Tutorial 1: Organisation, Disks and Files

Implementation of Databases (DBS2)

Arik Ermshaus

Table of Contents

- **Organisation**
- Exercise Sheet 1
- Magnetic Disks
- RAID
- External Sorting

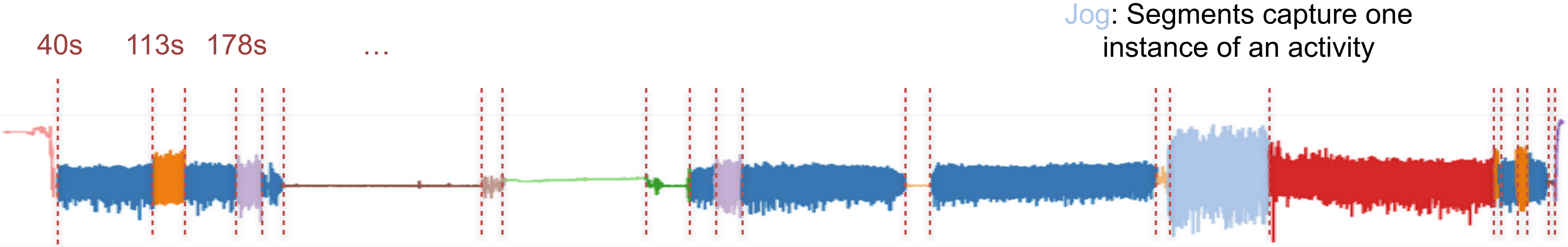
My Background

- Scientific employee and PhD student at “Knowledge management in bioinformatics” (WBI) chair, led by Prof. Leser

- My career (so far …)
  - Bachelor / master in computer science at HU Berlin
  - Working student at IVU Traffic Technologies / WBI

- Research interests
  - Unsupervised time series analysis
  - Specifically: segmentation and summarisation of time series
  - See published papers, open source code and data

Jog: Segments capture one instance of an activity



From Walk to Wait: Splits define the activity transitions

20 min mobile phone Y-acceleration recording (50 Hz) of student commuting to university by train.

Organisation of the Tutorial

- Goal: Exercise concepts, algorithms from lecture content
  - Theory: Calculations, analyses and scenarios
  - Practice: Implementation of DBS parts in C++

- Organisation via Moodle course
  - Link: https://hu.berlin/dbs223 (key: btree23)
  - Communication, announcements, forum
  - Release of all slides, exercise sheets, materials
  - Submission of your task solutions
  - Everybody must be registered

Tutorial appointments

- In total: 15 tutorials (of 2 types)
  - Presentation of exercise sheets and results
  - Q&A sessions (optional)
- Appointments are weekly (1 date per group)

Group	Weekday	Time	Room
1	Tuesday	09-11	RUD 25, 3.101
2	Thursday	09-11	RUD 25, 3.101

Tutorial appointments

Week	Topic
16.10 - 20.10	-
23.10 - 27.10	**Organisation, Exercise Sheet 1**
30.10 - 03.11	Q&A
06.11 - 10.11	Q&A
13.11 - 17.11	Exercise Sheet 2
20.11 - 24.11	Q&A
27.11 - 01.12	Q&A
04.12 - 08.12	Exercise Sheet 3
11.12 - 15.12	Q&A
18.12 - 22.12	Q&A
25.12 - 29.12	-
01.01 - 05.01	-
08.01 - 12.01	Exercise Sheet 4
15.01 - 19.01	Q&A
22.01 - 26.01	Q&A
29.01 - 02.02	Exercise Sheet 5
05.02 - 09.02	Q&A
12.02 - 16.02	Exam preparation

Disclaimer: Timetable is provisional, and will (probably) change!

Exercise Sheets

- 5 exercise sheets with 30 points each
  - 2-3 weeks to complete the tasks
  - First sheet date: **24th Oct.** (release), **13th Nov.** (submission)
- Textual problems
  - Always justify your solutions
  - Be precise, write only key points
  - If you make assumptions, name and justify them
- Math problems
  - Always provide calculation paths
  - Use powers (of two or ten) and shorten fractures
  - Practice mental arithmetics ;-)

Exercise Sheets (contd.)

- Submit written assignments in separate PDFs per task
  - Use following naming schema:
    - A<Task>-<Person1>-<Person2><Person3>.pdf
    - Example: A03-Musterfrau-Mustermann-Beispiel.pdf for task 3 from Erika Musterfrau, Peter Mustermann and Mark Beispiel
  - Hand in before midnight **until 23:59 o'clock**
- Submit programming tasks (C++) as .cpp / .h files
  - Use predefined template files
  - Test your solutions on gruenau2-6 with "cmake" beforehand!
  - Write names, CMS user names, group ids in each file as comment

C++ in the Tutorial

- Good knowledge of C++ is a prerequisite for this course
    - We will **not** go into syntax, semantics, libraries, etc.
    - Many (external) resources to learn C++ exist, e.g. codeacademy
- Integrated development environments
    - Microsoft Visual Studio (Code)
    - JetBrains CLion
- Compile C++ projects
    - CMake files to setup build environment
    - make / gcc for compilation
    - Use gruenau2-6 as reference!
- Example: Compile and run your exercise solution
    - “mkdir build” (create build directory)
    - “cd build” (move to the directory)
    - “cmake ..” (generate build system files)
    - “make” (compile code)
    - “./Task_x” (run program)

How to get the „Übungsschein“?

- Registration in Agnes/Moodle is required
  - Formation of groups with three students
  - Groups can be spread over several exercise dates
  - Important messages sent by email via Moodle / Agnes
- Examination admission
  - 50% of the exercise points (75) required
  - Present results of at least one programming exercise
    - Explain how you solved the exercise
    - Contact me which exercise you want to present
    - Presentations are at dates where new exercise is presented
- 0 points for submissions without registration in Moodle / Agnes, invalid group size, suspected copying of other group's solutions, non-executable programs (test on gruenau2-6)

Checklist for you!

- Do you already have the “Übungsschein”?
  - If yes: you still can hand-in exercises for practice
- Are you registered in Agnes?
  - If not: Write name & student number on pass-through paper
- Enrol in Moodle course
  - Link: https://hu.berlin/dbs223 (key: btree23)
- Find 2 group partners
  - Use tutorial and forum
  - Exchange contact information

Table of Contents

- Organisation
- **Exercise Sheet 1**
- Magnetic Disks
- RAID
- External Sorting

Table of Contents

Decimal vs Binary Units

Dezimalpräfixe			Binärpräfixe	
Name (Symbol)	Bedeutung[G 1]	Unterschied (gerundet)	IEC-Name (IEC-Symbol)	Bedeutung
Kilobyte (kB)[G 2]	10^3 Byte = 1.000 Byte	2,40 %	Kibibyte (KiB)[G 3]	2^{10} Byte = 1.024 Byte
Megabyte (MB)	10^6 Byte = 1.000.000 Byte	4,86 %	Mebibyte (MiB)	2^{20} Byte = 1.048.576 Byte
Gigabyte (GB)	10^9 Byte = 1.000.000.000 Byte	7,37 %	Gibibyte (GiB)	2^{30} Byte = 1.073.741.824 Byte
Terabyte (TB)	10^{12} Byte = 1.000.000.000.000 Byte	9,95 %	Tebibyte (TiB)	2^{40} Byte = 1.099.511.627.776 Byte
Petabyte (PB)	10^{15} Byte = 1.000.000.000.000.000 Byte	12,6 %	Pebibyte (PiB)	2^{50} Byte = 1.125.899.906.842.624 Byte
Exabyte (EB)	10^{18} Byte = 1.000.000.000.000.000.000 Byte	15,3 %	Exbibyte (EiB)	2^{60} Byte = 1.152.921.504.606.846.976 Byte
Zettabyte (ZB)	10^{21} Byte = 1.000.000.000.000.000.000.000 Byte	18,1 %	Zebibyte (ZiB)	2^{70} Byte = 1.180.591.620.717.411.303.424 Byte
Yottabyte (YB)	10^{24} Byte = 1.000.000.000.000.000.000.000.000 Byte	20,9 %	Yobibyte (YiB)	2^{80} Byte = 1.208.925.819.614.629.174.706.176 Byte

