Amazon Dynamo Presentation

By Jacob Silbiger and Shai Vadnai

What is Amazon Dynamo?

- Amazon Dynamo is a distributed Key, Value Store
- Dynamo was outlined in a Whitepaper by Amazon presented at the SOSP Conference in 2007
- Unlike conventional distributed data stores, Dynamo avoids the notion of a strict quorum – allowing for continued operation during network partitions
- In order to provide this level of availability, Dynamo makes compromises in other areas of its operation by innovating the idea of Eventual Consistency

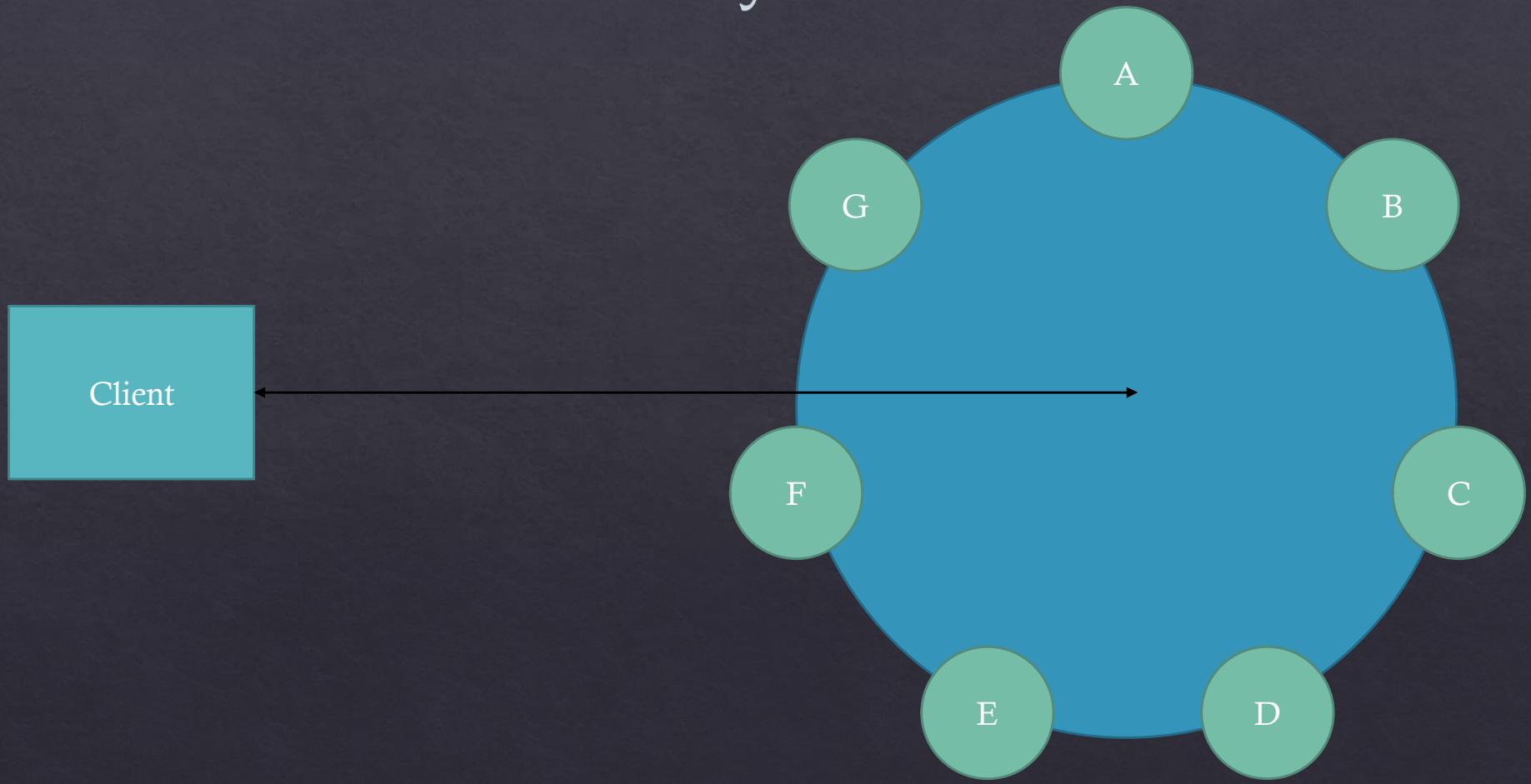
Eventual Consistency

- Provides continued read and write operations even during network partitions
  - This provides for a more seamless client experience during critical parts of the process
- Eventual Consistency means that nodes can temporarily have divergent versions of the same data
- As a result, operations that inherently require a consistent view of the data must conduct conflict resolution
  - We sacrifice the performance of some operations for overall availability and the increased performance of other operations

