NATIONAL RESEARCH UNIVERSITY

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Структуры данных для вторичной памяти B-Trees

Рамон Антонио Родригес Залепинос
arodriges@hse.ru

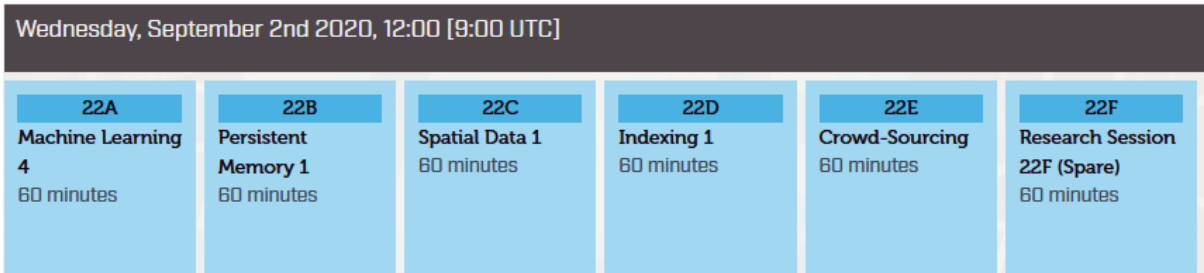
Структура модуля 2

	№		Дата	Тема лекции	№	Домашние задания
Модуль № 2	7	1	27 окт	Хеширование, хэш таблицы 2	2b	ДЗ-Каникулы
	8	2	03 ноя	Фильтры	3	Задание на C++
	9	3	10 ноя	СД для вторичной памяти	4	Задание на C++
	10	4	17 ноя	Пространственные СД	5	Задание на C++
	11	5	24 ноя	Параллельные СД	6	Задание на C++ и/или C# (зависит от выбранного уровня сложности)
	12	6	01 дек	Параллельные СД 2		
	13	?	08 дек	Деревья в оперативной памяти		Примерно за 2 недели заканчиваются ДЗ
	14	8	15 дек	Современные тренды		
				СЕССИЯ с 21.12.2020		

Highlights:

- лекции, семинары и ДЗ синхронизированы
- ? – возможно будет еще одна лекция, с другой темой
- некоторые представления об эффективности СД развеяны:
  - на семинаре вы собственноручно, на практике сравните производительность красно-черных деревьев и хэш таблиц (если не нужны next & prev)
- пространственные СД – обширный класс
  - в современном мире, около 80% всех данных содержат географическую привязку: ссылка 1 (Forbes), ссылка 2 (Carto)
  - отдельные секции на значимых конференциях (e.g., VLDB: https://vldb2020.org/program.html)

Требование к студентам на лекции: слушайте внимательно!



План лекции: СД для вторичной памяти

1. «Постановка задачи»
2. Особенности вторичной памяти: HDD, SSD, облако
3. Проблемы СД для оперативной памяти
4. Особенности оценки времени работы
5. Классическое B-дерево (SIGMOD)
6. Идеи устройства B-tree
7. Определение B-tree
8. Высота B-tree
9. Поиск в B-tree
10. Вставка в B-tree
11. Разбиение узлов и автобалансировка
12. Примеры работы алгоритма вставки
13. Примеры применения B-tree
14. Резюме лекции

«Постановка задачи»

Входные данные

- Ключи $\boldsymbol{key} \in \boldsymbol{K}$ (+ полезная нагрузка)

Особенность

- Все данные не помещаются полностью в основную (оперативную) память

Ключевое решение

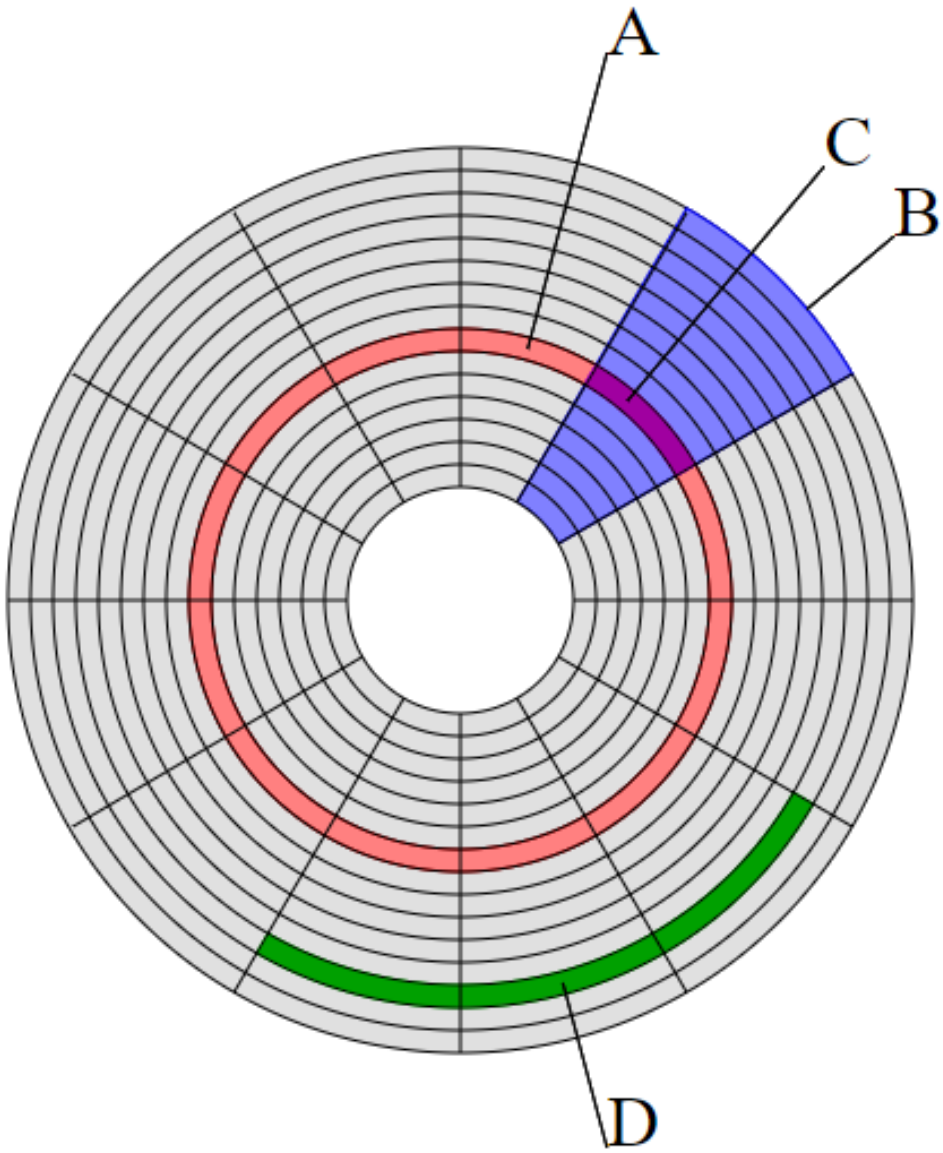
- Хранить данные во вторичной памяти

Цель – эффективное выполнение словарных операций

- По ключу
- Insert, delete, search, previous, next, …

Пример вторичной памяти: HDD

Особенность: блочный ввод/вывод



Структура диска:

(A) дорожка

(B) геометрический сектор

(C) сектор дорожки

512 … 4096 байт

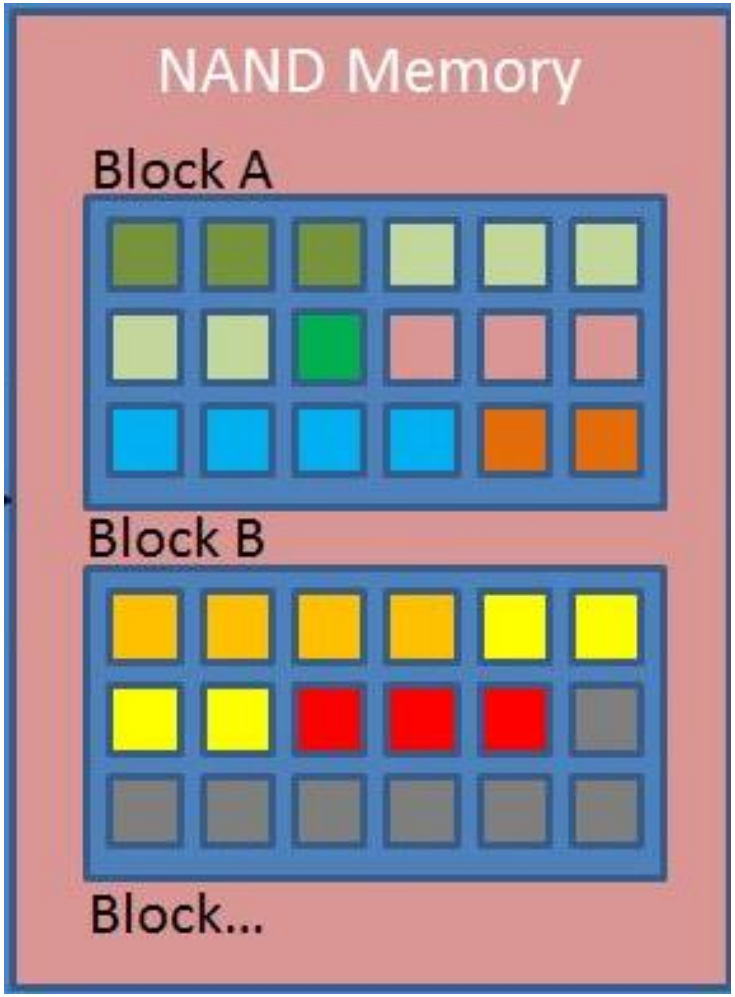
(D) кластер

512 bytes … 64 KB

единица I/O

I/O unit

Пример вторичной памяти: SSD



Структура диска:

Ячейка →

Страница ≈ 4 KB →

Блок ≈ 128 стр. ≈ 512 KB

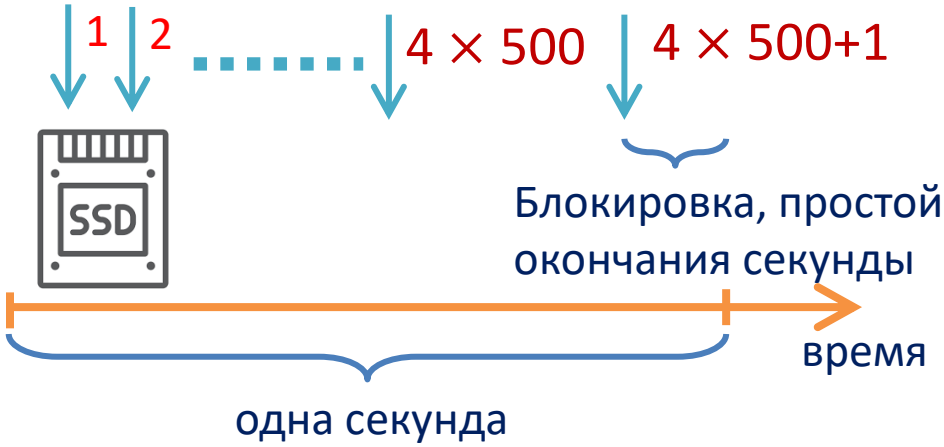
единица I/O

https://www.oo-software.com/en/docs/whitepaper/whitepaper_ssd.pdf

Вторичная память в Облаках: рамки еще жестче

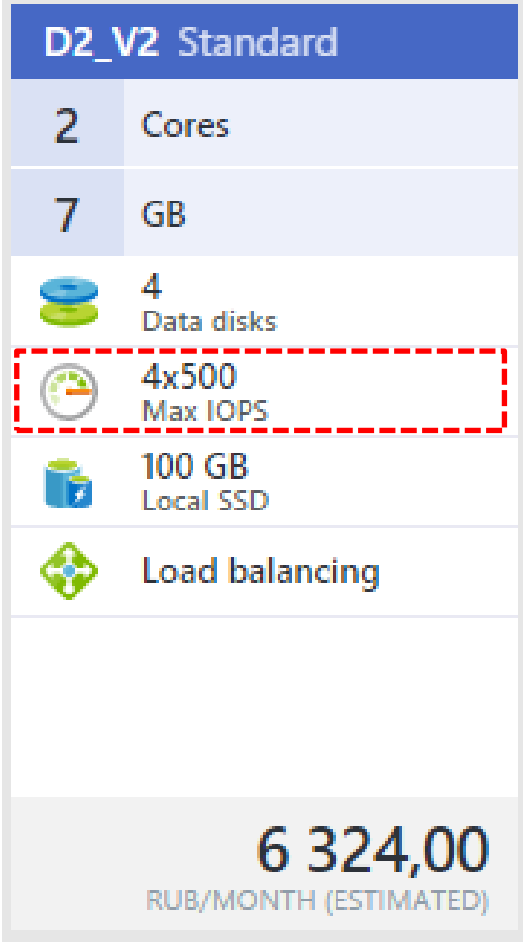
Зачем учитывать объем памяти и число операций ввода/вывода?

Современный пример
Стоимость (цена, руб.) работы алгоритма:
IOPS & объем памяти



4 × 500 =
max IOPS

Input/**O**utput
Operations
per **S**econd

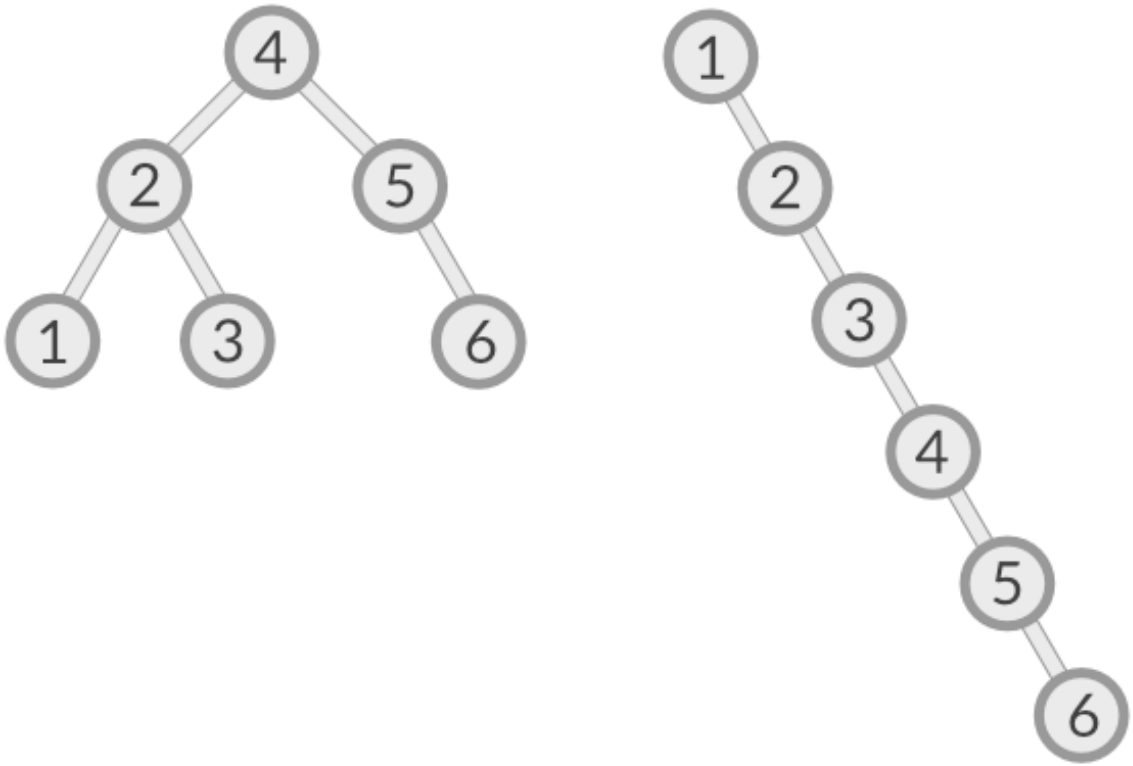


Мы говорили об этом на лекции по асимптотической сложности
Мы вернулись к этому при изучении структур данных для вторичной памяти

Проблемы с data structures для основной памяти

Например, binary search tree

Показаны минимум две проблемы…



https://appliedgo.net/balancedtree/

Время доступа к данным

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Особенности оценки времени работы

- Для алгоритмов в основной памяти

$T(n) =$ время работы процессора

- Для алгоритмов во вторичной памяти

$T(n) =$ время процессора **И**

число обращений к вторичной памяти

Классическое B-дерево

Rudolf Bayer, Edward M. McCreight, 1970 г.