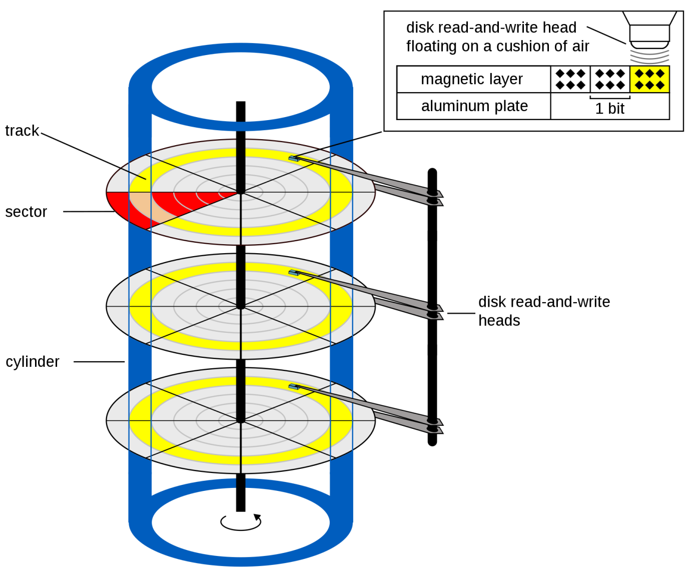
1. ↑ SI-Präfixe sind nur für SI-Einheiten standardisiert; Byte ist keine SI-Einheit
2. ↑ wird gelegentlich mit „KB" abgekürzt
3. ↑ wird gelegentlich mit „KB" abgekürzt, um den Unterschied zu „kB" zu kennzeichnen (nicht standardisiert)

In the exercises, read carefully which unit is asked for (e.g. GB vs GiB)

Recap: Magnetic Disks

- Magnetic disk contains multiple platters, each having 2 surfaces
- Vertically aligned heads move to requested spot and read/write data

- Common Definitions
  - **Track**: circle on surface
    - variables lengths
  - **Cylinder**: 3D circle on disk
    - Vertically aligned tracks
    - over all platters
  - **Sector**: section on track
    - fixed length
  - **Block**: 1 or more sectors
    - Depending on use case



Images Source: Wikipedia

Example: Magnetic Disks

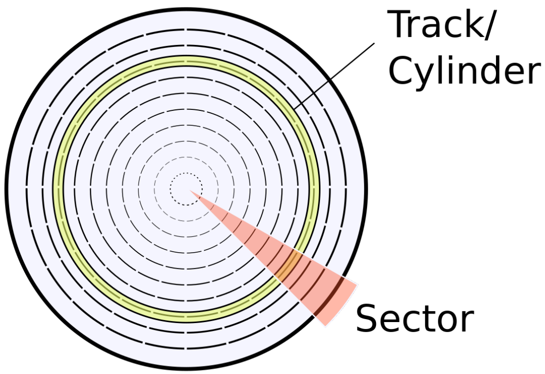


Image Source: Wikipedia

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

- Capacity of track: C_{track} = sector size x sectors per track
  - $C_{track} = 2^9 B \cdot 2^6 = 2^{15} B$
- Capacity of surface: $C_{surface}$ = C_{track} x tracks per surface
  - $C_{surface} = 2^{15} B \cdot 2^{11} = 2^{26} B$
- Capacity of disk: C_{disk} = $C_{surface}$ x number total of surfaces
  - $C_{disk} = 2^{26} B \cdot 2 \cdot 5 = 10 \cdot 2^{26} B$

Task 1: Magnetic Disks

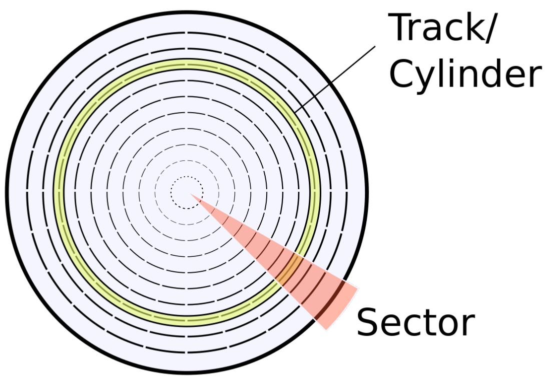


Image Source: Wikipedia

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

- **Question**: How many cylinders are on this disk?

(A) 512

(B) 1.024

(C) 2.048

(D) 4.096

Task 1: Magnetic Disks

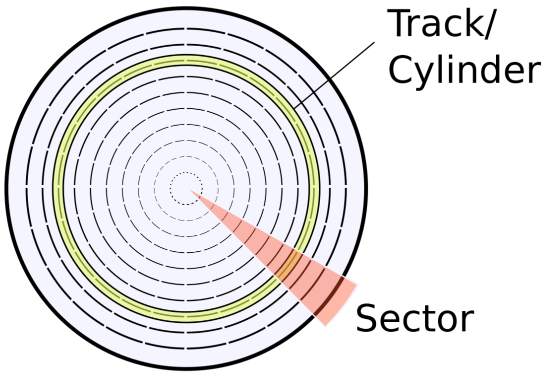


Image Source: Wikipedia

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

- **Question**: How many cylinders are on this disk?

(A) 512

(B) 1.024

(C) 2.048

(D) 4.096

Number of tracks = number of cylinders

Task 2: Magnetic Disks

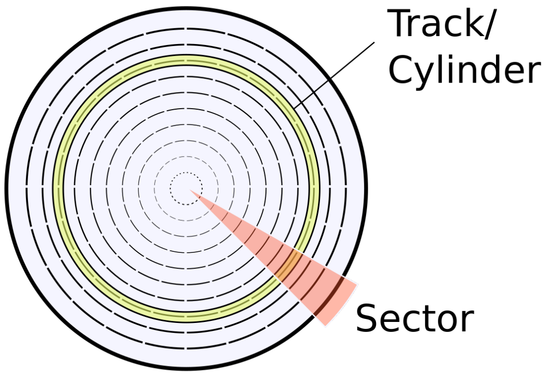


Image Source: Wikipedia

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

- **Question**: Which of these block sizes (in byte) are valid?

(A) 128

(B) 256

(C) 512

(D) 1.024

Task 2: Magnetic Disks

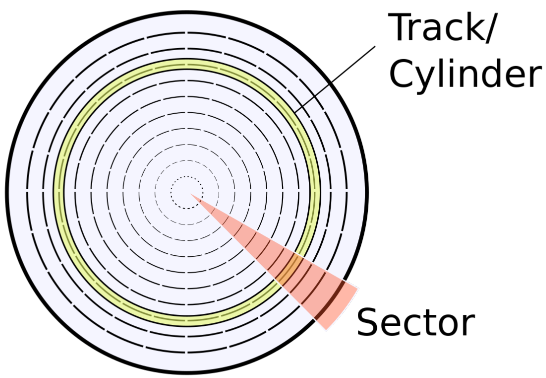


Image Source: Wikipedia

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

- **Question**: Which of these block sizes (in byte) are valid?

