# Стоимостный оптимизатор в YDB: как, зачем и почему?

Павел Велихов

Руководитель команды оптимизатора

YDB

HL 2024 HighLoad++

# Про себя

# Про себя



- Руковожу командой оптимизатора в YDB

# Про себя



- Руковожу командой оптимизатора в YDB
- Занимаюсь разработкой разных СУБД с 1999 года

# Про себя



- Руковожу командой оптимизатора в YDB
- Занимаюсь разработкой разных СУБД с 1999 года
- 6-ая СУБД

# Про себя



- Руковожу командой оптимизатора в YDB
- Занимаюсь разработкой разных СУБД с 1999 года
- 6-ая СУБД
- 4-ая MPP СУБД

# Про себя



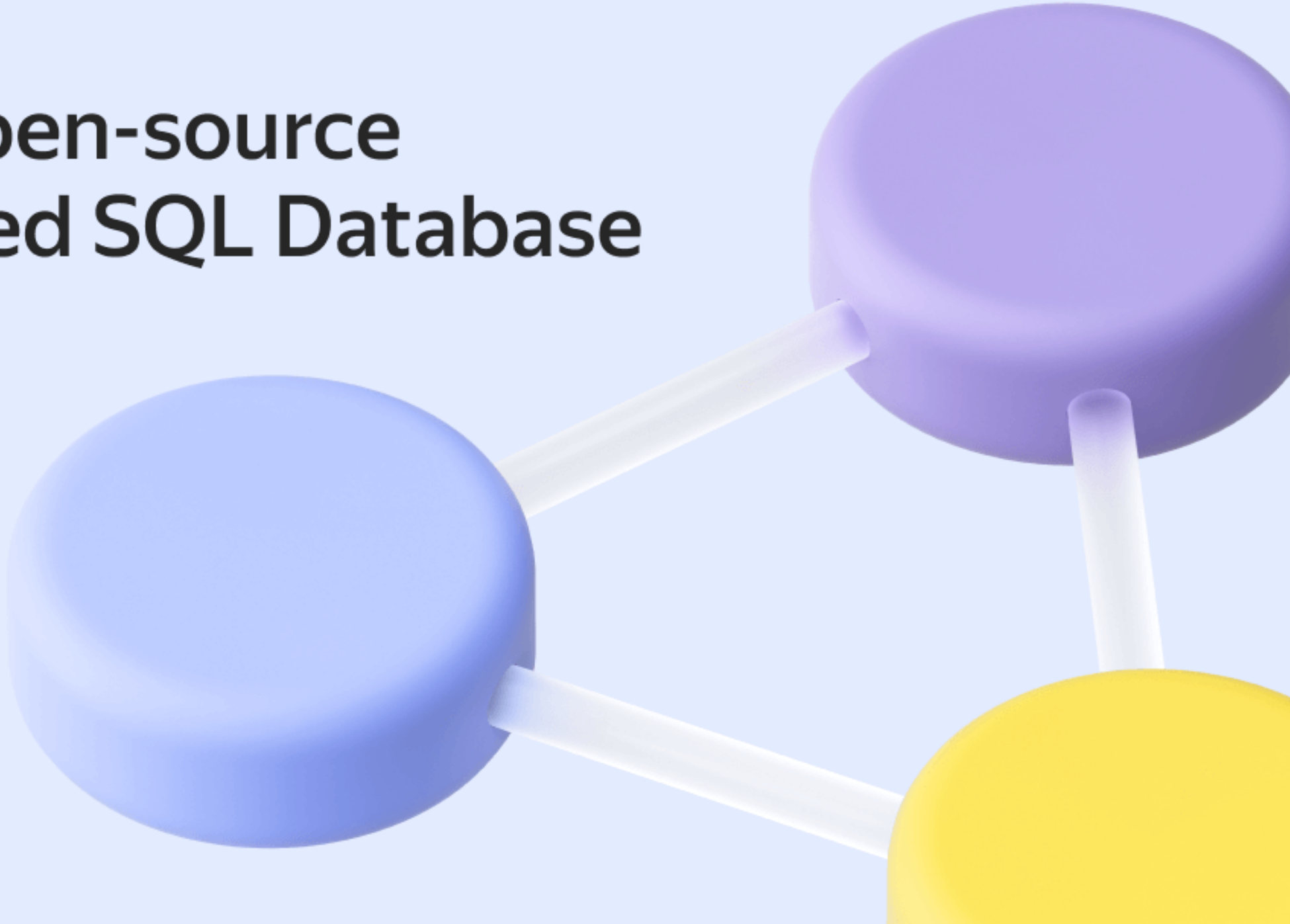
- Руковожу командой оптимизатора в YDB
- Занимаюсь разработкой разных СУБД с 1999 года
- 6-ая СУБД
- 4-ая MPP СУБД
- В каждом случае стараюсь построить оптимизатор запросов

# Про себя



- Руковожу командой оптимизатора в YDB
- Занимаюсь разработкой разных СУБД с 1999 года
- 6-ая СУБД
- 4-ая MPP СУБД
- В каждом случае стараюсь построить оптимизатор запросов
  - Не все проекты доживали до этой стадии

# YDB — open-source Distributed SQL Database



# YDB: Краткая история

# YDB: Краткая история



- 10 лет разработки

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система
- Частая повторяющаяся история в Яндексе

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система
- Частая повторяющаяся история в Яндексе
  - Создаем сервис с обычной СУБД

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система
- Частая повторяющаяся история в Яндексе
  - Создаем сервис с обычной СУБД
  - Резко растет масштаб, СУБД не справляется

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система
- Частая повторяющаяся история в Яндексе
  - Создаем сервис с обычной СУБД
  - Резко растет масштаб, СУБД не справляется
  - Команда тратит огромные усилия на масштабирование СУБД

# YDB: Краткая история



- 10 лет разработки
- Изначально — высоконагруженная масштабируемая OLTP система
- Частая повторяющаяся история в Яндексе
  - Создаем сервис с обычной СУБД
  - Резко растет масштаб, СУБД не справляется
  - Команда тратит огромные усилия на масштабирование СУБД
- Широко используется в инфраструктуре Яндекса, в Яндекс.Облаке и on-prem

# YDB: Аналитика

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику
  - Колоночное хранение

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику
  - Колоночное хранение
  - Блочный движок исполнения

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику
  - Колоночное хранение
  - Блочный движок исполнения
  - Workload Manager

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику
  - Колоночное хранение
  - Блочный движок исполнения
  - Workload Manager
  - Стоимостный оптимизатор

# YDB: Аналитика



- Всегда возникает потребность, когда накапливаются данные
- Данных в YDB накапливается много
- Несколько лет назад начали строить полноценную аналитику
  - Колоночное хранение
  - Блочный движок исполнения
  - Workload Manager
  - Стоимостный оптимизатор
- Самый сложный компонент — оптимизатор

# Зачем YDB качественный оптимизатор

# Зачем YDB качественный оптимизатор



- Бородин: “Оптимизатор — это как сюжет в порно фильме: он должен присутствовать, но его качество не имеет значения”

# Зачем YDB качественный оптимизатор



- Бородин: “Оптимизатор — это как сюжет в порно фильме: он должен присутствовать, но его качество не имеет значения”
- На самом деле это неправда

# Зачем YDB качественный оптимизатор



- Бородин: “Оптимизатор — это как сюжет в порно фильме: он должен присутствовать, но его качество не имеет значения”
- На самом деле это неправда
- Зависит еще сильно от области применения YDB

# Зачем YDB качественный оптимизатор



- Бородин: “Оптимизатор — это как сюжет в порно фильме: он должен присутствовать, но его качество не имеет значения”
- На самом деле это неправда
- Зависит еще сильно от области применения YDB
- OLTP — иногда можно прожить со средненьким оптимизатором

# Зачем YDB качественный оптимизатор



- Бородин: “Оптимизатор — это как сюжет в порно фильме: он должен присутствовать, но его качество не имеет значения”
- На самом деле это неправда
- Зависит еще сильно от области применения YDB
- OLTP — иногда можно прожить со средненьким оптимизатором
- OLAP — нужен хороший оптимизатор

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |
| **Чтение/запись данных** | Мало | Много |

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |
| **Чтение/запись данных** | Мало | Много |
| **Схема данных** | Нормализованная | Традиционно денормализованная, но есть тренд на гипернормализацию |

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |
| **Чтение/запись данных** | Мало | Много |
| **Схема данных** | Нормализованная | Традиционно денормализованная, но есть тренд на гипернормализацию |
| **Сложность запросов** | Лукапы | Много джоинов и агрегатов |

# OLTP vs OLAP



| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |
| **Чтение/запись данных** | Мало | Много |
| **Схема данных** | Нормализованная | Традиционно денормализованная, но есть тренд на гипернормализацию |
| **Сложность запросов** | Лукапы | Много джоинов и агрегатов |
| **Разнородность запросов** | Низкая | Высокая |

# OLTP vs OLAP

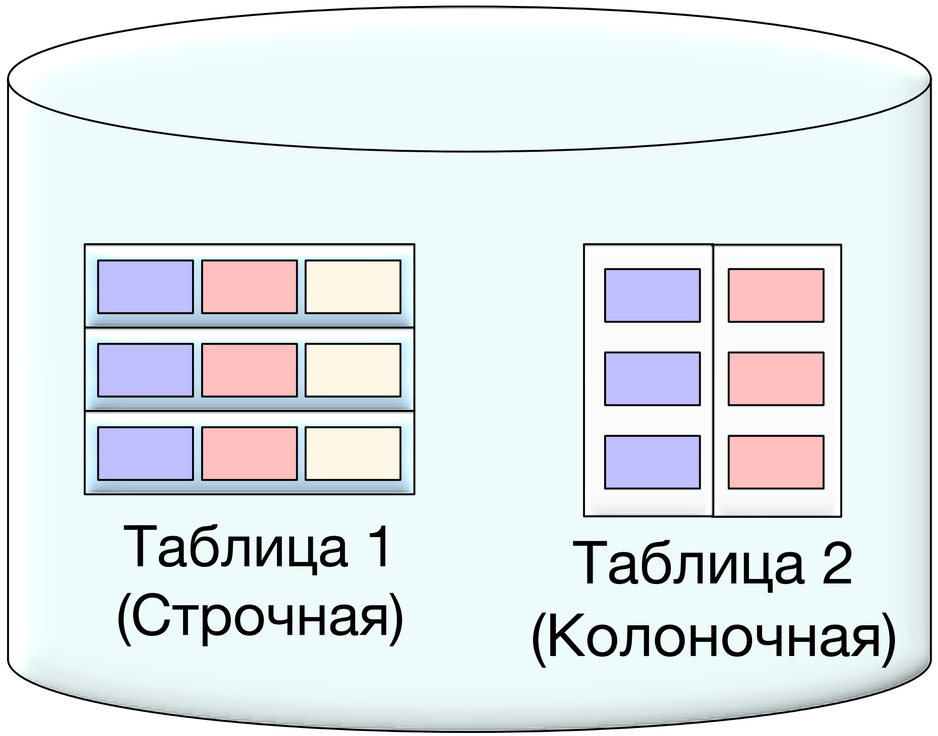


| | OLTP | OLAP |
|---|---|---|
| **Частота запросов** | Миллионы в секунду | Несколько в секунду |
| **Чтение/запись данных** | Мало | Много |
| **Схема данных** | Нормализованная | Традиционно денормализованная, но есть тренд на гипернормализацию |
| **Сложность запросов** | Лукапы | Много джоинов и агрегатов |
| **Разнородность запросов** | Низкая | Высокая |
| **Требования отклика** | до 200ms | От 200ms до нескольких суток |

# HTAP



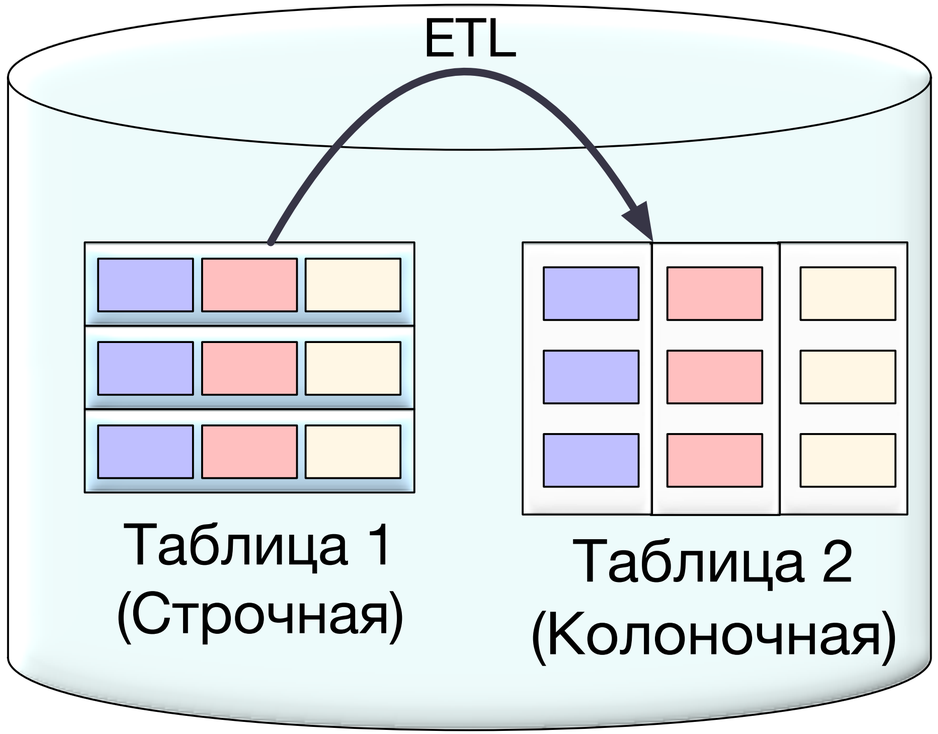
- Комбинация OLTP + OLAP
- Возможны разные “степени”



# HTAP



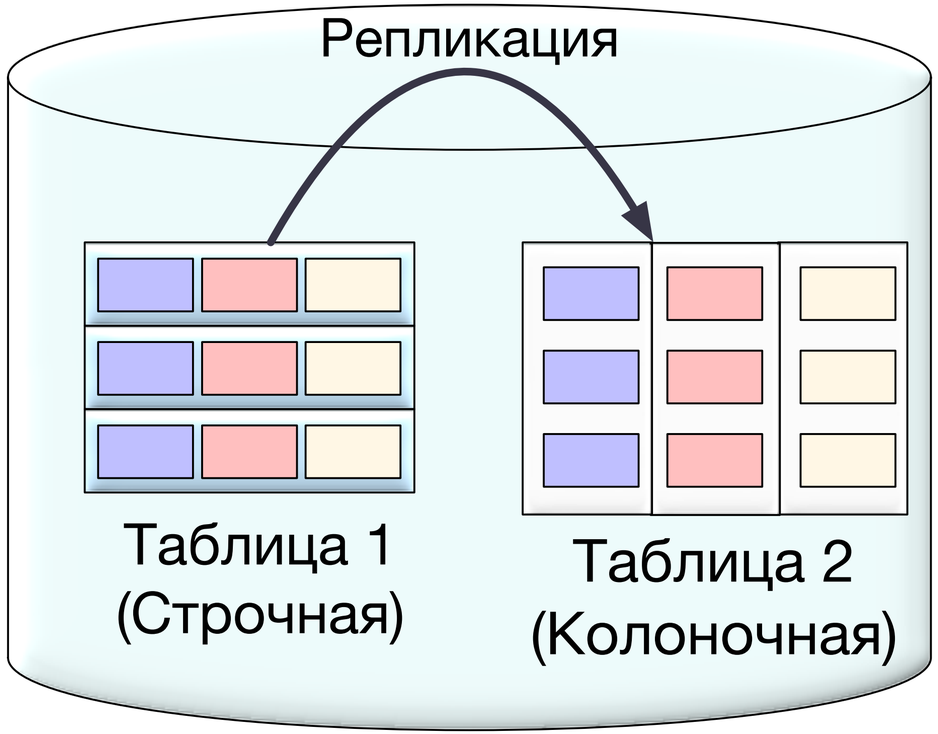
- Комбинация OLTP + OLAP
- Возможны разные “степени”
- Две базы в одной
- Разные схемы
- ETL процесс



