File Systems

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

June 10, 2017

Overview I

File System Requirements

- Ease of use for users
- Reliability
- Provide a common interface to numerous technologies
- Good performance (throughput/response time)
- Security (esp. for multi-user/networked systems)
- Ability to create/modify files
- Ability to manipulate file system structure
- Provide long term storage for user data
  - File system must be durable, files should not disappear
  - Robust to system crashes, error free r/w
- Ensure data is protected from invalid access, viruses, other users.

Definition

- File system is an OS layer that maps the blocks of data on the disk into files and directories
- Allows blocks to be allocated to files (file creation)
- Allows blocks to be specified by name (reference)

Data Hierarchy

- Lowest level stores sequences of bits: 1's and 0's
- Collections of bytes form words
- Word length depends on CPU architecture
- Characters are a mapping of bytes to symbols
- Fields are groups of characters
- Records are groups of fields
- Files are groups of records
- Volume is unit of storage for files

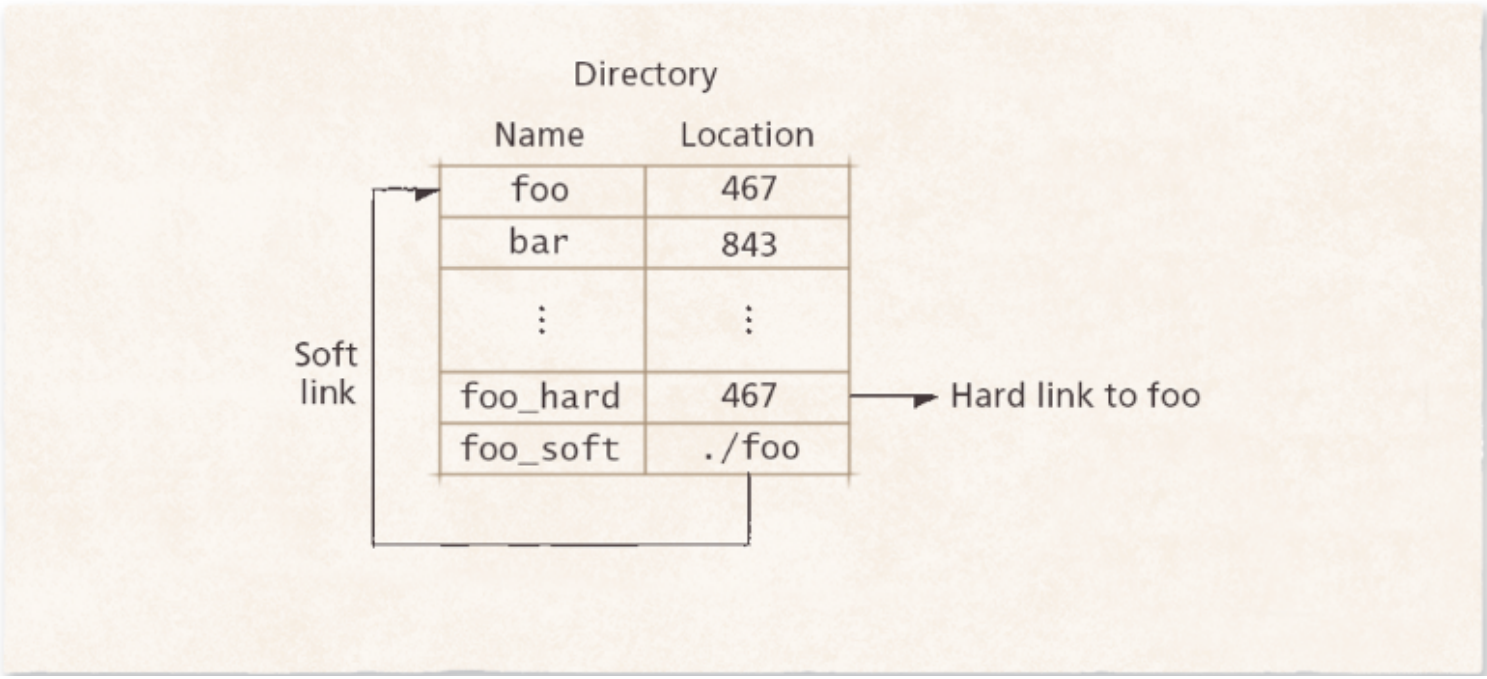
File System

- Provides support for acting on files as a whole
  - Opening, closing, creating, deleting and copying
- Provides support for accessing data within a file
  - Reading, writing, modifying, insert and delete file
- Maintains important file characteristics
  - Location, permission, type, access and creaation times
- Location is both logical and physical
  - Logical location is user view of file
  - Physical location is actual location on disk

