Structured and Unstructured Peer-to-Peer Computing

Peer-to-Peer Computing

- Quickly grown in popularity:
  - Dozens or hundreds of file sharing applications
  - In 2004:
    - 35 million adults used P2P networks – 29% of all Internet users in USA
    - 35% of Internet traffic is from BitTorrent
  - Upset the music industry, drawn college students, web developers, recording artists and universities into court
- But P2P is not new and is probably here to stay
- P2P is simply the next iteration of scalable distributed systems

What is P2P?

- Peers serve as both clients and servers
- Eliminates or minimizes the need for a centralized node

- P2P has a rich history
- Original Internet was a p2p system:
  - The original ARPANET connected UCLA, Stanford Research Institute, UCSB, and Univ. of Utah
  - No routing infrastructure, just connected by phone lines
  - Computers also served as routers

P2P Systems

- File Sharing
  - Napster
  - Gnutella
  - BitTorrent

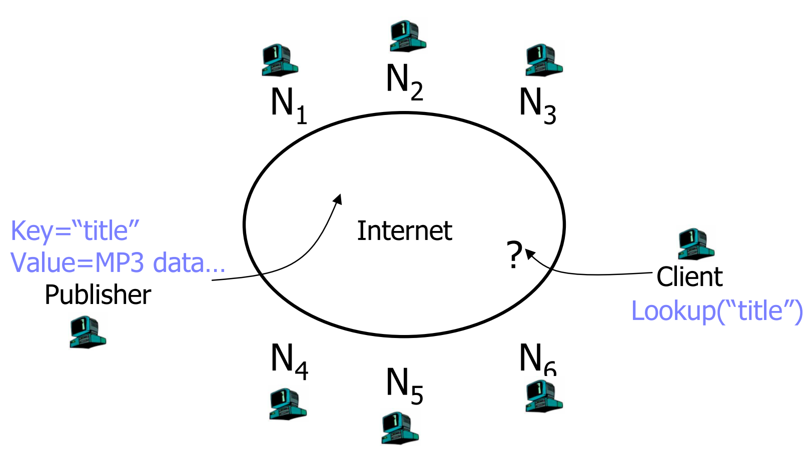
- Research systems
  - Distributed Hash Tables
  - Content distribution networks

- Collaborative computing:
  - SETI@Home project
  - Human genome mapping
  - Intel NetBatch: 10,000 computers in 25 worldwide sites for simulations, saved about 500million

Topic Outline

- Unstructured paradigm for p2p computing
  - Centralized Database: Napster
  - Query Flooding: Gnutella
  - Intelligent Query Flooding: Freenet
  - Swarming exchange: BitTorrent
- Structured paradigm for p2p computing
  - Distributed Hash Tables

The Lookup Problem



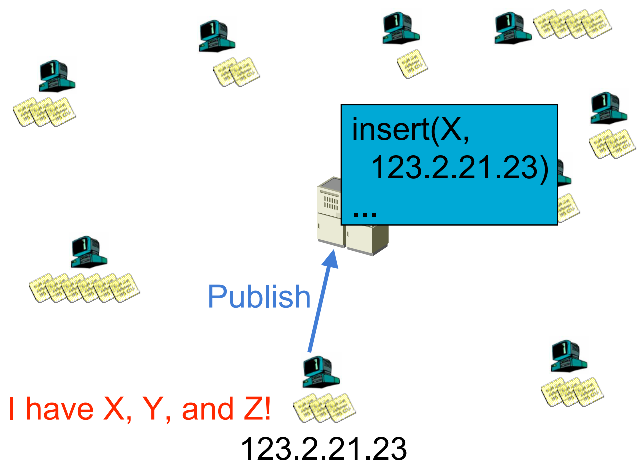
The Lookup Problem

- Common Primitives:
  - **Join**: how does a peer begin participating?
  - **Publish**: how does a peer advertise a file?
  - **Search**: how does a peer find a file?
  - **Fetch**: how does a peer retrieve a file?

