QINGCLOUD

MaterializeMySQL:
MySQL 到 ClickHouse 的高速公路

演讲人：邰翀

About Me

- Active contributor of ClickHouse & Xenon
  - MaterializeMySQL
  - MySQL Database Engine
  - And more 40+ features

https://tceason.github.io/

QINGCLOUD

目录

- MySQL 复制历程
- ClickHouse 同步 MySQL 数据
- MaterializeMySQL Engine
- HTAP 应用场景

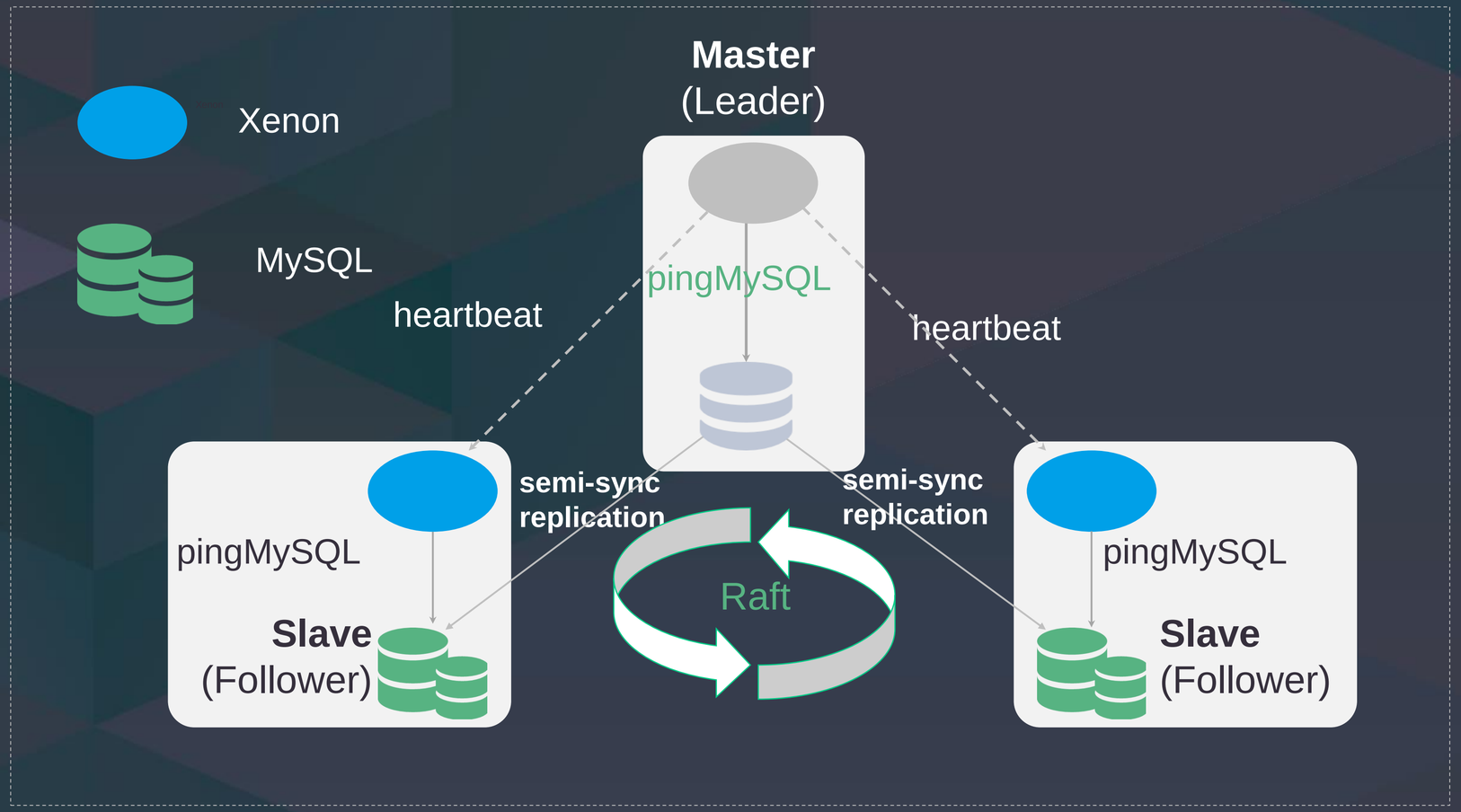
QINGCLOUD

MySQL 复制历程

MySQL 复制历程


MySQL 3.23 (2001) 开始支持复制
MySQL 5.1.5 (2006-01-10) binlog支持行模式（row-based）
MySQL 5.5.0 (2009-12-07) semi-sync replication
MySQL 5.6.0 (2011) delayed replication
MySQL 5.6.3 (2011-10-03) 基于库的并行复制
MySQL 5.6.5 (2012-04-10) GTID
MySQL 5.7.2 (2013-09-21) lossless replication
MySQL 5.7.5 (2014-09-25) mutli-source replication
MySQL 5.7.x 基于组提交（事务级）的并行复制
MySQL 5.7.17(2016-10-12) group replication（MGR）
MySQL 8.0.1(2017-04-10) group replication（MGR）、基于WriteSet（记录级）的并行复制
MySQL 5.7.22(2018-04-19) 基于WriteSet（记录级）的并行复制

