BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Big Data Systems

Session 4 - CAP Theorem, Big Data Lifecycle

Dr. Anindya Neogi,

Associate Professor

anindya.neogi@pilani.bits-pilani.ac.in

Topics for today

- Consistency, availability, partition tolerance
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra
- Big Data lifecycle

Consistency

- Big Data systems write replicas of a shard / partition
- Any write needs to update all replicas
- Any reads can happen in between
- Consistency —
  - **Do you allow a read of any replica in any thread to always read the latest value written in any thread ?**
    - RDBMS / OLTP systems / Systems of Record
    - ACID (Atomicity, Consistency, Isolation, Durability)
  - **Do you allow reads to return any value and eventually show the latest stable value**
    - Some BigData systems / Systems of Engagement e.g. social network comments
    - BASE (Basic Availability, Soft state, Eventual consistency)

Ref: https://www.dummies.com/programming/big-data/applying-consistency-methods-in-nosql/

ACID

- ACID is a database design principle related to transaction management
  - ✓ Atomicity
  - ✓ Consistency
  - ✓ Isolation
  - ✓ Durability

- ACID is the traditional approach to **database transaction management** as it is leveraged by relational database management systems

ACID (2)

- **Atomicity**
  - ✓ Ensures that all operations will always succeed or fail completely
  - ✓ No partial transactions

- **Consistency**
  - ✓ Ensures that only data that conforms to the constraints of database schema can be written to the database
  - ✓ Database that is in consistent state will remain in consistent stage following a successful transaction.

- **Isolation**
  - ✓ Ensures that results of transaction are not available to other operations until it is complete.
  - ✓ Critical for concurrency control.

- **Durability**
  - ✓ Ensures that results of transaction are permanent
  - ✓ Once transaction is committed , it can not be rolled back.

Consistency clarification

- C in ACID is different from C as in CAP Theorem (discussed next) *
- **C in ACID of RDBMS based OLTP systems**
  - A broader concept at a data base level for a *transaction involving multiple data items*
  - Harder to achieve
- **C in this context and in CAP Theorem for most NoSQL systems**
  - Applies to *ordering of operations for a single data item* not a transaction involving multiple data items
  - So it is a *strict subset* of C in ACID
  - Typically support of full ACID semantics for NoSQL systems defeats the purpose as it involves having a single transaction manager that becomes a scale bottleneck for large number of partitions and replicas

* https://hackingdistributed.com/2013/03/23/consistency-alphabet-soup/#:~:text=%22C%20as%20in%20ACID%22%20is,arbitrarily%20large%20groups%20of%20objects

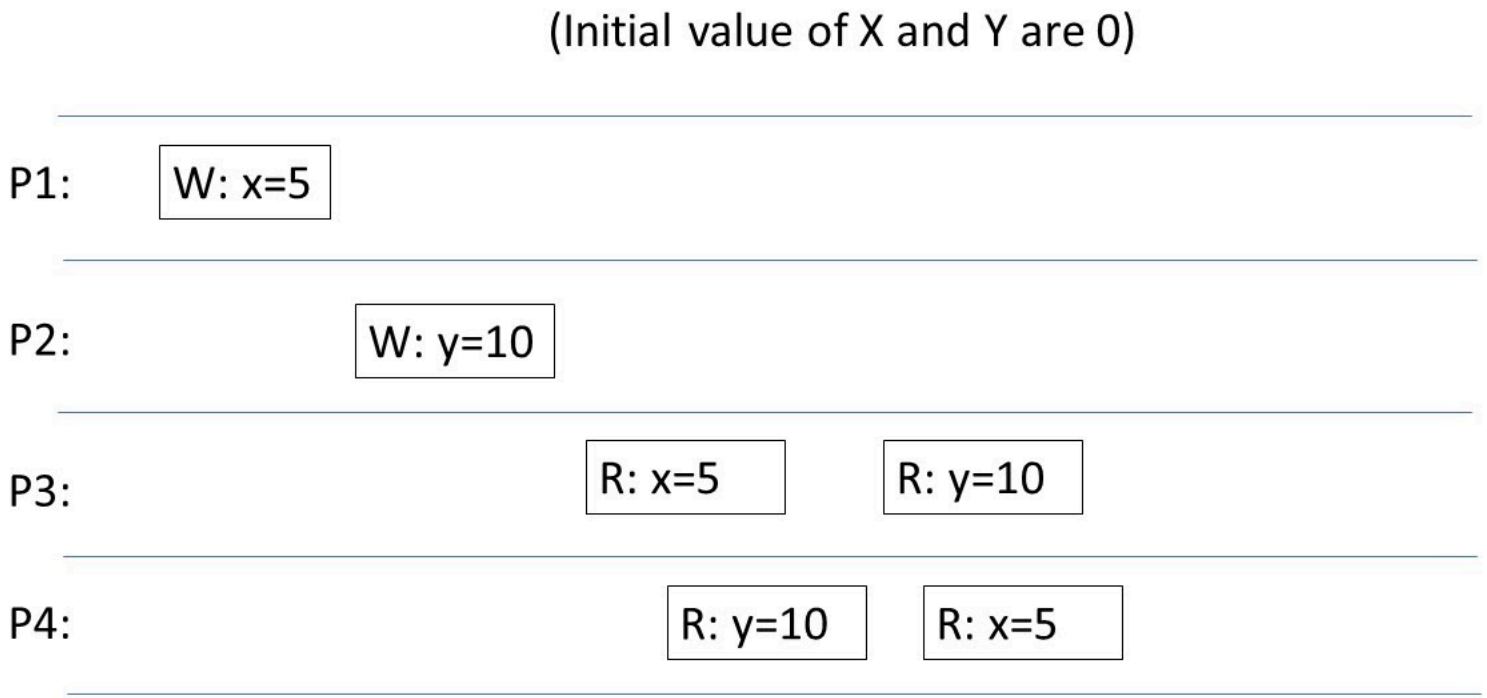
Levels of Consistency

- Strict
  - Requires **real time line ordering of all writes** - assumes actual write time can be known. So "reads" read the latest data in real time across threads.
- Linearisable
  - Acknowledges that **write requests take time to write all copies** - so does not impose ordering within overlapping time periods of read / write
- Sequential
  - All **writes across threads for all data items are globally ordered**. All threads must see the same order. But does not need real-time ordering.
- Causal
  - Popular and useful model where only **causally connected writes and reads need to be ordered**. So if a write of a data item (Y) happened after a read of same or another data item (X) in a thread then all threads must observe write X before write to Y.
- Eventual (most relaxed as in BASE)
  - If there are no writes for "some time" then all **threads will eventually agree on a latest value** of the data item

ref: http://dbmsmusings.blogspot.com/2019/07/overview-of-consistency-levels-in.html

Example: Strictly consistent

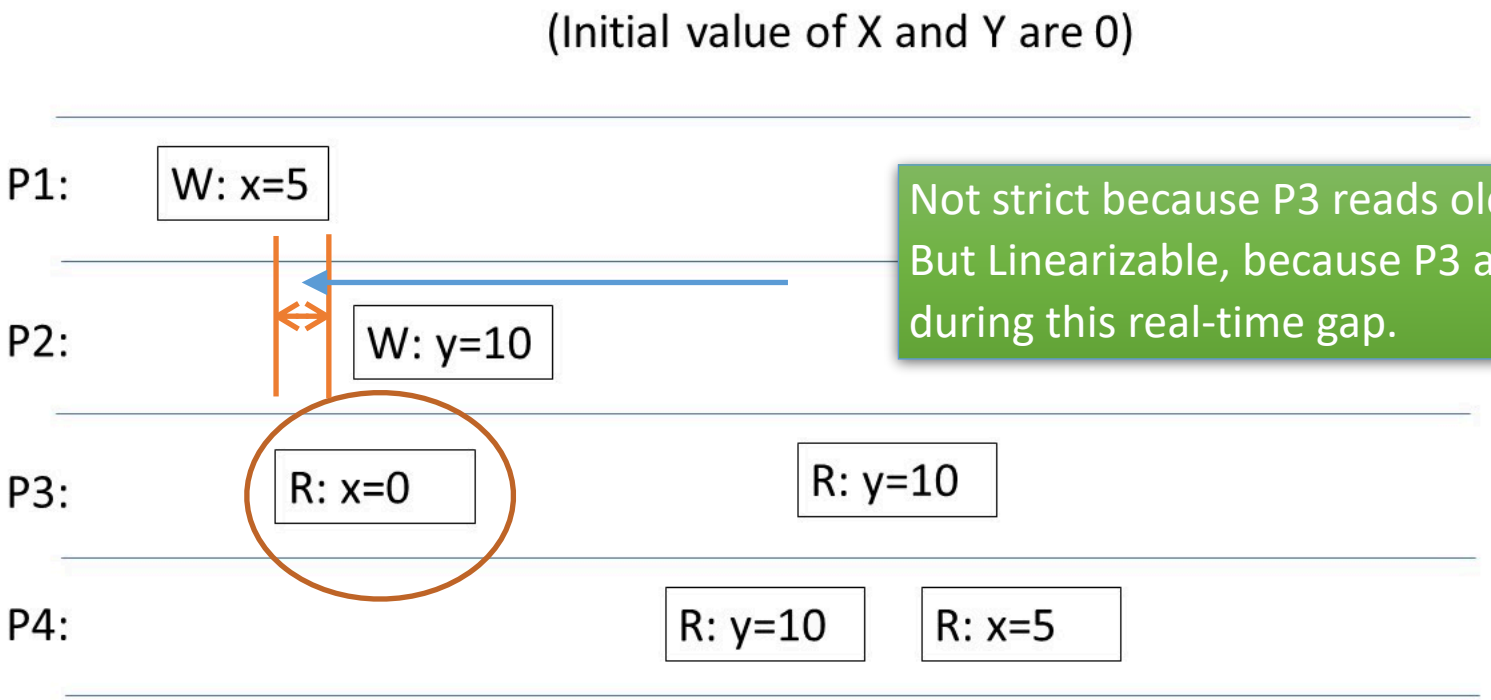
- Requires **real time line ordering of all writes** - assumes actual write time can be known. So "reads" read the latest data in real time across threads.



This schedule is sequentially consistent, causally consistent, linearizable and strictly consistent

Example: Linearizable

- Acknowledges that **write requests take time to write all copies** - so does not impose ordering within overlapping time periods of read / write

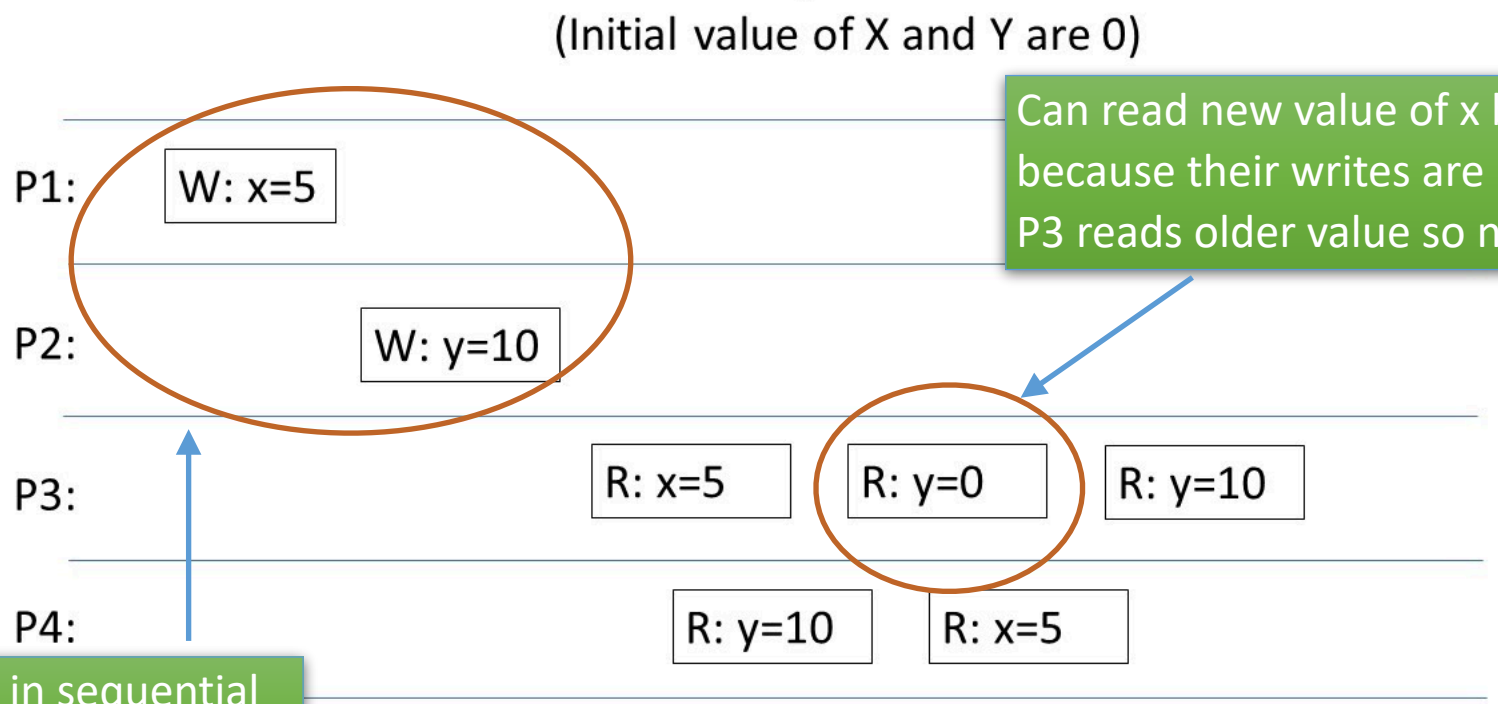


This schedule is sequentially consistent, causally consistent, and linearizable, but **not** strictly consistent

Linearizability takes into account the overlapping time when P3 could not read the latest write of x.

