

Storage Mechanics

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}

A DAY IN DATA

The exponential growth of data is undisputed, but the numbers behind this explosion - fuelled by internet of things and the use of connected devices - are hard to comprehend, particularly when looked at in the context of one day

500m

tweets are sent every day

Twitter



4PB

of data created by Facebook, including

350m photos

100m hours of video watch time

Facebook Research

DEMYSIFYING DATA UNITS

From the more familiar 'bit' or 'megabyte', larger units of measurement are more frequently being used to explain the masses of data

Unit	Value	Size
b bit	0 or 1	1/8 of a byte
B byte	8 bits	1 byte
KB kilobyte	1,000 bytes	1,000 bytes
MB megabyte	1,000 ² bytes	1,000,000 bytes
GB gigabyte	1,000 ³ bytes	1,000,000,000 bytes
TB terabyte	1,000 ⁴ bytes	1,000,000,000,000 bytes
PB petabyte	1,000 ⁵ bytes	1,000,000,000,000,000 bytes
EB exabyte	1,000 ⁶ bytes	1,000,000,000,000,000,000 bytes
ZB zettabyte	1,000 ⁷ bytes	1,000,000,000,000,000,000,000 bytes
YB yottabyte	1,000 ⁸ bytes	1,000,000,000,000,000,000,000,000 bytes

*A lowercase "b" is used as an abbreviation for bits, while an uppercase "B" represents bytes.

65bn

messages sent over WhatsApp and two billion minutes of voice and video calls made