MySQL 复制历程



QINGCLOUD

MySQL 复制历程

MySQL Plus

QINGCLOUD

MySQL 复制历程

集群数据强一致

主从秒级切换

无中心化自动选主

跨区容灾

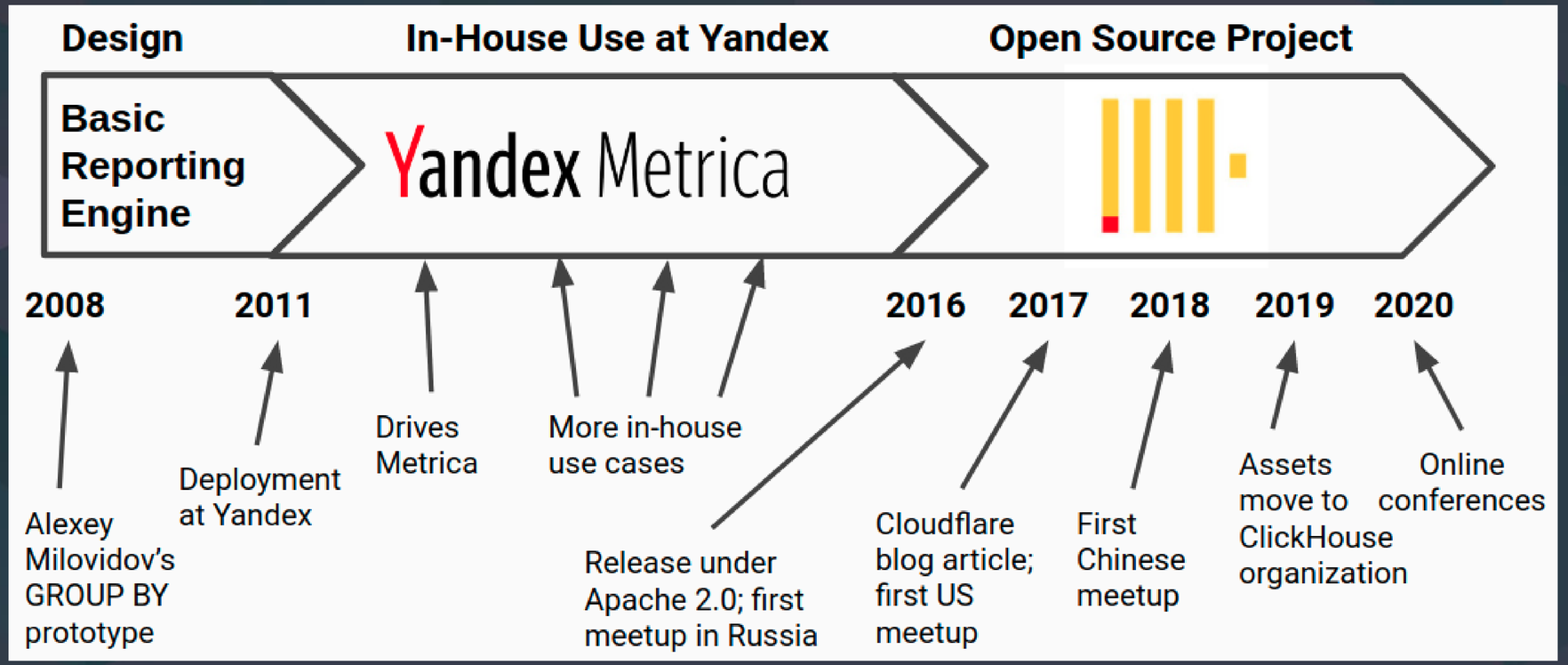
QINGCLOUD

但是，如何加速 OLAP 查询？？？

QINGCLOUD

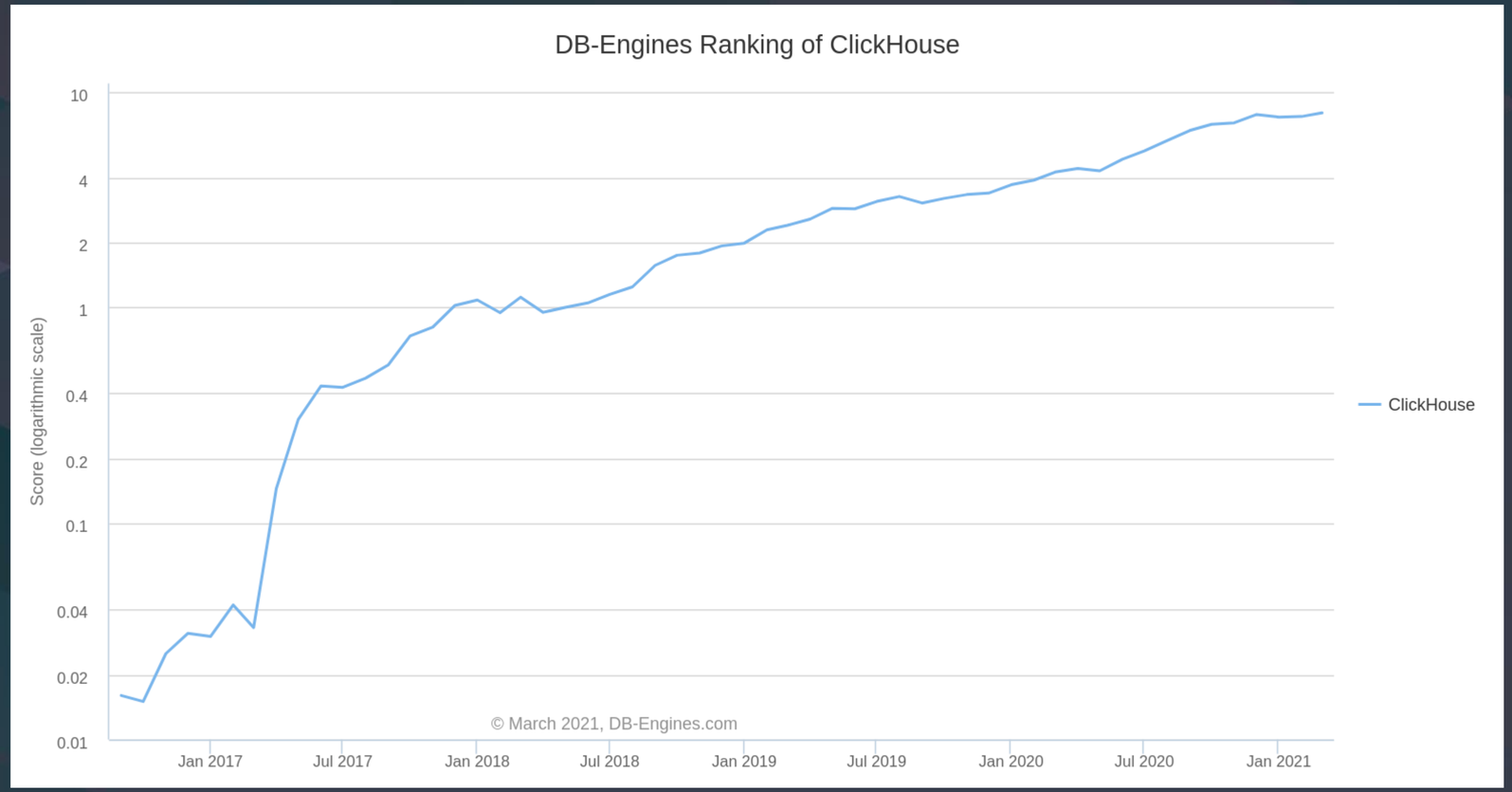
ClickHouse 同步 MySQL 数据

ClickHouse 同步 MySQL 数据



QINGCLOUD

ClickHouse 同步 MySQL 数据



QINGCLOUD

ClickHouse 同步 MySQL 数据

- MySQL Table Engine
  - Mapping to MySQL table
  - Fetch table struct from MySQL
  - Fetch data from MySQL when execute query

