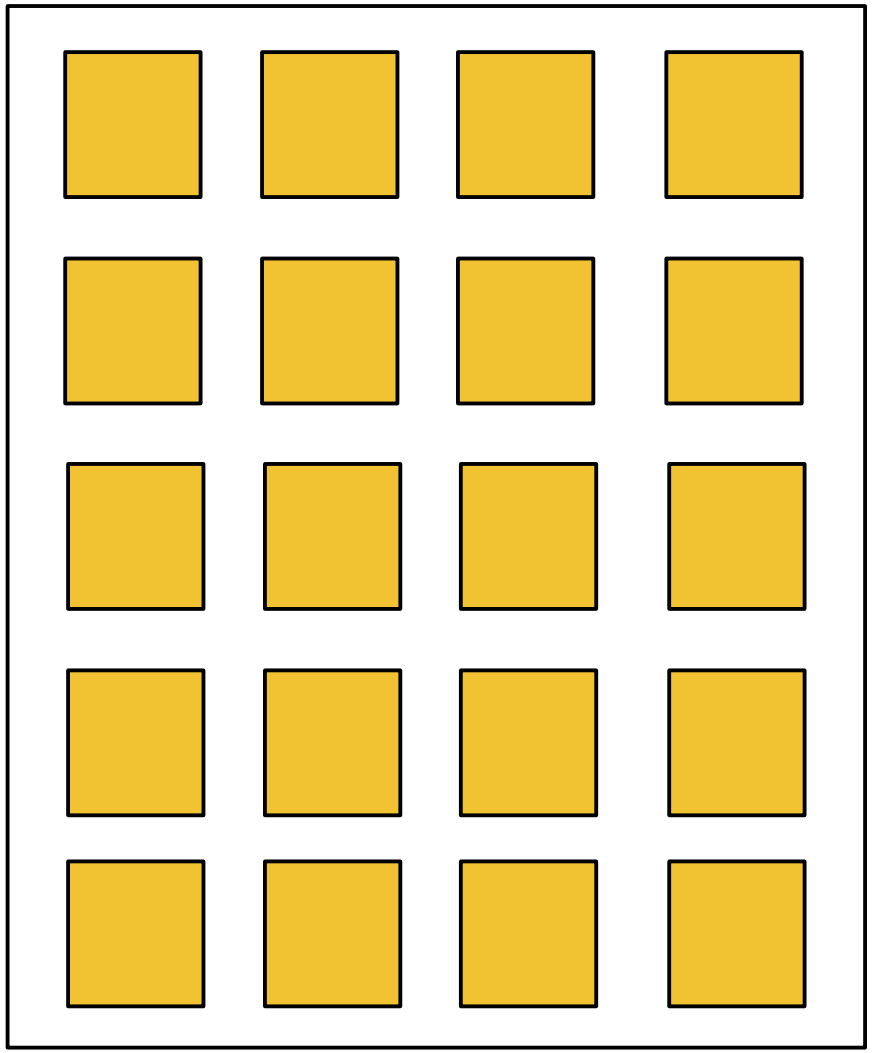
File Structures (contd.)

- A file is collection of bytes.
- The smallest unit of storage on a hard disk (the most common device used for storing a file) is a sector.
- Most OS, however, employ block as a basic unit of storage for improving effective data transfer rate.
- A block is a multiple of sectors.
- In short, a file (and for that matter even a directory) is stored as a collection of blocks on the hard disk.
- Different OS employ different allocation strategies i.e. the way blocks are allocated to a file.