Facebook

294bn

billion emails are sent

Radical Group

320bn

emails to be sent each day by 2021

306bn

emails to be sent each day by 2020

3.9bn

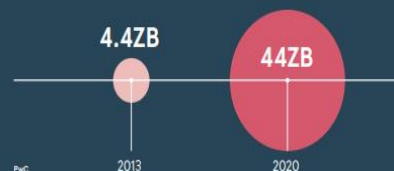
people use emails

4TB

of data produced by a connected car

Intel

ACCUMULATED DIGITAL UNIVERSE OF DATA



PwC

Searches made a day

5bn

Searches made a day from Google

3.5bn

Smart Insights



463EB

of data will be created every day by 2025

idc

95m

photos and videos are shared on Instagram

Instagram Business

28PB

to be generated from wearable devices by 2020

Statista

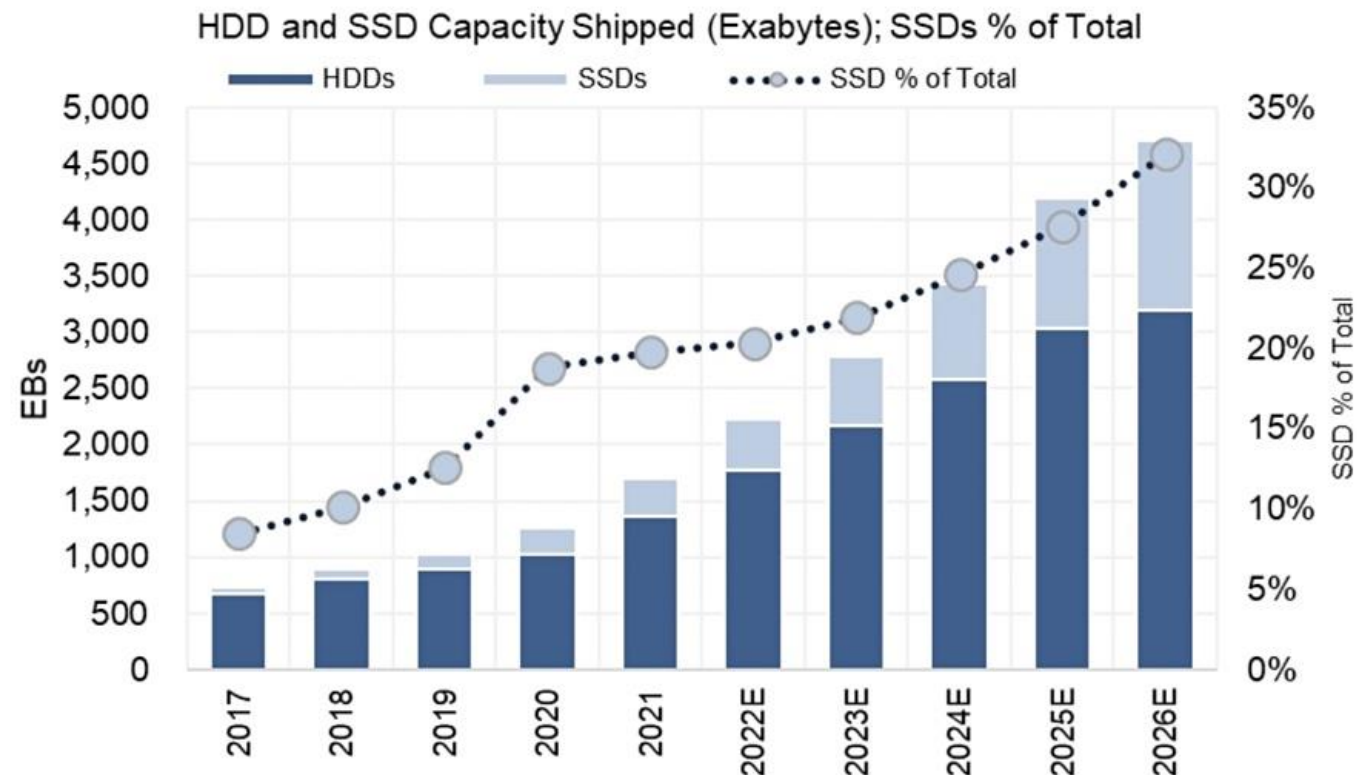


2021 *This Is What Happens In An Internet Minute*



Storage mechanics

- HDD, SSD: Which will reign supreme?



Source: Gartner; Wells Fargo Securities, LLC.

A Brief History of Disk Drives

IBM introduced it in **1956**.

- First HDD was the **size of a car**
- Only had **3.75 megabytes** of space.
- It cost **\$300,000**.

To put in perspective:

a **15-second video clip** taken from an older **iPhone 7 Plus** can take up **15MBs** of storage

a **single still photo** taken from an **iPhone 6S Plus** can already take **2.93MB**



Photo Source: Back Blaze

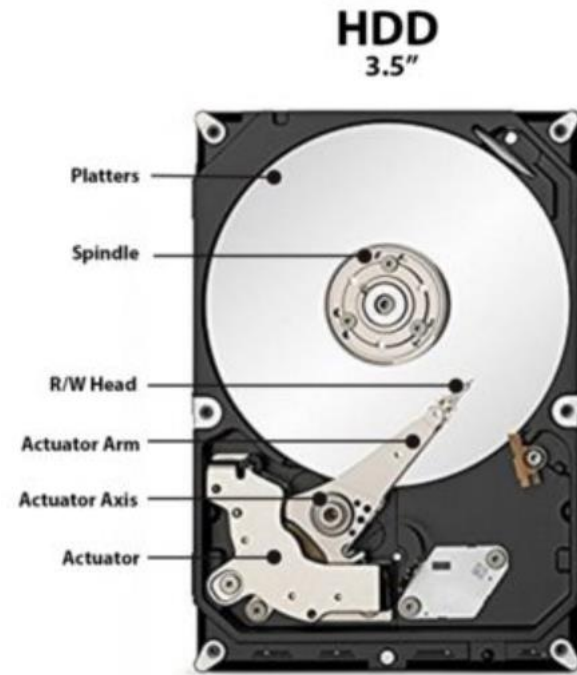
HDDs today

Today, one can buy an **HDD** with **10 TBs** of storage.

Although the amount of storage space has increased tremendously, the **simple mechanism** remains **similar to the original design**.

Traditional HDD **spins** and **rotates** platters.
A **fixed arm** that floats above reads and writes information onto the platters.

A typical hard drive spins at either **5400** revolutions per minute (RPM) or **7200** RPM.



SSDs

Solid State Drives have **no moving parts**.

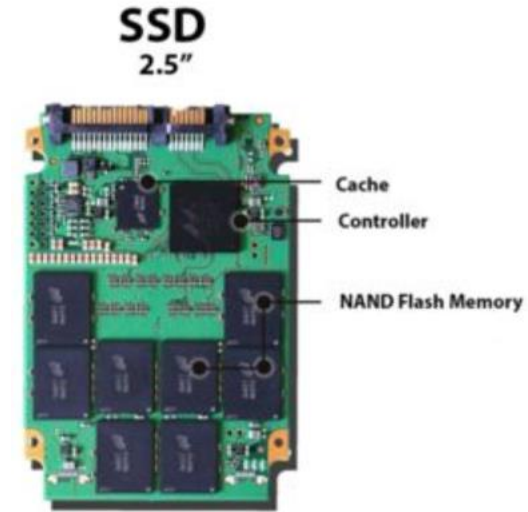
Faster than HDDs for **random access**.

Capacities.

Nimbus Data: ExaDrive DC100 at **100TB**.

Typically, **4TB**.

