Foundations of Information Processing

Data structures

Considerations about data structures

Consideration 1:

What is needed

to interpret

data and information?

Consideration 2:

How should
complex and many-sided data
be stored?

Data is
the regular representation of something
in a communicable or processable form.

Information is
processed data.

Knowledge is
a human interpretation of data,
or more typically information.

Data structures: concepts and implementation

- An algorithm needs suitable structured representations of data, depending on a problem to be solved, especially for the input and the output.
- Data types:
  - Integer (int), float (real), character (char), string (str), Boolean.
- Data structures.
  - Aggregates (records).
  - Arrays.
  - Lists, stacks and queues.
  - Trees: binary trees.
  - Graphs.

Source (partly/modified from): J. Boberg, Johdatus tietojenkäsittelytieteeseen, Turun yliopisto, 2010 (in Finnish), and Brookshear, J.G., Brylow, D., Computer Science - An Overview, 13th edition, Addison Wesley, 2003/2019 (figures).

Data types: numbers

Data type	**Size in bits**	**Size in bytes**	**Range**
Unsigned integer	8	1	[0, 255]
	16	2	[0, 65535]
long	32	4	[0, 4294967295]
	64	8	[0, 2^{64}-1]
Signed integer	8	1	[-128, +127]
	16	2	[-32768, +32767]
long	32	4	[-2147483648, +2147483647]
	64	8	[-2^{32}, +2^{32}-1]
Float (real)	32 (1+23+8)	4	[10^{-38}, 10^{+38}]
double	64 (1+52+11)	8	[10^{-308}, 10^{+308}]
	128 (1+112+15)	16	[10^{-4931}, 10^{+4931}]

the sign bit + the mantissa + the exponent (one bit for the sign)
32 bits => 7 bits for the exponent $\Rightarrow 2^7 = 128 \Rightarrow 2^{128} \Rightarrow 10^{38}$

Data types: Boolean and character

Data type	Size in bits	Size in bytes	Range
Boolean	1 (8)	(1)	[0, 1]
Character	8	1	ASCII ISO 8859-1, 2, ... UTF-8
	16	2	UTF-16 UCS-2
	32	4	UTF-32 UCS-4
String			A sequence of characters

In theory, the Boolean data type needs one bit only (value 0 or 1).
In practice, data is stored in bytes.
Note: different implementations in programming languages.

Data structures

- Data structures are abstract ways of storing encoded data and information in a structured form.
  - For example, how to present the players of an ice-hockey team and their information?
- The data structure is selected on an application-specific basis.
  - What data structures does the application support?
  - For example, RTF, HTML, XML, LaTeX, LISP, Prolog.
- Structured storing of data suitable to an algorithm makes problem solving more efficient.
  - The algorithm is suitable to the data structure.
  - The data structure is suitable to the algorithm.
- Next, let us see what data structures are available.

Aggregates (records)

- A data structure which consists of fields of different types and sizes, each field with the same structure = the aggregate type.
- The structure is defined in the pseudo language as follows:

record_name = RECORD
  field_name: *data_type*
  …
ENDRECORD

personaldata = RECORD
  givenname: STRING
  surname: STRING
  address: STRING
  phone: INTEGER
ENDRECORD

- The use of the structure:

record_name variable_name
variable_name.field_name

personaldata person
person.surname := ”Kälviäinen"

Arrays: vectors and matrices

- A data structure which consists of elements of the same data type.
- Dimensions:
    - Vectors (1-dimensional).
    - Matrices (2-dimensional).
- The definitions of the data structure in the pseudo language:

  datatype variable_name[*number_of_elements*]

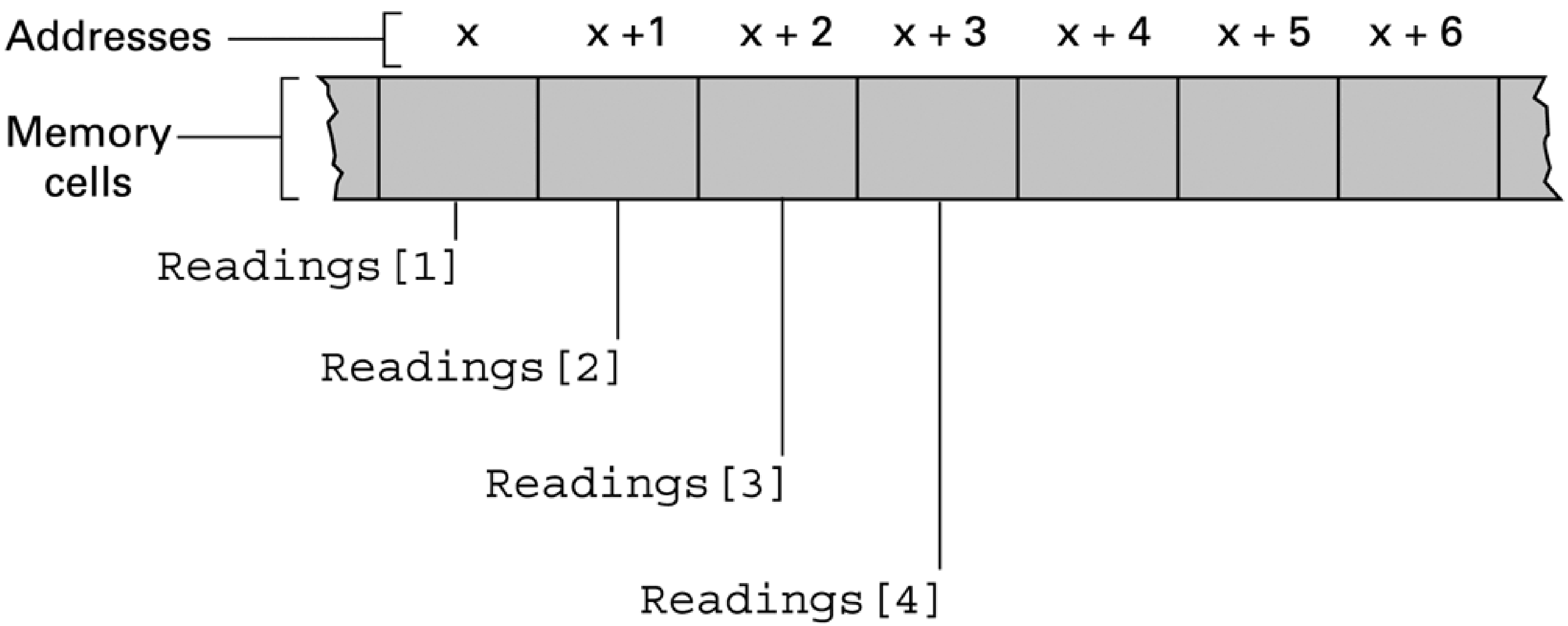
  INTEGER A[10]
  REAL B[3][3]

- The use of the data structure:
  variable_name[*index_of_element*]

