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

What is it?

- Weakly consistent, replicated storage system

Goals:

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

Bayou Update Protocol: Review from Class

- Client sends update to a server
- Updates uniquely identified by:
  <Commit Sequence Number (CSN), Local Timestamp, Node ID>

- Updates are either committed or tentative
  CSNs increase monotonically
  Tentative updates have commit-stamp = ∞

- Only Primary server can commit updates
  Allocates CSN in monotonically increasing order
  CSN is different from time-stamp

Anti-Entropy Exchange

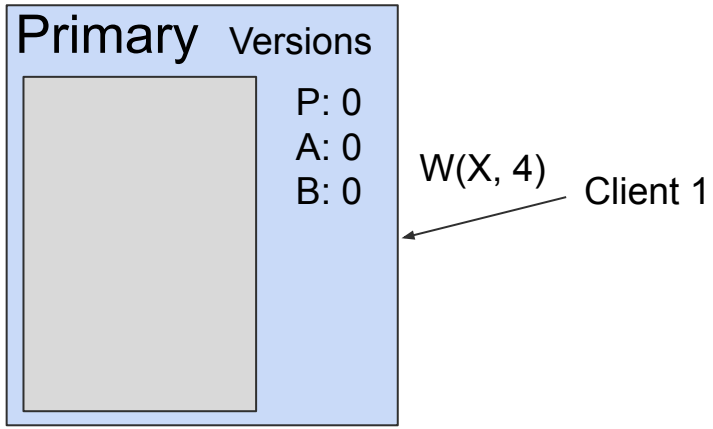
- Each server keeps a version vector:
  - R.V[X] is the latest timestamp from server X that server R has seen
- When two servers connect, exchanging the version vectors allows them to identify the missing updates
- These updates are exchanged in the order of the logs, so that if the connection is dropped 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

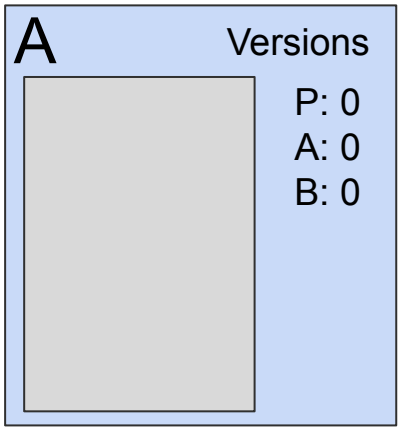
34

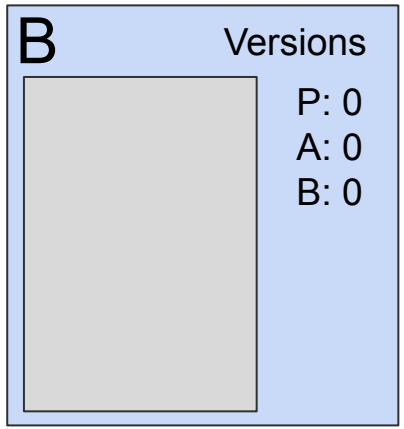
Source: Berkeley CS 268 Lecture 20 Slide Deck

Bayou Writes

value1 : value2 : value3 denotes
Commit Sequence Number (CSN) : Local Timestamp : Node ID

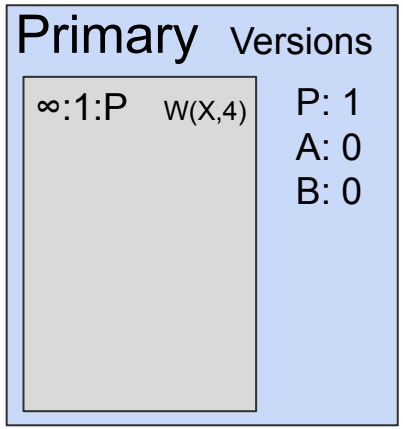




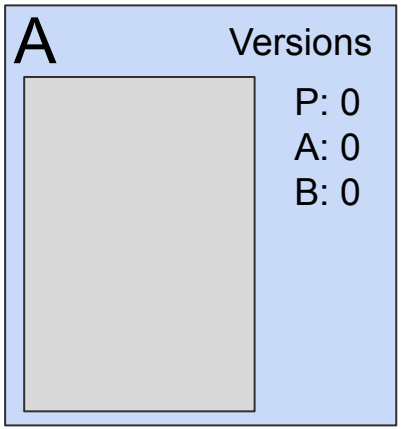


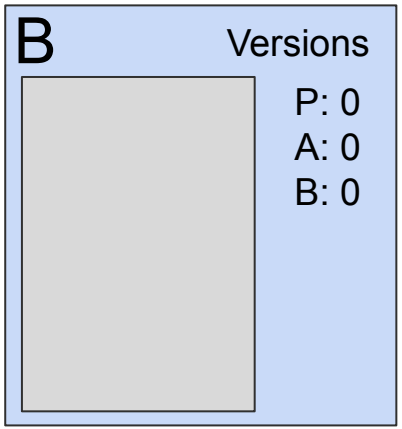
Bayou Writes

value1 : value2 : value3 denotes
Commit Sequence Number (CSN) : Local Timestamp : Node ID



