



# Методы доступа



Последовательное сканирование

Индексное сканирование

Сканирование битовой карты

Исключительно индексное сканирование

Сортировка, ограничение, группировка

Команда explain

# Sequential scan

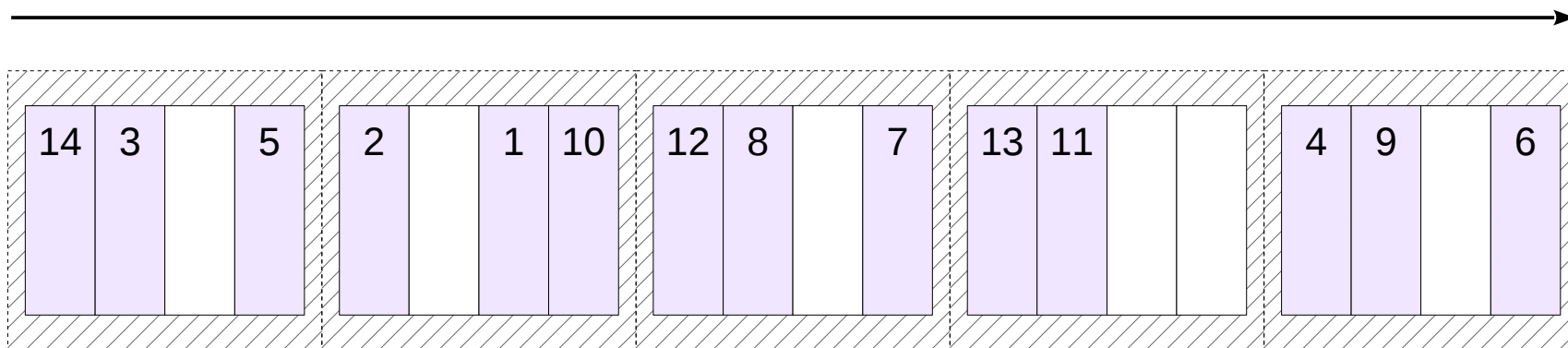
## Чтение всей таблицы

последовательно прочитать все страницы  
(в кэше используется буферное кольцо)

проверить видимость версий строк и получить данные

Эффективно для получения всех данных

Не эффективно для небольшой выборки



## Индекс

вспомогательная структура во внешней памяти  
сопоставляет ключи и идентификаторы строк таблицы