Directories

- Users view: Directories are containers for files/sub-directories
- File system view: Directories are files; rather than user data, store information about other files etc.
- Most file systems are hierarchial.

- Symbolic/Soft links:
  - Directory entry containing pathname to linked file
  - Breaks if file is moved or renamed
- Hard links:
  - Directory entry containing a physical block number
  - Can become invalid if physical location is changed
  - Unaffected by moving the logical location of the file

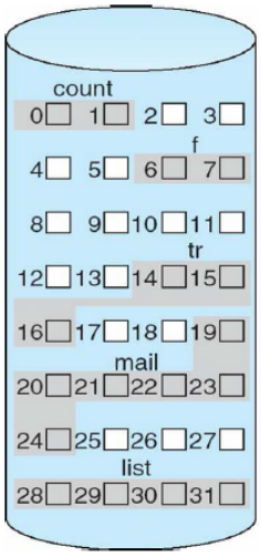


File Allocation Schemes

- Main idea behind allocaation is effective utilization of file space and fast access of the files
- Three types:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

- Each file has to occupy contiguous blocks on disk
- The location of a file is defined by the disk address of the first block
- Stores file data at contiguous address on the device.
- Successive logical records are physically adjacent
- Provides fast access to data
- Leads to fragmented file system (like memory)
- Suffficient contiguous space needed to create the file
- Best suited to write-once media (e.g. DVDs)

