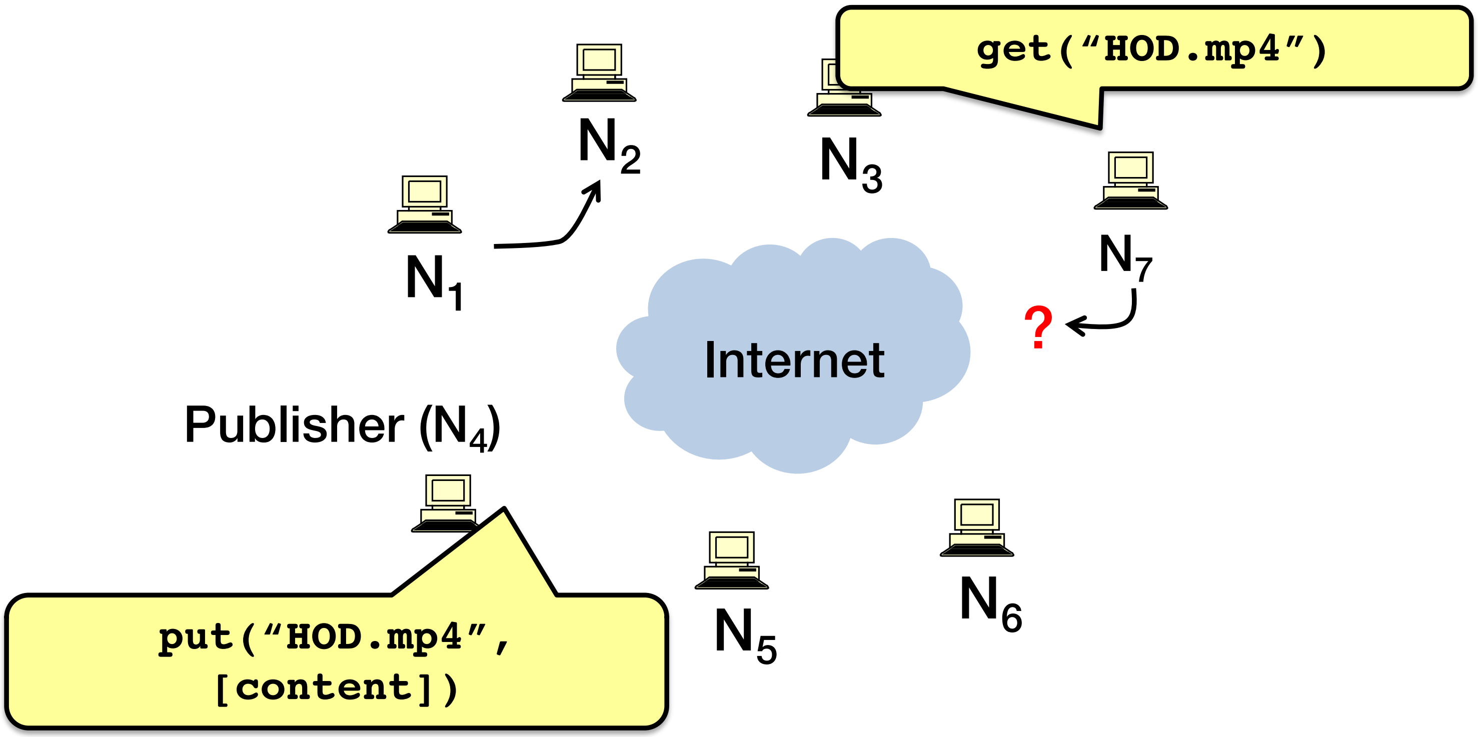
Distributed Hash Tables & Chord

COS 418/518: Distributed Systems

Lecture 8

Wyatt Lloyd

The lookup problem: locate the data



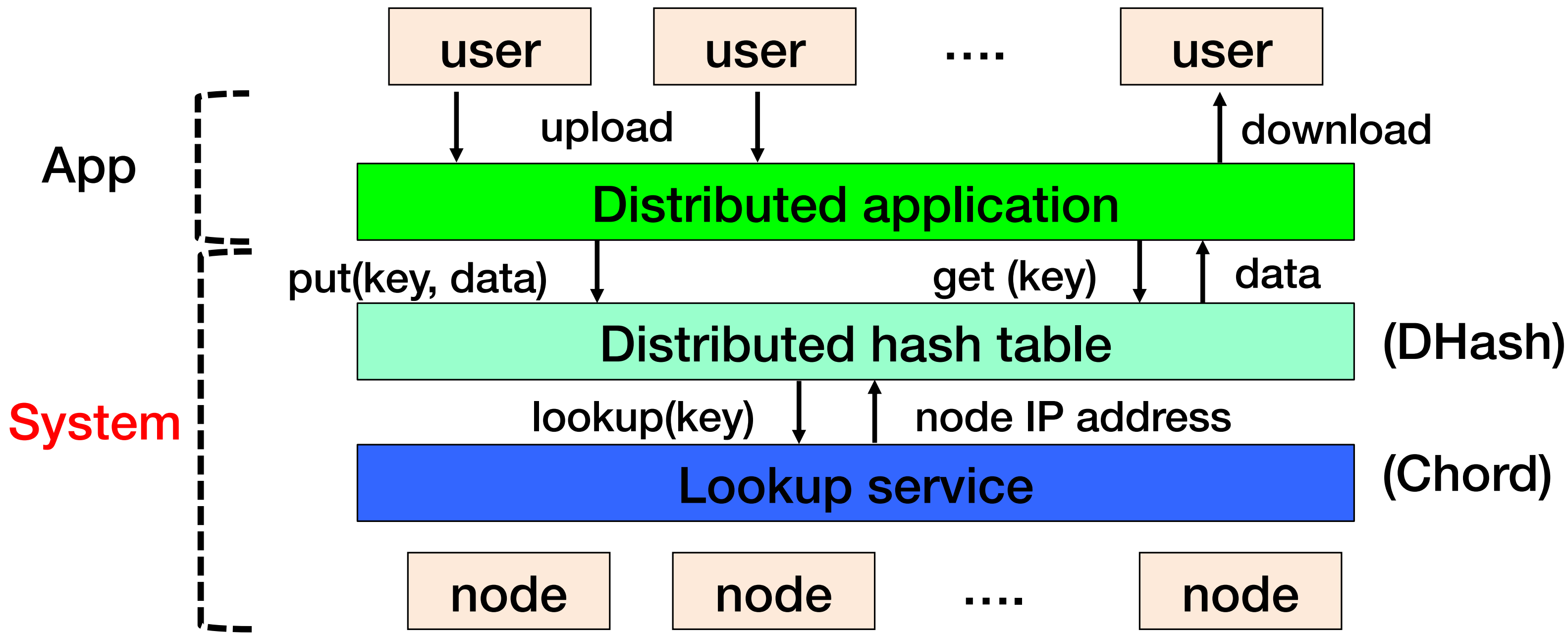
What is a DHT (and why)?

- Distributed Hash Table: an abstraction of hash table in a distributed setting

```
key = hash(data)
lookup(key) → IP addr (Chord lookup service)
send-RPC(IP address, put, key, data)
send-RPC(IP address, get, key) → data
```

- **Partitioning data** in large-scale distributed systems
  – Tuples in a global database engine
  – Data blocks in a global file system
  – Files in a P2P file-sharing system

Cooperative storage with a DHT



DHT is expected to be

- Decentralized: no central authority
- Scalable: low network traffic overhead
- Efficient: find items quickly (latency)
- Dynamic: nodes fail, new nodes join

