University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 23: File System 3

University of Nottingham, Ningbo China
2024

Goals for Today

Overview

- **File implementations**
  - Contiguous
  - Linked lists
  - File Allocation Table (FAT)
  - I-nodes
- **Directories implementation**
  - I-nodes lookup
- **Hard and soft links**

File access
Sequential vs. Random Access

- Files will be composed of a number of blocks.
- Files are **sequential** or **random access**. Random access is essential for example in database systems.

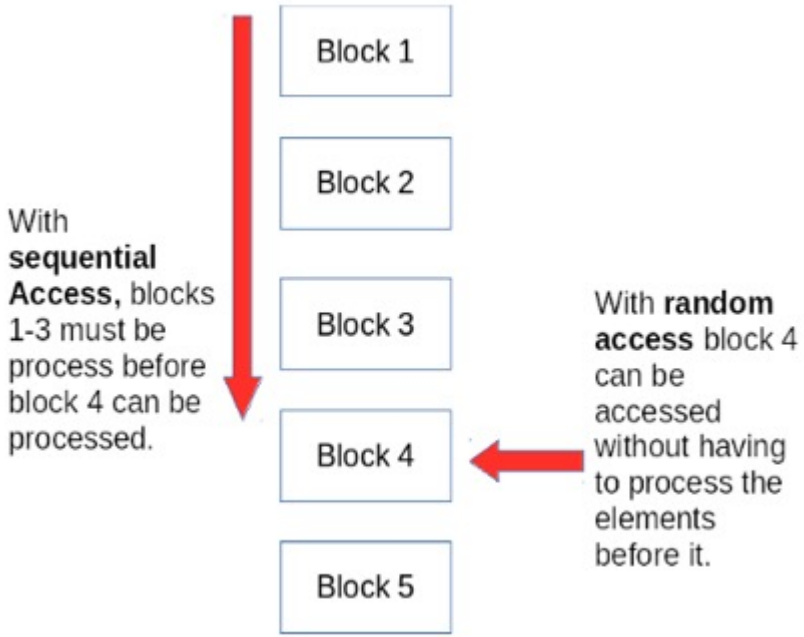


Figure: Types of access to a file

Contiguous Allocation
Concept

- **Contiguous file systems** are similar to **dynamic partitioning** in memory allocation:
- Each file is stored in a single group of **adjacent blocks** on the hard disk E.g. 1KB blocks, 100KB file, we need 100 contiguous blocks
- Allocation of free space can be done using **first fit**, **best fit**, **next fit**, etc.

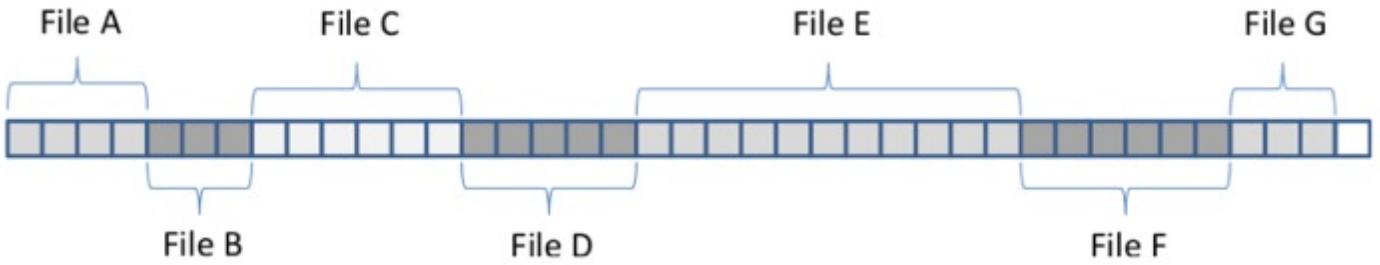


Figure: External Fragmentation when Removing Files

Contiguous Allocation
Advantages

- **Simple to implement**: only location of the **first block** and the **length** of the file must be stored (in the directory entry)

- **Optimal read/write performance**: blocks are co-located/clustered in nearby/adjacent sectors, hence the seek time is minimised (remember the example in lecture on disks!)

