CSCI 1380 : Day 10

Today

Tapestry Conclusion

Time

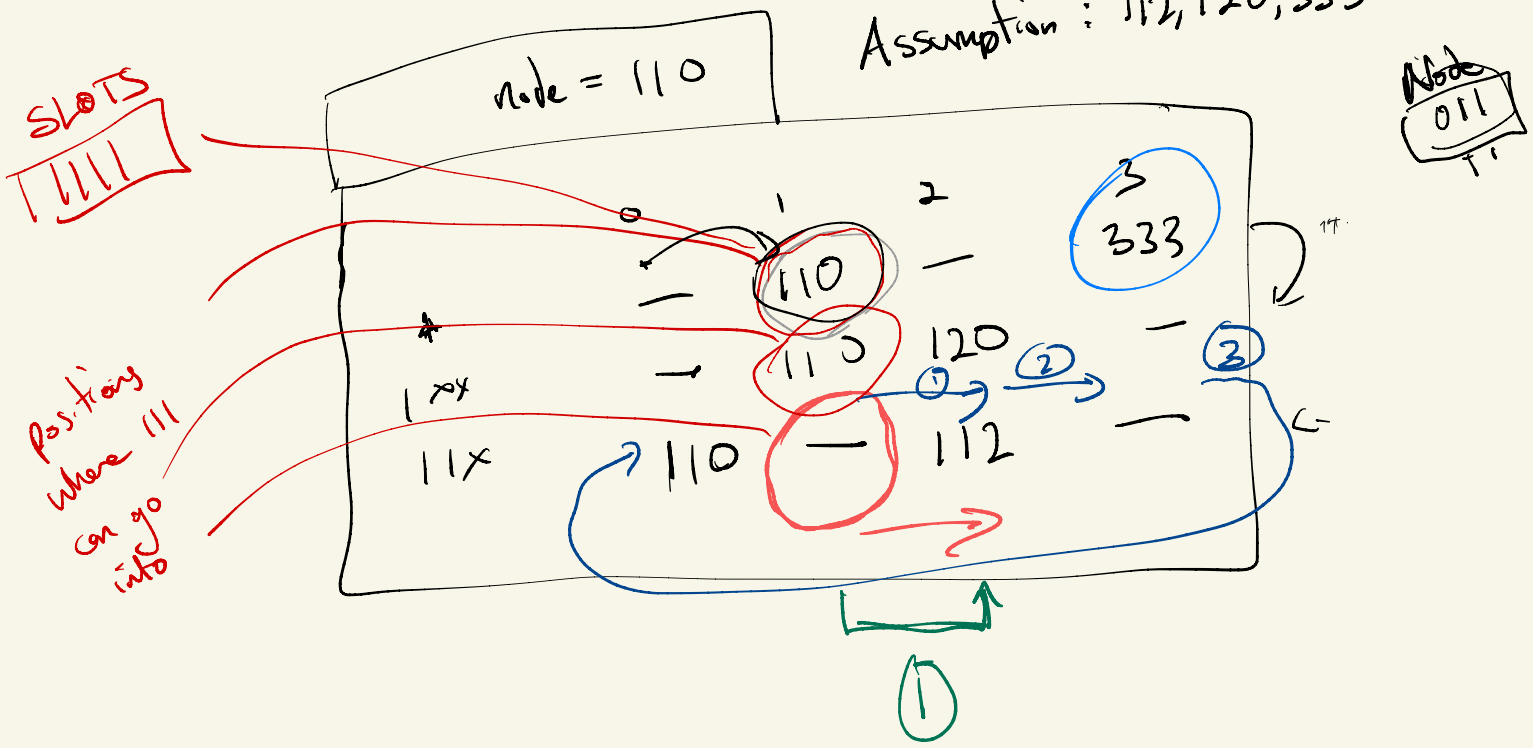
* why real time is bad
* logical clock (used in YugaDB / CockroachDB)
* Vector clocks (used in Amazon Dynamo)

Better Choice

Key confusion

$(2-3)\%4 \rightarrow (-1)\%4 \rightarrow 3$

Assumption : 112, 120, 333



Add a new Node (111)

① findRoot (112)