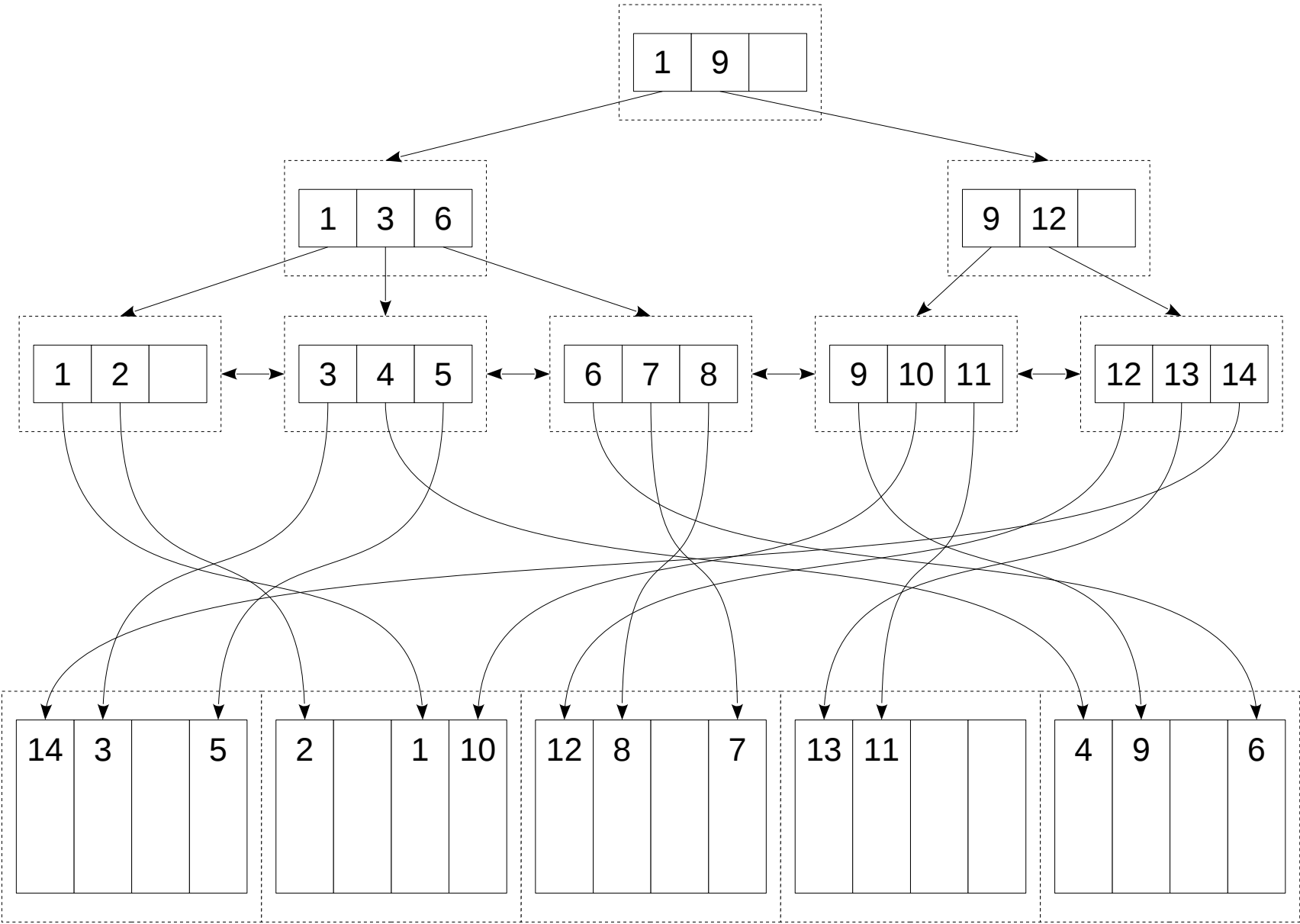
## Устройство: дерево поиска

сбалансированное  
сильно ветвистое

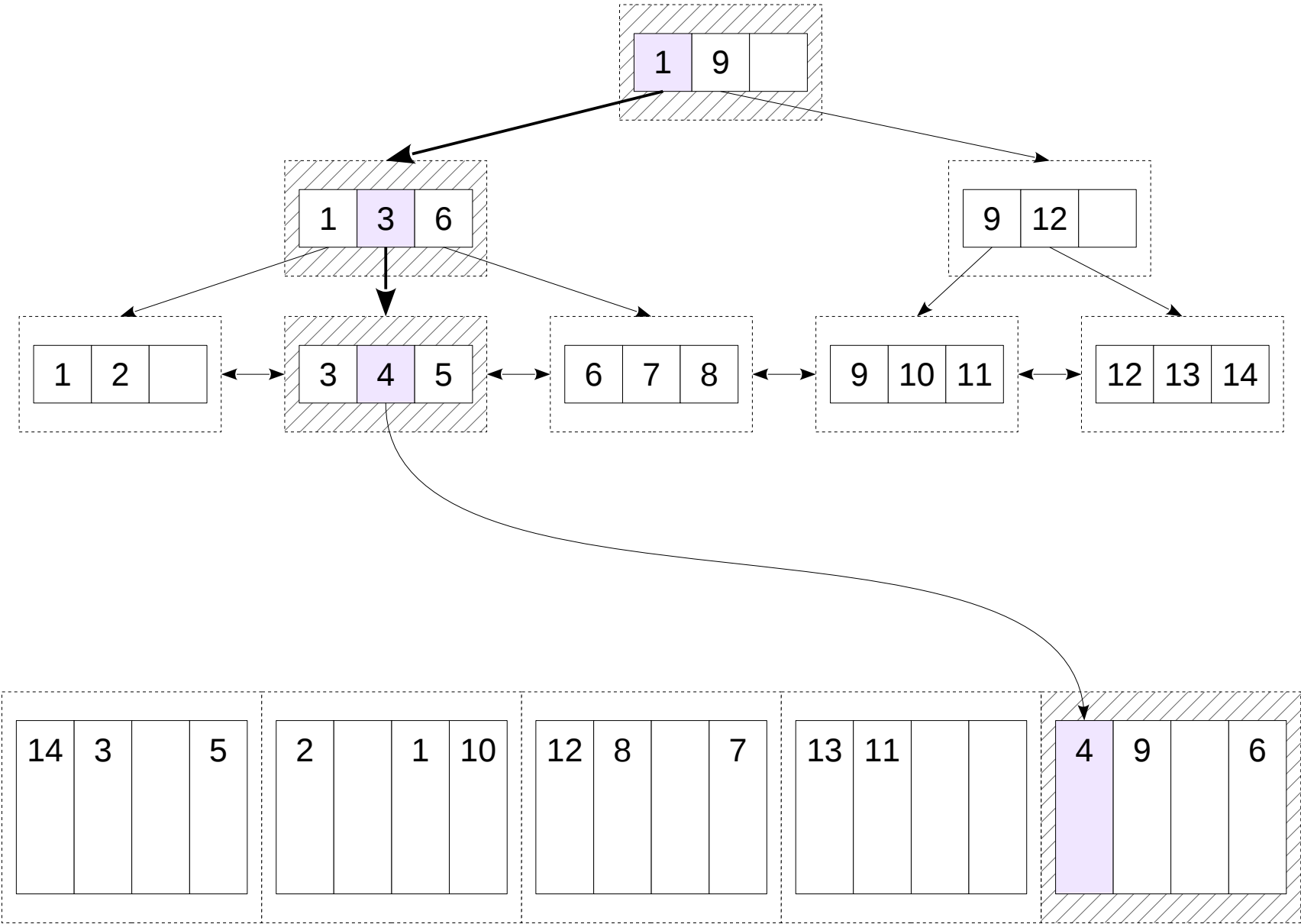
## Использование

только сортируемые типы данных (операции «больше», «меньше»)  
ускорение доступа  
поддержка ограничений целостности

