

MySQL Under the Hood

Sergei Golubchik
Senior Software Developer
MySQL AB

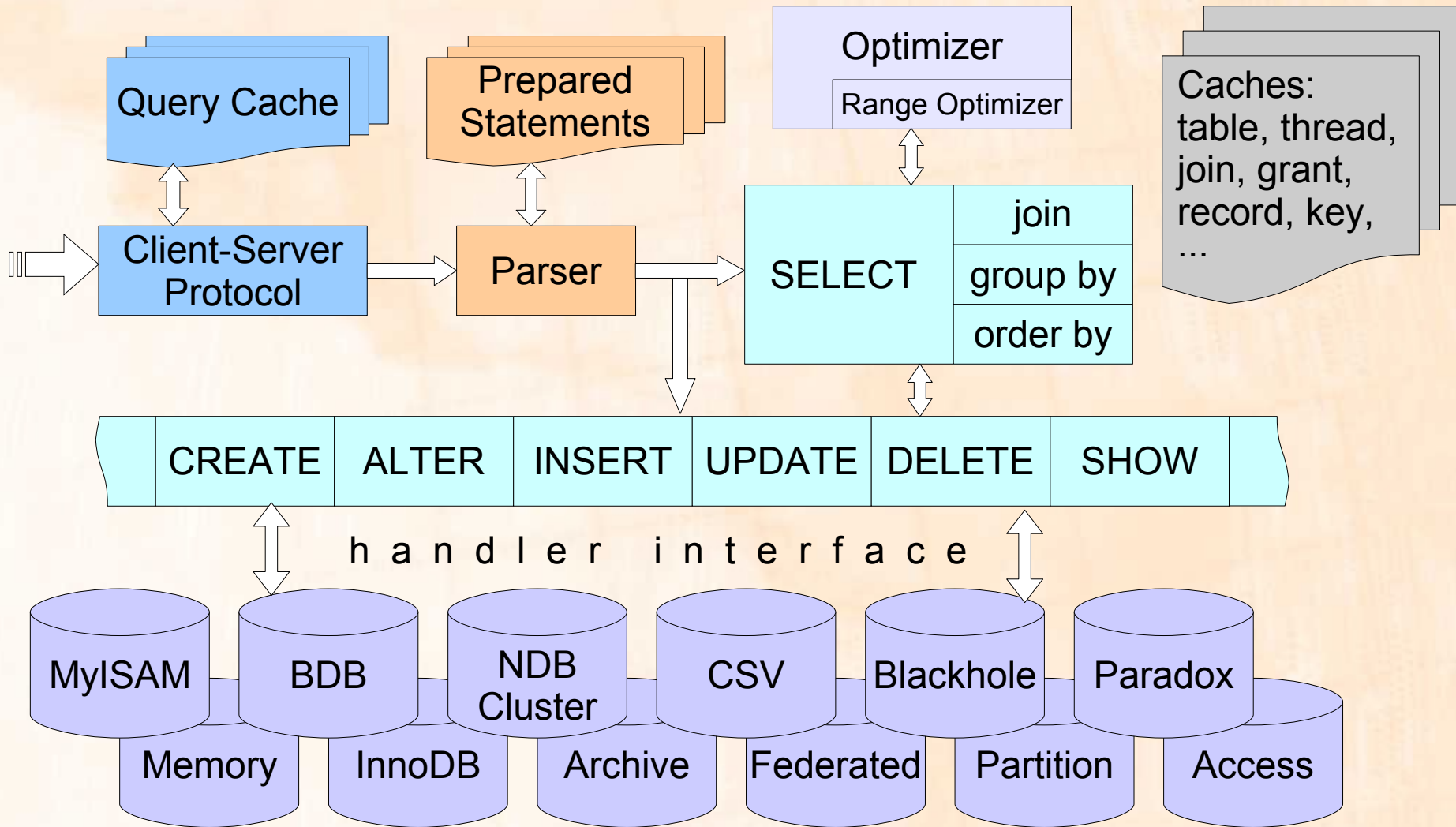
Session Outline

- Big Picture
- Life cycle of a query
 - Receiving
 - Parsing
 - Optimizing
 - Executing
- Handler API
- Directories and Files
- Extending MySQL

Who am I ?

