

CS 414 – Multimedia Systems Design
Lecture 23 – Introduction to
P2P Systems

Klara Nahrstedt

Ramsés Morales

Based on slides by Indranil
Gupta

Why P2P?

- People like to get together
 - Share things
 - Share information
- How can I find others to share?
- How can I find what they are sharing?

Why P2P?

■ WWW to share?

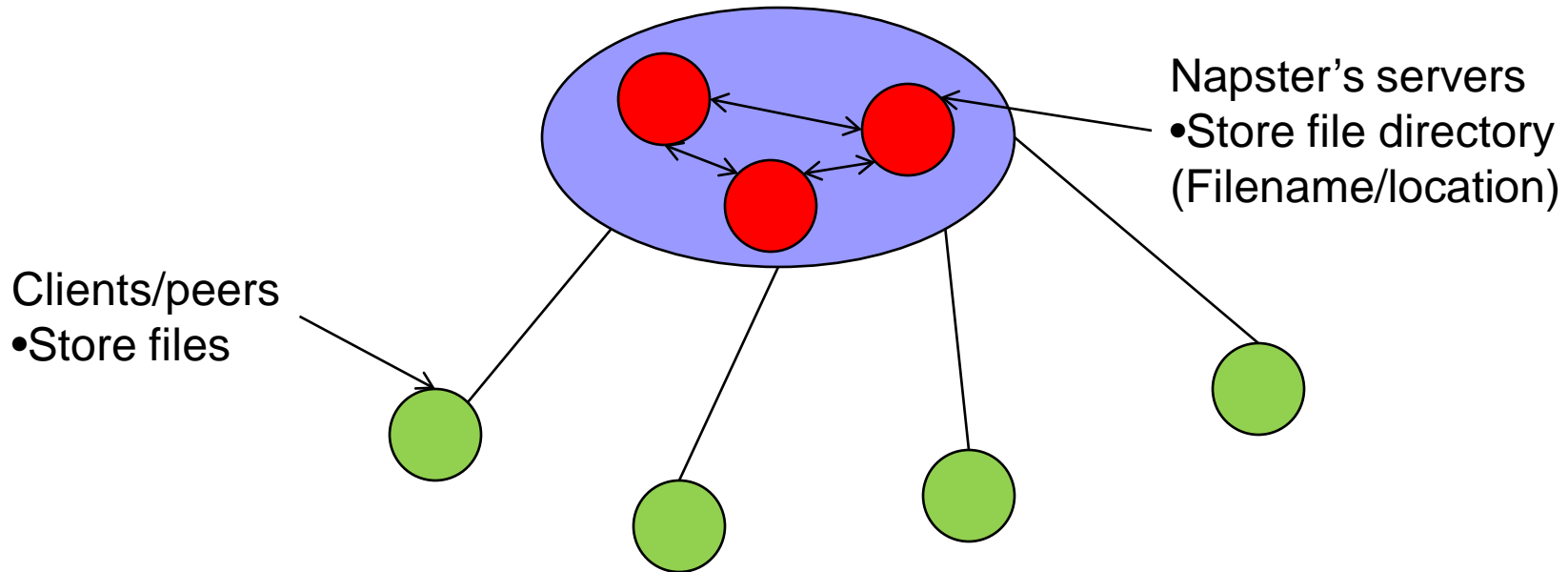
- Not everyone can set up and maintain a website
- Might not have a good enough search engine rank

■ Sharing on a social network?

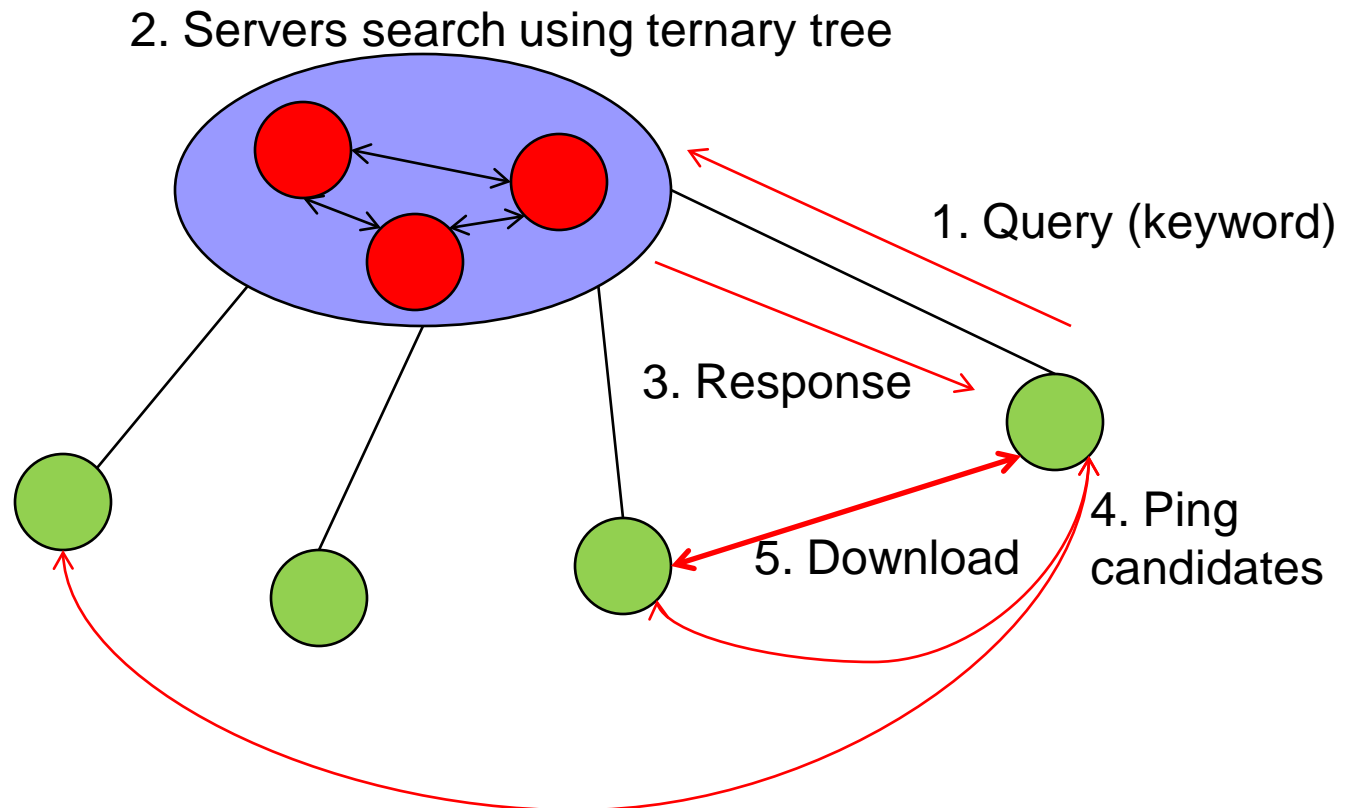
- Only possible if you created the content
- People have to find you, instead of the content

