MariaDB 5.5

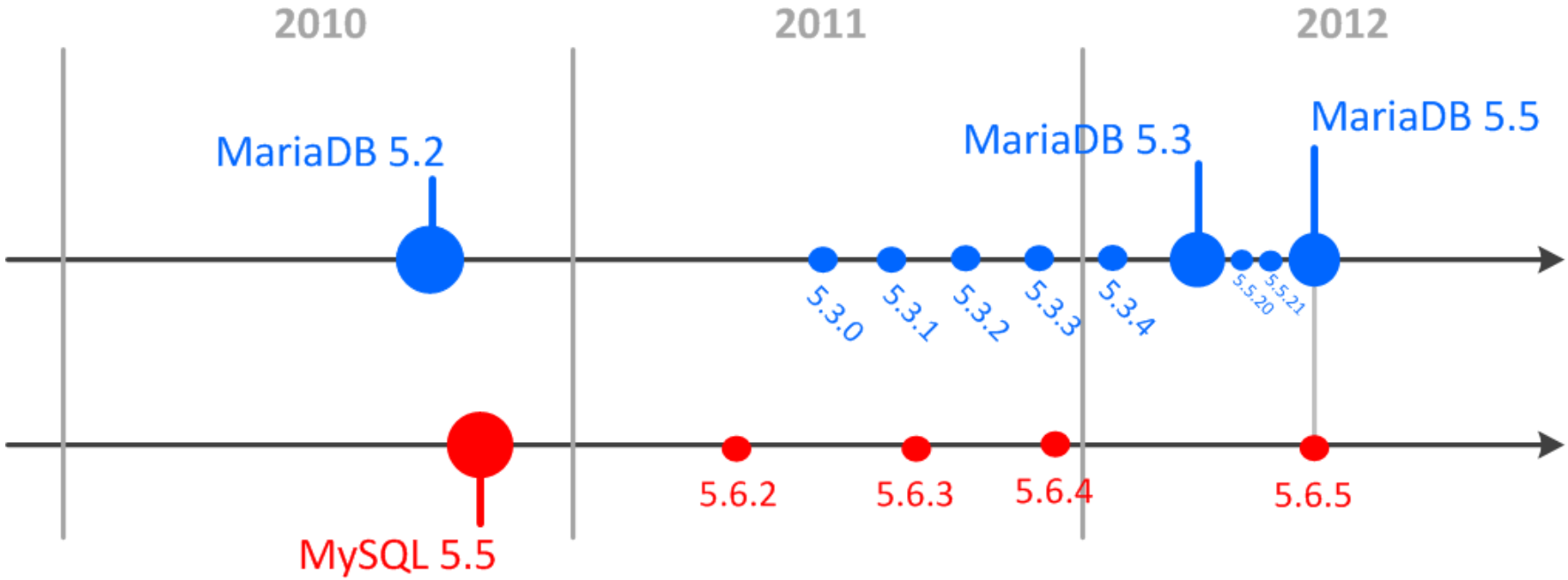
ветка MySQL с эволюционными и революционными изменениями

DevConf 2012, Сергей Петруня, Monty Program Ab

Что такое MariaDB

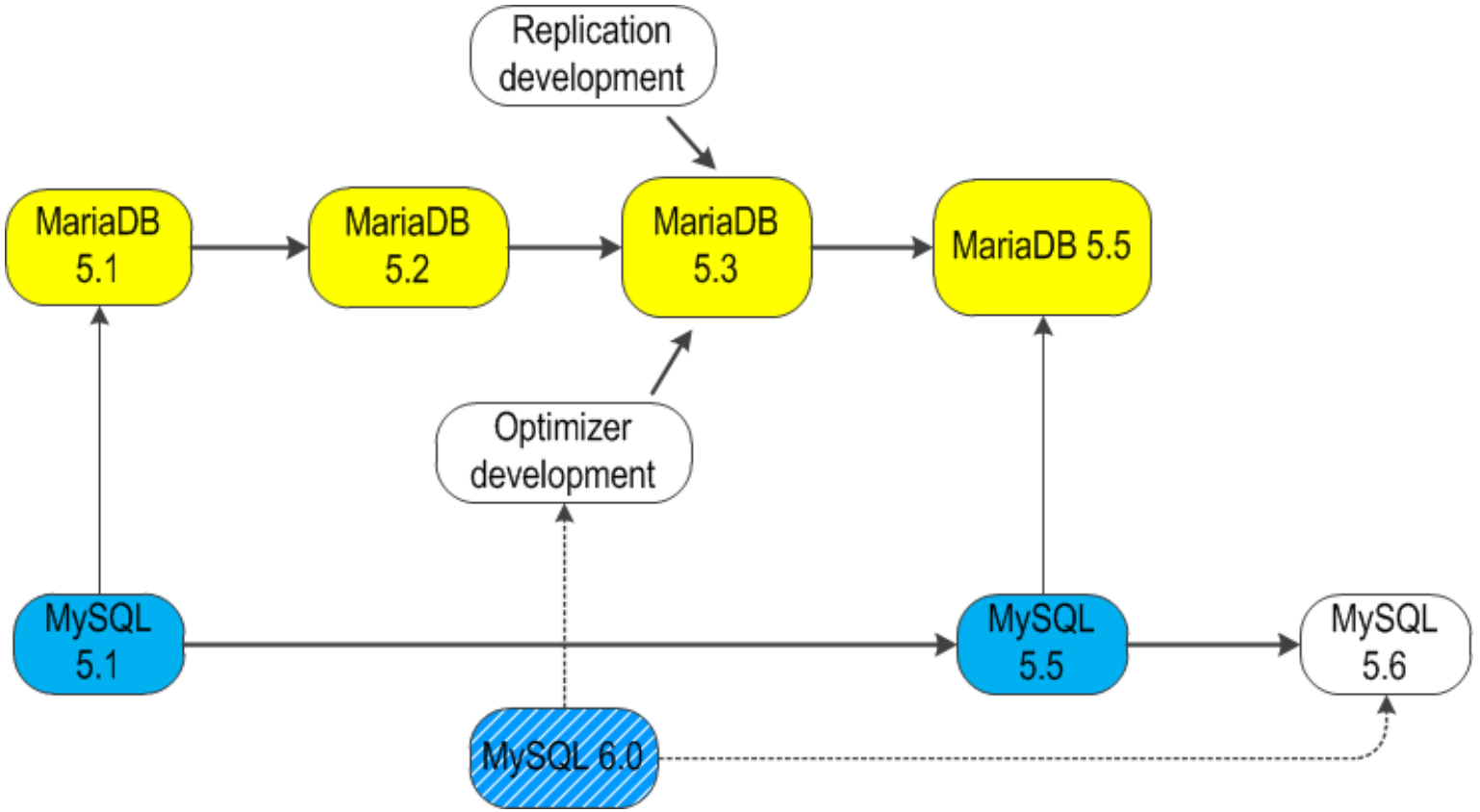
- Это ветка (branch) субд MySQL
- Лицензия - GPL
- Полностью совместима с «основным» MySQL
  - форматы файлов
  - соединения master<->slave
  - имена бинарников, init-скрипты и тд.
- Отличия от MySQL
  - Собственные фичи
  - По умолчанию вместо InnoDB - Percona's XtraDB
  - Интегрированы разные патчи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

Релизы MariaDB и MySQL



- Работа над MariaDB 5.3 заняла более года
- Сразу после MariaDB 5.3 вышла MariaDB 5.5
- Стабильная версия MySQL 5.5, есть “milestone releases” (беты) MySQL 5.6

