Selected Topics in CN

Nguyễn Quốc Đính

Faculty of IT, Ho Chi Minh City University of Industry

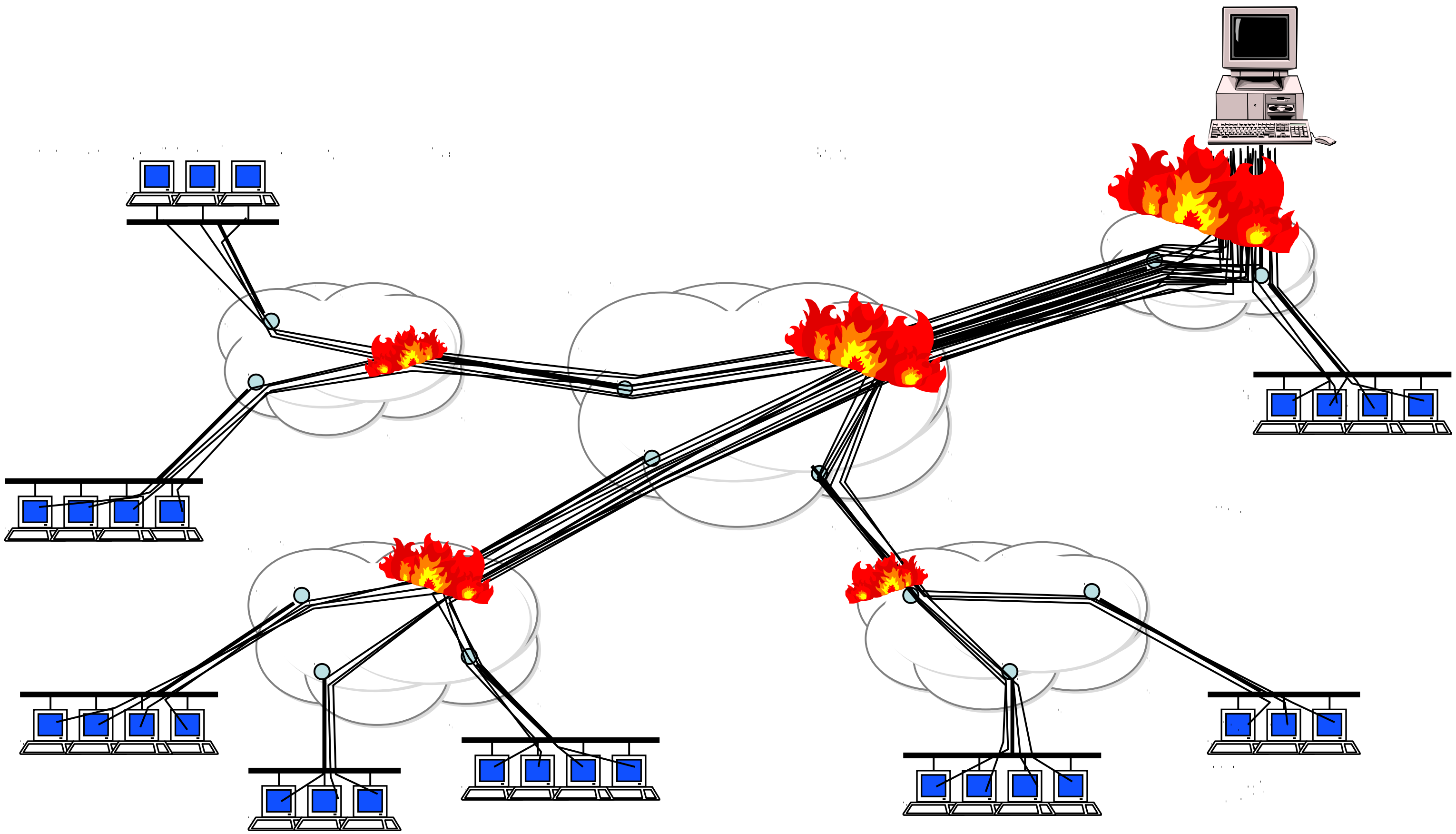
Aug 2015

Peer to Peer
P2P

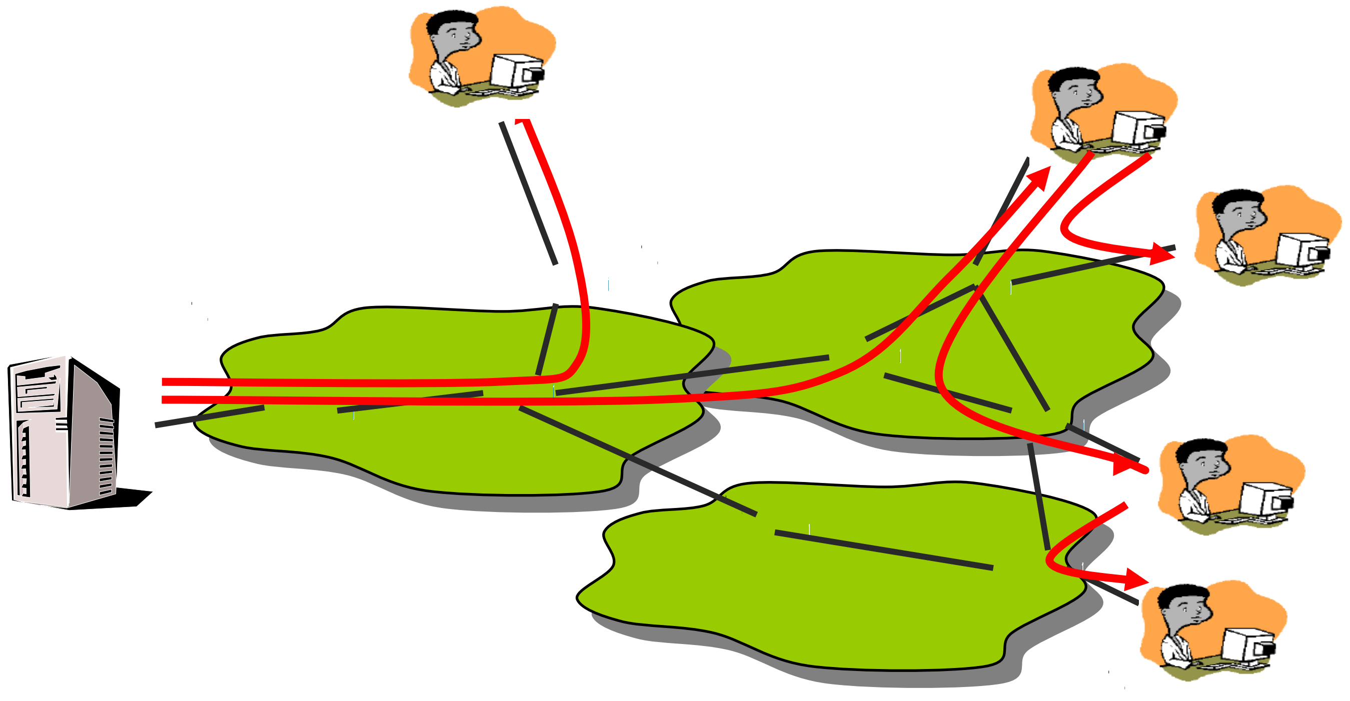
Majority part of this presentation are copied from Roger Dannenberg, Srinivasan Seshan, lecture notes in Computer Network, CMU

Scaling problem

- Million of client → server and network meltdown



P2P system



- Leverage the resources [computation, bandwidth, storage] of client machines

Why P2P

Harness lots of spare capacity

- 1 Big Fast Server: 1Gbit/s, $10k/month++
- 2,000 cable modems: 1Gbit/s, $??
- 1M end-hosts: ?

	Start	Silver	Silver Plus	Gold	Gold Plus	Platinum	Platinum Plus
	32.000 VNĐ / tháng	81.000 VNĐ / tháng	108.000 VNĐ / tháng	135.000 VNĐ / tháng	252.000 VNĐ / tháng	360.000 VNĐ / tháng	900.000 VNĐ / tháng
Dung lượng	300MB	800MB	1000MB	1200MB	2GB	3GB	7,5GB
Băng thông	5GB	15GB	20GB	30GB	50GB	120GB	300GB
Email	10 địa chỉ	50 địa chỉ	70 địa chỉ	100 địa chỉ	150 địa chỉ	500 địa chỉ	1.500 địa chỉ
Sub domain	1	2	5	8	12	30	90
PHP	yes	yes	yes	yes	yes	yes	yes
My SQL	1	2	4	7	10	12	20

http://www.matbao.net/Hosting/Bang-gia-Linux-Hosting.aspx

Why P2P

- Build self-managing systems / Deal with huge scale
  - Same techniques attractive for both companies / servers / p2p
    - E.g., Akamai’s 14,000 nodes
    - Google’s 100,000+ nodes
- First distributed systems that seriously focused on scalability with respect to number of nodes
- P2P techniques abound in cloud computing systems
  - Key-value stores (e.g., Cassandra, Riak, Voldemort) use Chord p2p hashing

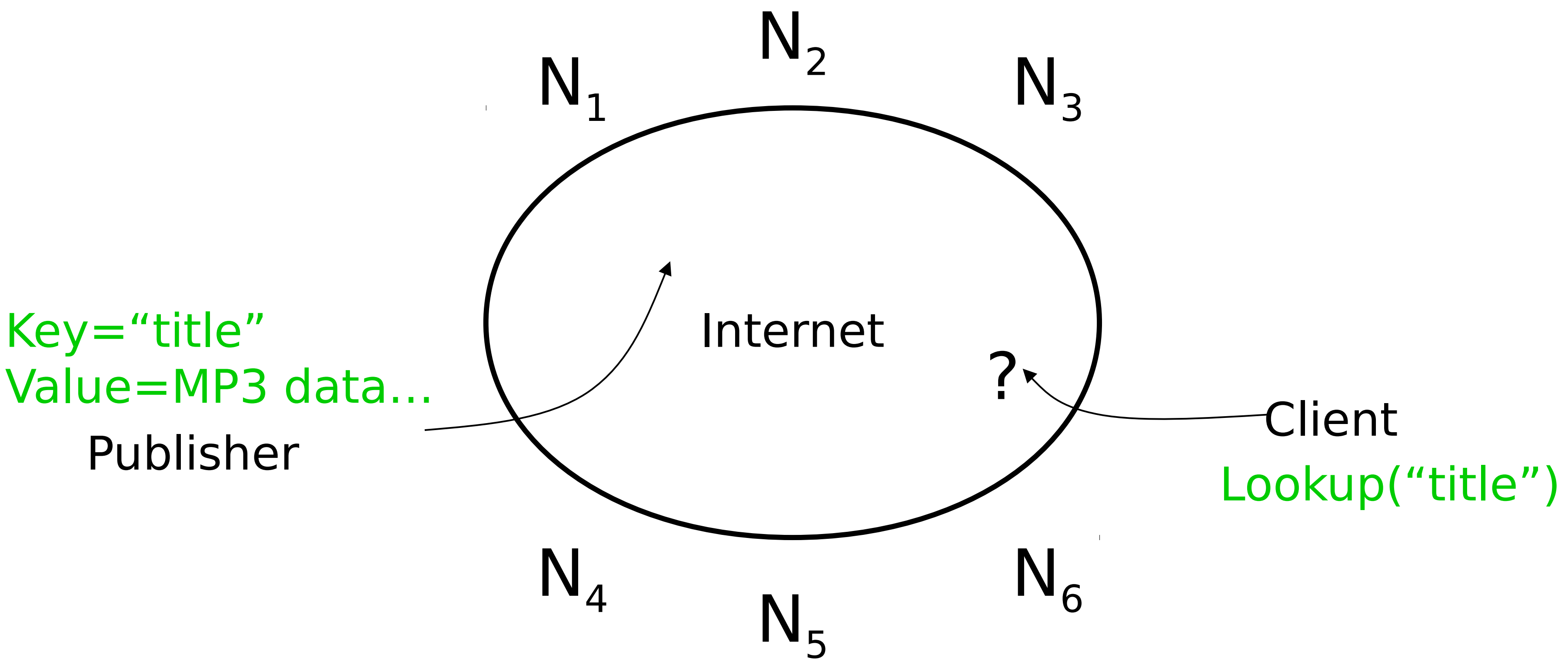
Outline

- p2p file sharing techniques
  - Downloading: Whole-file vs. chunks
- Searching
  - Centralized index (Napster, etc.)
  - Flooding (Gnutella, etc.)
  - Smarter flooding (KaZaA, …)
  - Routing (Freenet, etc.)
- Uses of p2p - what works well, what doesn’t?
  - servers vs. arbitrary nodes
- Challenges
  - Fairness, freeloading, security, …

We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

Searching


N_1
N_2
N_3
Key="title"
Value=MP3 data…
Publisher
Internet
?
Client
Lookup("title")
N_4
N_5
N_6

Searching 2

- Needles vs. Haystacks
  - Searching for top 40, or an obscure punk track from 1981 that nobody’s heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search?

Framework

- Common Primitives:
  - Join: how do I begin participating?
  - Publish: how do I advertise my file?
  - Search: how to I find a file?
  - Fetch: how to I retrieve a file?

We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

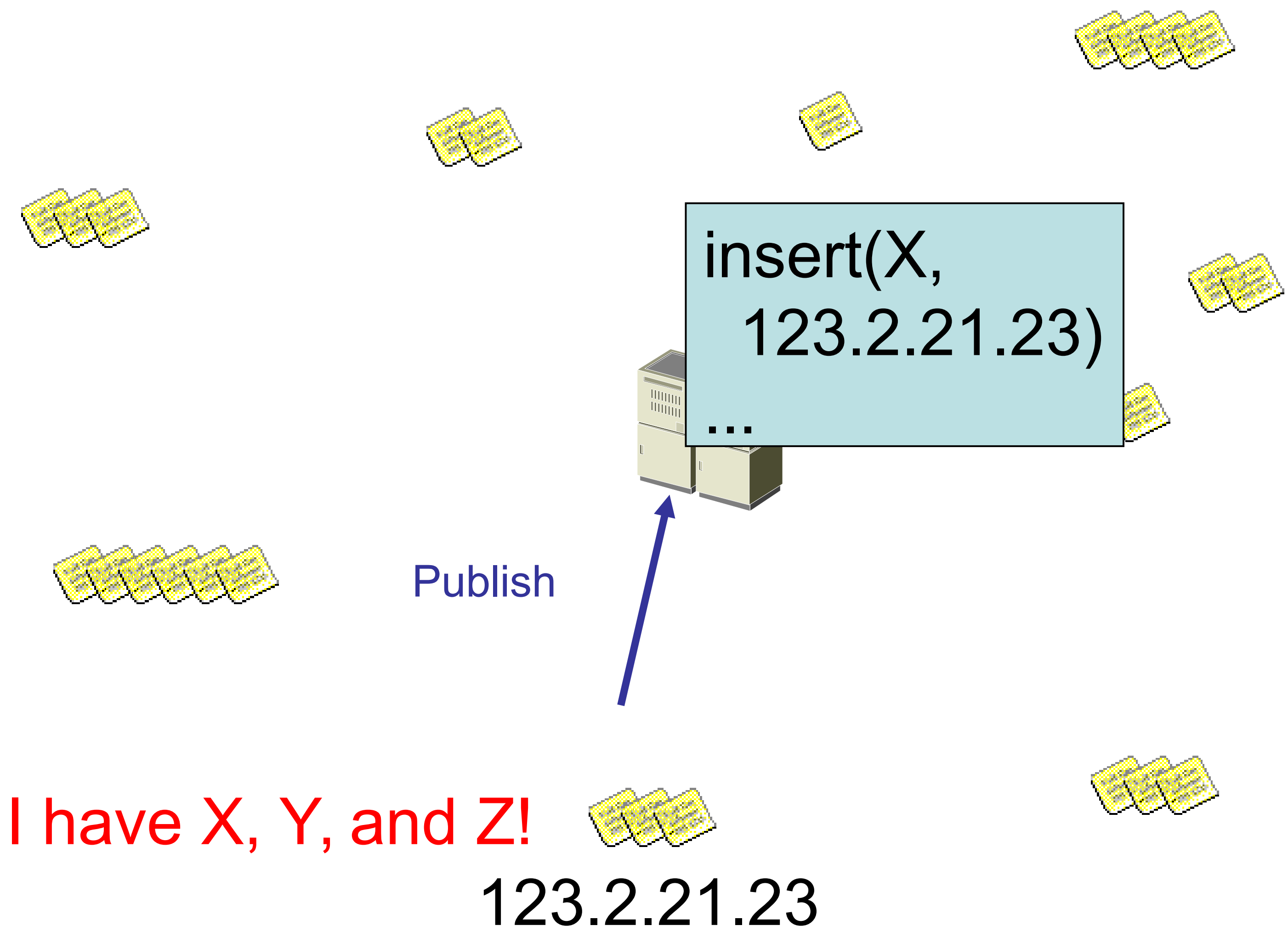
A Brief History

- [6/99] Shawn Fanning (freshman Northeastern U.) releases Napster online music service
- [12/99] RIAA sues Napster, asking $100K per download
- [3/00] 25% UWisc traffic Napster, many universities ban it
- [00] 60M users
- [2/01] US Federal Appeals Court: users violating copyright laws, Napster is abetting this
- [9/01] Napster decides to run paid service, pay % to songwriters and music companies
- [Today] Napster protocol is open, people free to develop opennap clients and servers http://opennap.sourceforge.net
  - Gnutella: http://www.limewire.com (deprecated)
  - Peer to peer working groups: http://p2p.internet2.edu