  A[1] := 1
  B[2][2] := 3.1415926536

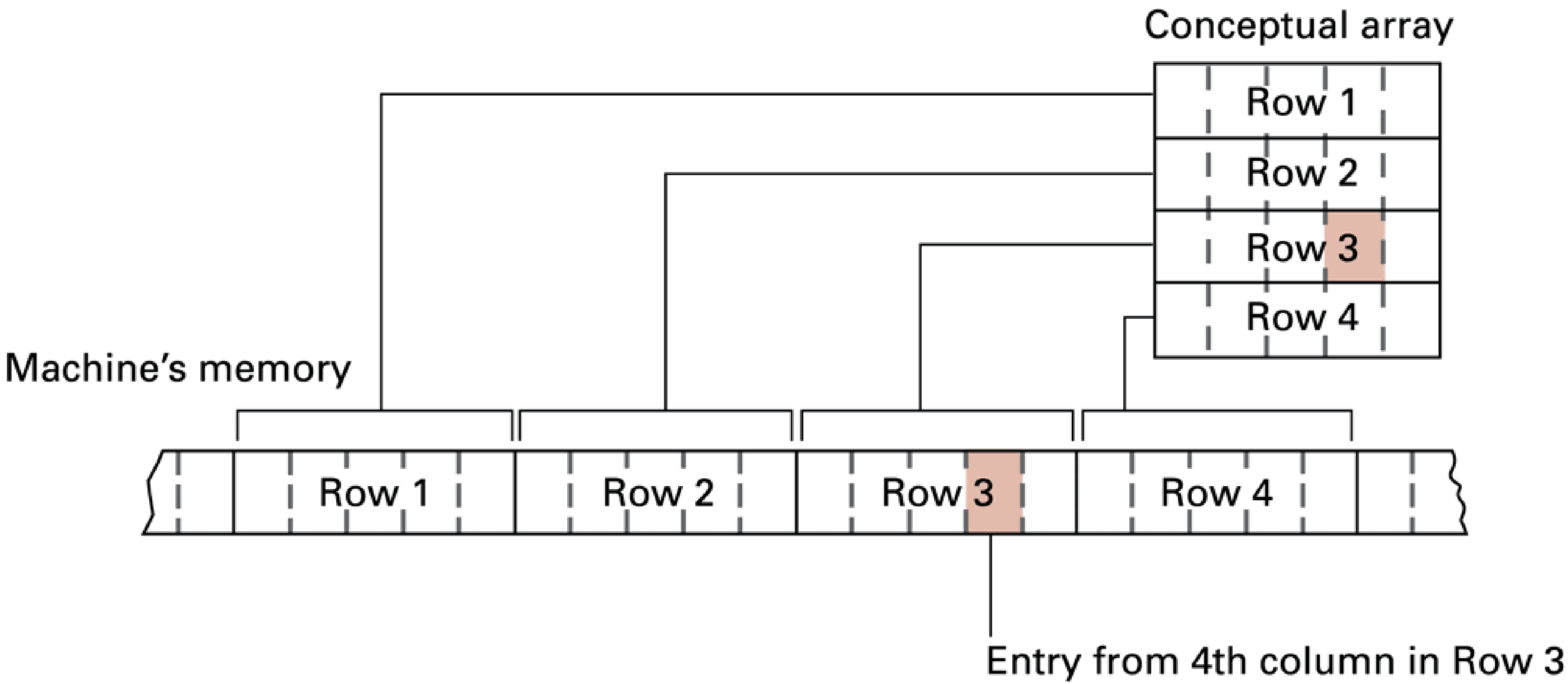
One-dimensional arrays in memory

- Memory cells and their addresses in case of a one-dimensional array (a vector).
- “Readings” is the name of a variable where measurements are stored.

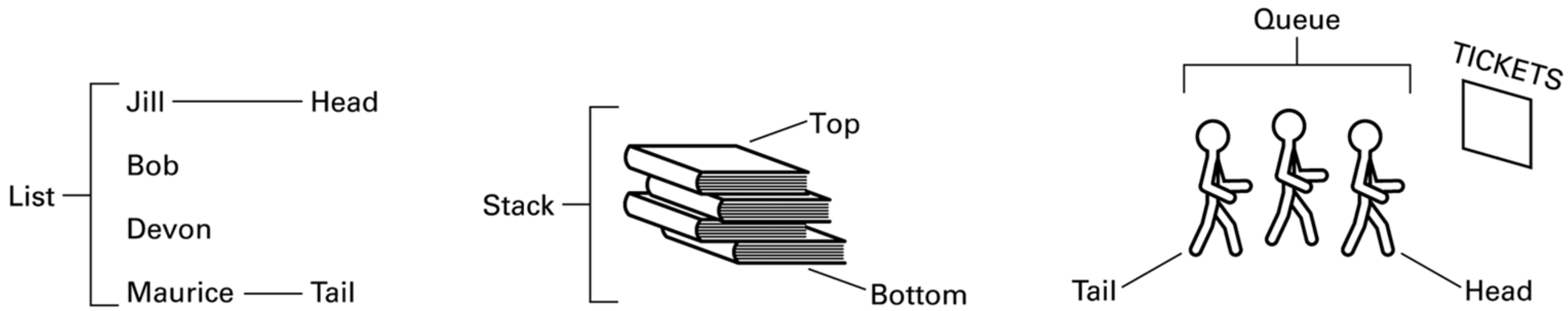


Two-dimensional arrays in memory

- Rows and columns in case of a two-dimensional array (a matrix).
- Entries of each row represent the corresponding columns.



Lists, stacks, queues



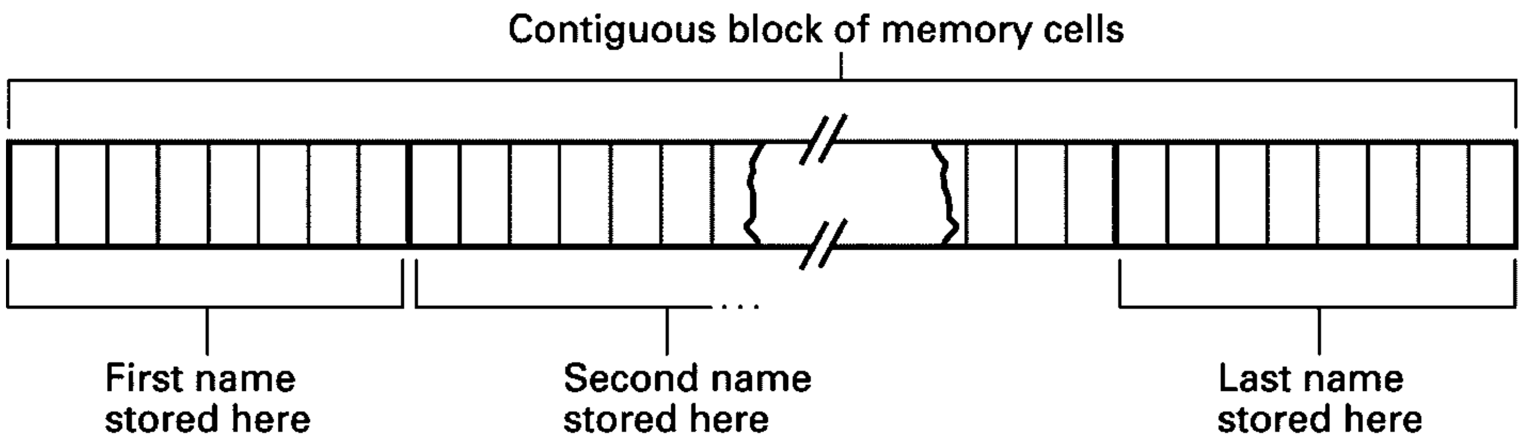
a. A list of names **b.** A stack of books **c.** A queue of people

Definitions:

- **List:** a collection whose entries (data elements) are arranged sequentially.
- **Head:** the beginning of a list.
- **Tail:** the end of a list.
- **Stack**: entries are inserted and removed only at the head.
- **Queue**: entries are removed only at the head and new entries are inserted only at the tail.

Storing lists: contiguous and linked

- **A contiguous list:** the entire list is stored in one large block of memory, with successive entries following each other in contiguous memory cells.