# HTAP



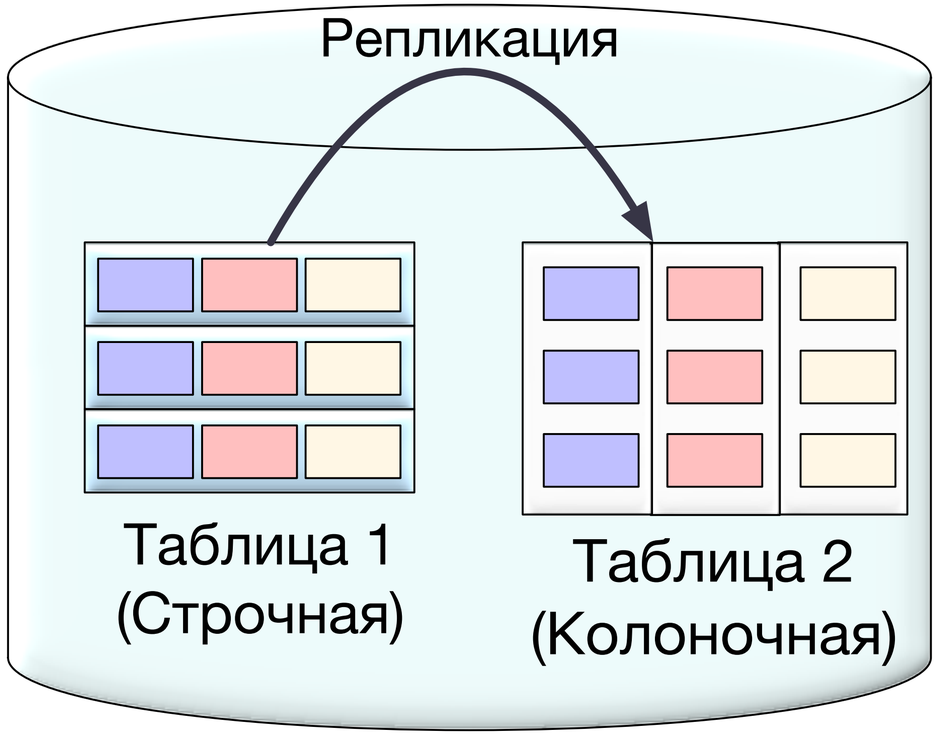
- Комбинация OLTP + OLAP
- Возможны разные “степени”
- Две базы в одной
- Одинаковые схемы
- Авторепликация



# HTAP



- Комбинация OLTP + OLAP
- Возможны разные “степени”
- ~~Две базы в одной~~
- Одна ACID база
- Одинаковые схемы
- Авторепликация
- Оптимизатор выбирает подходящие таблицы для каждого запроса



# Как мы решили развивать оптимизатор

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим
- Сначала делаем для тяжелой аналитики

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим
- Сначала делаем для тяжелой аналитики
  - Годы разработки

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим
- Сначала делаем для тяжелой аналитики
  - Годы разработки
  - MVP!

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим
- Сначала делаем для тяжелой аналитики
  - Годы разработки
  - MVP!
  - Затем уже OLTP

# Как мы решили развивать оптимизатор



- OLTP может долго жить без стоимостного оптимизатора
- Если его делать, сразу нужен очень хороший
- Для аналитики стоимостной оптимизатор необходим
- Сначала делаем для тяжелой аналитики
  - Годы разработки
  - MVP!
  - Затем уже OLTP
  - В будущем HTAP

# Основные компоненты оптимизатора



- Энумератор планов
- Оценка кардинальности
- Оценочная функция
- Сбор статистики для оптимизатора

# Энумератор планов

# Энумератор: Top Down

# Энумератор: Top Down



- Очень удобный и гибкий подход для разработчиков

# Энумератор: Top Down



- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов

# Энумератор: Top Down

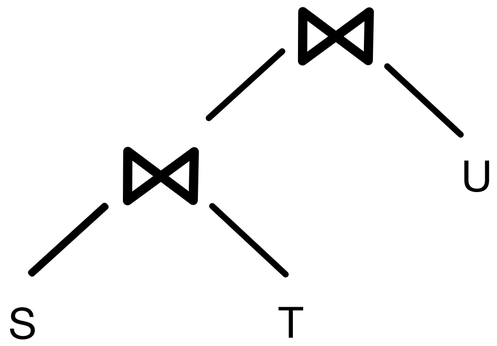


- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов
  - Например

# Энумератор: Top Down



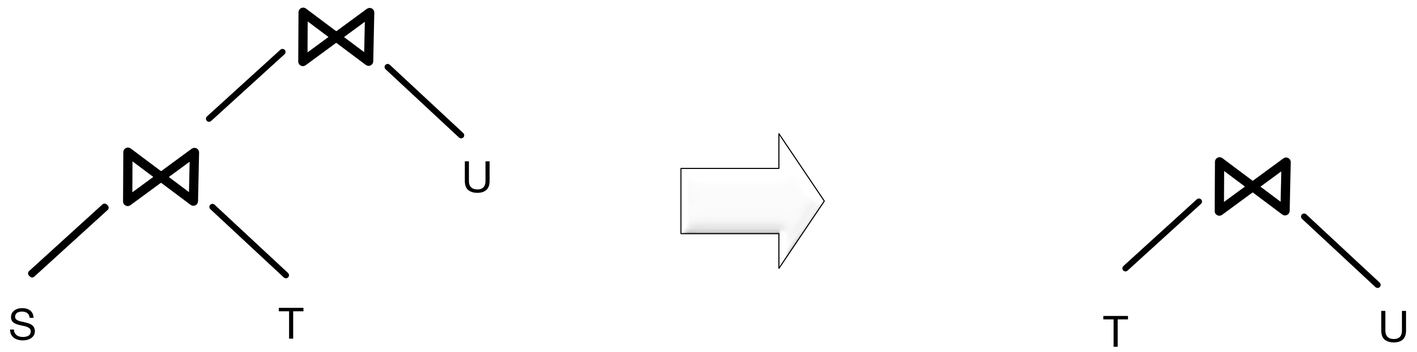
- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов:
  - Например:



# Энумератор: Top Down



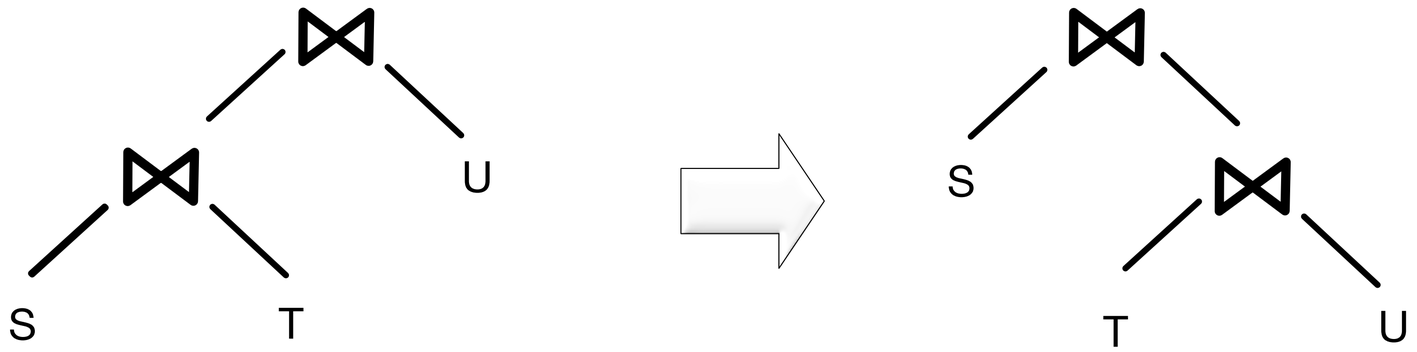
- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов:
  - Например:



# Энумератор: Top Down



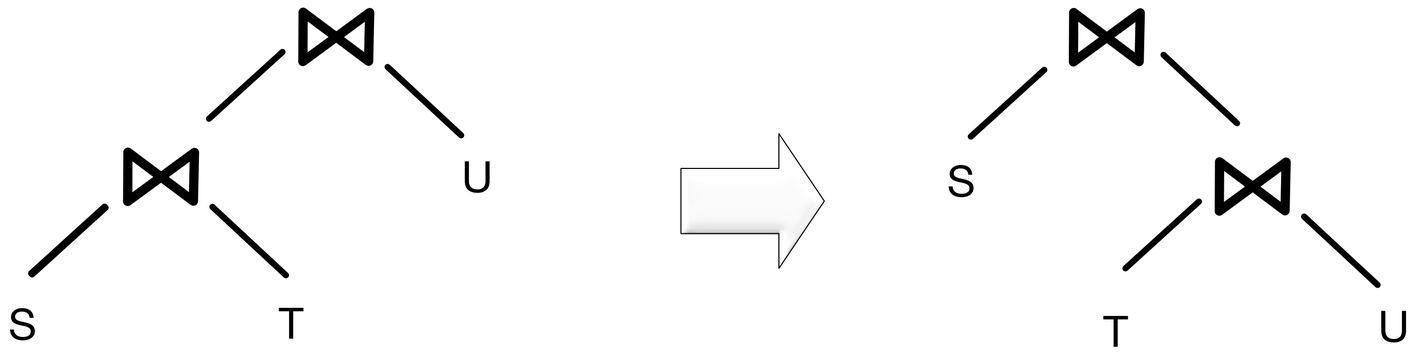
- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов:
  - Например:



# Энумератор: Top Down



- Очень удобный и гибкий подход для разработчиков
- Пишем локальные правила трансформации планов:
  - Например:




- Оптимизатор сам применяет правила, оценивает стоимости, выбирает оптимальный план
- Поддерживает произвольные правила

# Энумератор: Top Down



- Проблема — не масштабируется
- Не поддерживает планы с 10+ джоинами

# Энумератор: Bottom Up

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования
- Можно честно перебирать больше 100 джоинов

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования
- Можно честно перебирать больше 100 джоинов
- Для типичных аналитических запросов: 18-30 джоинов

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования
- Можно честно перебирать больше 100 джоинов
- Для типичных аналитических запросов: 18-30 джоинов
- Можно добавить в перебор агрегаты и сложные фильтры

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования
- Можно честно перебирать больше 100 джоинов
- Для типичных аналитических запросов: 18-30 джоинов
- Можно добавить в перебор агрегаты и сложные фильтры
- Для распределенных систем: сортировки и распределения

# Энумератор: Bottom Up



- Перебор джоинов с помощью динамического программирования
- Можно честно перебирать больше 100 джоинов
- Для типичных аналитических запросов: 18-30 джоинов
- Можно добавить в перебор агрегаты и сложные фильтры
- Для распределенных систем: сортировки и распределения
- Итог: выбрали Bottom Up

# Динамическое программирование

# Динамическое программирование



- Перебираем все возможные деревья

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план
  - Принцип оптимальности Беллмана

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план
  - Принцип оптимальности Беллмана
  - Оптимальное дерево состоит из оптимальных поддеревьев

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план
  - Принцип оптимальности Беллмана
  - Оптимальное дерево состоит из оптимальных поддеревьев
  - Заводим табличку и там эффективно храним

# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план
  - Принцип оптимальности Беллмана
  - Оптимальное дерево состоит из оптимальных поддеревьев
  - Заводим табличку и там эффективно храним
    - Разные варианты планов

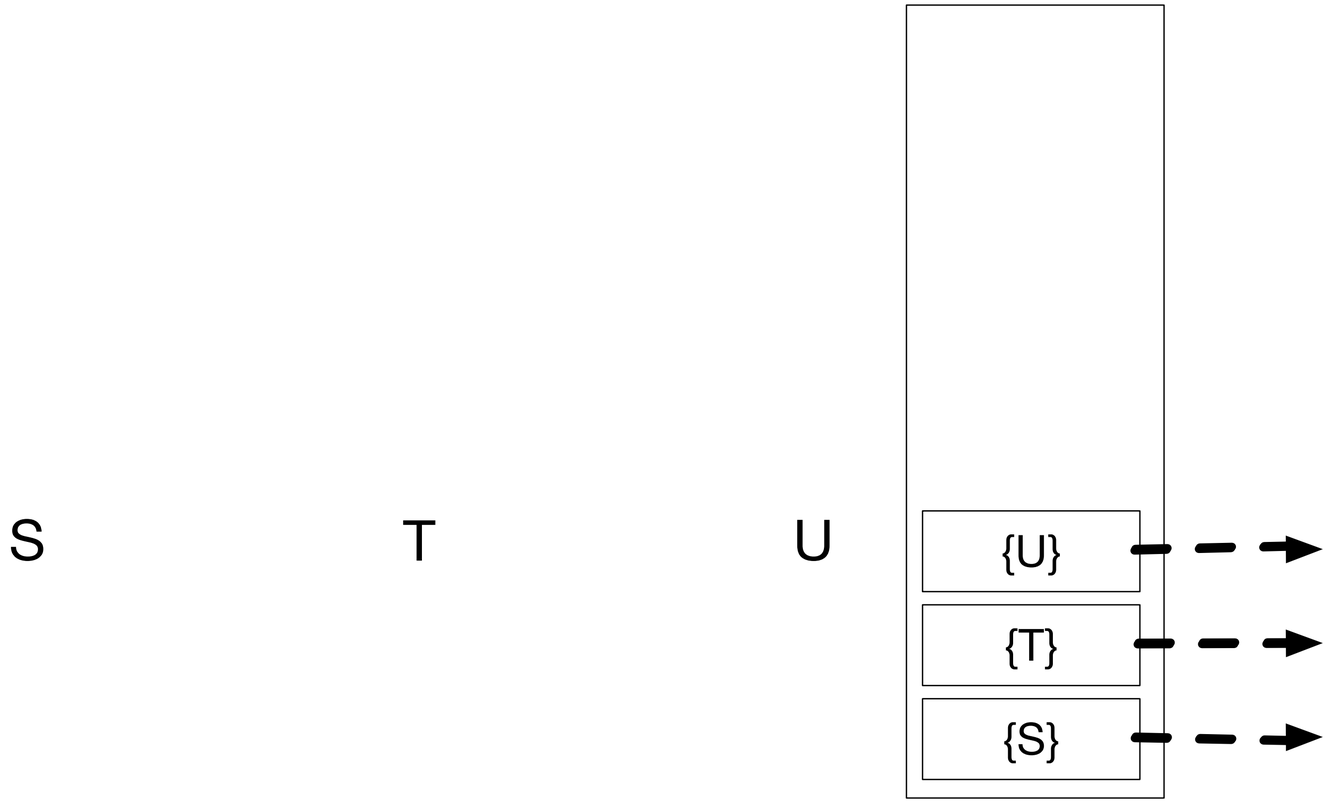
# Динамическое программирование



- Перебираем все возможные деревья
- Их очень много: Порядка n!
- Но: для каждого дерева не строим и не оцениваем заново весь план
  - Принцип оптимальности Беллмана
  - Оптимальное дерево состоит из оптимальных поддеревьев
  - Заводим табличку и там эффективно храним
    - Разные варианты планов
    - Оценки кардинальности, стоимости, метаданные

