MySQL® Conference & Expo

Understanding and control of MySQL Query Optimizer

traditional and novel tools and techniques

Sergey Petrunya
Sun Microsystems
2009

Query optimizer 101

def

Query Optimizer is a part of the server that takes a parsed SQL query and produces a query execution plan

- When do I need to care about it?
  - When your query is not fast enough
  - And that's because the server has picked a wrong query execution plan
- Can I make the optimizer pick a better plan?
  - Yes. You can use hints, optimizer settings, rewrite the query, run ANALYZE TABLE, add index(es), etc etc ...
- Required knowledge:

١.Understand the query plan that was picked by the optimizer and what are the other possible plans

٢.Know how to direct the optimizer to the right plan

Optimizer walkthrough - selects

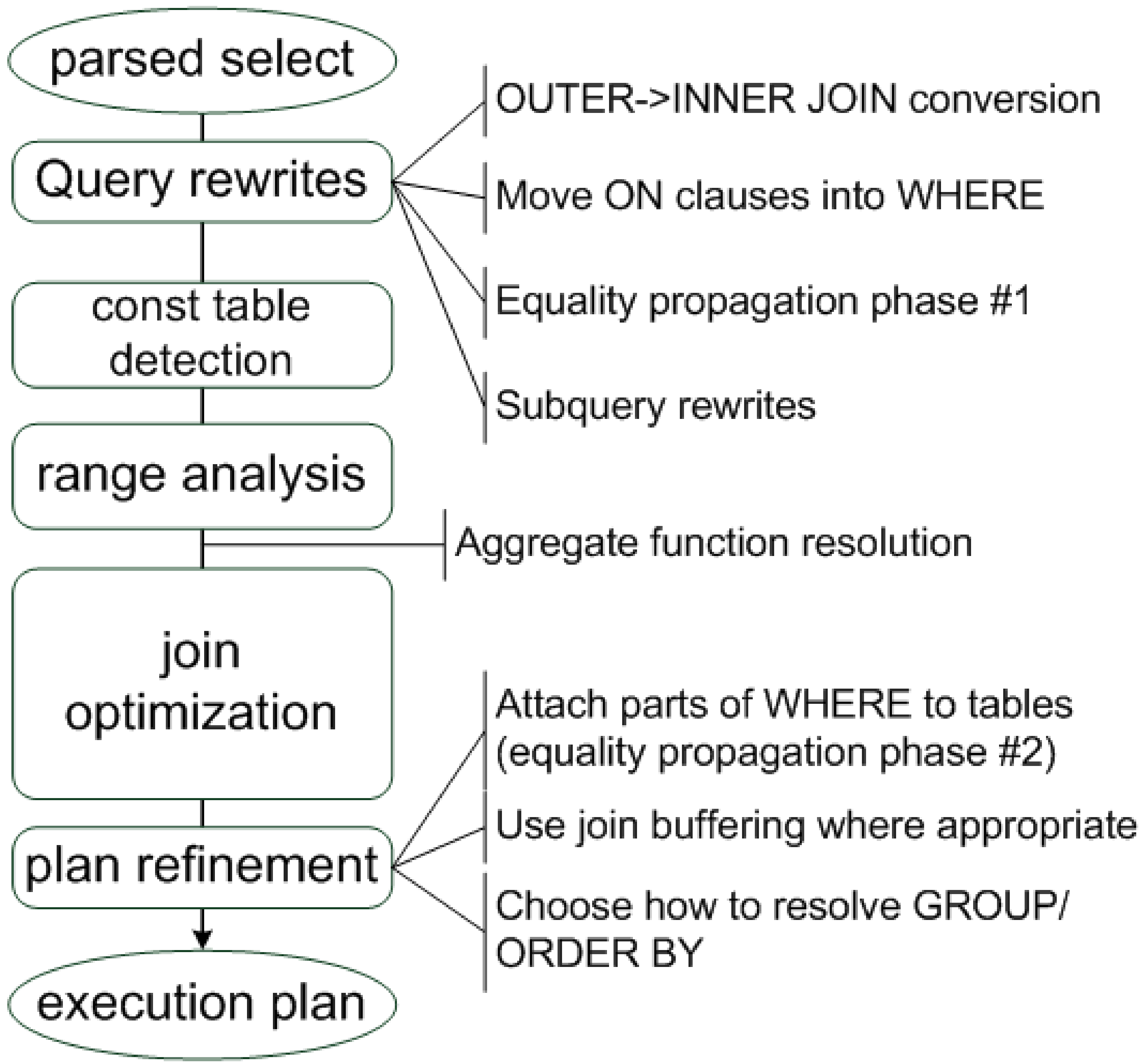
- Biggest optimization unit: a “select”:

```
SELECT select_list
FROM from_clause -- not counting FROM subqueries
WHERE condition -- not counting subqueries
GROUP BY group_list HAVING having_cond
ORDER BY order_list LIMIT m,n
```

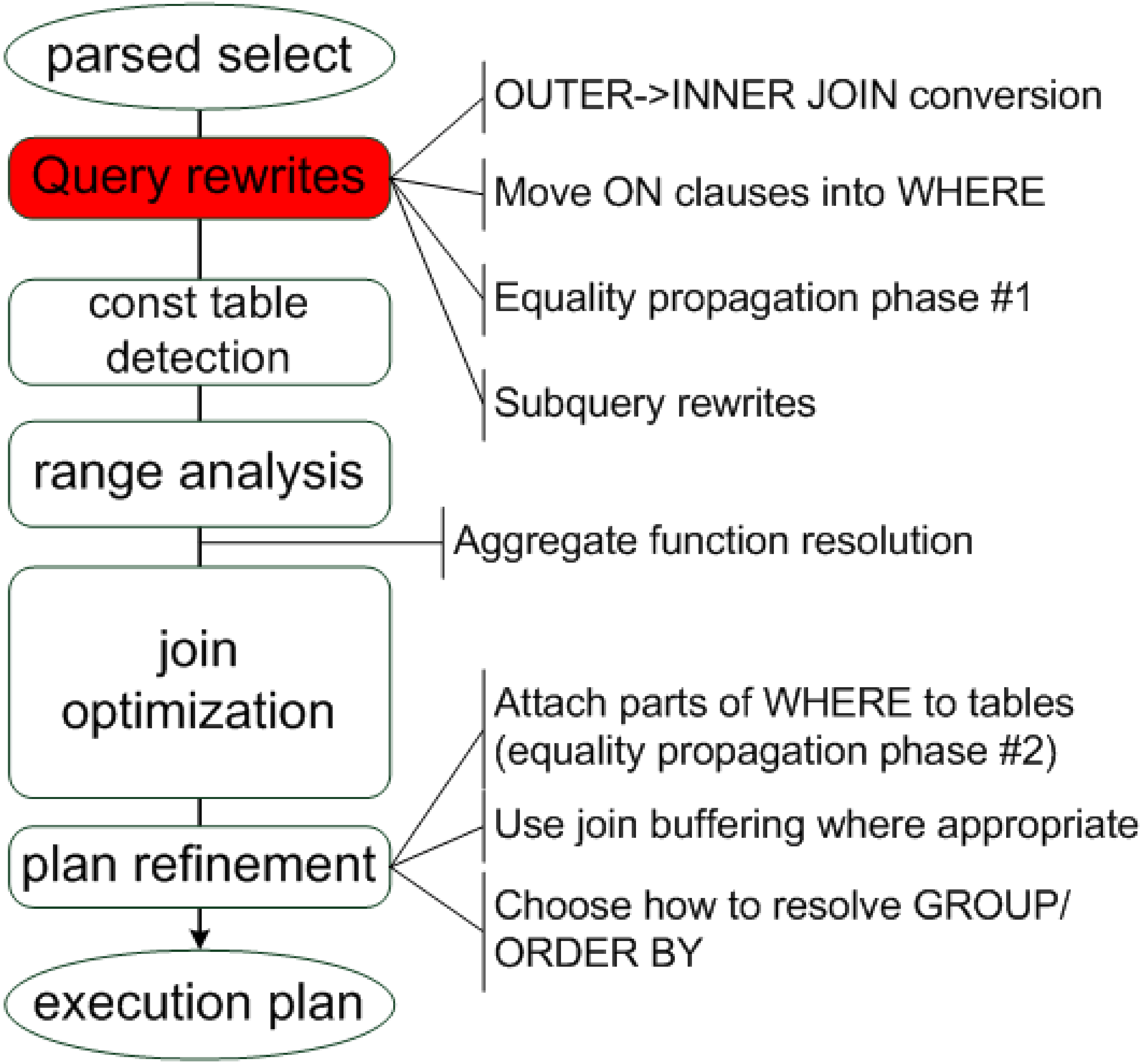
- UNION branches and subqueries are optimized [almost] separately (won't be true for subqueries from 5.1.x)
- How can you see it? EXPLAIN, “id” column:

```
explain select * from t1, t2 where ...
union
select * from t10, t11
where t10.col in (select t20.col from t20 where ...);
+----+--------------------+-------------+------+-...
| id | select_type        | table       | type |
+----+--------------------+-------------+------+-...
|  1 | PRIMARY            | t1          | ALL  |
|  1 | PRIMARY            | t2          | ALL  |
|  2 | UNION              | t10         | ALL  |
|  2 | UNION              | t11         | ALL  |
|  3 | DEPENDENT SUBQUERY | t20         | ALL  |
| NULL | UNION RESULT     | <union1,2>  | ALL  |
+----+--------------------+-------------+------+-...
```

Optimizer walkthrough – select optimization



Select optimization: rewrites



08:55:50 AM

Rewrites: join simplification

- If the WHERE clause is such that it would filter out all NULL-complemented records, outer join is equivalent to inner
- Copy ON clauses into the WHERE
- Can see the conversion in EXPLAIN EXTENDED:

```
mysql> explain extended
  select * from t1 [left] join t2 on t1.col=t2.col
  where t2.col2=1;
+----+-------------+-------+------+-...
| id | select_type | table | type |
+----+-------------+-------+------+-...
|  1 | SIMPLE      | t1    | ALL  |
|  1 | SIMPLE      | t2    | ALL  |
+----+-------------+-------+------+-...

mysql> show warnings;
select ... from `db`.`t1` join `db`.`t2` where
((`db`.`t2`.`col` = `db`.`t1`.`col`) and
(`db`.`t2`.`col2` = 1))
```

- Conclusions
  - If you have an outer join, check if you really need it
  - For inner joins, it doesn't matter if condition is in the WHERE clause or in the ON clause.

08:55:50 AM

Rewrites: equality propagation

- Basic idea:

  col1=const AND col1=col2 → col2=const

- This allows to
  - Infer additional equalities
  - Make expressions like func(col1) or func(col2) constant, evaluate them and use their value in optimization

```
explain extended
select * from t1
where t1.col1=4 and t1.col1=t1.col2 and
      t1.col3 like concat(t1.col2,' %');

. . .

show warnings;
select ... from ... where ((`db`.`t1`.`col1` = 4) and
(`db`.`t1`.`col2` = 4) and
 (`db`.`t1`.`col3` like concat(4,' %')))
```

- Anything to do besides watching it?
  - Check for cross-type comparisons or tricky collation cases
  - This may cause slowdown by generating too many options to consider (alas, one can't turn it off)

Rewrites: subquery conversions

- There are two rewrites:
  - IN->EXISTS rewrite

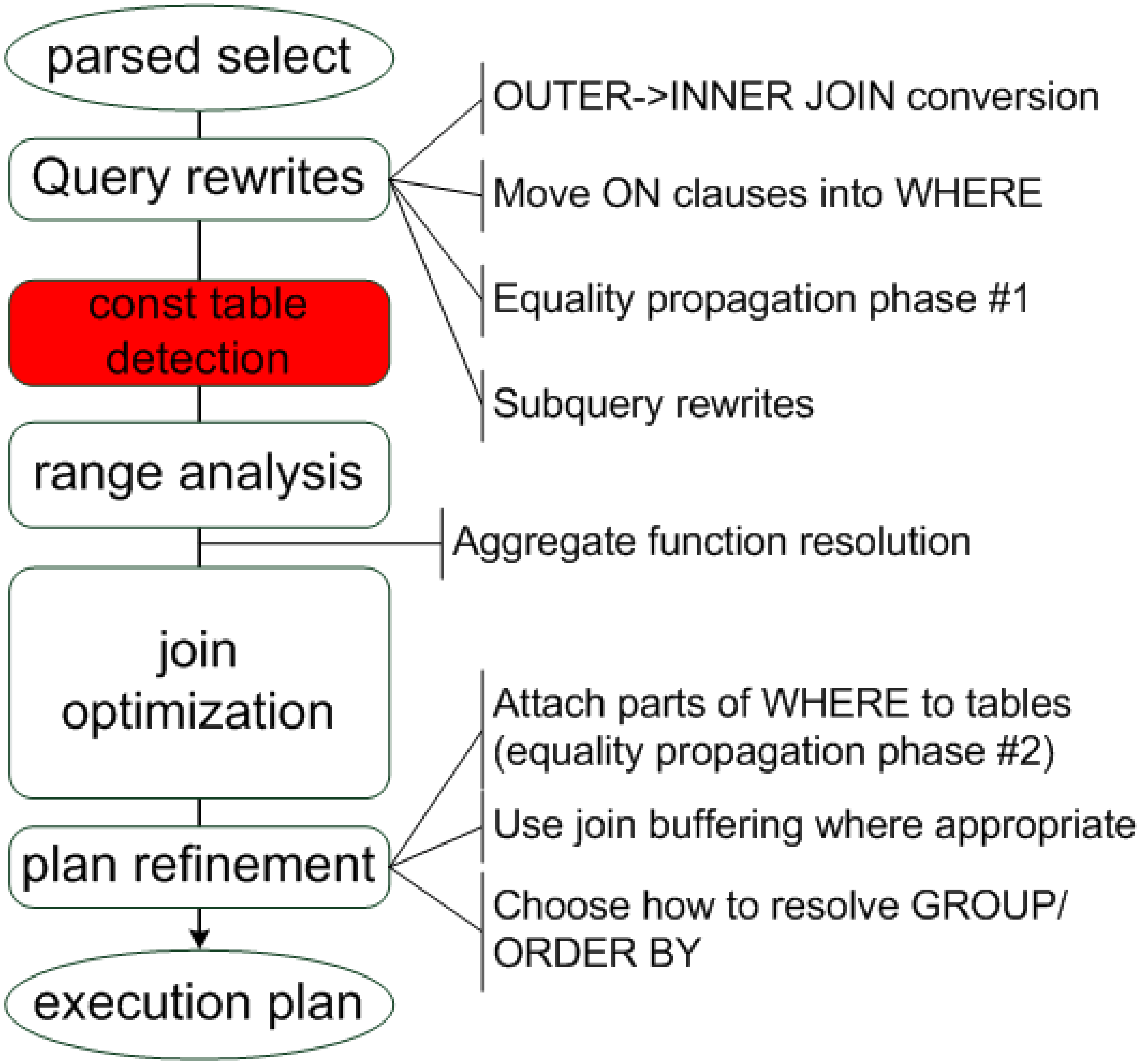
```
x IN (SELECT y  … ) y
EXISTS (SELECT 1 FROM .. WHERE|HAVING x=y)
```

  - MIN/MAX rewrite

x > ANY (SELECT) → x > (SELECT max(...) …)

- No way to turn them off (no code to execute the original forms)
- Lots of changes coming in 5.4.x/6.0
- See last year's talk for more details on current and future behavior
  http://www.mysqlconf.com/mysql2008/public/schedule/detail/595

Select optimization: const table detection



MySQL Conference & Expo

Select optimization: constant table detection

- Constant table is a table that has one of:
  - WHERE/ON contains a clause that will select one row:

    `uniq_key_part1=const AND ... AND uniq_key_partN=const`
  - The storage engine can guarantee that the table has exactly 1 or 0 rows (only MyISAM ones can)
- When a table is found to be constant, all references to its columns are substituted for constants.
- How can one see this?

