Лекция 5

Индексы

Индексы

- Упорядоченные индексы
  - Индексы в виде B-дерева
  - Индексы в виде B^+-дерева
- Хеш-индексы (в следующей лекции)
- Bitmap индексы

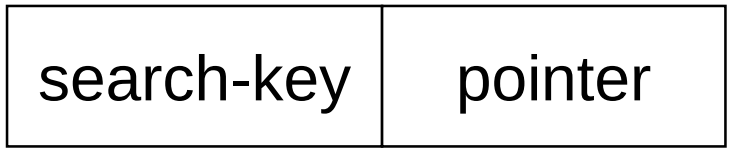
Основные понятия

Индекс БД – это структура данных, которая повышает скорость операций поиска данных в таблице БД за счет дополнительных операций записи и хранения для поддержания структуры данных индекса.

Механизмы индексации используются для ускорения доступа к данным

- Например, каталог по авторам в библиотеке

- **Ключ поиска** – атрибут (или набор атрибутов), используемый для поиска записей в файле.
- **Файл индекса** состоит из записей (называемые записями индекса) следующей формы:



Основные понятия

Файлы индекса обычно гораздо меньше оригинального файла

Два базовых типа индексов:

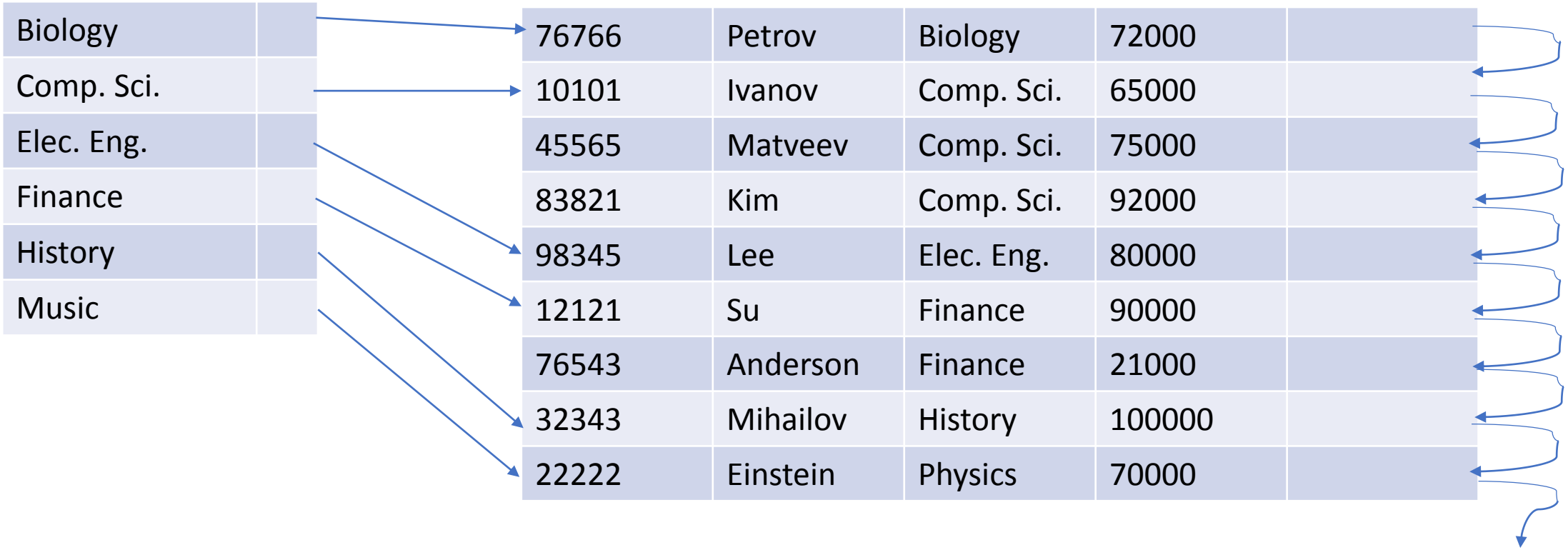
- **Упорядоченные индексы:** записи индекса хранятся в порядке, отсортированных по ключам поиска
  - (далее сегодня только про них)
- **Хэш индексы:** ключи поиска распределены равномерно по бакетам (bucket), которые используют хэш-функции.

