Материалы презентации предназначены для размещения только для использования студентами кафедры «Компьютерные системы и технологии» НИЯУ МИФИ дневного и вечернего отделений, изучающими курс «Программирование (Алгоритмы и структуры данных)».

Публикация (размещение) данных материалов полностью или частично в электронном или печатном виде в любых других открытых или закрытых изданиях (ресурсах), а также использование их для целей, не связанных с учебным процессом в рамках курса «Программирование (Алгоритмы и структуры данных)» кафедры «КСиТ» НИЯУ МИФИ, без письменного разрешения автора запрещена.

7. B-деревья

Шустова Л.И.

Варианты деревьев

1970 г. Джон Хопкрофт (John E. Hopcroft)

– 2-3 деревья: каждый внутренний узел имеет 2 или 3 дочерних узла, все листья – на одном уровне

2-3-4 деревья: каждый внутренний узел имеет 2, 3 или 4 дочерних узла

1972 г. Р. Байер (R. Bayer) и Э. Мак-Крэйт (E.M. McCreight) – B деревья

B-деревья

Сбалансированные деревья поиска, созданные специально для работы с внешней памятью

Сильно ветвящиеся (многоходовые) деревья: каждый узел имеет n ($n > 1$) дочерних узлов и $n - 1$ ключей

Значения ключей в узле упорядочены по возрастанию: $k_1 \le k_2 \le \ldots \le k_{n-1}$

Определение B-дерева

B-дерево степени t:

- Каждый узел, кроме корня и листьев, имеет минимум t, максимум $2t$ дочерних узлов
- Каждый узел, кроме корня, содержит минимум $t - 1$, максимум $2t - 1$ ключей; узел заполнен, если он имеет ровно $2t - 1$ ключей
- Корень имеет минимум *один* ключ и *два* дочерних узла
- Все листья расположены на одном и том же расстоянии от корня, равном высоте дерева h

Структура узла

- Текущее количество ключей в узле
- Значения ключей и сопутствующая ключам информация
- Указатели на дочерние узлы (для листьев здесь *NULL*)
- Указатель на родительский узел

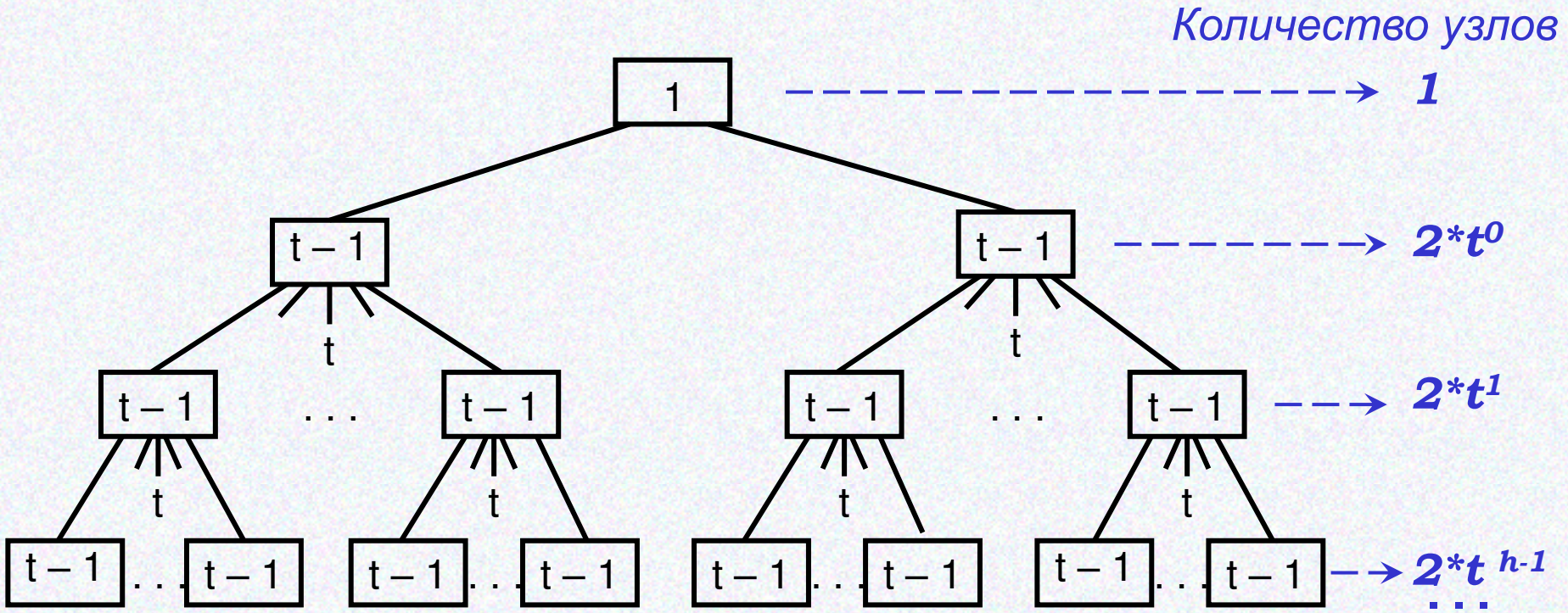
Простейшее B-дерево степени 2 – ***2-3-4 дерево***

Структура узла

```
const int T = …; – степень дерева
struct Node {
    int n; – количество ключей в узле
    int key[2 * T – 1]; – значения ключей;
                          key[0] ≤ key[1] ≤ …
    struct Node *ptr[2 * T]; – указатели на дочерние
                               узлы
    struct Node *parent; – указатель на
                           родительский узел
};
```

Пример B-дерева

B-дерево степени t с минимально возможным количеством ключей



Высота B-дерева

Теорема

Высота B-дерева с $n > 1$ узлами и минимальной степенью $t \geq 2$ не превышает $log_t((n + 1) / 2)$

Минимальное количество ключей:

$$1 + (2t^0 + 2t^1 + 2t^2 + \ldots + 2t^{h-1})(t - 1) = 1 + 2(t^h - 1);$$

$$n \geq 2t^h - 1$$

Поиск в B-дереве

Результат успешного поиска – указатель на узел дерева и положение искомого ключа в данном узле

Результат не успешного поиска – $NULL$ значение указателя

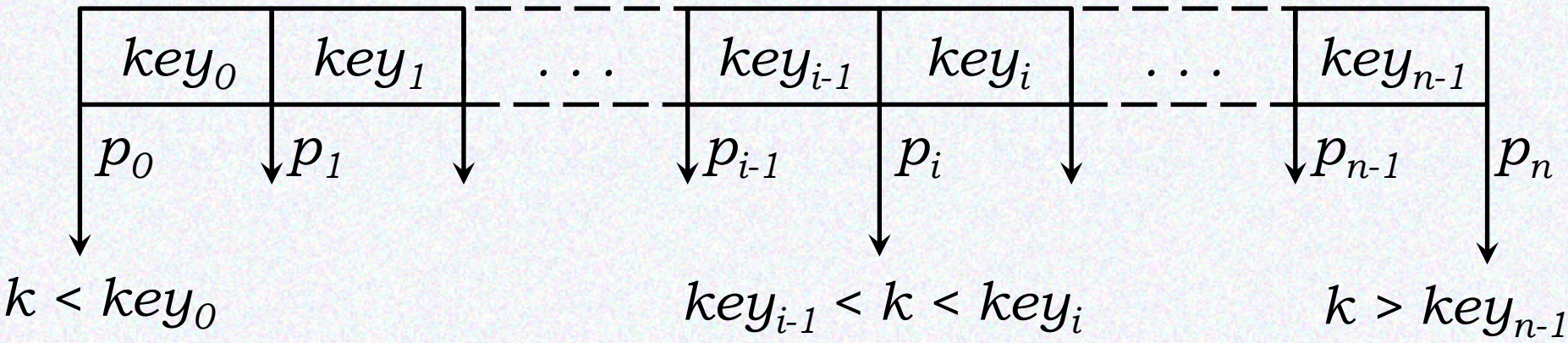
Обозначения:

k – искомый ключ

x – указатель на корень дерева

Поиск в B-дереве

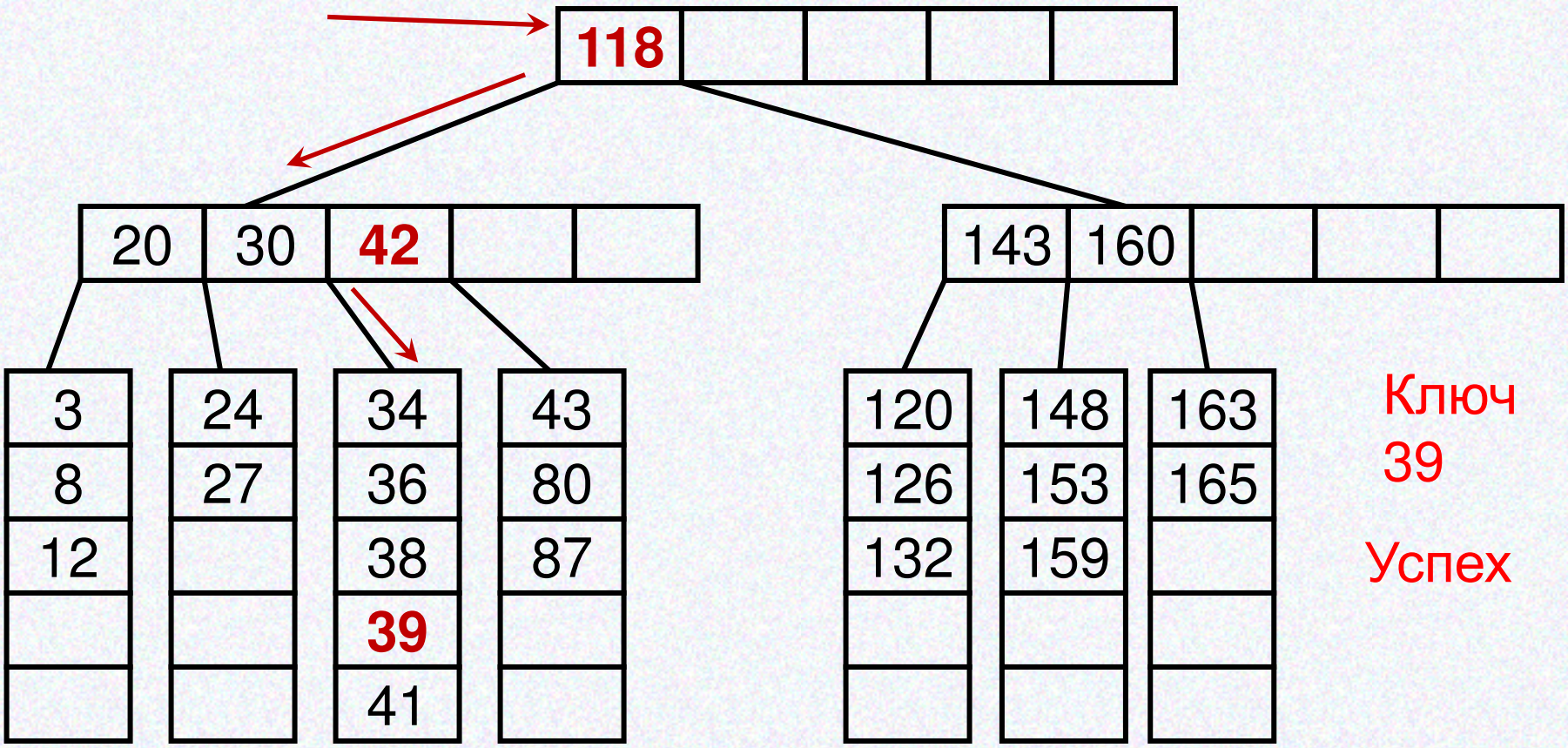
$key_0 < key_1 < \ldots < key_{n-1}$




```
for(i = 0; i < n && k > key[i]; ++i)
    ;
```

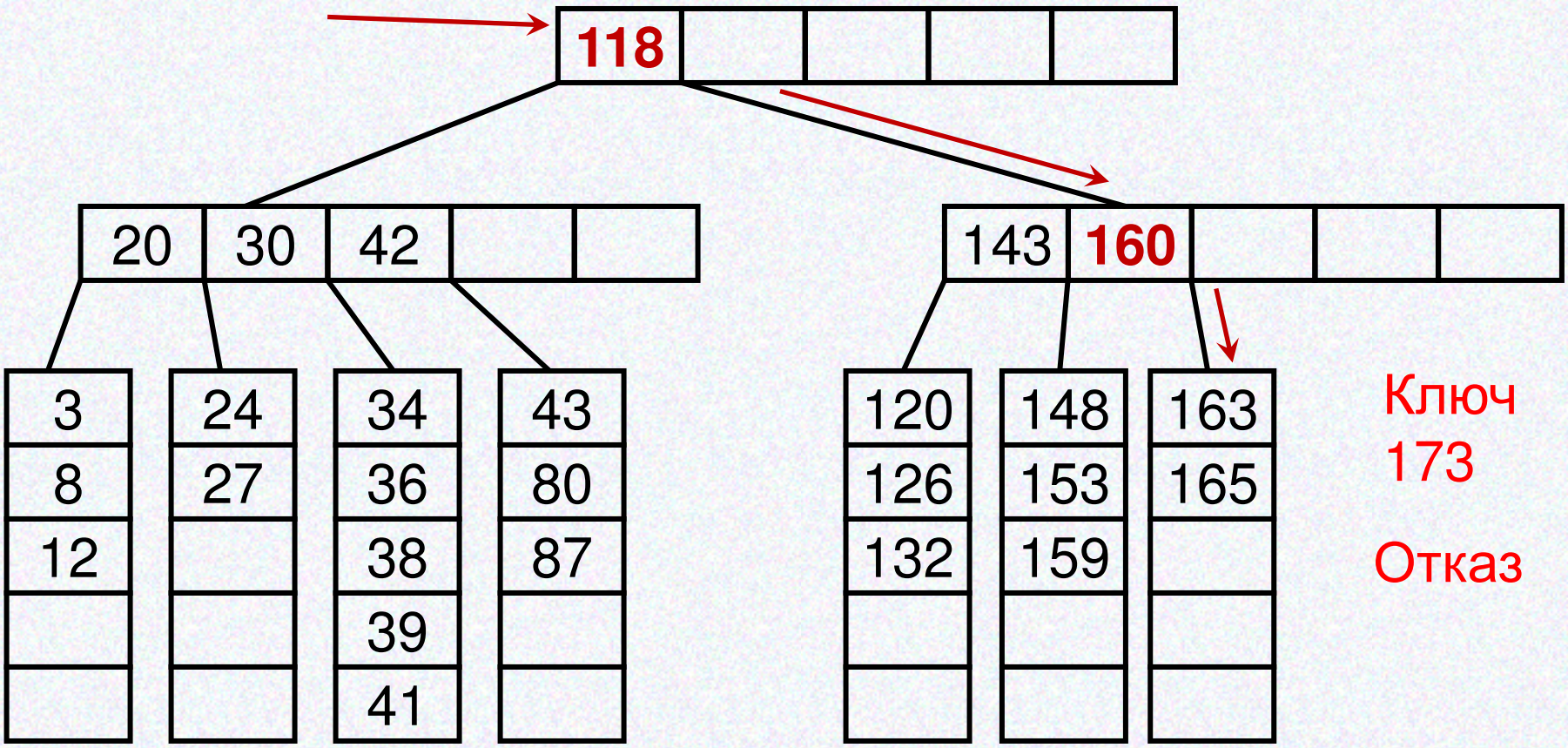
Поиск в B-дереве

B-дерево степени 3

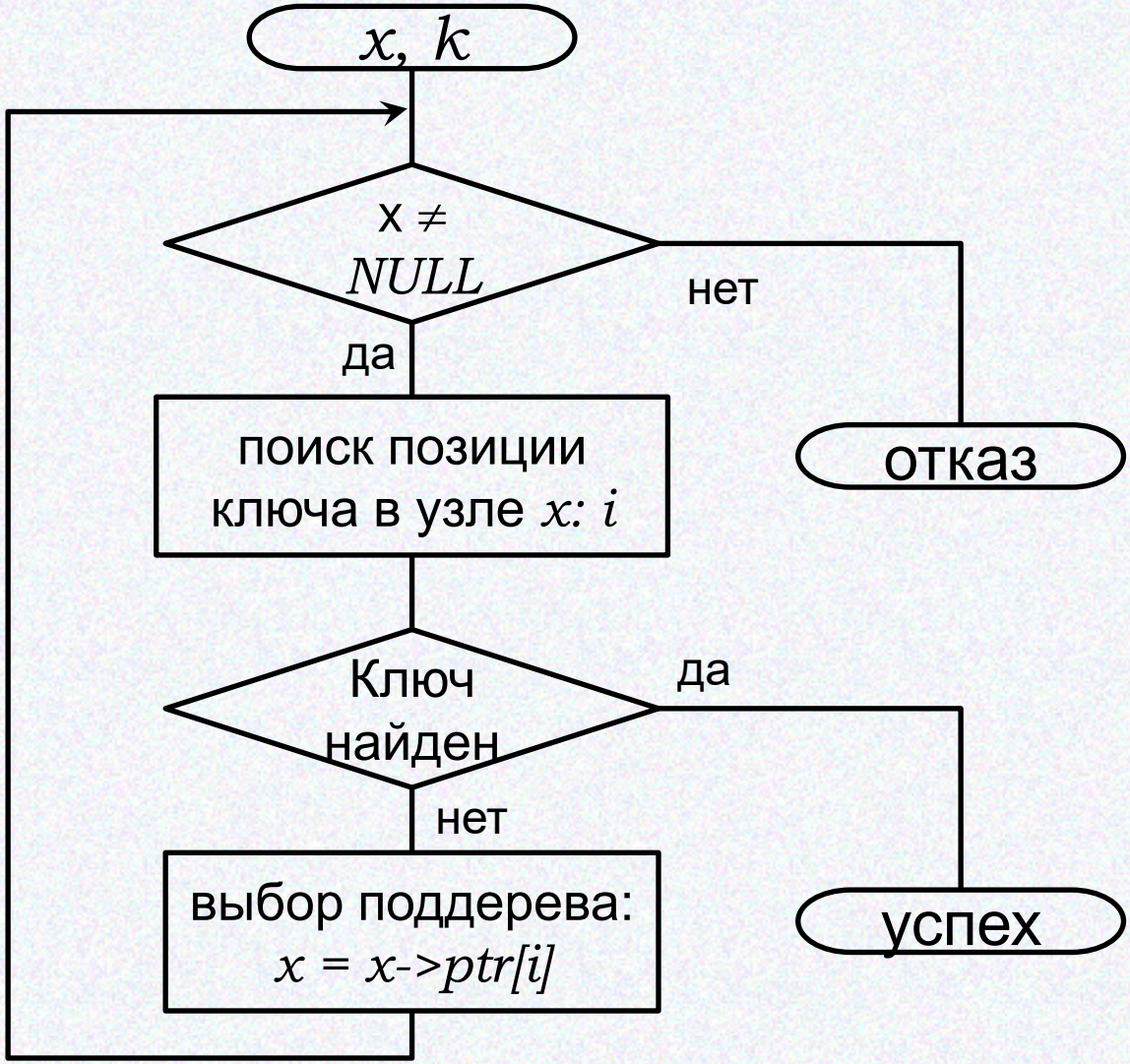


Поиск в B-дереве

B-дерево степени 3



B_Tree_Search(x, k)



