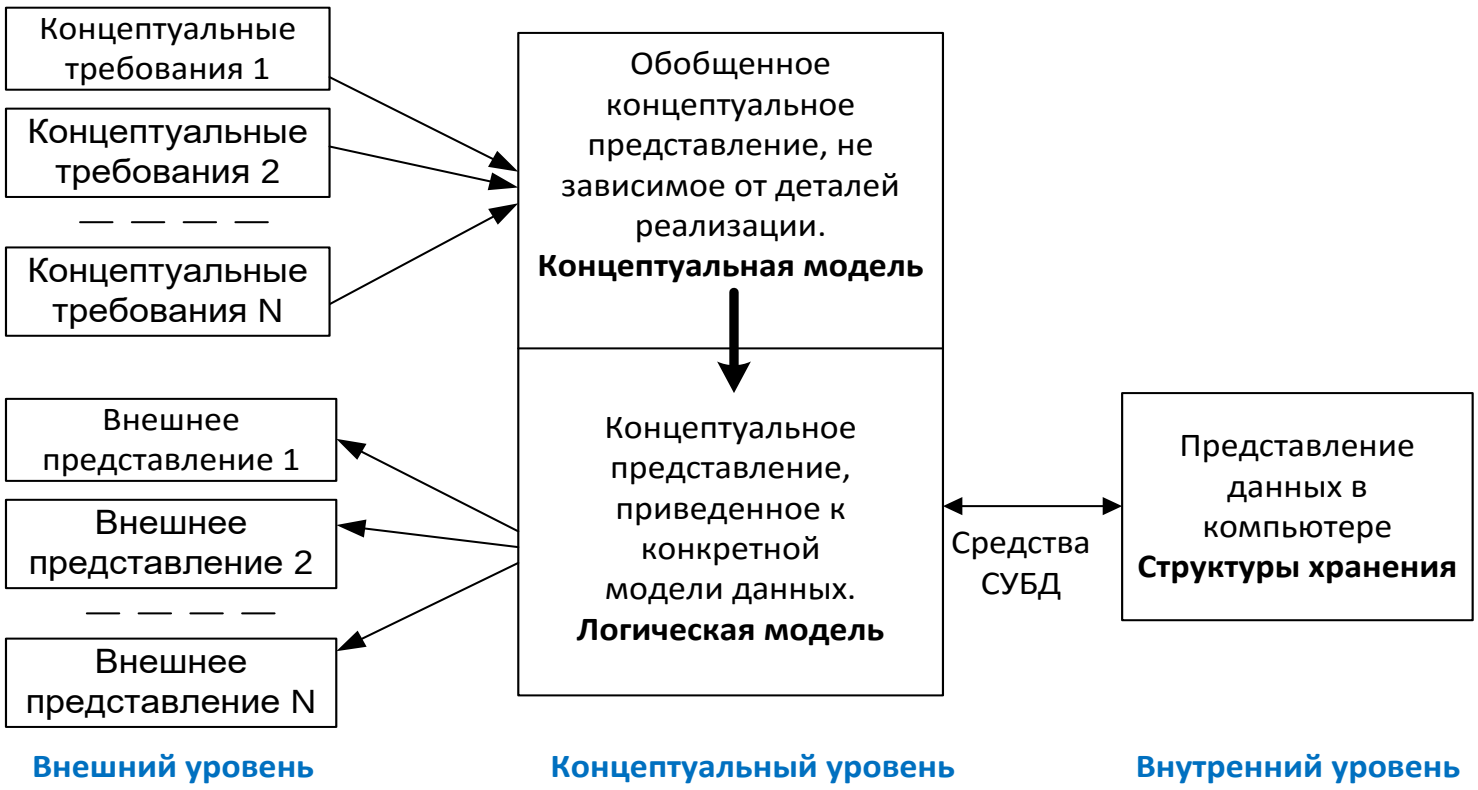
Лекция 6.