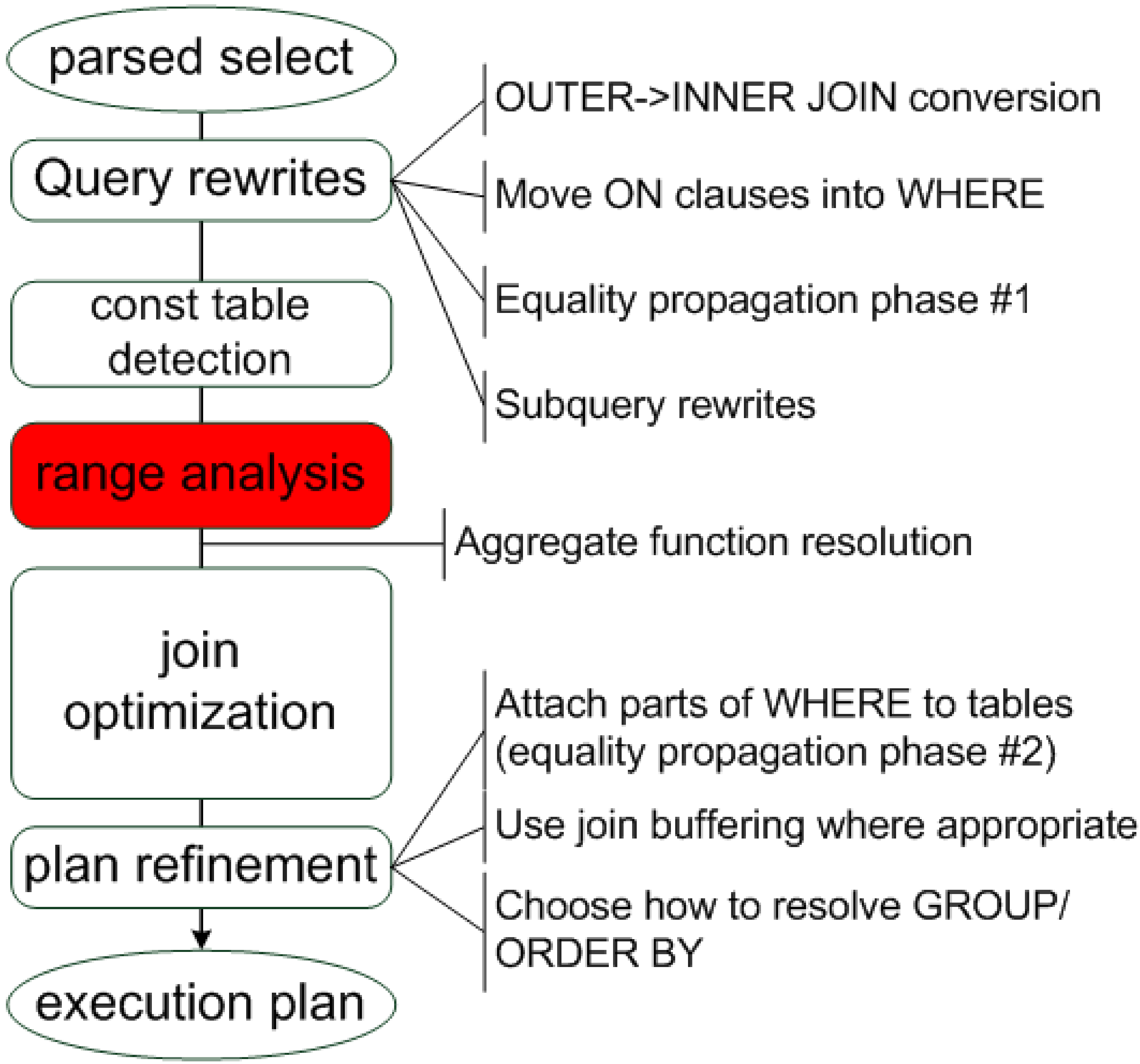
```
explain extended
select * from t1, t2 where t1.pk=1 and t2.col>t1.col;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	t١	const	PRIMARY	PRIMARY	٤	const	١	
١	SIMPLE	t٢	ALL	NULL	NULL	NULL	NULL	١٠	Using where

```
show warnings;
select ... from `db`.`t1` join `db`.`t2` where
((`db`.`t2`.`a` > '1'))
```

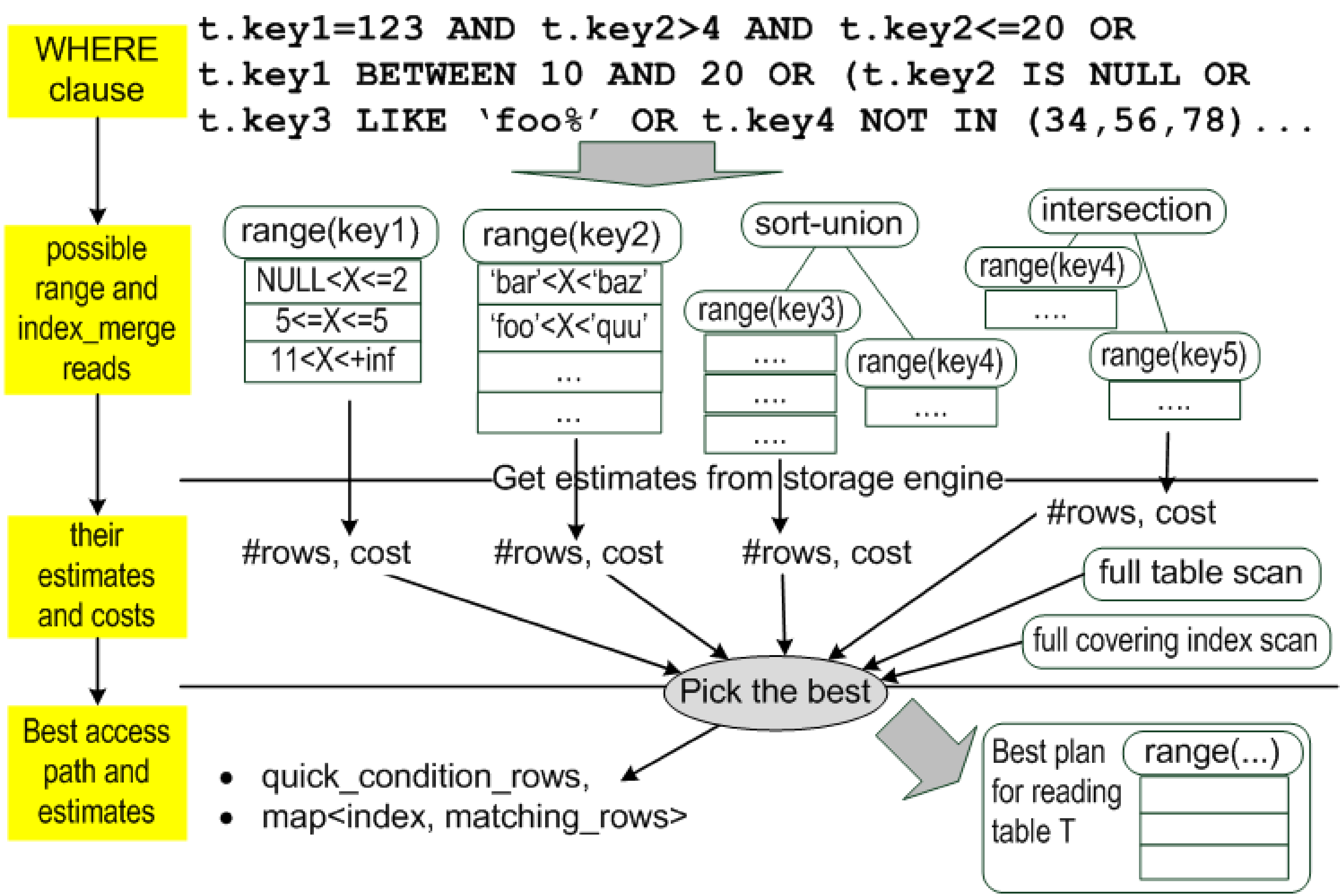
- Conclusions
  - UNIQUE indexes are better than “de-facto unique”
  - One-row “world-constant” tables should have PK or be MyISAM

Select optimization: range analysis



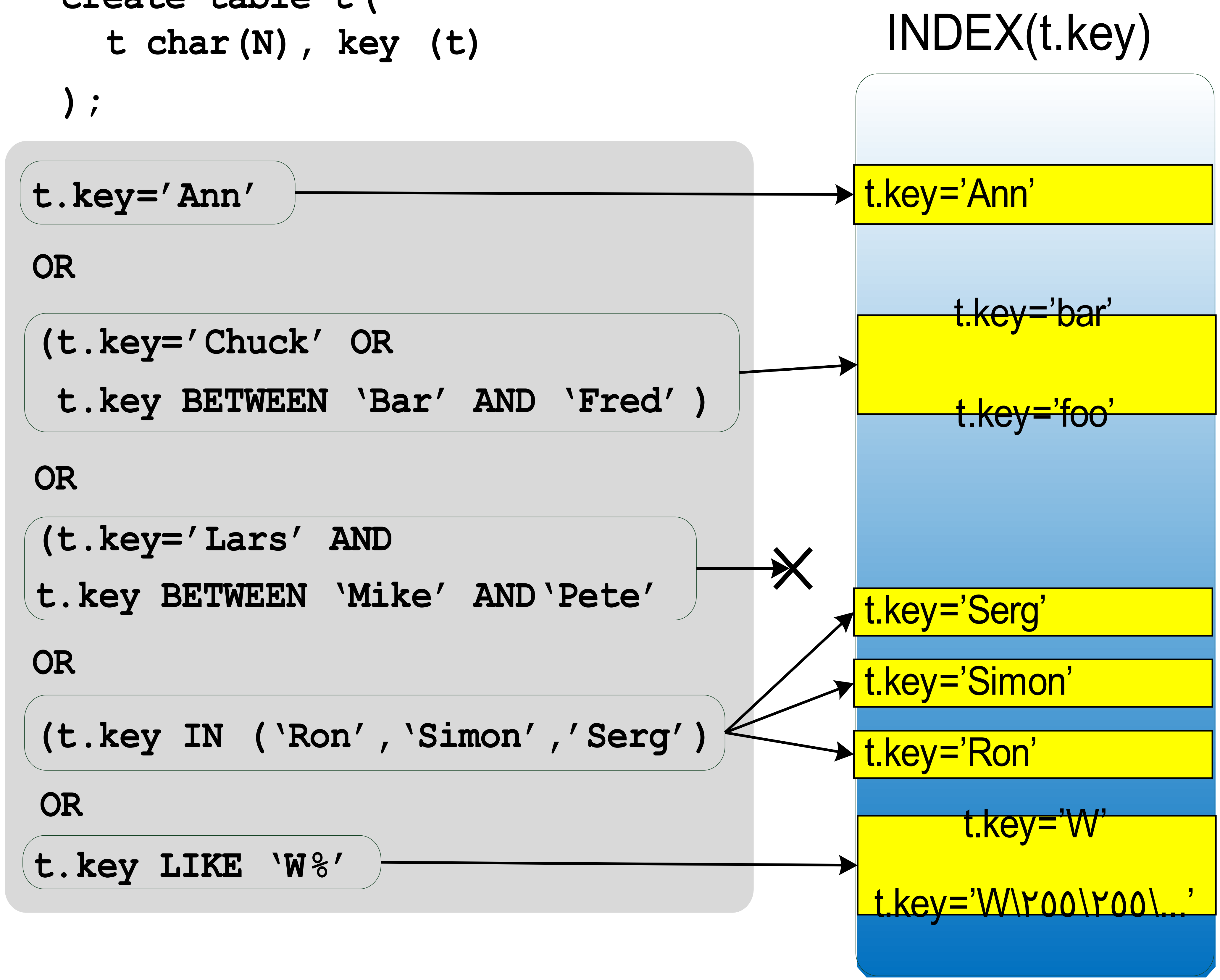
range analysis overview

- Done for each table (which has indexes/predicates)



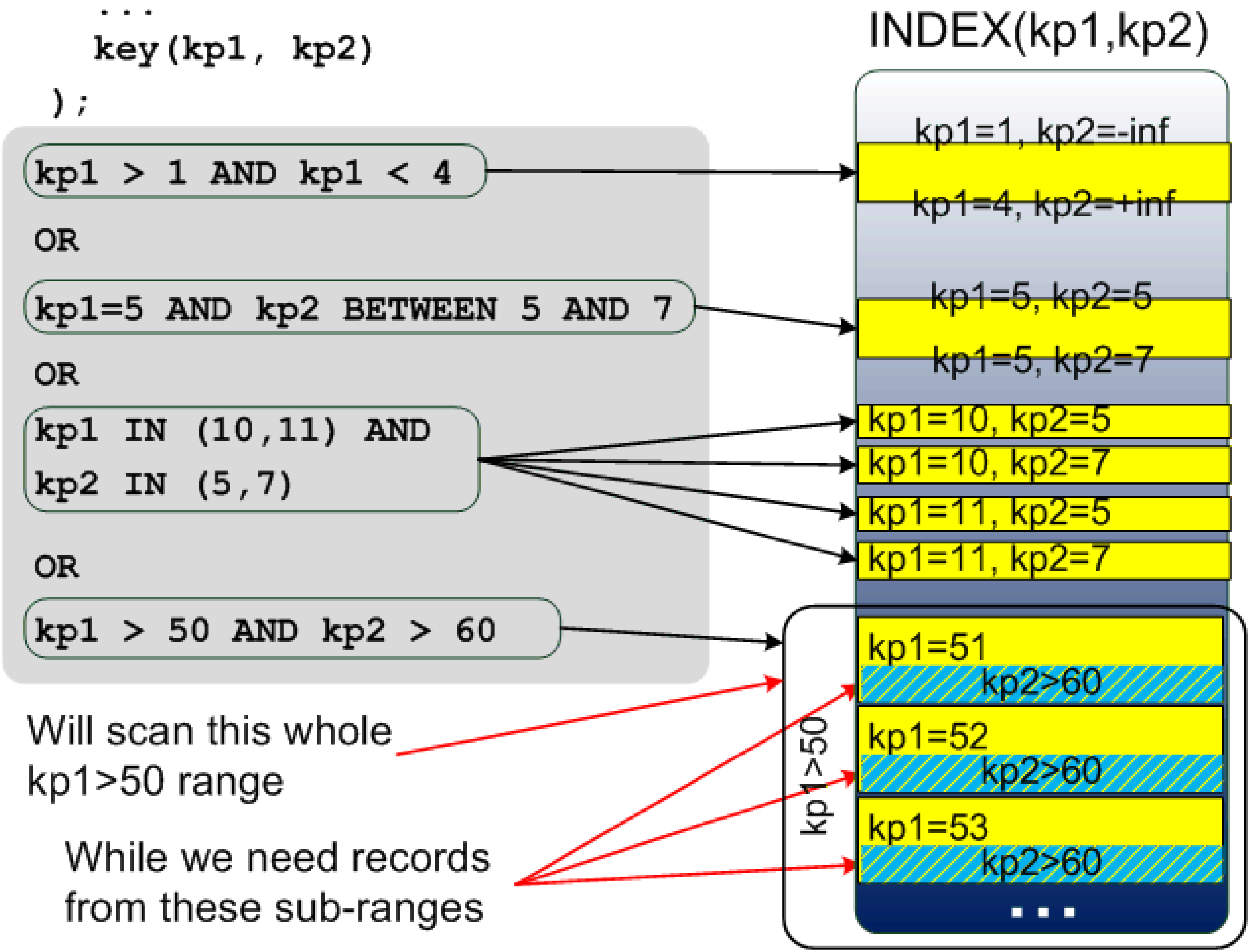
range analysis: from condition to range list (1)

