Bayou / Chord

2025 Spring

[Adapted from Andrew Or and Ion Stoica's original slides]

Context on Bayou: Disconnected Nodes [Stoica]

Early days: Nodes always on when not crashed

- Network bandwidth always plentiful
- Never needed to work on a disconnected node

Now: Nodes **detach** then **reconnect** elsewhere

- Even when attached, bandwidth is variable
- Reconnection elsewhere means often talking to different replica
- Work done on detached nodes

Bayou

- ”Replicated, [eventually] consistent storage system designed for portable machines with less-than-ideal network connectivity.”
- System developed at PARC in the mid-90’s
- First coherent attempt to fully address the problem of disconnected operation

Bayou – Goals

- Maximize **availability**, support offline collaboration
- **Minimize network** communication
- Agree on all values (**eventually consistent**)

Bayou Update Protocol: Review from Class

- Client sends updates to a server
- Each update is uniquely identified by: ***⟨Commit Sequence Number (CSN), Local Timestamp, Node ID⟩***

- Updates can be **committed** or **tentative**
    - CSNs increase monotonically (Tentative updates have commit-stamp = ∞)

- Only the Primary server can commit updates
    - Assigns CSNs in monotonically increasing order
    - CSN ≠ Timestamp

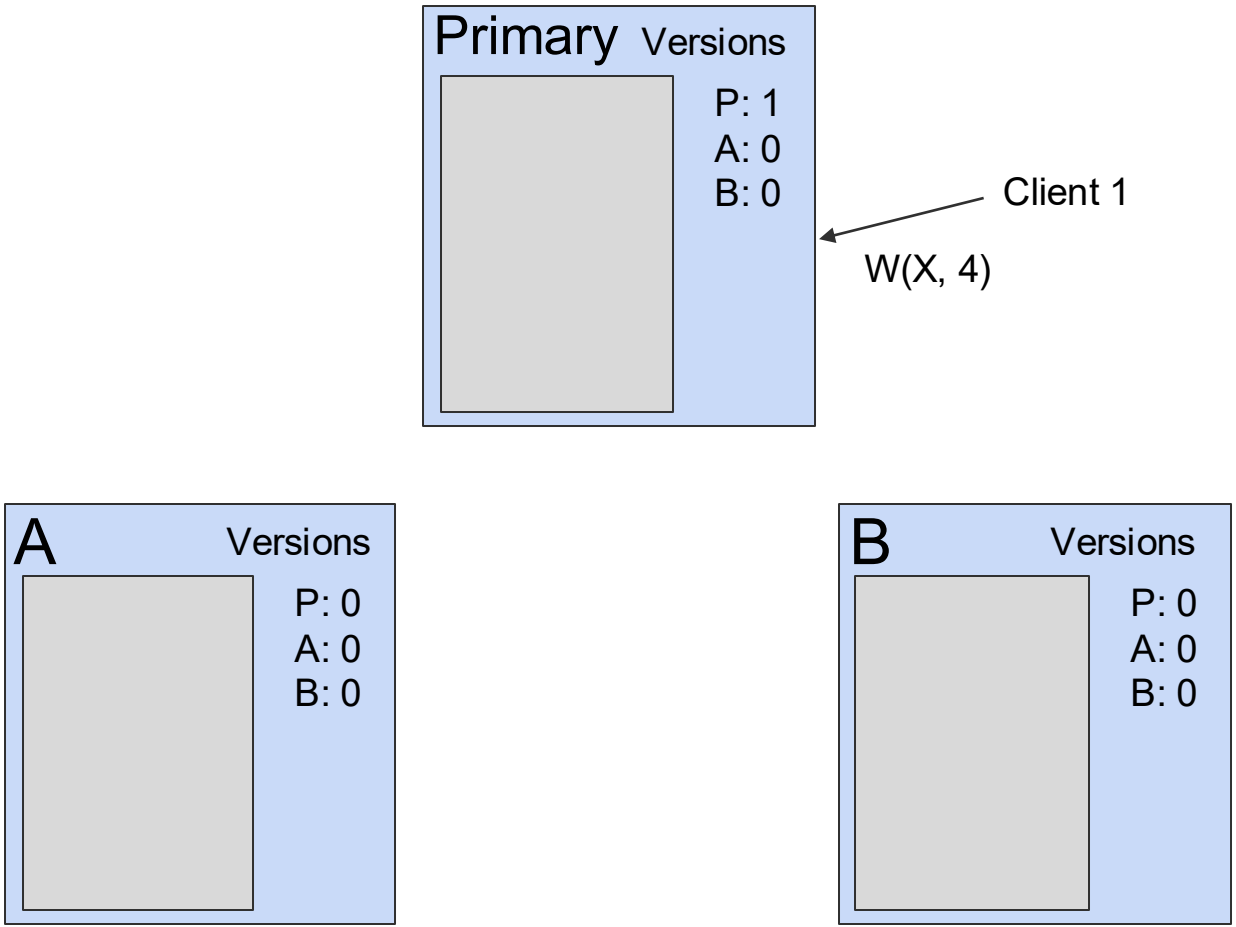
Anti-Entropy Exchange

- Each server keeps a version vector:
  - R.V[X] is the latest timestamp from server X that server R has seen
- When two servers connect, exchanging the version vectors allows them to identify the missing updates
- These updates are exchanged in the order of the logs, so that if the connection is dropped the crucial monotonicity property still holds
  - If a server X has an update accepted by server Y, server X has all previous updates accepted by that server

