PART II

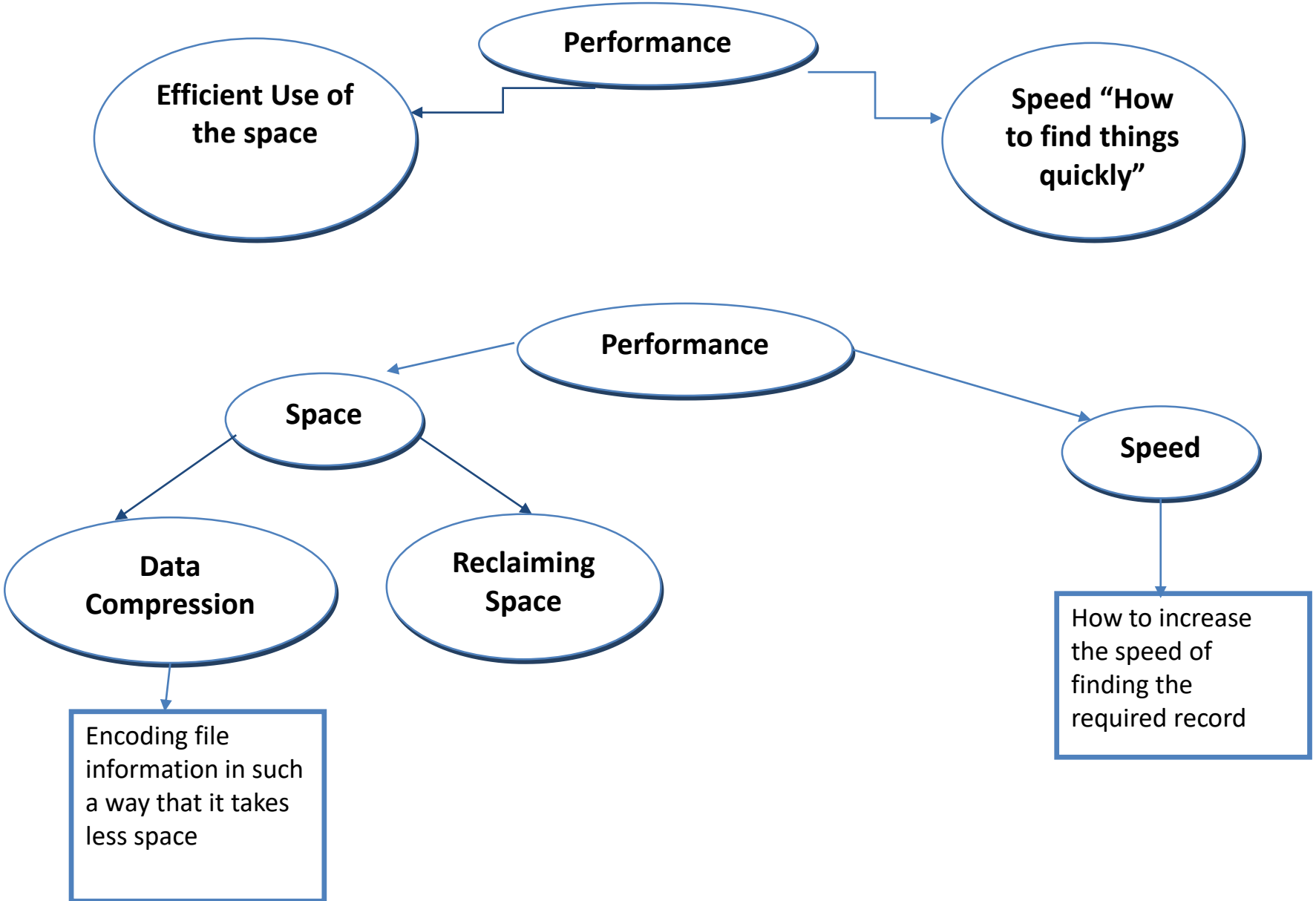
File Organization

Organizing File for Performance

2020

Prof. Mohamed Hashem

Performance



Contents

Data compression

Reclaiming space in files

Finding things quickly: *An Introduction to internal sorting*

binary searching

Keysorting

- We will be looking at four different issues:
  - Data Compression: how to make files smaller
  - Reclaiming space in files that have undergone deletions and updates
  - Sorting Files in order to support binary searching ==> Internal Sorting
  - A better Sorting Method: KeySorting

Data Compression

- Reasons for data compression
  - less storage
  - transmitting faster, decreasing access time
  - processing faster sequentially
- **Techniques**:

  1- Using compact notation

  2-Using Run Length indicator

  3- Using Variable length coding

  - Huffman Code

Data Compression:

1 - Using a different notation

- Fixed-Length fields are good candidates
- Decrease the # of bits by finding a more compact notation

  Example:

  1- original state field notation is 16bits, but we can encode with 6bit notation because of the # of all states are 50
- *2- same for Egypt Governorance 30 we can use 5 bits*
- *3- for AS FCIS 9 department we can use 4 bits*
- *Disadvantages:*
- . unreadable by human
  - cost in encoding time
  - decoding modules => increase the complexity of s/w

  => used for particular application

Data Compression

2-Run-length encoding algorithm

- read through pixels, copying pixel values to file in sequence, except the same pixel value occurs more than once in succession
- when the same value occurs more than once in succession, substitute the following three bytes
  - special run-length code indicator((ex) ff)
  - pixel value repeated
  - the number of times that value is repeated
    - ex) 22 23 *24 24 24 24 24 24 24* 25 **26 26 26 26 26 26** 25 24
      - 22 23 ff24 07 25 **ff 26 06** 25 24

Disadvantages

not guarantee any particular amount of space savings

under some circumstances, *compressed image* is larger than original image

Why? Can you prevent this?

Data Compression

3-variable-length codes

- *Morse code*: oldest & most common scheme of variable-length code
- Some values occur more frequently than others
  - that value should take the least amount of space
- *Huffman coding*
  - base on probability of occurrence
    - determine probabilities of each value occurring
    - build binary tree with search path for each value
    - more frequently occurring values are given shorter search paths in tree

Data Compression

Huffman coding

Problem

you give a file

File :

I		A	M		S	A	M	M	y

Find Huffman tree ?

- Size of tree=10*8=80 byte

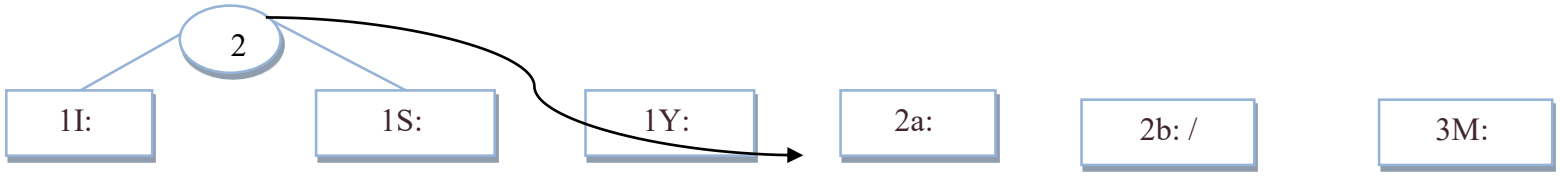
Put According to frequency (ascending), then according to character (alphabetic).

Character	A	I	M	S	Y	/b
Frequency	2	1	3	1	1	2
Code	00	1010	11	1011	100	01
# bits	2*2	1*4	3*2	1*4	1*3	2*2

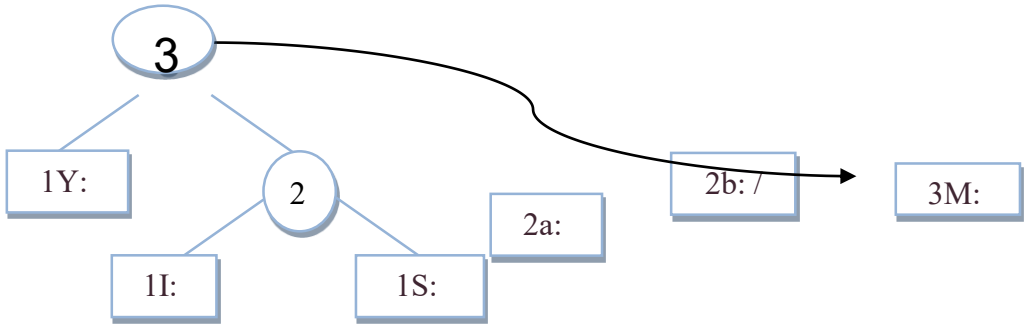
Data Compression

Huffman coding

Put According to frequency (ascending), then according to character alphabetic).