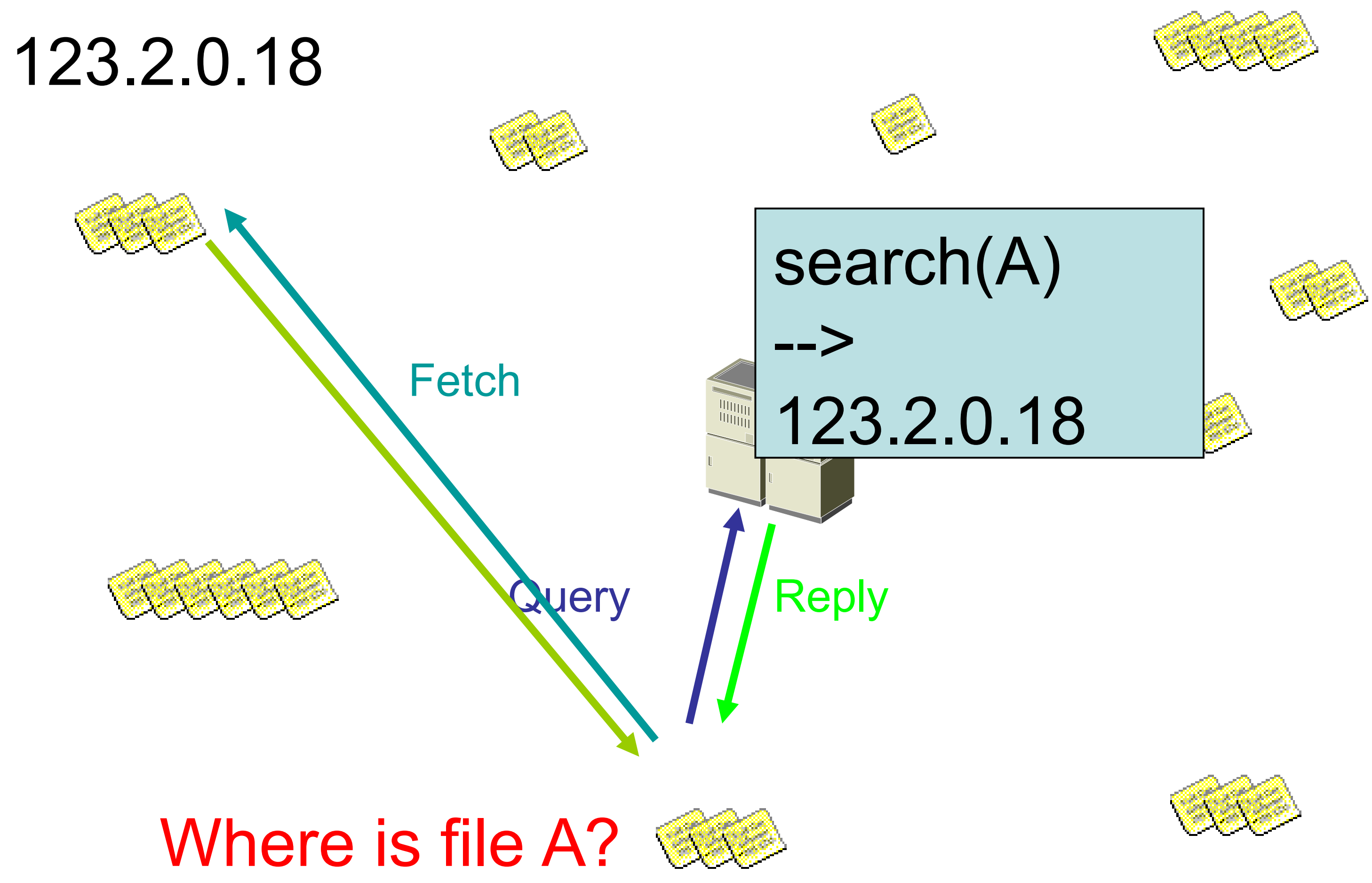
Napster Overview

- 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

Napster: Publish



Napster: Search



Napster Operation

- Server maintains list of <filename, ip_address, portnum> tuples. Server stores no files.

- All communication uses TCP
  - Reliable and ordered networking protocol

- No security: plaintext messages and passwds

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

We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

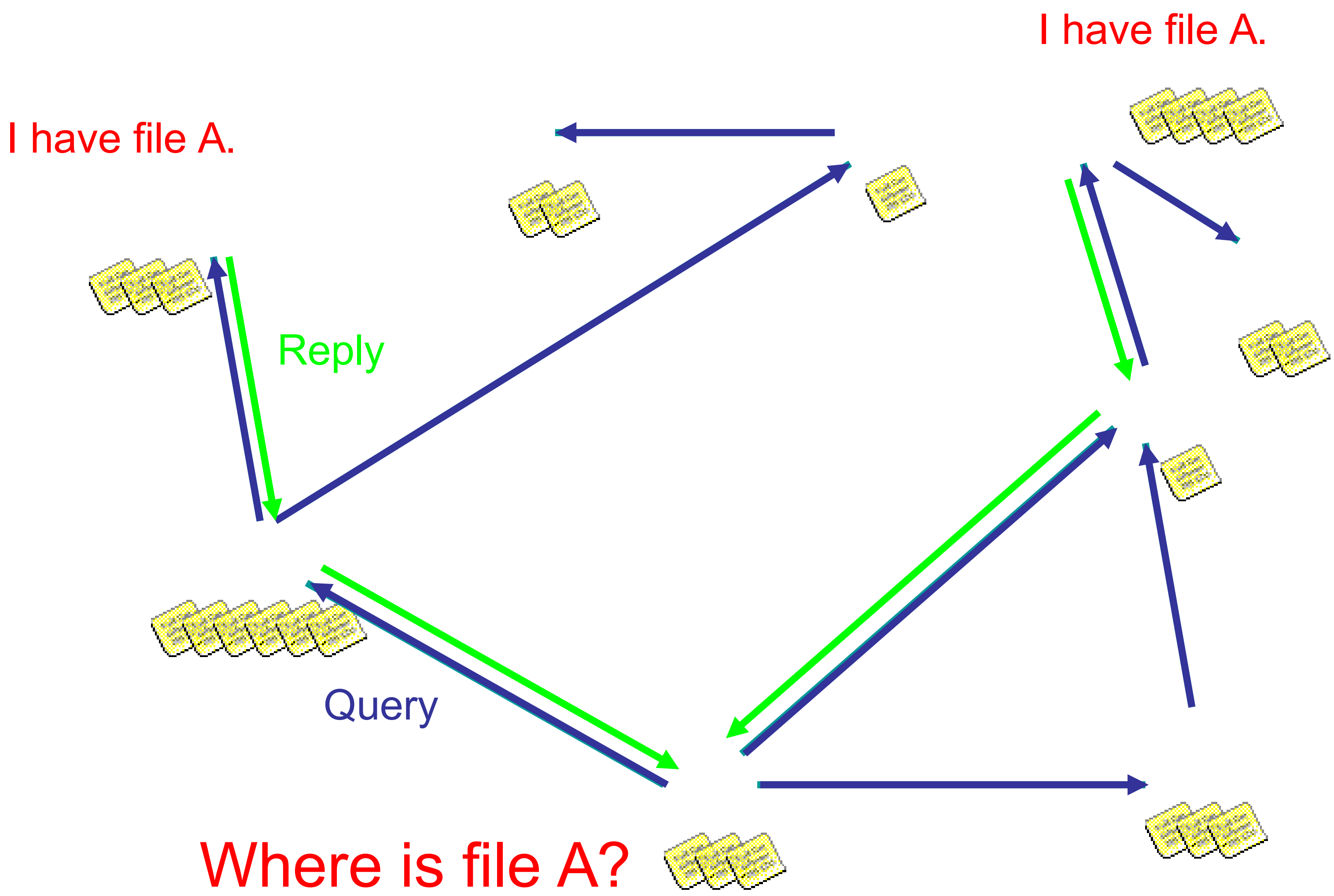
Gnutella: History

- In 2000, J. Frankel and T. Pepper from Nullsoft released Gnutella
- Soon many other clients: Bearshare, Morpheus, LimeWire, etc.
- In 2001, many protocol enhancements including “ultrapeers”

Gnutella: Overview

- 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.
    - TTL limits propagation
  - Fetch: get the file directly from peer

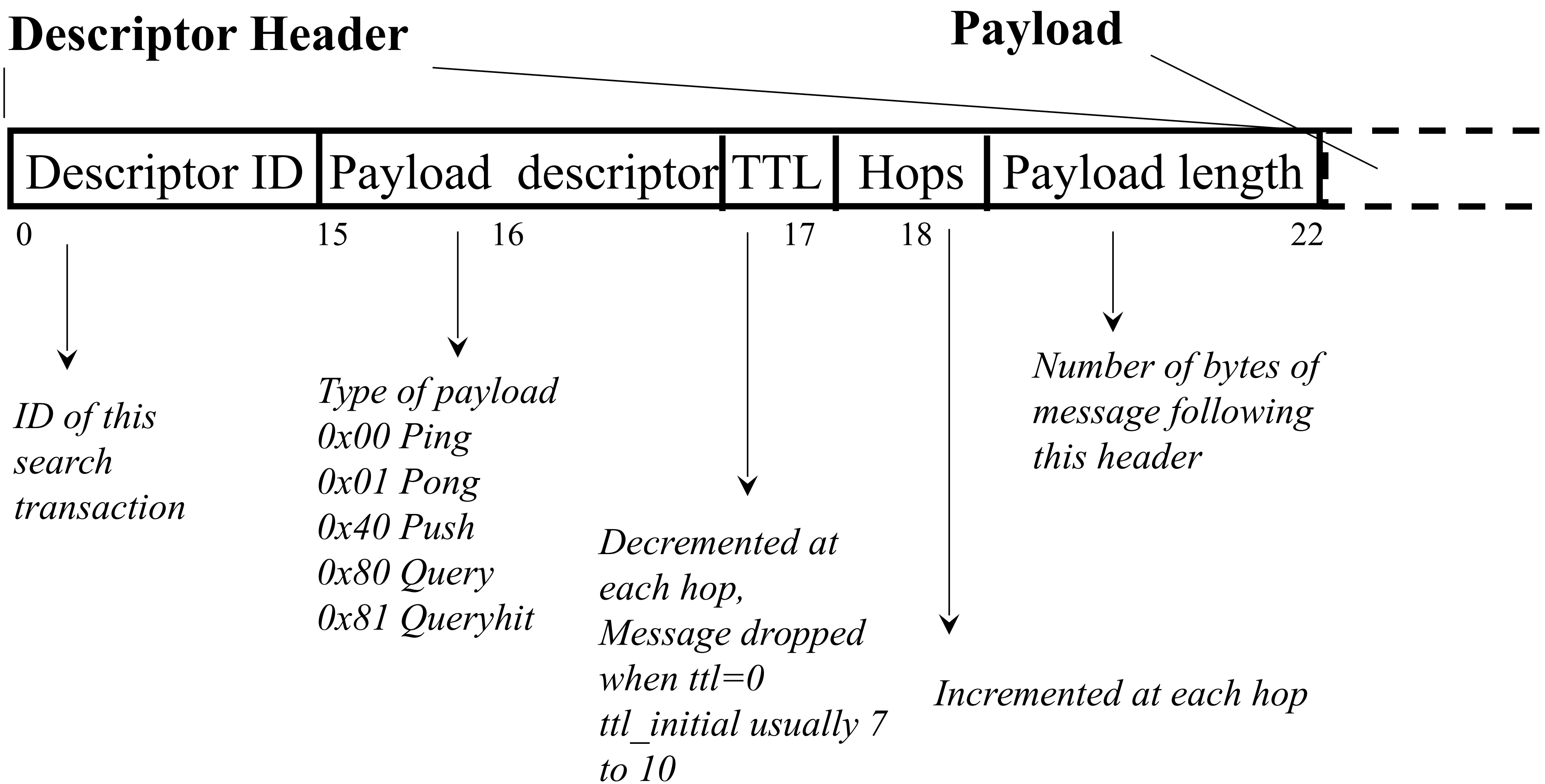
Gnutella: Joint/Search



Gnutella: Operation

- Gnutella routes different messages within the overlay graph
- Gnutella protocol has 5 main message types
  - Query (search)
  - QueryHit (response to query)
  - Ping (to probe network for other peers)
  - Pong (reply to ping, contains address of another peer)
  - Push (used to initiate file transfer)
- We’ll go into the message structure and protocol now
  - All fields except IP address are in little-endian format
  - 0x12345678 stored as 0x78 in lowest address byte, then 0x56 in next higher address, and so on.

How Do I Search for M-TP file?



Gnutella Message Header Format

How Do I Search for M-TP file?

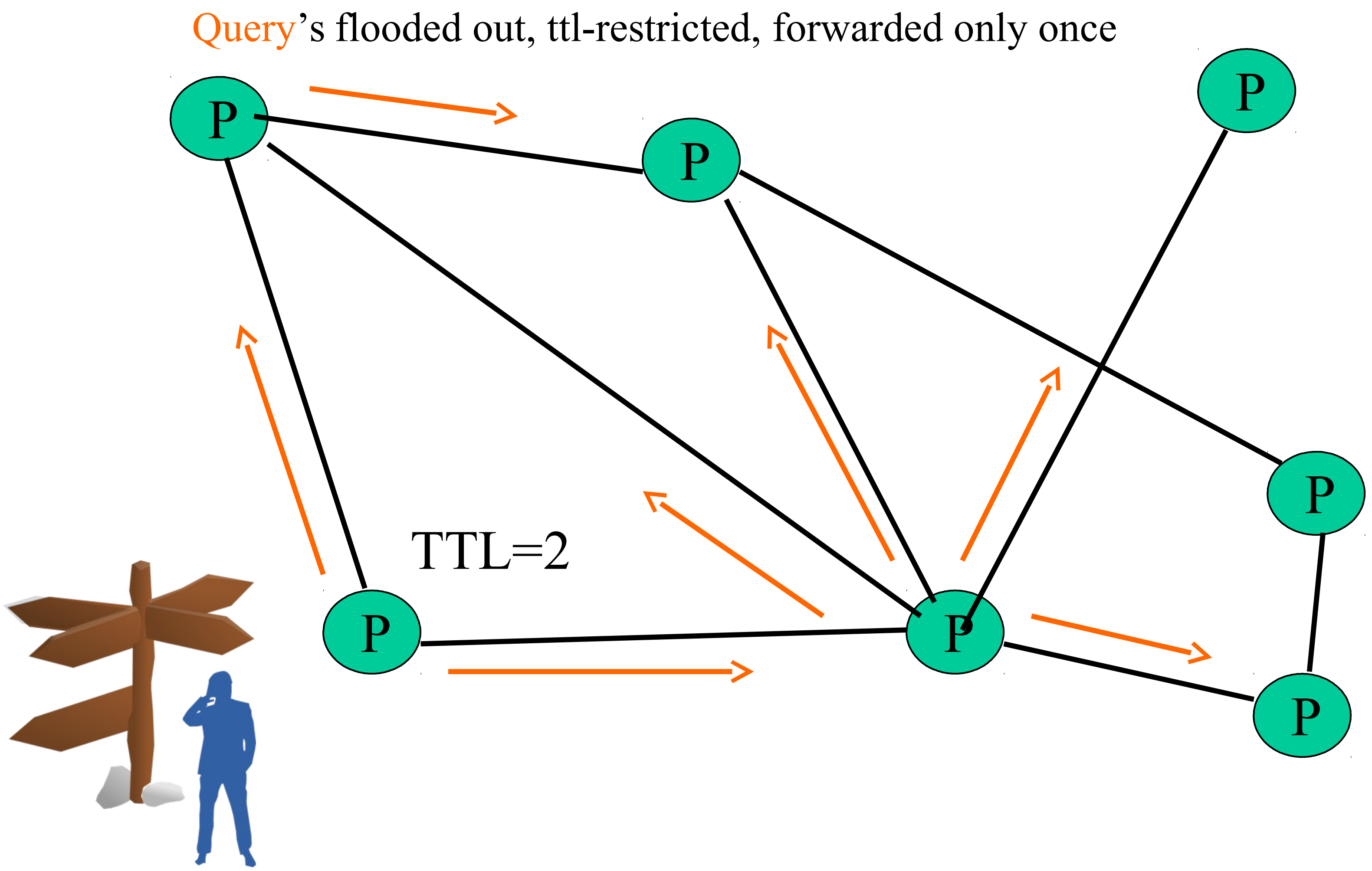
Query (0x80)

Minimum Speed	Search criteria (keywords)

0 1 …..