Example: Sequentially consistent

- All **writes across threads for all data items are globally ordered**. All threads must see the same order. But does not need real-time ordering.



Can read new value of x but ok to read old value of y because their writes are parallel.
P3 reads older value so not linearizable but sequential

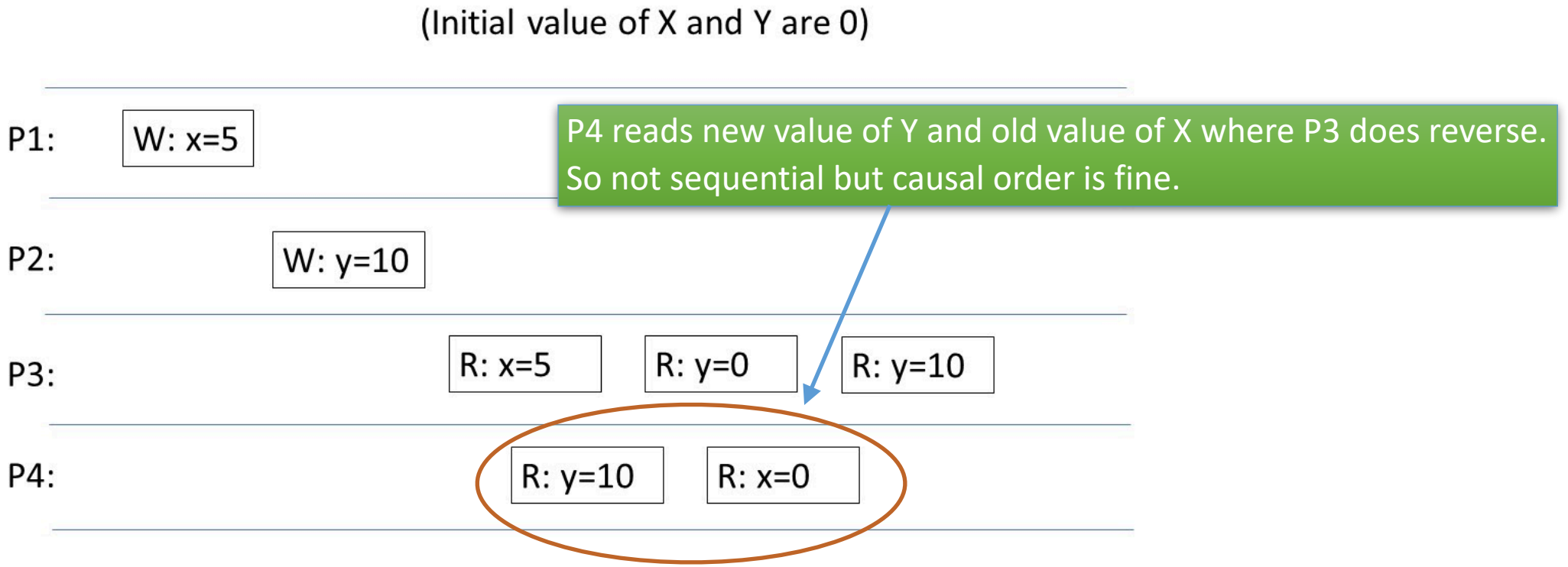
considered parallel in sequential order - real time doesn't matter

This schedule is sequentially consistent and causally consistent, but not linearizable or strictly consistent

All writes across threads are globally ordered but not in real-time. Here P3 reads older value of Y in real-time but P1 and P2 writes are deemed parallel in sequential order.

Example: Causally consistent

- Popular and useful model where only **causally connected writes and reads need to be ordered**. So if a write of a data item (Y) happened after a read of same or another data item (X) in a thread then all threads must observe write X before write to Y.

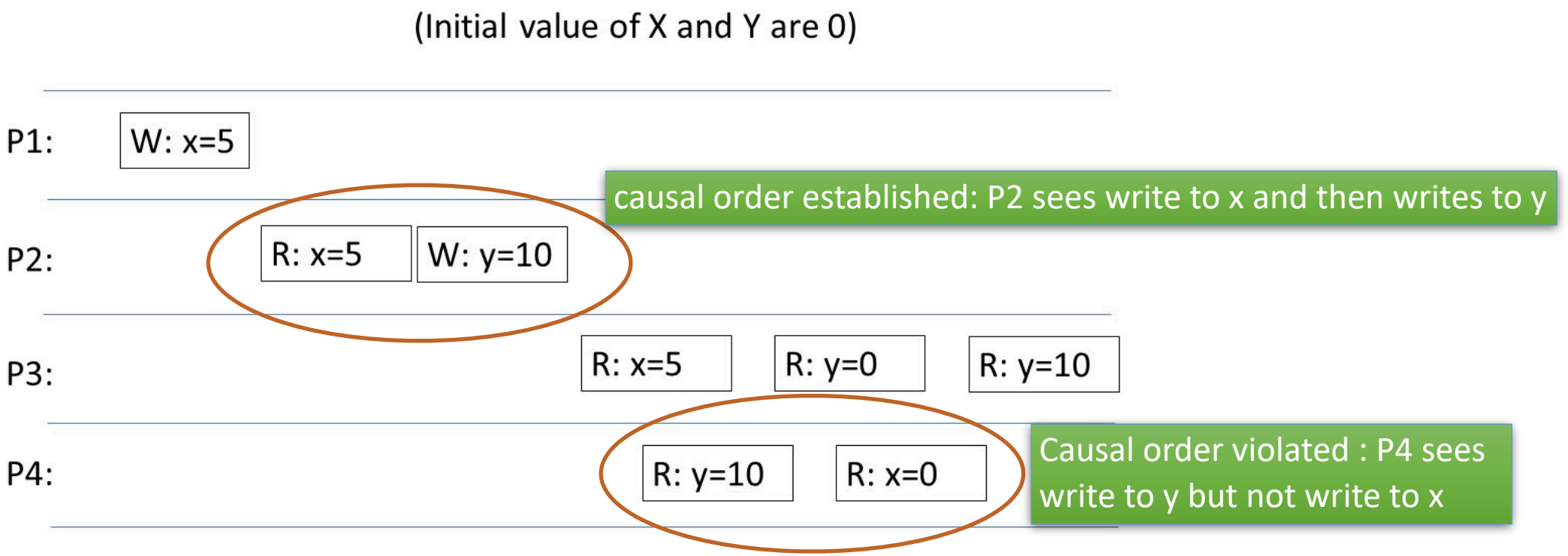


This schedule is causally consistent, but not linearizable or strictly consistent or even sequentially consistent

Cannot be sequentially consistent because P3 and P4 read X and Y values in different order.

But no causal order violated because x and y are not causally related.

Example: Eventually consistent



This schedule is **not** causally consistent, nor linearizable or strictly consistent or sequentially consistent

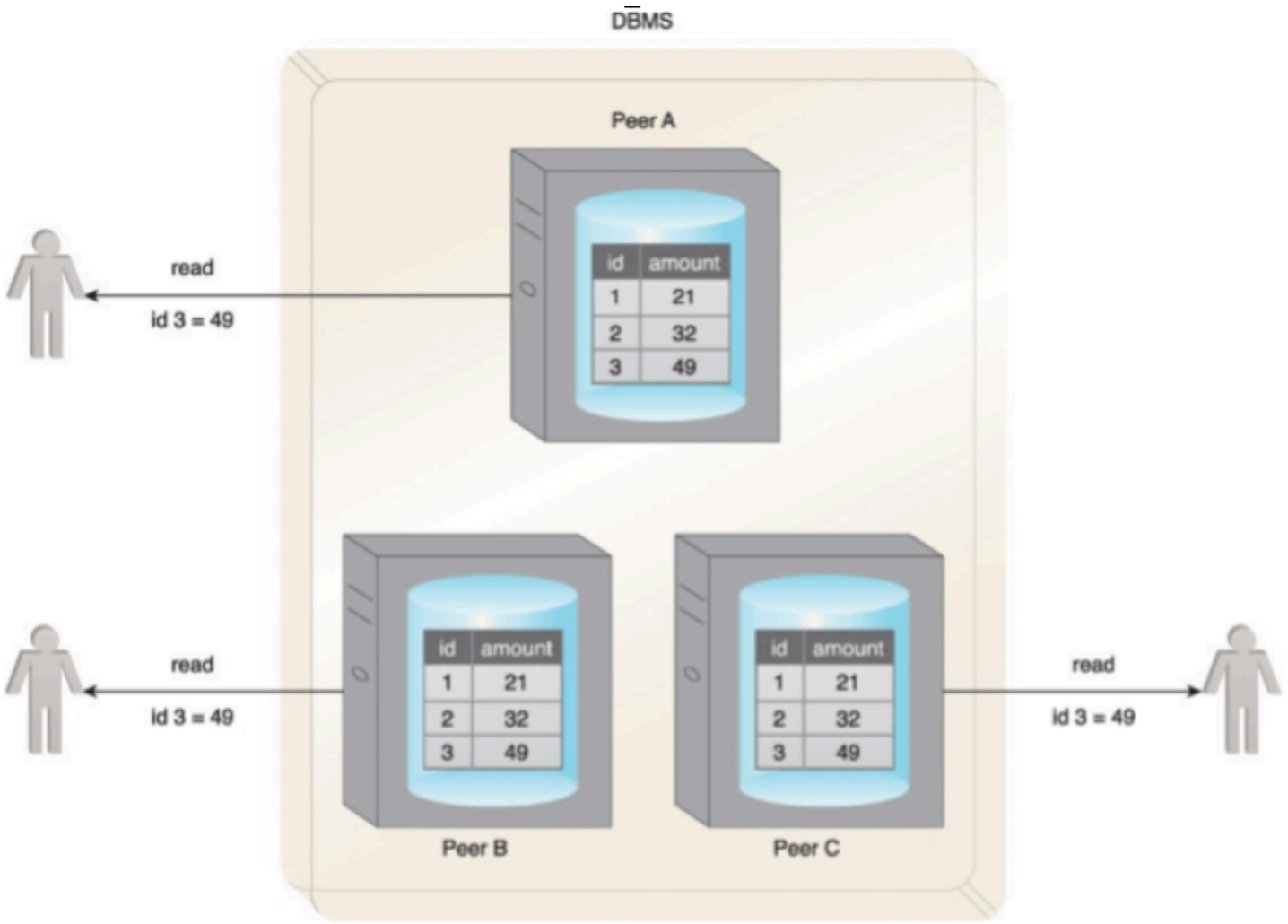
CAP

- Stands for Consistency (C) , Availability (A) and Partition Tolerance (P)
- Triple constraint related to the distributed database systems

- **Consistency**
  - ✓ A read of **a data item** from any node results in same data across multiple nodes
- **Availability**
  - ✓ A read/write request will always be acknowledged in form of success or failure in reasonable time
- **Partition tolerance**
  - ✓ System can continue to function when communication outages split the cluster into multiple silos and can still service read/write requests