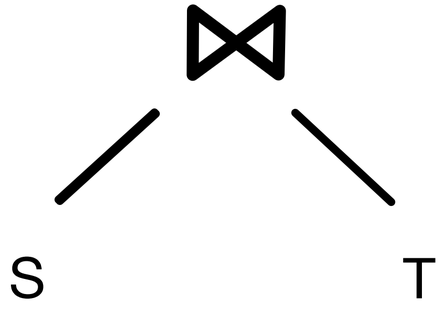
# Динамическое программирование

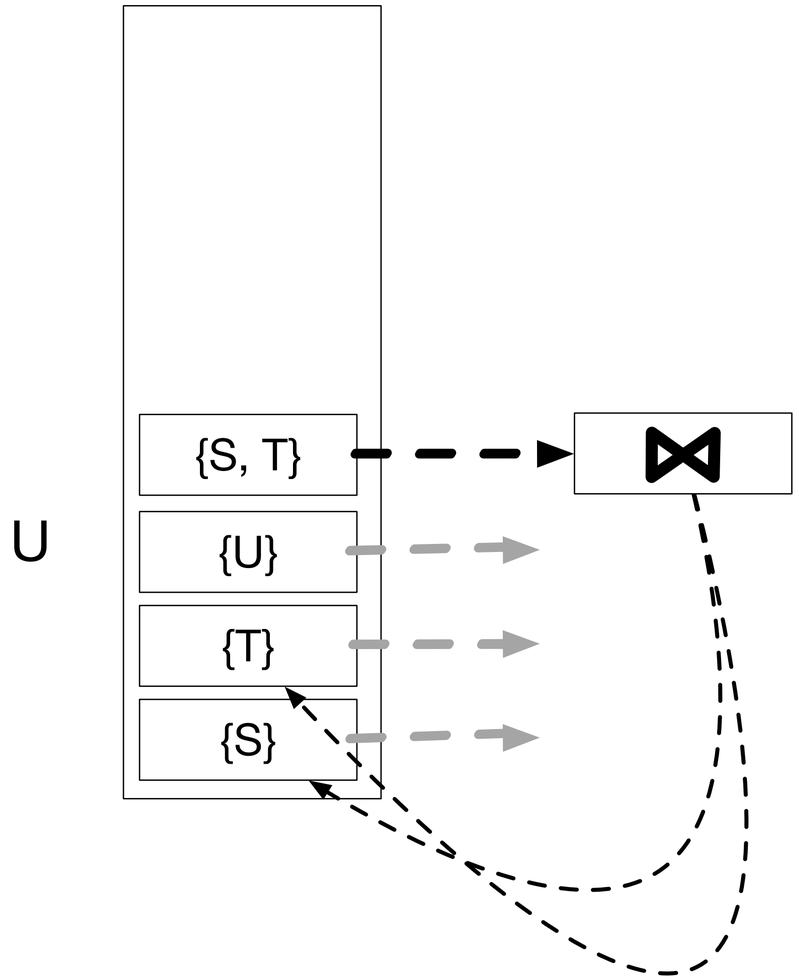




# Динамическое программирование

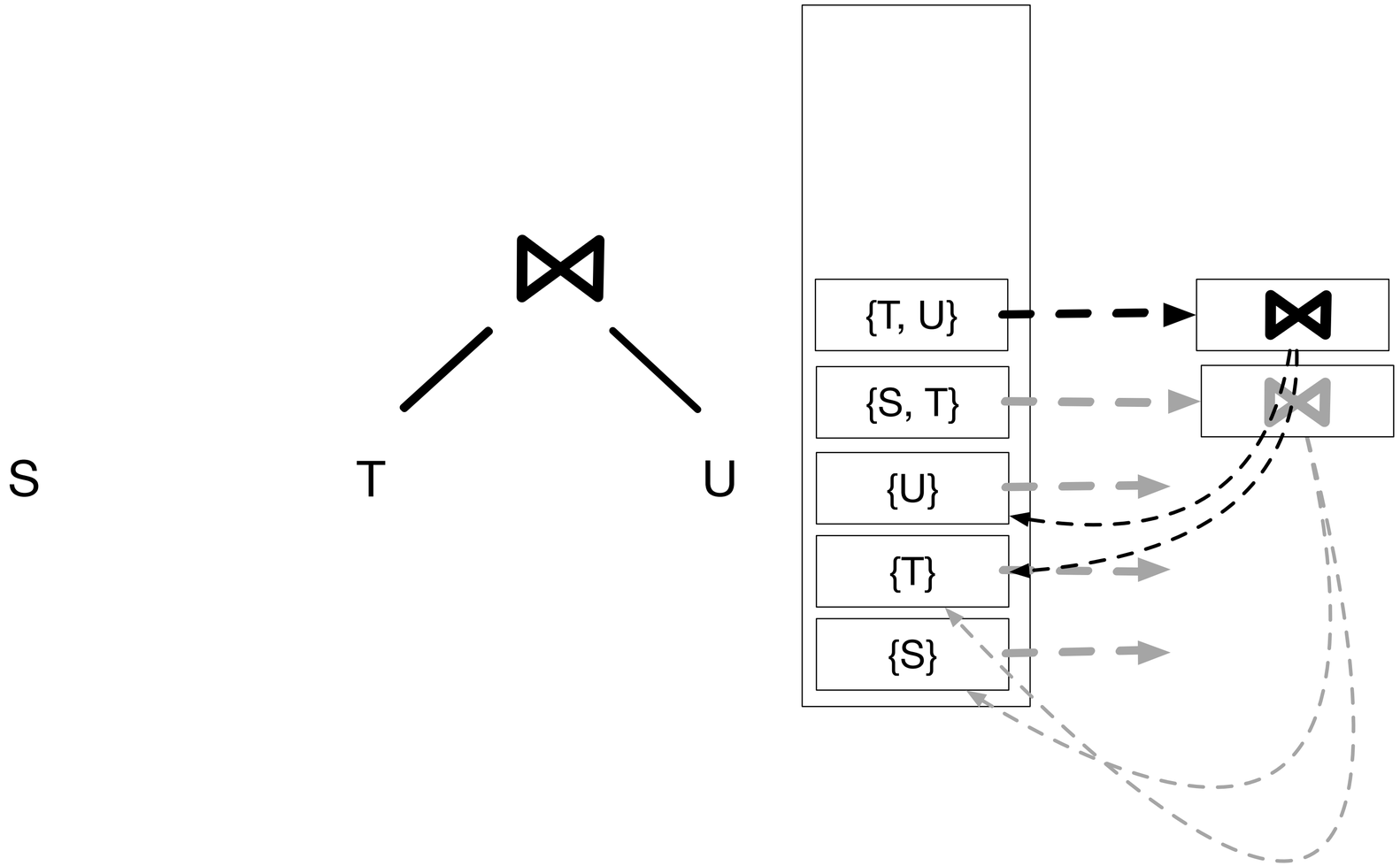




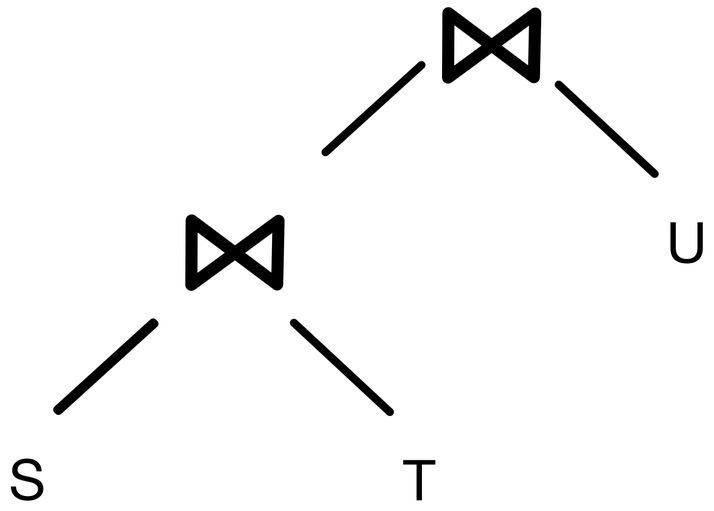
# Динамическое программирование

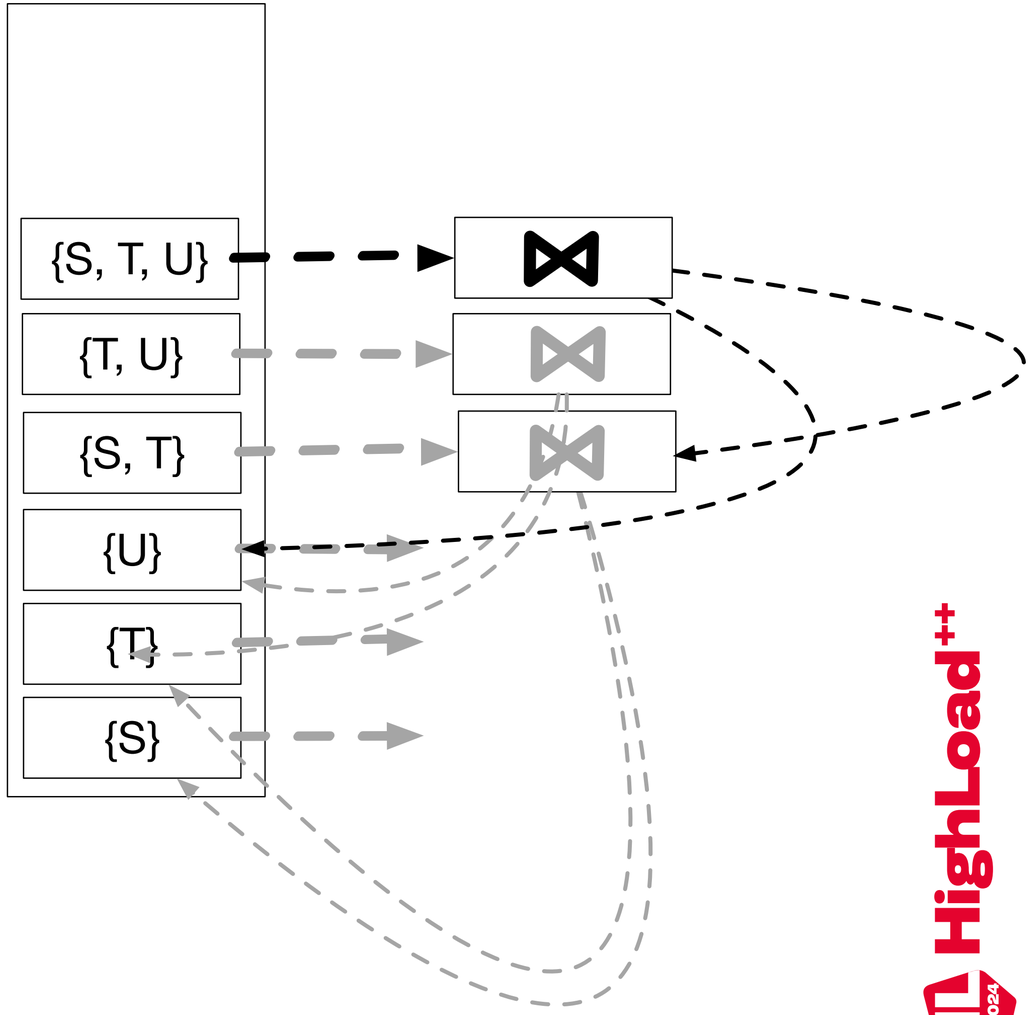




# Динамическое программирование





# Алгоритм DPhyp

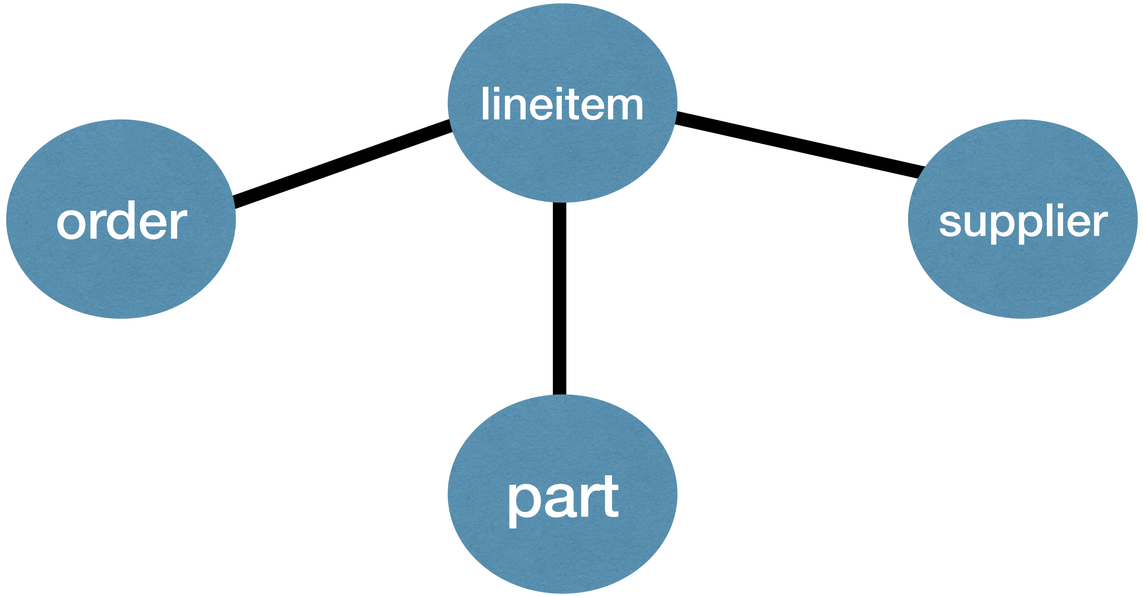


```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```

# Алгоритм DPhyp



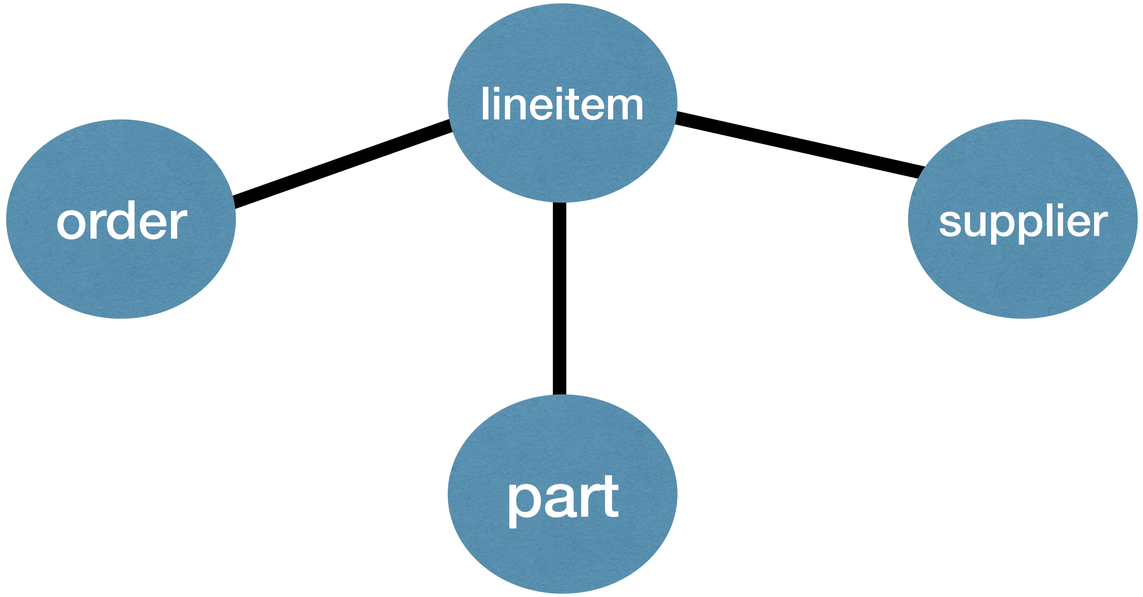
```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



