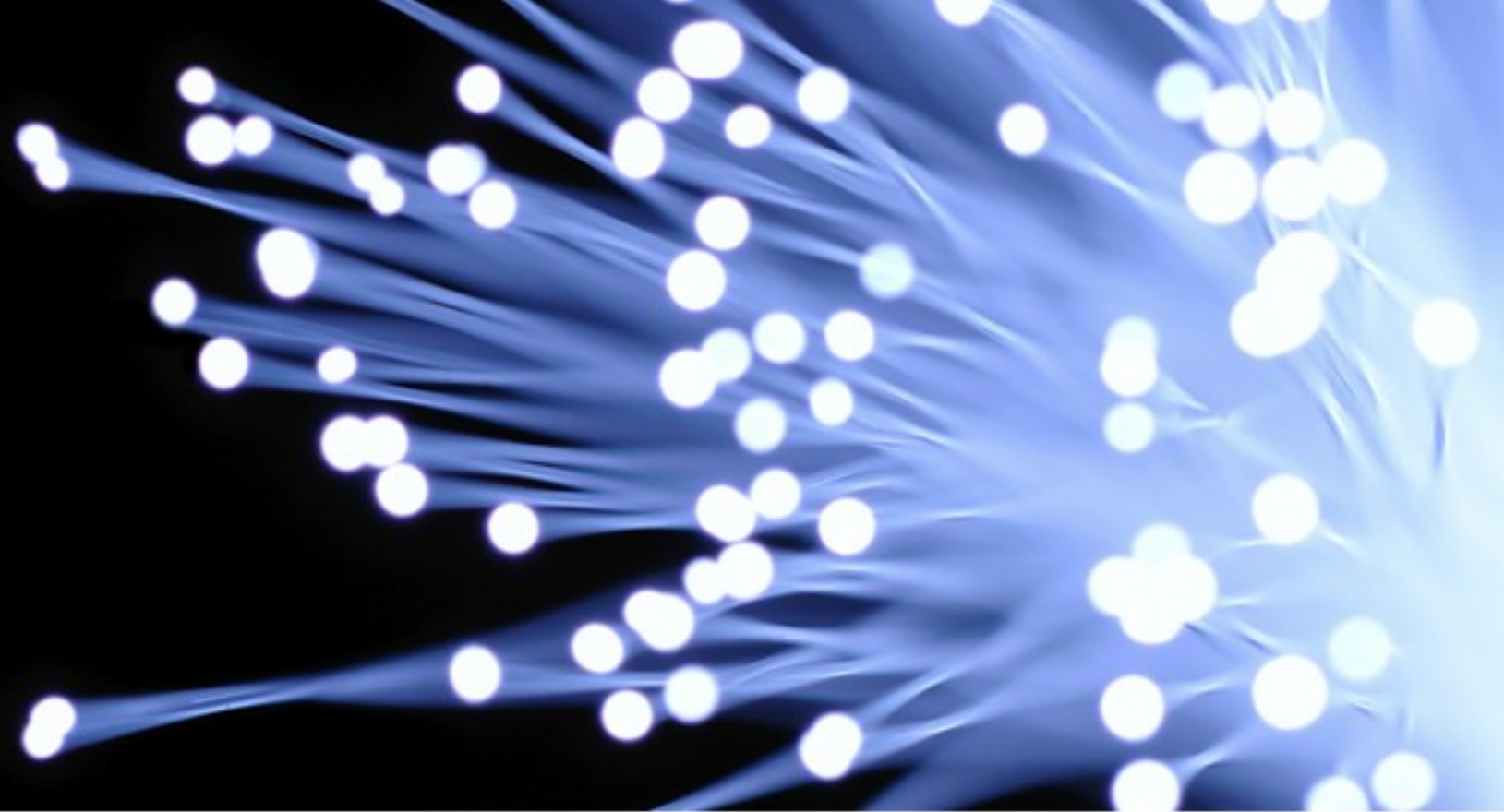


O'REILLY

**MySQL**

Conference & Expo

INFORMATION UNLEASHED



# New query engine features in MariaDB

Sergey Petrunya, Monty Program Ab



- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Predicate caching
  - FROM subquery optimizations



- **Table elimination (MariaDB 5.1)**
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Predicate caching
  - FROM subquery optimizations



- Applicable for queries over normalized data
- Present in “big” databases like Oracle, SQL Server
  - And will be in PostgreSQL 9.0 (they call it “join removal”)

- Basic idea

*Detect outerjoins that have “unused” inner sides and delete those inner sides*

```
SELECT tbl1.*  
FROM  
  tbl1 LEFT JOIN tbl2 ON tbl2.primary_key=tbl1.id  
WHERE  
  condition(tbl1.*)
```

- It is guaranteed that for each record of tbl1
  - tbl2 will have not more than one match (tbl2.primary\_key=..)
  - If tbl2 has no match, LEFT JOIN will generate a NULL-record

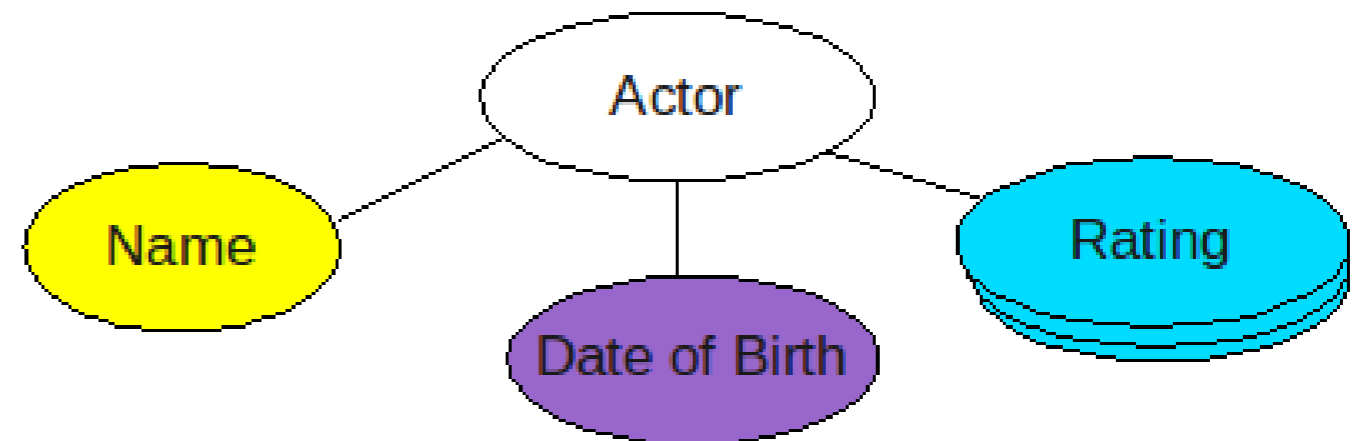
# The case for table elimination



- Highly-normalized data:

```
actor(name,  
      date_of_birth,  
      rating)
```

is stored as:



```
create table ac_anchor(AC_ID int primary key);
```

```
create table ac_name(AC_ID int, ACNAM_name char(N),  
                    primary key(AC_ID));
```

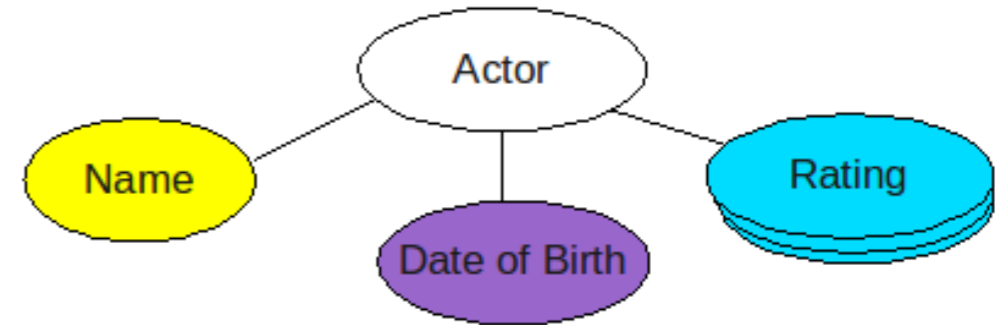
```
create table ac_dob(AC_ID int, ACDOB_birthdate date,  
                  primary key(AC_ID));
```

```
create table ac_rating(AC_ID int,  
                      ACRAT_rating int, ACRAT_fromdate date,  
                      primary key(AC_ID, ACRAT_fromdate));
```

# The case for table elimination (2)



- Then select back:



```
create view actors as select * from
```

```
select
```

```
  ac_anchor.AC_ID, ACNAM_Name, ACDOB_birthdate, ACRAT_rating
```

```
from
```

```
  ac_anchor
```

```
  left join ac_name on ac_anchor.AC_ID=ac_name.AC_ID
```

```
  left join ac_dob on ac_anchor.AC_ID=ac_dob.AC_ID
```

```
  left join ac_rating on (ac_anchor.AC_ID=ac_rating.AC_ID and
    ac_rating.ACRAT_fromdate =
    (select max(sub.ACRAT_fromdate)
     from ac_rating sub
     where sub.AC_I=ac_rating.AC_ID))
```

```
select ACRAT_rating from actors where ACNAM_name='Gary Oldman';
```

# Table elimination – examples



```
explain select ACRAT_rating, ACDOB_birthdate from actors where ACNAM_name='Gary Oldman';
```

id	select_type	table	type	possible_keys	key	key_len	ref
1	PRIMARY	ac_anchor	index	PRIMARY	PRIMARY	4	NULL
1	PRIMARY	ac_name	eq_ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
1	PRIMARY	ac_dob	eq_ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
1	PRIMARY	ac_rating	ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
3	DEPENDENT SUBQUERY	sub	ref	PRIMARY	PRIMARY	4	ac_rating.AC_ID

```
explain select ACRAT_rating from actors where ACNAM_name='Gary Oldman';
```

id	select_type	table	type	possible_keys	key	key_len	ref
1	PRIMARY	ac_anchor	index	PRIMARY	PRIMARY	4	NULL
1	PRIMARY	ac_name	eq_ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
1	PRIMARY	ac_rating	ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
3	DEPENDENT SUBQUERY	sub	ref	PRIMARY	PRIMARY	4	ac_rating.AC_ID

