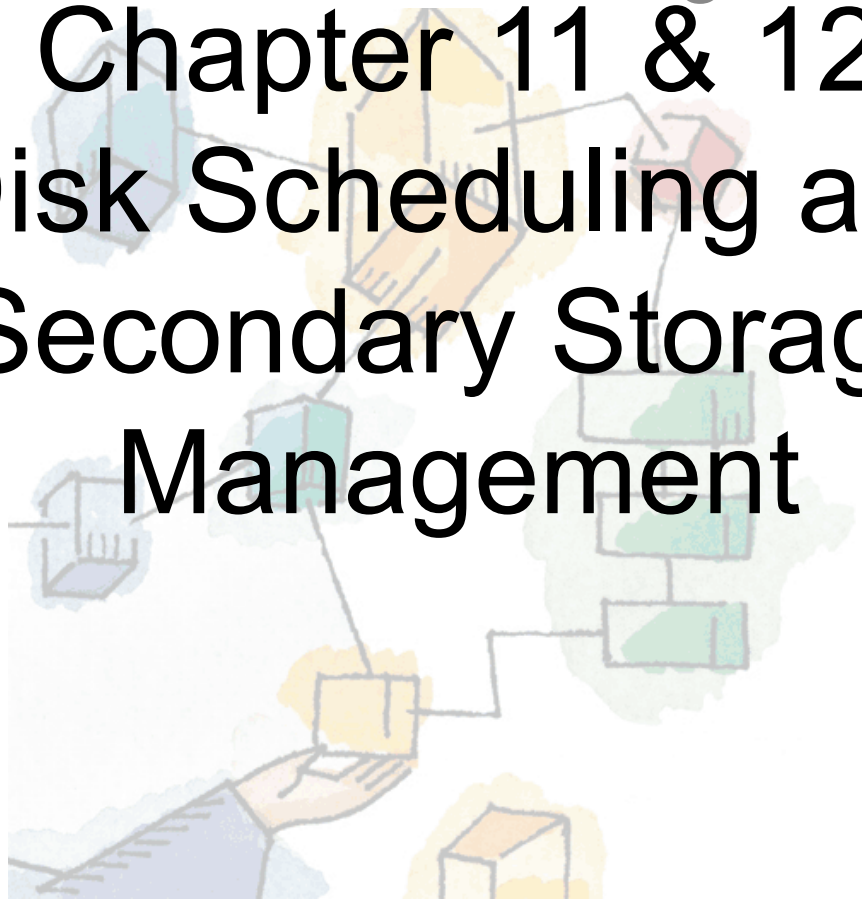
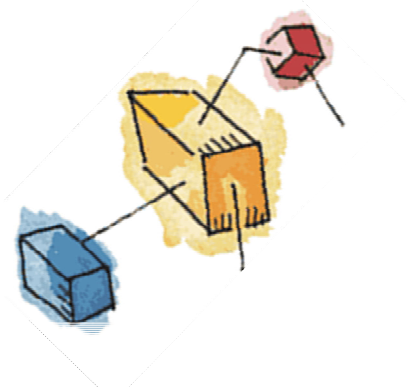
Operating Systems:
Internals and Design Principles
William Stallings

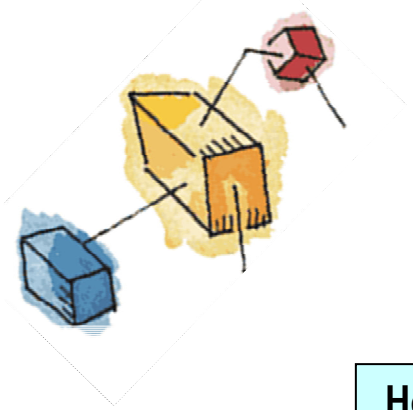
Chapter 11 & 12
Disk Scheduling and Secondary Storage Management



