Peer-to-Peer Networks

Outline

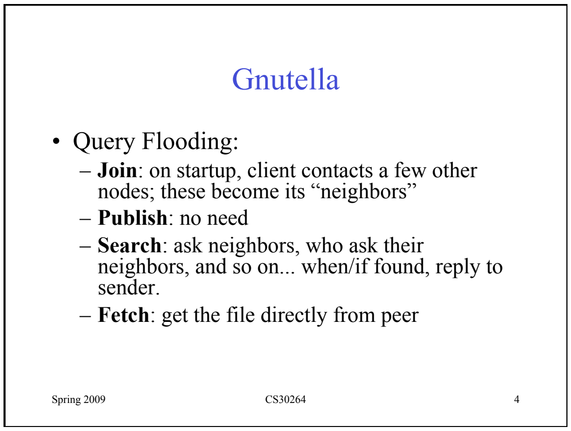
Domain Name System
Peer-to-Peer Networks

P2P

- Overview:
  - centralized database: Napster
  - query flooding: Gnutella
  - intelligent query flooding: KaZaA
  - swarming: BitTorrent
  - unstructured overlay routing: Freenet
  - structured overlay routing: Distributed Hash Tables

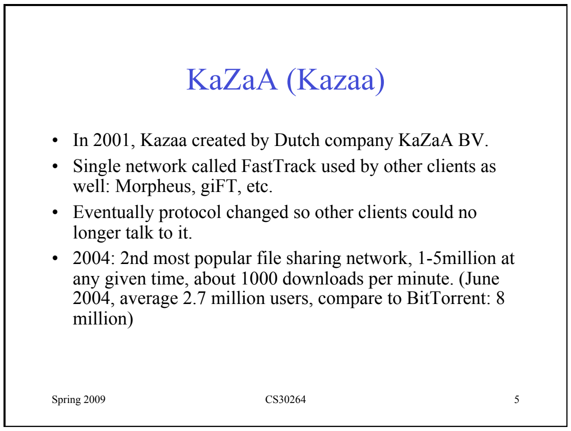
Napster

- Centralized Database:
  - **Join**: on startup, client contacts central server
  - **Publish**: reports list of files to central server
  - **Search**: query the server => return someone that stores the requested file
  - **Fetch**: get the file directly from peer



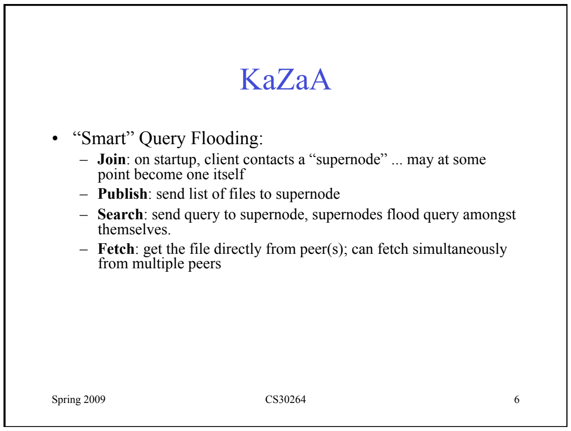
Gnutella

- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its “neighbors”
  - **Publish**: no need
  - **Search**: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
  - **Fetch**: get the file directly from peer



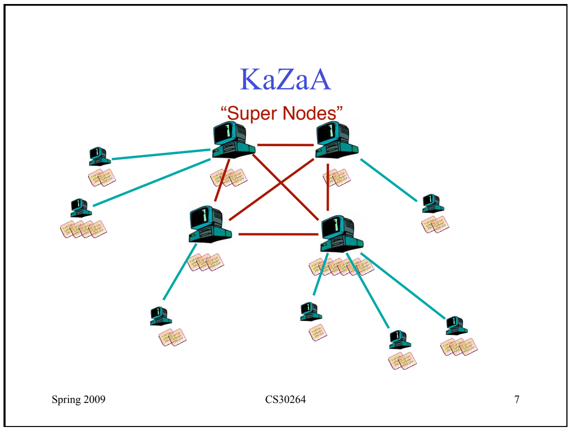
KaZaA (Kazaa)

