# Quick Tour of ClickHouse Internals

Aleksey Zatelepin

Yandex


v1.2

# ClickHouse use cases

A stream of events

- Actions of website visitors Yandex Metrica
- Ad impressions
- Financial transactions
- DNS queries
- ...

> We want to save info about these events and then glean some insights from it

# ClickHouse philosophy

- Interactive queries on data updated in real time
- Cleaned structured data is needed
- Try hard not to pre-aggregate anything
- Query language: a dialect of SQL + extensions

# Sample query in a web analytics system

Top-10 referers for a website for the last week.

```
SELECT Referer, count(*) AS count
FROM hits
WHERE CounterID = 111 AND Date >= today() - 7
GROUP BY Referer
ORDER BY count DESC
LIMIT 10
```

# How to execute a query *fast*?

## Read data fast

- Only needed columns: `CounterID, Date, Referer`
- Locality of reads (an index is needed!)
- Data compression

# How to execute a query *fast*?

## Read data fast

- Only needed columns: `CounterID, Date, Referer`
- Locality of reads (an index is needed!)
- Data compression

## Process data fast

- Vectorized execution (block-based processing)
- Parallelize to all available cores and machines
- Specialization and low-level optimizations

# Index needed!

## The principle is the same as with classic DBMSes

A majority of queries will contain conditions on CounterID and (possibly) Date

## `(CounterID, Date)` fits the bill

Check this by mentally sorting the table by primary key

## Differences

- The table will be physically sorted on disk
- Is not a unique constraint

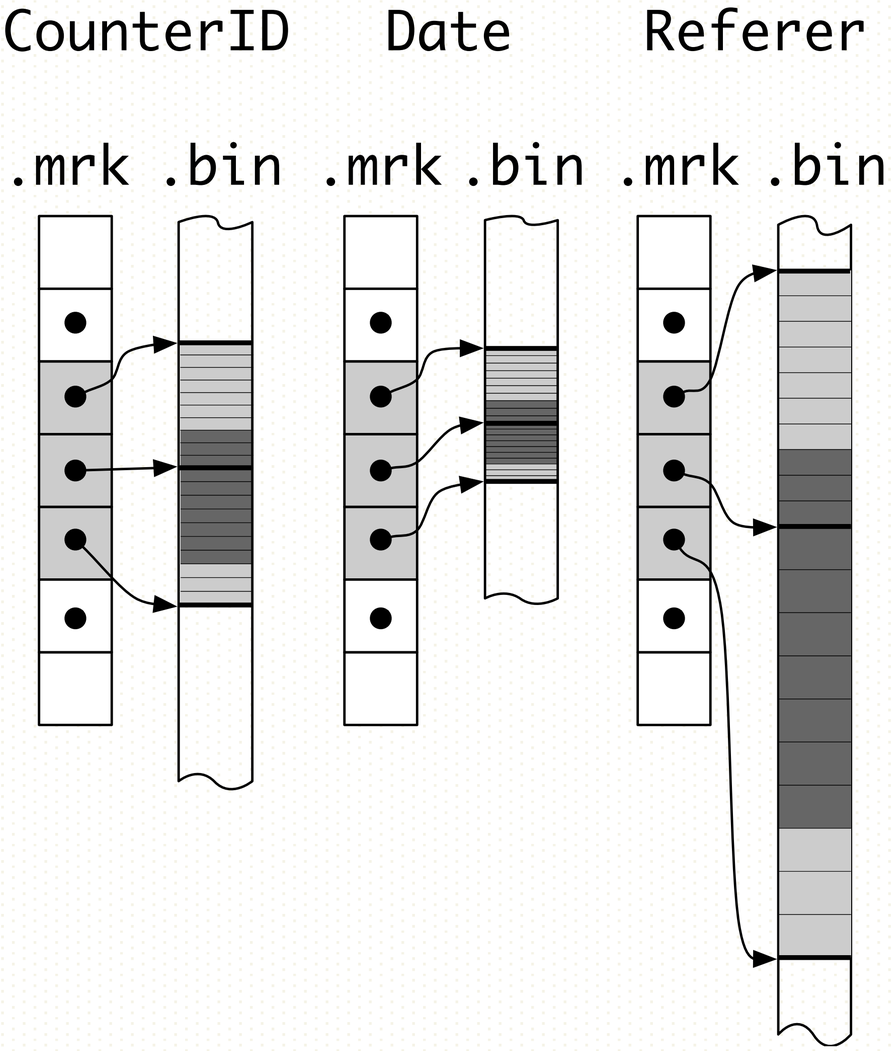
# Index internals

(CounterID, Date)

primary.idx

| | | |
|---|---|---|
| | … | … |
| | 111 | 2017-09-11 |
| N | 111 | 2017-09-19 |
| N+8192 | 111 | 2017-09-27 |
| N+16384 | 222 | 2013-02-16 |
| | 222 | 2013-03-12 |
| | … | … |

(One entry each 8192 rows)