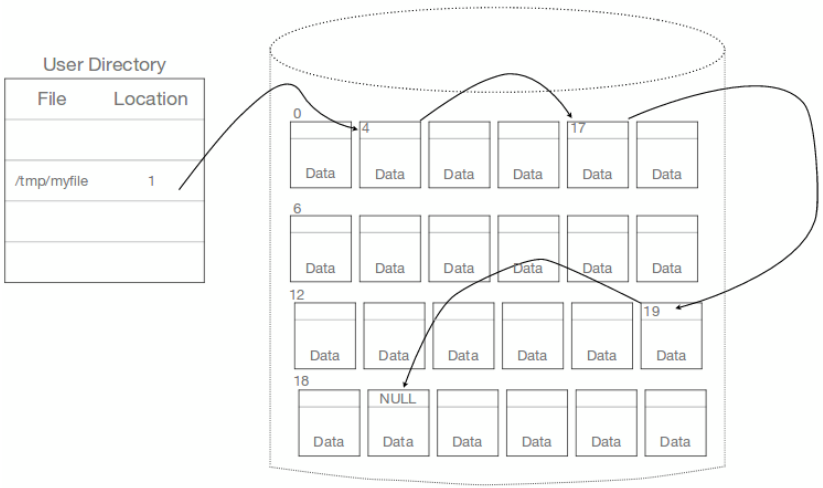


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked-List File Allocation

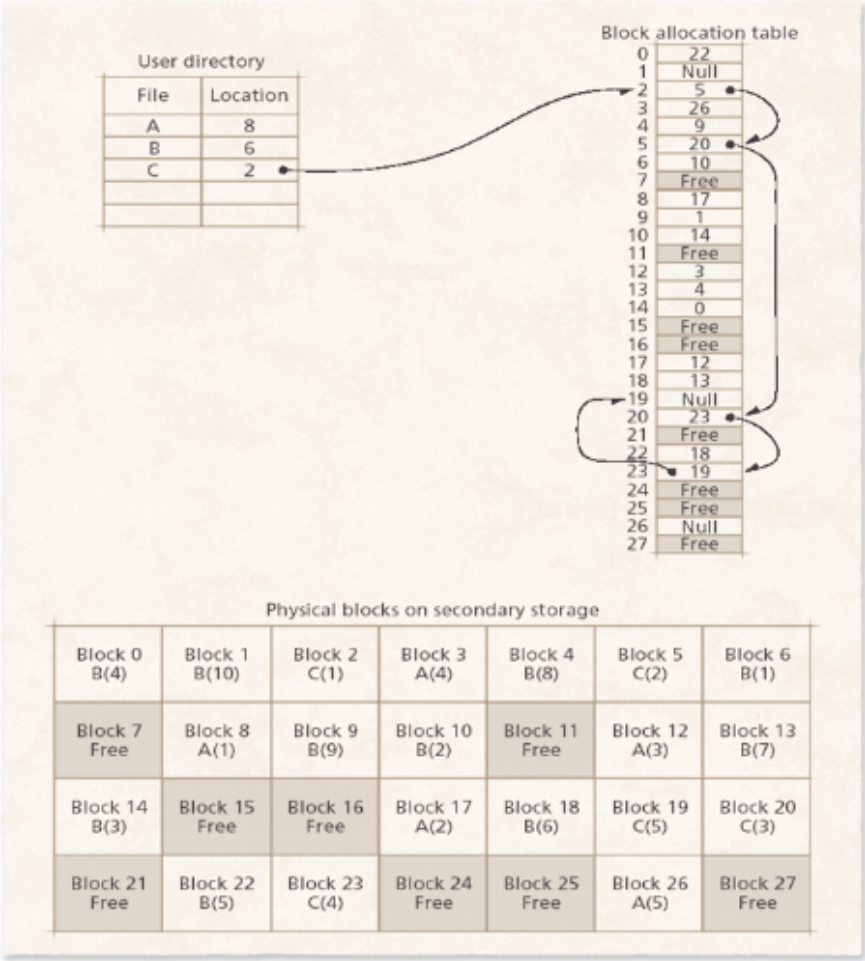
- Each file is a linked list of data "chunks"
  - Could be linked at sector level, or block level
  - Sector level provides finer grained segmentation
  - Block level provides fast access, but suffers internal fragmentation
- Files can be easily grown:
  - There is no external fragmentation with linked allocation
  - Performance might be an issue
- Only supports sequential access:
  - Need to traverse the linked list to locate particular segments



File Allocation Table (FAT)

- File stores reference into a block allocation table called the FAT
- Like linked allocation, exceoot don't keep the

  links in the blocks themselves, use FAT instead:
  - Each FAT entry is a disk sector or number
  - Special values for "last block in file" and "free block"
- FAT stored on disk but cached when file system is mounted
- Can support random access to files
- Originally each FAT enter was 16 bits:
  - Unsustainable for large disks
  - Problem addressed by FAT32 standard