- In 2001, Kazaa created by Dutch company KaZaA BV.
- Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
- Eventually protocol changed so other clients could no longer talk to it.
- 2004: 2nd most popular file sharing network, 1-5million at any given time, about 1000 downloads per minute. (June 2004, average 2.7 million users, compare to BitTorrent: 8 million)



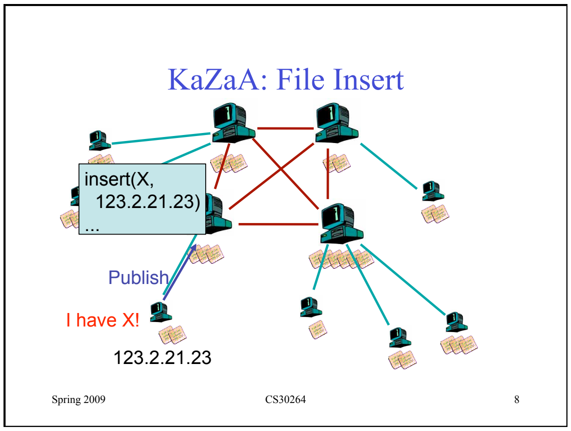
KaZaA

- “Smart” Query Flooding:
  - **Join**: on startup, client contacts a “supernode” ... may at some point become one itself
  - **Publish**: send list of files to supernode
  - **Search**: send query to supernode, supernodes flood query amongst themselves.
  - **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers



KaZaA

“Super Nodes”



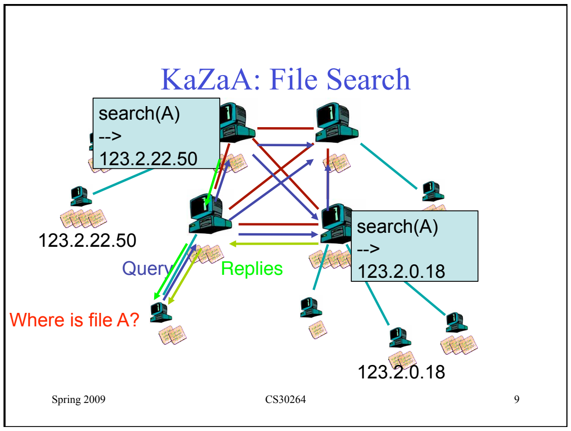
KaZaA: File Insert

insert(X,
  123.2.21.23)
...

Publish

I have X!

123.2.21.23



KaZaA: File Search

search(A)
-->
123.2.22.50

123.2.22.50

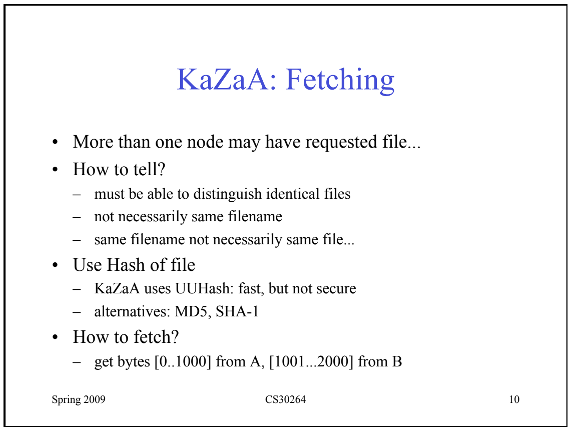
search(A)
-->
123.2.0.18

Query

Replies

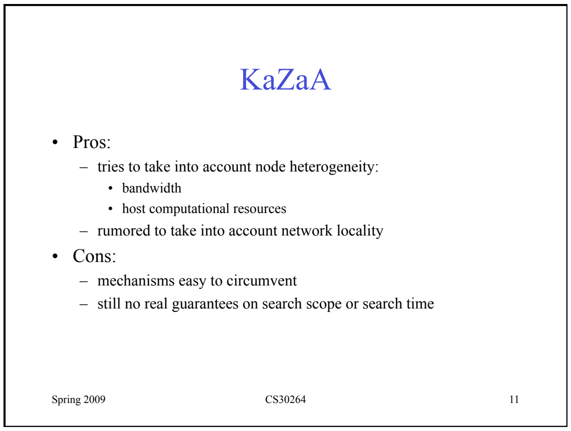
Where is file A?

