High-level Architecture

This lesson gives a brief overview of Cassandra's architecture.

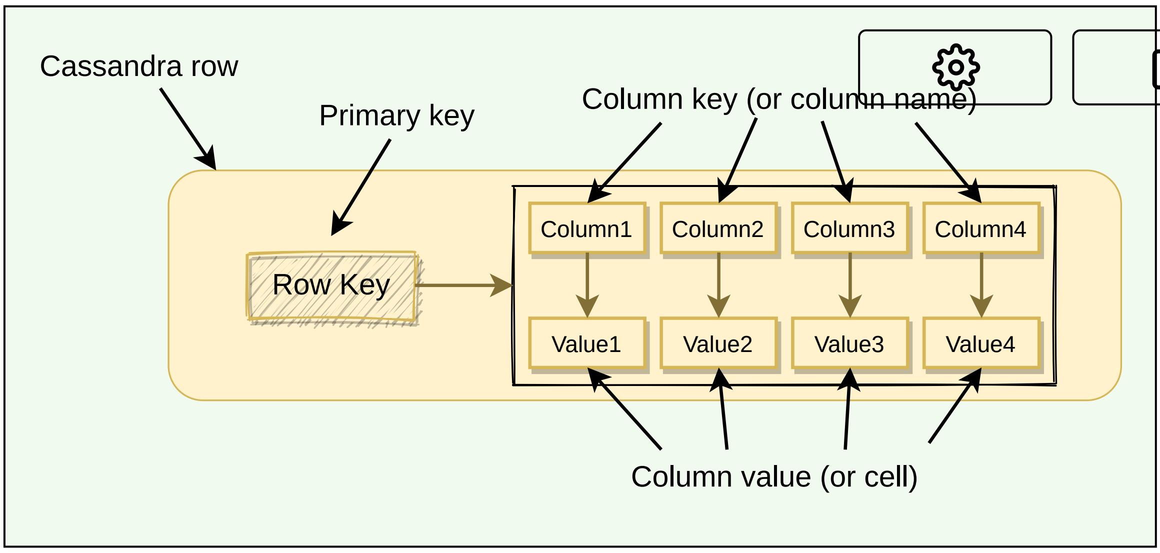
Cassandra common terms#

Before digging deep into Cassandra's architecture, let's first go through some of its common terms:

Column: A column is a key-value pair and is the most basic unit of data structure.

- **Column key:** Uniquely identifies a column in a row.
- **Column value:** Stores one value or a collection of values.

Row: A row is a container for columns referenced by primary key. Cassandra does not store a column that has a null value; this saves a lot of space.



Components of a Cassandra row

Table: A table is a container of rows.

Keyspace: Keyspace is a container for tables that span over one or more Cassandra nodes.

Cluster: Container of Keyspaces is called a cluster.

Node: Node refers to a computer system running an instance of Cassandra. A node can be a physical host, a machine instance in the cloud, or even a Docker container.

NoSQL: Cassandra is a NoSQL database which means we cannot have `joins` between tables, there are no `foreign keys`, and while querying, we cannot add any column in the `where` clause other than the primary key. These constraints should be kept in mind before deciding to use Cassandra.

High-level architecture#

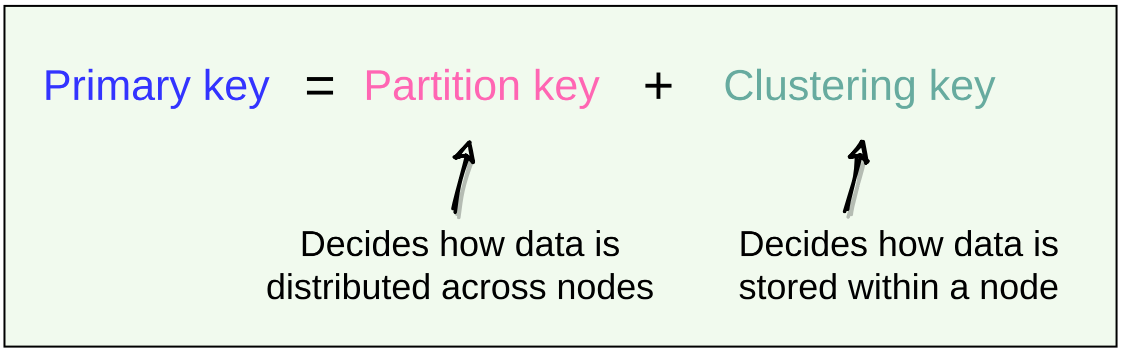
Data partitioning#

Cassandra uses **consistent hashing** for data partitioning. Please take a look at Dynamo's data partitioning (https://www.educative.io/collection/page/5668639101419520/5559029852536832/6501426287607808); all consistent hashing details described in it applies to Cassandra too.

Let's look into mechanisms that Cassandra applies to uniquely identify rows.

Cassandra keys#

The **Primary key** uniquely identifies each row of a table. In Cassandra primary key has two parts:


Primary key = Partition key + Clustering key
Decides how data is distributed across nodes
Decides how data is stored within a node

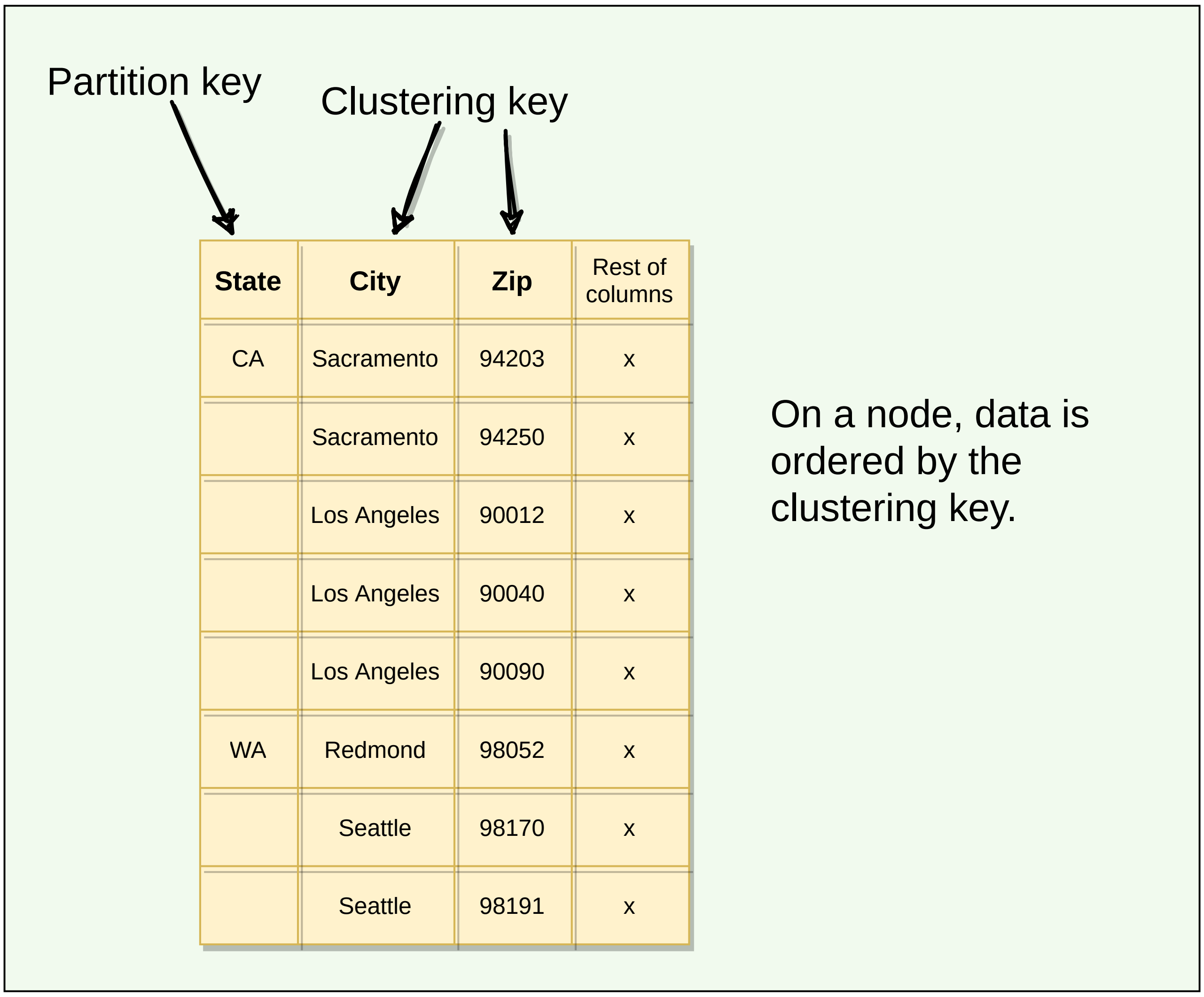
Parts of a primary key

The partition key decides which node stores the data, and the clustering key decides how the data is stored within a node. Let's take the example of a table with PRIMARY KEY (`city_id`, `employee_id`). This primary key has two parts represented by the two columns:

1. `city_id` is the partition key. This means that the data will be partitioned by the `city_id` field, that is, all rows with the same `city_id` will reside on the same node.
2. `employee_id` is the clustering key. This means that within each node, the data is stored in sorted order according to the `employee_id` column.

Clustering keys#

As described above, clustering keys define how the data is stored within a node. We can have multiple clustering keys; all columns listed after the partition key are called clustering columns. Clustering columns specify the order that the data is arranged on a node.

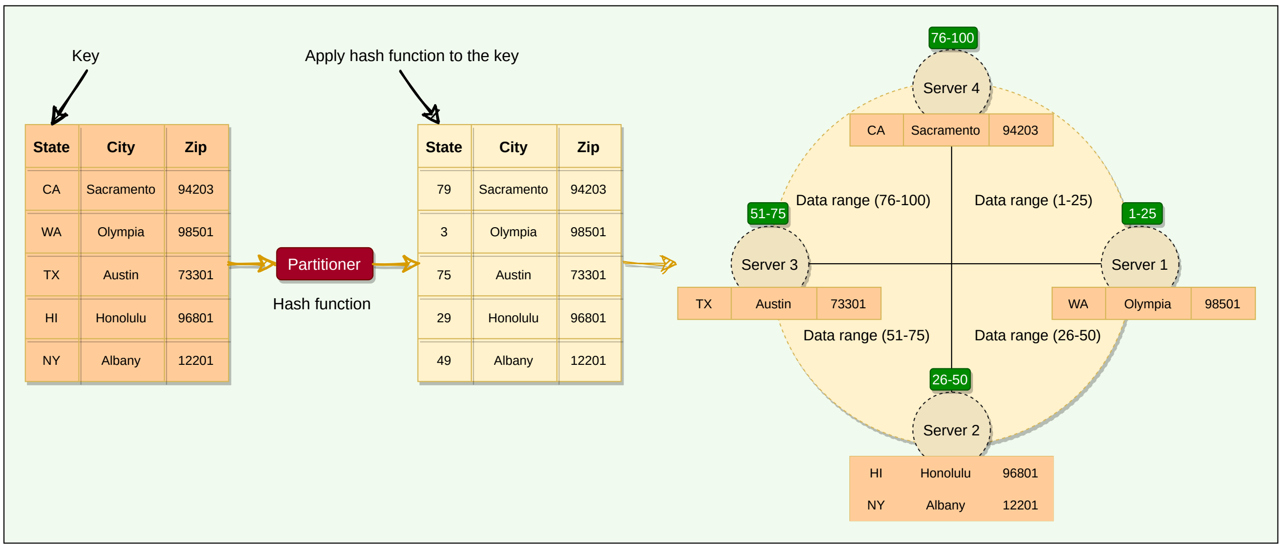


State	City	Zip	Rest of columns
CA	Sacramento	94203	x
	Sacramento	94250	x
	Los Angeles	90012	x
	Los Angeles	90040	x
	Los Angeles	90090	x
WA	Redmond	98052	x
	Seattle	98170	x
	Seattle	98191	x

Clustering key

Partitioner#

Partitioner is the component responsible for determining how data is distributed on the Consistent Hash ring. When Cassandra inserts some data into a cluster, the partitioner performs the first step, which is to apply a hashing algorithm to the partition key. The output of this hashing algorithm determines within which range the data lies and hence, on which node the data will be stored.



State	City	Zip
CA	Sacramento	94203
WA	Olympia	98501
TX	Austin	73301
HI	Honolulu	96801
NY	Albany	12201

State	City	Zip
79	Sacramento	94203
3	Olympia	98501
75	Austin	73301
29	Honolulu	96801
49	Albany	12201

CA	Sacramento	94203

TX	Austin	73301

WA	Olympia	98501

HI	Honolulu	96801
NY	Albany	12201

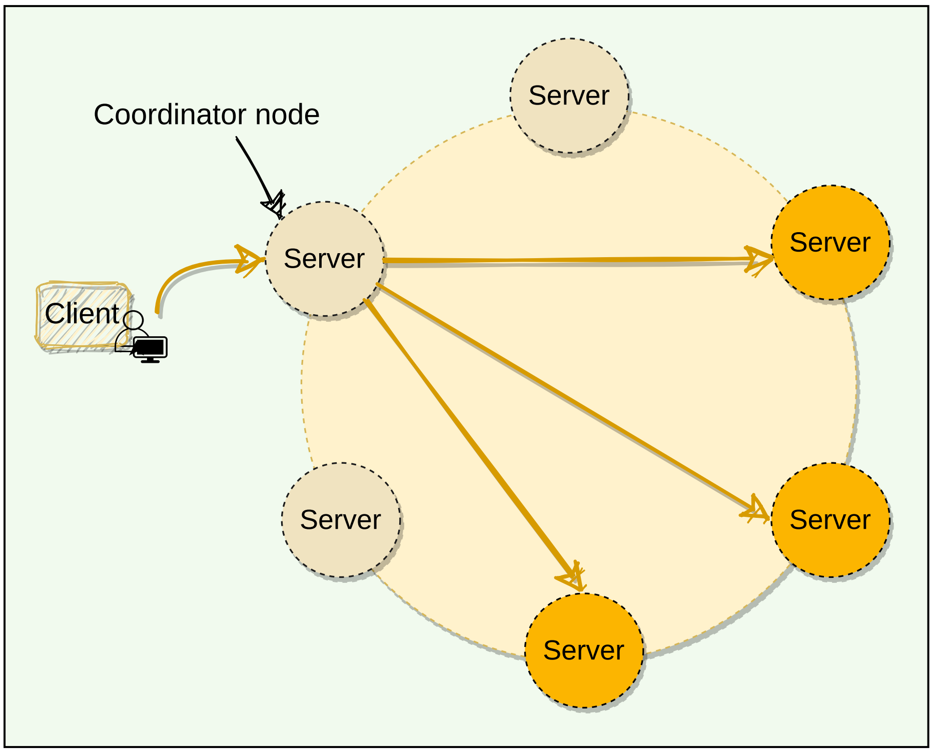
Distributing data on the Consistent Hashing ring

By default, Cassandra uses the Murmur3 hashing function. Murmur3 will always produce the same hash for a given partition key. This means that we can always find the node where a specific row is stored. Cassandra does allow custom hashing functions, however, once a cluster is initialized with a particular partitioner, it cannot be changed later. In Cassandra's default configuration, a token is a 64-bit integer. This gives a possible range for tokens from -2^{63} to $2^{63} - 1$.

All Cassandra nodes learn about the token assignments of other nodes through gossip (discussed later). This means any node can handle a request for any other node's range. The node receiving the request is called the **coordinator**, and any node can act in this role. If a key does not belong to the coordinator's range, it forwards the request to the replicas responsible for that range.

Coordinator node#

A client may connect to any node in the cluster to initiate a read or write query. This node is known as the coordinator node. The coordinator identifies the nodes responsible for the data that is being written or read and forwards the queries to them.



Client connecting to the coordinator node

As of now, we discussed the core concepts of Cassandra. Let's dig deeper into some of its advanced distributed concepts.

← Back
Cassandra: Introduction

Next →
Replication

Mark as Completed

Report an Issue