Distributed Databases

Lukasz Antoniak

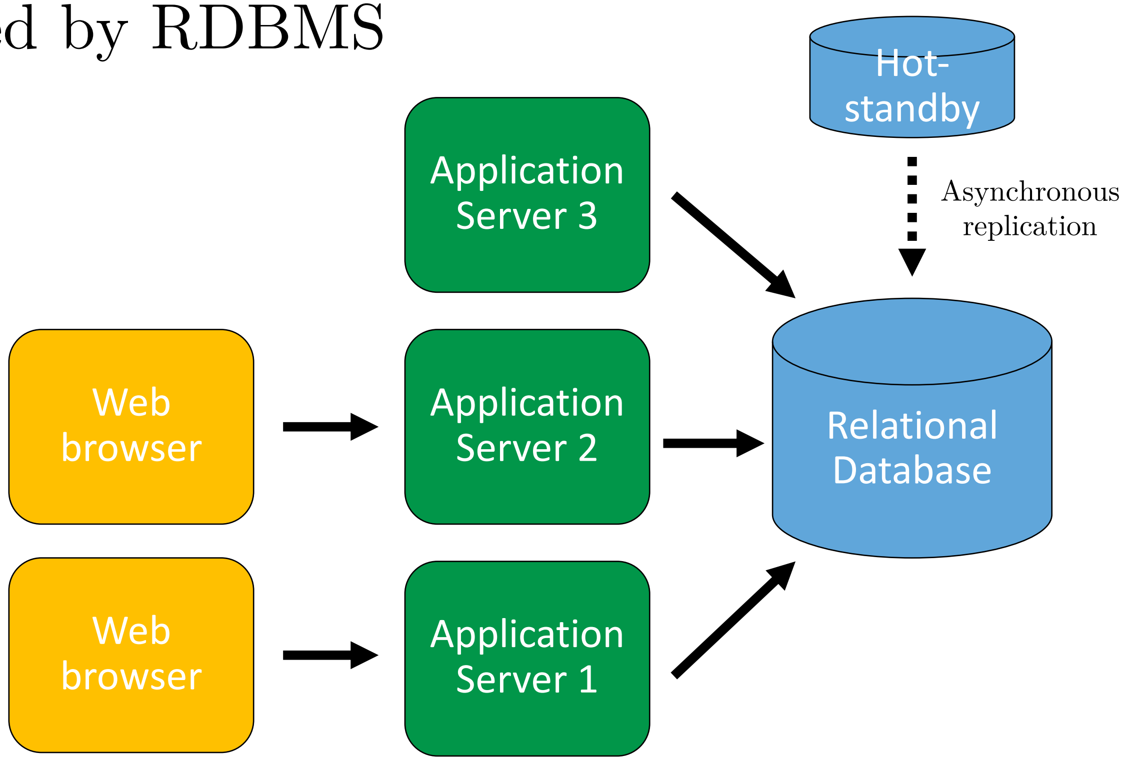
26.01.2022

Agenda

- Three-tier architecture
- Traditional RDBMS systems vs. NoSQL
  - Relations and strong ACID guarantees
  - Scalability and fault-tolerance of distributed databases
- Data sharding and consistent hashing
- CAP theorem
- Short introduction to Apache Cassandra

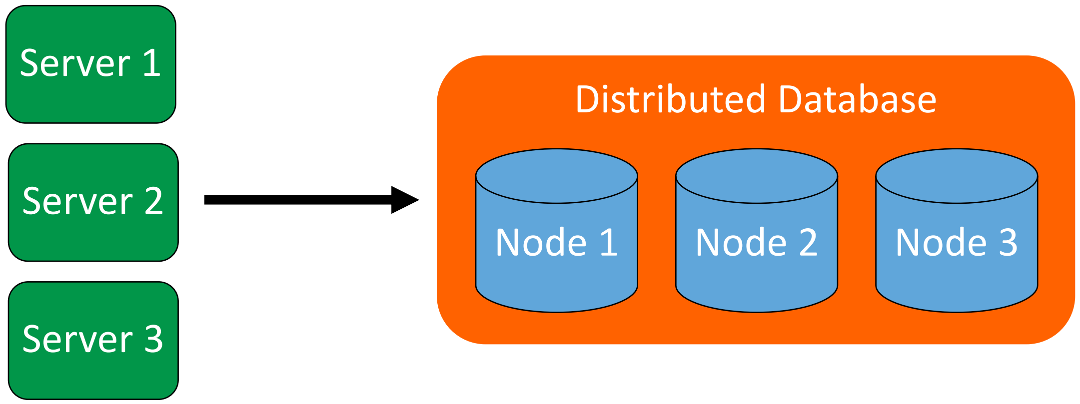
A.D. 2010

- Three-tier architecture
- Wide adoption of relational databases (e.g., Oracle, MySQL, PostgreSQL)
- Strong guarantees and flexibility offered by RDBMS
  - ACID transactions
  - Flexible SQL queries
- Provides only vertical scalability