Физические модели данных

Трехуровневая архитектура данных ANSI/SPARC

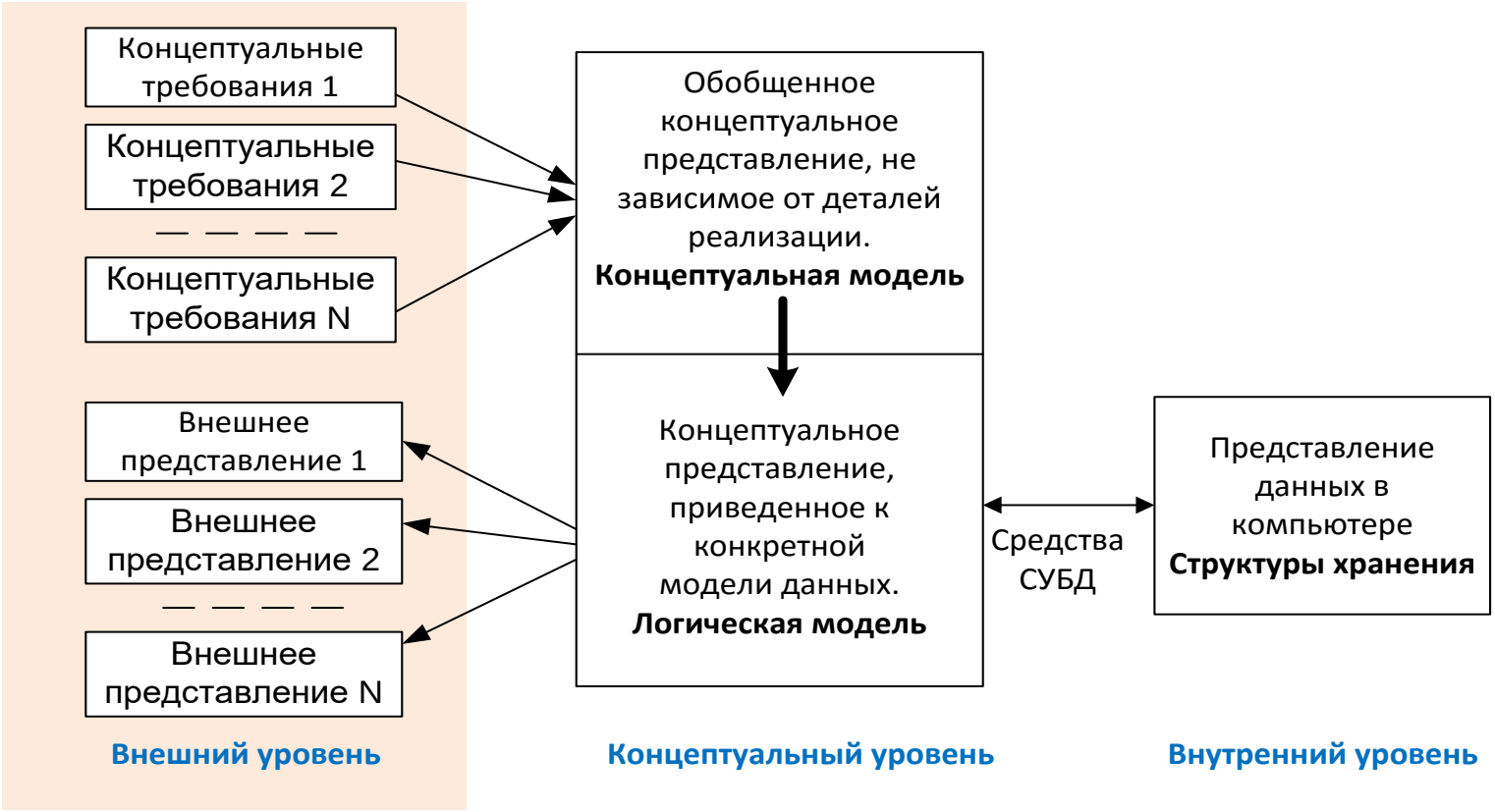


Основная цель - отделение пользовательского представления о данных от их физического представления.

Прикладные программы работают с логической моделью.

Пользователю представляется подмножество логической модели, отражающее его представление о предметной области.

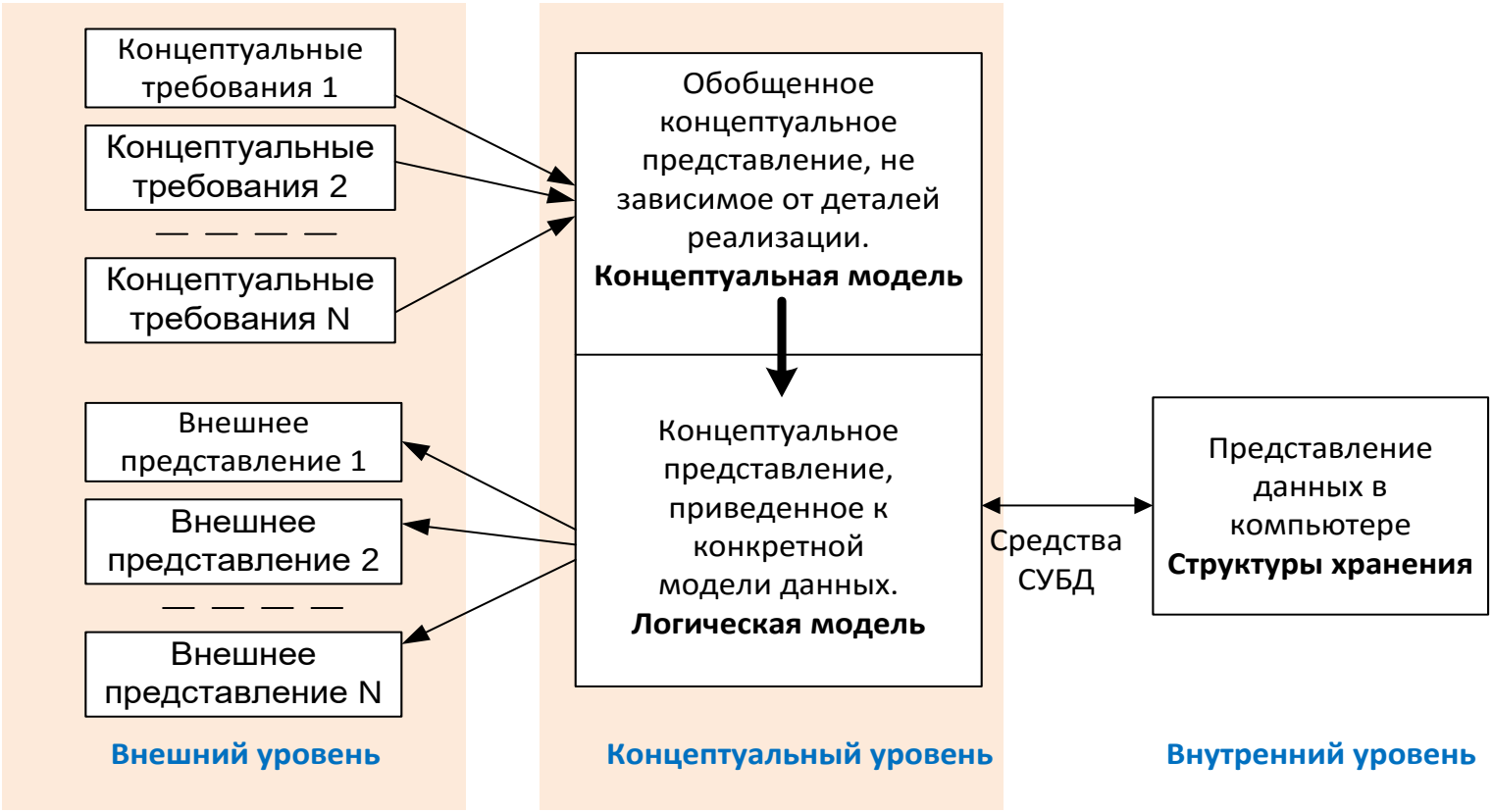
Трехуровневая архитектура данных ANSI/SPARC



Внешнее представление – для специалиста предметной области (пользователя). Работают только с интерфейсом, с элементами управления.

Пользовательский интерфейс должен быть удобным, понятным, связанным с предметной областью.