QINGCLOUD

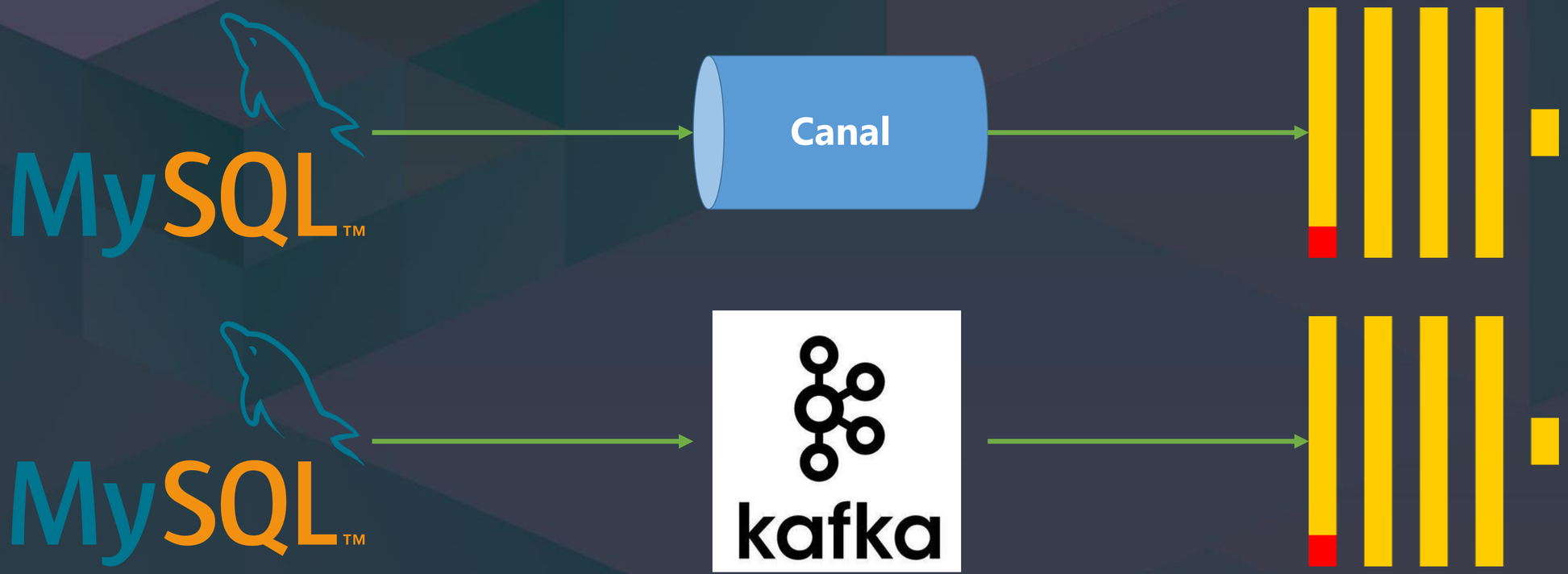
ClickHouse 同步 MySQL 数据

- MySQL Database Engine
  - Mapping to MySQL Database
  - Fetch table list from MySQL
  - Fetch table struct from MySQL
  - Fetch data from MySQL when execute query

QINGCLOUD

ClickHouse 同步 MySQL 数据

- Some others



QINGCLOUD

ClickHouse

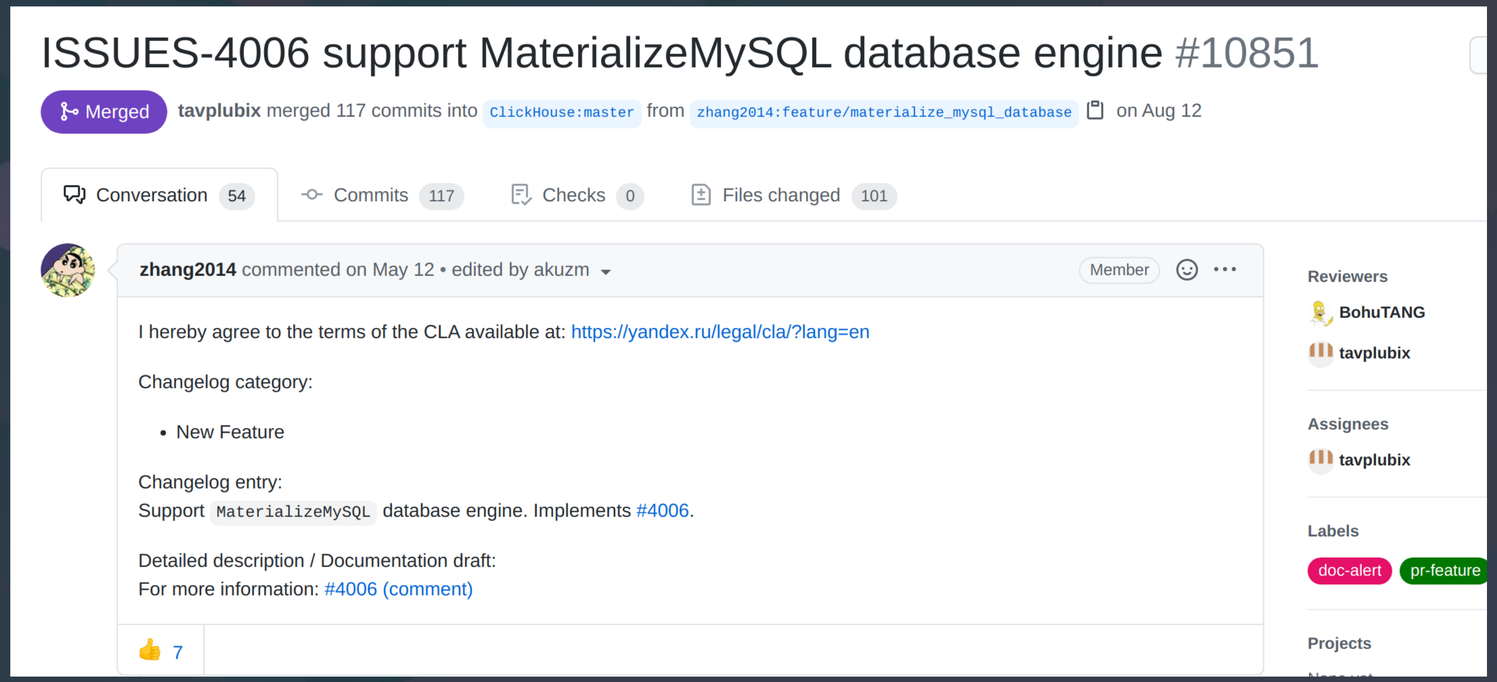
能否主动同步并订阅

MySQL ？？？

QINGCLOUD

MaterializeMySQL Engine

MaterializeMySQL Engine



ISSUES-4006 support MaterializeMySQL database engine #10851

Merged **tavplubix** merged 117 commits into `ClickHouse:master` from `zhang2014:feature/materialize_mysql_database` on Aug 12

Conversation 54 | Commits 117 | Checks 0 | Files changed 101

zhang2014 commented on May 12 • edited by akuzm — Member

I hereby agree to the terms of the CLA available at: https://yandex.ru/legal/cla/?lang=en

Changelog category:

- New Feature

Changelog entry:
Support `MaterializeMySQL` database engine. Implements #4006.

Detailed description / Documentation draft:
For more information: #4006 (comment)

👍 7

Reviewers
BohuTANG
tavplubix

Assignees
tavplubix

Labels
doc-alert pr-feature

Projects

由青云ClickHouse 团队贡献

https://github.com/ClickHouse/ClickHouse/pull/10851

QINGCLOUD

MaterializeMySQL Engine

- • Mapping to MySQL database
- • Consume MySQL BINLOG and store to MergeTree
- • Experimental feature (20.8, recommend latest stable version)

QINGCLOUD

MaterializeMySQL Engine

QINGCLOUD

MaterializeMySQL Engine


```
CREATE DATABASE test
ENGINE =
MaterializeMySQL('172.17.0.3:3306',
'demo', 'root', '123')
```