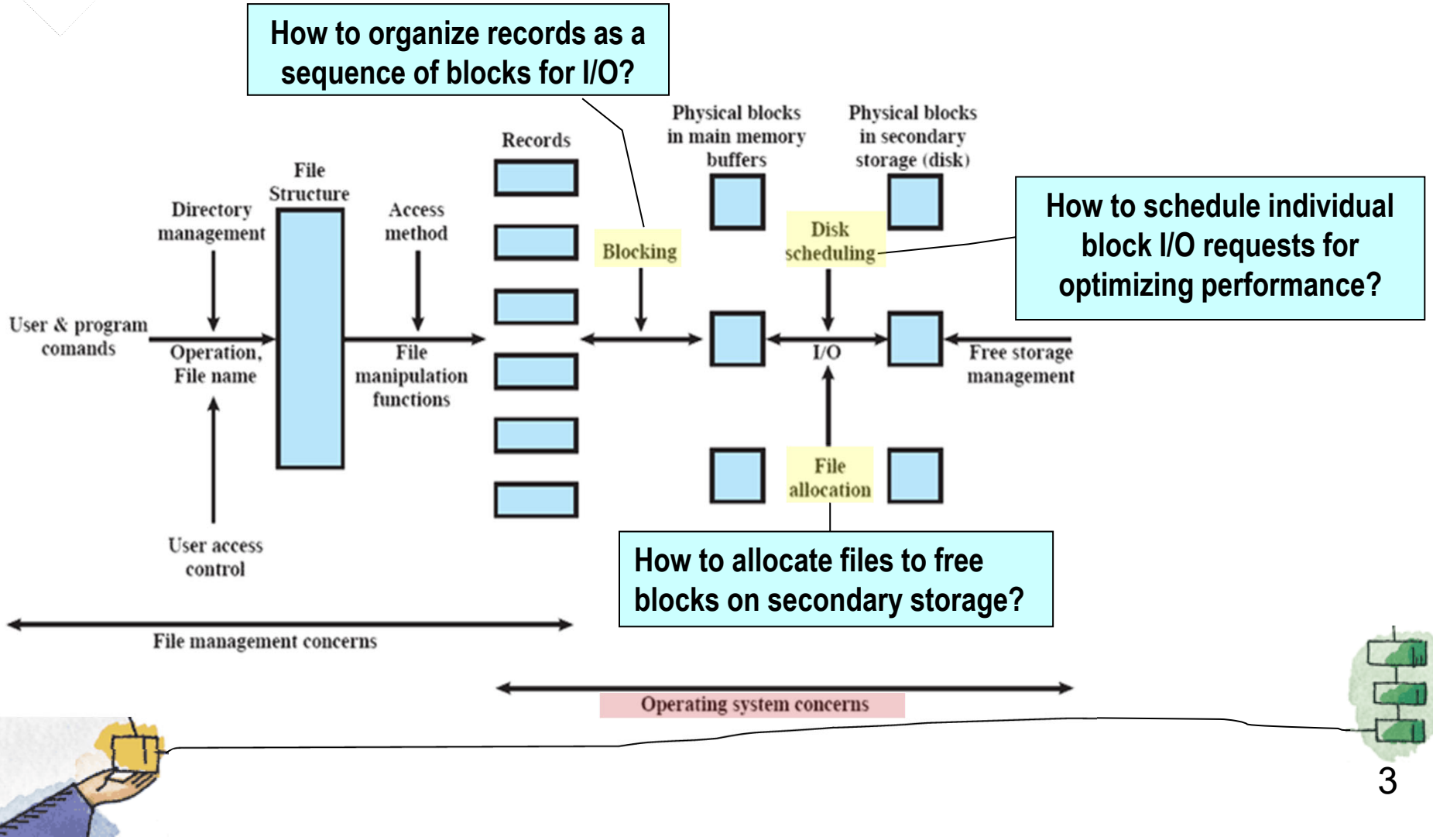


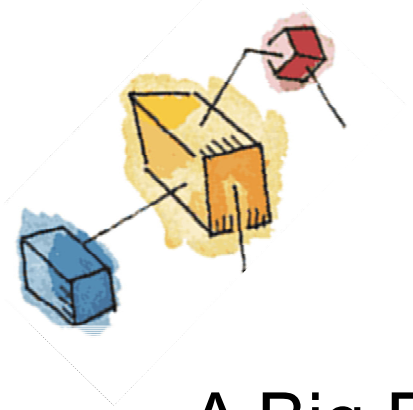
Roadmap

- A Big Picture
- Record Blocking
- Disk Scheduling
- File Allocation



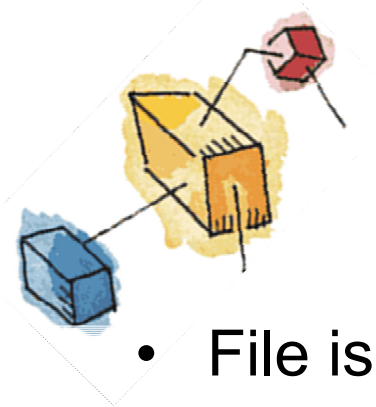
A Big Picture of File Management





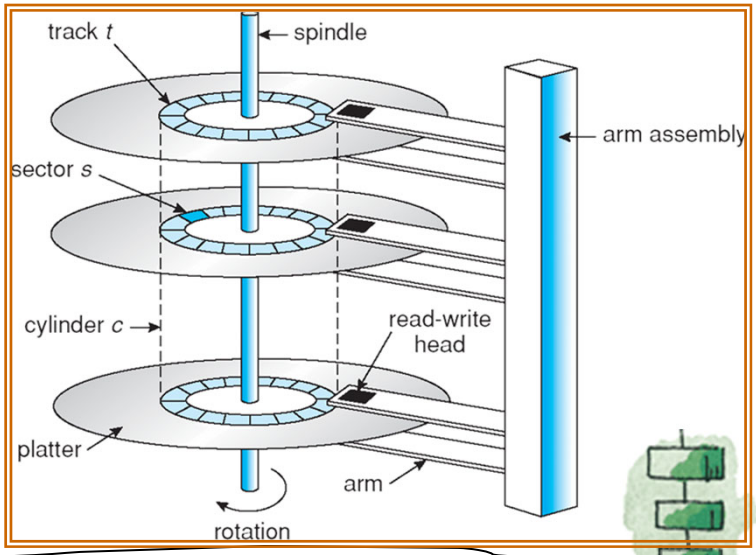
Roadmap

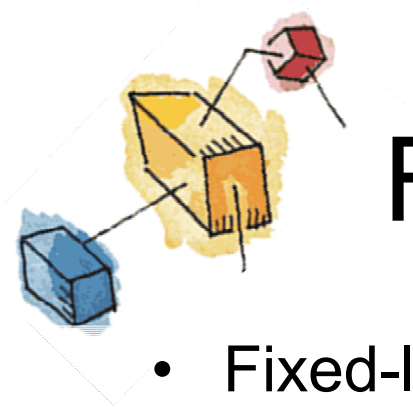
- A Big Picture
- Record Blocking
- Disk Scheduling
- File Allocation



File, Record and Block

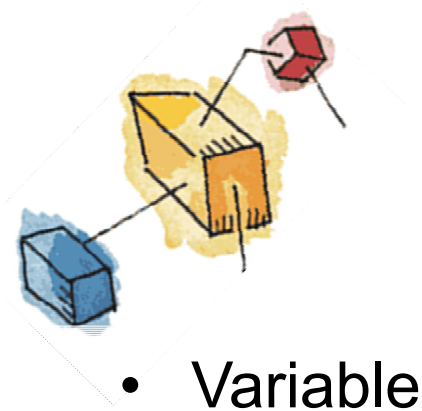
- File is a collection of similar records
- Record can be treated as a unit by application programs
  - fixed or variable length
- Blocks are the unit for I/O with secondary storage
  - Blocks are mapped onto sectors of the disk sequentially
- For I/O to be performed, records must be organized as blocks
- Three approaches are common
  - Fixed-length blocking
  - Variable-length spanned blocking
  - Variable-length unspanned blocking





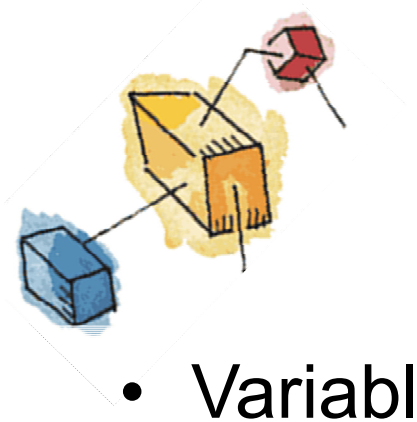
Fixed-Length Blocking

- Fixed-length records are used, and an integral number of records are stored in a block
- 👎 Unused space at the end of a block is ***internal fragmentation***



Variable-Length Spanned Blocking

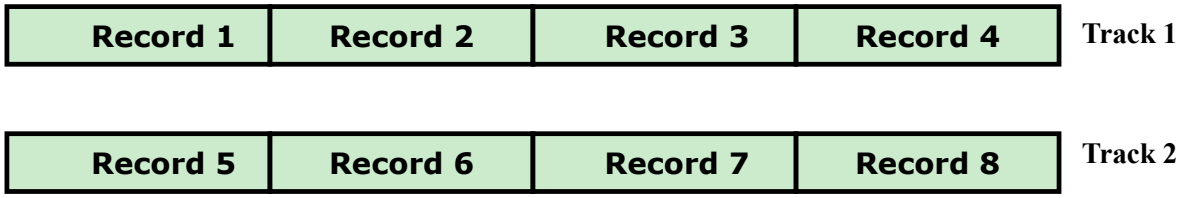
- Variable-length records are used and are packed into blocks with **no** unused space
- Some records may span multiple blocks
  - Continuation is indicated by a pointer to the successor block
- 👍 Efficient for storage
- 👍 Does not limit the size of records
- 👎 Difficult to implement
- 👎 Records that span two blocks require two I/O operations



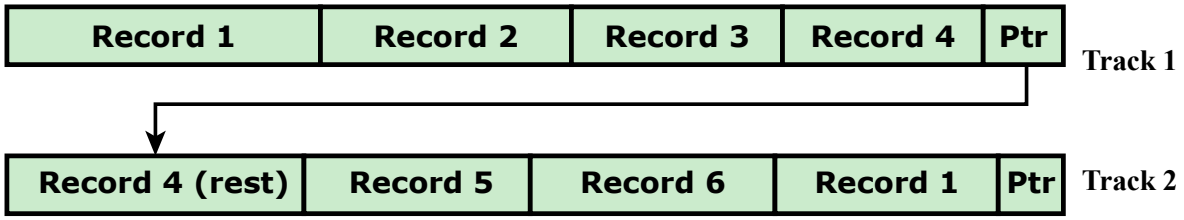
Variable-Length Unspanned Blocking

- Variable length records are used but no spanning
- 👎 Wasted space in most blocks because of the remainder of a block cannot be used if the next record is larger than the remaining unused space
- 👎 Limits record size to the size of a block