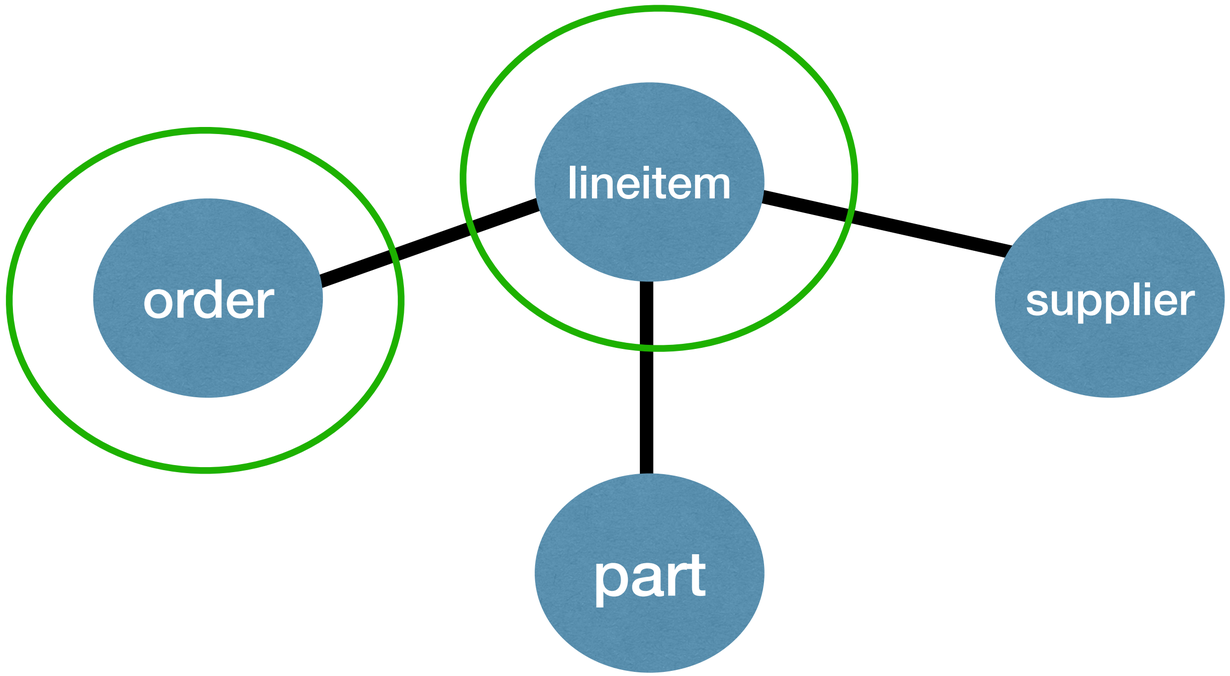

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



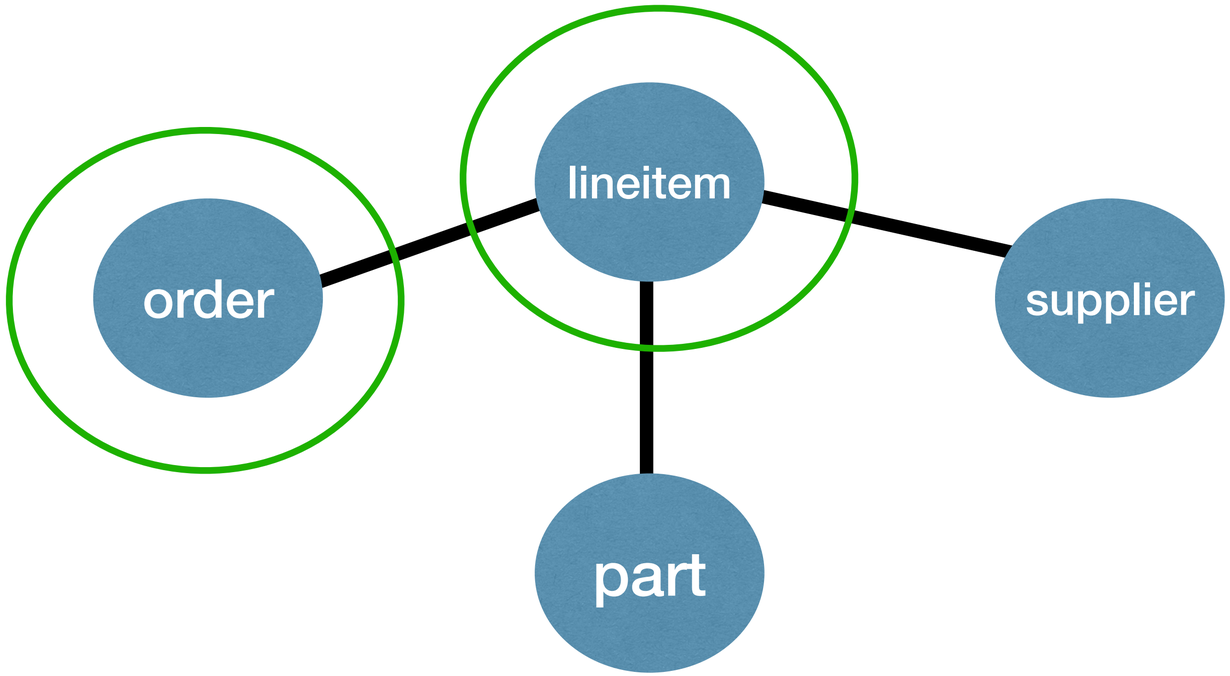

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



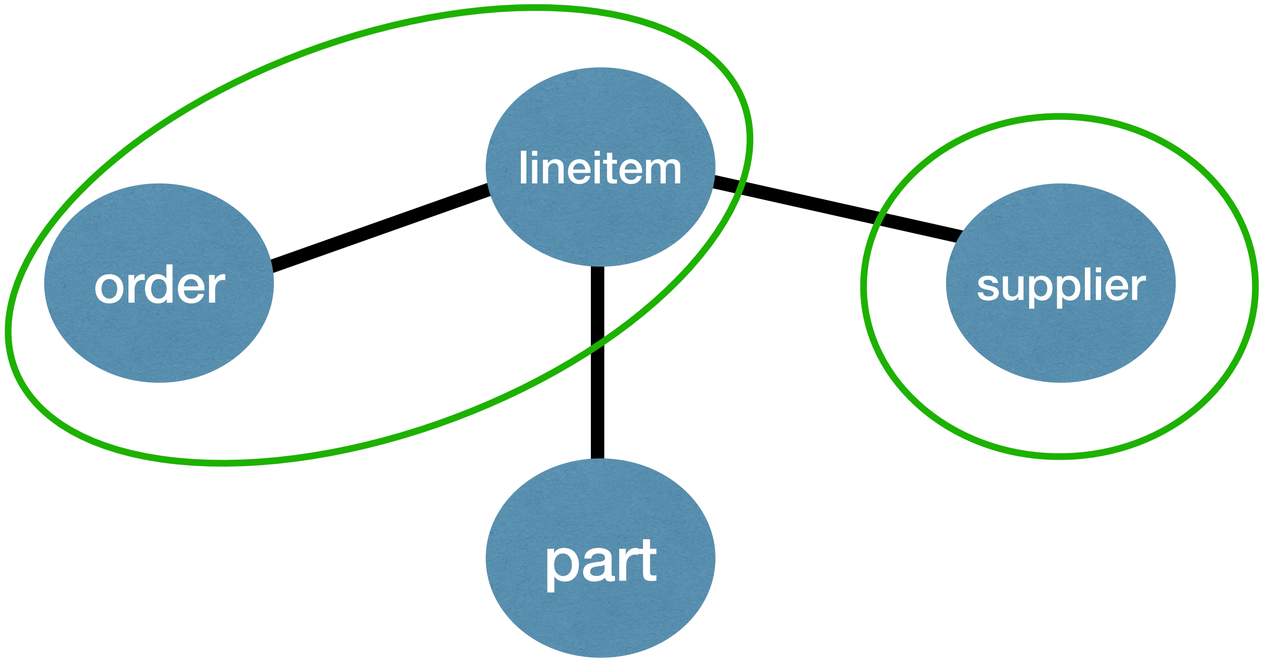

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



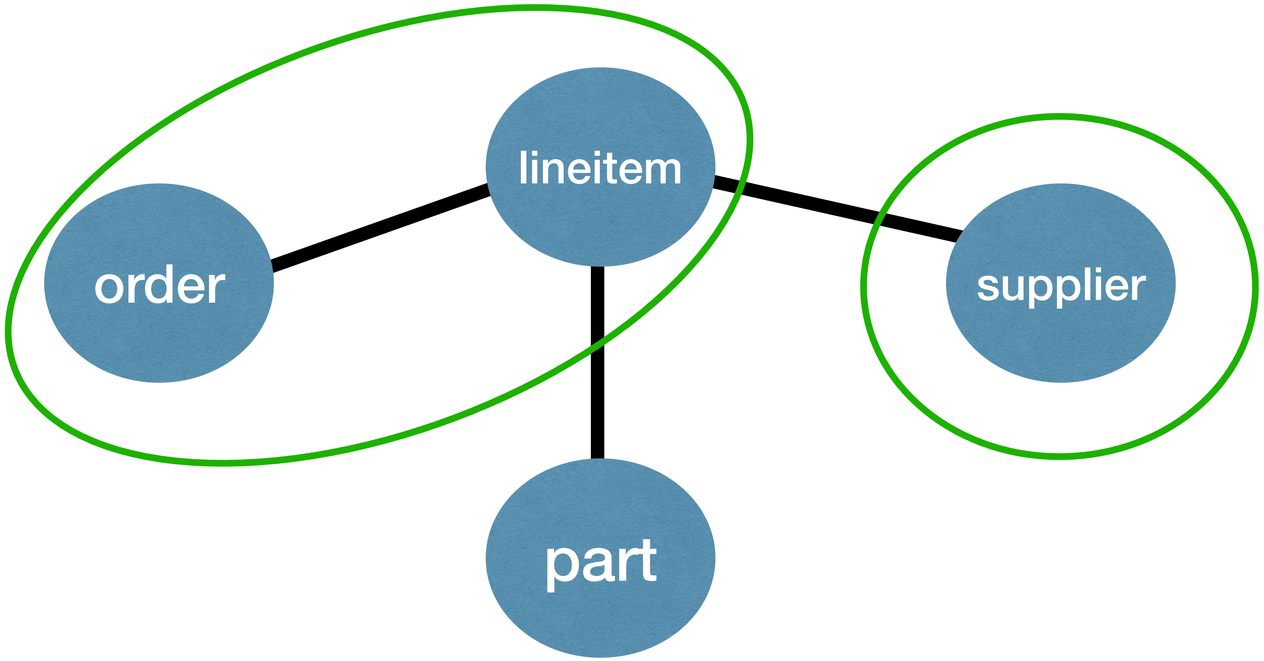

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



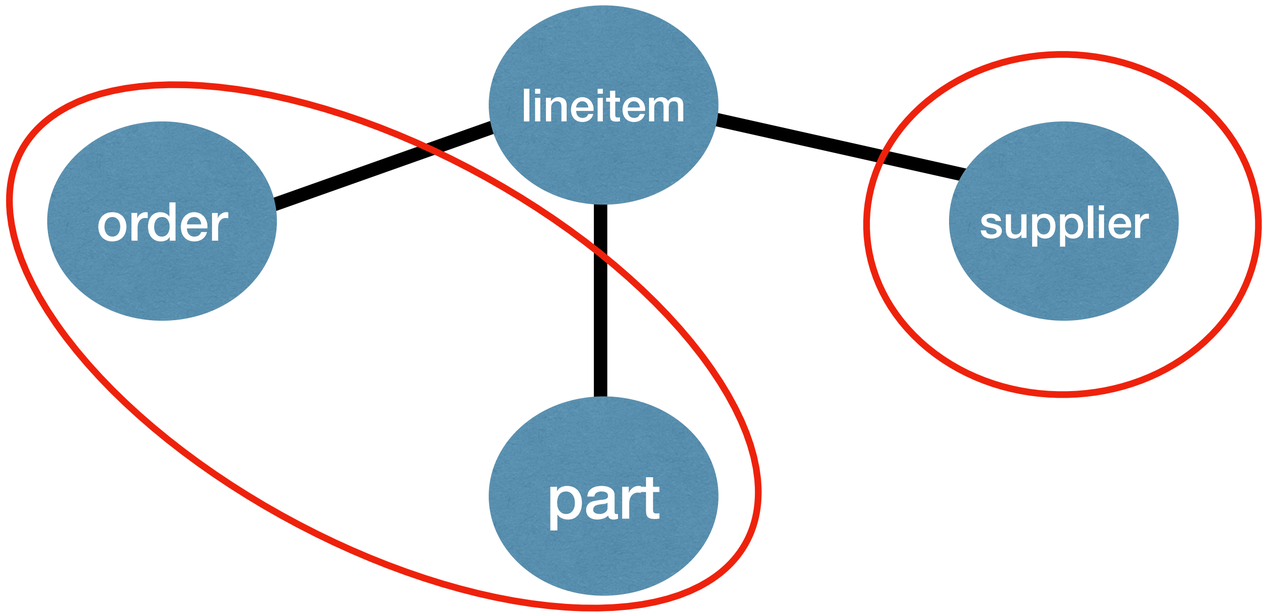

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```



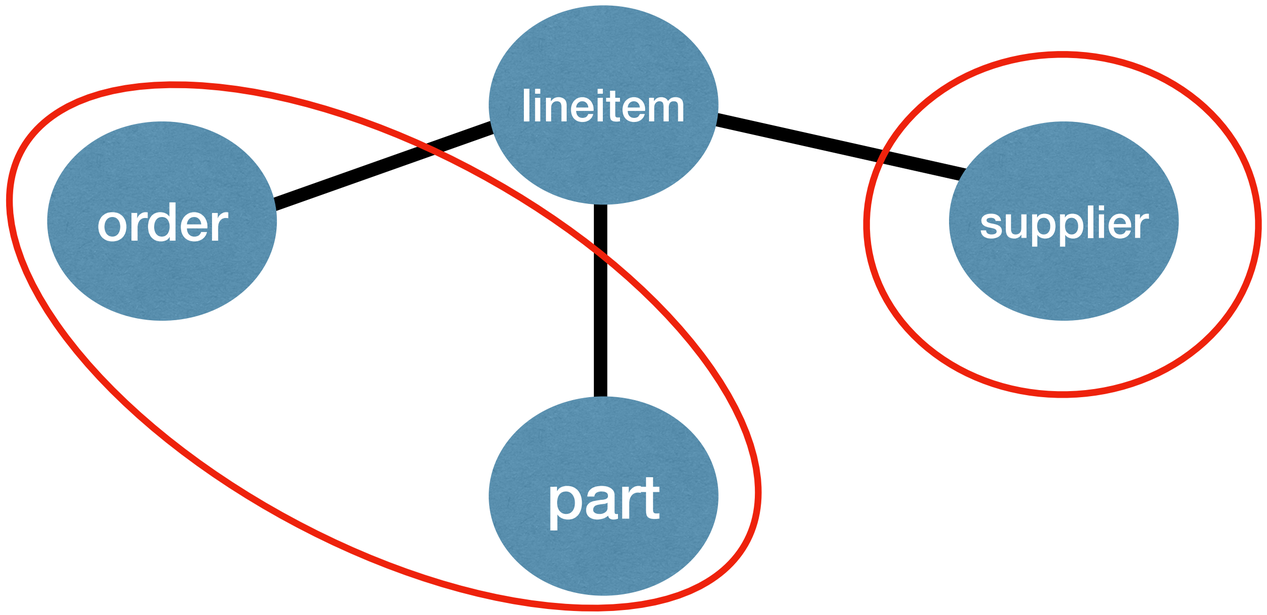

Перебираем пары

множеств вершин

# Алгоритм DPhyp



```
SELECT count(*)
FROM order, lineitem, part, supplier
WHERE
     o_orderkey = l_orderkey AND
     l_partkey = p_partkey AND
     l_suppkey = s_suppkey
```

