TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

NewSQL Database: Cockroach DB

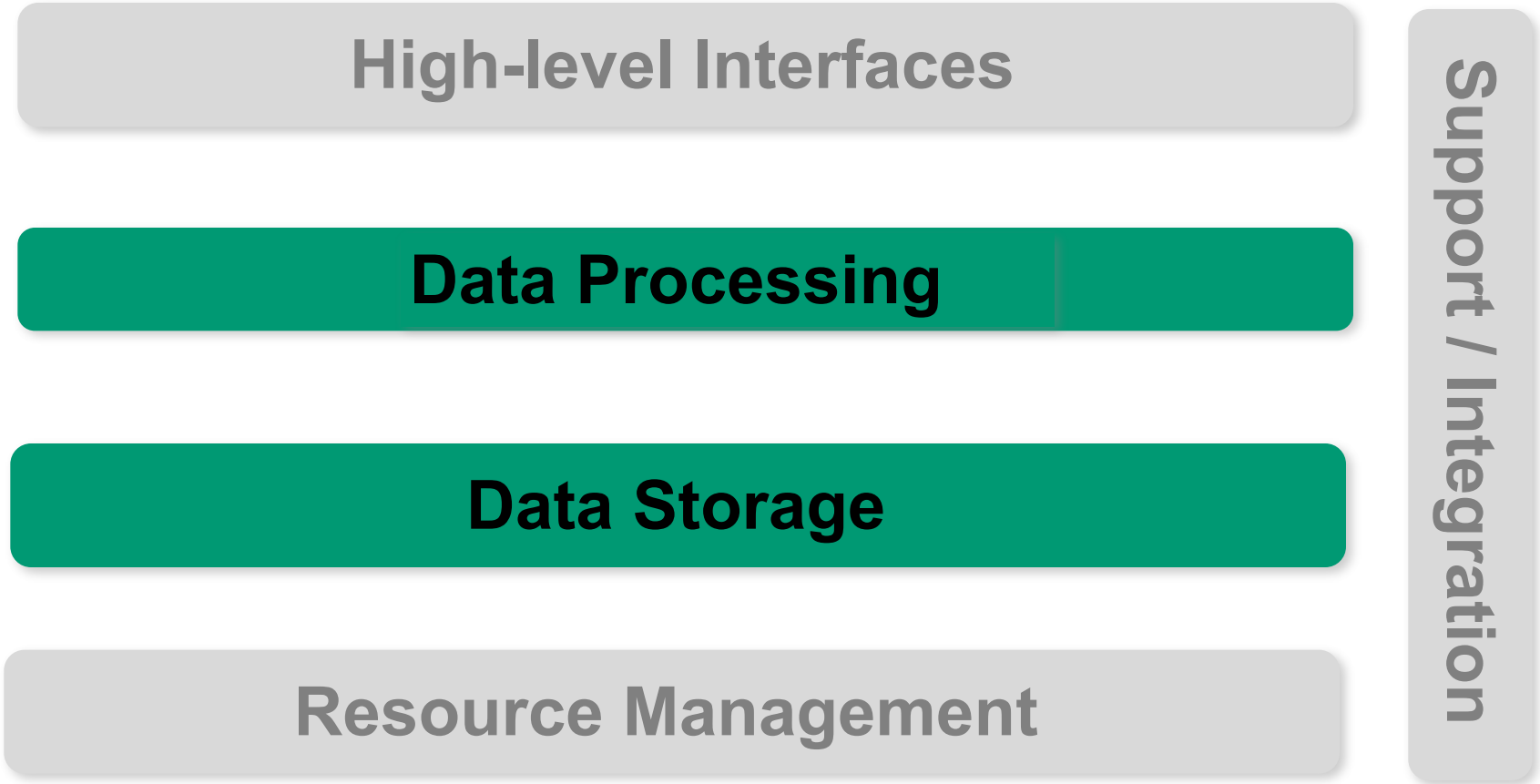
Corso di Sistemi e Architetture per Big Data

A.A. 2023/24

Matteo Nardelli

Laurea Magistrale in Ingegneria Informatica

The reference Big Data stack



Relational database systems

- RDBMS pros:
  - ACID transactions
  - Relational schemas (and schema changes without downtime)
  - SQL queries
  - Strong consistency

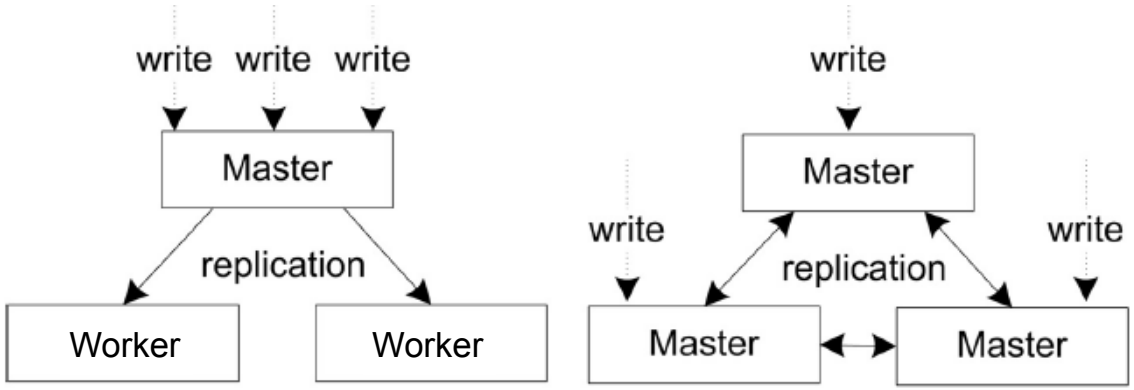
- RDBMS cons:
  - Lack of horizontal scalability (to 100s or 1000s of servers)

NewSQL databases

- **NewSQL**: a class of modern RDBMS
- Goals
  - Provide scalability of NoSQL systems for OLTP workloads, while maintaining ACID support of traditional RDBMS
  - Support SQL
- Examples (mostly closed source)
  - Google's Spanner
  - CockroachDB (Open-source, born as Spanner clone, then evolved differently)
  - VoltDB
  - MariaDB Xpand
  - NuoDB

Replication in NewSQL

- Hot to scale? Multi-master (or master-less) schemes
  - Any node can receive data update statements



Primary-based replication

Multi-master replication

Cockroach DB: Overview

- NewSQL a.k.a. distributed SQL database
  - Scalability
  - Strong consistency
  - Survivability: tolerate disk, machine, rack, datacenter failures with minimal latency disruption and no manual intervention
- Multi-master architecture
  - Each node acts as SQL gateway:
    - Transforms and executes SQL statements to key-value (KV) operations;
    - Distributes KV operations across the cluster and returns results to the client
- CockroachDB is considered a CP system under the CAP theorem.

Read more: https://rcs.uwaterloo.ca/~ali/cs854-f23/papers/cockroachdb.pdf

Cockroach DB: Overview

- Internal data model
  - Single, sorted map from key to value;
  - Map is divided into **ranges**;
  - Range is stored in a local KV storage engine (Pebble) and replicated to additional nodes;
    - Pebble is an embedded KV inspired by RocksDB and developed by Cockroach Labs
  - Ranges are merged and split to maintain target size (e.g., 64MB)
- Horizontal Scalability
  - Ranges of the same key can be stored on different nodes;
  - Adding nodes increases storage capacity and overall throughput of queries;

Cockroach DB: Overview

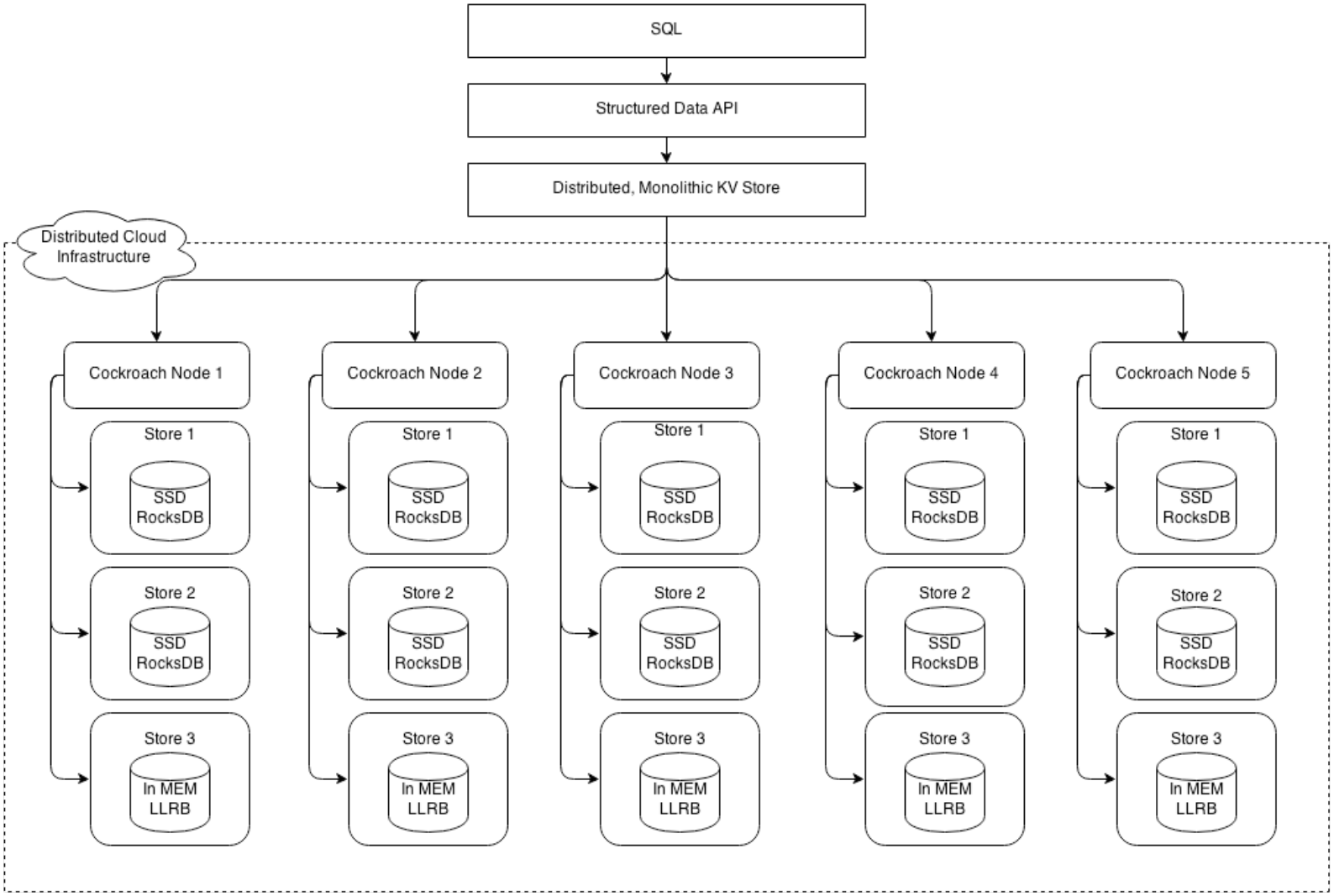
- Strong consistency
  - Distributed consensus (Raft) for synchronous replication in each KV range;
  - Mutations across multiple ranges employ **distributed transactions**
  - Raft and distributed transactions guarantee ACID properties
- Fault-tolerance
  - Range replicas can be:
    - co-located within a single data center for low latency;
    - distributed across racks (survive to (some) network failures);
    - Distributed across different data centers.

Cockroach DB: Overview

- CockroachDB's architecture is organized into layers:
  - **SQL**: translates SQL queries to KV operations;
  - **Transactional**: Allow atomic changes to multiple KV entries;
  - **Distribution**: Present replicated KV ranges as a single entity;
  - **Replication**: Consistently and synchronously replicate KV ranges across nodes;
  - **Storage**: read and write KV data on disk.

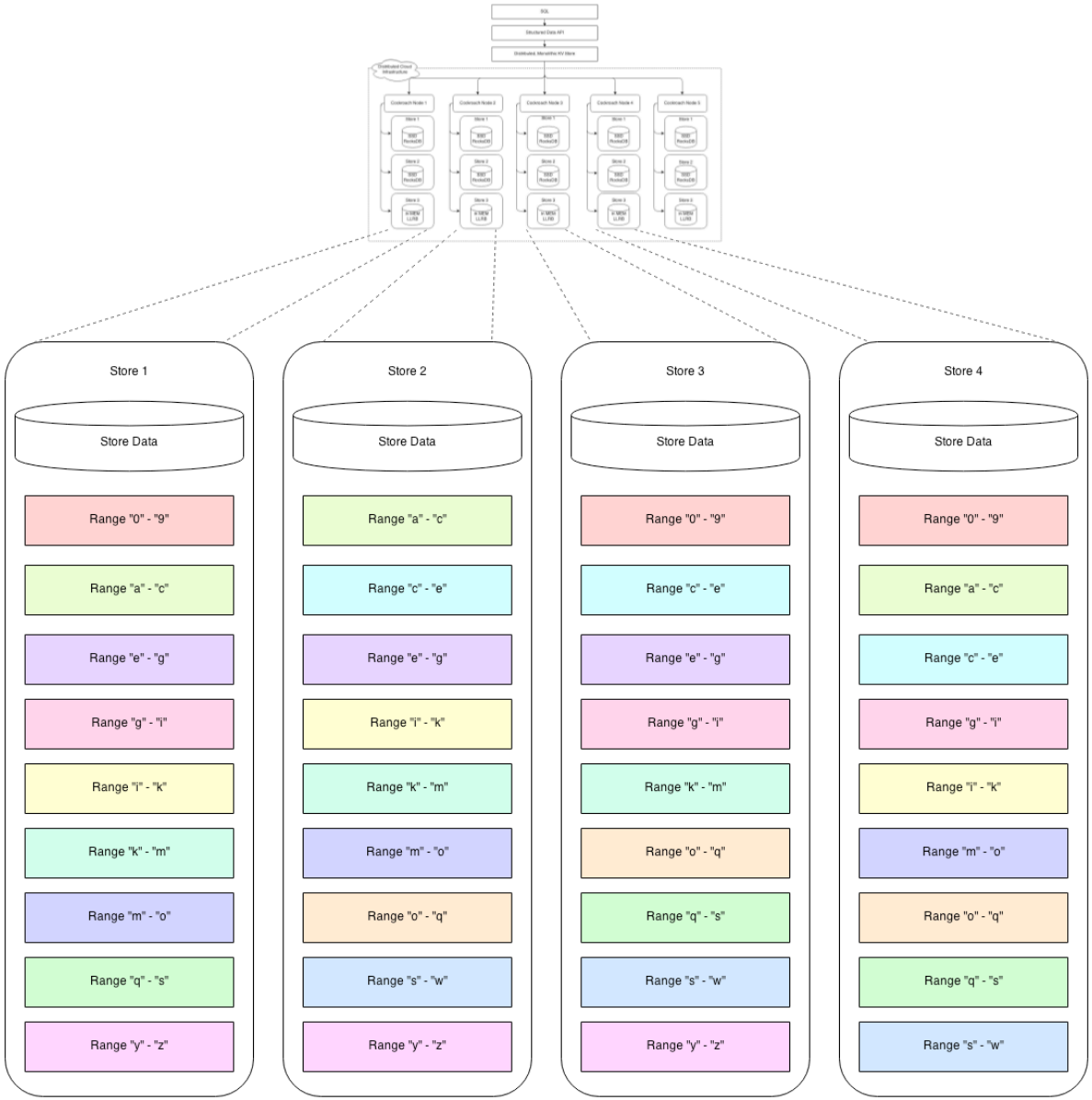
Read more: https://www.cockroachlabs.com/docs/stable/architecture/overview/

Cockroach DB: Architecture



- LLRB: Left-Leaning Red-Black tree, a special kind of self-balancing binary search tree

Cockroach DB: Architecture



Recall:
Raft is a consensus fault-tolerant protocol.

To tolerate F failures,
we need at least N = 2F + 1 nodes

Storage Layer: Versioned Data

- CockroachDB maintains multi-version data
  - Historical versions of values are stored with associated commit timestamps
  - Reads can specify a snapshot time to return data at a specific time;
  - Expiration interval: enables to garbage collect older versions of data;
- CockroachDB relies on multi-version concurrency control (MVCC)
  - To process concurrent requests and guarantee consistency
  - Further details later in this lesson

Storage and Self-repair

- CockroachDB nodes contains one or more stores
  - Each store should be placed on a unique disk;
  - Each store internally contains an instance of RockDB;
  - All stores of the same node share a block cache;
  - The concept of node's zone enable to control the range's replication factor, adding constraints as to where the replicas can be located.
  - Stores gossip their descriptors periodically;
    - If a store appears to be failed, affected replicas will autonomously up-replicate themselves to other available stores to meet the replication factor.

Replication Layer

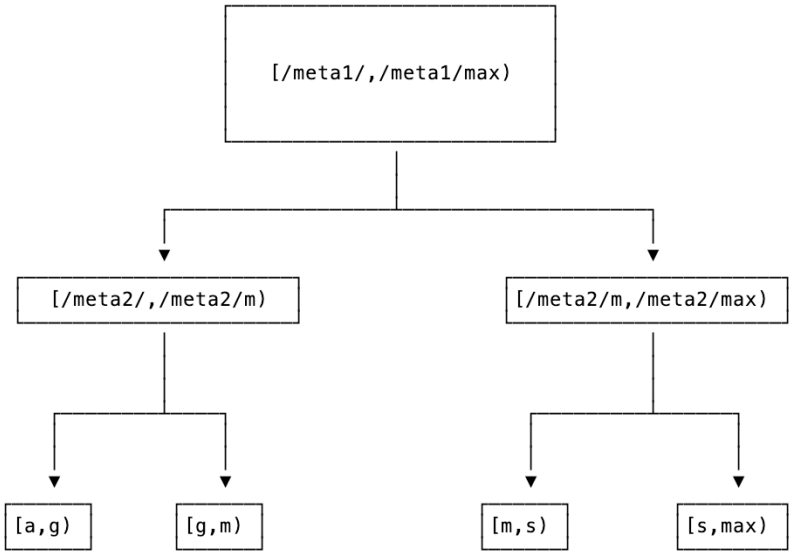
- CockroachDB uses a Raft instance at replica level
  - Recently, to provide better support for multi-region clusters, a new type of replica was introduced: the non-voting replica.
    - Non-voting replicas follow the Raft log (and are thus able to serve follower reads), but do not participate in quorum;
  - However, performing all operations through Raft consensus is an expensive operation;
  - Hence, they also introduced the concept of *Replica Lease;*
  - A single node in the Raft group acts as the *leaseholder*, which is the only node that can serve reads or propose writes to the Raft group leader

Replication Layer

- *Replica Lease*
  - A lease held for a slice of time;
  - A replica establishes itself as owning the lease on a range by committing a special lease acquisition log entry through Raft;
  - The replica becomes the lease holder as soon as it applies the lease acquisition command,
    - This guarantees that the replica has already applied all prior writes and can see them locally.
  - The replica holding the lease may satisfy reads locally (with no Raft consensus);
  - The replica with the lease is in charge of handling Range-specific maintenance (splitting, merging, rebalancing)
  - Raft leadership and the range lease might not be held by the same replica
    - Making the same node both Raft leader and the leaseholder optimizes query performance

Distribution Layer

- CockroachDB stores data in a monolithic sorted map of key-value pairs; this enables:
  - *Simple lookups:* to identify nodes responsible for ranges;
  - *Efficient scans:* leveraging the order of data.



This meta range structure enables addressing up to 4EiB of user data by default

Transaction Layer: Hybrid Logical Clock (HLC)

- Each cockroach node maintains a hybrid logical clock (HLC);
- HLC time uses timestamps which are composed of a physical component and a logical component
  - The logical component is used to distinguish between events with the same physical component
- HLC allows us to track **causality** for related events (similar to vector clocks, but with less overhead)
  - When events are received by a node, it informs the local HLC about the timestamp supplied with the event by the sender;
  - When events are sent a timestamp generated by the local HLC is attached.
- The HLC is updated by every read/write event on the node, and the HLC time >= wall time;
- HLC is used to track versions of values (MVCC) and provide transactional isolation guarantees.

Distributed Transaction

- Physical actors involved in executing a query:
  1. SQL Client sends a query to the CockroachDB cluster;
  2. Load Balancing routes the request to node in the cluster;
  3. Gateway: node that processes the SQL request and responds to the client;
  4. Leaseholder: node responsible for serving reads and coordinating writes of a specific range of keys in your query.
  5. Raft leader: node responsible for maintaining consensus among your CockroachDB replicas.

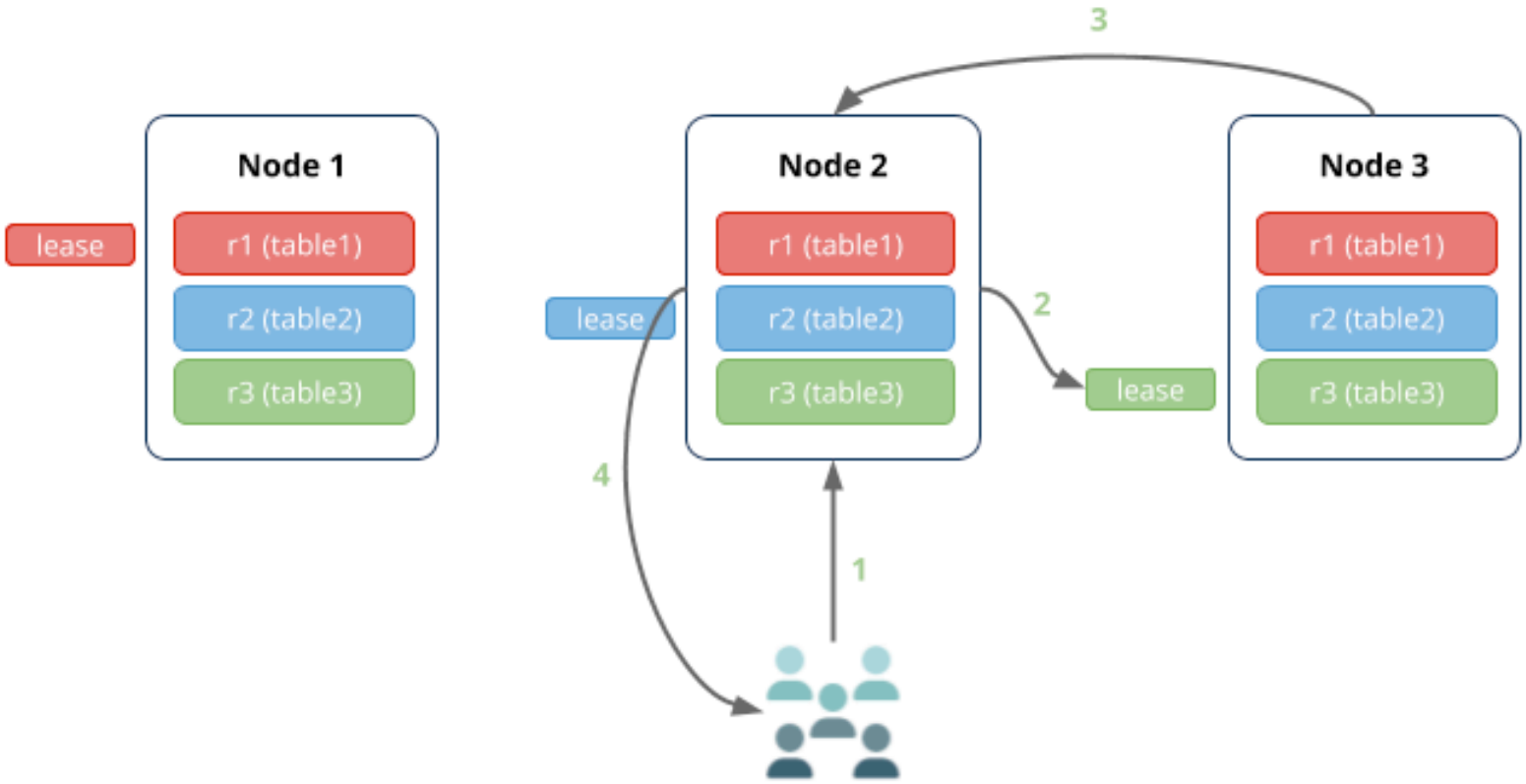
Transaction Layer

- CockroachDB provides:
  - Stale reads: read from local replica (using the clause AS OF SYSTEM TIME)
  - Strongly-consistent reads: go through the leaseholder; see all committed writes; default.

Transaction Layer

- Read scenario

- There are 3 nodes in the cluster.
- There are 3 small tables, each fitting in a single range.
- Ranges are replicated 3 times (the default).
- A query is executed against **node 2** to read from **table 3**

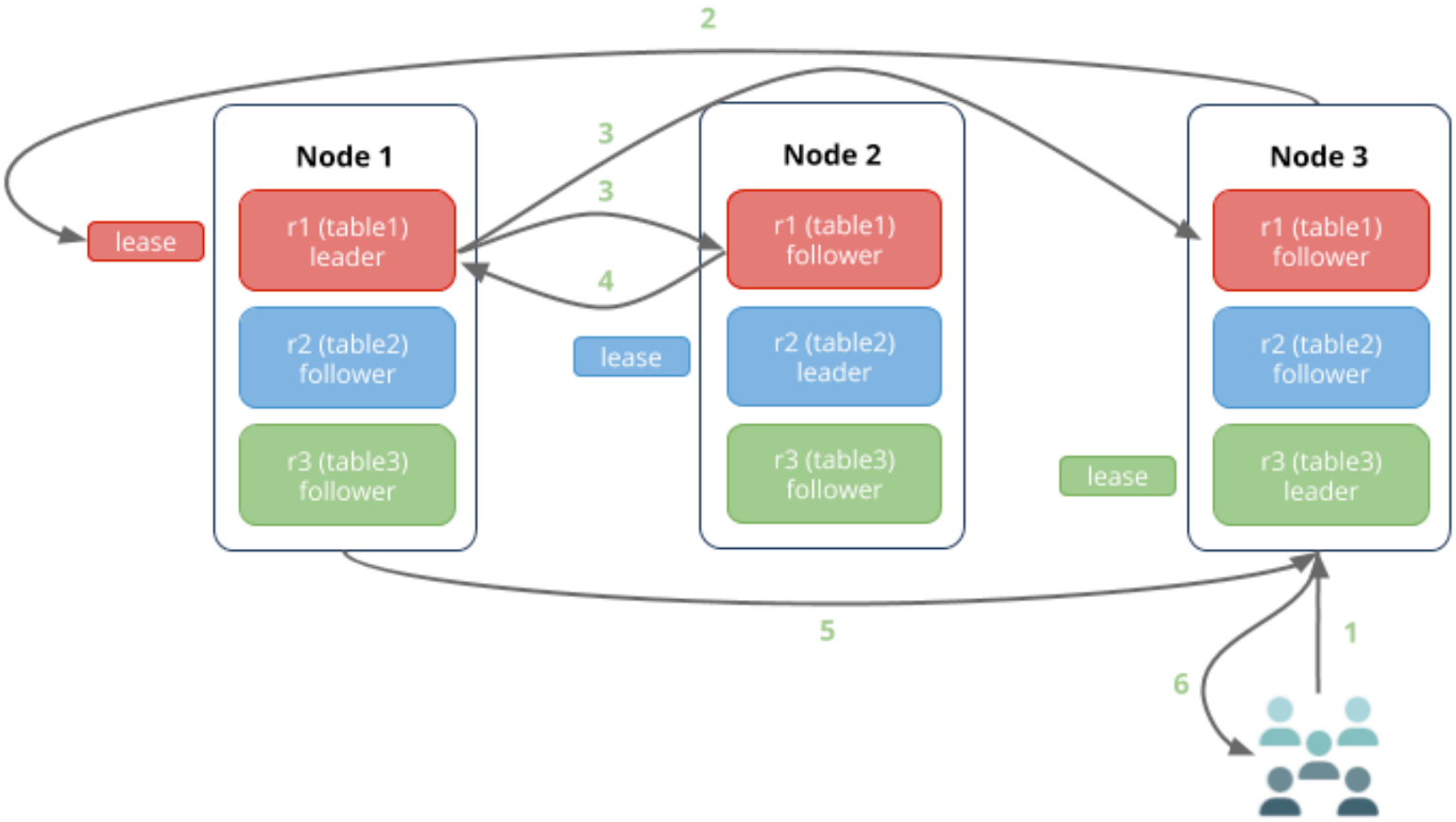


https://www.cockroachlabs.com/docs/v23.2/architecture/reads-and-writes-overview

Transaction Layer

- Write scenario

- There are 3 nodes in the cluster.
- There are 3 small tables, each fitting in a single range.
- Ranges are replicated 3 times (the default).
- A query is executed against **node 3** to write to **table 1**.



https://www.cockroachlabs.com/docs/v23.2/architecture/reads-and-writes-overview

Transaction Layer

- All transactions operate on a **read snapshot**
  - Select a new MVCC timestamp from HLC and capture all writers previously committed

- Isolation level:
  - **Serializable** transactions maintain **per-transaction** read snapshot (default)
  - **Read committed** transactions use **per-statement** read snapshot
    - Writes in concurrent READ COMMITTED transactions *can interleave* without aborting transactions;
    - Each subsequent statement uses a new read snapshot, reads in a READ COMMITTED transaction can return different results.

Read more on isolation levels: https://www.cs.umb.edu/cs734/CritiqueANSI_Iso.pdf

Transaction Layer

- To reduce the occurrence of *concurrency anomalies* in *read committed* isolation, you can use locking reads;

- **Locking reads**:
  - A locking read in a transaction will always have the latest version of a row when the transaction commits;
    - It guarantees that the accessed data will not be changed by intermediate writes.
  - **Exclusive locks** block concurrent writes and locking reads on a row;
    - SELECT … FOR UPDATE
  - **Shared locks** block concurrent writes and **exclusive** locking reads on a row.
    - SELECT … FOR SHARE