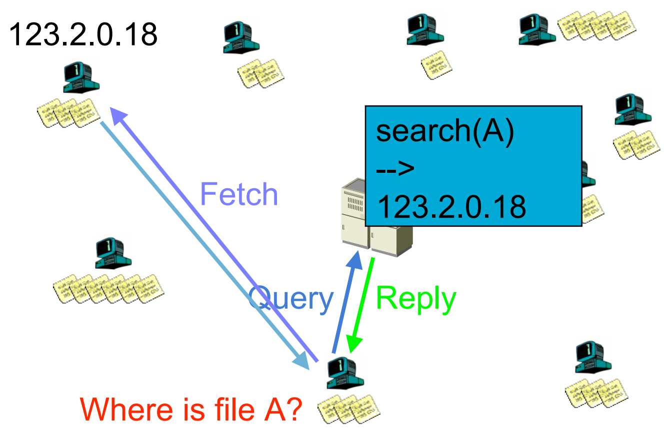
Centralized Database: Napster

- Shawn Fanning a freshman from NorthEastern develops Napster in May 1999
- Uses a centralized database
- RIAA sues Napster in December 1999
- Napster peaked at 1.5 million simultaneous users and 2.79 billion files in Feb 2001
- In July 2001, Napster is shut down

Napster: Publish



Napster: Search



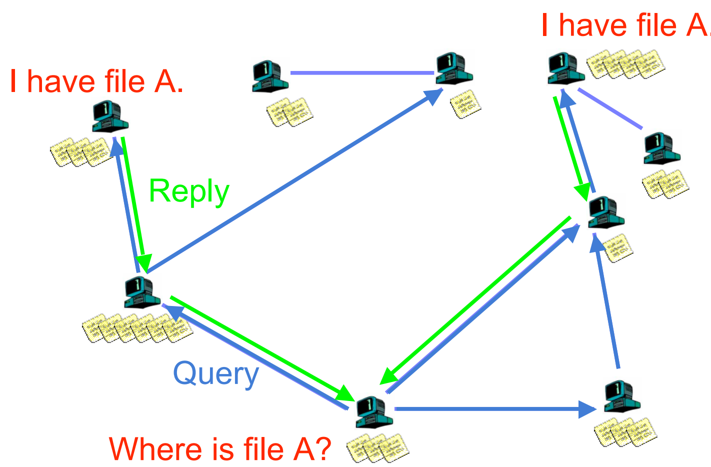
Napster: Discussion

- Pros:
  - Simple
  - Search scope is O(1)
  - Controllable (pro or con?)
- Cons:
  - Server maintains O(N) State
  - Server does all processing
  - Single point of failure

Query Flooding: Gnutella

- On March 14th 2000, J. Frankel and T. Pepper from AOL's Nullsoft division (also the developers of the popular Winamp mp3 player) released Gnutella
- Within hours, AOL pulled the plug on it
- Quickly reverse-engineered and soon many other clients became available: Bearshare, Morpheus, LimeWire, etc.
- In 2001, many protocol enhancements including "ultrapeers"

