New Tips and Tricks Every ClickHouse Developer Should Know

Robert Hodges &
Altinity Engineering

Let's make some introductions - Us…

Robert Hodges

Database geek with 30+ years on DBMS systems. Day job: Altinity CEO

Altinity Engineering

Database geeks with centuries of experience in DBMS and applications

ClickHouse support and services including Altinity.Cloud
Authors of Altinity Kubernetes Operator for ClickHouse
and other open source projects

And ClickHouse, a real-time analytic database

Understands SQL

Runs on bare metal to cloud

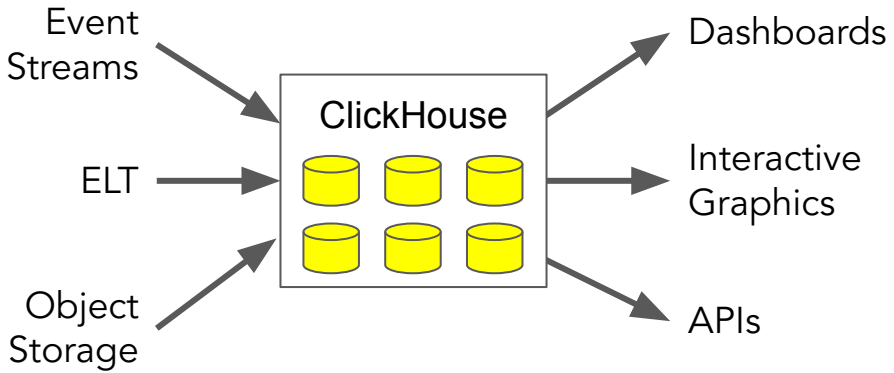
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's the core engine for low-latency analytics

ClickHouse tips and tricks from 2019

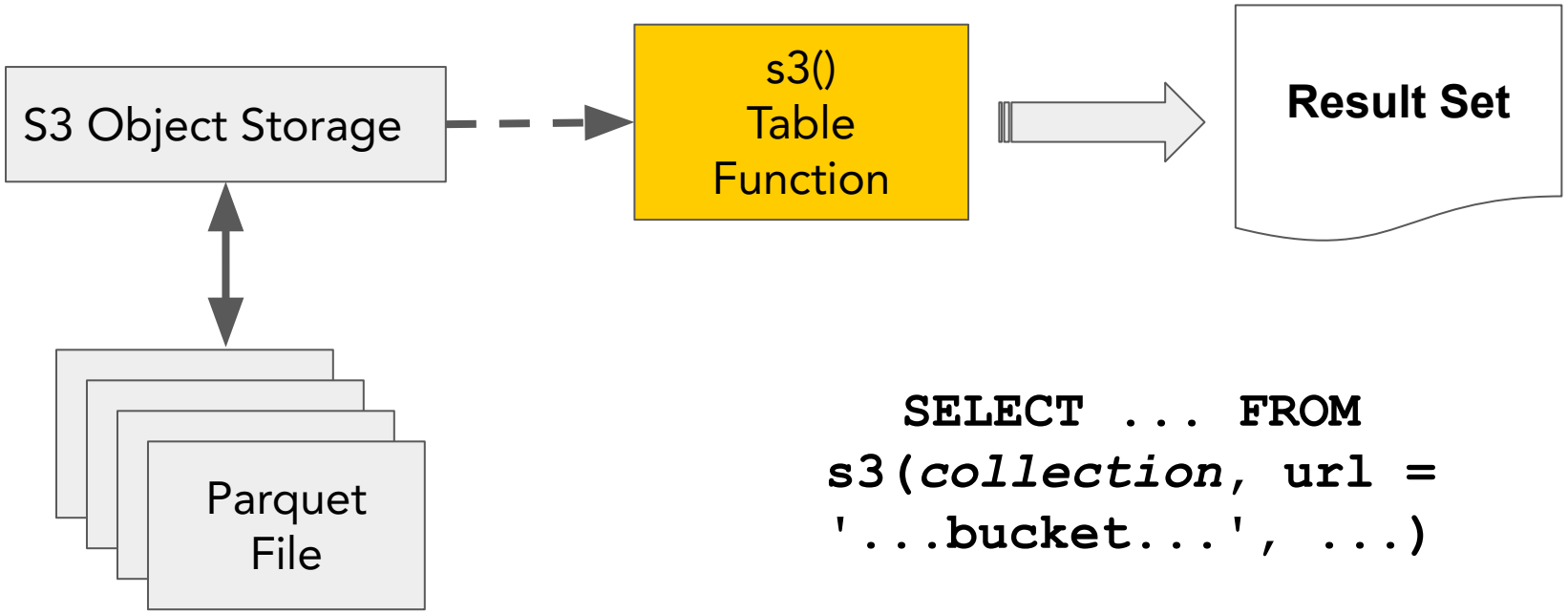
- Use send_logs_level = ‘trace’ to see what ClickHouse is doing
- Use encodings to reduce data size
- Use materialized views to find last point data
- Use arrays to store key-value pairs
- Use materialized columns to precompute values
- Use dictionaries instead of joins for dimension data
- Use MySQL database engine instead of dictionaries
- Use TTLs to delete data
- Use replication instead of backups

From: *Tips and Tricks Every ClickHouse User Should Know*

(YouTube Video, 2019)

Read S3 data fast with wildcards and named collections

Selecting data from S3




```
SELECT ... FROM
s3(collection, url =
'...bucket...', ...)
```

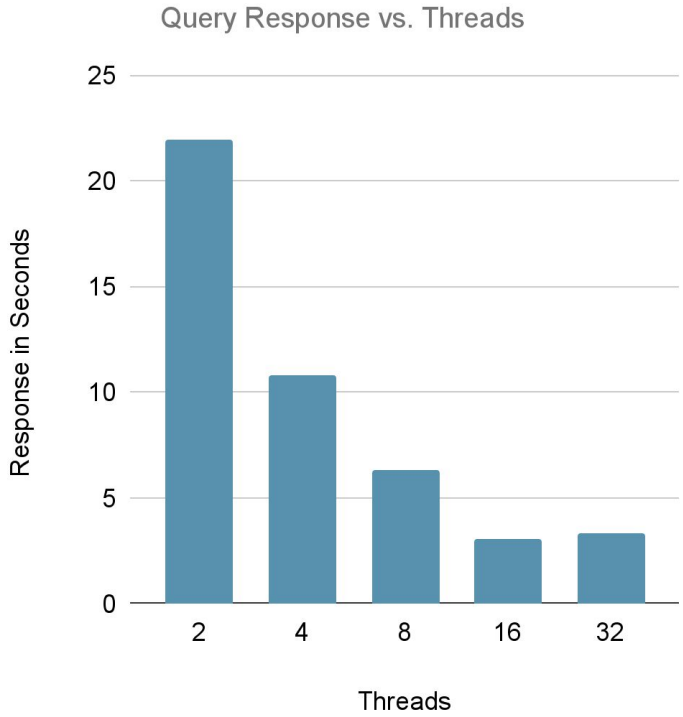
Define a named collection to specific url + credentials

