

Midterm Review



COS 418/518: Distributed Systems

Jialin Ding, Mike Freedman

Agenda

- Recap of material in the class so far
- Clarifying some details
 - Bayou
 - Chord
 - Paxos
- Q&A

What we've learned (and will be on the midterm)

- Network communication and RPCs
 - Think about: steps that RPC takes from client to server back to client
 - Don't need: details of the networking stack
- Failure semantics
 - At-most-once vs. at-least-once vs. exactly-once
 - Think about: how systems must be designed to achieve these semantics, what kinds of applications are appropriate for each semantic
- Time
 - Lamport clocks vs. vector clocks, totally-ordered multicast
 - Think about: how exactly they work, differences between the two logical clocks

What we've learned (and will be on the midterm)

- Distributed snapshots
 - Think about: how exactly Chandy-Lamport works, and why it works
 - Don't need to read the paper (this applies for other lectures as well)
- Eventual consistency with Bayou
 - Think about: the meaning of eventual consistency, how Bayou achieves eventual consistency
- Chord
 - Think about: tradeoffs between different peer-to-peer architectures, how exactly Chord works
 - Don't need: precise details about how Chord works when nodes join and leave

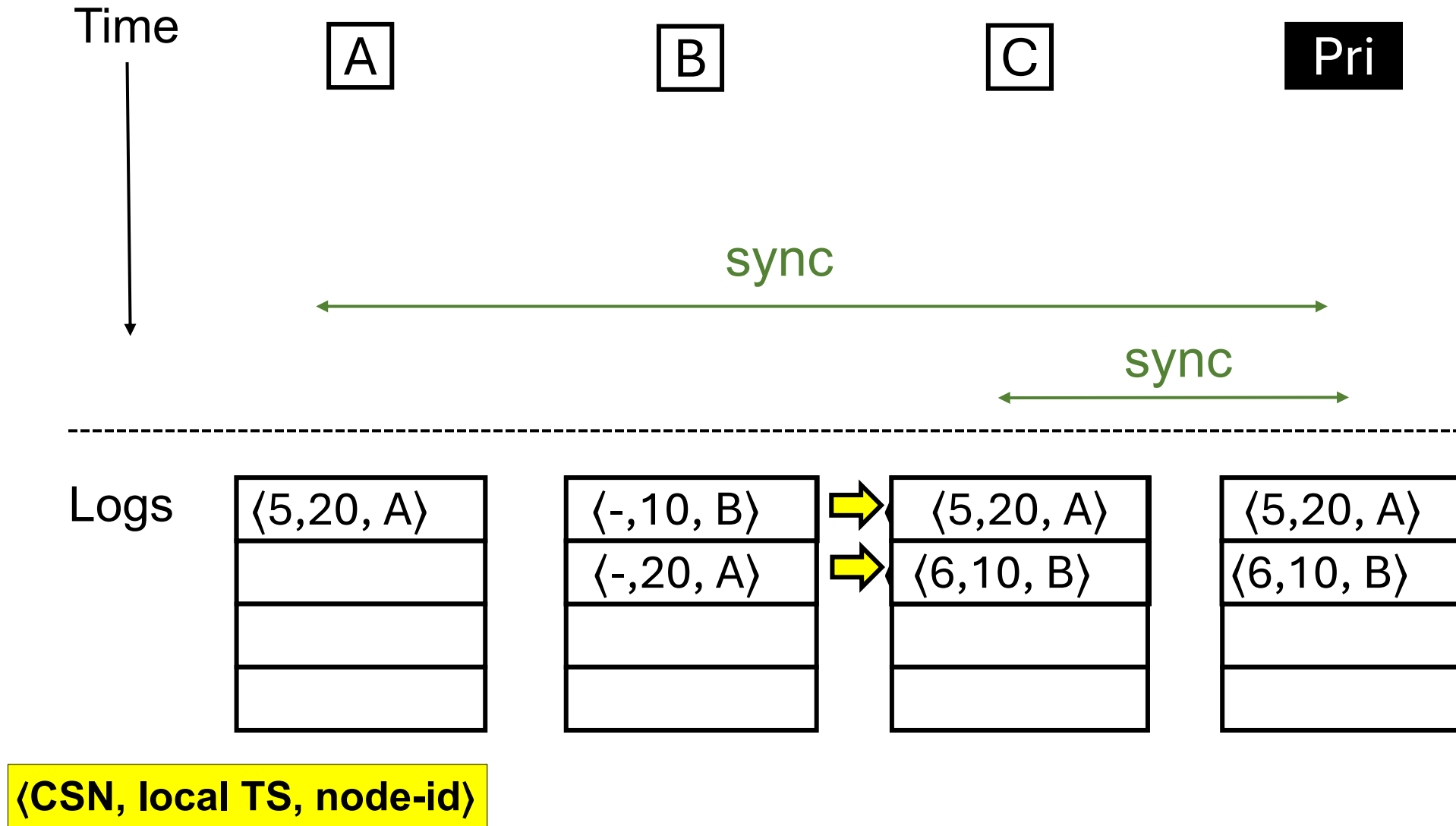
What we've learned (and will be on the midterm)

