Введение в компьютерные науки

ЛЕКТОР К.Т.Н. МОХОВ В.А.

ГЛАВА 8. СТРУКТУРЫ ДАННЫХ

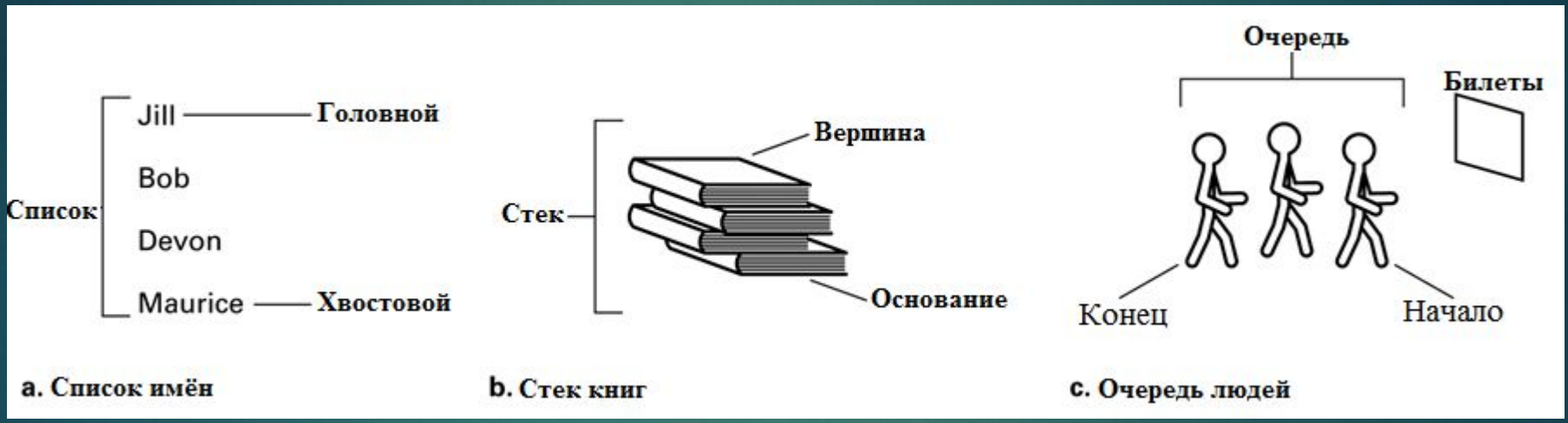
Часть 8: Структуры данных

- 8.1. Массивы.
- 8.2. Списки.
- 8.3. Стеки.
- 8.4. Очереди.
- 8.5. Древовидные структуры.
- 8.6. Специализированные типы данных.
- 8.7. Указатели в машинном языке.

Основные структуры данных

- Однородный массив
- Неоднородный массив
- Список
  - Стек
  - Очередь
- Дерево

Рисунок 8.1 Список, стеки, и очередь



a. Список имён

b. Стек книг

c. Очередь людей

Терминология списков

- **Список:** Набор данных, элементы которого расположены последовательно
- **Головной:** Начало списка
- **Хвостовой:** Конец списка

Терминология стеков

- **Стек:** список, в котором элементы удаляются и вставляются только на одном конце структуры
- **LIFO:** последним пришёл — первым ушел
- **Вершина:** Начало списка(стека)
- **Основание:** Конец списка(Стека)
- **Извлечение :** Процесс удаления объекта из стека
- **Вставка:** Процесс влючения объекта в стек

Терминология очереди

- **Очередь:** список, в котором включение элементов выполняется на одном конце, а извлечение - на другом
- **FIFO:** Первым вошёл, первым вышел

Рисунок 8.2 Пример организационной структуры

Терминология для структуры «дерево»

- **Дерево:** Набор данных, элементы которого имеют иерархическую организацию
- **Вершина:** Элемент дерева
- **Корневая вершина:** Самая верхняя точка
- **Листы** или **Конечные вершины:** Вершины на противоположенной стороне дерева

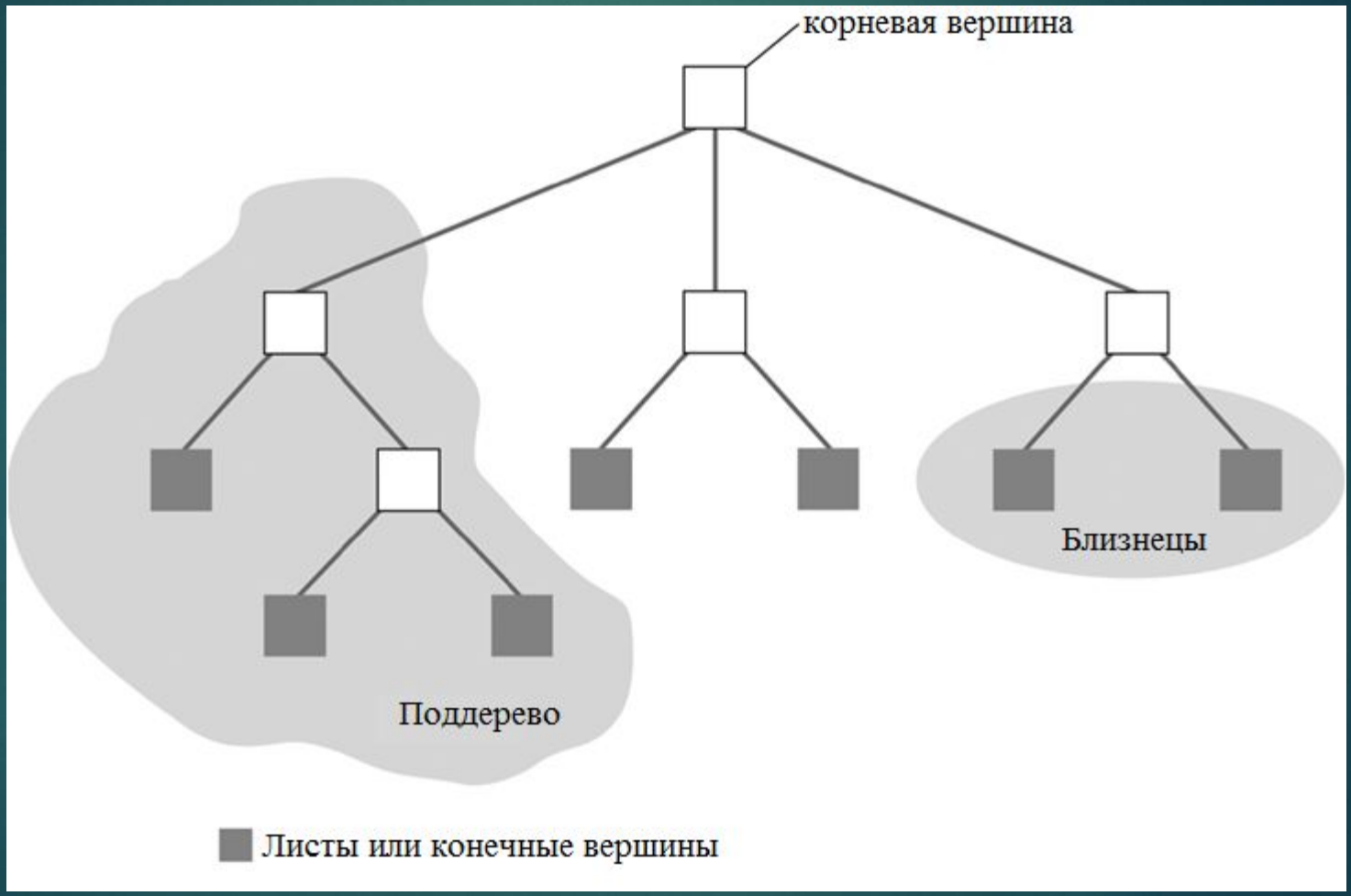
Терминология для структуры «дерево» (продолжение)

- **Родительские вершины:** Непосредственные предки некоторых вершин
- **Дочерние вершины:** Непосредственные потомки некоторых вершин
- **Предок:** Родитель, родитель родителя и т.д.
- **Потомок:** Ребёнок, ребёнок ребёнка, и т.д.
- **Близнецы:** Узлы имеющие общих предков

Терминология для структуры «дерево» (продолжение)

- **Бинарное дерево:** дерево, в котором каждая вершина имеет не более двух дочерних вершин
- **Глубина:** количество вершин на самом длинном пути от корня к листу

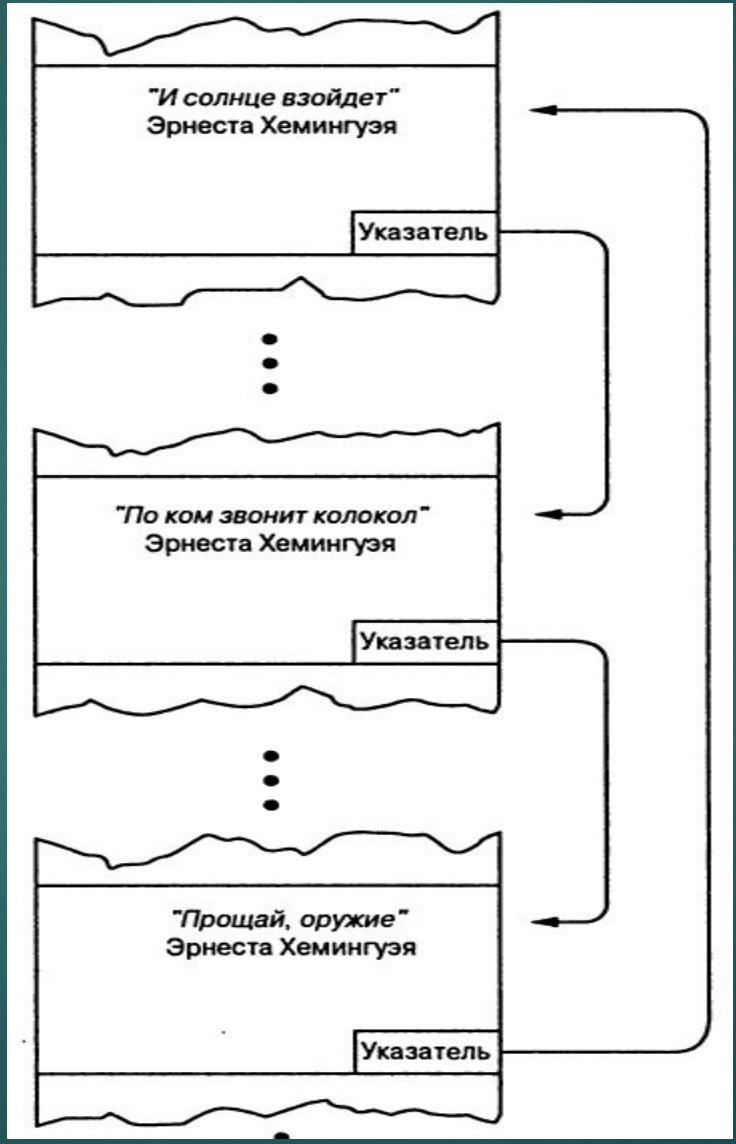
Рисунок 8.3 Структура «дерево»