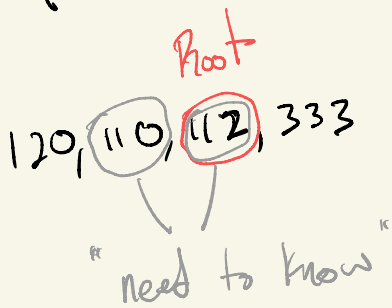
② Build Route Table

③ Inform "need-to-know" nodes that you've joined the network

④ optimize Route table (closer)

working tapestry

fast tapestry



"Need to know" → a current node in the network with a missing cell that this new node fills in

Prefix = (111, 112) ∈ 2

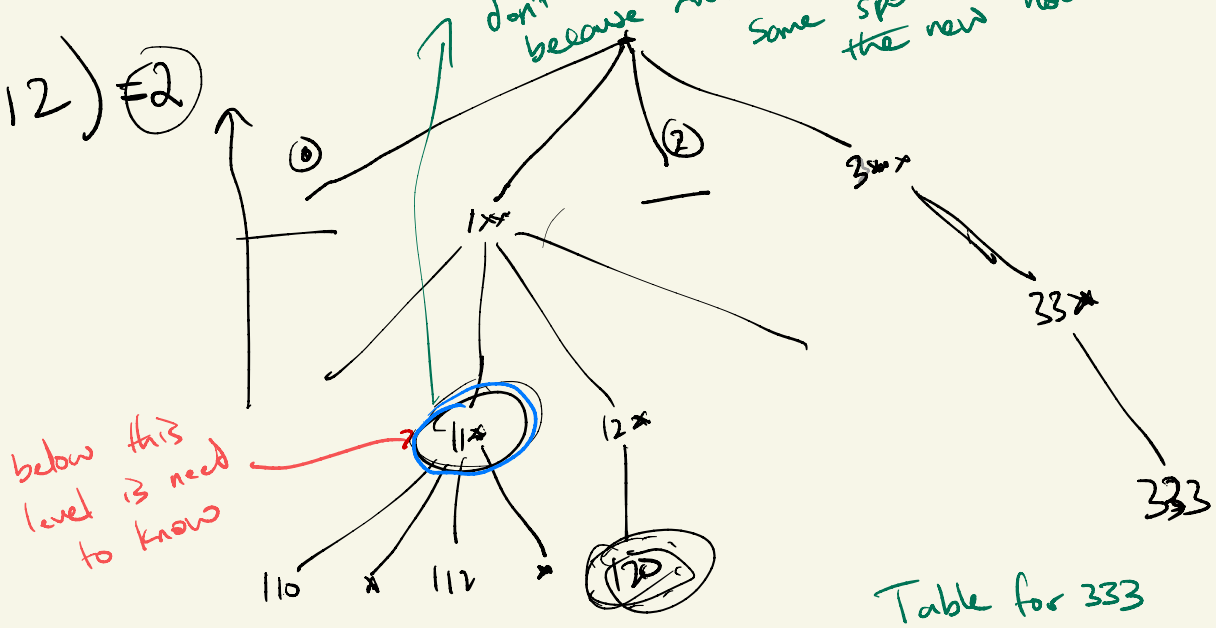
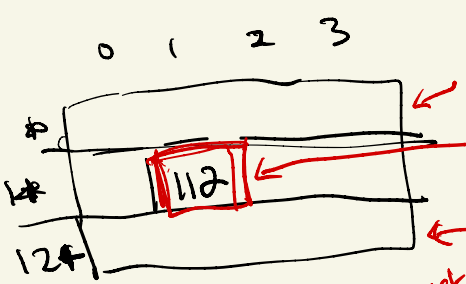