Conflict Resolution

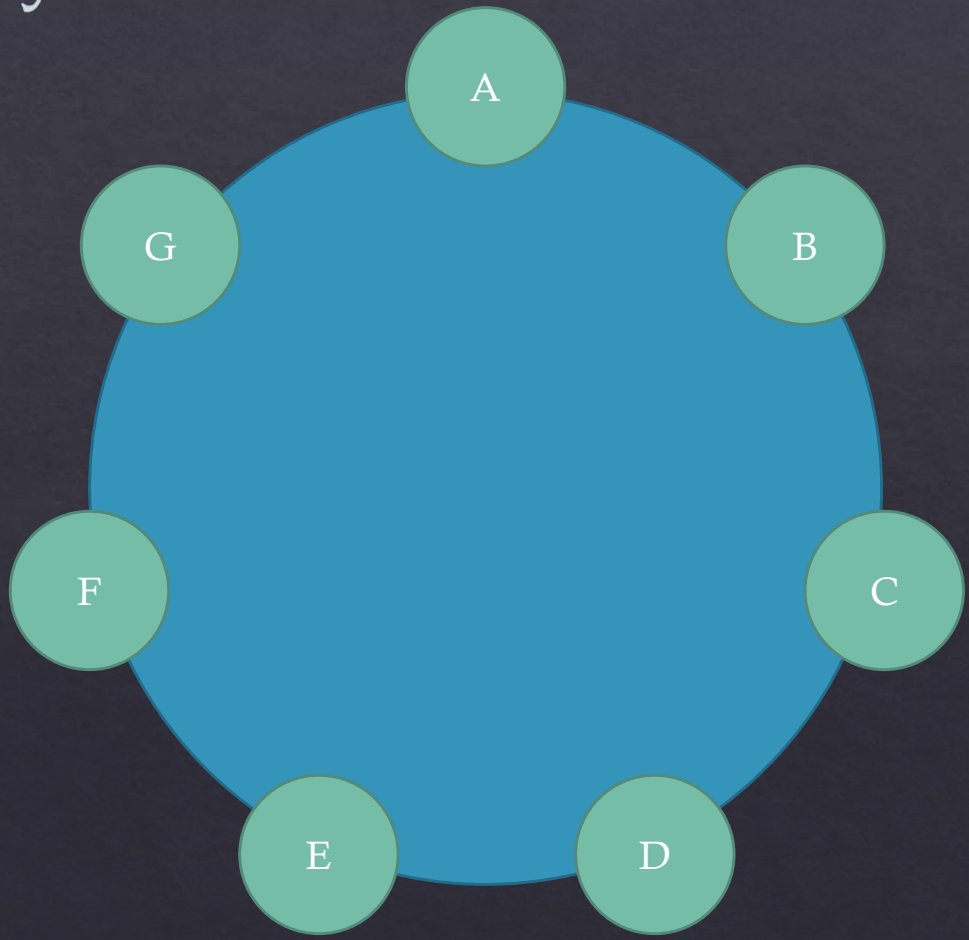
- We've established that one operation will be more performant at the expense of another, but which operations should those be?
- Many data stores prefer to perform conflict resolution on writes to keep read complexity simple, the assumption being that those data stores are more Read-Heavy or Read Latency Sensitive
- Dynamo was designed to handle certain use cases in which write operations are given priority
  - Specifically, a digital shopping cart – in which the ability to add to the shopping cart is more latency-sensitive vis a vis the user experience
  - Only when an order is placed must all nodes have a consistent view of the data
- Therefore, Dynamo reconciles on *reads* and allows for more performant *write* operations in which partitions may occur

Architecture of a Dynamo Cluster



Architecture of a Dynamo Cluster

- The hash space for a given cluster is divided into address spaces
- Each node in the cluster is assigned to replicate the data in an address space
- Every node can accept a client request
  - If the Key hashes to its address space, the node handles the request
  - Otherwise, the request is passed to the first node (clockwise) which supports said address space



Dynamo's Notion of Quorum

- N is the number of nodes on which a given address space should be replicated
  - In our example N = 4
- W is the number of nodes which need to confirm a given Write
  - In our example W = 2
- R is the number of nodes which need to confirm a given Read
  - In our example R = 3
- By configuring Dynamo such that W + R > N, we ensure that a given read will contain at least 1 node from every write

