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

Alexander Zaitsev and Altinity Engineering
26 March 2024

Let's make some introductions

Us

Database geeks with centuries of experience in DBMS and applications

You

Applications developers looking to learn about ClickHouse



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

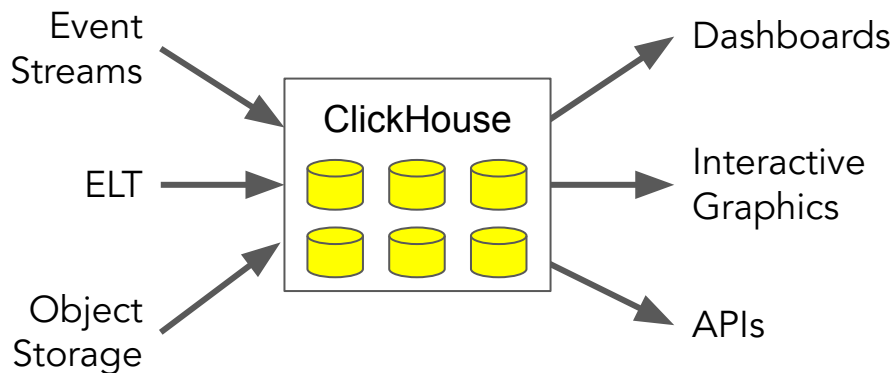
Shared nothing architecture

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”

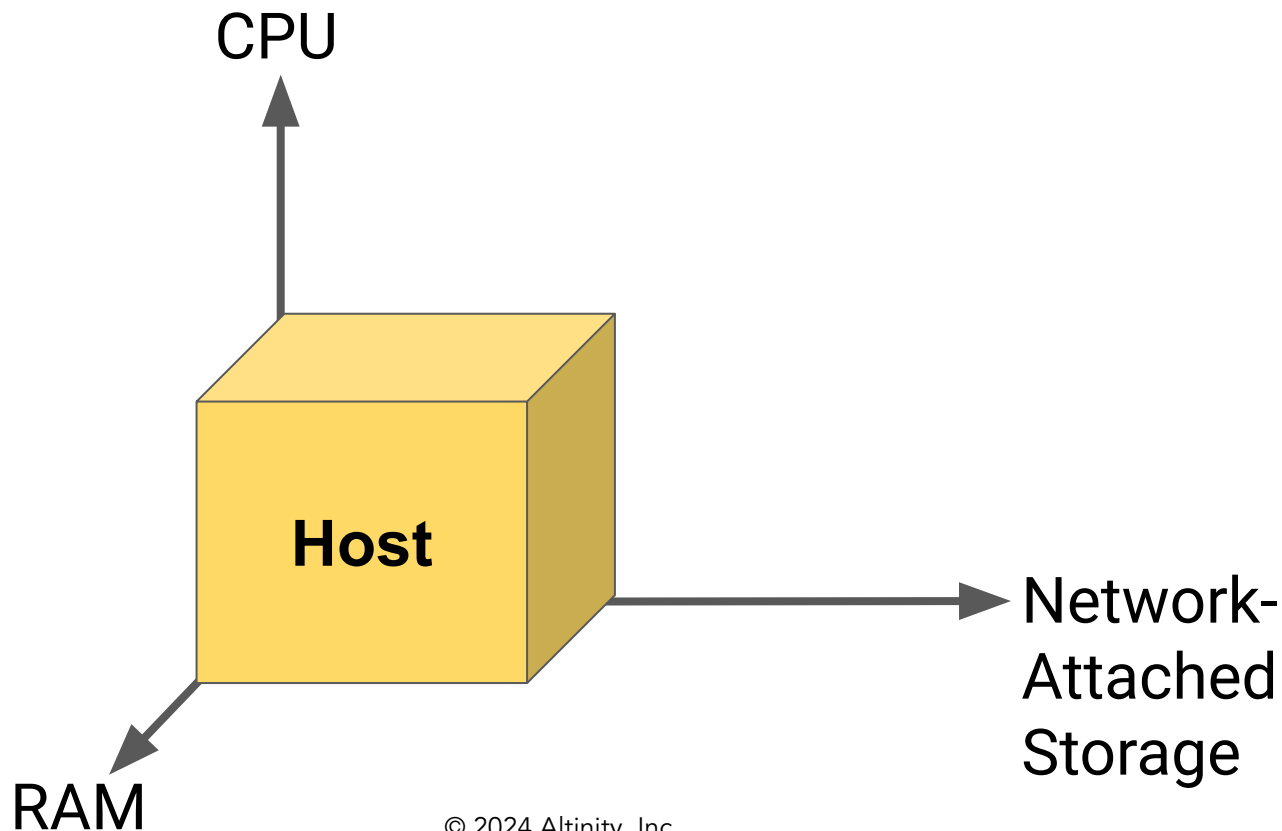


Width:
2 meters

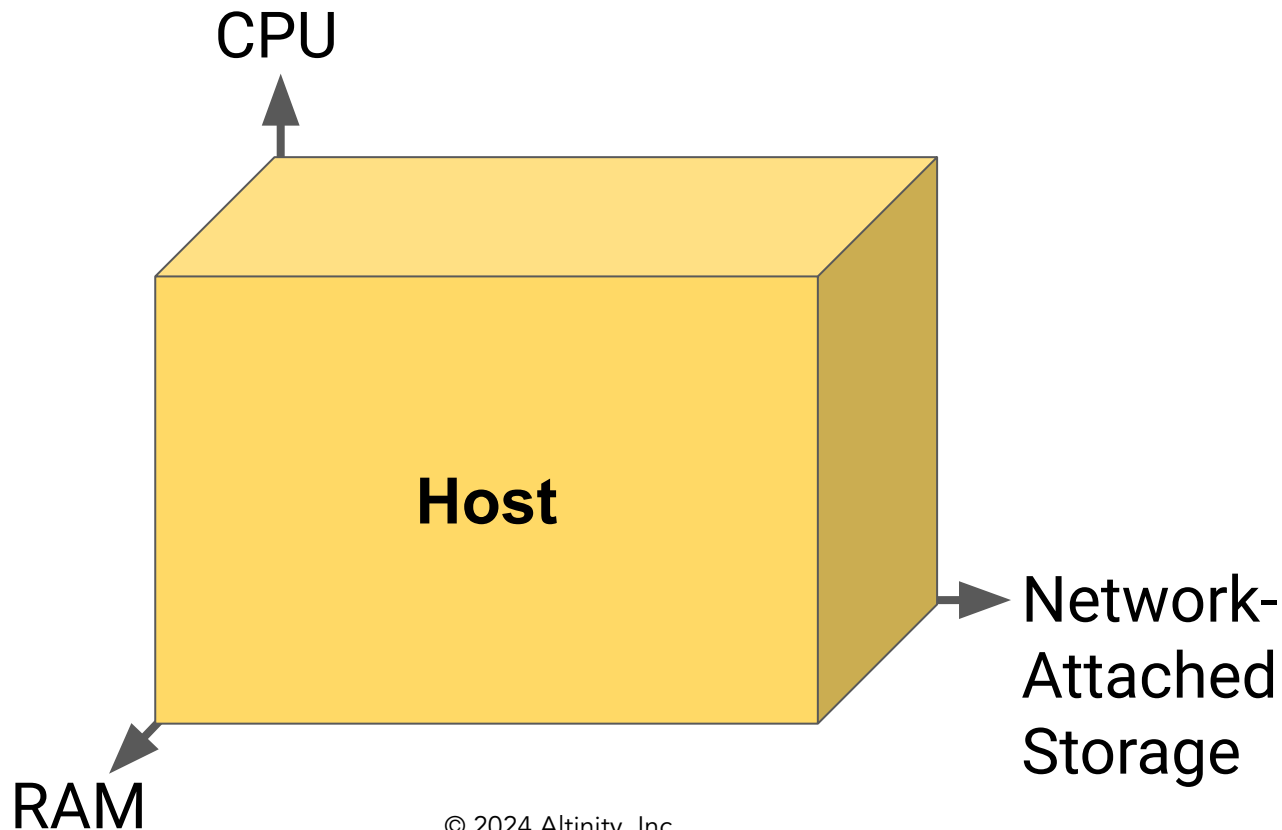
Depth:
60 meters

Introducing sharding and replication

Clickhouse nodes can scale vertically

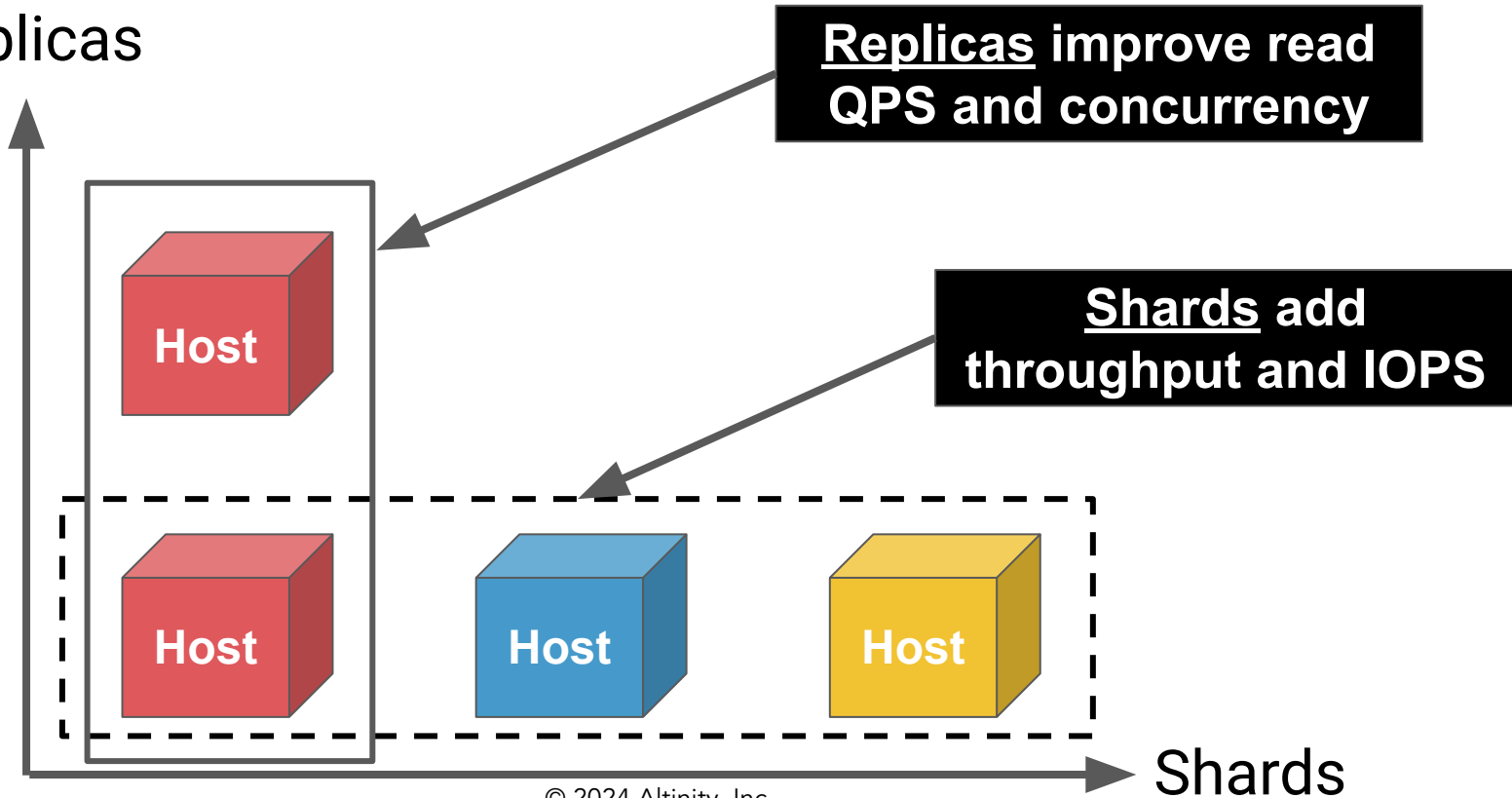


Clickhouse nodes can scale vertically



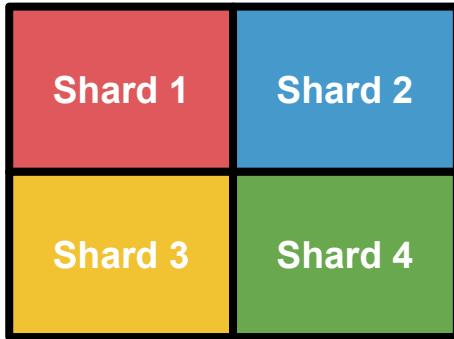
Clusters introduce horizontal scaling

Replicas



Different sharding and replication patterns

All Sharded



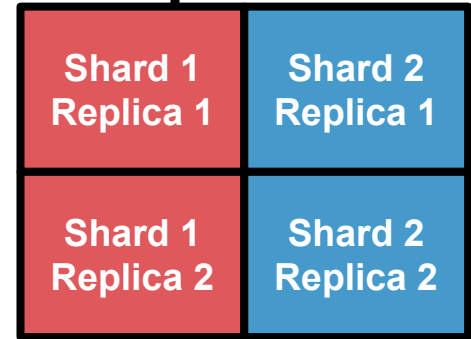
Data sharded 4 ways without replication

All Replicated



Data replicated 4 times without sharding

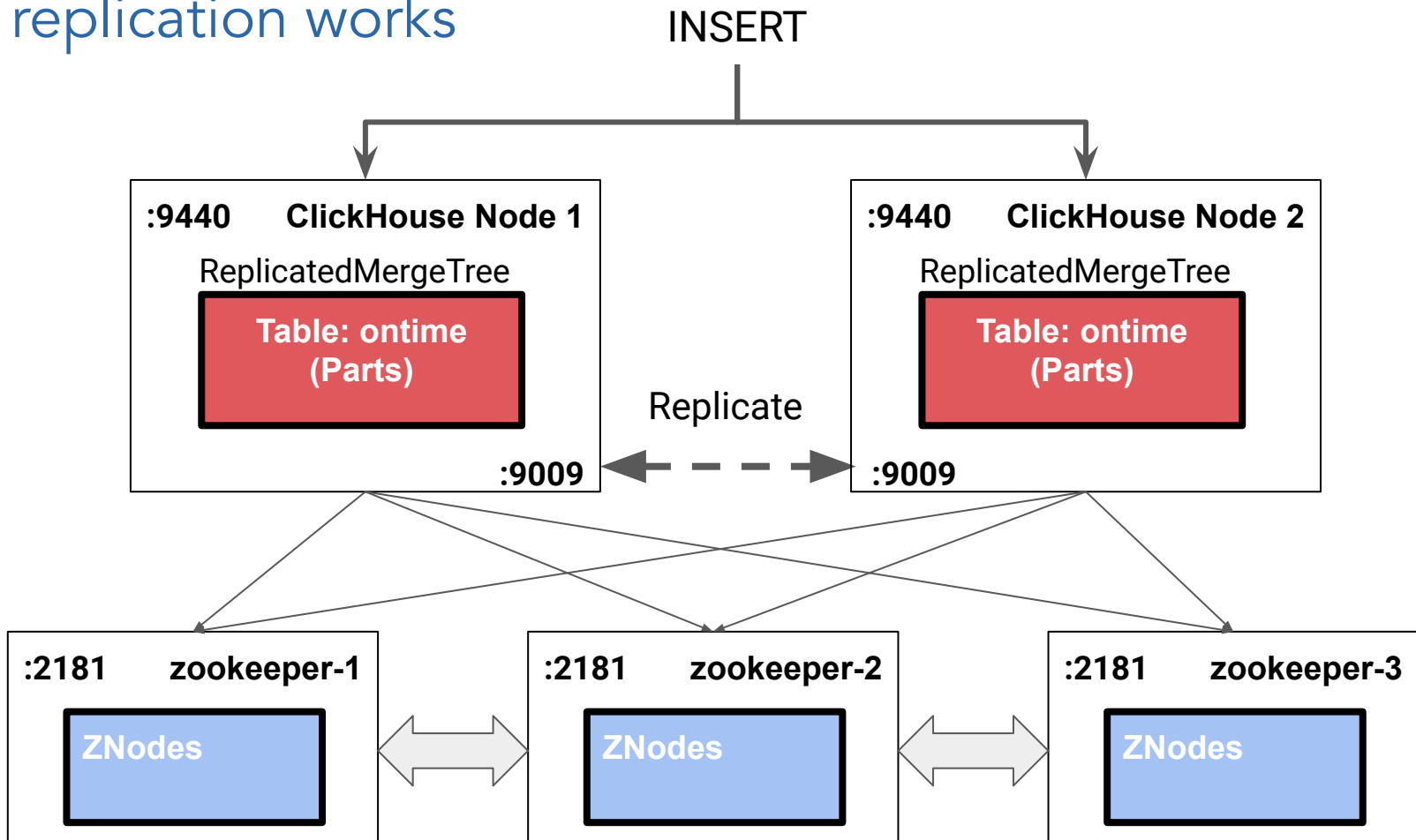
Sharded and Replicated



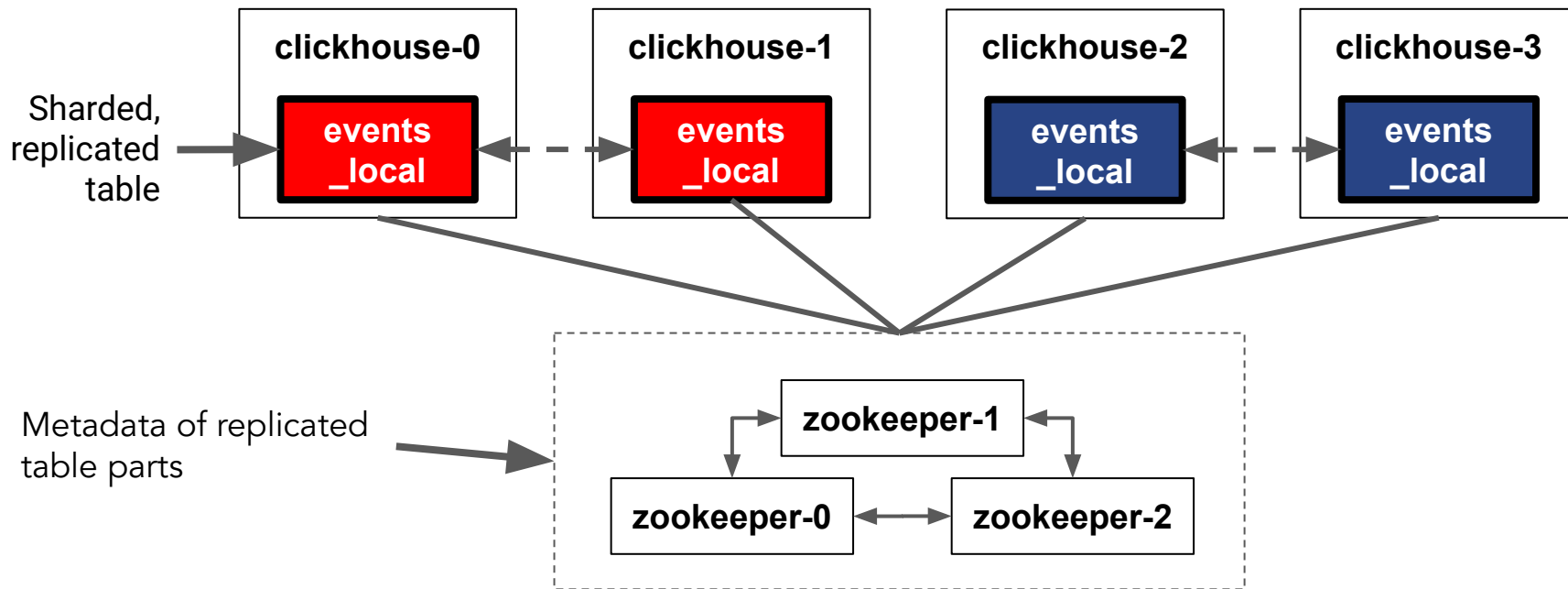
Data sharded 2 ways and replicated 2 times

ClickHouse Replication

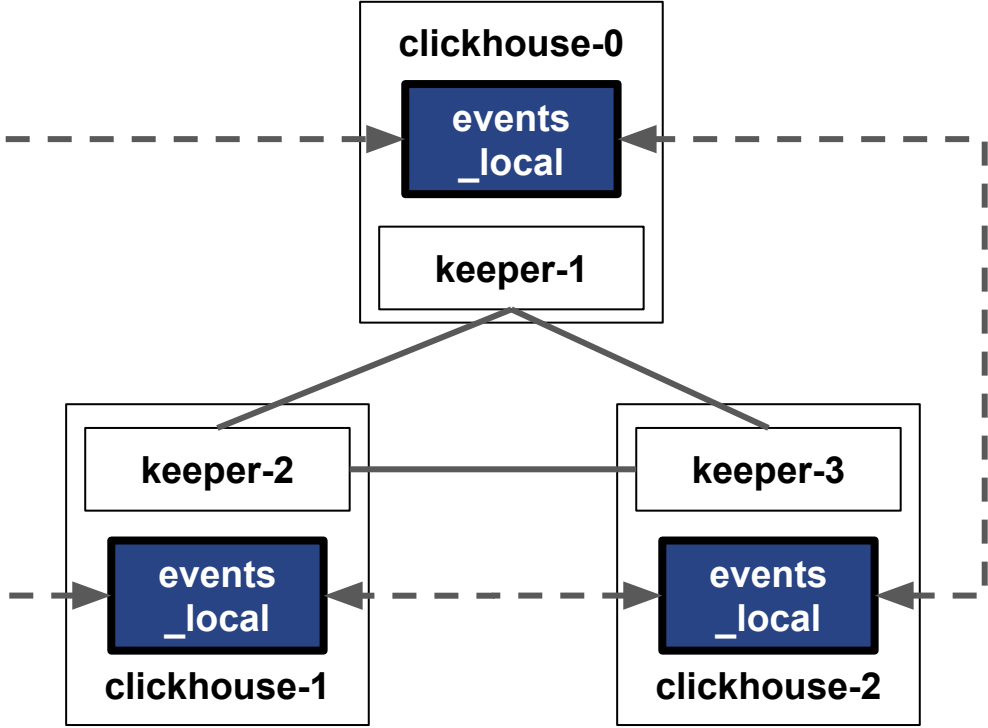
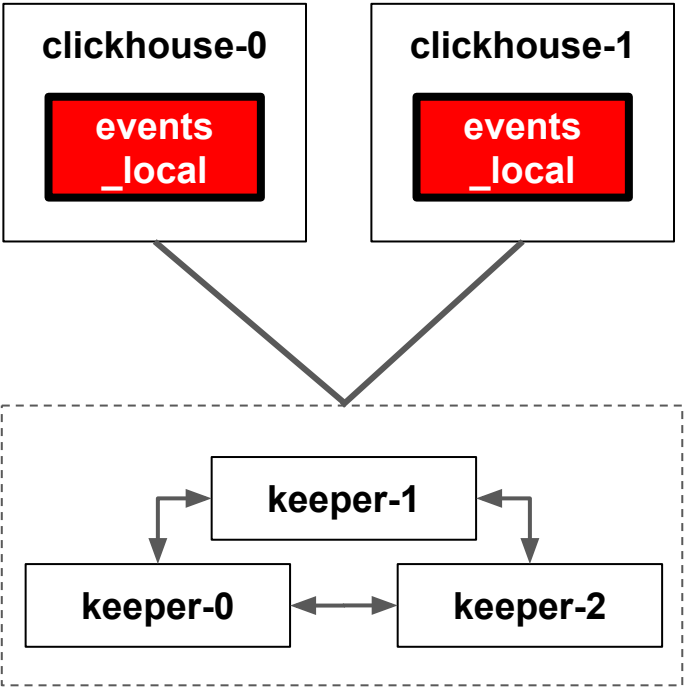
How replication works



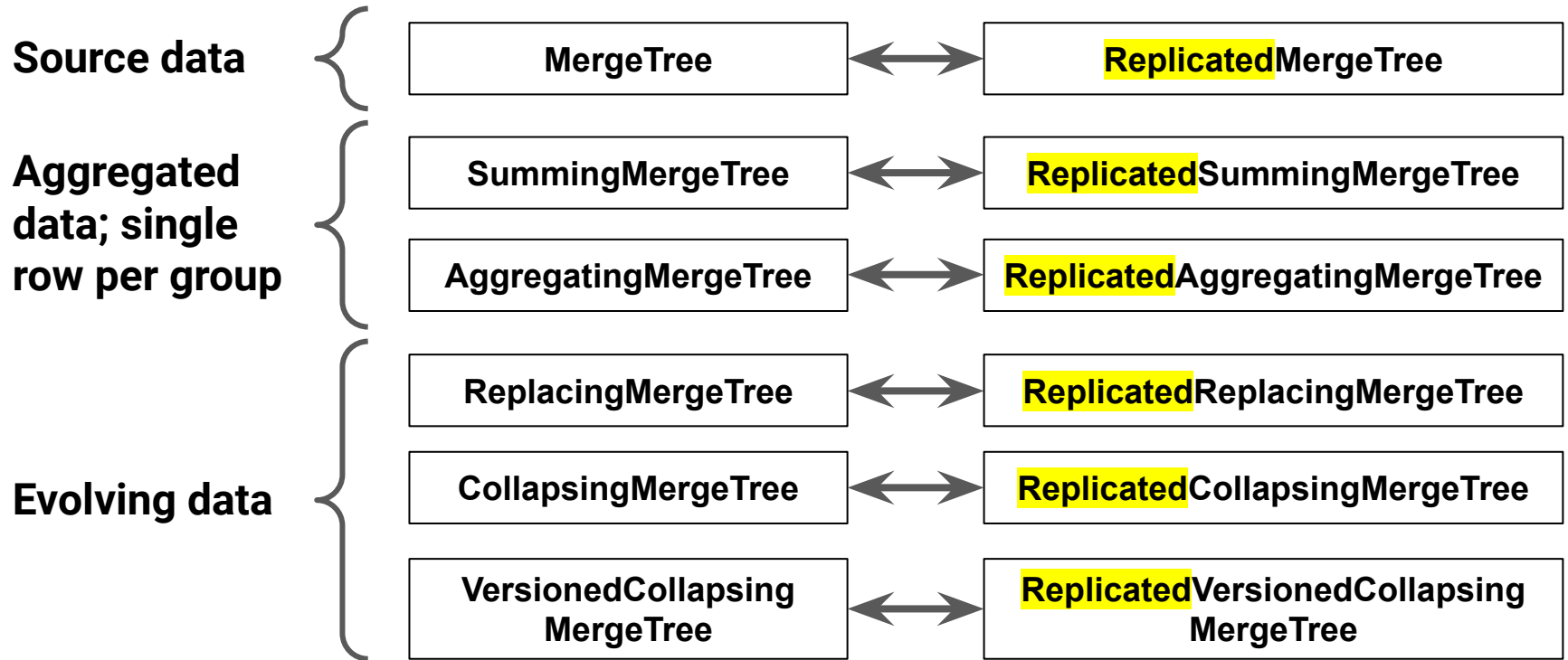
(Zoo)Keeper solves the distributed consistency problem



Keeper can run separately or directly in ClickHouse itself!



All MergeTree tables support replication

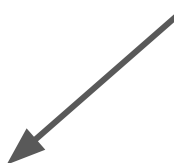


Replicated table


```
CREATE TABLE IF NOT EXISTS airports (  
  ...  
)  
Engine=ReplicatedMergeTree(  
  '/clickhouse/{cluster}/tables/all/{database}/airports',  
  '{replica}')
```

PARTITION BY tuple()
PRIMARY KEY AirportID
ORDER BY AirportID

**Replica path
uniquely identifies
data**



**Replica name
refers to data copy**



Replicated table

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

Replica path can be automatic

server.xml:

```
<default_replica_path>/clickhouse/tables/{uuid}/{shard}</default_replica_path>  
<default_replica_name>{replica}</default_replica_name>
```

What is replicated?

Replicated*MergeTree ONLY

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

Converting non-Replicated table to Replicated

Manual (works in all versions):

1. Create Replicated table
2. ATTACH partitions one-by-one from non-replicated

See

<https://kb.altinity.com/altinity-kb-setup-and-maintenance/altinity-kb-converting-mergetree-to-replicated/>

24.2: `convert_to_replicated` flag in table data directory. See <https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/replicatedmergetree#converting-from-mergetree-to-replicatedmergetree>

24.x? ALTER TABLE MODIFY ENGINE.
WIP

<https://github.com/ClickHouse/ClickHouse/pull/58746>

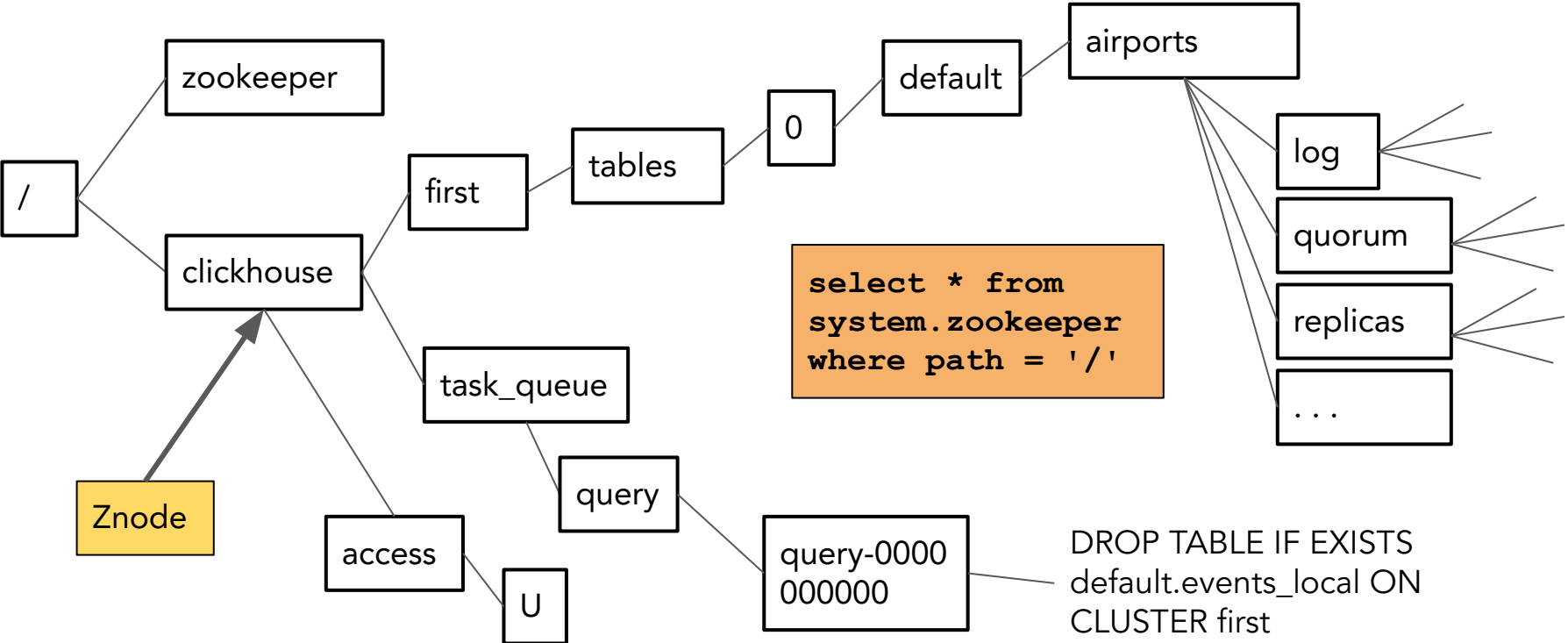
What else can be replicated?

- DDL statements, when using ON CLUSTER statements
- Users/RBAC – requires user_directories configuration:

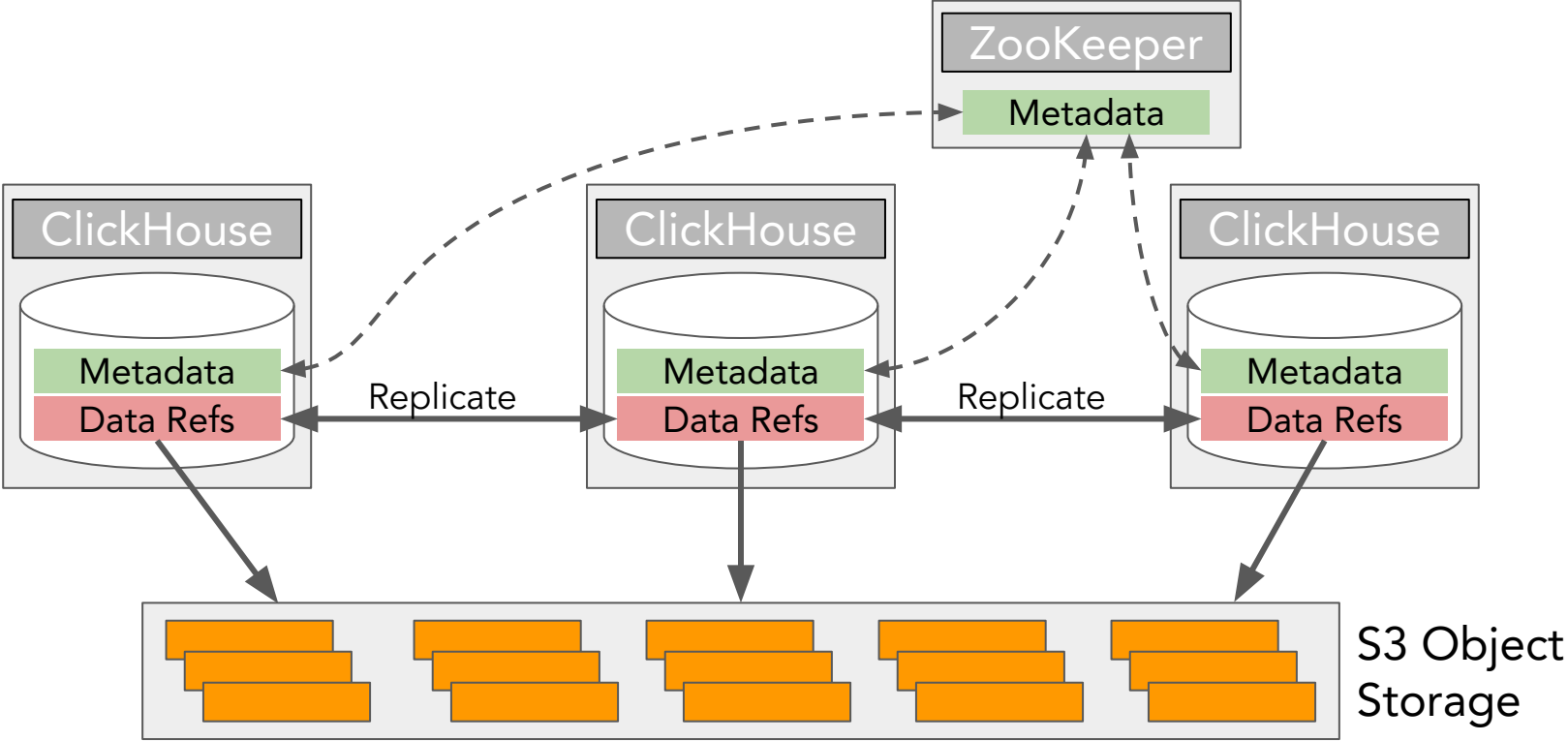
```
<user_directories replace="replace">
  <users_xml>
    <path>/etc/clickhouse-server/users.xml</path>
  </users_xml>
  <replicated>
    <zookeeper_path>/clickhouse/access/</zookeeper_path>
  </replicated>
</user_directories>
```

- UDFs – requires user_defined_zookeeper_path setting
- Parts of server configuration – `<include from_zk="path_in_zookeeper"/>`
 - Managed outside of ClickHouse in this case

How is it stored in (Zoo)Keeper

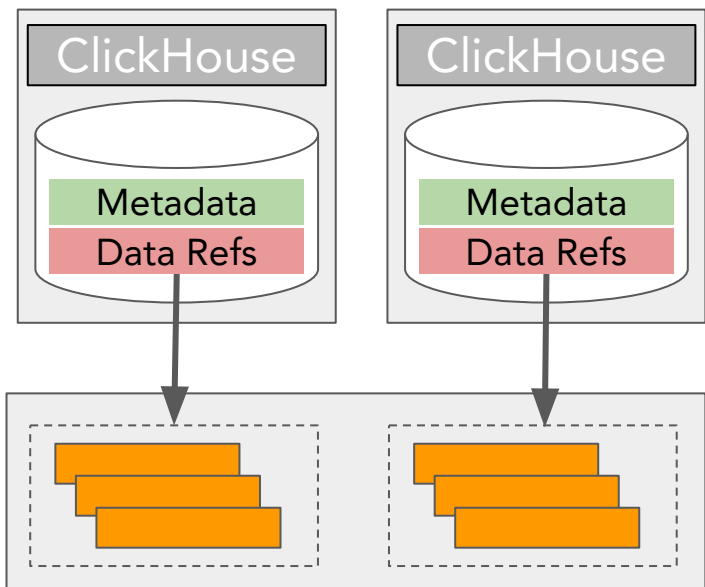


ReplicatedMergeTree over Object Storage

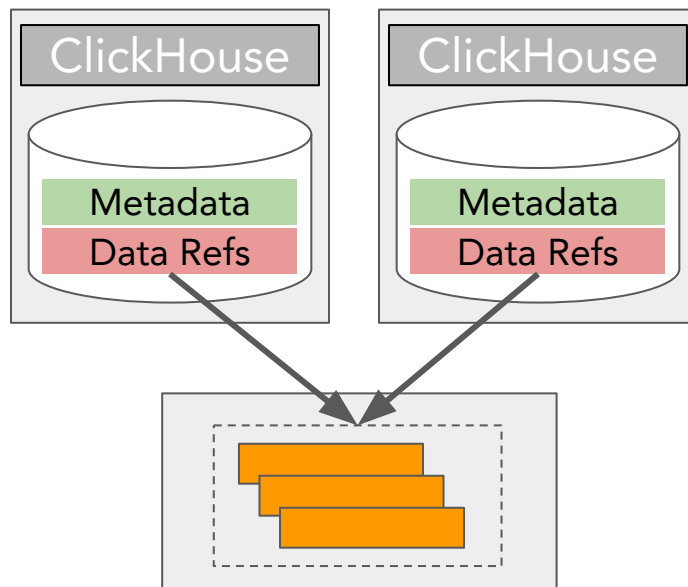


Two models for storing S3 table data

Multiple copies of S3 data

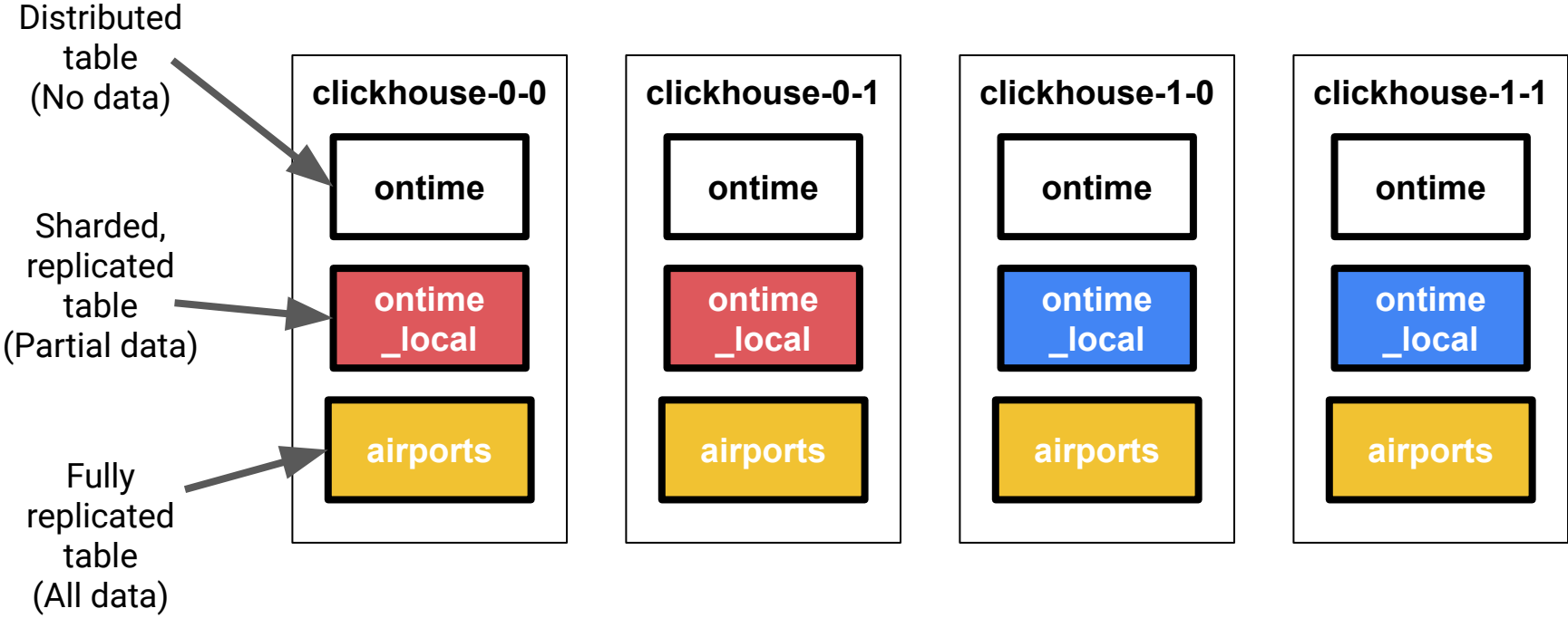


"Zero Copy"



Building distributed schema

Example of a distributed data set with shards and replicas



Step 1. Define a cluster

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

```
<clickhouse>
  <remote_servers>
    <demo>
      <!-- <secret>top secret</secret> -->
      <shard>
        <replica><host>clickhouse-0-0</host><port>9000</port></replica>
        <replica><host>clickhouse-0-1</host><port>9000</port></replica>
        <internal_replication>true</internal_replication>
      </shard>
      <shard>
        . . .
      </shard>
    </demo>
  </remote_servers>
</clickhouse>
```

Cluster name (points to `<demo>`)

Shared secret (points to `<secret>top secret</secret>`)

“It’s a cluster because I said so!”

List layouts using system.clusters


```
-- List 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
```

Step 2. 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
```

Step 3: A sharded, replicated fact table

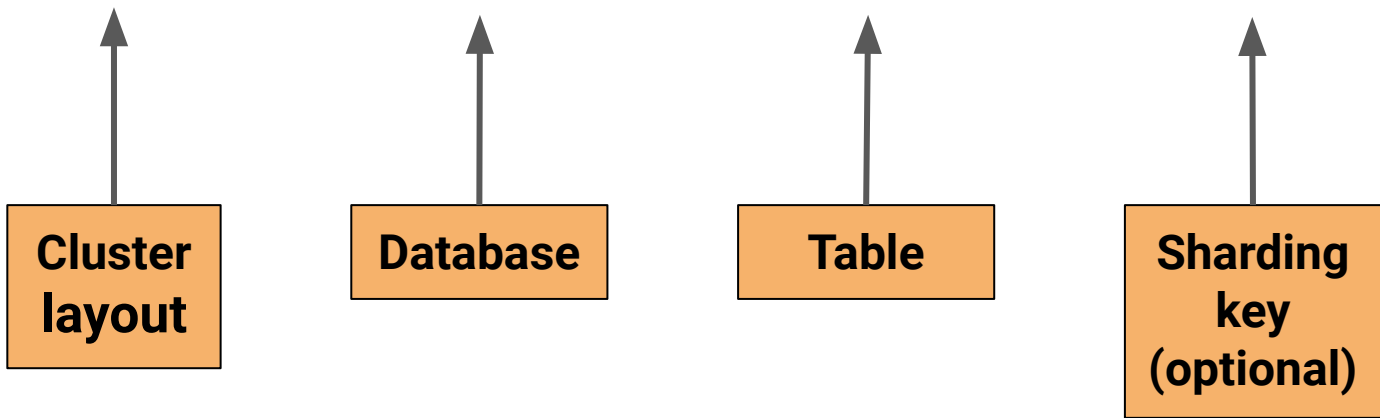
```
CREATE TABLE IF NOT EXISTS ontime_local (  
    `Year` UInt16 CODEC(DoubleDelta, ZSTD(1)),  
    `Quarter` UInt8,  
    `Month` UInt8,  
    `DayofMonth` UInt8,  
    `DayOfWeek` UInt8, ...  
) Engine=ReplicatedMergeTree(  
    '/clickhouse/{cluster}/tables/{shard}/{database}/{table}',  
    '{replica}')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

Replication is at the table level!

Use a Replicated% Engine

Step 4: A distributed table to find data

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



Step 5: 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

What does ON CLUSTER do?

ON CLUSTER executes a command over a set of nodes

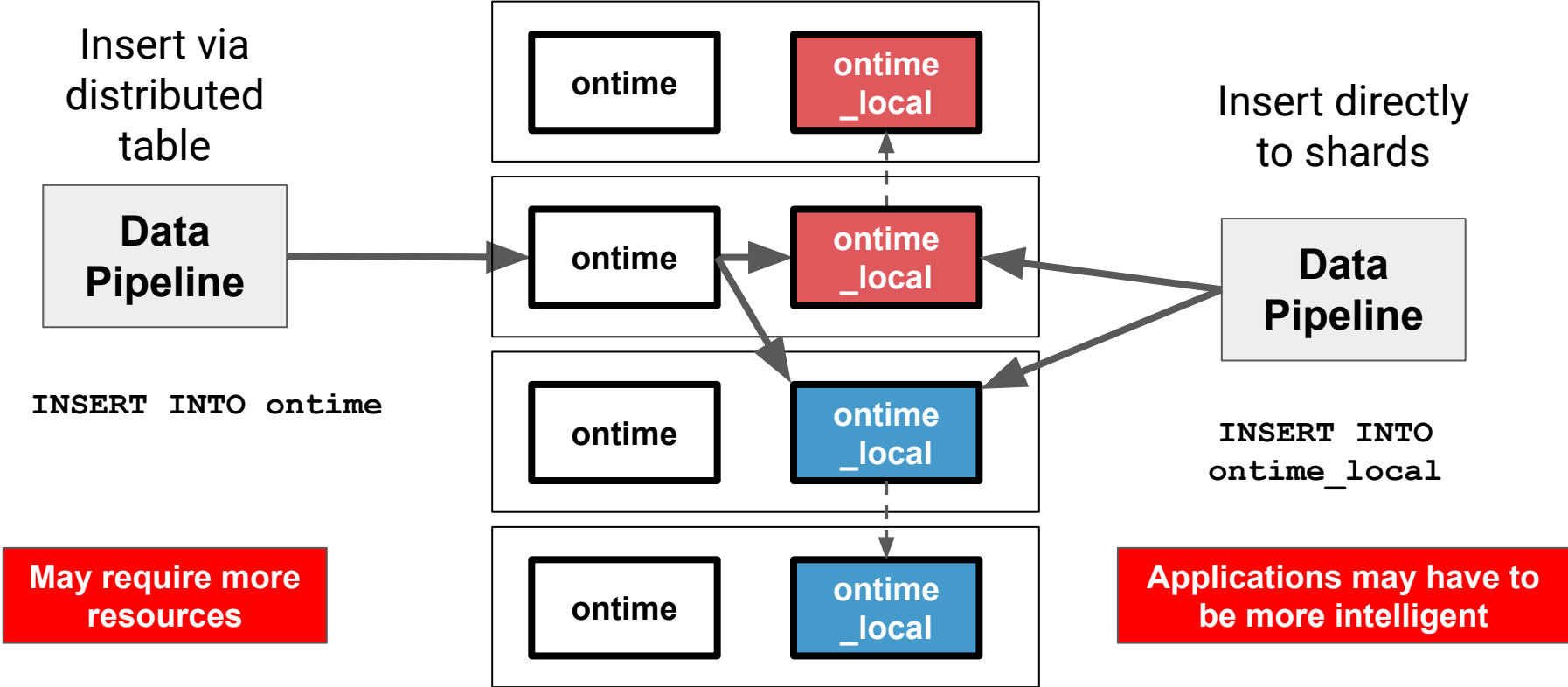
```
CREATE TABLE IF NOT EXISTS `ontime_local` ON CLUSTER `{cluster}` ...
```

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

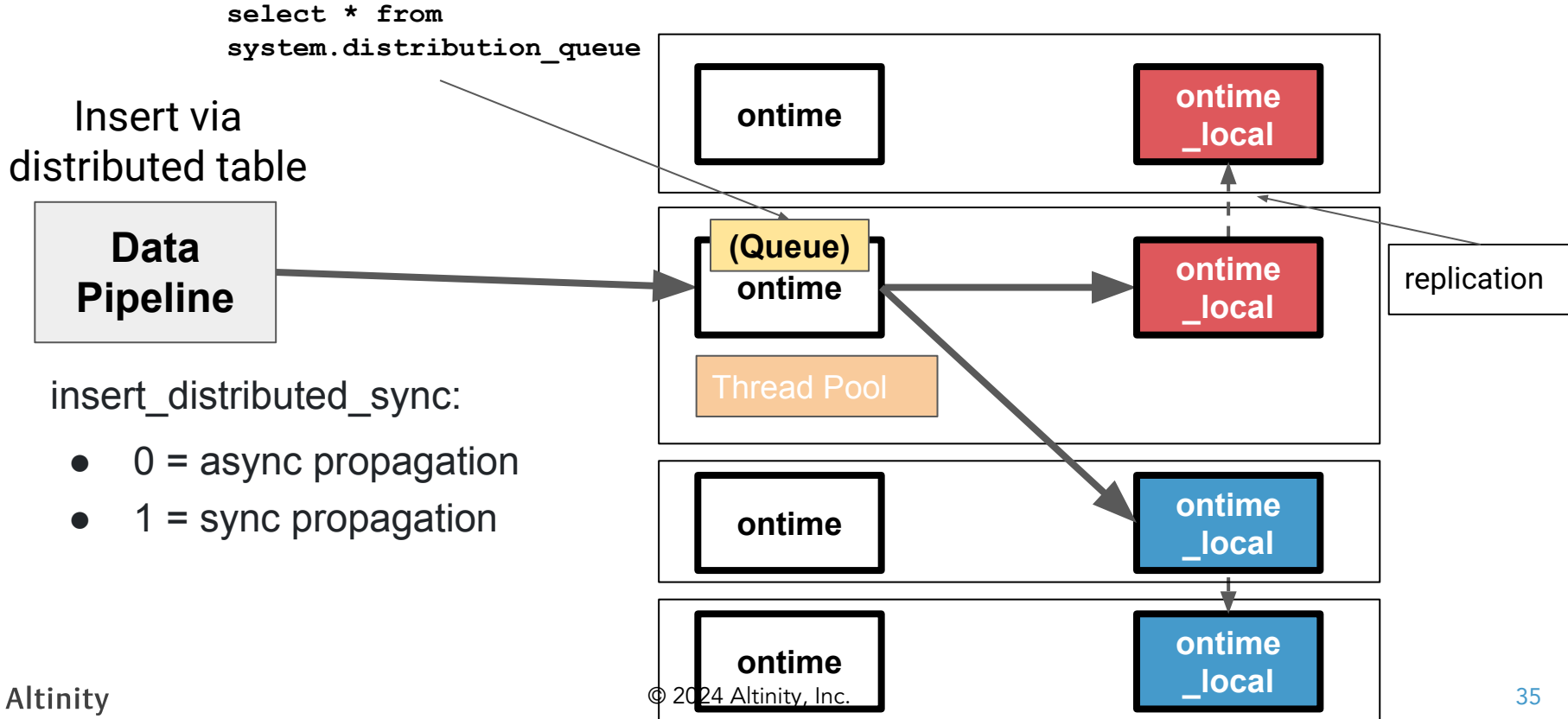
```
ALTER TABLE `ontime_local` ON CLUSTER `{cluster}` ...
```


Loading and querying data

Data loading: Distributed vs. local INSERTs



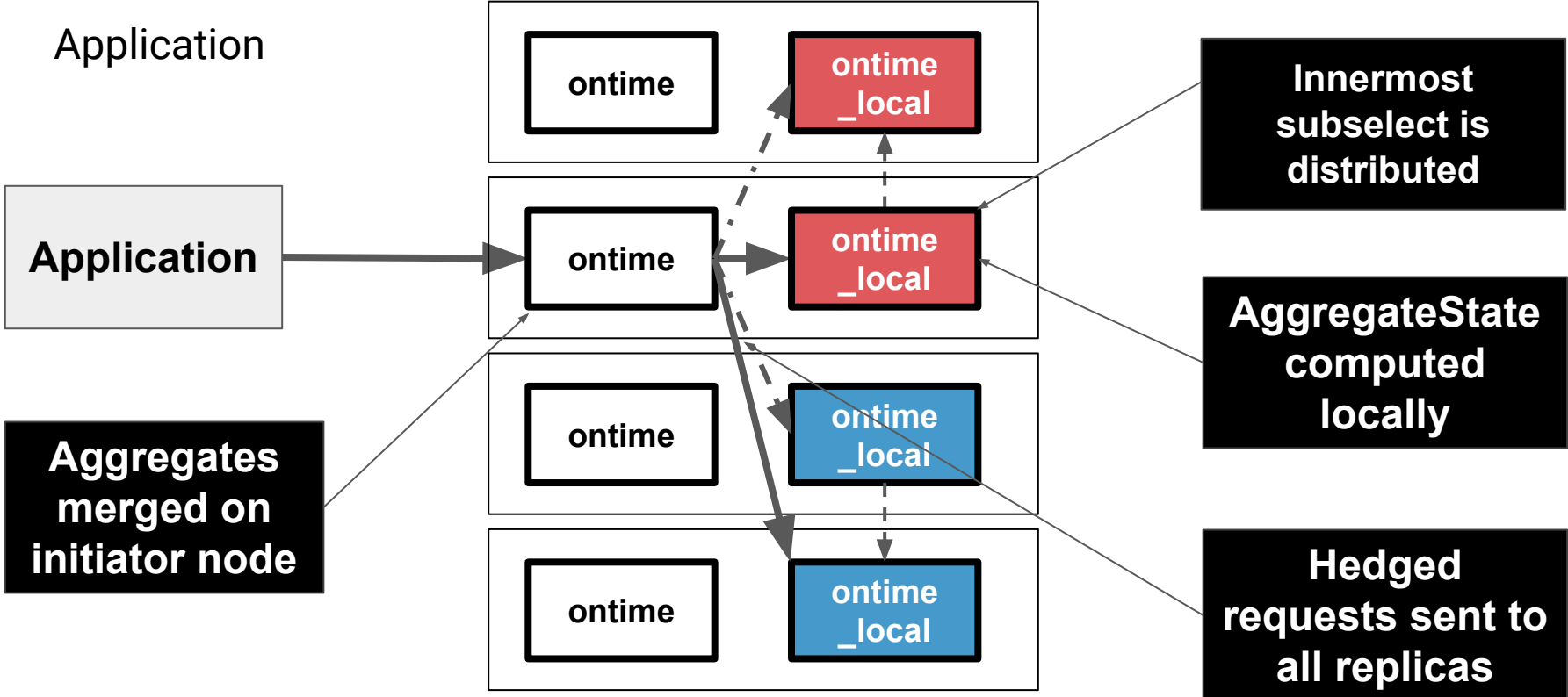
How does a distributed INSERT work?



Options for processing INSERTs

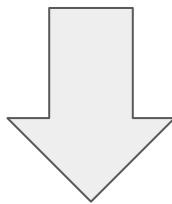
- Local vs distributed data insertion
 - INSERT to local table – no need to sync, larger blocks, faster
 - INSERT to Distributed table – sharding by ClickHouse
 - CHProxy -- distributes transactions across nodes, only works with HTTP connections
- Asynchronous (default) vs synchronous insertions
 - `insert_distributed_sync`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

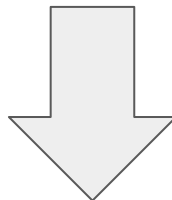


**-State for partial
aggregation**

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

ClickHouse pushes down JOINS by default

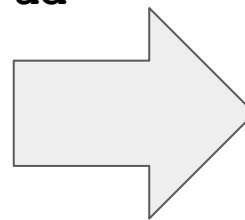
```
SELECT o.Dest d, a.Name n, count(*) c, avg(o.ArrDelayMinutes) ad
FROM default.ontime o
JOIN default.airports a ON (a.IATA = o.Dest)
GROUP BY d, n HAVING c > 100000 ORDER BY d DESC
LIMIT 10
```



```
SELECT Dest AS d, Name AS n, count() AS c, avgState(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
```



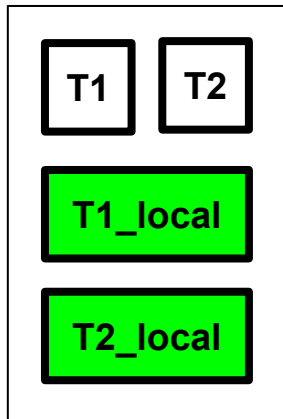
**Remote
Servers**

Distributed JOIN/IN is supported – distributed_product_mode

```
SELECT foo FROM T1 WHERE a IN (SELECT a FROM T2)  
  
SELECT foo FROM T1 JOIN T2
```



local

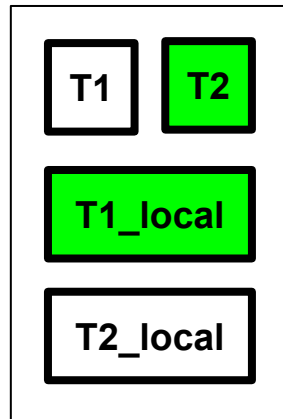


Right part runs
on local table

Mapping from initiator to local nodes



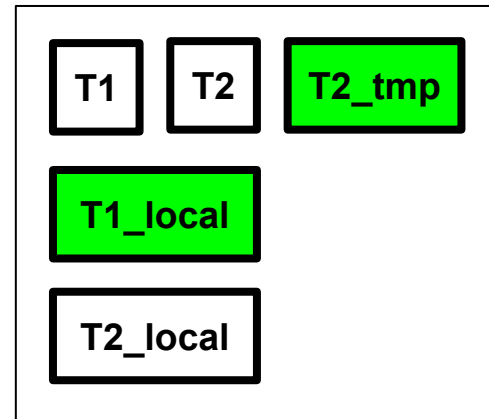
allow



Right part runs
on distributed



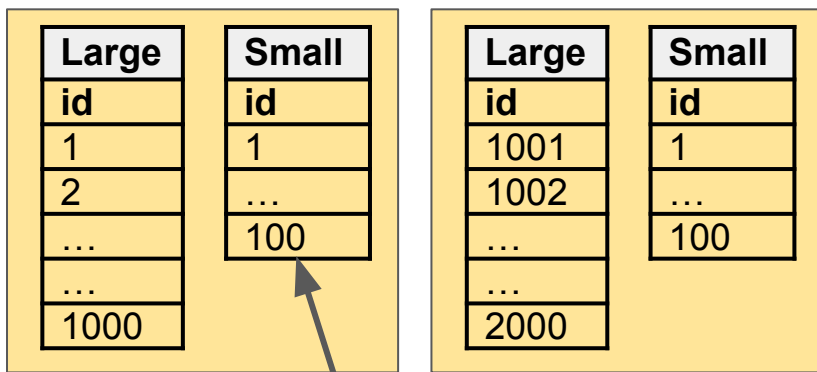
global



Initiator runs subquery
and broadcasts to nodes

Thinking about distributed data and joins

“Big Table Model”

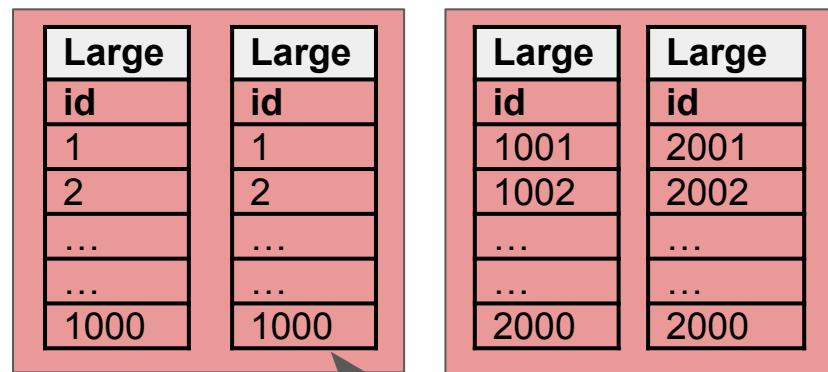


Shard 1

Shard 2

All keys in Small are replicated

“Bucketing Model”



Shard 1

Shard 2

Matching keys in each bucket

Parallel replicas (experimental)

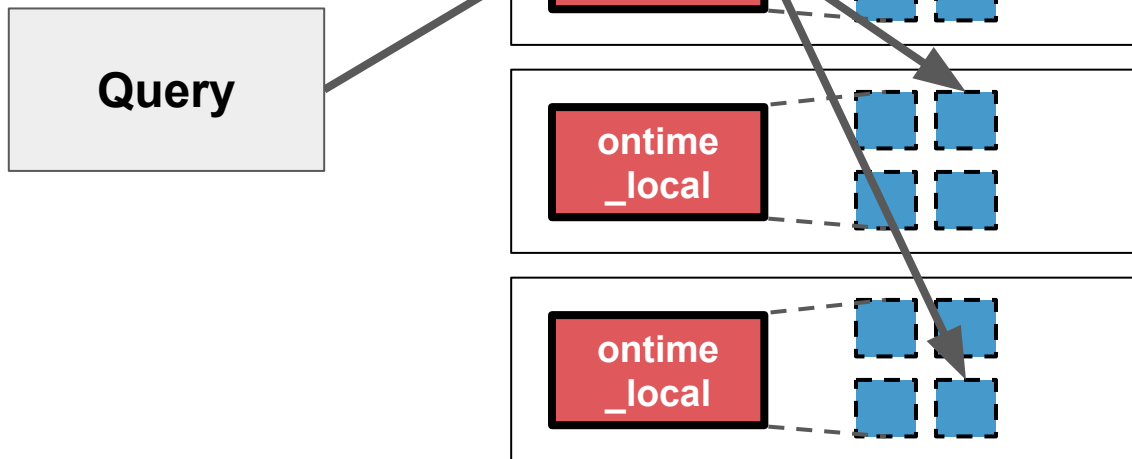


Table is sharded
'dynamically'

Every replica
server part of
the data

Requires good
sharding key

https://clickhouse.com/docs/en/operations/settings/settings#max_parallel_replicas

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

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

Free Consultation - <https://altinity.com/free-clickhouse-consultation/>

A close-up photograph of various mechanical components, including gears, shafts, and bearings, rendered in a dark, moody, and slightly blurred style. The lighting highlights the metallic textures and complex shapes of the machinery.

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!](#)