QINGCLOUD

MaterializeMySQL Engine

History data

New data

QINGCLOUD

MaterializeMySQL Engine

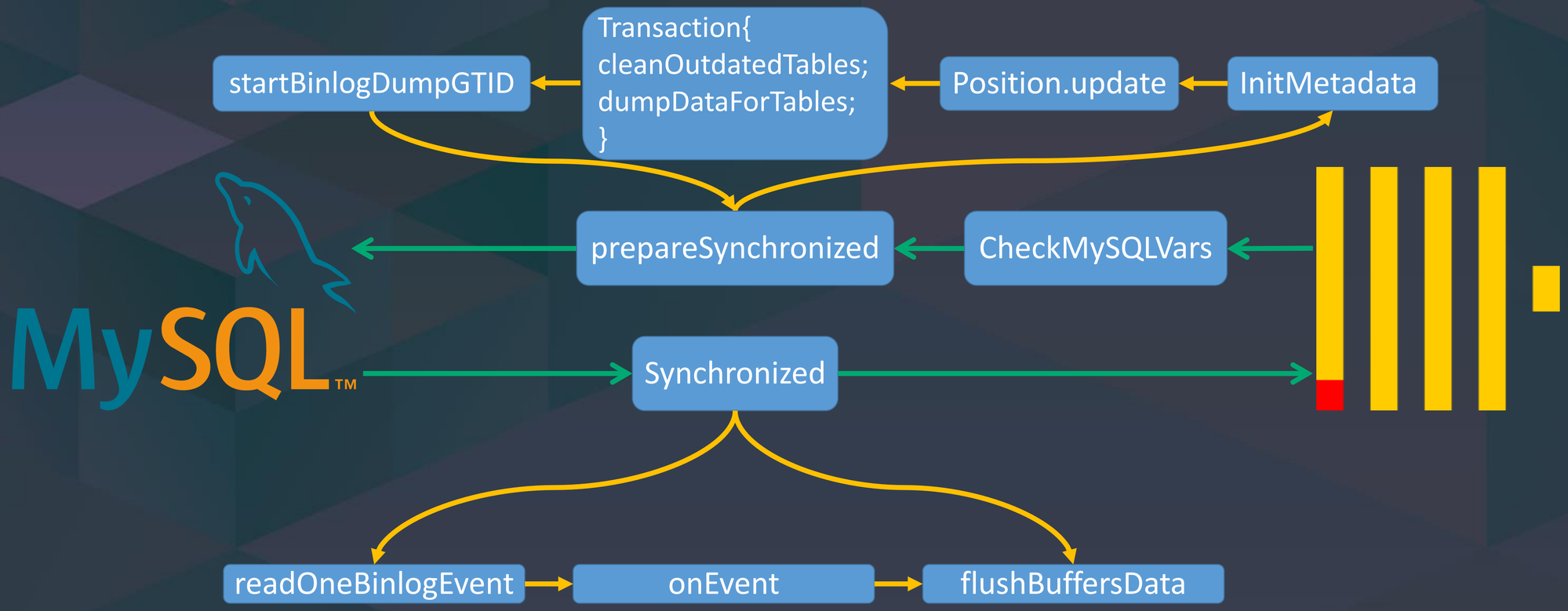
Check MySQL Vars

Select history data

Consume new data

QINGCLOUD

MaterializeMySQL Engine



QINGCLOUD

MaterializeMySQL Engine

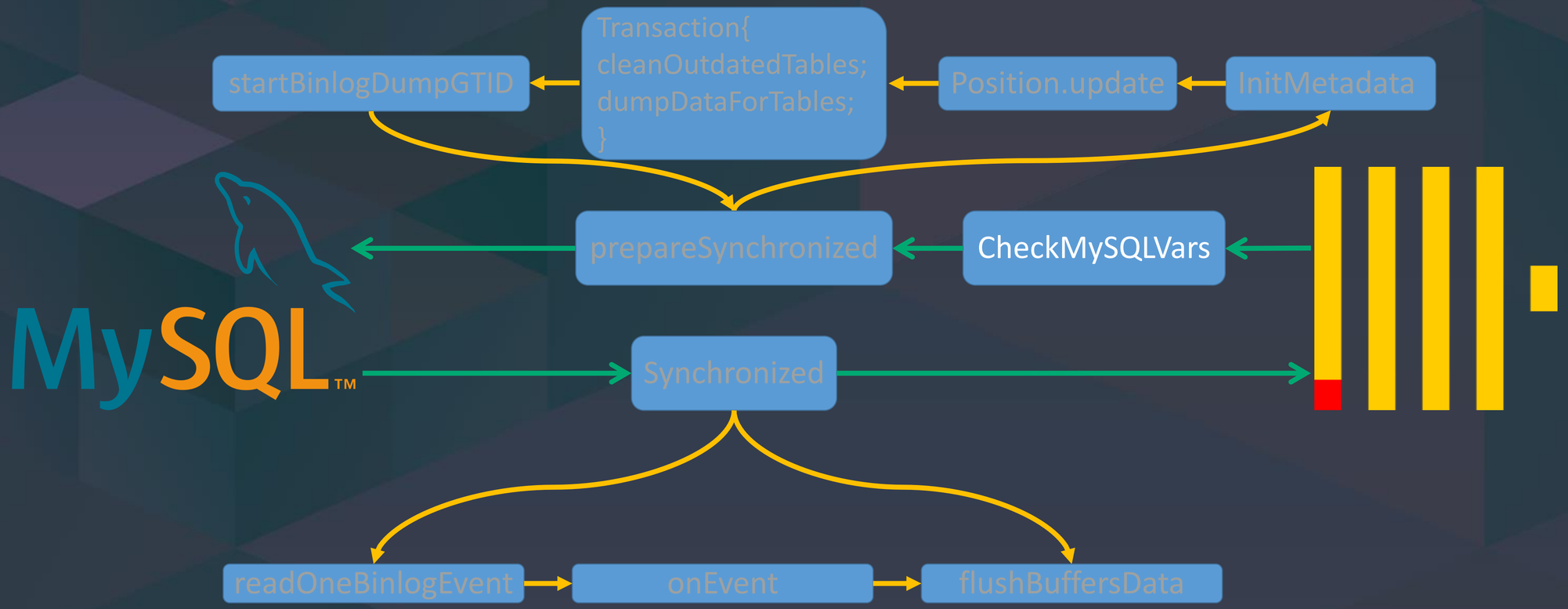
Check MySQL Vars

Select history data

Consume new data

QINGCLOUD

MaterializeMySQL Engine



QINGCLOUD

MaterializeMySQL Engine

Check MySQL Vars

```
SHOW VARIABLES WHERE (Variable_name = 'log_bin'
        AND upper(Value) = 'ON')
OR (Variable_name = 'binlog_format'
        AND upper(Value) = 'ROW')
OR (Variable_name = 'binlog_row_image'
        AND upper(Value) = 'FULL')
OR (Variable_name = 'default_authentication_plugin'
        AND upper(Value) = 'MYSQL_NATIVE_PASSWORD')
OR (Variable_name = 'log_bin_use_v1_row_events'
        AND upper(Value) = 'OFF');
```