Chord identifiers

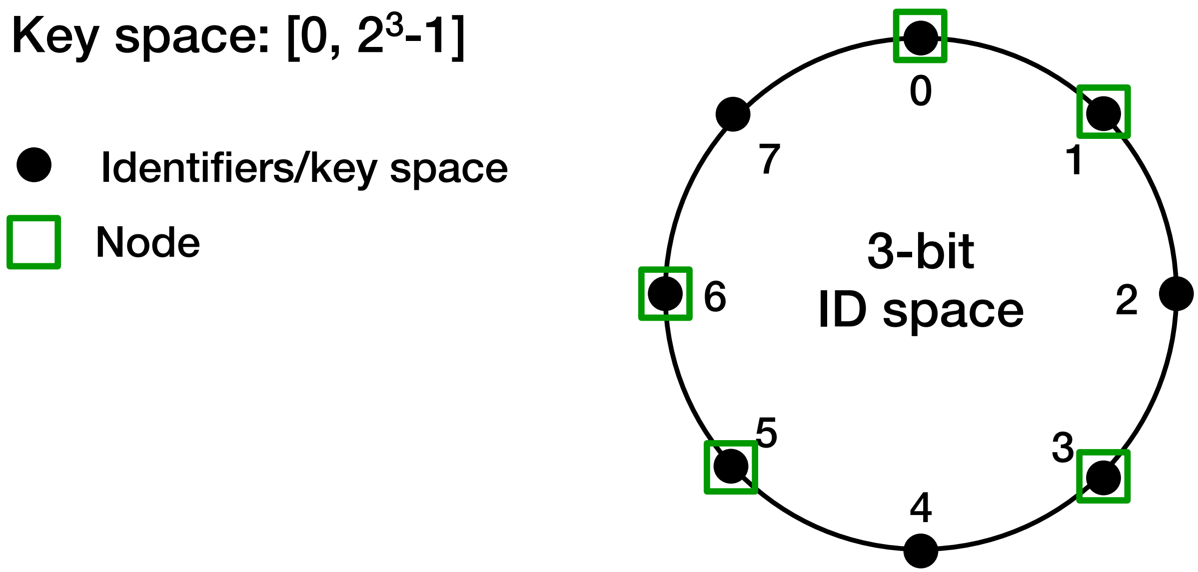
- **Hashed values (integers) using the same hash function**
  - **Key identifier** = SHA-1(key)
  - **Node identifier** = SHA-1(IP address)

- **How does Chord partition data?**
  - i.e., map key IDs to node IDs

- **Why hash key and address?**
  - Uniformly distributed in the ID space
  - Hashed key → load balancing; hashed address → independent failure

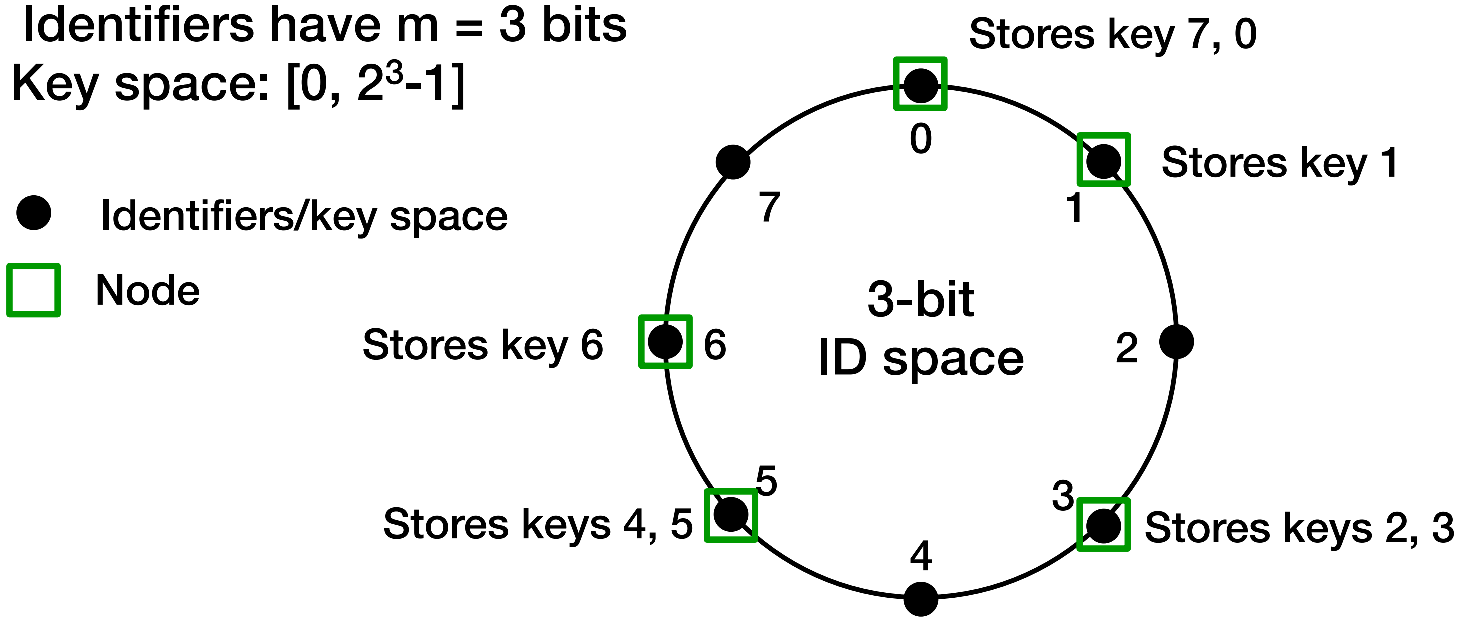
Consistent hashing [Karger '97] – data partition

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Consistent hashing [Karger '97] – data partition

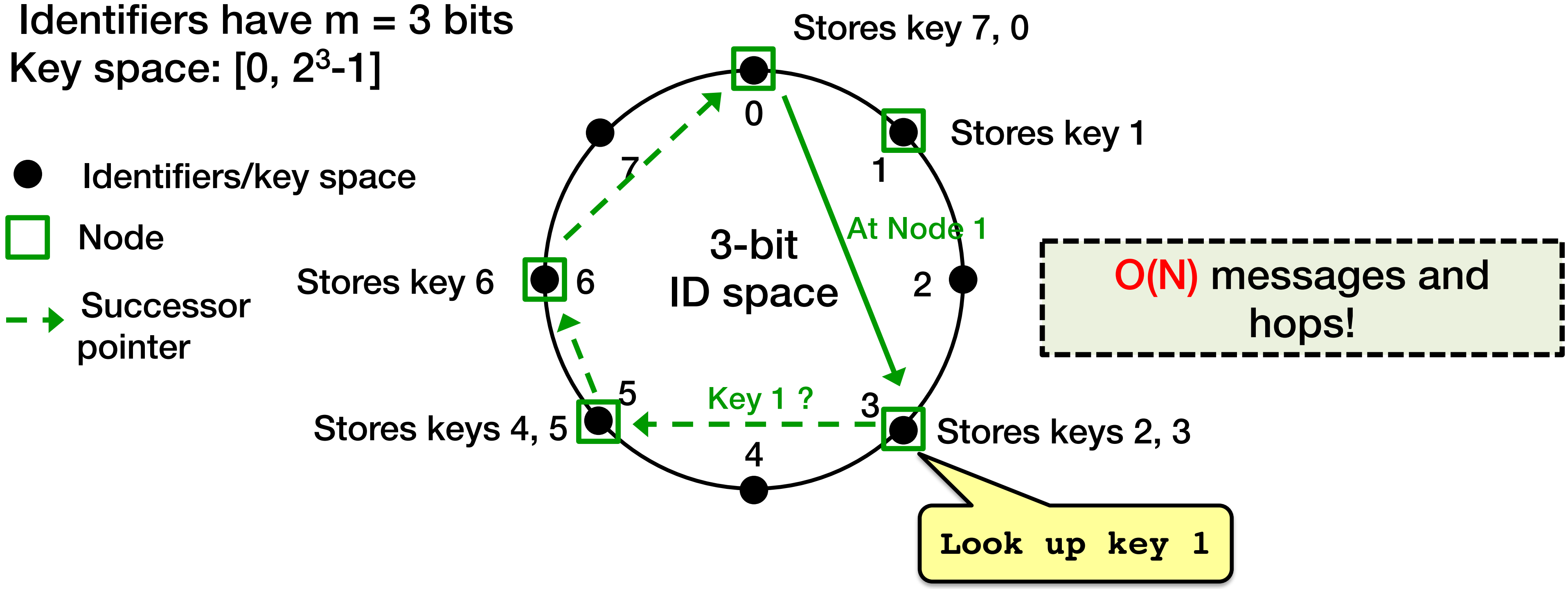
Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$