Файлы с плотными индексами

- **Плотный (Dense) индекс**— для каждого значения ключа поиска появляется запись индекса в файле
- Например, атрибут *ID* отношения *instructor*

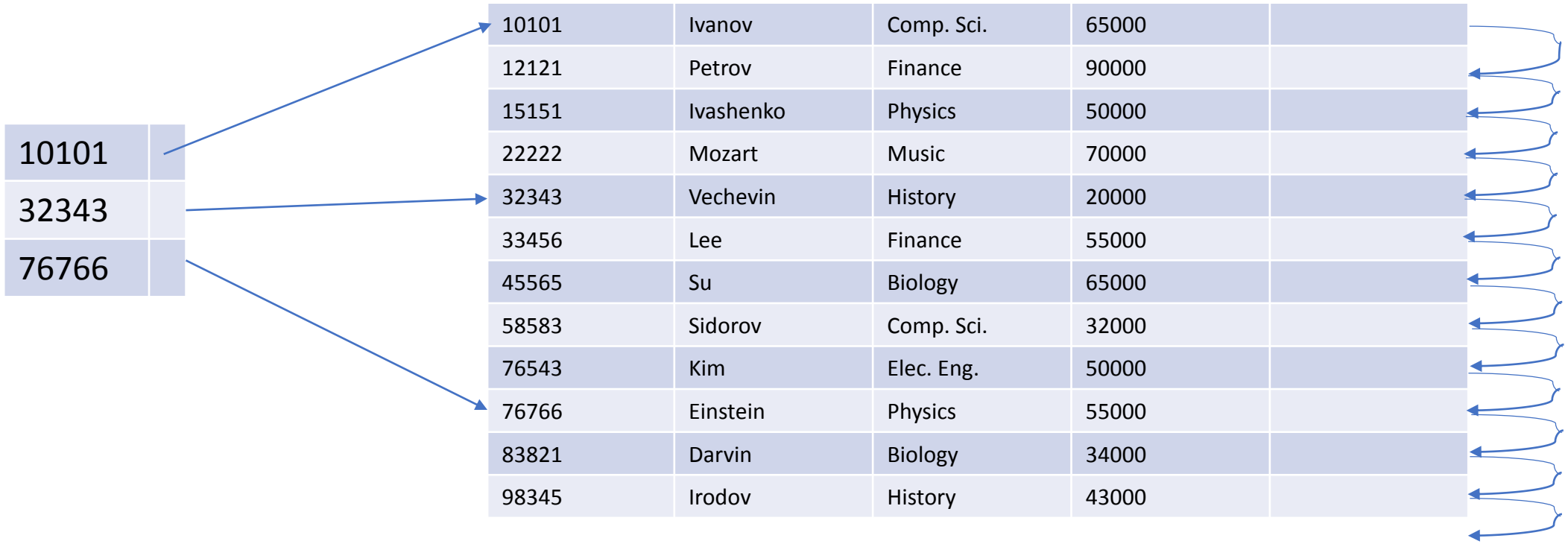
Файлы с плотными индексами

- Плотный индекс *dept_name*, и файл *instructor,* отсортированный по *dept_name*



Файлы с разреженными индексами

- **Разреженный индекс**: содержит записи индекса только для некоторых значений ключей поиска
  - Применимо, когда записи последовательно отсортированы по ключу поиска
- Чтобы найти запись с ключом поиска *K*:
  - Найти запись индекса с наибольшим значением ключа поиска <= *K*
  - Производится последовательный поиск в файле, начиная с записи, на которую ссылает запись индекса