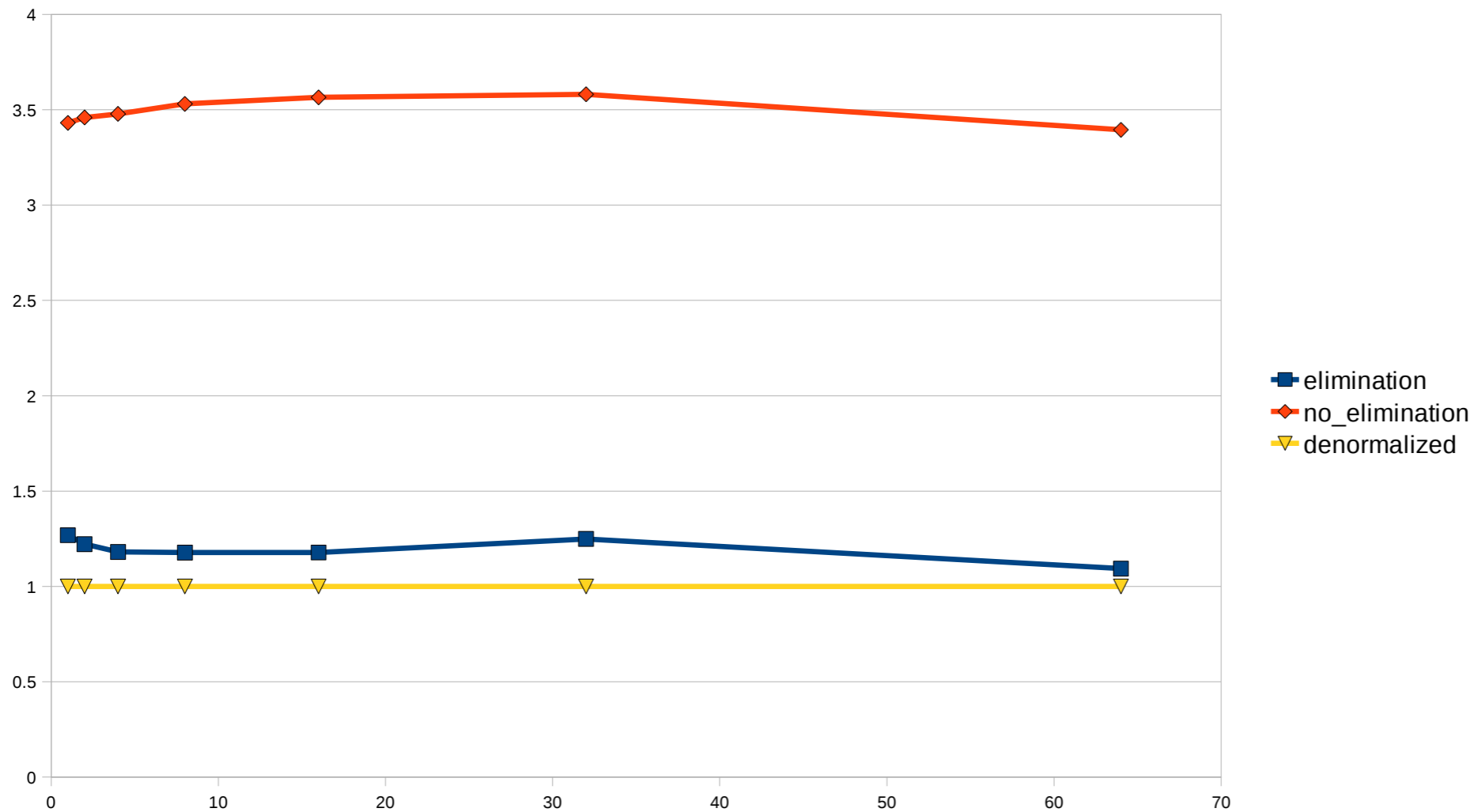
```
explain select ACDOB_birthdate from actors where ACNAM_name='Gary Oldman';
```

id	select_type	table	type	possible_keys	key	key_len	ref
1	PRIMARY	ac_anchor	index	PRIMARY	PRIMARY	4	NULL
1	PRIMARY	ac_name	eq_ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID
1	PRIMARY	ac_dob	eq_ref	PRIMARY	PRIMARY	4	ac_anchor.AC_ID

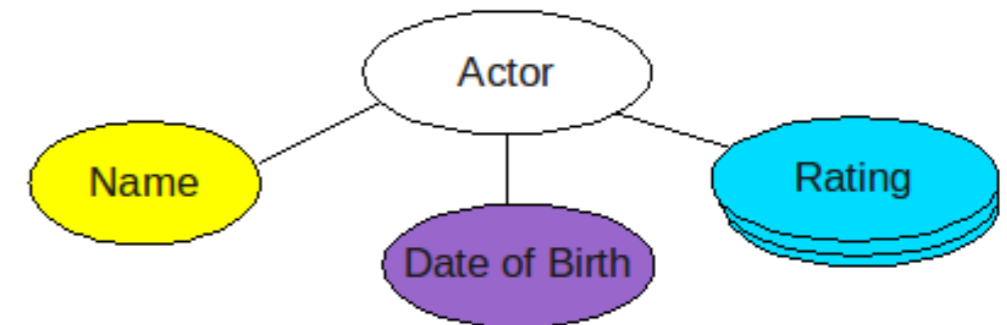
# Table elimination benchmark



- Not a lot of gains with lookups on primary key
- For DBT-3 data (no blobs, no historized data):



- With historized data: 2-3-4x benefit (depending on how much history)



- With table elimination, you can:
  - Do normalization on optional/historic data
  - Create a view with LEFT JOINS that presents denormalized view of the data
  - Use this view and get something like “index only” scans when you're accessing only some attributes



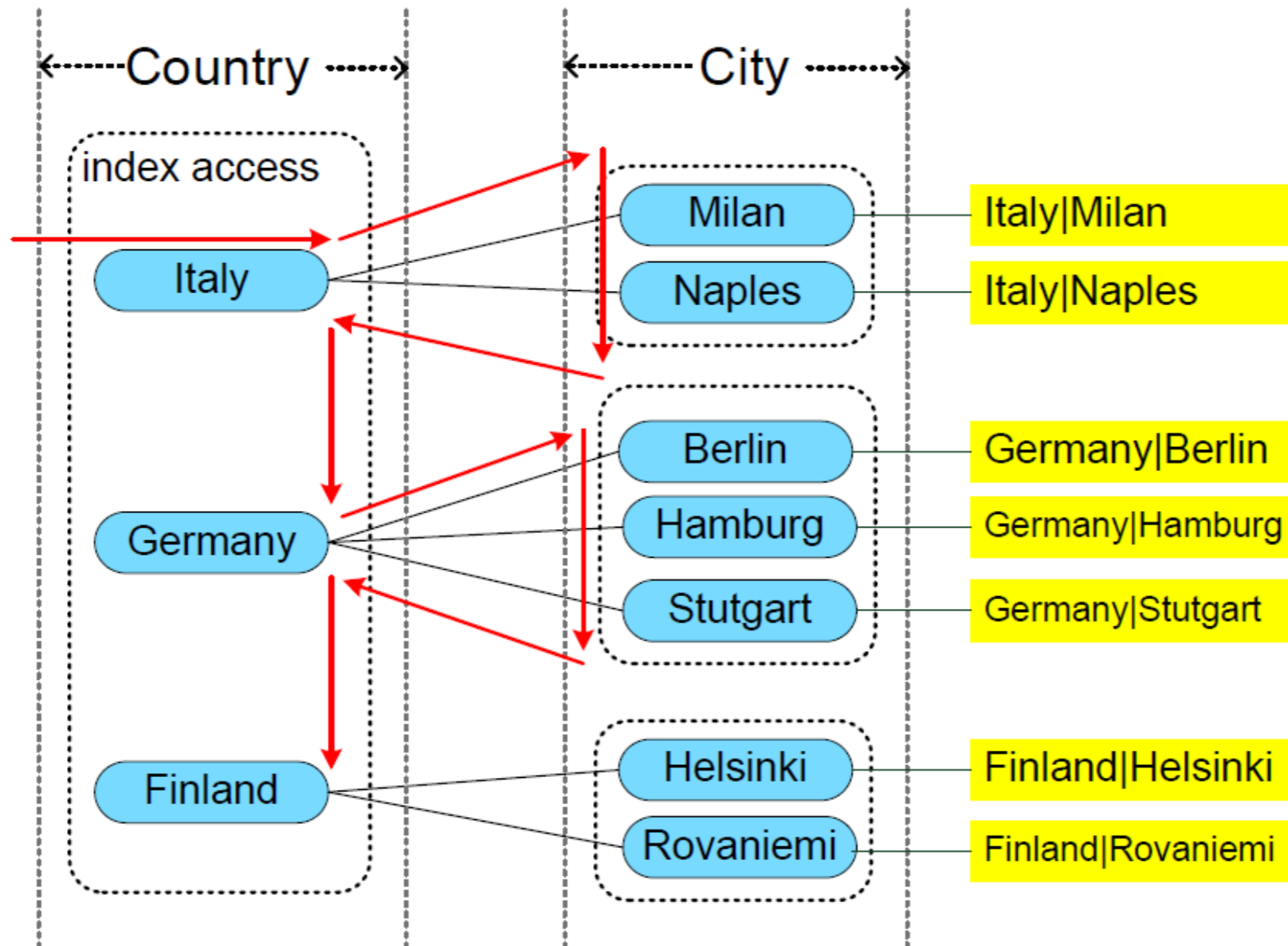
- Table elimination (MariaDB 5.1)
- **Batched Key Access (MariaDB 5.3)**
  - Join buffering now works with outer joins
- Index Condition Pushdown (MariaDB 5.3)
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Predicate caching
  - FROM subquery optimizations

# Batched Key Access – idea

- Background: nested loops join execution

```
mysql> select * from Country, City
      where Country.Continent='Europe' and City.CountryCode=Country.code;
```

table	type	possible_keys	key	key_len	ref	rows	Extra
Country	ref	PRIMARY,Continent	Continent	1	const	37	Using index condition
City	ref	CountryCode	CountryCode	3	Country.Code	18	

