Новые фичи в MariaDB и в MySQL

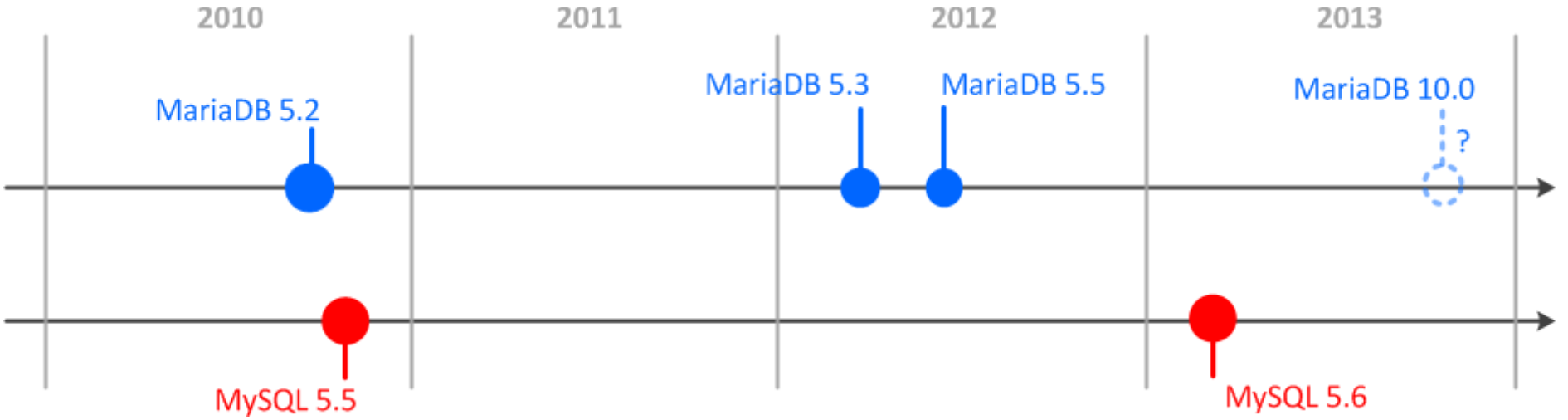
что общего и в чем разница

DevConf 2013, Сергей Петруня, Monty Program Ab

Что такое MariaDB

- Это ветка (branch) субд MySQL
- Лицензия - GPL
- Полностью совместима с «основным» MySQL
  - форматы файлов
  - соединения master<->slave*
  - имена бинарников, init-скрипты и тд.
- Отличия от MySQL
  - Собственные фичи
  - Percona's XtraDB (vanilla InnoDB то же есть, как plugin)
  - Интегрированы разные патчи от Percona, Sphinx, и других авторов

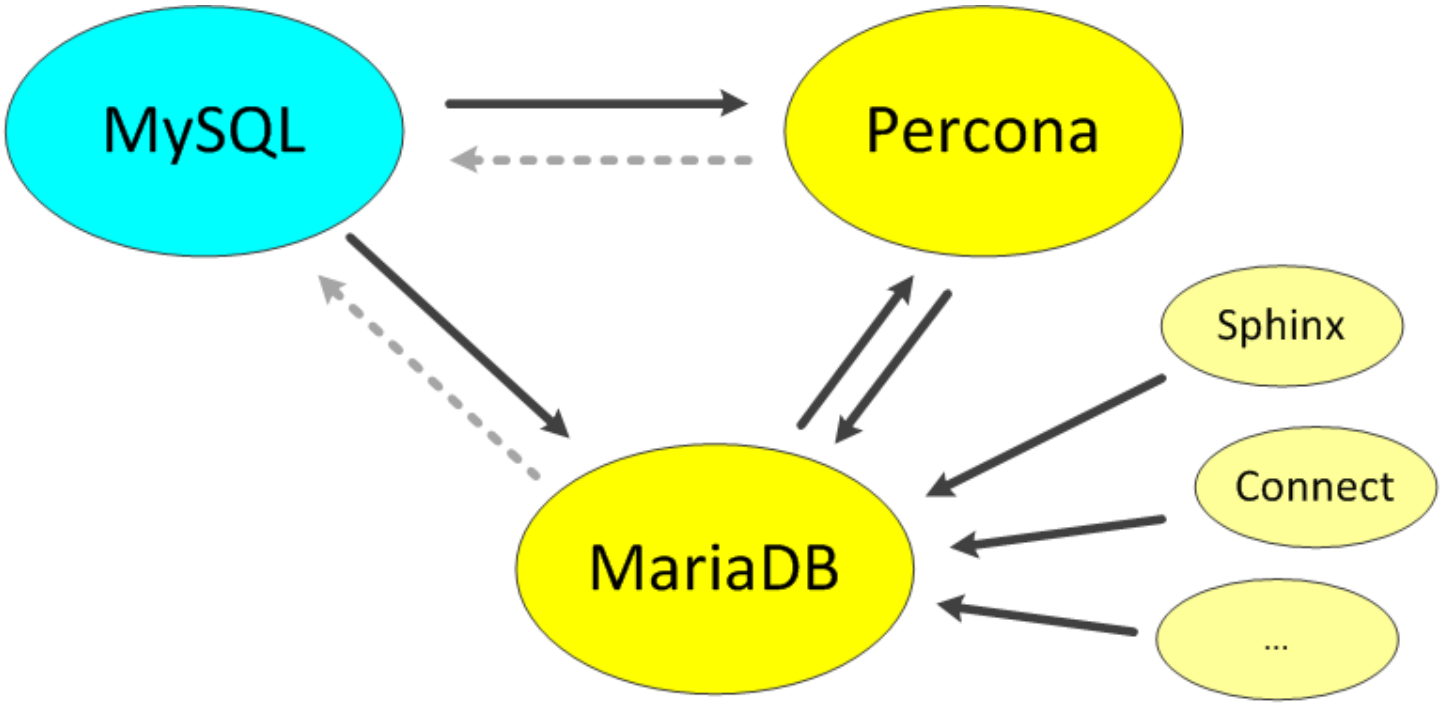
История релизов MariaDB и MySQL



- 2012 – MariaDB 5.5
  - это MySQL 5.5 + еще фичи
- 2013 – MySQL 5.6
- 2013? – MariaDB 10.0

