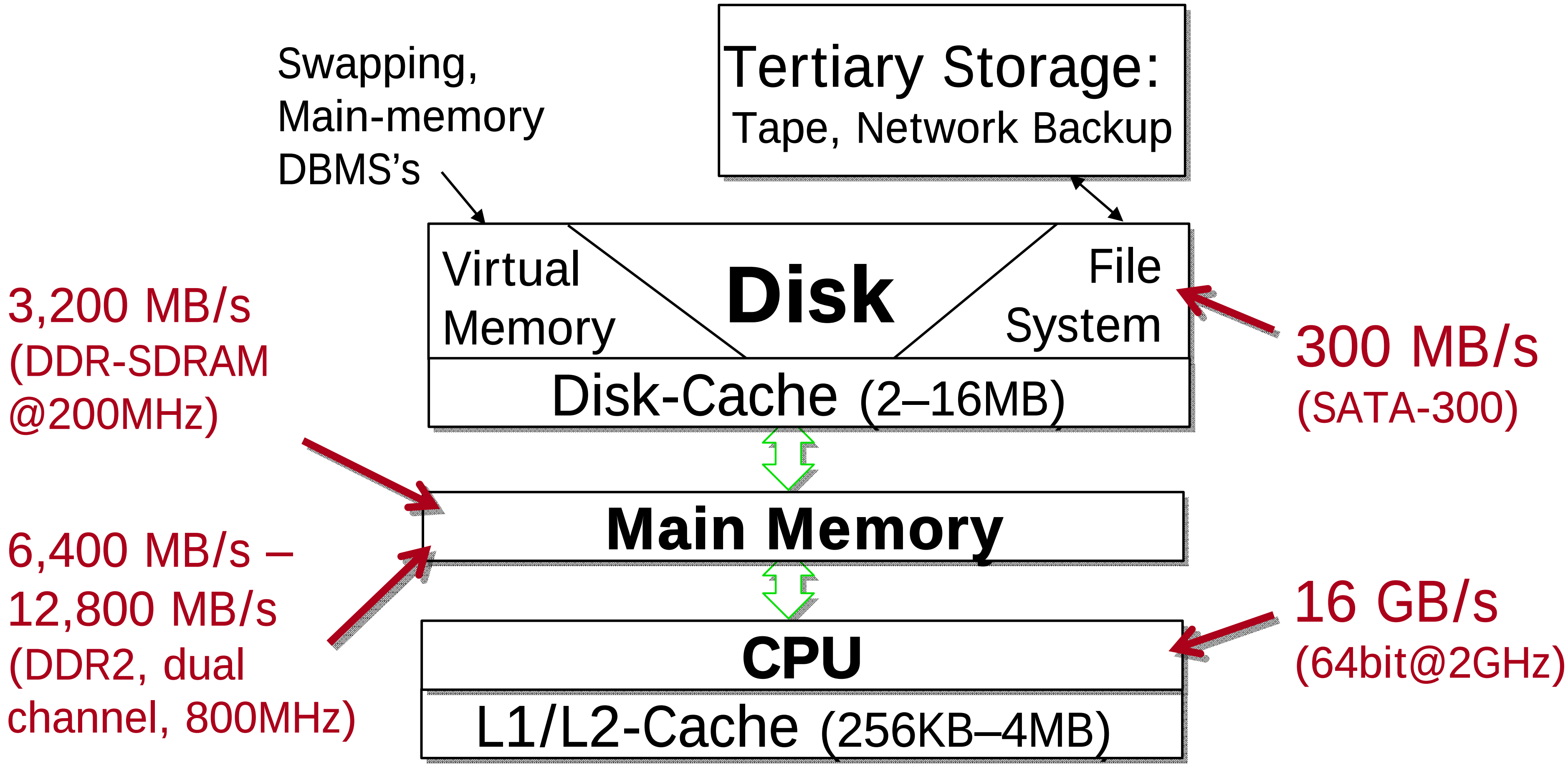
Data Storage and Query Answering

Data Storage and Disk Structure (2)