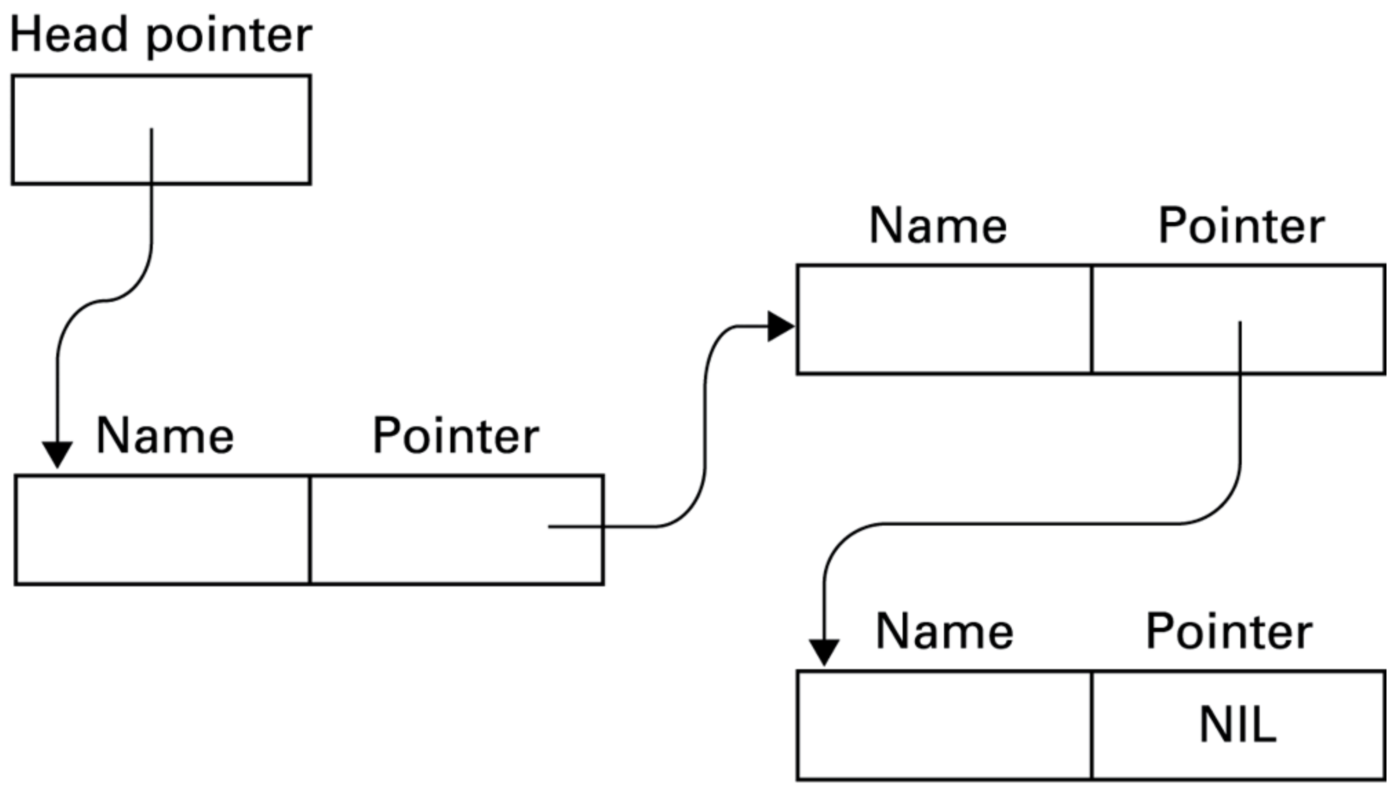


- How about the deletion/insertion? Time-consuming shuffling of entries?

⇒ **Linked lists:** a list where memory cells are linked by pointers:

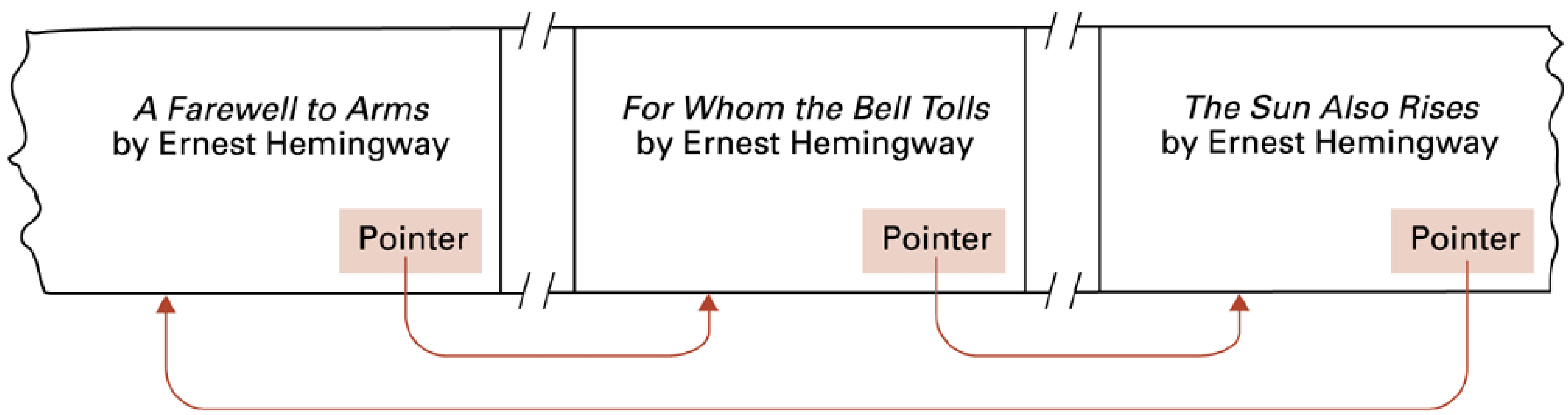
  - **Head pointer:** the pointer points to the beginning (head) of the list.
  - **Null pointer:** the pointer marks the end of the list.

The structure of a linked list

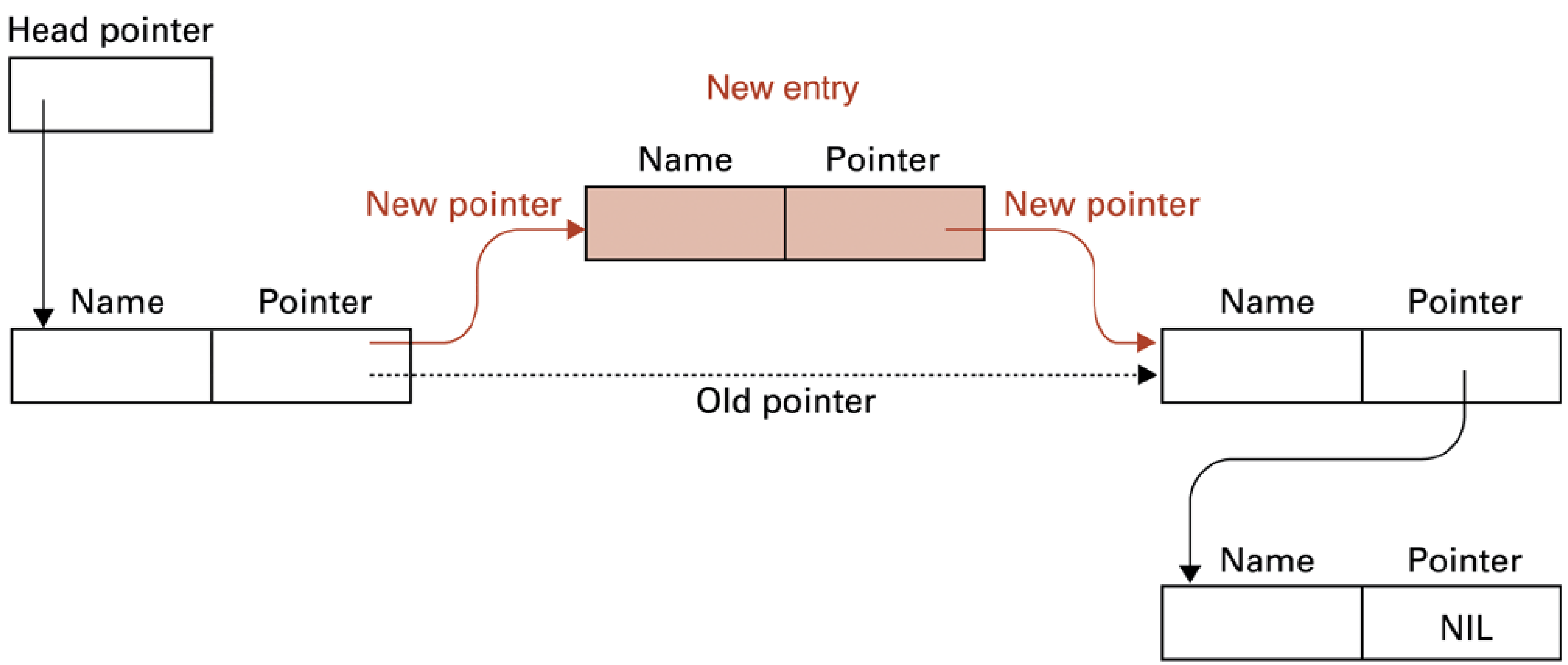


- Which one is better? The contiguous list or the linked list?
- How about memory allocation?
- Memory for the pointers?
- A need of changing memory cells/fields when deleting/inserting?