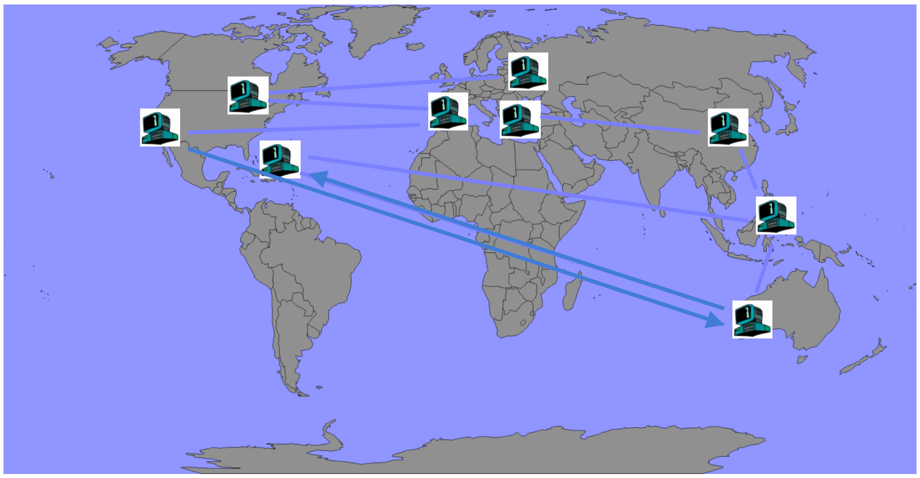
Gnutella: Search



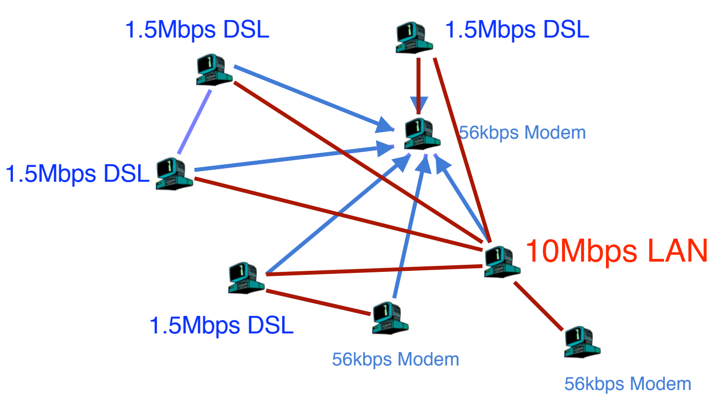
Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
- Cons:
  - Search scope is O(*N*)
  - Search time is O(???)
  - Nodes leave often, network unstable

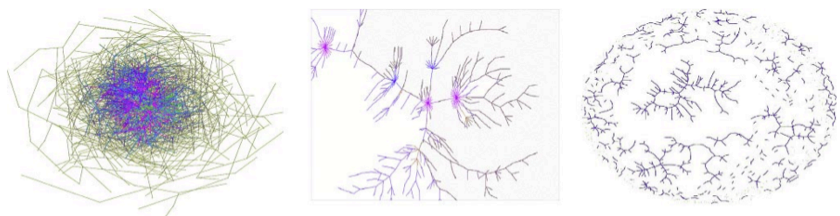
Aside: Search Time?



Aside: All Peers Equal?



Aside: Network Resilience



Partial Topology — Random 30% die — Targeted 4% die

from Saroiu *et al.*, *MMCN* 2002

Flooding: FastTrack (aka Kazaa)

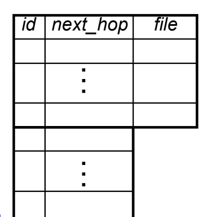
- Modifies the Gnutella protocol into two-level hierarchy
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
- Search
  -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

Freenet: Smart Routing

- In 1999, I. Clarke started the Freenet project
- Basic Idea:
  - Employ Internet-like routing on the overlay network to publish and locate files
- Additional goals:
  - Provide anonymity and security
  - Make censorship difficult

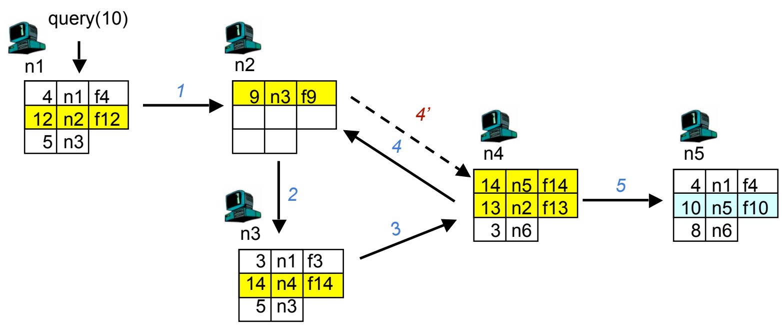
Freenet: Routing Tables

- ***id*** – file identifier (e.g., hash of file)
- ***next_hop*** – another node that stores the file id
- ***file*** – file identified by *id* being stored on the local node



- Forwarding of query for file *id*
  - If file *id* stored locally, then stop
    - Forward data back to upstream requestor
  - If not, search for the "closest" *id* in the table, and forward the message to the corresponding *next_hop*
  - If data is not found, failure is reported back
    - Requestor then tries next closest match in routing table

Freenet: Routing



Freenet: Overview

- Routed Queries:
  - **Search**: route query for *file id* toward the closest *node id*
  - **Fetch**: when query reaches a node containing *file id*, it returns the file to the sender through the intermediate nodes
    - Update routing table entries
  - **Publish**: route file contents toward the *file id*. File is stored at node with *id* closest to *file id*

Freenet: Routing Properties

- "Close" file ids tend to be stored on the same node
  - Why? Publications of similar file ids route toward the same place
- Network tend to be a "small world"
  - Small number of nodes have large number of neighbors (i.e., ~ "six-degrees of separation")
- Consequence:
  - Most queries only traverse a small number of hops to find the file