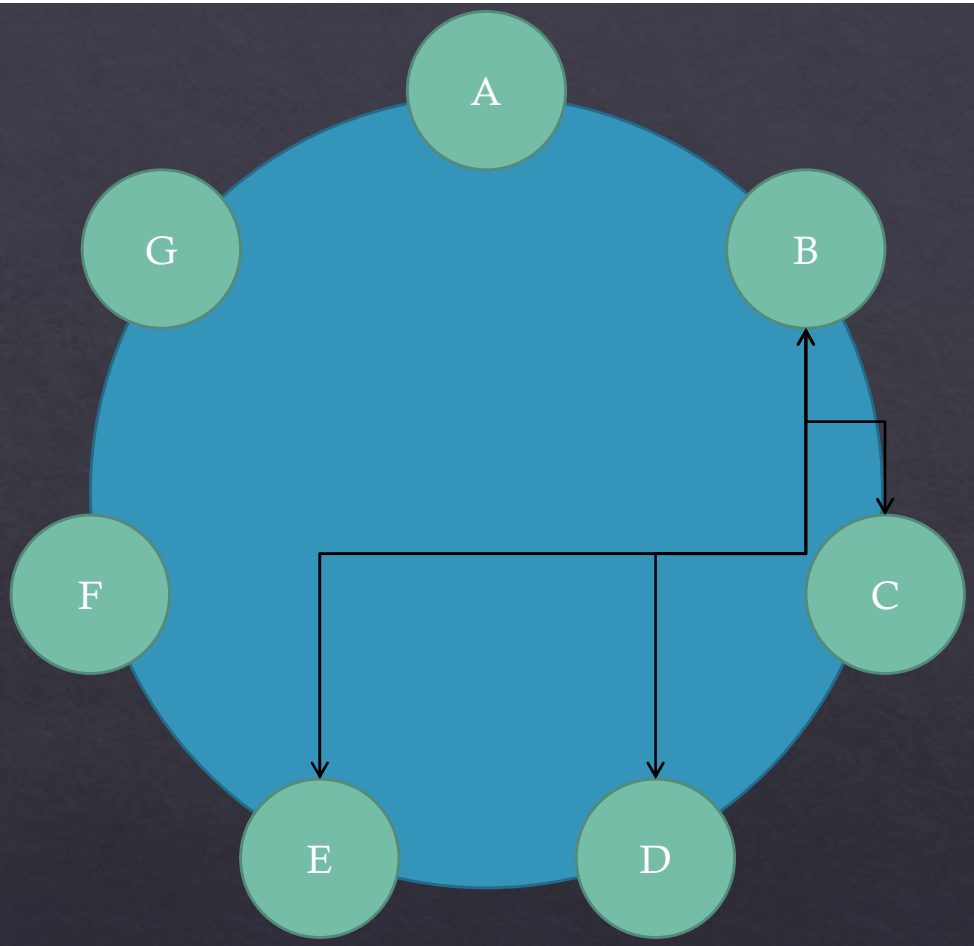
Case 1

No Partition

A single coordinator handles all requests for a given object

Case 1: Populate Cart, No Partition

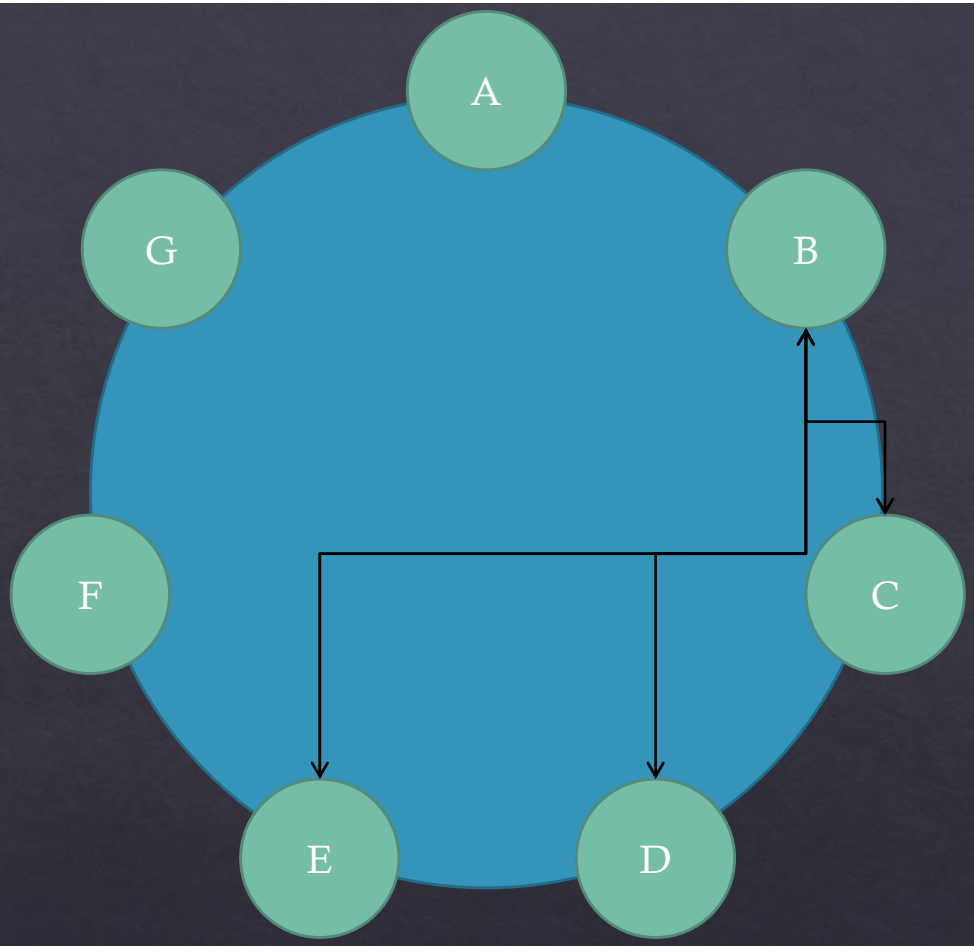
1. Node B accepts X
    1. Where X is a K,V pair
    2. K is a key representing a customer and V is a shopping cart
2. Node B generates a vector clock for X and adds it to its own store
3. B sends X and the vector clock to C, D and E for replication
4. C, D and E all send write confirmations to B
    1. Write succeeds