Вставка в B-дерево

Пустое B-дерево – B-дерево с единственным листом, в котором нет ключей

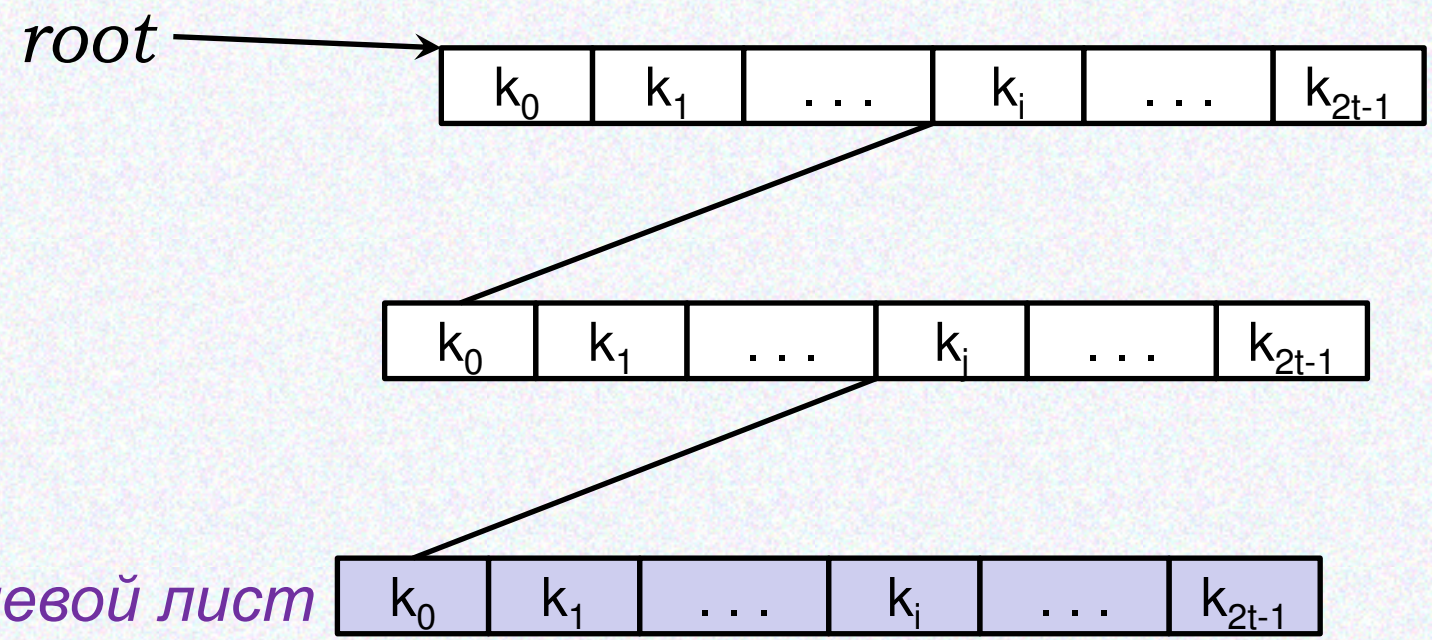
Новый элемент вставляется всегда в лист.

Если лист заполнен – он разбивается на два листа, и медианный ключ перемещается в родительский узел (*разбиение листа*)

При создании новых узлов дерева все указатели устанавливаются в NULL

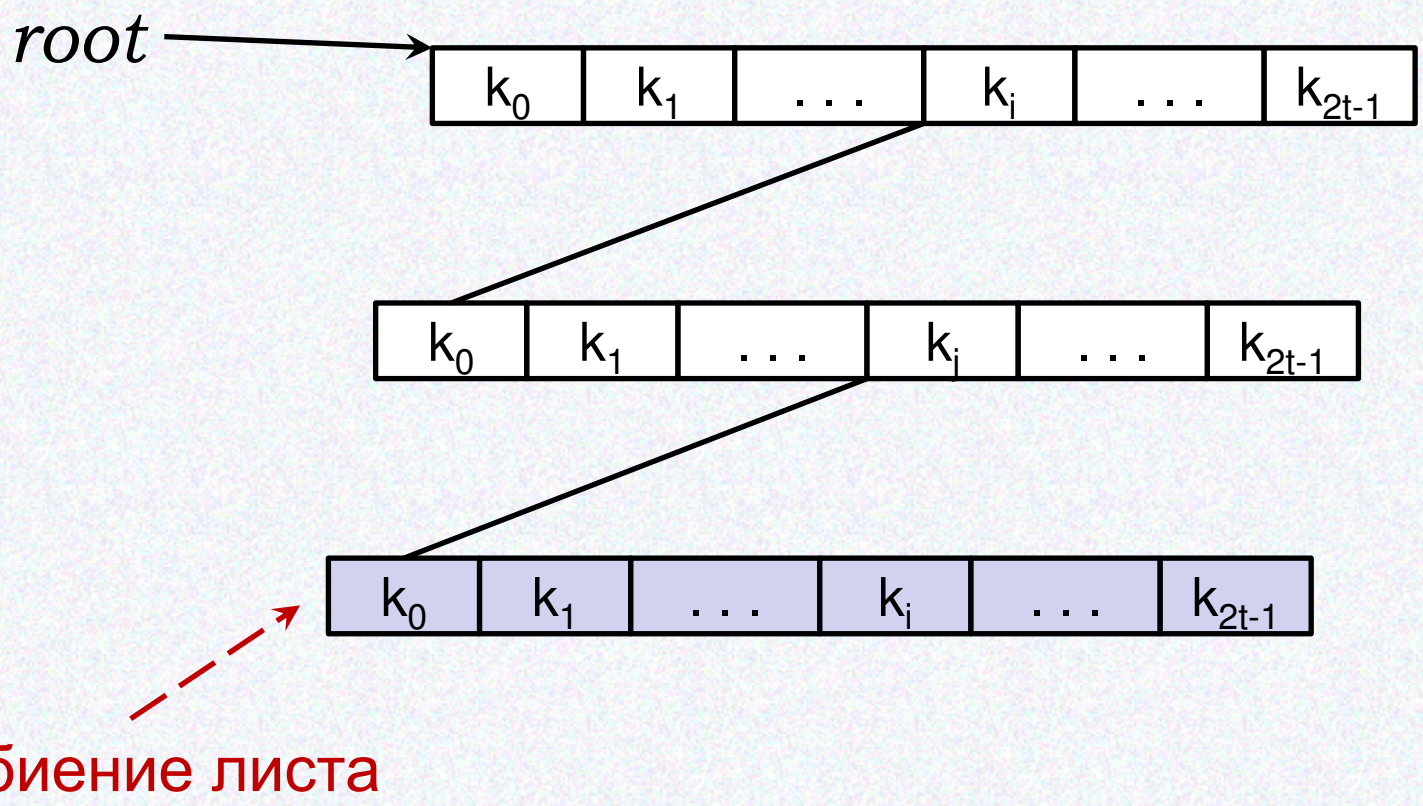
Вставка в B-дерево

B-дерево степени t



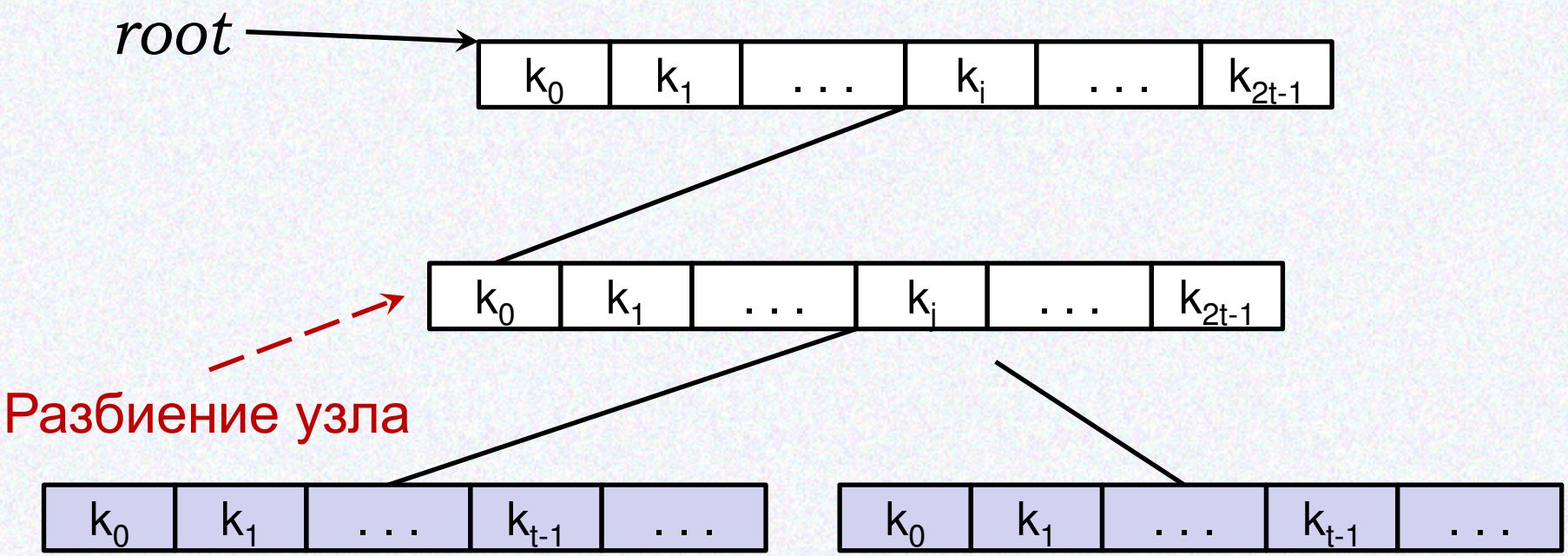
Вставка в B-дерево

B-дерево степени t



Вставка в B-дерево

B-дерево степени t



Вставка в B-дерево

Поиск целевого узла – листа (x)