Key is stored at its successor: node with next-higher ID

Consistent hashing [Karger '97] – basic lookup

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

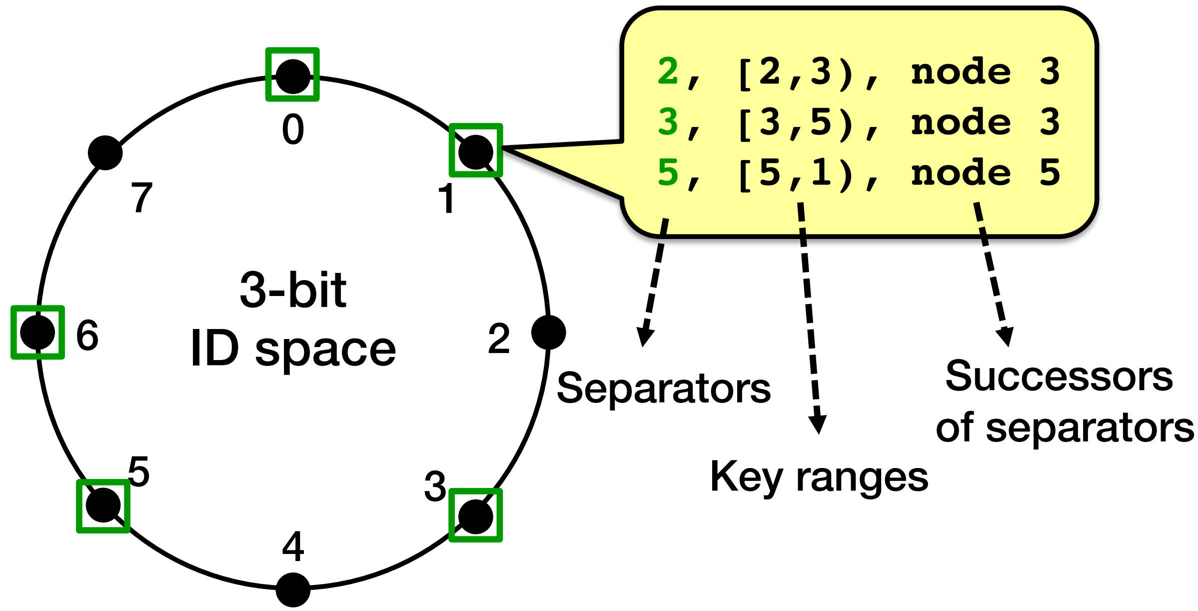


Chord – finger tables

Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node



Each node keeps m states
Key space → m ranges via
$(N+2^{k-1}) \bmod 2^m$, $1 \le k \le m$

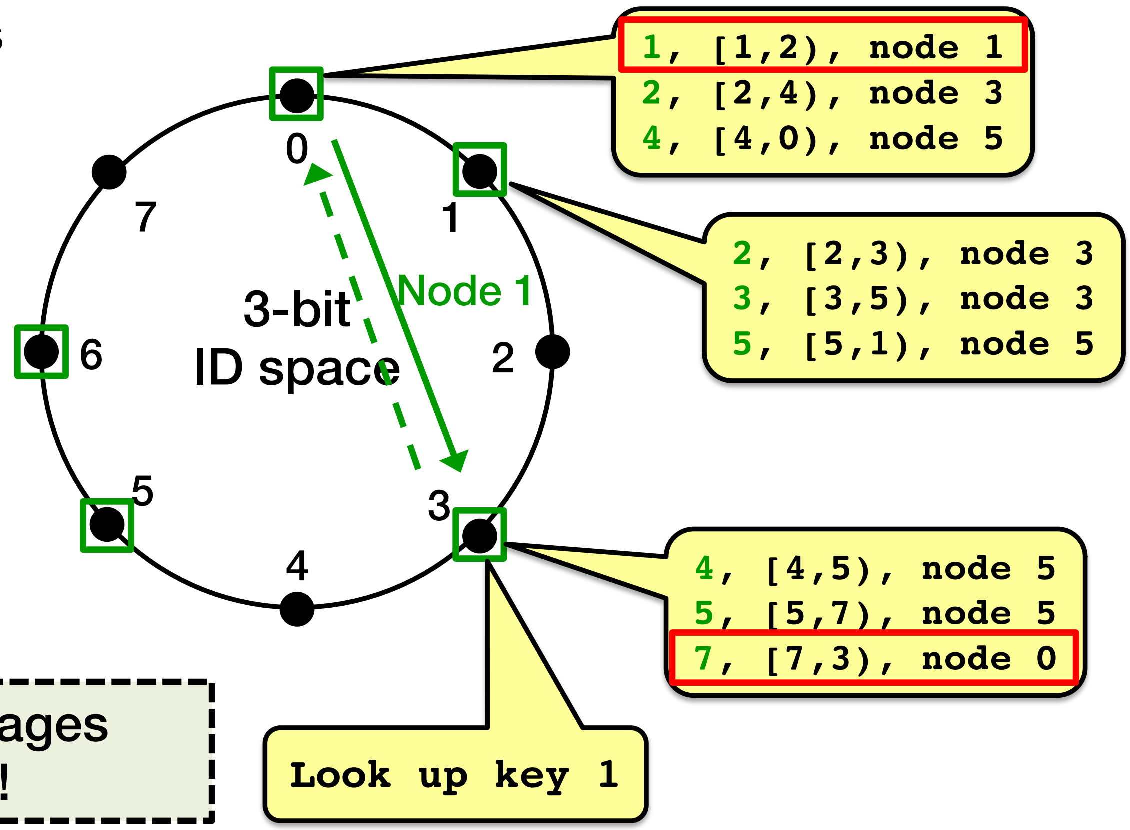
Chord – finger tables

Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$

● Identifiers/key space

□ Node

Each node keeps m states
Key space → m ranges via
$(N+2^{k-1}) \bmod 2^m$, $1<=k<=m$



O(logN) messages
and hops!

Implication of finger tables

- A **binary lookup tree** rooted at every node
  - Threaded through other nodes' finger tables
- Better than arranging nodes in a single tree
  - Every node acts as a root
    - So there's **no root hotspot**
    - **No single point** of failure
    - But a **lot more state** in total

Chord lookup algorithm properties

- **Interface:** lookup(key) → IP address

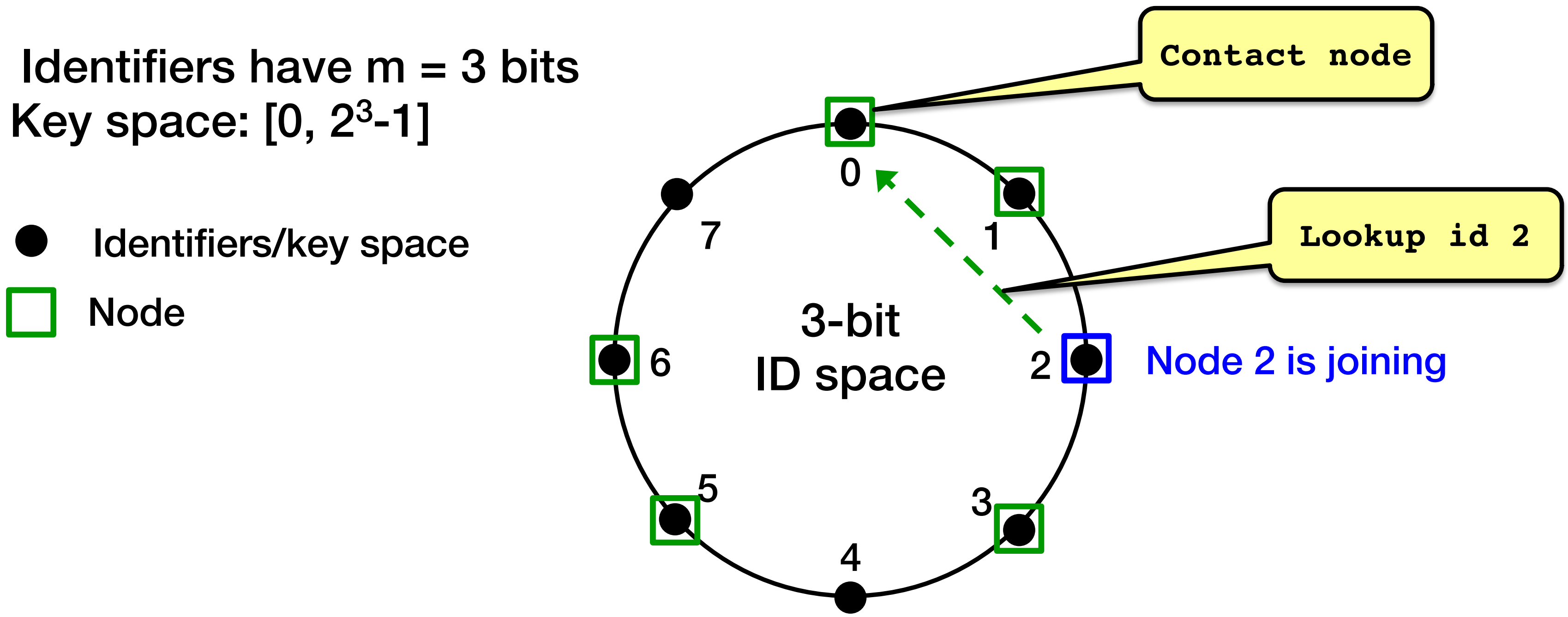
- **Efficient:** O(log N) messages per lookup
  – N is the total number of nodes (peers)