(A) 128

(B) 256

(C) 512

(D) 1.024

Block sizes can only be **integer multiples** of sector sizes

Recap: Read/Write Latency

can be ignored

Move head to right track

Rotate sector to head

Latency = Communication Time + Seek Time + Rotation Time

On average, we have to cross **1/3 of all tracks**

On average, we have to rotate **1/2 of the disk**

Example: Read/Write Latency

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

Property	Value
Rotational speed	5.000 R/min
Moving head over *n* tracks	(1 + 0.002 x n) ms
Block size	1.024 Byte

- Avg. seek time: $T_{seek} = 1 + 0.002 \cdot \frac{2.048}{3} ms \approx 2.355 ms$
- Avg. rotation time: $T_{rotate} = \frac{60.000}{5.000} ms \cdot \frac{1}{2} = 6 ms$
- Block read time: $T_{block} = \frac{60.000}{5.000} ms \cdot \frac{1}{32} = 0.375 ms$

Task 3: Read/Write Latency

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

Property	Value
Rotational speed	5.000 R/min
Moving head over *n* tracks	(1 + 0.002 x n) ms
Block size	1.024 Byte

- **Question**: What makes the difference between sequential and random reads from a disk?

(A) Seek time

(B) Rotation time

(C) Block read time

(D) Latency

Task 3: Read/Write Latency

Property	Value
Sector size	512 Byte
Sectors per track	64
Tracks per surface	2.048
# Platters	5

Property	Value
Rotational speed	5.000 R/min
Moving head over *n* tracks	(1 + 0.002 x n) ms
Block size	1.024 Byte

- **Question**: What makes the difference between sequential and random reads from a disk?

(A) Seek time

(B) Rotation time

(C) Block read time

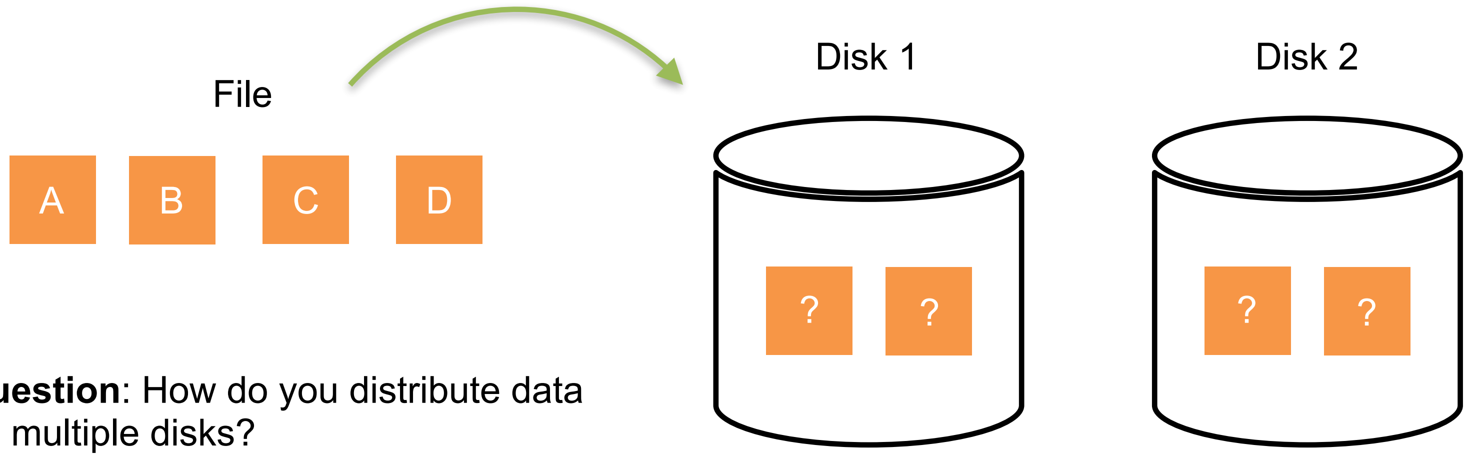
(D) Latency

Random reads need latency for every block

Table of Contents

- Organisation
- Exercise Sheet 1
- Magnetic Disks
- **RAID**
- External Sorting

Recap: RAID

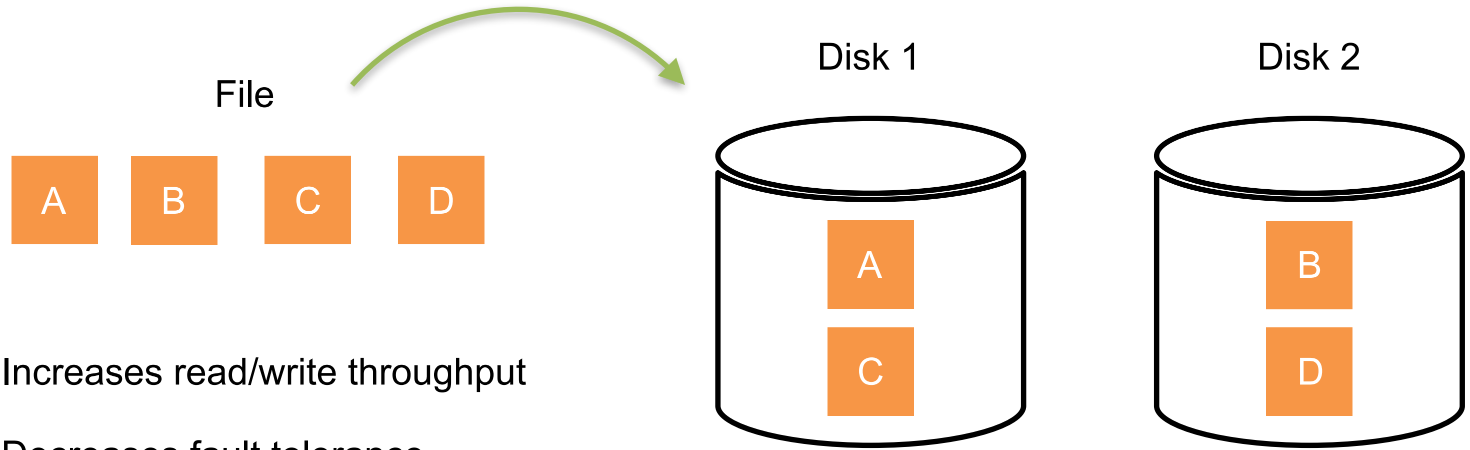


Question: How do you distribute data on multiple disks?

- RAID: redundant array of inexpensive disks
- Goals: Improve data fault tolerance, read/write performance
- Different levels: specification of data distribution over disks

Patterson, D. A., Gibson, G., & Katz, R. H. (1988, June). A case for redundant arrays of inexpensive disks (RAID). In Proceedings of the 1988 ACM SIGMOD international conference on management of data (pp. 109-116).