QINGCLOUD

MaterializeMySQL Engine

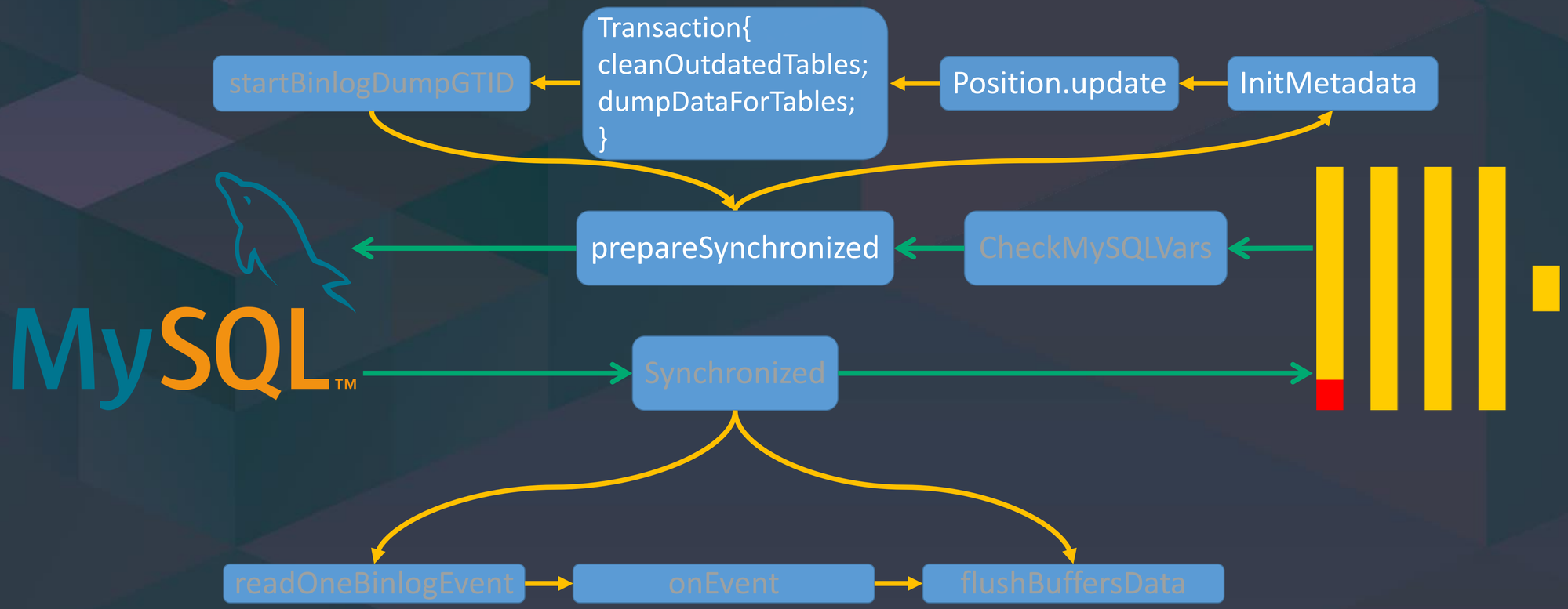
Check MySQL Vars

Select history data

Consume new data

QINGCLOUD

MaterializeMySQL Engine



QINGCLOUD

MaterializeMySQL Engine

Select history data

```
std::optional<MaterializeMetadata> MaterializeMySQLSyncThread::prepareSynchronized()
{
    connection = pool.get();

    MaterializeMetadata metadata(
        connection, DatabaseCatalog::instance().getDatabase(database_name)->getMetadataPath() + "/.metadata",
mysql_database_name, opened_transaction);
    if (!metadata.need_dumping_tables.empty())
    {
        Position position;
        position.update(metadata.binlog_position, metadata.binlog_file, metadata.executed_gtid_set);

        metadata.transaction(position, [&]()
        {
            cleanOutdatedTables(database_name, global_context);
            dumpDataForTables(connection, metadata, query_prefix, database_name, mysql_database_name, global_context, [this] {
return isCancelled(); });
        });
    }
    connection->query("COMMIT").execute();
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
std::optional<MaterializeMetadata> MaterializeMySQLSyncThread::prepareSynchronized()
{
    connection = pool.get();

    MaterializeMetadata metadata(
        connection, DatabaseCatalog::instance().getDatabase(database_name)->getMetadataPath() + "/.metadata",
mysql_database_name, opened_transaction);
    if (!metadata.need_dumping_tables.empty())
    {
        Position position;
        position.update(metadata.binlog_position, metadata.binlog_file, metadata.executed_gtid_set);

        metadata.transaction(position, [&]()
        {
            cleanOutdatedTables(database_name, global_context);
            dumpDataForTables(connection, metadata, query_prefix, database_name, mysql_database_name, global_context, [this] {
return isCancelled(); });
        });
    }
    connection->query("COMMIT").execute();
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
connection->query("FLUSH TABLES;").execute();
connection->query("FLUSH TABLES WITH READ LOCK;").execute();

fetchMasterStatus(connection);
fetchMasterVariablesValue(connection);
connection->query("SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;").execute();
connection->query("START TRANSACTION /*!40100 WITH CONSISTENT SNAPSHOT */;").execute();

opened_transaction = true;
need_dumping_tables = fetchTablesCreateQuery(connection, database, fetchTablesInDB(connection, database));
connection->query("UNLOCK TABLES;").execute();
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
std::optional<MaterializeMetadata> MaterializeMySQLSyncThread::prepareSynchronized()
{
    connection = pool.get();

    MaterializeMetadata metadata(
        connection, DatabaseCatalog::instance().getDatabase(database_name)->getMetadataPath() + "/.metadata",
mysql_database_name, opened_transaction);
    if (!metadata.need_dumping_tables.empty())
    {
        Position position;
        position.update(metadata.binlog_position, metadata.binlog_file, metadata.executed_gtid_set);

        metadata.transaction(position, [&]()
        {
            cleanOutdatedTables(database_name, global_context);
            dumpDataForTables(connection, metadata, query_prefix, database_name, mysql_database_name, global_context, [this] {
return isCancelled(); });
        });
    }
    connection->query("COMMIT").execute();
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
void MaterializeMetadata::transaction(const MySQLReplication::Position & position, const std::function<void()> & fun)
{

    String persistent_tmp_path = persistent_path + ".tmp";

    {
        WriteBufferFromFile out(persistent_tmp_path, DBMS_DEFAULT_BUFFER_SIZE, O_WRONLY | O_TRUNC | O_CREAT);

        /// TSV format metadata file.
        writeString("Version:\t" + toString(meta_version), out);
        writeString("\nBinlog File:\t" + binlog_file, out);
        writeString("\nExecuted GTID:\t" + executed_gtid_set, out);
        writeString("\nBinlog Position:\t" + toString(binlog_position), out);
        writeString("\nData Version:\t" + toString(data_version), out);

        out.next();
        out.sync();
        out.close();
    }
    commitMetadata(std::move(fun), persistent_tmp_path, persistent_path);
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
std::optional<MaterializeMetadata> MaterializeMySQLSyncThread::prepareSynchronized()
{
    connection = pool.get();

    MaterializeMetadata metadata(
        connection, DatabaseCatalog::instance().getDatabase(database_name)->getMetadataPath() + "/.metadata",
mysql_database_name, opened_transaction);
    if (!metadata.need_dumping_tables.empty())
    {
        Position position;
        position.update(metadata.binlog_position, metadata.binlog_file, metadata.executed_gtid_set);

        metadata.transaction(position, [&]()
        {
            cleanOutdatedTables(database_name, global_context);
            dumpDataForTables(connection, metadata, query_prefix, database_name, mysql_database_name, global_context, [this] {
return isCancelled(); });
        });
    }
    connection->query("COMMIT").execute();
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
auto iterator = master_info.need_dumping_tables.begin();

for (; iterator != master_info.need_dumping_tables.end() && !is_cancelled(); ++iterator)
{
    const auto & table_name = iterator->first;
    Context query_context = createQueryContext(context);
    String comment = "Materialize MySQL step 1: execute MySQL DDL for dump data";
    tryToExecuteQuery(query_prefix + " " + iterator->second, query_context, database_name, comment); /// create table.

    auto out = std::make_shared<CountingBlockOutputStream>(getTableOutput(database_name, table_name, query_context));
    MySQLBlockInputStream input(
        connection, "SELECT * FROM " + backQuoteIfNeed(mysql_database_name) + "." + backQuoteIfNeed(table_name),
        out->getHeader(), DEFAULT_BLOCK_SIZE);

    Stopwatch watch;
    copyData(input, *out, is_cancelled);
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
auto iterator = master_info.need_dumping_tables.begin();

for (; iterator != master_info.need_dumping_tables.end() && !is_cancelled(); ++iterator)
{
    const auto & table_name = iterator->first;
    Context query_context = createQueryContext(context);
    String comment = "Materialize MySQL step 1: execute MySQL DDL for dump data";
    tryToExecuteQuery(query_prefix + " " + iterator->second, query_context, database_name, comment); /// create table.

    auto out = std::make_shared<CountingBlockOutputStream>(getTableOutput(database_name, table_name, query_context));
    MySQLBlockInputStream input(
        connection, "SELECT * FROM " + backQuoteIfNeed(mysql_database_name) + "." + backQuoteIfNeed(table_name),
        out->getHeader(), DEFAULT_BLOCK_SIZE);

    Stopwatch watch;
    copyData(input, *out, is_cancelled);
}
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
const auto & create_defines = create_query.columns_list->as<MySQLParser::ASTCreateDefines>();
NamesAndTypesList columns_name_and_type = getColumnsList(create_defines->columns);
const auto & [primary_keys, unique_keys, keys, increment_columns] = getKeys(create_defines->columns, create_defines->indices, context,
columns_name_and_type);


auto columns = std::make_shared<ASTColumns>();
columns->set(columns->columns, InterpreterCreateQuery::formatColumns(columns_name_and_type));
columns->columns->children.emplace_back(create_materialized_column_declaration(sign_column_name, "Int8", UInt64(1)));
columns->columns->children.emplace_back(create_materialized_column_declaration(version_column_name, "UInt64", UInt64(1)));

auto storage = std::make_shared<ASTStorage>();

if (ASTPtr partition_expression = getPartitionPolicy(primary_keys))
    storage->set(storage->partition_by, partition_expression);

if (ASTPtr order_by_expression = getOrderByPolicy(primary_keys, unique_keys, keys, increment_columns))
    storage->set(storage->order_by, order_by_expression);

storage->set(storage->engine, makeASTFunction("ReplacingMergeTree", std::make_shared<ASTIdentifier>(version_column_name)));
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
CREATE TABLE `demo`.`t` (
  `id` int(11) NOT NULL,
  `col_1` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

QINGCLOUD

MaterializeMySQL Engine

Select history data