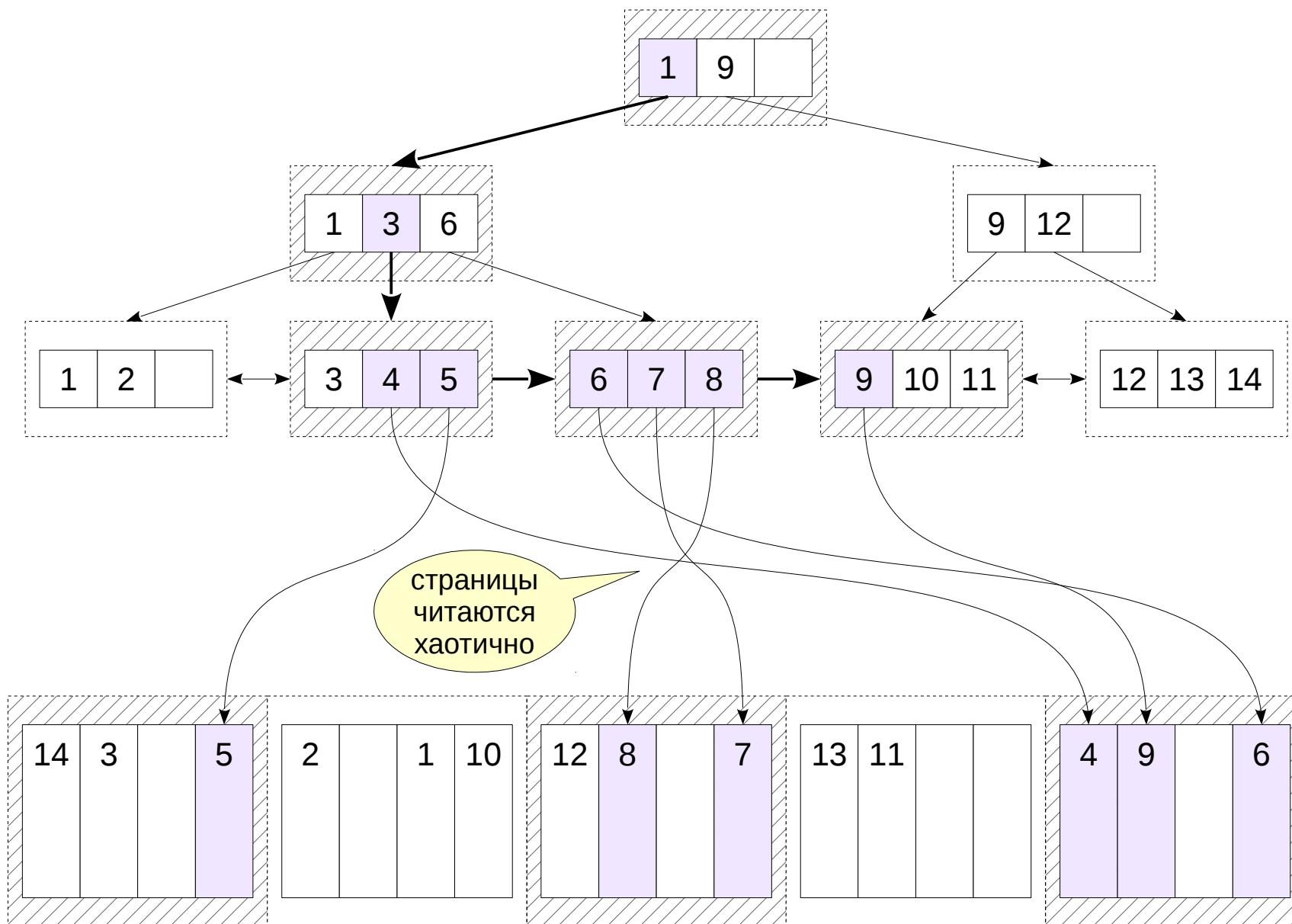
# B-дерево



# Index scan: одно значение



# Index scan: диапазон



## Получение одного значения — эффективно

пройти от корня дерева до листовой страницы,  
найти в ней подходящий ключ

прочитать страницу таблицы,  
проверить видимость версии строки таблицы и получить данные

## Получение всех значений — не эффективно

пройти от корня дерева до листовой страницы

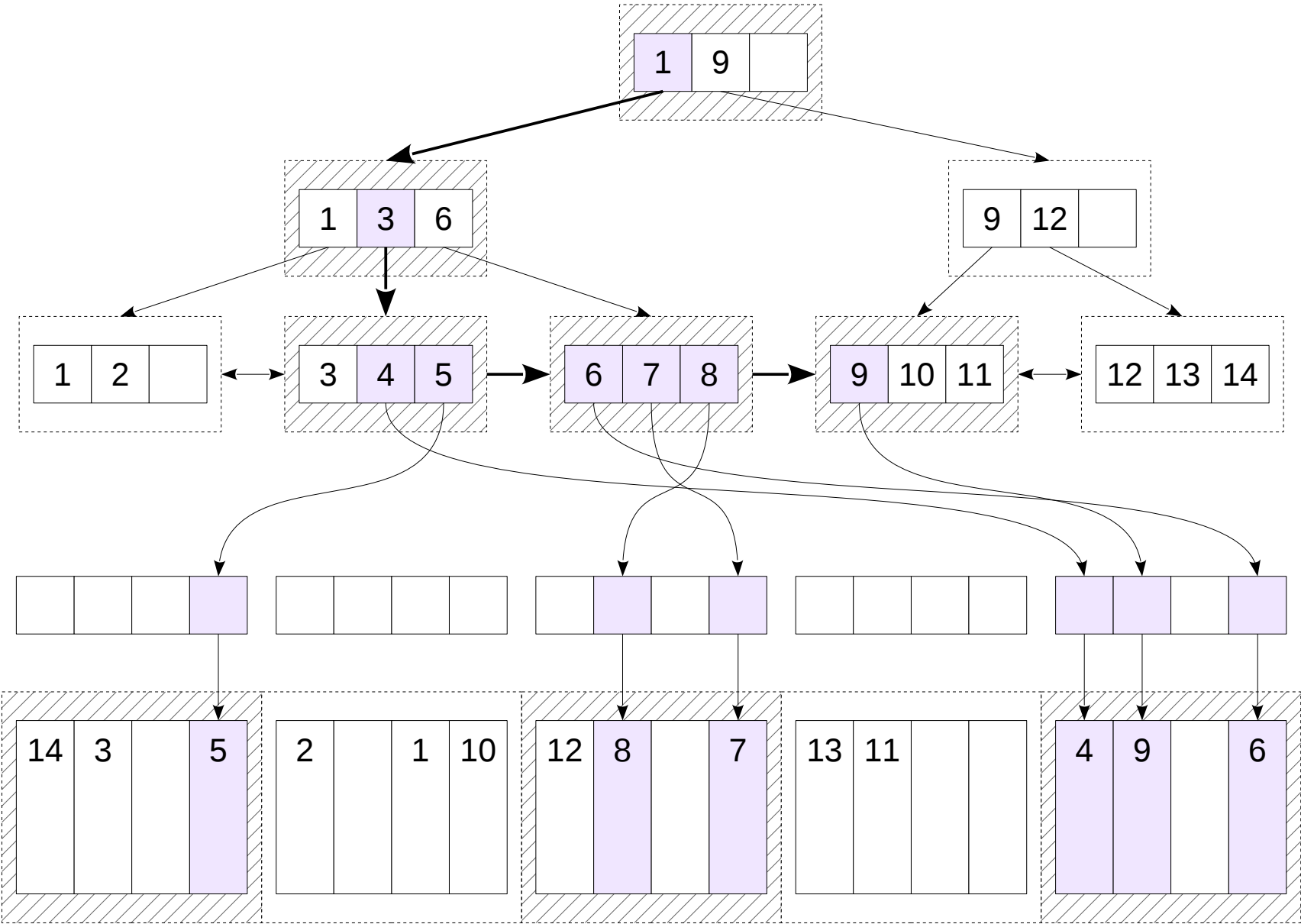
пройти все листовые страницы по ссылкам

для каждого ключа прочитать соответствующую страницу таблицы,  
проверить видимость версии строки таблицы и получить данные