  = важное из MySQL 5.6 + фичи + альтернативные реализации

Процесс разработки



GNU GPL

Dual licencing

Включение кода

Заимствование идеи. Или случайное совпадение ☺

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

InnoDB в MySQL 5.6

- Fulltext index
- Производительность
  - Масштабируемость на многоядерных системах (16+ ядер)
  - Более равномерный сброс данных на диск (adaptive flushing)
  - Настройки сброса для SSD/HDD
  - Read-only транзакции
- Высокая доступность
  - Online ALTER TABLE для многих операций
  - Transportable tablespaces
  - Сохранение/загрузка содержимого Buffer Pool
    - как в XtraDB (в Percona Server 5.5, MariaDB 5.5)

InnoDB в MySQL 5.6 (2)

- Сжатие данных
  - Настройка компрессии, уровень 1...9
  - Обработка провалов компрессии
  - Диагностика компрессии для каждого индекса
- Redo log > 4G
- и т. д..

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

Репликация : MariaDB 5.3/5.5

- Group Commit (→ Percona Server)
  - Писать в binlog и InnoDB надо синхронно
  - Писать надо группами
- Контрольные суммы событий в binlog
  - бекпорт из MySQL 5.6
- RBR: текст исходного запроса в binlog
- RBR: оптимизация работы с таблицами без PK
  - использование индексов

Репликация: MySQL 5.6

- Group commit
  - Похож на MariaDB
  - из-за разных InnoDB пока не сравнить
- RBR: Batch Applier для таблиц без PK
- Контрольные суммы событий в binlog
- RBR: текст исходного запроса в binlog
- **Global Transaction IDs**
- **Параллельный slave**
- RBR: partial row images
- Slave crash safety
- Утилиты для администрирования
  - mysqlrpl*

Репликация: MySQL 5.6: Параллельный slave

- Базовое предположение

  Изменения в разных БД (schema) можно применять в любом порядке

- Линейное ускорение, если транзакции размазаны по разным БД.
- MariaDB: альтернативная идея
  - Group commit помечает, какие он делал группы
  - Транзакции-члены группы были совместно “готовы к коммиту”
    - Конфликтов нет
  - Slave может применять членов группы параллельно

Репликация: MySQL 5.6: GTID

- Вместо binlog position – set<server-uuid:trx-no>
- Цели
  - Простое администрирование

```
CHANGE MASTER TO MASTER_AUTO_POSITION=1
```

  - Простая смена мастера
  - master->slave promotion
  - ...
- Проблемы
  - Репликация между GTID и не-GTID серверами невозможна
  - В случае ошибки починить будет нелегко
  - ...
- MariaDB: будет другая реализация

