

# Five Things Every New **ClickHouse®** User Should Know

## Part 1: Applications

Robert Hodges - Altinity CEO  
22 July 2025



# ALTINITY®

Run Open Source ClickHouse® Better

**Altinity.Cloud      Enterprise Support**

Altinity® is a Registered Trademark of Altinity, Inc.  
ClickHouse® is a registered trademark of ClickHouse, Inc.;  
Altinity is not affiliated with or associated with ClickHouse, Inc.



**What's a  
ClickHouse  
???**



# ClickHouse® is a real-time analytic database

Understands SQL

Runs on bare metal to cloud

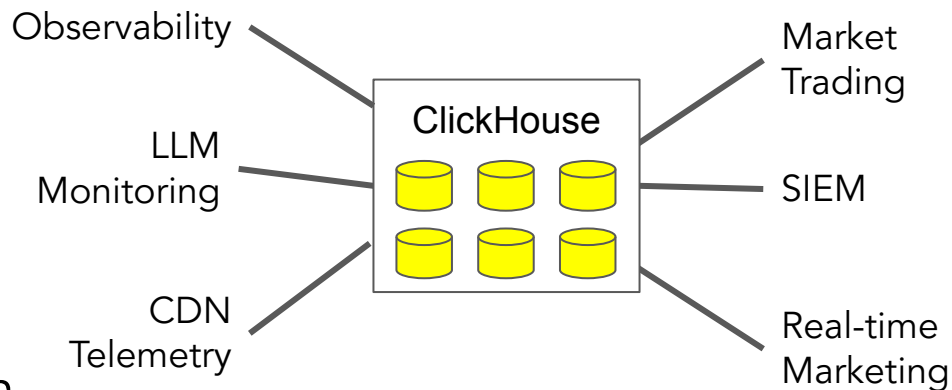
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



41.8k GitHub Watchers  
Can't Be Wrong!



# Lesson #1

ClickHouse runs anywhere.  
Find the one that works for  
you.



ClickHouse installation is great on Linux! (Including WSL2)

```
curl https://clickhouse.com | sh
```



Easiest install ever



Hard to configure server

# For a more conventional install use apt or rpm

# Ubuntu example

```
sudo apt-get install -y clickhouse-server clickhouse-client  
sudo systemctl start clickhouse-server
```

## ClickHouse Official Builds

Great for early adopters

Monthly + 2 LTS releases per year

1 year of support

<https://clickhouse.com/install>

## Altinity Stable Builds

Great for enterprises seeking stability

Based on upstream LTS releases

3 years of support, feature backports

<https://builds.altinity.cloud>



# Use Docker on MacOS or for clusters (docker compose)

## **ClickHouse Official Builds**

Tag: clickhouse/clickhouse-server

## **Altinity Stable Builds**

Tag: altinity/clickhouse-server

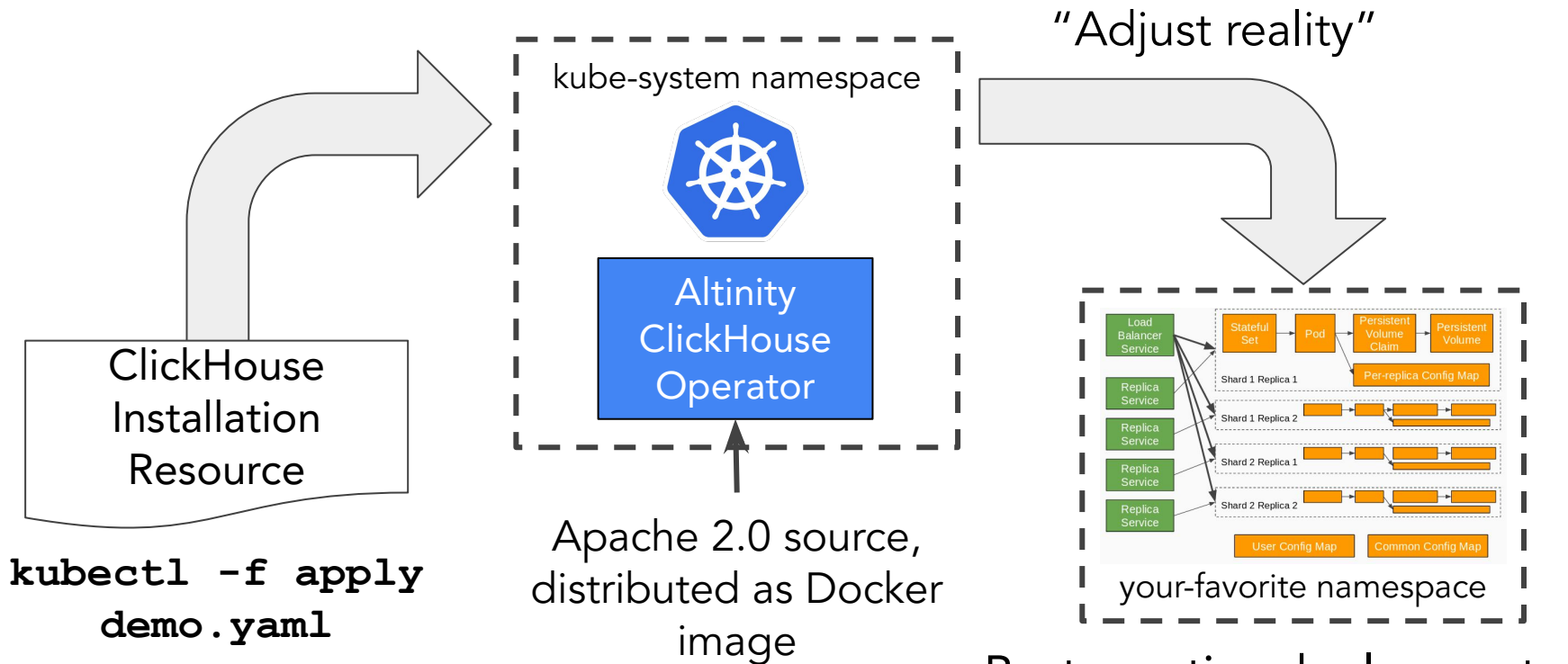
```
mkdir $HOME/clickhouse-data
```

```
docker run -d --name altinity-stable-build \  
  --ulimit nofile=262144:262144 \  
  --network=host \  
  --volume=$HOME/clickhouse-data:/var/lib/clickhouse \  
  altinity/clickhouse-server:24.8.14.10501.altinitystable
```





# Run ClickHouse on Kubernetes to build clusters quickly



Best practice deployment 9



# Or you could skip the setup and just run in a cloud

**ClickHouse Cloud**

<https://clickhouse.com>

SaaS version of ClickHouse with  
Snowflake-like convenience and  
built-in tools.

**Altinity.Cloud**

<https://altinity.com>

Cloud platform with SaaS and  
BYOC models. Runs any version  
of open source ClickHouse.



## Lesson #2

Pay attention to partitioning,  
sorting, and compression.

They make column storage  
work.



# ClickHouse tables are built for fast query

```
CREATE TABLE default.ontime_ref(  
    `Year` UInt16,  
    `Quarter` UInt8,  
    `Month` UInt8,  
    `FlightDate` Date,  
    `Carrier` FixedString(2),  
    . . .  
)  
ENGINE = MergeTree  
PARTITION BY Year  
ORDER BY (Carrier, FlightDate)
```

Columns default to  
LZ4 compression

Standard engine for  
fast analytics

How to break table  
into parts

How to sort rows  
within parts



## Finding airlines with the most cancellations in a year

```
SELECT Carrier, toYear(FlightDate) AS Year,  
       (sum(Cancelled) / count(*)) * 100. AS cancelled_pct  
FROM default.ontime_ref  
GROUP BY Carrier, Year HAVING cancelled_pct > 1.  
ORDER BY cancelled DESC LIMIT 10
```

	Carrier	Year	cancelled_pct
1.	G4	2020	16.733186040434276
2.	EA	1989	10.321500966388536
3.	WN	2020	9.284307653599388

. . .

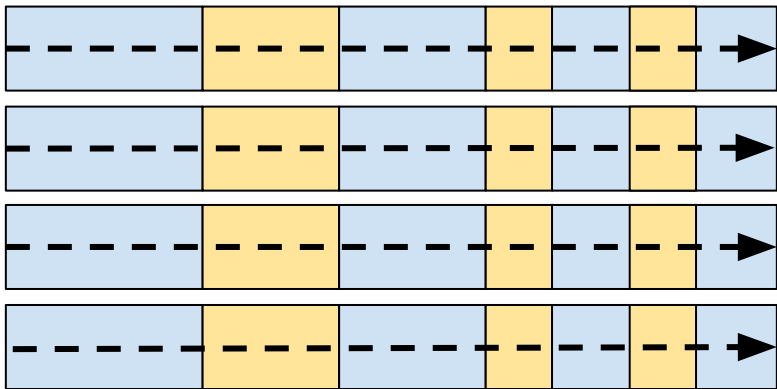
10 rows in set. Elapsed: 0.674 sec. Processed 196.51 million  
rows, 982.57 MB (291.68 million rows/s., 1.46 GB/s.)10 rows



# ClickHouse stores table data in compressed columns

PostgreSQL, MySQL

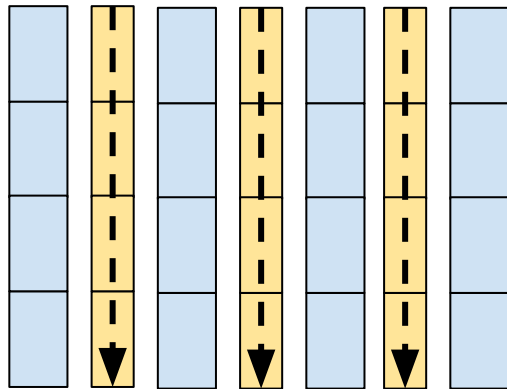
Read all columns in row



Rows minimally or not compressed

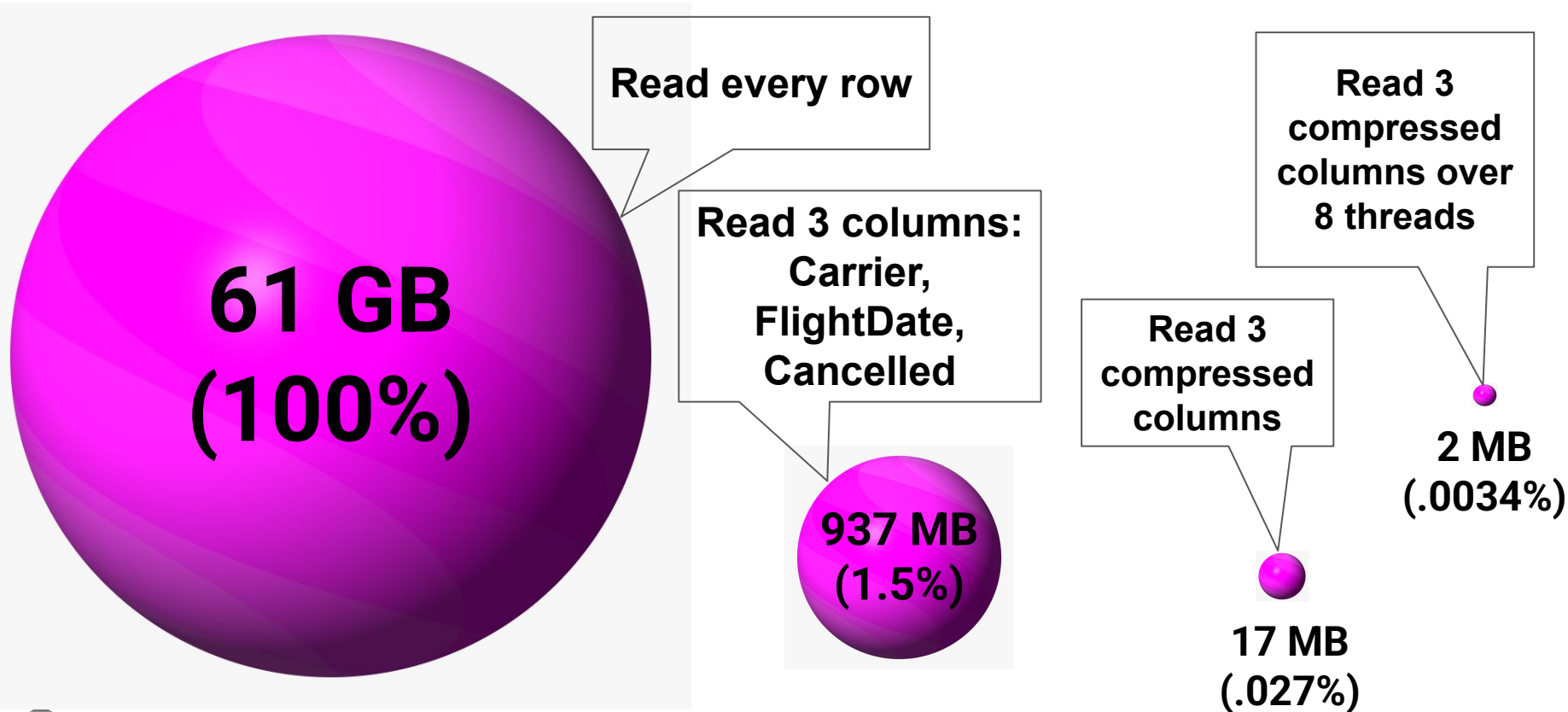
ClickHouse

Read only selected columns



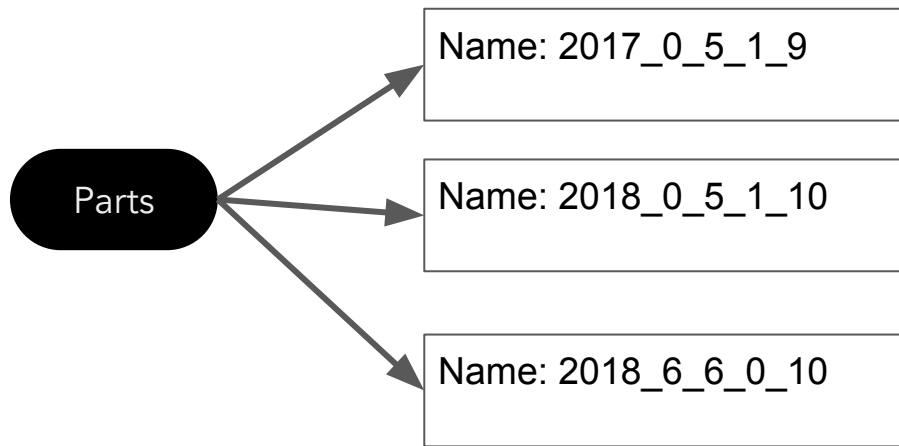
Columns highly compressed

# Visualizing effect of columns and compression on I/O



## Best practice: partition by time

```
CREATE TABLE default.ontime_ref( . . .)  
ENGINE = MergeTree  
PARTITION BY Year ORDER BY (Carrier, FlightDate)
```

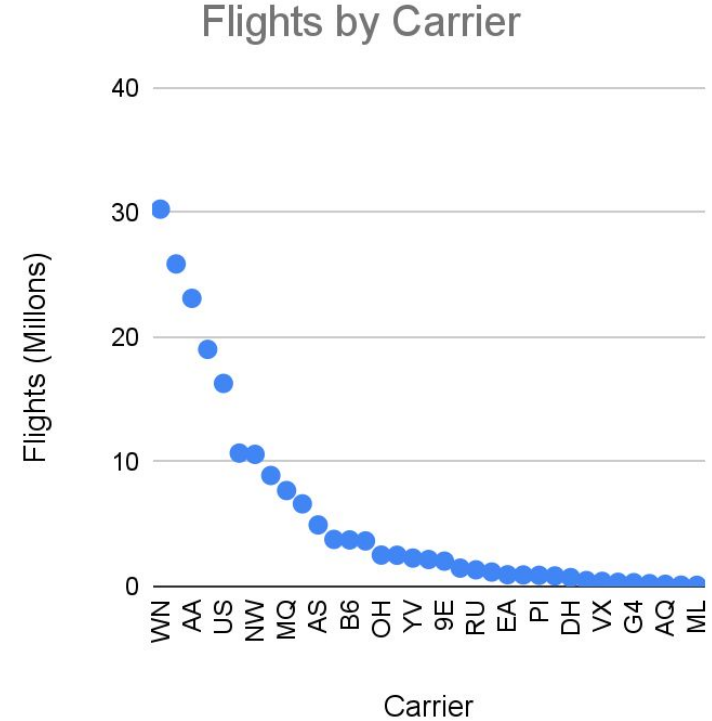
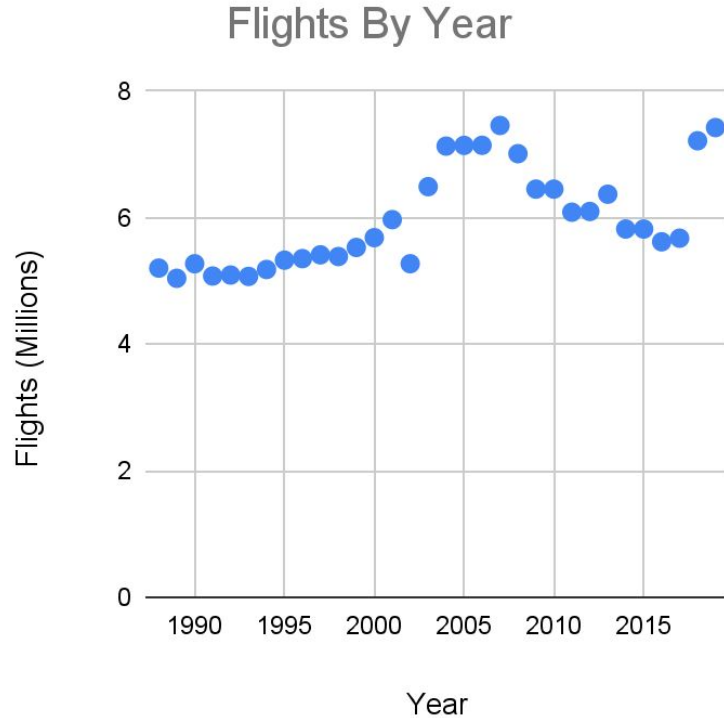


Rule of thumb:

Choose partitions that  
result in ~1000 parts  
or less



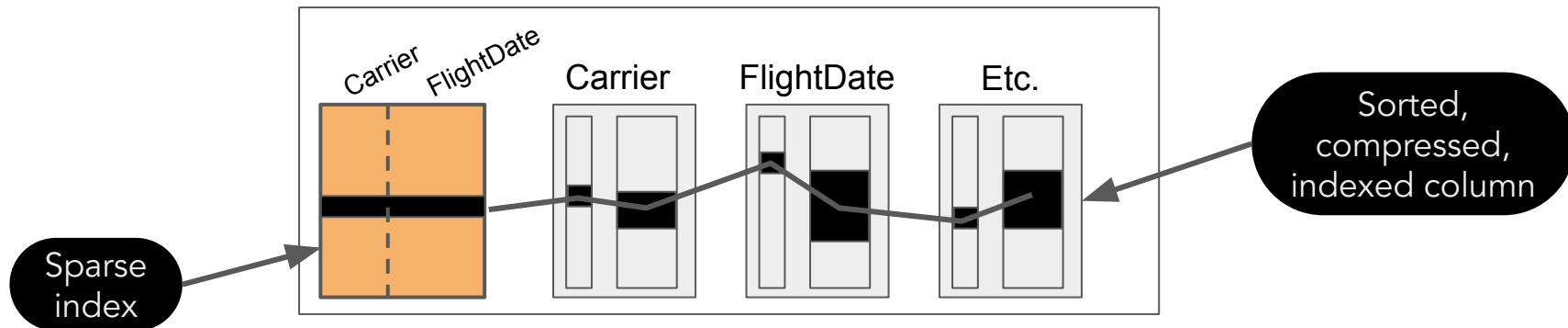
# Why it's better to partition by time



# Order by increasing cardinality, with tenant first

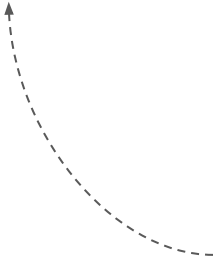
```
CREATE TABLE default.ontime_ref( . . .)  
ENGINE = MergeTree  
PARTITION BY Year ORDER BY (Carrier, FlightDate)
```

Name: 201905\_510\_815\_3



# Compress to taste, any time you want to

```
CREATE TABLE default.ontime_ref( . . .)  
ENGINE = MergeTree  
PARTITION BY Year ORDER BY (Carrier, FlightDate)  
TTL FlightDate + INTERVAL 6 MONTH RECOMPRESS CODEC (ZSTD(1)),  
    FlightDate + INTERVAL 12 MONTH RECOMPRESS CODEC (ZSTD(10))
```



Automatically increase  
compression over time



# Figure out compression with amazing system tables!

```
SELECT
    count() ,
    formatReadableSize(sum(data_compressed_bytes) ,
    formatReadableSize(sum(data_uncompressed_bytes)
FROM system.columns
WHERE (database = 'default') AND (`table` = 'ontime_ref')
AND (name IN ('Carrier', 'FlightDate', 'Cancelled'))
```

Other great tables: system.parts and system.tables



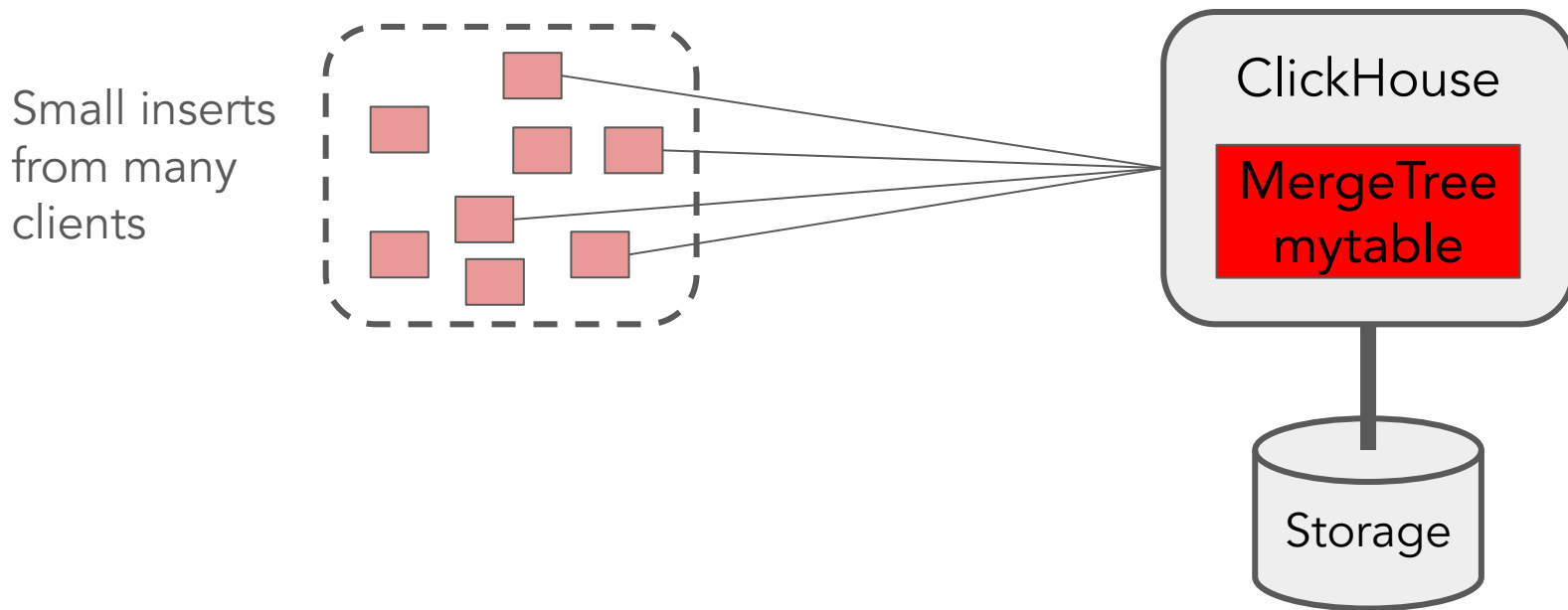
## Lesson #3

ClickHouse likes big parts!

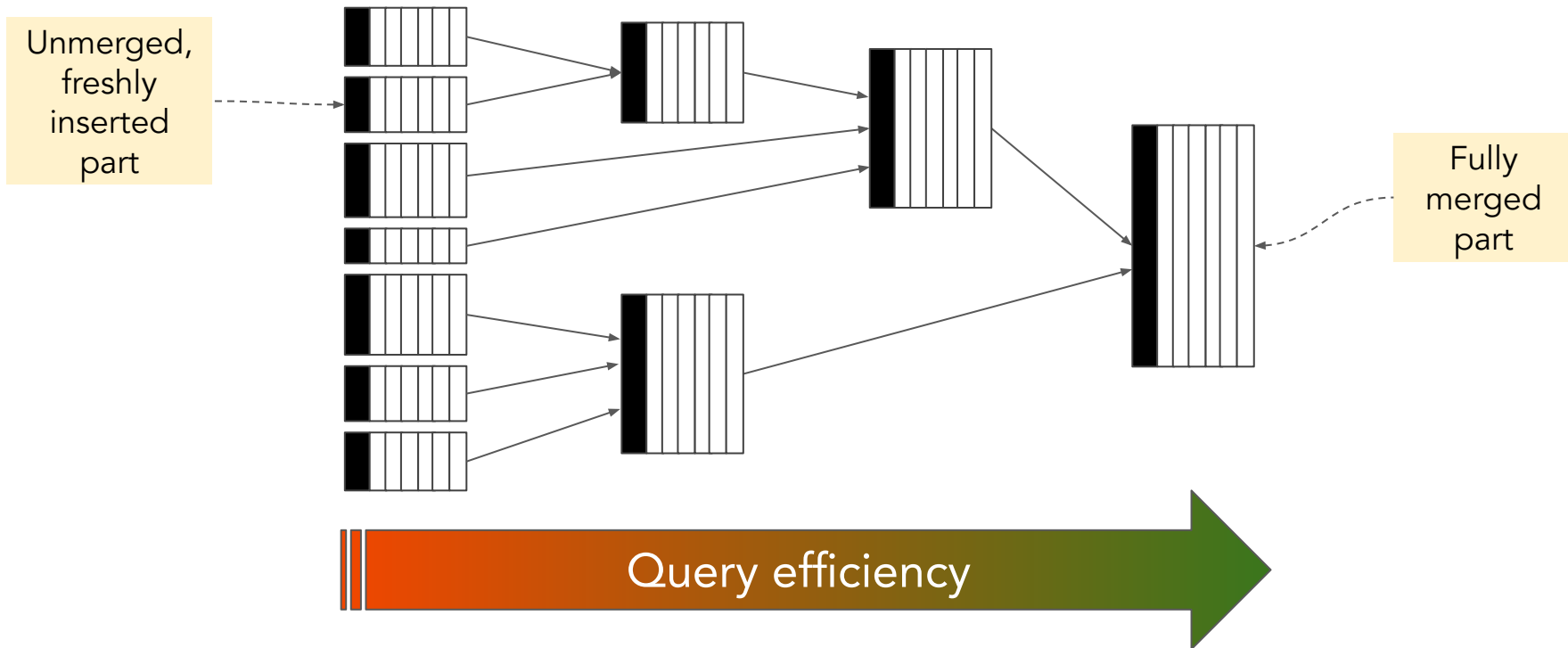
Help out by making inserts as big as possible.



# Small inserts can crush your ClickHouse server



# Lots of small parts == slow queries and high merge load



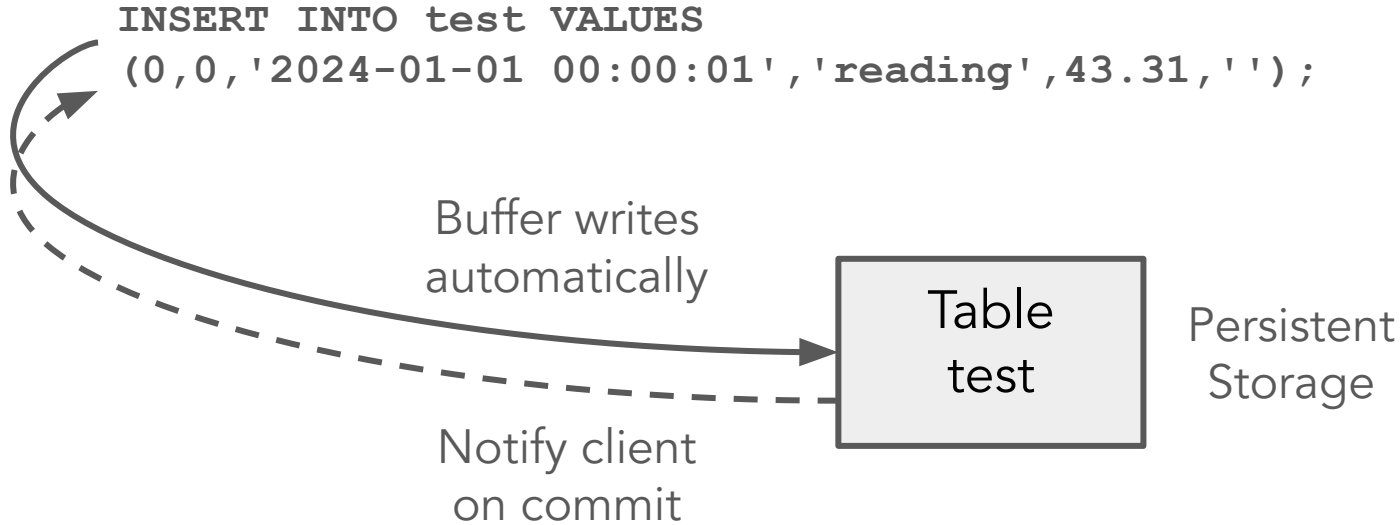
## Fix #1: Use big batches in your application

```
#!/bin/bash
INSERT='INSERT+INTO+mytable+Format+CSVWithNames'
cat test.csv | curl -X POST --data-binary @- \
    "http://localhost:8123/?query=${INSERT}"
```





## Fix #2: Enable async inserts



<https://kb.altinity.com/altinity-kb-queries-and-syntax/async-inserts/>

# Enable async inserts using property settings

```
CREATE SETTINGS PROFILE IF NOT EXISTS `async_profile`  
ON CLUSTER '{cluster}'  
SETTINGS
```

```
    async_insert = 1,  
    wait_for_async_insert=1,  
    async_insert_busy_timeout ms = 10000,  
    async_insert_use_adaptive_busy_timeout = 0
```

```
;
```

```
CREATE USER IF NOT EXISTS async ON CLUSTER '{cluster}'  
    IDENTIFIED WITH sha256_password BY 'topsecret' HOST ANY  
    SETTINGS PROFILE `async_profile`  
;
```

Use async insert  
and wait for answer

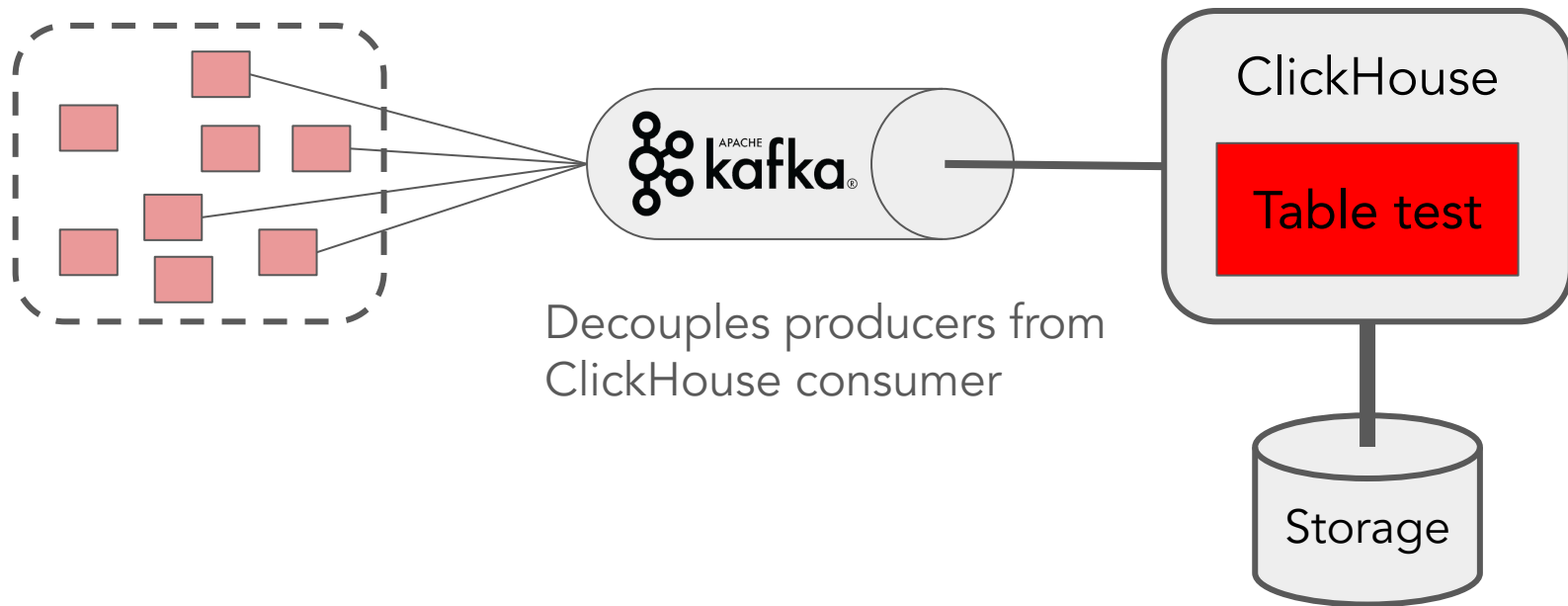
Wait this long

Don't let  
ClickHouse set  
automatic values

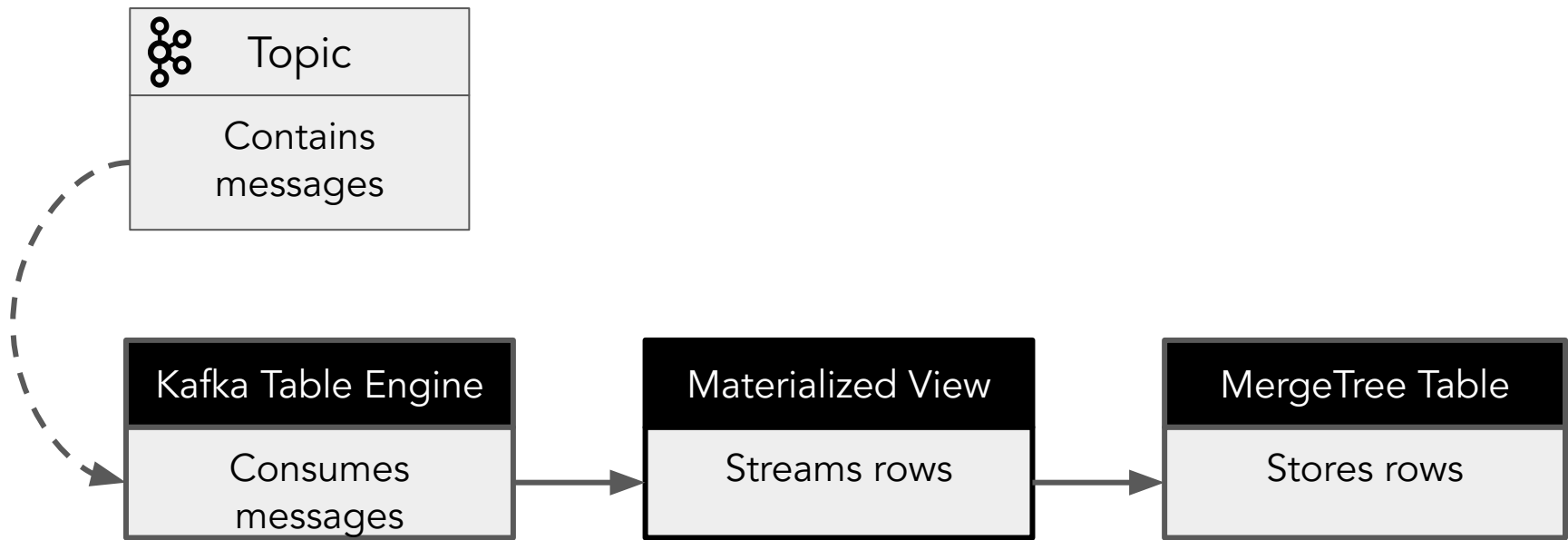
User with settings



## Fix #3: Use Kafka to buffer data from upstream producers



# Using the Kafka table engine to read from Kafka



<https://kb.altinity.com/altinity-kb-integrations/altinity-kb-kafka/>

## Lesson #4

Joins are different in ClickHouse.  
Learn to use them properly.



## ClickHouse can do joins (of course!)

```
SELECT Dest, Name as AirportName, count(*) Flights
FROM default.ontime_ref o
LEFT JOIN default.dot_airports a ON (a.AirportID = o.DestAirportID)
GROUP BY Dest, AirportName ORDER BY Flights DESC LIMIT 10
```

Dest	AirportName	Flights
DEN	Denver International	12103062
ATL	Hartsfield Jackson Atlanta International Airport	10605117
...		
LAS	McCarran International Airport	4361486

10 rows in set. Elapsed: 2.581 sec. Processed 196.52 million rows,  
982.84 MB (76.13 million rows/s., 380.75 MB/s.)



# ClickHouse has a rich set of algorithms and join types

## Join Algorithms

hash  
partial\_merge  
parallel\_hash  
grace\_hash  
full\_sorting\_merge  
direct

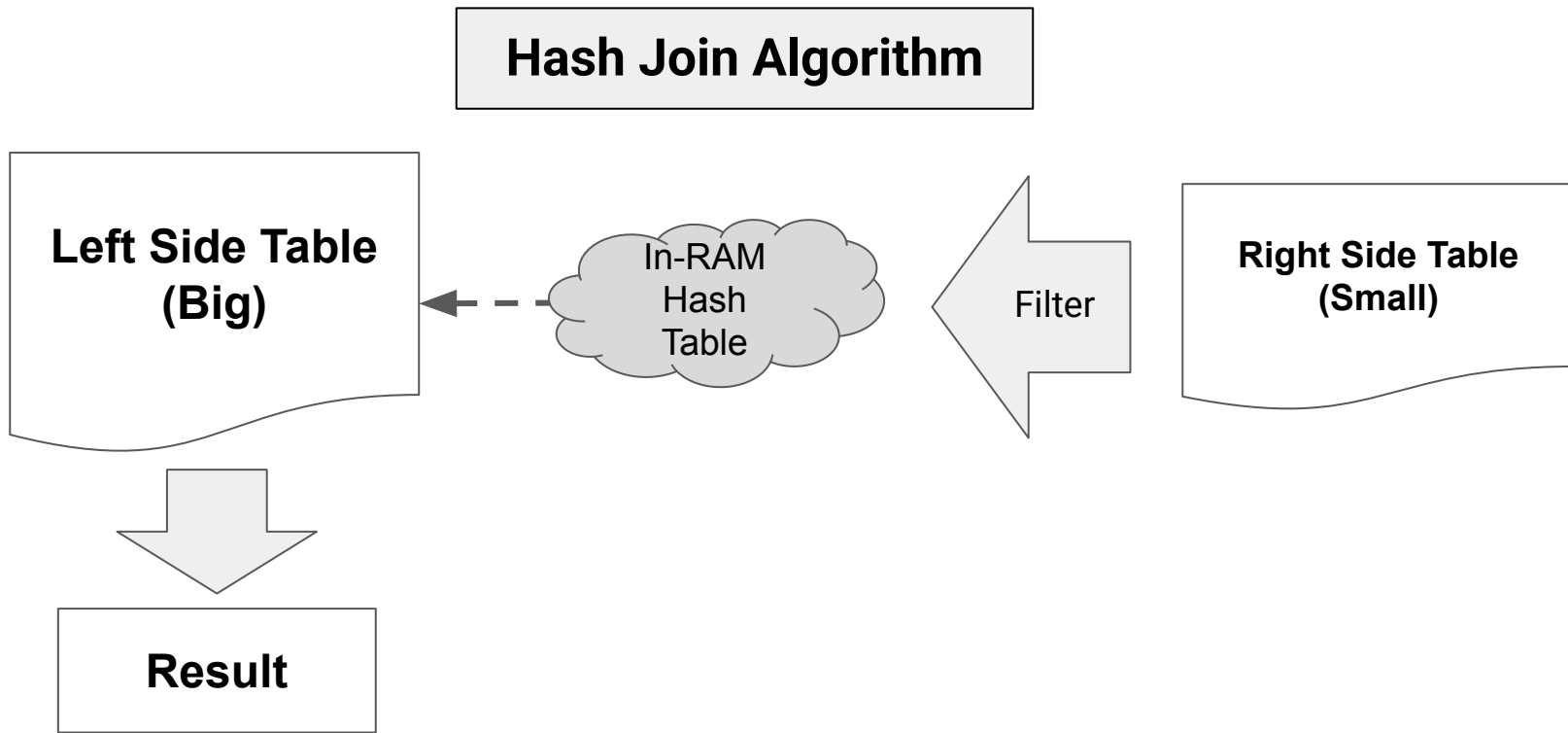
Caveat: BI-style large table  
joins are "difficult"

## Join Types

inner join  
left [outer] join  
full [outer] join  
right [outer] join  
cross join  
semi join  
anti join  
asof join  
paste join

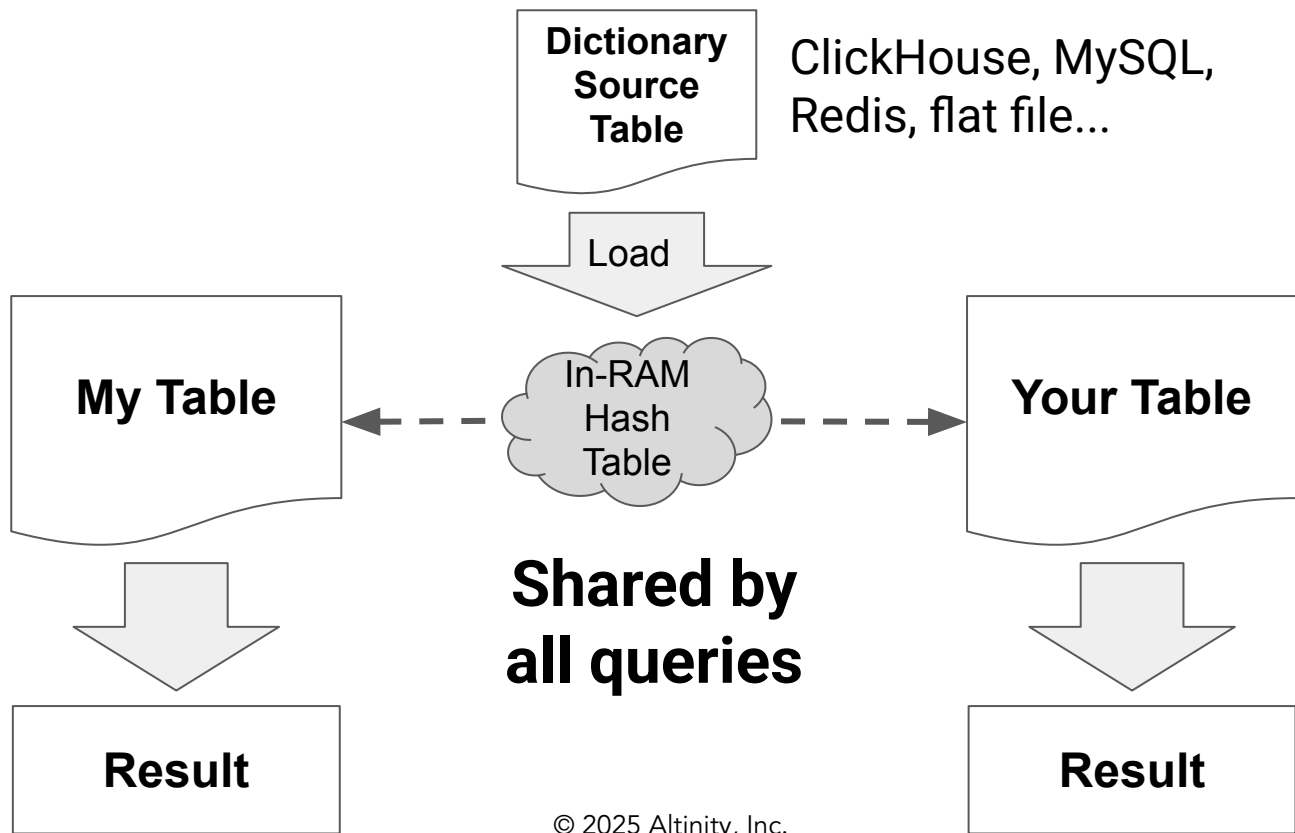


# How ClickHouse does joins between tables





# Dictionaries are an alternative to joins



# Creating a dictionary on a table

```
CREATE DICTIONARY default.dot_airports_dict
(
    `AirportID` UInt64,
    `City` String,
    `State` String,
    `Name` String
)
PRIMARY KEY AirportID
SOURCE (CLICKHOUSE (TABLE 'dot_airports' DB 'default'))
LIFETIME (MIN 180 MAX 300)
LAYOUT (FLAT ())
```

<https://clickhouse.com/docs/en/sql-reference/statements/create/dictionary/>



# Restructure joins to reduce data scanning

```
SELECT Dest, Name, count(*) c, avg(ArrDelayMinutes) ad
  FROM default.ontime_ref
LEFT JOIN default.dot_airports ON DestAirportID = AirportID
GROUP BY Dest, Name HAVING c > 100000
ORDER BY ad DESC LIMIT 10
```

15.820 sec.



```
SELECT Dest, Name, c AS flights, ad
FROM (SELECT DestAirportID, any(Dest) as Dest,
           count(*) c, avg(ArrDelayMinutes) ad
      FROM default.ontime_ref
     GROUP BY DestAirportID HAVING c > 100000
     ORDER BY ad DESC LIMIT 10) a
LEFT JOIN default.dot_airports ON DestAirportID = AirportID
```

3.472 sec.

Smaller base  
query



# How can we handle table joins on very large tables??

Use case: join sensor restart with temperature reading data

## Restart times

msg_type	sensor_id	time
'restart'		

JOIN key

msg_type	sensor_id	time	temperature
'reading'			
'reading'			
'reading'			

## Temperature readings

## Aggregates on joined data

month	count of readings	count of restarts	min temp reading

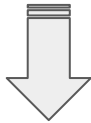
## Hint: Aggregation runs in a single pass

**No need to  
move data**

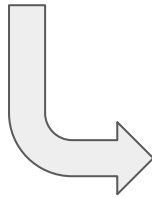
**Parallelizes!**

**Intermediate  
results are  
reusable**

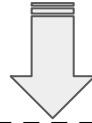
1	2	3
---	---	---



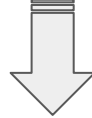
Sum = 6
Count = 3



1	3	5
---	---	---

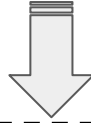


Sum = 9
Count = 3

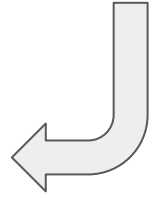


$$\frac{6 + 9 + 5}{3 + 3 + 4} = 2$$

0	5	0	0
---	---	---	---



Sum = 5
Count = 4



# Conditional aggregation on different entities in one table

```
SELECT toYYYYMM(time) AS month,  
       countIf(msg_type = 'reading') AS readings,  
       countIf(msg_type = 'restart') AS restarts,  
       minIf(temperature, msg_type = 'reading') AS min,  
       round(avgIf(temperature, msg_type = 'reading')) AS avg,  
       maxIf(temperature, msg_type = 'reading') AS max  
FROM test.readings_multi WHERE sensor_id = 3  
GROUP BY month ORDER BY month ASC
```

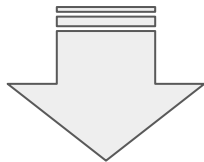
month	readings	restarts	min	avg	max
201901	44640	1	0	75	118.33
201902	40320	0	68.09	81	93.98
201903	15840	0	73.19	84	95.3

## Lesson #5

ClickHouse prioritizes speed and scalability over anything else.  
Lean on the trade-offs.

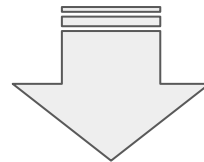


# Size



## Immutable data

# Speed



## Eventual consistency

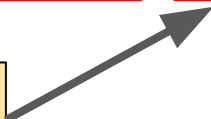


# ReplacingMergeTree deduplicates rows in ORDER BY


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

Pro tip: Use PRIMARY KEY to reduce size of index if ORDER BY is long

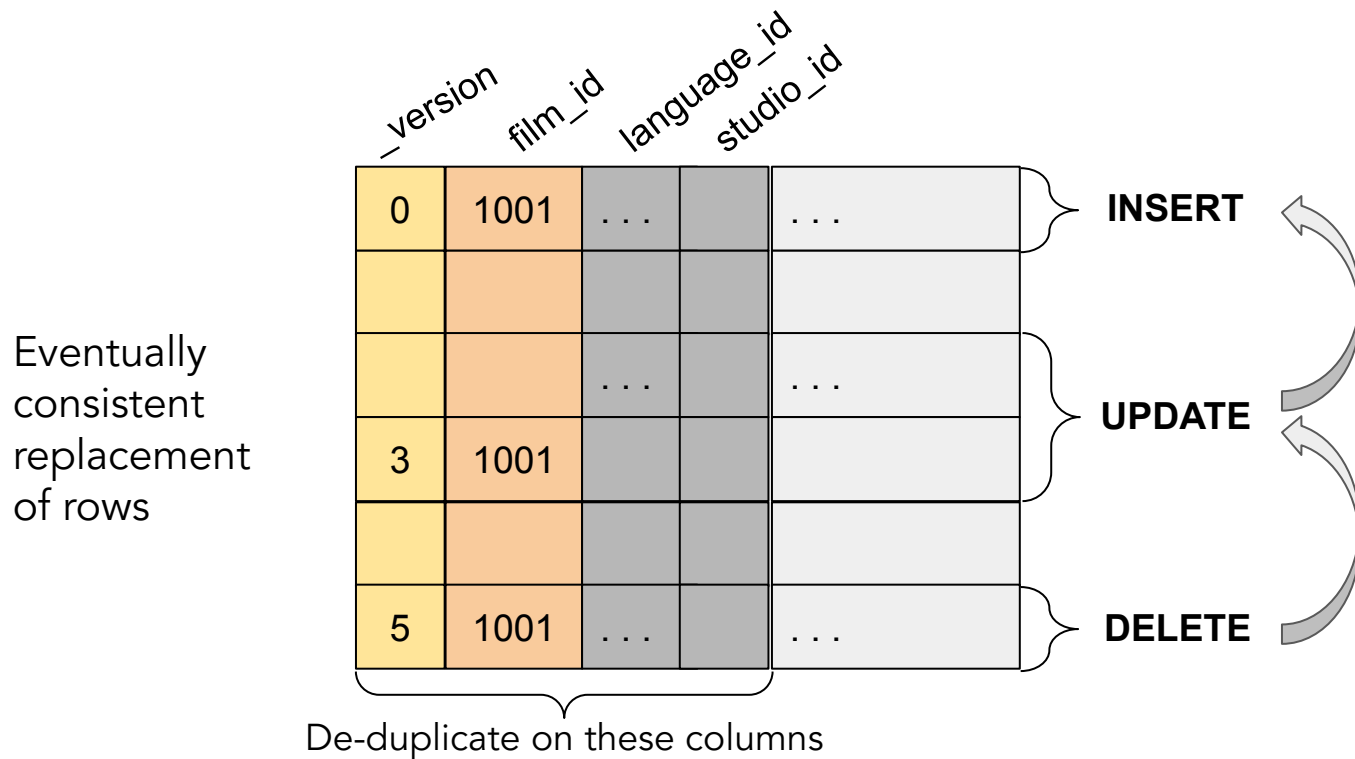
Other cols go to left



Row key goes on right (if you have one)



# How ReplacingMergeTree works



# Updating a row in the RMT table

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

```
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

Unmerged  
rows!

# Rows are replaced when merges occur

Part

0	1001	1		...

Merged Part

3	1001	2		

Part


				...
3	1001	2		

Pro tip: never assume rows will merge fully

# FINAL keyword merges data dynamically

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

Adds initial scan to  
merge rows



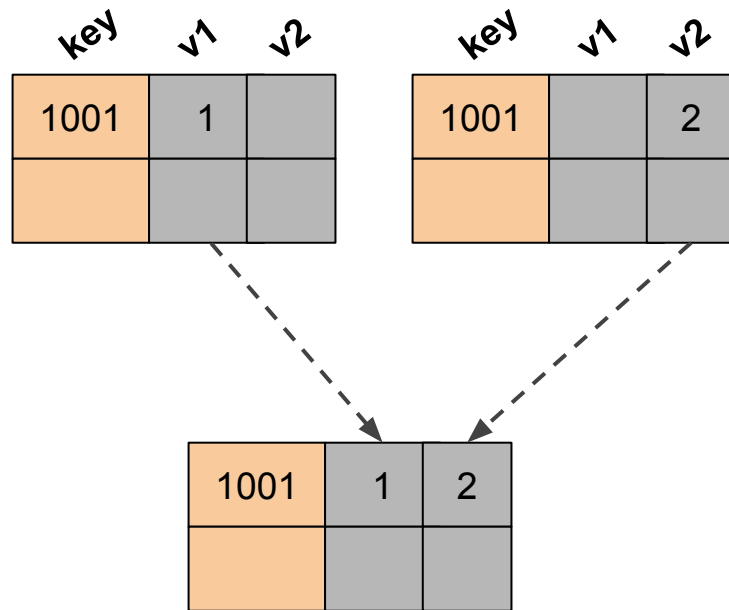
title	release_year
Blade Runner - Director's Cut	1982

<https://altinity.com/blog/clickhouse-replacingmergetree-explained-the-good-the-bad-and-the-ugly>



# ClickHouse has a raft of tables built on similar principles

- ReplacingMergeTree
- SummingMergeTree
- AggregatingMergeTree
- CollapsingMergeTree
- VersionedCollapsingMergeTree
- CoalescingMergeTree



# Wrap-up and Questions



# Summary of 5 things every beginner should know

- Pick the right install for your work. There are lots of them!
- Pay close attention to partitioning, ordering, and compression in tables.
- Make your inserts as big as possible.
- Joins are different in ClickHouse. You need to think about the mechanisms.
- Immutable data and eventual consistency are the foundation of big systems.  
Lean in on features that use them.







Join our **Open Source Analytics Festival**

**Where:**

IBM Silicon Valley Lab  
555 Bailey Ave  
San Jose, CA 95141

**When:** Thursday 31 July 2025

# Thank you! Questions?

Contact us to learn more about  
Altinity.Cloud and Enterprise Support

<https://altinity.com>

<https://altinity.com/slack>