Дополнительные понятия

- Статические структуры данных: Размеры и формы структур данных, не изменяются
- Динамические структуры данных: Размеры и формы структур данных можно изменять
- Указатели: Используется для поиска данных

Рисунок 8.4 Романы расположены по названию, но связаны по авторству



Хранение массивов

- Однородные массивы
  - **Развёртка по столбцам** против **Развёртки по строкам**
  - Адресный полином
- Неоднородные массивы
  - Компоненты могут быть сохранены друг за другом в непрерывном блоке
  - Компоненты могут быть сохранены в различных местах, определенных указателей

Рисунок 8.5 Массив Readings, хранящийся в памяти, начиная с ячейки памяти с адресом x

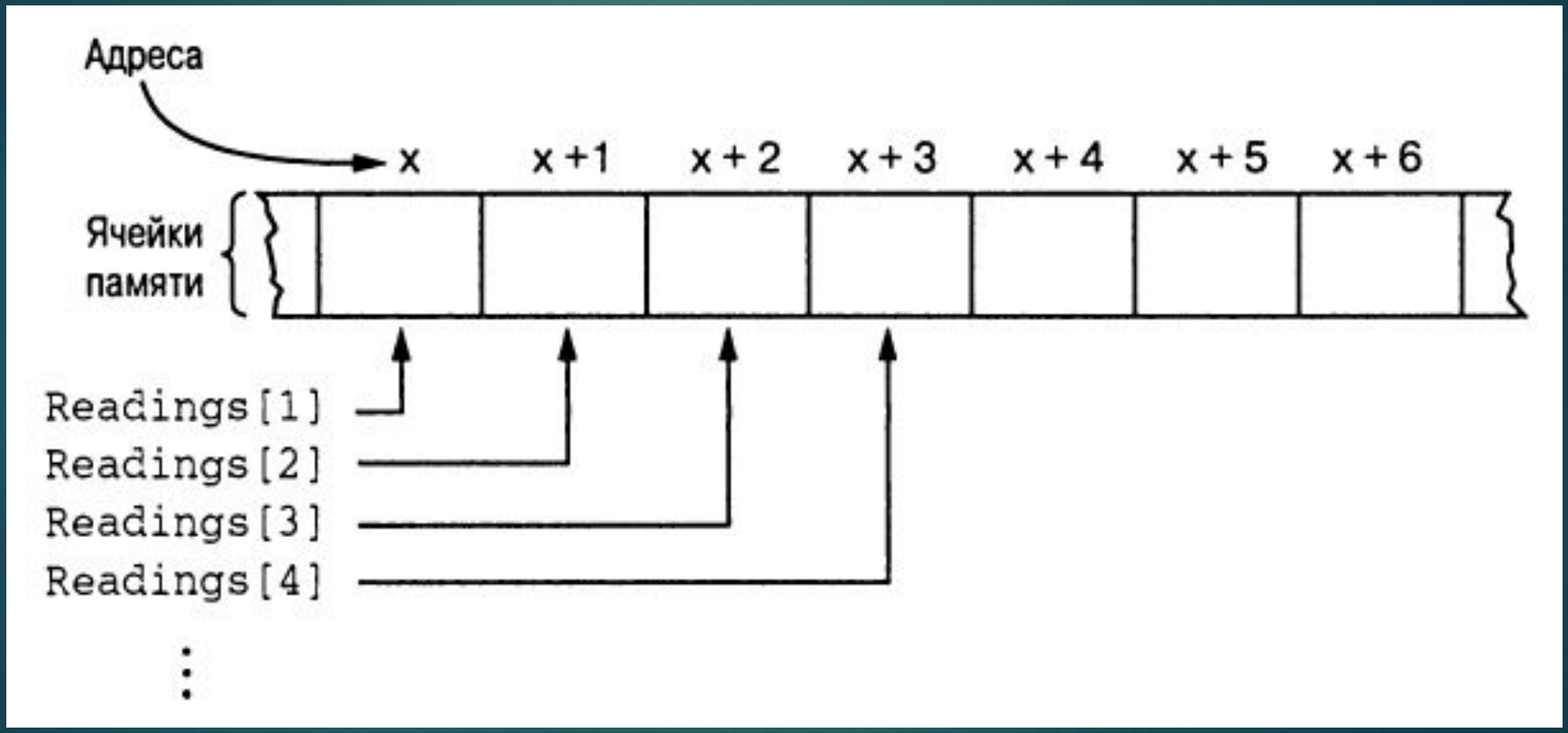


Рисунок 8.6 двухмерный массив с четырьмя строками и пятью столбцами, записанный в памяти с развёрткой по строкам

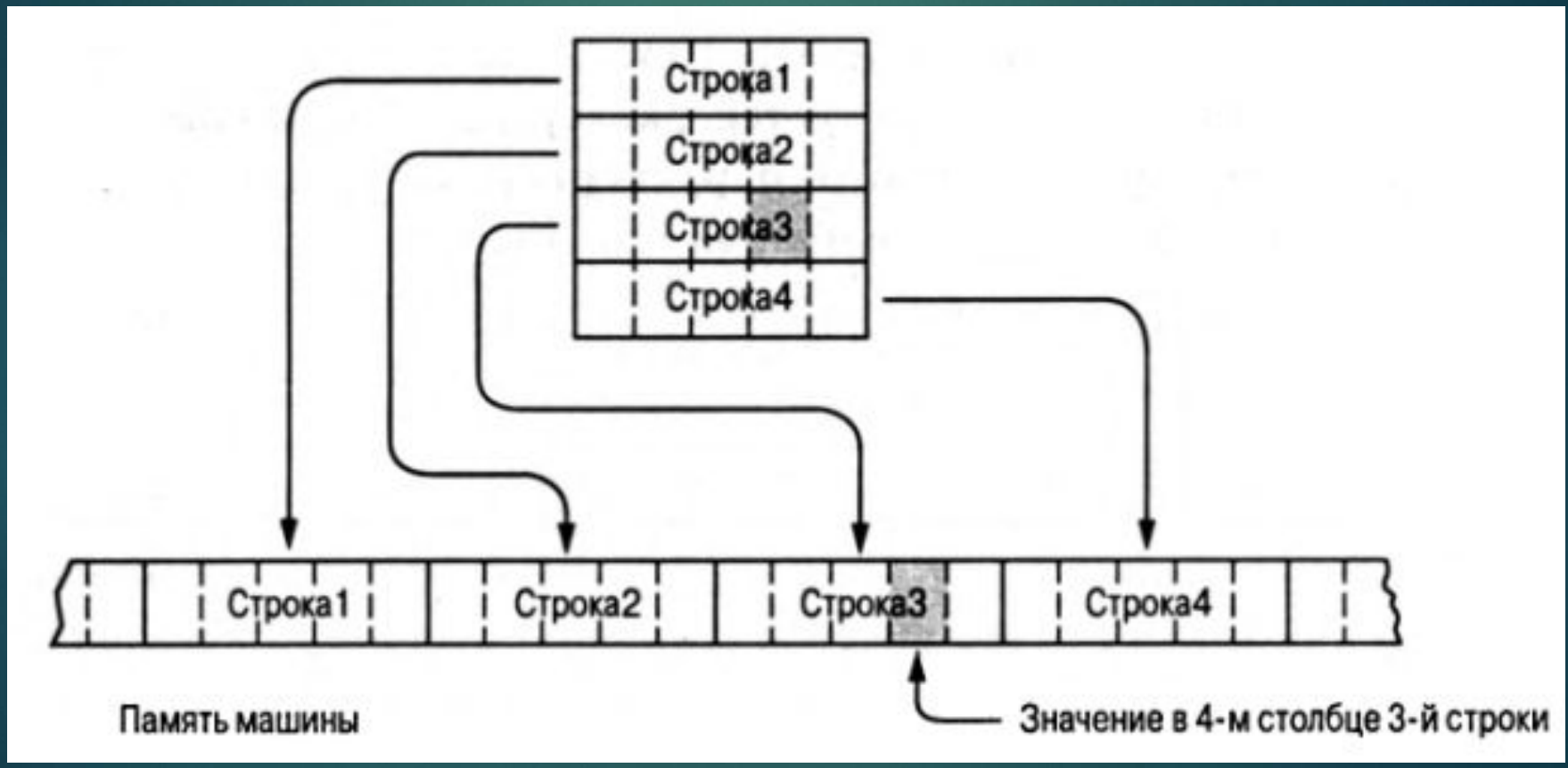
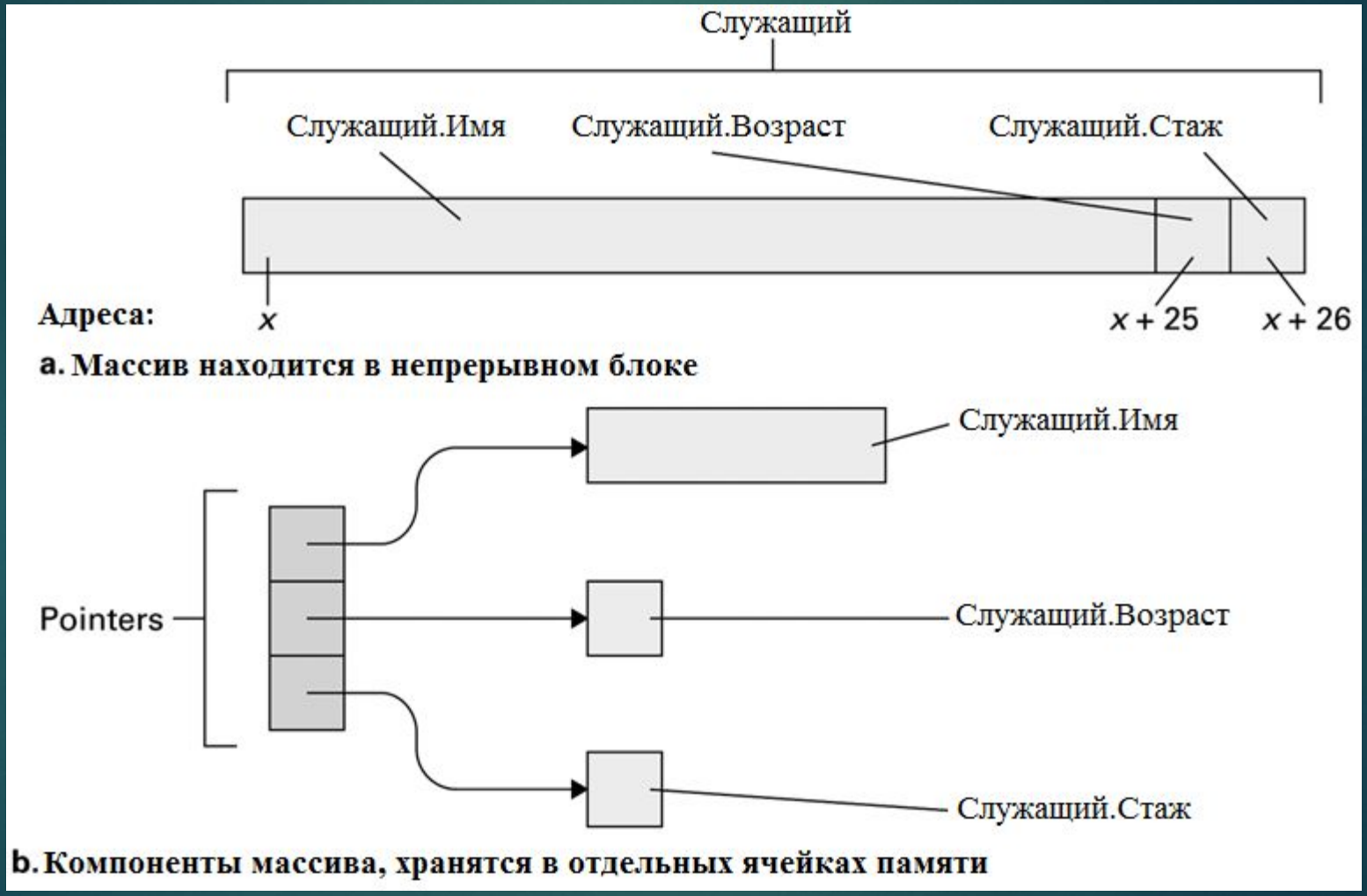