```
create table t (
   t char(N), key (t)
);
```



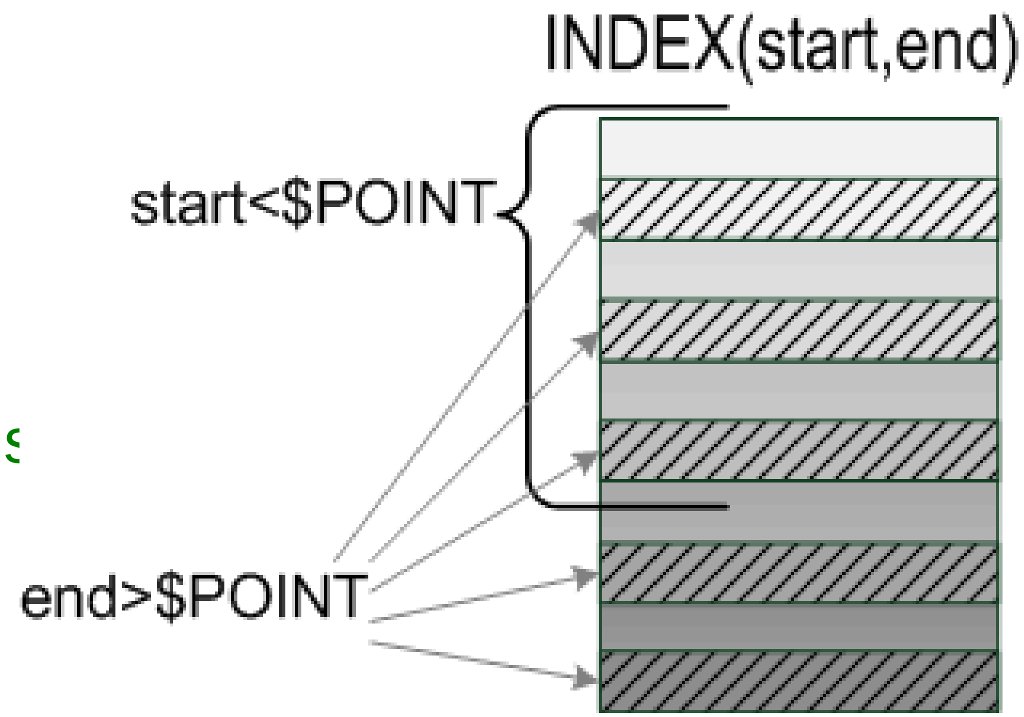
range analysis: from condition to range list (2)

```
create table t(
  ...
  key(kp1, kp2)
);
```



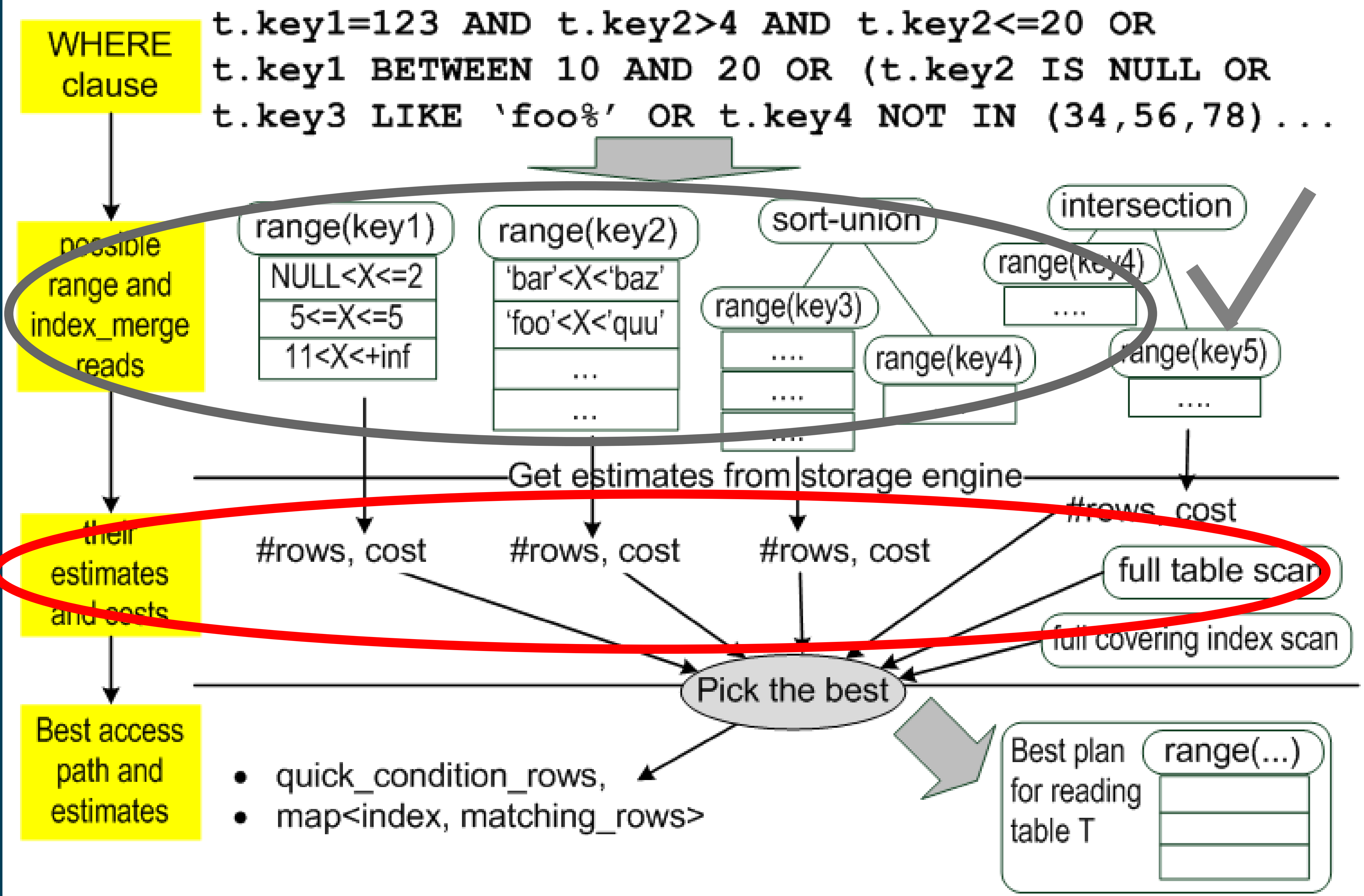
Example: workaround for infinite # of ranges

```
-- a table of disjoint IP ranges/shift times/etc
create table some_ranges (
  start int,
  end   int,
  ...
  index(start, end)
);

-- find the range that encloses
-- some given point $POINT
select * from some_ranges
where start <= $POINT and
      end >= $POINT
```




```
-- The solution
-- Make a table of range endpoints:
create table range_bounds (
  bound int,
  is_range_start bool,
  index(bound)
);
-- Find the nearest endpoint to the left of $POINT,
-- Check if it is a left endpoint
select * from (select * from range_bounds
               where bound < $POINT
               order by bound desc limit 1)
where is_range_start=1;
```

Next range analysis part: estimates



#records estimates for range access

- **range** estimates are obtained from the storage engine

```
ha_rows
handler::records_in_range(uint key_no, key_range min_key,
                          key_range max_key);
```

  - Estimate quality
    - Overall better than histograms
    - MyISAM/Maria – index dives, quite precise
    - InnoDB – some kind of dives too, but the result is not as precise (up to 2x misses)
- Effect of ANALYZE TABLE depends on the engine
  - For MyISAM/InnoDB it **will not help**.
- Can be seen in #rows in EXPLAIN:

```
mysql> explain select * from tbl where tbl.key1<10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	tbl	range	key1	key1	5	NULL	10	Using where

#records estimates for index_merge

- **index_merge** estimates are calculated from estimates of merged range scans
- They are inherently poor due to correlations:

```
explain select * from cars_for_sale
  where brand='Ford' and price < 15K
...
  where brand='Ferrari' and price < 15K
```

- [sort_]union: assumes the worst (ORed parts have no duplicates, rows(x OR y) = rows(x) + rows(y)
- intersection: assumes conditions are independent (common DBMS textbook approach)
- EXPLAIN shows number of rows produced by the access method (not number of scanned index tuples)

```
mysql> explain select * from tbl where tbl.key1<10 or tbl.key2<10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	tbl	index_merge	key١, key٢	key١, key٢	٥,٥	NULL	٢٠	Using sort_union(key1, key2); Using where

```
mysql> explain select * from tbl2 where tbl.key1=20 or tbl.key2=20;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	tbl٢	index_merge	key١, key٢	key٢, key١	٥,٥	NULL	١	Using intersect(key2, key1); Using where

Range optimizer observability

- Can see #records produced by access method in EXPLAIN
- Cannot see #records for merged scans
  - Use them individually:

```
select … where key1='foo' or key2='bar'
explain … where key1='foo'; explain … where key2='bar'
```

- Cannot easily see what ranges are scanned
- The only way at the moment: the debug log

```
less /tmp/mysqld.trace
grep for 'query:'
T@4    : | | query: explain select * from tbl where tbl.key1
between 10 and 20
T@4    : | | >mysql_parse
...
grep for 'print_quick:
T@4    : | | | | | | | | | | >print_quick
quick range select, key key1, length: 5
  10 <= X <= 20
other_keys: 0x0:
T@4    : | | | | | | | | | | <print_quick
```

- Possible future ways
  - DTrace probe (will need server source modifications to decode index lookup tuple into a readable string)
  - WL#4800 Optimizer trace (more about it later)

Controlling range optimization

- Index hints affect both range and index_merge
  - IGNORE INDEX (key1 ...)
  - USE INDEX (key1, …) - consider only those
  - FORCE INDEX(key1,...) - same as above but also consider full scan to be expensive
- Unwanted predicates can be made unusable:
  - "t.key=1" → "t.key+0=1" or "(t.key=1 OR always-true-cond)
- Hints are sufficient to control range, but not for index_merge
  - No way to force index_merge over range
  - Until now: no way allow range scans on key1,key2 but disallow index_merge
- New feature in 5.1.34: @@optimizer_switch

```
mysql> set optimizer_switch='opt_flag=value,opt_flag=value,...';
```

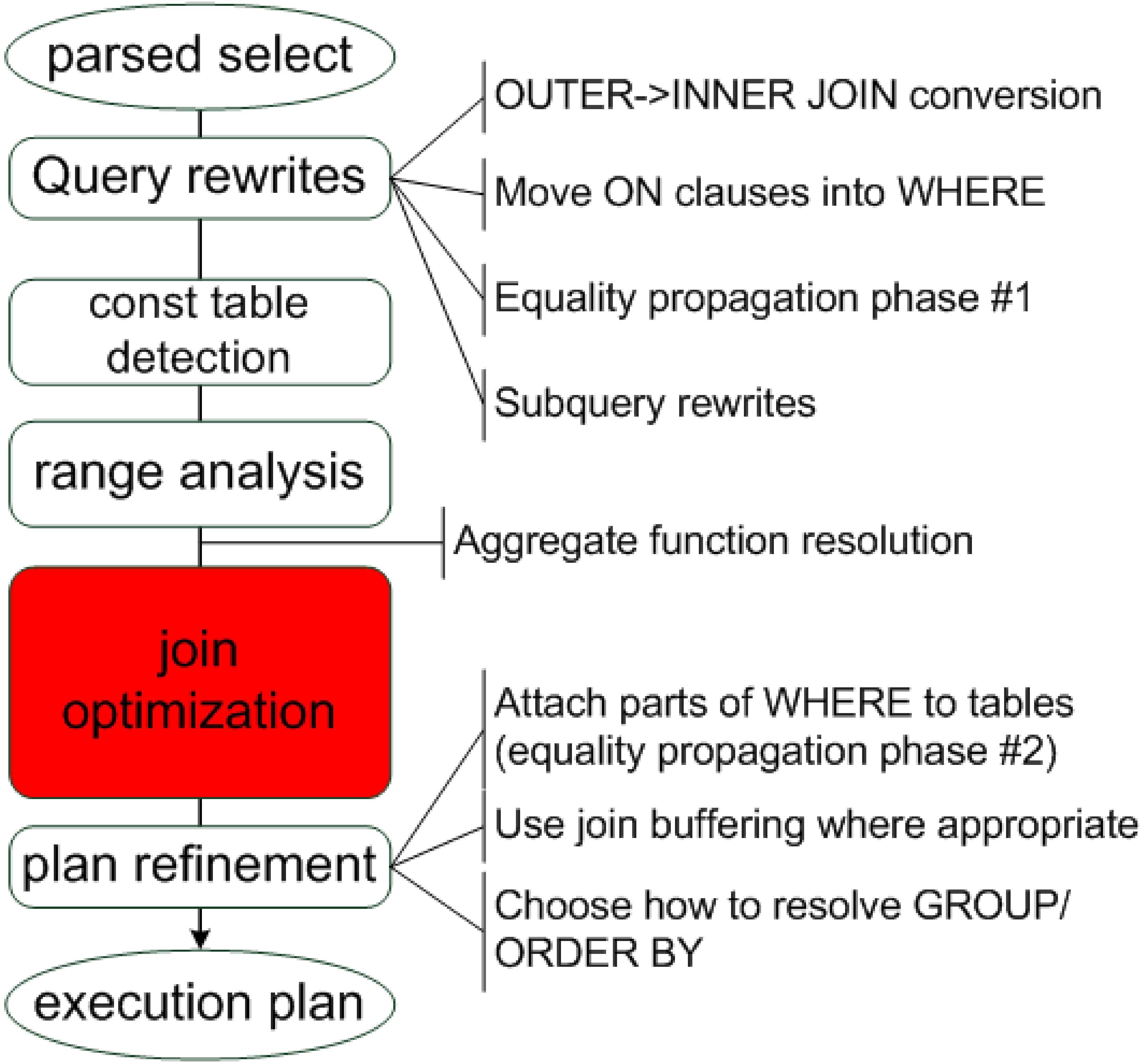
opt_flag:

- index_merge
- index_merge_union
- index_merge_sort_union
- index_merge_intersection

value:

- on
- off
- default

Join optimization



Join execution: basic NL-join algorithm

MySQL's join algorithm: nested loop join

```
select * from t1, t2
where (t1.col1='foo' and t2.col2=1 and t2.col3=t1.col3;

// Variant #1
for each record R1 in t1
{
  for each record R2 in t2
  {
    if (R1.col1='foo' && R2.col2=1 && R1.col3=R2.col3)
    {
      pass (R1,R2) to output;
    }
  }
}
```

EXPLAIN:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	t١	ALL	NULL	NULL	NULL	NULL	١٠	
١	SIMPLE	t٢	ALL	NULL	NULL	NULL	NULL	١٠	Using where

Improvement #1: use index for ref access

Suppose there is an INDEX(t2.col3):

```
select * from t1, t2
where t1.col1='foo' and t2.col2=1 and t2.col3=t1.col3;

// Variant #2: use ref access
for each record R1 in t1
{
  for each record R2 in t2 such that t2.col3=R1.col3
  {
    if (R1.col1='foo' && R2.col2=1)
    {
      pass (R1,R2) to output;
    }
  }
}
```

EXPLAIN:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	t١	ALL	NULL	NULL	NULL	NULL	١٠	
١	SIMPLE	t٢	ref	col٣	col٣	٥	t١.col٣	١	Using where

Improvement #2: join condition pushdown

Evaluate parts of WHERE condition as soon as possible

```
select * from t1, t2
where t1.col1='foo' and t2.col2=1 and t2.col3=t1.col3;

// Variant #3: evaluate parts of WHERE early
for each record R1 in t1
{
  if (R1.col1='foo')
  {
    for each record R2 in t2 such that t2.col3=R1.col3
    {
      if (R2.col2=1)
        pass (R1,R2) to output;
    }
  }
}
```

EXPLAIN:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	t١	ALL	NULL	NULL	NULL	NULL	١٠	Using where
١	SIMPLE	t٢	ref	col٣	col٣	٥	t١.col٣	١	Using where

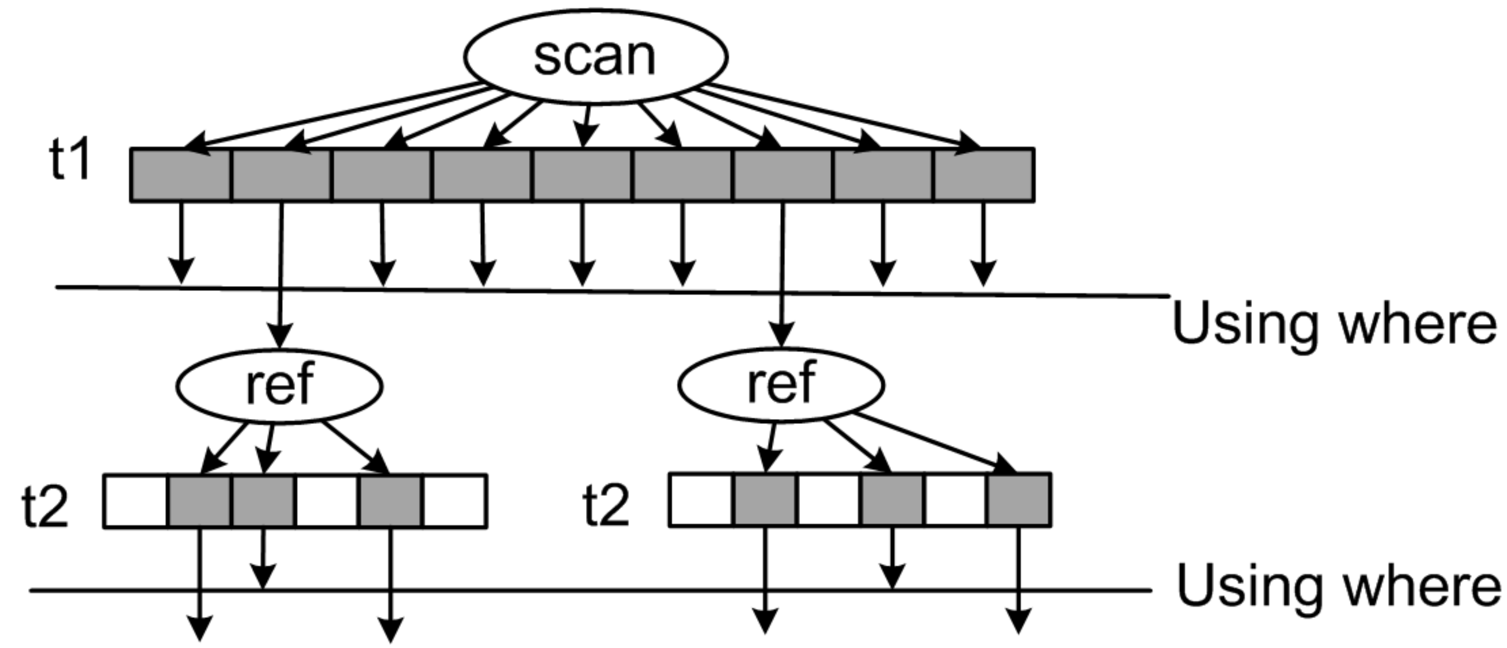
Generalizing: it is convenient to think of this in this way:

- AND-parts of the WHERE* are evaluated as soon as possible
- Pushed down predicates are used to construct table accesses (ref, range, etc)

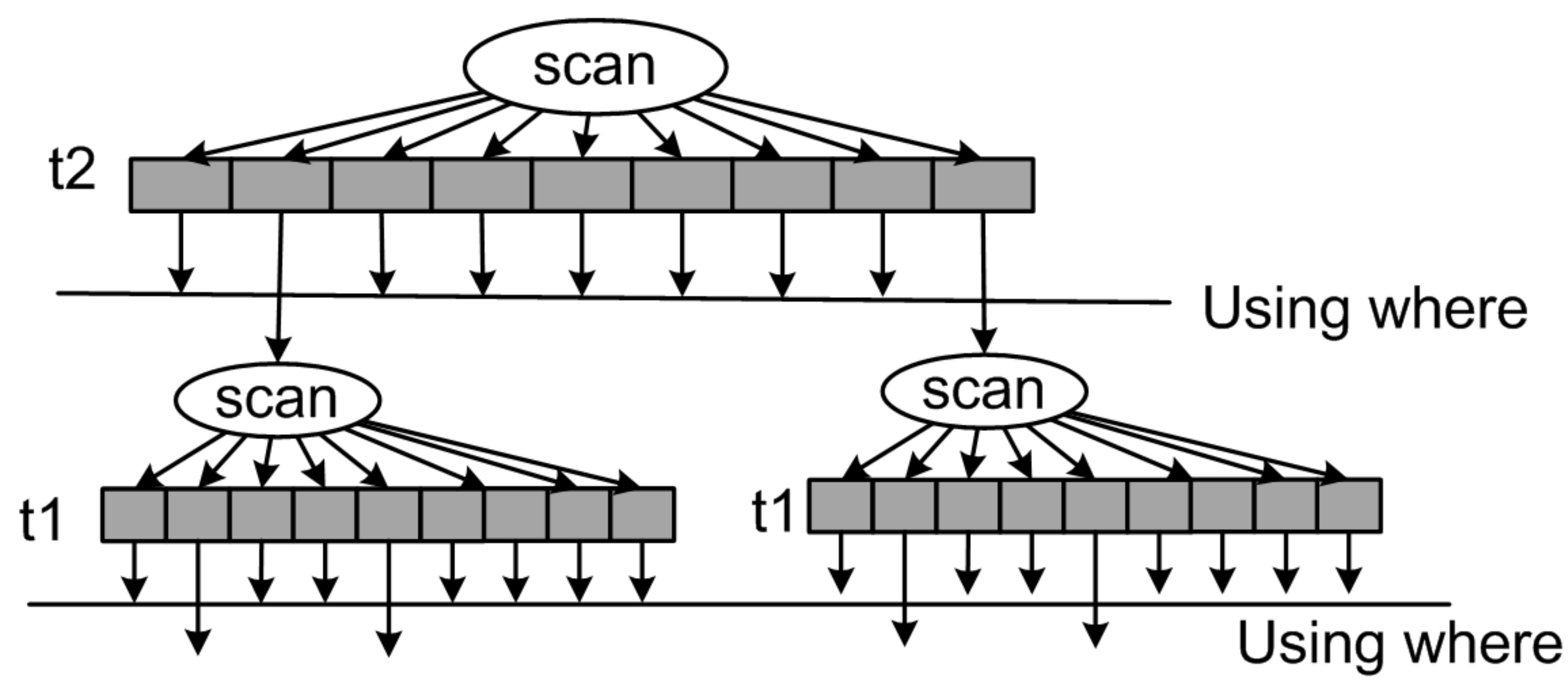
Join execution: table order matters

```
select * from t1, t2
where t1.key='foo' and t2.col2=1 and t2.key1=t1.col3;
```

- t1, t2
  can use ref(t2)



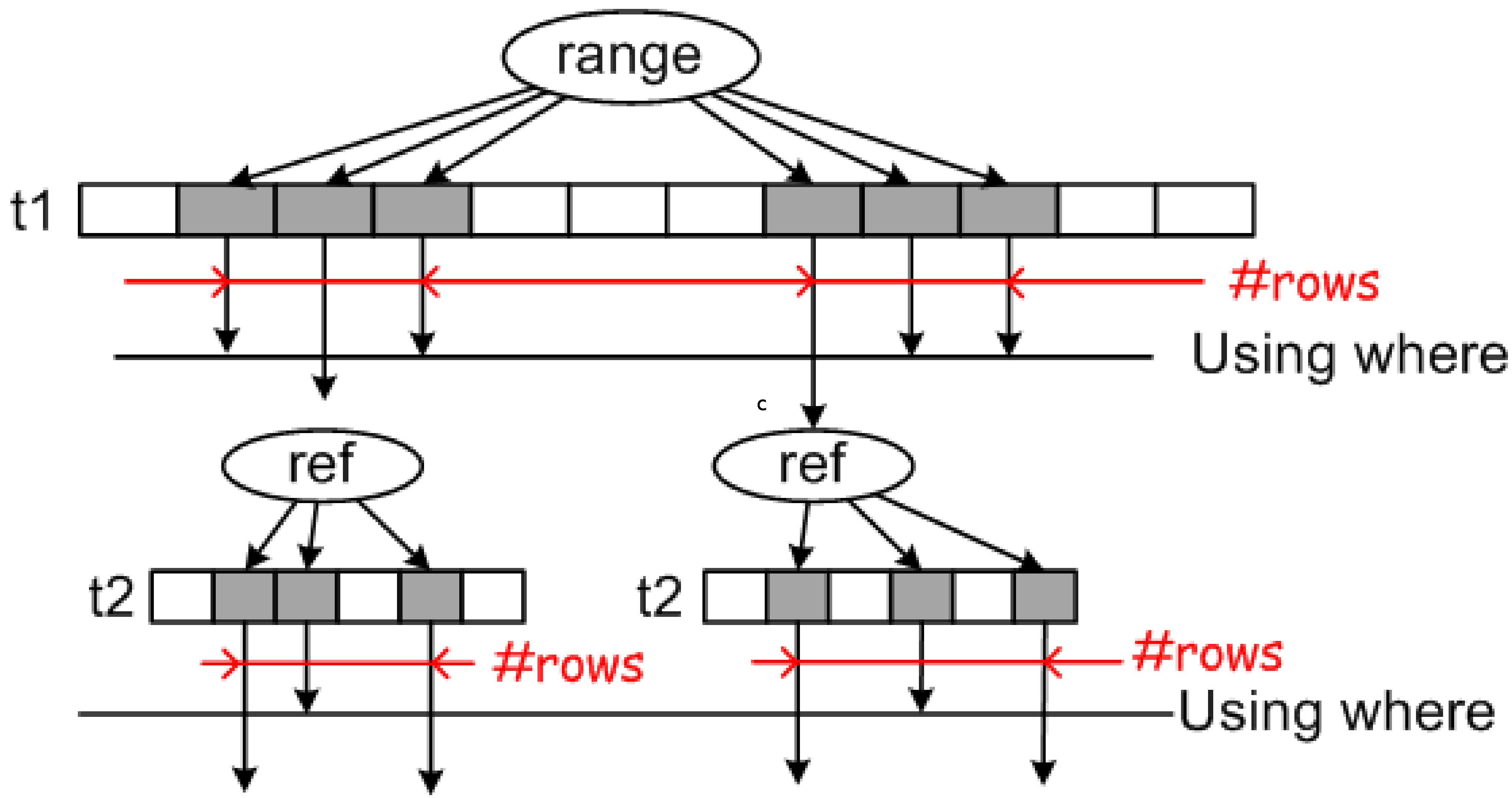
- t1, t2
  can't use ref, doing a full scan