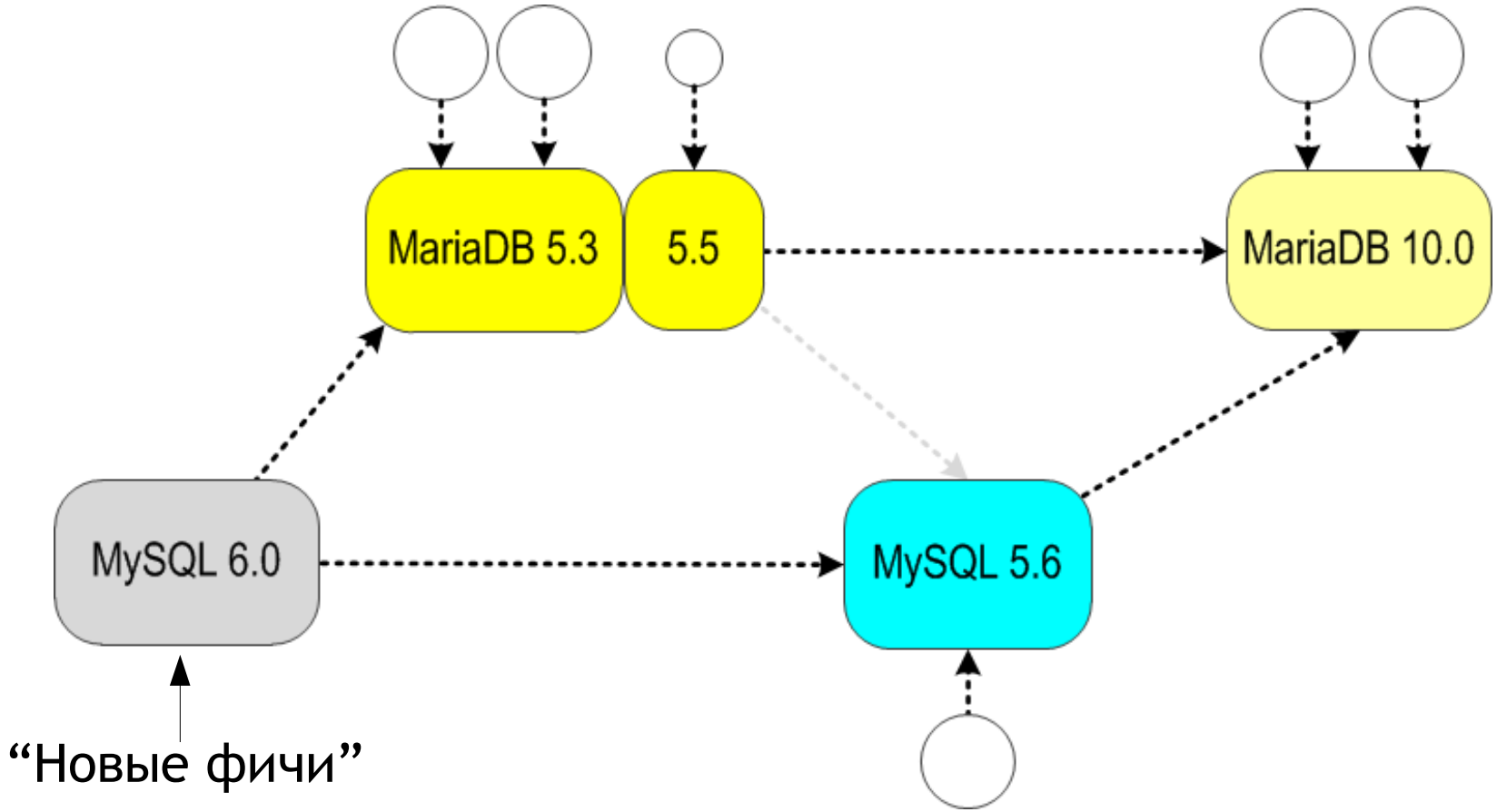
Репликация в MariaDB 10.0

- Своя реализация GTID
  - Вместо set<server-uuid:trx-no>
    set<domain:server-id:trx-no>
  - Домен – последовательность транзакций, в которой должен сохраняется порядок
    - Между доменами порядок не сохраняется
  - Возможность репликации между GTID и не GTID
- Multi-Source
  - Slave с несколькими masters
  - Предполагается отсутствие конфликтов
  - На основе кода от TaoBao.

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

Оптимизатор - общие корни



Подзапросы

Новые фичи - подзапросы

- “Не используйте в MySQL подзапросы”
- В Facebook их запретили на уровне парсера
- Причина
  - Отсутствие оптимизатора
  - “Наивное” выполнение подзапросов

Наивное выполнение подзапросов (1)

- Подзапрос IN (SELECT ...)

```
select * from hotel
where
  hotel.country='USA' and
  hotel.name IN (select hotel_stays.hotel
                 from hotel_stays
                 where hotel_stays.customer='John Smith')
```

- Наивное выполнение:

```
for (each hotel in USA ) {
  if (John Smith stayed here) {
     …
  }
}
```

- Медленно!

Наивное выполнение подзапросов (2)

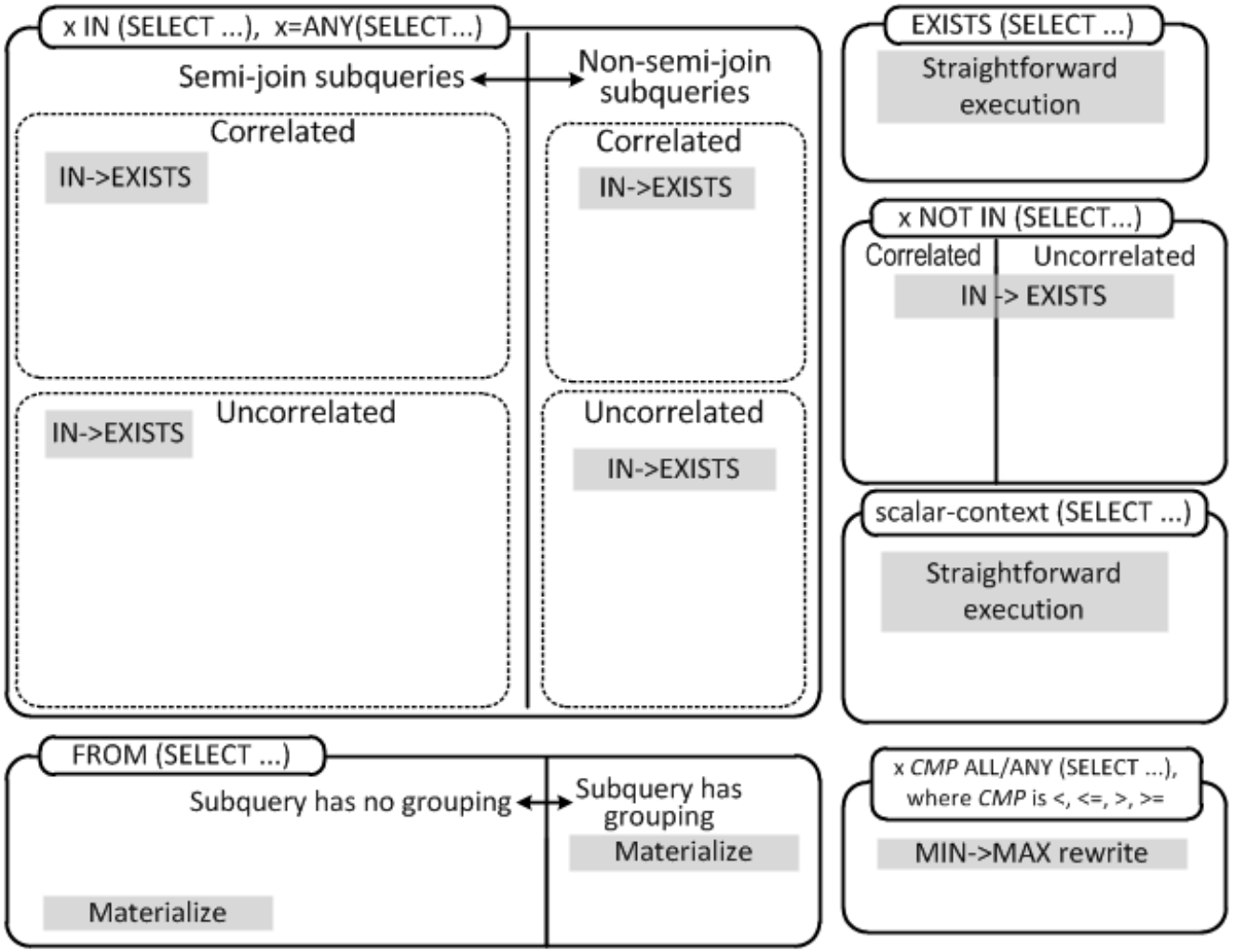
- Подзапрос FROM (SELECT ...)