123.2.0.18



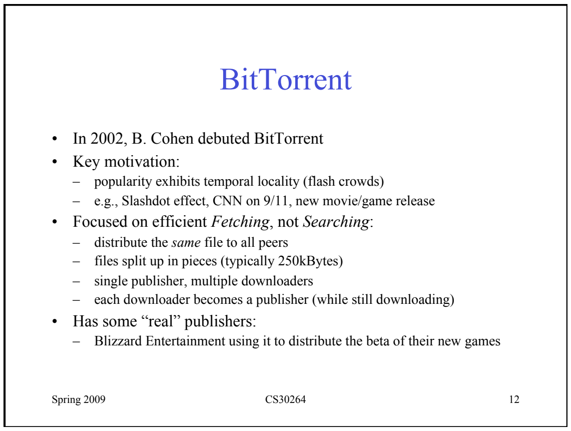
KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - must be able to distinguish identical files
  - not necessarily same filename
  - same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - alternatives: MD5, SHA-1
- How to fetch?
  - get bytes [0..1000] from A, [1001...2000] from B



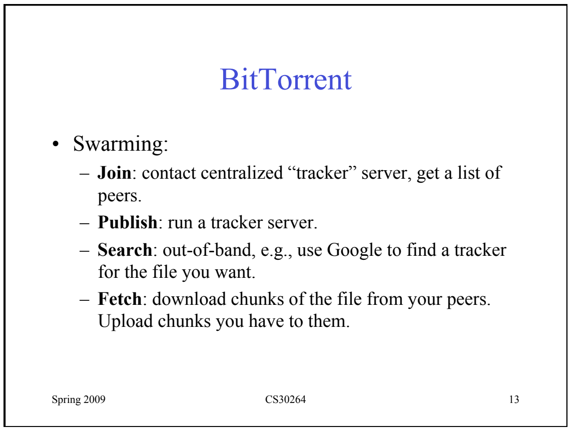
KaZaA

- Pros:
  - tries to take into account node heterogeneity:
    - bandwidth
    - host computational resources
  - rumored to take into account network locality
- Cons:
  - mechanisms easy to circumvent
  - still no real guarantees on search scope or search time



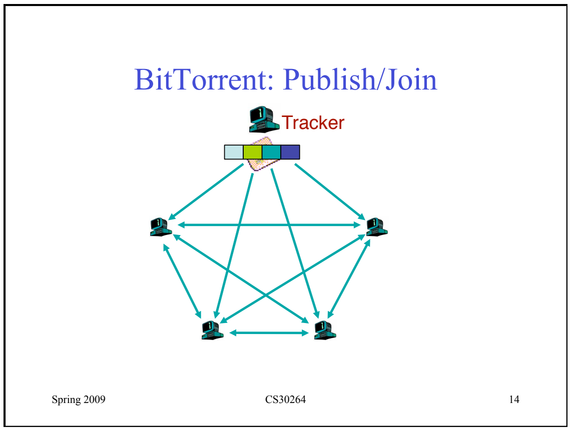
BitTorrent

- 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
- Focused on efficient *Fetching*, not *Searching*:
  - distribute the *same* file to all peers
  - files split up in pieces (typically 250kBytes)
  - single publisher, multiple downloaders
  - each downloader becomes a publisher (while still downloading)
- Has some “real” publishers:
  - Blizzard Entertainment using it to distribute the beta of their new games

