

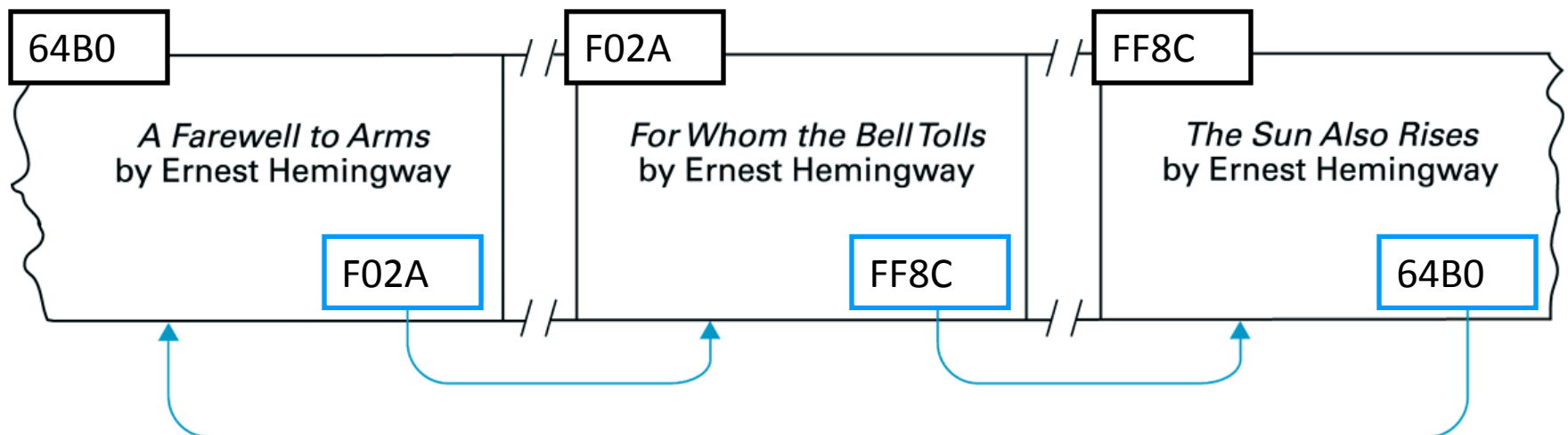
# CHAPTER 7

## 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:

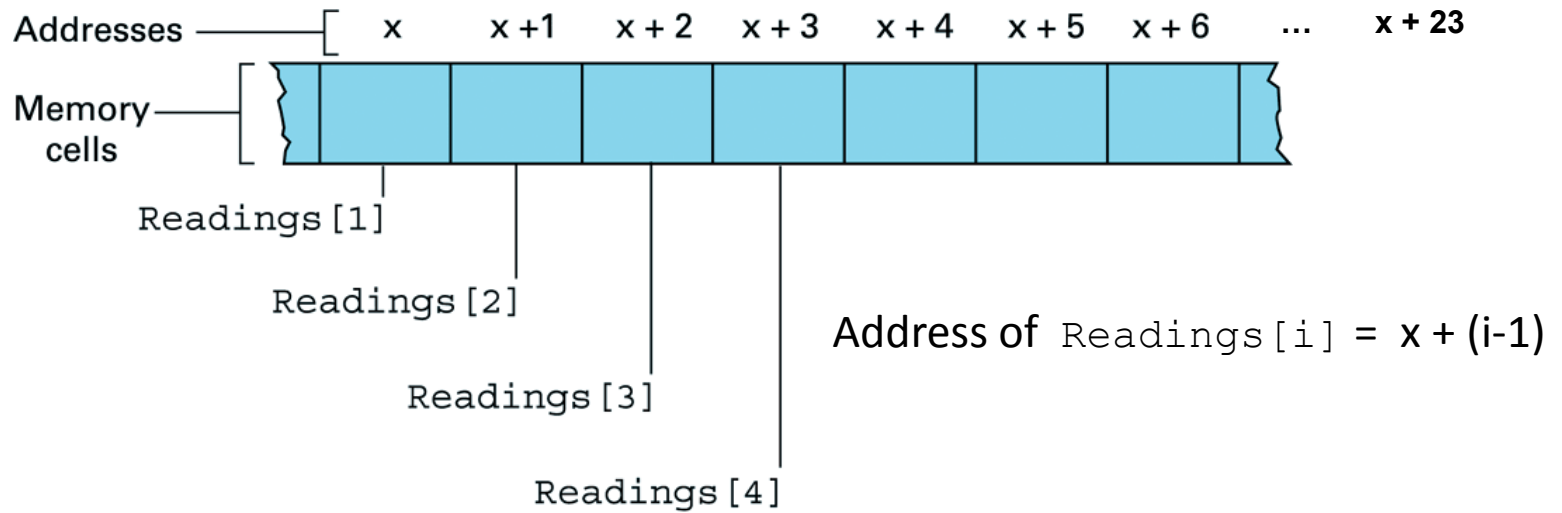

- Dynamic:
  - shape & size may change
  - example: Stack

Stack:

32
65
97
48
17

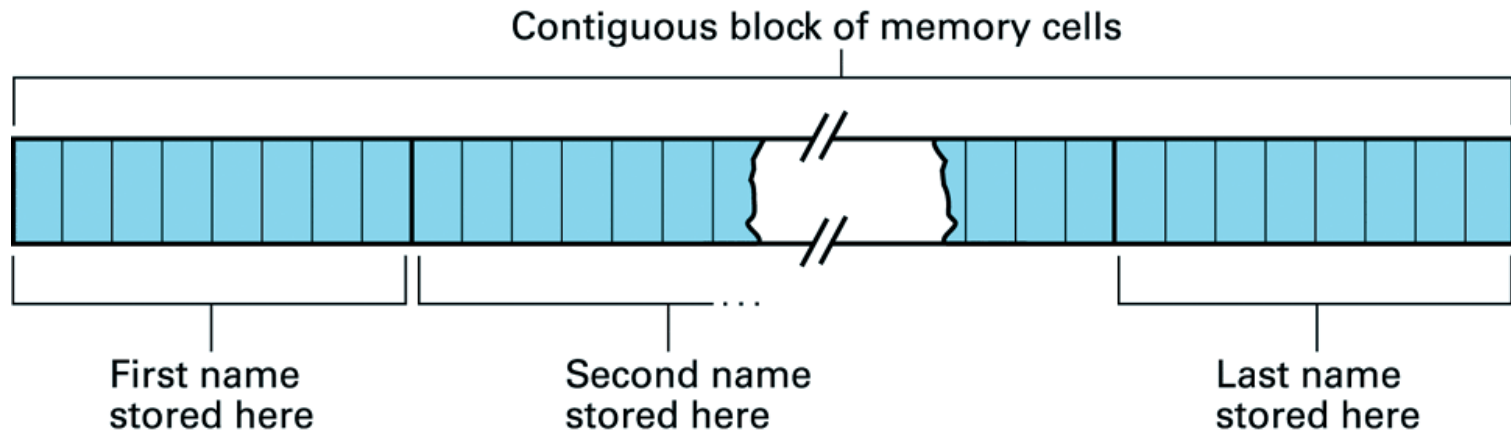
## 7.2: Arrays

- 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:



## 7.3: Lists

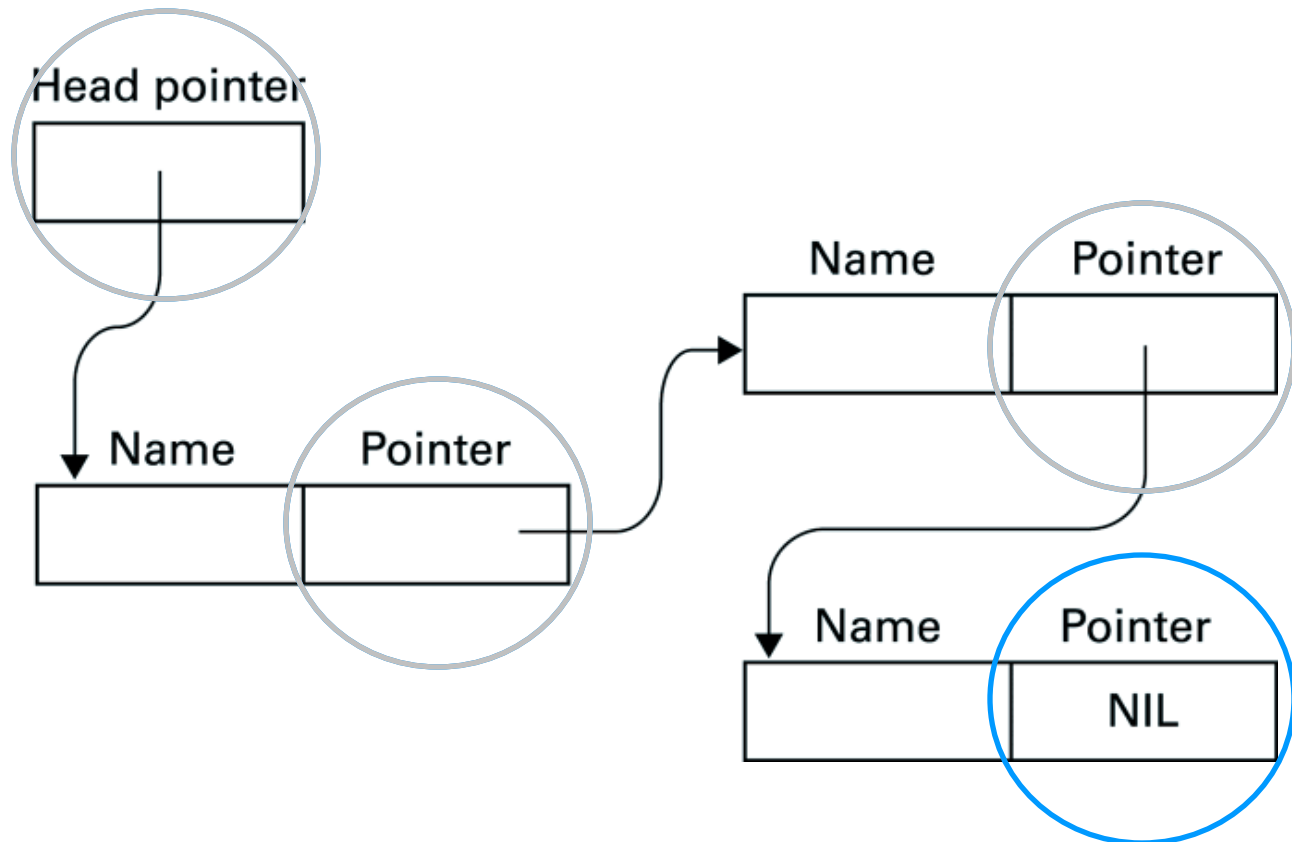
- To store an ordered list of names we could use 2-D homogeneous array (in C: `char Names[10][8]`)