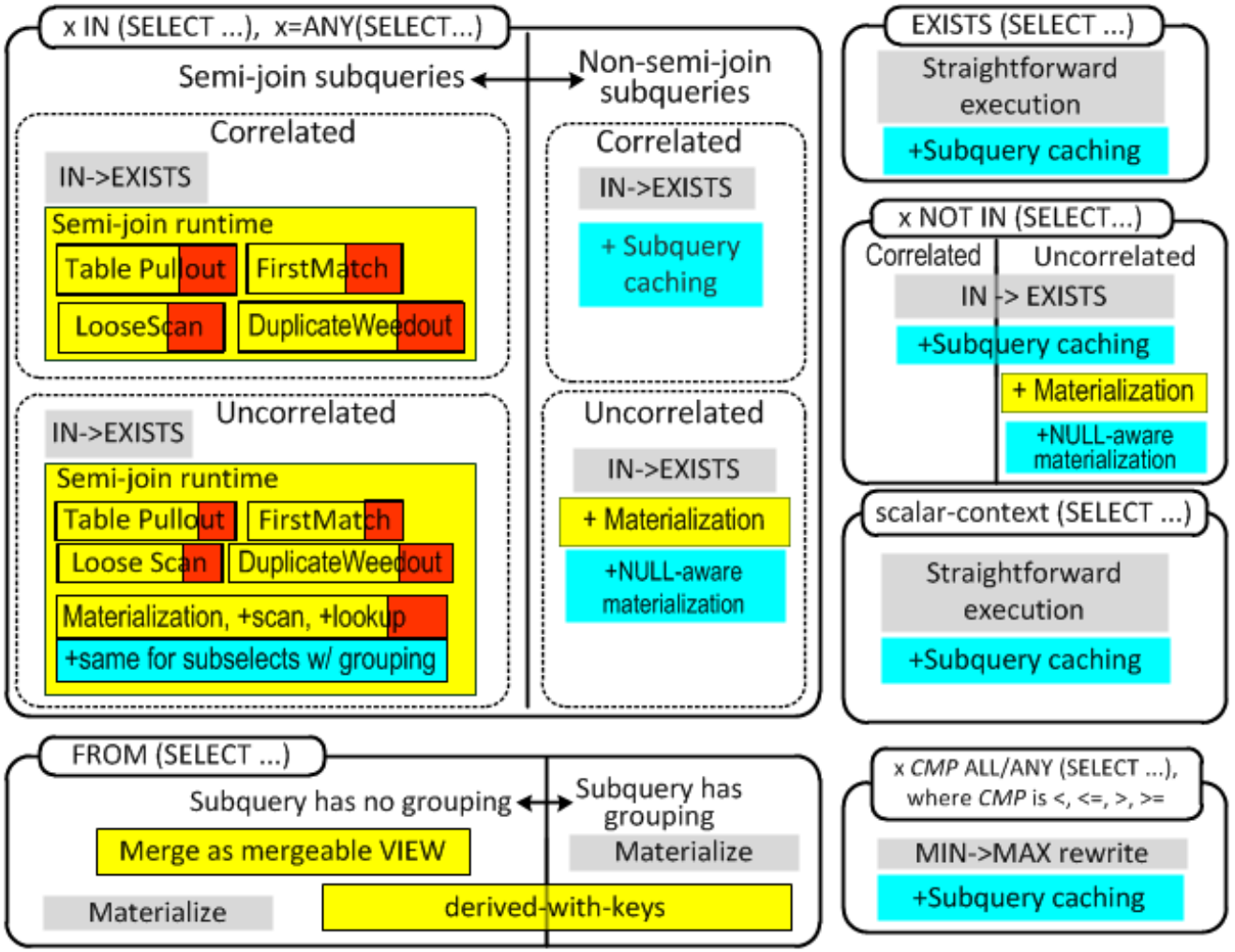
```
select *
from
   (select *
    from hotel
    where hotel.rooms > 500
   ) as big_hotel
where
    big_hotel.nearest_aiport='HEL'
```

- Наивное выполнение

1. Найти все отели, у которых **rooms>500**, записать во временную таблицу **big_hotel**
2. Просмотреть big_hotel и выбрать отели, для которых **nearest_aiport='HEL'**

- Медленно!


x IN (SELECT ...), x=ANY(SELECT...)
Semi-join subqueries
Non-semi-join subqueries
Correlated
IN->EXISTS
Correlated
IN->EXISTS
Uncorrelated
IN->EXISTS
Uncorrelated
IN->EXISTS
EXISTS (SELECT ...)
Straightforward execution
x NOT IN (SELECT...)
Correlated
Uncorrelated
IN -> EXISTS
scalar-context (SELECT ...)
Straightforward execution
FROM (SELECT ...)
Subquery has no grouping
Subquery has grouping
Materialize
Materialize
x CMP ALL/ANY (SELECT ...), where CMP is <, <=, >, >=
MIN->MAX rewrite
MySQL 5.1
MariaDB 5.5
MariaDB 5.5, MySQL 5.6
MySQL 5.6