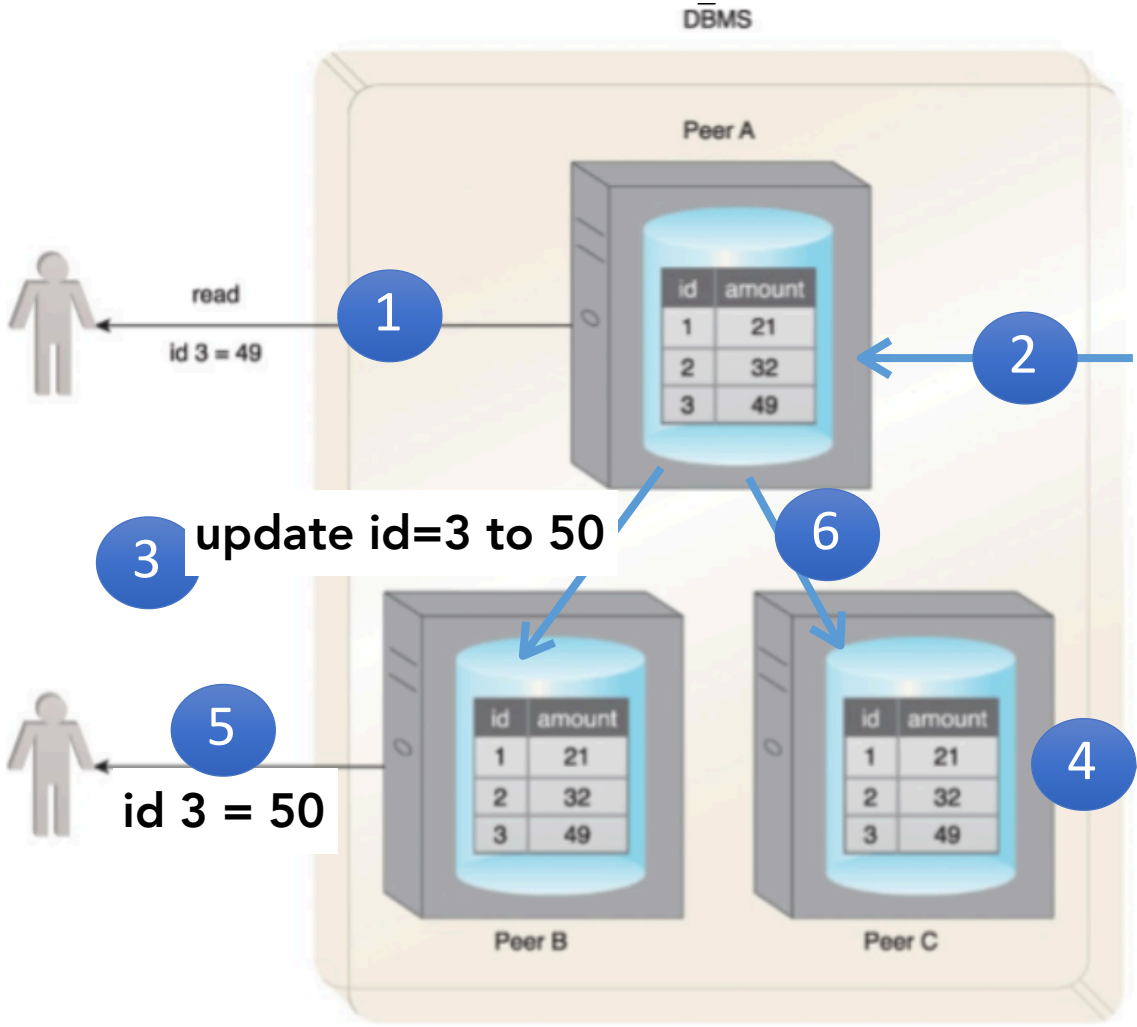
split brain problem

C: Consistency



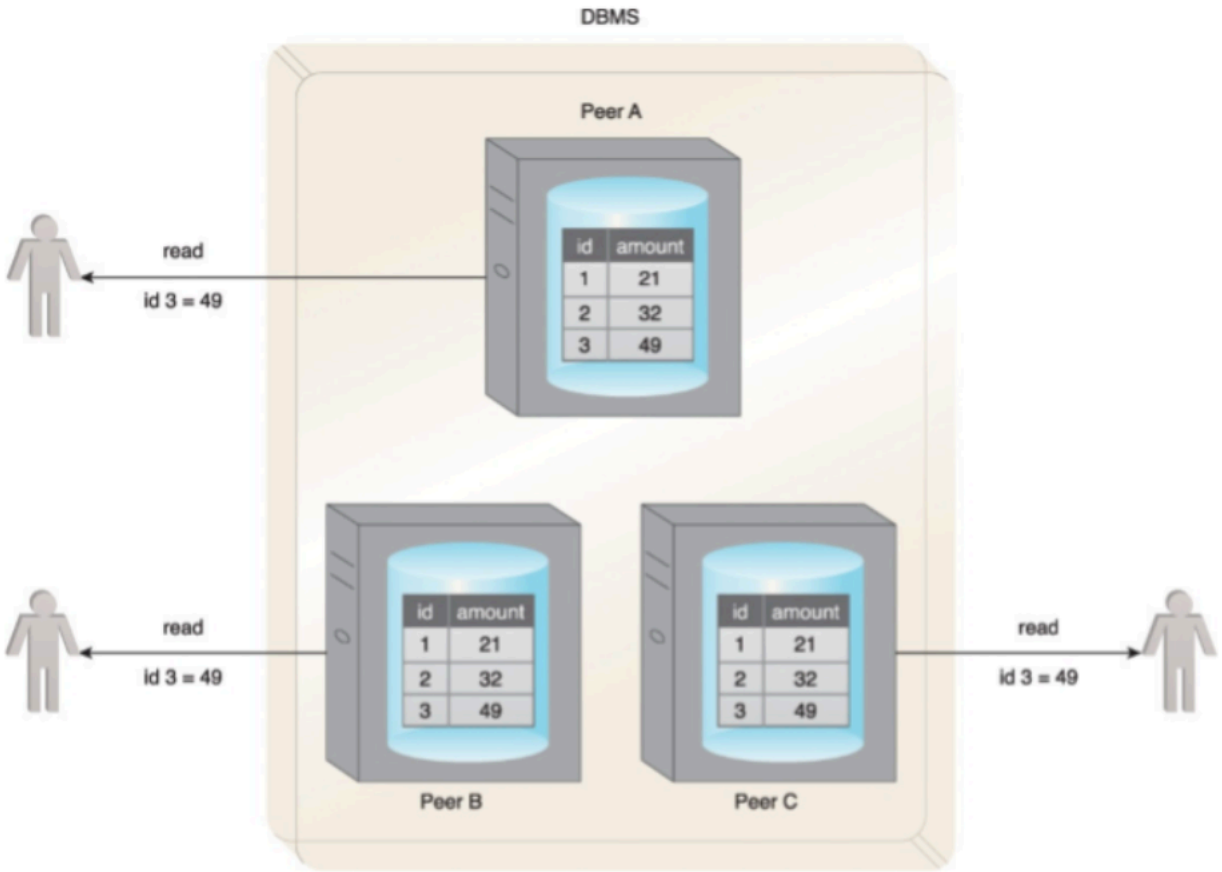
All users get the same value for the amount column even though different replicas are serving the record

When can it be in-consistent



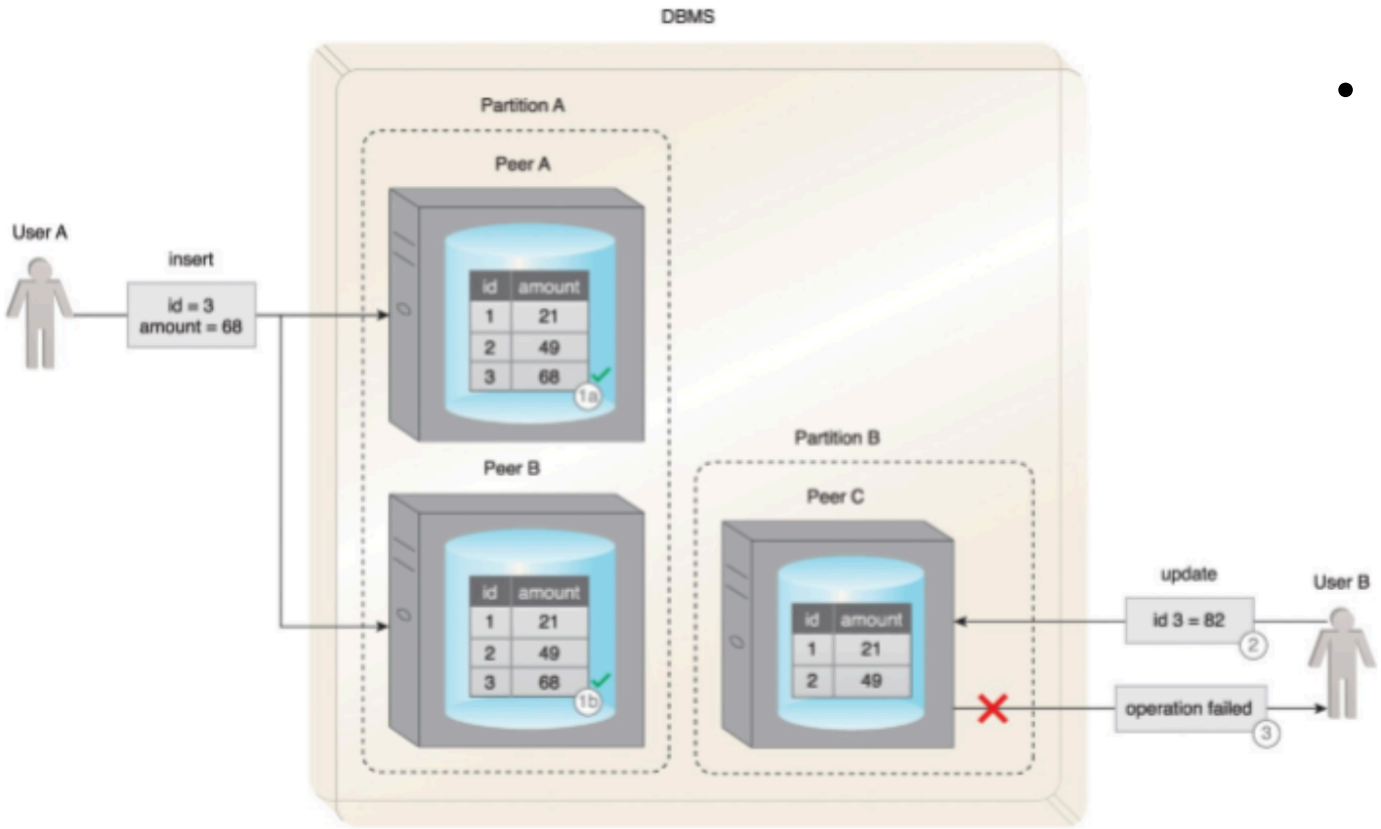
read id=3 is allowed
before peer C is updated even
though write "happens before"
read

A: Availability



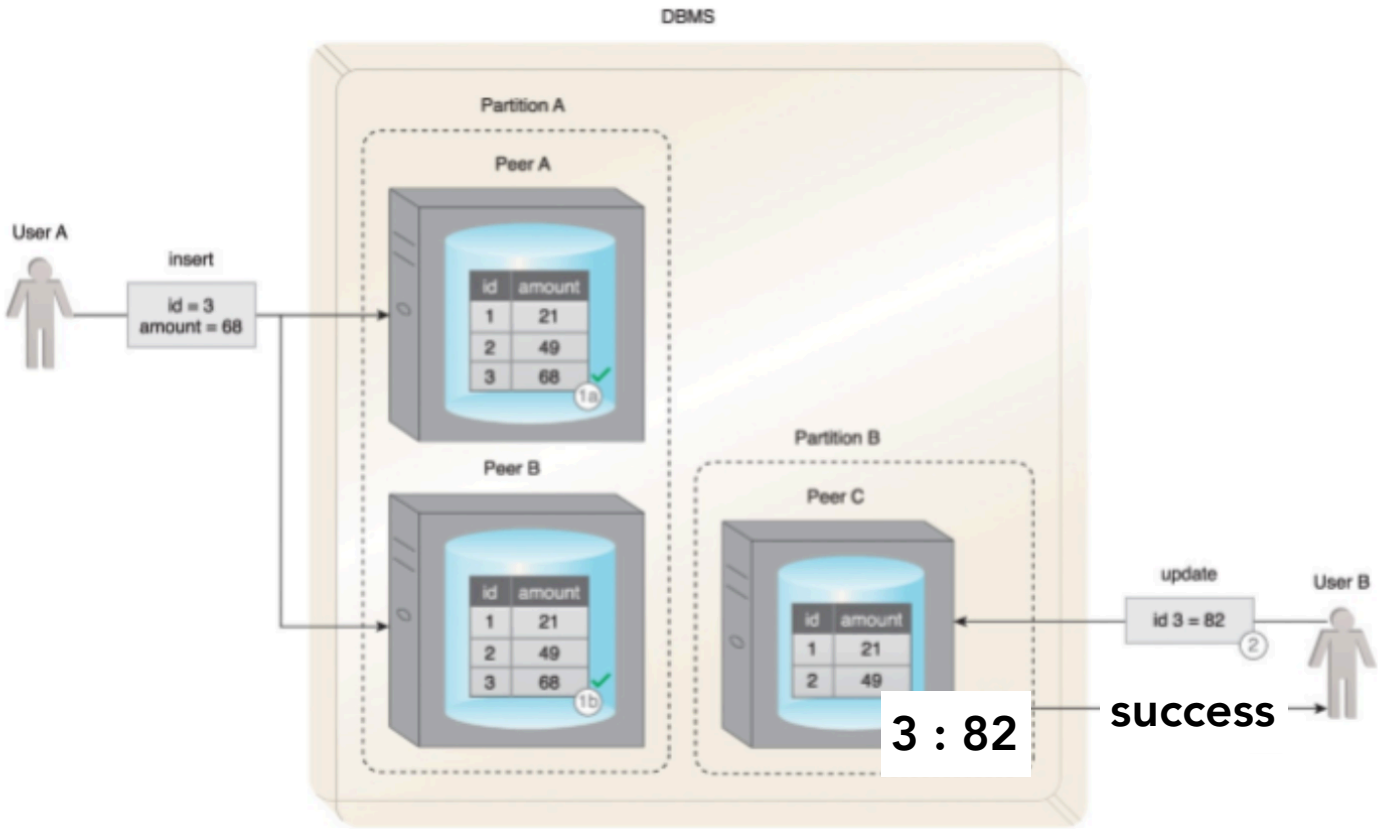
- A request from a user to a peer always responds with a success or failure within a **reasonable** time.
- Say Peer C is disconnected from the network, then it has 2 options
  - **Available**: Respond with a failure when any request is made, or
  - **Unavailable**: Wait for the problem to be fixed before responding, which could be **unreasonably** long and user request may time out