"Organization and maintenance of large ordered indices".

Proceedings of the 1970 **ACM SIGFIDET (now SIGMOD)** Workshop on Data Description, Access and Control - SIGFIDET '70. Houston, Texas: ACM Press: 107. doi:10.1145/1734663.1734671

SIGMOD – одна из самых значимых в мире конференций по

- **Управлению Данными**
- **Базам Данных**

Всего 7 статей от РФ за все время проведения SIGMOD

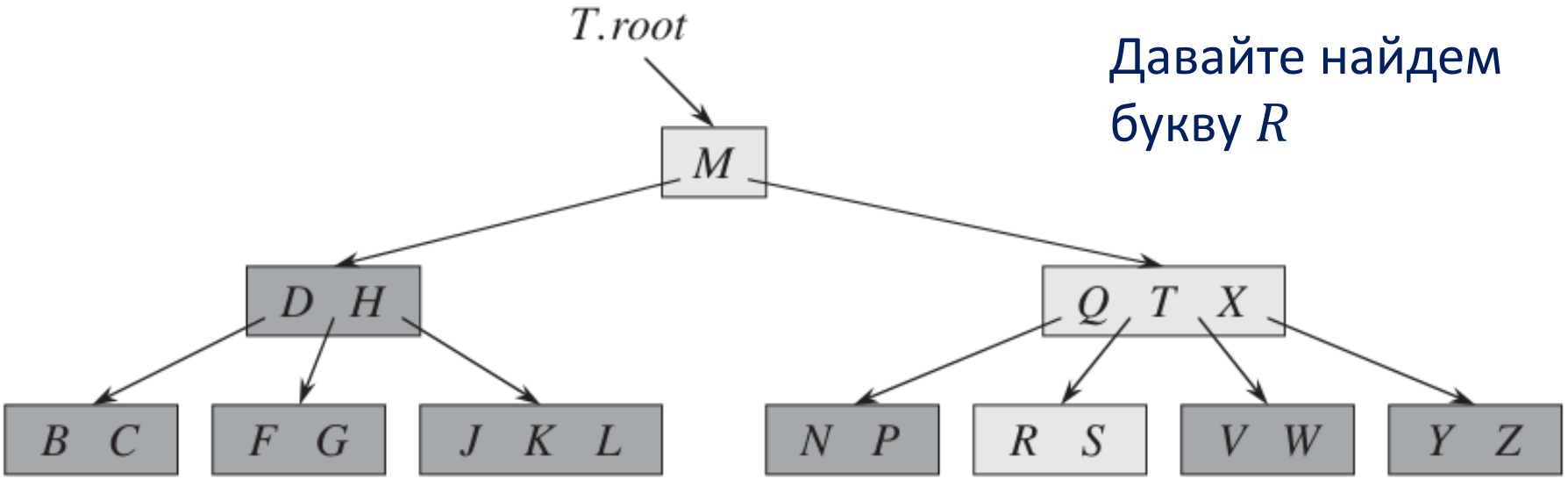
Сложность операций B-tree

- Хранит n элементов
- Каждый элемент имеет ключ key
- Для типа ключа определены операции сравнения, напр., <

Базовые операции	Худший случай
Занимаемое место	$O(n)$
Поиск по ключу	$O(log_t n)$
Вставка по ключу	$O(log_t n)$
Удаление по ключу	$O(log_t n)$

Идеи устройства B-tree

1. Поддерживать дерево **сбалансированным** (все листья находятся на одной высоте)
2. Поддерживать **«небольшую» высоту** (для прохода по дереву будет требоваться $O(\text{высота дерева})$ операций ввода/вывода)
3. Писать и читать **«по многу» и редко** из вторничной памяти (обычно размер одного узла B-tree $\approx$ размеру кластера: внутренний узел x, у которого $x.n$ ключей, имеет $x.n + 1$ детей.
4. В основной памяти хранить только $O(1)$ узлов дерева, которые необходимы для прохода по нему/его модификации

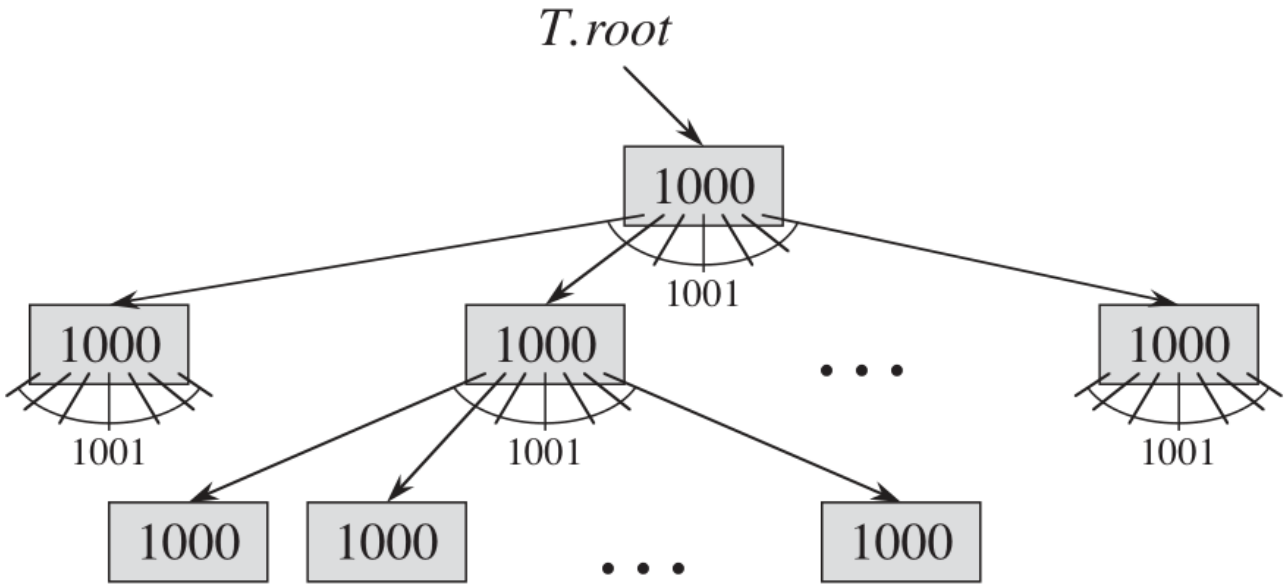


Давайте найдем букву R

[1]

Ветвление B-tree

1. Больш**а**я степень ветвления резко снижает высоту дерева
2. На рисунке ниже – B-дерево высотой 2, содержит более миллиарда ключей.
3. На практике часто данные не хранятся в B-дереве, при этом полезная нагрузка (хранится вместе с ключом) – указатель на область диска, в которой хранятся данные



Количество узлов и ключей на заданной высоте

1 node,
1000 keys

1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

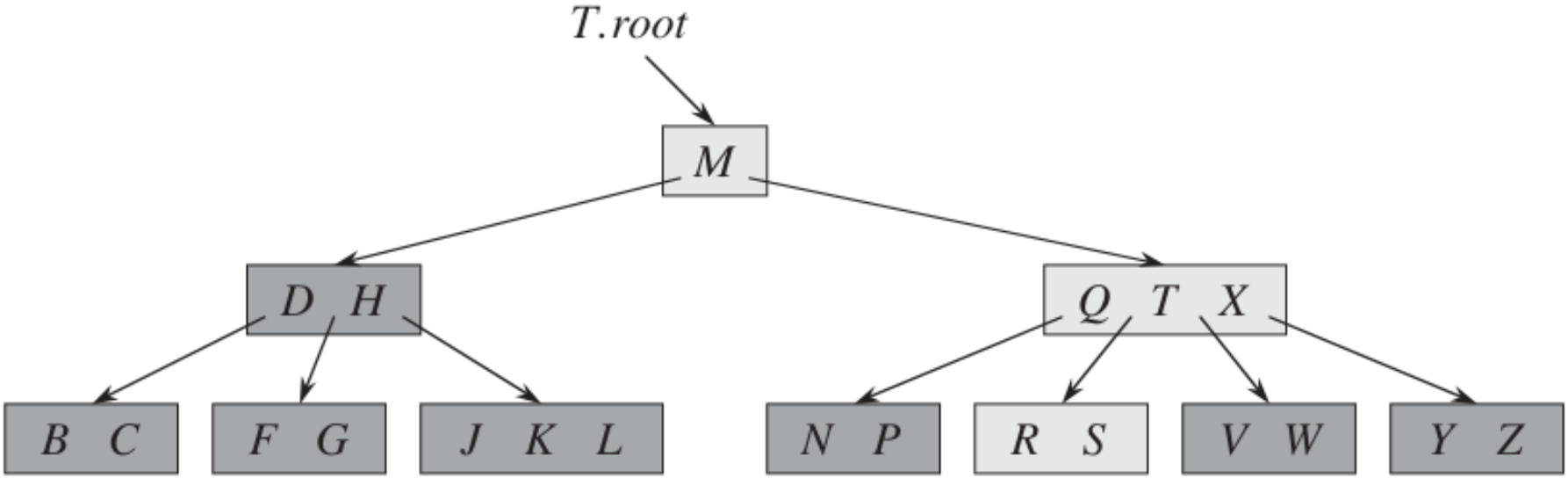
[1]