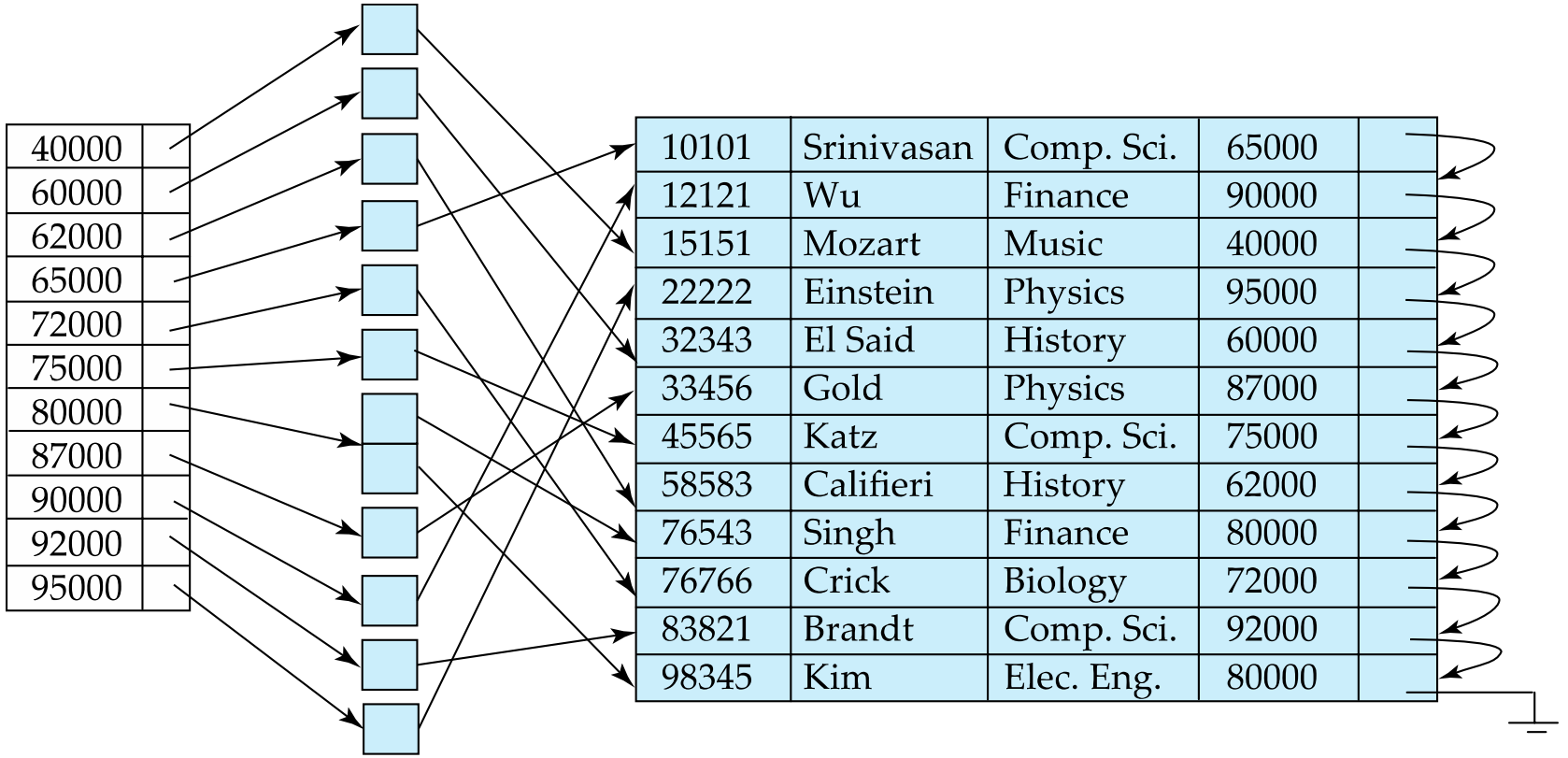
Файлы с разреженными индексами

- Сравнивая с плотными индексами:
  - Меньший размер и меньше накладных расходов для вставок и удалений
  - Обычно медленней, чем плотный индекс для поиска записей

Первичный и вторичный индекс

- **Первичный индекс** – уникальный индекс по полю первичного ключа.
- **Вторичный индекс** – индекс по другим полям (кроме поля первичного ключа).

Пример вторичного индекса



Вторичный индекс по полю on *salary* для отношения *salary*

- Записки индекса указывают на bucket, который содержит указатели для всех записей с определенным ключом поиска.
- Вторичный индекс должен быть плотным