Review: The Memory Hierarchy



CPU-to-Main-Memory:
~200 cycles latency

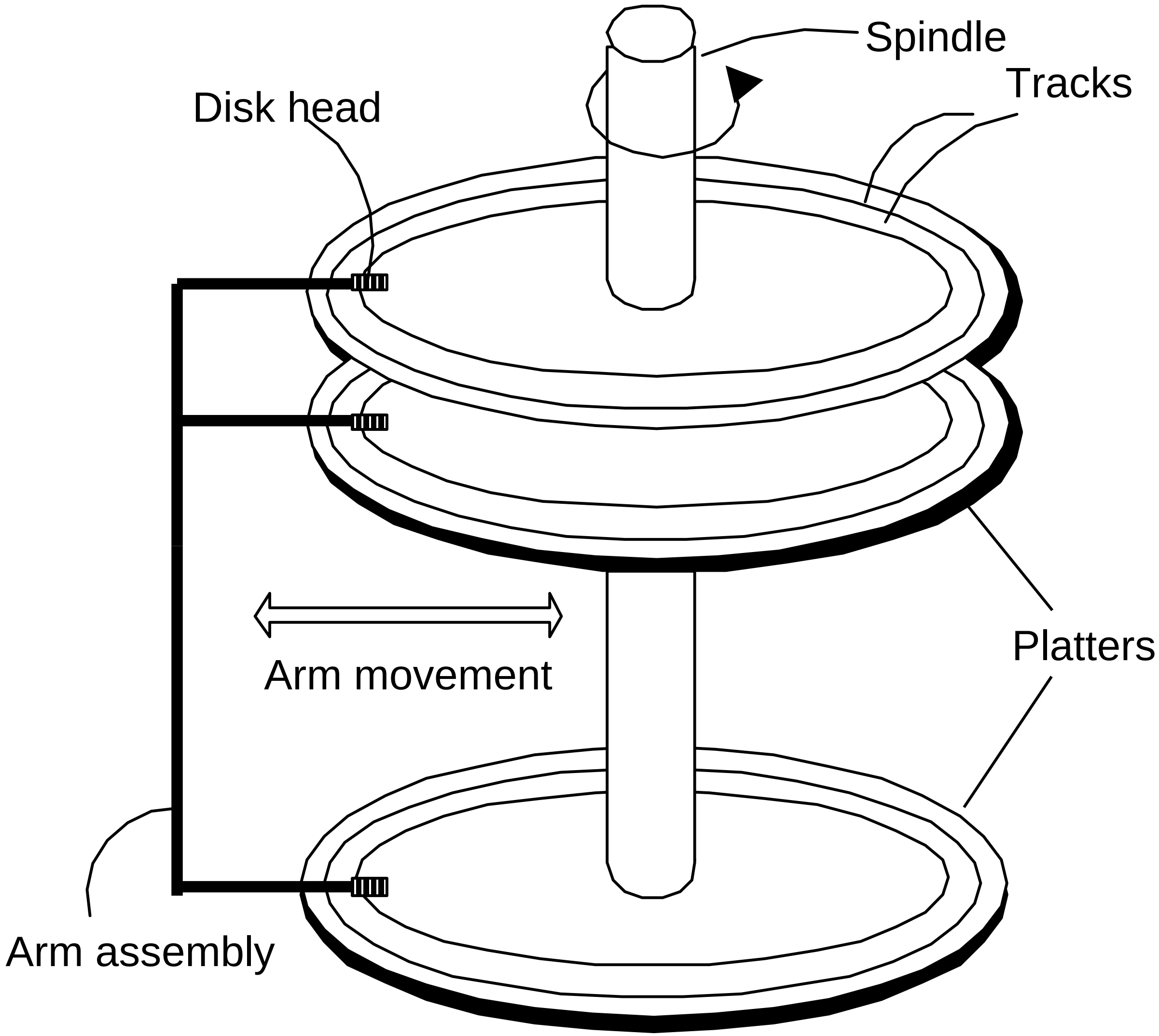
CPU-to-L1-Cache:
~5 cycles initial latency, then "burst" mode

Disks

- Secondary storage device of choice.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Main advantage over tapes: *random access* vs. *sequential access*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
- Therefore, relative placement of pages on disk has major impact on DBMS performance!

