Week 2: NoSQL Overview

DIANE WOODBRIDGE, PH.D

HW1

Due by November 3rd (Midnight).

Why are we doing this?!

If you are collaborating,

- List your collators and contributions of each member in Collaborators.txt.
- You can work up to 3 people together.
- At least change the variable names. (This collaboration doesn’t mean it is a group project. Understand and write your own code.)

HW1

- Collaborators.txt
- hw1_main.py
- json_key_value.py
- msan697_hw1.pdf
- postgres_function.py
- Q5_Answer.txt
- user_definition.py
- walmart_search_san_francisco.json

Objectives

Understand characteristics and needs of different NoSQL databases.

Cover important concepts of distribution models.

Cover topics of NoSQL/distributed database system interview questions.

NoSQL Interview Questions

What is NoSQL?

Relational Database vs. NoSQL

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

CAP Theorem

Eventual Consistency

NoSQL

Why NoSQL?

- Impedence Mismatch
  - Relation model ≠ In-memory data structure (object)
  - For application Development Productivity,
    - Better mapping with in-memory data structures for the application.
- Large volumes of data (2000s)
  - Scaling up vs. Scaling out?
    - Run large data on clusters of many smaller and cheaper machines.
    - Cheaper and reliable.
- Example of non-relational database.
  - Google BigTable and Amaon Dynamo.

NoSQL

Generally,

- Take schemaless data.
- Non-relational.
- Open-source.
- Trade off traditional consistency for other properties.
- Run on clusters.

SQL and NoSQL

SQL	VS	NoSQL
Pros Persistent data storage. Concurrency. Standard Model. **Cons** Impedence mismatch. Hard to scale. Fixed schema. **Example** MySQL, PostgreSQL, SQL Server, Oracle		**Pros** Mostly open-source. Schemaless. Good for non-relational data. Scalable. Runs well on distributed systems. **Cons** Installation, toolsets still maturing. **Example** Redis, MongoDB, Cassandra, OrientDB

Aggregate-oriented NoSQL Databases

Aggregate: Collection of related objects treated as a unit.

- For analyzing data, you might want to place some data together as an aggregate.
- On a cluster, an aggregate is stored together on a node.
- Aggregate-oriented databases use aggregates indexed by key for data lookup.
- A single aggregate is a unit of atomic updates.
- Aggregate-oriented databases don't have ACID transactions that span multiple aggregates.
- RDBMS/Graph DB are aggregate-ignorant.


```
{
  "_links":{
    "self":{
      "href":"/member/109087/cart"
    },
    "http://example.com/rels/payment":{
      "href":"/payment"
    }
  },
  "_embedded":{
    "http://example.com/rels/cart-item":[
      {
        "_links":{
          "self":{
            "href":"/member/109087/cart/14418796"
          }
        },
        "id":"14418796",
        "quantity":1,
        "expire_time":"2009-09-11T08:00:00-07:00",
        "_embedded":{
          "http://example.com/rels/sku":[
            {
              "_links":{
                "self":{
                  "href":"/skus/654654"
                },
                "http://example.com/images/cart/item":"http://example.com/product/6895/thumbnail.jp
              },
              "color":"Black",
              "size":"S",
              "returnable":false,
              "price":49
            }
          ]
        }
      }
    ]
  }
}
```


https://gist.github.com/hjr3/2289546

Aggregate-oriented NoSQL Databases

Pros

- Provides clearer semantics to consider by focusing on the aggregate unit used by applications.
- Better design choice for running on a cluster.

Cons

- Drawing boundaries of an aggregate is not easy.
- When a goal of data management/analysis is not clear, aggregate models might not be the best choice.
- Doesn’t support ACID transactions.

Aggregate-oriented NoSQL Databases

Types

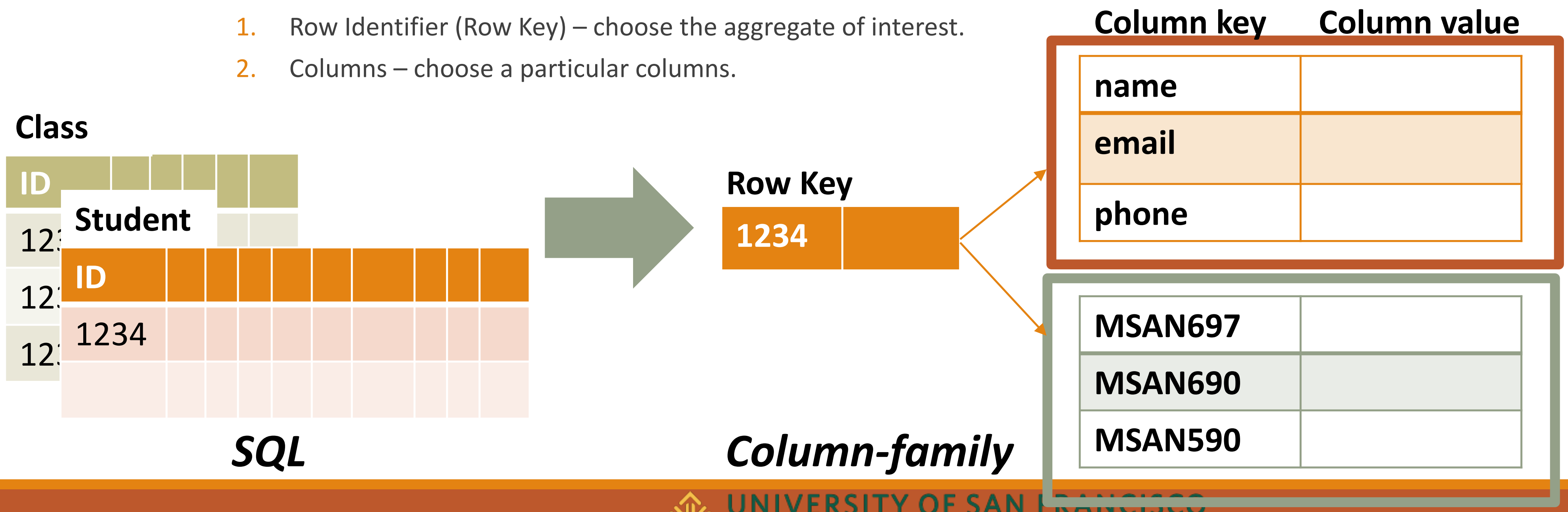
- Key-value and Document Database
  - Each aggregate has a key (ID).
  - Key-value database
    - We can store whatever we want in aggregates.
    - Key lookup for the entire aggregate.
  - Document database
    - It has allowable structures and types.
    - Access by key and also by the fields in the aggregates. (can retrieve fields in the value.)

➔Key-value and Document DB are similar, and their distinction is often blurry.
But with Document DB, you can submit a query based on the internal structure of the document.

Aggregate-oriented NoSQL Databases

Types

- Column Family
  - Optimize for cases when write is rare, but columns are read together in many rows together .
  - Organizes columns into column families (Unit of access).
  - Two level map structure.
    1. Row Identifier (Row Key) – choose the aggregate of interest.
    2. Columns – choose a particular columns.



Relationship-oriented NoSQL Database

Needs of relationship-oriented DB

- Relational database with complex schema
  - Hard to understand, query, generalize and integrate data.
- Aggregate-oriented NoSQL
  - Atomicity is only supported within a single aggregate.

Relationship-oriented NoSQL Database

Graph Database

- Nodes (Object) and edges (Relationships) representation.
- For data with complex relationships.
  - Focuses on graph traverse (more than insert.).
  - Cf) RDBMS : Many joins can cause poor performance.
- Running on a single server rather than distributed across clusters.

NoSQL Database Examples

Key-value

- Redis, Riak, Berkeley DB, etc.

Document

- MongoDB, CouchDB, OrientDB, RavenDB, etc.

Column Family

- Cassandra, Hbase, Amazon SimpleDB, etc.

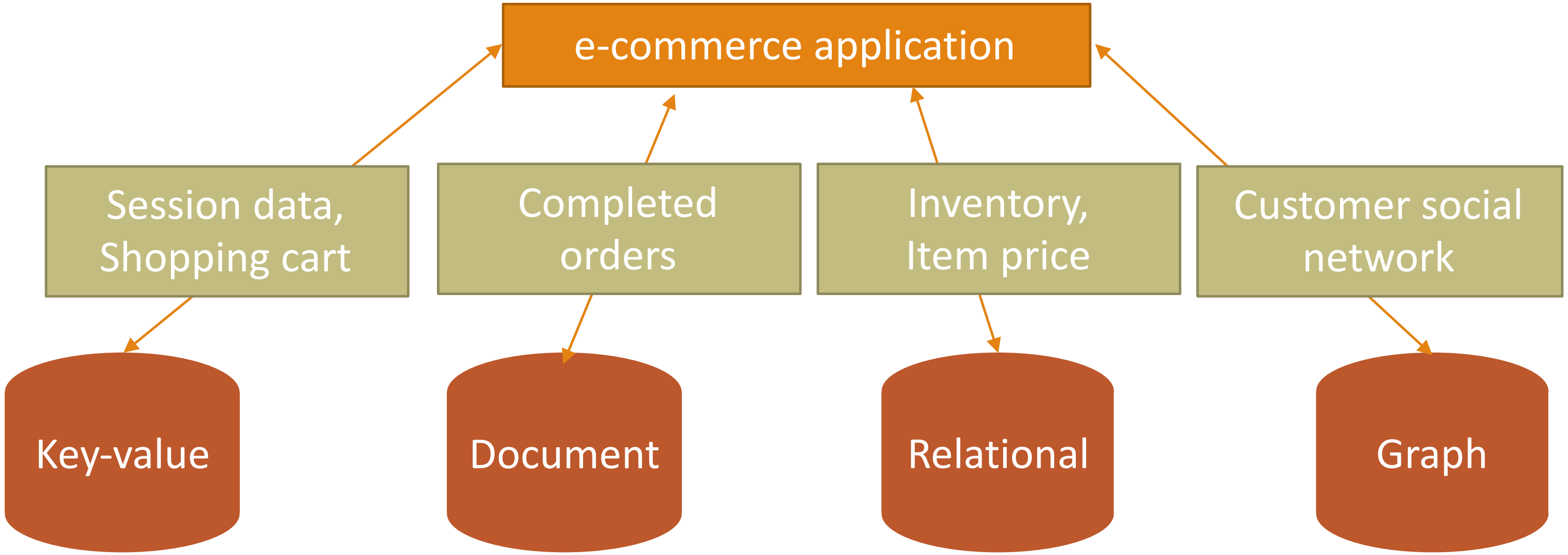
Graph

- OrientDB, Neo4J, FlockDB, etc.

Choice of DBMS

Polygot Persistence

- Using multiple data storage technologies, chosen based on the way data is being used by individual applications.
- NoSQL data stores do not replace relational databases.



Database Ranking (Sept 2016)

db-engines.com/en/ranking

- Graph DBMS
- Time Series DBMS
- RDF stores
- Search engines
- Object oriented DBMS
- Multivalue DBMS
- Native XML DBMS
- Wide column stores
- Content stores
- Event Stores
- Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

Read more about the method of calculating the scores.

trend chart

315 systems in ranking, September 2016

Rank Sep 2016	Rank Aug 2016	Rank Sep 2015	DBMS	Database Model	Score Sep 2016	Score Aug 2016	Score Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑5.	↑5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓4.	↓4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14
11.	11.	↑14.	Elasticsearch	Search engine	96.48	+3.99	+24.93
12.	12.	↑13.	Teradata	Relational DBMS	73.06	-0.57	-1.20
13.	13.	↓11.	SAP Adaptive Server	Relational DBMS	69.16	-1.88	-17.36
14.	14.	↓12.	Solr	Search engine	66.96	+1.19	-14.98
15.	15.	15.	HBase	Wide column store	57.81	+2.30	-1.22
16.	16.	↑17.	FileMaker	Relational DBMS	55.35	+0.34	+4.35
17.	17.	↑18.	Splunk	Search engine	51.29	+2.38	+9.06
18.	18.	↓16.	Hive	Relational DBMS	48.82	+1.01	-4.71
19.	19.	19.	SAP HANA	Relational DBMS	43.42	+0.68	+5.22
20.	20.	↑25.	MariaDB	Relational DBMS	38.53	+1.65	+14.31
21.	21.	21.	Neo4j	Graph DBMS	36.37	+0.80	+2.83
22.	↑24.	↑24.	Couchbase	Document store	28.54	+1.14	+2.28
23.	23.	↓22.	Memcached	Key-value store	28.43	+0.74	-3.99
24.	↓22.	↓20.	Informix	Relational DBMS	28.19	-0.86	-9.76
25.	25.	↑28.	Amazon DynamoDB	Document store	27.42	+0.82	+7.43
26.	26.	↓23.	CouchDB	Document store	21.48	+0.42	-5.12
27.	27.	↑30.	Vertica	Relational DBMS	21.06	+0.58	+3.31
28.	↑29.	↓27.	Netezza	Relational DBMS	19.81	+0.34	-1.24

http://db-engines.com/en/ranking/

Database Ranking (Sept 2016)

315 systems in ranking, September 2016

Rank			DBMS	Database Model	Score		
Sep 2016	Aug 2016	Sep 2015			Sep 2016	Aug 2016	Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑5.	↑5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓4.	↓4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14

Distribution Models

Single Server Distribution Model

- If we can get away without distribution, we should choose a single-server approach.
- Ex. Graph Database

Distribution Models

Aggregate

- Collection of related objects treated as a unit.
- Natural unit for distribution.