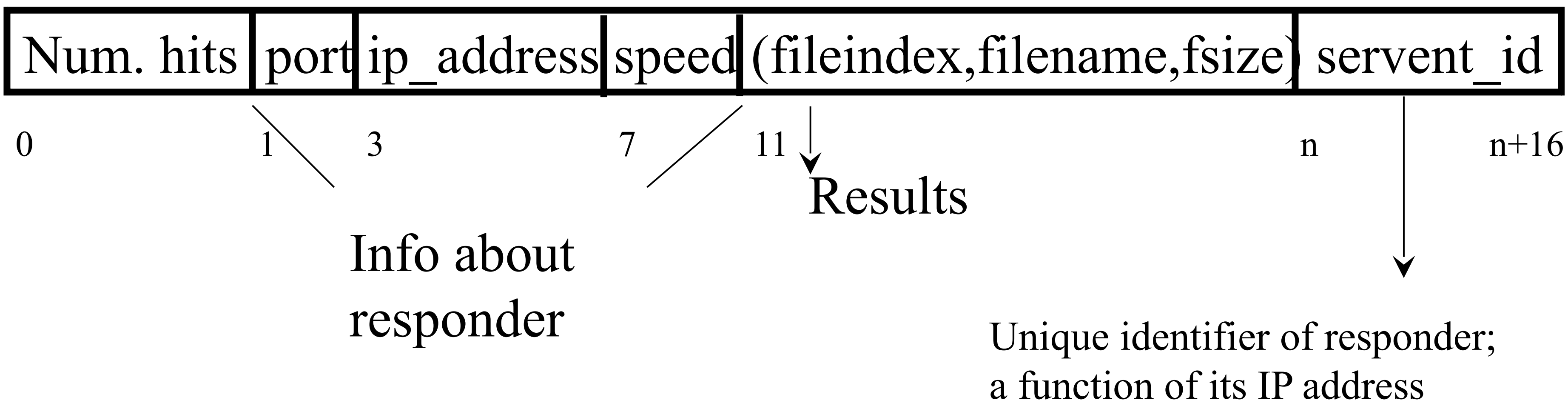
Payload Format in Gnutella Query Message

Gnutella Search



Gnutella Search

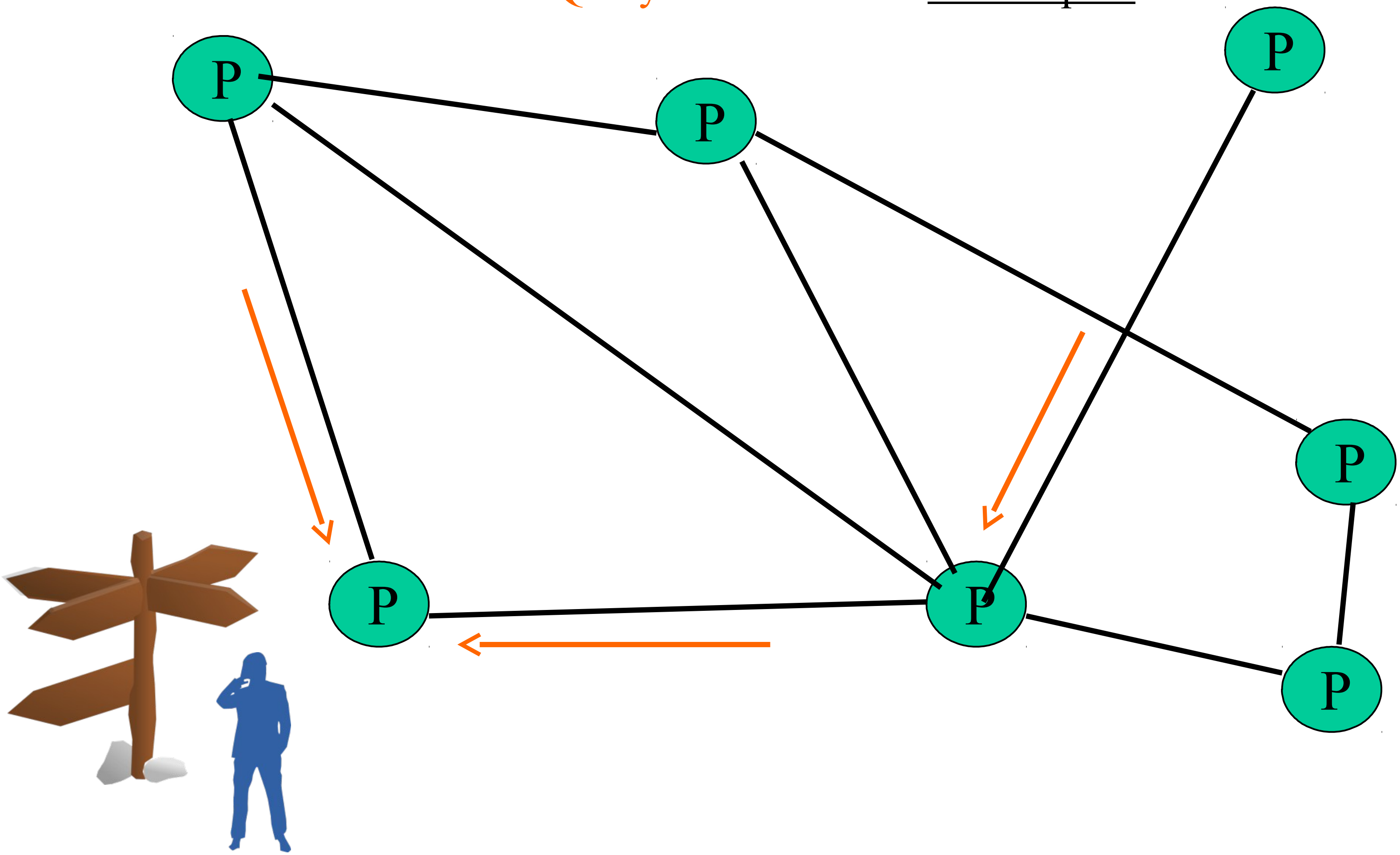
QueryHit (0x81) : successful result to a query



Payload Format in Gnutella QueryHit Message

Gnutella Search

Successful results QueryHit's routed on reverse path



Avoiding excessive traffic

- To avoid duplicate transmissions, each peer maintains a list of recently received messages
- Query forwarded to all neighbors except peer from which received
- Each Query (identified by DescriptorID) forwarded only once
- QueryHit routed back only to peer from which Query received with same DescriptorID
- Duplicates with same DescriptorID and Payload descriptor (msg type, e.g., Query) are dropped
- QueryHit with DescriptorID for which Query not seen is dropped

After receiving QueryHit messages

- Requestor chooses “best” QueryHit responder
  - Initiates HTTP request directly to responder’s ip+port

```
GET /get/<File Index>/<File Name>/HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella\r\n
\r\n
```

- Responder then replies with file packets after this message:

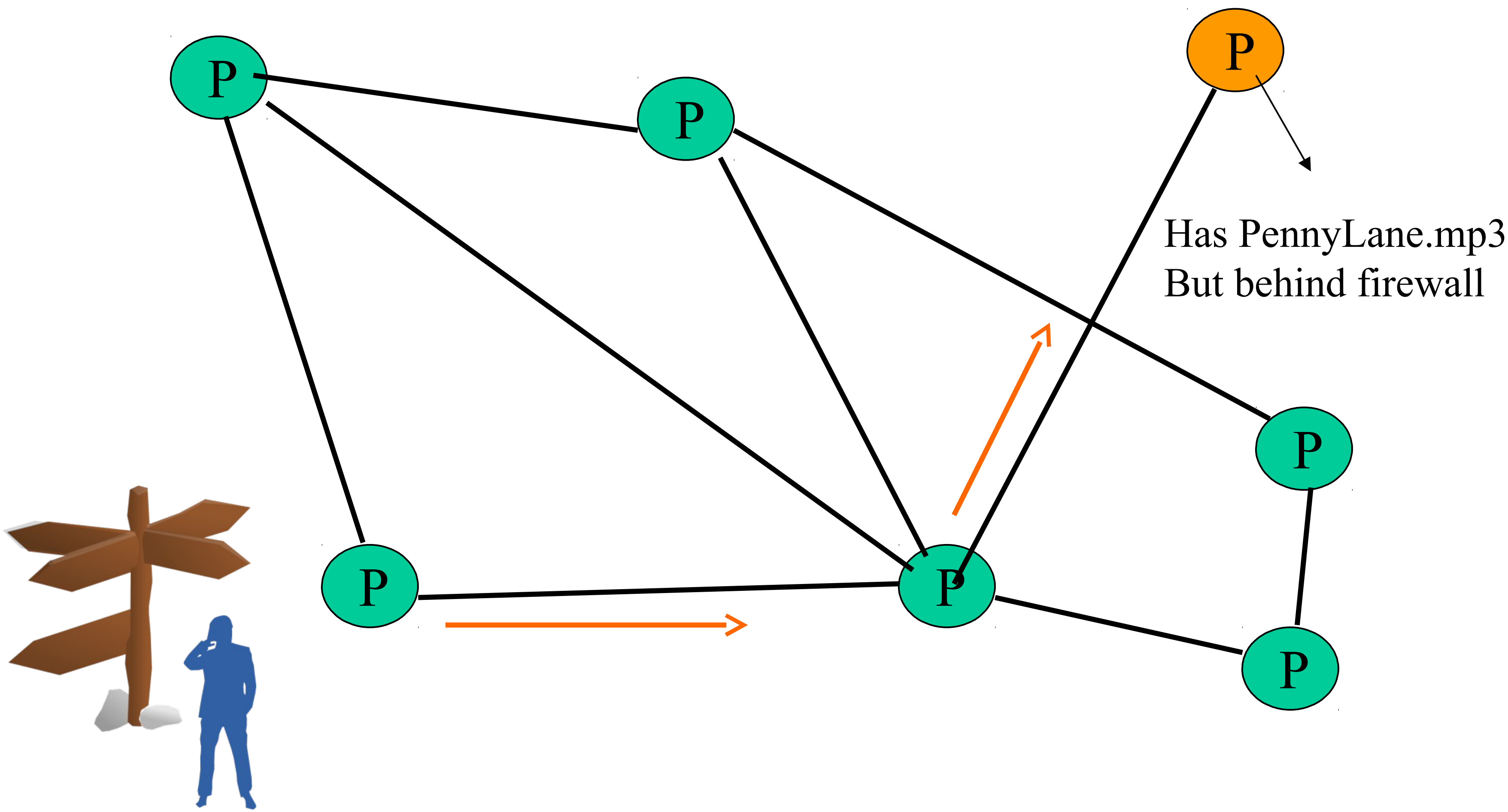
```
HTTP 200 OK\r\n
Server: Gnutella\r\n
Content-type:application/binary\r\n
Content-length: 1024 \r\n
\r\n
```

After receiving QueryHit messages

- HTTP is the file transfer protocol. Why?
  - Because it's standard, well-debugged, and widely used.
- Why the "range" field in the GET request?
  - To support partial file transfers.
- What if responder is behind firewall that disallows incoming connections?

Dealing with Firewalls

Requestor sends Push to responder asking for file transfer



Dealing with Firewalls

Push (0x40)

servent_id	fileindex	ip_address	port

same as in
received QueryHit

Address at which
requestor can accept
incoming connections

Dealing with Firewalls

- Responder establishes a TCP connection at ip_address, port specified. Sends

  ```
  GIV <File Index>:<Servent Identifier>/<File Name>\n\n
  ```

- Requestor then sends GET to responder (as before) and file is transferred as explained earlier
- What if requestor is behind firewall too?
  - Gnutella gives up
  - Can you think of an alternative solution?

PING – PONG

Ping (0x00)
no payload

Pong (0x01)

Port	ip_address	Num. files shared	Num. KB shared

- Peers initiate Ping's periodically
- Pings flooded out like Querys, Pongs routed along reverse path like QueryHits
- Pong replies used to update set of neighboring peers
  - to keep neighbor lists fresh in spite of peers joining, leaving and failing

Gnutella: Discussion

- No servers
- Peers/servents maintain "neighbors", this forms an overlay graph
- Peers store their own files
- Queries flooded out, ttl restricted
- QueryHit (replies) reverse path routed
- Supports file transfer through firewalls
- Periodic Ping-Pong to continuously refresh neighbor lists
  - List size specified by user at peer: heterogeneity means some peers may have more neighbors

Gnutella: Discussion

- Ping/Pong constituted 50% traffic
  - Solution: Multiplex, cache and reduce frequency of pings/pongs
- Repeated searches with same keywords
  - Solution: Cache Query, QueryHit messages
- Modem-connected hosts do not have enough bandwidth for passing Gnutella traffic
  - Solution: use a central server to act as proxy for such peers
  - Another solution? (NEXT)

Gnutella: Discussion

- Large number of freeloaders
  - 70% of users in 2000 were freeloaders
  - Only download files, never upload own files
- Flooding causes excessive traffic
  - Is there some way of maintaining meta-information about peers that leads to more intelligent routing?
  - Structured Peer-to-peer systems, e.g., Chord System

Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics
- Cons:
  - Search scope is O(N)
  - Search time is O(???)
  - Nodes leave often, network unstable
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node. May have to re-issue query later

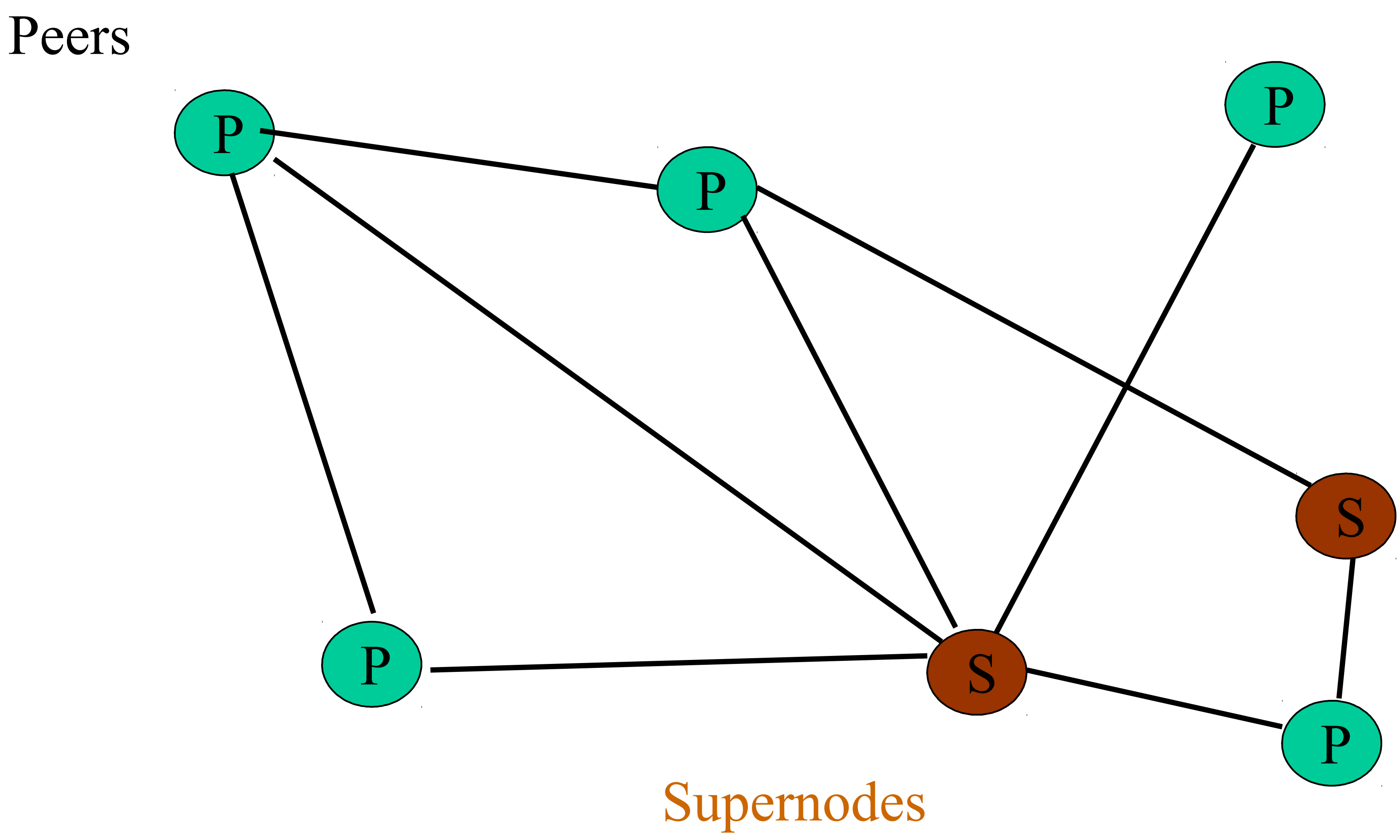
We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

KaZaA: Introduction

- 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
- Use to be the most popular file sharing network today with >10 million users (number varies)

