S3 Storage and ClickHouse®

Basic and Advanced Wizardry

Robert Hodges - Altinity CEO
Alexander Zaitsev - Altinity CTO

A brief message from our sponsor…

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Alexander Zaitsev

Expert in high scale analytics systems design and implementation. Altinity CTO

ClickHouse support and services: Altinity.Cloud and Altinity Stable Builds
Authors of Altinity Kubernetes Operator for ClickHouse

How Does S3 Fit into ClickHouse

Meet ClickHouse. It's a real-time analytic database

Understands SQL

Runs on bare metal to cloud

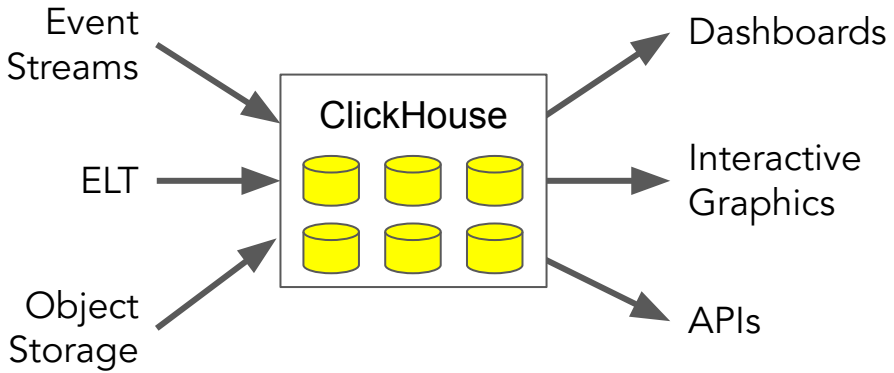
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's the core engine for low-latency analytics

Let's start with an ordinary ClickHouse table

```
CREATE TABLE test (
    `A` Int64,
    `S` String,
    `D` Date
)
ENGINE = MergeTree
PARTITION BY D
ORDER BY A;

INSERT INTO test
SELECT number, number,
'2023-01-01' FROM numbers(1e8);
```

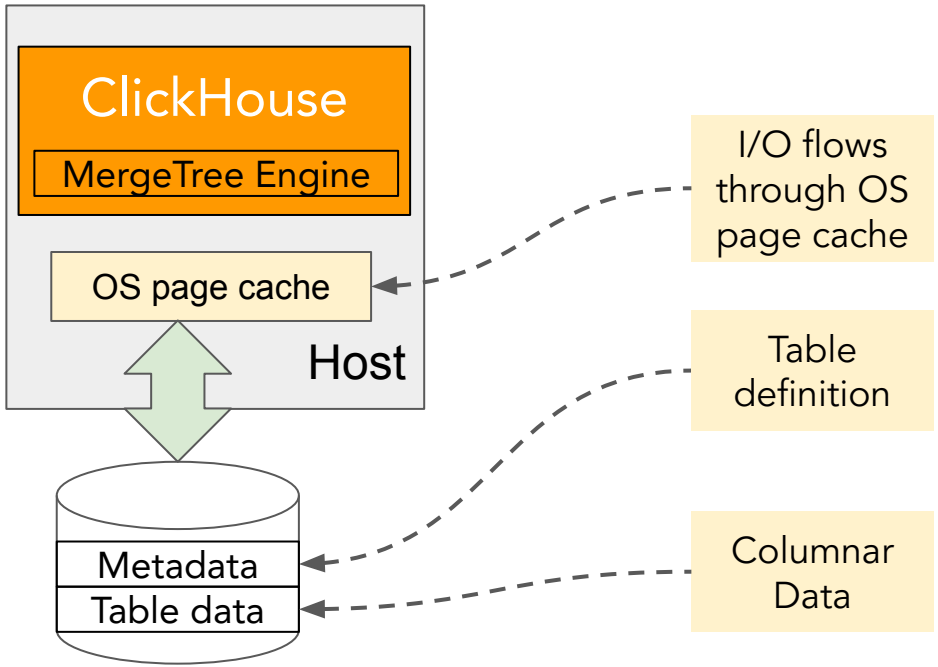
Use MergeTree for "big data"

Divide table into parts by day

Order data in parts by A value

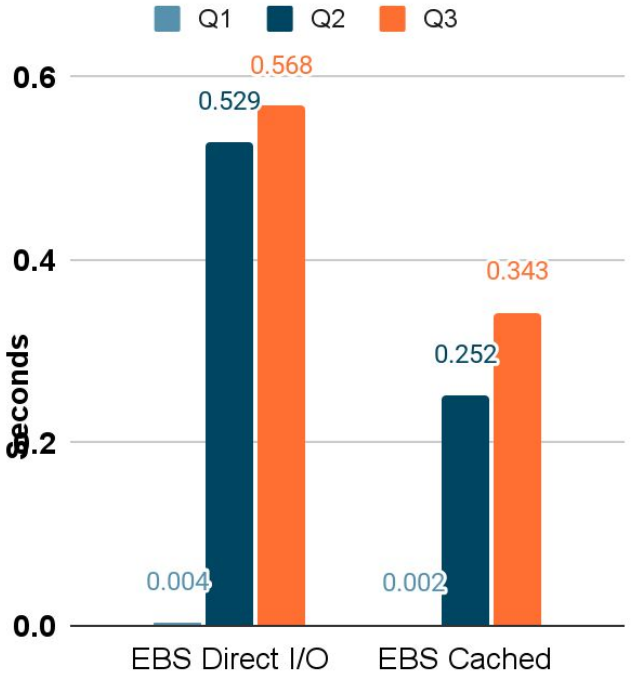
Load 100M rows of test data

Here's where the data goes in shared-nothing storage



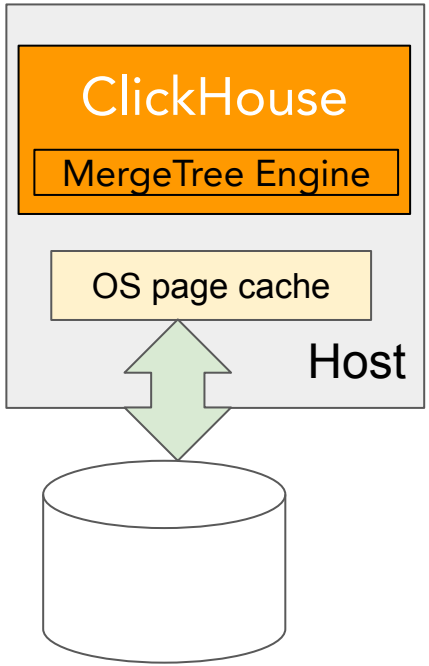
Performance is pretty good!

```
Q1: SELECT *
FROM test_s3_tiered WHERE A =
443

Q2: SELECT uniq(A)
FROM test_s3_tiered;

Q3: SELECT count()
FROM test_s3_tiered
WHERE S LIKE '%4422%'

[SETTINGS
min_bytes_to_use_direct_io=1]
```