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

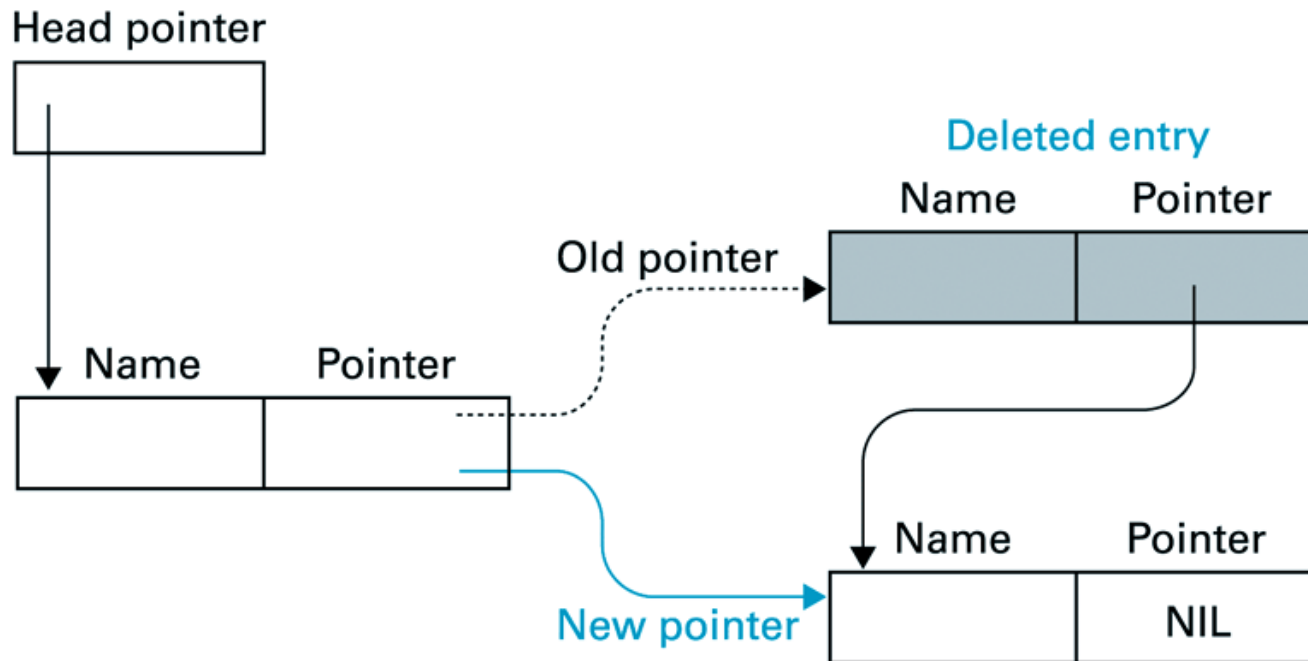
## 7.3: Linked Lists

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



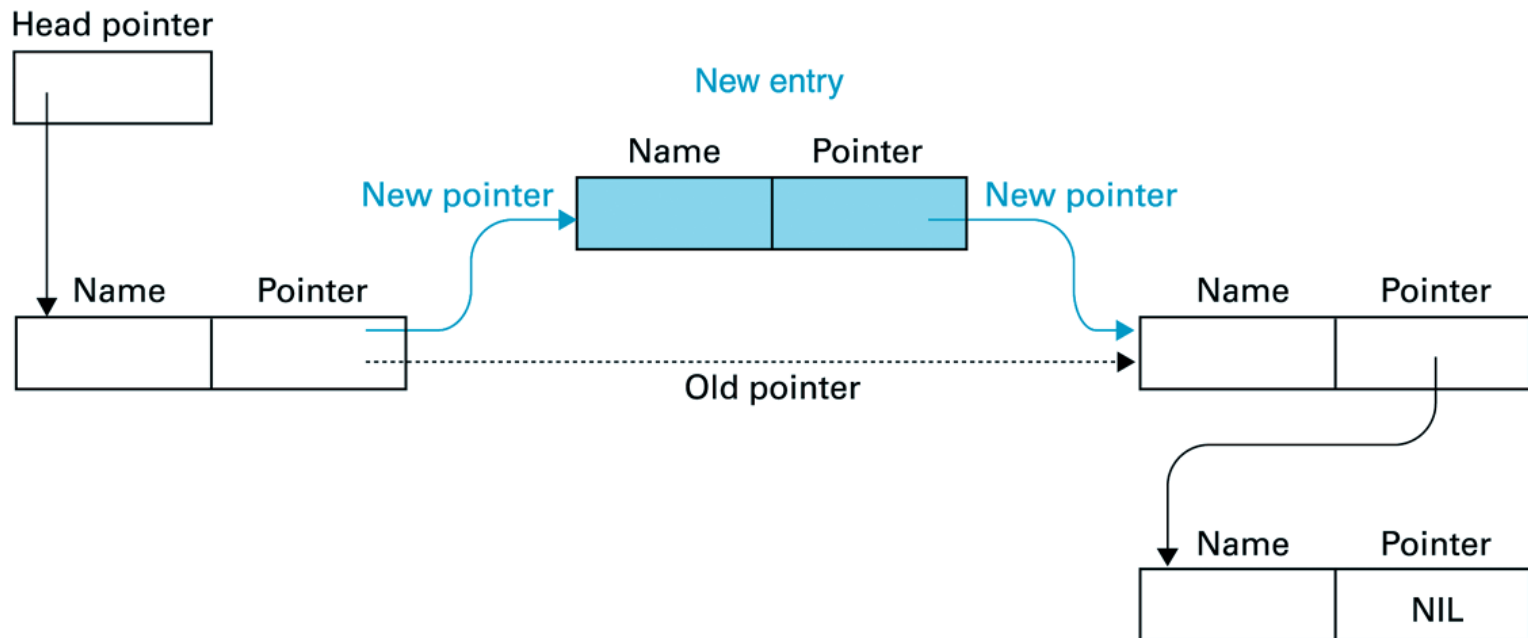
## 7.3: Deleting an Entry from a Linked List

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



## 7.3: Inserting an Entry into a Linked List

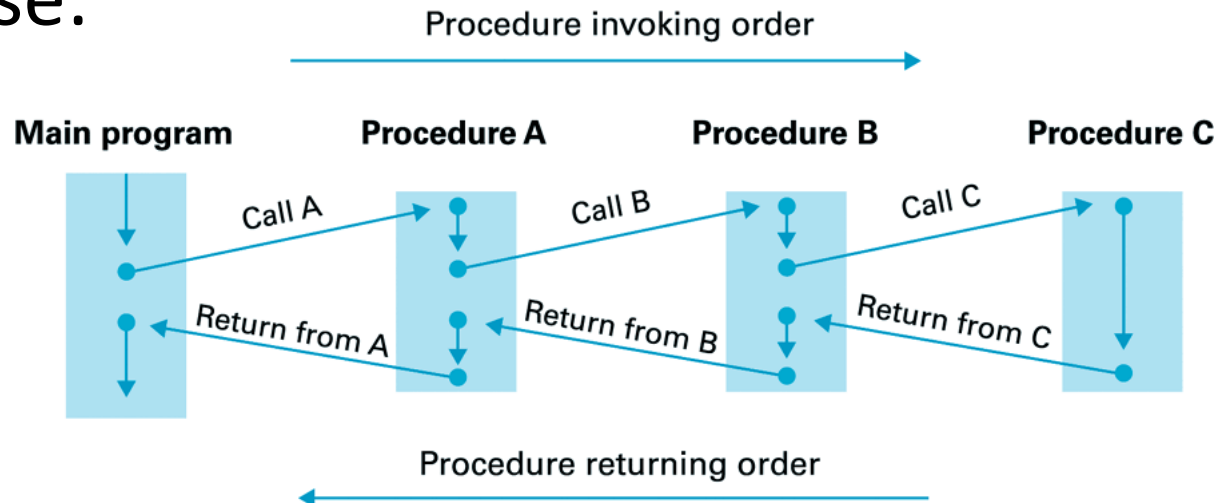
- A new entry is inserted by setting pointer of
  - (1) new entry to address of entry that is to follow
  - (2) preceding entry to address of new entry:



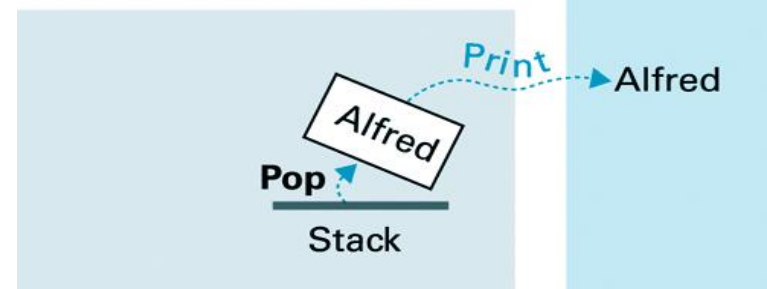
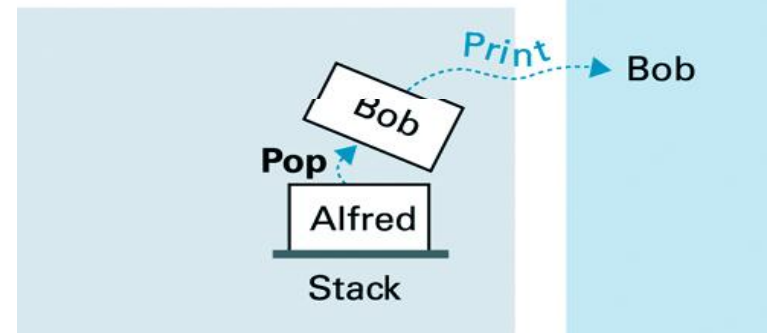
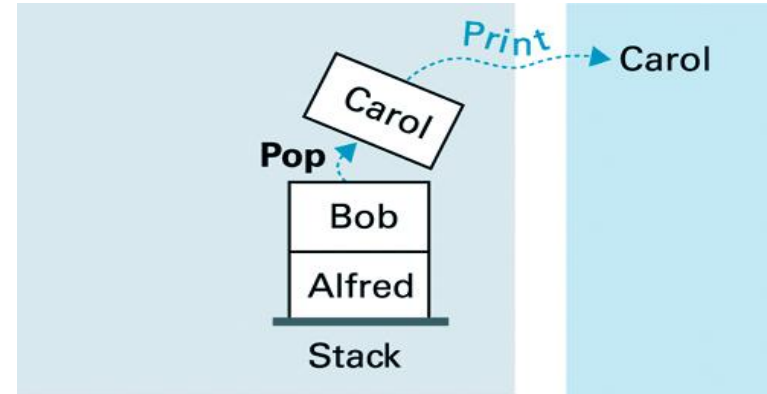
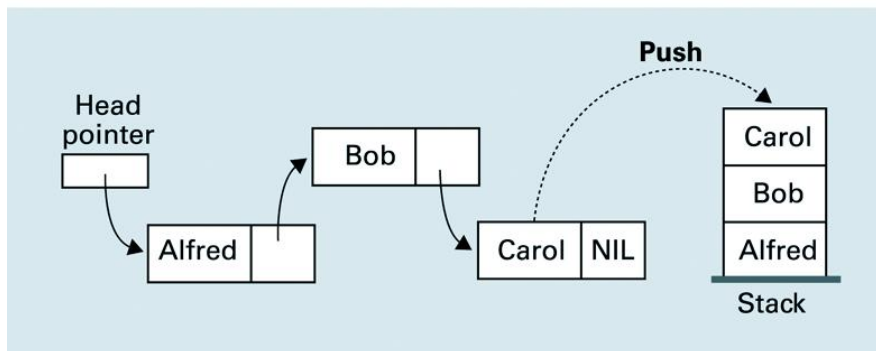
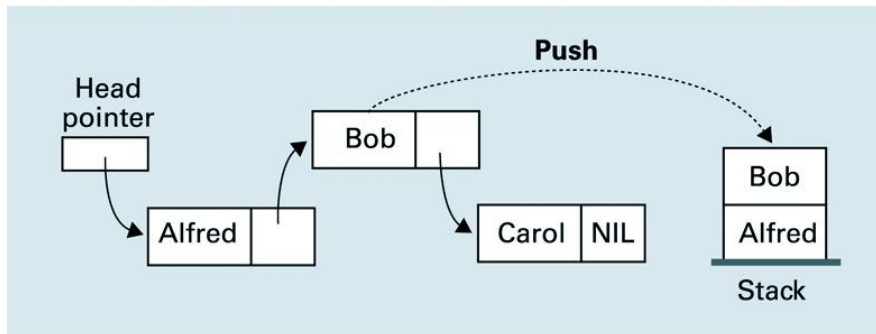
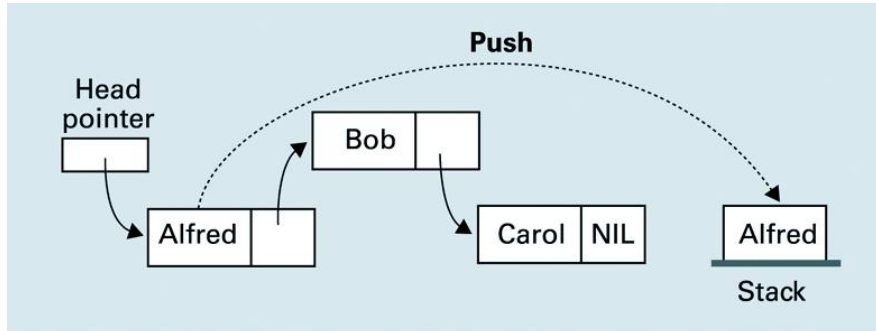


# 7.4: Stacks

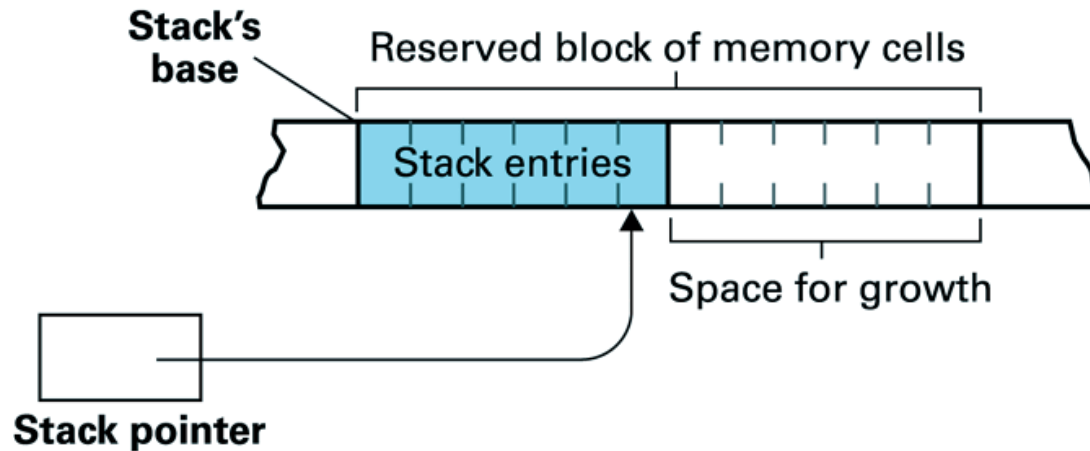
- Disadvantage of contiguous array structures:
  - insertion / removal requires costly data movements
- Still okay if insertion / removal restricted to end of array => stack (with *push* & *pop* operations)
- Typical use:



# 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 (  $\Rightarrow$  conceptual = actual structure )

# Chapter 7 - Data Structures:

## Conclusions

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

# CHAPTER 8

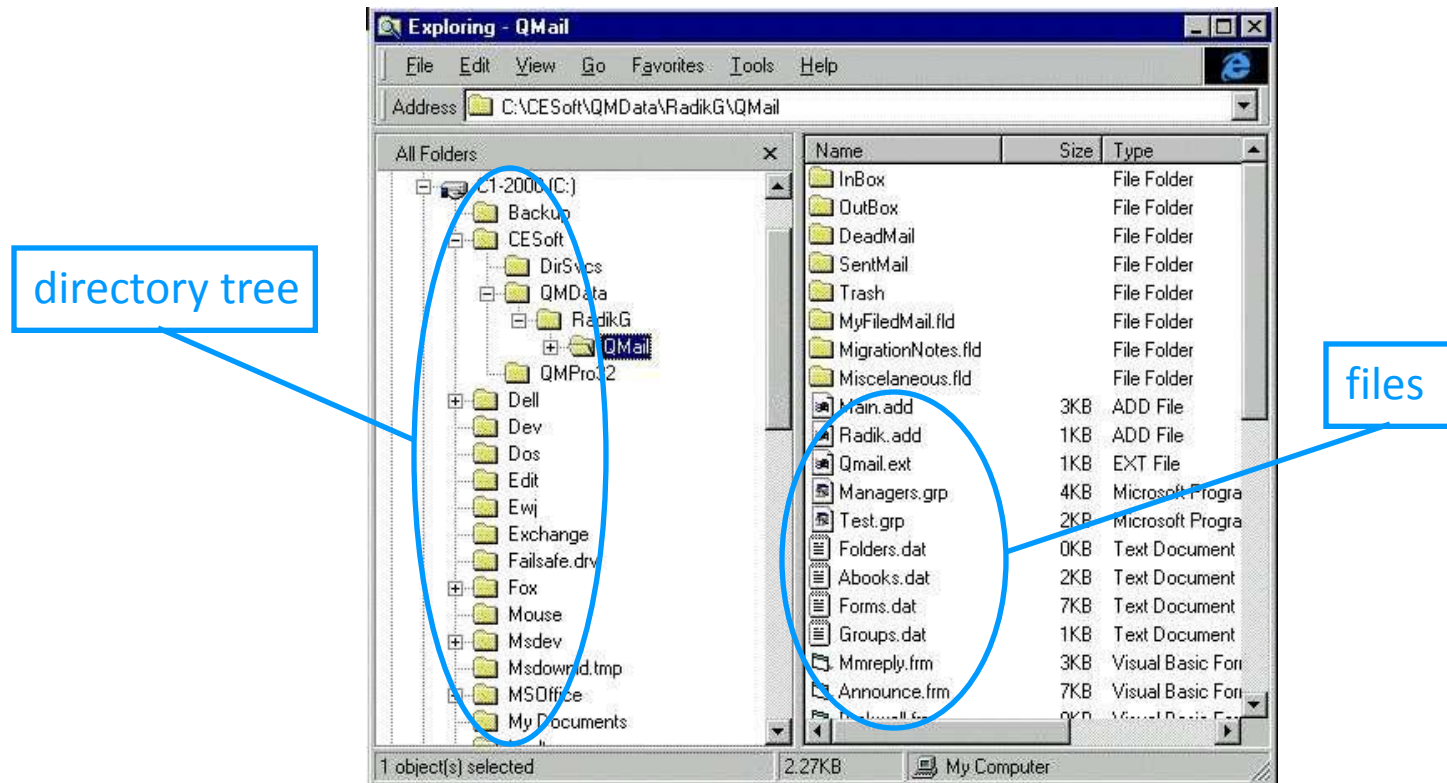
## File Structures

Reference: Computer Science an Overview  
Author: J. Glenn Brook Shear  
6<sup>th</sup> Edition

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

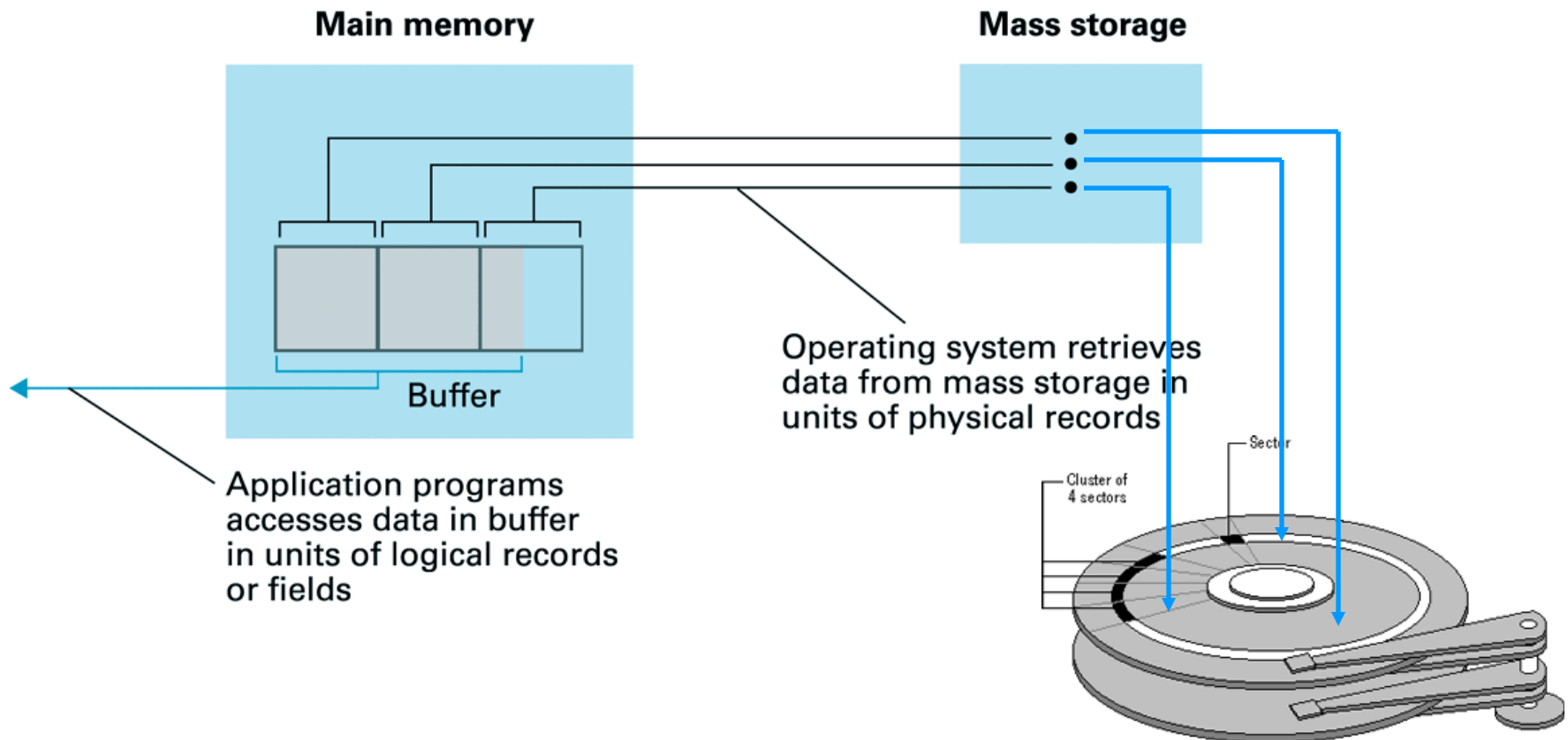
# 8.1: Files, Directories & the Operating System

