Amazon Dynamo

By: David Gelbtuch &
Raziel Siegman

CAP Theorem

- When discussing distributed systems, there are three properties to discuss:
  a. **Consistency**
  b. **Availability**
  c. **Partitioning/ reliability**
- According to the **“CAP” Theorem**, only 2 of these properties can be maintained in a distributed system
- **Amazon Dynamo** chooses to maintain **Availability** and **Partitioning**

Purpose & Eventual Consistency

- Amazon wants to ensure customer satisfaction; therefore, any “writes” or “reads” done by customer will not be rejected.
- In order to do this, Amazon chooses to maintain **Availability** and **Reliability** at the cost of consistency and instead ops for an **Eventual Consistency**.
- This means that information may not be consistent across all nodes at all times.

Amazon Dynamo Structure

Consistent Hashing

- Amazon Dynamo uses **Consistent Hashing** in order to scale incrementally and to maintain availability and reliability.
- Nodes are placed on a “circle” and each data item is hashed to a position on the circle.
- Each node is responsible for the items hashed between it and the preceding node.

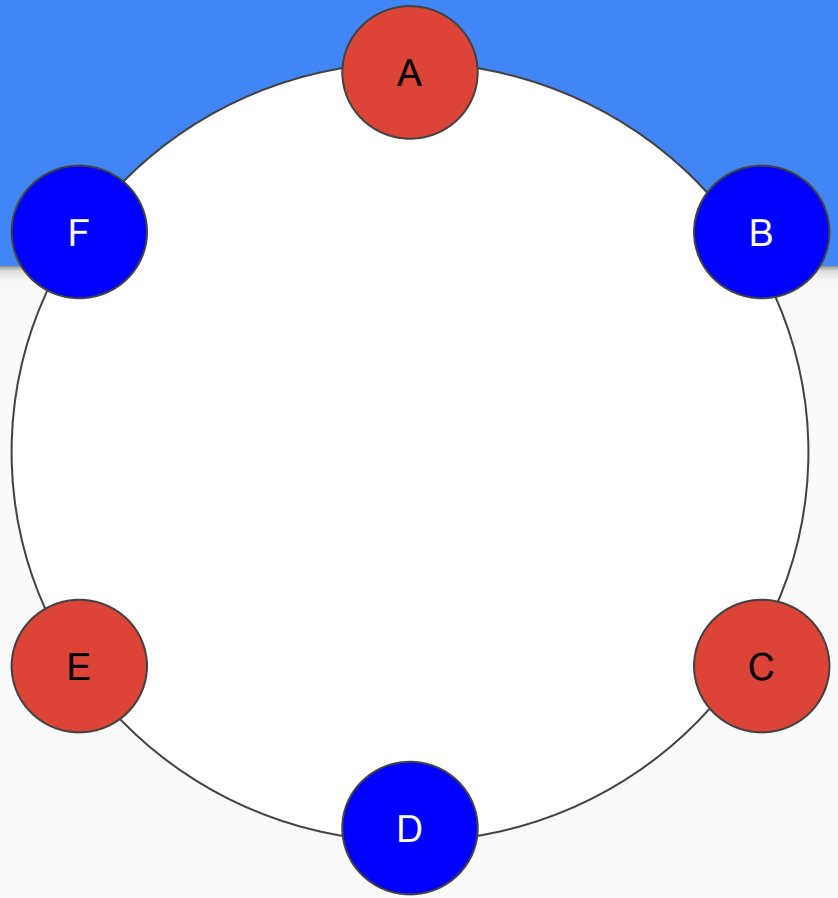