Freenet: Discussion

- Pros:
  - Intelligent routing makes queries relatively short
  - Search scope small (only nodes along search path involved); no flooding
  - Anonymity properties may give you "plausible deniability"
- Cons:
  - Still no provable guarantees!
  - Anonymity features make it hard to measure, debug

BitTorrent: Swarming Exchange

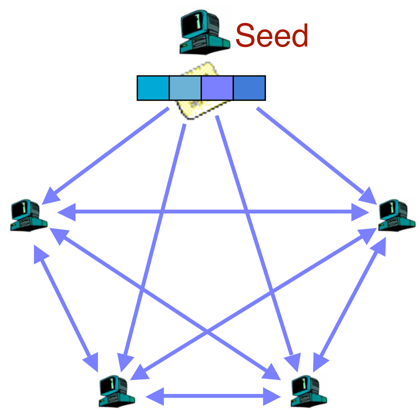
- In 2002, B. Cohen debuted BitTorrent
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release

  - Previous p2p systems had the problem with free-riding
  - 70% of Gnutella users didn't contribute
  - Used "tit-for-tat" after breaking up a file into blocks

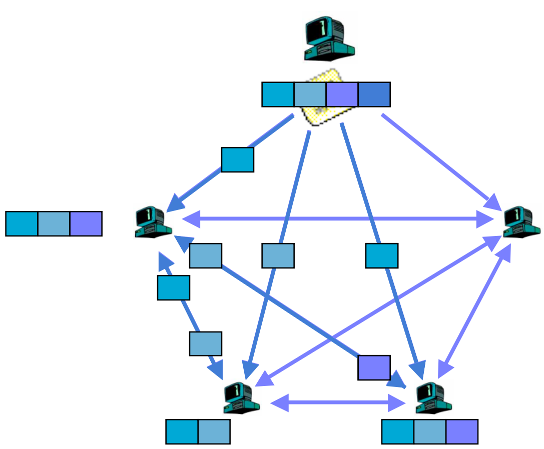
Overview

- Focused on Efficient *Fetching*, not *Searching (out-of-band)*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Swarming:
  - Join: contact centralized "tracker" server, get a list of peers.
  - Fetch: Download chunks of the file from your peers. Upload chunks you have to them.

BitTorrent: Publish/Join



BitTorrent: Fetch



BitTorrent: Sharing Strategy

- Employ "Tit-for-tat" sharing strategy
  - "I'll share with you if you share with me"
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no liveness guarantees
    - Also allows you to discover better peers to download from when they reciprocate

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Peer selection is crucial
  - Central tracker server needed to bootstrap swarm

Topic Outline

- Unstructured paradigm for p2p computing
  - Centralized Database: Napster
  - Query Flooding: Gnutella
  - Intelligent Query Flooding: Freenet
  - Swarming exchange: BitTorrent
- Structured paradigm for p2p computing
  - Distributed Hash Tables

Distributed Hash Tables (DHT): History

- In 2000-2001, academic researchers jumped on to the P2P bandwagon
- Motivation:
  - Frustrated by popularity of all these "half-baked" P2P apps. We can do better! (so they said)
  - Guaranteed lookup success for files in system
  - Provable bounds on search time
  - Provable scalability to millions of node
- Hot topic in networking ever since

DHT: Overview

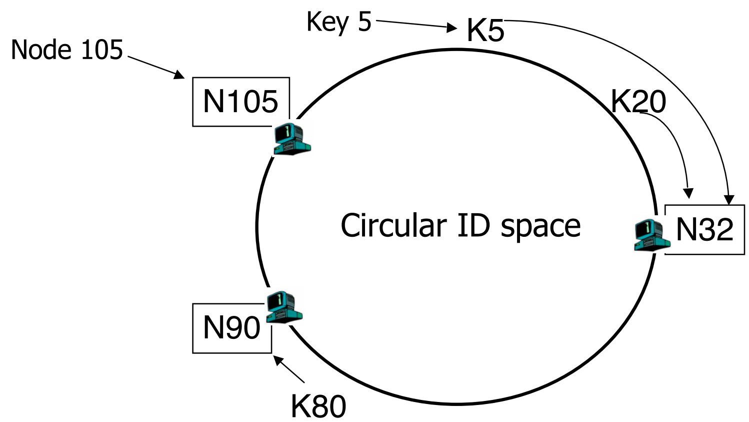
- **Abstraction**: a distributed "hash-table" (DHT) data structure:
  - put(id, item);
  - item = get(id);
- **Implementation**: nodes in system form an interconnection network
  - Can be Ring, Tree, Hypercube, Butterfly Network, ...

