



Использование памяти



Использование оперативной памяти разными операциями

Ограничение размера выделяемой памяти

Временные файлы

Для чего нужна память?

Сортировка

- ORDER BY, подготовка к соединению слиянием
- группировка
- создание индексов

Хэширование

- соединение хэшированием
- группировка

Временное хранение наборов строк

- результаты соединений
- материализация CTE
- оконные функции

Сканирование битовой карты

Оперативная память

память каждой операции ограничена параметром *work_mem*

при выполнении запроса несколько операций могут использовать память одновременно

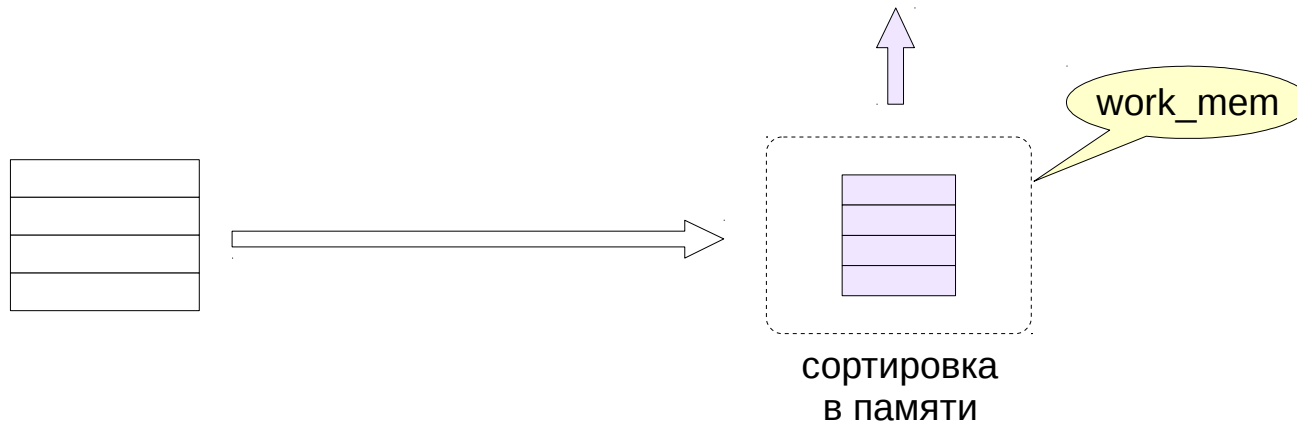
если выделенной памяти не хватает, используется диск (иногда операция может и превысить ограничение)

больше памяти — быстрее выполнение

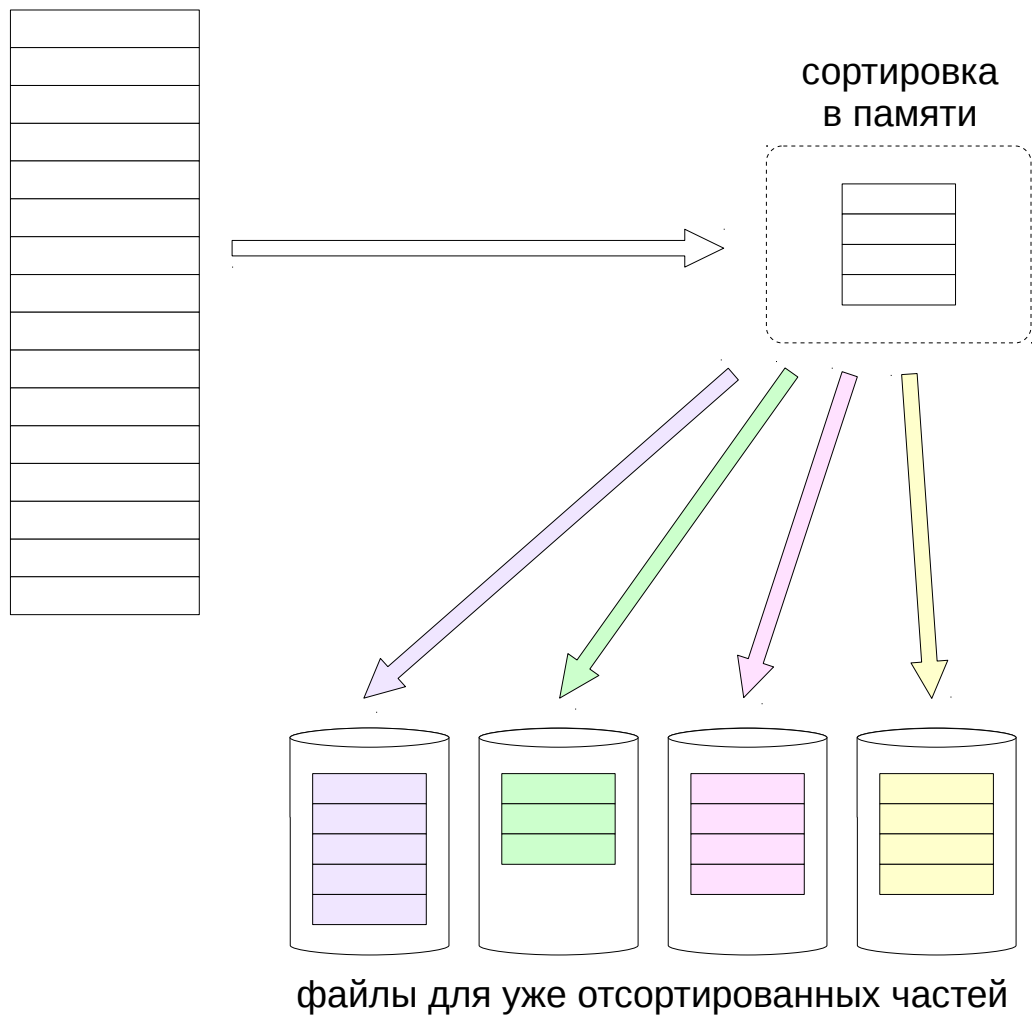
Дисковая память

общая дисковая память сеанса ограничена параметром *temp_file_limit* (без учета временных таблиц)

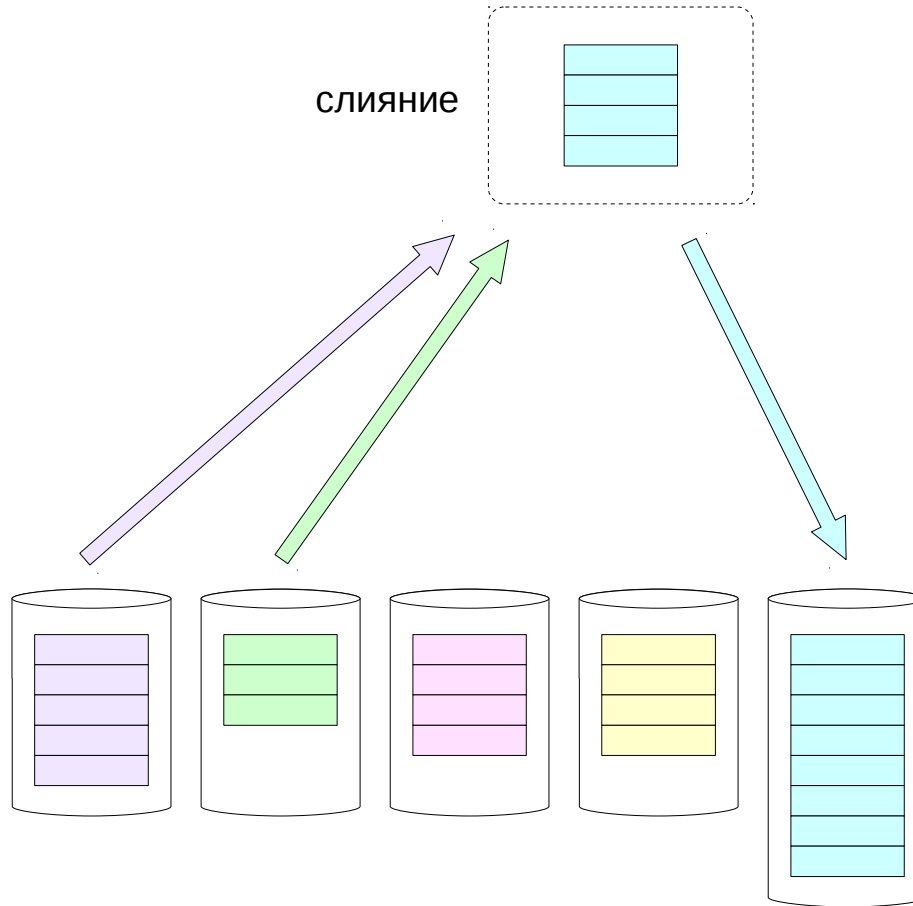
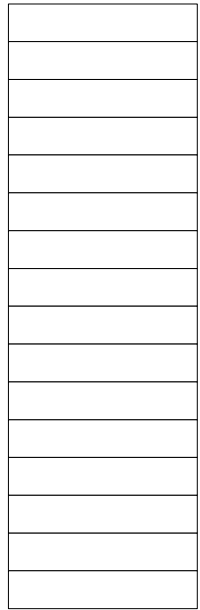
Сортировка – случай 1