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)

Case 1: Place Order, No Partition

1. Client requests X
2. B handles the request
3. B submits the request to C, D and E
4. B waits for R responses
    1. C, D and E respond
5. B sees no conflict in vector clocks between the first R nodes to respond and sends X
    1. Read succeeds



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)

Case 2

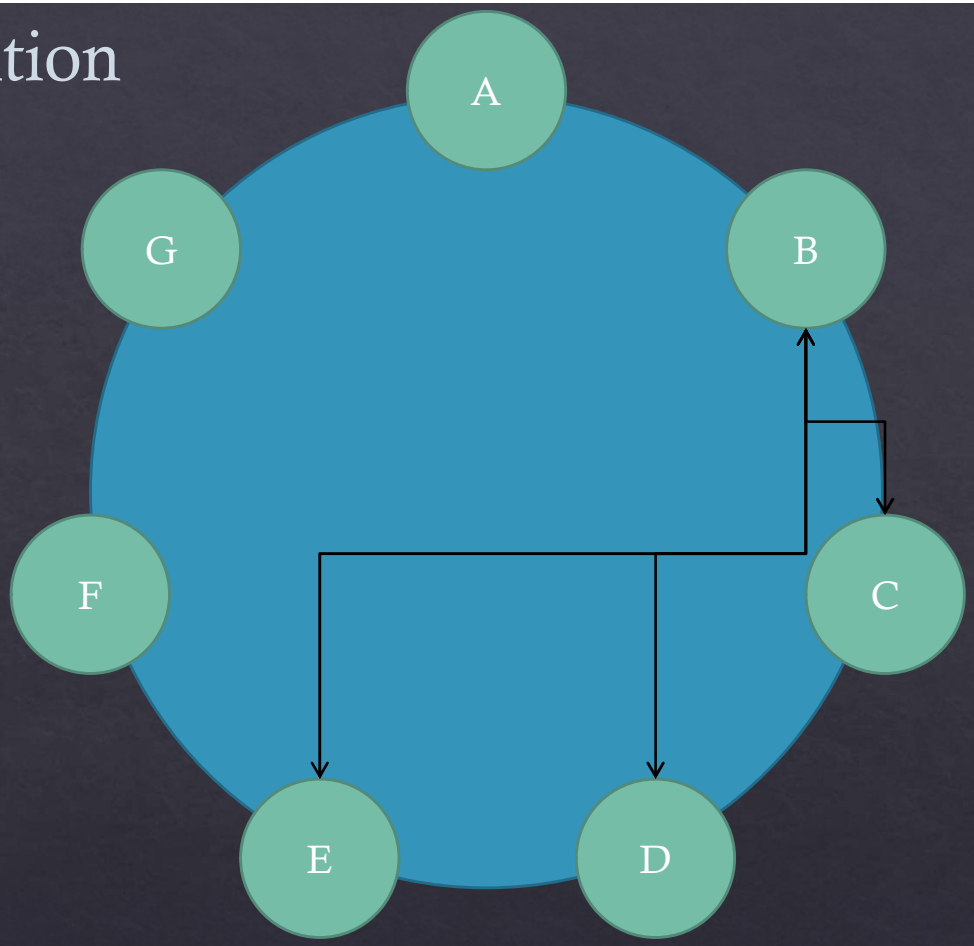
Semantic Partition

A single coordinator handles requests for a given object other nodes go down

Sematic partitions can be reconciled internally leveraging vector clock logic

Case 2: Populate Cart, Semantic Partition

