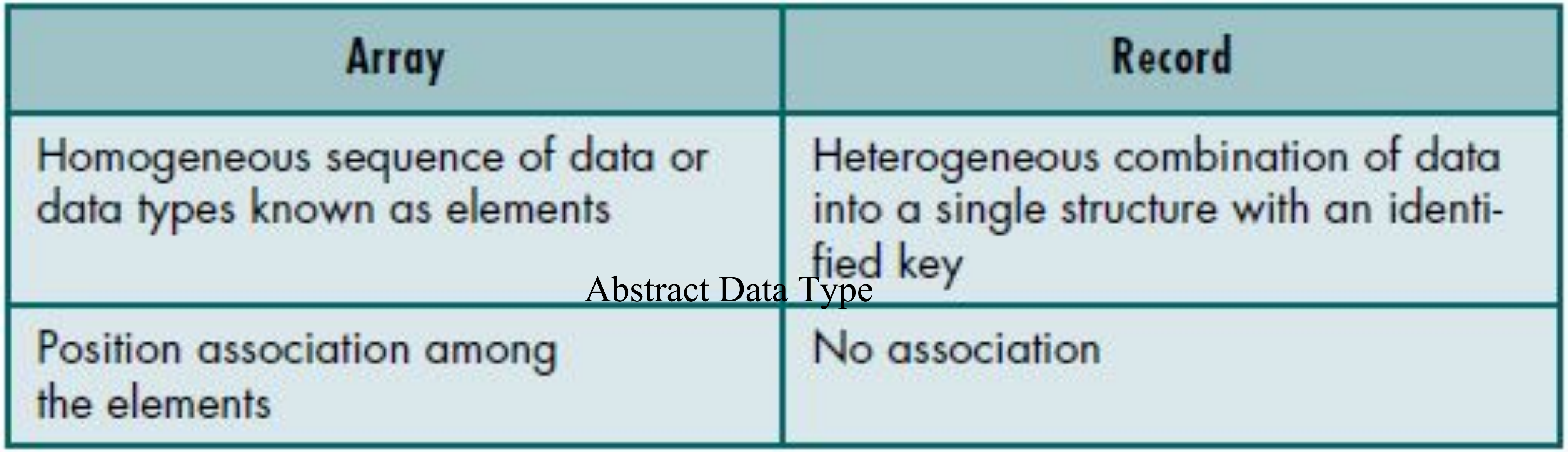
Linked Lists

Data Structure:

- A data structure is an aggregation of atomic and composite data into a set with defined relationships.
- In this definition structure means a set of rules that holds the data together.
- In other words, if we take a combination of data and fit them into a structure such that we can define its relating rules, we have made a data structure.
- Data structures can be nested. We can have a data structure that consists of other data structures.

Data Structure Example:



Array	Record
Homogeneous sequence of data or data types known as elements	Heterogeneous combination of data into a single structure with an identified key
Position association among the elements	No association

Abstract Data Type

Abstract Data Type (ADT)

ADT consists of a set of definitions that allow programmers to use the functions while hiding the implementation.

This generalization of operations with unspecified implementations is known as abstraction. We abstract the essence of the process and leave the implementation details hidden.

The concept of abstraction means:

1. We know *what* a data type can do.
2. *How* it is done is hidden.

ADT Concept:

- The concept of abstraction means:
  1. We know *what* a data type can do.
  2. *How* it is done is hidden.

List as ADT

Consider the concept of a list.