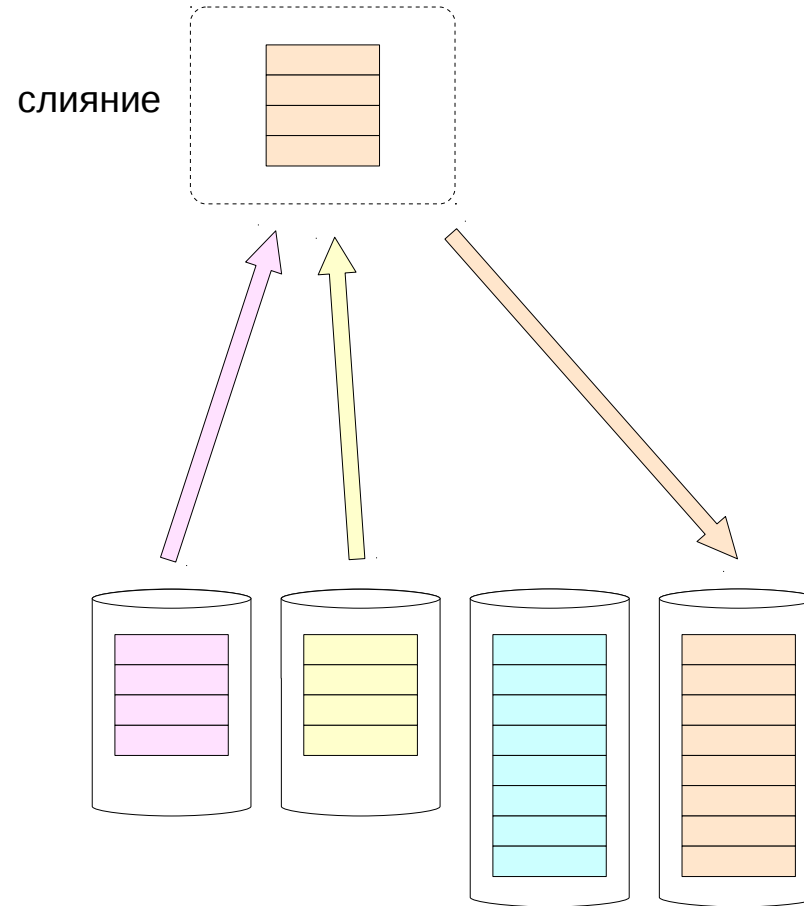
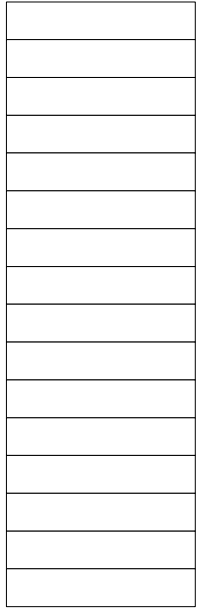
Сортировка – случай 2



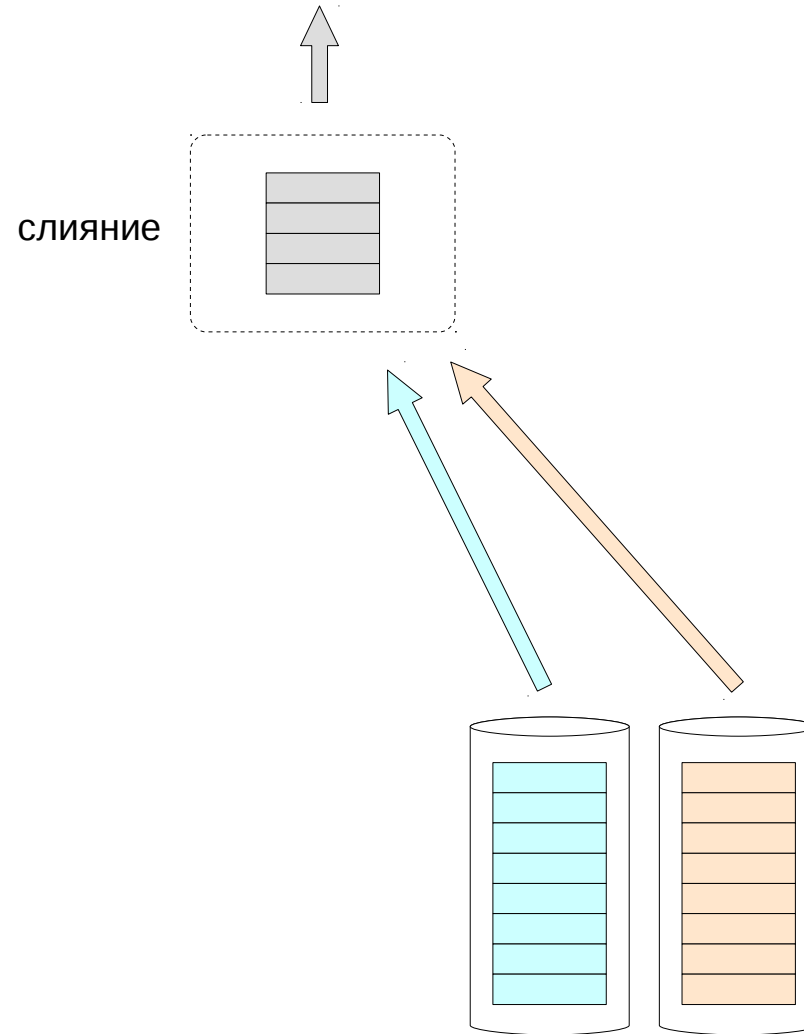
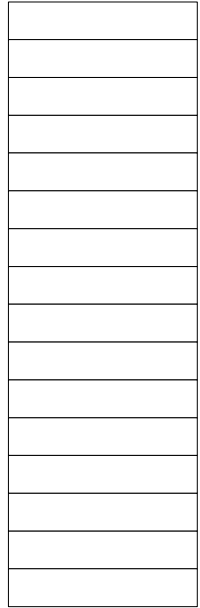
Сортировка – случай 2