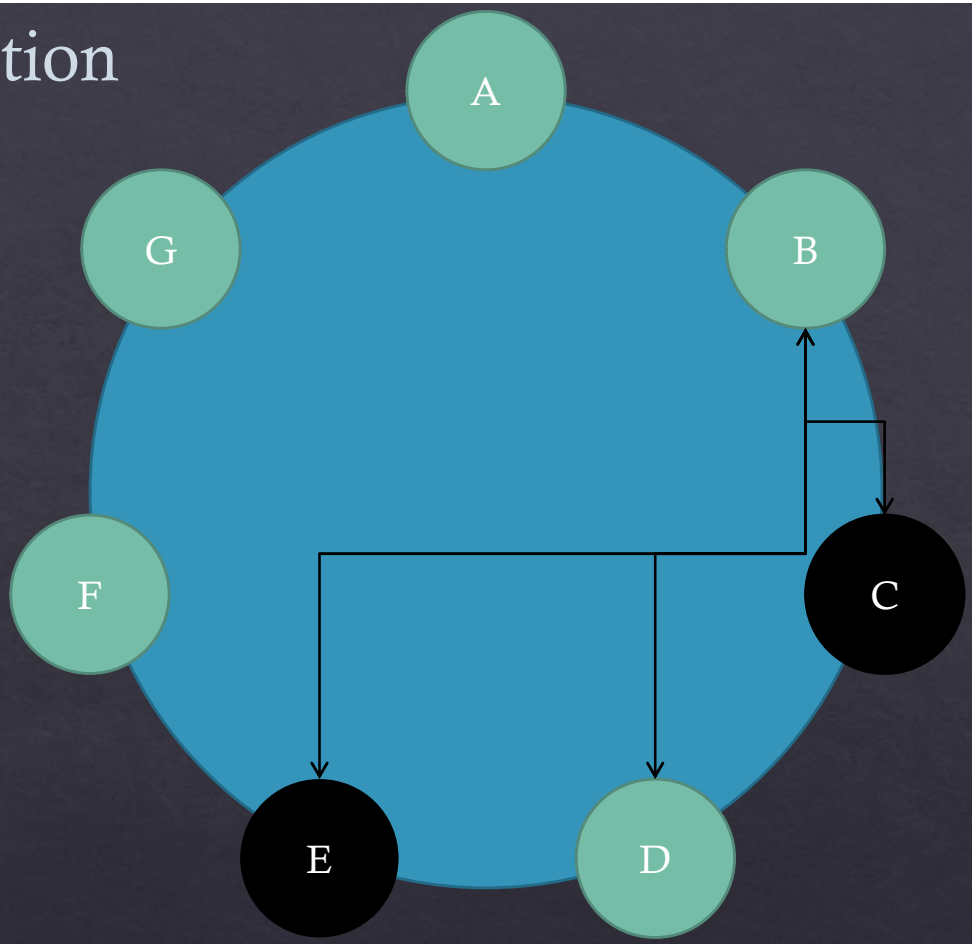
1. Node B accepts X
2. Node B generates a vector clock for X and adds it to its own store
3. B sends X and the vector clock to C, D and E for replication
4. C, D and E all send write confirmations to B
    1. Write is successful



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)

Case 2: Populate Cart, Semantic Partition

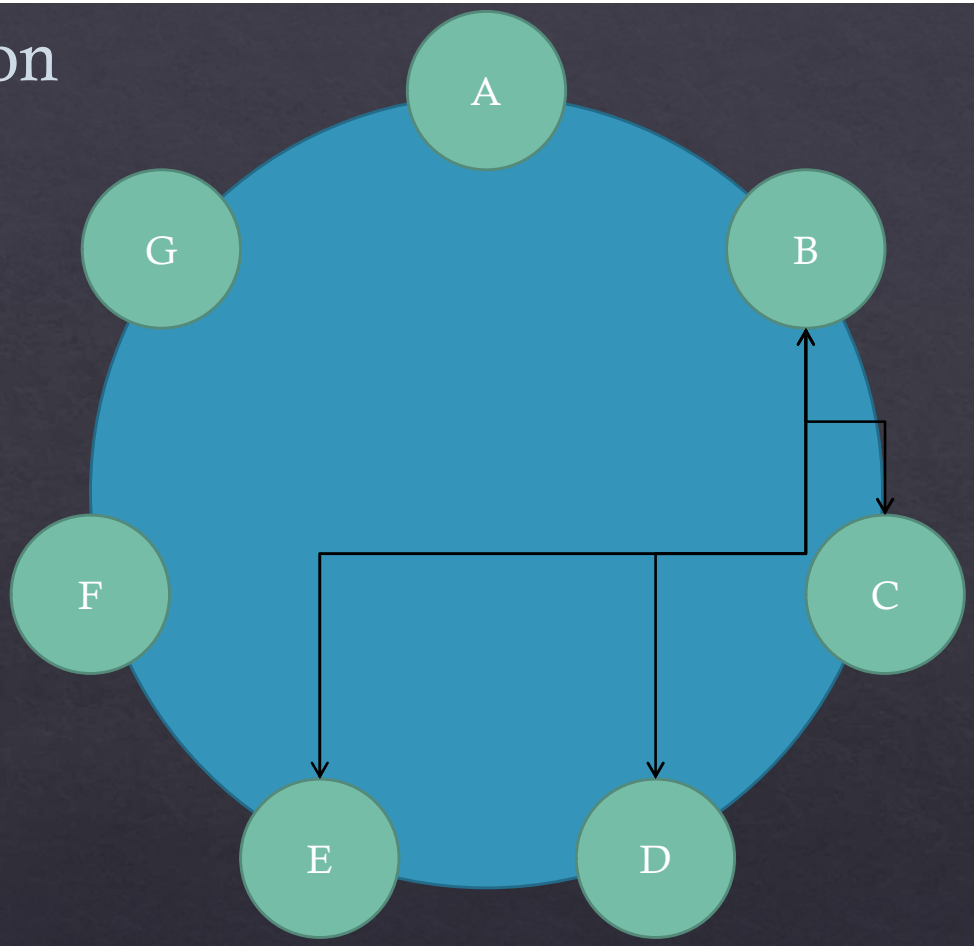
1. Nodes C and E go down
2. Node B accepts a new X
3. Node B appends to X's vector clock and adds it to its own store
4. B sends X and the vector clock to C, D and E for replication
5. Only D send write confirmations to B
    1. Write is confirmed since W = 2