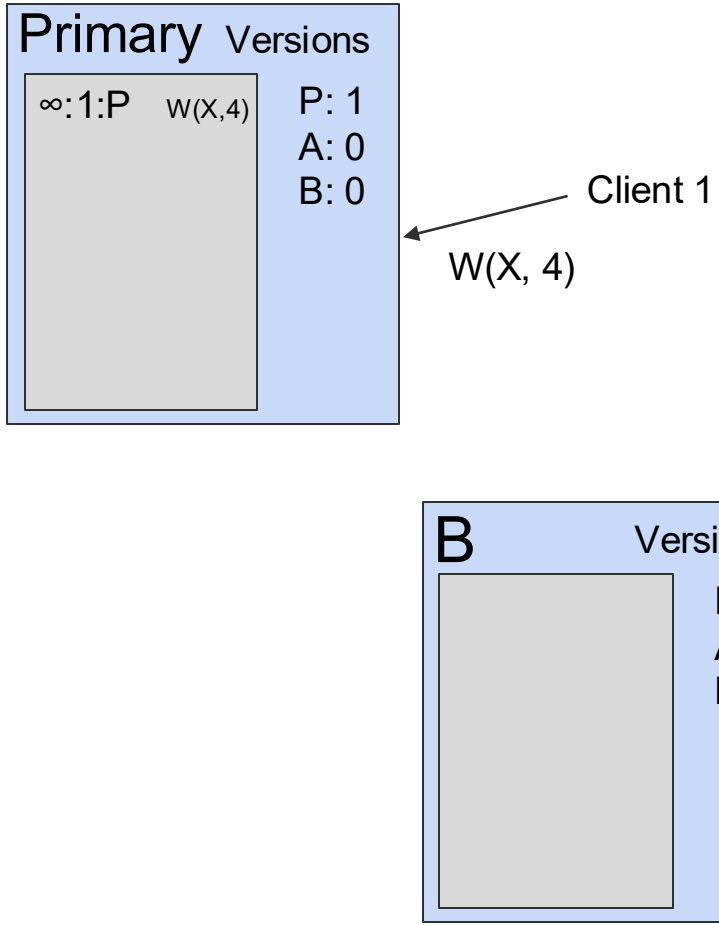
34

Source: Berkeley CS 268 Lecture 20 Slide Deck

Bayou Writes

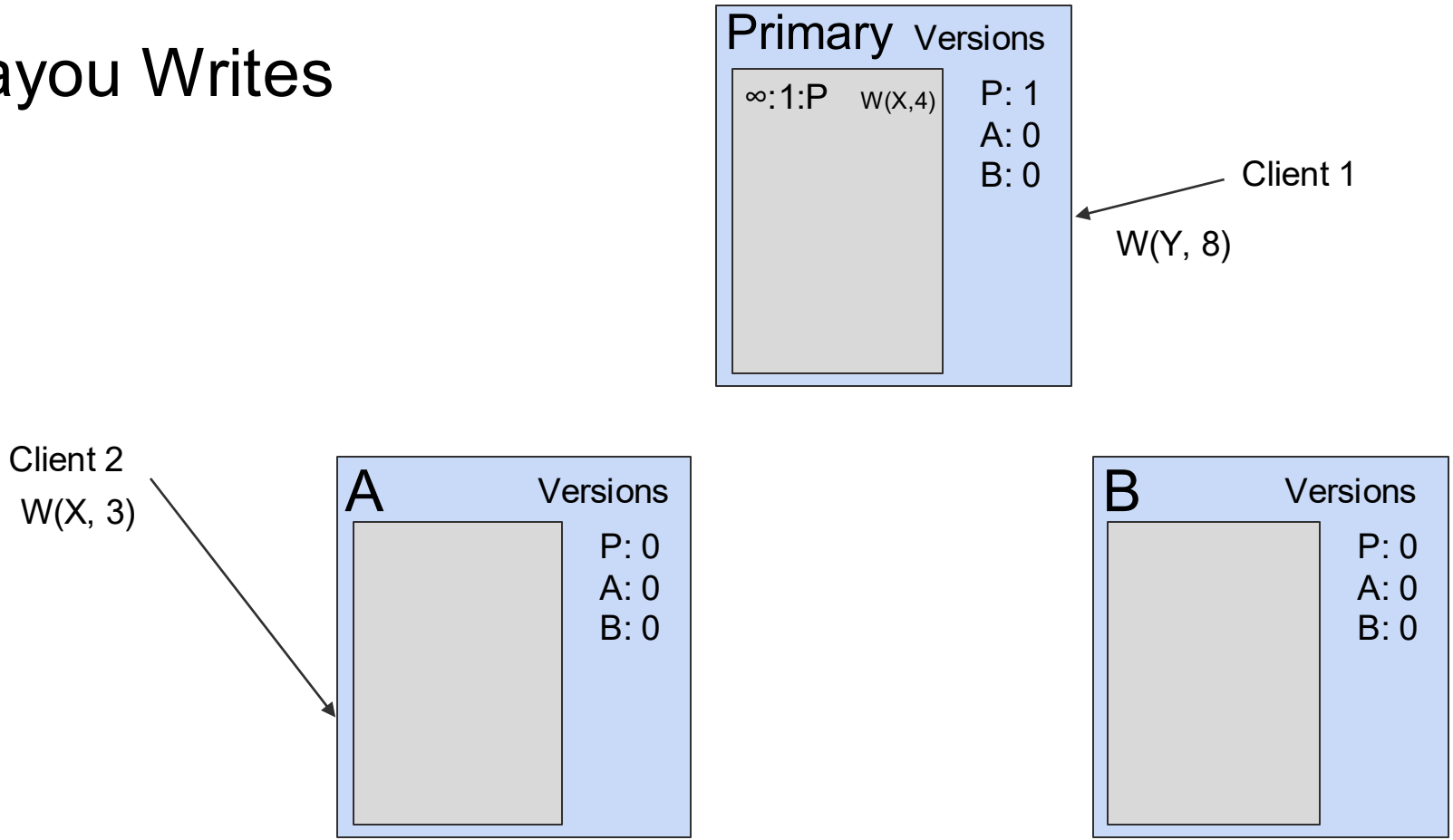


Bayou Writes



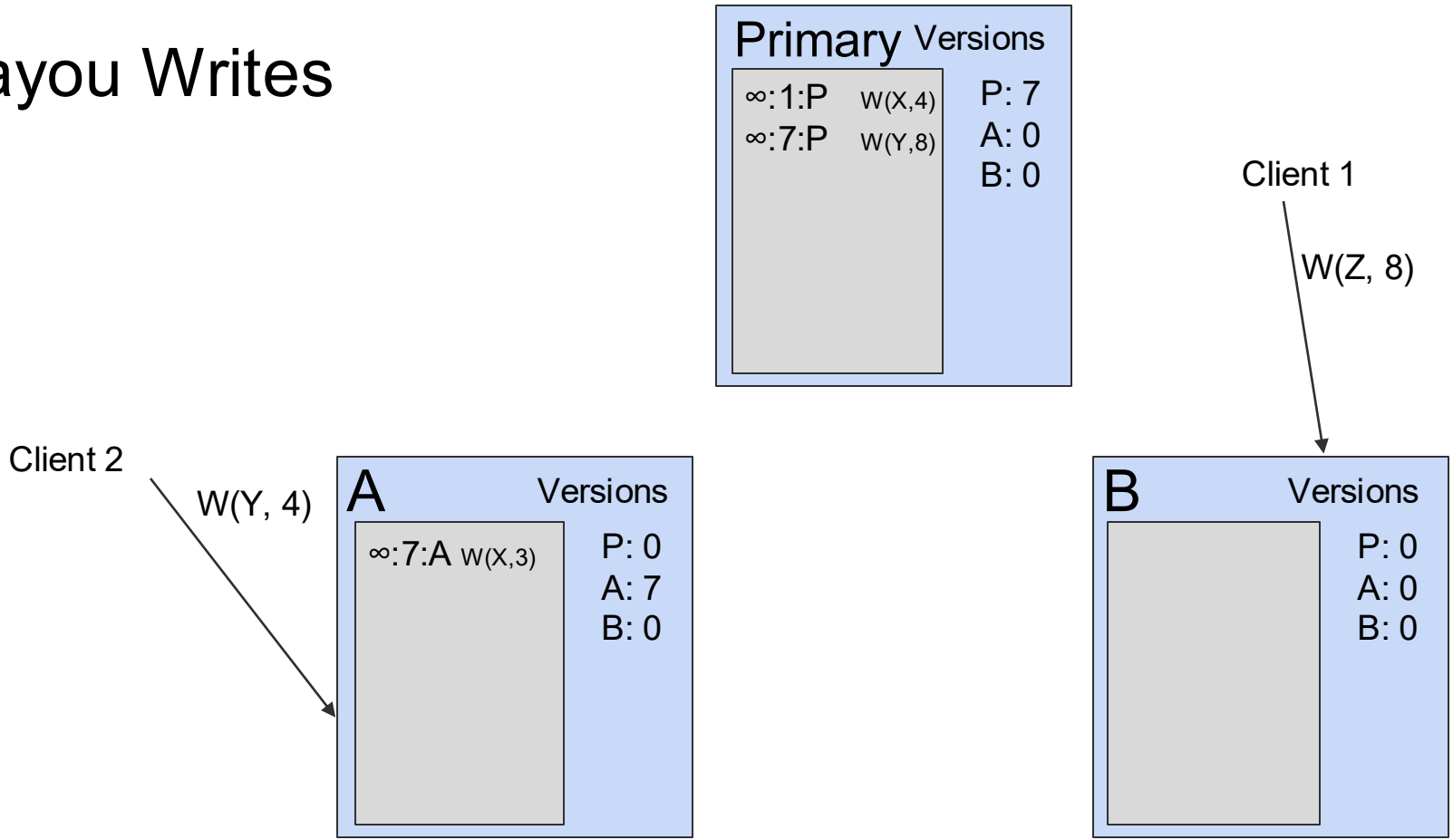
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Writes