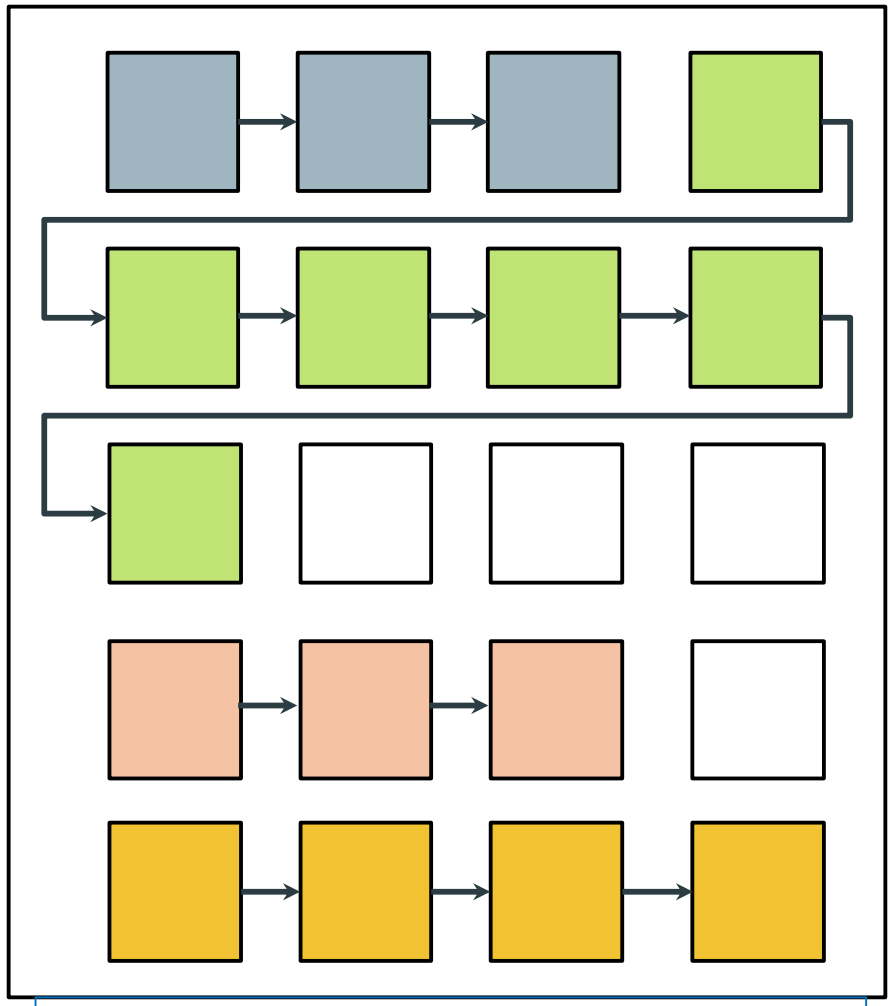
File Organization

- Types of file allocation strategies
  - Contiguous
  - Linked
  - Indexed
  - Inode (linux)

