OVERLAY NETWORKS, P2P NETWORKS, CHORD, AND DYNAMODB

Module 4

Fall 2020

George Porter

UC San Diego

ATTRIBUTION

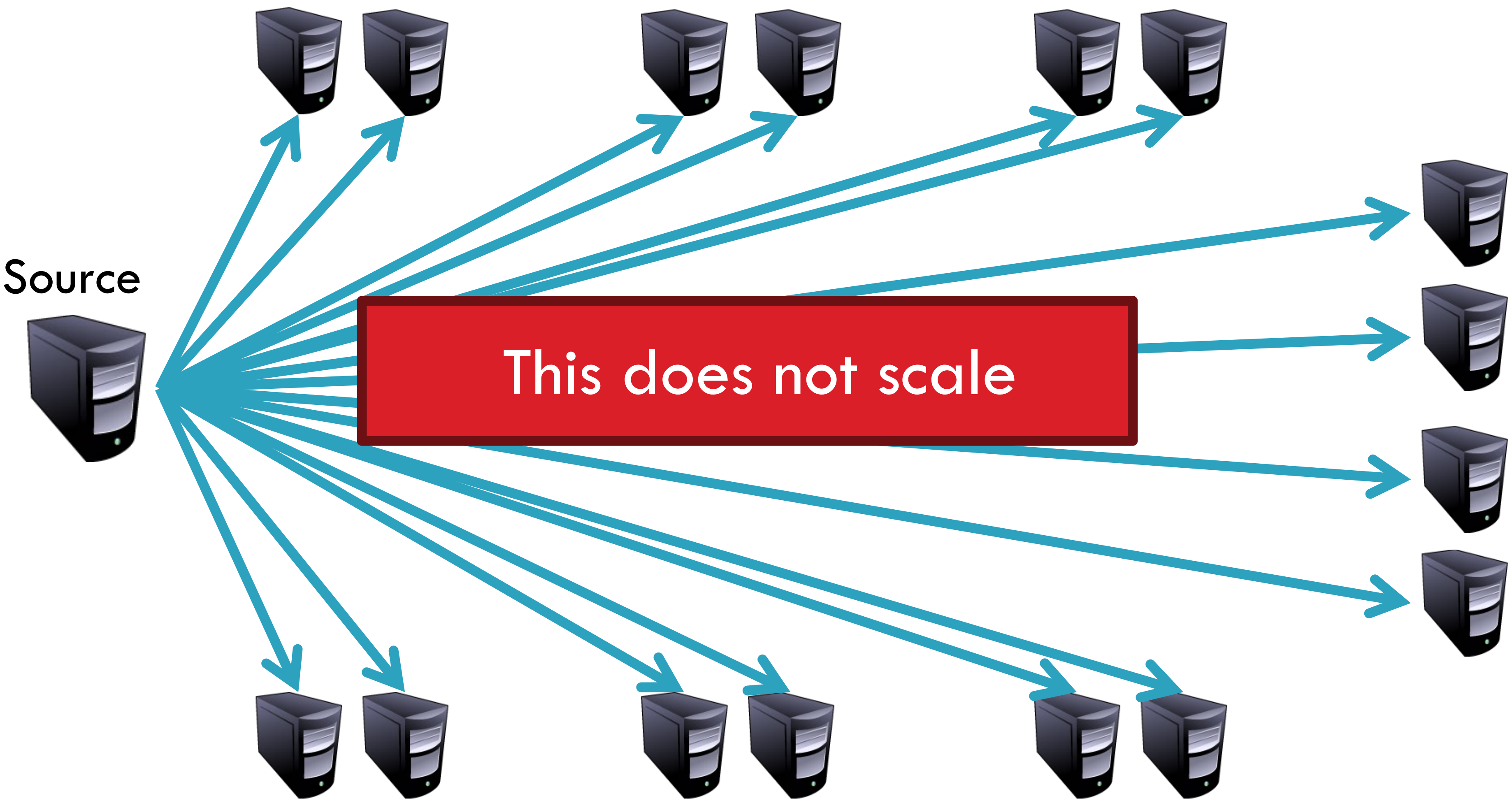
- These slides incorporate material from:
  - Christo Wilson, NEU (used with permission)
  - Tanenbaum and Van Steen, 3rd edition

UC San Diego

Outline

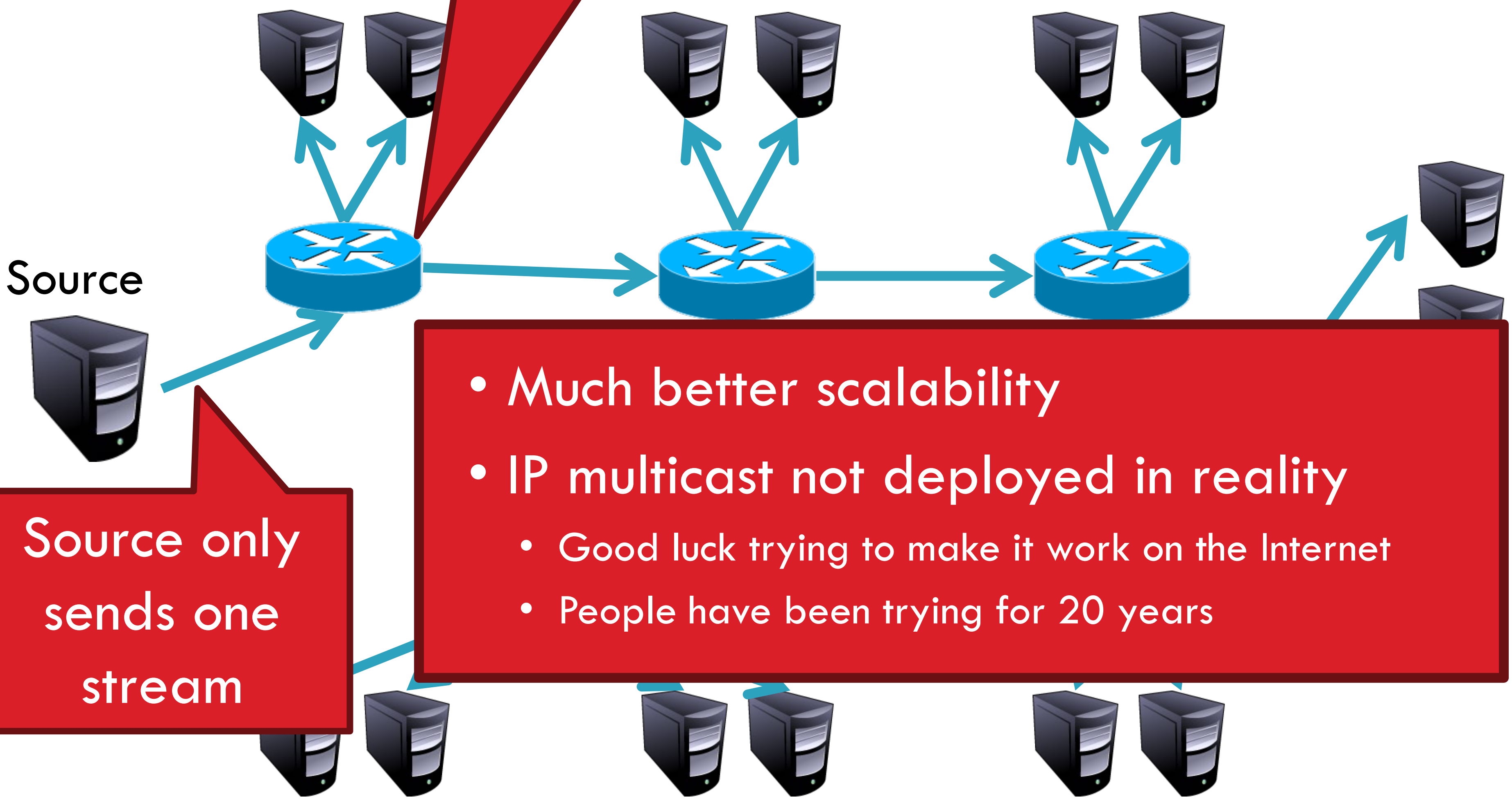
- Overlay networks
- Peer-to-peer networks
- Chord DHT
- DynamoDB DHT

Unicast Streaming Video



IP M… ming Video

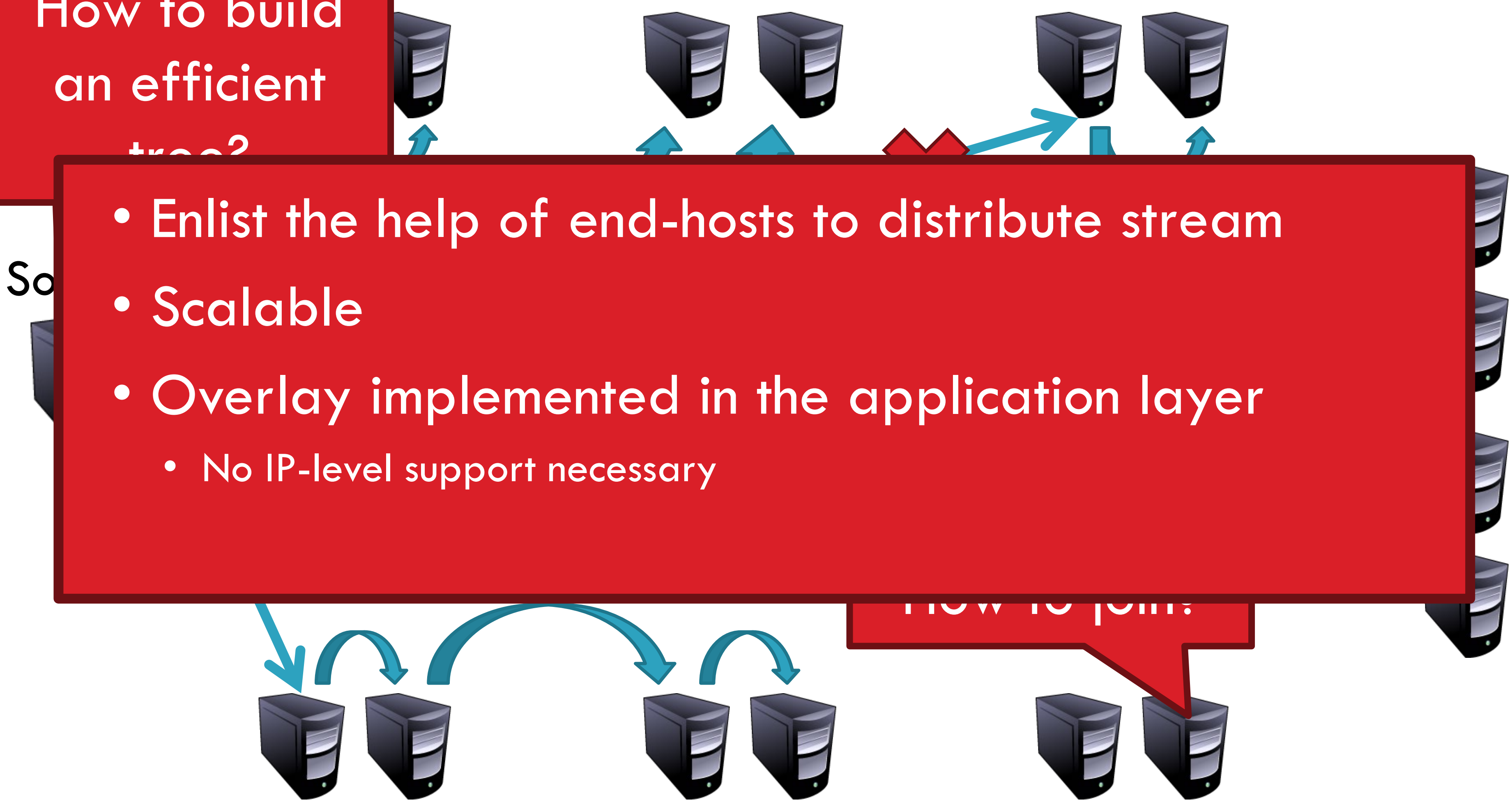
IP routers forward to multiple destinations



Source only sends one stream

- Much better scalability
- IP multicast not deployed in reality
  - Good luck trying to make it work on the Internet
  - People have been trying for 20 years

End System Multicast Overlay



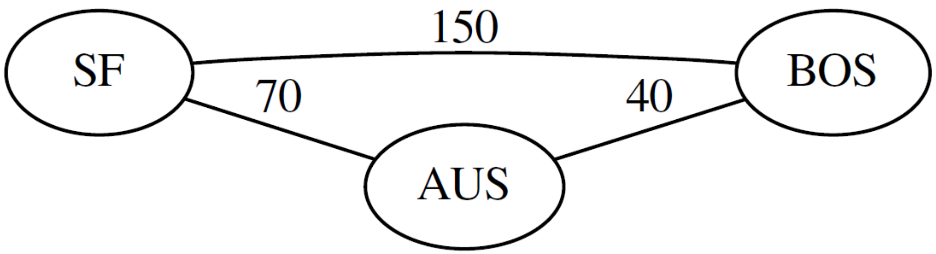
How to build an efficient tree?

- Enlist the help of end-hosts to distribute stream
- Scalable
- Overlay implemented in the application layer
  - No IP-level support necessary

Choosing multicast overlay paths

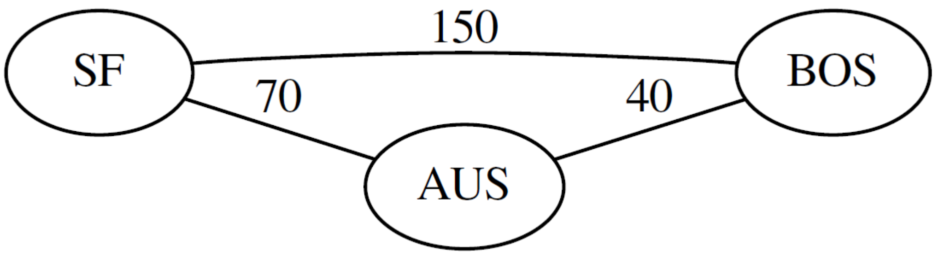
- Measurements
  - Time series of RTT measurements
  - Observed throughputs of transfers
- How to select overlay distribution tree edges?

INEFFICIENT PATHS



- SF->BOS is one hop, 150ms
- But why not two-hop, 70+40=110ms??