/etc/clickhouse-server/config.d/s3.xml

```
<clickhouse>
  <named_collections>
    <s3_data>
      <url>https://s3.us-east-1.amazonaws.com/bucket/ORDERS/*.parquet</url>
      <access_key_id>AK...UA</access_key_id>
      <secret_access_key>dy...qu</secret_access_key>
      <format>Parquet</format>
    </s3_data>
  </named_collections>
</clickhouse>
```

Create a table on ClickHouse

```
set max_threads = 2;
SELECT min(O_TOTALPRICE),
  avg(O_TOTALPRICE),
  max(O_TOTALPRICE)
FROM s3(s3_data);
set max_threads = 4;
. . .
set max_threads = 8;
. . .
set max_threads = 16;
. . .
set max_threads = 32;
```



More ways to go fast: parallel inserts

On a single host

```
-- Define parallelization
SET max_insert_threads=16
SET max_threads = 16

INSERT INTO default.ORDERS
SELECT * FROM s3(s3_data)
```

Across a cluster

```
-- Propagate max_insert_threads &
max_threads to other shards using
profile.

INSERT INTO default.ORDERS
SELECT * FROM s3Cluster(...)
```

More information:
https://altinity.com/blog/tips-for-high-performance-clickhouse-clusters-with-s3-object-storage

Reduce storage size using codecs and compression

Before: Unoptimized table for sensor readings

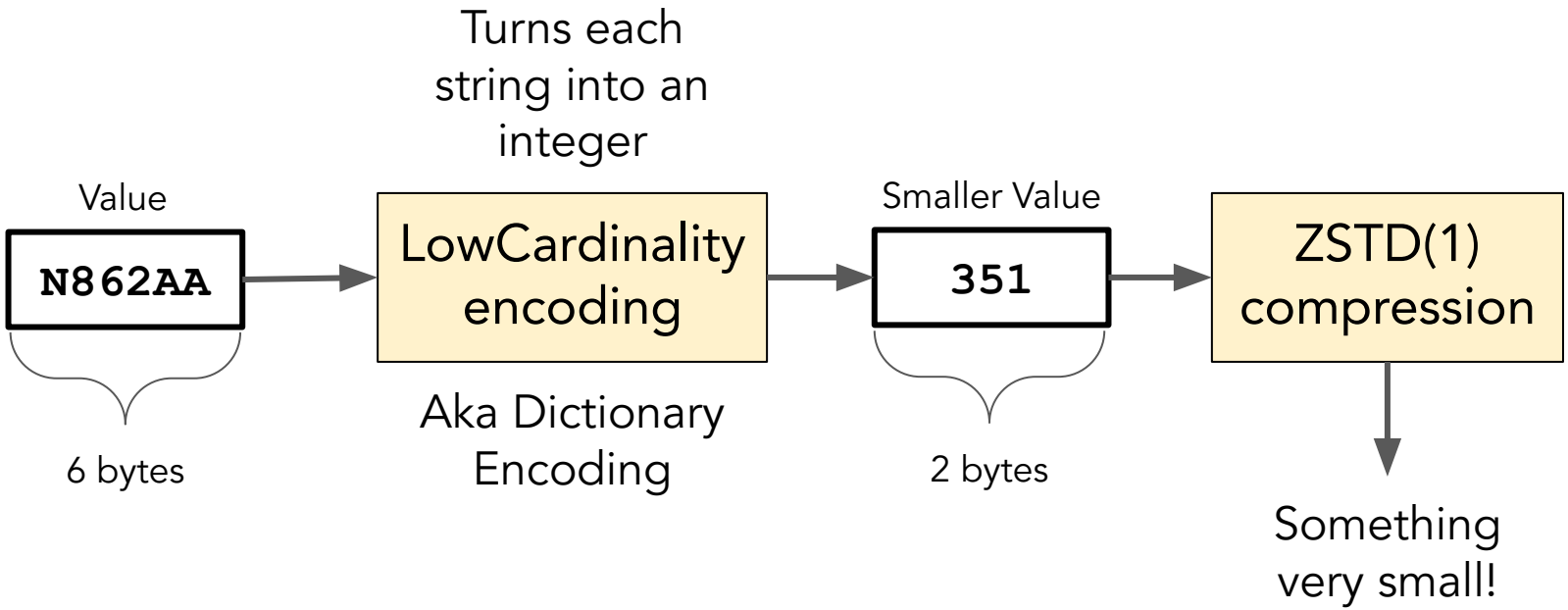
```
CREATE TABLE IF NOT EXISTS readings_unopt (
    sensor_id Int64,
    sensor_type Int32,
    location String,
    time DateTime,
    date Date DEFAULT toDate(time),
    reading Float32
) Engine = MergeTree
PARTITION BY tuple()
ORDER BY tuple();
```

Sub-optimal datatypes!

No codecs!

No partitioning or ordering!

Codecs reduce data before compression



After: Apply codecs and compression together!

```
CREATE TABLE IF NOT EXISTS readings_zstd (
  sensor_id Int32 Codec(DoubleDelta, ZSTD(1)),
  sensor_type UInt16 Codec(ZSTD(1)),
  location LowCardinality(String) Codec(ZSTD(1)),
  time DateTime Codec(DoubleDelta, ZSTD(1)),
  date ALIAS toDate(time),
  temperature Decimal(5,2) Codec(T64, ZSTD(10))
)
Engine = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (location, sensor_id, time);
```

Optimized data types

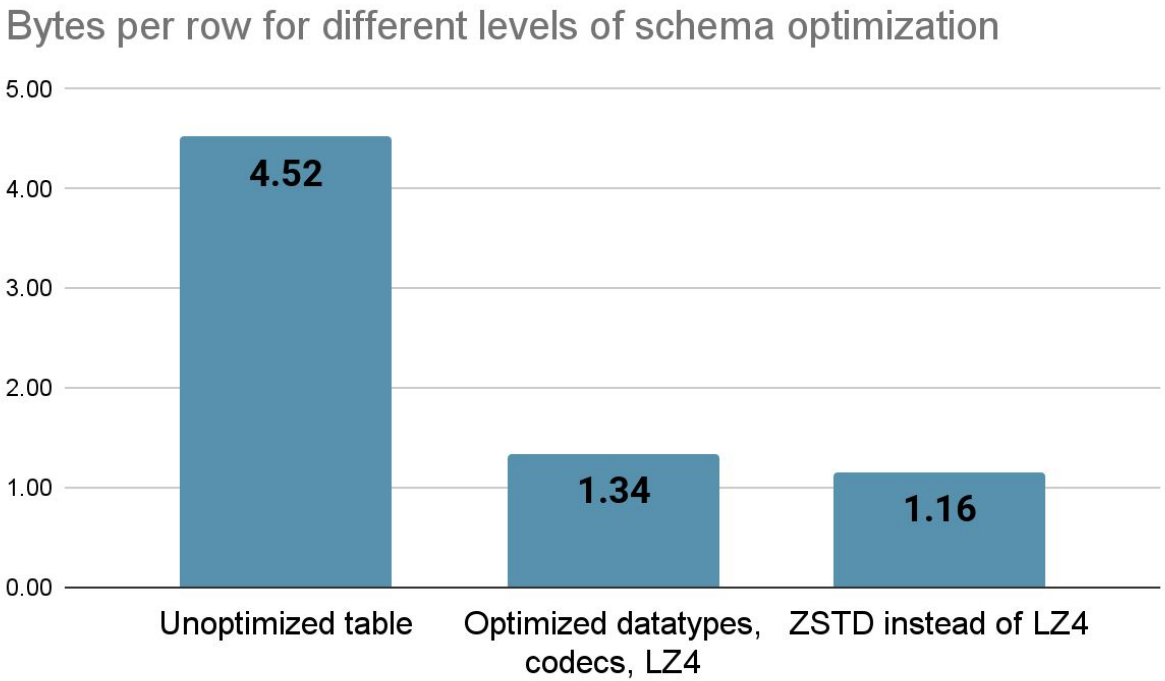
Codecs + ZSTD compression

ALIAS column

Time-based partitioning

Sorting by key columns + time

On-disk table size for different schemas



Quick Comparison of Codecs

Name	Best for
LowCardinality	Strings with fewer than 10K values
Delta	Time series
Double Delta	Increasing counters
Gorilla	Gauge data (bounces around mean)
T64	Integers other than random hashes
FPC	Floating point sequences