- My name is Sergei Golubchik
- I am Senior Software Developer in MySQL AB
- Originally from Ukraine, now living in Kerpen
- Working on MySQL server internals since 1998
- Full-time MySQL AB employee since March 2000
- Some of the projects: Fulltext Search, XA, HANDLER, Precision Math, parallel repair and bulk inserts in MyISAM, indexes in MERGE, but also during these years worked with almost every part of MySQL server source

MySQL Architecture



Query Life Cycle

Moment 1: Arrival

1. Protocol

- Vio (virtual io) layer to abstract socket/ssl/tcp from the upper level
- Non-blocking io and blocking with a separate alarm thread to handle timeouts
- Handles zlib compression of packet payload

Query Life Cycle

Moment 1: Arrival

2. Query Cache

- Caches the query as it came from the client
- And the result of a query – as network packets that were sent to the client
- That's why it adds almost no overhead to regular query processing
- That's why it's fast – if the query is in the cache it is found before any work on the query is done
- If the query is in the cache, the answer is sent at once – no processing required

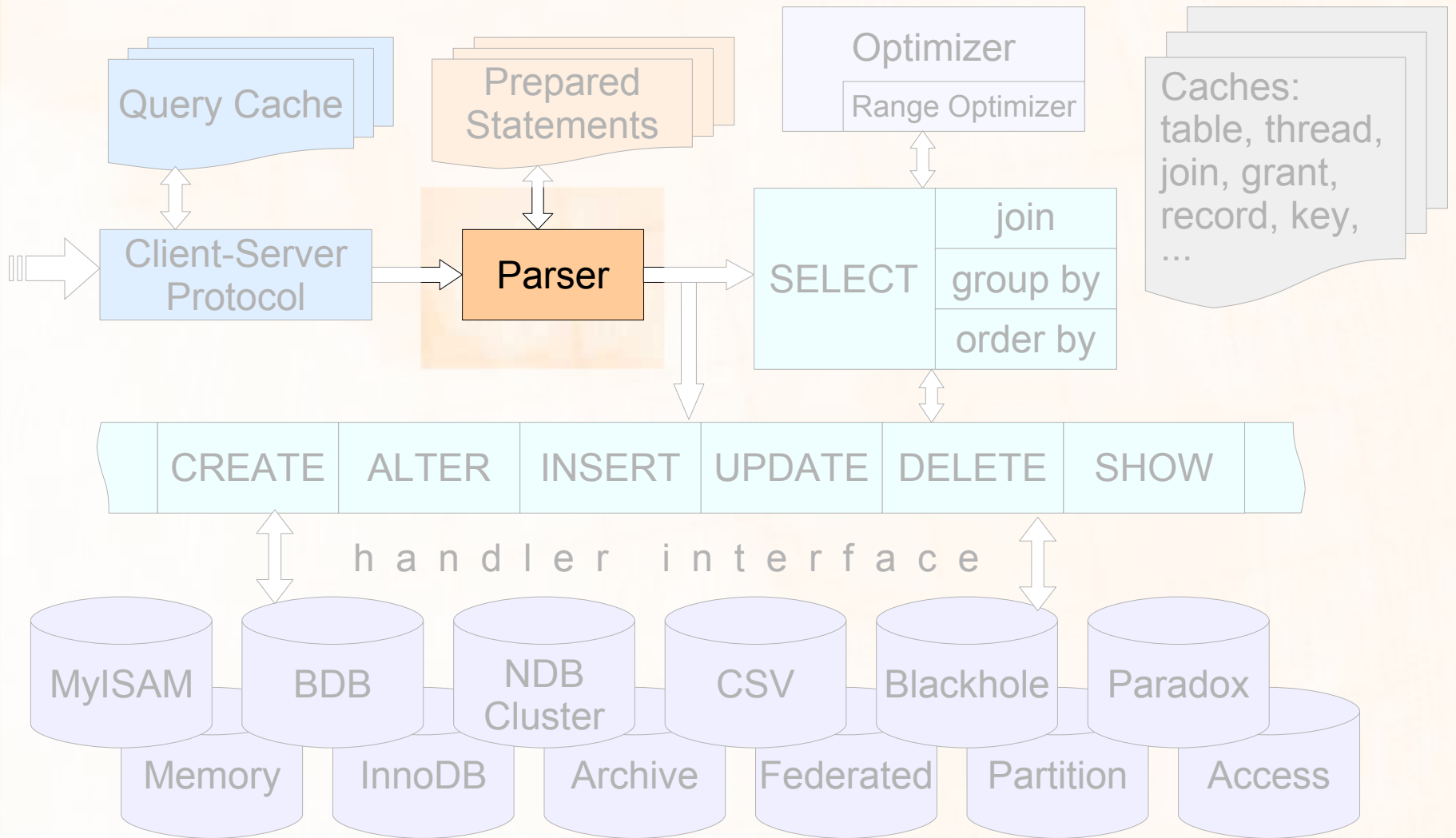
Query Life Cycle

Moment 1: Arrival

2a. Query Cache

- The query must match a cached one verbatim:
 - `SELECT * FROM foo WHERE a=1234;`
 - `SELECT * FROM foo WHERE 1234=a;`
- It can not use cached result in a subquery:
 - `SELECT a FROM foo;`
 - `SELECT a FROM (SELECT a FROM foo);`
- Or do other fancy things:
 - `SELECT * FROM foo WHERE pk=1234;`
 - `DELETE FROM foo WHERE pk=5678;`

MySQL Architecture



Query Life Cycle

Moment 2: Parsing

3. Parser

- LARL(1) parser built by bison
 - Only one look-ahead is a serious limitation:
 - `SELECT ... [WITH {ROLLUP|CUBE}]`
 - `CREATE VIEW ... [WITH CHECK OPTION]`
- Hand written lexer – flex is not used
- The query is parsed into memory data structures – table lists, select trees, expression trees. No bytecode.
- Can be used standalone 😊 – DBIx::MyParse

Query Life Cycle

Moment 2: Parsing

3a. class Item

- Base class for expressions
 - fields, constants, functions, variables, placeholders, subqueries – everything is an Item
 - properties: name, type, maybe_null, null_value, charset/collation, used_tables, const_item, args[], ...
 - methods: val_int(), val_str(), val_real(), val_decimal(); reset(), add(); eq(), print(), fix_fields(), ...
 - Evaluating an expression of any complexity is as simple as
 - `expr->val_int()` // or `val_str()`, `val_real()`, `val_decimal()`