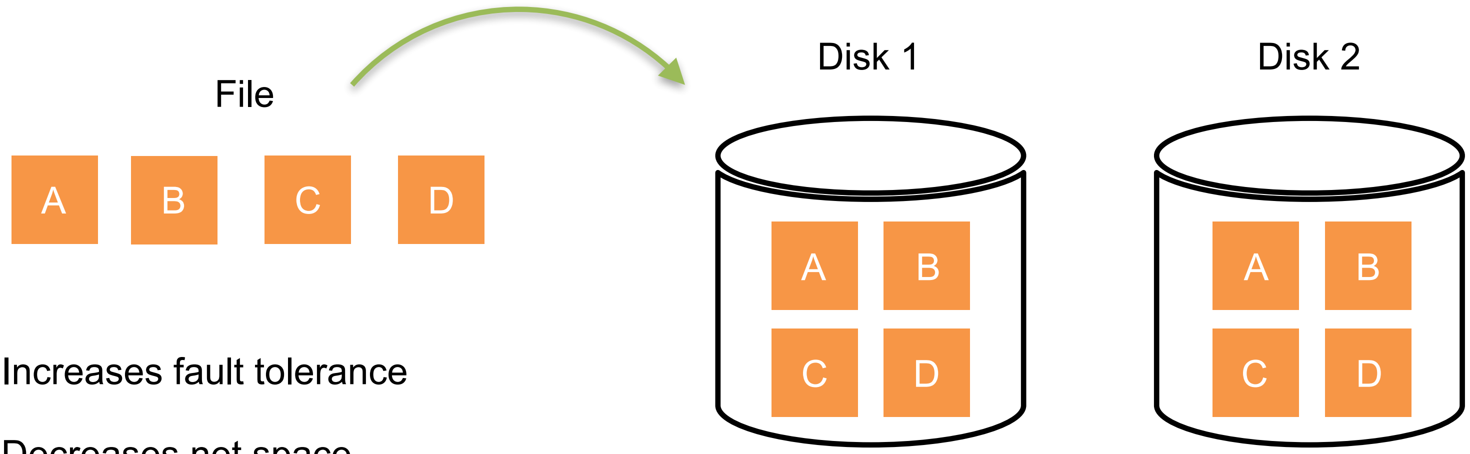
Example: RAID 0 (Striping)



- ✓ Increases read/write throughput
- ✗ Decreases fault tolerance

Property/Level	0				
Read Throughput	2x				
Write Throughput	2x				
Fault Tolerance	✗				
Net Space	1x				

Example: RAID 1 (Mirroring)

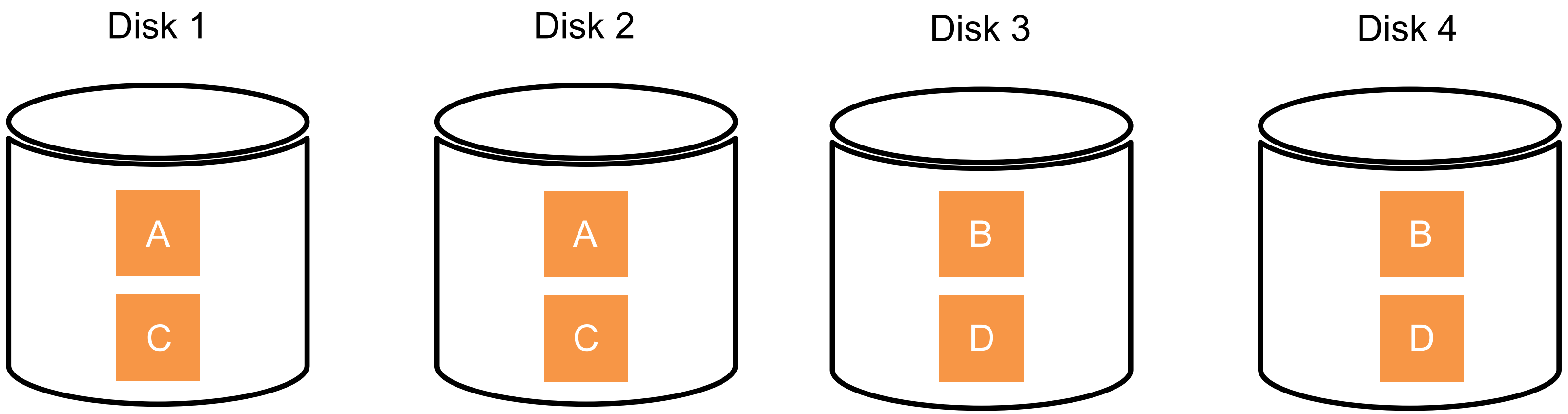


✓ Increases fault tolerance

✗ Decreases net space

Property/Level	0	1			
Read Throughput	2x	2x			
Write Throughput	2x	1x			
Fault Tolerance	✗	✓			
Net Space	1x	1/2x			

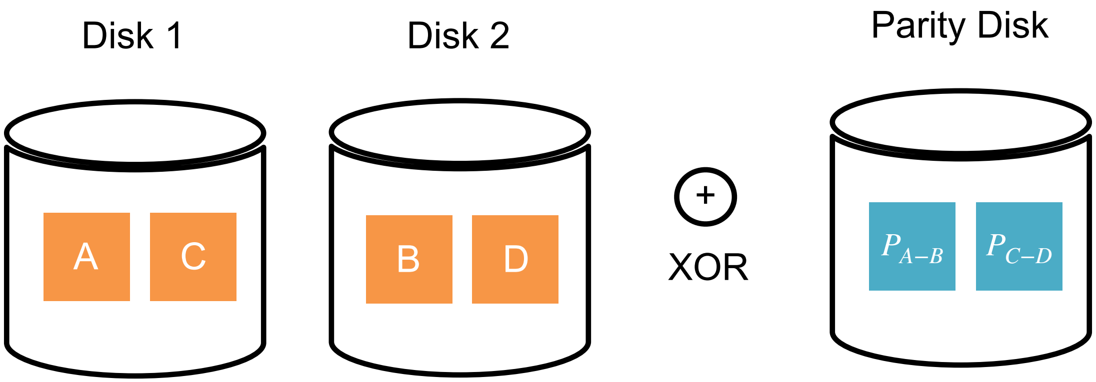
Example: RAID 0+1 (Mirroring)



✓ Increases read/write throughput + fault tolerance ✗ Decreases net space

Property/Level	0	1	0+1		
Read Throughput	2x	2x	4x		
Write Throughput	2x	1x	2x		
Fault Tolerance	✗	✓	✓		
Net Space	1x	1/2x	1/2x		

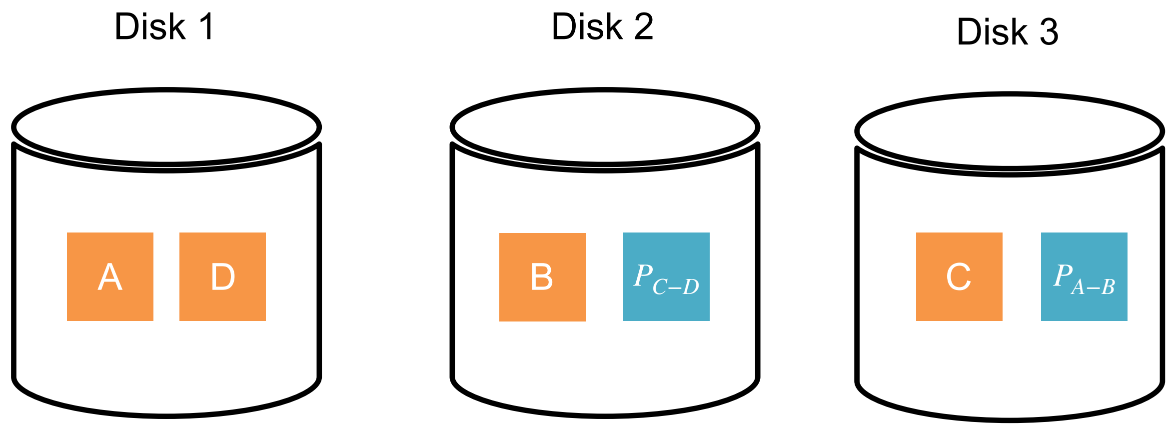
Example: RAID 4 (Block Striping + Parity)



✓ Increases fault tolerance ✗ Decreases net space

Property/Level	0	1	0+1	4	
Read Throughput	2x	2x	4x	2x	
Write Throughput	2x	1x	2x	0-1x	
Fault Tolerance	✗	✓	✓	✓	
Net Space	1x	1/2x	1/2x	2/3	

Example: RAID 5 (Block Striping + Parity)



✓ Increases read/write throughput + fault tolerance ✗ Decreases net space

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

Task 4: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: Which RAID level do you choose for a large intermediate result needed to answer a query?