Рисунок 8.7 Хранение неоднородного массива «Служащий»



Хранение списков

- **Непрерывный список:** Список хранится в однородном массиве
- **Связанный список:** Список, в котором каждая запись связана указателем
  - **Указатель головного элемента:** указатель на первую запись в списке
  - **Нулевой указатель:** Указывает на конец списка

Рисунок 8.8 Список имён, сохраняемый в памяти в виде непрерывного списка

Рисунок 8.9 Структура связанного списка

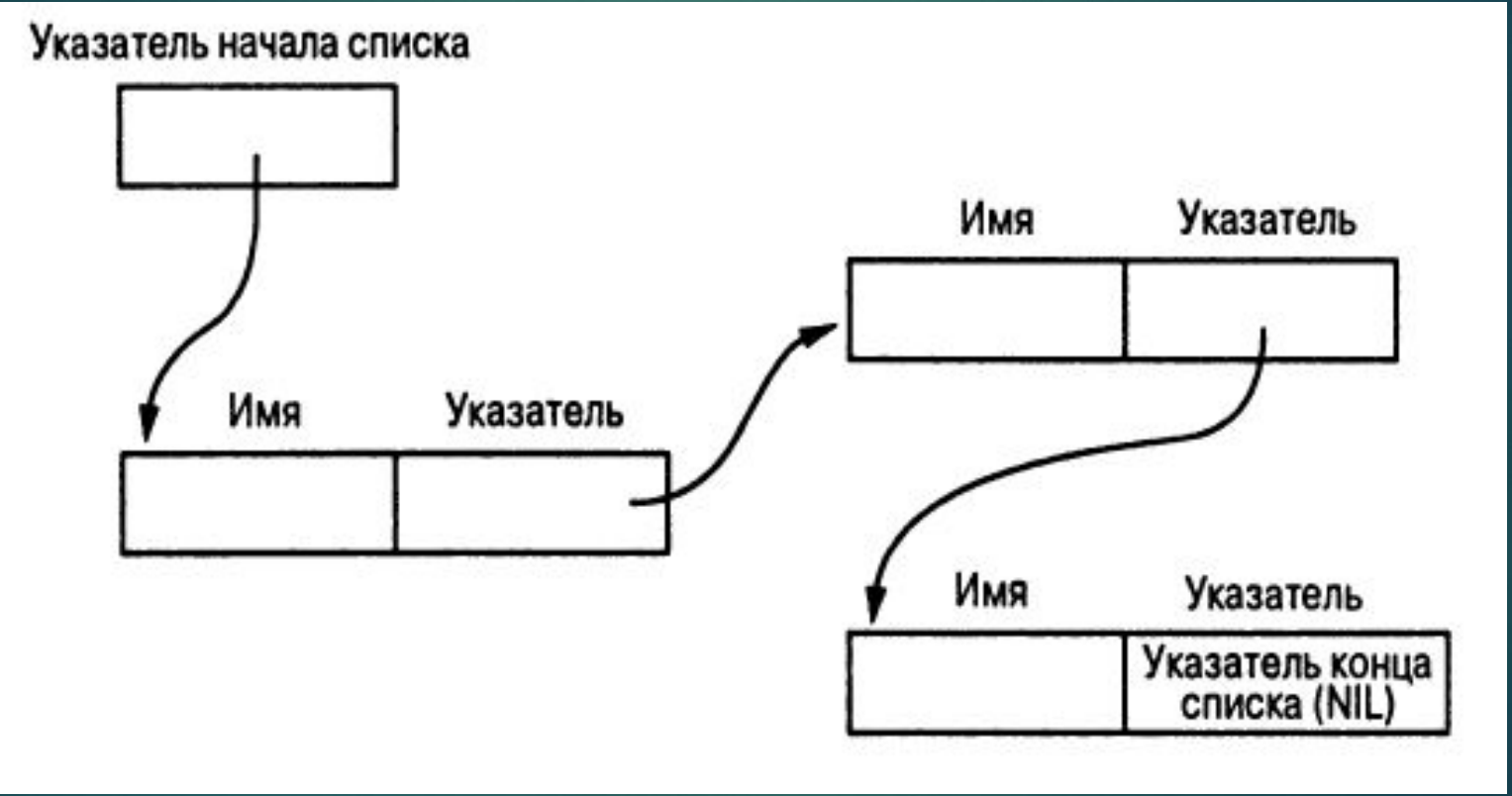


Рисунок 8.10 Удаление элемента из связанного списка

