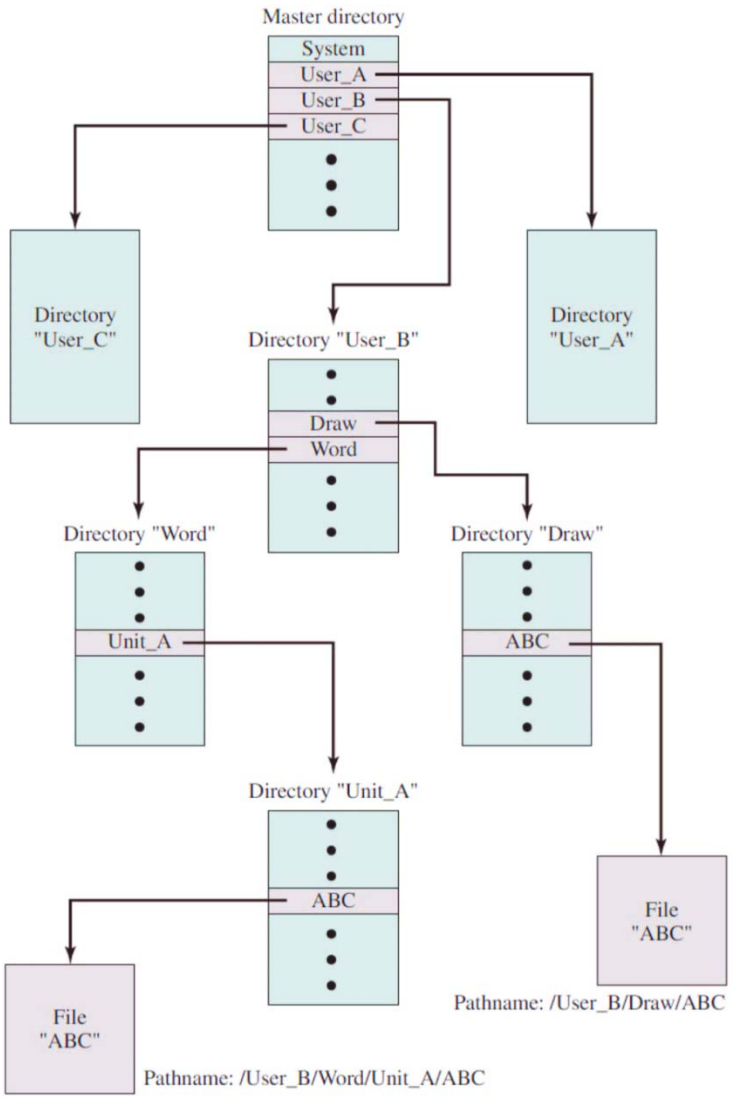
CSE 306/506 Operating Systems
File Management

YoungMin Kwon

File Directories



- Directory itself is a file
- A directory has a list of file name and its location information
- A process has a current directory (aka working directory)

File Sharing

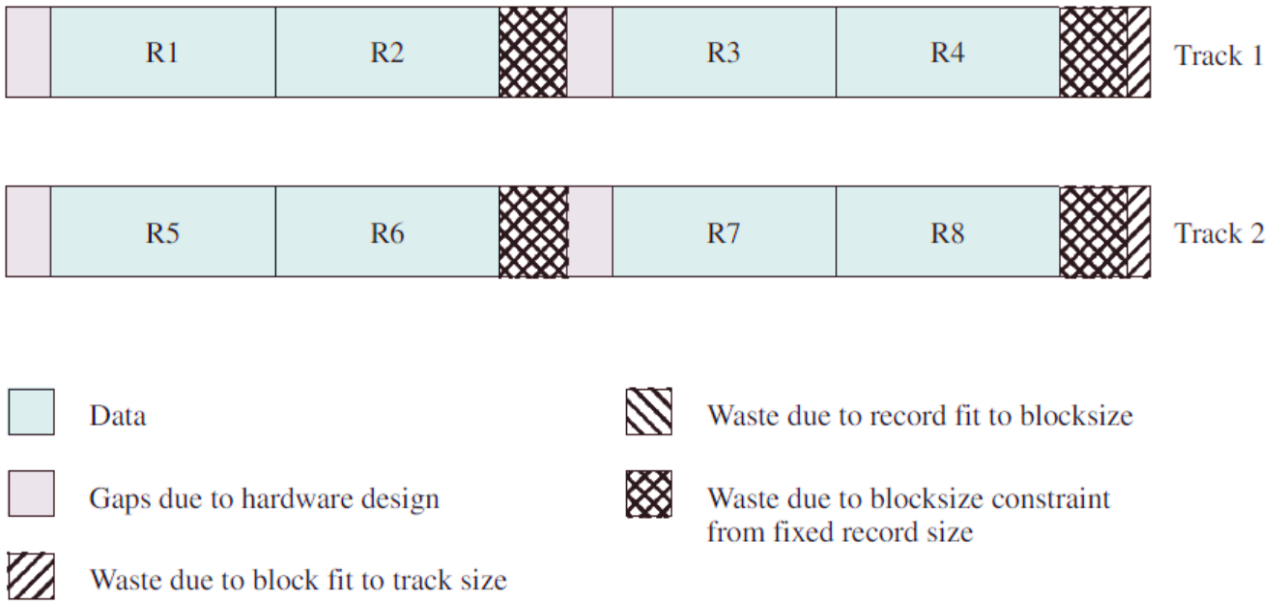
- Access rights
  - None: users may not know the existence of the file
  - Knowledge: users know the existence of the file and its owner
  - Execution: users can load and execute a program but cannot copy it
  - Reading: users can read the for copying and execution
  - Appending: users can add data at the end of the file, but cannot modify the existing contents
  - Updating: users can modify the contents of the file
  - Changing protection: users can change the access right
  - Deletion: users can delete the file