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)
X (B,2)		X (B,2)	

Case 2: Place Order, Semantic Partition

1. Nodes C and E come back online
2. Client requests X
3. Node B handles request
    1. Requests X from C, D and E
4. B sees vector clock conflict in first R nodes and reconciles
    1. C and E have no entries that B and D *don't* have, therefore C and E are just outdated since (B,2) supersedes (B,1)
5. B sends X to Client and updates C and E



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)
X (B,2)	X (B,2)	X (B,2)	X (B,2)

Case 3

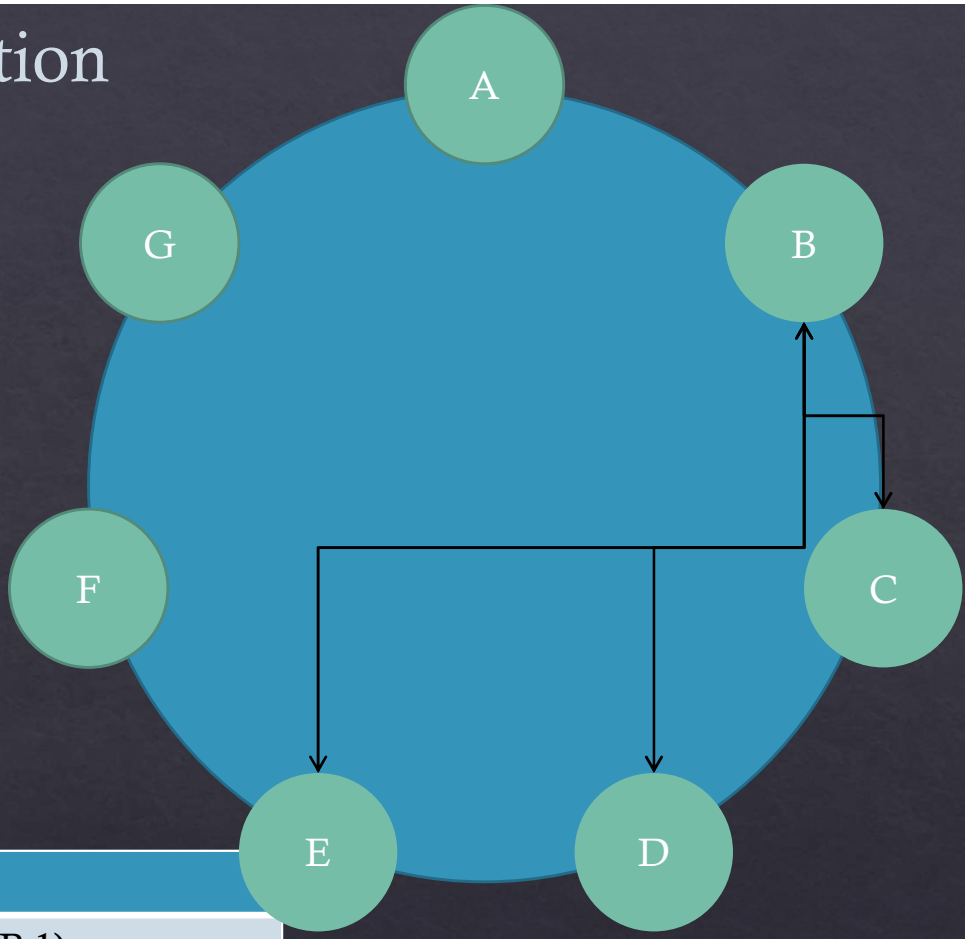
Syntactic Partition

Multiple Coordinators due to servers going down

Can not be reconciled internally due to lack of vector clock intersection

Case 3: Populate Cart, Syntactic Partition

1. Node B accepts X
2. Node B generates a vector clock for X and adds it to its own store
3. B sends X and the vector clock to C, D and E for replication
4. C, D and E all send confirmations to B
    1. Write succeeds



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)