Сортировка – случай 2



Сортировка – случай 2



Создание индекса b-tree

Используется сортировка

сначала все строки сортируются

затем строки собираются в листовые индексные страницы

ссылки на них собираются в страницы следующего уровня

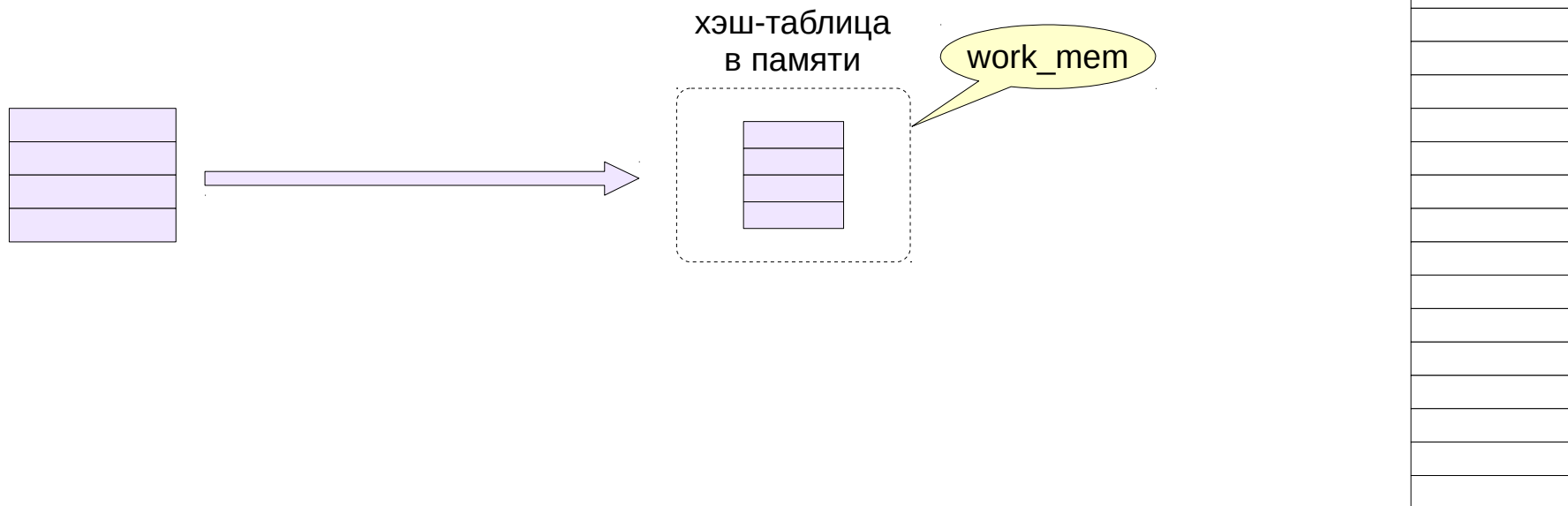
и так далее, пока не дойдем до корня

Ограничение

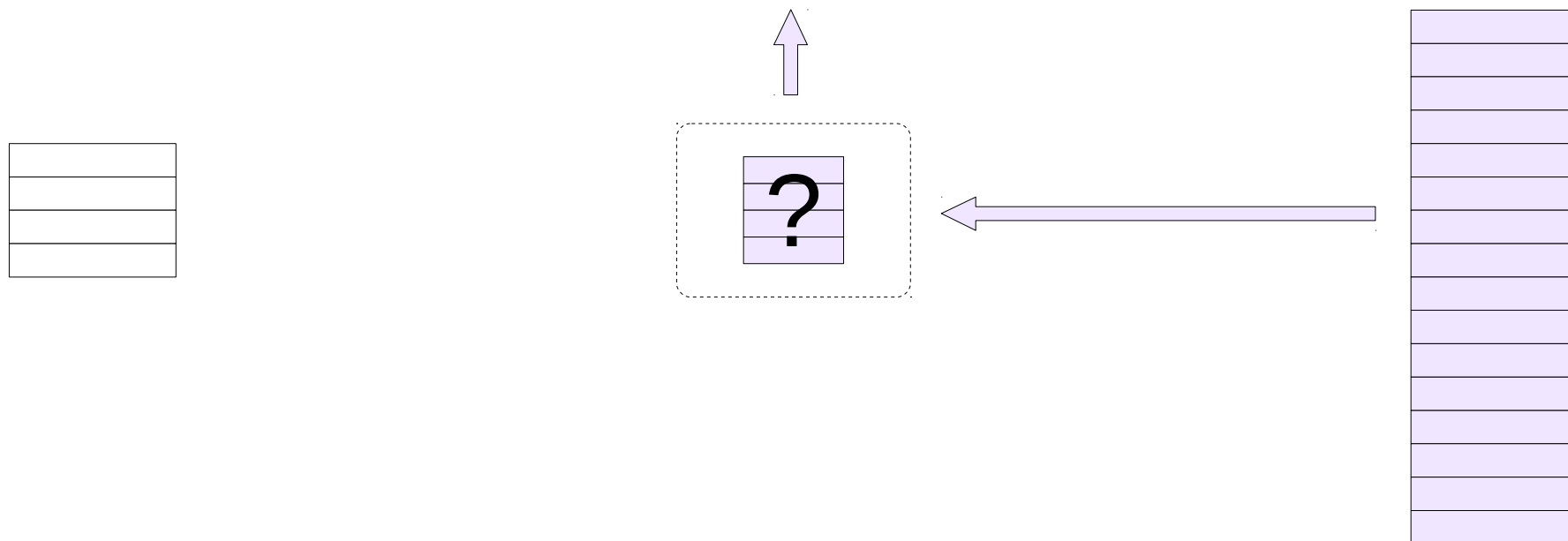
maintenance_work_mem, так как операция не частая