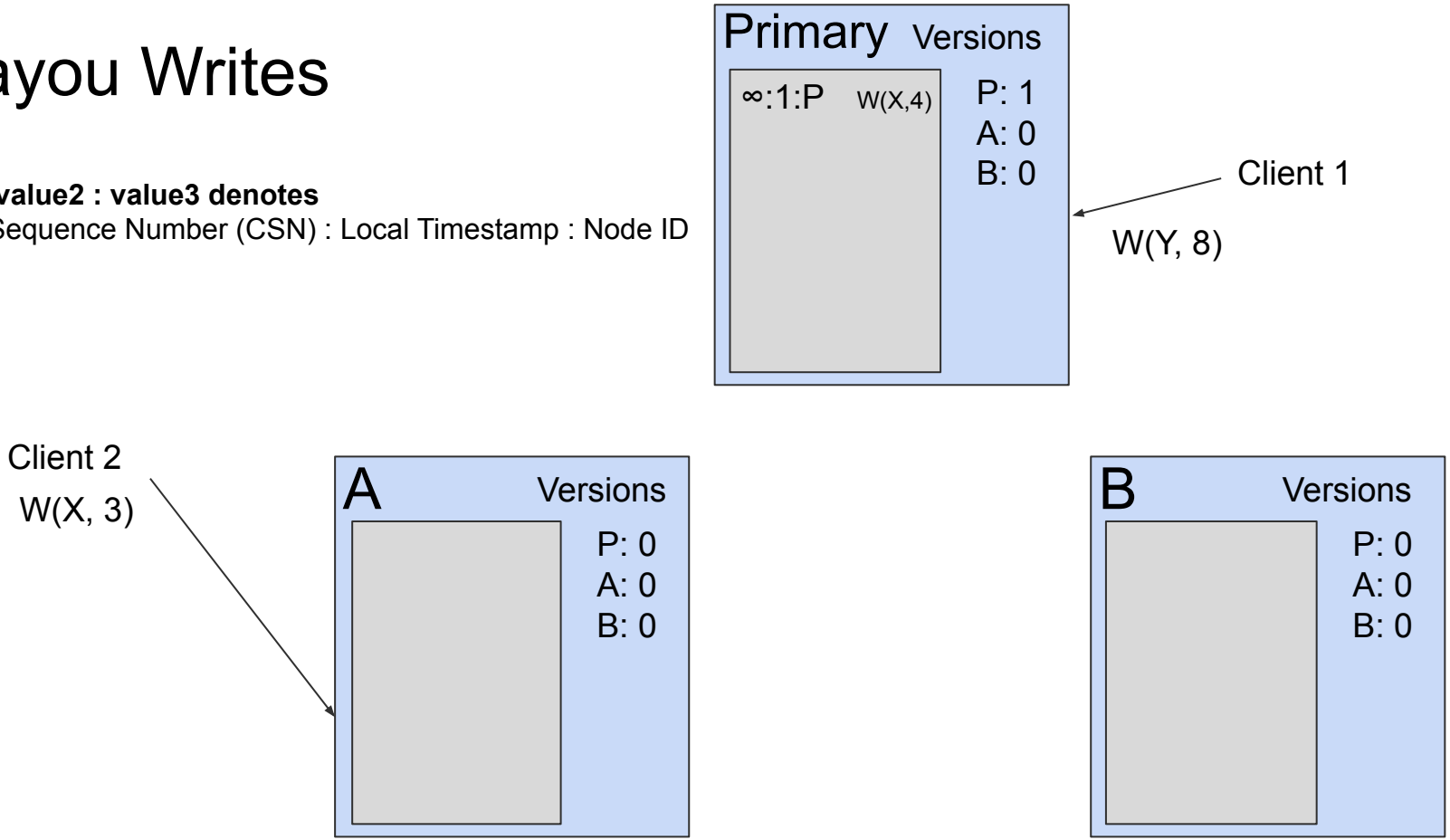
Client 1





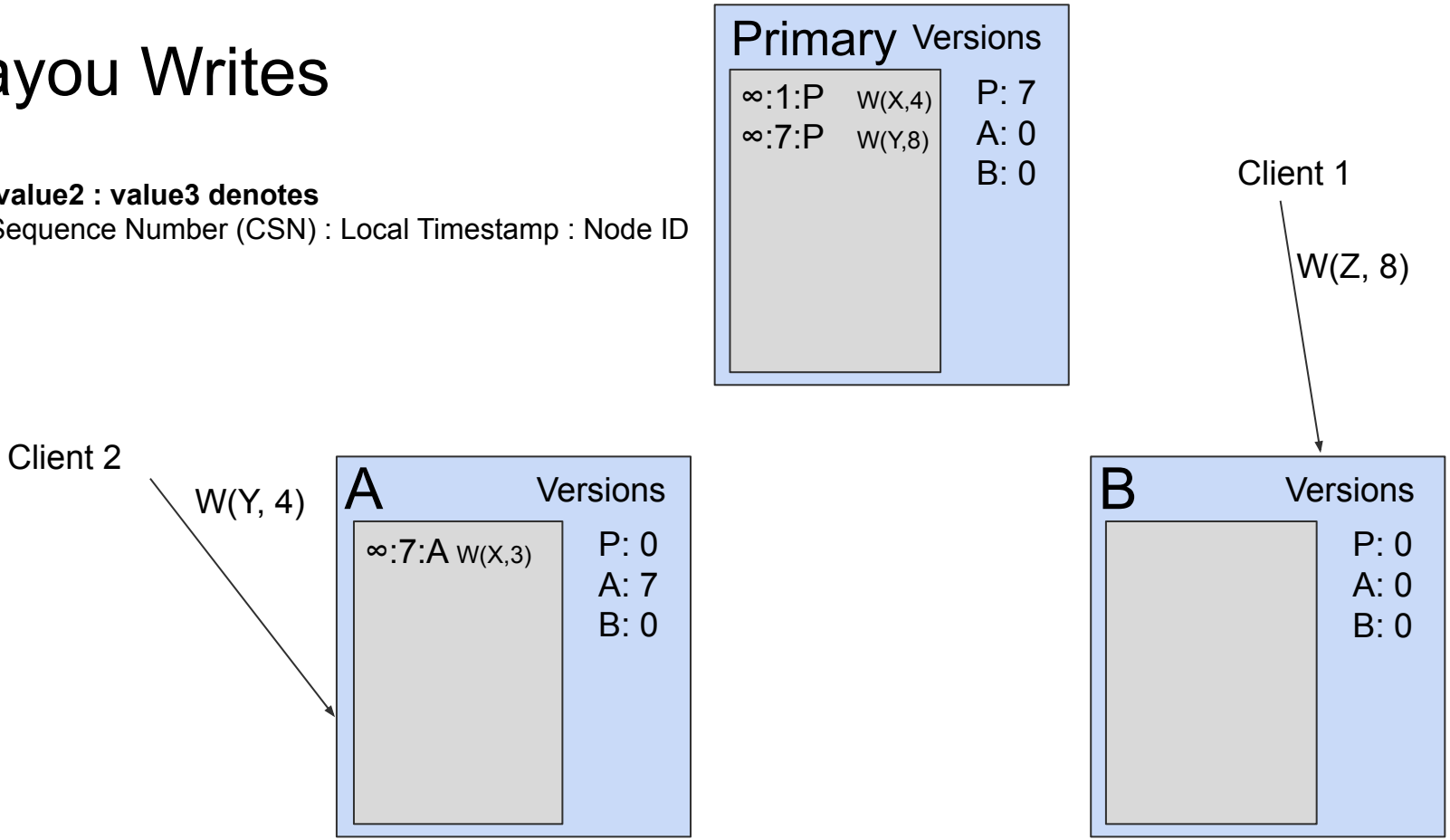
Bayou Writes

value1 : value2 : value3 denotes
Commit Sequence Number (CSN) : Local Timestamp : Node ID