- **Scalable:** O(log N) state per node

- **Robust:** survives massive failures

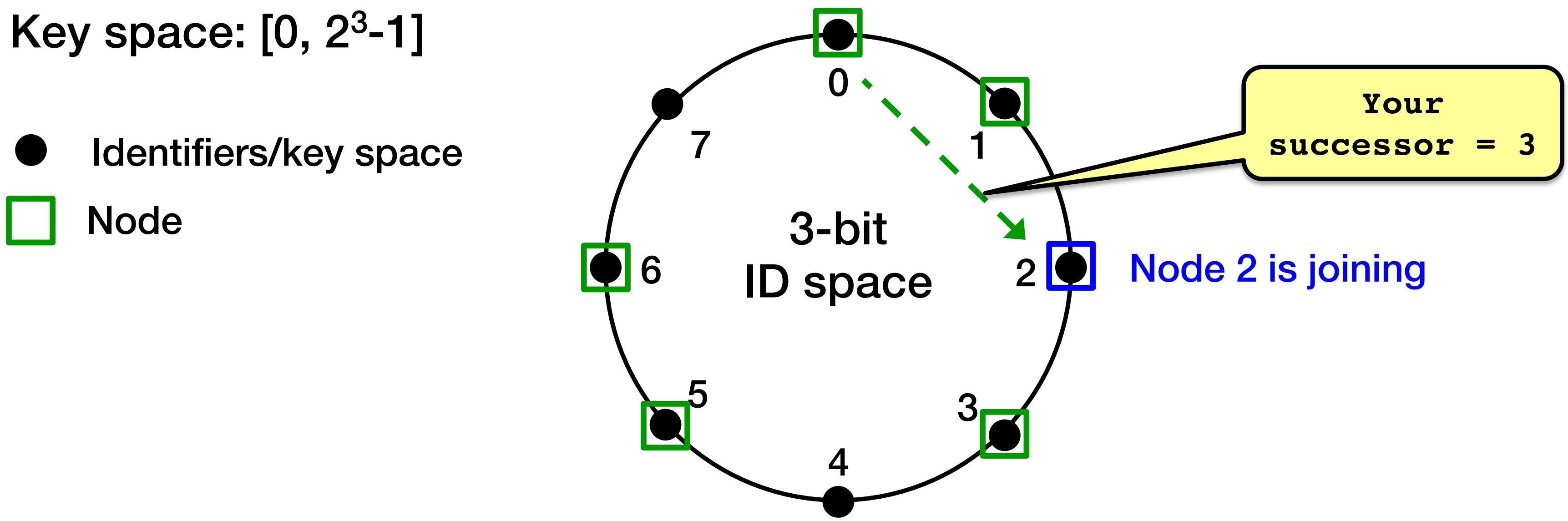
Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Chord – node joining

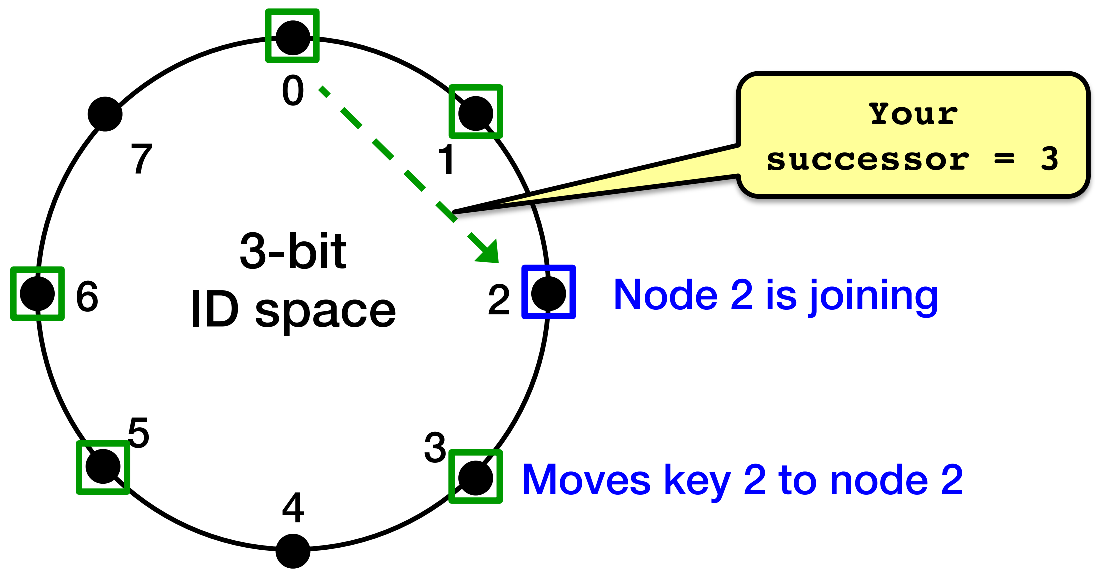
Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Chord – node joining

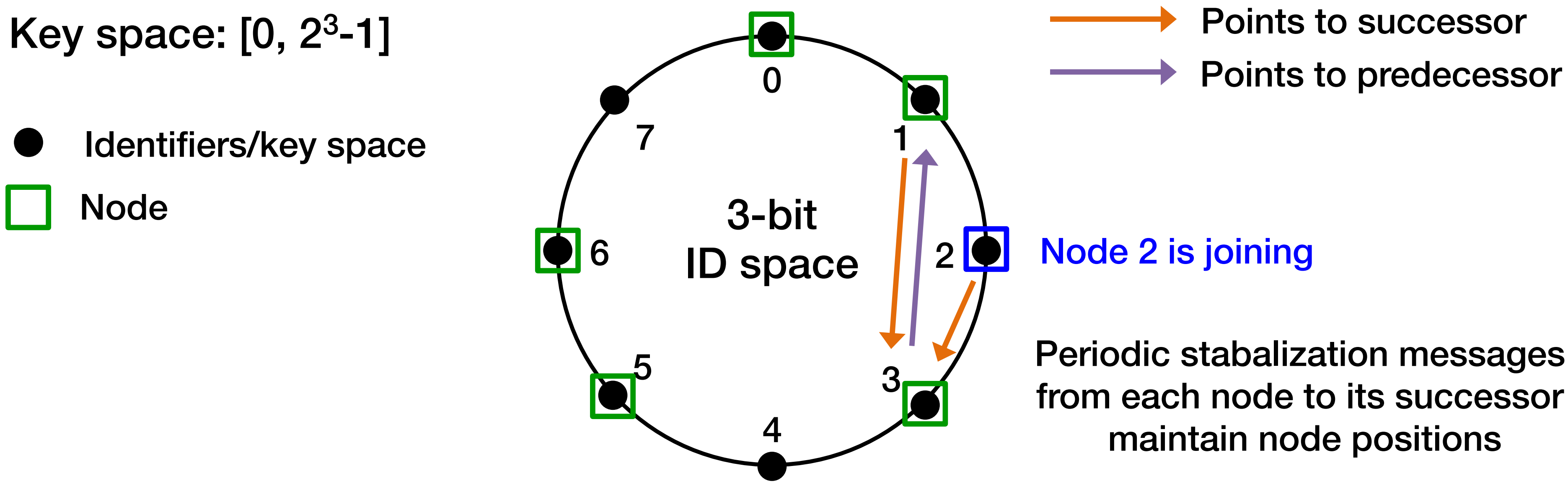
Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$

