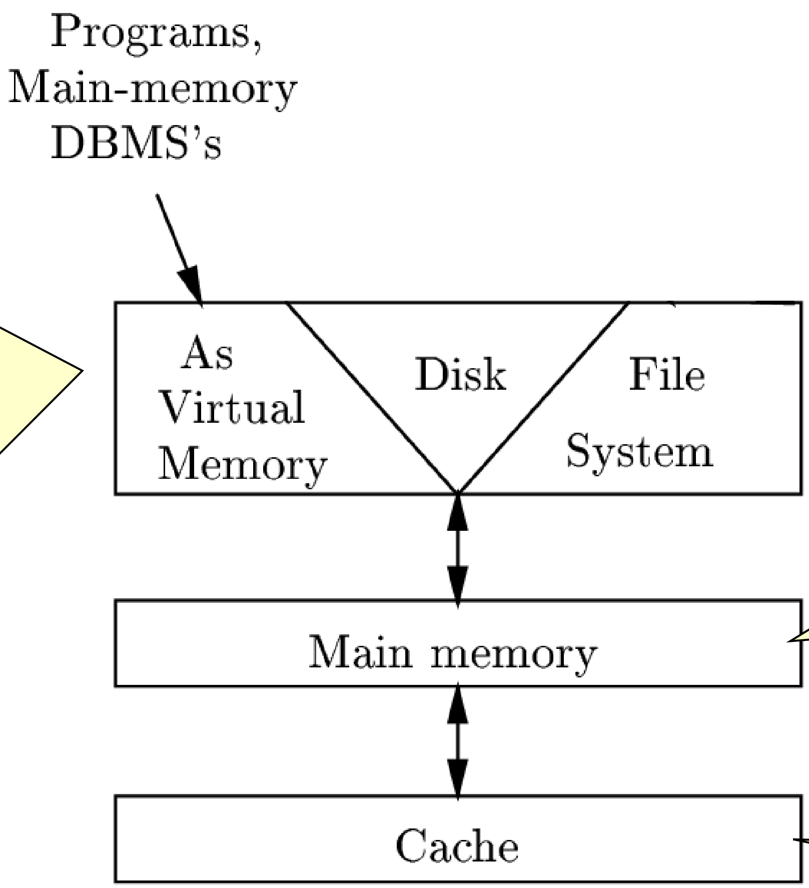
Storage

The Memory Hierarchy

Access times in milliseconds, great variability.

Unit of read/write = block or page, typically 16Kb.

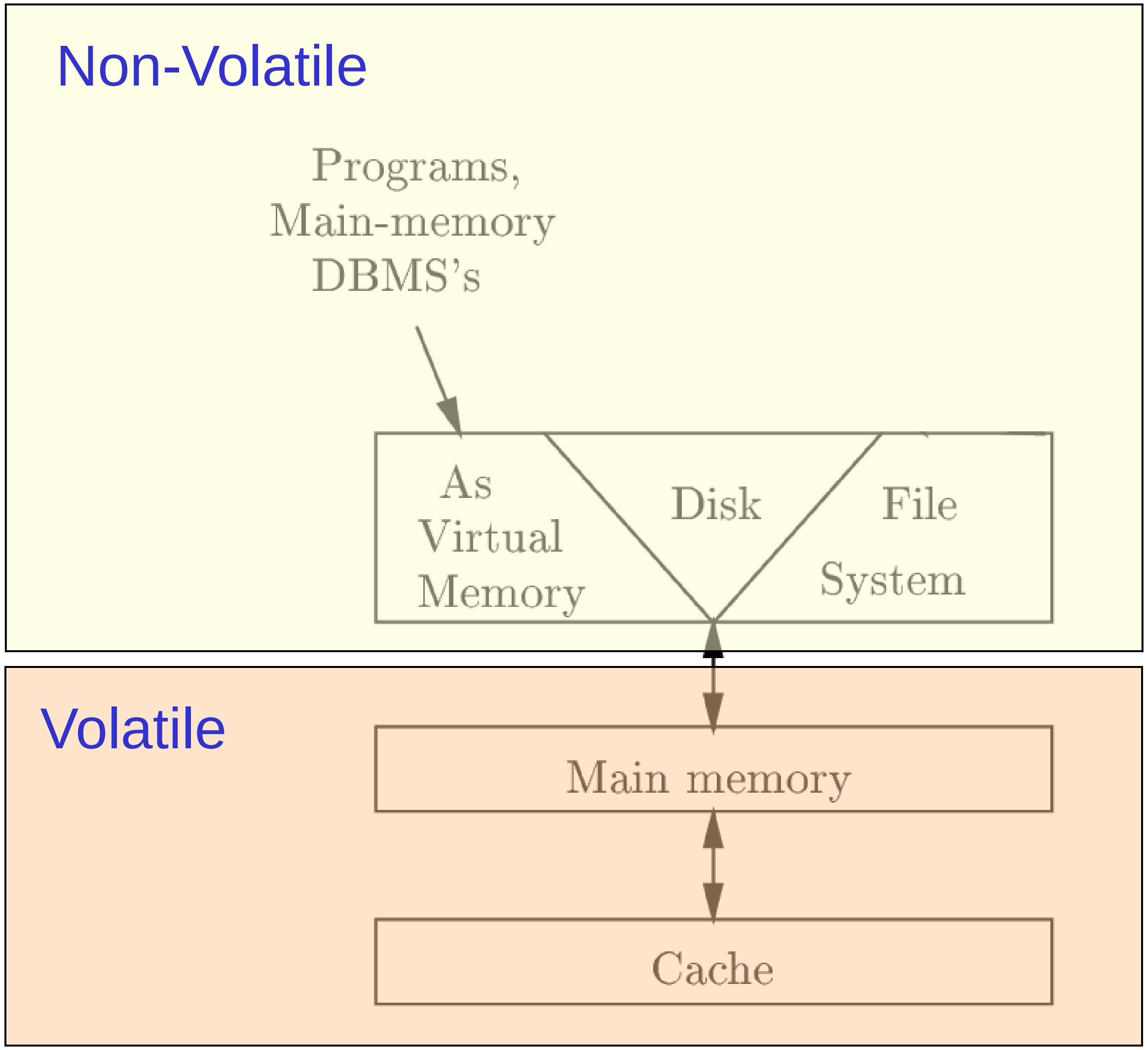
Capacities in GB/TB.