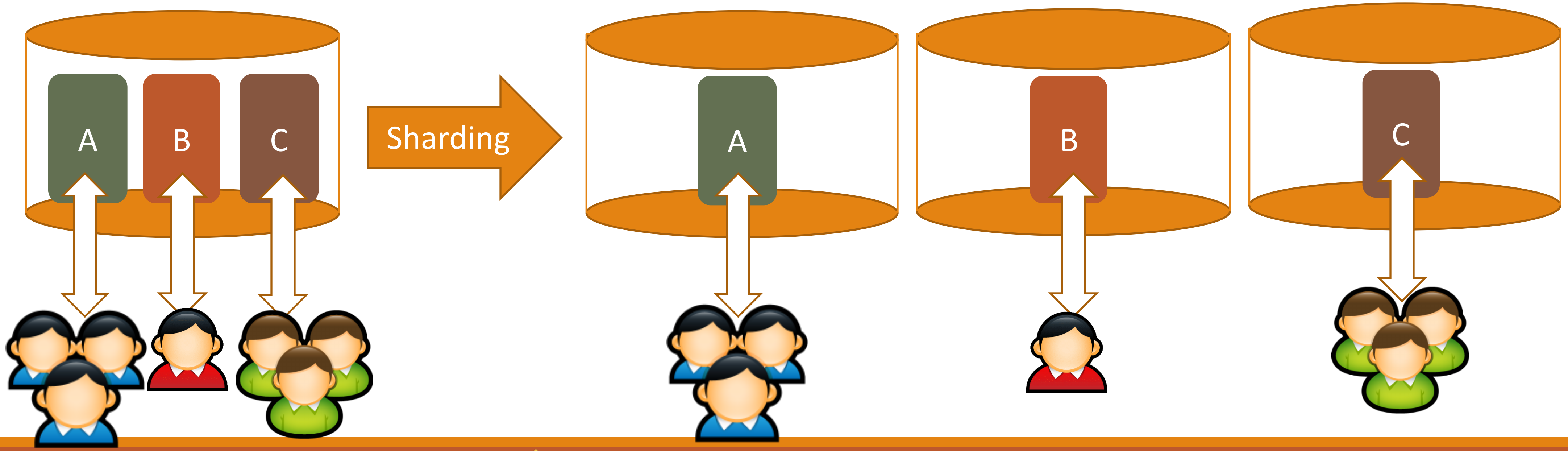
Two ways for data distribution

- Sharding : Place different data on different nodes.
- Replication : Copy the same data over multiple nodes.
  - Master-Slave Replication
  - Peer-to-Peer Replication

Distribution Models

Sharding

- Placing different parts of data into different servers, and each of them does its own reads and writes.
- Things to consider
  - Locate the data commonly accessed together on the same node (Aggregate or Data accessed sequentially together.).
  - Physical location.
  - Keep the load even.



Distribution Models

Sharding

- Options : auto-sharding in the DB vs. writing in your application code.
- Pros : Improves read and writes.
- Cons : Low resilience.

Distribution Models

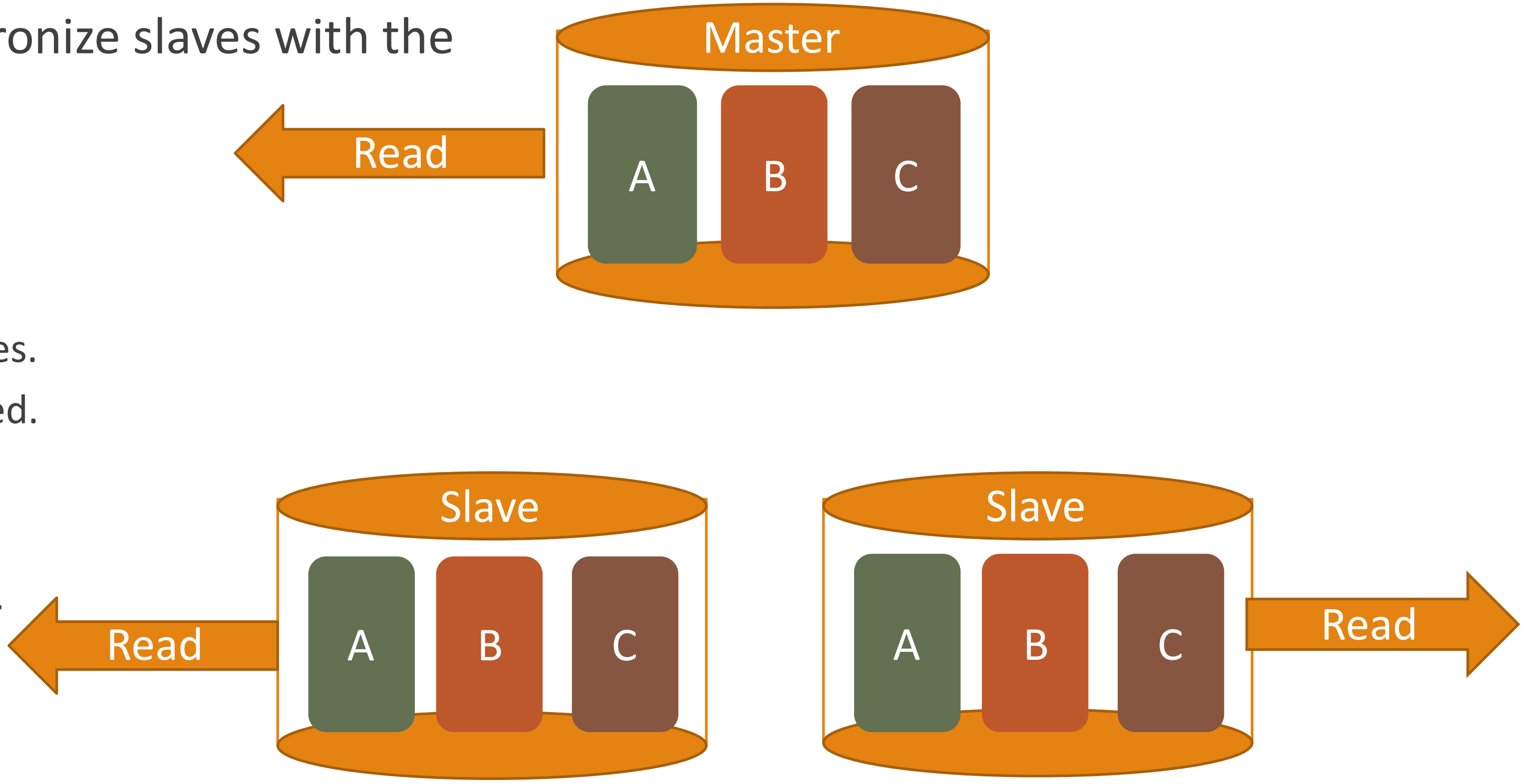
Replications: Copy the same data over multiple nodes.

- Master-slave replication
- Peer-to-peer replication

Distribution Models

Replications

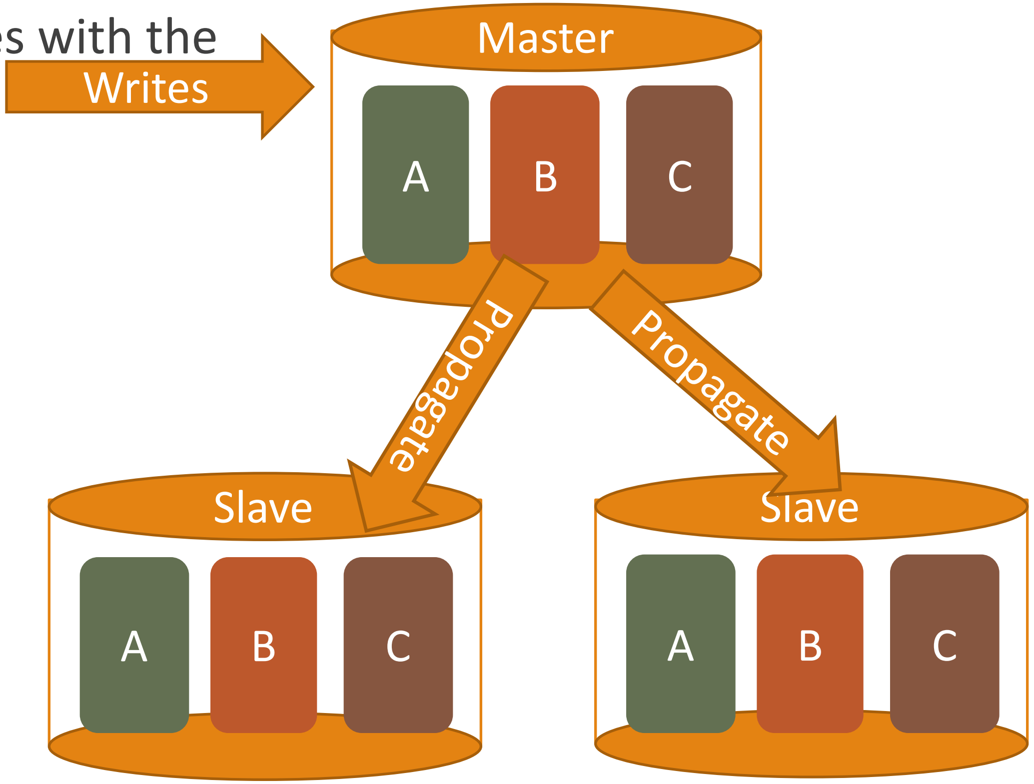
- Master-slave replication : Synchronize slaves with the master.
  - Structure
    - Master(primary)
      - Authorative source for the data.
      - Responsible for processing updates.
      - Manually or automatically assigned.
    - Slaves (secondaries)
  - Pros
    - Good scalability with intensive read.
    - Read resilience.
  - Cons
    - Poor with intensive writes.
    - Inconsistency.



Distribution Models

Replications

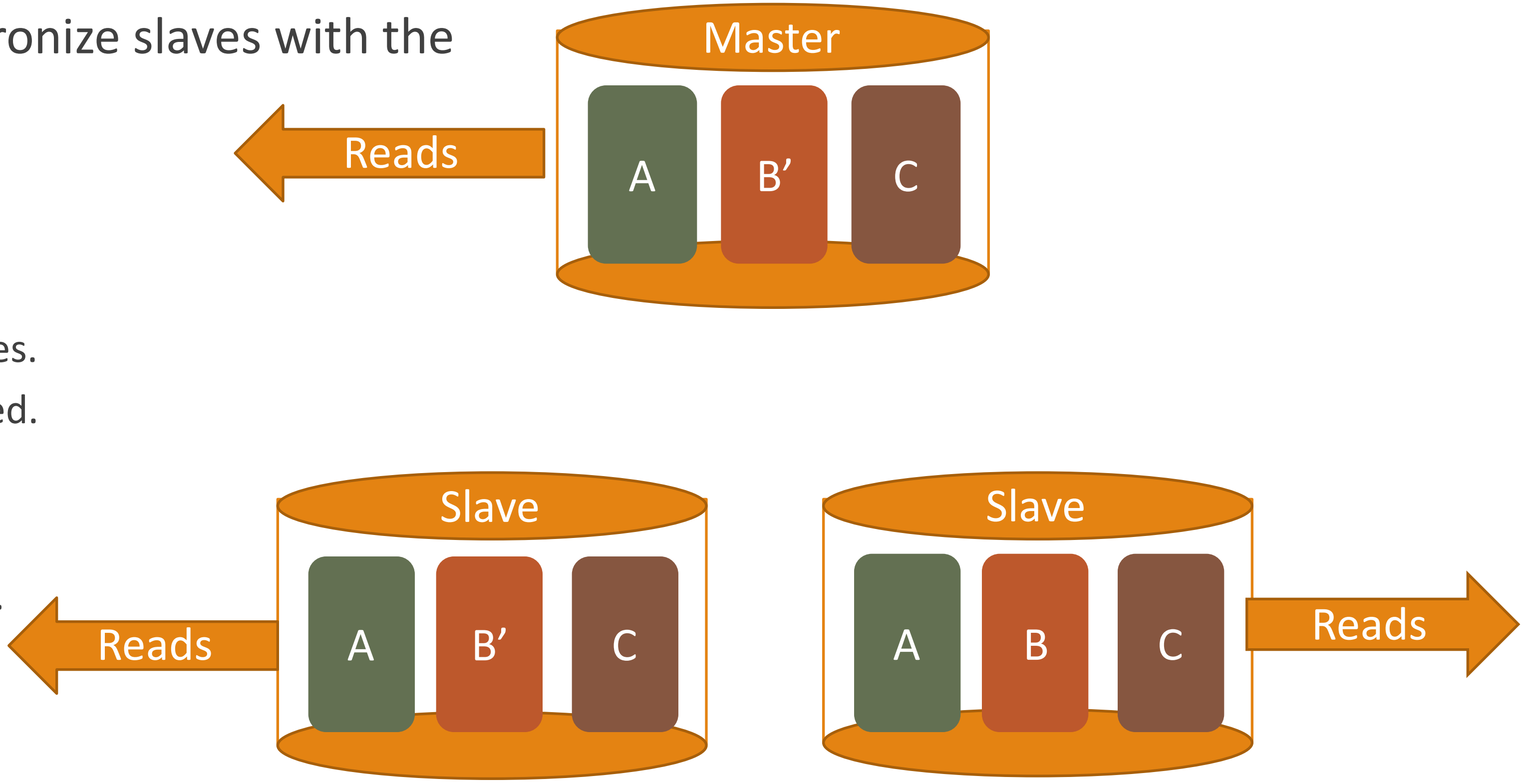
- Master-slave replication : Synchronize slaves with the master.
  - Structure
    - Master(primary)
      - Authorative source for the data.
      - Responsible for processing updates.
      - Manually or automatically assigned.
    - Slaves (secondaries)
  - Pros
    - Good scalability with intensive read.
    - Read resilience.
  - Cons
    - Poor with intensive writes.
    - Inconsistency.