(A) Level 0

(B) Level 1

(C) Level 0+1

(D) Level 4

(D) Level 5

Task 4: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: Which RAID level do you choose for a large intermediate result needed to answer a query?

(A) Level 0

(B) Level 1

(C) Level 0+1

(D) Level 4

(D) Level 5

Fast read/write needed,
fault tolerance can be ignored

Task 5: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: Which RAID level do you choose as a default option to store your database tables?

(A) Level 0

(B) Level 1

(C) Level 0+1

(D) Level 4

(E) Level 5

Task 5: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: Which RAID level do you choose as a default option to store your database tables?

(A) Level 0

(B) Level 1

(C) Level 0+1

(D) Level 4

(E) Level 5

Many benefits: fast read/write, fault tolerance, much net space

Task 6: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: What is the major tradeoff you have to consider storing data on multiple disks?

(A) Read vs. write throughput

(B) Fault tolerance vs. net space

(C) Throughput vs. net space

(D) Write throughput vs. fault tolerance

Task 6: RAID

Property/Level	0	1	0+1	4	5
Read Throughput	2x	2x	4x	2x	2x
Write Throughput	2x	1x	2x	0-1x	1-2x
Fault Tolerance	✗	✓	✓	✓	✓
Net Space	1x	1/2x	1/2x	2/3	2/3

- **Question**: What is the major tradeoff you have to consider storing data on multiple disks?

(A) Read vs. write throughput

(B) Fault tolerance vs. net space

(C) Throughput vs. net space

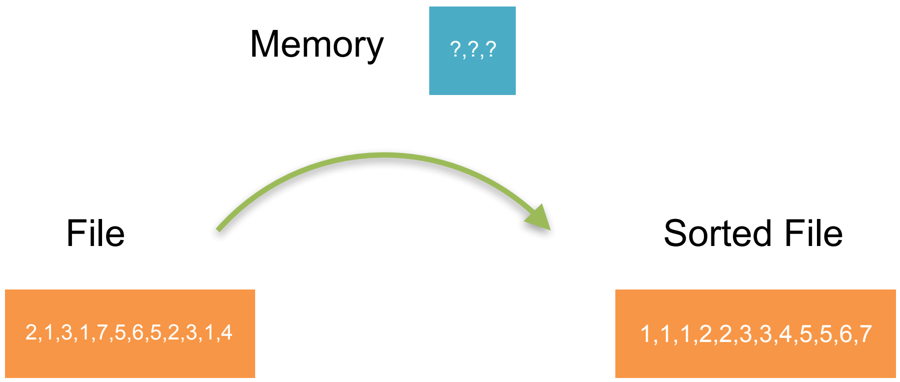
(D) Write throughput vs. fault tolerance

Fault tolerance comes at the cost of less net space

Table of Contents

- Organisation
- Exercise Sheet 1
- Magnetic Disks
- RAID
- **External Sorting**

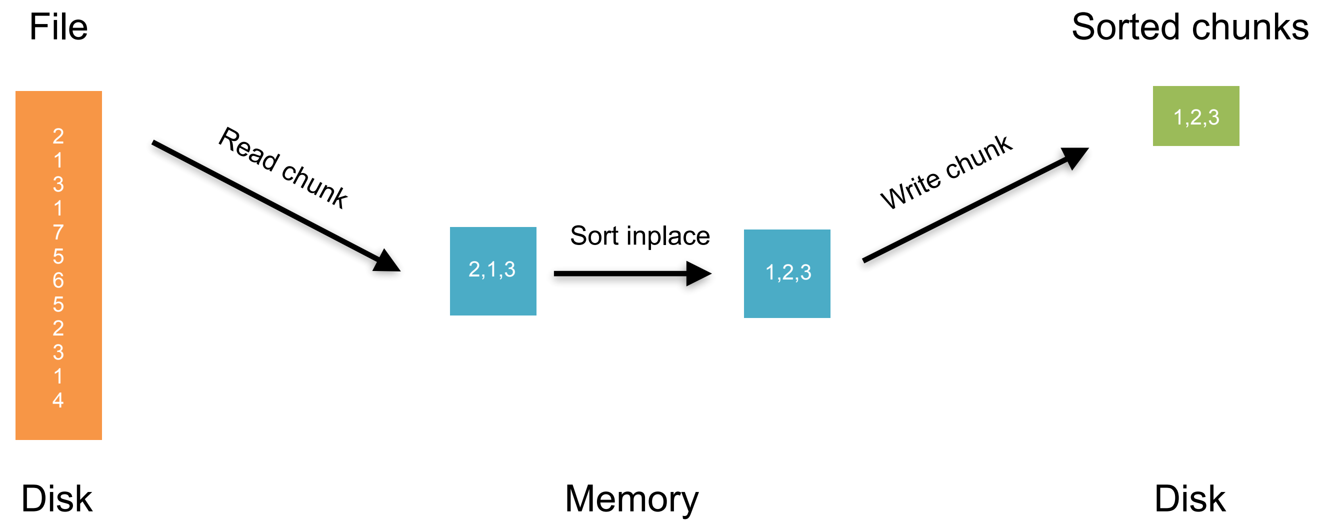
Recap: External Sorting



Question: How do you sort a large file with limited memory?

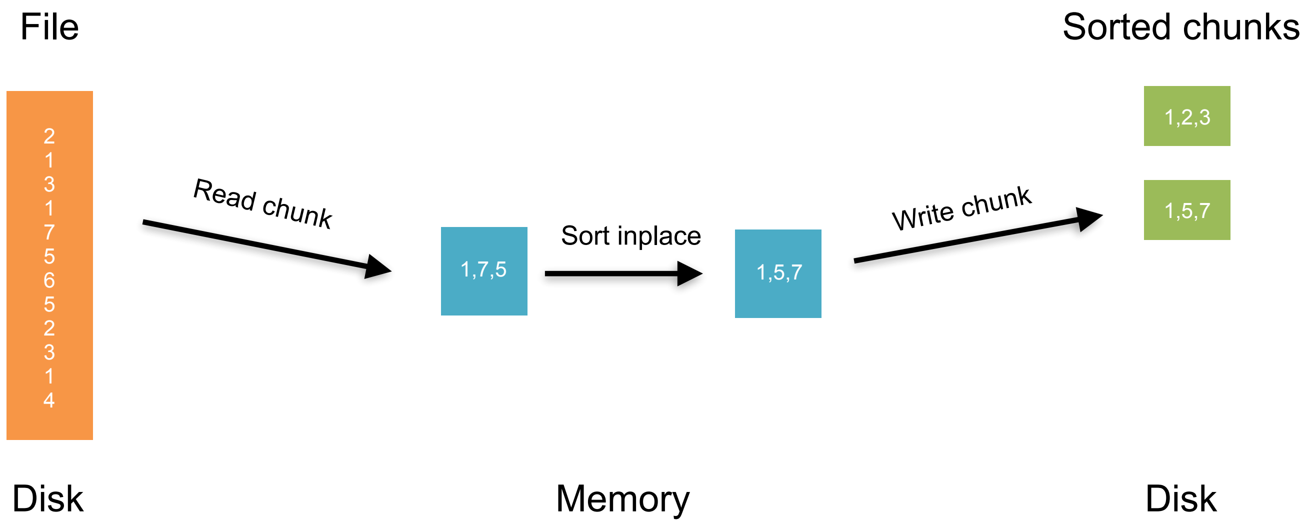
- Required when data does not fit into main memory
- Approaches based on merge or quicksort
- Many optimisations: concurrent reads, asynchronous IO, …