Finding it in EXPLAIN (1): 'rows'

```
select * from t1, t2
where t1.col1 in ('foo','bar') and
      t2.col2=1 and t2.col3=t1.col3;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
١	SIMPLE	t١	range	col١	col١	٤	NULL	٦	Using where
١	SIMPLE	t٢	ref	col٣	col٣	٥	t١.col٣	٣	Using where

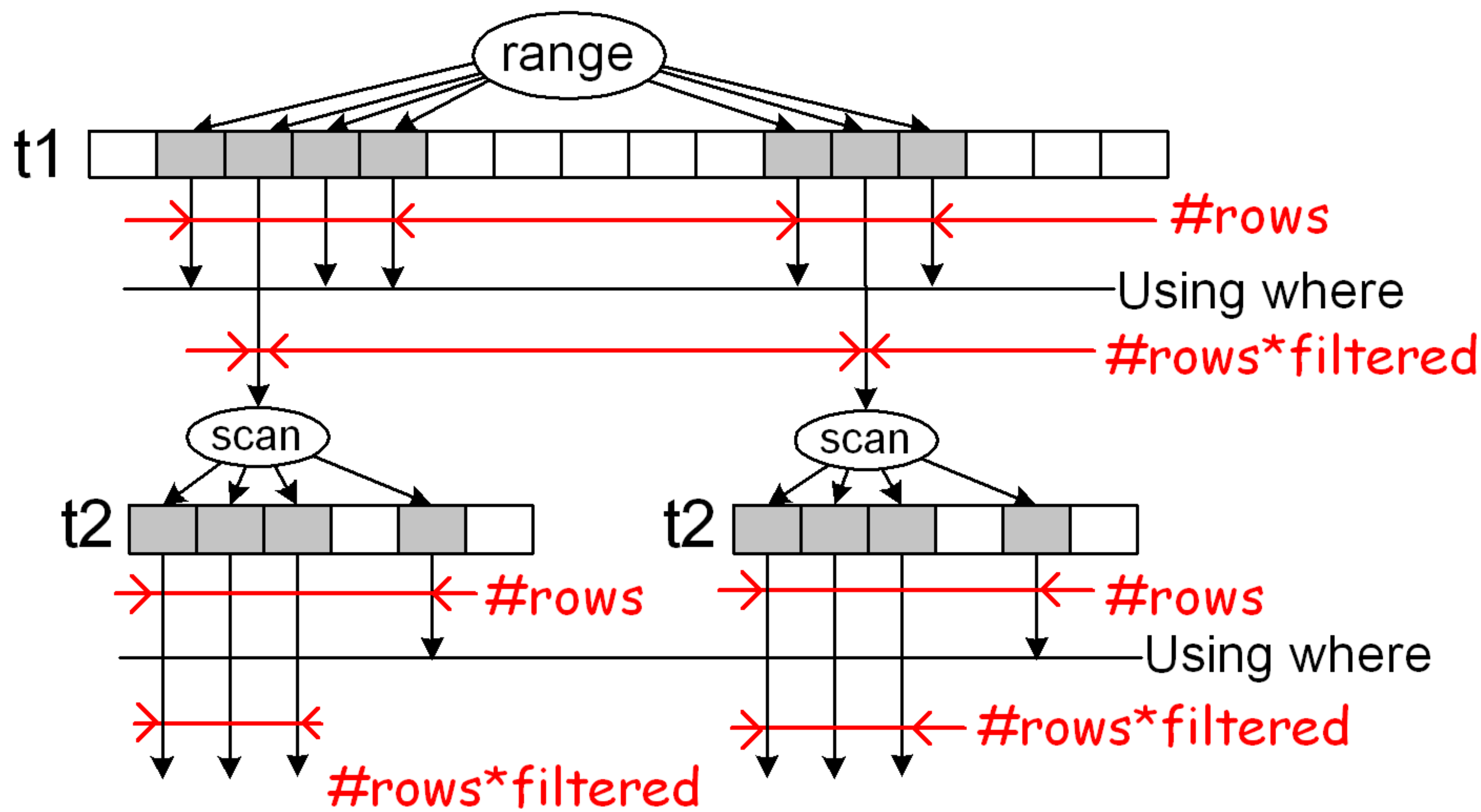


Finding it in EXPLAIN (2): 'filtered'

New in 5.1:

```
mysql> explain extended select ...
```

table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
t١	range	b	b	٥	NULL	٣٣٤	١٠٠	Using where
t٢	ALL	b	NULL	NULL	NULL	٨٠٠٠	٥١.٠٤	Using where



The bad news: 'filtered' in reality is typically either

- 100%
- Some really bad estimate, e.g. '75%'
- the only decent estimate comes from possible range access.

08:55:52 AM

Join execution recap

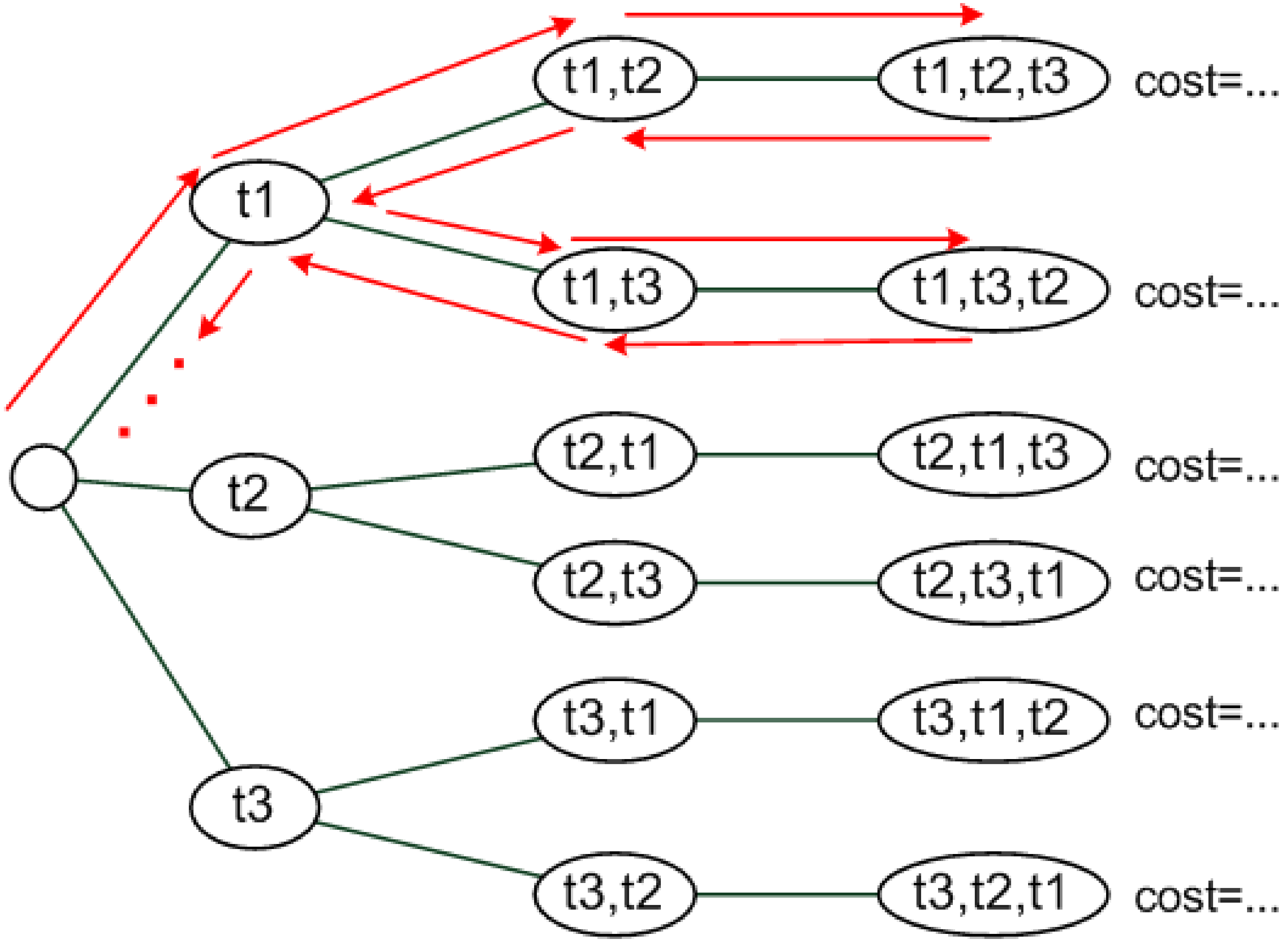
- MySQL uses nested-loops join
  - parts of WHERE are evaluated early
  - 'ref' is an access method to use indexes for joins
  - Join order matters

=> Join optimization is an art of picking the right join order.

MySQL's join optimization process

- Trivial approach: depth-first exhaustive search

```
select * from t1, t2, t3 where ...
```

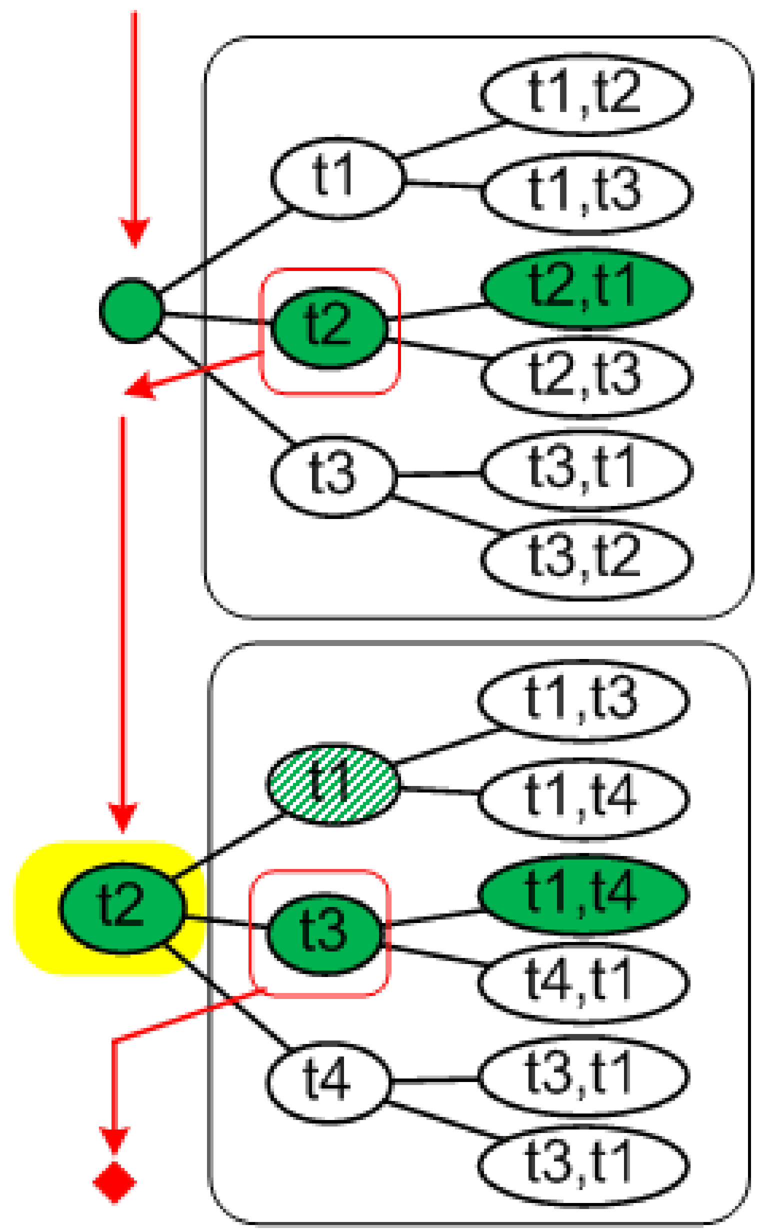


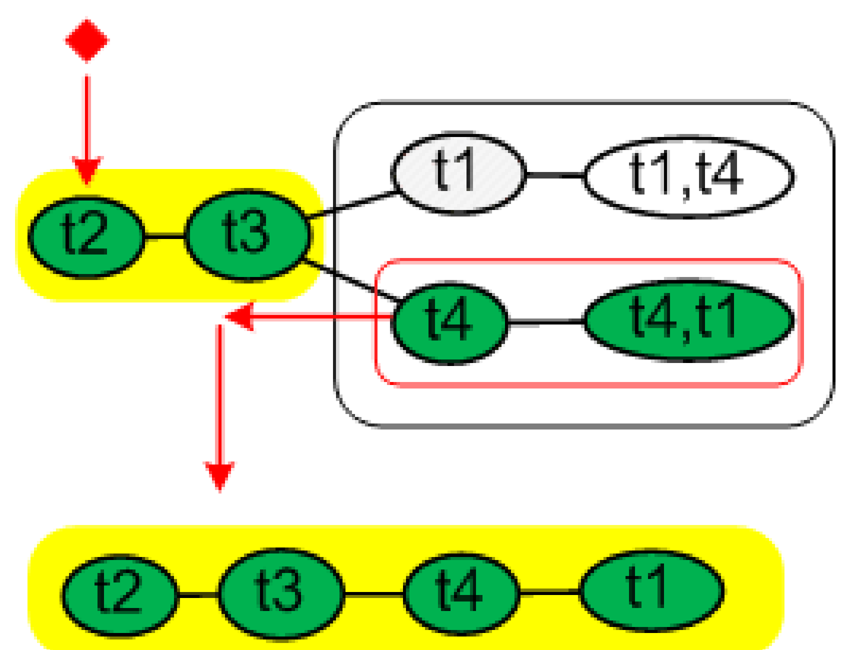
- The problem: n! combinations to consider 10!=3,6M

Greedy join optimization.

- Advanced approach: greedy optimization algorithm