- Primary-backup
 - Think about: how primary-backup is designed to achieve strong consistency
 - **Terminology clarification:** for this class, assume “primary-backup replication” is trying to achieve strong consistency, so it must use synchronous replication.
 - Don't need: specifics of VM-FT (except preventing split brain)
- Consensus with Paxos
 - Think about: how exactly the Paxos algorithm works, how it guarantees safety and almost-always-liveness
 - Don't need: Multi-Paxos

Bayou

- We called it eventual consistency, but also want to guarantee causal relationships are in the right order
 - Meaning, if A happens before B, then A should be before B in the committed log
 - We'll see later this is actually called causal consistency
- Q: Every update has a Lamport timestamp, associated with what?
- Q: Why does Bayou use Lamport clocks instead of vector clocks?
- Q: Can the Bayou primary have tentative updates in its log?
- Q: Is it possible for Bayou primary to hear about updates in non-causal order?

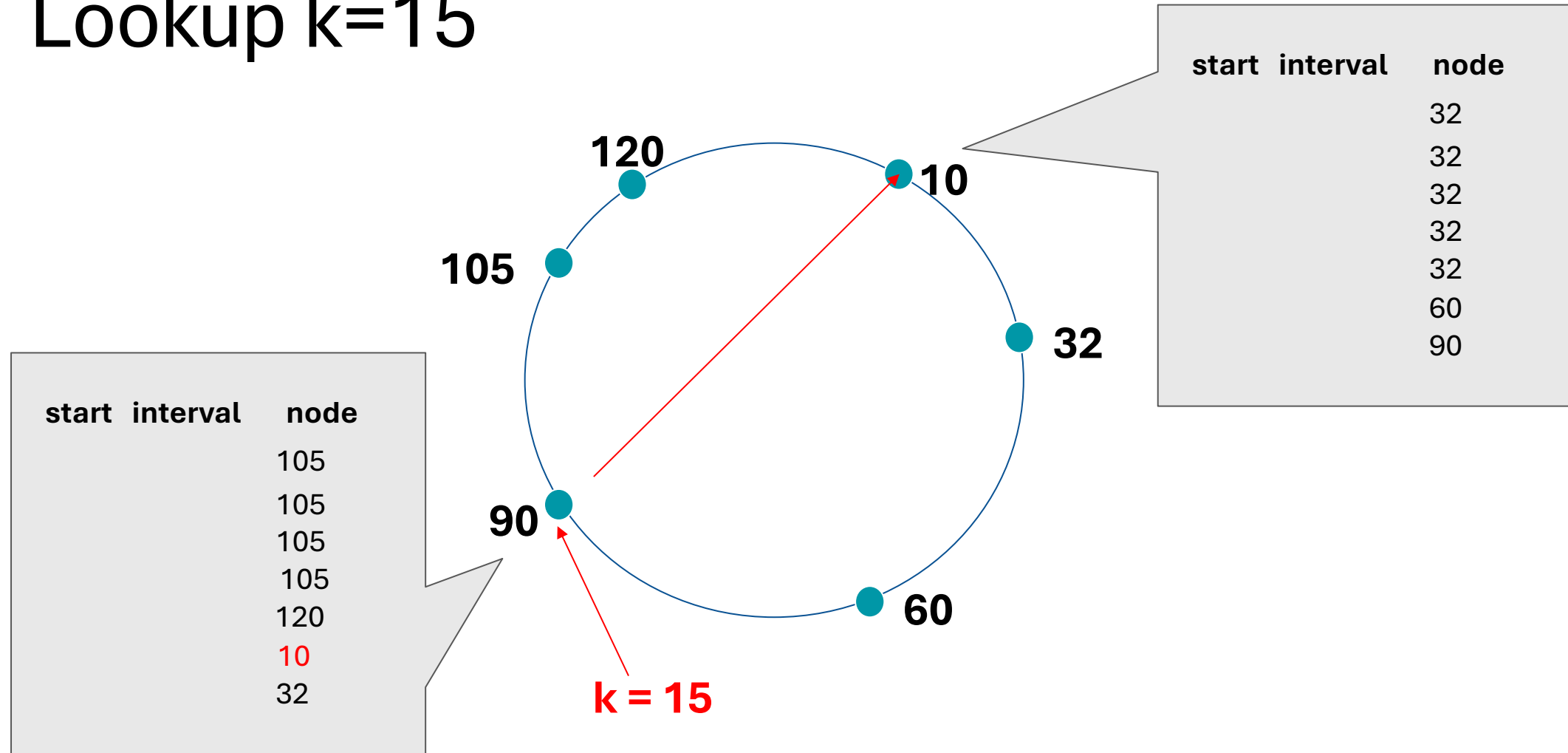
Tentative order $\neq$ commit order



