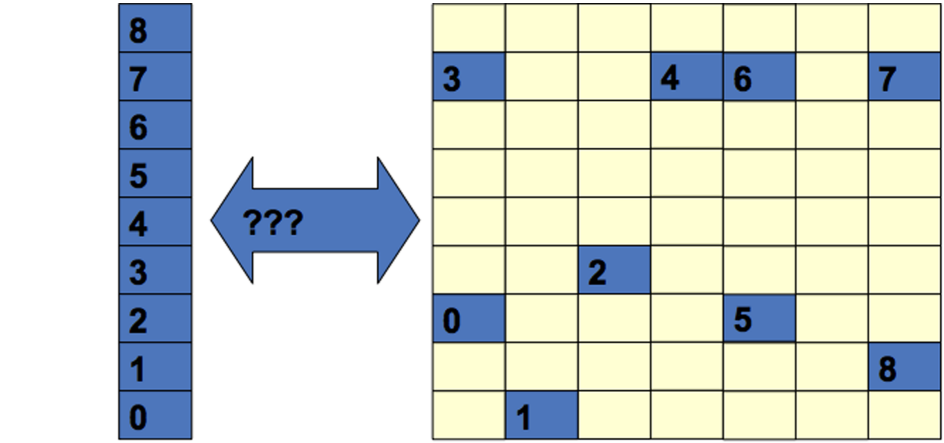
Stýrikerfi (Operating Systems)

17. Implementing File Systems (Overview and File Allocation)

Hans P. Reiser | Spring term 2025

Reykjavik University

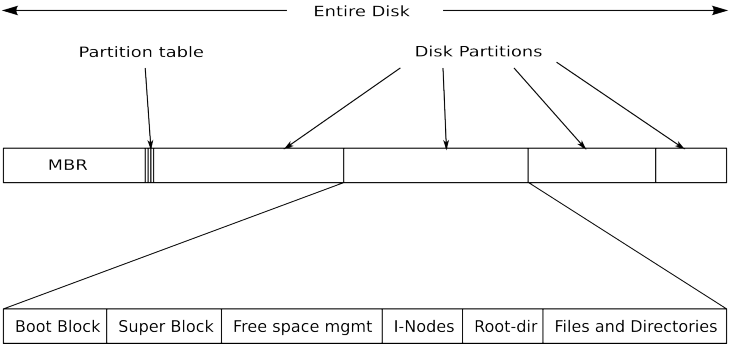
Implementing Files



Disk Structure

- Disk can be subdivided into partitions (sometimes also called minidisks or slices)
- Disks partitions can be used raw – without a file system –
  or formatted with a file system (FS)
- Entity containing a FS is known as a volume
- Many general-purpose FSs exist (and can co-exist on the same computer system)
  (Examples: FAT (FAT12/16/32), exFAT, NTFS, HFS/HFS+, HPFS, APFS, UFS, XFS,
  Files-11, ext2/3/4, btrfs, VMFS, ZFS, ReiserFS, JFFS/JFFS2, YAFFS, LogFS, F2FS)
- *Special-purpose FS:* Any file system that is not a disk file system
  (Examples: procfs, tmpfs, SSHFS, UnionFS, EncFS)

Implementing a FS on Disk [TB15]



- First sector of disk = MBR
  - Boot info (if PC is booting, BIOS reads and executes MBR)
  - Disk partition info
- Sector 0 of partition is volume boot record
- FS-specific internal organization within partition

Implementing Files

- Disk typically organized as fixed-sized **blocks**
- FS uses blocks as **data region** (for user data) and for **metadata**:
  - *Which logical block belongs to which file, in what order?*
  - *Size, owner, access rights, access/modifcation time stamps, etc. for each file*
  - *Which blocks are free for the next allocation?*
- Typical data structures for meta data
  - Inodes (index node) (usually fixed-size, multiple indes per block)
  - Allocation tracking: free list, data bitmap, inode bitmap
  - Superblock (magic number, file system type id, FS info: number of data and inode blocks)
- Creating (and updating) files might imply allocating new blocks (and modifying old blocks) on the disk

Allocation Policies

- How to allocate data blocks for a (new) file?

- Preallocation:
  - Need to know maximum size of a file at creation time
    (in some cases no problem, e.g. file copy)
  - Difficult to reliably estimate maximum size of a file
  - Users tend to overestimate file size, just to avoid running out of space
- Dynamic allocation:
  - Allocate in pieces as needed

Fragment Size

- Extremes
  - Fragment size = length of file
  - Fragment size = smallest disk block size (sector size)
- Tradeoffs:
  - Contiguity $\Rightarrow$ speedup for sequential accesses
  - Many small fragments $\Rightarrow$ larger tables needed to manage free storage management as well as to support access to files
  - Larger fragments help to improve data transfer
  - Fixed-size fragments simplify reallocation of space
  - Variable-size fragments minimize internal fragmentation, but can lead to external fragmentation

Implementing Files

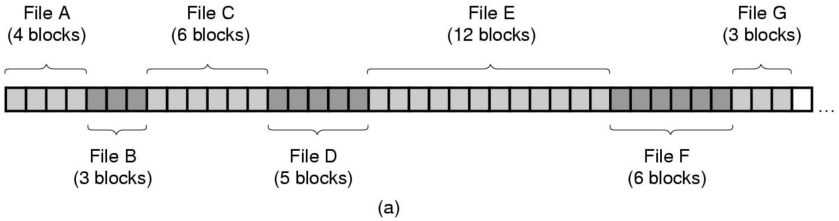
- 3 ways of allocating space for files:
  - contiguous
  - chained (also called linked)
  - indexed

Contiguous Allocation

- Array of N contiguous logical blocks reserved per file (to be created)
- Minimum meta data per entry in FAT/directory
  - Starting block address
  - N
- *What is a good value for N?*
- *What to do with an application that needs more than N blocks?*
- Discussion similar to ideal page size
  - Internal Fragmentation
  - External Fragmentation

$\Rightarrow$ scattered disk

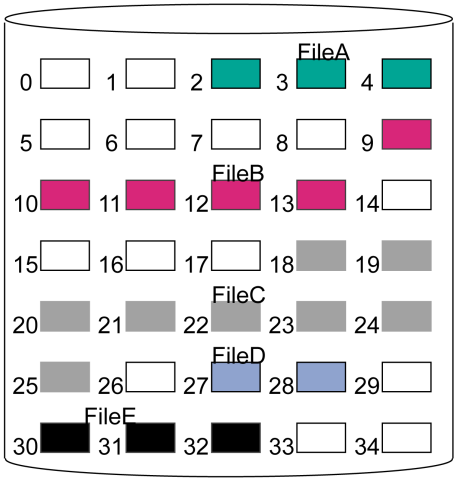
Contiguous Allocation: Scattered Disk [TB15]



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files D and F have been removed

Contiguous Allocation [Sta17]

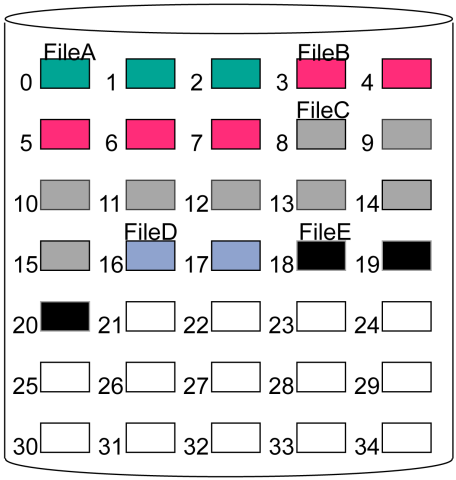


File Allocation Table

File Name	Start Block	Length
FileA	2	3
FileB	9	5
FileC	18	8
FileD	27	2
FileE	30	3

Remark: To overcome external fragmentation $\Rightarrow$ periodic compaction

Contiguous Allocation after Compaction [Sta17]



File Allocation Table

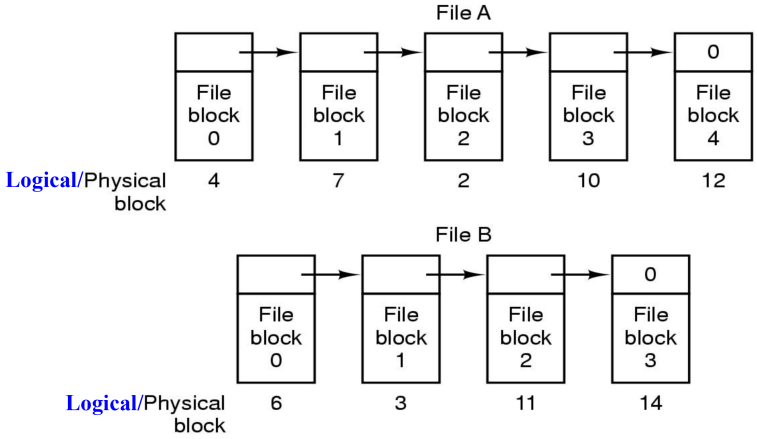
File Name	Start Block	Length
FileA	0	3
FileB	3	5
FileC	8	8
FileD	16	2
FileE	18	3

Chained Allocation (Linked List)

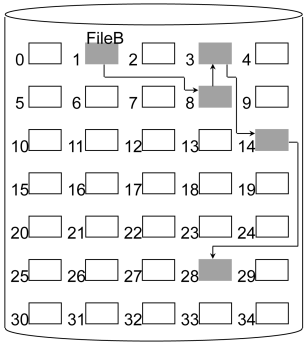
- Linked list of logical file blocks for each file:
  - Each file block has a pointer to next file block
  - If stored in block: the amount of data space per block is no longer a power of two
    $\Rightarrow$ Consequences?
  - Last block contains a NIL-pointer (e.g. -1)
- Directory (or file allocation table) contains address of first file block
- No external fragmentation: Any free block can be added to the chain
- Only suitable for sequential files: No easy backwards-seek
- No accommodation of the principle of disk locality
  - File blocks will end up scattered across the disk
  - Run a defragmentation utility to improve situation

Chained Allocation

- Storing a file as a linked list of disk blocks [TB15]



Chained Allocation [Sta17]



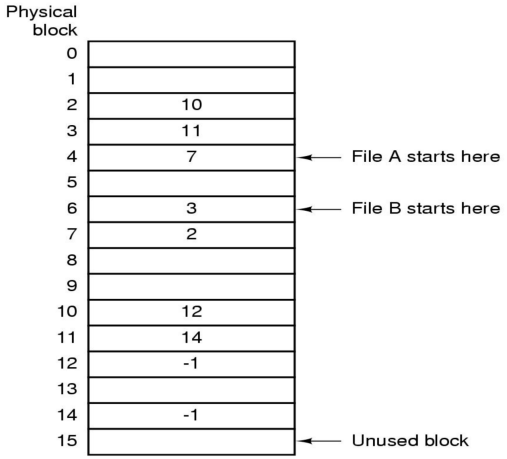
File Allocation Table

File Name	Start Block	Length
...	...	...
FileB	1	5
...	...	...

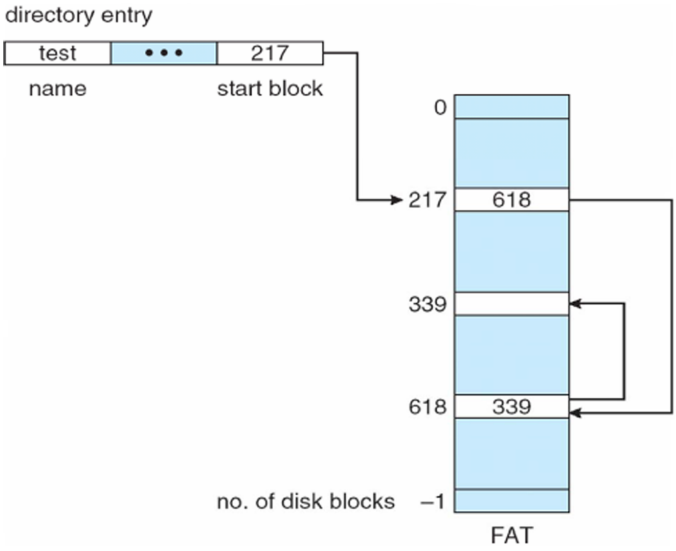
Remark: With sequential access only, this implementation is acceptable. However, requesting an individual record requires tracing through the chained blocks, which requires far too many disk accesses.

Linked List Allocation within RAM [TB15]

- Each file block used for file data only
- Separate file allocation table with linked list info
  - To be kept in RAM for fast(er) access
  - Avoids disk access when searching for block
  - Entire block is available for user data
  - Table gets far too large for modern disks
    - Can cache only, but still consumes significant RAM
    - Used in MS-DOS (FAT FS), OS/2
- Similar to an inverted page table, one entry per disk block



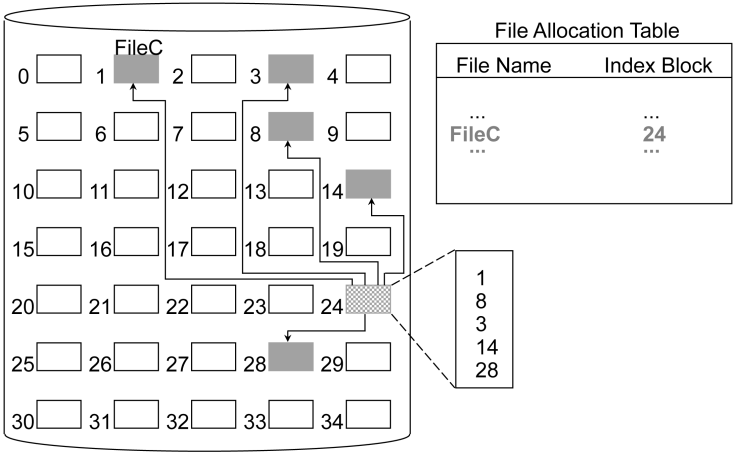
Linked List Allocation: File Allocation Table [SGG12]



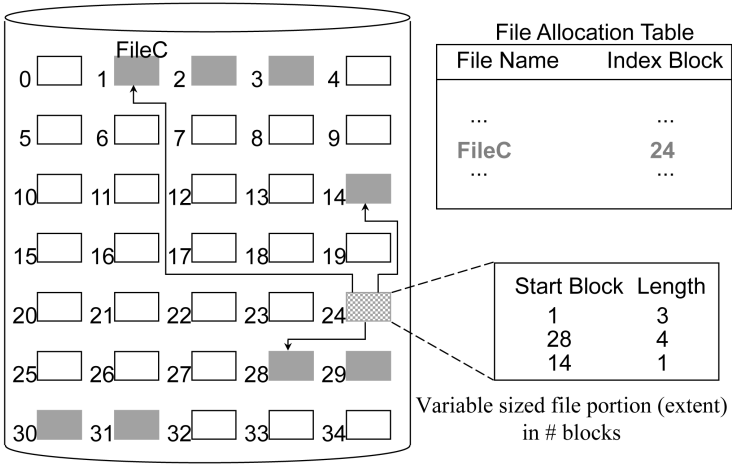
Indexed Allocation

- File allocation table contains a index table for each file
  - Fixed-size array of fragment pointers
    - Fixed block fragments
    - Variable block fragments
- Allocate space for pointers at file creation time

Indexed Allocation (fixed size) [Sta17]



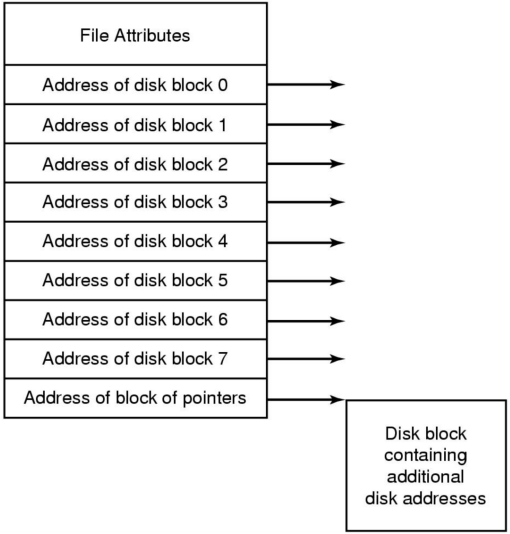
Indexed Allocation (variable size) [Sta17]



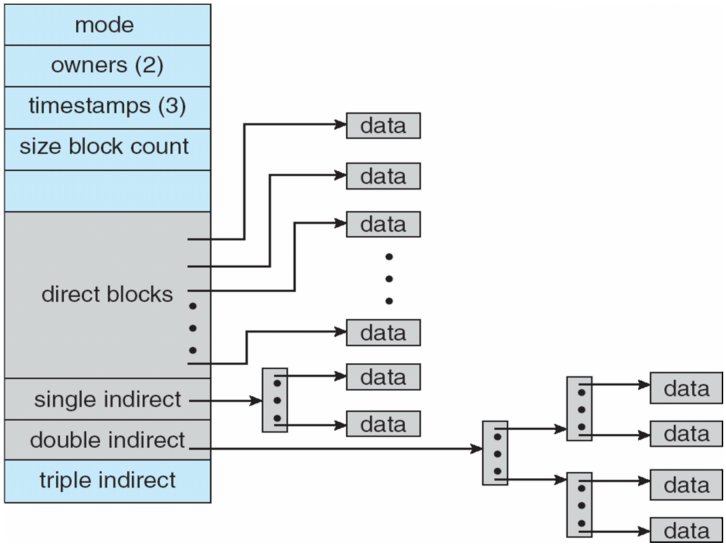
Analysis of Indexed Allocation

- Supports sequential and random access to a file
- Fragments
  - Block sized
    - Eliminates external fragmentation
    - Large overhead for unneeded pointers (most files are small!)
  - Variable sized
    - Improves contiguity
    - Reduces index size

Multi-Level-Indexing [TB15]



Example: UNIX (4k bytes per block) [SGG12]



Summary: File Allocation Methods

characteristic	contiguous	chained	indexed	
preallocation?	necessary	possible	possible	
fixed or variable size fragment?	variable	fixed	fixed	variable
fragment size	large	small	small	medium
allocation frequency	once	low to high	high	low
time to allocate	medium	long	short	medium
file allocation table size	one entry	one entry	large	medium

References I

[SGG12] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 9th edition, 2012.

[Sta17] William Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall Press, USA, 9th edition, 2017.

[TB15] Andrew S Tanenbaum and Herbert Bos. *Modern operating systems*. Pearson, 4th edition, 2015.