```
mysql> set optimizer_search_depth=2
mysql> select * from t1, t2, t3, t4 where ...
```





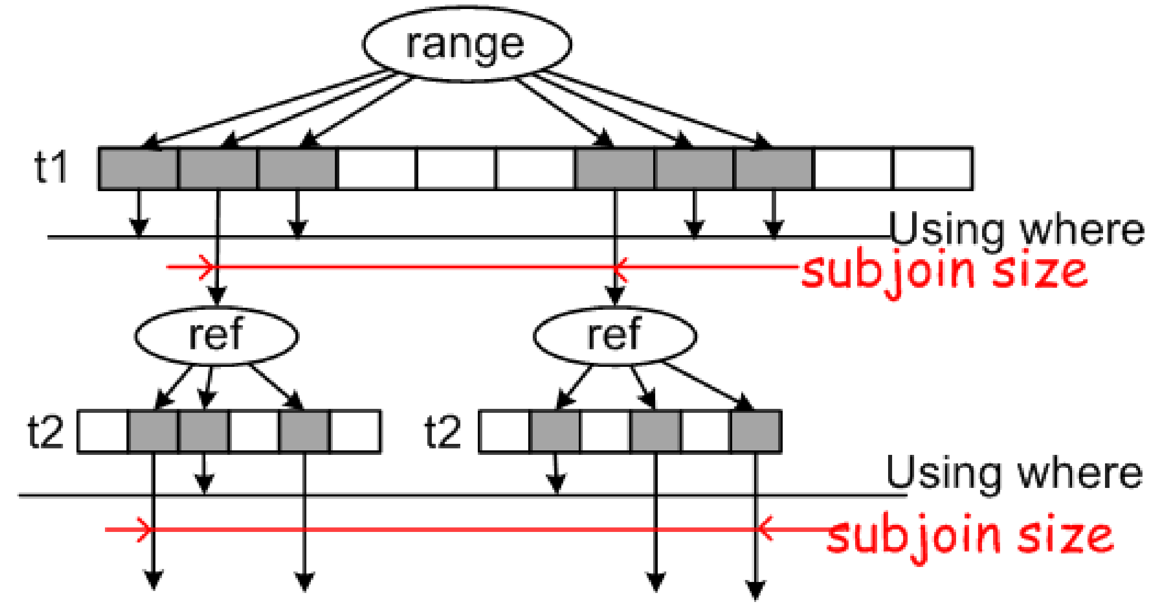
- Another parameter: @@optimizer_prune_level=0|1; controls whether the optimizer can cut off less-promising plans when considering an extension

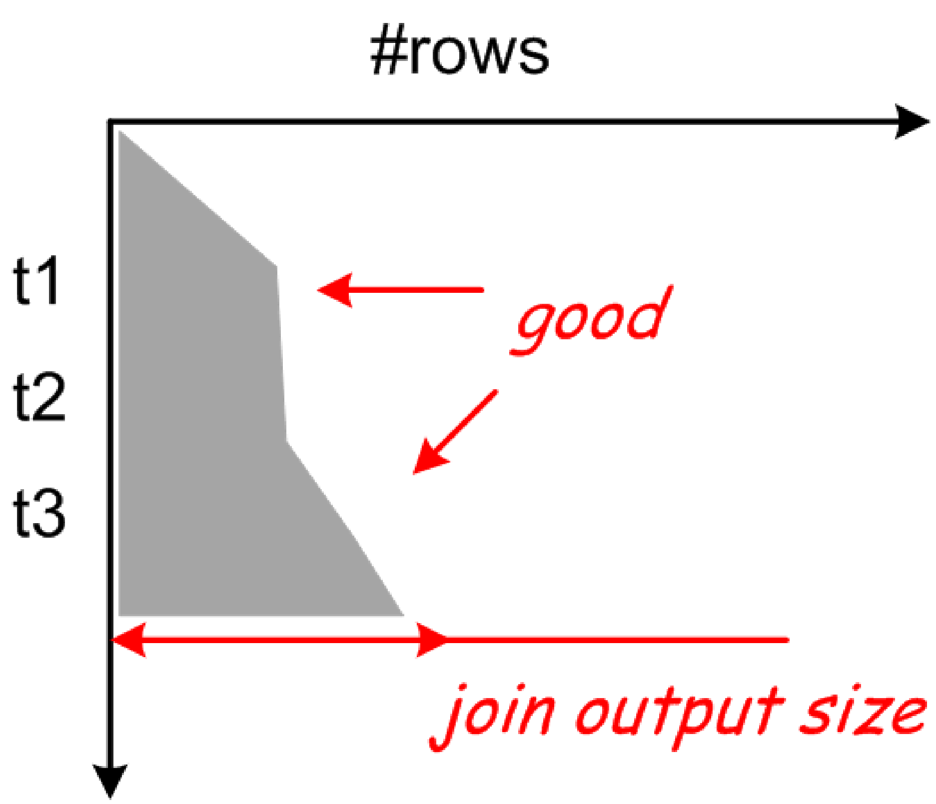
Analyzing join plan

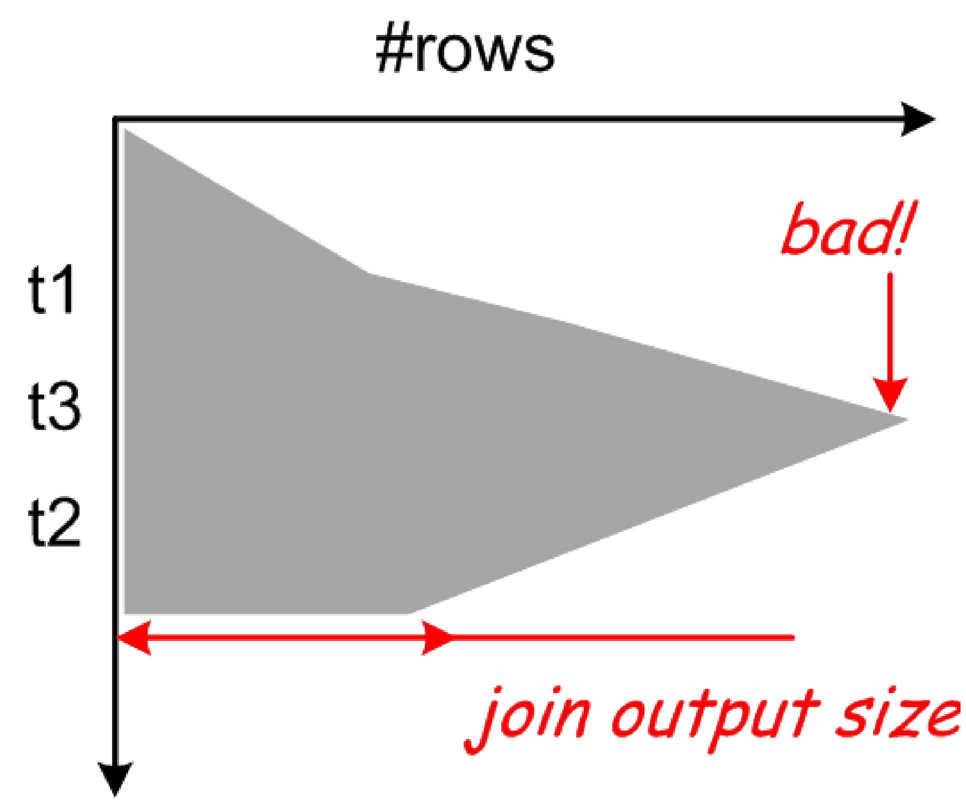
- 1. Check the join output size

```
select count(*) from t1,t2,t3, ... where …
```

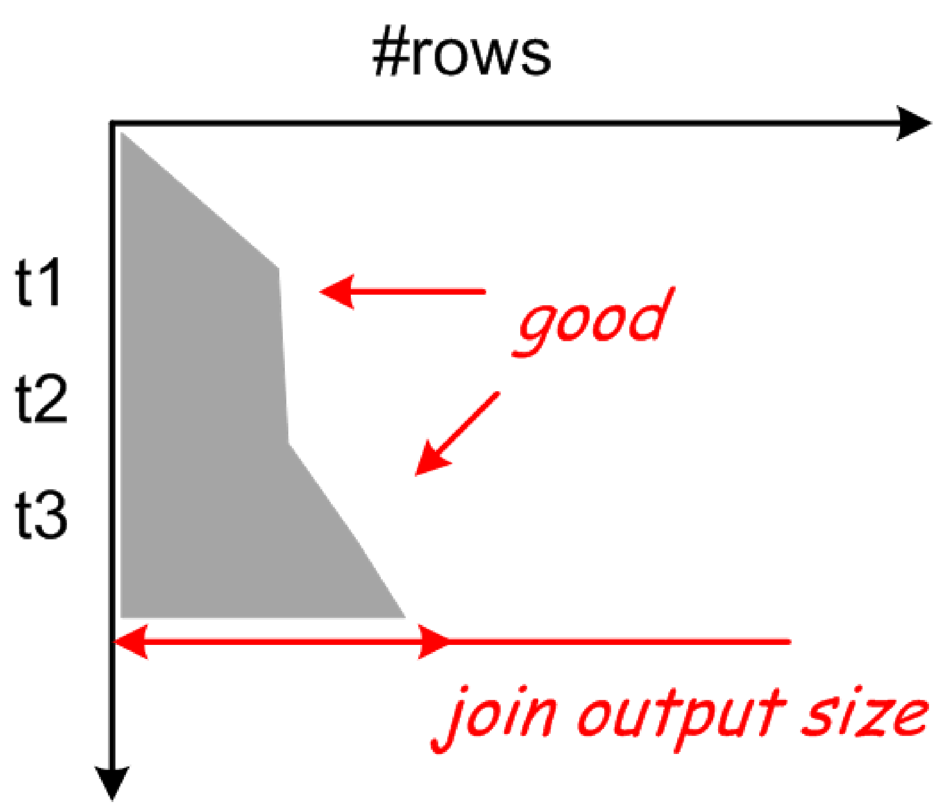
- 2. Analyze the size of sub-joins
  - t1 + Using where
  - (t1,t2) + Using where
  - ....

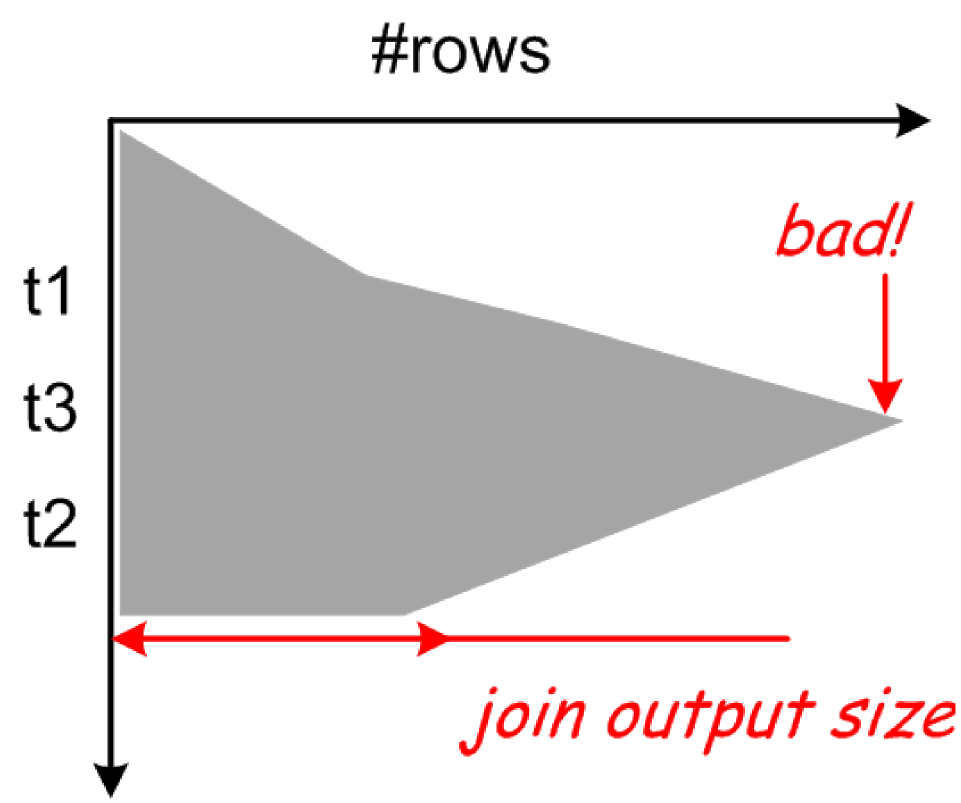






Sources of bad join plans





- Join optimizer picked a plan which it considers to be worse (can happen because of greedy optimization)
  - increase @@optimizer_search_depth
- Join optimizer picked what it considered the best, but that's not the case.
  - Incorrect estimates for 'ref' access
  - Errors in estimating 'Using where' filtering selectivity
  - Small tables at the end of join order.

ref access estimate problems

How many records we'll get for t2.col3={something}?

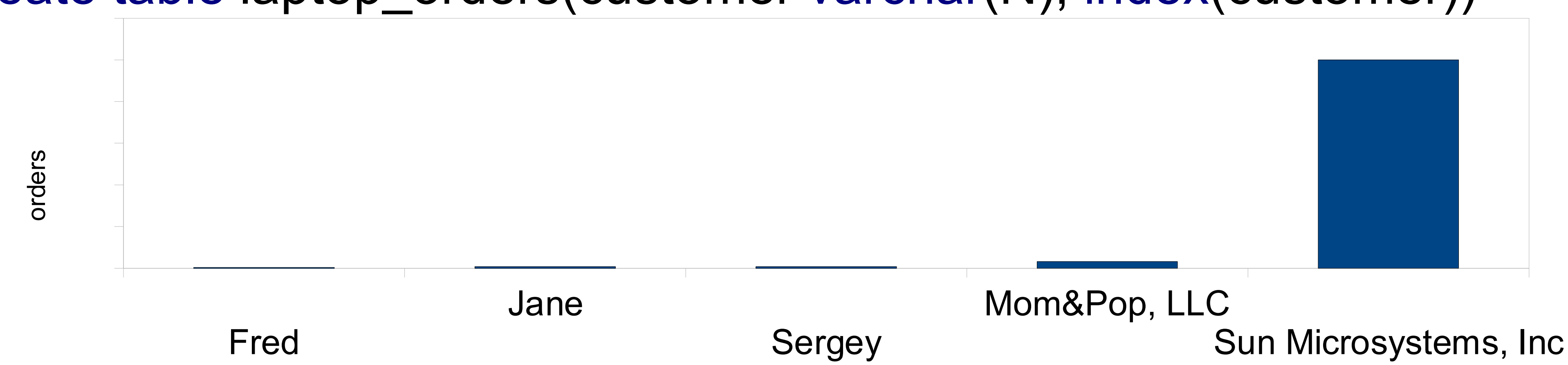
- MySQL's answer: index statistics
  - and heuristics if it is not available