Napster

- Created in 1999 to share music



Napster, operation



Napster's drawbacks

- Asymmetric operation – servers vs. client peers
 - Scalability and congestion issues
 - Single point of failure
- Napster responsible for abetting users's copyright violation

Contemporary P2P systems

- Symmetric operation – all nodes (peers) have the same functionality

- System naturally scales with new peers

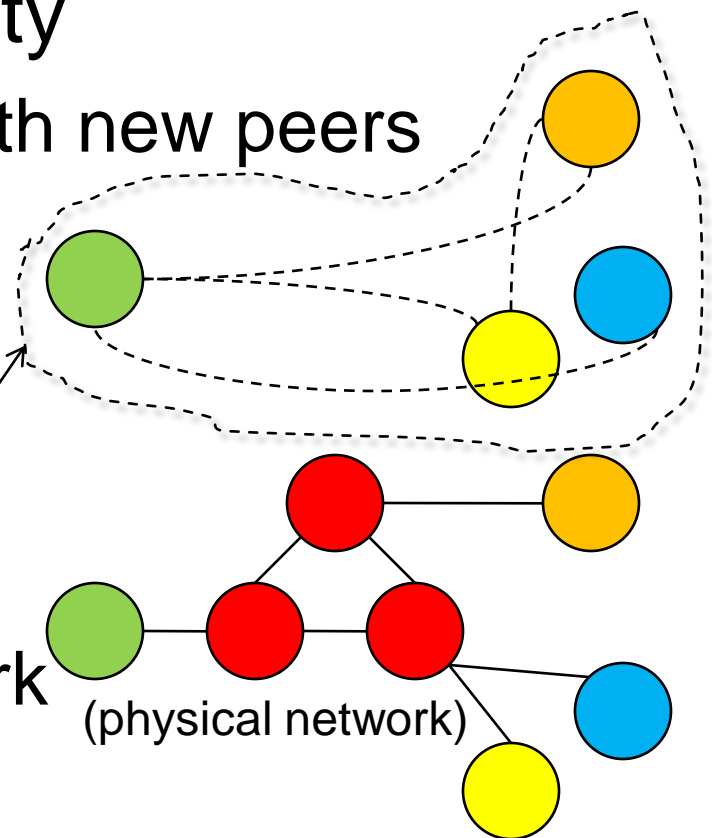
- Basic operations

- Insert file

- Search file

- Delete file

- Maintain an overlay network



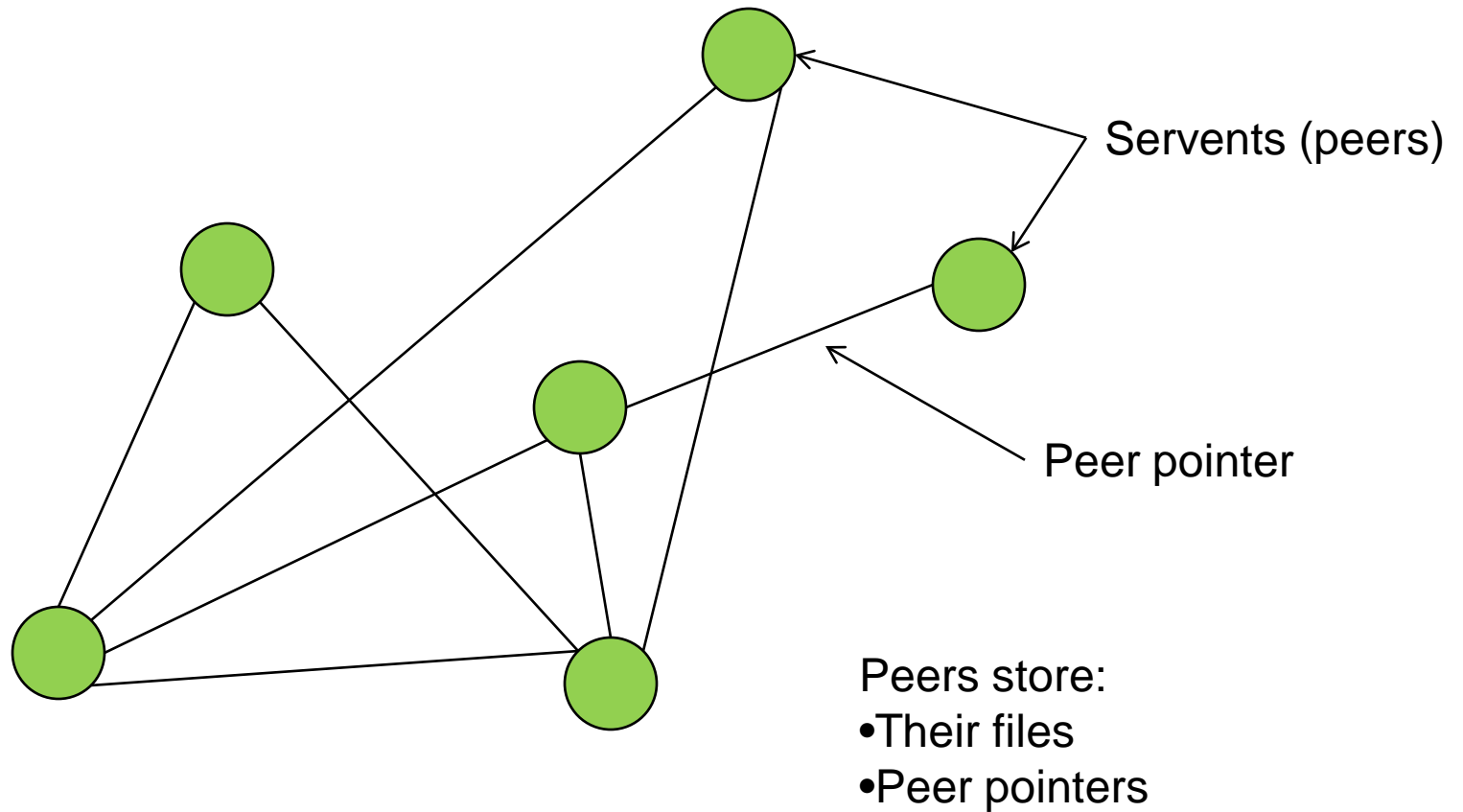
Contemporary P2P Systems (Classification)