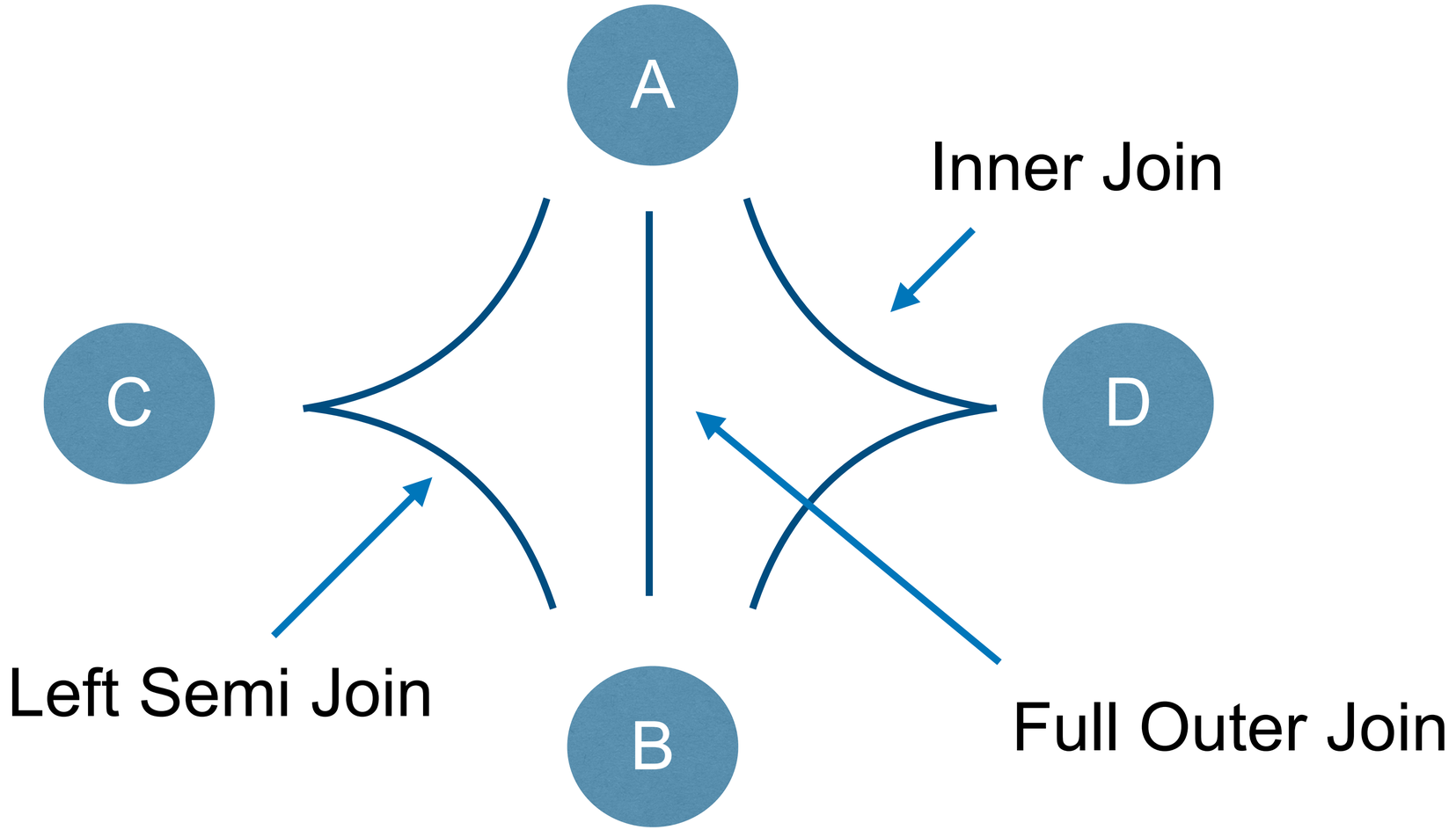



Перебираем пары

множеств вершин

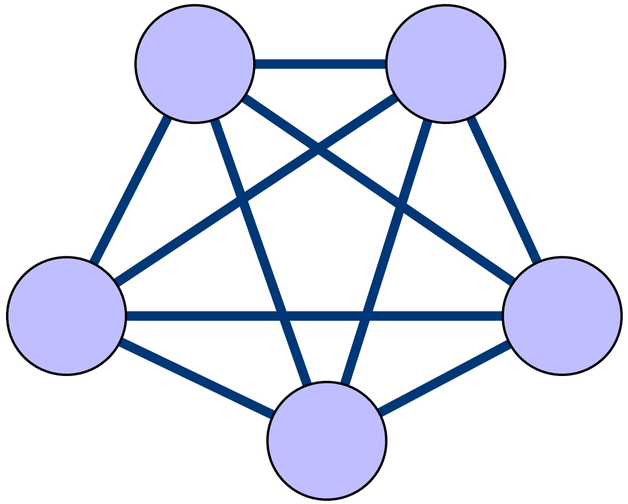
# Гиперграф запроса





# Что получилось на данный момент



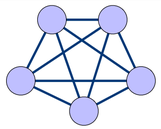


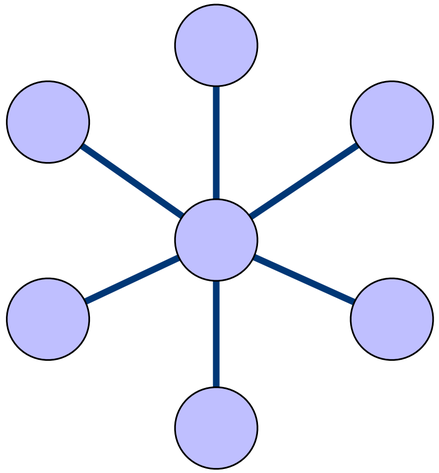
Клика

14-15 джоинов

# Что получилось на данный момент



| Топология Графа | Максимальное кол-во джоинов |
|---|---|
| **Клика**  | 14-15 |

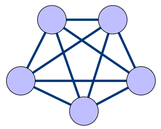
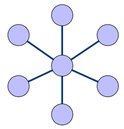


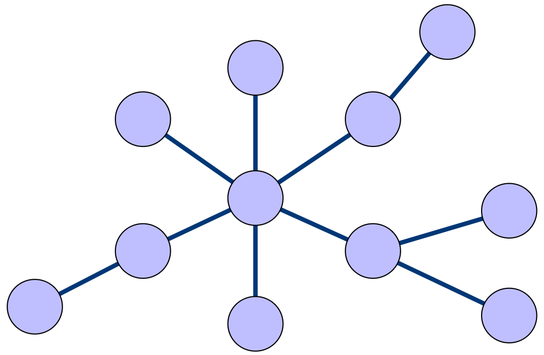
18 джоинов

Звезда

# Что получилось на данный момент



| Топология Графа | Максимальное кол-во джоинов |
|---|---|
| **Клика**  | 14-15 |
| **Звезда**  | 18 |

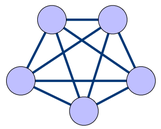
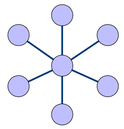
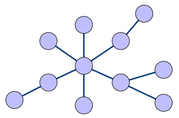


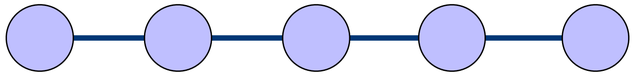
Снежинка

18-30 джоинов

# Что получилось на данный момент



| Топология Графа | Максимальное кол-во джоинов |
|---|---|
| **Клика**  | 14-15 |
| **Звезда**  | 18 |
| **Снежинка**  | 18-30+ |

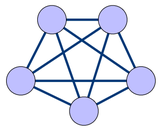
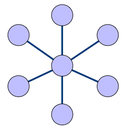
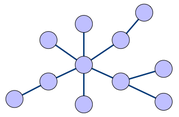
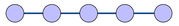


Цепочка

150 джоинов

# Что получилось на данный момент



| Топология Графа | Максимальное кол-во джоинов |
|---|---|
| **Клика**  | 14-15 |
| **Звезда**  | 18 |
| **Снежинка**  | 18-30+ |
| **Цепочка**  | 150 |

# Дальше — сложнее

# Дальше — сложнее



- Некоторые оптимизации нарушают принцип оптимальности

# Дальше — сложнее

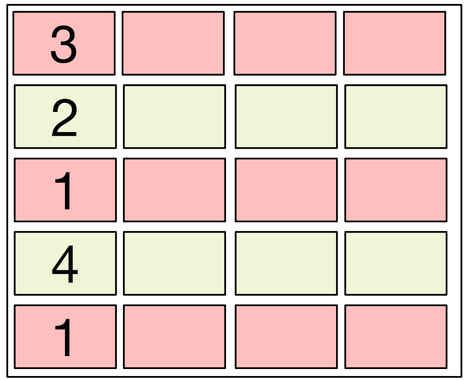


- Некоторые оптимизации нарушают принцип оптимальности
- Самый значимый: merge join vs hash join