Repeat time column in ORDER BY to speed up time-based queries

Time is a common component of table order

```
CREATE TABLE web_events_1x (
    `time` DateTime,
    `user_id` UInt32,
    `session_id` UInt64,
    `event_type` UInt16,
    `str_value` String,
    `float_value` Float32
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (user_id, session_id, time)
```

Partition by month to get <= 1000 parts total

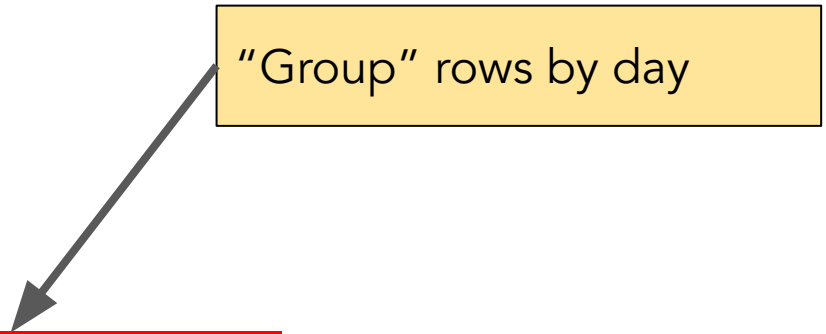
Lowest cardinality → Highest cardinality

Following the “rules” can lead to slow queries!

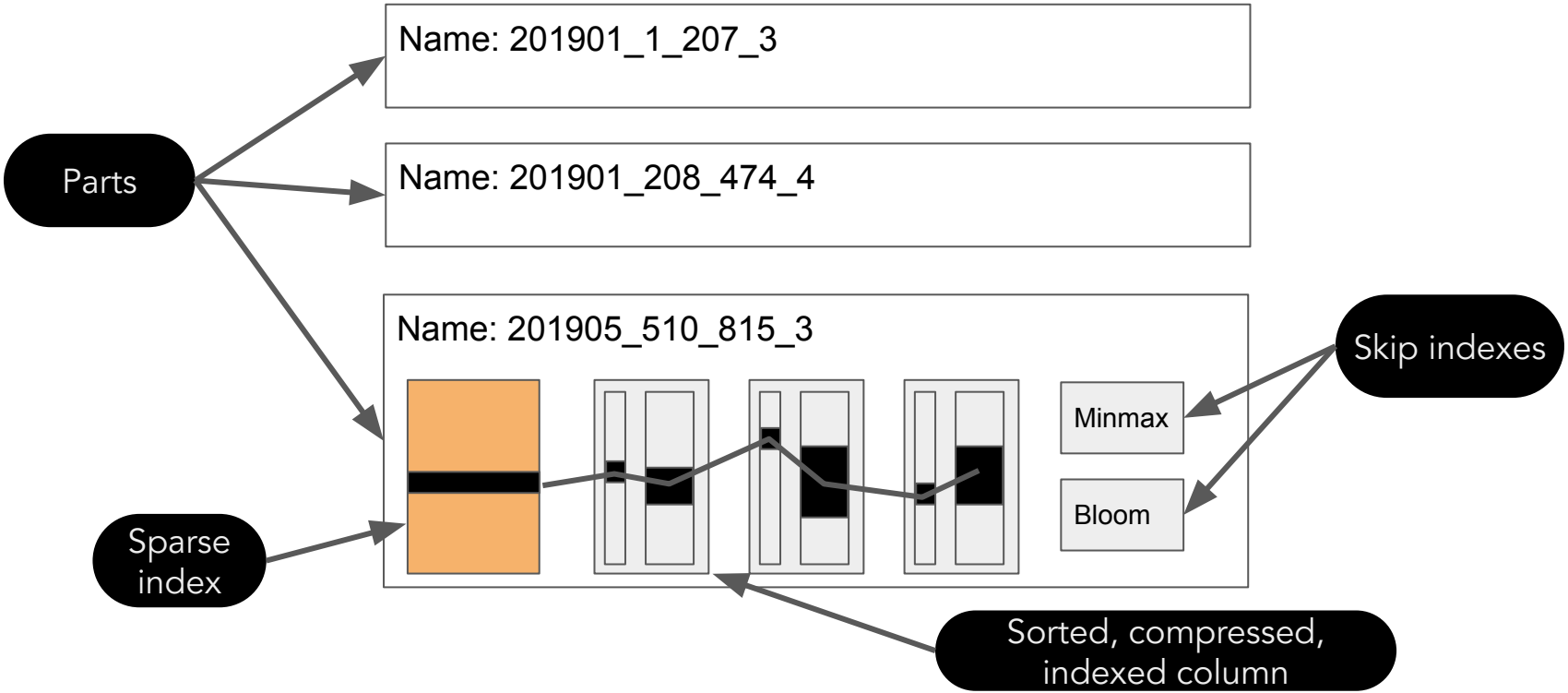
```
WITH toStartOfDay(toDateTime('2019-02-05 01:00:00')) AS day
SELECT
    avg(float_value), avg(length(str_value))
FROM web_events_1x
WHERE day = toStartOfDay(time)

[host] . . . Selected 1/2 parts by partition key, 1 parts by
primary key, 28544/28544 marks by primary key, 28544 marks
to read from 1 ranges

1 row in set. Elapsed: 1.322 sec. Processed 233.83 million
rows, 2.70 GB (176.88 million rows/s., 2.04 GB/s.)
```

Let's group data by day in the sort order

```
CREATE TABLE web_events_2x (
    `time` DateTime,
    `user_id` UInt32,
    `session_id` UInt64,
    `event_type` UInt16,
    `str_value` String,
    `float_value` Float32
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
PRIMARY KEY (user_id, toStartOfDay(time), session_id)
ORDER BY (user_id, toStartOfDay(time), session_id, time)
```



What goes on under the covers in MergeTree tables



Now we can query efficiently by day in monthly partitions

```
WITH toStartOfDay(toDateTime('2019-02-05 01:00:00')) AS day
SELECT
    avg(float_value), avg(length(str_value))
FROM web_events_1x
WHERE day = toStartOfDay(time)

[host] . . . Selected 1/1 parts by partition key, 1 parts by
primary key, 4690/61036 marks by primary key, 4690 marks to
read from 999 ranges

1 row in set. Elapsed: 0.646 sec. Processed 38.42 million
rows, 986.50 MB (59.43 million rows/s., 1.53 GB/s.)
```

Handle mutable data with ReplacingMergeTree

ReplacingMergeTree deduplicates rows in ORDER BY

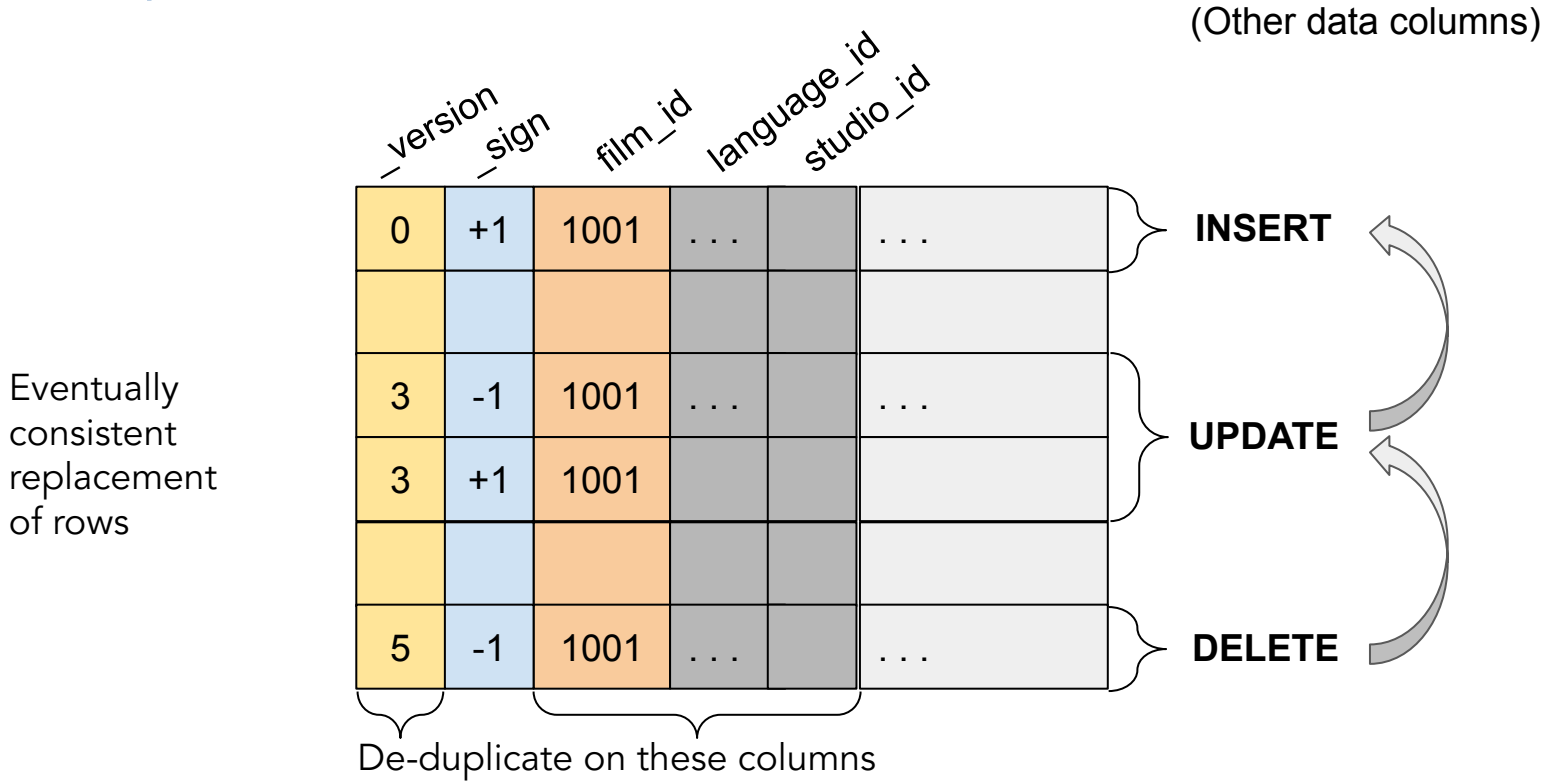
```
 CREATE TABLE sakila.film (
    `film_id` UInt16,
    `title` String,
     . . .
    `_version` UInt64 DEFAULT 0,
    `_sign` Int8 DEFAULT 1
)
ENGINE = ReplacingMergeTree(_version)
ORDER BY language_id, studio_id, film_id
```

Pro tip: Use PRIMARY KEY to prefix a long ORDER BY

Other cols go to left

Row key goes on right (if you have one)

How ReplacingMergeTree works

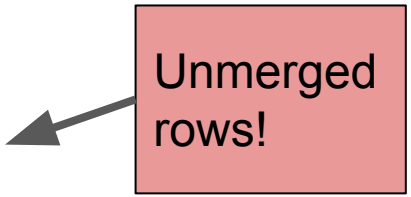


Adding a row to RMT table

```
INSERT INTO sakila.film VALUES
(1001,'Blade Runner','Best. Sci-fi. Film. Ever.',
'1982',1,NULL,6,'0.99',117,'20.99','PG',
'Deleted Scenes,Behind the Scenes',now()
,0,1)

SELECT title, release_year
FROM film WHERE film_id = 1001

┌─title────────┬─release_year─┐
│ Blade Runner │ 1982         │
└──────────────┴──────────────┘
```

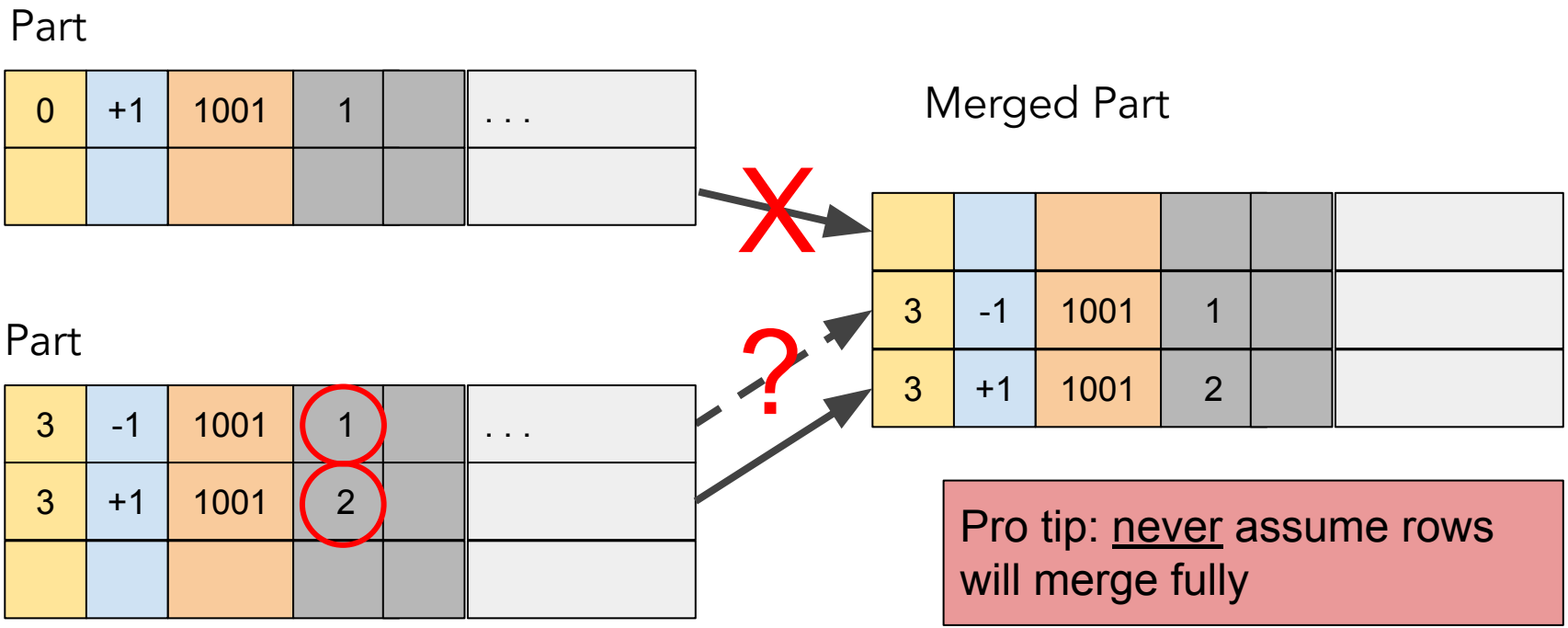
Updating a row in the RMT table

```
INSERT INTO sakila.film VALUES
(1001,'Blade Runner','Best. Sci-fi. Film. Ever.',...,3,-1),
(1001,'Blade Runner - Director''s Cut','Best. Sci-fi. Film.
Ever.',...,3,1)

SELECT title, release_year
FROM film WHERE film_id = 1001
```




```
┌─title──────────────────────────────┬─release_year─┐
│ Blade Runner - Director's Cut      │ 1982         │
└────────────────────────────────────┴──────────────┘

┌─title──────────┬─release_year─┐
│ Blade Runner   │ 1982         │
└────────────────┴──────────────┘
```

Rows are replaced when merges occur



FINAL keyword merges data dynamically

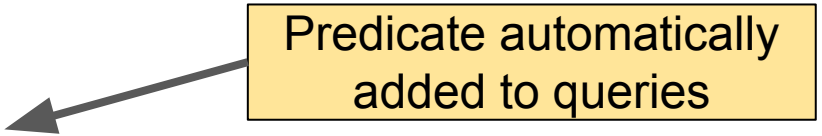
```
SELECT film_id, title
  FROM sakila.film FINAL
    WHERE film_id = 1001
```

Adds initial scan to merge rows

```
┌─title──────────────────────────────┬─release_year─┐
│ Blade Runner - Director's Cut      │ 1982         │
└────────────────────────────────────┴──────────────┘
```

Row policies prevent deleted rows from showing up

```
CREATE ROW POLICY
  sakila_film_rp ON sakila.film
    FOR SELECT USING sign != -1 TO ALL

SELECT title, release_year, _version, _sign
FROM sakila.film FINAL
WHERE film_id = 1001

Ok.

0 rows in set. Elapsed: 0.005 sec.
```

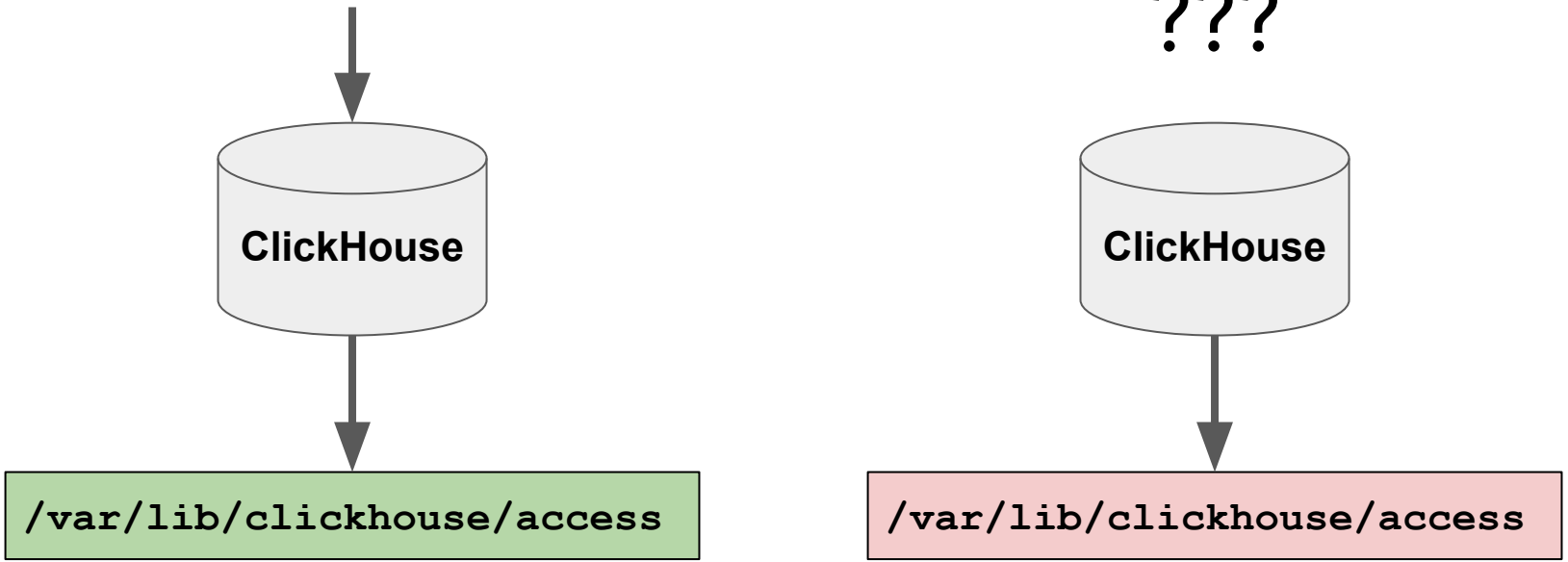


Store RBAC model in ZooKeeper

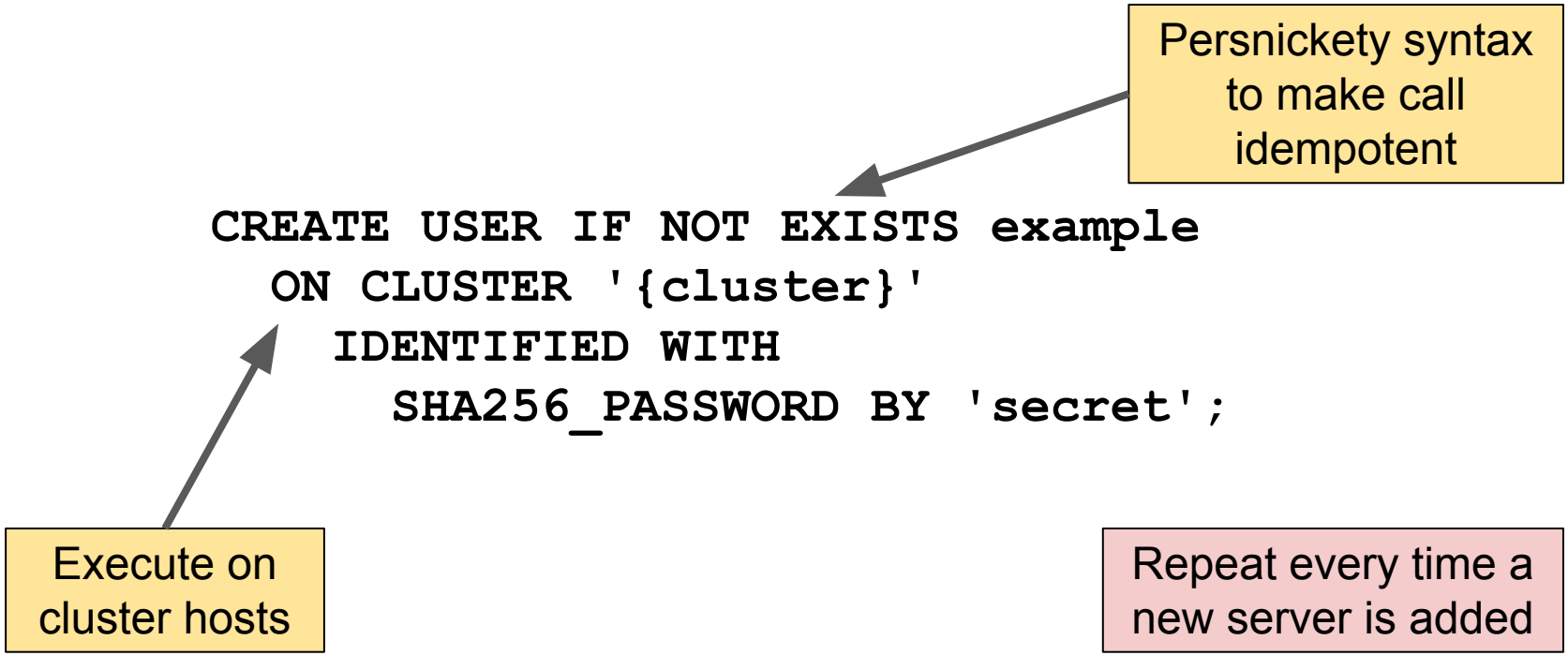
(Or ClickHouse Keeper!)

SQL RBAC is clumsy to manage across multiple servers

```
CREATE USER example IDENTIFIED WITH
    SHA256_PASSWORD BY 'secret';
```



Old school approach to propagate RBAC commands



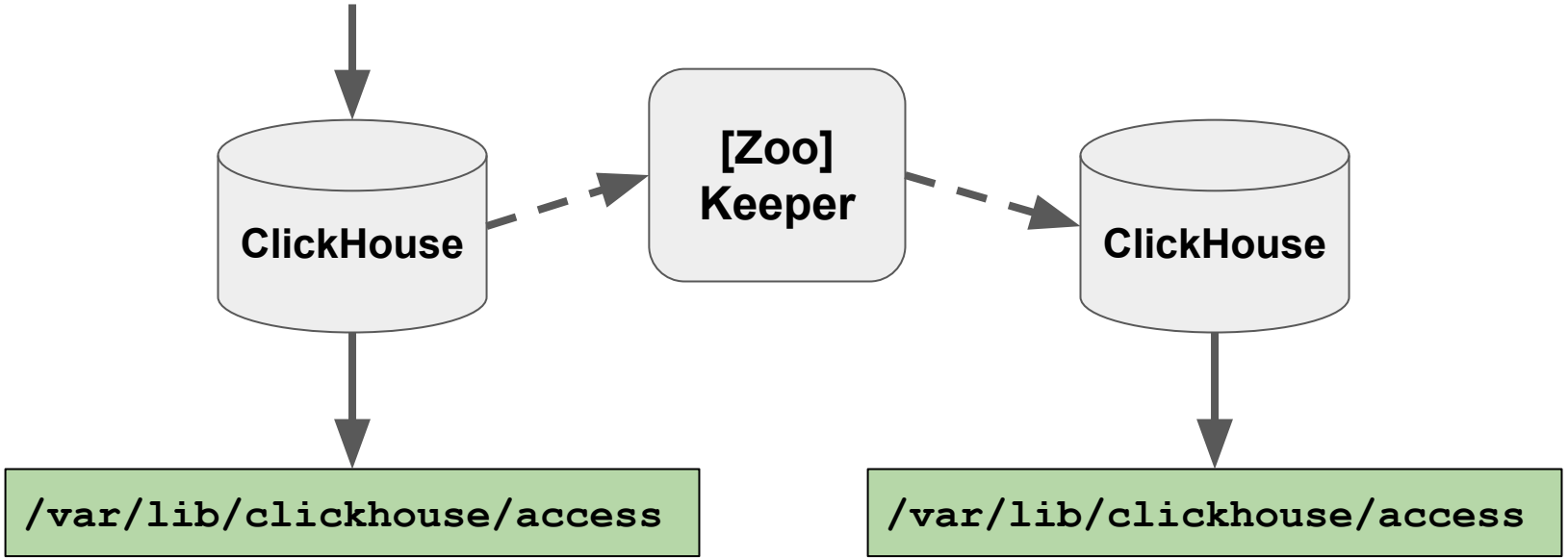
Configuration to enable RBAC replication

/etc/clickhouse-server/users.d/zk_rbac.xml

```
<clickhouse>
  <user_directories replace="replace">
    <users_xml>
      <path>/etc/clickhouse-server/users.xml</path>
    </users_xml>
    <replicated>
      <zookeeper_path>/clickhouse/access/</zookeeper_path>
    </replicated>
  </user_directories>
</clickhouse>
```

Now all changes will propagate automatically!

```
CREATE USER example IDENTIFIED WITH
    SHA256_PASSWORD BY 'secret';
```



How to see current RBAC settings

```
SELECT name, value
FROM system.zookeeper
WHERE path = '/clickhouse/access/uuid/'
FORMAT Vertical

Row 1:
──────
name:         ee48286d-e012-674d-4629-c395b84db6b0
value:        ATTACH USER example IDENTIFIED WITH sha256_hash
BY 'F9…06' SALT '6E…BF';
```

Reduce data size and cost with TTL clauses

TTLs started as a way to “time out” rows

```
CREATE TABLE default.web_events_with_ttl_2 (
    `time` DateTime,
    . . .
    `float_value` Float32
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (user_id, toStartOfDay(time), session_id, time)
TTL time + INTERVAL 12 MONTH DELETE
```

Now TTLs can move, aggregate, and recompress data

```
CREATE TABLE default.web_events_with_ttl_2 (
    `time` DateTime,
    . . .
    `float_value` Float32
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (user_id, toStartOfDay(time), session_id, time)
TTL time + INTERVAL 1 MONTH RECOMPRESS CODEC (ZSTD(1)),
    time + INTERVAL 6 MONTH RECOMPRESS CODEC (ZSTD(10)),
    time + INTERVAL 12 MONTH DELETE
```

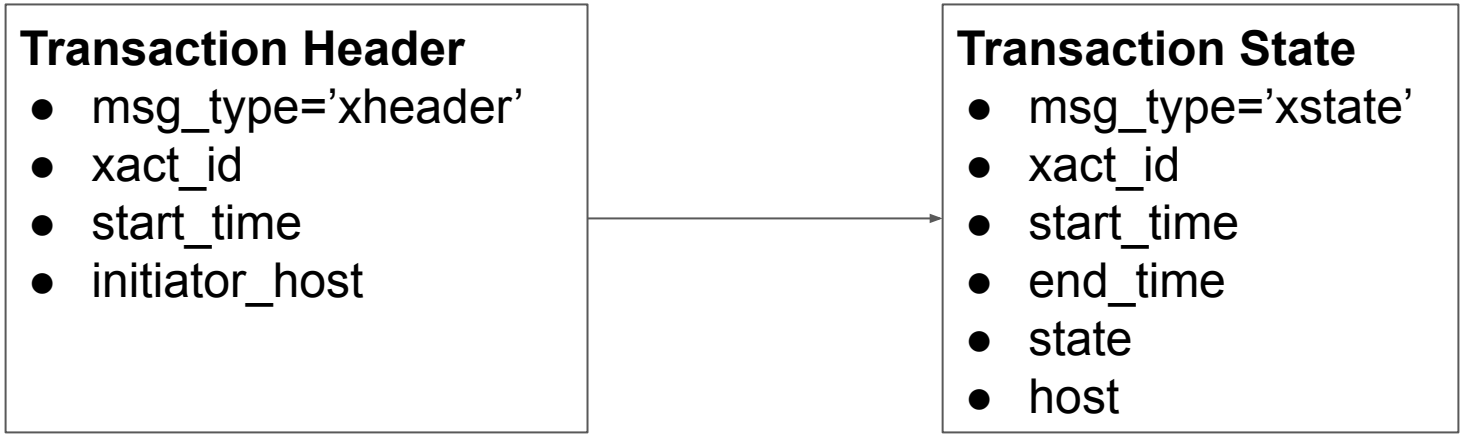
Let's prove it works!

```
SELECT partition, name, rows,
    data_compressed_bytes AS compressed,
    data_uncompressed_bytes AS uncompressed
FROM system.parts
WHERE (table = 'web_events_with_ttl_2') AND active
ORDER BY name DESC
```

partition	name	rows	compressed	uncompressed
202304	202304_1_1_0	50000	613930	1388890
202302	202302_2_2_1	50000	327461	1388890
202208	202208_3_3_1	50000	264054	1388890

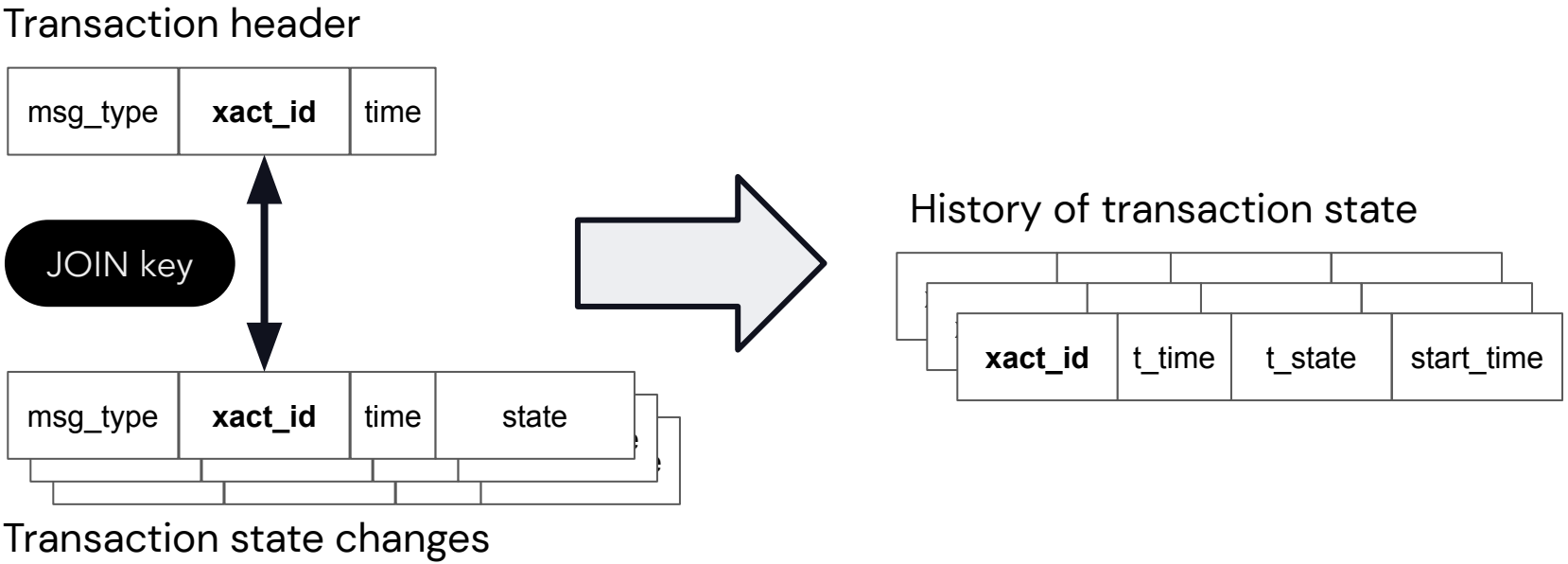
Use aggregation to simulate joins

Basic big data design: One table or many?



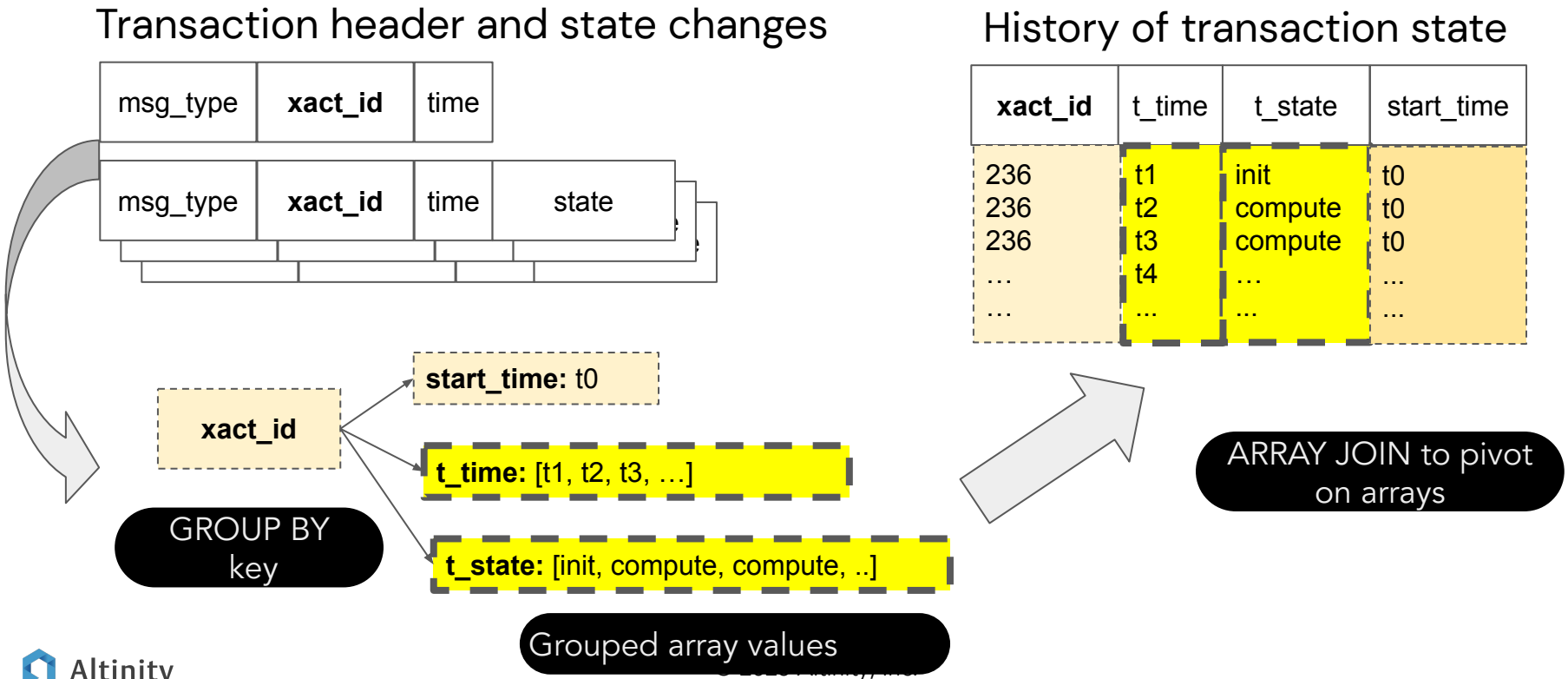
ClickHouse bias for big data: Put entities in a single table

How do we normally join master detail records?



Large table joins are an anti-pattern in low-latency apps!

Aggregation can implement master/detail joins!



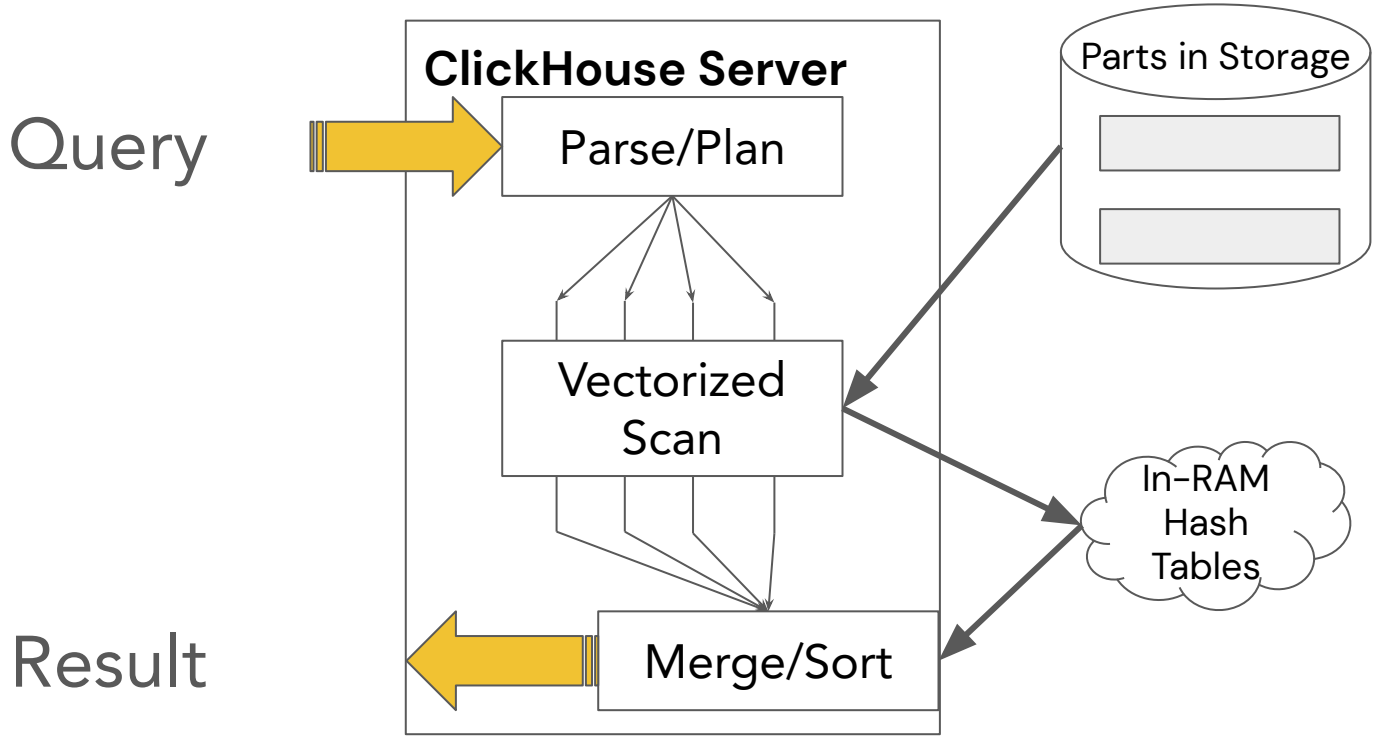
And here's the code…

```
SELECT xact_id, t_time, t_state, start_time
FROM
(
  SELECT xact_id, groupArrayIf(time, msg_type = 'xstate') AS t_time,
    groupArrayIf(state, msg_type = 'xstate') AS t_state,
    anyIf(time, msg_type = 'xheader') AS start_time
  FROM transaction
  GROUP BY xact_id
)
ARRAY JOIN t_time, t_state
```

And the output…

```
xact_id|t_time                    |t_state|start_time              |
-------+------------------------+-------+------------------------+
17     |2023-04-10 23:13:10.000|init   |2023-04-10 23:13:06.000|
17     |2023-04-10 23:13:11.000|compute|2023-04-10 23:13:06.000|
17     |2023-04-10 23:13:24.000|compute|2023-04-10 23:13:06.000|
14     |2023-04-10 23:13:06.000|init   |2023-04-10 23:13:05.000|
14     |2023-04-10 23:13:08.000|compute|2023-04-10 23:13:05.000|
14     |2023-04-10 23:13:22.000|compute|2023-04-10 23:13:05.000|
```

Q: Why does this work?? A: ClickHouse query model



Conclusion

Handy list of tips and tricks!

- Read S3 fast with max_threads and wildcards
- Reduce storage size using codecs and compression
- Use multiple time values in ORDER BY to query subsets of parts efficiently
- Handle rapidly changing data with ReplacingMergeTree
- Store RBAC model in ZooKeeper (or ClickHouse Keeper)
- Recompress data to reduce size over time with TTLs
- Use aggregation instead of joins on large datasets

More information!

- Altinity YouTube channel
  - Tips and Tricks Every ClickHouse User Should Know
  - Adventures with the ClickHouse ReplacingMergeTree Engine
  - A Day in the Life of a ClickHouse Query
- Altinity Blog – https://altinity.com/blog
- ClickHouse Documentation – https://clickhouse.com/docs/en/intro

Thank you! Questions?

Altinity.Cloud

Altinity Support

Altinity Stable Builds

https://altinity.com