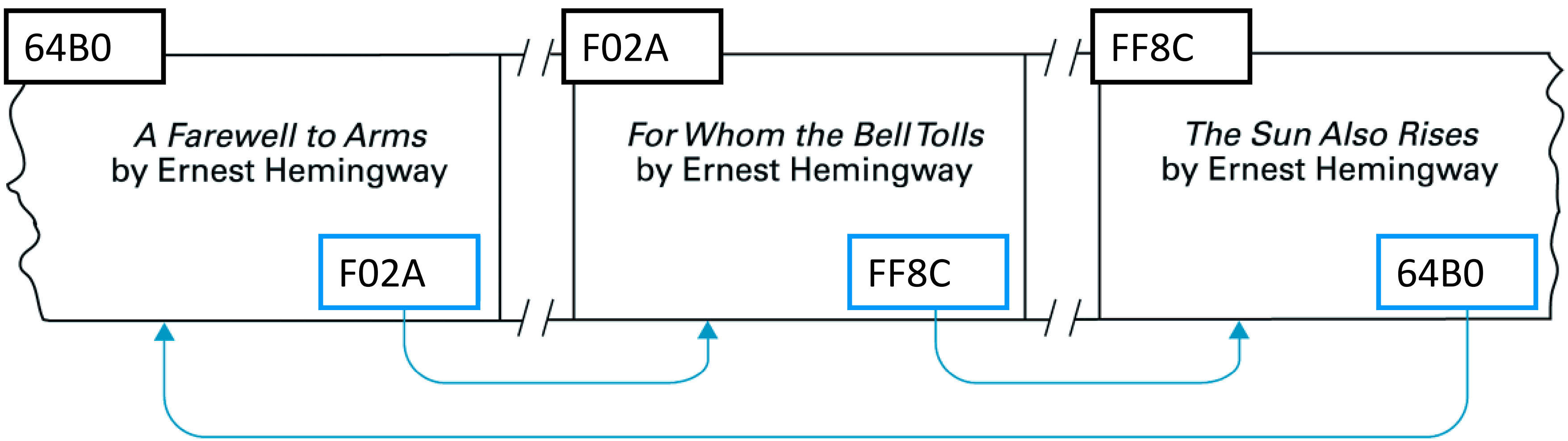
CHAPTER 7

Data Structures

- Abstractions of the actual data organization in main memory
- Allow users to perceive data as ‘logical units’ (e.g.: arrangement in rows and columns)

7.1: Data Structure Basics: Pointers

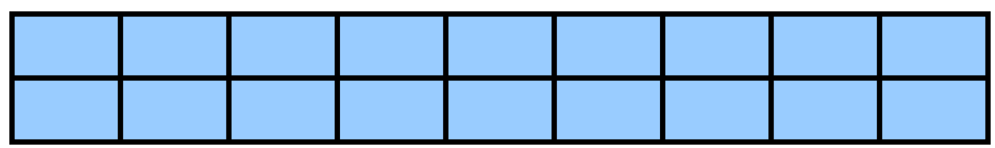
- Pointers:
  - pointer = location in memory that contains the address of another location in memory
  - so: pointer *points* to data positioned elsewhere in memory



7.1: Static versus Dynamic Data Structures

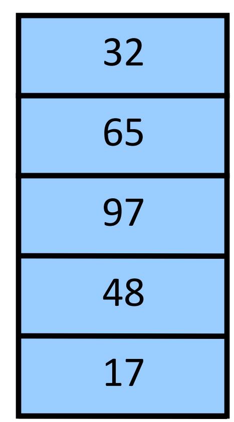
- Static:
  - shape & size of structure does not change over time
    - example in C: `int Table[2][9];`

Table:



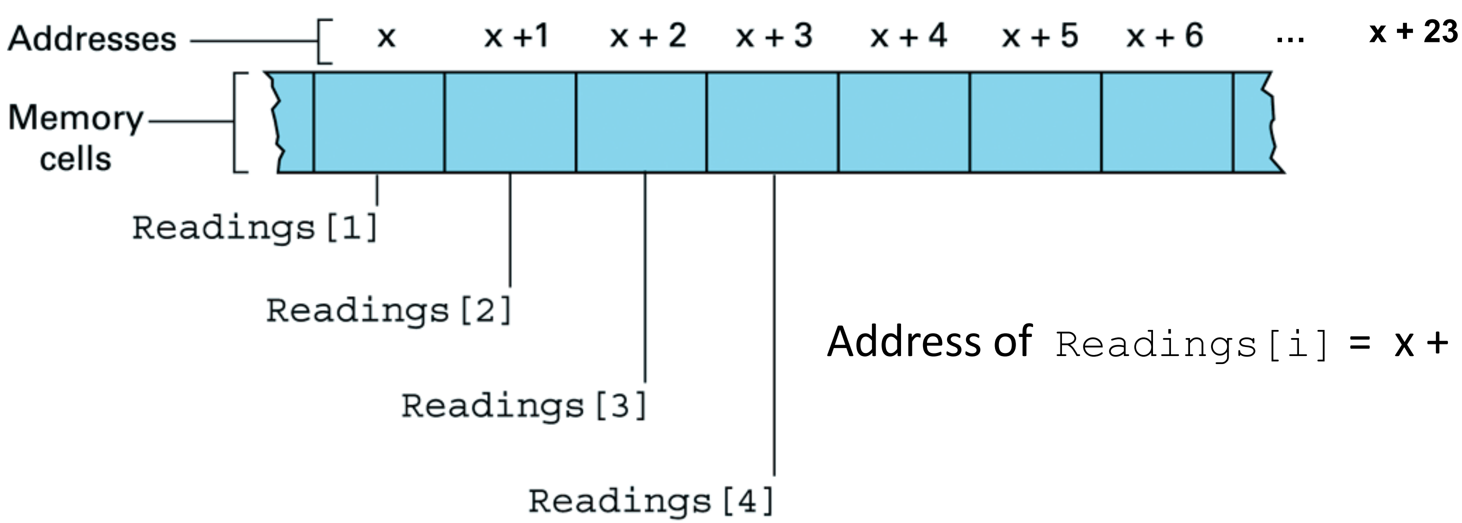
- Dynamic:
  - shape & size may change
  - example: Stack

Stack:



7.2: Arrays

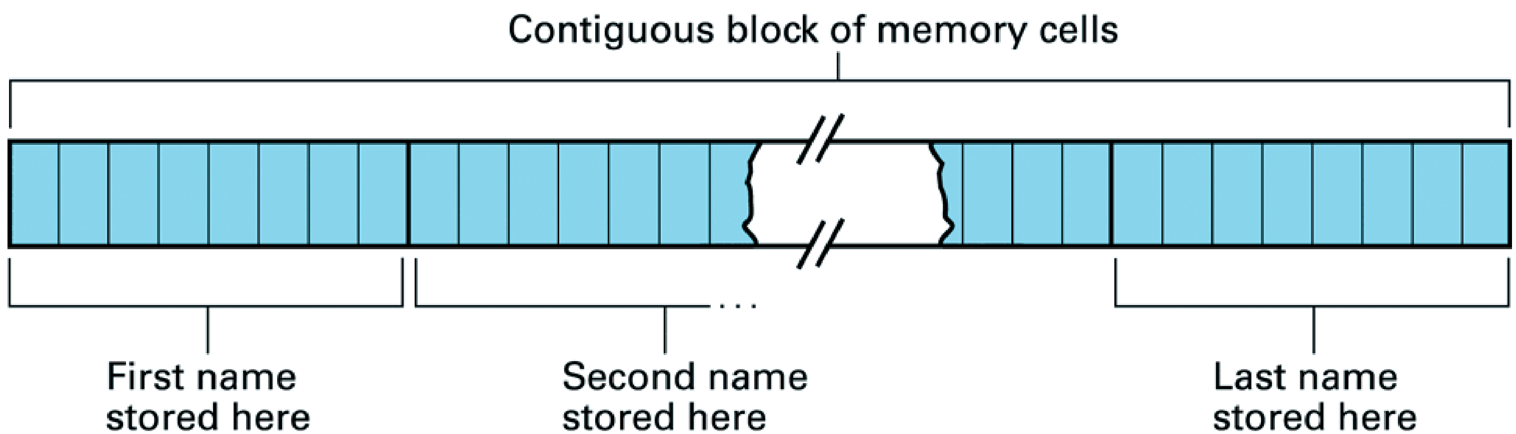
- Example: to store 24 hourly temperature readings…
- … a convenient storage structure is *1-D homogeneous array* of 24 elements (e.g. in C: `float Readings[24]`)
- In main memory:



Address of `Readings[i]` = x + (i-1)

7.3: Lists

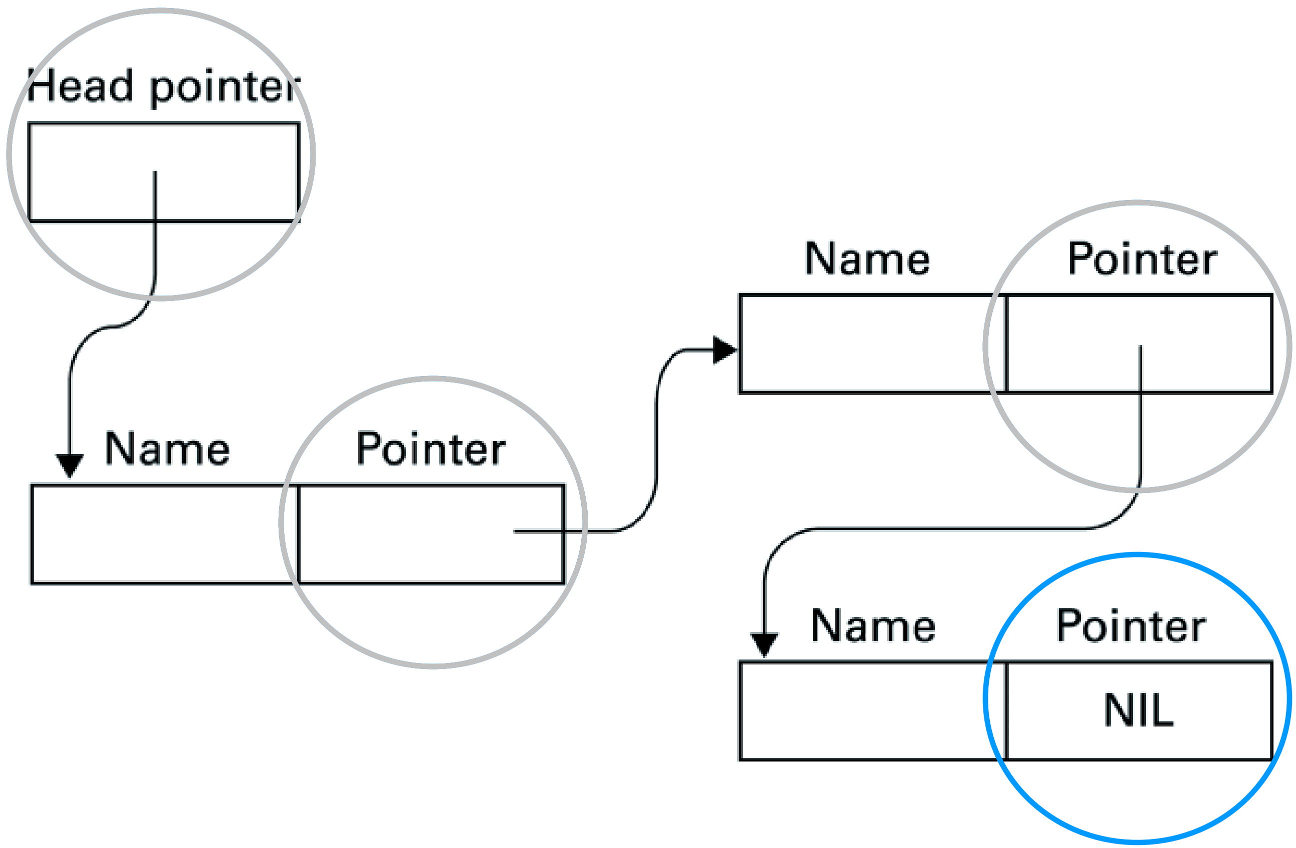
- To store an ordered list of names we could use 2-D homogeneous array (in C: `char Names[10][8]`)



- However:
  - addition & removal of names requires expensive data movements!

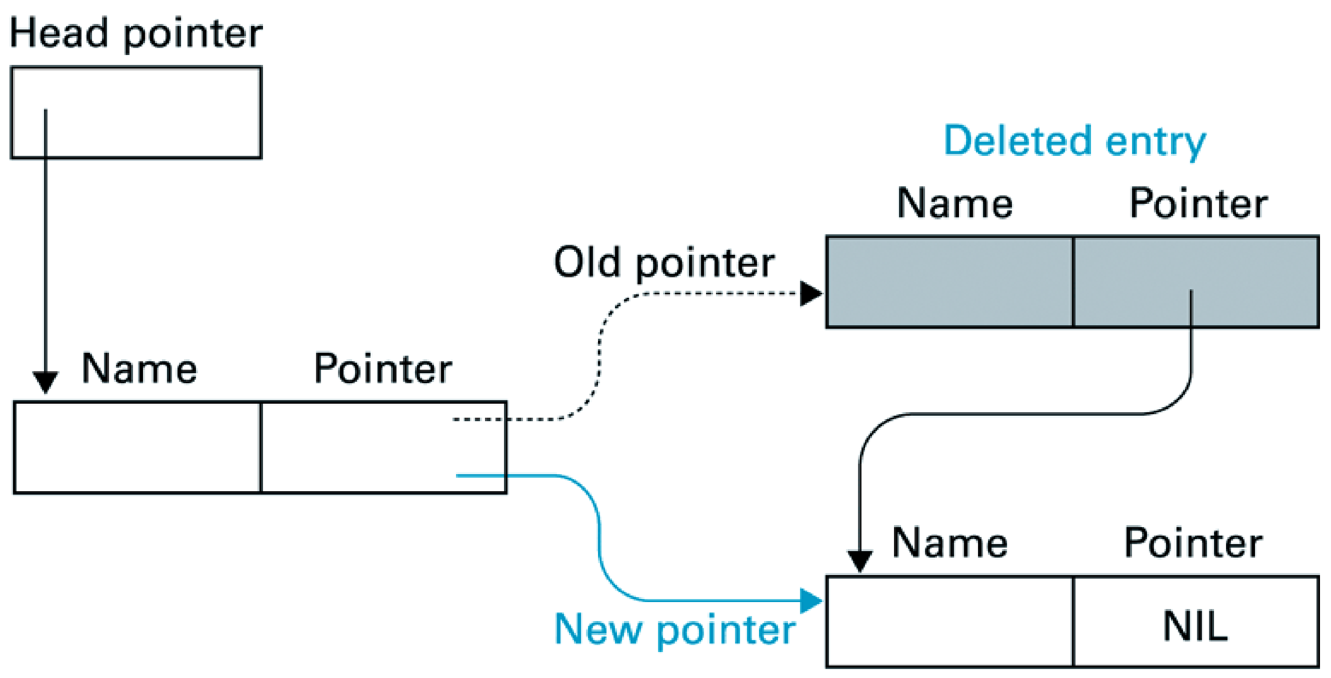
7.3: Linked Lists

- Data movements can be avoided by using a 'linked list', including pointers to list entries