# Hash Join

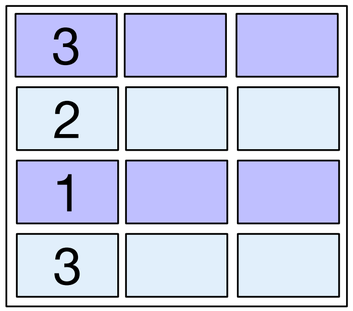


Join Key




Probe

Join Key

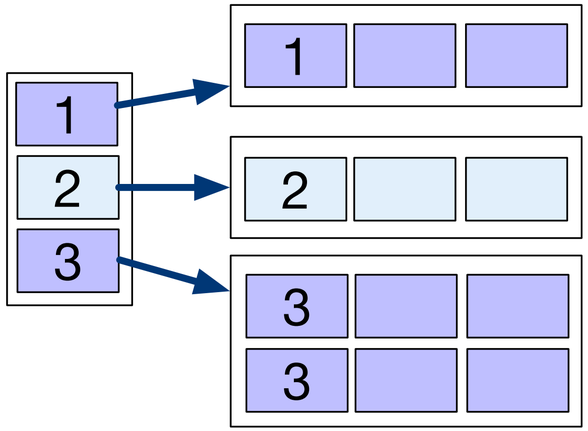



Build

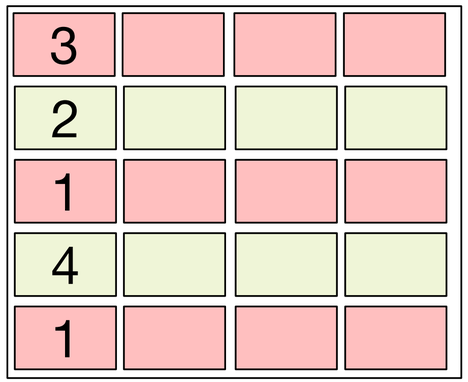
# Hash Join



Hashtable

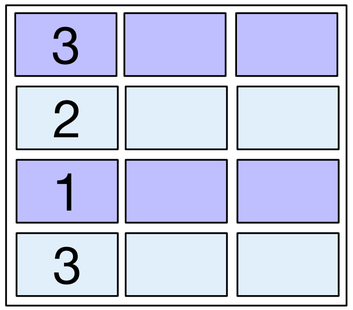



Join Key




Probe

Join Key




Build

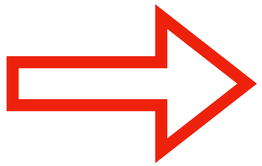
# Hash Join

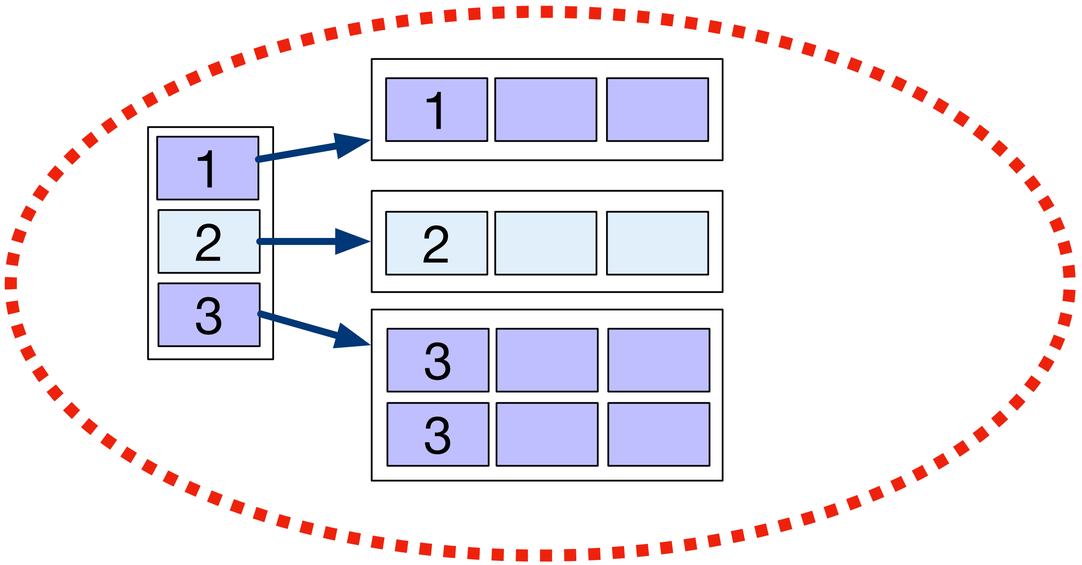


Очень дорого

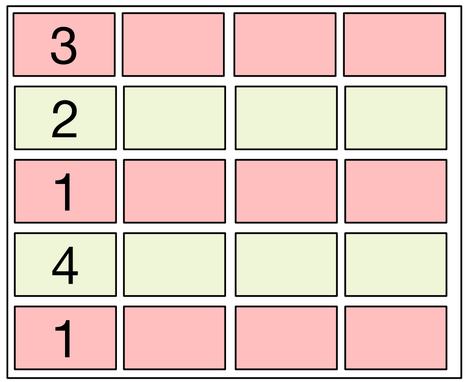
- по памяти

- по вычислениям



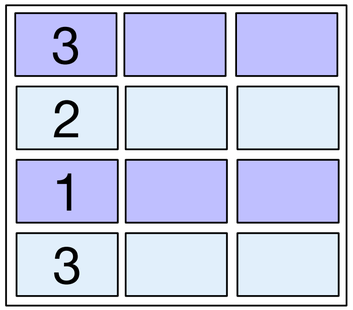



Join Key




Probe

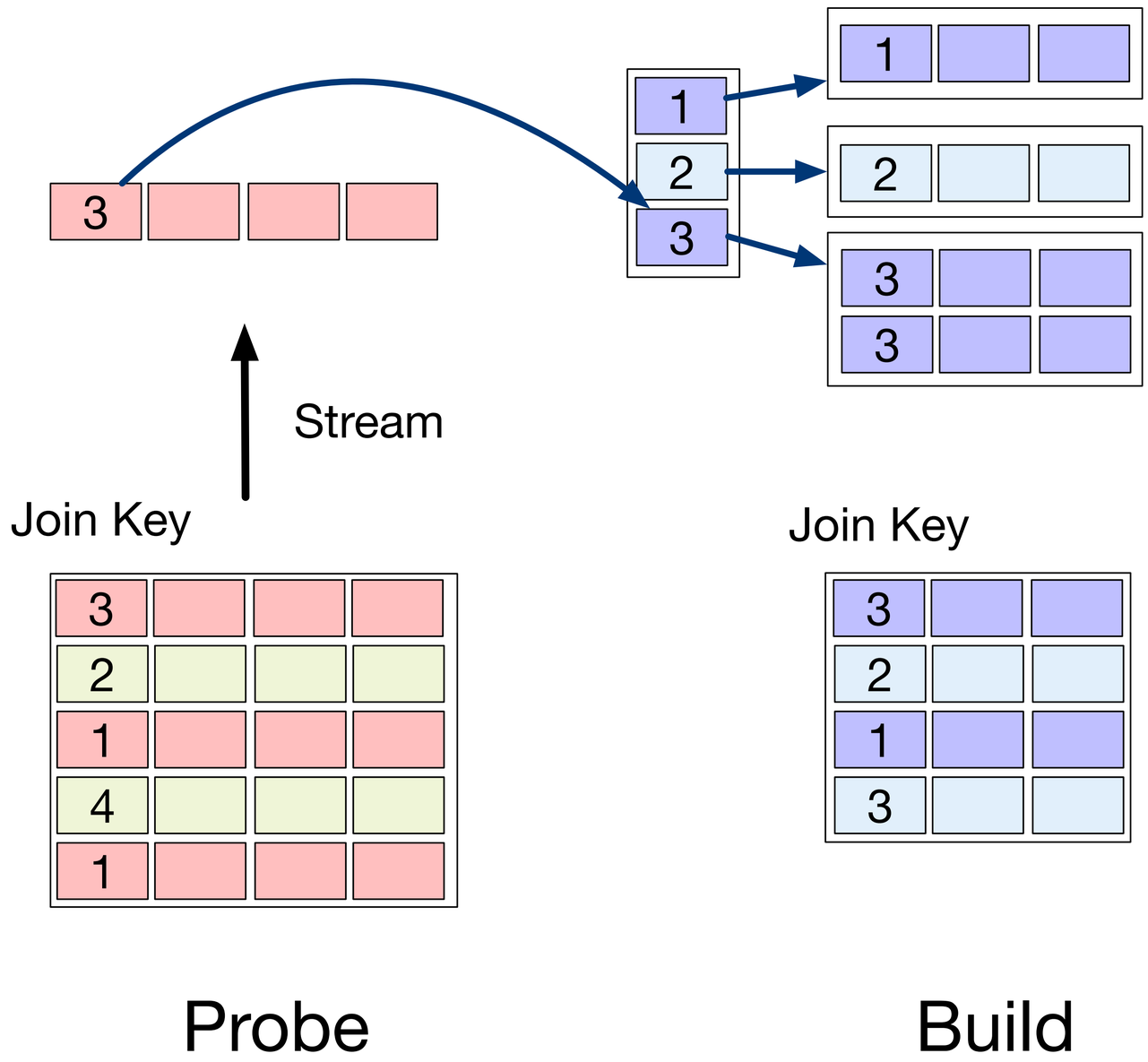
Join Key




Build

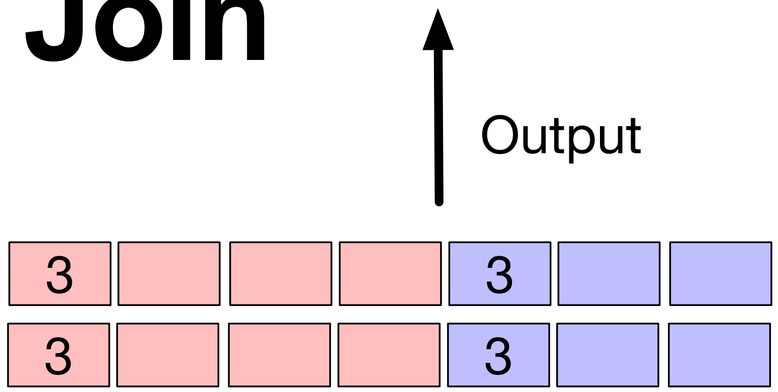
# Hash Join

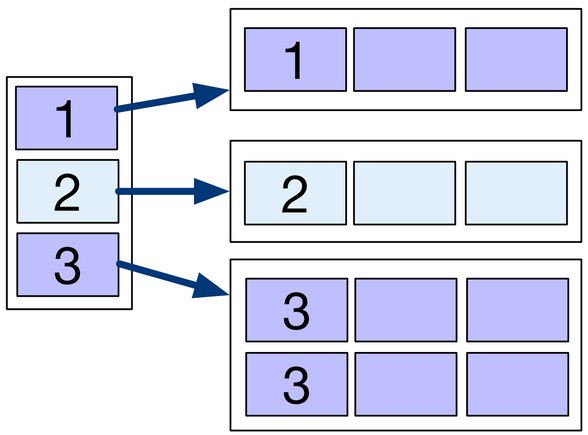


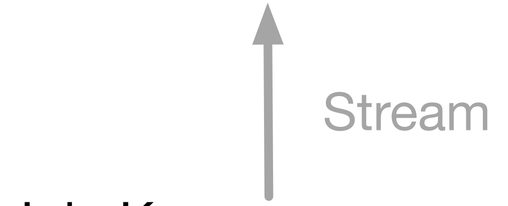

Hashtable
1
2
3
1
2
3
3
3
Stream
Join Key
3
2
1
4
1
Join Key
3
2
1
3
Probe
Build

# Hash Join

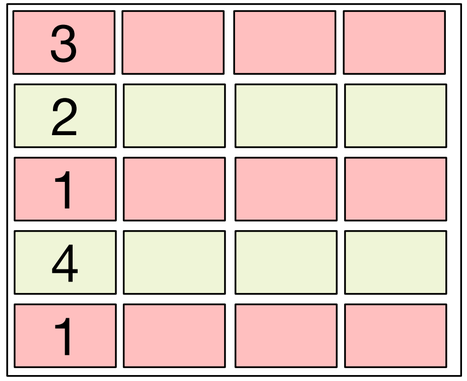




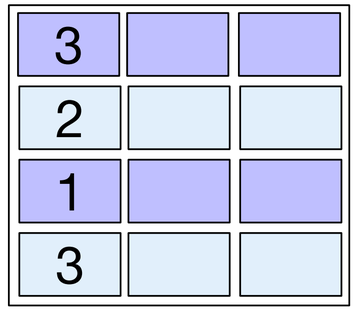




Join Key




Probe

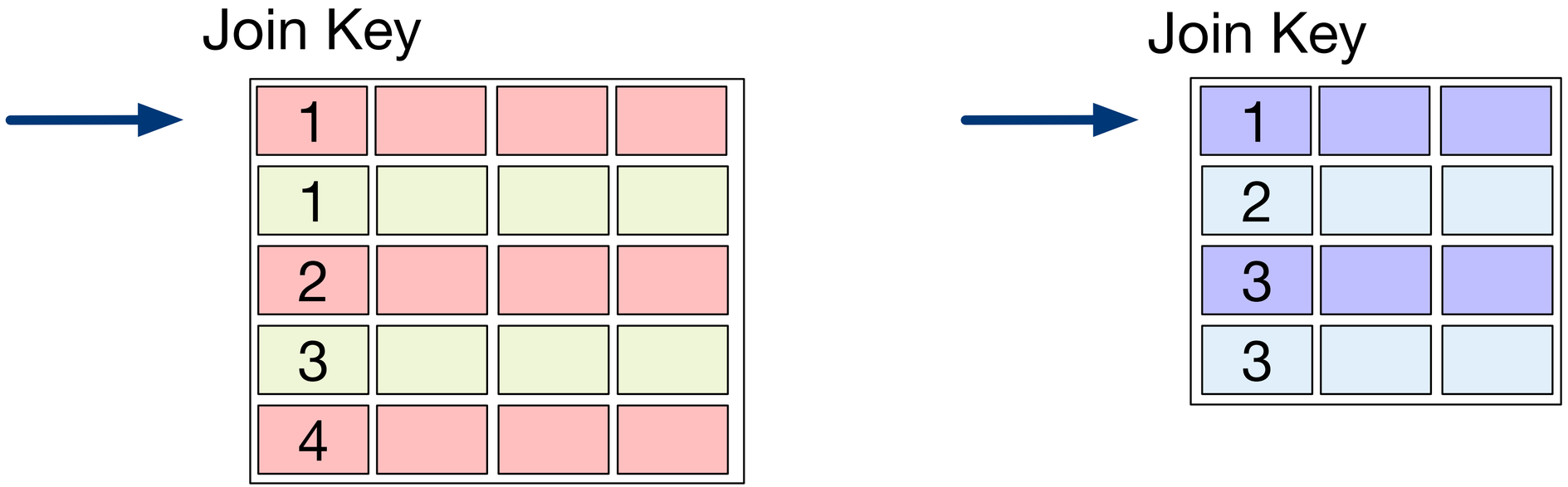
Join Key




Build

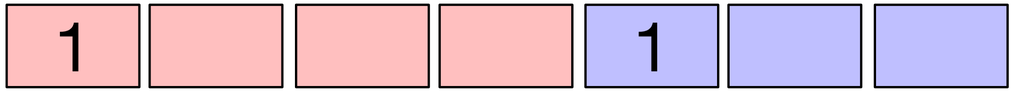
# Merge Join

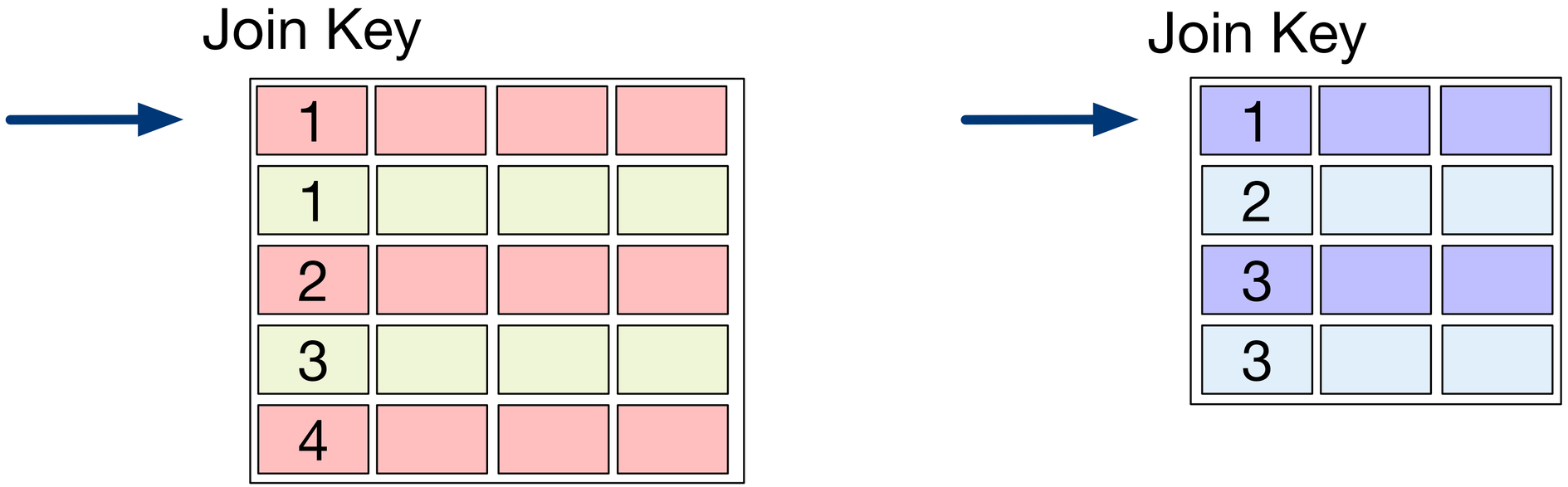




# Merge Join

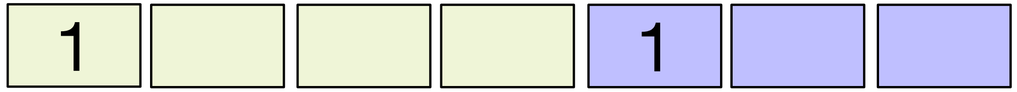


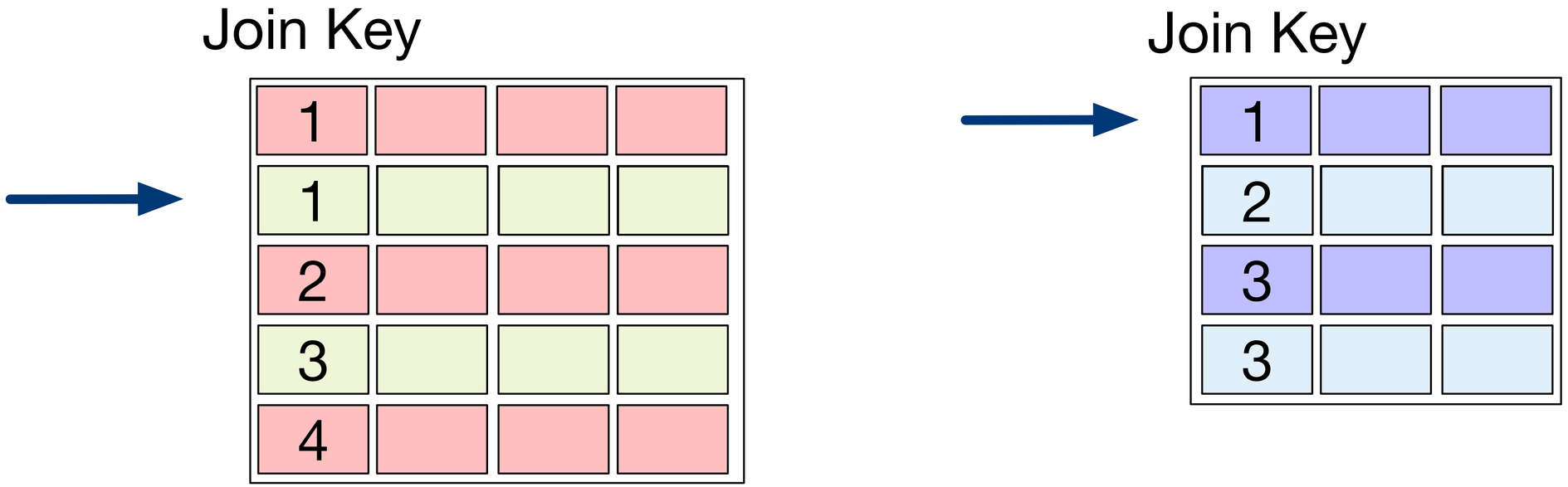


# Merge Join

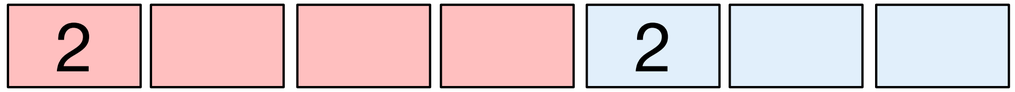


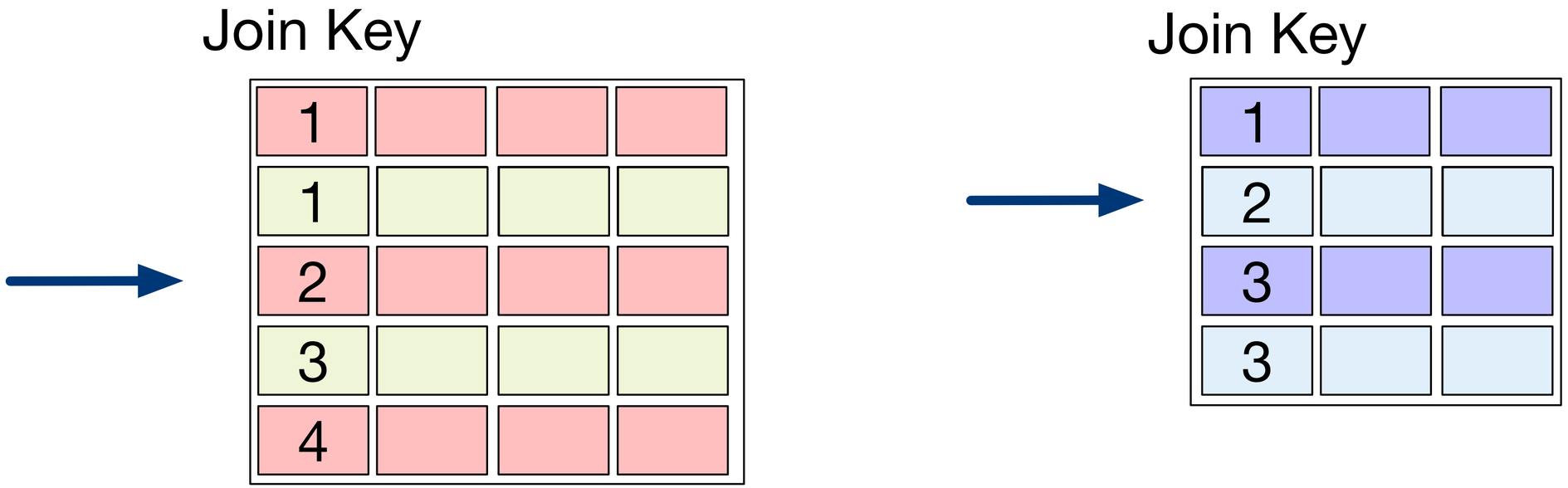


# Merge Join

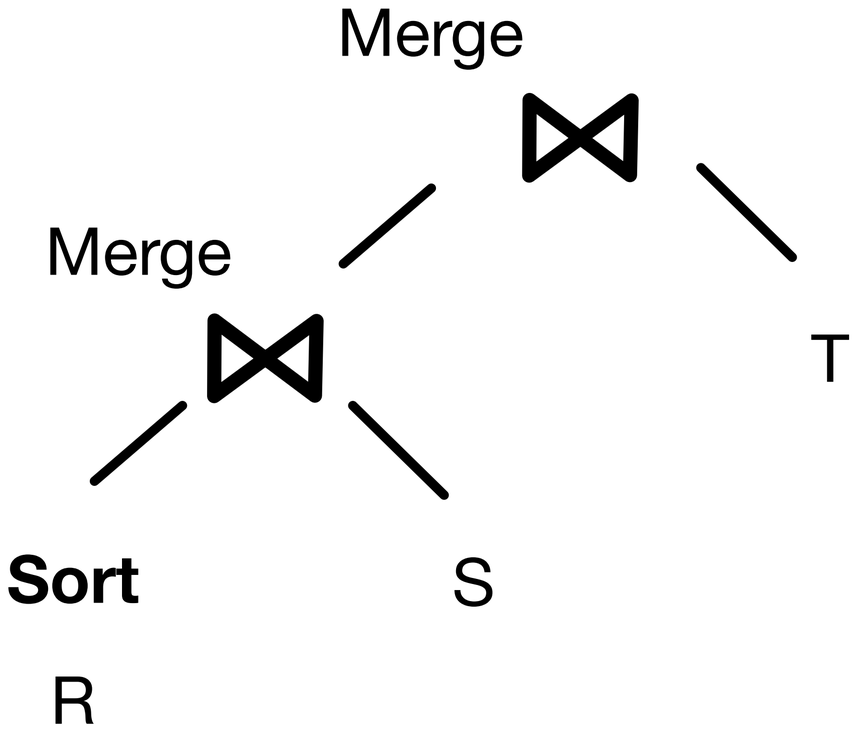


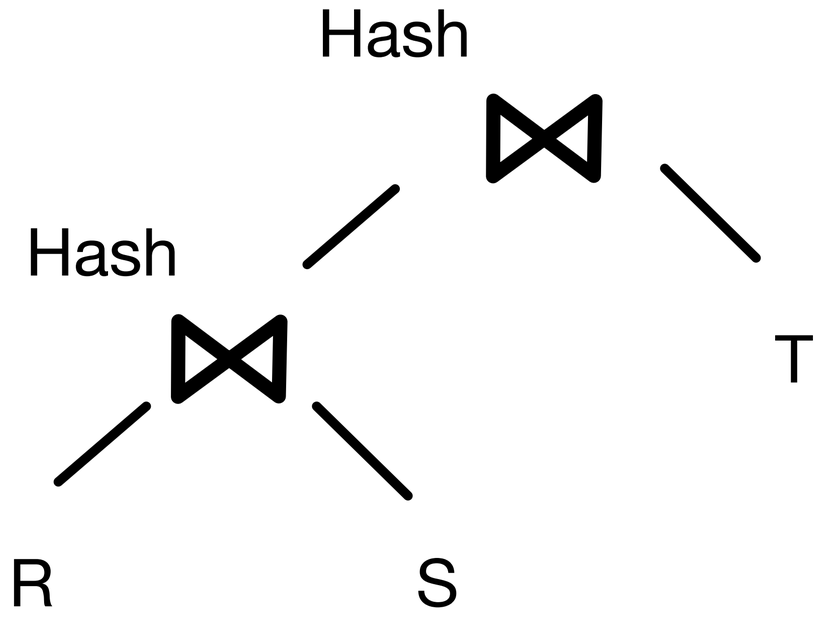


# Merge Join vs Hash Join

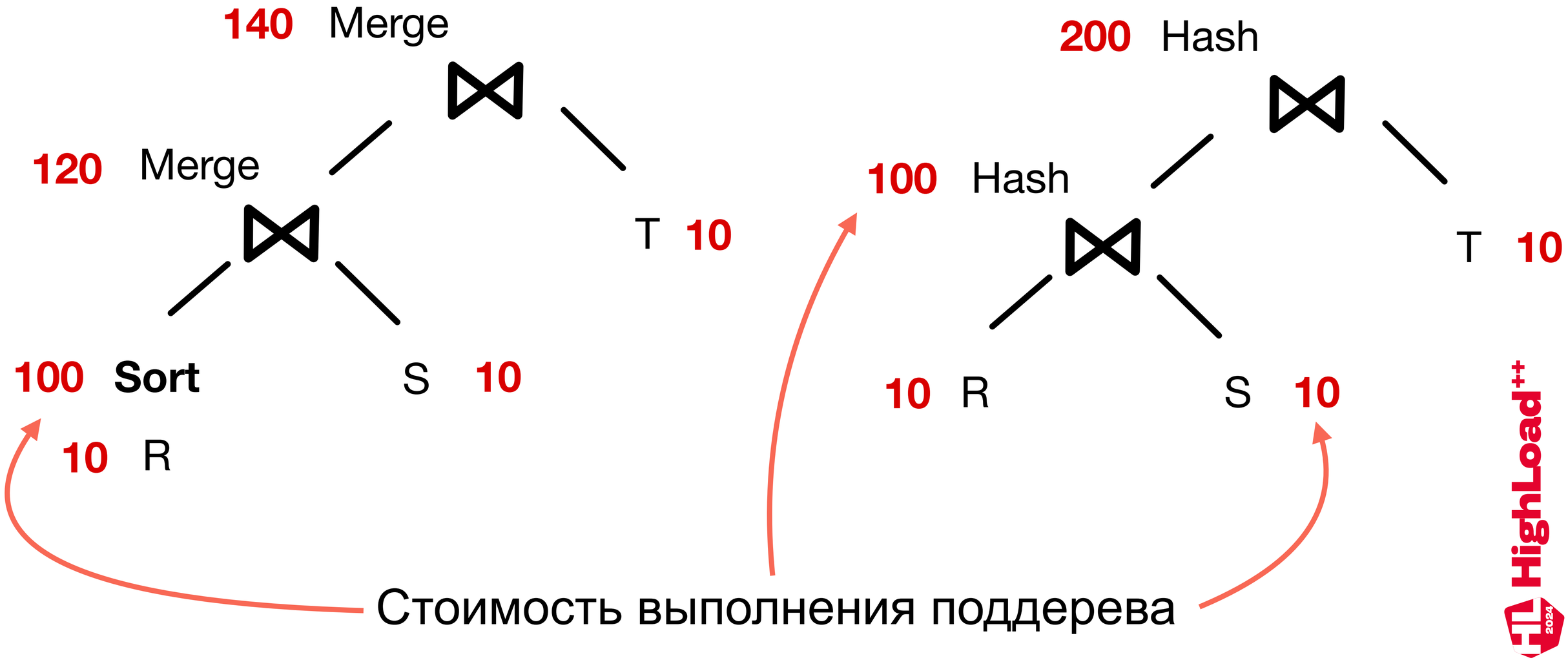




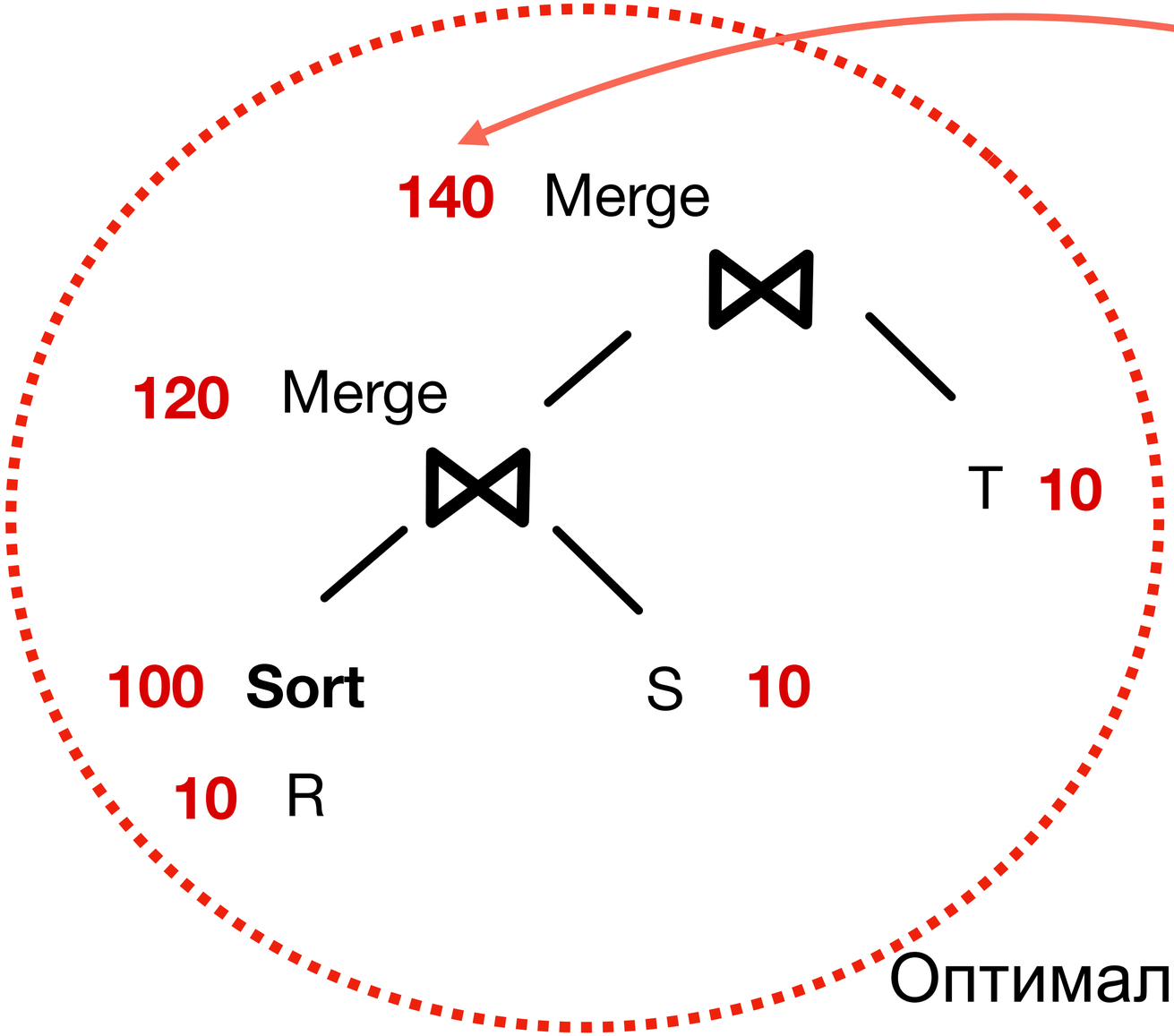
# Merge Join vs Hash Join

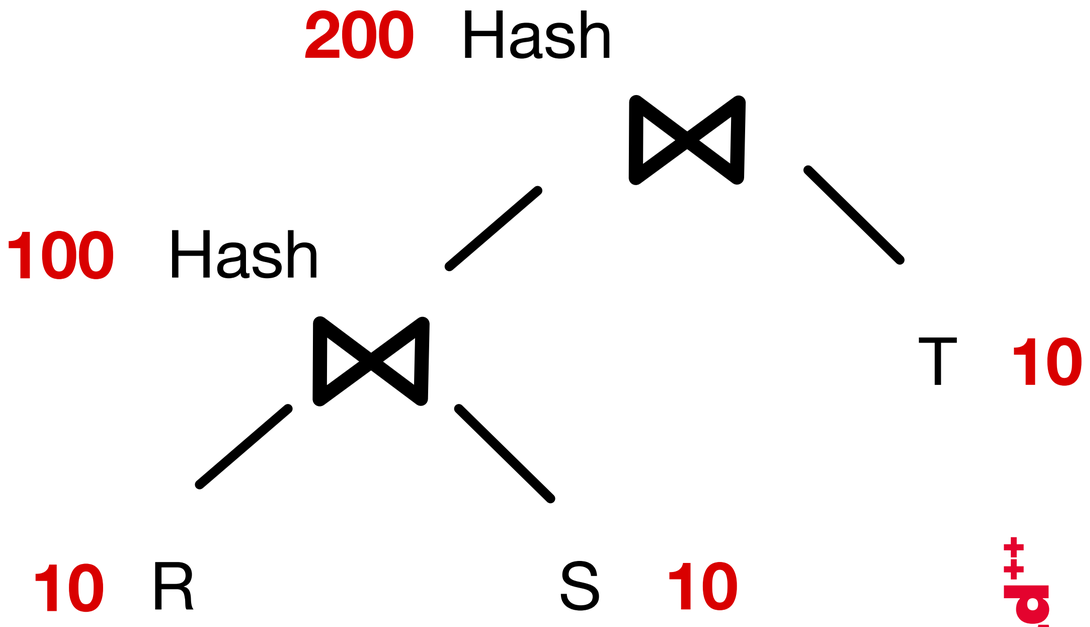




# Merge Join vs Hash Join



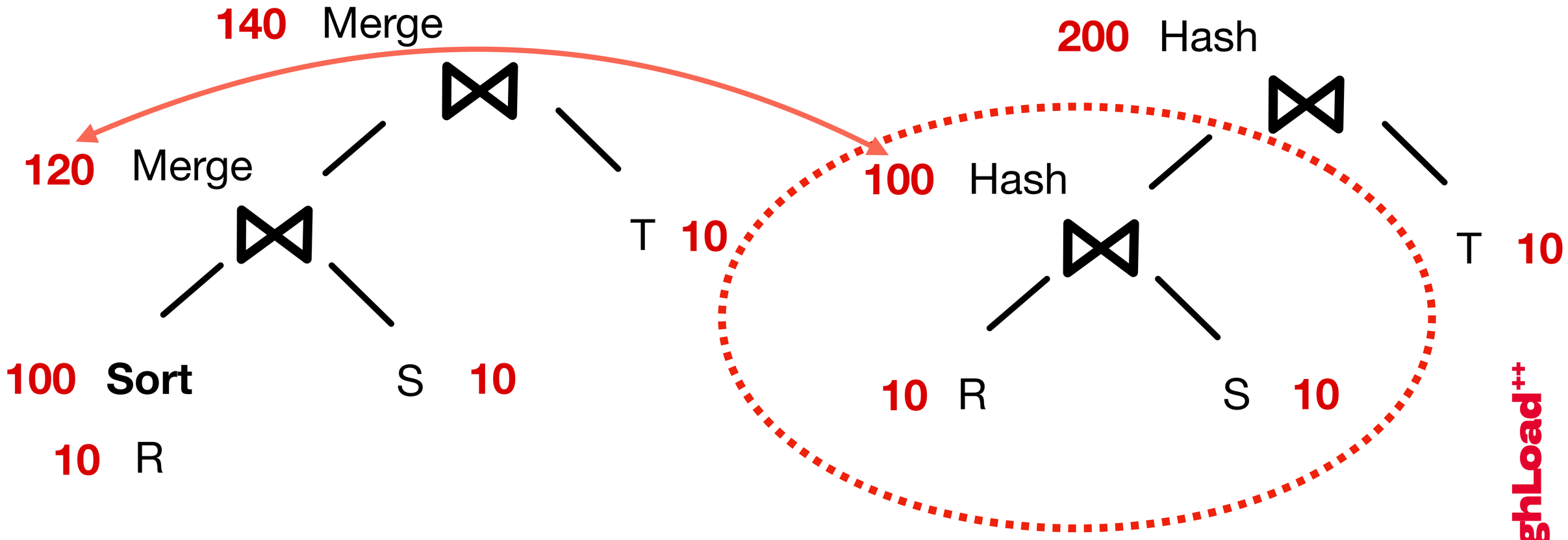



Оптимальный план для { R, S, T }

# Merge Join vs Hash Join

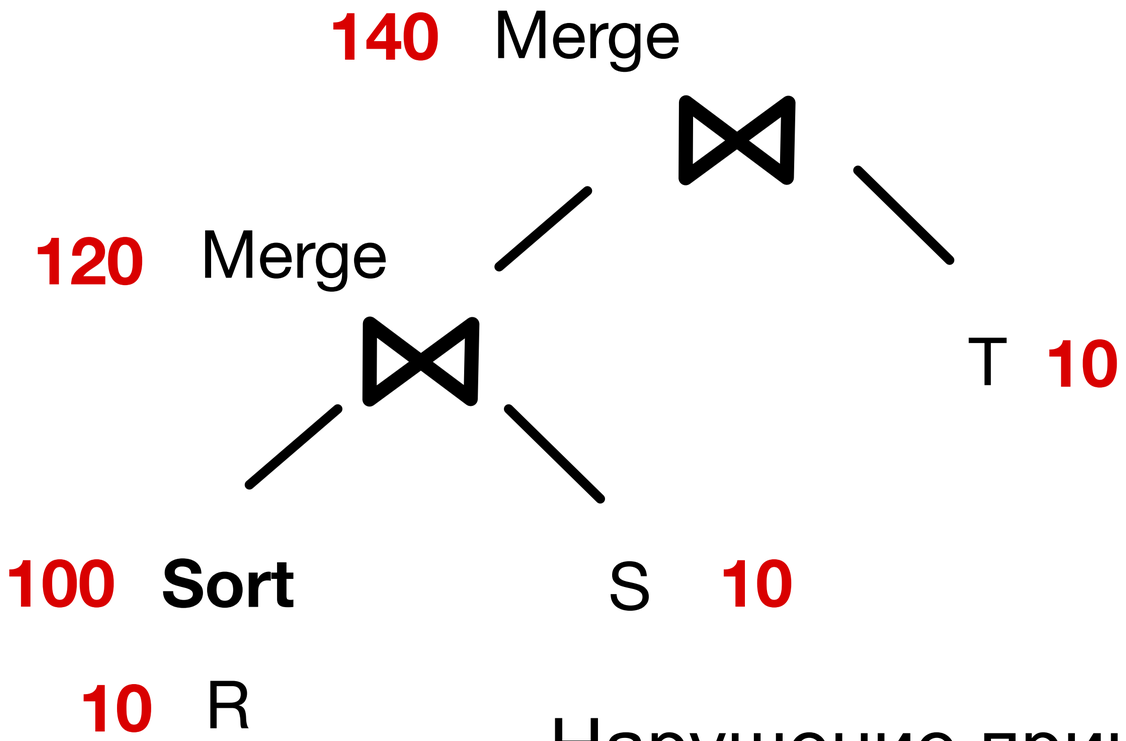


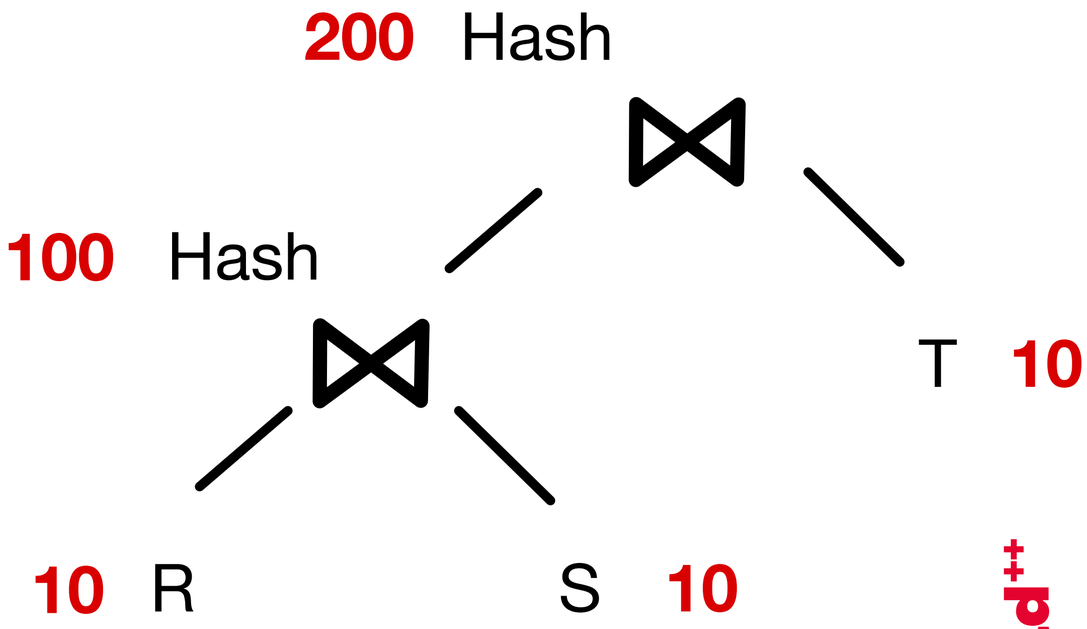



Оптимальный план для { R, S }

# Merge Join vs Hash Join






Нарушение принципа оптимальности:

оптимальный план включает в себя

неоптимальный подплан!

# Расходимся?

# Расходимся?



- Не все так плохо

# Расходимся?



- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности

# Расходимся?



- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности
  - Интересные сортировки

# Расходимся?



- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности
  - Интересные сортировки
  - Интересные ключи шардирования

# Расходимся?



- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности
  - Интересные сортировки
  - Интересные ключи шардирования
  - Одна группировка (Aggregate или Distinct)

# Расходимся?



- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности
  - Интересные сортировки
  - Интересные ключи шардирования
  - Одна группировка (Aggregate или Distinct)
- В таблице планов храним несколько вариантов

# Расходимся?

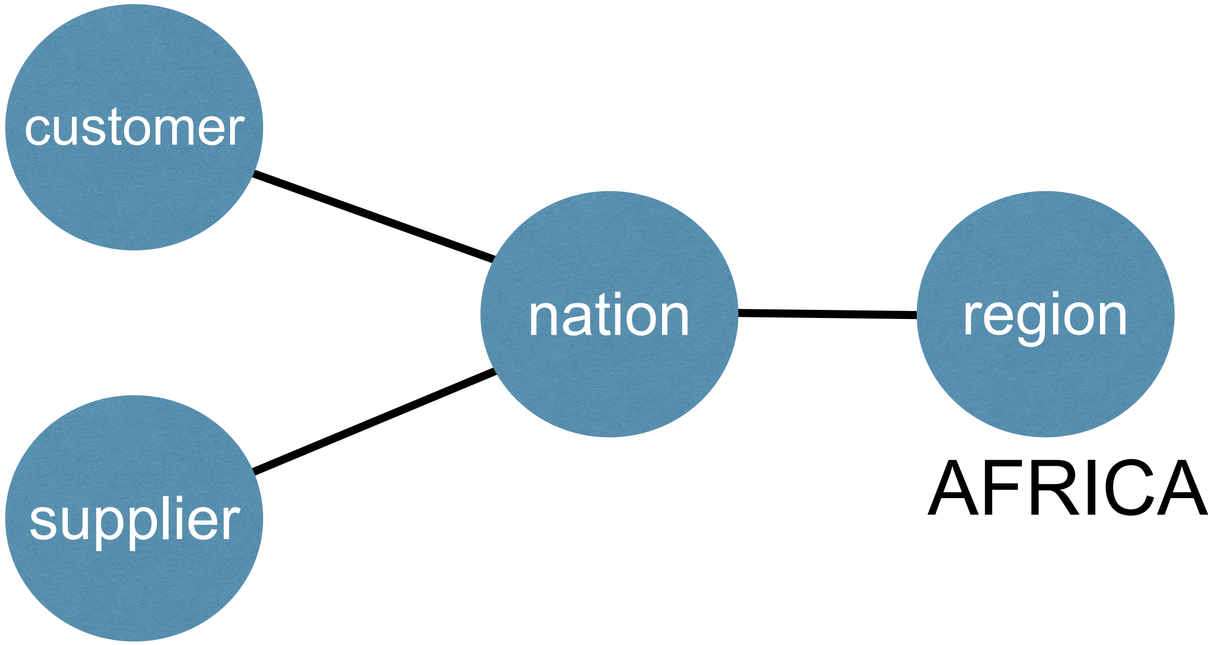


- Не все так плохо
- У нас ограниченное кол-во полезных физических свойств, нарушающих принцип оптимальности
  - Интересные сортировки
  - Интересные ключи шардирования
  - Одна группировка (Aggregate или Distinct)
- В таблице планов храним несколько вариантов
- Обычно приходится добавлять эвристики, чтобы сократить это число

# Fun fact — когда все наоборот



- Обычно в оптимизации мы пытаемся убрать лишние операции, тем более джоины
- Иногда их надо добавлять!



# Энумератор планов
# Оценка кардинальности

# Оценка кардинальности

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально
- После фильтра — сложнее

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально
- После фильтра — сложнее
- После одного джоина — еще сложнее

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально
- После фильтра — сложнее
- После одного джоина — еще сложнее
- После нескольких джоинов — все очень плохо

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально
- После фильтра — сложнее
- После одного джоина — еще сложнее
- После нескольких джоинов — все очень плохо
- Ошибка накапливается экспоненциально, после 6-7 джоинов ошибка в несколько порядков

# Оценка кардинальности



- Очень важная и самая проблемная часть оптимизатора
- Оценить кол-во записей после чтения — тривиально