Disks

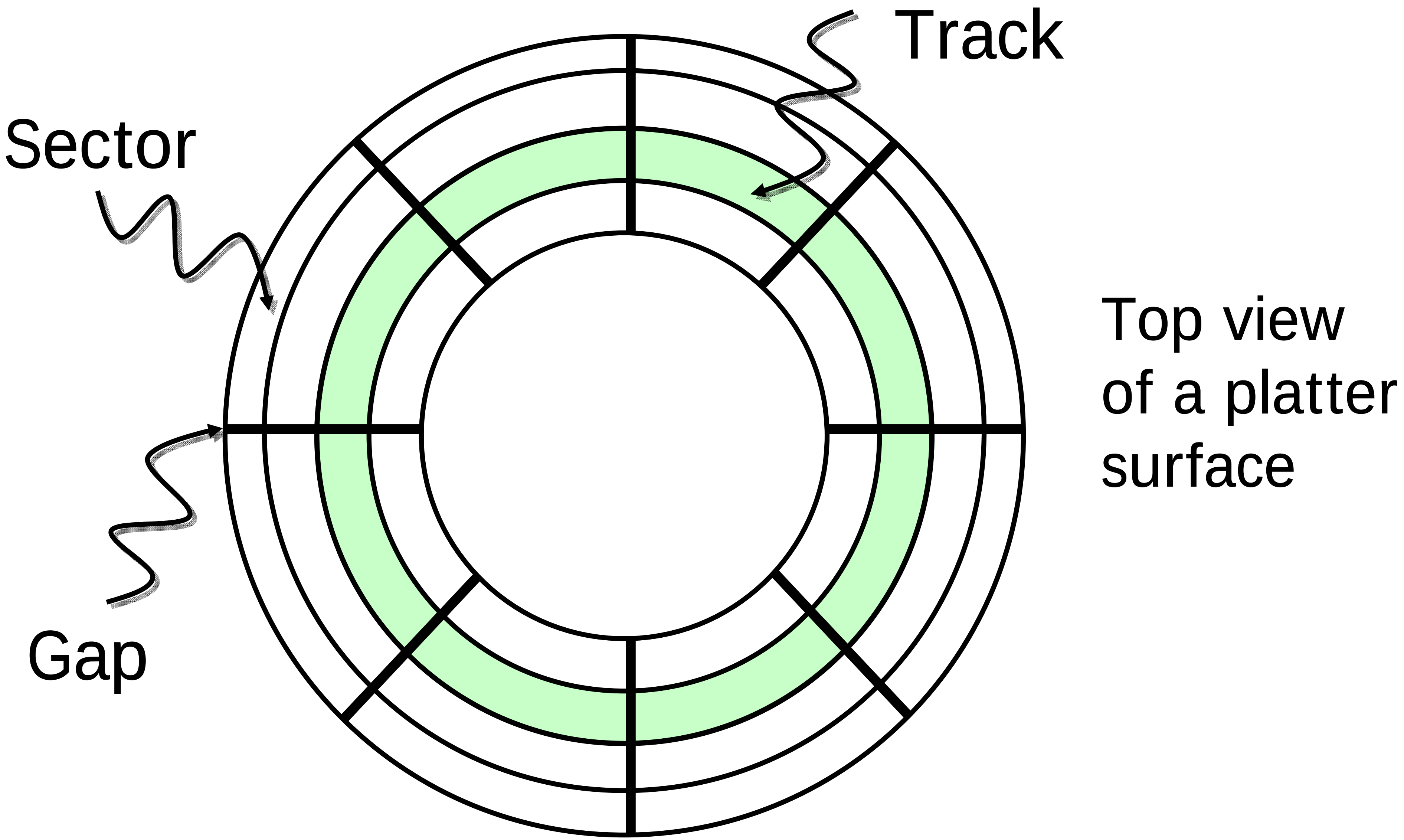
- Disk consists of two main, moving parts: disk assembly and head assembly.
- *Disk assembly* stores information, *head assembly* reads and writes information.



Disks

- The platters rotate around central spindle.
- Upper and lower platter surfaces are covered with magnetic material, which is used to store bits.
- The arm assembly is moved in or out to position a head on a desired track.
- All tracks under heads at the same time make a *cylinder* (imaginary!).
- Only one head reads/writes at any one time.

Disks



Disks

- *Block size* is a multiple of *sector size* (which is fixed).
- Time to access (read/write) a disk block (*disk latency*) consists of three components:
  - *seek time*: moving arms to position disk head on track,
  - *rotational delay* (waiting for block to rotate under head), and
  - *transfer time* (actually moving data to/from disk surface).
- Seek time and rotational delay dominate.