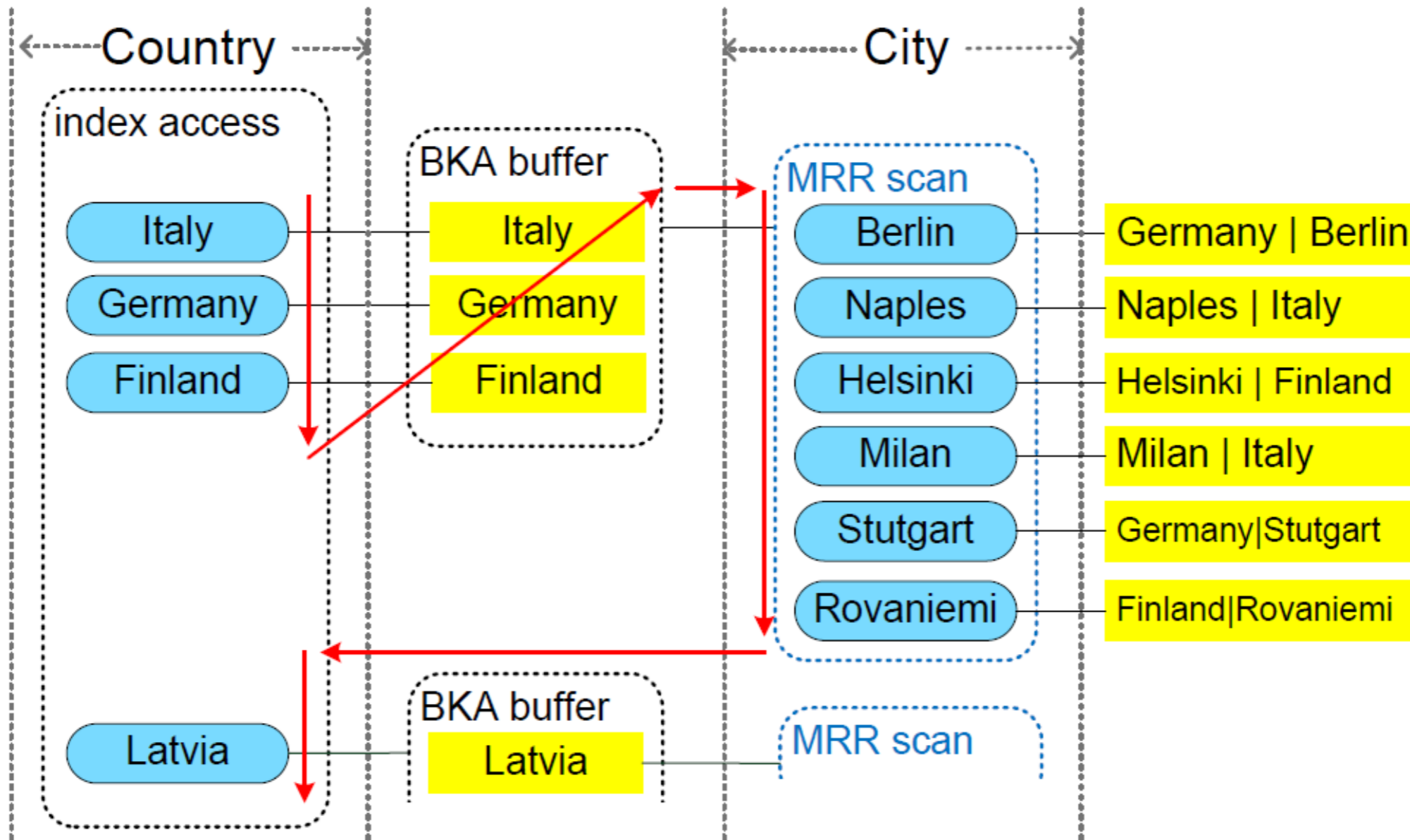


# Batched Key Access – idea

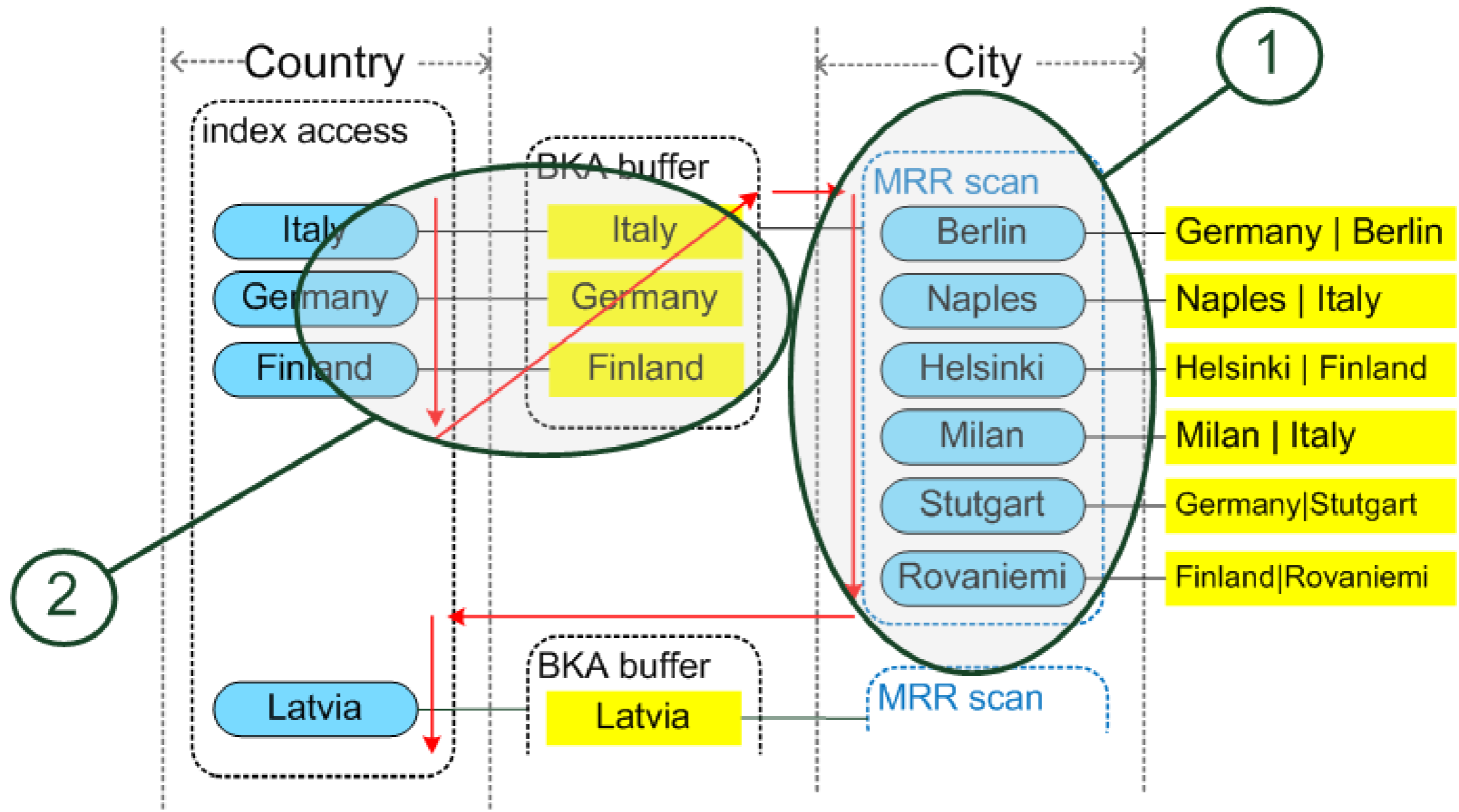
- Batched Key access execution

```
mysql> select * from City, Country
      where Country.Continent='Europe' and City.CountryCode=Country.code;
```

table	type	possible_keys	key	key_len	ref	rows	Extra
Country	ref	PRIMARY,Continent	Continent	1	const	37	Using index condition
City	ref	CountryCode	CountryCode	3	Country.Code	18	Using join buffer

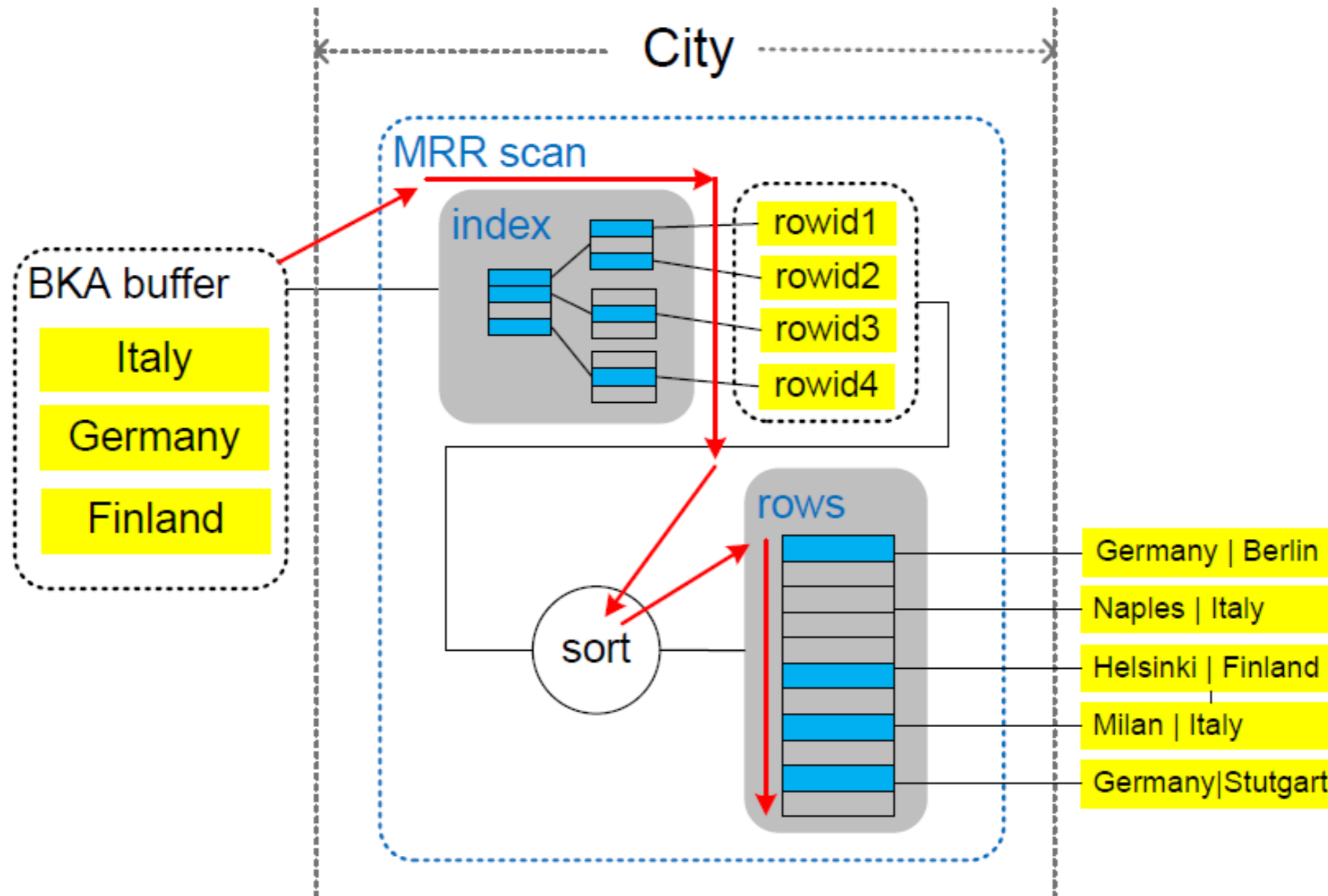


# Why use Batched Key Access



- 1. Storage engine can optimize record reads when scanning many ranges at once
- 2. Buffering allows not to make the same lookup multiple times

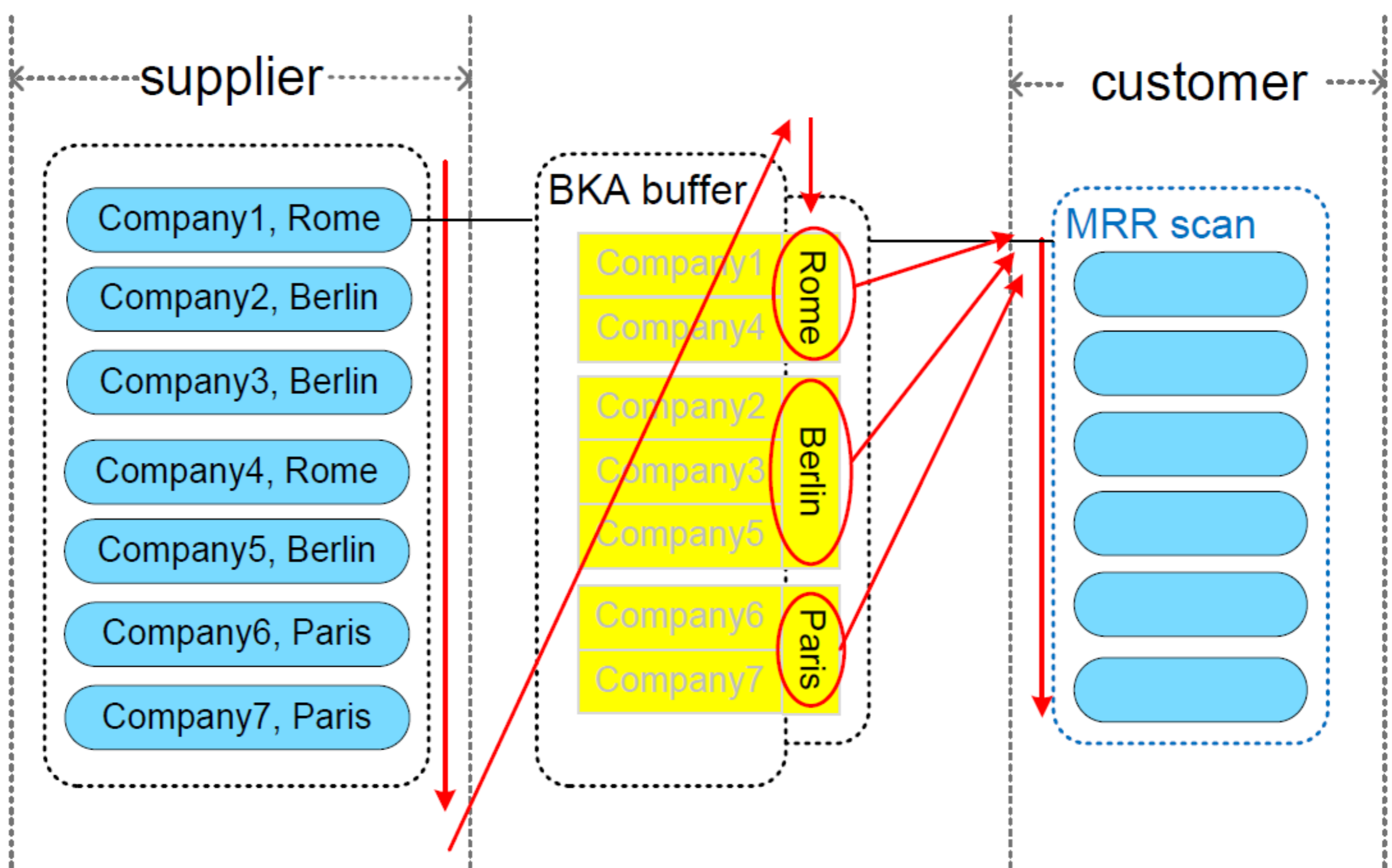
# BKA advantage #1



- InnoDB/MyISAM/Maria read table rows in disk order
- NDB Cluster reduces #roundtrips