under a microsecond, random access, Capacities few GB.

fastest, but small

Volatile vs. Non-Volatile



A storage device is nonvolatile if it can retain its data after a power shutoff.

Computer Quantities

			Roughly:
K	Kilo	2^{10}	10^{3}
M	Mega	2^{20}	10^{6}
G	Giga	2^{30}	10^{9}
T	Tera	2^{40}	10^{12}
P	Peta	2^{50}	10^{15}

Data Deluge

- 2012
  - every day 2.5 quintillion bytes of data (1 followed by 18 zeros) were created,

- 90% of the world's data created in the last two years alone.

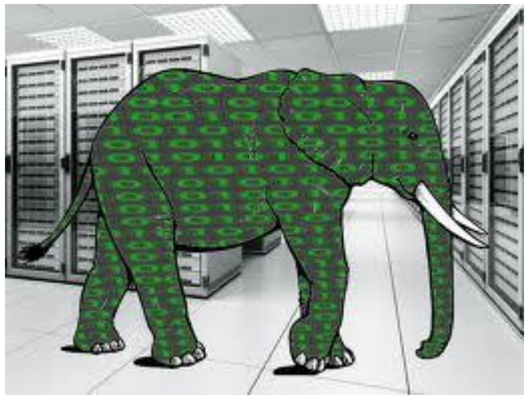
Largest databases

- Largest database: World Data Centre for Climate

  Max Planck Institute and *German Climate Computing Centre*
  - *220 terabytes of data on climate research and climatic trends*
  - *110 terabytes worth of climate simulation data*
  - *6 petabytes worth of additional information stored on tapes*

- AT&T
  - *323 terabytes of information*
  - *1.9 trillion phone call records*

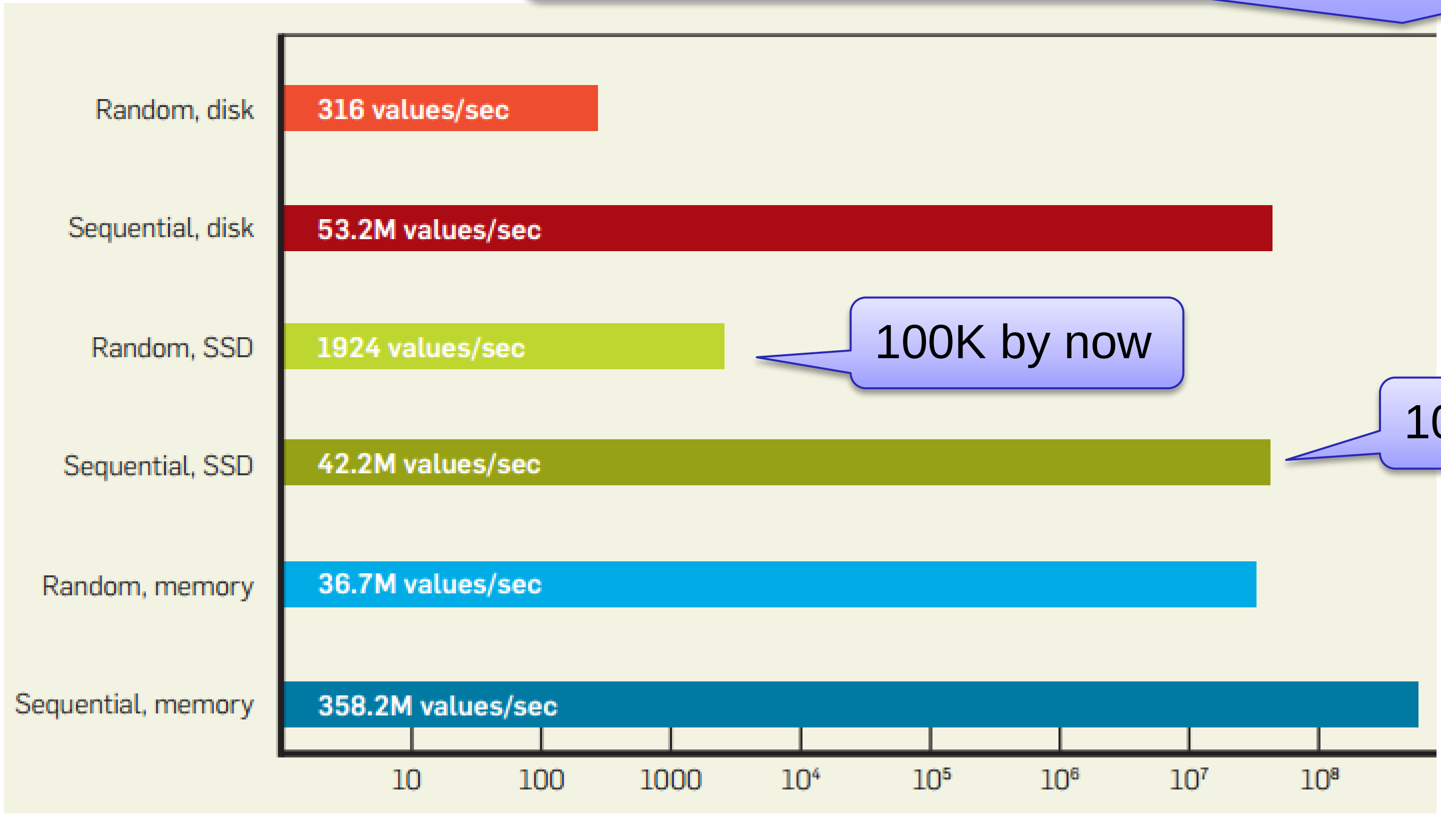
- Google
  - *91 million searches per day*
  - *33 trillion database entries a year*