Record Blocking

- Records must be organized as blocks
  - Records are the logical unit of access of a structured file
  - Blocks are the unit of I/O with secondary storage
- Fixed or Variable size block
  - On most systems, blocks are of fixed length
- Size of a block
  - Large block: more records per an I/O operation
    - Good for sequential access
    - Not good for random access (unnecessary read/write)

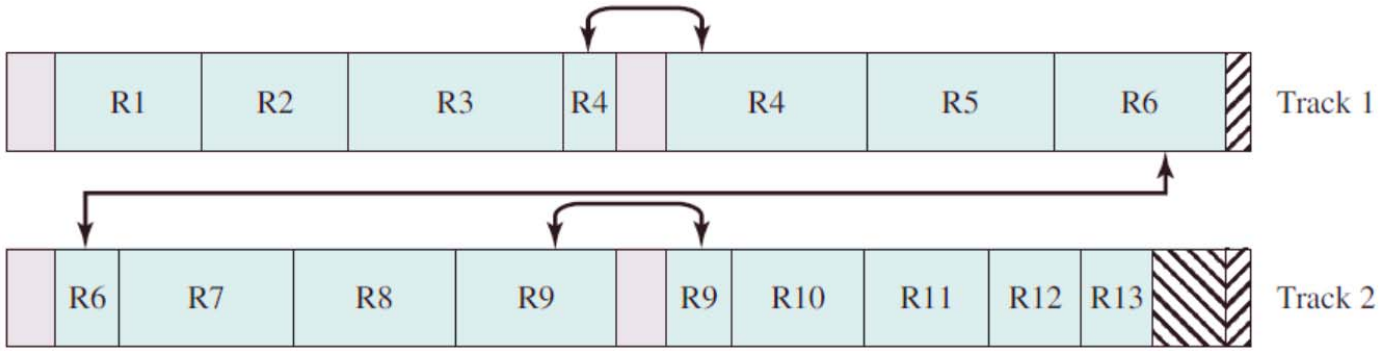
Record Blocking

- Fixed blocking
  - Integral number of fixed-length records in a block
  - May have unused space at the end of each block



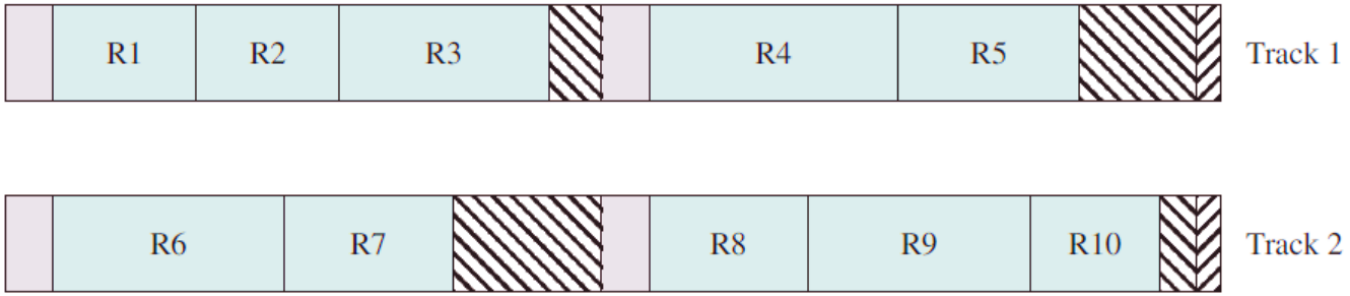
Record Blocking

- Variable-length spanned blocking
  - Variable length records are packed in a block
  - Some records may span two blocks
    - A pointer to the successor block
    - No unused space