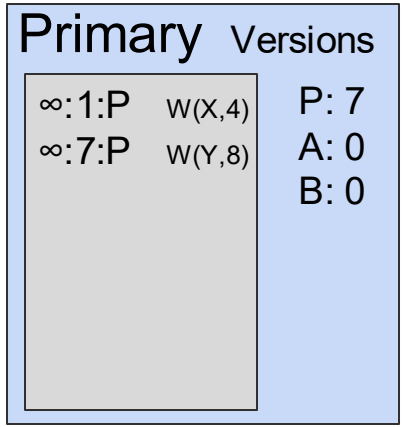
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

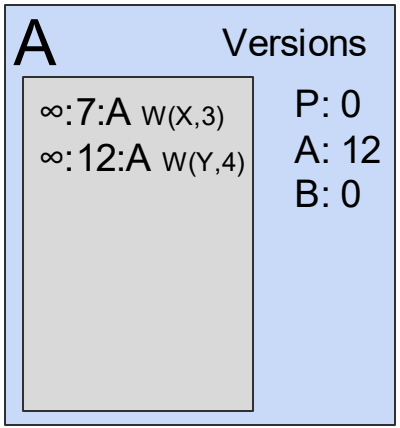
Bayou Writes

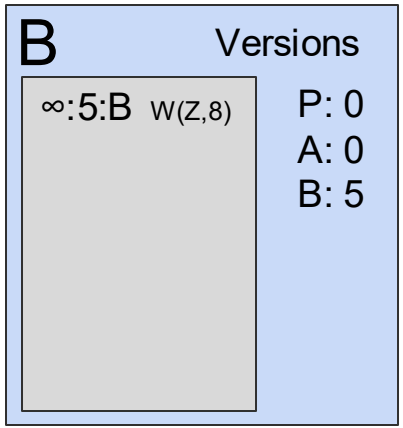


(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Writes



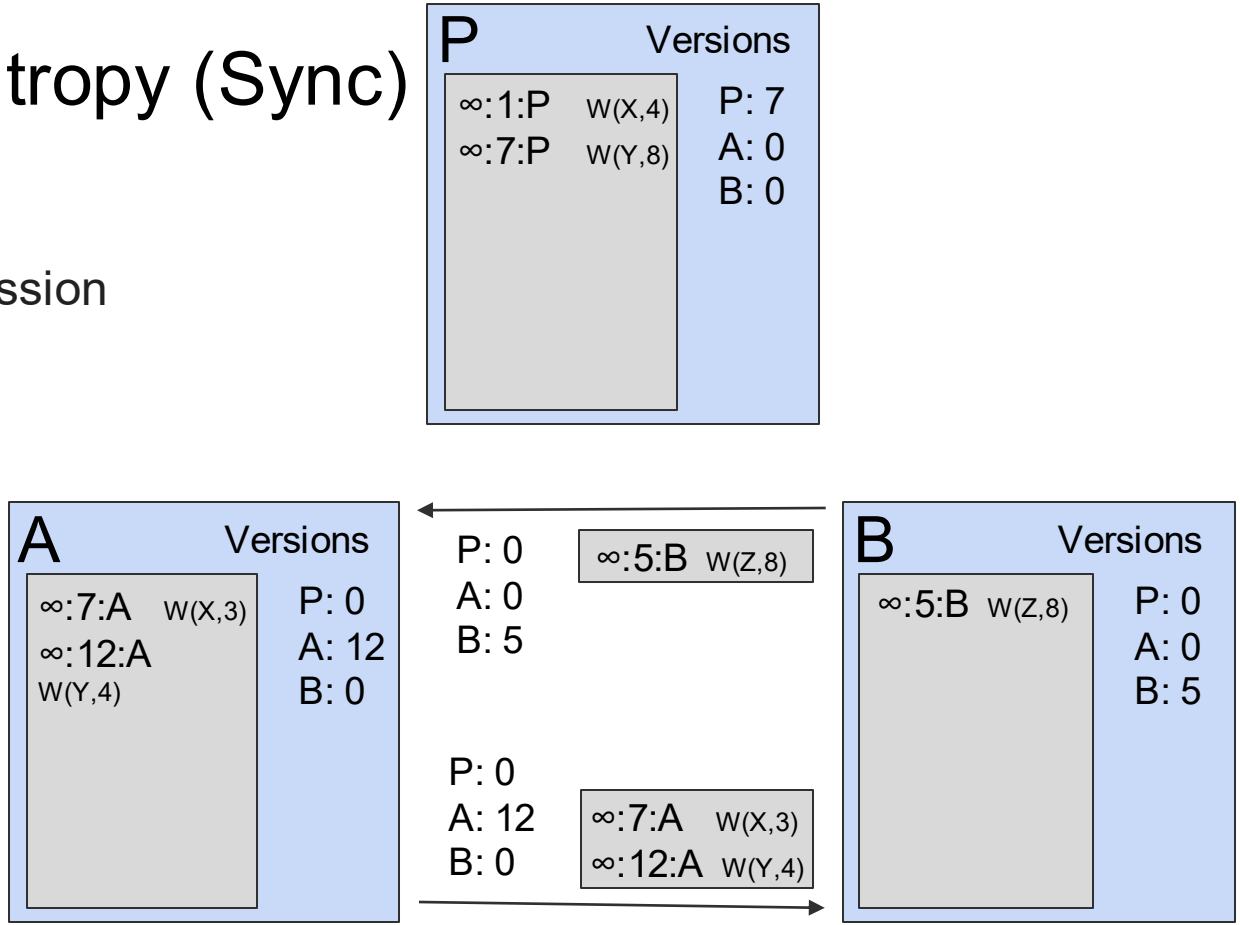




(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

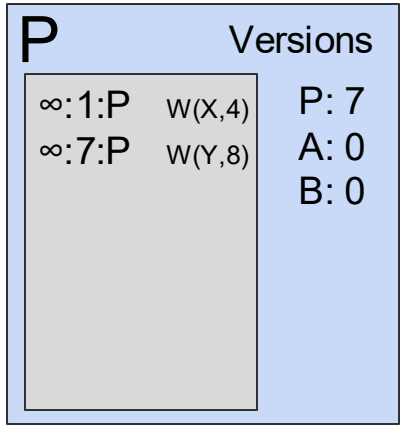
Bayou Anti-Entropy (Sync)