- Usually classified depending on how peers connect to each-other – i.e., how the overlay network is created and maintained
- Unstructured – no particular connection pattern (e.g., randomly connected)
 - Gnutella
 - Fast Track
 - Skype
 - BitTorrent, etc.

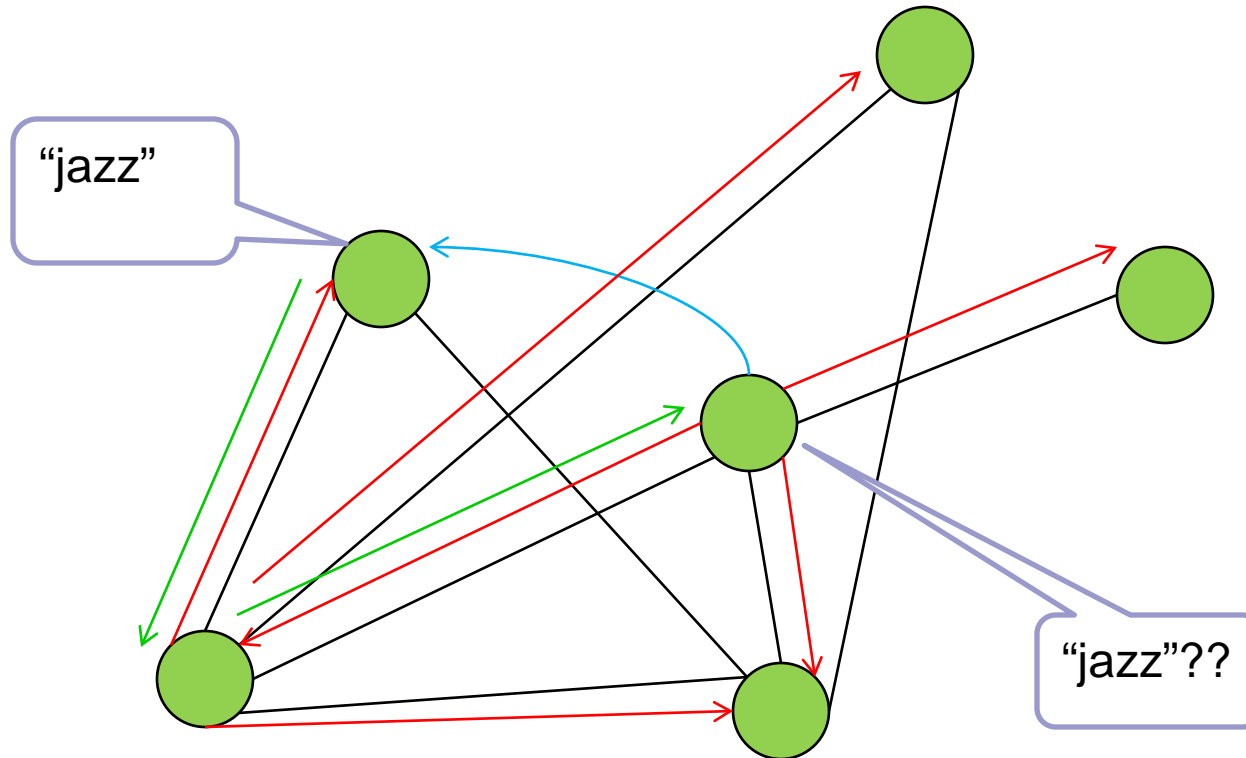
Contemporary P2P Systems (Classification)

- Structured – defines a distributed data structure (e.g., distributed hash table)
 - Chord
 - Pastry
 - CAN
 - Etc.