● Identifiers/key space
□ Node



Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Periodic stabalization messages from each node to its successor maintain node positions

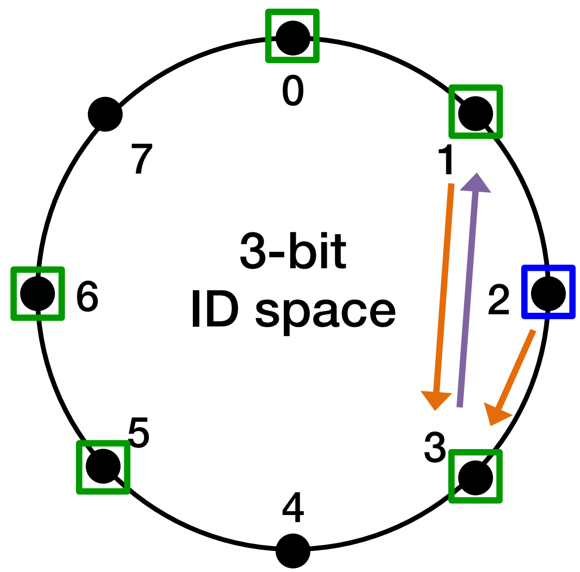
Chord – node joining

Identifiers have m = 3 bits

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

● Identifiers/key space

□ Node



Node 2 is joining

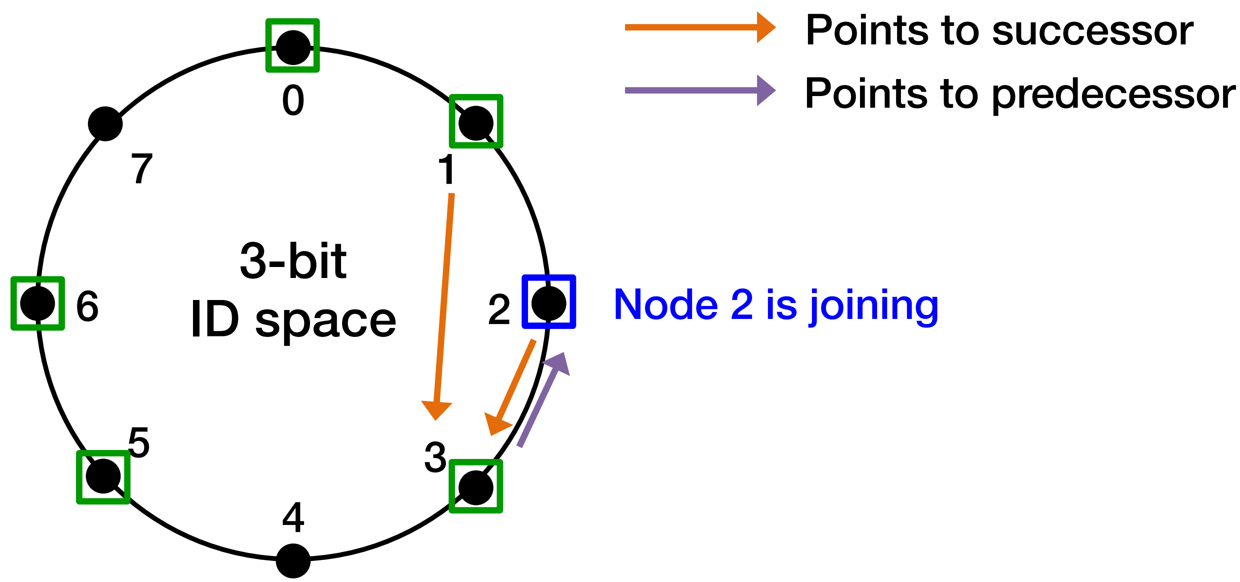
Points to successor

Points to predecessor

Chord – node joining

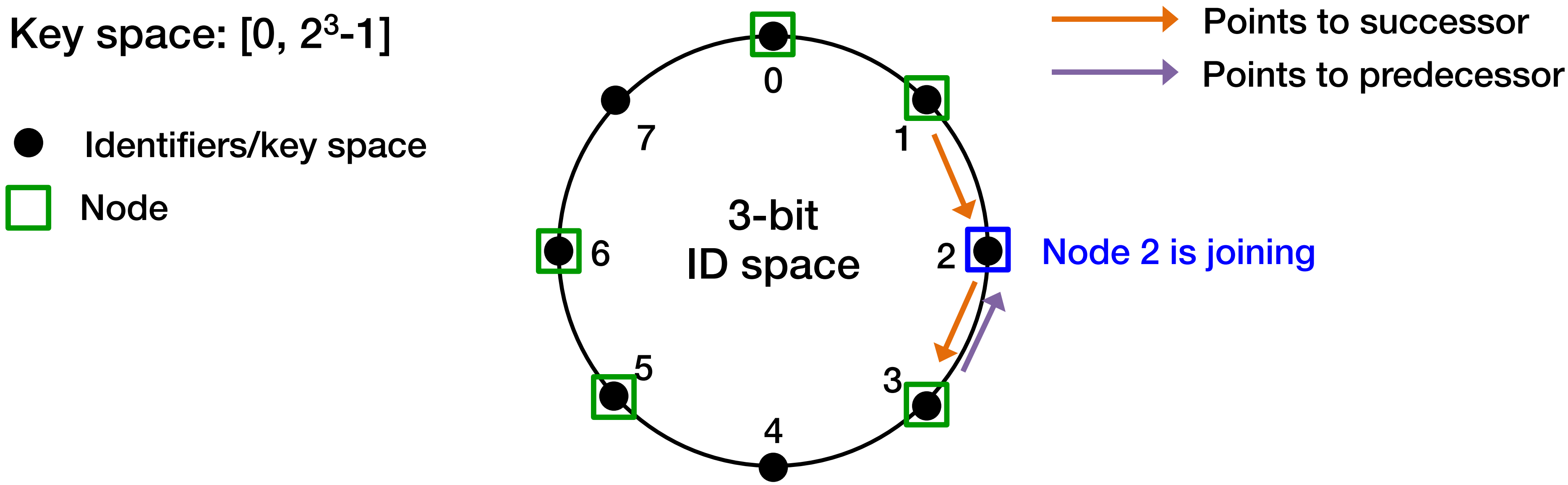
Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$

● Identifiers/key space
□ Node



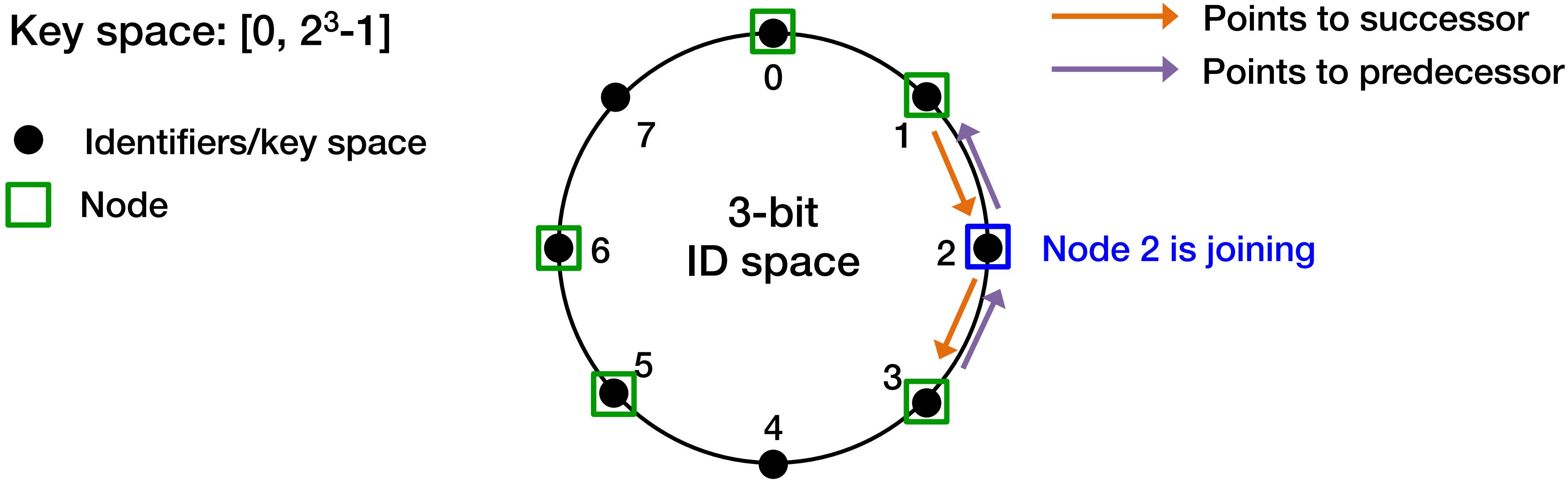
Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Chord – node joining