Storage Types and Speeds

Adam Jacobs: "Pathologies of Big Data", ACM Communications, August 2009.

Conclusion: Regardless of storage medium, random access costs orders of magnitude more than sequential access.



100K by now

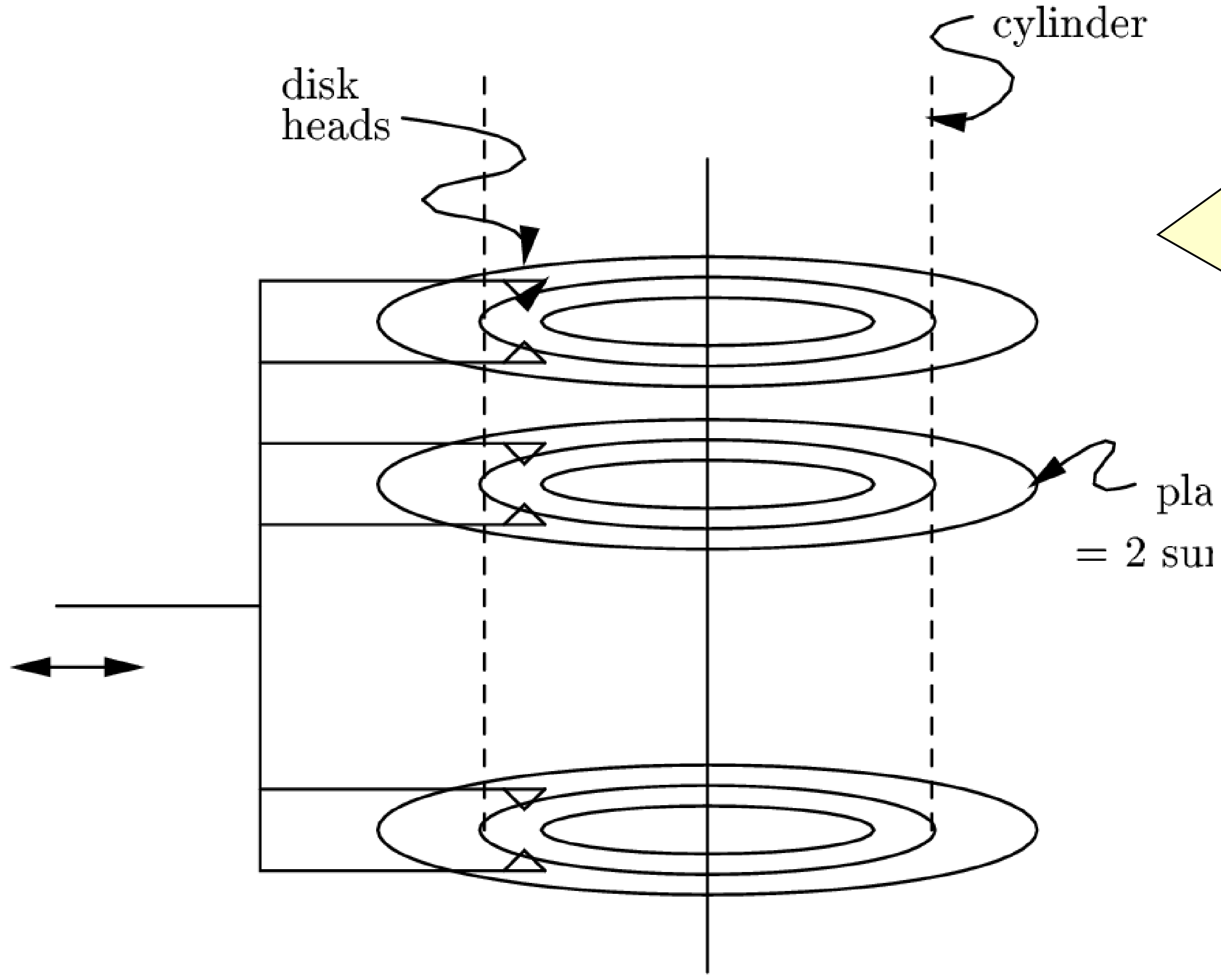
100M by now

"Values" are 4B integers

Now = 2016

Numbers are for consumer-grade machines

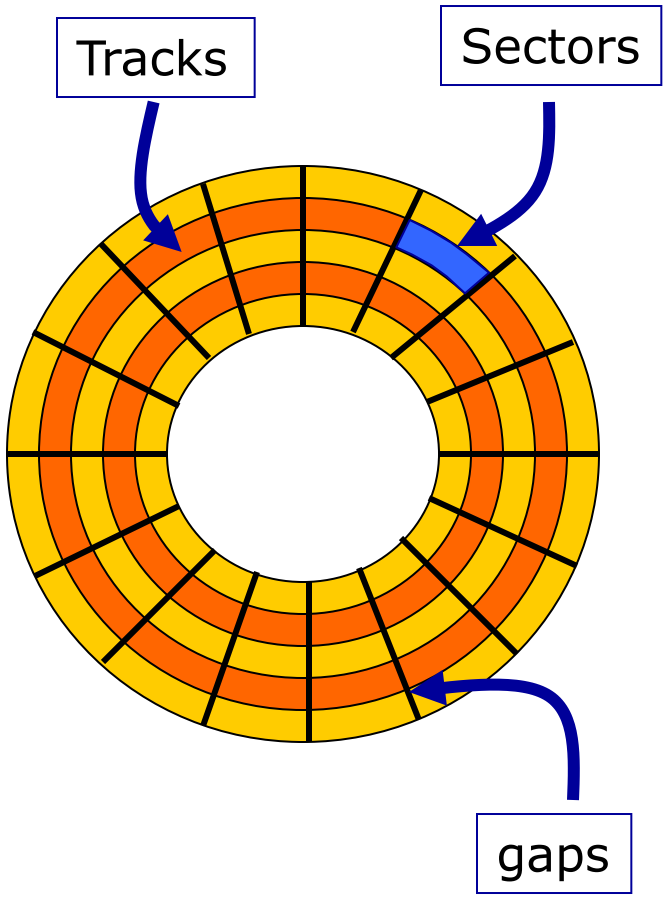
Disks



- Platters with top and bottom surfaces rotate around a spindle.
- 2--30 surfaces.
- Rotation speed: 3600--7200 rpm.
- One head per surface.