Gnutella



Gnutella, searching

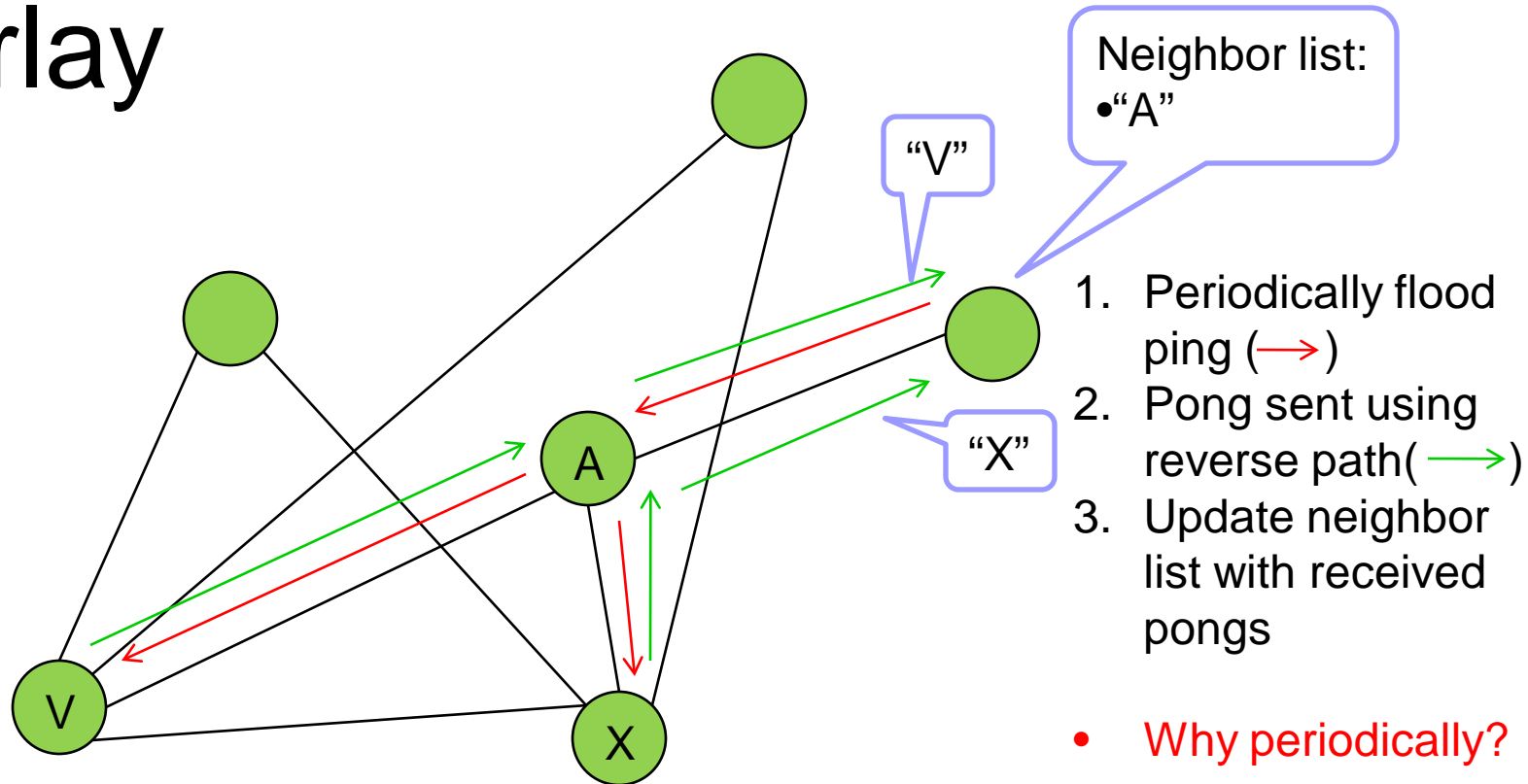


1. Flood query (→)
2. Ignore repeated messages
3. Answer if local match
4. Query hit sent using reverse path (→)
5. Establish connection and fetch file (←)

Query message: <id, QUERY, **t**tl, **h**ops, payload length, min speed, keywords>

Query hit message: <id, QUERY HIT, ttl, hops, payload length,
num hits, port, ip, speed, (fileindex, filename, filesize), servent id>