```
mysql> show keys from tbl\G
*************************** 1. row ***************************
       Table: tbl
  Non_unique: 1
    Key_name: col3
Seq_in_index: 1
 Column_name: col3
   Collation: A
 Cardinality: 160
    Sub_part: NULL
      Packed: NULL
        Null: YES
  Index_type: BTREE
     Comment:
Index_Comment:
```

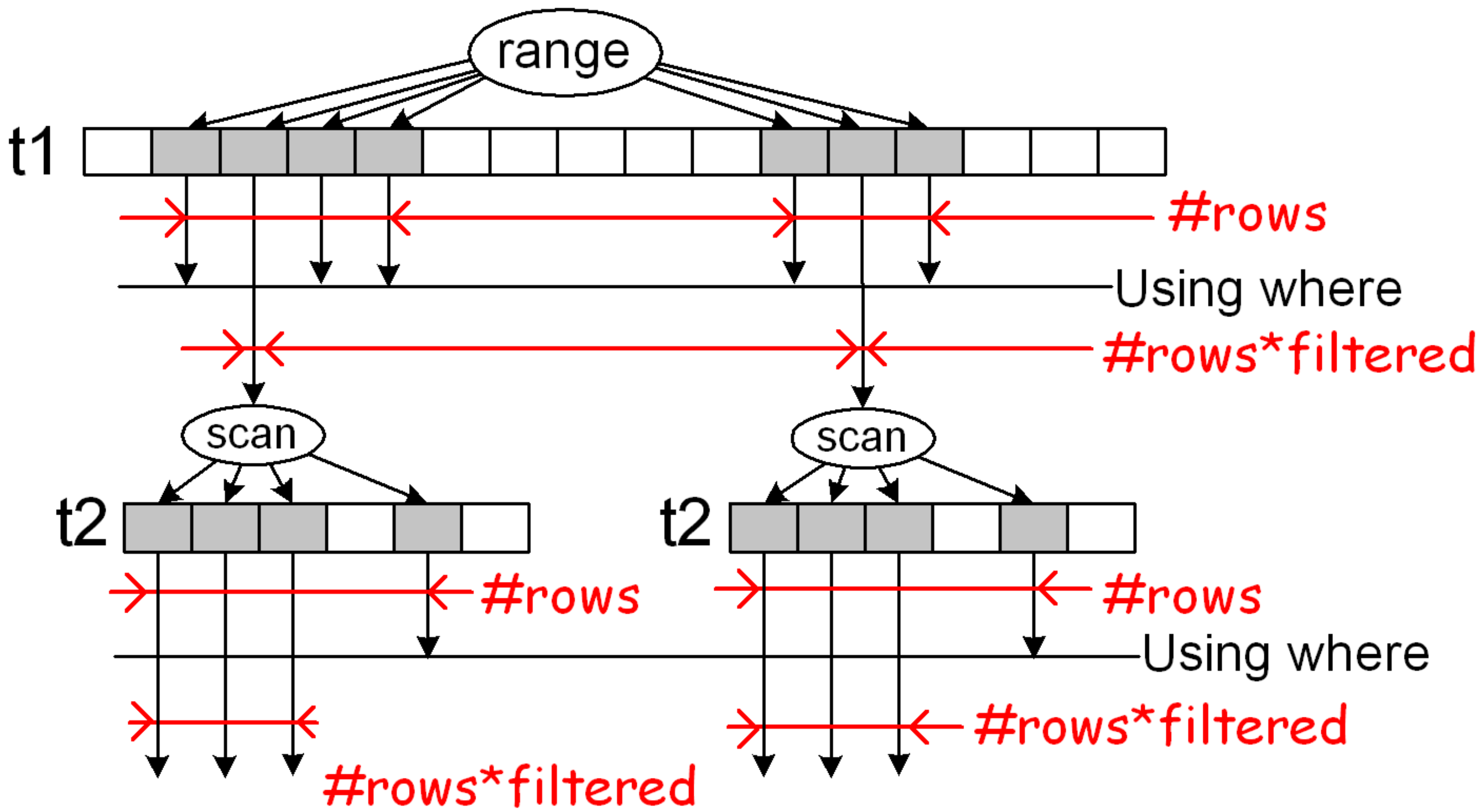
#rows / cardinality = records_per_key

- Problem: non-uniform distributions:

create table laptop_orders(customer varchar(N), index(customer))

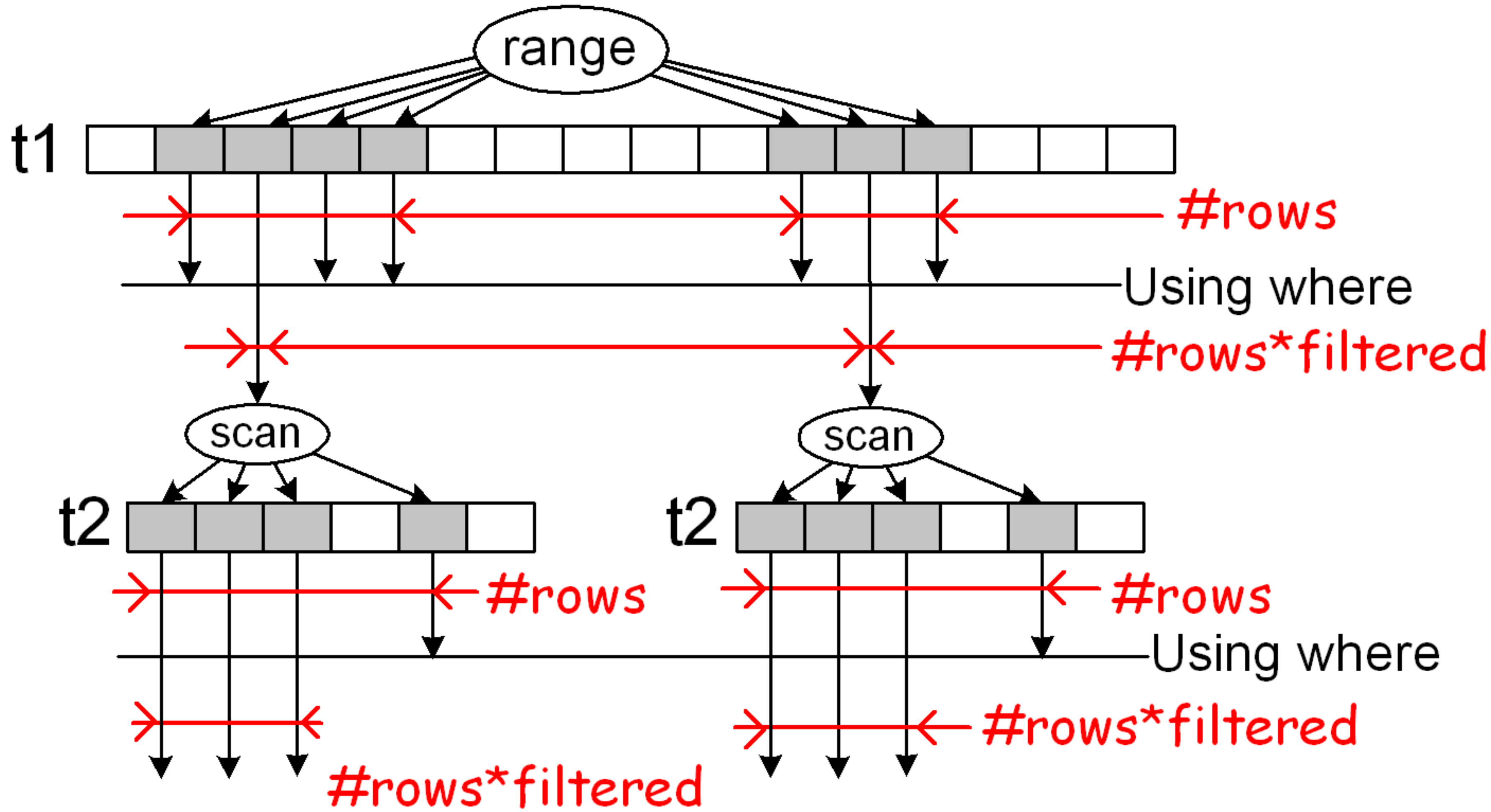


Errors in selectivity of “Using where”



- DBMS-textbook ways to find the filtered%:
  - Histograms
  - Dump estimates like “x < y” has 70% sel., “x=y” has 10% sel.
- MySQL's way:
  - Use data obtained from range optimizer.

Small tables at the end of join order



- Suppose t2 has very few rows
- They'll be in cache
- The optimizer has no idea about the cache
- It will multiply the number of reads by size-of-prefix-subjoin
  - and the error become huge.

Finding the problems

- MySQL has no EXPLAIN ANALYZE
  - Traditional way: Handler_xxx counter arithmetics
- New possibility #1: per-table statistics
- New possibility #2: DTrace-assisted

08:55:53 AM

Handler_XXX global counters

mysql> SHOW SESSION STATUS

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 0     |
| Handler_read_next     | 0     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

The problem: all table accesses increment coutners

```
ALL: n+1 * Handler_read_rnd_next

range: n_ranges * Handler_read_key,  n_rows*Handler_read_next
  (or _prev if doing a backward scan)

index: 1*Handler_read_first + N * Handler_read_rnd

index_merge:
  union/intersection: each of the branches is a scan,
  the merge op. itself is free
   + hander_read_rnd for reading post-merge record (unless "Using
   index")
  sort_union: +hander_read

ref: 1* Handler_read_key,  #records * Handler_read_next
```

08:55:53 AM

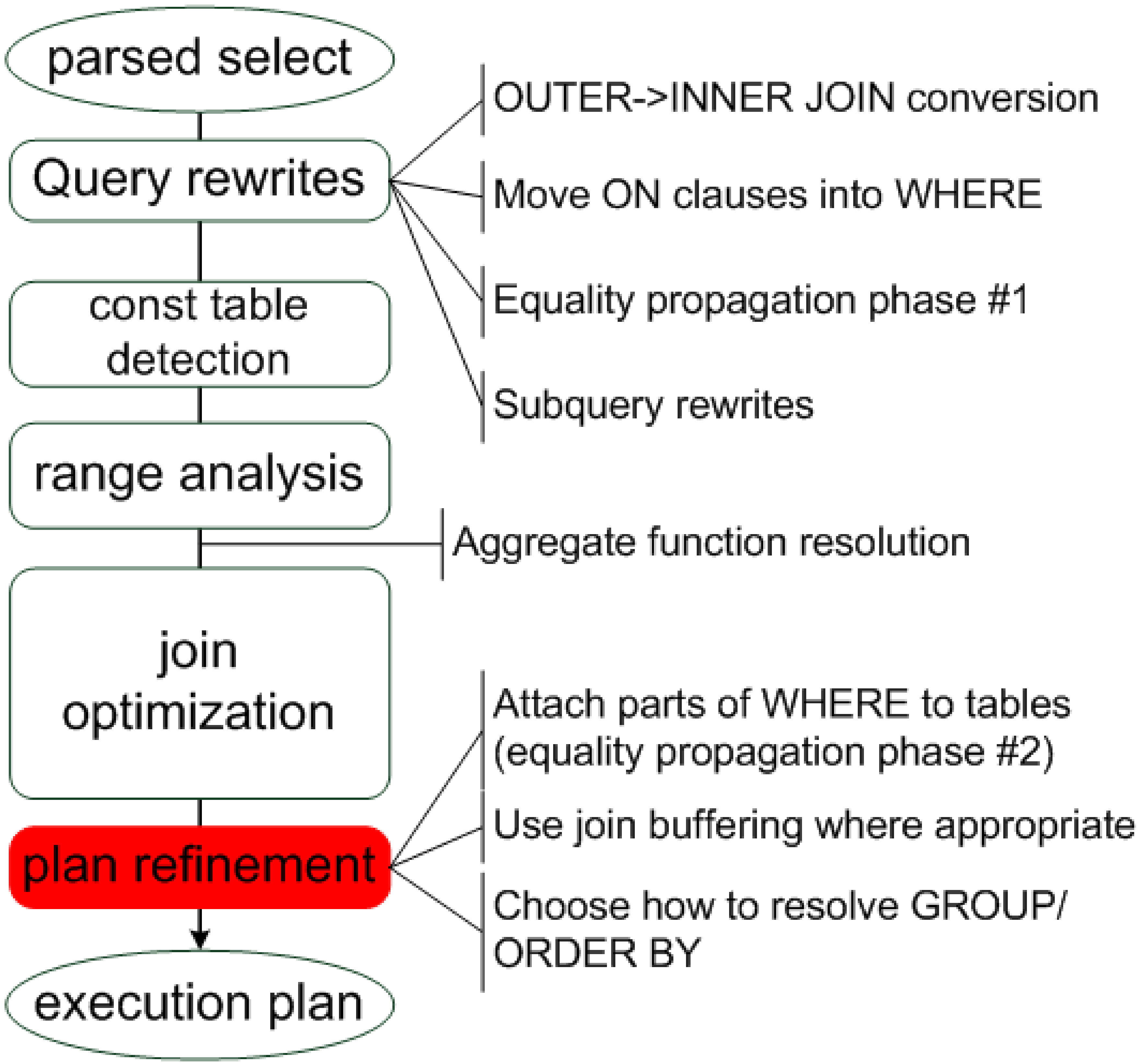
DTrace script to print real #rows:

```
#!/usr/sbin/dtrace -s

mysql$target:mysqld:*:select_start
{
  self->do_trace= 1;
}

pid$target:mysqld:ha_myisam*open*:entry
{
  printf("%d -> %s", arg0, copyinstr(arg1));
  names[arg0]= copyinstr(arg1);
}

pid$target:mysqld:ha_myisam*:entry
/!self->ts && self->do_trace/
{
  self->ts= timestamp;
  self->thisptr= names[arg0];
}