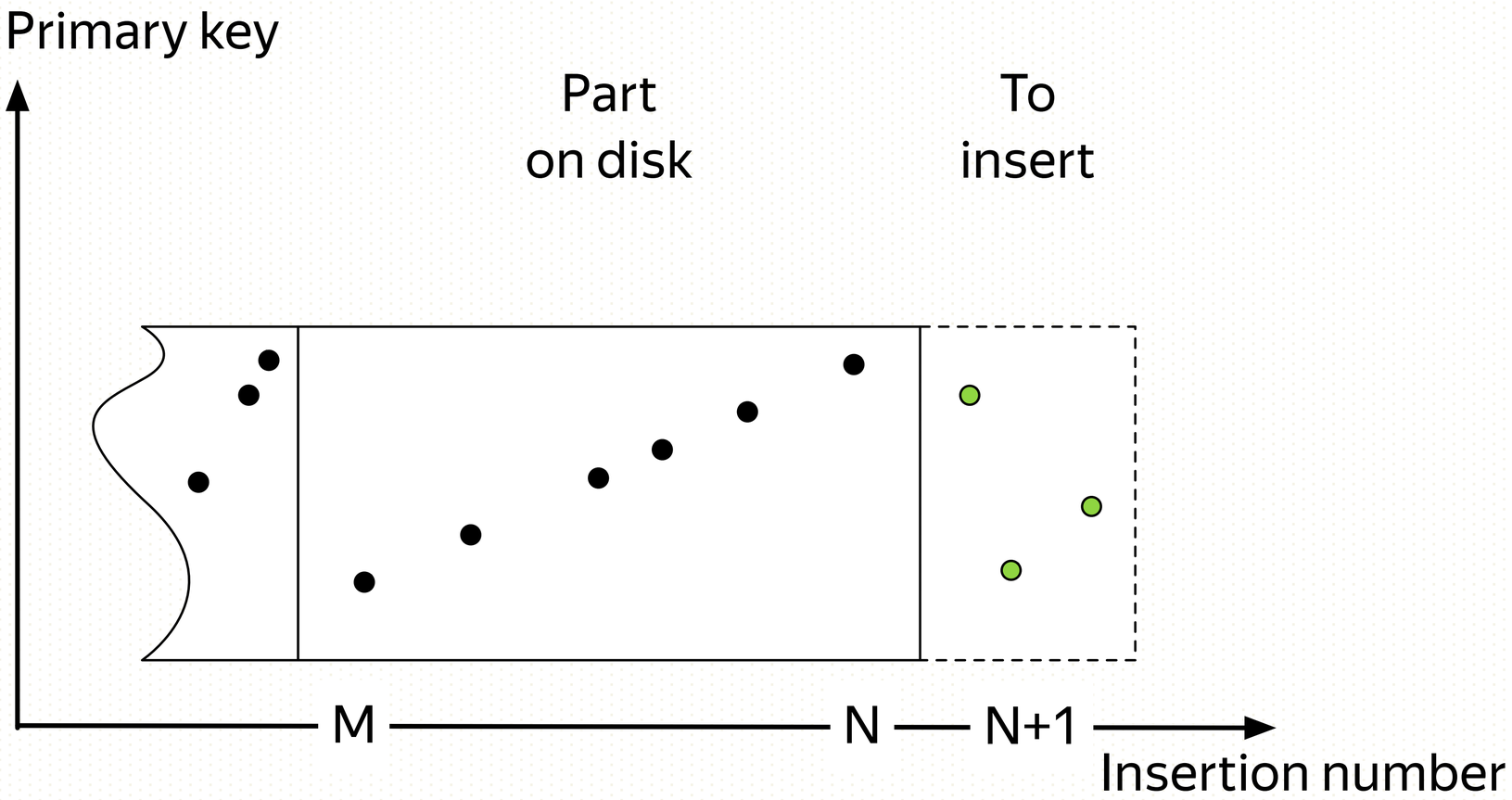
# Things to remember about indexes

- Index is sparse
  - Must fit into memory
  - Default value of granularity (8192) is good enough
  - Does not create a unique constraint
  - Performance of point queries is not stellar
- Table is sorted according to the index
  - There can be only one
  - Using the index is always beneficial