Определение B-tree

1. Каждый узел x содержит
   a) Число хранимых ключей $x.n$
   b) Ключи $x.key_1 \leq x.key_2 \leq \cdots \leq x.key_{x_n}$
   c) Boolean $x.leaf$ – является ли узел листом
   d) $x.n + 1$ указателей на дочерние узлы: $x.c_1, x.c_2, \ldots x.c_{x_n+1}$
2. Ключи $x.key_i$ разделяют диапазоны ключей поддеревьев

   $$k_1 \leq x.key_1 \leq k_2 \leq x.key_2 \leq \cdots \leq x.key_{x_n} \leq k_{x_n+1}$$

   Где k_i - ключ, хранящийся в поддереве с корнем $x.c_i$
3. Все листья расположены на одной глубине $h =$ высота дерева



[1]

Определение B-tree

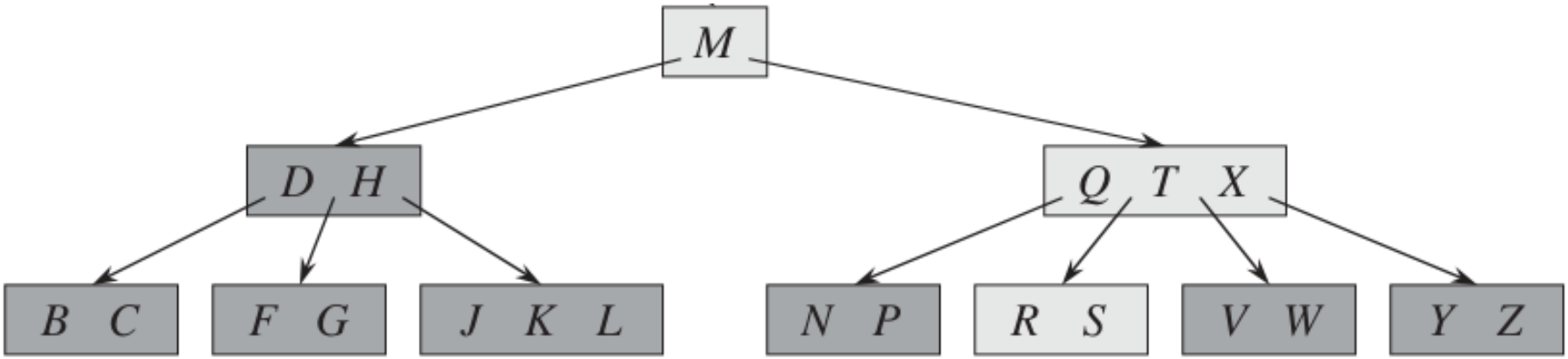
Самое сложное:

4. Задана верхняя и нижняя границы числа ключей в узле (для балансировки дерева). Задается одним числом ***t* – минимальная степень ветвления** (minimum degree) B-дерева

Узел должен содержать

- **Корень**: min 1 ключ (если дерево не пустое)
- ∀ **узлы** кроме корня: min $t-1$ ключей ($\Rightarrow min\ t$ детей)
- ∀ **узлы** включая корень: max $2t-1$ ключей ($\Rightarrow max\ 2t$ детей)

Узел заполнен (full), если содержит $2t-1$ ключей



[1]

Высота B-tree

Количество обращений к диску пропорционально высоте B-дерева

$\log_t$ - большое значение основания

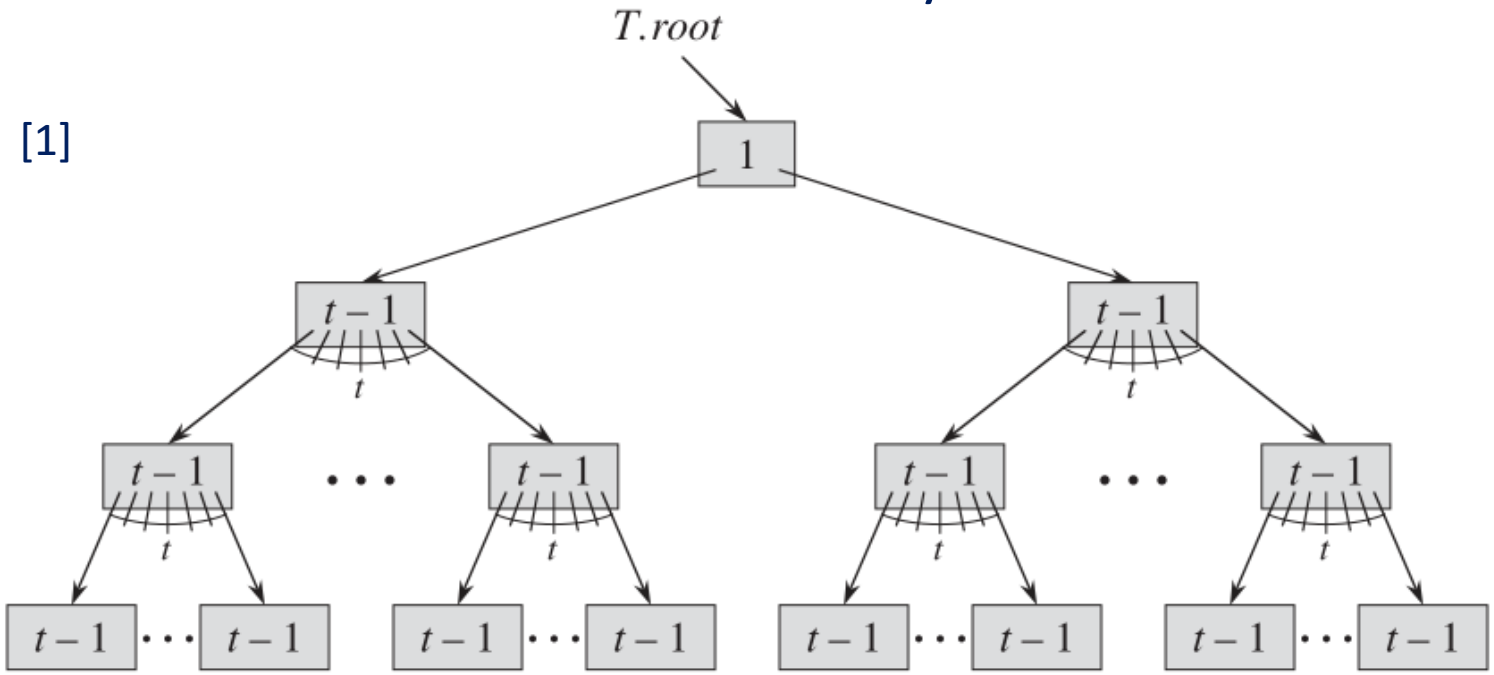
Высота в худшем случае:

$$h \leq \log_t \frac{n+1}{2}$$

Найдем высоту дерева с минимальным числом ключей в узлах

$$\begin{aligned} n &\geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\ &= 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) \\ &= 2t^h - 1 \,. \end{aligned}$$

[1]



depth	number of nodes
0	1
1	2
2	$2t$
3	$2t^2$

Соглашения в псевдокоде

- Disk-Read – чтение узла B-дерева с диска
- Disk-Write – запись узла B-дерева на диск
- Корень B-дерева всегда находится в оперативной памяти и не требует Disk-Read
- Корень дерева может быть изменен и потребовать Disk-Write