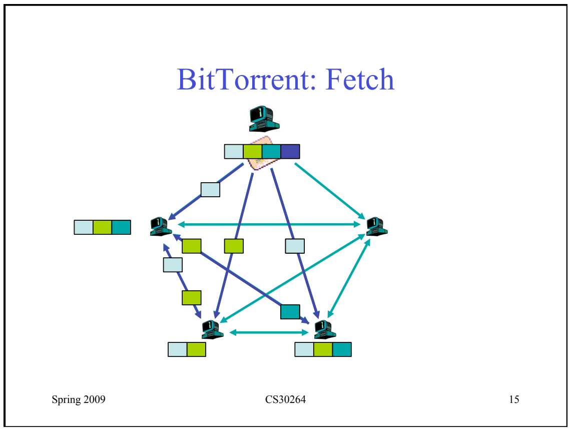


BitTorrent

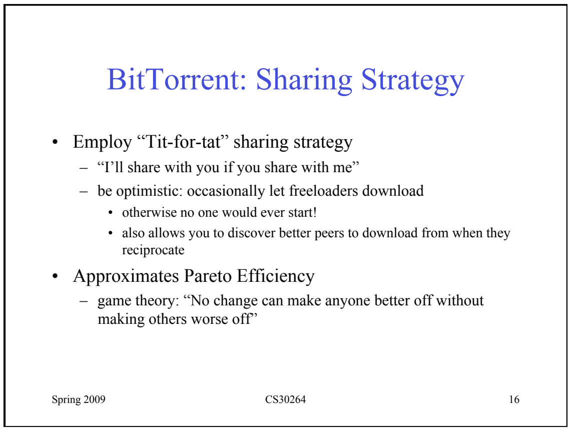
- Swarming:
  - **Join**: contact centralized “tracker” server, get a list of peers.
  - **Publish**: run a tracker server.
  - **Search**: out-of-band, e.g., use Google to find a tracker for the file you want.
  - **Fetch**: download chunks of the file from your peers. Upload chunks you have to them.



BitTorrent: Publish/Join

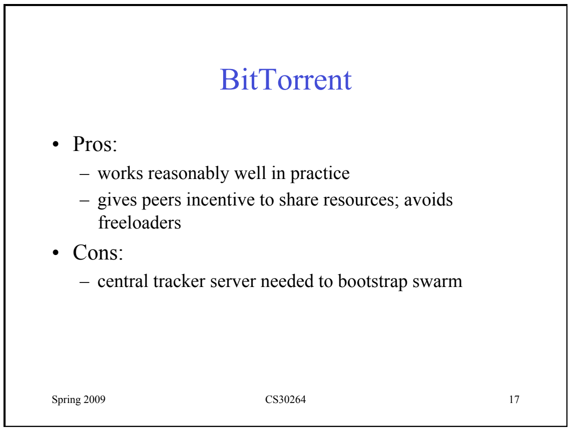


BitTorrent: Fetch



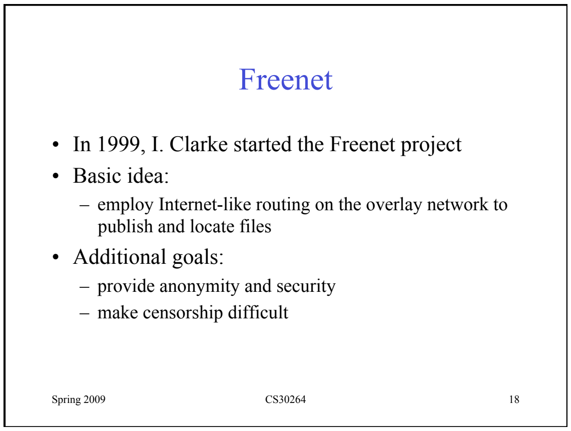
BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - “I’ll share with you if you share with me”
  - be optimistic: occasionally let freeloaders download
    - otherwise no one would ever start!
    - also allows you to discover better peers to download from when they reciprocate
- Approximates Pareto Efficiency
  - game theory: “No change can make anyone better off without making others worse off”



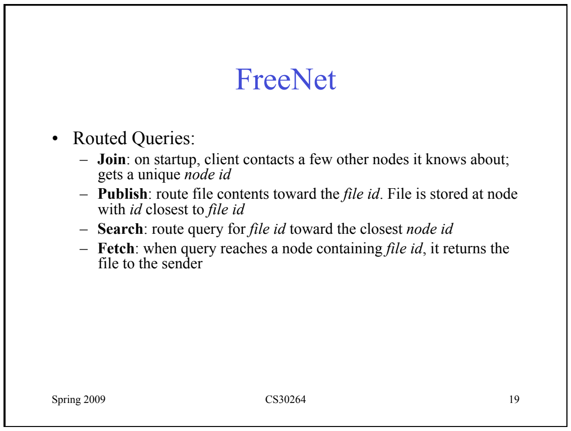
BitTorrent

- Pros:
  - works reasonably well in practice
  - gives peers incentive to share resources; avoids freeloaders
- Cons:
  - central tracker server needed to bootstrap swarm