Example: External Merge Sort



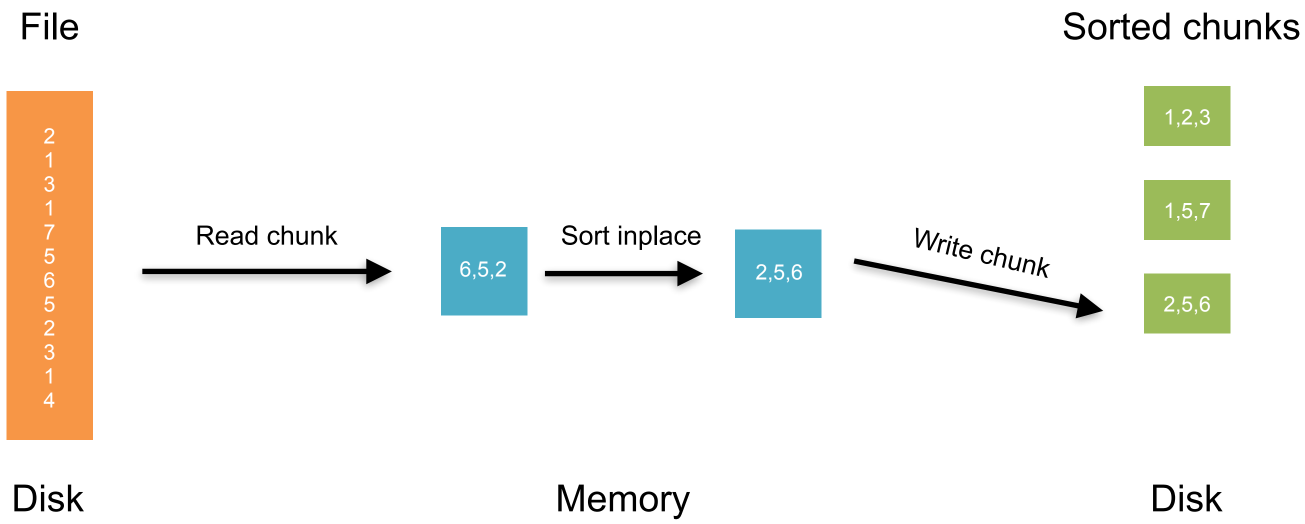
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 1**: Read File in 300 MB chunks, sort inplace, save 4 sorted files on disk

Example: External Merge Sort



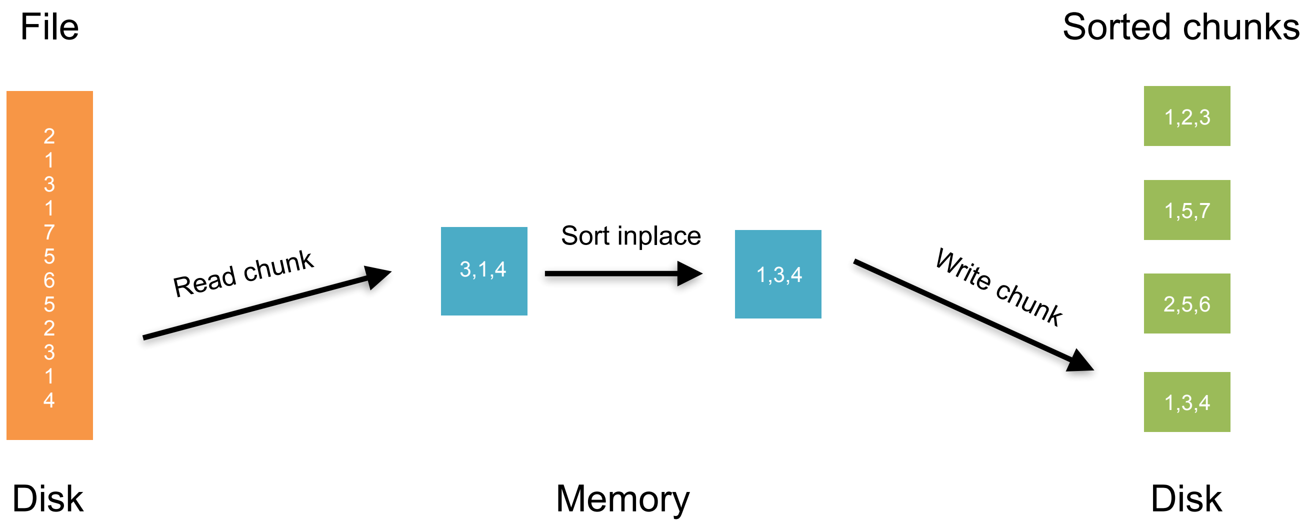
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 1**: Read File in 300 MB chunks, sort inplace, save 4 sorted files on disk

Example: External Merge Sort



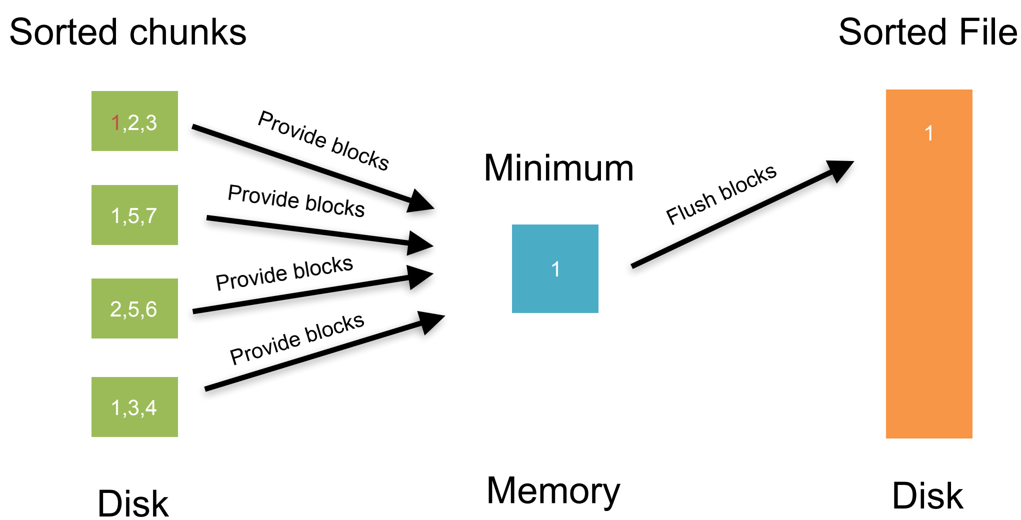
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 1**: Read File in 300 MB chunks, sort inplace, save 4 sorted files on disk