while узел x полон {

m = медианный ключ из узла x

разбиение (splitting) узла x и вставка ключа k^* в соответствующий узел

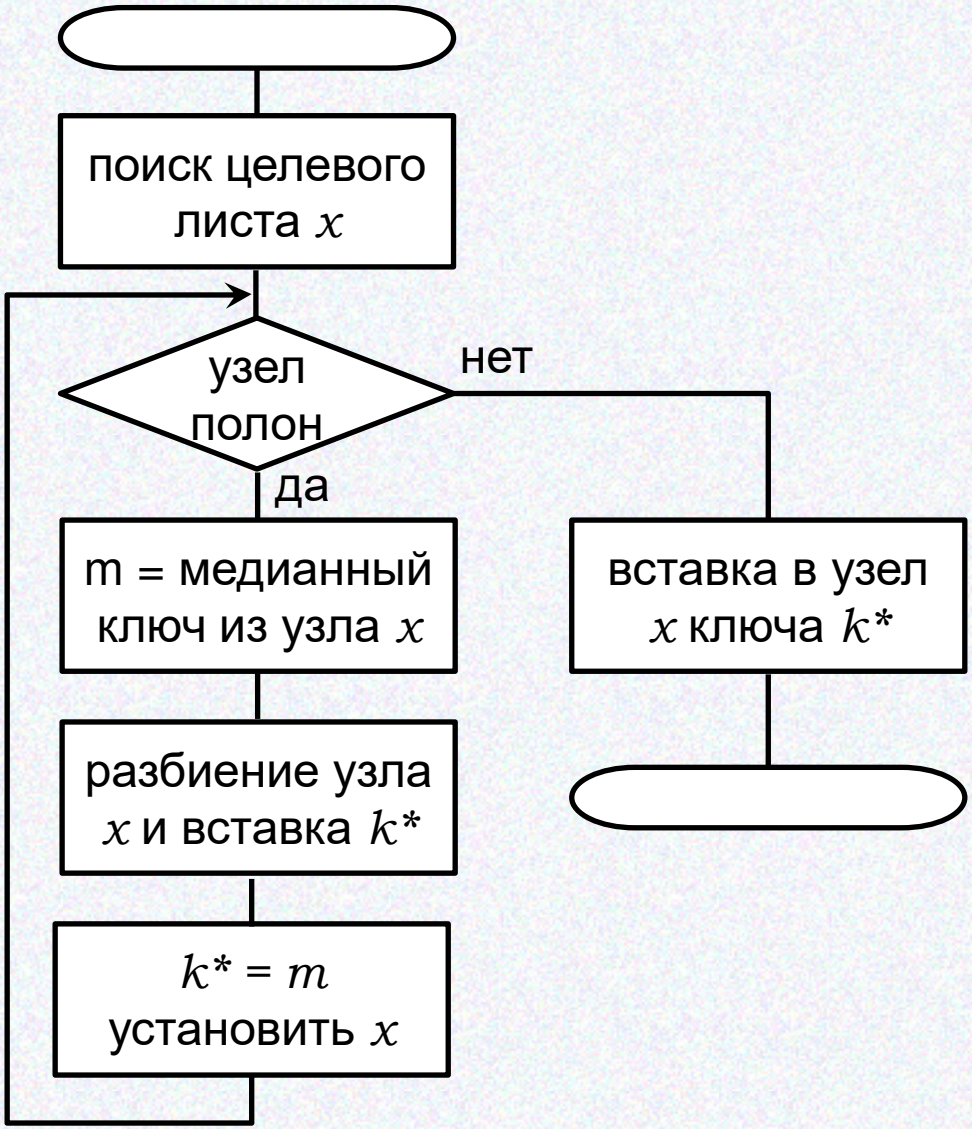
$k^* = m$

x = родительский узел

}

Вставка в узел x нового ключа k^*

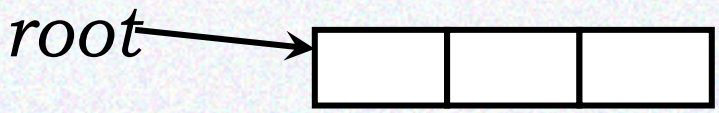
Вставка в B-дерево



Пример: вставка элементов

B-дерево порядка 2

B-дерево пусто

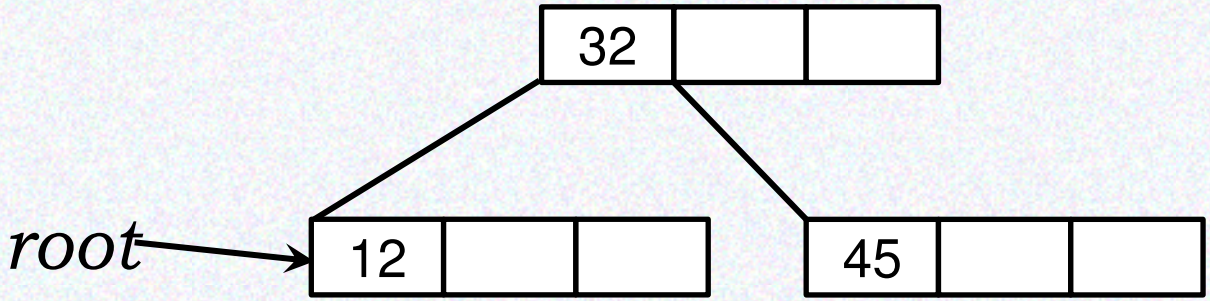


Последовательно вставляются ключи 12, 45 и 32

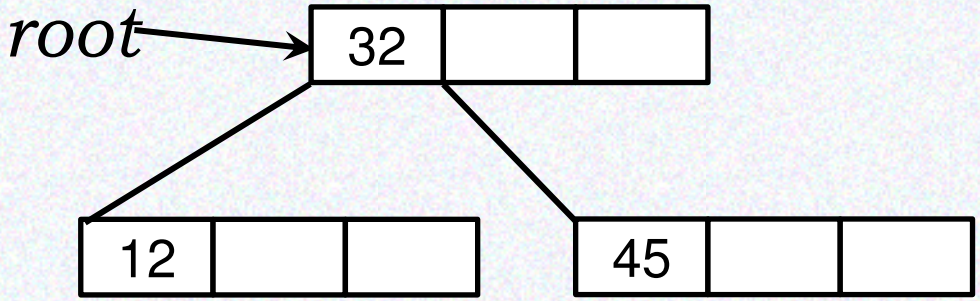
Вставка ключа 20

Пример: вставка элементов

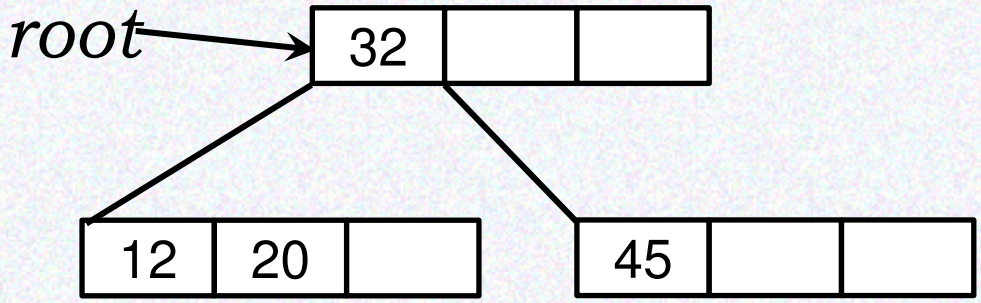
Разбиение листа



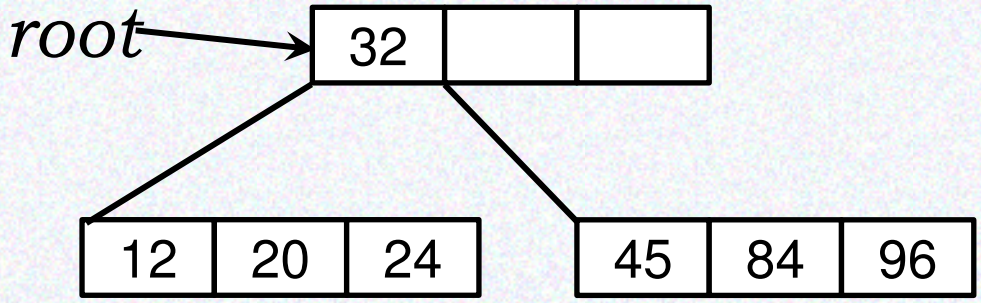
Вставка ключа 20



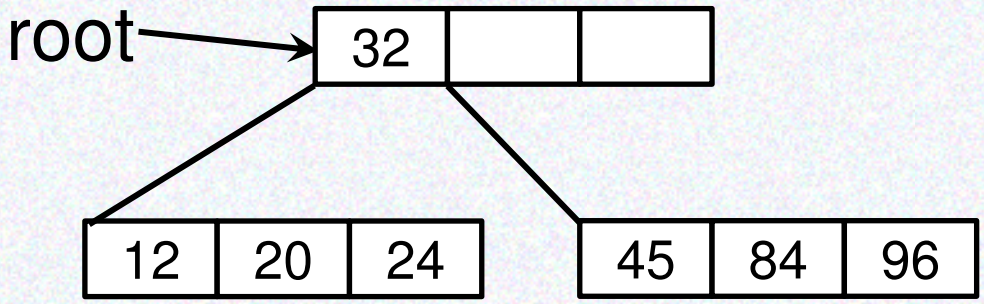
Пример: вставка элементов



Вставляются ключи 24, 84 и 96

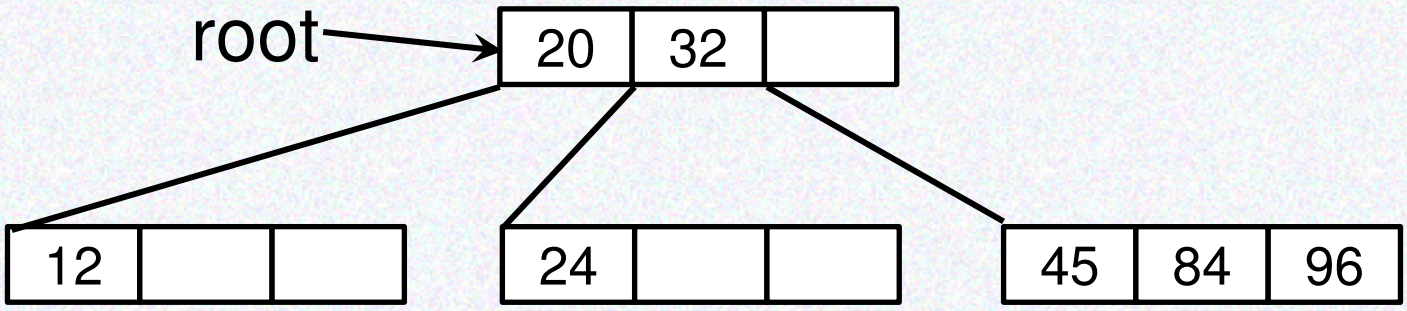


Пример: вставка элементов

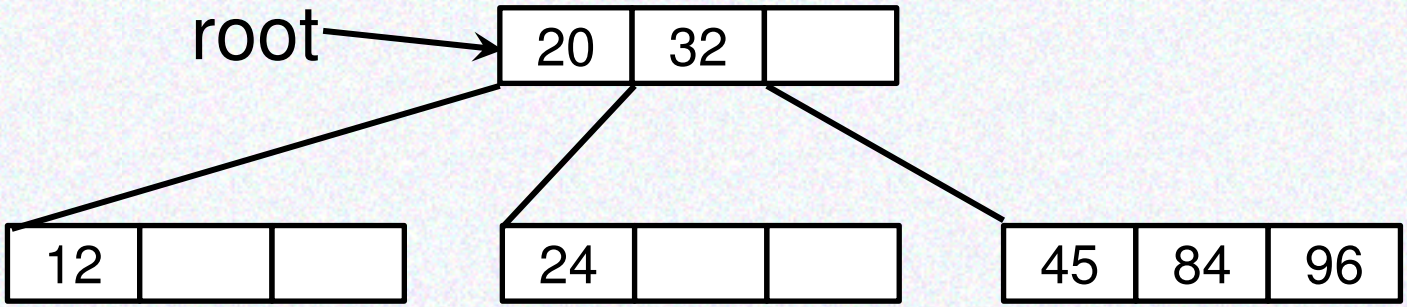


Вставка ключа 29

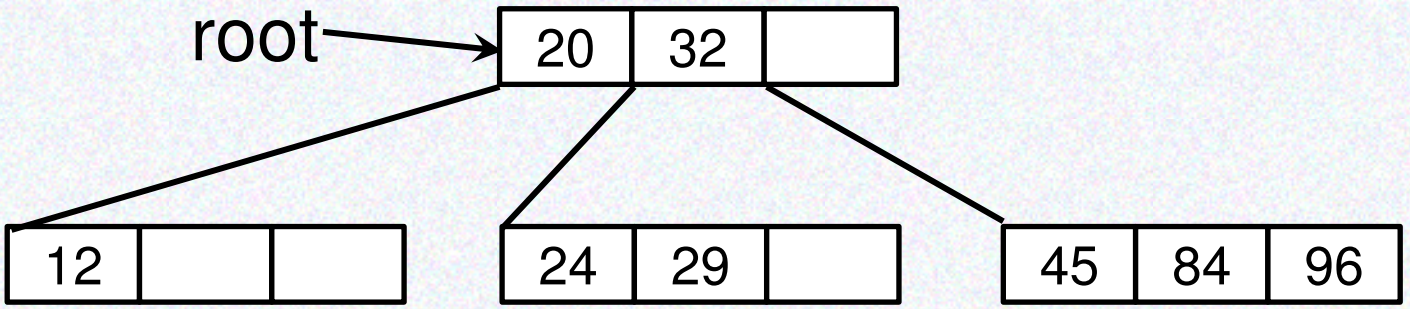
Разбиение листа



Пример: вставка элементов