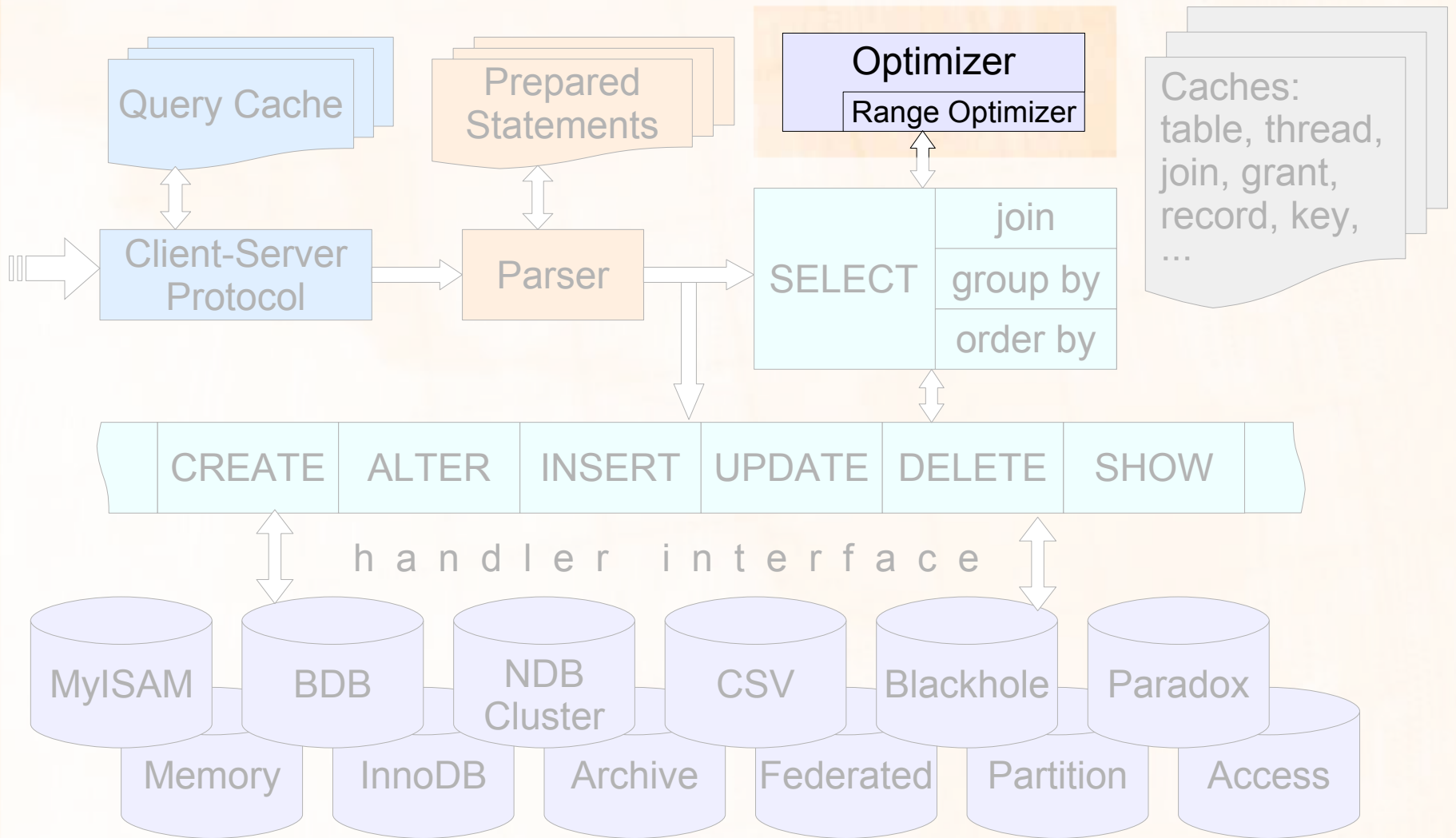
Query Life Cycle

Moment 2: Parsing

4. Preparing for execution

- Checking privileges
- Opening and locking tables
 - deadlock-free locking: all tables are locked at once and always in the same order
- Calling `item->fix_fields()`
 - fields find their tables
 - items define basic properties like `maybe_null`, `const_item`, `used_tables`
- Preparing for group by, rollup, fulltext search

MySQL Architecture



Query Life Cycle

Moment 3: Optimizing

5. Query transformations

- Equality propagation
 - WHERE $f1 = f2$ AND $f2 = f3 \rightarrow f1 = f2 = f3$
 - WHERE $f1 = 5$ AND $f2 > \sin(f1) \rightarrow \dots f2 > \sin(5)$
- Constant Folding
 - WHERE $f1 = 5+7 \rightarrow f1 = 12$
 - WHERE $f1 = 5$ AND $f2 = f1+7 \rightarrow f1 = 5$ AND $f2 = 12$
- NOT elimination
 - WHERE NOT $f1 > 5 \rightarrow f1 \leq 5$

Query Life Cycle

Moment 3: Optimizing

5a. More query transformations

- “const” tables
 - tables with no more than 1 row
 - tables, a unique key of which is compared to a constant: ... WHERE t1.pk = 12 AND t2.a = t1.b ...
- MIN()/MAX()/COUNT(*)
 - SELECT MAX(a) FROM t1;
 - SELECT MIN(b) FROM t1 WHERE a=5;
- Converting outer joins to inner joins
 - FROM t1 LEFT JOIN t2 WHERE t2.name LIKE 'a%'

Query Life Cycle

Moment 3: Optimizing

5b. Even more query transformations

- View merge
 - CREATE VIEW t AS SELECT x+1 AS a FROM t WHERE y=5
 - SELECT a*5 FROM v WHERE a=5
 - becomes
 - SELECT (x+1)*5 FROM t WHERE y=5 AND (x+1)=5
- Subquery transformations
 - IN → EXISTS
 - ALL/ANY → MIN/MAX, e.g. $f > \text{ALL}(\dots) \rightarrow f > \text{MAX}(\dots)$