Record Blocking

- Variable-length unspanned blocking
  - Variable length records are used
  - Spanning is not employed



Secondary Storage Management

- File allocation
  - Should allocate space in advance at once when files are created?
  - Space is allocated to a file as one or more contiguous units (portion)
    - What size of portion?
  - What sort of data structure to use to keep track of portions?
    - E.g. File Allocation Table (FAT) in DOS

Preallocation vs Dynamic Allocation

- Preallocation
  - Need the maximum file size at the time of creation
    - Program compilation, Summary data, Network file transfer
  - In general, it is difficult to know the maximum
    - Users tend to overestimate $\rightarrow$ waste of space
- Dynamic allocation
  - Allocate space to a file in portions as needed

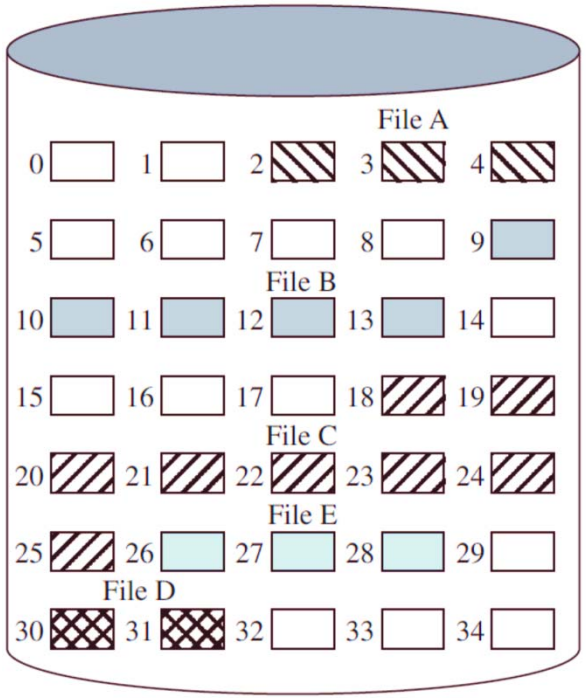
Portion Size

- Small portion vs large portion trade-offs
  - Contiguity of space increases performance
  - Large number of small portions → increase the size of file allocation table
  - Fixed size portions simplify the reallocation of space

- Two major alternatives
  - Variable, large contiguous portions
    - Better performance, no waste of space, small file allocation table
    - Space is hard to reuse
  - Blocks
    - Greater flexibility
    - Large table, contiguity is abandoned, blocks are allocated as needed

File Allocation Methods

- Contiguous allocation
  - A single contiguous set of blocks is allocated to a file at the time of creation



File allocation table

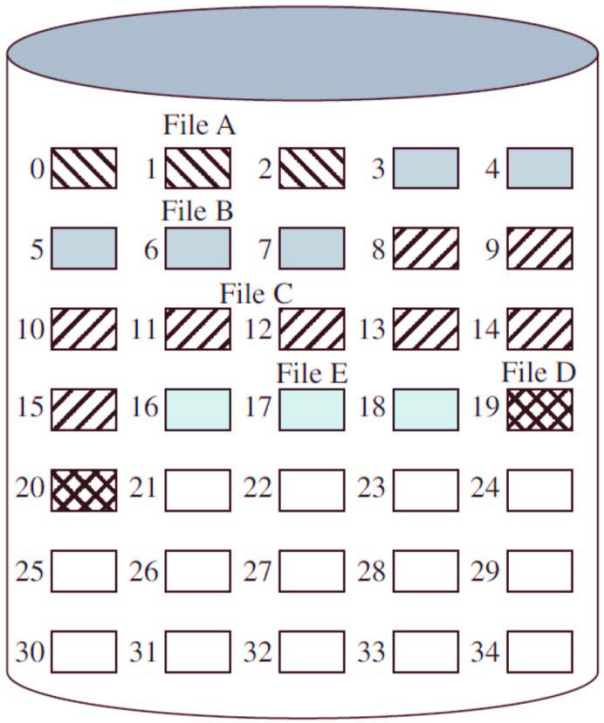
File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

File Allocation Methods

- Contiguous allocation
  - Preallocation strategy
  - Variable-size portions
  - File allocation table needs a single entry for each file (starting block, length)
  - Performs better for sequential accesses: multiple blocks can be read in at a time
  - External fragmentation will occur (need compaction)
  - Need to declare file size at the file creation time