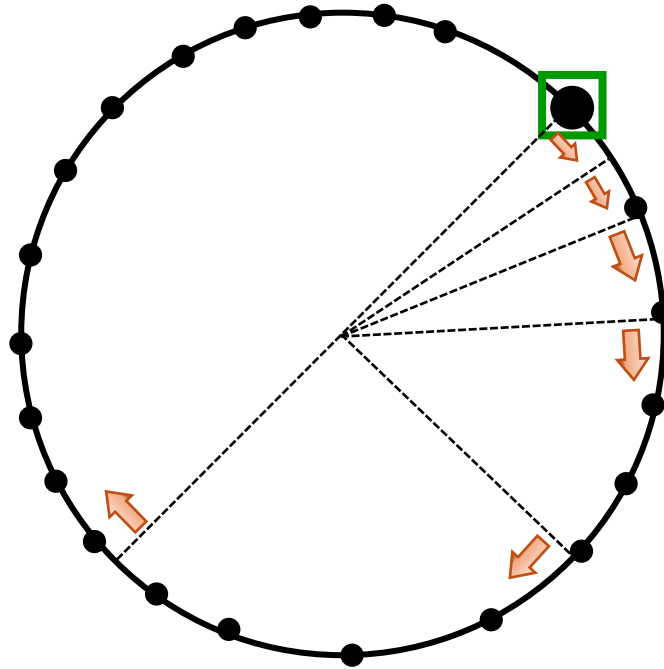
Chord

- What is stored for each Chord node
 - Successor pointer: required
 - Finger table: optional, for performance
 - Predecessor pointer: optional, for faster stabilization when nodes join and leave
- How lookups work: “never go *past* your target”
 - With finger tables “only”
 - With finger tables and successor pointers
 - Successor pointer is actually equal to the first successor node in the finger table!
- Q: Why is lookup $O(\log N)$ and not $O(m)$?

Lookup k=15



Chord – finger tables for *find_predecessor*



Chord under ring membership changes

- What happens when a node joins the Chord
 - Stabilization will fix the successor and predecessor pointers of the joined node and its two neighbors
 - Finger table can be fixed lazily
- What happens when a node leaves the Chord
 - Nothing happens immediately
 - Stabilization will fix things over time
- Does lookup still work?
 - Temporarily might not work! Chord is eventually consistent in this sense.
 - Intuition: “update pointers” and “transfer data” are not done atomically