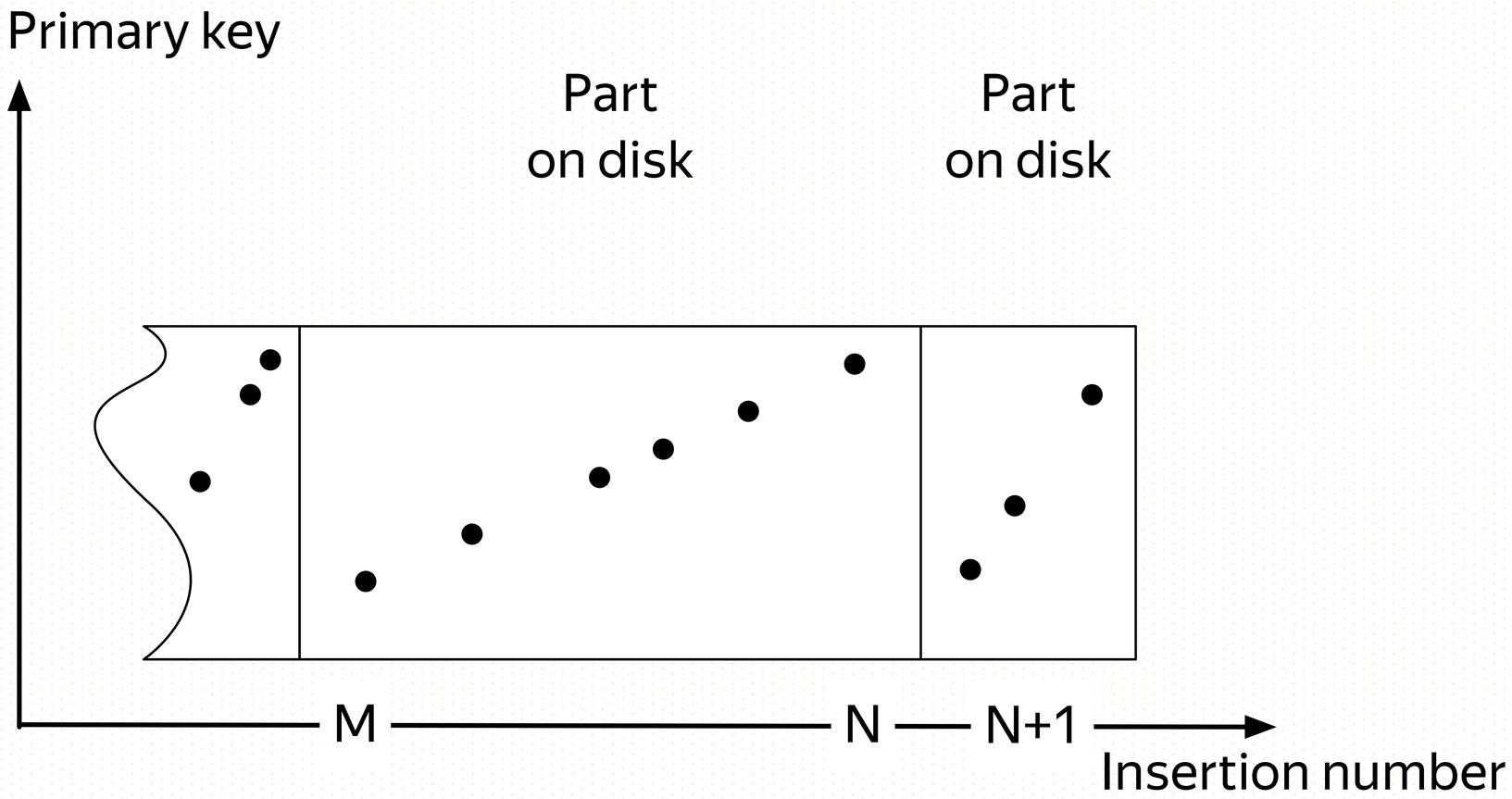
# How to keep the table sorted

- Inserted events are (almost) sorted by time
  - But we need to sort by primary key!
- MergeTree: maintain a small set of sorted parts
  - Similar idea to an LSM tree