Consistent Hashing: Scalability

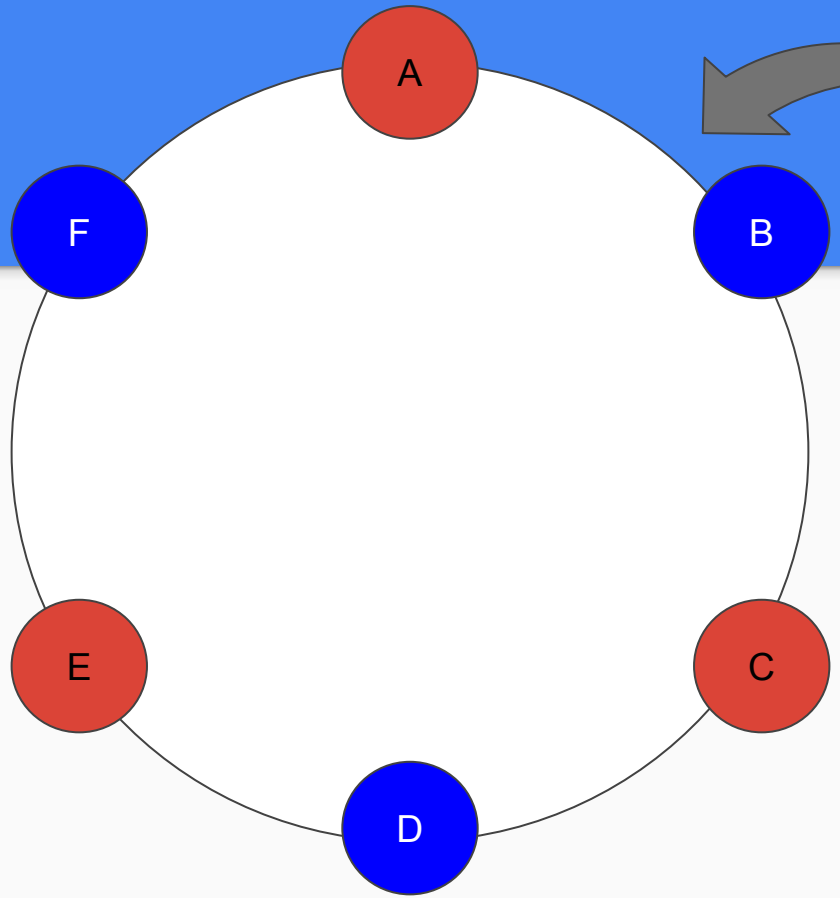
- Since nodes are responsible for only a specific region on the circle, any nodes that are added or removed from the circle will only affect the immediate neighboring nodes.
- Furthermore, to allow for consistent load distribution of the data, each node gets assigned multiple “virtual nodes” on the ring.
    - This means that if a node becomes unavailable, the load handled by that node is evenly distributed amongst the others.
    - When a new node is added, it accepts an even amount of load from each other node.
    - The number of virtual nodes can be decided based on a nodes capacity.

Consistent Hashing: Replication

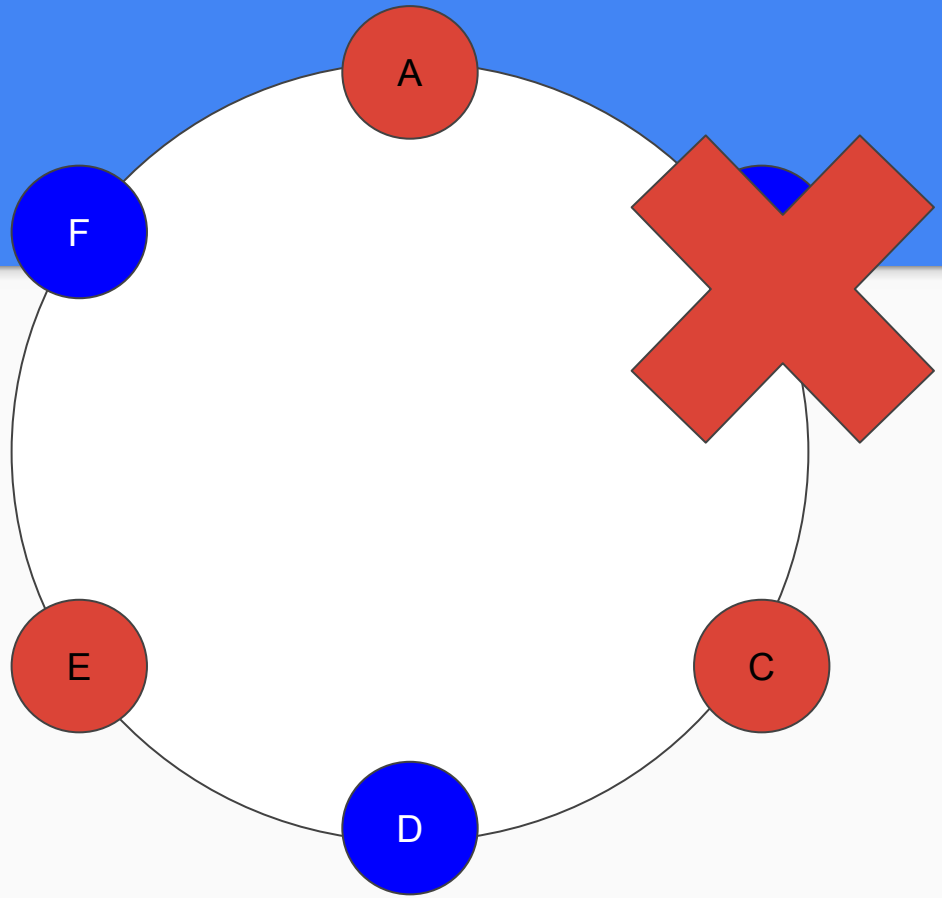
- Achieving high availability and durability
  - If a node fails, it must be ensured that all data does not get lost.
  - To achieve this, at the time of a “write”, a coordinator node is tasked with replicating the data to be stored, and placing it at the hashed node, as well as the N-1 subsequent nodes (totally N equivalent sets of data).
    - Note: For the purposes of replication, virtual nodes are skipped, as they do not qualify as one of the N nodes.