Add the two smallest values, and put it at the beginning of sequence of numbers•

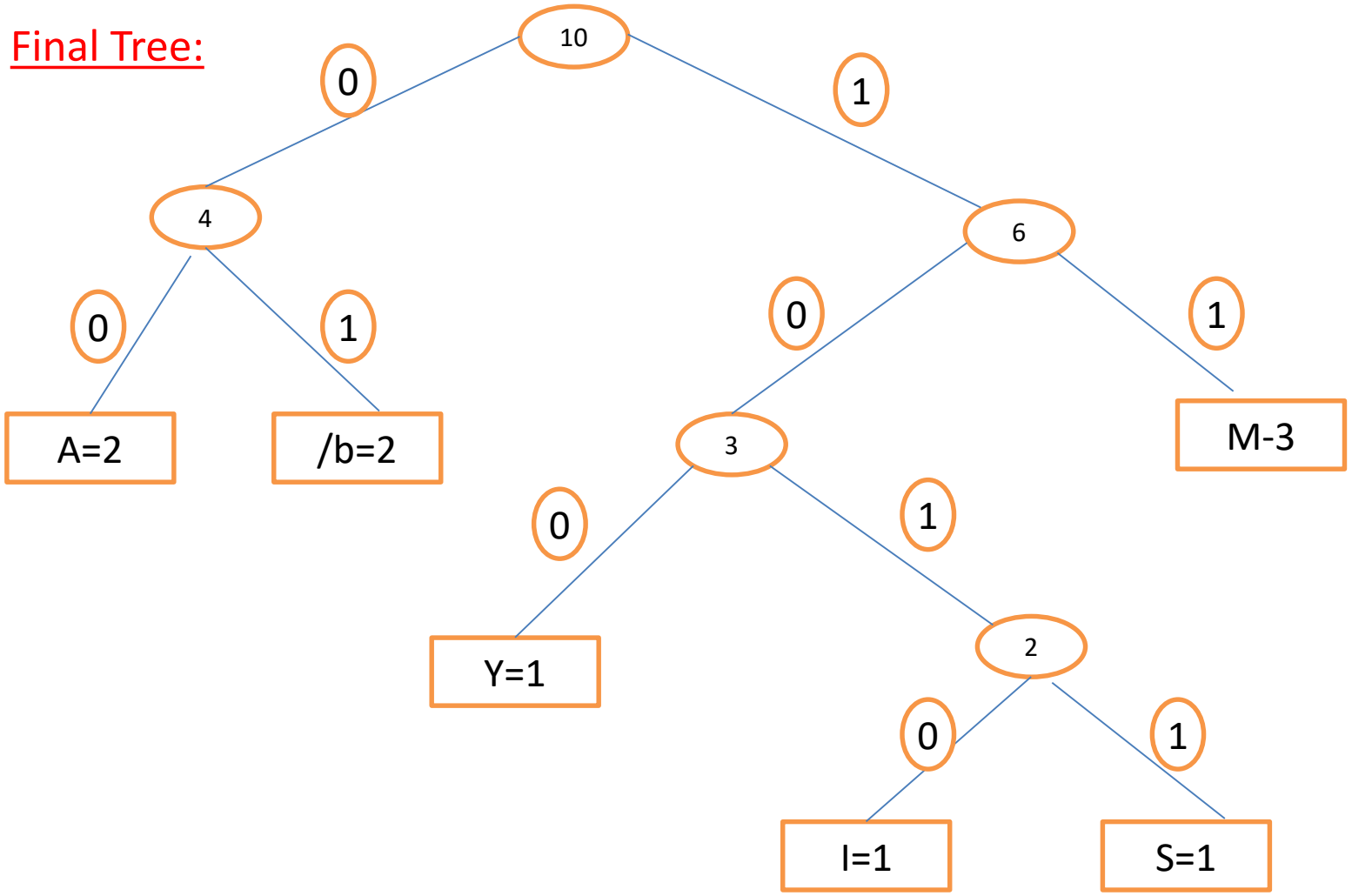


And continue up to the end tree as shown•

For each node assign 1 to right edge & 0 for left edge•

Huffman coding

Final Tree:



Given File (2) - - > 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0•

The original file was - - > M S A I M A•

for the string SYMA ⟶ 1011 100 11 00

Reclaiming Space

Basic File Operations:

- Open.
- Close
- Read
- Write
- Append.