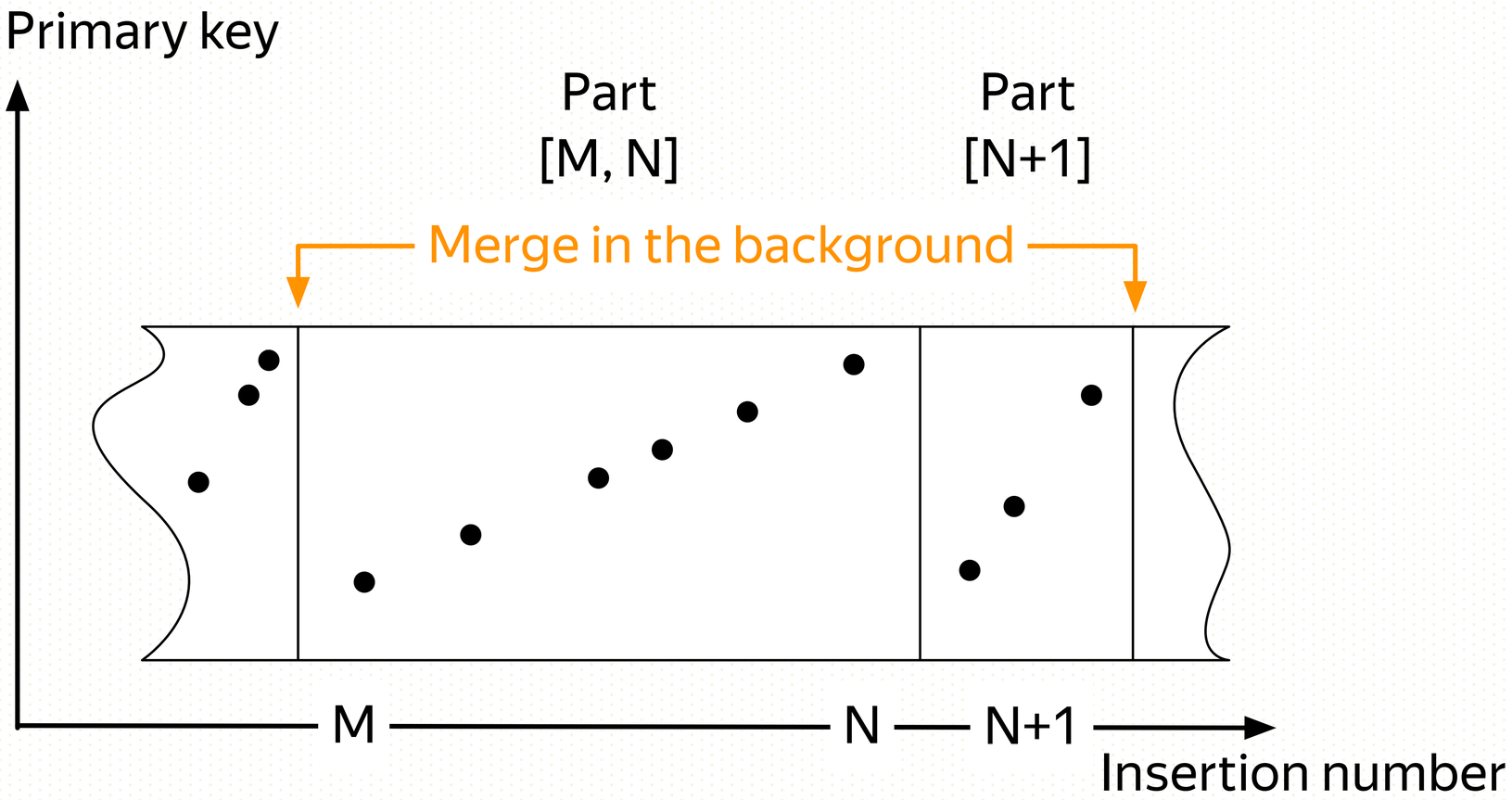
# How to keep the table sorted



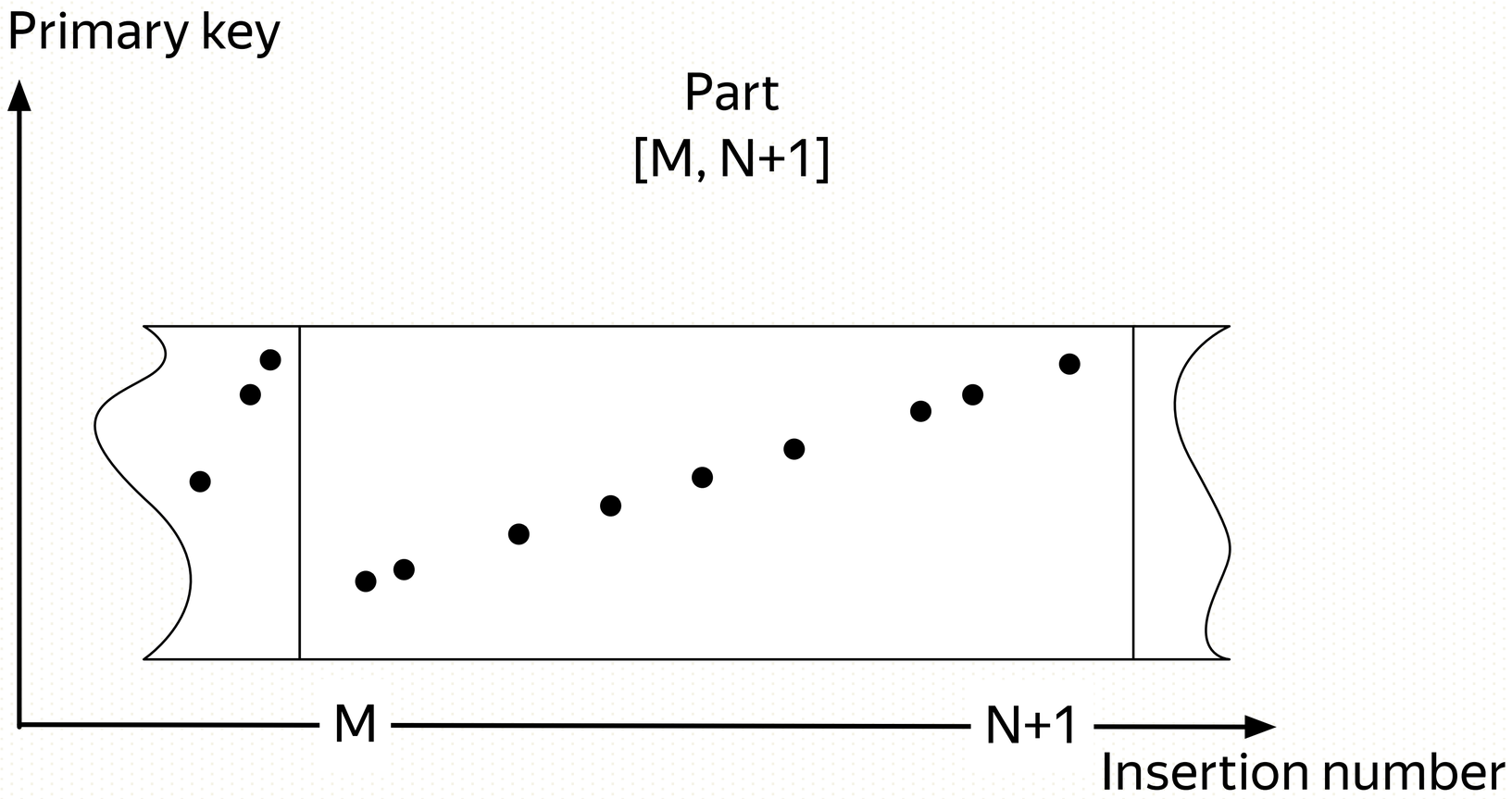
# How to keep the table sorted



# How to keep the table sorted



# How to keep the table sorted



# Things to do while merging

- **Replace/update records**
  - ReplacingMergeTree
  - CollapsingMergeTree
- **Pre-aggregate data**
  - AggregatingMergeTree
- **Metrics rollup**
  - GraphiteMergeTree

# MergeTree partitioning

```
ENGINE = MergeTree(Date,… )
```

- Table is partitioned by month or (soon) by any expression
- Parts from different partitions are not merged
- Easy manipulation of partitions

```
ALTER TABLE DROP PARTITION
ALTER TABLE DETACH/ATTACH PARTITION
```

- MinMax index by partition columns

# Things to remember about MergeTree

- Merging runs in the background
  - Even when there are no queries!
- Control total number of parts
  - Rate of INSERTs
  - `MaxPartsCountForPartition` and `DelayedInserts` metrics are your friends