Трехуровневая архитектура данных ANSI/SPARC

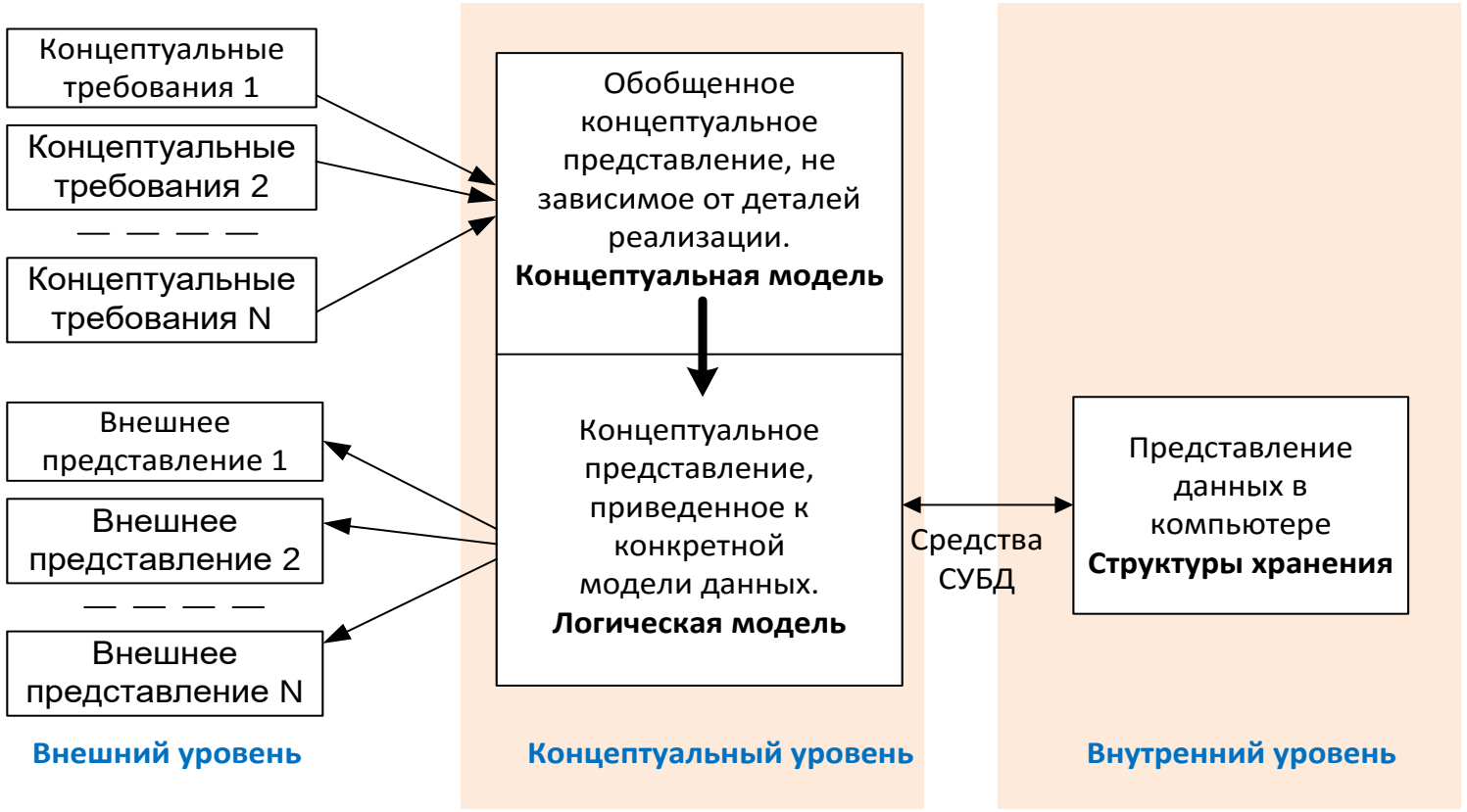


Внешнее представление и логическая модель – для прикладного программиста.

Программисты разрабатывают:

- пользовательский интерфейс и функционал приложения (это связано с предметной областью),
- взаимодействие с логическими структурами данных (абстракция, не связанная с предметной областью).

Трехуровневая архитектура данных ANSI/SPARC



Логическая модель и внутреннее представление – для администратора базы данных.

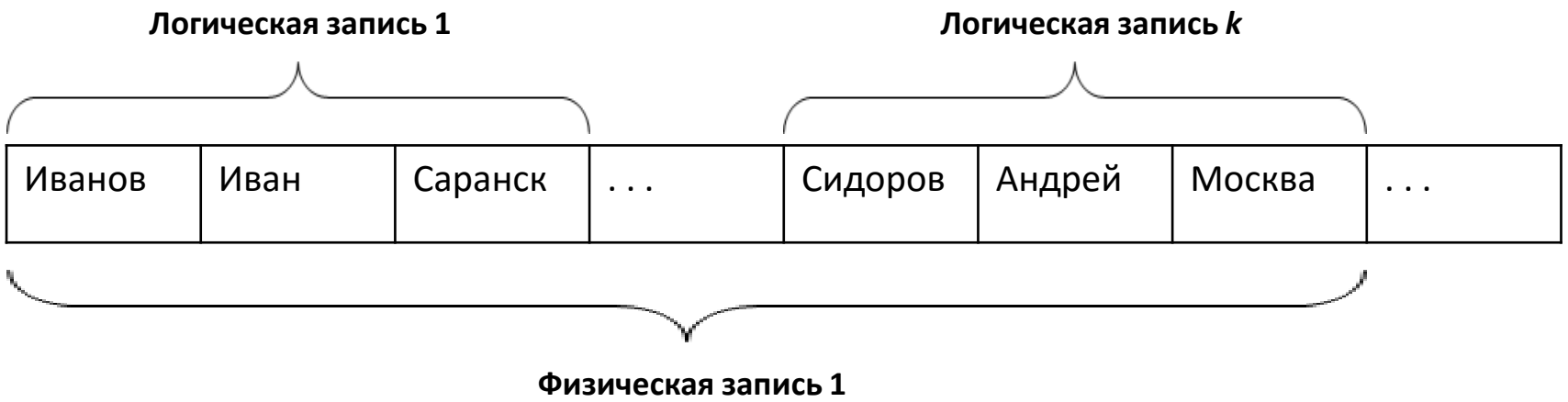
Администратор БД может быть полностью абстрагирован от предметной области.