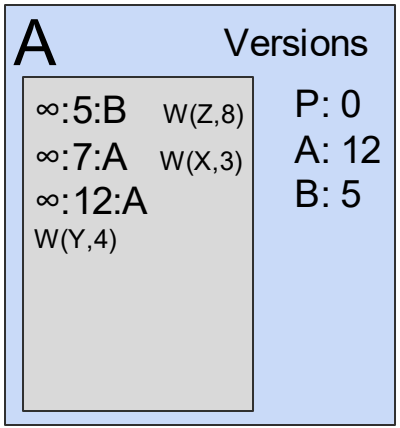
Anti-entropy Session
A & B

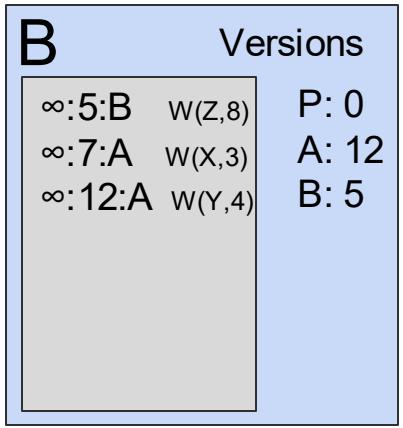


(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Anti-Entropy (Sync)



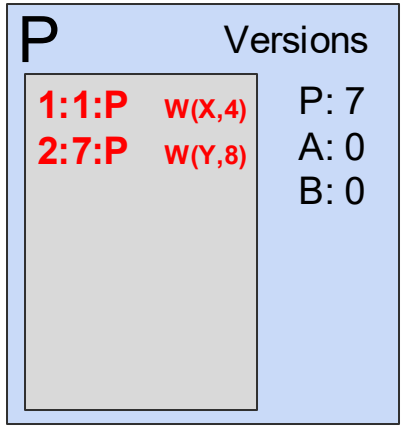


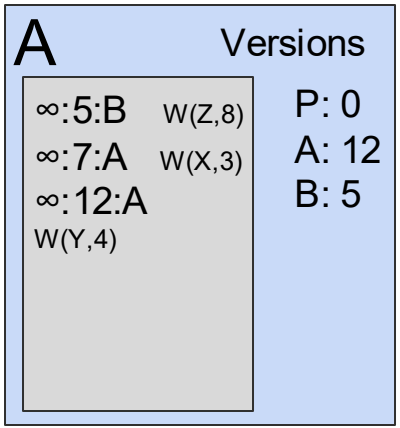


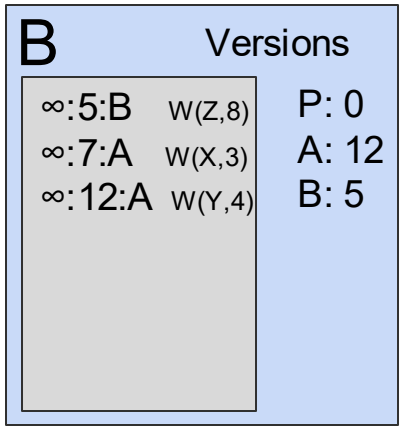
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Commit