Example: External Merge Sort



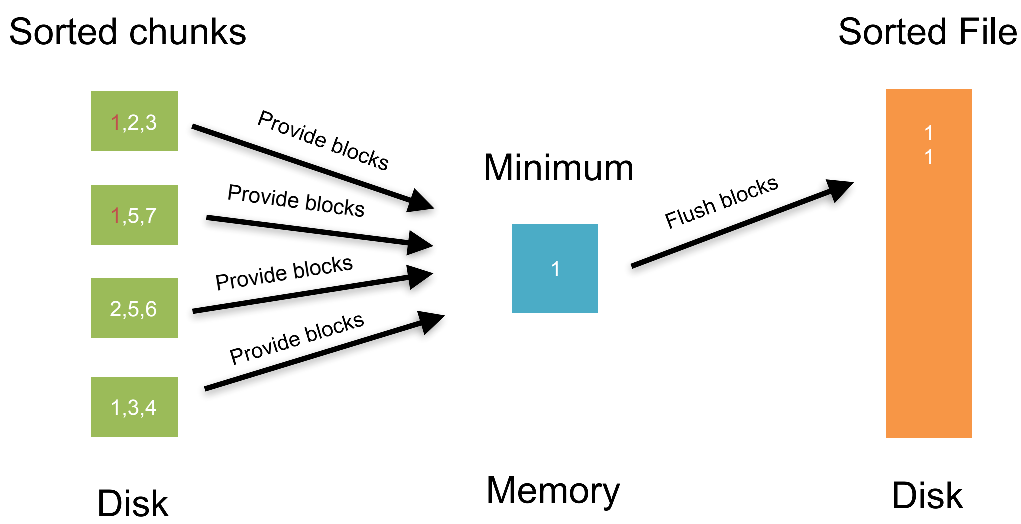
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 1**: Read File in 300 MB chunks, sort inplace, save 4 sorted files on disk

Example: External Merge Sort



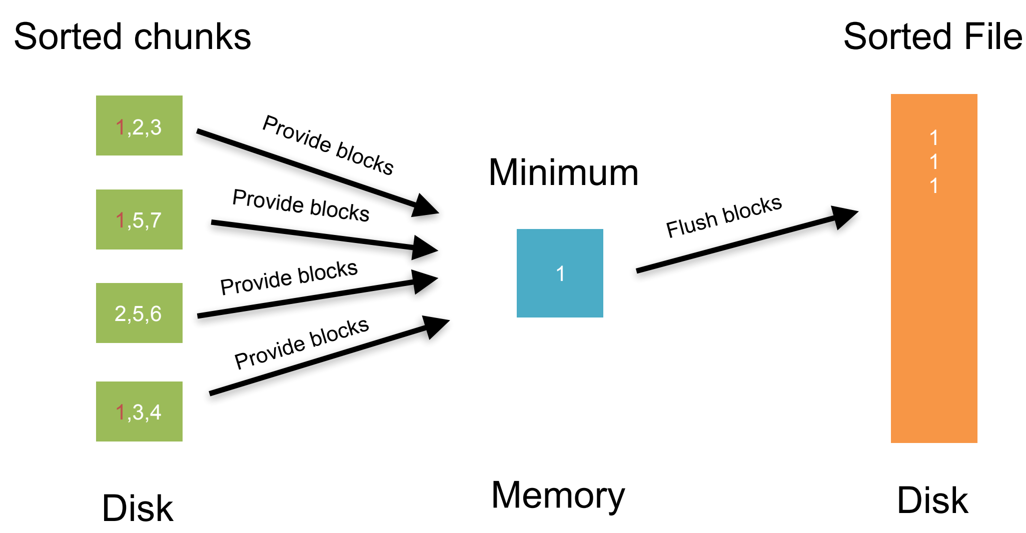
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 2**: Read 60 MB blocks from 4 chunks, merge and write to disk
    - K-way-merge: Repeatedly select minimal elements from loaded blocks and save in output buffer (also 60 MB), flush to disk when full
    - Read new 60 MB blocks from chunks at request, when needed

Example: External Merge Sort



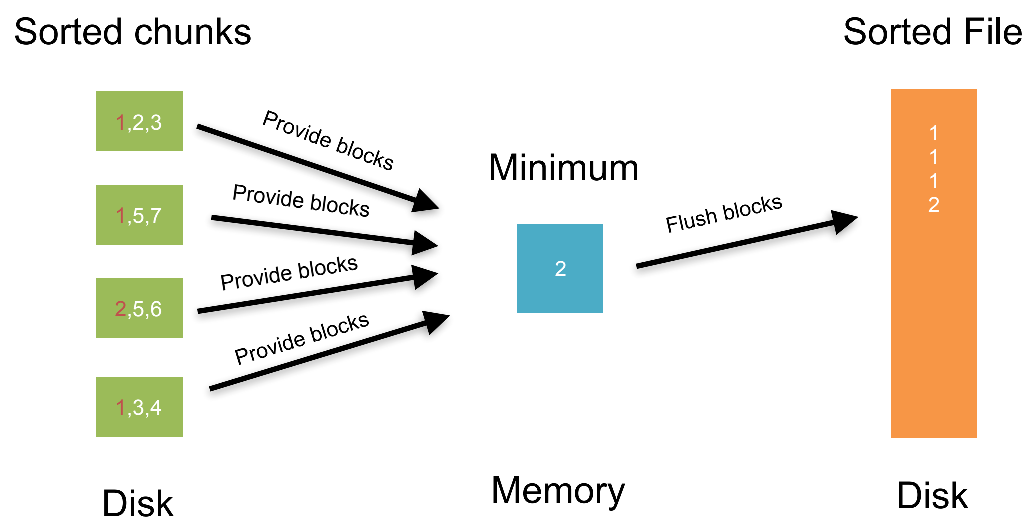
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 2**: Read 60 MB blocks from 4 chunks, merge and write to disk
  - K-way-merge: Repeatedly select minimal elements from loaded blocks and save in output buffer (also 60 MB), flush to disk when full
  - Read new 60 MB blocks from chunks at request, when needed

Example: External Merge Sort



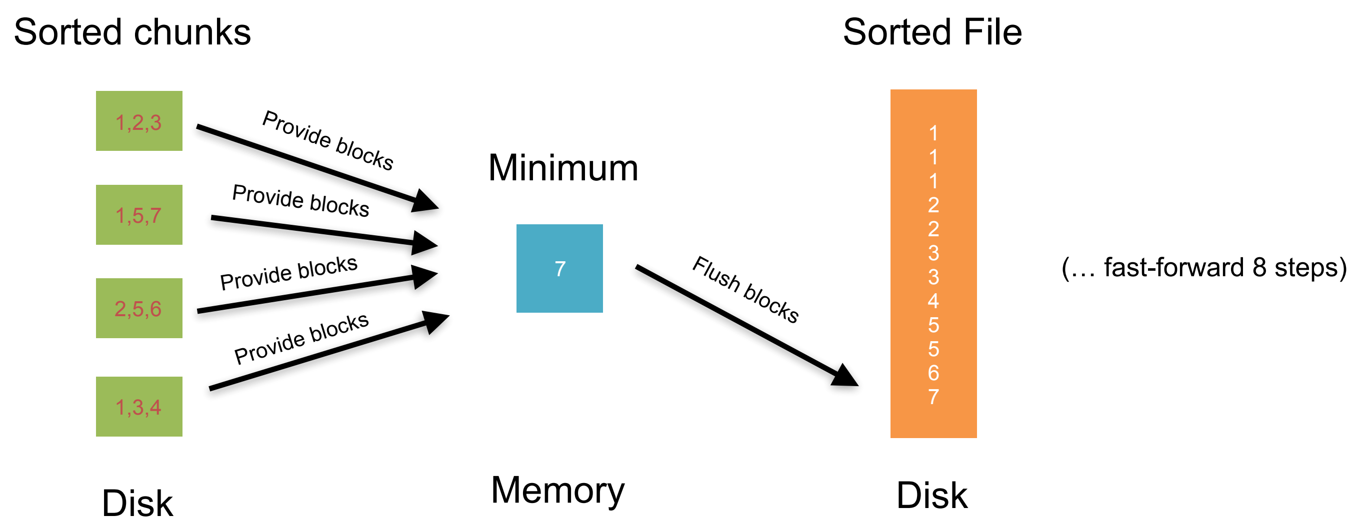
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 2**: Read 60 MB blocks from 4 chunks, merge and write to disk
  - K-way-merge: Repeatedly select minimal elements from loaded blocks and save in output buffer (also 60 MB), flush to disk when full
  - Read new 60 MB blocks from chunks at request, when needed