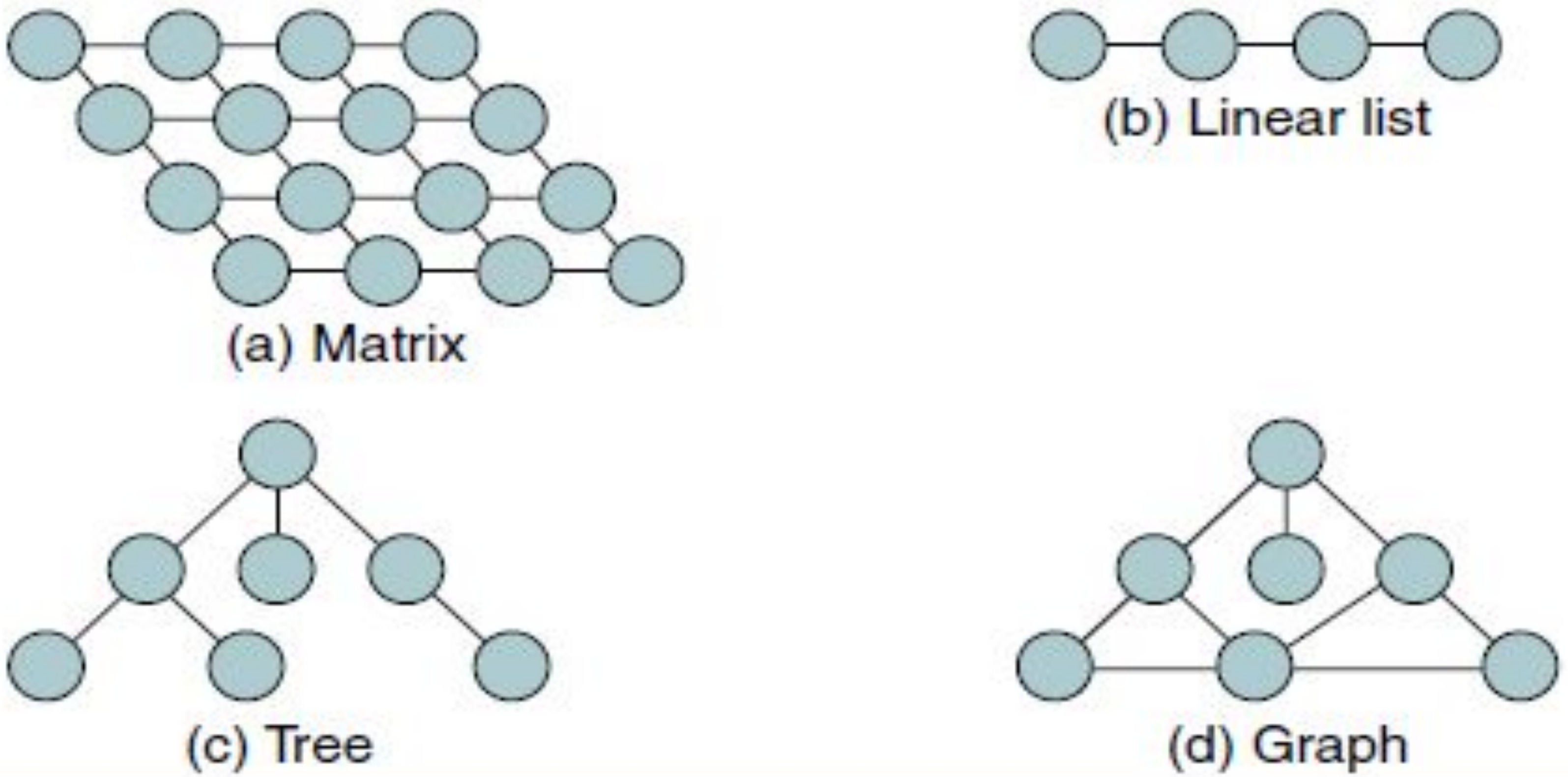
- At least four data structures can support a list.

We can use a

1. Matrix.
2. Linear list.
3. tree.
4. graph.

- If we place our list in an ADT, users should not be aware of the structure we use. As long as they can insert and retrieve data, it should make no difference how we store the data.

Examples of Data Structures:



Some Data Structures

Example of Data Structure:

- Consider the system analyst who needs to simulate the waiting line of a bank to determine how many tellers are needed to serve customers efficiently.

- This analysis requires the simulation of a queue. However,queues are not generally available in programming languages. Even if a queue type were available, our analyst would still need some basic queue

- operations, such as *enqueuing (insertion) and dequeuing (deleting),* for the simulation.