# When one server is not enough

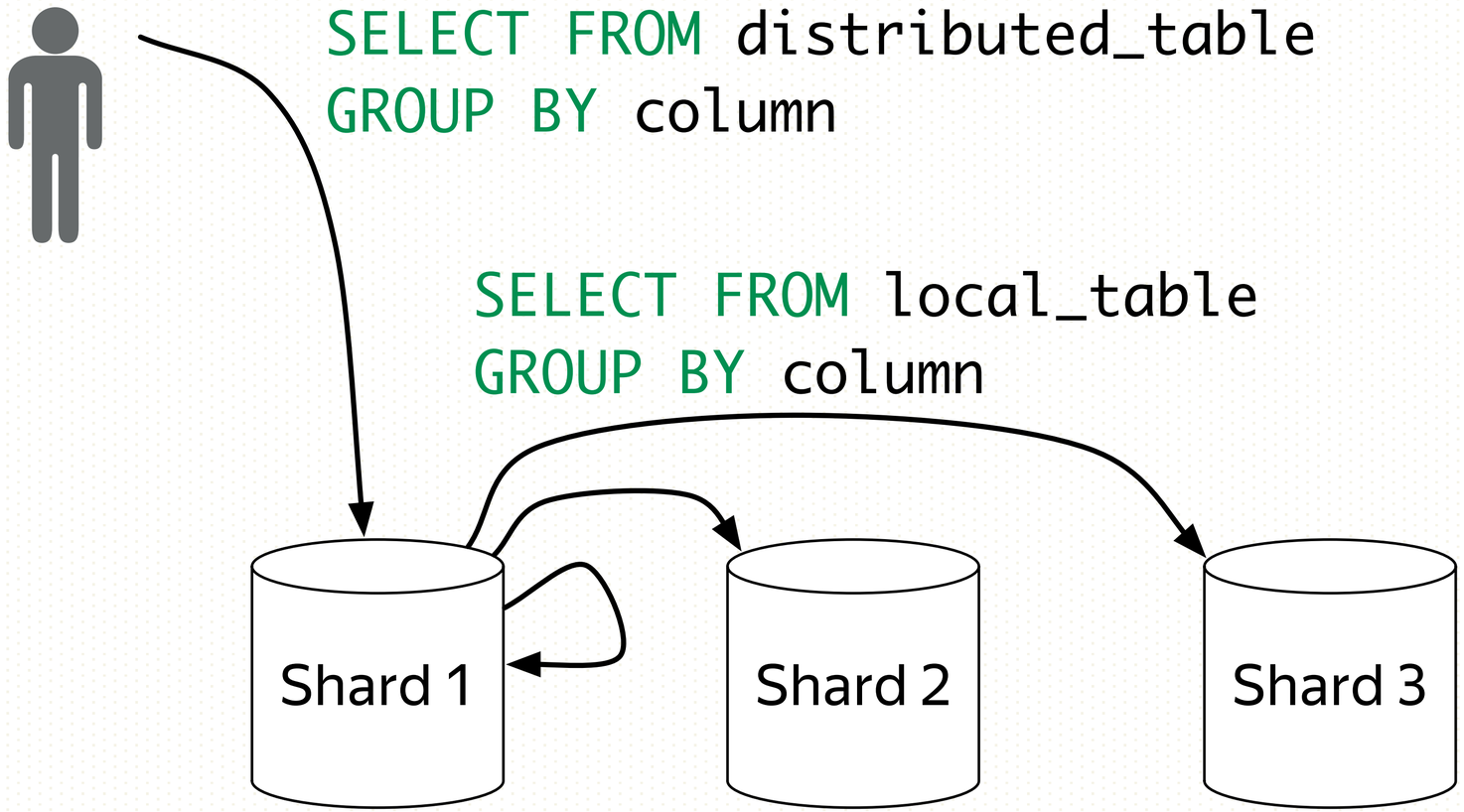
- The data won't fit on a single server...
- You want to increase performance by adding more servers...
- Multiple simultaneous queries are competing for resources...

# When one server is not enough

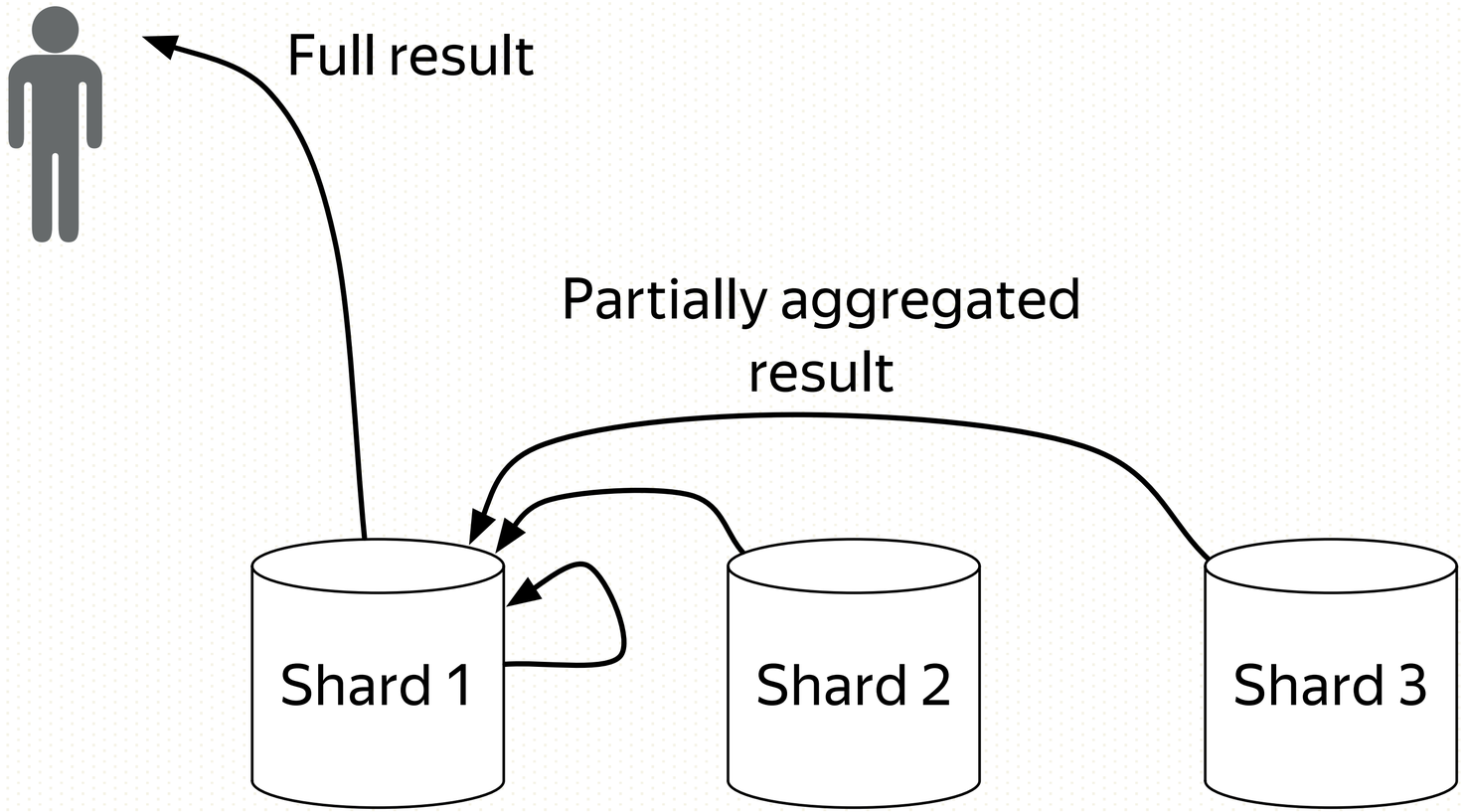
- The data won't fit on a single server...
- You want to increase performance by adding more servers...
- Multiple simultaneous queries are competing for resources...