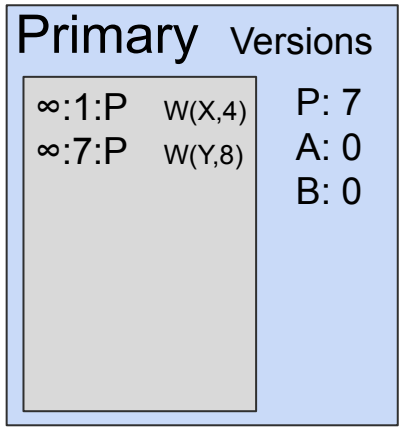
Bayou Writes

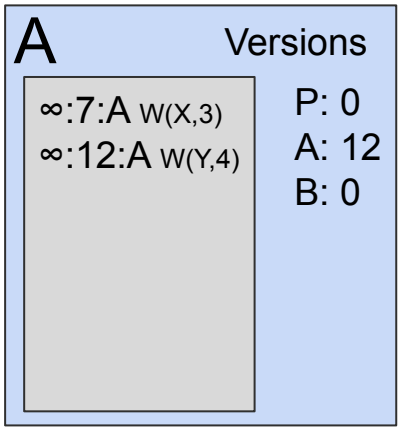
value1 : value2 : value3 denotes
Commit Sequence Number (CSN) : Local Timestamp : Node ID

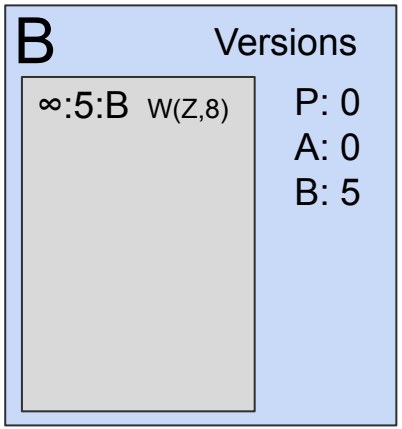


Bayou Writes

value1 : value2 : value3 denotes
Commit Sequence Number (CSN) : Local Timestamp : Node ID

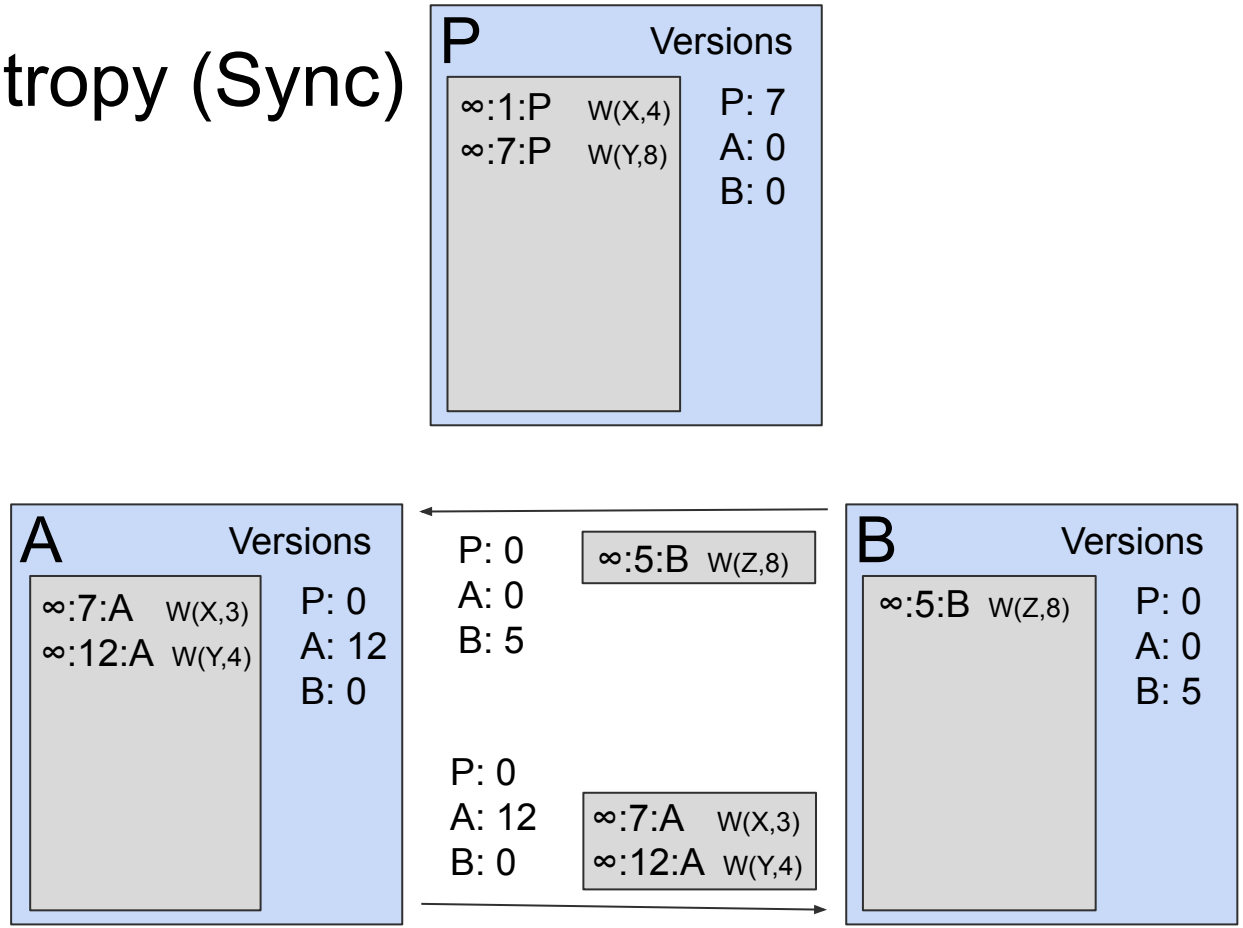




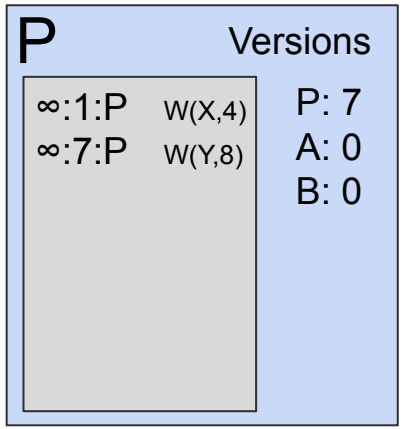


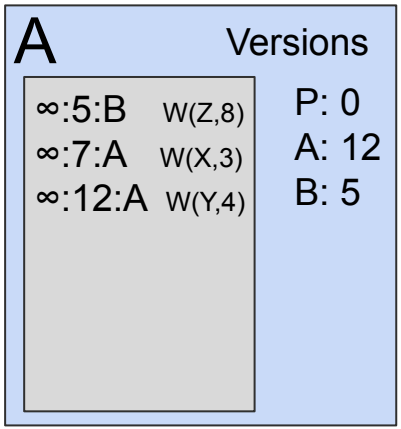
Bayou Anti-Entropy (Sync)