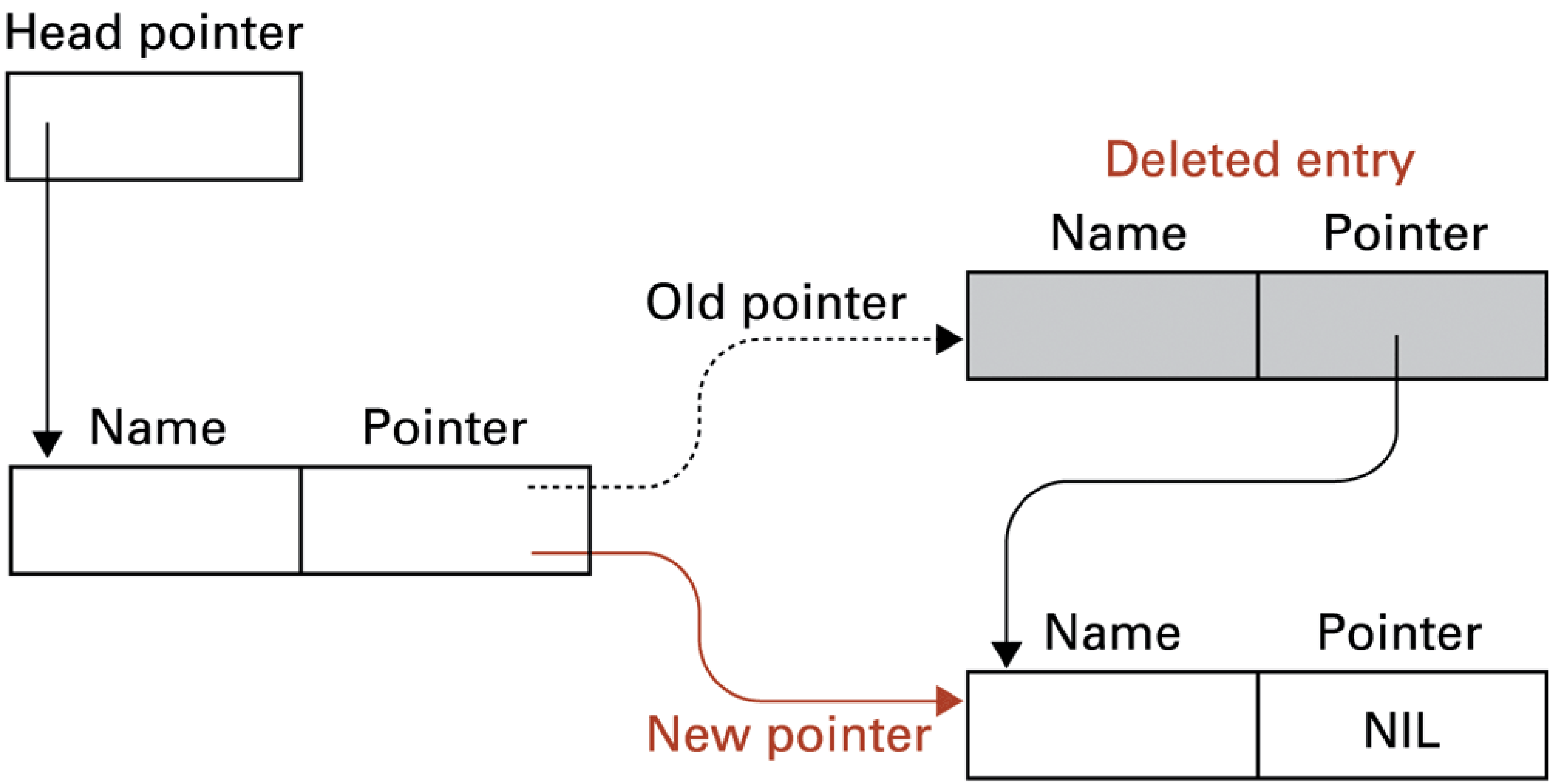
Example: books linked by the author's name



