Altinity

TRICKS EVERY CLICKHOUSE DESIGNER SHOULD KNOW

Robert Hodges
ClickHouse SFO Meetup
August 2019

Introduction to presenter

Robert Hodges - Altinity CEO

30+ years on DBMS plus virtualization and security.

ClickHouse is DBMS #20

Altinity

www.altinity.com

Leading software and services provider for ClickHouse

Major committer and community sponsor in US and Western Europe

Introduction to ClickHouse

Understands SQL

Runs on bare metal to cloud

Shared nothing architecture

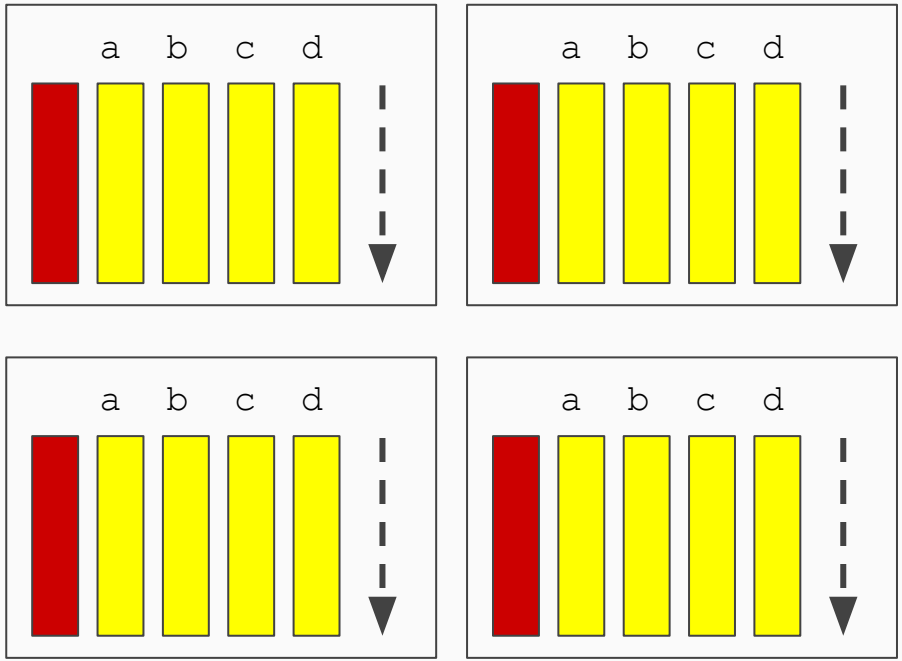
Uses column storage

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)

And it's really fast!



Use encodings to reduce data size

Applying encodings to table columns

```
CREATE TABLE test_codecs ( a String,
    a_lc LowCardinality(String) DEFAULT a,
    b UInt32,
    b_delta UInt32 DEFAULT b Codec(Delta),
    b_delta_lz4 UInt32 DEFAULT b Codec(Delta, LZ4),
    b_dd UInt32 DEFAULT b Codec(DoubleDelta),
    b_dd_lz4 UInt32 DEFAULT b Codec(DoubleDelta, LZ4)
)
Engine = MergeTree
PARTITION BY tuple() ORDER BY tuple();
```

Values with dictionary encoding

Differences between values

Differences between change of value

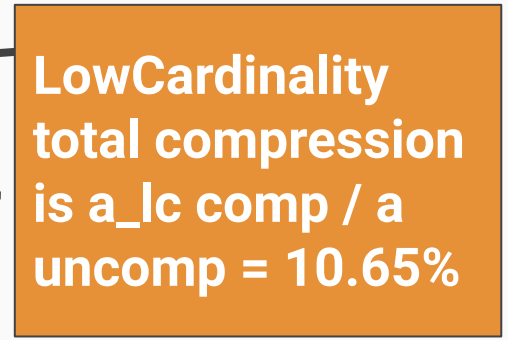
Load lots of data

```
INSERT INTO test_codecs (a, b)
  SELECT
    concat('a string prefix',
           toString(rand() % 1000)),
    now() + (number * 10)
  FROM system.numbers
  LIMIT 100000000
  SETTINGS max_block_size=1000000
```

Check data sizes

```
SELECT name, sum(data_compressed_bytes) comp,
  sum(data_uncompressed_bytes) uncomp,
  round(comp / uncomp * 100.0, 2) AS percent
FROM system.columns WHERE table = 'test_codecs'
GROUP BY name ORDER BY name
```

name	comp	uncomp	percent
a	393712480	1888998884	20.84
a_lc	201150993	200431356	100.36
b	401727287	400000000	100.43
b_delta	400164862	400000000	100.04
b_delta_lz4	1971602	400000000	0.49
b_dd	12738212	400000000	3.18
b_dd_lz4	476375	400000000	0.12



But wait, there's more! Encodings help query speed

```
SELECT a AS a, count(*) AS c FROM test_codecs
GROUP BY a ORDER BY c ASC LIMIT 10
. . .
10 rows in set. Elapsed: 0.681 sec. Processed 100.00 million
rows, 2.69 GB (146.81 million rows/s., 3.95 GB/s.)
```


```
SELECT a_lc AS a, count(*) AS c FROM test_codecs
GROUP BY a ORDER BY c ASC LIMIT 10
. . .
10 rows in set. Elapsed: 0.148 sec. Processed 100.00 million
rows, 241.16 MB (675.55 million rows/s., 1.63 GB/s.)
```