```
CREATE TABLE test.t
(
    `id` Int32,
    `col_1` Nullable(String),
    `_sign` Int8 MATERIALIZED 1,
    `_version` UInt64 MATERIALIZED 1
)
ENGINE = ReplacingMergeTree(_version)
PARTITION BY intDiv(id, 4294967)
ORDER BY tuple(id)
```

QINGCLOUD

MaterializeMySQL Engine

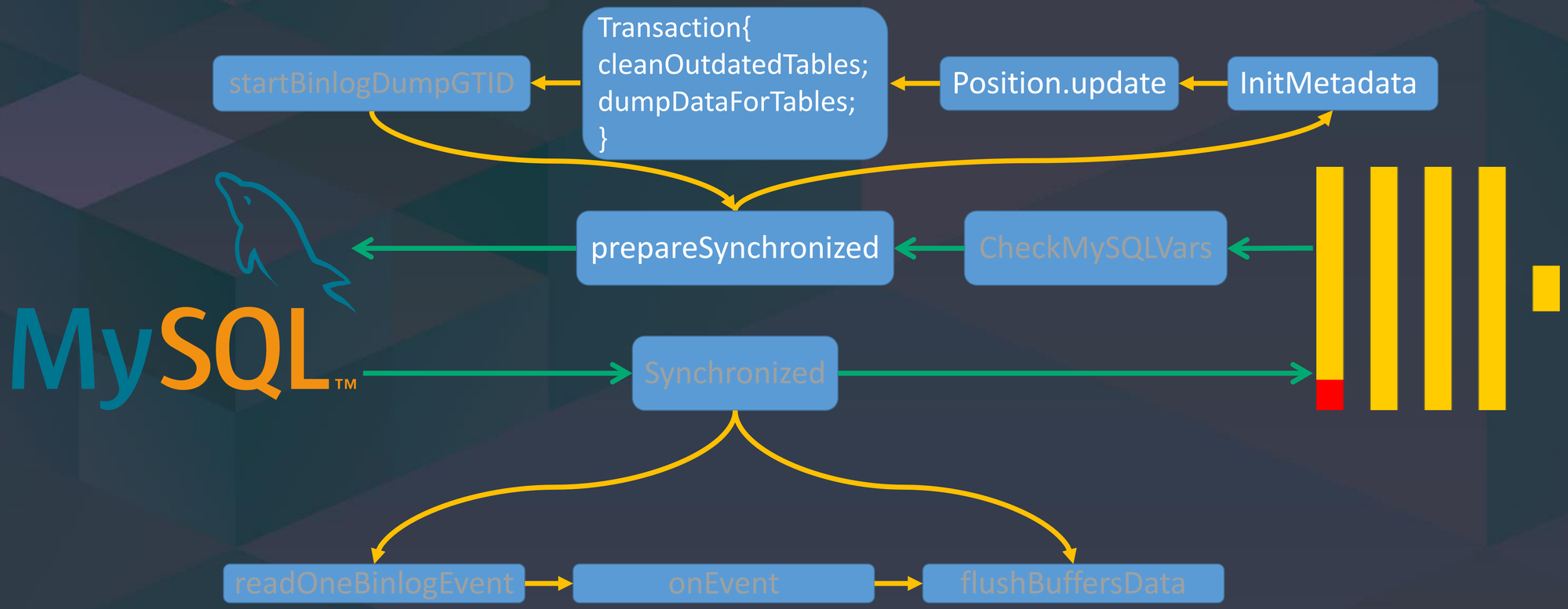
Check MySQL Vars

Select history data

Consume new data

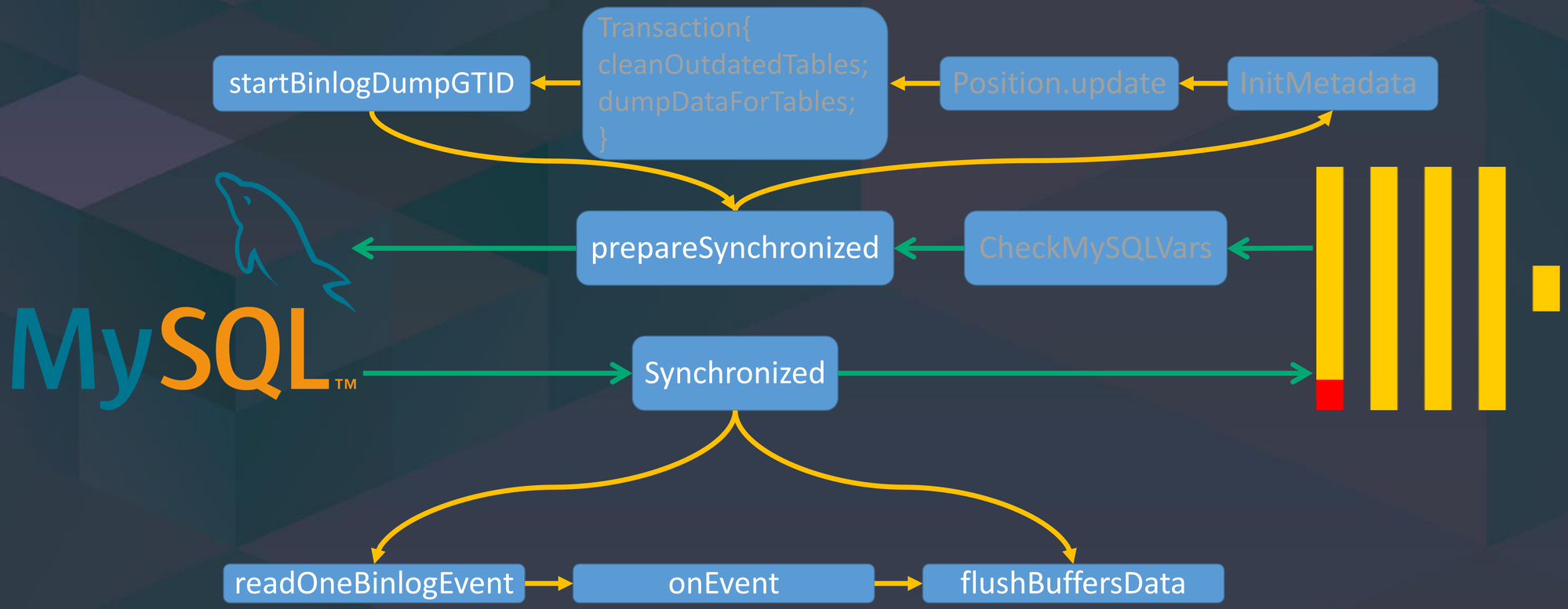
QINGCLOUD

MaterializeMySQL Engine



QINGCLOUD

MaterializeMySQL Engine



QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
client.connect();
client.startBinlogDumpGTID(randomNumber(), mysql_database_name, metadata.executed_gtid_set,
metadata.binlog_checksum);

Buffers buffers(database_name);
while (!isCancelled())
{
        BinlogEventPtr binlog_event = client.readOneBinlogEvent(std::max(UInt64(1), max_flush_time -
watch.elapsedMilliseconds()));
        if (binlog_event)
                onEvent(buffers, binlog_event, *metadata);
        if (!buffers.data.empty())
                flushBuffersData(buffers, *metadata);
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
client.connect();
client.startBinlogDumpGTID(randomNumber(), mysql_database_name,
metadata.executed_gtid_set, metadata.binlog_checksum);

Buffers buffers(database_name);
while (!isCancelled())
{
    BinlogEventPtr binlog_event = client.readOneBinlogEvent(std::max(UInt64(1),
max_flush_time - watch.elapsedMilliseconds()));
    if (binlog_event)
        onEvent(buffers, binlog_event, *metadata);
    if (!buffers.data.empty())
        flushBuffersData(buffers, *metadata);
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
client.connect();
client.startBinlogDumpGTID(randomNumber(), mysql_database_name, metadata.executed_gtid_set,
metadata.binlog_checksum);

Buffers buffers(database_name);
while (!isCancelled())
{
    BinlogEventPtr binlog_event = client.readOneBinlogEvent(std::max(UInt64(1), max_flush_time -
watch.elapsedMilliseconds()));
    if (binlog_event)
        onEvent(buffers, binlog_event, *metadata);
    if (!buffers.data.empty())
        flushBuffersData(buffers, *metadata);
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::onEvent(Buffers & buffers, const BinlogEventPtr & receive_event,
MaterializeMetadata & metadata)
{
    if (receive_event->type() == MYSQL_WRITE_ROWS_EVENT)
    { ... }
    else if (receive_event->type() == MYSQL_UPDATE_ROWS_EVENT)
    { ... }
    else if (receive_event->type() == MYSQL_DELETE_ROWS_EVENT)
    { ... }
    else if (receive_event->type() == MYSQL_QUERY_EVENT)
    { ... }
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::onEvent(Buffers & buffers, const BinlogEventPtr & receive_event,
MaterializeMetadata & metadata)
{
    if (receive_event->type() == MYSQL_WRITE_ROWS_EVENT)
    {
        WriteRowsEvent & write_rows_event = static_cast<WriteRowsEvent &>(*receive_event);
        Buffers::BufferAndSortingColumnsPtr buffer = buffers.getTableDataBuffer(write_rows_event.table,
global_context);
        size_t bytes = onWriteOrDeleteData<1>(write_rows_event.rows, buffer->first,