File Allocation Methods

- Contiguous allocation (after compaction)

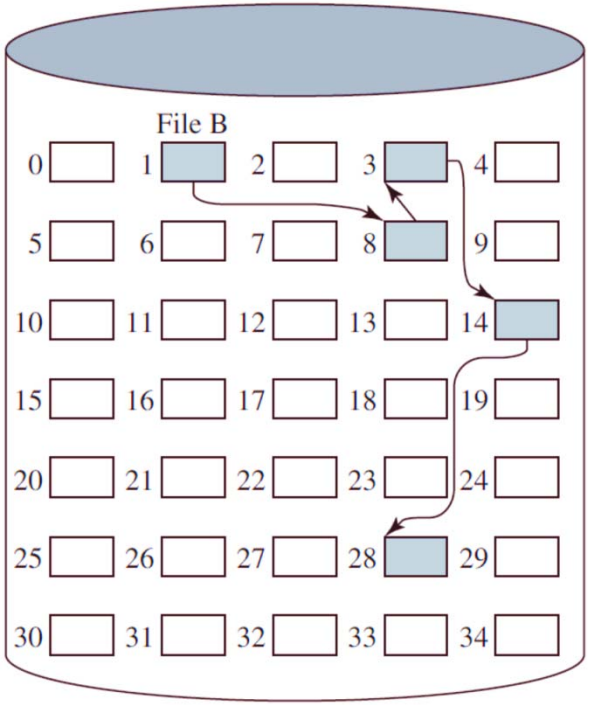


File allocation table

File name	Start block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

File Allocation Methods

- Chained allocation
  - Each block contains a pointer to the next block



File allocation table

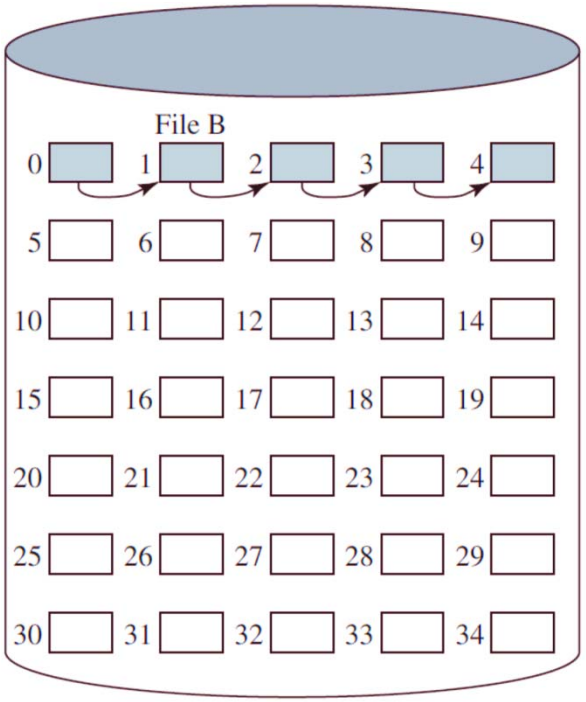
File name	Start block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

File Allocation Methods

- Chained allocation
  - Allocation is on an individual block basis
  - File allocation table needs a single entry for each file (starting block, length of the file)
  - Any free block can be allocated as needed
  - No external fragmentation
  - Performs well with sequential files with sequential access
    - Need to trace through chains to select a specific block
  - For locality, periodically consolidate files

File Allocation Methods

- Chained allocation (after consolidation)