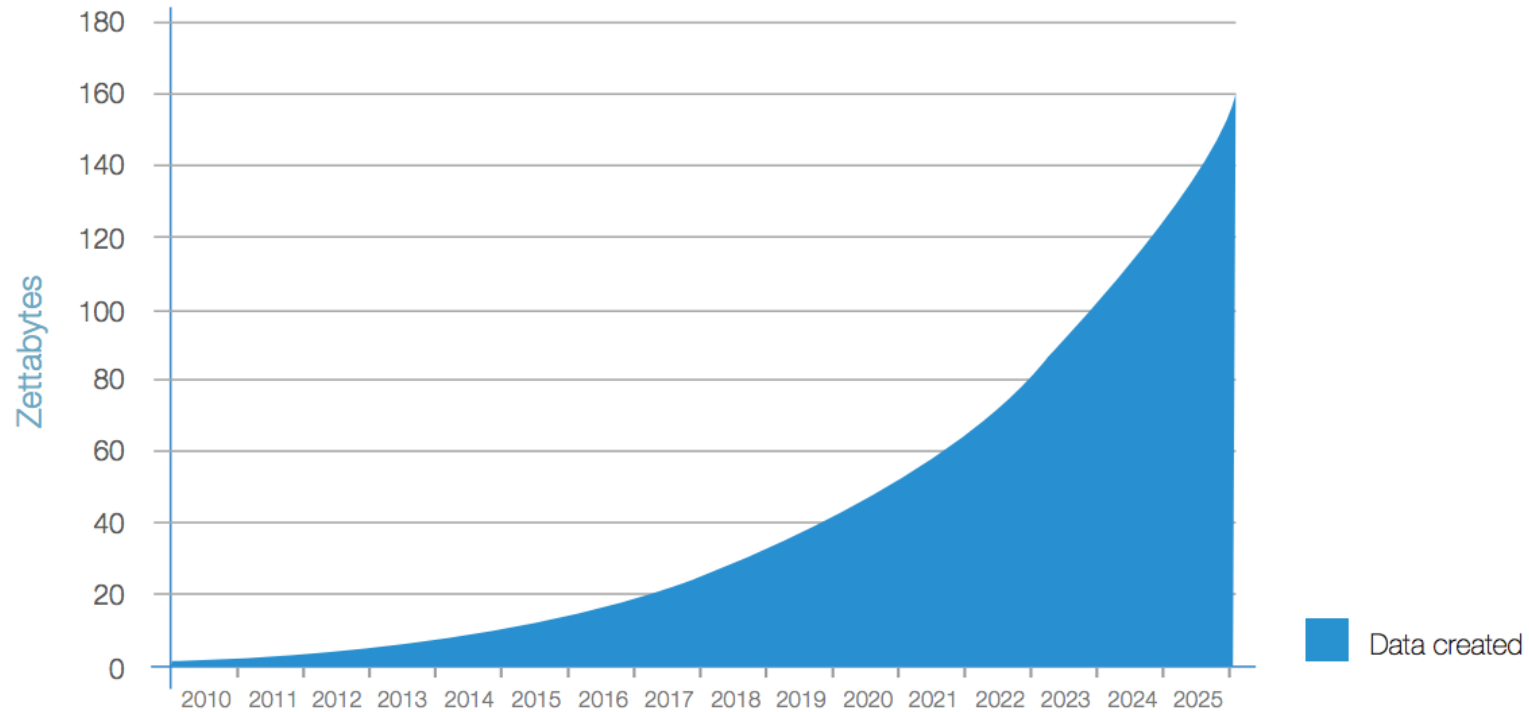
Also very **energy efficient**.



Similarities

- Both of HDDs and SSDs will eventually **crash**.
 - HDDs: have moving parts that will merely fail.
 - SSDs: each time information is written on it, the more it becomes **degraded**.
- **Longevity** for both HDDs and SSDs is almost identical.
 - HDDs last for about 6 years,
 - SSDs last for about 5-7 years.

Annual Size of the Global Datasphere



Source: IDC's Data Age 2025 study, sponsored by Seagate, April 2017

Price

- Average cost of a 1TB HDD is about \$40 – \$50.
- A 1TB SSD will average about \$250.
- HDDs will still give you the most cost per bit of storage than SSDs.