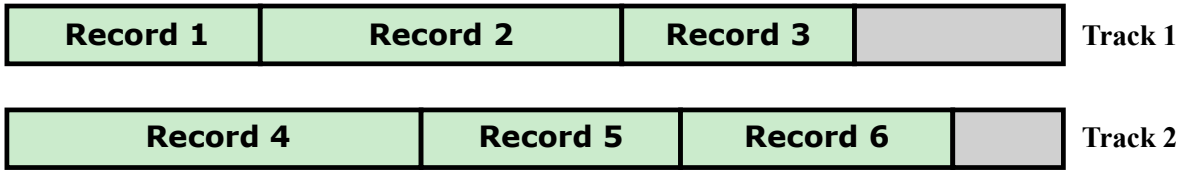
Record Blocking Methods



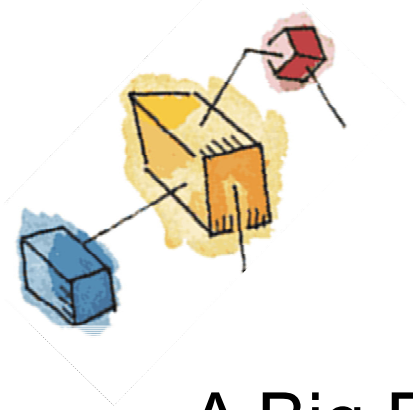
(a) Fixed Blocking



(b) Variable Blocking: Spanned

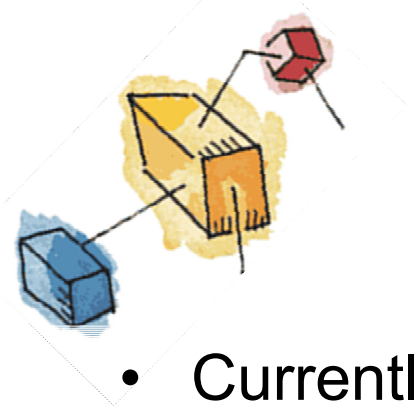


(c) Variable Blocking: Unspanned



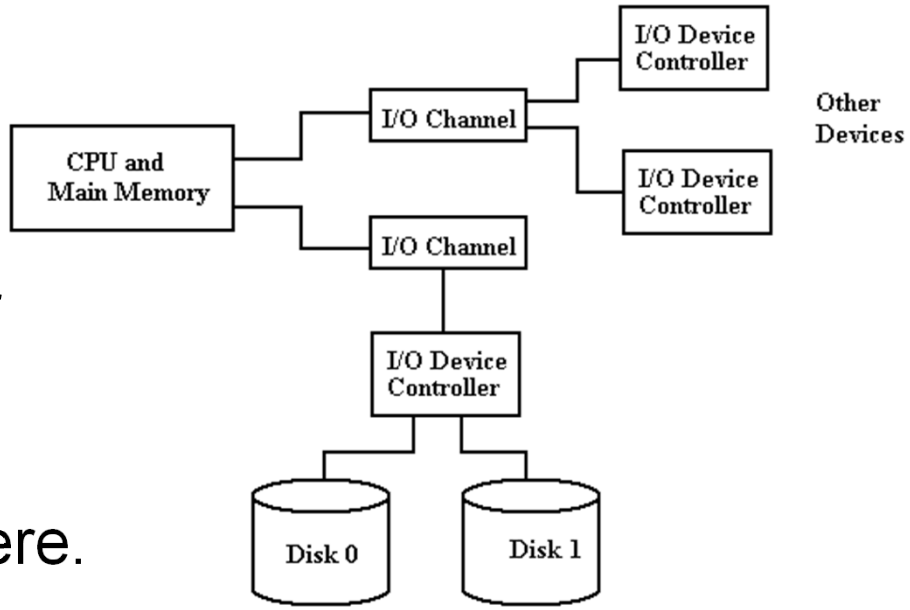
Roadmap

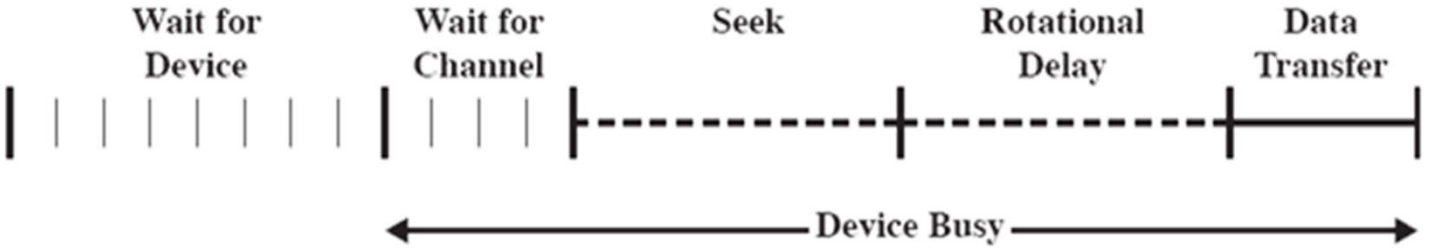
- A Big Picture
- Record Blocking
- Disk Scheduling
- File Allocation

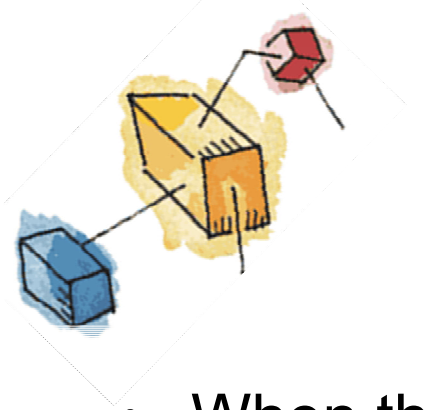


Disk Performance Parameters

- Currently, disks are at least four orders of magnitude slower than main memory
  → performance of disk storage subsystem is of vital concern
- A general timing diagram of disk I/O transfer is shown here.

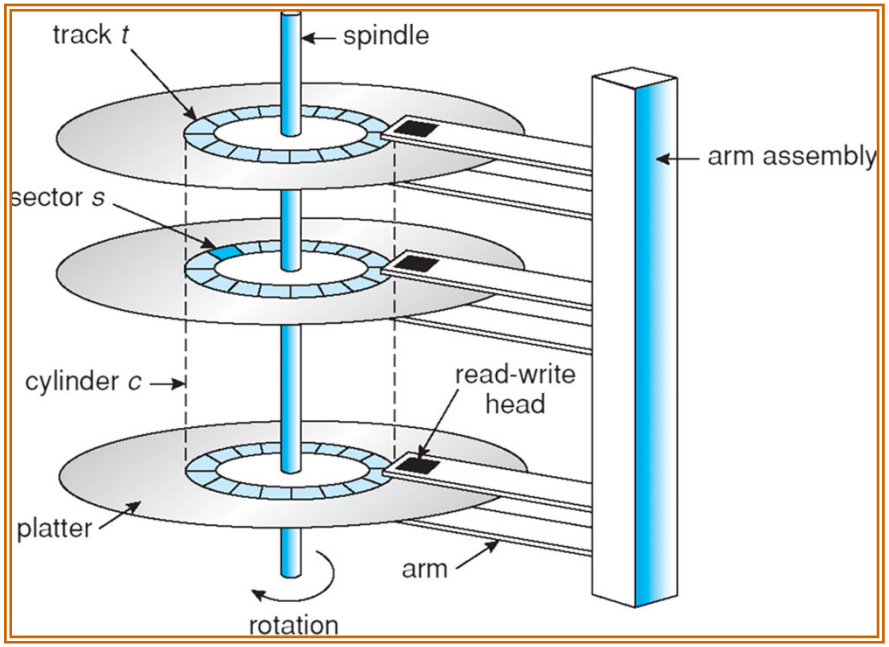






Positioning the Read/Write Heads

- When the disk drive is operating, the disk is rotating at **constant** speed
- To read or write, the ***head*** must be positioned at the desired ***track*** and at the beginning of the desired ***sector*** on that track



Disk Performance Parameters

- ***Seek time***: the time it takes to position the head at (move the disk arm to) the desired track
- ***Rotational delay*** or ***rotational latency***: the time it takes for the beginning of the desired sector to reach the head
- ***Transfer Time*** is the time taken to transfer the data (as the sector moves under the head)

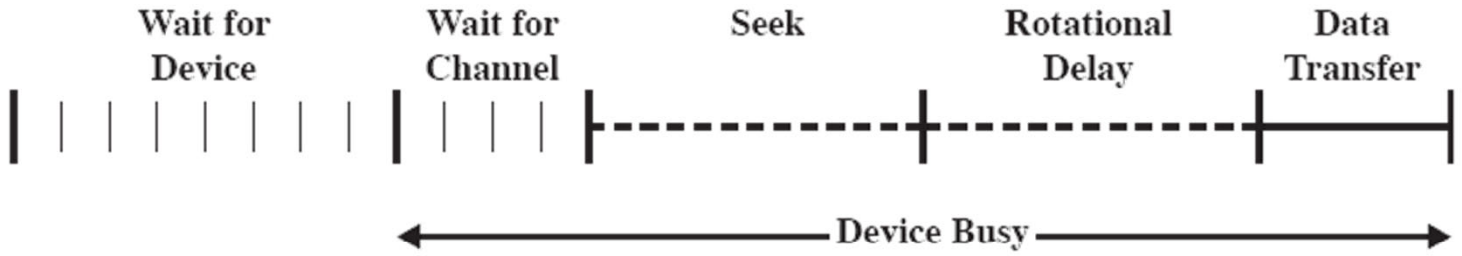
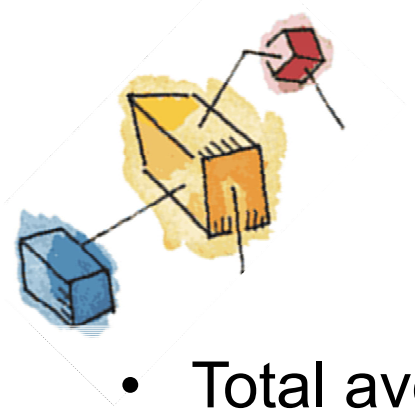


Figure 11.6 Timing of a Disk I/O Transfer



Disk Performance Parameters

- Total average access time T_a

$$T_a = T_s + 1 / (2r) + b / (rN)$$

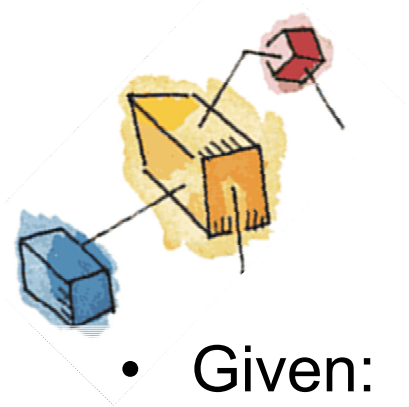
where T_s = average seek time

b = no. of bytes to be transferred

N = no. of bytes on a track

r = rotation speed, in revolutions / sec.