Primary commits its entries



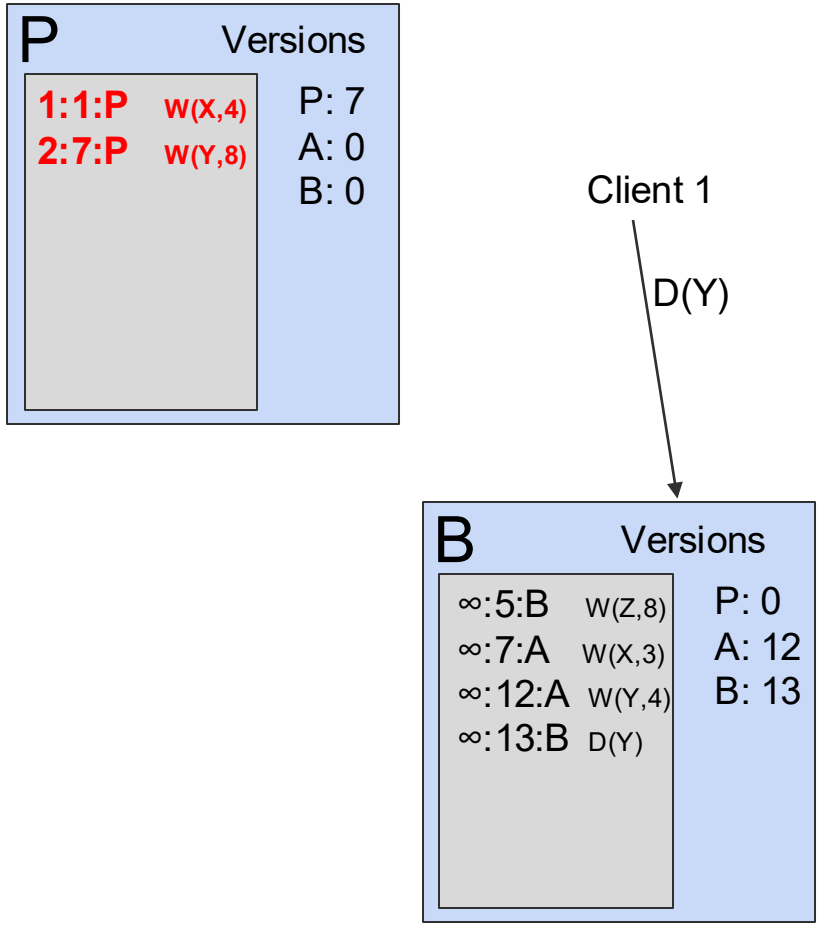




(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Write

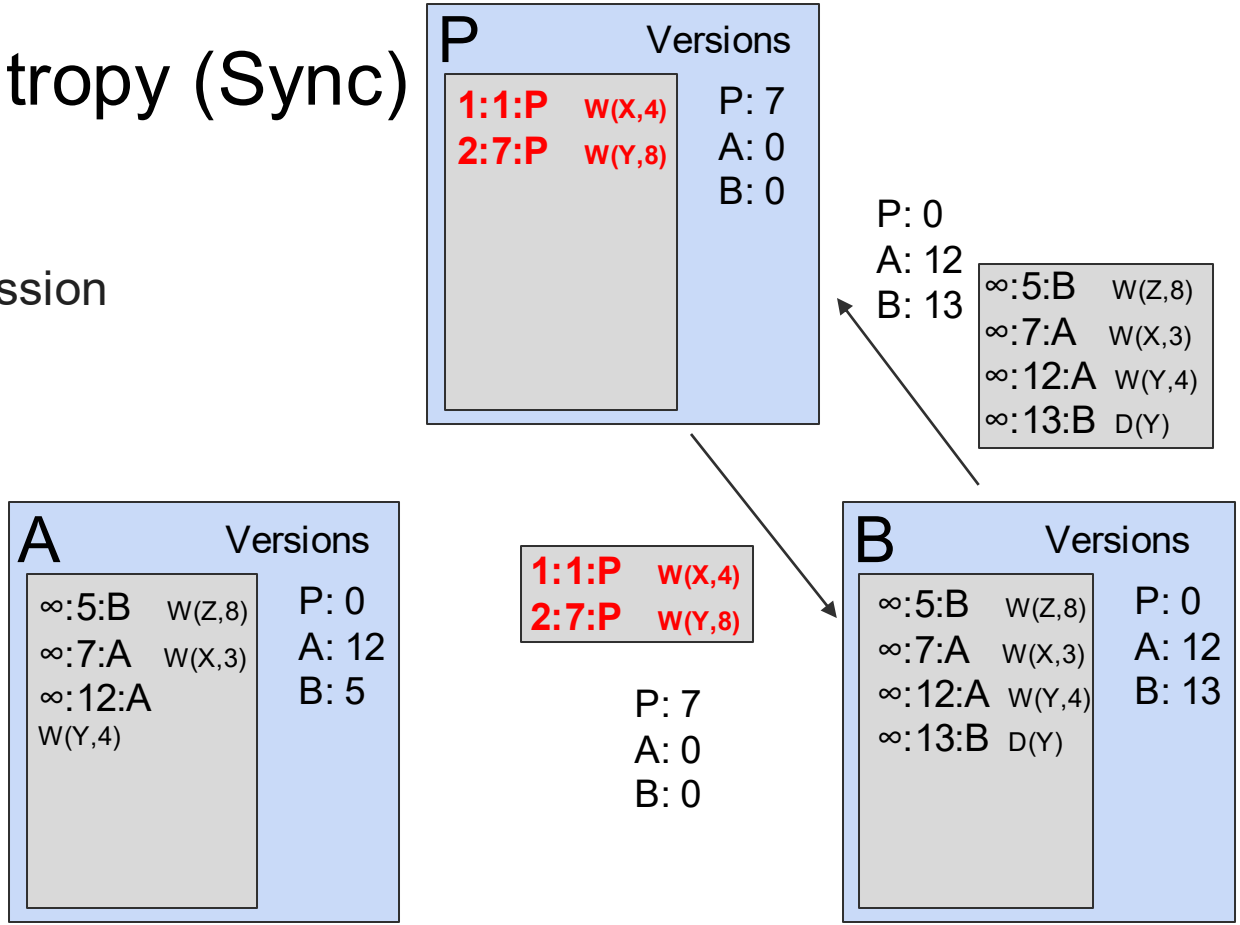
Write after anti-entropy session
Write timestamp = max(clock, max(TS)+1)



(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

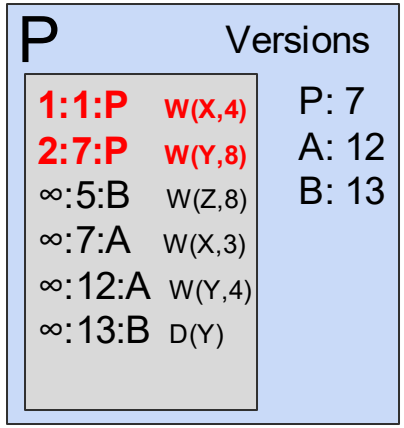
Bayou Anti-Entropy (Sync)

Anti-entropy Session
P & B

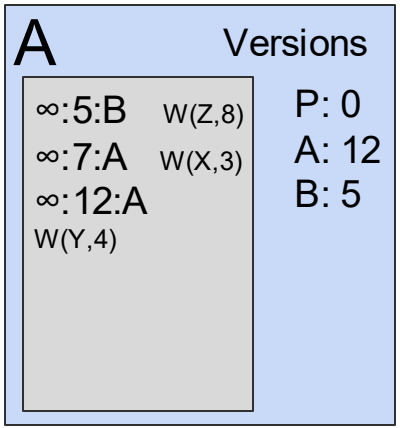


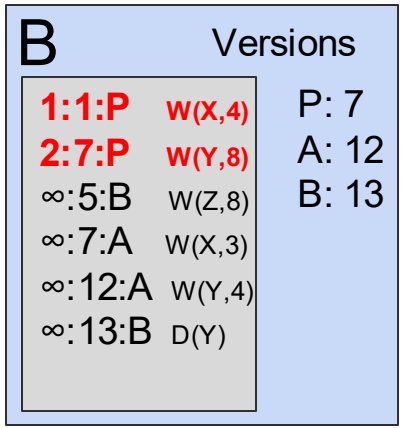
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Anti-Entropy



Primary respects causality!

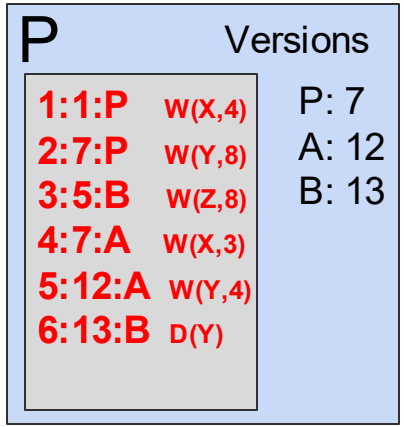


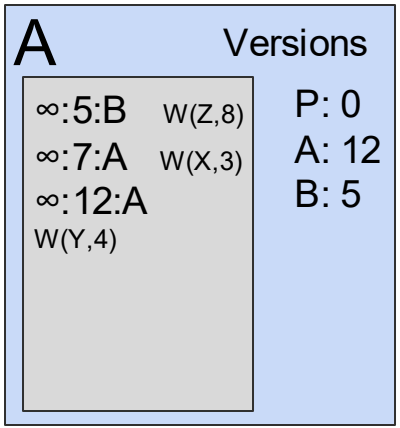


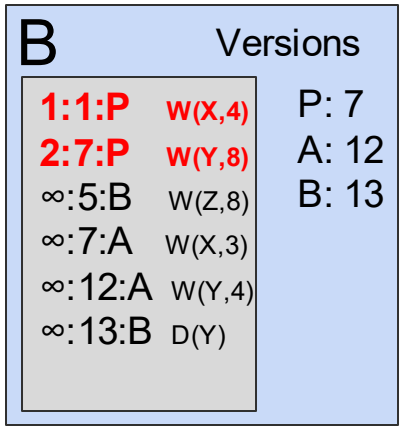
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou Commit

Primary commits its entries



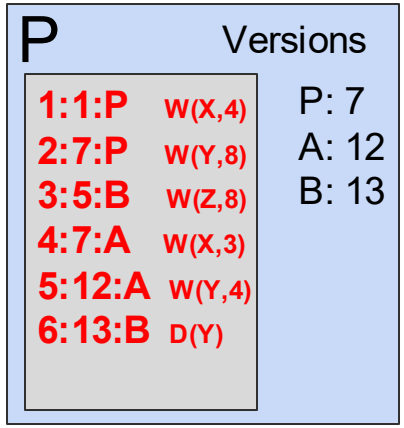


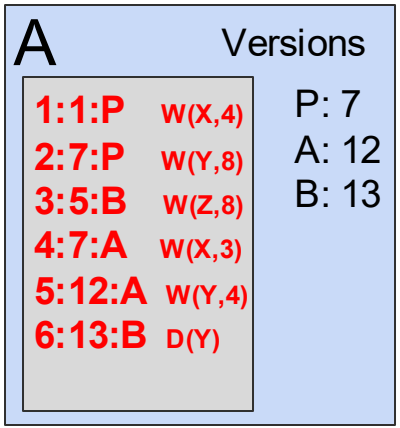


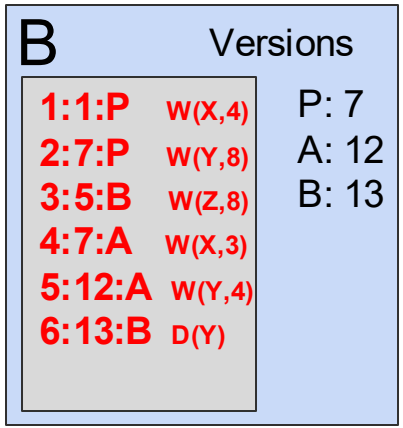
(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou

After a number of commits and anti-entropy sessions (**without further writes**), all nodes **converge** on the same state.