There are two potential solutions to this problem:

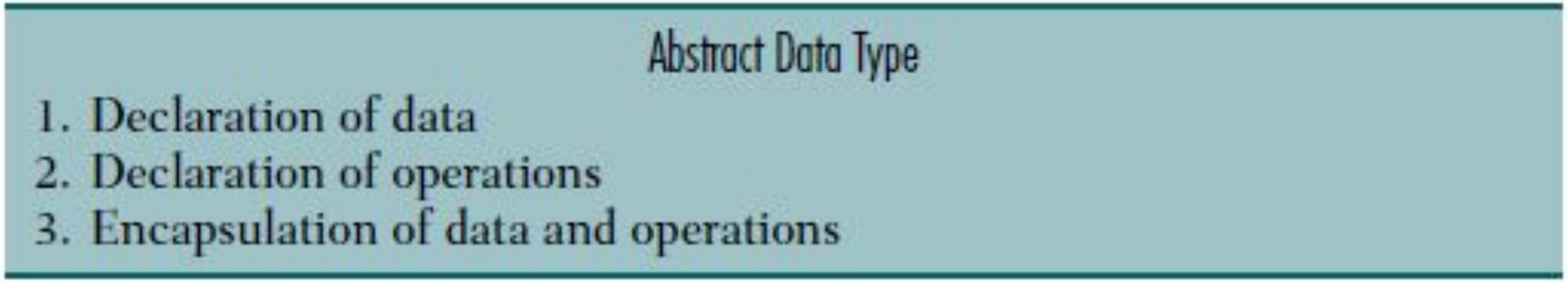
1) Write a program that simulates the queue our analyst needs (in this case, our solution is good only for the one application at hand)

2) Write a queue ADT that can be used to solve any queue problem.

- If we choose the latter solution, our analyst still needs to write a program to simulate the banking application, but doing so is much easier and faster because he or she can concentrate on the application rather than the queue.

How to Define ADT?

- An abstract data type is a data declaration packaged together with the operations that are meaningful for the data type.
- In other words, we encapsulate the data and the operations on the data, and then we hide them from the user.



Abstract Data Type

1. Declaration of data
2. Declaration of operations
3. Encapsulation of data and operations

General Linear Lists:

- A general linear list is a list in which operations, such as retrievals, insertions, changes, and deletions, can be done anywhere in the list, that is, at the beginning, in the middle, or at the end of the list.
- Examples:
  - ✓ lists of employees
  - ✓ student lists
  - ✓ lists of our favorite songs

Basic Operations:

- The four basic list operations are insertion, deletion, retrieval, and traversal.
  - ***Insertion*** is used to add a new element to the list.
  - ***Deletion*** is used to remove an element from the list.
  - ***Retrieval*** is used to get the information related to an element without changing the structure of the list.
  - ***Traversal*** is used to traverse the list while applying a process to each element.

Insertion:

- List insertion can be **ordered** or **random**.
- ***Ordered lists*** are maintained in sequence according to the data or, when available, a key that identifies the data.
- A key is one or more fields within a structure that identifies the data.
- ***Random lists*** there is no sequential relationship between two elements.
- There are **no restrictions** on **inserting** data into a random list, computer algorithms generally insert data at the end of the list. Thus, random lists are sometimes called **chronological lists**.