Первичный и вторичный индекс

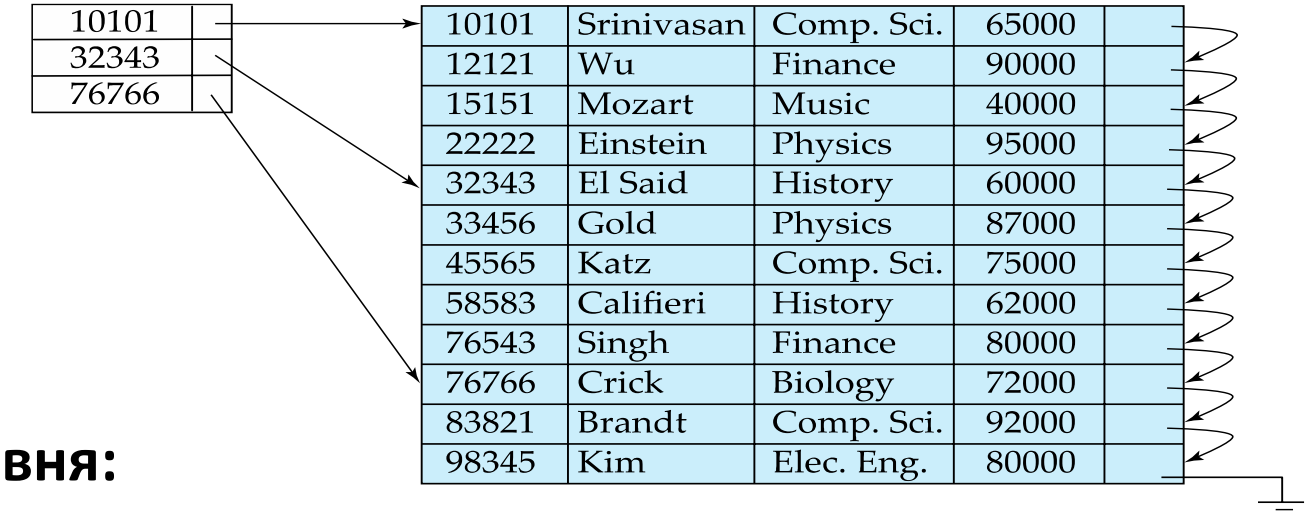
- Индексы предоставляют существенные преимущества, когда осуществляется поиск для записей.
- НО: Индексы дают накладные расходы при изменении базы данных
  - когда запись вставлена или удалена, каждый индекс в отношении должен быть обновлен
  - при обновлении записи любой индекс по обновленному атрибуту должен быть обновлен.
- Последовательное сканирование при использовании первичного индекса довольно эффективно, но последовательное сканирование вторичного индекса довольно дорогая операция на магнитном диске
  - Каждый доступ к записи может осуществлять забор нового блока из диска
  - Каждый забор блока с диска занимает время

Обновление индекса: Вставка

- **Вставка индекса одного уровня:**
  - Осуществляется поиск по индексу для ключа вставки.
  - **Плотные индексы** – если значение ключа поиска отсутствует в индексе, осуществляется вставка
    - Индексы поддерживаются в виде последовательных файлов
    - Требуется создание места для новой записи, возможно, потребуются overflow блоки.
- **Разреженные индексы** – если в индексе хранится запись для каждого блока файла, в индекс не нужно вносить никаких изменений, если только не создан новый блок.
    - Если создается новый блок, первое значение ключа поиска, появляющееся в новом блоке, вставляется в индекс.

Обновление индекса: Удаление

Если удаленная запись была единственной записью в файле для определенного ключа поиска, ключ поиска также удаляется из индекса



- **Удаление записи индекса из 1 уровня:**
  - **Плотный индекс** – удаление ключа поиска аналогично удалению записи из отношения
  - **Разреженный индекс** –
    - если запись для ключа поиска существует в индексе, она удаляется путем замены записи в индексе следующим значением ключа поиска в файле (в порядке ключа отношения).
    - Если следующее значение ключа поиска уже имеет запись в индексе, эта запись удаляется, а не заменяется.

Индексы по составным ключам

- **Составной поисковый ключ**
  - Например, индекс по отношению *instructor* по атрибутам (*name, ID*)
  - Значения хранятся лексикографически
    - Например, (John, 12121) < (John, 13514)

    и (John, 13514) < (Peter, 11223)
  - Могут быть запрос по *name*, или по (*name, ID*)