P: Partition tolerance (1)



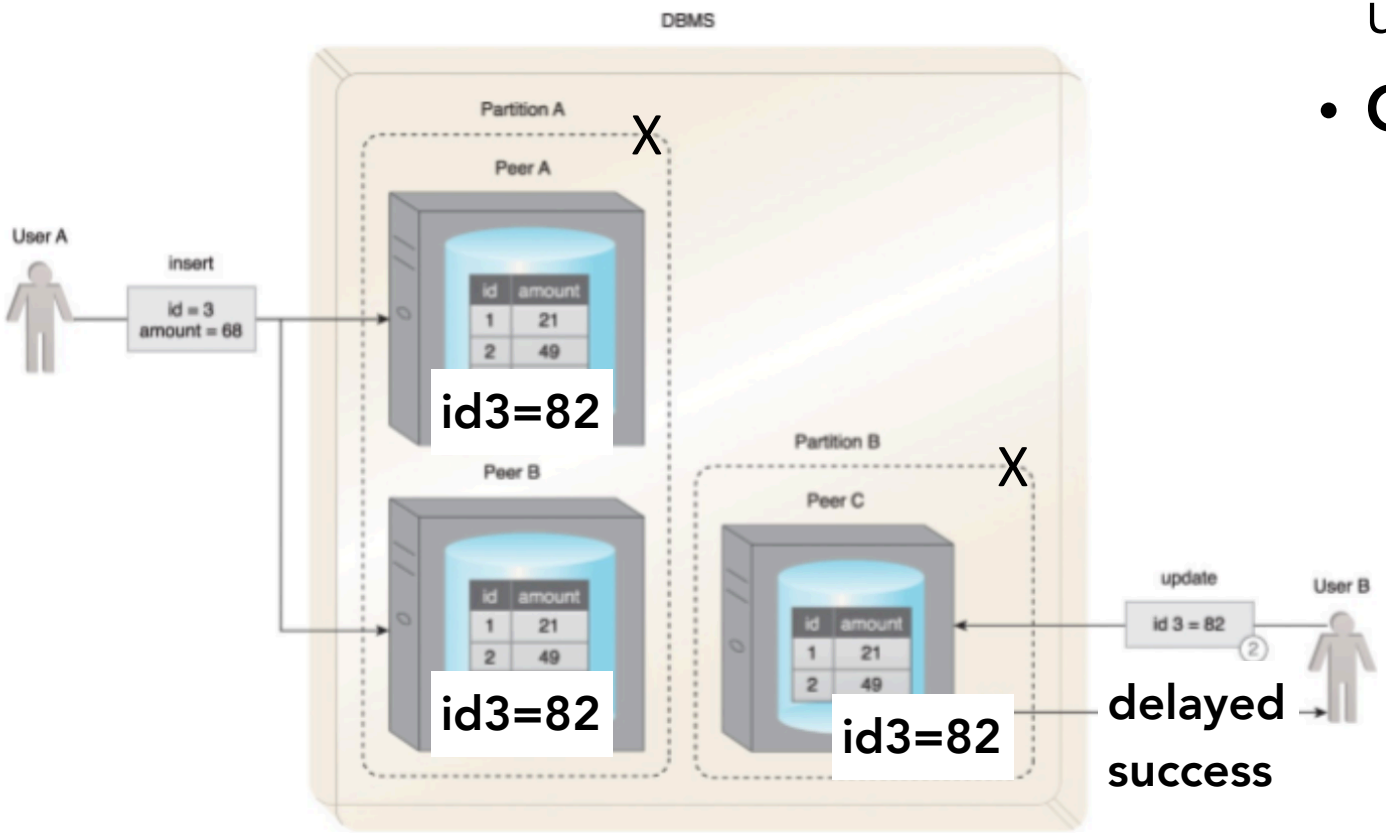
- A network partition happens and user wants to update peer C
- **Option 1**:
  - **Any access by user on peer C leads to failure message response**
  - System is available because it comes back with a response
  - There is consistency because user's update is not processed
  - But there is no partition tolerance
  - **<u>C and A no P</u>**

P: Partition tolerance (2)



- A network partition happens and user wants to update peer C
- **Option 2**:
  - **Peer C records the update by user with success**
  - System is still available and now partition tolerant.
  - But system becomes inconsistent
  - **<u>P and A but no C</u>**

P: Partition tolerance (3)



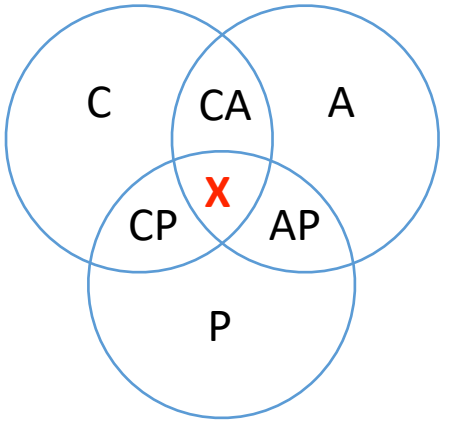
- A network partition happens and user wants to update peer C
- **Option 3**:
  - **User is made to wait till partition is fixed** (could be quite long) and data replicated on all peers before a success message is sent
  - System appears unavailable to user though it is partition tolerant and consistent
  - **P and C but no A**

Topics for today

- Consistency, availability, partition tolerance
- **CAP theorem**
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra
- Big Data lifecycle

CAP Theorem (Brewer's Theorem)

- ***A distributed data system, running over a cluster, can only provide two of the three properties C, A, P but not all.***
- So a system can be
  - CA or AP or CP but cannot support CAP
- In effect, when there is a partition, the system has to decide whether to pick consistency or availability.



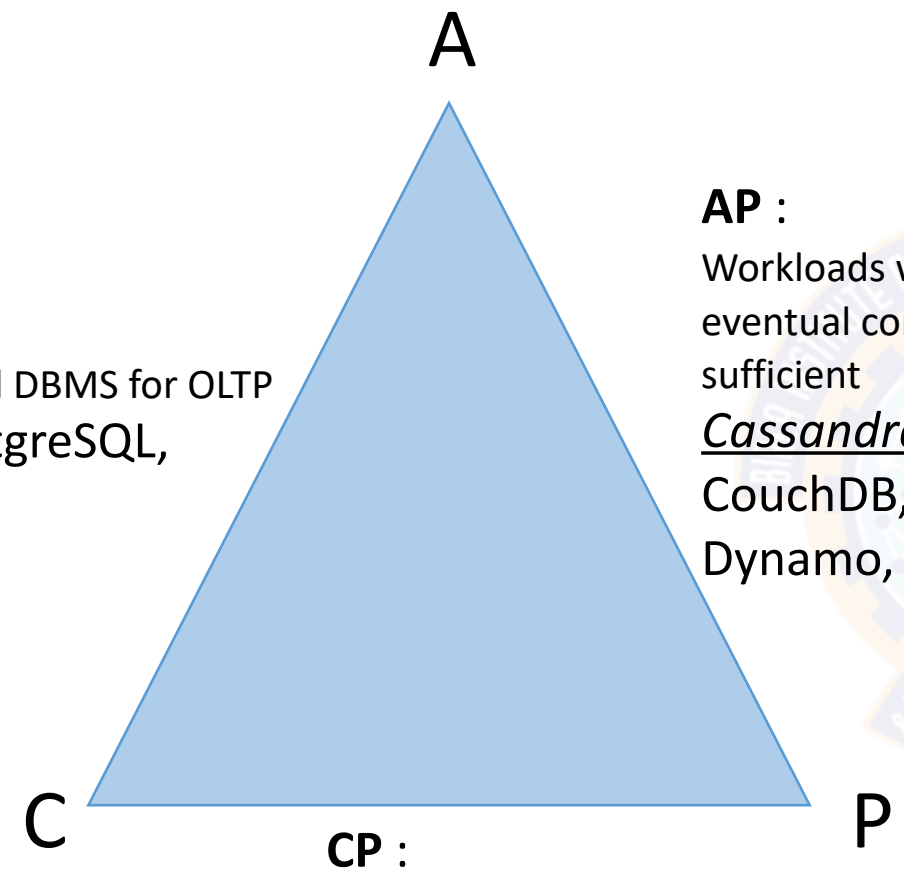
Consistency or Availability choices

- In a distributed database,
  - Scalability and fault tolerance can be improved through additional nodes, although this puts challenges on maintaining consistency (C).
  - The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
    - May have to update all replicas before sending success to client . so longer takes time and system may not be available during this period to service reads on same data item.

- Large scale distributed systems cannot be 100% partition tolerant (P).
  - Although communication outages are rare and temporary, partition tolerance (P) must always be supported by distributed database

- Therefore, **CAP is generally a choice between choosing either CP or AP**
- *Traditional RDBMS systems mainly provide CA for single data items and then on top of that provide ACID for transactions that touch multiple data items.*

Database options



CA :
Traditional DBMS for OLTP
e.g. PostgreSQL, MySQL

AP :
Workloads where eventual consistency is sufficient
Cassandra, CouchDB, Dynamo,

CP :
Can provide consistency with scale.
HBase, *MongoDB*, Redis

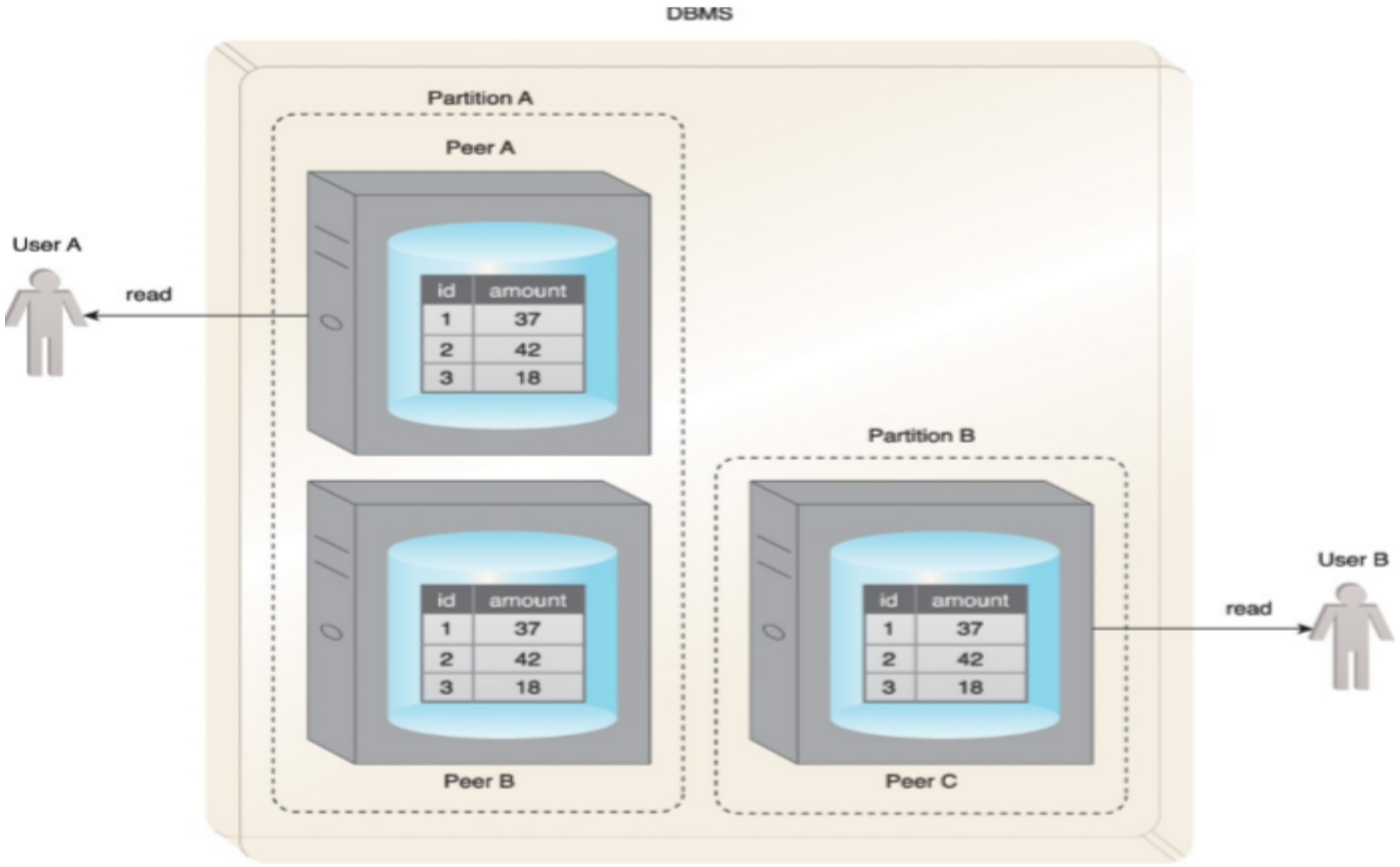
- Different design choices are made by Big Data DBs
- Faults are likely to happen in large scale systems
- Provides flexibility depending on use case to choose C, A, P behavior mainly around consistency semantics when there are faults

BASE

- BASE is a database design principle based on CAP theorem
- Leveraged by AP database systems that use distributed technology
- Stands for
  - ✓ Basically Available (BA)
  - ✓ Soft state (S)
  - ✓ Eventual consistency (E)

- It favours availability over consistency
- Soft approach towards consistency allows to **serve multiple clients without any latency albeit serving inconsistent results**
- **Not useful for transactional systems** where lack of consistency is concern
- Useful for **write-heavy workloads where reads need not be consistent in real-time**, e.g. social media applications, monitoring data for non-real-time analysis etc.

BASE – Basically Available



User A and User B receive data despite the database being partitioned by a network failure.

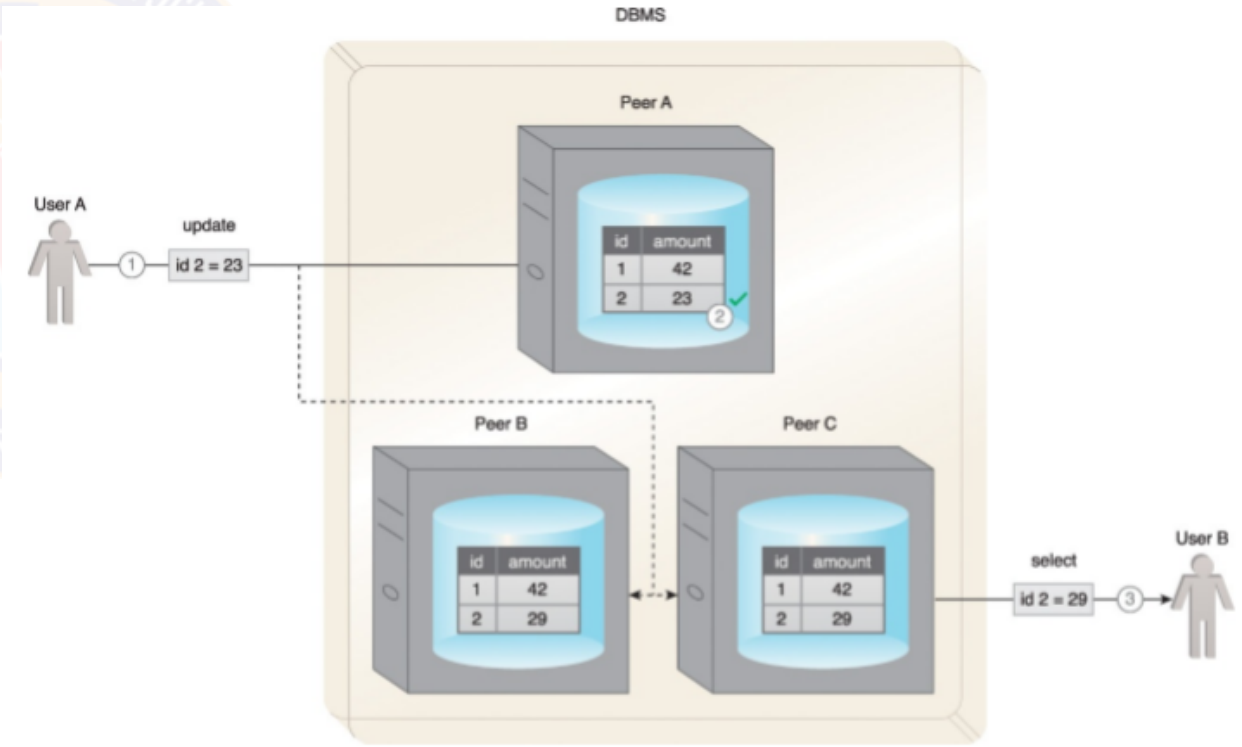
Database will always acknowledge a client's request, either in form of requested data or a failure notification

BASE – Soft State

- Database may be in inconsistent state when data is read, thus results may change if the same data is requested again
  - ✓ Because data could be uploaded for consistency, even though no user has written to the database between two reads

- User A updated record on node A
- Before the other nodes are updated,

  User B requests same record from C
- Database is now in a soft state and
- Stale data is returned to User B



An example of the soft state property of BASE is shown here.

BASE – Eventual Consistency

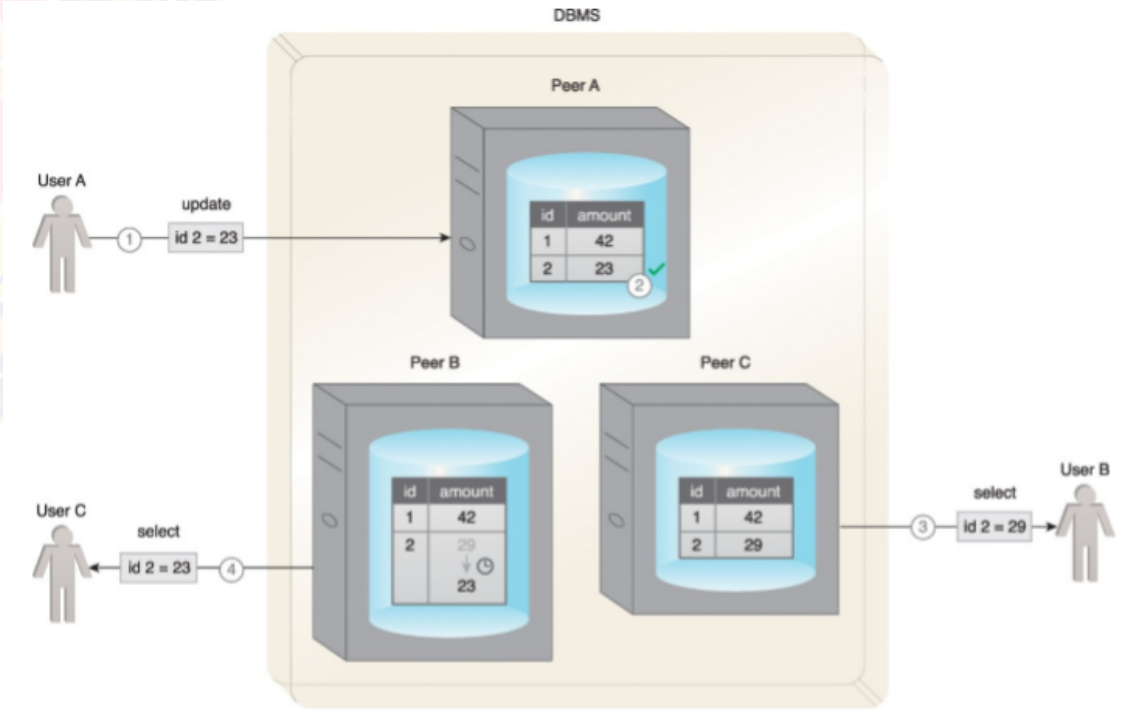
- State in which reads by different clients, immediately following a write to database, may not return consistent results
- Database only attains consistency once the changes have been propagated to all nodes
- While database is in the process of attaining the state of eventual consistency, it will be in a soft state

✓ User A updates a record

✓ Record only gets updated at node A, but before other peers can be updated, User B requests data

✓ Database is now in soft state, stale data is returned to User B from peer C

✓ **Consistency is eventually attained, User C gets correct value**

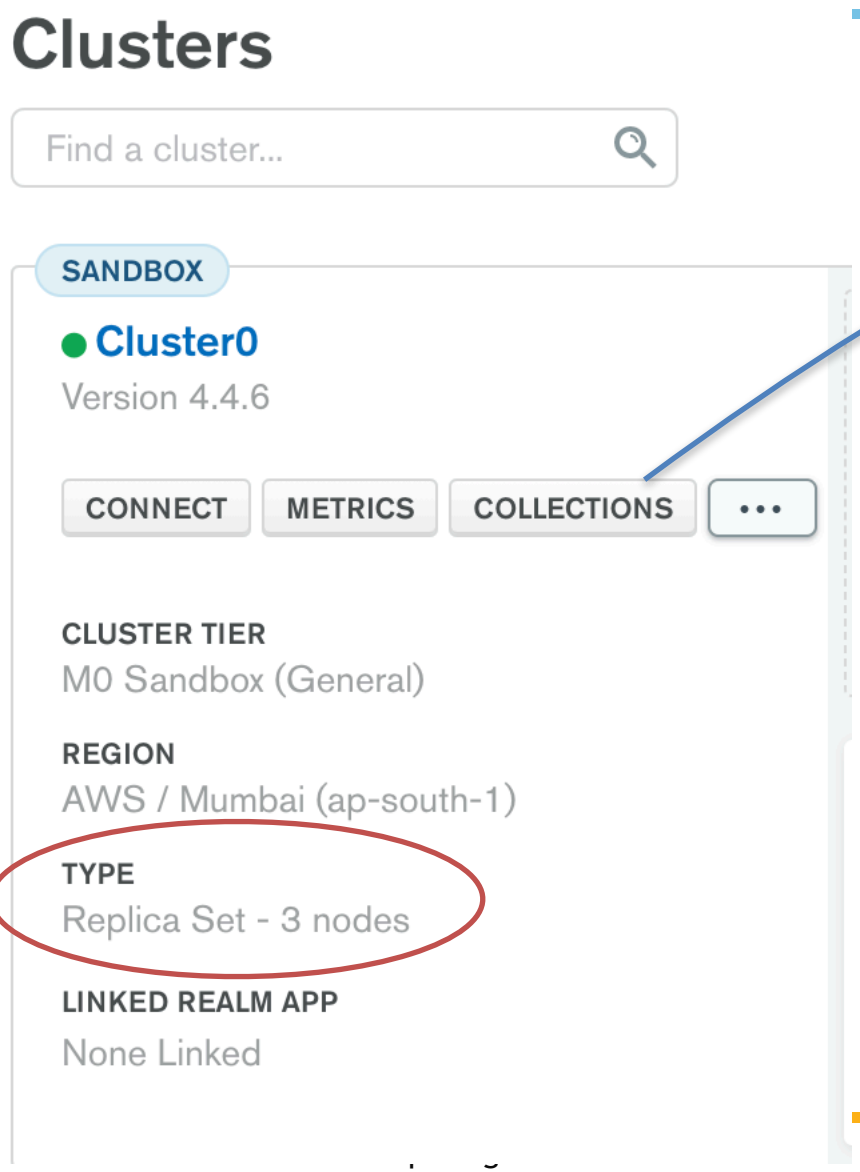


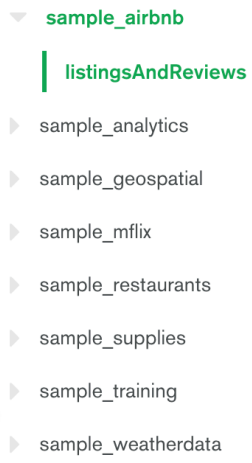
An example of the eventual consistency property of BASE.

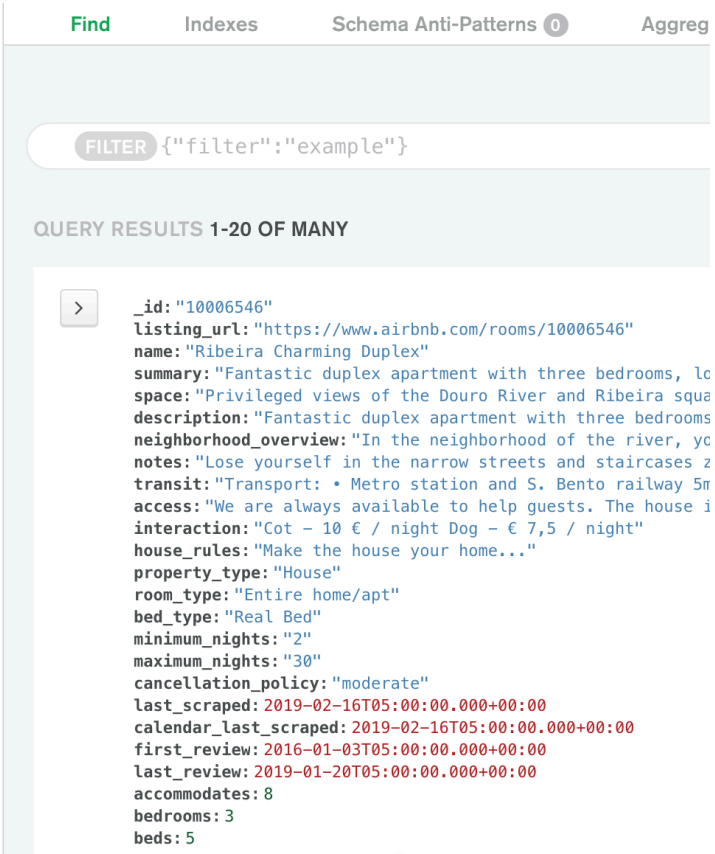
Topics for today

- Consistency, availability, partition tolerance
- CAP theorem
- **Example BigData store options based on CAP requirement**
  - **MongoDB**
  - **Cassandra**
- Big Data lifecycle

cloud.mongodb.com





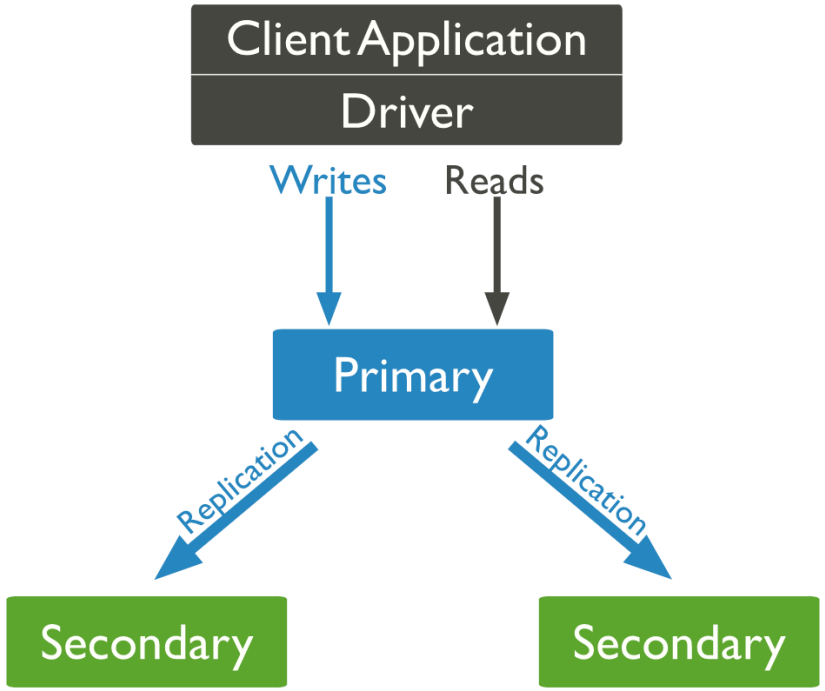


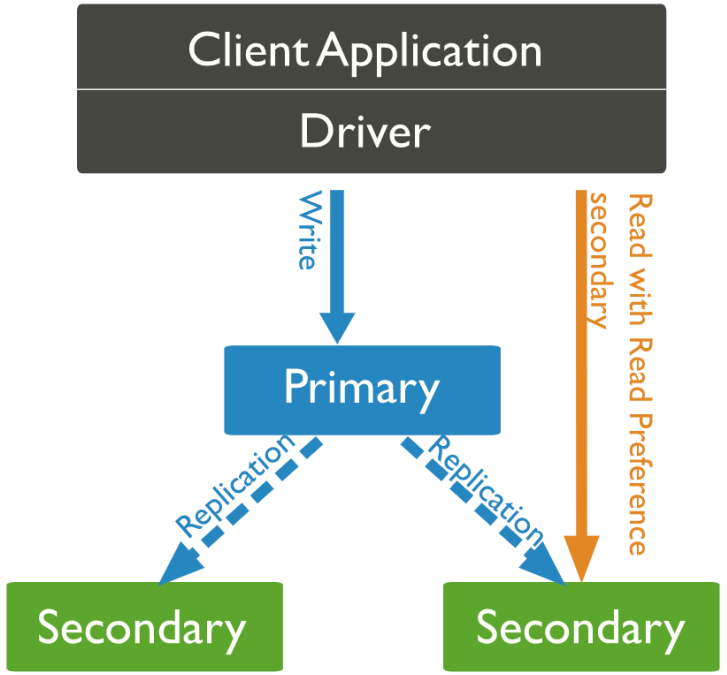
Get me top 10 beach front homes

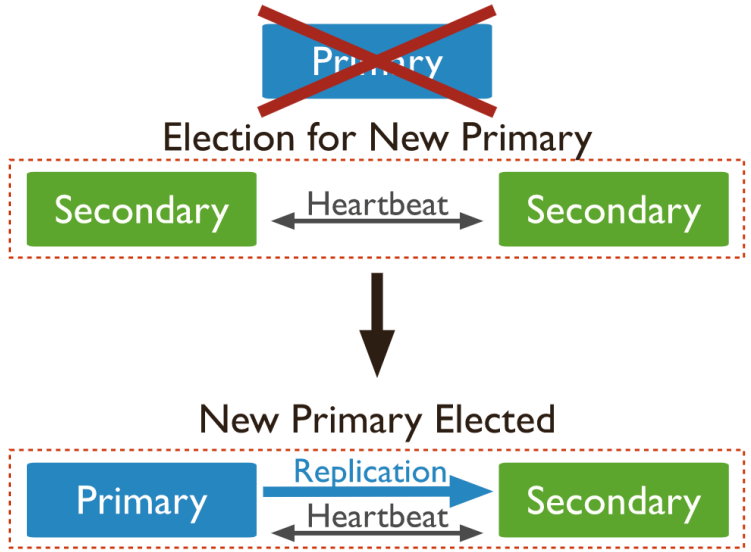
```
# sort search results by score
s = collection.aggregate([

                { "$match": { "$text": { "$search": "beach front" } } },
                { "$project": { "name": 1, "_id": 0, "score": { "$meta": "textScore" } } },
                { "$match": { "score": { "$gt": 1.0 } } },
                { "$sort": { "score": -1}},
                { "$limit": 10}
        ])
```

MongoDB

- Document oriented DB
- Various read and write choices for flexible consistency tradeoff with scale / performance and durability
- Automatic primary re-election on primary failure and/or network partition

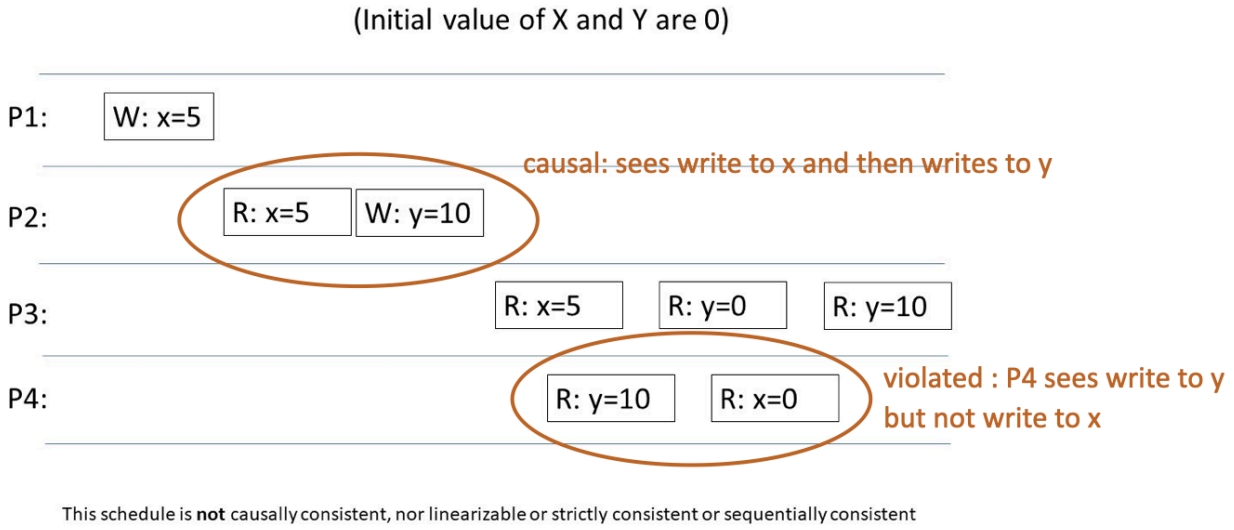






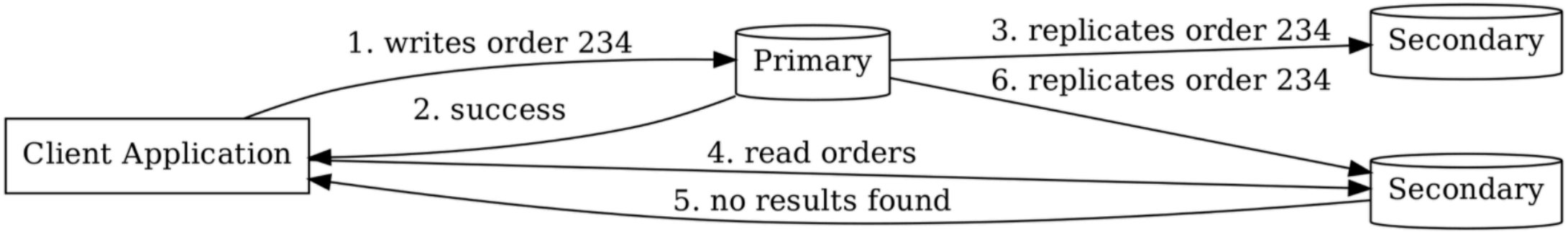
What is Causal Consistency (recap)

Read your writes	Read operations reflect the results of write operations that precede them.
Monotonic reads	Read operations do not return results that correspond to an earlier state of the data than a preceding read operation.
Monotonic writes	Write operations that must precede other writes are executed before those other writes.
Writes follow reads	Write operations that must occur after read operations are executed after those read operations.

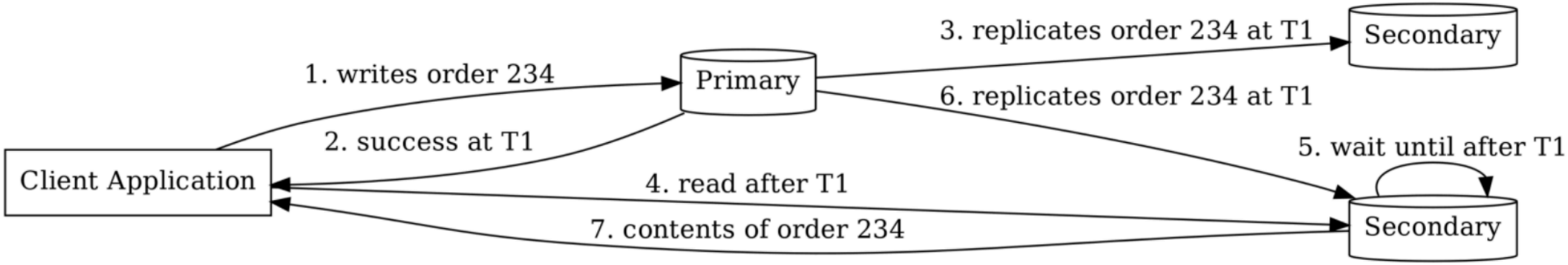


Example in MongoDB

- Case 1 : No causal consistency



- Case 2: Causal consistency by making read to secondary wait



https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

MongoDB “read concerns”

- **local** :
  - Client reads primary replica
  - Client reads from secondary in causally consistent sessions
- **available**:
  - Read on secondary but causal consistency not required
- **majority** :
  - If client wants to read what majority of nodes have. Best option for fault tolerance and durability.
- **linearizable** :
  - If client wants to read what has been written to majority of nodes before the read started.
  - Has to be read on primary
  - Only single document can be read