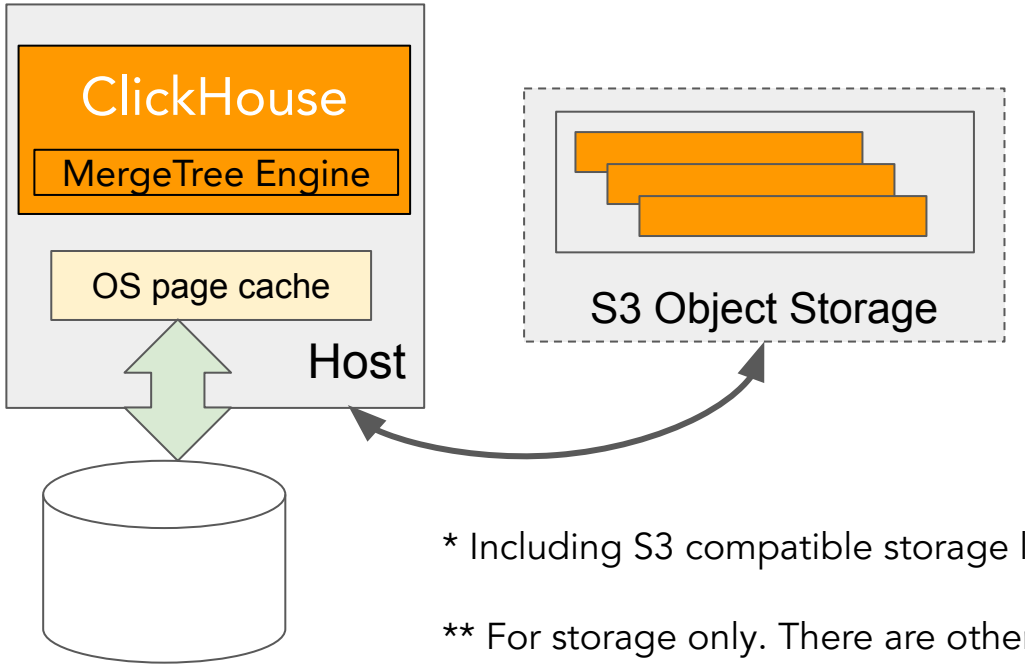
ClickHouse 23.7.4.5; M6i.2xlarge; EBS gp3

Shared nothing has limitations as data size grows



- X Limited capacity per server (< 50 Tb)
- X Block storage is expensive
- X Requires multiple copies when replicated

S3 storage* is a welcome addition to ClickHouse

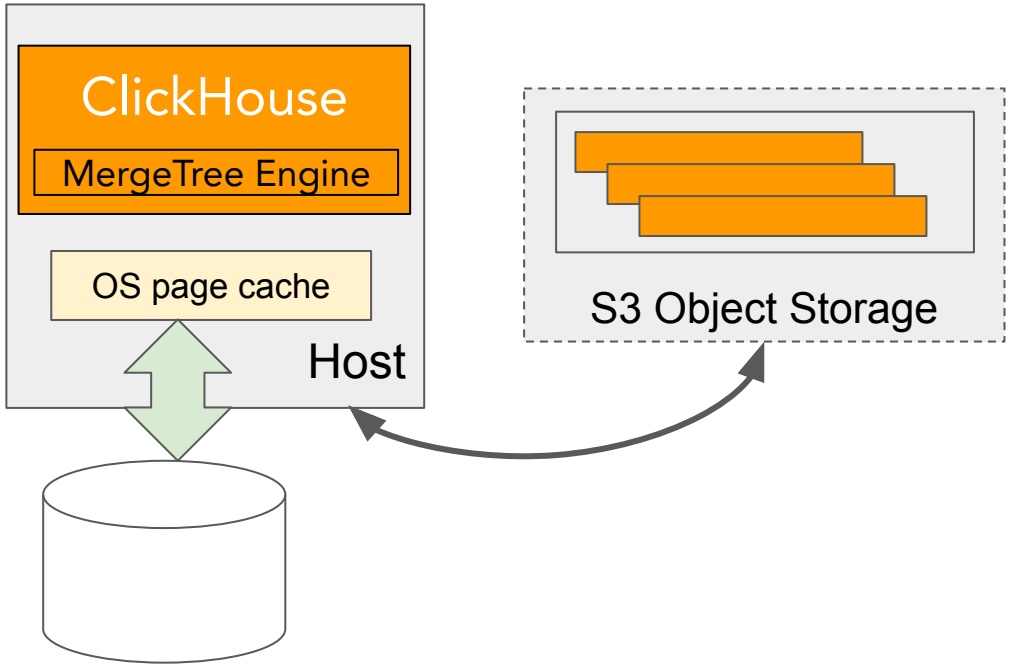


- ✅ Unlimited Capacity
- ✅ 4-5x Cheaper than EBS**
- ✅ Shareable
- ✅ Data lakes

* Including S3 compatible storage like GCS, MinIO, and Ceph

** For storage only. There are other costs that make it higher

But it’s not all peaches and cream



- X Very different API
- X Can’t update files
- X No automated cache
- X Extra cost for traffic and API calls

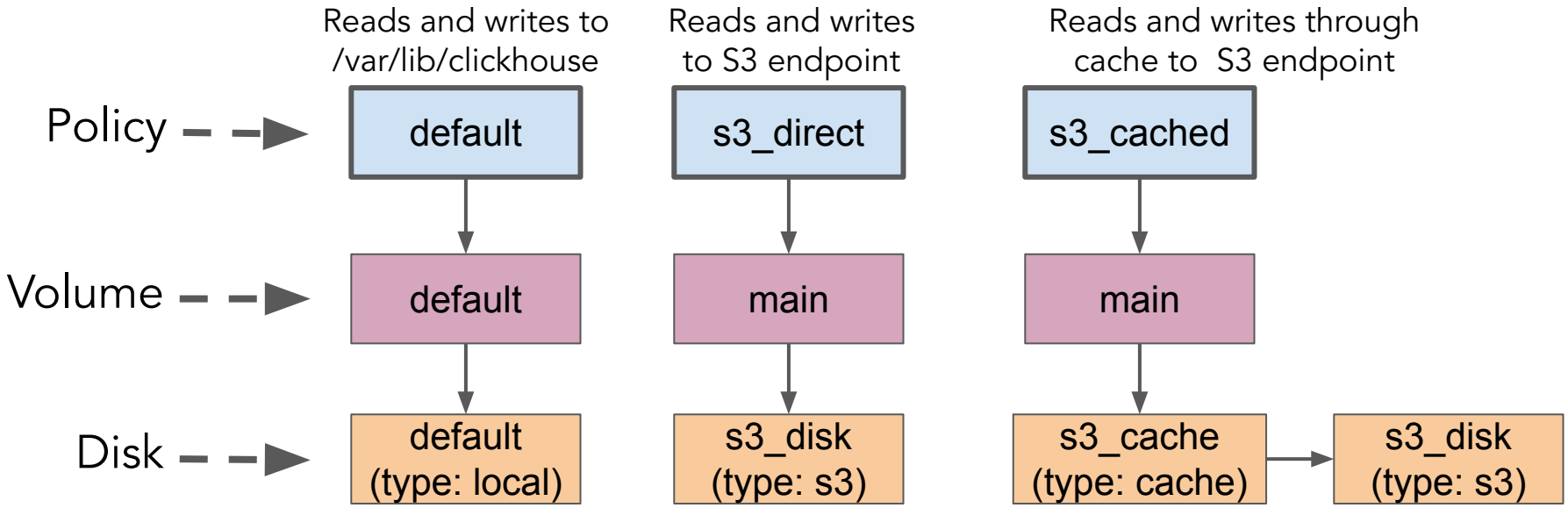
Using S3 for MergeTree Storage

ClickHouse Configuration for S3

You need to configure multiple things:

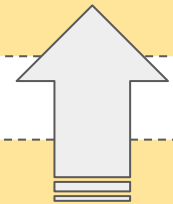
- “Disk” to specify how to access S3 bucket
- Optionally another “disk” to configure cache
- “Volume” and “Storage Policy” that specifies how S3 disk is used
- Attach policy to MergeTree table

