Πανεπιστήμιο Πατρών

# Προχωρημένα Θέματα σε Κατανεμημένα Συστήματα

Cassandra

# Cassandra

## A Decentralized Structured Storage System

# Overview

- Row store
- Decentralized
- Data Model: *similar* to relational
  - …but not entirely!
- High resilience to failures
- High scalability, w.r.t. the amount of data

# Properties

- Symmetric
  - No single point of failure
  - Linearly scalable
  - Ease of administration
- Flexible  partitioning, replica placement
- Automated provisioning
- High availability (eventual consistency)

# Influenced by

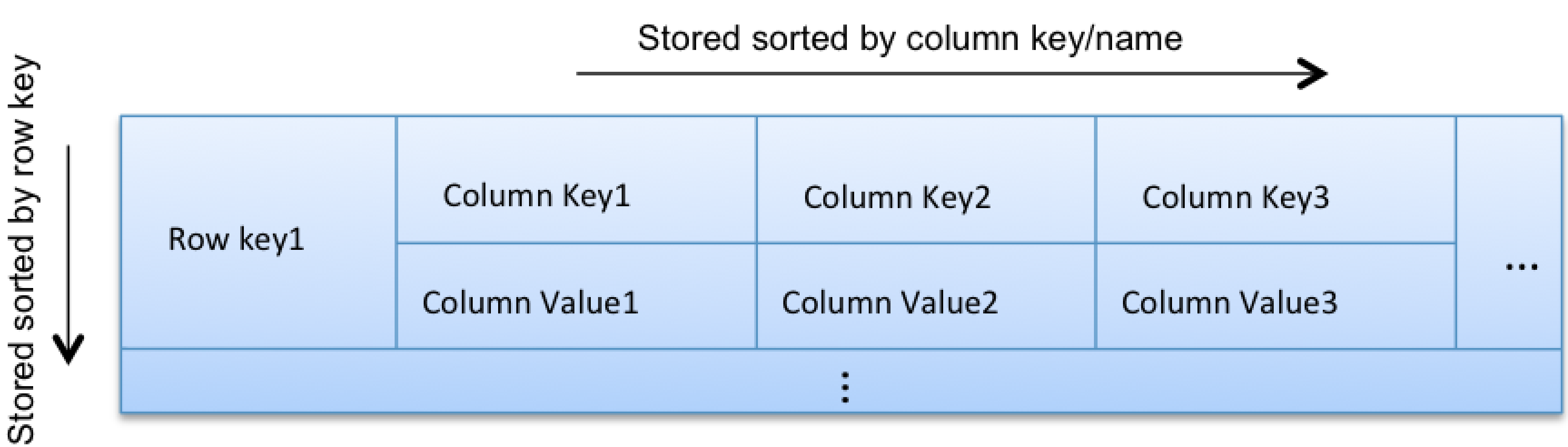
- BigTable (Google)
  - Strong consistency
  - Sparse map data model
  - GFS [Chubby et al.]
- Dynamo (Amazon)
  - O(1) distributed hash table (DHT)
  - BASE (aka eventual consistency)
  - Client tunable consistency/availability

# Dynamo

- Consistent Hashing
  - Routing, Load balancing, Replica placement
- Vector Clocks
  - Concurrent updates
- Gossip Protocol
- Hinted Handoffs
  - Failure detection/recovery

# Data Model

- Data
  - Rows indexed by row keys
  - Each row has any number of columns (key/value pairs)
  - Set of columns of a row: **column family** → Dynamic, Sparse
- Sorting
  - Columns are sorted by key
  - Rows *can* be sorted by key, but usually *are not*!
- Essentially, this schema is equivalent to:
  - **Map< RowKey , SortedMap<ColumnKey, ColumnValue> >**



# Data Model

- A column value may be composite, containing a set of columns (key/value pairs)
  - These are called **Super Columns**

- Essentially, this schema is equivalent to:
  - **Map< RowKey , SortedMap<SuperColumnKey, SortedMap<ColumnKey, ColumnValue>> >**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Super Column key1 | | | Super Column key2 | | | |
| Row key1 | Subcolumn Key1 | Subcolumn Key2 | … | Subcolumn Key3 | Subcolumn Key4 | … | … |
| | Column Value1 | Column Value2 | | Column Value3 | Column Value4 | | |
| ⋮ | | | | | | | |