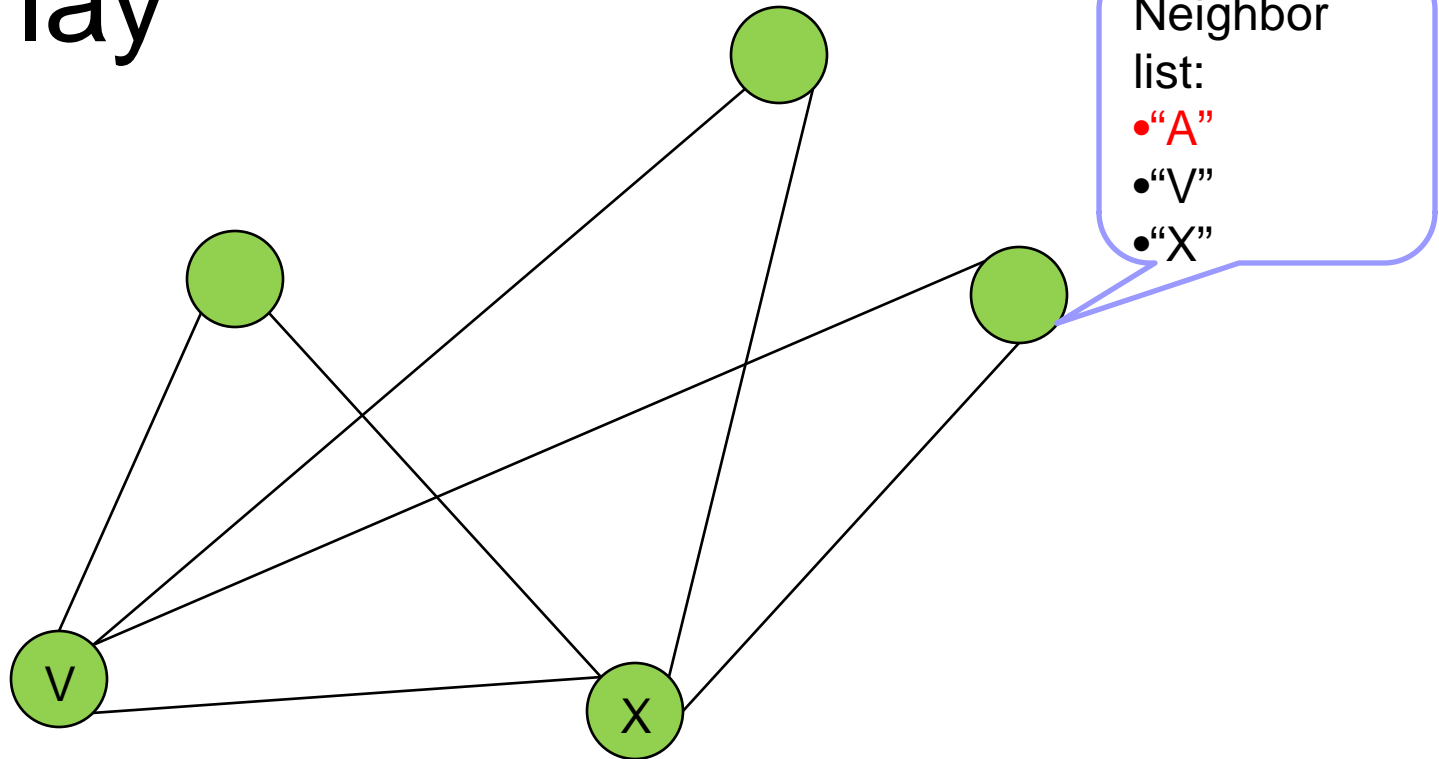
Gnutella, maintaining the overlay



Ping: <id, PING, ttl, hops, payload length (zero)>

Pong: <id, PONG, ttl, hops, payload length, port, ip, num. files, num. KBs>

Gnutella, maintaining the overlay



Peers can leave or fail at any time.

Gnutella: some issues

- Ping/Pong constitutes 50% of traffic
- Flooding causes excessive traffic
- Repeated searches with same keywords
- Large number of freeloaders (70% of users in 2000)

DHTs (Distributed Hash Tables)

- Hash table allows these operations on object identified by key:
 - Insert
 - Lookup
 - Delete
- Distributed Hash Table – same but in a distributed setting (object could be files)

DHT performance comparison

	Memory	Lookup latency	lookup overhead
Napster	$O(1)$ at client; $O(N)$ at server	$O(1)$	$O(1)$
Gnutella	$O(N)$	$O(N)$	$O(N)$
Chord (DHT)	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$

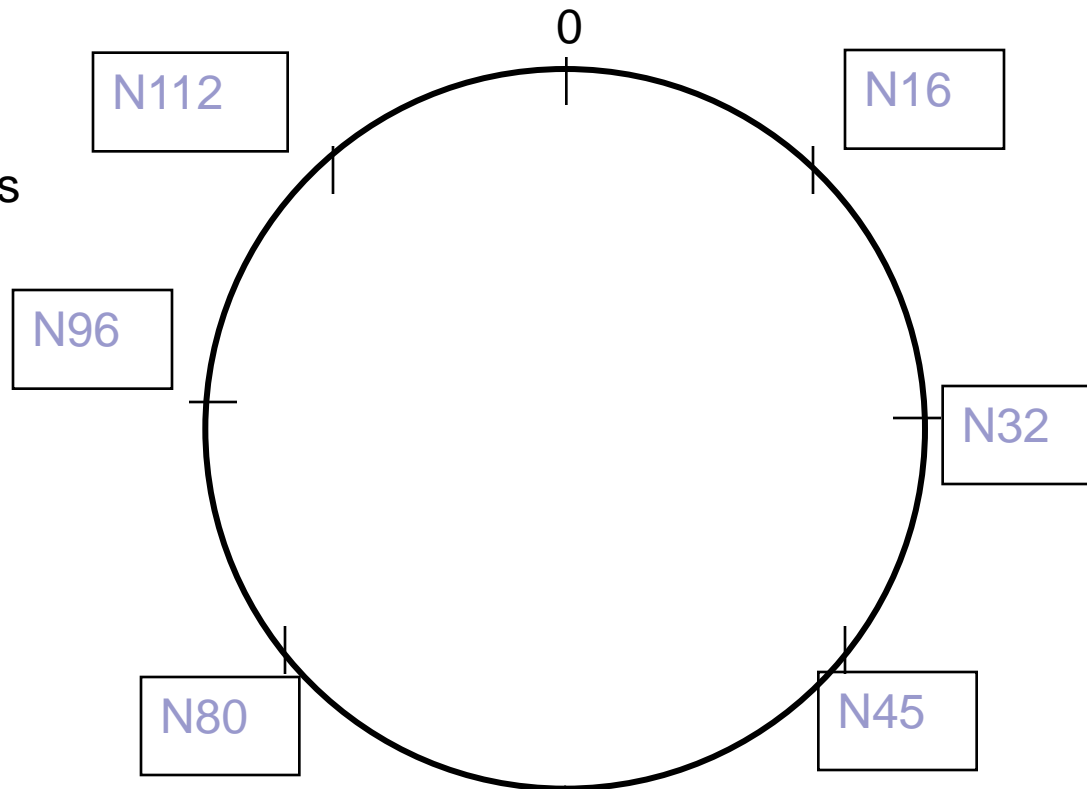
Chord

- Uses *Consistent Hashing* on peer's address
 - SHA-1 = Simple Hash Algorithm-1, a standard hashing function)
 - SHA-1(ip_address,port) → 160 bit string
 - Output truncated to m (< 160) bits
 - Called peer *id* (integer between 0 and $2^m - 1$)
 - Not unique but id conflicts very unlikely
 - “*Consistent*”? Any node A can calculate the peer id of any other node B, given B's ip address and port number
 - Can then map peers to one of 2^m logical points on a circle