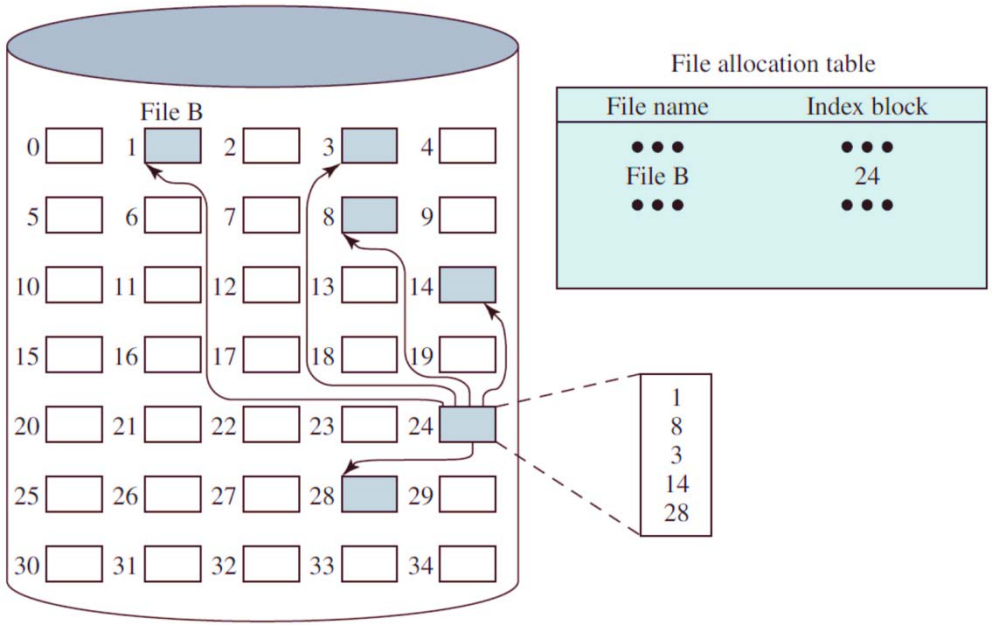


File allocation table

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

File Allocation Methods

- Indexed allocation
  - File allocation table contains a separate one-level index for each file

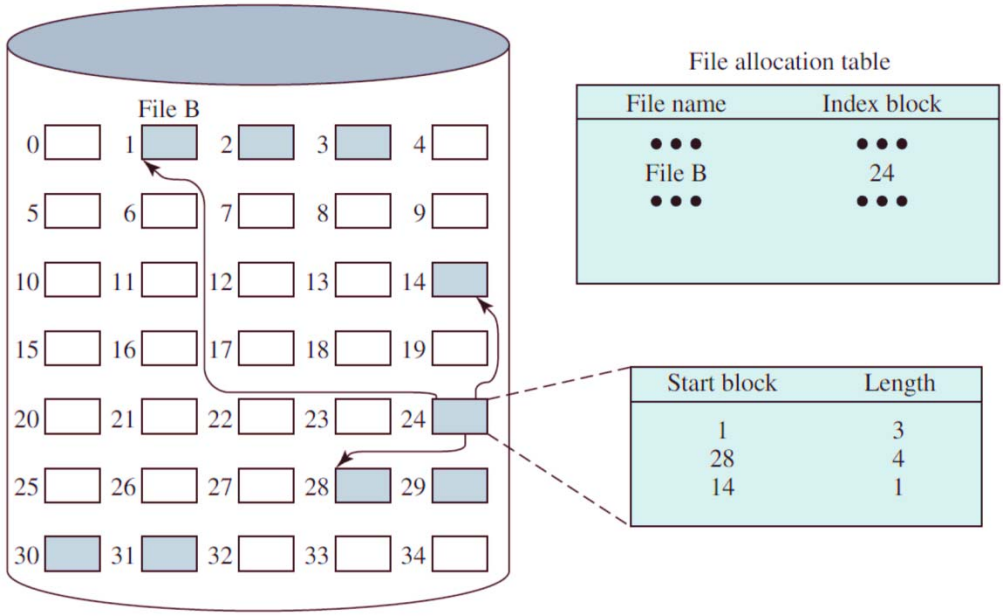


File Allocation Methods

- Indexed allocation
  - Typically, file indexes for a file are kept in a separate block
  - File allocation table has an entry pointing to this index block
  - Allocation can be on the basis of fixed-size blocks or variable-size portions
  - File consolidation can be done from time to time
  - Indexed allocation supports both sequential access and direct access to the file

File Allocation Methods

- Indexed allocation with variable-length portions



Free Space Management

- Bit table
  - Uses a vector containing one bit for each block
    - 0 means a free block, 1 means an allocated block
    - E.g. 00111000011111000011111111111011000
  - Easy to find a contiguous group of free blocks
  - Small in size: (disk size)/(8 x block size)

Free Space Management

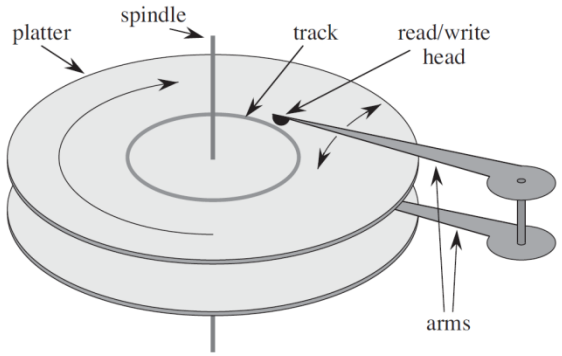
- Chained free portions
  - Free portions are chained by using a pointer and length
  - Negligible space overhead
  - Inefficient when the disk is quite fragmented
  - Inefficient when deleting a highly fragmented file