Inserting an entry into a linked list

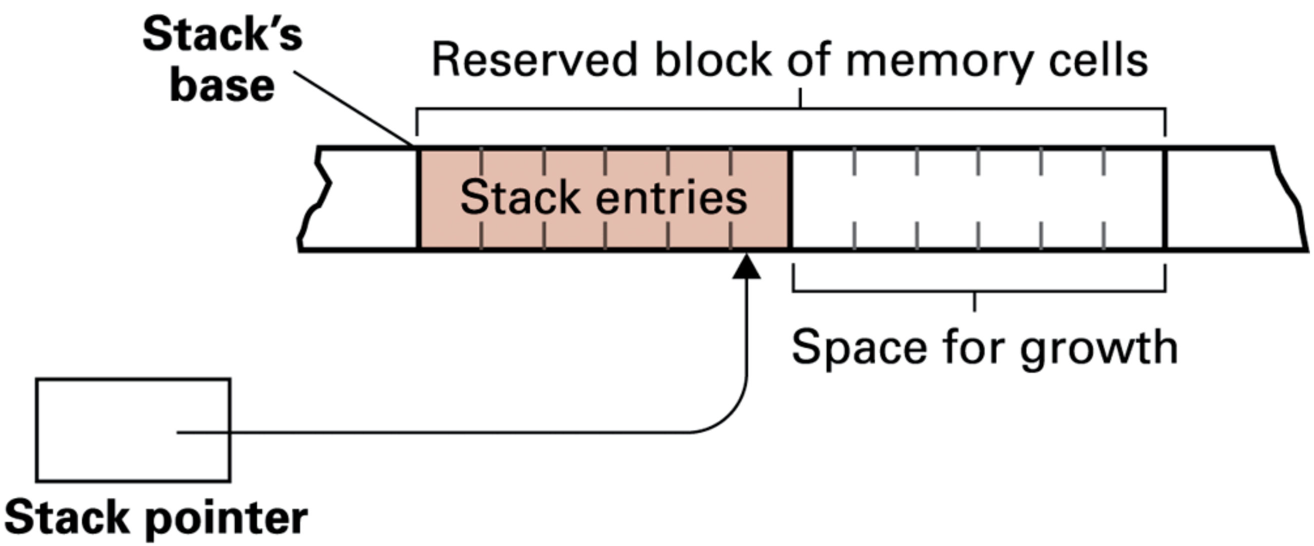


Deleting an entry from a linked list



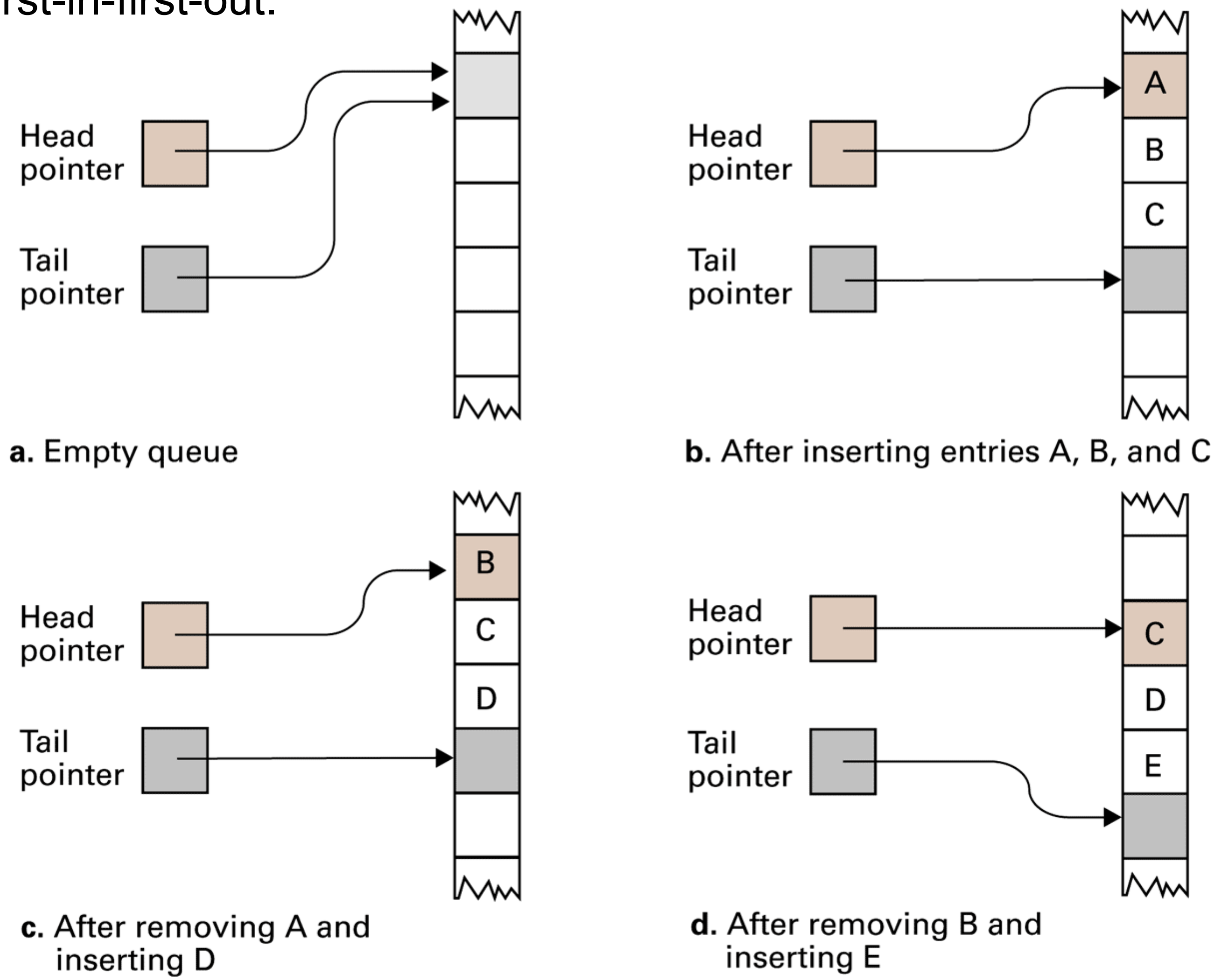
Stacks in memory

- Entries are inserted and removed only at the head.
- LIFO: Last-in-first-out.
- Stack's base: the location where the first entry is pushed onto the stack (memory is usually allocated in a contiguous manner).
- **Stack pointer:** a pointer to the top of a stack.
- **Deletion:** removes an entry from the top of a stack.
- **Insertion:** inserts an entry onto the top of a stack.



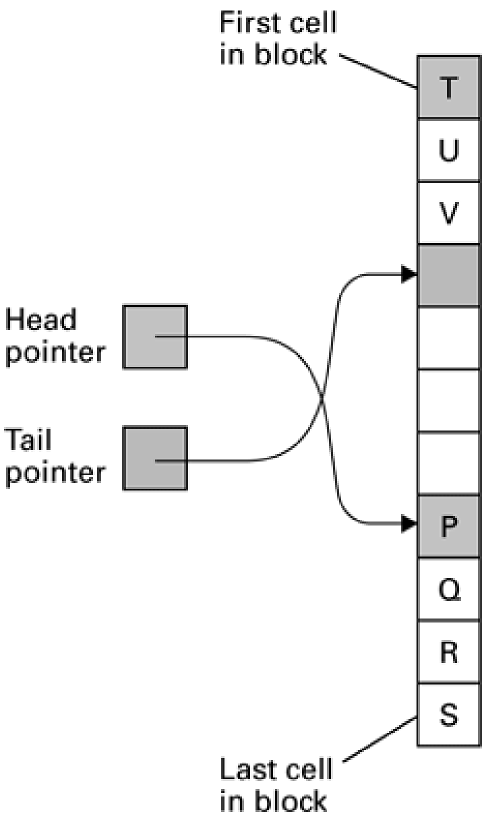
Queues in memory

- Entries are removed only at the head (the head pointer) and new entries are inserted only at the tail (the tail pointer).
- FIFO: First-in-first-out.

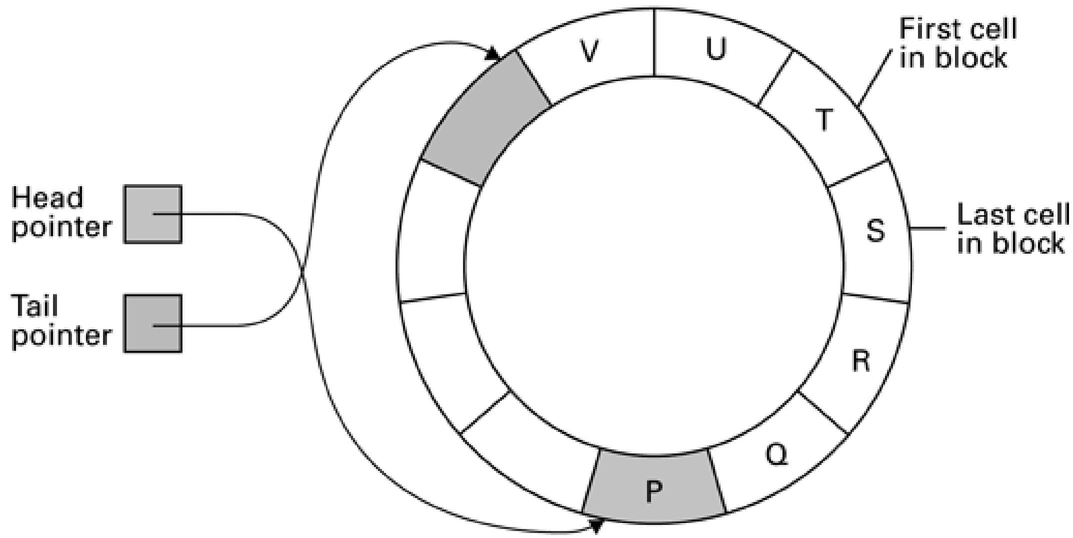


Circular queue: conceptual storage

In the previous page's example, the queue crawls through memory as entries are inserted and removed. => Confine the queue to its reserved block of memory => a circular queue.