Storage configurations organize storage into policies

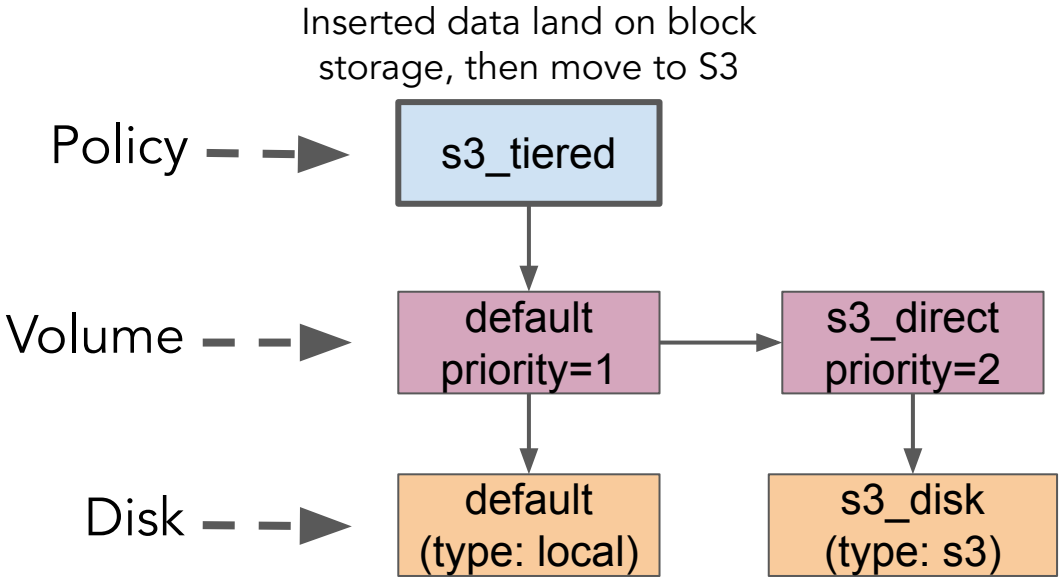


Example of storage configuration definition

```
<clickhouse> <storage_configuration> <disks>
    <s3_disk>
      <type>s3</type>
      <endpoint>https://s3.us-west-2.amazonaws.com/bucket/</endpoint>
      . . . # credentials here
      <metadata_path>/var/lib/clickhouse/disks/s3_disk/</metadata_path>
    </s3_disk>
  </disks>
  <policies>
    <s3_direct>
      <volumes>
        <main> <disk>s3_disk</disk> </main>
      </volumes>
    </s3_direct>
  </policies>. . .
```



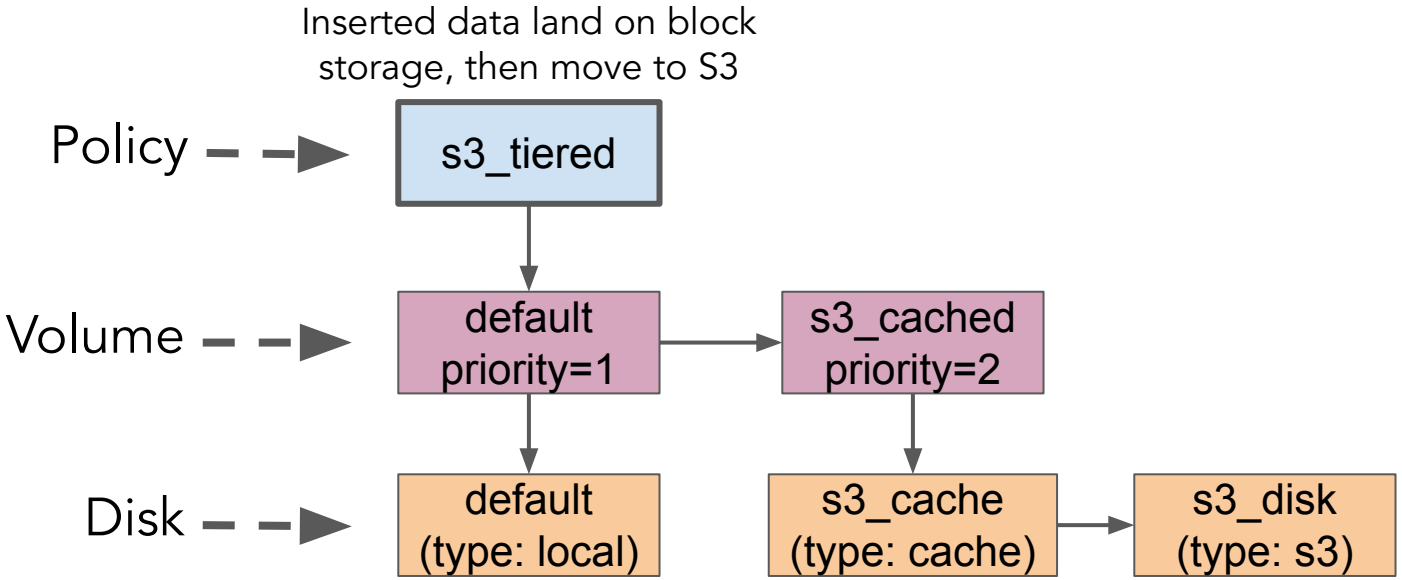
Tiered storage is a popular way to layer storage



Configuration for tiered storage

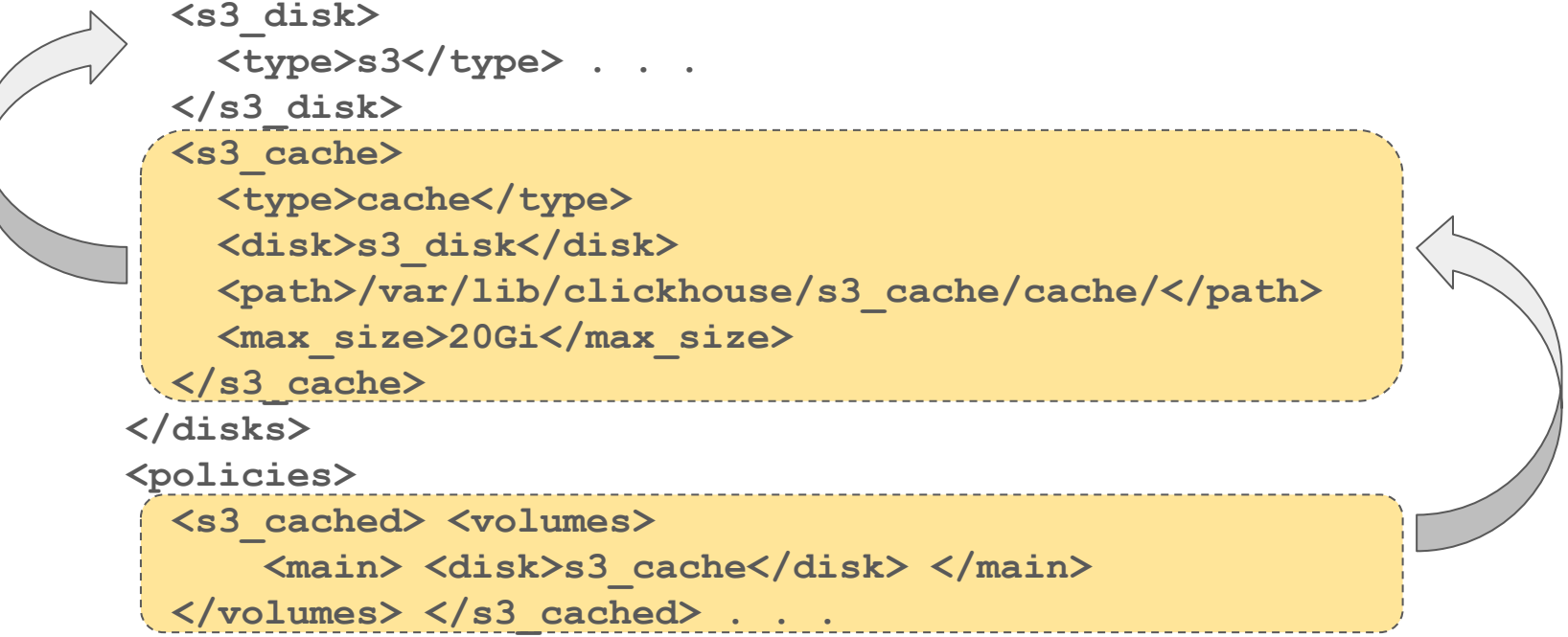
```
<clickhouse> <storage_configuration>
  <disks>
    <s3_disk> . . . </s3_disk>
  </disks>
  <policies>
    <s3_tiered>
      <volumes>
        <hot> <disk>default</disk>
              <move_factor>0.1</move_factor></hot>
        <cold> <disk>s3_disk</disk> </cold>
      </volumes>
    </s3_tiered>
  </policies>
</storage_configuration> </clickhouse>
```

