6.263 Data Communication Networks

Lecture 11

Dina Katabi

dk@mit.edu
www.ana.lcs.mit.edu/dina

Outline

- Content Distribution Networks
- Multicast

Next: Content Distribution Networks and Content-Based Routing

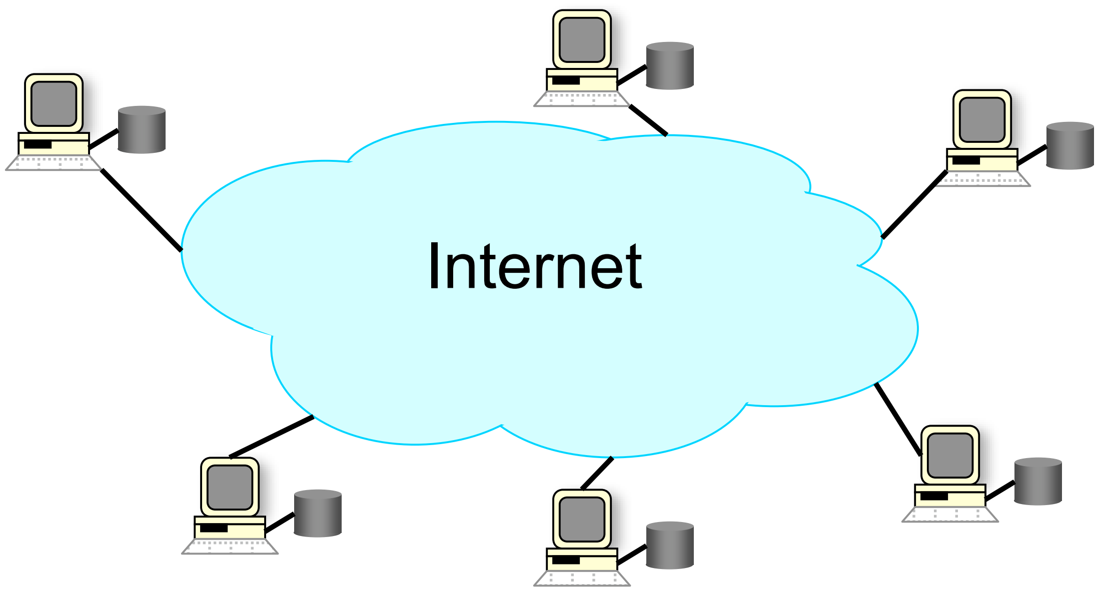
- Usually, a network user is not interested in accessing a particular machine but rather a particular service or data object
  - For example, find me the closest printer
  - Find me song *x*
- Should we route based on content?
  - Instead of advertising the destination IP address, I advertise its <service, service attributes>
  - Scalability is always a challenge

Example of Overlay with Content Routing: P2P File Sharing

- Problem:
  - Set of data items (files or objects) that are stored at various machines in the network.
  - Find a me particular data item
- Examples:
  - Music sharing using Kaza, Napster, or Gnutella

How Did it Start?

- A killer application: Naptser
  - Free music over the Internet
- Key idea: share the content, storage *and* bandwidth of individual (home) users

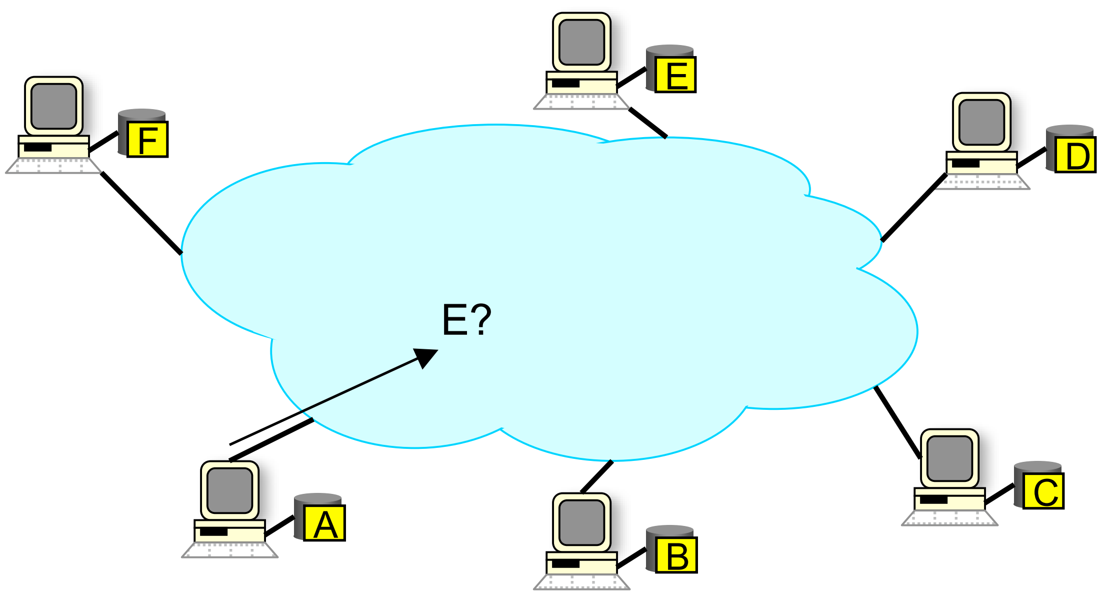


Peer-to-Peer Model

- Each user stores a subset of files
- Each user has access (can download) files from all users in the system
- Why is it called P2P?
  - Traditionally you run only a client and download files only from a server
    - Example: Web, FTP
  - Here every machine is both a client and a server

Main Challenge

- Find where a particular file is stored



Other Challenges

- Scale: up to hundred of thousands or millions of machines
- Dynamicity: machines can come and go any time