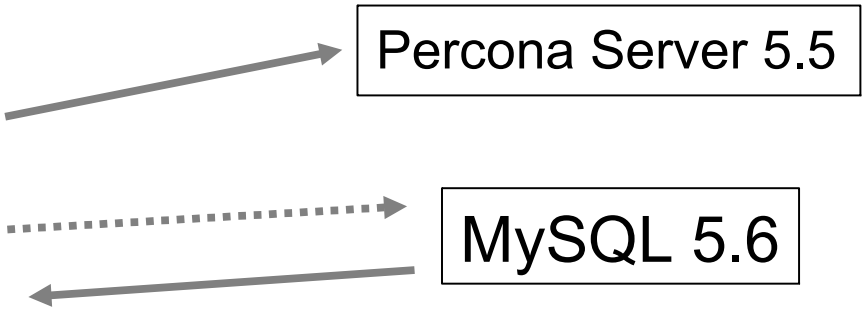
Что попало в MariaDB 5.5



Репликация в MariaDB 5.3/5.5

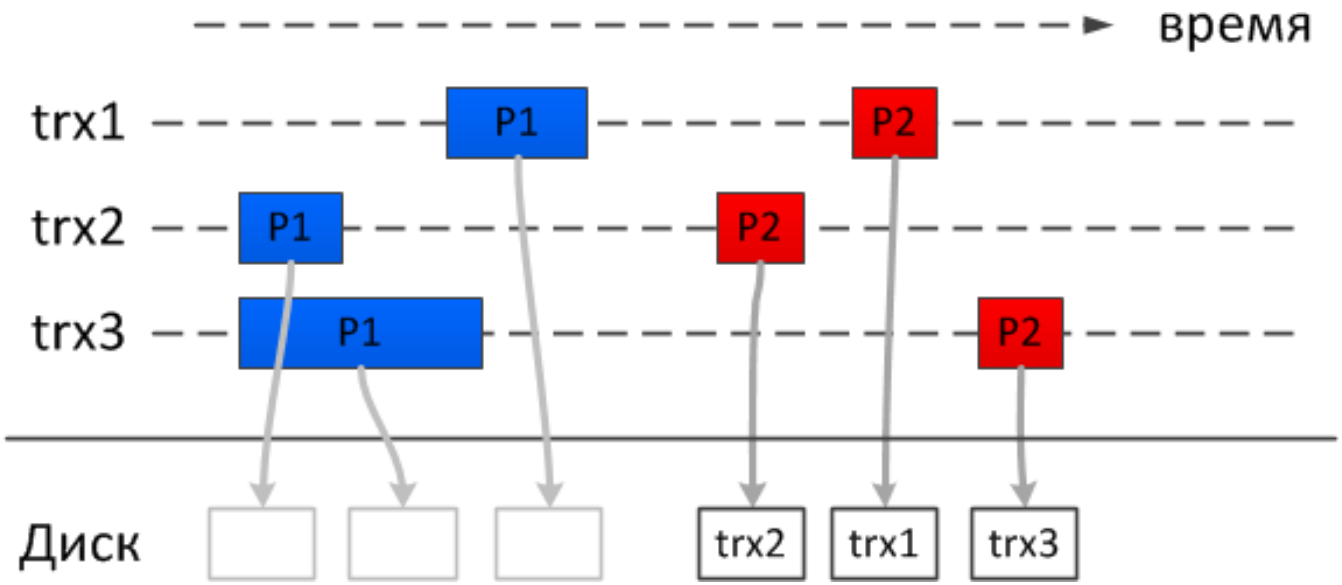
Репликация в MariaDB 5.3/5.5

- Group commit for binary log
- Annotations for RBR events
- Checksums for binlog events
- Неблокирующий mysqldump –single-transaction --master-data
- Optimized RBR for tables without primary key
- @@skip_replication
- Множество мелких улучшений



Что такое Group Commit

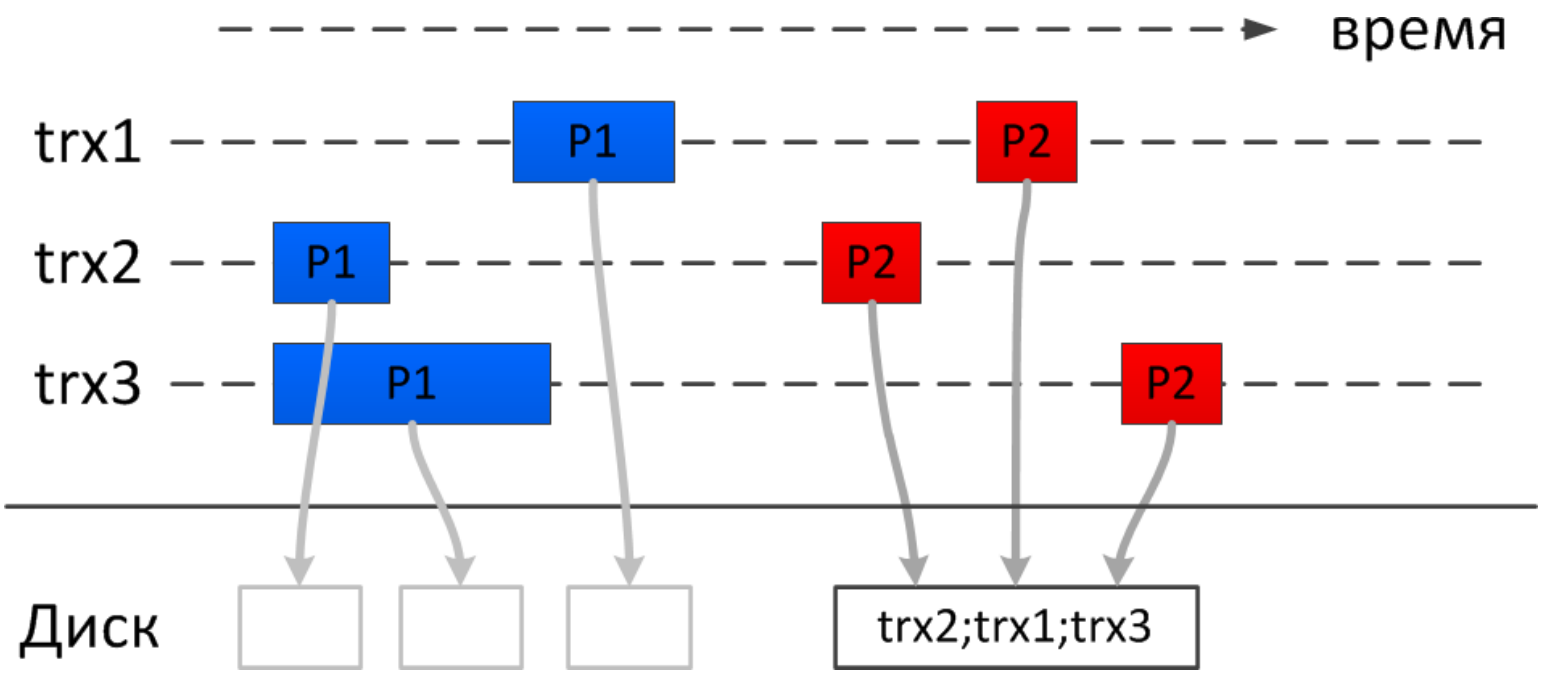
- Для обеспечения транзакционной работы СУБД должна в момент commit'a фиксировать сделанные изменения



- Если фиксировать всех отдельно - ~200 коммитов/сек.

