Building real-time applications with ClickHouse Materialized Views

Dale McDiarmid, Product@ClickHouse

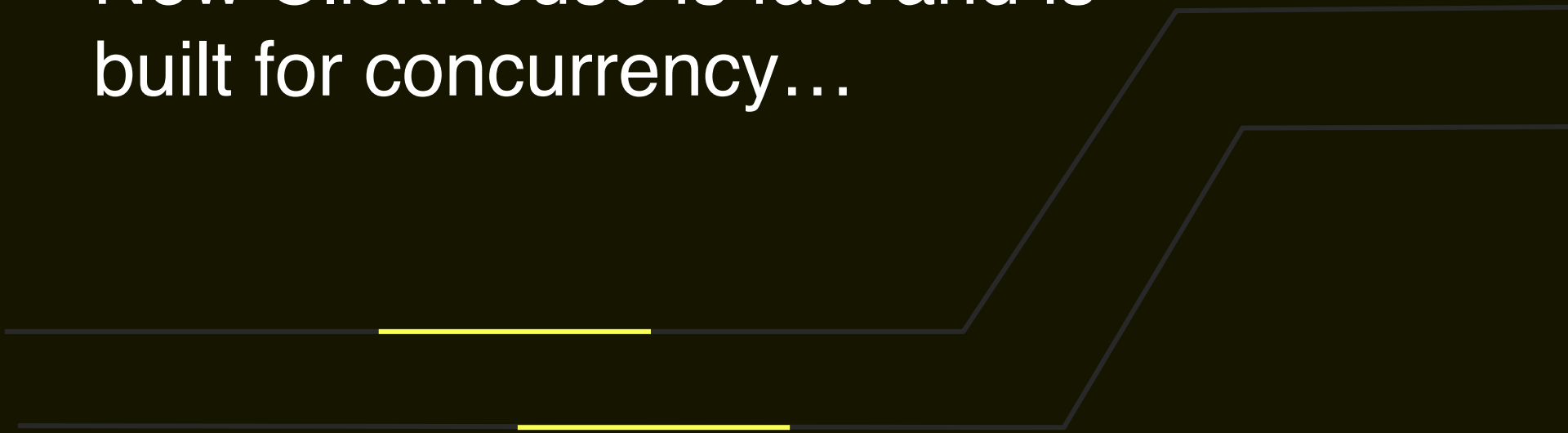
Sept, 2023

ClickHouse

What do applications powered by real-time analytics need?

- Fast queries
- Support for high concurrency
- Ability to deal with variable filters
- Typically aggregating for visuals e.g. over time
- Even with hundreds of billions of rows.…hence ClickHouse
- …a nice UI

Now ClickHouse is fast and is built for concurrency…



ClickHouse

ClickHouse

But there are always limits…

- Some queries just need to scan and summarise a lot of data
- You can't optimise the ordering key for all access patterns/ filters (projections can help)
- Your first tool in the ClickHouse box is materialised views

```
SELECT
    project,
    formatReadableQuantity(count()) AS c
FROM pypi
GROUP BY project
ORDER BY count() DESC
LIMIT 6
SETTINGS allow_experimental_parallel_reading_from_replicas = 1, max_parallel_replicas = 100

Query id: 14b514cf-6c7a-4679-a9d7-62b1501faa25

┌─project────┬─c─────────────┐
│ boto3      │ 2.14 billion  │
│ urllib3    │ 1.08 billion  │
│ botocore   │ 975.57 million │
│ requests   │ 934.64 million │
│ awscli     │ 842.87 million │
│ setuptools │ 824.49 million │
└────────────┴───────────────┘

6 rows in set. Elapsed: 22.171 sec. Processed 69.56 billion rows, 1.31 TB (3.14 billion rows/s., 59.26 GB/s.)
Peak memory usage: 766.14 MiB.

clickhouse-cloud :)
```

What are materialised views?

(In ClickHouse)

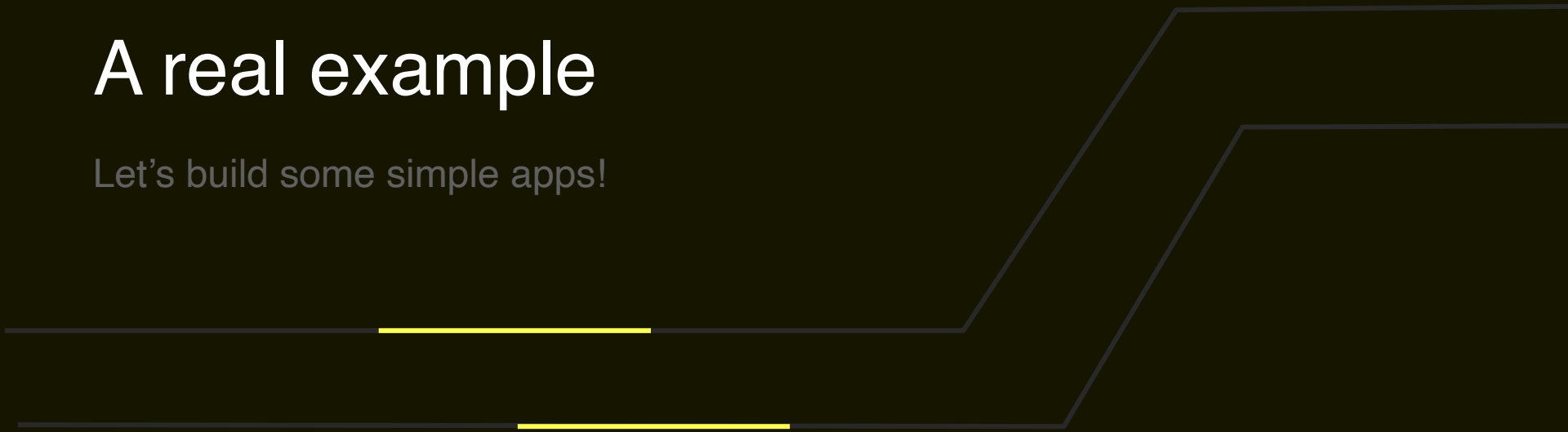
ClickHouse

Just an Insert Trigger!

- A little different than those you might be used to e.g. Postgres
- A query which triggers on inserts to a table
- Query executes on the inserted block of rows
- Results can be sent to another table
- ***Doesn't store any data itself***
- Can be used (amongst other things) to summarise data and massively improve query performance
- ***Move "work" from insert to query time by pre-calculating result sets***

A real example

Let’s build some simple apps!