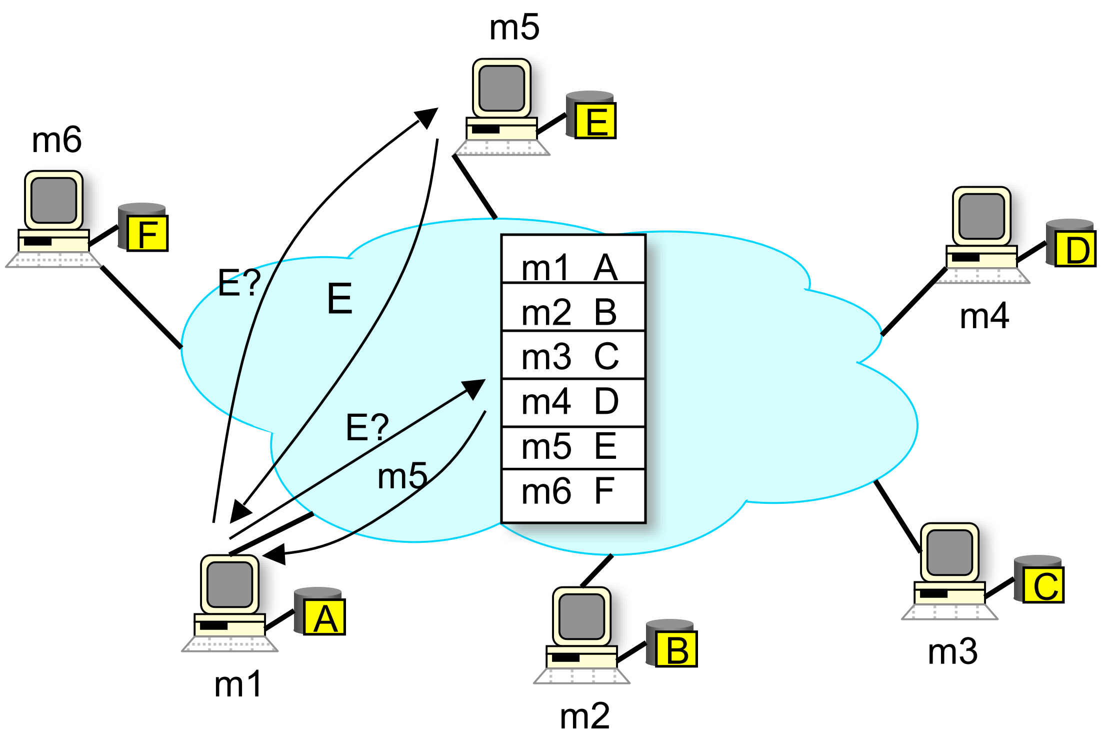
3 Approaches to P2P File Sharing

- Centralized
  - Napster
- Flooding-Based
  - Gnutella, Kazaa
- Routing-Based
  - Chord, CAN, Pastry, ...

Centralized: Napster

- Simple centralized scheme → the Napster server maintains an index of all files
- How to find a file:
  - On startup, client contacts central server and reports list of files
  - Query the index system → return a machine that stores the required file
    - Ideally this is the closest/least-loaded machine
  - Fetch the file directly from peer

Napster: Example



Centralized: Napster

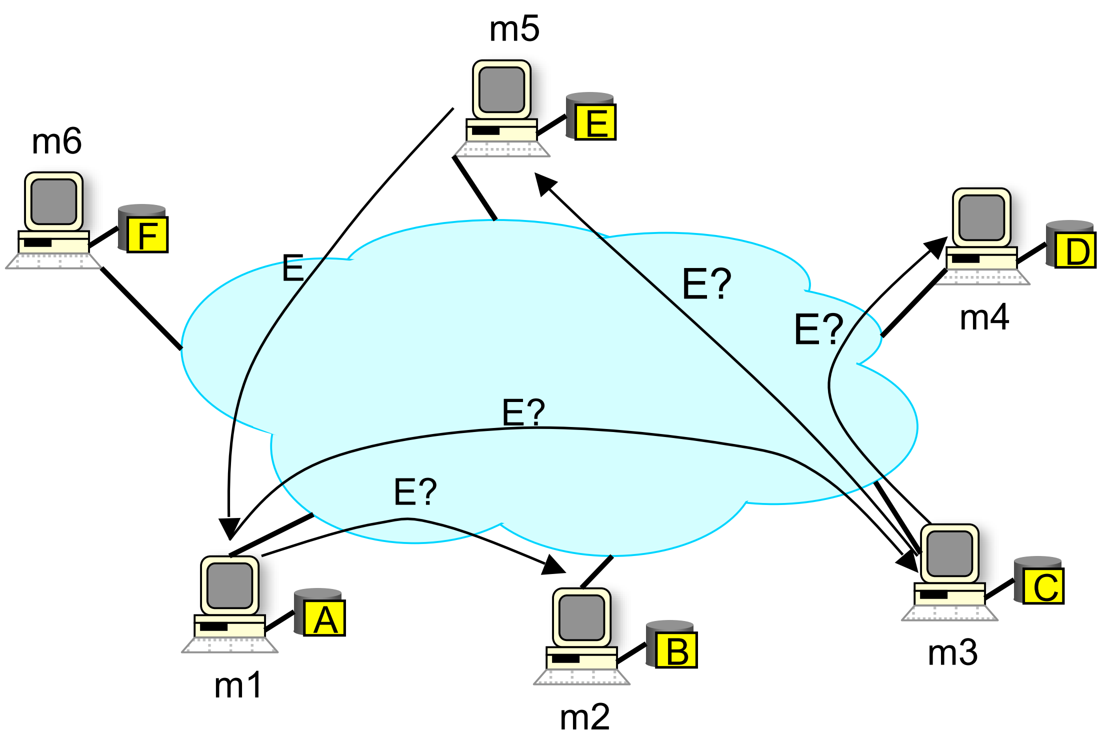
- Advantages:
  - Simple
  - Easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness
  - Scalability
  - Easy to sue!