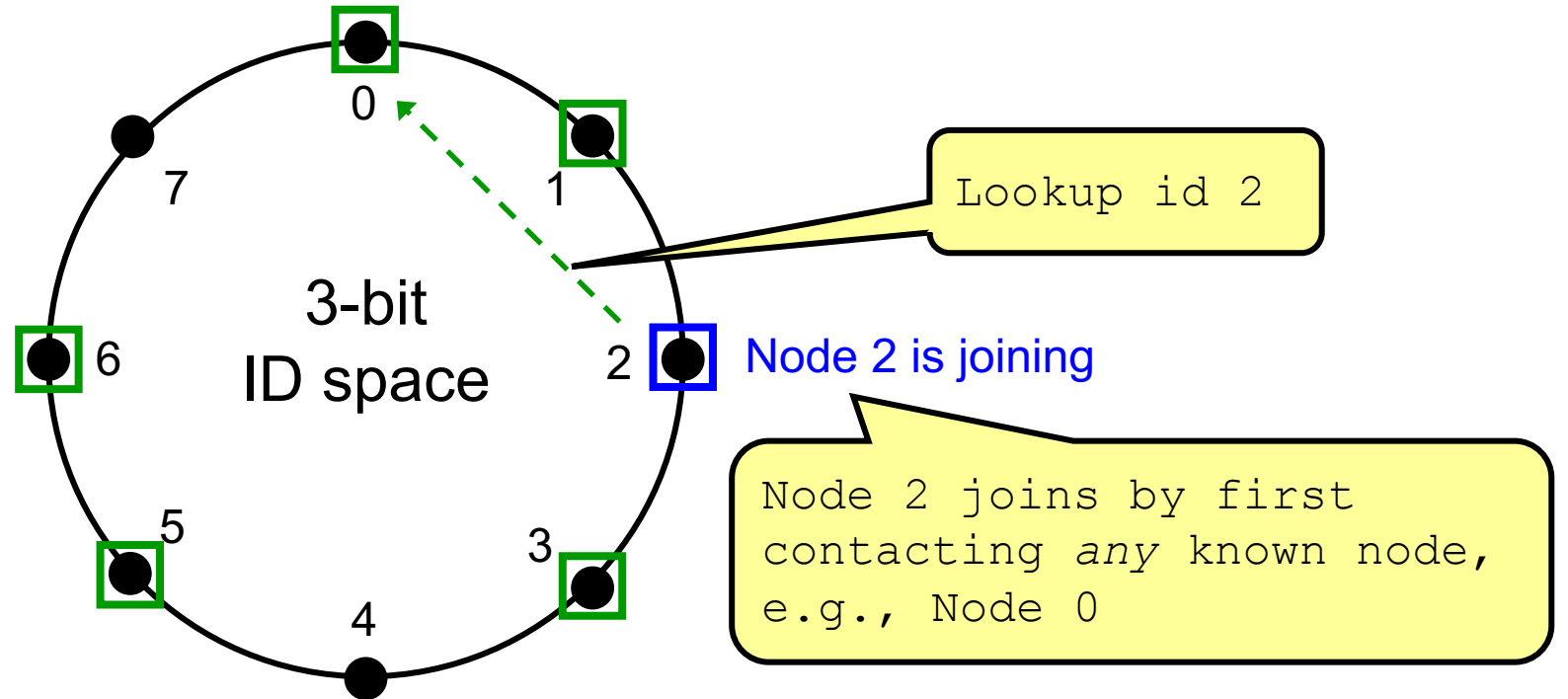
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