Example

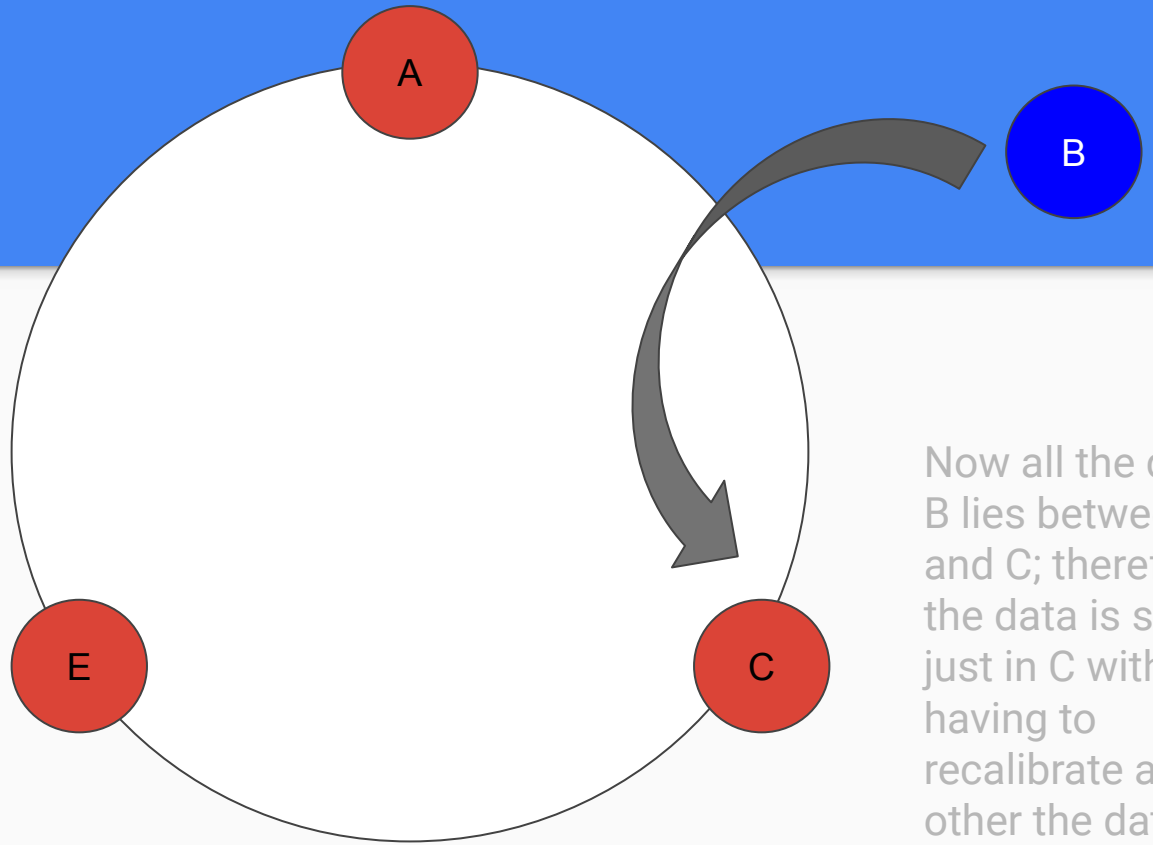

A
B
C
D
E
F



Key, K is placed between nodes A and B. Therefore the value will be stored in B.



Let’s Say Node B fails and all it’s virtual nodes fail.



Now all the data in B lies between A and C; therefore, the data is stored just in C without having to recalibrate all other the data.

Eventual Consistency: Data Versioning

- Although Dynamo does not maintain normal consistency, it ops instead for eventual consistency.
- Dynamo ensures high availability and durability. In order to do this, Dynamo treats each “put” as a new, immutable version of the data.
- Most of the time, the old version will be replaced with the new data without a partition.