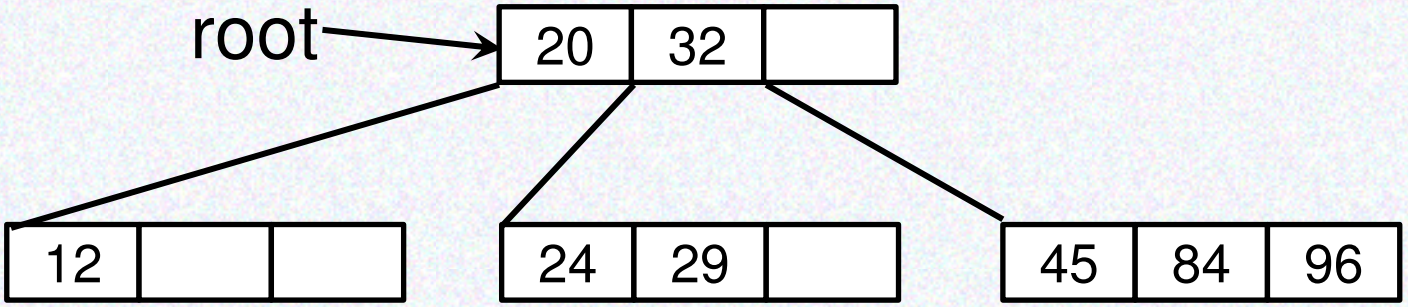


Вставка ключа

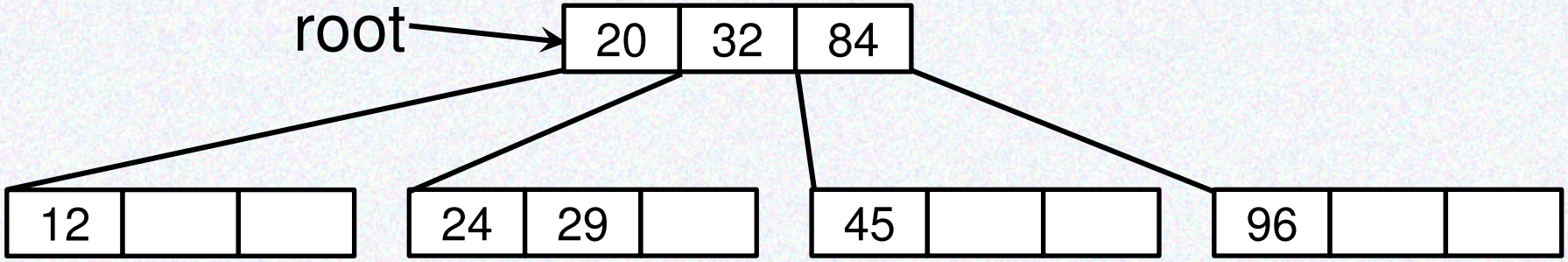


Пример: вставка элементов

Вставка ключа 35

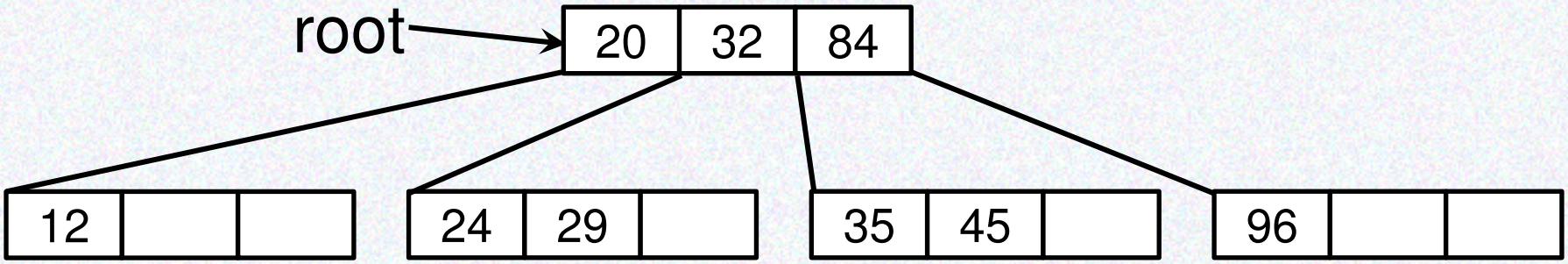


Разбиение листа



Вставка ключа

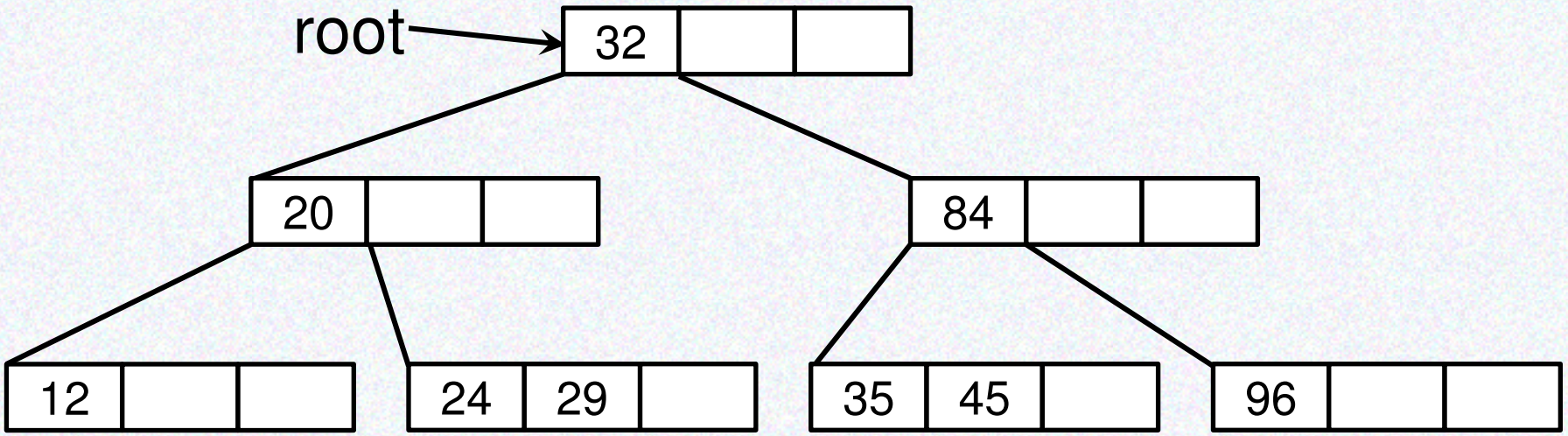
Пример: вставка элементов



Вставка ключа 50

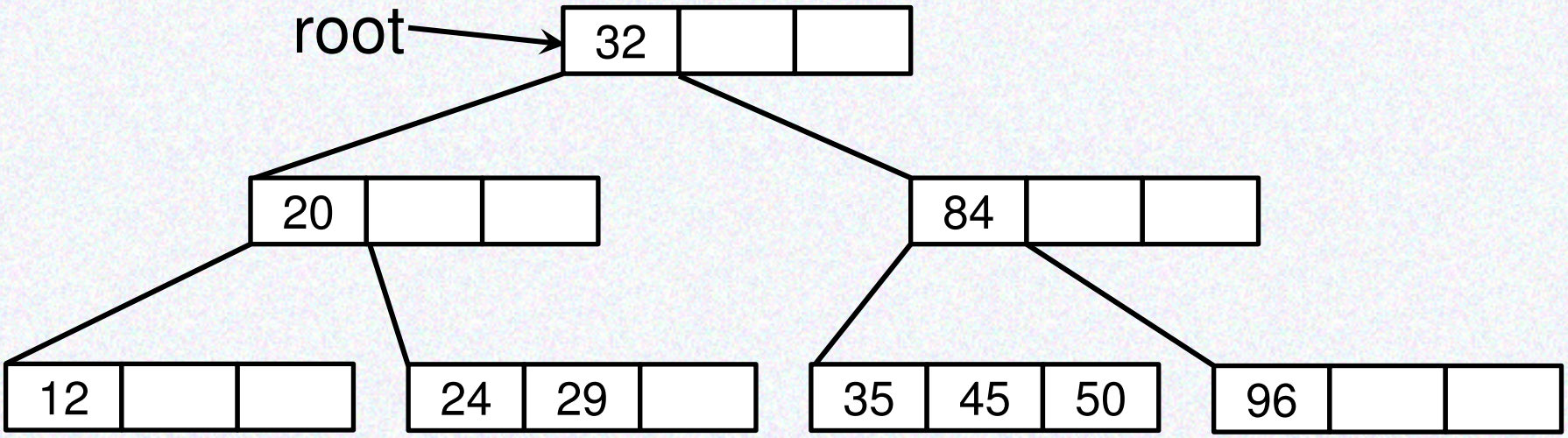
Разбиение корневого узла

Пример: вставка элементов



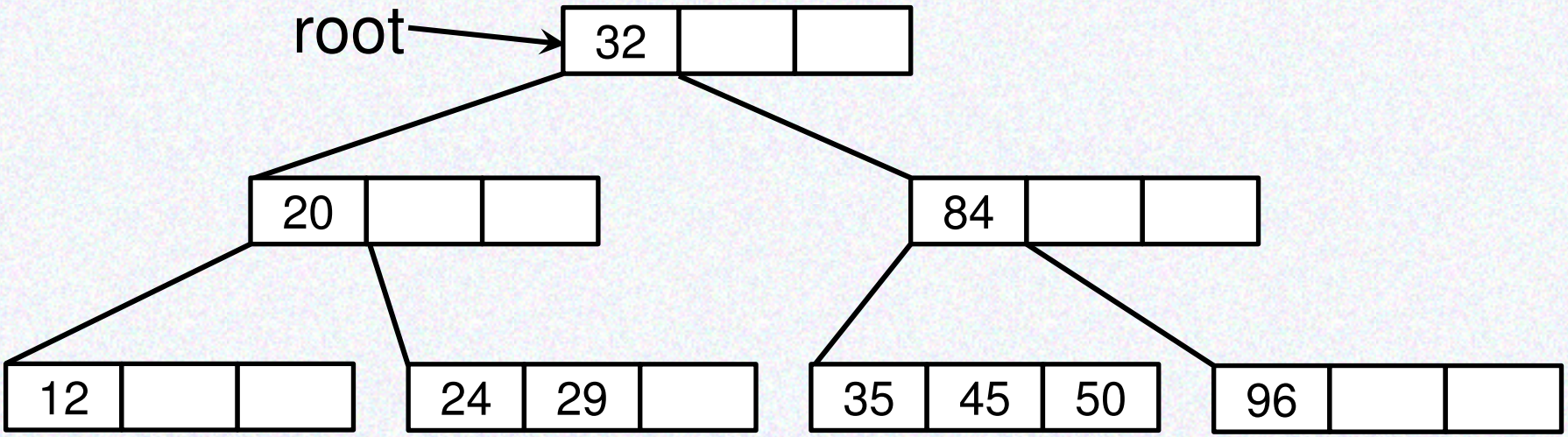
Вставка ключа 50

Пример: вставка элементов



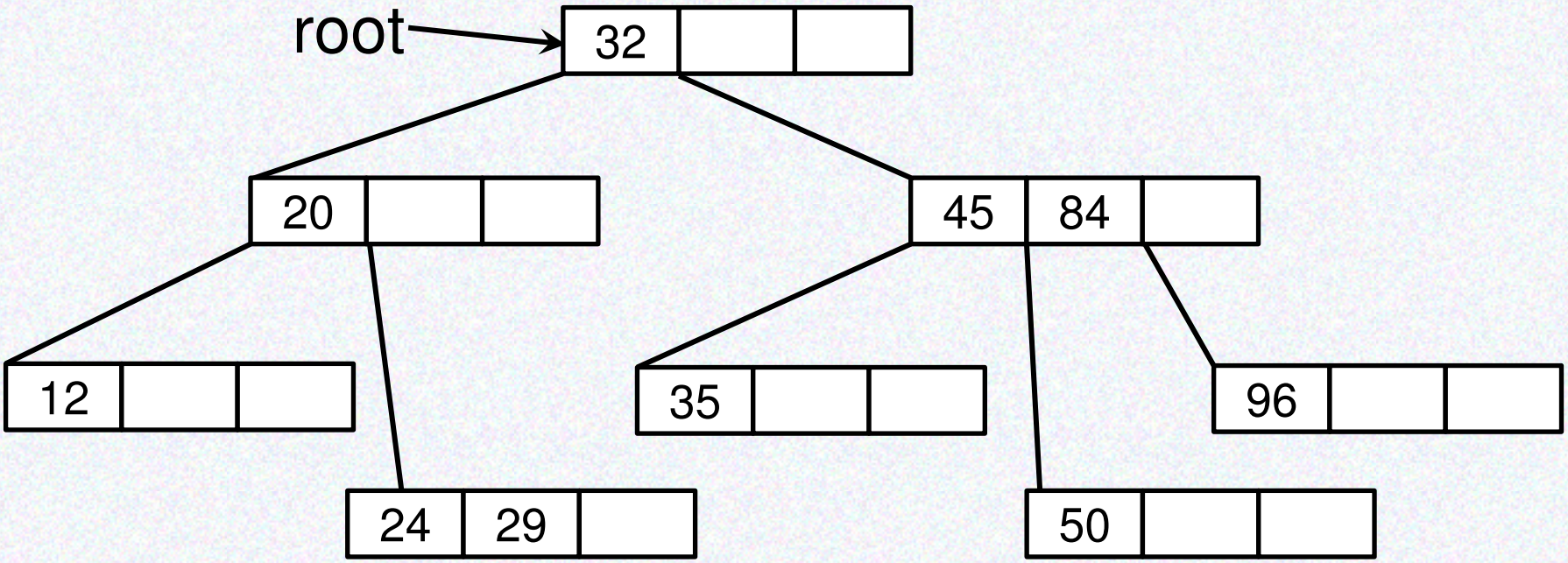
Пример: вставка элементов

Вставка ключа 63



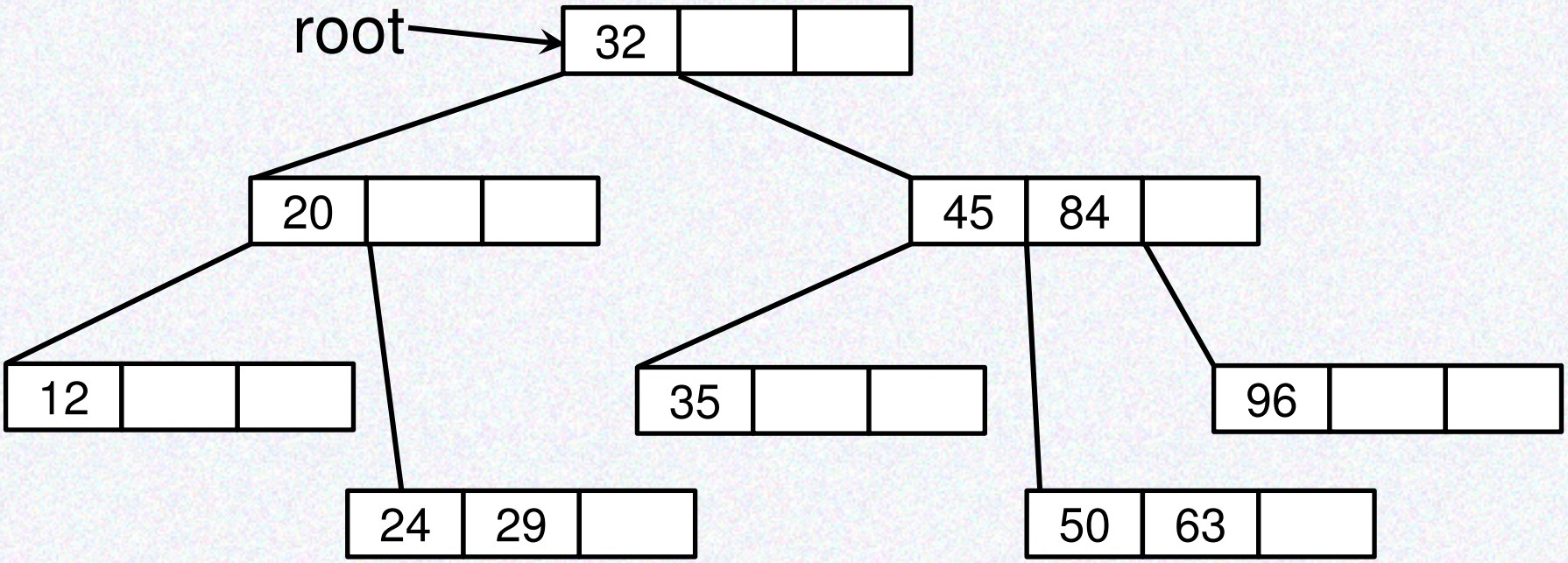
Разбиение листа

Пример: вставка элементов



Вставка ключа

Пример: вставка элементов



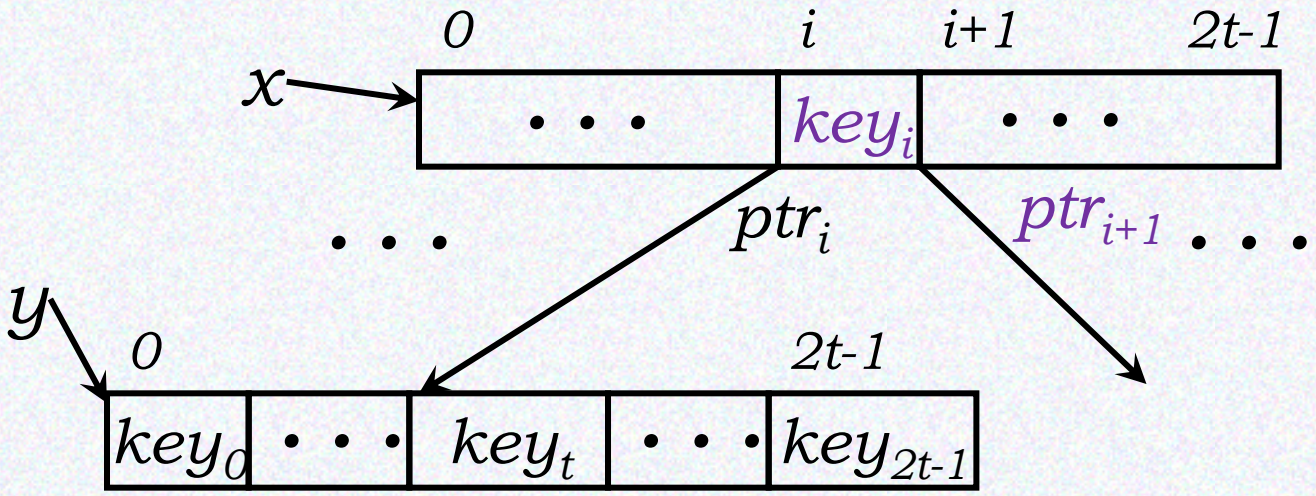
Разбиение узла

B-дерево степени t

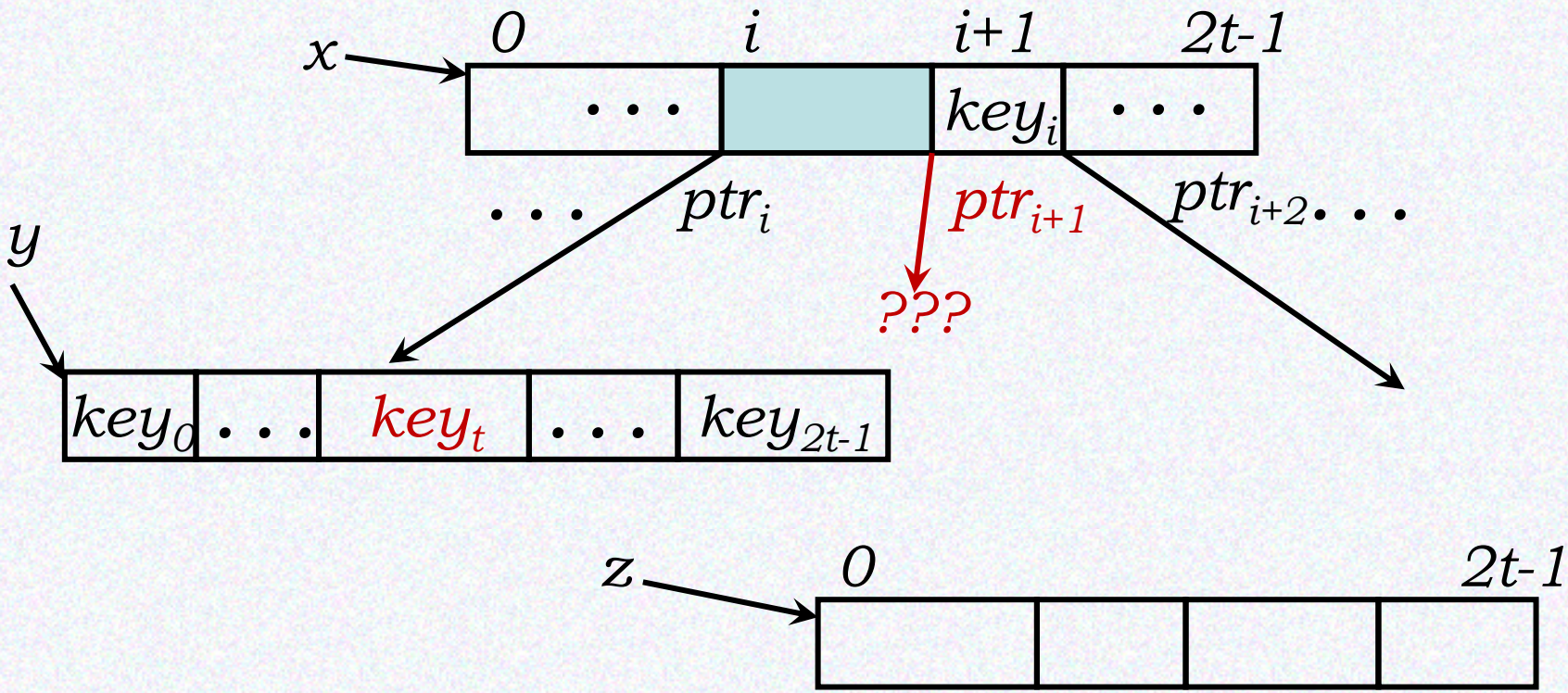
x – родительский узел

i – позиция, определяющая положение разбиваемого узла

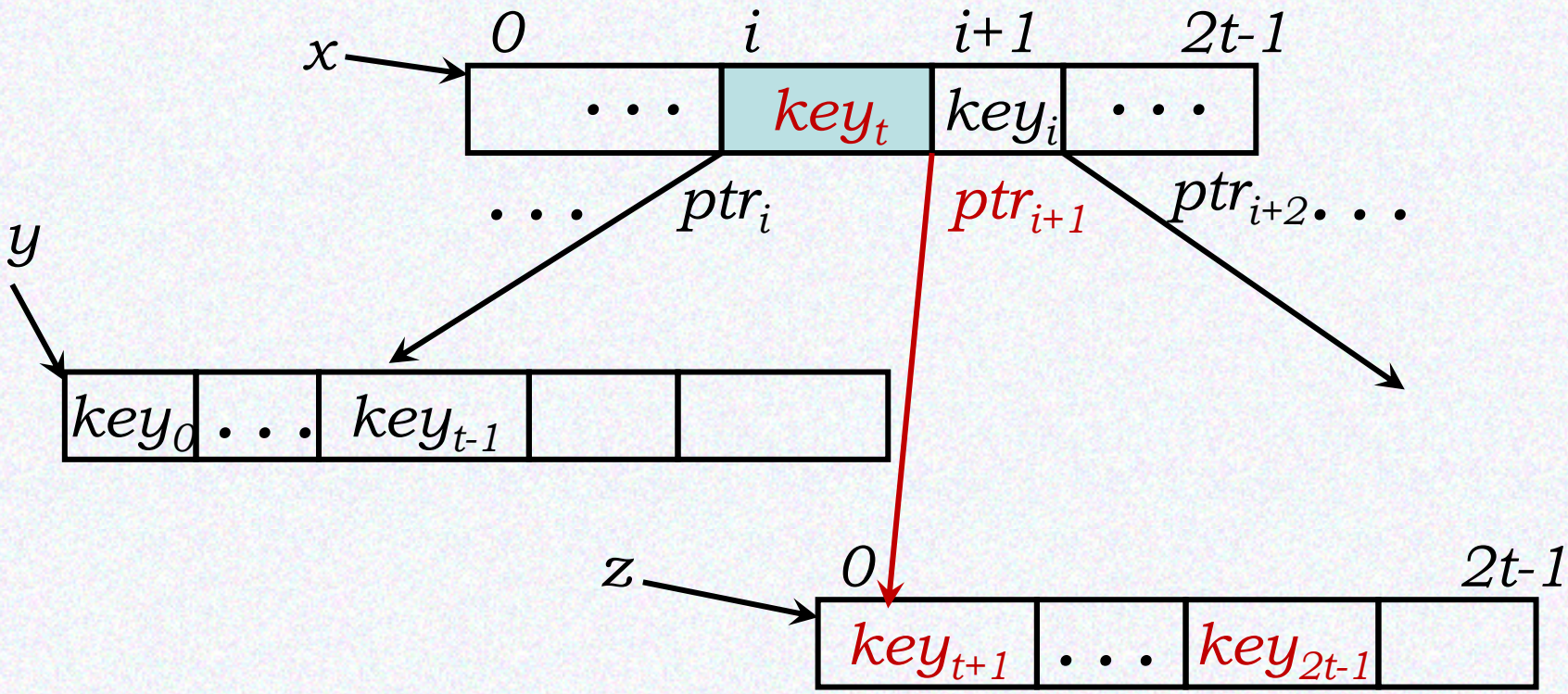
$y = ptr_i$ – указатель на разбиваемый узел