Cache can be used to speed up S3 queries



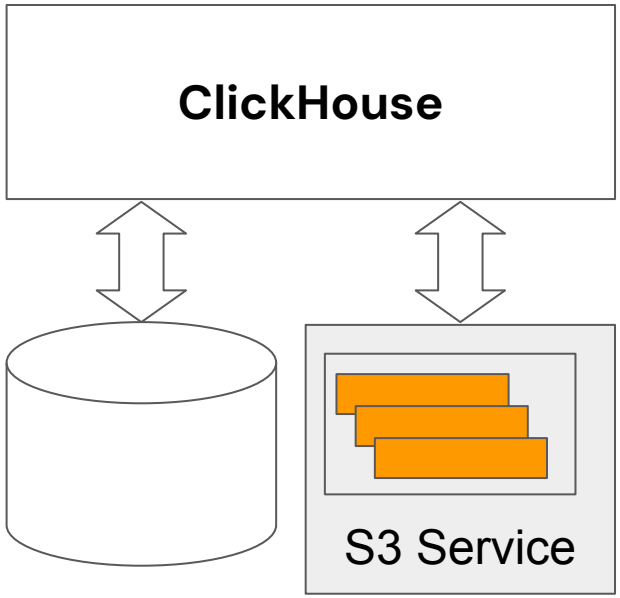
Using a cache instead of an s3 disk

```
<clickhouse> <storage_configuration> <disks>
    <s3_disk>
      <type>s3</type> . . .
    </s3_disk>
    <s3_cache>
      <type>cache</type>
      <disk>s3_disk</disk>
      <path>/var/lib/clickhouse/s3_cache/cache/</path>
      <max_size>20Gi</max_size>
    </s3_cache>
  </disks>
  <policies>
    <s3_cached> <volumes>
        <main> <disk>s3_cache</disk> </main>
    </volumes> </s3_cached> . . .
```



Now we are ready to use MergeTree with a storage policy

```
CREATE TABLE test_s3_direct
(
    `A` Int64,
    `S` String,
    `D` Date
)
ENGINE = MergeTree
PARTITION BY D
ORDER BY A
SETTINGS
  storage_policy = 's3_direct';
```



Moving from block storage to S3 using a TTL

```
CREATE TABLE test_s3_tiered(
    `A` Int64,
    `S` String,
    `D` Date
)
ENGINE = MergeTree
PARTITION BY D
ORDER BY A
TTL D + INTERVAL 7 DAY TO VOLUME 's3_cached'
SETTINGS storage_policy = 's3_tiered';
```

Two other ways to move data

Moves data automatically to next volume

```
<!-- Volume setting: move when less than 10% free -->
<move_factor>0.1</move_factor>
```

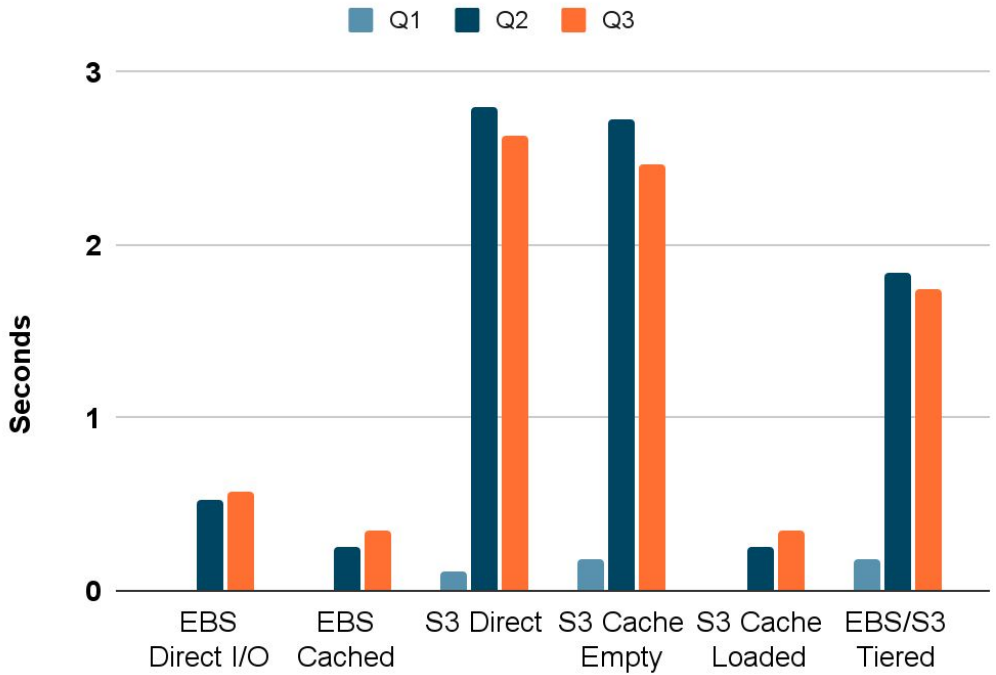
And you can always move data yourself

```
ALTER TABLE test_s3_tiered
MOVE PARTITION '2023-01-01' TO VOLUME 'cold'
```

Performance of tables with different storage policies

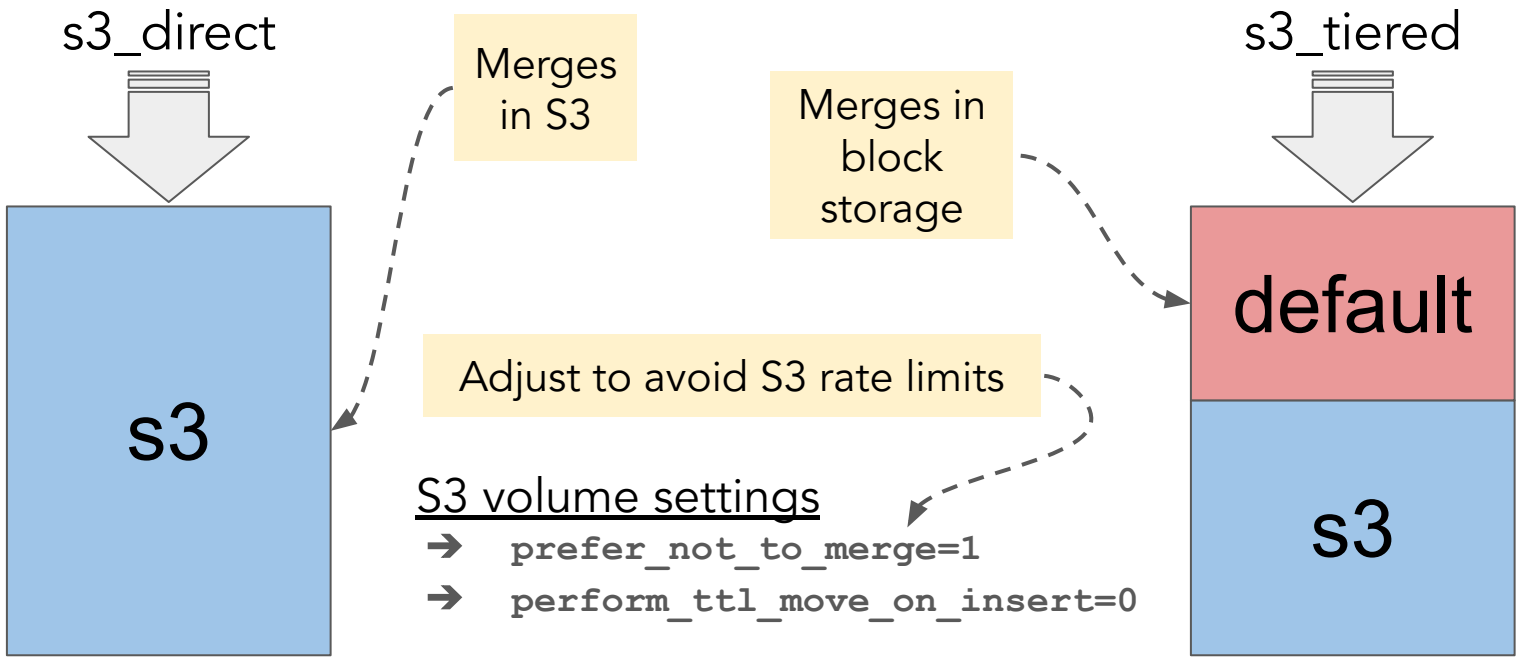
Test queries:

```
Q1: SELECT *
FROM test_s3_tiered
WHERE A = 443

Q2: SELECT uniq(A)
FROM test_s3_tiered;

Q3: SELECT count()
FROM test_s3_tiered
WHERE S LIKE '%4422%'
```