для уникальных индексов требуется дополнительно *work_mem*

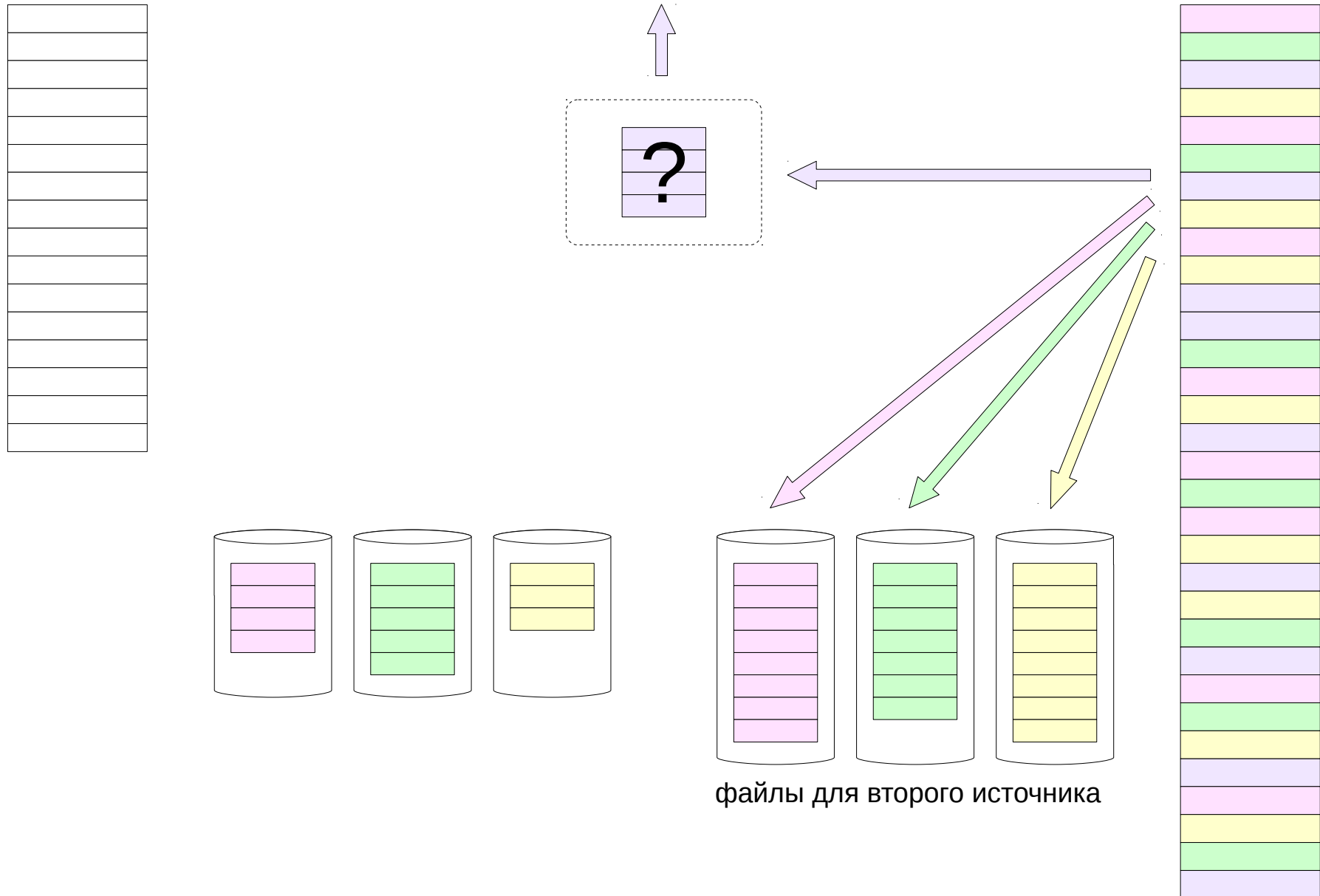
Hash Join – случай 1



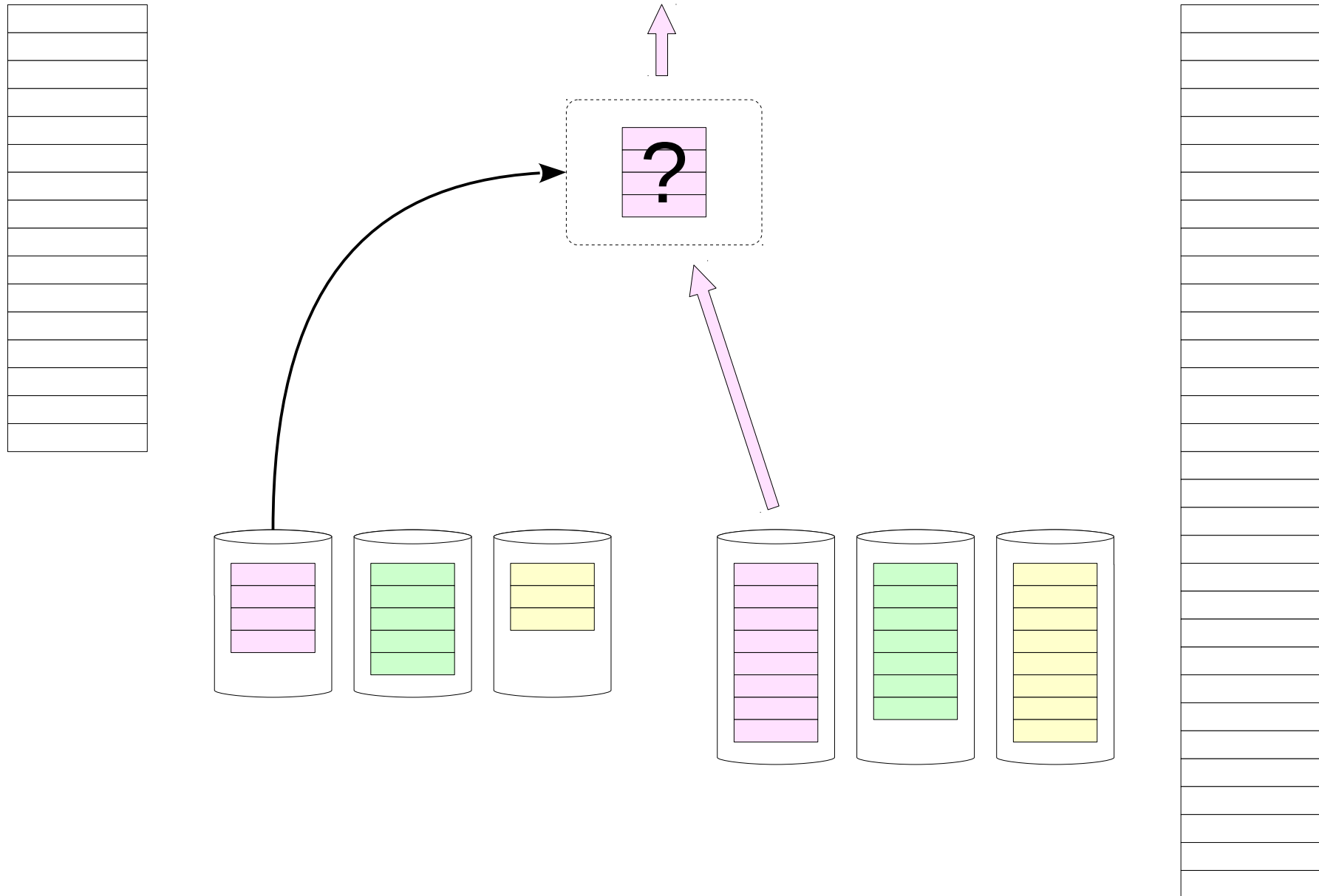
Hash Join – случай 1



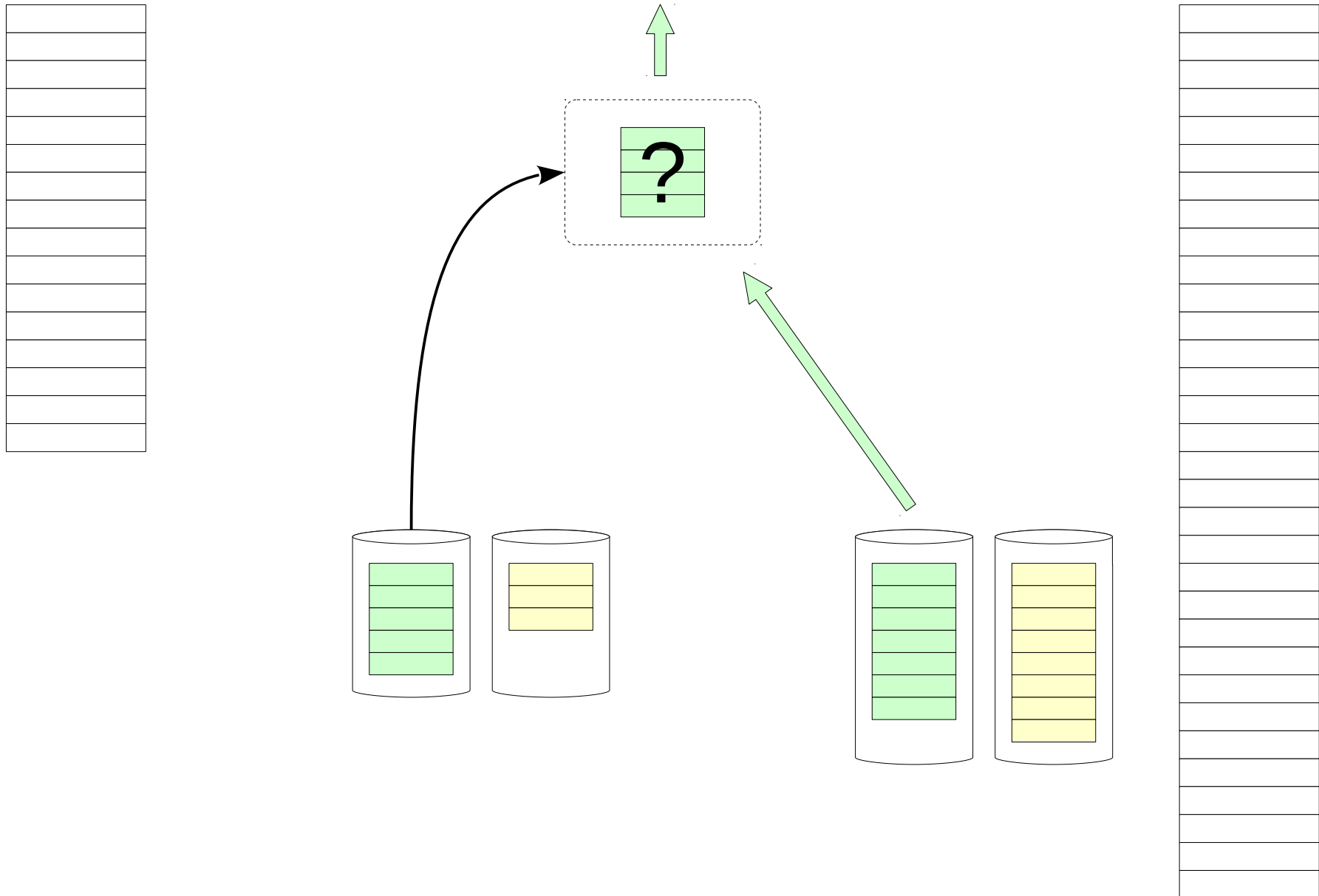
Hash Join – случай 2



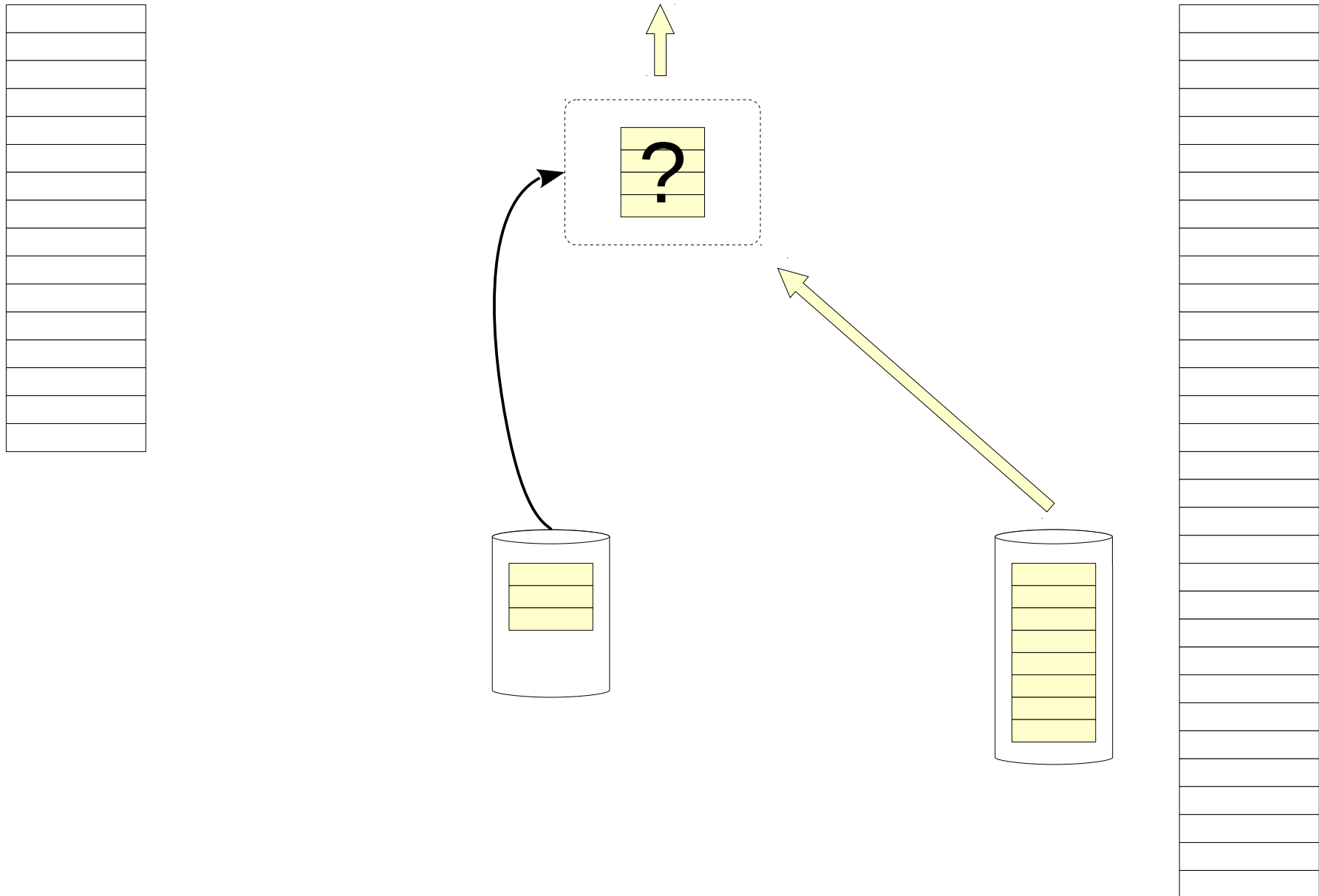
Hash Join – случай 2



Hash Join – случай 2



Hash Join – случай 2



Без потери точности

пока есть возможность, информация хранится с точностью до строки

С потерей точности

если память закончилась, происходит огрубление части уже построенной карты до отдельных страниц