Eventual Consistency: Data Versioning

- However, there may be times where different versions of the data shows up.
- In order to reconcile the data, Dynamo uses *synactic* or *semantic reconciliation* to merge different versions. This means that no “put” will ever be declined, but not every “delete” will work and deleted items can resurface.

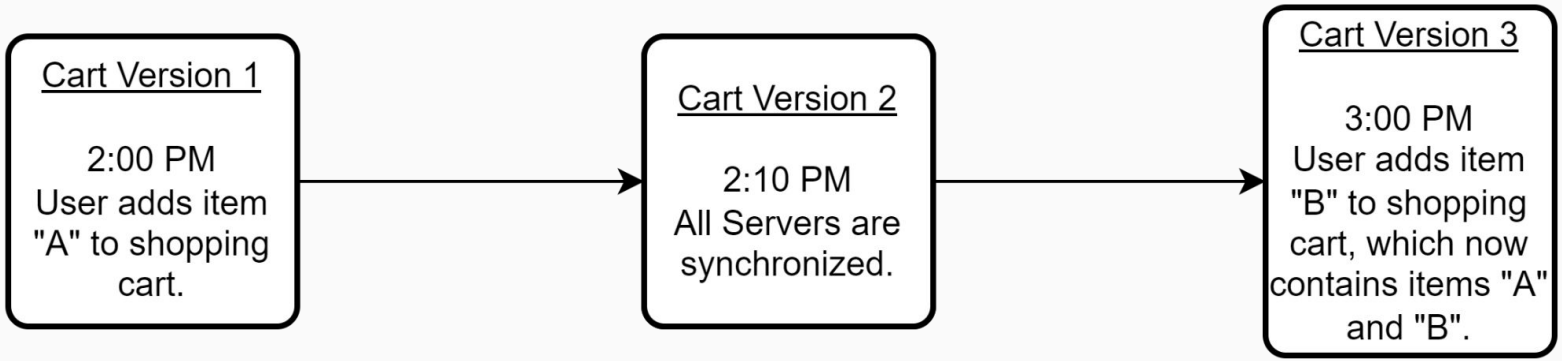
Shopping Cart

- No Partitions
- Partitions
  - Syntactic
  - Semantic

Shopping Cart: No Partitions

1. Client adds item to cart.
2. Server writes a new immutable object by taking the available cart information and combining it with the new item to create a new cart.
3. All nodes will eventually update with the new cart information.

