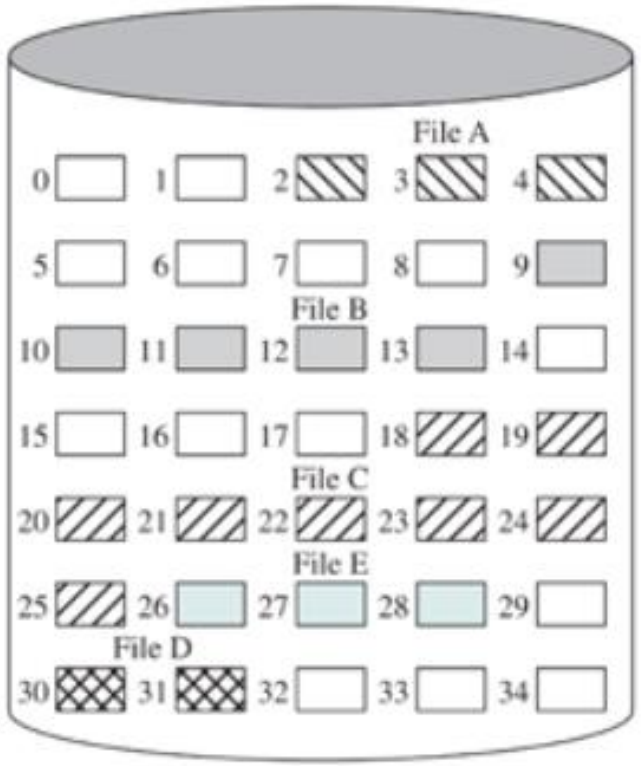
CS3841 – File Systems

Situation:

- User needs a mechanism to organize persistent data
- Disk represents data a contiguous array of sectors
- OS establishes a structure and abstraction of directories and files (file system)
- OS drivers translates user request (system calls) to access directories and files into I/O requests to external media

File Allocation - Contiguous

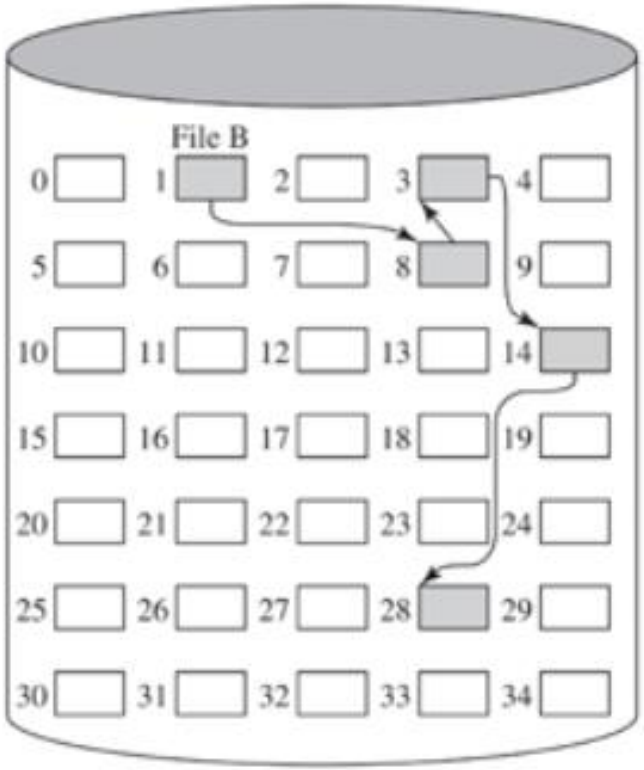


File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

- Sectors on disk are treated like an array
- Files are placed in contiguous free blocks
- File allocation table records starting block and length in blocks
- Advantage
  - Fast for reading files – next block is always in next sector
- Disadvantage
  - Can't increase file size
  - Prone to fragmentation

File Allocation - Chained

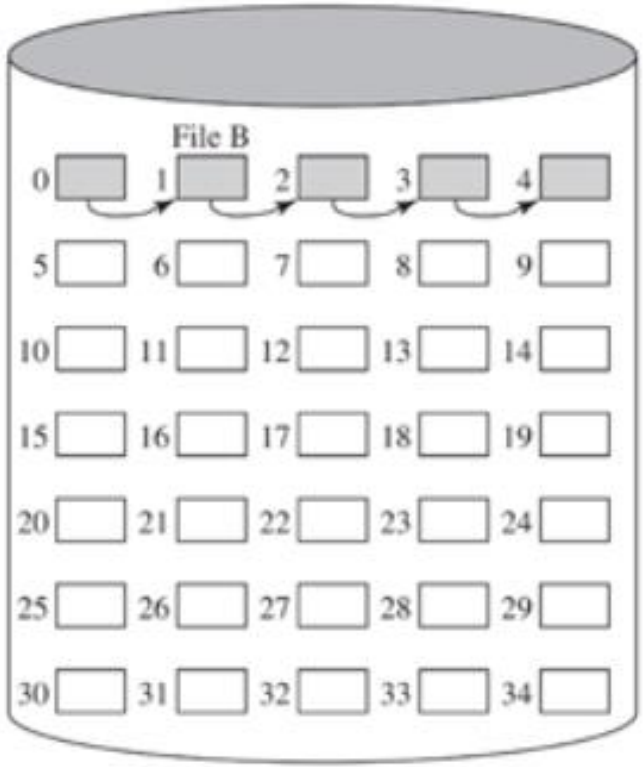


File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

- File sectors are treated like a large linked list
- Files are allocated in any free blocks
- A file block points to the next block in the file
- File allocation table records starting block and length in blocks
- Advantages
  - No fragmentation, any block can be used for the file
  - Can add/remove from beginning, middle, or end of file by just updating pointers
- Disadvantage
  - Must read the previous block before knowing the next block