There is NO delete Operation

Record deletion Operation

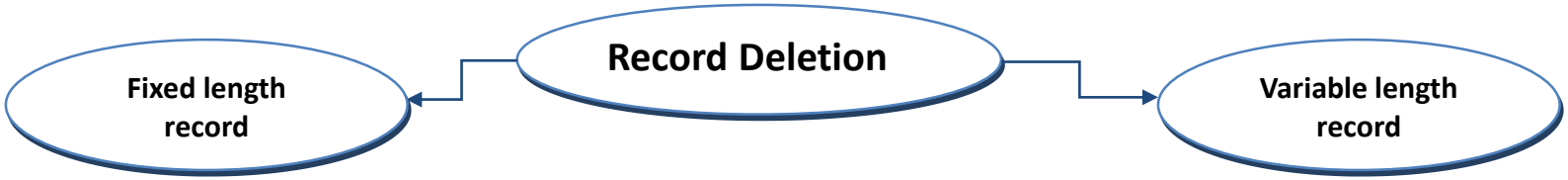
1. Use special character at the beginning of the deleted record.
2. Make a small software that when reading consider the record (which have symbol at its beginning) as deleted.
3. Put the deleted record in AVAIL List to keep track of deleted records

AVAIL List: Available List

List of all deleted record addresses and is implemented as a stack

Reclaiming Space

Record Deletion:



- Deletion is the same for both Fixed and Variable Length Records.
- Addition differs from fixed Length Records to Variable Length Records.
- RRN (Relative Record Number) - - > Order of the Record.
- Offsite byte is the address of first byte of the record

If we want 5 = (RRN-1) * Record length
= 4 * 100
= 400 Bytes

Reclaiming Space

Fixed-Length Record Deletion & Addition:

1-Delition :

Given the following file:

We need to delete
REC # 3,5,6 in order

1-Before deleting - - >AVAIL List =-1

AVAIL List =-1

1	2	3	4	5	6	7

2-After Deleting Rec 3

AVAIL List =3

1	2	*1	4	9	8	7

3-After Deleting Rec 5

AVAIL List =5

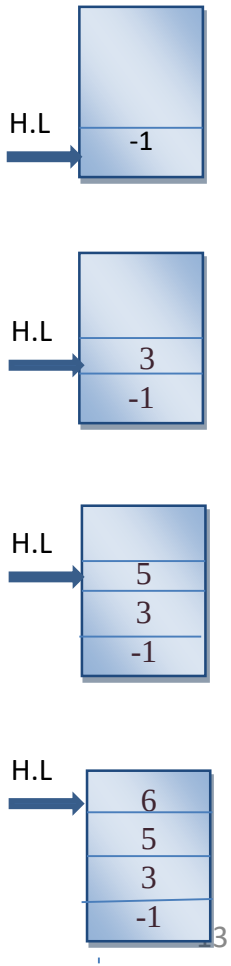
1	2	*1	4	*3	8	7

4-After Deleting Rec 6

AVAIL List =6

1	2	*-1	4	*3	*5	7

AVAIL List



Reclaiming Space

Fixed-Length Record Deletion & Addition:

Addition :

We need to add REC # 8,9,10,11,12 in order

1-Initially the file :

AVAIL List = 6

1	2	*-1	4	*3	*5	7

2-After Adding Rec 8

AVAIL List =5

1	2	*1	4	*3	8	7

3-After Adding Rec 9

AVAIL List =3

1	2	*1	4	9	8	7

4-After Deleting Rec 10

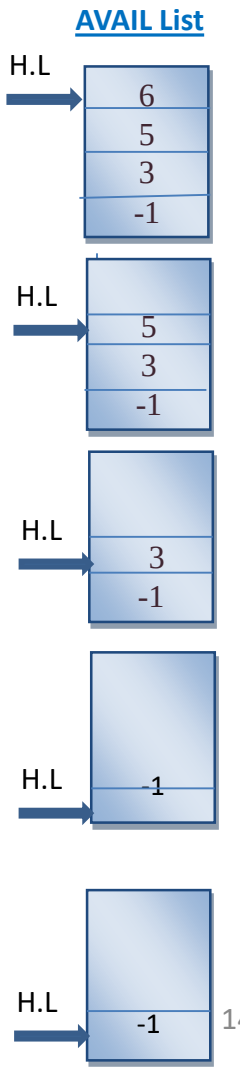
AVAIL List =-1

1	2	10	4	9	8	7

4-After Deleting Rec 11 &12

AVAIL List =-1

1	2	10	4	9	8	7	11	12



Reclaiming Space

Variable-Length Record Deletion & Addition:

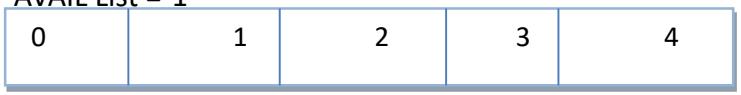
1-Delition :

Given the following file: We need to delete REC # 2,1,4 in order

AVAIL List

1-Before deleting - - >AVAIL List =-1

AVAIL List =-1

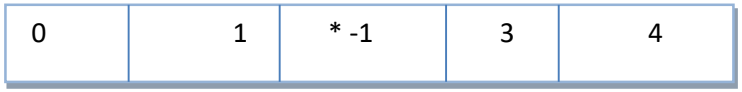


H.L

Offset	size	pointer

2-After Deleting Rec 2

AVAIL List =16

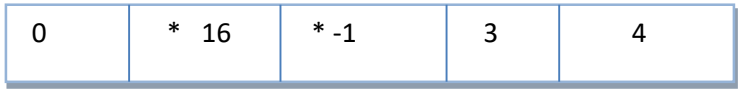


H.L

Offset1	Size	pointer
16	10	-1

3-After Deleting Rec 1

AVAIL List =11

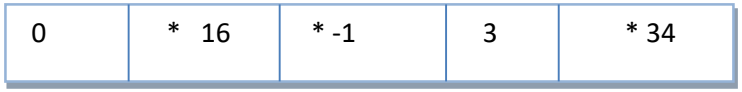


H.L

Off set	Size	pointer
11	5	16
16	10	-1

4-After Deleting Rec 4

AVAIL List =34



H.L

Offset	size	pointer
34	30	11
11	5	16
16	10	-1

Reclaiming Space

Variable-Length Record Deletion & Addition:

2-Addition:

In adding records, search through avail list for right size & insert record according the right selected fitting strategy

Placement Strategies:

- First-fit
  - select the first available record slot that can accommodate the new record.
  - suitable when lost space is due to internal fragmentation
- Best-fit
  - Select the first available smallest available record slot that can accommodate the new record
  - avail list in ascending order
  - suitable when lost space is due to internal fragmentation
- Worst-fit
  - select the largest available record slot
  - avail list in descending order
  - suitable when lost space is due to external fragmentation

Reclaiming Space

Variable-Length Record Deletion & Addition:

AVAIL List

Offset	size	pointer
34	30	11
11	5	16
16	10	-1

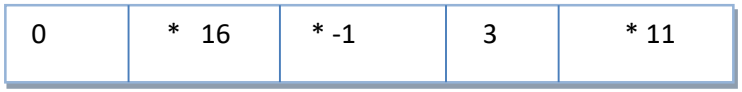
H.L →

2-Addition: **We want to add following records in order:**

with First Fit: R_5 of size 40 bytes, R_6 10 Bs, and R_7 of 7 Bs

1-Initially The file

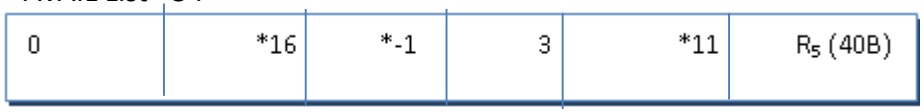
AVAIL List =34



10B 5B 10B 8B 30B

2-After Adding Rec 5

AVAIL List =34



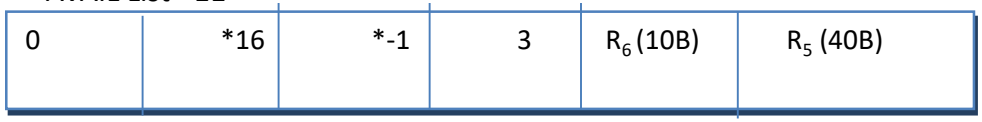
10B 5B 10B 8B 30B 40B

Off set	Size	pointer
11	5	16
16	10	-1

H.L →

3-After Deleting Rec 6

AVAIL List =11



10B 5B 10B

Offset1	Size	pointer
11	5	-1

H.L →

4-After Deleting Rec 7

AVAIL List =11

Offset1	Size	pointer
11	5	-1

H.L →

5- Final File

AVAIL List =11

Reclaiming Space

Exercise:

Apply Best-fit & worst -fit strategies to the problem above

File Fragmentation:

Due to the dynamic nature of file(deleting and adding) leads to logical fragmentation.