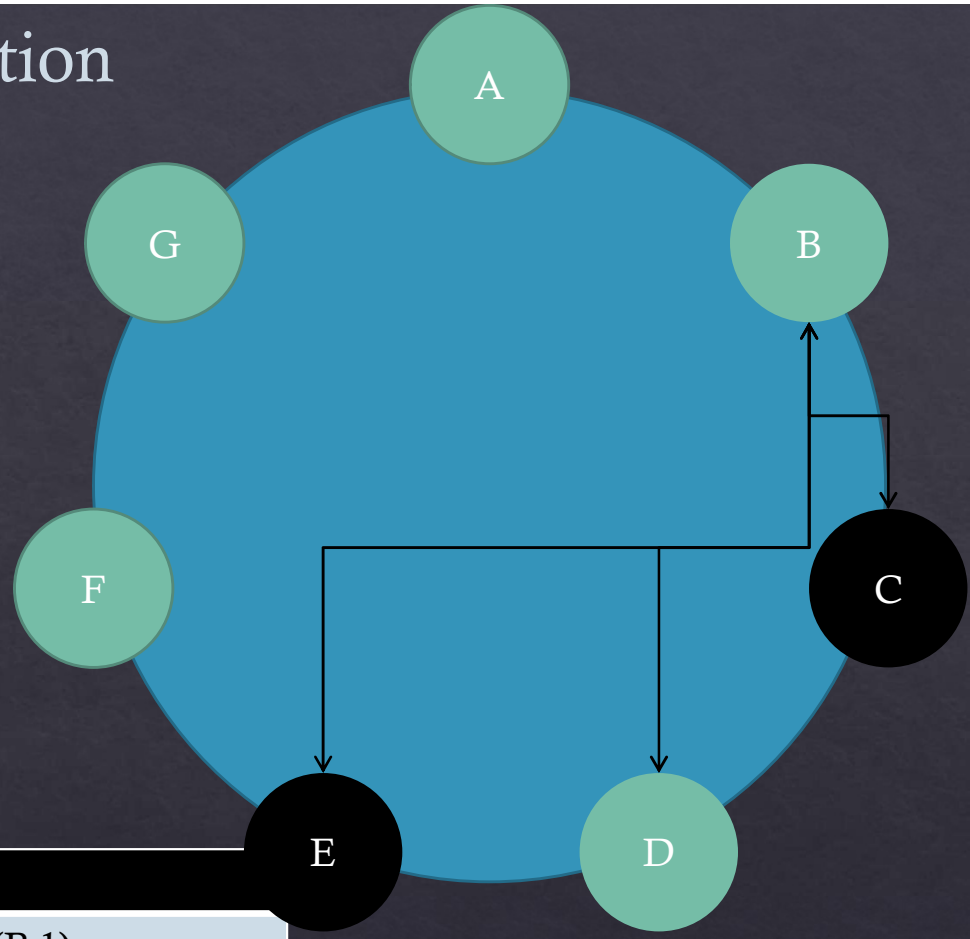
Case 3: Populate Cart, Syntactic Partition

1. Nodes C and E go down

2. Node B accepts a new X

3. Node B appends to X's vector clock
4. B sends X and the vector clock to C, D and E for replication

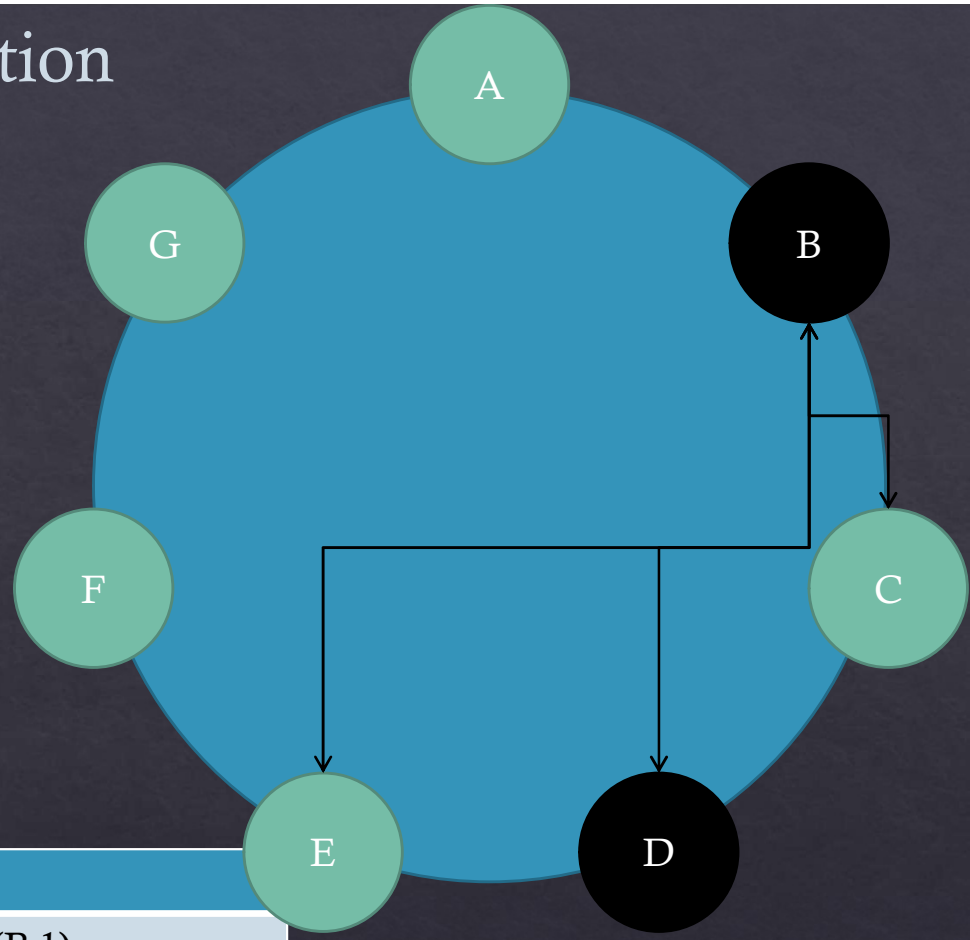
5. Only D sends a write confirmations to B
    1. Write succeeds because W=2