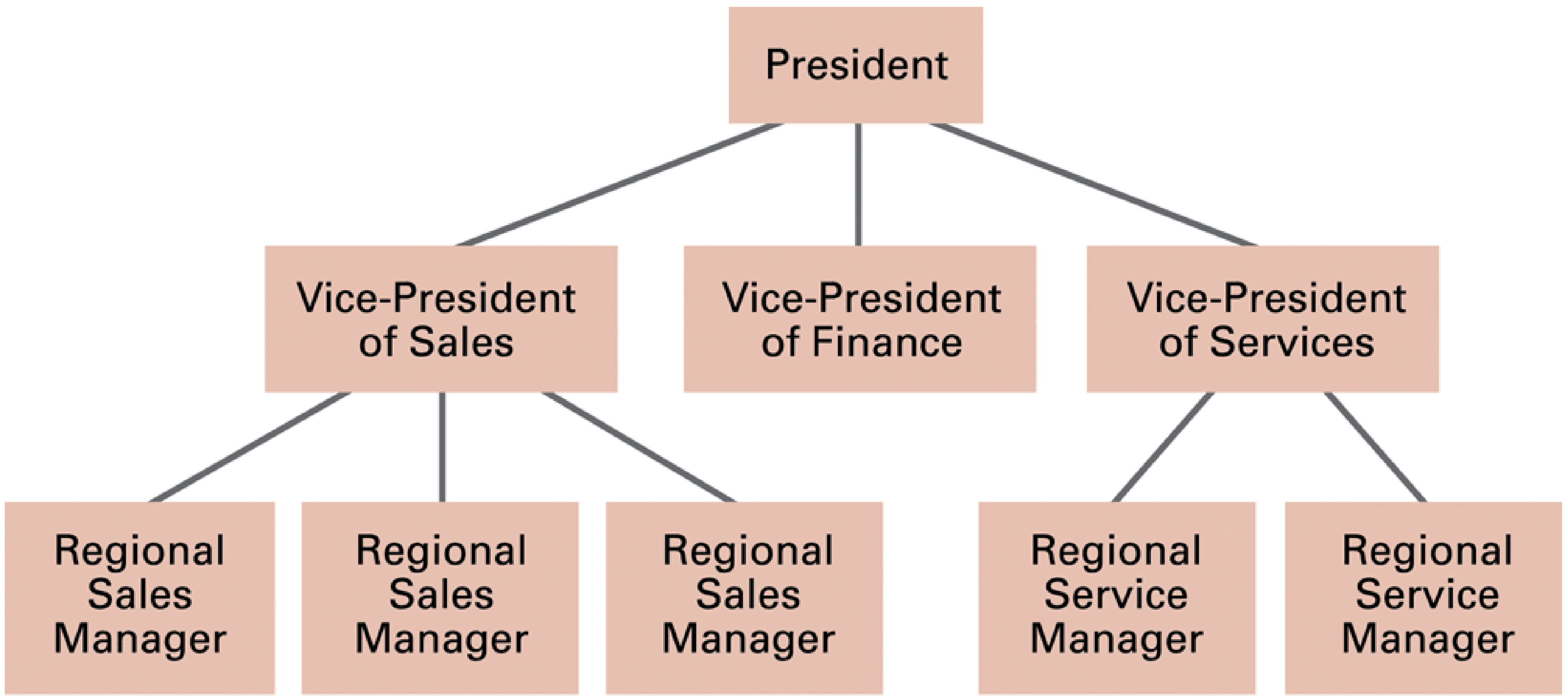


a. Queue as actually stored



b. Conceptual storage with last cell "adjacent" to first cell

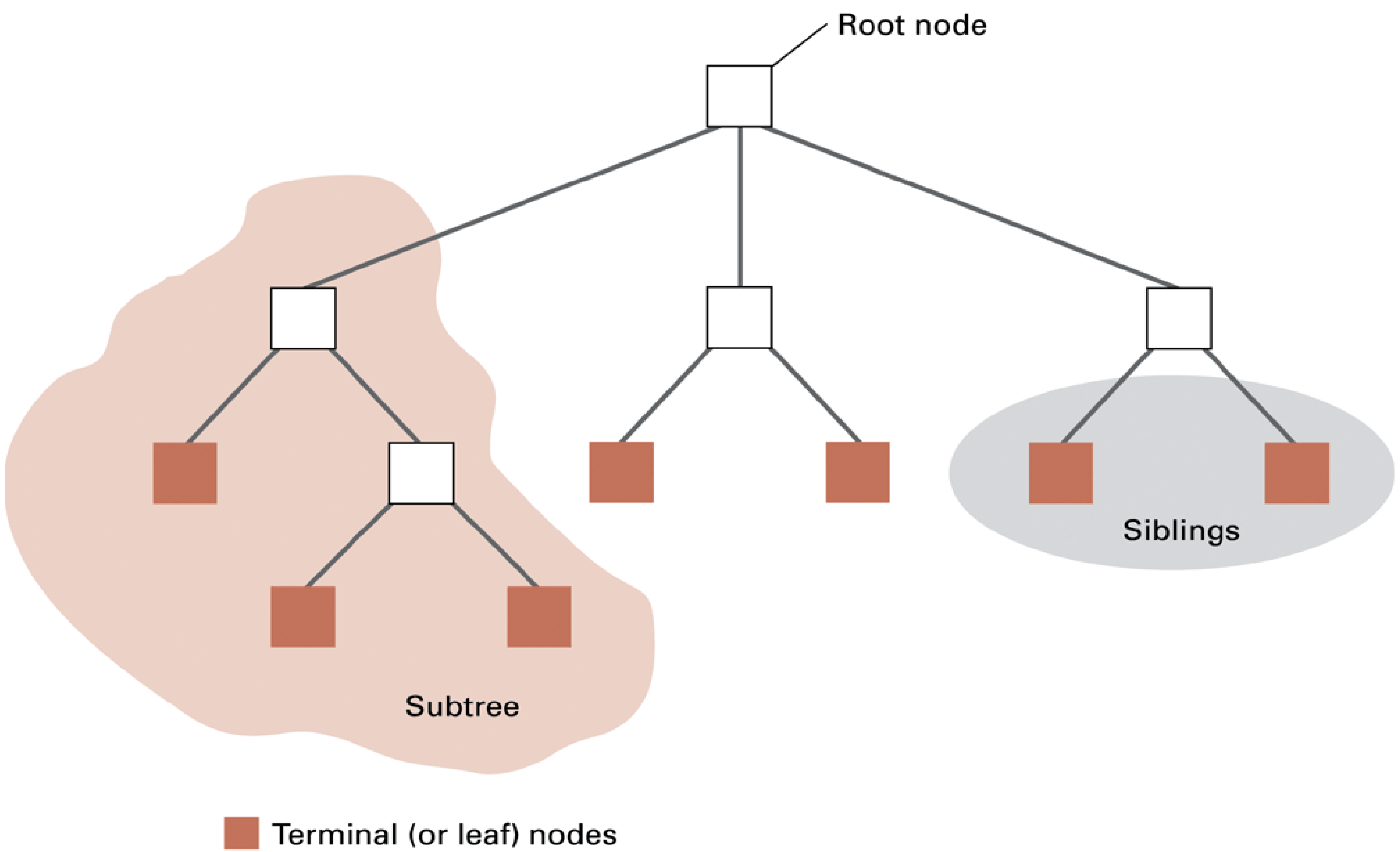
An organization chart: an example of a tree



- Hierarchy is needed. Items must have an order how to present.
- In a company maybe quite clear.