Query Life Cycle

Moment 3: Optimizing

6. Range Optimizer

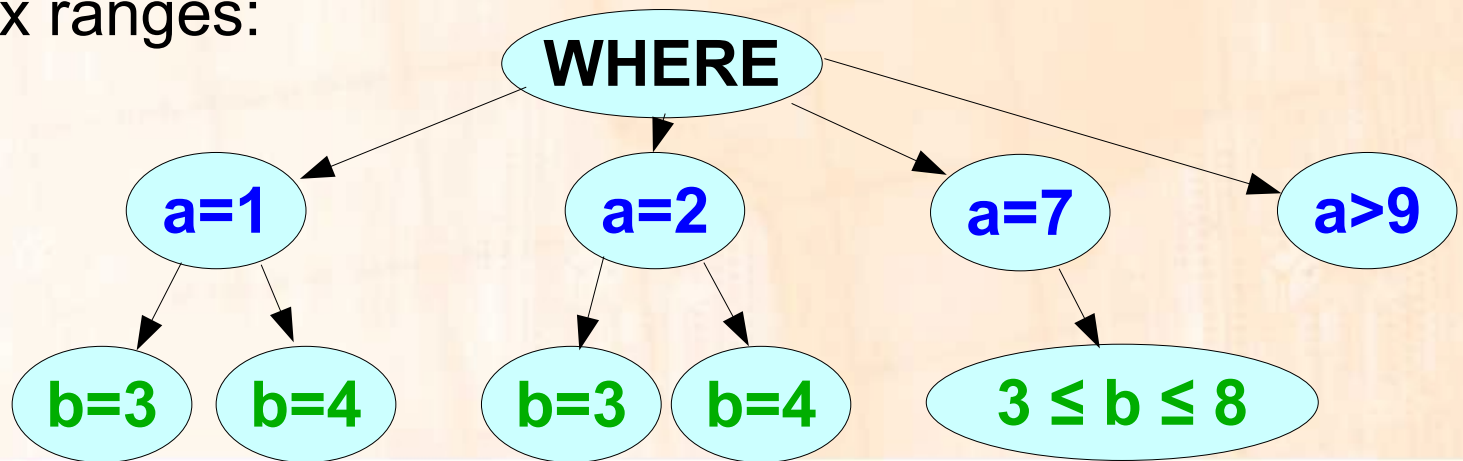
- Supports $>$, \geq , $<$, \leq , $=$, \neq , IS NULL, \Leftrightarrow , BETWEEN, IN, LIKE, and any combination of the above combined with AND/OR
- Supports only comparison with a constant
- Builds a tree of ranges (see next slide)
- Does not use index cardinality but asks storage engine for estimates for every key range from the tree – does not suffer from skewed distributions

Query Life Cycle

Moment 3: Optimizing

6a. Range Optimizer

- Builds a tree of ranges:
 - INDEX (a, b)
 - WHERE a IN (1,2) AND b IN (3,4) OR a = 7 AND b BETWEEN 3 AND 8 OR a > 9 AND b < 5
 - Six ranges:



Query Life Cycle

Moment 3: Optimizing

7. Choosing an execution plan

- Creating a list of possible indexes
- Getting statistics from the storage engine
 - index cardinality, table scan and index lookup costs
- Performing search for the best execution plan
 - best plan is the one that has the lowest cost
 - exhaustive search (up to MySQL 4.1), or greedy search (starting from 5.0)