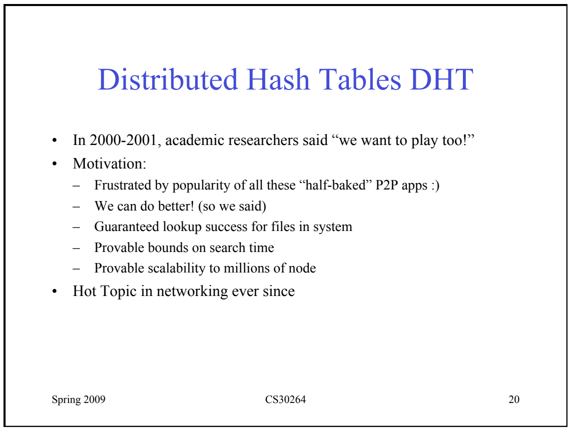
Freenet

- 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



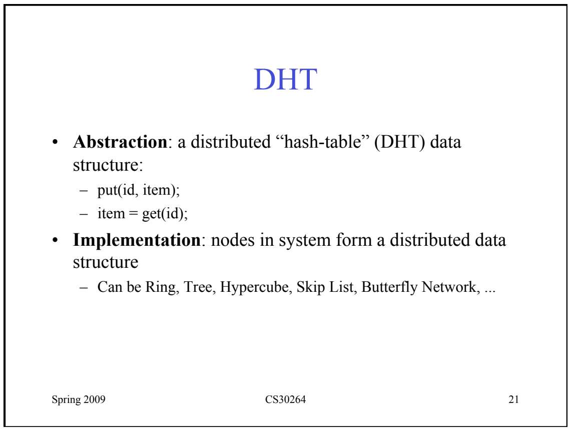
FreeNet

- Routed Queries:
  - **Join**: on startup, client contacts a few other nodes it knows about; gets a unique *node id*
  - **Publish**: route file contents toward the *file id*. File is stored at node with *id* closest to *file id*
  - **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



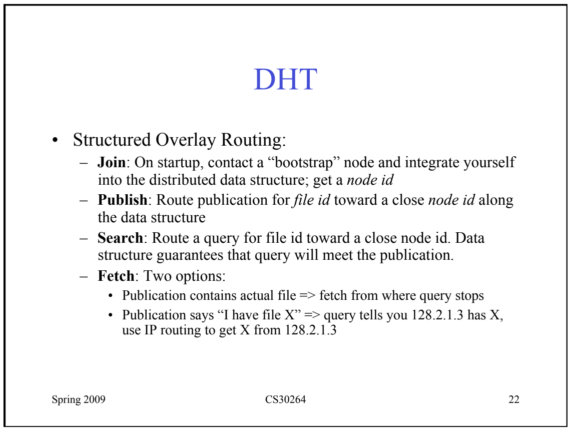
Distributed Hash Tables DHT

- In 2000-2001, academic researchers said “we want to play too!”
- Motivation:
  - Frustrated by popularity of all these “half-baked” P2P apps :)
  - We can do better! (so we 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