Flooding: Gnutella

- On startup, client contacts any servent (<u>**serv**</u>er + cli<u>**ent**</u>) in network
  - Servent interconnection used to forward control (queries, hits, etc)
- Idea: broadcast the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively forward the request
  - Eventually a machine that has the file receives the request, and it sends back the answer
  - Transfers are done with HTTP between peers

Gnutella: Example

- Assume: m1's neighbors are m2 and m3; m3's neighbors are m4 and m5;…



Flooding: Gnutella

- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  - Especially hard on slow clients
    - At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?
    - Modem users were effectively cut off!

Flooding: FastTrack (aka Kazaa)

- Modifies the Gnutella protocol into two-level hierarchy
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits ☹

Problems with the Previous Solutions

- Do not scale!
  - Either flood messages or require a single database

Problem Abstraction

- Input:
  - Large set of keys
  - a number of nodes that can be connected in any way (using an overlay)
- Output:
  - Store the keys at the nodes and create the links in an overlay s.t.
    - Minimize the time it takes to find a key
    - Minimize the amount of information the nodes need to store
    - Fairly distribute the load among the nodes (storage load and response to queries)
    - Minimize the work needed to maintain the system when nodes join or leave
- We are looking for a good solution, not necessarily the optimal

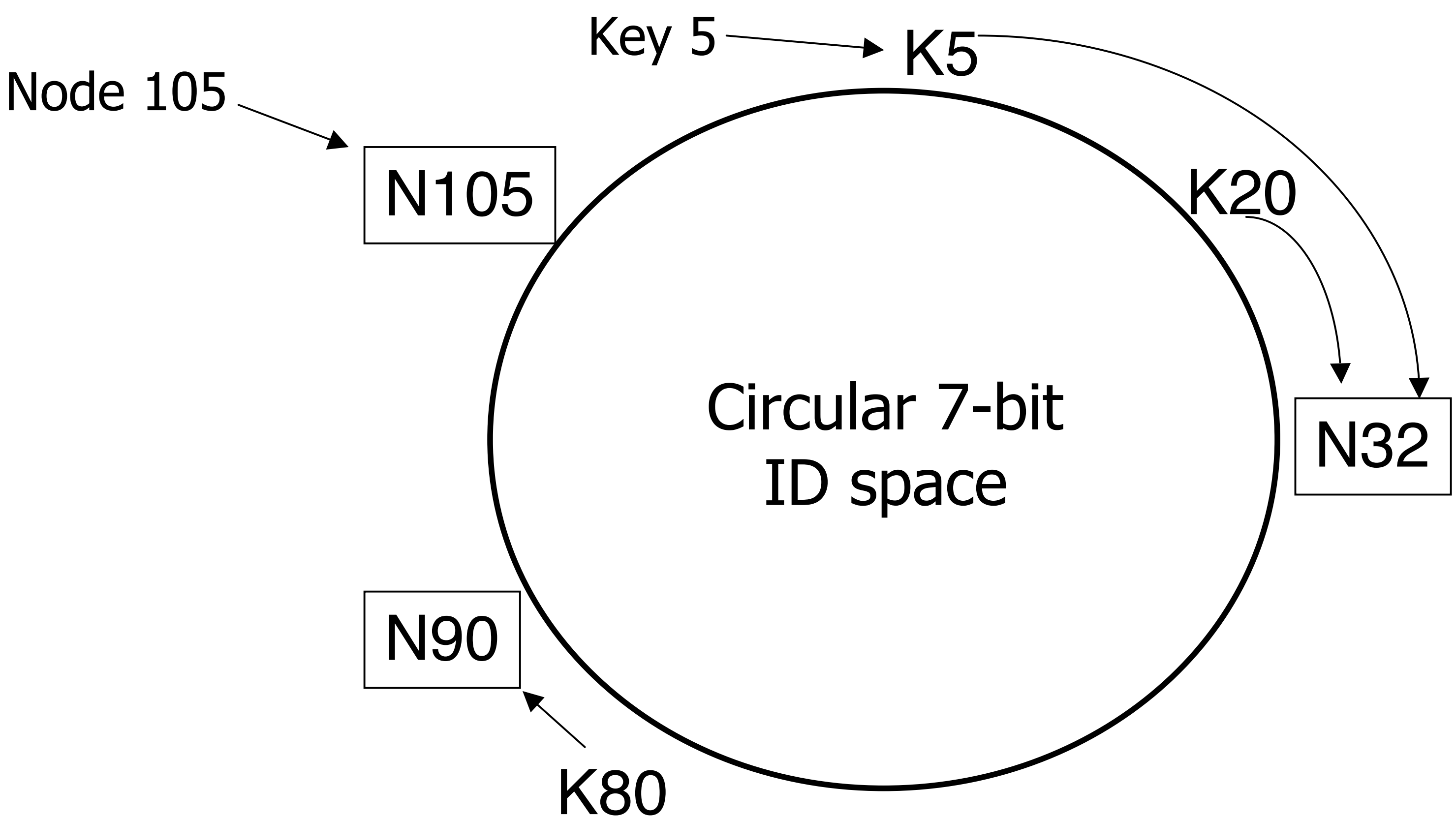
Attempts at Finding a Solution

- First hash the keys so that the space of keys after hashing is balanced
- Each node is responsible for a range of keys
- How does a node find the node which stores key_i?
  - Each node stores a list of all nodes and their ranges
    - unscalable
  - A distributed tree
    - But root node will have to receive all queries, which causes congestion
- Distributed Hash Tables (DHT): Chord, CAN, ..