- Due to the seek time, the order in which sectors are read from disk has a tremendous effect on I/O performance

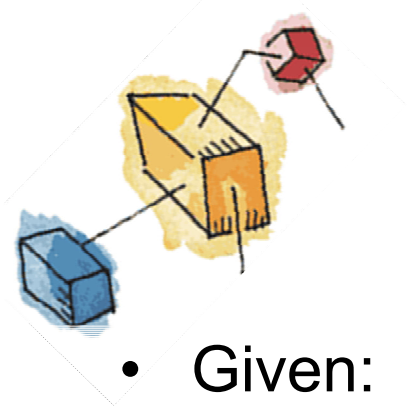


Disk Performance Example

- Given:
  - Average seek time = 5ms
  - Rotation speed = 7500rpm
    - Time for one rotation = 1/(7500/60) = 8 ms
  - 500 sectors / track
  - Read a file consisting of 2,500 sectors
- Total time for **sequential** access (the file occupies all the sectors on 5 adjacent tracks)

  = time to read 1st track + time to read successive tracks

  = 5 + 4 + 8 + 4 * (4 + 8) = 65 ms

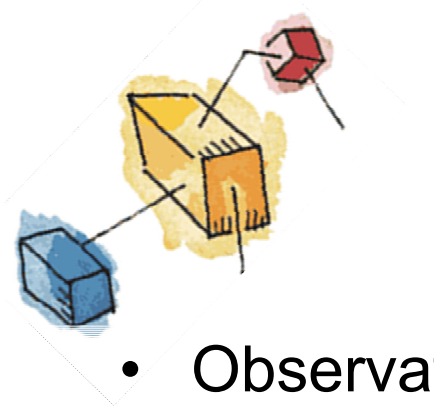


Disk Performance Example (cont.)

- Given:
  - Average seek time = 5ms
  - Rotation speed = 7500rpm
    - Time for one rotation = 1/(7500/60) = 8 ms
  - 500 sectors / track
  - Read a file consisting of 2,500 sectors
- Total time for **random** access (sectors of the file are distributed randomly over the disk)

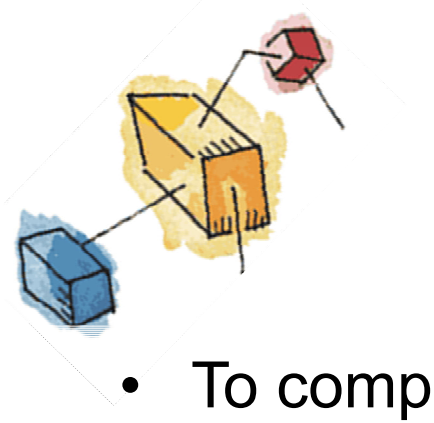
  = 2500 * time to read a sector

  = 2500 * (5 + 4 + 0.016) = 22,540 ms



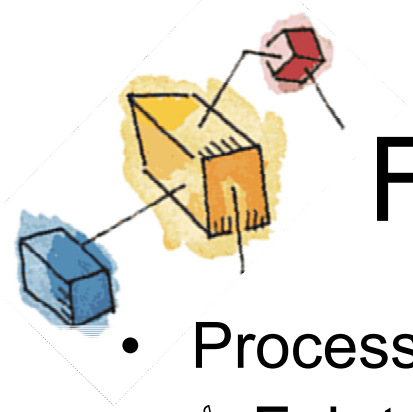
Disk Performance

- Observations:
  - The reason for the difference in performance can be traced to **seek time**.
  - In a multiprogramming environment, the OS maintains a queue of requests for each disk from various processes.
  - If requests (tracks) are selected (visited) at random, the performance will be very poor.
- So, there is a need to schedule access requests smartly in order to reduce the average time spent on seeks.



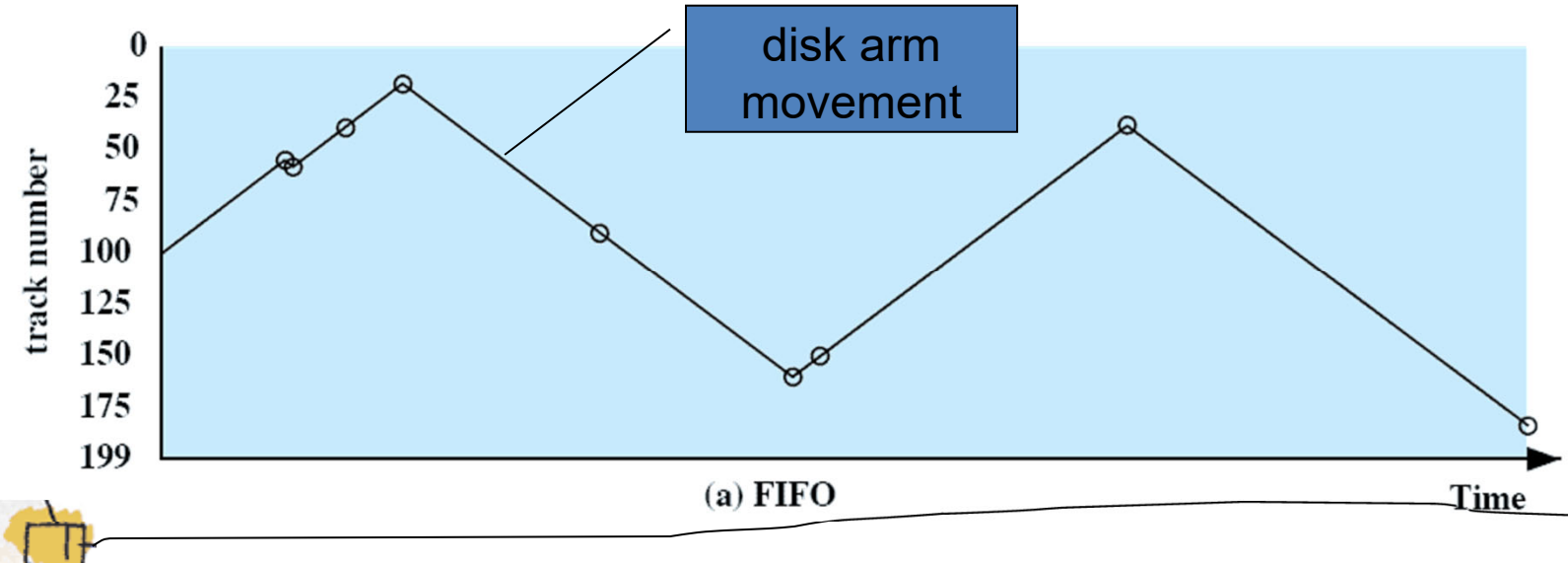
Disk Scheduling Policies

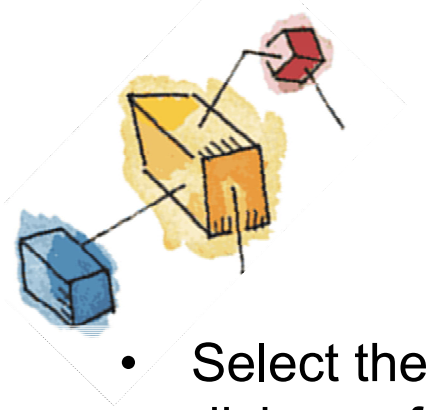
- To compare various schemes, consider a disk head initially located at track 100
  - assume a disk with 200 tracks and that the disk request queue has **random** requests from **various** processes in it
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184



First-in, first-out (FIFO)