# Analogy to the Relational Model (E-R)

| Relational Model | Cassandra Model |
|---|---|
| Database | Keyspace |
| Table | Column Family (CF) |
| Primary key | Row key |
| Column name | Column name/key |
| Column value | Column value |

# Basic API

- Simple API, very similar to DHTs:
  - insert (keyspace, key, rowMutation)
  - get (keyspace, key, columnName)
  - delete (keyspace, key, columnName)
- Requests may be sent to *any* node
- It collects the results and sends them to the client

# Additional API

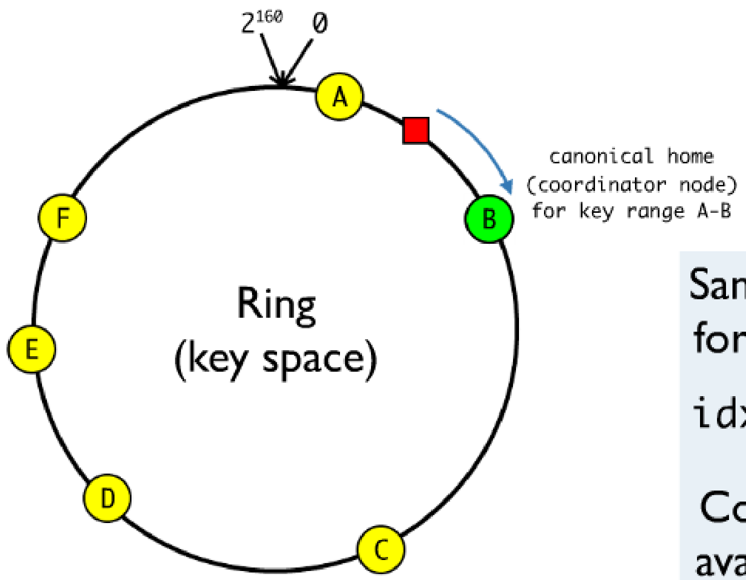
Cassandra has introduced the additional following API calls

- Range queries – for both keys and columns
  - list<KeySlice> get_range_slices(column_parent, predicate, range, consistency_level)
- Multiget – collect data from multiple rows, not necessarily contiguous
- Indexing and select
  - list<KeySlice> get_indexed_slices(column_parent, index_clause, predicate, consistency_level)

# Data Placement

# Consistent Hashing

- Data is partitioned across nodes, based on **consistent hashing**
  - Rings a bell?! ☺
- Hash function applied on each row's **partition key**
  - Typically, that' s the primary key
  - But can be defined otherwise
- Each row is, then, mapped to its (hashed) partition key's successor node