Chord

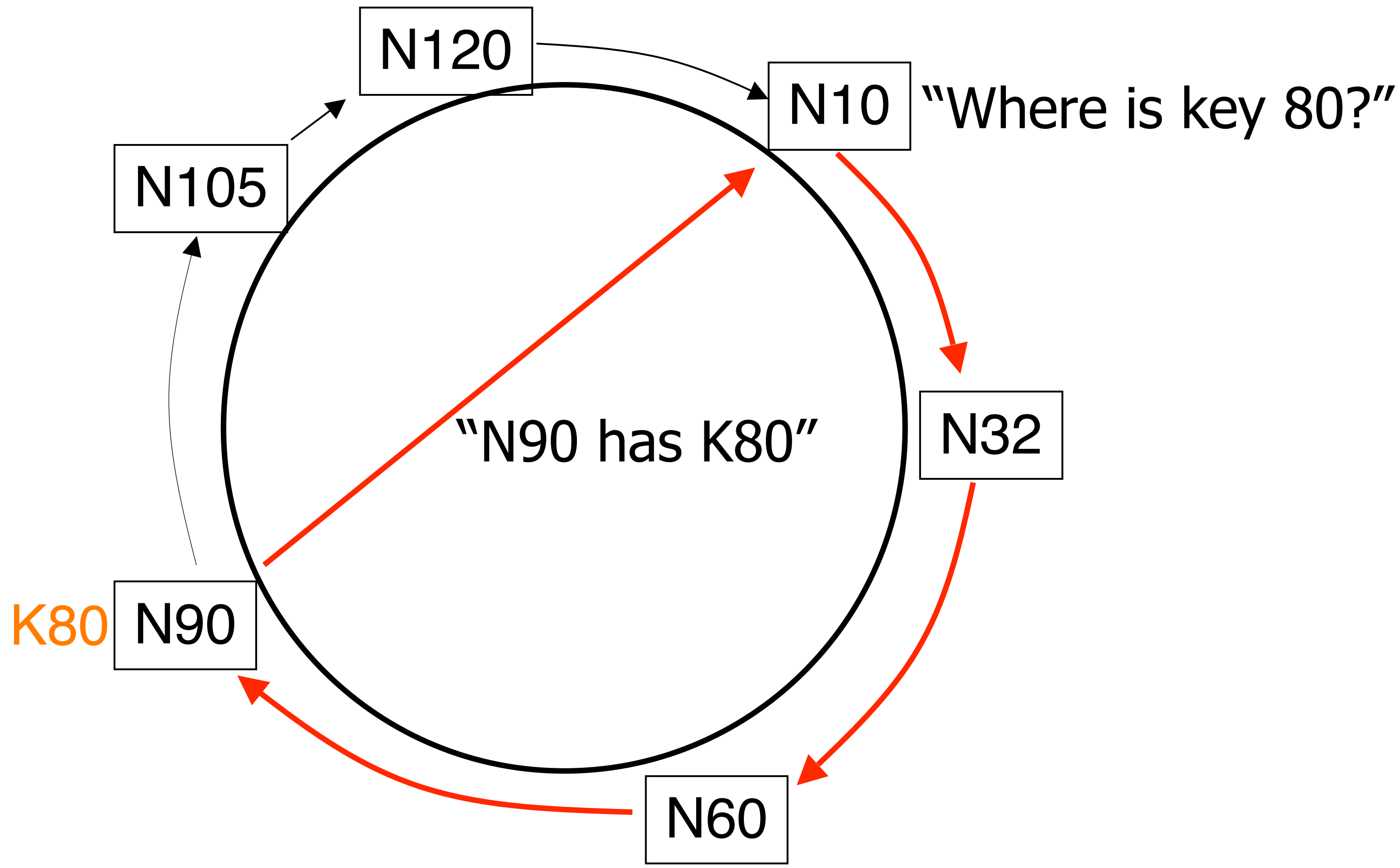
- Associate to each node and item (e.g. file) a unique *id* in an *uni*-dimensional space
- Goals
  - Scales to hundreds of thousands of nodes
  - Handles rapid arrival and failure of nodes
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Based on : Consistent Hashing [Karger 97]



A key is stored at its successor: node with next higher ID

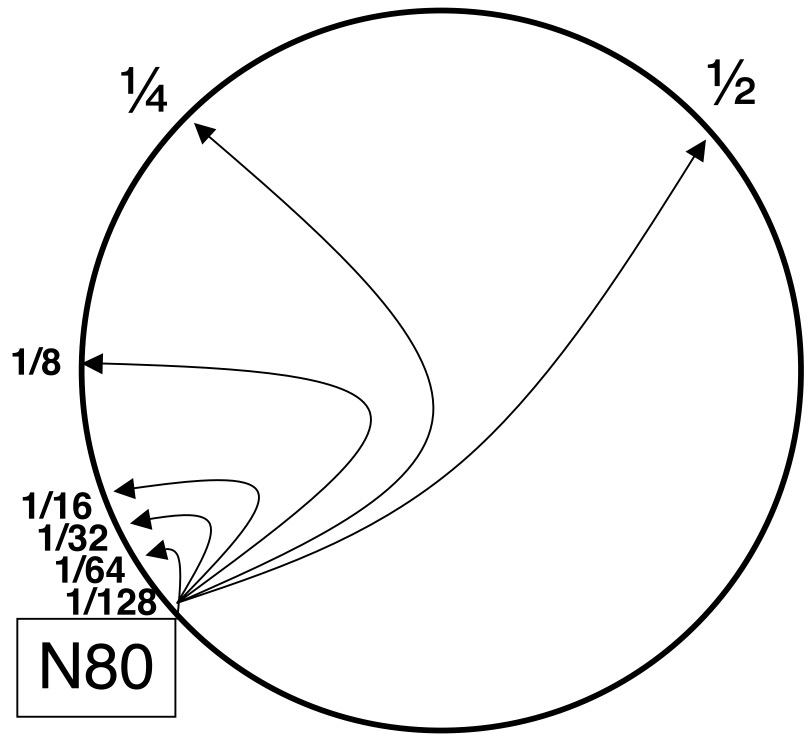
Routing: Chord Basic Lookup



Each node stores a pointer to its successor;