Disk representation: Logical



Contiguous Allocation



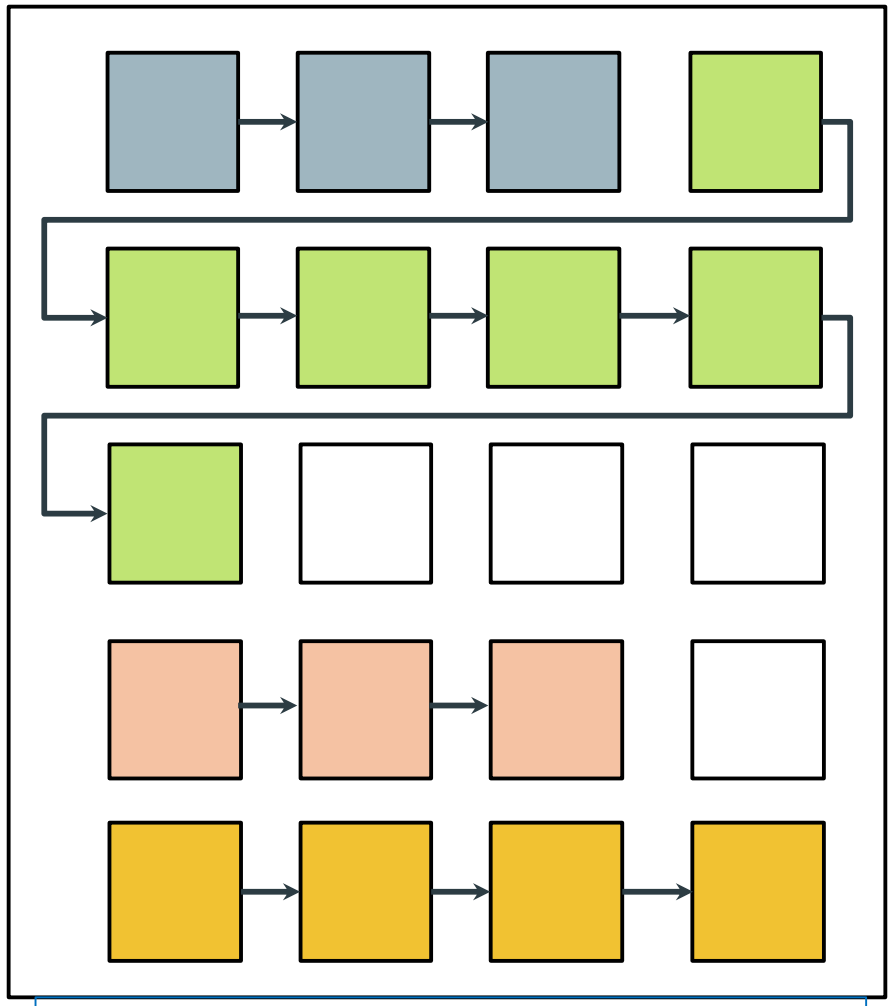
Same color indicates blocks allocated to a file.
White blocks are unallocated.

- All file data blocks are contiguous.
- Knowing address of first block (from directory), address of any block can be accessed in constant amount of time.
- Accessing any part of the file requires same amount of time (random access)
- one disk access (provided that part of the file is not already loaded in main memory).

Contiguous Allocation - Limitations

- A file can be modified only if the blocks next to the file are not allocated to another file. Otherwise the file needs to be move to another location. This, however, is an expensive solution as file need to be copied block-by-block to new location.
- Some OS allow user to specify maximum size requirement of the file. But this may also lead to wastage of space as many users shall try to specify maximum possible size.
- Creation and deletion of files can result in small free spaces interspersed between allocated blocks. External fragmentation of this type can lead to wastage of space.