- После фильтра — сложнее
- После одного джоина — еще сложнее
- После нескольких джоинов — все очень плохо
- Ошибка накапливается экспоненциально, после 6-7 джоинов ошибка в несколько порядков
- Можно потратить годы и много денег на эту задачу

# Наш подход

# Наш подход



- Начали с примитивной статистики по всей таблице

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов
  - WHERE city = “Moscow”

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов
  - WHERE city = “Moscow”
- Используем метаданные ключей

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов
  - WHERE city = “Moscow”
- Используем метаданные ключей
  - Если джоин по PK — то или будет меньше записей, или столько же

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов
  - WHERE city = “Moscow”
- Используем метаданные ключей
  - Если джоин по PK — то или будет меньше записей, или столько же
- Неплохо себя показывает на бенчмарках, например, TPCH

# Наш подход



- Начали с примитивной статистики по всей таблице
- Добавили пока только колоночную статистику для точечных запросов
  - WHERE city = “Moscow”
- Используем метаданные ключей
  - Если джоин по PK — то или будет меньше записей, или столько же
- Неплохо себя показывает на бенчмарках, например, TPCH
- Fun fact: для TPCH не нужна колоночная статистика

# Еще один фан факт

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]
  - Оператор сверху требует сортировку [b,c,d,e]

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]
  - Оператор сверху требует сортировку [b,c,d,e]
  - Вопрос: совместимы ли сортировки

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]
  - Оператор сверху требует сортировку [b,c,d,e]
  - Вопрос: совместимы ли сортировки
  - Выглядит тривиально, где сложность?

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]
  - Оператор сверху требует сортировку [b,c,d,e]
  - Вопрос: совместимы ли сортировки
  - Выглядит тривиально, где сложность?
  - Функциональные зависимости, порождаемые операторами:

# Еще один фан факт



- Все задачи в оптимизаторе, где надо оценить совместимость списков атрибутов — NP-Hard!
- Например
  - Текущий оператор гарантирует сортировку [a,b,c]
  - Оператор сверху требует сортировку [b,c,d,e]
  - Вопрос: совместимы ли сортировки
  - Выглядит тривиально, где сложность?
  - Функциональные зависимости, порождаемые операторами:
    - x = 15, y = sin(x), x = y, [a,b,c] -> d

# Энумератор планов
# Оценка кардинальности
# Оценочная функция

# Оценочная функция

# Оценочная функция



- Именно для оптимизации запросов точность не так важна

# Оценочная функция



- Именно для оптимизации запросов точность не так важна
- Почему?

# Оценочная функция



- Именно для оптимизации запросов точность не так важна
- Почему?
- Точные предсказания нужны для Workload Manager

# Оценочная функция



- Именно для оптимизации запросов точность не так важна
- Почему?
- Точные предсказания нужны для Workload Manager
- Пока обошлись простыми функциями, параметры нашли линейной регрессией на простом бенчмарке

Энумератор планов
Оценка кардинальности
Оценочная функция
# Сбор стаститики для оптимизатора

# Сбор статистики

# Сбор статистики



- Сбор колоночной статистики в распределенной системе — не самая простая задача

# Сбор статистики



- Сбор колоночной статистики в распределенной системе — не самая простая задача
- Пока сделали только count-min sketch

# Сбор статистики



- Сбор колоночной статистики в распределенной системе — не самая простая задача
- Пока сделали только count-min sketch
- Собираем в фоне, автоматически

# Сбор статистики



- Сбор колоночной статистики в распределенной системе — не самая простая задача
- Пока сделали только count-min sketch
- Собираем в фоне, автоматически
- Если срочно нужна точная статистика — команда ANALYZE

Энумератор планов
Оценка кардинальности
Оценочная функция
Сбор стаститики для оптимизатора

# Как оценивать оптимизатор?

# Как оценивать оптимизатор?

# Как оценивать оптимизатор?



- Главное, чтобы клиенты были счастливы

# Как оценивать оптимизатор?



- Главное, чтобы клиенты были счастливы