- Indexing
  - Treat free spaces as a file and use an index table for them

Free Space Management

- Free block list
  - A block number for each free block is maintained in a reserved portion of the disk
  - Stack can be used as a data structure
    - Allocation is done by popping from the stack
    - Deallocation is done by pushing to the stack
    - Only certain amount the stack top is maintained in the main memory
  - FIFO queue can be used also
    - Allocation is done by removing from the queue head
    - Deallocation is done by adding to the tail of the queue
    - Only certain amount of head and tail area is maintained in the main memory

Disk Scheduling Policies



- Goal: reduce the seek time
- First-In-First-Out
  - Process items in the queue in the sequential order
- Priority
  - Schedule based on the system priority (in a batch system, short jobs have higher priority than longer ones)

Disk Scheduling Policies

- Last-In-First-Out
  - Recent requests have the locality
- Shortest-Service-Time-First
  - Select the request that will move the arm the least
- SCAN
  - Except for FIFO, all scheduling algorithms have a starvation problem
  - With SCAN, the arm is required to move in one direction only until it reaches the last track or there are no more requests in that direction

Disk Scheduling Policies

- C-SCAN (Circular SCAN)
  - Restricts the scanning to one direction only
  - When the arm reaches to the last track, it returns to the opposite end and begins the scan again
  - Reduces the maximum delay

Disk Scheduling Policies

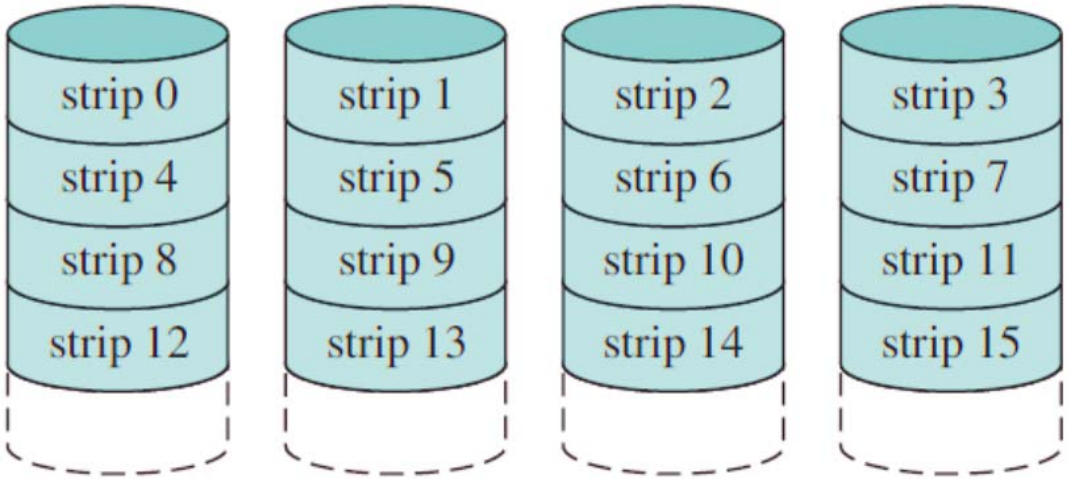
(a) FIFO (starting at track 100)		**(b) SSTF** (starting at track 100)		**(c) SCAN** (starting at track 100, in the direction of increasing track number)		**(d) C-SCAN** (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

RAID

- RAID (Redundant Array of Independent Disks)
  - A set of physical disk drives viewed as a single logical drive
  - Data are distributed across physical drives
  - Redundant disk capacity is used to store parity information (data recoverability)

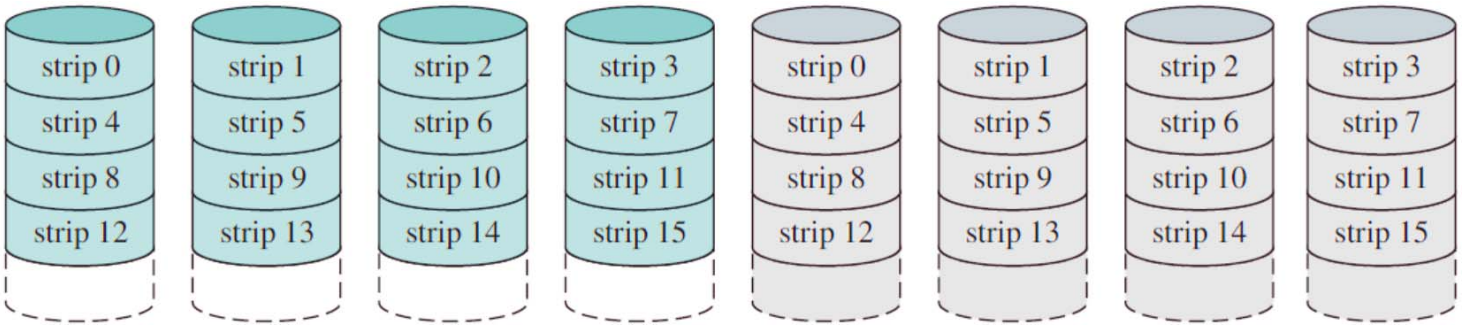
RAID Level 0

- No redundancy
- Data are striped across the available disks
- A unit of data in the logical disk (stripe) are mapped to consecutive physical disks