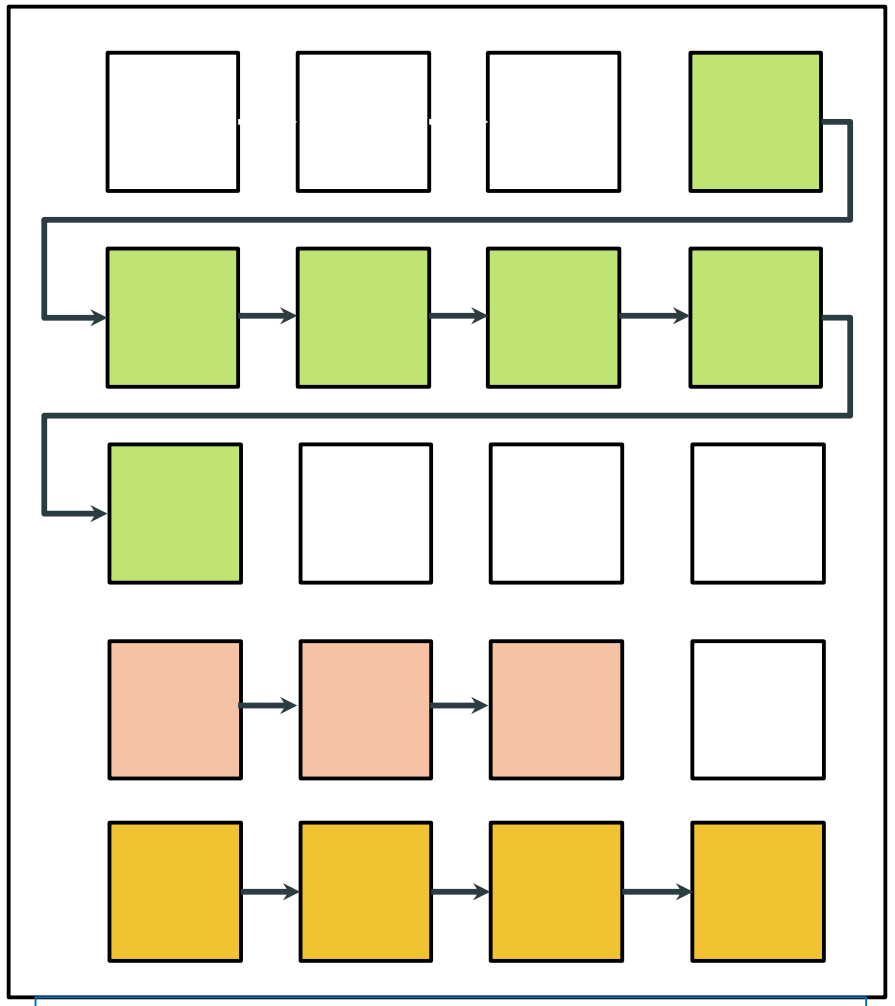
Contiguous Allocation: Fragmentation



Same color indicates blocks allocated to a file.
White blocks are unallocated.