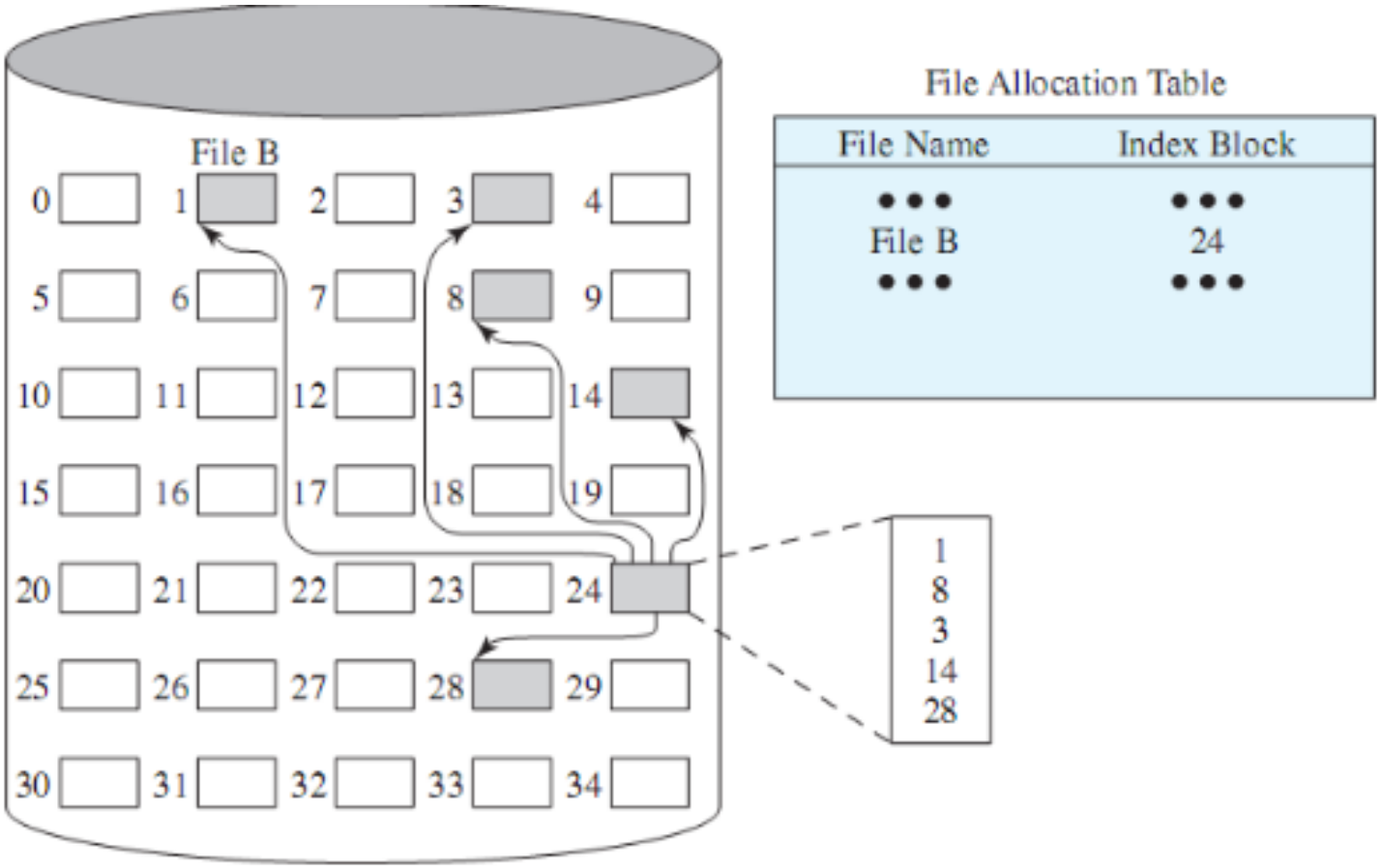


Indexed File Allocation I

- Supports random accesses by bringing pointers to blocks together into an index block
- Allocate block pointers contiguously in metadata (index block) when file is created
- Files can be easily grown within limit of index block size
- Allocate blocks on demand by filling in the pointers dynamically as file is written
- When applying this strategy, acccessing any place on the disk requires no more than two disk operations, one to acccess the index block, one to acces the data.
- But the size of the file is limited. For example, if disk blocks have 4kB and the block number occupies 32 bytes, the maximum size of the file is $1024 \times 4\text{kB} = 4\text{MB}$.

Indexed File Allocation II

- This constraint is addressed by Linked-Indexed and Multilevel-Indexed File allocation schemes.