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)
X (B,2)		X (B,2)	

Case 3: Populate Cart, Syntactic Partition

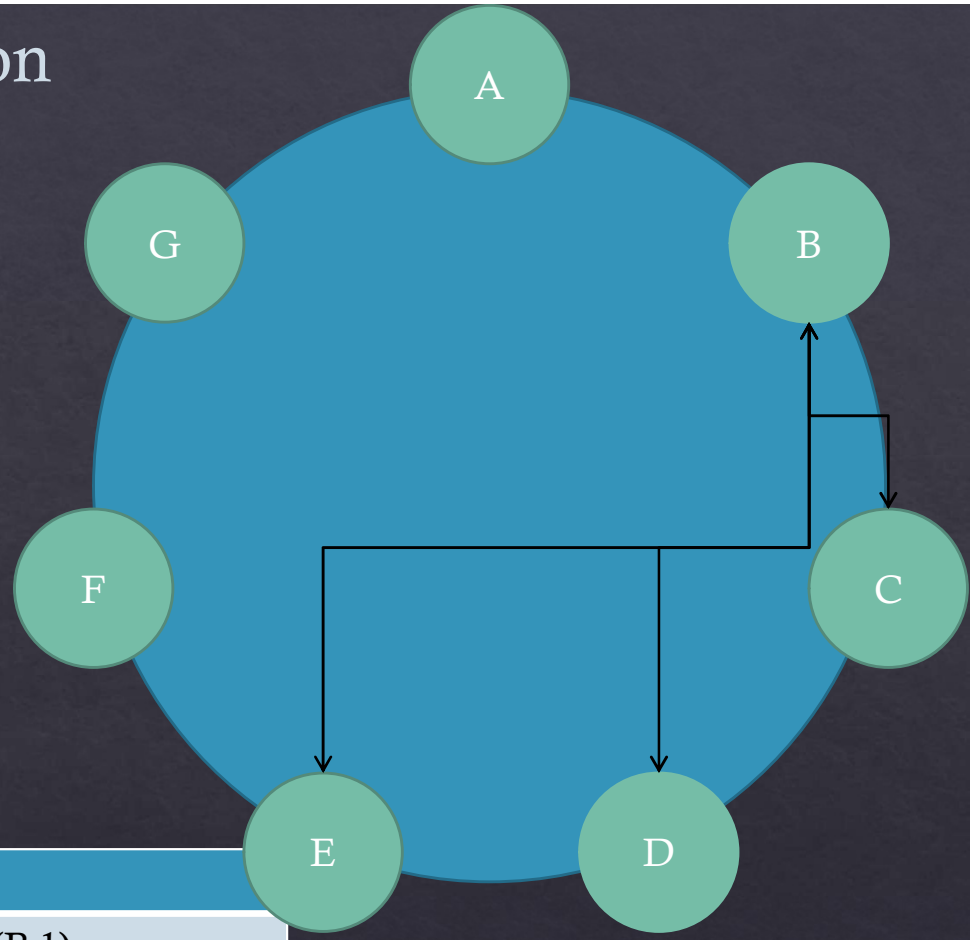
1. Nodes C and E come back up while B and D go down
2. Node C (which is now the coordinator) accepts a new X
3. Node C appends to X's vector clock
4. C sends X and the vector clock to D and E for replication
5. Only E sends a write confirmation to C
    1. Write succeeds because W=2



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)
X (B,2)		X (B,2)	
	X (B,1)(C,1)		X (B,1)(C,1)

Case 3: Place Order, Syntactic Partition

1. Nodes B and D come back online
2. B receives Client request for X
3. B requests X from C, D and E
4. B notices vector clock conflict
    1. B and D vs. C and E
5. Both options contain exclusive information, therefore B can not reconcile
6. B sends both versions [X (B,2) and X (C,1)] to client
7. Client reconciles and sends updated X
8. B appends and sends X (B,3)(C,1) to C, D and E



B	C	D	E
X (B,1)	X (B,1)	X (B,1)	X (B,1)
X (B,2)		X (B,2)	
	X (B,1)(C,1)		X (B,1)(C,1)
X (B,3)(C,1)	X (B,3)(C,1)	X (B,3)(C,1)	X (B,3)(C,1)