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



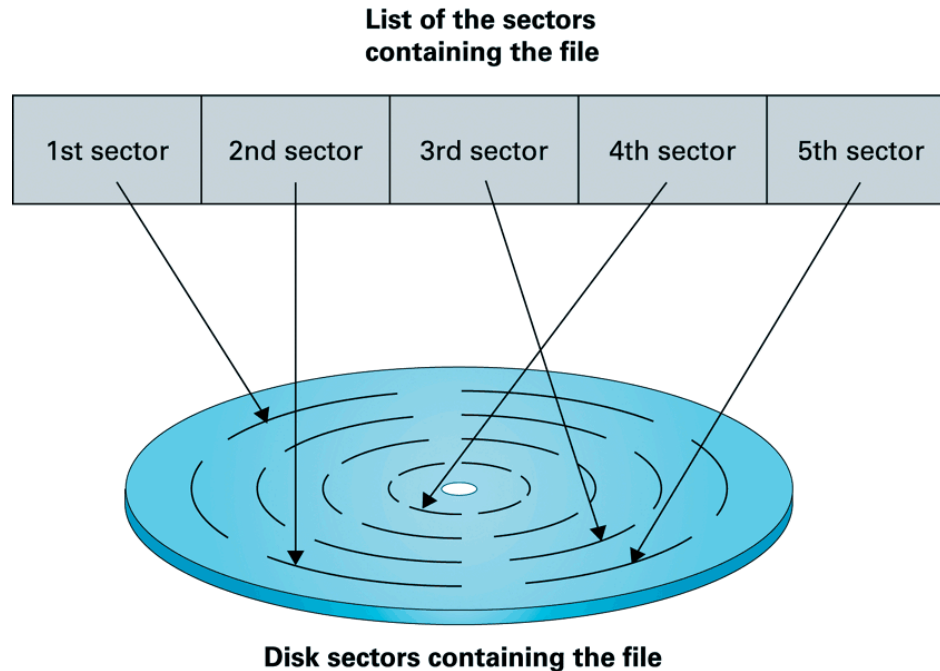
# 8.1: Files: Conceptual vs. Actual View

- 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

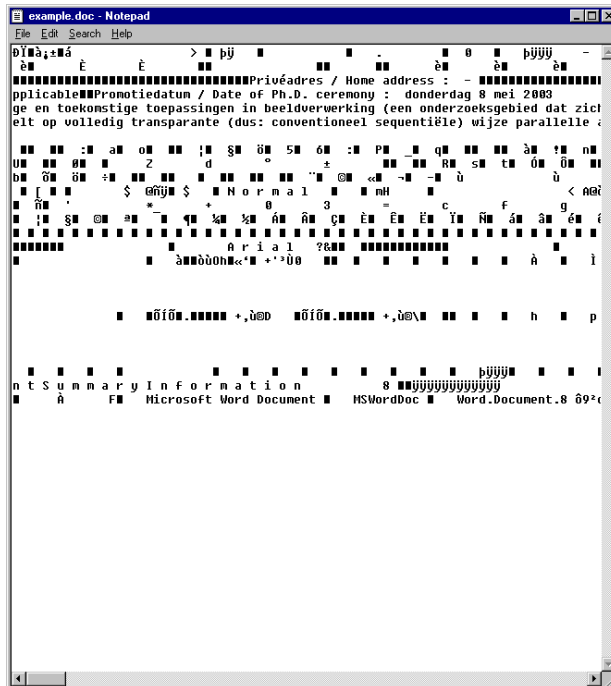


- Result: *sequential view* of scattered set of data

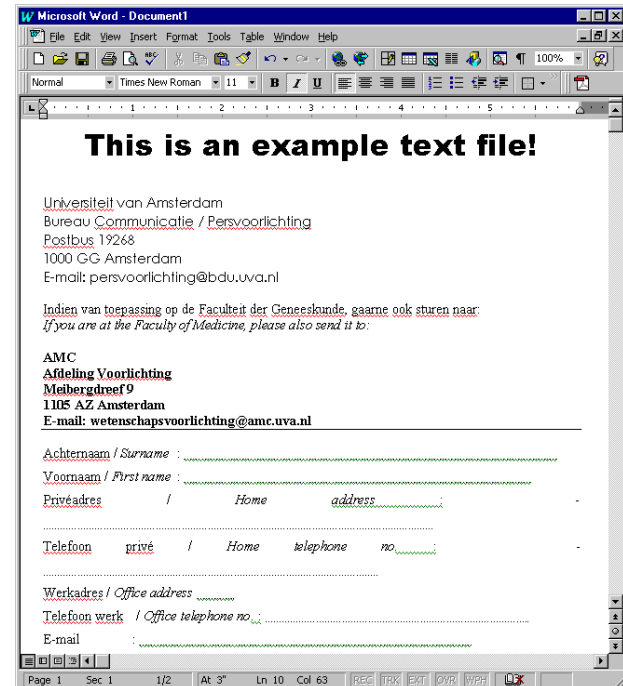


## 8.2: Text Files

- 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)

The screenshot displays a Netscape browser window with the address bar showing `http://carol.wins.uva.nl/~fjseins/isis/index.html`. The page title is "Home page of Frank Seinstra". The main content area features a header with a logo and the text "Home page of Frank Seinstra" and "Intelligent Sensory Information Systems". Below this is a section titled "College 'Overzicht Informatica', na...". The sidebar on the left contains a list of links: Index, Contact, Research, Research Group, Research Links, Publications, Teaching, Demos, Personal Activities, Biography, and Links. The main content area lists topics under "Onderwerpen:" and "Behandelde stof:". The source code window on the right shows the HTML structure, including tags for headings, lists, and links. Blue arrows connect the source code to the rendered page elements.

**Home page of Frank Seinstra**

"I've got my own home page, therefore I am..."

Intelligent Sensory Information Systems

**Index**

[Contact](#)

[Research](#)

[Research Group](#)

[Research Links](#)

[Publications](#)

[Teaching](#)

[Demos](#)

[Personal Activities](#)

[Biography](#)

[Links](#)

**College 'Overzicht Informatica', na...**

**Looptijd:**

- week 1 - week 9 (maandag 1 september - maandag 27 oktober)

**Studieboek:**

- [J.G. Brookshear, Computer Science: An Overview, 7th edition, Addison-Wesley](#)

**Onderwerpen:**

- architectuur van de computer
- werking van de computer
- besturingssystemen en computer netwerken
- algoritmisch ontwerp
- principes van programmeertalen
- software engineering
- data structuren
- bestandsstructuren
- database structuren
- kunstmatige intelligentie
- complexiteitstheorie