при чтении табличной страницы требуется перепроверка условий

требуется примерно 1 МБ памяти на 64 ГБ данных;

ограничение памяти может быть превышено

Память серверного процесса

выполнение запросов

временные таблицы

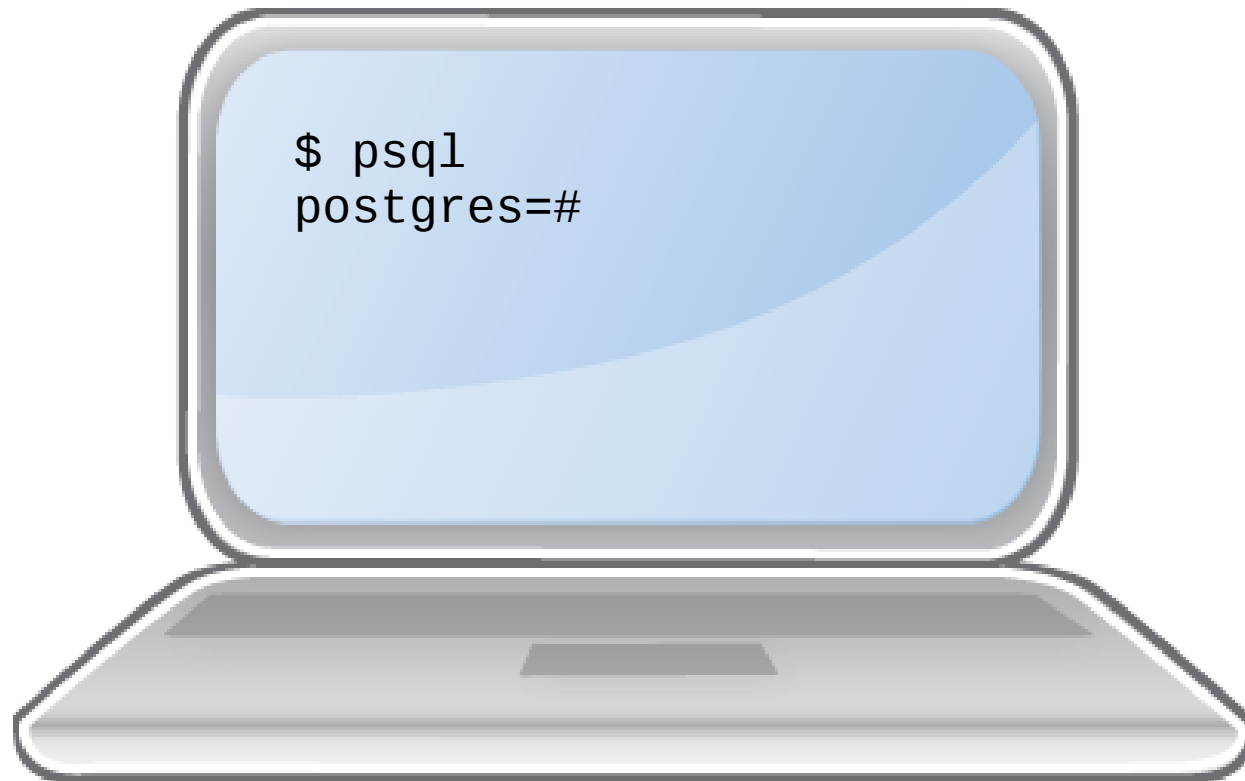
кэш системного каталога

кэш подготовленных операторов

$N \cdot work_mem$

$\leq temp_buffers$

Демонстрация

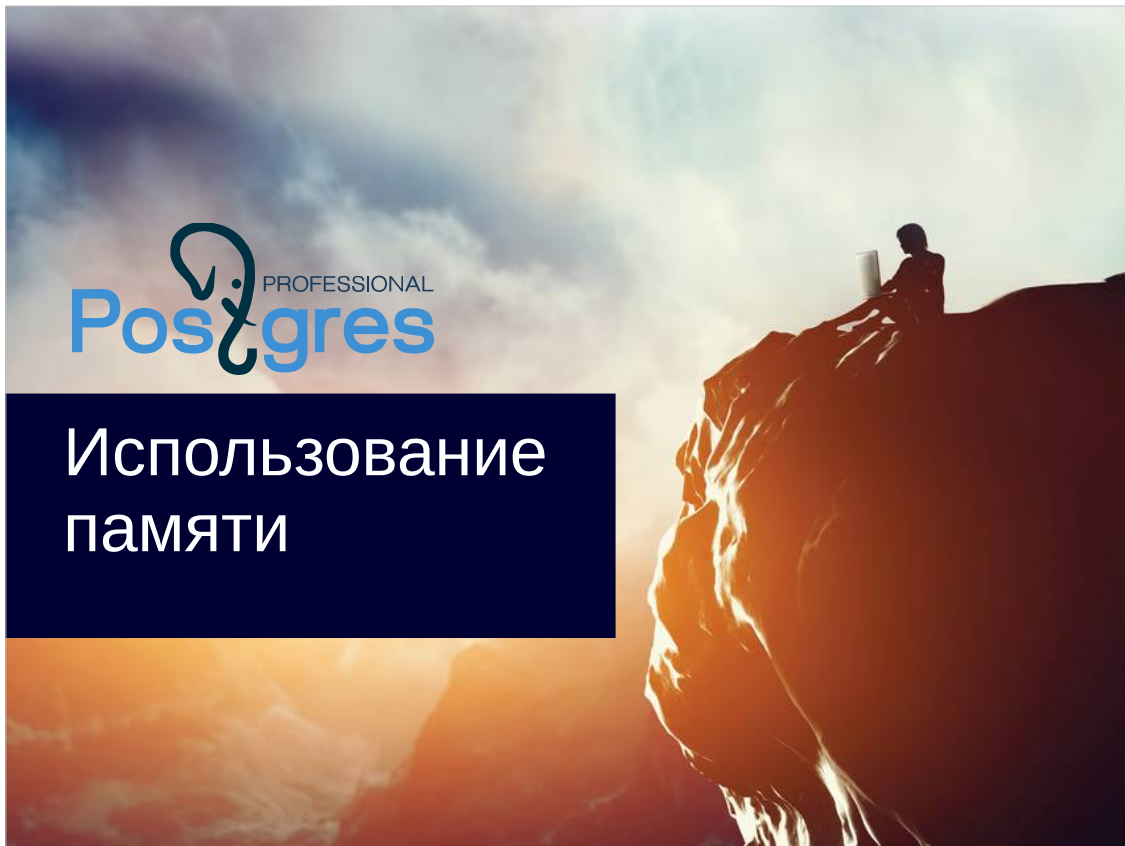


Ряд операций требуют оперативной памяти

Применяются алгоритмы, использующие доступную память и временные файлы на диске

Чем больше памяти, тем быстрее выполнение

1. Создайте базу DB18 и в ней повторите ситуацию с хэш-соединением из практики, однако вместо меньшей таблицы используйте СТЕ.
Как изменились расход памяти и время выполнения?
2. С помощью команды `explain analyze` исследуйте влияние доступной памяти на сканирование битовой карты.
3. Как выполняется сортировка, если в запросе указана фраза `limit`?
4. Включите вывод временных файлов в журнал сообщений сервера и проверьте, используется ли дисковая память:
 - а) при использовании функции `generate_series`,
 - б) при использовании СТЕ,
 - в) для расчета значения какой-либо оконной функции.
5. Исследуйте, как параметр `maintenance_work_mem` влияет на скорость создания индекса.



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

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.

Использование оперативной памяти разными операциями

Ограничение размера выделяемой памяти

Временные файлы

Сортировка

- ORDER BY, подготовка к соединению слиянием
- группировка
- создание индексов

Хэширование

- соединение хэшированием
- группировка

Временное хранение наборов строк

- результаты соединений
- материализация CTE
- оконные функции

Сканирование битовой карты

В процессе выполнения запросов серверный процесс требует определенного размера оперативной памяти.

Некоторые операции могут быть проделаны с минимальным использованием памяти, не зависящим от объема обрабатываемых данных. Например, при выполнении соединения вложенными циклами не имеет значения, насколько велики наборы объединяемых строк: память требуется только для одной строки из первого набора и одной строки из второго (разумеется, говоря очень упрощенно).

Но для других операций хорошо иметь память, пропорциональную размеру выборки. Например, чтобы отсортировать строки таблицы, удобно было бы загрузить их все в память. Необходимость в сортировке возникает как при явном указании отсортировать результат (конструкция `order by`), так и при группировке данных, создании индексов `b-tree`.

Другой часто встречающейся операцией, требующей большого объема памяти, является хэширование. Оно применяется при соединении хэшированием, а также может использоваться при группировке вместо сортировки.

Также оперативная память требуется, чтобы сохранять промежуточные наборы строк: результаты соединений, материализованные подзапросы, наборы строк для оконных функций и т. п.

Еще один пример операции, требующей памяти — операция сканирования битовой карты, так как битовая карта строится в оперативной памяти.

