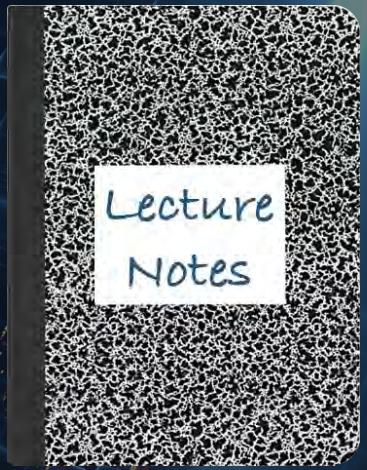
CS 417 – DISTRIBUTED SYSTEMS

Week 9: Distributed Databases

Part 2: Cassandra



Paul Krzyzanowski

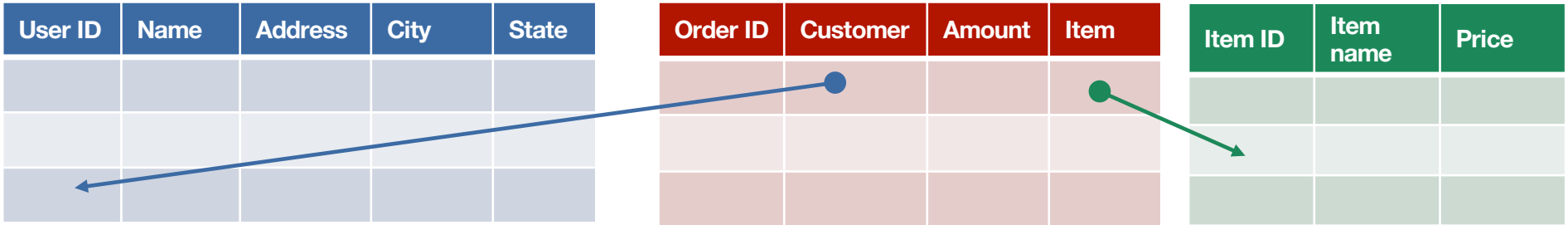
Cassandra

- Distributed, customizable, eventually-consistent NoSQL database
- Originated at Facebook in 2008
  - Currently an Apache open-source project
  - ACID relational database systems were too slow
  - CAP theorem: strong consistency → availability suffers
    - Performance also suffers (e.g., locking, running commit protocols)
  - Go with an eventually consistent model
  - Built as a combination of Amazon Dynamo DHT + Google Bigtable

SQL vs. NoSQL databases

SQL database

- Tables with predefined fields, fields that reference other tables (foreign keys)
- ACID semantics



NoSQL database

- **Wide-column** database: dynamically add columns per row – potentially unlimited amount
- No foreign keys and no integrity checking or locking if a cell contains a key for another table
- Eventually-consistent semantics