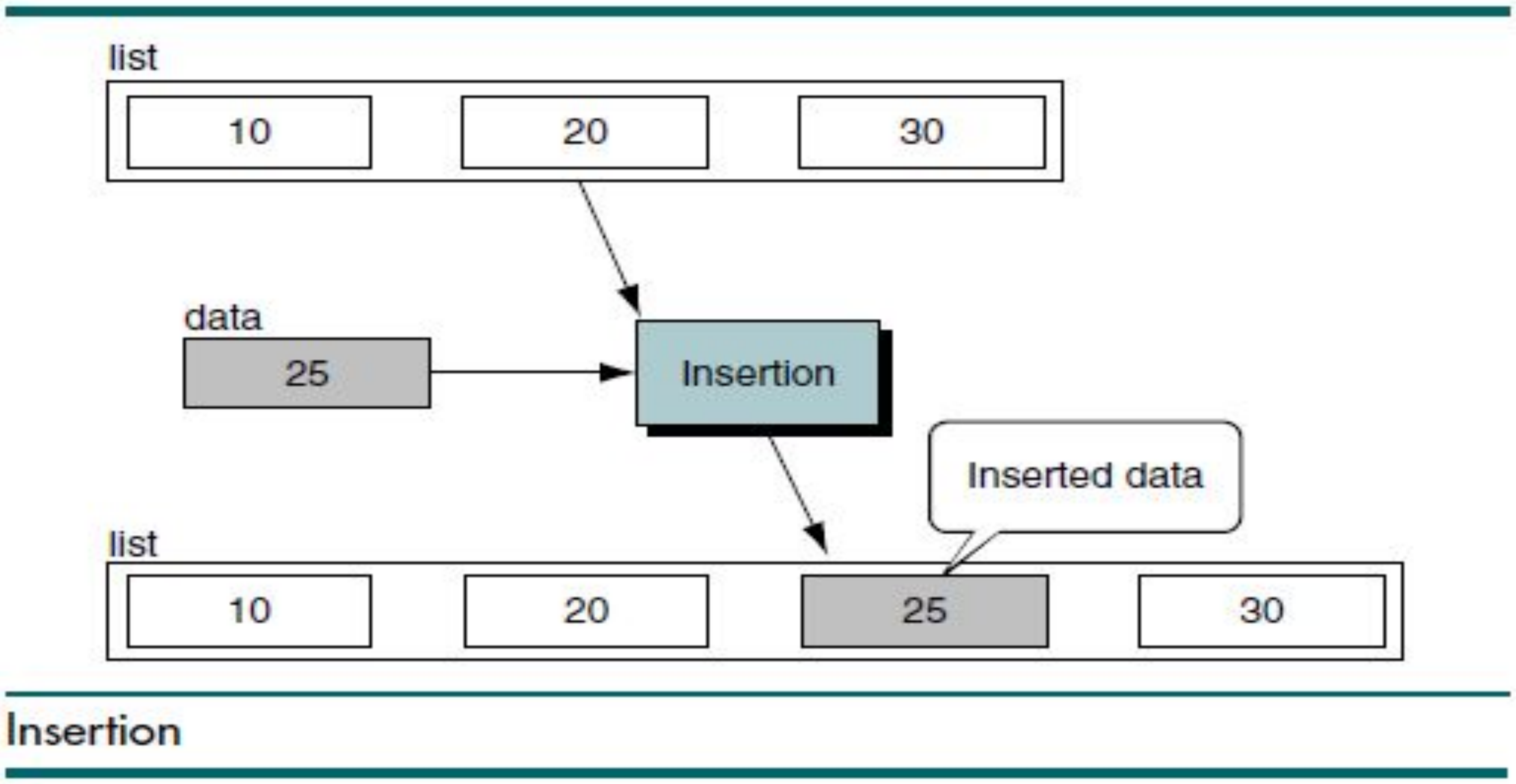
Insertion in Random Lists

- Applications that use random lists are found either in data-gathering applications, in which case the lists are chronological lists, or in situations in which the applications require randomness such as simulation studies or games

Insertion in Ordered Lists

- Data must be inserted into ordered lists so that the ordering of the list is maintained. Maintaining the order may require inserting the data at
  - at the beginning
  - at the end of the list
  - Some times time data are inserted somewhere in the middle of the list.
- To determine where the data are to be placed search algorithms are used.