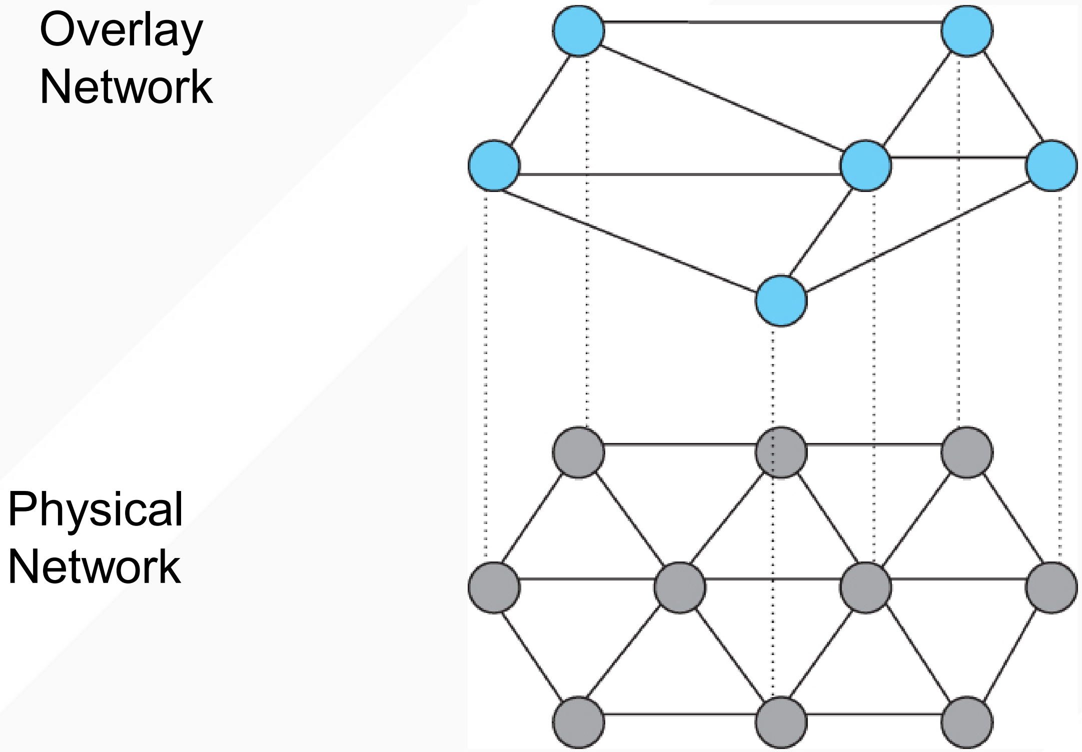
STANDARD INTERNET ROUTING



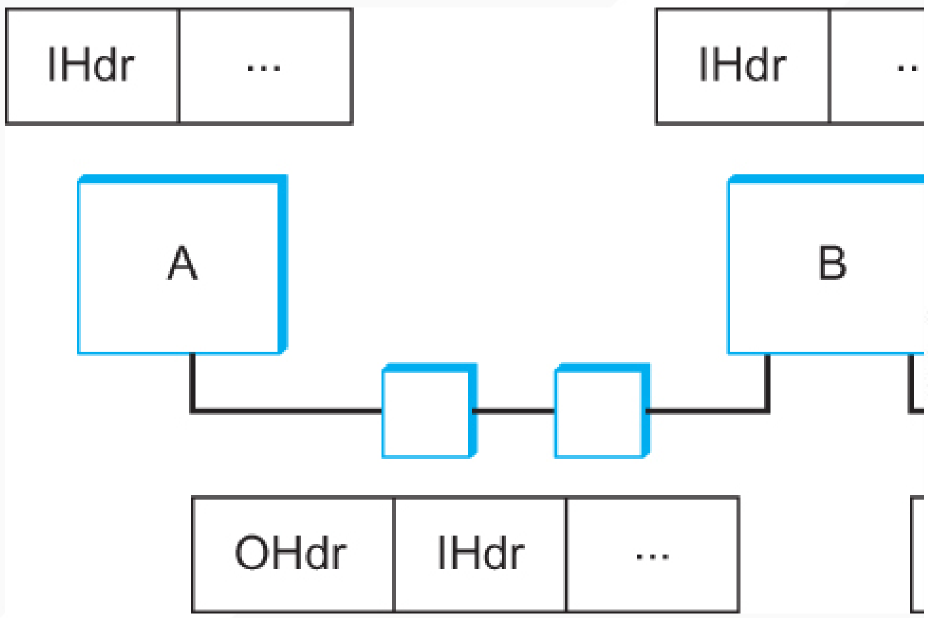
- Paths chosen via Border Gateway Protocol (BGP)
  - Based on business relationships, contracts, etc.
  - Not (necessarily) based on latency or bandwidth

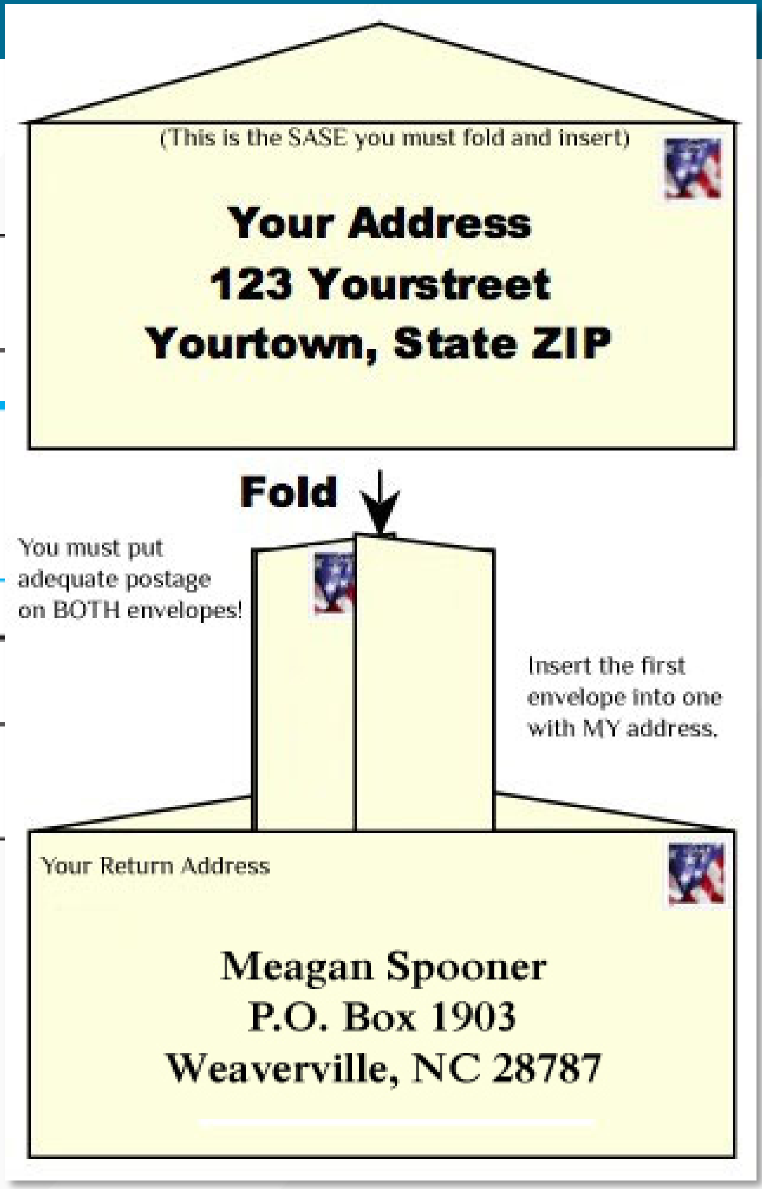
OVERLAY NETWORKS

- Control routing *at the application layer*



KEY TECHNIQUE: TUNNELS

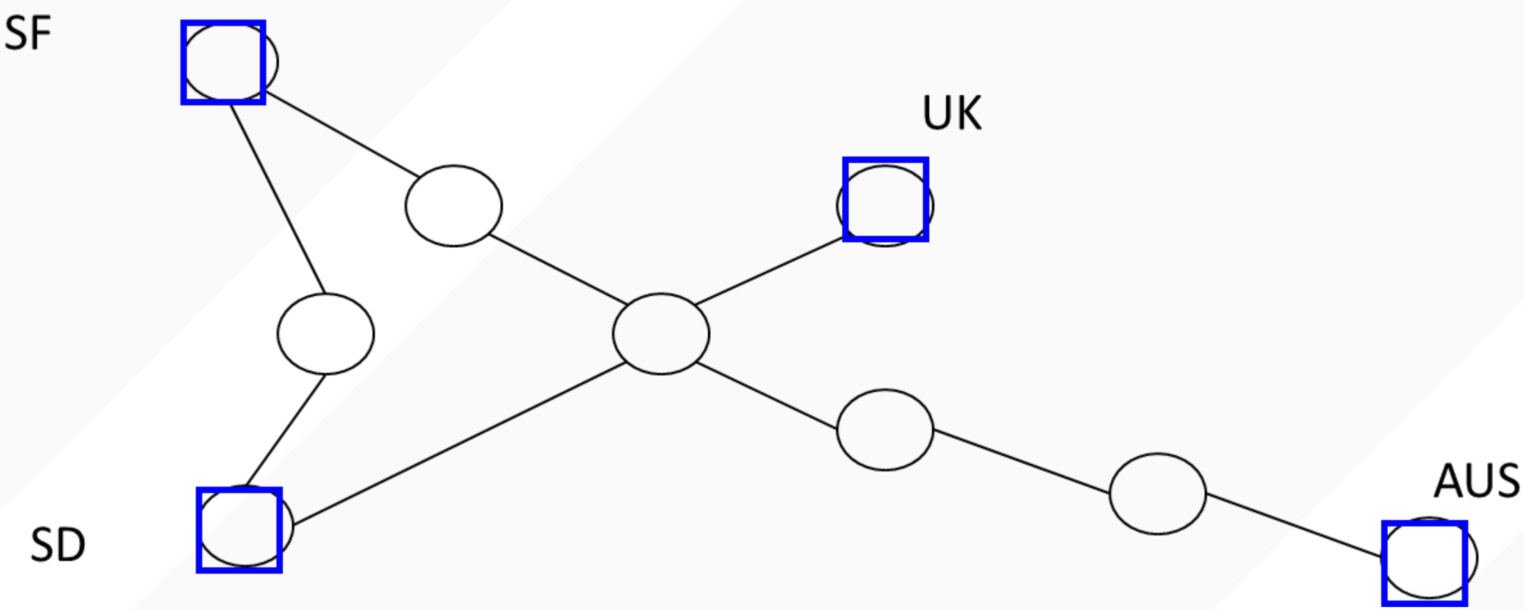




WHY OVERLAYS?

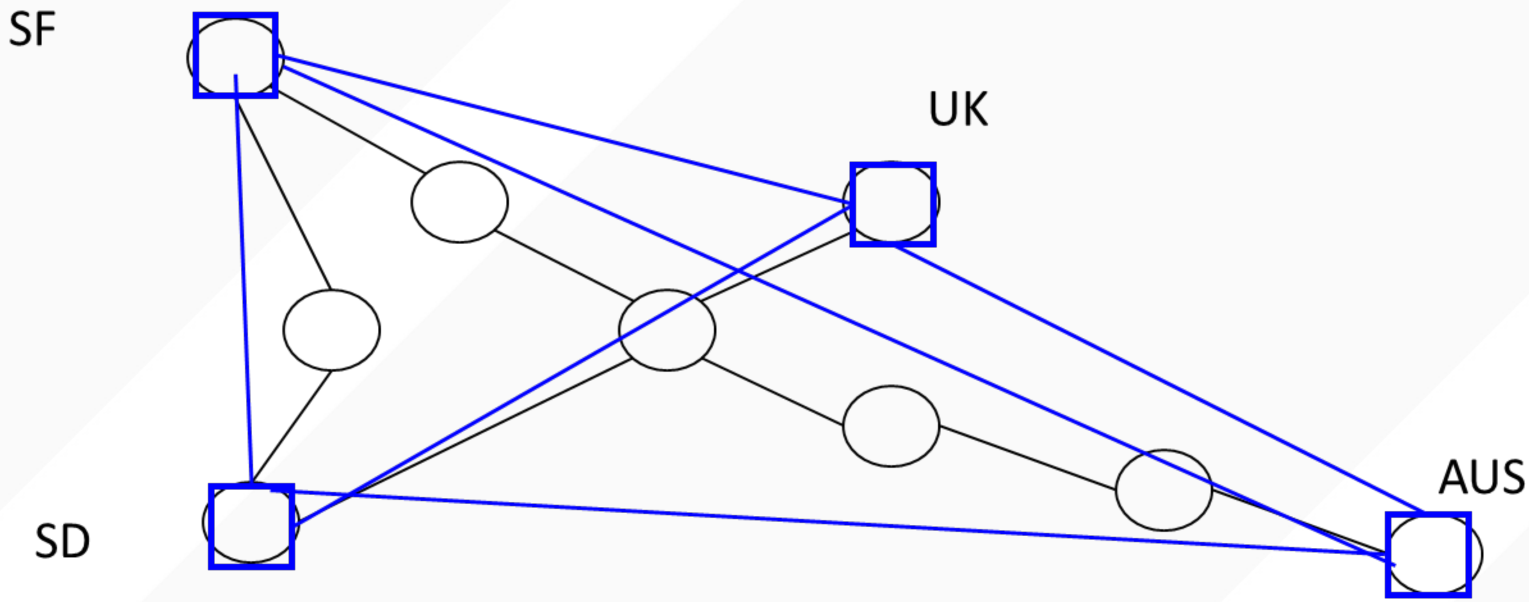
- For routing
  - Get better than best-effort service on the Internet
  - For reliability (what if a link on the internet drops p percent of packets?)
- To locate data
  - We’ll look at Chord and DynamoDB DHTs
- For security
  - What if you encrypt the overlay?
  - Also called a VPN (Virtual Private Network)