# BKA Advantage #2

- Buffering allows not to make the same lookup multiple times  
`select * from supplier, customer  
where supplier.city=customer.city and ...`



# Batched Key Access properties



- Initial implementation done at Sun/MySQL for MySQL 6.0
- MariaDB:
  - Backport from MySQL 6.0 to MariaDB 5.3
  - Infamous InnoDB's + ICP/BKA bugs fixed
- Benchmark
  - DBT-3 scale=10, Query #3.1 (enumerates about 6M rows) , IO-bound

```
select l_orderkey, sum(l_extendedprice * (1-l_discount)) as revenue,
       o_orderdate, o_shippriority
from customer, orders, lineitem
where c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey and
      l_orderkey = o_orderkey and
      o_orderdate >= '1994-07-01' and o_orderdate < '1995-01-01' and
      l_shipdate > '1995-01-14'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
limit 5;
```

No BKA	1 hour 4 min 37 sec	1 x
BKA, 1M buffer	10 min 25 sec	6.2 x
BKA, 10M buffer	6 min 16 sec	10.31 x

- DBT-3, scale=10, Query#3.2, IO-bound load

```
explain
select l_orderkey, sum(l_extendedprice * (1-l_discount)) as
revenue,
       o_orderdate, o_shippriority
from customer, orders, lineitem
where c_nationkey = 12 and c_custkey = o_custkey and
       l_orderkey = o_orderkey and l_shipdate > '1995-01-14'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
limit 5;
```

No BKA	2 hours 16 min 4 sec	1 x
BKA, 1M buffer	2 min 41 sec	50.71 x
BKA, 10M buffer	2 min 36 sec	52.33 x

- General observation from CPU-bound runs
  - 2x-10x improvement

# Join buffering and outer joins



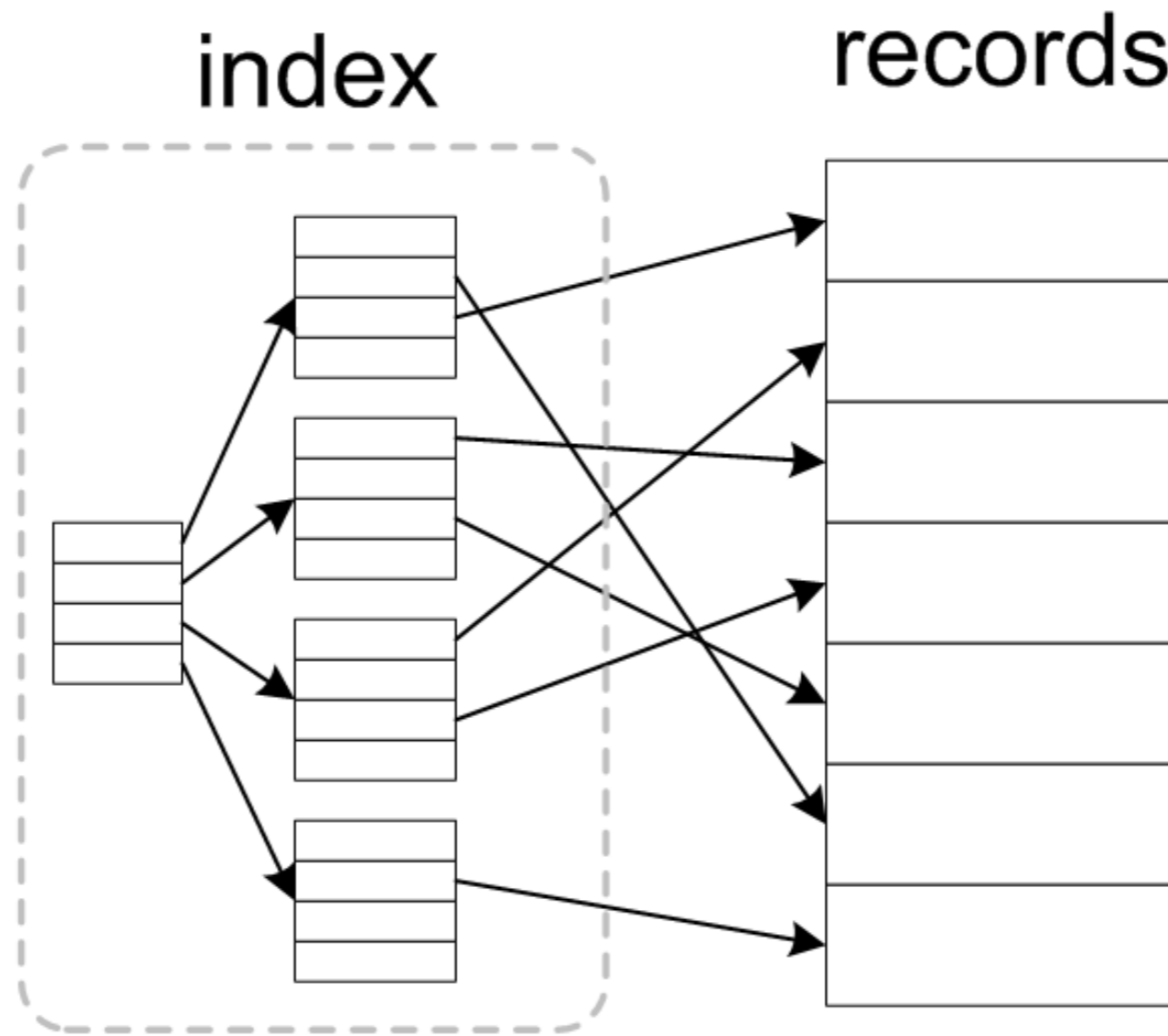
- It is not possible to use join buffering for outer joins in MySQL 5.x
- It was a low-hanging fruit when you have BKA code
- So it's there in MariaDB 5.3
  - Outer joins will use BKA when using key access
  - And will use join buffering with when not using key access



- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
  - Join buffering now works with outer joins
- **Index Condition Pushdown (MariaDB 5.3)**
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Subquery value caching
  - FROM subquery optimizations

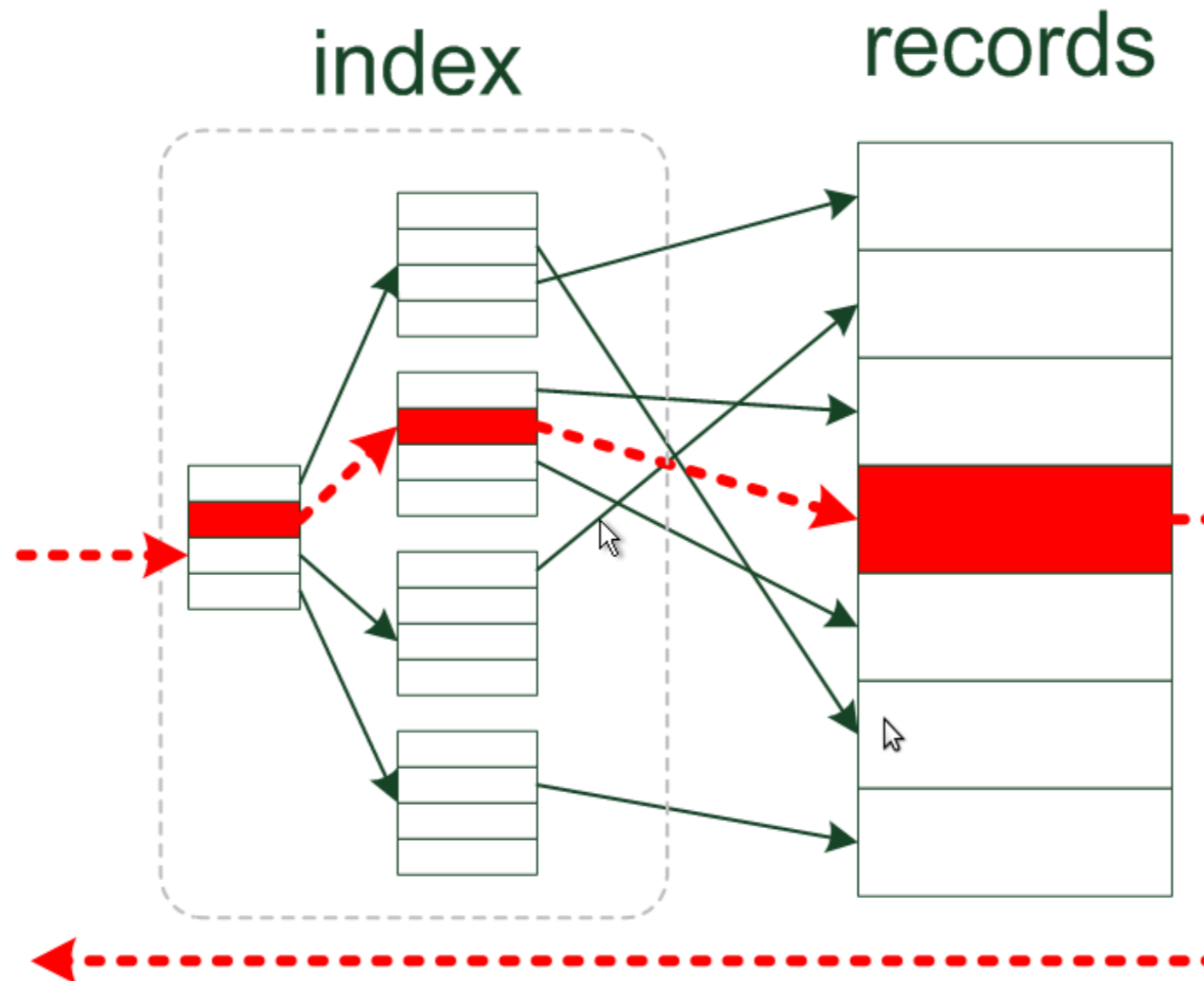
# Index Condition Pushdown

- The idea:  
*When using index-based access method, check condition on index columns before fetching the table record*