**ClickHouse: Sharding + Distributed tables!**

# Reading from a Distributed table



# Reading from a Distributed table



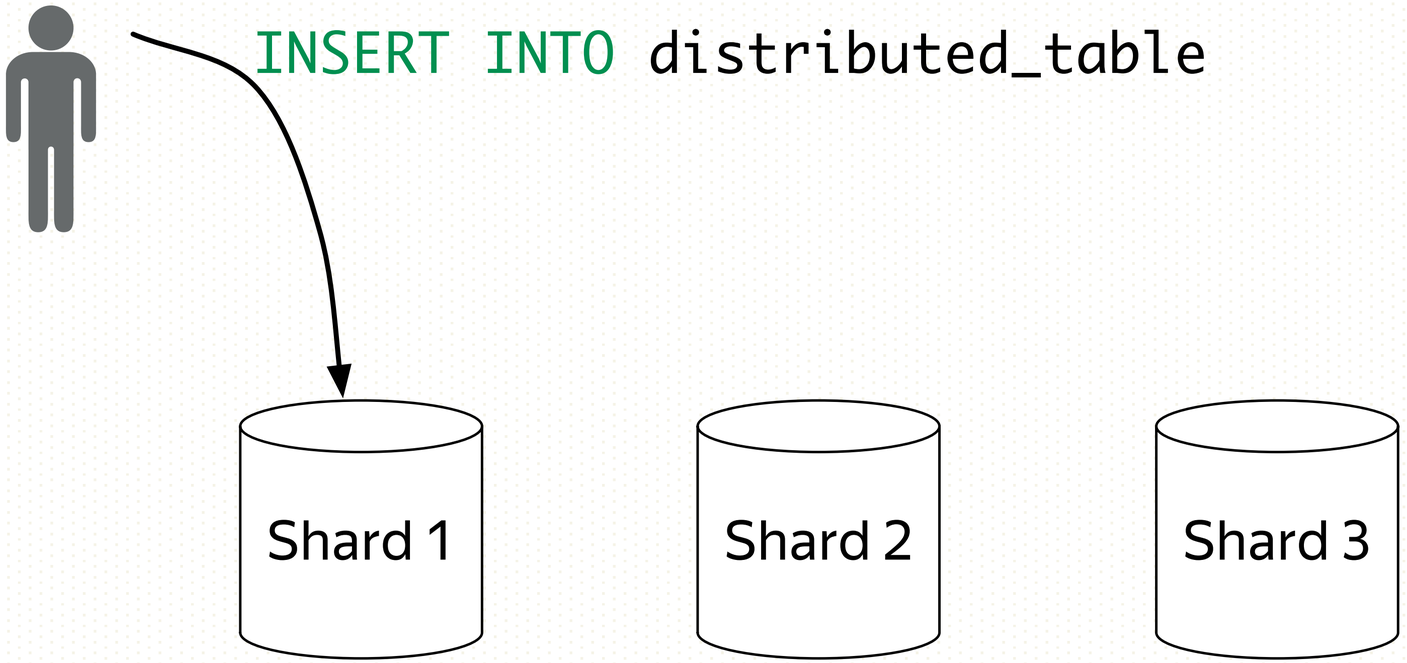
# NYC taxi benchmark

CSV 227 Gb, ~1.3 bln rows

```
SELECT passenger_count, avg(total_amount)
FROM trips GROUP BY passenger_count
```

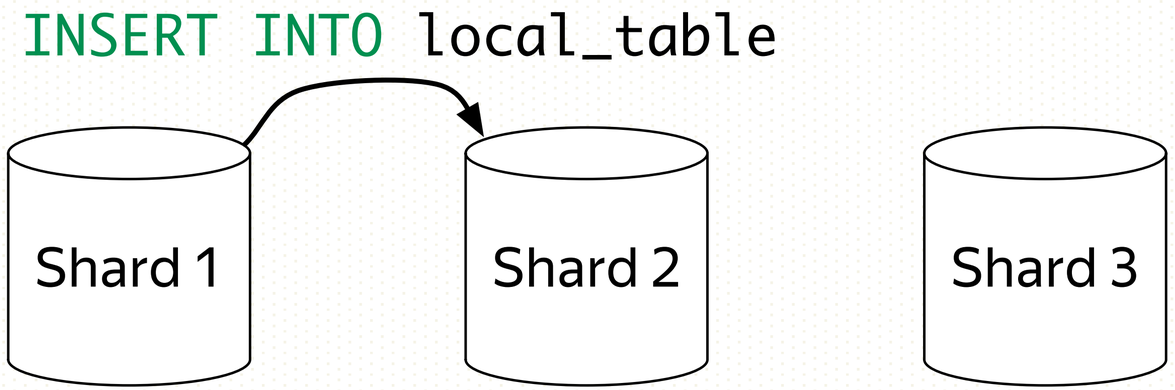
| Shards | 1 | 3 | 140 |
|---|---|---|---|
| Time, s. | 1,224 | 0,438 | 0,043 |
| Speedup | | x2.8 | x28.5 |