Физические модели данных

Логическое представление данных

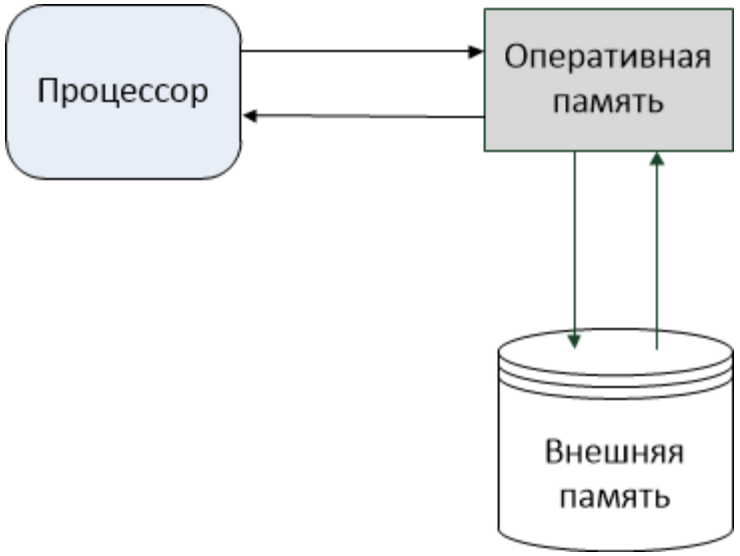
Логическая запись 1	Иванов	Иван	Саранск
Логическая запись 2	Петров	Борис	Пенза
Логическая запись 3	Сидоров	Андрей	Москва
. . .	. .	. . .	. . .
Логическая запись *N*	Яковлев	Петр	Саранск

Физическое размещение записей (в форме определенной структуры)



Физические модели данных

База данных хранится во внешней памяти, а процессор работает только с оперативной памятью.

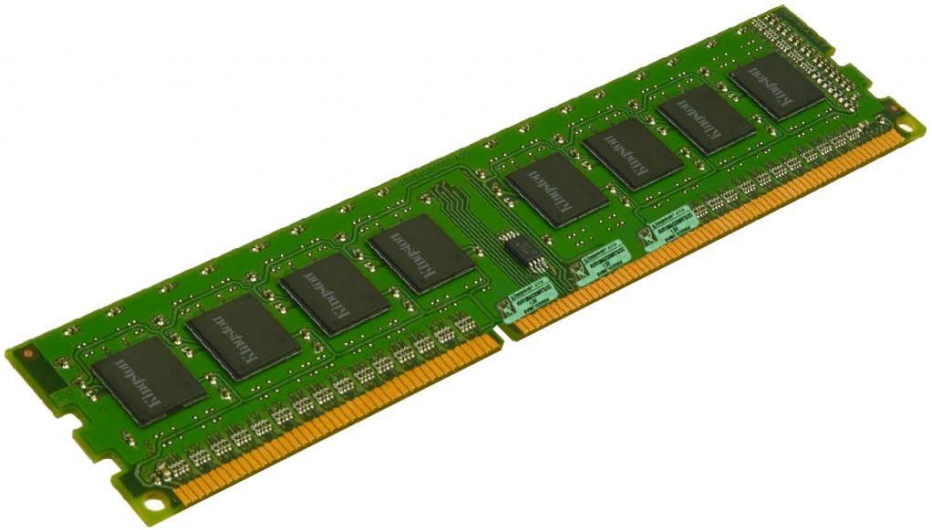


Поэтому организация данных должна учитывать:

- специфику каждого вида памяти,
- способы их взаимодействия.

Оперативная память

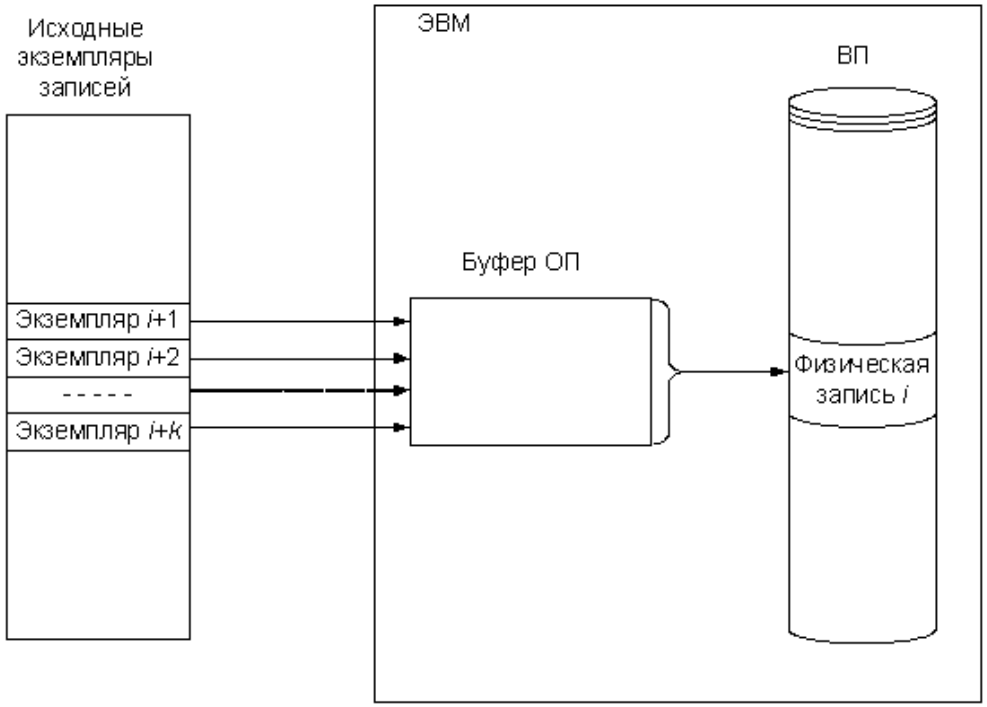
- Единицей памяти является **байт**.
- Память прямоадресуема (каждый байт имеет адрес).
- Процессор выбирает для обработки нужные данные, непосредственно адресуясь к последовательности байтов, содержащих эти данные.



Внешняя память

- Минимальная адресуемая единица – **физическая запись**.
- Для обработки (выделения полей) запись должна быть считана в оперативную память (ОП).
- Записи из внешней памяти в ОП читаются намного медленнее, чем обрабатываются процессором в ОП.
- Организация обмена осуществляется порциями, т.к. невозможно считать сразу всю базу данных.