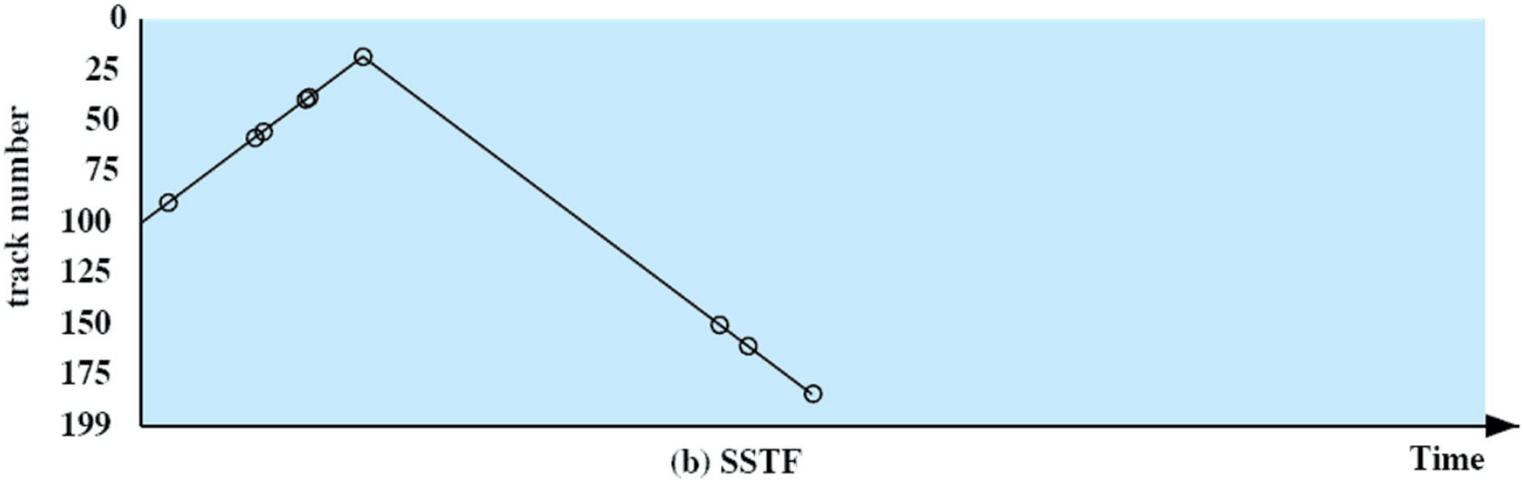
- Process requests sequentially
- 👍 Fair to all processes
- 👍 May have good performance if most requests are to clustered file sectors
- 👎 Approximate random scheduling in performance if there are many processes

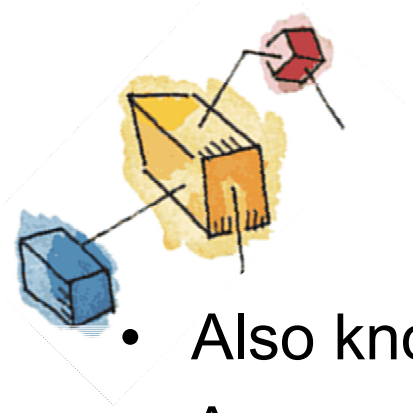




Shortest Service Time First (SSTF)

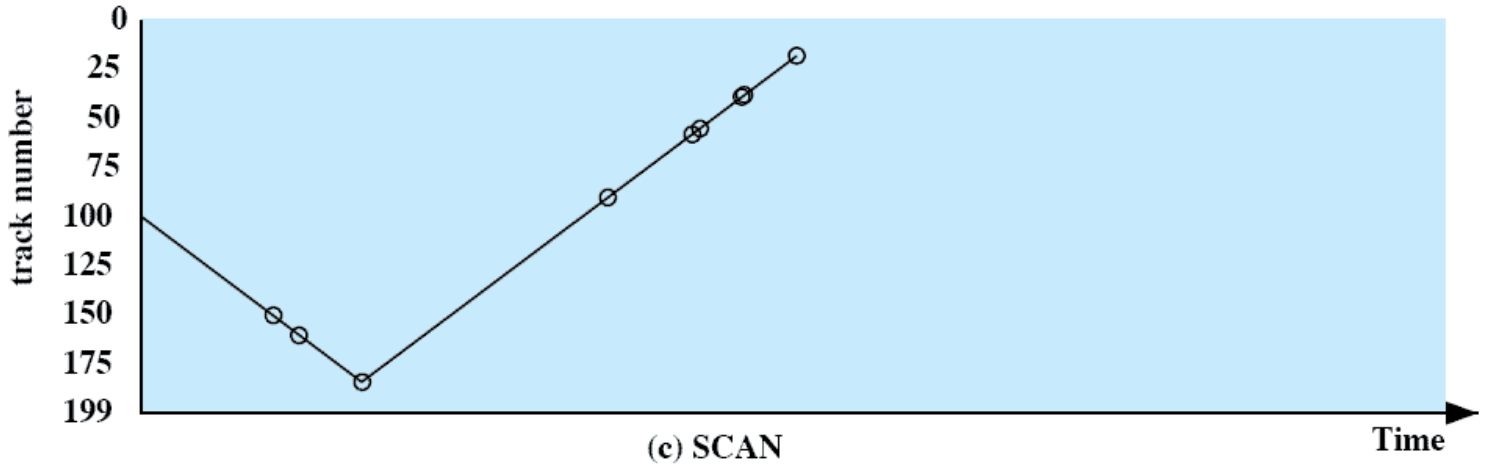
- Select the disk I/O request that requires the **least** movement of the disk arm from its current position
- 👍 Always choose the minimum seek time → better performance
- 👎 Does not ensure fairness
- 👎 Starvation may occur when new requests are always chosen before an existing request



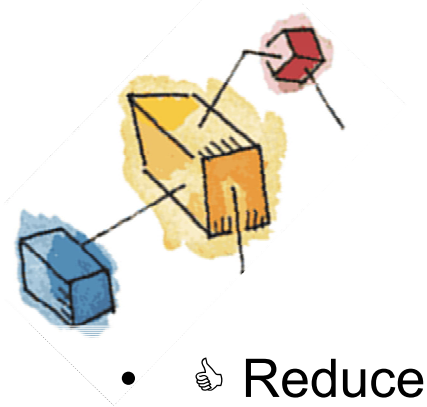


SCAN

- Also known as the elevator algorithm
- Arm moves in one direction, satisfying all outstanding requests until there are no more requests in that direction, then the service direction is reversed.



(c) SCAN



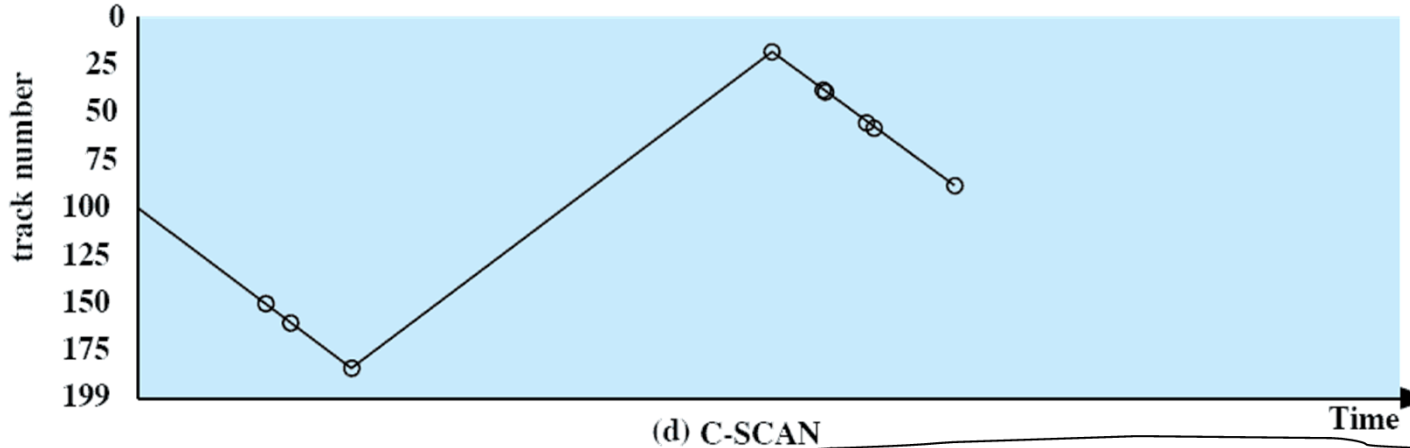
SCAN

- 👍 Reduces unfairness of SSTF
  - SCAN ensures that all requests in a given direction will be serviced before the requests in the opposite direction.
- 👎 SCAN is biased against the area most recently traversed

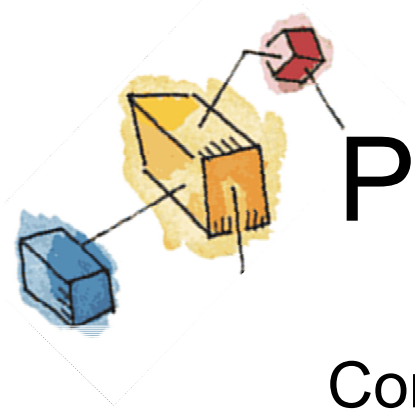
  → does not exploit locality as well as SSTF
- 👎 SCAN may still service arriving requests before waiting requests.
  - A request arriving just behind the head will have to wait until the arm completes one direction, reverses direction and comes back while a request arrives just in front of the head, it will be serviced almost immediately.
- At the point when the head reverses direction, relatively few requests are immediately in front of the head, since these tracks have recently been serviced.