Example: External Merge Sort



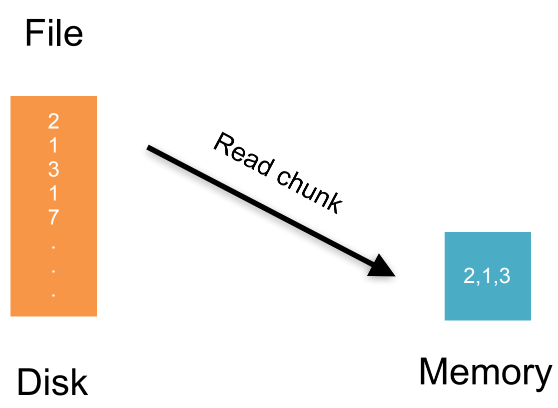
- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 2**: Read 60 MB blocks from 4 chunks, merge and write to disk
  - K-way-merge: Repeatedly select minimal elements from loaded blocks and save in output buffer (also 60 MB), flush to disk when full
  - Read new 60 MB blocks from chunks at request, when needed

Example: External Merge Sort



- Example: File contains 1.2 GB integers, we can store 300 MB in memory
- **Step 2**: Read 60 MB blocks from 4 chunks, merge and write to disk
  - K-way-merge: Repeatedly select minimal elements from loaded blocks and save in output buffer (also 60 MB), flush to disk when full
  - Read new 60 MB blocks from chunks at request, when empty

Task 7: External Merge Sort



File Size: 1.8 GB
Number of Chunks: 3

- **Question**: How many IO accesses do you need to create the sorted chunks in this example?

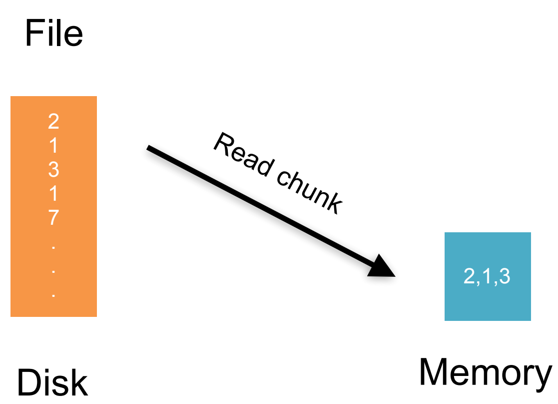
(A) 3

(B) 4

(C) 6

(D) 12

Task 7: External Merge Sort



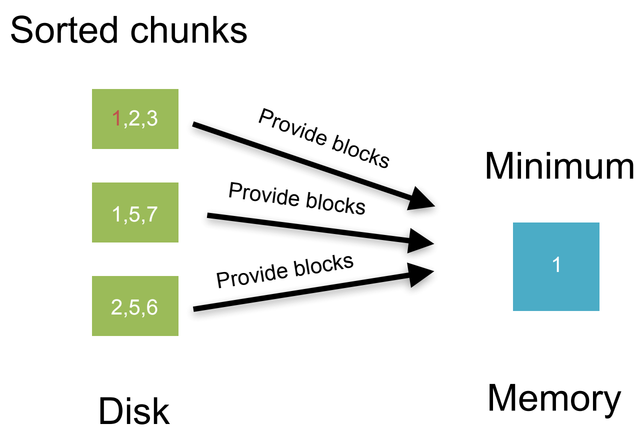
File Size: 1.8 GB
Number of Chunks: 3

- **Question**: How many IO accesses do you need to create the sorted chunks in this example?

(A) 3	(B) 4	(C) 6	(D) 12

We have to perform
3 reads and 3 writes

Task 8: External Merge Sort

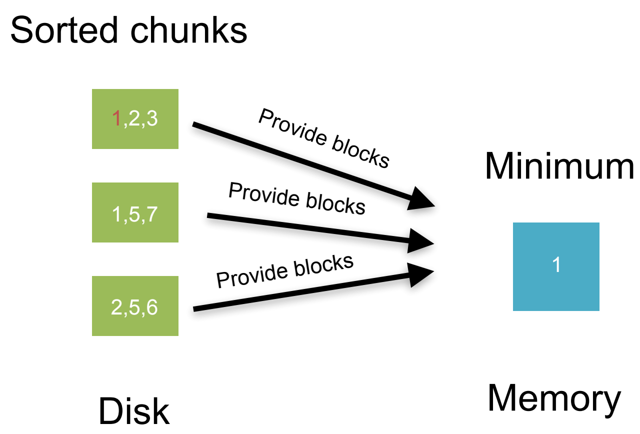


Memory Size: 120 MB
Block Size: 30 MB
Number of Chunks: 3

- **Question**: How large can the output buffer be in this example?

(A) 30 MB	(B) 60 MB	(C) 90 MB	(D) 120 MB

Task 8: External Merge Sort



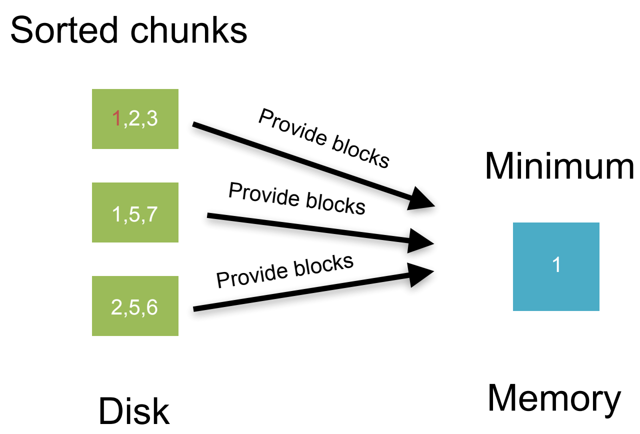
Memory Size: 120 MB
Block Size: 30 MB
Number of Chunks: 3

- **Question**: How large can the output buffer be in this example?

(A) 30 MB	(B) 60 MB	(C) 90 MB	(D) 120 MB

Each chunk allocates a block of 30 MB,
so 30 MB remains for the output buffer

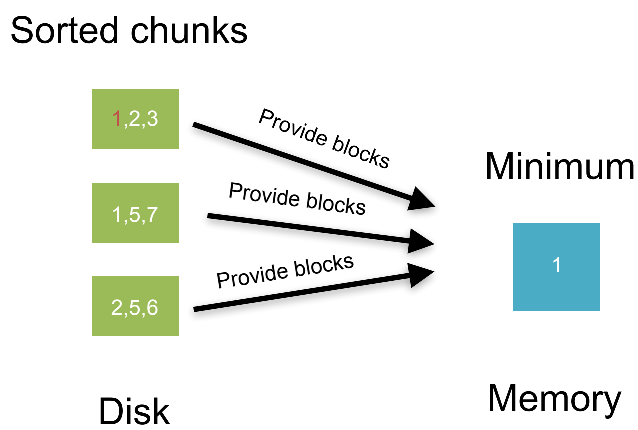
Task 9: External Merge Sort



Memory Size: 120 MB
Block Size: 30 MB
Number of Chunks: 3

- **Question**: How can we extract the minima repeatedly from the provided memory blocks?

Task 9: External Merge Sort



Memory Size: 120 MB
Block Size: 30 MB
Number of Chunks: 3

- **Question**: How can we extract the minima repeatedly from the provided memory blocks?

Linear Search

Min-Heap