## Проблема: хаотичное чтение страниц таблицы

одна и та же страница читается по несколько раз

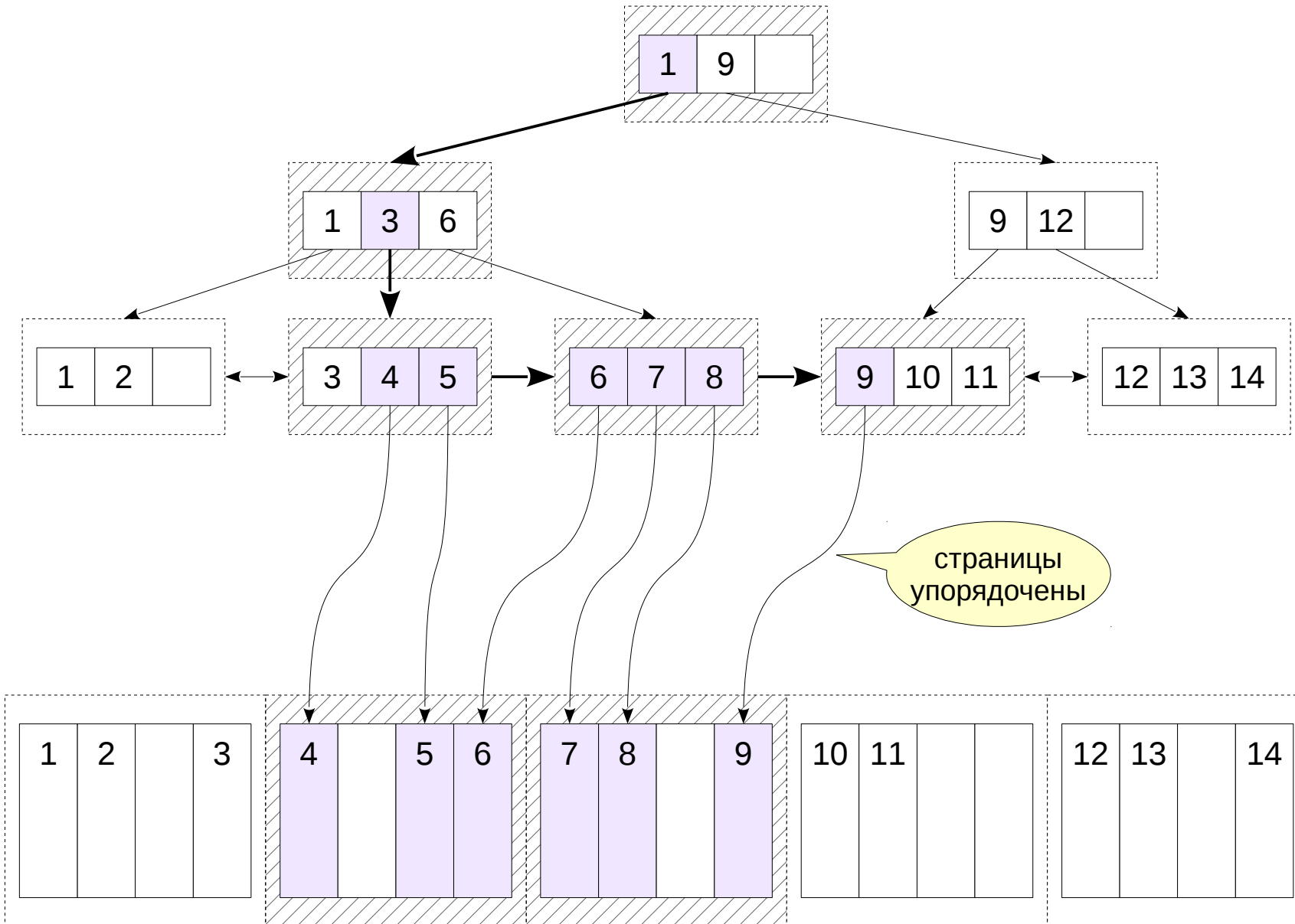
# Bitmap scan



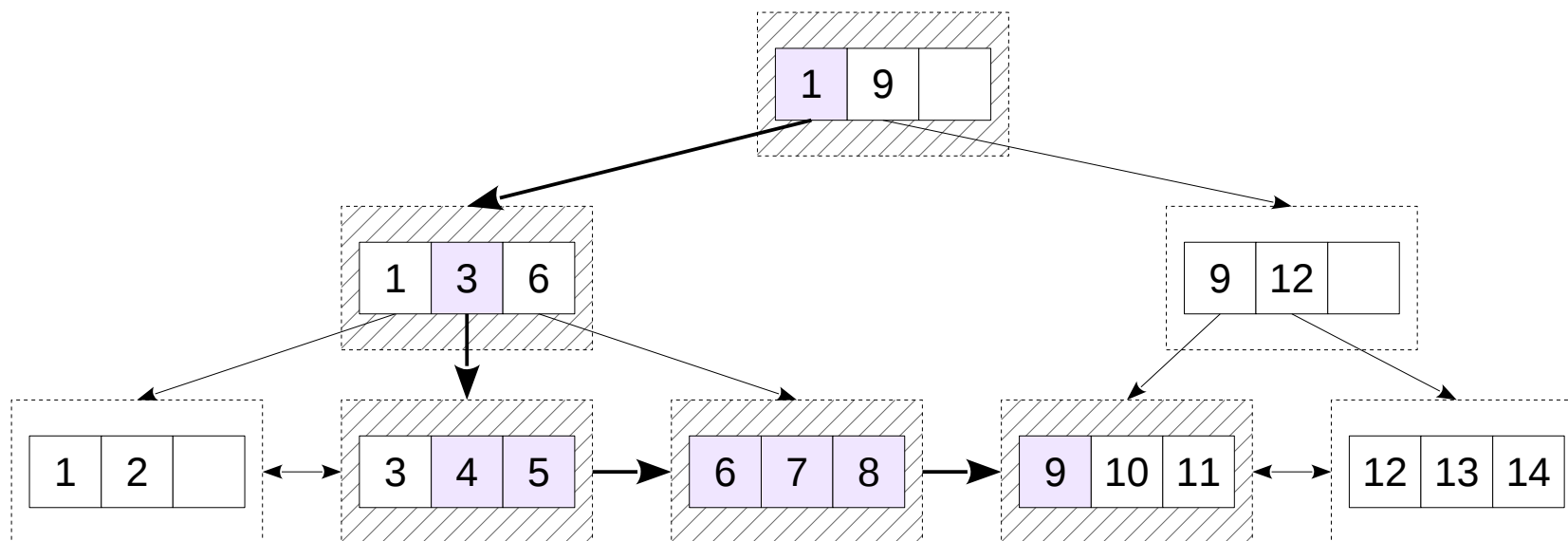
Эффективно для выборки среднего размера

- пройти от корня дерева до листовой страницы
- пройти необходимые листовые страницы по ссылкам
- построить в памяти битовую карту для всех найденных ключей
- последовательно прочитать страницы таблицы, отмеченные в битовой карте
- проверить видимость версий строк таблицы и получить данные