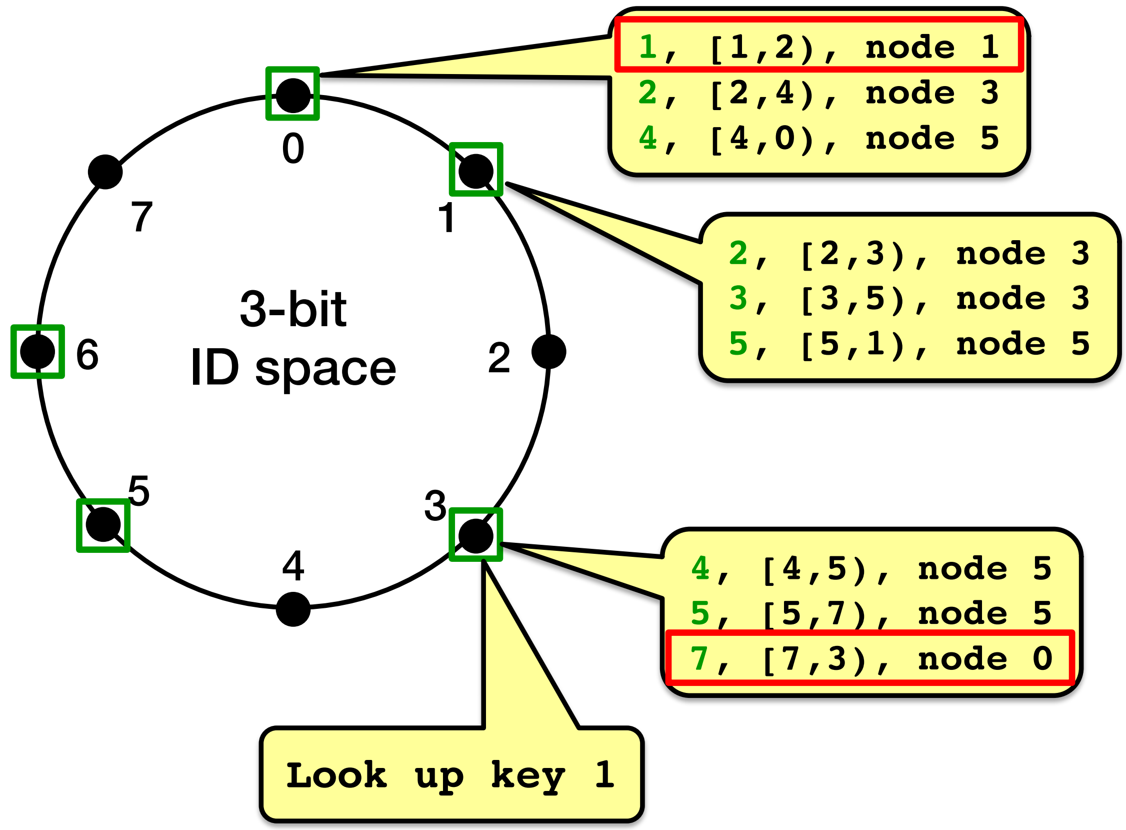
Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Chord – failures and successor list

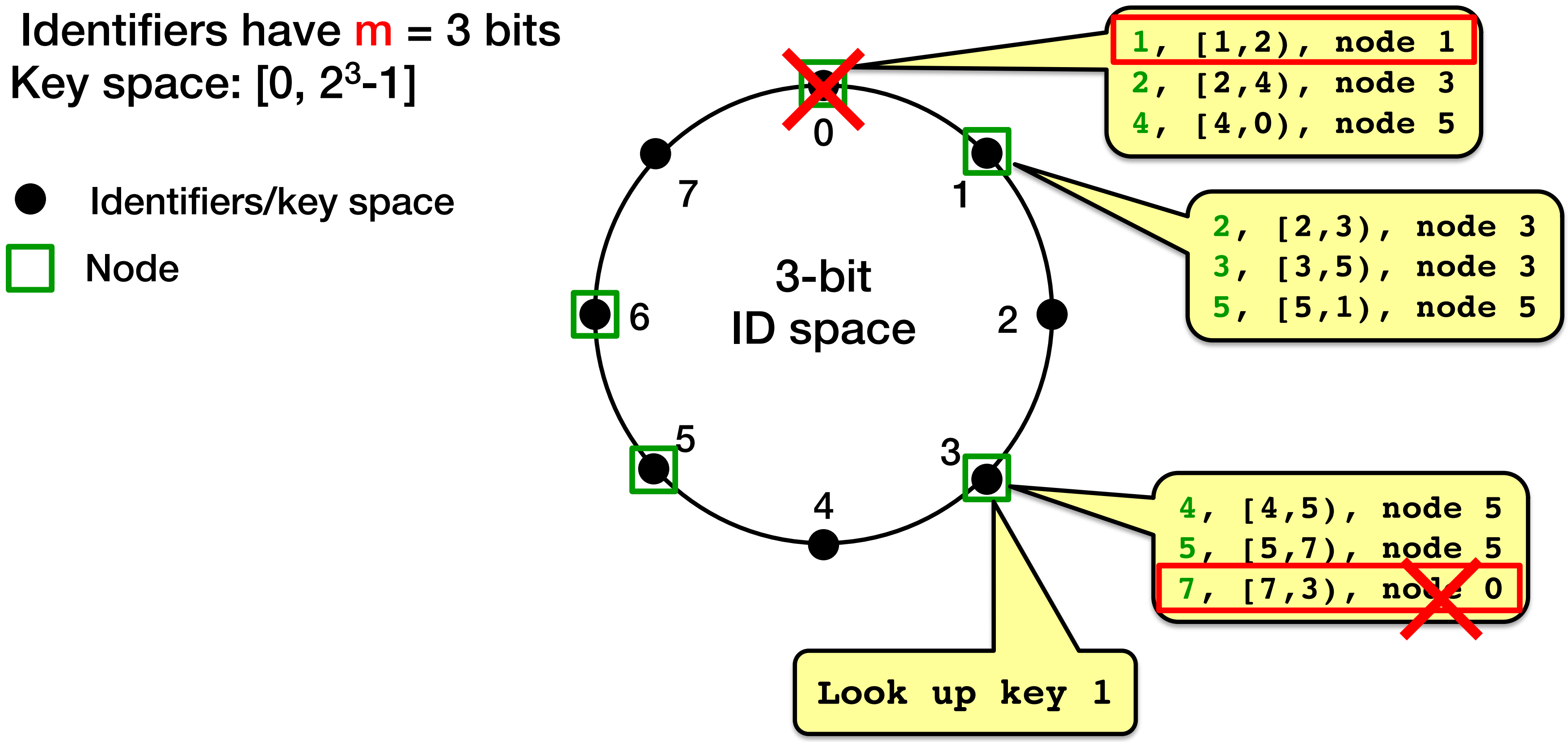
Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