# Inserting into a Distributed table

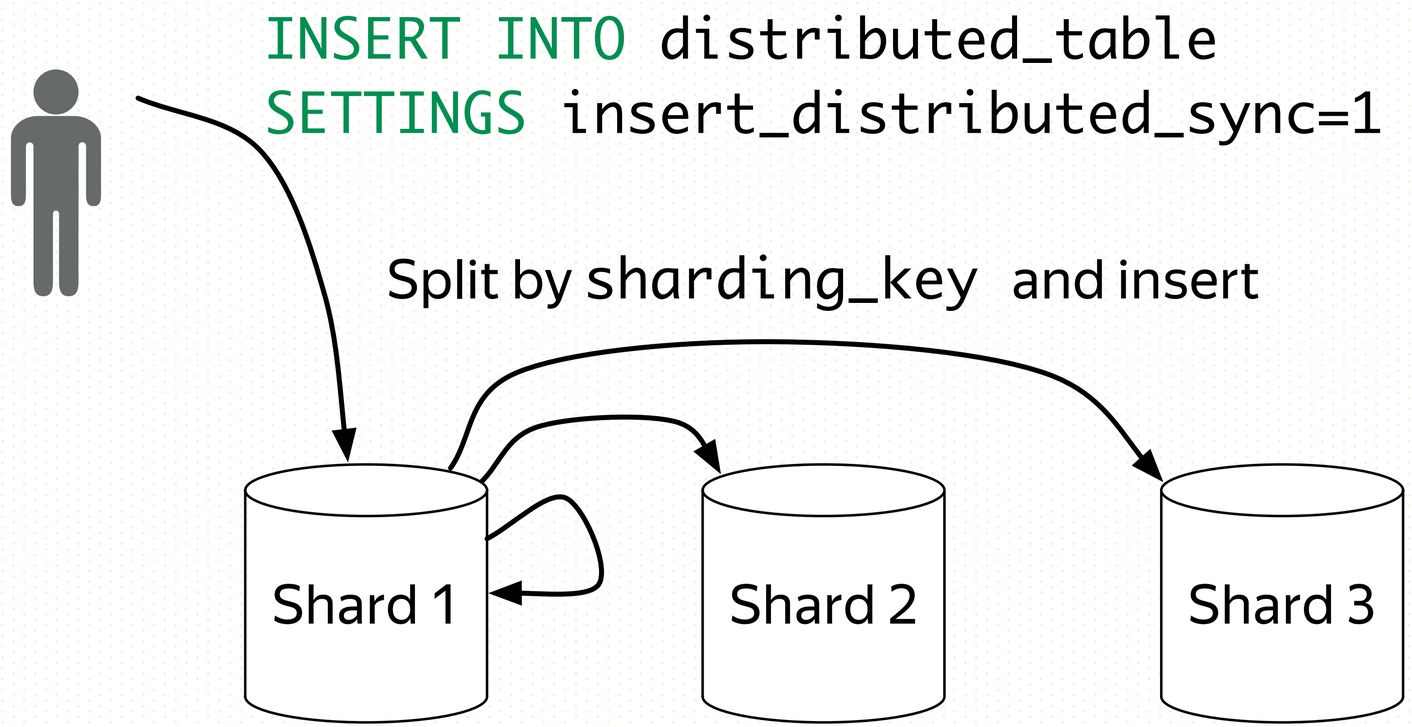


# Inserting into a Distributed table

Async insert into shard # `sharding_key % 3`



# Inserting into a Distributed table



# Things to remember about Distributed tables

**It is just a view**

- Doesn’t store any data by itself

**Will always query all shards**

**Ensure that the data is divided into shards uniformly**

- either by inserting directly into local tables
- or let the Distributed table do it
  (but beware of async inserts by default)

# When failure is not an option

- Protection against hardware failure
- Data must be always available for reading ***and*** writing

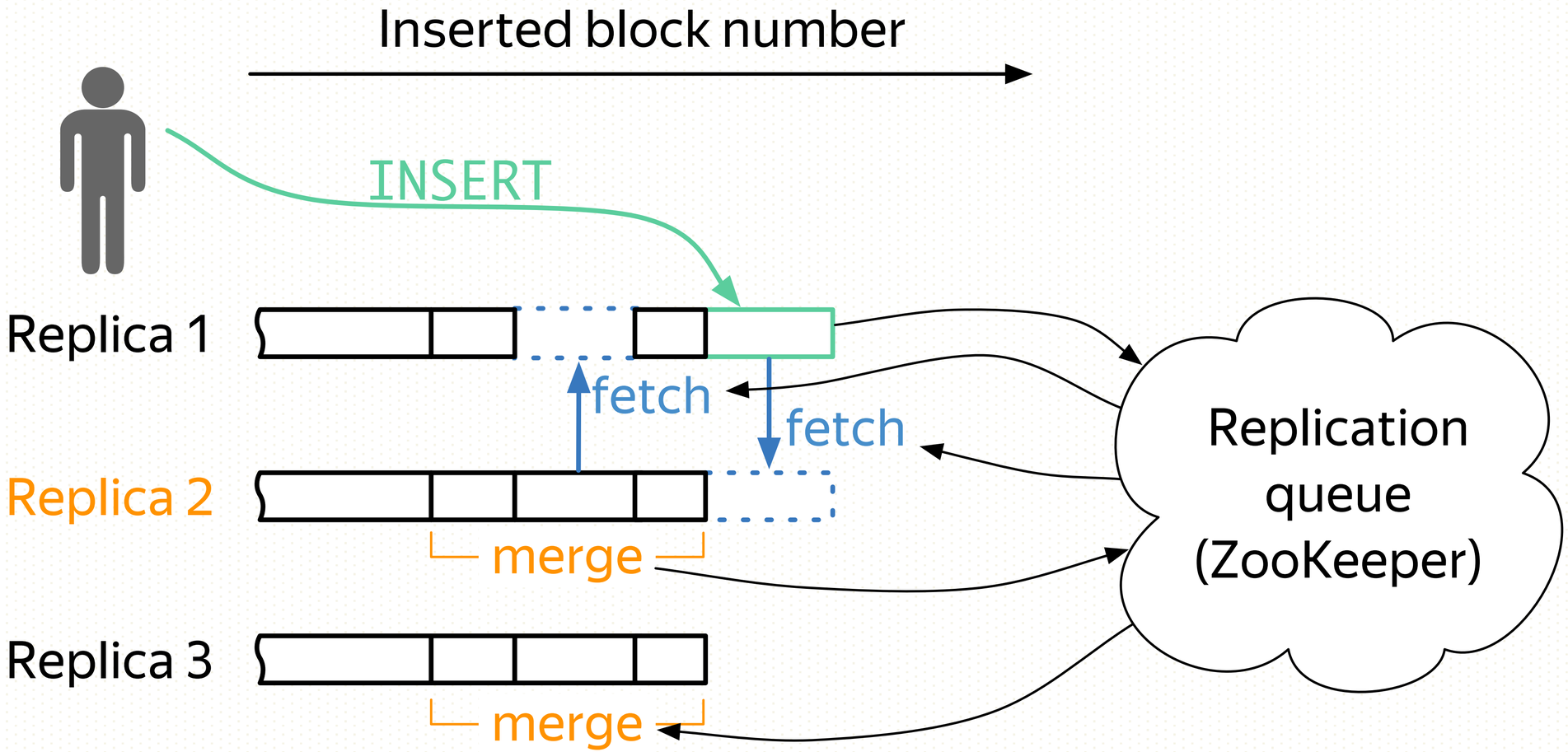
# When failure is not an option

- Protection against hardware failure
- Data must be always available for reading *and* writing