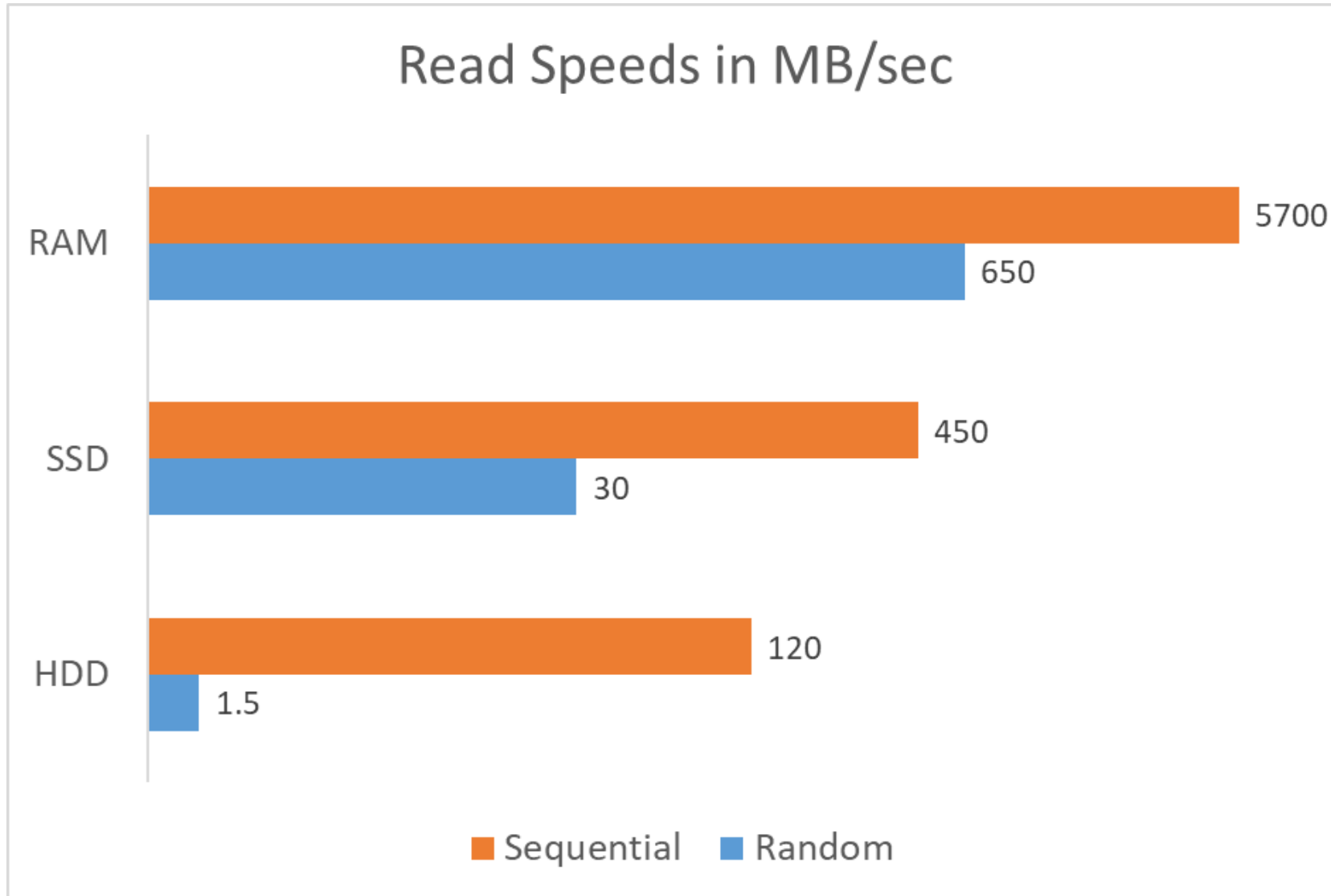


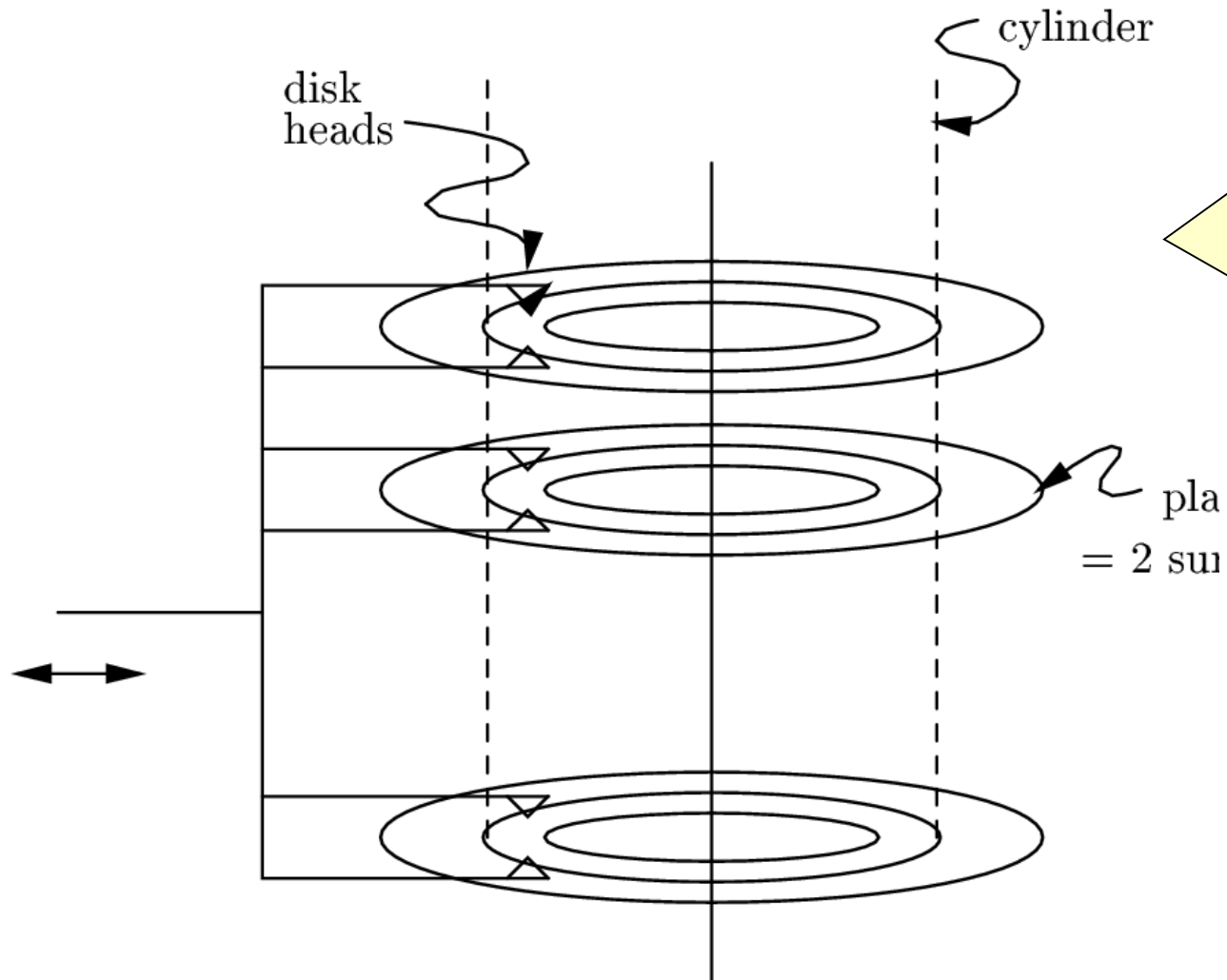
Chart is in log scale

I/O model of computation

- **Block** is the smallest unit of read/write from/to external storage (disk).
 - Typical size: 16KB
 - Typical time for random access of a block in HDD: 11 ms
 - Typical time for sequential access of a block in HDD: 0.13 ms
- I/O = read or write of a block is very expensive;
- 1,000,000 machine instructions in the time to do one random disk I/O.

HDD USE CASE