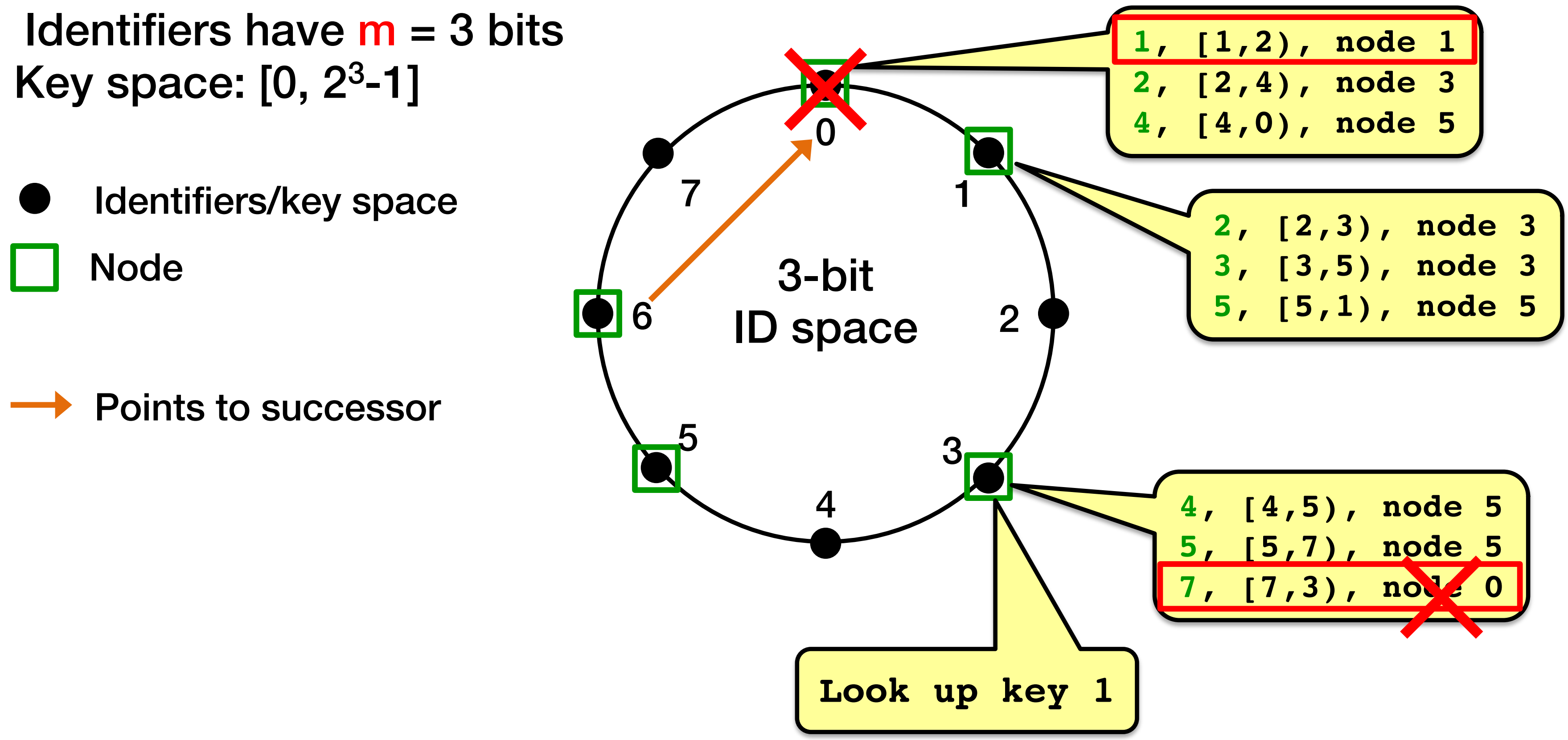
Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



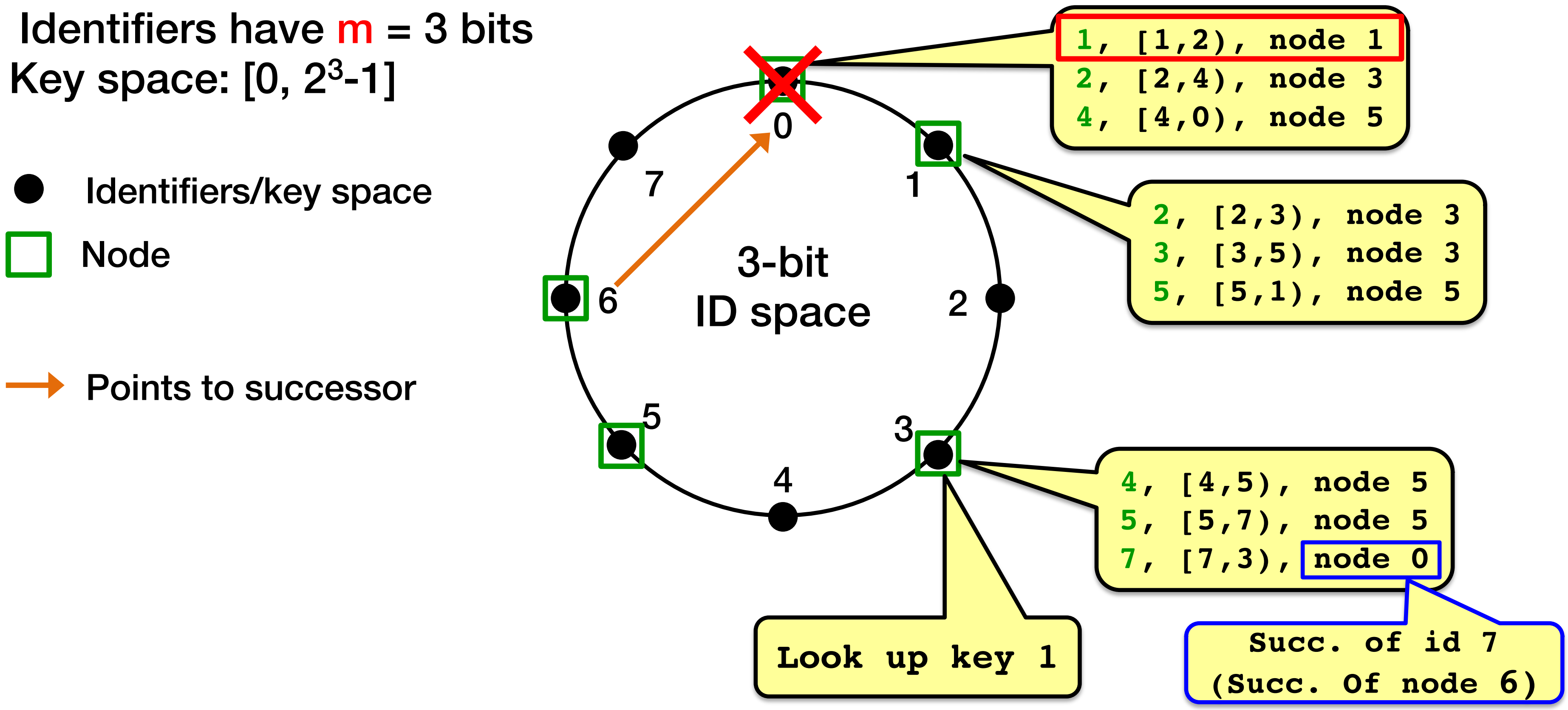
Chord – failures and successor list

Identifiers have m = 3 bits
Key space: $[0, 2^3-1]$



Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]