THE NEED FOR MULTICAST

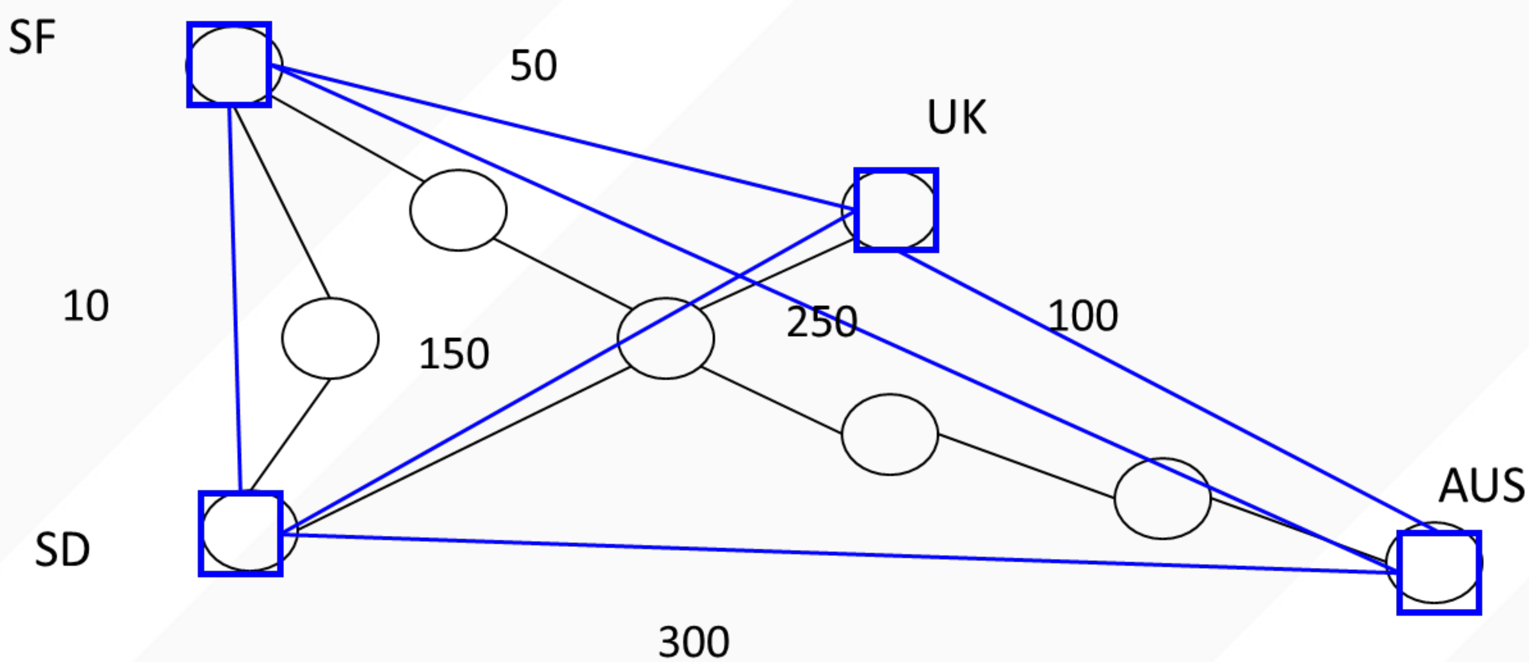


- Imagine each office wants to send a message to every other office

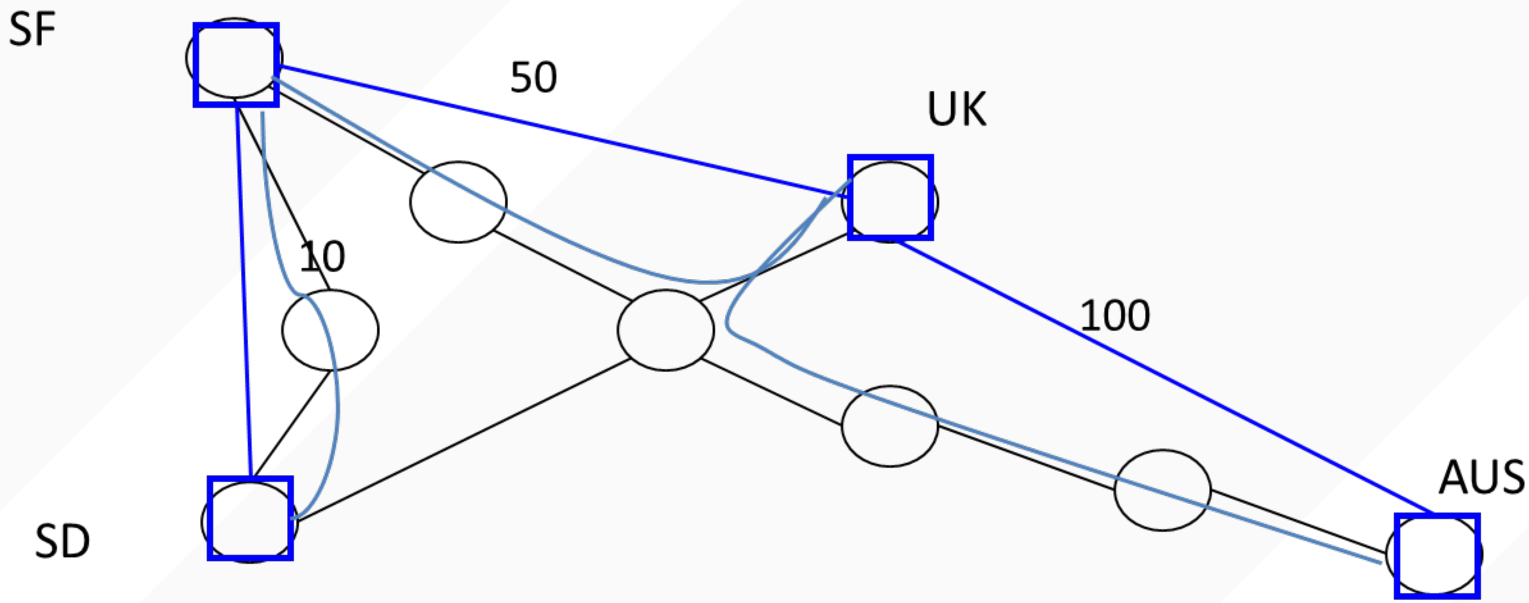
STRAWMAN SOLUTION: JUST SEND EVERYWHERE



CONSTRUCTING AN OVERLAY MULTICAST TREE



CHOOSE MINIMUM COST SPANNING TREE



OVERLAY NETWORKS FOR PERFORMANCE

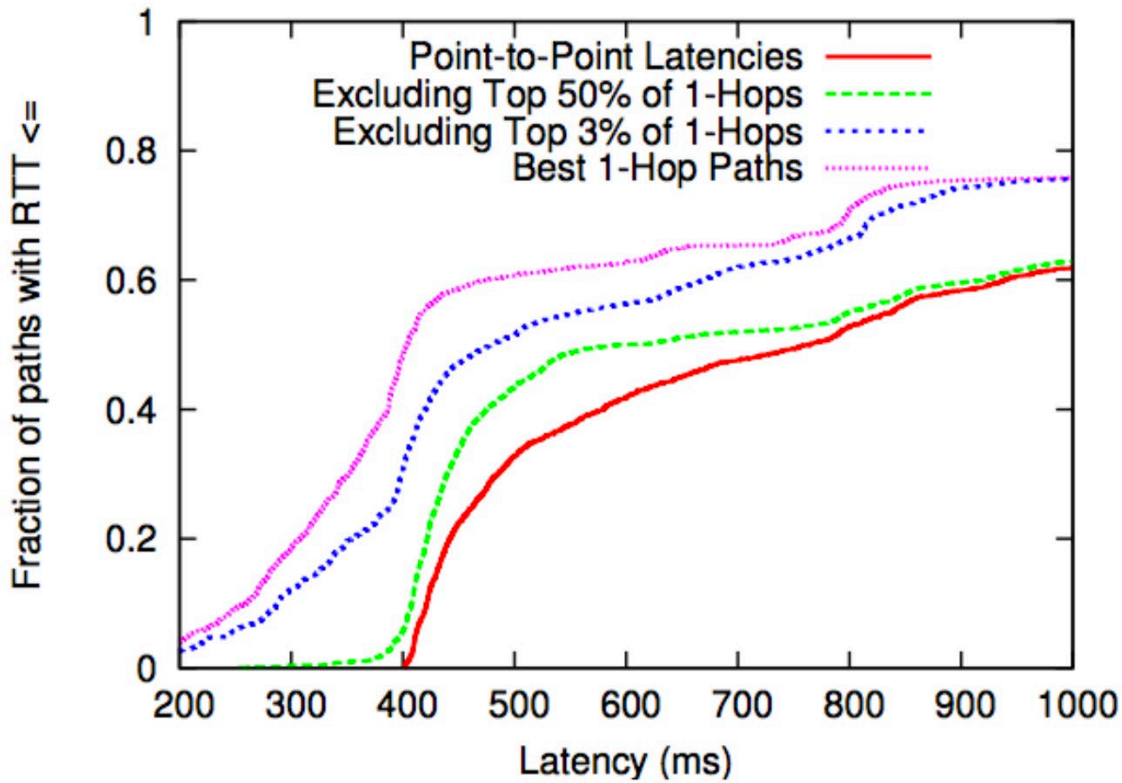
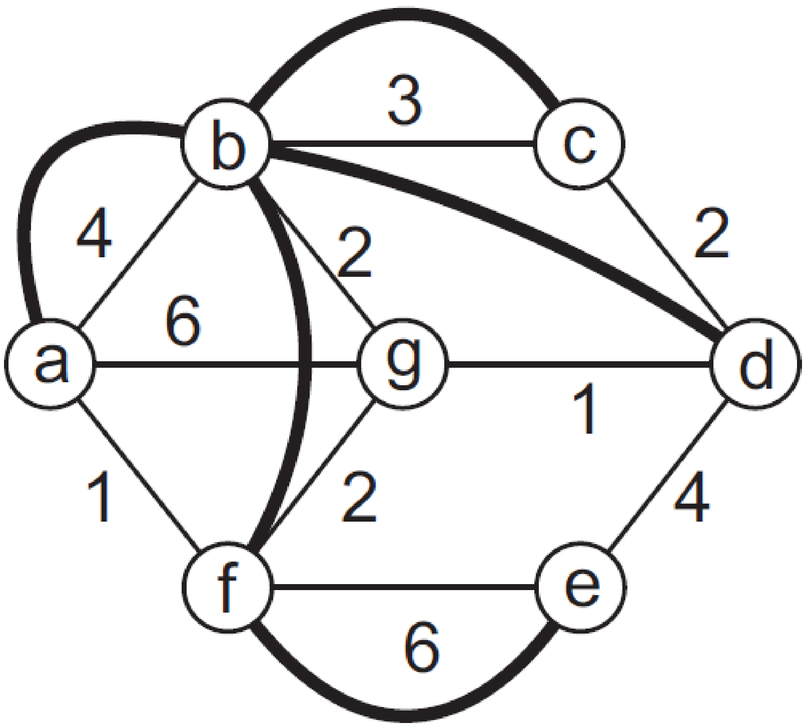


Figure 1: Comparison of RTT for pairs of PlanetLab hosts whose point-to-point latencies were larger than 400 ms (high-latency paths). For the "excluding top n%" graphs, we removed the top n% of one-hop alternatives for each high-latency path from consideration, then used the best remaining one-hop.

Sontag et al.,
CoNEXT 2009

APP-LAYER OVERLAY EXAMPLE

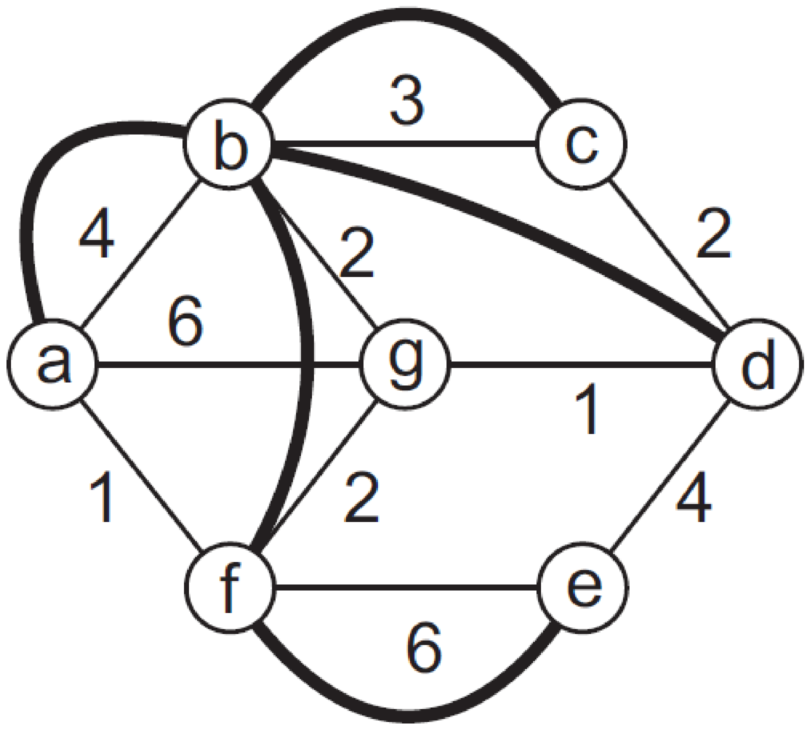


- "Tree" constructed using application-layer sockets
- Data flows along tree, not underlying network
- Why?
  - Can improve reliability
    - If link from B->G fails, can take few minutes for Internet to recover
      (meanwhile app can respond in milliseconds to create new path)
  - Disseminate data in a scalable way
  - Avoid censorship

KEY CONCEPTS

- Link stress
  - How often a packet transits a given link
- Relative delay penalty (aka “Stretch”)
  - Ratio of delay in overlay vs. underlying network

APP-LAYER OVERLAY EXAMPLE

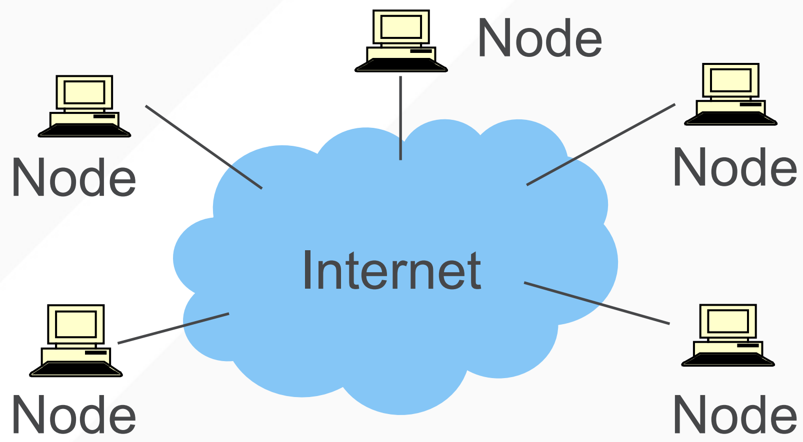


- Network cost A -> F
  - 1
- Overlay cost A -> F
  - 4 + 2 + 2 = 8
- Relative delay penalty A -> F
  - 8/1