C-SCAN (Circular SCAN)

- Restricts scanning to **one** service direction only.
- When the last request in the service direction is satisfied, the arm reverses its direction without servicing any requests on the return trip and the scan begins again at the first request in the service direction.
- 👍 Reduces the maximum delay experienced by new requests.



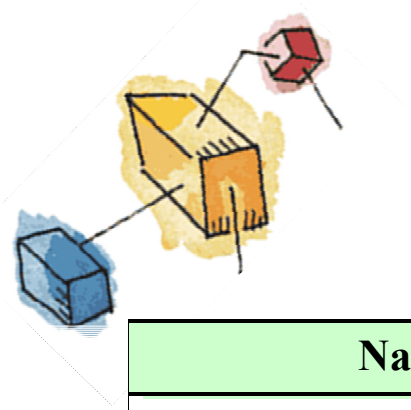
(d) C-SCAN



Performance Compared

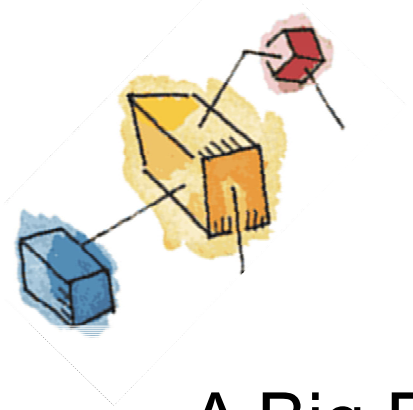
Comparison of Disk Scheduling Algorithms

(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



Disk Scheduling Policies

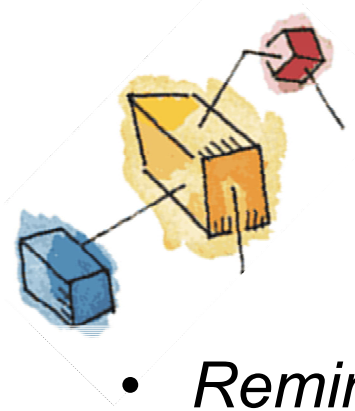
Name	Description	Remarks
Selection according to requestor		
Random	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability



Roadmap

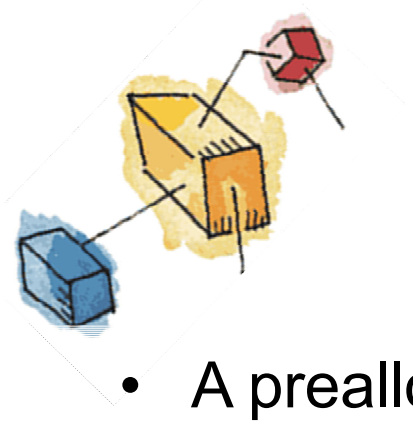
- A Big Picture
- Record Blocking
- Disk Scheduling

→ File Allocation



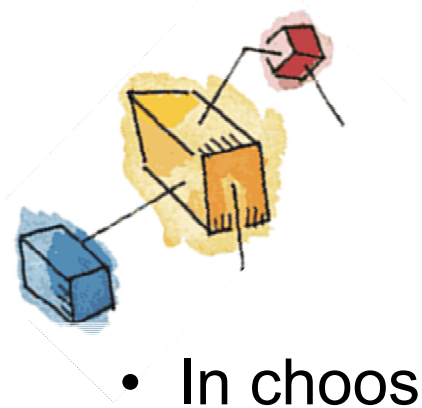
Secondary Storage Management

- *Reminder*: on secondary storage, a file consists of a collection of blocks.
- OS is responsible for allocating blocks to files BUT **when**: at creation time or as needed?
- Space is allocated to a file as one or more ***portions*** (contiguous set of allocated blocks) BUT **what** size of portion should be?
- ***File allocation table*** (FAT) is a data structure used to keep track of the portions assigned to a file.



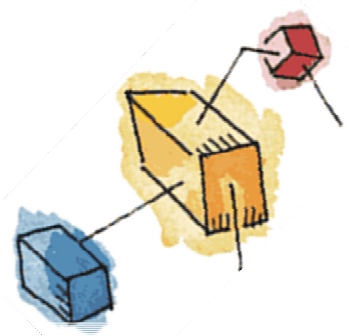
Pre-allocation vs Dynamic Allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
- For many applications, it is difficult to estimate reliably the maximum potential size of the file
  - tends to be wasteful because users and application programmers tend to overestimate size
- Dynamic allocation allocates space to a file in portions as needed



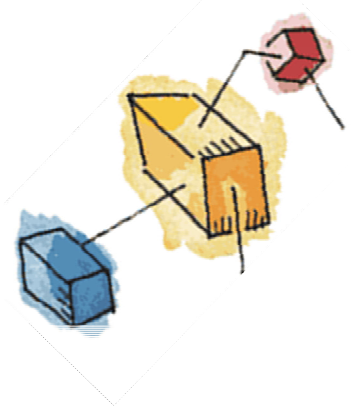
Portion Size

- In choosing a portion size, there is a trade-off between efficiency from the point of view of a single file vs. the overall system efficiency
- Issues to be considered:
  - Large portions: contiguity of space increases performance
  - Small portions: a large number of small portions increases the size of tables needed to manage the allocation information
  - Fixed-size portions: simplifies the reallocation of space
  - Variable-size or small fixed-size portions: minimizes waste of unused storage due to overallocation



Two Major Alternatives

- Variable, large contiguous portions
  - 👍 provides better performance
  - 👍 the variable size avoids waste
  - 👍 the file allocation tables are small
  - 👎 space is hard to reuse
- Blocks: small fixed-sized portions
  - 👍 small fixed portions provide greater flexibility
  - 👎 they may require large tables or complex structures for their allocation
  - 👎 contiguity has been abandoned as a primary goal; blocks are allocated as needed

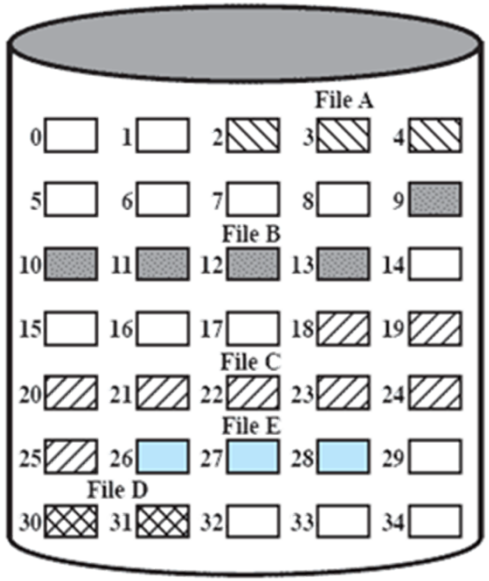


File Allocation Methods

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion 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

Contiguous File Allocation

- A single contiguous set of *blocks* is allocated to a file at the time of creation
- A preallocation with variable-size portions
- 👍 FAT needs a single entry for each file

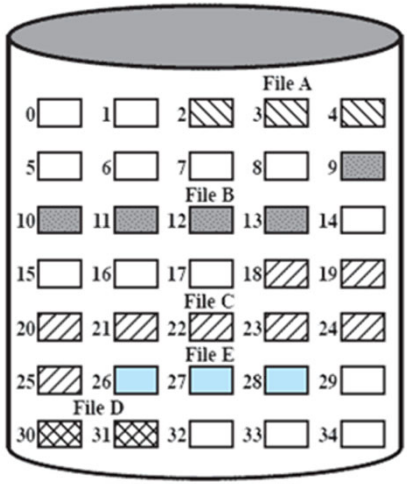


File Allocation Table

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

Contiguous File Allocation

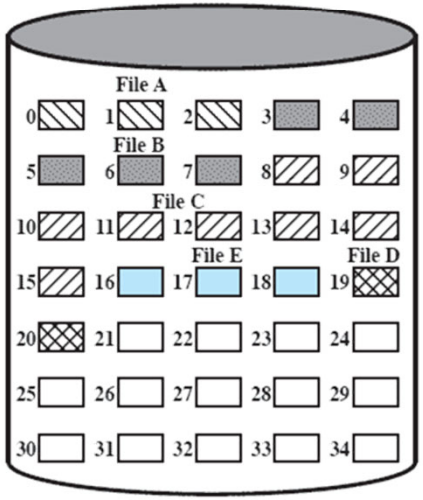
- 👍 Best for individual sequential file
  - multiple blocks can be read in at a time to improve I/O performance
  - Also easy to retrieve a single block
    - if a file starts at block *b*, and the *i* th block of the file is wanted, its location on secondary storage is simply *b* + *i* – 1.
- 👎 External fragmentation will occur
  - Need to perform *compaction (a.k.a. disk defragmentation)*