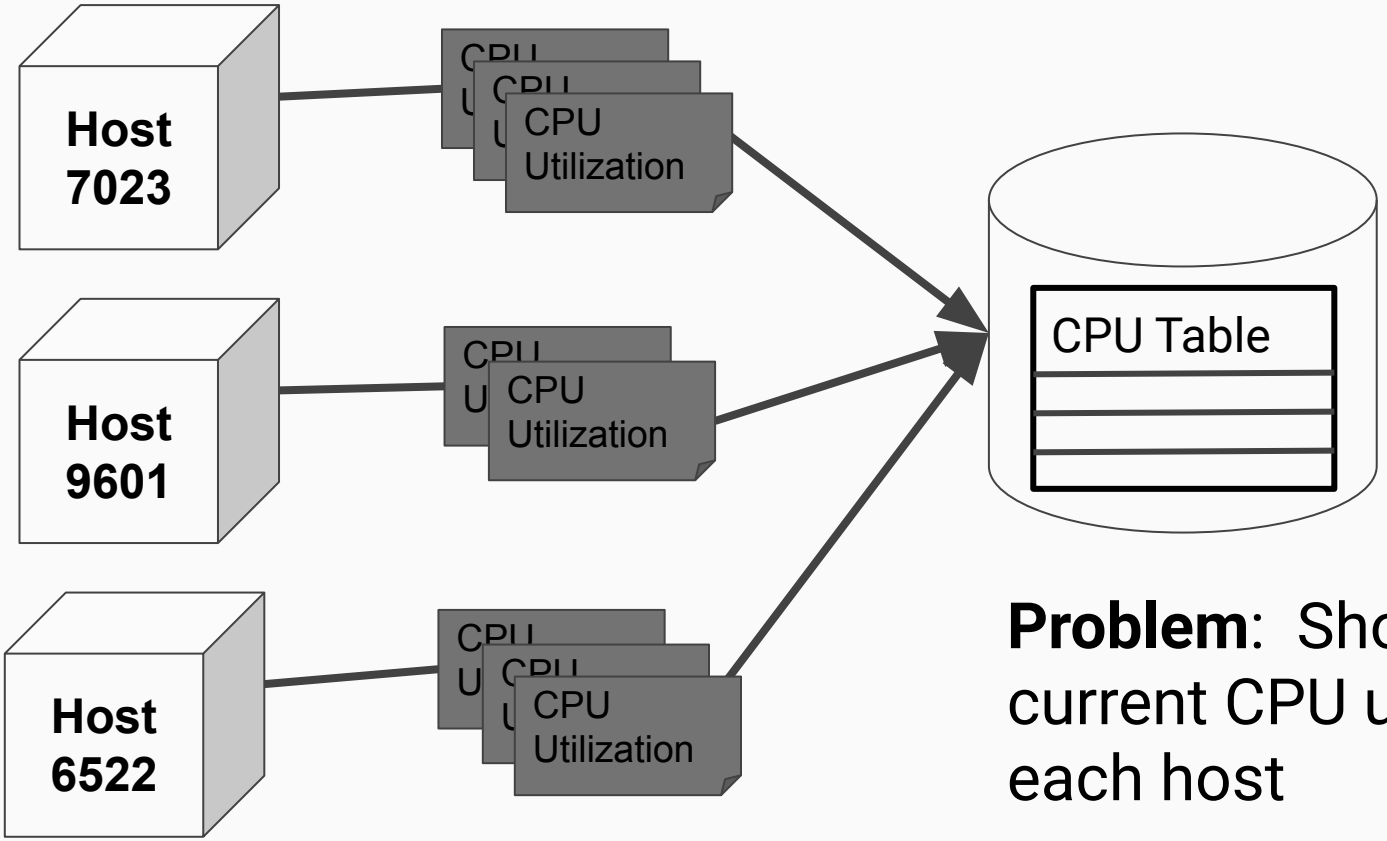
Quick Comparison of Encodings

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

Compression may vary across ZSTD and LZ4

Use materialized views to find last point data

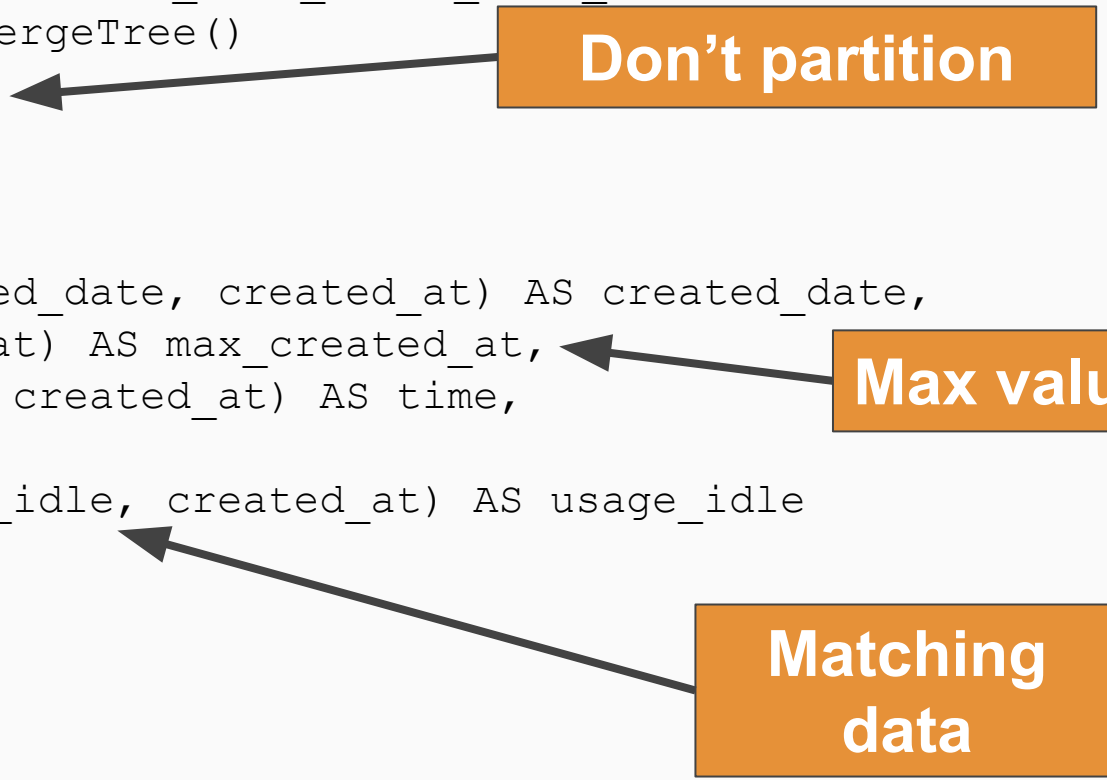
Last point problems are common in time series



Problem: Show the current CPU utilization for each host

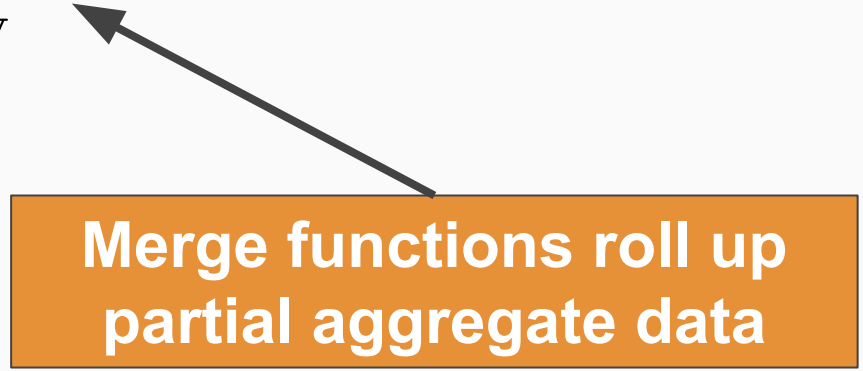
argMaxState links columns with aggregates

```
CREATE MATERIALIZED VIEW cpu_last_point_idle_mv
ENGINE = AggregatingMergeTree()
PARTITION BY tuple()
ORDER BY tags_id
POPULATE
AS SELECT
    argMaxState(created_date, created_at) AS created_date,
    maxState(created_at) AS max_created_at,
    argMaxState(time, created_at) AS time,
    tags_id,
    argMaxState(usage_idle, created_at) AS usage_idle
FROM cpu
GROUP BY tags_id
```



Let's hide the merge details with a view

```
CREATE VIEW cpu_last_point_idle_v AS
SELECT
    argMaxMerge(created_date) AS created_date,
    maxMerge(max_created_at) AS created_at,
    argMaxMerge(time) AS time,
    tags_id,
    argMaxMerge(usage_idle) AS usage_idle
FROM cpu_last_point_idle_mv
GROUP BY tags_id
```



...Select from the covering view

```
SELECT
  tags_id,
  100 - usage_idle usage
FROM cpu_last_point_idle_v
ORDER BY usage DESC, tags_id ASC
LIMIT 10

...

10 rows in set. Elapsed: 0.005 sec. Processed 14.00
thousand rows, 391.65 KB (2.97 million rows/s., 82.97
MB/s.)
```

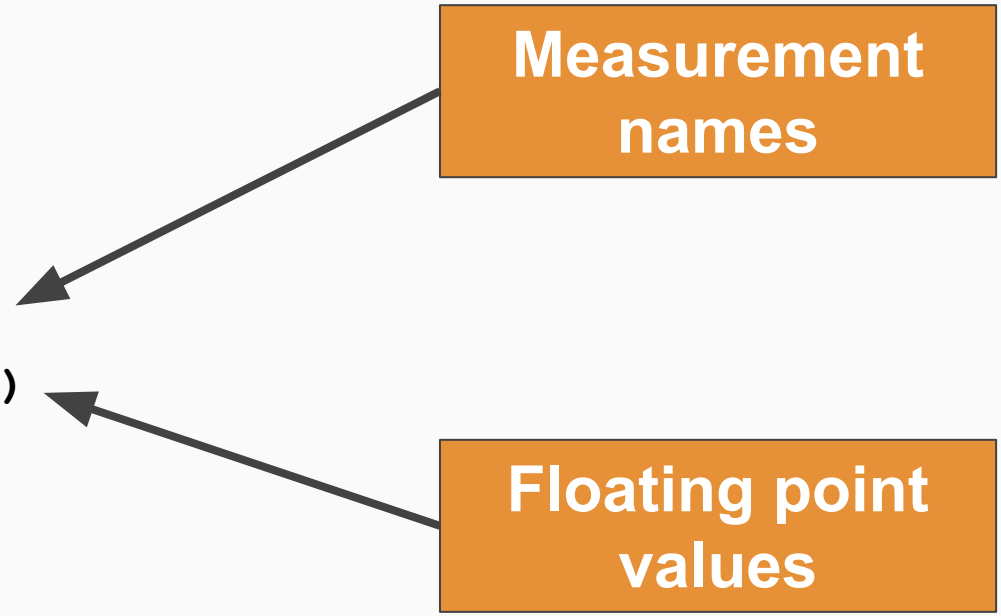
Use arrays to store key-value pairs

SQL tables typically have a tabular form

```
CREATE TABLE cpu (
  created_date Date DEFAULT today(),
  created_at DateTime DEFAULT now(),
  time String,
  tags_id UInt32,
  usage_user Float64,
  usage_system Float64,
  . . .
  additional_tags String DEFAULT '')
ENGINE = MergeTree()
PARTITION BY created_date
ORDER BY (tags_id, created_at)
```

Paired arrays allow flexible values

```
CREATE TABLE cpu_dynamic (
  created_date Date,
  created_at DateTime,
  time String,
  tags_id UInt32,
  metrics_name Array(String),
  metrics_value Array(Float64)
)
ENGINE = MergeTree()
PARTITION BY created_date
ORDER BY (tags_id, created_at)
```



Insert some values from JSON

```
clickhouse-client --database default \
--query="INSERT INTO cpu_dynamic FORMAT JSONEachRow"
<<DATA
{"created_date":"2016-01-03",
 "created_at":"2016-01-03 00:00:00",
 "time":"2016-01-03 00:00:00 +0000",
 "tags_id":6220,
 "metrics_name":["usage_user","usage_system",
   "usage_idle","usage_nice"],
 "metrics_value":[35,47,77,21]}
. . .
DATA
```

Now each row can have different key/value pairs

```
SELECT time, tags_id, metrics_name,metrics_value
FROM cpu_dynamic

Row 1:
──────
time:         2016-01-03 00:00:00 +0000
tags_id:      6220
metrics_name:
['usage_user','usage_system','usage_idle','usage_nice']
metrics_value: [35,47,77,21]
. . .
```

You can pivot back to a tabular form with ARRAY JOIN

```
SELECT created_at, tags_id, name, value
FROM cpu_dynamic
ARRAY JOIN metrics_name AS name, metrics_value AS value
```

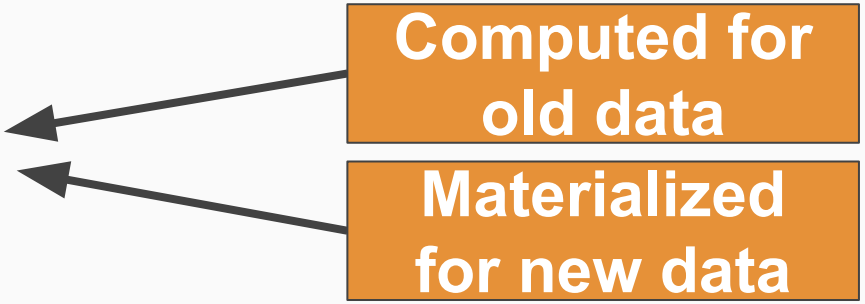
Correlated arrays

created_at	tags_id	name	value
2016-01-03 00:00:00	6220	usage_user	35
2016-01-03 00:00:00	6220	usage_system	47
2016-01-03 00:00:00	6220	usage_idle	77
2016-01-03 00:00:00	6220	usage_nice	21
2016-01-03 00:00:10	6220	usage_nice	21
2016-01-03 00:00:10	6220	usage_iowait	84
2016-01-03 00:00:10	6220	usage_irq	22

Use materialized columns to pre-compute values

You can create new columns using MATERIALIZED

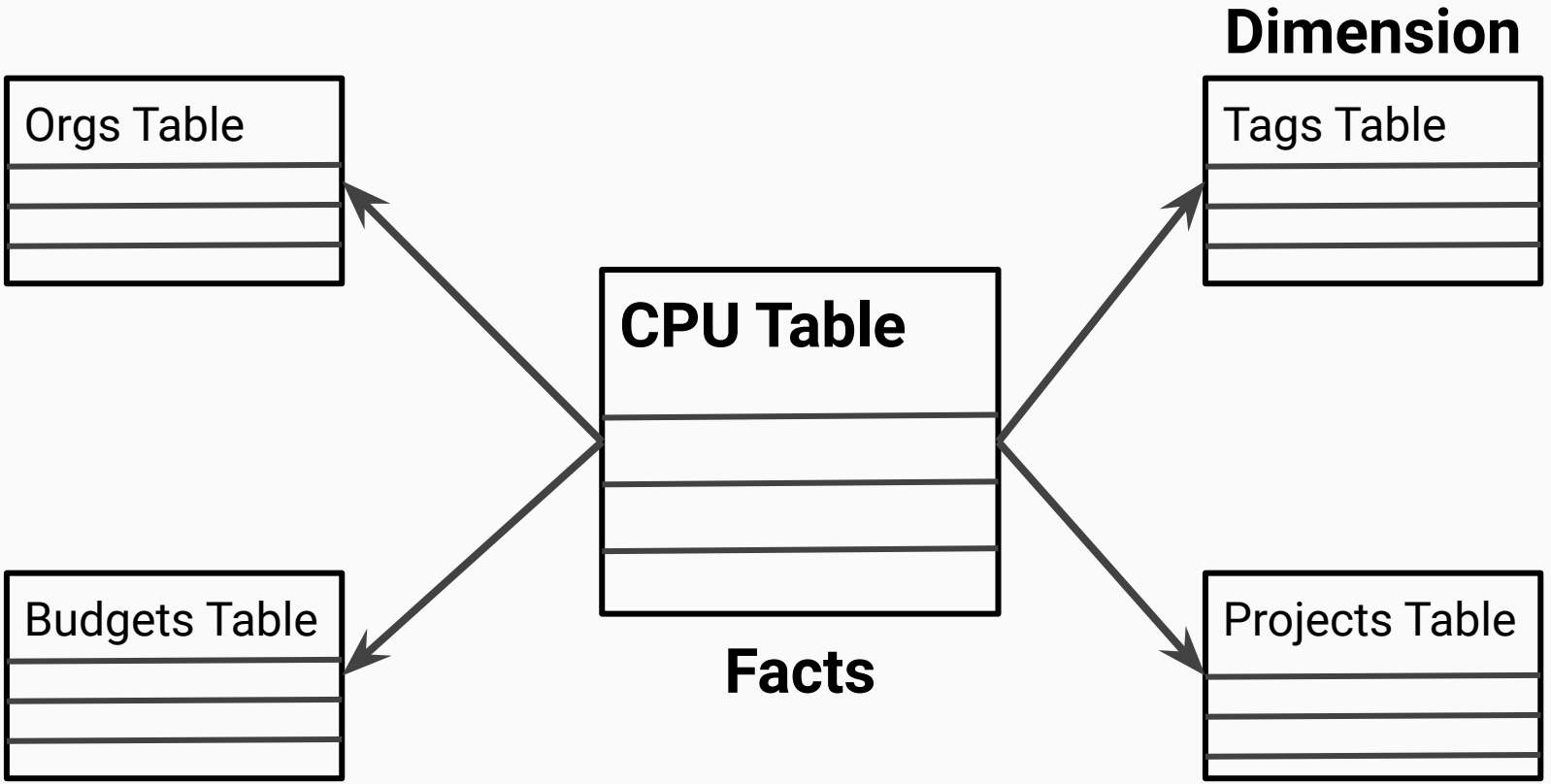
```
ALTER TABLE cpu_dynamic ADD COLUMN usage_user MATERIALIZED
  metrics_value[indexOf(metrics_name, 'usage_user')]
    AFTER tags_id

SELECT time, tags_id, usage_user
FROM cpu_dynamic
```




```
┌─time────────────────────────────┬─tags_id─┬─usage_user─┐
│ 2016-01-03 00:00:00 +0000 │    6220 │         35 │
│ 2016-01-03 00:00:10 +0000 │    6220 │          0 │
└─────────────────────────────┴─────────┴────────────┘
```

Use dictionaries instead of joins for dimensions

Star schemas are common in data warehouse



Joining facts and dimensions is not always convenient

Dimension values are often mutable

```
SELECT tags.rack rack, avg(100 - cpu.usage_idle) usage
FROM cpu
INNER JOIN tags AS t ON cpu.tags_id = t.id
GROUP BY rack
ORDER BY usage DESC
LIMIT 10
```

Need a join for every dimension

Matching data

Dictionaries are an alternative to joins

/etc/clickhouse-server/tags_dictionary.xml:

```
<yandex><dictionary>
  <name>tags</name>
  <source><clickhouse>
    <host>localhost</host><port>9000</port><user>default</user>
    <password></password><db>tsbs</db><table>tags</table>
  </clickhouse></source>
  <layout> <hashed/> </layout>
  <structure>
    <id> <name>id</name> </id>
    <attribute>
      <name>hostname</name><type>String</type>
      <null_value></null_value>
    </attribute> . . .
```

Now we can avoid costly joins

Hash table stored in memory

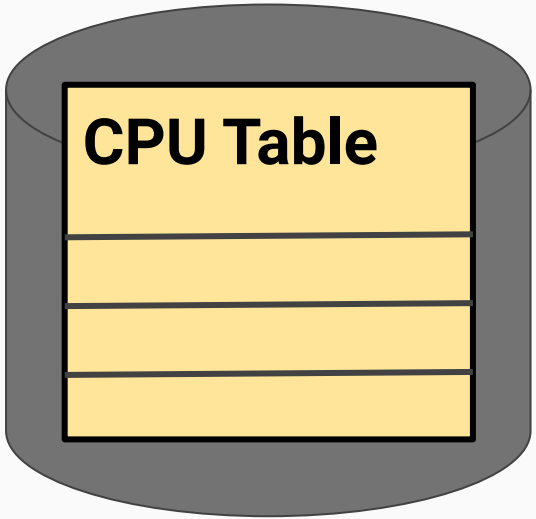
```
SELECT
  dictGetString('tags', 'rack', toUInt64(cpu.tags_id)) rack,
  avg(100 - cpu.usage_idle) usage
FROM cpu
GROUP BY rack
ORDER BY usage DESC
LIMIT 10
```

Dictionary key must be UInt64

Matching data

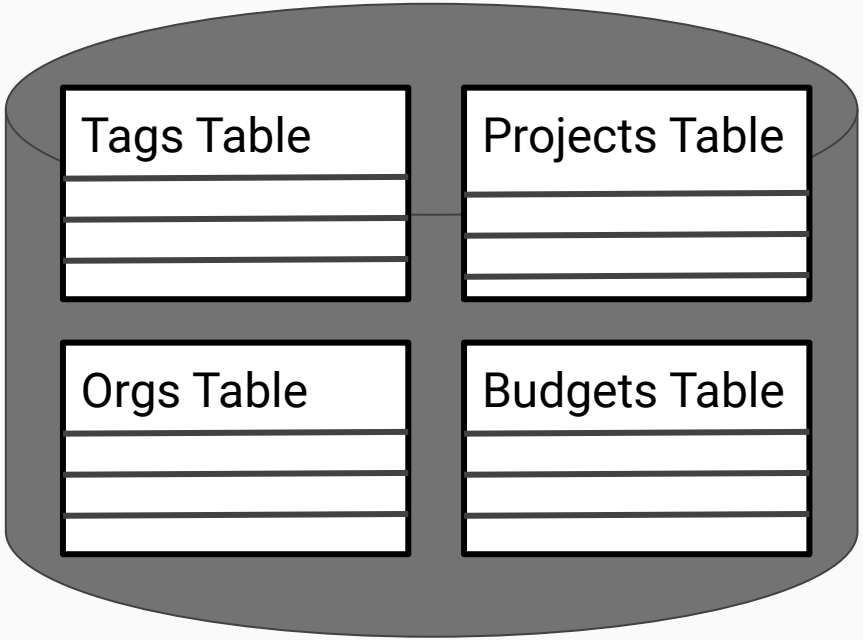
External dictionaries live in another location

Large, immutable facts



ClickHouse

Small, mutable dimensions



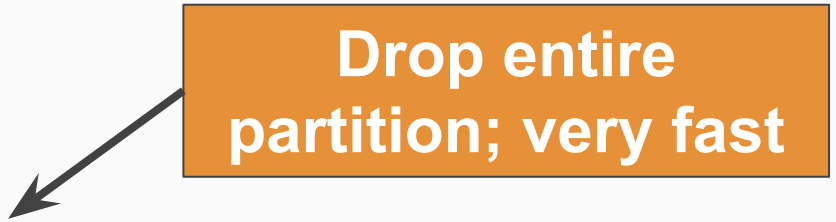
MySQL or PostgreSQL

Use TTLs to delete obsolete data

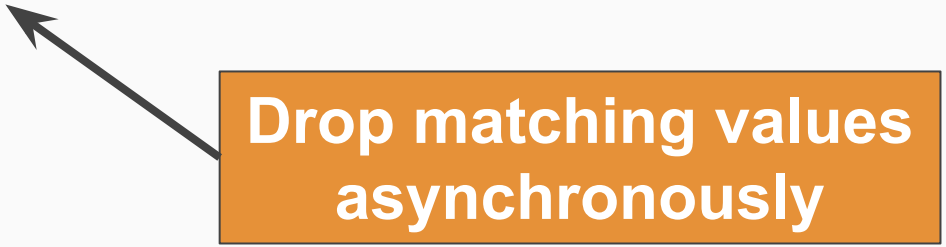
ClickHouse is not optimized for deletes

```
CREATE TABLE traffic (
  datetime DateTime,
  date Date,
  request_id UInt64,
  cust_id UInt32,
  sku UInt32
) ENGINE = MergeTree
PARTITION BY toYYYYMM(datetime)
ORDER BY (cust_id, date)
```

Until 2019 there were two options




```
ALTER TABLE traffic DROP PARTITION 201801

ALTER TABLE traffic DELETE WHERE
  datetime < toDateTime('2018-02-01 00:00:00')
```



You can delete automatically with a TTL

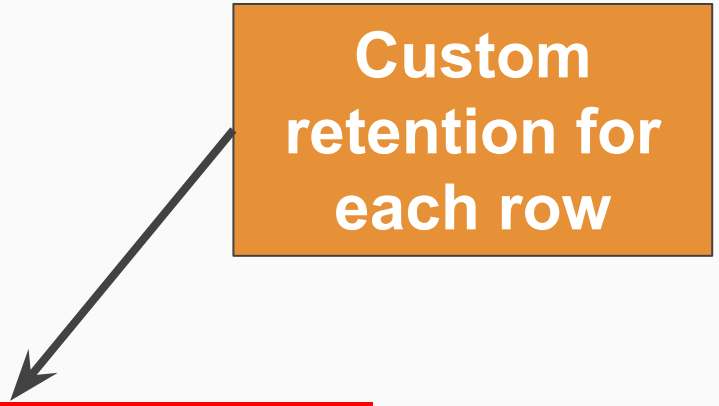
```
CREATE TABLE traffic (
  datetime DateTime,
  date Date,
  request_id UInt64,
  cust_id UInt32,
  sku UInt32
) ENGINE = MergeTree
PARTITION BY toYYYYMM(datetime)
ORDER BY (cust_id, date)
TTL datetime + INTERVAL 90 DAY
```

TTL application set by merge_with_ttl_timeout in merge_tree_settings

Drop rows after 90 days

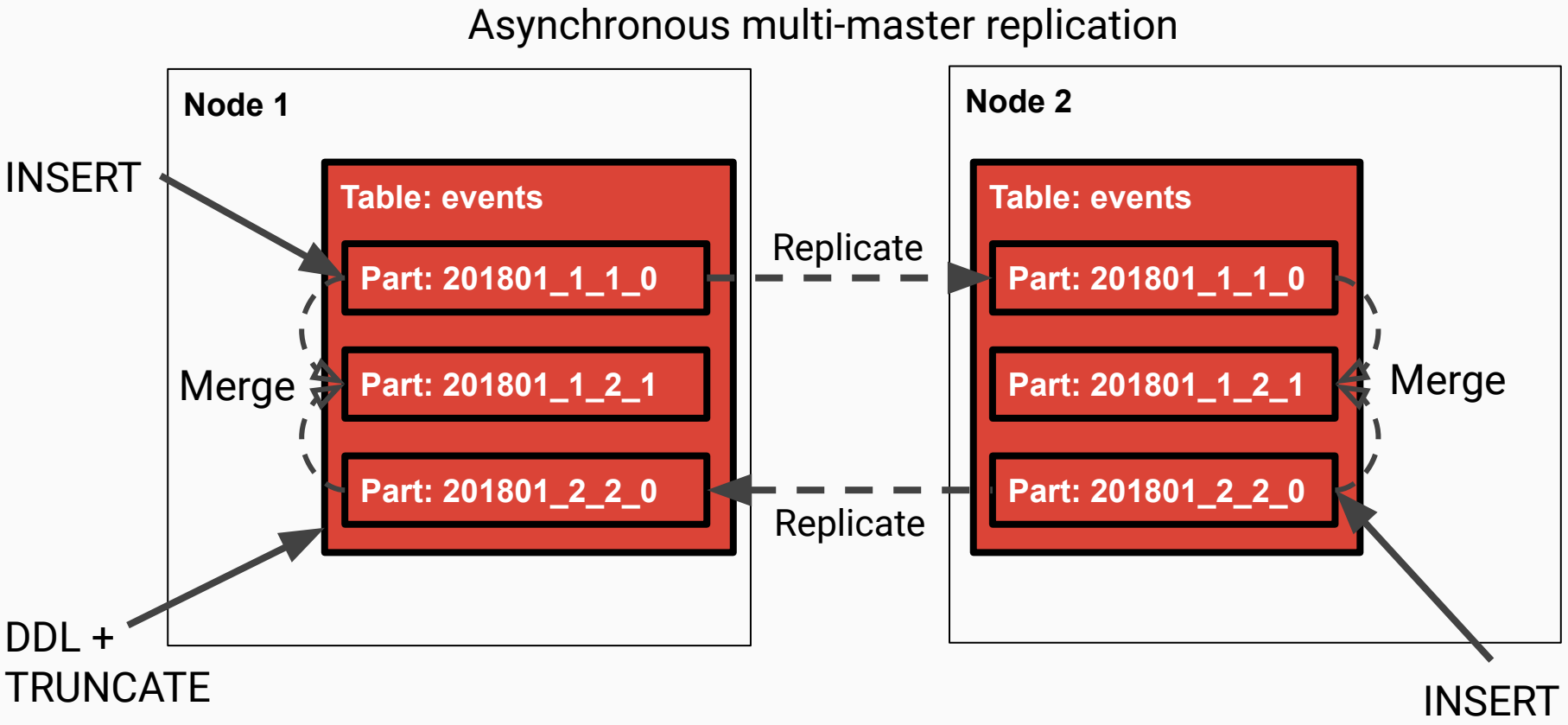
TTL value can be variable based on data

```
CREATE TABLE traffic_with_ttl_variable (
  datetime DateTime,
  date Date,
  retention_days UInt16,
  request_id UInt64,
  cust_id UInt32,
  sku UInt32
) ENGINE = MergeTree
PARTITION BY toYYYYMM(datetime)
ORDER BY (cust_id, date)
TTL date + INTERVAL (retention_days * 2) DAY
```

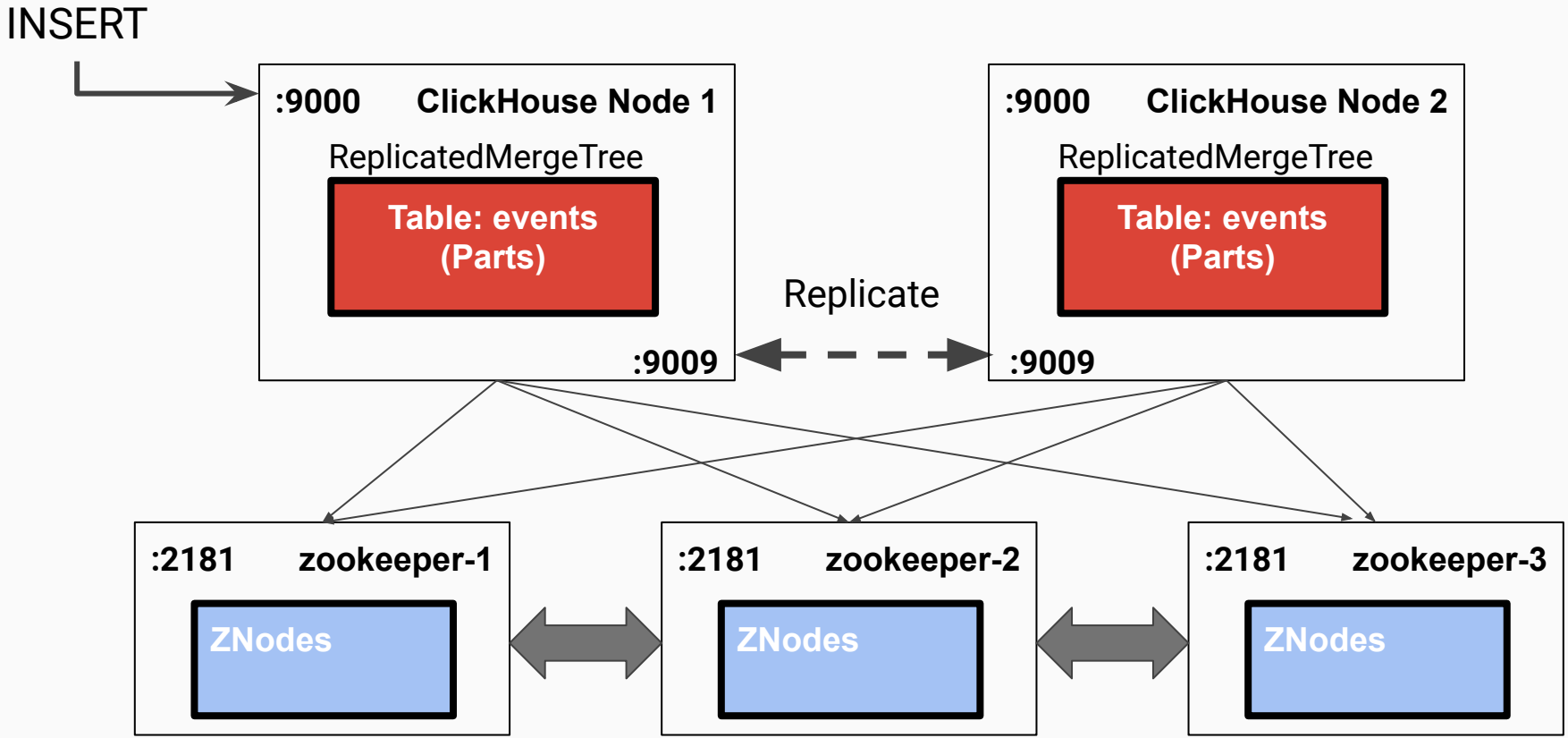


Use Replication instead of backups

Replication works on per-table basis



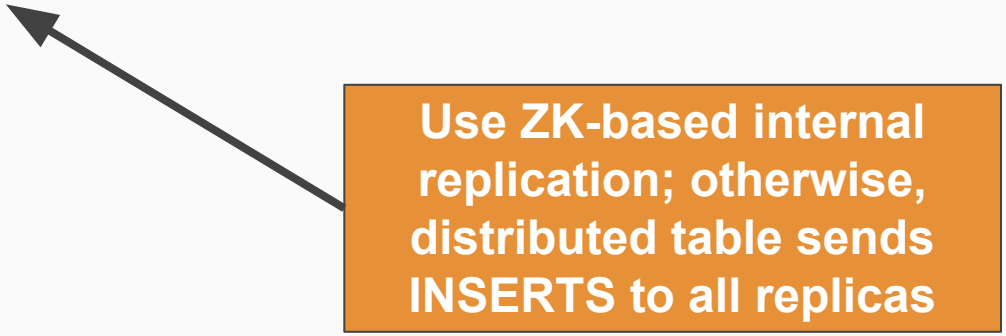
Clickhouse and Zookeeper implementation



Clusters define sharding and replication layouts

/etc/clickhouse-server/config.d/remote_servers.xml:

```
<yandex>
  <remote_servers>
    <Replicated>
      <shard>
        <replica><host>10.0.0.71</host><port>9000</port></replica>
        <replica><host>10.0.0.72</host><port>9000</port></replica>
        <internal_replication>true</internal_replication>
      </shard>
    <Replicated>
  </remote_servers>
</yandex>
```



Macros enable consistent DDL over a cluster

/etc/clickhouse-server/config.d/macros.xml:

```
<yandex>
  <macros>
    <cluster>Replicated</cluster>
    <shard>01</shard>
    <replica>01</replica>
  </macros>
</yandex>
```

Zookeeper tag defines servers and task queue

/etc/clickhouse-server/config.d/zookeeper.xml:

```
<yandex>
  <zookeeper>
    <node><host>10.0.0.61</host><port>2181</port></node>
    <node><host>10.0.0.62</host><port>2181</port></node>
    <node><host>10.0.0.63</host><port>2181</port></node>
  </zookeeper>
  <distributed_ddl>
    <path>/clickhouse/ShardedAndReplicated/task_queue/ddl</path>
  </distributed_ddl>
</yandex>
```

Clickhouse restart required after Zookeeper config changes

Create tables!

```
CREATE TABLE events ON CLUSTER '{cluster}' (
  EventDate DateTime,
  CounterID UInt32,
 UserID UInt32)
ENGINE =
ReplicatedMergeTree('/clickhouse/{cluster}/test/tables/events/{
shard}', '{replica}')
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
```

Thank you!

We’re hiring!

Presenter:
rhodges@altinity.com

ClickHouse Operator:
https://github.com/Altinity/clickhouse-operator

ClickHouse:
https://github.com/yandex/ClickHouse

Altinity:
https://www.altinity.com