MySQL 5.1 | MariaDB 5.5 | MariaDB 5.5, MySQL 5.6 | MySQL 5.6

Оптимизации для подзапросов

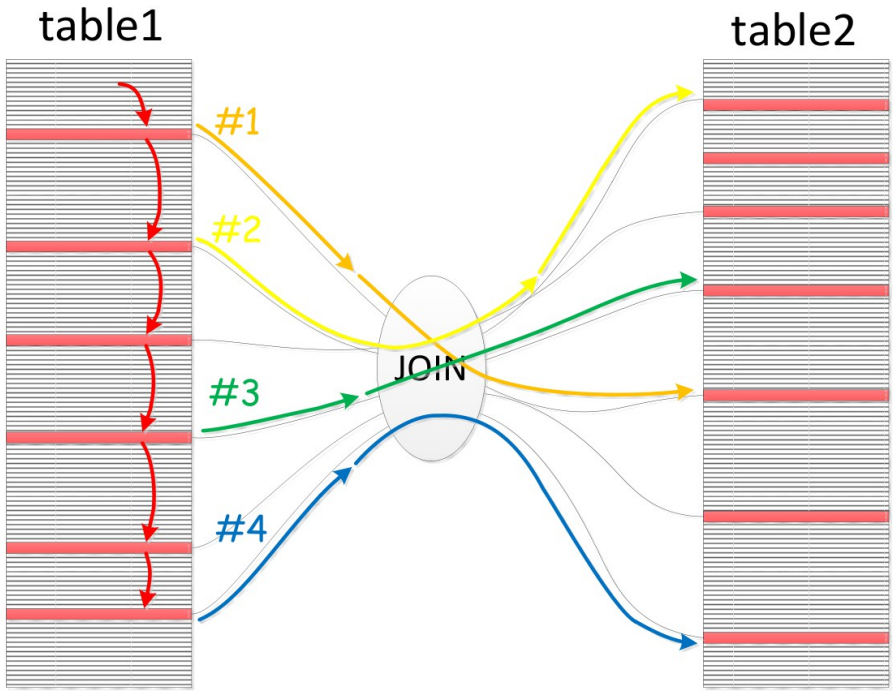
- Большая работа
- Оптимизации для
  - IN (SELECT…)
  - FROM (SELECT …)
  - И многих других случаев
- Сравнивали support cases с PostgreSQL
  - До: ~1000 раз медленнее
  - После: такой же порядок
- => Подзапросы работают как join'ы
- Релизы
  - MariaDB 5.3 (все)
  - MySQL 5.6 (подмножество)

Batched Key Access

Batched Key Access - зачем

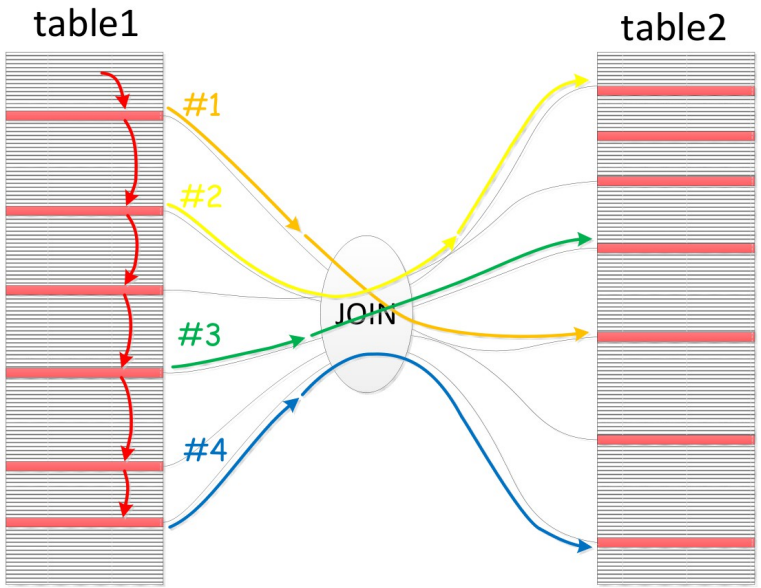
- Большие, аналитические join'ы работали медленно
  - DBT-3 scale=10, 30G data, аналитика
    есть запросы >24 часов

- Причина?
  - Nested Loops Join
  - Random disk IO

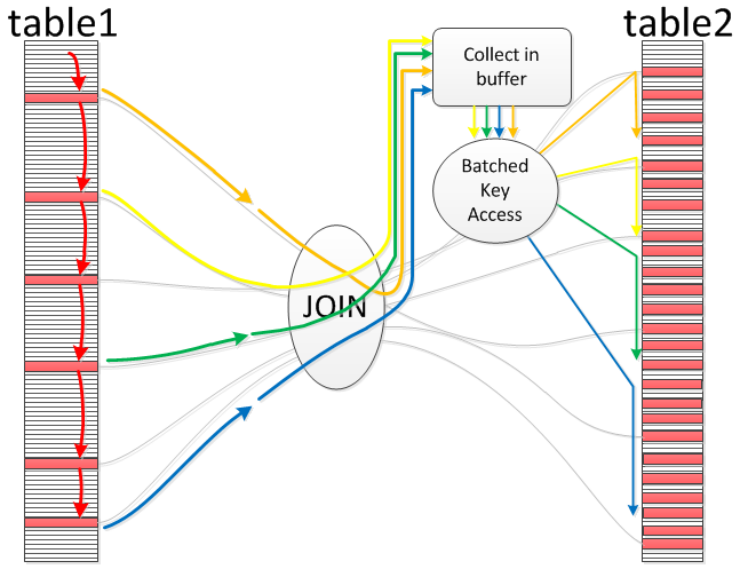


Batched Key Access - идея

Nested Loops Join



Batched Key Access



Причины ускорения

- Вращающийся диск – одно движение головки
- Дружественно к кешам
- Дружественно к prefetch