File Allocation – Chained Consolidation/Defragmentation

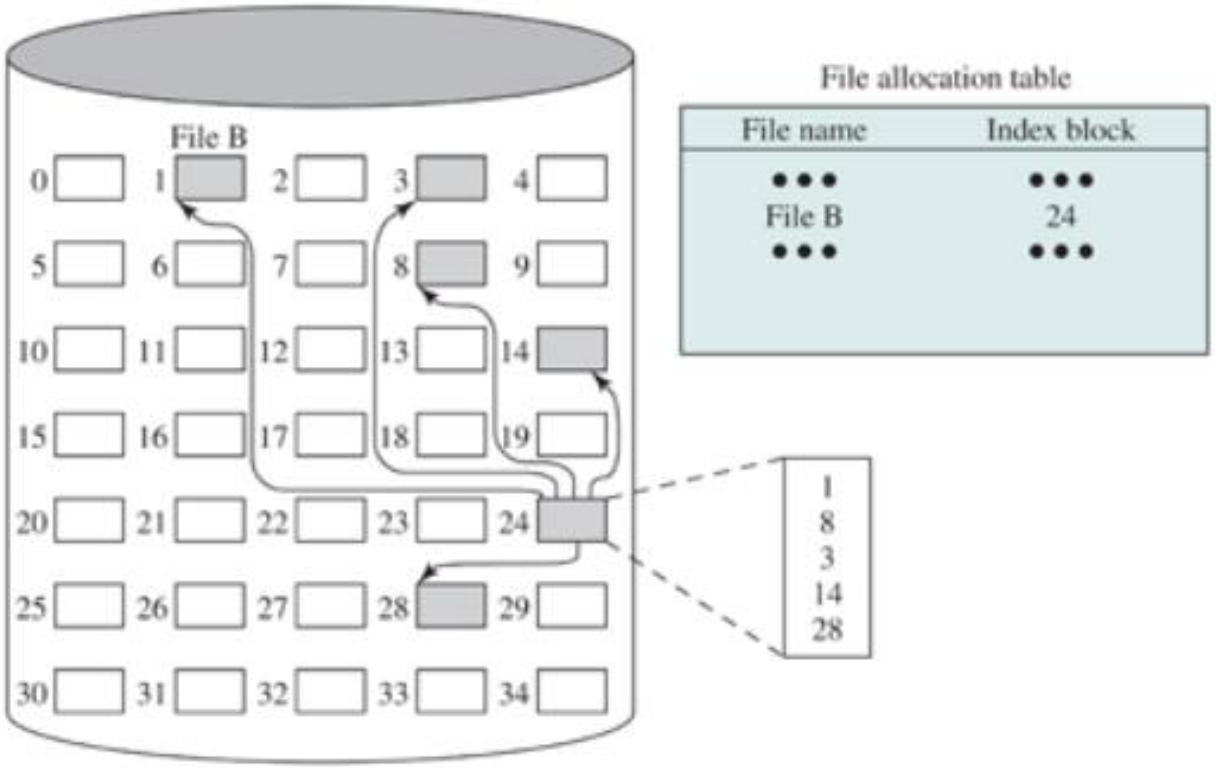


File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

- File sectors are reorganized so they are contiguous
- Advantages
  - Sectors are right next to each other so reads and writes are faster
- Disadvantage
  - Can't access the files while they are being reorganized

File Allocation - Indexed



- A single block is used to record all the blocks for the file
- Files are allocated in any free blocks
- File allocation table records the block of the index block
- Advantages
  - No fragmentation, any block can be used for the file
  - Can add/remove from beginning, middle, or end of file by just updating pointers
  - Only need to read the index block to find file data blocks
- Disadvantage
  - File size is limited to what fits in an index block

File Metadata

- What information do we want to store for a file?
  - File name
  - Type
  - Size
  - Permissions
  - Location
  - Data

File Allocation Table File System - FAT

- Uses chained and indexed allocation method
- Data blocks are divided into clusters
  - A cluster is a fixed # sectors, with a sector typically being 512 bytes

Region
Reserved Region (incl. Boot Sector)
File Allocation Table (FAT) * X
Root Directory
Data Region

FAT Reserved Region (AKA Boot Sector)