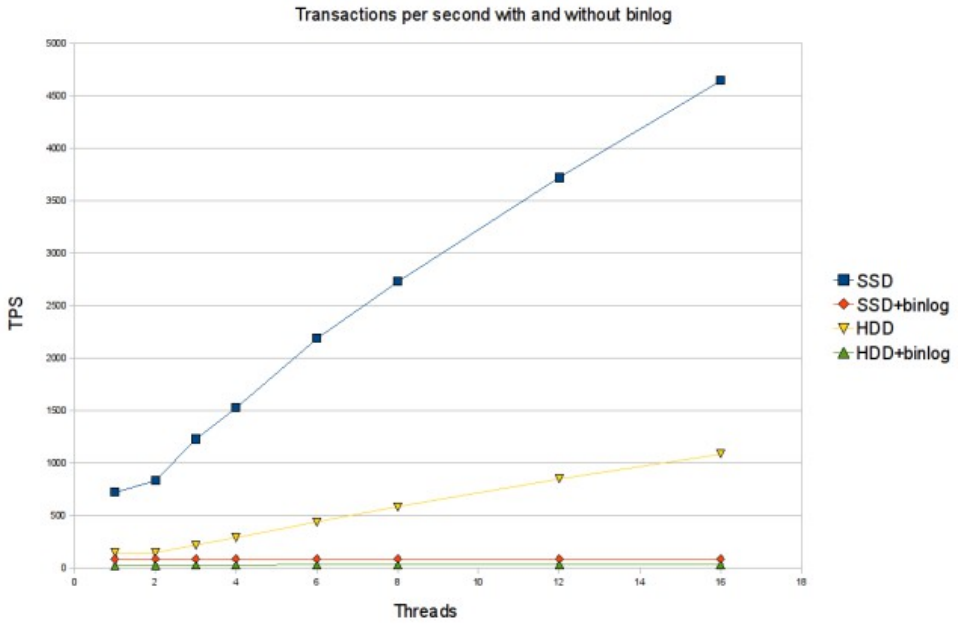
Идея Group Commit

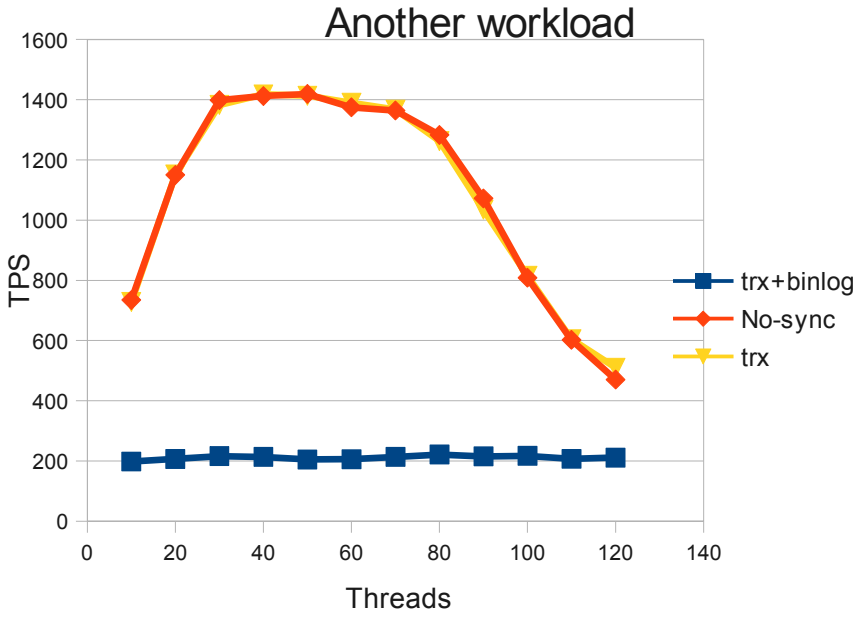
- фиксировать изменения для групп, а не отдельных транзакций.



Фиксация изменений в MySQL

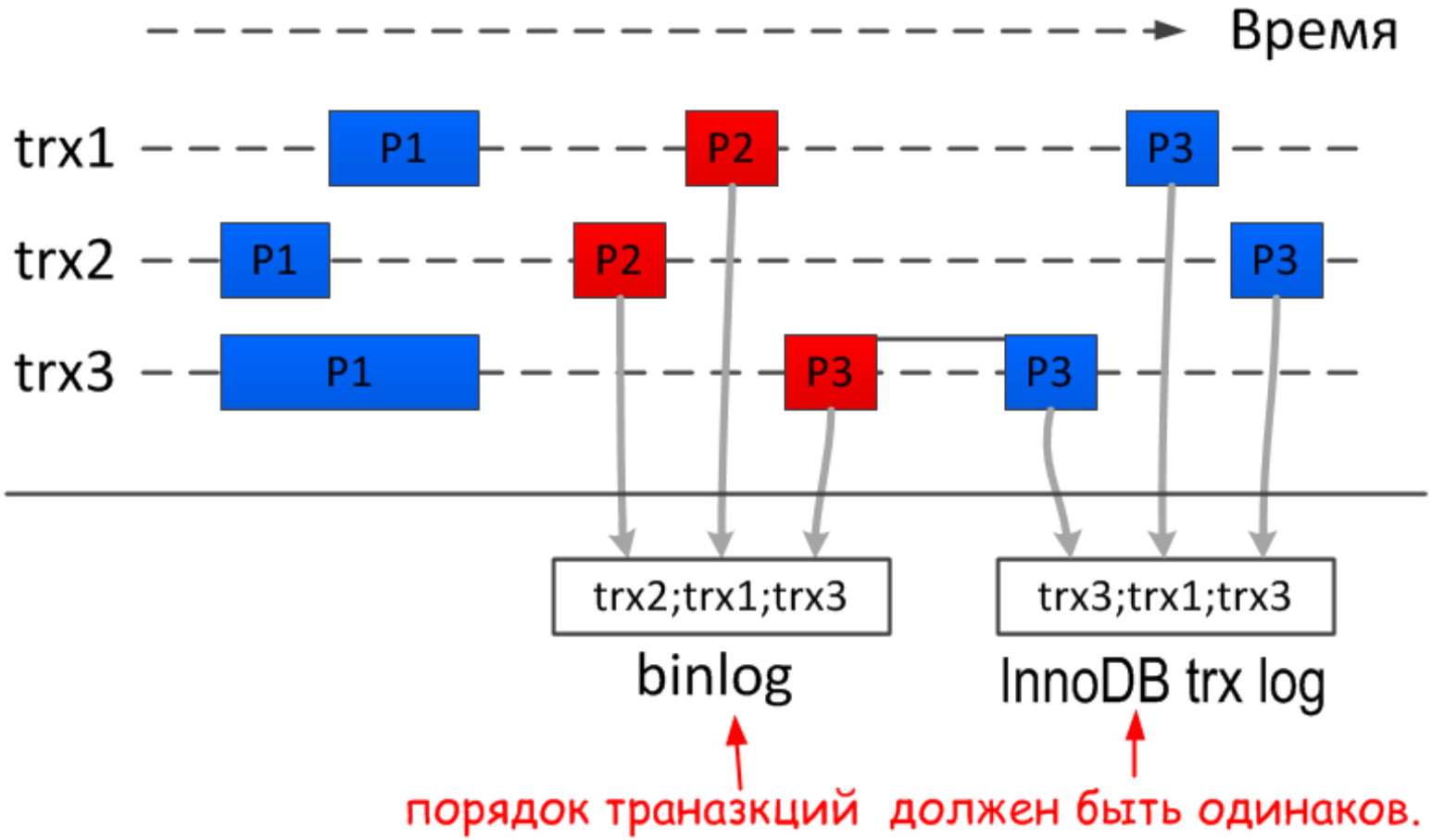
- Фиксировать нужно:
  - В InnoDB: **innodb_flush_log_at_trx_commit=1**
  - В binlog'e: **sync_binlog=1**
- Если включить оба, group commit не работает:





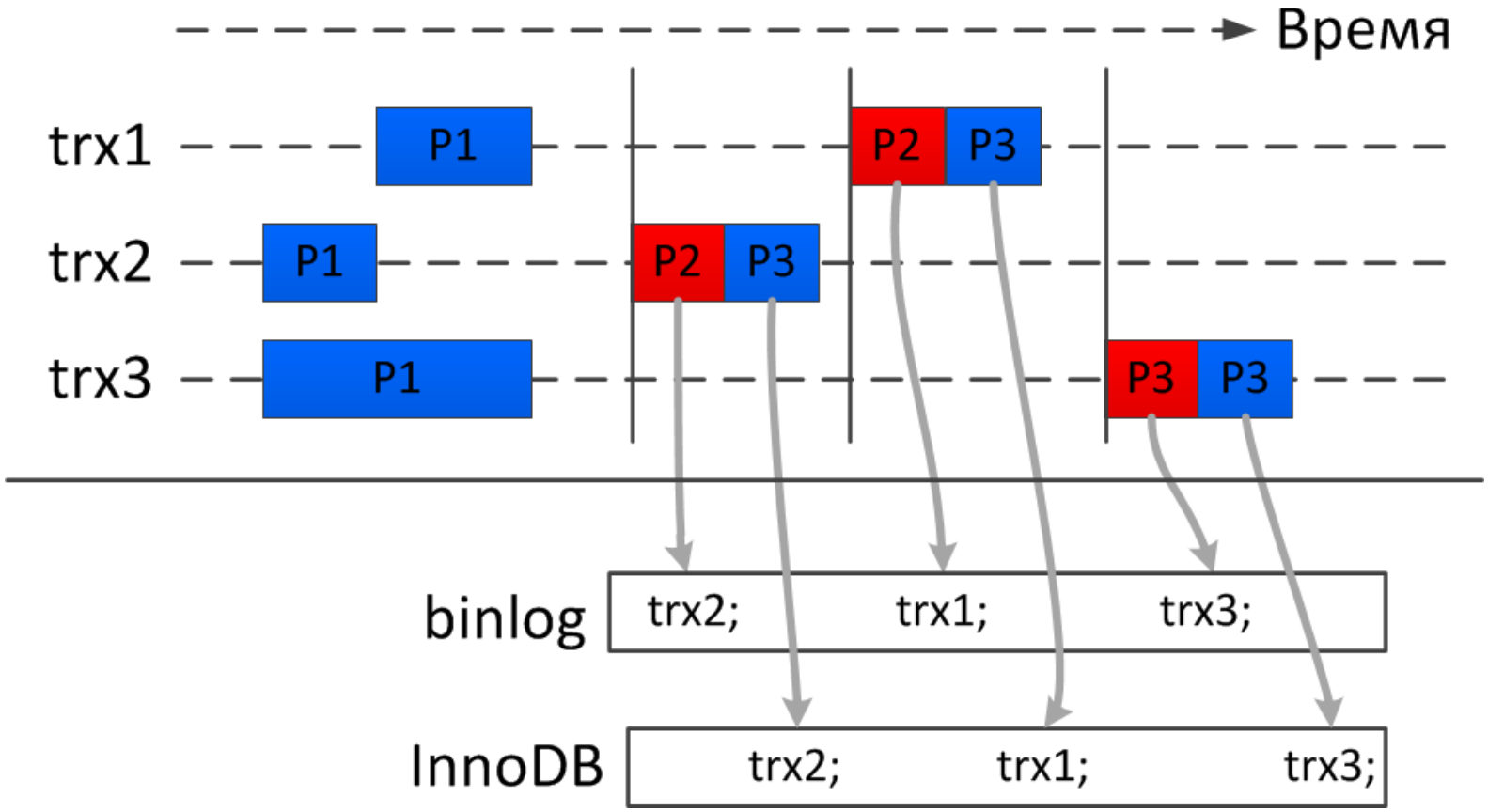
Почему не работает group commit?

- Нужен один и тот же порядок записи транзакций

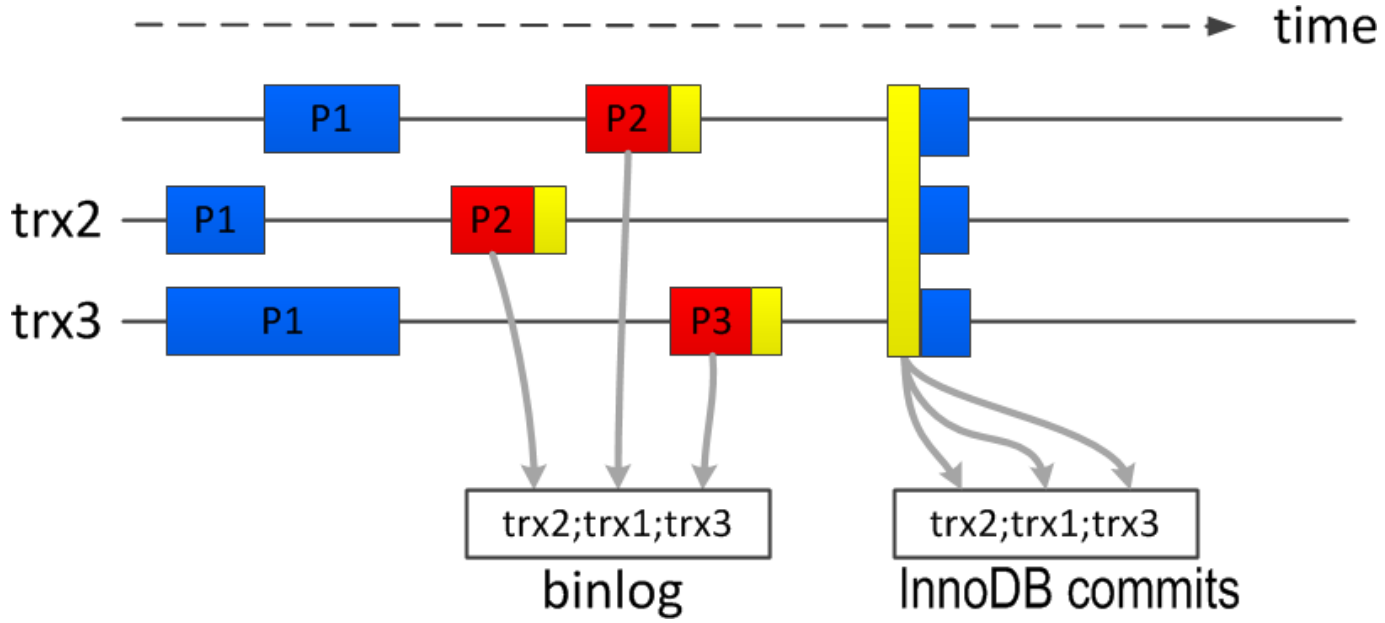


Почему не работает group commit (2)?

- Свойство один-и-тот-же порядок достигается за счет блокировок



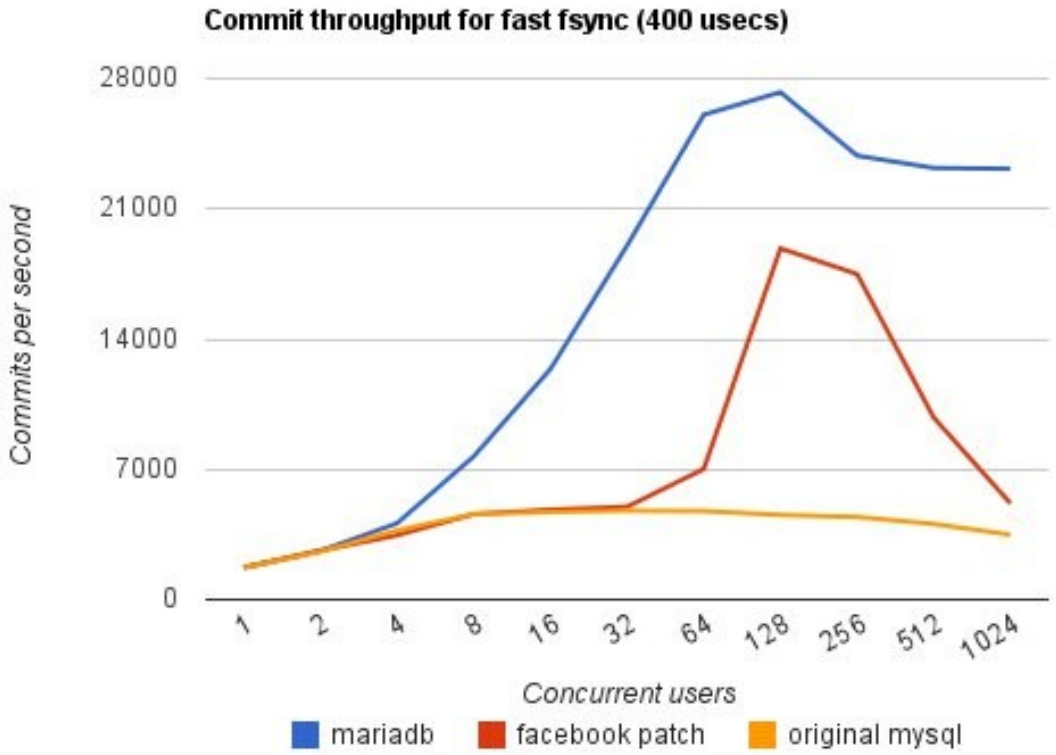
Group Commit с binlog'ом



- Реализации от: Facebook, MariaDB, preview tree от Oracle
- Общая идея
  - При записи в binlog фиксируется очередность
  - Потом, в InnoDB фиксируются изменения в том же порядке