Batched Key Access benchmark

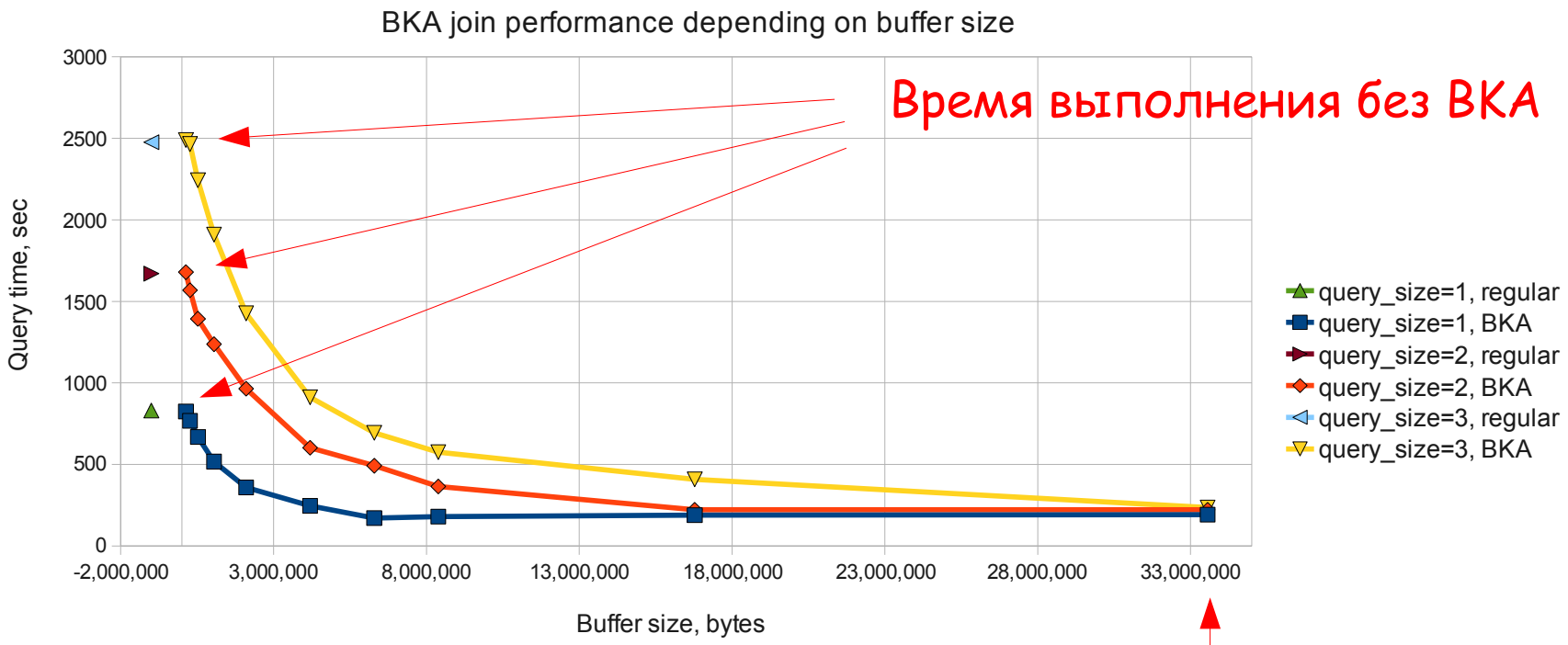
```
set join_cache_level=6; - enable BKA

select max(l_extendedprice)
  from orders, lineitem
where
  l_orderkey=o_orderkey and
  o_orderdate between $DATE1 and $DATE2
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	range	PRIMARY, i_o_orderdate	i_o_orderdate	4	NULL	142680	Using where; Using index
1	SIMPLE	lineitem	ref	PRIMARY	PRIMARY	4	orders. o_orderkey	2	Using join buffer (flat, BKA join); Key-ordered scan

Запускаем с

- Разными размерами **@@join_buffer_size**
- Разными размерами промежутка $DATE1...$DATE2

Batched Key Access benchmark (2)



Время выполнения без BKA

Если дать BKA достаточно большой буфер...

Результат

- 4 GB RAM
- DBT-3 benchmark, scale=10, InnoDB
  - 30GB data (InnoDB)
  - 20 “decision-support” аналитических запросов

	До	После
avg	**3.24 hrs**	**5.91 min**
median	**0.32 hrs**	**4.48 min**
max	**25.97 hrs***	**22.92 min**

* - надоело ждать

Batched Key Access - детали

- MariaDB 5.3
  - “Key-ordered scan” (см предыдущий пример)
  - “Rowid-ordered scan”
- MySQL 5.6
  - “Using MRR” - не работает для clustered PK
- How-to
  - Лучше использовать MariaDB :-)
  - Надо настроить параметры (kb.askmonty.org).

Index Condition Pushdown

Index Condition Pushdown

- Еще одна “общая” фича MariaDB 5.3 и MySQL 5.6

```
alter table lineitem add index s_r (l_shipdate, l_receiptdate);

select count(*) from lineitem
where
   l_shipdate between '1993-01-01' and '1993-02-01' and
   datediff(l_receiptdate,l_shipdate) > 25 and
   l_quantity > 40
```