Файл индекса в виде B^+-дерева

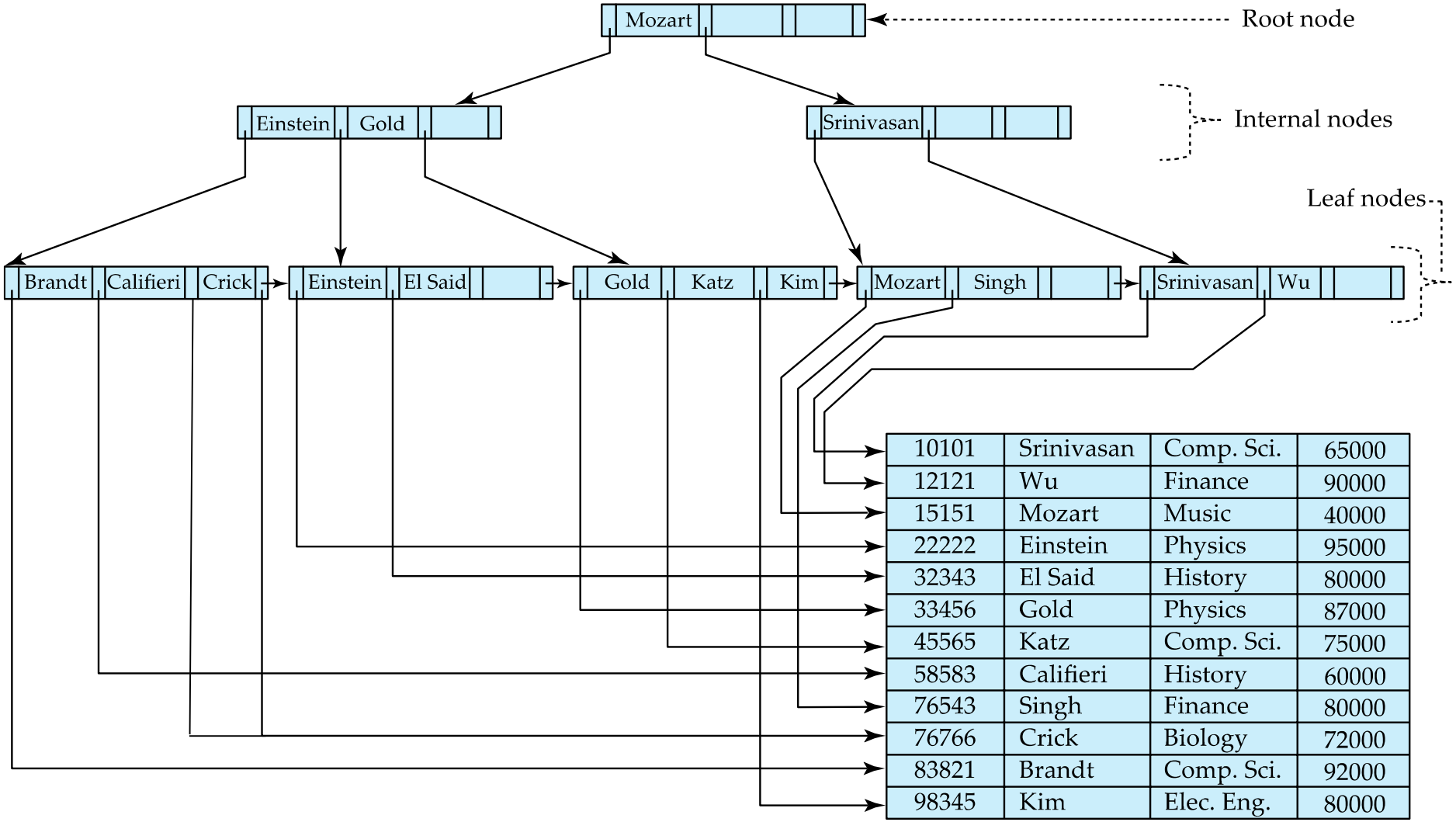
- Недостатки последовательных файлов индекса:
  - Производительность ухудшается при росте файла, так как могут быть созданы overflow блоки.
  - Требуется периодическая реорганизация всего файла

- Далее см. B+ Деревья

Файл индекса в виде B^+-дерева

- Недостатки последовательных файлов индекса:
  - Производительность ухудшается при росте файла, так как могут быть созданы overflow блоки.
  - Требуется периодическая реорганизация всего файла
- Преимущество файла в виде B^+-дерева:
  - Автоматически реорганизуется при небольших, локальных изменениях при вставках и удалениях
  - Реорганизация всего файла не требуется для управления производительностью.
- (Небольшой) недостаток B^+-дерева:
  - Дополнительные издержки на вставку и удаление, есть накладные расходы для места
- Обычно преимущества B^+-деревьев превышают недостатки
  - B^+-деревья часто используются на данный момент

B^+-Деревья. Пример.



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Файл индекса в виде B^+-дерева

B^+-дерево – корневое дерево со следующими свойствами:

- Все пути от корня до листа одинакового размера
- Каждая вершина, которая не корень и не лист имеет от $\lceil n/2 \rceil$ до n детей.
- Лист содержит от $\lceil (n–1)/2 \rceil$ до $n–1$ значений
- Специальные случаи:
  - Если корень не лист, у него есть как минимум, 2 ребенка.
  - Если корень является листом, то у него от 0 до $(n–1)$ значений.