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

Before compaction



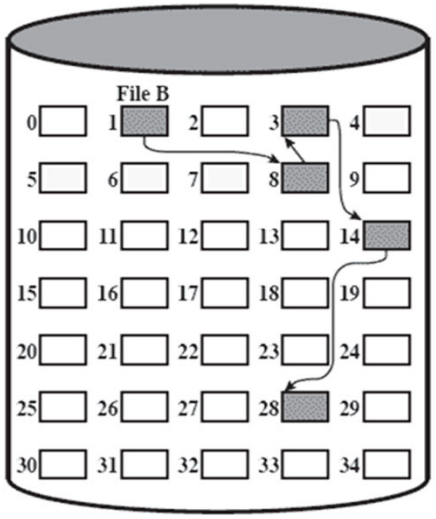
File Allocation Table

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

After compaction

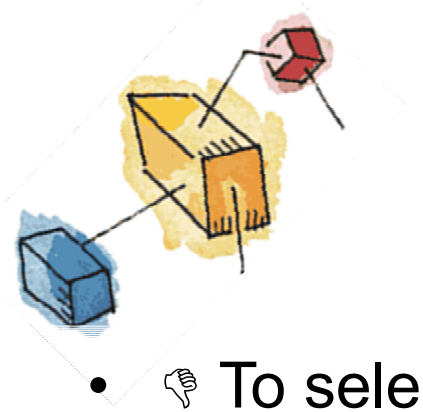
Chained Allocation

- Allocation is on basis of individual block
- Each block contains a pointer to the next block in the chain
- Allows both preallocation and dynamic allocation
- 👍 FAT needs a single entry for each file
- 👍 Selection of blocks is simple
  - any free block can be added to a chain
- 👍 No external fragmentation
- 👍 Best for sequential files that are to be processed sequentially



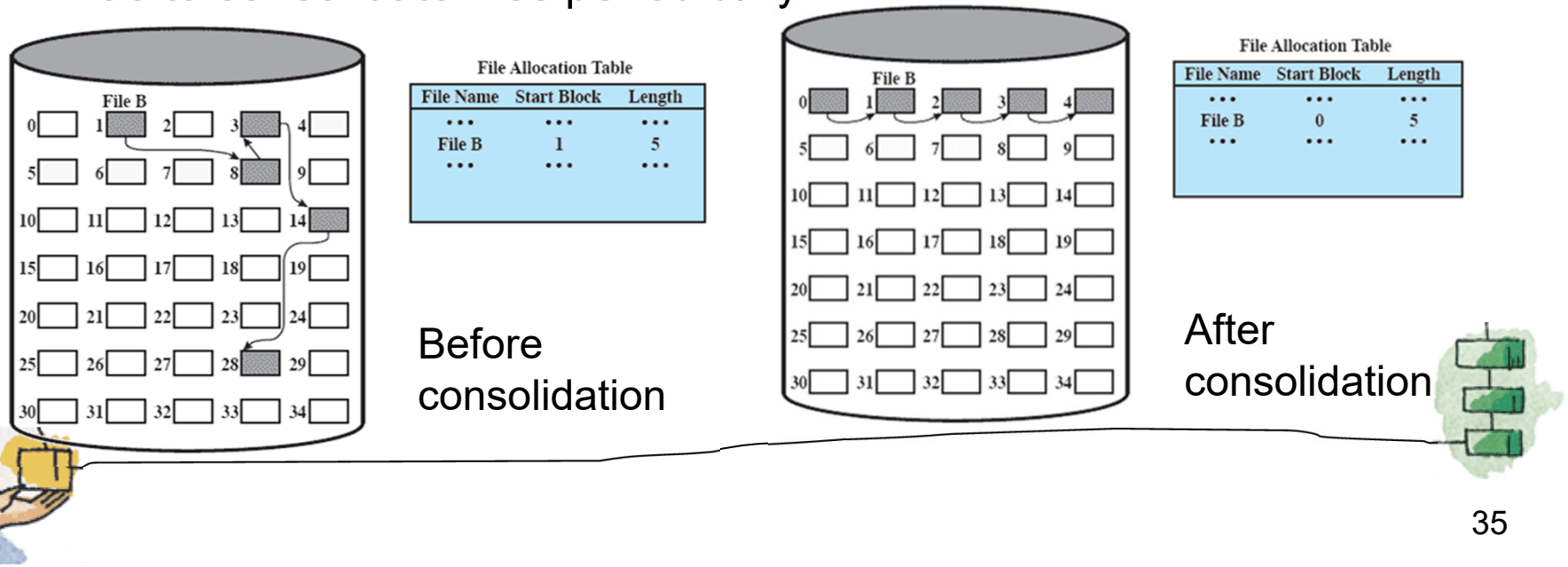
File Allocation Table

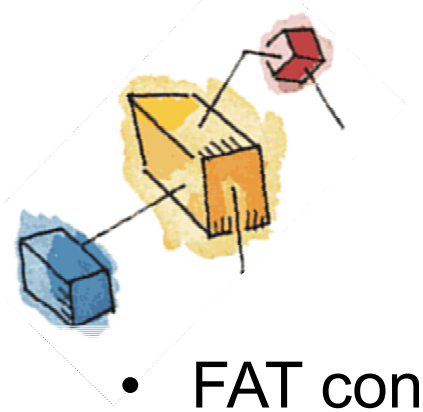
File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...



Chained Allocation

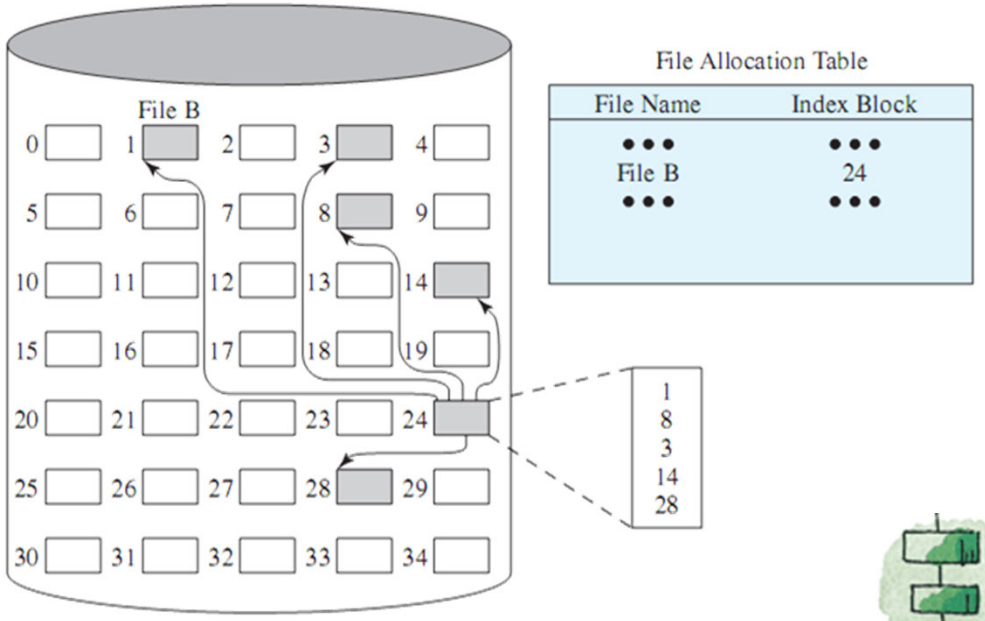
- 👎 To select an individual block of a file requires tracing through the chain to the desired block
- 👎 No accommodation of the principle of locality
  - a series of accesses to different parts of the disk are required for sequential processing
  - has to consolidate files periodically