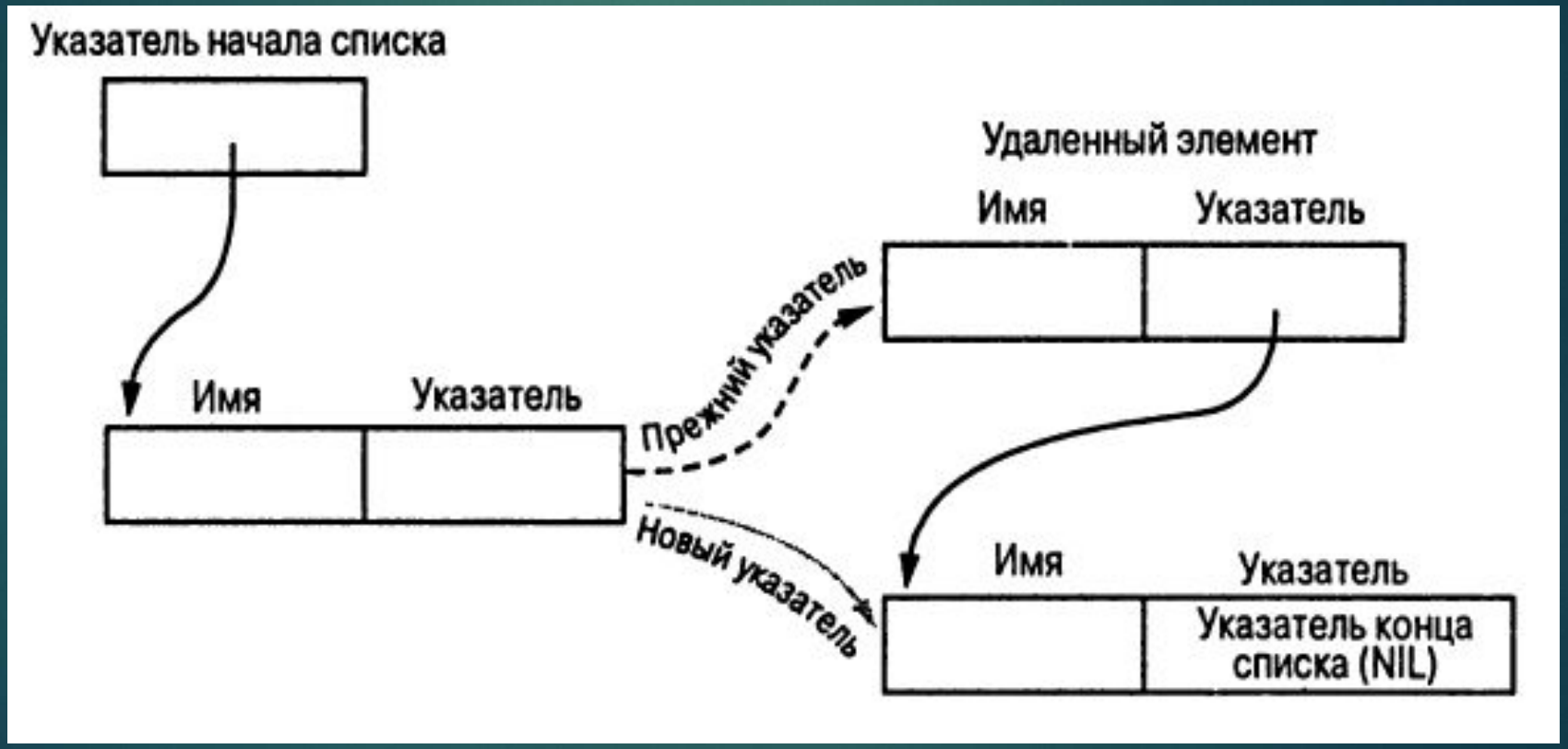
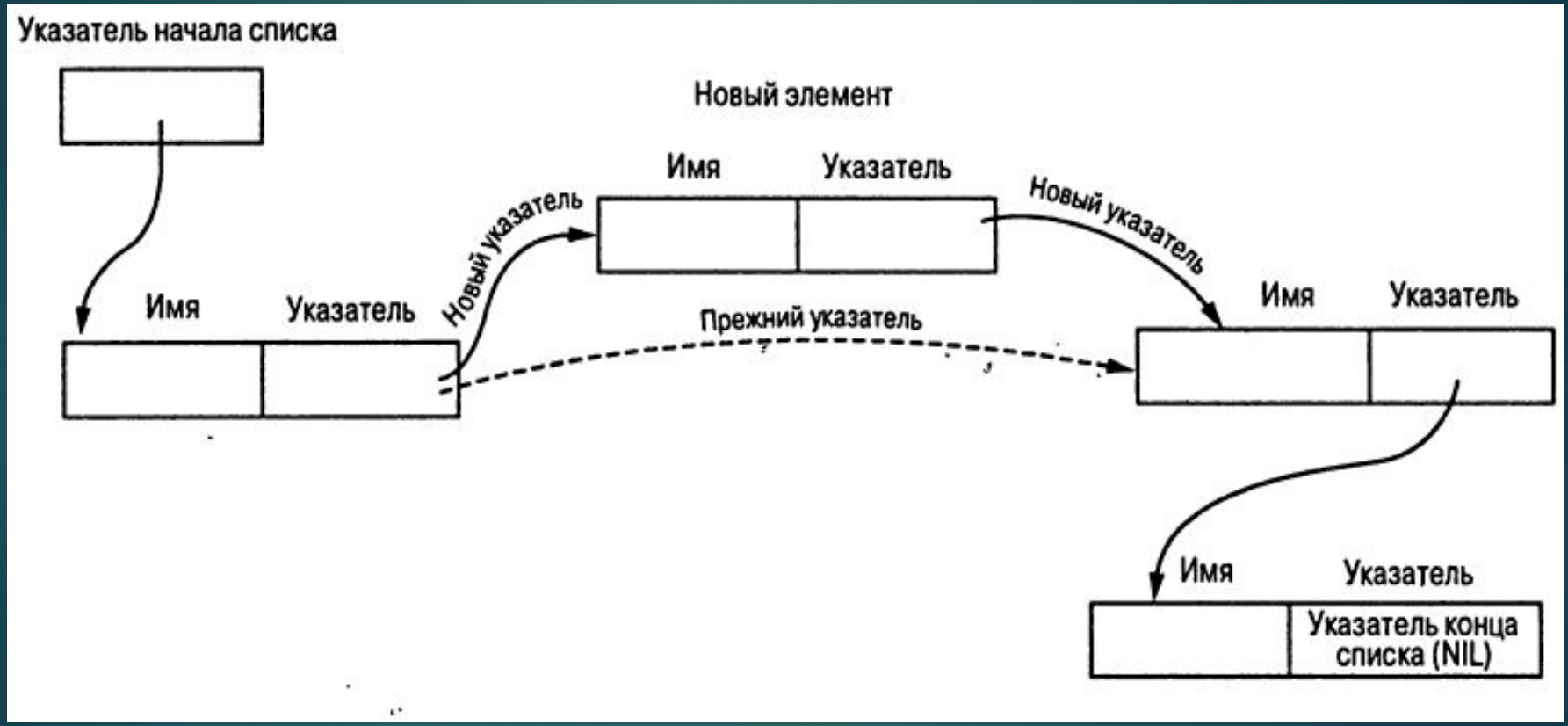


Рисунок 8.11 Включение элемента в связанный список



Хранение стеков и очередей

- Стеки обычно хранятся как смежные списки
- Очереди обычно хранятся как **циклические очереди**
  - Хранится в непрерывном блоке, в котором первая запись примыкает к последней
  - Предотвращает очередь, вылезая из своей выделенной памяти

Рисунок 8.12 Организация стека в памяти компьютера

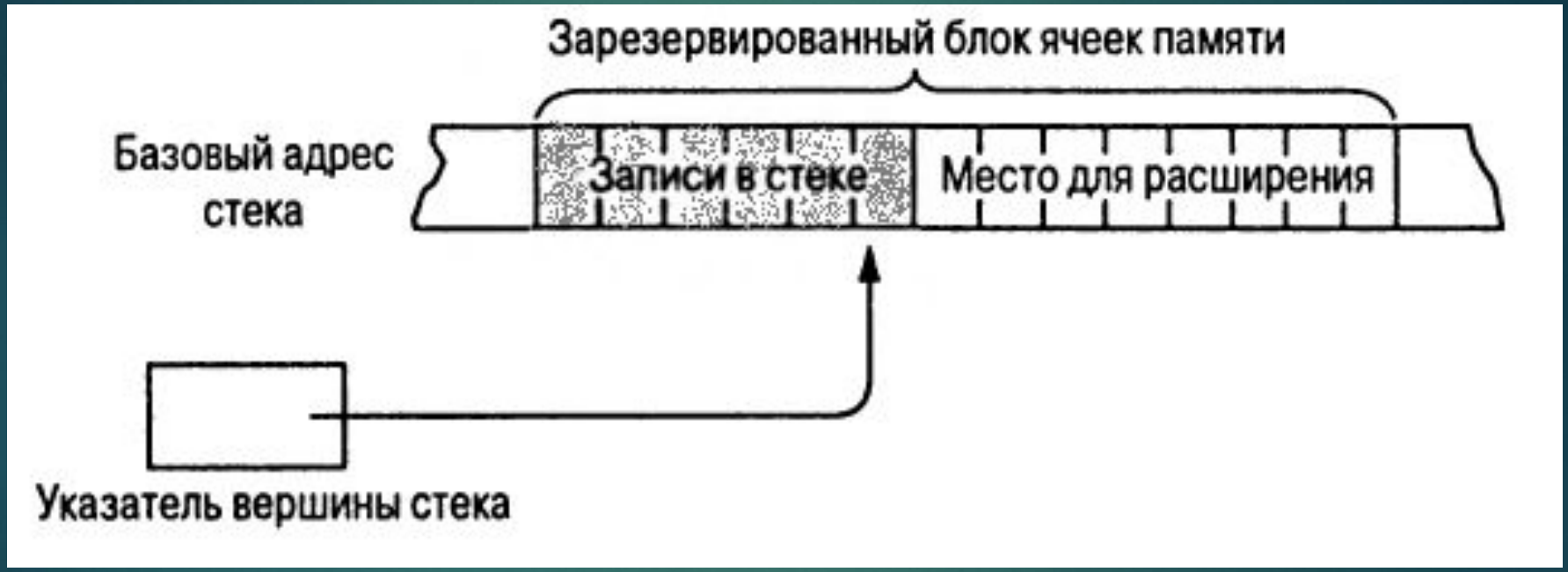
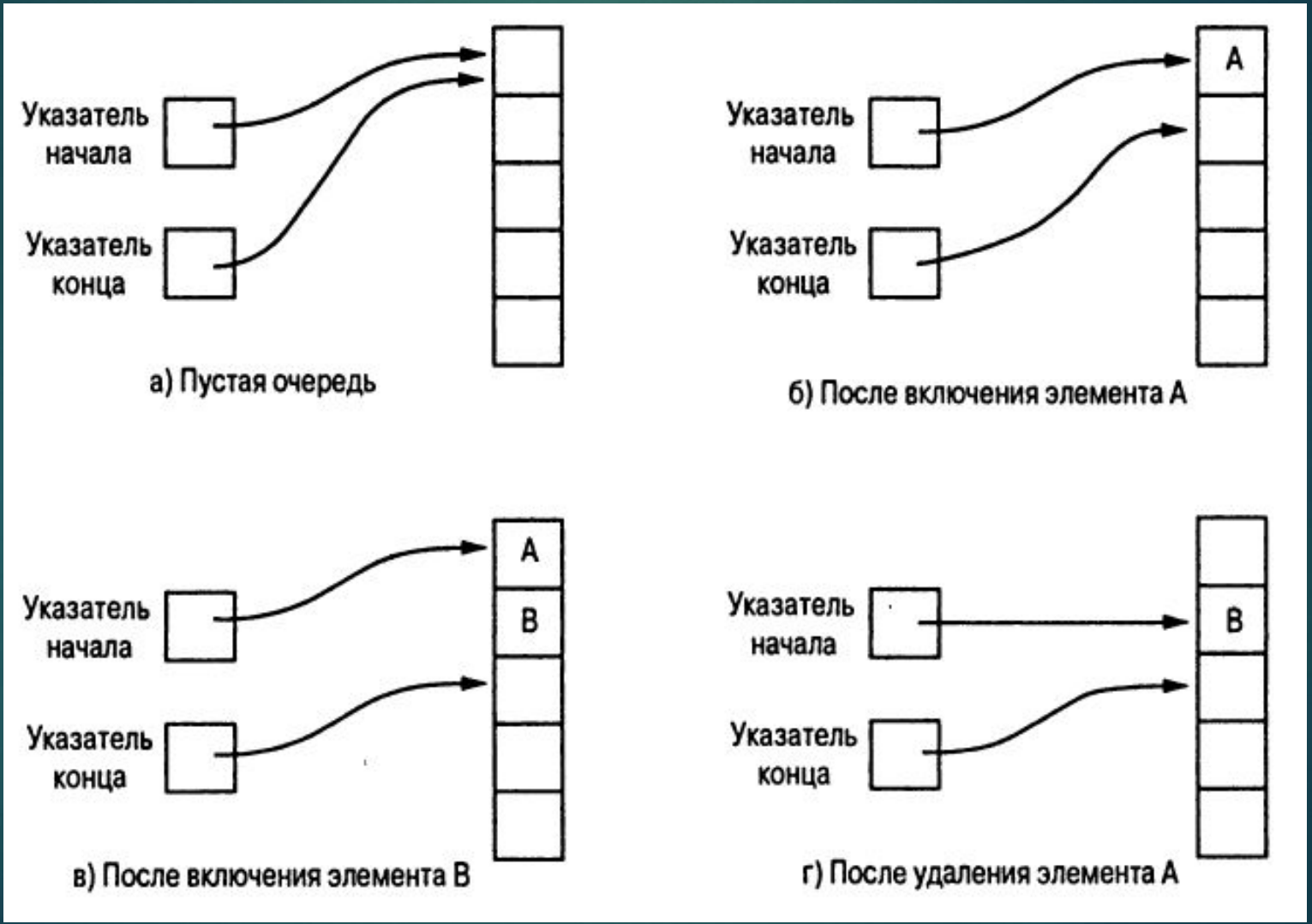