Distribution Models

Replications

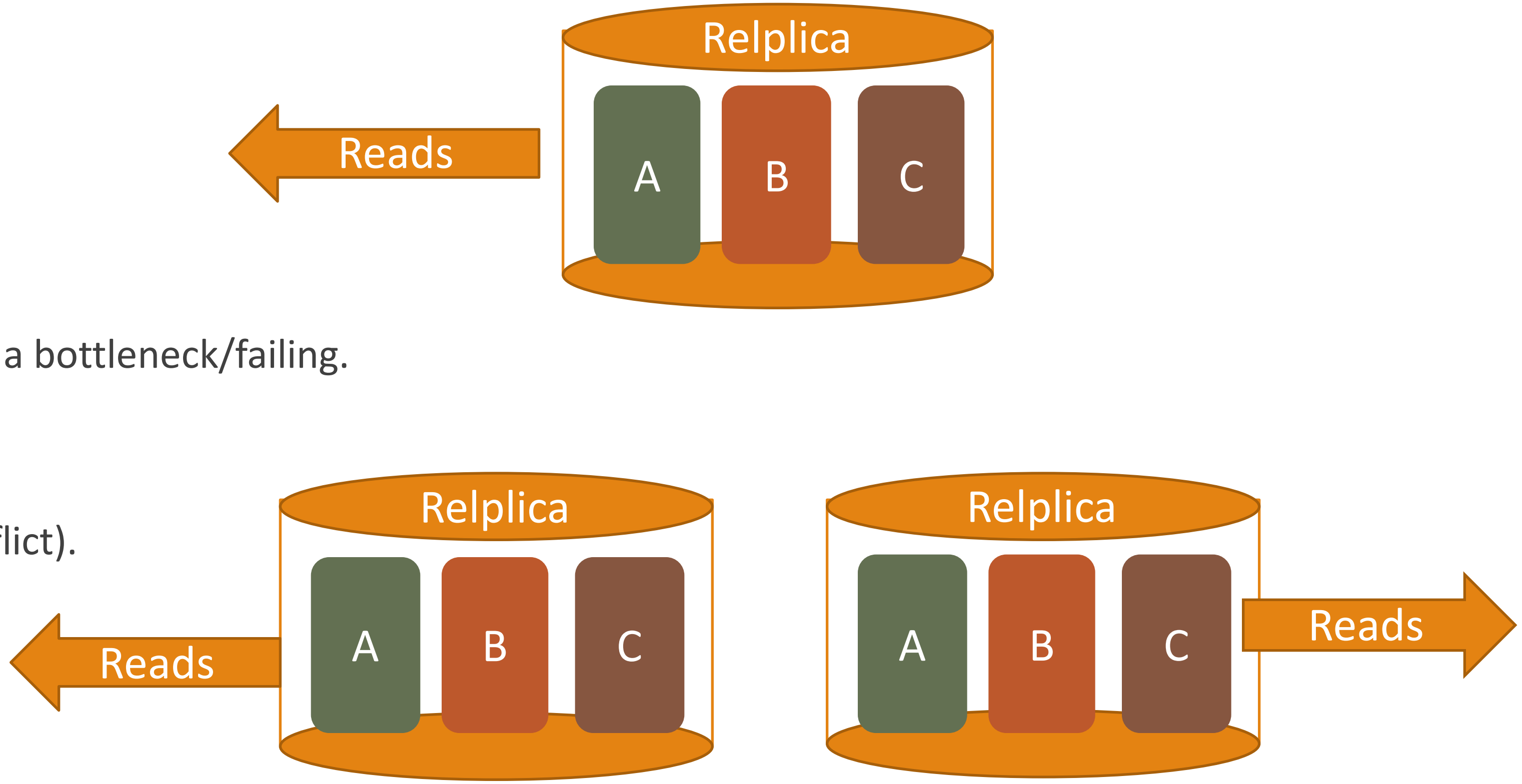
- Master-slave replication : Synchronize slaves with the master.
  - Structure
    - Master(primary)
      - Authorative source for the data.
      - Responsible for processing updates.
      - Manually or automatically assigned.
    - Slaves (secondaries)
  - Pros
    - Good scalability with intensive read.
    - Read resilience.
  - Cons
    - Poor with intensive writes.
    - Inconsistency.



Distribution Models

Replications

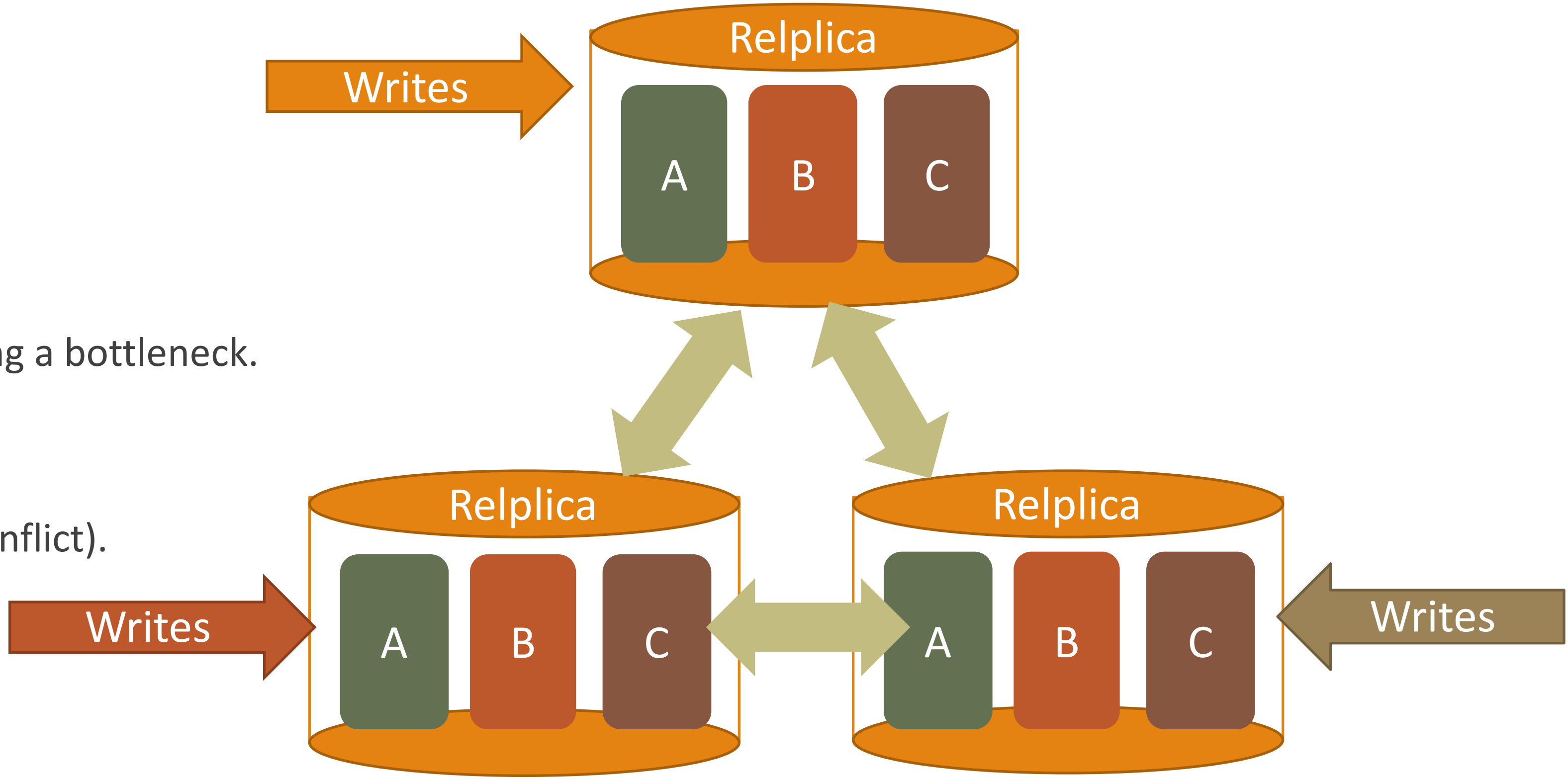
- Peer-to-peer replication
  - All replicas have equal weight.
  - All replicas accept reads/writes.
  - Pros
    - Higher availability.
      - No worries about one node being a bottleneck/failing.
    - Good performance.
  - Cons
    - Inconsistent write. (Write-write conflict).



Distribution Models

Replications

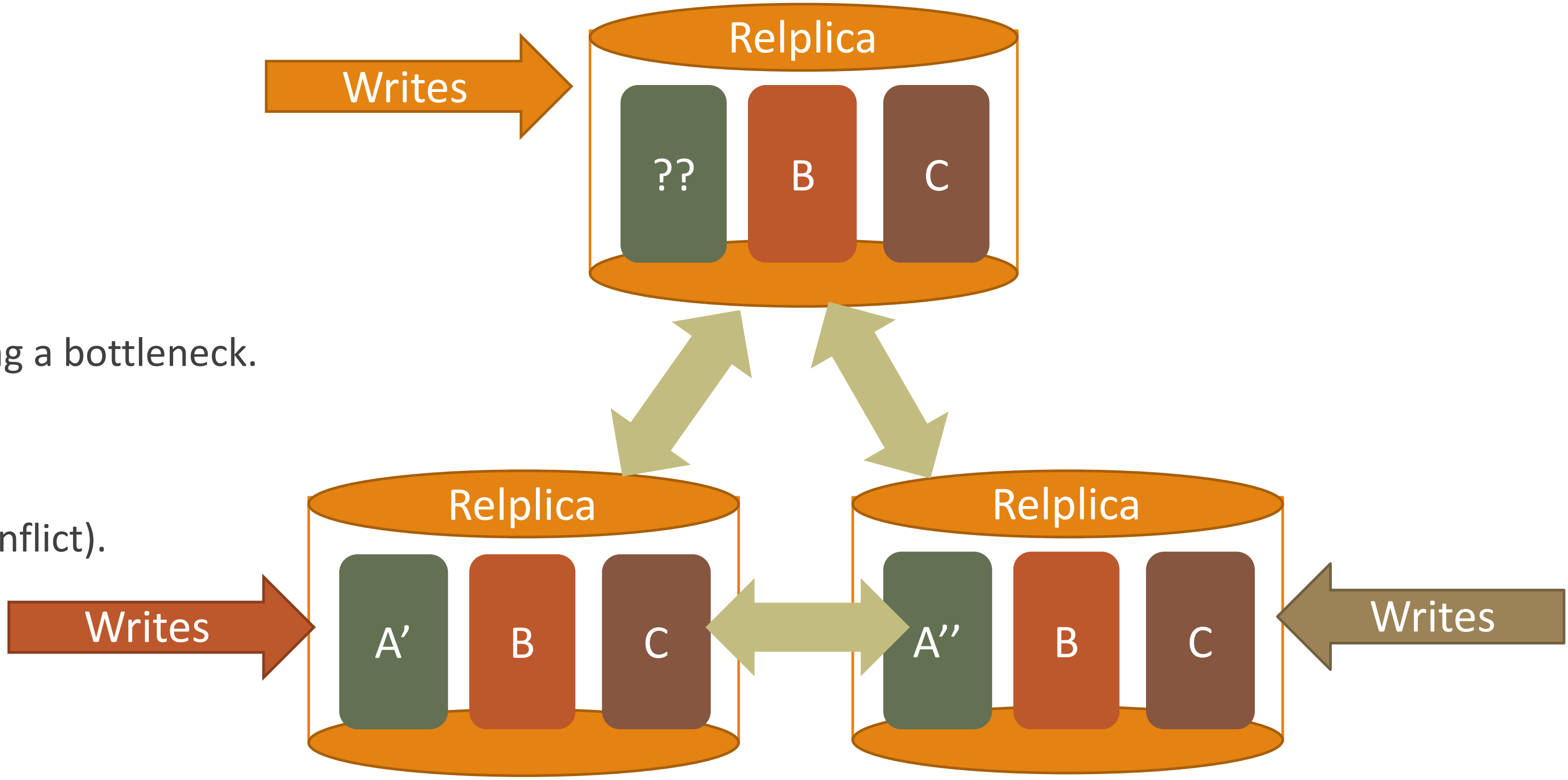
- Peer-to-peer replication
  - All replicas have equal weight.
  - All replicas accept reads/writes.
  - Pros
    - Higher availability.
      - No worries about one node being a bottleneck.
    - Good performance.
  - Cons
    - Inconsistent write. (Write-write conflict).