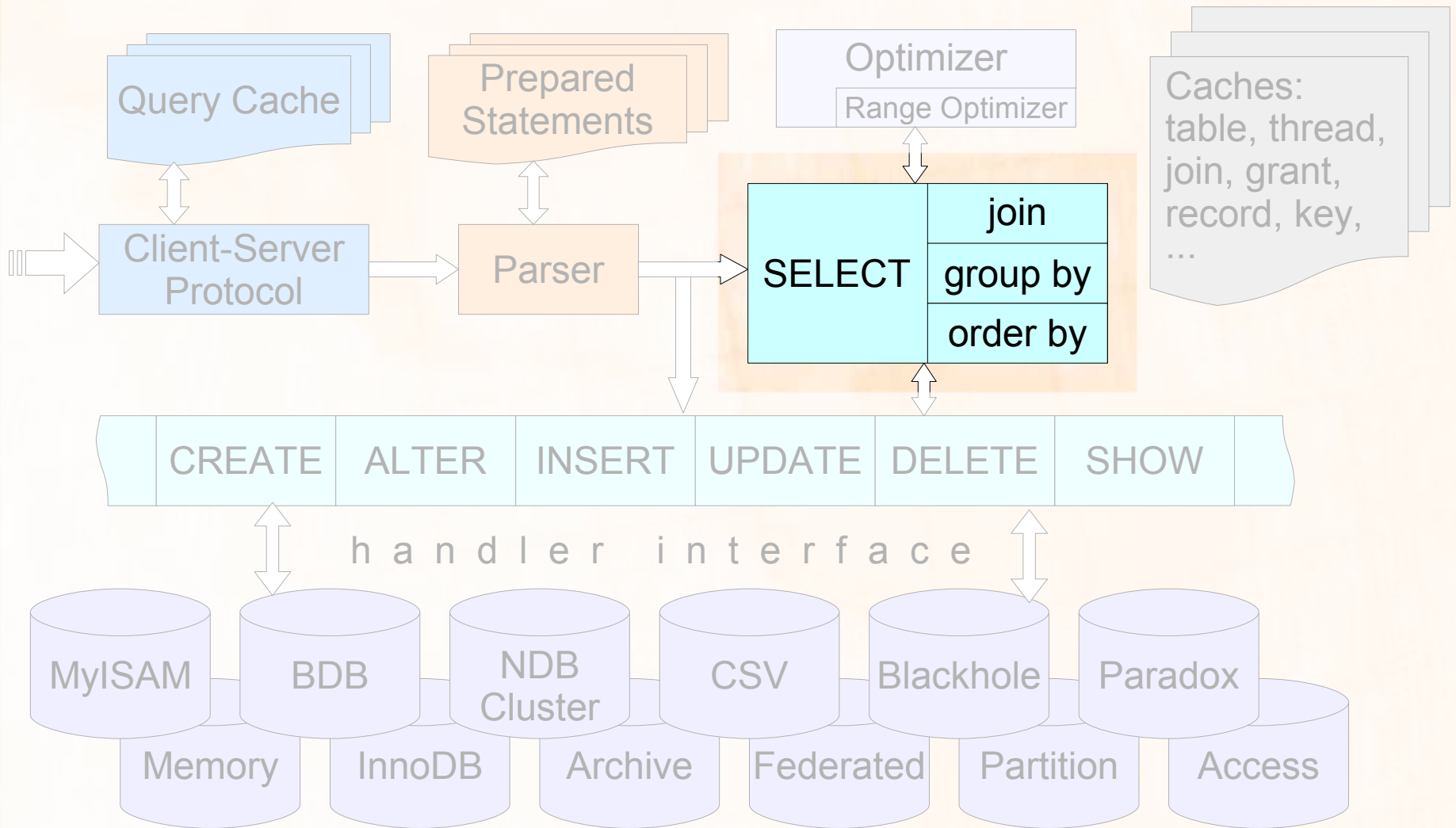
Query Life Cycle

Moment 3: Optimizing

7a. Access methods (“join types”)

- const `uniq_column=const`
- eq_ref `uniq_column=expr`
- ref `column=expr`
- index_merge `c1=expr1 OR/AND c2=expr2...`
- index full index scan
- ALL full table scan
- range range index scans

MySQL Architecture



Query Life Cycle

Moment 4: Executing

8. Join tables

- only nested-loop joins are implemented:
 - for every matching row in the first table
 - for every matched row in the second table
 - ... for every matched row in the last table
 - send results to the client
- “for every matched row” means
 - getting a next row (where “next” has different meaning for different join types)
 - checking the part of WHERE/ON condition that is applicable

Query Life Cycle

Moment 4: Executing

8a. Temporary table

- if optimizer decides it needs a temporary table, the nested-join loop populates this table, instead of sending the data to the client
- then the second run is performed fetching the data from temporary table
- temporary table can be used for aggregation (GROUP BY) or uniqueness (DISTINCT, UNION) – it has unique index created as necessary.