(value1 : value2 : value3) = (CSN : Local Timestamp : Node)

Bayou (review from class)

1. Eventual consistency: if updates stop, all replicas eventually have the same view.
2. Update functions for automatic app-driven conflict resolution.
3. Ordered update log is the real truth, not the DB.
4. Use Lamport clocks: eventual consistency that respects causality.

Context for Chord: Key Value Stores

Amazon:
- Key: customerID
- Value: customer profile (e.g., buying history, credit card, ..)

Facebook, Twitter:
- Key: UserID
- Value: user profile (e.g., posting history, photos, friends, …)

iCloud/iTunes:
- Key: Movie/song name
- Value: Movie, Song

Distributed file systems
- Key: Block ID
- Value: Block

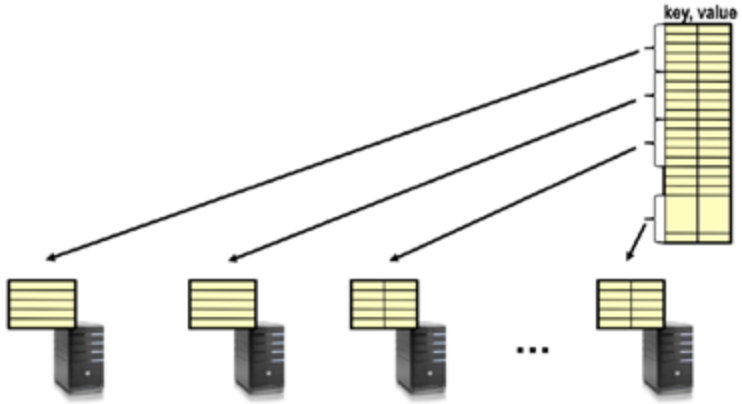
Credit: Ion Stoica's slide deck

Context for Chord: Key Value Stores

Key Value Store

Also called a Distributed Hash Table (DHT)

Main idea: partition set of key-values across many machines



Credit: Ion Stoica's slide deck

Chord

- Chord is a “distributed lookup protocol” for a peer-to-peer distributed hash table [Stoica ‘01]

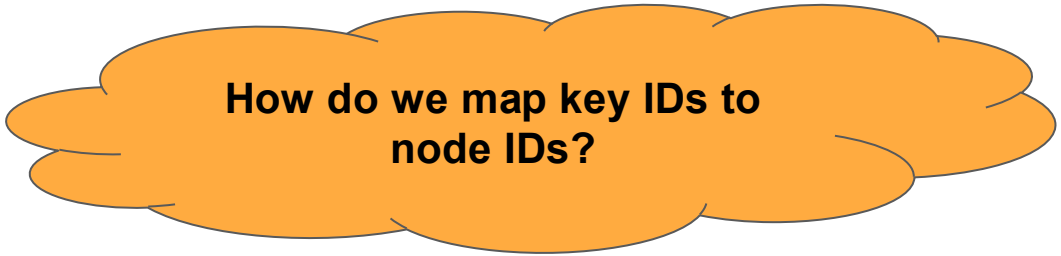
Main techniques:

- ***Consistent hashing*** for partitioning key space
- ***Finger Tables*** for lookup

Identifiers in Chord

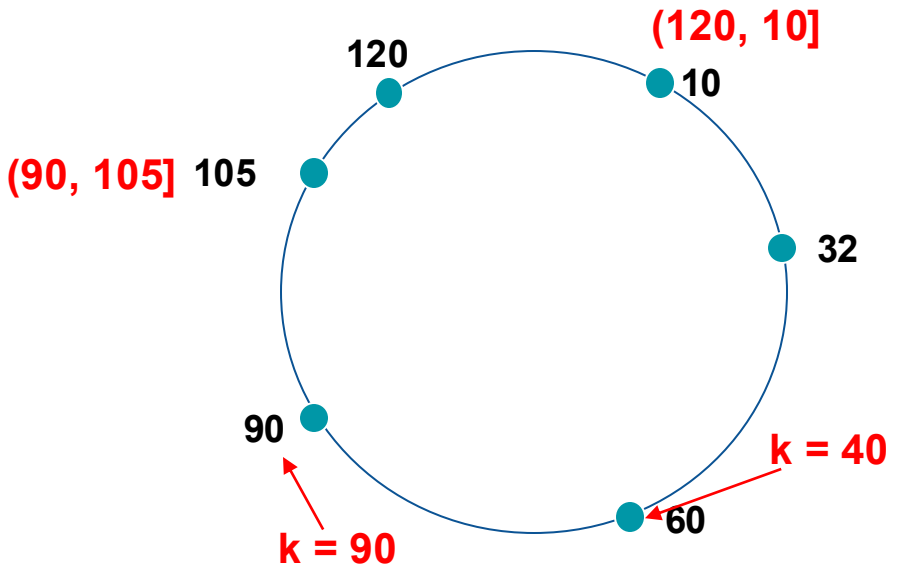
- Key identifier = SHA1(key) mod 2^m
- Node identifier = SHA1(IP address) mod 2^m
- Both are uniformly distributed in the same identifier space

- The identifier length, m, must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible (e.g. m = 160)



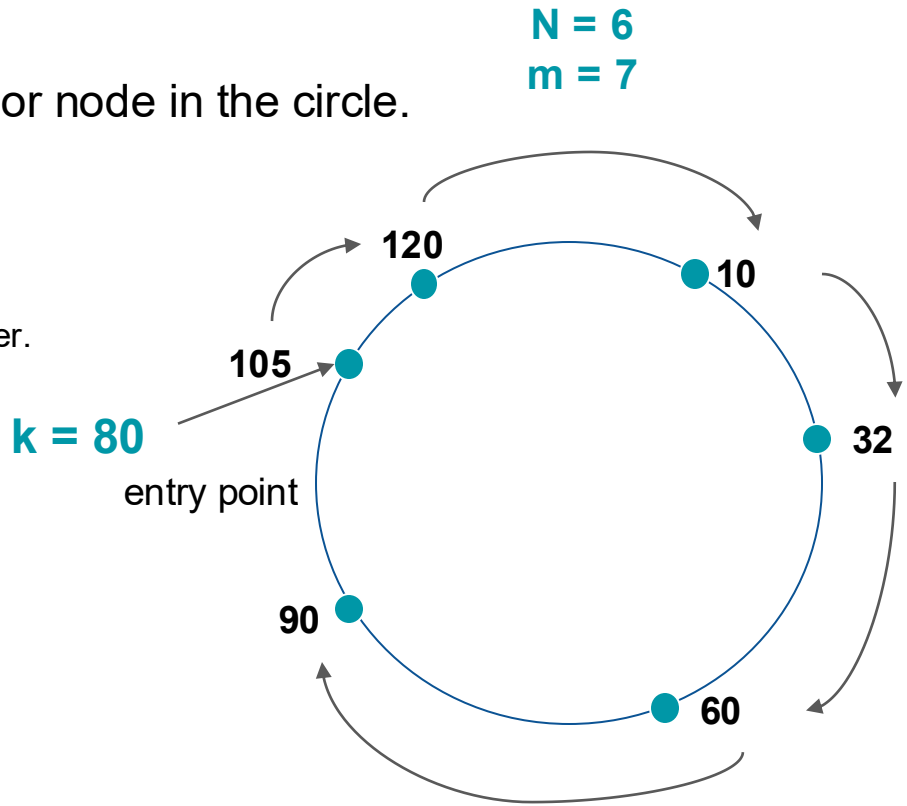
Consistent Hashing: Assigning Keys to Nodes

- A node owns the preceding key range, including its own identifier.
- Key k is stored at its successor node, the first node whose identifier is equal to or greater than the identifier of key k.



Basic Lookups

- Each node only remembers its successor node in the circle.
- Lookups in clockwise direction.
- Assume N nodes and K keys.
  - m: the number of bits in the node/key identifier.



What is the lookup time?

O(N) hops

Finger Table Notations

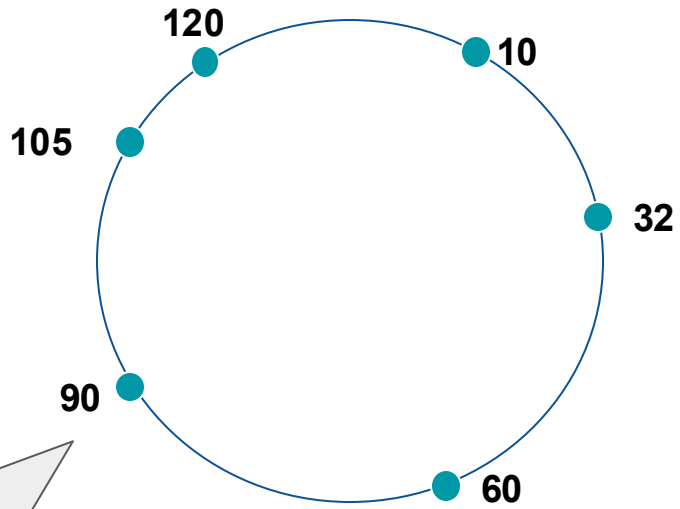
- Each node maintains additional routing information (e.g., finger tables) to accelerate lookups.
- A finger table contains m entries

Notation	Definition
$finger[k].start$	$(n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
$.interval$	$[finger[k].start, finger[k+1].start)$
$.node$	first node $\geq n.finger[k].start$

n: Identifier of the node
m: Number of bits in identifiers

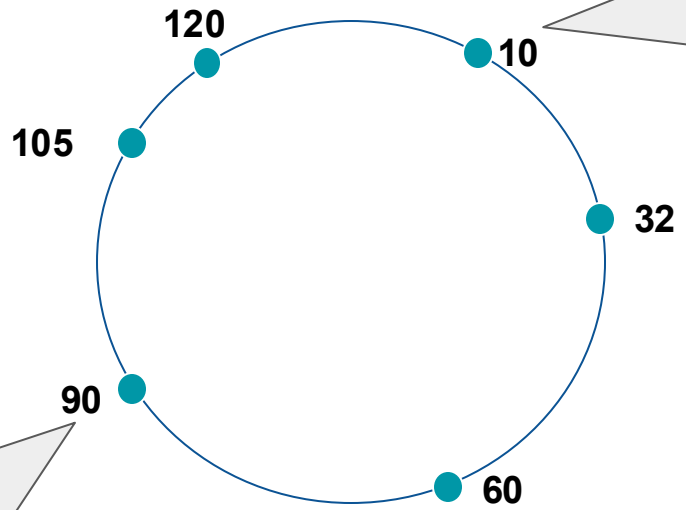
k = 1
k = 2
k = 3
k = 4
k = 5
k = 6
k = 7

start	interval	node
91	[91, 92)	105
92	[92, 94)	105
94	[94, 98)	105
98	[98, 106)	105
106	[106, 122)	120
122	[122, 26)	10
26	[26, 90)	32



Finger Table of Node 10

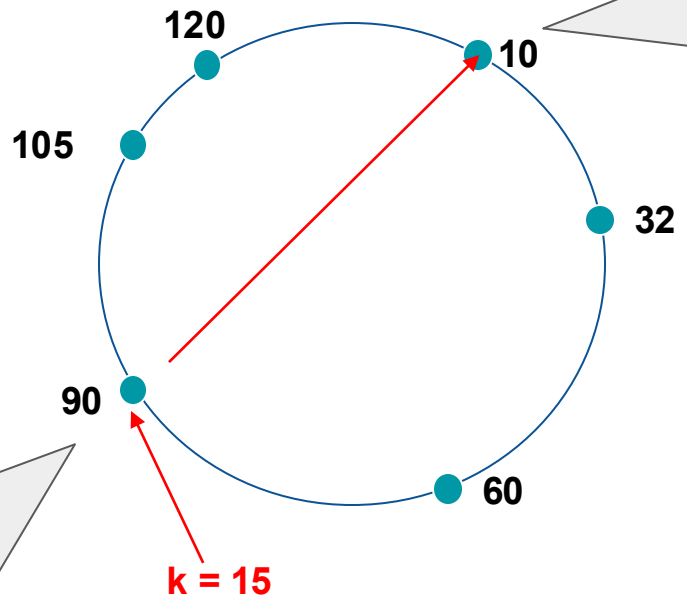
Notation	Definition
$finger[k].start$	$(n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
$.interval$	$[finger[k].start, finger[k+1].start)$
$.node$	first node $\geq n.finger[k].start$



start	interval	node
11	[11, 12)	32
12	[12, 14)	32
14	[14, 18)	32
18	[18, 26)	32
26	[26, 42)	32
42	[42, 74)	60
74	[74, 10)	90

start	interval	node
91	[91, 92)	105
92	[92, 94)	105
94	[94, 98)	105
98	[98, 106)	105
106	[106, 122)	120
122	[122, 26)	10
26	[26, 90)	32

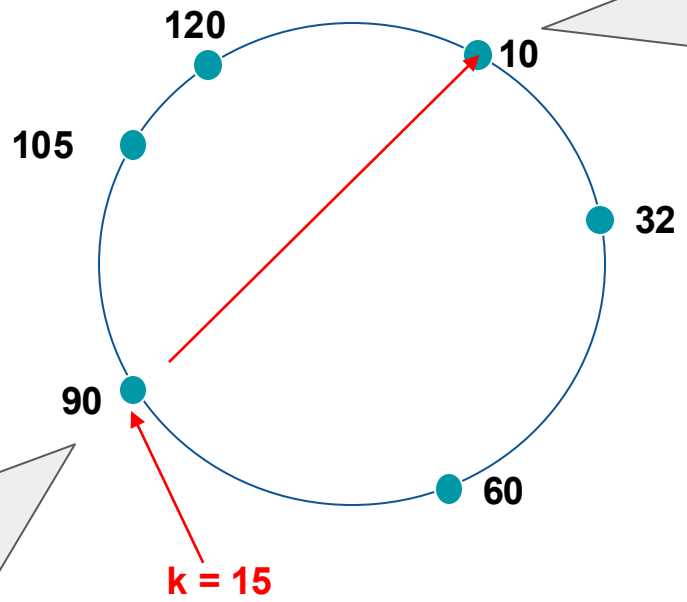
Route k = 15



start	interval	node
11		32
12		32
14		32
18		32
26		32
42		60
74		90

start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

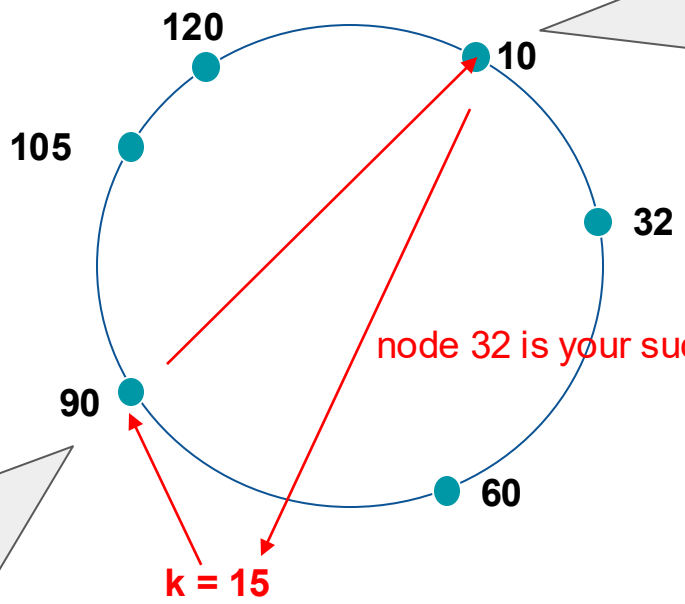
Route k = 15



start	interval	node
11		32
12		32
14		32
18		32
26		32
42		60
74		90

start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

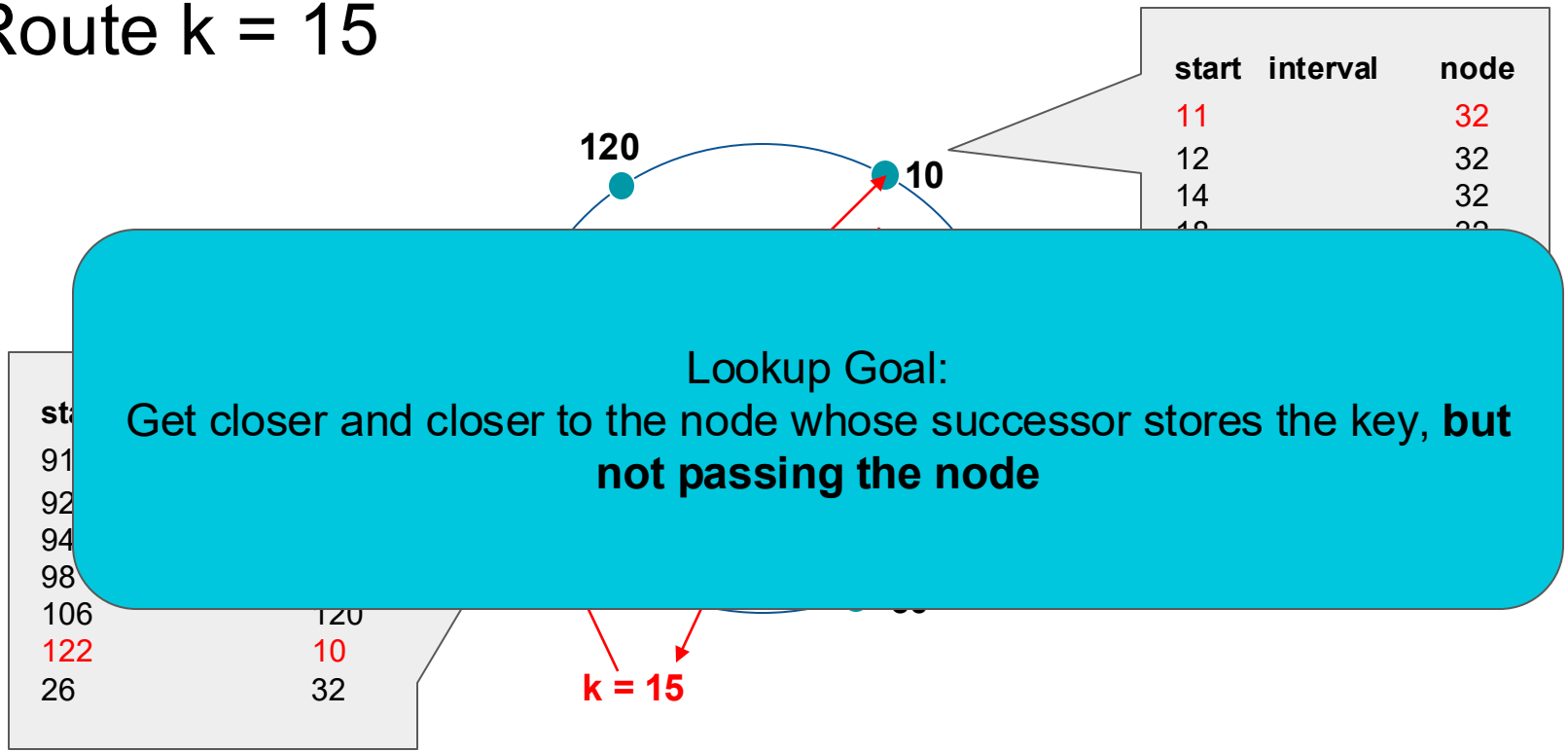
Route k = 15



start	interval	node
11		32
12		32
14		32
18		32
26		32
42		60
74		90

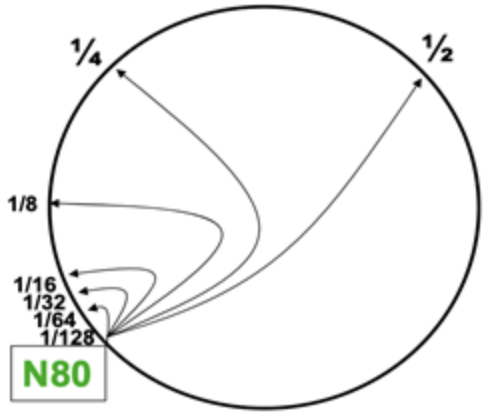
start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

Route k = 15


start interval node
11 32
12 32
14 32
120
10
k = 15
Lookup Goal:
Get closer and closer to the node whose successor stores the key, **but not passing the node**

Summary: Finger Table

- Each node maintains additional routing information (e.g., finger tables) to accelerate lookups.
  - This information is not essential for correctness, as long as the successor information is correct.
- A finger table contains m entries.
- We trade off **space for better lookup performance**



What is the lookup time?

O(logN) hops

Credit: UC Berkeley CS 262a Fall 2023 slide deck.