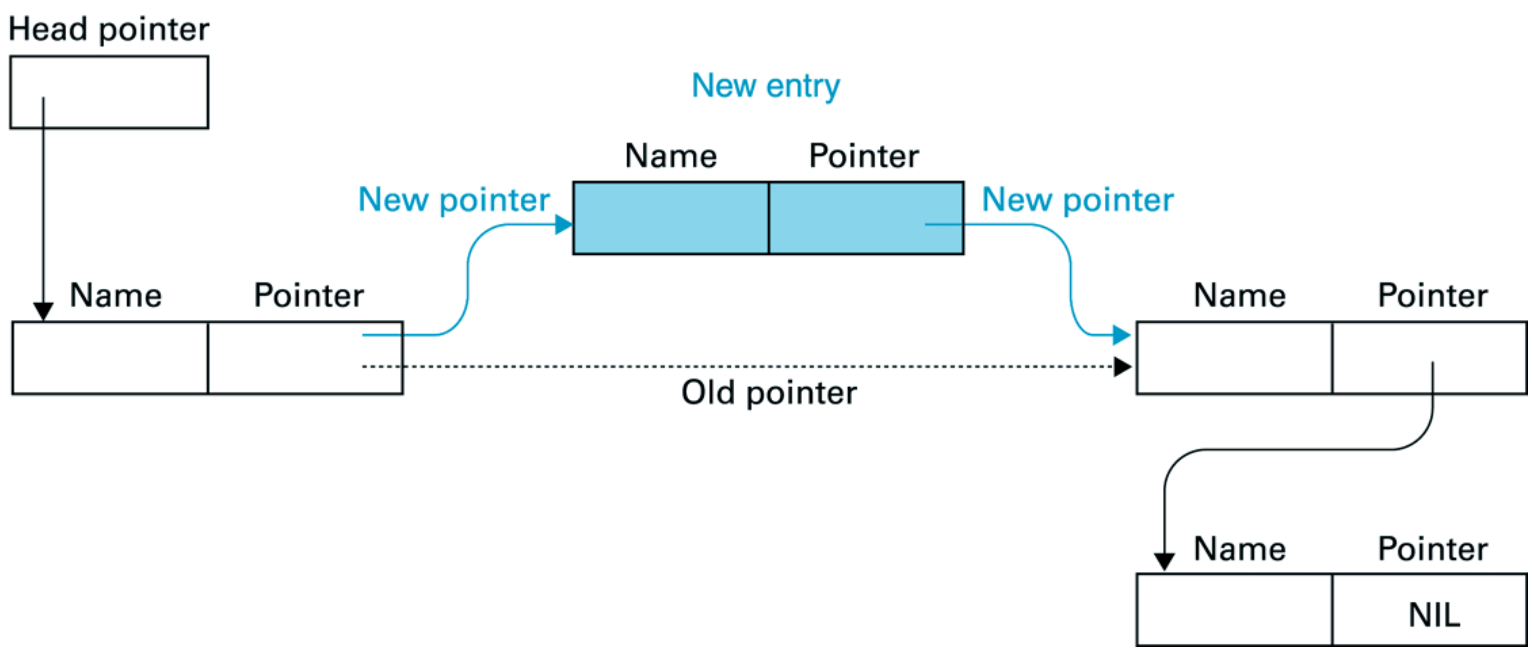
7.3: Deleting an Entry from a Linked List

- A list entry is removed by changing a single pointer:



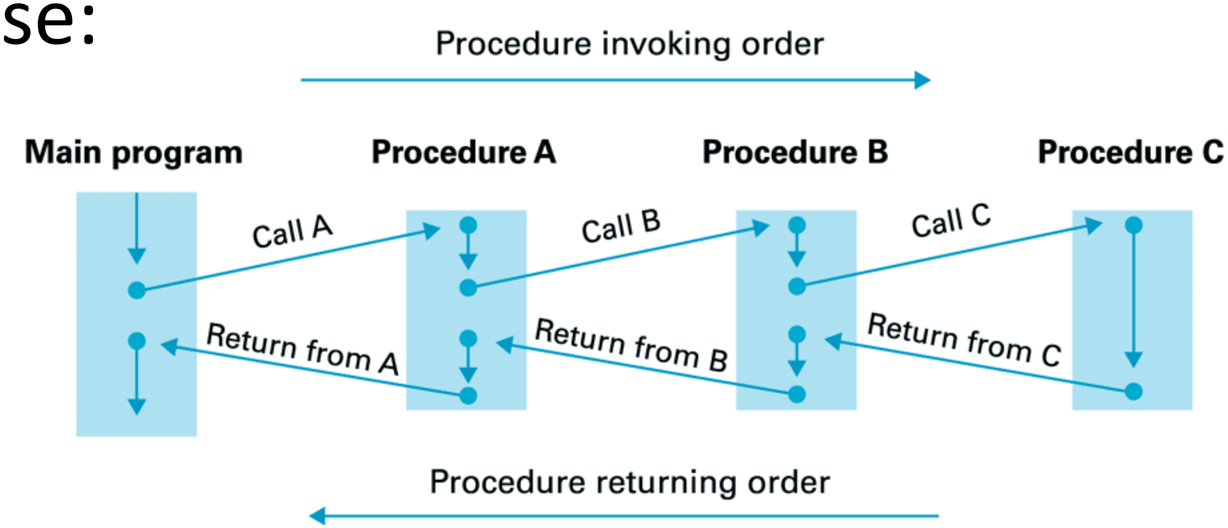
7.3: Inserting an Entry into a Linked List

- A new entry is inserted by setting pointer of
  - (1) new entry to address of entry that is to follow
  - (2) preceding entry to address of new entry:

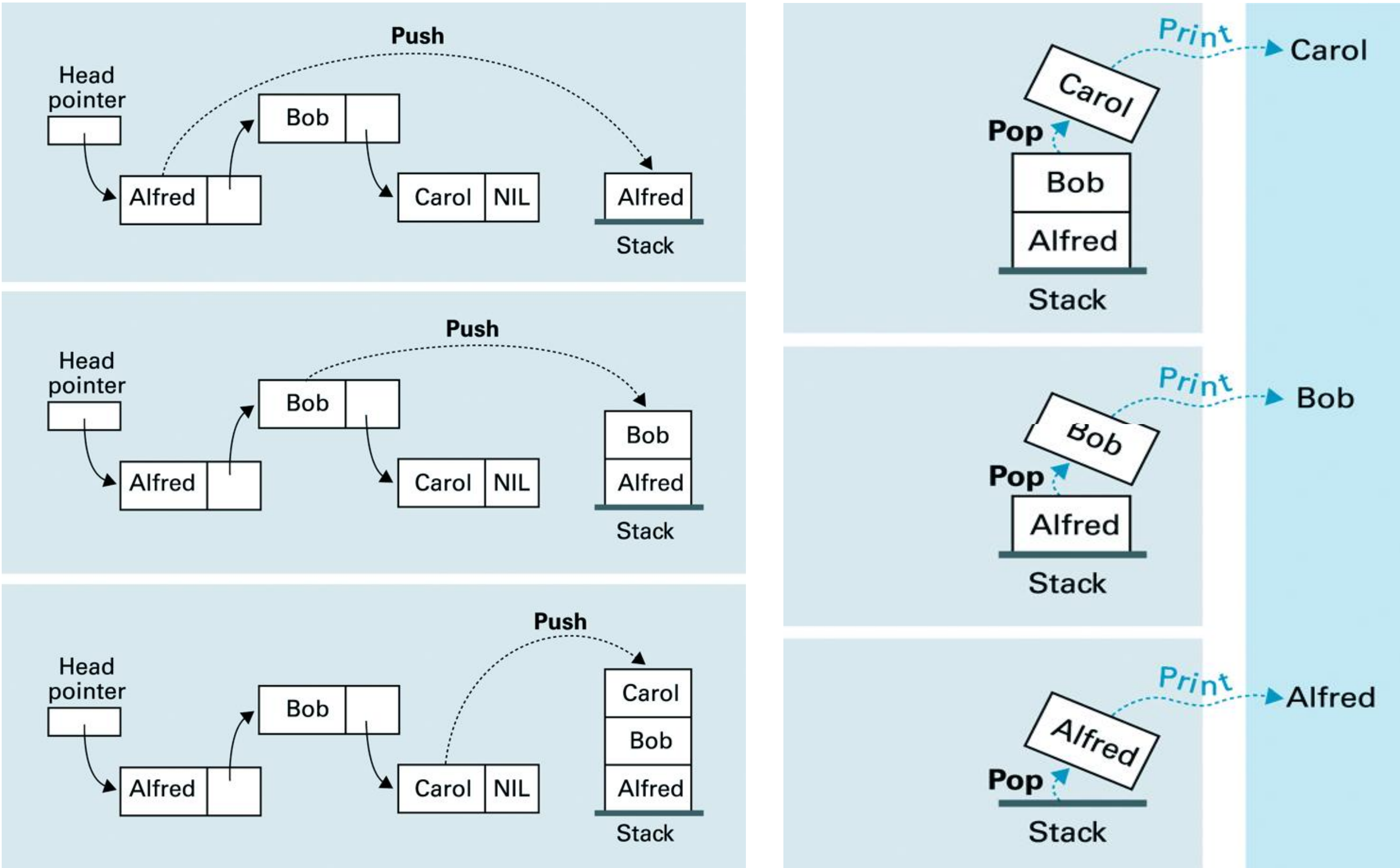


7.4: Stacks

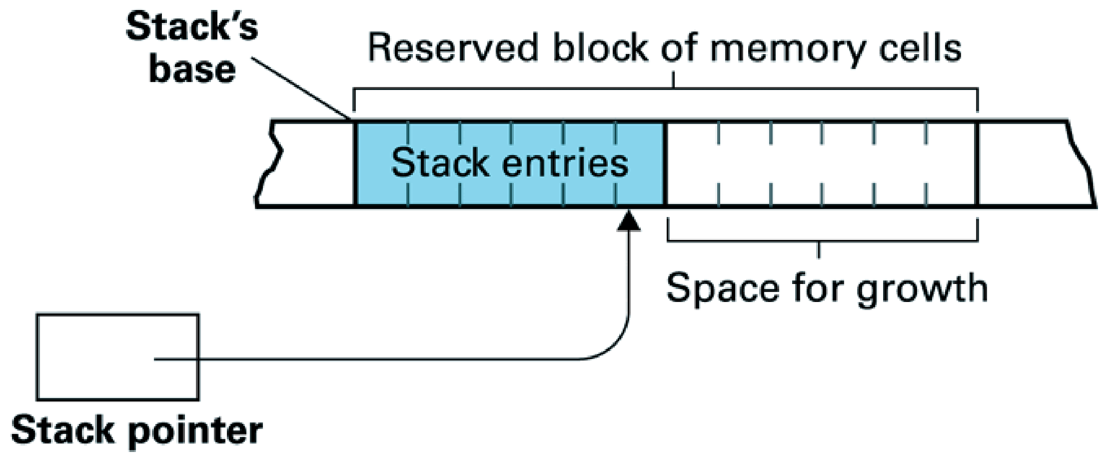
- Disadvantage of contiguous array structures:
  - insertion / removal requires costly data movements
- Still okay if insertion / removal restricted to end of array => stack (with *push* & *pop* operations)
- Typical use:



7.4: Push / Pop (to print inverse linked list)



7.4: A Stack in Memory



- Here:
  - conceptual structure close to identical to actual structure in memory
- If maximum stack-size unknown:
  - pointers can be used (=> conceptual = actual structure)

Chapter 7 - Data Structures: Conclusions

- Pointers:
  - basic aid in definition of *dynamic* data structures
- Often used data structures:
  - Arrays
  - Lists
  - Stacks
  - …