# Index Condition Pushdown

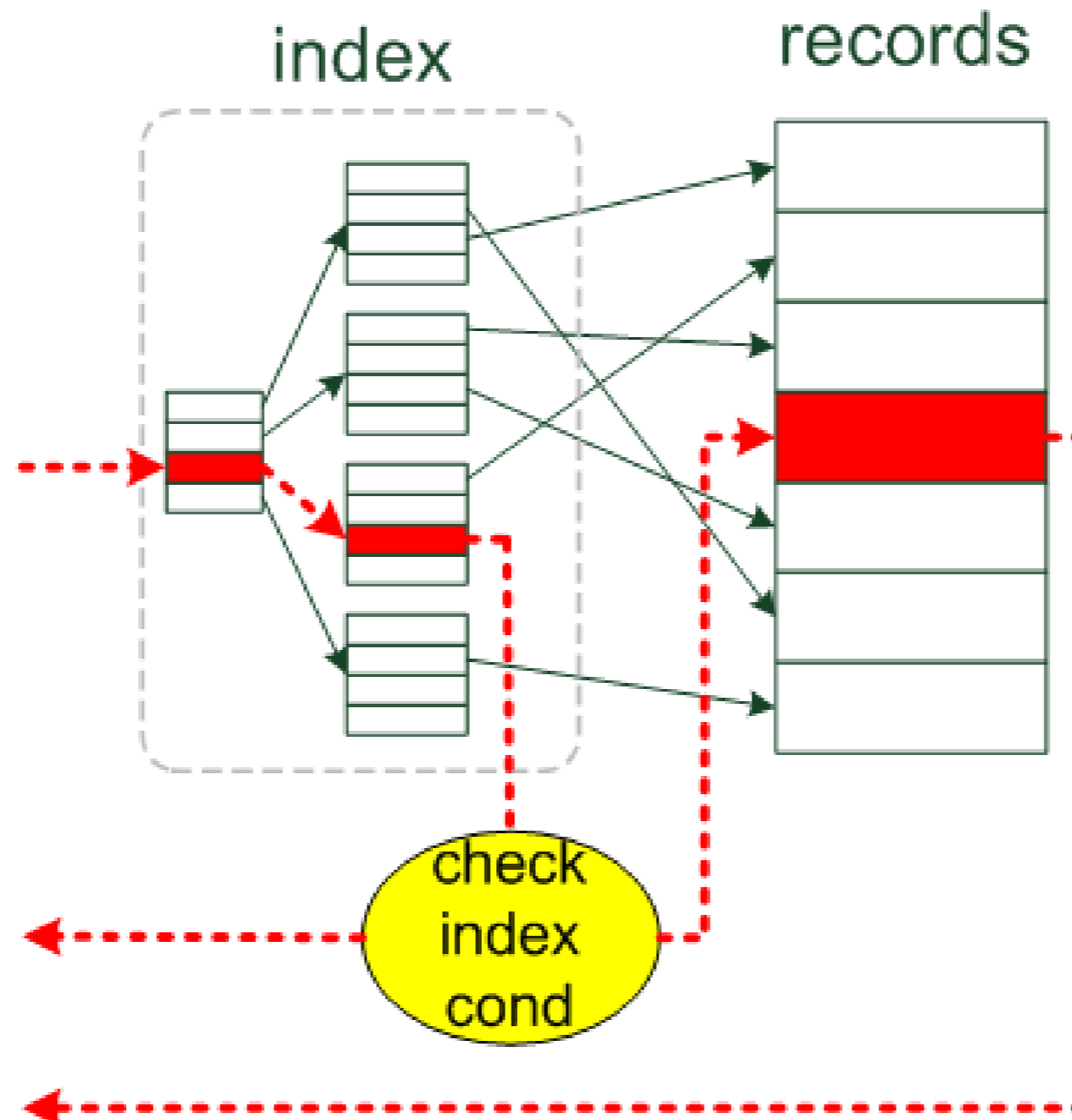
- The idea:  
*When using index-based access method, check condition on index columns before fetching the table record*



# Index Condition Pushdown

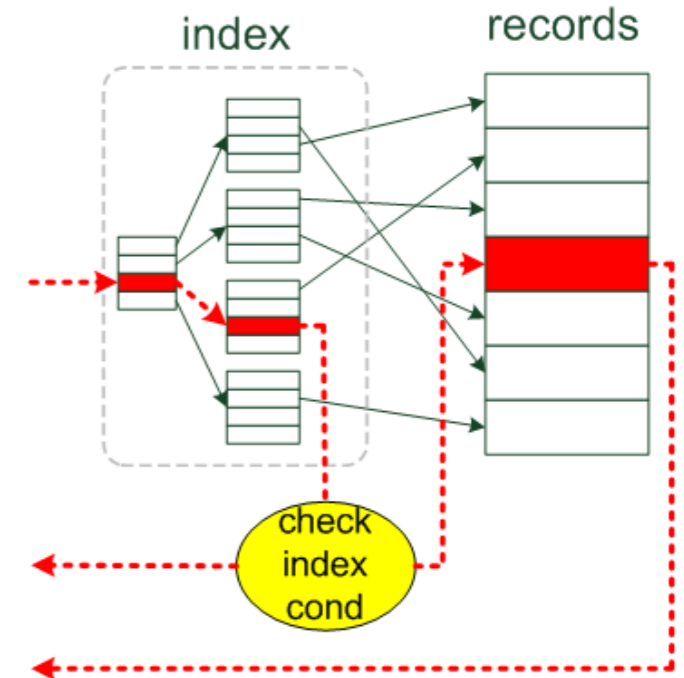
- The idea:

*When using index-based access method, check condition on index columns before fetching the table record*



# Index Condition Pushdown

- Works with any index-based access method: range, ref, eq\_ref, BKA
- Original implementation: MySQL 6.0
  - Infamous for bugs in Index Condition Pushdown + InnoDB
- MariaDB
  - 6.0 code backported to MariaDB 5.3
  - Problems with innodb are believed to be fixed
    - If you're a long timer and willing to check:  
SET engine\_condition\_pushdown=on|off;  
no longer controls Index Condition Pushdown, use  
SET @@optimizer\_switch= 'index\_condition\_pushdown=on|off'  
instead



- DBT-3 data, scale=1

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

```
select count(*) from lineitem
where
```

```
l_shipdate between '1993-01-01' and '1993-01-01' + interval 30 day and
datediff(l_receiptdate, l_shipdate) > 25 and
l_quantity > 40;
```

table	type	possible_keys	key	key_len	ref	rows	Extra
lineitem	range	s_r	s_r	4	NULL	152064	Using where

table	type	possible_keys	key	key_len	ref	rows	Extra
lineitem	range	s_r	s_r	4	NULL	152064	Using index condition; Using where

- Results

- Cold caches: from 5 min down to 1 min
- Hot caches: from 0.19 sec down to 0.07 sec



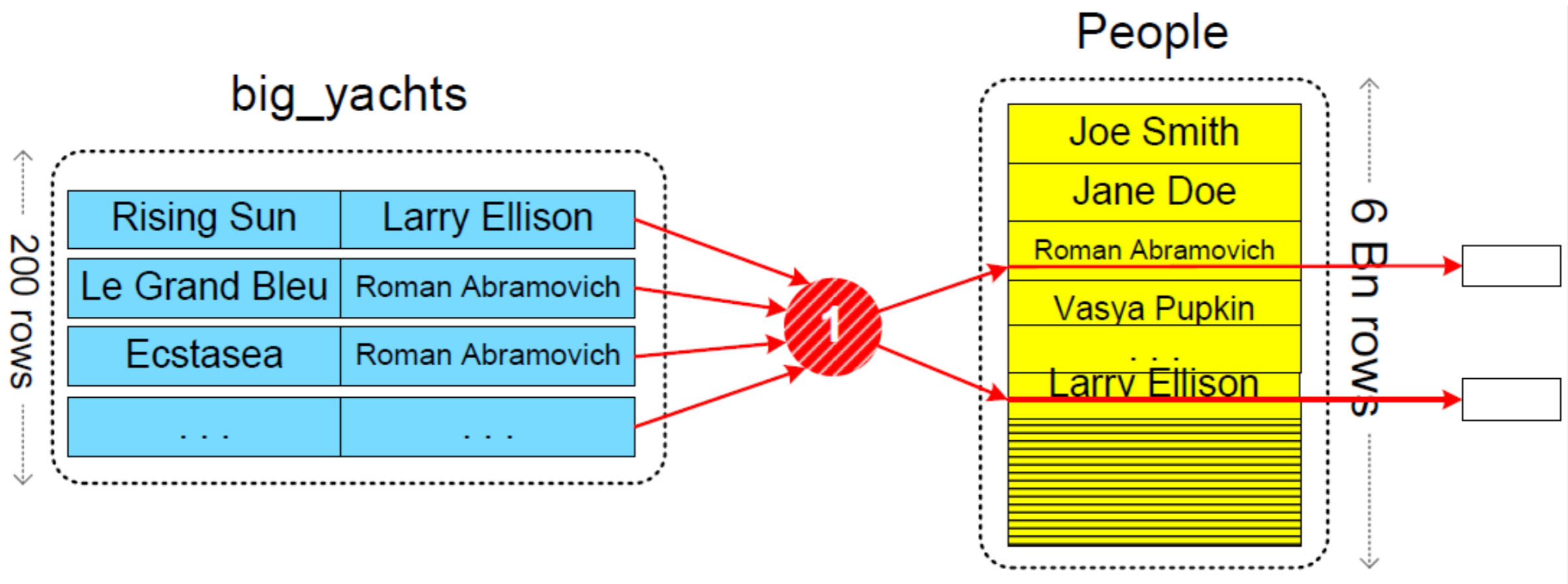
- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- **Subquery optimizations (MariaDB 5.3)**
  - **Backport of 6.0 features**
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Subquery value caching
  - FROM subquery optimizations



# Semi-join subqueries: MySQL 6.0/MariaDB

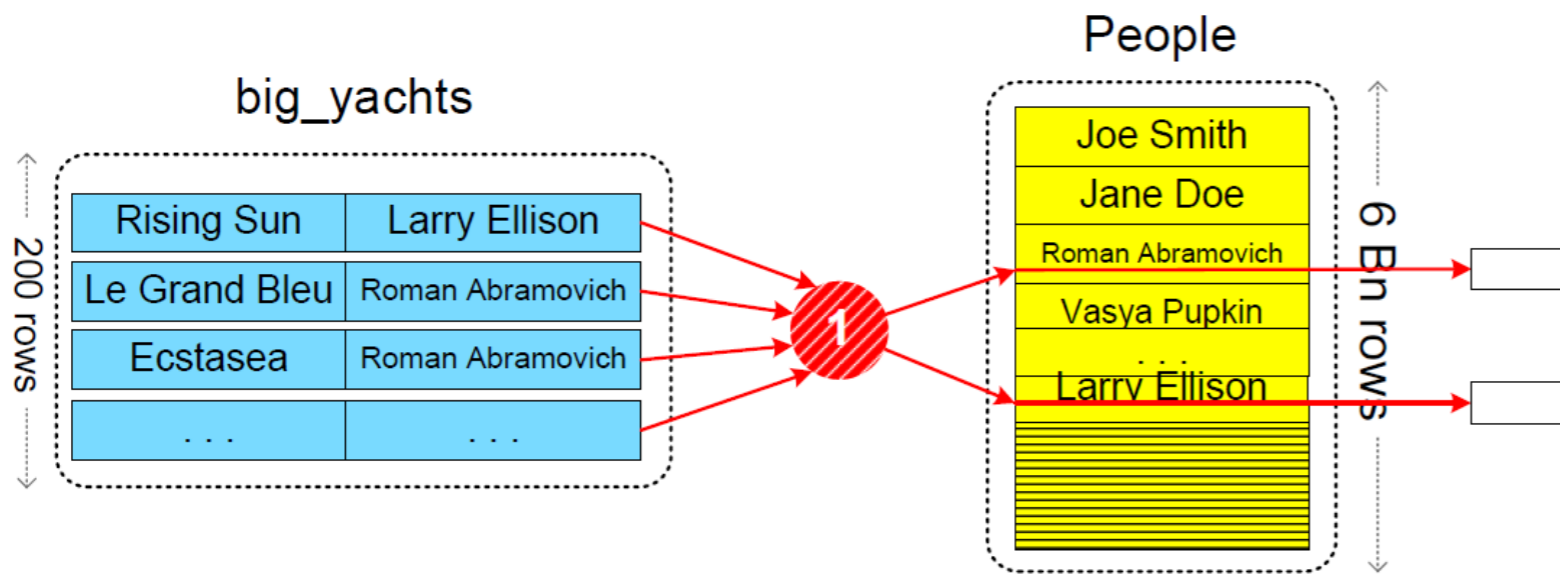
- MySQL 6.0/MariaDB: inside-to-outside execution is supported

```
SELECT * FROM people WHERE name IN (SELECT owner FROM big_yachts WHERE cost > 50M)
```