Поиск в B-tree

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3       i = i + 1
4  if i ≤ x.n and k == x.key_i
5       return (x, i)
6  elseif x.leaf
7       return NIL
8  else DISK-READ(x.c_i)
9       return B-TREE-SEARCH(x.c_i, k)
```

Одна итерация

Всего

$O(t)$

$O(th) = O(t \log_t n)$

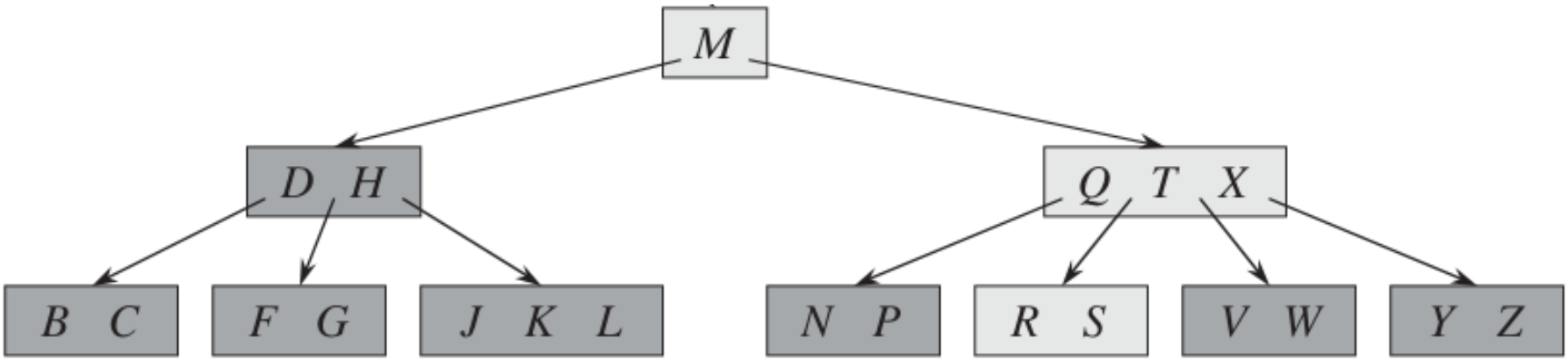
$x.n < 2t$

CPU

I/O

$O(1)$

$O(h) = O(\log_t n)$

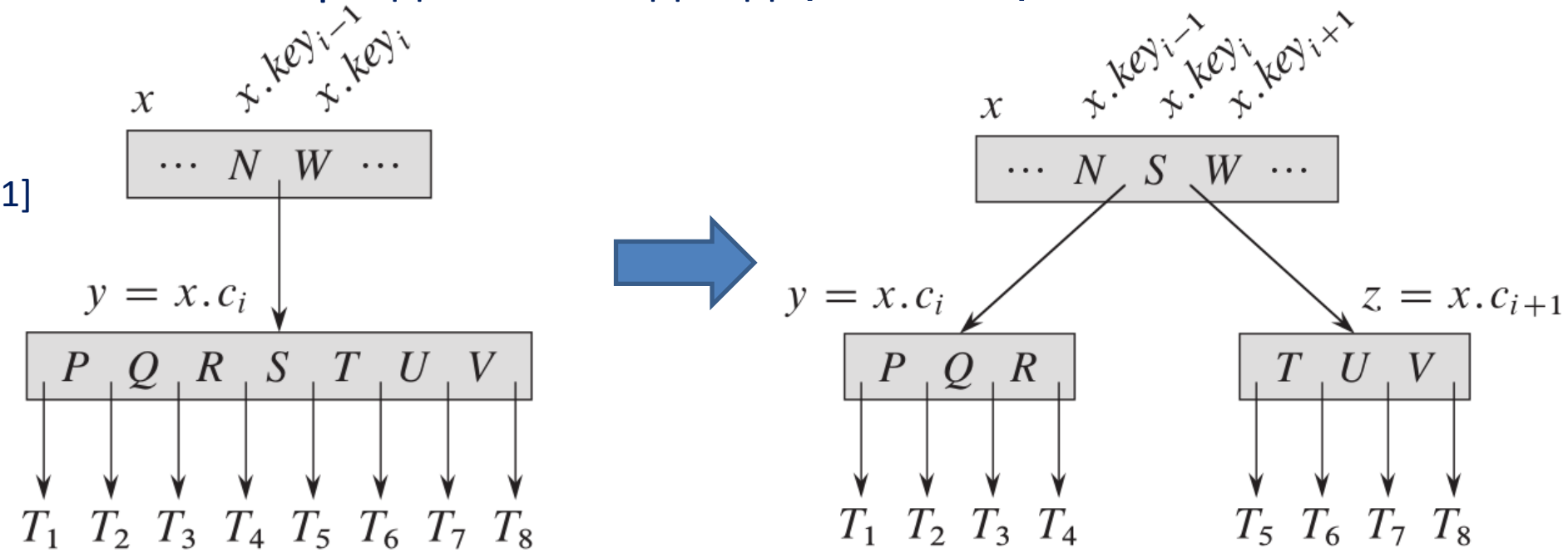


[1]

Вставка в B-tree

- Найти позицию, в которую будет вставлен новый ключ
- Нельзя просто создать лист и вставить в него ключ – такое дерево может оказаться несбалансированным
- Вставлять ключи будем только в существующие листья
- Будем **разбивать один** заполненный узел $\boldsymbol{y}$ с $(\boldsymbol{2t-1})$ ключами на **два узла** ровно с $(\boldsymbol{t-1})$ ключами
- Ключ-Медиана $y.key_t$ перемещается в родительский узел и становится разделителем для двух новых узлов

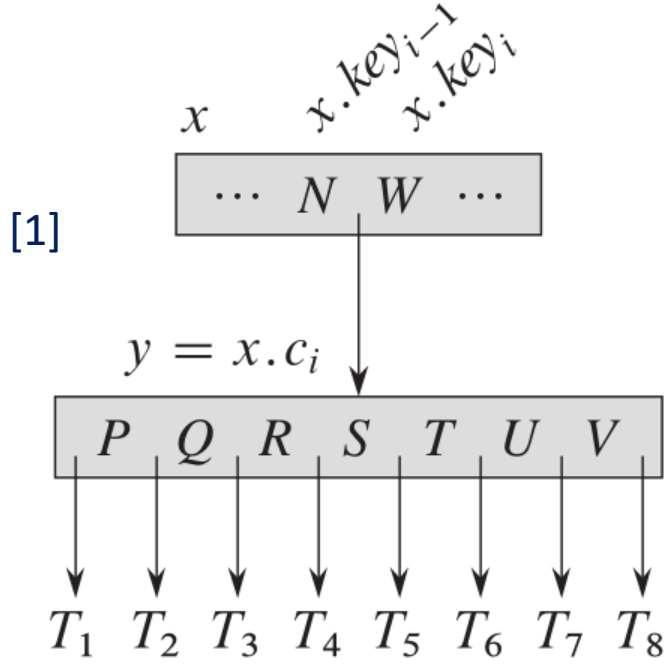
[1]

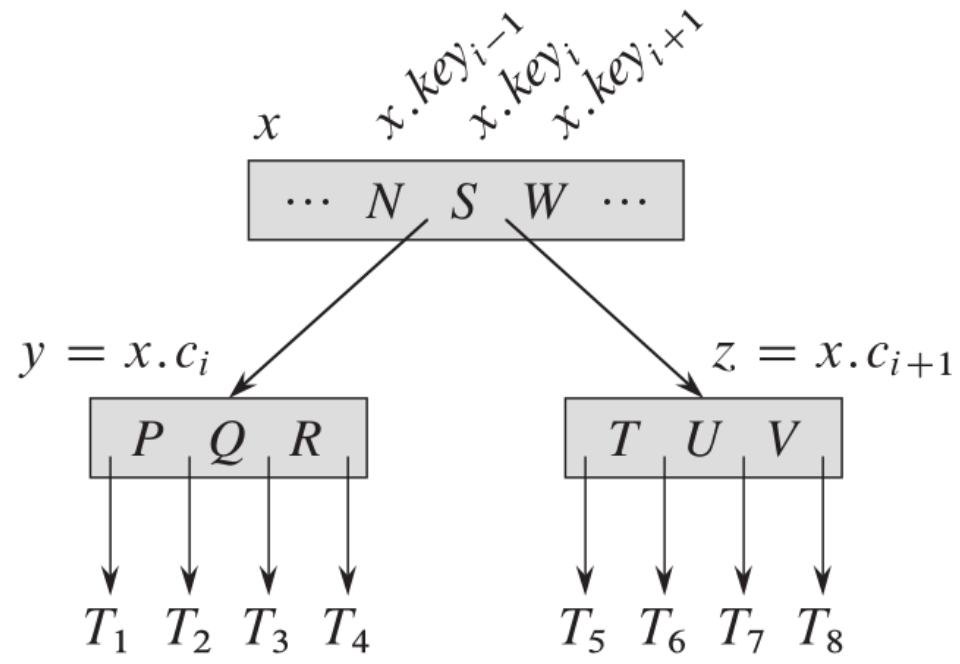


Разбиение узла B-tree