Структура вершин B^+-Дерева

- Формат вершины

  - K_i – ключи поиска
  - P_i – указатели на детей (для не листовых вершин) или указатели на записи или buckets записей (для листовых вершин).
- Ключи поиска в вершине отсортированы

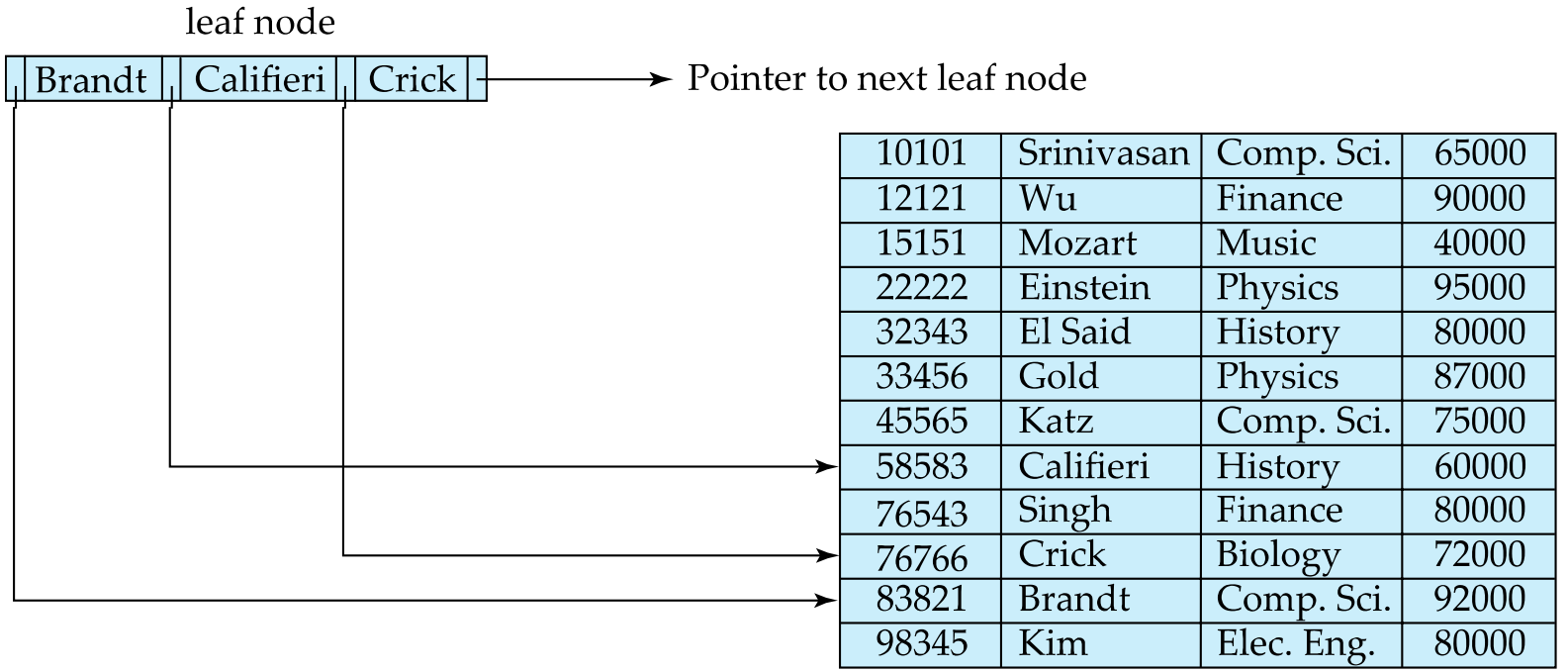
$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

(Предположение: отсутствуют ключи дубликаты)

Листья в B$^+$-Деревьях

Свойства листов:

- Для i = 1, 2, . . ., n–1, указатель P_i указывает на запись файла со значениям ключа поиска K_i,
- Если L_i, L_j – листовые вершины и $i < j$, значения ключей поиска L_i меньше, чем L_j
- P_n указывает на следующий лист в порядке ключа поиска



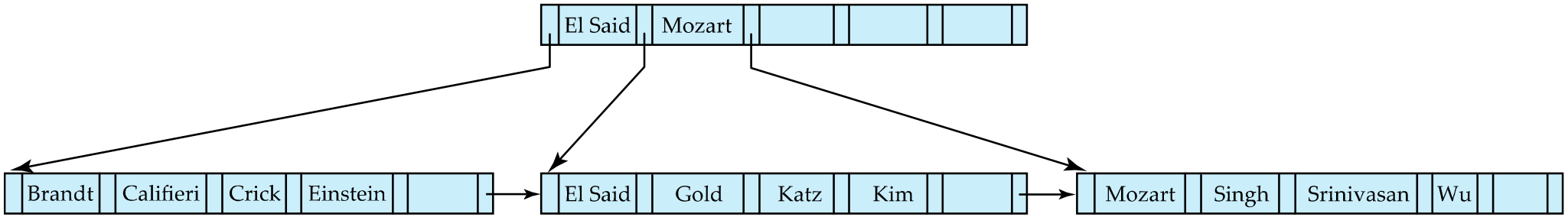
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Не листовые вершины в B^+-деревьях