Offset	Description	Size
00h	Jump Code + NOP	3 Bytes
03h	OEM Name	8 Bytes
0Bh	Bytes Per Sector	1 Word
0Dh	Sectors Per Cluster	1 Byte
0Eh	Reserved Sectors	1 Word
10h	Number of Copies of FAT	1 Byte
11h	Maximum Root Directory Entries	1 Word
13h	Number of Sectors in Partition Smaller than 32MB	1 Word
15h	Media Descriptor (F8h for Hard Disks)	1 Byte
16h	Sectors Per FAT	1 Word
18h	Sectors Per Track	1 Word
1Ah	Number of Heads	1 Word
1Ch	Number of Hidden Sectors in Partition	1 Double Word
20h	Number of Sectors in Partition	1 Double Word
24h	Logical Drive Number of Partition	1 Word
26h	Extended Signature (29h)	1 Byte
27h	Serial Number of Partition	1 Double Word
2Bh	Volume Name of Partition	11 Bytes
36h	FAT Name (FAT16)	8 Bytes
3Eh	Executable Code	448 Bytes
1FEh	Executable Marker (55h AAh)	2 Bytes

File Allocation Table File System - FAT

- FAT #1 follows reserved section, FAT #2 follows FAT #1, etc.
- Each FAT occupies SECTORS_PER_FAT
- Each FAT entry corresponds to a cluster in the volume and denotes:
  - the cluster number of the next cluster in a chain
  - a special end of cluster-chain (EOC) entry that indicates the end of a chain
  - a special entry to mark a bad cluster
  - a zero to note that the cluster is unused

- FAT 16 = 2 bytes per FAT entry (cluster)
- FAT cluster entries 0 and 1 are special…
  - Cluster 0's entry = FAT ID
  - Cluster 1's entry = EOC marker (usually all 1s)
- Cluster #2 starts right after root directory

FAT Directory Entry

Offset	Length	Value
0	8 bytes	Name
8	3 bytes	Extension
11	byte	Attribute (00ARSHDV) 0: unused bit A: archive bit, R: read-only bit S: system bit D: directory bit V: volume bit
22	word	Time
24	word	Date
26	word	Starting Cluster
28	dword	File Size

FAT Directory Entry

Offset	Length	Value
0	8 bytes	Name
8	3 bytes	Extension
11	byte	Attribute (00ARSHDV) 0: unused bit A: archive bit, R: read-only bit S: system bit D: directory bit V: volume bit
22	word	Time
24	word	Date
26	word	Starting Cluster
28	dword	File Size

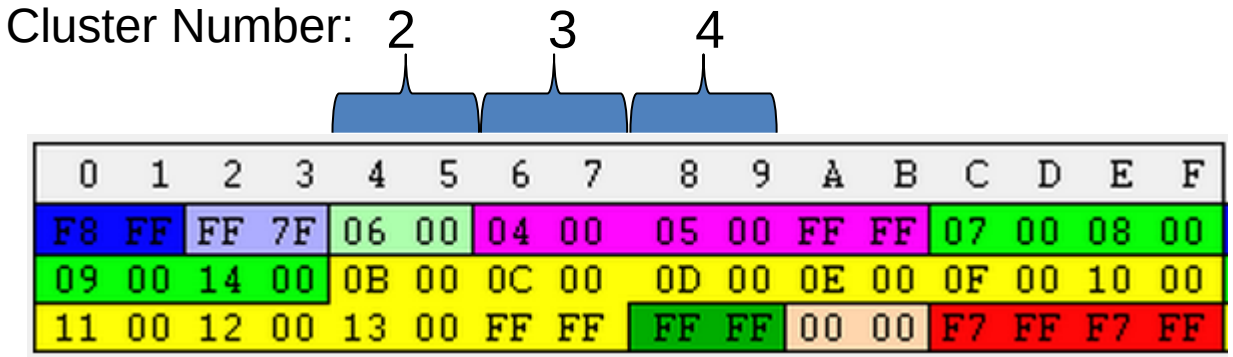
```
46 49 4c 45 31 20 20 20   54 58 54 20 00 13 7c 7b   |FILE1   TXT ..|{|
64 51 64 51 00 00 7c 7b   64 51 0b 00 06 00 00 00   |dQdQ..|{dQ......|
```

File Allocation Table File System - FAT

- File allocation table entry indicates the next cluster in the file or an indicator

- FAT entry special values:
  - 0x0000 – available cluster
  - 0x0001 – reserved / not used
  - 0x0002 to 0xFFEF – valid cluster next in chain value
  - 0xFFF0 – 0xFFF7 – various reserved and/or non-standard usages
  - 0xFFF8 – 0xFFFF – End of cluster-chain

- File starts at cluster 0x0002
  - Cluster chain is 0x0006, 0x0007, 0x0008, 0x0009, 0x0014



File Allocation Table File System - FAT

- File allocation table entry indicates the next cluster in the file or an indicator

- FAT entry special values:
  - 0x0000 – available cluster
  - 0x0001 – reserved / not used
  - 0x0002 to 0xFFEF – valid cluster next in chain value
  - 0xFFF0 – 0xFFF7 – various reserved and/or non-standard usages
  - 0xFFF8 – 0xFFFF – End of cluster-chain

- File starts at cluster 0x0002
  - Cluster chain is 0x0006, 0x0007, 0x0008, 0x0009, 0x0014