Оперативная память

память каждой операции ограничена параметром *work_mem*
при выполнении запроса несколько операций могут
использовать память одновременно
если выделенной памяти не хватает, используется диск
(иногда операция может и превысить ограничение)
больше памяти — быстрее выполнение

Дисковая память

общая дисковая память сеанса ограничена параметром *temp_file_limit*
(без учета временных таблиц)

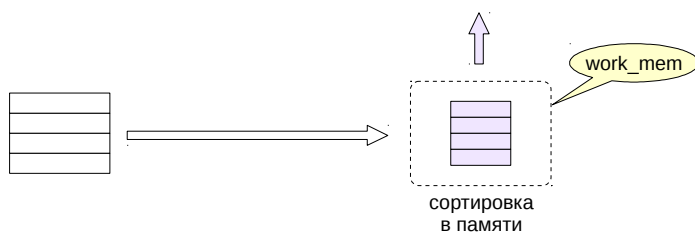
Конечно, в наличии обычно не бывает такого количества оперативной памяти, чтобы в нее поместилась вся необходимая информация. Поэтому применяются более хитрые алгоритмы, использующие имеющуюся память и обращающиеся по мере необходимости к диску. Как правило, чем больше памяти выделено, тем быстрее будет идти обработка.

Ограничение задается для каждой отдельной операции с помощью параметра *work_mem*. Следует учитывать, что при выполнении запроса может выполняться несколько операций, каждая из которых будет использовать память. Кроме того, само ограничение не является жестким: некоторые операции в случае безысходности могут выходить за ограничение. Поэтому сложно предугадать реальный размер памяти, который может быть задействован сеансом.

Использование временных файлов на диске также можно ограничить. Здесь параметром *temp_file_limit* определяется общее ограничение дисковой памяти для сеанса. Временные таблицы в это ограничение не входят.

<http://www.postgresql.org/docs/9.5/static/runtime-config-resource.html>

Сортировка – случай 1

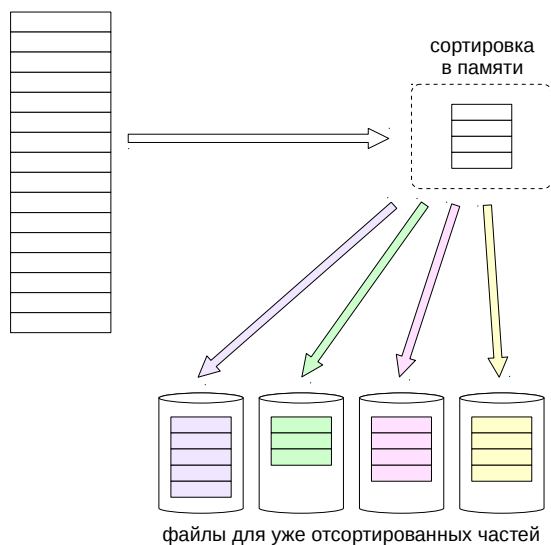


Рассмотрим несколько примеров алгоритмов, использующих доступную память и временные файлы, и начнем с сортировки.

В идеальном случае набор строк, подлежащий сортировке, целиком помещается в память, ограниченную параметром *work_mem*. В этом случае считываются все строки, сортируются (используется алгоритм быстрой сортировки *qsort*) и возвращается результат.

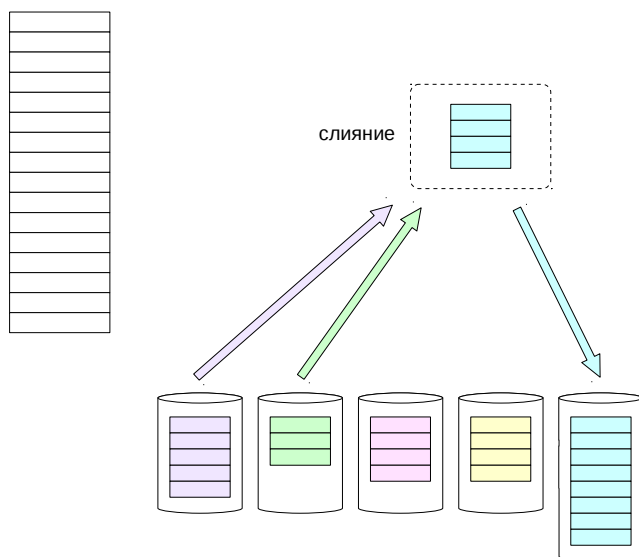
На рисунке цветом показаны отсортированные наборы данных.

Сортировка – случай 2



Если набор строк велик, он не поместится в память целиком. В таком случае используется алгоритм сортировки слиянием.

Набор строк читается в память, пока есть возможность, а затем по мере дальнейшего чтения значения вытесняются во временные файлы так, чтобы в каждом файле формировались упорядоченные данные. В итоге на диске получается некоторое количество файлов, каждый из которых по отдельности уже отсортирован.

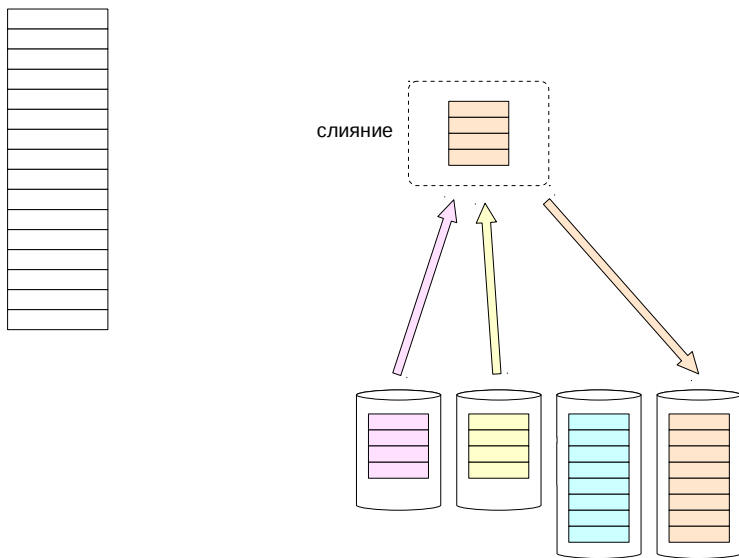


Далее несколько файлов сливаются в один по аналогии с тем, как работает соединение слиянием. Отличие в том, что сливаться могут более двух файлов одновременно.

Для слияния не требуется много места в памяти. Достаточно разместить по одной строке из каждого файла; но обычно строки читаются порциями, чтобы оптимизировать ввод-вывод.

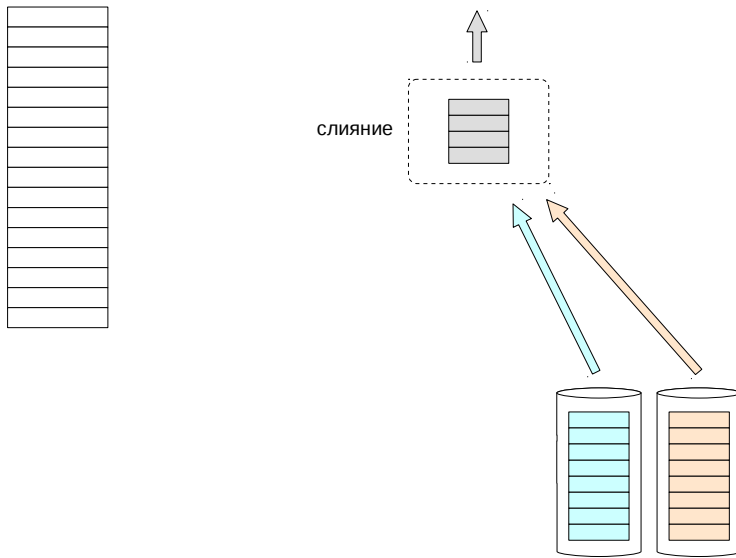
В примере мы сначала объединяем первые два файла, записывая результат в новый временный файл (так показано для простоты, а на практике переиспользуются существующие файлы, чтобы не расходовать зря место на диске).

Сортировка – случай 2



...затем мы объединяем вторые два файла...

Сортировка – случай 2



...и наконец объединяем два оставшихся файла, возвращая результат сортировки.

Для любопытствующих: история развития способов сортировки в PostgreSQL описана в презентации Грегори Старка «[Sorting Through The Ages](#)».

Используется сортировка

сначала все строки сортируются
затем строки собираются в листовые индексные страницы
ссылки на них собираются в страницы следующего уровня
и так далее, пока не дойдем до корня

Ограничение

maintenance_work_mem, так как операция не частая
для уникальных индексов требуется дополнительно *work_mem*

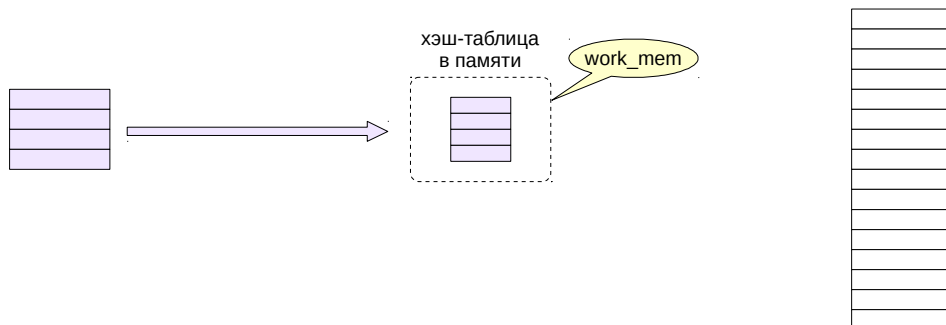
Индекс (речь идет про b-tree) можно строить, добавляя последовательно в пустой индекс по одной строке из таблицы. Но такой способ крайне неэффективен для неупорядоченных данных.