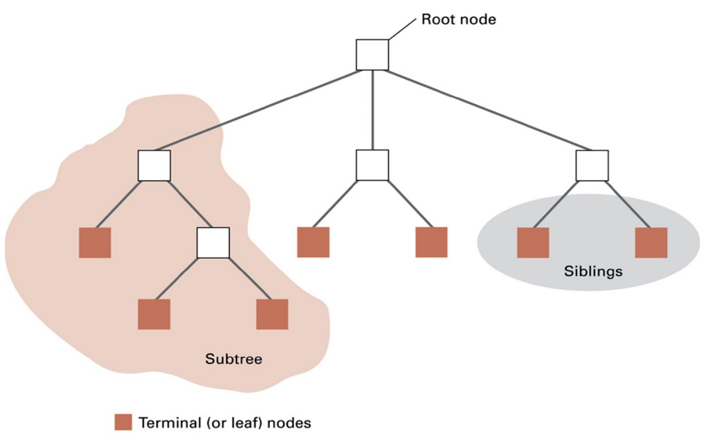
A tree as a hierarchical data structure

- The root node, the subtrees, the siblings, the leaf nodes.



Concepts of the tree structure

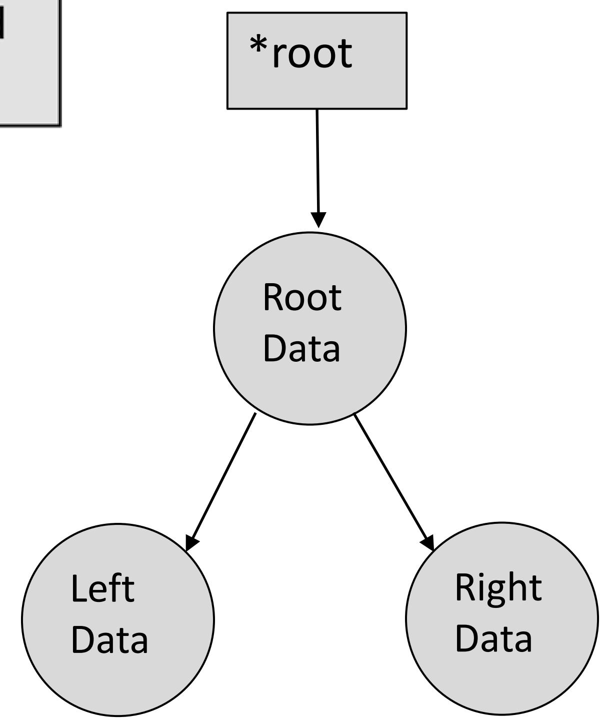
- **Tree**: a collection of hierarchically organized information.
- **Node**: a data element in a tree.
- **Root node**: the highest node.
- **Leaf node**: the lowest node in a branch.
- **Inner node**: a node which contains the parent and at least one child.
- **Parent:** a node immediately above the node.
- **Child**: a node immediately below the node.
- **Grandparent**: parents, grandparents, etc.
- **Descendant**: children, grandchildren, etc.
- **Siblings**: nodes having the same parent.
- **Subtree**: a tree below a node.
- **Binary tree:** a tree with two children at maximum.
- **Depth:** the number of nodes in the longest path from the root node to a leaf node. In the figure the depth is 4 (the root => the leaf, 4 nodes).



The structure of a node in a binary tree

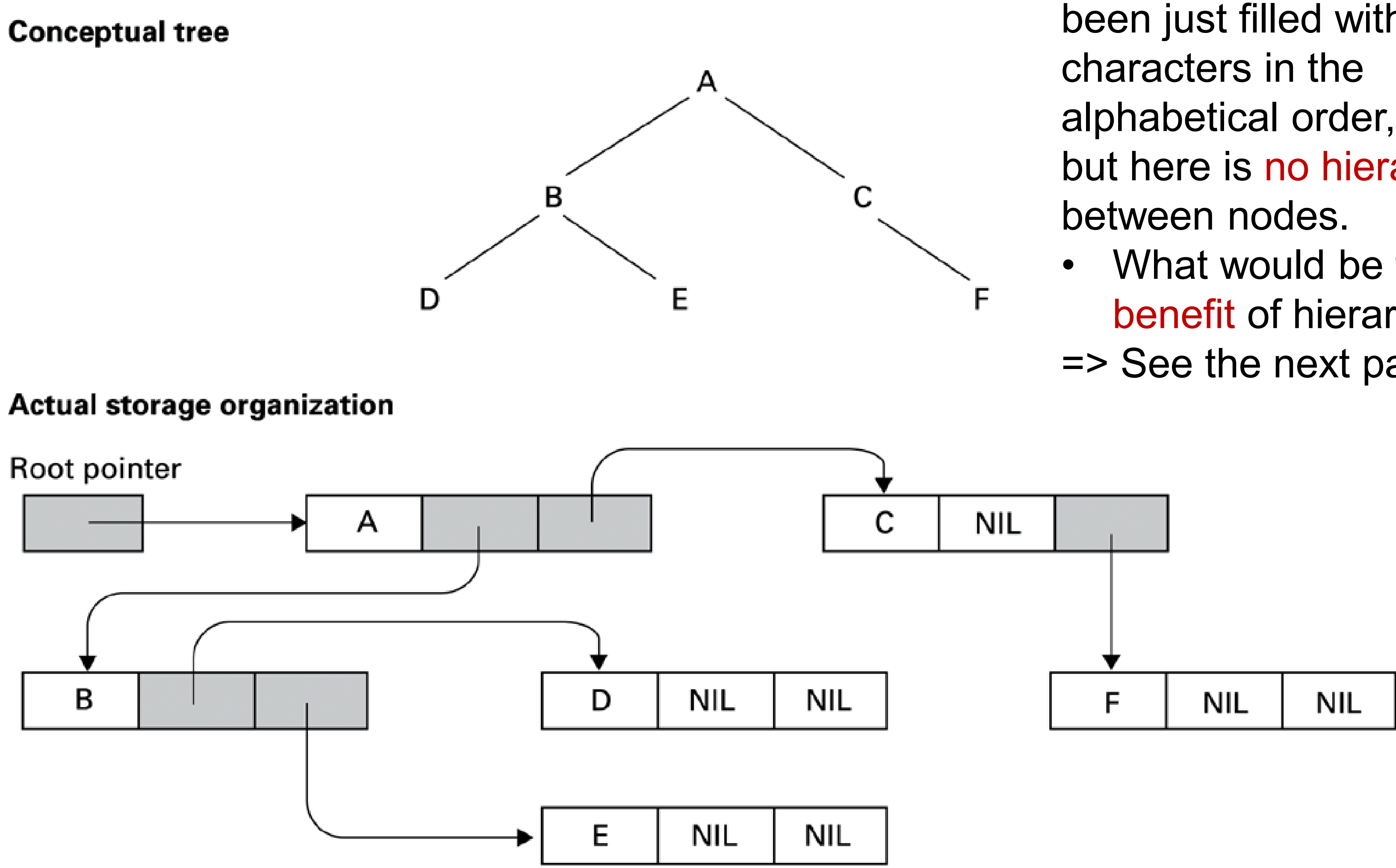
- The data and the pointers:

Cells containing the data	Left child pointer	Right child pointer



- A linked structure:
  - The pointer to the root node.
  - Node: the data + the pointers to the children.
- Defined as arrays:
  - A[1] = the root node.
  - A[2],A[3] = the children of A[1].
  - A[4],A[5],A[6],A[7] = the children of A[2]:n ja A[3]:n.

A binary tree using a linked storage



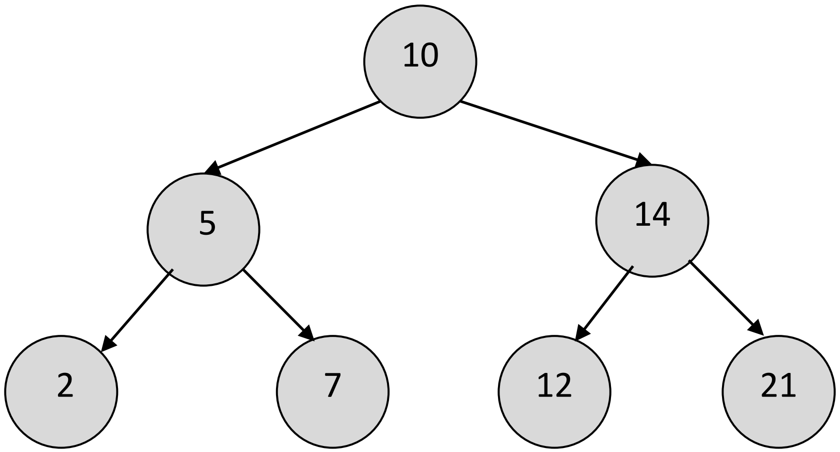
Note: that the tree has been just filled with the characters in the alphabetical order, but here is no hierarchy between nodes.

