# Chapter 2: CockroachDB architecture

This chapter will outline the CockroachDB architecture from APIs and transactional models through to distributed clusters. We’ll review the “familiar” bits: wire protocol, SQL language, transactions, then on to the architecture of a single node server. Finally, we’ll dive deep into the distributed database architecture covering topics such as Ranges, replication strategies, leaseholders, clock synchronization and the role of the Raft protocol.

## Architecture at a glance

Summary of all the sections below, with a master diagram

QUESTION: Are the correct layers:

* Wire protocol
* SQL
* Transactions
* Distribution
* Storage

Transactions are implemented across the SQL, Distribution and Storage layer. Maybe it’s best to push discussion on transactions until after all that has been made clear? I guess this is why you have the “life of a Distributed transaction” in the docs after everything else has been described?

Pyh



## The CockroachDB SQL LAYER (1)

How CDB processes SQL requests and manages data in relational form.

* **Table/KV Mapping overview**
  + We’ll start with an overview of how CockroachDB maps tables to KV storage since it impacts on everything to come.
* **Request Handling**
  + SQL clients: postgres compatible drivers
  + Typically, a load balancer is used to accept requests and route to one of the nodes in the CockroachDB cluster – this then serves as the Gateway for the request.
* **The Query optimizer**
  + How SQL statements are parsed. Process of establishing an execution plan. Cost based optimizer and table statistics.
* **Tabular data**
  + Introduction to table orientation. Primary and foreign keys. Datatypes esp. the non relational types?\]: arrays, json, etc.
* Materialized views
* **Operations**
  + Lookups, scans, joins. Sorts and aggregations.
* **Secondary indexes** 
  + May need an additional discussion in the Storage section I guess
* **Time Travel Queries**

## The CockroachDB SQL LAYER (2)

## The Transactional layer

Transactions at the logical/application level

* **MVCC**

MVCC is implemented by the storage layer but is integral to transaction processing. Basics of MVCC as it applies to SQL processing

* **Isolation levels**

CockroachDB’s serliazable isolation level. How it differs from Snapshot isolation

* **Managing transactions**

Transactional command BEGIN, COMMIT, SAVEPOINT. Note idioms and strategies for coding transactions are deferred until a later chapter.

Intent locks with FOR UPDATE

Setting transaction priority

* **Latches**

Lightwieght mutexes that serialize access to key ranges across transactions <https://www.cockroachlabs.com/docs/v20.2/architecture/transaction-layer#latch-manager>

* **Conflict management**

Transaction retries, manual and automatic. Reference forward to optimistic vs pessimistic locking in Chapter 6

## The distribution layer

How data is distributed across nodes within a cluster.

* **Ranges**
  + Monolithic map
  + Range Splits
* **Leaseholders**
  + Leaseholder rebalancing

## The replication layer

Q: your docs have distribution and replication as separate layers. I’m inclined to combine them – thoughts?

How redundant copies of data are distributed across the cluster

* **Clusters and nodes**
  + How multiple nodes co-operate to form a cluster.
* **Replication**
  + Replication log (RAFT log)
  + Parallel commits
* Hybrid logical clocks
* Gossip
* RAFT
* Distributed Query optimization (DistSQL)

## The storage Engine

Details of the Pebble KV store on each node.

* **The Pebble Key-value store**
* **Table mapping to KV**
* **Log Structured Merge Trees** 
  + Memtables, SSTables, CommitLog, Bloom filters
* **Column Families**
* **Vectorized execution**
* **Secondary indexes**
* **Caching**
* **KV transactions**
* **Hybrid logical clocks**

### Q: Is the only caching KV caching?

# Chapter resources

* Chapter 2: The CockroachDB Architecture
  + Core concepts
    - Cluster, node, range, replica, leaseholder, raft leader, raft log, gateway
      * [Docs]
        + [Architecture overview](https://www.cockroachlabs.com/docs/stable/architecture/overview.html)
        + [Reads and Writes Overview](https://www.cockroachlabs.com/docs/stable/architecture/reads-and-writes-overview.html)
      * [Cockroach U]
        + [The Keyspace, Replicas, and Ranges](https://www.youtube.com/watch?v=LgbrmIjH0cU&list=PL_QaflmEF2e8Prn7r7CIyBKsHPgsgNO_1&index=10)
        + [The Raft Protocol](https://www.youtube.com/watch?v=k5BR9m8o9ec&list=PL_QaflmEF2e8Prn7r7CIyBKsHPgsgNO_1&index=11)
  + Architecture layers
    - SQL, Transaction, Distribution, Replication, Storage
      * [[Docs] Architecture layer docs](https://www.cockroachlabs.com/docs/stable/architecture/overview.html)
      * [SIGMOD] Section 2, System Overview
  + Example: Life of a distributed transaction
    - [[Docs] Life of a distributed transaction](https://www.cockroachlabs.com/docs/v20.2/architecture/life-of-a-distributed-transaction.html)

<https://www.cockroachlabs.com/blog/raft-is-so-fetch/>

<https://www.cockroachlabs.com/blog/pebble-rocksdb-kv-store/>

<https://www.cockroachlabs.com/blog/cost-based-optimizer-20-1/>

<https://www.cockroachlabs.com/blog/nested-transactions-in-cockroachdb-20-1/>

<https://www.cockroachlabs.com/blog/time-travel-queries-select-witty_subtitle-the_future/>

<https://www.cockroachlabs.com/blog/living-without-atomic-clocks/>

<https://www.cockroachlabs.com/blog/joint-consensus-raft/>

<https://www.cockroachlabs.com/blog/parallel-commits/>

<https://www.cockroachlabs.com/blog/how-we-built-a-vectorized-execution-engine/>

<https://www.cockroachlabs.com/blog/what-write-skew-looks-like/>

<https://www.cockroachlabs.com/blog/distributed-sql-key-value-store/>