* Internal fragmentation
* External fragmentation

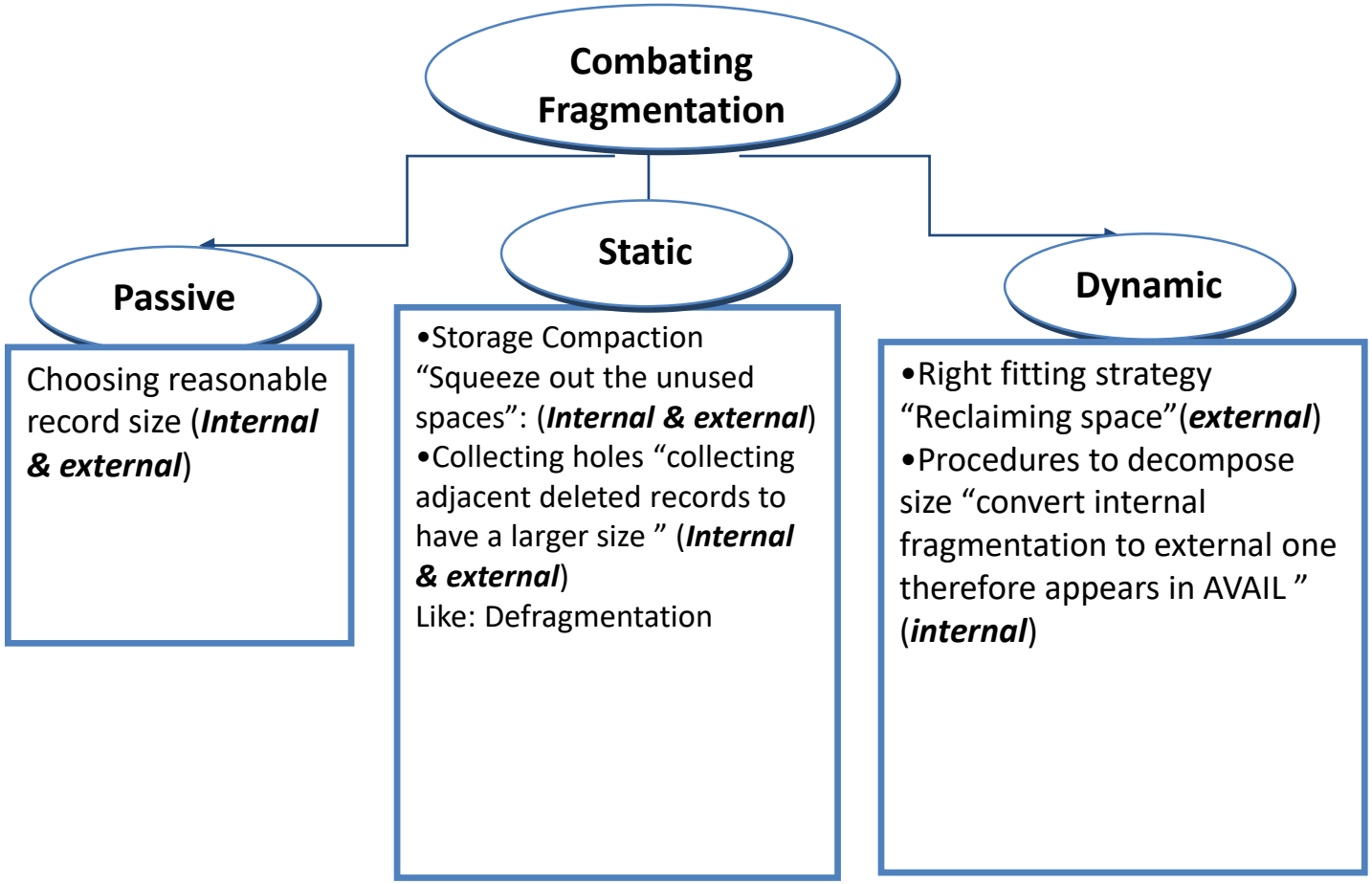
	Internal	External
Where	- within a record	- Between records
Av. List	doesn't appear in AVAIL list	appears in AVAIL
occurs when	- adding small record size in a larger size deleted record	- deleting any record

Physical fragmentation: Sector does not equal integer number of records

Cluster does not equal integer number of files

Reclaiming Space

How to Combat File Fragmentation:



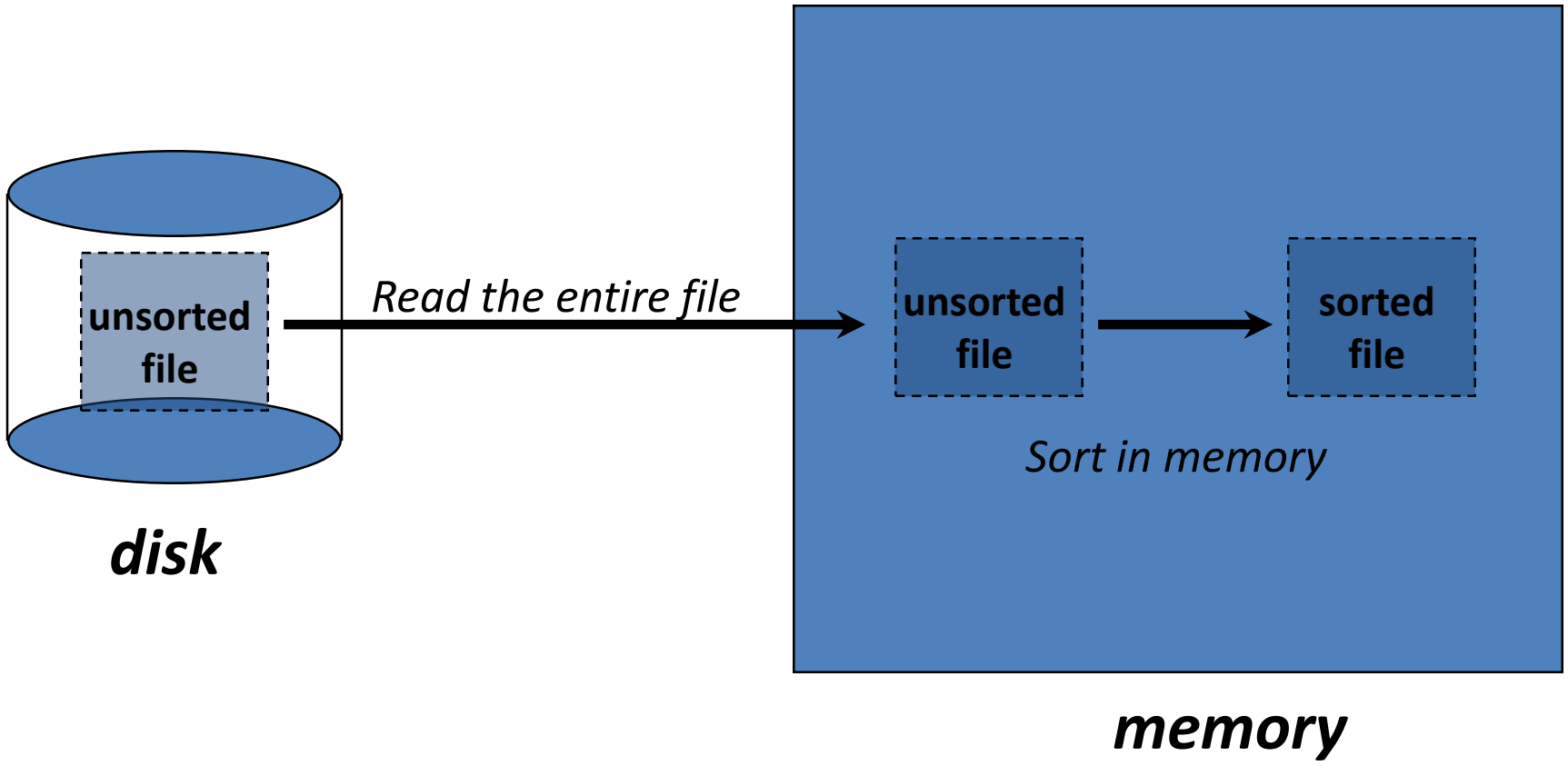
Finding Things Quickly

- The cost of Seeking is very high.
- This cost has to be taken into consideration when determining a strategy for searching a file for a particular piece of information.
- The same question also arises with respect to sorting, which often is the first step to searching efficiently.
- Rather than simply trying to sort and search, we concentrate on doing so in a way that minimizes the number of seeks.