- All heads move in and out in unison.

Tracks and sectors

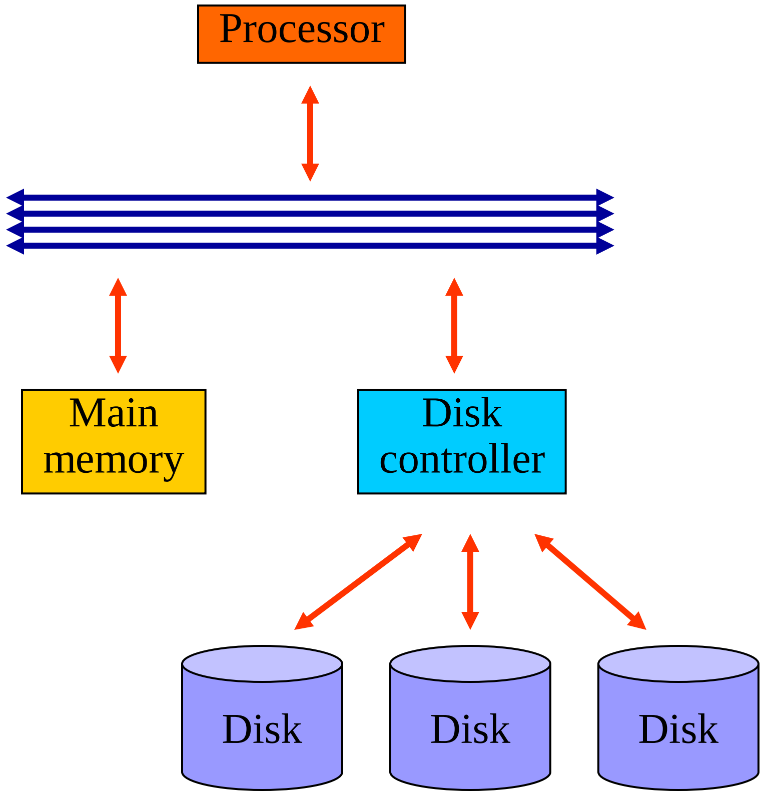
- Surfaces are covered with concentric tracks.
  - Tracks at a common radius = **cylinder**.
    - All data of a cylinder can be read quickly, without moving the heads.
    - A big magnetic disk: 2^{16} = 65,536 cylinders
- Tracks are divided into sectors by unmagnetized gaps (which are 10% of track).
  - Typical track: 256 sectors.
  - Typical sector: 4096 bytes.
- Sectors are grouped into blocks.
  - Typical block: 16K = 4 4096byte sectors.