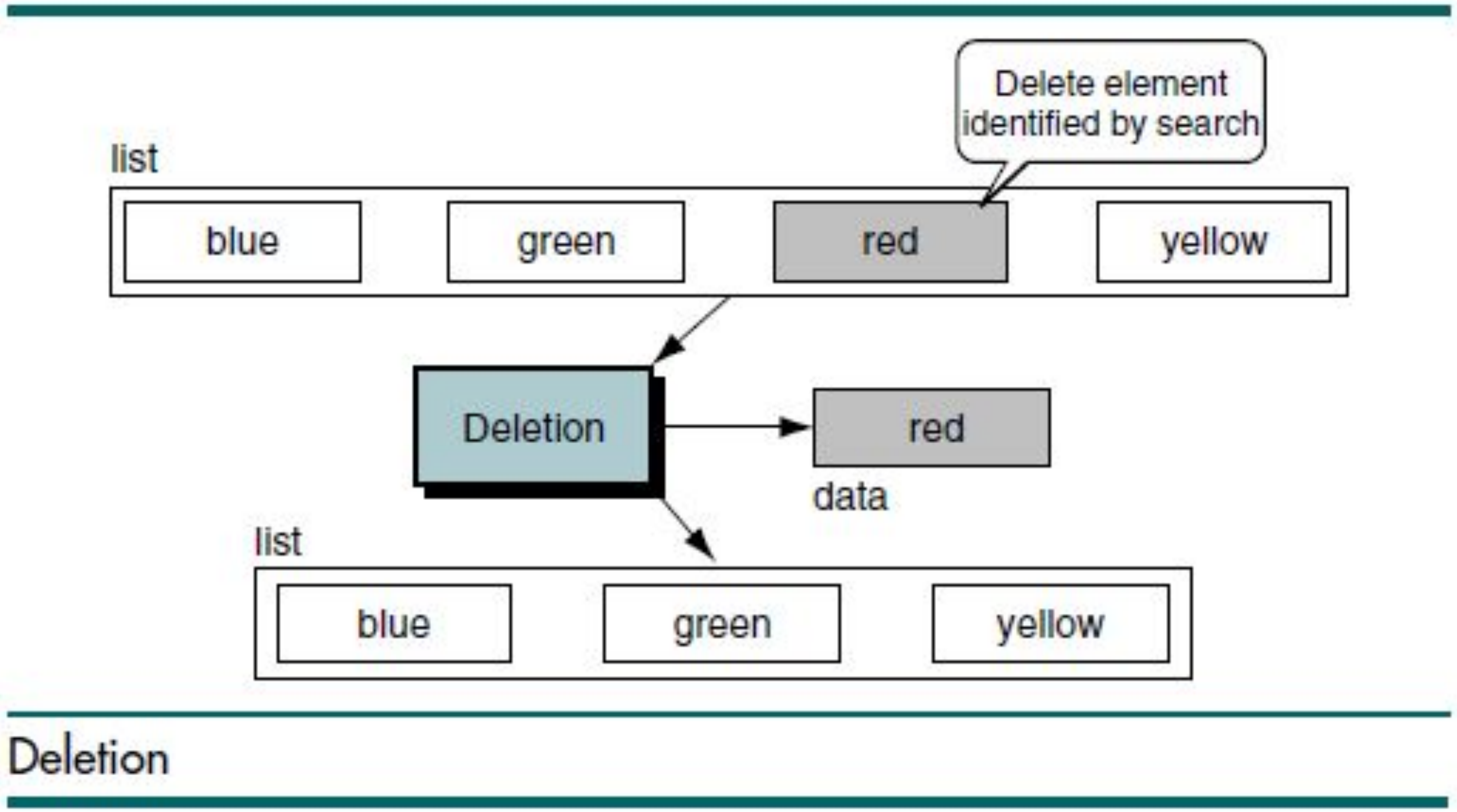
Insertion in ordered list:



Insertion

Deletion operation:

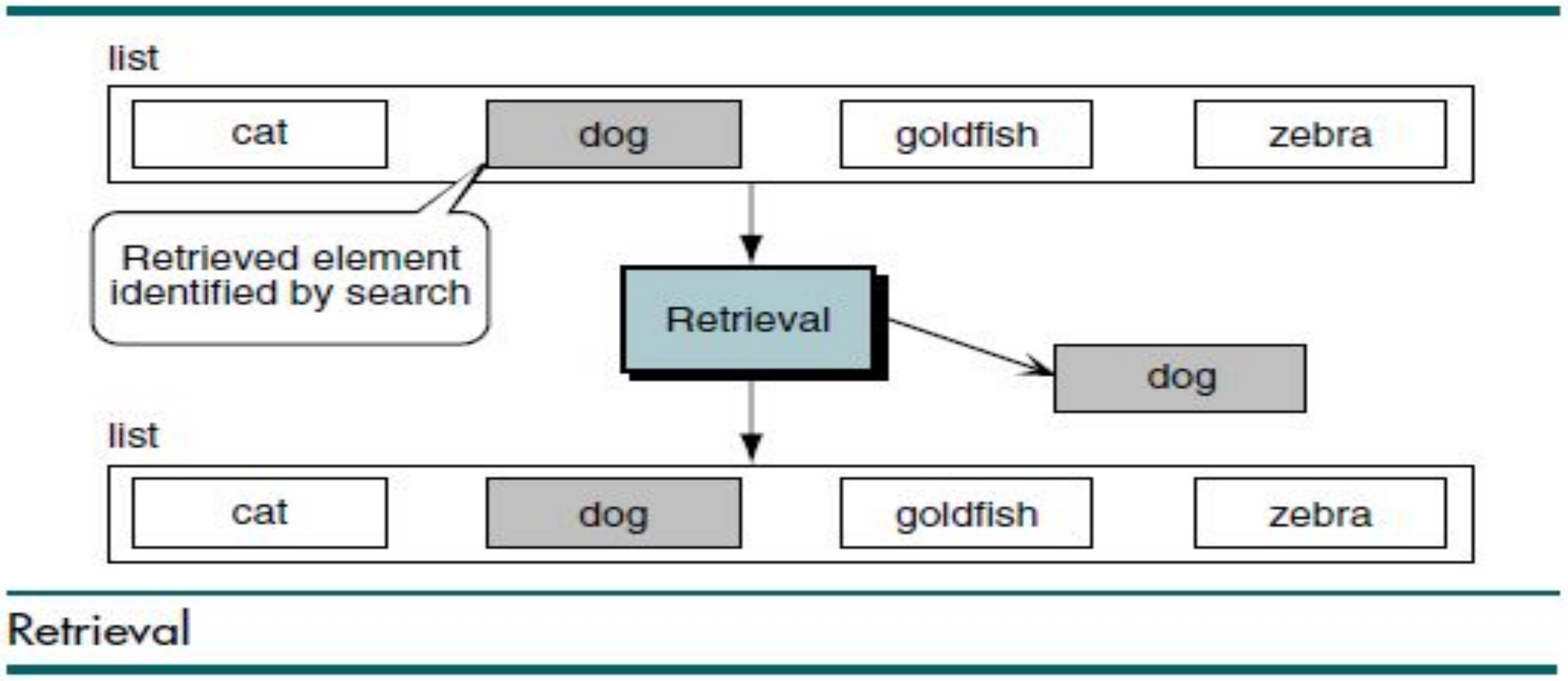
- Deletion from a list requires following set of operations:
- Searching of list to locate the data to be deleted.
- Once located, the data are removed from the list.



Deletion

Retrieval operation:

- List retrieval requires that data be located in a list and presented to the calling module without changing the contents of the list.
- A search algorithm can be used to locate the data to be retrieved from a list.



Traversal

- List traversal processes each element in a list in sequence.
- It requires a looping rather than a search.
- Each execution of the loop processes one element in the list.
- The loop terminates when all elements have been processed.

```
while( node!=NULL)
{
    //traversal logic
    //visit every node
    //get the next node address
}
```

List Implementation:

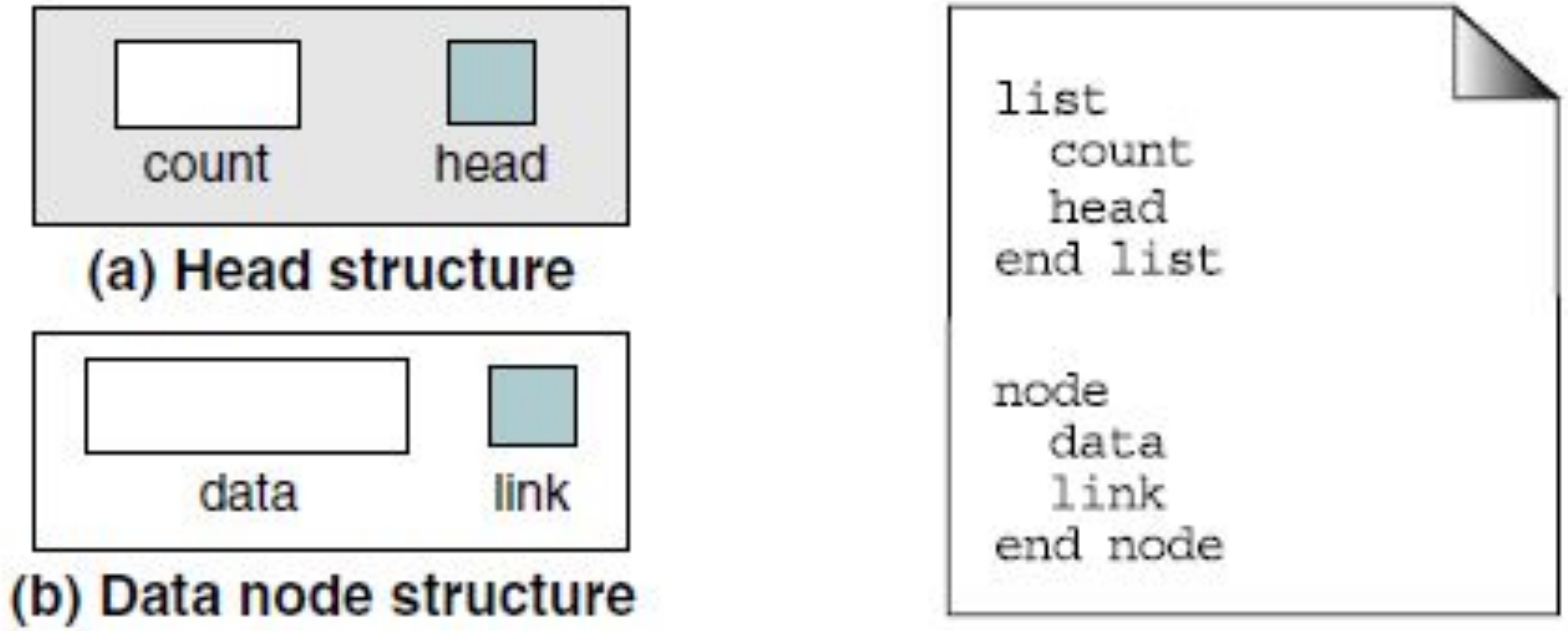
- Lists can be implemented using two methods
  1. Using arrays
  2. Using linked lists

- Linked list implementation is preferred over array implementation because of following reasons:

1. Data are easily inserted at the beginning, in the middle, or at the end of the list.
2. Data can be deleted at the beginning, in the middle, or at the end of the list.

Data Structure for List implementation:

- To implement a list, we need two different structures, a head node and data node



(a) Head structure

(b) Data node structure

```
list
   count
   head
end list

node
   data
   link
end node
```

Head Node and Data Node

Head Node

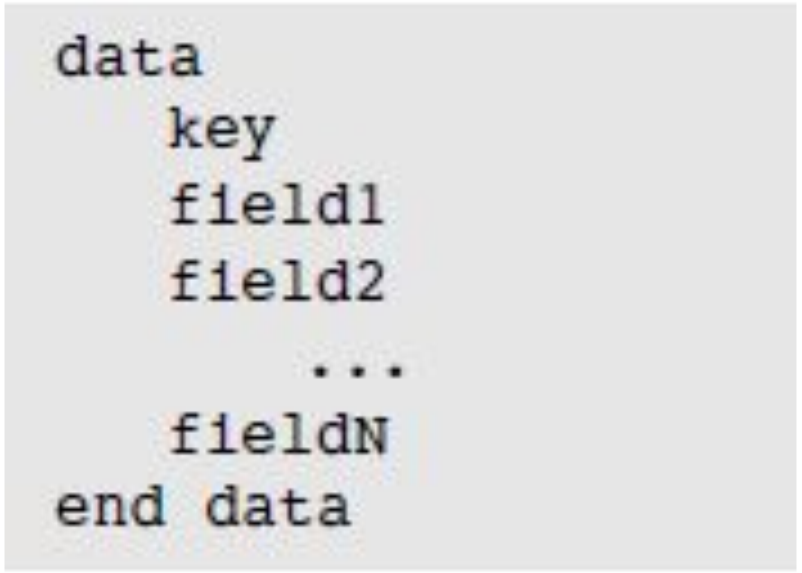
- Head node structure that stores the head pointer and other data about the list.
- Head node contains data about a list, the data are known as metadata; that is, they are data about the data in the list.