- Blue File deleted

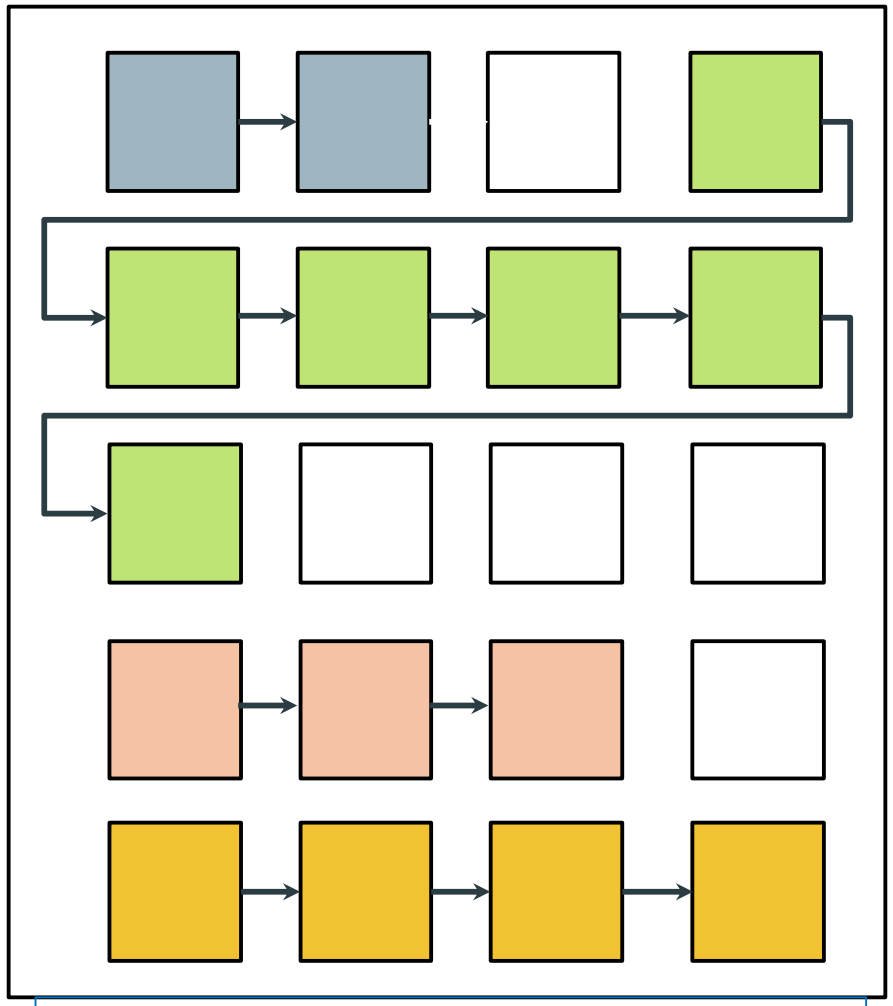
Contiguous Allocation: Fragmentation



Same color indicates blocks allocated to a file.
White blocks are unallocated.

- Blue File deleted

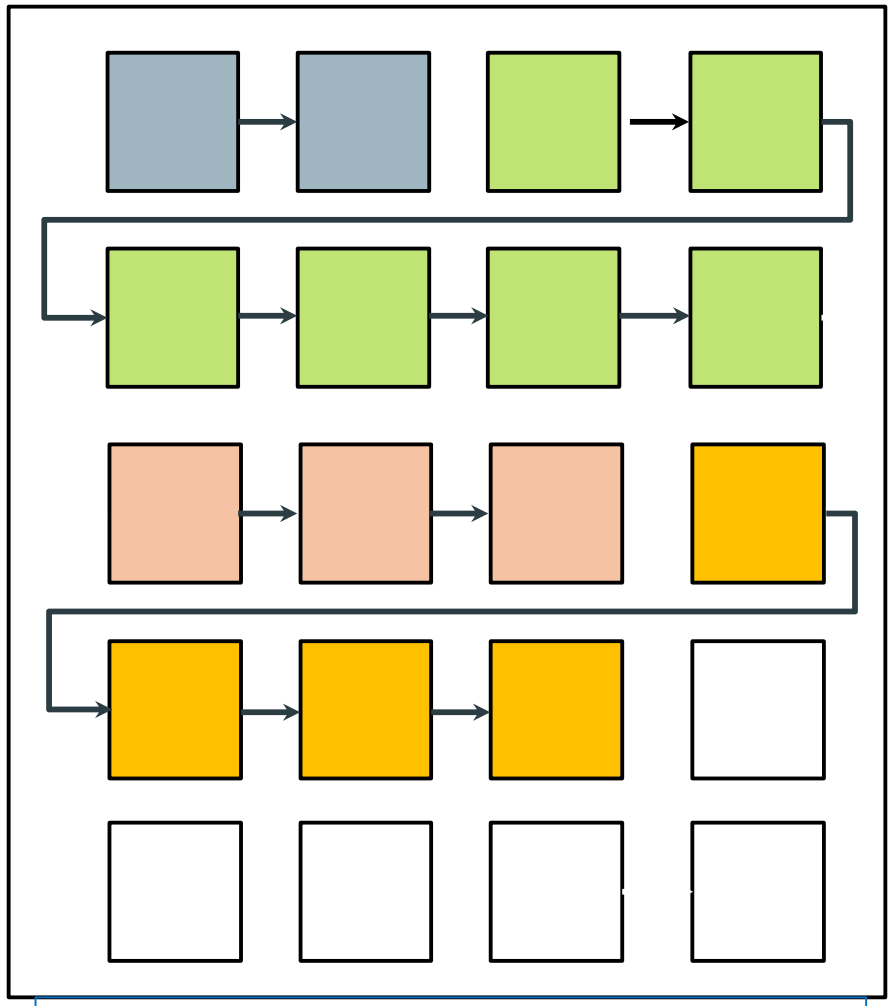
Contiguous Allocation: Fragmentation



Same color indicates blocks allocated to a file.
White blocks are unallocated.

- New file created.
- One block left unallocated.
- May not be useful to store new file.
- External fragmementation.