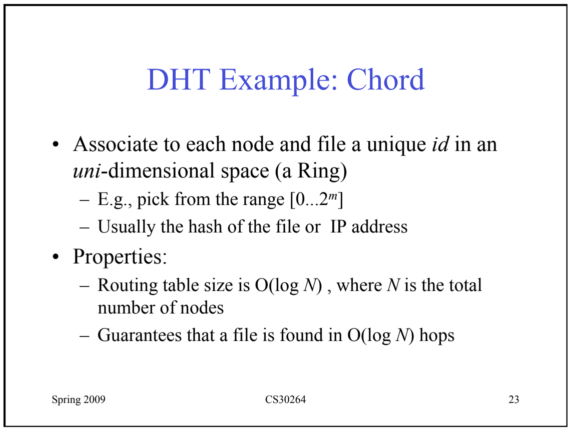
DHT

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



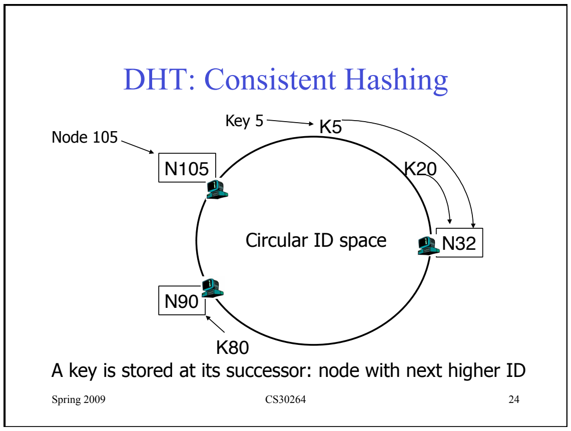
DHT

- Structured Overlay Routing:
  - **Join**: On startup, contact a “bootstrap” node and integrate yourself into the distributed data structure; get a *node id*
  - **Publish**: Route publication for *file id* toward a close *node id* along the data structure
  - **Search**: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - **Fetch**: Two options:
    - Publication contains actual file => fetch from where query stops
    - Publication says “I have file X” => query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3