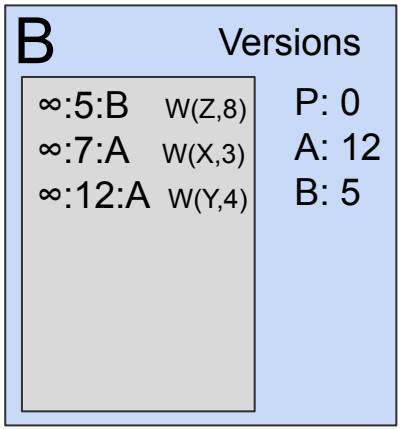
Anti-entropy Session
A & B



Bayou Anti-Entropy (Sync)

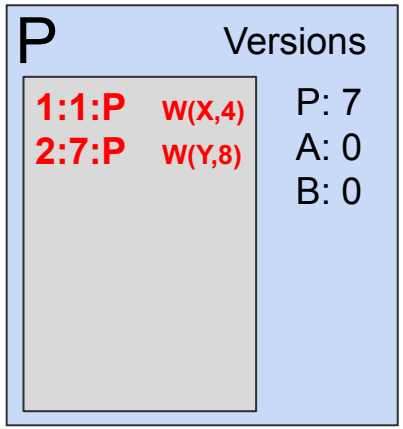


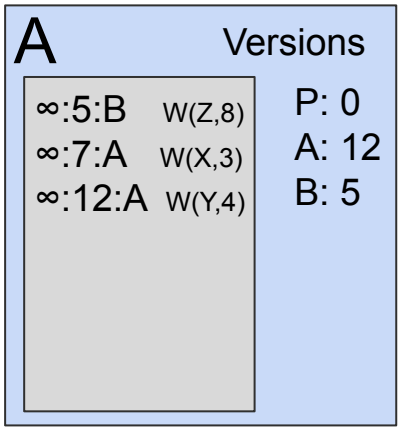


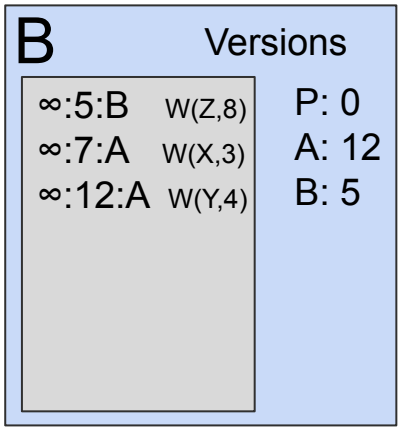


Bayou Commit