A FastTrack-like System



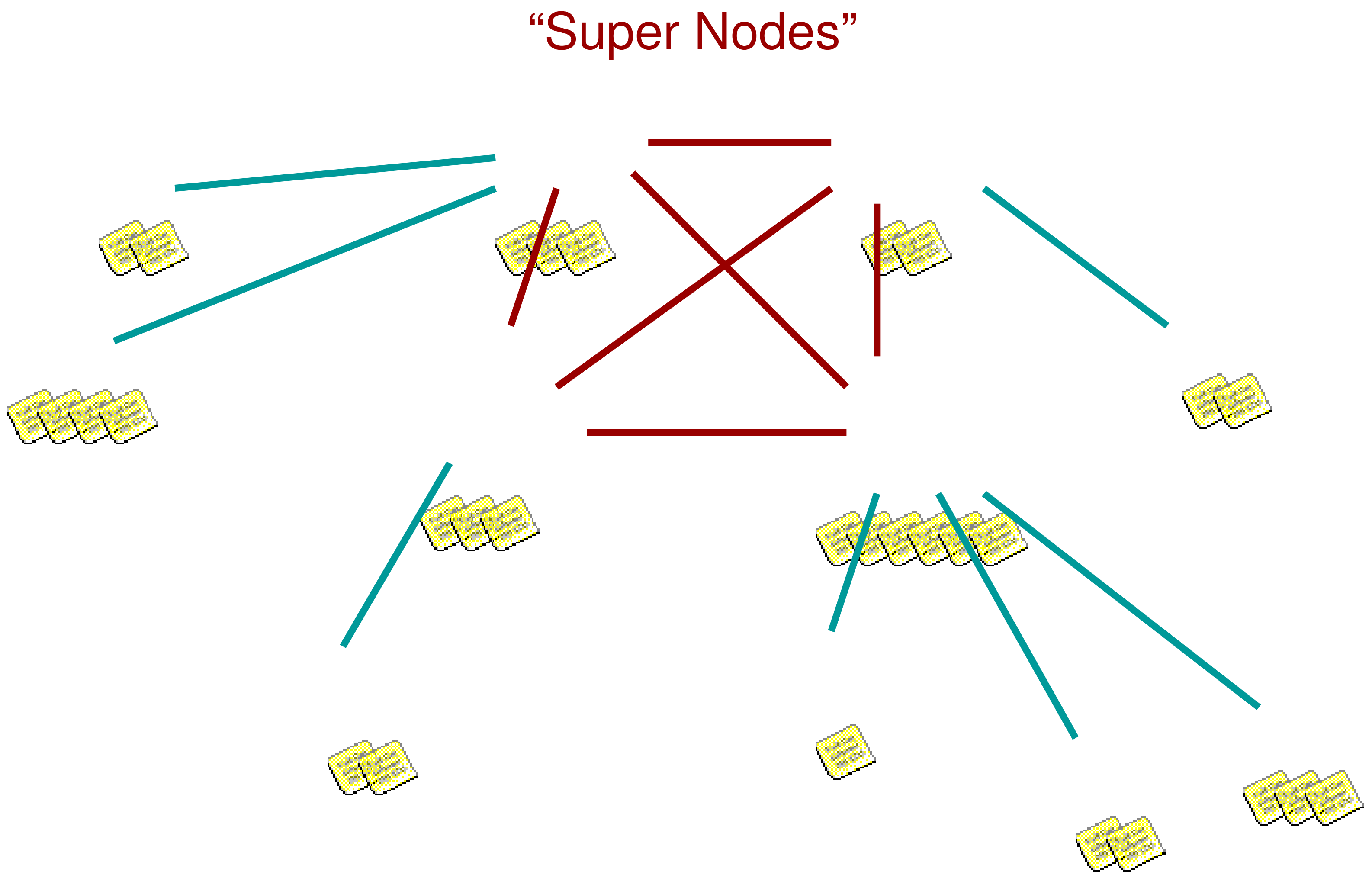
FastTrack

- A supernode stores a directory listing a subset of nearby (<filename,peer pointer>), similar to Napster servers
- Supernode membership changes over time
- Any peer can become (and stay) a supernode, provided it has earned enough **reputation**
  - Kazaalite: participation level (=reputation) of a user between 0 and 1000, initially 10, then affected by length of periods of connectivity and total number of uploads
  - More sophisticated Reputation schemes invented, especially based on economics (See P2PEcon workshop)
- A peer searches by contacting a nearby supernode

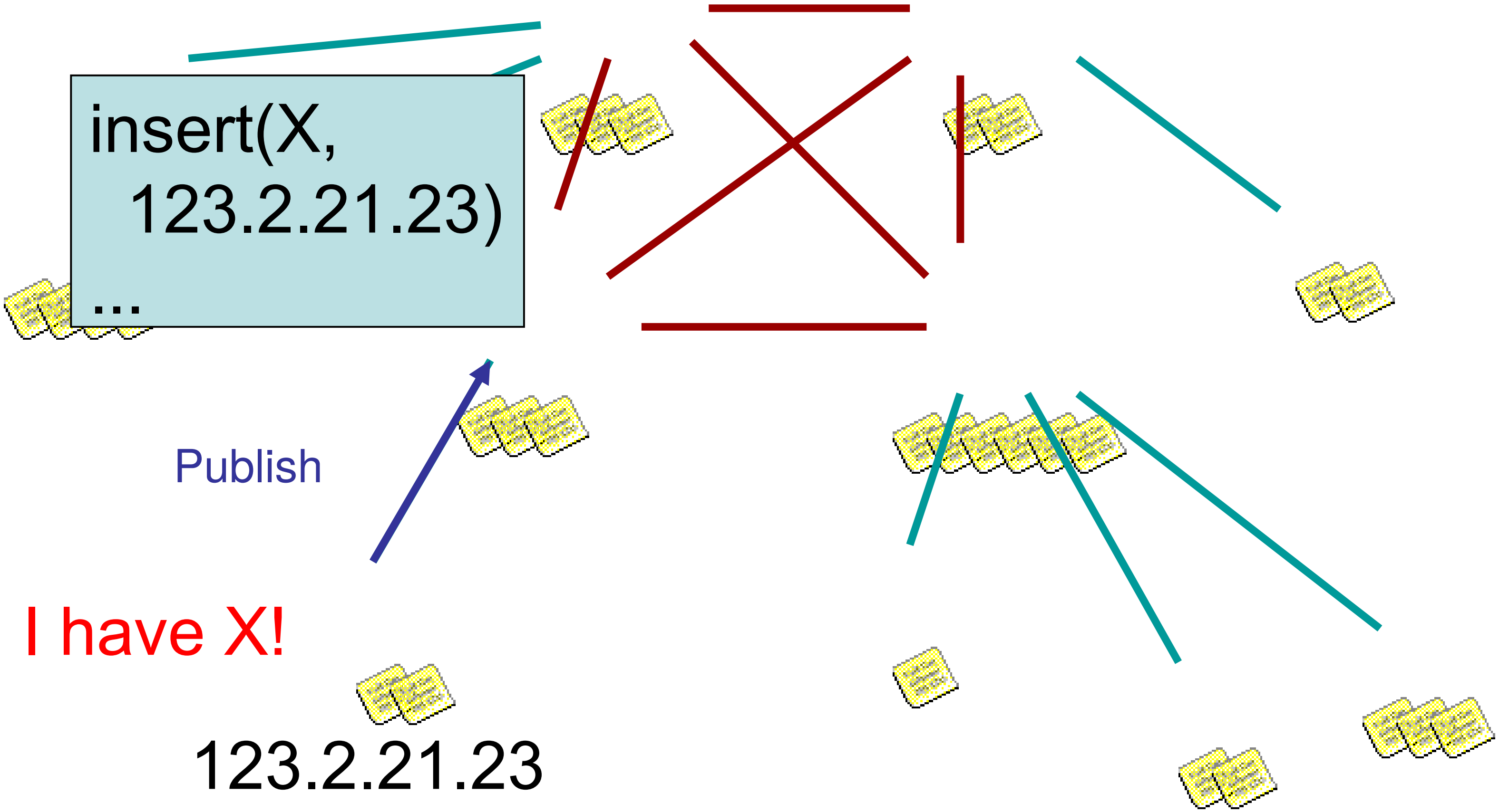
KaZaA: Overview

- "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

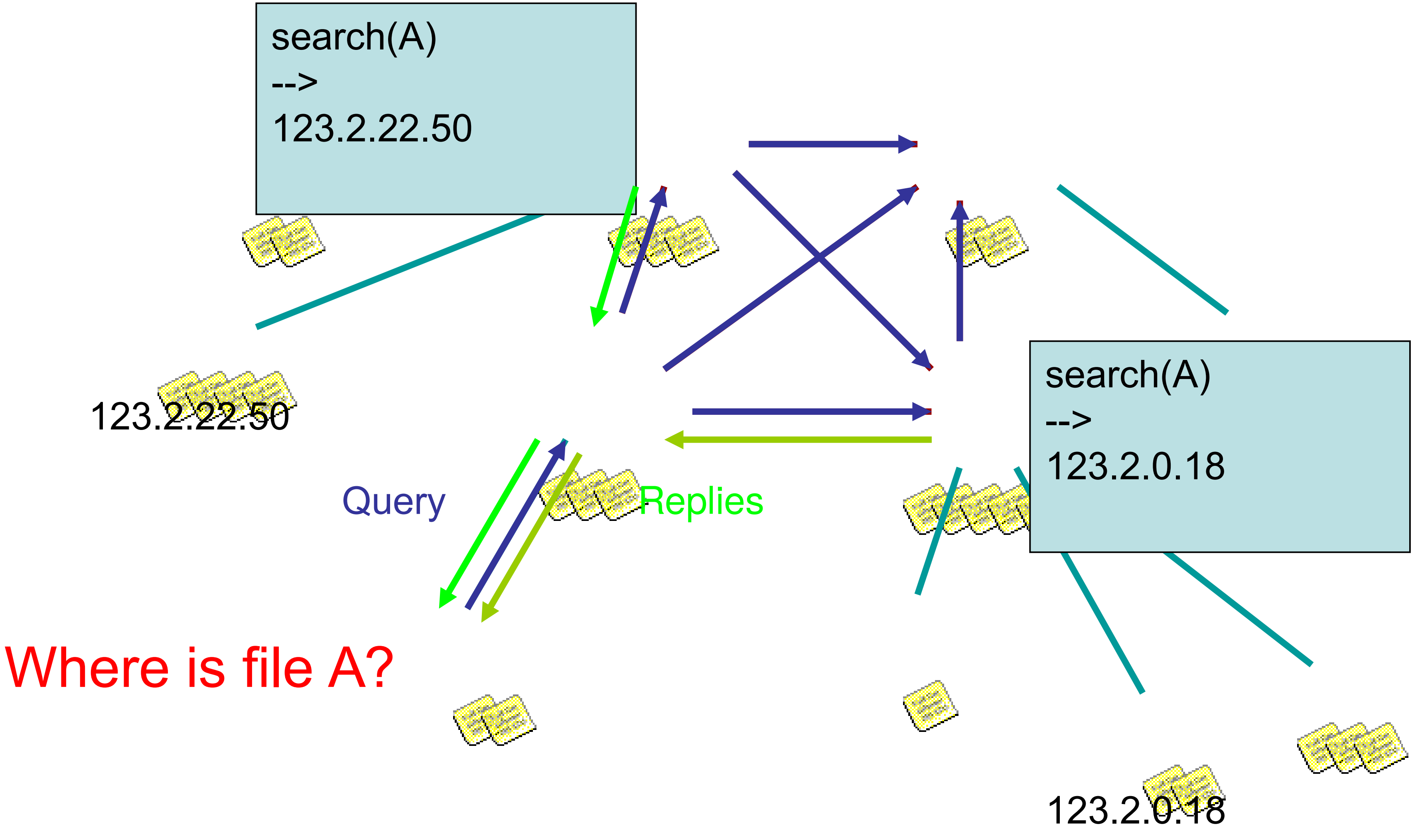
KaZaA: Network Design



KaZaA: File Insert



KaZaA: File Search



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
  - Alternative: Erasure Codes

KaZaA: Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
- Cons:
  - Mechanisms easy to circumvent
  - Still no real guarantees on search scope or search time
  - Similar behavior to gnutella, but better.

Stability and Superpeer

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating a query to a sub-node would take more b/w than answering it yourself
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you've been on is a good predictor of how long you'll be around.

We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

BitTorrent: History

- 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
  - Single publisher, multiple downloaders
- Has some “real” publishers:
  - Blizzard Entertainment using it to distribute the beta of their new game

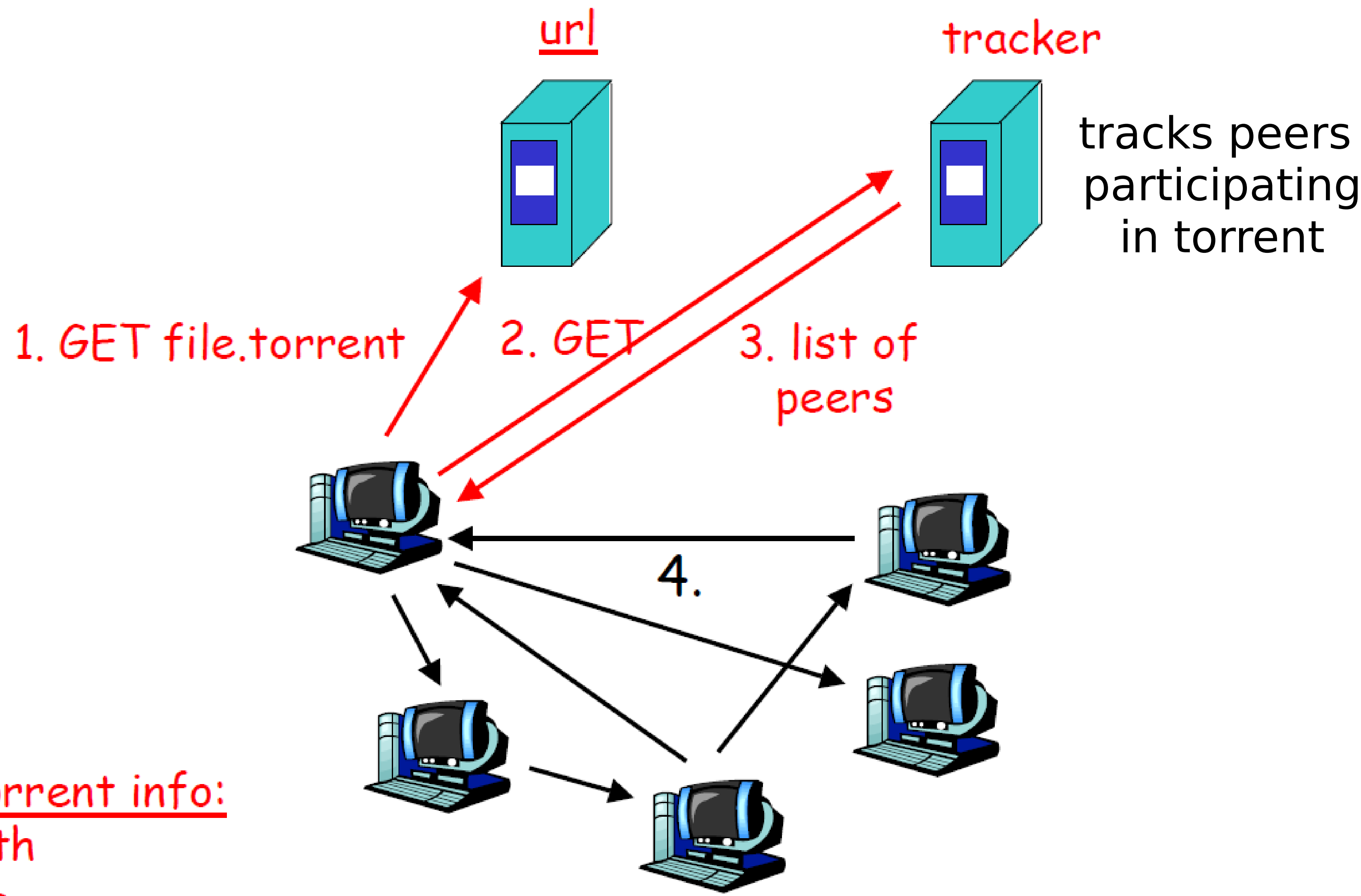
BitTorrent: Overview

- 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: Overview

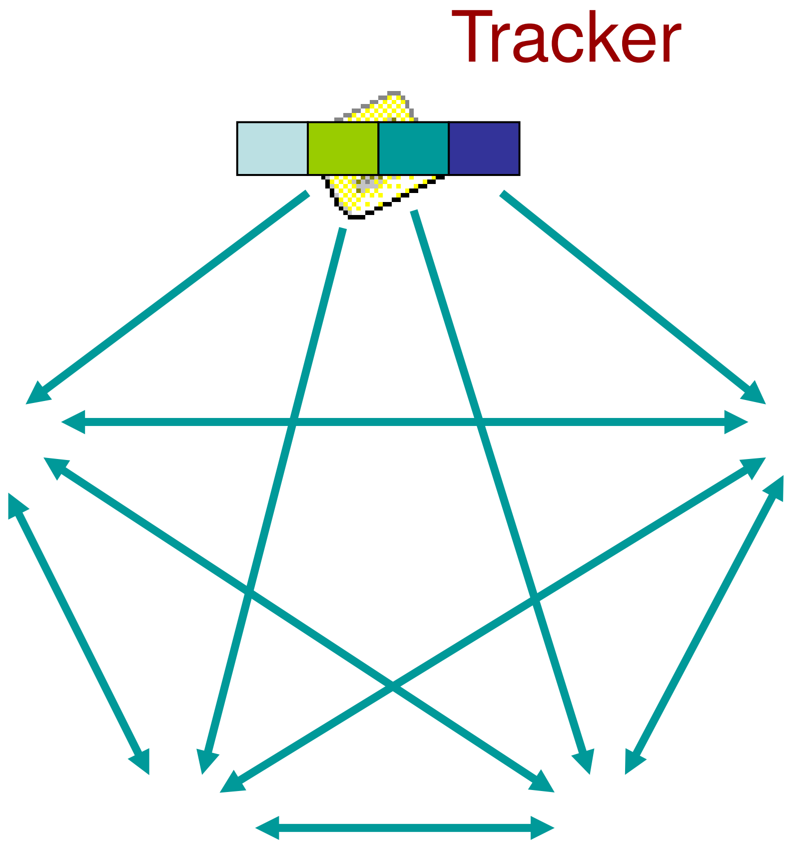
- Big differences from Napster:
  - Chunk based downloading (~256KB/chunk)
  - “few large files” focus
  - Anti-freeloading mechanisms (tit-for-tat)