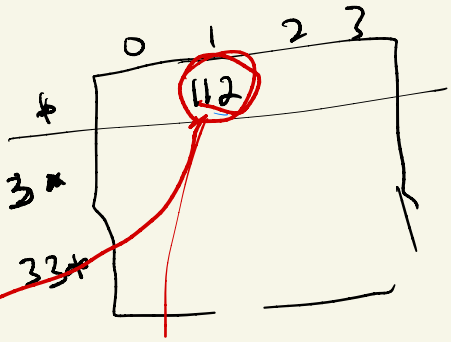
Table 120

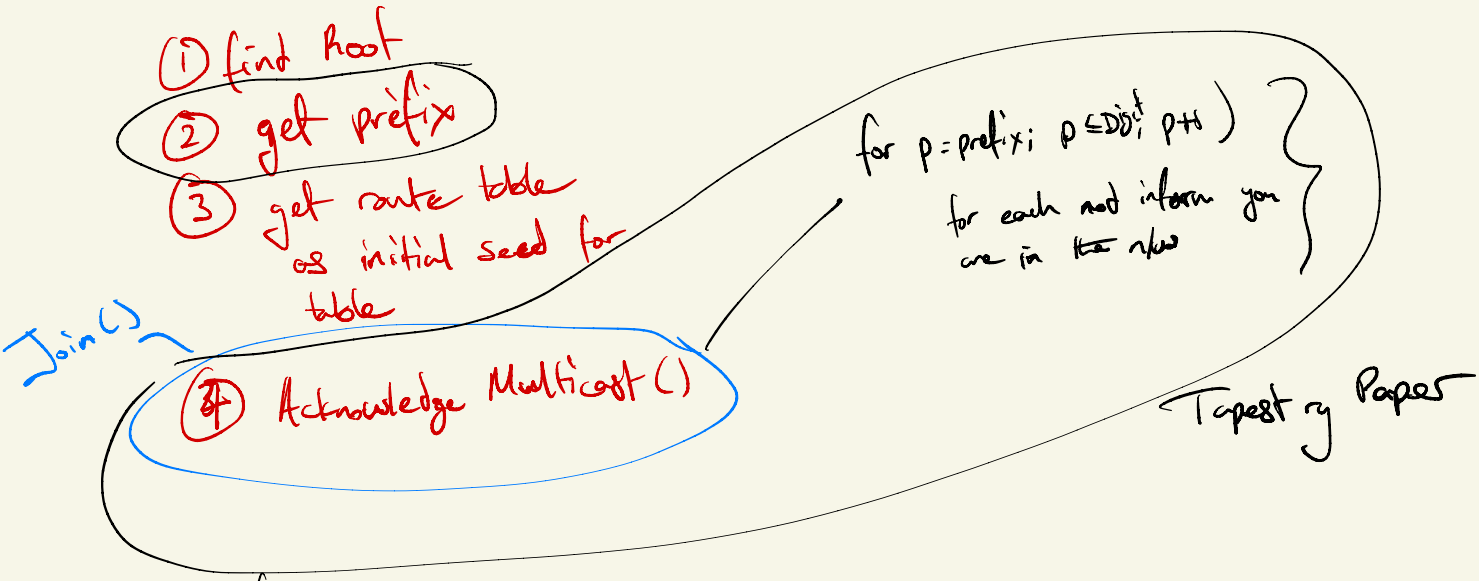


must start with 124

the only place 111 can go is here but root also meets the criteria and would likely go there first

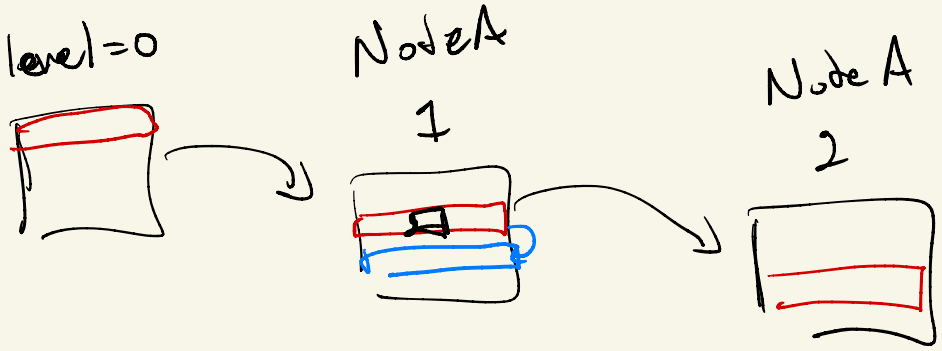
Table for 333






```
nodes = get nodes in Root's table
for (p = prefix; p >= 0; p--)
    backpointer = array()
    for each node in nodes {
        backpointer.push(node.getbackpointers((p))
    }
    local table.add(backpointer).
    nodes = merge(nodes, backpointer)
```

find Nexthop



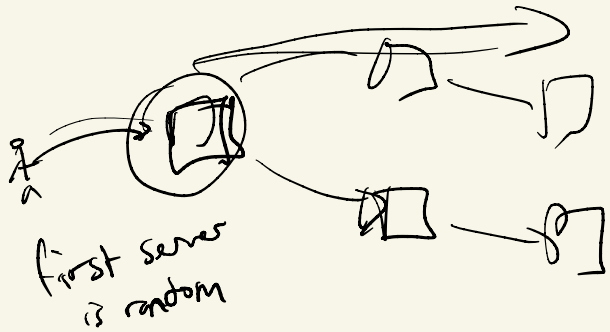
This is a recursive call from node to node where level changes between lookups