Same hash function for data and nodes

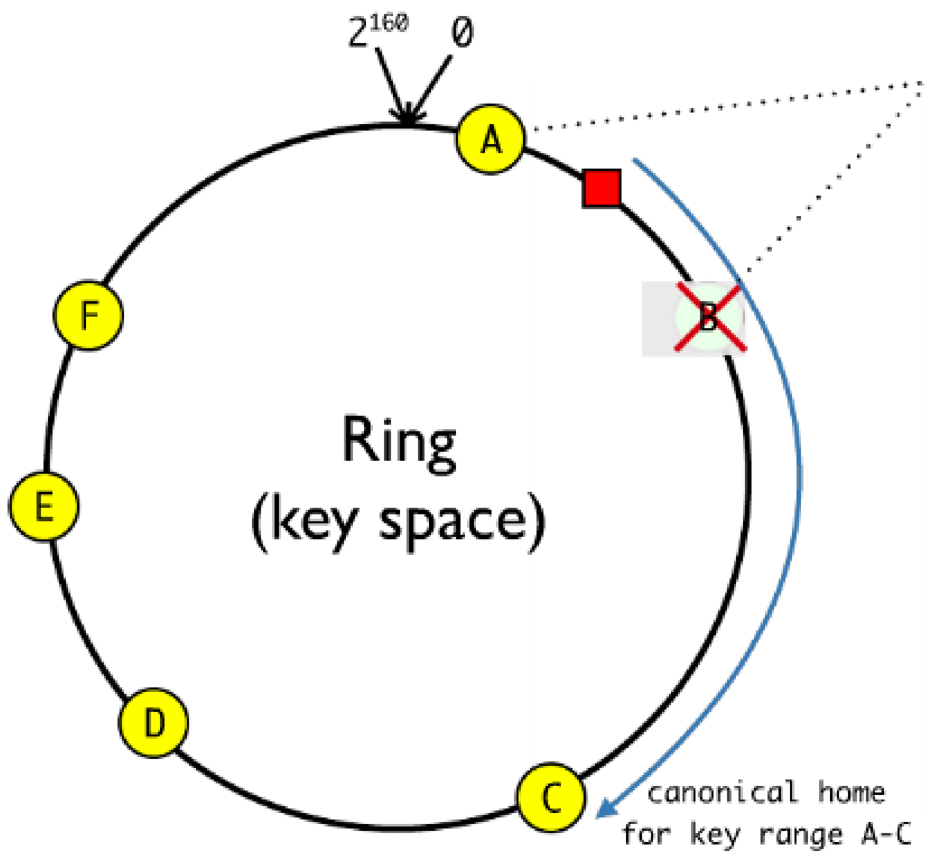
idx = hash(key)

Coordinator: next available clockwise node

# Partitioning

- Two hash function types
  - Random Partitioner
  - Order-Preserving Partitioner
- **RandomPartitioner** (**hash value** used as id)
  - Easy load balancing
  - Easy addition / removal of nodes
- **OrderPreservingPartitioner** (**string** used as id)
  - Range queries
  - Difficult load balancing
  - Difficult addition / removal of nodes