Primary commits its entries

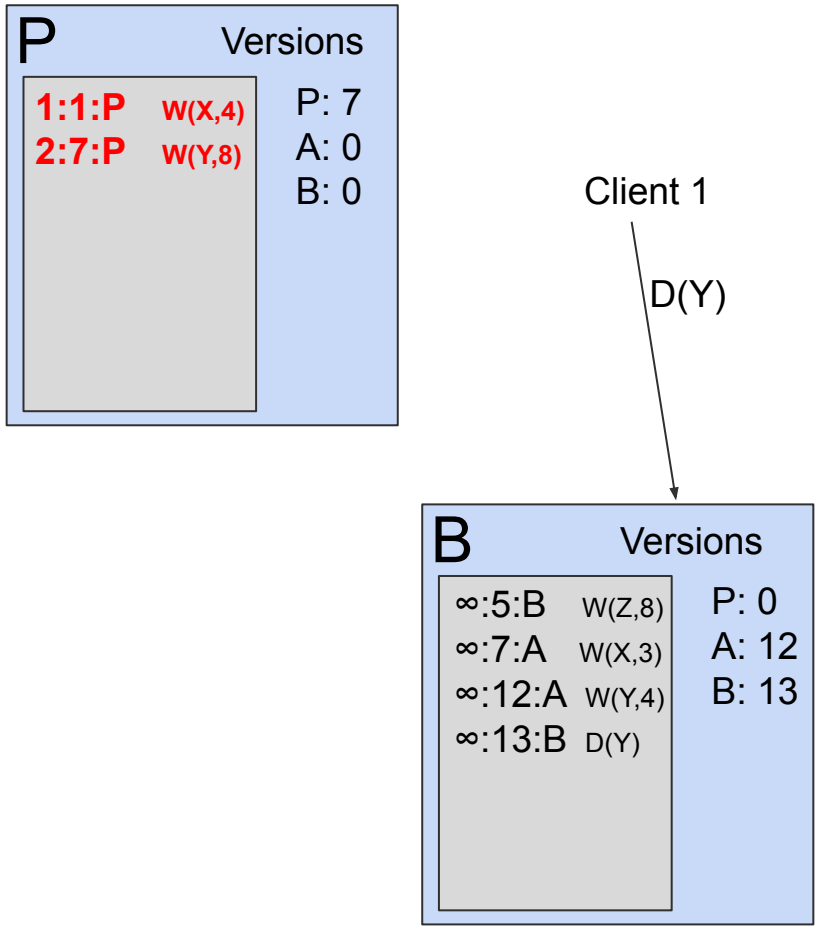






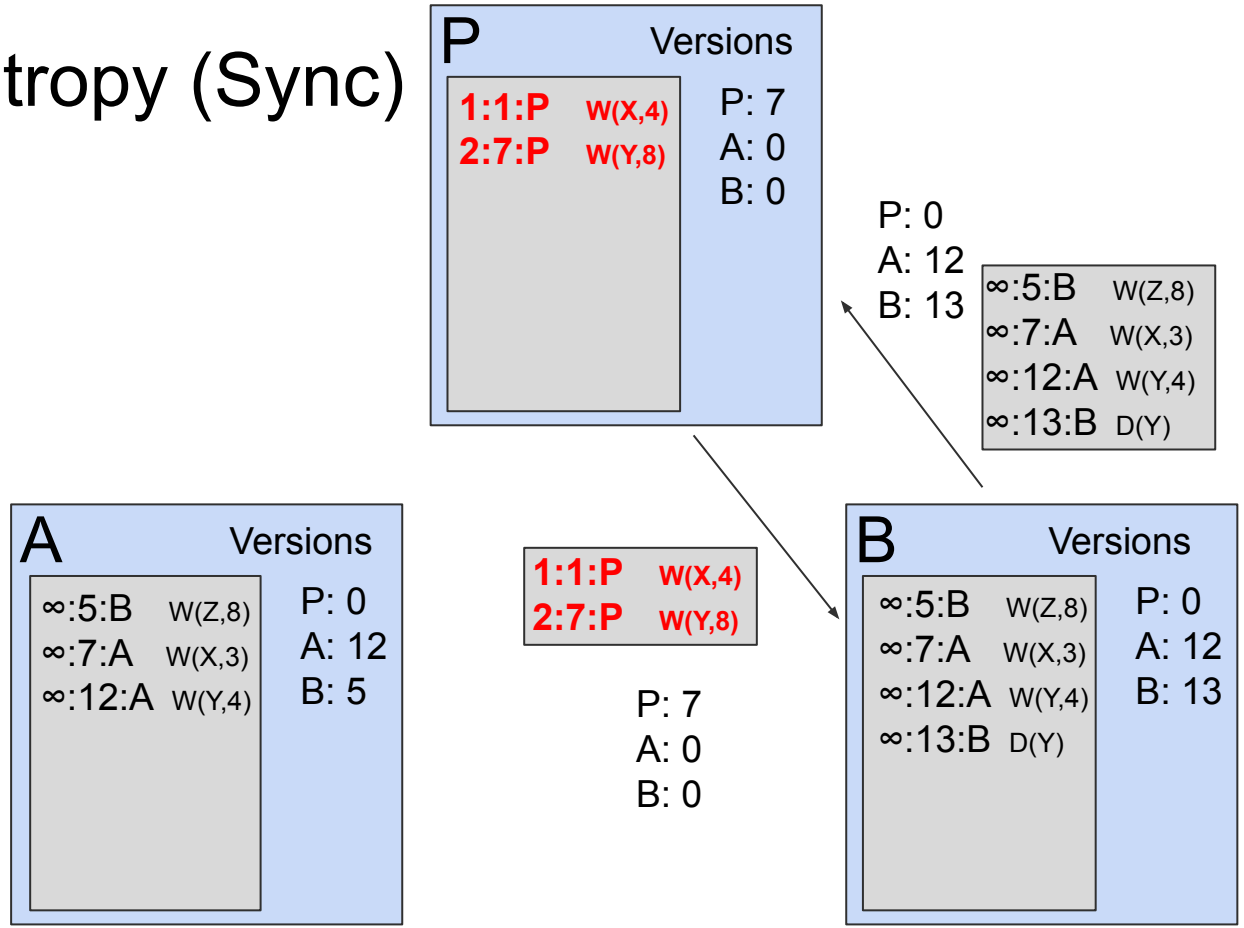
Bayou Write

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



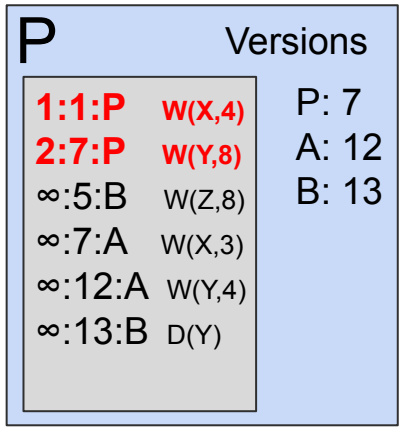
Bayou Anti-Entropy (Sync)