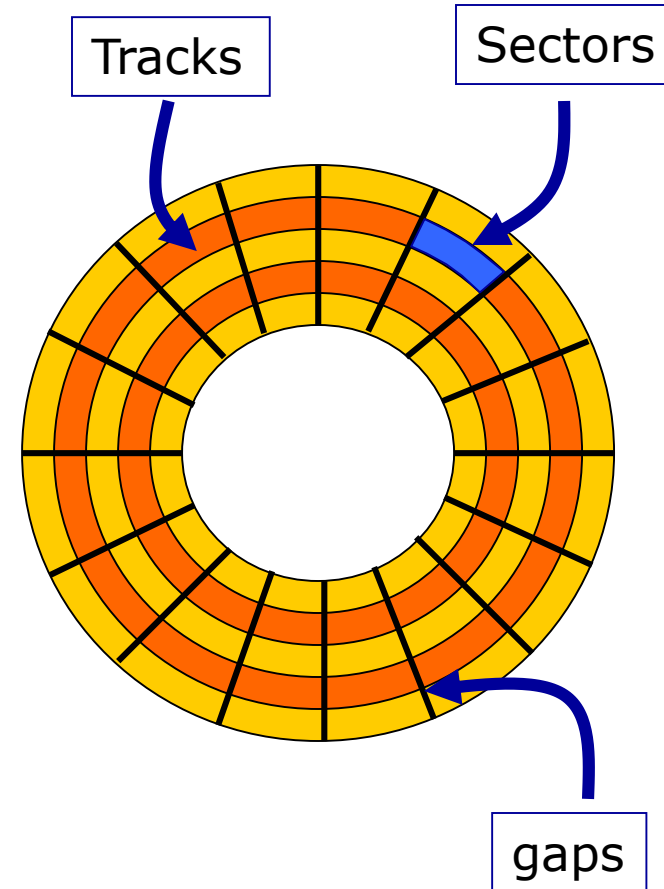
Disks



- **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 covered with concentric **tracks**.
 - Tracks at a common radius = **cylinder**.
 - Cylinder can be read quickly, no movement of heads.
 - Typical: $2^{16} = 65,536$ cylinders
- Tracks divided into **sectors** by **gaps** ($\approx 10\%$ of track).
 - Typical track: 256 sectors.
 - Typical sector: 4096 bytes (4KB).
- Sectors are grouped into **blocks**.
 - Typical block: 16KB = 4 x 4096byte sectors.