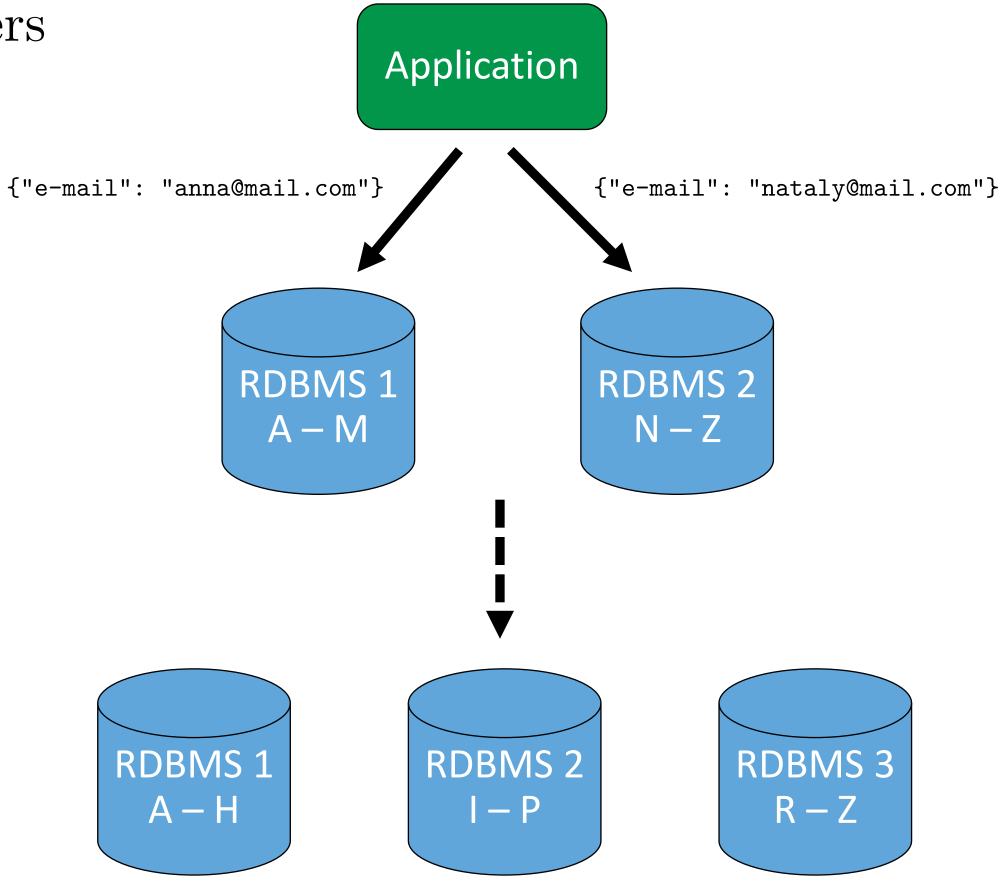
A.D. 2015

- Micro-services architecture, moving away from monolith applications
- Growing adoption of distributed NoSQL databases
  - Unsupported or limited transaction capabilities
  - Difficulties with ad-hock queries
- Horizontal scalability of persistent layer
- ACID properties guaranteed by application, not database



Data Sharding and Consistent Hashing

- Sharding the data by key
  - Divide complete data set across multiple servers
  - Each server contain the same schema, but manages only subset of data
  - Also referred as horizontal partitioning



- Challenges of manual sharding
  - Uneven data distribution
  - Lack of automatic balancing

Data Sharding and Consistent Hashing

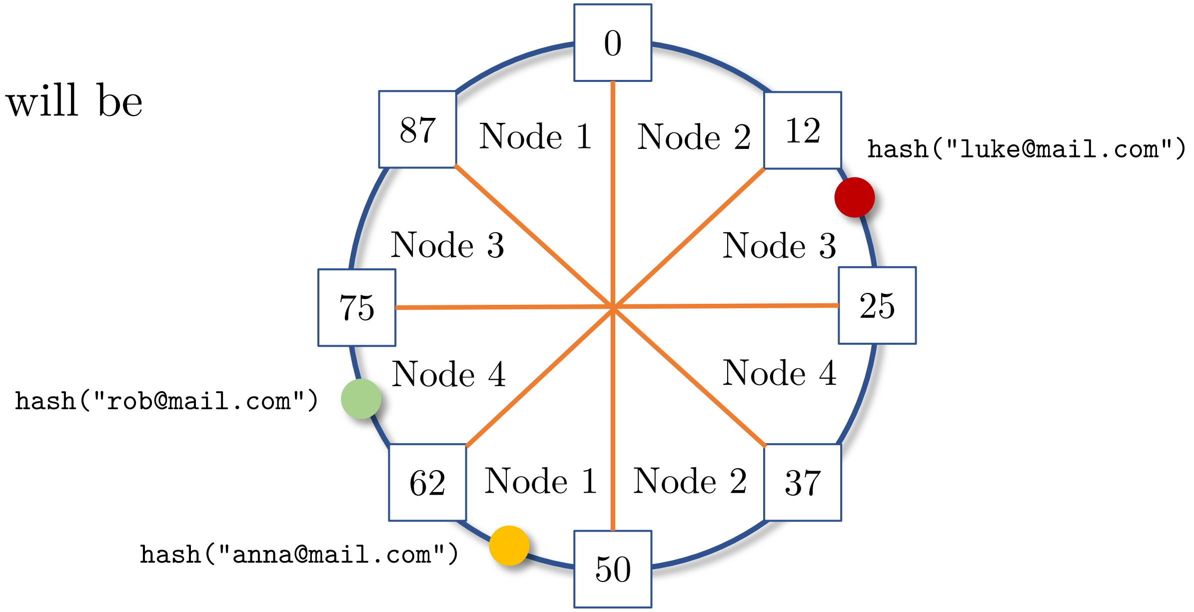
- Leverage mathematical hash function to provide even data distribution

$$
\begin{aligned}
serverID &= abs(\, Murmur3(\,\text{"}luke@mail.com\text{"}\,)\ \%\ serverCount\,) \\
&= abs(\,-309270184\ \%\ 3\,) = 1
\end{aligned}
$$

