CS4225/CS5425 Big Data Systems for Data Science

NoSQL and Basics of Distributed Databases

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg

School of Computing

Announcements

- HW1 has been released (due 10 Oct 11.59pm).
- Q&A session will be held at 8.30pm today (use Tutorial link in LumiNUS to access link). It is optional and will be recorded.

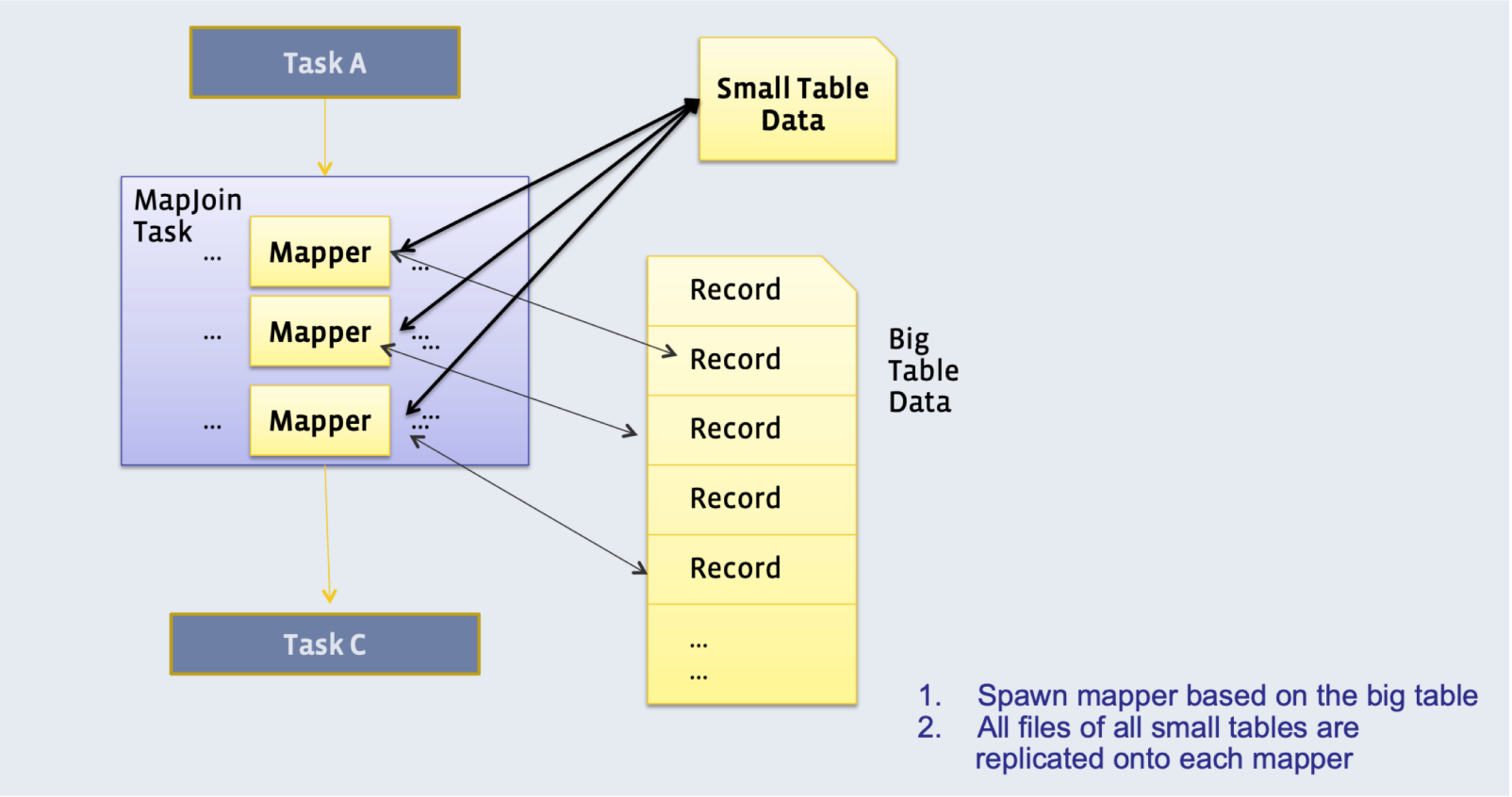
Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	19 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	
5	9 Sep	NoSQL Overview 1		Assignment 1 released
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 2 released
8	7 Oct	Apache Spark 2		Assignment 1 due (10 Oct)
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	Assignment 2 due (31 Oct)
12	4 Nov	Deepavali – No Class		
13	11 Nov	**Test**		

Announcements

- **Checklist if unable to login to cluster:**
  - You are using NUS VPN
  - You activated cluster permission (https://mysoc.nus.edu.sg/~myacct/)
  - You are using your SOC account userid and password (may be different from your NUS account details)
  - ssh command should be: "*ssh <SOC ID>@xcnd26.comp.nus.edu.sg*" (or *xcnd27*, …, *xcnd29*)
  - If there are still issues, please email me

Clarification: Map Join (or Broadcast Join)

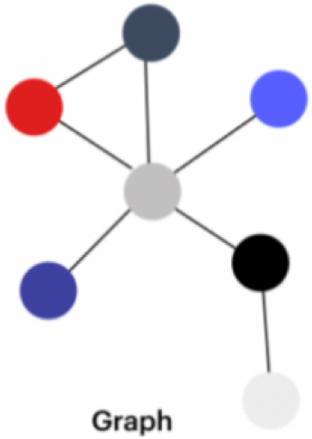
- Requires one of the tables to fit in memory
  - All mappers store a copy of the small table
  - They iterate over the big table, and join the records with the small table



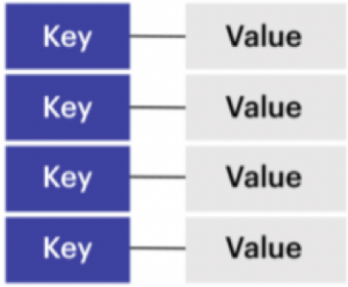
Recap: NoSQL

- NoSQL has come to stand for "Not Only SQL", i.e. using relational and non-relational databases alongside one another, each for the tasks they are most suited for

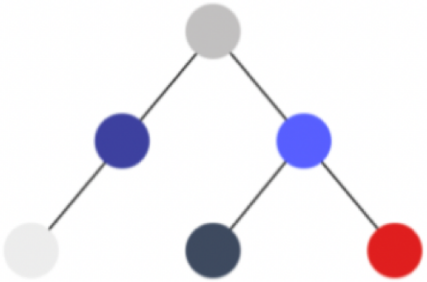
Column



Graph



Key-Value



Document

Recap: Key-Value Stores

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

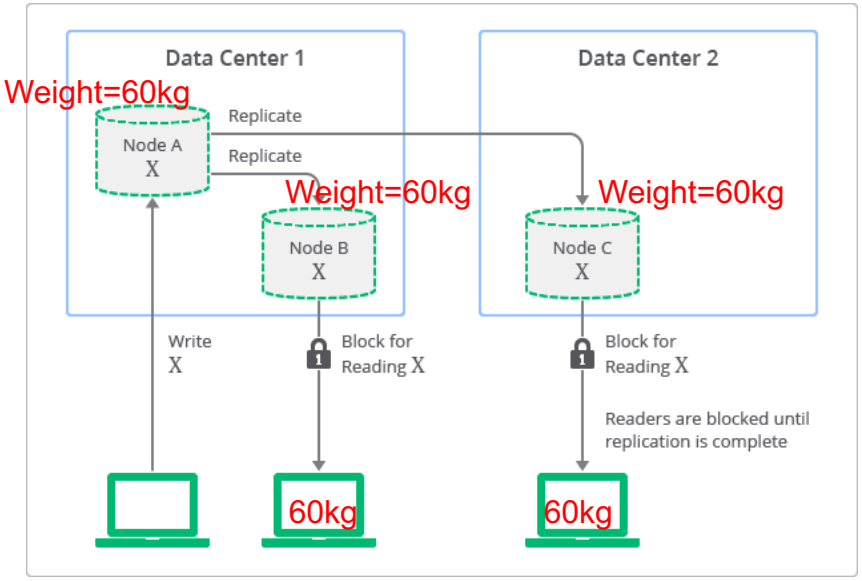
Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

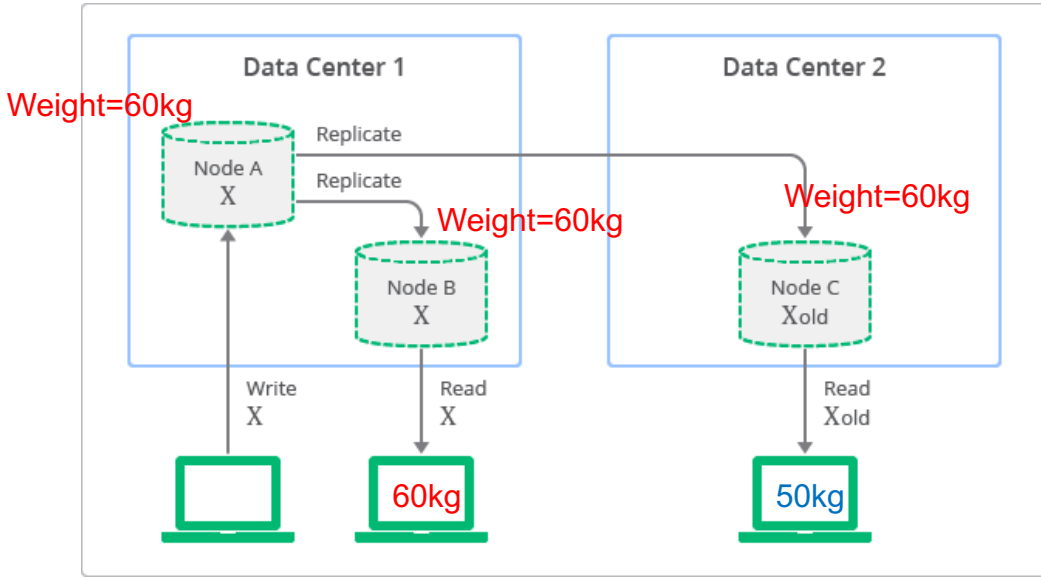
- Operations: get, put, multi-get, multi-put, range queries
- Types: persistent, non-persistent (in-memory)

Recap: Document Stores

- **Collections** have multiple **documents** (i.e. JSON object)
- Operations: CRUD (create, update, read, destroy) – similar to SQL queries, but without fixed schema

Recap: Strong vs Eventual Consistency





Strong consistency: any reads immediately after an update must give the same result on all observers

Eventual consistency: if the system is functioning and we wait long enough, eventually all reads will return the last written value

Recap: Duplication

Users

```
[
  {
    "user_id": 100,
    "name": "Alice"
  },
  {
    "user_id": 101,
    "name": "Bob"
  },
]
```

Posts

```
[
  {
    "post_id": "1",
    "post": "Hello world",
    "user_id": 100
  },
  {
    "post_id": "2",
    "post": "Bye world",
    "user_id": 101
  },
]
```


Posts (+ username)

```
[
  {
    "post_id": "1",
    "post": "Hello world",
    "user_id": 100,
    "name": "Alice",
  },
  {
    "post_id": "2",
    "post": "Bye world",
    "user_id": 101,
    "name: "Bob"
  },
]
```

- 'Storage is cheap: why not just duplicate data to improve efficiency?'
- Tables are designed around the queries we expect to receive
- Leads to a new problem: what if user changes their name? (this needs to be propagated to multiple tables)

Recap: Pros & Cons of NoSQL Systems

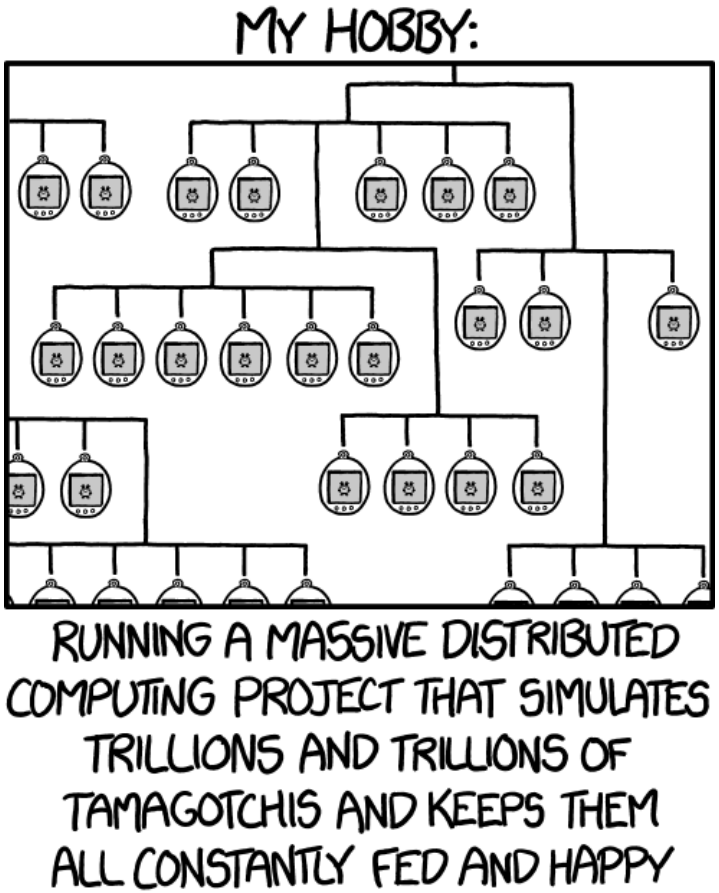
Pros	Cons
+ **Flexible / Dynamic Schema**: suitable for less well-structured data	- **No declarative query language**: query logic (e.g. joins) may have to be handled on the application side, which can add additional programming
+ **Massive Scalability / High Performance**	- **Relaxed consistency**: fewer guarantees; application may receive stale data that may need to be handled on the application side
+ **High Availability**	
~~+ **Relaxed consistency:** higher performance and availability~~	

Conclusion: no one size fits all. Depends on needs of application: complexity of queries (joins vs simple read/writes), importance of consistency (e.g. financial transactions vs tweets), data volume / need for availability.

Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

Today's Plan

- **Basic Concepts of Distributed Databases**
- Data Partitioning
- Query Processing in NoSQL



Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

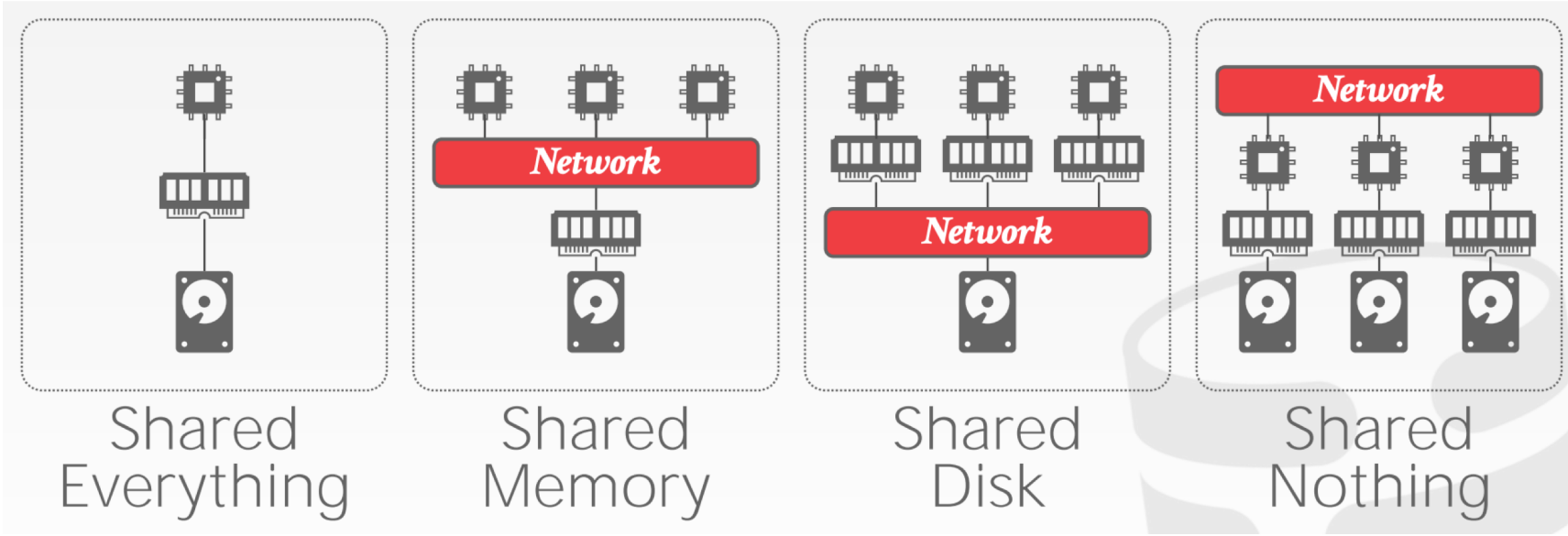
Data Transparency

- Users should not be required to know how the data is physically distributed, partitioned, or replicated.
- A query that works on a single node database should still work on a distributed database.

Assumption of Distributed Databases

- All nodes in a distributed database are well-behaved (i.e. they follow the protocol we designed for them)
  - (Out of syllabus: If we cannot trust the other nodes, we need a **Byzantine Fault Tolerant** protocol; blockchains fall into this category)

Distributed Database Architectures



Examples:

- Shared Everything: Single node DBMS
- Shared Memory: Supercomputers
- Shared Disk: Cloud databases
- Shared Nothing: NoSQL (mostly)

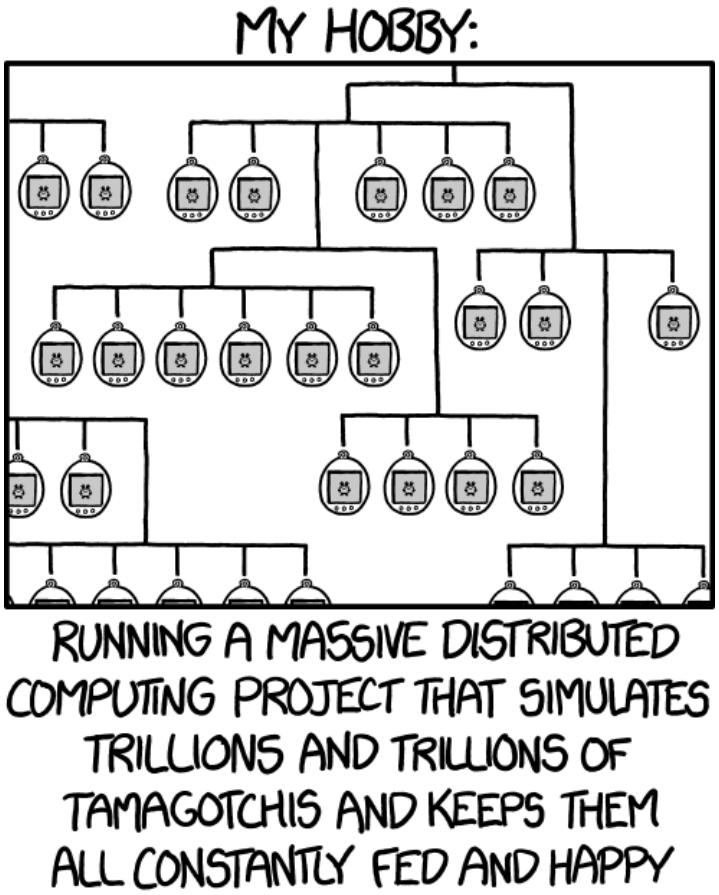
ORACLE RAC presto snowflake

Cockroach LABS H-Store redis cassandra mongoDB CouchDB relax MySQL Cluster

Credit: Andy Pavlo (CMU CS15-446)

Today's Plan

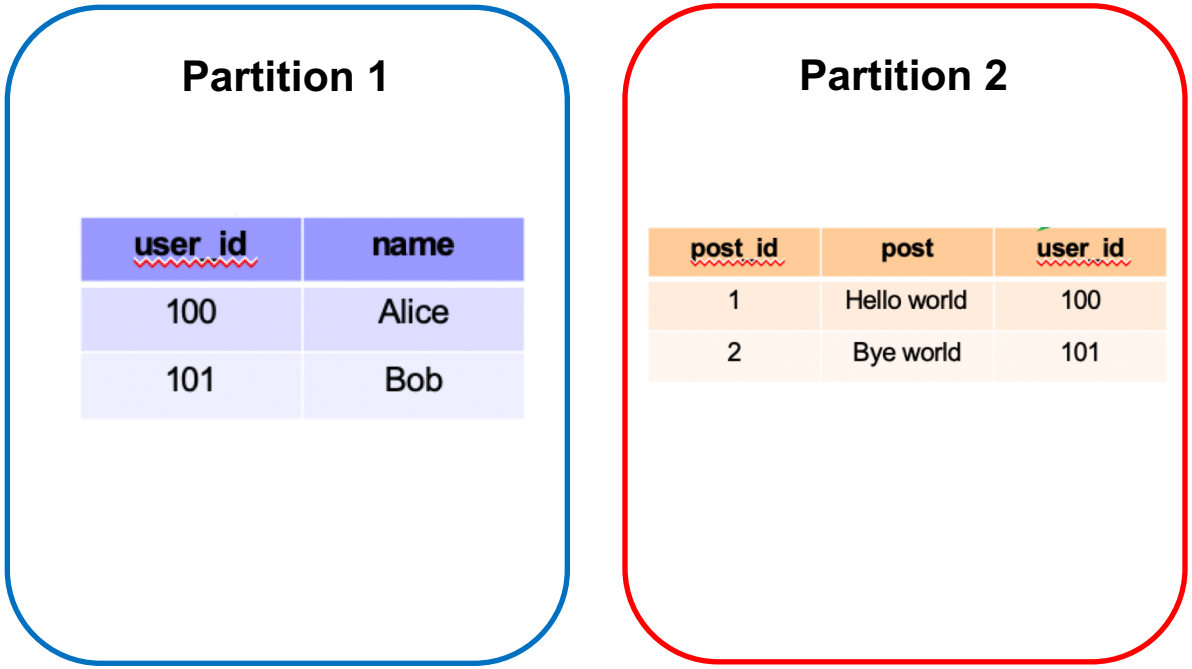
- Basic Concepts of Distributed Databases
- **Data Partitioning**
- Query Processing in NoSQL



Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

Table Partitioning

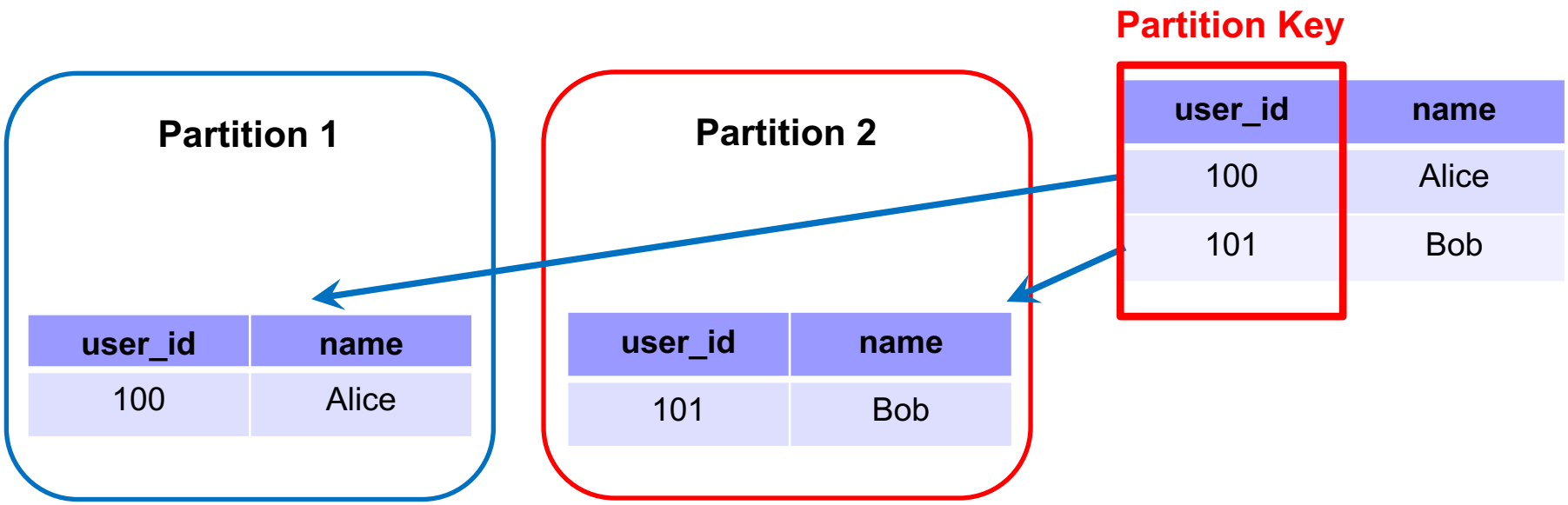
- Put different tables (or collections) on different machines



- **Problem**: scalability – each table cannot be split across multiple machines

Horizontal Partitioning

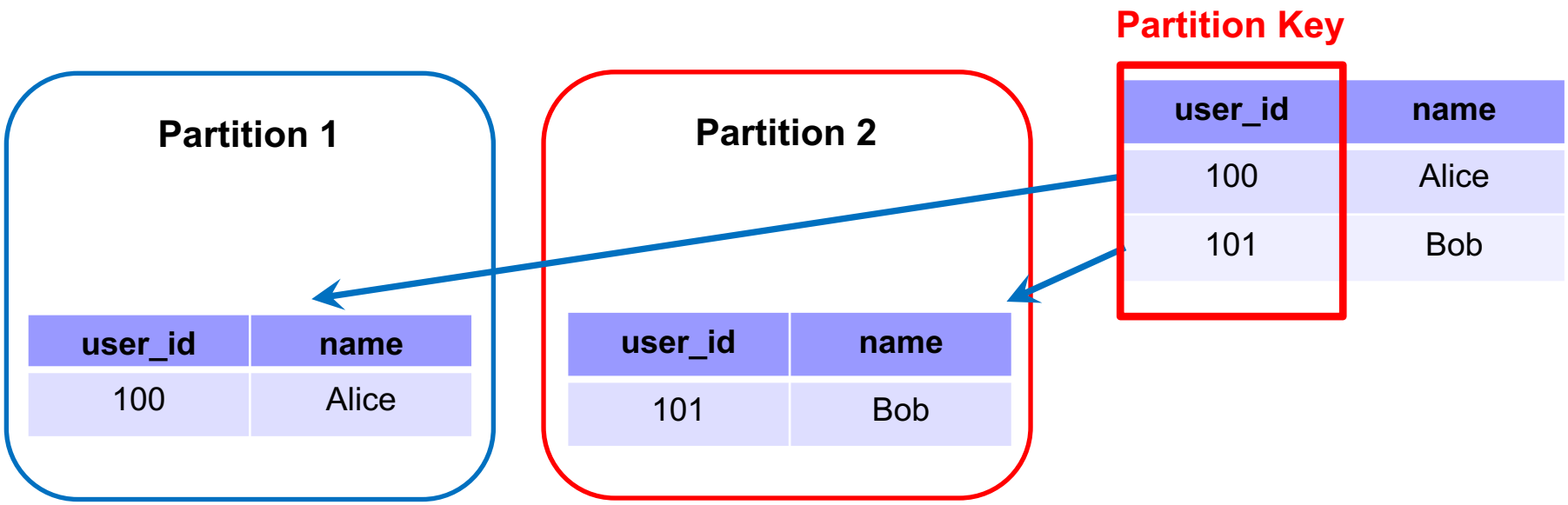
- Different tuples are stored in different nodes



- Also called "sharding"
- **Partition Key** (or "shard key") is the variable used to decide which node each tuple will be stored on
  - **How to choose partition key?** If we often need to filter tuples based on a column, or "group by" a column, then that column is a suitable partition key
  - Example: if we filter tuples by user_id=100, and user_id is the partition key, then all the user_id=100 data will be on the same partition, improving locality and reducing network I/O.

Horizontal Partitioning

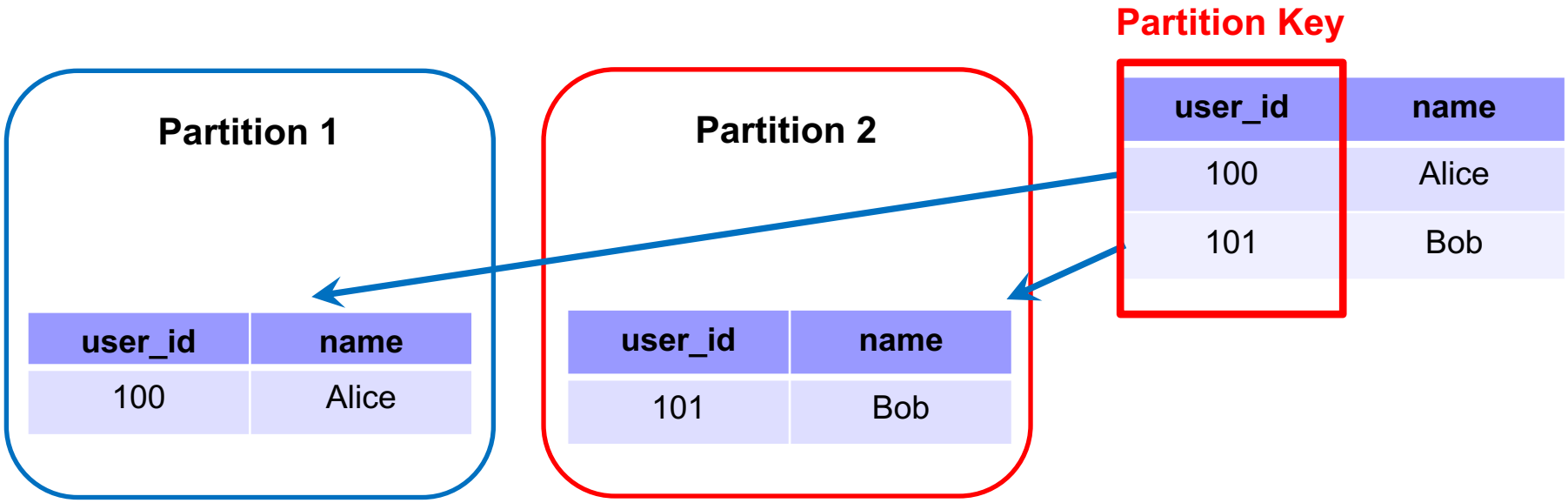
- Different tuples are stored in different nodes



- **Q**: imagine using each user's city, city_id as a partition key; when is this good / bad?

Horizontal Partitioning

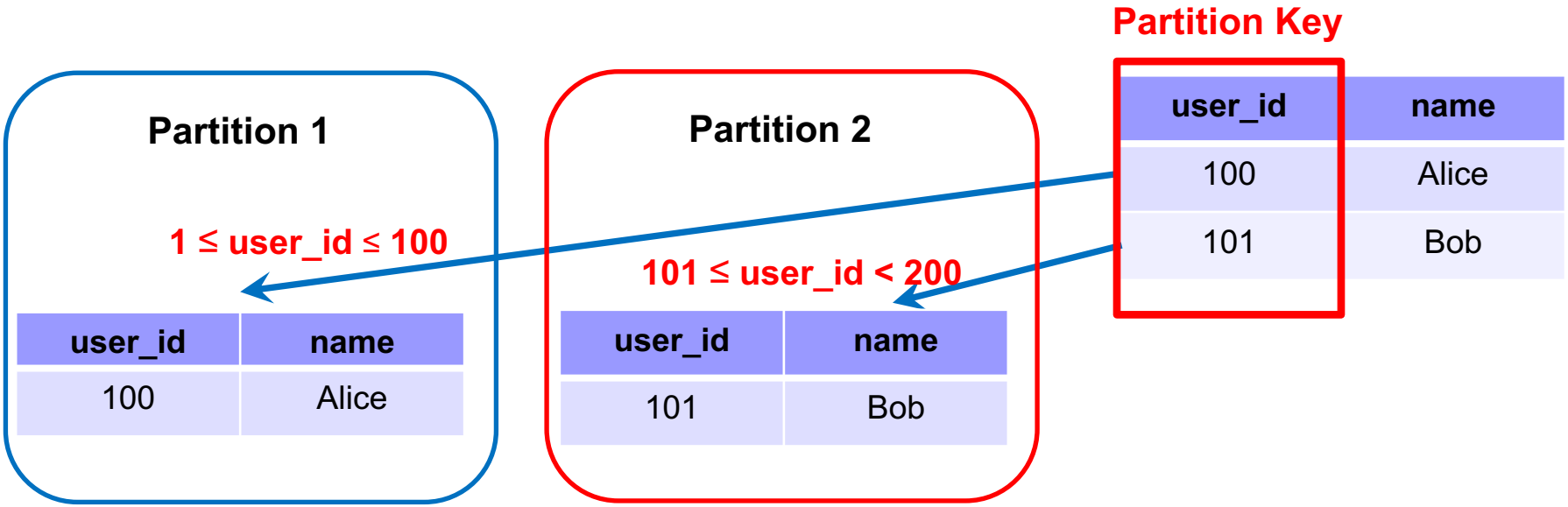
- Different tuples are stored in different nodes



- **Q**: imagine using each user's city, city_id as a partition key; when is this good / bad?
- **A**: good if we mostly aggregate data only within individual cities. Bad if there are too few cities (or cities are very imbalanced) and this causes a lack of scalability.

Horizontal Partitioning – Range Partition

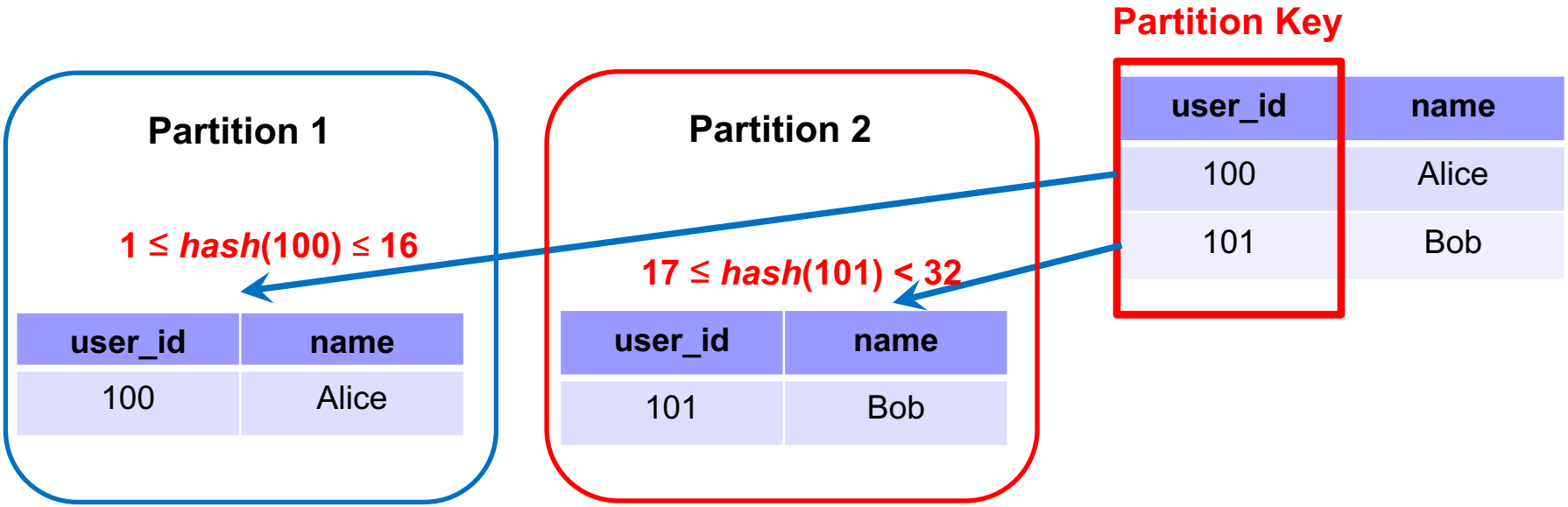
- Different tuples are stored in different nodes



- **Range Partition**: split partition key based on range of values
  - Beneficial if we need range-based queries, but can lead to imbalanced shards
  - Splitting the range is automatically handled by a balancer (it tries to keep the shards balanced)

Horizontal Partitioning – Hash Partition

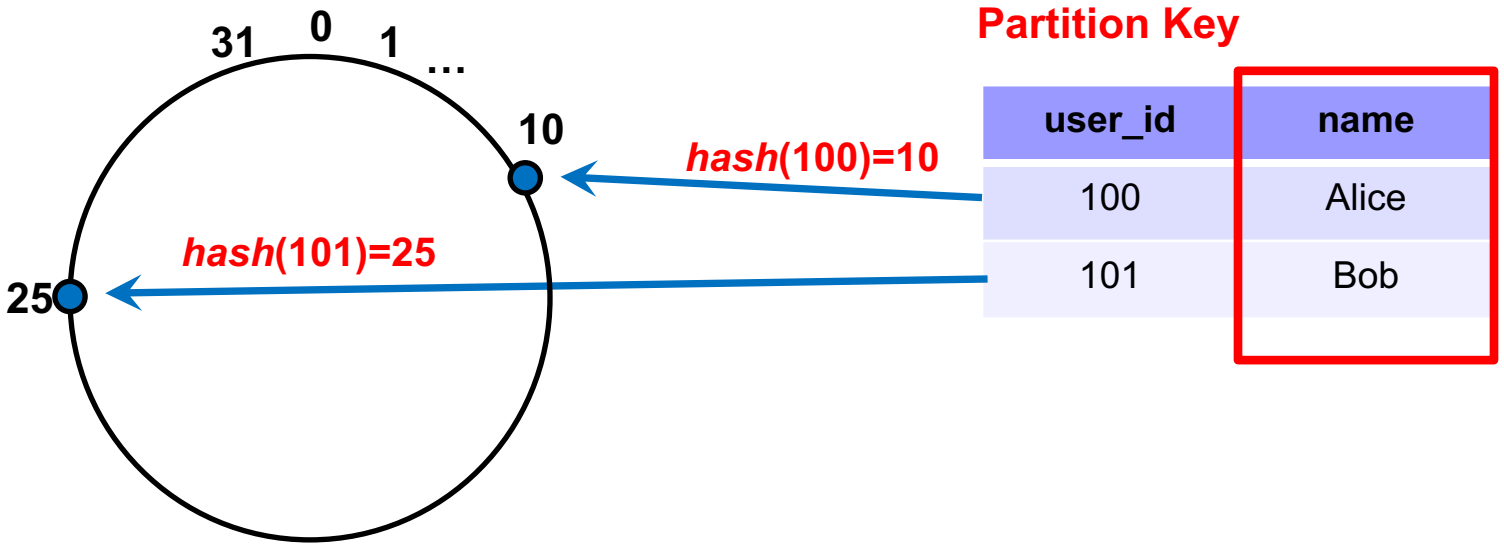
- Different tuples are stored in different nodes



- **Hash Partition**: hash partition key, then divide that into partitions based on ranges (or based on modulo)
  - Hash function automatically spreads out partition key values roughly evenly
  - **Question:** in previous approaches, how to add / remove a node? If we completely redo the partition, a lot of data may have to be moved around, which is inefficient.
  - **Answer**: consistent hashing

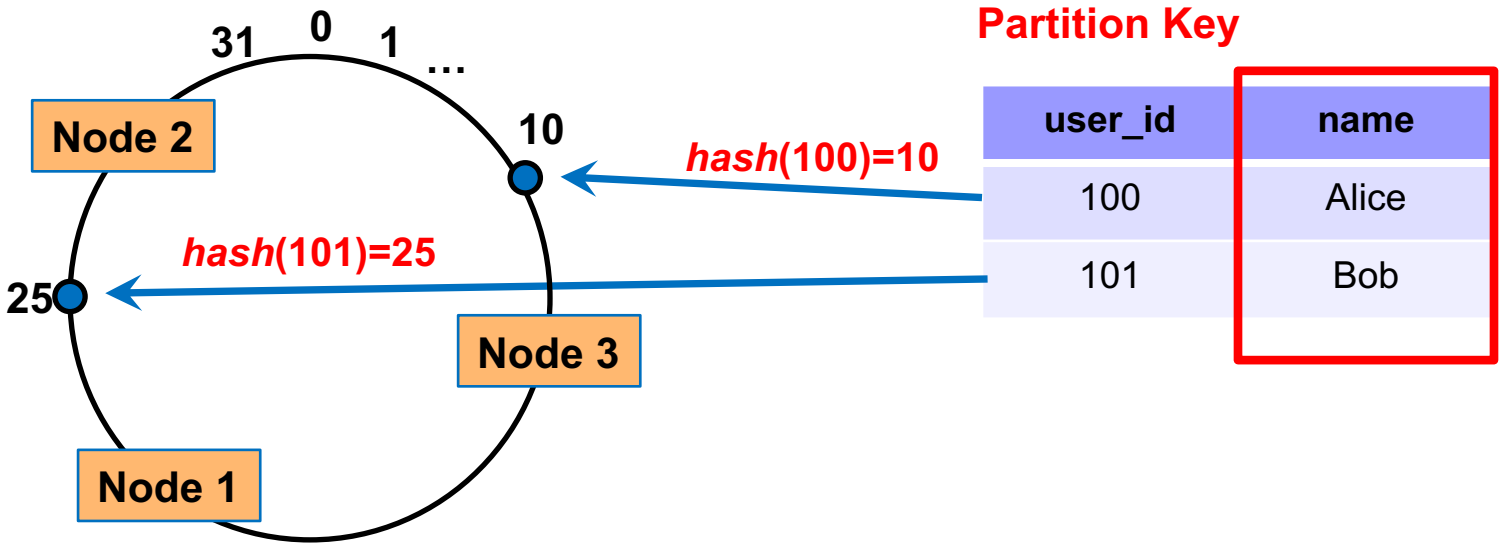
Consistent Hashing

- Think of the output of the hash function as lying on a circle:



Consistent Hashing

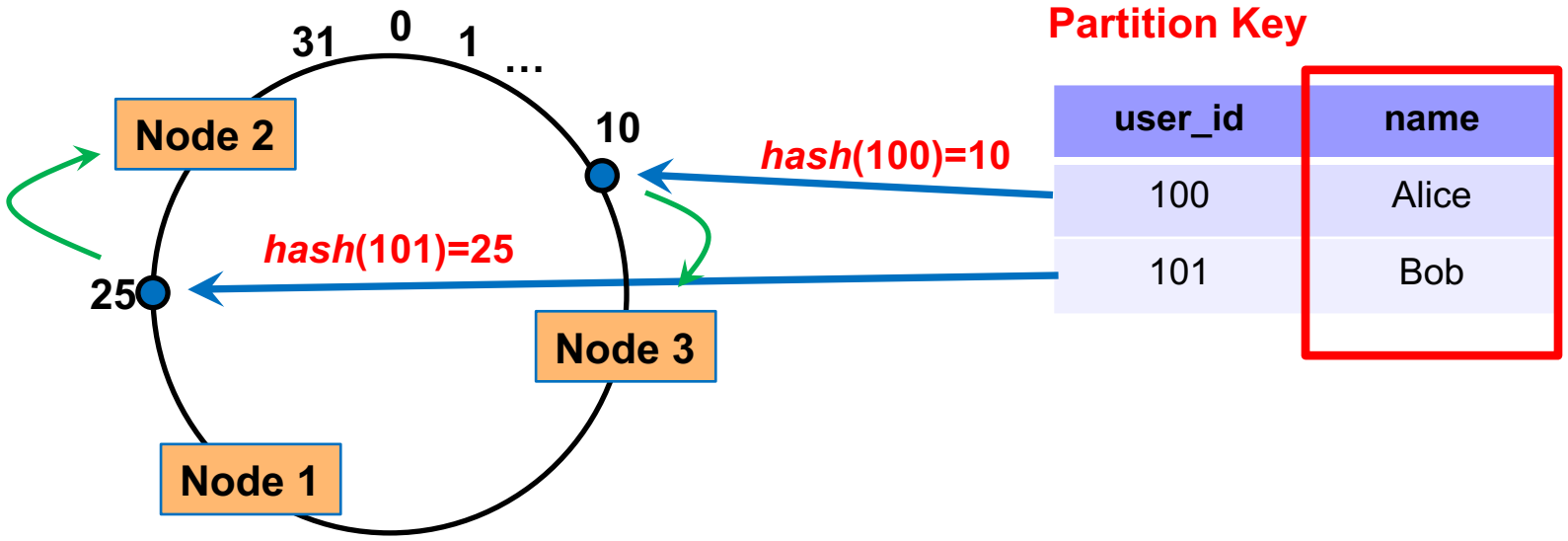
- Think of the output of the hash function as lying on a circle:



- **How to partition**: each node has a 'marker' (rectangles)

Consistent Hashing

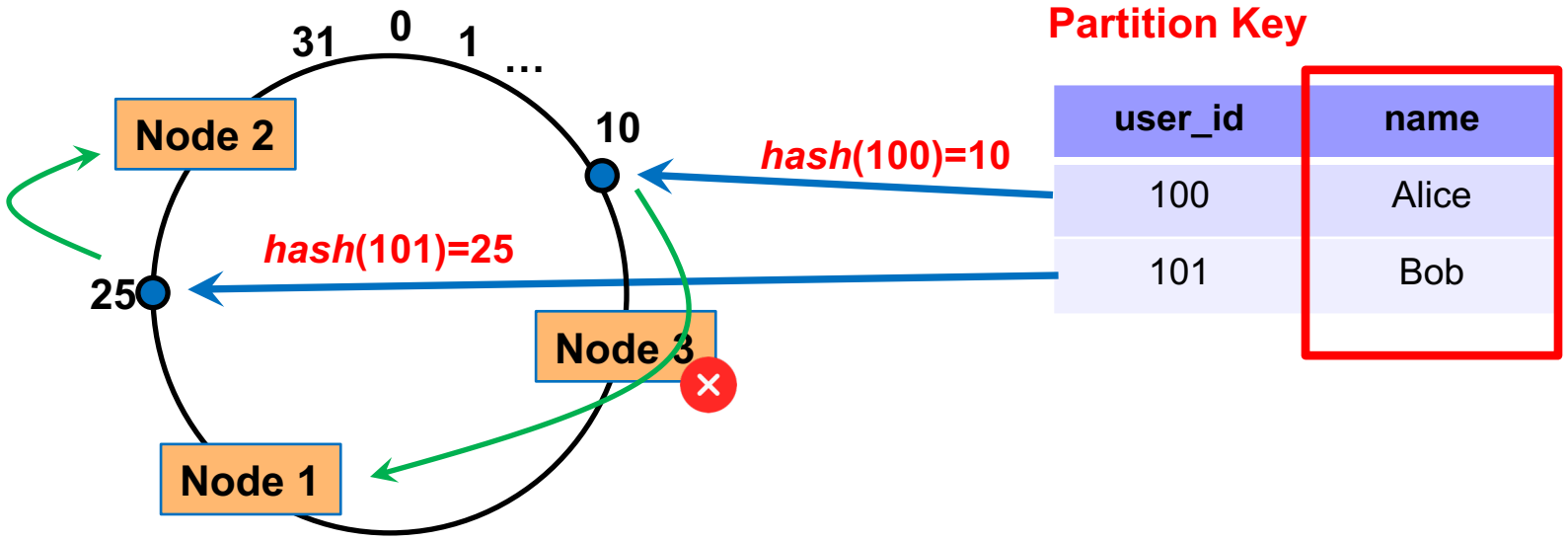
- Think of the output of the hash function as lying on a circle:



- **How to partition**: each node has a 'marker' (rectangles)
- Each tuple is placed on the circle, and assigned to the node that comes clockwise-after it

Consistent Hashing

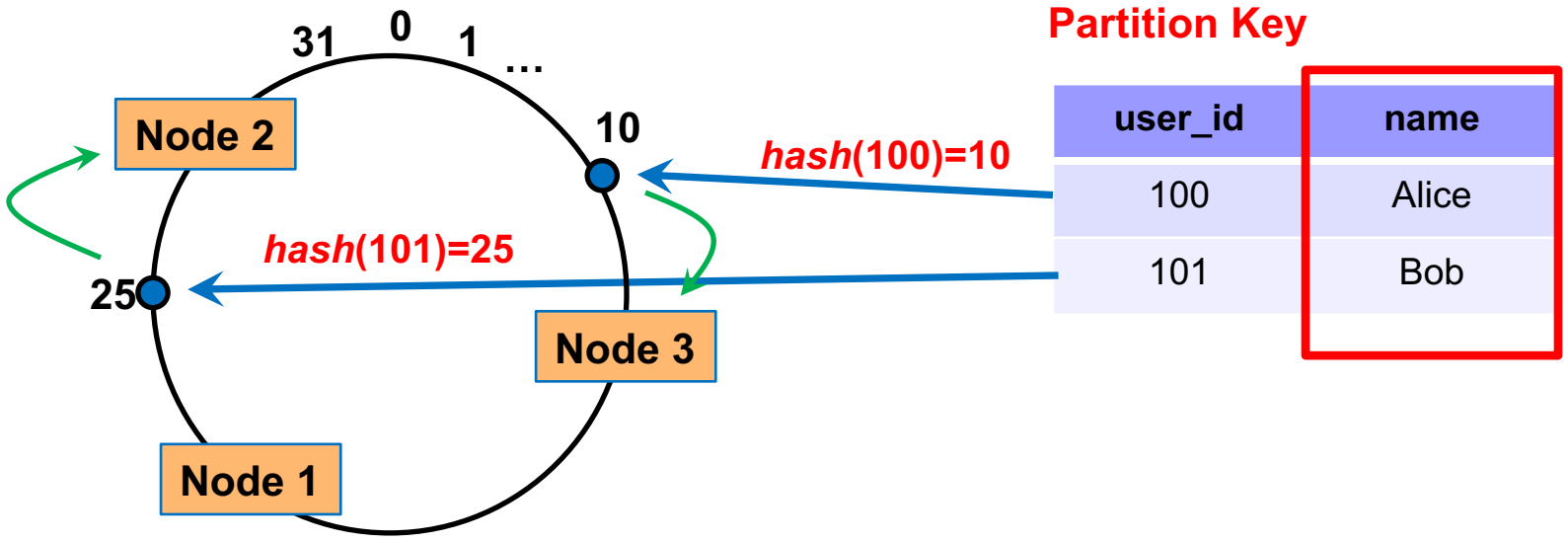
- Think of the output of the hash function as lying on a circle:



- **How to partition**: each node has a 'marker' (rectangles)
- Each tuple is placed on the circle, and assigned to the node that comes clockwise-after it
- To delete a node, we simply re-assign all its tuples to the node clock-wise after this one

Consistent Hashing

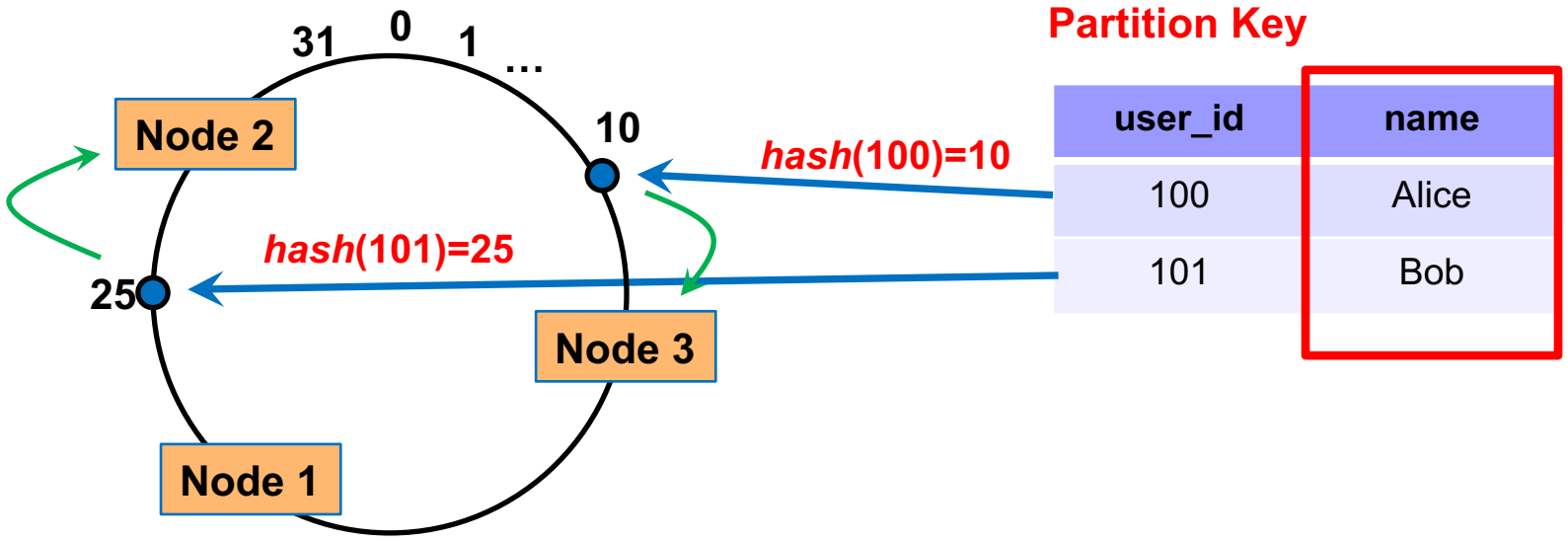
- Think of the output of the hash function as lying on a circle:



- **How to partition**: each node has a 'marker' (rectangles)
- Each tuple is placed on the circle, and assigned to the node that comes clockwise-after it
- To delete a node, we simply re-assign all its tuples to the node clock-wise after this one
- Similarly, nodes can be added by splitting the largest current node into two

Consistent Hashing

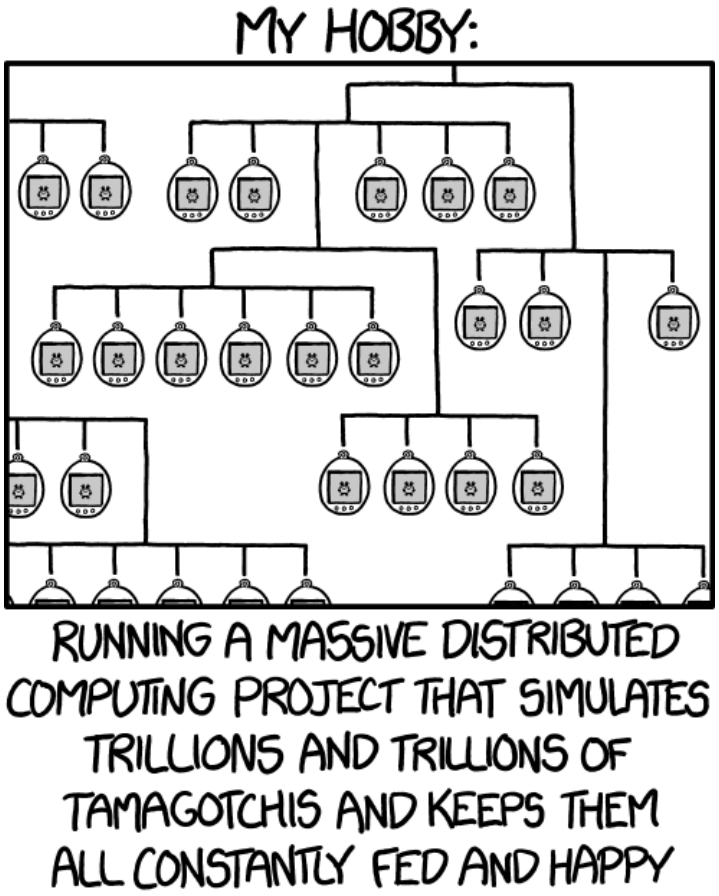
- Think of the output of the hash function as lying on a circle:



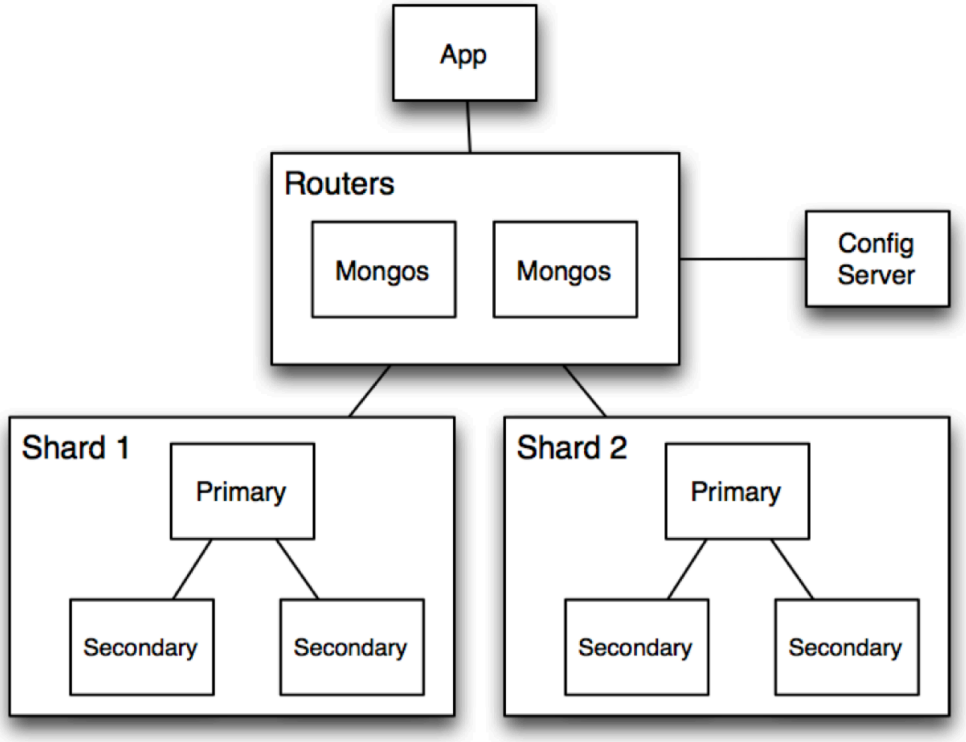
- **Simple replication strategy**: replicate a tuple in the next few (e.g. 2) additional nodes clockwise after the primary node used to store it

Today's Plan

- Basic Concepts of Distributed Databases
- Data Partitioning
- **Query Processing in NoSQL**

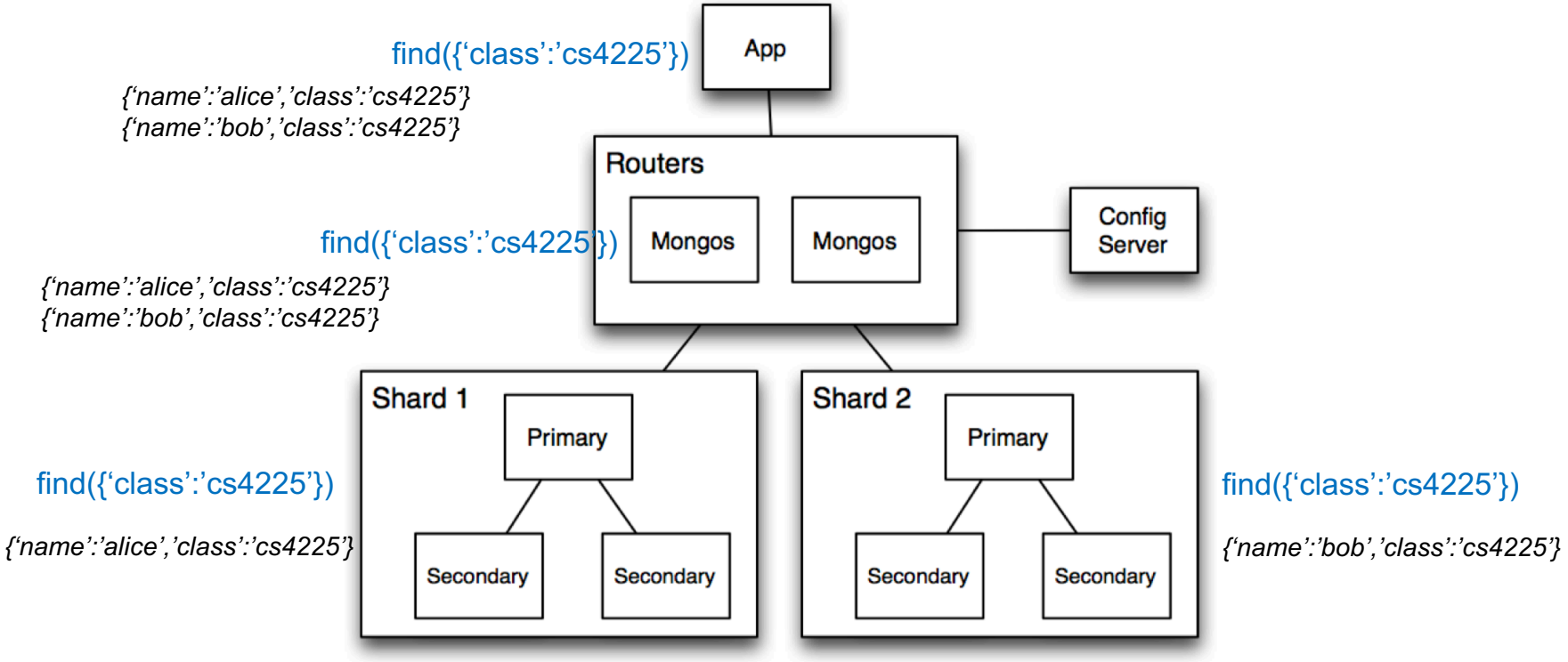


Architecture of MongoDB



- **Routers (mongos)**: handle requests from application and route the queries to correct shards
- **Config Server**: stores metadata about which data is on which shard
- **Shards**: store data, and run queries on their data

Example of Read or Write Query



1. Query is issued to a router (mongos) instance
2. With help of config server, mongos determines which shards should be queried
3. Query is sent to relevant shards
4. Shards run query on their data, and send results back to mongos
5. mongos merges the query results and returns the merged results

Conclusion: Reasons for Scalability & Performance of NoSQL

- **Horizontal partitioning**: as we get more and more data, we can simply partition it into more and more shards (even if individual tables become very large)
  - Horizontal partitioning improves speed due to parallelization.
- **Duplication:** Unlike relational DBs where queries may require looking up multiple tables (joins), using duplication in NoSQL allows queries to go to only 1 collection.
- **Relaxed consistency guarantees**: prioritize availability over consistency – can return slightly stale data

Acknowledgements

- CS4225 slides by He Bingsheng
- CMU CS15-445 slides by Andy Pavlo
- mongodb.com, https://learnmongodbthehardway.com/