Разбиение узла



Разбиение узла



B_Tree_Split(x, i)

m – количество ключей в узле x ($m < 2t - 1$)

$y = ptr_i$

n – количество ключей в узле y ($n = 2t - 1$)

Сдвинуть в узле x ключи, начиная с i, и указатели, начиная с $i+1$, на 1 позицию вправо

Создать новый узел z, установить в этом узле все указатели в $NULL$

B_Tree_Split(x, i)

Переместить в узел z из узла y ключи и указатели, начиная с позиции $j = n/2 + 1$

Включить в x указатель на новый узел z:

$x\text{->}ptr_{i+1} = z$

Включить в узел x медианный ключ из узла y:

$x\text{->}key_i = y\text{->}key_{n/2}$

Скорректировать количество ключей в узлах x, y и z

Вставка в B-дерево

root – указатель на корень B-дерева

k – ключ нового элемента

r = *root*

if корень заполнен полностью {

создать новый узел *s*

s->n = *0*

$s\text{->}ptr_0 = r$

Вставка в B-дерево

root = *s*

разбить корень дерева: *B_Tree_Split(s,0)*

r = *s*

}

Вставить новый элемент в незаполненный узел B-дерева: *B_Insert_Nonfull(r, k)*

B_Insert_Nonfull(x, k)

x – указатель на незаполненный узел B-дерева

k – ключ нового элемента

while x – не лист {

i = позиция, определяющая положение нового ключа: $x\text{->}key_{i-1} < k \leq x\text{->}key_i$

if $k == x\text{->}key_i$

отказ: дублирование ключей

$s = x\text{->}ptr_i$ – поддерево, в котором должен размещаться ключ k

B_Insert_Nonfull(x, k)

if узел s заполнен полностью {

разбить узел: *B_Tree_Split(x, i)*

выбрать нужное поддерево:

if $k > x\text{->}key_i$

$s = x\text{->}ptr_{i+1}$

} – конец *if*

$x = s$

} – конец *while*

B_Insert_Nonfull(x, k)

x – указывает на лист B-дерева, не заполненный полностью

if в листе x есть ключ k

отказ: дублирование ключей

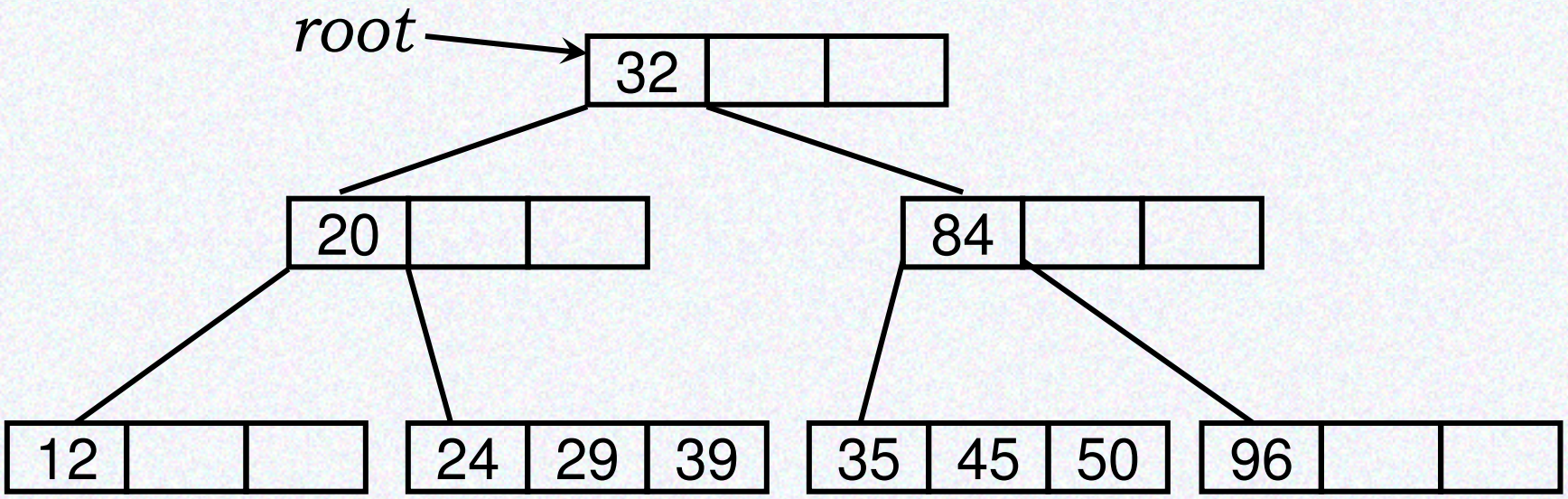
Вставить в узел x новый ключ k, не нарушая упорядоченности

Удаление элемента

- Удаляемый элемент находится в листе: корректируется лист
- Удаляемый элемент находится в промежуточном узле дерева

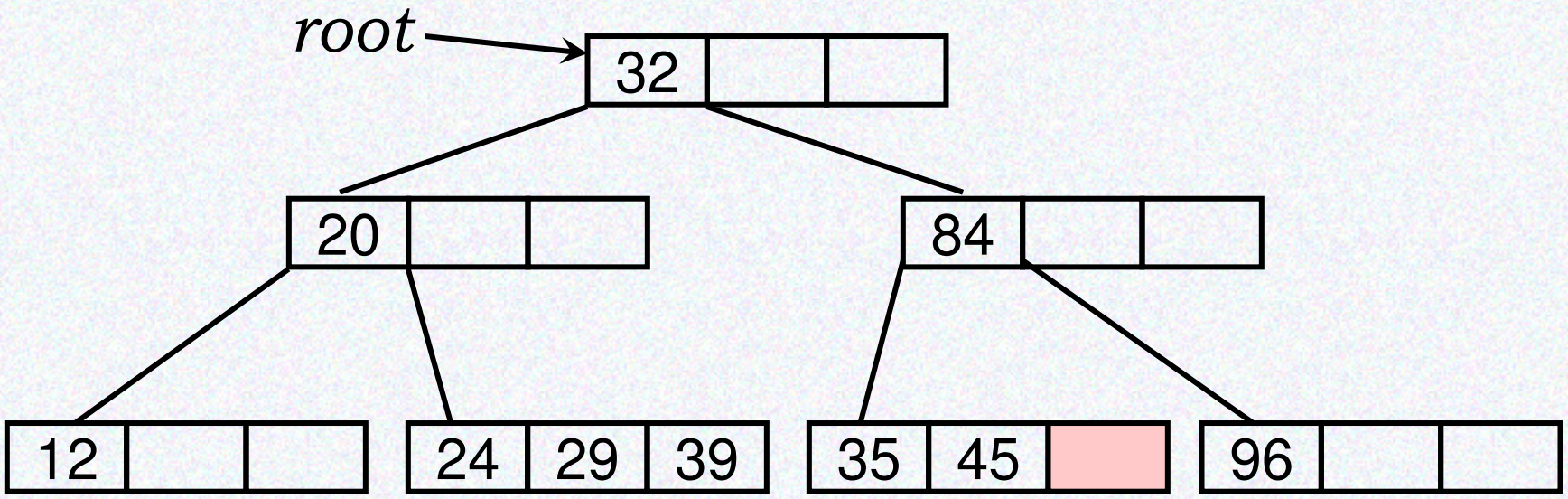
  – он заменяется следующим за ним (или предшествующим ему) значением: корректируется лист