Disks

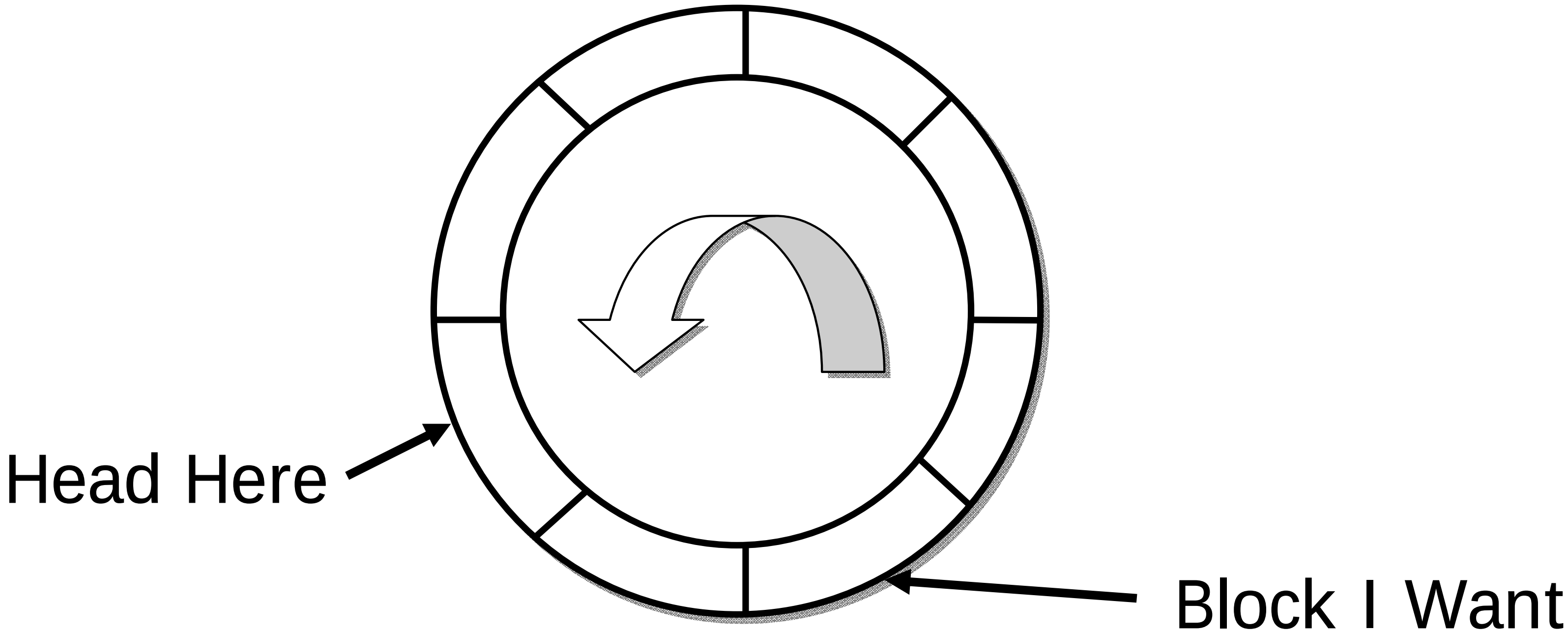
Average seek time

$$S = \frac{\sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \text{SEEKTIME}(i \to j)}{N(N-1)}$$

Typical average seek time = 5 ms

Disks

Average rotational delay



Average rotational delay R = 1/2 revolution

Typical R = 5 ms

Disks

Transfer time

Typical transfer rate: 100 MB/sec
Typical block size: 16KB

Transfer time: $\dfrac{\text{block size}}{\text{transfer rate}}$

Typical transfer time = 0.16 ms

Disks

- Typical average disk latency is 10 ms, maximum latency 20 ms.
- In 10 ms, a modern microprocessor can execute millions of instructions.
- Thus, the time for a block access by far dominates the time typically needed for processing the data in memory.
- The *number of disk I/Os* (block accesses) is a good approximation for the cost of a database operation.

Accelerating Disk Access

- *Organize data by cylinders* to minimize the seek time and rotational delay.
- *'Next'* block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder.
- Blocks in a file are placed sequentially on disk (by 'next').
- Disk latency can approach the transfer rate.

Accelerating Disk Access

Example

- Assuming 10 ms average seek time, no rotational delay, 40 MB/s transfer rate.
- Read a single 4 KB Block
  - Random I/O: ~ 10 ms
  - Sequential I/O: ~ 10 ms
- Read 4 MB in 4 KB Blocks (amortized)
  - Random I/O: ~ 10 s
  - Sequential I/O: ~ 0.1 s

→ Speedup factor of 100

Accelerating Disk Access

Block size selection

- Bigger blocks → amortize I/O cost.
- Bigger blocks → read in more useless stuff and takes longer to read.
- Good trade-off block size from 4KB to 16 KB.
- With decreasing memory costs, blocks are becoming bigger!