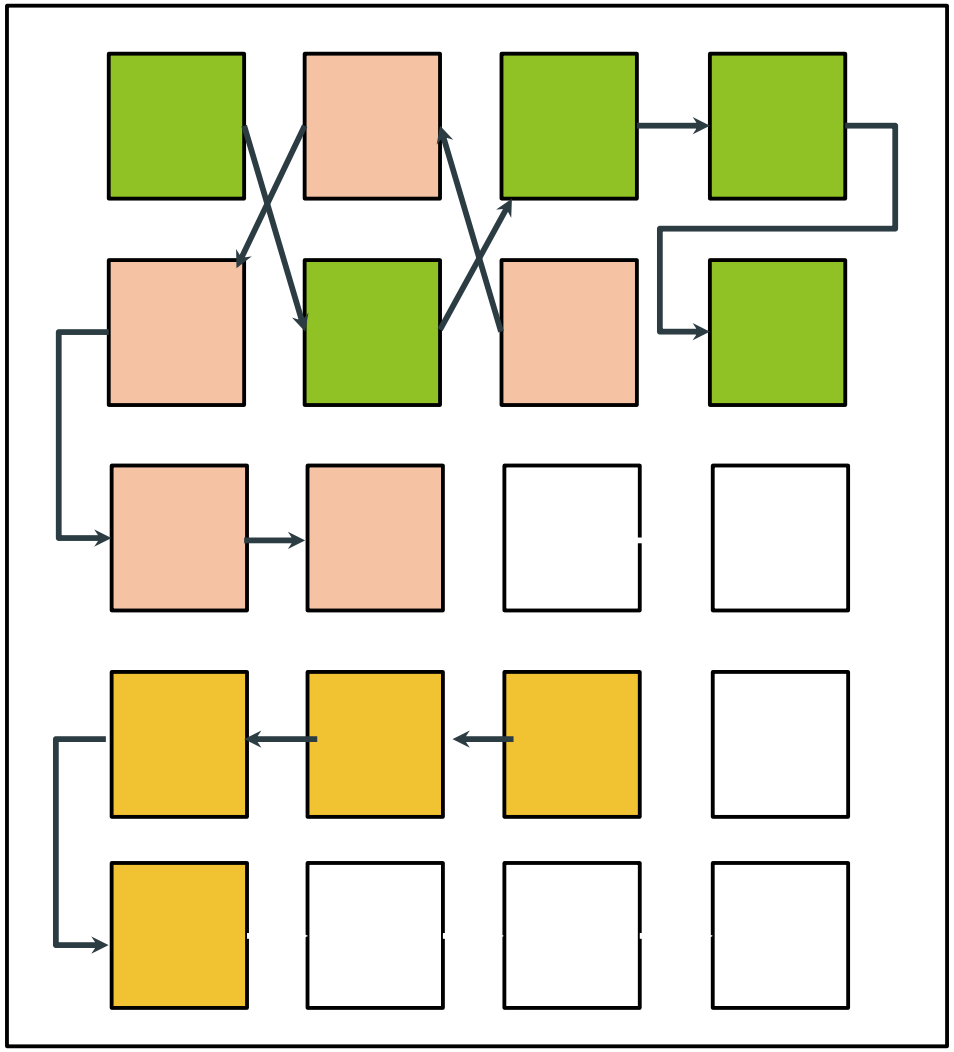
Defragmentation/Compaction



Same color indicates blocks allocated to a file.
White blocks are unallocated.

- Deletion of existing files and creation of new files lead to External fragmentation.
- Solution is defragmentation (also called disk compaction), which moves all small free spaces towards one end by moving files.
- Compaction/defragmentation effectively compacts all free space into one.
- Time consuming activity and may take hours on a disk.

Linked Allocation



- Each file block has address of the next block.
- No external fragmentation and disk can be utilized to last block.
- Access becomes sequential i.e n^{th} block can be accessed only when all previous (n-1) blocks have been accessed.
- Storing a pointer per block also consumes some space.