Outline

- Overlay networks
- Peer-to-peer networks
- Chord DHT
- DynamoDB DHT

PEER-TO-PEER (P2P) NETWORKS

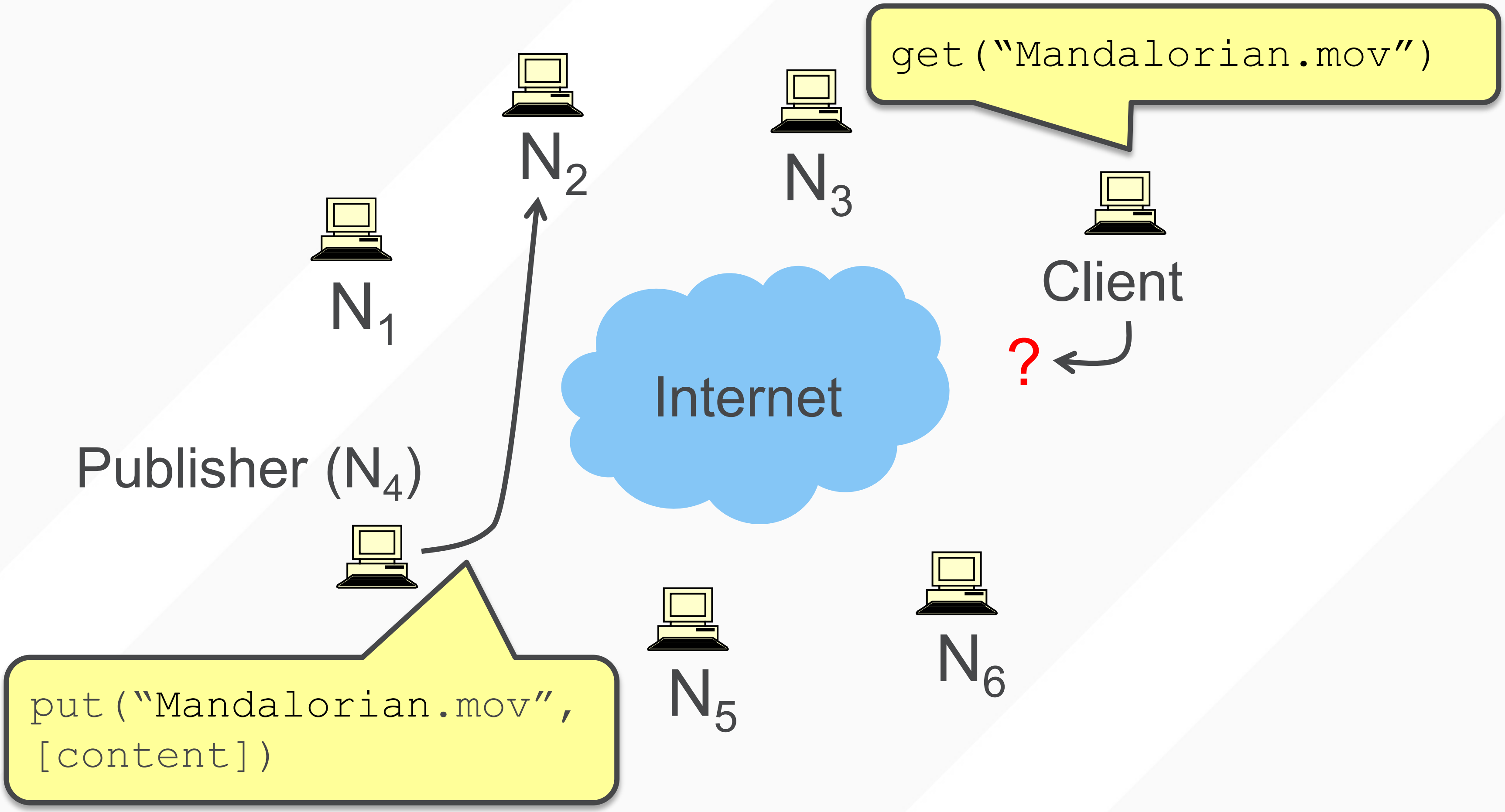


- A **distributed** system architecture:
  - **No centralized control**
  - Nodes are **roughly symmetric** in function
- **Large** number of **unreliable** nodes (could be reliable too)

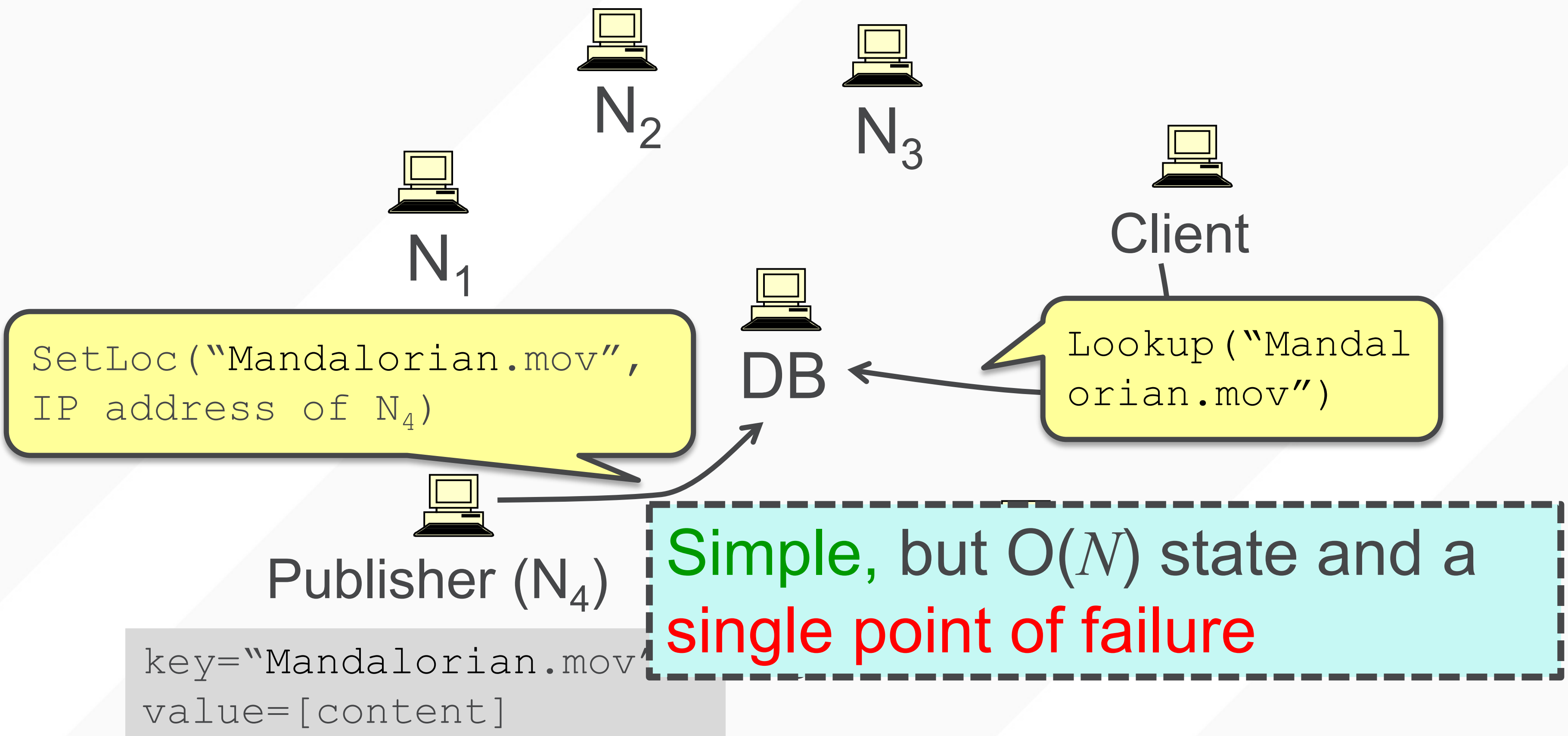
WHAT IS “FLAT” NAMING?

- The name doesn’t give you an indication of where the data is located
- Flat:
  - MAC address: 00:50:56:a3:0d:2a
- Vs hierarchical:
  - IP address: 206.109.2.12/24
  - DNS name: starbase.neosoft.com

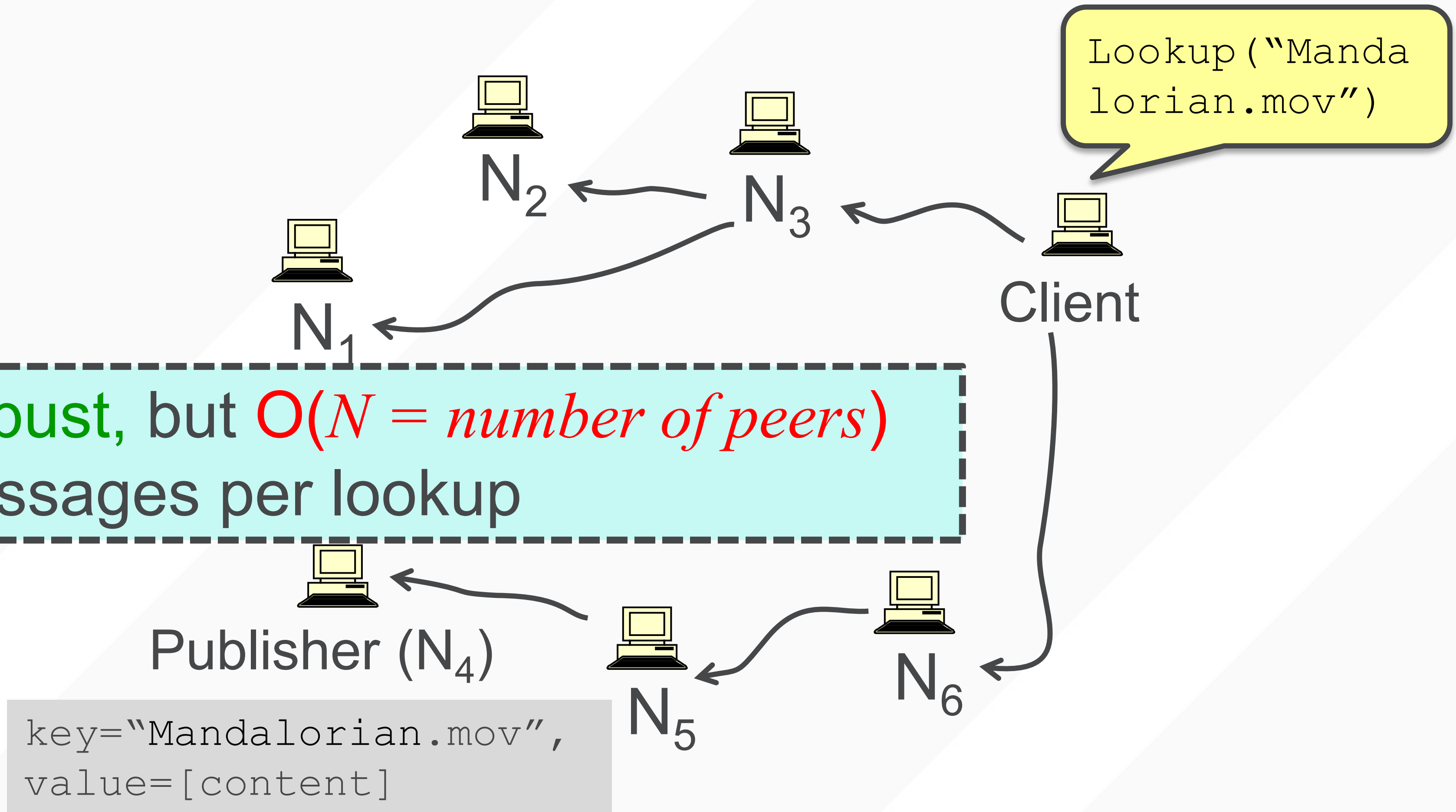
FLAT NAME LOOKUP PROBLEM



CENTRALIZED LOOKUP (NAPSTER)

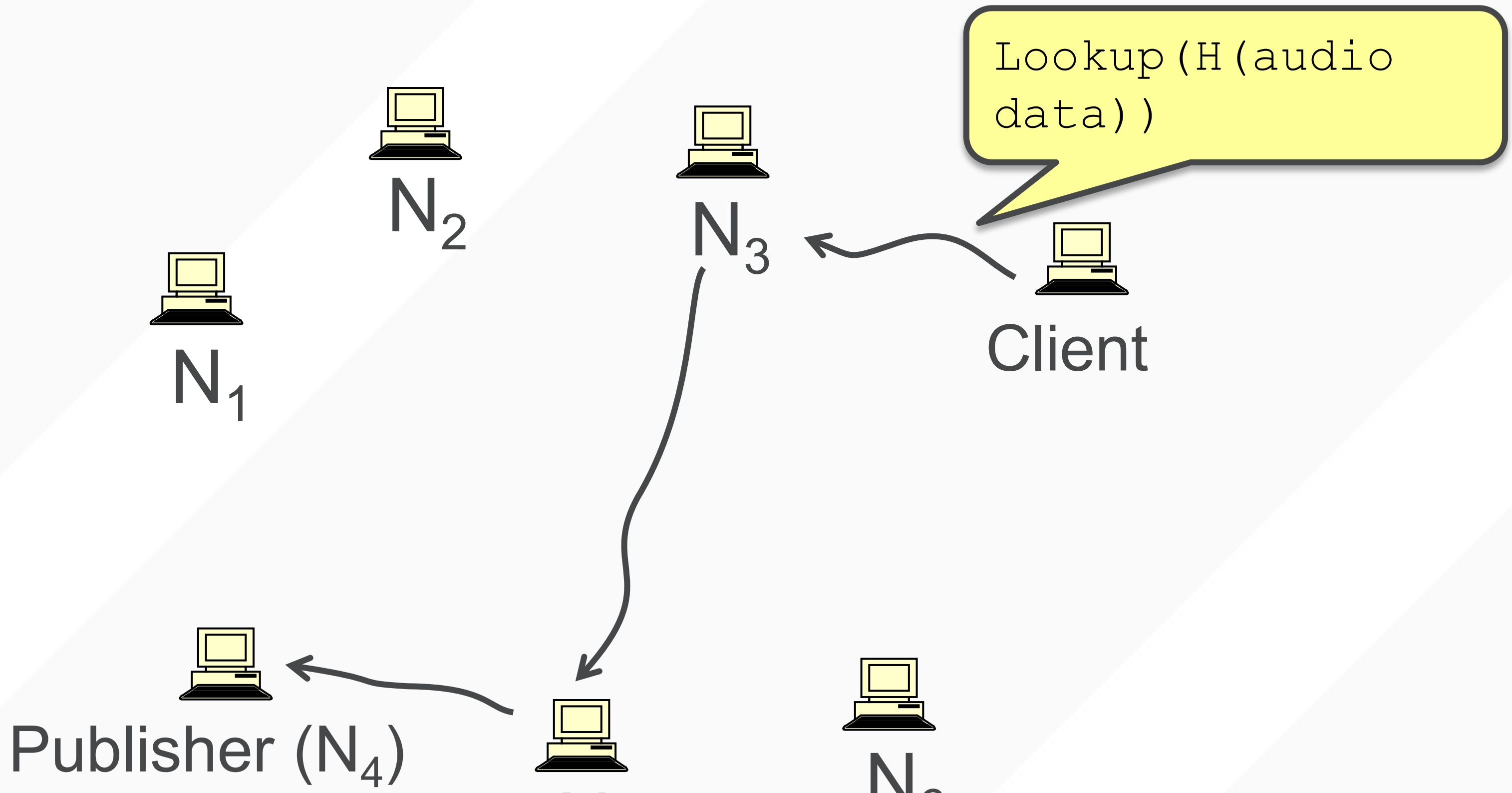


FLOODED QUERIES (ORIGINAL GNUTELLA)



Robust, but $O(N = number\ of\ peers)$ messages per lookup

ROUTED DHT QUERIES (CHORD AND DYNAMODB)



Can we make it robust, reasonable state, reasonable number of hops?

Outline

- Overlay networks
- Peer-to-peer networks
- Chord DHT
- DynamoDB DHT

SYSTEMATIC FLAT NAME LOOKUPS VIA DHTS

- Local hash table:

```
key = Hash(name)
put(key, value)
get(key) → value
```

- **Service:** Constant-time insertion and lookup

How can I do (roughly) this across millions of hosts on the Internet or within a giant datacenter application?
Distributed Hash Table (DHT)

WHAT IS A DHT (AND WHY)?