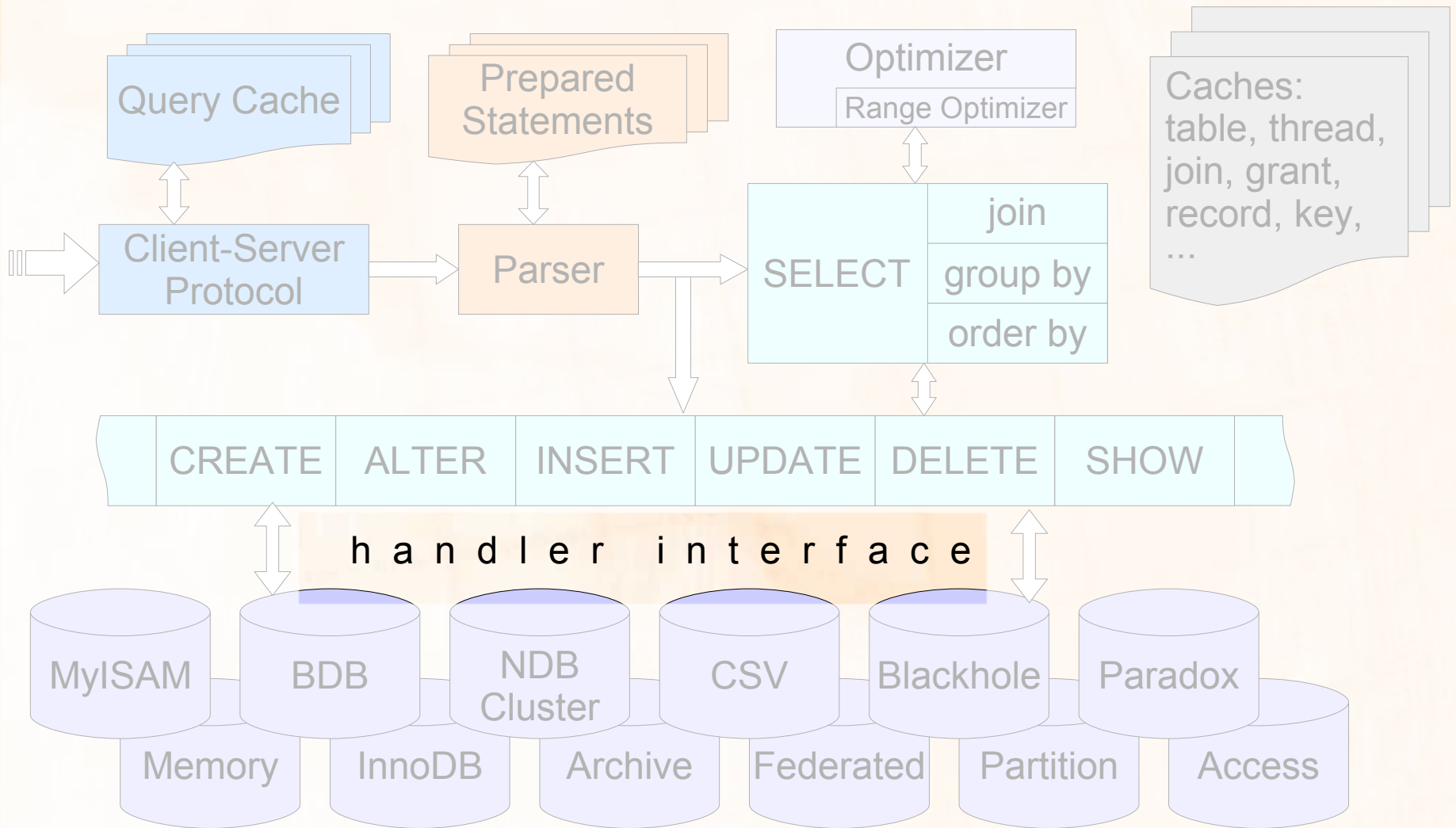
Query Life Cycle

Moment 4: Executing

9. Final step – sending data

- technically, this is not a separate step – result rows are sent as soon as they are found in the nested-join loop, row by row
- also, it does not necessarily involve sending
- class `select_result`
 - `select_send`
 - `select_export` `SELECT ... INTO OUTFILE`
 - `select_insert` `INSERT ... SELECT`
 - `select_exists_subselect, multi_delete ...`