Удаление элемента



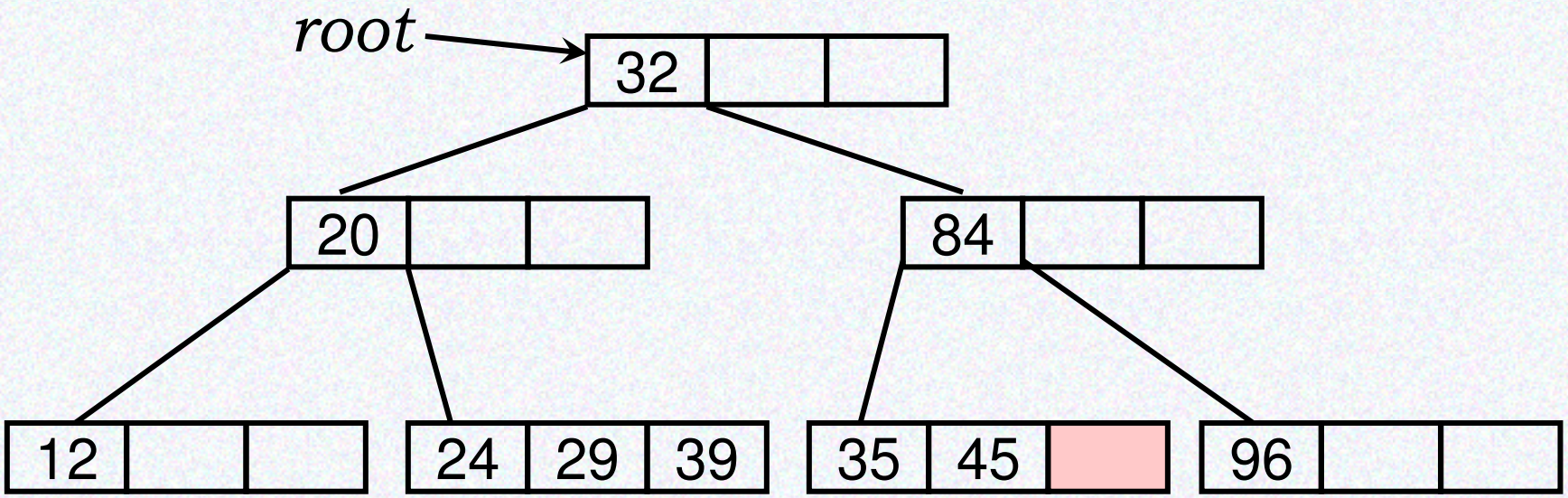
1. Удаляется элемент с ключом 50

Удаление элемента



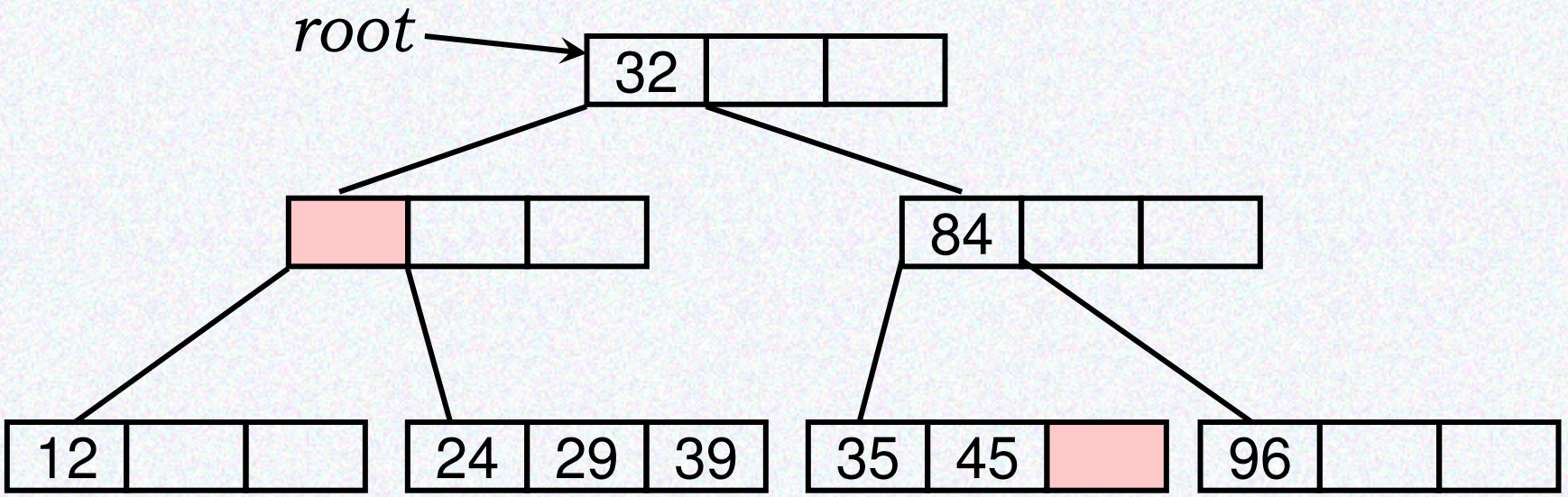
1. Удаляется элемент с ключом 50

Удаление элемента

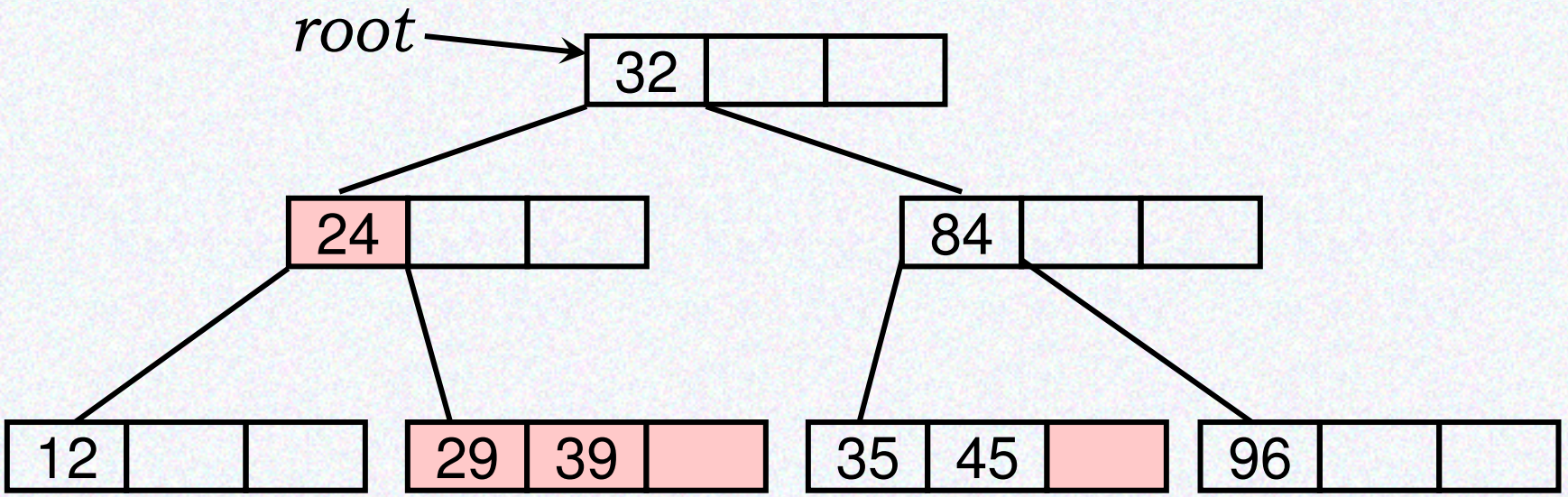


2. Удаляется элемент с ключом 20

Удаление элемента



Удаление элемента



Удаление элемента

Найти удаляемый ключ: x – узел дерева, содержащий удаляемый ключ k

if x – не лист {

y = самый левый лист из правого поддерева

на место удаляемого ключа переместить первый ключ из y: $y\text{->}key_0$

$x = y$, $k = y\text{->}key_0$

}

Удалить ключ k из листа x

if в листе x возникло антипереполнение

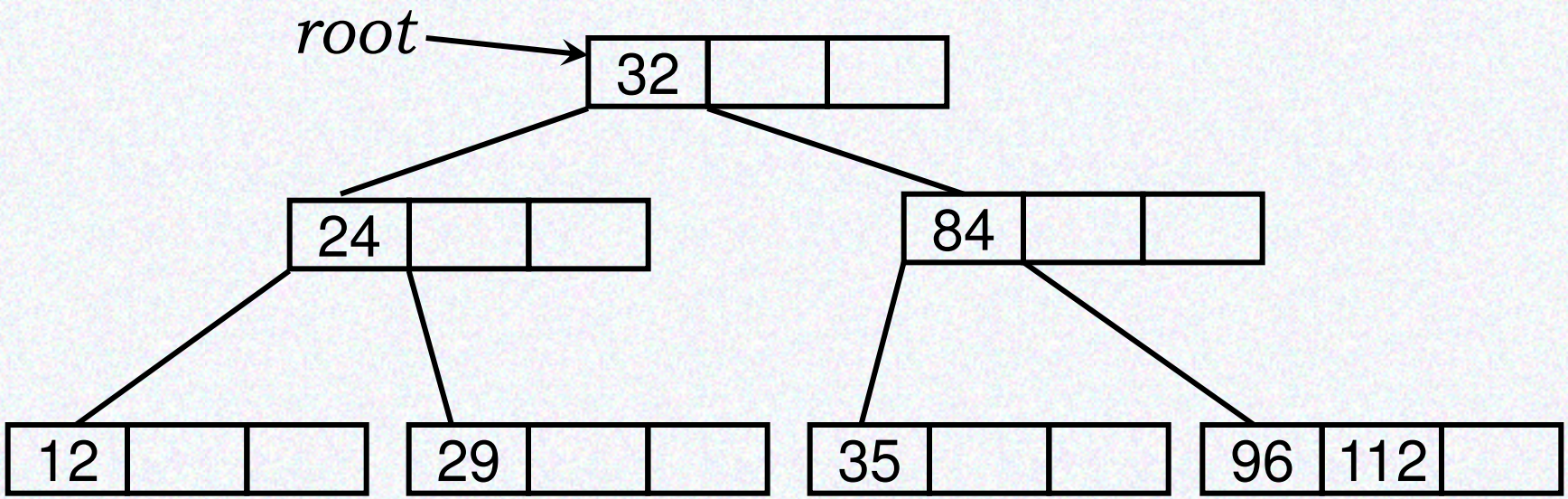
скорректировать структуру B-дерева

Способы коррекции

- Перераспределение ключей
- Слияние узлов

Примеры удаления

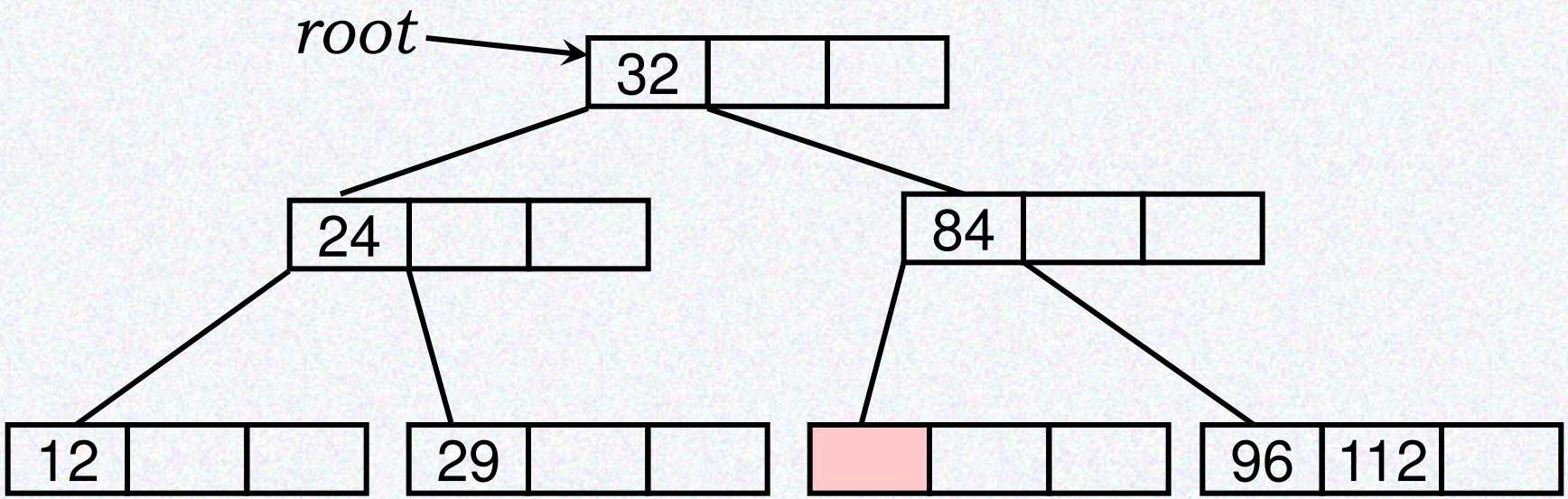
B-дерево степени 2



1. Удаляется элемент с ключом 35

Примеры удаления

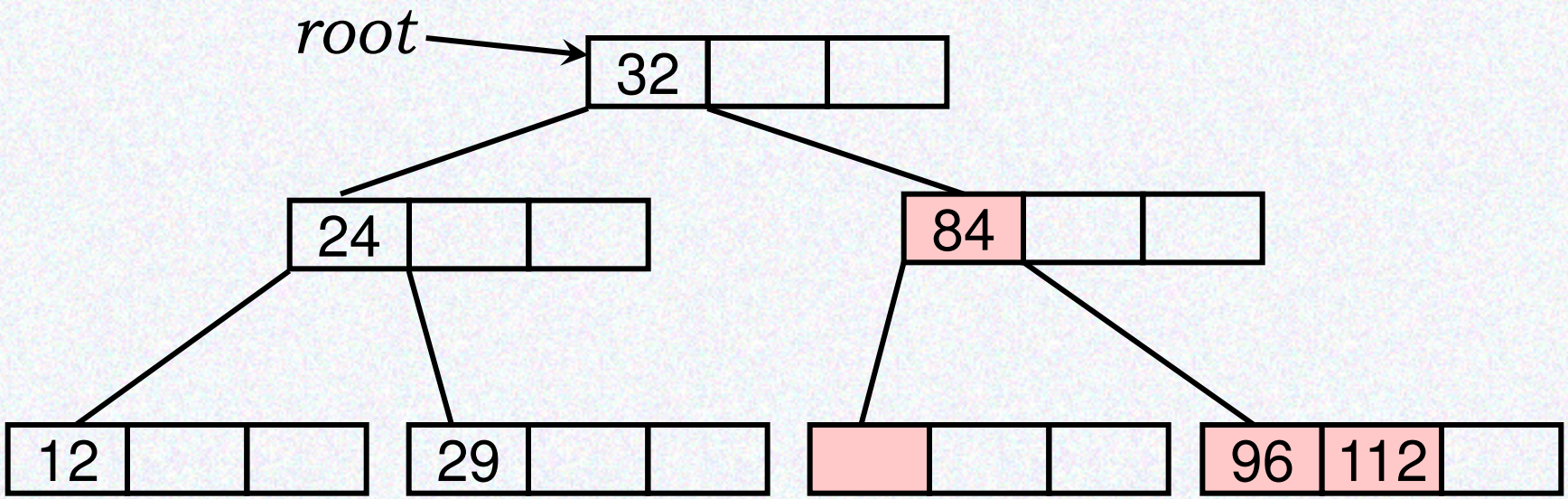
B-дерево степени 2



1. Удаляется элемент с ключом 35
   - *перераспределение ключей*

Примеры удаления

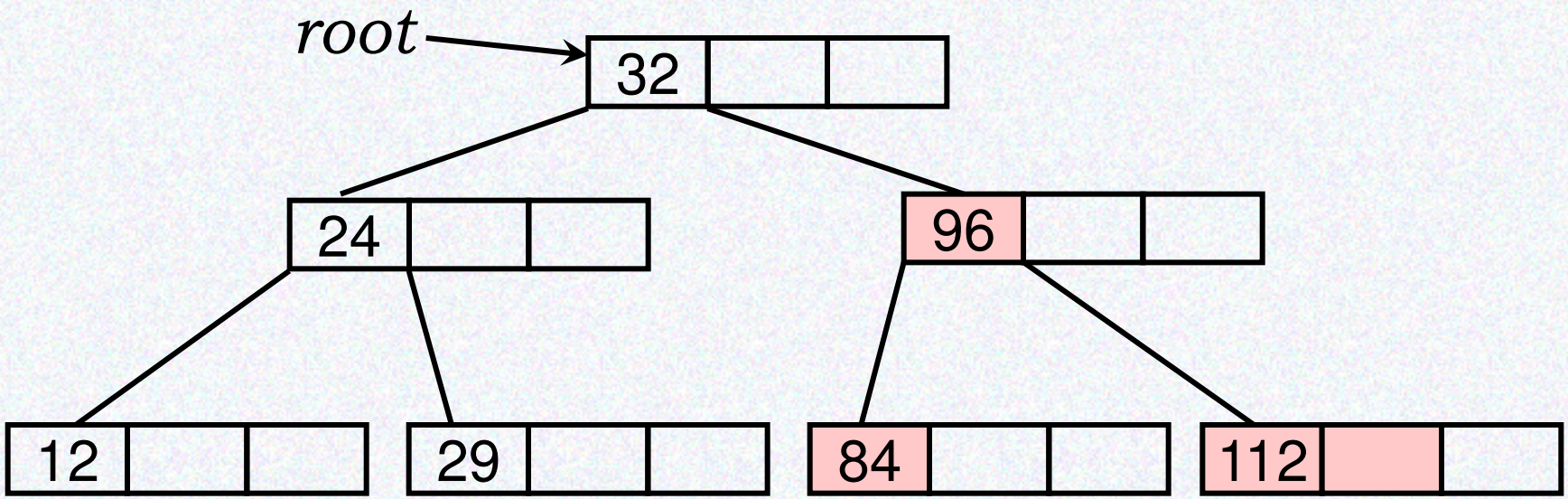
B-дерево степени 2



1. Удаляется элемент с ключом 35
   - *перераспределение ключей*

Примеры удаления

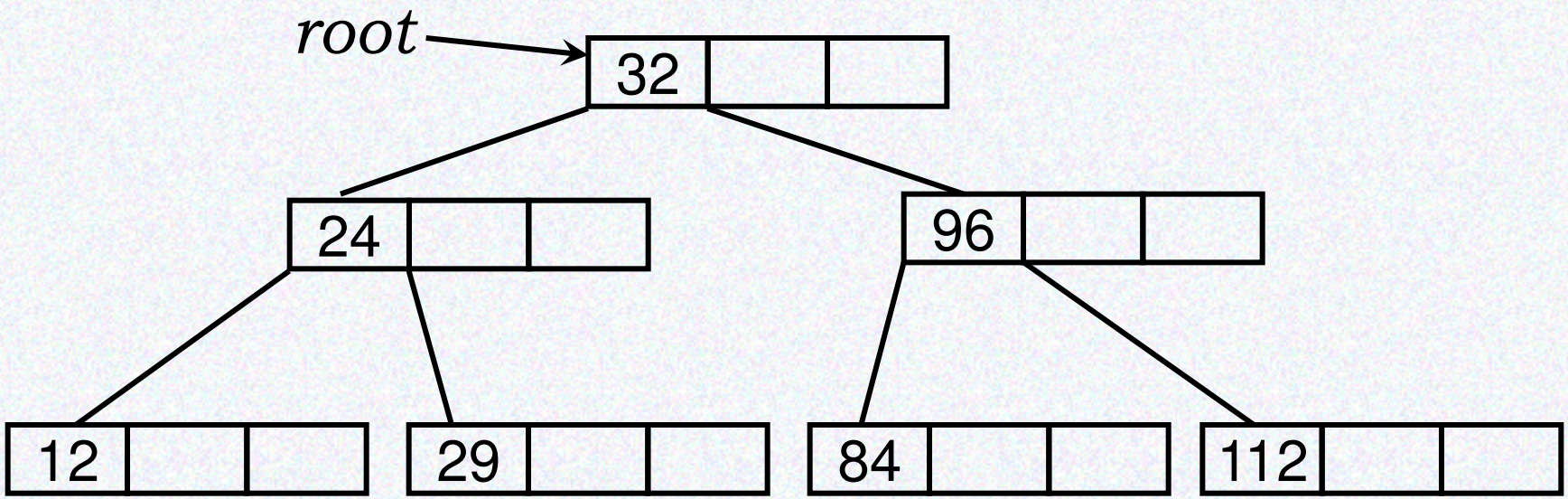
B-дерево степени 2



1. Удаляется элемент с ключом 35
   - *перераспределение ключей*

Примеры удаления

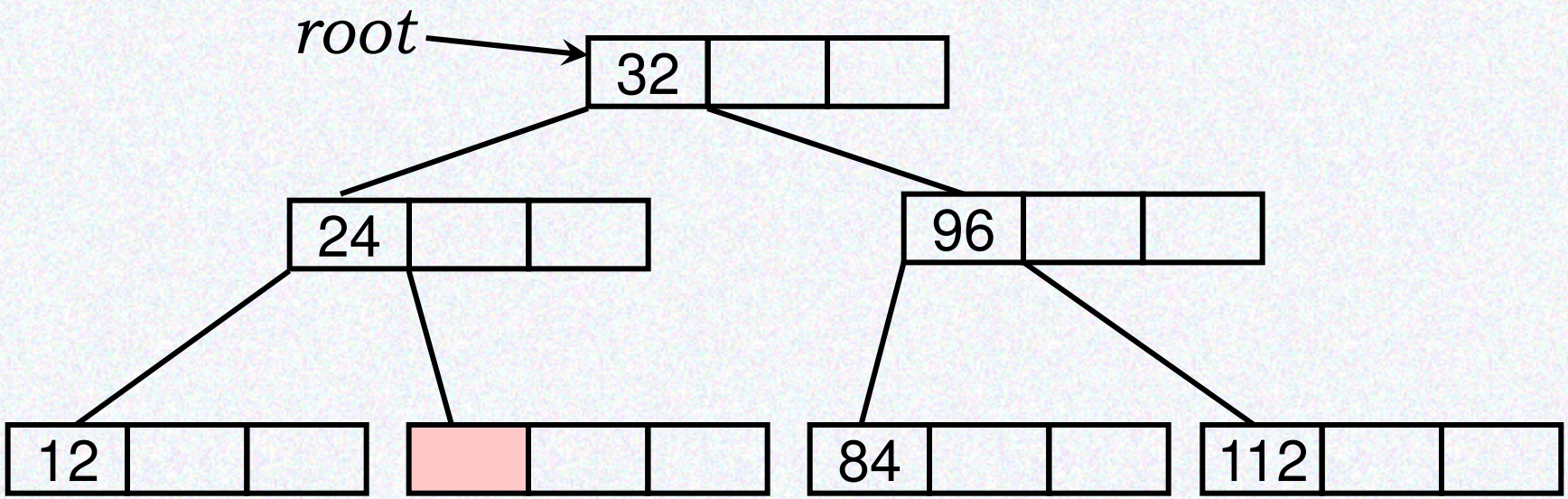
B-дерево степени 2



1. Удаляется элемент с ключом 29

Примеры удаления

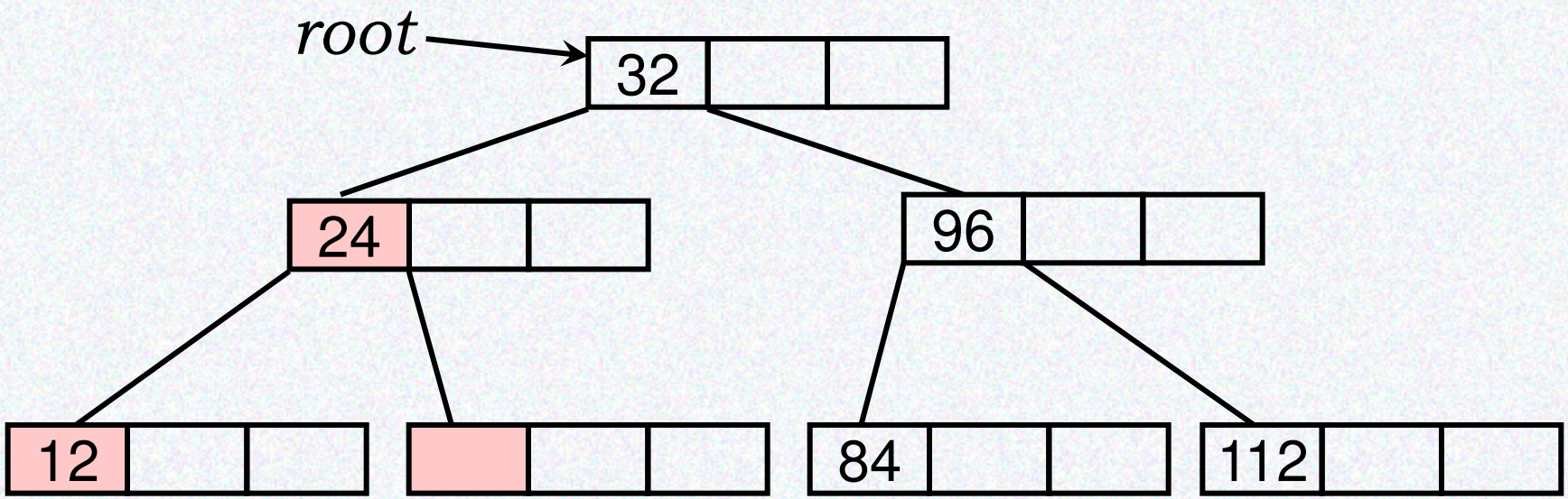
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