File	Start	Length
req1.c	0	12
req1.o	30	10
req1	15	5
req1.txt	41	20

Figure: Directory table

Contiguous Allocation
Disadvantages

- Disadvantages of contiguous file systems include:
  - The **exact size** of a file (process) is not always **known beforehand**: what if the file size exceeds the initially allocated disk space
  - **Allocation algorithms** needed to decide which free blocks to allocate to a given file (e.g., first fit, best fit, etc.)
  - Deleting a file results in **external fragmentation**: de-fragmentation must be carried out regularly (and is slower than for memory)
  - If a single block in a contiguous file becomes **corrupted**, the entire file can become **unreadable**, as all data is stored in a continuous block.
- Contiguous allocation is still in use: **CD-ROMS/DVDs**
  - External fragmentation is less of an issue here since they are write once only

Linked Lists
Concept

- To **avoid external fragmentation**, files are stored in **separate blocks** (similar to paging) that are **linked to one another**
- Only the **address of the first** block has to be stored to locate a file
- Each block contains a **data pointer** to the next block (which takes up space)

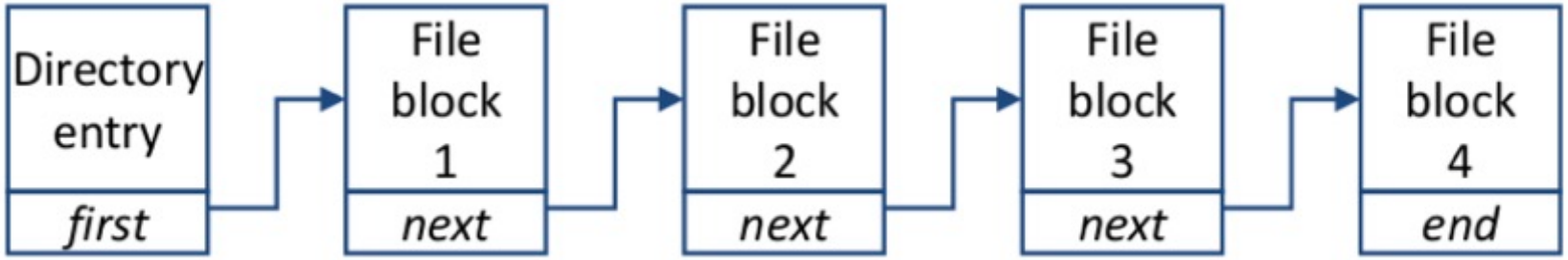


Figure: Linked List File Storage

Linked Lists
Advantages

- **Easy to maintain**: only the first block (address) has to be maintained in the **directory entry**
- File sizes can **grow dynamically** (i.e. file size does not have to be known beforehand): new blocks/sectors can be added to the end of the file
- Similar to paging for memory, every possible block/sector of disk space can be used: i.e., there is **no external fragmentation**!
- **Sequential access** is straightforward, although **more seek operations/disk access** may be required

Linked Lists
Disadvantages

- **Random access is very slow**, to retrieve a block in the middle, one has to walk through the list from the start
- Space is lost within the blocks due to the pointer, the data in a **block is no longer a power of 2**!
- **Diminished reliability**: if one block is corrupt/lost, access to the rest of the file is lost

File Allocation Tables
Key Concept

- Store the linked-list pointers in a **separate index table**, called a **File Allocation Table (FAT)**, in memory!