- Не листовые узлы формируют многоуровневый разреженный индекс на конечных узлах. Для не листовой вершине с *m* указателями:
  - Все ключи поиска в поддереве, на который указывает P_1 меньше, чем K_1
  - Для $2 \leq i \leq n - 1$, все ключи поиска в поддереве, на который указывают P_i имеют значения больше или равно, чем K_{i-1} и меньше, чем K_i
  - Все ключи поиска в поддереве, на который указывает P_n , имеют значения больше или равные, чем K_{n-1}

P_1	K_1	P_2	$\cdots$	P_{n-1}	K_{n-1}	P_n

Пример B^+-дерева



B^+-дерево для файла *instructor* (n = 6)

- Листовых вершин должно быть от 3 до 5 ($\lceil (n-1)/2 \rceil$ and $n-1$, with n = 6).
- Не листьевые и некорневые должны быть от 3 до 6 детей ($\lceil (n/2 \rceil$ and n with n =6).
- Корень должен содержать хотя бы 2 ребенка.

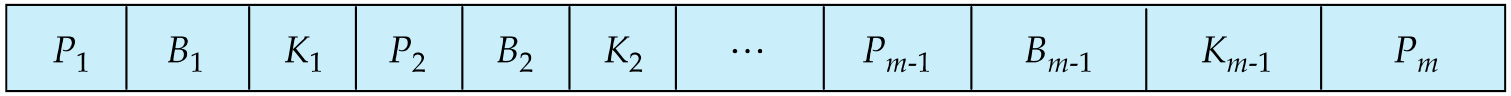
Наблюдения о B^+-деревьях

- Так как межузловые связи происходят через указатели, то “логически” близкие блоки не должны быть “физически” близкими.
- Не листьевые вершины B^+-дерева формируют иерархию разреженных индексов.
- B^+-дерево содержит относительно небольшое количество уровней
    - Уровни ниже корня имеют хотя бы $2*\lceil n/2 \rceil$ значений
    - Следующий уровень содержит в себе $2*\lceil n/2 \rceil * \lceil n/2 \rceil$ значений
    - .. т.д.
  - Если в файле значений ключей поиска *K*, высота дерева не более, чем $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$
  - Таким образом, поиск был эффективным
- Вставки и удаления для основного файла могут быть эффективно обработаны, так как индекс реконструируется за логарифмическое время

Файлы индексов в виде B-дерева

- Похоже на B+-деревья, но B-деревья позволяют ключам поиска возникать только один раз; исключает избыточное хранение ключей поиска.
- Ключи поиска в не листовых атрибутах нигде больше не возникают; требуется доп. указатель для каждого не ключевой вершины
- Обобщенная вершина листа B-дерева

(a)

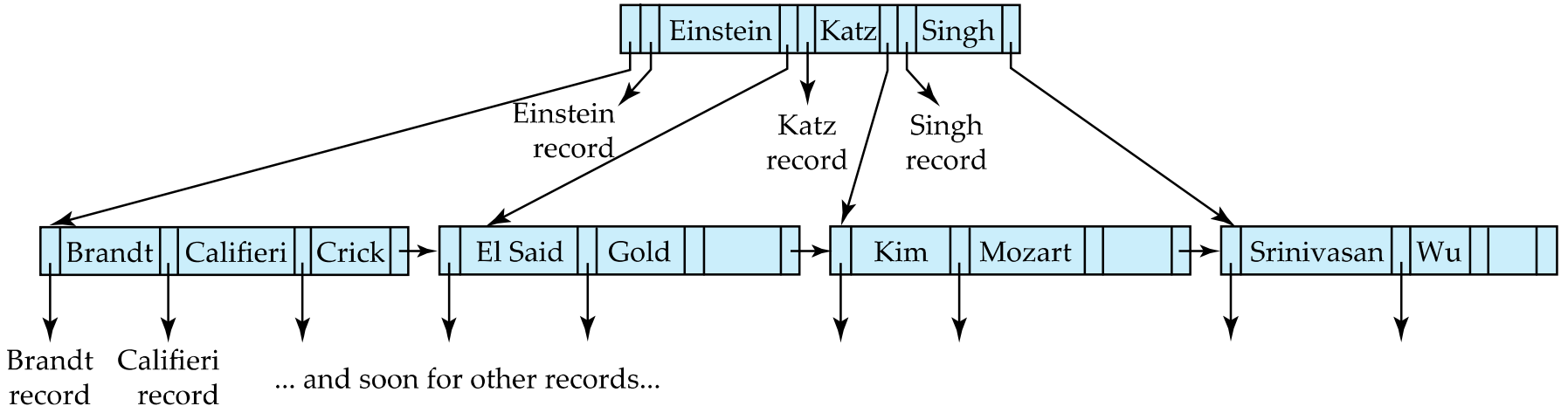


(b)