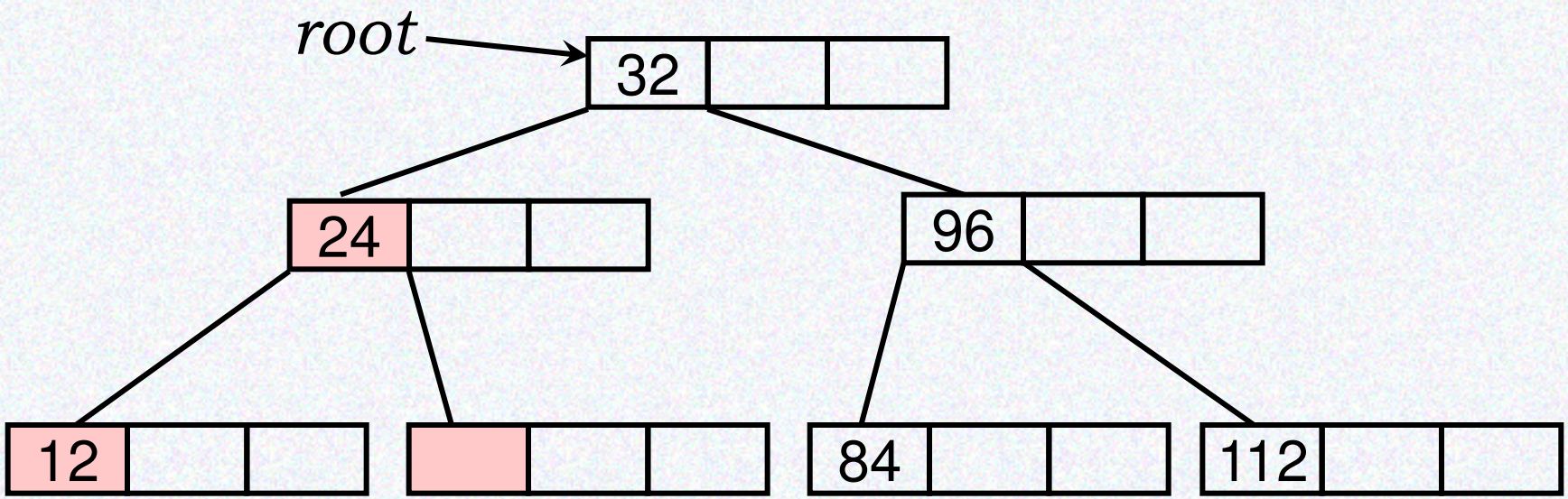
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

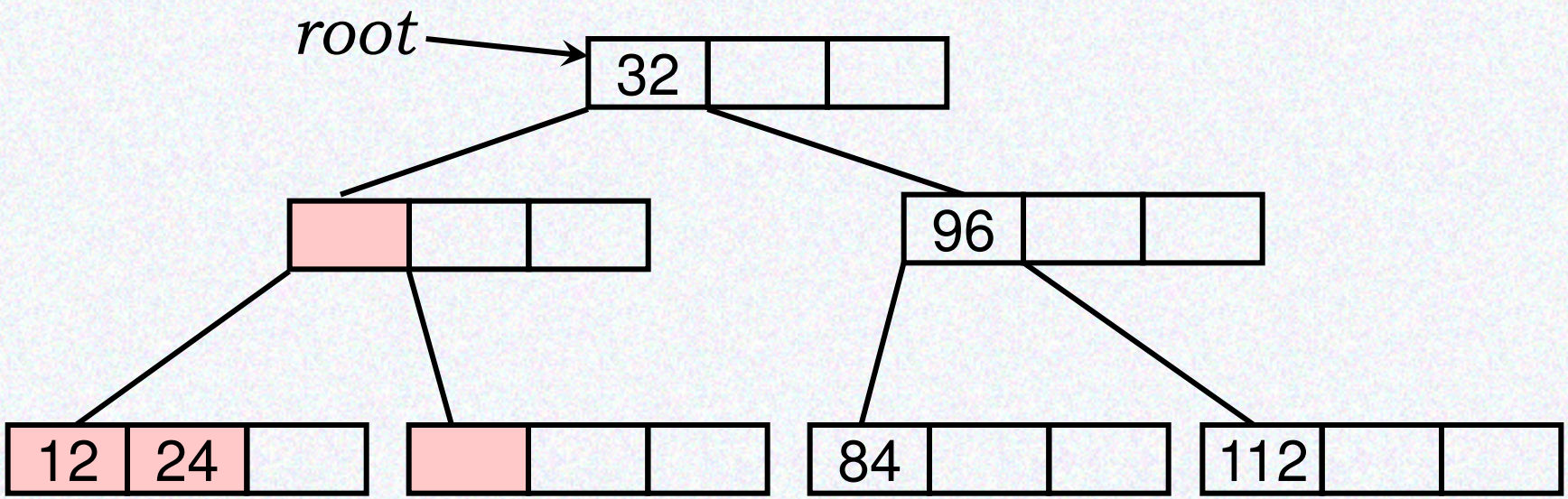
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

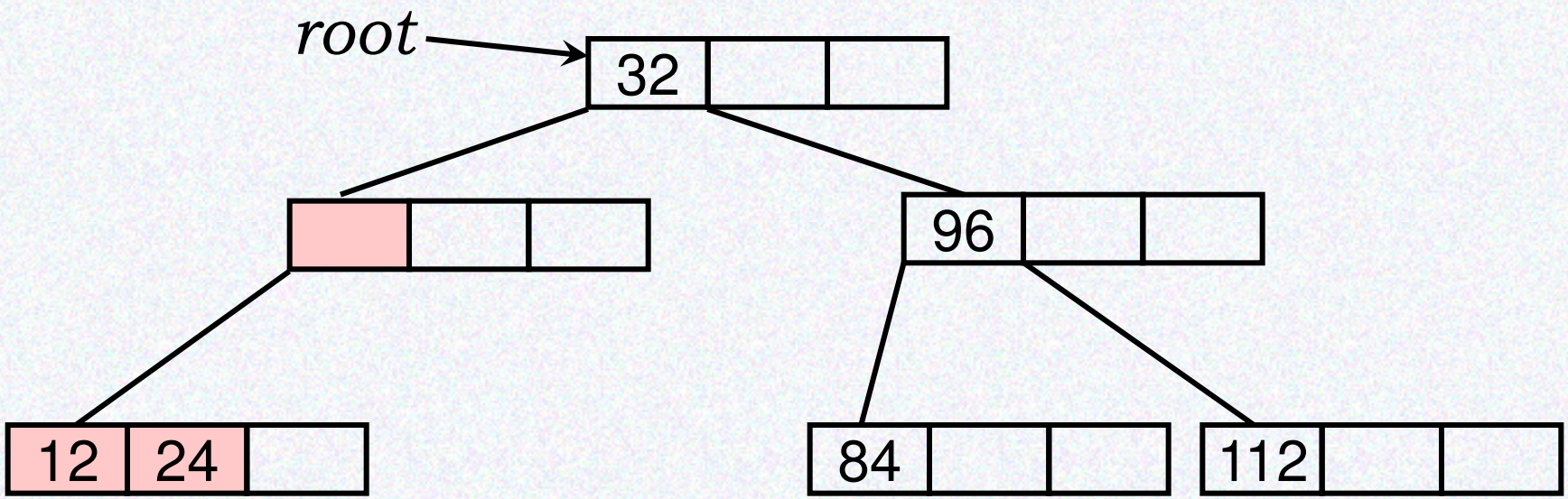
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

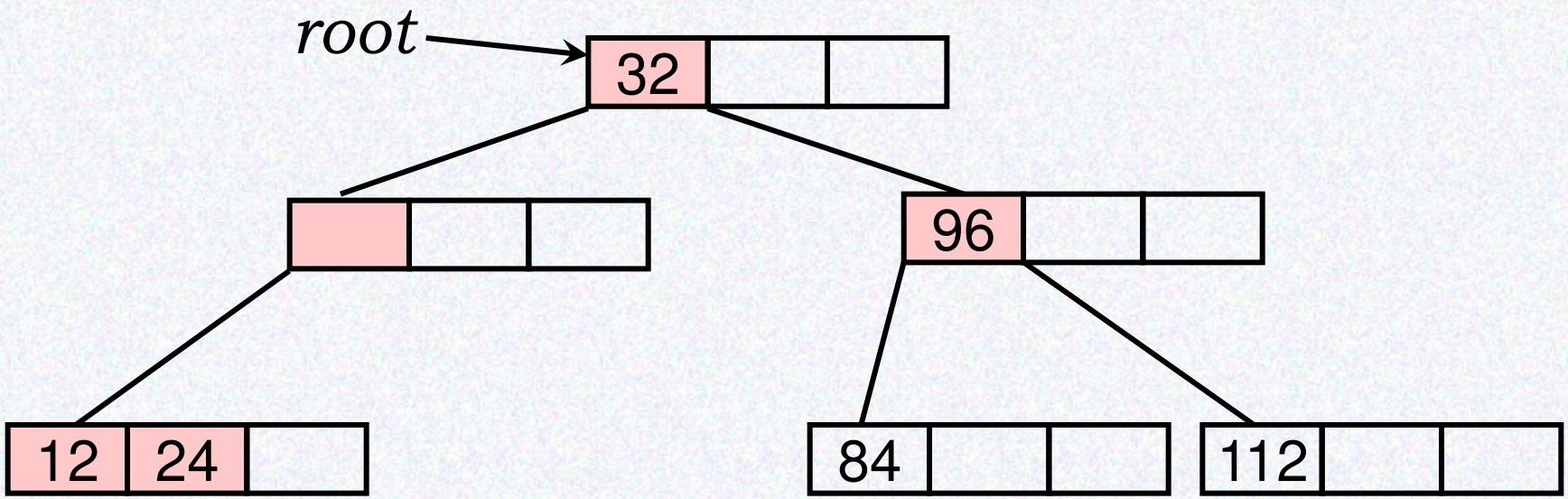
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

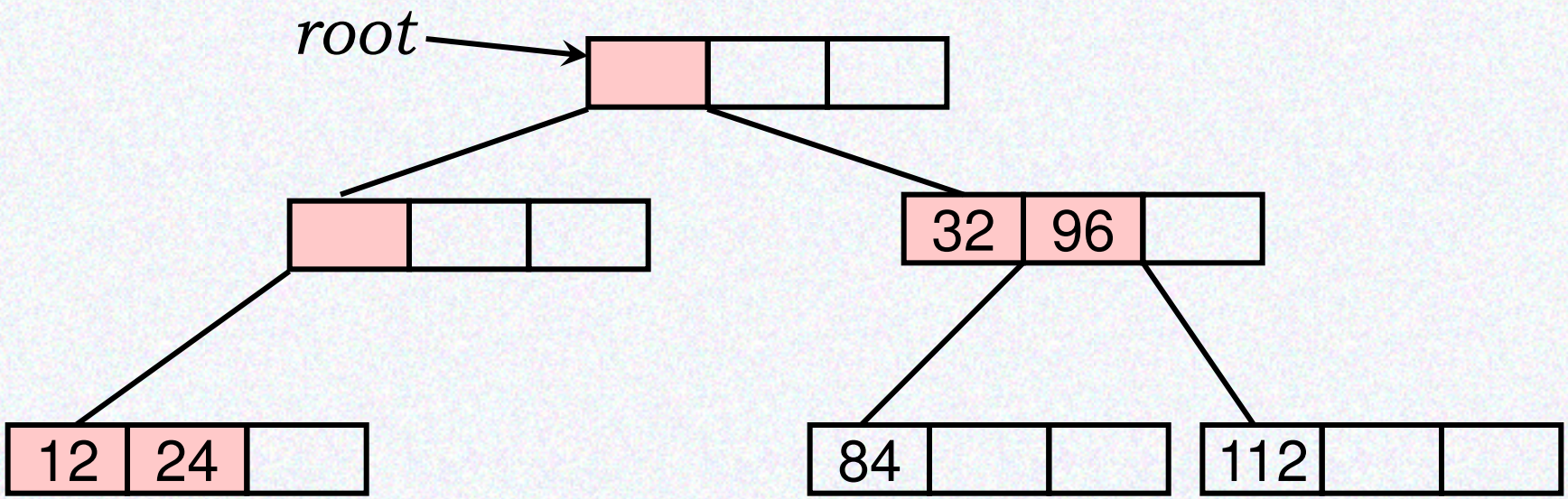
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

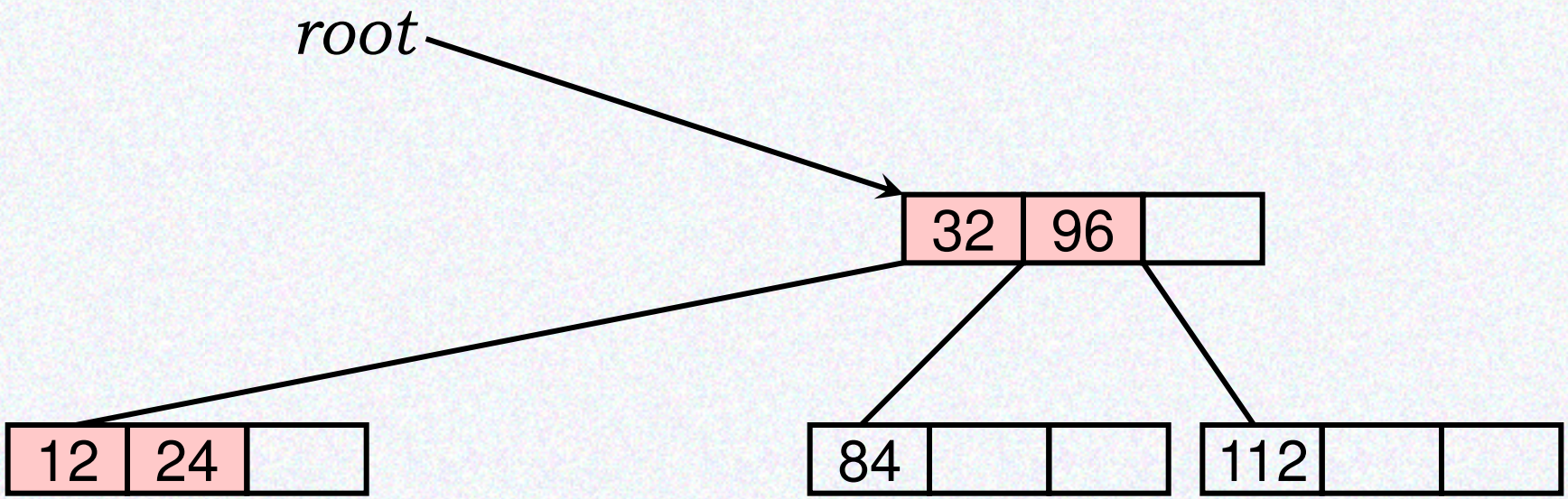
B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Примеры удаления