DHT: Example - Chord

- Associate with each node and file a unique *id* in an *uni-*dimensional space (a Ring)
  - E.g., pick from the range $[0...2^m]$
  - Usually the hash of the file or IP address
- Properties:
  - Routing table size is O(log *N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log *N*) hops

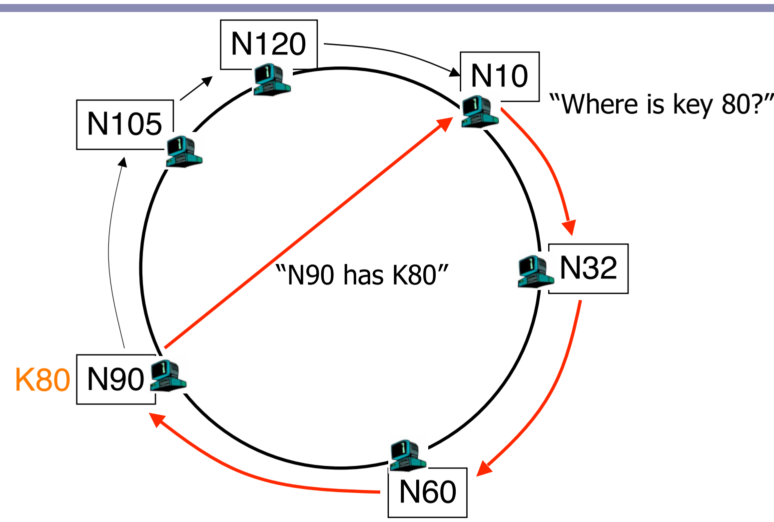
from MIT in 2001

DHT: Consistent Hashing

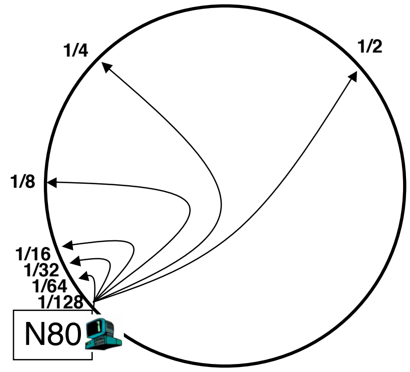


A key is stored at its successor: node with next higher ID

DHT: Chord Basic Lookup



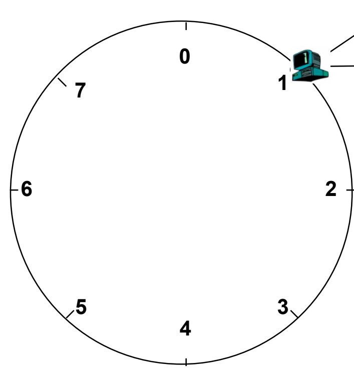
DHT: Chord "Finger Table"



- Entry *i* in the finger table of node *n* is the first node that succeeds or equals $n + 2^i$
- In other words, the ith finger points $1/2^{n-i}$ way around the ring