Accelerating Disk Access

Using multiple disks

- Replace one disk (with one independent head) by many disks (with many independent heads).
- *Striping* a relation R: divide its blocks over n disks in a round robin fashion.
- Assuming that disk controller, bus and main memory can handle n times the transfer rate, striping a relation across n disks can lead to a speedup factor of up to n.

Accelerating Disk Access

Disk scheduling

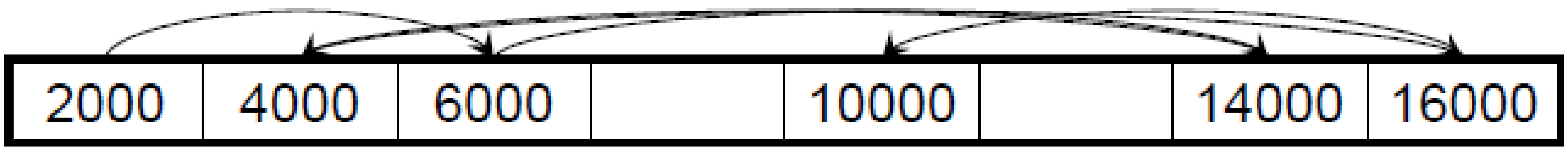
- For I/O requests from different processes, let the disk controller choose the processing order.
- According to the *elevator algorithm*, the disk controller keeps sweeping from the innermost to the outermost cylinder, stopping at a cylinder for which there is an I/O request.
- Can reverse sweep direction as soon as there is no I/O request ahead in the current direction.
- Optimizes the throughput and average response time.

Accelerating Disk Access

Example: First-Come-First-Serve Scheduling

	2000	2000
+	|6000-2000|	4000
+	|14000-6000|	8000
+	|4000-14000|	10000
+	|16000-4000|	12000
+	|10000-16000|	6000
=		42000

Cylinder of request	First time available	Time completed
2000	0	4.42
6000	0	13.84
14000	0	27.26
4000	10	42.68
16000	20	60.10
10000	30	71.52

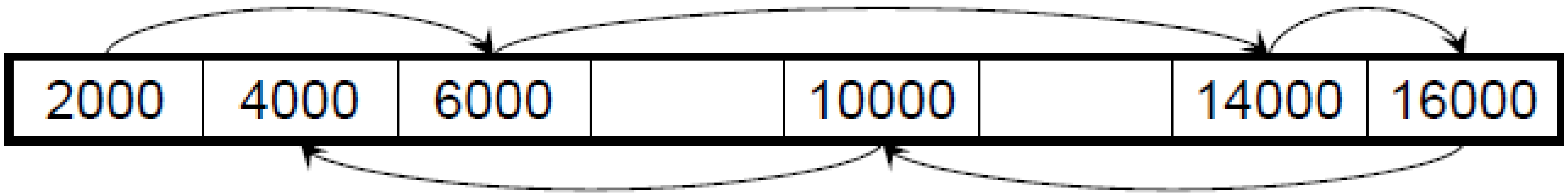


Accelerating Disk Access

Example: Elevator Algorithm

	2000	2000
+	|6000-2000|	4000
+	|14000-6000|	8000
+	|16000-14000|	2000
+	|10000-16000|	6000
+	|4000-10000|	6000
=		28000

Cylinder of request	First time available	Time completed
2000	0	4.42
6000	0	13.84
14000	0	27.26
4000	10	57.52
16000	20	34.68
10000	30	46.10



Comments

- The elevator algorithm can achieve good performance on average
- In our example, it saves 14000 (1/3 of 42000 in first-come-first-serve method)
- The more different request, the better performance the elevator algorithm
- The elevator algorithm is not optimal
  - Can you give an example where the elevator algorithm performs worse than the first-come-first serve method?

Accelerating Disk Access

Double buffering

- In some scenarios, we can predict the order in which blocks will be requested from disk by some process.
- *Prefetching* (*double buffering*) is the method of fetching the necessary blocks into the buffer in advance.
- Requires enough buffer space.
- Speedup factor up to n, where n is the number of blocks requested by a process.

Accelerating Disk Access

Single buffering

- (1) Read B1 → Buffer
  (2) Process Data in Buffer
  (3) Read B2 → Buffer
  (4) Process Data in Buffer
  ...
- Execution time = n(P+R)
  where
  P = time to process one block
  R = time to read in one block
  n = # blocks read.

Accelerating Disk Access

Double buffering

- (1) Read B1, . . ., Bn → Buffer
  (2) Process B1 in Buffer
  (3) Process B2 in Buffer
  ...
- Execution time = R + nP
  as opposed to n(P+R).

  → remember that R >> P