Обмен между ОП и ВП



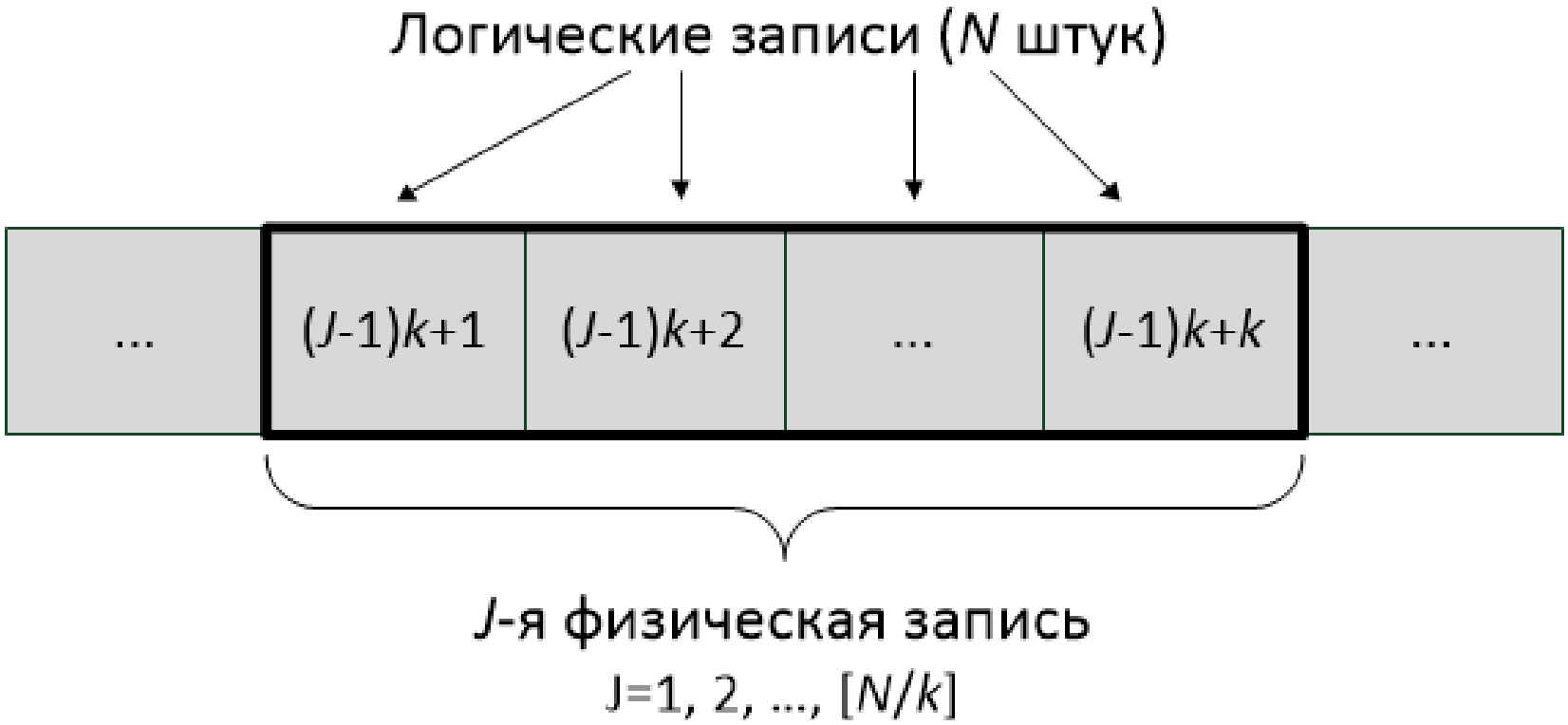
Занесение записей во внешнюю память

Обработка данных, хранящихся во внешней памяти:

- физическая запись (блок) считывается в оперативную память;
- обрабатываются экземпляры логических записей внутри блока (выбираются нужные поля, производится сравнение ключевого поля с заданным значением, осуществляется корректировка полей, выполняются операции удаления, …).

Структуры хранения данных во ВП

Последовательное размещение физических записей



Последовательное размещение данных

Оценим T – среднее число обращений к ВП при разных операциях.

- **Поиск записи по ключу**.

Последовательное размещение данных

Оценим T – среднее число обращений к ВП при разных операциях.

- **Поиск записи по ключу**. **$T=(1+[N/k])/2=O(N)$**

Последовательное размещение данных

Оценим T – среднее число обращений к ВП при разных операциях.

- **Поиск записи по ключу**. **$T=(1+[N/k])/2=O(N)$**
  - **Чтение записи**. $T=(1+[N/k])/2=O(N)$
  - **Корректировка записи**. $T=(1+[N/k])/2+1=O(N)$
  - **Удаление записи.** $T=(1+[N/k])/2+1=O(N)$

Последовательное размещение данных

Оценим T – среднее число обращений к ВП при разных операциях.

- **Поиск записи по ключу**. **$T=(1+[N/k])/2=O(N)$**
  - **Чтение записи**. $T=(1+[N/k])/2=O(N)$
  - **Корректировка записи**. $T=(1+[N/k])/2+1=O(N)$
  - **Удаление записи.** $T=(1+[N/k])/2+1=O(N)$
- **Добавление записи:**
  - **в конец таблицы**. 1 или 2 = O(1)
  - **в произвольное место**. $3+[N/k]=O(N)$