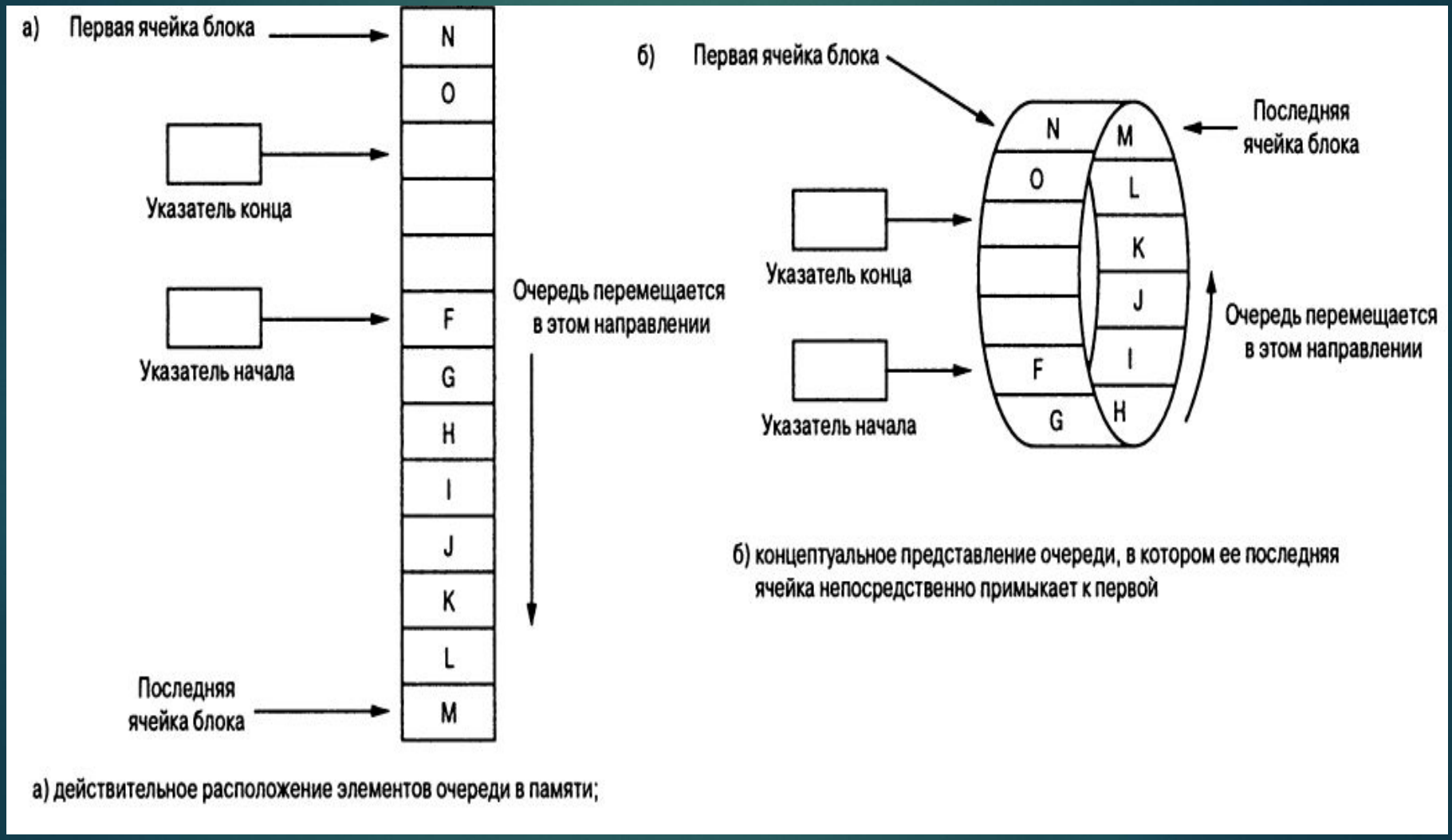
Рисунок 8.13 Реализация очереди с использованием указателей её начала и конца



Хранение бинарных деревьев

- Связанная структура
  - любой узел = данные ячейки + две дочерние вершины
  - Доступ через указатель к корневому узлу
- Структура смежного массива
  - A[1] = Корневая вершина
  - A[2],A[3] = Потомок A[1]
  - A[4],A[5],A[6],A[7] = Потомок A[2] и A[3]

Рисунок 8.14 Циклическая очередь, содержащая буквы F до O



а) действительное расположение элементов очереди в памяти;

б) концептуальное представление очереди, в котором ее последняя ячейка непосредственно примыкает к первой

Рисунок 8.15 Представление вершины бинарного дерева в памяти машины

Рисунок 8.16 Концептуальная и реальная организации бинарного дерева с использованием связанной системы хранения

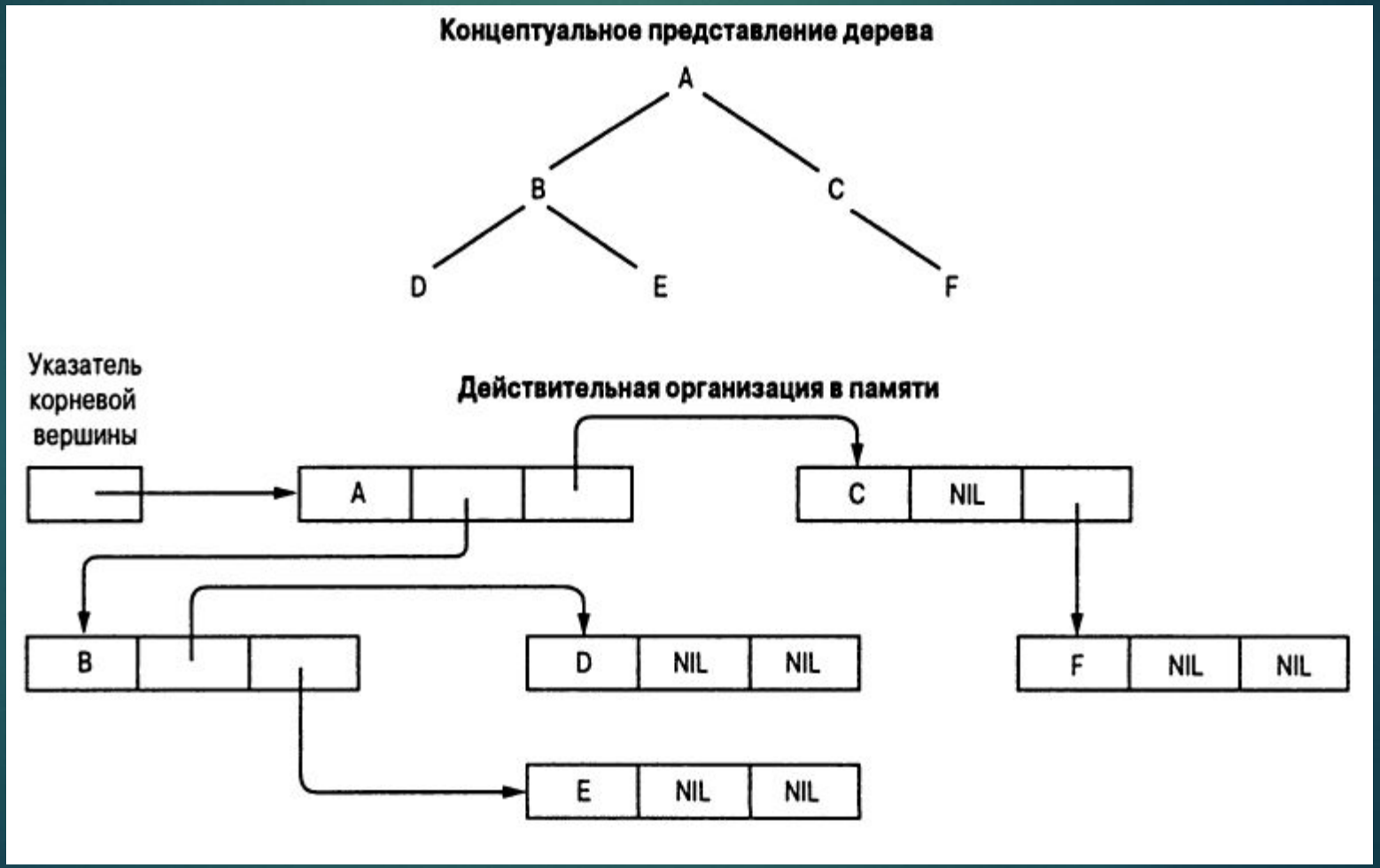


Рисунок 8.17 Древовидная структура, сохранённая в памяти без использования указателей