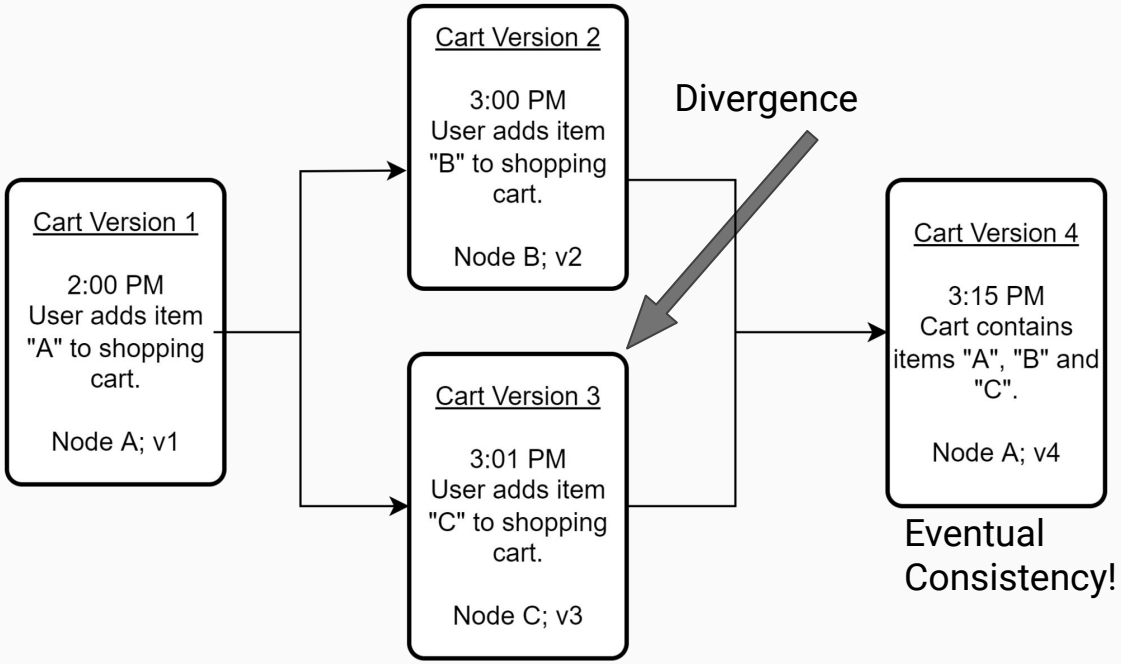
Data Versioning While Populating a shopping Cart (No Divergence)



Shopping Cart With Partitions - Syntactic

1. Client adds item to cart.
2. Client then tries adding two new items to cart, before servers have a chance to synchronize their data.
3. For each item, a different server writes a new immutable object by taking the old cart information and combining it with the new item to create a new cart, leading to two carts that differ in their contents.
4. The node corresponding to each server will eventually become aware of the partition, and create a new cart containing all the relevant items.