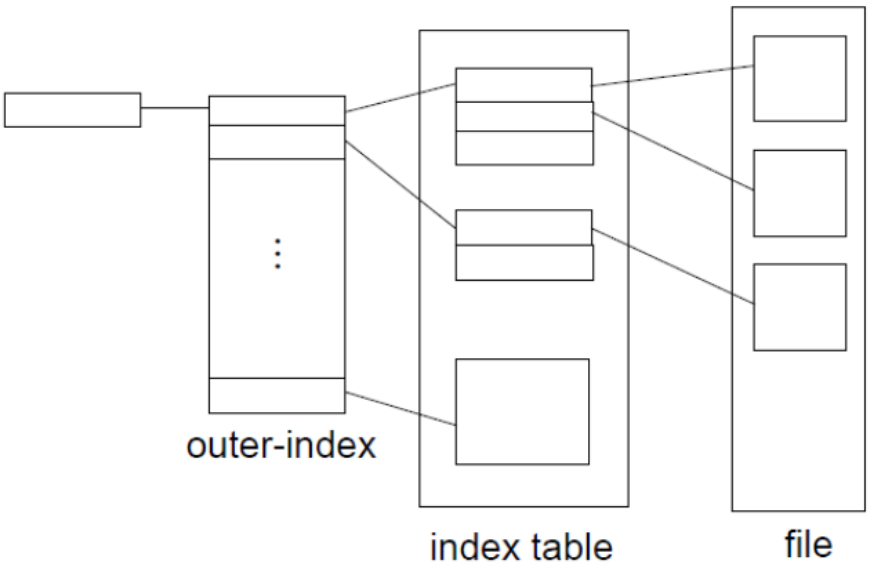


Linked-Indexed Allocation

- A combination of linked and index approaches
- The numbers of consecutive disk blocks with the file contents are gathered in the index block.
- A file is identified by the number of the first index block
- If the numebr of blocks creating the files doesn't fit into one index block, the last number in this block points to the next index block.
- For example - if the disk blocks have 4kB and the block number has 32 bytes, then one index block can contain 1023 block numbers with file contents and one number of following index block. IF we have a 42MB file, then we'll also have 11 index blocks. Access to data fragments at the end of the file may require 12 disk operations

Multilevel-Indexed File Allocation

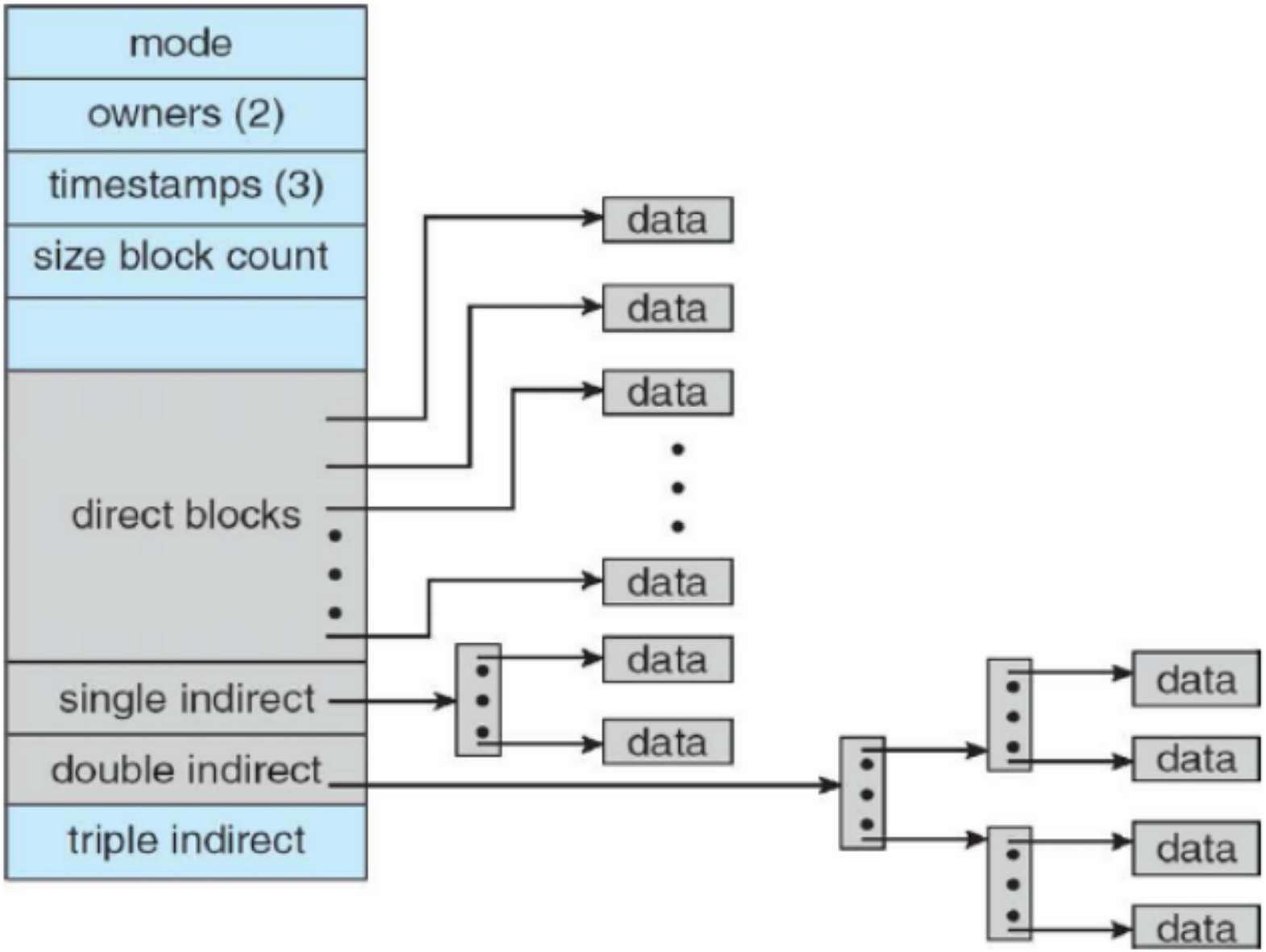
- Multilevel index allocation means that index blocks with disk blocks containing file contents create a tree of a specific depth
- For example, in the two-level indexing, the main index block (first level) contains the numbers of index blocks of the second level, which in turn contain the number of blocks constituting the file contents.
- For example, if the disk blocks have 4kB and block number occupies 32 bytes, then with two-level indexing the size of files is limited to 4GB and with third level indexing it is limited to 4TB.



UNIX - inodes I

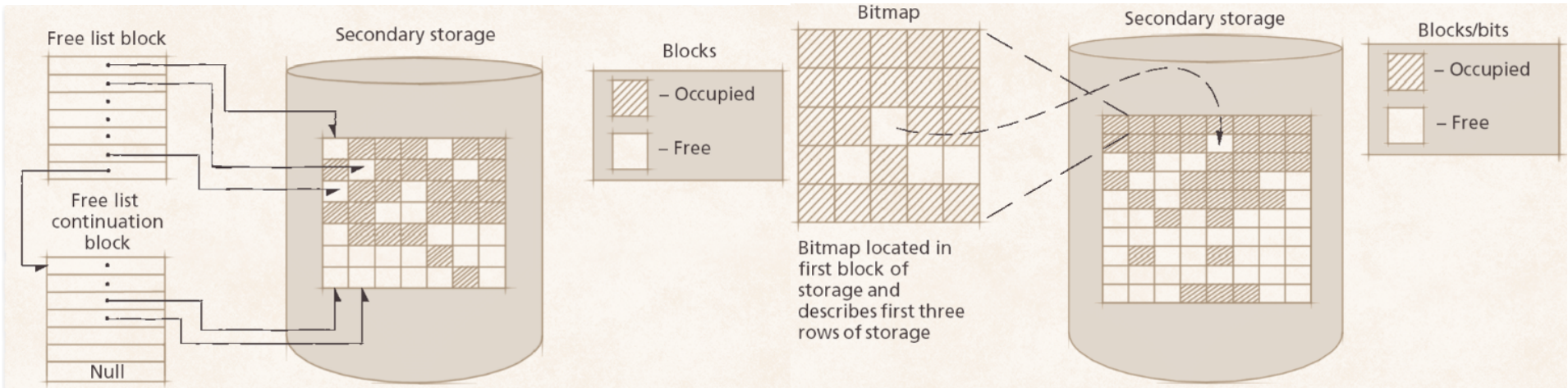
- In the UNIX system files are represented by so called inodes
- One inode for each
- An inode contains an index of disk block numbers
- The first 10 are the numbers of the first 13 file contents blocks
- The eleventh number is a number of the index block (first level) to the next file blocks
- The last number is the nuumber of the third level index block of following file blocks
- Thanks to this approach, the number of disk operations required to access any place in the file depends on the file size and oscillates between 1 and 4.

UNIX - inodes II



Managing Free Space

- Not all blocks are occupied.
- There has to be a way of storing informaaation concerning empty blocks
- Two methods considered:
  - Maintaining a list of empty blocks
  - Maintaining a bitmap image of blocks



The End