Поэтому для создания индекса используется сортировка: все строки таблицы сортируются и раскладываются по листовым индексным страницам. Затем достраиваются верхние уровни дерева, состоящие из ссылок на элементы страниц нижележащего уровня, до тех пор, пока на очередном уровне не получится одна страница — она и будет корнем дерева.

Сортировка устроена точно так же, как рассматривалось выше. Однако размер памяти ограничен не *work_mem*, а *maintenance_work_mem*, поскольку операция создания индекса не слишком частая и имеет смысл выделить для нее больше памяти.

Для создания уникальных индексов требуется дополнительная память размером *work_mem*. Она нужна для хранения неактуальных версий строк, чтобы исключить их из проверки на уникальность.

Hash Join – случай 1

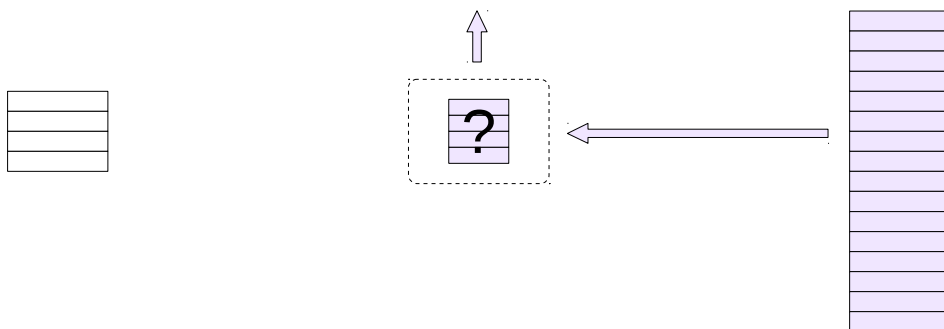


Еще одна операция, использующая доступную память и временные файлы — хэширование, которое мы рассмотрим на примере соединения хэшированием.

Напомним (тема «Способы соединения»), что соединение хэшированием начинается с того, что считывается первый набор строк и в памяти строится хэш-таблица, ключами которой являются значения столбцов, по которым происходит соединение.

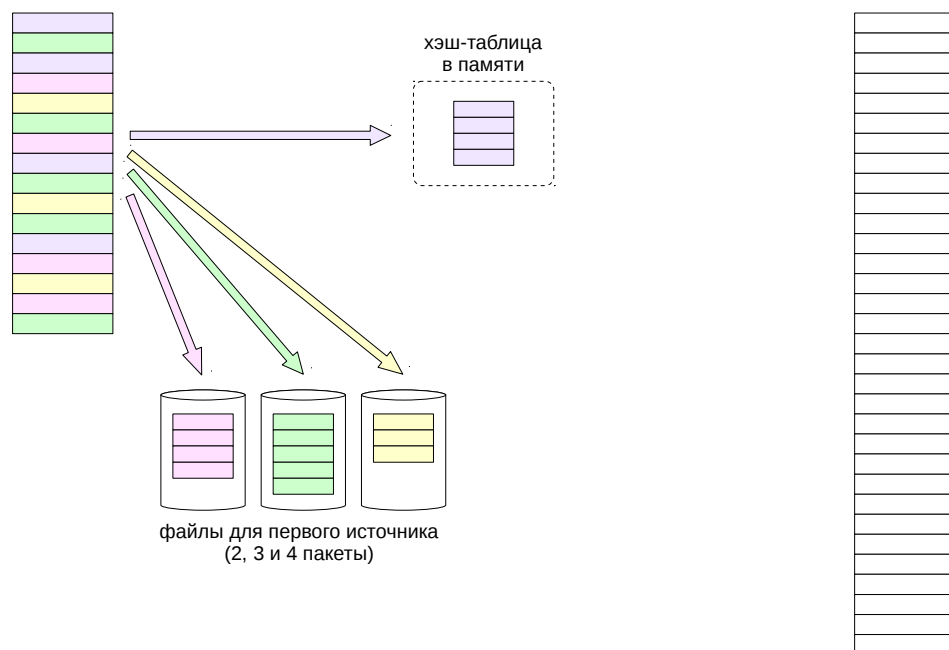
В идеальном случае хэш-таблица целиком помещается в память (ограниченную параметром *work_mem*). Именно поэтому важно, чтобы первый набор строк был меньше второго: от правильности оценки кардинальности зависит эффективность соединения.

Hash Join – случай 1



Дальше считывается второй набор строк, и по мере чтения происходит вычисление хэш-кодов строк и сопоставление их с хэш-таблицей. Строки с совпадающими хэш-кодами возвращаются как результат соединения (после проверки дополнительных условий фильтрации).

Hash Join – случай 2



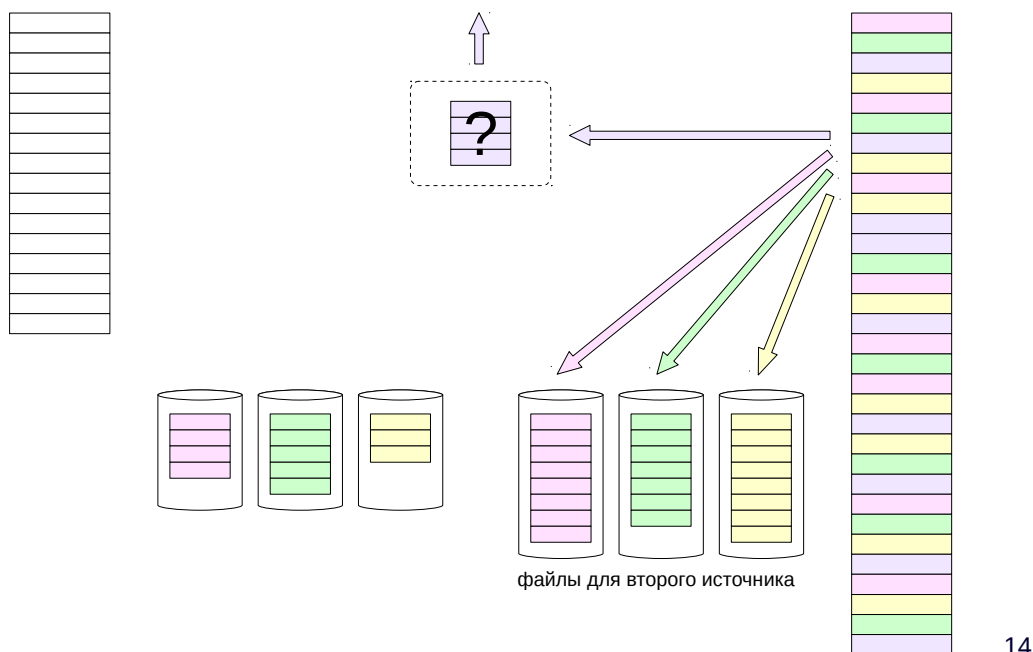
Однако хэш-таблица может не поместиться в *work_mem*. В таком случае первый набор строк разбивается на отдельные пакеты — так, чтобы в каждый пакет попало примерно одинаковое число строк. На рисунке пакеты выделены разными цветами.

При планировании запроса заранее вычисляется минимально необходимое число пакетов так, чтобы хэш-таблица для каждого пакета помещалась в памяти. Это число не уменьшается, даже если оптимизатор ошибся с оценками, но при необходимости может динамически увеличиваться.

В принципе, может так не повезти, что увеличение числа пакетов не приведет к уменьшению хэш-таблицы. В этом случае хэш-таблица будет занимать больше памяти, чем *work_mem* (в надежде на то, что общей памяти сервера хватит).

Так или иначе, но хэш-таблица первого пакета остается в памяти, а строки, принадлежащие другим пакетам, сбрасываются на диск во временные файлы — каждый пакет в свой файл.