Производительность с binlog'ом

- По последним данным, MariaDB 5.3/5.5 лучше
  - Она же спортирована в Percona Server 5.5



* график из “Group commit in MariaDB is fast” поста Mark Callaghan

Аннотации RBR events в MariaDB

К каждой RBR-записи в binlog'е приписывается текст исходного запроса

- mysqld --binlog_annotate_row_events

```
MariaDB [test]> show binlog events in 'pslp.000299';
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| Log_name    | Pos | Event_type  | Server_id | End_log_pos | Info                                          |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------+
| pslp.000299 |   4 | Format_desc |         1 |         245 | Server ver: 5.3.4-MariaDB-rc-log, Binlog ver: 4 |
| pslp.000299 | 245 | Query       |         1 |         313 | BEGIN                                         |
| pslp.000299 | 313 | Annotate_rows |       1 |         379 | /* app1 */ insert into t1 values('foo'),('bar') |
| pslp.000299 | 379 | Table_map   |         1 |         422 | table_id: 15 (test.t1)                        |
| pslp.000299 | 422 | Write_rows  |         1 |         461 | table_id: 15 flags: STMT_END_F                |
| pslp.000299 | 461 | Query       |         1 |         530 | COMMIT                                        |
| pslp.000299 | 530 | Query       |         1 |         598 | BEGIN                                         |
| pslp.000299 | 598 | Annotate_rows |       1 |         664 | /* app2 */ update t1 set a='test' where a='foo' |
| pslp.000299 | 664 | Table_map   |         1 |         707 | table_id: 15 (test.t1)                        |
| pslp.000299 | 707 | Update_rows |         1 |         759 | table_id: 15 flags: STMT_END_F                |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------+
```

```
# at 379
# at 422
#120203 16:25:11 server id 1  end_log_pos 379   Annotate_rows:
#Q> /* app1 */ insert into t1 values('foo'),('bar')
#120203 16:25:11 server id 1  end_log_pos 422   Table_map: `test`.`t1` mapped to number 15
#120203 16:25:11 server id 1  end_log_pos 461   Write_rows: table id 15 flags: STMT_END_F

BINLOG '
J9IrTxMBAAAAKwAAAKYBAAAAAA8AAAAAAAEABHRlc3QAAnQxAAEPAgwAAQ==
```

MySQL 5.6 тоже будет поддерживать аннотации

- Реализация очень схожа с реализацией в MariaDB
- “интерфейс” другой
  - mysqld --binlog-rows-query-log-events

```
MySQL [test]> show binlog events;
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
| Log_name    | Pos | Event_type  | Server_id | End_log_pos | Info                                                |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
| pslp.000001 |   4 | Format_desc |         1 |         114 | Server ver: 5.6.5-m8-debug-log, Binlog ver: 4       |
| pslp.000001 | 114 | Query       |         1 |         215 | use `test`; create table t1 (a varchar(12))         |
| pslp.000001 | 215 | Query       |         1 |         283 | BEGIN                                               |
| pslp.000001 | 283 | Rows_query  |         1 |         350 | # /* app1 */ insert into t1 values('foo'),('bar')   |
| pslp.000001 | 350 | Table_map   |         1 |         393 | table_id: 66 (test.t1)                              |
| pslp.000001 | 393 | Write_rows  |         1 |         432 | table_id: 66 flags: STMT_END_F                      |
| pslp.000001 | 432 | Xid         |         1 |         459 | COMMIT /* xid=5 */                                  |
| pslp.000001 | 459 | Query       |         1 |         527 | BEGIN                                               |
| pslp.000001 | 527 | Rows_query  |         1 |         594 | # /* app2 */ update t1 set a='test' where a='foo'   |
| pslp.000001 | 594 | Table_map   |         1 |         637 | table_id: 66 (test.t1)                              |
| pslp.000001 | 637 | Update_rows |         1 |         678 | table_id: 66 flags: STMT_END_F                      |
| pslp.000001 | 678 | Xid         |         1 |         705 | COMMIT /* xid=6 */                                  |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
```

Назревающая проблема совместимости

- В MariaDB 5.3 добавили **Annotate_rows**
- В MySQL 5.6 добавили **Rows_query**
- Это разные типы записей
  - MariaDB 5.3/5.5 не понимает **Rows_query**
  - MySQL 5.6 не понимает **Annotate_rows**
- При встрече неизвестной записью в binlog, репликация остановится с ошибкой (продолжать было бы небезопасно)
- В MySQL 5.6 будет флаг “эту запись можно игнорировать”
  - Мы его сразу же смержим в MariaDB
  - И binary logs станут опять совместимы

Оптимизация RBR для таблиц без primary key

- Row Based Replication пишет в binlog записи вида:

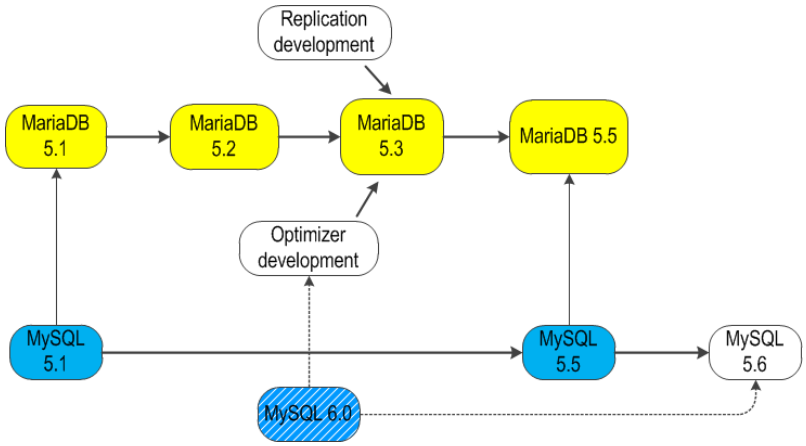
```
with table
        $tbl=`dbname.tablename` (col1 INT, col2 CHAR(N),...)
do
        delete in $tbl row with (col1,col2) = (10, 'foo')
```

- До MariaDB 5.3, для поиска записи использовался **первый** индекс в
  - Если есть PRIMARY KEY, он первый
  - Если его нету - могли выбрать плохой индекс
- В MariaDB 5.3
  - Если есть PRIMARY KEY, использовать его
  - или первый UNIQUE INDEX без NULL-ов
  - Или, наиболее селективный не-fulltext индекс

Изменения в Оптимизаторе

Изменения в оптимизаторе

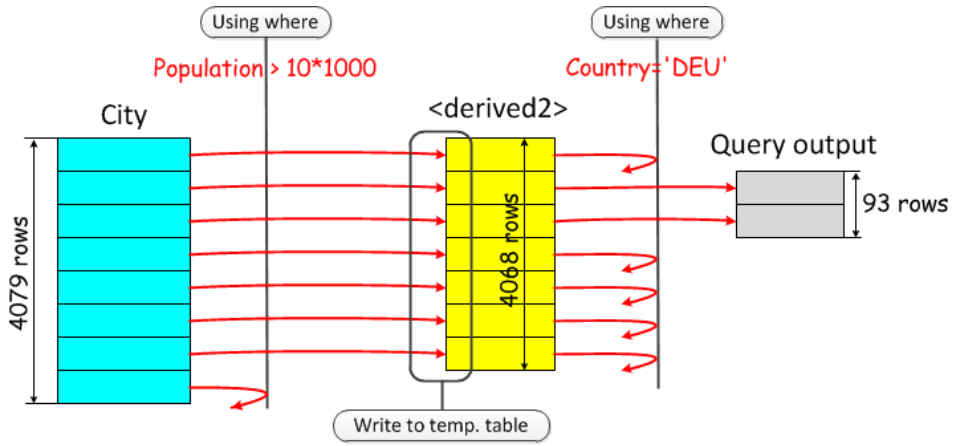
- Переписана оптимизация подзапросов
  - WHERE … IN (SELECT ..)
  - FROM (SELECT ...)
  - прочих
- Добавлены оптимизации для “больших” запросов
  - Batched Key Access
  - Hash Join
  - Index Condition Pushdown
  - Улучшенный Index Merge