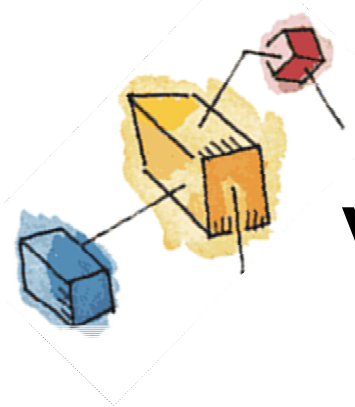




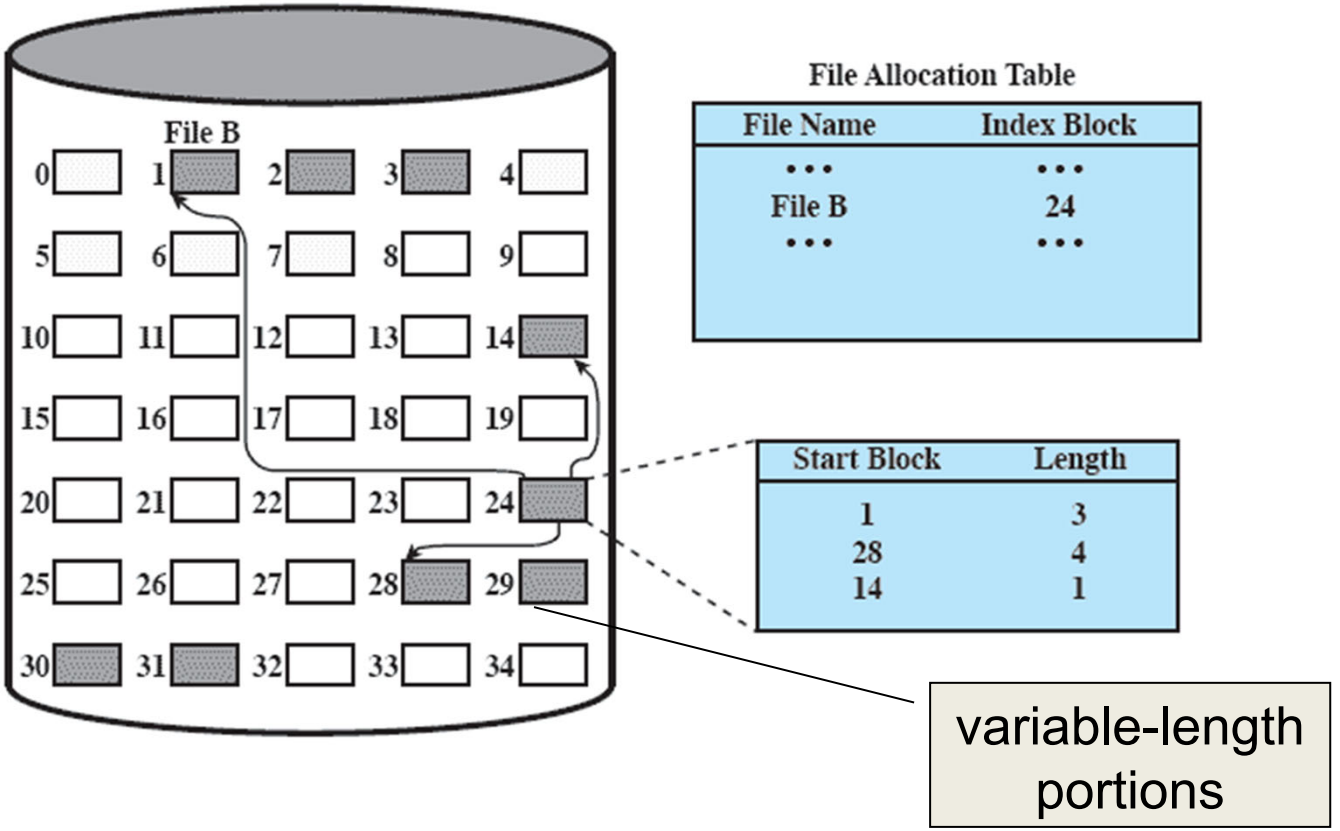
Indexed Allocation with Block Portions

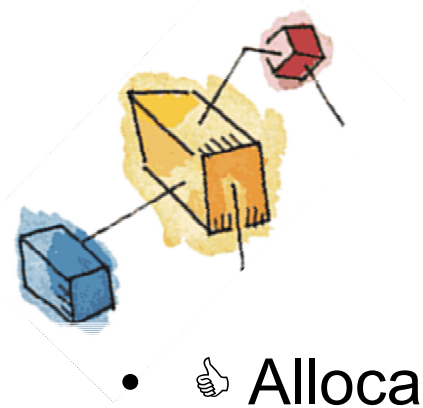
- FAT contains an entry for a file that points to the ***file index block***
  - The file index block has one entry for each portion allocated to the file
- Allocation may be either
  - Fixed-size blocks
  - Variable-size portions





Indexed Allocation with Variable Length Portions





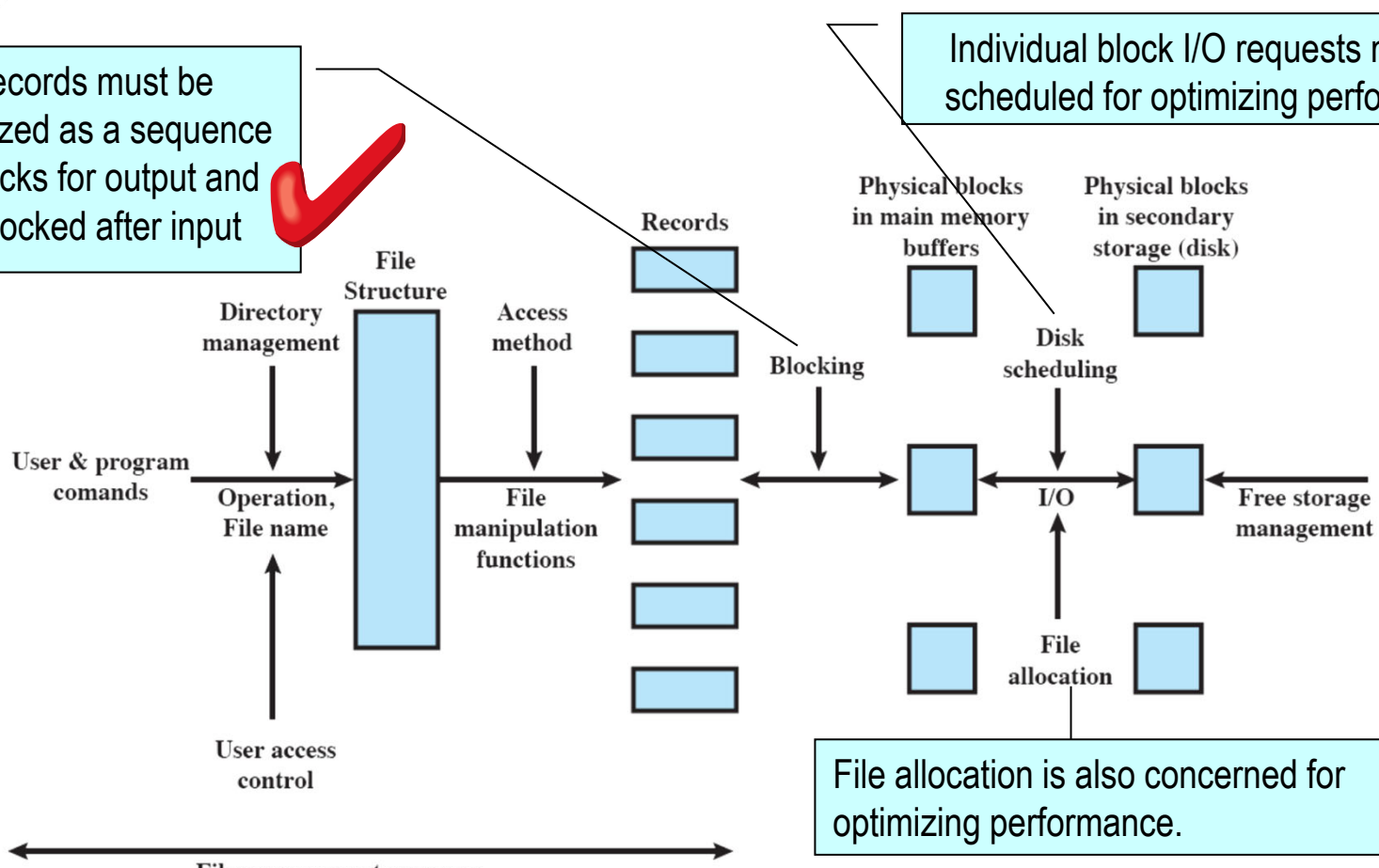
Indexed Allocation

- 👍 Allocation by blocks eliminates external fragmentation
- 👍 Allocation by variable-size portions improves locality
- 👍 Supports both sequential and direct access to the file and thus is the most popular
- 👎 File consolidation may be done
  - reduces the size of the index in the case of variable-size portions

Revisit the Big Picture

Records must be organized as a sequence of blocks for output and unblocked after input

Individual block I/O requests must be scheduled for optimizing performance



File allocation is also concerned for optimizing performance.

Figure 12.2 Elements of File Management