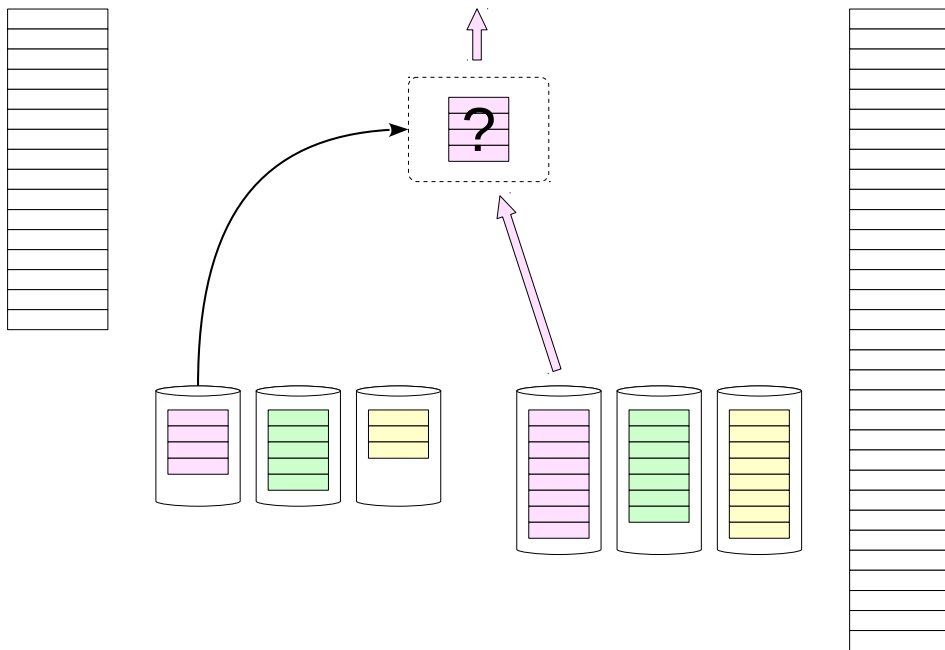
Hash Join – случай 2



Далее мы читаем второй набор строк. Если строка принадлежит первому пакету, сопоставляем ее с хэш-таблицей так же, как и в простом варианте. Если же строка принадлежит другому пакету, сбрасываем ее на диск во временный файл — опять же, каждый пакет в свой файл.

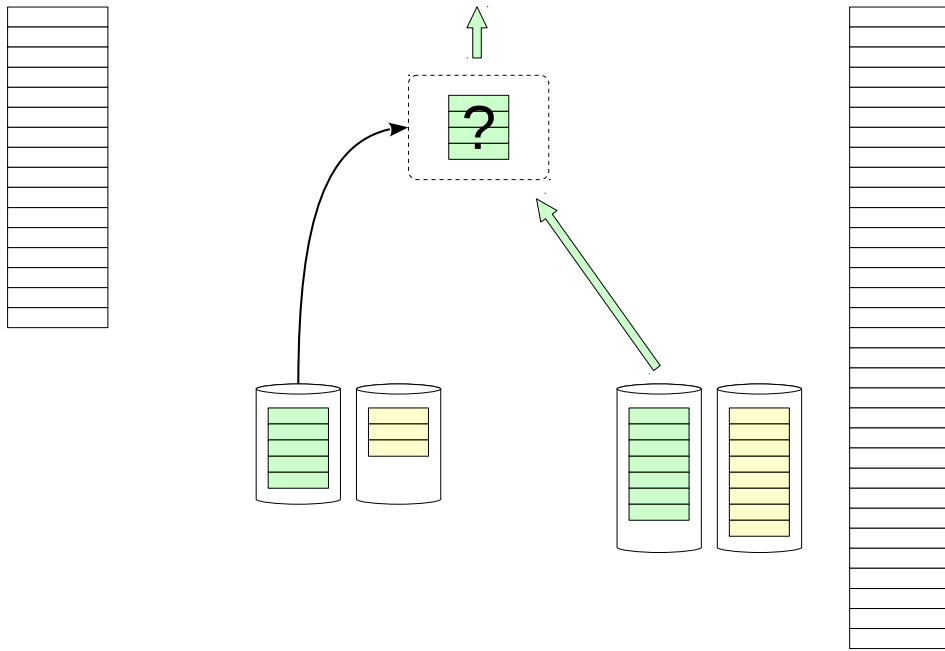
Таким образом, при N пакетах обычно будет использоваться $2(N-1)$ файлов (возможно меньше, если часть пакетов окажется пустыми).

Hash Join – случай 2



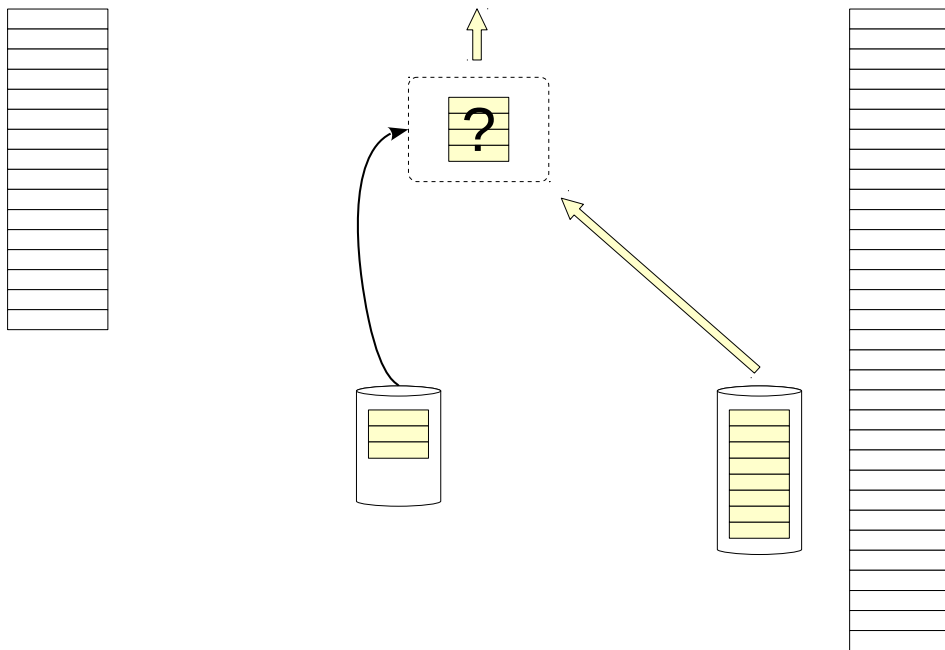
Далее по очереди обрабатываются пакеты со второго и дальше. Сначала из временного файла считываются строки первого набора, образующие хэш-таблицу, затем из другого временного файла считываются и сопоставляются строки второго набора.

Hash Join – случай 2



Процедура повторяется для третьего набора...

Hash Join – случай 2



И для четвертого...

Пока, наконец, все данные не будут обработаны.

Ненужные файлы освобождаются.

Без потери точности

пока есть возможность, информация хранится с точностью до строки

С потерей точности

если память закончилась, происходит огрубление части уже построенной карты до отдельных страниц

при чтении табличной страницы требуется перепроверка условий

требуется примерно 1 МБ памяти на 64 ГБ данных;

ограничение памяти может быть превышено

Как мы помним из темы «Методы доступа», при операции сканирования битовой карты сначала в памяти строится битовая карта, в которой отмечены строки, удовлетворяющие условиям. Затем битовая карта используется, чтобы последовательно прочитать страницы и взять из них нужные строки.

Если битовая карта перестает помещаться в *work_mem*, часть ее «огрубляется» — вместо отдельных строк сохраняется информация с точностью до страниц. Освободившееся место используется для того, чтобы продолжить строить карту. В принципе, при сильно ограниченном *work_mem* и большой выборке, битовая карта может не поместиться в памяти, даже если в ней совсем не останется информации на уровне строк. В таком случае ограничение *work_mem* нарушается — под карту будет выделено столько памяти, сколько необходимо.

Хотя битовая карта всегда находится в оперативной памяти (временные файлы не используются), но при потере точности она требует перепроверки условий при чтении табличных страниц, что сказывается на производительности. А если две битовые карты объединяются, причем часть одной карты точная, а соответствующая часть другой — неточная, то и результат тоже будет вынужденно неточным.

Память серверного процесса

выполнение запросов	$N \cdot work_mem$
временные таблицы	$\leq temp_buffers$
кэш системного каталога	
кэш подготовленных операторов	

Для полноты картины обрисуем использование оперативной памяти серверным процессом в целом. Основные крупные фрагменты:

- память, которая используется при выполнении запросов — то, чему посвящена данная тема. Может использоваться несколько раз по *work_mem* байт.
- память под страницы временных таблиц. Ограничена сверху параметром *temp_buffers*.
- кэш системного каталога. Заполняется по необходимости.
- кэш деревьев и планов запросов. Заполняется при использовании подготовленных операторов.



Ряд операций требуют оперативной памяти

Применяются алгоритмы, использующие доступную память и временные файлы на диске

Чем больше памяти, тем быстрее выполнение

Замечание. В версии 9.5 может наблюдаться эффект деградации производительности сортировки при увеличении `work_mem`. Эта проблема исправлена в 9.6 ([New-9-6-external-sort-guidance](#)).

1. Создайте базу DB18 и в ней повторите ситуацию с хэш-соединением из практики, однако вместо меньшей таблицы используйте СТЕ.
Как изменились расход памяти и время выполнения?
2. С помощью команды `explain analyze` исследуйте влияние доступной памяти на сканирование битовой карты.
3. Как выполняется сортировка, если в запросе указана фраза `limit`?
4. Включите вывод временных файлов в журнал сообщений сервера и проверьте, используется ли дисковая память:
 - а) при использовании функции `generate_series`,
 - б) при использовании СТЕ,
 - в) для расчета значения какой-либо оконной функции.
5. Исследуйте, как параметр `maintenance_work_mem` влияет на скорость создания индекса.

4. Надо учесть, что временные файлы будут использоваться только при нехватке оперативной памяти.