# Failures: Node B crashes



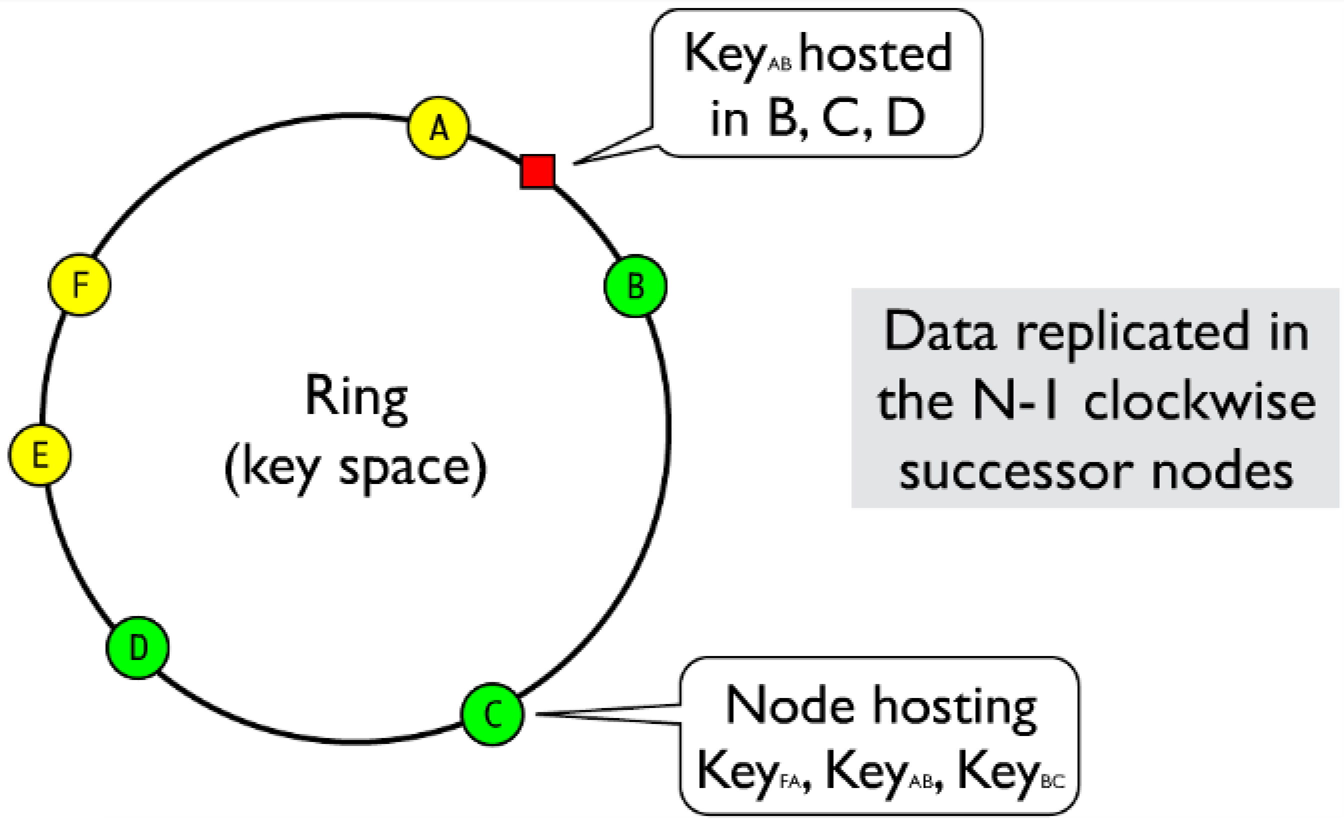

Same hash function for data and nodes

idx = hash(key)

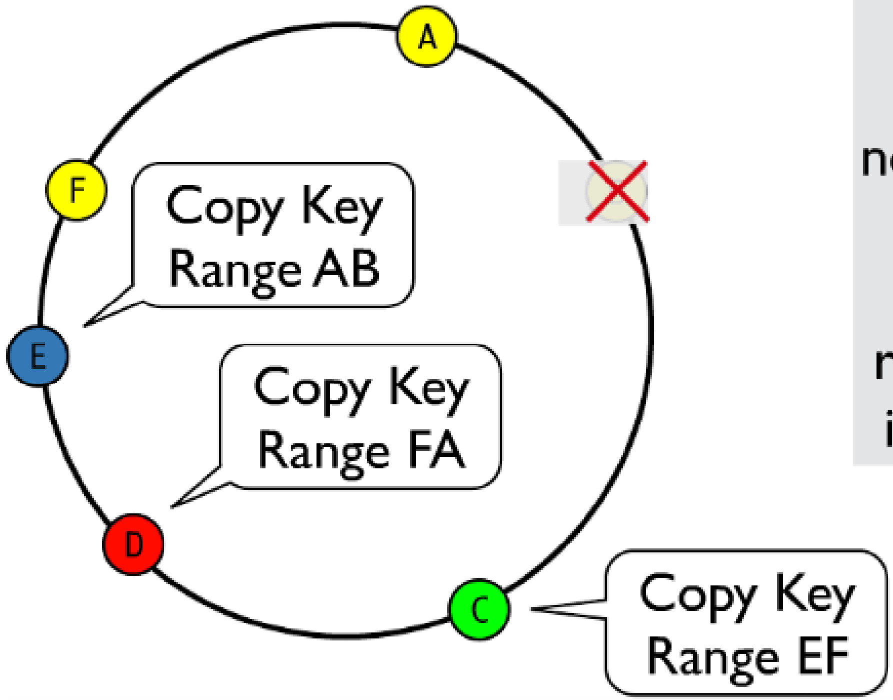
Coordinator: next available clockwise node