C H A P T E R 8

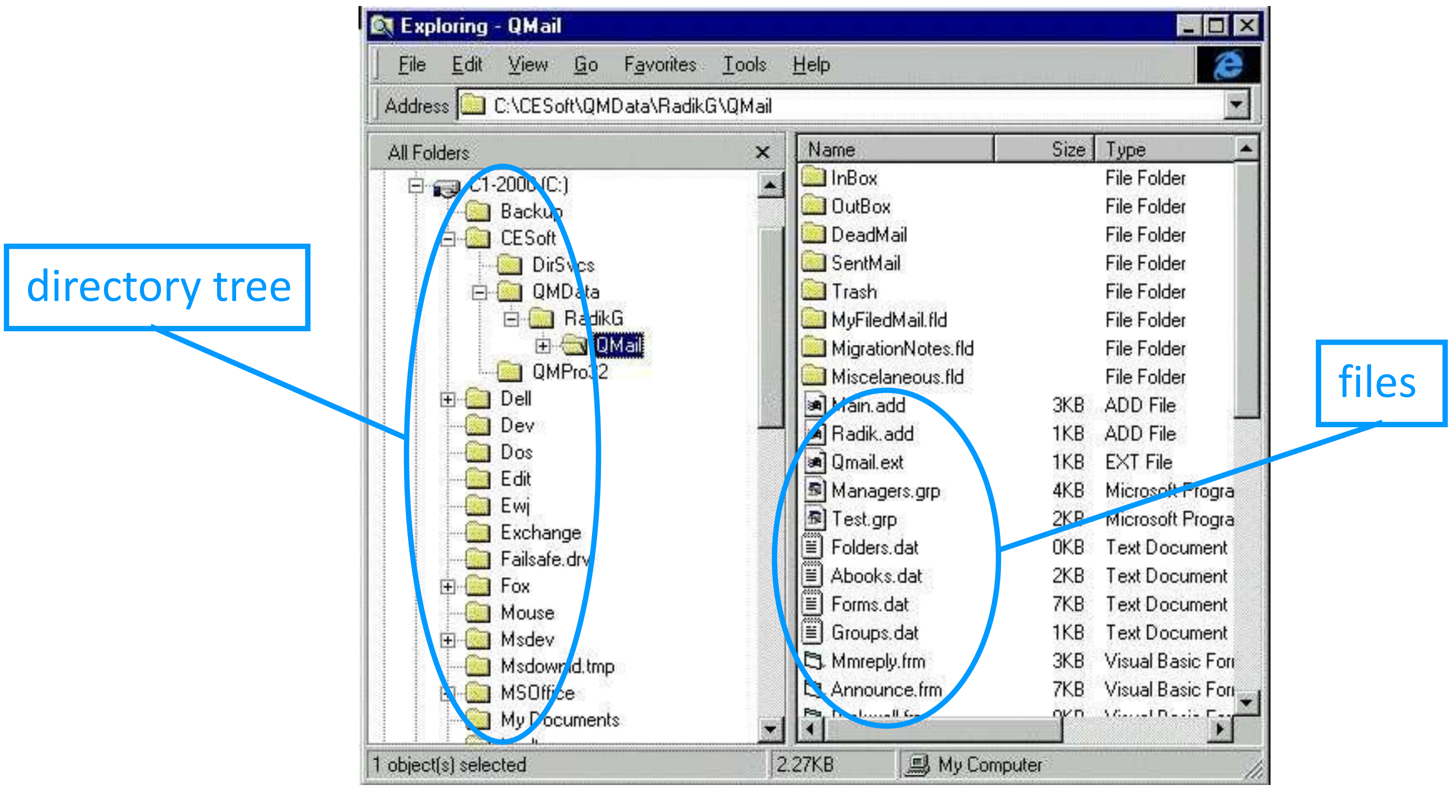
File Structures

Reference: Computer Science an Overview
Author: J. Glenn Brook Shear
6th Edition

- Abstractions of the actual data organization on *mass storage*
- Again: differences between *conceptual* and *actual* data organization

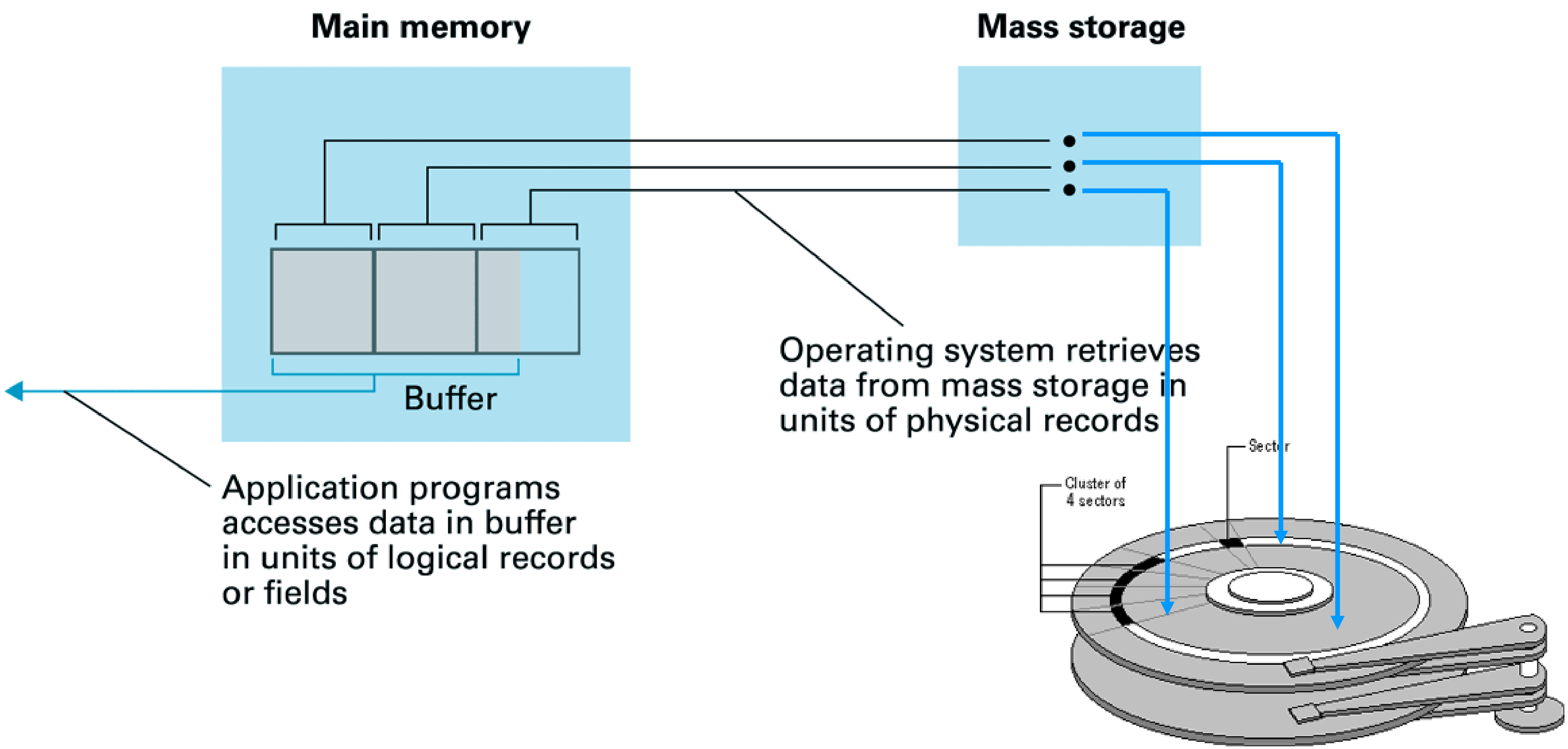
8.1: Files, Directories & the Operating System

- OS storage structure:
  - conceptual hierarchy of *directories* and *files*



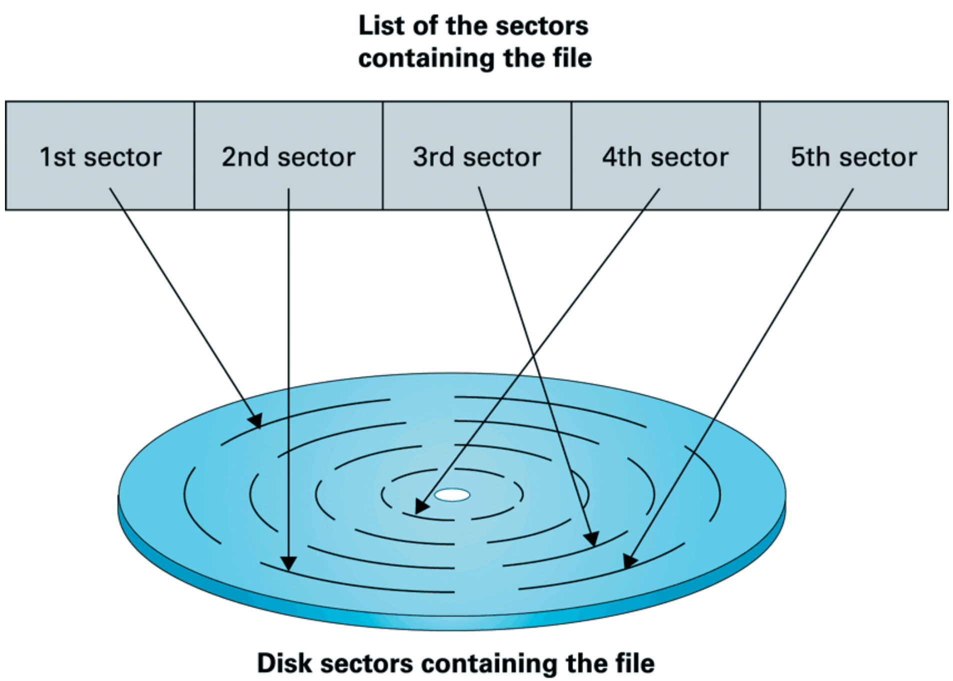
8.1: Files: Conceptual vs. Actual View

- View at OS-level is conceptual
  - actual storage may differ significantly!