Chord – failures and successor list

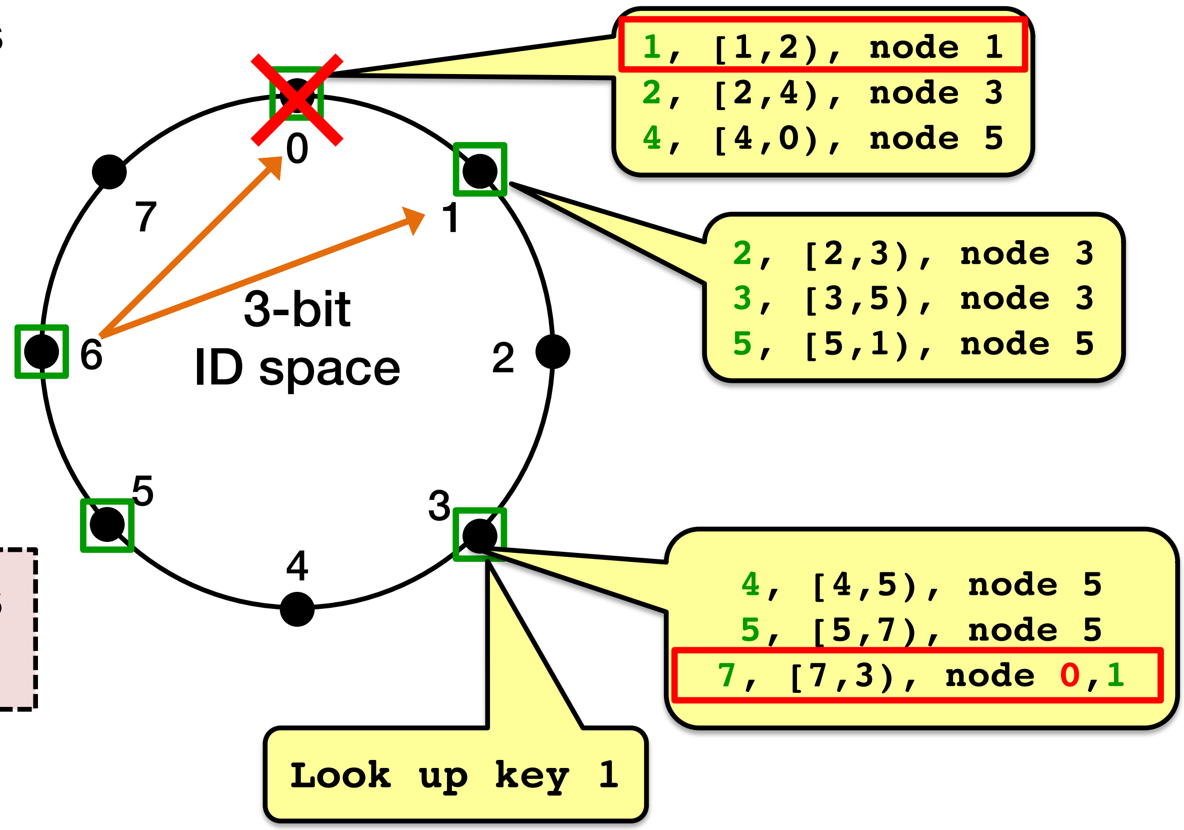
Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space

□ Node

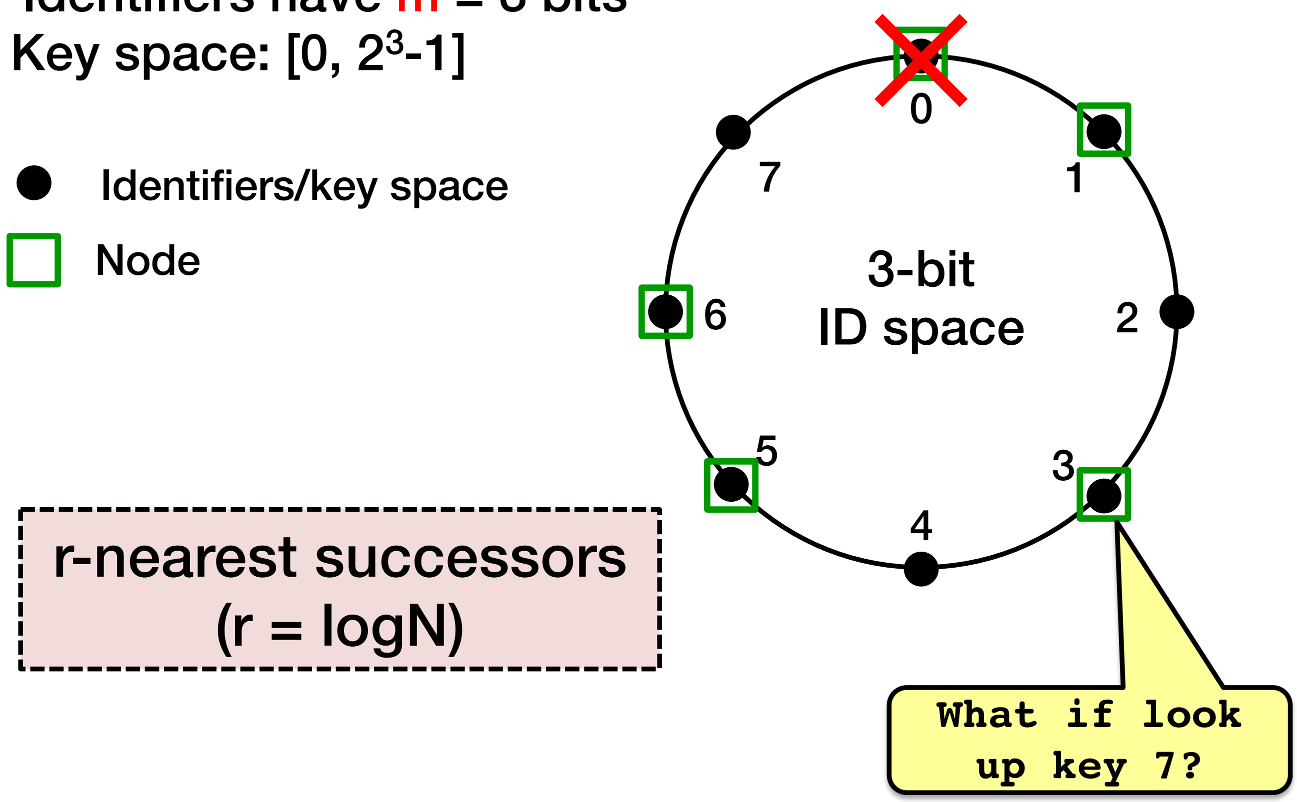
→ Points to successor

r-nearest successors
(r = logN)



Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

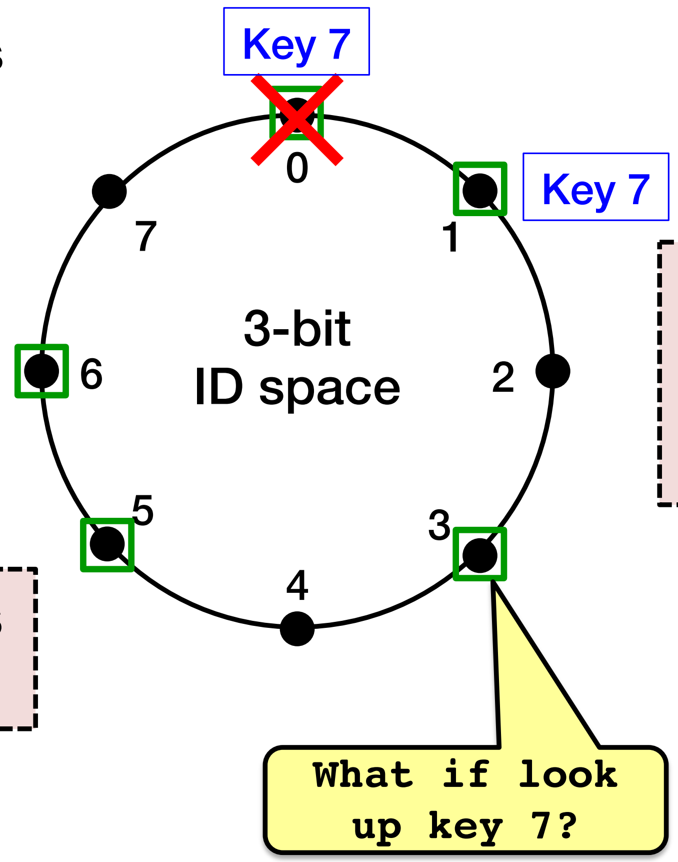


r-nearest successors
(r = logN)

DHash replicates blocks at *r* successors

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



r-nearest successors
(r = logN)

"Adjacent" nodes in the ring may be far away in the network
→ Independent failures

What DHTs got right

- **Consistent hashing**
  - Elegant way to divide a workload across machines
  - Very useful in clusters: actively used in Amazon Dynamo and other systems
- **Replication** for high availability, efficient recovery
- **Incremental scalability**
  - Peers join with capacity, CPU, network, etc.
- **Self-management:** minimal configuration