Оптимизация подзапросов: FROM

- Часто используется для “структурирования” запроса:

```
SELECT * FROM
  (SELECT * FROM City WHERE Population > 10*1000) AS big_city
WHERE
  big_city.Country='DEU'
```



- Выполнение в MySQL:
  1. Материализовать таблицу big_city
  2. Вычислять запрос дальше
- Вывод пользователей: не писать FROM-запросов

Оптимизация FROM в MariaDB

В MariaDB (и в будущем, в MySQL 5.6)

- Если можно, FROM-подзапросы “сливаются” со своими родителями
- Если без материализации не обойтись, рассматривается вариант с добавлением индекса на материализованную таблицу

Оптимизация подзапросов: WHERE … IN (SELECT…)

- По статистике, самые часто используемые

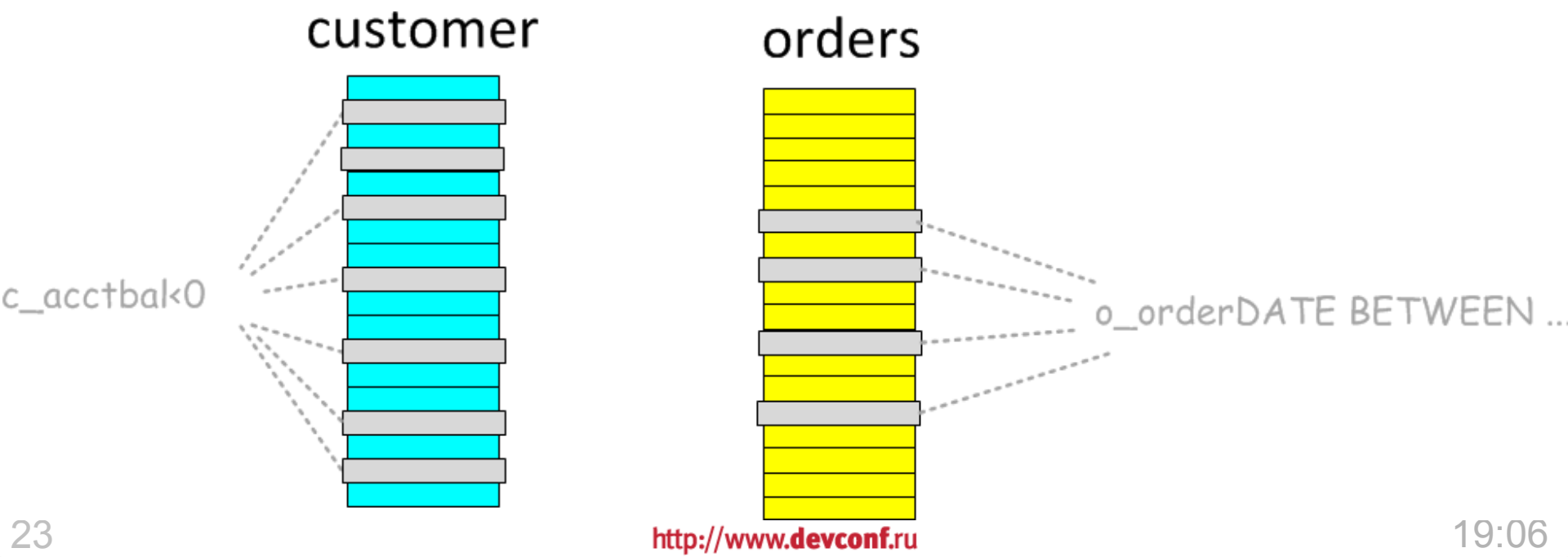
```
select * from customer
where
  customer.c_acctbal < 0 and
customer.c_custkey in (select orders.o_custkey
                       from orders
                       where orders.o_orderDATE between $DAY1 and $DAY2
                      )
```

- В MySQL: вычисление только “снаружи-внутрь”:

```
+----+--------------------+----------+----------------+---------------+-------------+---------+------+---------+-------
| id | select_type        | table    | type           | possible_keys | key         | key_len | ref  | rows    | Extra
+----+--------------------+----------+----------------+---------------+-------------+---------+------+---------+-------
|  1 | PRIMARY            | customer | ALL            | NULL          | NULL        | NULL    | NULL | 1494351 | Using
|  2 | DEPENDENT SUBQUERY | orders   | index_subquery | ...           | i_o_custkey | 5       | func |       7 | Using
+----+--------------------+----------+----------------+---------------+-------------+---------+------+---------+-------
```

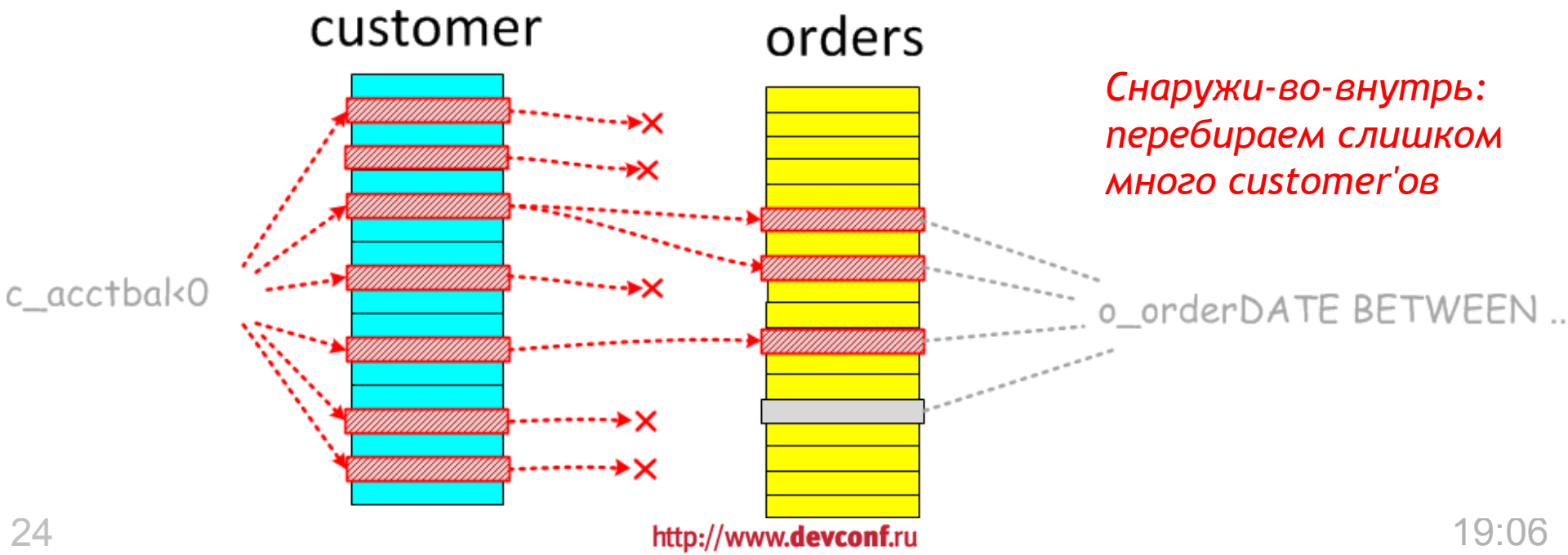
Выполнение подзапроса WHERE … IN (SELECT…)

```
select * from customer
where
  customer.c_acctbal < 0 and
customer.c_custkey in (select orders.o_custkey
                       from orders
                       where orders.o_orderDATE between $DAY1 and $DAY2
                      )
```



Выполнение подзапроса WHERE … IN (SELECT…)