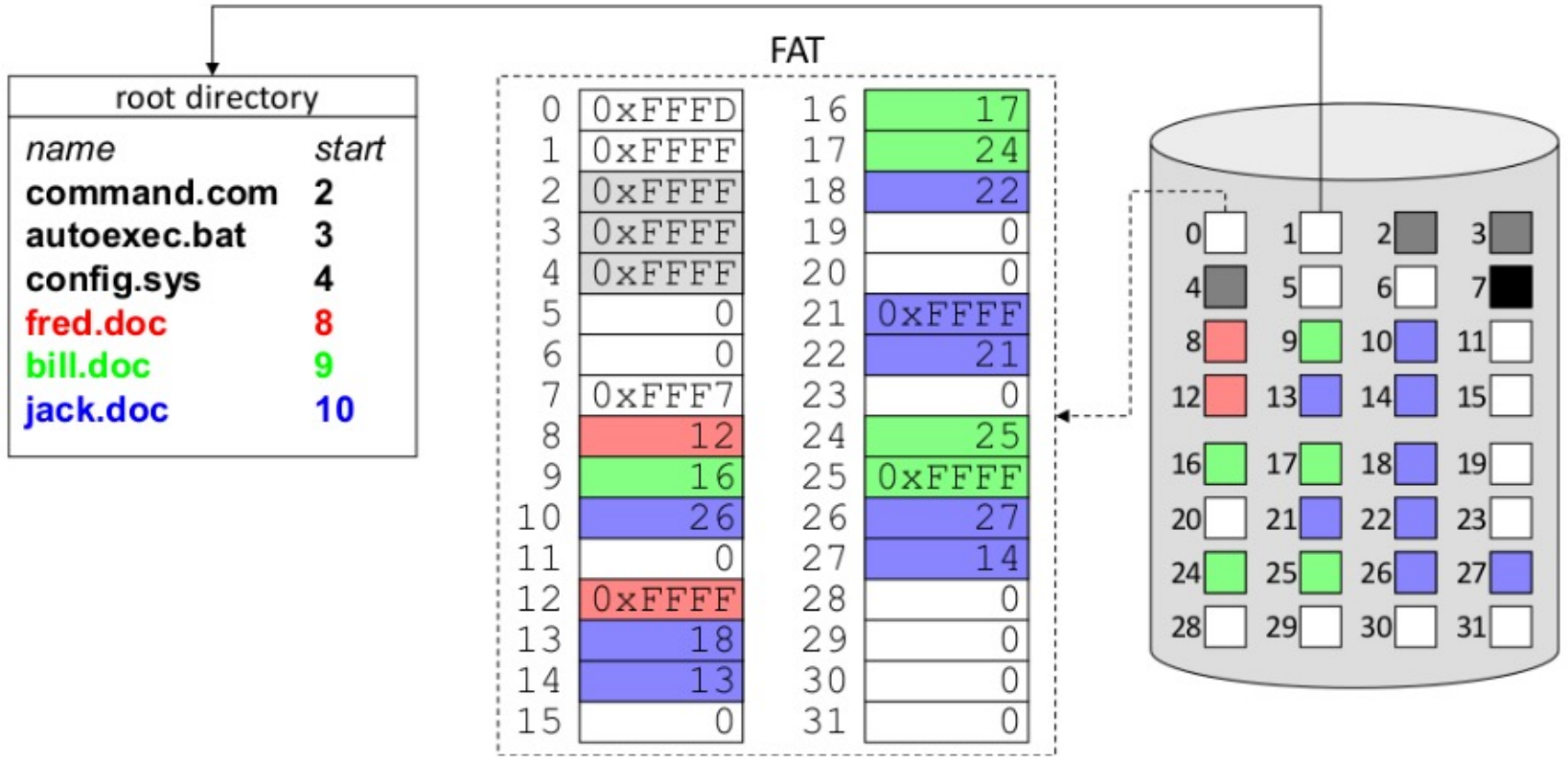


Figure: File Allocation Tables

File Allocation Tables
Advantages and disadvantages

- Advantages:
  - **Block size remains power of 2**, i.e., no space is lost due to the pointer
  - **Index table** can be **kept in memory** allowing fast non-sequential/random access (one still has to walk through the table though)
- Disadvantages:
  - The **size** of the file **allocation table grows** with the number of blocks, and hence the size of the disk
  - For a 200GB disk, with a 1KB block size, 200 million entries are required. Assuming that each entry at the table occupies 4 bytes, this requires **800Mb of main memory!**

I-nodes
Concept

- **Each file** has a small data structure (on disk) called **I-node** (index-node) that contains its **attributes** and **block pointers**.
  - In contrast to FAT, an I-node is **only loaded when the file is open** (stored in system wide open file table)
  - If every I-node consists of *n* bytes, and at most *k* files can be open at any point in time, at most $n \times k$ bytes of main memory are required

- I-nodes are composed of **direct block pointers** (usually 10), **indirect block pointers**, or a combination thereof (e.g., similar to **multi-level page tables**)

I-nodes
Concept

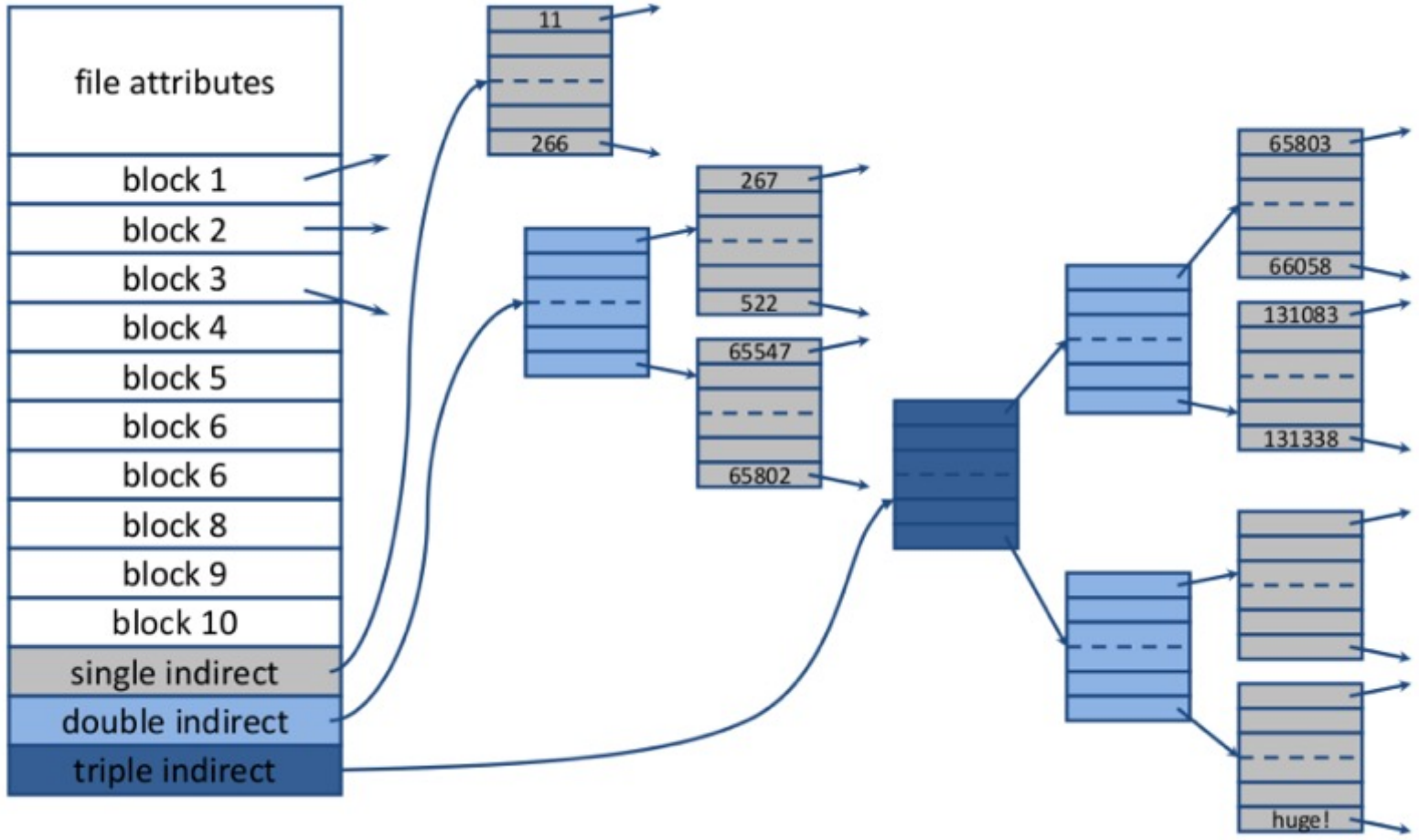


Figure: File Storage Using I-Nodes

I-nodes
Concept

- Assuming a block size of **1KB**, and **32-bits disk address space**. With only direct block pointers, the maximum size of file could be 1KB×number of direct blocks (e.g 10, a total of 10KB).
- A single indirect block can store up to **256 block pointers** (32 bits = 4 bytes per pointer; $2^{10}/2^2 = 2^8 = 256$ pointers). That is, with 10 direct blocks + 1 indirect block, we can have files with up to 266 blocks (⇒ 266KBs).
- A double indirect points to a block of 256 block pointers. Each of which points to 256 indirect blocks. Therefore, with the additional one double indirect block, we could have files with size up to 266KBs + (256 ×256 = 65536) ⇒ 65802 KBs.
- If we need files larger than 64M, we will need **a triple indirect**.

Directories
Possible Implementations

- Directories contain a list of **human readable file names** that are mapped onto **unique identifiers** and **disk locations**
  - They provide a **mapping** of the **logical file** onto the **physical location**

- Retrieving a file comes down to **searching a directory file** as fast as possible:
  - A simple **random order** of **directory entries** might be insufficient (search time is linear as a function of the number of entries)
  - Indexes or **hash tables** can be used

Directories
Possible Implementations

- They can store all file related **attributes** (e.g. file name, disk address – Windows) or they can contain **a pointer** to the data structure (i-node) that contains the details of the file (Unix)

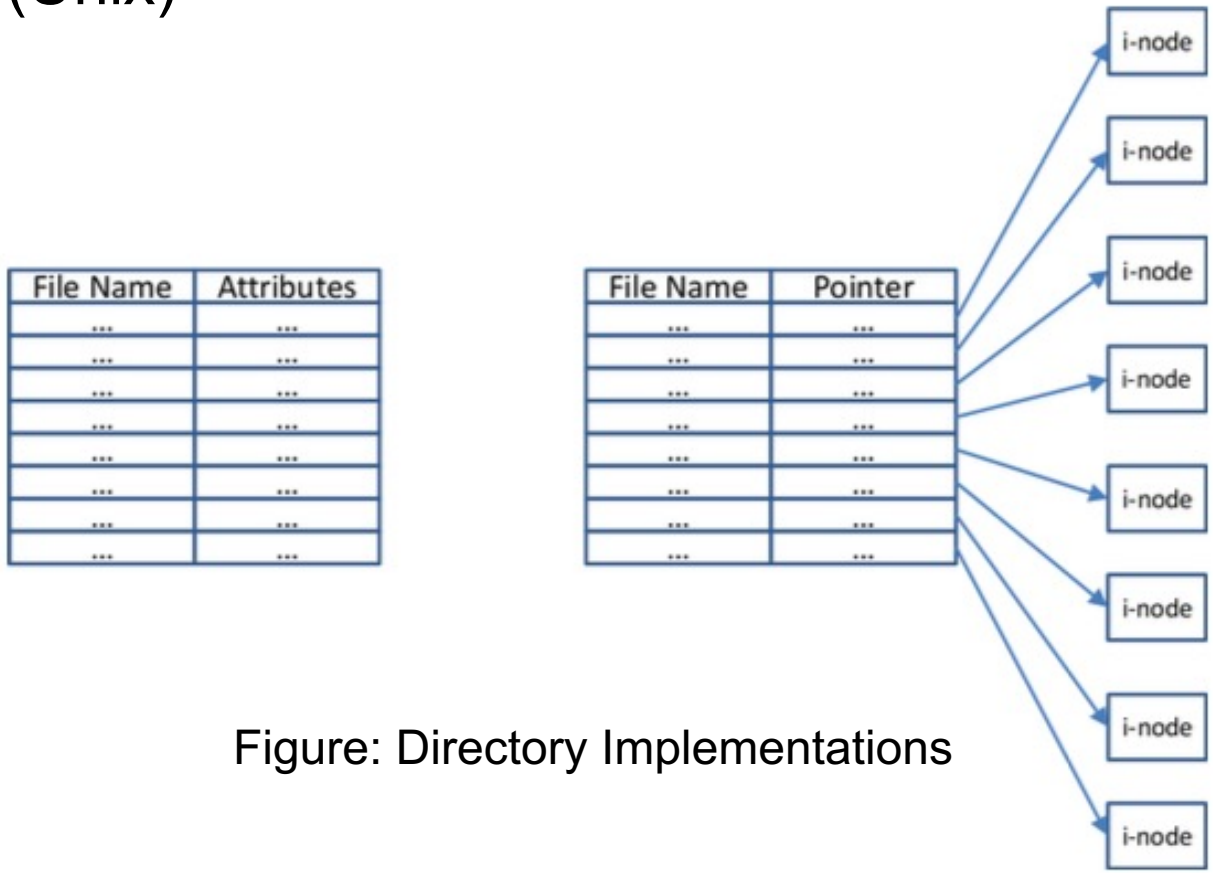


Figure: Directory Implementations

Directories
Implementation with i-nodes

- In **UNIX**, all information about the file (type, size, date, owner, and block pointers) is stored in its i-node.
- Therefore, **directory tables** are very simple data structures composed of file name and a pointer to the i-node.
- Note that directories are no more than **a special kind of file**, so they have their **own i-node**.

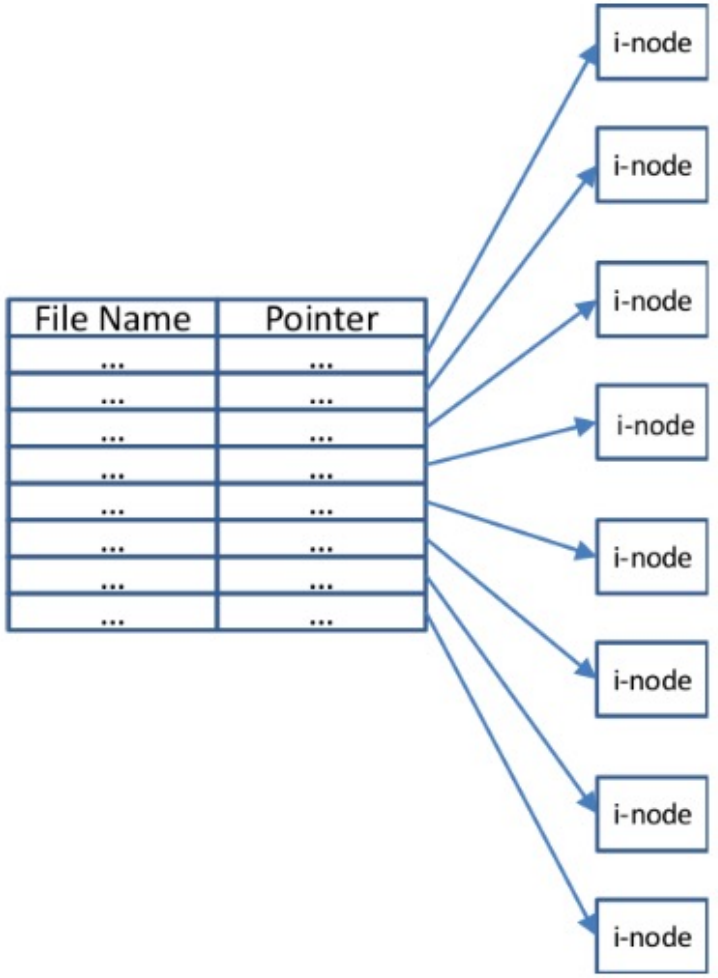


Figure: I-node Directory Structure

File System Comparison
Contiguous vs. Linked vs. Indexed

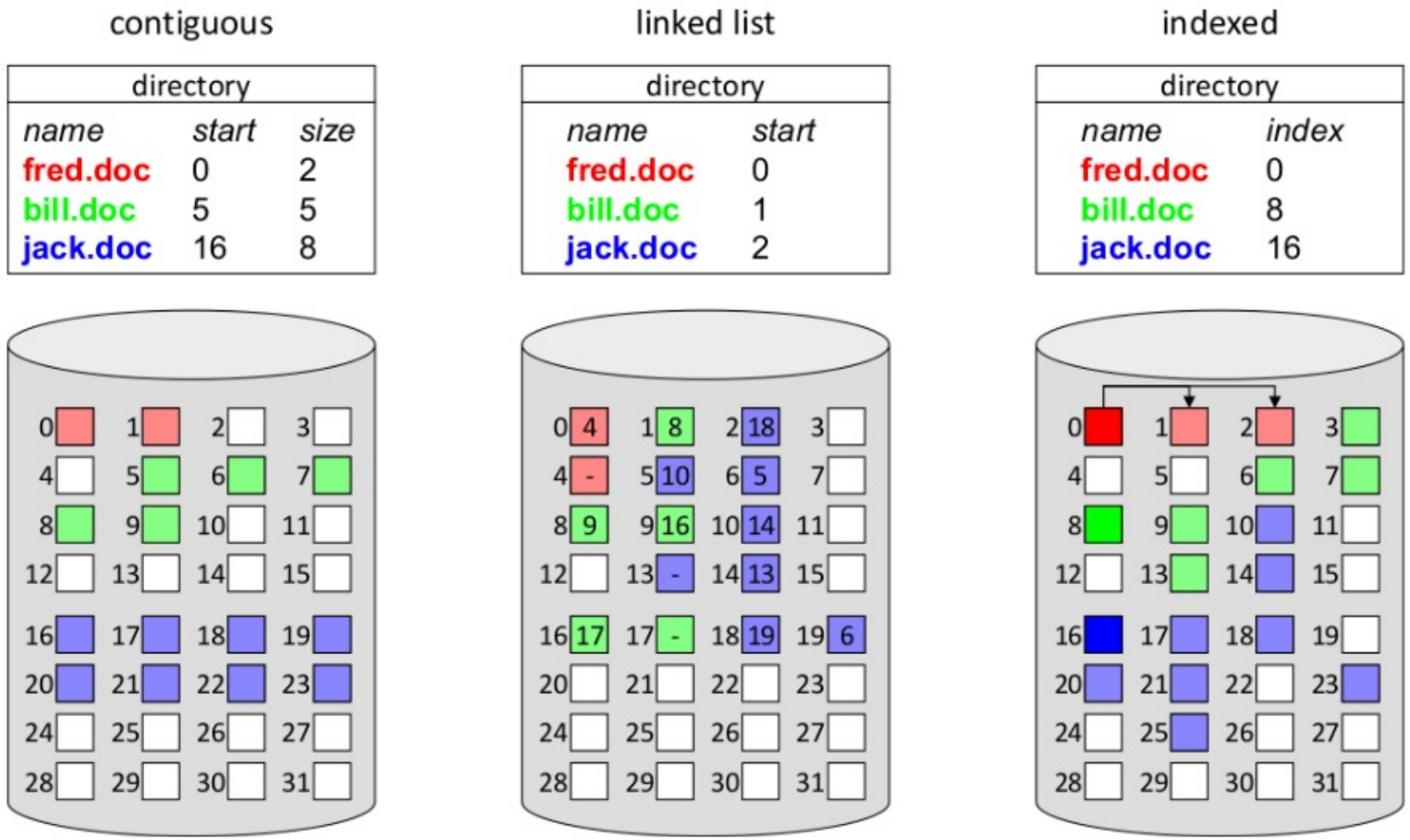


Figure: Contiguous vs. Linked List (or FAT) vs. I-nodes (or indexed)

I-nodes
Lookups

- **Opening a file** requires the **disk blocks** to be **located**
  - **Absolute file names** are located relative to the root directory
  - **Relative file names** are located based on the current working directory
- E.g. Try to locate /usr/gdm/mbox

/ inode 1

size
mode
times
2

/ contents 2

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

/usr inode 6

size
mode
times
132

/usr contents 132

6	.
1	..
19	fred
30	bill
51	jack
26	gdm
45	cfi

/usr/gdm inode 26

size
mode
times
406

/usr/gdm contents 406

26	.
6	..
64	research
92	teaching
60	mbox
17	grants

Figure: Locating a File

I-nodes

Lookups

Locate the root directory of the file system

- Its i-node sits on a fixed location at the disk (the directory itself can sit anywhere)

Locate the directory entries specified in the path:

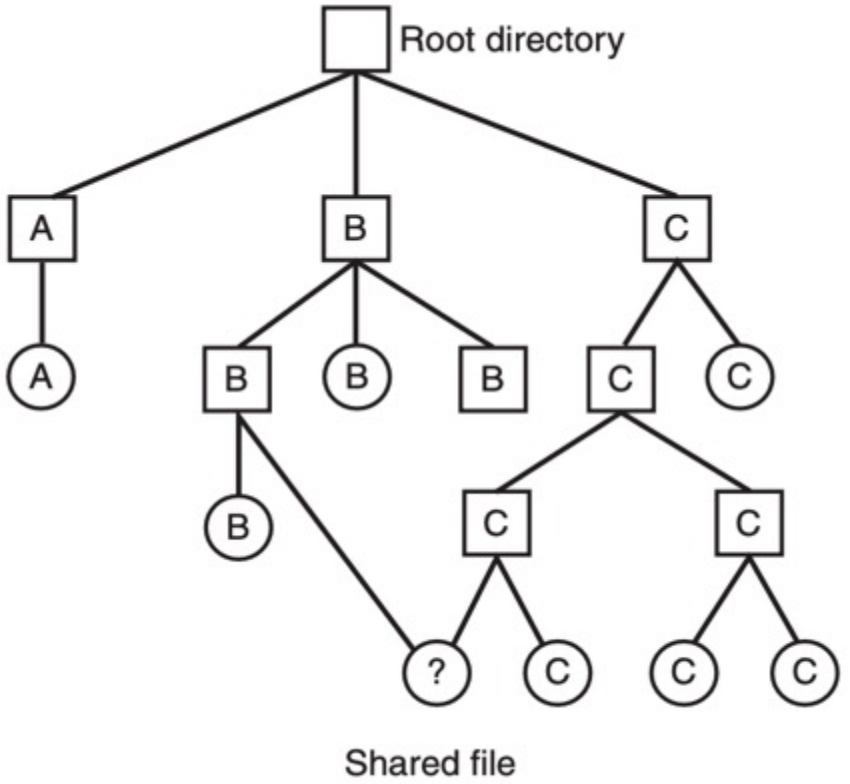
- Locate the i-node number for the first component (directory) of the path that is provided
- Use the i-node number to index the i-node table and retrieve the directory file
- Look up the remaining path directories by repeating the two steps above

Once the file's directories have been located, locate the file's i-node and cache it into memory

I-nodes: Sharing files between directories

Hard and Soft Links

- In **Directed acyclic graph (DAG)**, there are two approaches to **share a file**, e.g. between directory B and C, where C is the 'real' owner:
  - **Hard links**:
    - maintain two (or multiple) references to the same i-node in B and C.
      (In Unix: ln /C/file1 /B/file2)
    - the i-node link reference counter will be set to 2
  - **Symbolic(soft) links:**
    - The owner maintains a reference to the i-node in, e.g., directory C
    - The "referencer" maintains a small file (that has its own i-node) that contains the location and name of the shared file in directory C.
      (In Unix: ln -s /C/file1 /B/file2)
- What is the **best approach**? ⇒ both have advantages and disadvantages