Последовательное размещение данных

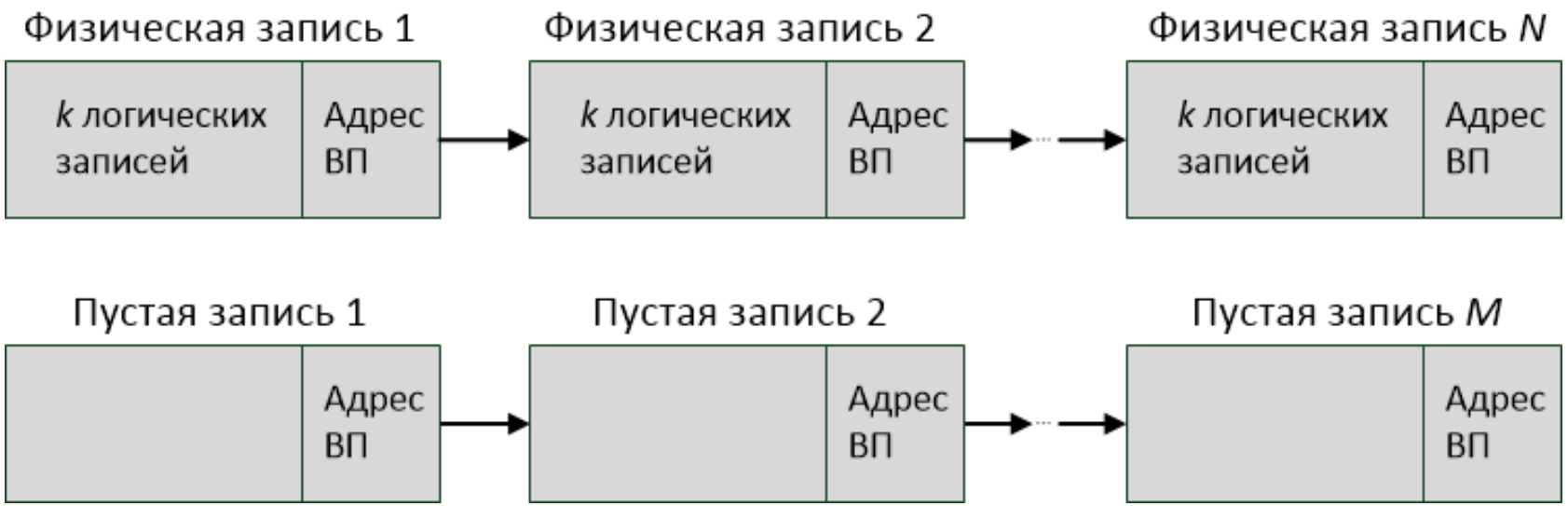
Оценим T – среднее число обращений к ВП при разных операциях.

- **Поиск записи по ключу**. **T=(1+[*N*/*k*])/2=O(*N*)**
  - **Чтение записи**. T=(1+[*N*/*k*])/2=O(*N*)
  - **Корректировка записи**. T=(1+[*N*/*k*])/2+1=O(*N*)
  - **Удаление записи.** T=(1+[*N*/*k*])/2+1=O(*N*)
- **Добавление записи:**
  - **в конец таблицы**. 1 или 2 = O(1)
  - **в произвольное место**. **3+[*N*/*k*]=O(*N*)**

Проблемные операции: **поиск и вставка записи**

Структуры хранения данных во ВП

Размещение физических записей в виде списка



Размещение данных в виде списка

Оценим T – среднее число обращений к ВП.

- **Добавление записи в произвольное место**.

Размещение данных в виде списка

Оценим T – среднее число обращений к ВП.

- **Добавление записи в произвольное место**. **З=O(1)**

Размещение данных в виде списка

Оценим T – среднее число обращений к ВП.

- **Добавление записи в произвольное место**. **З=О(1)**
- **Поиск записи по ключу**.
  - **Чтение записи**.
  - **Корректировка записи**.
  - **Удаление записи**.

Размещение данных в виде списка

Оценим T – среднее число обращений к ВП.

- **Добавление записи в произвольное место**. **З=O(1)**
- **Поиск записи по ключу**. **T=(1+[*N*/*k*])/2=O(*N*)**
  - **Чтение записи**. T=(1+[*N*/*k*])/2=O(*N*)
  - **Корректировка записи**. T=(1+[*N*/*k*])/2+1=O(*N*)
  - **Удаление записи**. T=(1+[*N*/*k*])/2+1=O(*N*)

Проблемная операция: **поиск записи**

Индексирование

= Логическое упорядочение физических записей

Основные данные

N	Фамилия (PK)	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
3	Иванова	Ольга	19
4	Антонов	Борис	43
5	Ливанов	Антон	25
6	Козлов	Сергей	26

Индекс

Фамилия (FK)	Номер записи
Антонов	4
Иванова	3
Козлов	6
Ливанов	5
Петров	1
Сидоров	2

Индексирование

= Логическое упорядочение физических записей

Основные данные

N	Фамилия (PK)	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
3	Иванова	Ольга	19
4	Антонов	Борис	43
5	Ливанов	Антон	25
6	Козлов	Сергей	26

Индекс

Фамилия (FK)	Номер записи
Антонов	4
Иванова	3
Козлов	6
Ливанов	5
Петров	1
Сидоров	2

Поиск в индексированной таблице:

1. Дихотомический поиск в индексе
2. Определение номера записи в основной таблице
3. Считывание нужной записи в основной таблице

Индексированная таблица

Важно: в идеале индексный файл имеет малый размер и полностью считывается в оперативную память за один раз.

Для больших индексов актуальна проблема сжатия ключа.

Индексированная таблица

Оценим T – среднее число обращений к внешней памяти.

Индексированная таблица

Оценим T – среднее число обращений к внешней памяти.

- **Поиск записи по ключу**. **2=O(1)**
  - **Чтение записи**. T=2=O(1)
  - **Корректировка записи**. T=3=O(1)
  - **Удаление записи.** T=3=O(1)
- **Добавление записи**. **3=O(1)**

Структуры хранения данных во ВП

Размещение физических записей в виде B-дерева

Сбалансированное дерево - путь от корня до любого листа одинаков. На базе многоуровневого индекса (индекс над индексом)

Структуры хранения данных во ВП

Размещение физических записей в виде B-дерева

Сбалансированное дерево - путь от корня до любого листа одинаков. На базе многоуровневого индекса (индекс над индексом)