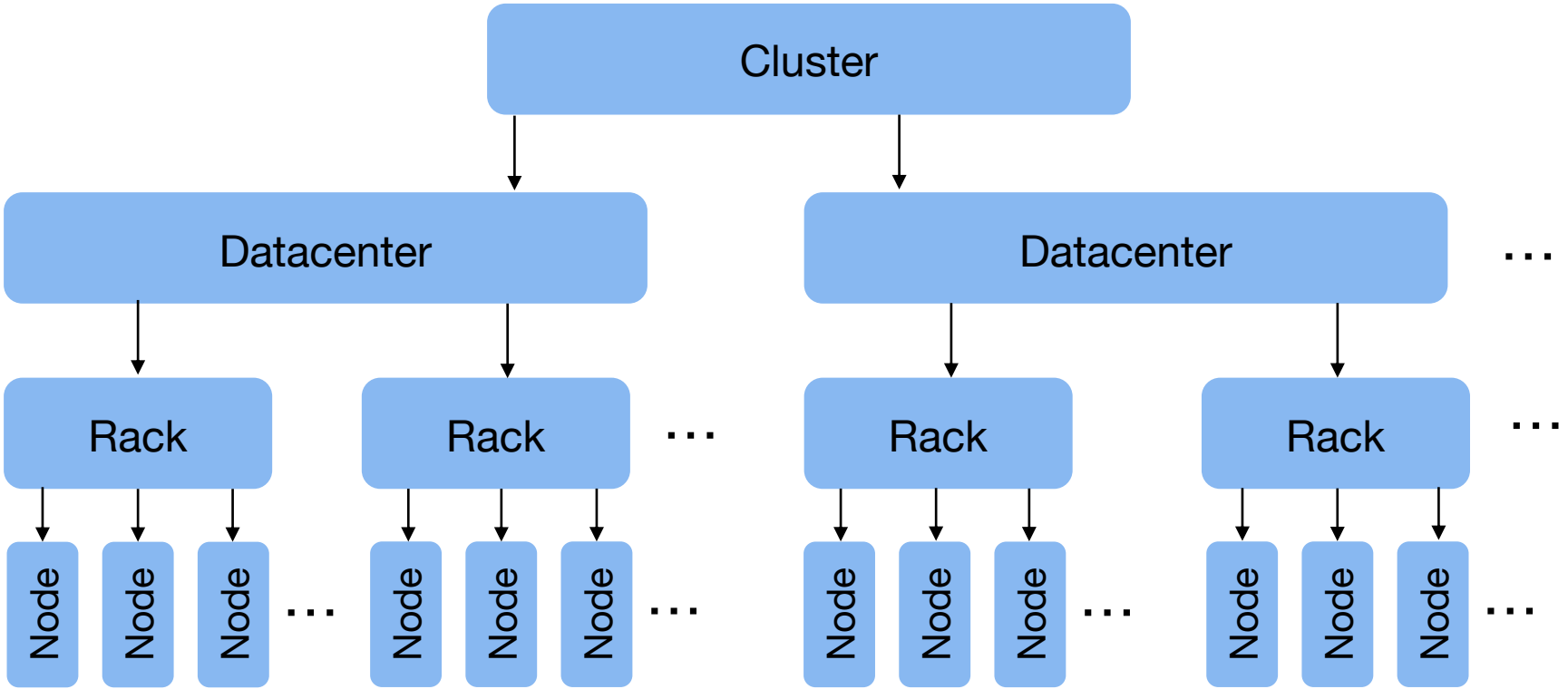
row key = user					
user=user1	key1=value	key2=value	key3=value		
user=user2	key1=value				
user=user3	key1=value	key3=value	key4=value	key5=value	key6=value

Design Goals

- High availability – no single point of failure - no central coordinator
- Low latency
- Run on commodity hardware
- Linear performance increase with additional nodes
- Tunable consistency: Define replication # and policy
- Key-oriented queries
- Flexible data model: row can have different columns with different data types
- SQL-like query language (CQL - Cassandra Query Language)

Server model

Machine hierarchy

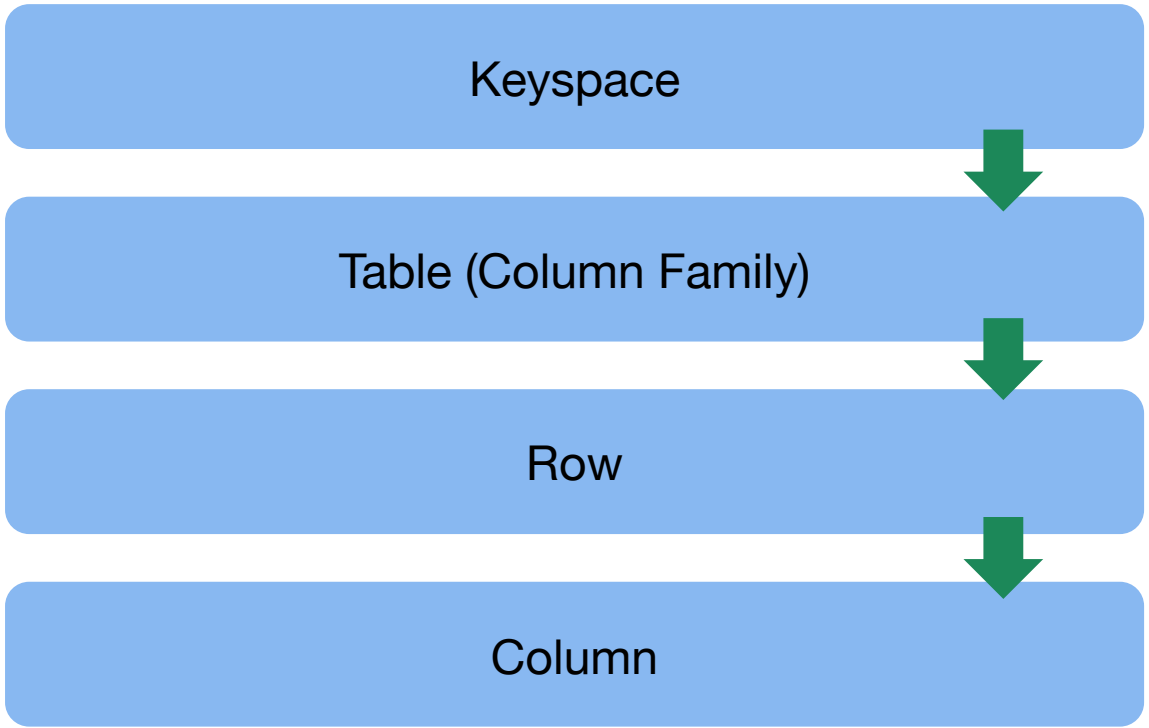


Machine hierarchy

- **Cluster**
  - Collection of machines that run Cassandra
  - Nodes are arranged in a logical ring (like Chord/Dynamo) and data is replicated
- **Datacenter**
  - Machines in one location (data center)
- **Rack**
  - Machines in one rack (low latency – single switch connection)
- **Node**
  - Individual machine

Data Model

Data Hierarchy



Instance of a database

Collection of rows

Set of columns with a primary key for each row

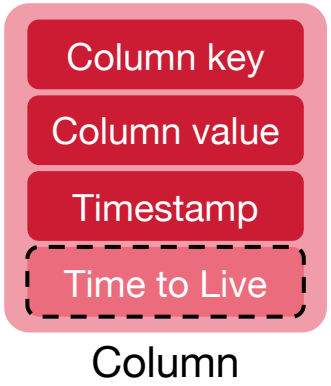
{ key, value, timestamp }

Data hierarchy: **Column**

Column: Fundamental unit of storage

{ name, value, timestamp }

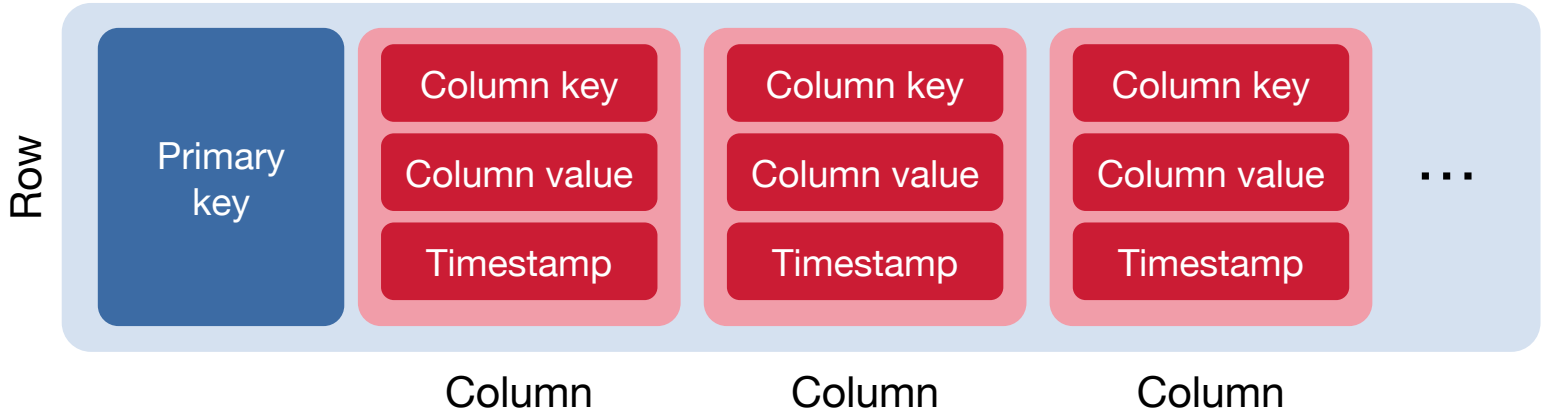
- Each column has a **name** (key) that can be queried for a **value**
  - Data types (>20) include alphabetic, numeric, blob, time, set, map
- **Timestamp** enables conflict resolution among replicas
  - Usually created by the client – synchronized clocks assumed
- Columns can also be given an optional **expiration** timestamp (time to live)



Data hierarchy: Row

Row: Ordered collection of columns identified by a row key

- Each row can have a different – and almost unlimited – number of columns
- All data for a row must fit within a single node
- Columns are often sparse – the names can be used to store values
  - Each row can contain an unlimited number of columns as needed
  - E.g., "*visited_url*" could be a column name; "*time_of_visit*" could be a value

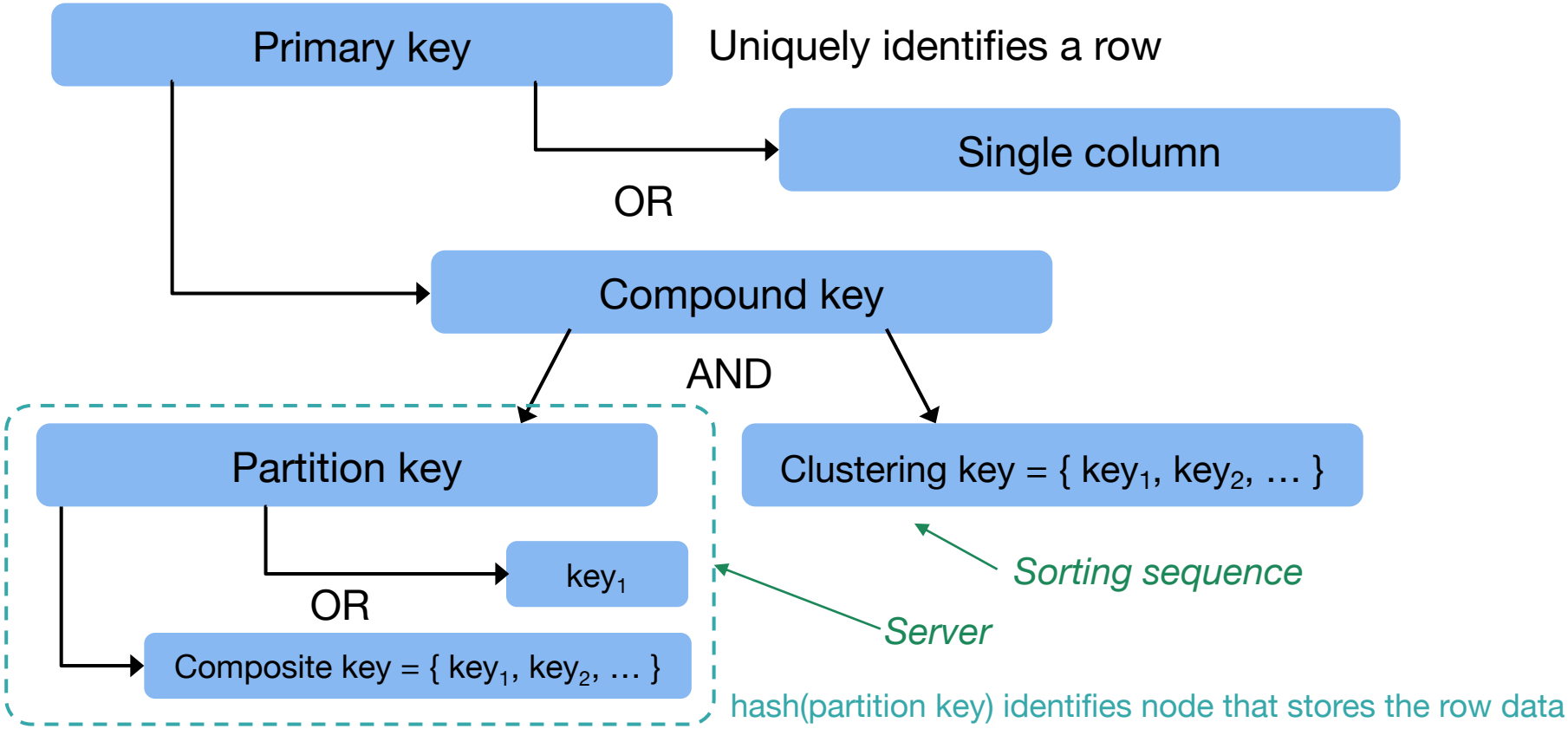


Data hierarchy: Row Key

Keys are columns that are identified as keys when the table is created

- A row is uniquely identified by a **primary key**
  - This is the name of a single column
  - or **compound key** = list of columns = { partition key and clustering key(s) }
- **Partition key**: used to find the node containing the row of data
  - Partition key hashed to determine the machine that stores the data
  - Partition key used to distribute data across machines in the cluster
  - **Composite key**: you can specify multiple column keys to be the partition key: *hash(key_1, key_2, …)*
    - Allows you to control which rows will be stored on the same machine (efficient access) or which get distributed among available node in the DHT (spread the data evenly across the cluster)
- **Clustering key**(s): sorts data within the node
  - Responsible for sorting data within each partition (portion of a table on one node)
  - Rows in each table within a partition are sorted by the clustering (column) keys
  - The sort order is in the sequence that the column keys are listed
  - Picking the right set of keys can enable efficient access to related rows

Row Key

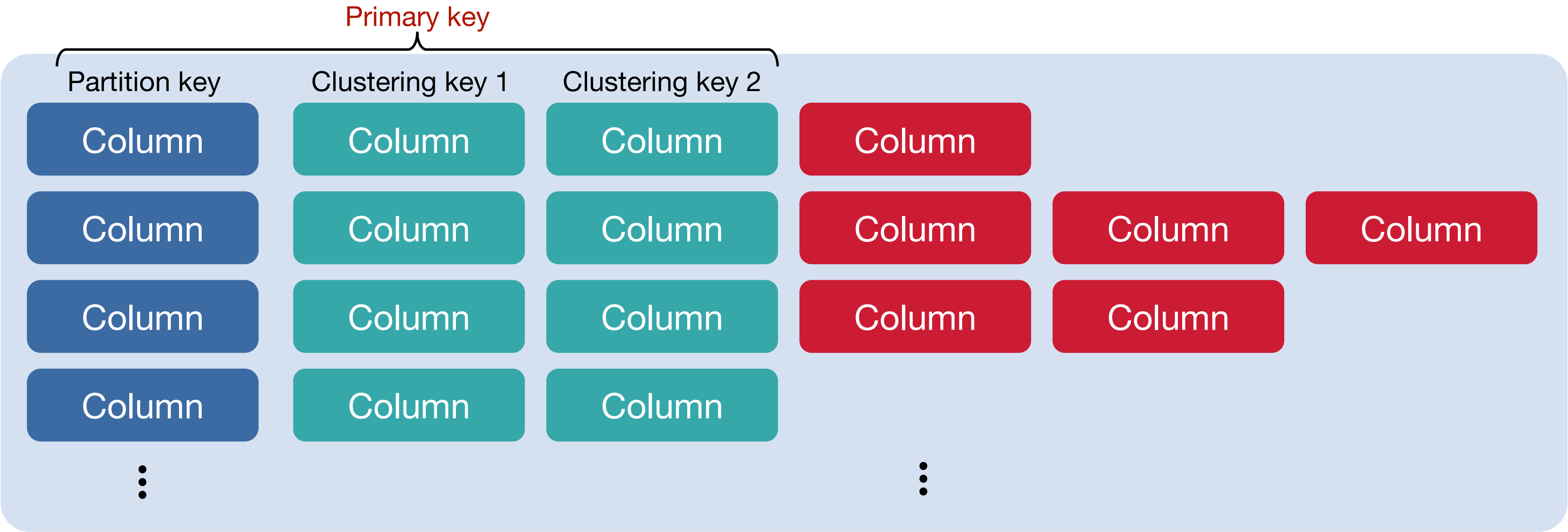

Primary key
Uniquely identifies a row
Single column
OR
Compound key
AND
Partition key
Clustering key = { key1, key2, … }
key1
OR
Composite key = { key1, key2, … }
Sorting sequence
Server
hash(partition key) identifies node that stores the row data

Data hierarchy: Table

Table (**column family**): container for rows of data

- Partition key(s) uniquely identifies a row
- Each row can have different columns – essentially unlimited
  - Can add columns during runtime

Note: Cassandra used to call this a ***column family*** (and you'll see this in many of the papers). They avoided the term *table* to distinguish this from a relational database with its fixed columns. More recently, they stopped using *column family* and started to use *table*.



Data hierarchy: Table (column family)

- Tables are not related
  - Relational databases support foreign keys and *join* operations – not Cassandra
- A table may be distributed among multiple nodes
  - Based on *hash(partition_key)*
  - The set of rows from that table stored on each node defines a ***partition***
- Table partitions are stored in separate files
  - A partition will usually be composed of several files

Data hierarchy: Keyspace

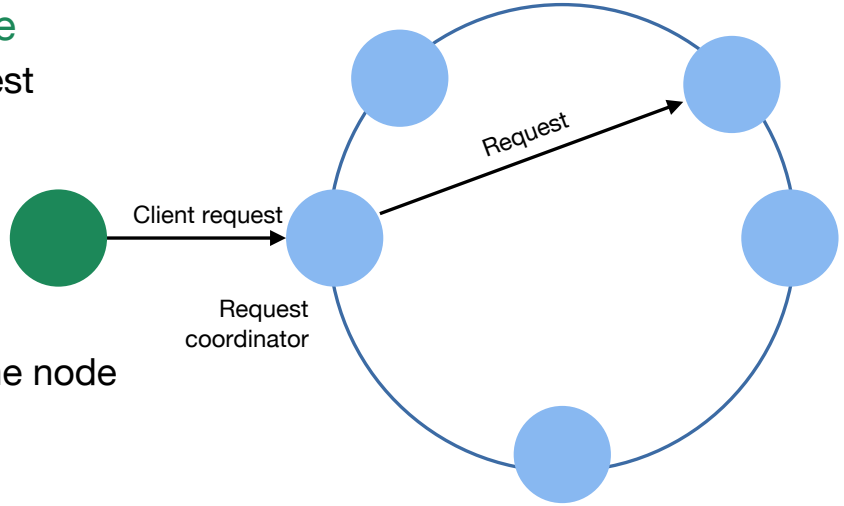
Keyspace: An instance of a Cassandra database

- Container of tables
- Defines
  - Replication factor: # copies
  - Replica placement strategy: defines where replicas go
  - Set of tables

Data Storage

Storage/query model

- A client request can go to any Cassandra node
  - That node will act as a ***coordinator*** for that request
  - Communication uses Apache Thrift RPC
- Dynamo/Chord style distributed hash table
  - Systems arranged in a logical ring – each node responsible for a hash range
  - Coordinator hashes the partition key to identify the node
  - **Virtual nodes**, *vnodes* (like Dynamo)
    - Each node can own a many hash ranges
    - Makes it easy to alleviate hotspots and give more vnodes to more powerful nodes
  - Replicas are found by following the ring clockwise



Write operations

A write can take place at any node

- Data first written to a commit log – then the *write* may be acknowledged
  - This is a crash recovery mechanism
- Data added to memtable at the node (memory-based table)
  - Portion of the table that's stored in memory
  - Periodically, or if the *memtable* becomes full, the *memtable* is written to an **SSTable** (sorted string table, similar to that used in Bigtable) disk structure
  - Table partition = set of *SSTables* for that table on the node
  - If there are multiple identical rows in the *memtable*, only the most recent is written
  - The *commit log* is purged once the data is written

Read operations

Need to combine results from *memtable* and possibly multiple *SSTables*

- Check Bloom Filter for each SSTable
  - Efficient hash-based structure that tells you if data ***might*** be in the table or is ***definitely not*** in the table
- If the data might be in an SSTable
  - Check the key cache for an index entry to find & read the block of data
  - Search partition summary to find the approximate location of the data
    - Do a sequential read to find the data

Replication

Data can be replicated onto multiple nodes

- User defines amount of replication
  - Replication factor, ***N***: # copies
  - Placement strategy: defines where replicas go (rack-aware, datacenter-aware)
- **Tunable consistency**
  - User defines # of responses before write is acknowledged
    - ***W*** = number of nodes to block for writes
    - ***R*** = number of nodes to block for reads
  - Options:
    no response from servers, one success, quorum (majority), or all responses

Replication – Conflicts & dead nodes

- **Replica reconciliation** – what happens when there are conflicts?
  - Not like Dynamo's vector clocks
  - Last write wins model where every mutation is timestamped (including deletes) and then the latest version of data is the "winning" value
- **Dead nodes**
  - Coordinator implements ***Hinted Handoff***
  - Stores a "hint" about dead replicas
    - Location of replica
    - Version information
    - Data being written
  - After a node discovers that a node for which it has hints recovered, it sends the data to that node

Where is Cassandra used?

Tons of places (the list may change!) – almost 7,000 companies

- Apple: over 160,000 Cassandra nodes across 1,000+ clusters – 100 petabytes
- Huawei – over 30,000 instances across 300+ clusters
- Netflix – over 10,000 instances across 100+ clusters – 6 petabytes of data
  - back-end DB for streaming – over 1 trillion requests per day
- Discord for storing messages – over 120M/day
  - All chat history stored forever (but Discord migrated from Cassandra to ScyllaDB in 2023)
- Uber for storing data for live model predictions
  - Each car & user sends location data every 30 seconds
  - Over a million writes per second
  - Over 20 clusters & 300 machines (6 years ago – more now)
- Also
  - Adobe, American Express, AT&T, eBay, HBO, Home Depot, Facebook, PayPal, Verizon, Target, Visa, Salesforce, Palantir, Spotify, Weather Channel, *and many more*

Cassandra downsides

- Not a relational database: no ACID consistency, no locking, no join operations
  - Designed to make it efficient to query data from a single partition & single node – not to gather and merge data from the cluster
- The ordering of keys is defined when the table is created
- Data for a single partition (e.g., partitioned table) must fit within one node
- Timestamps may mess up consistency & create ambiguity
  - Concurrent modifications to a single table may be treated as a tie if they have the same timestamp

The End

References

- Apache Project, Cassandra Documentation.
- Google Cloud, Cloud Bigtable for Cassandra Users: good architectural comparison between Bigtable & Cassandra
- eBay, Cassandra Data Modeling Best Practices, Part 1.
- Jeff Carpenter, Eben Hewitt, Cassandra: The Definitive Guide, 2nd Edition, Chapter 4. The Cassandra Query Language, O'Reilly
- TutorialsPoint, Cassandra Data Model.
- Arin Sarkissian, WTF is a SuperColumn?, PhatDuckk – Digg engineering
- Cassandra Explained with concepts of Distributed Hash Tables and Consistent Hashing
- DataStax, About Hinted Handoff Writes, DataStax Documentation
- Jonathan Ellis, Facebook's Cassandra paper, annotated and compared to Apache Cassandra 2.0
- Brief summaries (student reports)
  - Maitrey J. Soparia, Apache Cassandra (Distributed Hash Table).
  - Vaibhav Shankar, Cassandra DHT-based storage system. (short but not totally accurate)
- About partition keys
  - Piyush Rana, Cassandra Data Modeling: Primary, Clustering, Partition, and Compound Keys, Dzone, October 19, 2016
  - Baeldung, Cassandra Partition Key, Composite Key, and Clustering Key, Oct 9, 2021.
  - Christopher Sherman, Designing a Cassandra Data Model, April 26, 2017