Chord's ring

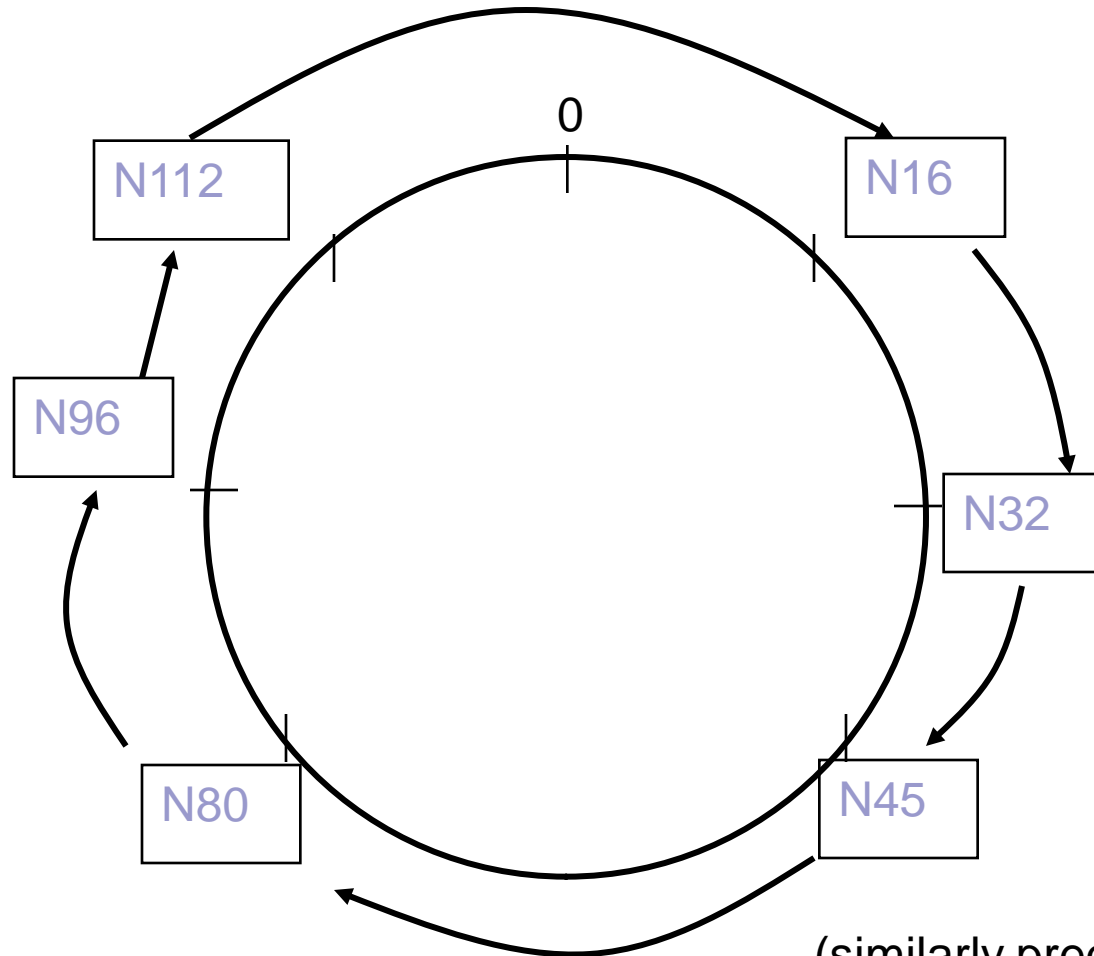
Say $m=7$

And 6 peers/nodes



Chord's peer pointers, *successors*

Say $m=7$



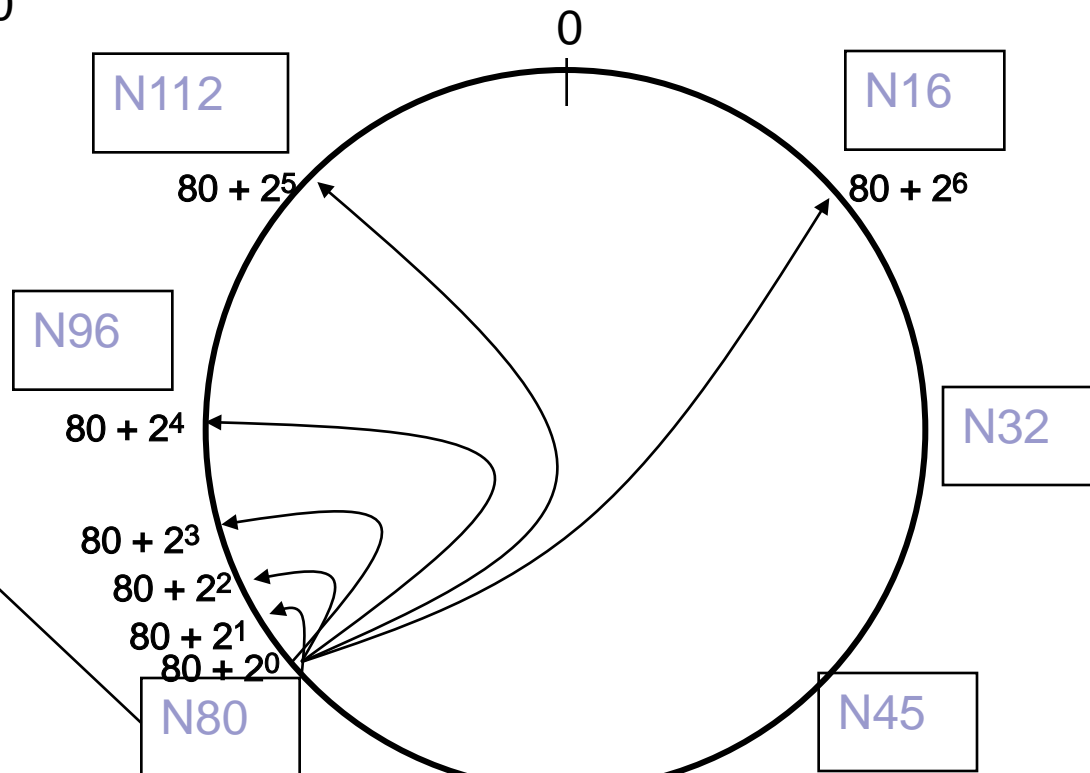
(similarly predecessors)