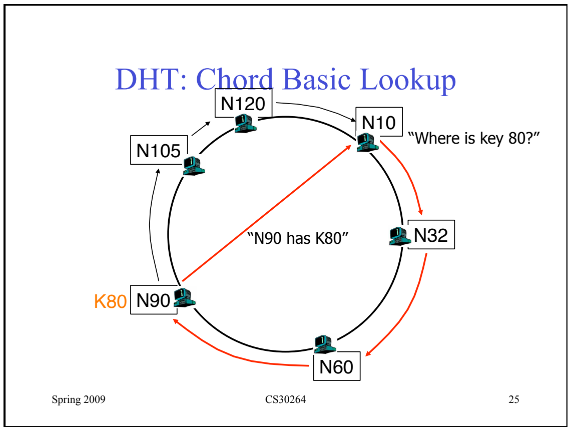
DHT Example: Chord

- Associate to 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

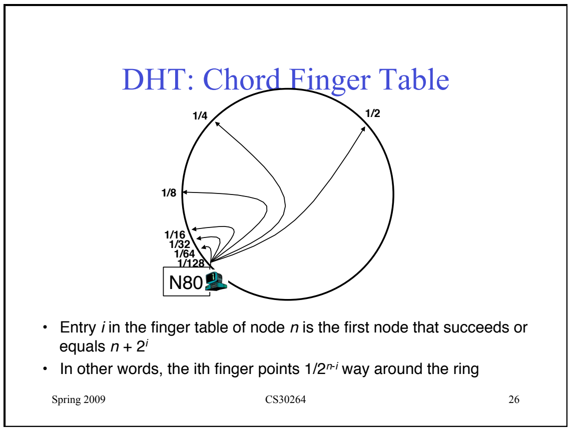


DHT: Consistent Hashing

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

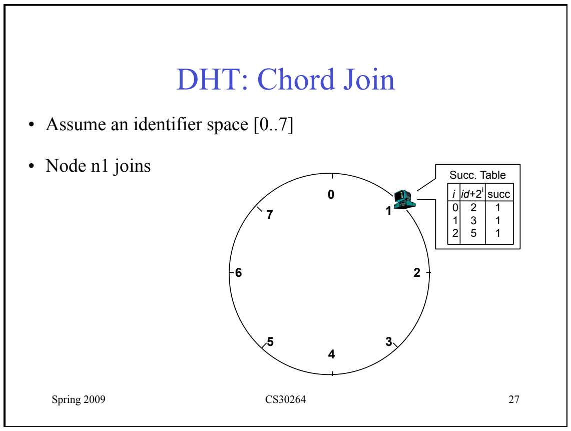


DHT: Chord Basic Lookup



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

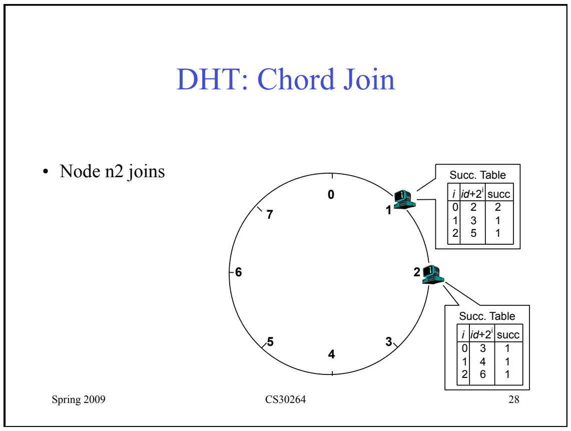


DHT: Chord Join

- Assume an identifier space [0..7]
- Node n1 joins

Succ. Table

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



DHT: Chord Join

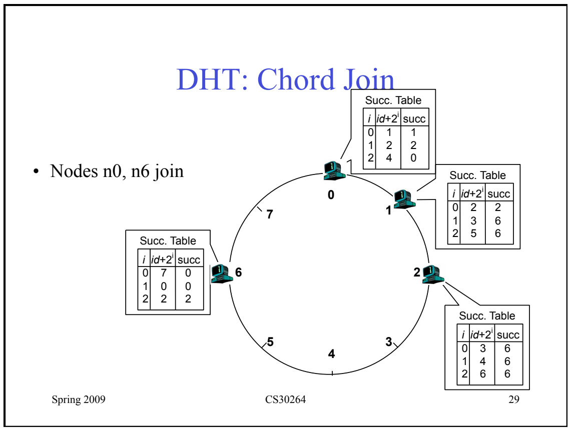
- Node n2 joins

Succ. Table

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

Succ. Table

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



DHT: Chord Join

- Nodes n0, n6 join

Succ. Table

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

Succ. Table

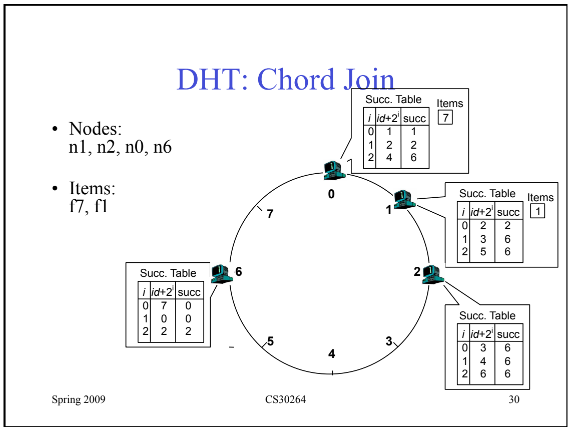
i	$id+2^i$	succ
0	2	2
1	3	6
2	5	6

Succ. Table

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

Succ. Table

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



DHT: Chord Join

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

Succ. Table

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

Items
7

Succ. Table

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

Items
1

Succ. Table

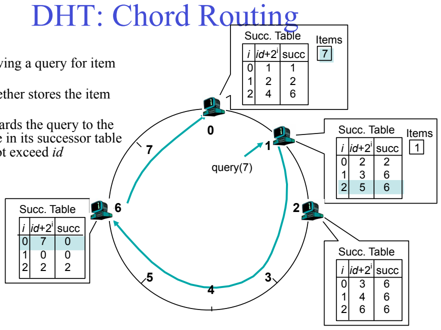
i	$id+2^i$	succ
0	7	0
1	0	0
2	2	2

Succ. Table

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

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*



Succ. Table (node 0)

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

Items: 7

Succ. Table (node 1)

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

Items: 1

Succ. Table (node 2)

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

Succ. Table (node 6)

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

DHT

- Pros:
  - Guaranteed Lookup
  - O(log *N*) per node state and search scope
- Cons:
  - No one uses them? (only one file sharing app)
  - Supporting non-exact match search is hard

P2P Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees