Operating Systems and Concurrency

Lecture 20: File Systems III

G52OSC

Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

Recap

Last Lecture

1. **User view** of file systems
   - System calls
   - Structures, organisation, file types
2. **Implementation view** of file systems
   - Disk and partition layout
   - File tables
   - Free space management
   - ...
3. There is a lot more happening than expected at first sight!

Goals for Today

Overview

- File system **implementations**
  1. Contiguous
  2. Linked lists
  3. File Allocation Table (FAT)
  4. I-nodes (lookups)

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.

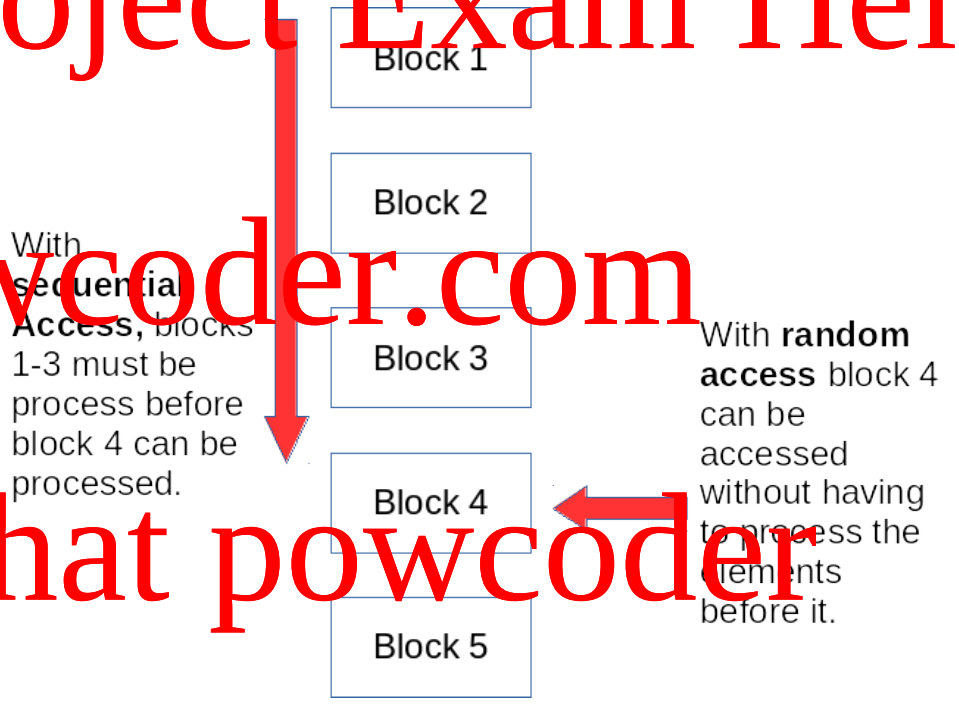

Figure: Types of access to a file

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)
- 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)

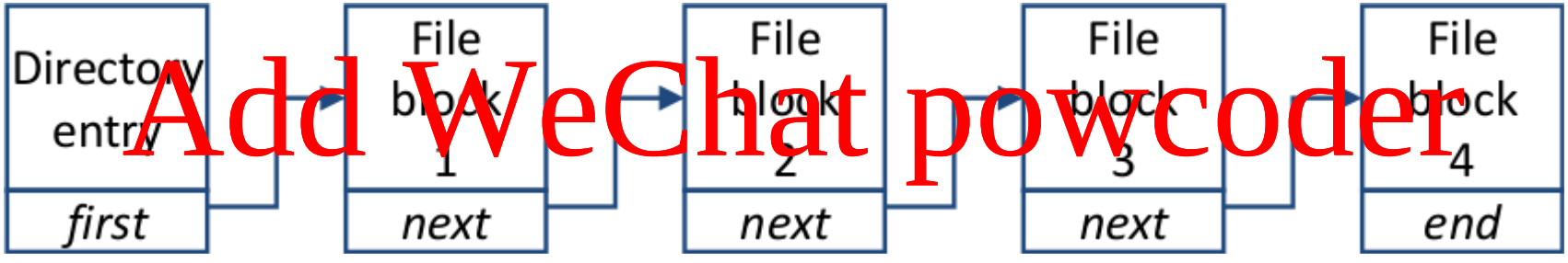

Figure: Linked List File Storage

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
- There is some **internal fragmentation**; on average the last half of the block is left unused
  - Internal fragmentation will reduce for **smaller block sizes**
- May result in **random disk access**, which is very slow (remember the example in lecture on disks)
  - **Larger blocks** (containing multiple sectors) will be faster

Linked Lists

Disadvantages (Cont'ed)

- 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!

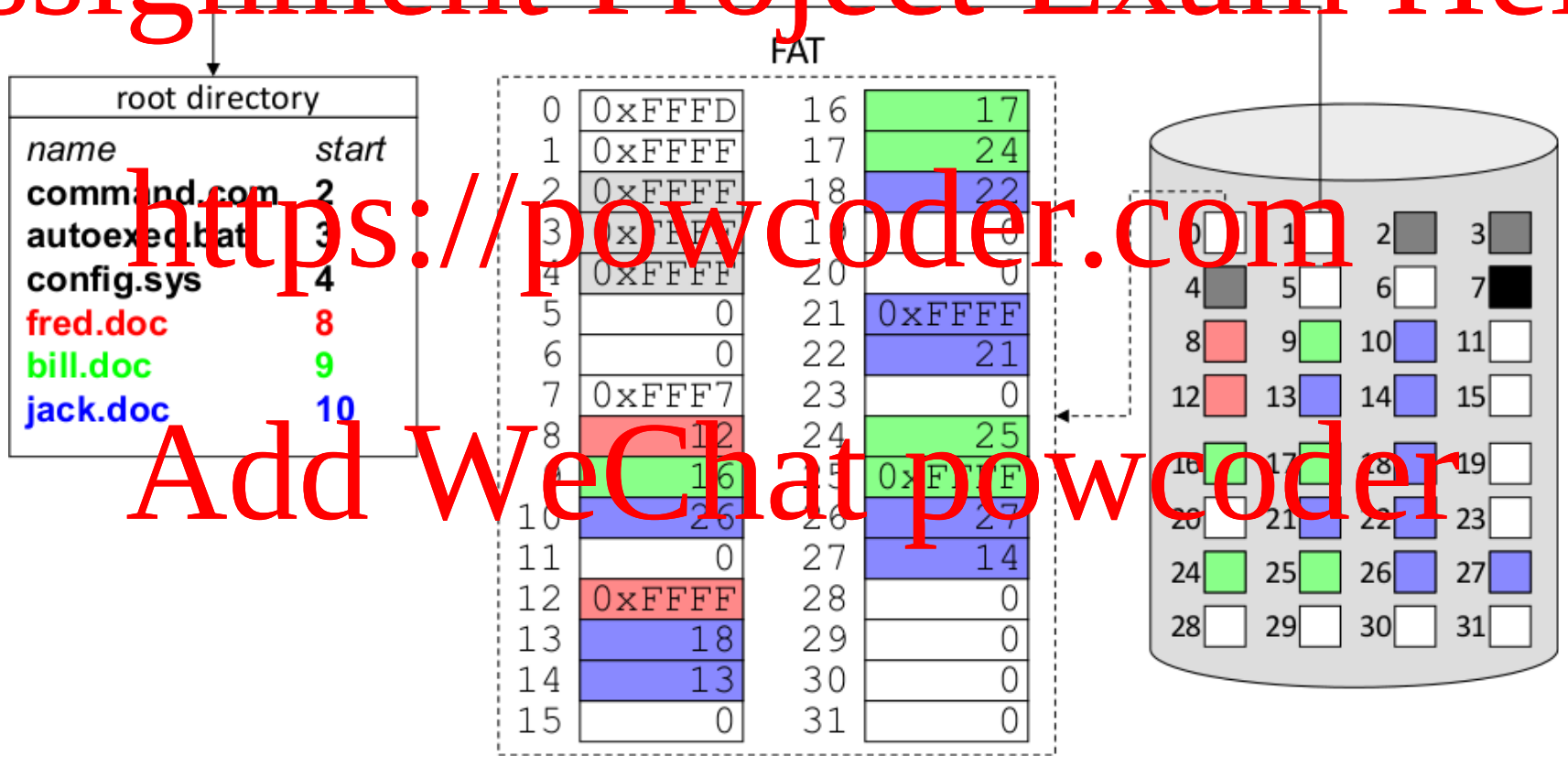

Figure: File Allocation Tables

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

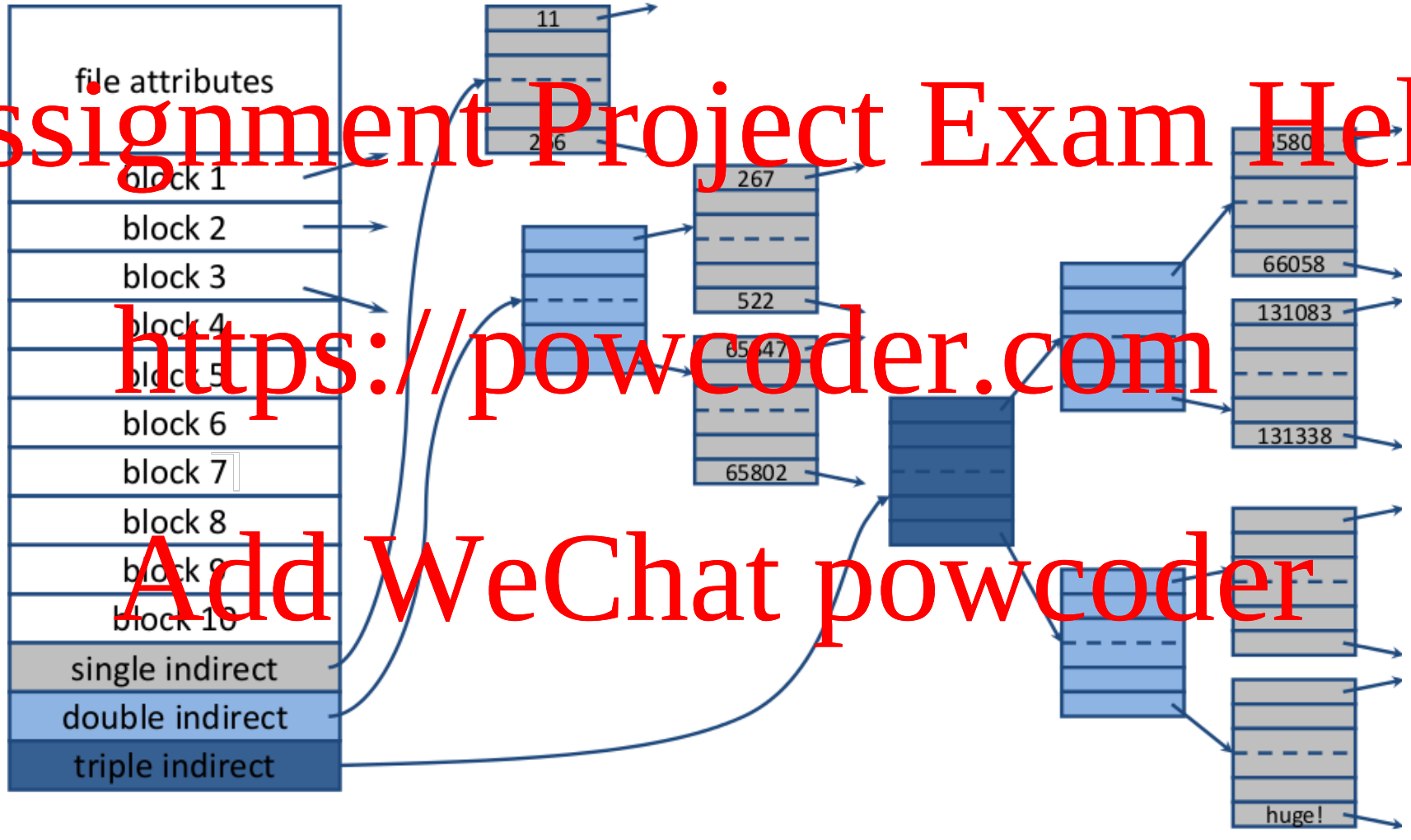

Figure: File Storage Using I-Nodes

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$\times$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; $2^{10}/2^2 = 2^8 = 256$). That is, with 10 direct blocks + 1 indirect block, we can have files with up to 266 blocks ($\Rightarrow$ 266KBs).

- A double indirect points to a block of 256 block pointers. Each of which points to 256 indirect blocks. Therefore, with the double indirect we could have files with size up to 266KBs + (256 $\times$256 = 65536) $\Rightarrow$ 65802 KBs.
- If we need files larger than 64M, we will need a triple indirect.

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.

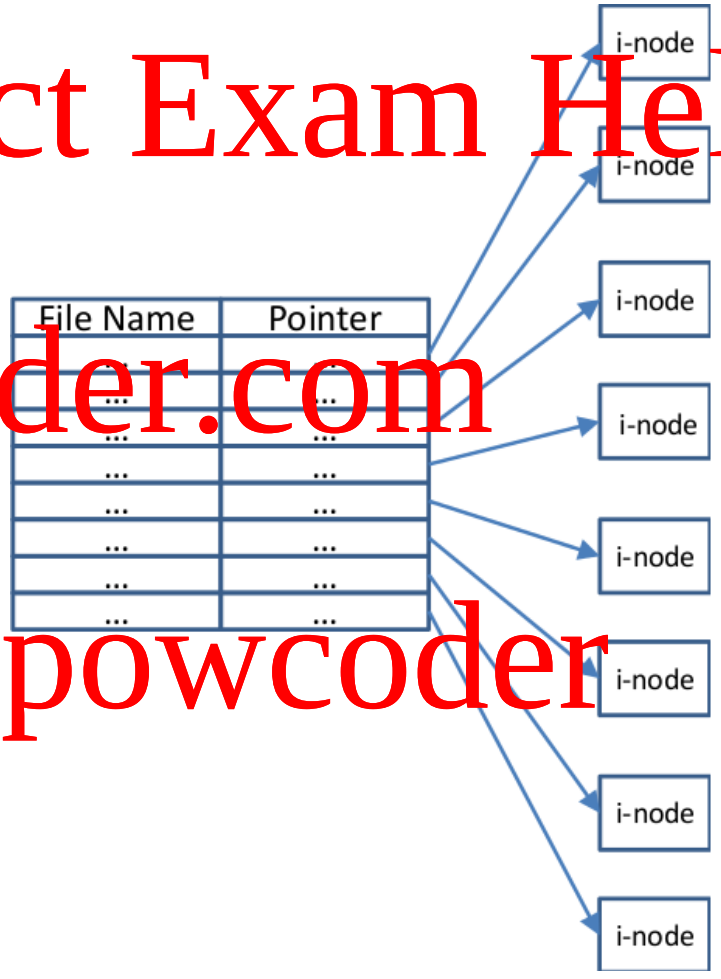

Figure: I-node Directory Structure

File System Comparison

Contiguous vs. Linked vs. Indexed

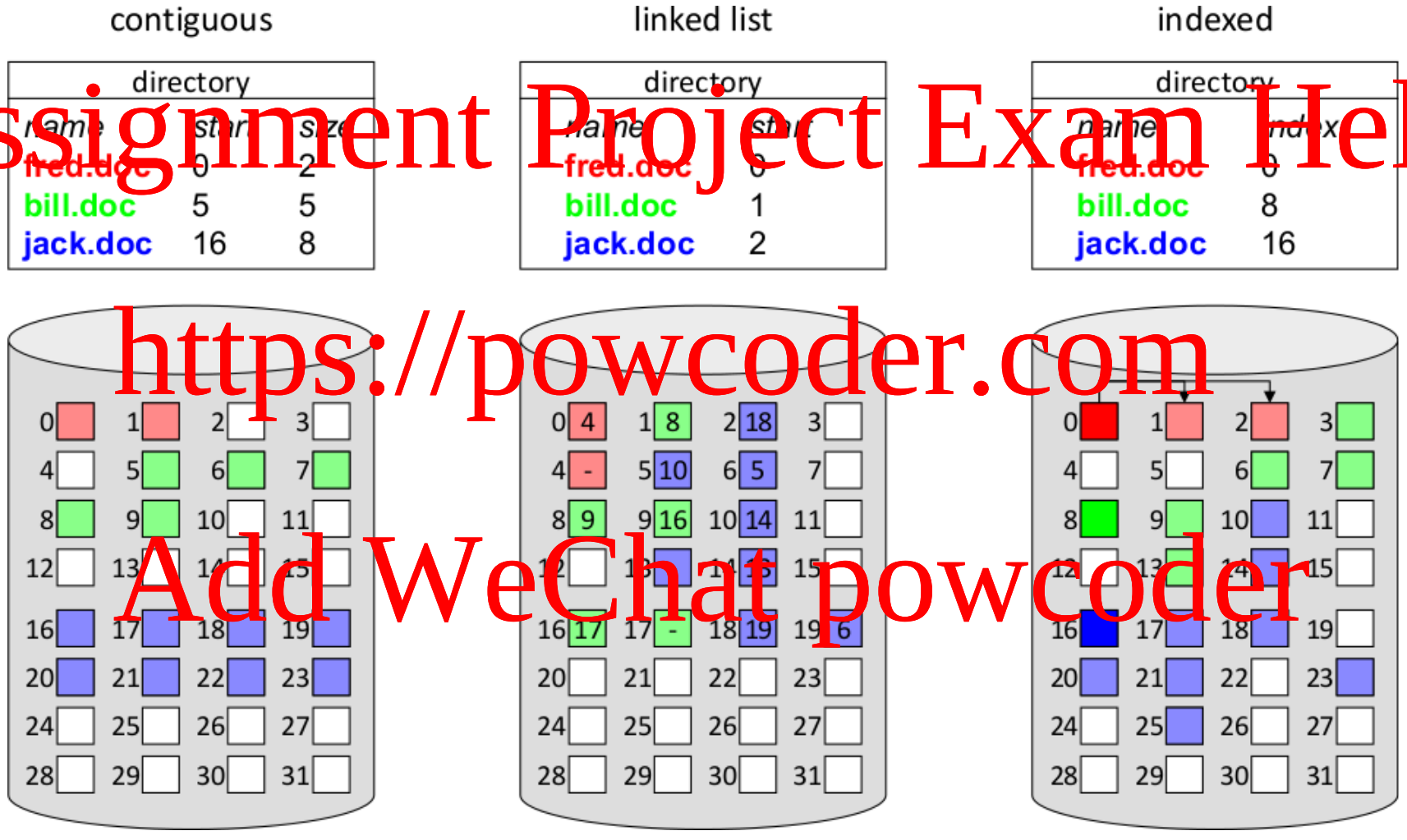

Figure: Contiguous vs. Linked List (or FAT) vs. I-nodes (or indexed)

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`

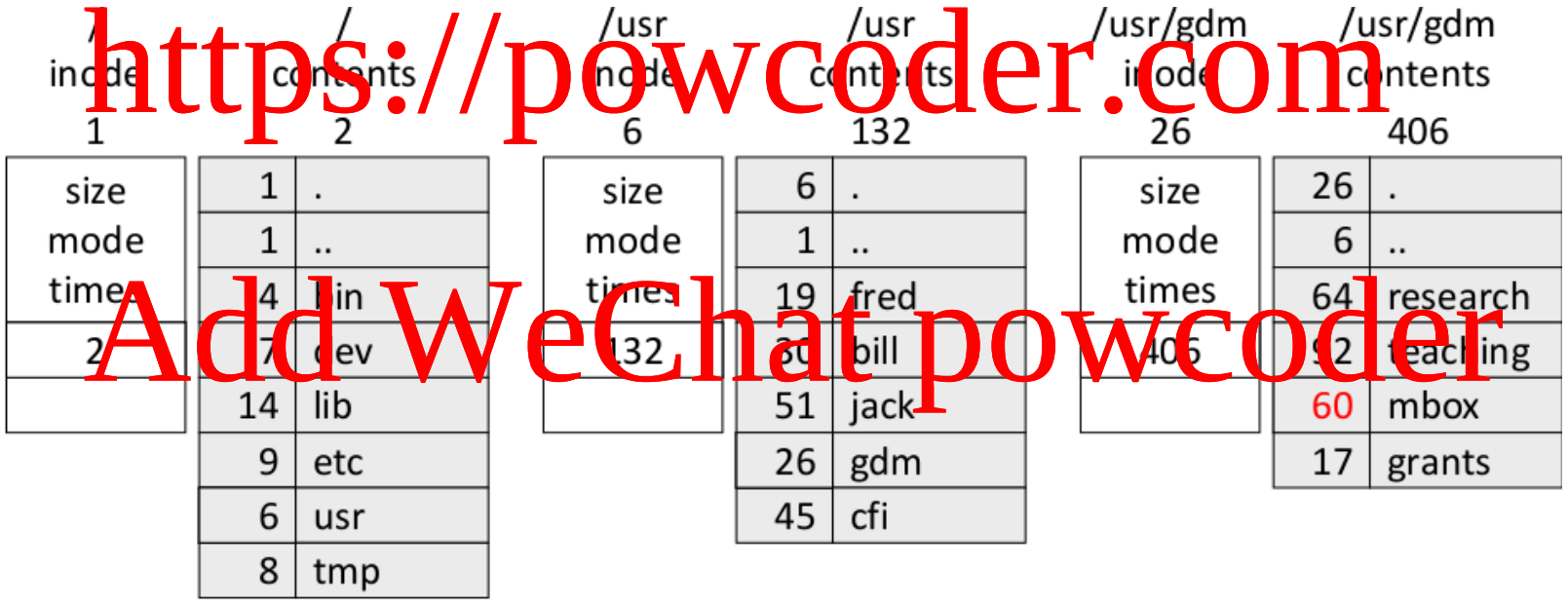

Figure: Locating a File

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

File System Examples

Unix vs. Windows

- The Unix V7 File System
  - **Tree structured** file system with links
  - Directories contain **file names** and **i-node numbers**
  - I-nodes contain **user and system attributes** (e.g. count variable)
  - One single, double, and triple **indirect blocks** can be used

- More sophisticated File Systems were later developed (e.g. ext3)
- Windows (up to XP) used FAT-16 and FAT-32.

- Windows (from XP) moved to NTFS (64 bits) because of file size limitations.
  - NTFS uses a similar idea to i-nodes (Master File Tables), with bigger i-nodes that can also contain small files and directories.

Recap

Take-Home Message[1]

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

[1] Tanenbaum Section 4.3, 4.5.2

File system implementations

Food for thought

We have seen that with i-nodes, the maximum file size that we can have depends on the block size and the number of indirections.

- Assuming a 32-bit disk address space, what would be the maximum (theoretical) file size for the FAT file system with a drive of 500GB and a block size of 1KB? (without accounting for directory metadata)
- The most used implementation of FAT is known as FAT-32. Investigate why there is a theoretical limitation of 4GB per file (and sometimes even less than 2GB).

Submit your answer at:

`https://b.socrative.com/login/student/`

Room name: G52OSC