ClickHouse

ClickHouse

PYPI package downloads

- PYPI = Python Package Index
- Every download of a Python package
- Once row for every package download
- Every pip install!
- 600b rows
- Let’s build some analytics!

```
{
    "timestamp": "2023-07-21 03:20:25.000000",
    "country_code": "JP",
    "url": "\/packages\/2d\/61\/08076519c80041bc0ffa1a8af0cbd3bf3e2b62af10435d269a9d0f40564d\/requests-2.27.1-py2.py3-none-any.whl",
    "project": "requests",
    "file": {
        "filename": "requests-2.27.1-py2.py3-none-any.whl",
        "project": "requests",
        "version": "2.27.1",
        "type": "bdist_wheel"
    },
    "installer": {
        "name": "pip",
        "version": "20.1"
    },
    "python": "3.6.13",
    "implementation": {
        "name": "CPython",
        "version": "3.6.13"
    },
    "distro": {
        "name": "Debian GNU\/Linux",
        "version": "10",
        "id": "buster",
        "libc": {
            "lib": "glibc",
            "version": "2.28"
        }
    },
    "system": {
        "name": "Linux",
        "release": "4.14.314-164.539.amzn1.x86_64"
    },
    "cpu": "x86_64",
    "openssl_version": "OpenSSL 1.1.1d  10 Sep 2019",
    "setuptools_version": "46.1.3",
    "rustc_version": "",
    "tls_protocol": "TLSv1.3",
    "tls_cipher": "TLS_AES_256_GCM_SHA384"
}
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
 ) ENGINE = MergeTree
ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27     DE      boto3
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
 ) ENGINE = MergeTree
ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27     DE      boto3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27     DE      boto3
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
GROUP BY project;
```

```
2023-03-29 04:28:27    DE    boto3
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

pypi_downloads

```
boto3                  1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:28:27    DE    boto3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

pypi_downloads

```
boto3                  1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

pypi_downloads

```
boto3    1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
```

pypi_downloads

```
boto3    1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

```
2023-03-29 04:28:27    DE    boto3
2023-03-29 04:30:07    US    urllib3
```

pypi_downloads

```
boto3    1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
GROUP BY project;
```

```
2023-0 urllib3    1 b3
```

pypi

```
2023-03-29 04:28:27    DE    boto3
2023-03-29 04:30:07    US    urllib3
```

pypi_downloads

```
boto3    1
```

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:30:07	US	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

```
2023-03-29 04:30:07    US    urllib3
```

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:30:07	US	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:30:07	US	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:30:07	US	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

CREATE MATERIALIZED VIEW

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	1
urllib3	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	1
urllib3	1

boto3	2
requests	3
urllib3	2

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

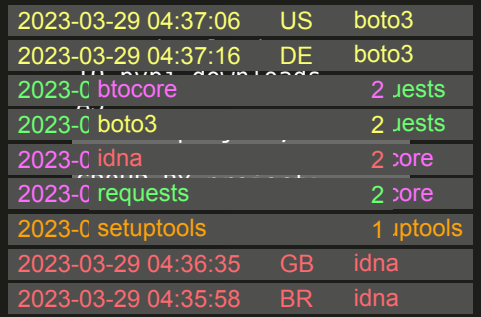
pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna



pypi

2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

boto3	3
requests	3
urllib3	3

btocore	2
boto3	2
idna	2
requests	2
setuptools	1

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
CREATE TABLE pypi
(
  timestamp DateTime,
  country_code String,
  project String,
  ...
) ENGINE = MergeTree
  ORDER BY (project, timestamp)
```

2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna

```
CREATE MATERIALIZED VIEW
pypi_downloads_mv
TO pypi_downloads
AS
SELECT project, count()
FROM pypi
GROUP BY project;
```

pypi

2023-03-29 04:36:11	AU	btocore
2023-03-29 04:36:23	CA	btocore
2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:37:06	US	boto3
2023-03-29 04:37:16	DE	boto3
2023-03-29 04:36:35	GB	idna
2023-03-29 04:35:58	BR	idna
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:38:24	IN	requests
2023-03-29 04:37:52	FR	requests
2023-03-29 04:38:52	US	setuptools
2023-03-29 04:30:07	US	urllib3
2023-03-29 04:31:12	IT	urllib3
2023-03-29 04:31:23	JP	urllib3

pypi_downloads

btocore	2
boto3	5
idna	2
requests	5
setuptools	1
urllib3	3

```
CREATE TABLE pypi_downloads
(
  project String,
  count UInt64
) ENGINE = SummingMergeTree
  ORDER BY project
```

```
SELECT
    project,
    count() AS c
FROM pypi
GROUP BY project
ORDER BY c DESC
LIMIT 6
```

project	c
boto3	1.82 billion
urllib3	1.03 billion
requests	928.54 million
btocore	925.56 million
charset-normalizer	785.69 million
setuptools	775.68 million

~65 billion rows
14 TB (1 TB cmpr.)

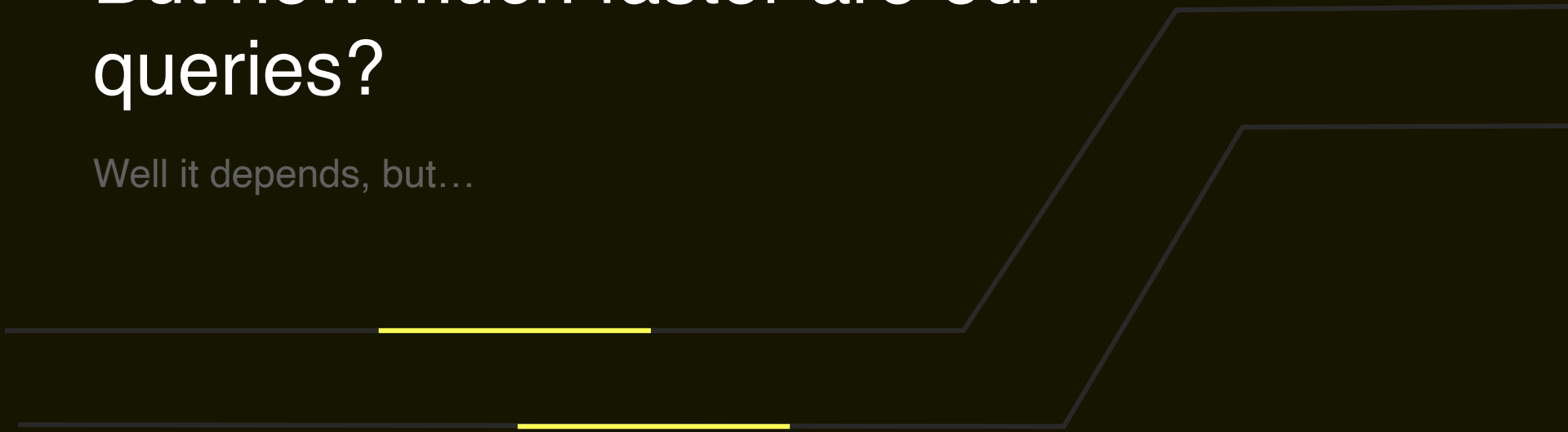
2023-03-29 04:36:11	AU	btocore
...		
2023-03-29 04:28:27	DE	boto3
2023-03-29 04:32:45	NL	boto3
2023-03-29 04:35:16	FR	boto3
2023-03-29 04:37:06	US	boto3
...		
2023-03-29 04:36:35	GB	idna
...		
2023-03-29 04:34:42	US	requests
2023-03-29 04:36:14	US	requests
2023-03-29 04:35:21	GB	requests
2023-03-29 04:38:24	IN	requests
...		
2023-03-29 04:38:52	US	setuptools
...		
2023-03-29 04:31:12	IT	urllib3
...		

~500k rows
10 MB (4 MB cmpr.)

btocore	925.56 million
boto3	1.82 billion
idna	764.01 million
requests	928.54 million
setuptools	775.68 million
urllib3	1.03 billion

But how much faster are our queries?

Well it depends, but…



ClickHouse

Demo

A few considerations

- ClickPy is summing counts over time in most cases. A simple SummingMergeTree is sufficient in these cases.
- More complex aggregations e.g. average, percentiles, require an AggregatingMergeTree and more careful configuration. But its supported!
- MV's can also be chained!
- With applications with many filters, you can't have every column in the MV - at some point you just have the original table.
- For ClickPy we have many materialised views, of increasing detail. If every column is added as a filter, we default to the base table.

ClickHouse

One of several key tools for speeding up queries

- Always optimise you schema first and ordering key
- Re-write queries
- Consider projections
- Dictionaries

But all you need is ClickHouse

Thank you!

Content heavy slide

Content heavy slide (2 columns)

Content slide - less text heavy

What is ClickHouse?

Open source

- Developed since 2009
- OSS 2016
- 28k+ Github stars
- 1k+ contributors
- 300+ releases

column-oriented

- Best for aggregations
- Files per column
- Sorting and indexing
- Background merges

distributed

- Replication
- Sharding
- Multi-master
- Cross-region

OLAP database

- Analytics use cases
- Aggregations
- Visualizations
- Mostly immutable data

“

Just

ROKT

Attributor, role, company

What is ClickHouse? (highlighted)

Open source

- Developed since 2009
- OSS 2016
- 28k+ Github stars
- 1k+ contributors
- 300+ releases

column-oriented

- Best for aggregations
- Files per column
- Sorting and indexing
- Background merges

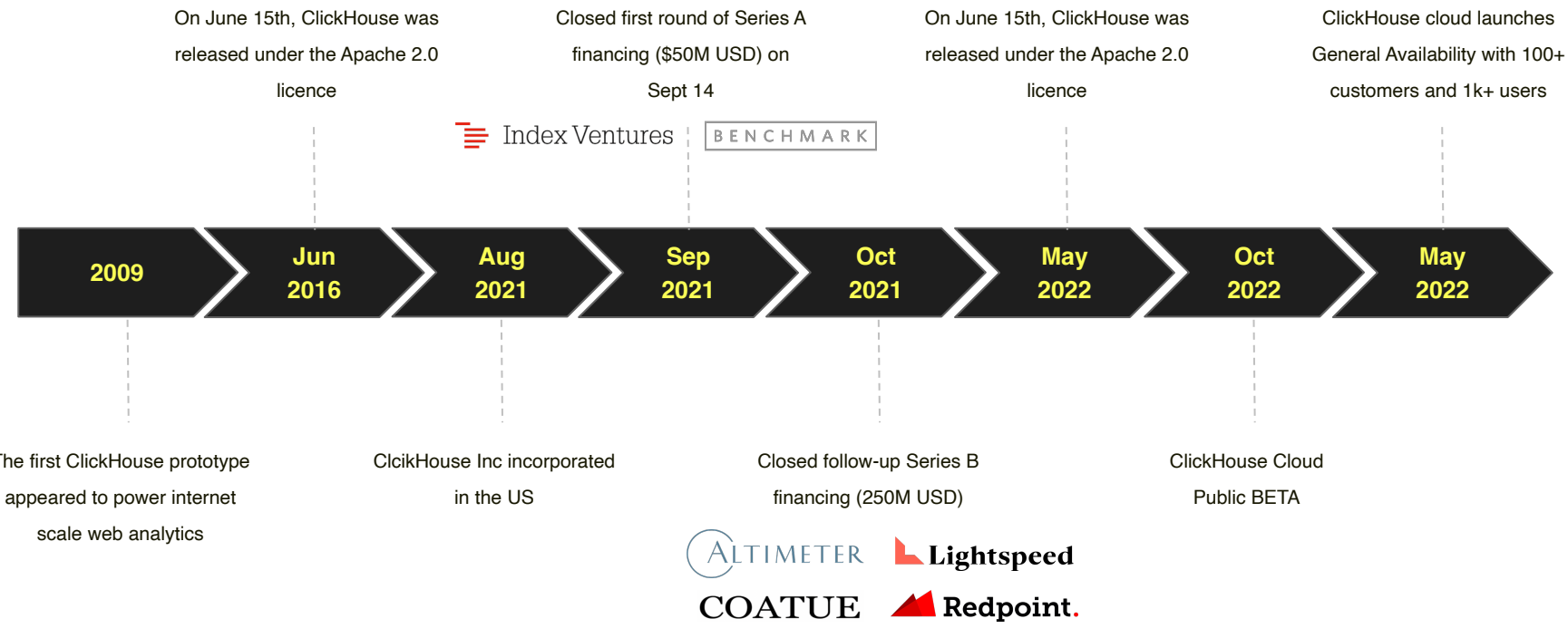
distributed

- Replication
- Sharding
- Multi-master
- Cross-region

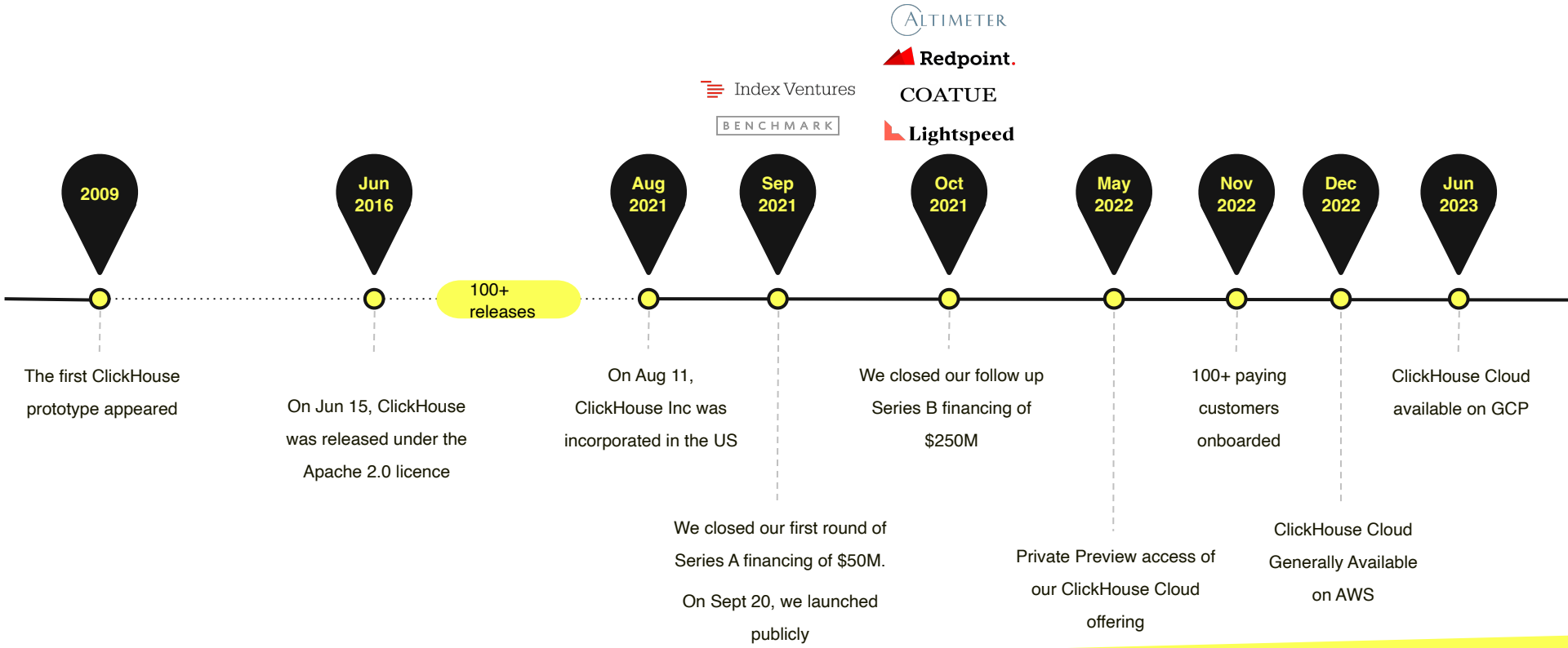
OLAP database

- Analytics use cases
- Aggregations
- Visualizations
- Mostly immutable data

ClickHouse Journey



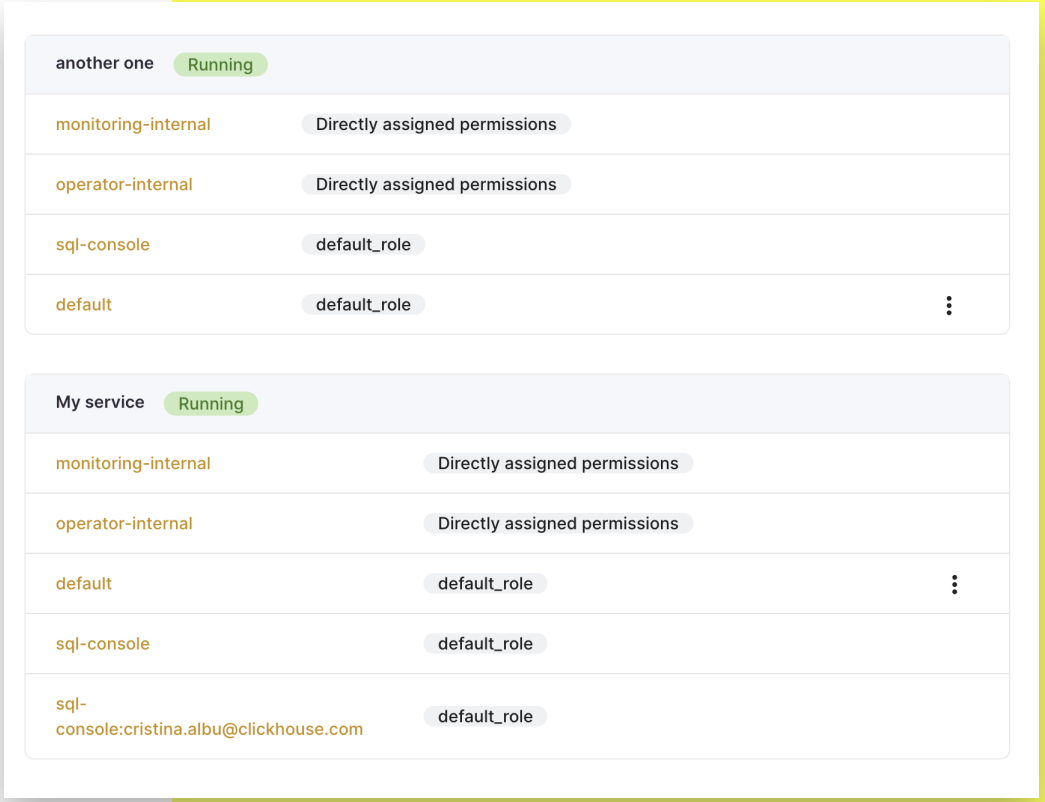
The ClickHouse Journey



ClickHouse

Feature highlight

- list item 1
- list item 2
- list item 3



Optimized for row-based operations and transactions

- Hits performance limitations with analytical workloads
- Scales inadequately as data volumes increase
- Inevitably leads to growing operational complexity

Delivers unparalleled performance for analytical workloads at scale

- ✓ The fastest real-time database for analytics
- ✓ Purpose-built to manage massive volumes of data. Scales both vertically and horizontally
- ✓ With ClickHouse, real-time just works. There is no need for the operational complexity that exists when retrofitting another system for these workloads

Two cards highlight

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Three cards highlight

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Cloud Pricing

Development

Great for smaller workloads and starter projects

$50 - $193 / Month

- ✓ Up to 1 TB storage
- ✓ 16 GiB total memory
- ✓ Burstable CPU
- ✓ Backups every 24h, retained 1 day
- ✓ Replicated across 2 AZs
- ✓ Expert support with 24h response time

Storage	Compute
$35.33 per TB/mo	**$0.2160 per unit/hr**

Production

Designed to handle production workloads

Usage Based

- ✓ Unlimited storage
- ✓ 24GiB+ total memory
- ✓ Dedicated CPU
- ✓ Backups every 24h, retained 1 day
- ✓ Replicated across 3 AZs
- ✓ Expert support with 1h SLA
- ✓ AWS private link support
- ✓ Automatic scaling

Storage	Compute
$47.10 per TB/mo	**$0.6888 per unit/hr**

Dedicated

Designed for the most demanding latency-sensitive workloads

Capacity Based

- ✓ Unlimited storage
- ✓ Custom compute options
- ✓ Dedicated environment
- ✓ Advanced isolation and security
- ✓ Customized backup controls
- ✓ Scheduled upgrades
- ✓ Uptime SLAs
- ✓ Consultative migration guidance
- ✓ Dedicated support engineers

Contact us

Use this slide for code (dark)

```
SELECT
    toStartOfMonth(upload_date) AS month,
    sum(view_count) AS `Youtube Views`,
    bar(sum(has_subtitles) / count(), 0.55, 0.7, 100) AS `% Subtitles`
FROM youtube
WHERE (month >= '2020-08-01') AND (month <= '2021-08-01')
GROUP BY month
ORDER BY month ASC

13 rows in set. Elapsed: 0.823 sec Processed 1.07 billion rows, 11.75 GB (1.30 billion rows/s., 14.27 GB/s.)
```


ClickHouse Cloud - from 0 to 1 in under a year

Private preview
May -July 2022

Serverless hosted ClickHouse for key design partners

- AWS with 3 regions initially
- Limited scalability within predefined limits
- Basic cloud console for ClickHouse service and user management
- Strong security and privacy with SOC 2 Type I compliance

Public Beta
Oct 2022

Serverless hosted ClickHouse for early adopters

- Ecosystem of first-party connectors to onboard and work with data
- Automatic scaling up and down for compute
- Integrated billing using pay as you go pricing model
- Enhanced security features such as Private Link, IP Filtering, Auditing

General Availability
Dec 2022

Serverless hosted ClickHouse for the broader market

- Enhanced cloud console for analytics and operational controls
- Support for more AWS regions and AWS marketplace billing
- Uptime SLA and additional operations tools
- Advanced security features with SOC 2 Type II compliance

“

Rokt has been an eager partner of ClickHouse as we modernize our analytics stack. By offloading operations to the experts our developers are focused on delivering the best experience possible while the business scales. We we are thrilled to see the path ClickHouse is forging.

Attributor, role, company

ROKT

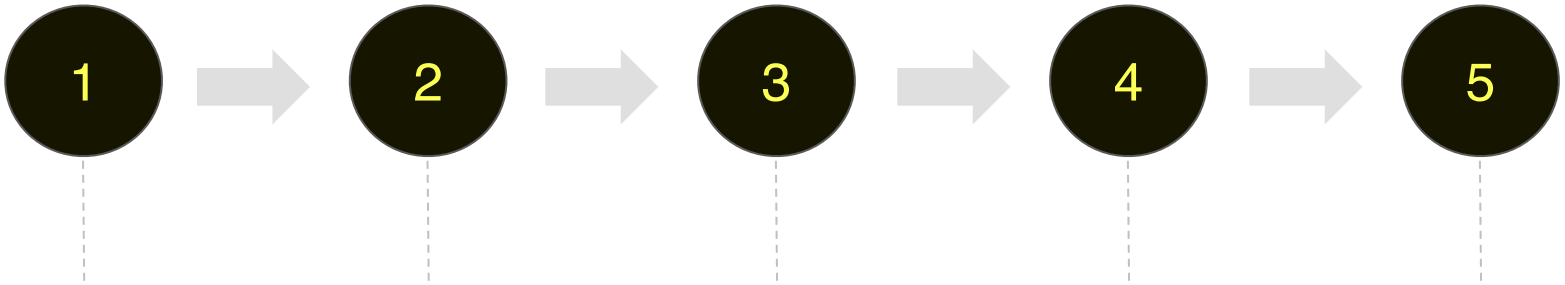
Use this slide for code (light)

```
SELECT
    toStartOfMonth(upload_date) AS month,
    sum(view_count) AS `Youtube Views`,
    bar(sum(has_subtitles) / count(), 0.55, 0.7, 100) AS `% Subtitles`
FROM youtube
WHERE (month >= '2020-08-01') AND (month <= '2021-08-01')
GROUP BY month
ORDER BY month ASC
```

13 rows in set. Elapsed: 0.823 sec **Processed 1.07 billion rows**, 11.75 GB (1.30 billion rows/s., 14.27 GB/s.)

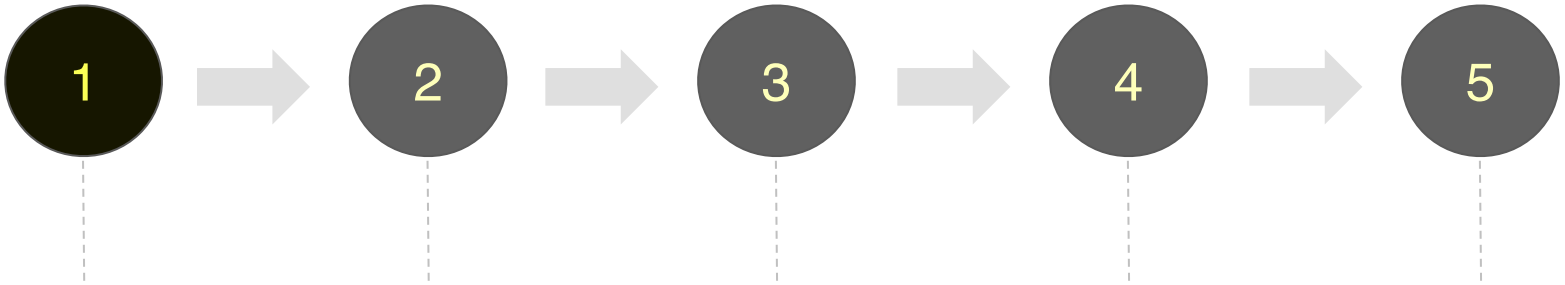
Process diagram, 5 ideas

Here you could describe the topic of the section, or not



Process diagram, 5 ideas

Highlighting one of the steps



Big number treatment

2.3k

Header here

Supporting text goes here, under the header

1.2M

Header here

Supporting text goes here, under the header

45

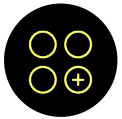
Header here

Supporting text goes here, under the header

Features

Blazing fast

Uses all available hardware to its full potential to process each query as fast as possible. Peak processing performance for a single query stands at more than 2 terabytes per second.



Fault tolerant

Supports async replication and can be deployed across multiple datacenters. All nodes are equal, which allows avoiding having single points of failure.

Easy to use

ClickHouse is simple and works out-of-the-box. Simplifies data processing by instantly processing structured data using a user-friendly SQL dialect and eliminating non-standard API requirements.

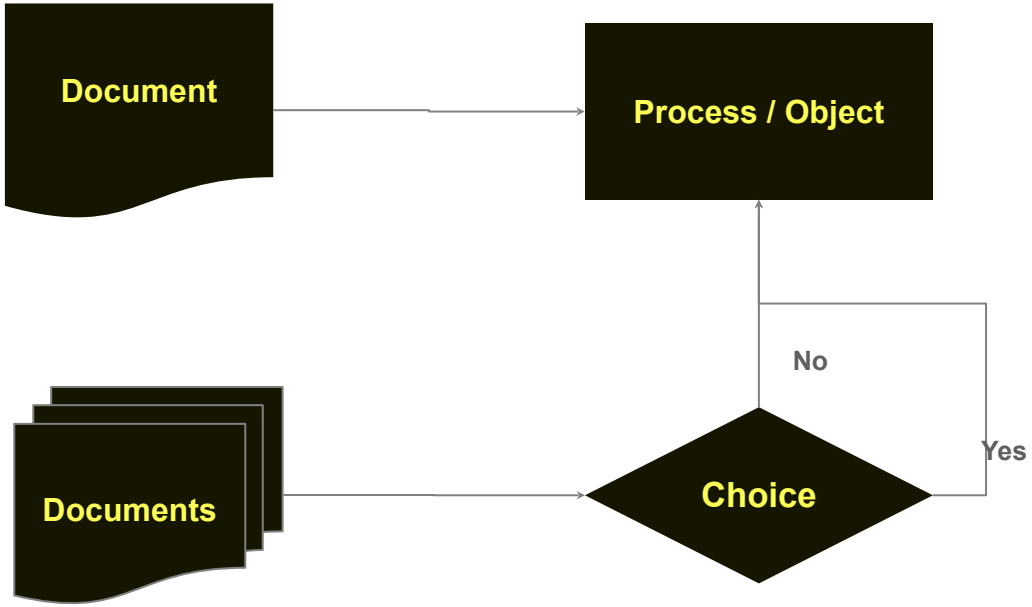
Highly reliable

Can be configured as a purely distributed system located on independent nodes, without any single points of failure. It also includes a lot of enterprise-grade security features and fail-safe mechanisms against human errors.

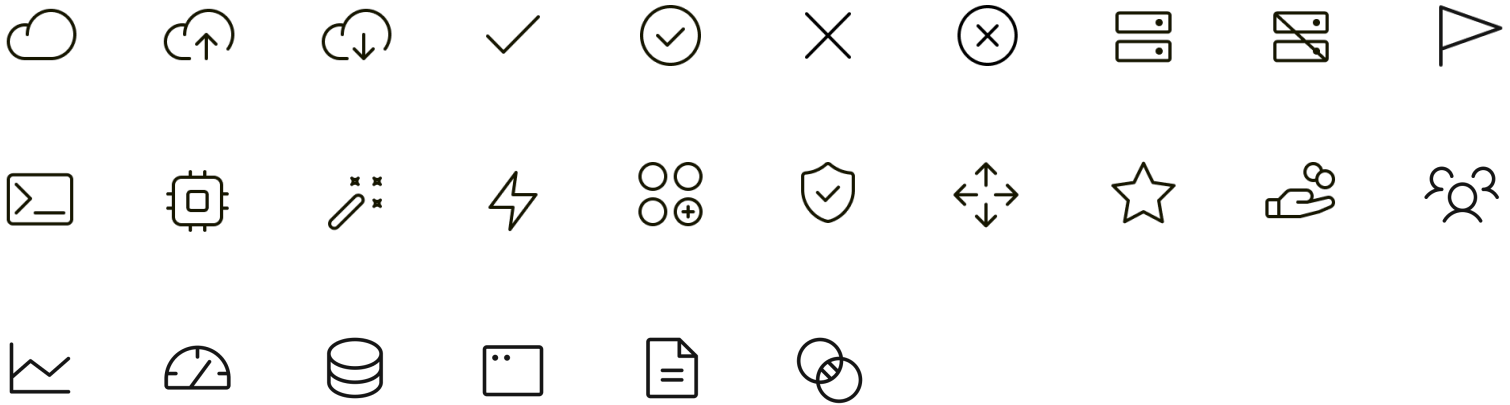
Table format

Header 1	Header 2	Header 3	Header 4	Header 5

For architecture/business process drawings



Icons for light background



Speakers (dark mode)

Aaron Katz

CEO @ ClickHouse

Alexey Milovidov

CTO @ClickHouse

Yury Izrailevsky

President and
VP of Engineering

Table of contents - many topics (dark mode)

01 **Content title**
Presenter / brief description

02 **Content title**
Presenter / brief description

03 **Content title**
Presenter / brief description

04 **Content title**
Presenter / brief description

05 **Content title**
Presenter / brief description

06 **Content title**
Presenter / brief description

Table of contents - fewer topics(dark mode)

01 **Content title**
Presenter / brief content description

02 **Content title**
Presenter / brief content description

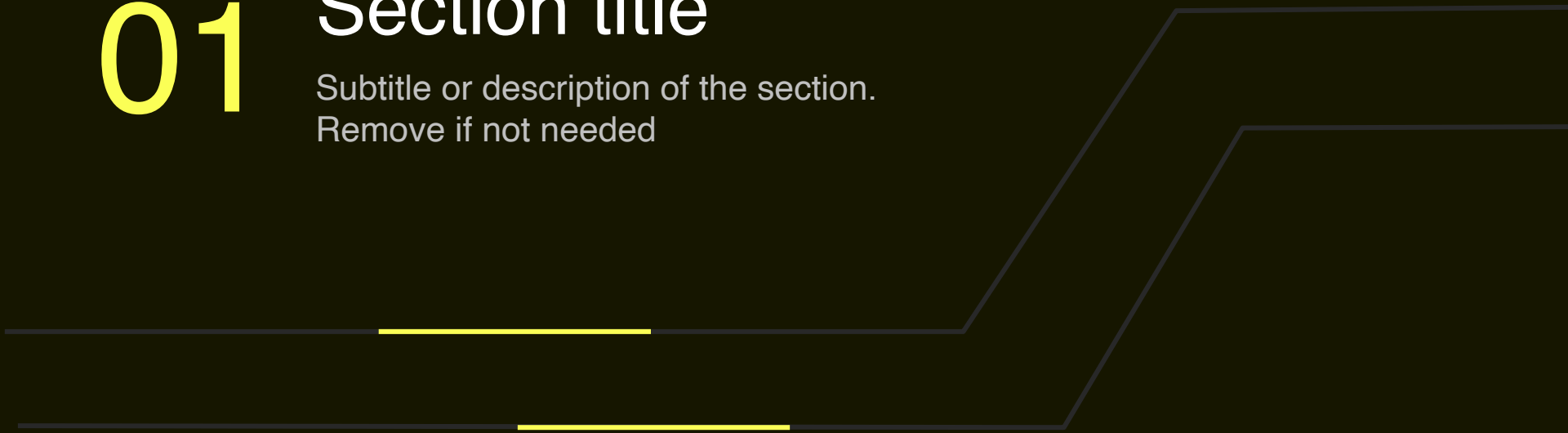
03 **Content title**
Presenter / brief content description

04 **Content title**
Presenter / brief content description

01

Section title

Subtitle or description of the section.
Remove if not needed



ClickHouse

Content slide - less text heavy

Two cards highlight

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Three cards highlight

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet her

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet here

Title here

Subtitle or a brief description

- Another bullet here
- Another bullet here
- Another bullet here
- Another bullet her

“

Rokt has been an eager partner of ClickHouse as we modernize our analytics stack. By offloading operations to the experts our developers are focused on delivering the best experience possible while the business scales. We we are thrilled to see the path ClickHouse is forging.

Attributor, role, company

ROKT

Features

Blazing fast

Uses all available hardware to its full potential to process each query as fast as possible. Peak processing performance for a single query stands at more than 2 terabytes per second.

Fault tolerant

Supports async replication and can be deployed across multiple datacenters. All nodes are equal, which allows avoiding having single points of failure.

Easy to use

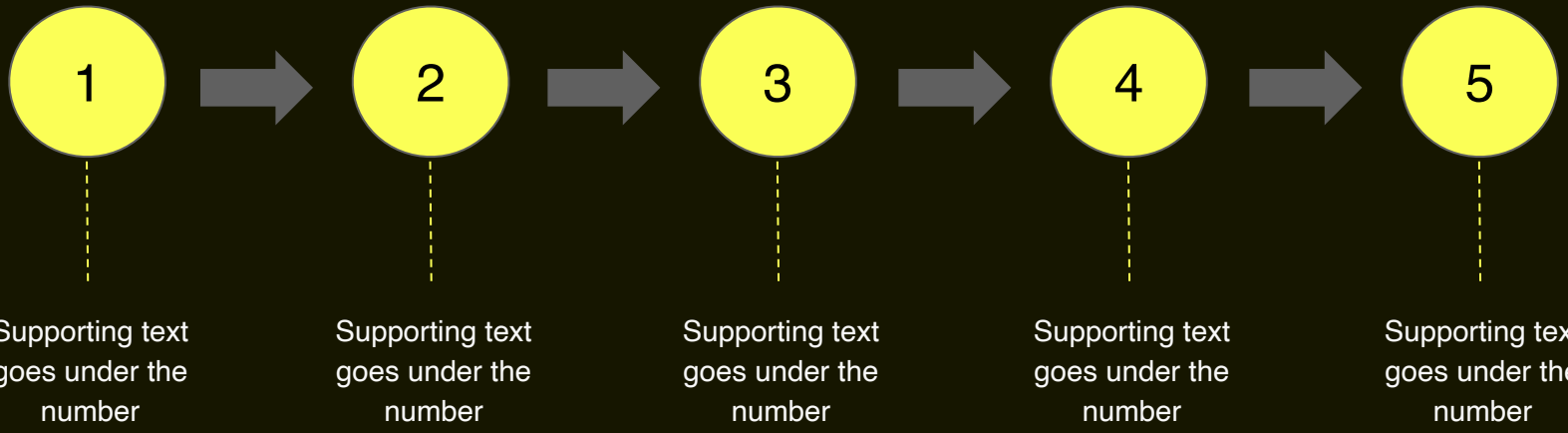
ClickHouse is simple and works out-of-the-box. Simplifies data processing by instantly processing structured data using a user-friendly SQL dialect and eliminating non-standard API requirements.

Highly reliable

Can be configured as a purely distributed system located on independent nodes, without any single points of failure. It also includes a lot of enterprise-grade security features and fail-safe mechanisms against human errors.

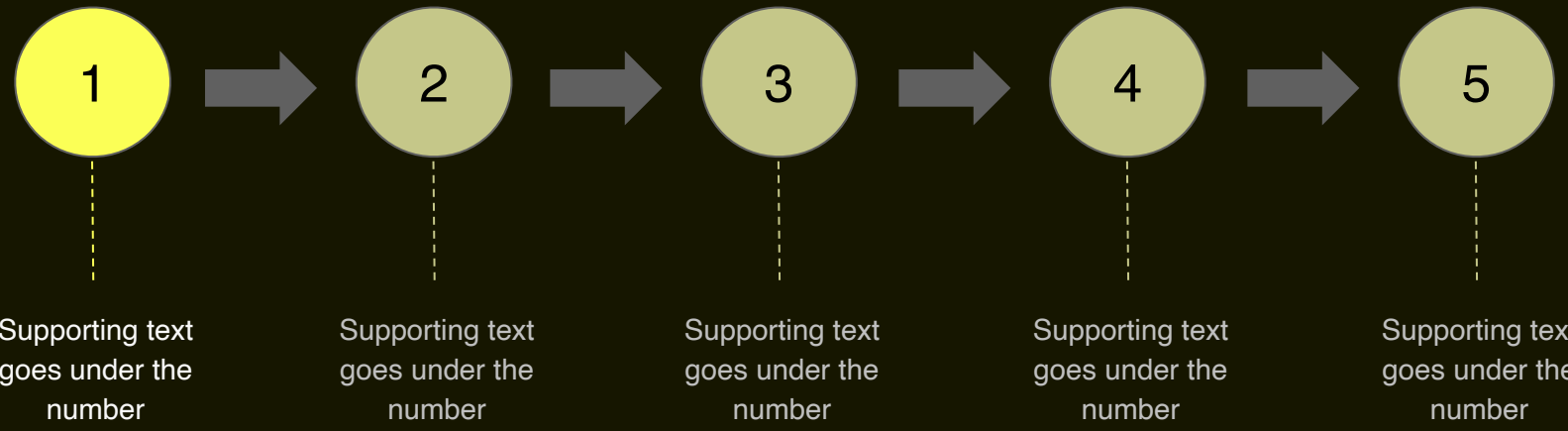
Process diagram, 5 ideas

Here you could describe the topic of the section, or not



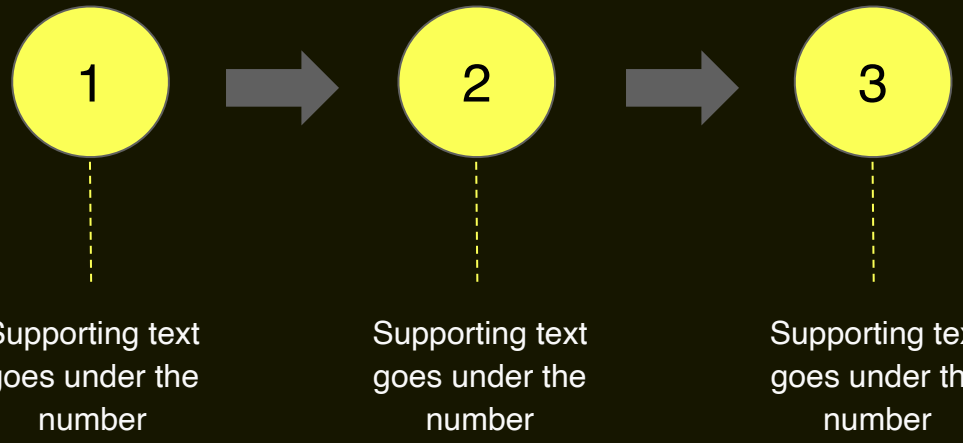
Process diagram, 5 ideas

Highlighting one of the steps



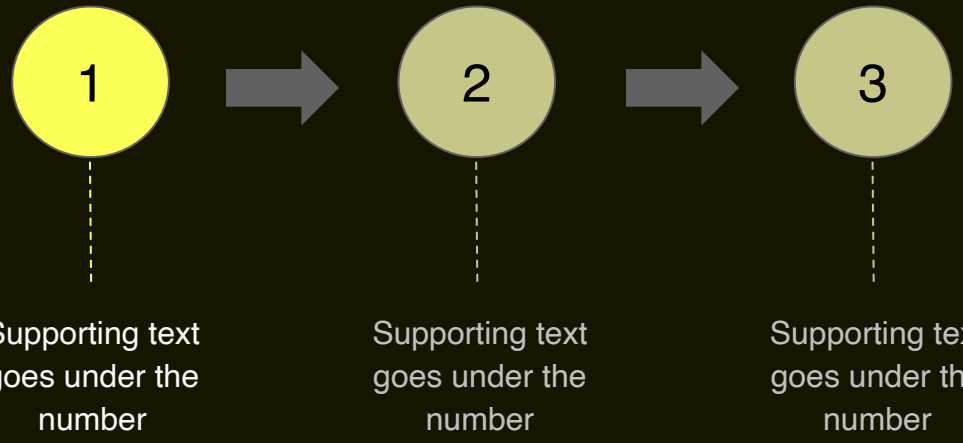
Process diagram, 3 ideas

Here you could describe the topic of the section, or not



Process diagram, 3 ideas

Highlighting one of the ideas



Big number treatment

1.2k

Header here

Supporting text goes here, under the header

1M

Header here

Supporting text goes here, under the header

45

Header here

Supporting text goes here, under the header

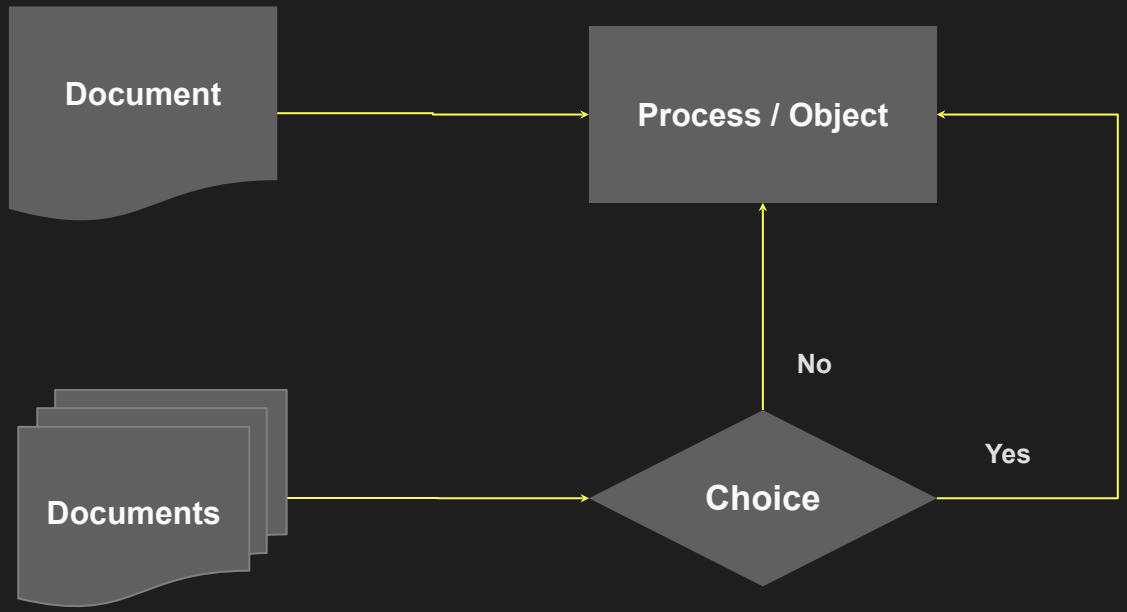
Use this slide for code

Use template colors to highlight code. Feel free to remove this paragraph.

```
SELECT
    toStartOfMonth(upload_date) AS month,
    sum(view_count) AS `Youtube Views`,
    bar(sum(has_subtitles) / count(), 0.55, 0.7, 100) AS `% Subtitles`
FROM youtube
WHERE (month >= '2020-08-01') AND (month <= '2021-08-01')
GROUP BY month
ORDER BY month ASC

13 rows in set. Elapsed: 0.823 sec Processed 1.07 billion rows, 11.75 GB (1.30 billion rows/s., 14.27 GB/s.)
```


For architecture / business process drawings



Icons for dark background