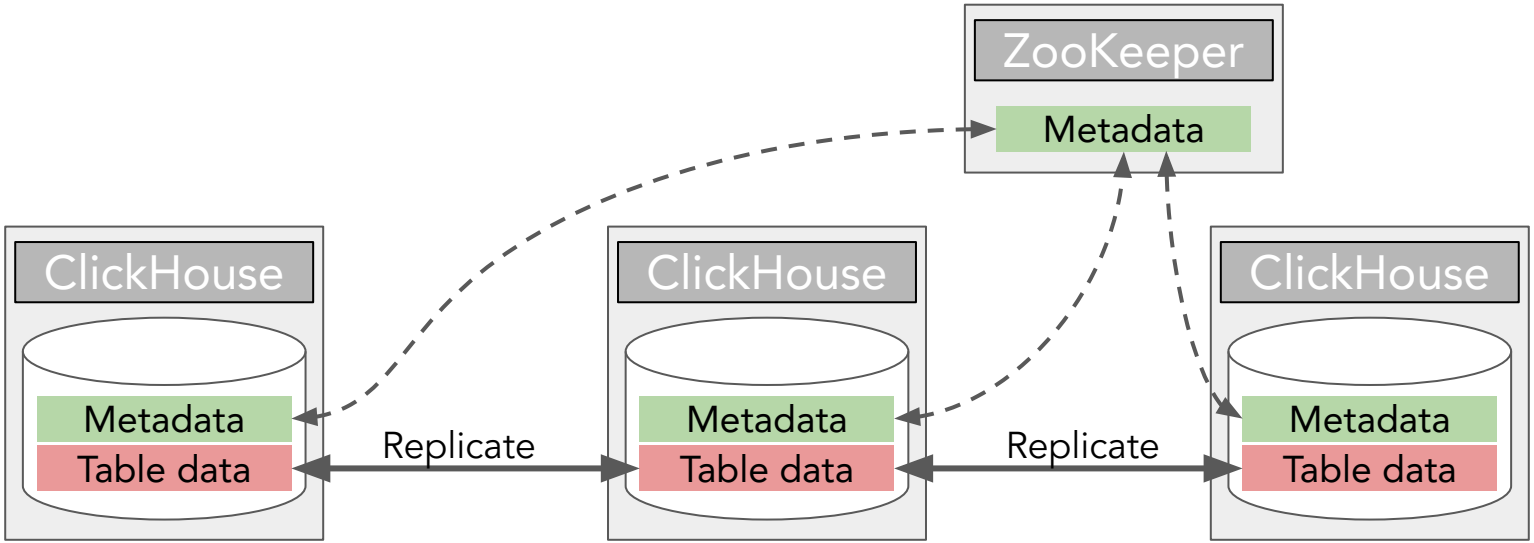
ClickHouse 23.7.4.5; M6i.2xlarge; EBS gp3