**Behandelde stof:** (let op: kan nog wat veranderen!!)

**Source of: `http://carol.wins.uva.nl/~fjseins/isis/teaching.html` - Netscape**

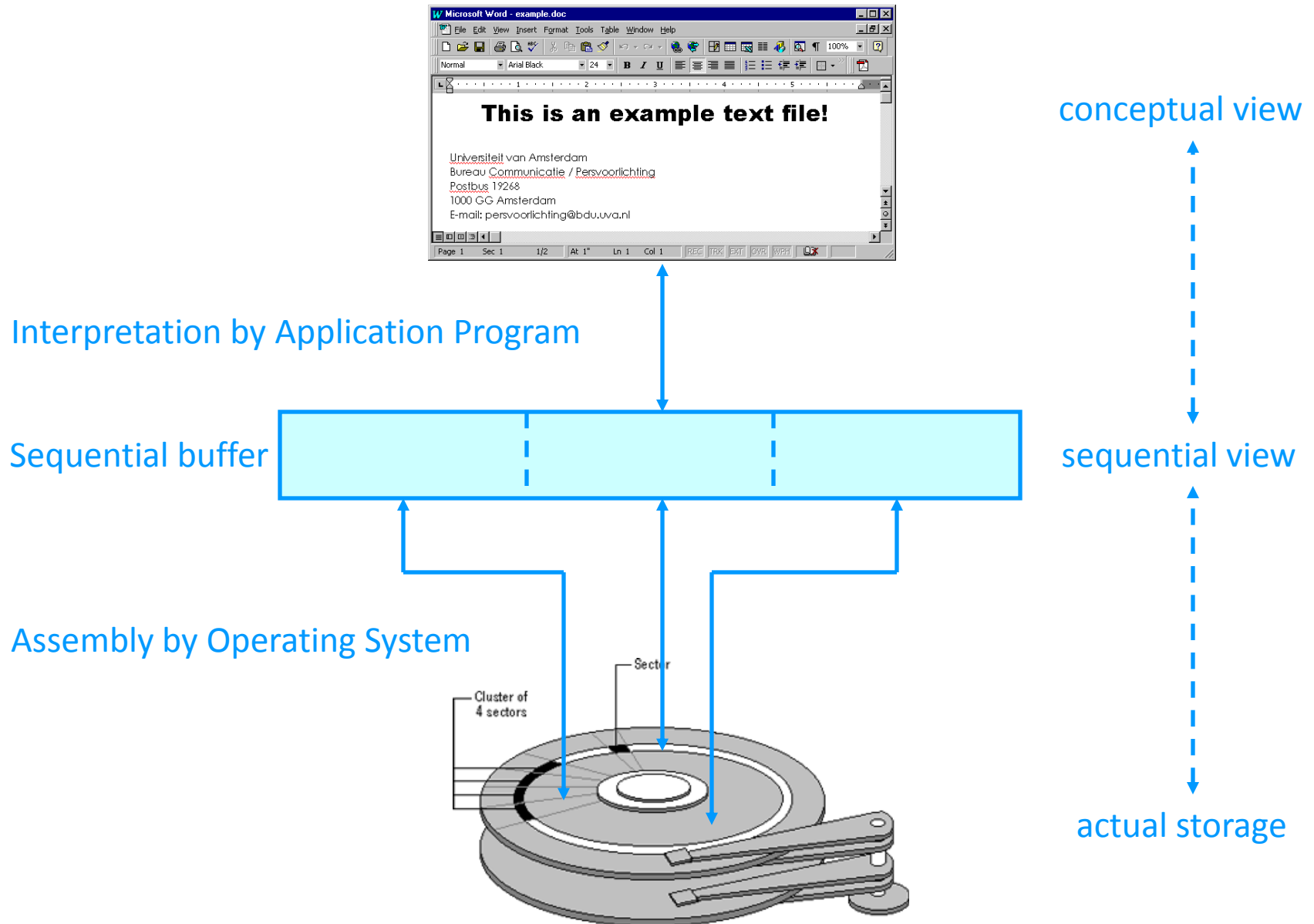
```
<html>
<head>
<title>Teaching</title>
</head>
<body>
<center>
<br>
<h1>College 'Overzicht Informatica', najaar 2003</h1>
</center>
<hr>

<br>
<b>Looptijd:</b></b><br><br>
<ul>
<li>
week 1 - week 9 (maandag 1 september - maandag 27 oktober)
</li>
</ul>

<b>Studieboek:</b></b><br><br>
<ul>
<li>
<a href="http://www.awlonline.com/brookshear">
J.G. Brookshear, Computer Science: An Overview, 7th edition, Addison-W
</li>
</ul>

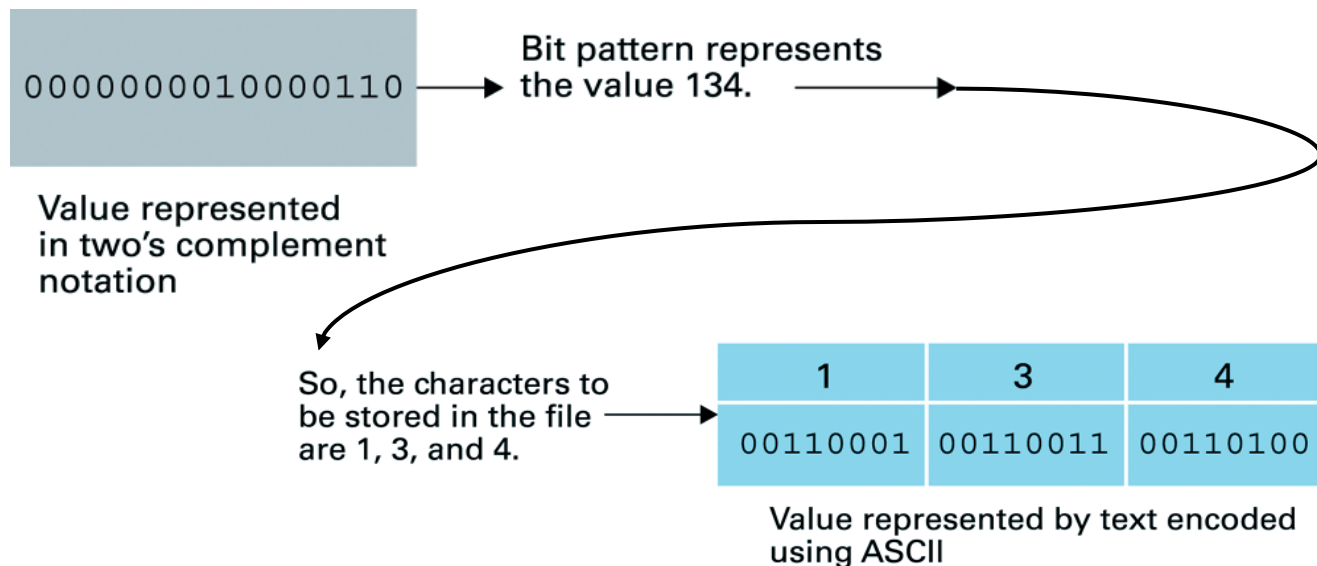
<b>Onderwerpen:</b></b><br><br>
<ul>
<li>architectuur van de computer
<li>werking van de computer
<li>besturingssystemen en computer netwerken
<li>algoritmisch ontwerp
<li>principes van programmeertalen
<li>software engineering
<li>data structuren
<li>bestandsstructuren
<li>database structuren
<li>kunstmatige intelligentie
<li>complexiteitstheorie
</li>
</ul>
```