++metadata.data_version);
        buffers.add(buffer->first.rows(), buffer->first.bytes(), write_rows_event.rows.size(), bytes);
    }
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::onEvent(Buffers & buffers, const BinlogEventPtr & receive_event,
MaterializeMetadata & metadata)
{
    if (receive_event->type() == MYSQL_WRITE_ROWS_EVENT)
    { ... }
    else if (receive_event->type() == MYSQL_UPDATE_ROWS_EVENT)
    { ... }
    else if (receive_event->type() == MYSQL_DELETE_ROWS_EVENT)
    {
        DeleteRowsEvent & delete_rows_event = static_cast<DeleteRowsEvent &>(*receive_event);
        Buffers::BufferAndSortingColumnsPtr buffer = buffers.getTableDataBuffer(delete_rows_event.table,
global_context);
        size_t bytes = onWriteOrDeleteData<-1>(delete_rows_event.rows, buffer->first,
++metadata.data_version);
        buffers.add(buffer->first.rows(), buffer->first.bytes(), delete_rows_event.rows.size(), bytes);
    }
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::onEvent(Buffers & buffers, const BinlogEventPtr & receive_event,
MaterializeMetadata & metadata)
{
    else if (receive_event->type() == MYSQL_UPDATE_ROWS_EVENT) {
        for (size_t index = 0; index < rows_data.size(); index += 2) {
            if (likely(!writeable_rows_mask[index])) {
                sign_column_data.emplace_back(1);
                version_column_data.emplace_back(version); }
            else {
                /// If the sorting keys is modified, we should cancel the old data, but this should not happen frequently
                sign_column_data.emplace_back(-1);
                sign_column_data.emplace_back(1);
                version_column_data.emplace_back(version);
                version_column_data.emplace_back(version);}}}}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::onEvent(Buffers & buffers, const BinlogEventPtr & receive_event,
MaterializeMetadata & metadata)
{   else if (receive_event->type() == MYSQL_QUERY_EVENT)
    {
        QueryEvent & query_event = static_cast<QueryEvent &>(*receive_event);
        position_before_ddl.update(metadata.binlog_position, metadata.binlog_file,
metadata.executed_gtid_set);
        metadata.transaction(position_before_ddl, [&]() { buffers.commit(global_context); });
        if (query->as<ASTDropQuery>())
            ...
        else if (query->as<ASTRenameQuery>())
            ...
        else if (query->as<MySQLParser::ASTAlterQuery>())
            ...
        else if (query->as<MySQLParser::ASTCreateQuery>())
            ...
    }
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
client.connect();
client.startBinlogDumpGTID(randomNumber(), mysql_database_name, metadata.executed_gtid_set,
metadata.binlog_checksum);

Buffers buffers(database_name);
while (!isCancelled())
{
    BinlogEventPtr binlog_event = client.readOneBinlogEvent(std::max(UInt64(1), max_flush_time -
watch.elapsedMilliseconds()));
    if (binlog_event)
        onEvent(buffers, binlog_event, *metadata);
    if (!buffers.data.empty())
        flushBuffersData(buffers, *metadata);
}
```