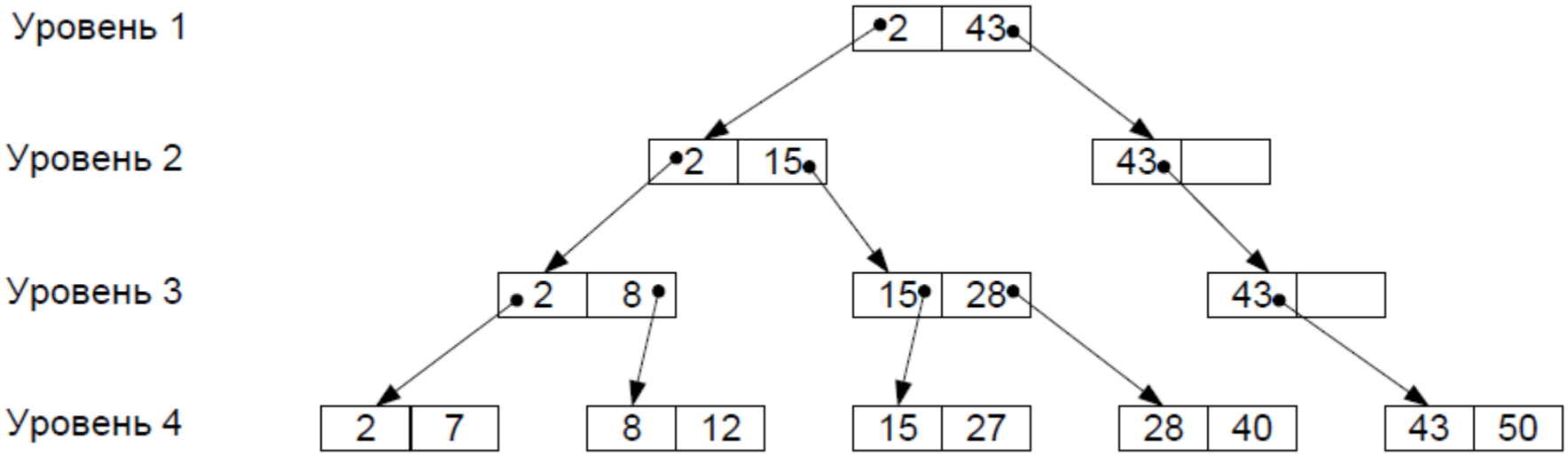
Построение:

1. Записи в основном файле упорядочиваются по значению первичного ключа и объединяются в блоки по *k* записей.
2. За значение ключа блока принимается минимальное значение ключа у записей, входящих в блок. Последовательность блоков = последний уровень B-дерева.
3. Строится индекс предыдущего уровня. Записи этого уровня содержат значение ключа блока следующего уровня и указатель-адрес связи соответствующего блока.
4. Записи построенного уровня также объединяются в блоки по *k* записей.
5. Продолжаем построение индексов более высокого уровня, пока количество записей индекса на определенном уровне будет не более *k*.

Число уровней $\approx log_k N$

B-дерево

Пример. Пусть ключи записей принимают значения 2, 7, 8, 12, 15, 27, 28, 40, 43, 50. В блок будем объединять по 2 экземпляра записей (*k*=2).

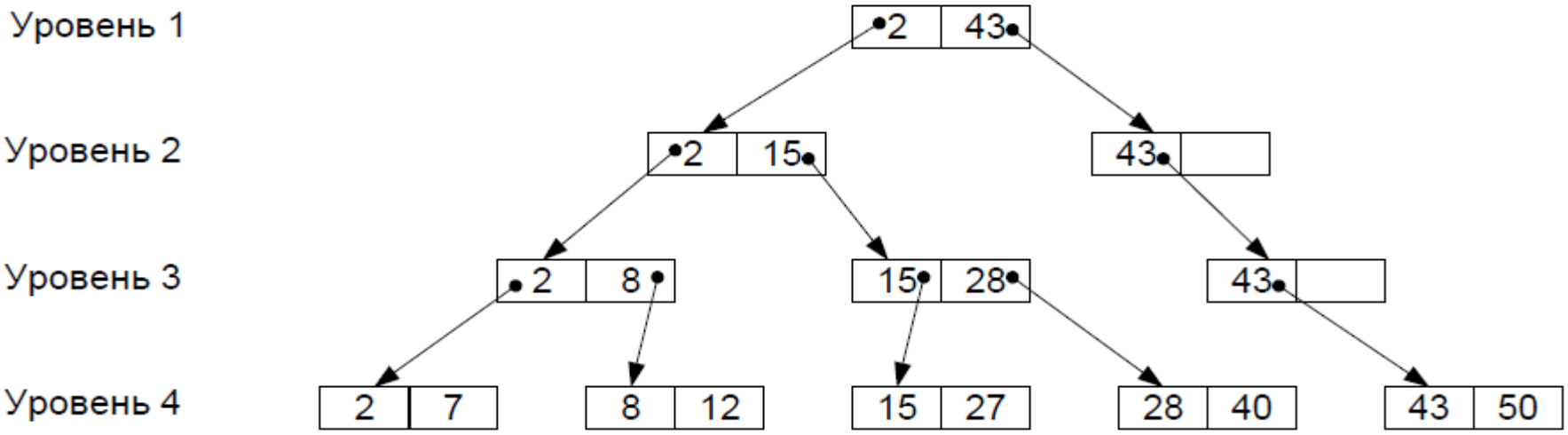


B-дерево

Поиск и чтение записи с заданным значением M ключа

1. Читаем верхний индекс.
2. Сравниваем число M со значением ключа записей индекса, начиная с большего значения. Если M больше или равно значению ключа очередной записи индекса, то по адресу связи, указанному в текущей записи, читаем блок записей индекса следующего уровня.
3. Повторяем процесс, пока не дойдем до последнего уровня.

Если все блоки расположены во внешней памяти, то $T=O(\log_2 N)$



B-дерево

Корректировка записи

1. Нужная запись находится и считывается.
2. Если корректируется не ключ записи, то измененная запись заносится на свое место. Если меняется значение ключа, то старая запись удаляется (в блоке появляется «пустая» запись), а измененная запись заносится так же, как вновь добавляемая.

Удаление записи

1. Нужная запись находится.
2. В соответствующий блок на место этой записи заносится «пустая» запись.