# Index scan или bitmap scan?



# Index only scan



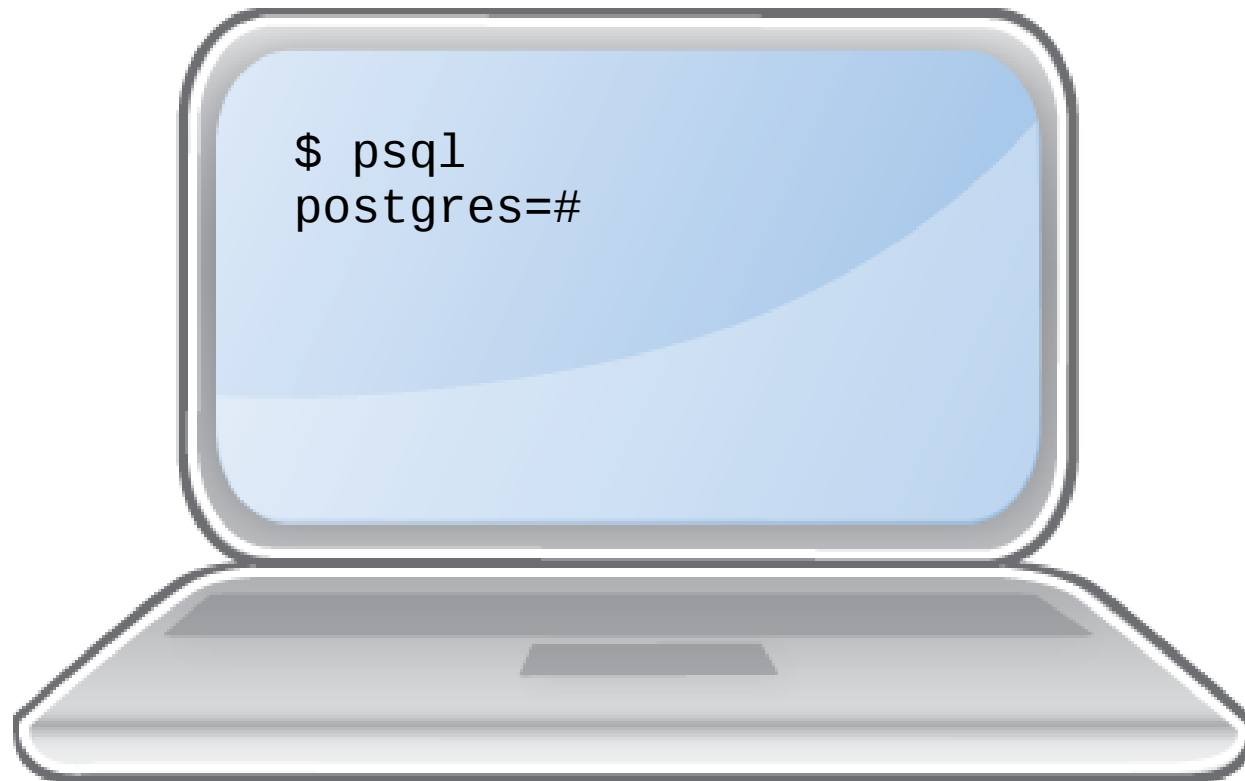
Если нужны только проиндексированные столбцы

не требуется обращение к таблице за данными

но проверка видимости по-прежнему нужна

помогает карта видимости

# Демонстрация



Получать данные можно разными методами

- индексное сканирование

- сканирование битовой карты

- последовательное сканирование

На эффективность методов влияет много факторов

- селективность условия

- необходимость обращаться к таблице и карта видимости

- соответствие порядка данных в таблице и в индексе

- нужен ли отсортированный результат

- результат нужен весь сразу или по мере получения

- и другие

1. Создайте базу DB11, в ней таблицу с индексом по одному из полей. Заполните таблицу большим объемом данных, выполните очистку, анализ и запретите сканирование битовой карты.
2. Напишите запрос, выбирающий 1% данных. Какой метод доступа из двух оставшихся выбран, сколько времени выполняется запрос?
3. Запретите выбранный метод доступа, снова выполните запрос и сравните время выполнения. Прав ли был оптимизатор?
4. Повторите пункты 2 и 3 для выборки 50% данных.
- 5\*. При создании индекса можно указать порядок сортировки столбца. Зачем, если индекс можно просматривать в любом направлении?



### **Авторские права**

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 год.

Авторы: Егор Рогов, Павел Лузанов

### **Использование материалов курса**

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

### **Обратная связь**

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

### **Отказ от ответственности**

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Последовательное сканирование

Индексное сканирование

Сканирование битовой карты

Исключительно индексное сканирование

Сортировка, ограничение, группировка

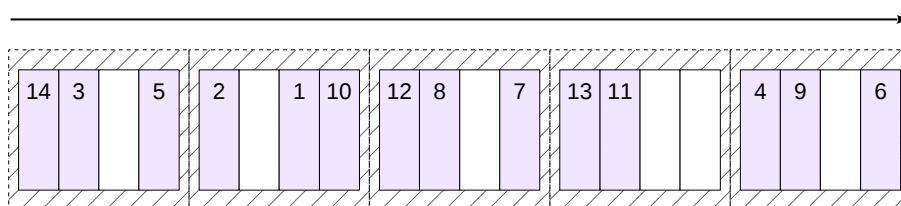
Команда explain

## Чтение всей таблицы

последовательно прочитать все страницы  
(в кэше используется буферное кольцо)  
проверить видимость версий строк и получить данные

Эффективно для получения всех данных

Не эффективно для небольшой выборки



3

В распоряжении оптимизатора имеется несколько способов доступа к данным. Самый простой из них — последовательное сканирование таблицы. Файл (или файлы) таблицы читается постранично от начала до конца. При этом рассматриваются все версии строк на каждой странице: удовлетворяют ли они условиям запроса и соблюдены ли правила видимости.

Напомним, что чтение происходит через буферный кэш; чтобы большая таблица не вытеснила все полезные данные, для последовательного сканирования создается «кольцо буферов» небольшого размера.

Последовательное чтение файла позволяет использовать тот факт, что операционная система обычно читает данные порциями больше, чем размер страницы: с большой вероятностью несколько следующих страниц уже окажутся в кэше ОС.

Последовательное сканирование эффективно работает, когда надо прочитать всю таблицу или значительную ее часть (если селективность условия низка). Если же из всей таблицы нужна только небольшая часть записей, более предпочтительными являются методы доступа, использующие индекс.