QINGCLOUD

MaterializeMySQL Engine

Consume new data

```
void MaterializeMySQLSyncThread::flushBuffersData(Buffers & buffers, MaterializeMetadata &
metadata)
{
    metadata.transaction(client.getPosition(), [&]() { buffers.commit(global_context); });

    const auto & position_message = [&]()
    {
        WriteBufferFromOwnString buf;
        client.getPosition().dump(buf);
        return buf.str();
    };
    LOG_INFO(log, "MySQL executed position: \n {}", position_message());
}
```

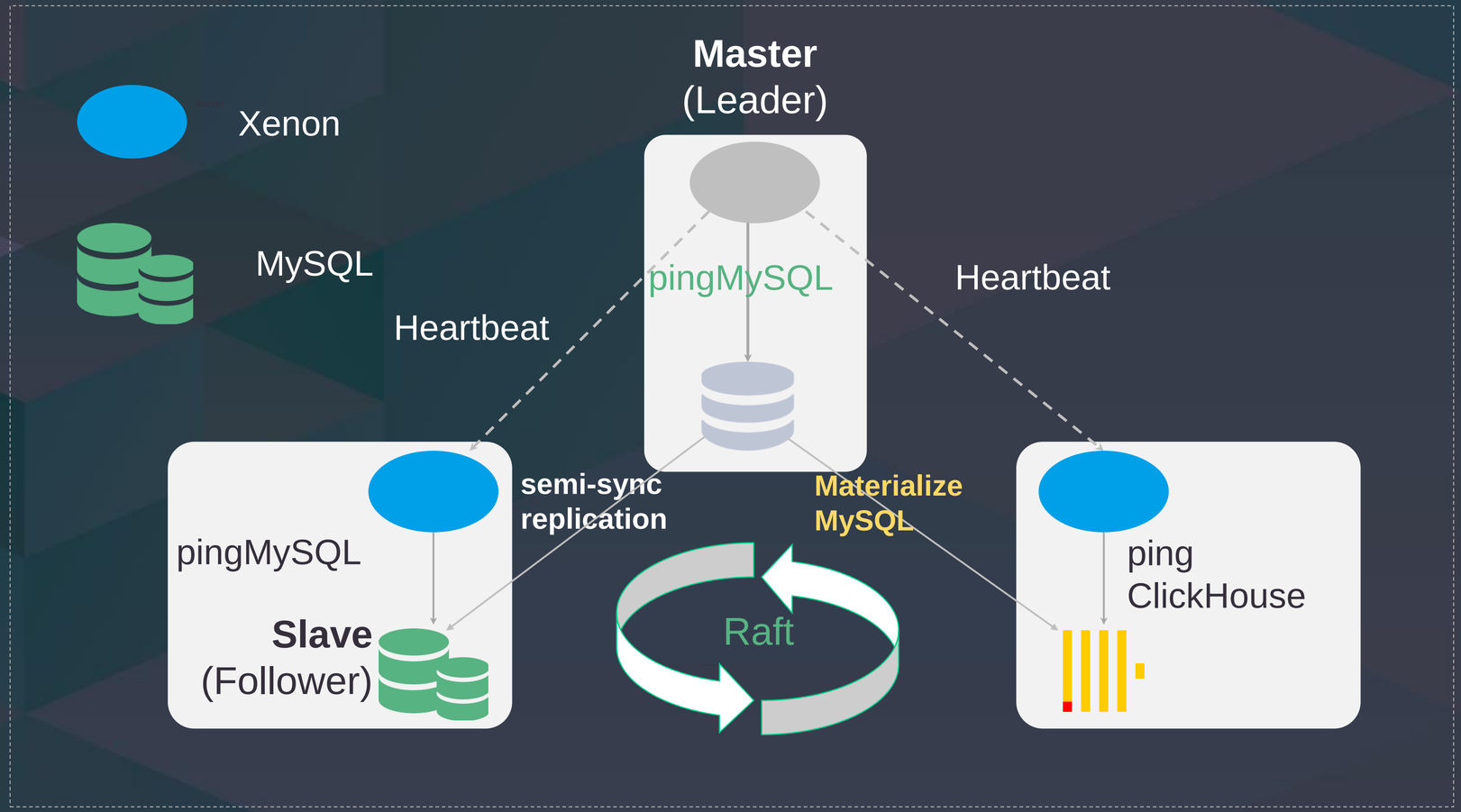
QINGCLOUD

HTAP 应用场景

MySQL Plus + ClickHouse

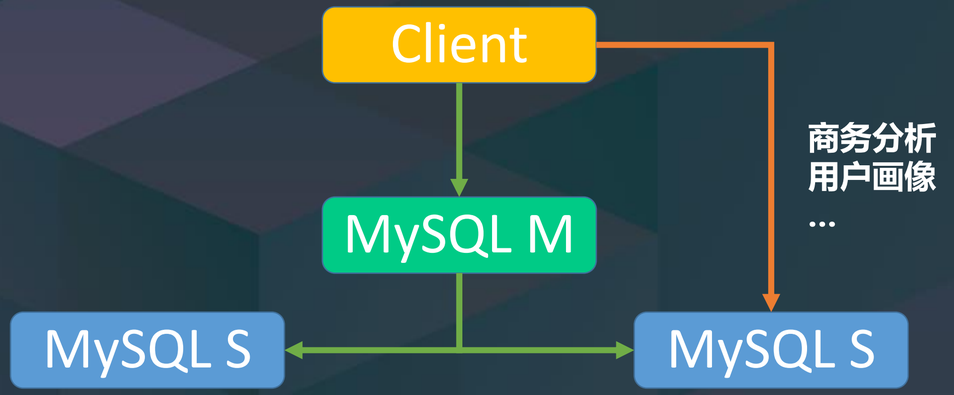
QINGCLOUD

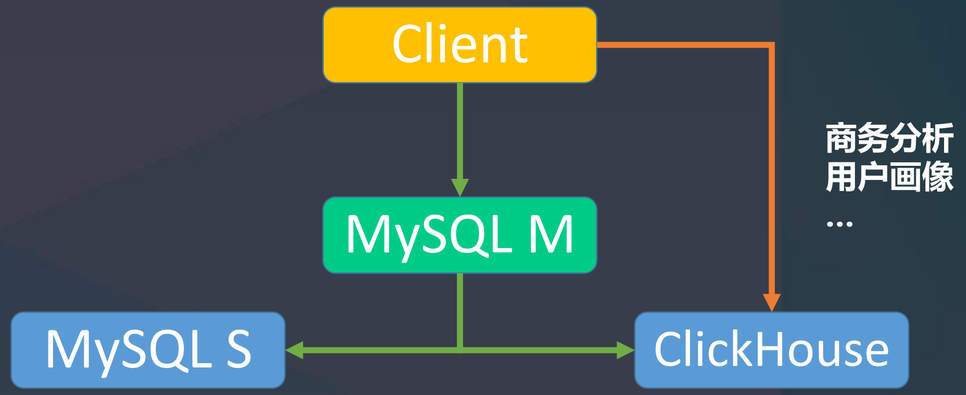
HTAP 应用场景



QINGCLOUD

HTAP 应用场景





QINGCLOUD


MySQL Plus
关系型数据库
RadonDB
分布式数据库
PostgreSQL
关系型数据库
数据库
DATABASE
ClickHouse
分析型数据库
PolonDB
分布式关系型数据库
ChronusDB
时序数据库
MongoDB
非关系型数据库
Redis
非关系型数据库
QINGCLOUD

THANKS FOR WATCHING

QINGCLOUD