# Replication

- **N replicas** per row



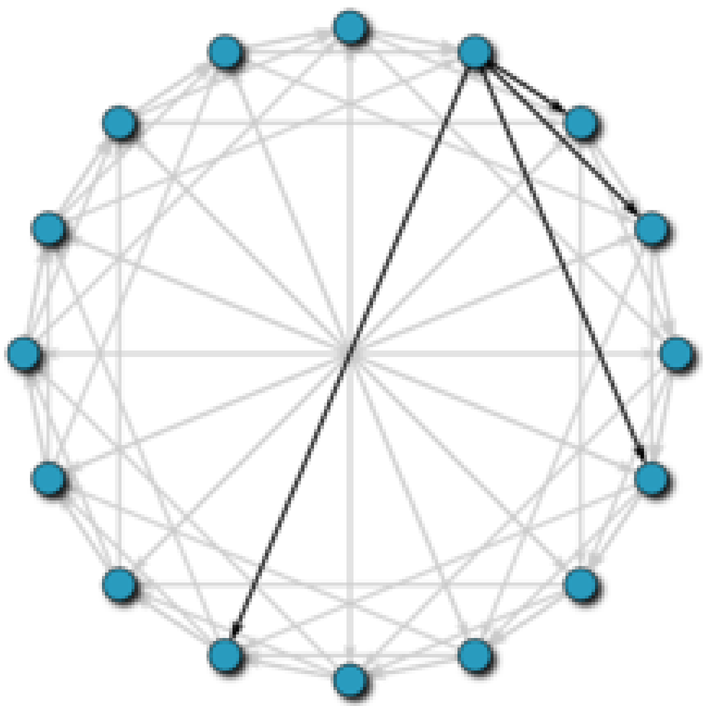
# Node B crashes with replication factor 3



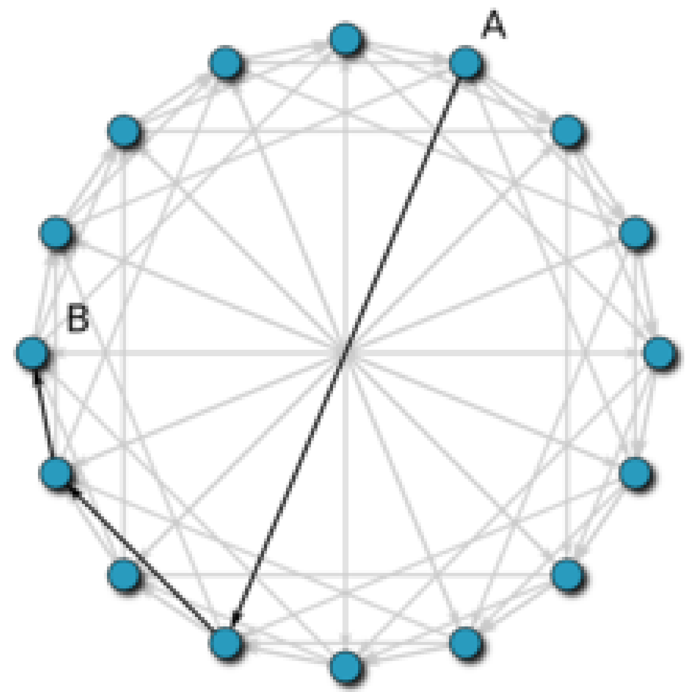

Key membership and replicas are updated when a node joins or leaves the network.
The number of replicas for all data is kept consistent.

# Message Routing

- Distributed Hash Table (Peer to Peer)