MEGATRON 747 Disk Parameters

- There are 8 platters providing 16 surfaces.
- There are 2^{16}, or 65,536 tracks per surface.
- There are (on average) 2^8= 256 sectors per track.
- There are 2^{12}=4096=4K bytes per sector.
- Capacity = $16*2^{16}*2^8*2^{12} = 2^{40}$ = 1 TB

Disk Controller

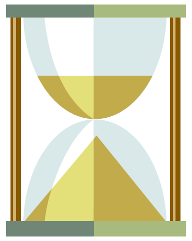


1. Schedule the disk heads.
2. Buffer data in and out of disk.
3. Manage the "bad blocks" so they are not used.

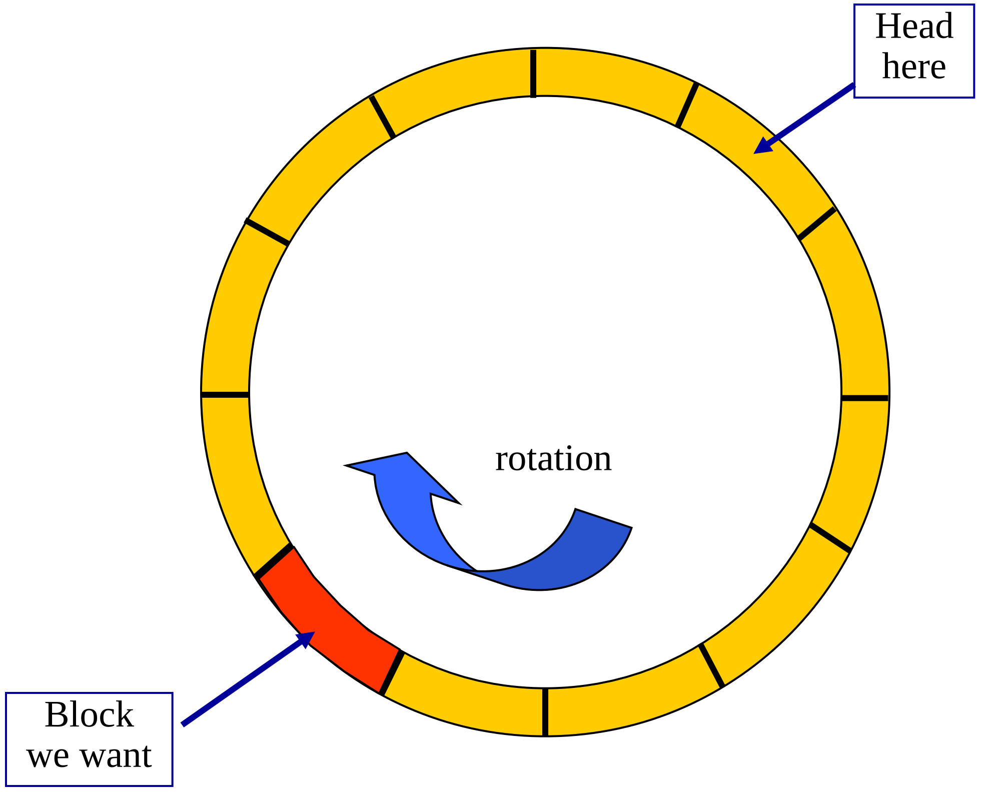
Disk access time

- Latency of the disk (access time): The time to bring a block to main memory from disk.

- Main components of access time are:
  - **Seek time** = time to move heads to proper cylinder.
  - **Rotational delay** = time for desired block to come under head.
  - **Transfer time** = time during which the block passes under head.



Cause of rotational delay



On average, the desired sector will be about half way around the circle when the heads arrive at the cylinder.

MEGATRON 747 Timing Example

- To move the head assembly between cylinders takes

  1 ms to **start** and **stop**, plus

  1 ms for every **4000 cylinders** traveled.

- Thus, moving from the innermost to the outermost track, a distance of 65,536 tracks, is about **17.38 ms**.

- Disk rotates at **7200 rpm**;

  one rotation in **8.33 ms**

- Gaps occupy 10% of the space around a track.

MIN time to read a 16,384-byte block

- The minimum time, is just the transfer time.
- Since there are 4096 bytes per sector on the Megatron 747, the block occupies 4 sectors.
  - The heads must therefore pass over 4 sectors and the 3 gaps between them.

- Recall that:
  - the gaps represent 10% of the circle and sectors the remaining 90%,
  - i.e. 36 degrees are occupied by gaps and
    324 degrees by the sectors.
  - there are 256 gaps and 256 sectors around the circle,
- So
  - a gap is 36/256 = 0.14 degrees, and
  - a sector is 324/256 = 1.265 degrees
- The total degrees covered by 3 gaps and 4 sectors is: 3*0.14+4*1.265 = 5.48 degrees

- The transfer time is thus (5.48/360) x 8.33 = **0.13 ms**.

That is, the block might be on a track over which the head is positioned already, and the first sector of the block might be about to pass under the head.

That is, 5.48/360 is the fraction of a rotation need to read the entire block, and 8.33 ms is the amount of time for a 360-degree rotation.

MAX time to read a 16,384-byte block

- Worst-case latency is 17.38+8.33+0.13=**25.84 ms**.

The heads are positioned at the innermost cylinder, and the block we want to read is on the outermost cylinder (or vice versa).

AVG time to read a 16,384-byte block

- Transfer time is always **0.13** milliseconds.
- Average rotational latency is the time to rotate the disk half way around, or **4.17** milliseconds.

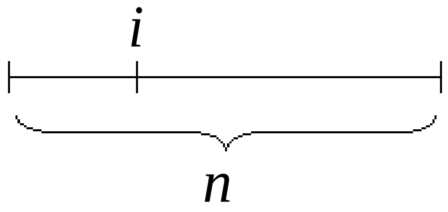
- What about the average seek time? Is it just the time to move across half the tracks?

- Not quite right, since typically, the heads are initially somewhere near the middle and therefore will have to move less than half the distance, on average, to the desired cylinder.