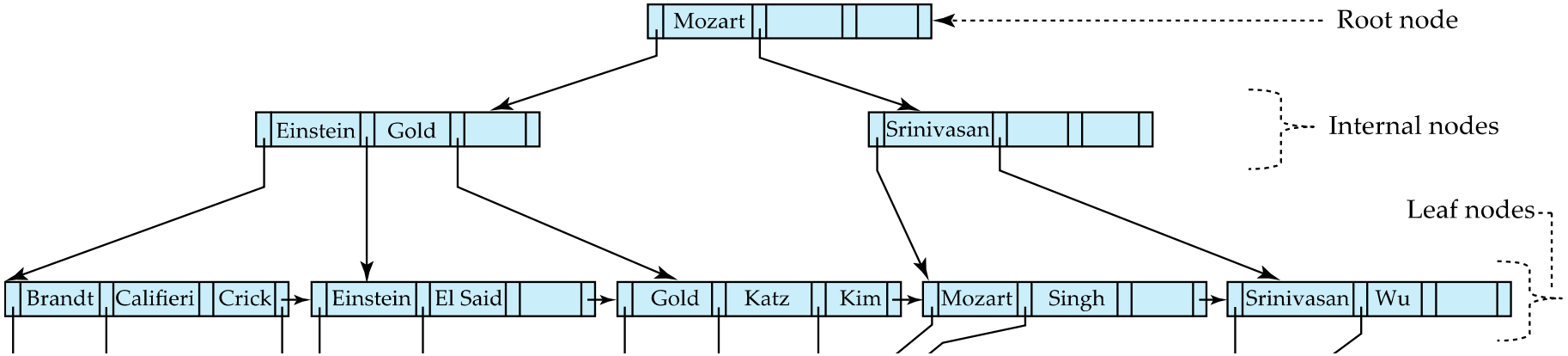
Файлы индексов в виде B-дерева

- Преимущества индексов в виде B-Дерева:
  - Может быть меньшее число вершин, чем в B^+-Дереве.
  - Иногда можно найти ключ поиска до листа.
- Недостатки индексов в виде B-Дерева:
  - Только небольшое число ключей поиска находится до листа
  - Вставка и удаление более сложна, чем в B^+-Деревьях
  - Реализация сложнее, чем B^+-Деревья.
- Обычно преимущества B-Деревьев не превышают недостатки

Файлы индексов в виде B-дерева



B-дерево(выше) и B+-дерево(ниже) для тех же данных



Bitmap индексы

- Bitmap индексы – специальный тип индексов, разработанный специально для эффективных запросов по нескольким ключам
- Записи в отношении предполагаются пронумерованными последовательно от, например, 0
  - Имея номер *n* довольно просто получить запись *n*
    - Особенно, если записи фикс. длины
- Применимо, если атрибуты содержат в себе малое число уникальных значений
  - Например, пол, город, страна
  - Или уровень дохода (0-9999, 10000-19999, 20000-50000, 50000-бесконечность)
- Bitmap (битовая карта) – это просто массив битов

Bitmap индексы

- В простейшей форме bitmap индекс по атрибуту содержит битовую карту для каждого значения атрибута
  - Битовая карта содержит столько битов сколько записей
  - В битовой карте значения v, the bit для записи 1 запись содержит значения v для атрибута, иначе - 0

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Bitmap индексы

- Bitmap индексы полезны для запросов по нескольким атрибутам
  - не всегда полезны для запросов с одним атрибутом
- Запросы работают с
  - Пересечением (and)
  - Объединением (or)
- Каждая операция берет две битовые карты того же размера и применяет битовые операции
  - Пример. 100110 AND 110011 = 100010

    100110 OR 110011 = 110111

    NOT 100110 = 011001
  - Мужчины с уровнем дохода L1:10010 AND 10100 = 10000
    - Затем можно получить необходимые кортежи
    - Подсчет количества очень быстрый