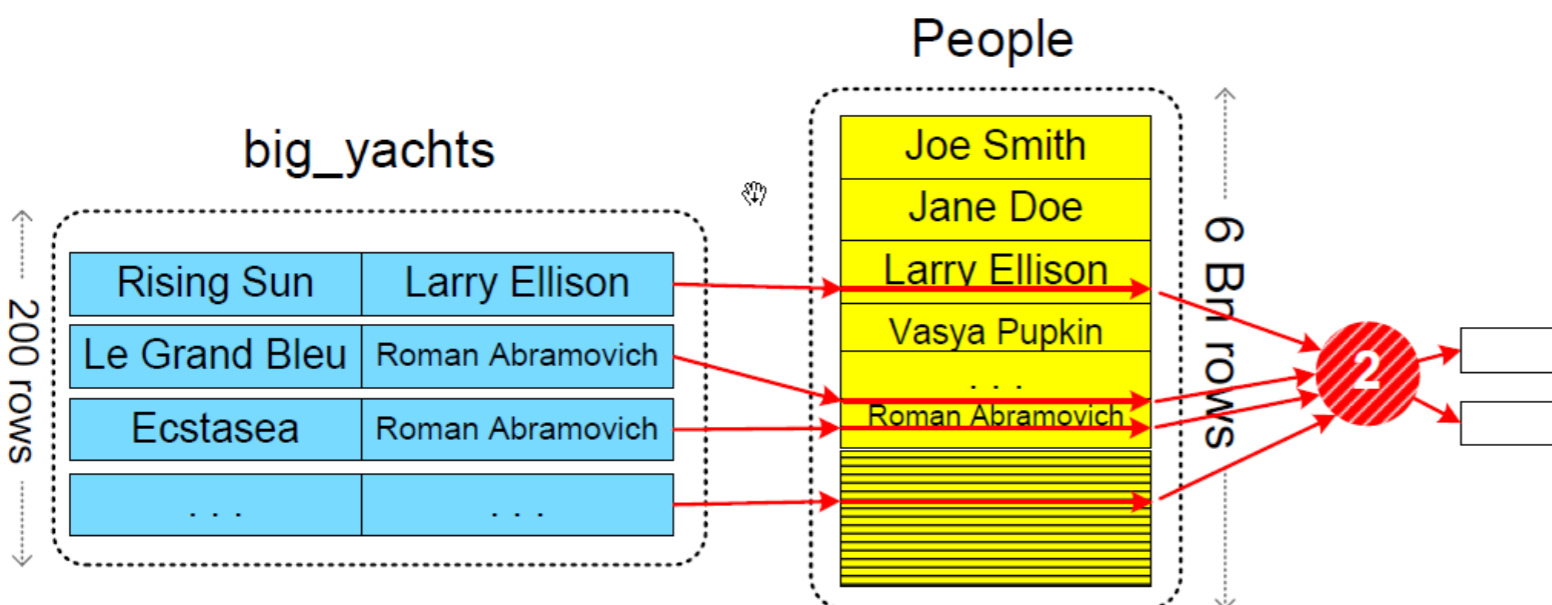


```
SELECT * FROM people WHERE name IN (SELECT owner FROM big_yachts WHERE cost > 50M)
```



1. Remove duplicates before joining:

- Materialization
- Loose scan

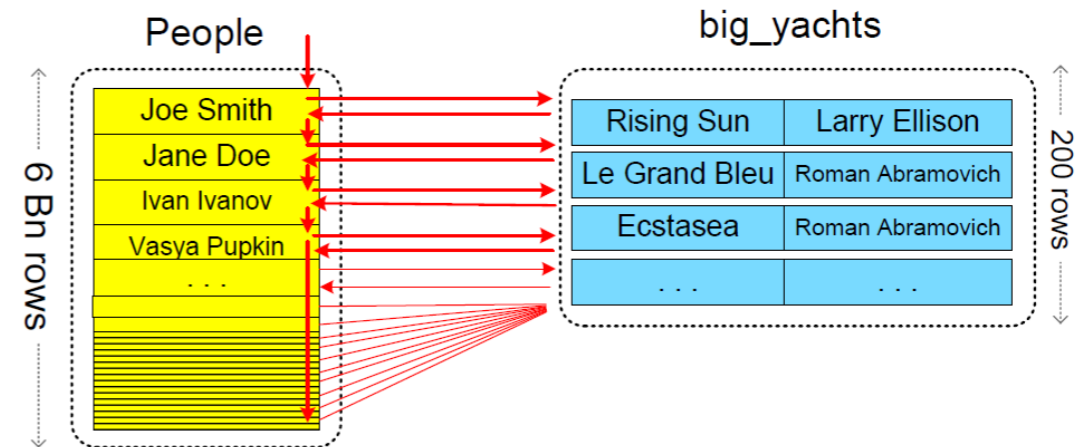


2. Remove duplicates after joining:

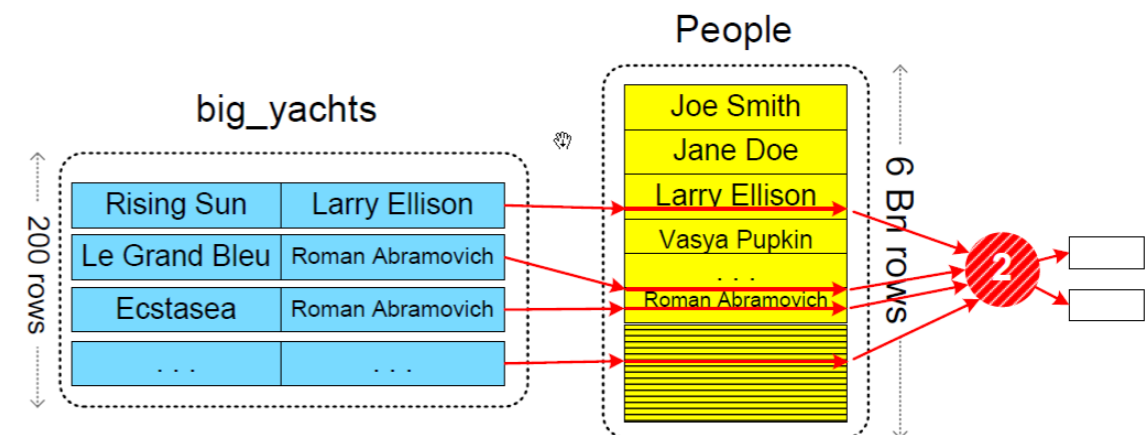
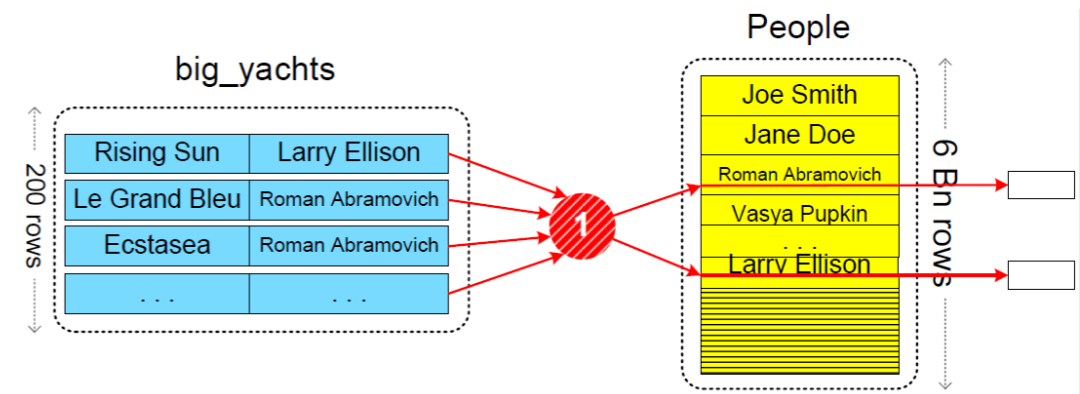
- Duplicate Weedout



- WHERE ... IN (SELECT w/o grouping) subqueries are handled with a comprehensive set of optimizations



	Outer-to-inner	Inner-to-outer
First Match	+	
Loose Scan	+	
Materialization	+	+
Duplicate Weedout	+	+





Quoting previous results with MySQL 6.0:

- MySQL bugs/customer issues that are easily repeatable
  - Found 10 subquery cases
  - Taking PostgreSQL's speed as 1.0:

	No 6.0 optimizations	Materialization	Semi-join
	67285.714	34.286	1.429
	59490.000	780.000	n/a
	9.477	2.109	0.004
	151.429	206.667	0.476
	1360.000	490.000	10.000
	670.453	0.264	1.052
	16.364	0.455	0.182
	10.000	0.625	n/a
	5648.649	3.243	0.270
	962.500	1.500	n/a
Medians:	<b>816.48</b>	<b>2.68</b>	<b>0.48</b>

Run parameters

- MySQL 6.0.3
- PostgreSQL 8.3.0
- No tuning, all default settings
- Small query population
- => numbers only show order of magnitude

- MariaDB's time should be MIN(column2, column3), or MAX(column2, column3)
  - Can't re-run because data is Sun-private



- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - **NULL-aware materialization**
  - Materialization-scan for grouping queries
  - Subquery value caching
  - FROM subquery optimizations



- With subqueries, SQL's semantics for NULL value is “unknown”.
- This gives us:

`NULL IN (SELECT ... FROM something) = NULL`

`NULL IN (SELECT ... FROM no_data) = FALSE`

`'foo' IN (SELECT ... FROM 

'foo'
'bar'
NULL

) = TRUE`

`'foo' IN (SELECT ... FROM 

'foo'
'baz'
NULL

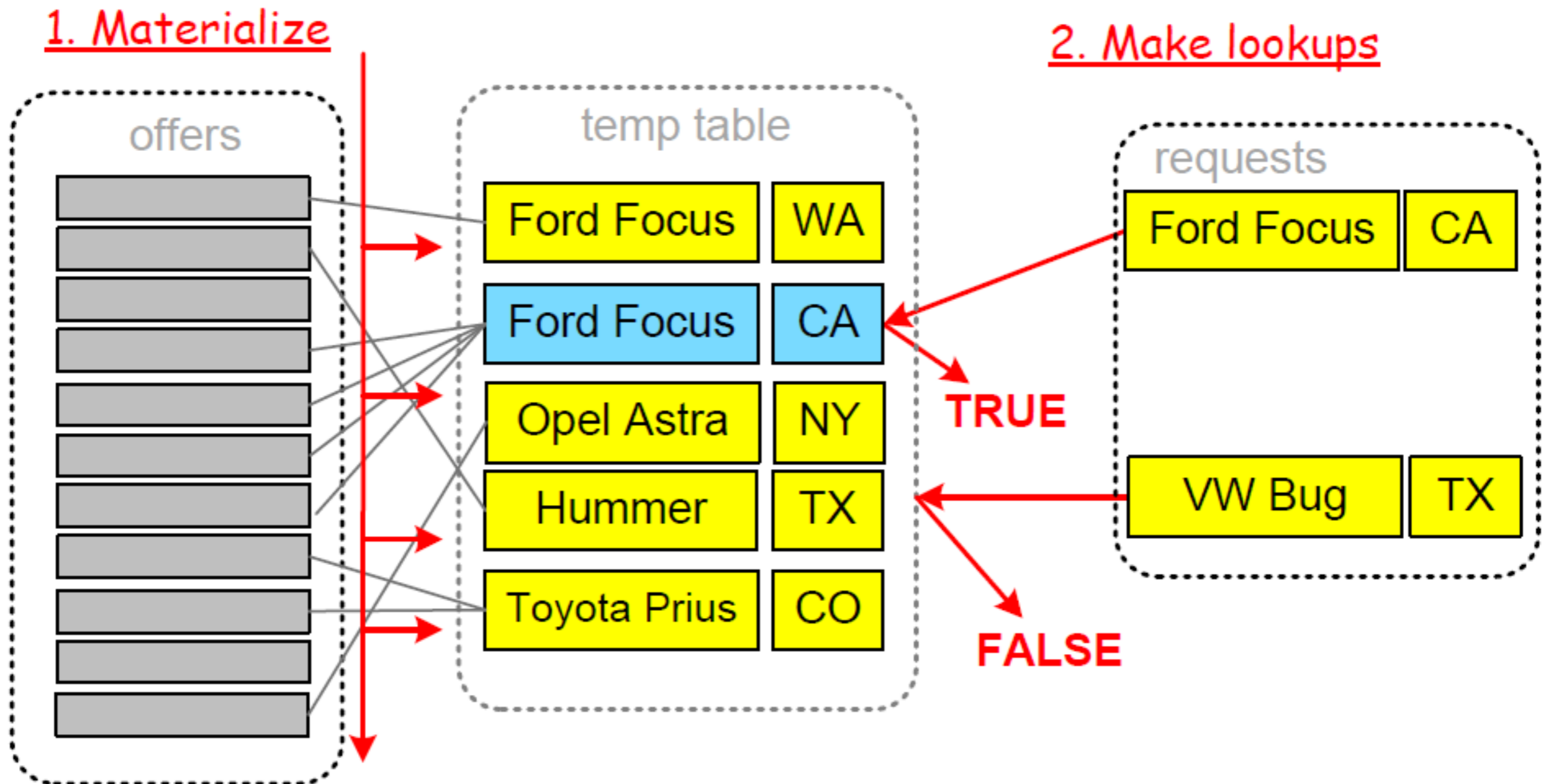
) = NULL`

- An important special case where we don't care whether result is NULL or FALSE:  
`... WHERE smth IN (SELECT ...) AND|OR ...`
- In all other cases, including `WHERE ... NOT IN(SELECT ...)` : we do care and need correct evaluation