```
B-TREE-SPLIT-CHILD(x, i)
 1  z = ALLOCATE-NODE()
 2  y = x.c_i
 3  z.leaf = y.leaf
 4  z.n = t − 1
 5  for j = 1 to t − 1
 6       z.key_j = y.key_{j+t}
 7  if not y.leaf
 8       for j = 1 to t
 9            z.c_j = y.c_{j+t}
10  y.n = t − 1
11  for j = x.n + 1 downto i + 1
12       x.c_{j+1} = x.c_j
13  x.c_{i+1} = z
14  for j = x.n downto i
15       x.key_{j+1} = x.key_j
16  x.key_i = y.key_t
17  x.n = x.n + 1
18  DISK-WRITE(y)
19  DISK-WRITE(z)
20  DISK-WRITE(x)
```

[1]





Вставка в B-tree

- Выполняется за один нисходящий проход по дереву
- Требует $O(h)$ обращений к диску
- Требует $O(th) = O(tlog_t n)$ времени CPU
- **Высота увеличивается только вверх**
- **Высота h растет очень медленно** (B-дерево эффективно для big number of keys после многих операций вставки, keep I/O small)

```
B-TREE-INSERT(T, k)
 1  r = T.root
 2  if r.n == 2t − 1
 3      s = ALLOCATE-NODE()
 4      T.root = s
 5      s.leaf = FALSE
 6      s.n = 0
 7      s.c_1 = r
 8      B-TREE-SPLIT-CHILD(s, 1)
 9      B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)
```

[1]

Внимание: мы не выясняем нужно ли разбить узел, в поддерево которого будет вставлен ключ, мы разбиваем все встреченные нами по пути заполненные узлы при спуске от корня к листьям во время поиска позиции для нового ключа