- Distributed Hash Table:

  `key = hash(data)`

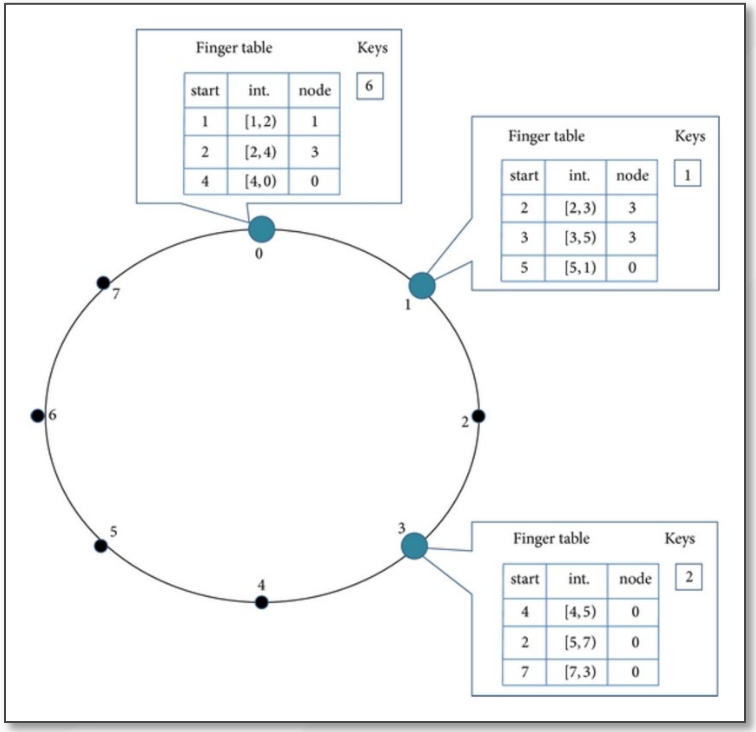
  `lookup(key)` → **`IP addr`**

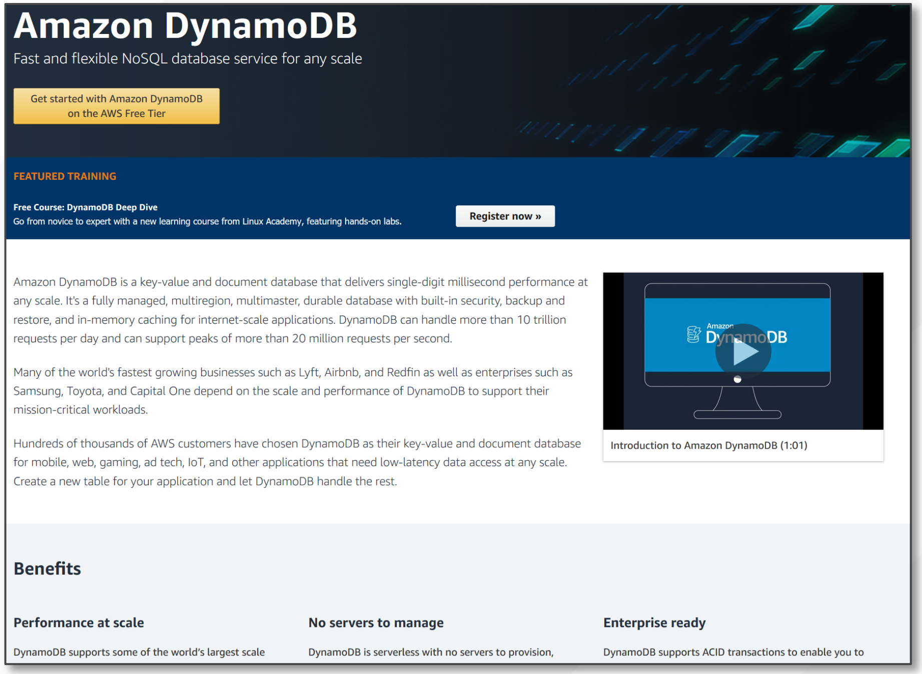
  `send-RPC(IP address, `**`put`**`, key, data)`

  `send-RPC(IP address, `**`get`**`, key)` → `data`

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

TWO EXAMPLES OF DHTS





- Chord
  - Fully decentralized
  - Over wide-area Internet
  - Designed for millions of end points
- DynamoDB
  - Managed within a single datacenter
  - Some centralization
  - 10s to 100s of end points

CHORD LOOKUP ALGORITHM PROPERTIES

- **Interface:** lookup(key) $\rightarrow$ IP address

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

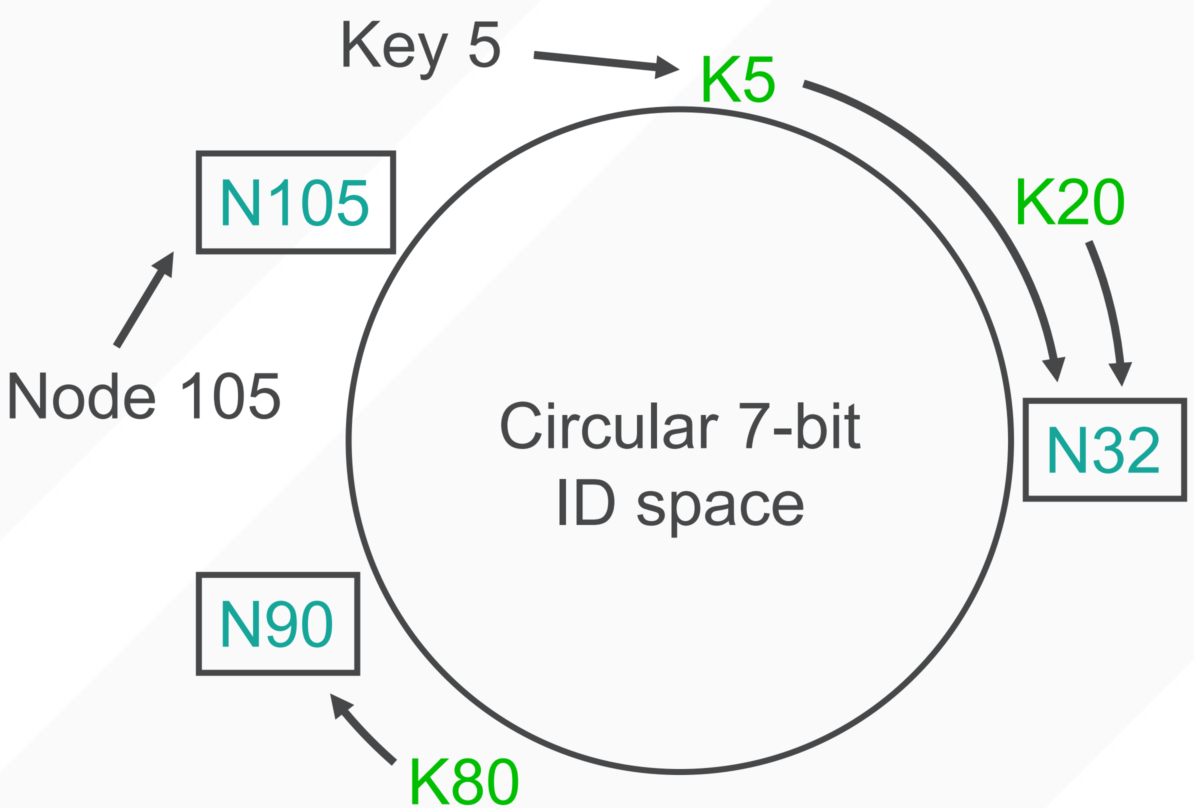
- **Scalable:** O(log N) state per node

- **Robust:** survives massive failures

CHORD IDENTIFIERS

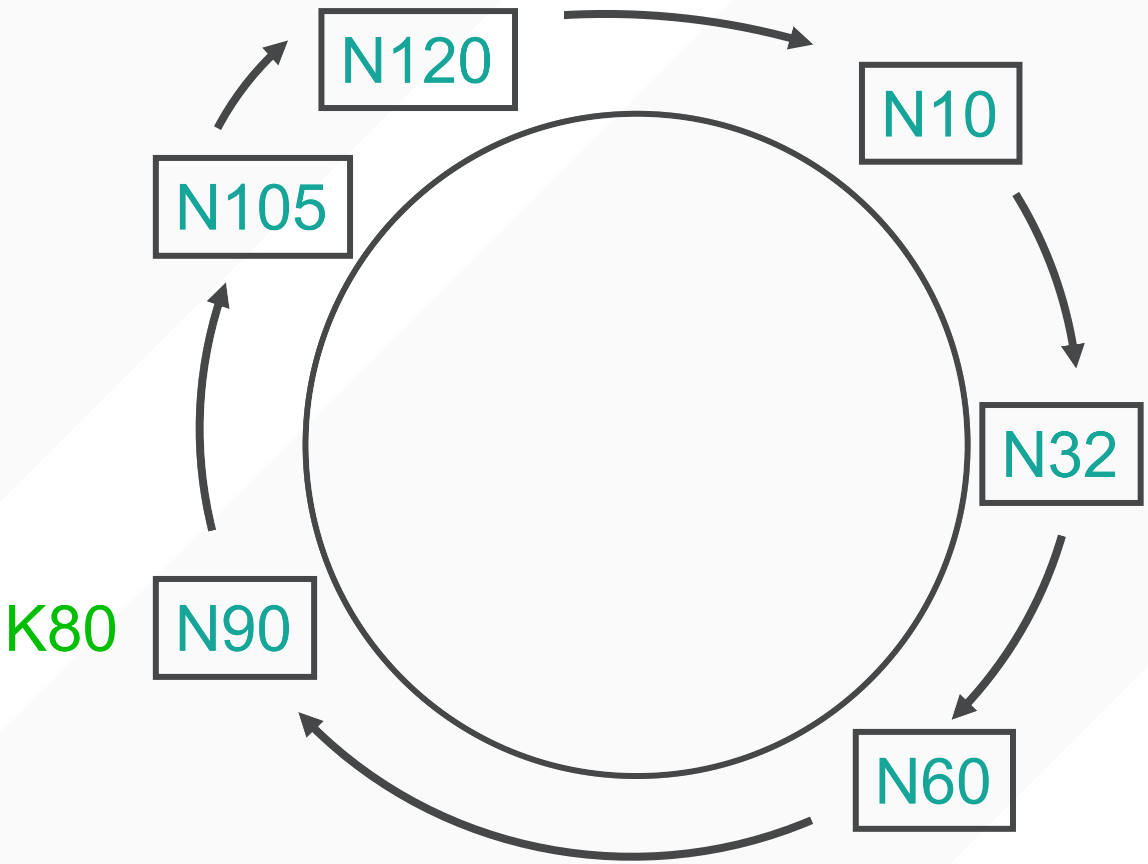
- **Key identifier** = SHA-1(key)
- **Node identifier** = SHA-1(IP address)
- SHA-1 distributes both uniformly
- ***How does Chord partition data?***
    - *i.e.*, map key IDs to node IDs

CONSISTENT HASHING [KARGER '97]

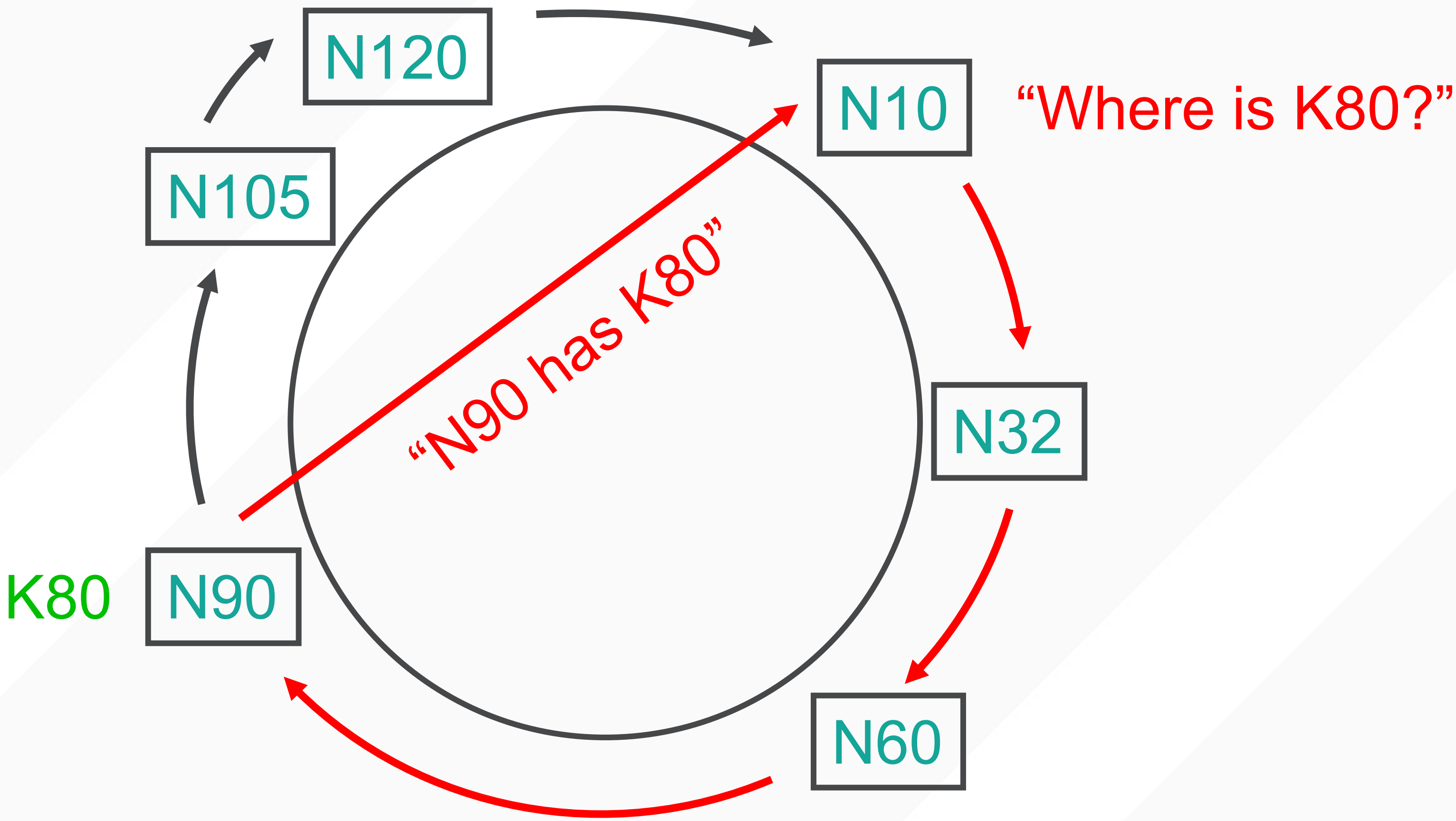


Key is stored at its successor: node with next-higher ID

CHORD: SUCCESSOR POINTERS



BASIC LOOKUP



SIMPLE LOOKUP ALGORITHM

```
Lookup(key-id)
   succ ← my successor
   if my-id < succ < key-id //next hop
      call Lookup(key-id) on succ
   else                    //done
    return succ
```

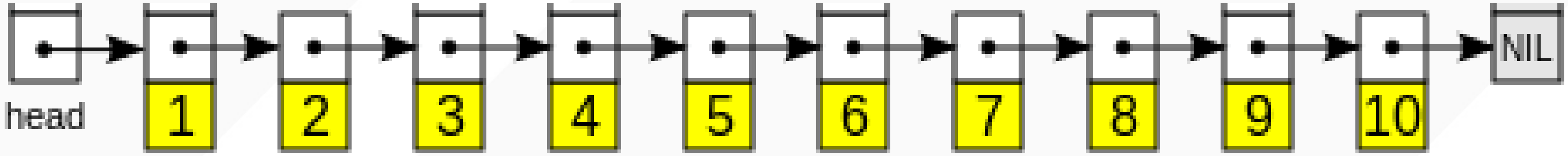
- **Correctness** depends only on **successors**