Shared file

I-nodes: Sharing files between directories
Hard Links

- **Hard links** are the **fastest way of linking files**!
- Disadvantages of hard links:
  - Assume that the owner of the file **deletes** it:
    - If the i-node is also deleted, any hard link will, in the best case, **point to an invalid i-node**
    - If the i-node gets **deleted** and "**recycled**" to point to **another file**, the hard links will **point to the wrong file**!
  - The only solution is to **delete the file**, and **leave the i-node intact** if the "**reference count**" is larger than 0 (the original owner of the file still gets "charged" for the space until B decides to remove it)

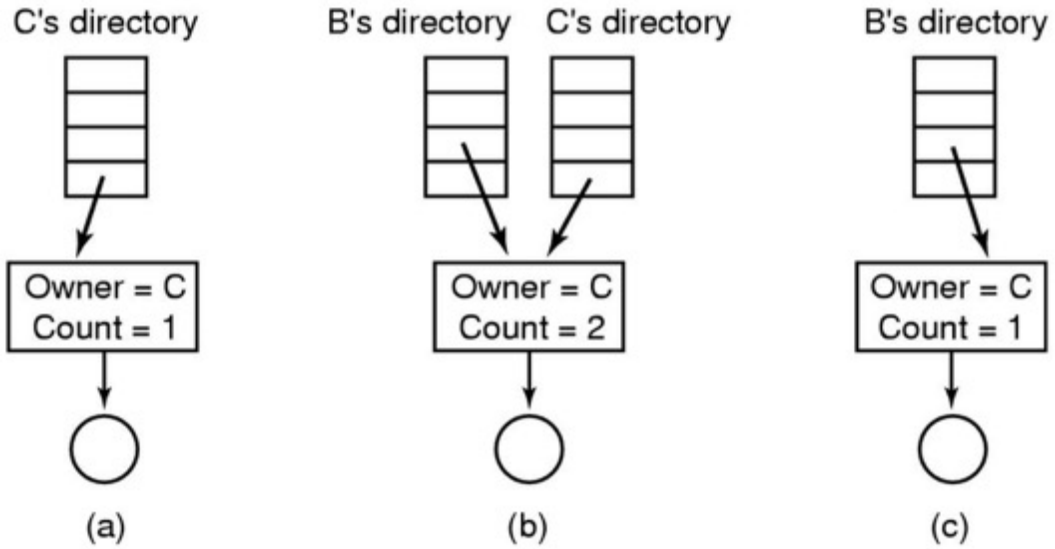


Figure: (a) Before creating a link. (b) After creating a link. (c) after the original owner removes the file

I-nodes
Soft Links

- Disadvantages of **soft links**:
  - They result in an **extra file lookup** (once the link file has been found, the original file needs to be found as well)
  - They require an **extra i-node** for the link file

- Advantages of soft links:
  - There are **no problems with deleting** the original file ⇒ the file simply does not exist any more
  - They can **cross the boundaries of machines**, i.e. the linked file can be located on a different machine

I-nodes

Hard and soft Links in Unix

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
parallels@ubuntu-linux-22-04-desktop:~$ cd Documents/
parallels@ubuntu-linux-22-04-desktop:~/Documents$ ls
test1.c
parallels@ubuntu-linux-22-04-desktop:~/Documents$ ln ./test1.c hltest
parallels@ubuntu-linux-22-04-desktop:~/Documents$ ln -s ./test1.c sltest
parallels@ubuntu-linux-22-04-desktop:~/Documents$ ls
hltest  sltest  test1.c
parallels@ubuntu-linux-22-04-desktop:~/Documents$ ls -ali
total 16
3015658 drwxr-xr-x  2 parallels parallels 4096 Nov 20 19:53 .
2621442 drwxr-x--- 16 parallels parallels 4096 Nov 20 19:52 ..
3014690 -rw-rw-r--  2 parallels parallels    4 Nov 20 19:52 hltest
3014686 lrwxrwxrwx  1 parallels parallels    9 Nov 20 19:53 sltest -> ./test1.c
3014690 -rw-rw-r--  2 parallels parallels    4 Nov 20 19:52 test1.c
parallels@ubuntu-linux-22-04-desktop:~/Documents$
```

What will happen if I delete test1.c and open the two links?

Summary
Take-Home Message

- Contiguous, linked list, FAT and i-nodes as file system implementations.
- Lookups with I-nodes.
- Hard and Soft Links