```
B-TREE-INSERT-NONFULL(x, k)
 1  i = x.n
 2  if x.leaf
 3       while i ≥ 1 and k < x.key_i
 4            x.key_{i+1} = x.key_i
 5            i = i − 1
 6       x.key_{i+1} = k
 7       x.n = x.n + 1
 8       DISK-WRITE(x)
 9  else while i ≥ 1 and k < x.key_i
10            i = i − 1
11       i = i + 1
12       DISK-READ(x.c_i)
13       if x.c_i.n == 2t − 1
14            B-TREE-SPLIT-CHILD(x, i)
15            if k > x.key_i
16                 i = i + 1
17       B-TREE-INSERT-NONFULL(x.c_i, k)
```

Вставка ключа в B-tree

Внимание: вставляем всегда только в лист

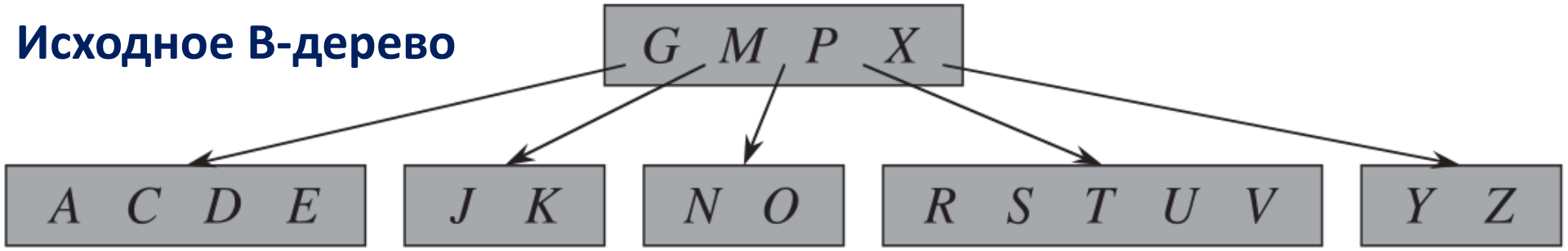
Ключи во внутренних узлах появляются только в результате подъема ключей вверх по дереву

[1]

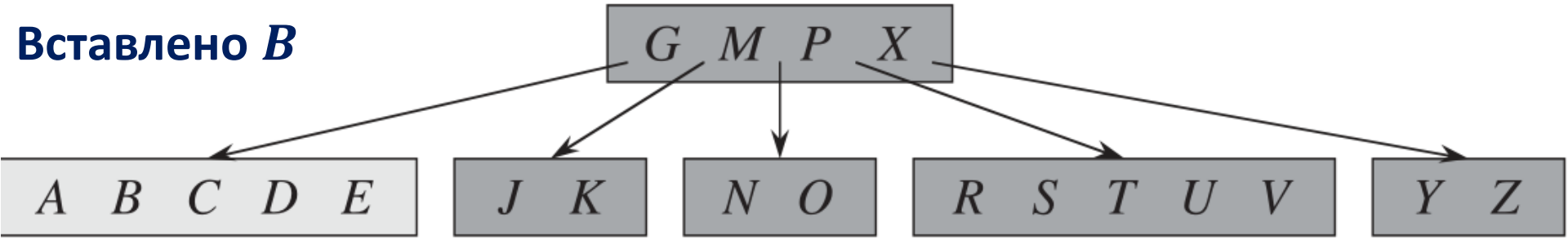
Вставка в B-tree: примеры

B-дерево с минимальной степенью $t = 3 \Rightarrow$
содержит не более $2t - 1 = 5$ ключей

Исходное B-дерево



Вставлено B



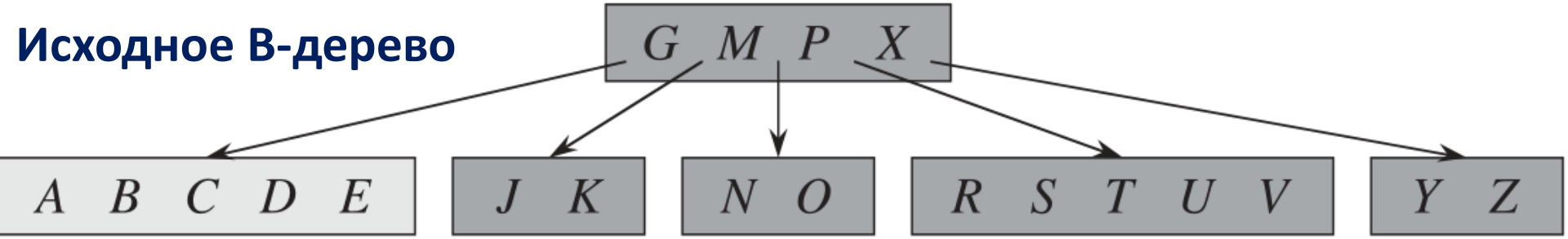
Вставка в лист

[1]

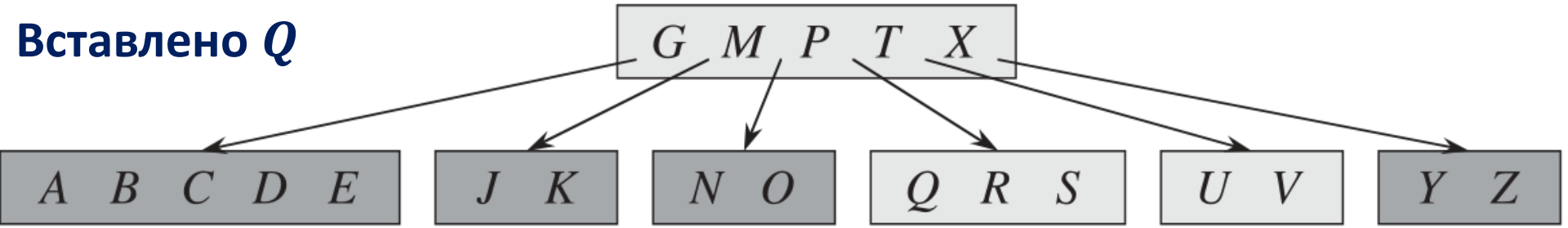
Вставка в B-tree: примеры

B-дерево с минимальной степенью $t = 3 \Rightarrow$
содержит не более $2t - 1 = 5$ ключей

Исходное B-дерево



Вставлено Q



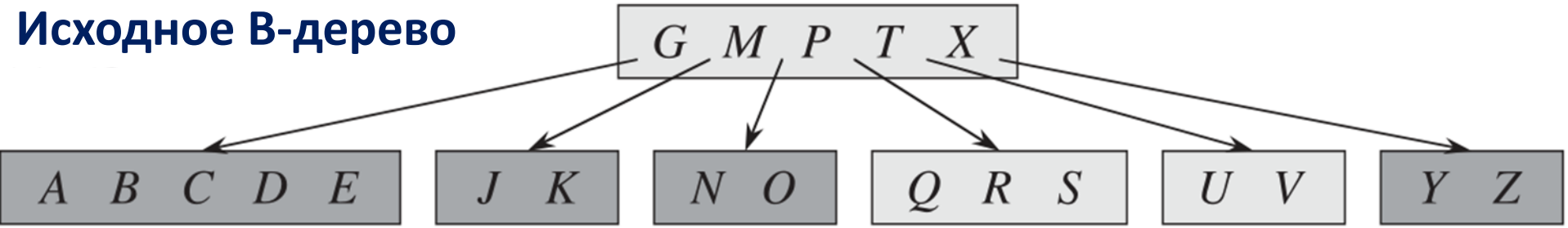
RSTUV разбили на два узла,
T перенесли вверх

[1]

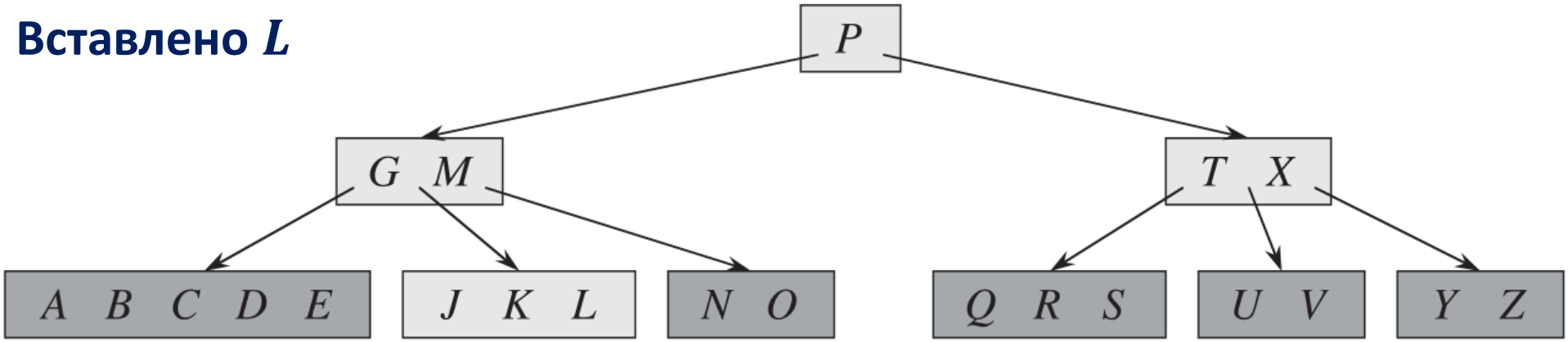
Вставка в B-tree: примеры

B-дерево с минимальной степенью $t = 3 \Rightarrow$
содержит не более $2t - 1 = 5$ ключей

Исходное B-дерево



Вставлено L



Корень был разбит на два узла,
высота дерева выросла ВВЕРХ

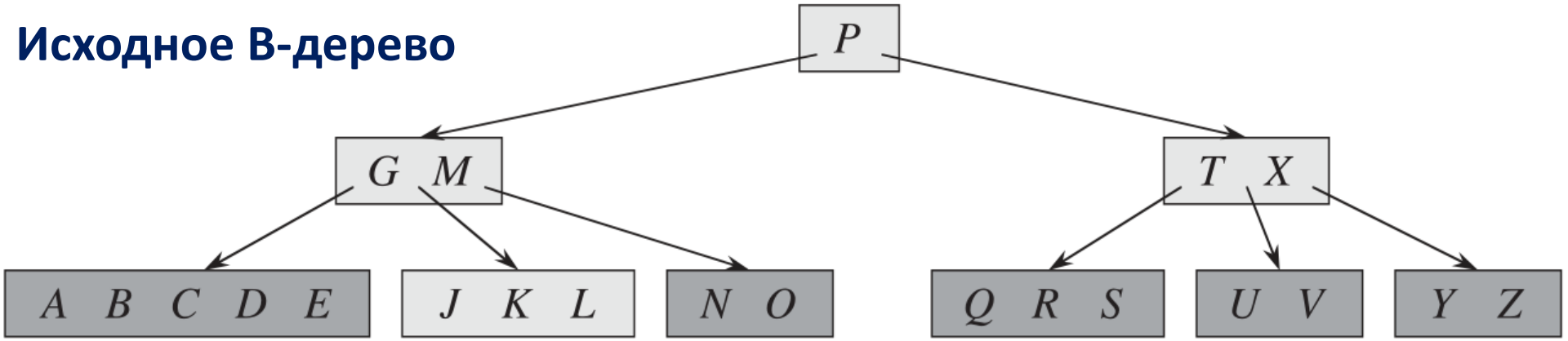
[1]

Вставка в B-tree: примеры

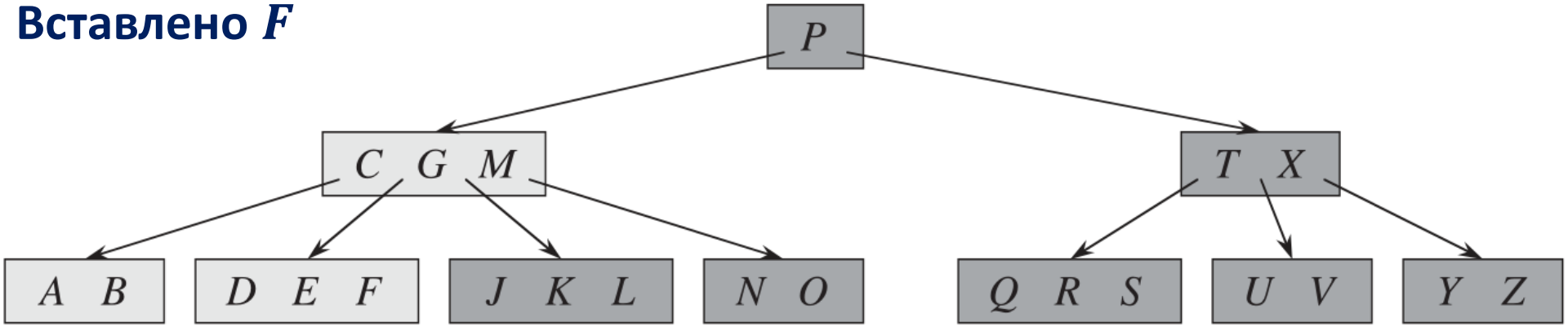
B-дерево с минимальной степенью $t = 3 \Rightarrow$

содержит не более $2t - 1 = 5$ ключей

Исходное B-дерево



Вставлено F