## Индекс

вспомогательная структура во внешней памяти  
сопоставляет ключи и идентификаторы строк таблицы

## Устройство: дерево поиска

сбалансированное  
сильно ветвистое

## Использование

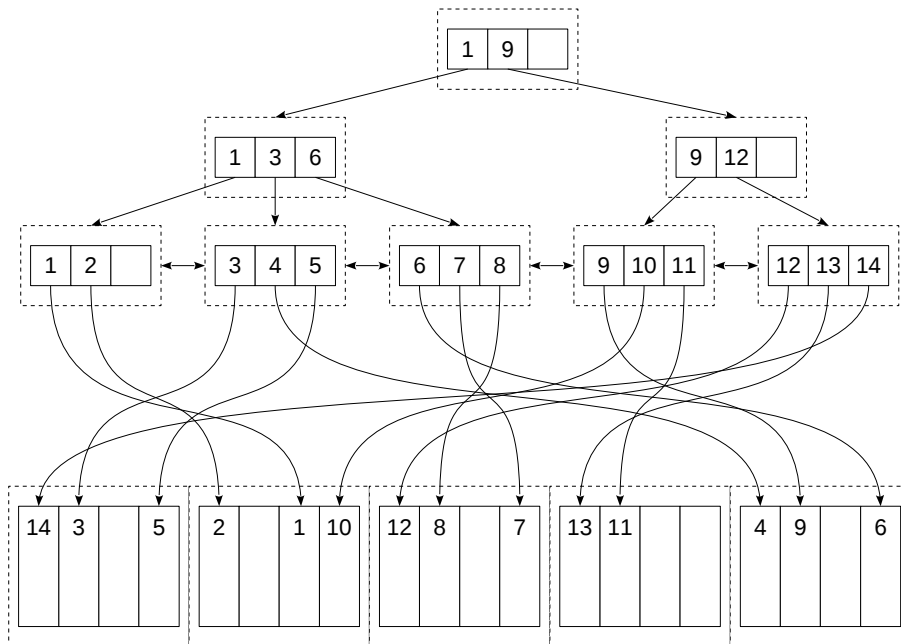
только сортируемые типы данных (операции «больше», «меньше»)  
ускорение доступа  
поддержка ограничений целостности

В этом курсе мы будем рассматривать только один из доступных в PostgreSQL типов индексов: В-дерево (B-tree). Это самый распространенный индекс, наиболее часто применяющийся на практике.

Как и все индексы в PostgreSQL, В-дерево является вторичной структурой — индекс не содержит в себе никакой информации, которую нельзя было бы получить из самой таблицы. Индекс можно удалить и пересоздать. Если бы не поддержка ограничений целостности (первичные и уникальные ключи), можно было бы сказать, что индексы влияют только на производительность, но не на логику работы.

Как всякий индекс, В-дерево сопоставляет значения проиндексированных полей (ключи поиска) и идентификаторы строк таблицы. Особенности В-дерева являются его сбалансированность (постоянная глубина) и сильная ветвистость. Хотя размер дерева зависит от проиндексированных столбцов, на практике деревья обычно не имеют глубину больше 4-5.

Кроме поддержки ограничений целостности, задача В-деревьев (как и всех индексов) состоит в том, чтобы ускорять доступ к данным. Есть и ограничение: проиндексировать можно только данные, допускающие сортировку (должны быть определены операции «больше», «меньше»). Например, числа, строки и даты могут быть проиндексированы, а точки на плоскости — нет (для них существуют другие типы индексов).



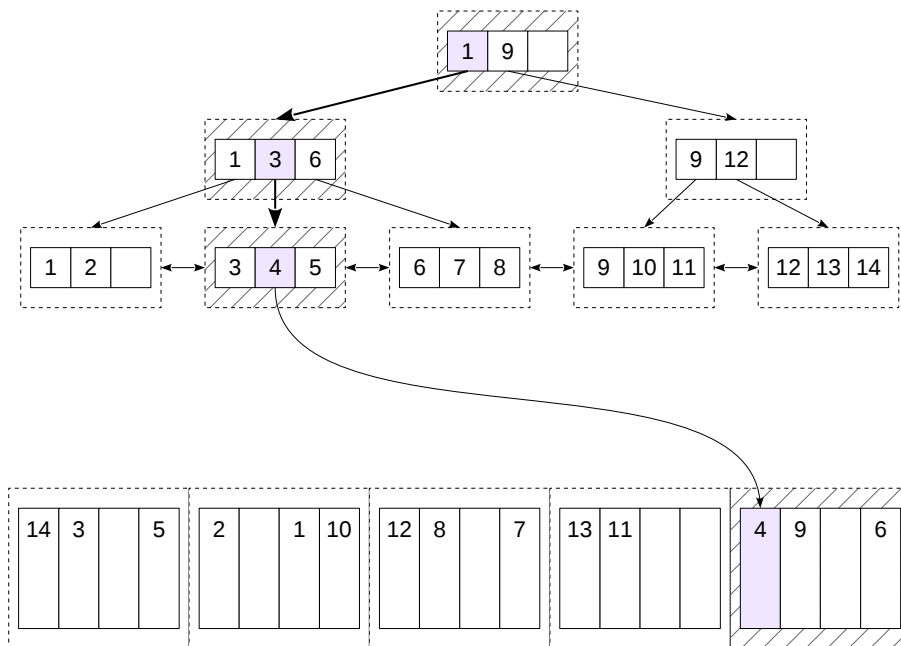
На слайде представлен пример В-дерева (верхняя часть рисунка). У дерева есть корневая, внутренние и листовые страницы. Страницы состоят из строк (или элементов), в которых содержатся значения столбцов, по которым построен индекс (ключи), и ссылки — либо на страницы более низкого уровня, либо на строки таблицы (в листовых страницах индекса).

Внутри страниц все ключи упорядочены.

Часть строк может быть не заполнена, как и в таблице. Эти места используются для вставки в индекс новых значений, а если на странице не хватает места — она разделяется на две новых страницы.

Разделившиеся страницы никогда не объединяются, и это в некоторых случаях может приводить к разрастанию индекса.

Листовые страницы дерева объединены двунаправленным списком. Это позволяет быстро переходить от одного значения к следующему за ним (или предыдущему).



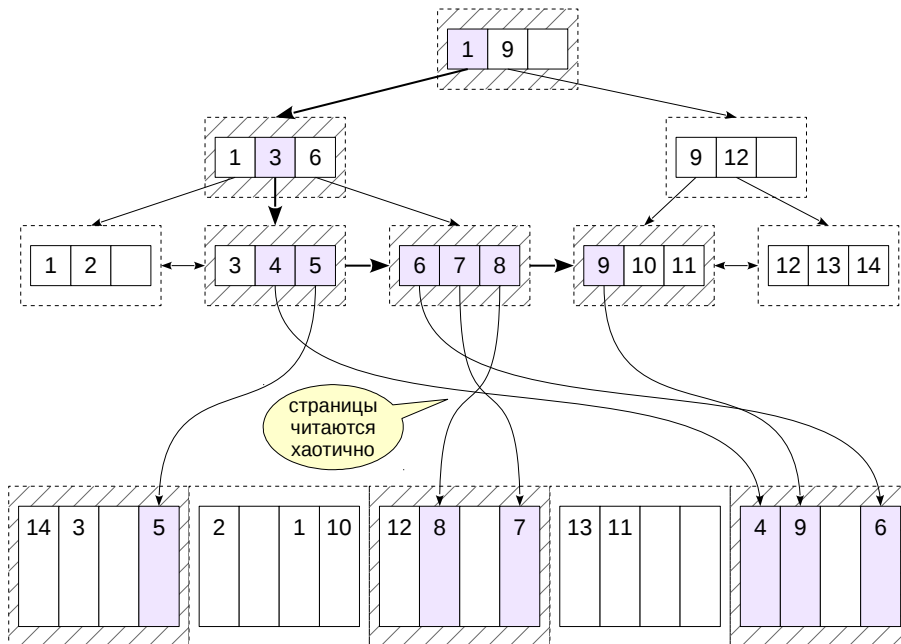
Рассмотрим поиск одного значения с помощью индекса. Например, нам нужно найти в таблице строку, где значение проиндексированного столбца равно четырем.

