CSE 124/224 Week 9 Discussion

Patrick Liu

Overview and Goals

- Implement a subset of DynamoDB
  - No consistent hashing or joining/leaving
- Understand causality using vector clocks
- Learn to implement and debug basic distributed systems

Vector Clocks

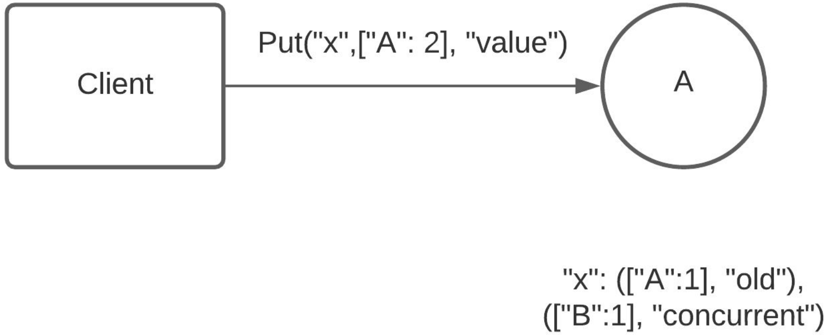
- Methods
    - LessThan
        - Implements < from lecture
    - Equals
        - Implements = from lecture
    - Combine
        - Changes the current vector clock such that all clocks in the argument list as well as the clock this is called on are <= the resulting vector clock
    - Concurrent
        - For vector clocks A and B, A < B and B < A are both false

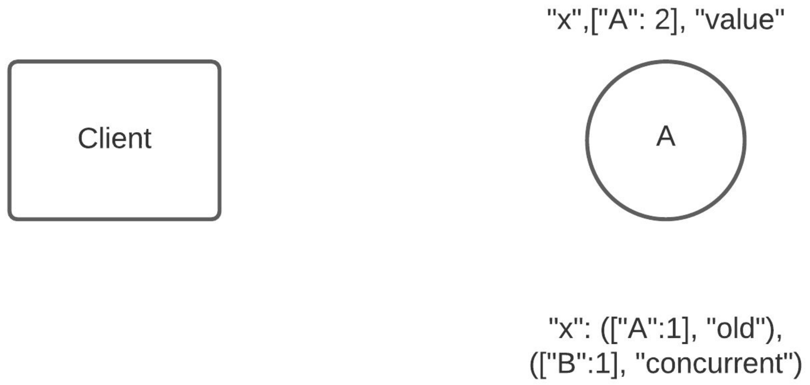
Vector Clocks

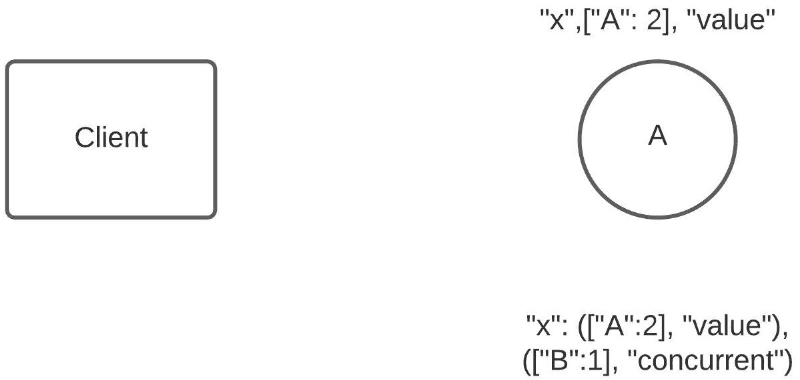
- Internal representation
  - Must be able to associate nodeID with version
  - Choices: Fixed size vector clock, or dynamic size vector clock
    - Fixed-size: We know cluster_size, so we know how big vector clocks can get
    - Dynamic size: Add elements whenever you increment for a nodeID that isn't tracked yet. Need logic for if one vector clock tracks one nodeID, and another one doesn't.

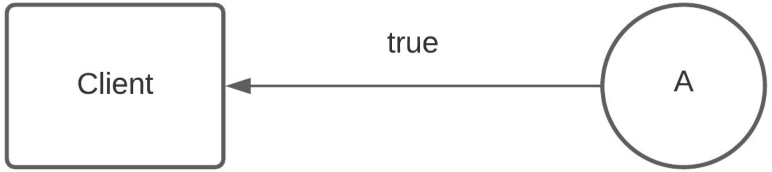
Put(on one node)

- I recommend thinking about Put on a single node separately from a Put to W nodes
    - Easy to compartmentalize
- Put on a single server does:
    - Retrieve (Context, value) pairs associated with this key
    - Check for causality
        - Keep only those (Context, value) pairs that have no causal descendant
        - If new Context == one of existing Contexts, keep existing value


Client
Put("x",["A": 2], "value")
A
"x": (["A":1], "old"),
(["B":1], "concurrent")


"x",["A": 2], "value"
Client
A
"x": (["A":1], "old"),
(["B":1], "concurrent")

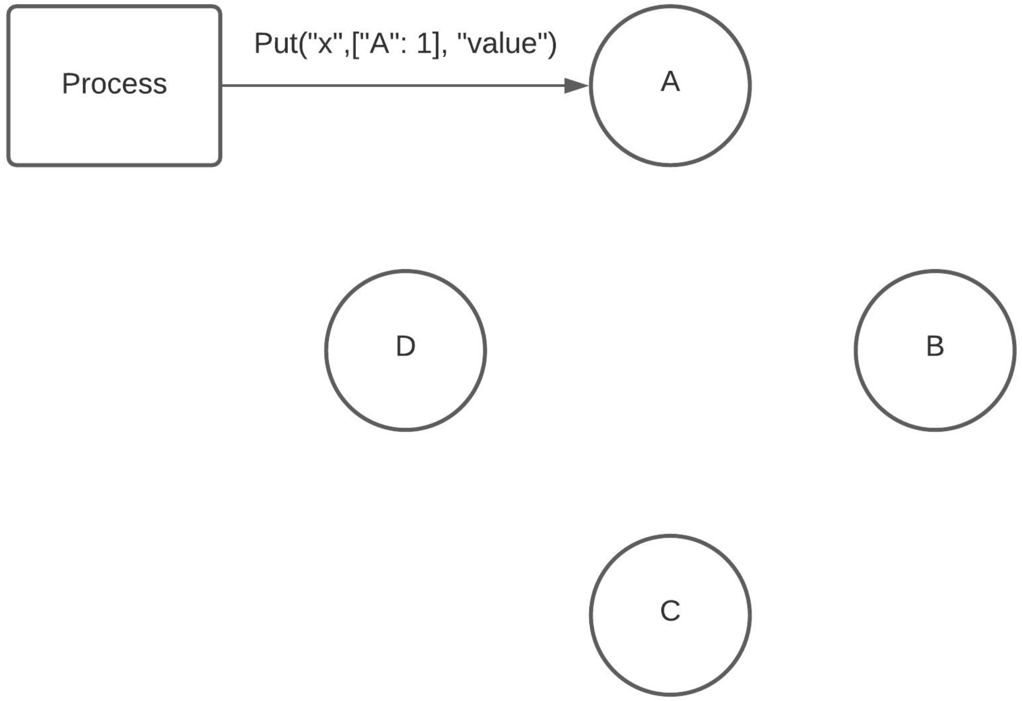

"x",["A": 2], "value"
Client
A
"x": (["A":2], "value"),
(["B":1], "concurrent")

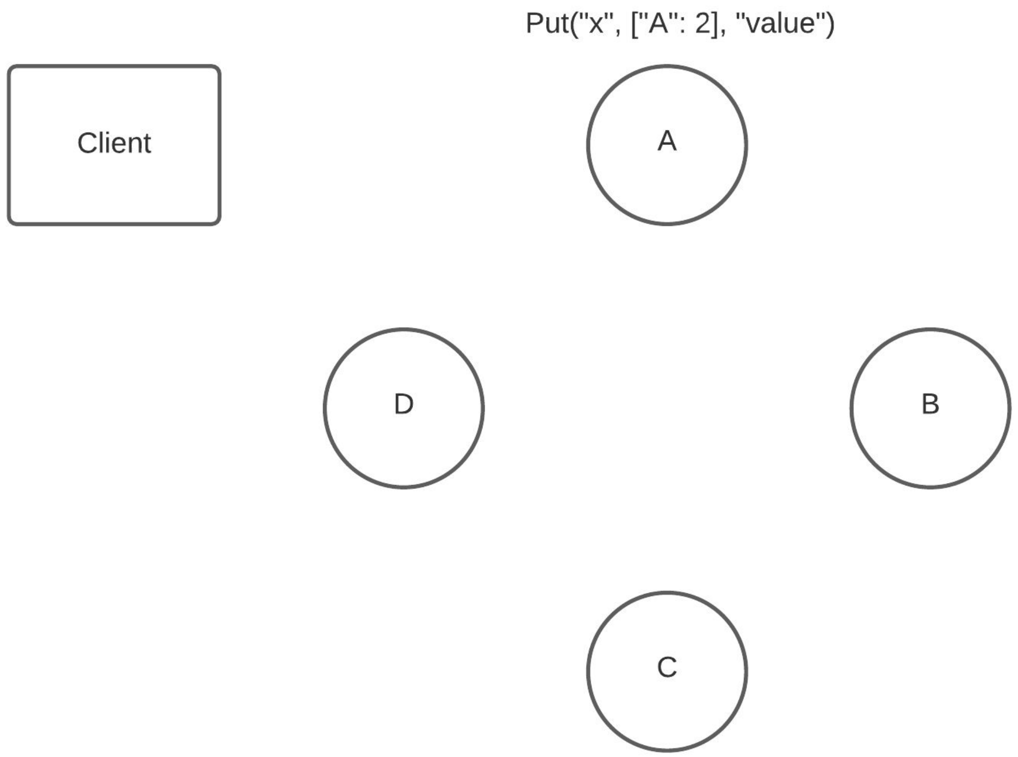


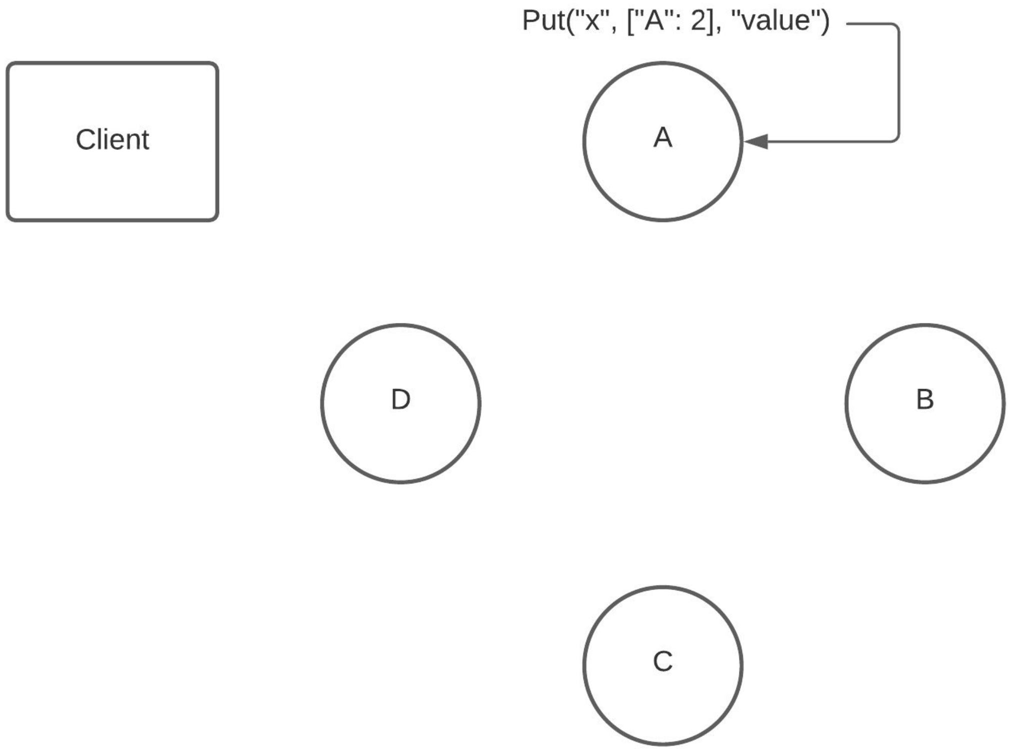
"x": (["A":2], "value"),
(["B":1], "concurrent")

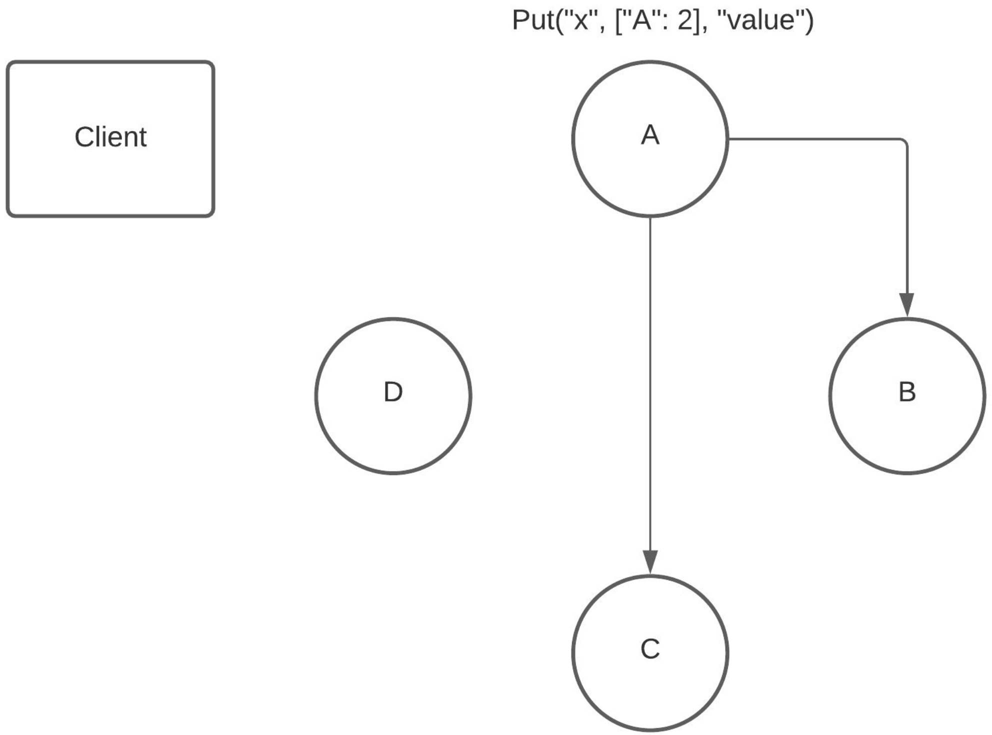
Put

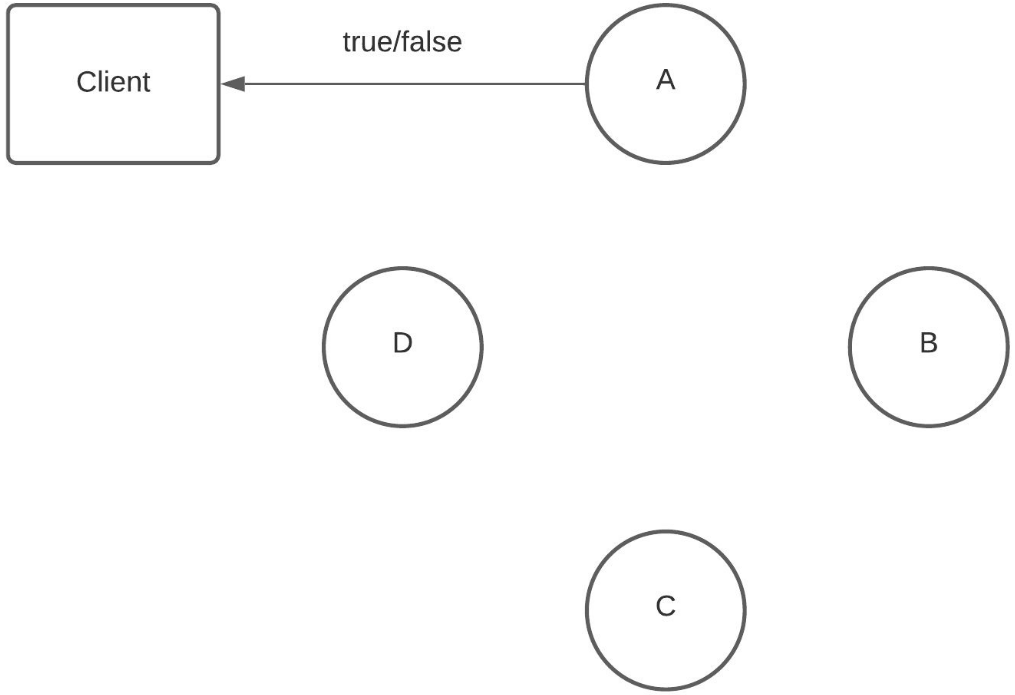
- Put will do the following things:
    - Increment the incoming Context's version associated with this node
    - Store the (Context, value) pair in local key-value store
    - Try to store the the same (Context, value) pair at W - 1 other nodes, in the order of this node's preference list.
    - Return true if everything went well, otherwise return false


Process
Put("x",["A": 1], "value")
A
D
B
C


Put("x", ["A": 2], "value")
Client
A
D
B
C


Put("x", ["A": 2], "value")
Client
A
D
B
C


Put("x", ["A": 2], "value")
Client
A
D
B
C

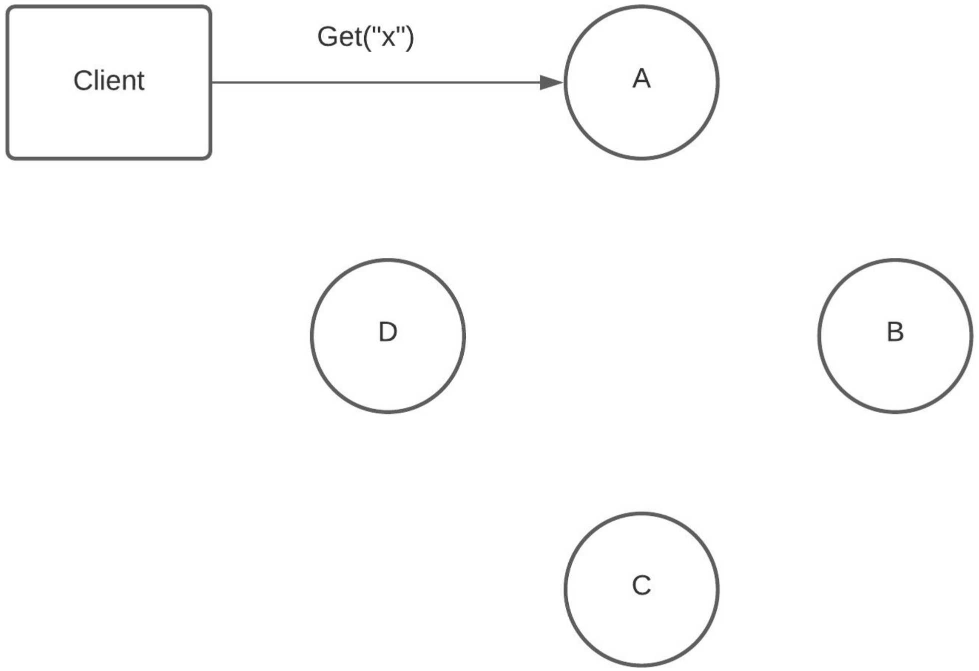

Client
true/false
A
D
B
C

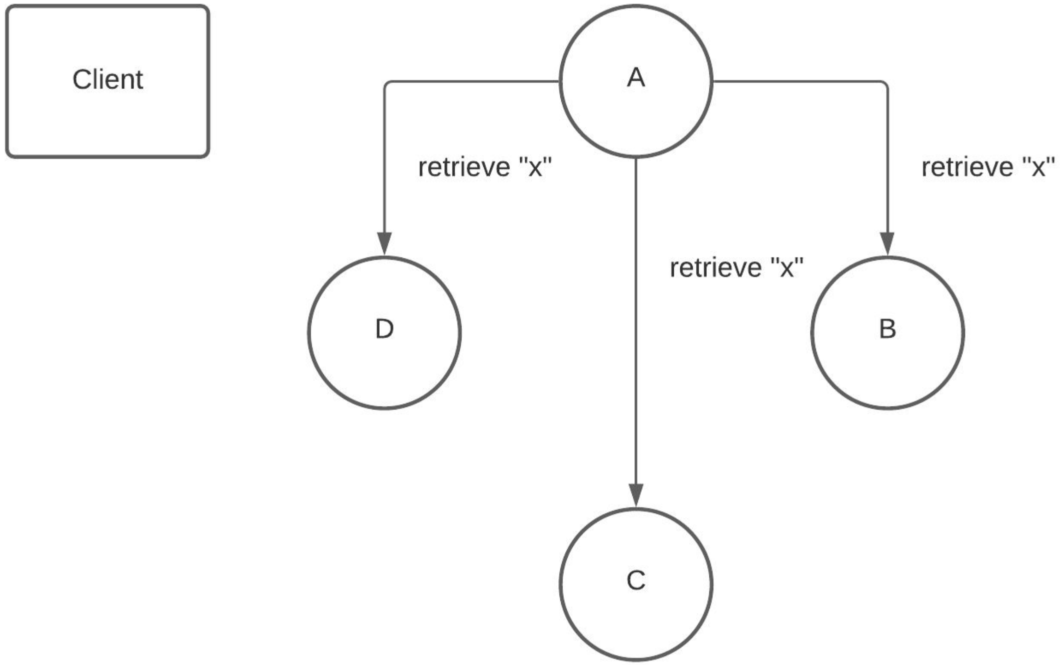
Get(from one node)

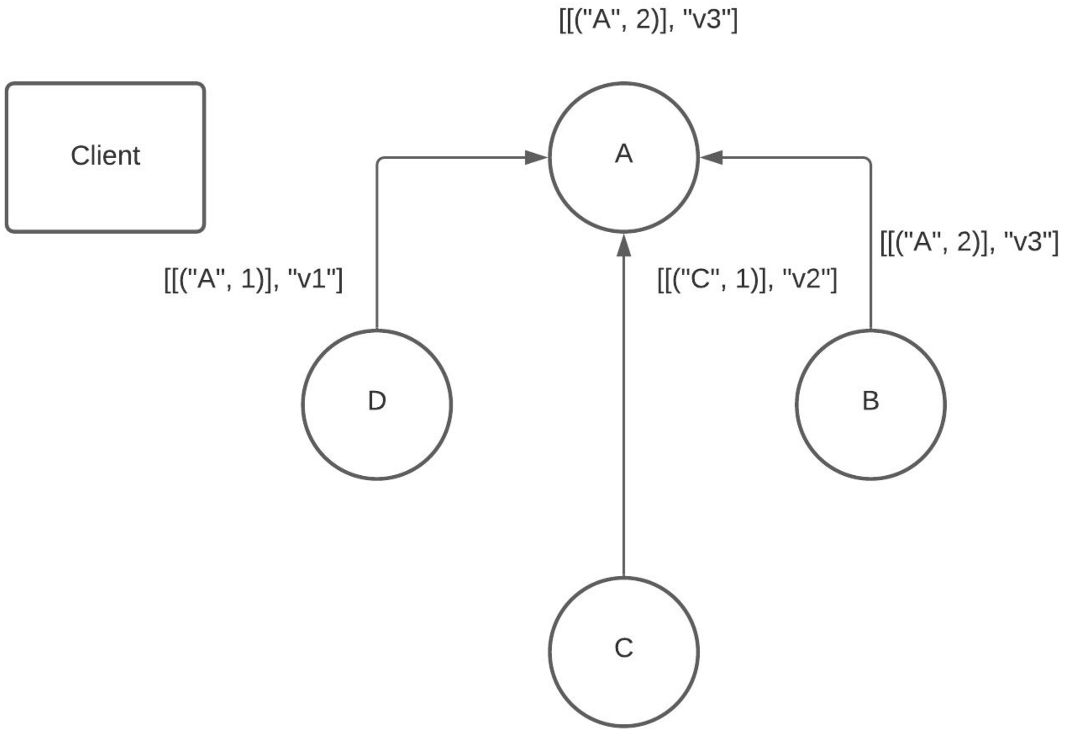
- Simply return from local key/value store
- No causality checks required(why?)

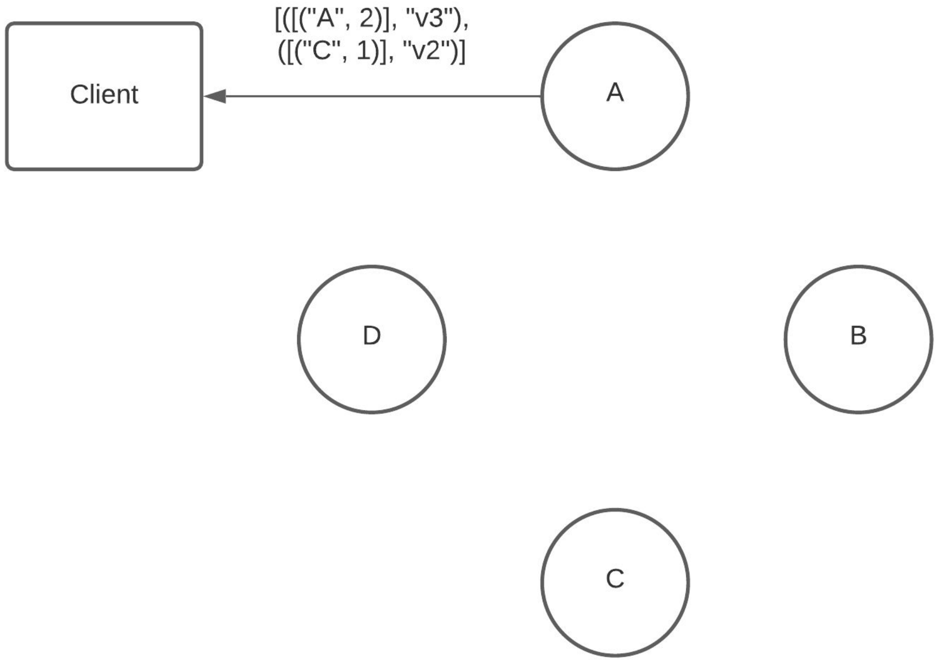
Get(from multiple nodes)

- Get from local storage, and R - 1 additional nodes
- Return only those elements that have no causal descendants.
    - This is mostly the same operation as in Put


Client
Get("x")
A
D
B
C


Client
A
retrieve "x"
retrieve "x"
retrieve "x"
D
B
C


[[("A", 2)], "v3"]
Client
A
[[("A", 1)], "v1"]
[[("C", 1)], "v2"]
[[("A", 2)], "v3"]
D
B
C

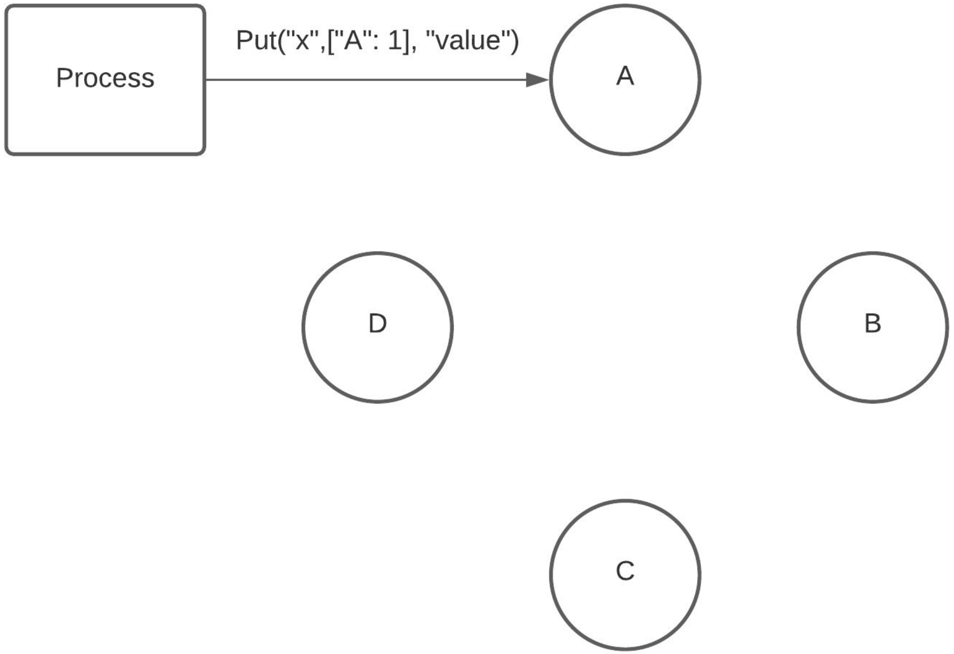

[(["A", 2)], "v3"),
([("C", 1)], "v2")]
Client
A
D
B
C

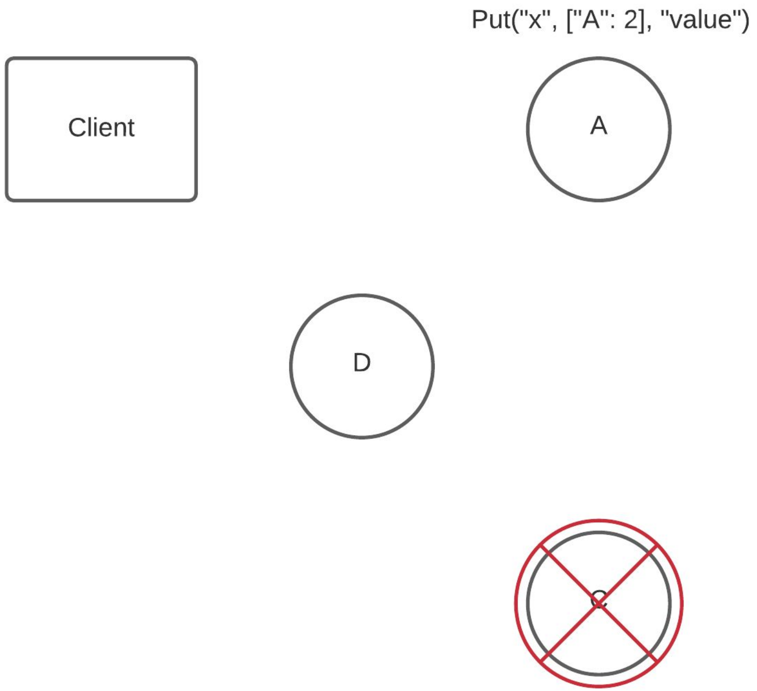
Gossip

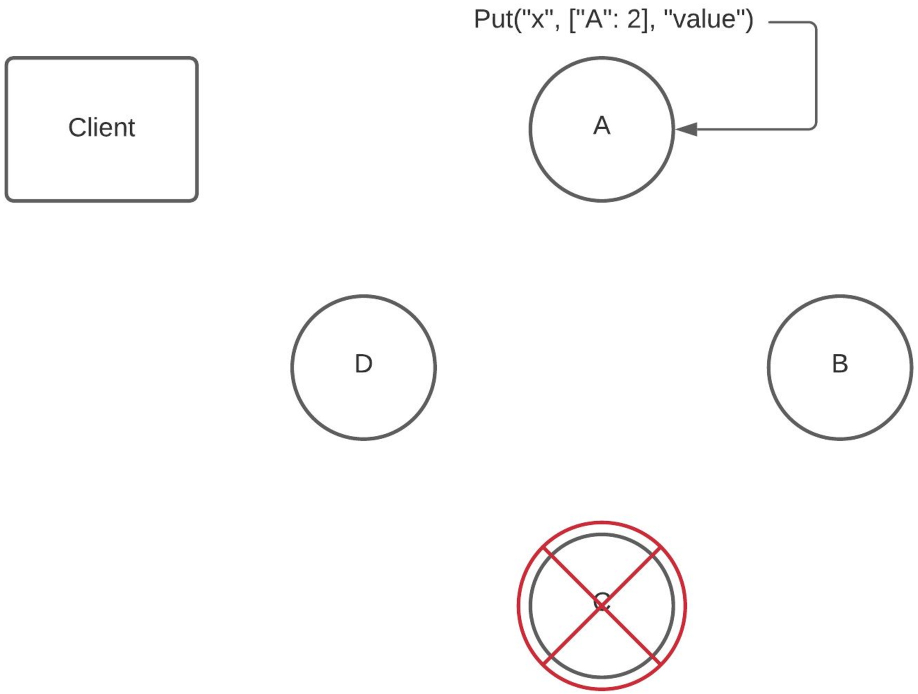
- Replicates this node's key/value store to all other nodes.
- Each individual key replication should look pretty similar to Put to a single node
- Not two-way
    - The node running Gossip will not update its own store

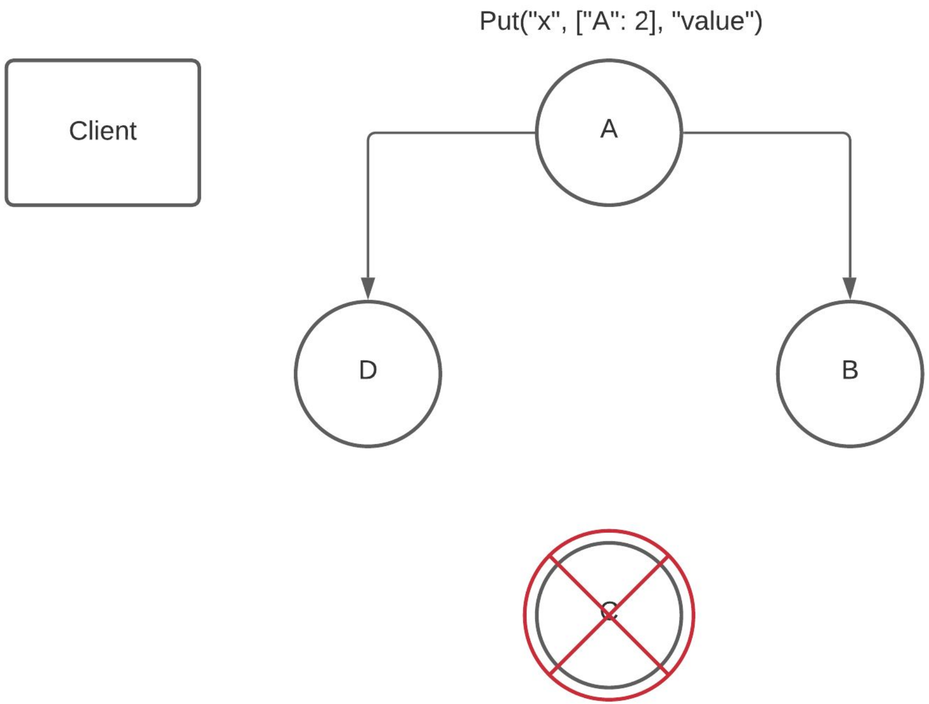
Crash

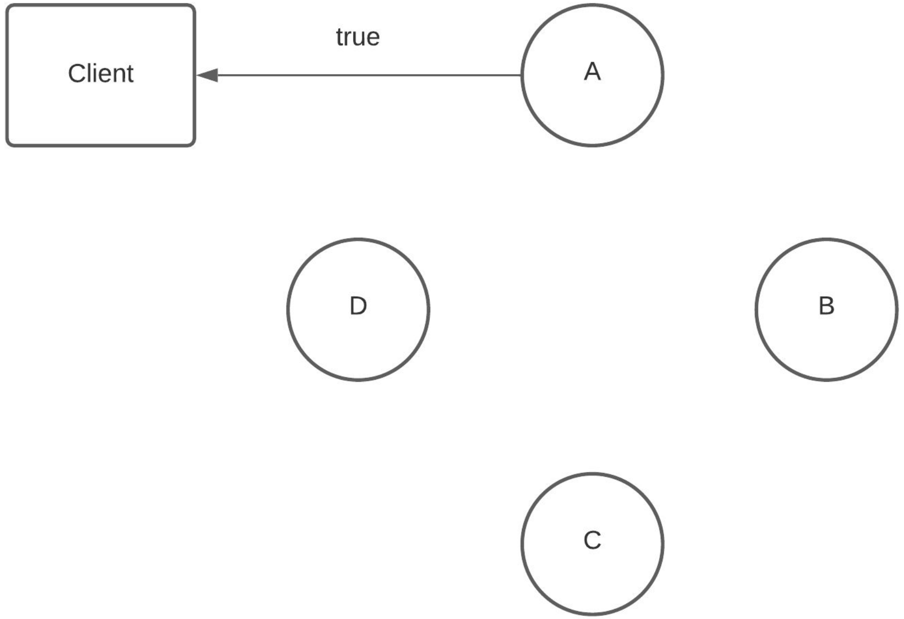
- Makes this node unresponsive to other RPC queries for some amount of seconds
- How to implement?
  - Likely have to keep some kind of crash state.
  - Crash state changes in only one place, but is checked everywhere.

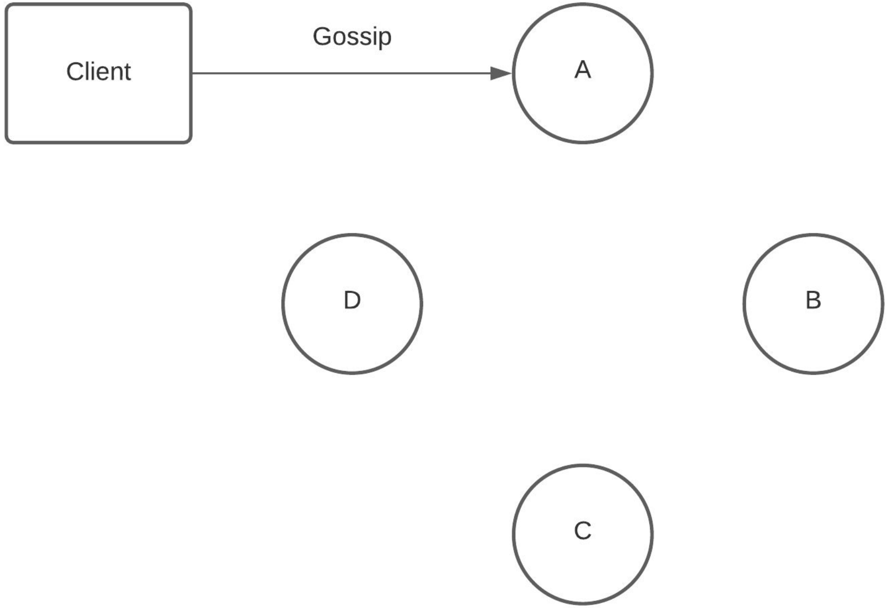

Process
Put("x",["A": 1], "value")
A
D
B
C

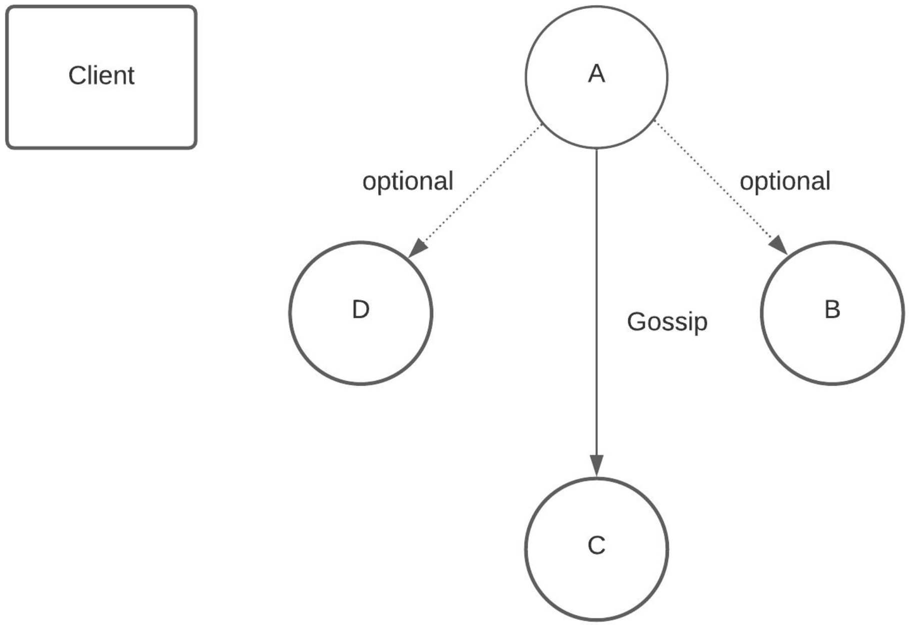

Put("x", ["A": 2], "value")
Client
A
D
B
C


Put("x", ["A": 2], "value")
Client
A
D
B
C


Put("x", ["A": 2], "value")
Client
A
D
B
C


Client
true
A
D
B
C


Client
Gossip
A
D
B
C


Client
A
optional
optional
D
Gossip
B
C

FAQ

- Failures in DynamoCoordinator
  - Most likely server isn't spun up when DynamoCoordinator tries to send preference list
  - Try adding a small wait
- Can I add struct fields/functions?
  - Yes! In fact, you will most likely need to do so.

Tips

- Don't worry about W > 1, R > 1 for your first iteration. Ensure your implementation works for W = 1, R = 1, then add functionality for W > 1, R > 1
- Use unit testing extensively. TDD is perhaps helpful here.
- Pay attention to your logic for determining causality: most of your bugs will likely occur here.