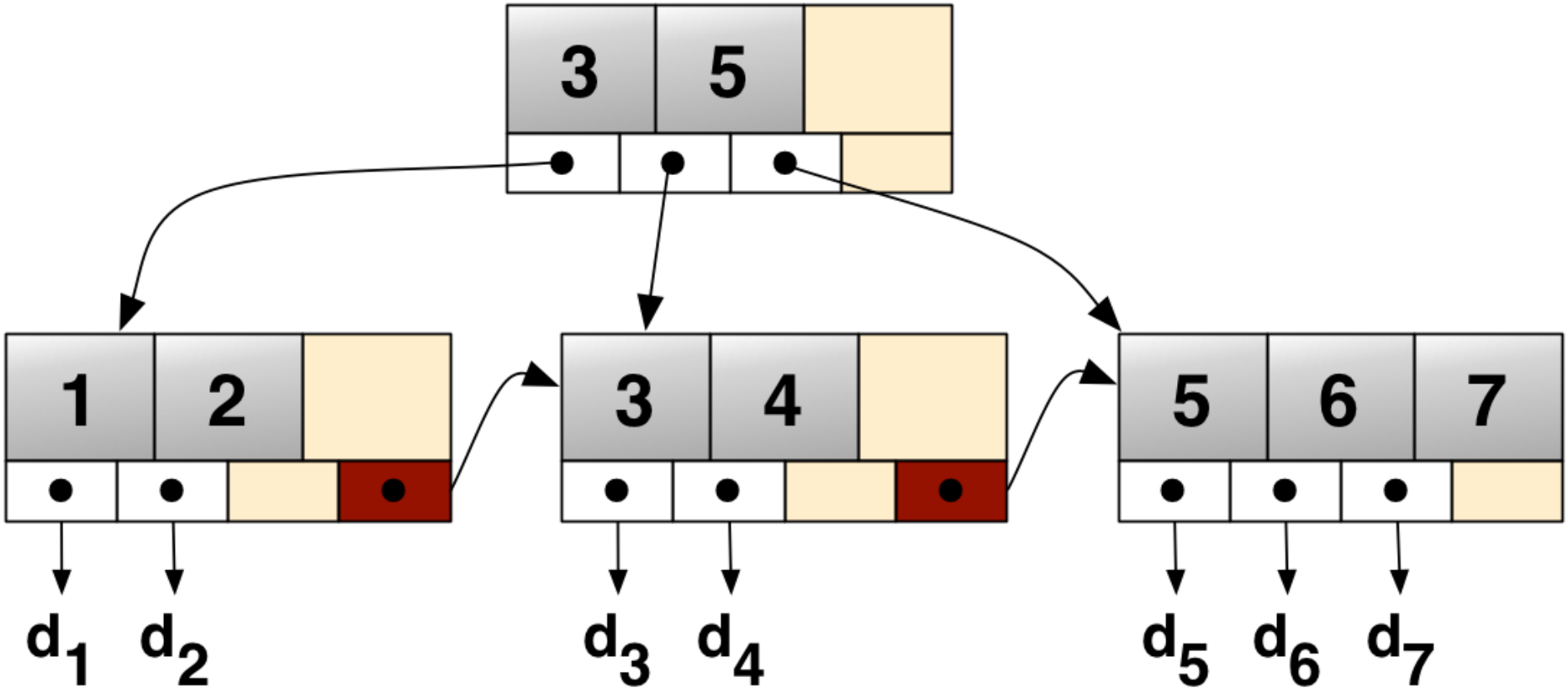


ABCDE разбили, F вставили в правый узел

[1]

B+-дерево: классическая разновидность B-tree

Вся сопутствующая информация хранится в листьях, во внутренних узлах только указатели на дочерние узлы: удается получить максимальную степень ветвления.



https://en.wikipedia.org/wiki/B%2B_tree

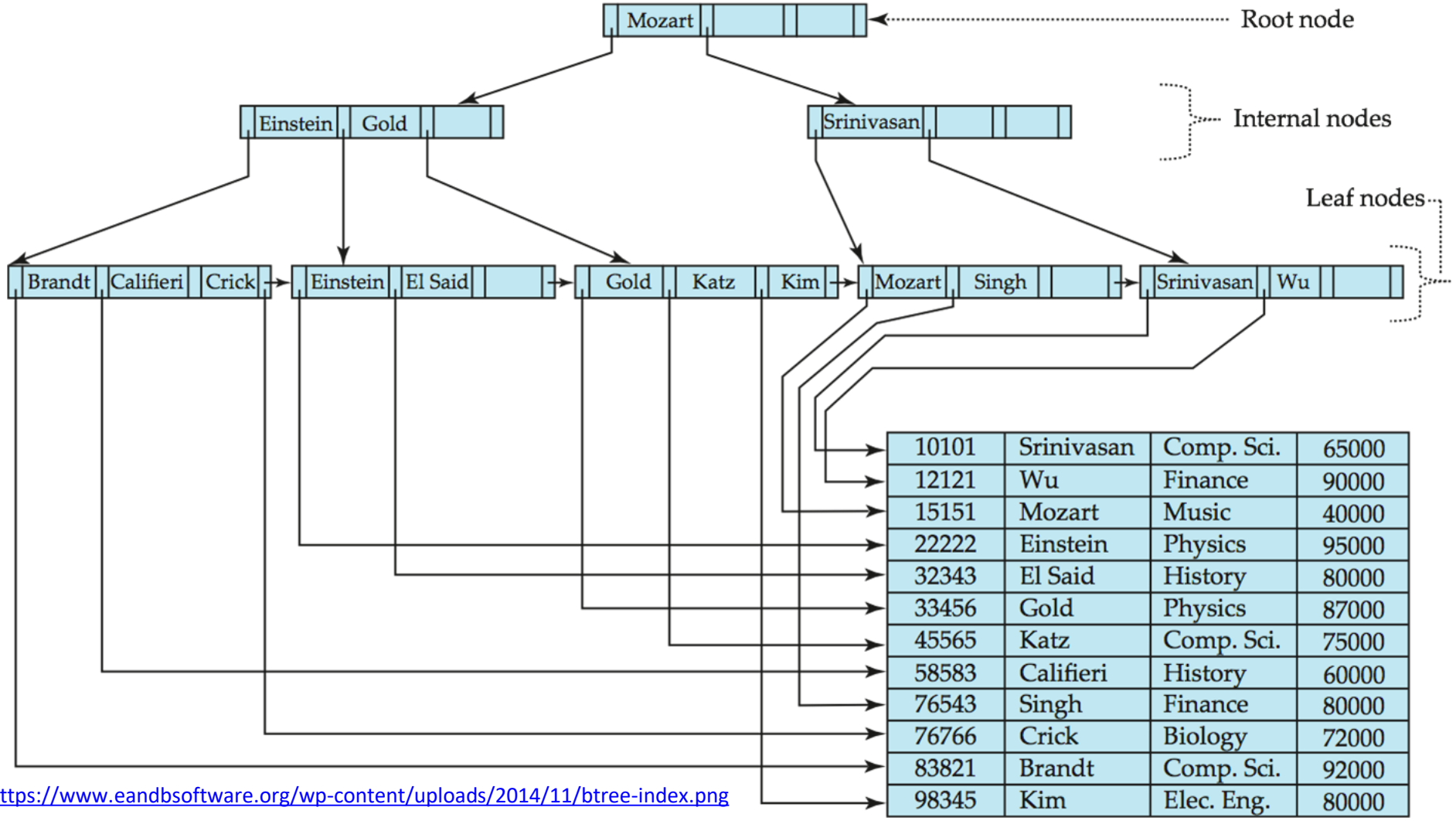
Примеры использования

- Файловые системы — NTFS, ReiserFS, NSS, XFS, JFS, ReFS и BFS - индексирование метаданных;
- Реляционные системы управления базами данных – DB2, Informix, Microsoft SQL Server, Oracle Database, Adaptive Server Enterprise и SQLite – табличные индексы
- NoSQL-СУБД – CouchDB, MongoDB

https://ru.wikipedia.org/wiki/B%E2%81%BA-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE

Примеры использования: индекс в БД

- Отсортирована обычно только колонка с первичным ключом, остальные как правило не отсортированы ⇒ поиск за $O(n)$ I/O
- Создаем **индекс** для заданной колонки ⇒ поиск за $O(h)$ I/O



https://www.eandbsoftware.org/wp-content/uploads/2014/11/btree-index.png

Список литературы

1. Т. *Кормен*, Ч. Лейзерсон, Р. Ривест, К. Штайн - *Алгоритмы*. Построение и анализ.

Некоторые рисунки в данной презентации из [1]

Резюме: СД для вторичной памяти

1. «Постановка задачи»
2. Особенности вторичной памяти: HDD, SSD, облако
3. Проблемы СД для оперативной памяти
4. Особенности оценки времени работы
5. Классическое B-дерево (SIGMOD)
6. Идеи устройства B-tree
7. Определение B-tree
8. Высота B-tree
9. Поиск в B-tree
10. Вставка в B-tree
11. Разбиение узлов и автобалансировка
12. Примеры работы алгоритма вставки
13. Примеры применения B-tree
14. Резюме лекции

NATIONAL RESEARCH UNIVERSITY

Благодарю за внимание!

Рамон Антонио Родригес Залепинос
arodriges@hse.ru