```
+----+-------------+----------+-------+---------------+------+---------+------+--------+--------------------------------------+
| id | select_type | table    | type  | possible_keys | key  | key_len | ref  | rows   | Extra                                |
+----+-------------+----------+-------+---------------+------+---------+------+--------+--------------------------------------+
|  1 | SIMPLE      | lineitem | range | s_r           | s_r  | 4       | NULL | 158854 | Using index condition; Using where |
+----+-------------+----------+-------+---------------+------+---------+------+--------+--------------------------------------+
```

1. Читать индекс в диапазоне
   `l_shipdate between '1993-01-01' and '1993-02-01'`
2. Проверить условие на колонки из индекса
   `datediff(l_receiptdate,l_shipdate) > 25`
3. Читать полные записи
4. Проверить оставшуюся часть WHERE
   `l_quantity > 40`

← Новое!
← Уменьшает число читаемых записей

Index Condition Pushdown - итого

- MariaDB 5.3 → MySQL 5.6
- Работает для любого чтения через индекс (**ref, range, eq_ref**)
- Проверяет часть WHERE до чтения полной записи
  - Можно иметь “почти **Using index**”
- Ускорение
  - IO-bound – 5x-10x
  - CPU-bound: 2x.

Extended keys

Extended Keys (расширенные индексы?)

- У каждого индекса в InnoDB есть невидимый “хвост”

```
CREATE TABLE tbl (
  pk1  sometype,
  pk2  sometype,
  ...
  col1 sometype,
  col2 sometype,
  ...
  KEY indexA (col1, col2)
  ...
  PRIMARY KEY (pk1, pk2)
) ENGINE=InnoDB
```

- Раньше: бОльшая часть оптимизатора про “хвост” не знала
- MariaDB 5.5, повтор в MySQL 5.6:
  - Все части оптимизатора могут ипользовать “хвост”.

Улучшенный EXPLAIN

MySQL 5.6: улучшения в EXPLAIN

- EXPLAIN UPDATE/DELETE/INSERT … SELECT

```
mysql> explain update customer set c_acctbal = c_acctbal - 100 where c_custkey=12354;
+----+-------------+----------+-------+---------------+---------+---------+------+------+-------------+
| id | select_type | table    | type  | possible_keys | key     | key_len | ref  | rows | Extra       |
+----+-------------+----------+-------+---------------+---------+---------+------+------+-------------+
|  1 | SIMPLE      | customer | range | PRIMARY       | PRIMARY | 4       | NULL |    1 | Using where |
+----+-------------+----------+-------+---------------+---------+---------+------+------+-------------+
```

- EXPLAIN FORMAT=JSON
  - Выдает EXPLAIN в JSON
  - Показывает, куда прицеплены части WHERE
  - Показывает, зачем используются “Using temporary” и “Using filesort”
  - А так, тот же EXPLAIN, вид сбоку.

Что такое “части WHERE”

```
explain
select
  count(*)
from
  orders, customer
where
  customer.c_custkey=orders.o_custkey and
  customer.c_mktsegment='BUILDING' and
  orders.o_totalprice > customer.c_acctbal and
  orders.o_orderpriority='1-URGENT'
```

```
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
| id | select_type | table    | type | possible_keys | key         | key_len | ref                         | rows    | Extra       |
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
|  1 | SIMPLE      | customer | ALL  | PRIMARY       | NULL        | NULL    | NULL                        | 1509871 | Using where |
|  1 | SIMPLE      | orders   | ref  | i_o_custkey   | i_o_custkey | 5       | dbt3sf10.customer.c_custkey |       7 | Using where |
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
```

```
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
| id | select_type | table    | type | possible_keys | key         | key_len | ref                         | rows    | Extra       |
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
|  1 | SIMPLE      | customer | ALL  | PRIMARY       | NULL        | NULL    | NULL                        | 1509871 | Using where |
|  1 | SIMPLE      | orders   | ref  | i_o_custkey   | i_o_custkey | 5       | dbt3sf10.customer.c_custkey |       7 | Using where |
+----+-------------+----------+------+---------------+-------------+---------+-----------------------------+---------+-------------+
```

```
{
  "query_block": {
    "select_id": 1,
    "nested_loop": [
      {
        "table": {
          "table_name": "customer",
          "access_type": "ALL",
          "possible_keys": [
            "PRIMARY"
          ],
          "rows": 1509871,
          "filtered": 100,
          "attached_condition": "(`dbt3sf10`.`customer`.`c_mktsegment` = 'BUILDING')"
        }
      },
      {
        "table": {
          "table_name": "orders",
          "access_type": "ref",
          "possible_keys": [
            "i_o_custkey"
          ],
          "key": "i_o_custkey",
          "used_key_parts": [
            "o_custkey"
          ],
          "key_length": "5",
          "ref": [
            "dbt3sf10.customer.c_custkey"
          ],
          "rows": 7,
          "filtered": 100,
          "attached_condition": "((`dbt3sf10`.`orders`.`o_orderpriority` = '1-URGENT') and
(`dbt3sf10`.`orders`.`o_totalprice` > `dbt3sf10`.`customer`.`c_acctbal`))"
        }
      }
    ]
  }
}
```

Статистика

Статистика по индексам

```
select
  count(*)
from
  customer, orders
where
  customer.c_custkey=orders.o_custkey and customer.c_mktsegment='BUILDING';
```

```
+------+-------------+----------+------+---------------+-------------+---------+--------------------+--------+-------------+
| id   | select_type | table    | type | possible_keys | key         | key_len | ref                | rows   | Extra       |
+------+-------------+----------+------+---------------+-------------+---------+--------------------+--------+-------------+
|    1 | SIMPLE      | customer | ALL  | PRIMARY       | NULL        | NULL    | NULL               | 148305 | Using where |
|    1 | SIMPLE      | orders   | ref  | i_o_custkey   | i_o_custkey | 5       | customer.c_custkey |      7 | Using index |
+------+-------------+----------+------+---------------+-------------+---------+--------------------+--------+-------------+
```

?