- Consistent hashing
  - Solves the problem of rehashing complete data set when adding or removing servers
    - Requires $\frac{K}{\max(N1,N2)}$ keys to rehash ("k" – total number of keys; "N1", "N2" – total number of nodes before and after cluster resize)
  - Provides a distribution schema that does not depend directly on number of servers

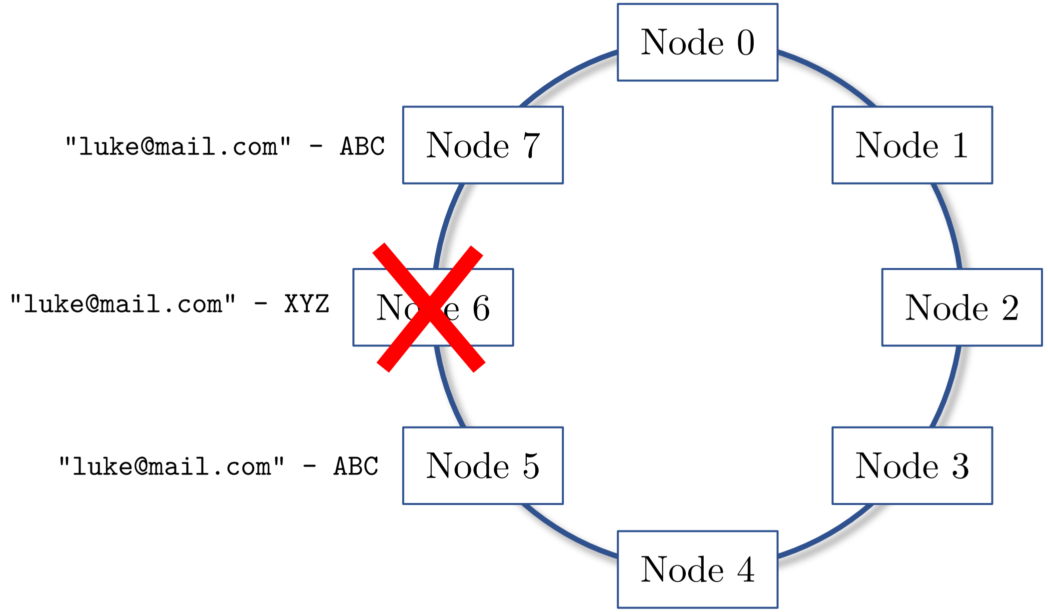
- Consistent hashing
  - Output range of the hash function forms a hash ring
  - Serves are assigned (multiple) tokens within output range of hash function
  - Procedure to assign data to a node:
    - Apply hash function to the key
    - Walk ring clockwise and place the data on first node with token larger than hash value
    - Information about e-mail “luke@mail.com” will be stored on fourth node

Hash function returns values between 0 and 99



Data Replication

- Highly available database cannot maintain just one copy of data
- Issues with data consistency
- Data immediately consistent
  - Read always sees most recent write
- Data eventually consistent
  - Read will see most recent write at some point in future (best effort)



Definition

Any **shared-data** distributed system, can fulfil **at most two** of the following properties:

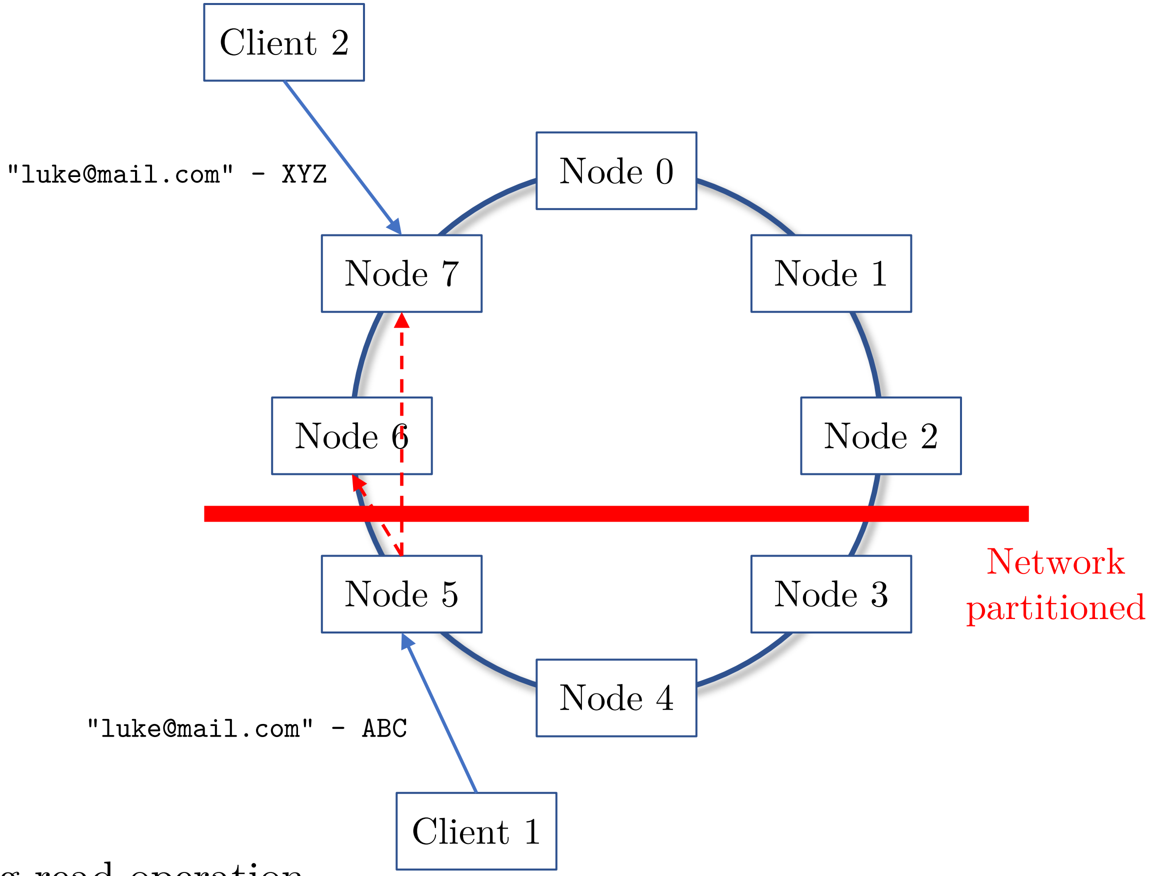
- Consistency – every read operation receives result of the most recent write
- Availability – every read operation receives a non-error response, but returned data may be stale
- Partition Tolerance – system continues to operate without errors despite network disruption between nodes (e.g. "split-brain")

Simplified:

In presence of network partitioning, system's designer must choose between data consistency and availability of his solution.

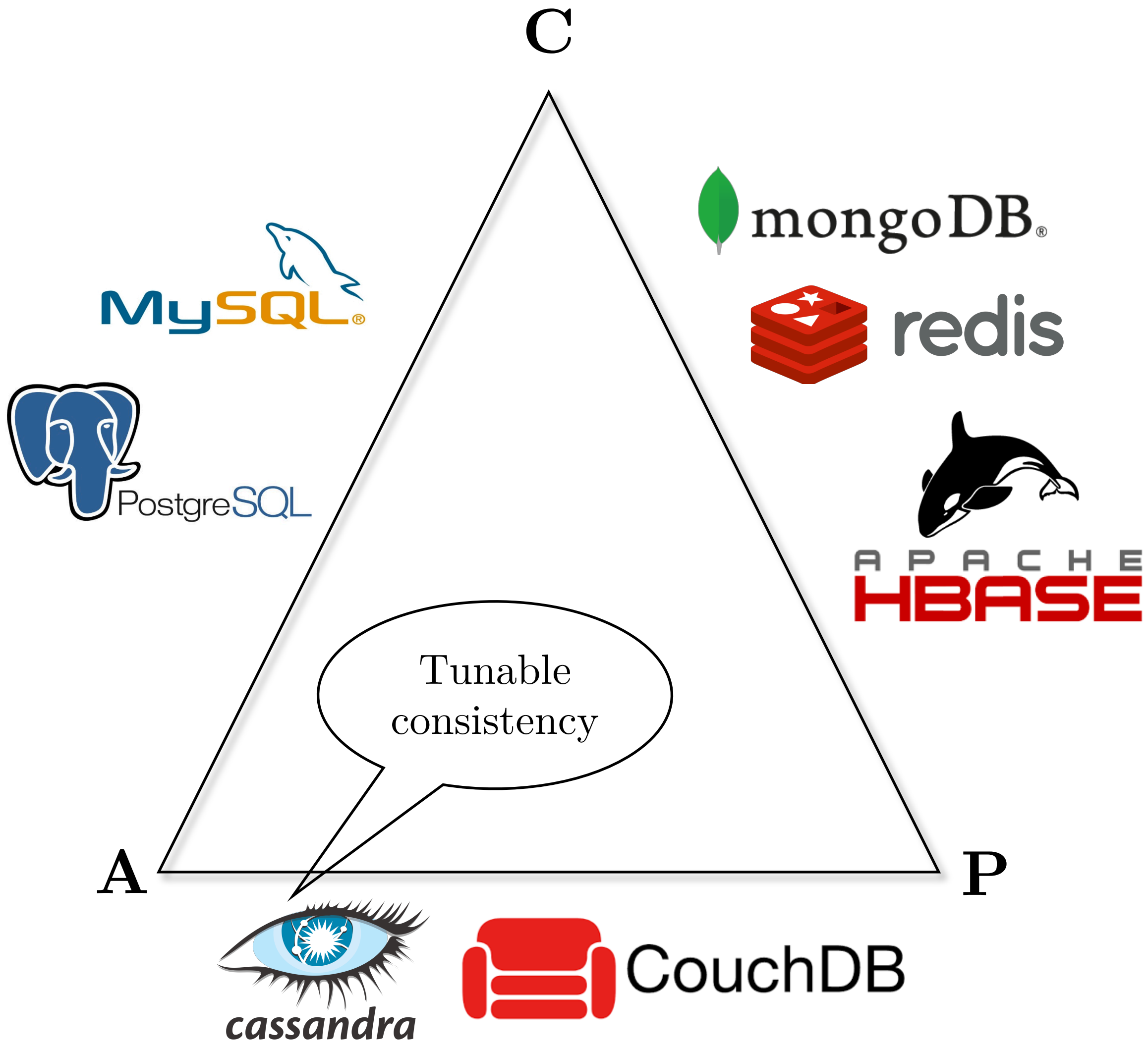
CAP Theorem

- Scenario:
  - Client tries to update record
  - Node cannot replicate updated data to other peers to guarantee consistency
- Available options:
  - Favor consistency – do not support write request and return error to the client
  - Favor availability – accept write request and handle data inconsistency later
    - Read-repair – discover and fix inconsistency during read operation
    - Hinted Handoff – when destination node is down, we store a "hint" of updated value on alive server. When unavailable node comes up again, hints are pushed to him making data consistent.
    - Different ways to resolve conflicts: require application logic, vector clocks and Last-Writer-Wins algorithms



CAP Theorem

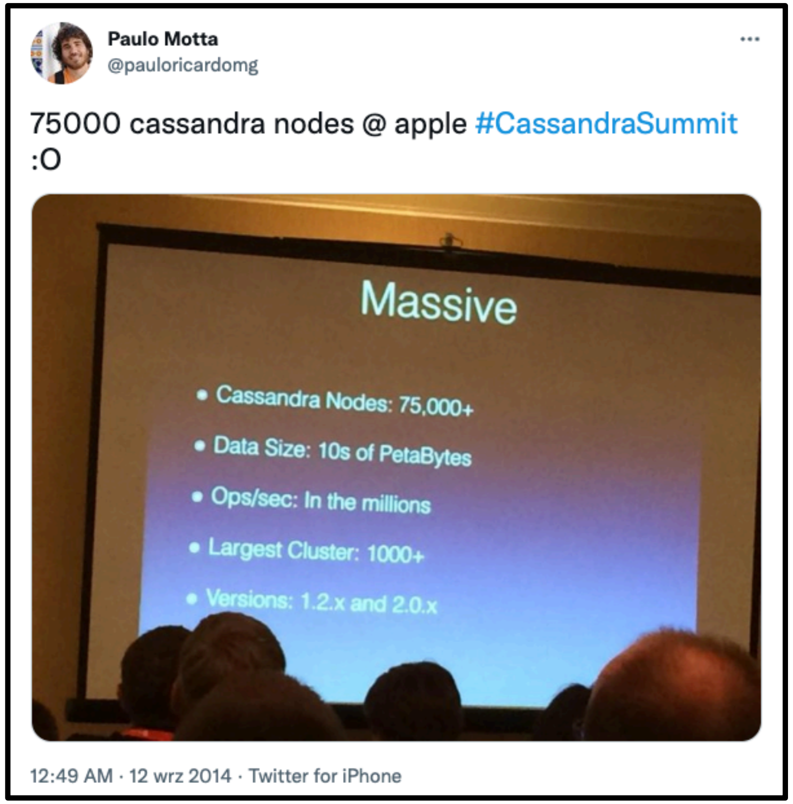
- CA systems are consistent and available, but do not support network partitioning
- AP systems always support read and write operations, but stored data may not be consistent at all time
- CP systems are always consistent, but not available during network partitioning
- Over-simplified database classification, correct only when using their default configuration settings
- Distributed systems battle-testing blog: https://jepsen.io/analyses



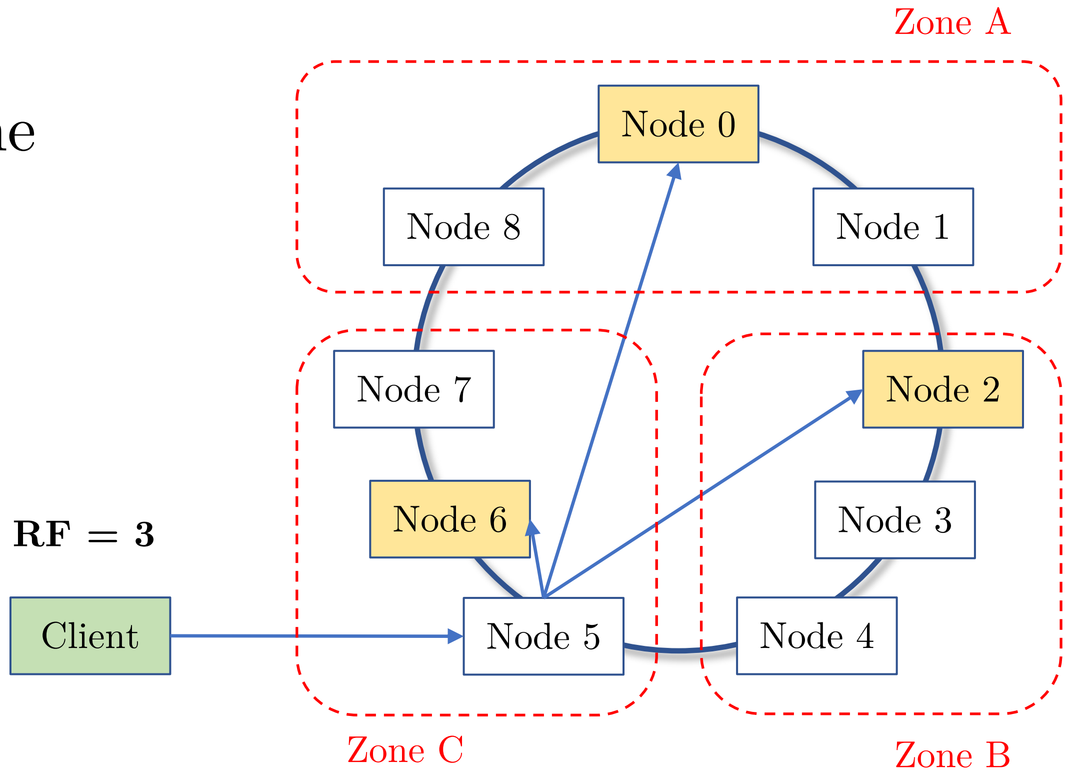
Introduction to Apache Cassandra

- Based on Amazon Dynamo paper
- Horizontally scalable and fault-tolerant NoSQL database
  - Scales linearly with addition of new nodes due to master-less architecture
  - Deployment can span multiple data centers
- Rich, flexible and SQL-like query language CQL
- CAP Available and Partition tolerant, with tunable Consistency

```
CREATE TABLE urls (
   id TEXT,
   url TEXT,
   PRIMARY KEY (id)
);
```



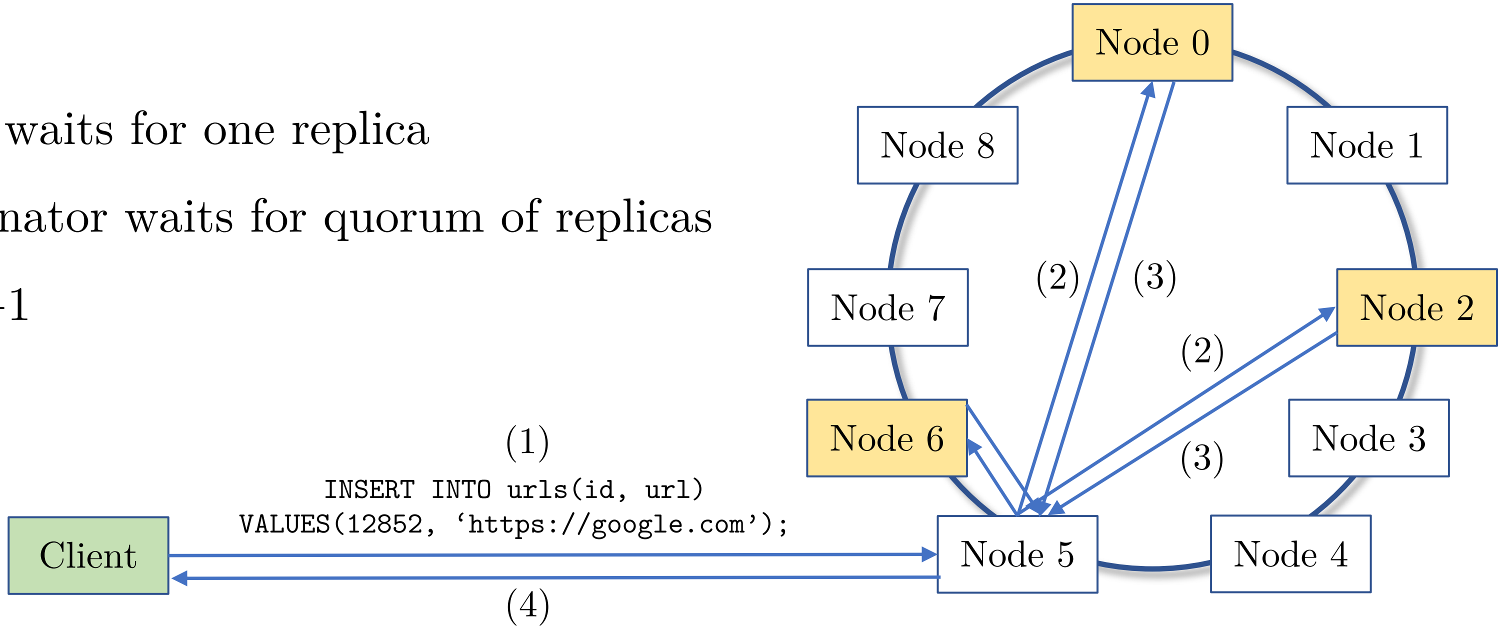
- Cassandra uses data sharding and consistent hashing algorithms
- Data is replicated multiple times according to configured replication factor
  - C* nodes run in different availability zones, making system tolerant to hardware failures (RACK-awareness)
- All nodes in C* cluster perform the same function (master-less architecture)
  - Store data and service client requests
  - A client can read or write to any node, which becomes a coordinator of given request



Cassandra Query Execution

- Client sends CQL query to any node, which becomes the coordinator of given request (1)
- Coordinator immediately tries to replicate data to relevant peers (2)
- Coordinator waits for replica acknowledgements according to specified consistency level (3)
  - CL = ONE – coordinator waits for one replica
  - CL = QUORUM – coordinator waits for quorum of replicas

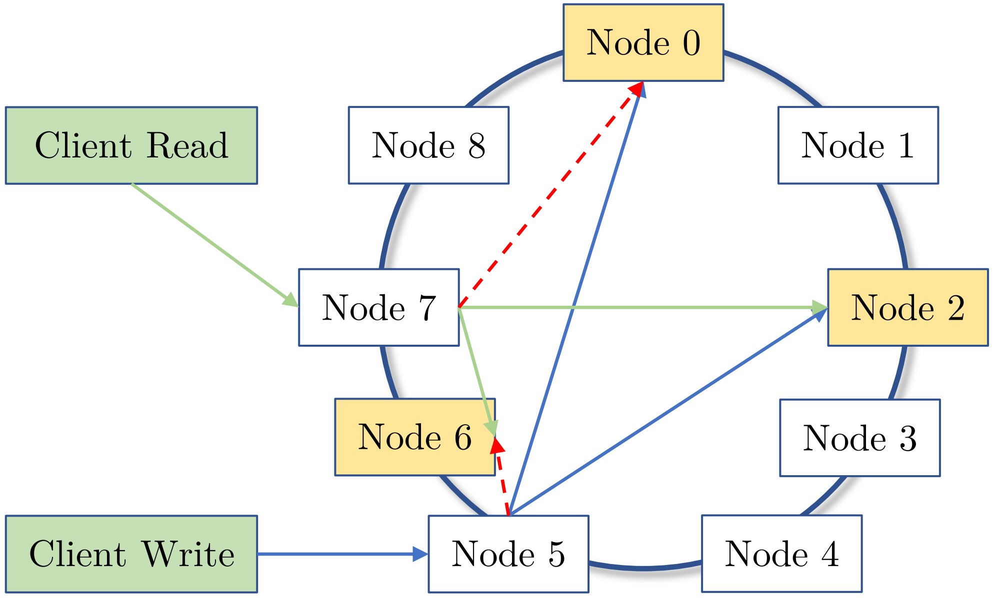
  $quorum(RF) = floor(RF/2) + 1$
  $quorum(3) = 2$
  $quorum(4) = 3$
  $quorum(5) = 3$



- Coordinator sends acknowledgement to the client (4)

Cassandra Consistency

- Data consistency depends on chosen write and read consistency levels
  - Immediate consistency formula: $number\_reads + number\_writes > replication\_factor$
  - Eventually consistent otherwise
- Immediate consistency example
  - Read and write using QUORUM
    - $quorum(3) = 2$ (accepts 1 node failure)
    - $quorum(5) = 3$ (accepts 2 nodes failure)
  - Read using ALL, write using ONE (do not do this)



Cassandra Consistency

- Write operation
  - Write request is always immediately sent to all replicas by coordinator
  - Consistency level specifies the number of replicas that must commit before write operation succeeds
  - Client is notified only after required number of nodes acknowledge to coordinator (consistency level)
  - Some replicas may be not in-sync (even after client application receives acknowledgement)
- Read operation
  - Consistency level defines number of replicas that must respond for read request to succeed
    - Timeout or technical error are not correct responses
  - From received replies, coordinator returns most recent data
  - Inconsistent read responses from replicas may cause read-repair process
    - Anti-pattern of first writing given row and immediately trying to read it

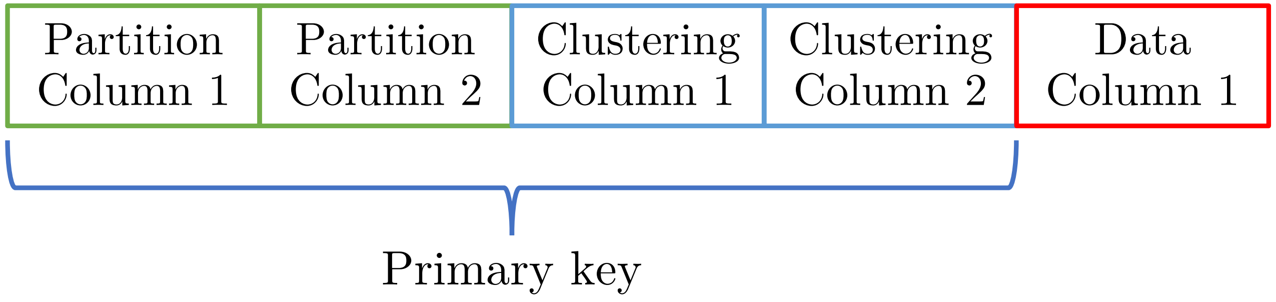
```
PreparedStatement insertUrl = session.prepare("INSERT INTO urls(id, url) VALUES(?, ?)");
BoundStatement statement = insertUrl.bind(id, url).setConsistencyLevel(QUORUM);
session.execute(statement);
// continue processing
```

At this point (if **`execute(…)`** does not throw exception), we are sure that data is replicated to at least quorum of replicas

Cassandra Query Language

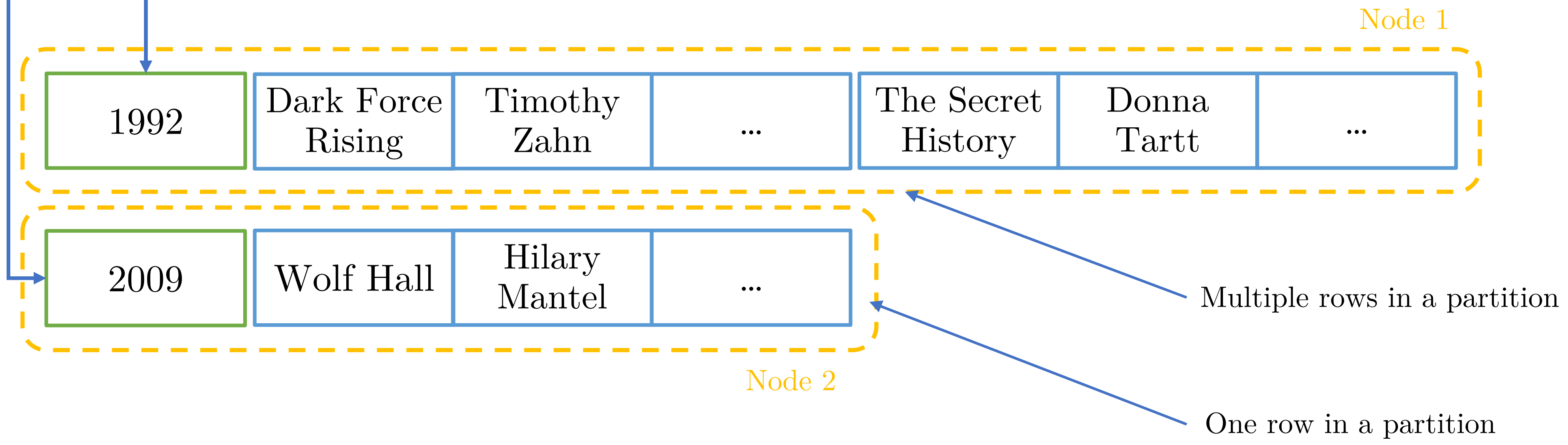
- Primary key uniquely identifies a row
  - Partition key – defines on which nodes data resides (must be present in WHERE clause).
  - Clustering key – defines a place within data files stored inside a node
- Cassandra does not distinguish inserts from updates. Everything is an “upsert” operation
  - Perfect fit for idempotent service design.
  - **`INSERT INTO urls(id, url) VALUES(?, ?) ==`**
    `UPDATE urls SET url = ? WHERE id = ?`

```
CREATE TABLE books_by_year (
   year INT,
   title TEXT,
   author TEXT,
   content TEXT,
   PRIMARY KEY ((year), title, author)
);
```



- Clustering columns allow to order rows within partition

Year	Title	Author	Content
1992	Dark Force Rising	Timothy Zahn	...
1992	The Secret History	Donna Tartt	...
2009	Wolf Hall	Hilary Mantel	...



Cassandra Query Language

- CQL DELETE statement marks given row or partition for removal
  - Immediate data removal could indicate that deleting replica is missing write operation
  - Records marked for deletion and called “Tombstones”
  - Tombstones are removed during data file compaction (background process)
- Every cell in Cassandra may have associated time-to-live (TTL)
  - Default TTL defined on table level
  - Specific TTL defined per row, or even per value of each column
- Do no take anything for granted in NoSQL world and ignore your RDBMS knowledge. Always check documentation and test
  - `INSERT INTO urls(id, url) VALUES(?, ?) IF NOT EXISTS`
  - `UPDATE books SET title = ? WHERE id = ? IF title = ?`
  - Implicit compare-and-set operation that requires C* nodes to run Paxos algorithm (consensus algorithm). Very costly operation

- Complex data types – maps, sets, list and counters
  - Example:

```
CREATE TABLE my-table (
   id     text,
   my-set set<text>,
   my-map map<text, int>,
   PRIMARY KEY (id)
);
```

  - Add and remove idempotently elements from map or set

```
UPDATE my-table SET my-set = my-set + ? WHERE id = ?
```

Cassandra in Tiny-URL Application

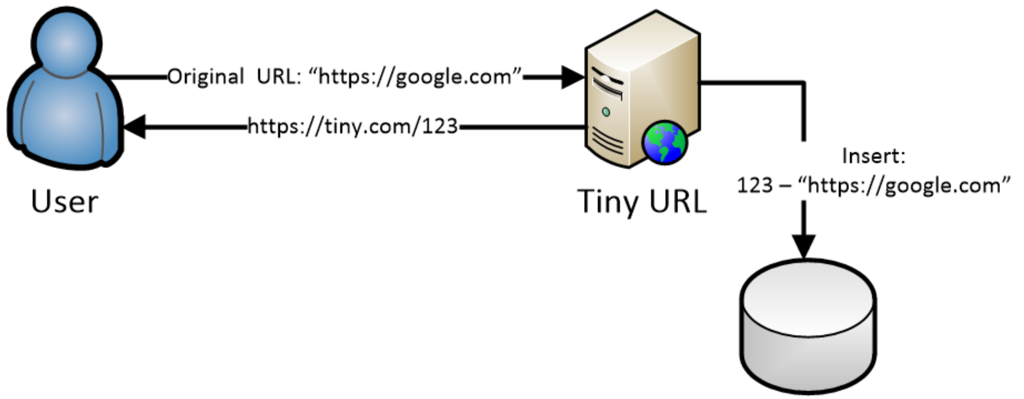
- Simple data model

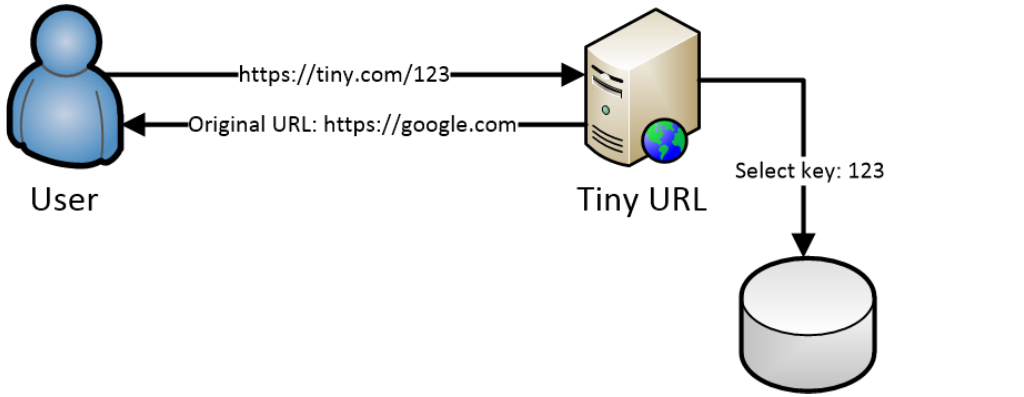
```
INSERT INTO urls(id, url) VALUES(?, ?) USING TTL ?

SELECT url FROM urls WHERE id = ? LIMIT 1

CREATE TABLE urls (
   id TEXT,
   url TEXT,
   PRIMARY KEY (id)
);
```

- Quorum reads and writes to guarantee immediate consistency





Summary

- Review Three-tier architecture
- Comparison of traditional RDBMS systems (ACID, SQL) and NoSQL
- Data sharding and consistent hashing
- CAP theorem and challenges of distributed stateful applications
- Short introduction to Apache Cassandra