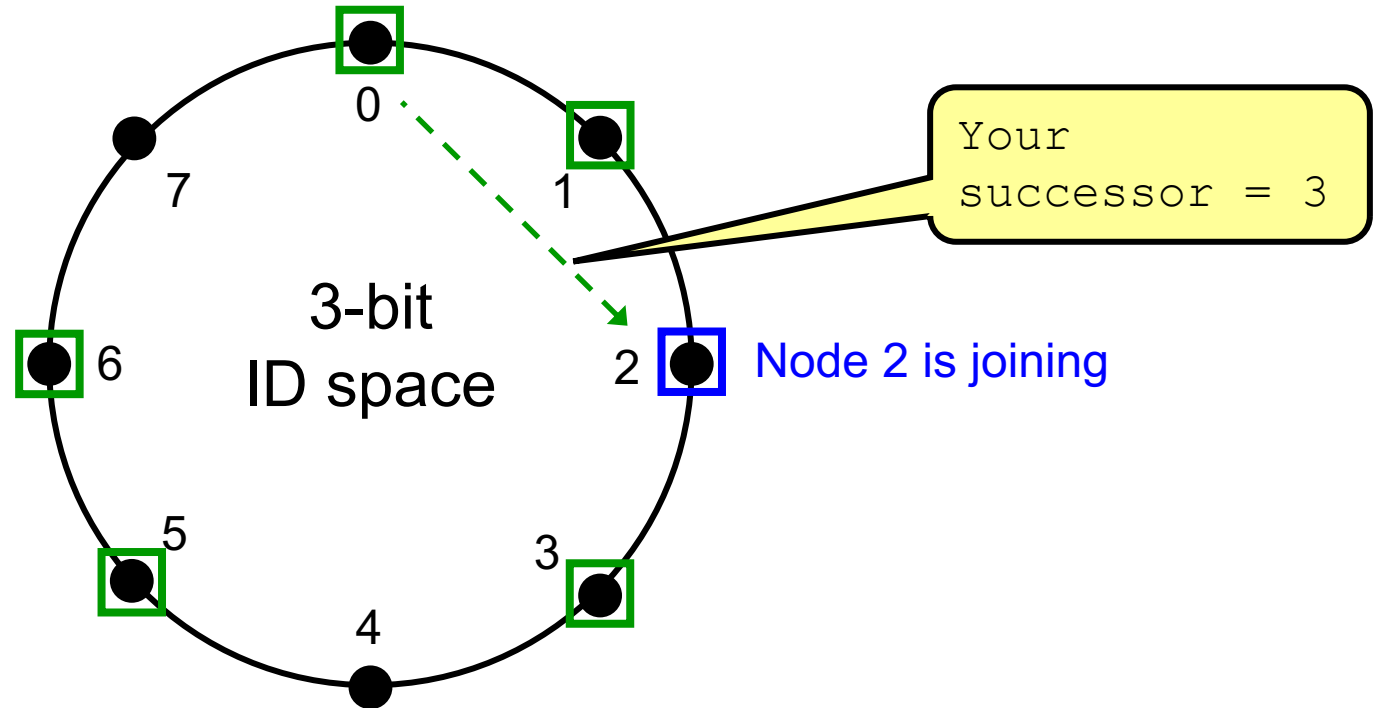
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



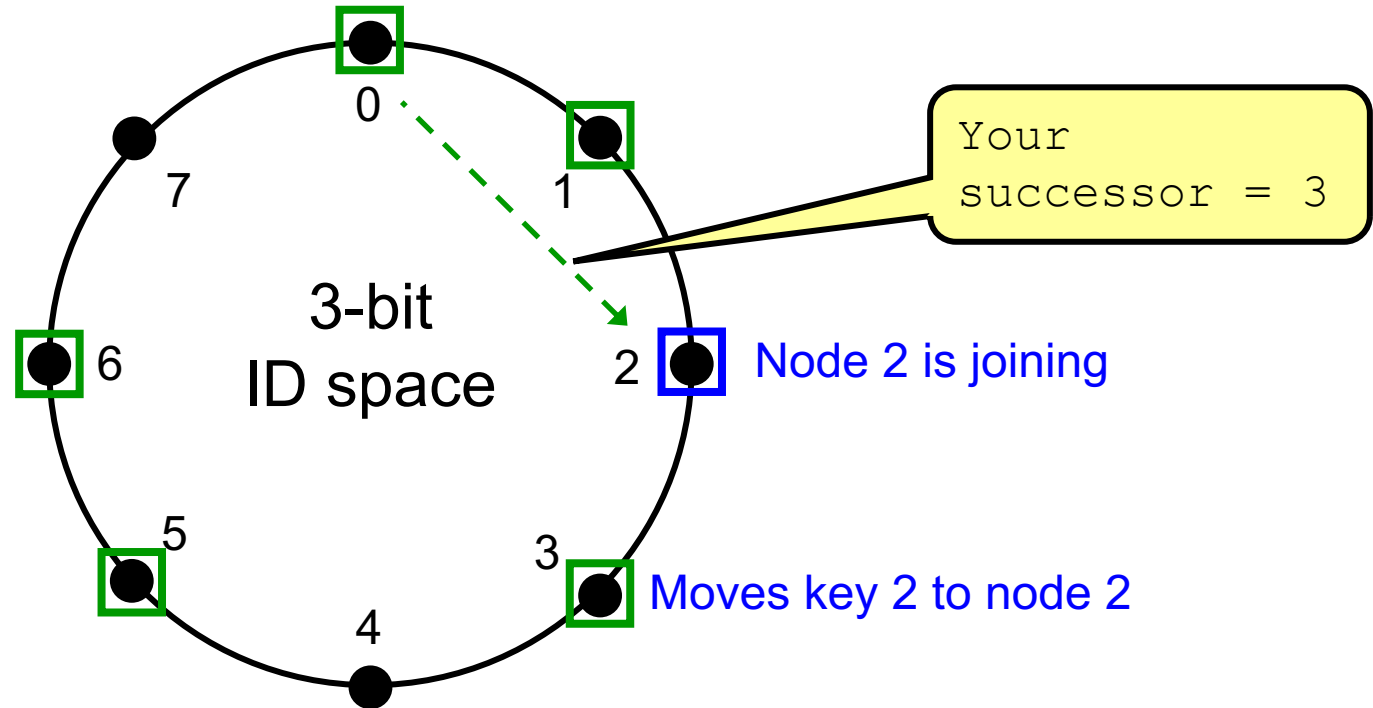
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