```
MariaDB > show table status like 'orders'\G
*************************** 1. row ***************************
           Name: orders
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 1495152
           .............
```

```
MariaDB > show keys from orders where key_name='i_o_custkey'\G
*************************** 1. row ***************************
        Table: orders
   Non_unique: 1
     Key_name: i_o_custkey
  Seq_in_index: 1
  Column_name: o_custkey
    Collation: A
  Cardinality: 212941
     Sub_part: NULL
     ................
```

1495152 / 212941 = 7

7 заказов/ заказчик

Проблемы со статистикой

MySQL 5.5, InnoDB

- Статистика вычисляется на лету
  - Когда отркрывается таблица (перезапуск сервера, ALTER TABLE)
  - После изменения n% записей
  - …
- Статистика вычисляется методом случайных проб
  - @@innodb_stats_sample_pages

- Результат
  - Статистика меняется без предупреждения
    => А с ней и планы запросов тоже
- DBT-3 benchmark
  - 22 аналитических запросов
  - Планов на запрос: avg=2.8, max=7.c

Решение проблем со статистикой

Persistent statistics v1

- Percona Server 5.5 (спортировано в MariaDB 5.5)
  - Надо включить: **innodb_use_sys_stats_table=1**
- Статистика хранится внутри InnoDB
  - Можно посмотреть (но не менять) через information_schema.innodb_sys_stats
- Можно отключить авто-перевычисление: **innodb_stats_auto_update=OFF**

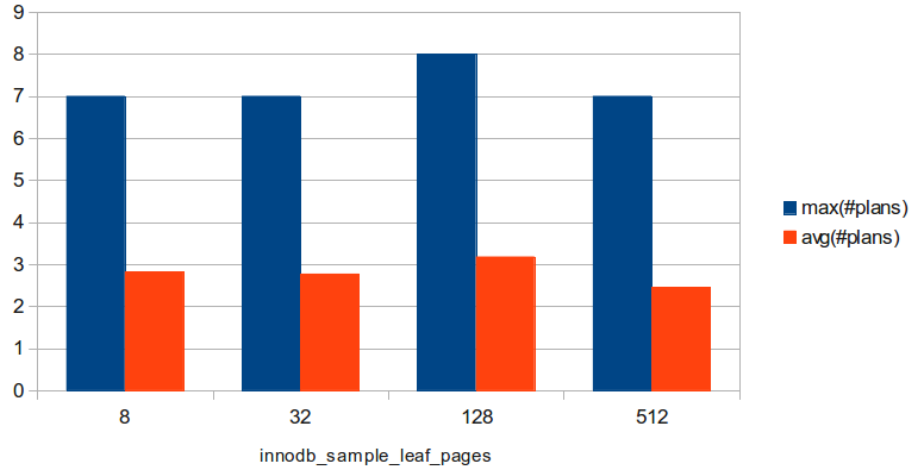
Persistent statistics v2

- MySQL 5.6
  - Включена по умолчанию (innodb_stats_persistent=1)
- Хранится в обычных таблицах (engine=InnoDB)
  - mysql.innodb_table_stats, mysql.innodb_index_stats
- Можно отключить авто-перевычисление: **innodb_stats_auto_recalc=OFF**
- Можно настраивать статистику для каждой таблицы

Статистика – текущее состояние

- Сначала в Percona, потом в MySQL
  - Хранение статистики
  - Запрет авто-обновления

- Проблема #1:
  метод случайных проб
  - DBT-3
  - Scale=30
  - Гоняем EXPLAINы
  - Считаем планы



- Проблема #2: Статистики не хватает. Есть только
  - Число записей в таблице
  - Число разных значений в *индексе*

MariaDB 10.0: Engine independent statistics

- Собирается/используется на SQL уровне
  - mysql.{table,index,column}_stat
- Автообновления нет
  - ANALYZE TABLE
- 100% точная, детерминированная статистика
- Больше данных
  - Статистика по индексам (уже есть)
  - Статистика по таблицам (уже есть)
  - Статистика по колонкам
    - MIN/MAX значение
    - Число NULL / not-NULL
  - Гистограммы нескольких типов

=> Оптимизатор будет умен и предсказуем.

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

PERFORMANCE_SCHEMA

- Новое средство диагностики
  - “На каких lock'ах ждет соединение X?”
  - “На каких lock'ах вообще ждут?”
  - “Сколько MB/sec (или rows/sec) мы пишем/читаем в таблицу/index $NAME?”
  - ...
- Появилось в MySQL 5.5, доработано в 5.6
- Оверхед стал меньше
  - Но все еще есть (5%...15%)

Направления разработки

- InnoDB
- Репликация
- Оптимизатор
- PERFORMANCE_SCHEMA
- NoSQL

NoSQL

- MariaDB 5.3
  - Handlersocket плагин
  - Dynamic columns
- MySQL 5.6
  - Memcached плагин для InnoDB
- MariaDB 10.0
  - Поддержка имен колонок в Dynamic Columns
    - В 5.5 в качестве имен были integers.
  - Табличный движок Cassandra
  - Табличный движок Connect.

Спасибо!

Вопросы / Ответы

Итого

- Oracle движет вперед InnoDB
  - Некоторые идеи подает Percona
- Oracle движет вперед PERFORMANCE_SCHEMA
  - В 5.6 она стала интересной.
- MariaDB 5.5 имеет продвинутый оптимизатор
  - Даже MySQL 5.6 не догнал
- В репликации
  - MariaDB 5.5 двинулась вперед
  - MySQL 5.6 решил вообще все перевернуть с GTID.
    - В MariaDB 10.0 будет другой дизайн