DHT: Chord Join

- Assume an identifier space [0..8]
- Node n1 joins

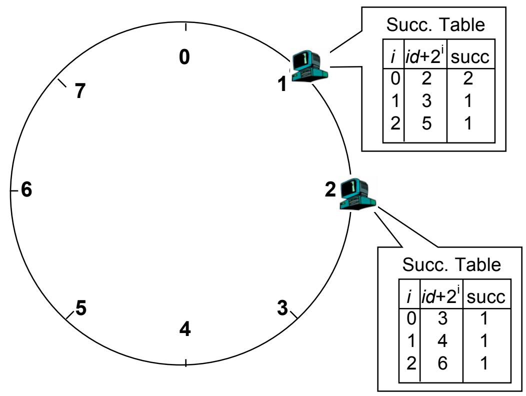


Succ. Table

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

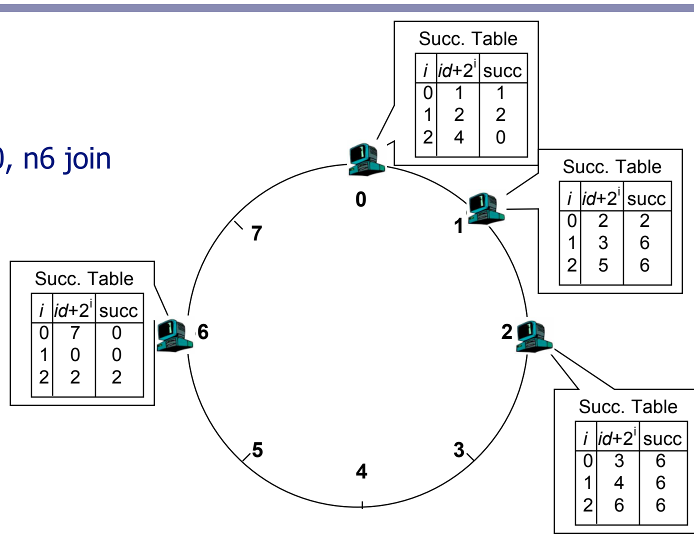
DHT: Chord Join

- Node n2 joins



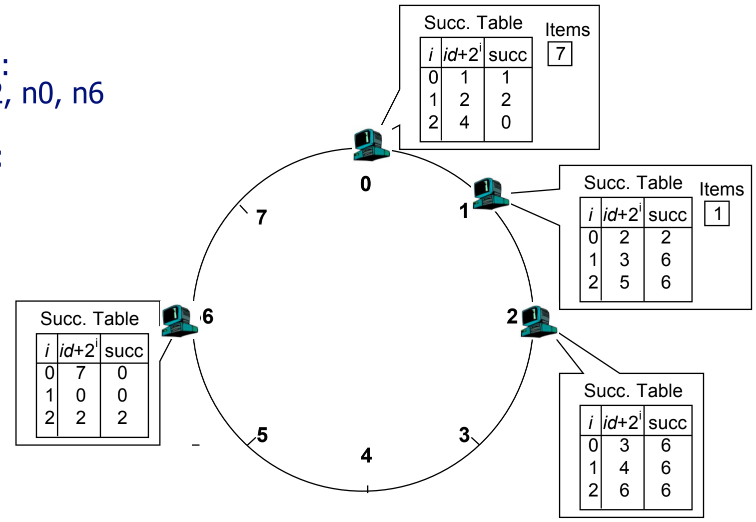
DHT: Chord Join

- Nodes n0, n6 join



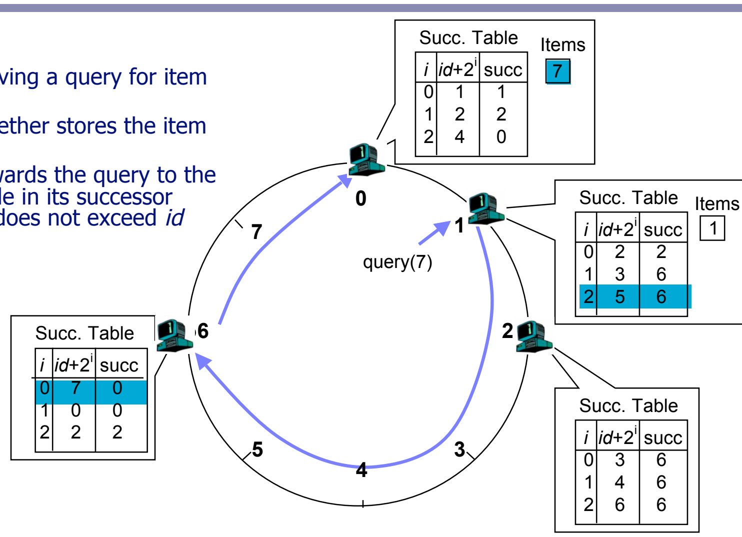
DHT: Chord Join

- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f1



DHT: Chord Routing

- Upon receiving a query for item *id*, a node:
- Checks whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



DHT: Chord Summary

- Routing table size?
  - Log *N* fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log *N*) hops.

- What is good/bad about Chord?