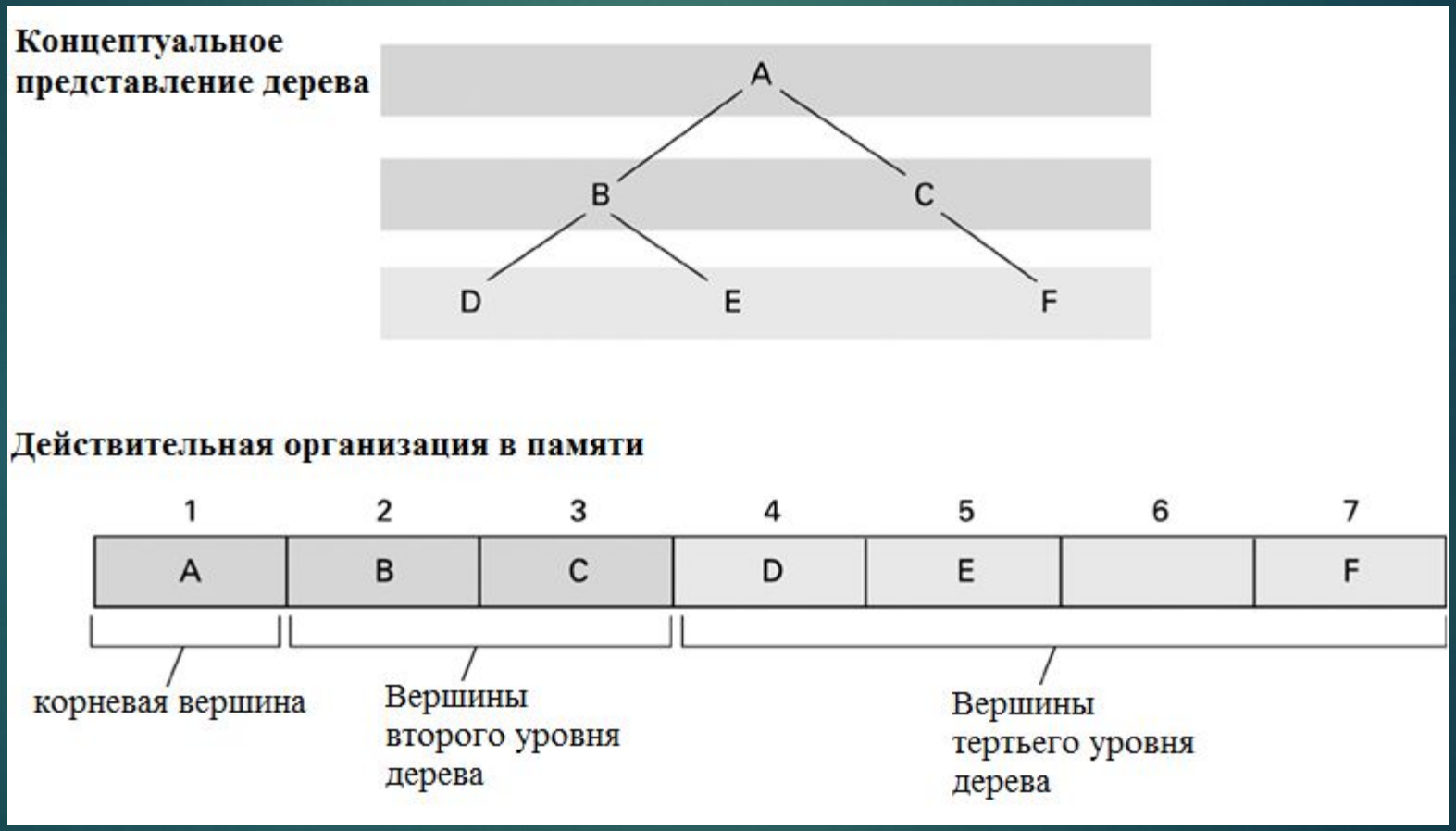
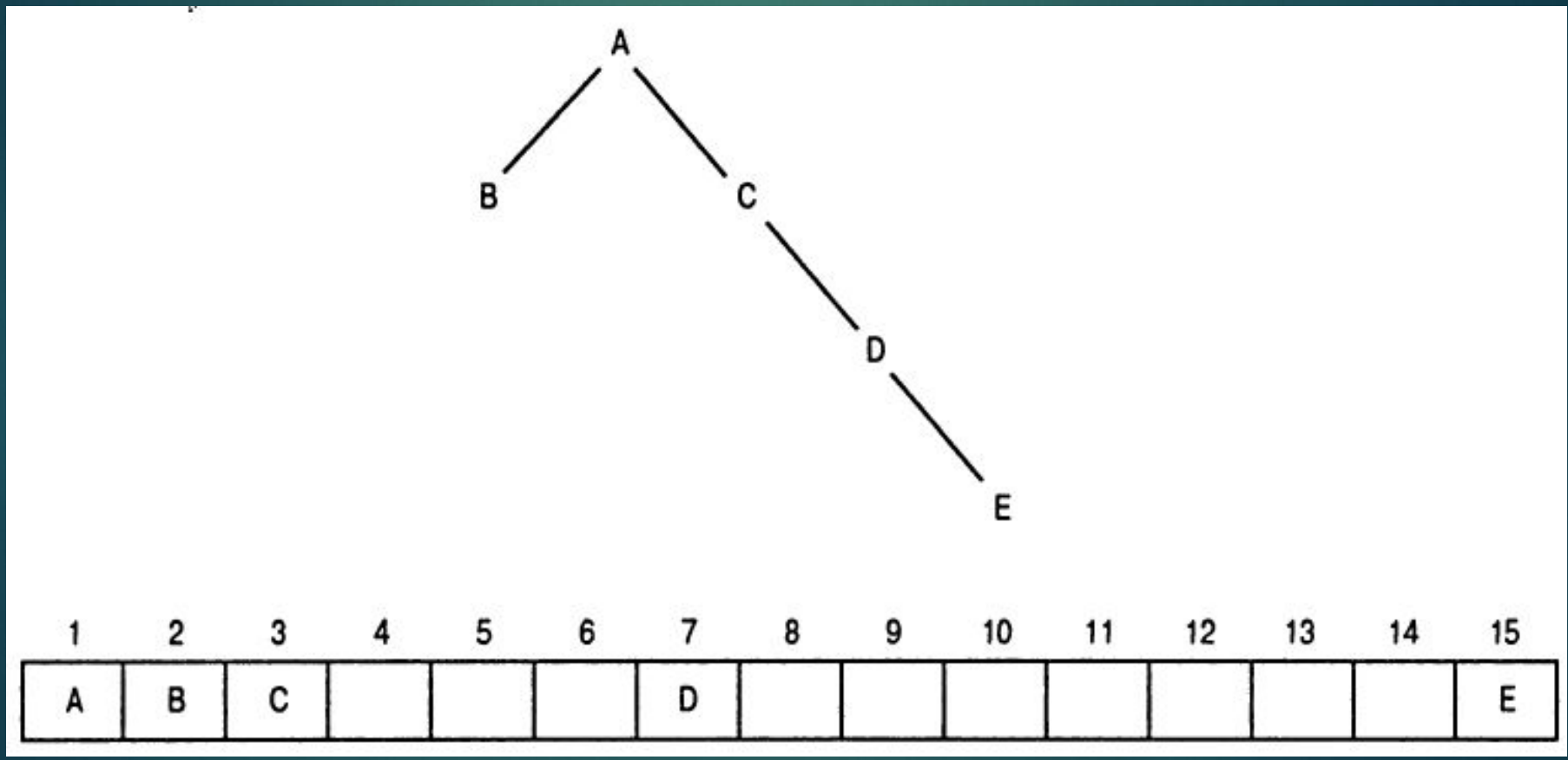


Рисунок 8.18 Пример неполного и несбалансированного дерева в концептуальной форме и схема его размещения в памяти в системе без указателей



Управление структурами данных

- В идеале, структуры данных следует использовать исключительно в предопределенных процедурах.
  - Пример: Типичный стек нуждается в минимальном количестве процедур push и pop.
  - Структура данных вместе с этими процедурами составляет полностью абстрактный инструмент.

Рисунок 8.19 процедура распечатки связанного списка

```
procedure PrintList (<список>)
 Assign <текущий указатель> the value значение указателя на
      головной элемент списка <список>.
 while (<текущий указатель> не равен NIL) do
   (Печатать поле имени из элемента списка <список>, на
    который указывает <текущий указатель>;
   получить значение из поля указателя элемента списка
    <список>, на который указывает <текущий указатель>,
    и присвоить это значение переменной <текущий указатель>.)
```

Исследование на конкретном примере

Проблема: Построить абстрактный инструмент, состоящий из списка имен в алфавитном порядке вместе с операциями поиска, печати, и вставки.

Рисунок 8.20 Латинские буквы от A до M, организованные в виде упорядоченного дерева