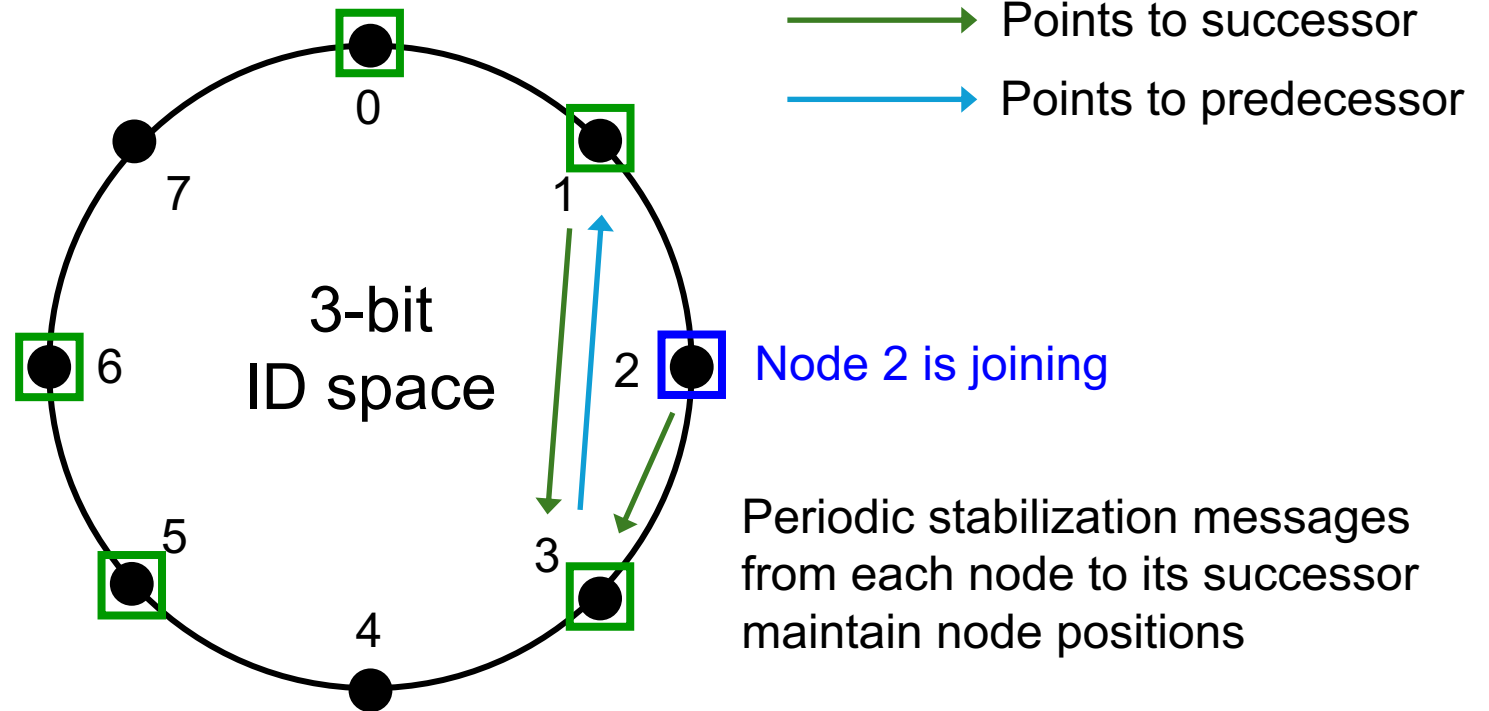
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



