ClickHouse Data Management Internals

MergeTree Storage, Merges, and Replication

Tatiana Saltykova - Robert Hodges - Alexander Zaitsev

Let's make some introductions

Alexander Zaitsev

Altinity.Cloud architect and CTO

Robert Hodges

Database geek and Altinity CEO

Tatiana Saltykova

Altinity senior support engineer

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

ClickHouse is a SQL Data Warehouse

Understands SQL

Runs on bare metal to cloud

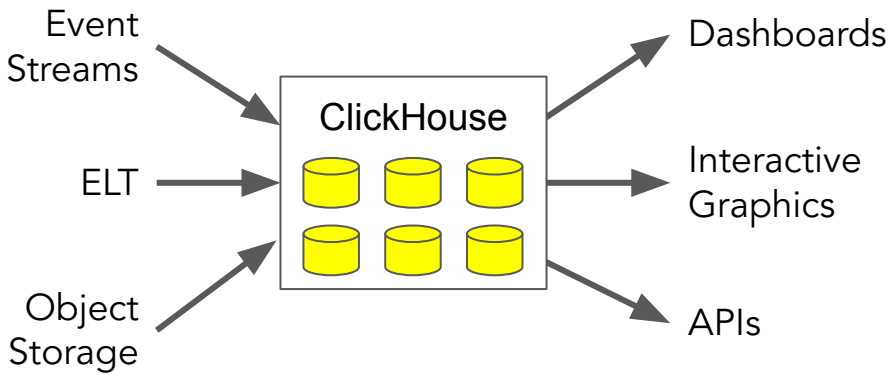
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



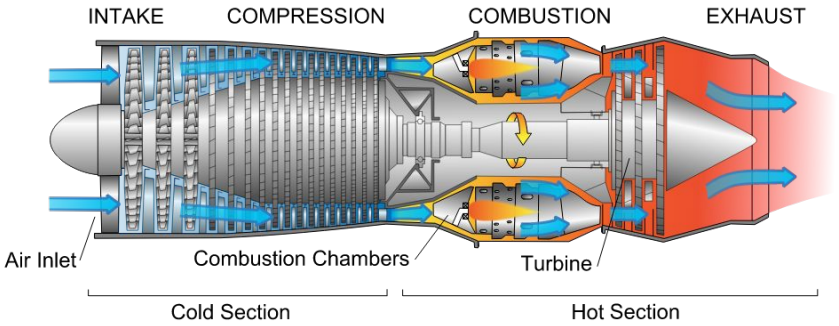
It's a popular engine for real-time analytics

If you understand the engine you can make it faster

ClickHouse has a simple execution model–there's no magic

Any developer can understand how it works

Knowledge leads to faster and more efficient queries

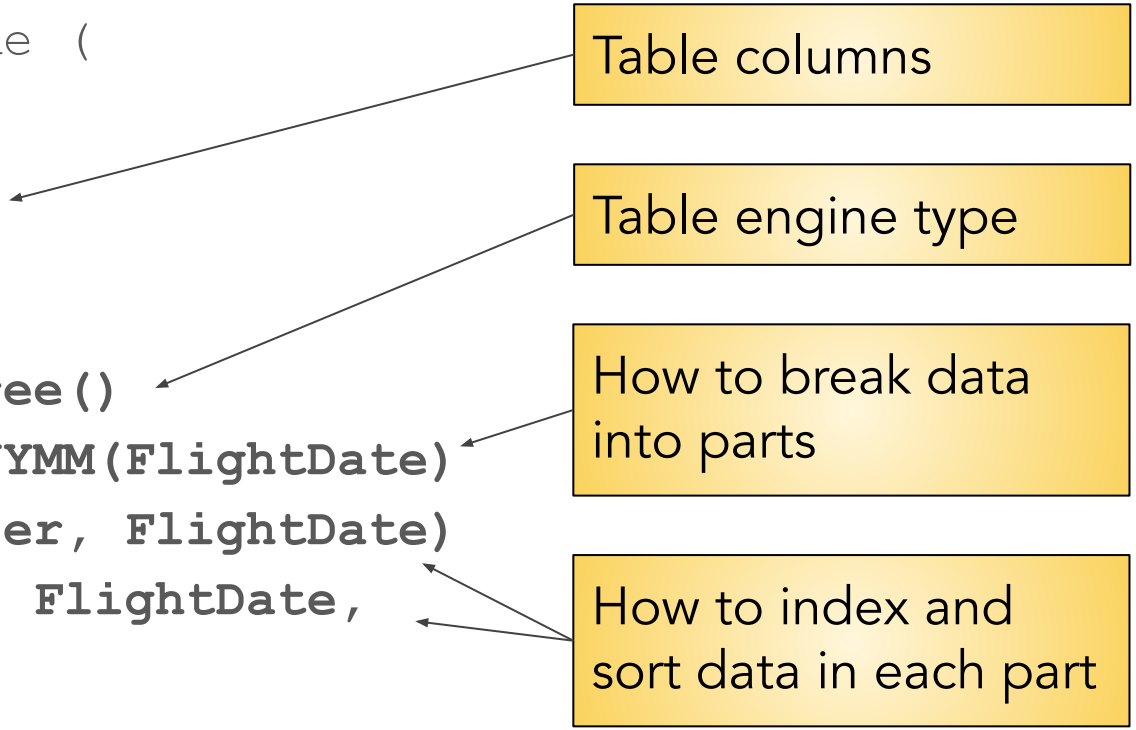


(Another fast engine!)

MergeTree Storage Organization

Table definition

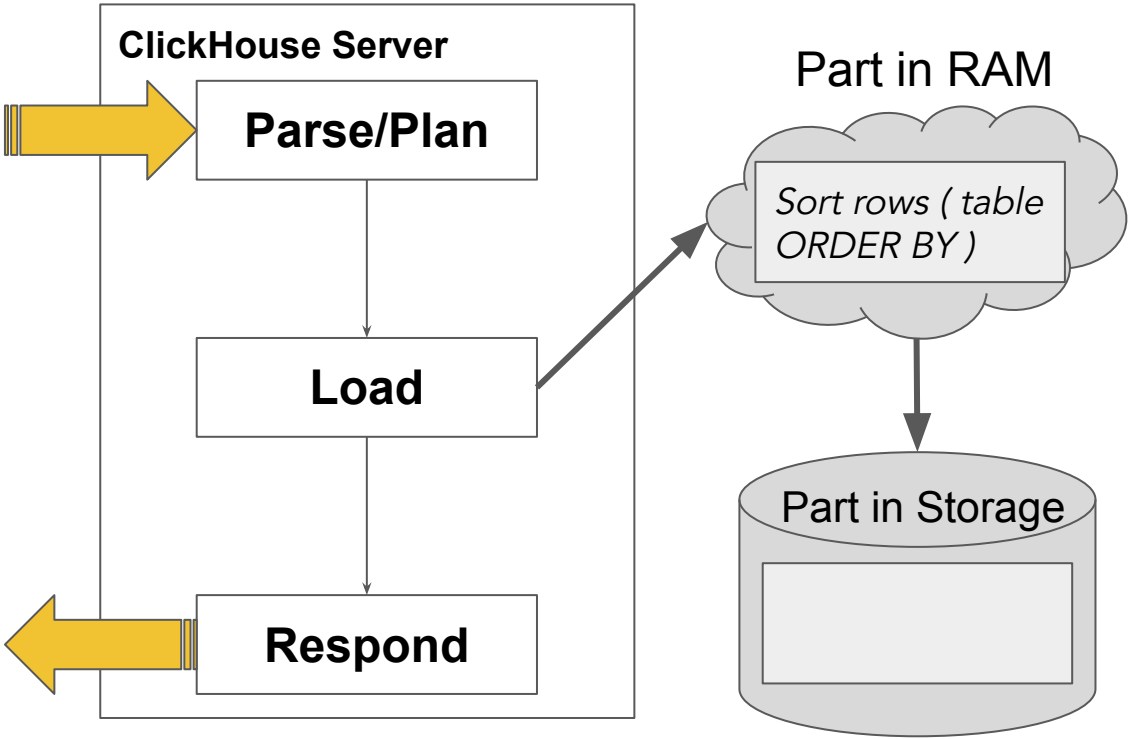
```
CREATE TABLE ontime (
   Year UInt16,
   Quarter UInt8,
   Month UInt8,
   ...
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(FlightDate)
PRIMARY KEY (Carrier, FlightDate)
ORDER BY (Carrier, FlightDate,
   DepTime)
```



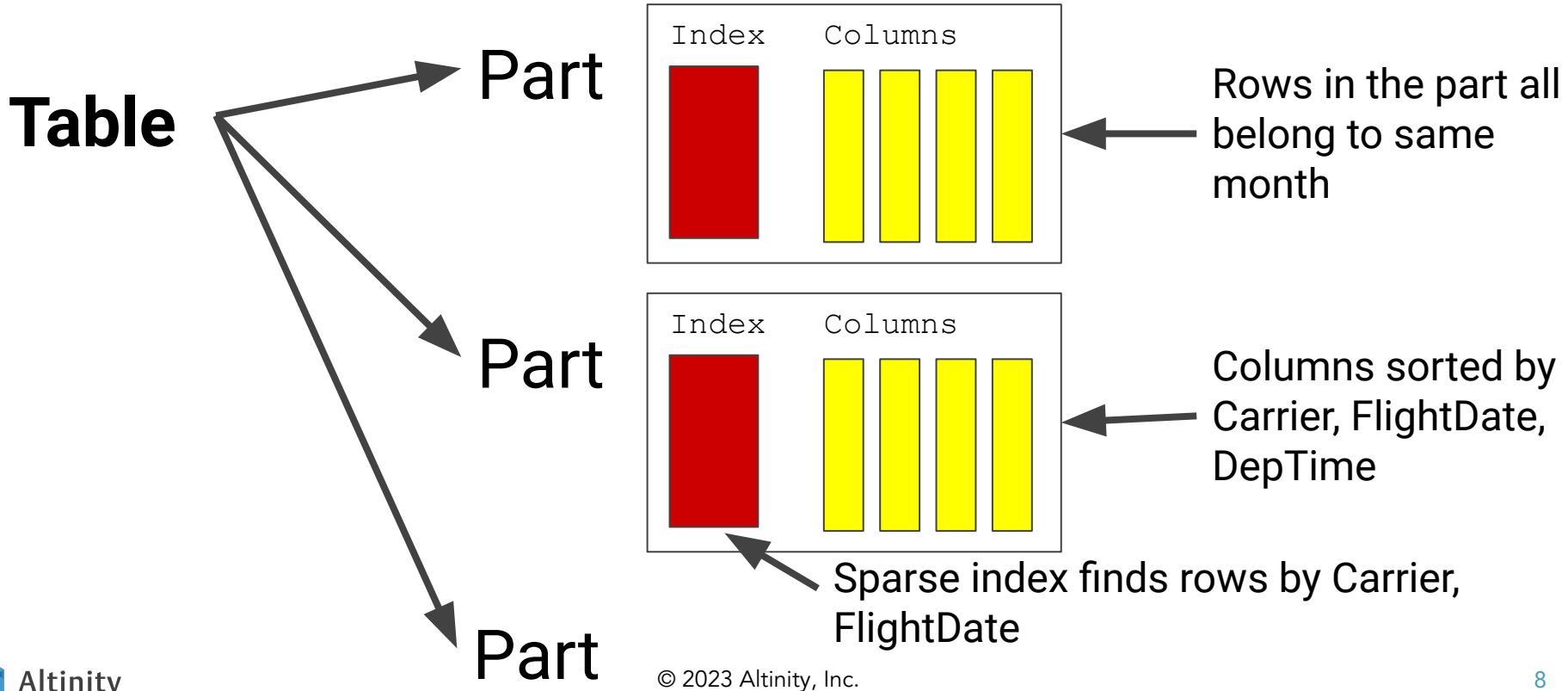
How does ClickHouse process an insert?

```
INSERT INTO ontime
VALUES
(15, 'TEMP', . . .),
(15, 'TEMP', . . .)
```

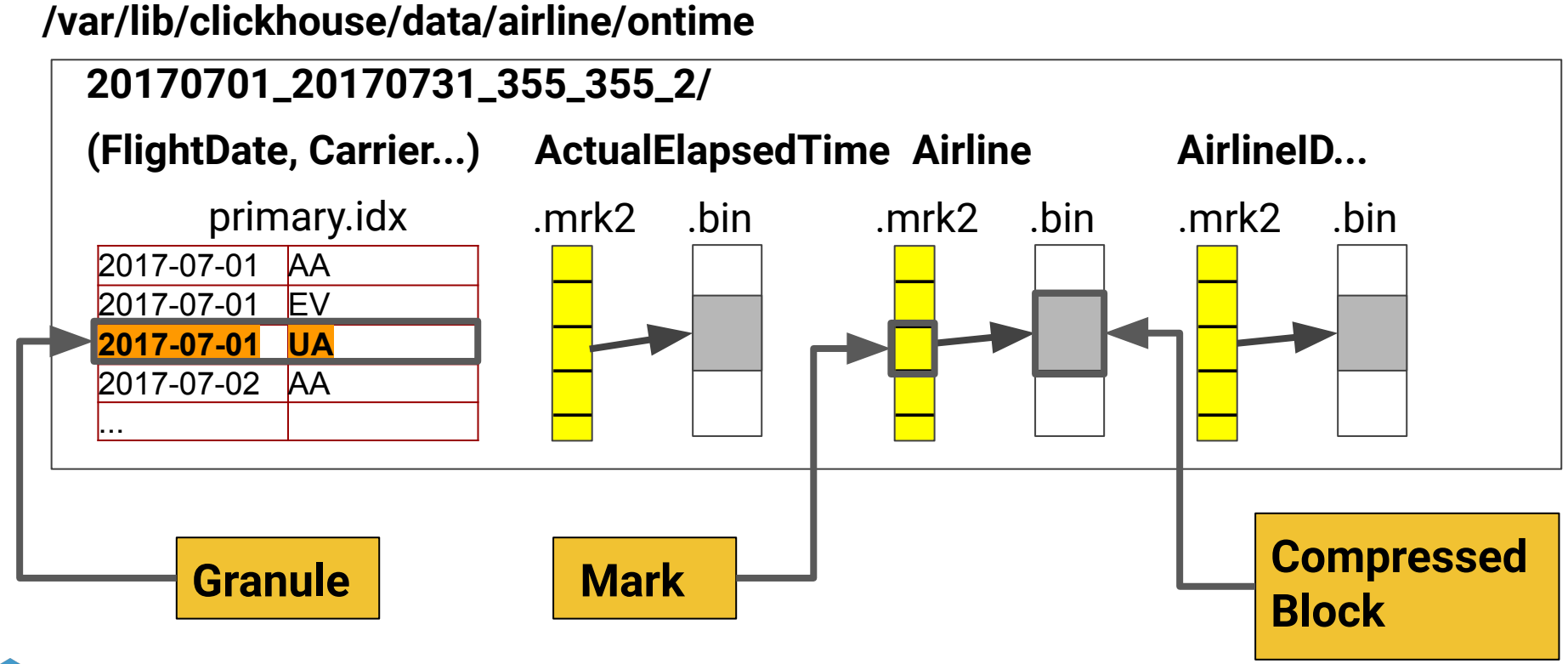
```
2 rows in set. Elapsed:
0.271 sec.
```



What's going on down there when you INSERT?



Understanding what's in a MergeTree part



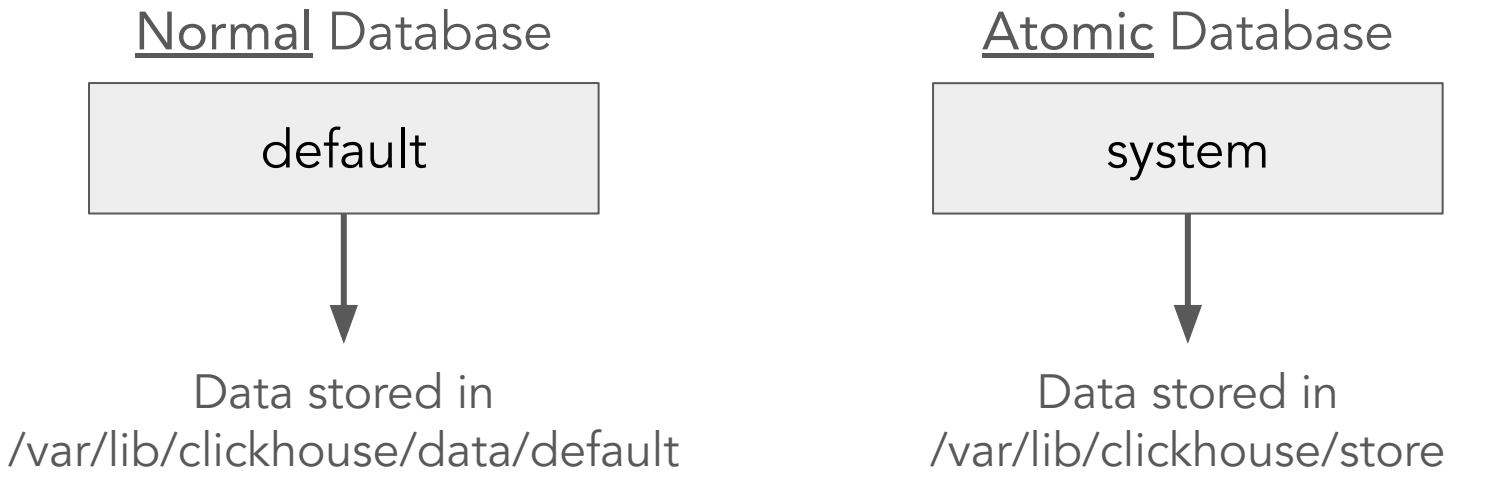
Finding data

```
ls -l /var/lib/clickhouse
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 access
drwxr-x---  5 clickhouse clickhouse  4096 May 30 12:32 data
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 dictionaries_lib
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 flags
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 format_schemas
drwxr-x---  5 clickhouse clickhouse  4096 May 30 12:32 metadata
drwxr-x---  2 clickhouse clickhouse  4096 May 30 12:40 metadata_dropped
drwxr-x---  2 clickhouse clickhouse 12288 May 26 13:51 preprocessed_configs
-rw-r-----  1 clickhouse clickhouse    60 May 26 13:49 status
drwxr-x--- 76 clickhouse clickhouse  4096 May 30 18:31 store
drwxr-x---  2 clickhouse clickhouse  4096 Jun  5 21:41 tmp
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 user_defined
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 user_files
drwxr-x---  2 clickhouse clickhouse  4096 May 26 13:49 user_scripts
-rw-r-----  1 clickhouse clickhouse    36 May 26 13:49 uuid
```

Atomic and Normal database engines

Original format in older ClickHouse versions

Can RENAME/EXCHANGE table atomically without moving data



See https://kb.altinity.com/engines/altinity-kb-atomic-database-engine/

Finding a Normal database

```
ls -ld /var/lib/clickhouse/metadata/default*
drwxr-x--- 2 … default
-rw-r----- 1 … default.sql

SELECT name, engine, data_path, uuid FROM system.databases
WHERE name = 'default' FORMAT Vertical

Row 1:
──────
name:      default
engine:    Ordinary
data_path: /var/lib/clickhouse/data/default/
uuid:      00000000-0000-0000-0000-000000000000
```

Finding an Atomic database

```
ls -ld /var/lib/clickhouse/metadata/system*
… /var/lib/clickhouse/metadata/system ->
/var/lib/clickhouse/store/e9f/e9fa7e20-0ea7-44d6-9adc-8c0842
46d5a3/
… /var/lib/clickhouse/metadata/system.sql

SELECT name, engine, data_path, uuid FROM system.databases
WHERE name = 'system' FORMAT Vertical
Row 1:
──────
name:      system
engine:    Atomic
data_path: /var/lib/clickhouse/store/
uuid:      e9fa7e20-0ea7-44d6-9adc-8c084246d5a3
```

Finding a table in an Atomic database

```
ls -l /var/lib/clickhouse/data/system/query_log
lrwxrwxrwx 1 … /var/lib/clickhouse/data/system/query_log ->
/var/lib/clickhouse/store/51c/51c33ee6-00f9-4cca-a543-05d1557f3768/

SELECT database, name, metadata_path, data_paths
FROM   system.tables
WHERE  database='system' and name like 'query_log' FORMAT Vertical

Row 1:
──────
database:      system
name:          query_log
metadata_path: /var/lib/clickhouse/store/e9f/e9fa7e20-0ea7-44d6-9adc-8c084246d5a3/query_log.sql
data_paths:    ['/var/lib/clickhouse/store/51c/51c33ee6-00f9-4cca-a543-05d1557f3768/']
```

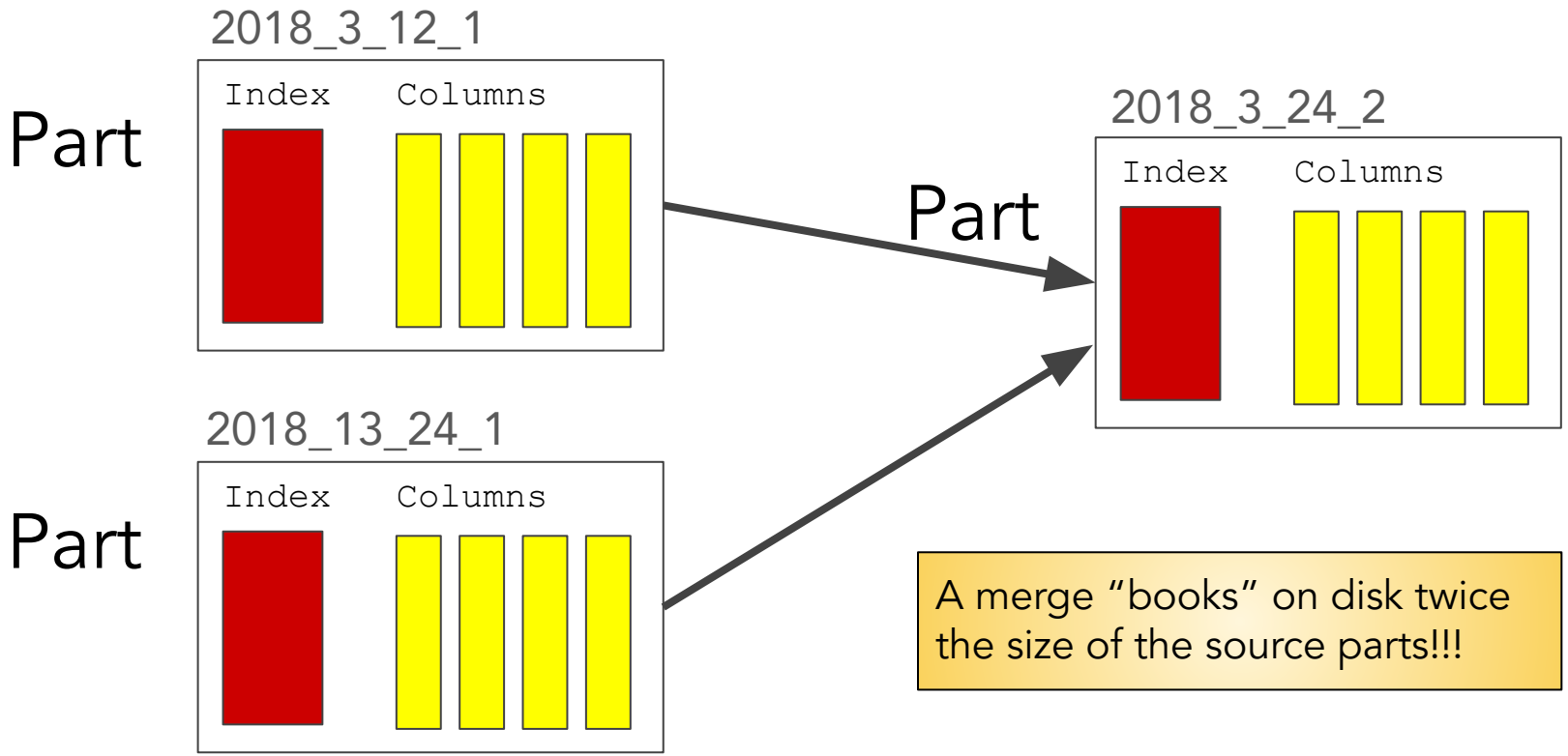
Find table data in an Ordinary database

```
ls -l /var/lib/clickhouse/data/default/ontime/
drwxr-x--- … 2018_3_39_2
drwxr-x--- … 2019_1_40_2
drwxr-x--- … detached
-rw-r----- … format_version.txt

SELECT name, rows, bytes_on_disk, path FROM system.parts WHERE
database = 'default' AND table = 'ontime' FORMAT Vertical

name:          2018_3_39_2
rows:          7213446
bytes_on_disk: 582315096
path:          /var/lib/clickhouse/data/default/ontime/2018_3_39_2/
```

Why MergeTree? Because it merges!



Merging parts: how it works

1. MergeSelector
    - periodically scans all tables
    - chooses parts that need to be merged according to its algorithm
    - schedules merges
2. MergeExecutor
    - picks a merge and runs it

system.merges

- Progress for running merges

system.part_log

- Statistics for merges

What's on disk?

```
ls -l /var/lib/clickhouse/data/default/ontime/
total 548
drwxr-x--- … 2018_15_15_0
drwxr-x--- … 2018_15_26_1
drwxr-x--- … 2018_16_16_0
…
drwxr-x--- … detached
-rw-r----- … format_version.txt

SELECT name, active, level, rows, bytes_on_disk
FROM system.parts WHERE database = 'default' AND table = 'ontime'

┌─name────────┬─active─┬─level─┬──rows─┬─bytes_on_disk─┐
│ 2018_15_15_0 │      0 │     0 │ 393216 │      31947995 │
│ 2018_15_26_1 │      1 │     1 │ 2359114 │     195292349 │
│ 2018_16_16_0 │      0 │     0 │ 393216 │      32867527 │
│ 2018_19_19_0 │      0 │     0 │ 393034 │      35711823 │
…
```

Manipulating parts and mutating data

Commands to manipulate partitions/parts/files

- DROP PARTITION|PART – Deletes a partition or a part.
- DETACH PARTITION|PART – Moves a partition or a part to the detached directory and forget it.
- ATTACH PARTITION|PART – Adds a partition or a part to the table from the detached directory or from another table.
- REPLACE PARTITION – Copies a partition from another table and replaces the existing one (the existing data is lost).
- MOVE PARTITION TO TABLE – Moves a partition from one table to another.
- CLEAR COLUMN IN PARTITION – Resets the value of a specified column in a partition.
- CLEAR INDEX IN PARTITION – Resets the specified secondary index in a partition.
- FETCH PARTITION|PART – Downloads a partition or a part from another server into the detached directory.
- MOVE PARTITION|PART – Moves a partition or a part to another disk or volume.

Identifying partition IDs and part names

```
SELECT database, table, partition, partition_id, name
FROM system.parts WHERE database='default' AND table='ontime'
```

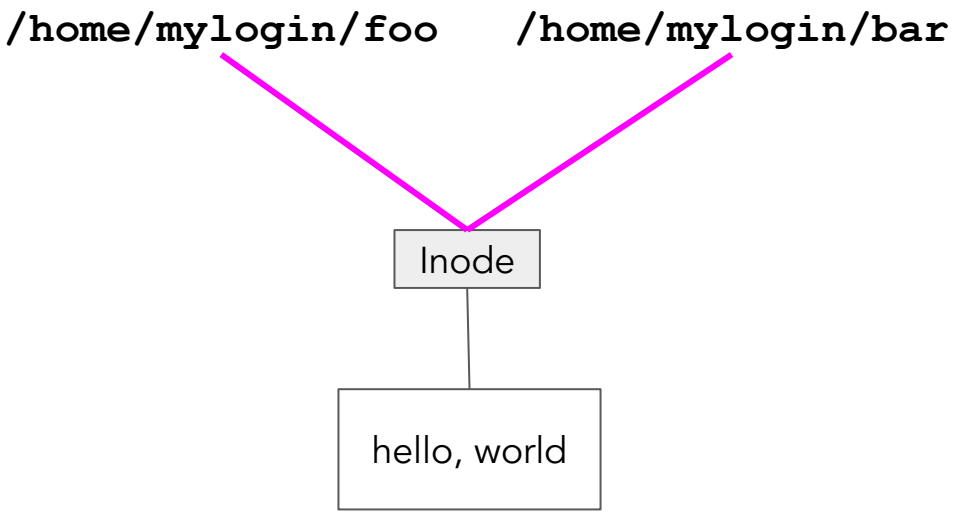
```
┌─database─┬─table──┬─partition─┬─partition_id─┬─name─────────┐
│ default  │ ontime │ 2018      │ 2018         │ 2018_3_12_1  │
│ default  │ ontime │ 2018      │ 2018         │ 2018_15_26_1 │
…
```

```
ALTER TABLE ontime DETACH PART '2018_15_26_1'

ALTER TABLE ontime DROP PARTITION ID '2018'
```

Quick primer on hard links

```
$ echo "hello, world" > foo
$ ln foo bar
$ ls --inode foo bar
4206300 bar  4206300 foo
$ cat bar
hello, world
$ rm foo
$ ls --inode bar
4206300 bar
```

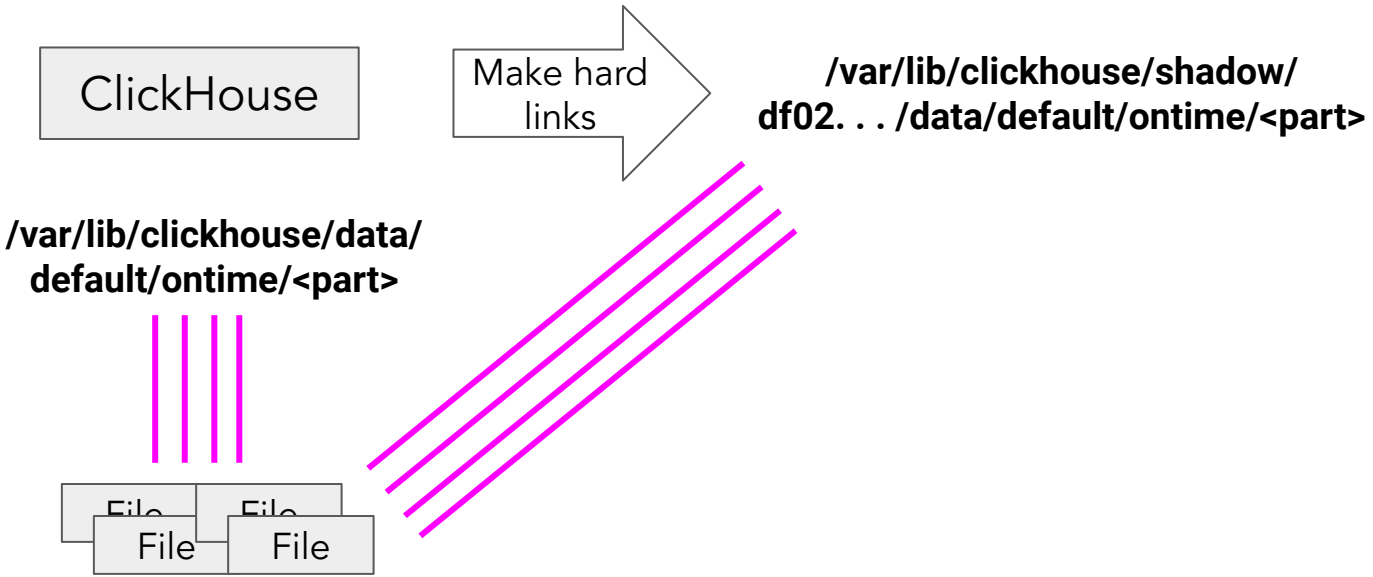


Tip: Cross device and remote hard links are not possible. Hard links only work within a single file system.

Using ALTER TABLE FREEZE to get a stable copy of data

```
ALTER TABLE default.ontime
FREEZE with name 'df02...';
```

```
ALTER TABLE default.ontime
UNFREEZE WITH NAME 'df02...'
```

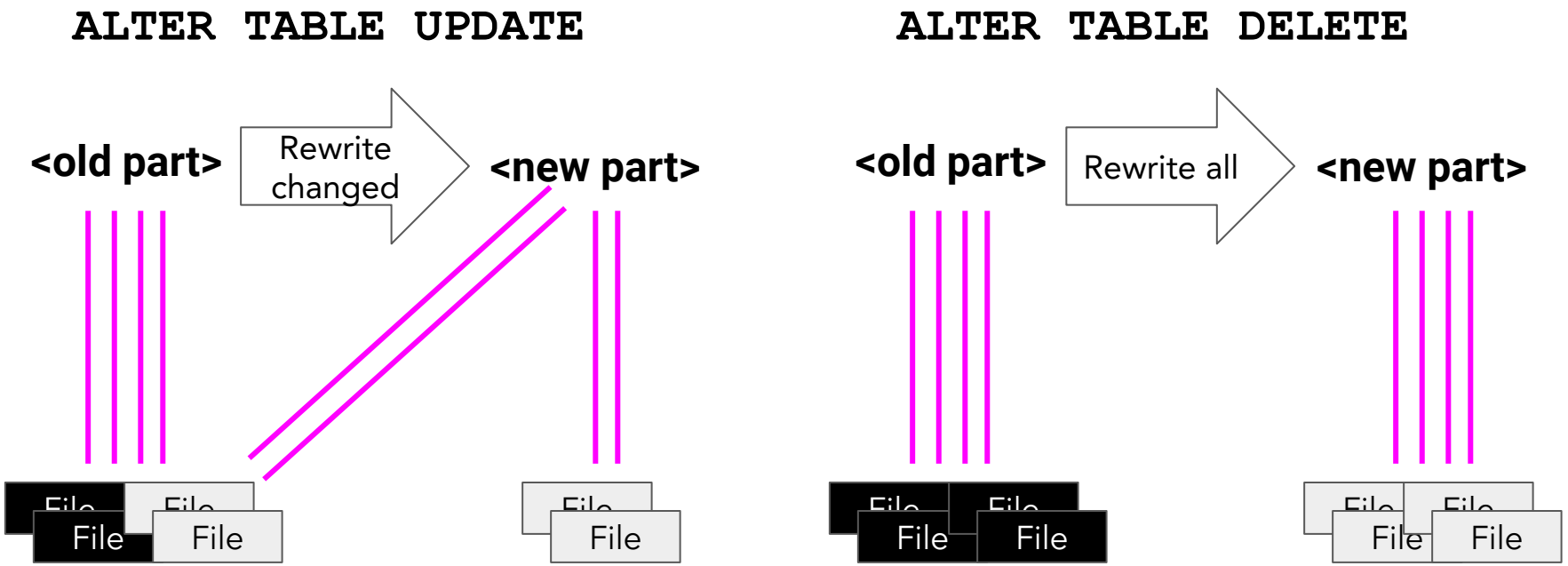


Want to change data? Run a mutation!

Mutations run asynchronously and can be very heavy

- Mutate all columns' files
  - ALTER TABLE … DELETE
  - ALTER TABLE … MATERIALIZE TTL
- Mutate one or several columns' files
  - ALTER TABLE … UPDATE
  - ALTER TABLE … MATERIALIZE COLUMN
  - ALTER TABLE … MODIFY COLUMN (change data type)
  - DELETE FROM … (lightweight delete)
- Drop old and/or create new files
  - ALTER TABLE … MATERIALIZE INDEX/PROJECTION
  - ALTER TABLE … DROP/RENAME/CLEAR COLUMN

Storage level differences between DELETE and UPDATE



What is happening with my mutation?

```
-- Use system.merges to see if your mutation is running
SELECT * FROM system.merges
WHERE is_mutation

-- Use system.mutations to check the status
SELECT * FROM system.mutations
WHERE NOT is_done

-- Use system.mutations to find out if mutations are failing
SELECT * FROM system.mutations
WHERE latest_fail_time > toDateTime(0)

-- Use system.mutations to KILL mutations
KILL MUTATION
WHERE database='db1' AND table='table1' AND mutation_id='0000000001'
```

Some Common MergeTree Discontents

(And solutions to them!)

Common problems: Too many parts

Caused by a high number of parts in a single partition

Why: merges cannot keep up with inserts

system.merge_tree_settings

- Parts to delay/reject
- Prioritize smaller merges

system.settings

- Parts to delay/reject
- Async inserts

system.part_log

- Parts created by a query
- https://kb.altinity.com/altinity-kb-useful-queries/ingestion-rate-part_log/

Common problems: Long start time

Caused by a high total number of parts

Why:

1. A lot of tables
2. Small partitions

system.merge_tree_settings

- max_parts_in_total = 100000 by default
- max_part_loading_threads (up to 23.5)

system.server_settings

- 23.6+ max_active_parts_loading_thread_pool_size

Common problems: Too many mutations

Caused by underestimating mutations' effect on the system

Why:

1. ALTER statements return OK immediately
2. Mutations are invisible to the user because they run in background

system.merge_tree_settings

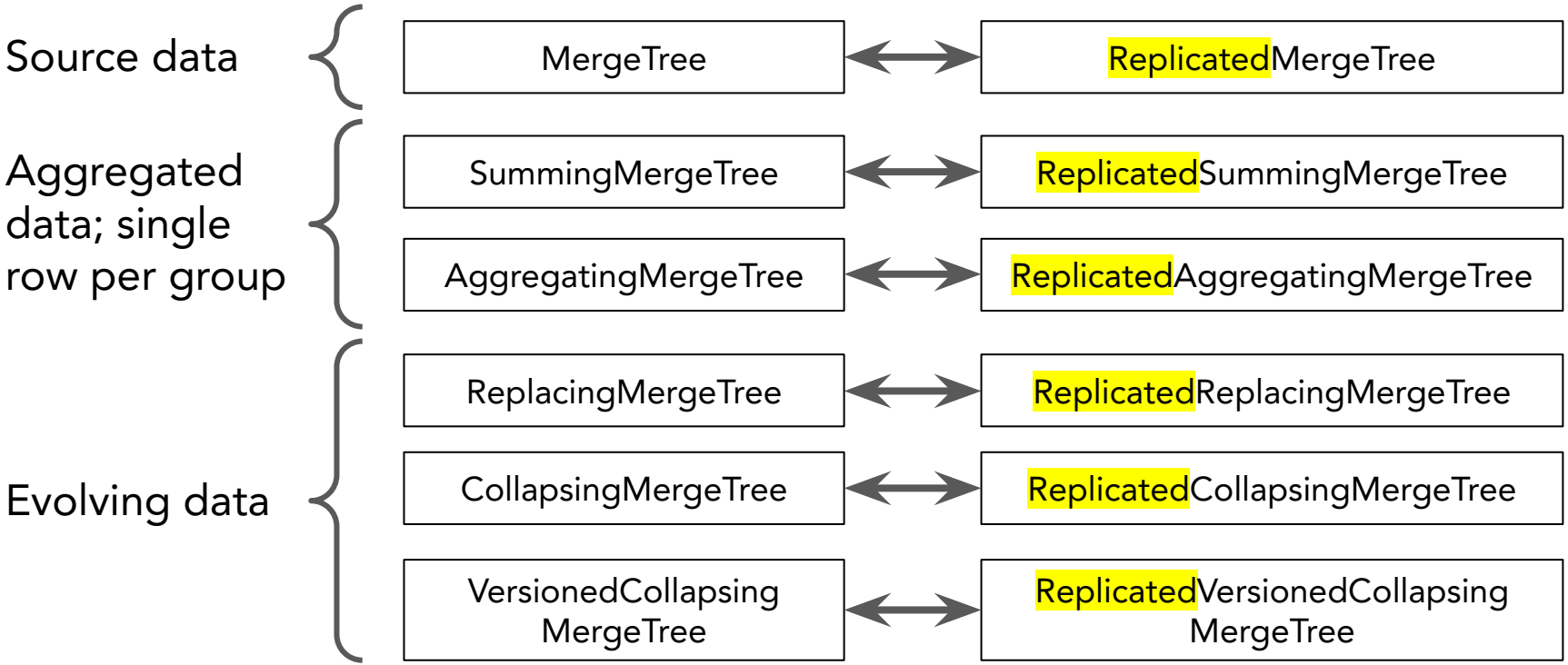
- Concurrency settings (ex. number_of_free_entries_in_pool_to_execute_mutation)
- 23.5+ Backoff settings (ex. number_of_mutations_to_delay)

system.settings (23.3+ system.server_settings)

- Background merges pool (ex. background_pool_size)
- Concurrency settings (ex. background_merges_mutations_concurrency_ratio)

Replication and Distributed DDL

MergeTree tables support replication

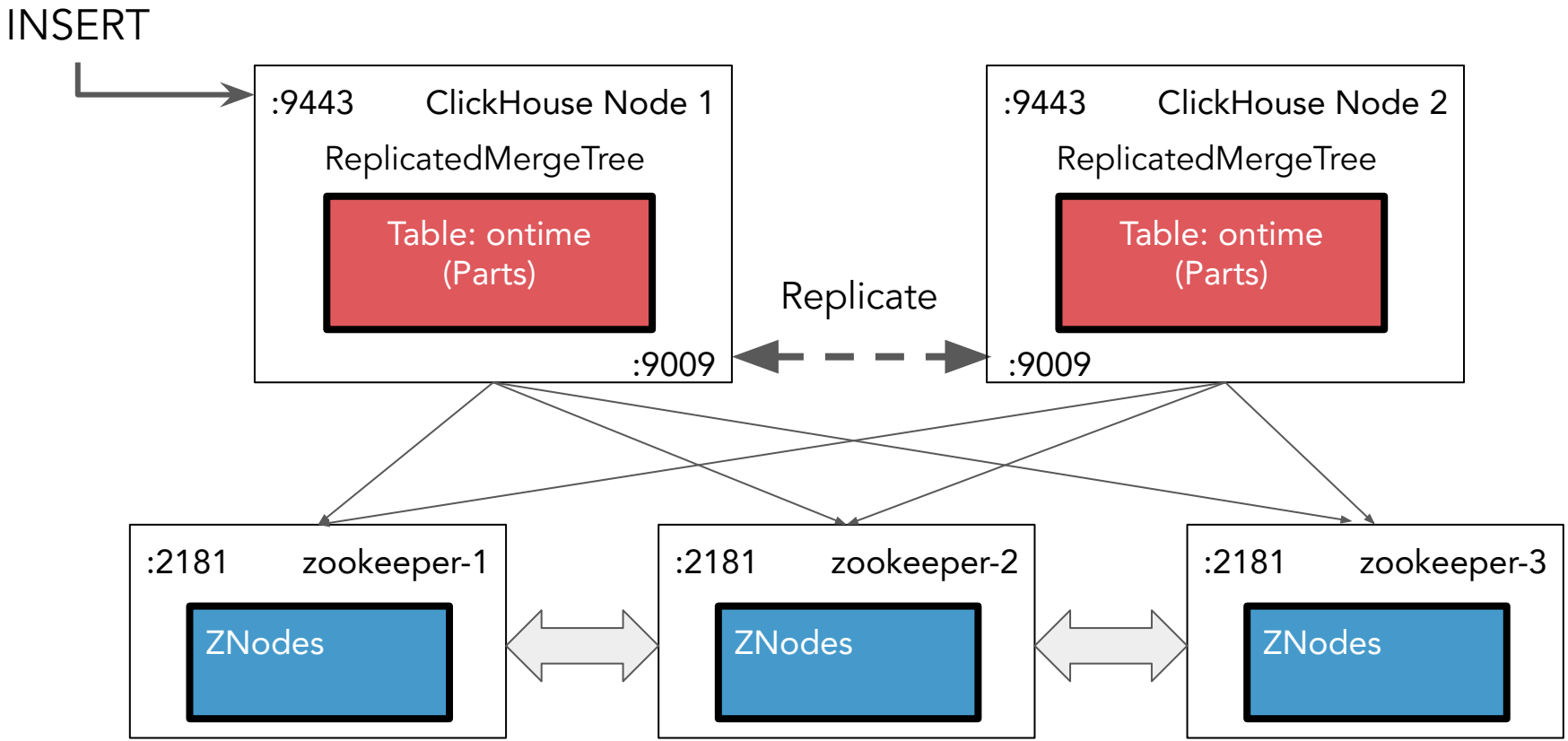


Example of a sharded, replicated fact table

```
CREATE TABLE IF NOT EXISTS `ontime_local`
AS default.ontime_ref
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!

How replication works

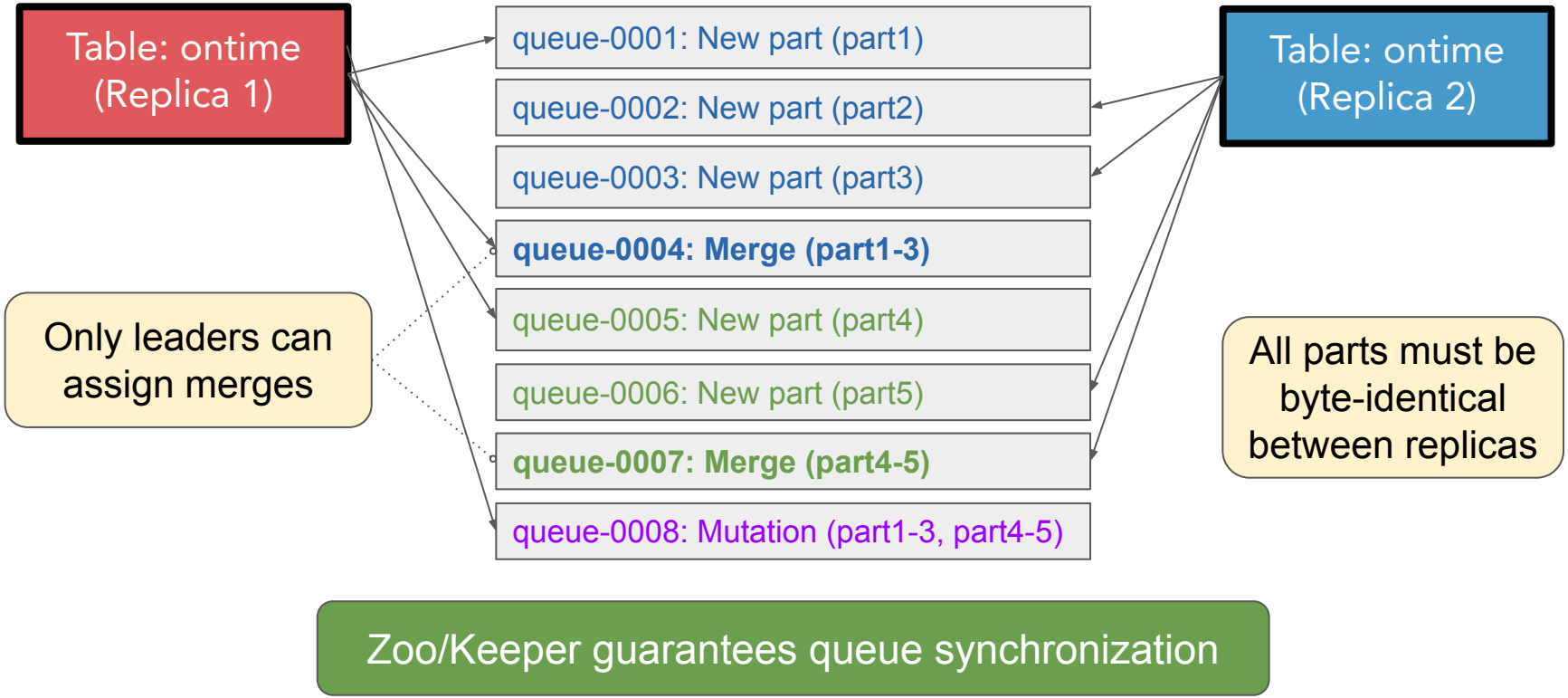


What is replicated?

Replicated*MergeTree ONLY

Replicated statements	Not replicated statements
• INSERT • ALTER TABLE exceptions: FREEZE, MOVE TO DISK, FETCH, some settings • OPTIMIZE • TRUNCATE	• CREATE table • DROP table • RENAME table • DETACH table • ATTACH table

Replication is asynchronous but sequential



What's cooking in the replication queue?

```
select * from system.replication_queue limit 1 format Vertical
```

```
Row 1:
──────
database:                default
table:                   my metric log
replica name:            chi-github-github-0-0
position:                4
node name:               queue-0002314922
type:                    MERGE_PARTS
create time:             2023-07-22 16:26:33
required quorum:         0
source replica:          chi-github-github-0-1
new part name:           20230722_4038_4041_1
parts_to_merge:
['20230722_4038_4038_0','20230722_4039_4039_0','20230722_4040_4040_0','20230722_4041_4
041_0']
is_detach:               0
…
```

https://clickhouse.com/docs/en/operations/system-tables/replication_queue

Information about state of replicas

```
select * from system.replicas where database=currentDatabase() and
table='my_metric_log'
```

```
database:                       default
table:                          my_metric_log
...
future_parts:                   0
parts_to_check:                 0
columns_version:                -1
queue_size:                     0
inserts_in_queue:               0
merges_in_queue:                0
part_mutations_in_queue:        0
queue_oldest_time:              1970-01-01 00:00:00
inserts_oldest_time:            1970-01-01 00:00:00
merges_oldest_time:             1970-01-01 00:00:00
…
```

What does ON CLUSTER do?

ON CLUSTER executes a command over a set of nodes. It uses the DDL queue.

```
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} ...
```

P.S. The experimental Replicated database engine takes care of schema synchronization

DDLWorker

- Runs as a thread in the ClickHouse process
- Cannot be restarted separately
- Coordinates through Zoo/Keeper (config.xml:`distributed_ddl.path`)
- Deletes old completed tasks from Zoo/Keeper periodically

```
CREATE TABLE table_name ON CLUSTER cluster_name …
```

What happens?

1. The initiator registers the task in Zoo/Keeper and the task is assigned a unique number
2. The task is configured according to the cluster definition (shards and replicas)
3. The initiator waits for the nodes to complete the task (`distributed_ddl_task_timeout`)
4. The initiator reports the result (how to troubleshoot)

The statement itself runs in background!!!

Distributed DDL queue

1. The nodes (including the initiator) process their queue independently and asynchronously
2. Depending on the task it is executed by one replica of every shard or by all nodes
3. All statements are executed in order
4. Some statements can take a long time to process
5. In case of an exception the node will retry 3 times for an hour (hardcoded)
6. Until the current statement is processed the node does not move on to the next statement

Troubleshooting:

```
SELECT * FROM system.distributed_ddl_queue
SELECT * FROM system.zookeeper where path='/clickhouse/task_queue/'

select hostname(), * from clusterAllReplicas('{cluster}',system.metrics)
where metric in ('MaxDDLEntryID', 'MaxPushedDDLEntryID')
```

What happens if DDL gets stuck?

1. Wait.
2. Wait.
    a. Check system.distributed_ddl_queue
3. Remove task(s) from [Zoo]Keeper

Wrap-up and where to find more information

There is *much* more to learn about ClickHouse

- Compact vs. wide storage parts (what's the difference?)
- Deeper dive into files (columns.txt, checksums.txt, etc.)
- Deeper dive into column structure (blocks, markets, compressed marks)
- Storage policies (tiered storage)
- MergeTree and S3 object storage
- How queries execute and access storage
- Caching!!

Shameless plug!

Sign up for the Altinity class on ClickHouse Administration!

Learn how ClickHouse works inside and how to administer it

- MergeTree innards
- Resource management
- Cluster management
- And more…

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/

ClickHouse code - https://github.com/ClickHouse/ClickHouse

"Use the source, Luke!"

Thank you!
Questions?

https://altinity.com

Altinity.Cloud
Altinity Stable Builds
Altinity Kubernetes Operator for ClickHouse