# NULL-aware materialization (1)

- Materialization strategy without NULLs:

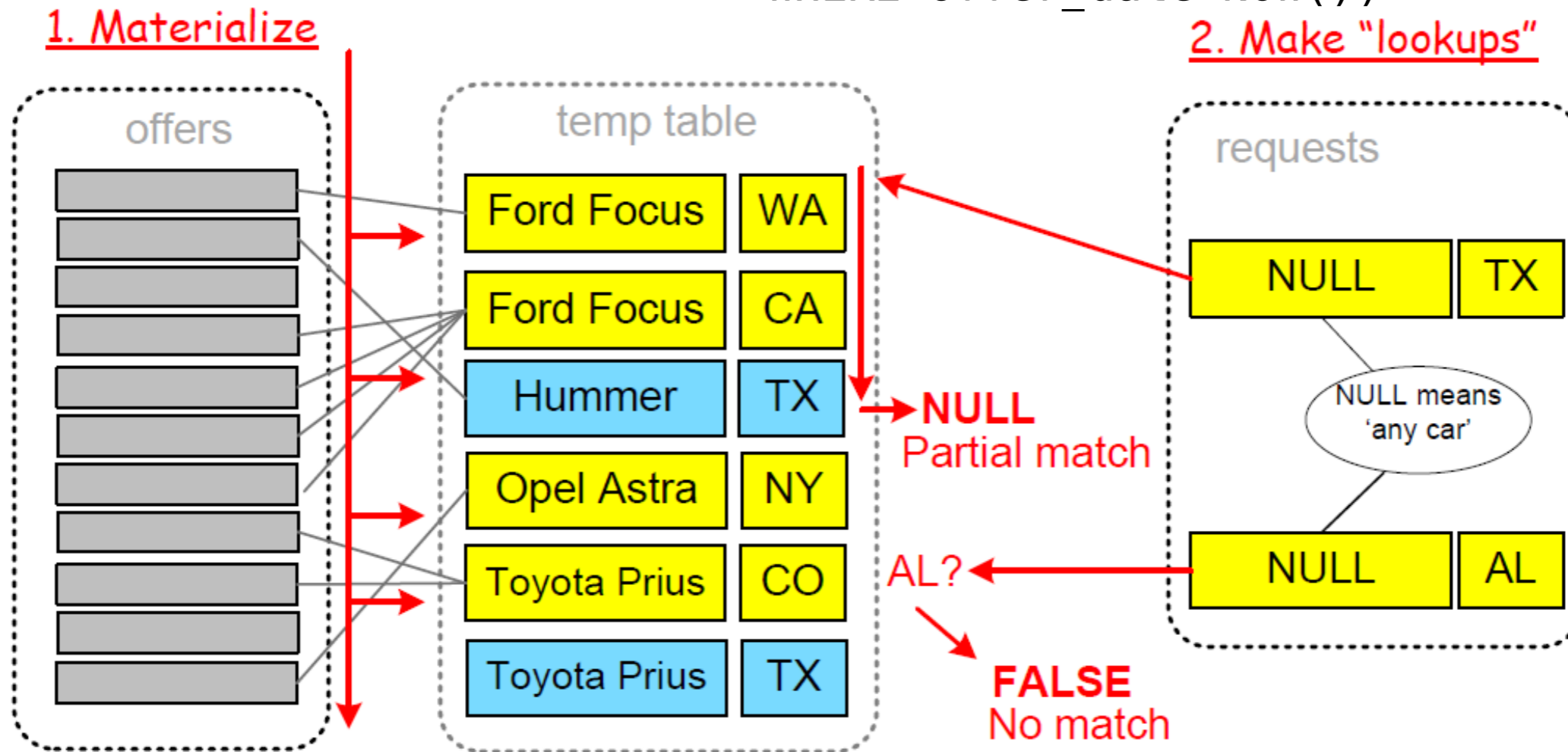
```
SELECT ... FROM requests  
WHERE (car_model, state) NOT IN (SELECT car_model, state FROM offers  
WHERE offer_date=NOW())
```



# NULL-aware materialization (3)

- NULLs outside

```
SELECT ... FROM requests
WHERE (car_model, state) NOT IN (SELECT car_model, state FROM offers
WHERE offer_date=NOW())
```

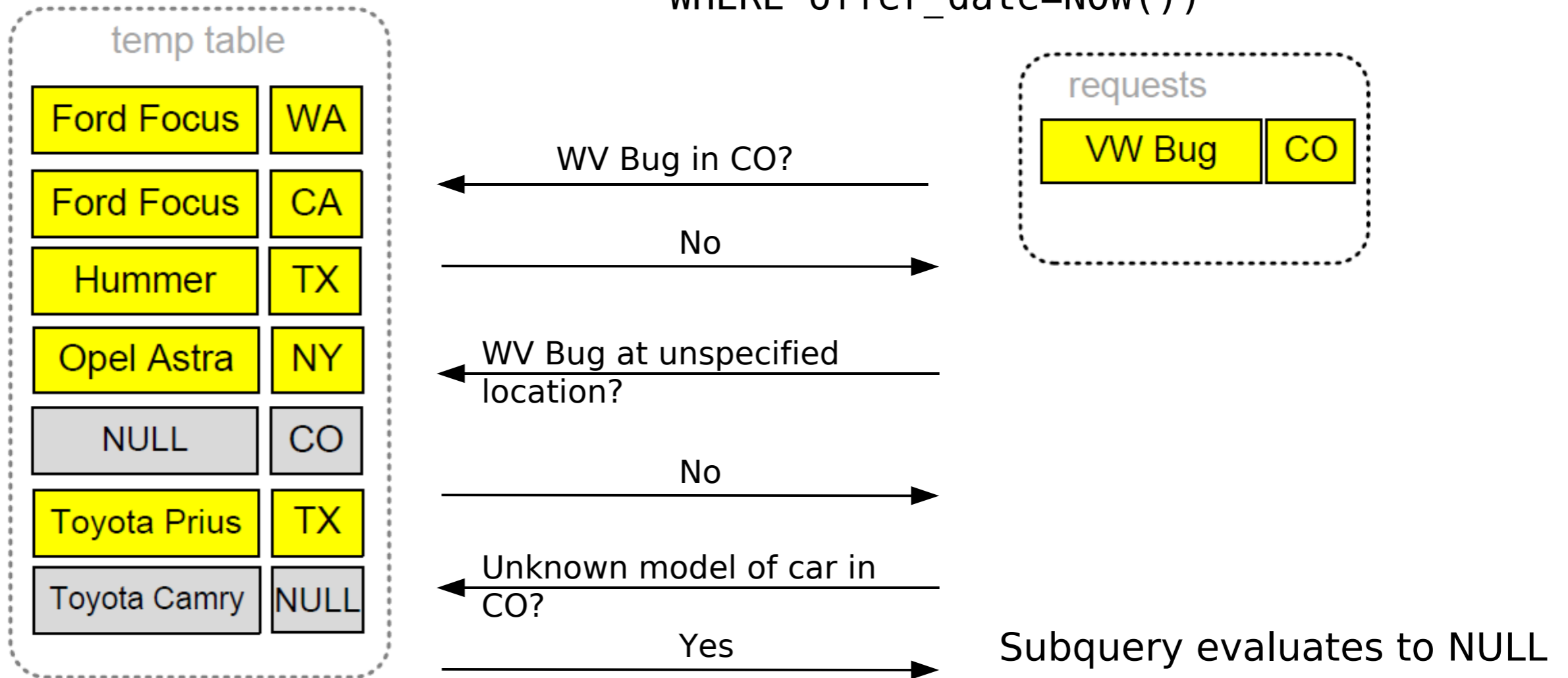


- If we get NULL on the left of IN, we must search for partial match
  - Generally, hash indexes used for materialization table do not support partial matching, so one has to do full scan on the temptable

# NULL-aware materialization (2)

- Now, NULLs inside the subquery

```
SELECT ... FROM requests
WHERE (car_model, state) NOT IN (SELECT car_model, state FROM offers
WHERE offer_date=NOW())
```



- NULLs on the inside of subquery also require partial match searches



- MariaDB 5.3 will include advanced materialization/lookup strategy that will
  - Support handling of NULLs inside/outside of the subqueries
  - Has no overhead over regular materialization if columns are NULLable but no NULLs were encountered
  - No significant degradation for big fractions of NULLs on either side

```
select count(*) from customer
where (c_custkey, c_pref_nationkey_05, c_pref_brand_05) NOT IN
(select o_custkey, s_nationkey, p_brand
 from orders, supplier, part, lineitem
 where l_orderkey = o_orderkey and
       l_suppkey = s_suppkey and
       l_partkey = p_partkey and
       p_retailprice < 1200 and
       l_shipdate >= '1996-04-01' and l_shipdate < '1996-04-05' and
       o_orderdate >= '1996-04-01' and o_orderdate < '1996-04-05');
```

	IN->EXISTS	Materialization	Times faster
DBT3 scale=1, cold run	34	30	1.13
DBT3 scale=1, hot run	3.55	0.27	13.15
DBT3 scale=10, cold run	532	712	0.75
DBT3 scale=10, hot run	40	2	20



- Experiments show that
  - There are cases when Materialization is much better
  - There are cases where subquery re-execution (old approach) is much better
    - Roughly speaking: when one of the tables already has an index that materialization would create
- Will need to work on a cost-based decision between materialization and subquery re-execution



- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - **Materialization-scan for grouping queries**
  - Subquery value caching
  - FROM subquery optimizations

- Detected while analyzing user cases

```
select * from customer
where
  customer_id in (select customer_id
                  from orders
                  group by customer_id
                  having sum(cost) > 1M)
```

Big table

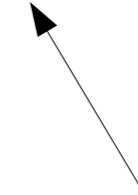
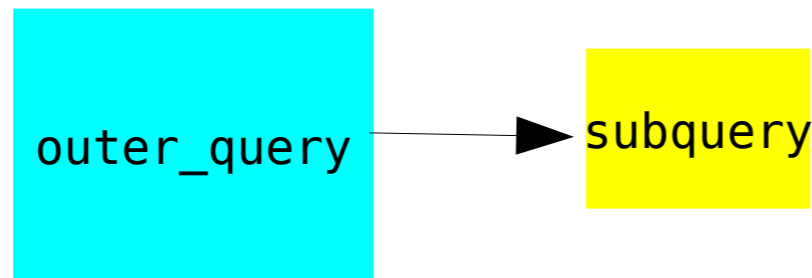
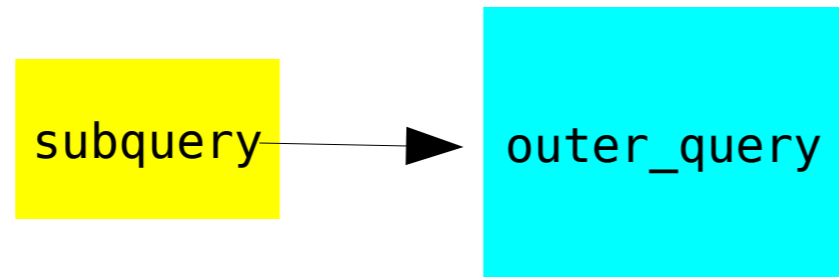