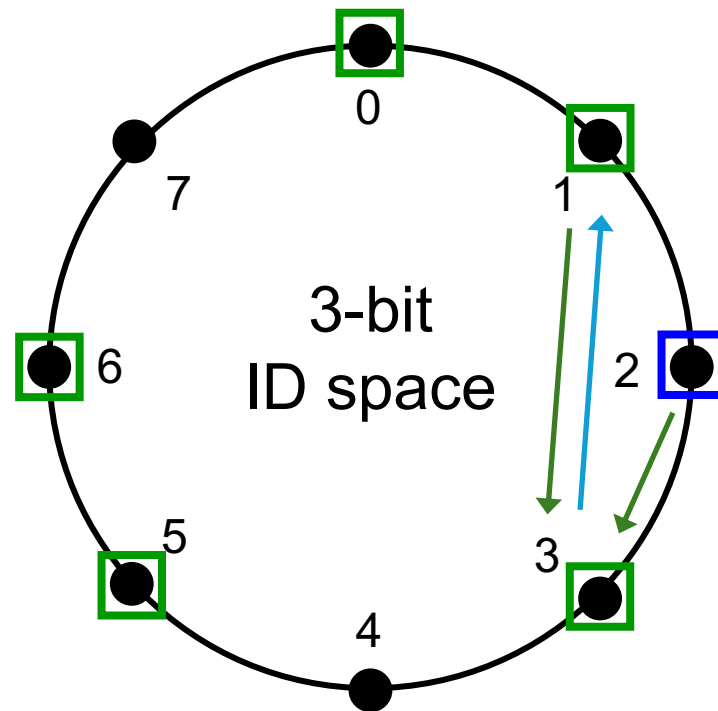
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

STABILIZE() [N.successor = M]

N->M: "What is your predecessor?"

M->N: "x is my predecessor"

if x between (N,M), N.successor = x

N->N.successor: NOTIFY()

NOTIFY()

N->N.successor: "I think you are my successor"

M: upon receiving NOTIFY from N:

If (N between (M.predecessor, M))

M.predecessor = N

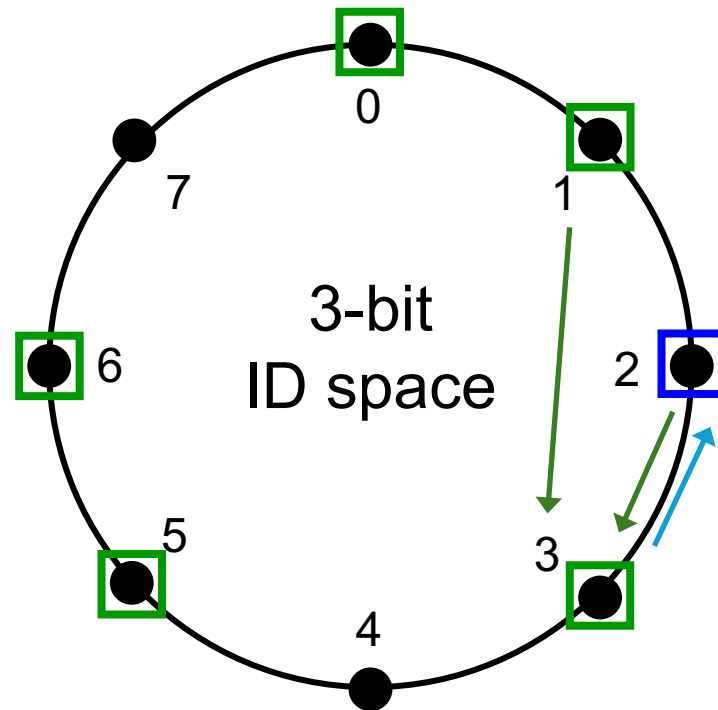
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

STABILIZE() [N.successor = M]

N->M: "What is your predecessor?"