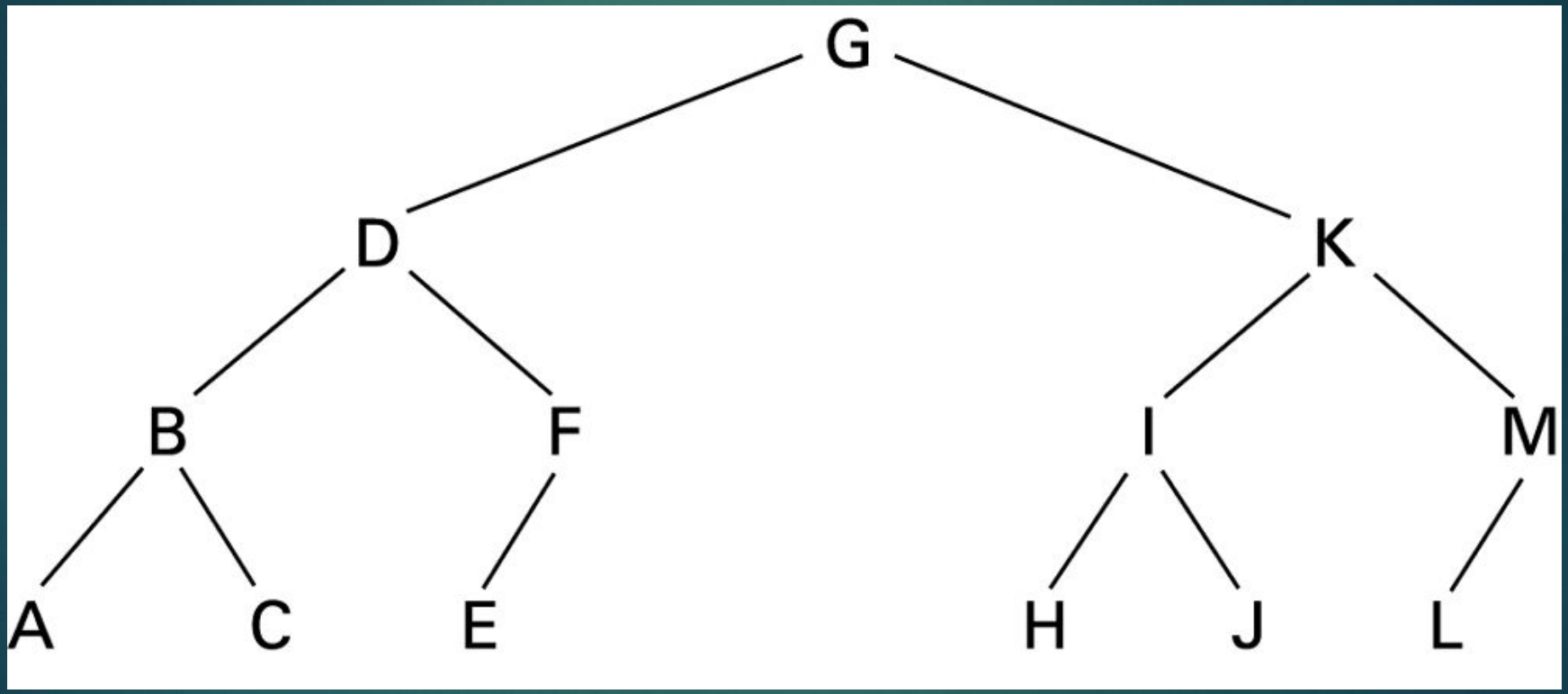


Рисунок 8.21 Процедура двоичного поиска в связанном бинарном дереве

```
** Переменная <текущий указатель> используется для хранения **
** указателя на текущую позицию в дереве. Вершина в этой     **
** позиции называется текущей вершиной.                      **

procedure BinarySearch (<дерево>, <искомое значение>)
assign <текущий указатель> the value значение корневого
    указателя дерева <дерево>;
assign <найдено> the value "ложь";
while (<найдено> равно "ложь" и <текущий указатель> не NIL) do
  [Выбрать подходящий вариант (case) из перечисленных ниже и
      выполнить предусмотренные в нем действия;
  case 1, <искомое значение> = значение из текущей вершины:
    (assign <найдено> the value "истина")
  case 2, <искомое значение> < значения из текущей вершины:
    (assign <текущий указатель> the value указатель на левую
        дочернюю вершину текущей вершины)
  case 3, <искомое значение> > значения из текущей вершины:
    (assign <текущий указатель> the value указатель на правую
        дочернюю вершину текущей вершины)
  ]
if (<найдено> = "ложь") then (объявить поиск неудачным)
                        else (объявить поиск успешным)
```

Рисунок 8.22 Поиск наикратчайшего пути к J

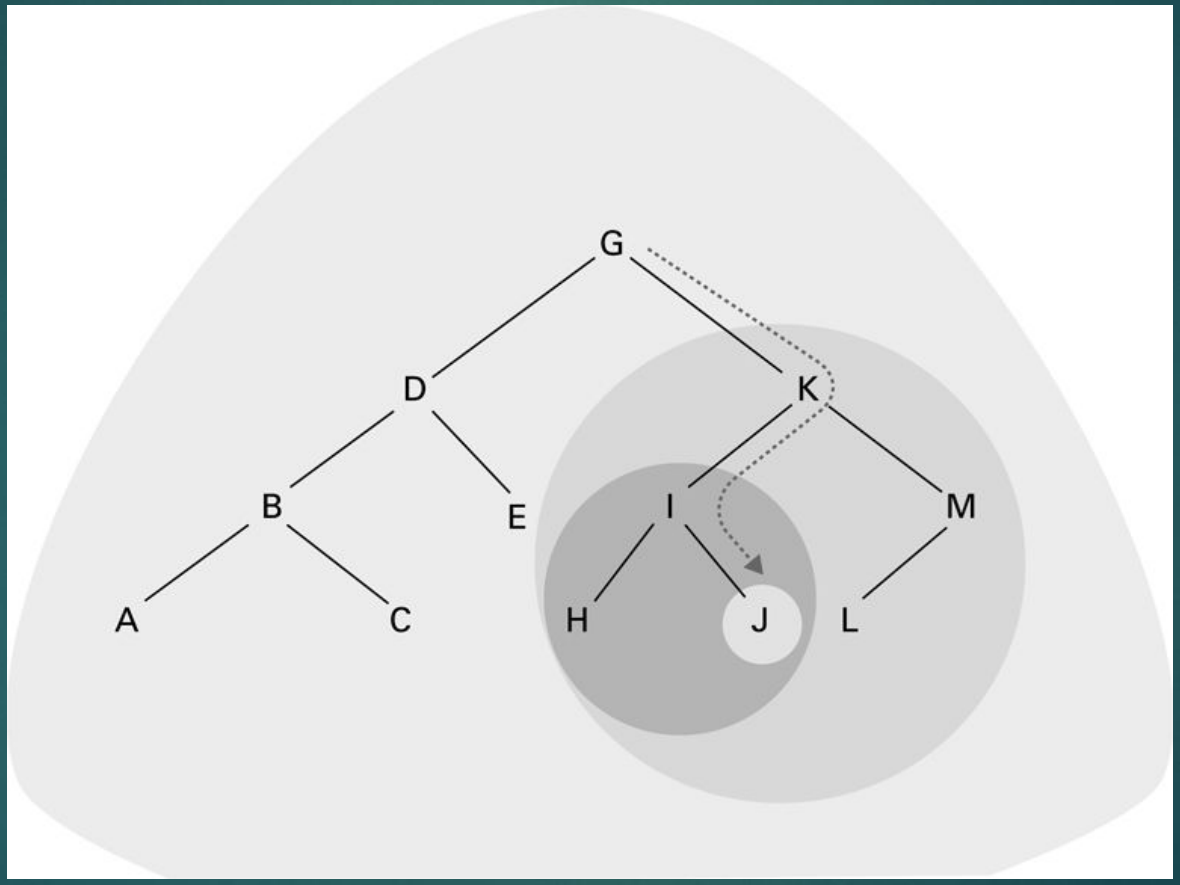


Рисунок 8.23 Печать связанного бинарного дерева в алфавитном порядке

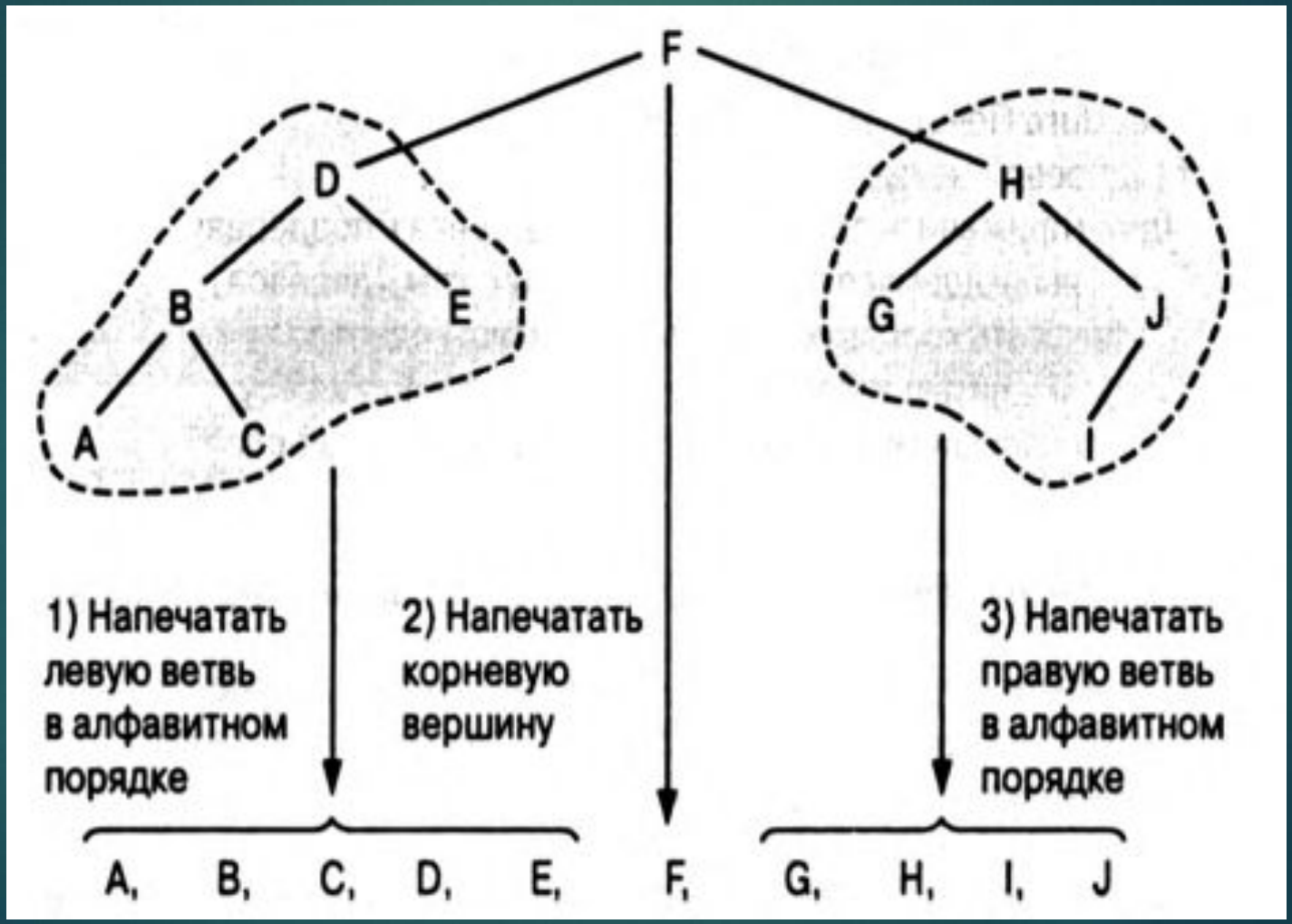


Рисунок 8.24 Процедура печати связанного бинарного дерева в алфавитном порядке

```
procedure Печать_Дерева (<дерево>)
 if (<дерево> не пусто)
   then (применить процедуру Печать_Дерева к поддереву,
          являющемуся левой ветвью структуры <дерево>;
        печатать корневую вершину структуры <дерево>;
        применить процедуру Печать_Дерева к поддереву,
          являющемуся правой ветвью структуры <дерево>)
```

Рисунок 8.25 Включение элемента M в список B,E,G,Y,J,K,N,P, хранящийся в виде бинарного дерева