Если все блоки расположены во внешней памяти, то $T=O(\log_2 N)$

B-дерево

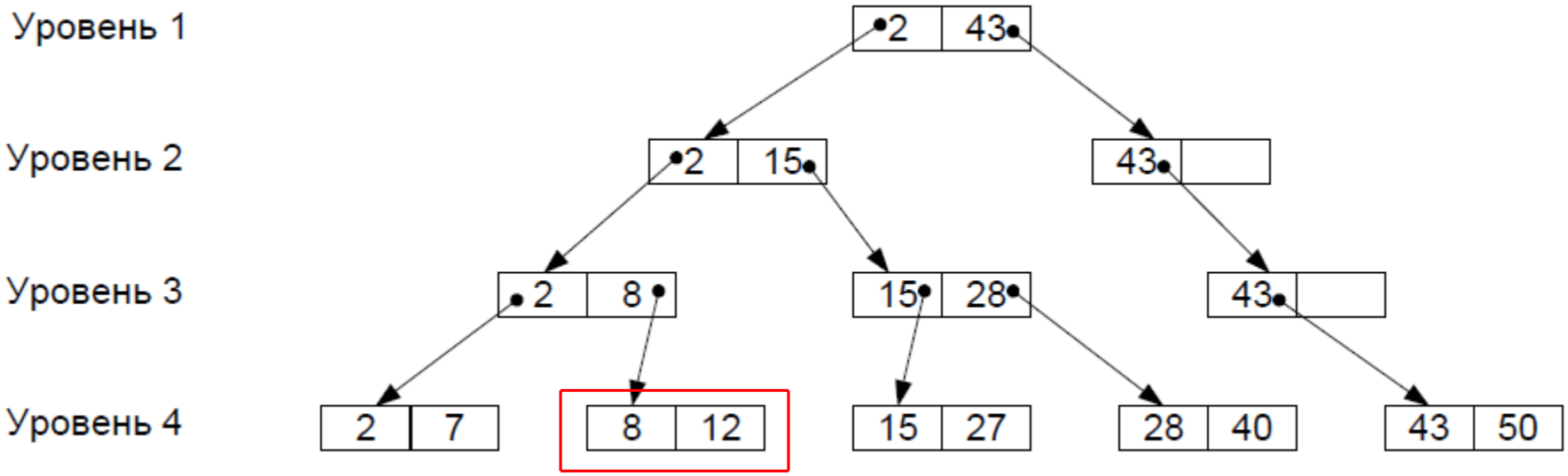
Добавление записи с ключом M

1. Ищется блок, куда нужно вставить запись (аналогично поиску записи по ключу). Если в блоке есть «пустая» запись, то добавляем в эту запись.
2. Иначе блок делится на два блока. В первый из них заносится $[k/2]$ записей, во второй заносятся остальные.
3. Определяются новые значения ключа этих блоков.
4. Добавляемая запись заносится в тот блок, значение ключа которого меньше ключа M.
5. Для нового блока с новым значением ключа формируется соответствующая новая запись в индексе на предыдущем уровне. Процедура добавления такой записи аналогична п. 1-4.
6. Аналогичные операции проделываются на всех уровнях до первого. Возможен вариант, когда придется делить блок самого верхнего уровня и формировать еще один уровень дерева.

B-дерево

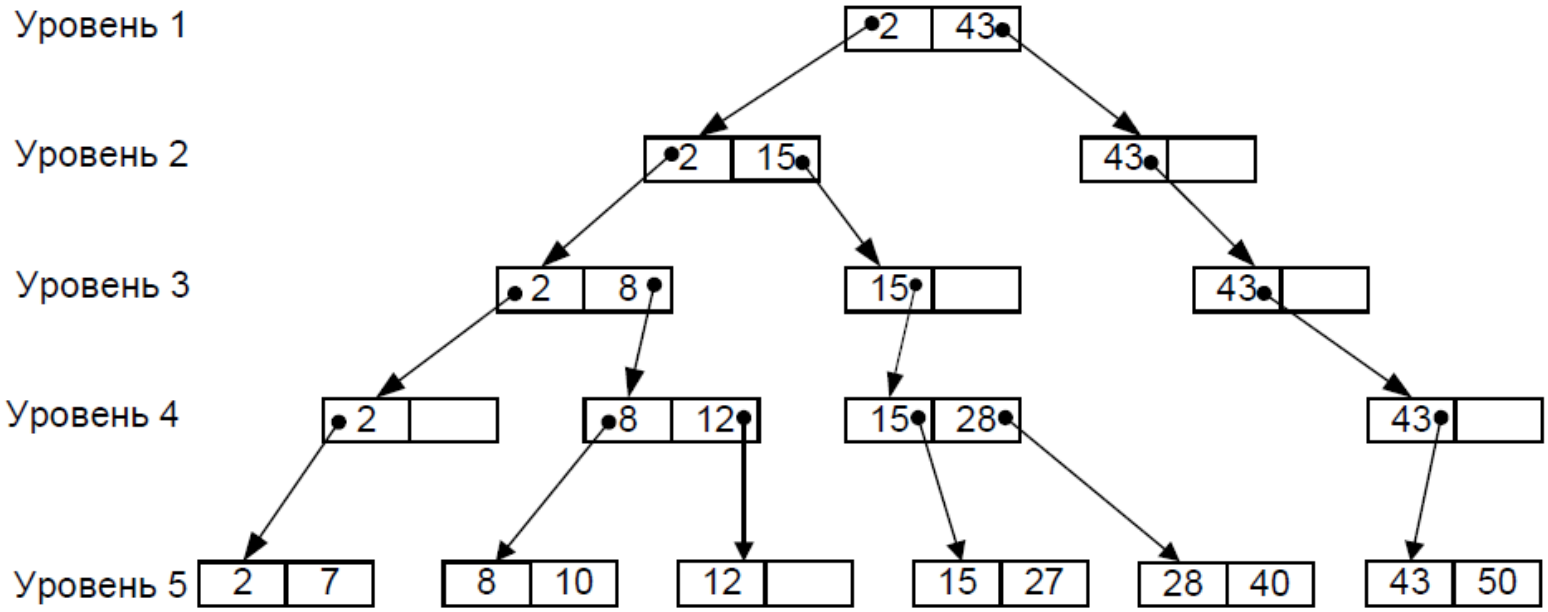
Пример. Добавим запись с ключом 10. Пусть ключи записей принимают значения 2, 7, 8, 12, 15, 27, 28, 40, 43, 50. В блок будем объединять по 2 экземпляра записей (*k*=2).

Добавим запись с ключом 10.



B-дерево

После добавления записи:



После нескольких вставок записей блоки будут заполнены, в среднем, наполовину и процедура добавления записи станет менее трудоемкой.