pid$target:mysqld:ha_myisam*:return
/self->ts/
{
   @time[self->thisptr]= sum(timestamp - self->ts);
   @counts[self->thisptr]= count();
   self->ts= 0;
   self->thisptr= 0;
}
```

Forcing the right join execution plan

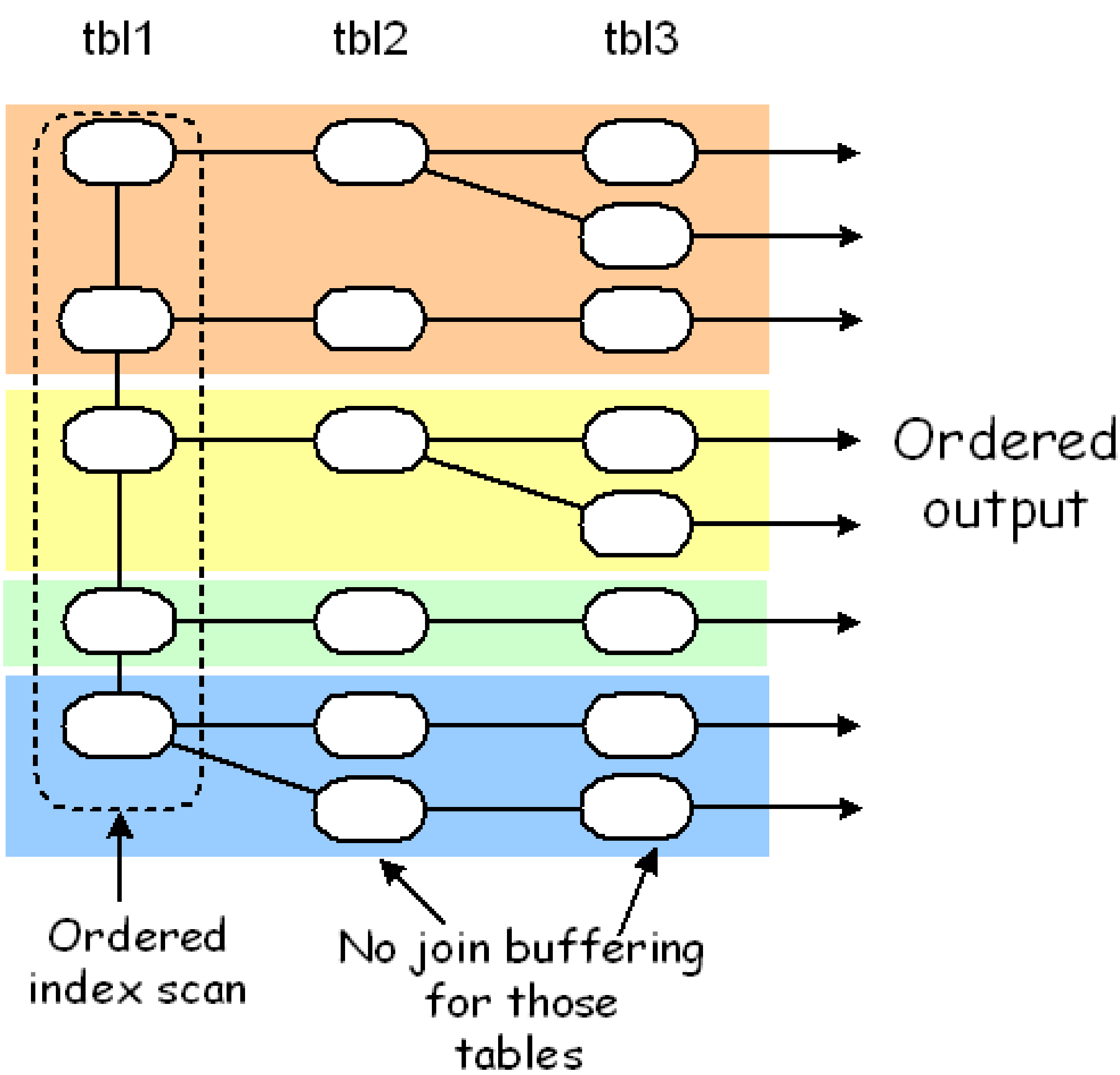
- Not easy, if it was, the optimizer would have done it for you :)
- Check index statistics for ref accesses.
  - Run ANALYZE TABLE to re-collect.
- Use IGNORE/USE/FORCE INDEX hint to force the choice of good indexes.
- Use STRAIGHT_JOIN hints to force the right join order.

Join optimization



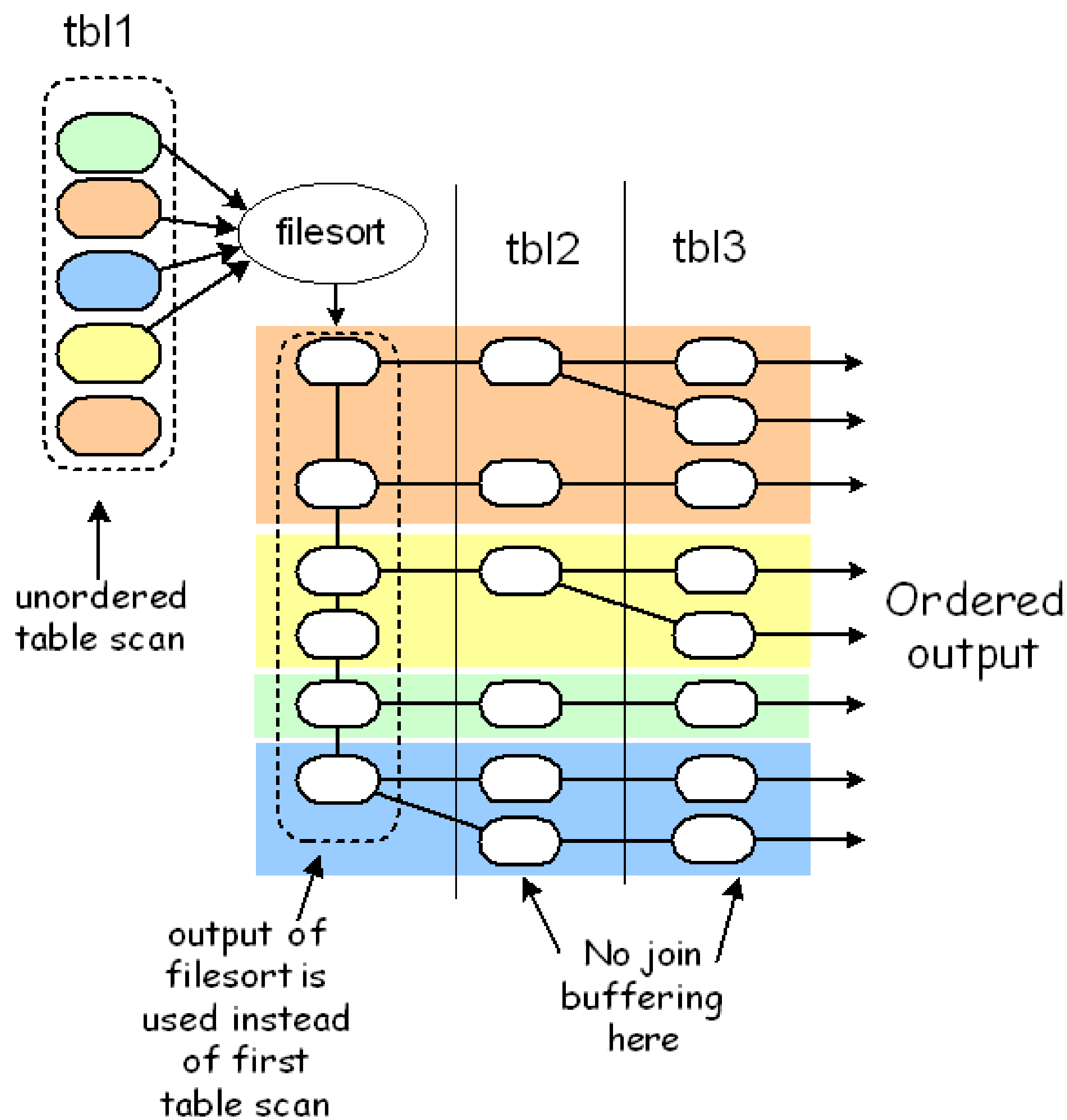
Plan refinement

- The only cost-based part: ORDER BY … LIMIT handling
- Strategy#1 (efficient LIMIT handling)



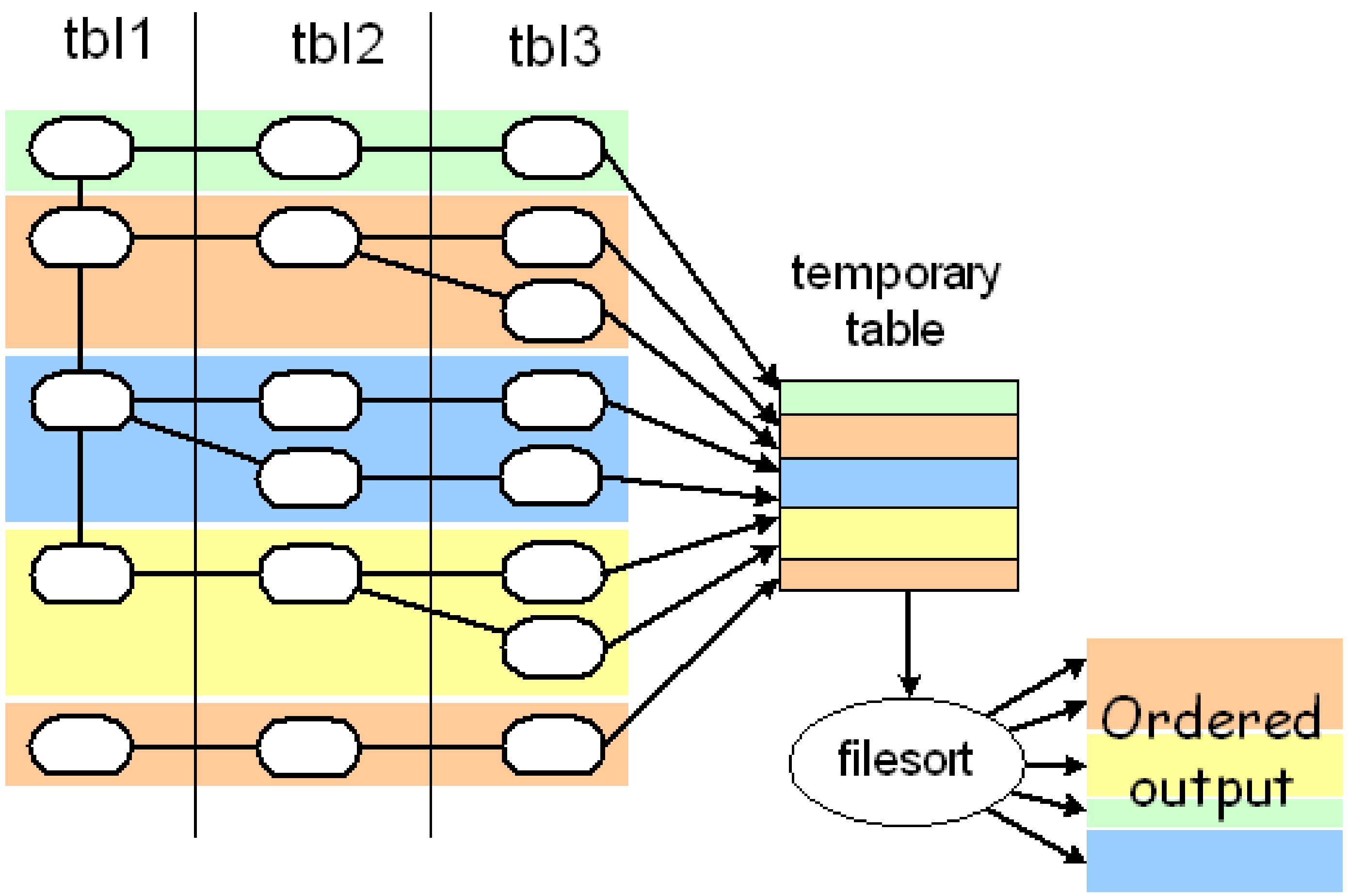
Plan refinement

- The only cost-based part: ORDER BY … LIMIT handling
- Strategy#2 (semi-efficient LIMIT handling):



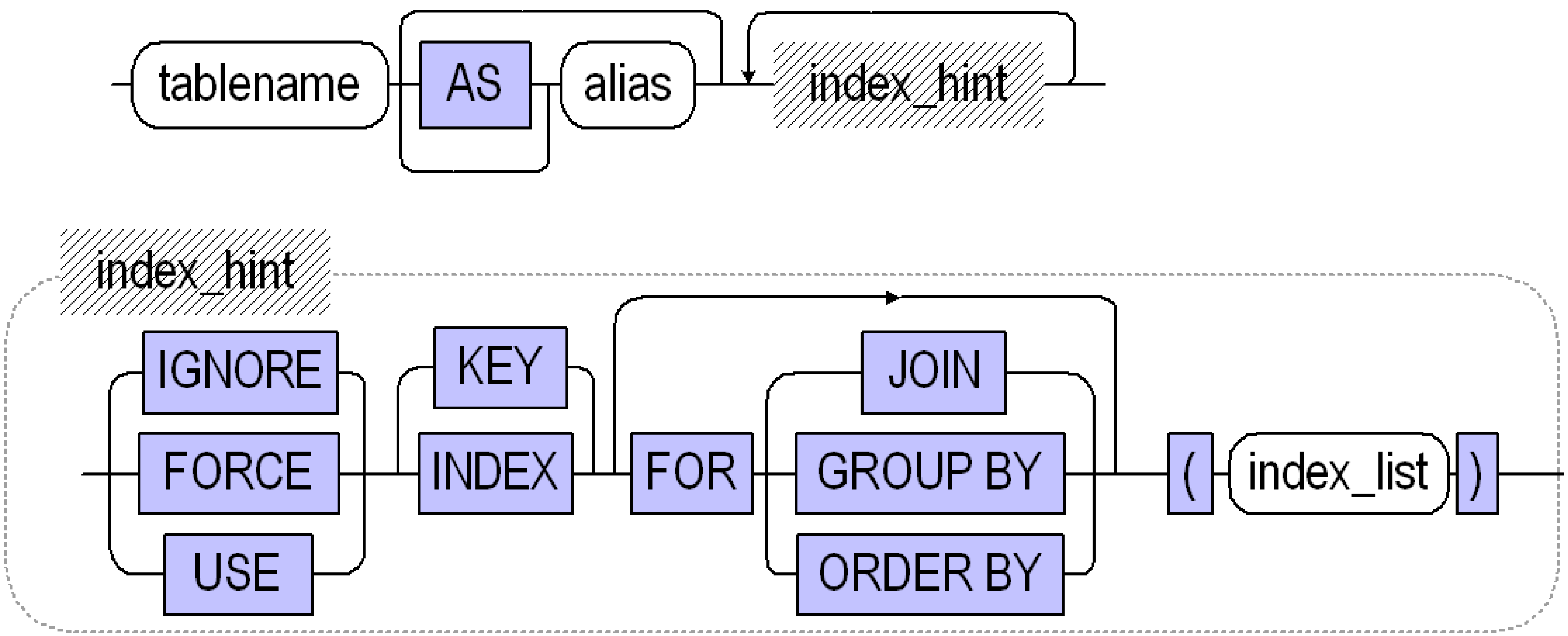
Plan refinement

- The only cost-based part: ORDER BY … LIMIT handling
- Strategy#3: no efficient LIMIT handling



Plan refinement

- Fixing ORDER BY … LIMIT problem: use hint (new in 5.1)



08:55:53 AM

References

- Optimizer resources page: http://forge.mysql.com/wiki/Optimizer_Resources
- @@optimizer_switch docs: http://s.petrunia.net/blog/?p=52
- SergeyP's optimizer blog http://s.petrunia.net/blog/
- WL#4800: Optimizer trace: http://forge.mysql.com/worklog/task.php?id=4800
- EXPLAIN CONDITIONS tree https://code.launchpad.net/~sergefp/mysql-server/mysql-6.0-explain-conds

Call for bugs

- Please do report bugs, http://bugs.mysql.com/report.php
- We can't guarantee prompt fixes (but it doesn't hurt to try:)
- But [detailed] bug reports are highly appreciated and are a valuable contribution.

Thanks and good luck with optimizer troubleshooting!

MySQL Conference & Expo

08:55:54 AM