Managing background merges in S3

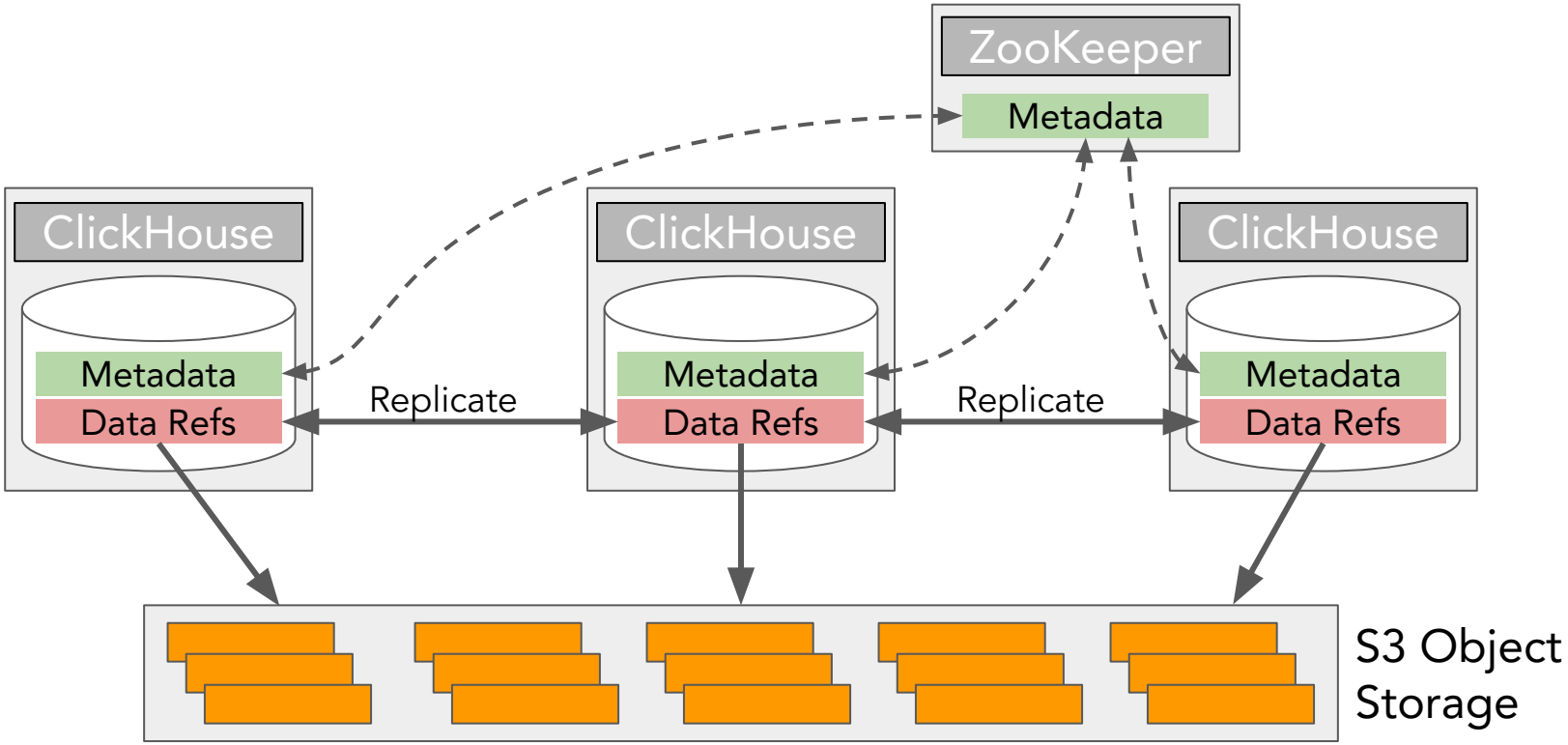


S3 storage in ClickHouse clusters

ReplicatedMergeTree over LocalStorage

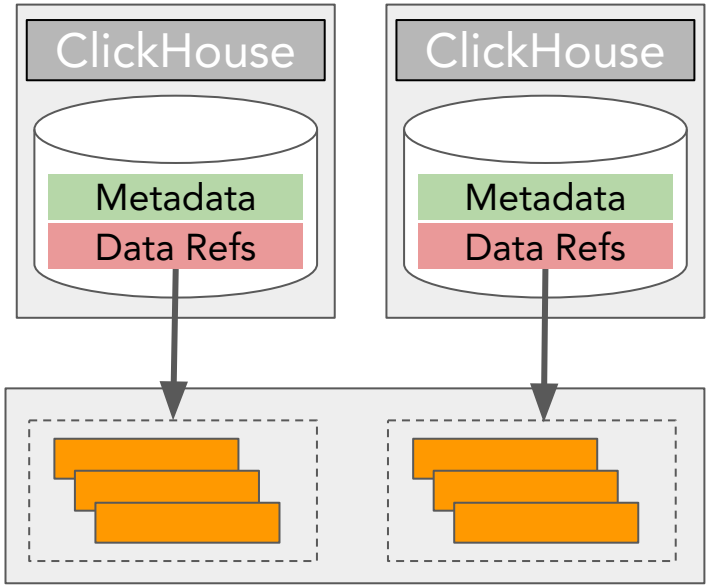


ReplicatedMergeTree over Object Storage

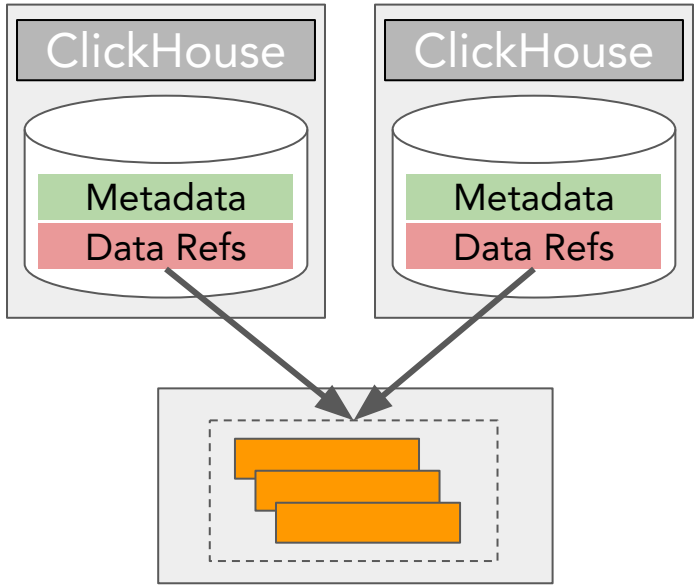


Two models for storing S3 table data

Multiple copies of S3 data



"Zero Copy"



MergeTree property to enable “zero copy” replication

```
<clickhouse>
  <merge_tree>
    <allow_remote_fs_zero_copy_replication>
      true
    </allow_remote_fs_zero_copy_replication>
  </merge_tree>
</clickhouse>
```

Current state of using S3 table storage in clusters

Good

- S3 is usable for long tail data that does not change a lot
- Replication works – use latest 23.7 or above
  - Lots of fixes
- max_parallel_replicas setting enables scale-out

Bad

- Refs and data are separated
- Zero-copy replication is still messy
  - Preserving ref counters across cluster is tricky
  - Hard to do backups
  - Needs more verification of recent fixes
- Every node needs all metadata => does not scale-out to many nodes

Using S3 as a Data Lake

Using S3 table engine to read and write Parquet

```
CREATE TABLE
test_s3_table_parquet (
    `A` Int64,
    `S` String,
    `D` Date
)
ENGINE = S3(<S3  Url>, 'Parquet')

INSERT INTO test_s3_table_parquet
SELECT number, number,
'2023-01-01' FROM numbers(1e8);
```

Table definition gives location and columns

Load 100M rows of test data

Multiple ways to select data from S3

```
SELECT count()
FROM test_s3_table_parquet
WHERE S LIKE '%4422%';
```

Selects using threads on the current server

```
SELECT count() FROM
s3(<S3  Url>, 'Parquet')
WHERE S LIKE '%4422%';
```

Selects using threads on the current server

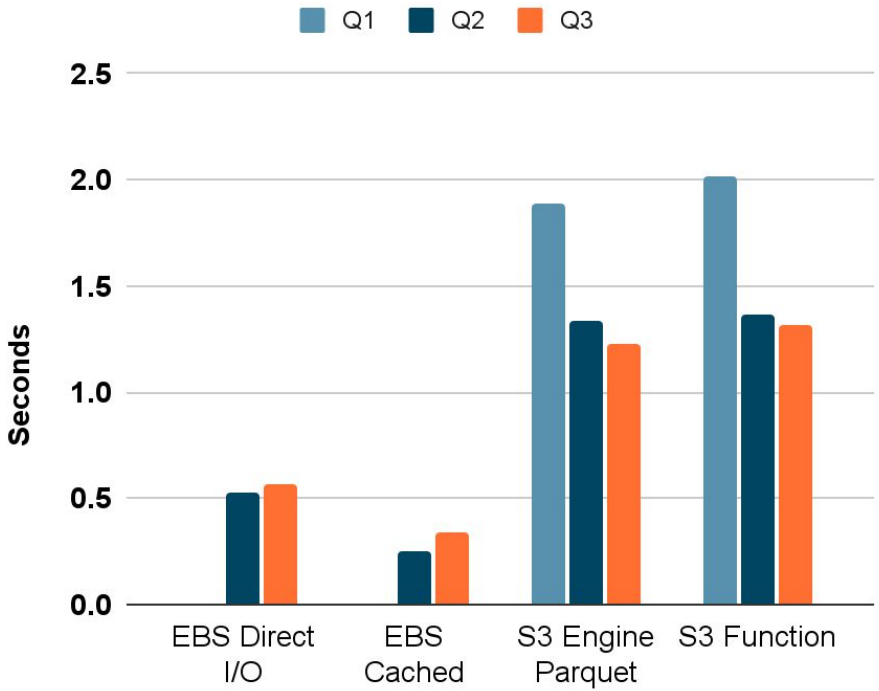
```
SELECT count() FROM
s3Cluster('<cluster>', '<S3  Url>', 'Parquet')
WHERE S LIKE '%4422%';
```

Selects using all servers in cluster

S3 Parquet is slower than block storage (but it's cheaper!)

Test queries:

```
Q1: SELECT *
FROM <test_table> WHERE
A = 443

Q2: SELECT uniq(A)
FROM <test_table>;

Q3: SELECT count()
FROM <test_table>
WHERE S LIKE '%4422%'
```



ClickHouse 23.7.4.5; M6i.2xlarge; EBS gp3

Using S3 table function is more powerful

```
INSERT INTO FUNCTION s3('<s3
url>/{_partition_id}.parquet',
<credentials>, 'Parquet')
PARTITION BY D
SELECT
  number as A,
  number as S,
  toDate('2023-01-01') +
intDiv(A,1e7) as D
  FROM numbers(1e8);
```

_partition_id macro allows to generate multiple files

Split written files by day

Load 100M rows of test data

Querying multiple Parquet files using globs

```
Q1: SELECT * FROM s3('<s3 url>/*.parquet',
<credentials>) WHERE A = 443

Q2: SELECT uniq(A) FROM s3('<s3 url>/*.parquet',
<credentials>);

Q3: SELECT count() FROM s3('<s3 url>/*.parquet',
<credentials>)
WHERE S LIKE '%4422%';
```

'_path' and '_file' virtual columns are available.