- What would be the benefit of hierarchy?

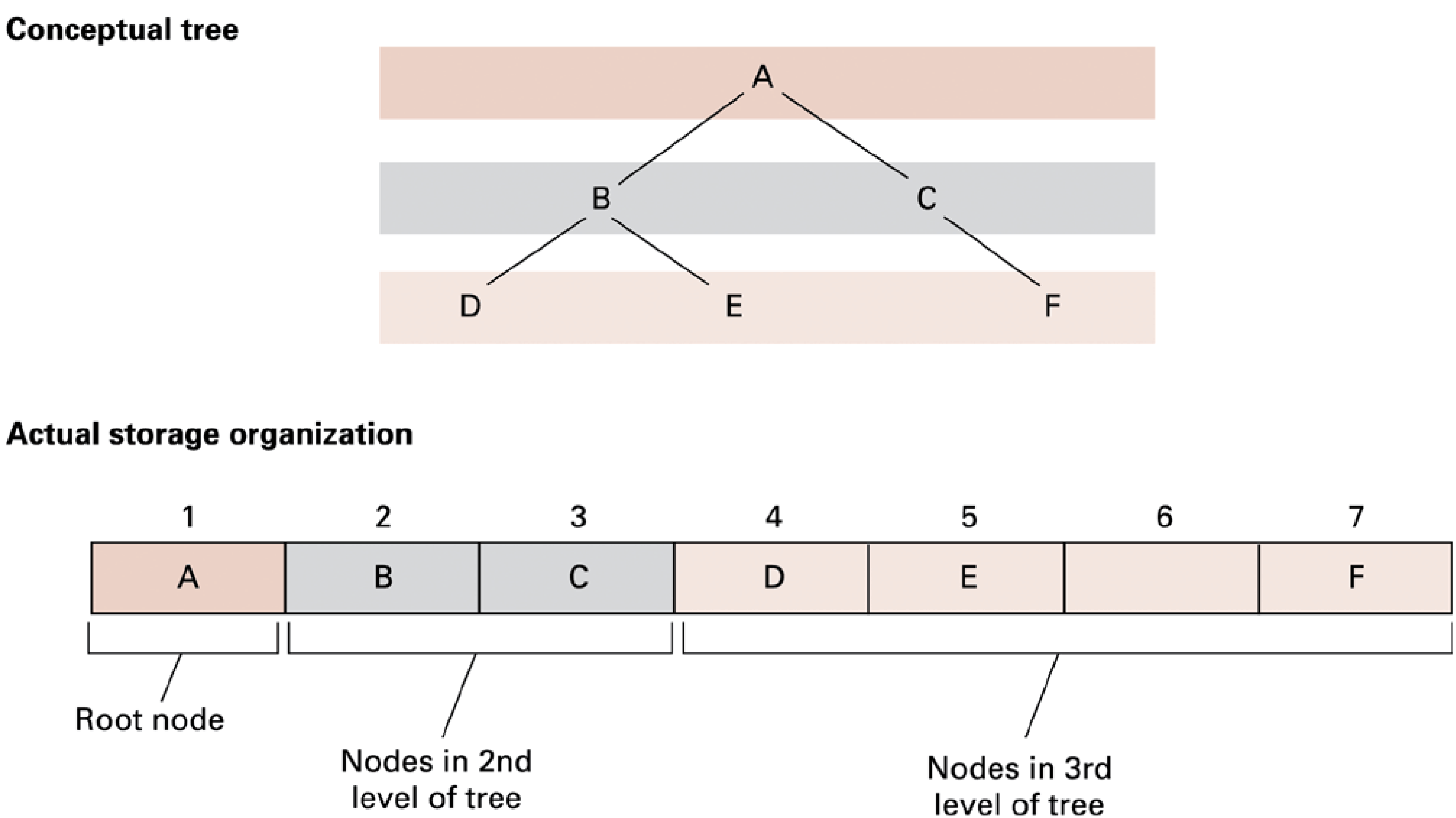
=> See the next page.

Balanced and ordered binary trees

- A tree becomes a robust data structure if the elements of a tree are ordered so that a tree is balanced and ordered.
- Balanced: the heights of leaf nodes differ by one at maximum.
- Ordered: the alphabetical or numeric order can be applied.
  - Example: place smaller numbers in the left and larger numbers in the right in each node, keeping the tree balanced at the same time.
- Thus, the search of an element in the tree is efficient since the search is logarithmic $O(\log_2 n)$, instead of linear $O(n)$ (as in the list).
  - See the lecture notes about the complexity of algorithms for details.

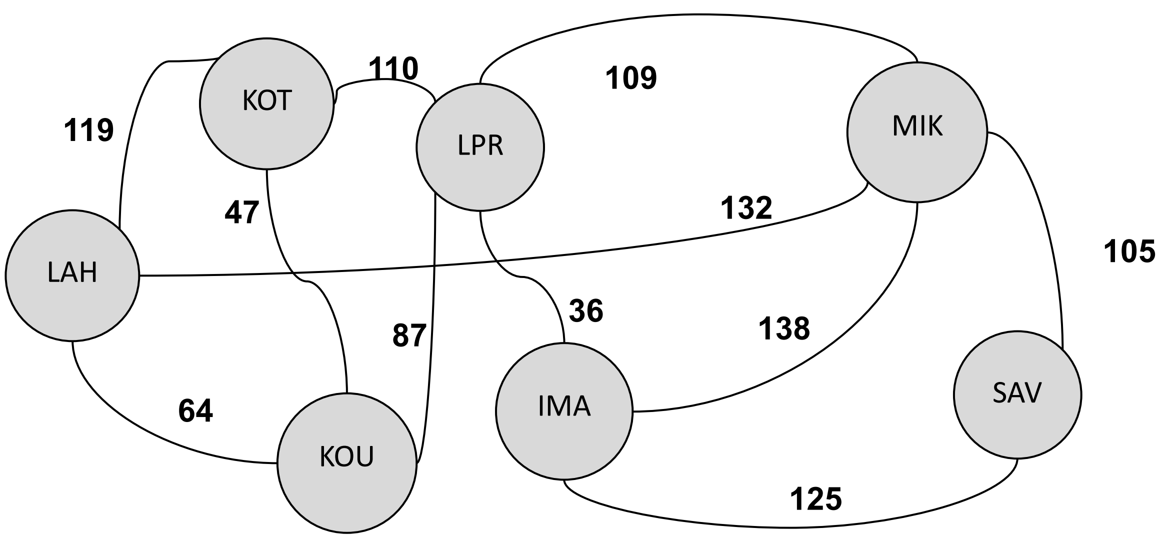


A tree stored without pointers



Graphs: no limitations to linking nodes

- The tree structure is hierarchical, starting from the root node:
  - Nodes contain *child nodes* which contain *child nodes*, etc.
- How about if the nodes could be connected without limitations?
- This kind of a structure is called as a graph.
  - A tree is the special case of a graph (the subset of a graph).
- Example: travelling salesperson
  - How to visit each town once so that
    the total length of the journey is minimized?



Summary

- The data type is needed
  when exploring data and information.

- The data structures are abstract ways
  to store the encoded data in memory
  in a structured form.

- A structured way to store
  which is suitable for the purpose
  makes the use of data more efficient.