Linked File Organization

- Links are used to connect previous records to the next records.
- Pros
  - Easy insertion and deletion
  - Sequential processing
- Cons
  - Random accesses slow down file processing.
  - Unreliable
    - **A corrupted pointer can lead to loss of the remaining file**

Indexed File Organization

- In this allocation strategy, an index block is used to hold addresses of the data blocks of the file.
- To access any data block of the file, its address can be accessed from index block.
- As addresses are stored sequentially in the index block, accessing any disk block shall require same amount of time.
- If index block is in memory, only one disk access is needed to access any data block.

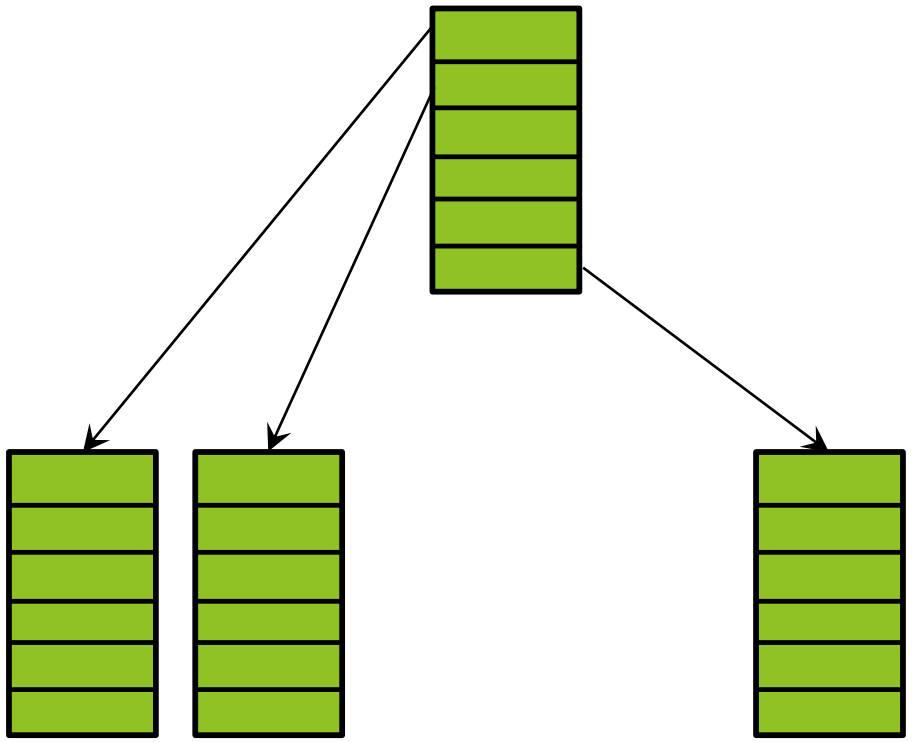
Indexed File Organization

- Limits the size of the file.
- One block = 1024 bytes and one disk address is 8 byte.
- Number of data blocks (that index block can hold) = 1024/8 = 128
- Max. File size = 128KB

Indexed File Organization

- Use multiple blocks as index blocks.
- Index blocks need to be contiguous for random access.
- Two accesses required per data block
  - First index block
  - Data block
- Max. File size = $n.B.\text{floor}(B/p)$
  - Number of index blocks n
  - Block size B (bytes)
  - Pointer size p (bytes)

Indexed File Organization



- To support large file size, index blocks can be organized as a tree.
- For 2-level tree hierarchy; maximum file size shall be
  - $B.x^2$ where
  - x = floor(B/p)
  - B = block size in bytes
  - And p = pointer size in bytes

