Altinity Quickstart for ClickHouse

Creating your first application

Let's make some introductions

Us

Database geeks with centuries of experience in DBMS and applications

You

Applications developers looking to learn about ClickHouse

Altinity

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

What's a ClickHouse?

ClickHouse is a SQL Data Warehouse

Understands SQL

Runs on bare metal to cloud

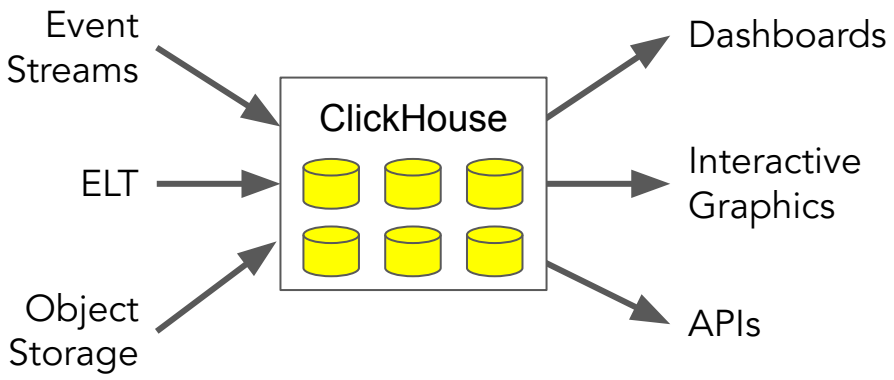
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's the core engine for real-time analytics

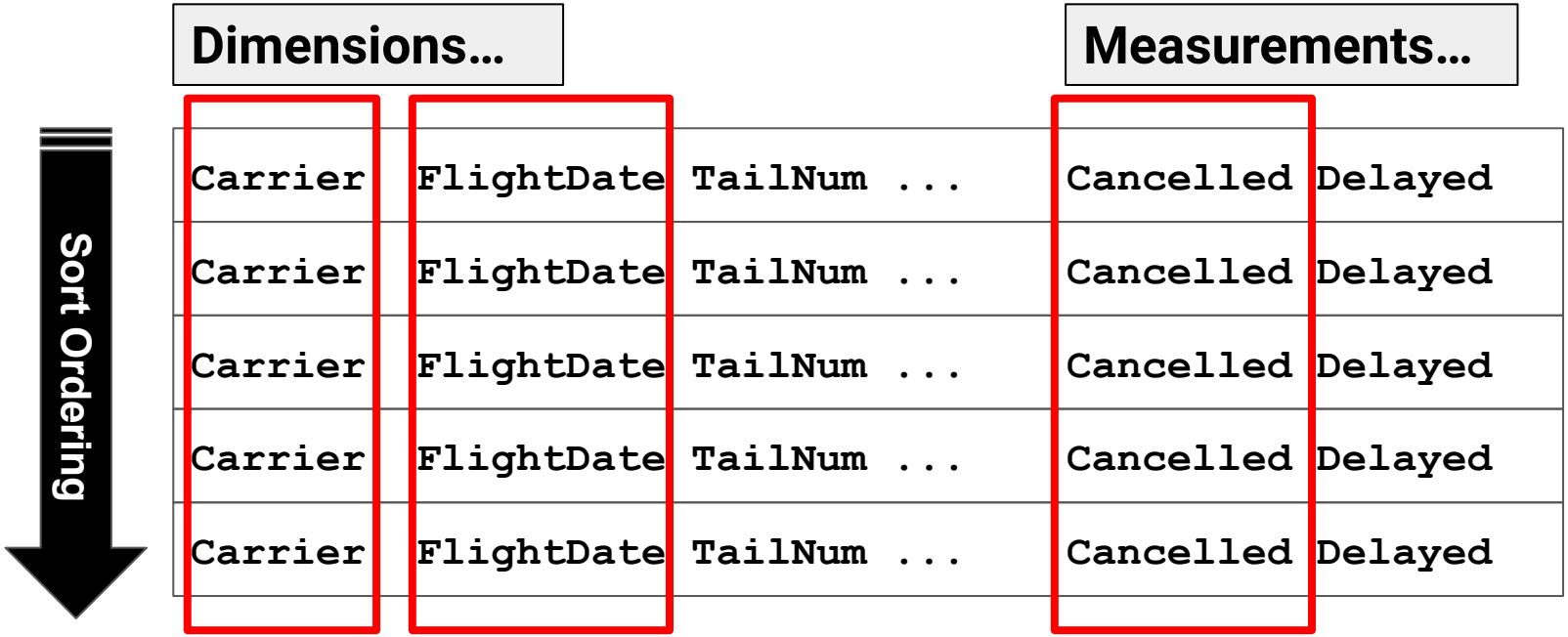
What ClickHouse Is Not…

ClickHouse does not have full ACID transactions and updates are slow

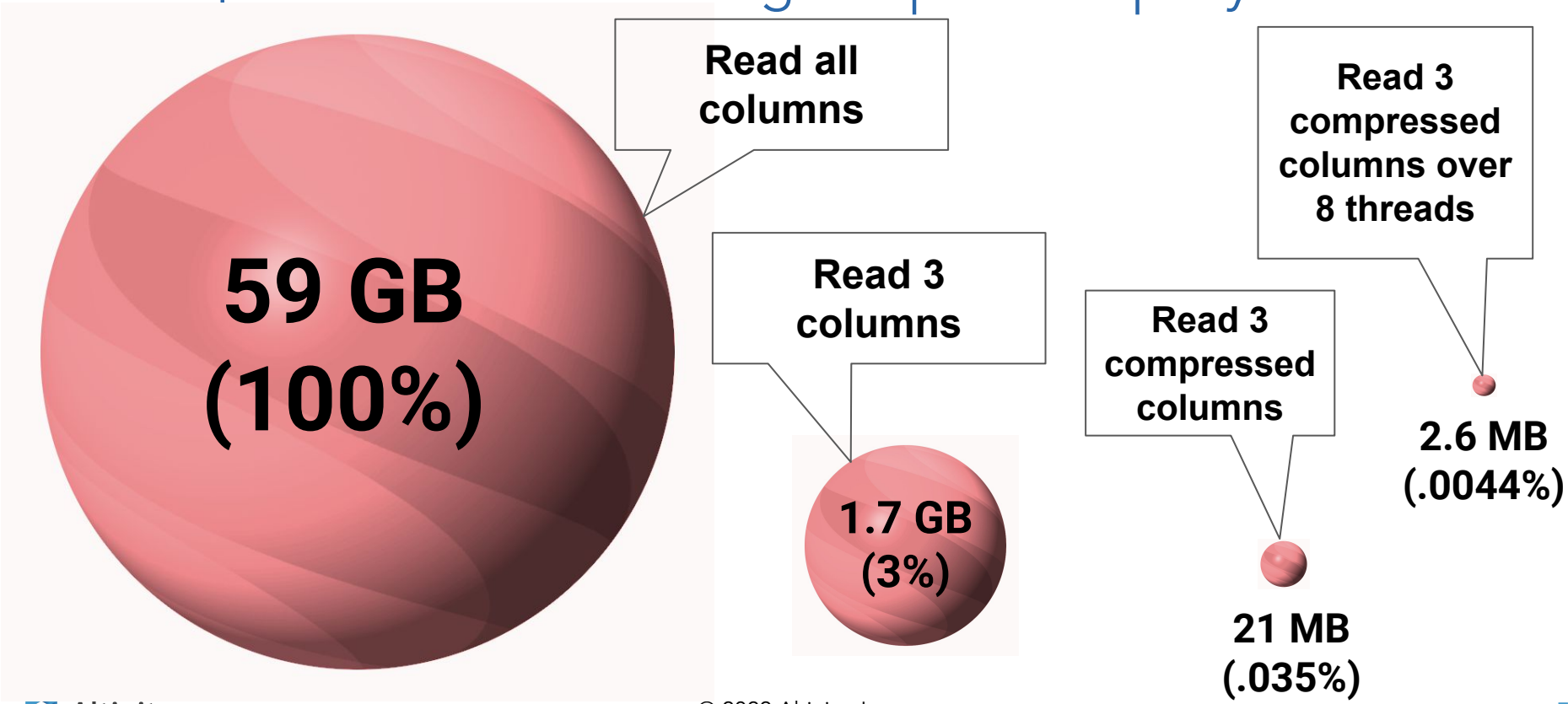
ClickHouse values speed over SQL standards and runs anywhere

ClickHouse is designed to read large blocks of data for a few users

ClickHouse stores data in columns, not rows



It compresses data and has great parallel query



Installing ClickHouse and Connecting

Install ClickHouse from a build repo

ClickHouse Community Builds
Monthly builds
LTS builds every 6 months
(1 year of support)
https://clickhouse.com

Altinity Stable Builds

Prod-ready LTS builds only
(3 years of support)
https://docs.altinity.com

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

Install ClickHouse using Docker

ClickHouse Community Builds
Tag: clickhouse/clickhouse-server

Altinity Stable Builds
Tag: altinity/clickhouse-server

```
mkdir $HOME/clickhouse-data

docker run -d --name clickhouse-server \
  --ulimit nofile=262144:262144 \
  --volume=$HOME/clickhouse-data:/var/lib/clickhouse \
  -p 8123:8123 -p 9000:9000 \
  clickhouse/clickhouse-server
```

Make ClickHouse happy

Persist data

Make ports visible

Install ClickHouse in the cloud

(Lots of vendors)

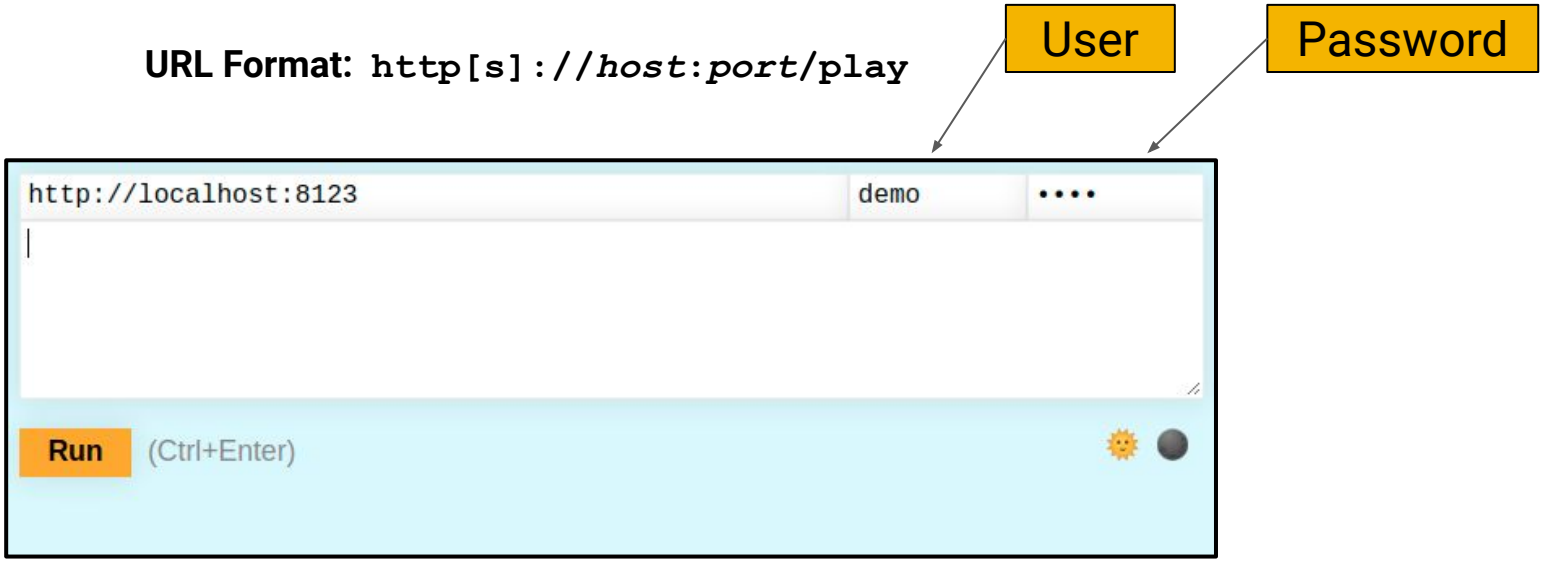
- Altinity.Cloud -- Runs in AWS and GCP as well as in user Kubernetes clusters; tightly integrated support
- DoubleCloud – ClickHouse + Kafka + DataLens
- Aiven - Hosted ClickHouse (and many others, too)
- ClickHouse Inc – Hosted ClickHouse a la BigQuery

Connecting from the command line

```
-- Connect to ClickHouse running locally on your laptop.
clickhouse-client

-- Same thing with explicit options.
clickhouse-client \
  --host=localhost \
  --port=9000 --user=default

-- Connect to a secure ClickHouse running in the Cloud.
clickhouse-client \
  --host=github.demo.altinity.cloud \
  --port=9440 --secure \
  --user=demo --password=demo
```

Connecting to built-in web UI

URL Format: `http[s]://host:port/play`



Example: https://github.demo.altinity.cloud:8443/play (user=demo/pw=demo)

Example: http://localhost:8123/play (user=default/pw=null)

Finding your way around ClickHouse

Databases and tables

- Every ClickHouse has databases and tables
- ClickHouse databases are like folders:
  - MySQL databases
  - PostgreSQL schemas
- Every connection has a “current” database
  - Tables in the current database don’t need a database name

Handy commands for navigation

SQL Command	What it does
`show databases`	List databases in ClickHouse
`select currentDatabase()`	What's my current database?
`show tables [from` *name*`]`	Show tables [in a particular database]
`describe table` *name*	Show the structure of table
`SELECT count() FROM ontime`	Select from table ontime in current database
`SELECT count() FROM anotherdb.ontime`	Select from table ontime in database anotherdb

Selecting the database

- clickhouse-client allows you to switch between databases in a session

```
clickhouse101 :) use system

Ok.
```

- Most APIs require you to set the database in the URL.
  - HTTP: https://rhodges_59945:<pw>@clickhouse101.demo.altinity.cloud:8443?database=system
  - JDBC: jdbc:clickhouse://clickhouse101.demo.altinity.cloud:8443/rhodges_59945
  - Python SQLAlchemy: clickhouse+native://demo:demo@github.demo.altinity.cloud/default?secure=true

Special databases in ClickHouse

default -- Standard database for user data in a new ClickHouse installation

config.xml: `<default_database>default</default_database>`

system -- Database for system tables

```
SHOW TABLES FROM system
┌─name──────────────────────────────┐
│ aggregate_function_combinators │
│ asynchronous_metric_log        │
│ asynchronous_metrics           │
│ build_options                  │
│ clusters                       │
│ collations                     │
│ columns                        │
```

Your friends :)

Creating Your First Table and Loading Data

Let's create a table!

```
CREATE TABLE IF NOT EXISTS sdata (
    DevId Int32,
    Type String,
    MDate Date,
    MDatetime DateTime,
    Value Float64
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(MDate)
ORDER BY (DevId, MDatetime)
```

Table columns

Table engine type

How to break data into parts

How to index and sort data in each part

Let's insert some data the hard way…

```
INSERT INTO sdata VALUES
(15, 'TEMP', '2018-01-01', '2018-01-01 23:29:55', 18.0),
(15, 'TEMP', '2018-01-01', '2018-01-01 23:30:56', 18.7)
```

OK, nobody really does this.

Use input formats to load raw data

Format	Description
Values	Normal INSERT tuples like (‘a’, 1, 35.7)
CSV, CSVWithNames	Comma-separated values, CSV with column names
JSONEachRow	Single JSON record per row (***many*** JSON formats supported)
Protobuf	Google Protocol Buffers format
Avro, AvroConfluent	Apache Avro and Confluent/Kafka variant
Parquet	Apache Parquet columnar data format
ORC	Apache ORC (another columnar format from Hadoop)

* Try this: `select * from system.formats`
* Or this: https://clickhouse.com/docs/en/interfaces/formats/

Use clickhouse-client to load raw data from command line

Input file sdata.csv

```
DevId,Type,MDate,MDatetime,Value
59,"TEMP","2018-02-01","2018-02-01 01:10:13",19.5
59,"TEMP","2018-02-01","2018-02-01 02:10:01",18.8
59,"TEMP","2018-02-01","2018-02-01 03:09:58",18.6
59,"TEMP","2018-02-01","2018-02-01 04:10:05",15.1
59,"TEMP","2018-02-01","2018-02-01 05:10:31",12.2
```

```
cat sdata.csv | clickhouse-client \
--database=sense \
--query='INSERT INTO sdata FORMAT CSVWithNames'
```

Or with HTTP commands

Uses HTTP POST

```
#!/bin/bash

insert='INSERT%20INTO%20sense.sdata%20Format%20CSVWithNames'

curl -v "http://localhost:8123/?query=${insert}" \
 --data-binary @sdata.csv
```

Read data from file

Or you could insert from S3

```
SET max_insert_threads=32

INSERT INTO sdata
SELECT * FROM s3(
  'https://s3.us-east-1.amazonaws.com/d1-alt/d1/sdata*.csv.gz',
  'aws_access_key_id',
  'aws_secret_access_key',
  'CSVWithNames',
  'DevId Int32, Type String, MDate Date, MDatetime DateTime,
Value Float64')
```

Parallelize!

Read multiple files

Input format

Schema (Not needed after 22.1)

How to update and delete rows

```
ALTER TABLE ontime
UPDATE UniqueCarrier = '002'
WHERE UniqueCarrier = '00'

ALTER TABLE ontime
DELETE WHERE UniqueCarrier = '002'
```

Asynchronous; differs from standard SQL

Can be expensive!

SQL DELETE command available starting in 22.8!

How to find out if update/delete has finished

```
SELECT command, is_done
FROM system.mutations
WHERE table = 'ontime'
```

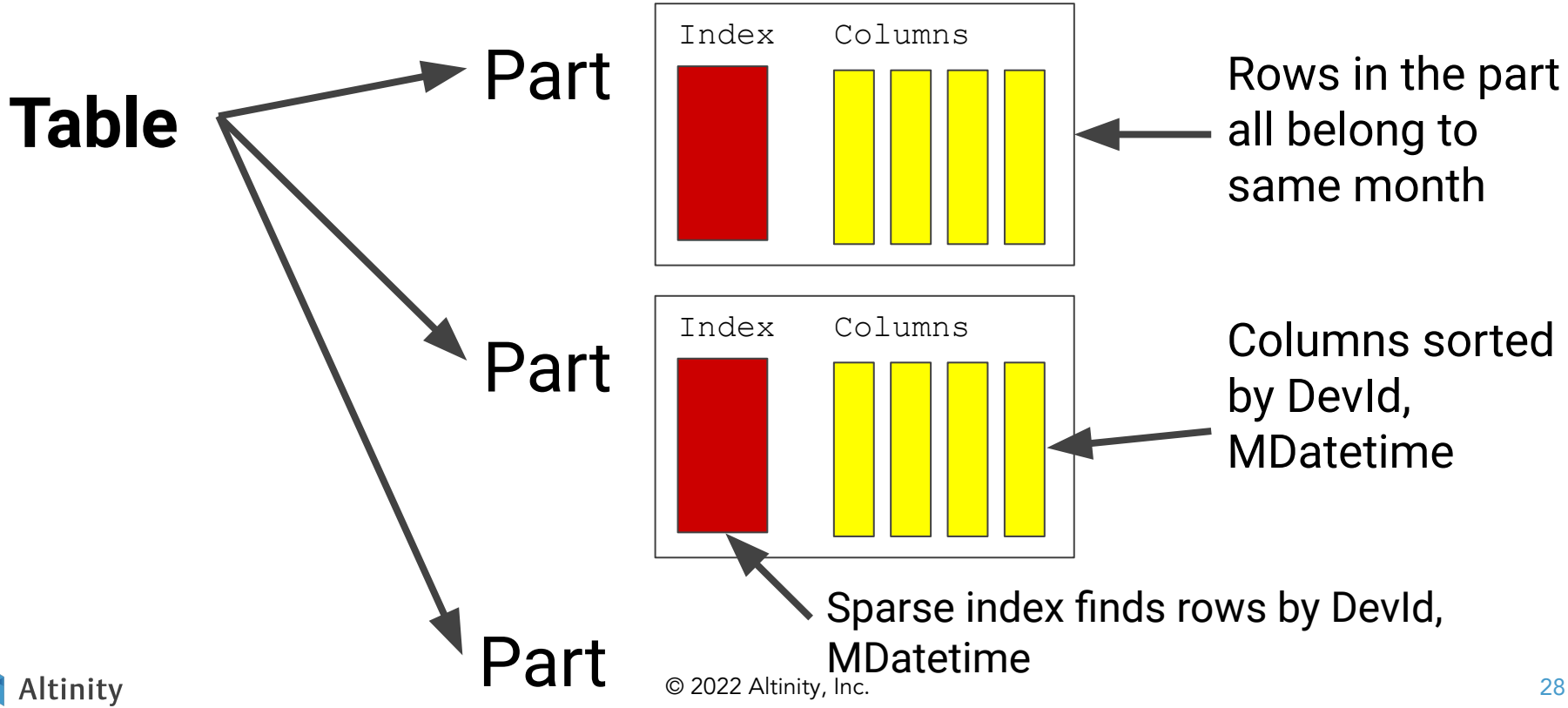
Tracks progress of “mutations”

```
┌─command──────────────────────────────────────────────────────────┬─is_done─┐
│ UPDATE UniqueCarrier = 'OO2' WHERE UniqueCarrier = 'OO'          │       1 │
│ DELETE WHERE UniqueCarrier = 'OO2'                               │       1 │
└──────────────────────────────────────────────────────────────────┴─────────┘
```

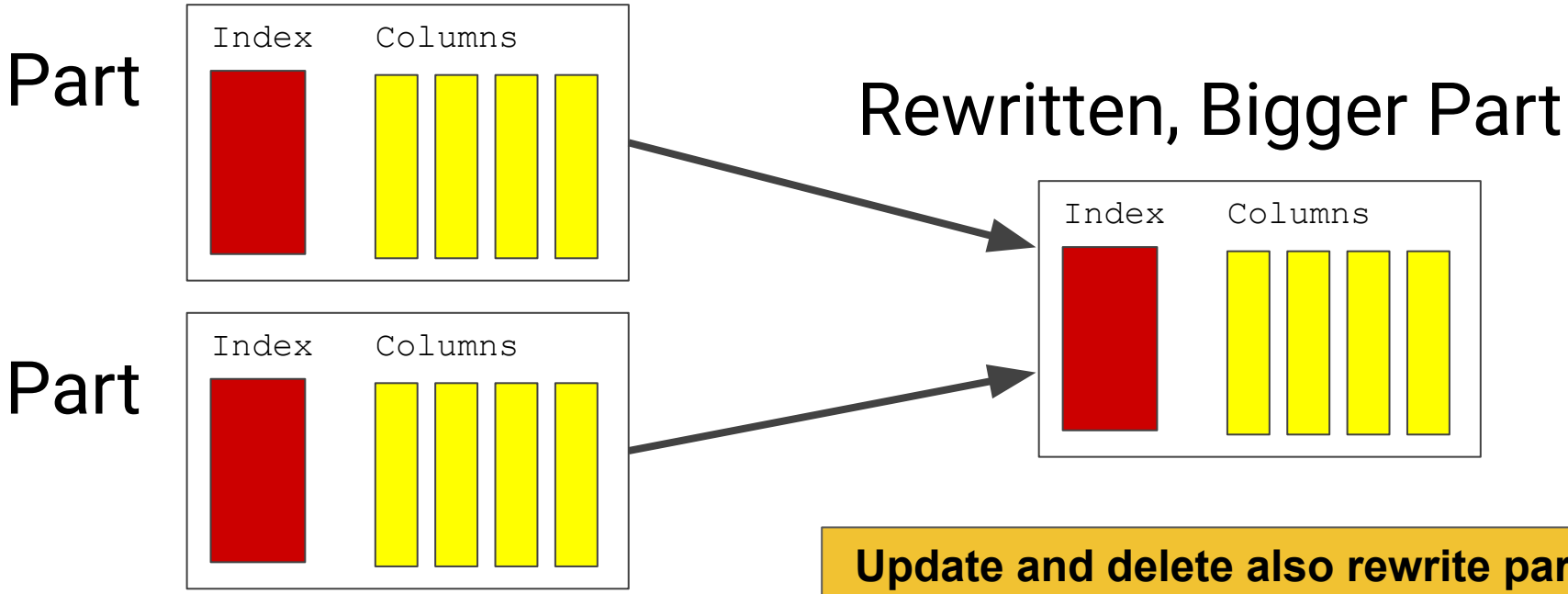
```
KILL mutation WHERE table = 'ontime'
```

Cancels mutation

What's going on down there when you INSERT?



Why MergeTree? Because it merges!



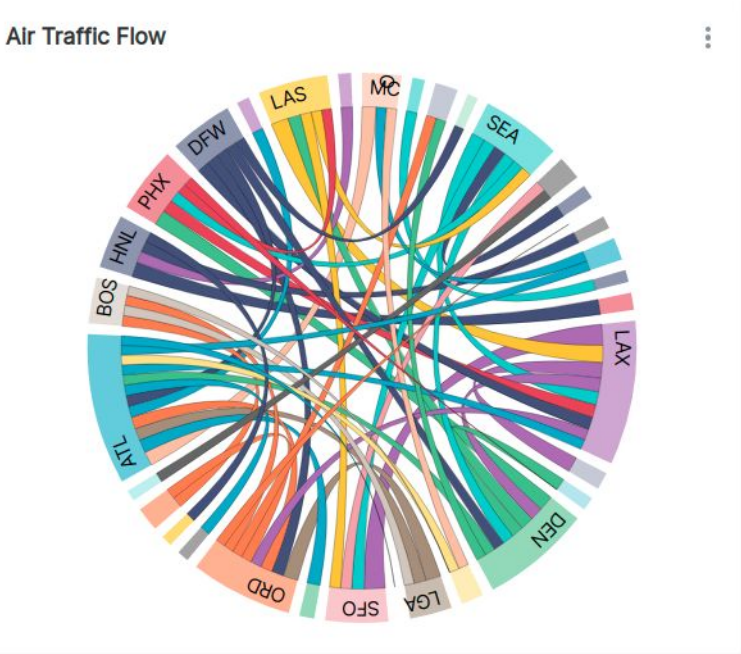
Update and delete also rewrite parts

Top three tips for table design and data loading

- Pick a PARTITION BY that gives nice, fat partitions (1-300GB, < 1000 total parts)
  - Can't decide? Partition by month.
- Use input formats to load raw data directly
- Insert large blocks of data to avoid lots of merges afterwards
  - ClickHouse will default up to 1,048,576 rows in a single part. You can increase it to tens of millions

Selecting Data

SELECT allows us to fetch interesting data



Who had the most cancelled flights in 2017?

```
SELECT Carrier,
  toYear(FlightDate) AS Year,
  sum(Cancelled)/count(*) * 100. AS cancelled
FROM ontime
WHERE Year = 2017
GROUP BY Carrier, Year
HAVING cancelled > 1.0
ORDER BY cancelled DESC
LIMIT 10
```

Dimensions

Aggregate

Range Filter

Aggregate Filter

Order of results

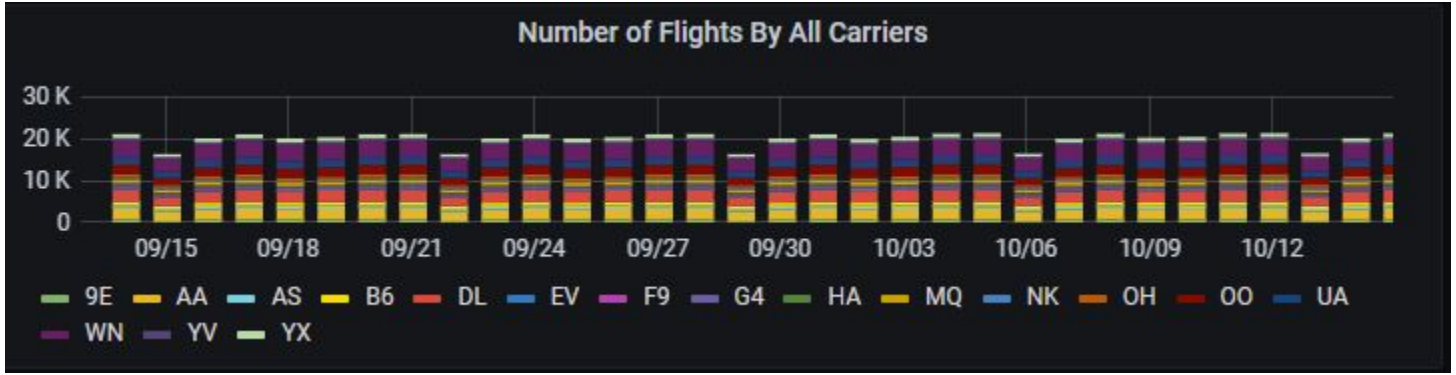
Max results

Rule #1 for selecting data in ClickHouse

Most SQL SELECT syntax
"just works" in ClickHouse

So what's different?

If you are using ClickHouse, time is probably important



Time series == Data with built-in time ordering

Using time dimensions in queries

```
SELECT
  toStartOfWeek(FlightDate) AS Week,
  Carrier,
  count() AS Flights
FROM default.ontime_ref
WHERE FlightDate BETWEEN toDate('2015-01-01')
  AND toDate('2015-06-30')
GROUP BY Week, Carrier
ORDER BY Carrier, Week
```

ClickHouse has excellent date-time support

Date -- Precision to day

DateTime -- Precision to second

DateTime64 -- Precision to nanosecond

BI tools like Grafana like DateTime values

toYear(), toMonth(), toWeek(), toDayOfWeek, toDay(), toHour(), ...

toStartOfYear(), toStartOfQuarter(), toStartOfMonth(), toStartOfHour(), toStartOfMinute(), …, toStartOfInterval()

toYYYYMM()

toYYYYMMDD()

toYYYYMMDDhhmmsss()

And many more!

ClickHouse has a very rich set of aggregates, too

Standard SQL aggregation functions

- count()
- avg()
- sum()
- min()
- max()

And many more native to ClickHouse

- any()
- anyLast()
- avgWeighted()
- uniq()
- uniqExact()
- quantile()
- quantileExact()
- simpleLinearRegression()
- groupArray()
- …

Drilling into GROUP BY syntax

Dimensions

```
SELECT Carrier,
   toYear(FlightDate) AS Year,
   count() AS TotalFlights,
   sum(Cancelled) AS TotalCancelled
FROM ontime
GROUP BY Carrier, Year
```

Every thing else <u>must be</u> an aggregate

Dimensions must be in GROUP BY

Another example: find "Top N"with ORDER BY and LIMIT

```
SELECT o.Origin,
  any(o.OriginCityName) AS OriginCityName,
  any(o.OriginState) AS OriginState,
  COUNT() as Flights
FROM ontime o
GROUP BY Origin
ORDER BY Flights DESC
LIMIT 10
```

Dimension

Correlated values

Measure

Unique aggregates in ClickHouse, example #1

-If Combinators

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

Aggregates from multiple entities in a single scan!

Unique aggregates in ClickHouse, example #2

```
SELECT
  sensor_id,
  max(time) AS last_time,
  argMaxState(message, time)
    AS last_message
FROM readings_multi rm
WHERE msg_type = 'err'
AND sensor_id = 236
GROUP BY sensor_id
```

Find the max time value

Show the message in the same row as the max time

How might you do #2 in "another" analytic database?

```
SELECT message
FROM readings_multi
WHERE (msg_type, sensor_id, time) IN
   (SELECT msg_type, sensor_id, max(time)
    FROM readings_multi
    WHERE msg_type = 'err'
      AND sensor_id = 236
    GROUP BY msg_type, sensor_id)
```

Matching values

Join

Subselect

Can be expensive in large datasets

JOIN combines data between tables

```
SELECT o.Dest,
  any(a.Name) AS AirportName,
  count(Dest) AS Flights
FROM ontime o
LEFT JOIN airports a ON a.IATA = toString(o.Dest)
GROUP BY Dest
ORDER BY Flights
DESC LIMIT 10
```

Join type

Include ontime data even if we can't find the airport name

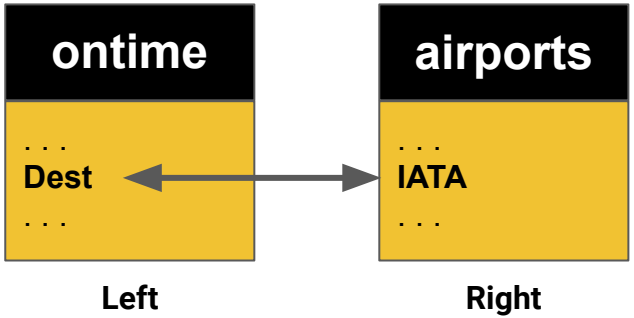
Join condition

Let's look more closely at joins

```
SELECT . . . FROM ontime o JOIN airports a ON a.IATA = toString(o.Dest)
```

LEFT [OUTER] JOIN

All ontime rows, plus matching airport rows



FULL [OUTER] JOIN

Matching rows, plus non-matching rows from both sides

RIGHT [OUTER] JOIN

All airport rows, plus matching ontime rows

[INNER] JOIN

Only rows that match on both sides

CROSS JOIN

Matches all rows with each other. Aka "cartesian join"

You can check this yourself with sample tables

Table Left	
id	value
1	Left
2	Left

Table Right	
id	value
2	Right
3	Right

[INNER] JOIN

```
Id value id value
 2 Left   2 Right
```

```
SELECT l.id,  l.value,  r.id,  r.value
FROM left AS l INNER JOIN right AS r ON l.id = r.id
```

```
Id value id value
 1 Left   0 -
 2 Left   2 Right
```

LEFT [OUTER] JOIN

```
Id value id value
 1 Left   0 -
 2 Left   2 Right
 0 -      3 Right
```

FULL [OUTER] JOIN

```
Id value id value
 2 Left   2 Right
 0 -      3 Right
```

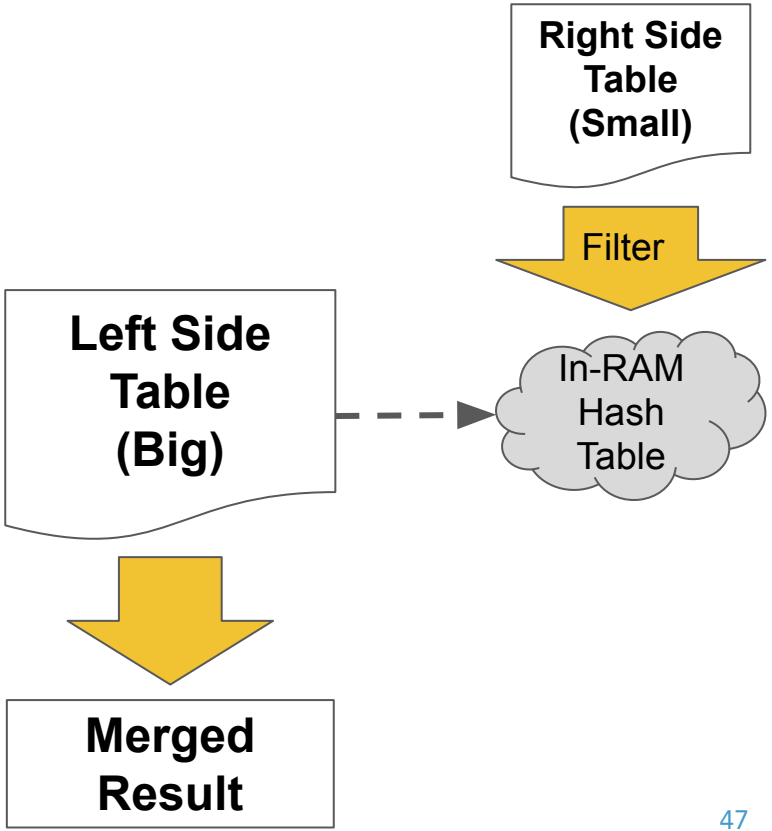
RIGHT [OUTER] JOIN

How does ClickHouse process a query with a join?

Load the right side table

Scan left side table in parallel, computing aggregates and adding joined data

Merge and sort results



Top tips for querying data in ClickHouse

- Use time functions to pull out data grouped by time intervals
- Use aggregates to summarize data inside ClickHouse
  - Use -If combinators and others to avoid multiple scans
- The big table always goes on the <u>LEFT</u> side of the join
  - ClickHouse cannot decide the join order [yet] for you

ClickHouse Performance Self-Defense

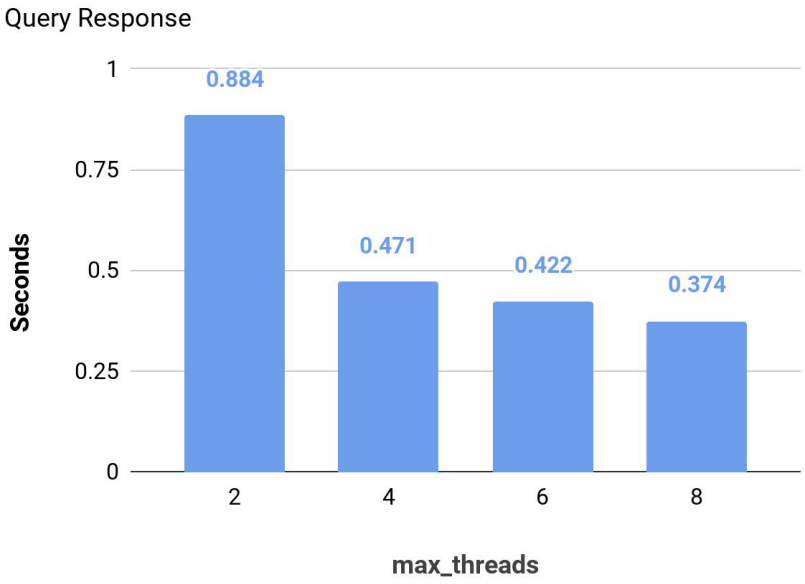
ClickHouse performance in four easy steps

- Add more CPU
- Limit columns and rows in queries
- Fix table design
- Store data more efficiently on disk (or SSD)

Add more compute using max_threads property

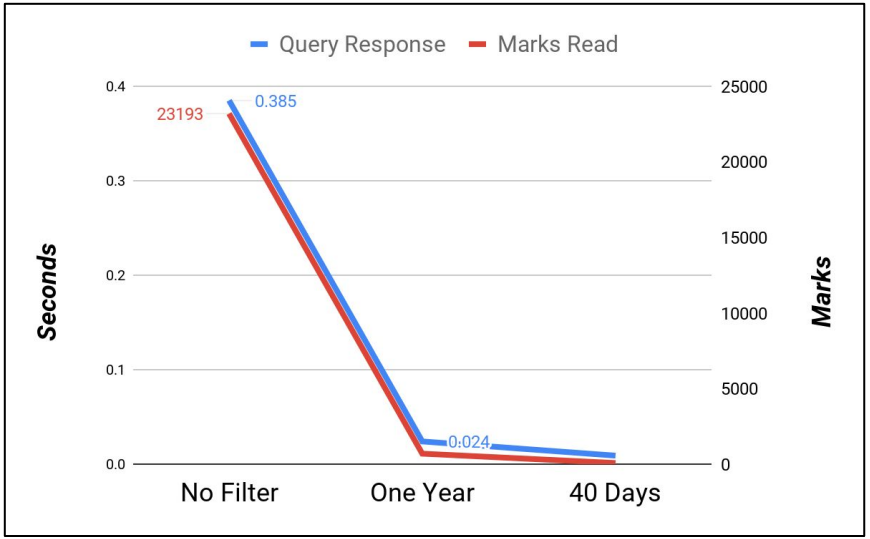
```
SELECT toYear(FlightDate) year,
  sum(Cancelled)/count(*) cancelled,
  sum(DepDel15)/count(*) delayed_15
FROM airline.ontime
GROUP BY year ORDER BY year LIMIT 10

SET max_threads = 2

SET max_threads = 4

. . .
```

max_threads defaults to half the number of CPU cores



Read fewer rows to get better performance

```
SELECT
  FlightDate,
  count(*) AS total_flights,
  sum(Cancelled) / count(*) AS cancelled,
  sum(DepDel15) / count(*) AS delayed_15
FROM airline.ontime
WHERE (FlightDate >= toDate('2016-01-01'))
  AND (FlightDate <= toDate('2016-02-10'))
GROUP BY FlightDate
```




```
(Log messages)
Selected 2 parts by date,
2 parts by key,
73 marks to read from 2 ranges
```

Understanding what's going on in MergeTree

/var/lib/clickhouse/data/airline/ontime

20170701_20170731_355_355_2/

(FlightDate, Carrier...) **ActualElapsedTime** **Airline** **AirlineID...**

primary.idx

2017-07-01	AA
2017-07-01	EV
2017-07-01	**UA**
2017-07-02	AA
...	

.mrk .bin .mrk .bin .mrk .bin

Granule

Mark

Compressed Block

Ensure table design is optimal for queries

```
CREATE TABLE ontime
(
   `Year` UInt16,
   `Quarter` UInt8, . . .
)
ENGINE = MergeTree
PARTITION BY Year
PRIMARY KEY (Carrier, FlightDate)
ORDER BY (Carrier, FlightDate, DepTime)
```

Results in ~125 parts for this dataset, avg 107MB per part

Use columns you filter on

Order columns by increasing cardinality* left to right

*Cardinality = How many unique values in column

Finding out column compression level

```
SELECT
  formatReadableSize(sum(data_compressed_bytes)) AS tc,
  formatReadableSize(sum(data_uncompressed_bytes)) AS tu,
  sum(data_compressed_bytes) / sum(data_uncompressed_bytes) AS ratio
FROM system.columns
WHERE database = currentDatabase()
  AND (table = 'ontime')
  AND (name = 'TailNum')
```

tc	tu	ratio
19.33 MiB	37.80 MiB	0.5112778354359176

Make a column smaller with compression

Compression reduces a string of bits to a different and hopefully smaller string of bits

TailNum
N3HYAA
N377AA
N377AA
N922AA
N862AA
N3EEAA
N536AA
N665AA
N862AA
N936AA
N4YBAA

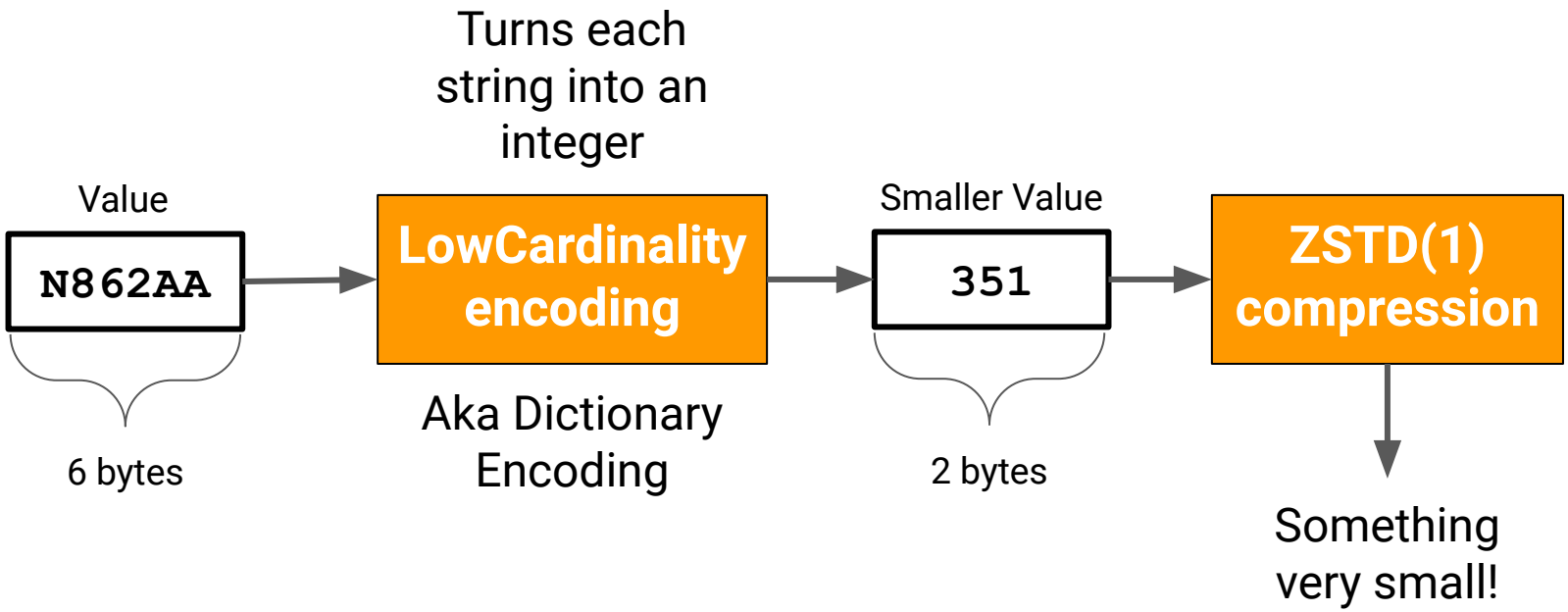
LZ4 Compression

Fast to compress and decompress. Default for ClickHouse

ZSTD Compression

Slower to compress but generally smaller than LZ4. You have to ask for it

Codecs reduce data before compression



Overview of encodings

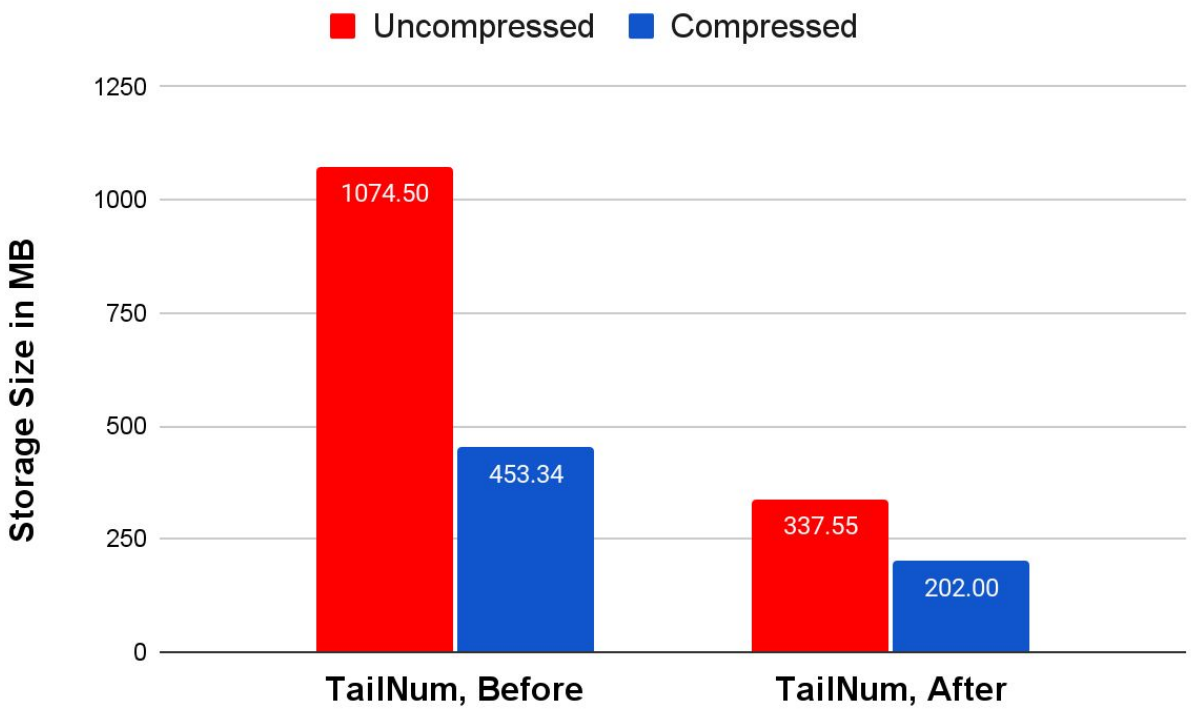
Name	Best for
LowCardinality	Strings with fewer than 10K values
Delta	Time series
Double Delta	Increasing counters
Gorilla	Gauge data (bounces around mean)
T64	Integers other than random hashes

Codec compression may vary between LZ4 and ZSTD

How to add codecs and change compression

```
ALTER TABLE ontime
  MODIFY COLUMN TailNum LowCardinality(String)
CODEC(ZSTD(1))

ALTER TABLE ontime
  MODIFY COLUMN Year CODEC(DoubleDelta, ZSTD(1))

-- Force ClickHouse to change data
ALTER TABLE ontime UPDATE TailNum = TailNum, Year = Year
WHERE 1
```

Effect of codec and compression changes



Getting started on applications

Where is the software? (All open source!)

Event streaming

- Apache Kafka
- Apache Pulsar
- Vectorized Redpanda

ELT

- Apache Airflow
- Rudderstack

Rendering/Display

- Apache Superset
- Cube.js
- Grafana

Client Libraries

- C++ - ClickHouse CPP
- Golang - ClickHouse Go
- Java - ClickHouse JDBC
- Javascript/Node.js - Apla
- ODBC - ODBC Driver for ClickHouse
- Python - ClickHouse Driver, ClickHouse SQLAlchemy

More client library links HERE

Kubernetes

- Altinity Operator for ClickHouse

Where is the documentation?

ClickHouse official docs – https://clickhouse.com/docs/

Altinity Blog – https://altinity.com/blog/

Altinity Youtube Channel – https://www.youtube.com/channel/UCE3Y2lDKl_ZfjaCrh62onYA

Altinity Knowledge Base – https://kb.altinity.com/

Meetups, other blogs, and external resources. Use your powers of Search!

Where can I get help?

Telegram - ClickHouse Channel

Slack

- ClickHouse Public Workspace - clickhousedb.slack.com
- Altinity Public Workspace - altinitydbworkspace.slack.com

Education - Altinity ClickHouse Training

Support - Altinity offers support for ClickHouse in all environments

Thank you and good luck!

Website: https://altinity.com
Email: info@altinity.com
Slack: altinitydbworkspace.slack.com

Altinity.Cloud

Altinity Support

Altinity Stable Builds

We're hiring!