8.2: Sequential Files

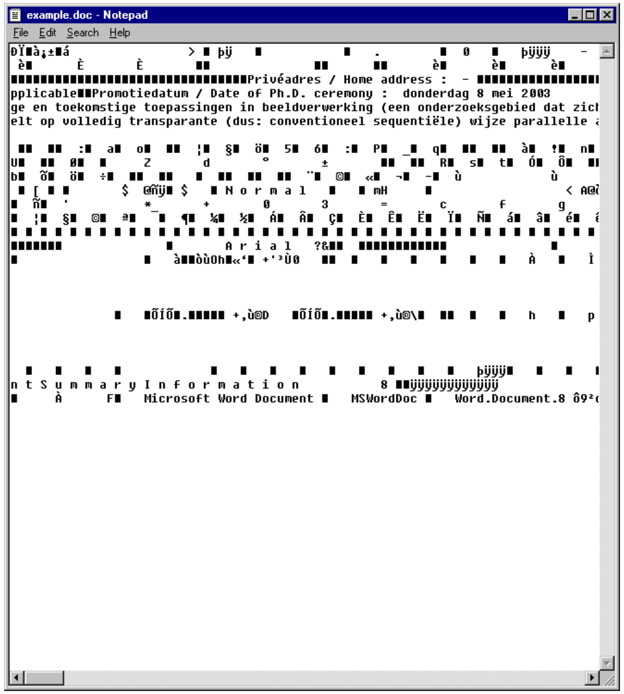
- To ‘remember’ where data resides on disk, the OS maintains a list of sectors for each file



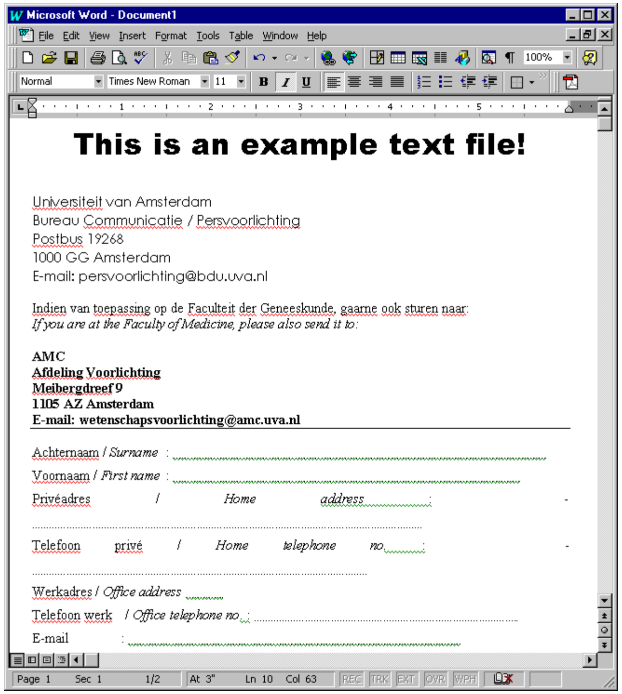
- Result: *sequential view* of scattered set of data

8.2: Text Files

- Sequential file consisting of long string of encoded characters (e.g. ASCII-code)
  - But: character-string still interpreted by word processor!

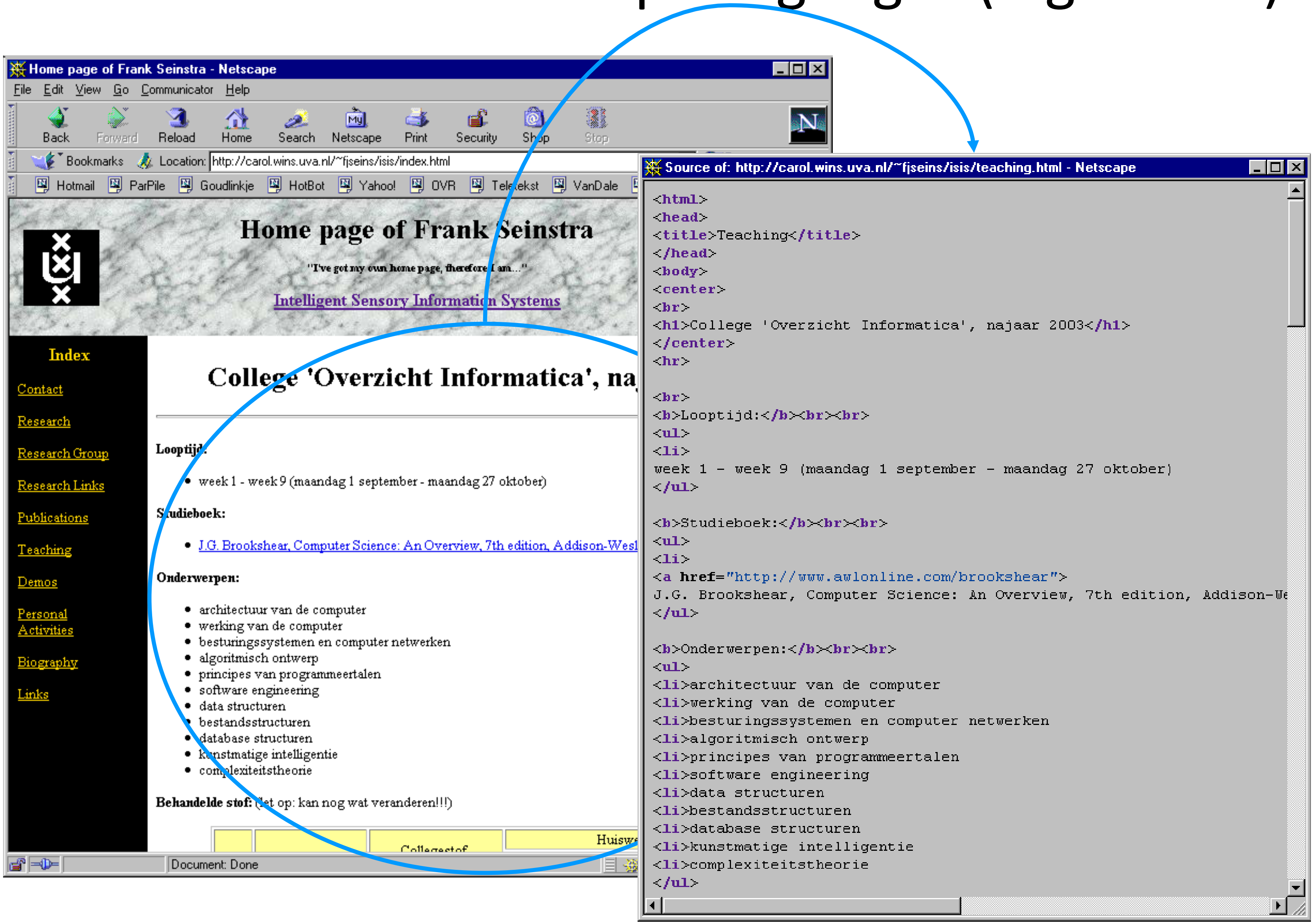


File in "Notepad"