**ClickHouse: ReplicatedMergeTree engine!**

- Async master-master replication
- Works on per-table basis

# Replication internals



# Replication and the CAP–theorem

## What happens in case of network failure (partition)?

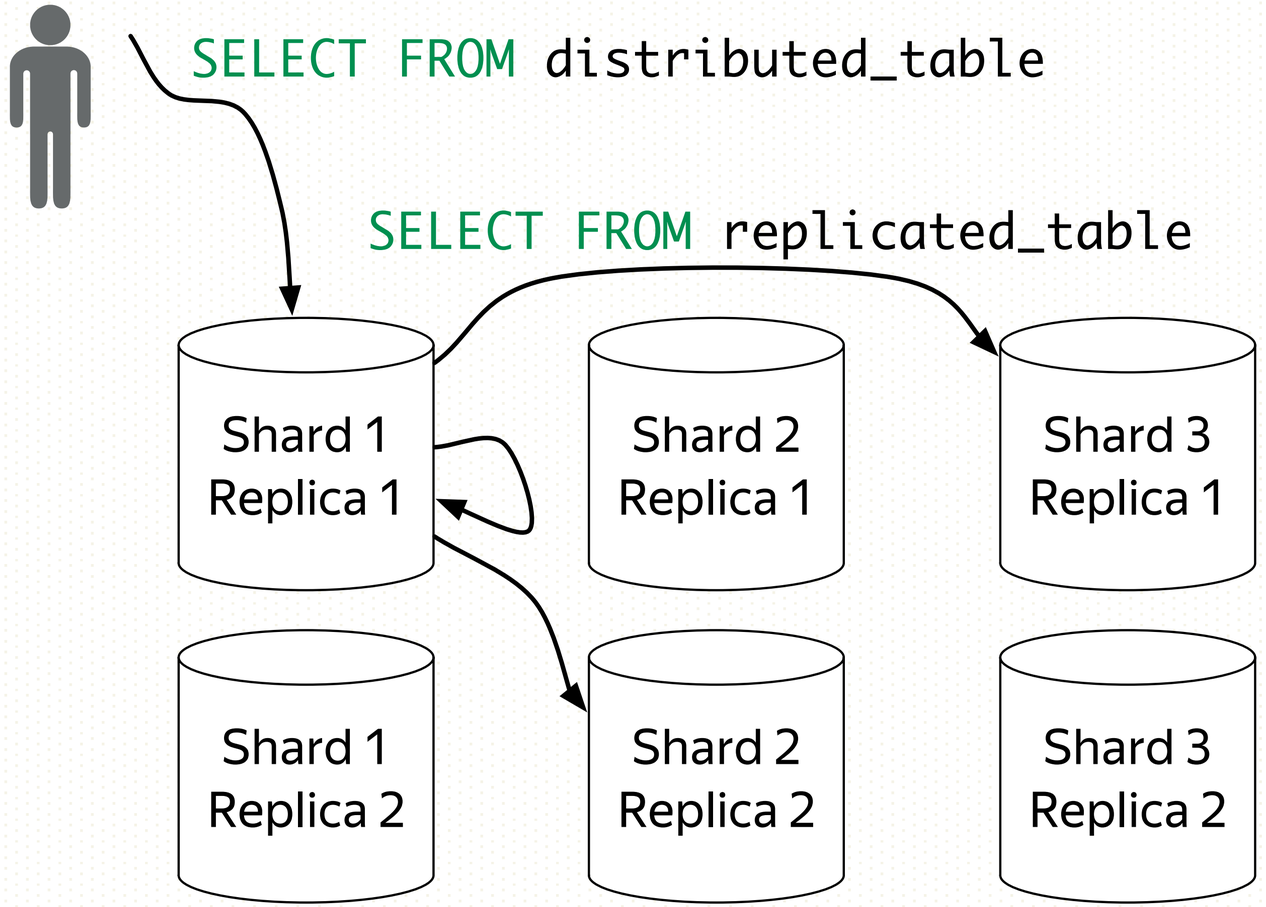
- Not consistent*
  As is any system with async replication

  *But you can turn linearizability on

- Highly available (almost)*
  Tolerates the failure of one datacenter, if ClickHouse replicas are in min 2 DCs and ZK replicas are in 3 DCs.

  *A server partitioned from ZK quorum is unavailable for writes

# Putting it all together



# Things to remember about replication

## Use it!

- Replicas check each other
- Unsure if INSERT went through?
  Simply retry - the blocks will be deduplicated
- ZooKeeper needed, but only for INSERTs
  (No added latency for SELECTs)

## Monitor replica lag

- `system.replicas` and `system.replication_queue` tables are your friends

# Brief recap

- Column-oriented
- Fast interactive queries on real time data
- SQL dialect + extensions
- Bad fit for OLTP, Key-Value, blob storage
- Scales linearly
- Fault tolerant
- Open source!

# Thank you

Start using ClickHouse today!

**Questions? Or reach us at:**

- clickhouse-feedback@yandex-team.com
- **Telegram:** https://t.me/clickhouse_en
- **GitHub:** https://github.com/yandex/ClickHouse/
- **Google group:** https://groups.google.com/group/clickhouse