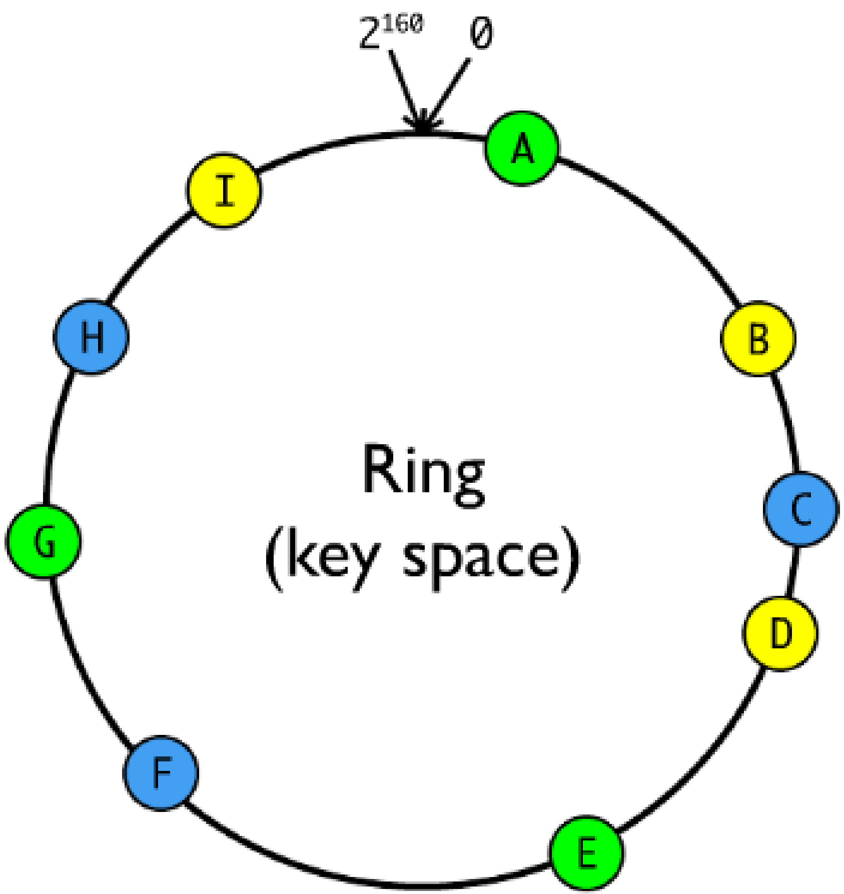


Routing tables




Getting to B in 3 hops

# Load Balancing? Virtual Nodes [1/2]



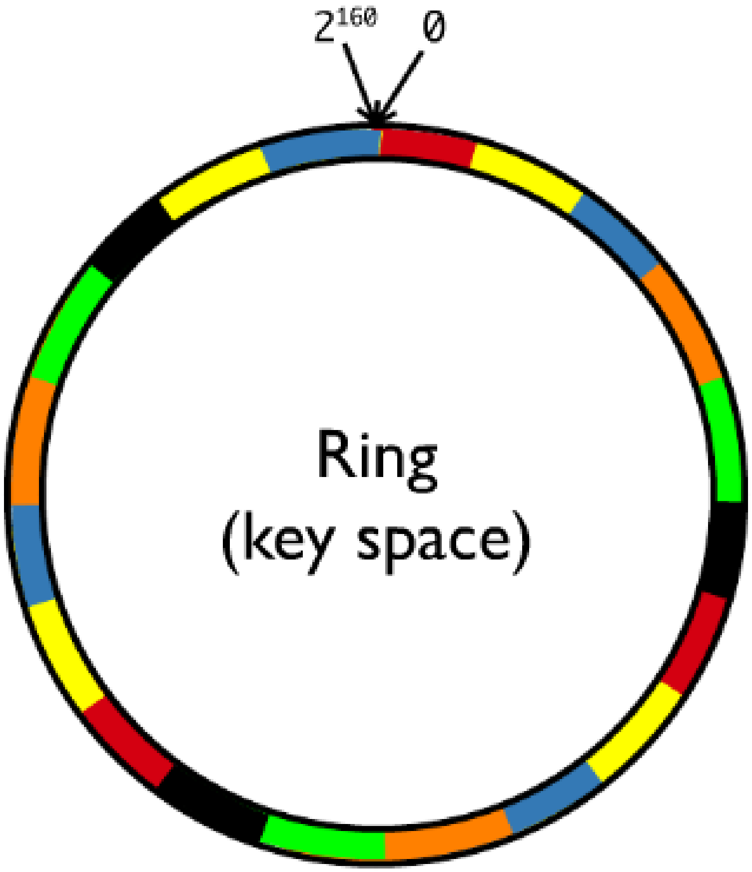

**Different Strategies**

*Virtual Nodes*

Random tokens per each physical node, partition by token value

- Node 1: tokens A, E, G
- Node 2: tokens C, F, H
- Node 3: tokens B, D, I

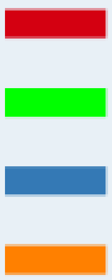
# Load Balancing? Virtual Nodes [2/2]




**Different Strategies**

*Virtual Nodes*

**Q** equal-sized partitions, ***S*** nodes, ***Q/S*** tokens per node (with **Q** >> **S**)



# Replica Placement Strategies

- Rack unaware
  - Simply choose N-1 successors
- Rack/DC aware:
  - Place one replica on different Data Center
  - Place another replica on different Rack of the same Data Center
- Totally configurable
  - Any replica placement strategy can be defined.