Distribution Models

Replications

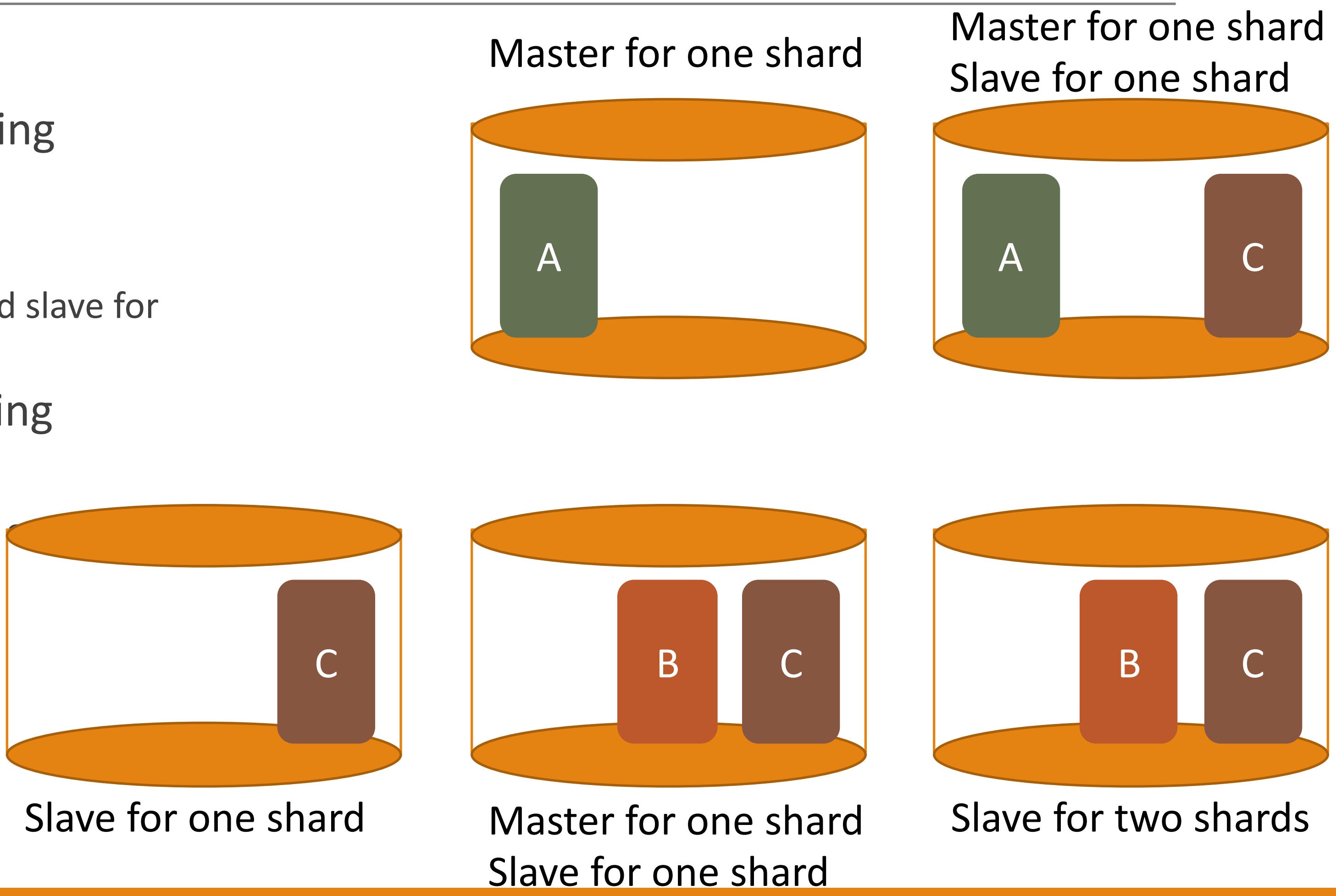
- Peer-to-peer replication
  - All replicas have equal weight.
  - All replicas accept reads/writes.
  - Pros
    - Higher availability.
      - No worries about one node being a bottleneck.
    - Good performance.
  - Cons
    - Inconsistent write. (Write-write conflict).



Distribution Models

Sharding + Replications

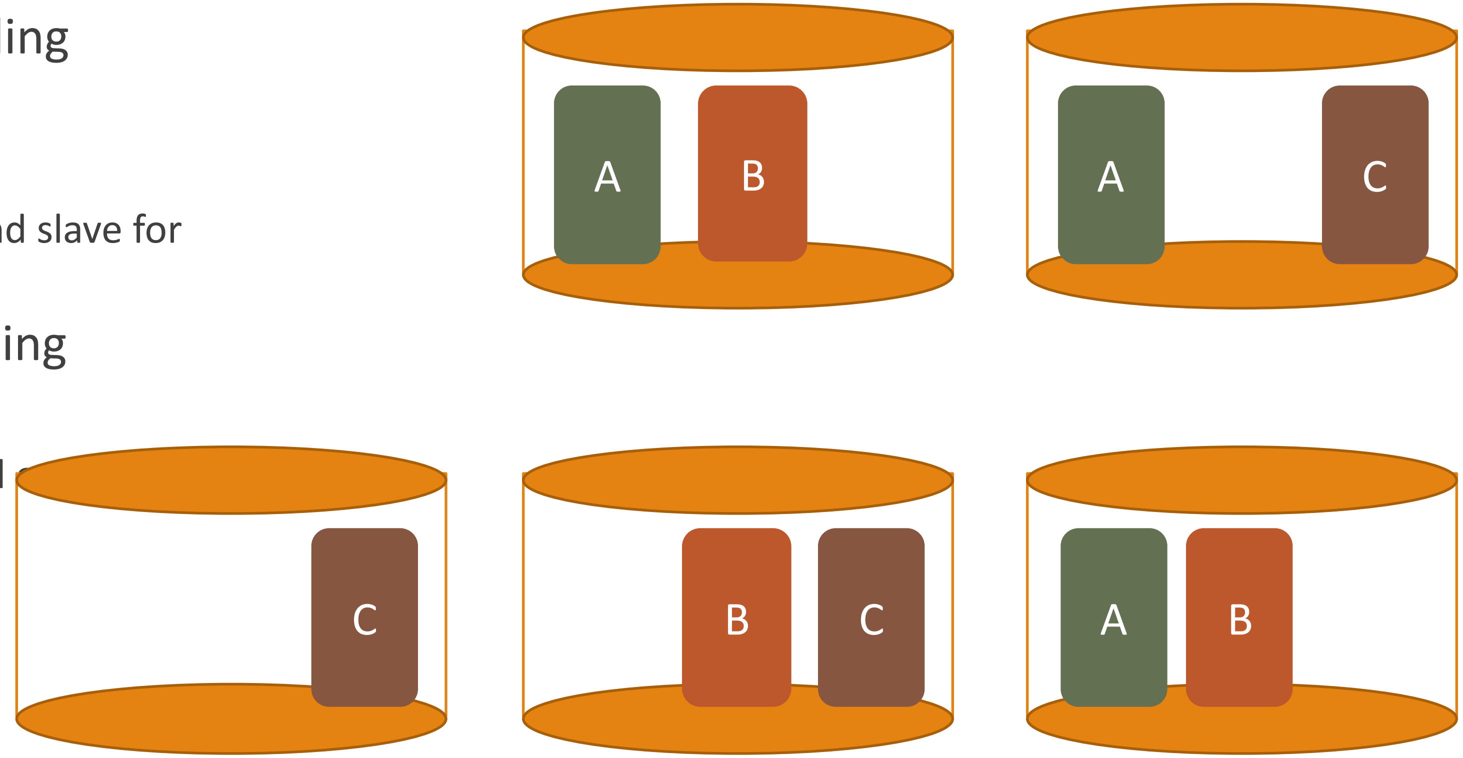
- Master-slave replication and sharding
  - Multiple masters.
  - Each data only has one master.
  - A node can be a master for some data and slave for others.
- Peer-to-peer replication and sharding
  - Common for column-family databases.
  - Many nodes in a cluster with data shared



Distribution Models

Sharding + Replications

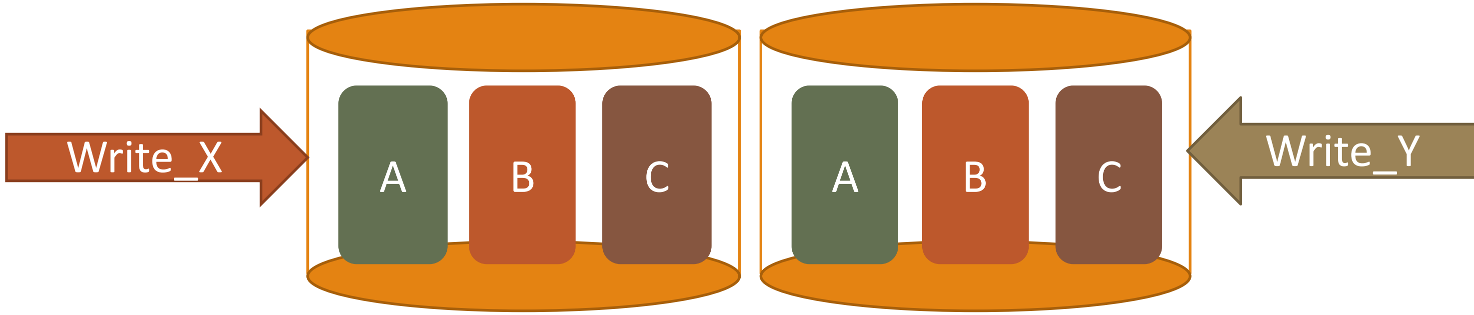
- Master-slave replication and sharding
  - Multiple masters.
  - Each data only has one master.
  - A node can be a master for some data and slave for others.
- Peer-to-peer replication and sharding
  - Common for column-family databases.
  - Many nodes in a cluster with data shared



Consistency

Update Consistency

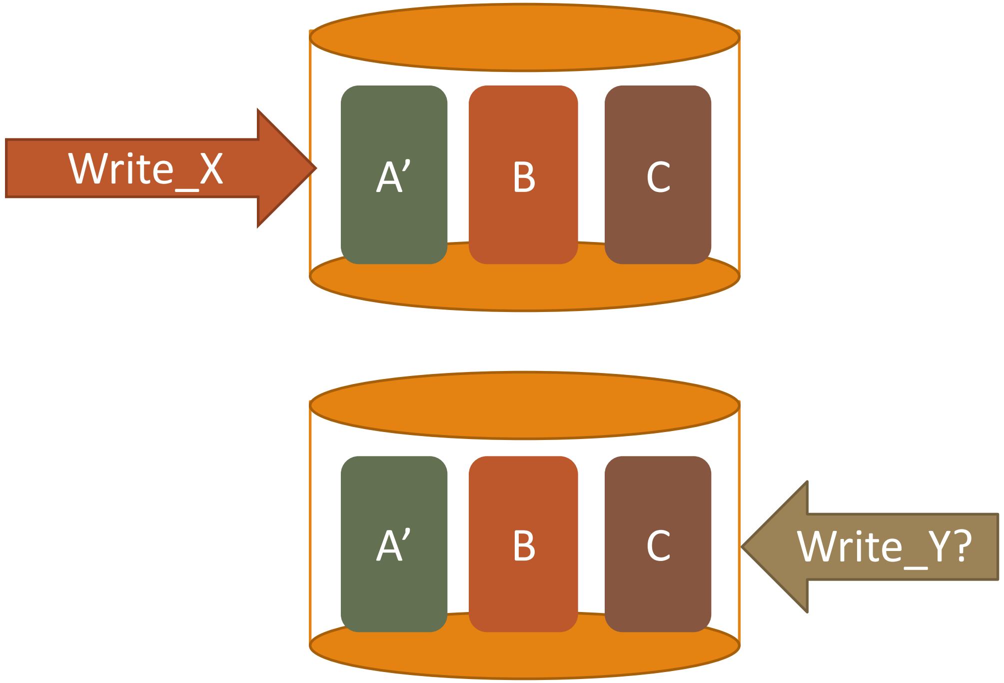
- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.



Consistency

Update Consistency

- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.

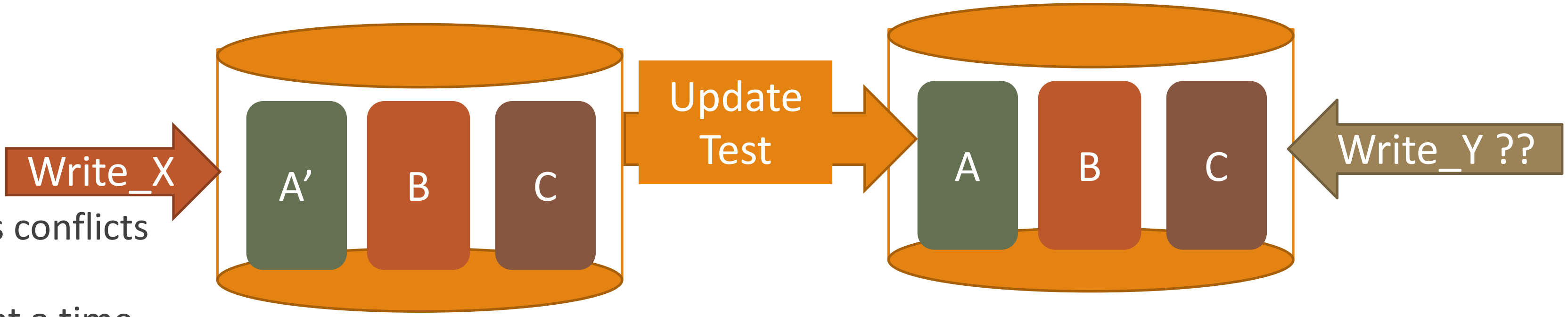


Consistency

Update Consistency

- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.

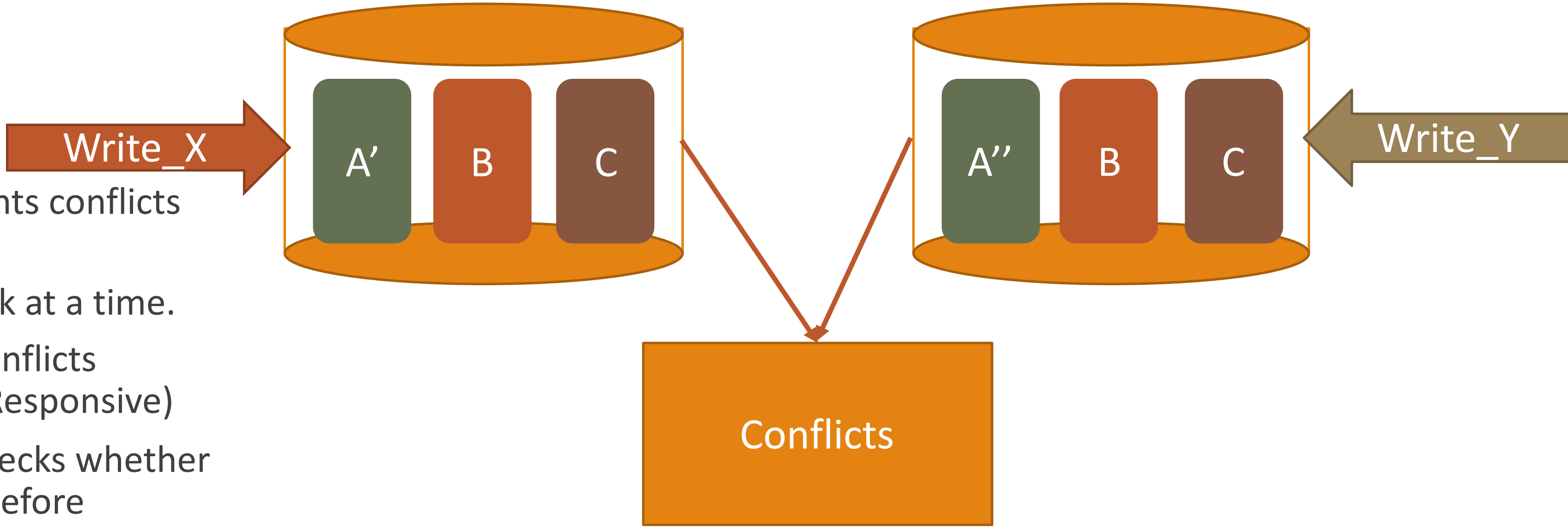
Requires sequential consistency.



Consistency

Update Consistency

- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
  - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.



Consistency

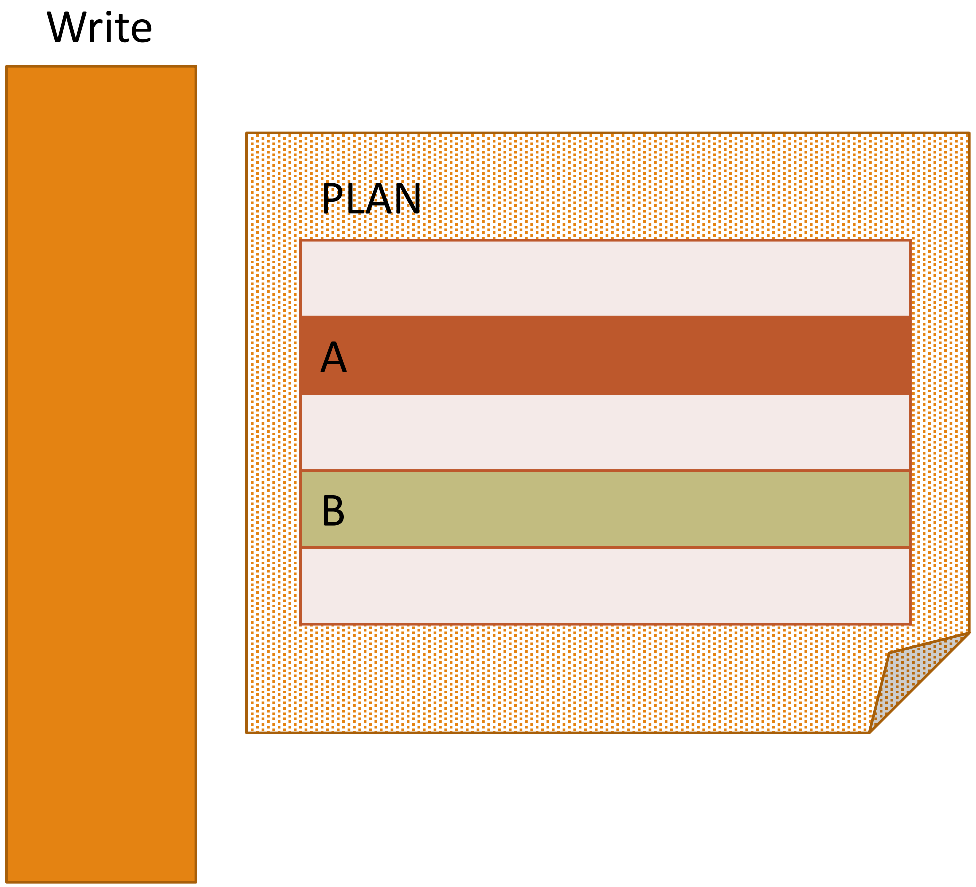
Read Consistency

- Read-write conflict (inconsistent read).
- Replication inconsistency.
- Session inconsistency.

Consistency

Read Consistency

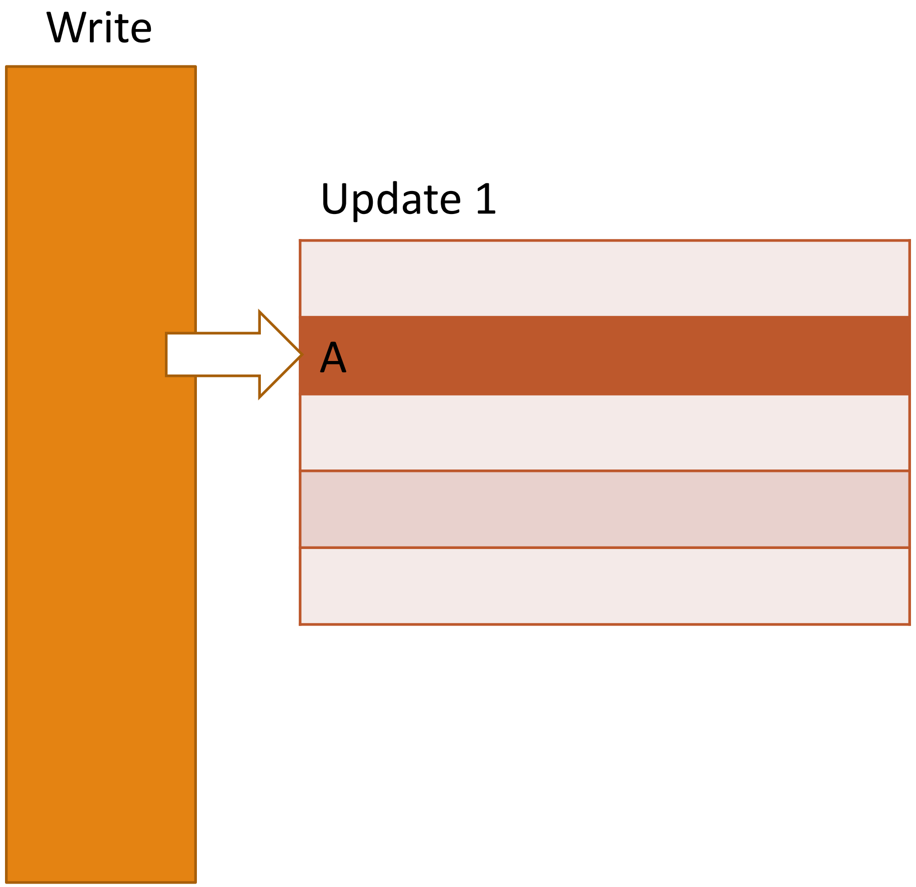
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

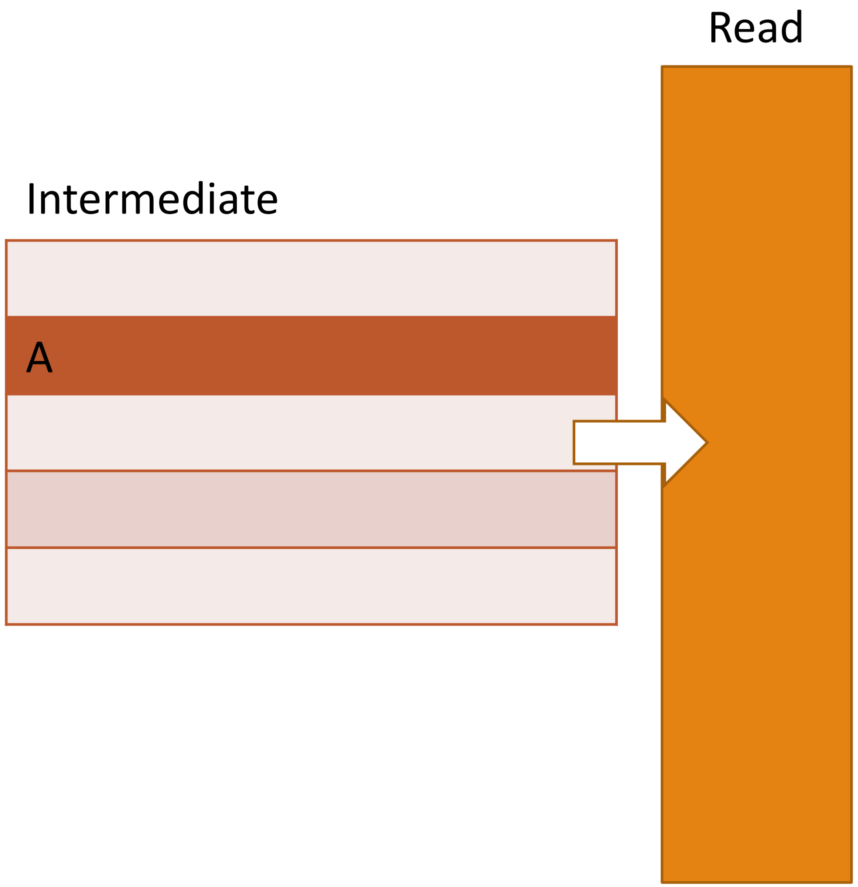
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

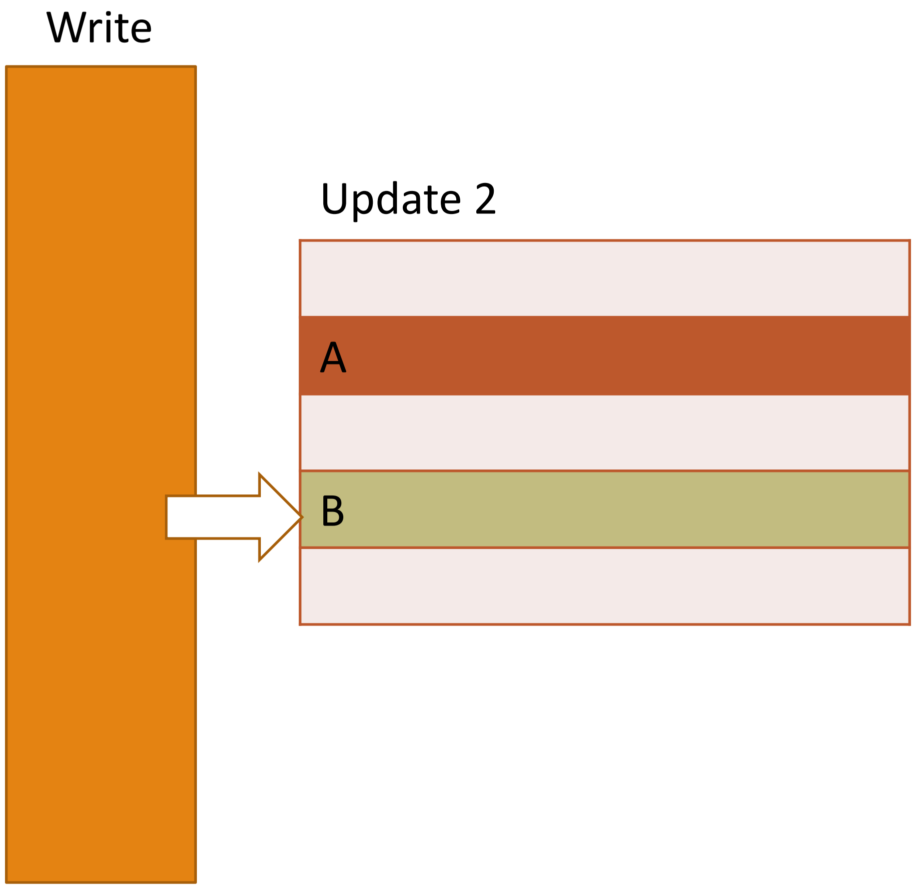
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

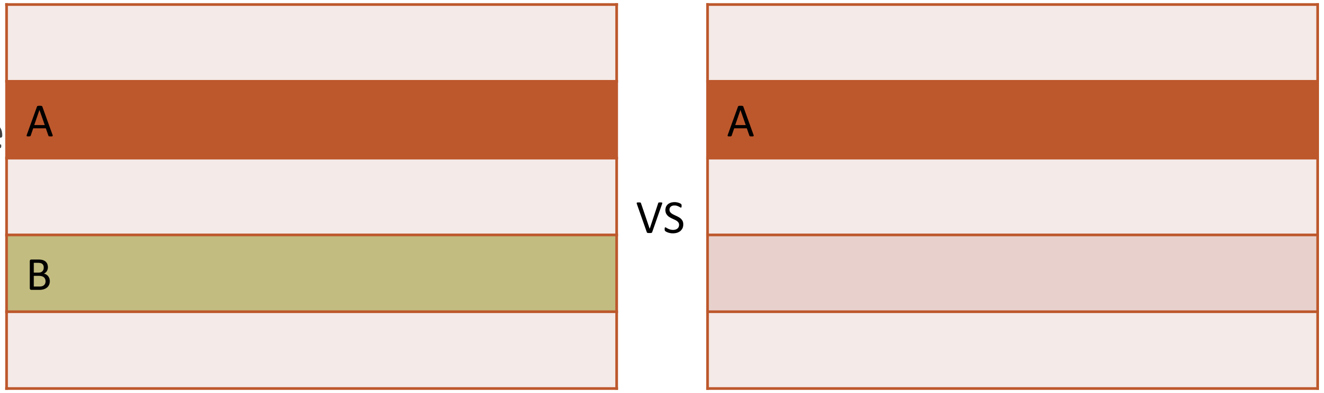
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

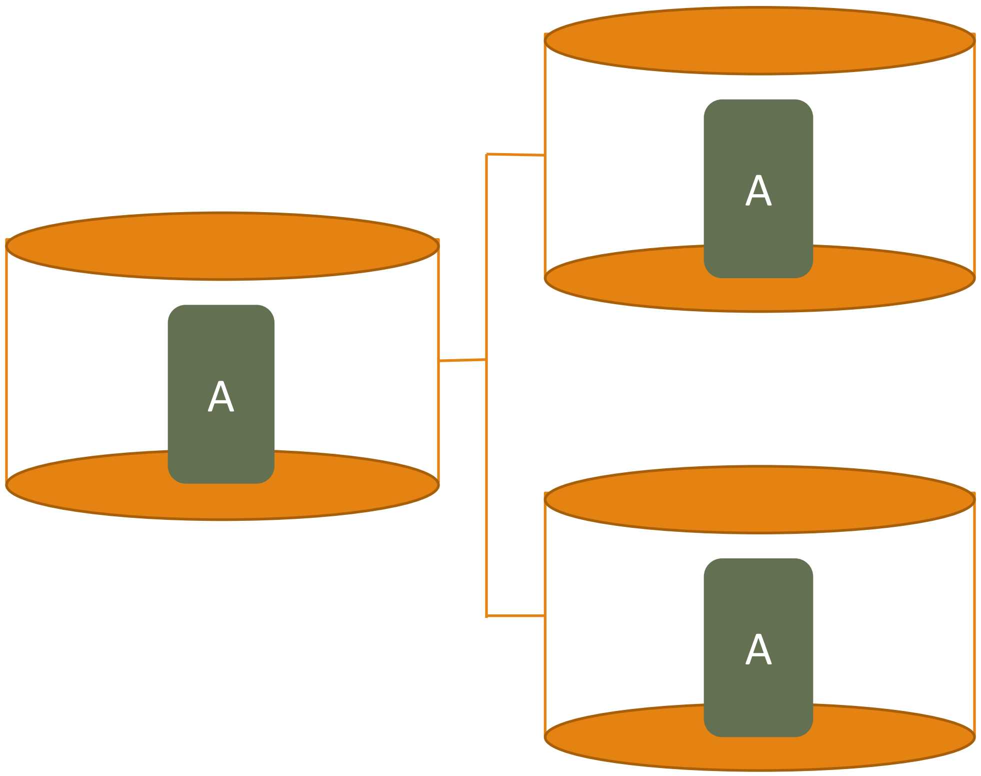
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

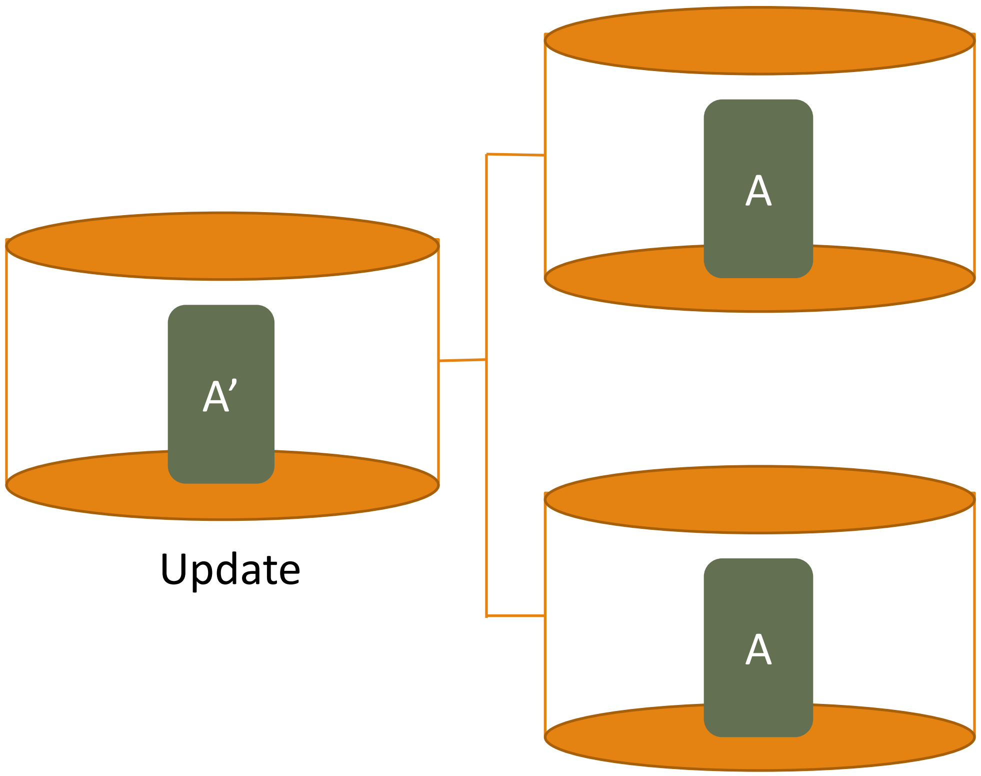
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

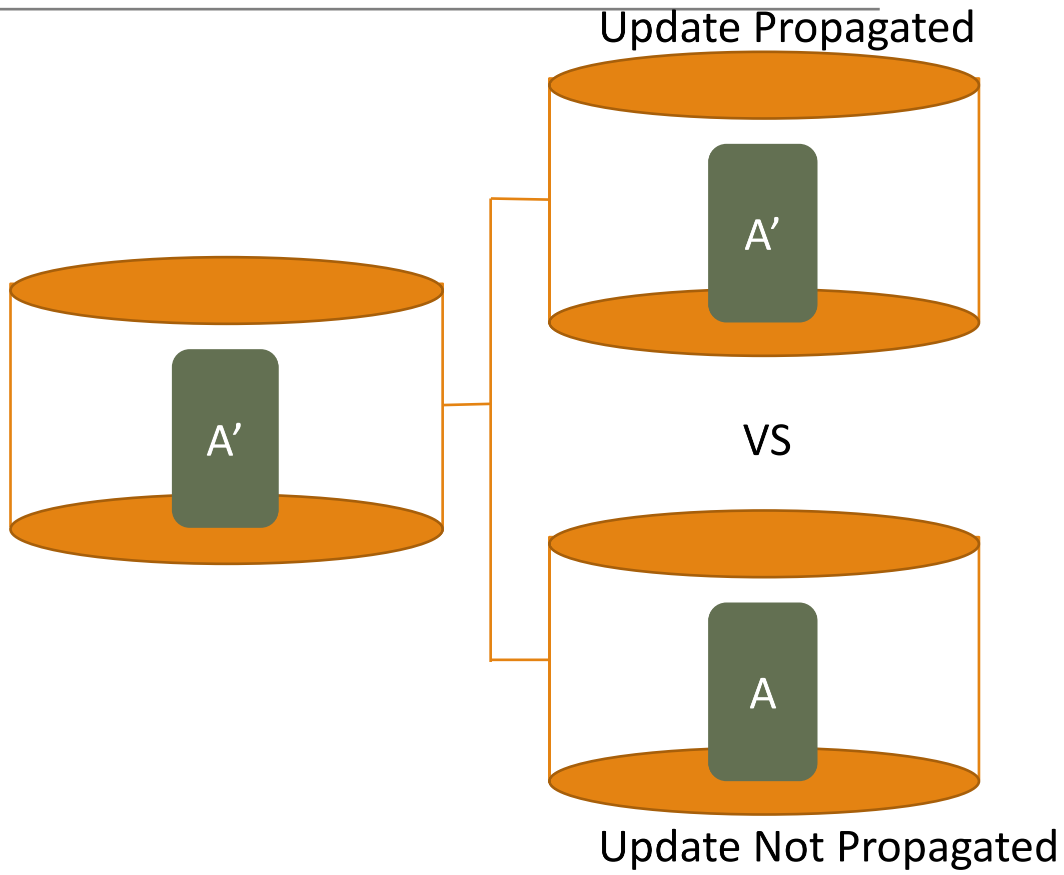
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

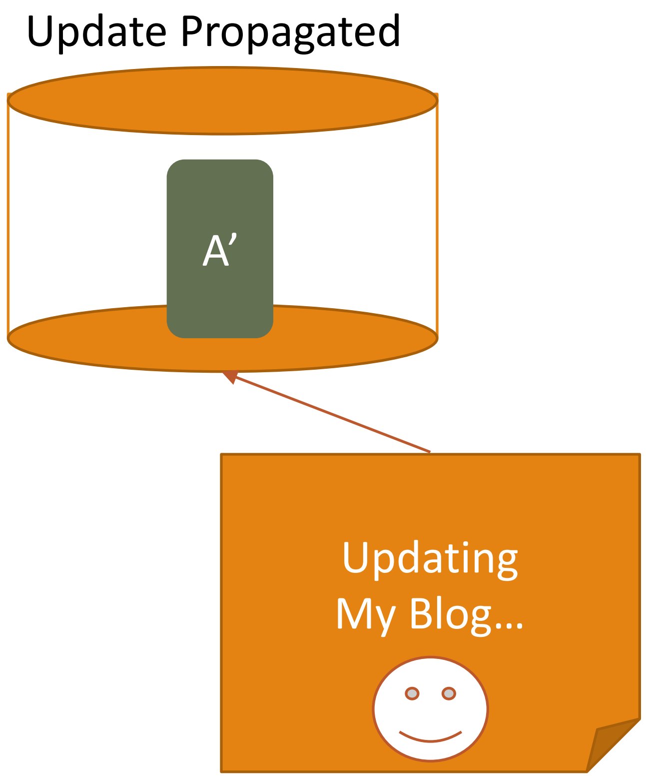
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

- Session Consistency (Read-your-writes Consistency)
  - Sticky Session (Session affinity): A session is tied to one node and keep session consistency on the node.
  - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.

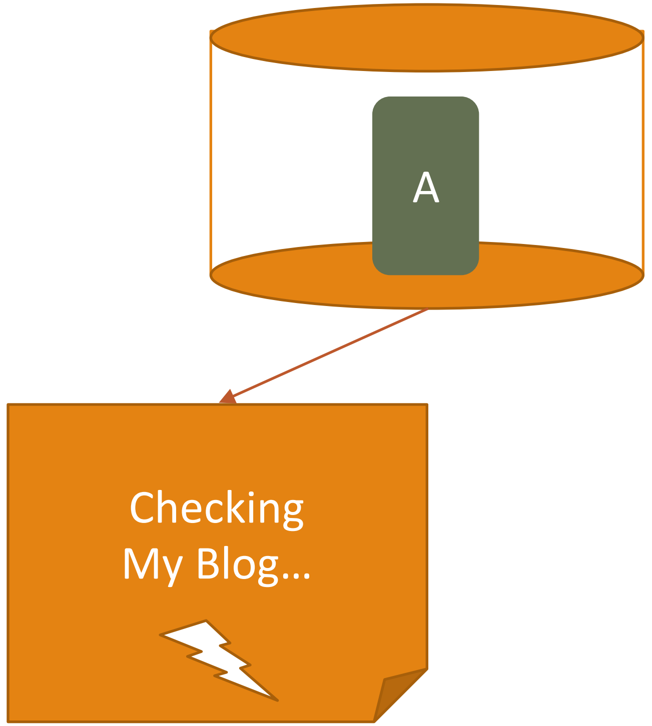


Consistency

Read Consistency

- Session Consistency (Read-your-writes Consistency)
  - Sticky Session (Session affinity) : A session is tied to one node and keep session consistency on the node.
  - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.

Update Not Propagated

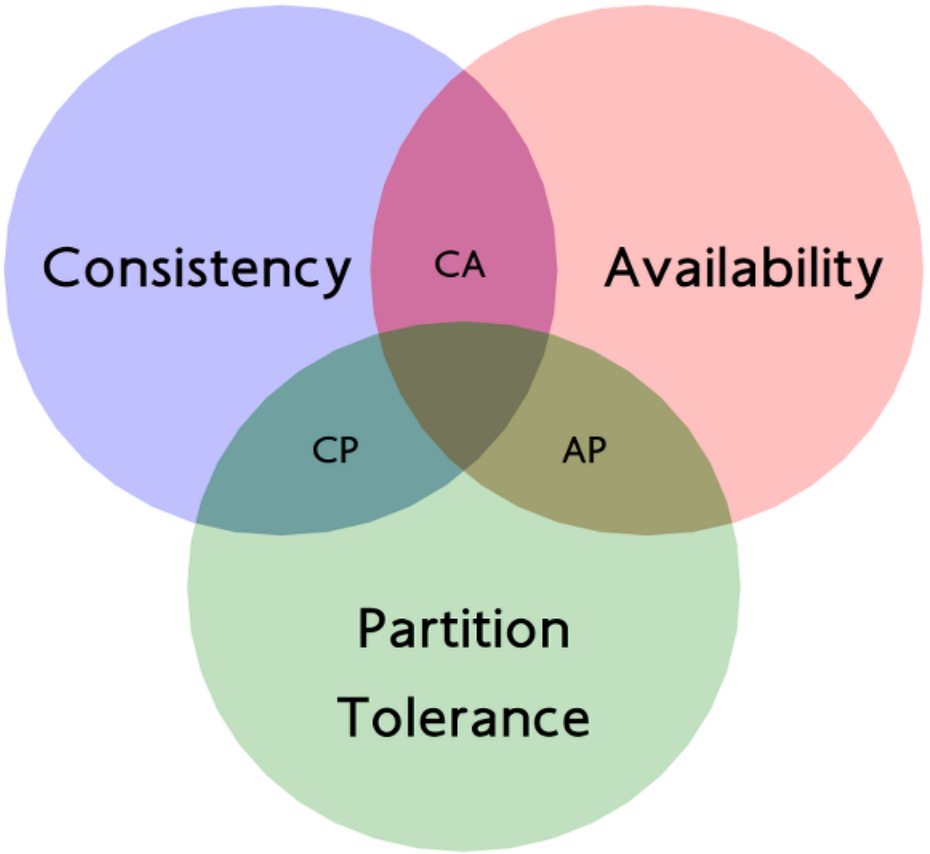


Consistency

Relaxing Consistency/Availability

- CAP Theorem
  - Consistency
    - All nodes have most recent data via eventual consistency.
  - Availability
    - Every request received by a non-failing node must return a response.
  - Partition Tolerance
    - Clusters can survive from communication breakages in the cluster.

  ➔ You can only get two.



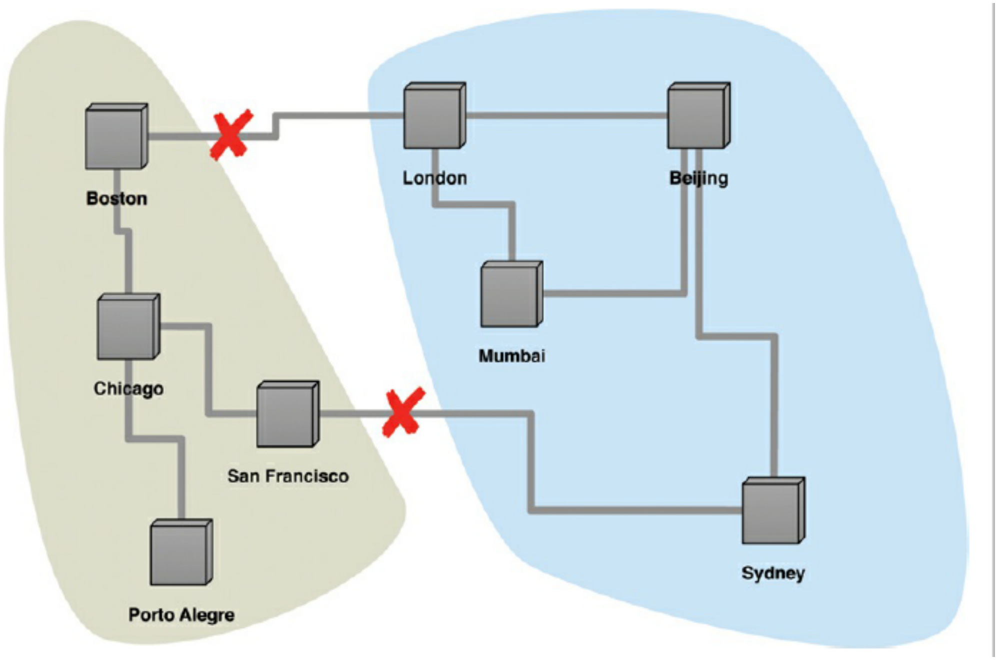
http://blingtechs.blogspot.com/2016/02/cap-theorem.html

Consistency

Relaxing Consistency/Availability

- CAP Theorem
  - Consistency
    - All nodes have most recent data via eventual consistency.
  - Availability
    - Every request received by a non-failing node must return a response.
  - Partition Tolerance
    - Clusters can survive from communication breakages in the cluster.
  - ➔ You can only get two.

ACID addresses an individual node's data consistency.
CAP addresses cluster-wide data consistency .

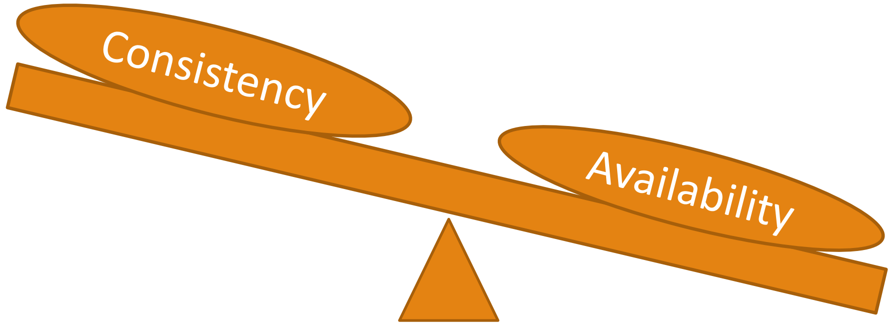


http://blingtechs.blogspot.com/2016/02/cap-theorem.html

Consistency

Relaxing Consistency/Availability

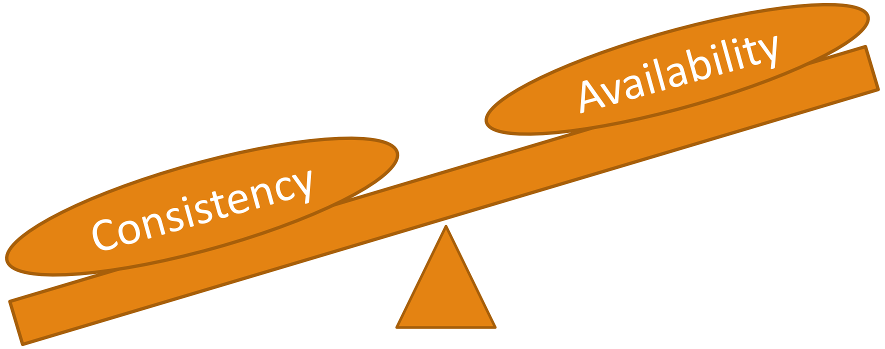
- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market



Consistency

Relaxing Consistency/Availability

- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market



Consistency

Relaxing Consistency/Availability

- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market

Consistency

Quorum

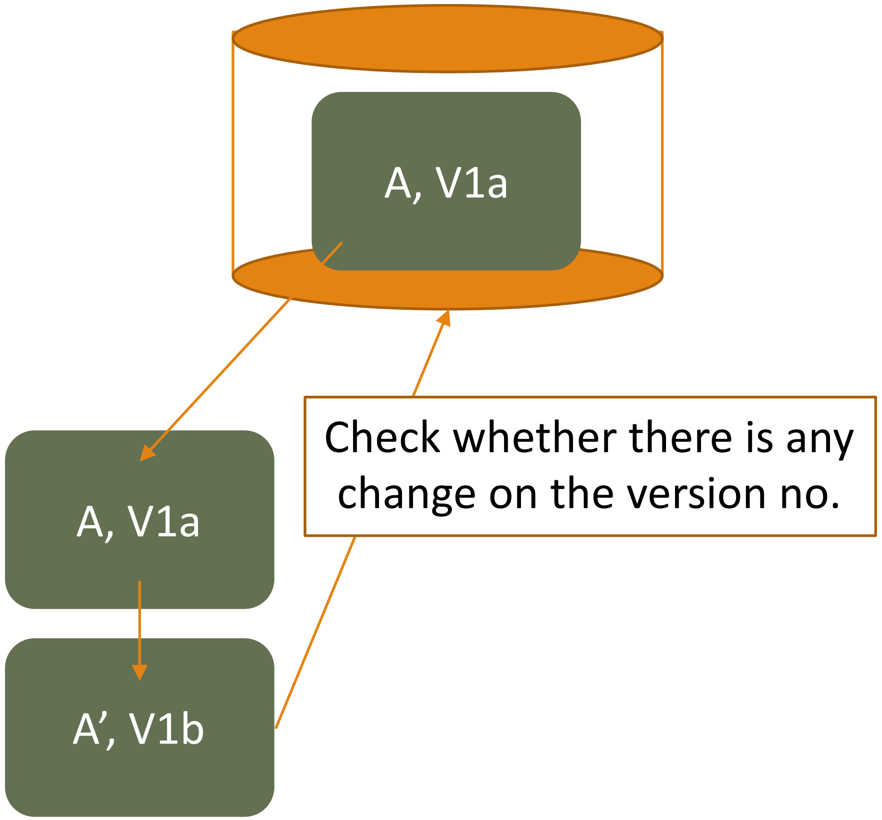
- Write Quorum : How many nodes should acknowledge your writes?
  - W > N/2 (W: nodes sending confirmation in write, N – nodes involved in replication (replication factor))
- Read Quorum : How many nodes you need to contact to make sure you have the most up-to-date value?
  - Depending on W.
  - For strong read consistency, R+W > N (R: nodes to be contacted for read consistency)

Version Stamps

Version Stamps : A field that changes every time when the data value is changed.