Chord's peer pointers, *finger table*

Say $m=7$

Finger Table at N80

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

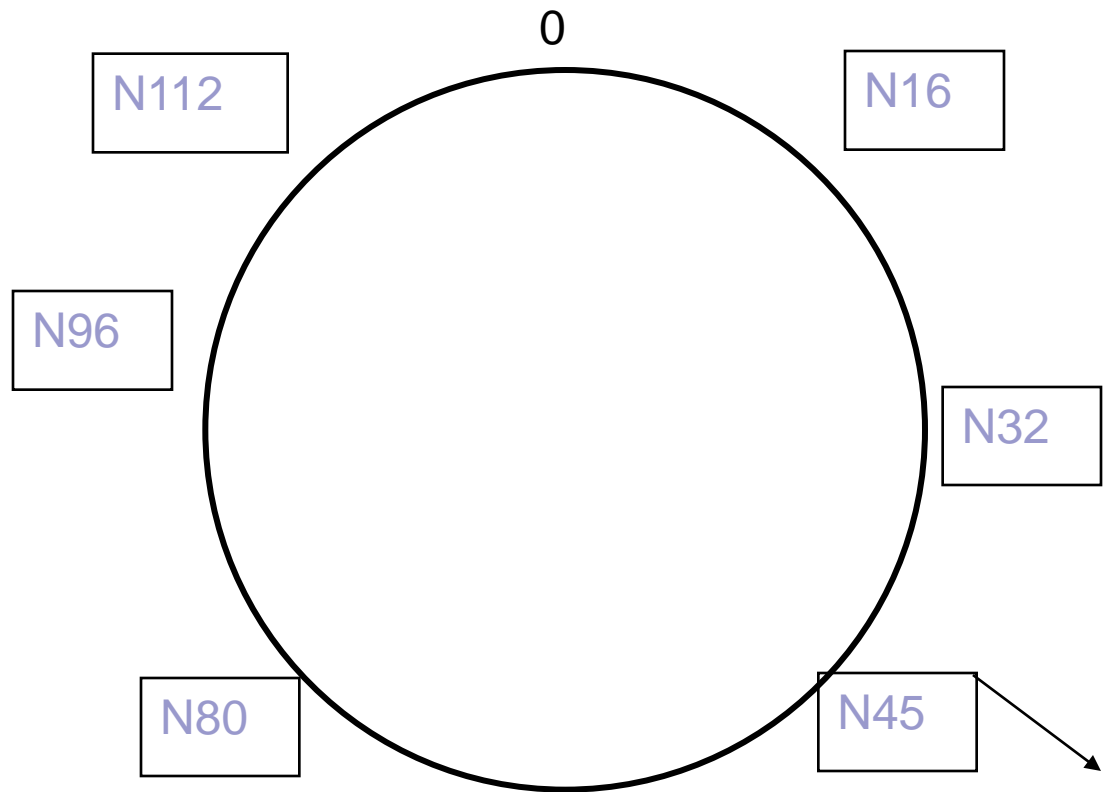


i th entry at peer with id n is first peer with id $\geq n + 2^i \pmod{2^m}$

Mapping files

Say $m=7$

- SHA-1(filename) \rightarrow 160 bits, truncated to m bits=file id or *key* (just like with peers)
- File `hello.txt` that maps to file id /key 42 is stored at *first peer* with id ≥ 42