Ways to pass S3 credentials to ClickHouse

```
<clickhouse>
  <s3>
    <data-lake>
      <endpoint from_env="AWS_S3_DATALAKE_URL"/>
      <use_environment_credentials>1</use_environment_credentials>
    </data-lake>
  </s3>
</clickhouse>
```

Method 1: pass values as environment variables using <s3> configuration tag

Method 2: Pass keys as strings in <s3> configuration tag

Method 3: Grant cloud IAM role to ClickHouse VM

Method 4: Use a named collection with keys

Trade-offs associated with Parquet in S3

Good

- Works with files already in S3
- Parquet compression is good
- Data is accessible to applications outside of ClickHouse
- Parquet is faster than MergeTree on S3 in some cases
- Reads and writes with s3Cluster() are very fast due to parallelization

Bad

- Difficult to update/delete data from ClickHouse
- S3 table engine does not do what you expect; use s3() table function
- S3 tables do not merge or otherwise optimize file layout
- Hot/cold tiering is an application exercise

S3 Telemetry

Accessing ClickHouse counters for S3

```
-- Select all event counters for S3.
SELECT * FROM system.events WHERE event ILIKE '%s3%';

-- Print some of our faves.
SELECT
  sumIf(value, event = 'S3PutObject') as S3PutObject,
  sumIf(value, event = 'S3GetObject') as S3GetObject,
  sumIf(value, event = 'WriteBufferFromS3Bytes') as WriteBufferFromS3Bytes,
  sumIf(value, event = 'ReadBufferFromS3Bytes') as ReadBufferFromS3Bytes
FROM system.events;
```

S3PutObject	S3GetObject	WriteBufferFromS3Bytes	ReadBufferFromS3Bytes
4026	6987	5862641197	32284663529

Fetching data about the file system cache

```
-- Find file system cache metrics.
SELECT * FROM system.metrics
WHERE metric ILIKE '%filesystemcache%' ORDER BY metric;

-- Size of the disk cache.
SELECT cache_name, formatReadableSize(sum(size)) AS size
FROM system.filesystem_cache
GROUP BY cache_name;
```

```
┌─cache_name─┬─size─────┐
│ s3_cache   │ 1.07 GiB │
└────────────┴──────────┘
```

Checking allocated storage size

```
-- Get size of parts on each disk type.
SELECT disk_name, formatReadableSize(sum(bytes_on_disk))
FROM system.parts WHERE active
GROUP BY disk_name ORDER BY disk_name;
```

disk_name	formatReadableSize(sum(bytes_on_disk))
default	1.23 GiB
s3_disk	1.83 GiB

```
-- Sum files currently managed in s3.
SELECT formatReadableSize(sum(size)) FROM system.remote_data_paths;
```

formatReadableSize(sum(size))
3.66 GiB

Coming Attractions

Topics for further discussion

- SharedMergeTree (closed source from ClickHouse Inc.)
- Deliverables for improved open source S3 capabilities
  - Robust zero-copy replication
  - Simplification of storage organization
  - Better integration with existing tiered storage
  - Backup
  - Testing and robustness
  - Documentation

Want to assist with or sponsor S3 support improvements?
Contact Altinity at http://altinity.com

Wrap-Up

Best practices for MergeTree on S3

- Avoid a lot of changes to data in S3
  - Don’t refresh data every day for example - will drive up S3 API costs
- Add S3 and caching telemetry to monitoring
- Use zero copy replication with caution

Best practices for data lake / Parquet approach

- Data should be readonly
- Ensure data is properly merged and sorted before archiving to Parquet
- If you want to delete data you'll need to design for it
  - E.g., split up Parquet files by tenant
- Be careful with file names when writing data to S3

References

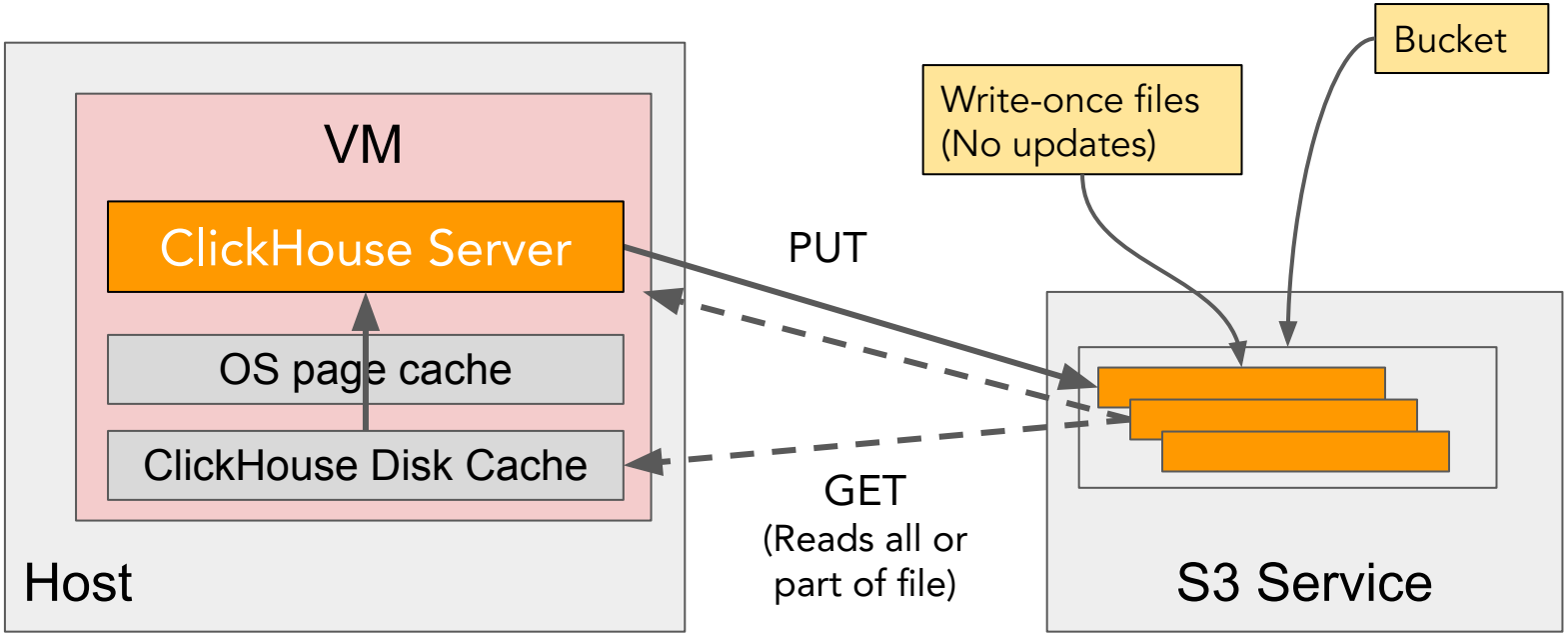
- Samples for this talk: https://github.com/Altinity/clickhouse-sql-examples
- ClickHouse Inc documentation
- DoubleCloud Blog: How S3-based ClickHouse hybrid storage works under the hood
- Altinity KB articles on S3 policies and cache behavior (such as this one)
- Altinity Blog articles tagged with S3

Thank you!

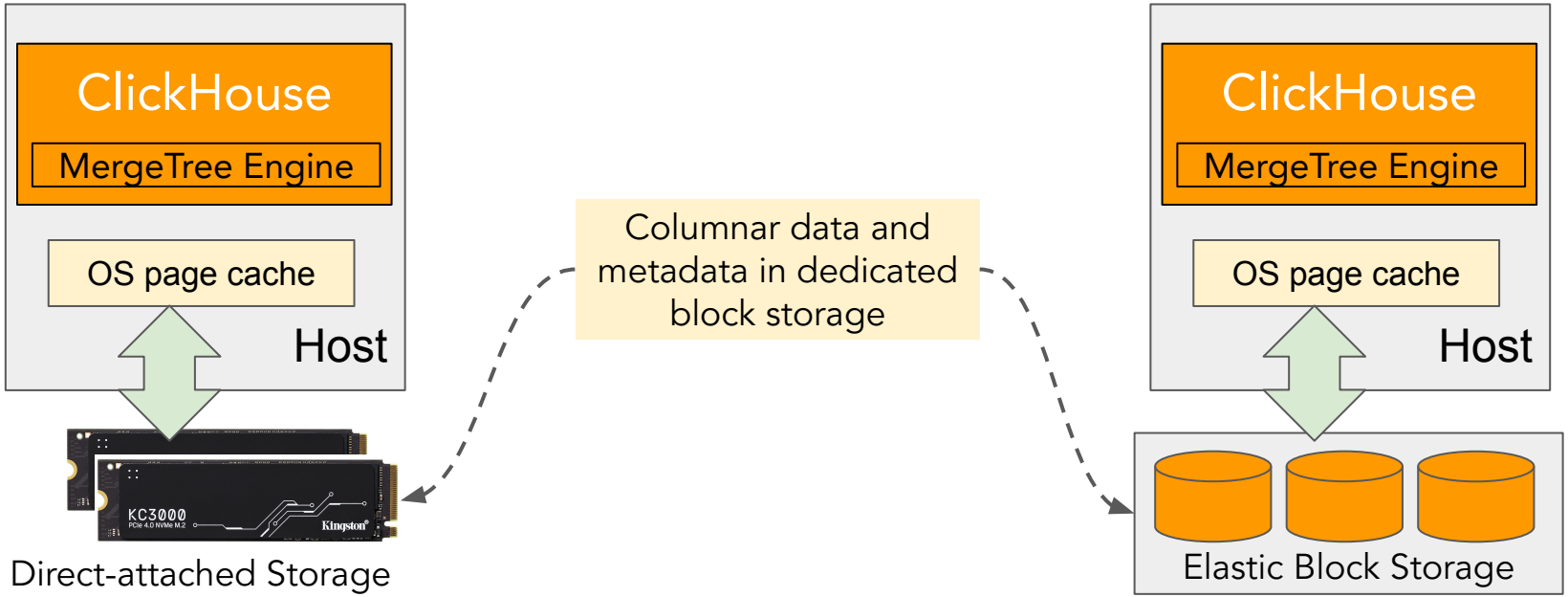
Visit us:
https://altinity.com
Altinity Slack (Invite Link)