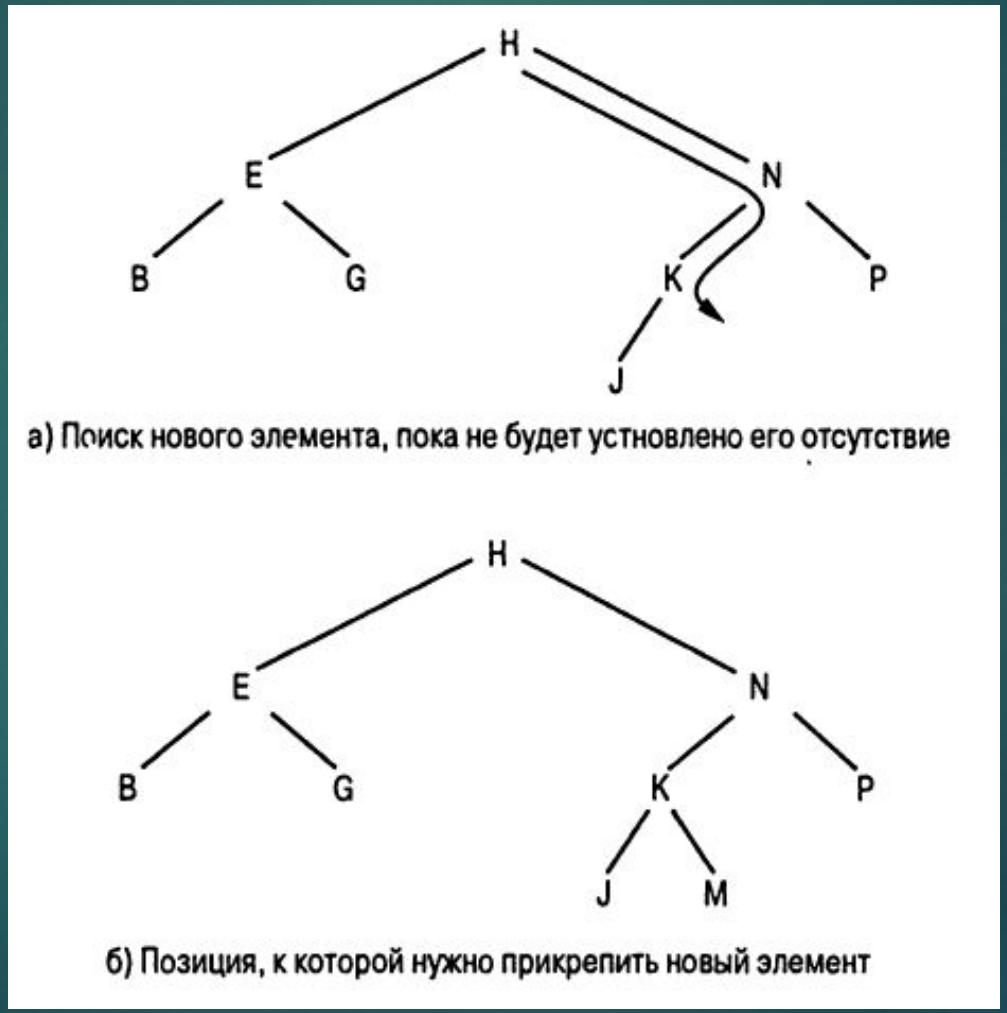


Рисунок 8.26 Процедура включения элемента в связанное упорядоченное бинарное дерево

```
** Переменная <текущий указатель> используется для хранения   **
** указателя на текущую позицию в дереве. Вершина,            **
** находящаяся на этой позиции, называется текущей.           **
** Переменная <предыдущий указатель> содержит указатель на    **
** родительскую вершину текущей вершины, которая называется   **
** предшествующей.                                            **

procedure Включение (<дерево>, <вставляемый элемент>)

** Прежде всего найдем место вставки новой вершины            **

assign <текущий указатель> the value значение корневого
       указателя структуры <дерево>;
assign <найдено> the value "ложь";
while (<найдено> есть "ложь" и <текущий указатель> не NIL) do
  [Выбрать подходящий вариант (case) из перечисленных ниже и
       выполнить предусмотренные в нем действия;
  case 1, <вставляемый элемент> = значение в текущей вершине:
    (assign <найдено> the value "истина")
  case 2, <вставляемый элемент> < значения в текущей вершине:
    (assign <предыдущий указатель> the value <текущий указатель>;
     assign <текущий указатель> the value указатель на левую
       дочернюю вершину текущей вершины)
  case 3, <вставляемый элемент> > значения в текущей вершине:
    (assign <предыдущий указатель> the value <текущий указатель>;
     assign <текущий указатель> the value указатель на правую
       дочернюю вершину текущей вершины)
  ]
** Теперь следует добавить новый элемент как дочернюю         **
** вершину текущей вершины. Заметим, что если переменная      **
** <текущий указатель> по-прежнему равна значению корневого   **
** указателя, то исходное дерево пусто                        **

if (<найдено> = "ложь")
  then (Создать новую вершину, содержащую <вставляемый элемент>;
       Выбрать подходящий вариант (case) из перечисленных
         ниже и выполнить предусмотренные в нем действия;
    case 1, <текущий указатель> = <корневой указатель>:
      (включить новую вершину в качестве корневой)
    case 2, <вставляемый элемент> < значения в предыдущей вершине:
      (включить новую вершину в качестве левой дочерней
       вершины предыдущей вершины)
    case 3, <вставляемый элемент> > значения в предыдущей вершине:
      (включить новую вершину в качестве правой дочерней
       вершины предыдущей вершины)
    )
```

Типы данных определяемы пользователем

- Шаблон для неоднородной структуры
- Пример:

```
определить тип EmployeeType
{char    Name[25];
 int     Age;
 real    SkillRating;
}
```

Абстрактные типы данных

- Типы данных определяемы пользователем с процедурами доступа и обработки
- Пример:

```
Определить тип StackType
{int StackEntries[20];
 int StackPointer = 0;
 procedure push(value)
     {StackEntries[StackPointer] ← value;
      StackPointer ¬ StackPointer + 1;
     }
 procedure pop . . .
}
```

Классы

- Абстрактный тип данных с дополнительными функциями
  - Характеристики могут быть унаследованы
  - содержимое может быть инкапсулировано
  - конструктор методов для инициализации новых объектов

Рисунок 8.27 Стек целых чисел, реализованных в Java и C

```
class StackOfIntegers
{private int[] StackEntries = new int[20];
 private int StackPointer = 0;


 public void push(int NewEntry)
 {if (StackPointer < 20)
    StackEntries[StackPointer++] = NewEntry;
 }

 public int pop()
 {if (StackPointer > 0) return StackEntries[--StackPointer];
  else return 0;
 }
}
```

Указатели в машинном языке

- **Непосредственная адресация:** Инструкция содержит доступ к данным
- **Прямая адресация:** Инструкция содержит адрес данных, которые будут доступны
- **Косвенная адресация:** Инструкция содержит местоположение адресов данных, которые должны быть доступны

Рисунок 8.28 Наша первая попытка на расширение машинного языка в Приложении С, чтобы воспользоваться указателями

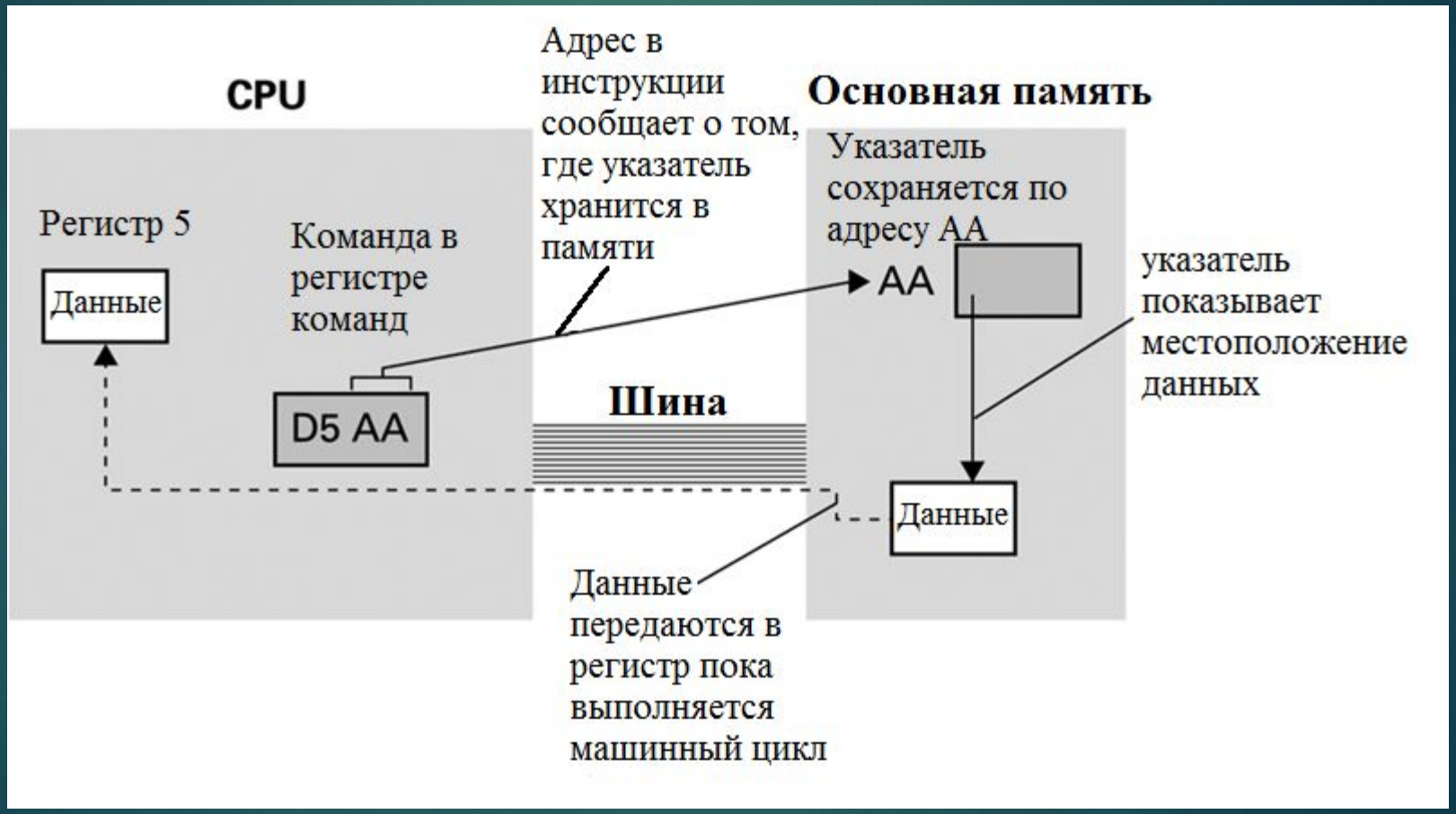


Рисунок 8.29 Загрузка регистра из ячейки памяти, которая находится с помощью указателя хранящегося в регистре