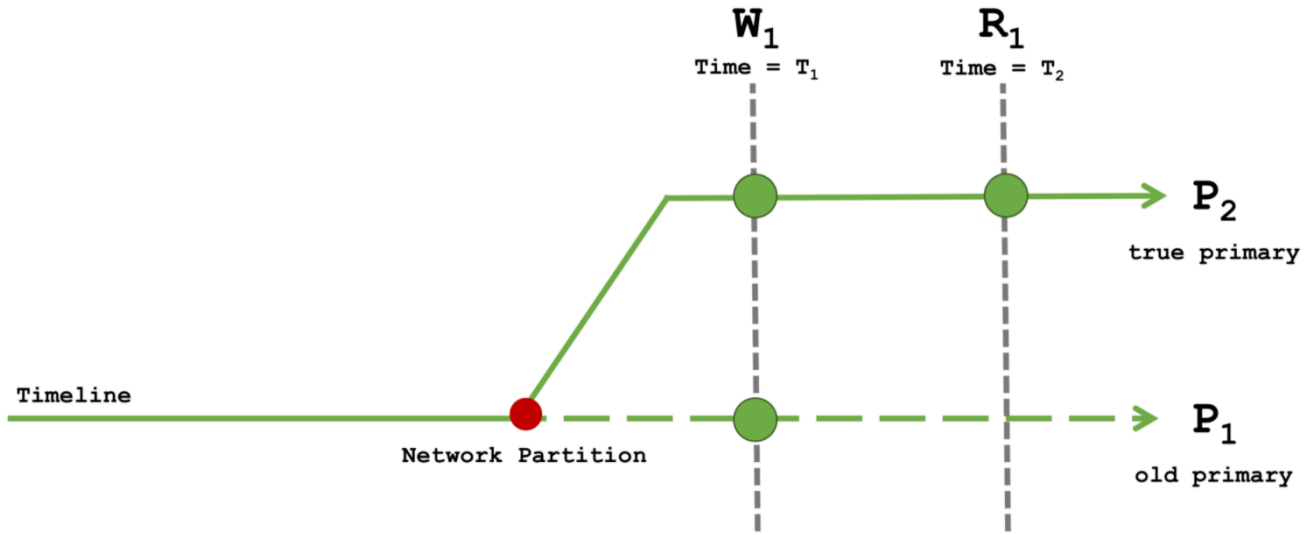
https://docs.mongodb.com/v3.4/core/read-preference-mechanics/

MongoDB “write concerns”

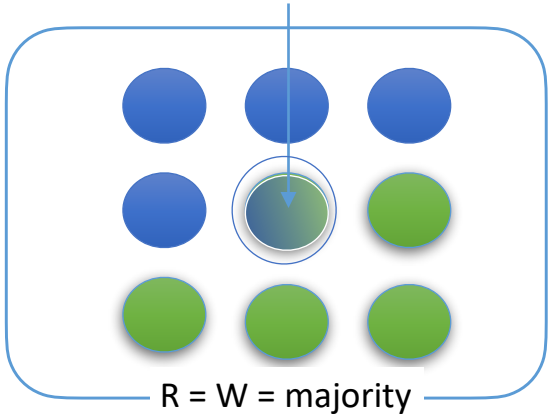
- how many replicas should ack
  - 1 - primary only
  - 0 - none
  - n - how many including primary
  - majority - a majority of nodes (preferred for durability)
- journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)
- timeout for write operation

https://docs.mongodb.com/manual/reference/write-concern/

Consistency scenarios - causally consistent and durable



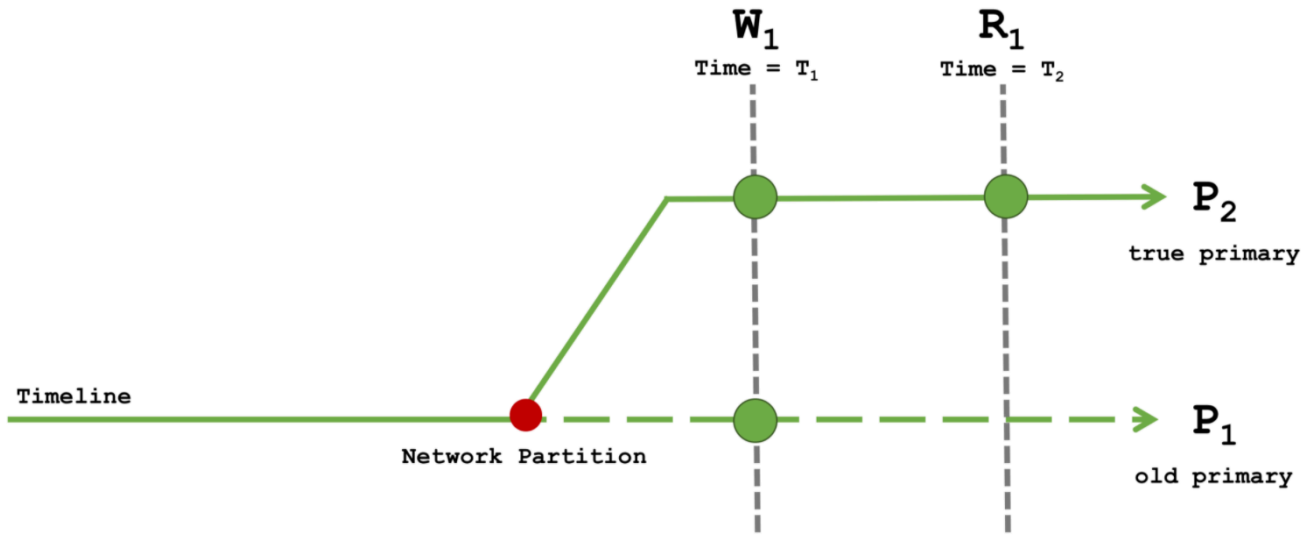
Read latest written value from common node



- **read=majority, write=majority**
- W1 and R1 for P1 will fail and will succeed in P2
- So causally consistent, durable even with network partition sacrificing performance
- ***Example*: Used in critical transaction oriented applications, e.g. stock trading**

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

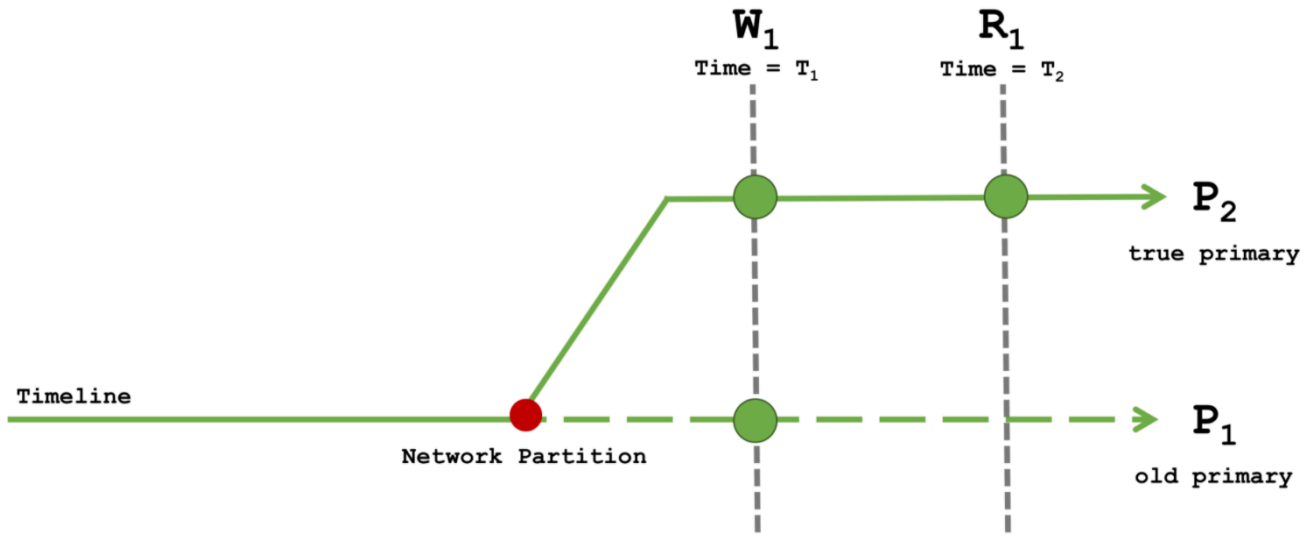
Consistency scenarios - causally consistent but not durable



- **read=majority, write=1**
- W1 may succeed on P1 and P2. R1 will succeed only on P2. W1 on P1 may roll back.
- So causally consistent but not durable with network partition. Fast writes, slower reads.
- ***Example*: Twitter - a post may disappear but if on refresh you see it then it should be durable, else repost.**

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

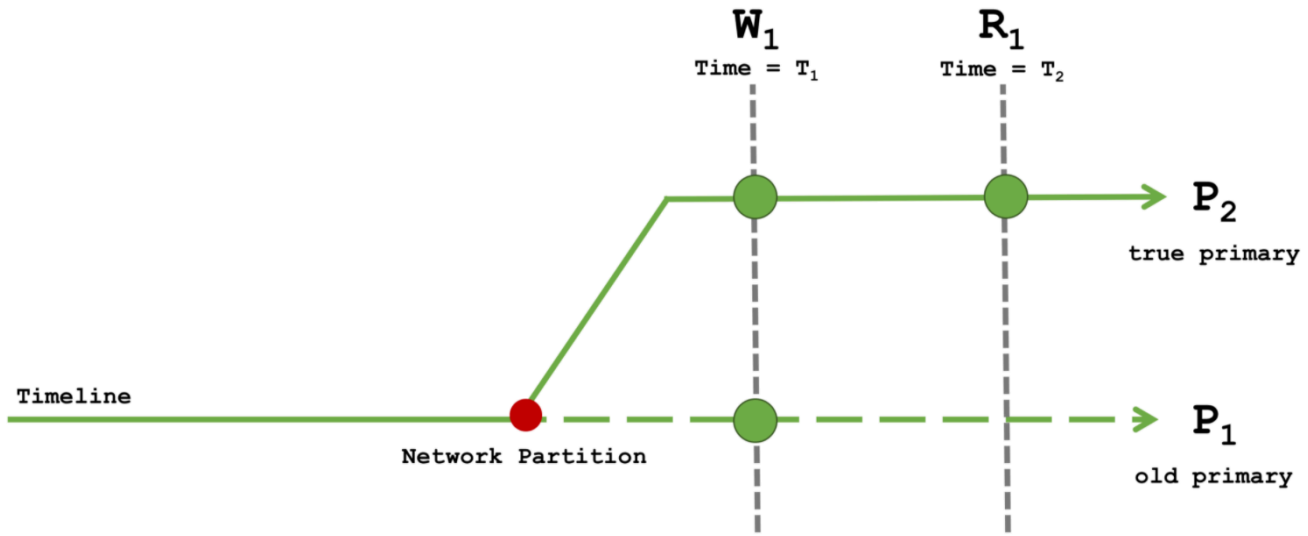
Consistency scenarios - eventual consistency with durable writes



- **read=local, write=majority**
- W1 will succeed only for P1 and reads may not succeed to see the last write. Slow durable writes and fast non-causal reads.
- ***Example:*** **Review site where write should be durable but reads don't need causal guarantee as long as it appears some time (eventual consistency).**

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

Consistency scenarios - eventual consistency but no durability



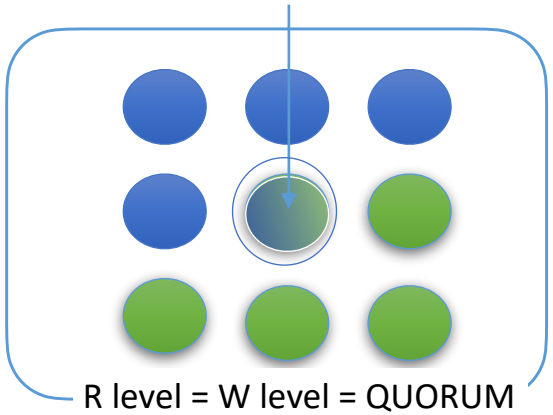
- **read=local, write=1**
- Same as previous scenario and not writes are also not durable and may be rolled back.
- ***Example:*** **Real-time sensor data feed that needs fast writes to keep up with the rate and reads should get as much recent real-time data as possible. Data may be dropped on failures.**

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

Cassandra

- Key-value store with columnar storage
- **No primary replica** - high partition tolerance and availability and levels of consistency
- Support for light transactions with "linearizable consistency"
- A Read or Write operation can pick a consistency level
  - ONE, TWO, THREE, ALL - 1,2,3 or all replicas respectively have to ack
  - QUORUM - majority have to ack
  - LOCAL_QUORUM - majority within same datacenter have to ack
  - …
- **For "causal consistency" pick Read consistency level = Write consistency level = QUORUM**
- Why ? At least one node will be common between write and read set so a read will get the last write of a data item
- What happens if read and write use LOCAL_QUORUM ?
- **If no overlap read and write sets then "Eventual consistency"**

Read latest written value from common node



https://cassandra.apache.org/doc/latest/architecture/dynamo.html
https://cassandra.apache.org/doc/latest/architecture/guarantees.html#

Topics for today

- Consistency, availability, partition tolerance
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra
- **Big Data lifecycle**

Big Data Analytics Lifecycle

- Big Data analysis differs from traditional data analysis primarily
  - ✓ due to the volume, velocity and variety characteristics of the data being processes

- A step-by-step methodology is needed to organize the activities and tasks involved with
  - ✓ Acquiring
  - ✓ Processing
  - ✓ Analyzing
  - ✓ Repurposing data

- Explore a specific data analytics lifecycle that organizes and manages the tasks and activities associated with the analysis of Big Data