Altinity.Cloud
Altinity Stable Builds for ClickHouse
Altinity Kubernetes Operator for ClickHouse

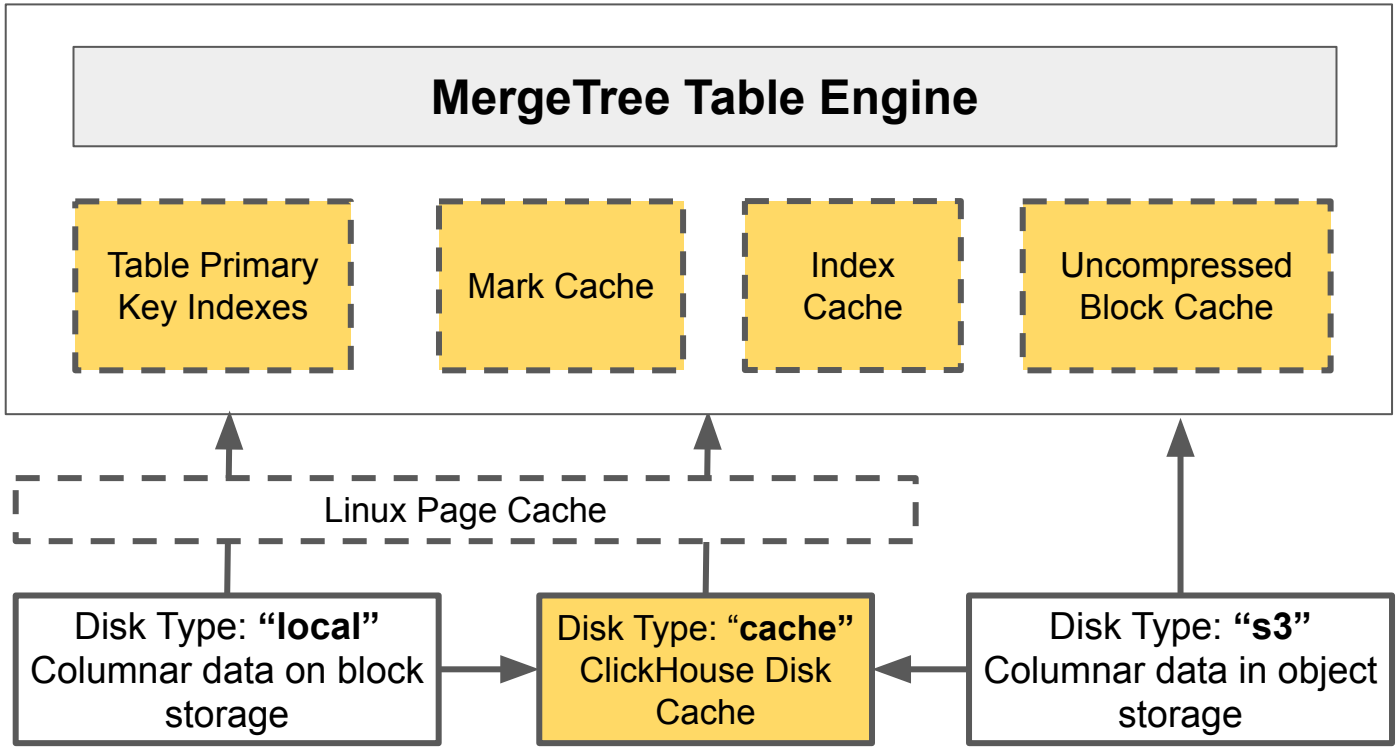
Here's how object storage works in Altinity.Cloud



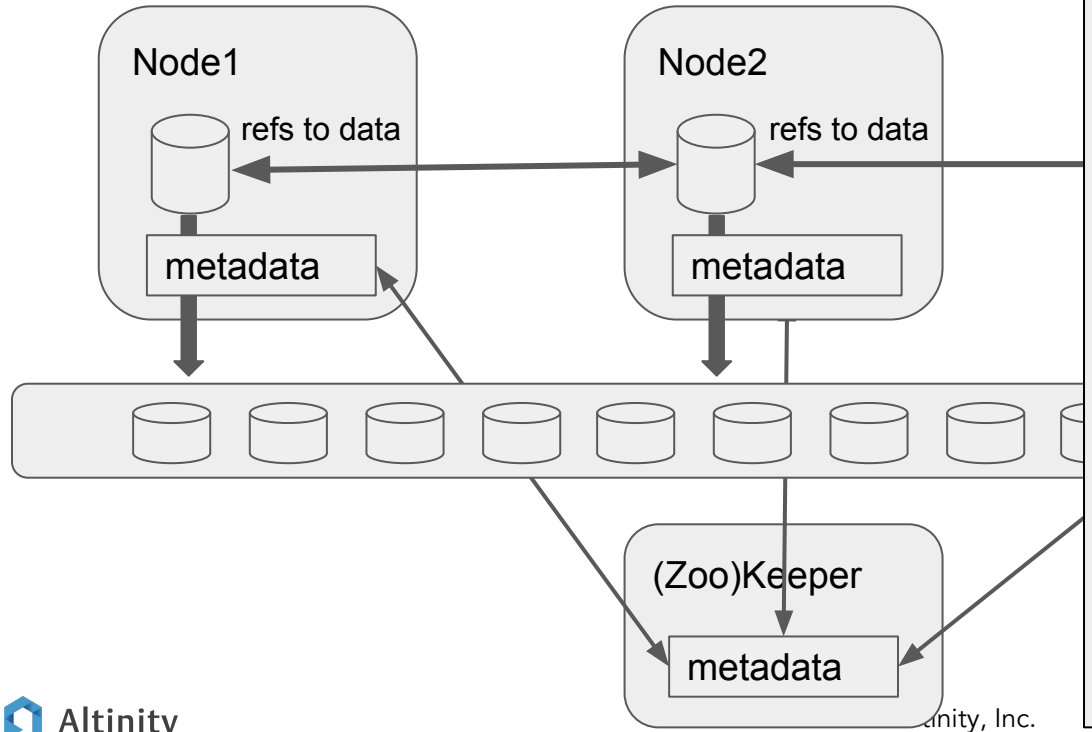
Here's where the data goes



How MergeTree manages storage and caches in general



ReplicatedMergeTree over Object Storage



Goods:

- It works
- max_parallel_replicas for scaling out

Bads:

- Refs and data are separated
- Zero-copy replication is still messy
- Every node needs all metadata => does not scale-out to many nodes