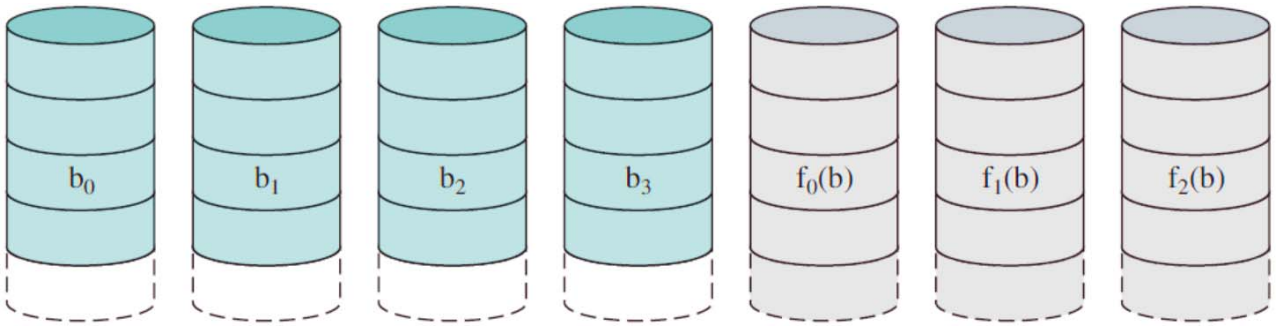
RAID Level 1

- Redundancy is achieved by duplicating data
- Read request
  - Served by the faster of the two (smaller seek time + rotational delay)
- Write performance
  - Dictated by the slower of the two



RAID Level 2

- For RAID 2 and RAID 3, spindles of the individual drives are synchronized
- Redundancy: Hamming code
  - Can correct 1-bit error, can detect 2-bit errors

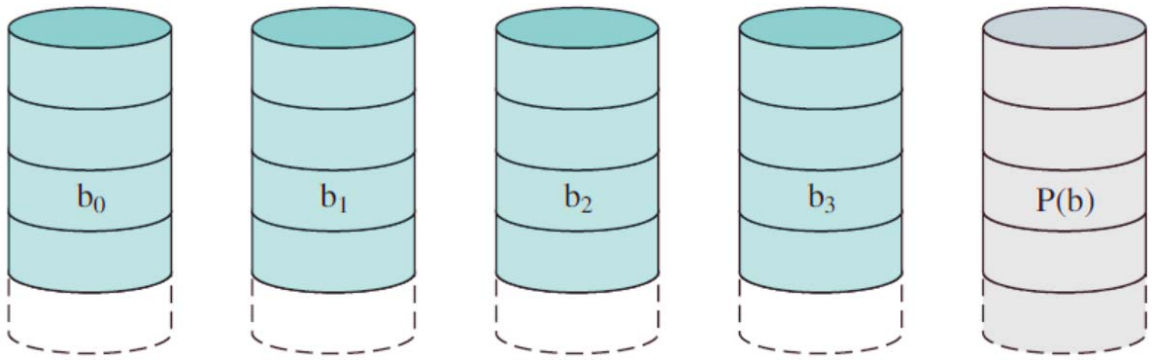


RAID Level 3

- A single redundancy disk is used
  - Can detect 1-bit error, no error correction
  - Can recover from a disk drive failure