If the lookup happens at a row (say 1) of Node A but there is only one entry in the row, i.e., Node A is the only entry then you have two options

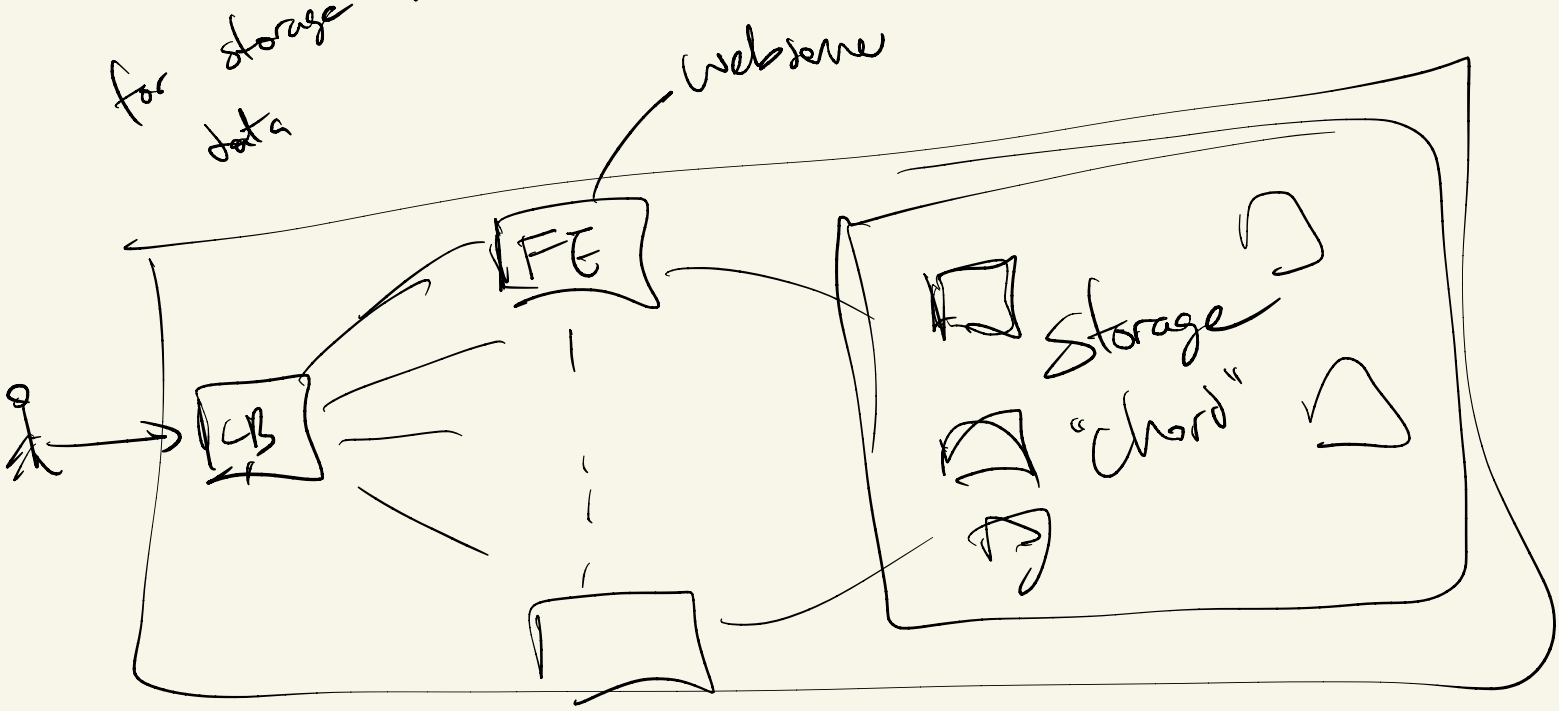
1. recursively call Node A & go down a Row
2. simply go down a Row

size of keyspace (0 .... 63) N=64

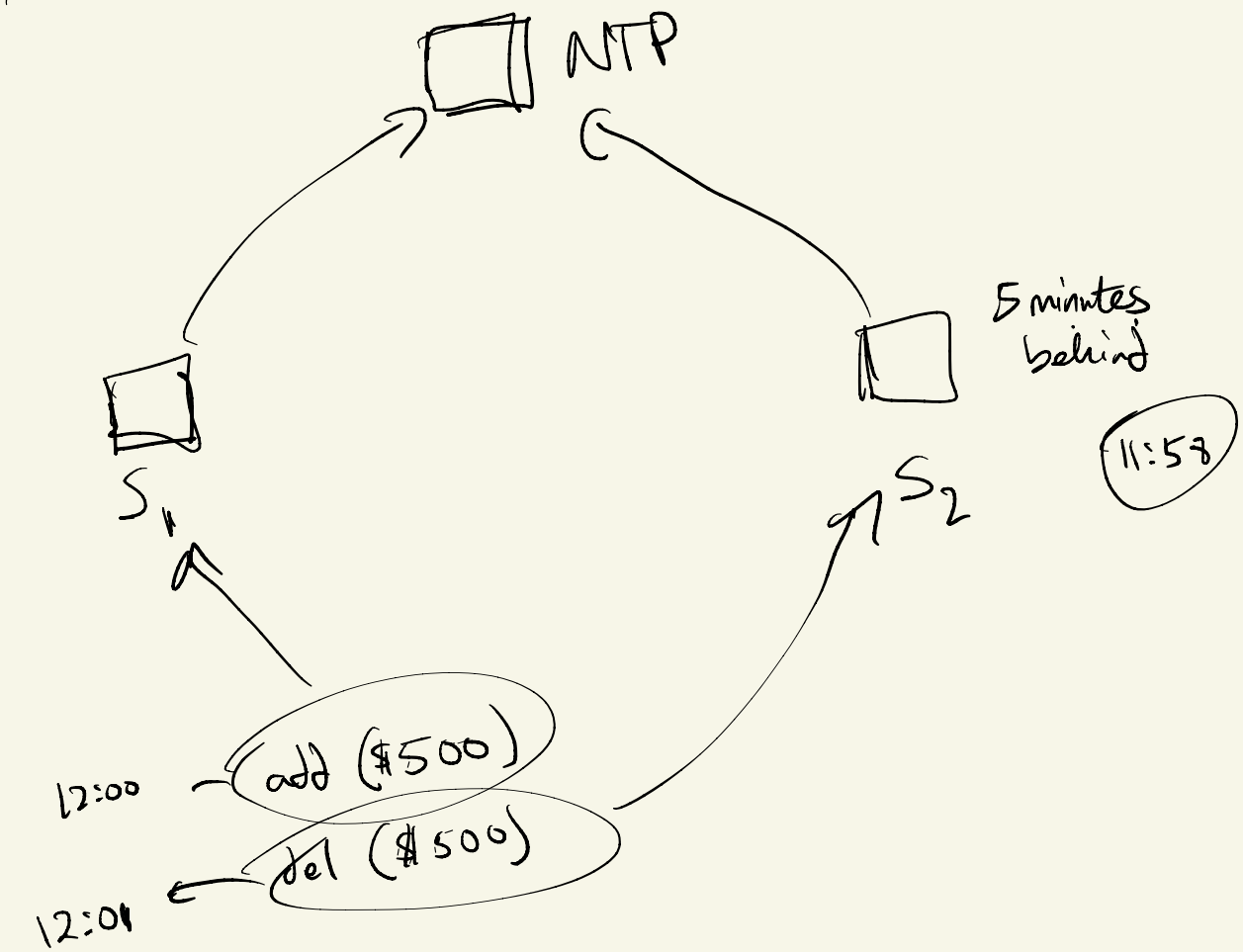
# of servers	Consistent hash	Chord	Tapestry
uses	LB	Storage	Storage
table size	$\|S\|$	$\log_2(N)$	$B(\log_B(N))$
# of servers to lookup	$O(1)$	$O(\log_2(N))$	$O(\log_B(N))$



Azure's Service Fabric for storage of microservice data



Time



Conclusion

1. Discussed common problems
   a. Closer
   b. findNextHop
   c. Join()
2. Compared different consistent hash implementations
3. started a discussion on Time