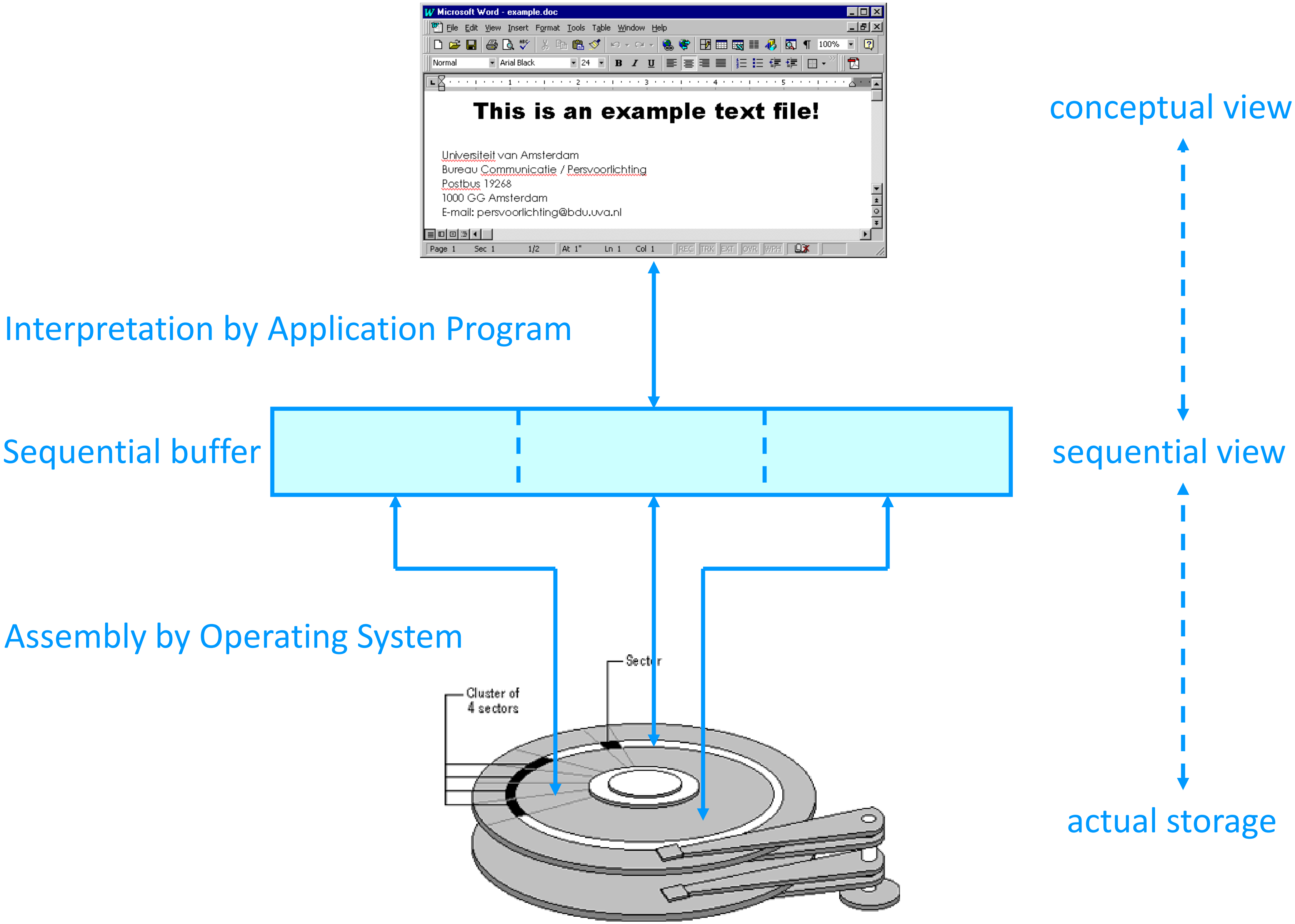


Same file in "MS Word"

8.2: Text files & Markup Languages (e.g. HTML)

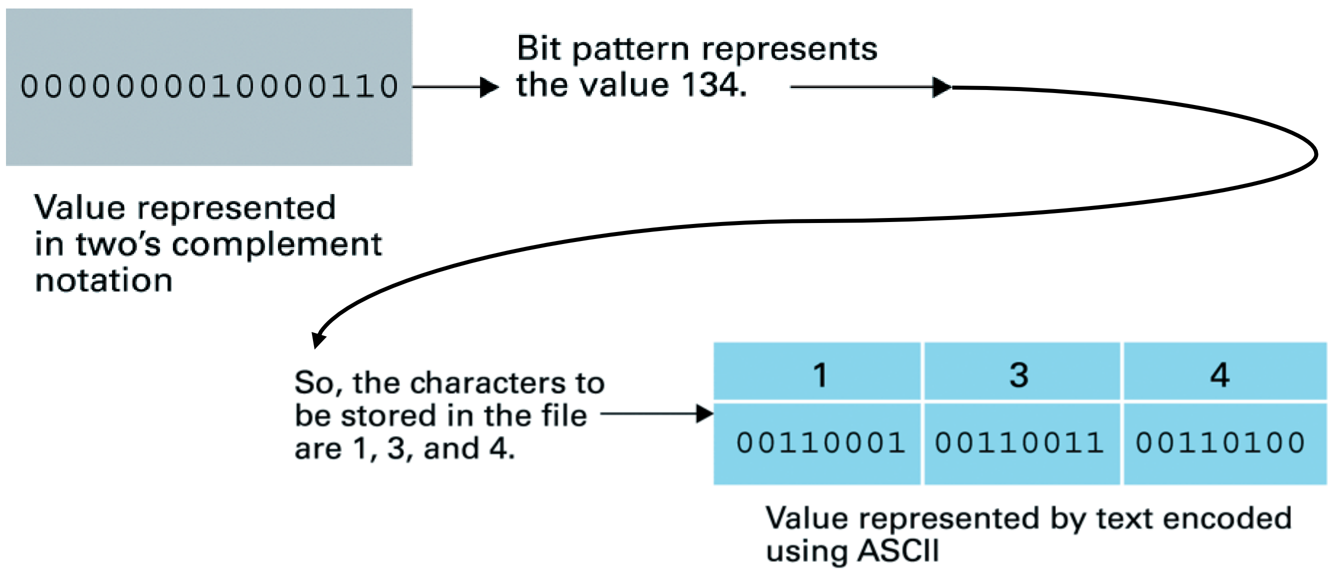


8.2: From actual storage to conceptual



8.2: Data Conversion

- When programming: note that data transfer to/from file may involve data conversion:
  - e.g., from two's complement notation to ASCII:



- So: again it's about the *interpretation* of data

8.3: Quick File Access

- Disadvantage of sequential files:
  - no quick access to particular file data
- Two techniques to overcome this problem:
  - (1) *Indexing* or (2) *Hashing*
- Indexing:

loaded into main memory when opened

Indexed File

12N67	John Smith	23-Jul-71	17,000.00	New York	…
13C08	Andrew White	27-Jun-70	24,500.00	Boston	…
23G19	Mary Jackson	5-Mar-39	41,000.00	San Francisco	…
24X17	Eleanor Tracy	17-Sep-63	9,635.00	Fort Lauderdale	…
26X28	Michael Flanagan	1-Nov-44	18,800.00	Washington	…
32E76	Glenn White	29-Feb-68	17,000.00	Detroit	…
36Z05	Virginia Moore	27-Jun-70	32,000.00	San Francisco	…
:	:	:	:	:	…
:	:	:	:	:	…
:	:	:	:	:	…

keys

Index

12N67	location
13C08	location
23G19	location
24X17	location
26X28	location
32E76	location
36Z05	location
:	:
:	:
:	:

Chapter 8 - Problem 10

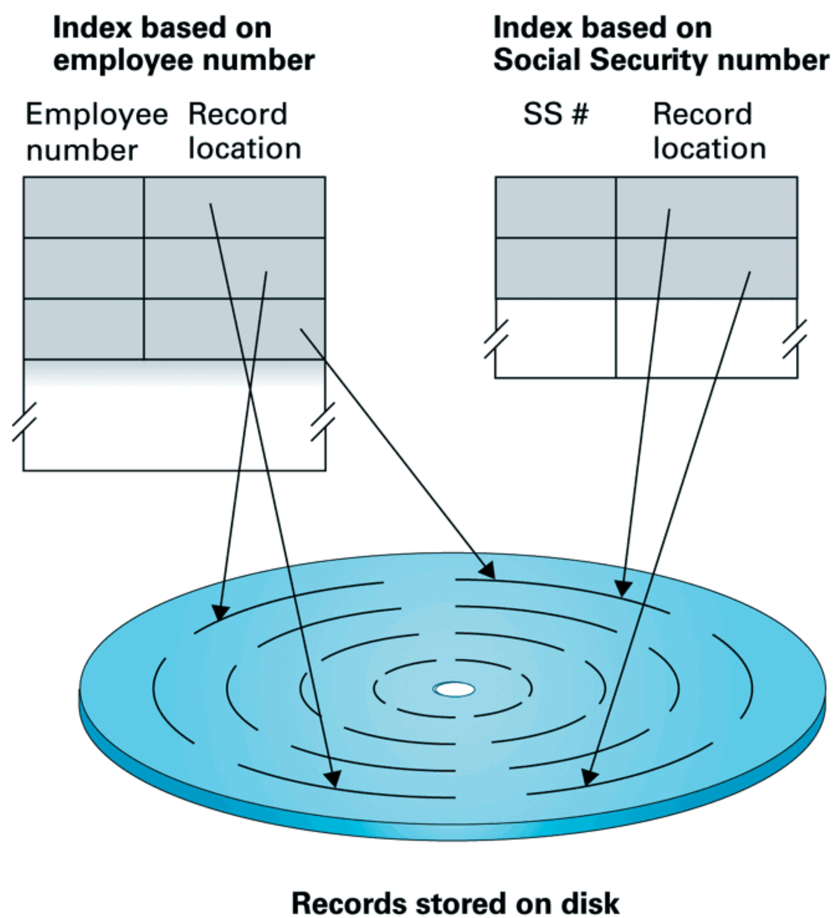
Why is a ‘*patient identification number*’ a better choice for a key field than the last name of each patient?

- If key unique:
  - additional sequential search never required

- Patient’s last name is not always unique

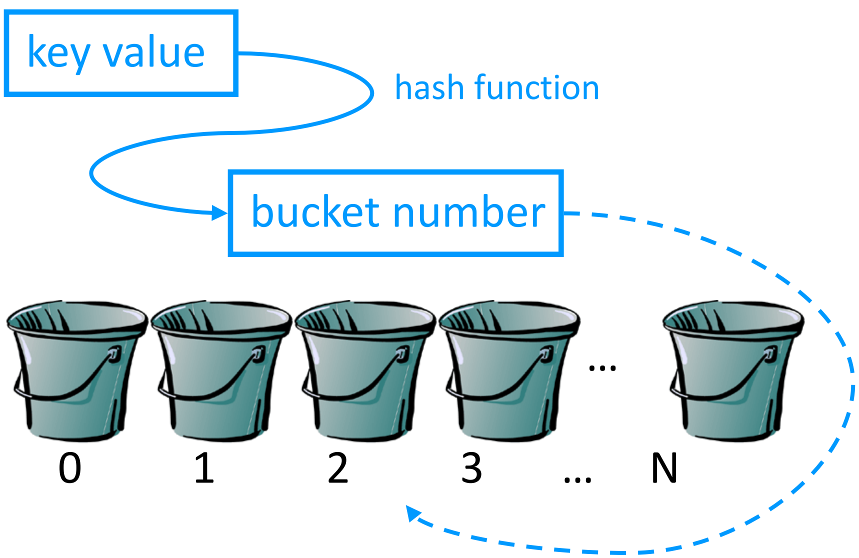
8.3: Inverted Files

- Variation to (single) indexing: inverted file



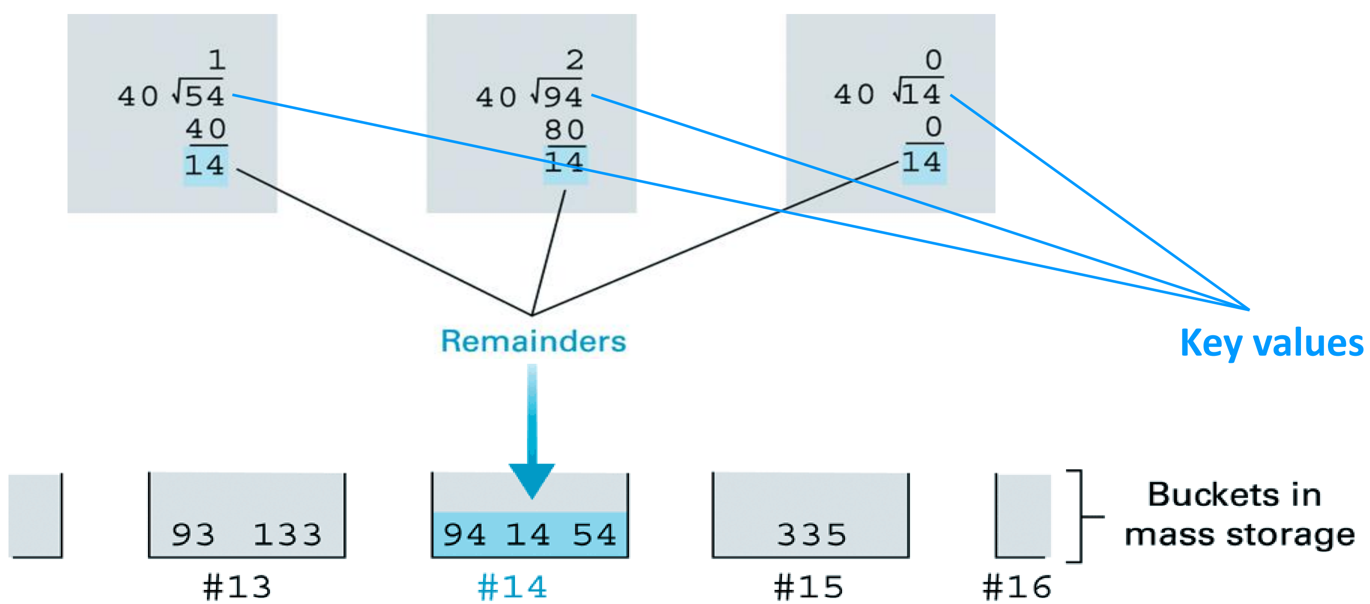
8.4: Hashing

- Disadvantage of indexing is… the index
  - requires extra space
- Solution: '*hashing*'
  - finds position in file using a key value (as in indexing)…
  - … simply by identifying location *directly from the key*
- How?
  - define set of '*buckets*' & '*hash function*' that converts keys to bucket numbers



8.4: Hash Function: Example

- If storage space divided into *40 buckets* and hash function is *division*:
  - key values 14, 54, & 94 all map onto same bucket (collision)



Chapter 8 - File Structures: Conclusions

- File Structures:
  - abstractions of actual data organization on mass storage
- Changes of ‘view’:
  - actual storage -> sequential view by OS -> conceptual view presented to user
- Quick access to particular file data by
  - (1) indexing (many forms)
  - (2) hashing (requires no index, *but requires bucket search*!)


Any
Questions