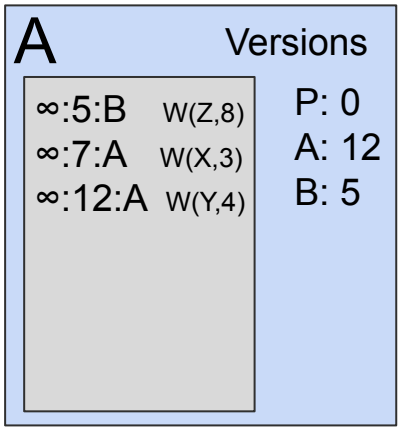
Anti-entropy Session
P & B

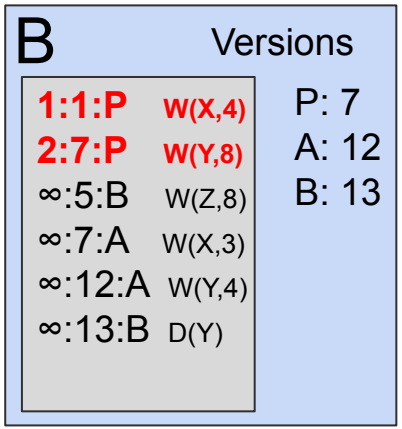


Bayou Anti-Entropy

Anti-entropy Session
P & B
Primary respects causality

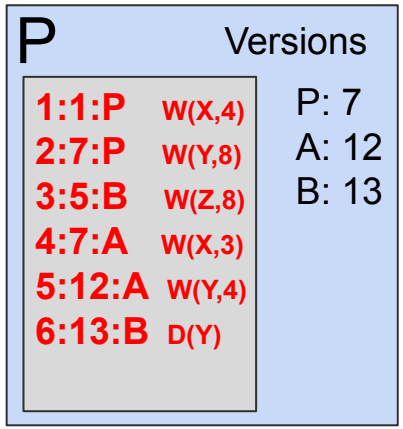


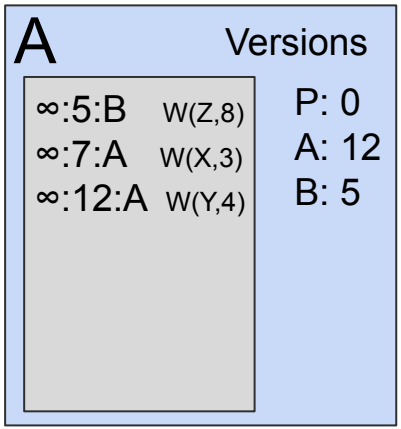


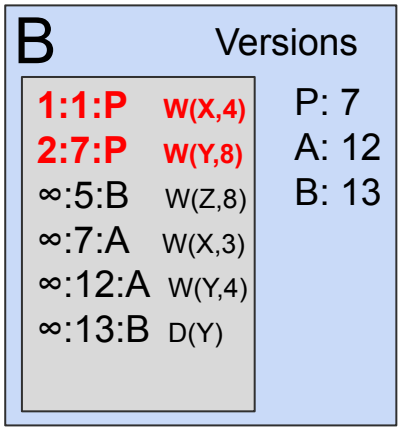


Bayou Commit

Primary commits Its entries

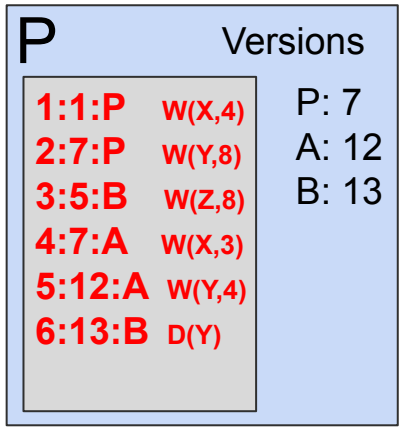


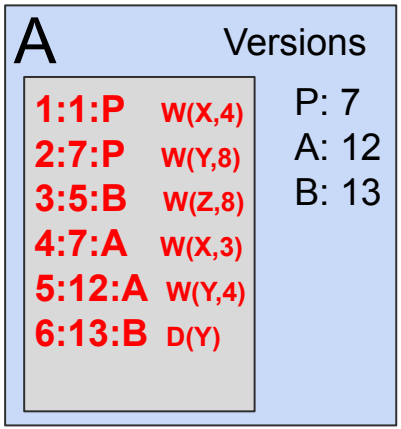


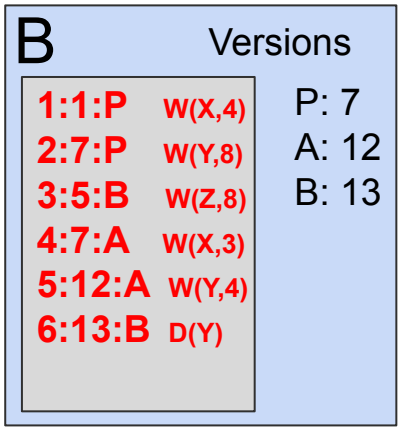


Bayou

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







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

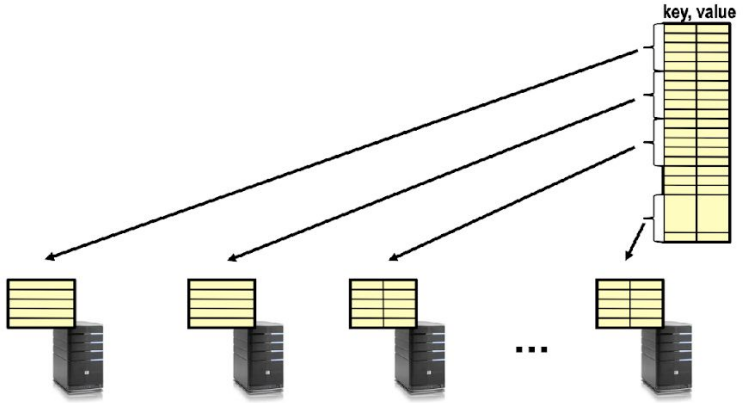
Credit: Ion Stoica's slide deck

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



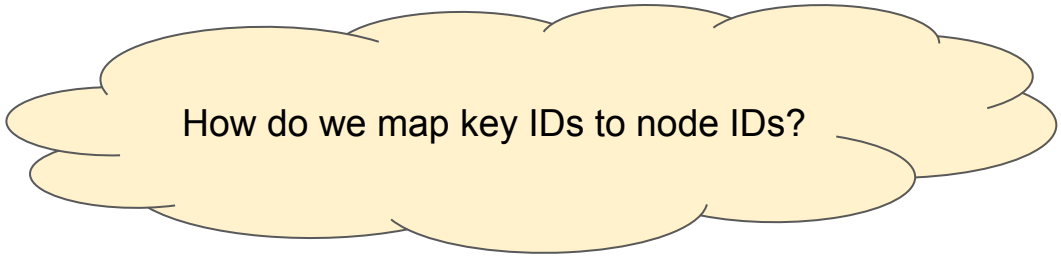
Credit: Ion Stoica's slide deck

Chord

- Chord: “a distributed lookup protocol” for a peer-to-peer distributed hash table [Stoica ‘01]
- *Consistent hashing* for partitioning key space + lookup

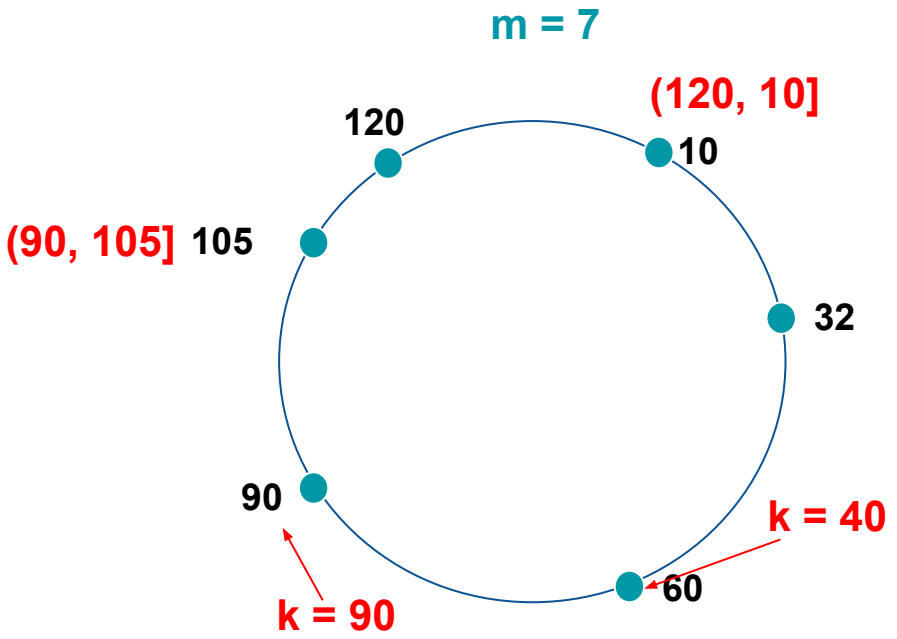
Identifiers in Chord

- 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

- 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.