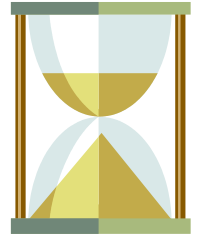
MEGATRON 747 Disk Parameters

- $2^4 = 16$ surfaces (8 platters).
- $2^{16} = 65,536$ tracks per surface.
- $2^8 = 256$ sectors per track.
- $2^{12} = 4096 = 4\text{KB}$ per sector.

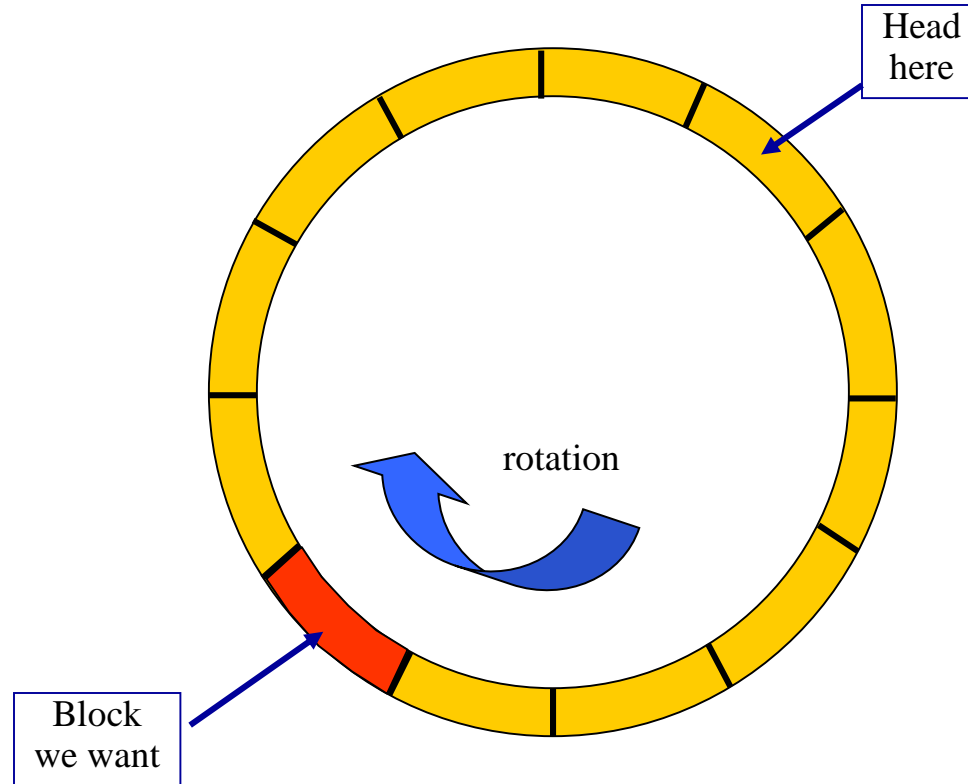
$$\text{Capacity} = 2^4 * 2^{16} * 2^8 * 2^{12} = 2^{40} = \text{1TB}$$

Disk access time

- **Latency of disk** (access time): Time to read/write a block.
- Main components of access time are:
 - **Seek time** = time to move heads to proper cylinder (track).
 - **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.
- Heads must pass over 4 sectors and the 3 gaps between them.
- Gaps represent 10% of the circle and sectors the remaining 90%,
 - 36 degrees are occupied by gaps and
 - 324 degrees by the sectors.
 - 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
- Total degrees covered by 3 gaps and 4 sectors is:
 $3 \times 0.14 + 4 \times 1.265 = 5.48$ degrees
- Transfer time is thus $(5.48/360) \times 8.33 = 0.13$ 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

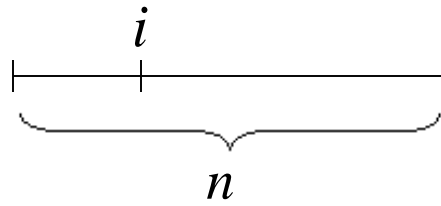
- **Transfer time** is always **0.13** milliseconds.
- **Average rotational delay** is the time to rotate disk **half way around**, or **4.17** ms.
- What about the **average seek time**?
 - Is it just the time to move across **half the tracks**?
 - Yes, if the heads are at the outermost or innermost cylinder.
 - No, otherwise.

AVG time to read a 16,384-byte block

Average seek time

- Assume the next request for a block to read will be at any of the 16,384 cylinders with equal probability.
- If the heads are currently at cylinder 1 or cylinder 65,536, the average number of tracks to move is 8192 tracks (1/2 of the tracks)
- If the heads are currently at middle cylinder, 8192, they are equally likely to move in or out, and either way, they will move on average about a quarter of the tracks, 4096 (1/4 of the tracks).
- It can be shown that on average the heads need to move about **1/3 of the tracks**.