File `hello.txt` with key **K42** stored here

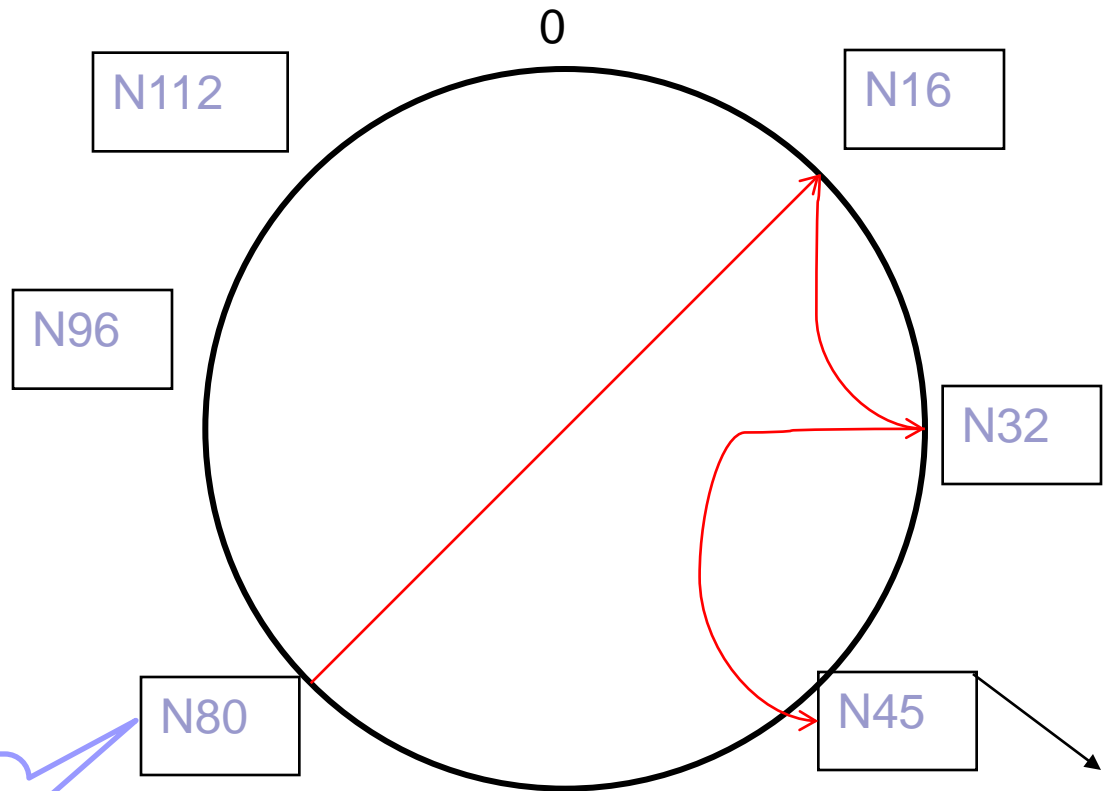
Searching

Say $m=7$

- At node n , send query for key k to largest successor/finger entry $< k$ (all mod m)
- if none exist, send query to $successor(n)$

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

“hello.txt”?
(K42)



File hello.txt with
key **K42** stored here

Dealing with dynamism

■ Peers

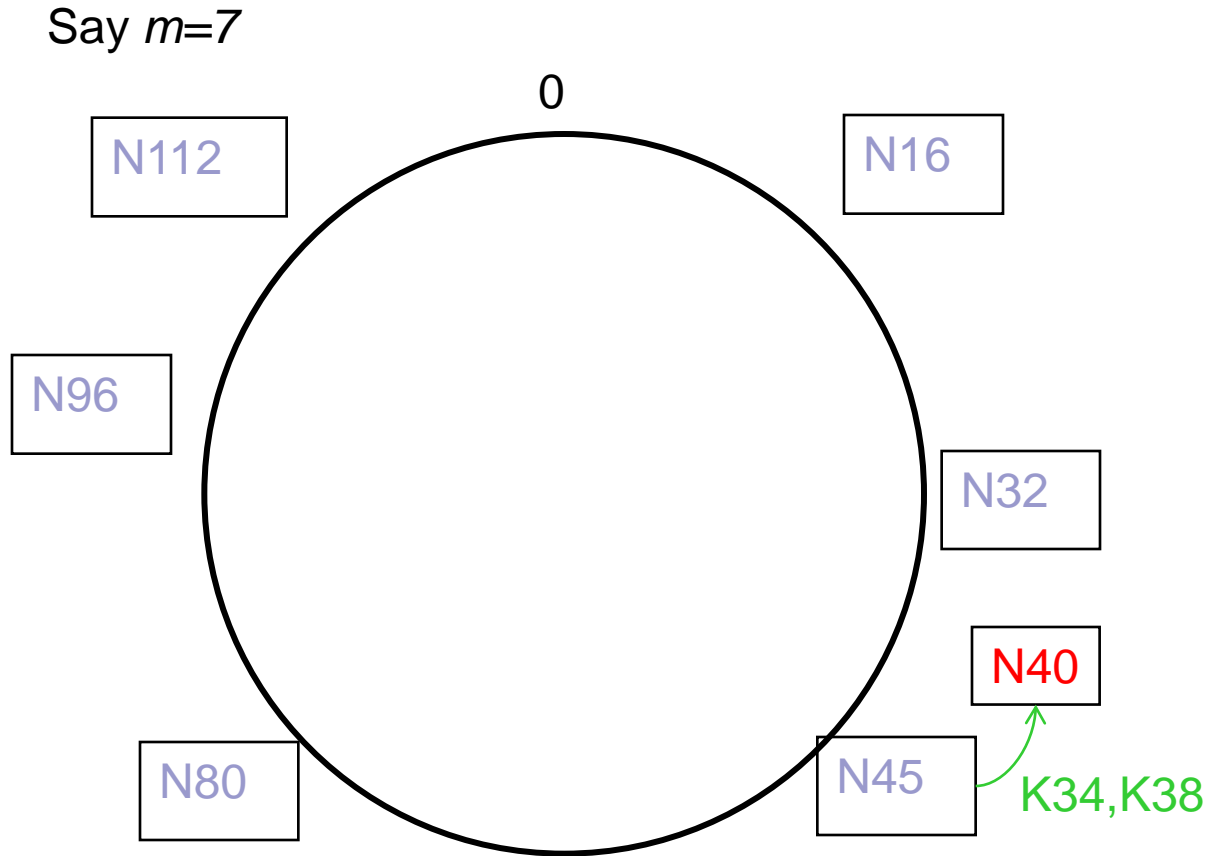
- Join
- Leave
- Fail

■ Update

- Successors
- Fingers
- Key locations

Joining

- Introducer directs N40 to N32
- N32 updates successor to N40
- N40 initializes successor to N45, and obtains fingers from it
- N40 periodically talks to neighbors to update own finger table (**stabilization protocol**)
- N40 may need to copy some files/keys from N45 (files with fileid between 32 and 40)



Some Chord issues

- Can get partitioned
- Dynamism
 - traces from the Overnet system show *hourly* peer turnover rates (**churn**) could be 10-15% of total number of nodes in system
 - Leads to excessive (unnecessary) key copying
 - Stabilization algorithm may need to consume more bandwidth to keep up