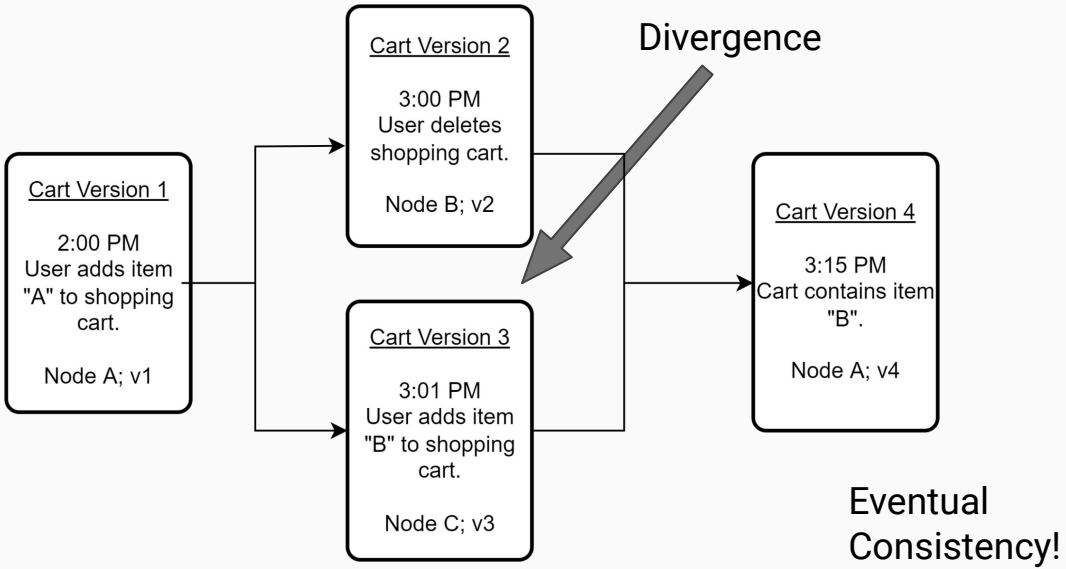
Shopping Cart: Partitions - Syntactic



Shopping Cart With Partitions - Semantic

1. Client adds item to cart.
2. Client then performs two operations: an addition to the cart, and a deletion of the cart. A different server writes a new immutable object for each operation, leading to two carts that differ in their contents.
3. The node corresponding to each server will eventually become aware of the partition, and be unsure as to how to resolve the divergence. Should the cart be deleted, or contain two items? The partition must be resolved semantically and the client is asked to choose the cart.

Shopping Cart: Partitions - Semantic



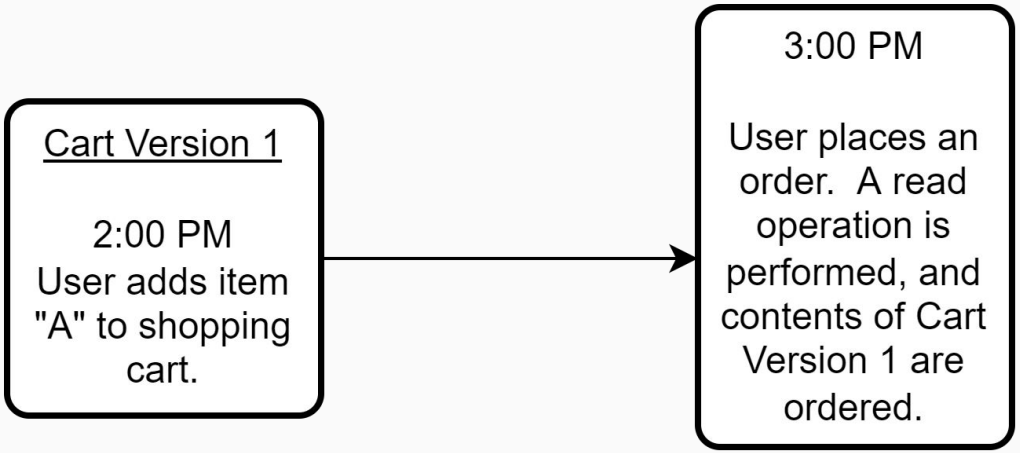
Ordering

- No Partitions
- Partitions
  - Syntactic
  - Semantic

Ordering: No Partitions

- Client wants order something from cart.
- Server will receive the order from the client, read the contents of the cart, and order the item.
- Client will then be charged for the order.