# Consistency level

## CL.Options

WRITE

| Level | Description |
|---|---|
| ZERO | Cross fingers |
| ANY | $1^{st}$ Response (including HH) |
| ONE | $1^{st}$ Response |
| QUORUM | N/2 + 1 replicas |
| ALL | All replicas |

READ

| Level | Description |
|---|---|
| | |
| | |
| ONE | $1^{st}$ Response |
| QUORUM | N/2 + 1 replicas |
| ALL | All replicas |

WEAK

STRONG

# Creating a Schema

# Relational Schema

User

| UserID | Name | Email |
|---|---|---|
| 123 | Jay | jp@ebay.com |
| 456 | John | jh@ebay.com |
| ⋮ | | |

User_Item_Like

| ID | UserID <fk> | ItemID <fk> | Timestamp |
|---|---|---|---|
| 1 | 123 | 111 | 120101010000 |
| 2 | 123 | 222 | 120101020000 |
| 3 | 456 | 111 | 120101030000 |
| ⋮ | | | |

Item

| ItemID | Title | Desc |
|---|---|---|
| 111 | iphone | It's a phone |
| 222 | ipad | It's a tablet |
| ⋮ | | |

# Exact replica of relational model

User

| | | |
|---|---|---|
| 123 | Name | Email |
| | Jay | jp@ebay.com |
| ⋮ | | |

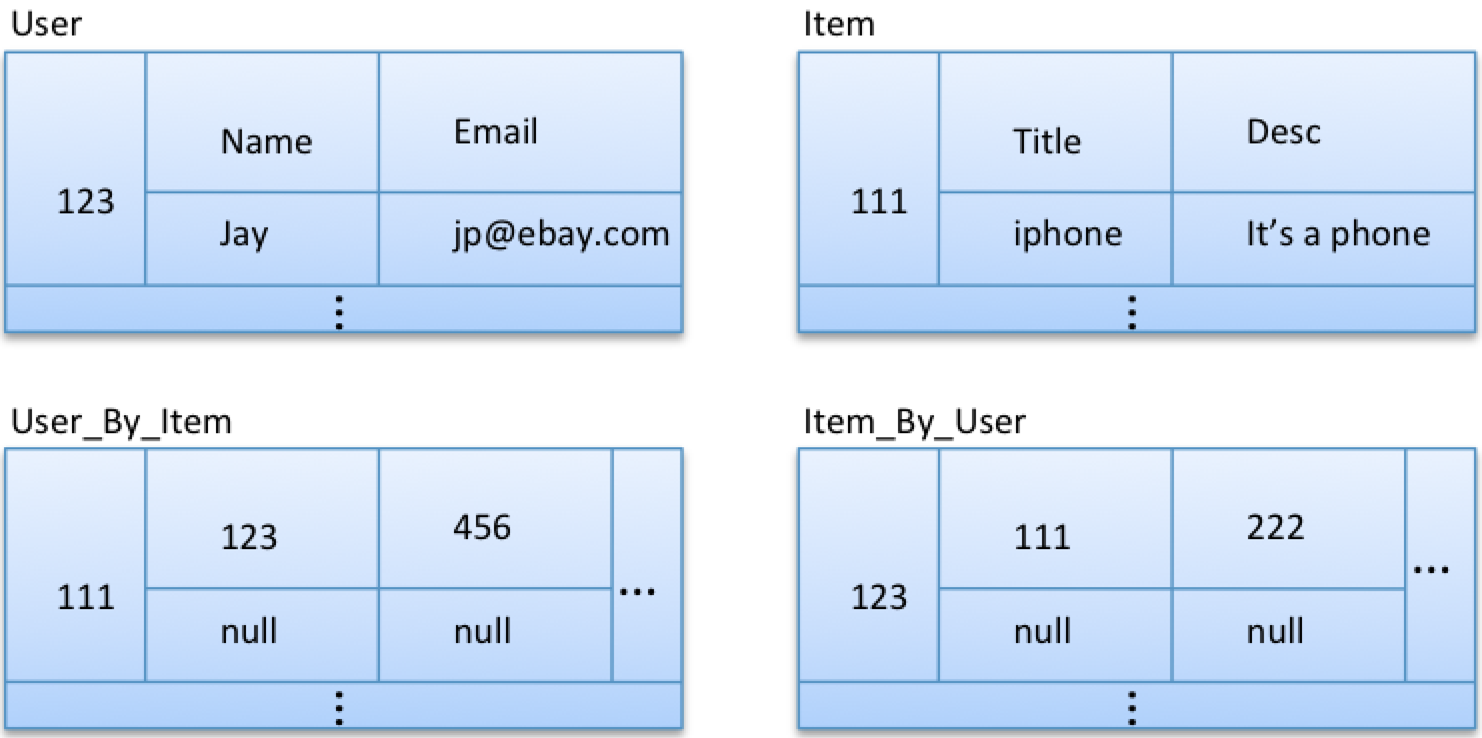
Item

| | | |
|---|---|---|
| 111 | Title | Desc |
| | iphone | It's a phone |
| ⋮ | | |

User_Item_Like

| | | |
|---|---|---|
| 1 | UserID | ItemID |
| | 123 | 111 |
| ⋮ | | |

# Normalized entities w/custom indexes



# More denormalization

- Normalized entities with de-normalization into custom indexes

User

| | | |
|---|---|---|
| 123 | Name | Email |
| | Jay | jp@ebay.com |
| ⋮ | | |

Item

| | | |
|---|---|---|
| 111 | Title | Desc |
| | iphone | It’s a phone |
| ⋮ | | |

User_By_Item

| | | | |
|---|---|---|---|
| 111 | 123 | 456 | … |
| | Jay | John | |
| ⋮ | | | |

Item_By_User

| | | | |
|---|---|---|---|
| 123 | 111 | 222 | … |
| | iphone | ipad | |
| ⋮ | | | |

# Partially de-normalized entities

User

| | UserInfo | | Likes | | |
|---|---|---|---|---|---|