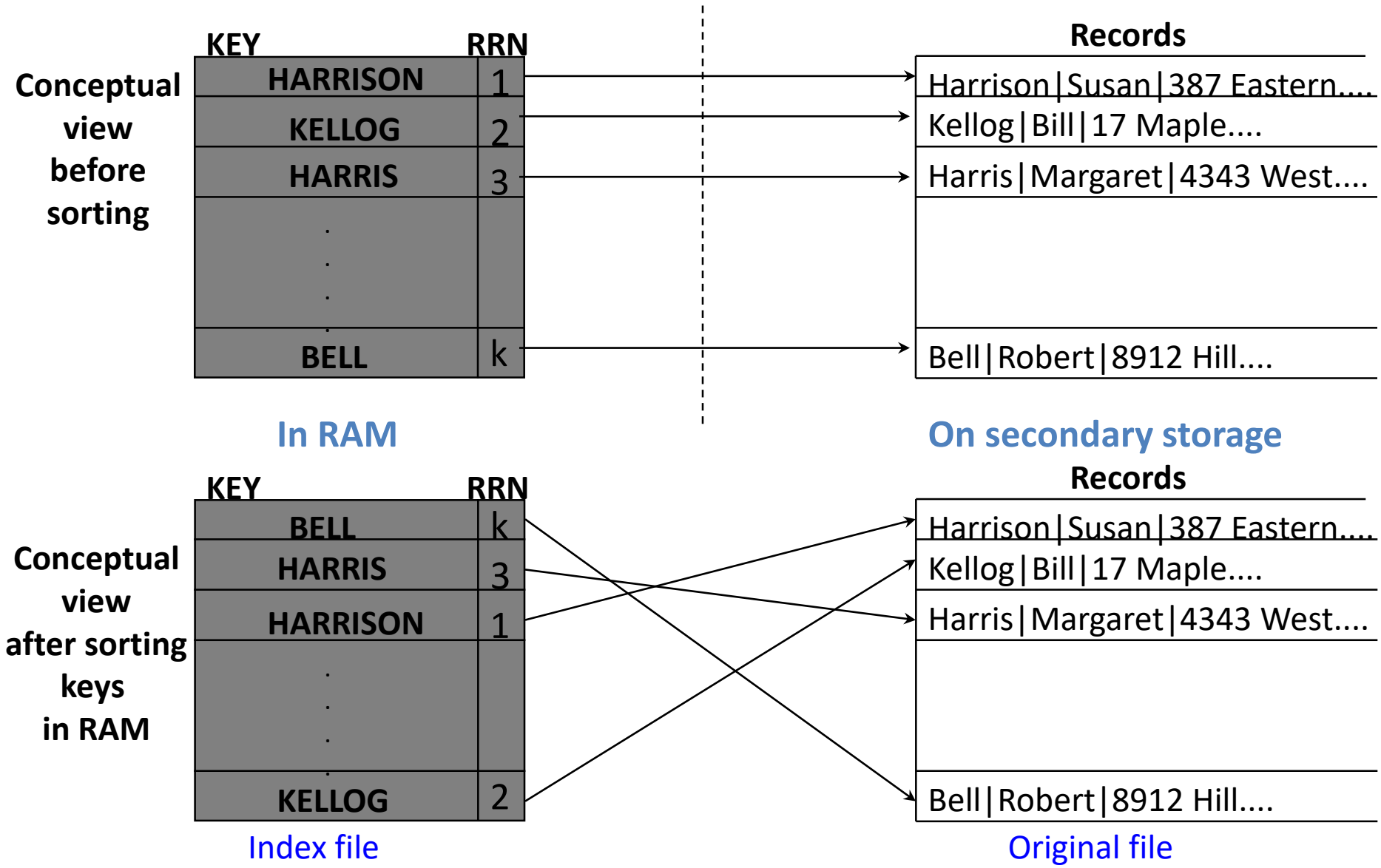
- Binary search vs. Sequential search
  - binary search
    - O(log n)
    - list is **sorted** by key
  - sequential search
    - O(n)

- Limitations of binary search & internal sort
  - binary search requires more than one or two access
    c.f.) single access by RRN
  - keeping a file sorted is very expensive
  - an internal sort works only on small files

Internal Sort



Key Sorting


KEY
RRN
Records
Conceptual view before sorting
HARRISON
1
KELLOG
2
HARRIS
3
BELL
k
Harrison|Susan|387 Eastern....
Kellog|Bill|17 Maple....
Harris|Margaret|4343 West....
Bell|Robert|8912 Hill....
In RAM
On secondary storage
KEY
RRN
Records
Conceptual view after sorting keys in RAM
BELL
k
HARRIS
3
HARRISON
1
KELLOG
2
Harrison|Susan|387 Eastern....
Kellog|Bill|17 Maple....
Harris|Margaret|4343 West....
Bell|Robert|8912 Hill....
Index file
Original file

Finding Things Quickly

- Why we rewrite the file again to the secondary storage
- If this the case and not rewrite the index it is the IDEXING