BitTorrent: Overview

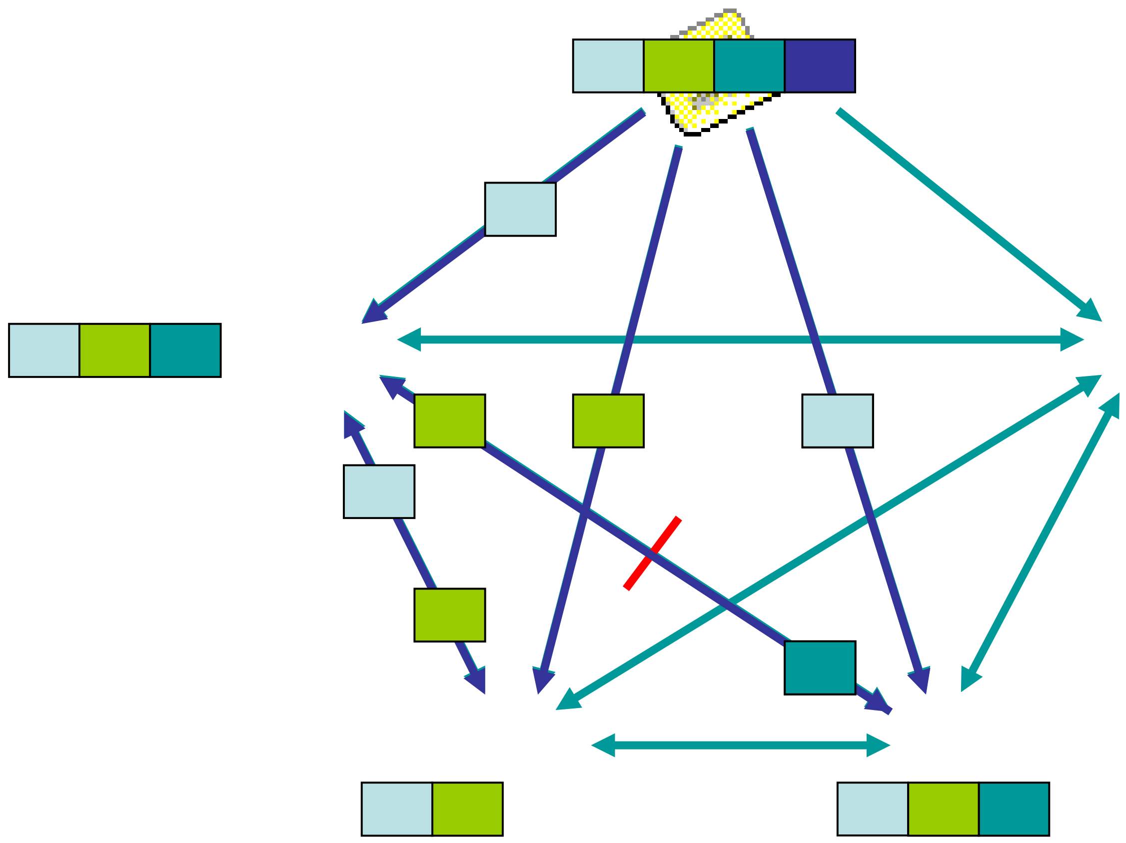


file.torrent info:
- length
- name
- hash
- url of tracker

BitTorrent: Publish/Join



BitTorrent: Fetch



One node asks herself for 2 questions:

- First, which chunks should she request first from her neighbors?
- Second, to which of her neighbors should she send requested chunks?

What to get?

- **Rarest first**
- Determine from among the chunks she does not have the
  - chunks that are the rarest among her neighbors
  - the chunks that have the fewest repeated copies among her neighbors
- Request those rarest chunks first.
- The rarest chunks get more quickly redistributed

Where to give?

- Gives priority to the neighbors that are currently supplying one data at the highest rate.
- Node continually measures the rate at which she receives bits
  - determines the four peers that are feeding at the highest rate.
  - then reciprocates by sending chunks to these same four peers.
  - recalculate every 10 seconds
  - every 30 seconds, she also picks one additional neighbor at random and sends it chunks

Where to give?

- “Tit-for-tat” sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - 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

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Central tracker server needed to bootstrap swarm
  - (Tracker is a design choice, not a requirement. Could easily combine with other approaches.)
  - Newer BT variants use a “distributed tracker” - a Distributed Hash Table

We're having

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000

Directed Searches

- Idea:
  - Assign particular nodes to hold particular content (or pointers to it)
  - When node wants that content, go to node that is supposed to have or know about it
- Challenges:
  - Distributed: want to distribute responsibilities among existing nodes in overlay
  - Adaptive: nodes join and leave P2P overlay
  - Distribute knowledge responsibility to joining nodes
  - Redistribute responsibility knowledge from leaving nodes

Distributed Hash Tables

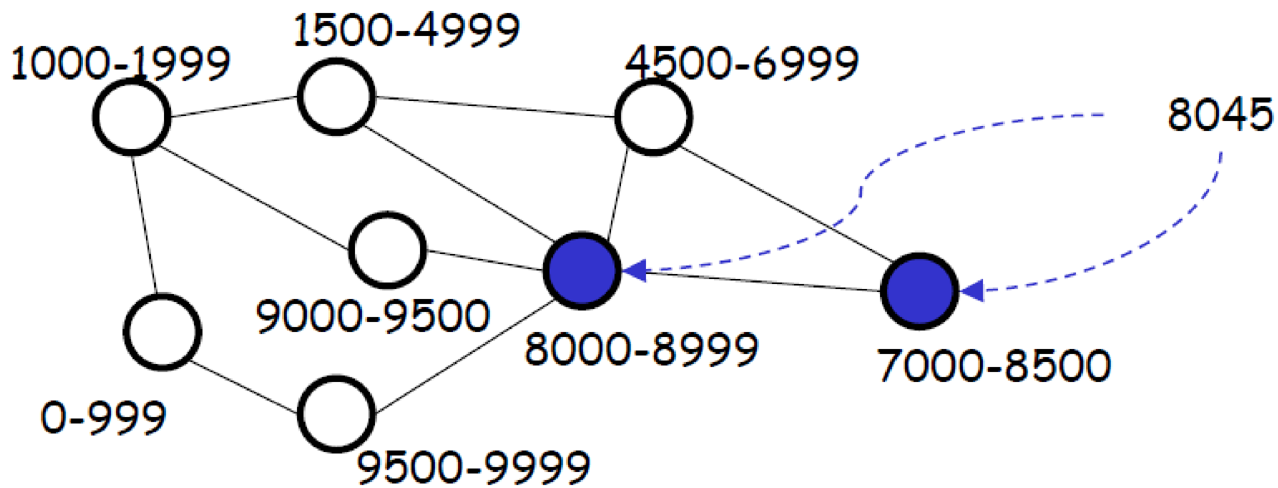
- Academic answer to p2p ;)
- Goals
  - Guaranteed lookup success
  - Provable bounds on search time
  - Provable scalability
- Read-write, not read-only
- Hot Topic in networking since introduction in ~2000/2001

DHT: Overview

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

DHT: Step 1 – The Hash

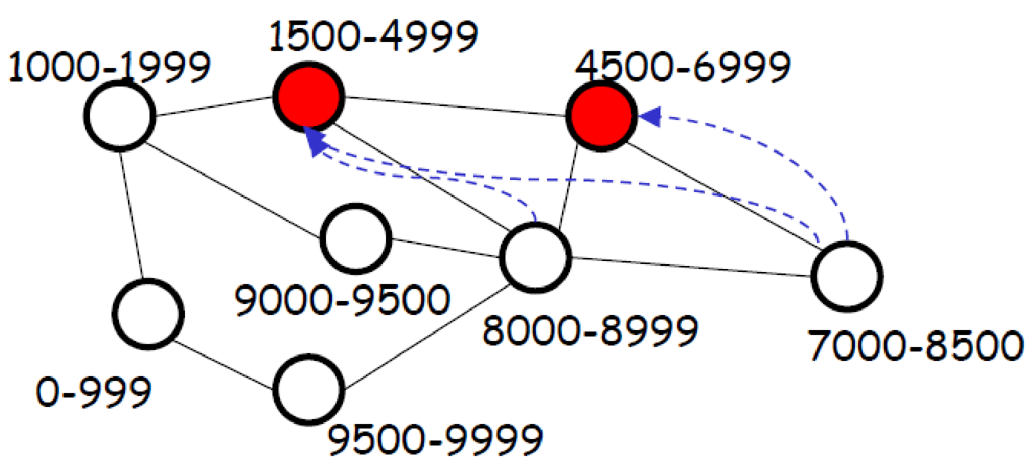
- Introduce a hash function to map the object being searched for to a unique identifier:
  - e.g., h(“Led Zepplin”) → 8045
- Distribute range of hash function among all nodes in network



- Each node must “know about” at least one copy of each object that hashes within its range (when one exists)

DHT: Knowing about Objects

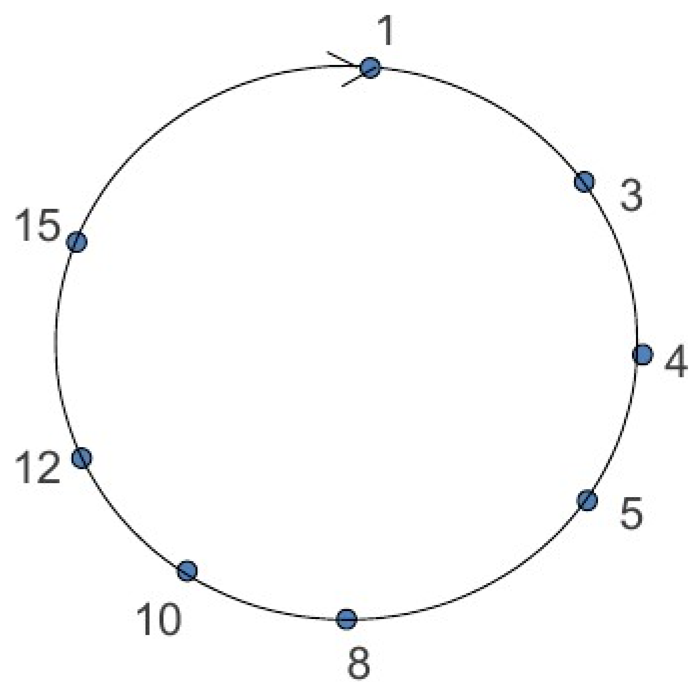
- Two alternatives
  - Node can cache each (existing) object that hashes within its range
  - Pointer-based: level of indirection – node caches pointer to location(s) of object



DHT: Step 2 – Routing

- For each object, node(s) whose range(s) cover that object must be reachable via “short” path
  - by querier node (assumed can be chosen arbitrarily)
  - by nodes that have copies of object (when pointer-based approach is used)
- Different approaches (CAN, Chord, Pastry, Tapestry) differ fundamentally only in routing approach
  - Any “good” random hash function will suffice

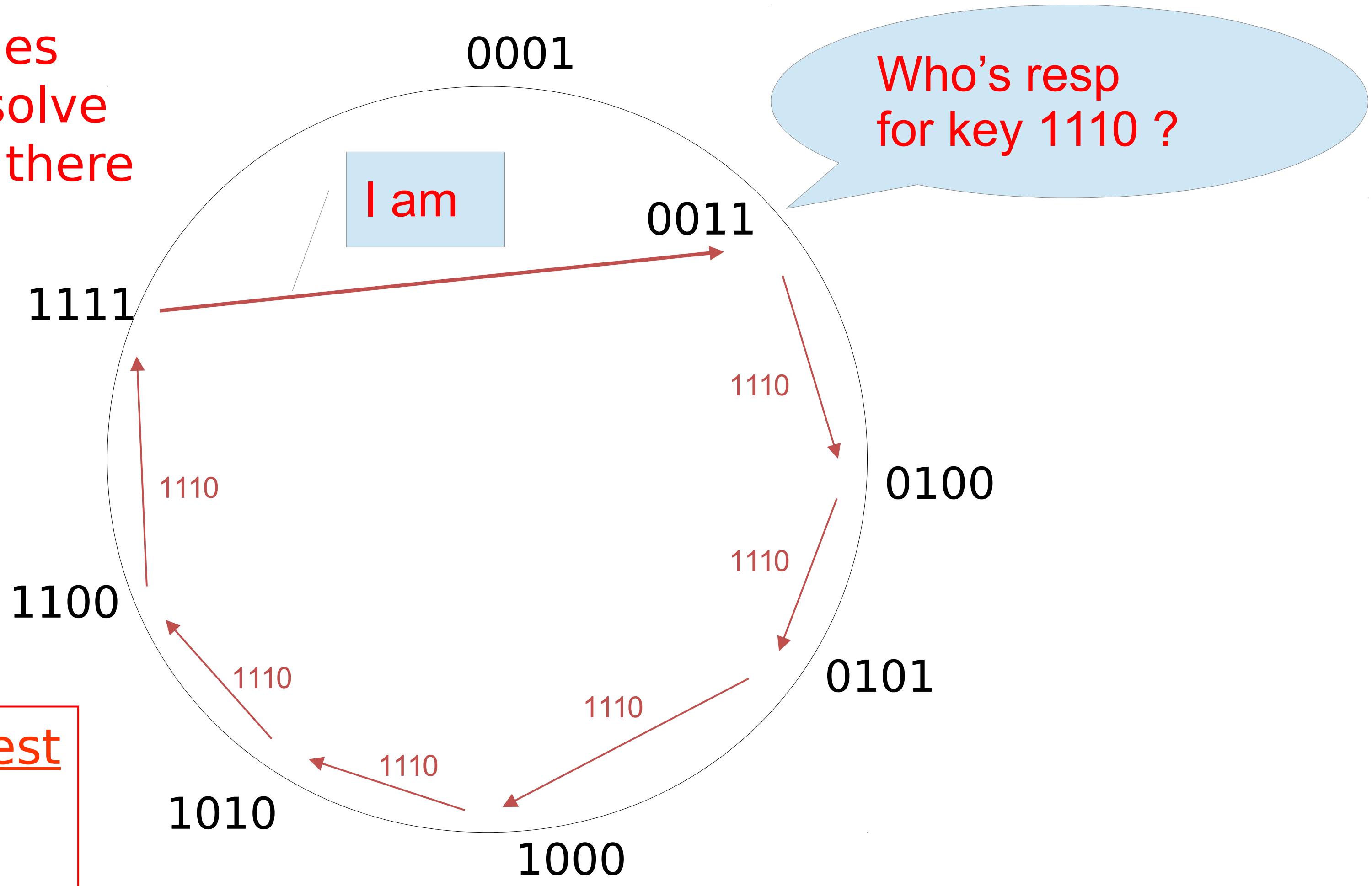
Circular DHT (1)



- Each peer only aware of immediate successor and predecessor.
- “Overlay network”

Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



Define closest as closest successor

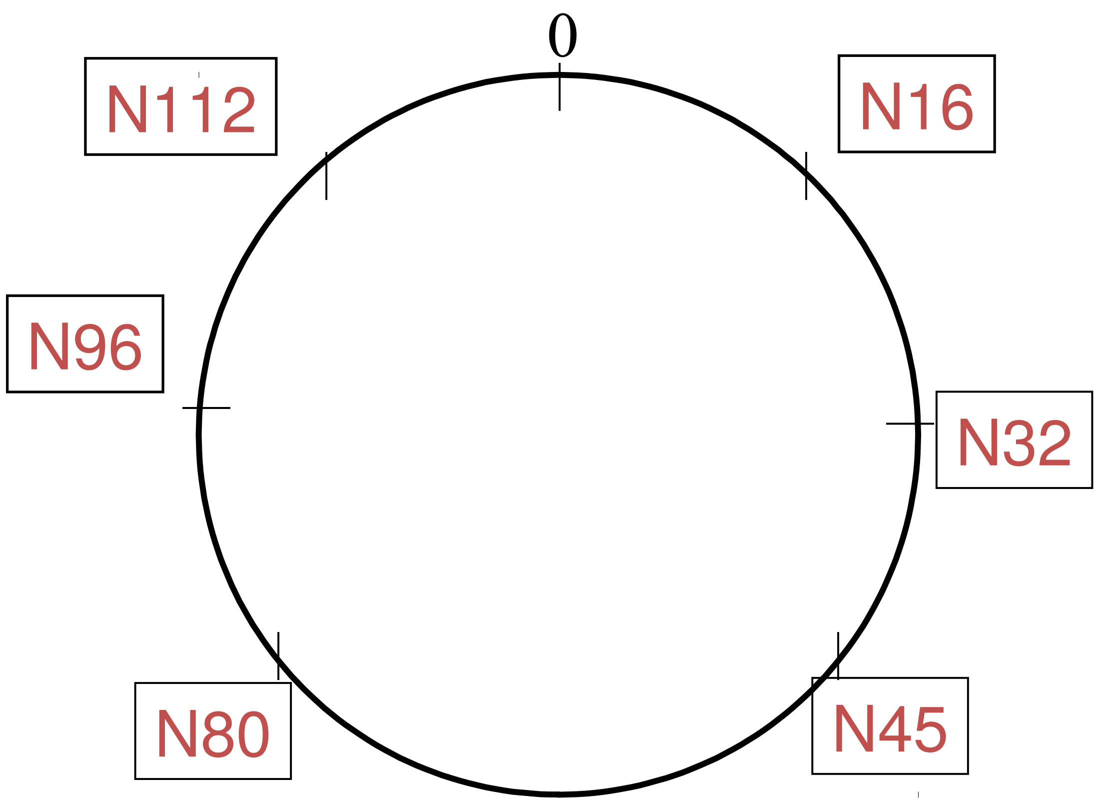
Chord

- Developers: I. Stoica, D. Karger, F. Kaashoek, H. Balakrishnan, R. Morris, Berkeley and MIT
- Intelligent choice of neighbors to reduce latency and message cost of routing (lookups/inserts)
- Uses Consistent Hashing on node's (peer's) address
- SHA-1(ip_address, port) → 160 bit string
  - Truncated to m bits
  - Called peer id (number between 0 and 2^m - 1)
  - Not unique but id conflicts very unlikely
  - Can then map peers to one of 2^m logical points on a circle

Ring of peers

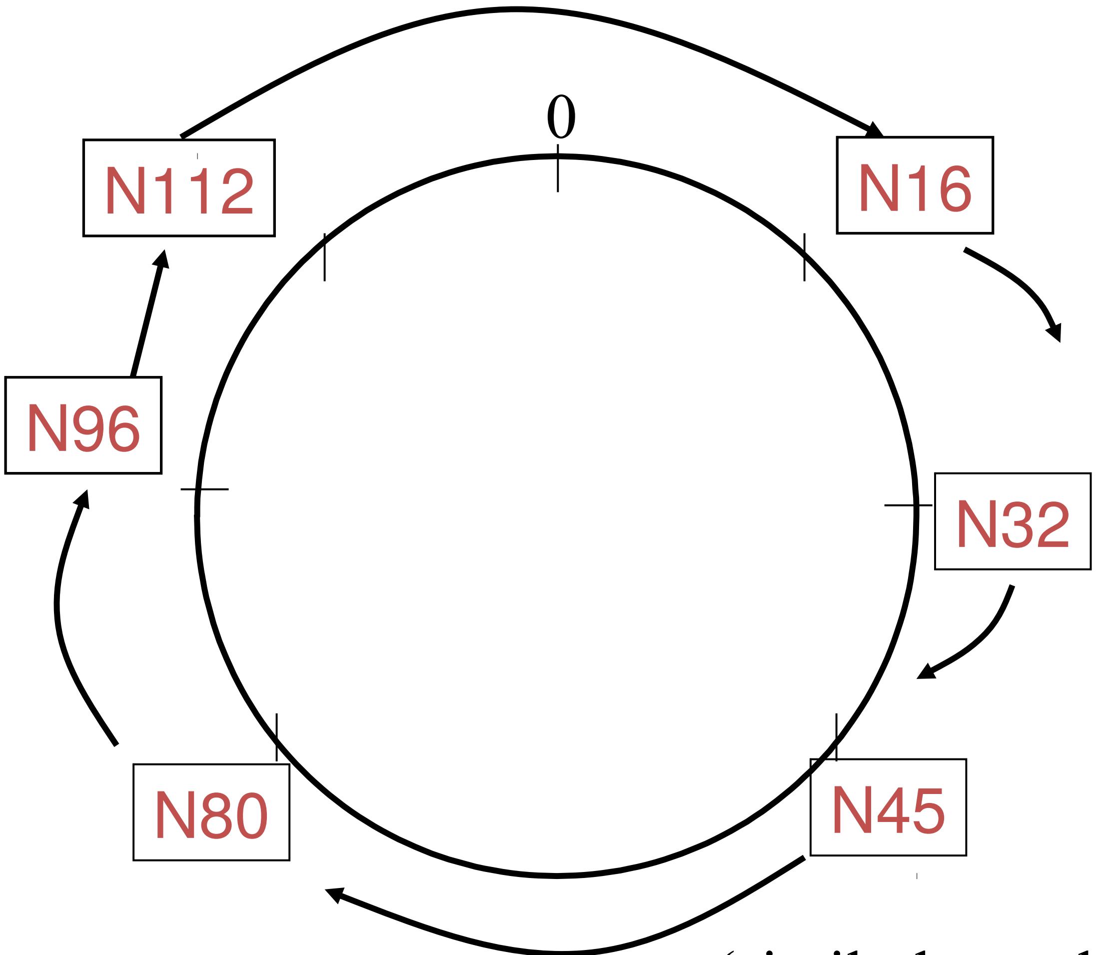
Say $m=7$

6 nodes



Peer pointers (1): successors

Say m=7



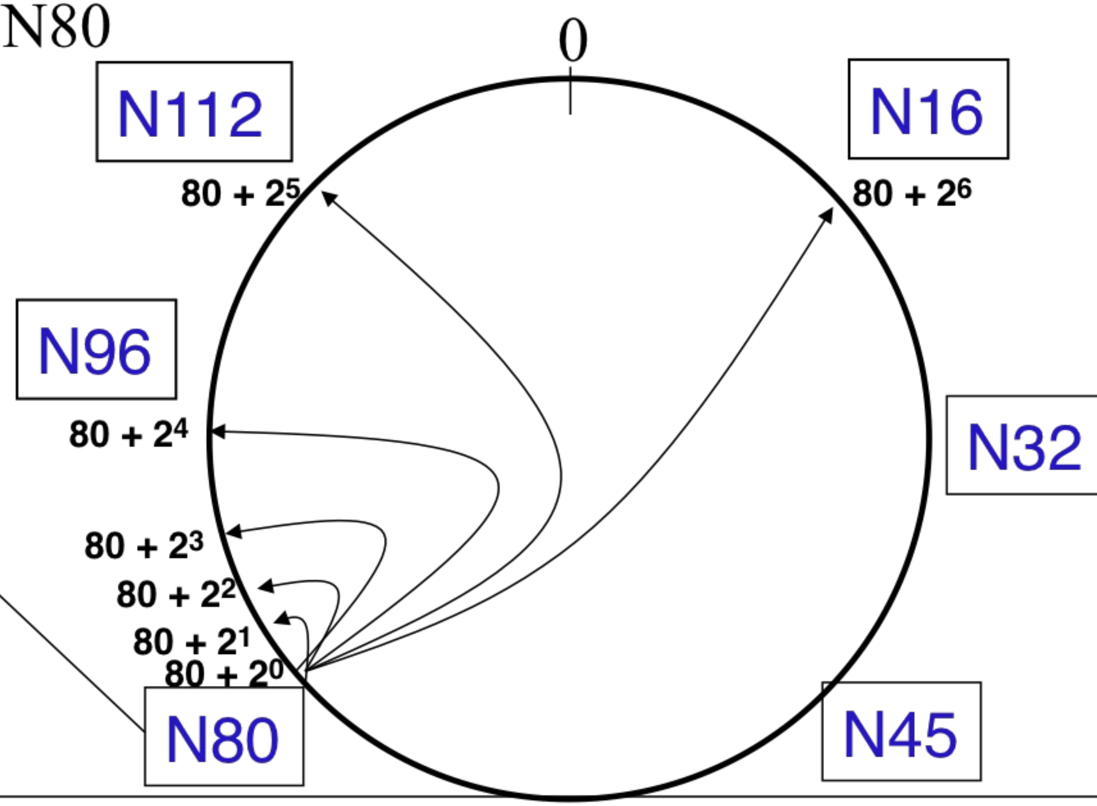
(similarly predecessors)

Peer pointers (2): finger tables

Say m=7

Finger Table at N80

i	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	112
6	16



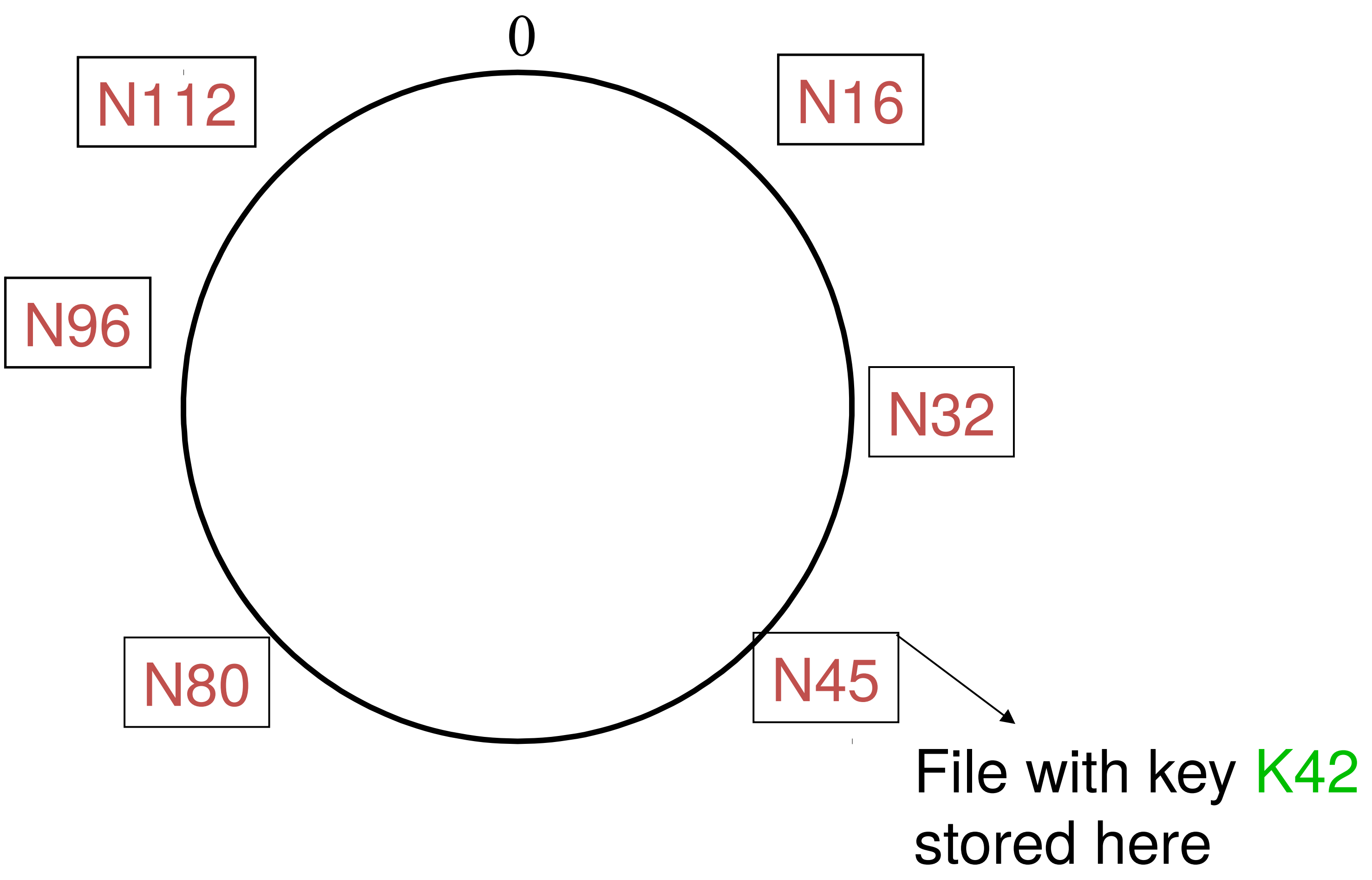
ith entry at peer with id n is first peer with id >= $n + 2^i (\mathrm{mod}\, 2^m)$

What about the file?

- Filenames also mapped using same consistent hash function
  - SHA-1(filename) → 160 bit string (key)
  - File is stored at first peer with id greater than or equal to its key (mod 2^m)
- File cnn.com/index.html that maps to key K42 is stored at first peer with id greater than 42
  - Note that we are considering a different file-sharing application here : cooperative web caching
  - The same discussion applies to any other file sharing application, including that of mp3 files.
- Consistent Hashing => with K keys and N peers, each peer stores O(K/N) keys. (i.e., < c.K/N, for some constant c)