| 123 | Name | Email | 111 | 222 | ... |
| | Jay | jp@ebay.com | iphone | ipad | |
| ⋮ | | | | | |

Item

| | ItemInfo | | LikedBy | | |
|---|---|---|---|---|---|
| 111 | Title | Desc | 123 | 4556 | ... |
| | iphone | It's a phone | Jay | John | |
| ⋮ | | | | | |

# Easy "get N most recent" queries

User

| | | |
|---|---|---|
| 123 | Name | Email |
| | Jay | jp@ebay.com |
| ⋮ | | |

Item

| | | |
|---|---|---|
| 111 | Title | Desc |
| | iphone | It's a phone |
| ⋮ | | |

<timeuuid|userid>

User_By_Item

| | | | |
|---|---|---|---|
| 111 | 120101010000 \| 123 | 120101030000 \| 456 | … |
| | Jay | John | |
| ⋮ | | | |

Item_By_User

| | | | |
|---|---|---|---|
| 123 | 120101010000 \| 111 | 120101020000 \| 222 | … |
| | iphone | ipad | |
| ⋮ | | | |

# Extra material

- Lamport, Leslie (2001). Paxos Made Simple ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001) 51-58
  - http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf
  - http://en.wikipedia.org/wiki/Paxos_%28computer_science%29

- Lakshman, A. and Malik, P. Cassandra - A Decentralized Structured Storage System. in ACM SIGOPS Operating Systems Review 2010
  - http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf
  - http://en.wikipedia.org/wiki/Apache_Cassandra

- Giuseppe de Candia et al. Dynamo: Amazon's highly available key-value store. In SIGOPS, pp 205–220. ACM, 2007.
  - http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
  - http://en.wikipedia.org/wiki/Dynamo_(storage_system)

- Distributed Hash Tables και Consistent hashing
  - Balakrishnan, H. and Kaashoek, M.F. and Karger, D. and Morris, R. and Stoica, I. Looking up data in P2P systems in Communications of the ACM, 2003
  - http://www.nms.lcs.mit.edu/papers/p43-balakrishnan.pdf
  - http://en.wikipedia.org/wiki/Consistent_hashing