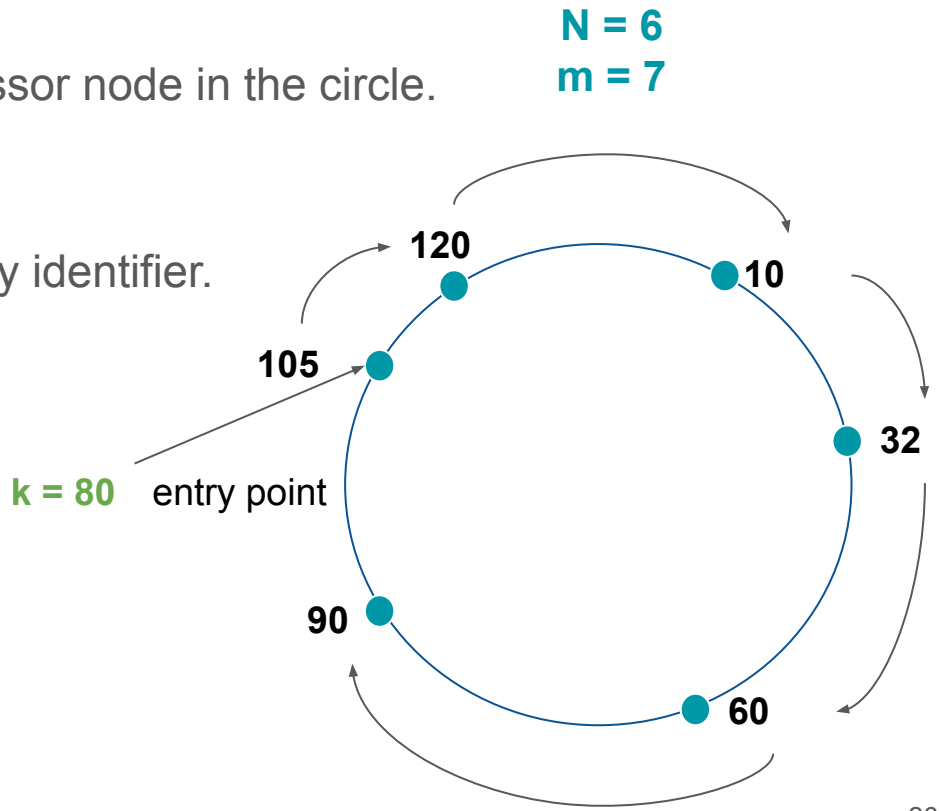


Basic Lookups

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

What is the lookup time?

O(N) hops



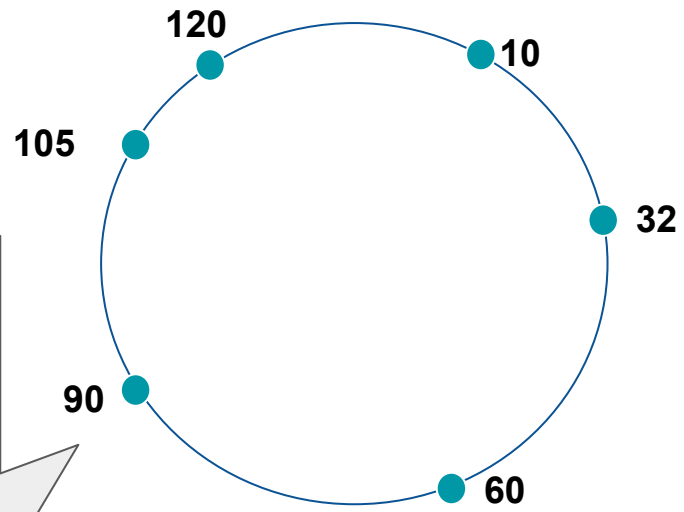
Finger Table Notations

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

n is the identifier of the node

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 $\geq n.finger[k].start$

m = 7



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

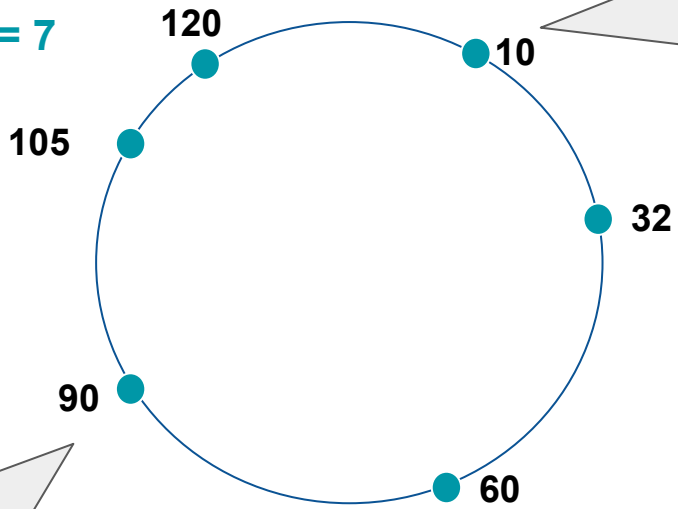
Finger Table of Node 10

n is the identifier of the node

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 $\geq n.finger[k].start$

- A finger table contains m entries

N = 6
m = 7



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

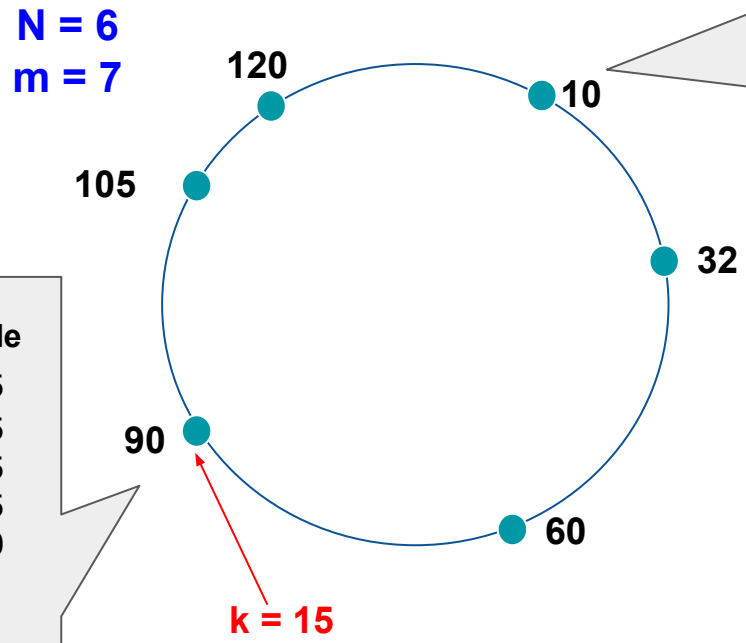
start	interval	node
91	[91, 92)	105
92	[92, 94)	105
94	[94, 98)	105
98	[98, 106)	105
106	[106, 122)	120
122	[122, 26)	10
26	[26, 90)	32