MySQL Architecture



Handler API

- Low-level API to talk to a simple ISAM storage engine:
 - `write_row()`, `delete_row()`, `update_row()`, ...
 - `index_search()`, `index_next()`, ...
 - `rnd_init()`, `rnd_next()`, `rnd_pos()`, ...
- High-level API for complex storage engines
 - transactions, savepoints, two-phase commit
 - condition pushdown
 - table structure discovery

Handler API

- Opaque – MySQL provides a memory area to store private data:
 - connection local data
 - savepoint local data
 - “rowid” – used in `rnd_pos()`
- Transparent – handler has a lot of ways to tell about itself
 - capability flags
 - row/index access costs
 - upgrading/downgrading locks

Handler API

- Low Entry Barrier
 - Very few of the handler methods are mandatory to exist
 - About 10 methods to write for a simple indexless, read-only storage engine (and half of them are one-liners)
- But handlers can be very complex too
 - `ha_innodb` – 7739 lines of code
 - `ha_ndbcluster` – 8177 lines of code

Handler API

- class handler – one instance of the class per open table. Contains methods that acts on tables (open, close, write_row, etc)
- struct handlerton – singleton structure, only one instance per storage engine. Contains global methods such as commit, rollback, savepoint methods, init/deinit, etc.

Directories and Files in 5.1

- Clients and scripts
 - libmysql/ client library, C API
 - there's also thread-safe version in libmysql_r/
 - client/ C clients (mysql, mysqldump, ...)
 - extra/ misc. utilities (perror, comp_err, ...)
 - scripts/ mysql_install_db, mysqld_multi, ...
 - server-tools/ Instance Manager

Directories and Files in 5.1

- Server sources
 - config/ autoconf macros (*.m4)
 - msys/ portability wrappers (my_chsize), various algorithms (tree, hash, ...)
 - sql/ server kernel, handlers
 - storage/ storage engines (myisam, heap, merge, bdb, innodb, ndb)
 - strings/ charsets, xml, utility functions (strtol, strxmov, strmake, vsnprintf, ...)
 - vio/ “virtual i/o” library

Directories and Files in 5.1

- Third-party libraries
 - `cmd-line-utils/` GNU Readline and BSD libedit
 - `dbug/` Fred Fish's *C debugging package*
 - `extra/yassl/` Todd Ouska's *Yet Another SSL*
 - `regex/` Henry Spencer's regex
 - `zlib/` zlib
- Test suite
 - `mysql-test/`
- Embedded MySQL with examples
 - `libmysqld/`

Extending MySQL

What can be added

- Traditional options
 - A function
 - A procedure
 - Stored engine
- New in 5.1
 - Plugin API
 - Loadable storage engines
 - Fulltext parser
- Todo

Adding a Function

- A compiled-in function
 - allows to use statically-compiled mysqld binary
 - requires constant maintainance
 - MySQL sources have to be patched on every upgrade (and a patch won't apply cleanly, believe me)
 - difficult to distribute, less user-friendly
- An UDF
 - User-defined function, loaded dynamically
 - easier to maintain, but can crash when UDF API changes

Adding a Procedure

- A filter that it is put after the select but before the data are sent, can transform the result any way it wants, also change the number of columns and their types
- Only statically compiled
- Unfortunately, poorly documented at the moment

Plugin API

- New in 5.1
- Generic – allows to load any functionality in the running mysqld
- Built-in versioning
- Easy to maintain and distribute
- User friendly – no cryptic arguments to install
 - `INSTALL PLUGIN foo SONAME 'bar.so'`

Plugin Can:

- provide the code for mysqld to execute
- add new status variables
 - SHOW STATUS
- add new command-line options
 - --plugin-option (todo)
- add new server variables
 - SHOW VARIABLES, SET @@var (todo)
- add new SQL keywords
 - CREATE TABLE ... METHOD="deflate" (todo)

Plugin Types in 5.1

- Fulltext Search [pre-]parser
 - The code that changes the text before it is going into FULLTEXT index.
 - Can be used to search non-plaintext data formats, such as pdf, doc, mp3
 - Can be used to parse CJK texts
 - Can be used to use your own word splitting rules, apply stemming, thesaurus, anything
- Loadable Storage Engine

Future Plugin Types

- UDF (User Definable Function)
 - because of versioning, security, easy of use
- Language Modules for Stored Procedures
- PAM (Pluggable Authentication Module)
- New SQL commands, may be ? Who knows...

Questions ?