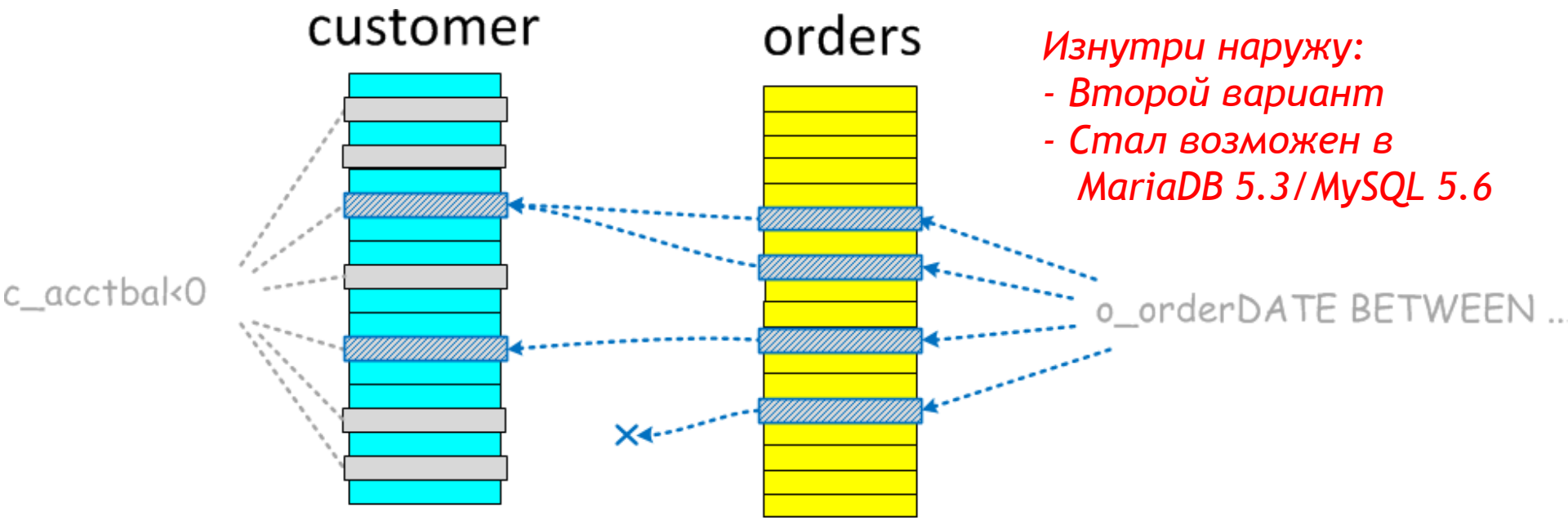
```
select * from customer
where
  customer.c_acctbal < 0 and
customer.c_custkey in (select orders.o_custkey
                       from orders
                       where orders.o_orderDATE between $DAY1 and $DAY2
                      )
```



Снаружи-во-внутрь: перебираем слишком много customer'ов

Выполнение подзапроса WHERE ... IN (SELECT...)

```
select * from customer
where
  customer.c_acctbal < 0 and
customer.c_custkey in (select orders.o_custkey
                       from orders
                       where orders.o_orderDATE between $DAY1 and $DAY2
                      )
```



Изнутри наружу:
- Второй вариант
- Стал возможен в MariaDB 5.3/MySQL 5.6

Два варианта выполнения

```
select * from customer
where
  customer.c_acctbal < 0 and
customer.c_custkey in (select orders.o_custkey
                       from orders
                       where orders.o_orderDATE between $DAY1 and $DAY2
                      )
```

MySQL 5.1: только снаружи-внутрь

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	customer	ALL	NULL	NULL	NULL	NULL	1494351	Using w
2	DEPENDENT SUBQUERY	orders	index_subquery	...	i_o_custkey	5	func	7	Using w

MariaDB 5.5 (и MySQL 5.6, когда выпустят): добавляются стратегии изнутри-наружу

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<subquery2>	ALL	distinct_key	NULL	NULL	NULL	22906	
1	PRIMARY	customer	eq_ref	PRIMARY	PRIMARY	4	orders.o_custkey	1	Using
2	MATERIALIZED	orders	range	...	i_o_orderdate	4	NULL	22906	Using

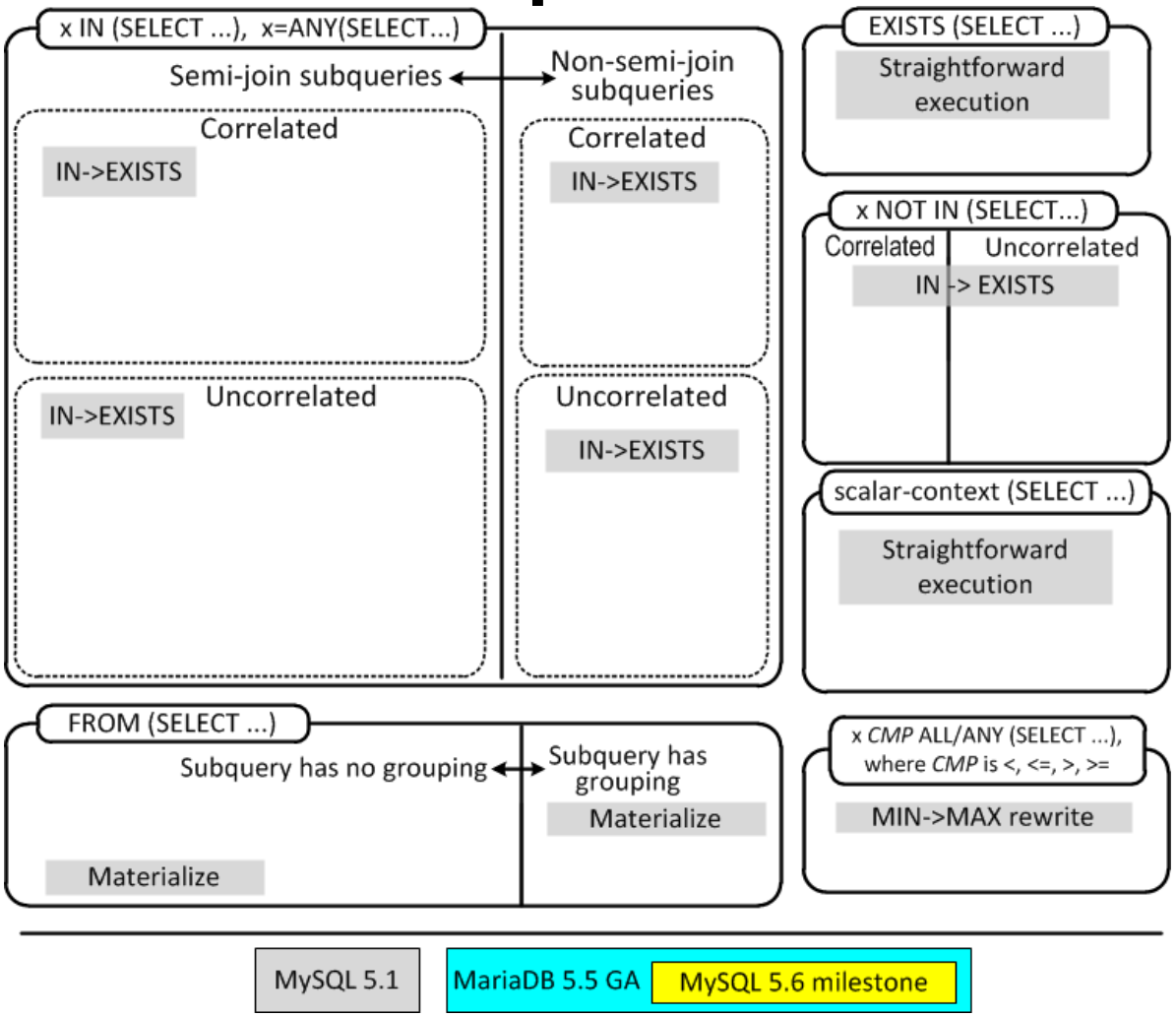
WHERE … IN (SELECT…), итого

- Теперь оптимизатор может выбрать порядок выполнения
- Если запрос можно переписать как JOIN, он будет автоматически переписан.
- Оптимизация и скорость – теперь похожи на JOIN'ы