Route k = 15

- A finger table contains m entries

n is the identifier of the node

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

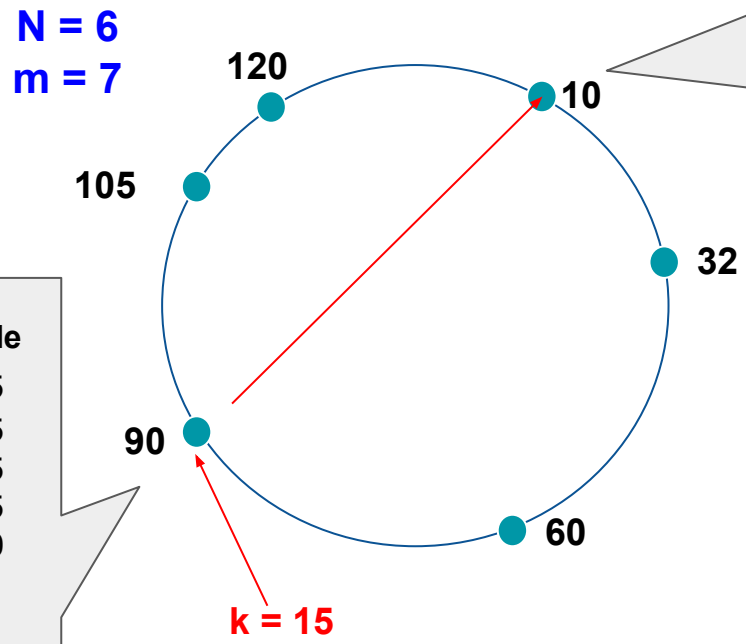
start	interval	node
91	[91, 92)	105
92	[92, 94)	105
94	[94, 98)	105
98	[98, 106)	105
106	[106, 122)	120
122	[122, 26)	10
26	[26, 90)	32

Route k = 15

- A finger table contains m entries

n is the identifier of the node

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		32
12		32
14		32
18		32
26		32
42		60
74		90

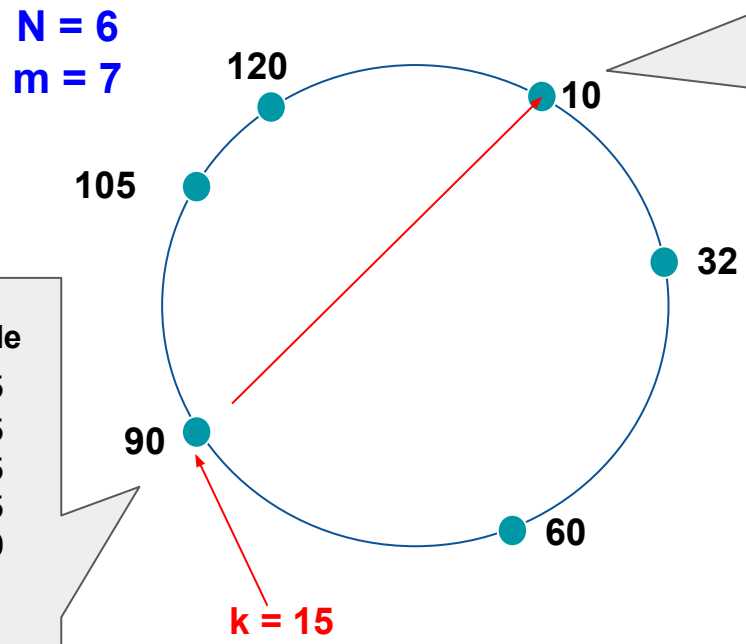
start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

Route k = 15

- A finger table contains m entries

n is the identifier of the node

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		32
12		32
14		32
18		32
26		32
42		60
74		90

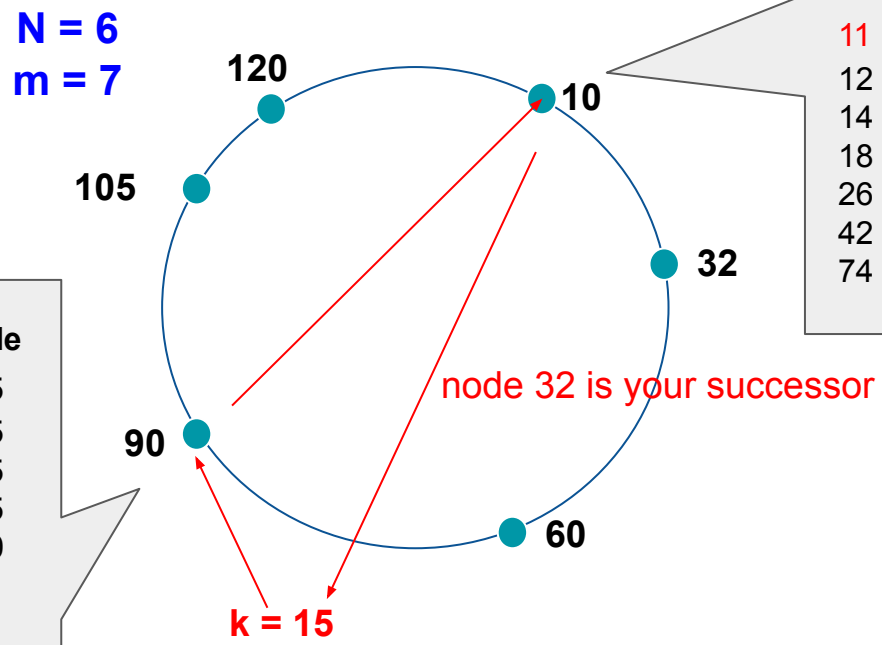
start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

Route k = 15

- A finger table contains m entries

n is the identifier of the node

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		32
12		32
14		32
18		32
26		32
42		60
74		90

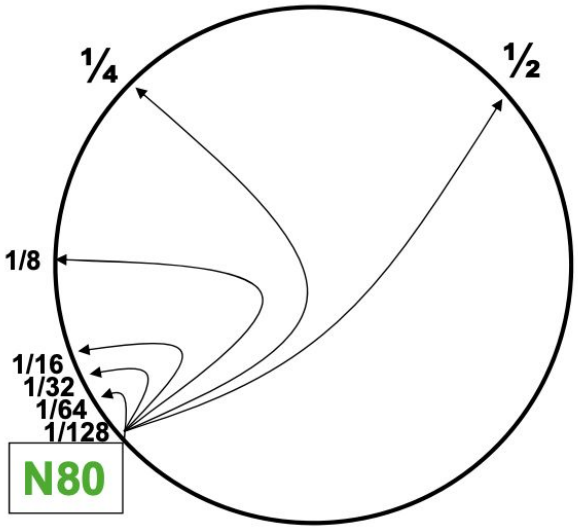
start	interval	node
91		105
92		105
94		105
98		105
106		120
122		10
26		32

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.