AVG time to read a 16,384-byte block



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

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

Probability the block is on the **left**

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

Probability the block is on the **right**

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

Optional Material

AVG

$$= \frac{1}{n} \int_0^n \frac{i^2}{2n} + \frac{(n-i)^2}{2n} di$$

$$= \frac{1}{2n^2} \int_0^n i^2 di + \frac{1}{2n^2} \int_0^n (n-i)^2 di$$

$$= \frac{1}{2n^2} \int_0^n i^2 di + \frac{1}{2n^2} \int_{-n}^0 j^2 dj$$

substituting $i - n = j$

$$= \frac{1}{2n^2} \frac{i^3}{3} \Big|_0^n + \frac{1}{2n^2} \frac{j^3}{3} \Big|_{-n}^0$$

$$= \frac{1}{2n^2} \frac{n^3}{3} + \frac{1}{2n^2} \frac{n^3}{3}$$

Optional Material

$$= \frac{n}{3}$$

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

AVG time to read a 16,384-byte block

- Transfer time is always **0.13** milliseconds and
- Average rotational delay is the time to rotate the disk half way around, or **4.17** milliseconds.
- Average seek time is: $1 + (65536/3) * (1/4000) = \mathbf{6.46}$ ms
- Total: $6.46 + 4.17 + 0.13 = \mathbf{10.76}$ ms (or about **11 ms**)

Writing and Modifying Blocks

- Writing same as reading, unless we verify written blocks.
- “one I/O” = “one block read or write”
- Modifying a block requires:
 1. Read the block into main memory.
 2. Modify the block there.
 3. Write the block back to disk.

SORTING IN EXTERNAL STORAGE

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.

Merge Sort

- 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.
- Merge Sort in its original form is not very good in disk I/O model.
 - $\log_2 n$ passes,
 - so each record is read/written from disk $\log_2 n$ times.
 - E.g. if we have 10 million records, we need 23 passes.

TwoPhase, Multiway Merge Sort

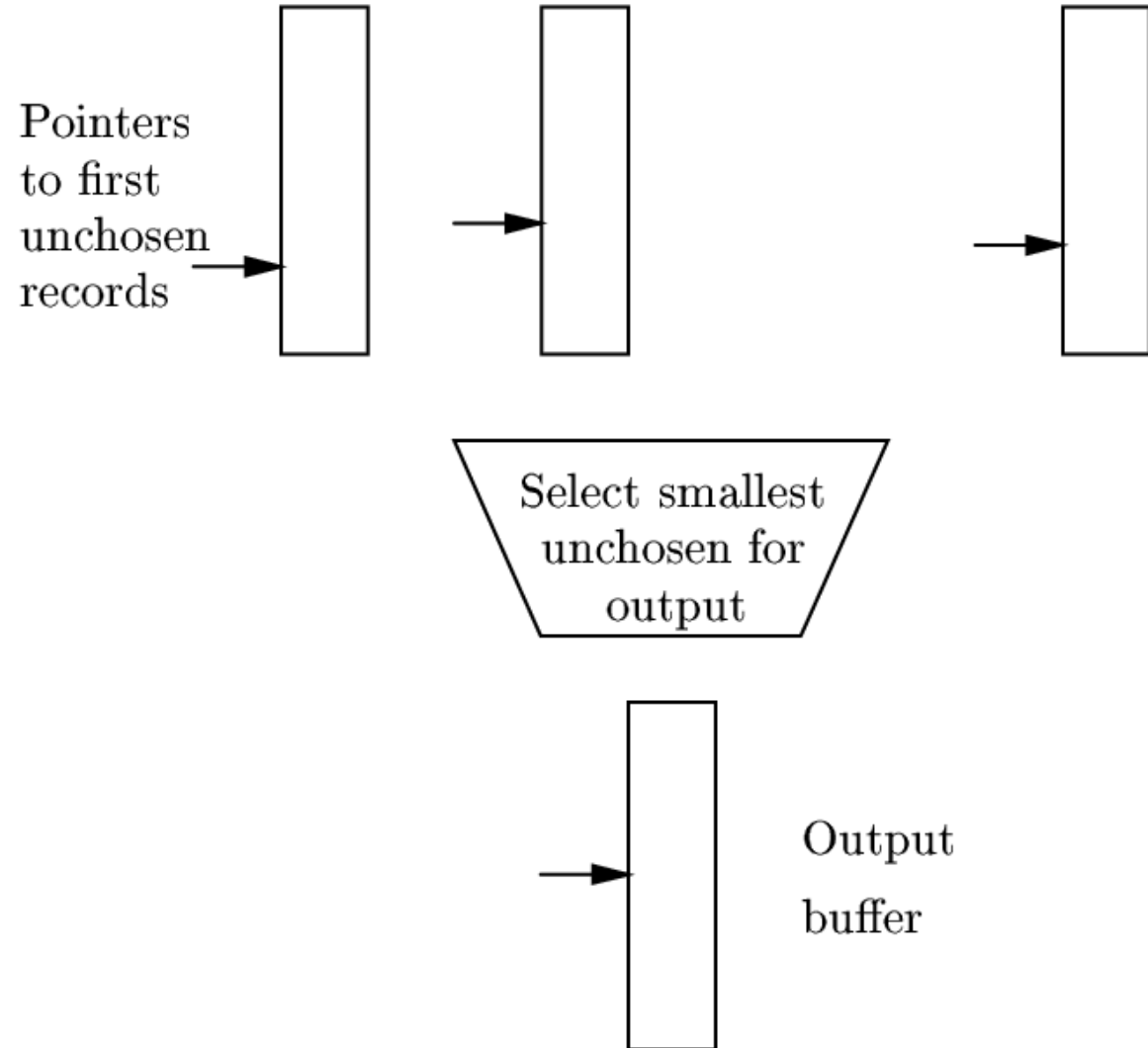
- Secondary memory variant (2PMMS) operates in a small number of *passes*;
 - in one pass every record is read into main memory once and written out to disk once.
- 2PMMS: 2 reads + 2 writes per block.