# 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:

**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

loaded into main  
memory when opened

**Index**

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

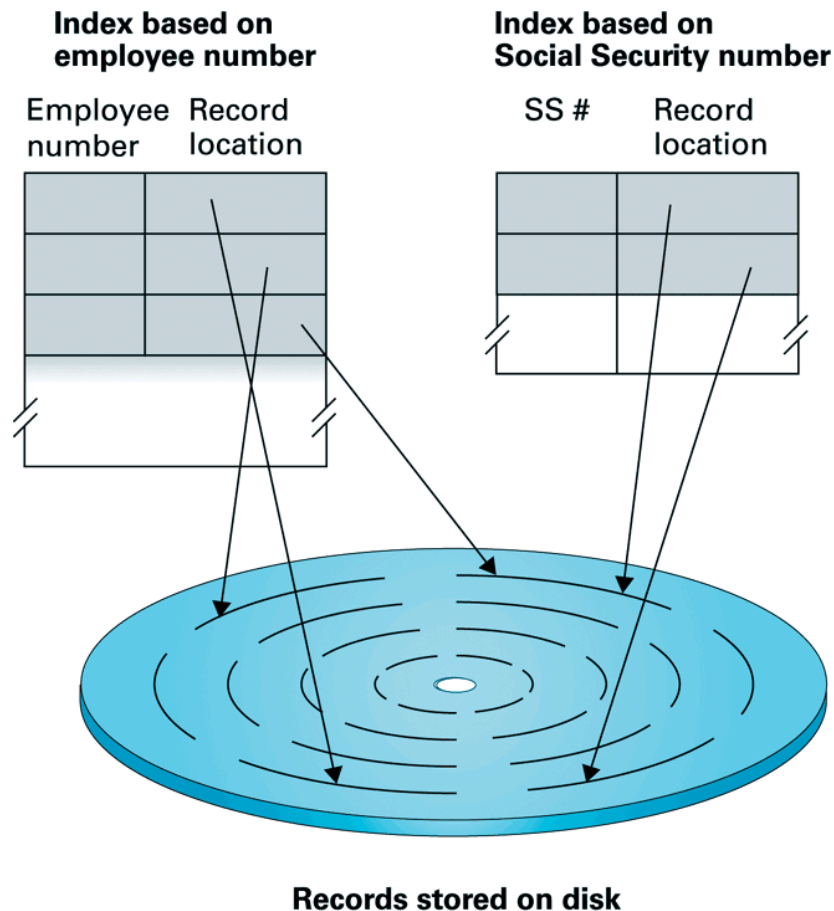
# Chapter 8 - Problem 10

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

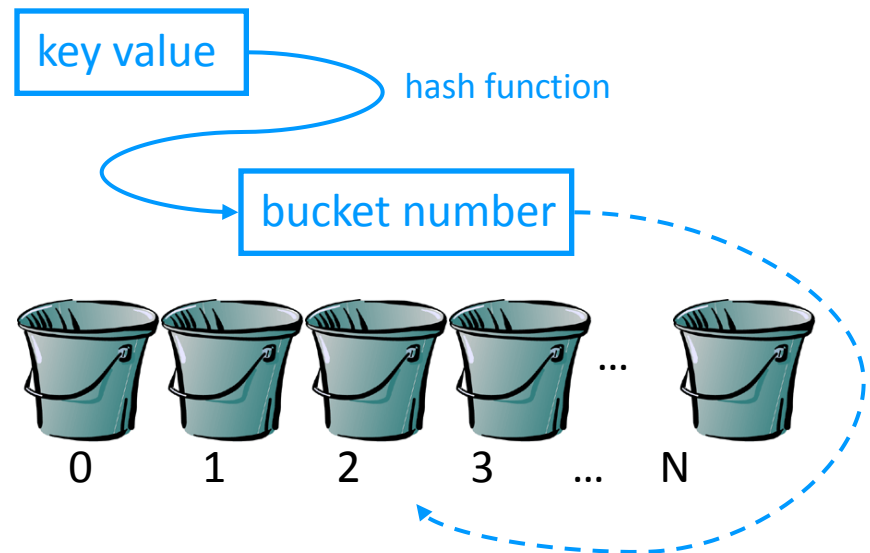


# 8.4: Hashing

- 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?

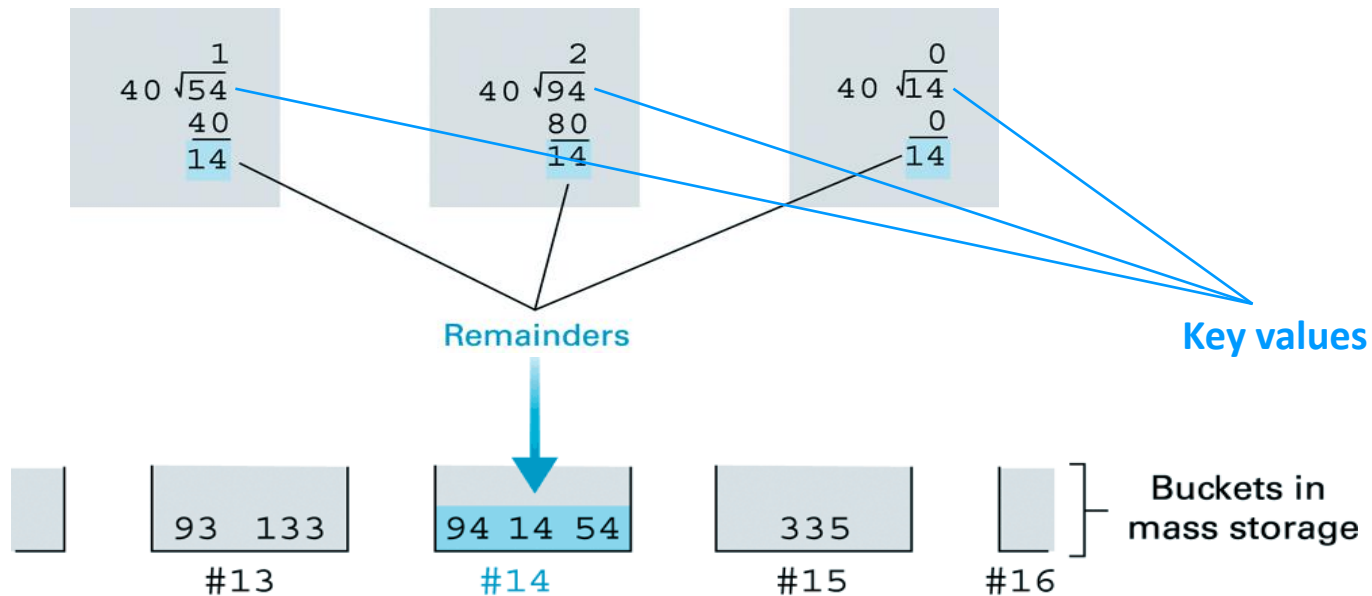
- define set of '*buckets*' & '*hash function*' that converts keys to bucket numbers





## 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!*)