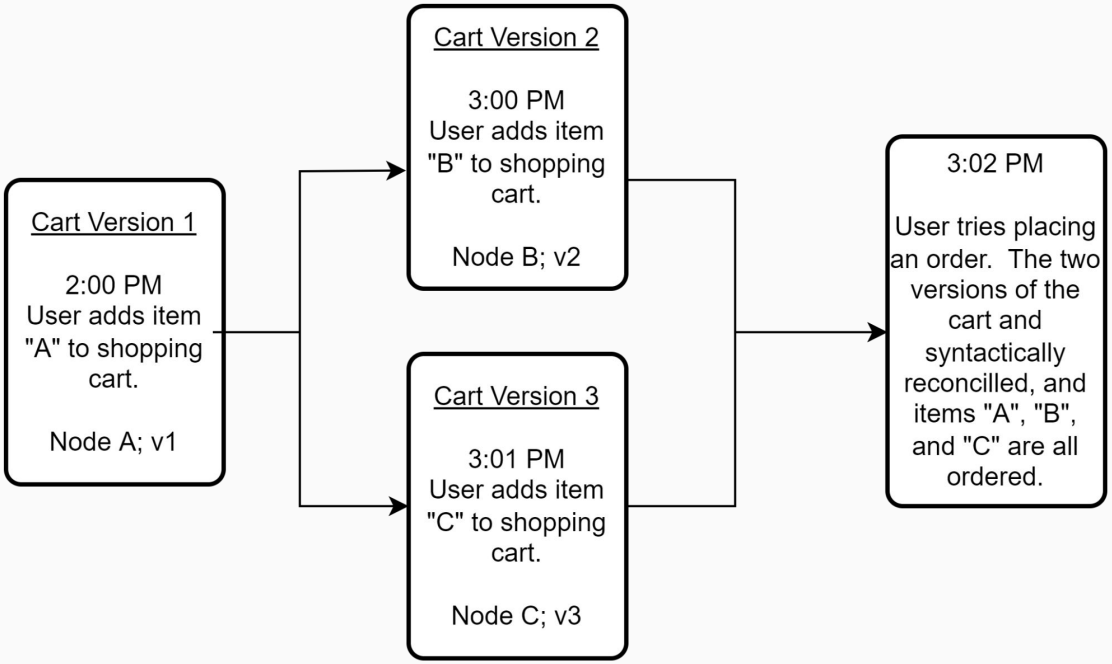
Ordering: No Partitions



Ordering: Partitions - Syntactic

- Client wants to order items from the cart.
- However, when the server reads the cart object, there are two versions of the cart.
- This problem is solved through *syntactic reconciliation*, where the carts are merged and then the contents of the cart are ordered.

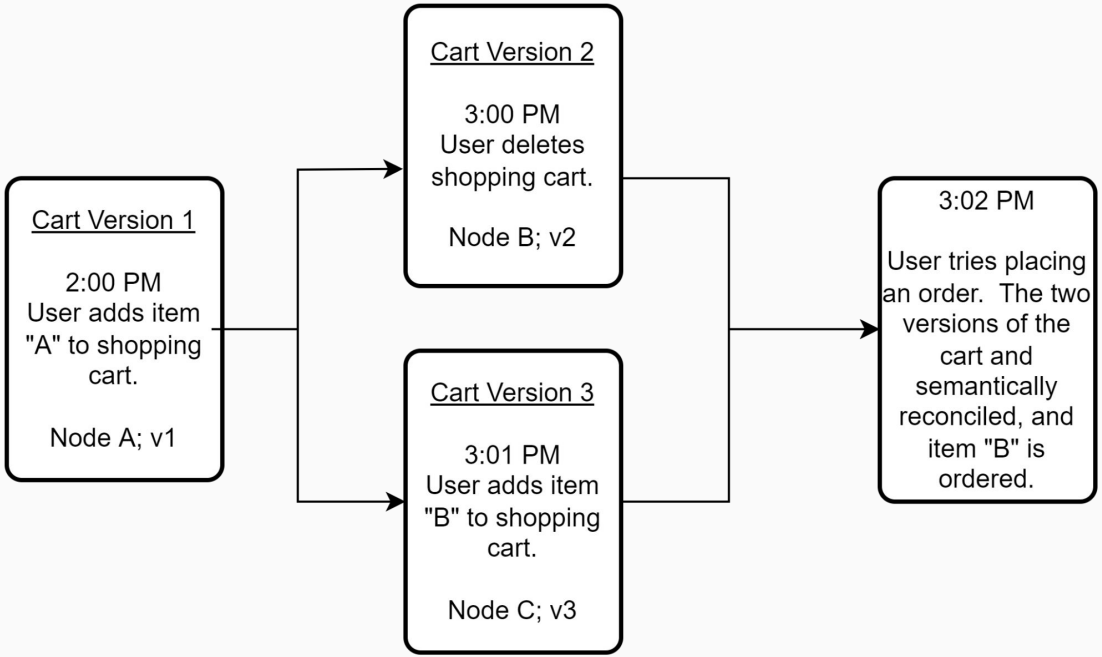
Ordering: Partitions - Syntactic



Ordering: Partitions - Semantic

- Client wants to order items from the cart.
- However, when the server reads the cart object, there are two versions of the cart. This time, one cart was deleted, while the other was not.
- Semantic resolution is performed, and the user selects the cart that had an item added, allowing the order to be processed.

Ordering: Partitions - Semantic



Ordering: Partitions - Resolutions

- In a case of a partition where there is no resolution, there could be a case where the client attempts to make an order, but when the server tries to read the cart, there are multiple versions of the cart that contain differing data.
- Due to syntactic/semantic resolution, a single cart is decided on and Amazon is able to order the proper items to the client.