- Что если клиентов сильно меньше, чем у Amazon Redshift?

# Как оценивать оптимизатор?



- Главное, чтобы клиенты были счастливы
- Что если клиентов сильно меньше, чем у Amazon Redshift?
- Бенчмарки

# Как оценивать оптимизатор?



- Главное, чтобы клиенты были счастливы
- Что если клиентов сильно меньше, чем у Amazon Redshift?
- Бенчмарки
  - А есть ли хорошие?

# Бенчмарки по аналитике



| | Объем | Кол-во запросов | Сложность запросов | Сложность данных |
|---|---|---|---|---|

# Бенчмарки по аналитике



| | Объем | Кол-во запросов | Сложность запросов | Сложность данных |
|---|---|---|---|---|
| **TPCH** | 1GB - 100TB | 22 | Низкая | Очень низкая |

# Бенчмарки по аналитике



| | Объем | Кол-во запросов | Сложность запросов | Сложность данных |
|---|---|---|---|---|
| **TPCH** | 1GB - 100TB | 22 | Низкая | Очень низкая |
| **TPCDS** | 1GB - 100TB | 100 | Средняя | Низкая |

# Бенчмарки по аналитике



| | Объем | Кол-во запросов | Сложность запросов | Сложность данных |
|---|---|---|---|---|
| **TPCH** | 1GB - 100TB | 22 | Низкая | Очень низкая |
| **TPCDS** | 1GB - 100TB | 100 | Средняя | Низкая |
| **IMDB** | 2 GB | 100 | Высокая | Нормальная |

# Что делать с бенчмарками

# Что делать с бенчмарками



- Что на них сравнивать?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?
  - Насколько хуже чем оптимальные планы?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?
  - Насколько хуже чем оптимальные планы?
  - С конкурентами?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?
  - Насколько хуже чем оптимальные планы?
  - С конкурентами?
    - Потраченные ресурсы на выполнение?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?
  - Насколько хуже чем оптимальные планы?
  - С конкурентами?
    - Потраченные ресурсы на выполнение?
    - Качество планов?

# Что делать с бенчмарками



- Что на них сравнивать?
  - С оптимизатором / без оптимизатора?
  - Насколько хуже чем оптимальные планы?
  - С конкурентами?
    - Потраченные ресурсы на выполнение?
    - Качество планов?
    - Что делать с динамическими оптимизациями?

# Наш план

# Наш план



- Провести честные сравнения с open source конкурентами

# Наш план



- Провести честные сравнения с open source конкурентами
  - GreenplumDB

# Наш план



- Провести честные сравнения с open source конкурентами
  - GreenplumDB
  - Trino

# Наш план



- Провести честные сравнения с open source конкурентами
  - GreenplumDB
  - Trino
  - Может быть, Starrocks

# Наш план



- Провести честные сравнения с open source конкурентами
  - GreenplumDB
  - Trino
  - Может быть, Starrocks
- На бенчмарках TPCH, TPCDS

# Наш план



- Провести честные сравнения с open source конкурентами
  - GreenplumDB
  - Trino
  - Может быть, Starrocks
- На бенчмарках TPCH, TPCDS
- Самое сложное — честно настроить конкурентов

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |
| Amazon Redshift | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |
| Amazon Redshift | | | |
| Oracle | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |
| Amazon Redshift | | | |
| Oracle | | | |
| Snowflake | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |
| Amazon Redshift | | | |
| Oracle | | | |
| Snowflake | | | |
| Greenplum DB | | | |

# Функциональное сравнение с конкурентами



| СУБД | Много джоинов | Качество планов | Общее качество оптимизатора |
|---|---|---|---|
| Teradata | | | |
| Vertica | | | |
| Amazon Redshift | | | |
| Oracle | | | |
| Snowflake | | | |
| Greenplum DB | | | |
| Trino | | | |

# Куда ушло время в разработке



- Сам стоимостный оптимизатор — 30%
- Сбор статистики — 30%
- Интеграция — 30%
- Другое — 10%
- Отдельно — perf-тестирование

# Заключение



- Строим state-of-the-art оптимизатор по кусочкам
- Строить с нуля приятно и современно
  - Есть возможность пересмотреть все старые “истины”
- Open source
- По некоторым свойствам уже получилось обойти конкурентов
  - Кол-во джоинов, реордеринг сложных видов джоинов
- Дальнейшие планы еще амбициознее

# Голосуйте за мой доклад ➔

ydb.tech/

pavelvelikhov@yandex-team.ru

www.linkedin.com/in/velikhov/

HL 2024 HighLoad++