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



We will use “sorted list” or “sorted sublist” interchangeably.

Phase 2 in words

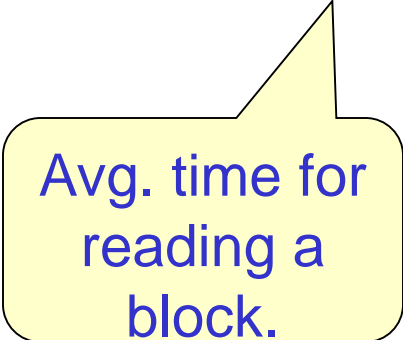
- Use one buffer for each of the sorted sublists and one buffer for an output block.
- Initially load input buffers with the first blocks of their respective sorted lists.
- Repeatedly run a competition among the first unchosen records of each of the buffered blocks.
 - Move the record with the least 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.

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 100×2^{20} bytes), is $100 \times 2^{20} / (16 \times 2^{10})$ or 6400 blocks $\approx 1/16^{\text{th}}$ 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 $200,000 * 11\text{ms} = 2200$ seconds, or **37 minutes**.



Avg. time for
reading a
block.

Analysis – Phase 2

- Each 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.
- 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**.
- So, **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 0.13 ms for transfer time of a 16K block and 10.63 ms for average seek time and rotational latency.
- **Only the transfer time changes.**
- It increases to $0.13 * 2 = 0.26$ ms, i.e. slightly more than before.
- We would thus approximately halve the time the sort takes.

Another example: Block Size = 512K

- Now the transfer time is $0.13 * 32 = 4.16$ milliseconds.
- Total = $10.63 + 4.16 \approx 15$ ms (as opposed to 11ms we had)
- However, table will be $10,000 / 32 = 3125$ blocks (as opposed to 10,000 blocks we had)
- Number of I/O's for 2PMMS is: $3125 * (2 + 2)$
- Total time for 2PMMS is: $3125 * 4 * 15 = 187,500\text{ms} \approx 3.12$ min.
- Speedup: $74 / 3.12 = 23$ fold.
- Well we are completely neglecting the time for main memory operations...(somewhat optimistic!)

Reasons to limit the block size

1. Small tables would occupy only a fraction of a block, so large blocks would waste space on the disk.
2. 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:

$M=100\text{MB} = 100,000,000 \text{ Bytes} = 10^8 \text{ Bytes}$ (100*2²⁰ more precisely)

$B = 16,384 \text{ Bytes}$

$R = 160 \text{ Bytes}$

So, $M^2/RB = (100*2^{20})^2 / (160 * 16,384) = 4.2 \text{ billion records, or } 2/3 \text{ of a TeraByte.}$

Sorting larger files

- If our file 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!!

Toy example for 2PMMS

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)

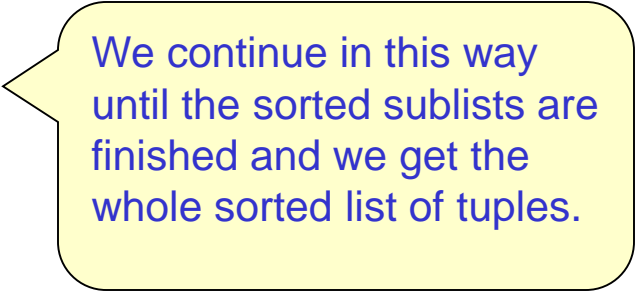
Input Buffer1: 20 25

Input Buffer2: 18 23

Input Buffer3: 21 22

Output Buffer:

Sorted list: 10 11 12 14



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