Оптимизация подзапросов: другие виды

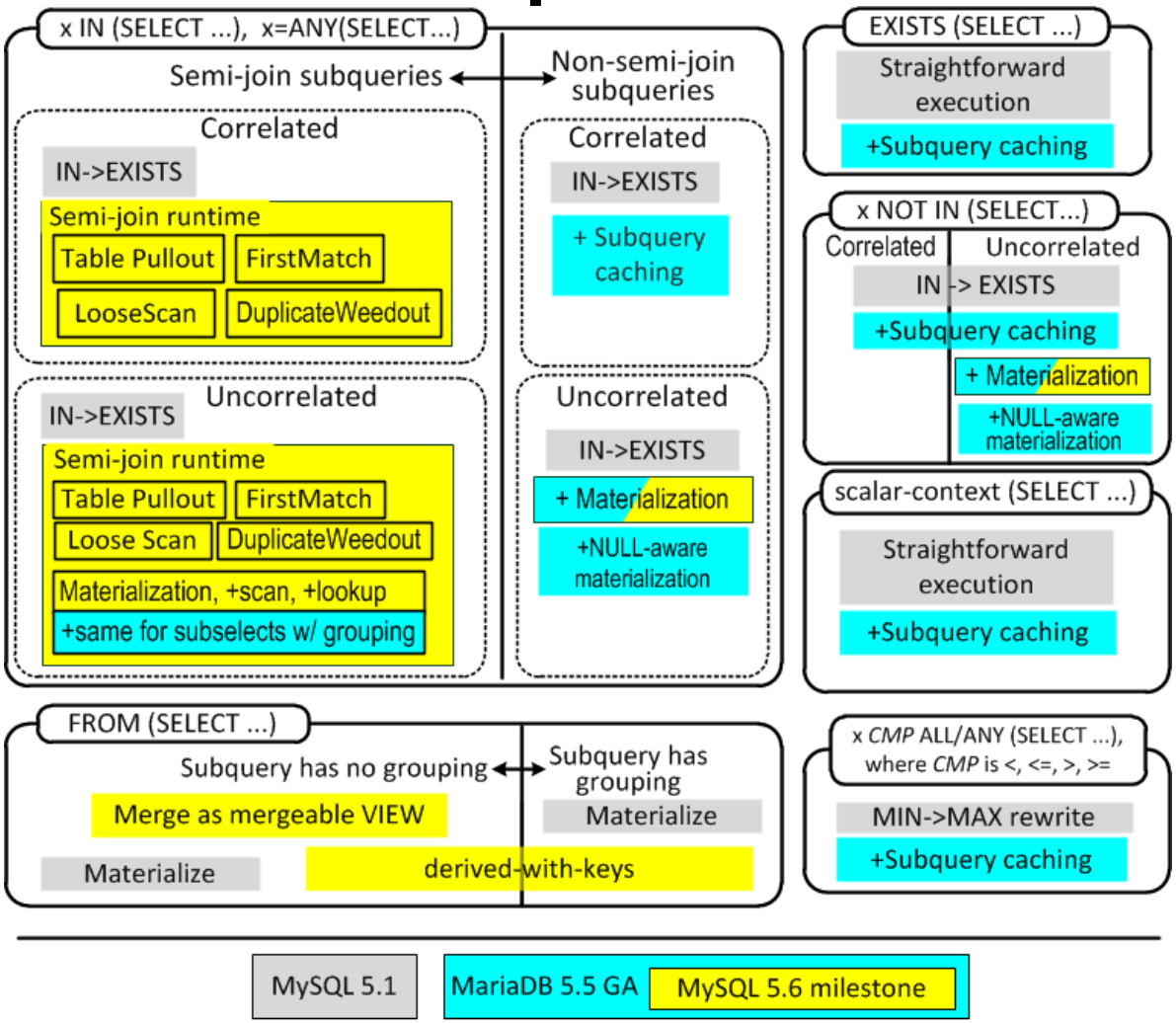
- Еще бывают подзапросы
  - expr IN (SELECT …) не во WHERE
  - EXISTS (SELECT …)
  - Просто (SELECT …) в скалярном контексте
- Для их обработки добавлены
  - Materialization для IN(SELECT …)
    (в MySQL 5.6 – ее подмножество)
  - Subquery cache

Карта оптимизаций



- В MySQL 5.1 для каждого типа подзапроса была только одна стратегия выполнения

Карта оптимизаций



- В MySQL 5.6 добавлены оптимизации для самых важных случаев
- В MariaDB 5.3/5.5 добавлены оптимизации для всех видов подзапросов

Подзапросы: итого

- На основе выборки “что попалось”:
  - Были в ~1000 раз медленнее PostgreSQL
  - Стали: тот же порядок
- MariaDB 5.5 сейчас имеет надмножество того, что будет в MySQL 5.6

Оптимизации для больших запросов

Что это и зачем

- Проекты начинают с базовых вещей
  - Раздача контента
  - Обработка транзакций (OLTP)
- SQL запросы
  - Либо затрагивают одну запись:
    SELECT * FROM web_pages WHERE key=...
    UPDATE user SET score=score+1 WHERE user.id=...
- Либо просматривают небольшие интервалы:
  SELECT * FROM forum_posts WHERE thread_id=123
    ORDER BY post_date DESC LIMIT 10
- MySQL такие запросы обрабатывает хорошо

Что это и зачем (2)

Дальше хочется отчетов и аналитики

- Данных становится больше
  - данные за сегодня → вся история
- Запросы просматривают много данных
  - “какова общая сумма в заказе X?” →
    “Какой процент заказов был на сумму > 5000 руб?”
- Запросы становятся сложными
  - “Выдать содержимое заказа X” →
    “Выдать список товаров, которые часто покупают вместе с товарами X, Y, Z”
- “MySQL не тянет большие запросы”

MariaDB- комплексное решение

- Подзапросы всех видов
  - (уже осветили)
- Оптимизации доступа к диску
  - Index Condition Pushdown
  - Batched Key Access
  - Hash join
  - Index Merge/sort-intersect

Запуск больших запросов на MariaDB

- Новые стратегии выполнения требуют больше памяти
- Бесполезны для мелких запросов
  - Могут даже вызвать замедление
- => Выключены по умолчанию
  - Включайте их перед тяжелыми запросами
    - Перед сбором аналитики, отчетов и т д.

Оптимизации доступа к диску

- Могут вызвать замедление на маленьких запросах
  - И поэтому выключены
  - Надо включать (перед аналитикой/отчетами)
- http://kb.askmonty.org/en/big-query-settings/

```
# Batched Key Access
optimizer_switch='mrr=on'
optimizer_switch='mrr_cost_based=off
'

join_cache_level = 6
join_buffer_space_limit = 300M
join_buffer_size = 100M
```

+

```
# HASH join
join_cache_level=8
```

```
# Index Merge sort-intersect
optimizer_switch='index_merge_sort_intersection=on
'
```

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

Еще пара интересных фич из MariaDB 5.5

- Асинхронное клиентское API
  - “Родная” клиентская библиотека libmysql поддерживает асинхронный режим работы
    - Можно установить несколько соединений (с MariaDB и/или MySQL серверами)
    - И параллельно запустить несколько запросов.
- Thread pool в сервере
  - Для всех платформ, включая Windows
  - У Oracle – только в Enterprise-версии (распространяется за плату и без исходного кода)

LIMIT ROWS EXAMINED

- В MySQL есть @@max_join_size

```
MariaDB [test]> select * from big_table where col=10;
ERROR 1104 (42000): The SELECT would examine more than
MAX_JOIN_SIZE rows...
```

  - Использует *оценку* числа прочитанных записей
  - Не учитывает подзапросы и т д.

- LIMIT ROWS EXAMINED n – задает максимальное число записей, которое разрешено просмотреть
  - Ограничение времени выполнения. Действует на весь запрос

```
MariaDB [test]> select * from big_table
             ->   where col=10 limit rows examined 100;
...
Warning (Code 1931): Query execution was interrupted. The
query examined at least 101 rows, which exceeds LIMIT ROWS
EXAMINED (100). The query result may be incomplete.
```

Миграция между MySQL и MariaDB

- “Drop-in replacement”
  - Просто запустите новый сервер с тем же --datadir
  - Репликация: цепляйте MariaDB slave к MySQL master и наоборот
  - Клиент-серверный протокол такой же, как и у MySQL
- Почти все фичи оптимизатора выключаемые
  `set optimizer_switch='optimization_name=off'`
  - По умолчанию включены только безопасные

Дальнейшая информация

- MariaDB Knowledgebase
  http://kb.askmonty.org/
- https://lists.launchpad.net/maria-developers/
- irc: #maria на FreeNode