- Check updates won't be based on stale data.
- Example
  - Counter – increment when you update the data. Need a single master.
  - GUID – A large random number (Unique). Can't tell the recentness.
  - Resource content hash - Deterministic. Can't tell the recentness.
  - Timestamp - Clocks have to be kept in sync. Can't take care of too granular updates.

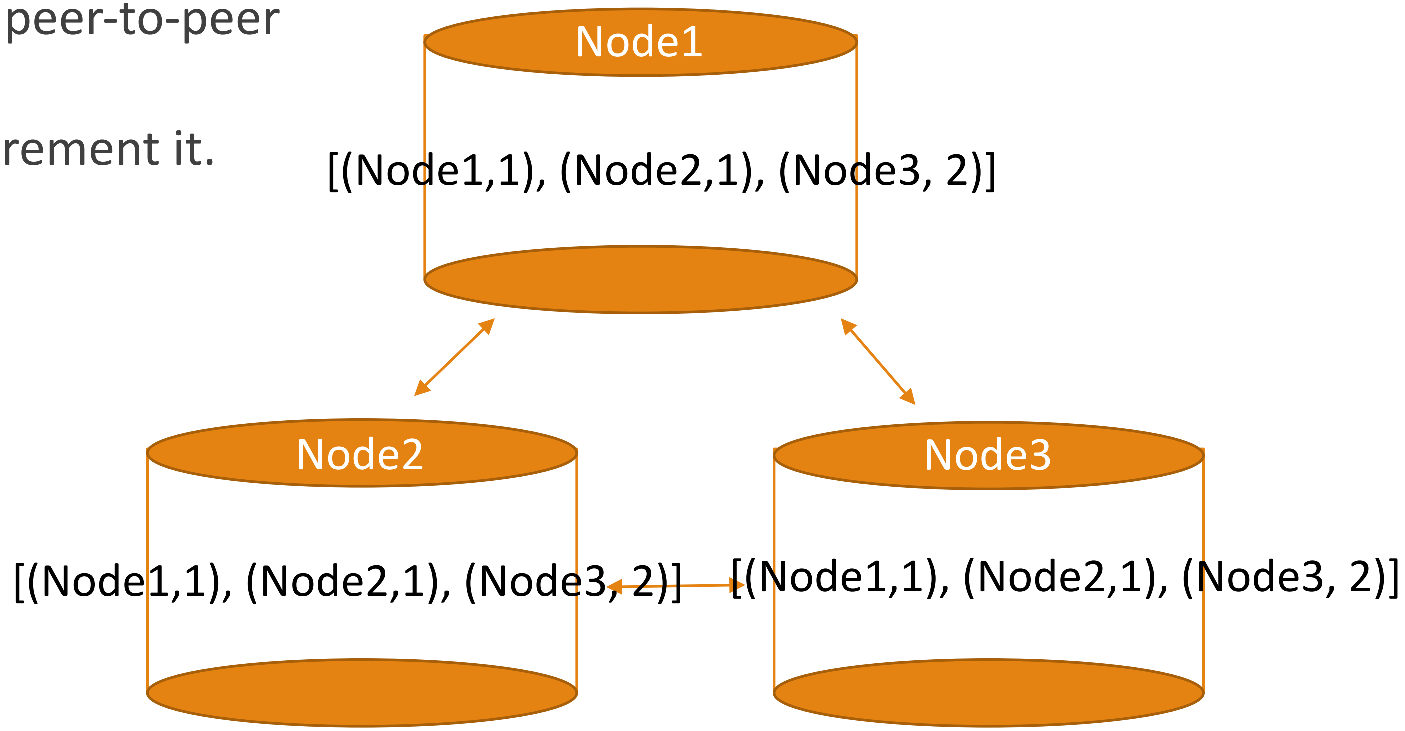
  ➔ Use more than one.



Version Stamps

Vector Stamp

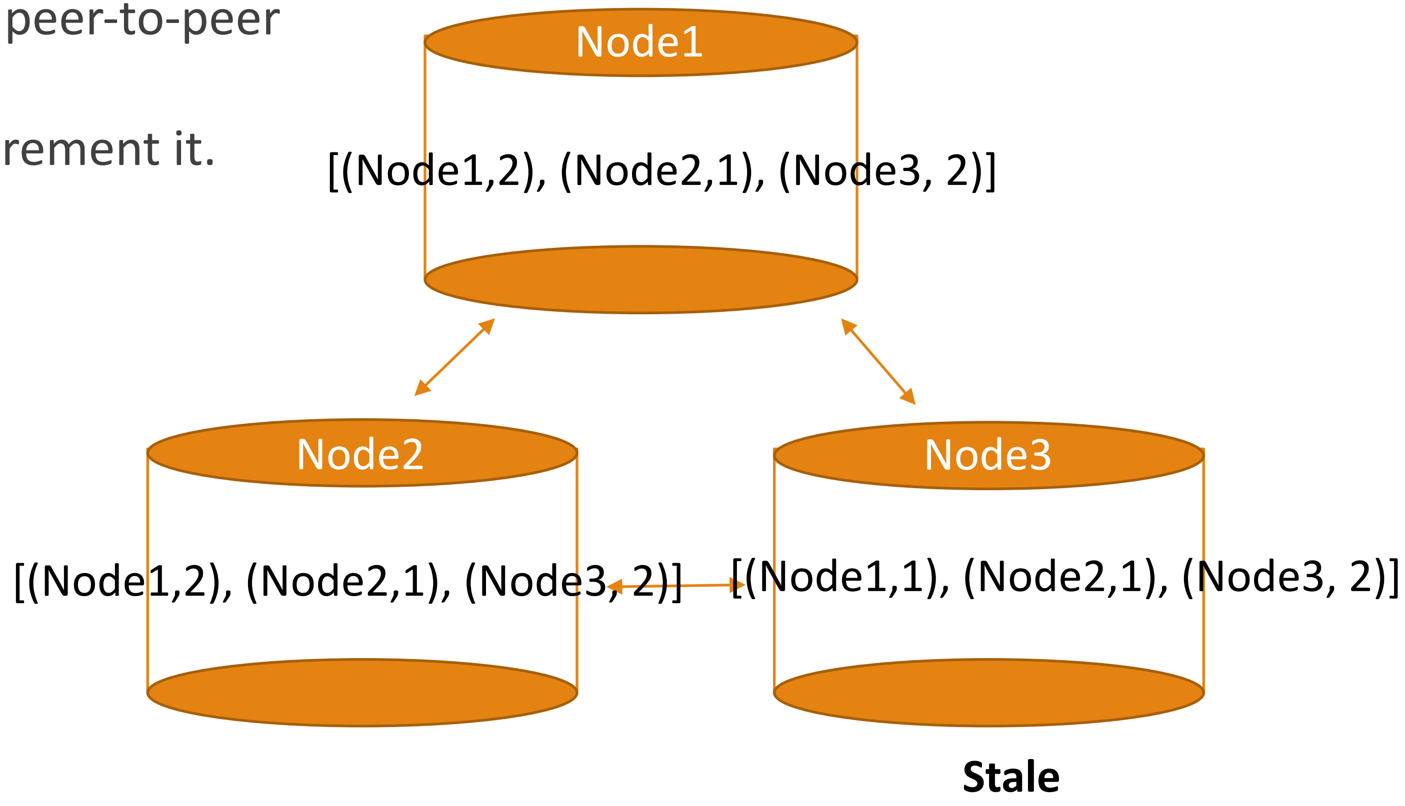
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



Version Stamps

Vector Stamp

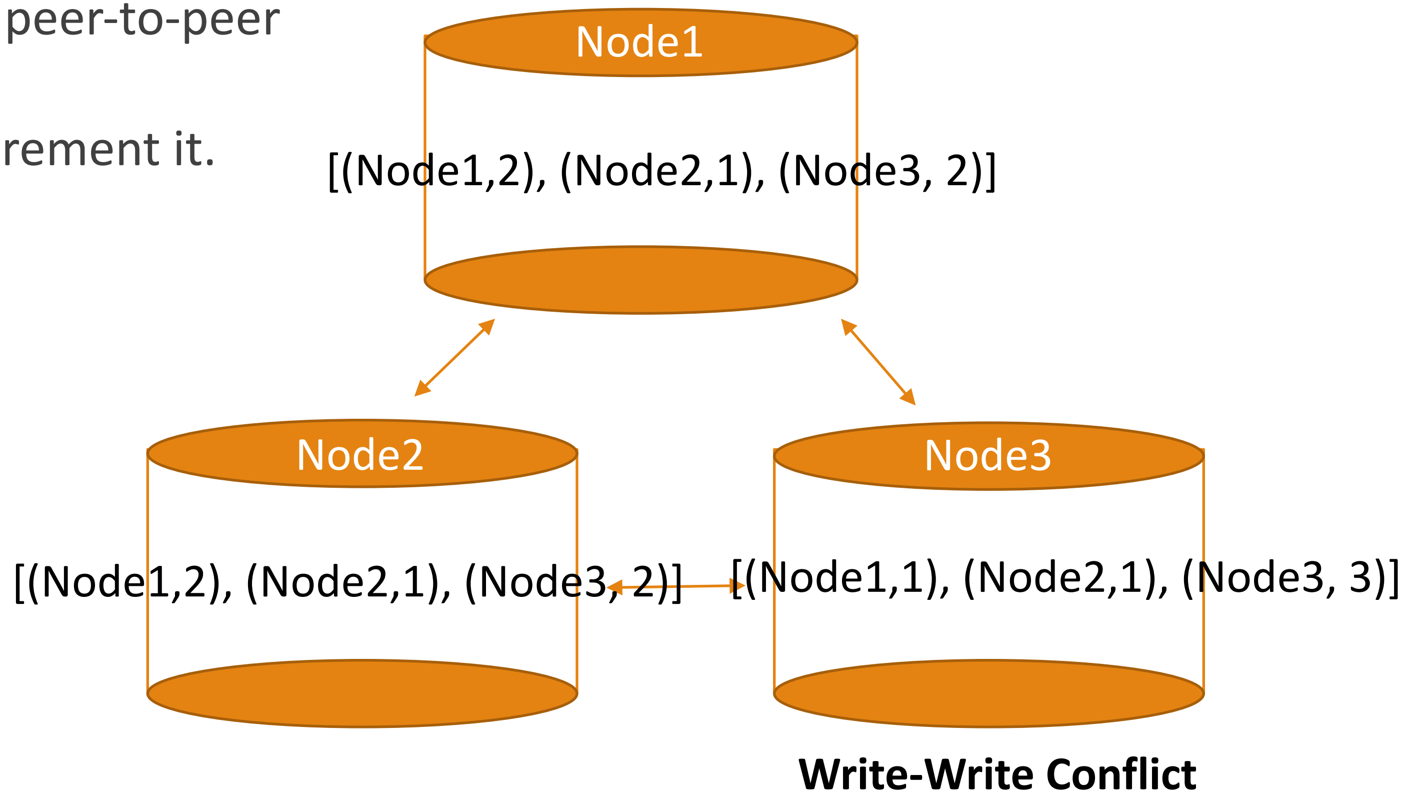
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



Version Stamps

Vector Stamp

- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



Quiz

Date: November 11th.

Topics : Week 1, Week2 and Week3.

(Those topics are going to be also in the final exam, too.)

NoSQL Interview Questions

What is NoSQL?

Relational Database vs. NoSQL

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

CAP Theorem

Eventual Consistency

Reference

Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

Redmond, Eric, and Jim R. Wilson. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Pragmatic Bookshelf, 2012.

Date, C. J. "Database systems." *Vols. I & II, Narosa Pub* (1986).