For example, the head structure in contains one piece of metadata: count, an integer that contains the number of nodes currently in the list.

Other metadata, such as the greatest number of nodes during the processing of the list, are often included when they serve a useful purpose.

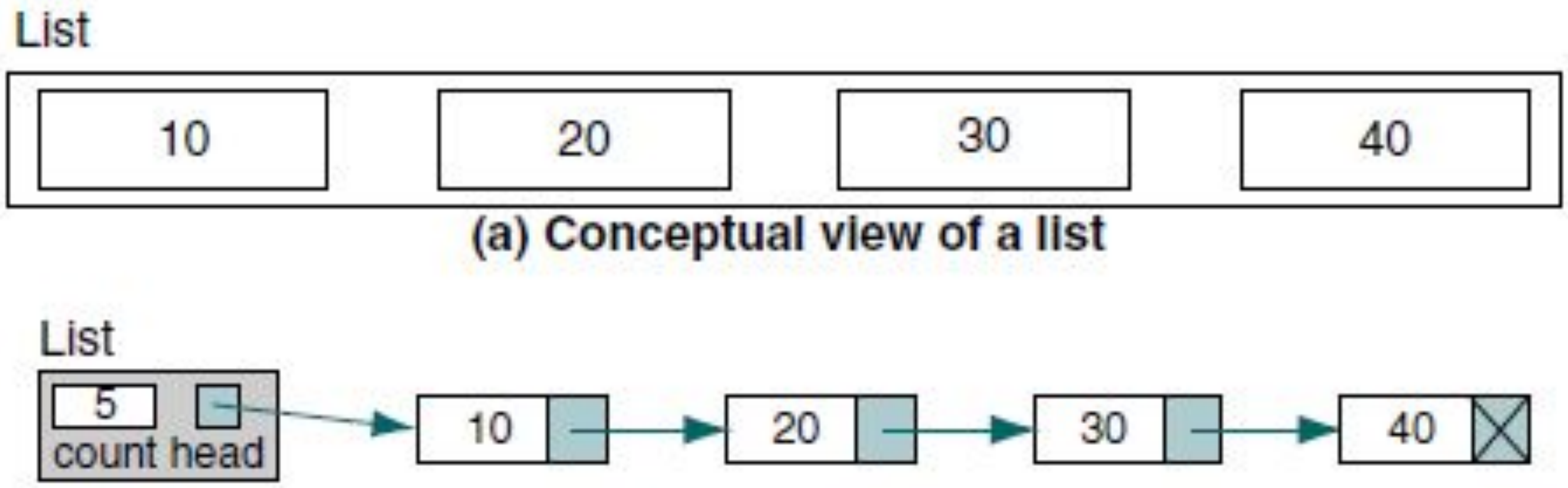
Data Node

- The data type for the list depends entirely on the application.
- Key field for applications that require searching by key.



```
data
   key
   field1
   field2
       ...
   fieldN
end data
```

Linked List Implementation of a List:



(a) Conceptual view of a list

(b) Linked list implementation

Linked List Implementation of a List

Algorithms for List ADT

- Create list

- Insert Node
- Insert into Empty List
- Insert at Beginning
- Insert in Middle
- Insert at End

- Delete Node
- Delete First Node