AVG time to read a 16,384-byte block

- What about the average seek time?
- Assume the heads are initially at any of the 16,384 cylinders with equal probability.
- If at cylinder 1 or cylinder 16,384, then the average number of tracks to move is (1 + 2 +... + 16383)/16384, or about 8192 tracks.
- At the middle cylinder 8192, the head is equally likely to move in or out, and either way, it will move on average about a quarter of the tracks (4096)

AVG time to read a 16,384-byte block



$$\frac{i}{2}\cdot\frac{i}{n}+\frac{n-i}{2}\cdot\frac{n-i}{n}$$

Average number of cyls to travel, if the heads are currently positioned at cyl *i*.

Avg number of cyls to travel if the block is on the **left**.

Probabilit y the block is on the **left**

Avg number of cyls to travel if the block is on the **right**.

Probabilit y the block is on the **right**

$$
\begin{aligned}
&AVG \\
&= \frac{1}{n}\int_0^n \frac{i^2}{2n} + \frac{(n-i)^2}{2n}\,di \\
&= \frac{1}{2n^2}\int_0^n i^2 di + \frac{1}{2n^2}\int_0^n (n-i)^2\,di \\
&= \frac{1}{2n^2}\int_0^n i^2 di + \frac{1}{2n^2}\int_{-n}^0 j^2 dj \qquad \text{substituting} \quad i-n=j \\
&= \frac{1}{2n^2}\left.\frac{i^3}{3}\right|_0^n + \frac{1}{2n^2}\left.\frac{j^3}{3}\right|_{-n}^0 \\
&= \frac{1}{2n^2}\frac{n^3}{3} + \frac{1}{2n^2}\frac{n^3}{3} \\
&= \frac{n}{3}
\end{aligned}
$$

In other words: The average number of cylinders traveled is n/3.

(The integration math to arrive here is optional material)

AVG time to read a 16,384-byte block

- Transfer time is always **0.13** milliseconds and
- Average rotational latency is the time to rotate the disk half way around, or **4.17** milliseconds.
- Average seek time is: 1+(65536/3)*(1/4000) = **6.46** ms

- Total: 6.46 + 4.17 + 0.13 = 10.76 ms (or about 11 ms)

Writing and Modifying Blocks

- Writing same as reading, unless we verify written blocks.

- Modifying a block requires:

1. Read the block into main memory.
2. Modify the block there.
3. Write the block back to disk.

Disk Access Model

Using Secondary Storage Effectively

- In most studies of algorithms, one assumes the “RAM model”:
  - Data is in main memory,
  - Access to any item of data takes as much time as any other.

- When implementing a DBMS, one must assume that the **data does *not* fit in main memory**.

- Great advantage in choosing an algorithm that does **few random disk accesses**, even if the algorithm is not very efficient when viewed as a main-memory algorithm.

I/O model of computation

- Disk I/O = read or write of a block is very expensive compared with what is likely to be done with the block once it arrives in main memory.
  - Perhaps 1,000,000 machine instructions in the time to do one **random** disk I/O.

Assumptions

- One processor
- One disk controller, and one disk.
- The database itself is much too large to fit in main memory.

- Many users, and each user issues disk-I/O requests frequently,
- Disk controller serving on a first-come-first-served basis.
- Requests for a given user might appear random even if the table that a user is reading is stored on a single cylinder of the disk.

- The disk is a Megatron 747, with 16K blocks and the timing characteristics determined before.
- In particular, the average time to read or write a block is about 11ms

Merge Sort

- Common main memory sorting algorithms don't look so good when you take disk I/O's into account. Variants of Merge Sort do better.

- Merge = take two sorted lists and repeatedly chose the smaller of the “heads” of the lists (head = first of the unchosen).
  - Example: merge 1,3,4,8 with 2,5,7,9 = 1,2,3,4,5,7,8,9.

- Merge Sort based on recursive algorithm: divide records into two parts; recursively mergesort the parts, and merge the resulting lists.

TwoPhase, Multiway Merge Sort

- Merge Sort still not very good in disk I/O model.
  - $\mathbf{log_2 n}$ passes,
  - so each record is read/written from disk $\mathbf{log_2 n}$ times.