Начинаем с корня дерева. Строки в ней определяют диапазоны значений ключей в нижележащих страницах: «от 1 до 9» и «9 и больше». Нам подходит диапазон «от 1 до 9», что соответствует строке с ключом 1. Заметим, что ключи хранятся упорядоченно, следовательно, поиск по странице выполняется очень эффективно.

Ссылка из найденной строки приводит нас к странице второго уровня. В ней мы находим диапазон «от 3 до 5» (ключ 3) и переходим к странице третьего уровня.

Эта страница является листовой. В ней мы находим ключ, равный 4, и переходим на страницу таблицы.

В действительности процесс более сложен: мы не говорили про неуникальные ключи, про проблемы одновременного доступа и так далее. Но не будем углубляться в детали реализации.



B-дерево позволяет эффективно искать не только отдельные значения, но и диапазоны значений (по условиям «меньше/больше», «меньше/больше или равно», «between»).

Вот как это происходит. Сначала мы ищем крайний ключ условия. Например, для условия «от 4 до 9» мы можем выбрать значение 4 или 9, а для условия «меньше 9» надо взять 9. Затем спускаемся до листовой страницы индекса так, как мы рассматривали в предыдущем примере, и получаем первое значение из таблицы.

Дальше остается двигаться по листовым страницам вправо (или влево, в зависимости от условия), перебирая строки этих страниц до тех пор, пока мы не встретим ключ, выпадающий из диапазона. В нашем случае мы перебираем ключи 5, 6, и так далее до 9. Встретив ключ 10, прекращаем поиск.

Нам помогают два свойства: упорядоченность ключей на всех страницах и связанность листовых страниц двунаправленным списком.

Обратите внимание, что к одной и той же табличной странице нам пришлось обращаться несколько раз. Мы прочитали последнюю страницу (значение 4), затем первую (5), затем опять последнюю (6) и так далее.

## Получение одного значения — эффективно

пройти от корня дерева до листовой страницы,  
найти в ней подходящий ключ  
прочитать страницу таблицы,  
проверить видимость версии строки таблицы и получить данные

## Получение всех значений — не эффективно

пройти от корня дерева до листовой страницы  
пройти все листовые страницы по ссылкам  
для каждого ключа прочитать соответствующую страницу таблицы,  
проверить видимость версии строки таблицы и получить данные

## Проблема: хаотичное чтение страниц таблицы

одна и та же страница читается по нескольку раз

Сколько страниц нам пришлось прочитать для поиска одного значения?

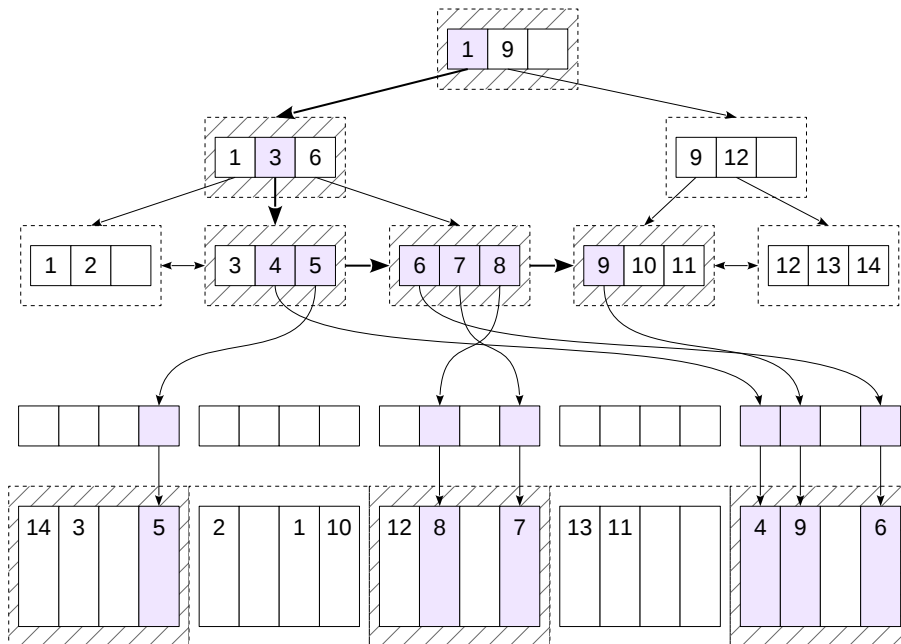
1. Спуститься от корня дерева к листовой странице;
2. Одна страница из таблицы.

Очевидно, что для такой выборки индекс гораздо эффективнее полного сканирования (если, конечно, таблица не состоит всего из нескольких страниц).

А если бы потребовалось прочитать все данные?

1. Спуститься от корня дерева к листовой странице;
2. Перебрать все листовые страницы индекса;
3. Перебрать все страницы таблицы (возможно, по нескольку раз).

Такой способ заведомо менее эффективен, чем последовательное сканирование. Хотя могут быть и исключения: ведь при сканировании индекса мы получаем уже отсортированные данные.



Чтобы не тратить ресурсы на просмотр одних и тех же табличных страниц по несколько раз (ведь это требует поиска страницы в буферном кэше и выставления блокировки), есть еще один способ доступа — сканирование битовой карты. Он похож на индексный доступ, но вместо того, чтобы сразу же обращаться к табличным страницам, в памяти строится битовая карта. В этой карте отмечаются те строки, которые должны быть прочитаны.

Когда индекс просканирован и битовая карта готова, мы читаем табличные страницы, но:

- делаем это в порядке последовательного доступа (увеличивается шанс воспользоваться кэшем ОС);
- каждую страницу просматриваем ровно один раз.

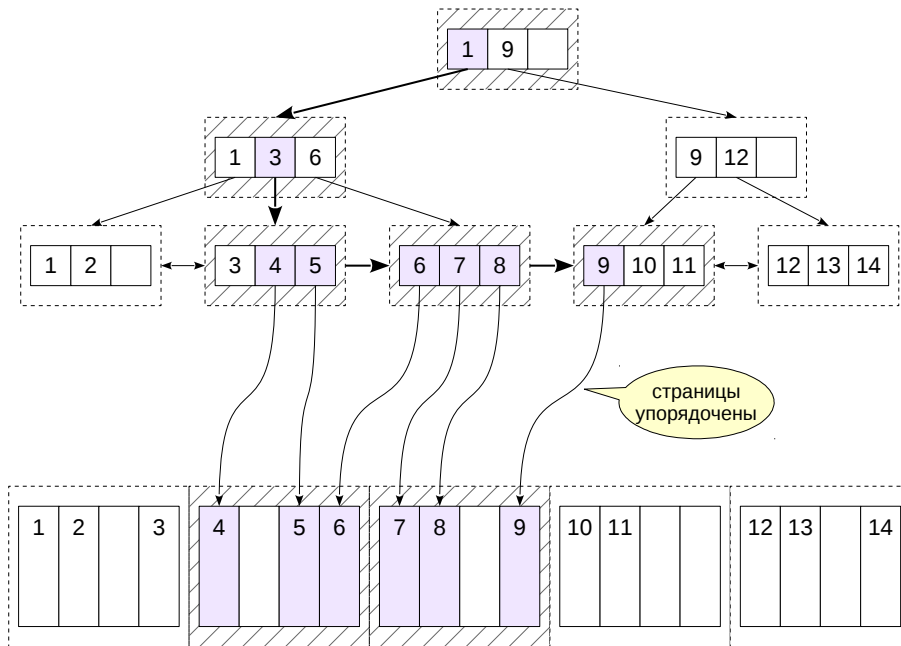
## Эффективно для выборки среднего размера

- пройти от корня дерева до листовой страницы
- пройти необходимые листовые страницы по ссылкам
- построить в памяти битовую карту для всех найденных ключей
- последовательно прочитать страницы таблицы,  
отмеченные в битовой карте
- проверить видимость версий строк таблицы и получить данные

Итак. сканирование битовой карты экономит на многократных чтениях табличных страниц, но проигрывает за счет необходимости строить карту.

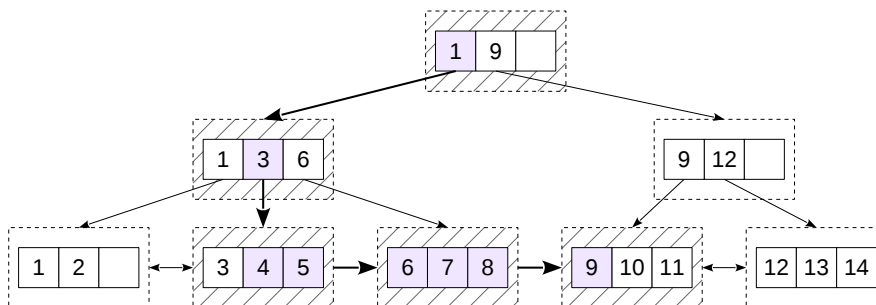
Поэтому обычно оно более эффективно, чем сканирование индекса, на выборках средних размеров: для одного значения строить битовую карту бессмысленно, а всю таблицу в любом случае проще прочитать без использования индекса.

# Index scan или bitmap scan?



11

Если данные в таблице физически упорядочены, обычное индексное сканирование не будет возвращаться к одной и той же табличной странице повторно и без всякой карты. В таком (нечастом на практике) случае метод сканирования битовой карты теряет свой смысл.



## Если нужны только проиндексированные столбцы

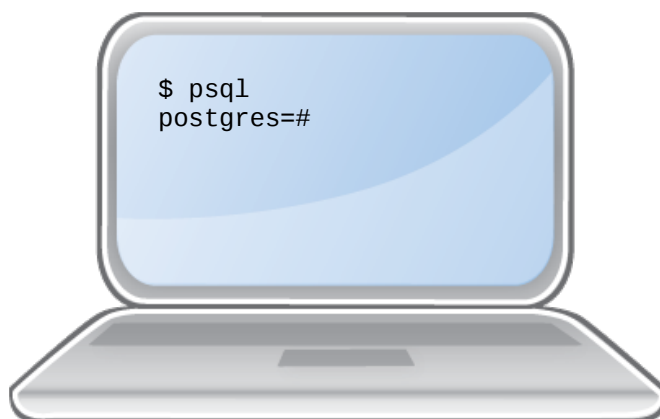
не требуется обращение к таблице за данными  
но проверка видимости по-прежнему нужна  
помогает карта видимости

12

Наконец, если в запросе требуются только проиндексированные данные, то они уже есть в самом индексе — к таблице в этом случае обращаться не надо.

Это было бы прекрасно, если бы не проверка видимости — ведь эта информация не содержится в индексе. Поэтому критическую роль в эффективности исключительно индексного сканирования играет карта видимости. Если большинство страниц таблицы содержат гарантированно видимые данные, и это отражено в карте видимости, то к этим табличным страницам действительно не надо обращаться. Но страницы, не отмеченные в карте видимости, посетить все-таки придется.

Планировщик не знает, сколько страниц таблицы потребуют проверки, но учитывает приблизительную оценку этого числа. Карта видимости непосредственно просматривается уже при выполнении запроса, и для каждой строки принимается решение, надо ли заглядывать в таблицу или нет.



## Получать данные можно разными методами

- индексное сканирование
- сканирование битовой карты
- последовательное сканирование

## На эффективность методов влияет много факторов

- селективность условия
- необходимость обращаться к таблице и карта видимости
- соответствие порядка данных в таблице и в индексе
- нужен ли отсортированный результат
- результат нужен весь сразу или по мере получения
- и другие

1. Создайте базу DB11, в ней таблицу с индексом по одному из полей. Заполните таблицу большим объемом данных, выполните очистку, анализ и запретите сканирование битовой карты.
2. Напишите запрос, выбирающий 1% данных. Какой метод доступа из двух оставшихся выбран, сколько времени выполняется запрос?
3. Запретите выбранный метод доступа, снова выполните запрос и сравните время выполнения. Прав ли был оптимизатор?
4. Повторите пункты 2 и 3 для выборки 50% данных.
- 5\*. При создании индекса можно указать порядок сортировки столбца. Зачем, если индекс можно просматривать в любом направлении?