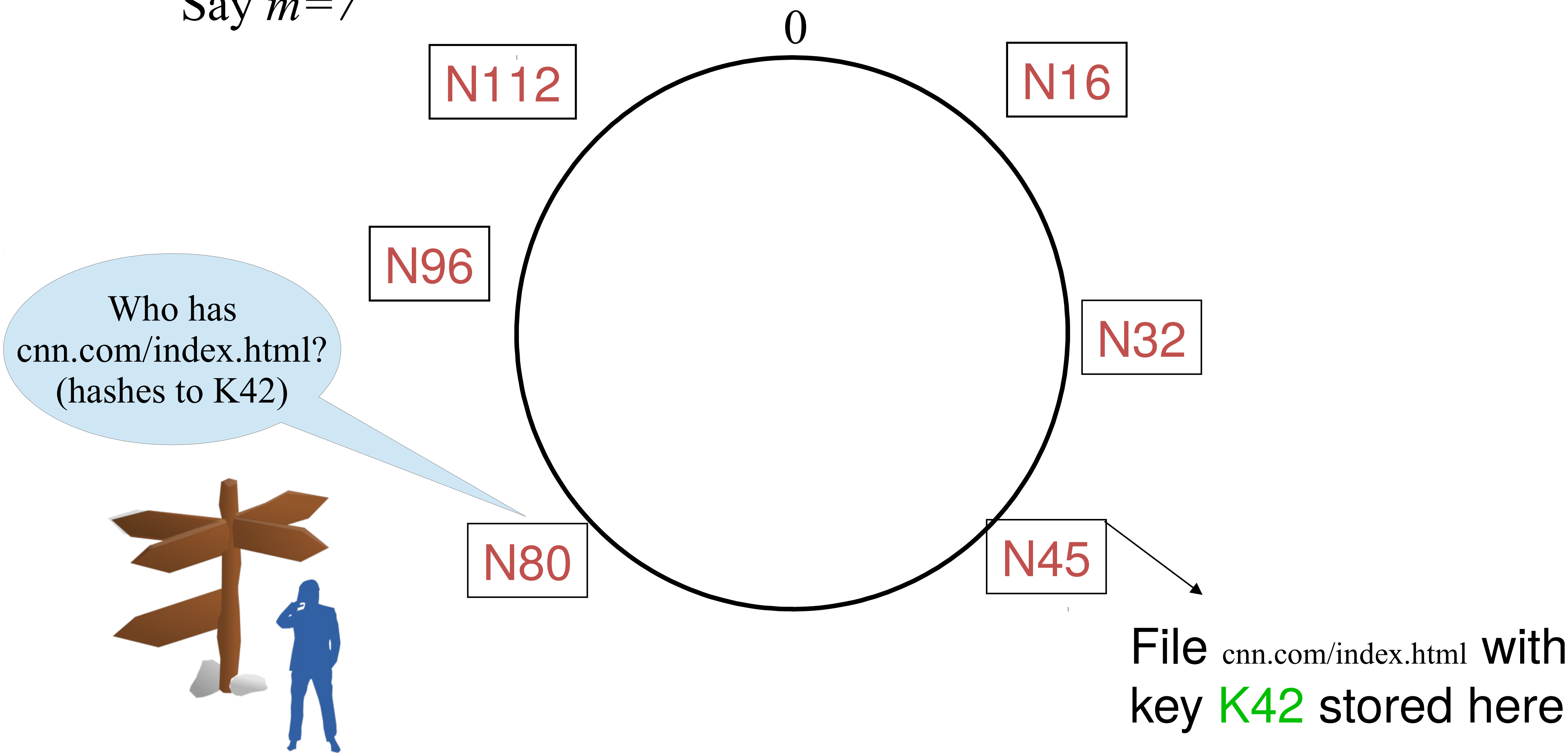
Mapping File

Say m=7



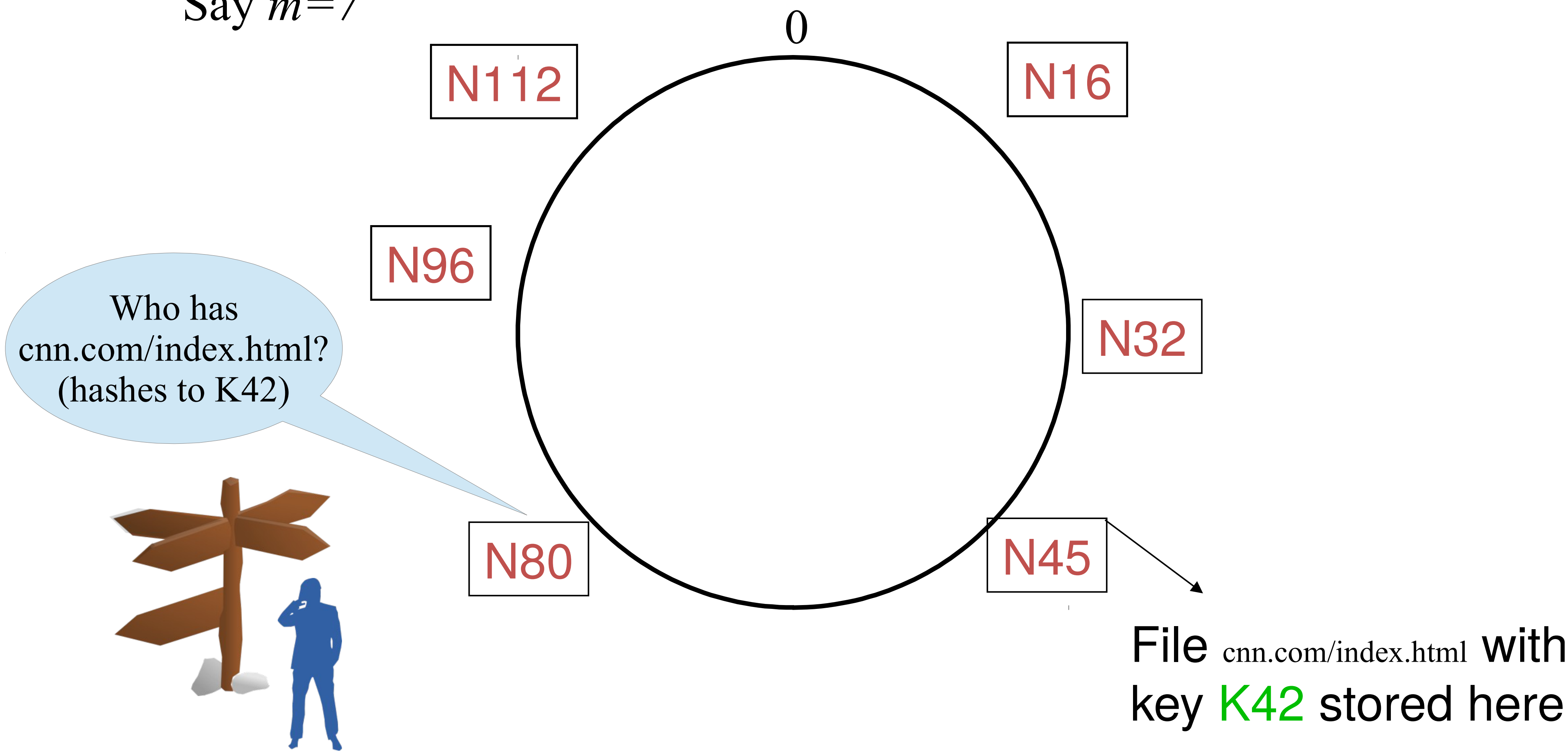
Search

Say m=7



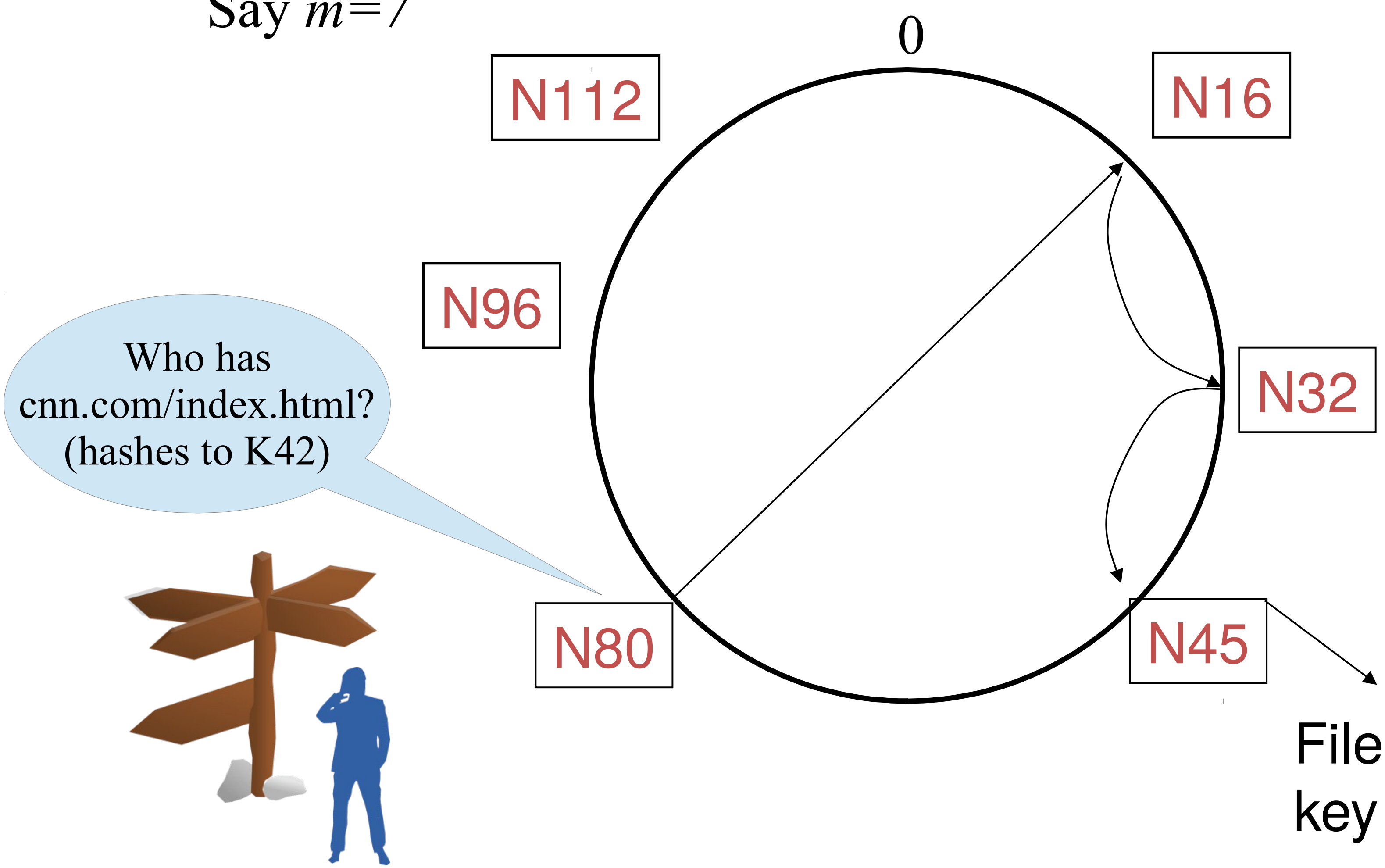
At node n, send query for key k to largest successor/finger entry <= k if none exist, send query to *successor(n)*

Say $m=7$



At node *n*, send query for key *k* to largest successor/finger entry <= *k* if none exist, send query to *successor(n)*

Say *m=7*



File cnn.com/index.html with key K42 stored here

Analysis

- Search takes O(log(N)) time
- Proof
  - (intuition): at each step, distance between query and peer-with-file reduces by a factor of at least 2
  - (intuition): after log(N) forwardings, distance to key is at most 2^m / 2^log(N) = 2^m / N
- Number of node identifiers in a range of is O(log(N)) with high probability (why? SHA-1). So using successors in that range will be ok, using another O(log(N)) hops

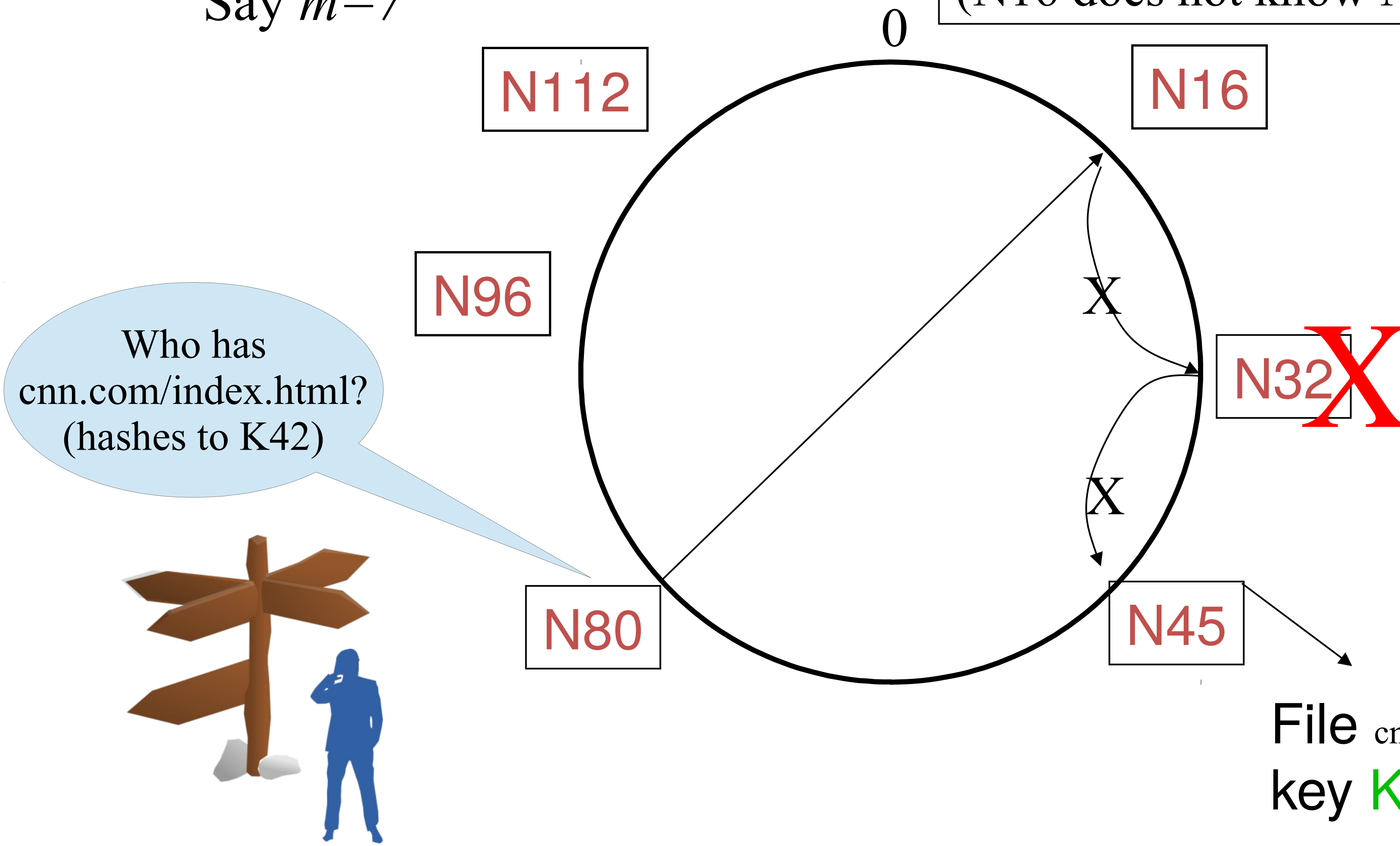
Analysis

- O(log(N)) search time holds for file insertions too (in general for routing to any key)
  - “Routing” can thus be used as a building block for all operations: insert, lookup, delete
- O(log(N)) time true only if finger and successor entries correct
- When might these entries be wrong?
  - When you have failures

Search under peer failures

Lookup fails
(N16 does not know N45)

Say m=7

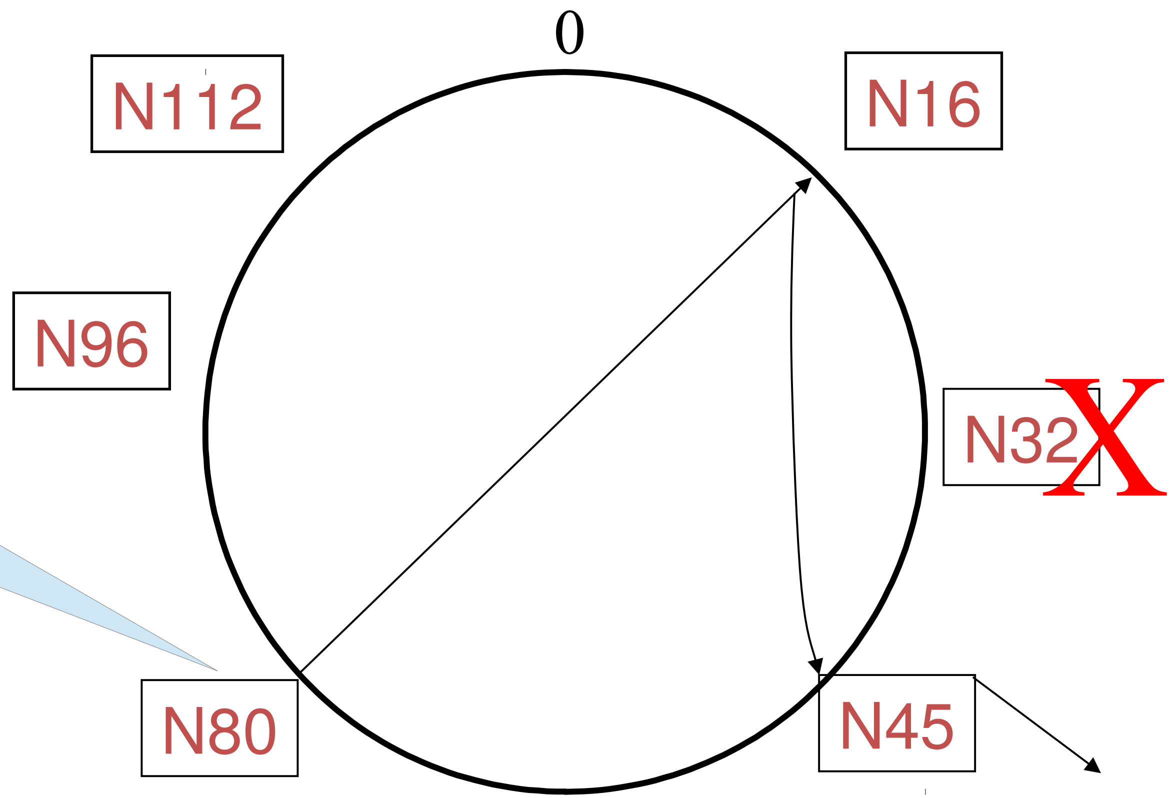


File cnn.com/index.html with key K42 stored here

One solution: maintain r multiple *successor* entries. In case of failure, use successor entries