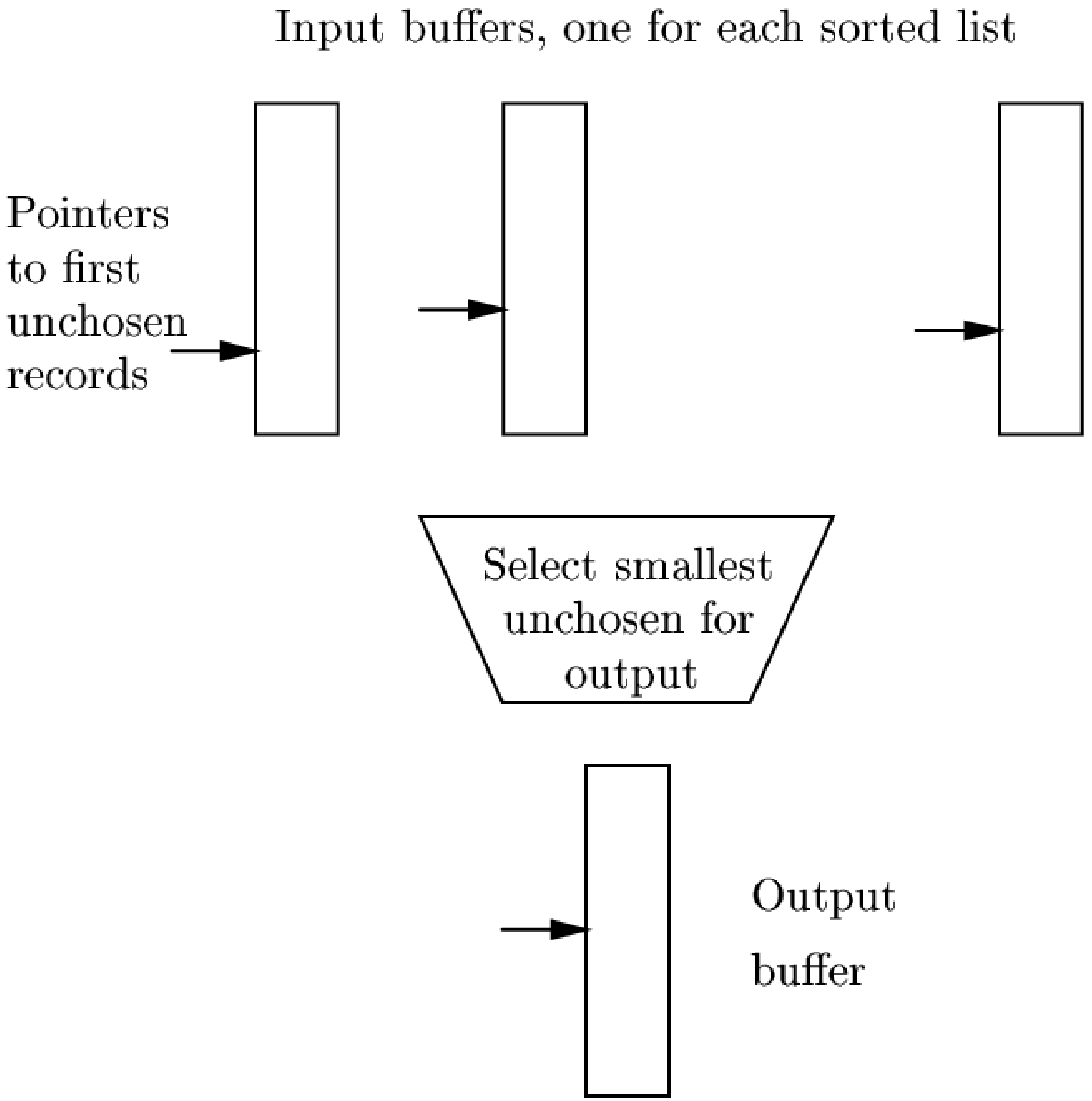
- Secondary memory variant (2PMMS) operates in a small number of *passes;*
  - in one pass every record is read into main memory once and written out to disk once.

- 2PMMS: 2 reads + 2 writes per block.

Phase 1

1. Fill main memory with records.
2. Sort using favorite main memory sort.
3. Write sorted sublist to disk.
4. Repeat until all records have been put into one of the sorted lists.

Phase 2


Input buffers, one for each sorted list
Pointers to first unchosen records
Select smallest unchosen for output
Output buffer

Phase 2 in words

- Use one buffer for each of the sorted sublists and one buffer for an output block.
- Initially load input buffers with the first blocks of their respective sorted lists.
- Repeatedly run a competition among the first unchosen records of each of the buffered blocks.
  - Move the record with the least key to the output block; it is now “chosen.”
- Manage the buffers as needed:
  - If an input block is exhausted, get the next block from the same file.
  - If the output block is full, write it to disk.

Toy Example

- 24 tuples with keys:
  - 12 10 25 20 40 30 27 29 14 18 45 23 70 65 35 11 49 47 22 21 46 34 29 39
- Suppose 1 block can hold 2 tuples.
- Suppose main memory (MM) can hold 4 blocks i.e. 8 tuples.

Phase 1.

- Load 12 10 25 20 40 30 27 29 in MM, sort them and write the sorted sublist: 10 12 20 25 27 29 30 40
- Load 14 18 45 23 70 65 35 11 in MM, sort them and write the sorted sublist: 11 14 18 23 35 45 65 70
- Load 49 47 22 21 46 34 29 39 in MM, sort them and write the sorted sublist: 21 22 29 34 39 46 47 49

Toy example (continued)

Phase 2.

Sublist 1: 10 12 20 25 27 29 30 40

Sublist 2: 11 14 18 23 35 45 65 70

Sublist 3: 21 22 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:

Input Buffer2:

Input Buffer3:

Output Buffer:

Sorted list:

Toy example (continued)

Phase 2.

Sublist 1: 20 25 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 10 12

Input Buffer2: 11 14

Input Buffer3: 21 22

Output Buffer:

Sorted list:

Toy example (continued)

Phase 2.

Sublist 1: 20 25 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 12

Input Buffer2: 11 14

Input Buffer3: 21 22

Output Buffer: 10

Sorted list:

Toy example (continued)

Phase 2.

Sublist 1: 20 25 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 12

Input Buffer2: 14

Input Buffer3: 21 22

Output Buffer: 10 11

Sorted list:

Toy example (continued)

Phase 2.

Sublist 1: 20 25 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 12

Input Buffer2: 14

Input Buffer3: 21 22

Output Buffer:

Sorted list: 10 11

Toy example (continued)

Phase 2.