IMPROVING PERFORMANCE

- **Problem:** Forwarding through successor is slow
- Data structure is a linked list: O(n)
- **Idea:** Can we make it more like a binary search?
  - Need to be able to halve distance at each step

CHORD INTUITION

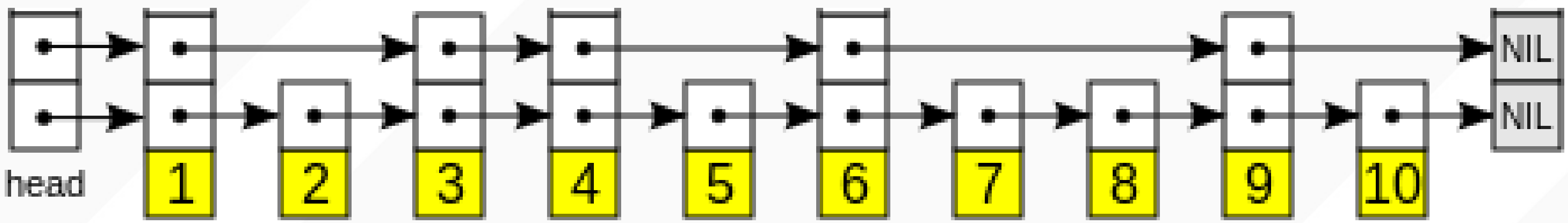
- Skip Lists (Pugh, 1989)
- Consider a linked list:



- Lookup time: O(n)

CHORD INTUITION

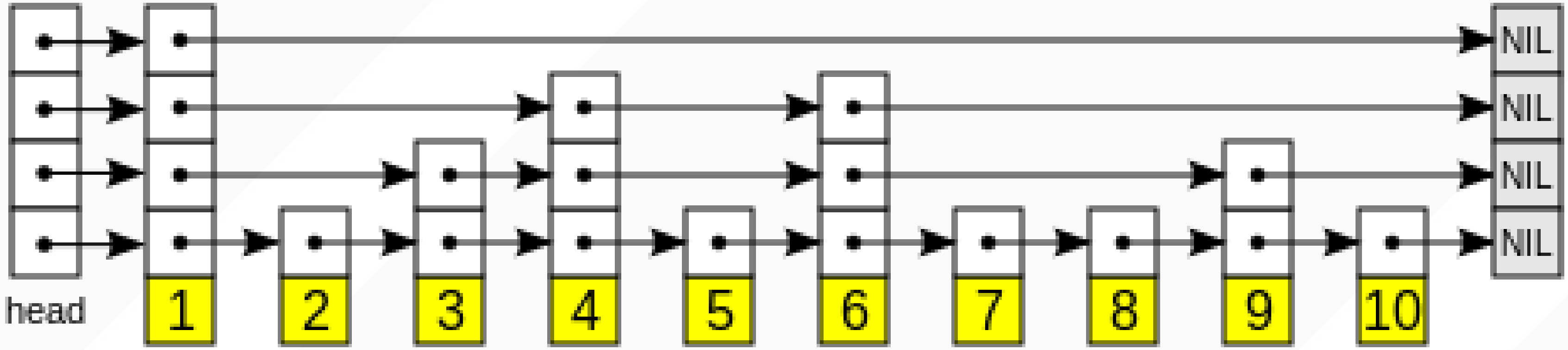
- Skip Lists (Pugh, 1989)
- Consider a linked list:



- Add 2nd row of pointers spaced further apart
  - Still O(n), but more efficient
  - Use 2nd row to get as close as possible without going over
  - Then last row to get to the desired element

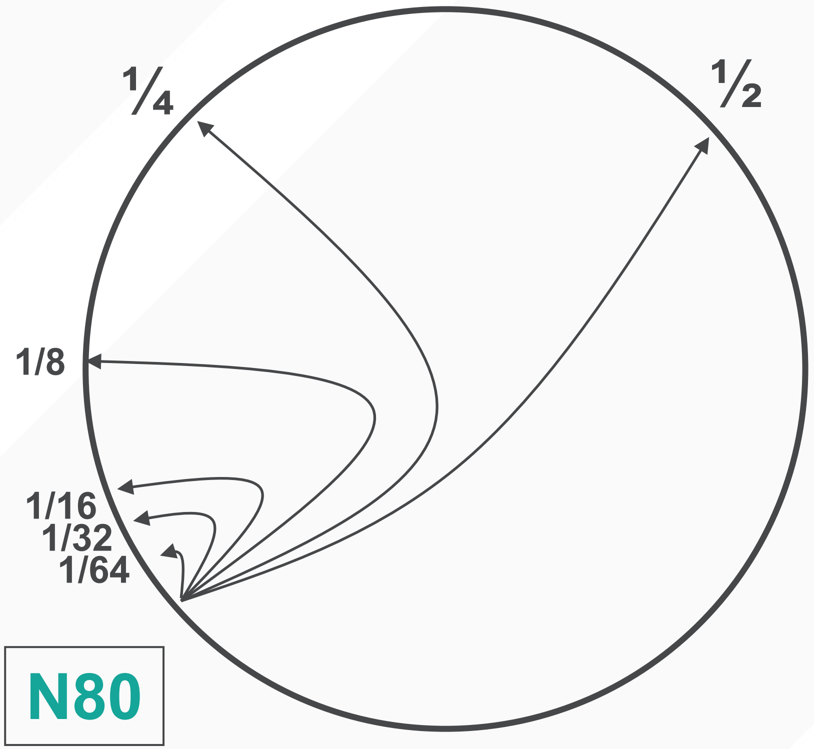
CHORD INTUTION

- Skip Lists (Pugh, 1989)
- Consider a linked list:

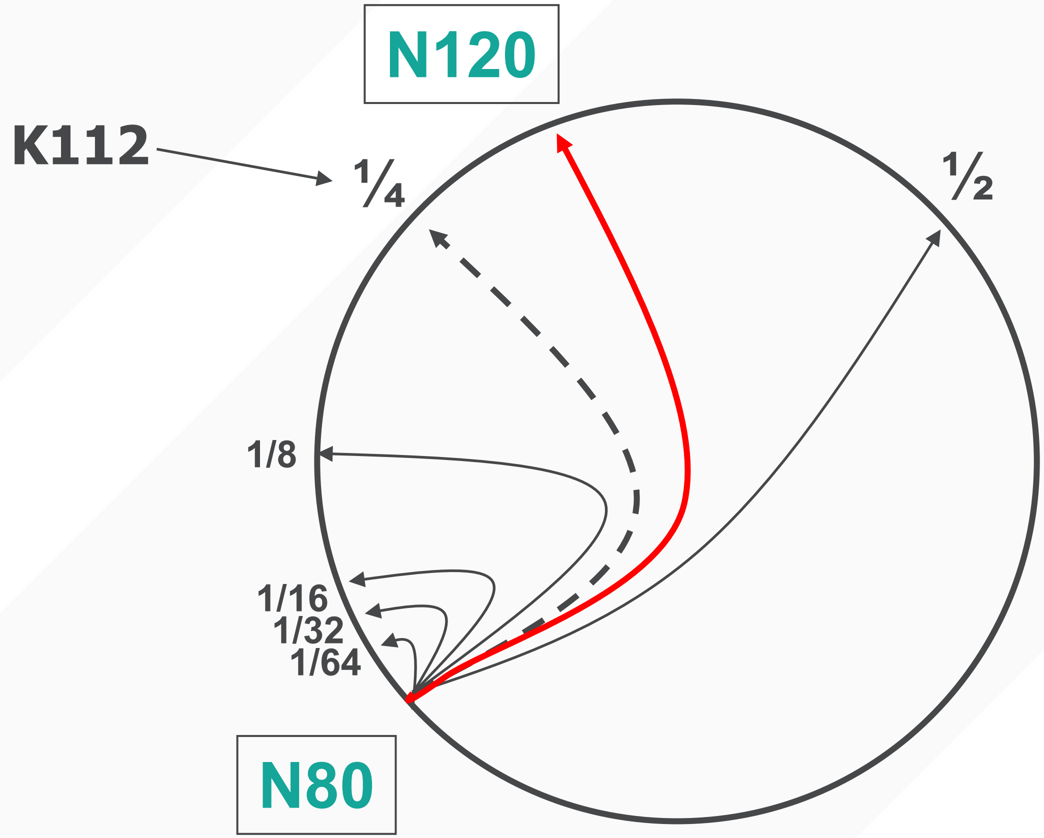


- Add log(N) rows
  - Get as close as possible on top row, then drop down a row, then drop down another row, until the bottom row
  - O(log N) lookup time

"FINGER TABLE" ALLOWS LOG N-TIME LOOKUPS



FINGER I POINTS TO SUCCESSOR OF $N+2^I$



IMPLICATION OF FINGER TABLES

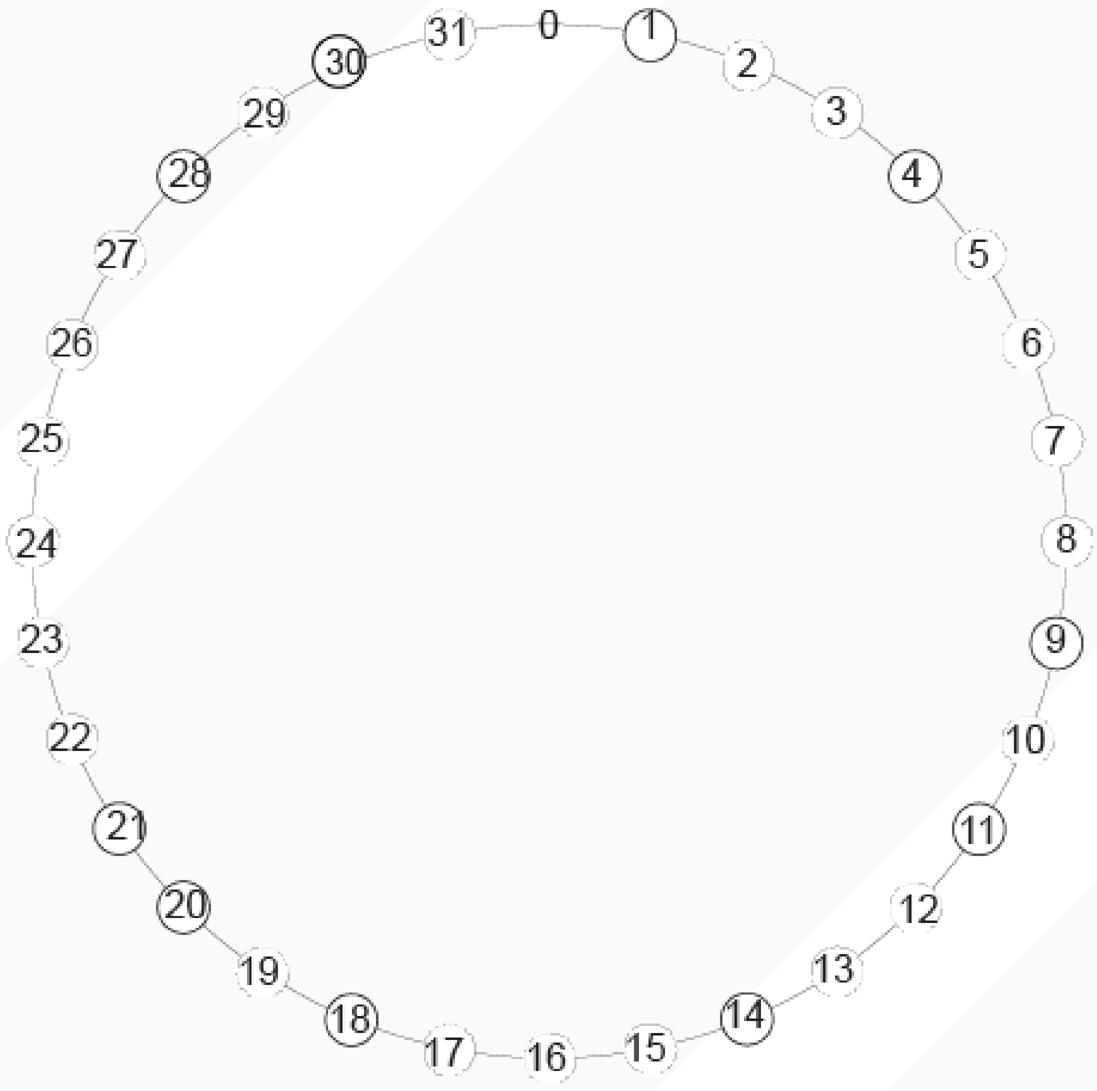
- A **binary lookup tree** rooted at every node
  - Threaded through other nodes' finger tables

- This is **better** than simply arranging the 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

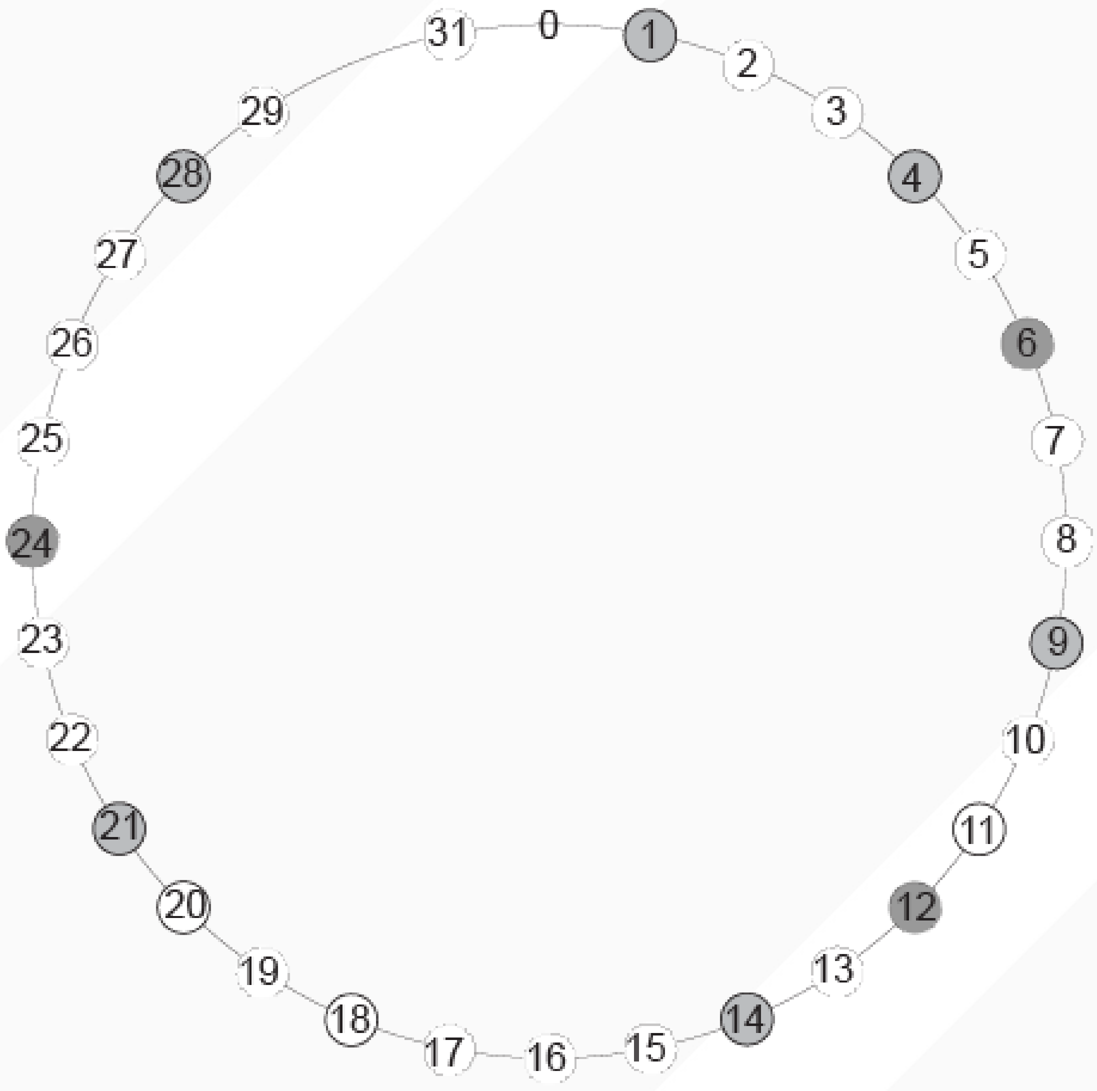
LOOKUP WITH FINGER TABLE

```
Lookup(key-id)
  look in local finger table for
      highest n: my-id < n < key-id
  if n exists
      call Lookup(key-id) on node n //next hop
  else
      return my successor //done
```

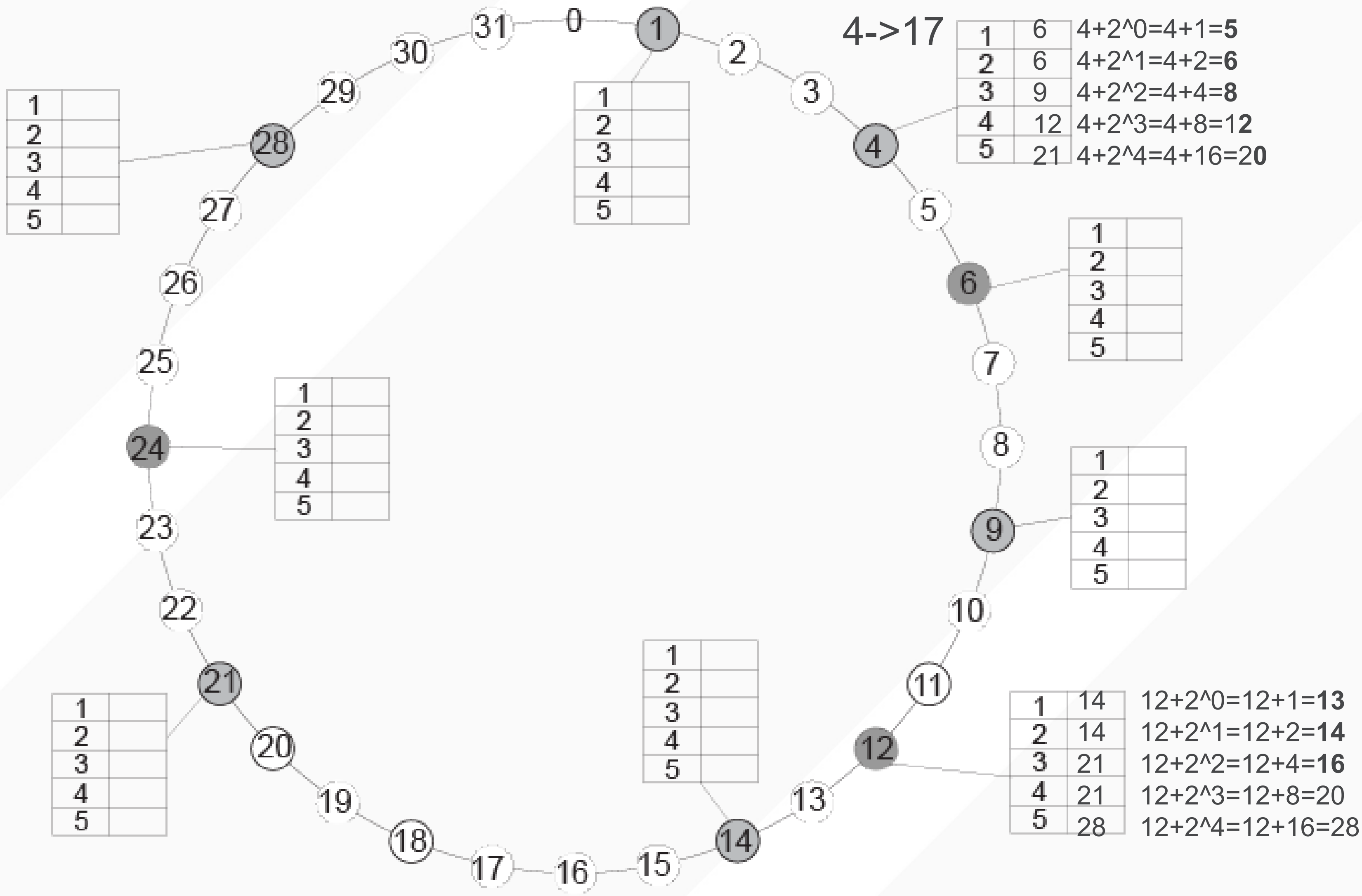
THE CHORD RING (2^5=32)



CHORD RING WITH SERVERS {1,4,6,9,12,14,21,24,28}



ADDING FINGER TABLES



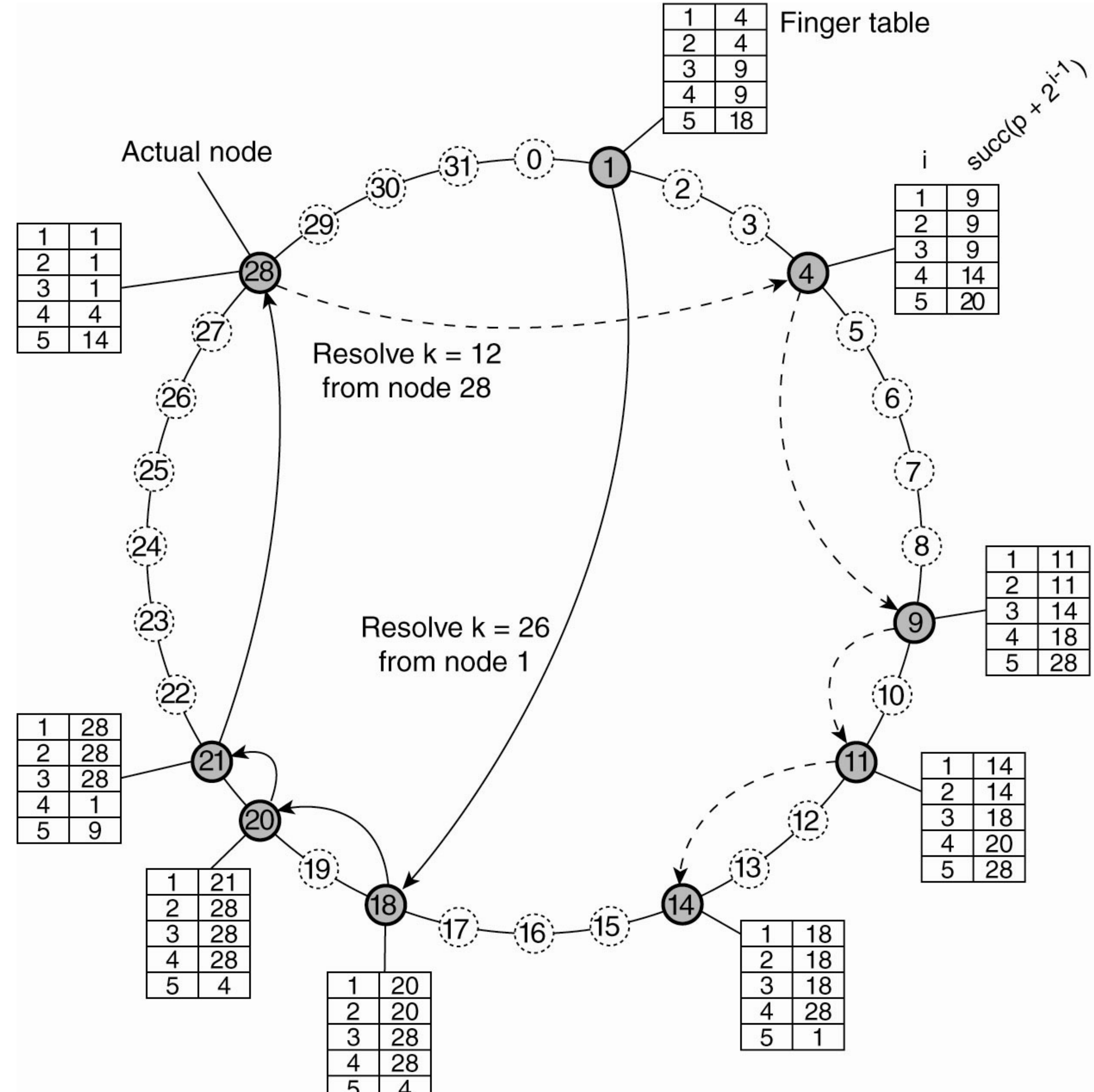
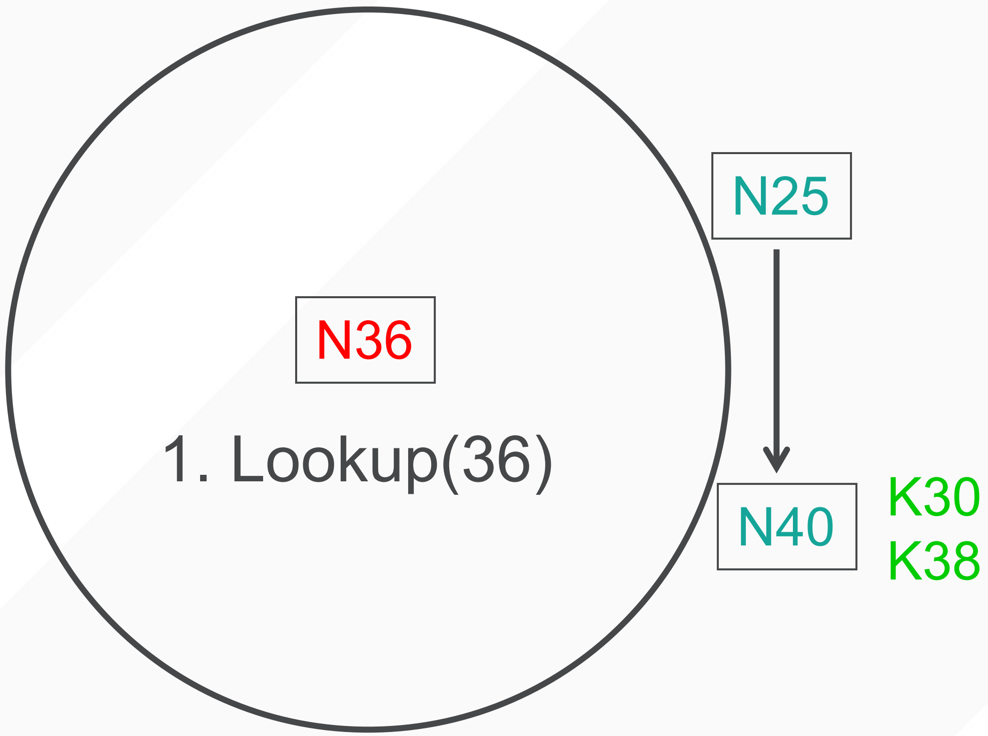


Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.

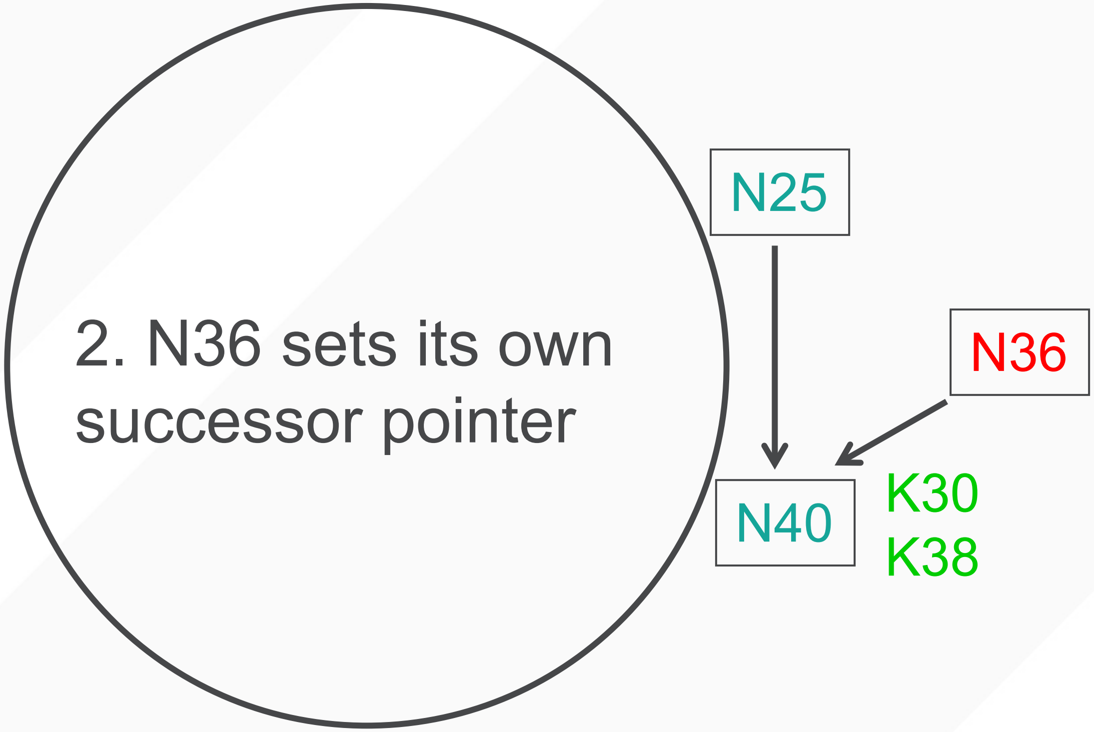
AN ASIDE: IS LOG(N) FAST OR SLOW?

- For a million nodes, it's 20 hops
- If each hop takes 50 milliseconds, lookups take **a second**
- If each hop has 10% chance of failure, it's a couple of timeouts
- So in practice log(n) is better than O(n) but **not great**

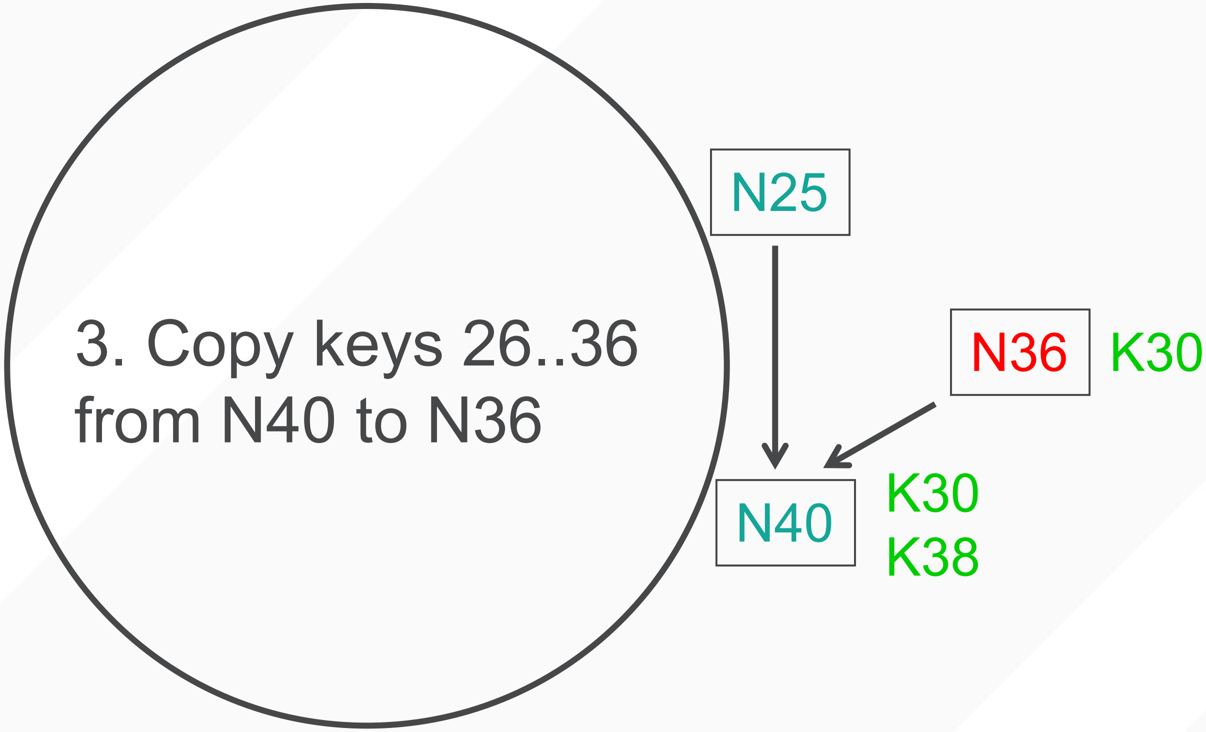
JOINING: LINKED LIST INSERT



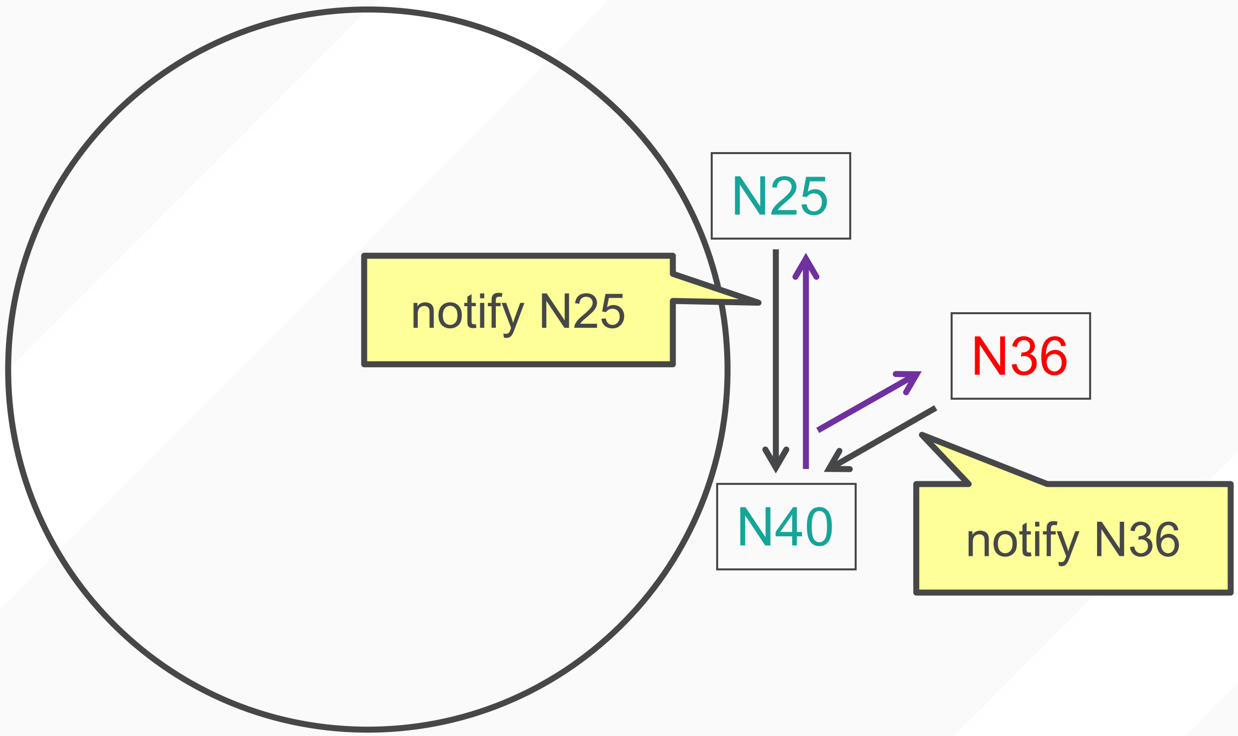
JOIN (2)



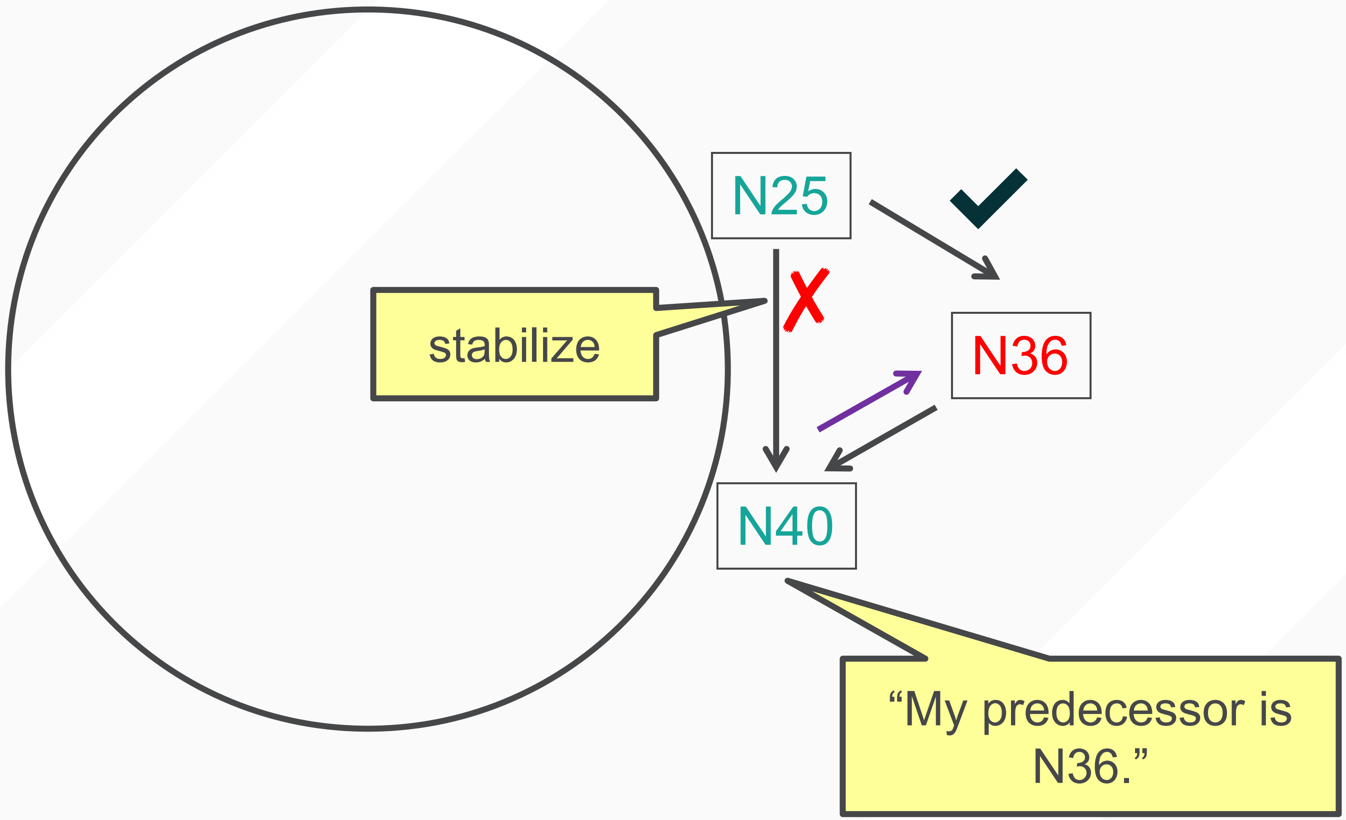
JOIN (3)



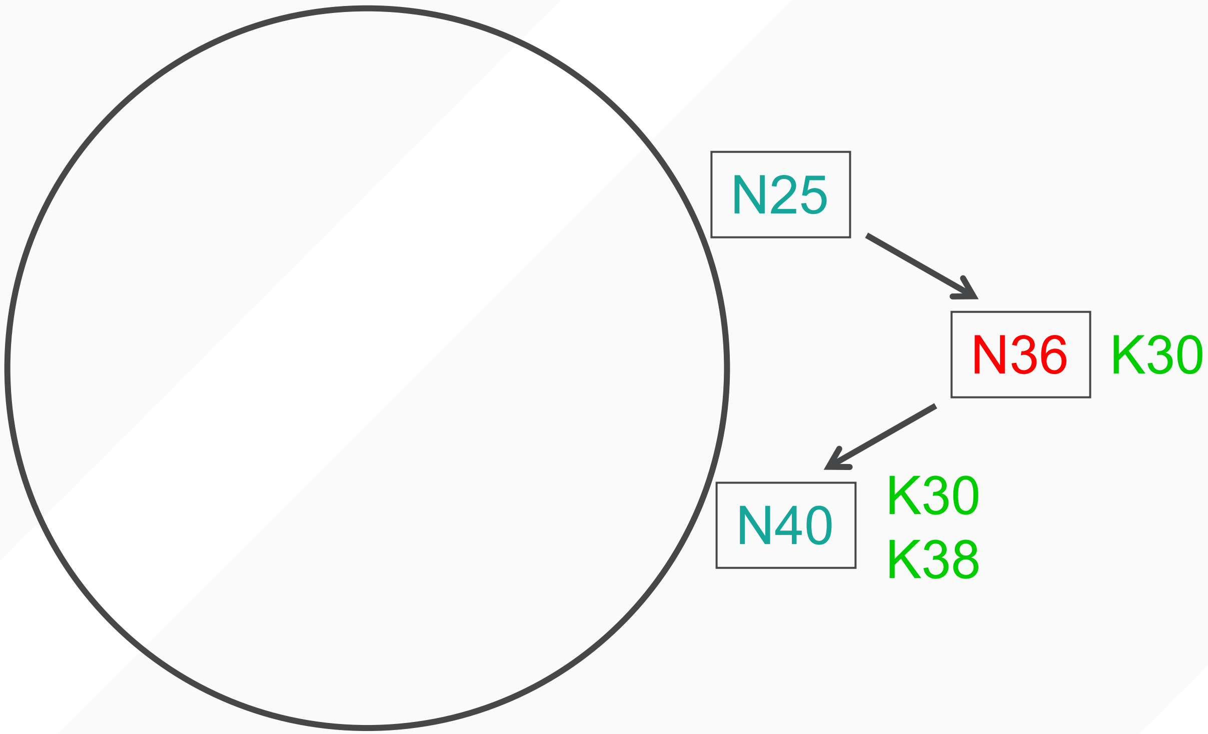
NOTIFY MESSAGES MAINTAIN PREDECESSORS



STABILIZE MESSAGE FIXES SUCCESSOR

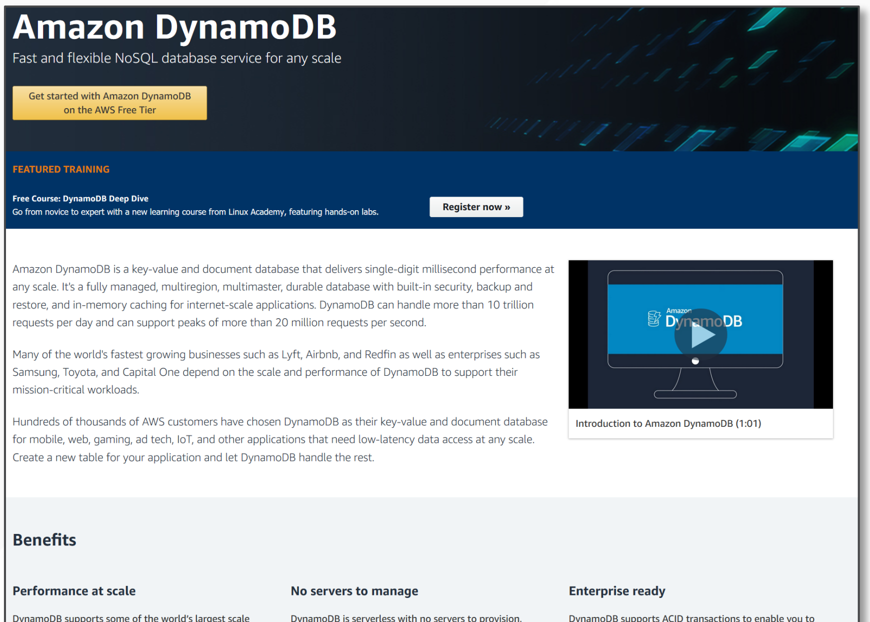


JOINING: SUMMARY



- Predecessor pointer allows link to new node
- Update finger pointers in the background
- Correct successors produce correct lookups

DYNAMODB



- On Canvas, under module 4
  - Read the SOSP 2007 paper linked off module 4
  - Watch the 30 minute SOSP 2007 talk
- In project 4, you will build a simplified version of DynamoDB
  - Specifically the replication code
  - You will not be implementing the consistent hashing based preference list

WHAT DHTS GOT RIGHT

- **Consistent hashing**
  - Elegant way to divide a workload across machines
  - Very useful in clusters: actively used today in Amazon Dynamo and other systems
- **Replication** for high availability, efficient recovery after node failure
- **Incremental scalability:** “add nodes, capacity increases”
- **Self-management:** minimal configuration
- **Unique trait:** no single server to shut down/monitor

UC San Diego