Say m=7

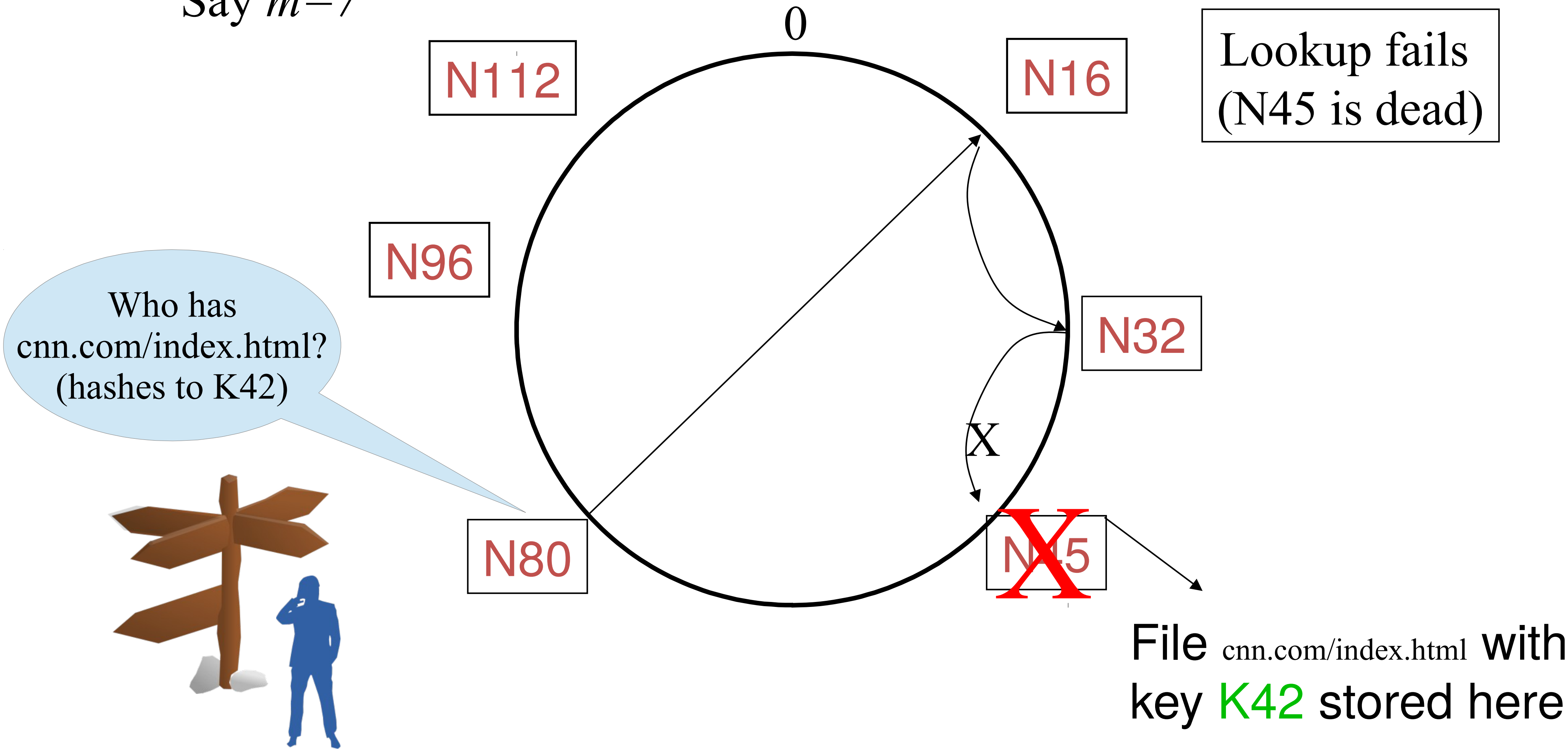
Who has cnn.com/index.html? (hashes to K42)



File cnn.com/index.html with key K42 stored here

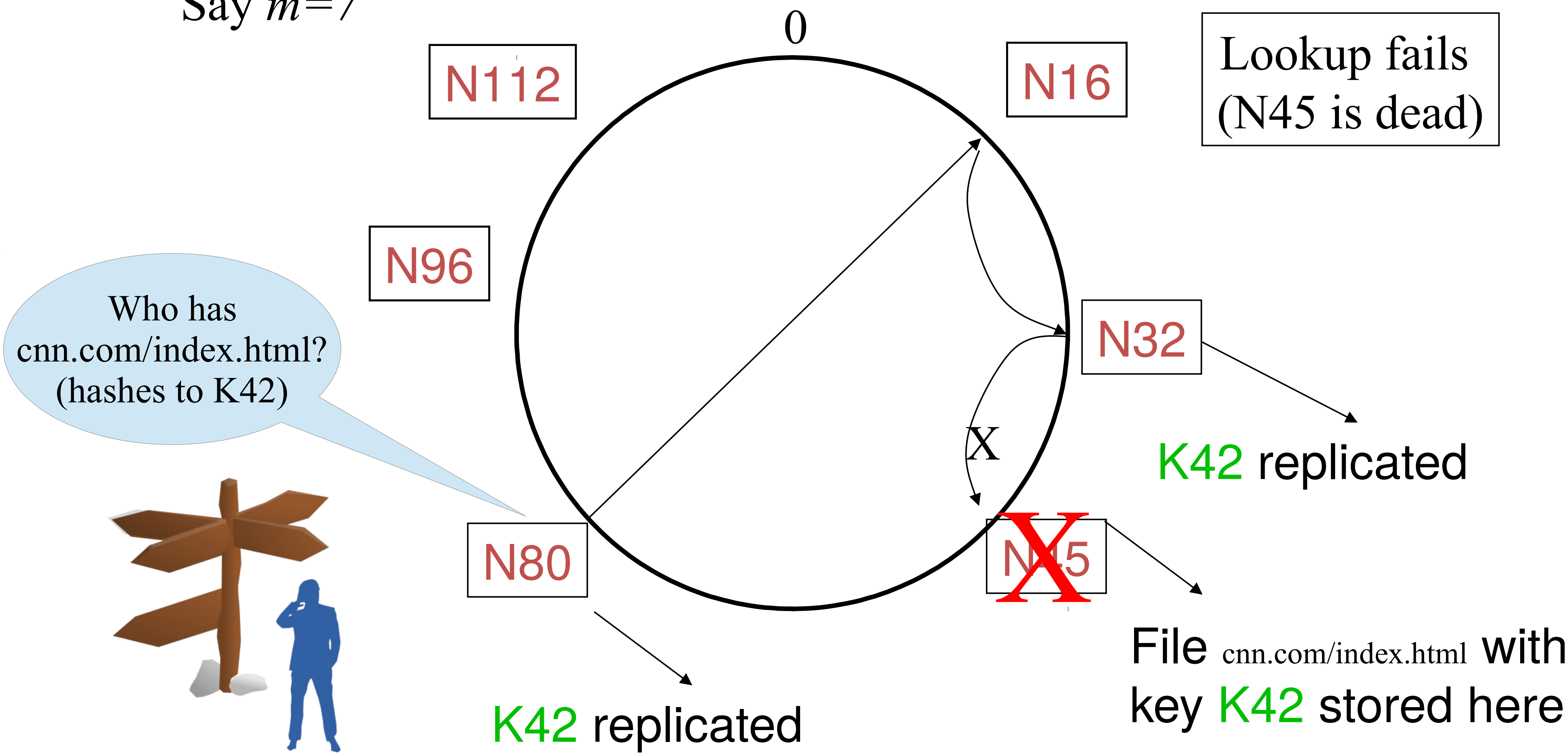
Search under peer failures (2)

Say m=7



One solution: replicate file/key at *r* successors and predecessors

Say m=7



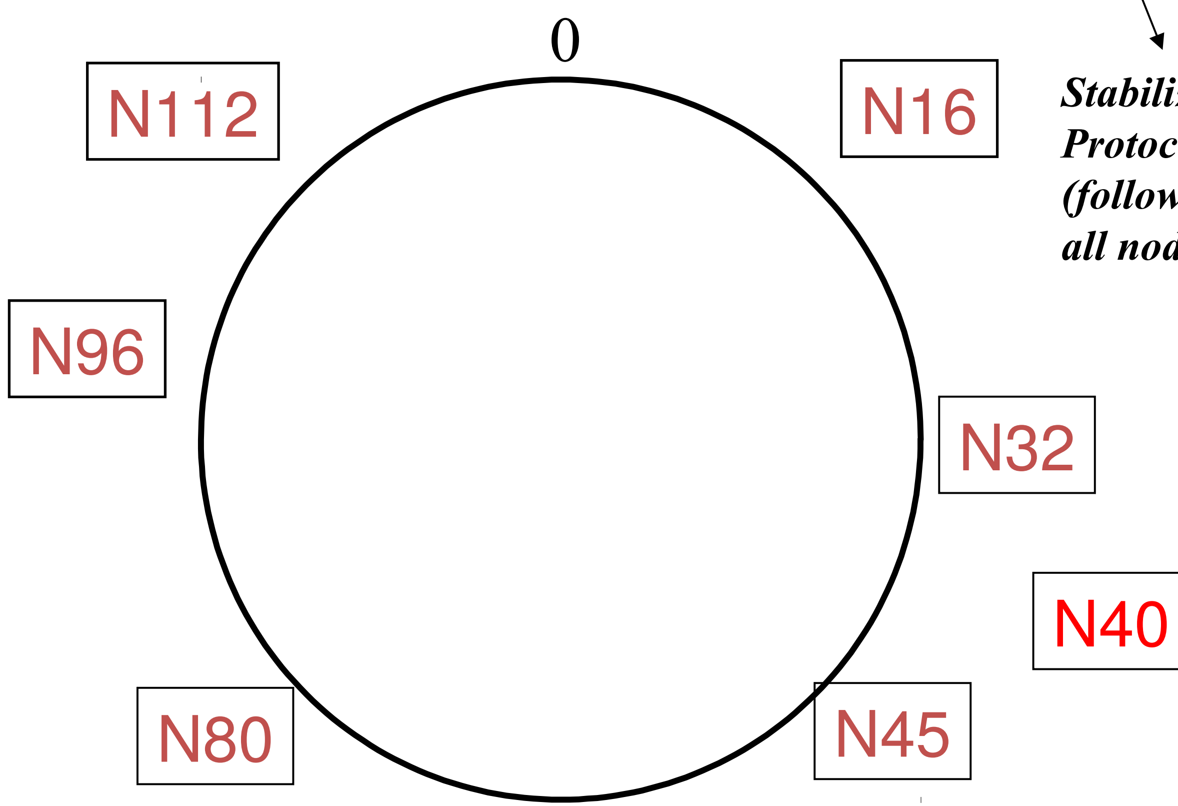
Need to deal with dynamic changes

- Peers fail
- New peers join
- Peers leave
  - P2P systems have a high rate of churn (node join, leave and failure)
  - 25% per hour in Overnet (eDonkey)
  - 100% per hour in Gnutella
  - Lower in managed clusters
  - Common feature in all distributed systems, including wide-area (e.g., PlanetLab), clusters (e.g., Emulab), clouds (e.g., AWS), etc.
- So, all the time, need to update successors and fingers, and copy keys

New Peers Joining

Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it
N40 periodically talks to neighbors to update finger table

Say *m*=7

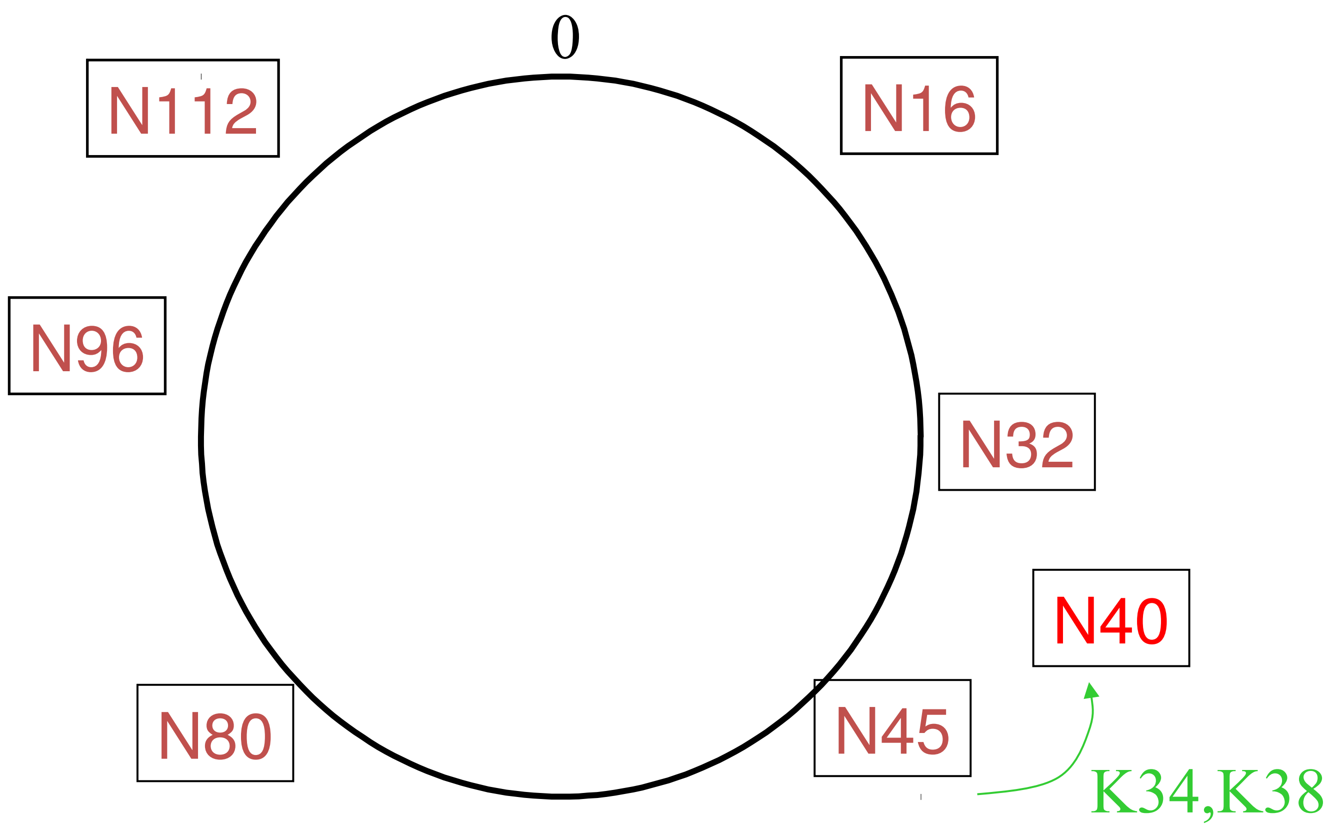


Stabilization Protocol (followed by all nodes)

New Peers Joining (2)

N40 may need to copy some files/keys from N45
(files with fileid between 32 and 40)

Say m=7



New Peers Joining (3)

- A new peer affects O(log(N)) other finger entries in the system, on average.
- Number of messages per peer join= O(log(N)*log(N))
- Similar set of operations for dealing with peers leaving
  - For dealing with failures, also need failure detectors

Stabilization Protocol

- Concurrent peer joins, leaves, failures might cause loopiness of pointers, and failure of lookups
  - Chord peers periodically run a stabilization algorithm that checks and updates pointers and keys
  - Ensures non-loopiness of fingers, eventual success of lookups and O(log(N)) lookups w.h.p.
  - Each stabilization round at a peer involves a constant number of messages
  - Strong stability takes O(N^2) stabilization rounds
  - For more see [TechReport on Chord webpage]

Chunk

When nodes are constantly joining, leaving, failing

- Significant effect to consider: traces from the Overnet system show hourly peer turnover rates (churn) could be 25-100% of total number of nodes in system
- Leads to excessive (unnecessary) key copying (remember that keys are replicated)
- Stabilization algorithm may need to consume more bandwidth to keep up
- Main issue is that files are replicated, while it might be sufficient to replicate only meta information about files
- Alternatives
  - Introduce a level of indirection (any p2p system)
  - Replicate metadata more, e.g., Kelips

Virtual Node

- Hash can get non-uniform → Bad load balancing
- Treat each node as multiple virtual nodes behaving independently
- Each joins the system
- Reduces variance of load imbalance

Chord Wrap-up Notes

- Virtual Ring and Consistent Hashing used in Cassandra, Riak, Voldemort, DynamoDB, and other key-value stores

- Current status of Chord project:
  - File systems (CFS,Ivy) built on top of Chord
  - DNS lookup service built on top of Chord
  - Internet Indirection Infrastructure (I3) project at UCB
  - Spawned research on many interesting issues about p2p systems

http://www.pdos.lcs.mit.edu/chord/

Pastry

- Designed by Anthony Rowstron (Microsoft Research) and Peter Druschel (Rice University)
- Assigns ids to nodes, just like Chord (using a virtual ring)
- Leaf Set - Each node knows its successor(s) and predecessor(s)

Pastry Neighbors

- Routing tables based prefix matching
  - Think of a hypercube

- Routing is thus based on prefix matching, and is thus log(N)
  - And hops are short (in the underlying network)

Pastry Routing

- Consider a peer with id 01110100101. It maintains a neighbor peer with an id matching each of the following prefixes (* = bit different from this peer's corresponding bit):
  - *
  - 0*
  - 01*
  - 011*
  - … 0111010010*
- When it needs to route to a peer, say 01110111001, it starts by forwarding to a neighbor with the largest matching prefix, i.e., 011101*

Pastry Locality

- For each prefix, say 011*, among all potential neighbors with the matching prefix, the neighbor with the shortest round-trip-time is selected
- Since shorter prefixes have many more candidates (spread out throughout the Internet), the neighbors for shorter prefixes are likely to be closer than the neighbors for longer prefixes
- Thus, in the prefix routing, early hops are short and later hops are longer
- Yet overall “stretch”, compared to direct Internet path, stays short

Summary of Chord and Pastry

- More structured than Gnutella
- Black box lookup algorithms
- Churn handling can get complex
- O(log(N)) memory and lookup cost
  - O(log(N)) lookup hops may be high

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