Table with a few groups

- MySQL 5.x/6.0:



- MariaDB 5.3





- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - **Subquery value caching**
  - FROM subquery optimizations



- The idea: take any correlated subquery

```
select ...  
from outer_table_1, outer_table_2  
where/having  
    ... (select from inner_table  
        where outer_table_1.column1=... and  
            ...  
            inner_column < outer_table_2.column2)
```

- Subquery value is a function of (outer\_table\_1.column1, outer\_table\_2.column2).
- Generally, the subquery is evaluated many times for different values of outer\_table\_1.column1 and outer\_table\_2.column2
  - But values do repeat
- Create/use a cache  
(outer\_table\_1.column1, outer\_table\_2.column2) → subquery\_result

# What exactly is being cached



- The cache will hold scalars for all kinds of subqueries:

```
EXISTS (SELECT * FROM promotions WHERE  
        state= outer_table.state OR  
        industry= outer_table.industry)
```

(outer\_table.state, outer\_table.industry) -> TRUE/FALSE

---

```
SELECT * FROM brands as outer_tbl  
WHERE (SELECT avg(discount) FROM sales WHERE brand=  
outer_tbl.brandname) > 0.25
```

outer\_tbl.brandname -> avg(discount)

---

```
outer_table.brand_name IN (SELECT brand_name FROM promotions WHERE  
state=outer_table.state)
```

(outer\_table.brand\_name, outer\_table.state) -> TRUE/FALSE/NULL

- HEAP temporary table is used as hashtable
- The cache is of infinite size
- Convert to MyISAM/Maria if HEAP table overflows
  - MyISAM lookups are slow but still faster than subquery re-executions
- Use of cache can be controlled from SQL with  
`SET optimizer_switch='subquery_cache=on|off'`



- DBT-3 data, scale 1.
- Query #17, total revenue on low-quantity orders

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where
  p_partkey = l_partkey and
  p_brand = 'Brand#42' and p_container = 'JUMBO BAG' and
  l_quantity < (select 0.2 * avg(l_quantity) from lineitem
                 where l_partkey = p_partkey) ;
```

2.3x speedup, from 3.28 to 1.34 sec, cache hit ratio: 96%

- Own query: customers with balance near top in their nation

```
select count(*) from customer
where c_acctbal > 0.8 * (select max(c_acctbal)
                        from customer C
                        where C.c_nationkey=customer.c_nationkey
                        group by c_nationkey);
```

2640 times speedup, from 43 min to 0.69 sec.

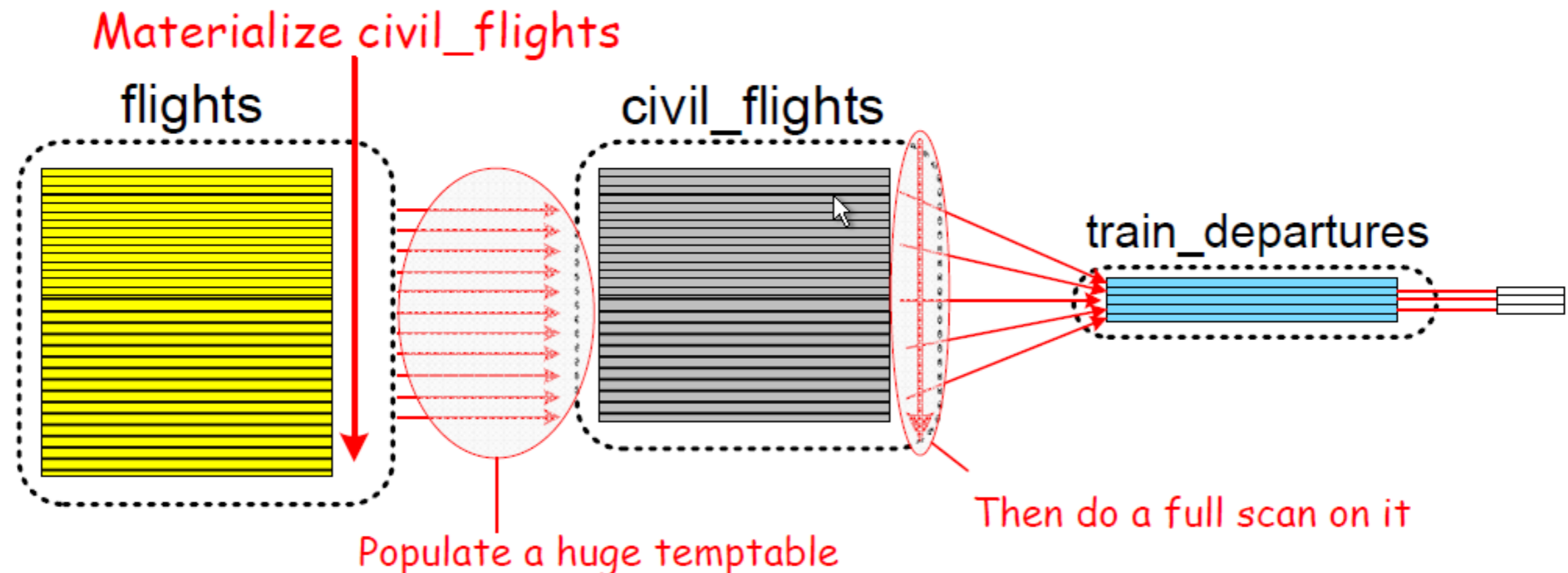


- Table elimination (MariaDB 5.1)
- Batched Key Access (MariaDB 5.3)
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 features
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Subquery value caching
  - **FROM subquery optimizations**

# FROM subquery optimizations

- Long-time MySQL problem:

```
select * from  
  (select * from flights where flight_type <> 'Military') as civil_flight,  
  train_departure  
where  
  train_departure.hour = flight.arrival_hour;
```

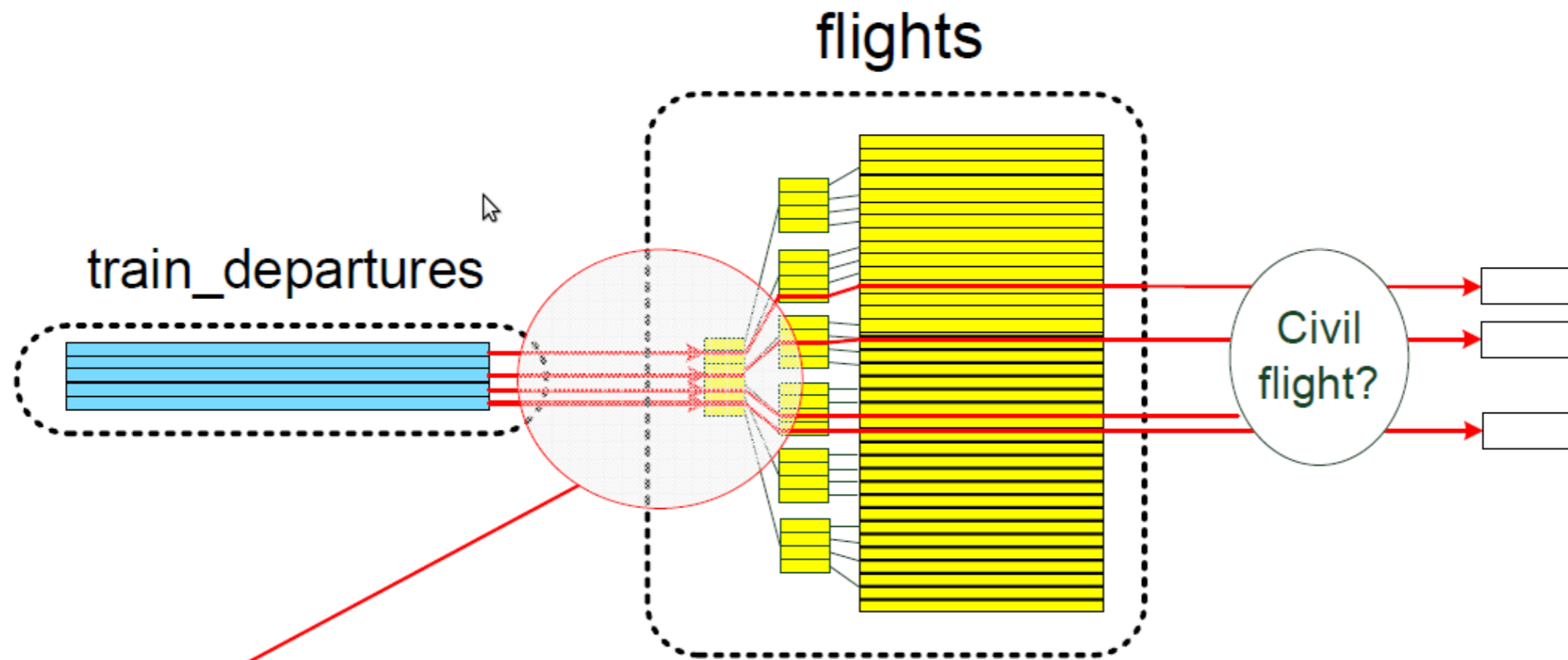


- Worse: EXPLAIN does materialization too, so will be very slow, too.
- Can work around with `CREATE VIEW civil_flight`

# FROM subquery optimizations(2)

- Initial work started at Sun as WL#3485, never finished
- Backported and feature-finished in MariaDB 5.3

```
select * from  
  (select * from flights where flight_type <> 'Military') as civil_flight,  
  train_departure  
where  
  train_departure.hour = flight.arrival_hour;
```



We don't have to materialize, so can access to original flights table, including its indexes

- DBT-3, scale=1, "hot", query#8

```
select o_year, sum(case when nation = 'BRAZIL'
                        then volume else 0 end) / sum(volume) as mkt_share
from
(
  select substr(o_orderdate, 1, 4) as o_year,
         l_extendedprice * (1 - l_discount) as volume,
         n2.n_name as nation
  from part, supplier, lineitem, orders, customer,
       nation n1, nation n2, region
  where p_partkey = l_partkey and s_suppkey = l_suppkey and
        l_orderkey = o_orderkey and o_custkey = c_custkey and
        c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and
        r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and
        o_orderdate between '1995-01-01' and '1996-12-31' and
        p_type = 'ECONOMY ANODIZED STEEL'
) as all_nations
group by o_year
order by o_year;
```

## EXPLAIN execution time

Before	46.91 sec
After	0.01 sec

## Query execution time

Before	47.06 sec
After	10.46 sec

4.5x speedup

- DBT-3, scale=1, "hot", query#9

```
select nation, o_year, sum(amount) as sum_profit
from
(
  select n_name as nation, substr(o_orderdate, 1, 4) as o_year,
  l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity
  as amount
  from part, supplier, lineitem, partsupp, orders, nation
  where s_suppkey = l_suppkey and ps_suppkey = l_suppkey and
  ps_partkey = l_partkey and p_partkey = l_partkey and
  o_orderkey = l_orderkey and s_nationkey = n_nationkey and
  p_name like '%green%'
) as profit
group by nation, o_year
order by nation, o_year desc
limit 5;
```

## EXPLAIN execution time

Before	2 min 26 sec
After	0.02 sec

## Query execution time

Before	2 min 36 sec
After	53 sec

2.9x speedup



- MariaDB 5.1

- Table elimination



Highly normalized data,  
outer joins

- Maria DB 5.2

- -

- MariaDB 5.3

- Batched Key Access
- Join buffering now works with outer joins
- Index Condition Pushdown
- Subquery optimizations (MariaDB 5.3)
  - Backport of 6.0 optimizations
  - NULL-aware materialization
  - Materialization-scan for grouping queries
  - Subquery value caching
  - FROM subquery optimizations



\* Big joins,  
\* Big range scans on  
composite indexes  
\* Subqueries



Q & A