B-дерево степени 2



1. Удаляется элемент с ключом 29
   - *слияние узлов*

Алгоритм удаления

Обработать корень

$x = root$

while x не лист {

i = позиция ключа в x

if ключ не найден

ситуация 1

else

ситуация 2

}

Алгоритм удаления

x – лист

if x содержит ключ {
    удалить ключ
    Успех
}
Отказ

Обработка корня

$x = root$ – корень дерева

x содержит только один ключ

y и z – левый и правый узлы-потомки, содержат только по t – 1 ключу

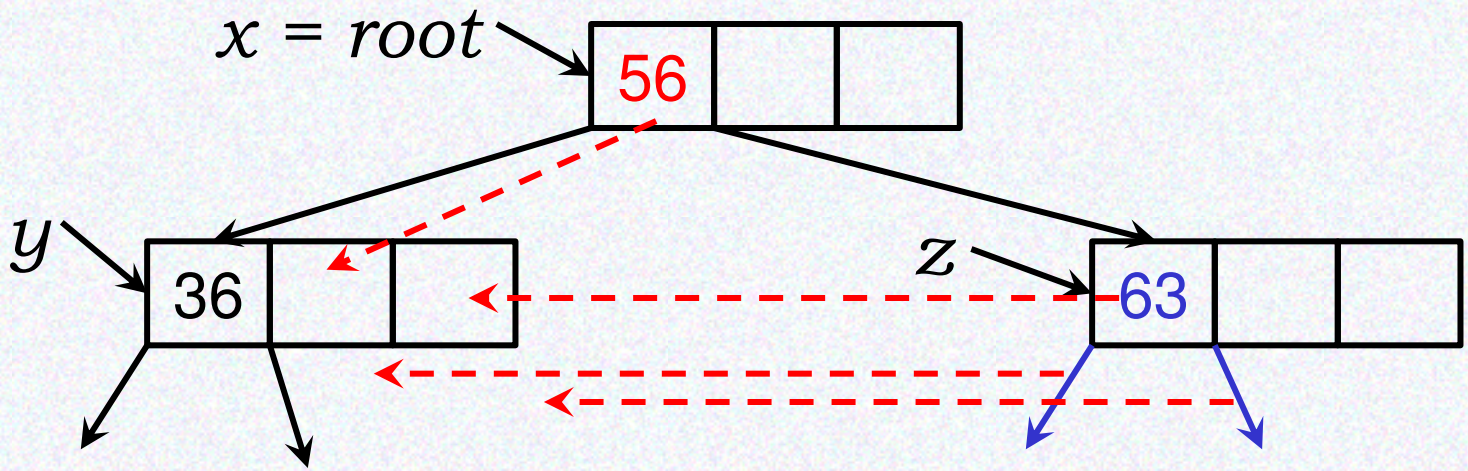
Узлы y, x, z объединяются в узле y

$root = y$

Узлы x и z удаляются

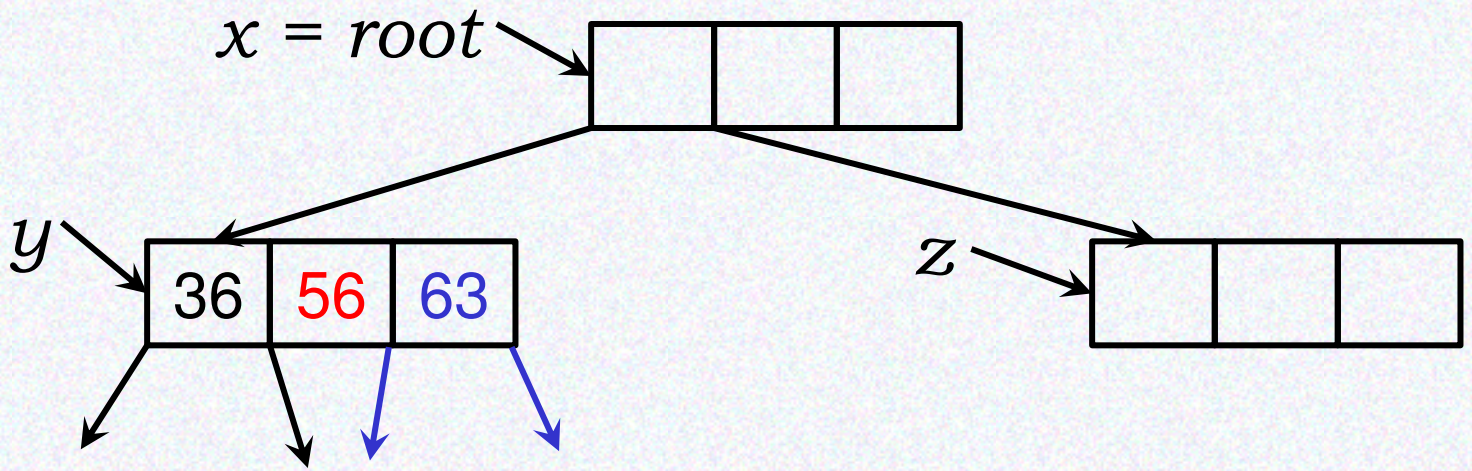
Обработка корня

Пример: B-дерево степени 2



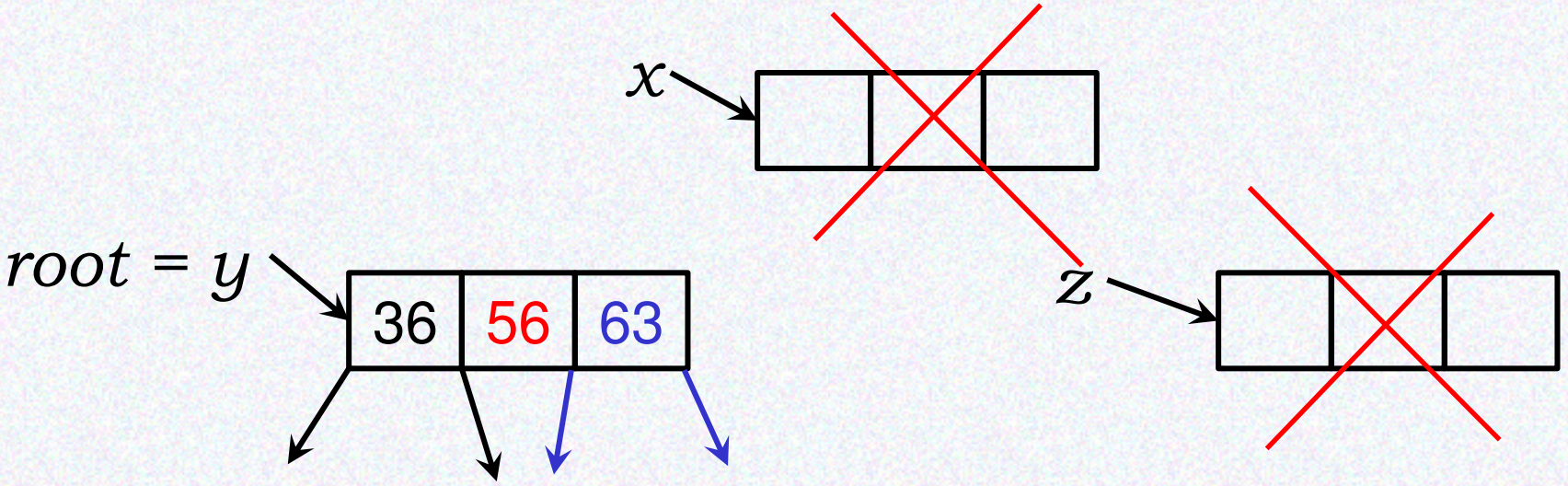
Обработка корня

Пример: B-дерево степени 2



Обработка корня

Пример: B-дерево степени 2



Ситуация 1

x – внутренний узел дерева, ключ k отсутствует в узле x

$y = ptr_i$ – поддерево, которое должно содержать ключ k

if y содержит только t – 1 ключей, но один из его непосредственных соседей (z) содержит $\geq t$ ключей {

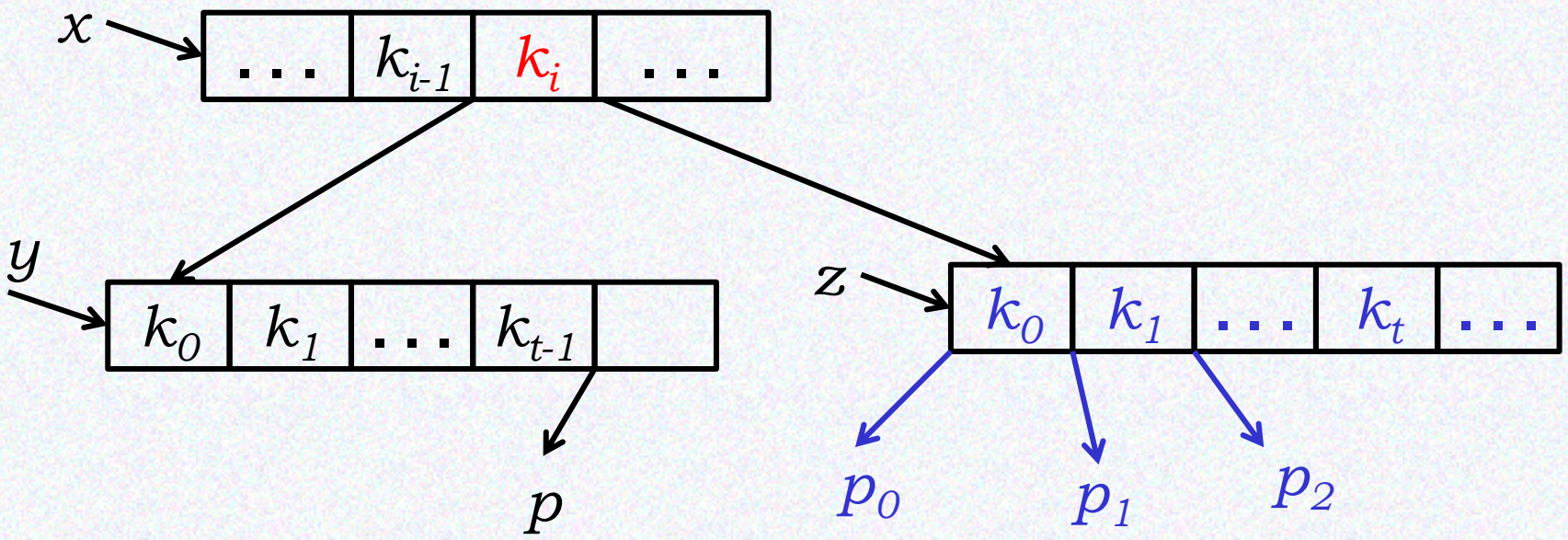
переместить ключ-разделитель из x в y

переместить крайний ключ из z в x

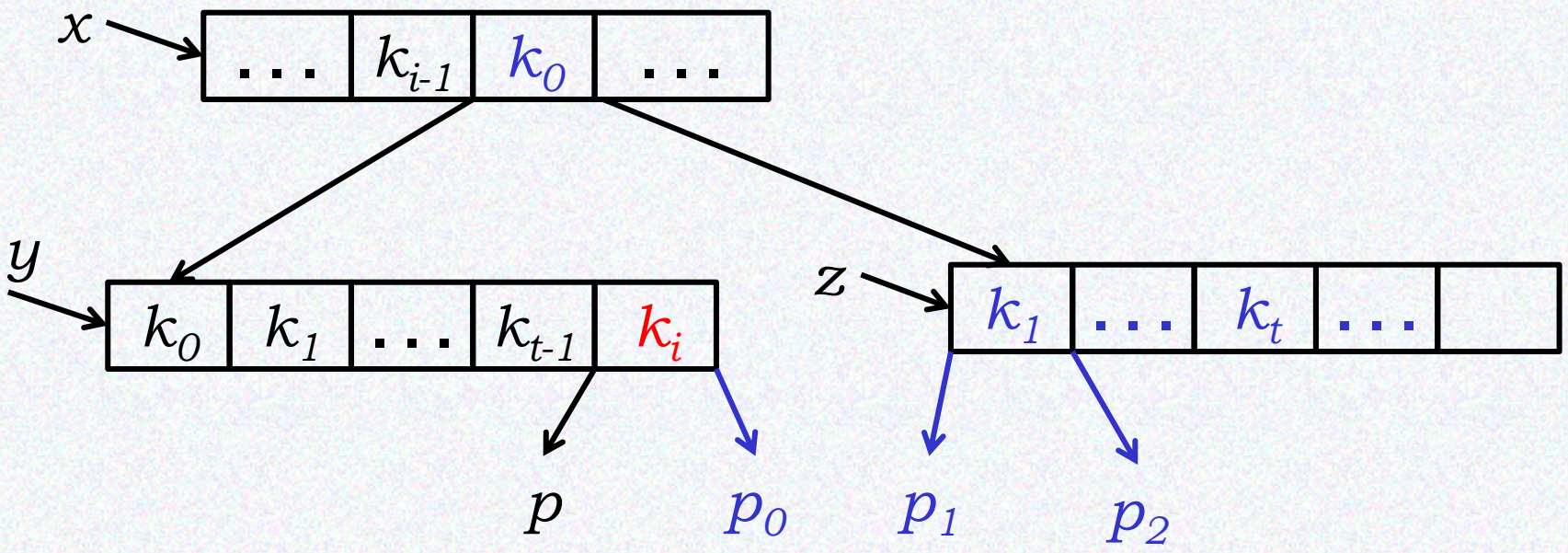
переместить соответствующий указатель из z в y

}

Ситуация 1



Ситуация 1



Ситуация 1

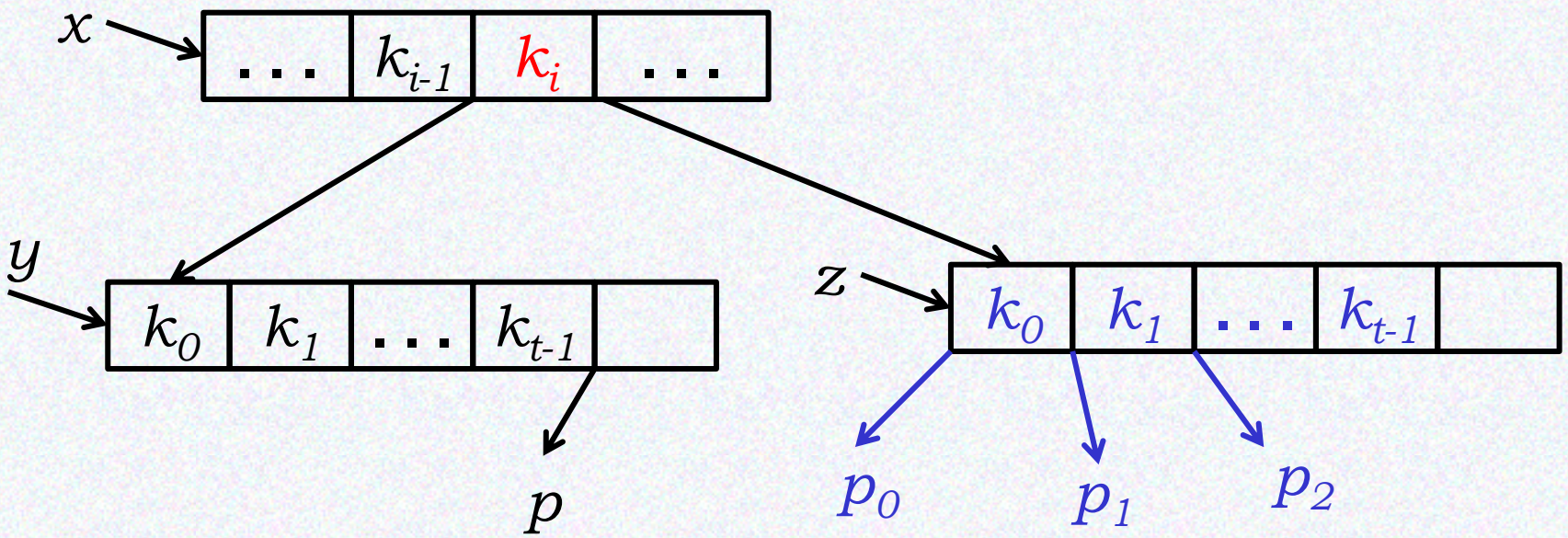
else if y и оба его непосредственных соседа содержат только по t – 1 ключей {

переместить ключ-разделитель из x в y

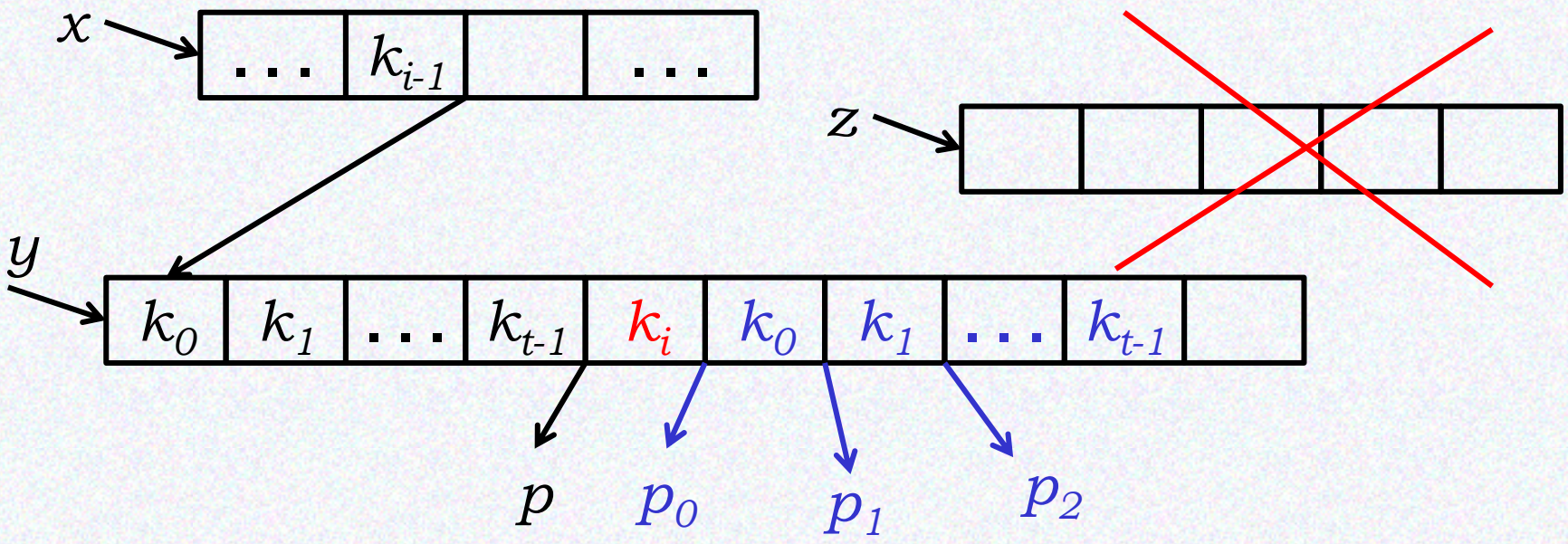
объединить y с одним из его соседей

}

Ситуация 1



Ситуация 1



Ситуация 1

Гарантированно узел y будет содержать $\geq t$ ключей

Продолжить итерации: $x = y$

Ситуация 2

x – внутренний узел дерева, ключ k находится в узле x

y – узел, левый потомок для ключа k

z – правый потомок для ключа k

Ситуация 2

if y содержит $\geq t$ ключей {

найти k' – предшественника k в поддереве с корнем y

заменить в x ключ k на k'

выполнить процедуру удаления ключа k' из поддерева с корнем y: $x = y$

}

Ситуация 2

else if z содержит $\geq t$ ключей {

найти k' – следующий за k в поддереве с корнем z

заменить в x ключ k на k'

выполнить процедуру удаления ключа k' из поддерева с корнем z: $x = z$

}

Ситуация 2

else {

и y, и z содержат только по t – 1 ключей

переместить k и все ключи и указатели из z в y

освободить z

выполнить процедуру удаления ключа k из поддерева с корнем y: $x = y$

}