M->N: "x is my predecessor"

if x between (N,M), N.successor = x

N->N.successor: NOTIFY()

NOTIFY()

N->N.successor: "I think you are my successor"

M: upon receiving NOTIFY from N:

If (N between (M.predecessor, M))

M.predecessor = N

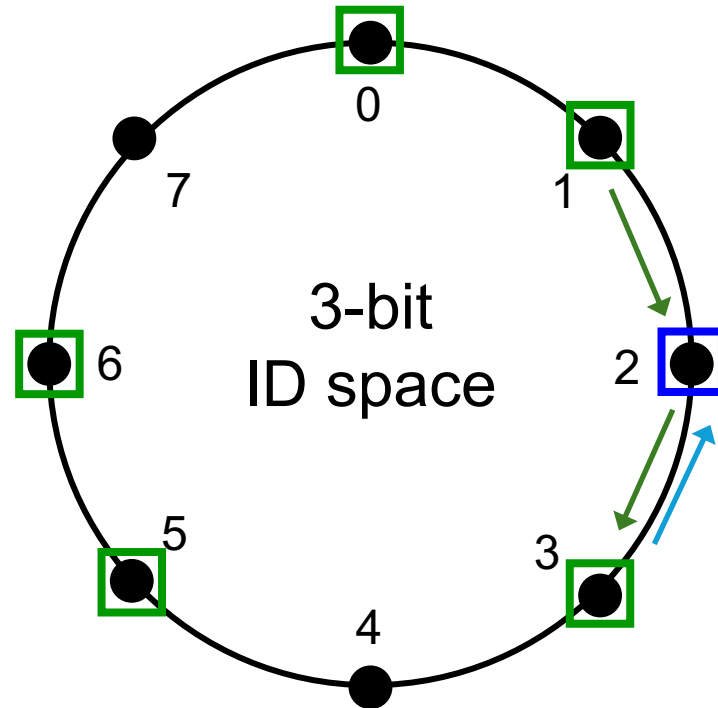
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

STABILIZE() [N.successor = M]

N->M: "What is your predecessor?"

M->N: "x is my predecessor"

if x between (N,M), N.successor = x

N->N.successor: NOTIFY()

NOTIFY()

N->N.successor: "I think you are my successor"

M: upon receiving NOTIFY from N:

If (N between (M.predecessor, M))

M.predecessor = N

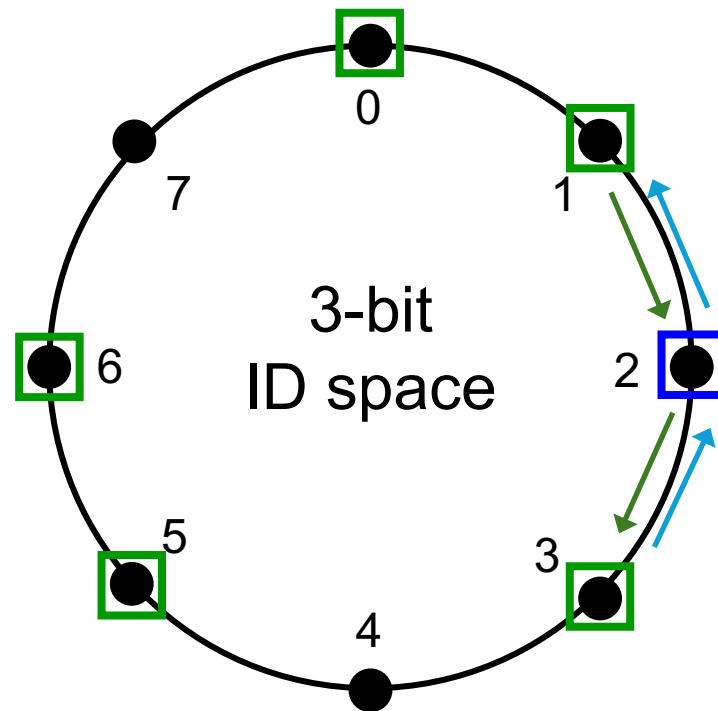
Chord – node joining

Identifiers have $m = 3$ bits

Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

STABILIZE() [N.successor = M]

N->M: "What is your predecessor?"

M->N: "x is my predecessor"

if x between (N,M), N.successor = x

N->N.successor: NOTIFY()

NOTIFY()

N->N.successor: "I think you are my successor"

M: upon receiving NOTIFY from N:

If (N between (M.predecessor, M))

M.predecessor = N

Paxos

- Q: If an acceptor sends a Promise message for proposal number n , will that acceptor definitely accept an Accept message for proposal number n ?
- Q: If an acceptor sends a Promise message for proposal number n , in what situation will that acceptor definitely accept an Accept message for proposal number n ?
- Safety and liveness
 - Always safe: once majority of acceptors accept the same (n, v) pair (i.e., value is “chosen”), a majority will never accept a new value
 - Not live in adversarial cases (e.g., two dueling proposers)