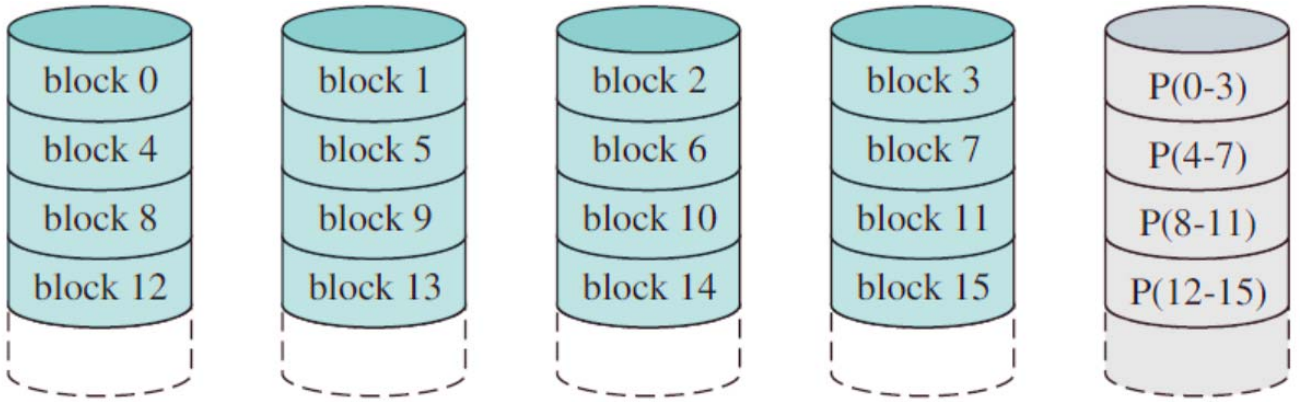
- Redundancy: Parity
  - E.g. $X_0 \sim X_3$ data disk, X_4 parity disk, $\oplus$: exclusive or
    $$X_4(i) = X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i)$$

  - Suppose that drive X_1 has failed
    $$X_1(i) = X_4(i) \oplus X_3(i) \oplus X_2(i) \oplus X_0(i)$$



RAID Level 4

- For RAID 4 to RAID 6, disks are not synchronized
  - Separate I/O requests can be satisfied in parallel
  - More suitable for high I/O requests than high data transfer rates



RAID Level 4

- RAID 4 penalizes small writes
  - E.g. Initial configuration

    $$X_4(i) = X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i)$$

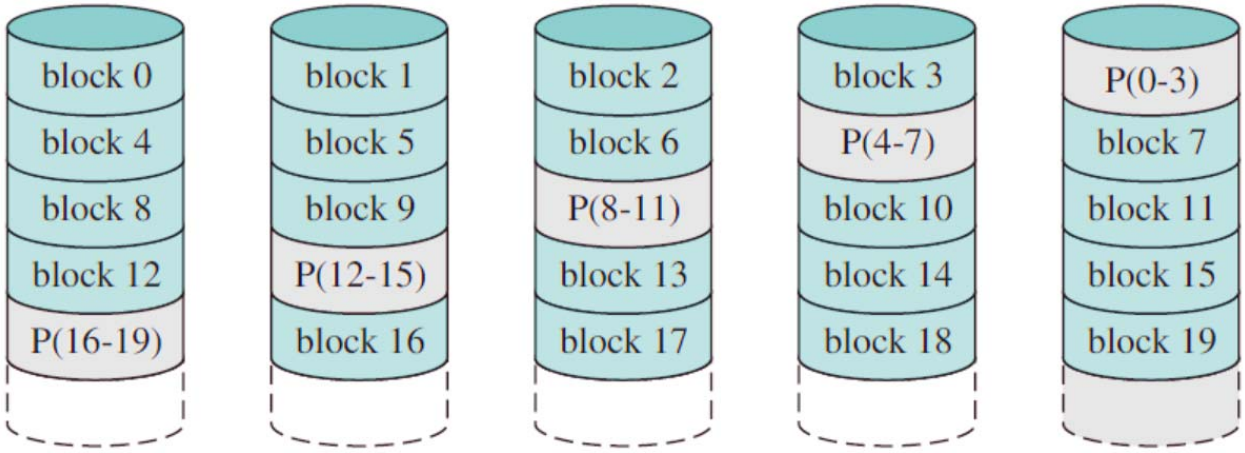
  - When $X_1(i)$ is modified to $X_1(i)'$

    $$\begin{aligned} X_4(i)' &= X_3(i) \oplus X_2(i) \oplus X_1(i)' \oplus X_0(i) \\ &= X_3(i) \oplus X_2(i) \oplus X_1(i)' \oplus X_0(i) \oplus X_1(i) \oplus X_1(i) \\ &= X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i) \oplus X_1(i) \oplus X_1(i)' \\ &= X_4(i) \oplus X_1(i) \oplus X_1(i)' \end{aligned}$$

  - Write involves 4 disk access
    - Read old data and parity
    - Write new data and parity

RAID Level 5

- In RAID 4, the parity disk drive can be a bottleneck
- In RAID 5, distribute the parity data across all disk drives
  - The potential I/O bottleneck issue can be fixed



RAID Level 6

- 2 different parity calculations (P, Q) are used
  - Can recover from 2 disk drive failures
  - Extremely high data availability
  - Incurs a substantial write penalty