File organization: Linux

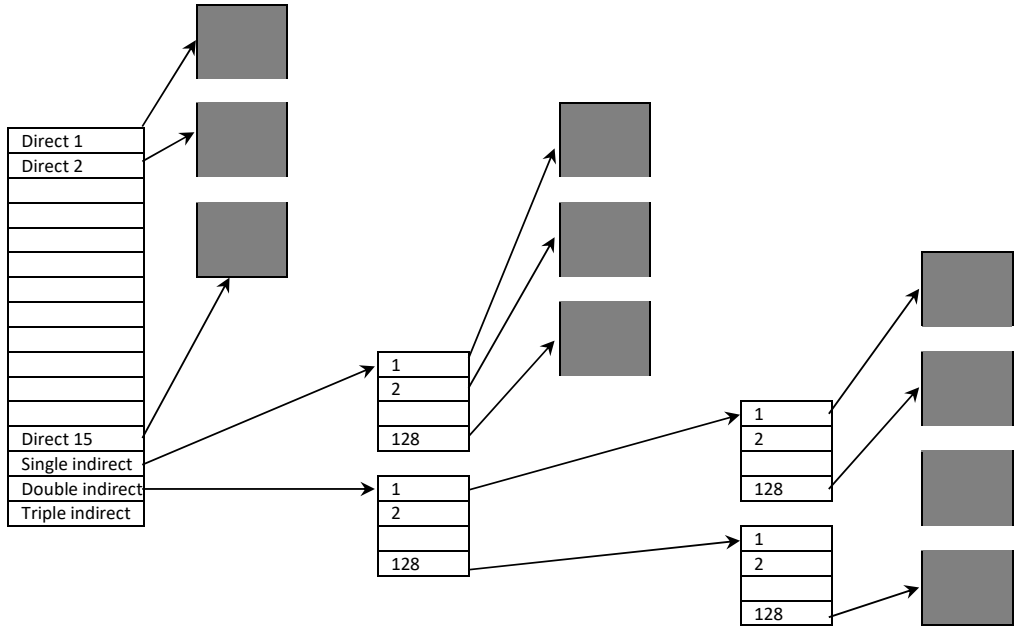
- Unix based systems employ indexed allocation. An i-block is used to store pointers to blocks.
- To ensure support of large file sizes, an i-block has
  - n (n ranges from 13-16) direct pointers,
  - one single indirect pointer,
  - one double indirect pointer and
  - one triple indirect pointer.
- Direct pointers ensure that smaller files can be accessed faster (random access).
- Each direct pointer points to exactly one data block

File organization: Linux

- Each direct pointer points to exactly one data block.
- Each single indirect pointer points to one block of pointers (each pointer pointing to a data block).
- Each double indirect pointer points to one block of pointers; each pointer of this block pointing to another block of pointers, each of which points to a data block).
- Each triple indirect pointer points to one block of pointers. Each pointer of this block points to second level block of pointers. Each pointer of second level block points to third level block. Each pointer of third block points to a data block. (not shown in diagram)

Linux File Structure

Shaded blocks are the data blocks of the file.



File Size Calculation

Size of a file stored in a linux system:

Block size = 1 KB = 1024 bytes,

64-bit disk address i.e. pointer = 8 bytes

Number of pointers per block = 1024/8 = 128

Number of direct pointers = 13

No. of data blocks per direct pointer = 1

No. of data blocks per single indirect pointer = 128 = 2^7

No. of data block per double indirect pointer = 128 x 128 = 2^{14}

No. of data block per triple indirect pointer = 128 x 128 x 128 = 2^{21}

Size of file = $(13 + 2^7 + 2^{14} + 2^{21})$ x 1024 bytes

= 13 KB + 128 KB + 16 MB + 2 GB

Linux File Organization

- Home Assignment: Compute maximum file size that can be stored in a linux system where 15 direct, 1 single indirect, 1 double indirect and 1 triple indirect pointers are stored per i-node. Block size = 512 bytes and disk has 32-bit address.
- Home assignment: How is stat command used to gain information on i-node of a file?

Thank you.

Next Topic: