Dynamo: Amazon's Highly Available Key-value Store

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Database and Database Management System

- Database: an organized collection of data.

Database and Database Management System

- Database: an organized collection of data.

- Database Management System (DBMS): a software that interacts with users, other applications, and the database itself to capture and analyze data.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.
- 1980s
  - Object databases: information is represented in the form of objects.

History of Databases

- 1960s
  - Navigational data model: hierarchical model (IMS) and network model (CODASYL).
  - Disk-aware
- 1970s
  - Relational data model: Edgar F. Codd paper.
  - Logical data is disconnected from physical information storage.
- 1980s
  - Object databases: information is represented in the form of objects.
- 2000s
  - NoSQL databases: BASE instead of ACID.
  - NewSQL databases: scalable performance of NoSQL + ACID.

Relational Databases Management Systems (RDMBSs)

- RDMBSs: the dominant technology for storing structured data in web and business applications.

- SQL is good
  - Rich language
  - Easy to use and integrate
  - Rich toolset
  - Many vendors

- They promise: ACID

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.

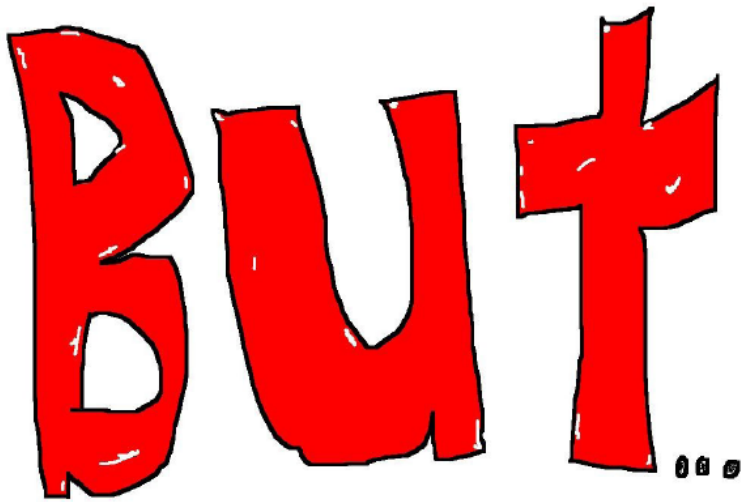
ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.
- Durability
  - Changes are written to a disk before a database commits a transaction so that committed data cannot be lost through a power failure.

RDBMS is Good ...



RDBMS Challenges

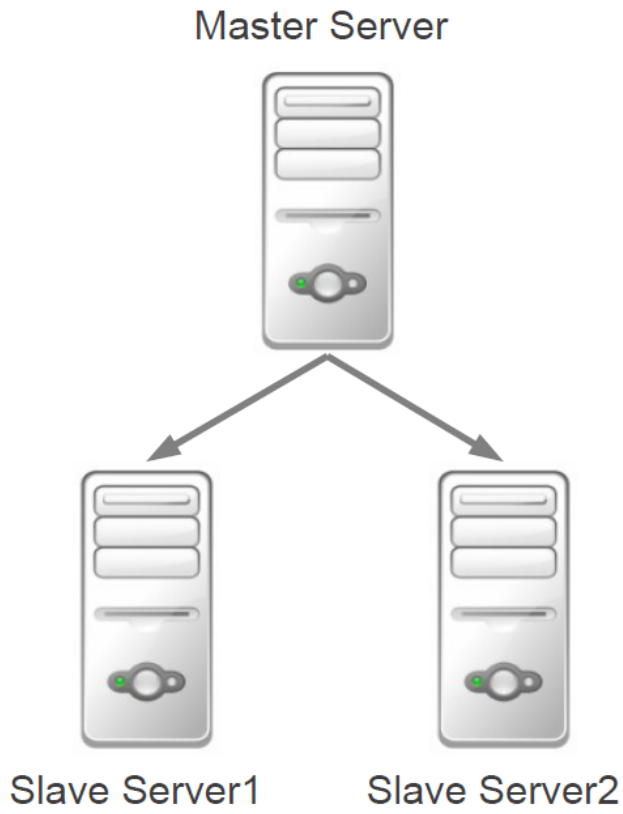
- Web-based applications caused spikes.
  - Internet-scale data size
  - High read-write rates
  - Frequent schema changes

Let's Scale RDBMSs

- RDBMS were not designed to be distributed.
- Possible solutions:
  - Replication
  - Sharding

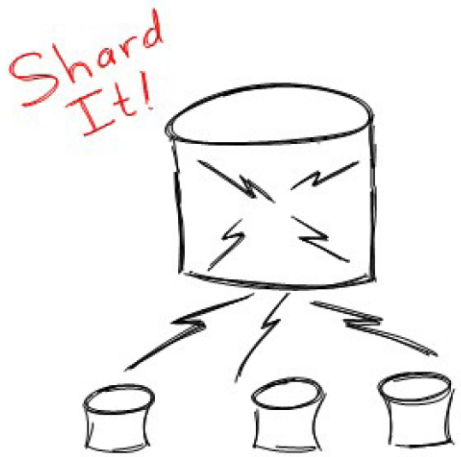
Let's Scale RDBMSs - Replication

- Master/Slave architecture
- Scales read operations

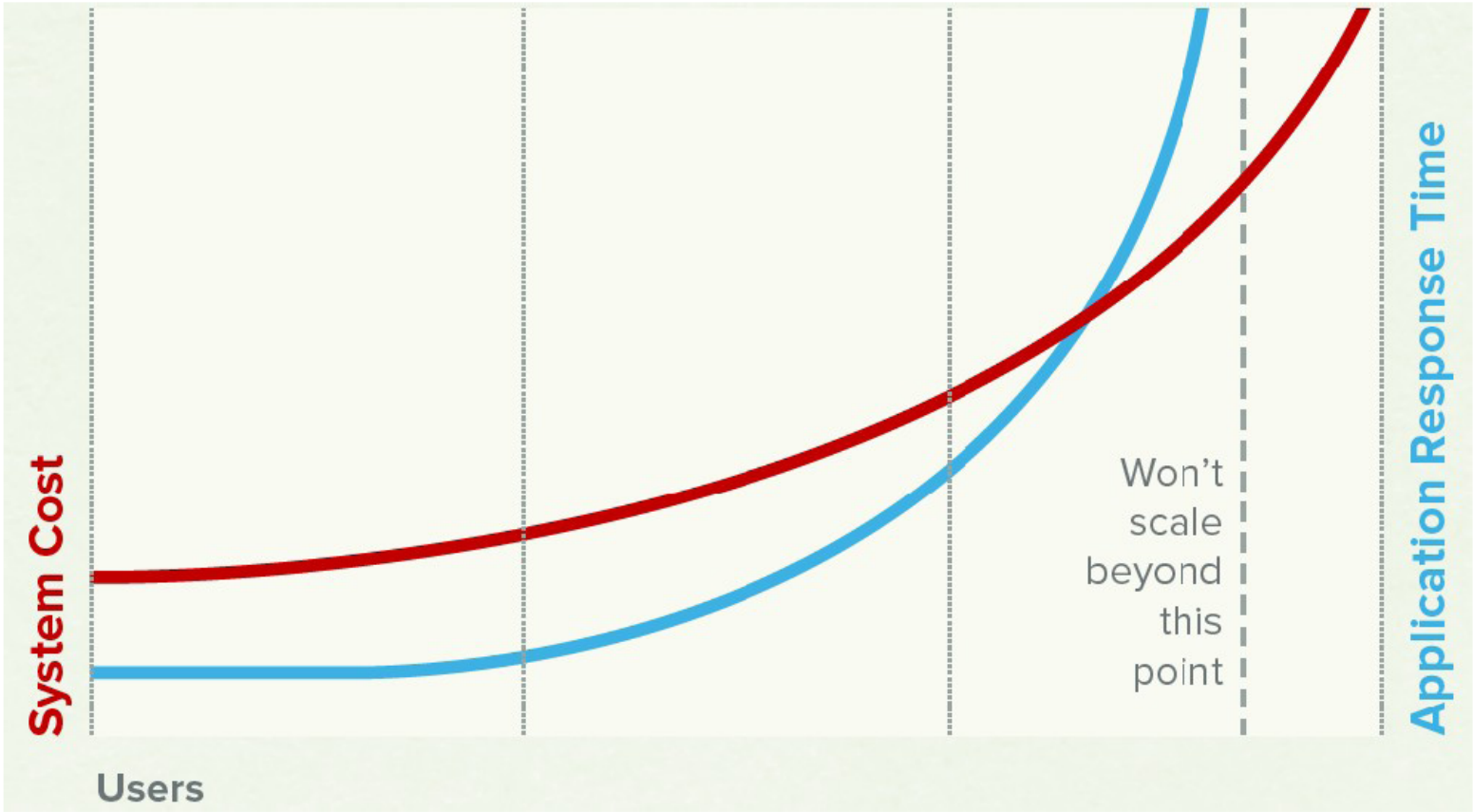


Let's Scale RDBMSs - Sharding

- Dividing the database across many machines.
- It scales read and write operations.
- Cannot execute transactions across shards (partitions).



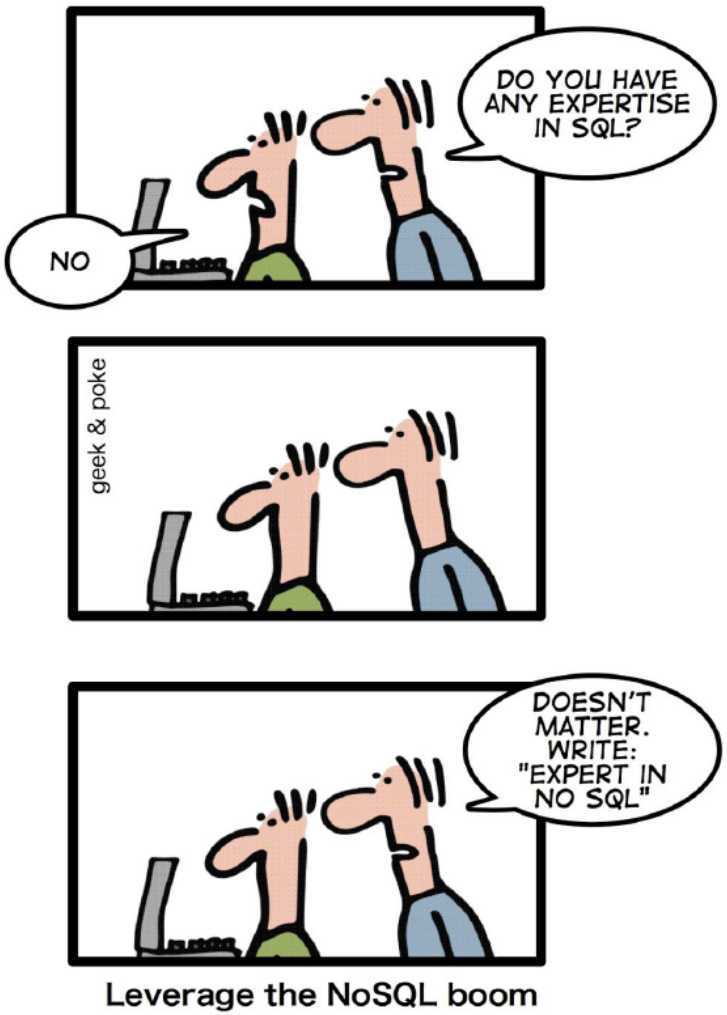
Scaling RDBMSs is Expensive and Inefficient



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]


Not
Only
SQL

HOW TO WRITE A CV



Leverage the NoSQL boom

NoSQL

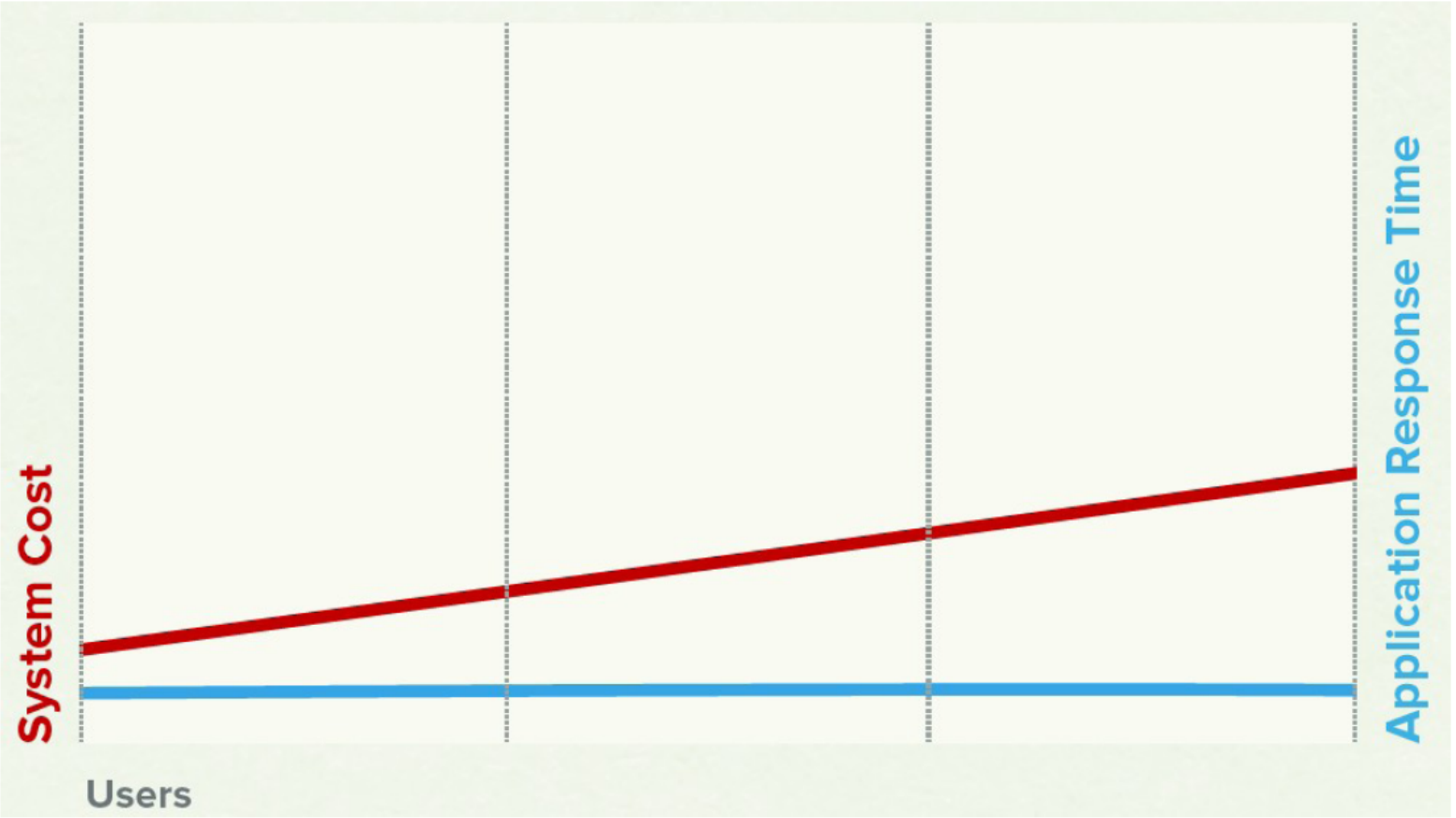
- Avoidance of unneeded complexity
- High throughput
- Horizontal scalability and running on commodity hardware
- Compromising reliability for better performance

The NoSQL Movement

- It was first used in 1998 by Carlo Strozzi to name his relational database that did not expose the standard SQL interface.
- The term was picked up again in 2009 when a Last.fm developer, Johan Oskarsson, wanted to organize an event to discuss open source distributed databases.

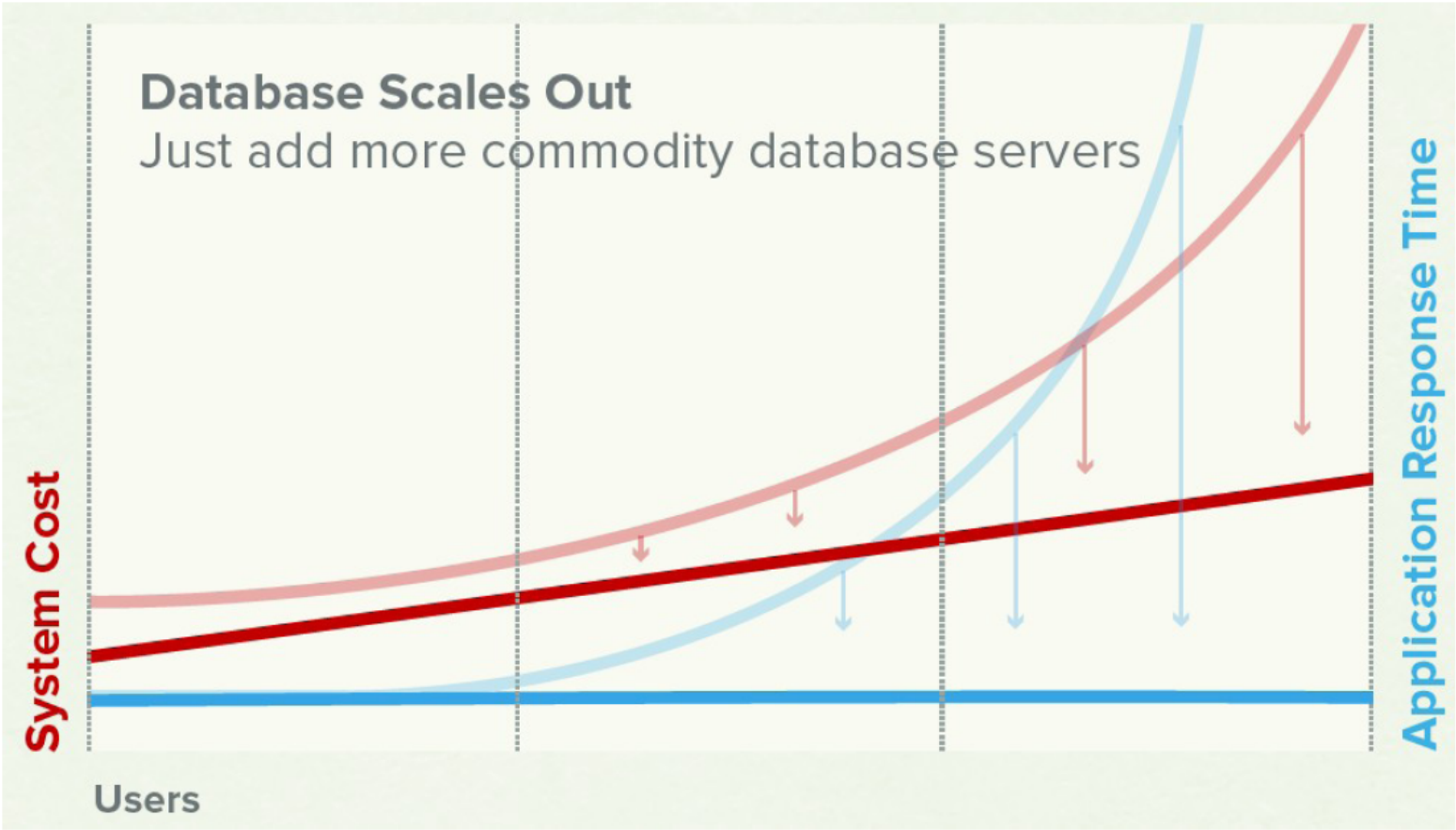
- The name attempted to label the emergence of a growing number of non-relational, distributed data stores that often did not attempt to provide ACID.

NoSQL Cost and Performance



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

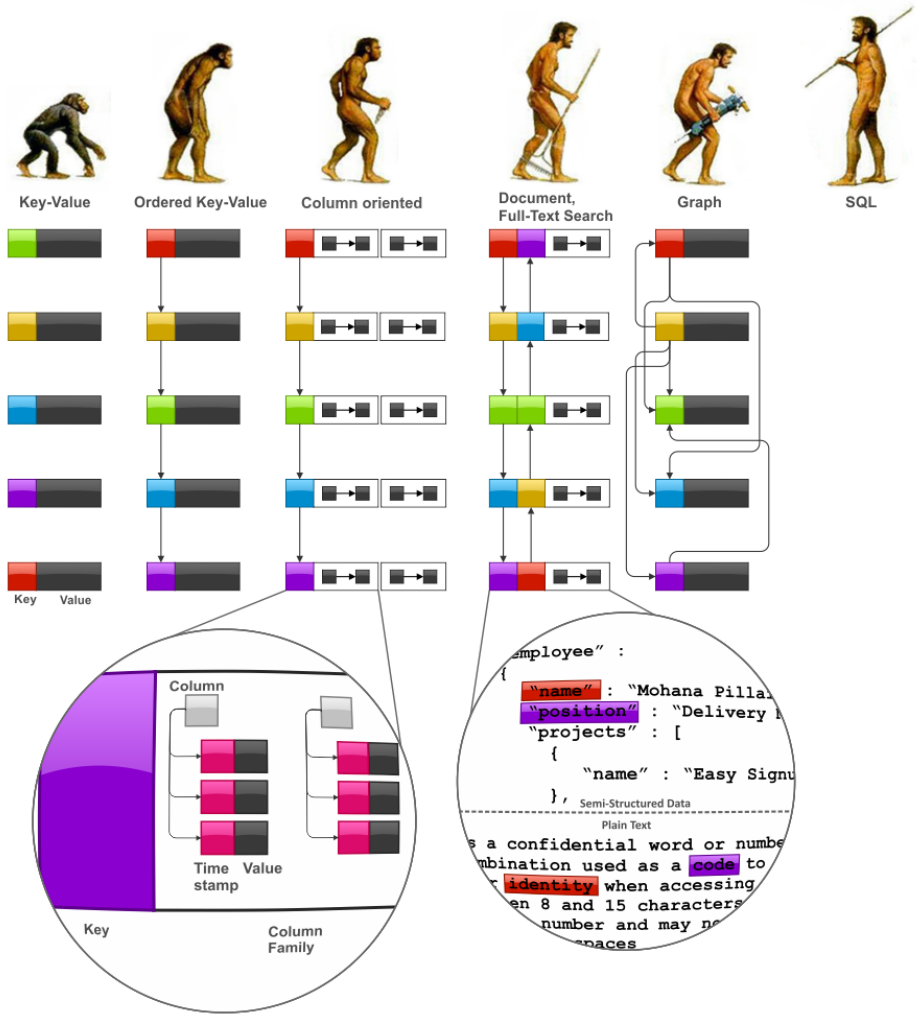
RDBMS vs. NoSQL



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

NoSQL Data Models

NoSQL Data Models



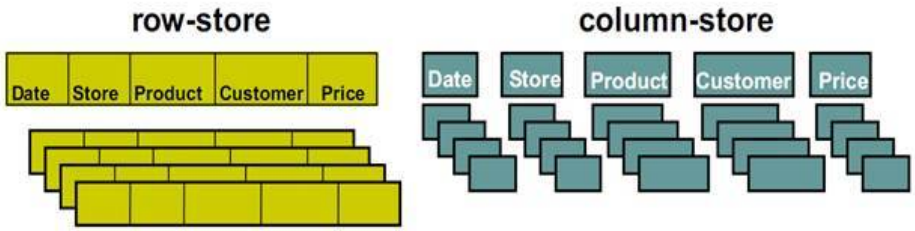
[http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques]

Key-Value Data Model

- Collection of key/value pairs.
- Ordered Key-Value: processing over key ranges.
- Dynamo, Scalaris, Voldemort, Riak, ...

Column-Oriented Data Model

- Similar to a key/value store, but the value can have multiple attributes (Columns).
- Column: a set of data values of a particular type.
- Store and process data by column instead of row.
- BigTable, Hbase, Cassandra, ...



Document Data Model

- Similar to a column-oriented store, but values can have complex documents, instead of fixed format.
- Flexible schema.
- XML, YAML, JSON, and BSON.
- CouchDB, MongoDB, ...

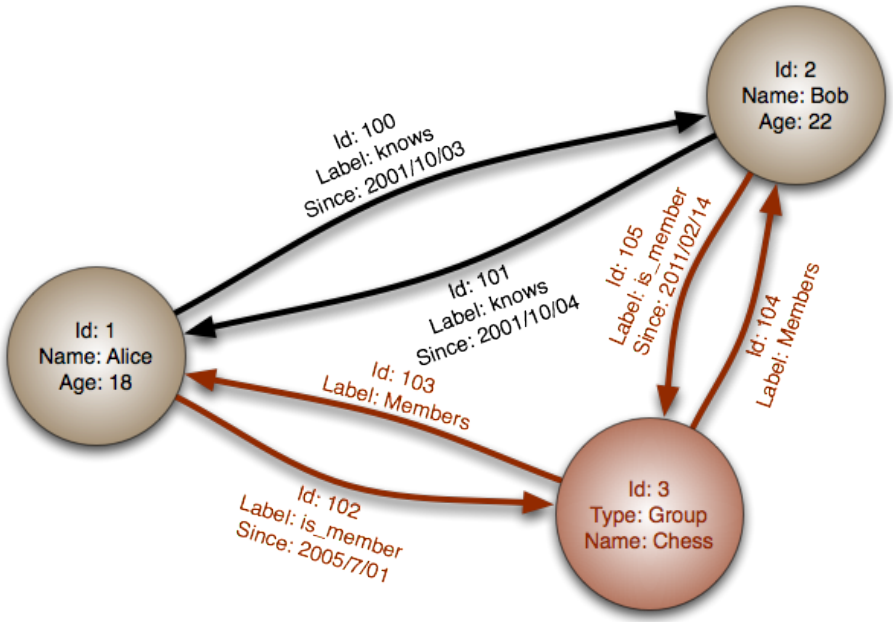
```
{
    FirstName: "Bob",
    Address: "5 Oak St.",
    Hobby: "sailing"
 }

 {
   FirstName: "Jonathan",
   Address: "15 Wanamassa Point Road",
   Children: [
        {Name: "Michael", Age: 10},
        {Name: "Jennifer", Age: 8},
   ]
  }
```

Graph Data Model

- Uses graph structures with nodes, edges, and properties to represent and store data.

- Neo4J, InfoGrid, ...

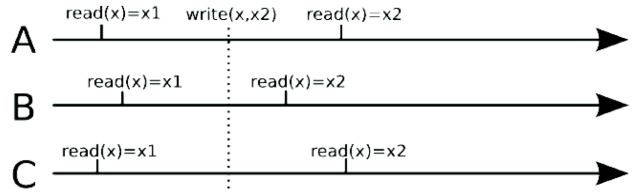


[http://en.wikipedia.org/wiki/Graph_database]

Consistency

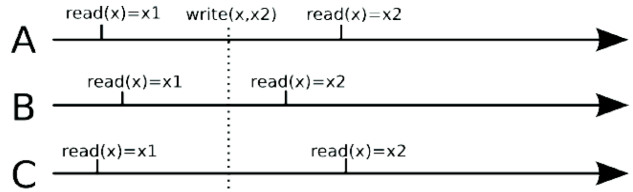
Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.

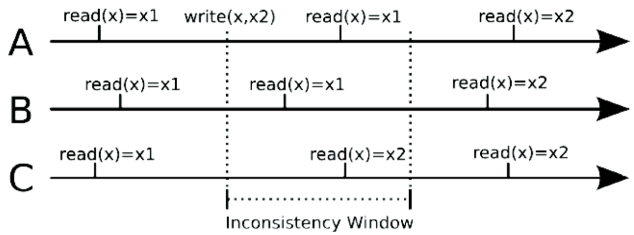


Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.



- Eventual consistency
  - Does not guarantee that subsequent accesses will return the updated value.
  - Inconsistency window.
  - If no new updates are made to the object, eventually all accesses will return the last updated value.



Quorum Model

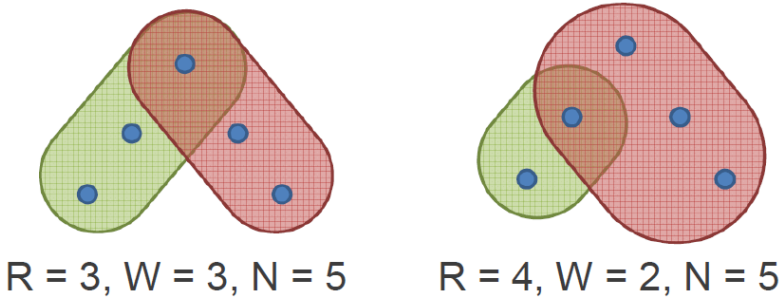
- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$

Quorum Model

- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$



Consistency vs. Availability

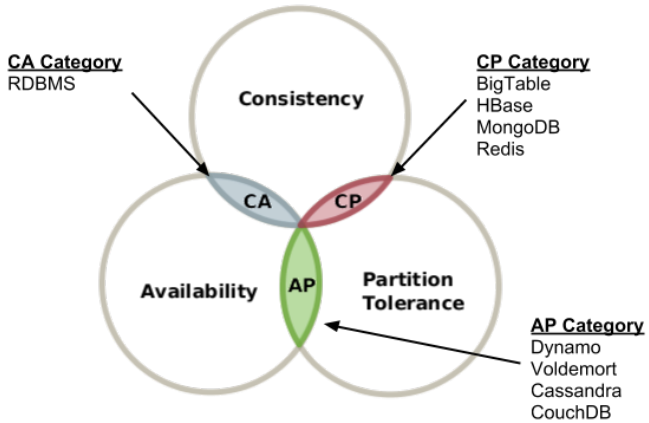
- The large-scale applications have to be reliable: availability + redundancy
- These properties are difficult to achieve with ACID properties.
- The BASE approach forfeits the ACID properties of consistency and isolation in favor of availability, graceful degradation, and performance.

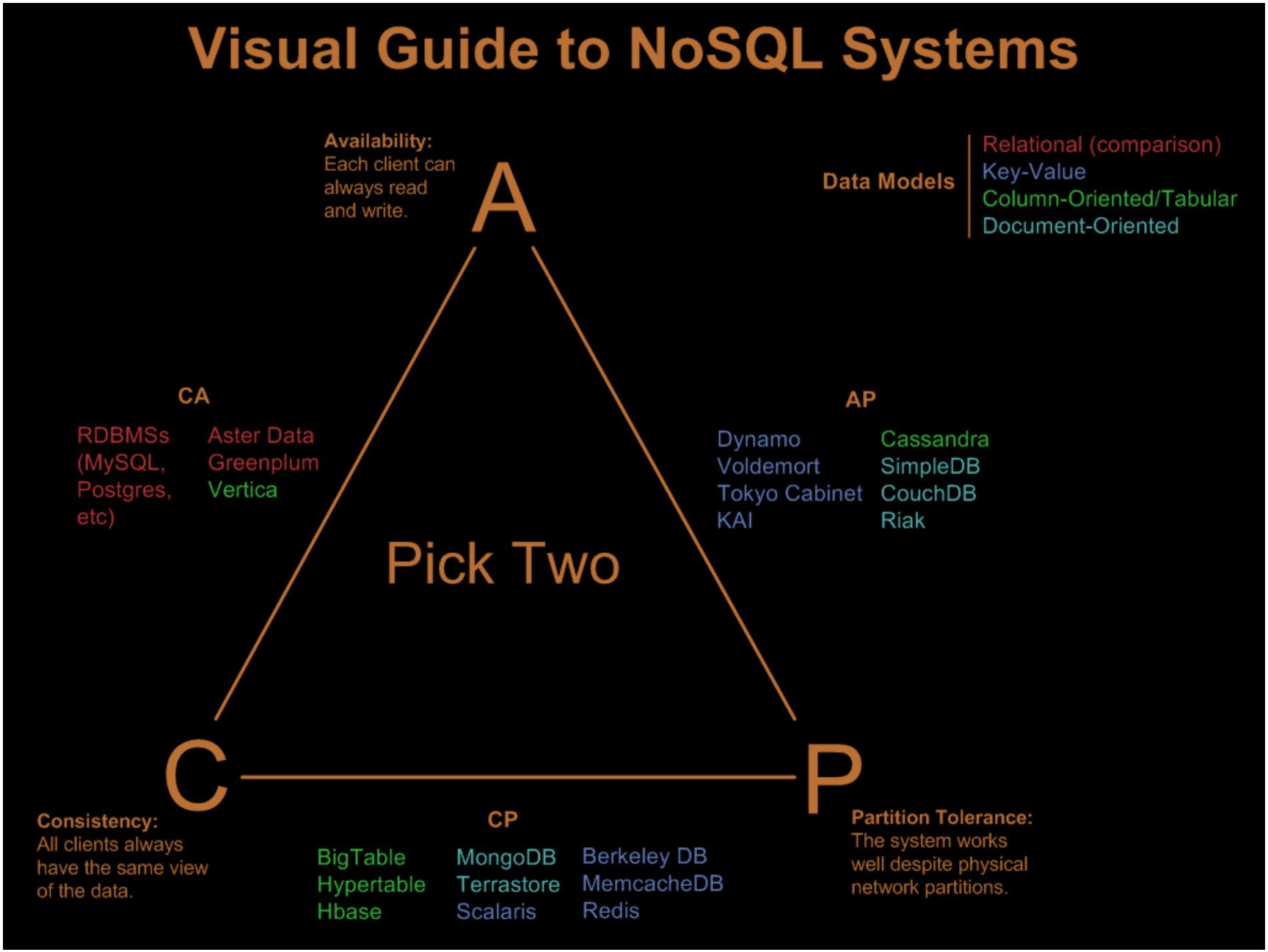
BASE Properties

- Basic Availability
  - Possibilities of faults but not a fault of the whole system.
- Soft-state
  - Copies of a data item may be inconsistent
- Eventually consistent
  - Copies becomes consistent at some later time if there are no more updates to that data item

CAP Theorem

- Consistency
  - Consistent state of data after the execution of an operation.
- Availability
  - Clients can always read and write data.
- Partition Tolerance
  - Continue the operation in the presence of network partitions.
- You can choose only two!




Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

Dyanmo

Dynamo

- Distributed key/value storage system
- Scalable
- Highly available
- CAP

Design Consideration

- It sacrifices strong consistency for availability: always writable

Design Consideration

- It sacrifices strong consistency for availability: always writable
- Incremental scalability

Design Consideration

- It sacrifices strong consistency for availability: always writable
- Incremental scalability
- Symmetry: every node should have the same set of responsibilities as its peers

Design Consideration

- It sacrifices strong consistency for availability: always writable
- Incremental scalability
- Symmetry: every node should have the same set of responsibilities as its peers
- Decentralization

Design Consideration

- It sacrifices strong consistency for availability: always writable
- Incremental scalability
- Symmetry: every node should have the same set of responsibilities as its peers
- Decentralization
- Heterogeneity

System Architecture

- Data partitioning
- Replication
- Data versioning
- Dynamo API
- Membership management

Data Partitioning

Partitioning

- If size of data exceeds the capacity of a single machine: partitioning
- Sharding data (horizontal partitioning).
- Consistent hashing is one form of automatic sharding.

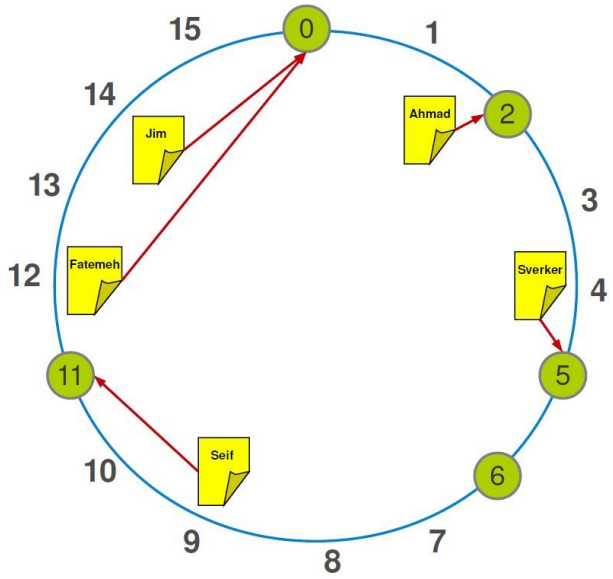
Consistent Hashing

- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

Consistent Hashing

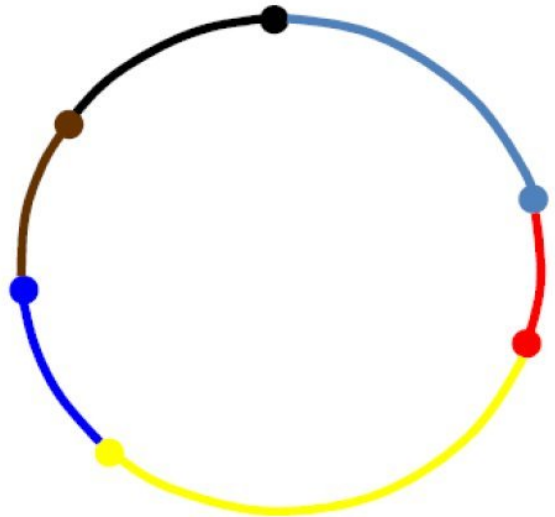
- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

```
hash("Fatemeh") = 12
hash("Ahmad") = 2
hash("Seif") = 9
hash("Jim") = 14
hash("Sverker") = 4
```



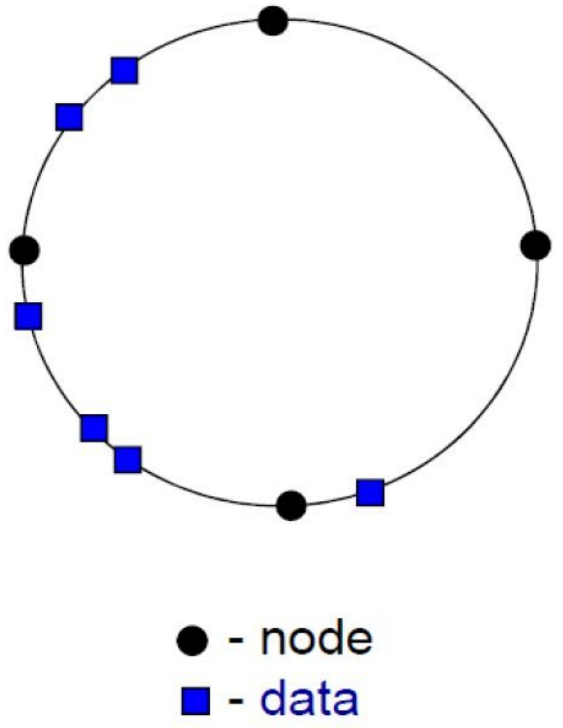
Load Imbalance (1/4)

- Consistent hashing may lead to imbalance.
- Node identifiers may not be balanced.



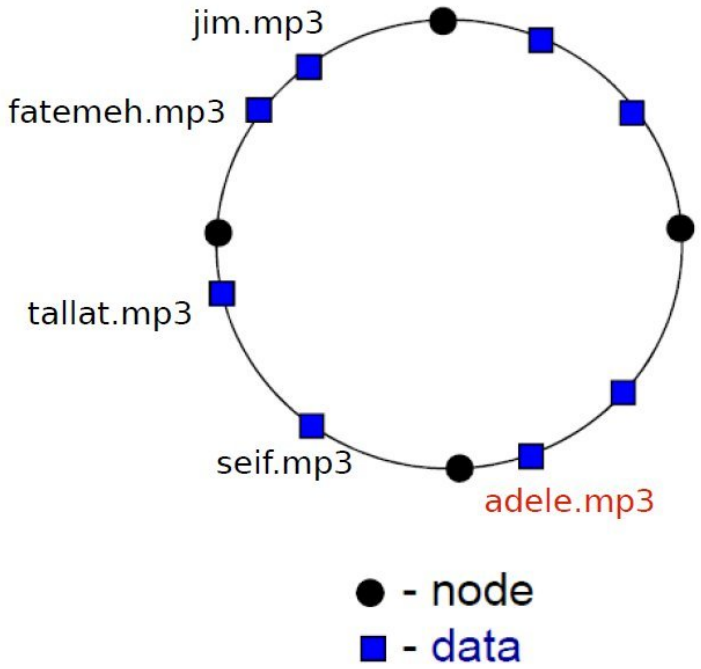
Load Imbalance (2/4)

- Consistent hashing may lead to imbalance.
- Data identifiers may not be balanced.



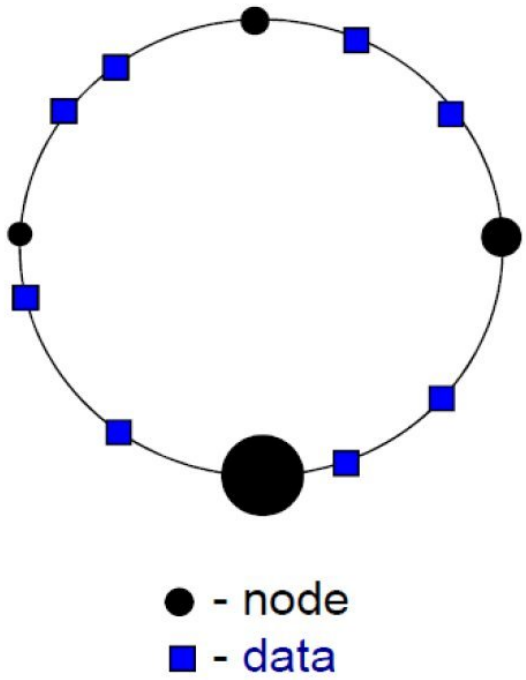
Load Imbalance (3/4)

- Consistent hashing may lead to imbalance.
- Hot spots.



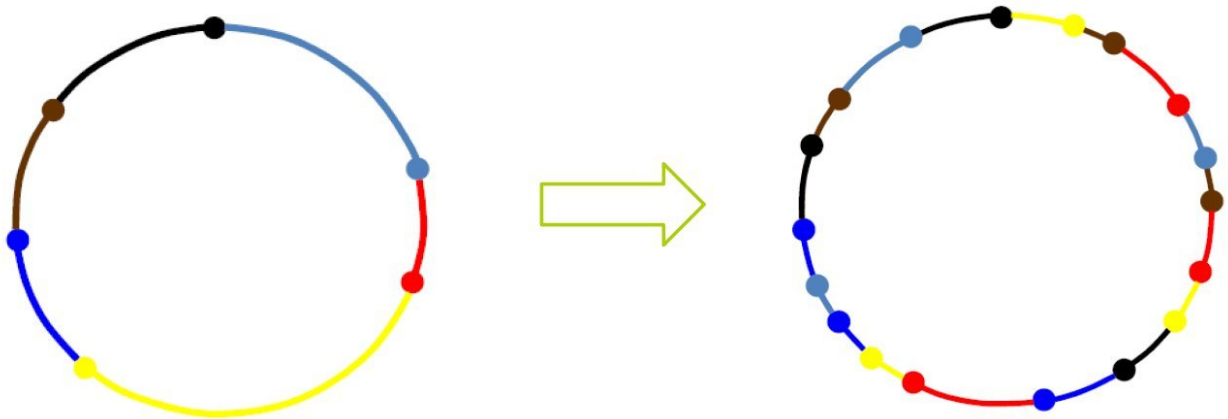
Load Imbalance (4/4)

- Consistent hashing may lead to imbalance.
- Heterogeneous nodes.



Load Balancing via Virtual Nodes

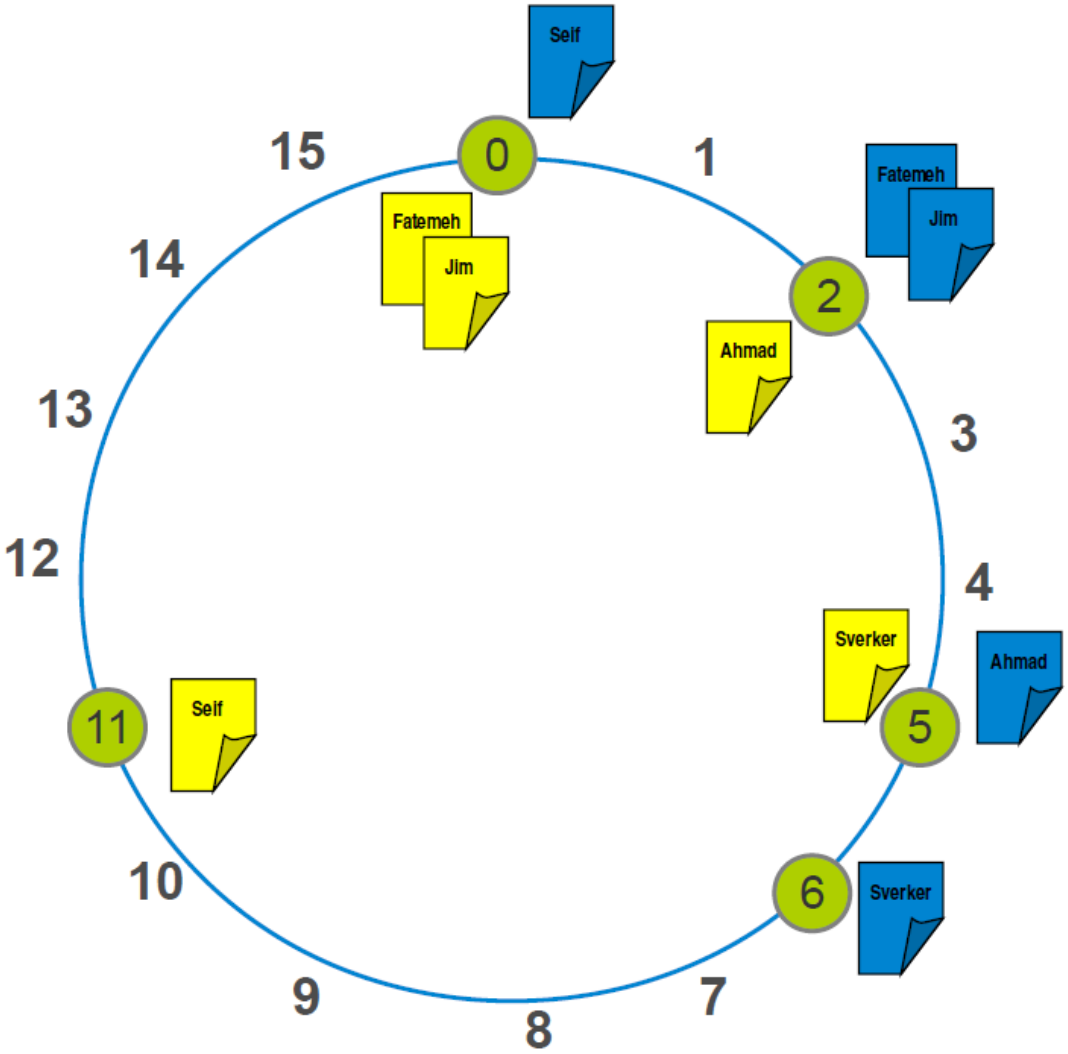
- Each physical node picks multiple random identifiers.
- Each identifier represents a virtual node.
- Each node runs multiple virtual nodes.



Replication

Replication

- To achieve high availability and durability, data should be replicates on multiple nodes.



Data Versioning

Data Versioning (1/3)

- Updates are propagated asynchronously.

Data Versioning (1/3)

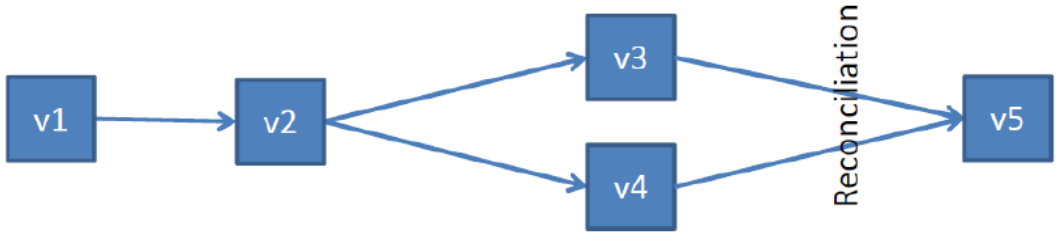
- Updates are propagated asynchronously.
- Each update/modification of an item results in a new and immutable version of the data.
  - Multiple versions of an object may exist.

Data Versioning (1/3)

- Updates are propagated asynchronously.
- Each update/modification of an item results in a new and immutable version of the data.
  - Multiple versions of an object may exist.
- Replicas eventually become consistent.

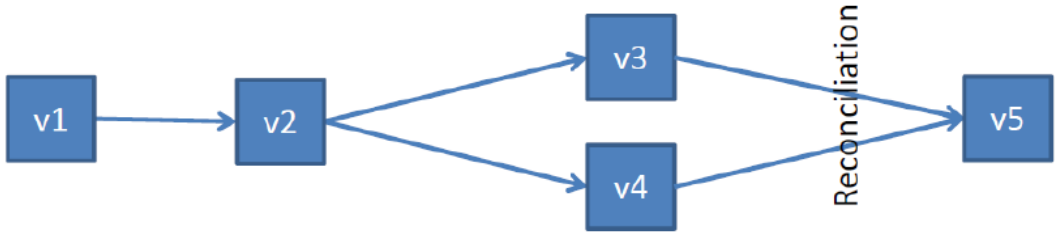
Data Versioning (2/3)

- Version branching can happen due to node/network failures.



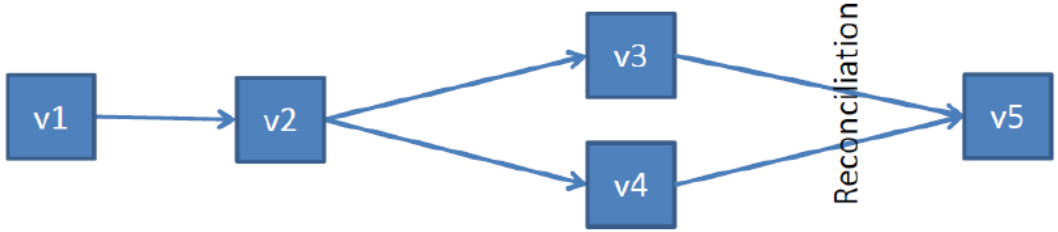
Data Versioning (2/3)

- Version branching can happen due to node/network failures.
- Use vector clocks for capturing causality, in the form of (node, counter)



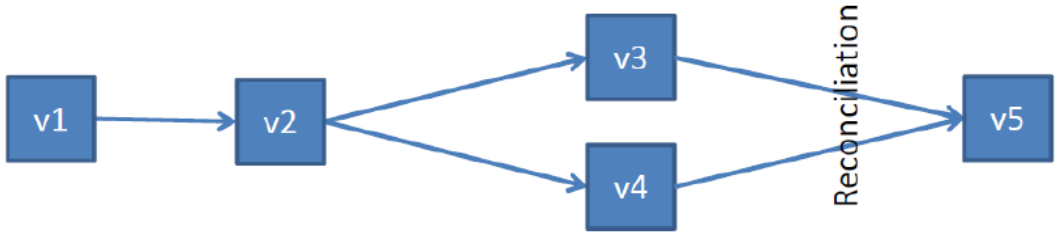
Data Versioning (2/3)

- Version branching can happen due to node/network failures.
- Use vector clocks for capturing causality, in the form of (node, counter)
  - If causal: older version can be forgotten



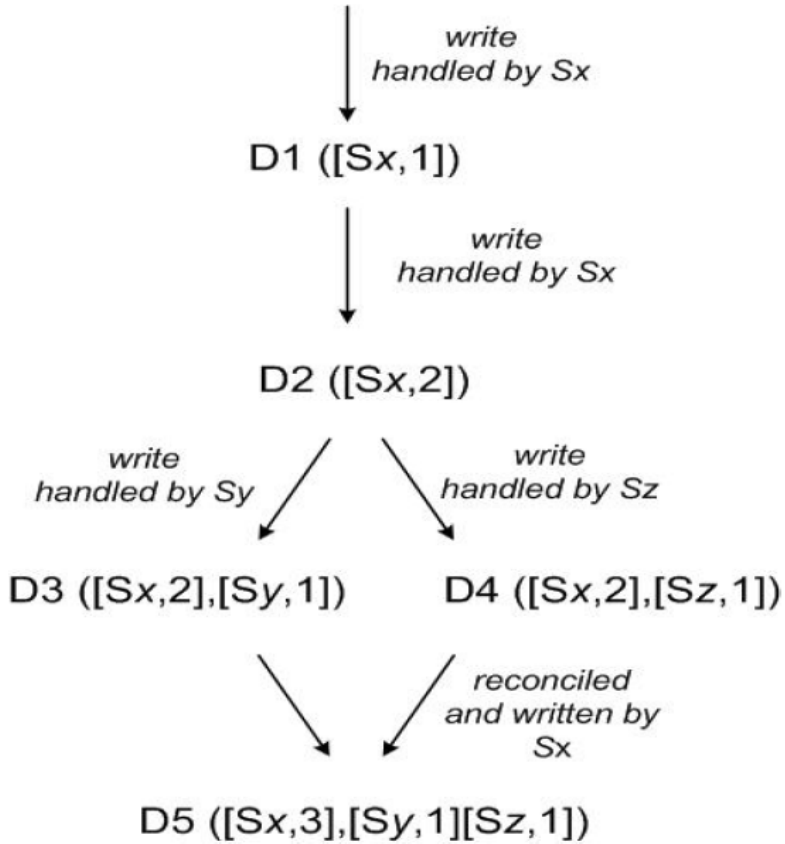
Data Versioning (2/3)

- Version branching can happen due to node/network failures.
- Use vector clocks for capturing causality, in the form of (node, counter)
  - If causal: older version can be forgotten
  - If concurrent: conflict exists, requiring reconciliation



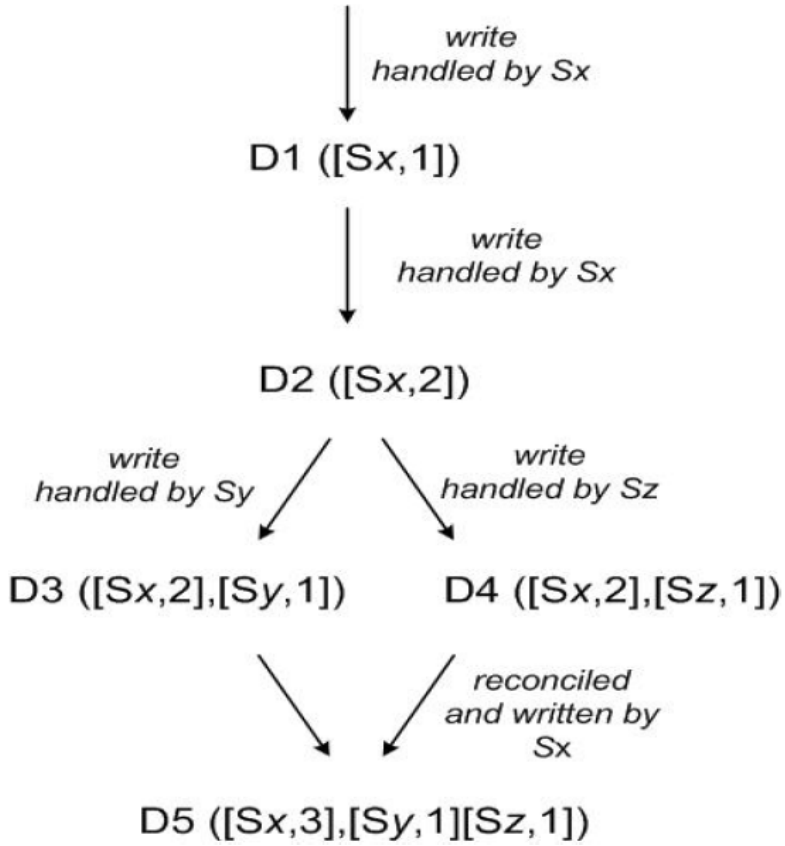
Data Versioning (3/3)

- Client C1 writes new object via Sx.



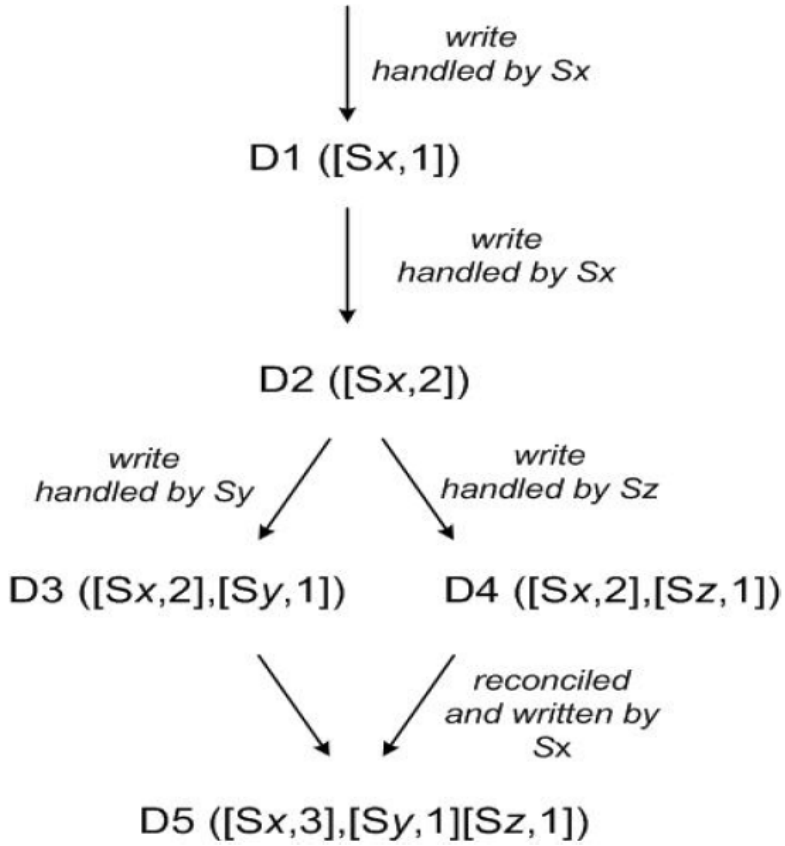
Data Versioning (3/3)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.



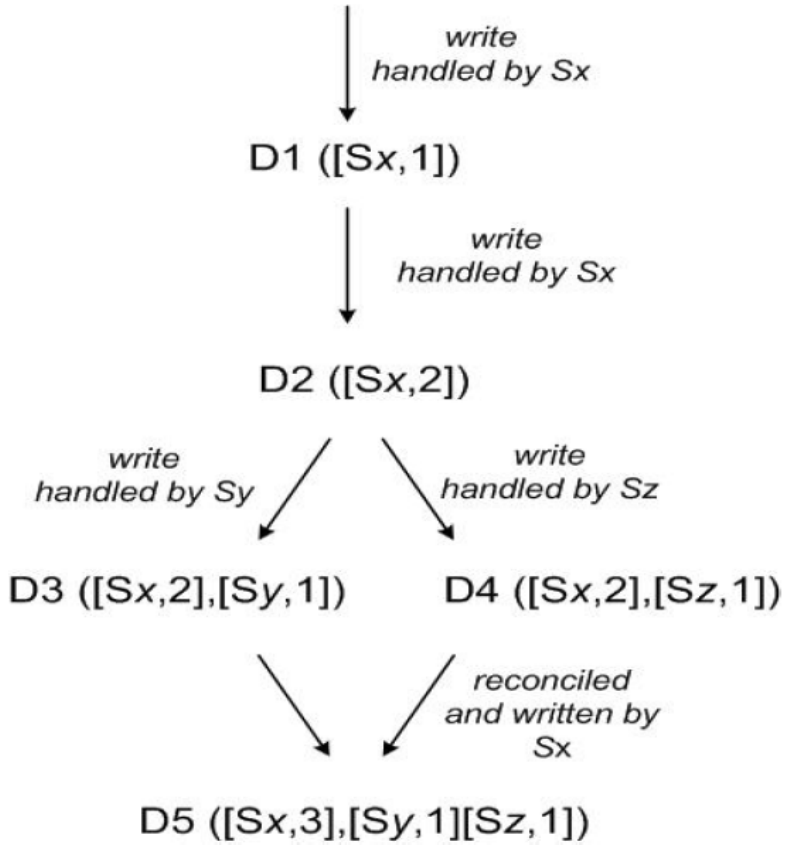
Data Versioning (3/3)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.



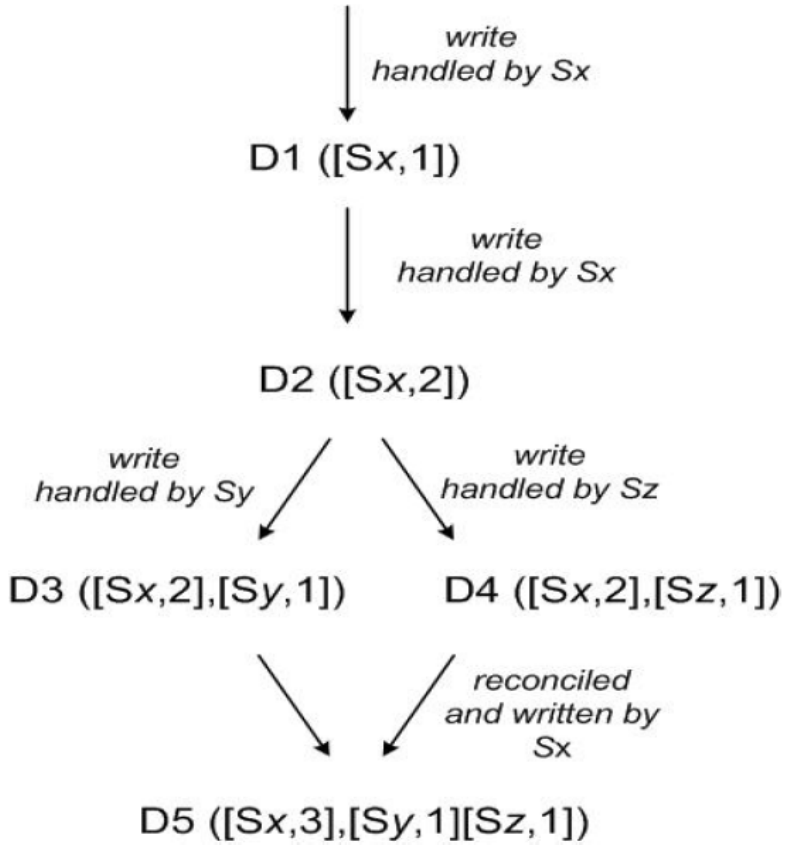
Data Versioning (3/3)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.



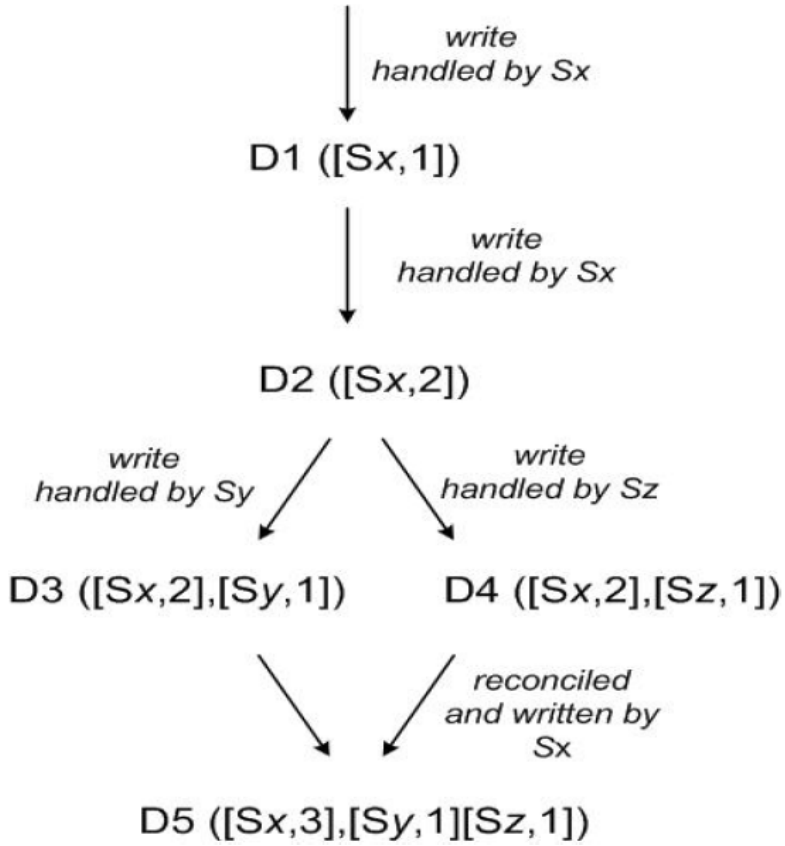
Data Versioning (3/3)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.
- C3 reads D3 and D4 via Sx.
  - The read context is a summary of the clocks of D3 and D4: [(Sx, 2), (Sy, 1), (Sz, 1)].



Data Versioning (3/3)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.
- C3 reads D3 and D4 via Sx.
  - The read context is a summary of the clocks of D3 and D4: [(Sx, 2), (Sy, 1), (Sz, 1)].
- Reconciliation



Dynamo API

Dynamo API

- `get(key)`
  - Return single object or list of objects with conflicting version and context.

Dynamo API

- `get(key)`
  - Return single object or list of objects with conflicting version and context.
- `put(key, context, object)`
  - Store object and context under key.
  - Context encodes system metadata, e.g., version number.

Execution of Operations

- Client can send the request:
  - To the node responsible for the data (coordinator): save on latency, code on client
  - To a generic load balancer: extra hope

put Operation

- Coordinator generates new vector clock and writes the new version locally.

put Operation

- Coordinator generates new vector clock and writes the new version locally.
- Send to N nodes.

put Operation

- Coordinator generates new vector clock and writes the new version locally.
- Send to N nodes.
- Wait for response from W nodes.

put Operation

- Coordinator generates new vector clock and writes the new version locally.
- Send to N nodes.
- Wait for response from W nodes.
- Using W=1
  - High availability for writes
  - Low durability

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.
- If multiple versions, return all versions that are causally unrelated.

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.
- If multiple versions, return all versions that are causally unrelated.
- Divergent versions are then reconciled.

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.
- If multiple versions, return all versions that are causally unrelated.
- Divergent versions are then reconciled.
- Reconciled version written back.

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.
- If multiple versions, return all versions that are causally unrelated.
- Divergent versions are then reconciled.
- Reconciled version written back.
- Using R=1
  - High performance read engine.

Membership Management

Membership Management

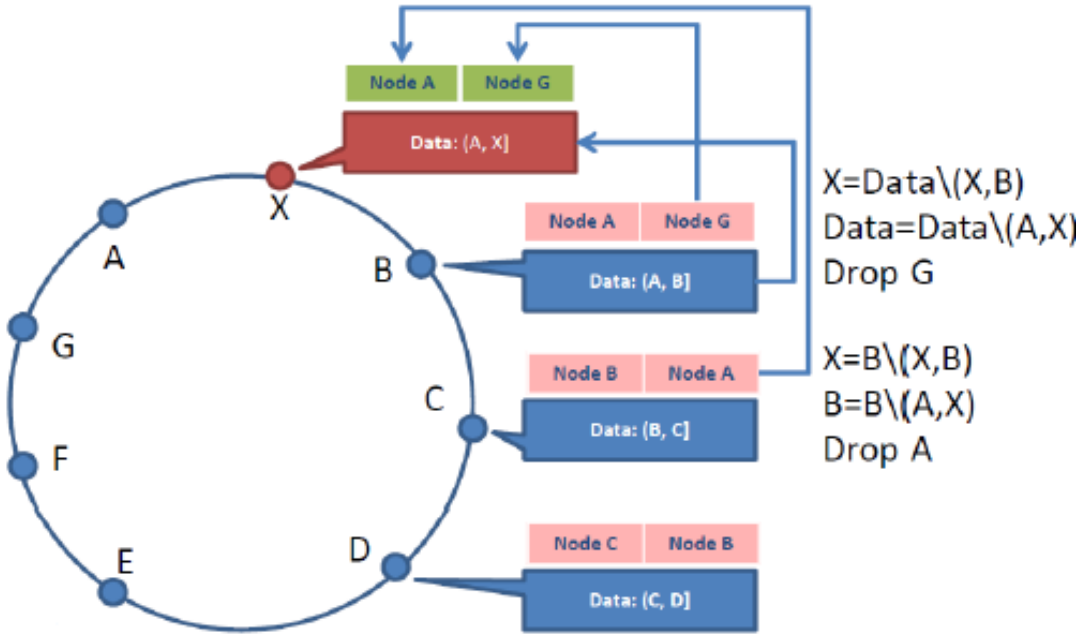
- Administrator explicitly adds and removes nodes.

Membership Management

- Administrator explicitly adds and removes nodes.
- Gossiping to propagate membership changes.
  - Eventually consistent view.
  - O(1) hop overlay.

Adding Nodes

- A new node X added to system.
  - X is assigned key ranges w.r.t. its virtual servers.
  - For each key range, it transfers the data items.



Removing Nodes

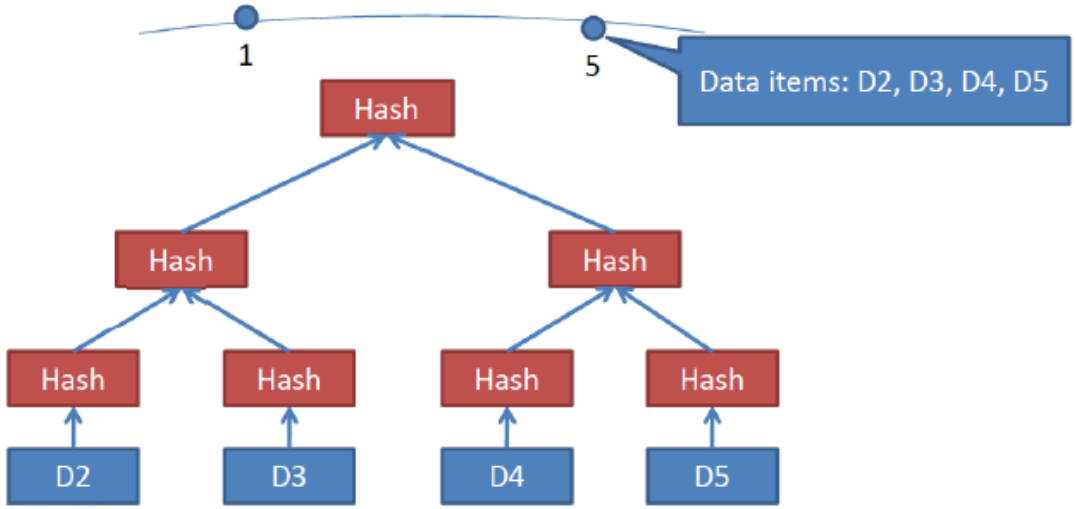
- Reallocation of keys is a reverse process of adding nodes.

Failure Detection

- Passive failure detection.
  - Use pings only for detection from failed to alive.
- In the absence of client requests, node A doesn't need to know if node B is alive.

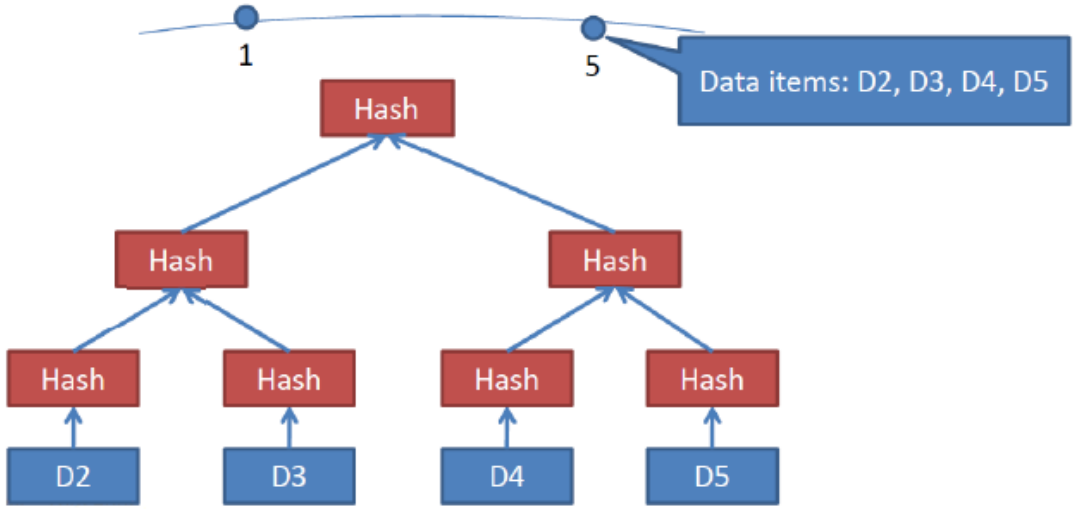
Handling Permanent Failure

- Anti-entropy for replica synchronization.



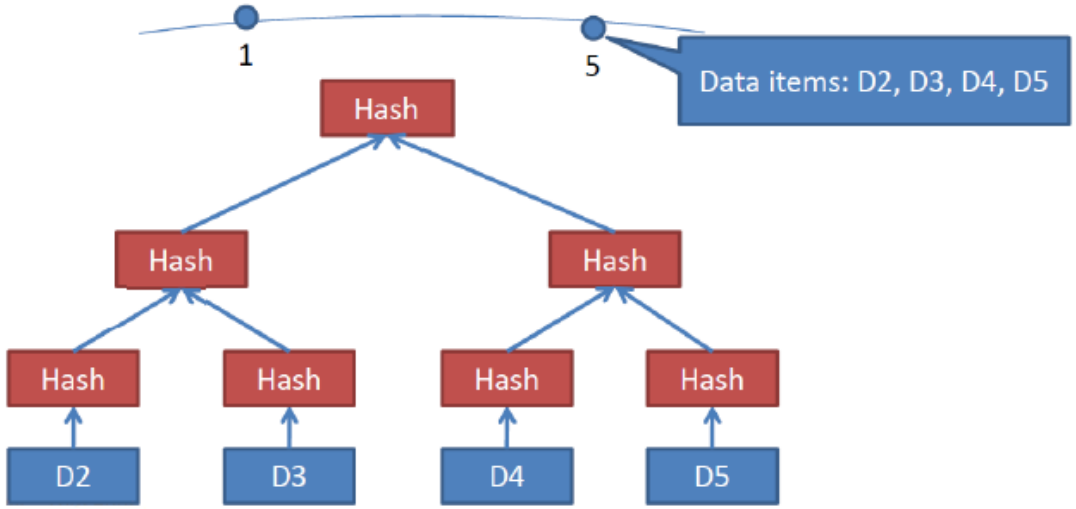
Handling Permanent Failure

- Anti-entropy for replica synchronization.
- Use Merkle trees for fast inconsistency detection and minimum transfer of data.



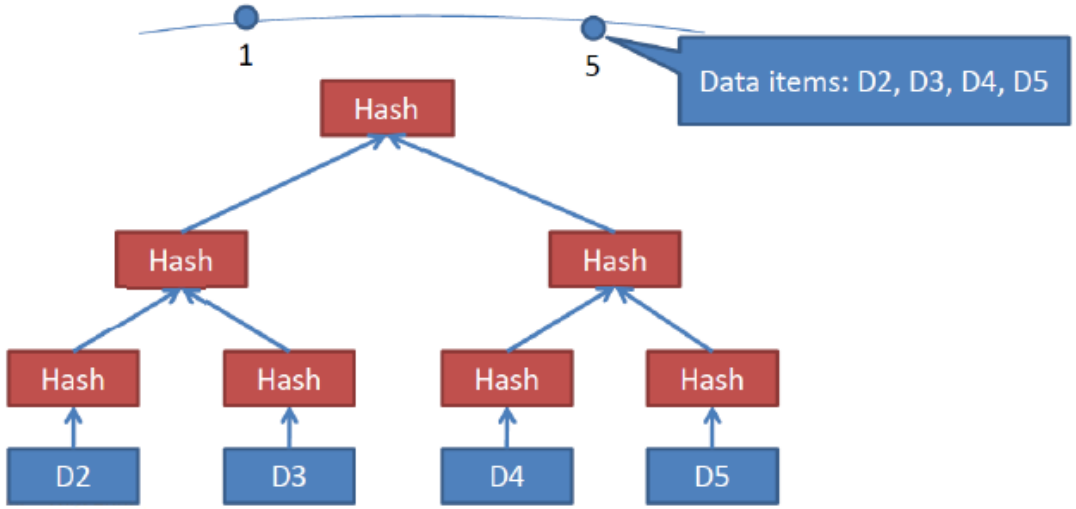
Handling Permanent Failure

- Anti-entropy for replica synchronization.
- Use Merkle trees for fast inconsistency detection and minimum transfer of data.
  - Nodes maintain Merkle tree of each key range.



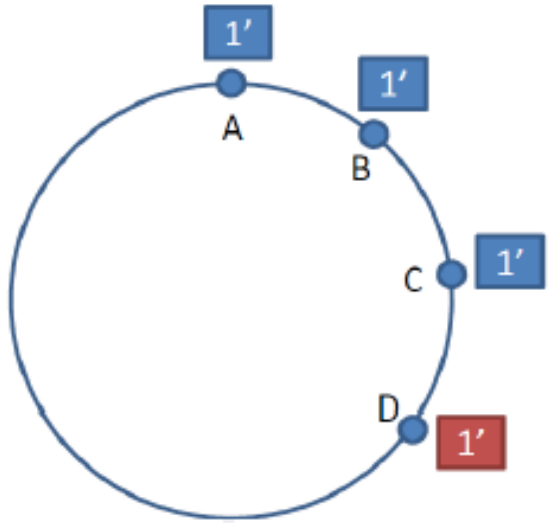
Handling Permanent Failure

- Anti-entropy for replica synchronization.
- Use Merkle trees for fast inconsistency detection and minimum transfer of data.
  - Nodes maintain Merkle tree of each key range.
  - Exchange root of Merkle tree to check if the key ranges are updated.



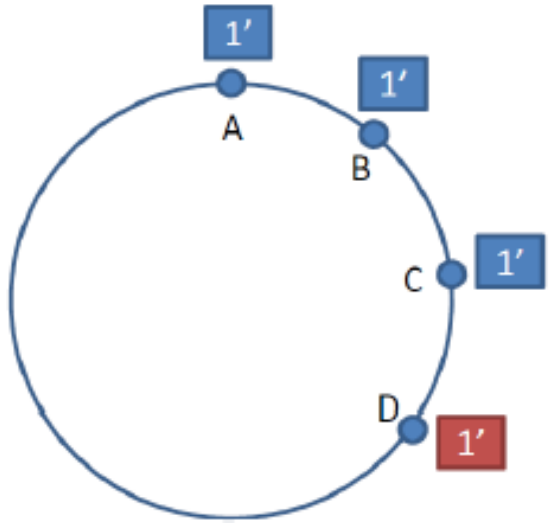
Failure Transient Detection

- Due to partitions, quorums might not exist.
  - Sloppy quorum.
  - Create transient replicas: N healthy nodes from the preference list.
  - Reconcile after partition heals.



Failure Transient Detection

- Due to partitions, quorums might not exist.
  - Sloppy quorum.
  - Create transient replicas: N healthy nodes from the preference list.
  - Reconcile after partition heals.
- Say A is unreachable.
- `put` will use D.
- Later, D detects A is alive.
  - Sends the replica to A
  - Removes the replica.



Summary

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- Sharding and consistent hashing
- ACID vs. BASE
- CAP (Consistency vs. Availability)

Summary

- Dynamo: key/value storage: put and get
- Data partitioning: consistent hashing
- Load balancing: virtual server
- Replication: several nodes, preference list
- Data versioning: vector clock, resolve conflict at read time by the application
- Membership management: join/leave by admin, gossip-based to update the nodes' views, ping to detect failure
- Handling transient failure: sloppy quorum
- Handling permanent failure: Merkle tree

Questions?