In the worst case, lookups take N overlay hops

Routing: "Finger table" - Faster Lookups



Each Node Stores log(N) pointers;

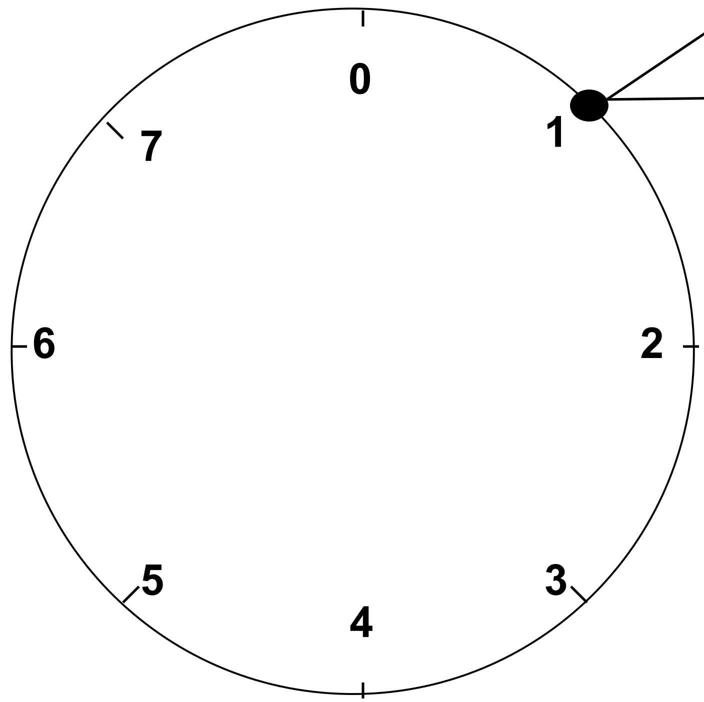
In worst case, lookups take log (N) overlay hops

Data Structure

- Assume identifier space is $0..2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the successor node of *id*

Chord Example

- Assume an identifier space 0..8
- Node 1 joins→all entries in its finger table are initialized to itself

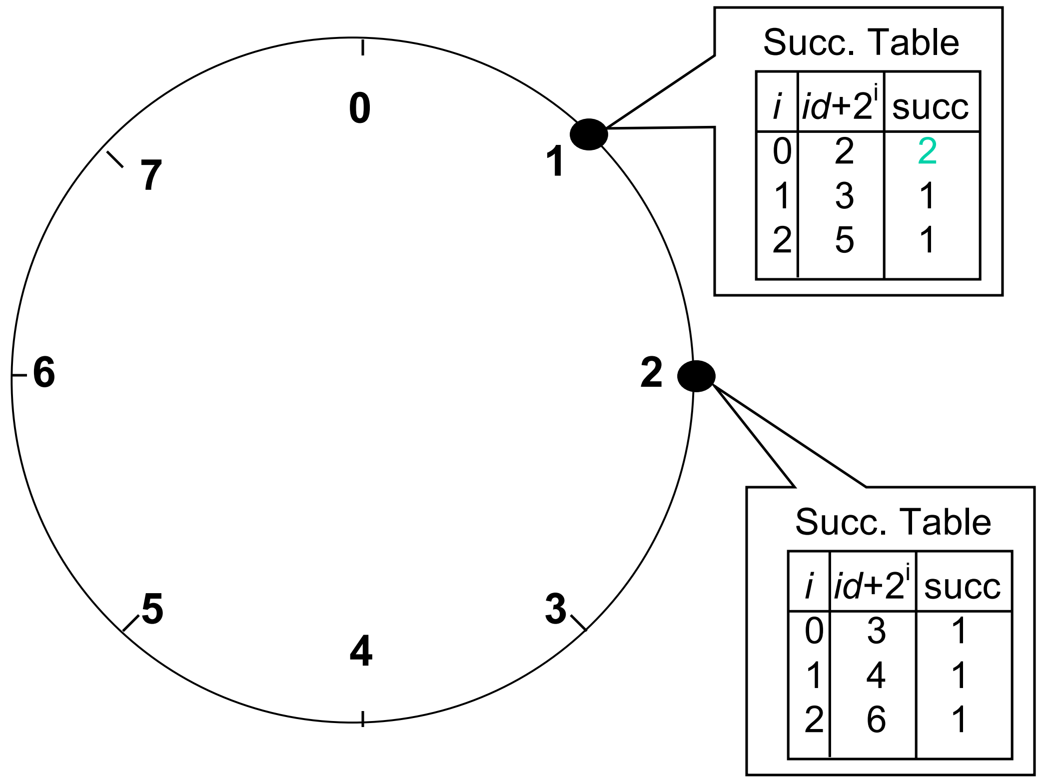


Succ. Table

i	*id*+2^i	succ
0	2	1
1	3	1
2	5	1

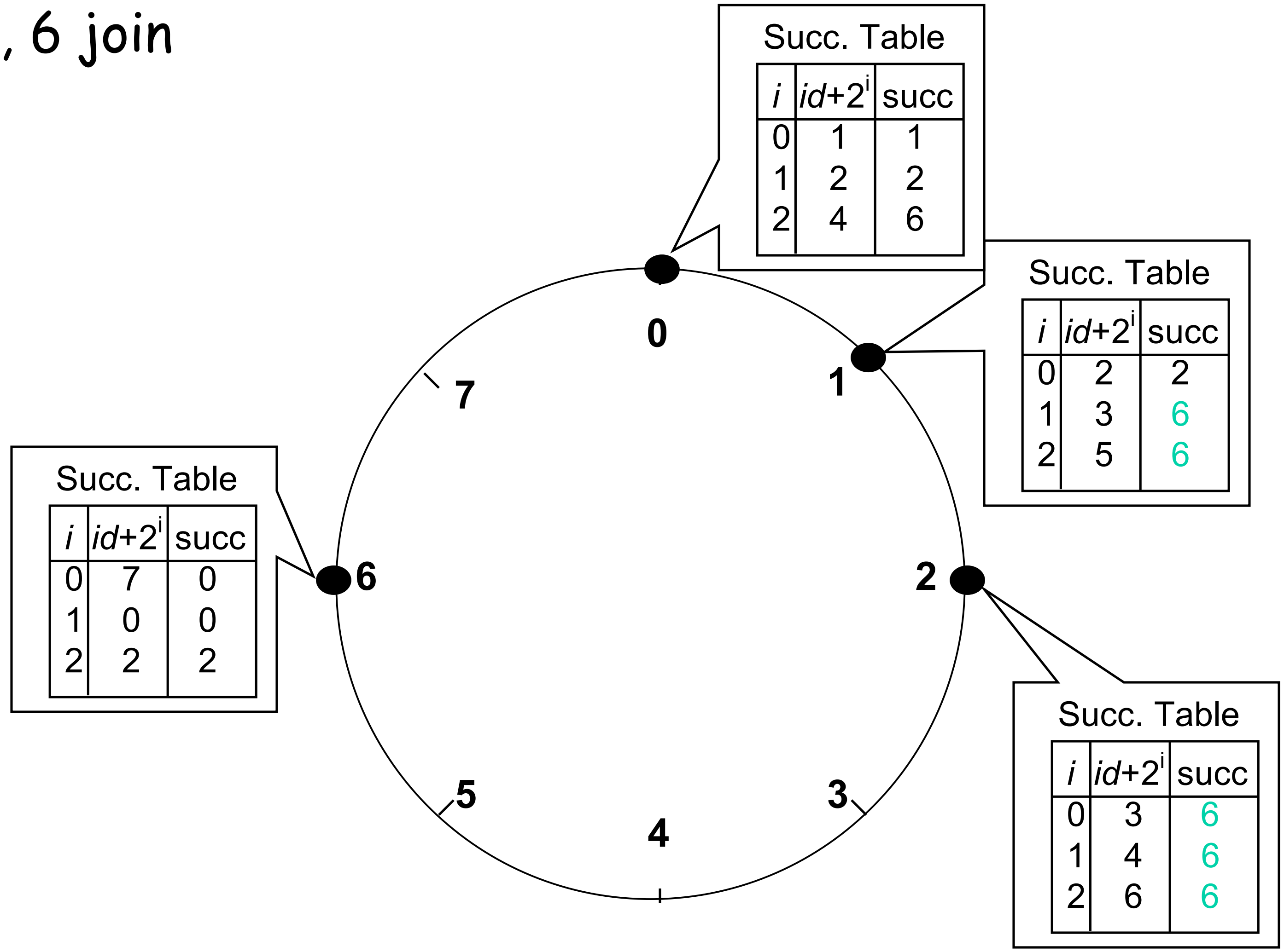
Chord Example

- Node 2 joins



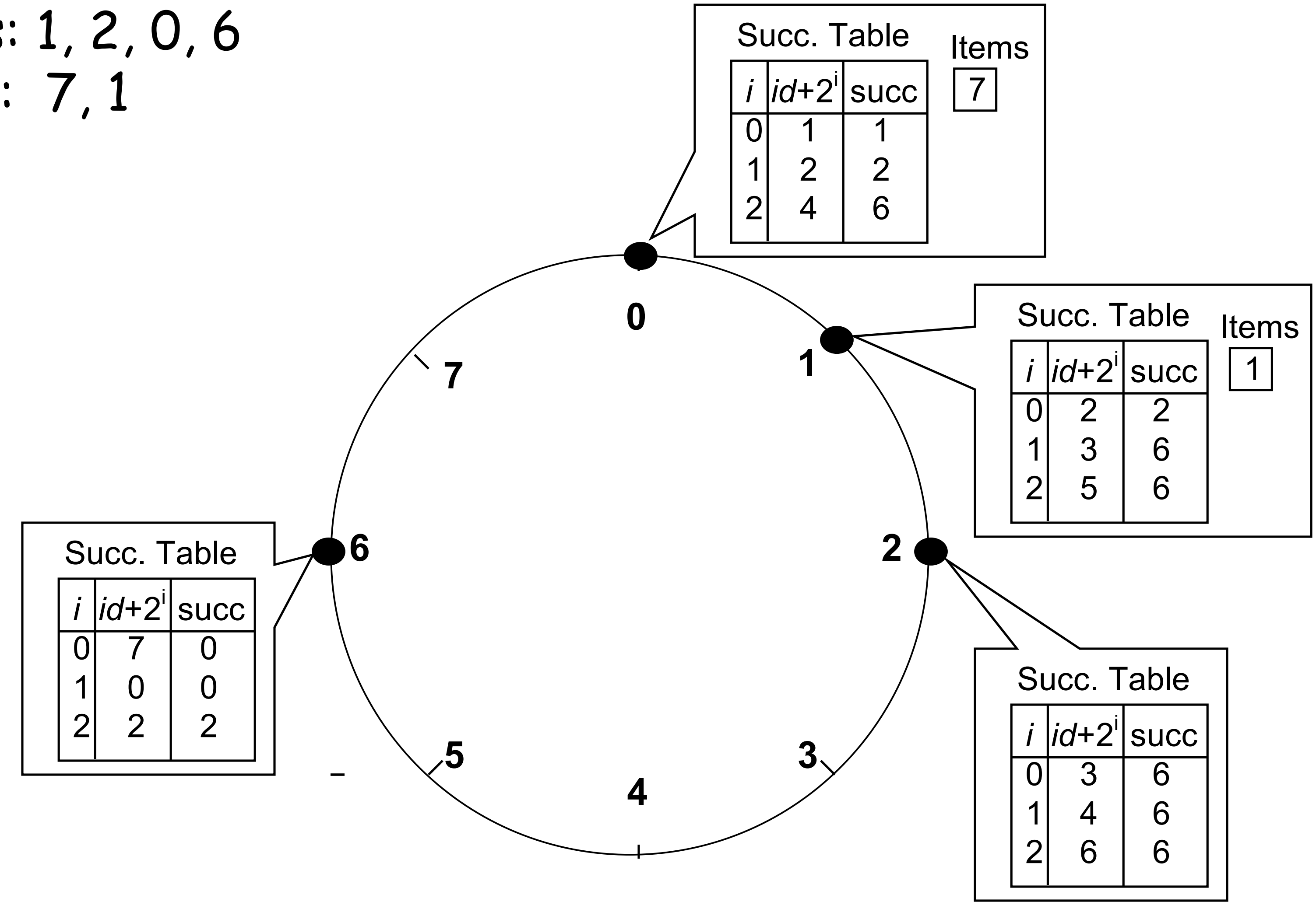
Chord Example

❖ Nodes 0, 6 join



Chord Examples

- Nodes: 1, 2, 0, 6
- Items: 7, 1



Node 0 — Succ. Table (Items: 7)

i	$id+2^i$	succ
0	1	1
1	2	2
2	4	6

Node 1 — Succ. Table (Items: 1)

i	$id+2^i$	succ
0	2	2
1	3	6
2	5	6

Node 2 — Succ. Table

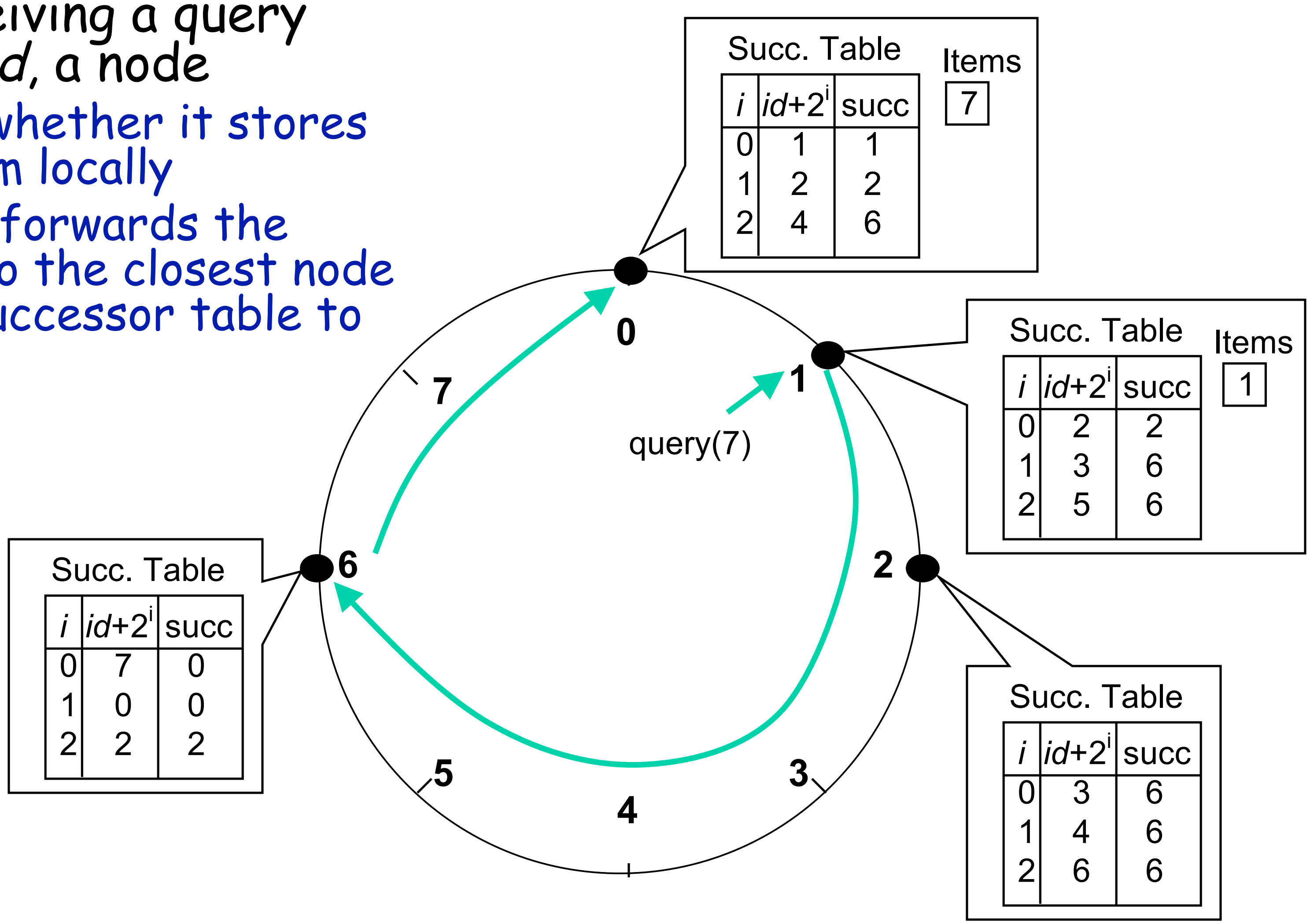
i	$id+2^i$	succ
0	3	6
1	4	6
2	6	6

Node 6 — Succ. Table

i	$id+2^i$	succ
0	7	0
1	0	0
2	2	2

Query

- Upon receiving a query for item *id*, a node
  - Check whether it stores the item locally
  - If not, forwards the query to the closest node in its successor table to *id*



Node Joining

- Node n joins the system:
  - node picks a random identifier, id
  - node performs n' = lookup(id)
  - node->successor = n'

Discussion

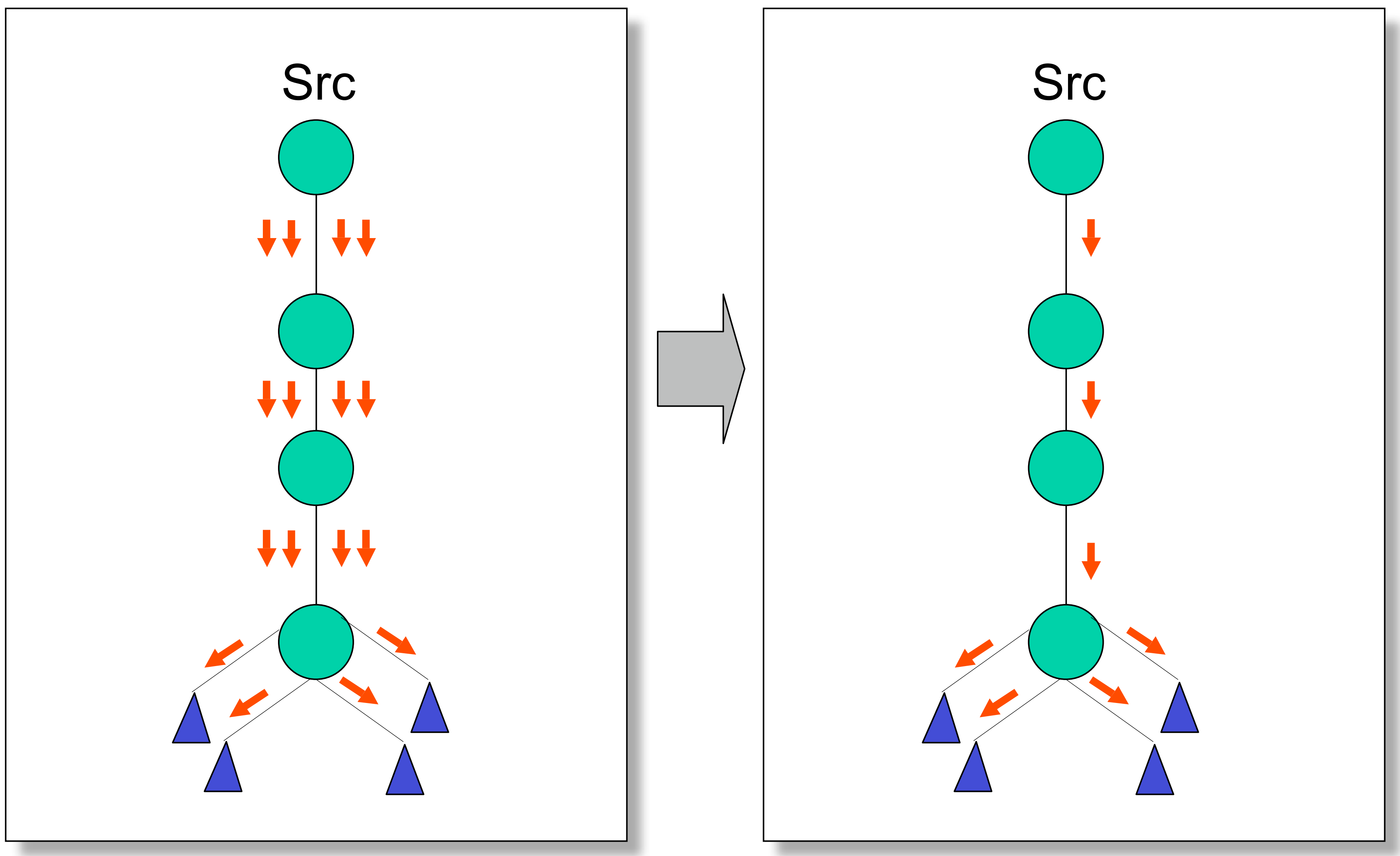
- Advantage
  - log(N) overlay hops and log(N) storage at each node
- Problems: Each hop in a overlay-based P2P network can be expensive
  - No correlation between neighbors and their location
  - A query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!

Multicast

Multicast

- Unicast is one-to-one
- Multicast is one-to-many, or many-to-many
- Applications of Multicast
  - Single sender to many receivers
    - Online TV
    - Publish-subscribe
    - Web-cache updates
  - Many senders to many receivers
    - Interactive learning
    - Teleconferencing

Why do we need multicast routing?



IP Multicast

- Multicast Addressing: we need to identify the intended receivers of a multicast, which we call the multicast group
  - Each group has an ID
    - an IP address with a multicast prefix
  - Note that the group is location-independent (i.e., an IP multicast address is a name not an address)
- Multicast Routing: allows routers to learn how to deliver multicast packets and where to duplicate them

IP Multicast Semantics

- Analogy:
  - Each multicast address is like a radio frequency, on which anyone can transmit, and to which anyone can tune-in.
- Sender sends to the multicast IP address
- Receivers can join or leave the multicast group at will
- Routers deliver packets from sender to receivers
- *Why this model?*