A close-up photograph of various mechanical components, including springs, bolts, and metal brackets, arranged in a complex, layered structure. The lighting is dramatic, highlighting the metallic textures and creating deep shadows. The text is overlaid on the left side of the image.

# Deep Dive on ClickHouse Sharding and Replication

Robert Hodges and Altinity Engineering  
22 September 2022

# 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

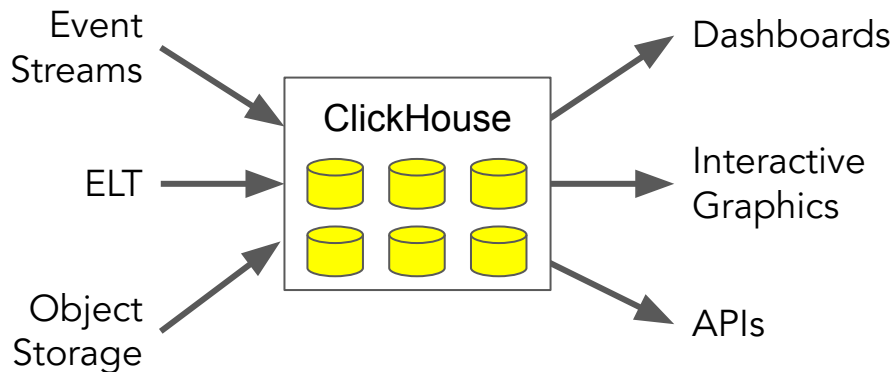
Shared nothing architecture

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

# Distributed data is deeper than it looks

“The  
Bolton  
Strid”

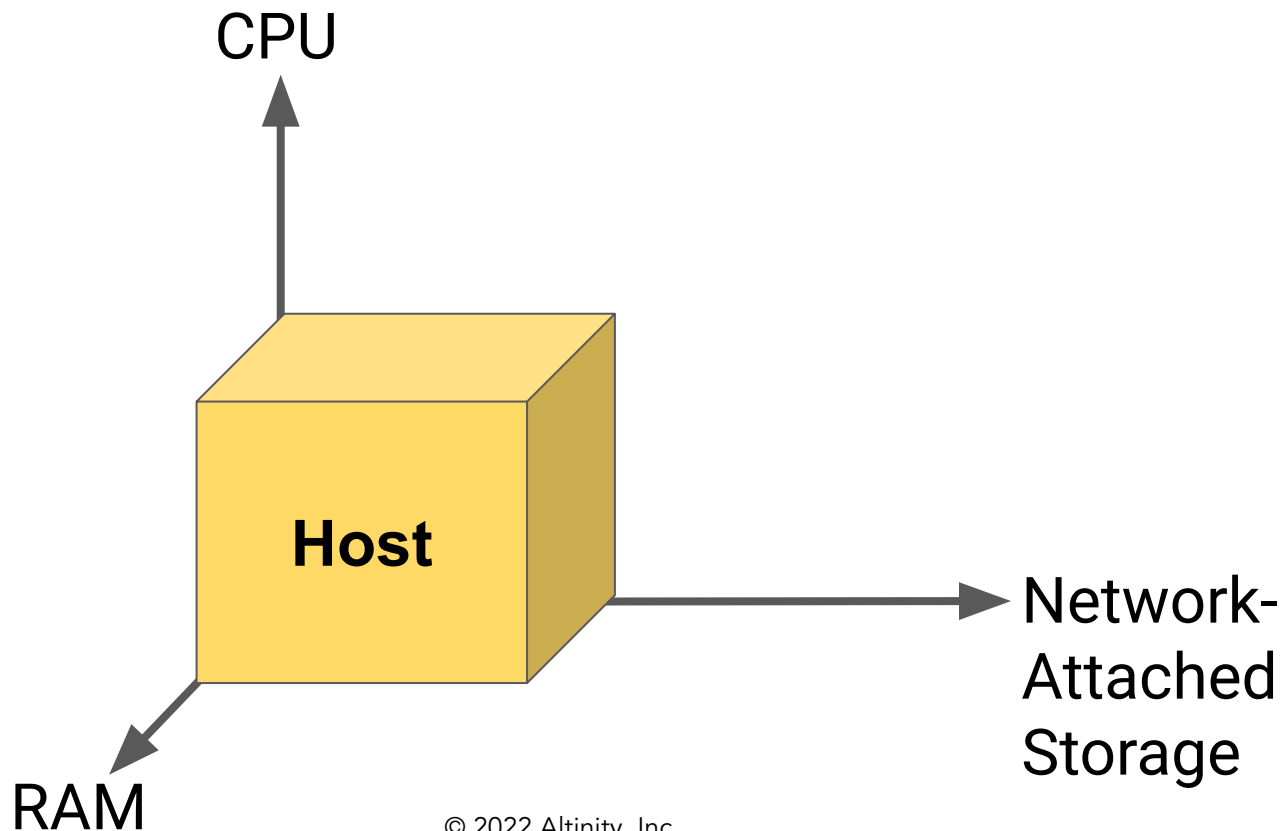


Width:  
2 meters

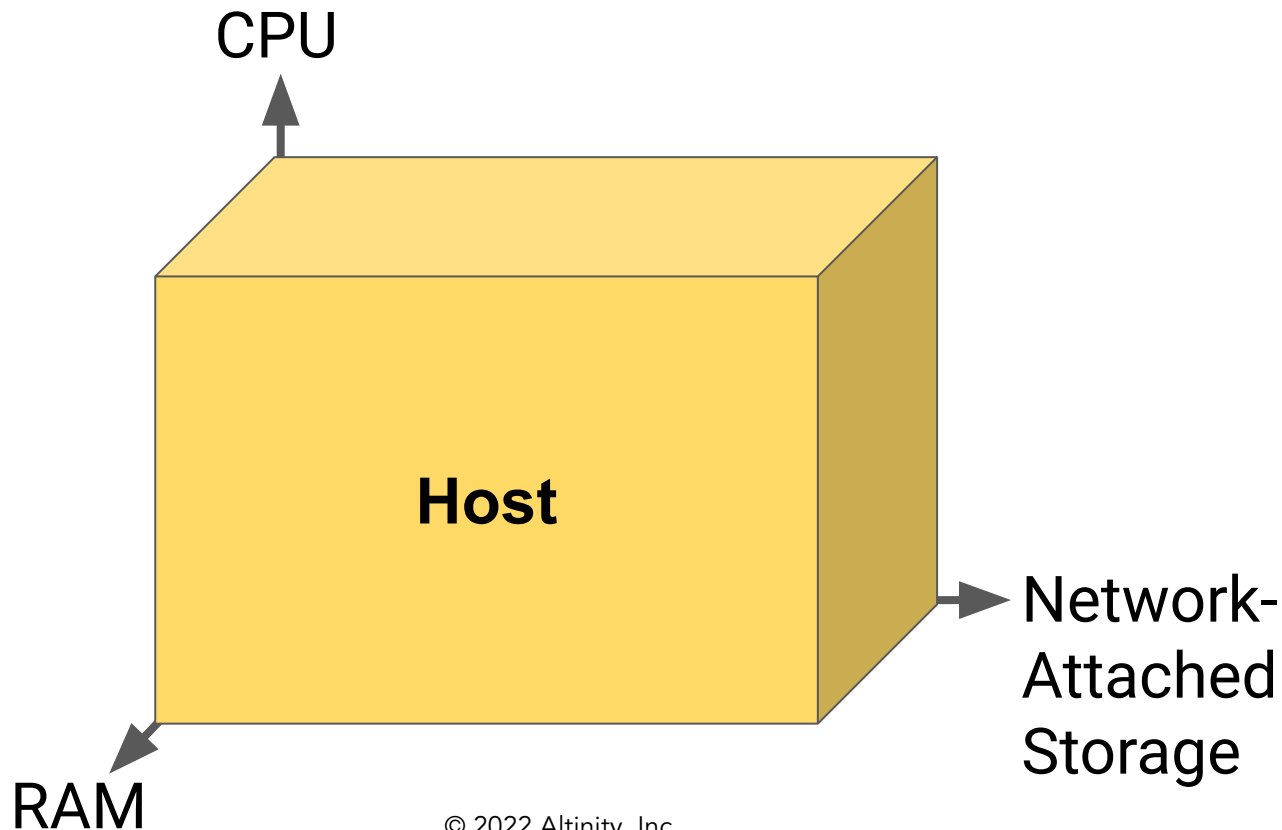
Depth:  
60 meters

# Introducing sharding and replication

# Clickhouse nodes can scale vertically



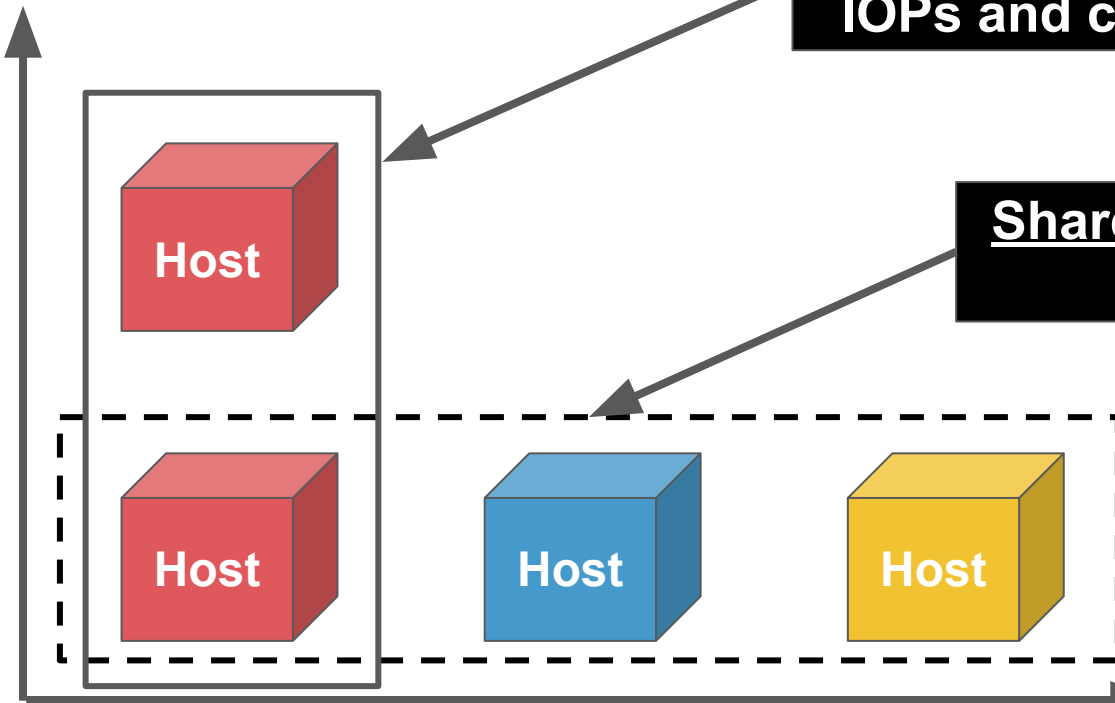
# Clickhouse nodes can scale vertically





# Clusters introduce horizontal scaling

Replicas

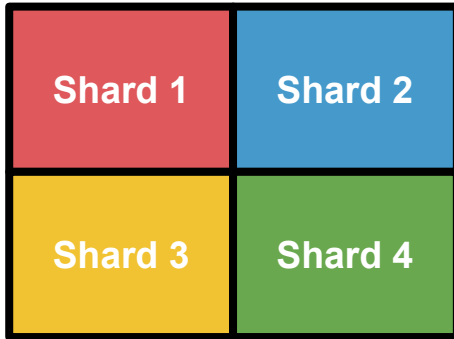


**Replicas improve read IOPs and concurrency**

**Shards add write IOPS**

# Different sharding and replication patterns

## All Sharded



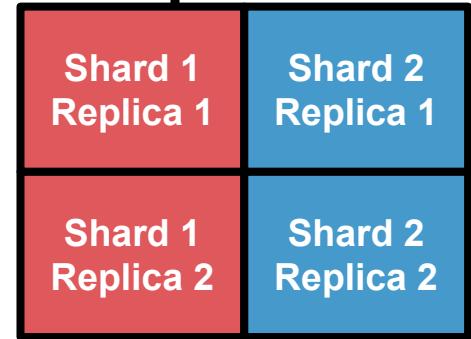
Data sharded 4 ways without replication

## All Replicated



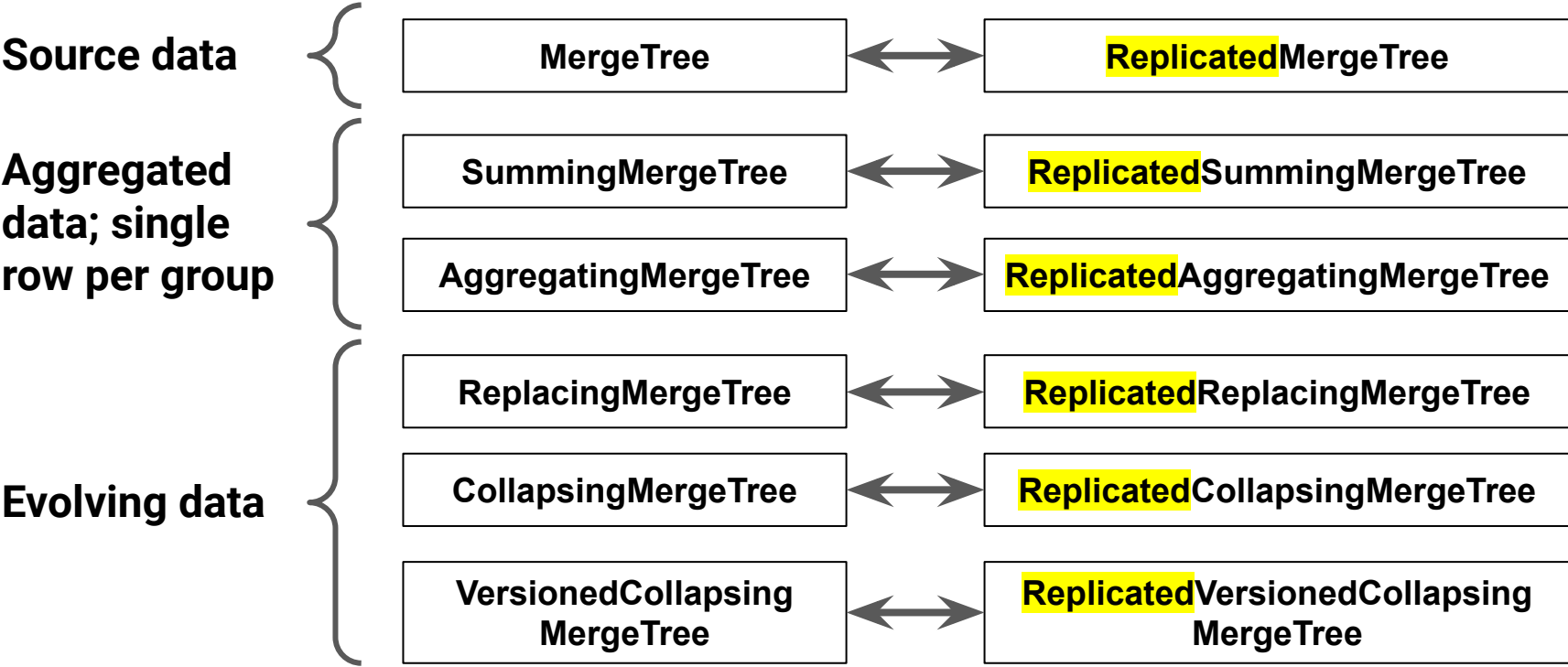
Data replicated 4 times without sharding

## Sharded and Replicated

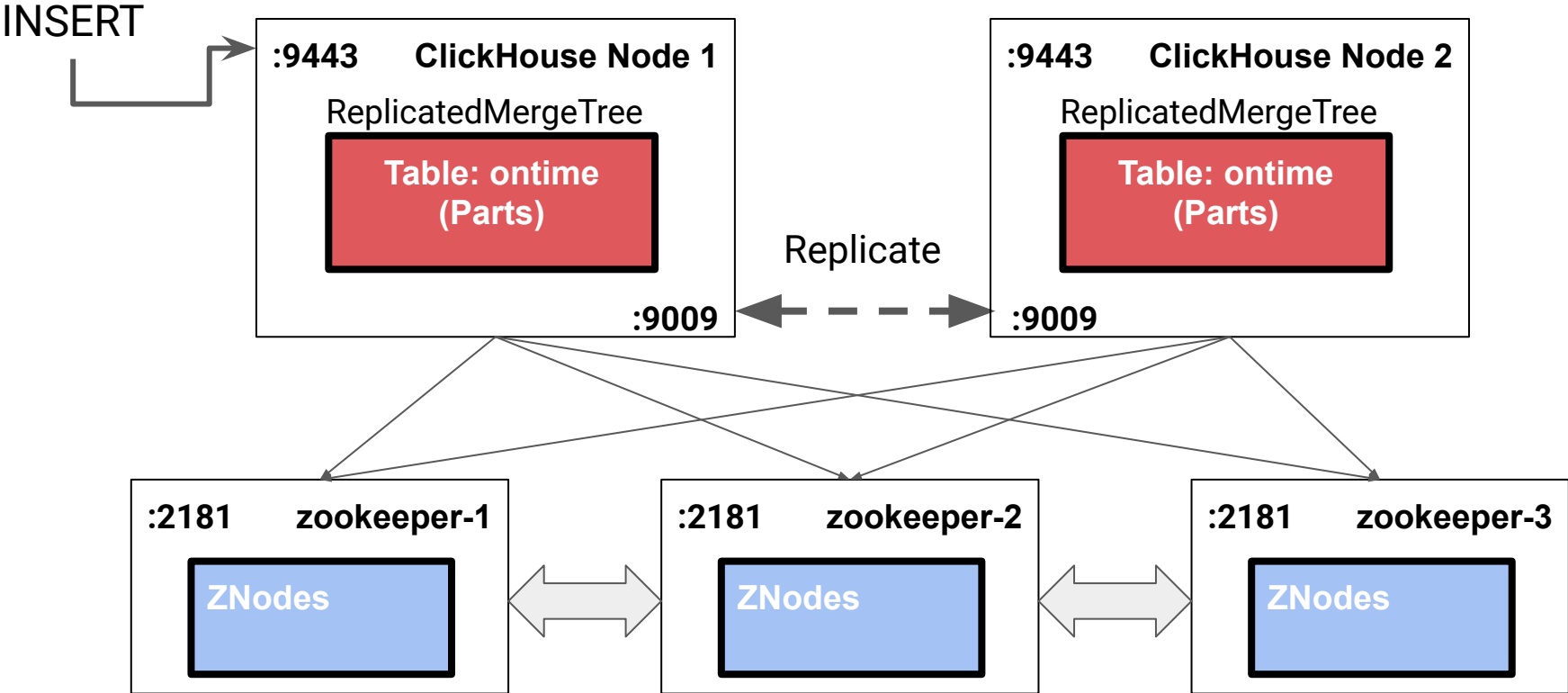


Data sharded 2 ways and replicated 2 times

# MergeTree tables support replication



# How replication works



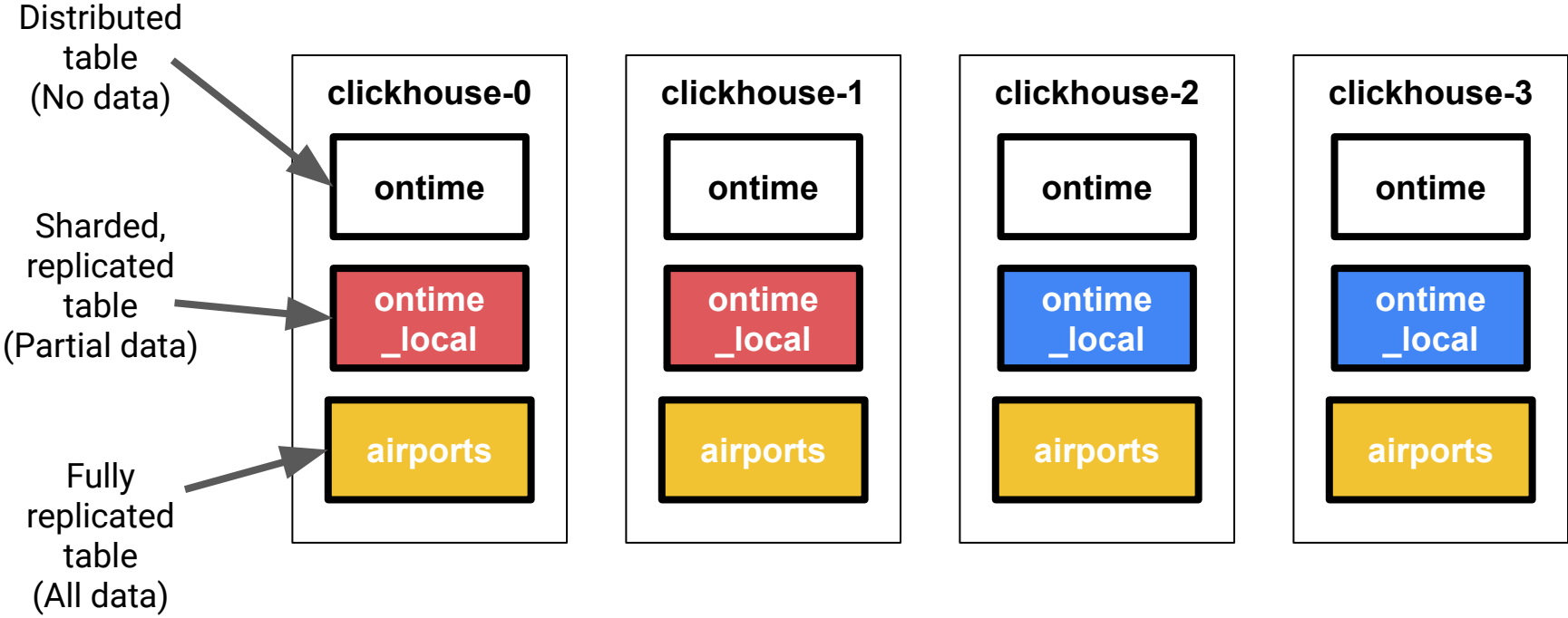
# What is replicated?

Replicated\*MergeTree ONLY

<b>Replicated statements</b>	<b>Non-replicated statements</b>
<ul style="list-style-type: none"><li>● INSERT</li><li>● ALTER TABLE exceptions: FREEZE, MOVE TO DISK, FETCH</li><li>● OPTIMIZE</li><li>● TRUNCATE</li></ul>	<ul style="list-style-type: none"><li>● CREATE table</li><li>● DROP table</li><li>● RENAME table</li><li>● DETACH table</li><li>● ATTACH table</li></ul>

# Building distributed schema

# Example of a distributed data set with shards and replicas



## Step 1: A sharded, replicated fact table

```
CREATE TABLE IF NOT EXISTS `ontime_local` (  
    `Year` UInt16 CODEC(DoubleDelta, ZSTD(1)),  
    `Quarter` UInt8,  
    `Month` UInt8,  
    `DayofMonth` UInt8,  
    `DayOfWeek` UInt8, ...  
) Engine=ReplicatedMergeTree(  
    '/clickhouse/{cluster}/tables/{shard}/{database}/ontime_local',  
    '{replica}')  
PARTITION BY toYYYYMM(FlightDate)  
ORDER BY (FlightDate, `Year`, `Month`, DepDel15)
```

Replication is at the table level!

Use a Replicated% Engine



## Step 2: A distributed table to find data

```
CREATE TABLE IF NOT EXISTS ontime  
AS ontime local  
ENGINE = Distributed(  
    '{cluster}', currentDatabase(), ontime_local, rand())
```

Cluster  
layout

Database

Table

Sharding  
key  
(optional)

## Step 3: A fully replicated dimension table

```
CREATE TABLE IF NOT EXISTS airports
AS default.dot airports
Engine=ReplicatedMergeTree(
  '/clickhouse/{cluster}/tables/all/{database}/airports',
  '{replica}')
PARTITION BY tuple()
PRIMARY KEY AirportID
ORDER BY AirportID
```

**Resolves to current database**


**Don't bother with partitions for small tables**

# Macros help CREATE TABLE ON CLUSTER

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

```
<clickhouse>
  <macros>
    <all-sharded-shard>2</all-sharded-shard>
    <cluster>demo</cluster>
    <shard>0</shard>
    <replica>clickhouse-0-1</replica>
  </macros>
</clickhouse>
```

**Replica names  
should be unique  
per host**



```
select * from system.macros
```

# What does ON CLUSTER do?

ON CLUSTER executes a command over a set of nodes

```
CREATE TABLE IF NOT EXISTS `ontime_local` ON CLUSTER `{cluster}` ...
```

```
DROP TABLE IF EXISTS `ontime_local` ON CLUSTER `{cluster}` ...
```

```
ALTER TABLE `ontime_local` ON CLUSTER `{cluster}` ...
```

# How does ON CLUSTER know where to go?

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

```
<clickhouse>
  <remote_servers>
    <demo>
      <!-- <secret>top secret</secret> -->
      <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>
      <shard>
        . . .
      </shard>
    </demo>
  </remote_servers>
</clickhouse>
```

Cluster name

Shared secret

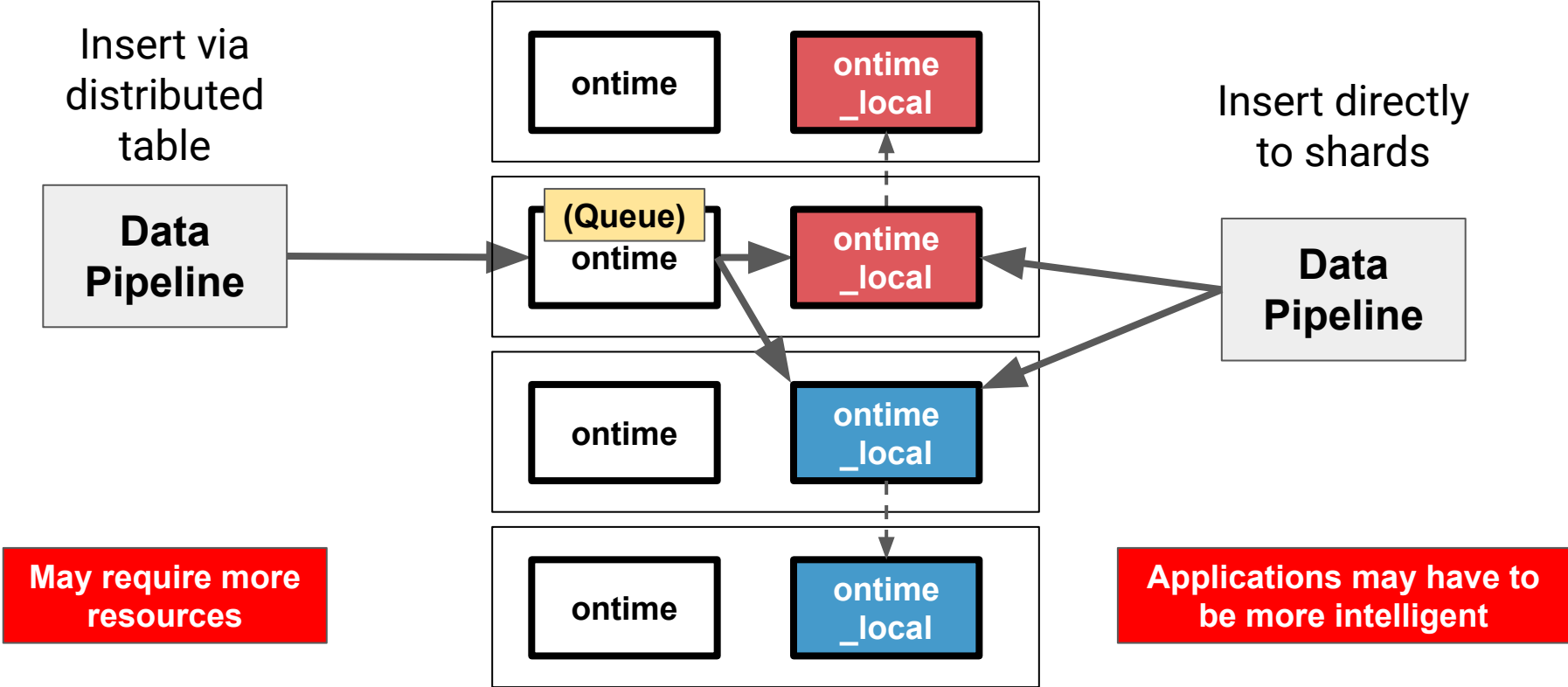
“It’s a cluster  
because I said so!”

## List layouts using system.clusters

```
-- Find name and hosts in each layout
SELECT
  cluster,
  groupArray(concat(host_name, ':', toString(port))) AS hosts
FROM system.clusters
GROUP BY cluster ORDER BY cluster
```

# Loading and querying data

# Data loading: Distributed vs. local INSERTs





## INSERT into a distributed vs. local table

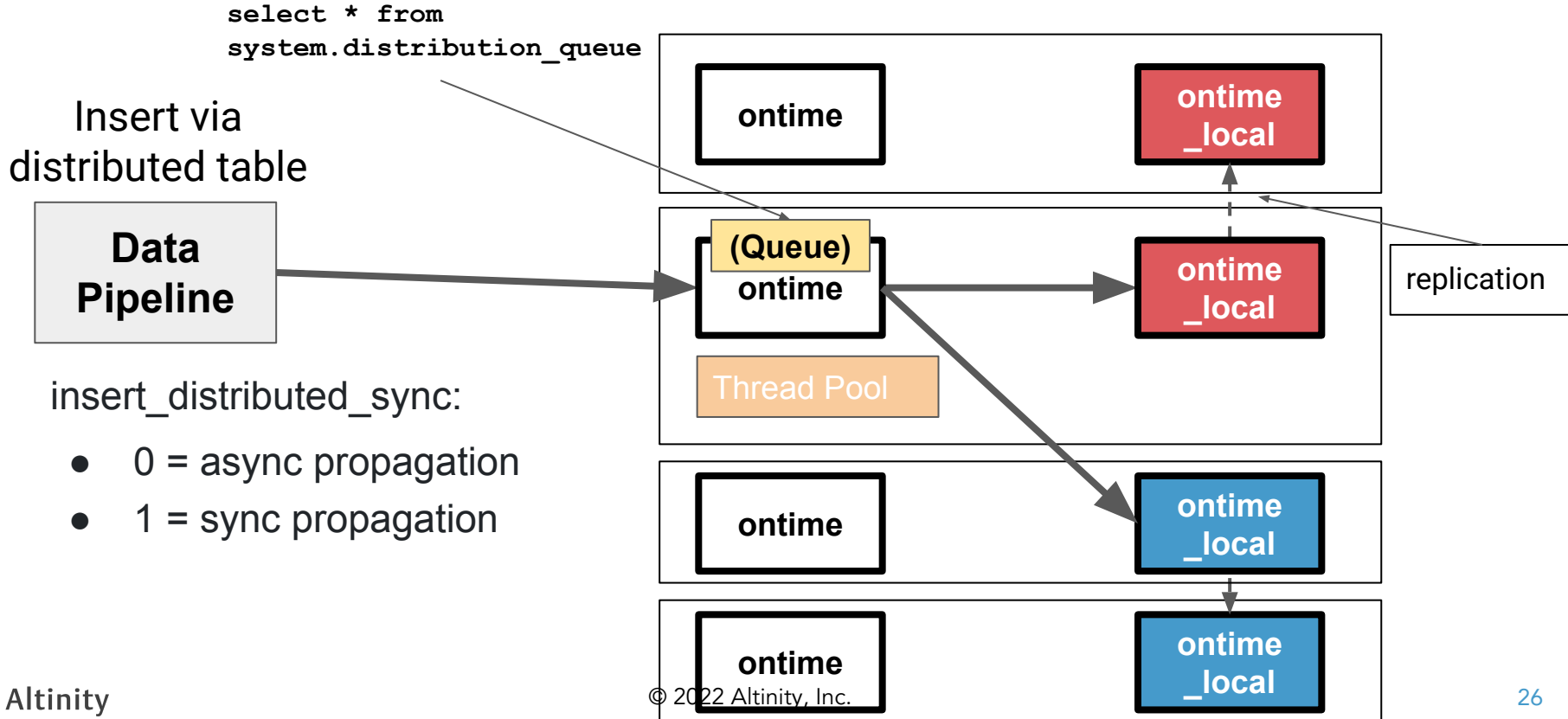
```
-- Insert into distributed table
```

```
INSERT INTO ontime VALUES  
(2017,1,1,1,7,'2017-01-01','AA',19805,...),  
(2017,1,1,1,7,'2017-01-01','AA',19805,...),  
...
```

```
-- Insert into a local table
```

```
INSERT INTO ontime local VALUES  
(2017,1,1,1,7,'2017-01-01','AA',19805,...),  
(2017,1,1,1,7,'2017-01-01','AA',19805,...),  
...
```

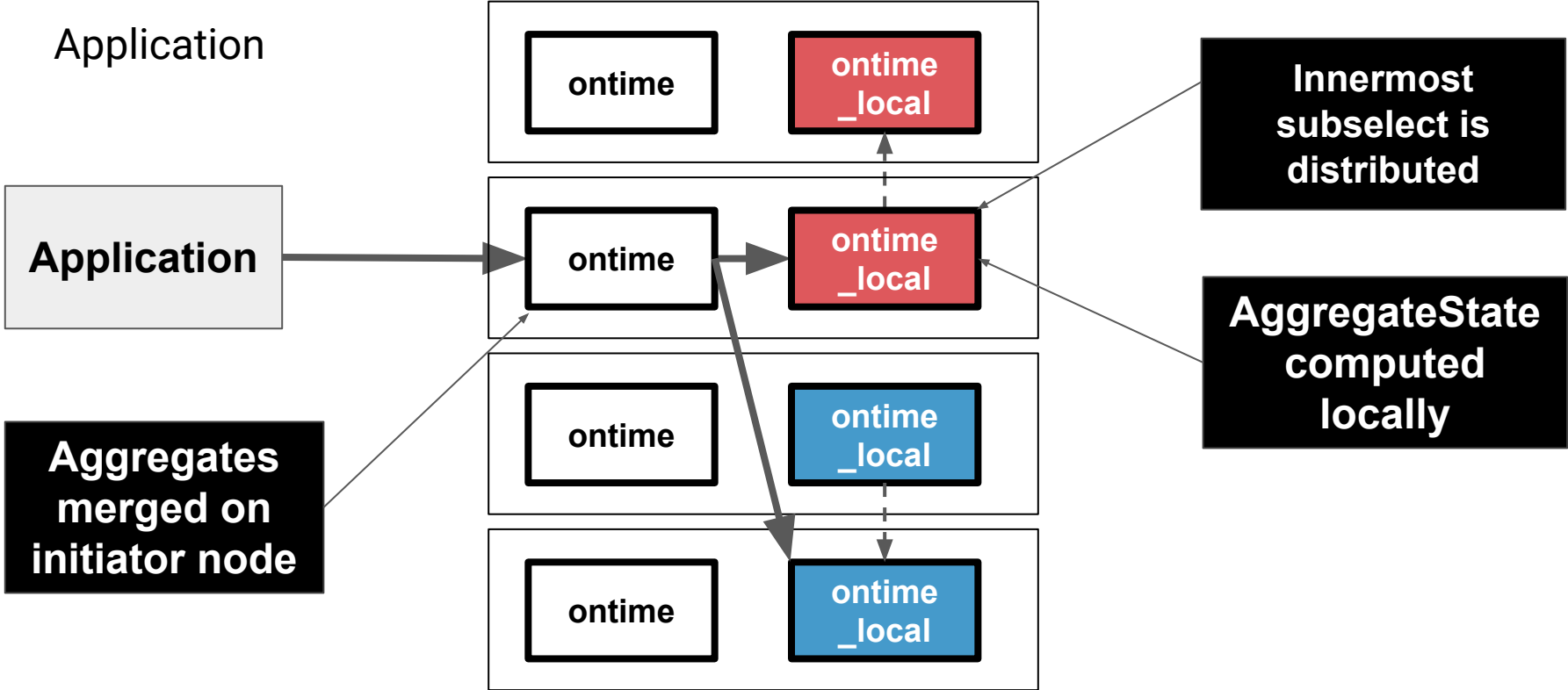
# How does a distributed INSERT work?



# Options for processing INSERTs

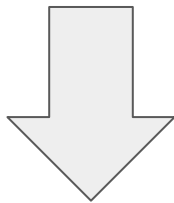
- Local vs distributed data insertion
  - INSERT to local table – no need to sync, larger blocks, faster
  - INSERT to Distributed table – sharding by ClickHouse
  - CHProxy -- distributes transactions across nodes, only works with HTTP connections
- Asynchronous (default) vs synchronous insertions
  - `insert_distributed_sync` - Wait until batches make it to local tables
  - `insert_quorum`, `select_sequential_consistency` – Wait until replicas sync

# How do distributed SELECTs work?



## Queries are pushed to all shards

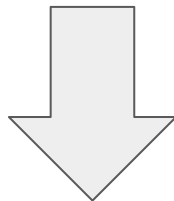
```
SELECT Carrier, avg(DepDelay) AS Delay  
FROM ontime  
GROUP BY Carrier ORDER BY Delay DESC
```



```
SELECT Carrier, avg(DepDelay) AS Delay  
FROM ontime local  
GROUP BY Carrier ORDER BY Delay DESC
```

# ClickHouse pushes down JOINS by default

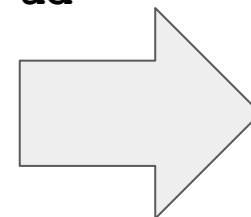
```
SELECT o.Dest d, a.Name n, count(*) c, avg(o.ArrDelayMinutes) ad
FROM default.ontime o
JOIN default.airports a ON (a.IATA = o.Dest)
GROUP BY d, n HAVING c > 100000 ORDER BY d DESC
LIMIT 10
```



```
SELECT Dest AS d, Name AS n, count() AS c, avg(ArrDelayMinutes) AS
ad
FROM default.ontime local AS o
ALL INNER JOIN default.airports AS a ON a.IATA = o.Dest
GROUP BY d, n HAVING c > 100000 ORDER BY d DESC LIMIT 10
```

...Unless the left side “table” is a subquery

```
SELECT d, Name n, c AS flights, ad
FROM
(
  SELECT Dest d, count(*) c, avg(ArrDelayMinutes) ad
  FROM default.ontime
  GROUP BY d HAVING c > 100000
  ORDER BY ad DESC
) AS o
LEFT JOIN airports ON airports.IATA = o.d
LIMIT 10
```



**Remote  
Servers**

## It's more complex when multiple tables are distributed

```
select foo from T1 where a in (select a from T2)
```

distributed\_product\_mode=?

### local

```
select foo  
from T1_local  
where a in (  
  select a  
  from T2_local)
```

(Subquery runs on  
local table)

### allow

```
select foo  
from T1_local  
where a in (  
  select a  
  from T2)
```

(Subquery runs on  
distributed table)

### global

```
create temporary table  
tmp Engine = Set  
AS select a from T2;
```

```
select foo from  
T1_local where a in  
tmp;
```

(Subquery runs on initiator;  
broadcast to local temp table)

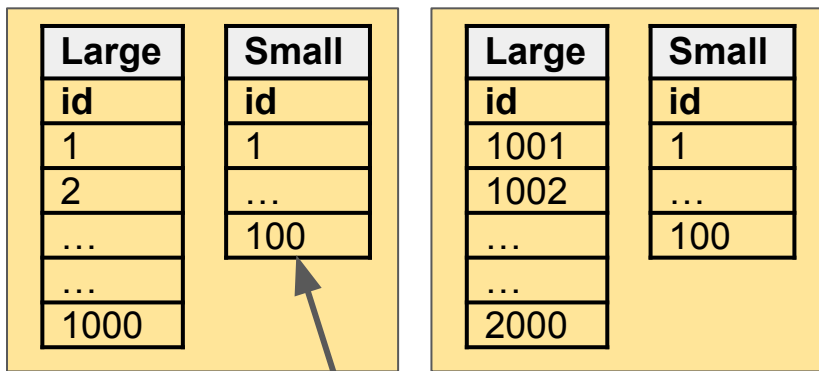


# What's actually happening with queries? Let's find out!

```
SELECT hostName() host, event_time, query_id,  
       is_initial_query AS initial,  
       if(is_initial_query, '', initial_query_id) as initial_q,  
       query  
FROM cluster('{cluster}', system.query_log) AS st  
WHERE type = 'QueryFinish' AND has(databases, 'test')  
ORDER BY st.event_time DESC LIMIT 25
```

# Thinking about distributed data and joins

“Big Table Model”

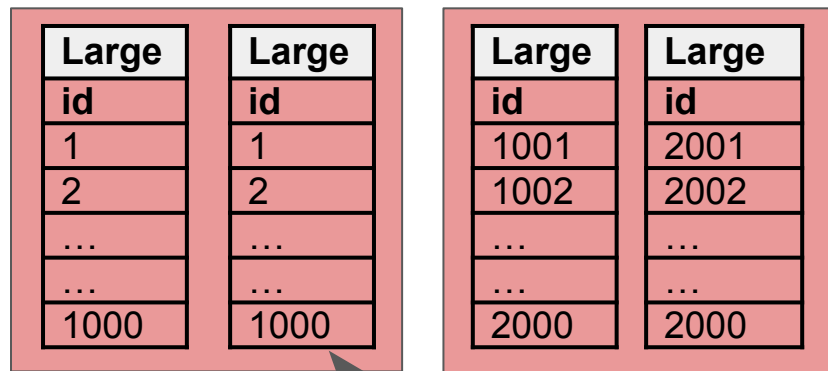


Shard 1

Shard 2

All keys replicated

“Bucketing Model”



Shard 1

Shard 2

Matching keys in each bucket

# Tricks to query distributed tables

## Use remote() to select from another node

```
SELECT count()  
FROM remote('host-2', currentDatabase(), 'ontime_ref')
```

```
SELECT count()  
FROM remoteSecure('host-2', currentDatabase(), 'ontime_ref')
```

```
count()  
196508419
```

```
-- You can insert too, with FUNCTION keyword.  
INSERT INTO FUNCTION remote(host, database, table, login,  
password)  
VALUES . . .
```

# More remote query tricks!

```
SELECT hostName() AS h, count() AS c FROM sdata GROUP BY h
```

h	c
chi-test-rh-test-rh-1-0-0	492
chi-test-rh-test-rh-0-0-0	508

**Distributed table**

```
SELECT hostName() AS h, count() AS c  
FROM remote('chi-test-rh-test-rh-{0,1}-{0,1}', default, sdata)  
GROUP BY h
```

h	c
chi-test-rh-test-rh-1-0-0	984
chi-test-rh-test-rh-1-1-0	984
chi-test-rh-test-rh-0-1-0	1016
chi-test-rh-test-rh-0-0-0	1016

**Remote query all 4  
hosts**

## cluster() distributes queries dynamically

```
SELECT
  hostName() AS host, count() AS tables
FROM cluster('{cluster}', system.tables)
WHERE database = 'default'
GROUP BY host
```

host	tables
chi-test-rh-test-rh-1-0-0	2
chi-test-rh-test-rh-0-1-0	2

## clusterAllReplicas() goes to every node

```
SELECT
  hostName() AS host, count() AS tables
FROM clusterAllReplicas('{cluster}', system.tables)
WHERE database = 'default'
GROUP BY host
```

host	tables
chi-test-rh-test-rh-1-0-0	2
chi-test-rh-test-rh-1-1-0	2
chi-test-rh-test-rh-0-1-0	2
chi-test-rh-test-rh-0-0-0	2

# Scaling up

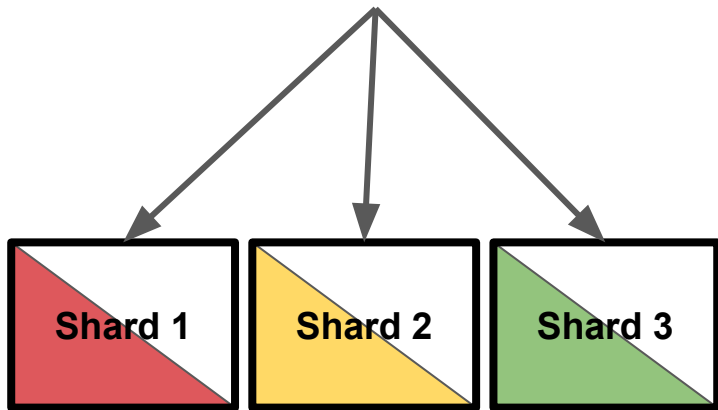


# Load testing and capacity planning made simple...

1. Establish single node baseload
  - Use production data
  - Max out SELECT & INSERT capacity with load tests
  - Adjust schema and queries, retest
2. Add replicas to increase SELECT capacity
3. Add shards to increase INSERT capacity

# Selecting the sharding key

## Randomized Key, e.g., cityHash64(Url)

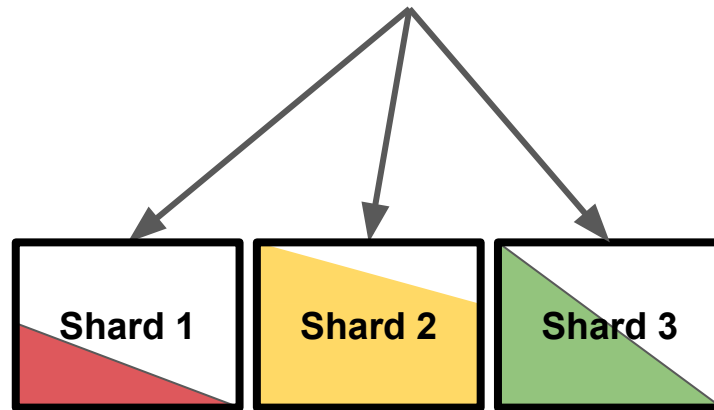


**Must query  
all shards**

**Nodes are  
balanced**

**Easier to  
add nodes**

## Specific Key e.g., cityHash64(TenantId)



**Unbalanced  
nodes**

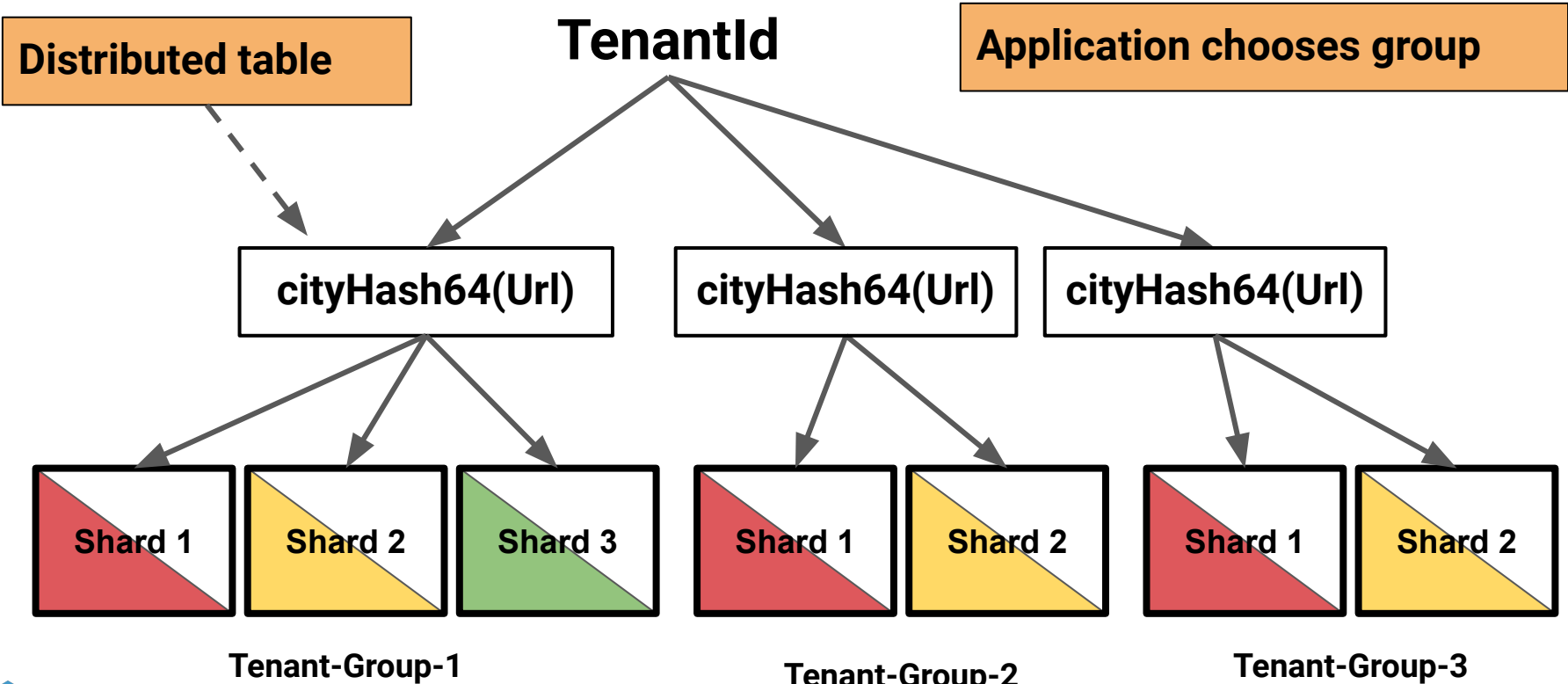
**Hard to  
add nodes**

**Queries can  
skip shards**

# Options for shard rebalancing

- INSERT INTO new\_cluster SELECT FROM old\_cluster
  - Clickhouse-copier automates this
- Use (undocumented) [ALTER TABLE MOVE PART TO SHARD](#)
  - Example: ALTER TABLE test\_move MOVE PART 'all\_0\_0\_0' TO SHARD '/clickhouse/shard\_1/tables/test\_move'
- Move parts manually
  - ALTER TABLE FREEZE PARTITION
  - rsync to new host
  - ALTER TABLE ATTACH PARTITION
  - Drop original partition

# Bi-level sharding combines both approaches



# Wrap-up and more information

# 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/UCE3Y2IDKl\\_ZfjaCrh62onYA](https://www.youtube.com/channel/UCE3Y2IDKl_ZfjaCrh62onYA)

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

[ClickHouse Capacity Planning by Mik Kocikowski of CloudFlare](#)

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](https://clickhousedb.slack.com)
- Altinity Public Workspace - [altinitydbworkspace.slack.com](https://altinitydbworkspace.slack.com)

Education - [Altinity ClickHouse Training](#)

Support - Altinity offers [support for ClickHouse](#) in all environments

A close-up photograph of various mechanical components, including gears, bearings, and metal plates, arranged in a complex, overlapping manner. The lighting is dramatic, highlighting the metallic textures and creating deep shadows.

# Thank you and good luck!

Website: <https://altinity.com>

Email: [info@altinity.com](mailto:info@altinity.com)

Slack: [altinitydbworkspace.slack.com](https://altinitydbworkspace.slack.com)

[Altinity.Cloud](#)

[Altinity Support](#)

[Altinity Stable  
Builds](#)

[We're hiring!](#)