Sublist 1: 20 25 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1:

Input Buffer2: 14

Input Buffer3: 21 22

Output Buffer: 12

Sorted list: 10 11

Toy example (continued)

Phase 2.

Sublist 1: 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 20 25

Input Buffer2: 14

Input Buffer3: 21 22

Output Buffer: 12

Sorted list: 10 11

Toy example (continued)

Phase 2.

Sublist 1: 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 20 25

Input Buffer2:

Input Buffer3: 21 22

Output Buffer: 12 14

Sorted list: 10 11

Toy example (continued)

Phase 2.

Sublist 1: 27 29 30 40

Sublist 2: 18 23 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 20 25

Input Buffer2:

Input Buffer3: 21 22

Output Buffer:

Sorted list: 10 11 12 14

Toy example (continued)

Phase 2.

Sublist 1: 27 29 30 40

Sublist 2: 35 45 65 70

Sublist 3: 29 34 39 46 47 49

Main Memory (4 buffers)

Input Buffer1: 20 25

Input Buffer2: 18 23

Input Buffer3: 21 22

Output Buffer:

We continue in this way until the sorted sublists are finished and we get the whole sorted list of tuples.

Sorted list: 10 11 12 14

Real Life Example

- 10,000,000 tuples of 160 bytes = 1.6 GB file.
  - Stored on Megatron 747 disk, with blocks of 16K, each holding 100 tuples
  - Entire file takes 100,000 blocks

- 100MB available main memory
  - The number of blocks that can fit in 100MB of memory (which, recall, is really 100 x 2^{20} bytes), is

    100 x 2^{20} / (16*2^{10}) or **6400** blocks $\approx$1/16th of file.

Analysis – Phase 1

- 6400 of the 100,000 blocks will fill main memory.
- We thus fill the memory $\lceil 100{,}000/6{,}400 \rceil = 16$ times, sort the records in main memory, and write the sorted sublists out to disk.

- How long does this phase take?
- We read each of the 100,000 blocks once, and we write 100,000 new blocks. Thus, there are 200,000 disk I/O's for 200,000*11ms = 2200 seconds, or **37 minutes**.

Avg. time for reading a block.

Analysis – Phase 2

- Every block holding records from one of the sorted lists is read from disk exactly once.
  - Thus, the total number of block reads is 100,000 in the second phase, just as for the first.
- Likewise, each record is placed once in an output block, and each of these blocks is written to disk once.
  - Thus, the number of block writes in the second phase is also 100,000.

- We conclude that the second phase takes another **37 minutes**.
- Total: Phase 1 + Phase 2 = **74 minutes.**

How Big Should Blocks Be?

- We have assumed a 16K byte block in our analysis.
- Would a larger block size be advantageous?

- If we doubled the size of blocks, we would halve the number of disk I/O's. But, how much a disk I/O would cost in such a case?

- Recall it takes about 0.13 ms for transfer time of a 16K block and 10.63 ms for average seek time and rotational latency.

- Now, the only change in the time to access a block would be that the transfer time increases to **0.13*2=0.26** ms, i.e. only slightly more than before.

- We would thus approximately halve the time the sort takes.

Another example: Block Size = 512K

- For a block size of 512K the transfer time is 0.13***32**=4.16 milliseconds.

- Average block access time would be

  10.63 + 4.16 approx. 15 ms, (as opposed to 11ms we had)

- However, now a block can hold 100***32** = 3200 tuples and the whole table will be 10,000,000 / 3200 = 3125 blocks (as opposed to 100,000 blocks we had before).

- Thus, we would need only 3125 * 2 * 2 disk I/Os for 2PMMS for a total time of 3125 * 2 * 2 * 15 = 187,500ms or about 3.12 min.

- Speedup: 74 / 3.12 = 23 fold.

Reasons to limit the block size

1. First, we cannot use blocks that cover several tracks effectively.
2. Second, small tables would occupy only a fraction of a block, so large blocks would waste space on the disk.
3. Third, the larger the blocks are, the fewer records we can sort by 2PMMS (see next slide).

- Nevertheless, as machines get more memory and disks more capacious, there is a tendency for block sizes to grow.

How many records can we sort?

1. Block size is B bytes.
2. Main memory available for buffering blocks is M bytes.
3. Record is R bytes.

- Number of main memory buffers = M/B blocks
- We need one output buffer, so we can actually use $(M/B)-1$ input buffers.

- How many sorted sublists can we merge?
- $(M/B)-1$.

- What's the total number of records we can sort?
- Each time we fill in the memory with M/R records.
- Hence, we are able to sort $(M/R)*[(M/B)-1]$ or approximately M^2/RB.

If we use the parameters in the example about 2PMMS we have:

M=100MB = 100,000,000 Bytes = 10^8 Bytes

B = 16,384 Bytes

R = 160 Bytes

So, $M^2/RB = (10^8)^2 / (160 * 16{,}384)$ = 4.2 billion records, or 2/3 of a TeraByte.

Sorting larger relations

- If our relation is bigger, then, we can use 2PMMS to create sorted sublists of **M^2/RB** records.
- Then, in a third pass we can merge **(M/B)-1** of these sorted sublists.
- Thus, the third phase let's us sort
  - **$[(M/B)-1]*[M^2/RB] \approx M^3/RB^2$** records

- For our example, the third phase let's us sort 75 trillion records occupying 7500 Petabytes!!