بسم الله الرحمن الرحیم

تکنولوژی کامپیوتر

جلسه‌ی بیست و دوم

معرفی اولیه‌ی CockroachDB و NewSQLها و معرفی اولیه ClickHouse

جلسه گذشته

کاساندرا

- CAP: AP
- SSTables
  - *دربارهٔ SSTable summary سوال شد.*
  - *Bloom filter*

آنچه جا ماند

- کار کردن با کاساندرا و cql
- شبیه SQL است، از داکیومنت خودتان بخوانید ☺

- One of Casandra issues is Java!
  - *Garbage collection -> unpredictable performance.*
- A rewrite with C++:
  - *ScyllaDB*
  - *Better Upper 99%*

جلسه‌ی جدید

NEW SQL

SQL Vs. NoSQL

- SQL:
  - *Usually single node or single leader databses*
  - *Lack of Scalability*
  - *With Strong safety guarantees*
    - ACID
  - *Normalized Data Model and can query with SQL*

SQL Vs. NoSQL

- NoSQL
  - *Not sql*
  - *Usually don’t have ACID guarantees*
  - *May be highly scalable*
  - *Usually denormalized data with lack of join query*

NewSQL

- Scalability of NoSQL
- With ACID guarantees

Major NewSQL Databases

- Google Spanner
- VoltDB
- TiDB
- YugabyteDB
- CockroachDB

COCKROACHDB

Trusted by enterprises for mission-critical use cases

Payments systems, IAM, logistics, user accounts, and more — powered by CockroachDB

NETFLIX BOSE Adobe yahoo! Booking.com Hewlett Packard Enterprise

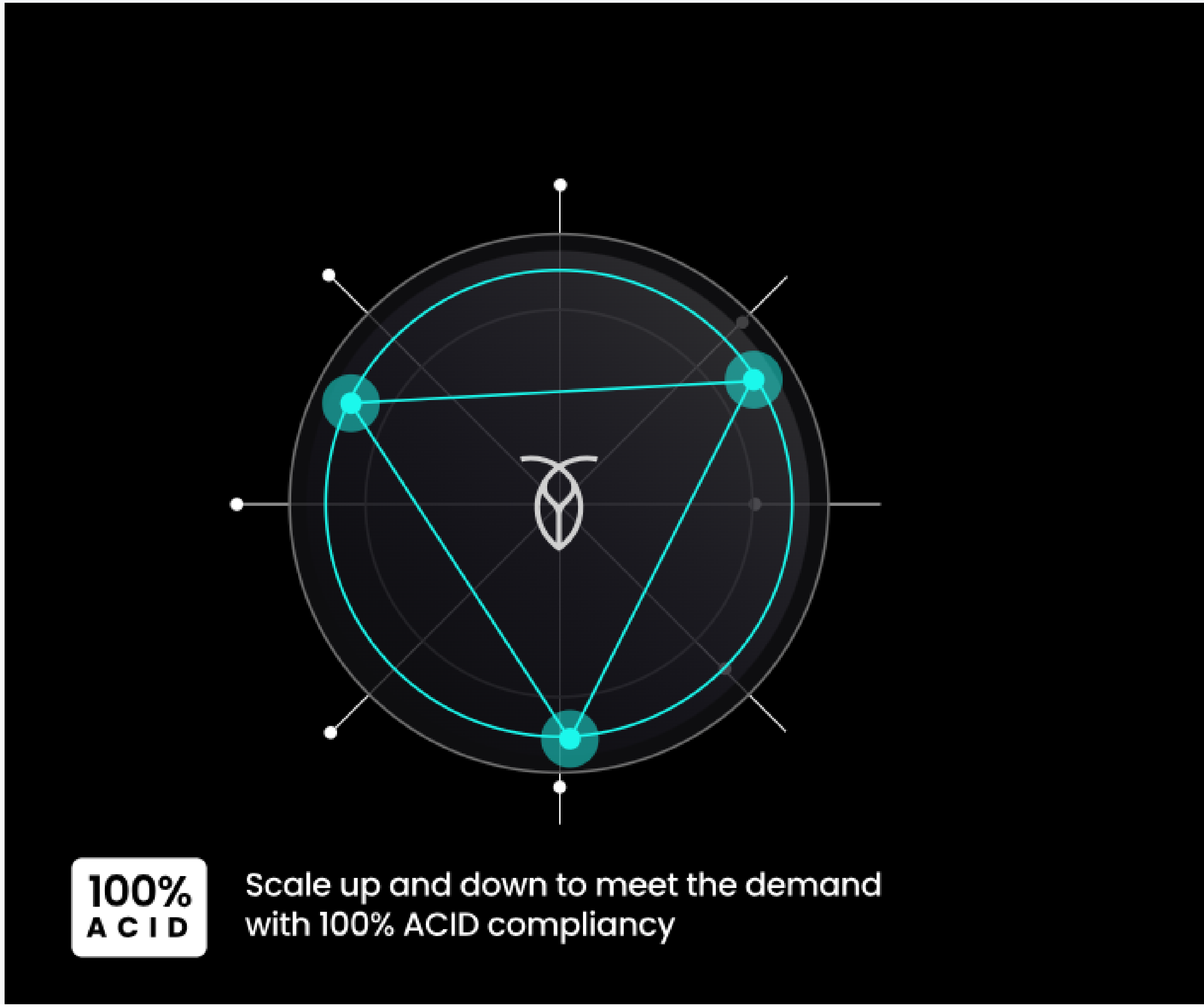
Wabtec CORPORATION Hard Rock DIGITAL FANDUEL VA U.S. Department of Veterans Affairs Continental CoreWeave

THE HOME DEPOT DOORDASH NS NORFOLK SOUTHERN sumup P&G ESTĒE LAUDER COMPANIES

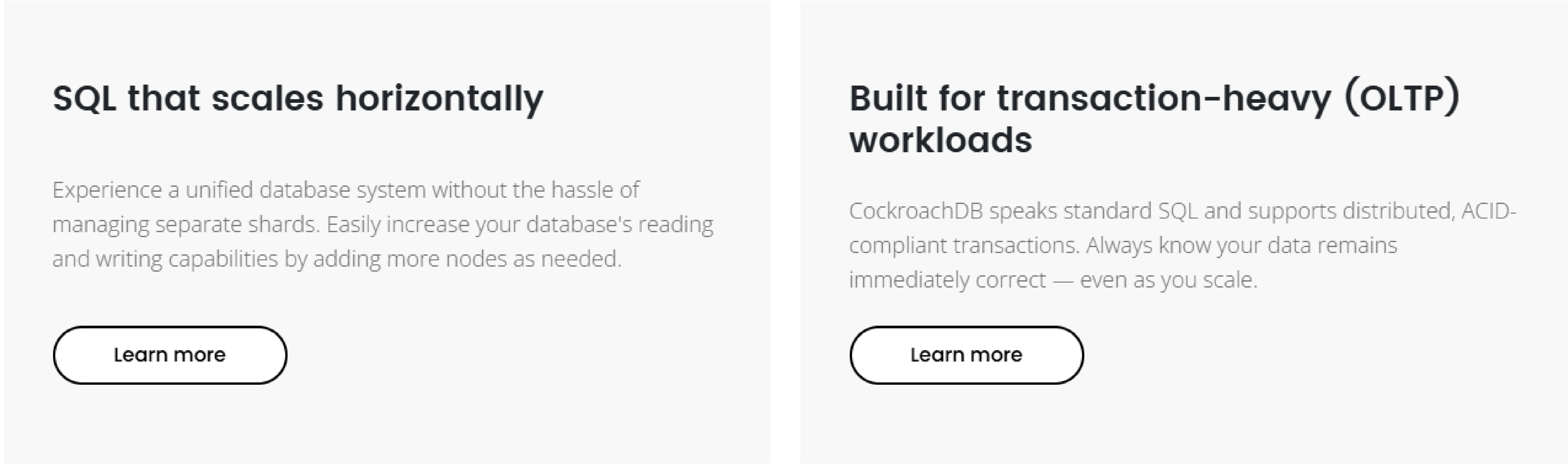
SQUARESPACE BEST BUY Five9 UiPath™ Allianz FORTINET

Scale to meet demand

Avoid database slowdowns that hinder your business growth. CockroachDB eliminates manual sharding, allowing your database to expand seamlessly as your workload grows.



Scale up and down to meet the demand with 100% ACID compliancy



SQL that scales horizontally

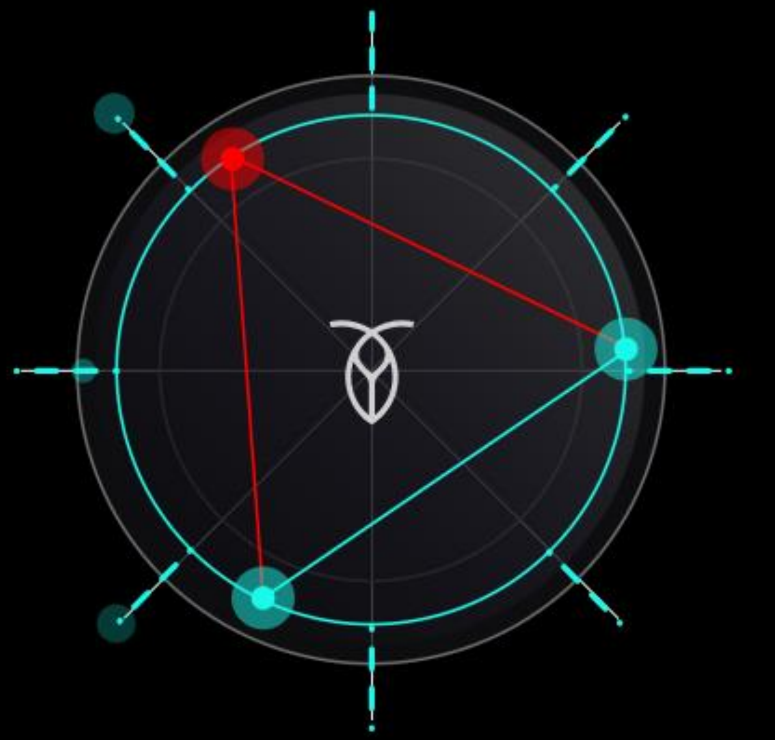
Experience a unified database system without the hassle of managing separate shards. Easily increase your database's reading and writing capabilities by adding more nodes as needed.

Built for transaction-heavy (OLTP) workloads

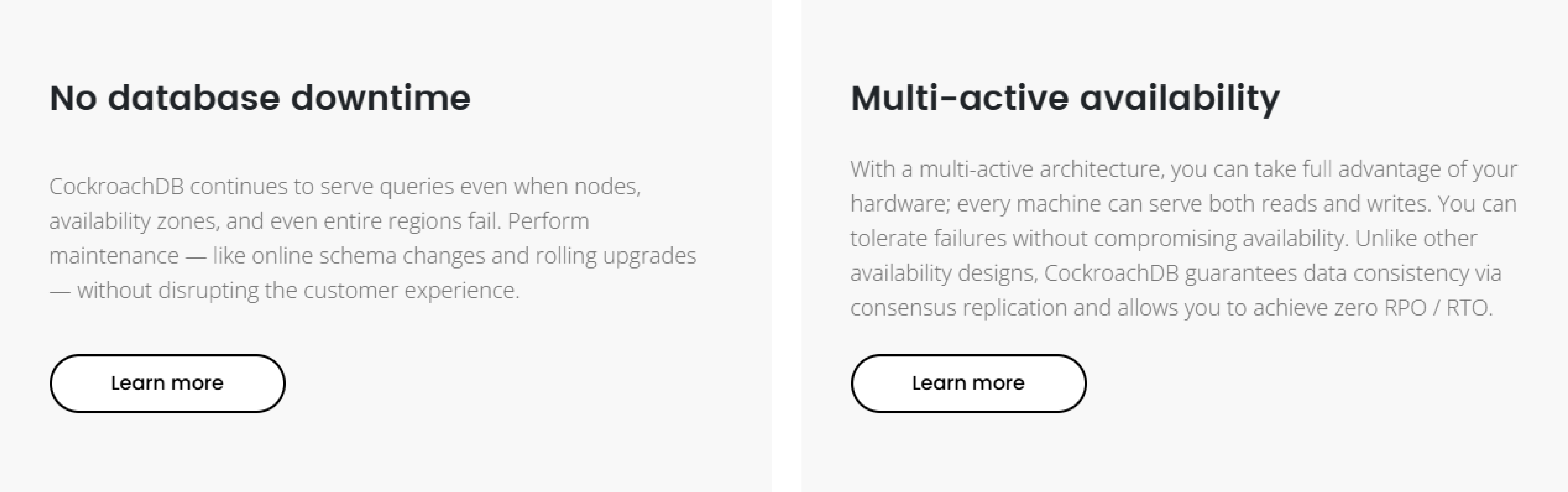
CockroachDB speaks standard SQL and supports distributed, ACID-compliant transactions. Always know your data remains immediately correct — even as you scale.

Highly available by design

Help ensure your essential applications stay online, even during cloud outages or updates. Maintain uninterrupted service for your customers, safeguarding your revenue.



With inherent high availability, infrastructure failures are nothing to fear. On CockroachDB data isn't lost and transactions keep going.

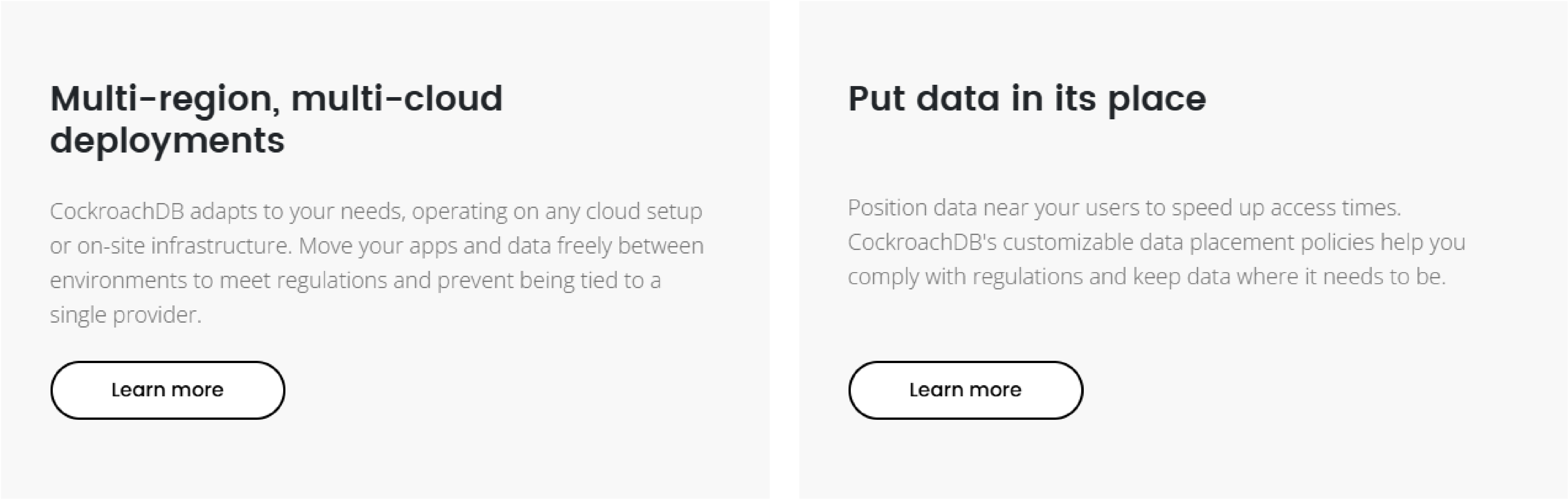


No database downtime

CockroachDB continues to serve queries even when nodes, availability zones, and even entire regions fail. Perform maintenance — like online schema changes and rolling upgrades — without disrupting the customer experience.

Multi-active availability

With a multi-active architecture, you can take full advantage of your hardware; every machine can serve both reads and writes. You can tolerate failures without compromising availability. Unlike other availability designs, CockroachDB guarantees data consistency via consensus replication and allows you to achieve zero RPO / RTO.



Multi-region, multi-cloud deployments

CockroachDB adapts to your needs, operating on any cloud setup or on-site infrastructure. Move your apps and data freely between environments to meet regulations and prevent being tied to a single provider.

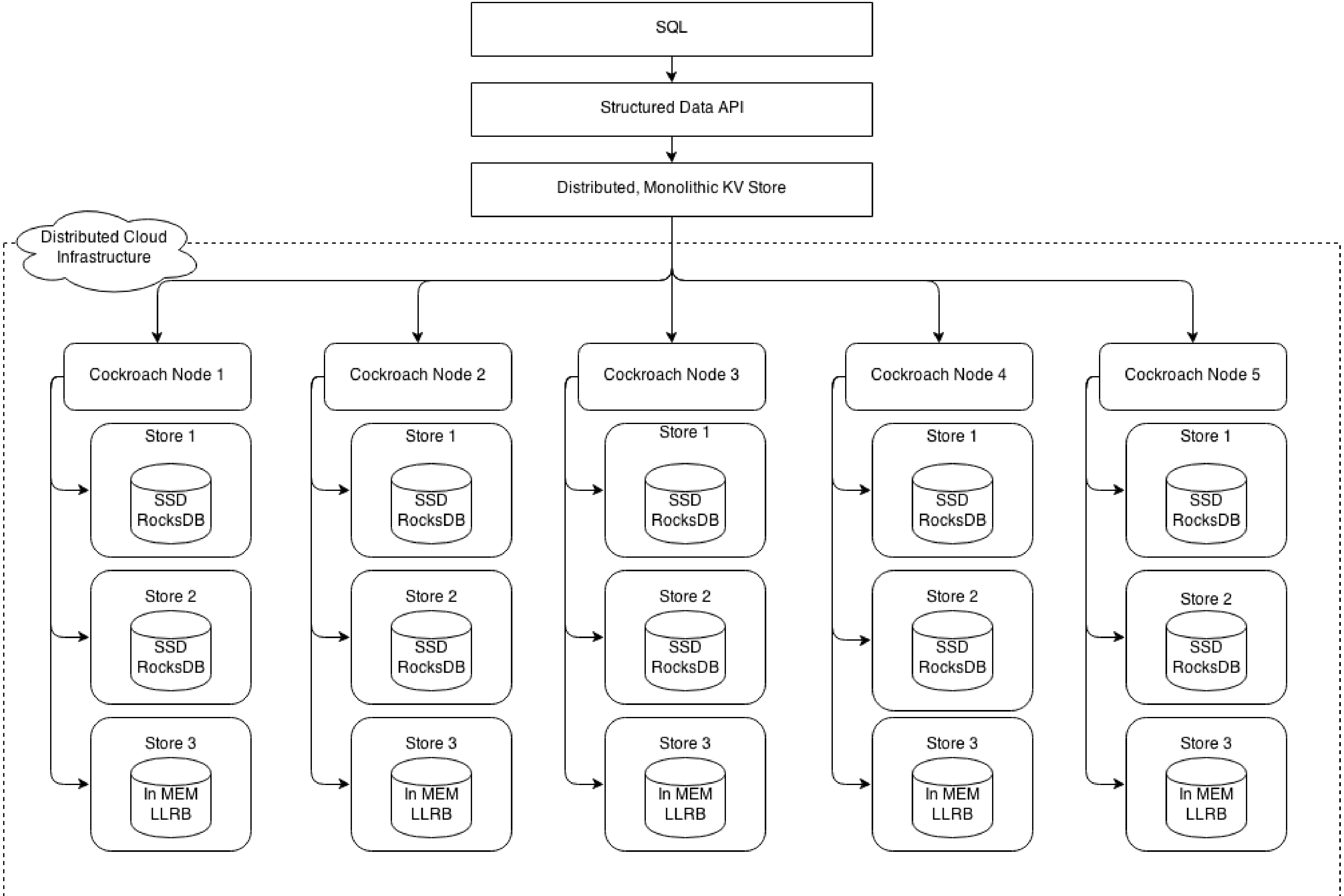
Put data in its place

Position data near your users to speed up access times. CockroachDB's customizable data placement policies help you comply with regulations and keep data where it needs to be.

Postgresql-compatible

CAP Theorem

- CP

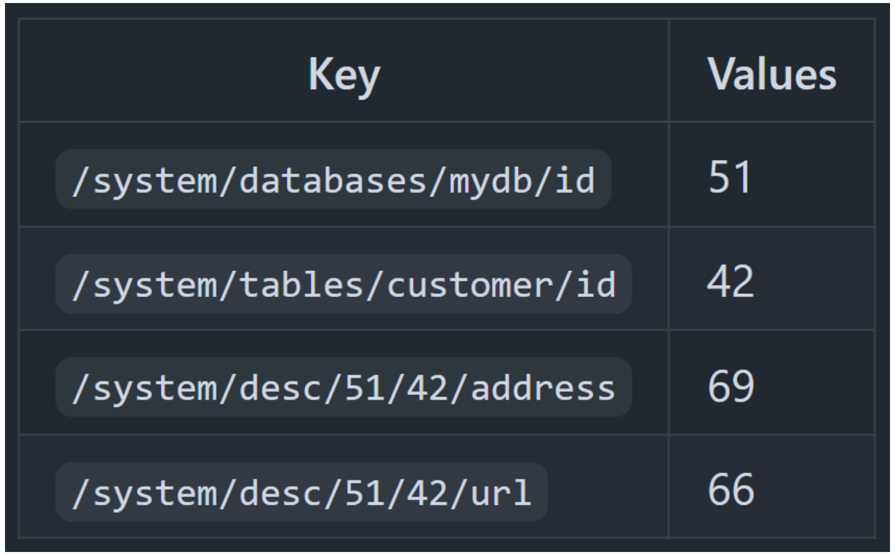

SQL
Structured Data API
Distributed, Monolithic KV Store
Distributed Cloud Infrastructure
Cockroach Node 1
Store 1
SSD RocksDB
Store 2
SSD RocksDB
Store 3
In MEM LLRB
Cockroach Node 2
Store 1
SSD RocksDB
Store 2
SSD RocksDB
Store 3
In MEM LLRB
Cockroach Node 3
Store 1
SSD RocksDB
Store 2
SSD RocksDB
Store 3
In MEM LLRB
Cockroach Node 4
Store 1
SSD RocksDB
Store 2
SSD RocksDB
Store 3
In MEM LLRB
Cockroach Node 5
Store 1
SSD RocksDB
Store 2
SSD RocksDB
Store 3
In MEM LLRB

Range KV Queries

- Why ranges?
- On RocksDB (LSM Tree / SSTable)
- Bloom filter on Range?!

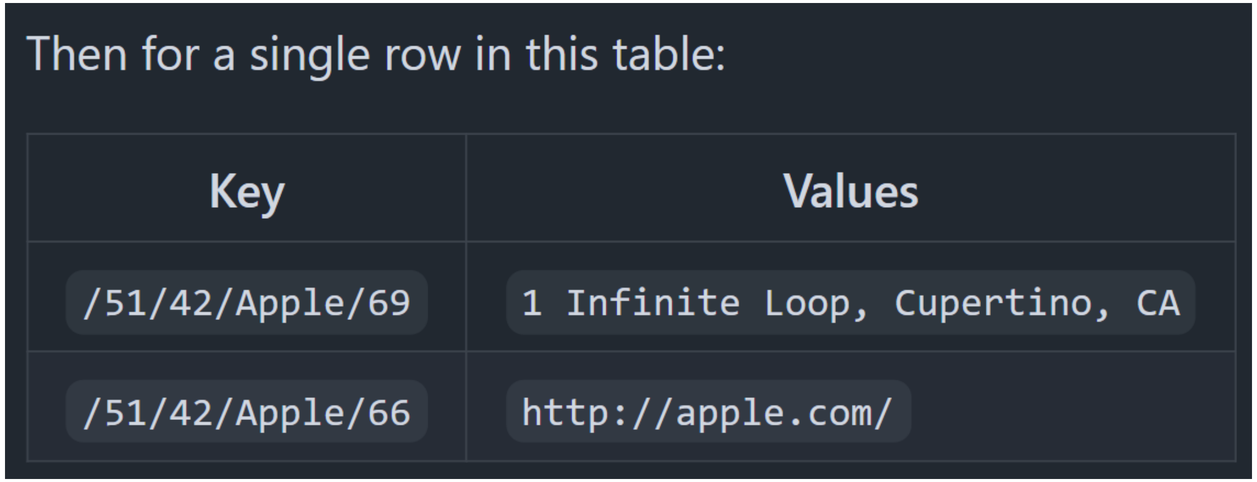
SQL to KV Queries

- Example:
  - *after table customers is created in a database mydb with a primary key column name and normal columns address and URL, the KV pairs to store the schema would be:*



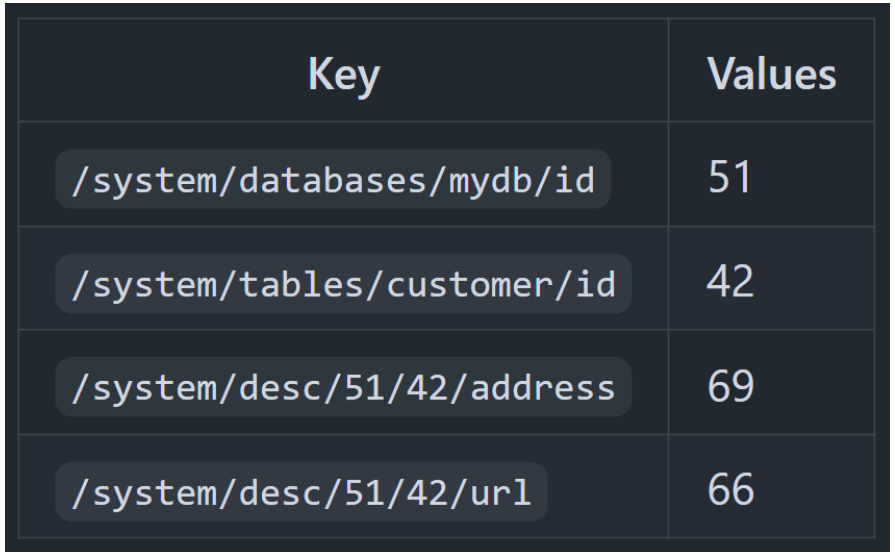
Key	Values
`/system/databases/mydb/id`	51
`/system/tables/customer/id`	42
`/system/desc/51/42/address`	69
`/system/desc/51/42/url`	66

SQL to KV Queries



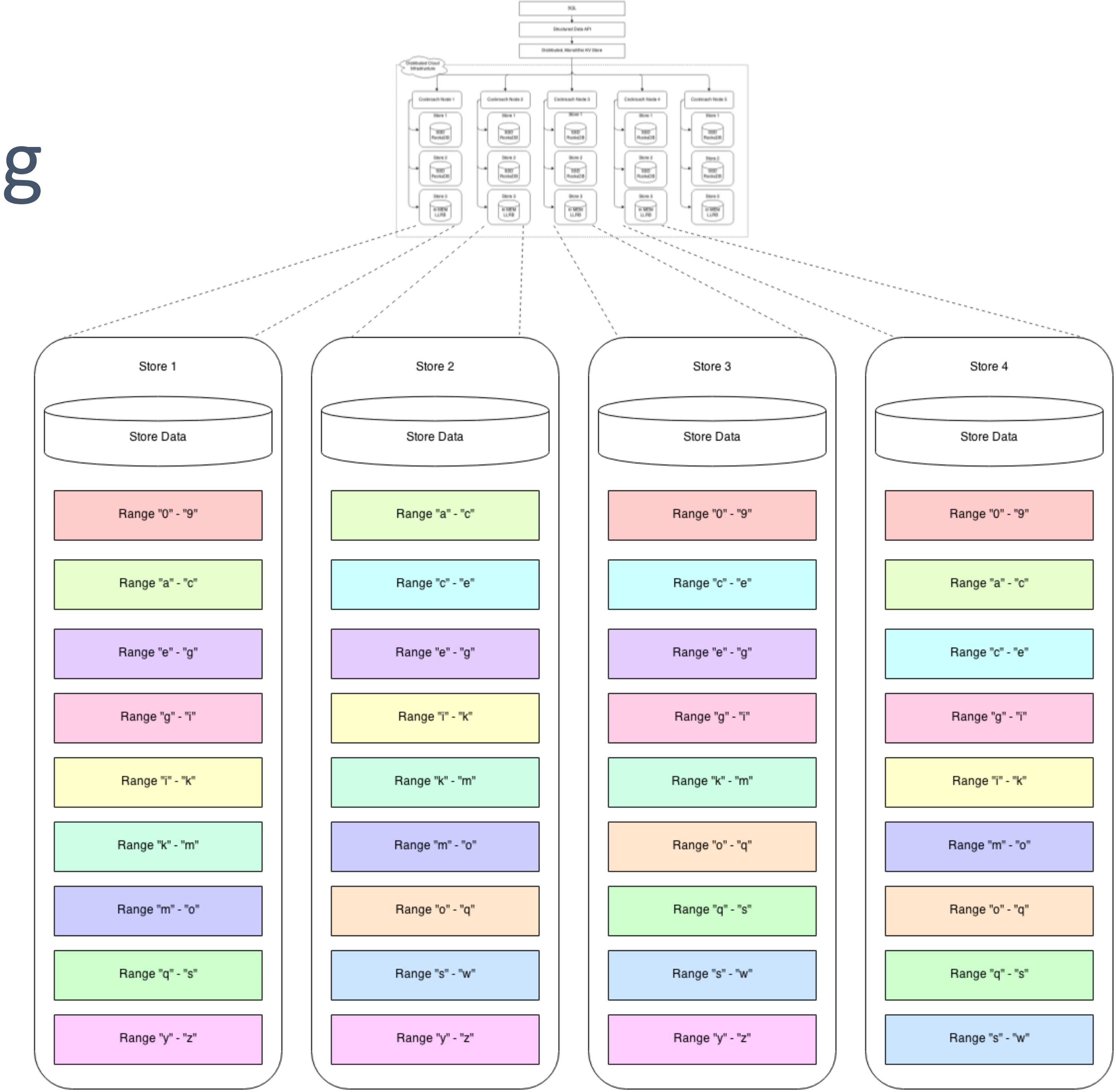
Then for a single row in this table:

Key	Values
`/51/42/Apple/69`	`1 Infinite Loop, Cupertino, CA`
`/51/42/Apple/66`	`http://apple.com/`



Key	Values
`/system/databases/mydb/id`	51
`/system/tables/customer/id`	42
`/system/desc/51/42/address`	69
`/system/desc/51/42/url`	66

Partitioning



Store 1
Store Data
Range "0" - "9"
Range "a" - "c"
Range "e" - "g"
Range "g" - "i"
Range "i" - "k"
Range "k" - "m"
Range "m" - "o"
Range "q" - "s"
Range "y" - "z"

Store 2
Store Data
Range "a" - "c"
Range "c" - "e"
Range "e" - "g"
Range "i" - "k"
Range "k" - "m"
Range "m" - "o"
Range "o" - "q"
Range "s" - "w"
Range "y" - "z"

Store 3
Store Data
Range "0" - "9"
Range "c" - "e"
Range "e" - "g"
Range "g" - "i"
Range "k" - "m"
Range "o" - "q"
Range "q" - "s"
Range "s" - "w"
Range "y" - "z"

Store 4
Store Data
Range "0" - "9"
Range "a" - "c"
Range "c" - "e"
Range "g" - "i"
Range "i" - "k"
Range "m" - "o"
Range "o" - "q"
Range "q" - "s"
Range "s" - "w"

Partitioning

- How each node determines where is a range?
- We have two special ranges!
  - *Meta1 – the root range*
    - Broadcast the location using gossip
    - Contains the location of meta2
  - *Meta2*
    - Contains the location of other ranges

Range size?

- Ranges may split / merge
- Each range has a raft group for consensus

Lock free serializable

- SSI

Lock free serializable

- Hybrid Logical Clock
  - *Wall clock + Lamport clock*
- Transaction timestamp for read and write
- 250ms clock maximum skew

Distributed Transaction

- A version of two phase commit with Raft consensus

Membership and Node allocation

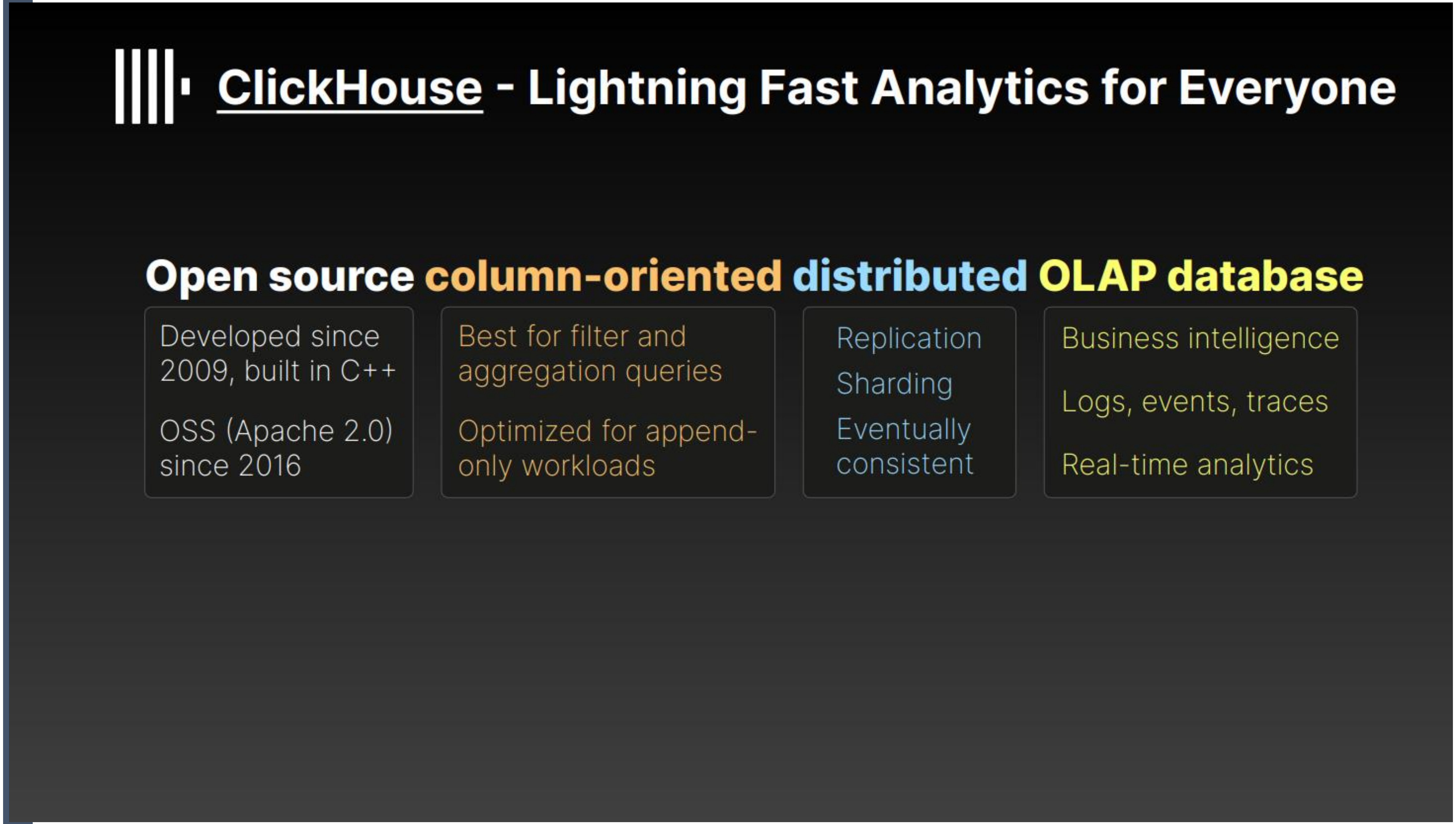
- Gossip protocol (eventual consistent)!
  - *Where is meta1 range*
  - *Data of each node – free space, load and …*
- But the data is strong consistent!

Distributed SQL

CockroachDB Vs. Postgresql

- Choose Postgresql when:
  - *You need an exotic index, FDW federation, LISTEN/NOTIFY, RLS, or a specific extension.*
  - *Low complexity*
  - *A very mature software*
  - *Your data fit on a single node.*
- Choose CockroachDB when:
  - *You need global, horizontally-scalable OLTP with zero-touch failover and strong consistency.*

CLICKHOUSE AND COLUMNAR DBS



ClickHouse - Lightning Fast Analytics for Everyone

Open source column-oriented distributed OLAP database

Developed since 2009, built in C++

OSS (Apache 2.0) since 2016

Best for filter and aggregation queries

Optimized for append-only workloads

Replication

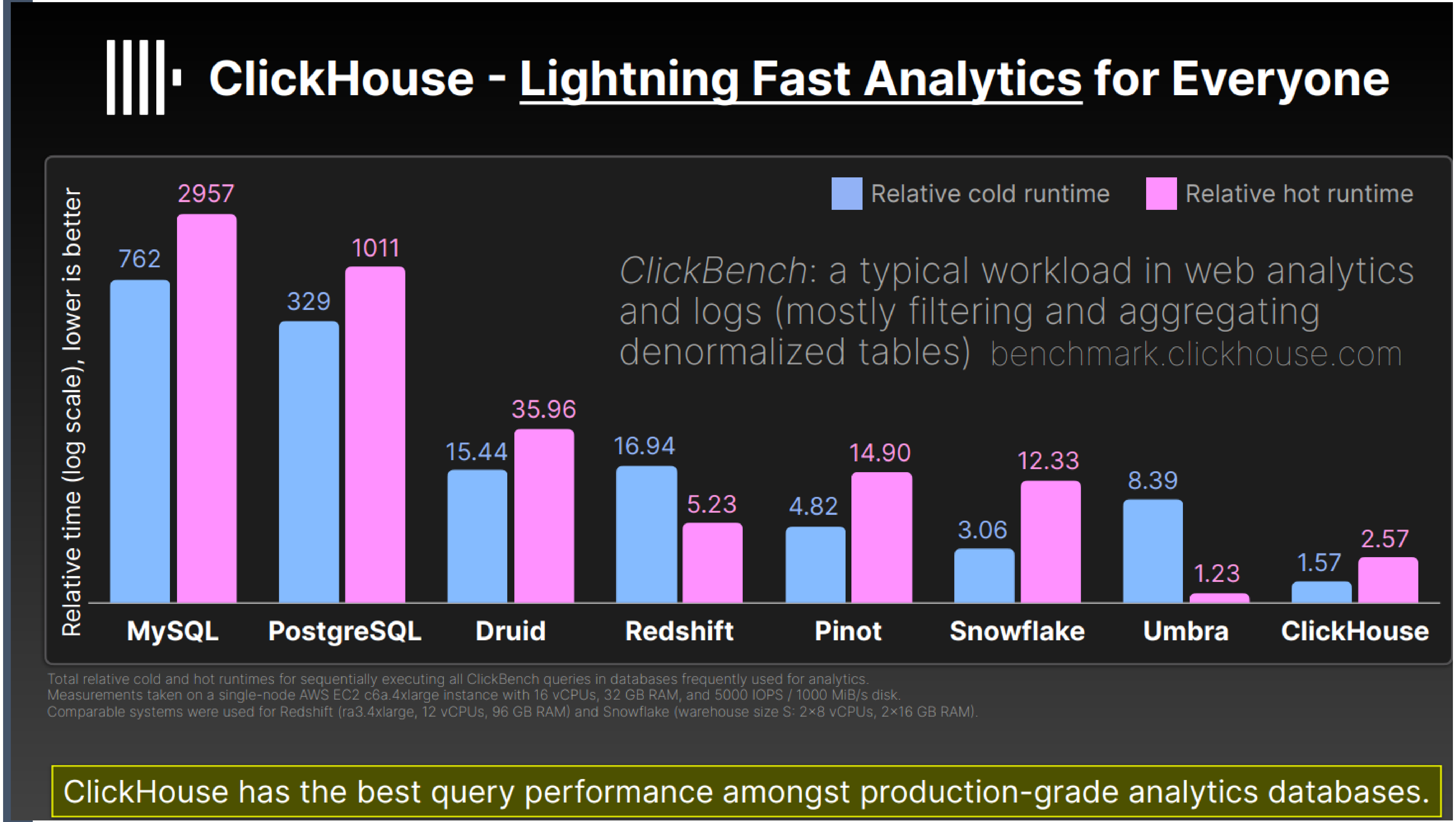
Sharding

Eventually consistent

Business intelligence

Logs, events, traces

Real-time analytics



ClickHouse - Lightning Fast Analytics for Everyone

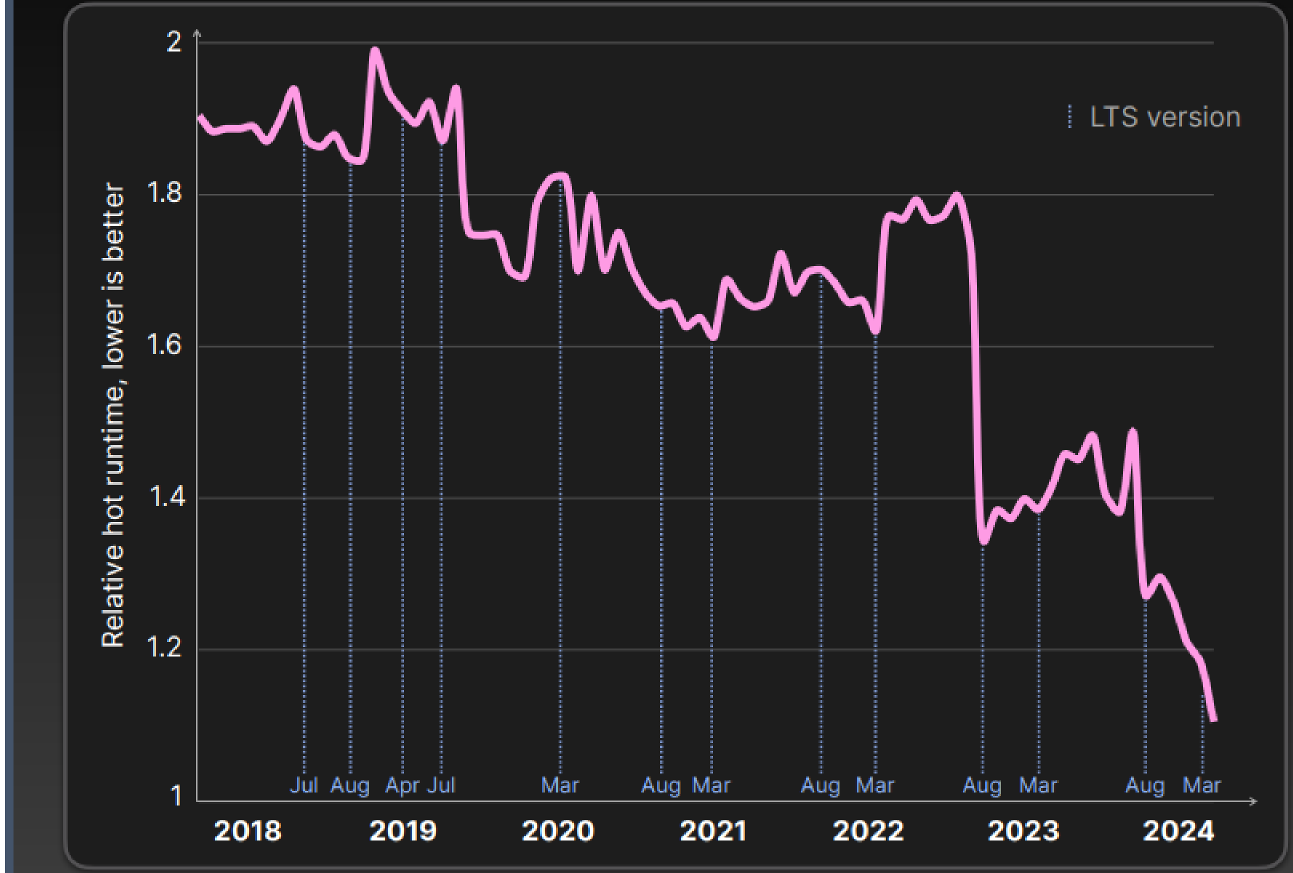
ClickBench: a typical workload in web analytics and logs (mostly filtering and aggregating denormalized tables) benchmark.clickhouse.com

Total relative cold and hot runtimes for sequentially executing all ClickBench queries in databases frequently used for analytics.
Measurements taken on a single-node AWS EC2 c6a.4xlarge instance with 16 vCPUs, 32 GB RAM, and 5000 IOPS / 1000 MiB/s disk.
Comparable systems were used for Redshift (ra3.4xlarge, 12 vCPUs, 96 GB RAM) and Snowflake (warehouse size S: 2×8 vCPUs, 2×16 GB RAM).

ClickHouse has the best query performance amongst production-grade analytics databases.

ClickHouse - Lightning Fast Analytics for Everyone

Performance improvements by 1.72 × since 2018



- *VersionBench* benchmark is run when a new release is published to check its performance and identify regressions.
- Combination of four benchmarks:

	# Queries	# Rows
ClickBench	42	100 million
MgBench	15	200 million
Star Schema Benchmark (denormalized schema)	13	600 million
NYC Taxi Rides Benchmark	4	3.4 billion

Query performance is a top priority and continuously improved.

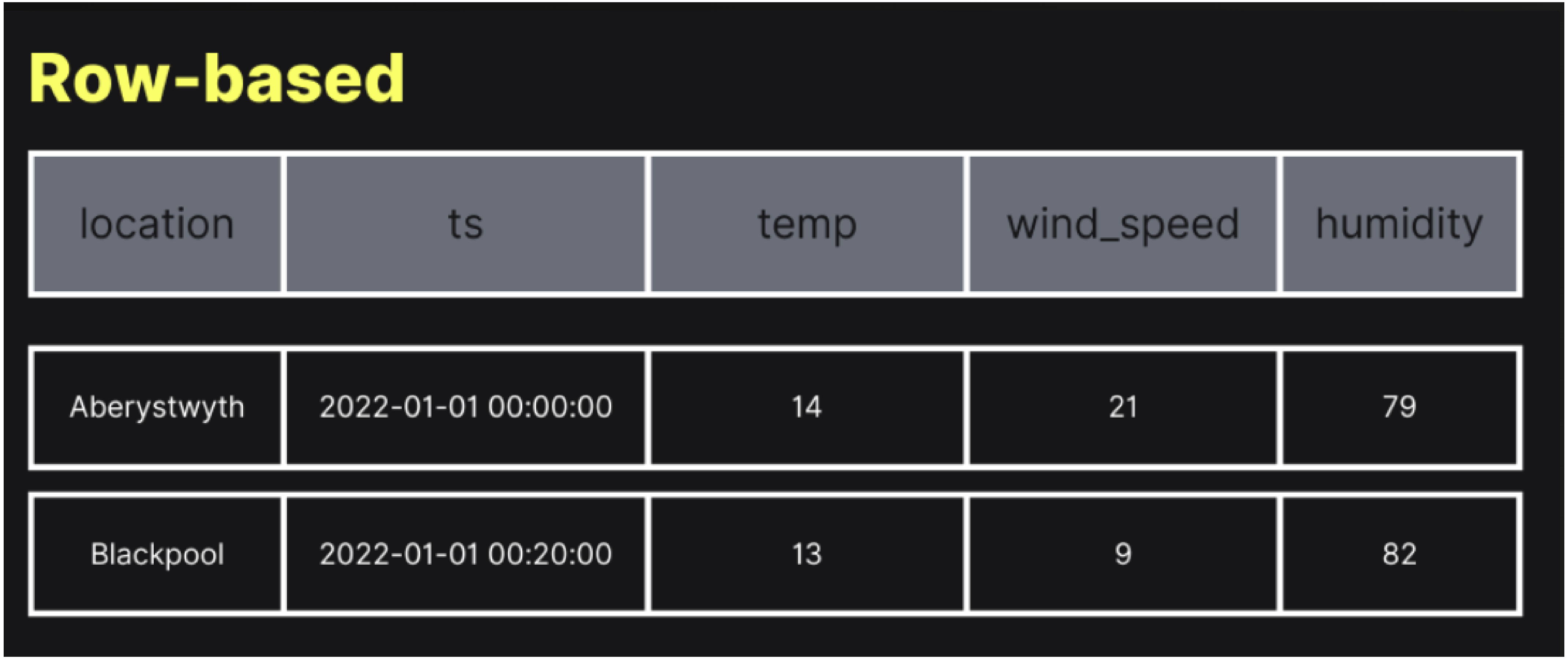
OLAP

- OLTP: Online Transaction Processing
  - *Handles day-to-day transactional operations*
  - *Manages real-time data updates and insertions*
  - *Ensures ACID (Atomicity, Consistency, Isolation, Durability) compliance*

OLAP

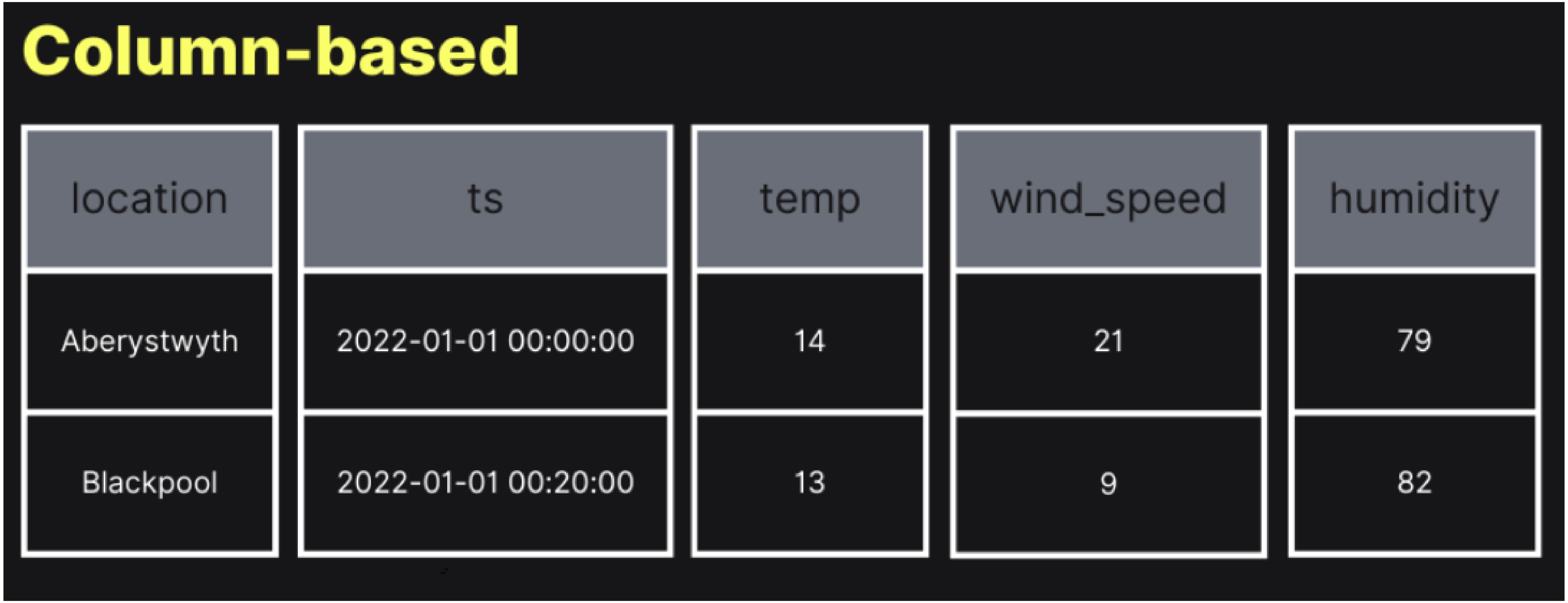
- OLAP: Online Analytical Processing
  - *Manages large-scale data analytics*
  - *Handles analytical queries on historical and real-time data*
  - *Optimizes for read-heavy workloads and aggregations*

Row oriented Vs. Columnar DB



Row-based

location	ts	temp	wind_speed	humidity
Aberystwyth	2022-01-01 00:00:00	14	21	79
Blackpool	2022-01-01 00:20:00	13	9	82



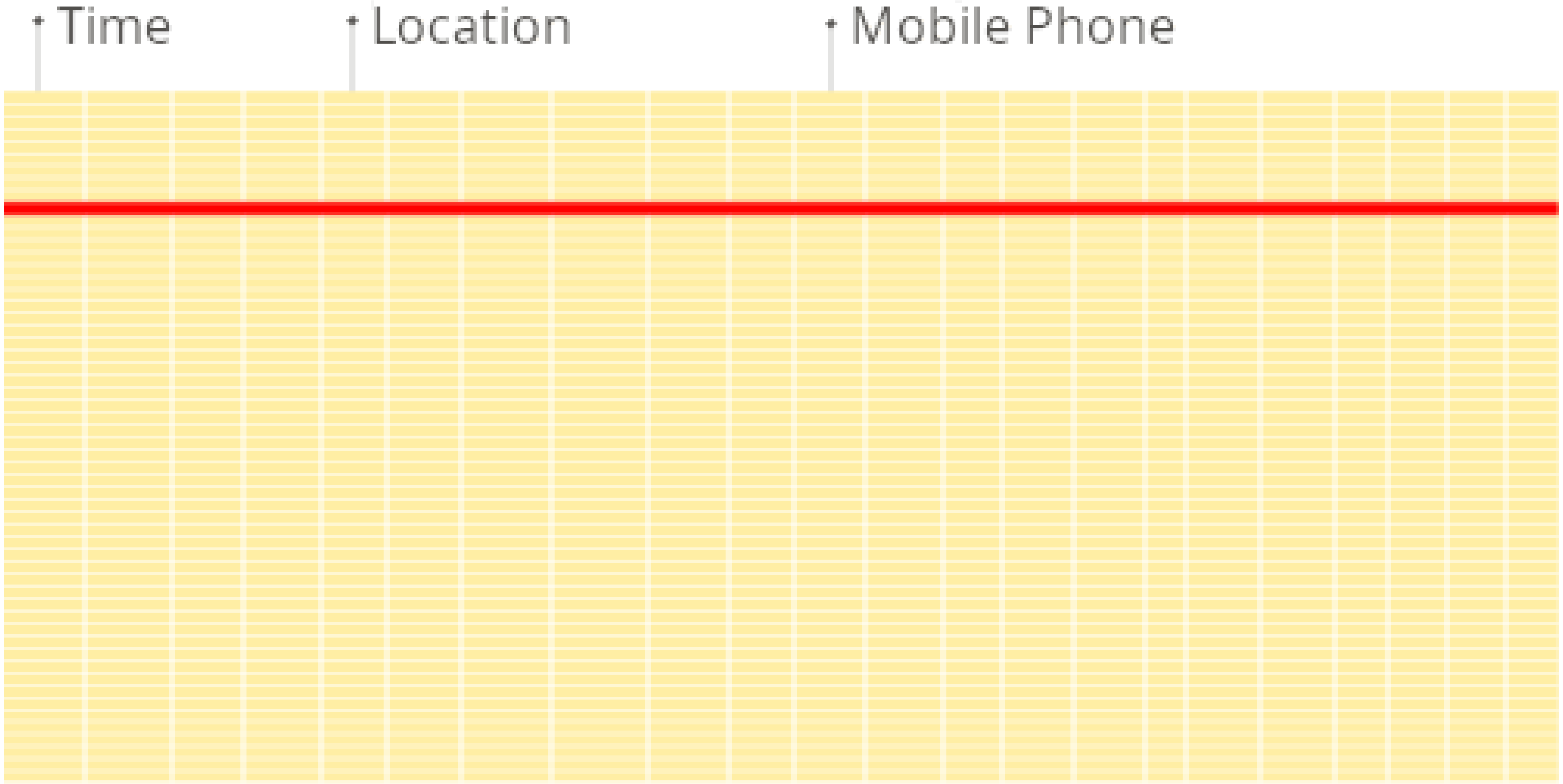
Column-based

location	ts	temp	wind_speed	humidity
Aberystwyth	2022-01-01 00:00:00	14	21	79
Blackpool	2022-01-01 00:20:00	13	9	82

- Example,
  - *Columns:*
  - *columns: ['WatchID','JavaEnable','Title','GoodEvent','EventTime','EventDate','CounterID','ClientIP','RegionID','UserID','CounterClass','OS','UserAgent','URL','Referer','Refresh','RefererCategoryID','RefererRegionID','URLCategoryID','URLRegionID','ResolutionWidth','ResolutionHeight','ResolutionDepth','FlashMajor','FlashMinor','FlashMinor2','NetMajor','NetMinor','UserAgentMajor','UserAgentMinor','CookieEnable','JavascriptEnable','IsMobile','MobilePhone','MobilePhoneModel','Params','IPNetworkID','TraficSourceID','SearchEngineID','SearchPhrase','AdvEngineID','IsArtifical','WindowClientWidth','WindowClientHeight','ClientTimeZone','ClientEventTime','SilverlightVersion1','SilverlightVersion2','SilverlightVersion3','SilverlightVersion4','PageCharset','CodeVersion','IsLink','IsDownload','IsNotBounce','FUniqID','OriginalURL','HID','IsOldCounter','IsEvent','IsParameter','DontCountHits','WithHash','HitColor','LocalEventTime','Age','Sex','Income','Interests','Robotness','RemoteIP','WindowName','OpenerName','HistoryLength','BrowserLanguage','BrowserCountry','SocialNetwork','SocialAction','HTTPError','SendTiming','DNSTiming','ConnectTiming','ResponseStartTiming','ResponseEndTiming','FetchTiming','SocialSourceNetworkID','SocialSourcePage','ParamPrice','ParamOrderID','ParamCurrency','ParamCurrencyID','OpenstatServiceName','OpenstatCampaignID','OpenstatAdID','OpenstatSourceID','UTMSource','UTMMedium','UTMCampaign','UTMContent','UTMTerm','FromTag','HasGCLID','RefererHash','URLHash','CLID']*

```
SELECT MobilePhoneModel, COUNT() AS c
FROM metrica.hits
WHERE
      RegionID = 229
  AND EventDate >= '2013-07-01'
  AND EventDate <= '2013-07-31'
  AND MobilePhone != 0
  AND MobilePhoneModel not in ['', 'iPad']
GROUP BY MobilePhoneModel
ORDER BY c DESC
LIMIT 8;
```

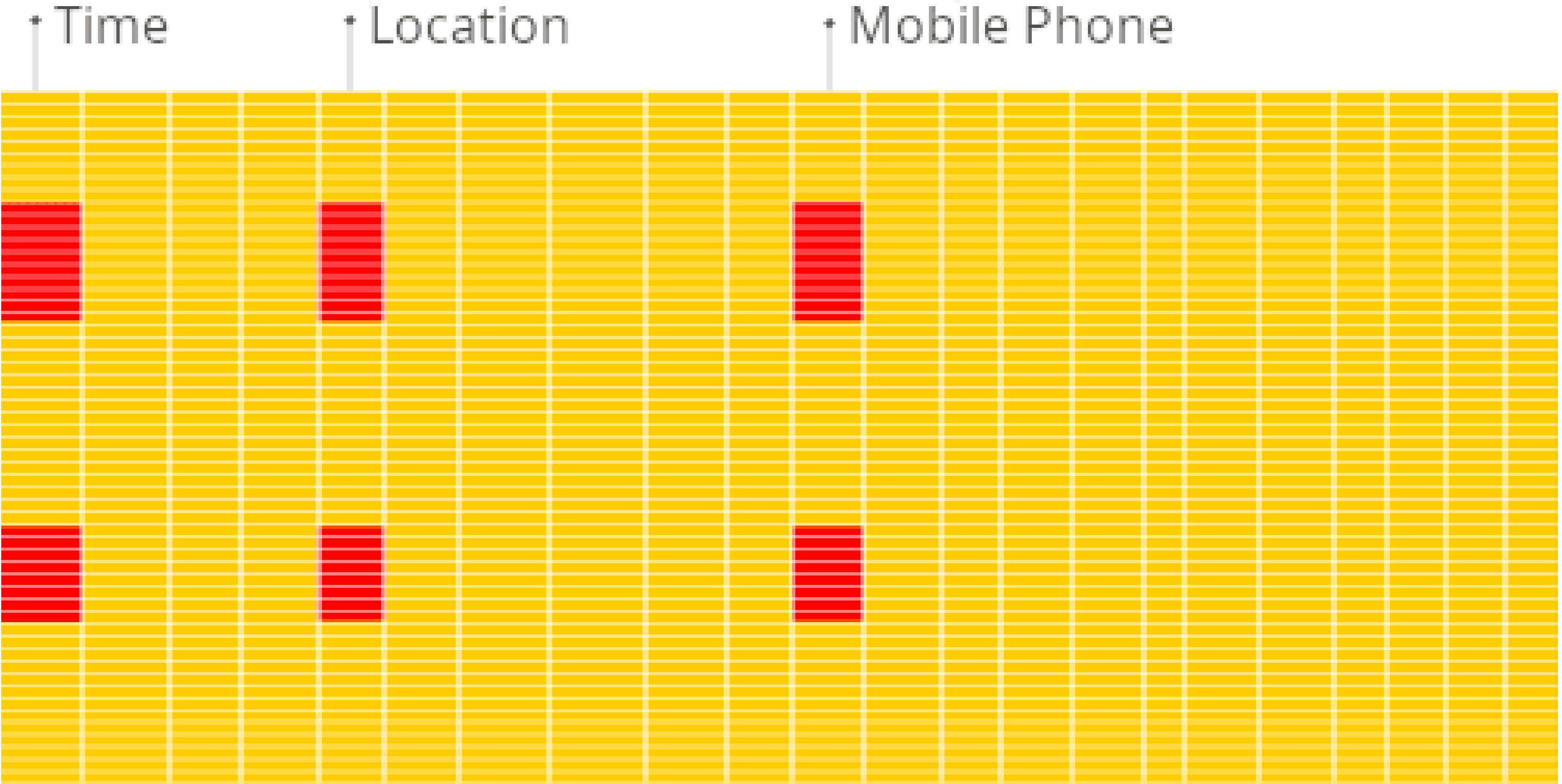
```
SELECT *
FROM metrica.hits
WHERE WatchID = 8120543446287442873;
```



Time

Location

Mobile Phone


Time
Location
Mobile Phone

Storage Layer: Concurrent inserts and selects are isolated

Storage Layer: Merge-time computation

- Replacing merges which retain only the most recent version of a row in the input parts and discard all other row versions. Replacing merges can be thought of as a merge-time cleanup operation.
- Aggregating merges which combine intermediate aggregation states in the input part to a new aggregation state. While this seems difficult to understand, it really actually only implements an incremental aggregation.
- TTL (time-to-live) merges compress, move, or delete rows based on certain time-based rules.

Storage Layer: Data pruning

- Primary key indexes which define the sort order of the table data. A well-chosen primary key allows to evaluate filters (like the WHERE clauses in the above query) using fast binary searches instead of full-column scans. In more technical terms, the runtime of scans becomes logarithmic instead of linear in the data size.
- Table projections as alternative, internal versions of a table, storing the same data but sorted by a different primary key. Projections can be useful when there is more than one frequent filter condition.
- Skipping indexes that embed additional data statistics into columns, e.g. the minimum and maximum column value, the set of unique values, etc. Skipping indexes are orthogonal to primary keys and table projections, and depending on the data distribution in the column, they can greatly speed up the evaluation of filters.

Meticulous attention to detail