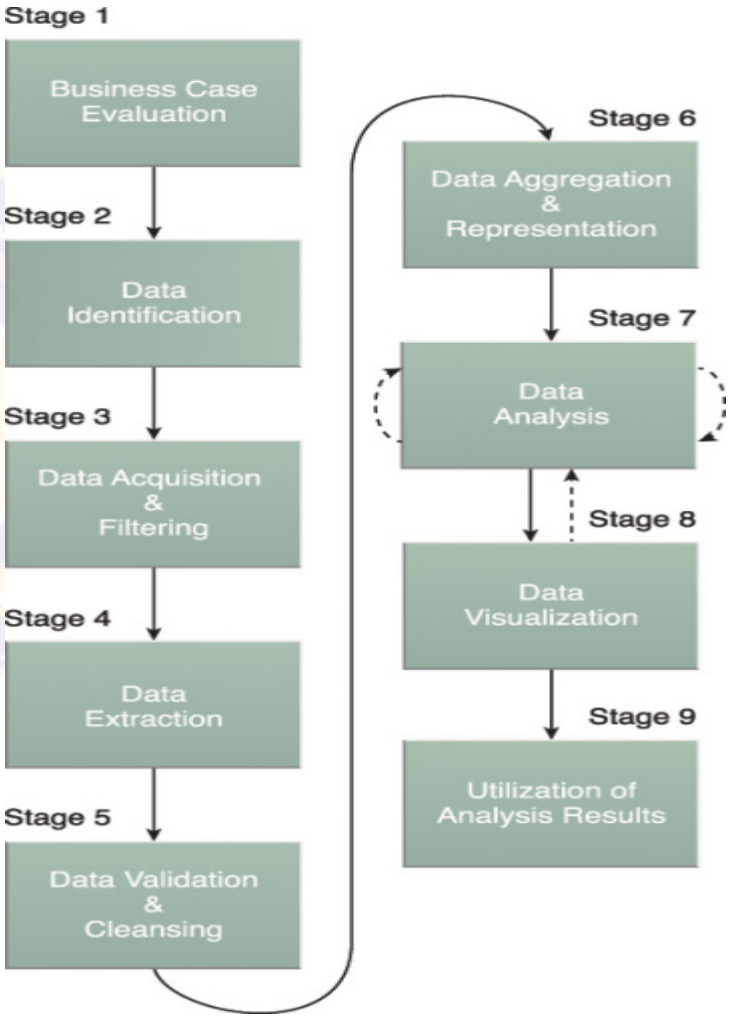
Example

1. A market research firm creates and runs surveys to understand specific market segments for their client, e.g. consumer electronics - LED TVs
2. These surveys contain questions that have structured : numeric, boolean, categorical, grade as well as unstructured : free form text answers
3. A survey is rolled out to many users with various demographic attributes. The list of survey users could be provided by the client and/or MR firm
4. The results are collected and analyzed for business insights often using multiple tools and analysis techniques
5. The insights are curated and shared to create a presentation for the client of the MR firm
6. The client makes critical business decisions about their product based on the survey results

Big Data Analytics Lifecycle

Stages

1. Business Case Evaluation
2. Data Identification
3. Data Acquisition & Filtering
4. Data Extraction
5. Data Validation & Cleansing
6. Data Aggregation & Representation
7. Data Analysis
8. Data Visualization
9. Utilization of Analysis Results



1. Business Case Evaluation

- Based on business requirements determine whether the business problems being addressed is really Big Data problem
  - ✓ A business problem needs to be directly related to one or more of the Big Data characteristics of volume, velocity, or variety.

High volume
Unstructured data

- Must begin with a well-defined business case that presents a clear understanding of the
  - ✓ justification
  - ✓ motivation
  - ✓ goals of carrying out the analysis.

Find market fit
for new product

- A business case should be created, assessed and approved prior to proceeding with the actual hands-on analysis tasks.

What are the business questions ?
Define thresholds
on survey stats

- Helps decision-makers to
  - ✓ understand the **business resources** that will need to be utilized
  - ✓ Identify which **business challenges** the analysis will tackle.
  - ✓ Identify **KPIs** can help determine assessment criteria and guidance for the evaluation of the analytic results

2. Data Identification

- Main objective is to identify the datasets required for the analysis project and their sources
  - ✓ Wider variety of data sources may increase the probability of finding hidden patterns and correlations.
    - ✓ Caution: Too much data variety can also confuse - overfitting problem.
  - ✓ The required datasets and their sources can be internal and/or external to the enterprise.

Identify respondents
Demographics
What questions to ask
Do we need other surveys

- For internal datasets
  - ✓ A list of available datasets from internal sources, such as data marts and operational systems, are typically compiled and verified for data required

- For external datasets
  - ✓ A list of possible third-party data providers, such as data markets and publicly available datasets needs to be compiled
  - ✓ Data may be embedded within blogs or other types of content-based web sites, automated tools needs to be used to extract it

3. Data Acquisition & Filtering

- The data is gathered from all of the data sources that were identified during the last stage

- The acquired data is then looked upon for
  - ✓ filtering / removal of corrupt data
  - ✓ removal of unusable data for analysis

Clean bad data, e.g. empty responses
Junk text inputs
Filter a subset if we don't need to look at all attributes, all demographics

- In many cases involving unstructured external data, some or most of the acquired data may be irrelevant (noise) and can be discarded as part of the filtering process.

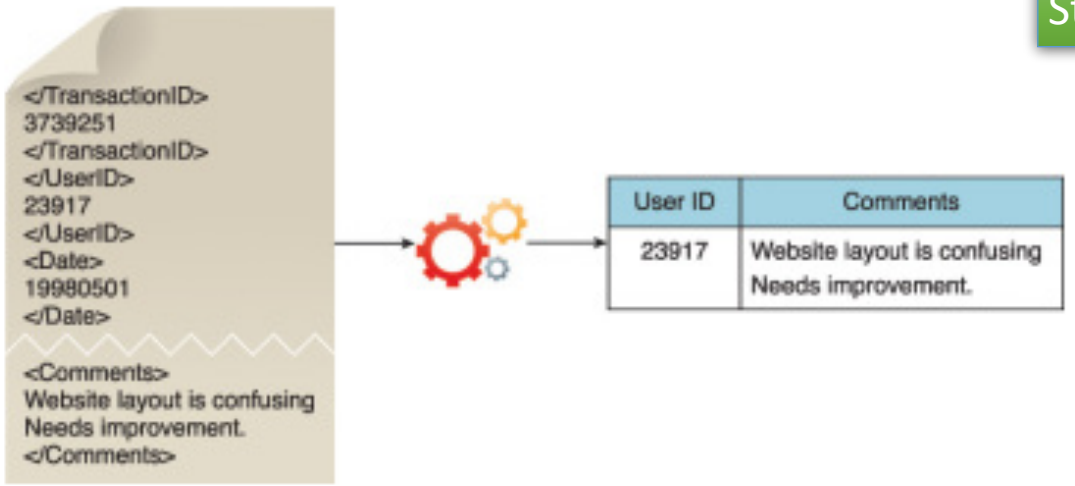
- "Corrupt" data can include records with missing or nonsensical values or invalid data types
  - ✓ Advisable to store a verbatim copy of the original dataset before proceeding with the filtering

- Data needs to be persisted once it gets generated or enters the enterprise boundary
  - ✓ For batch analytics, this data is persisted to disk prior to analysis
  - ✓ For real-time analytics, the data is analyzed first and then persisted to disk

4. Data Extraction

- Dedicated to extracting data and transforming it into a format that the underlying Big Data solution can use for the purpose of the data analysis
- The extent of extraction and transformation required depends on the types of analytics and capabilities of the Big Data solution.

Structure the unstructured responses



Comments and user IDs are extracted from an XML document.

5. Data Validation & Cleansing

Validate survey responses
Contradictory answers
Identify population skews, e.g. responses have inherent gender bias so no point in making a gender based analysis
Codify certain columns for easier analysis

- Invalid data can skew and falsify analysis results

- Data input into Big Data analyses can be unstructured without any indication of validity
  - ✓ Complexity can further make it difficult to arrive at a set of suitable validation constraints
  - ✓ Dedicated stage is required to establish complex validation rules and removing any known invalid data.

- Big Data solutions often receive redundant data across different datasets.
  - ✓ This can be exploited to explore interconnected datasets in order to
    - assemble validation parameters
    - fill in missing valid data

- For batch analytics, data validation and cleansing can be achieved via an offline ETL operation
- For real-time analytics, a more complex in-memory system is required to validate and cleanse the data as it arrives from the source

6. Data Aggregation & Representation

- Dedicated to integrating multiple datasets together to arrive at a unified view
  - ✓ Needs to merge together the data spread across multiple datasets through a common field
  - ✓ Needs reconciliation of data coming from different sources
  - ✓ Needs to identify the dataset representing the correct value needs to be determined.

Final joined data set (e.g. current with old survey or 3rd party demographics data) with certain aggregations done for downstream analysis

- Can be complicated because of :
  - ✓ Data Structure – Although the data format may be the same, the data model may be different
  - ✓ Semantics – A value that is labeled differently in two different datasets may mean the same thing, for example "surname" and "last name."

- The large volumes makes data aggregation a time and effort-intensive operation
  - ✓ Reconciling these differences can require complex logic that is executed automatically without the need for human intervention

- Future data analysis requirements need to be considered during this stage to help foster data reusability.

7. Data Analysis

- Dedicated to carrying out the actual analysis task, which typically involves one or more types of analytics
  - ✓ Can be iterative in nature, especially if the data analysis is exploratory
  - ✓ Analysis is repeated until the appropriate pattern or correlation is uncovered

- Depending on the type of analytic result required
  - ✓ Can be as simple as querying a dataset to compute an aggregation for comparison
  - ✓ Can be as challenging as combining data mining and complex statistical analysis techniques to discover patterns and anomalies or to generate a statistical or mathematical model to depict relationships between variables.

Various types of descriptive / predictive analysis on survey data to understand market fit for new product. Writing SQL on data and create charts. Build models on the data for hypothesis testing, prediction.

7. Confirmatory / Exploratory data analysis

- Confirmatory data analysis
  - ✓ A deductive approach where the cause of the phenomenon being investigated is proposed beforehand - a hypothesis
  - ✓ Data is then analyzed to prove or disprove the hypothesis and provide definitive answers to specific questions
  - ✓ Data sampling techniques are typically used
  - ✓ Unexpected findings or anomalies are usually ignored since a predetermined cause was assumed

- Exploratory data analysis
  - ✓ Inductive approach that is closely associated with data mining
  - ✓ No hypothesis or predetermined assumptions are generated
  - ✓ Data is explored through analysis to develop an understanding of the cause of the phenomenon
  - ✓ May not provide definitive answers
  - ✓ Provides a general direction that can facilitate the discovery of patterns or anomalies

Confirm findings or exception cases in the survey data through adhoc exploration. E.g. in how many cases young males like feature X but don't like feature Y.

8. Data Visualization

Visual results will need to be shared with the stakeholders for the new product launch. e.g. show top features that appeal to each segment of target product user (gender, age group).

- Dedicated to using data visualization techniques and tools to graphically communicate the analysis results for effective interpretation by business users
  - ✓ The ability to analyze massive amounts of data and find useful insights carries little value if the only ones that can interpret the results are the analysts.
  - ✓ Business users need to be able to understand the results in order to obtain value from the analysis and subsequently have the ability to provide feedback

- Provide users with the ability to perform visual analysis, allowing for the discovery of answers to questions that users have not yet even formulated
  - ✓ A method of drilling down to comparatively simple statistics is crucial, in order for users to understand how the rolled up or aggregated results were generated

- Important to use the most suitable visualization technique by keeping the business domain in context
  - ✓ Interpretation of result can vary based on the visualization shown

9. Utilization of Analysis Results

- Dedicated to determining how and where processed analysis data can be further leveraged
  - ✓ Apart from dashboards

- Possible for the analysis results to produce "models" that
  - ✓ encapsulate new insights and understandings about the nature of the patterns and relationships that exist within the data that was analyzed
  - ✓ May look like a mathematical equation or a set of rules
  - ✓ Can be used to improve business process logic and application system logic

Surveys and analysis output, models built are reusable assets
Reports created to capture the insights.

9. Utilization of Analysis Results

- Input for Enterprise Systems
  - ✓ Results may be automatically or manually fed directly into enterprise systems to enhance and optimize their behaviors and performance
  - ✓ Online store can be fed processed customer-related analysis results that may impact how it generates product recommendations

- Business Process Optimization
  - ✓ The identified patterns, correlations and anomalies discovered during the data analysis are used to refine business processes
  - ✓ Consolidating transportation routes as part of a supply chain process

- Alerts
  - ✓ Results can be used as input for existing alerts or may form the basis of new alerts
  - ✓ Alerts may be created to inform users via email or SMS text about an event that requires them to take corrective action

Summary

- What is C, A, P and various options when partitions happen
- Why is this important for Big Data environment where faults may happen in large distributed systems
- CAP Theorem
- How example NoSQL DBs, such as MongoDB and Cassandra, can enable multiple consistency options depending on the application requirement
- What process steps must one should take in a Big Data Analytics project

Next Session:
Distributed programming