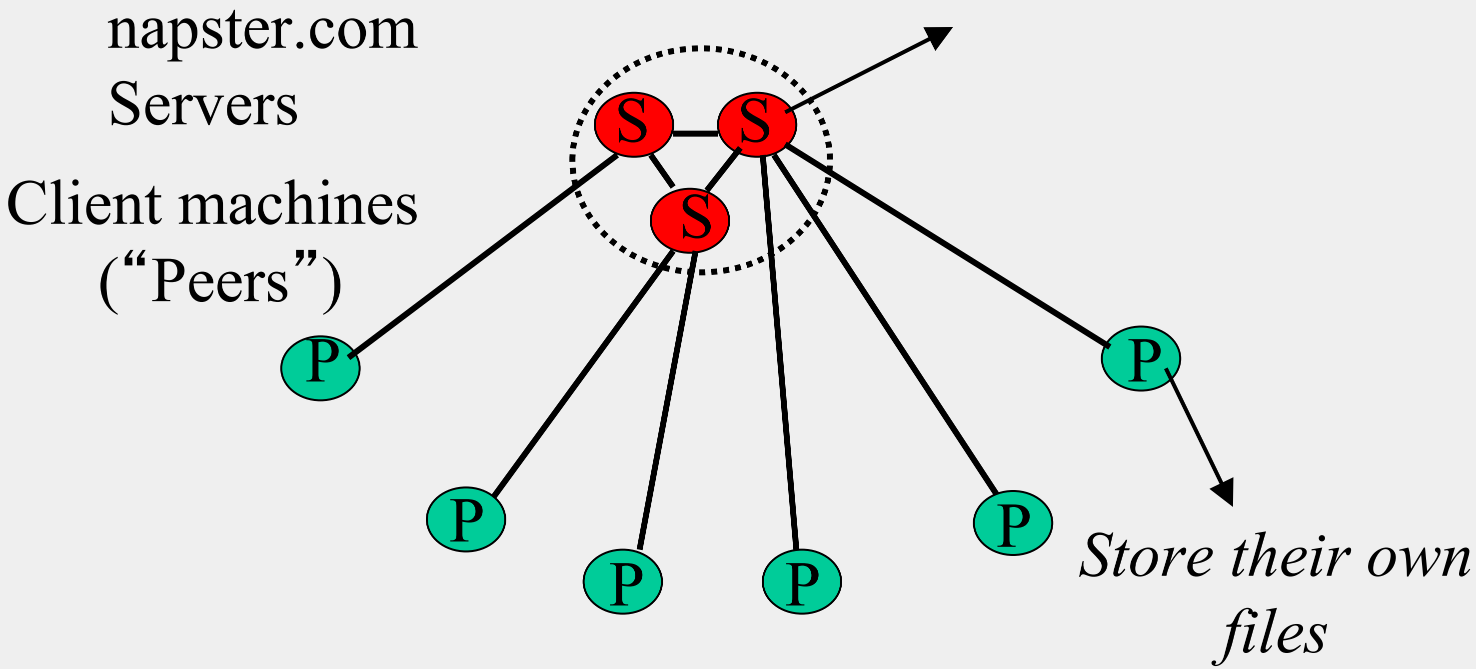
CS 425 / ECE 428
Distributed Systems
Fall 2015

Indranil Gupta (Indy)

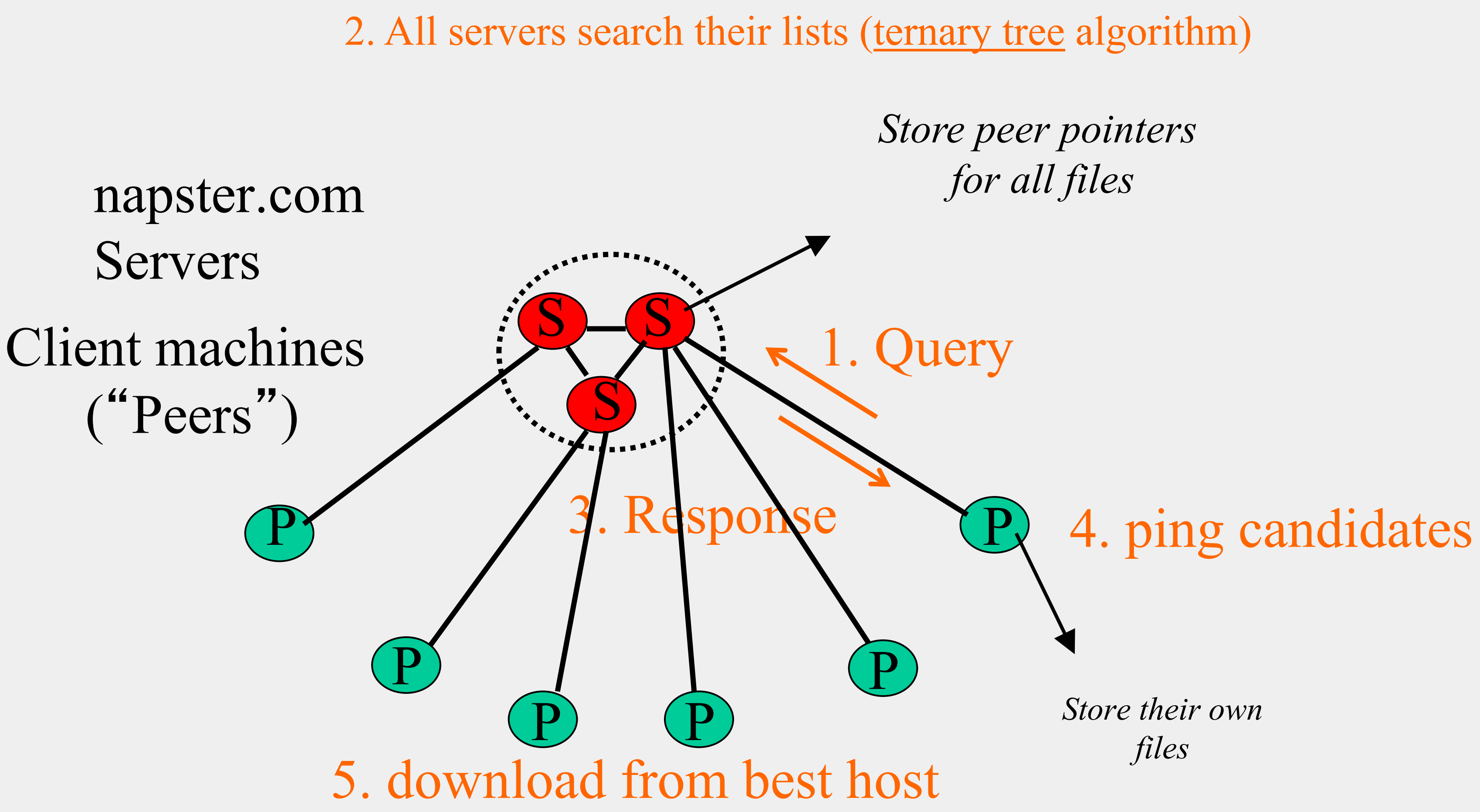
Peer-to-peer Systems

Napster Structure

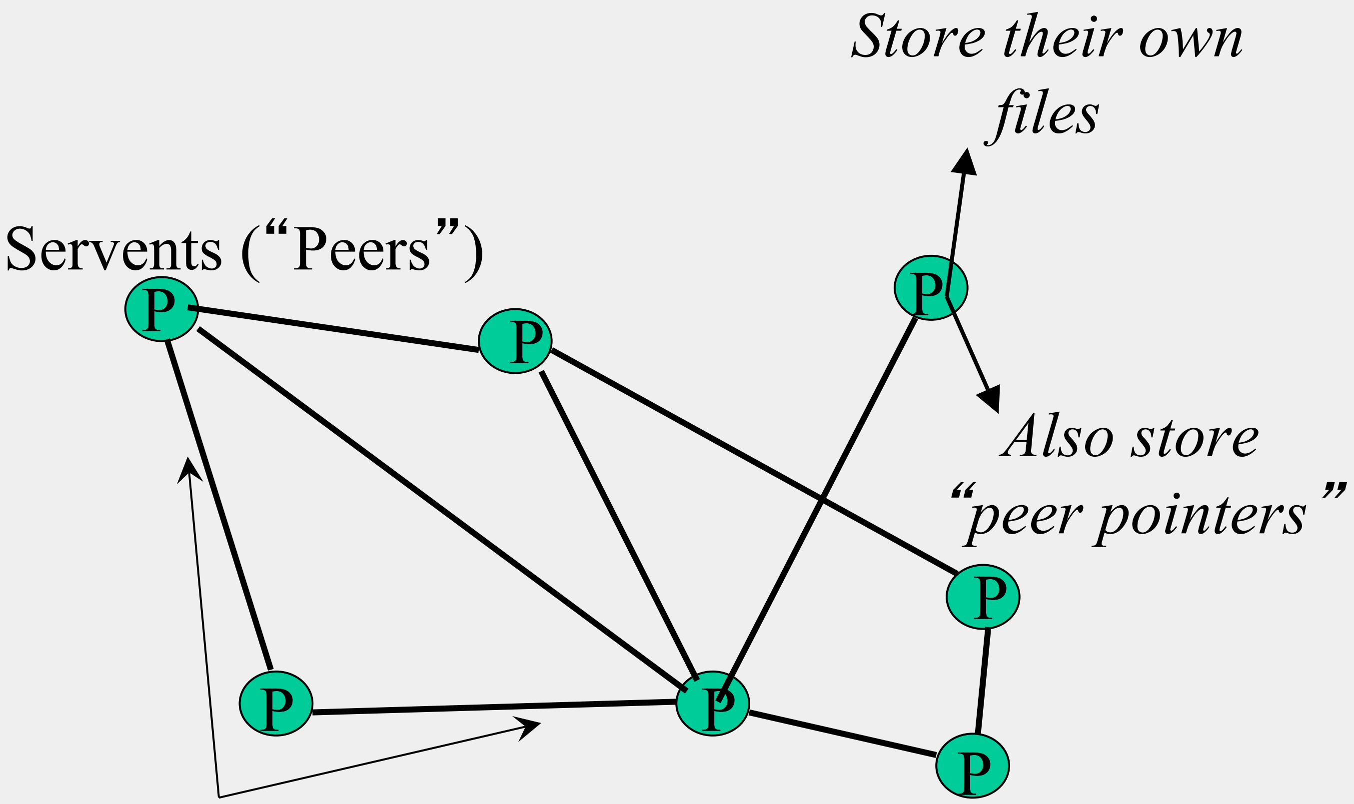
Filename	Info about
PennyLane.mp3	Beatles, @ 128.84.92.23:1006 …..



Napster Search



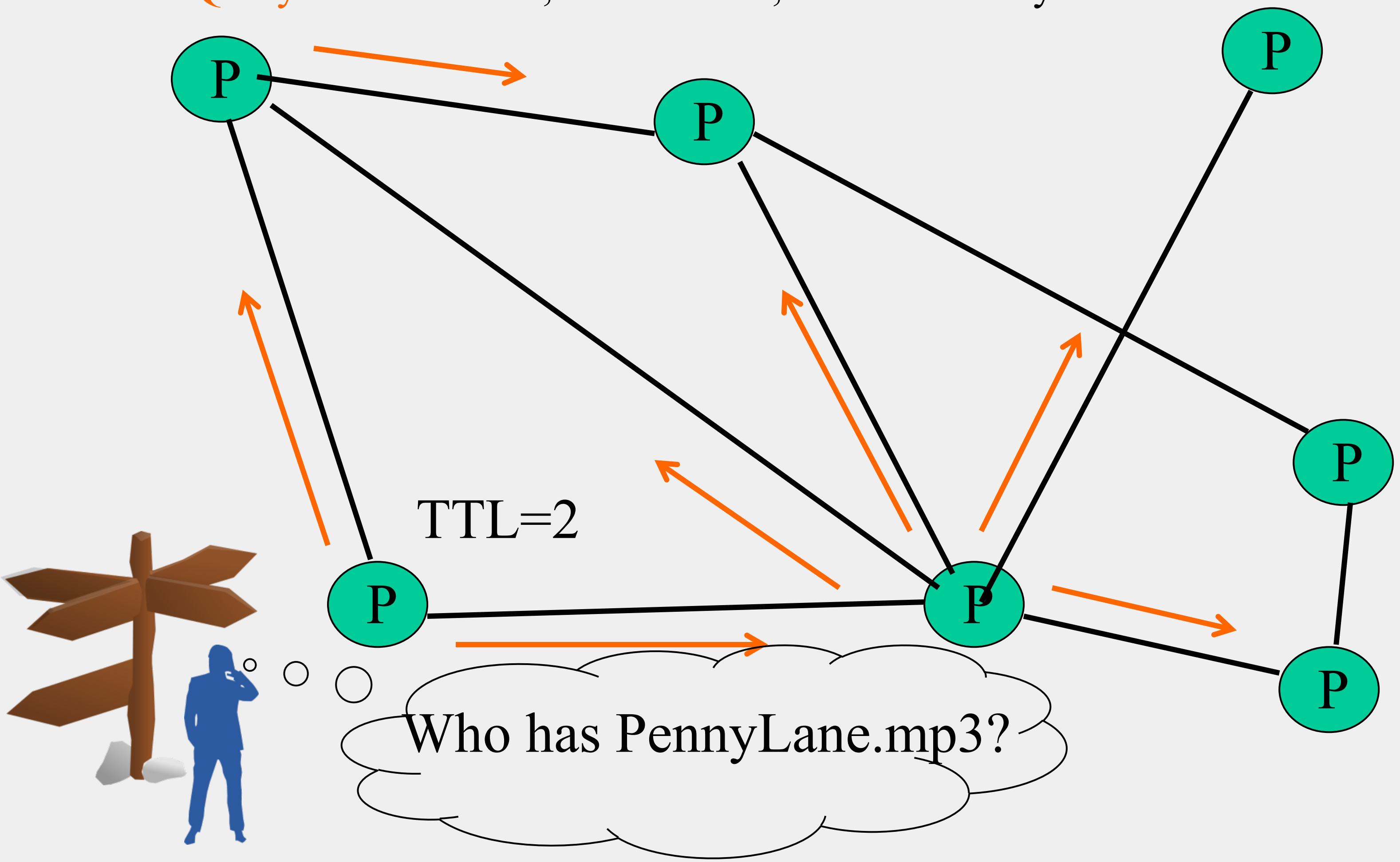
Gnutella



Connected in an **overlay** graph
(== each link is an implicit Internet path)

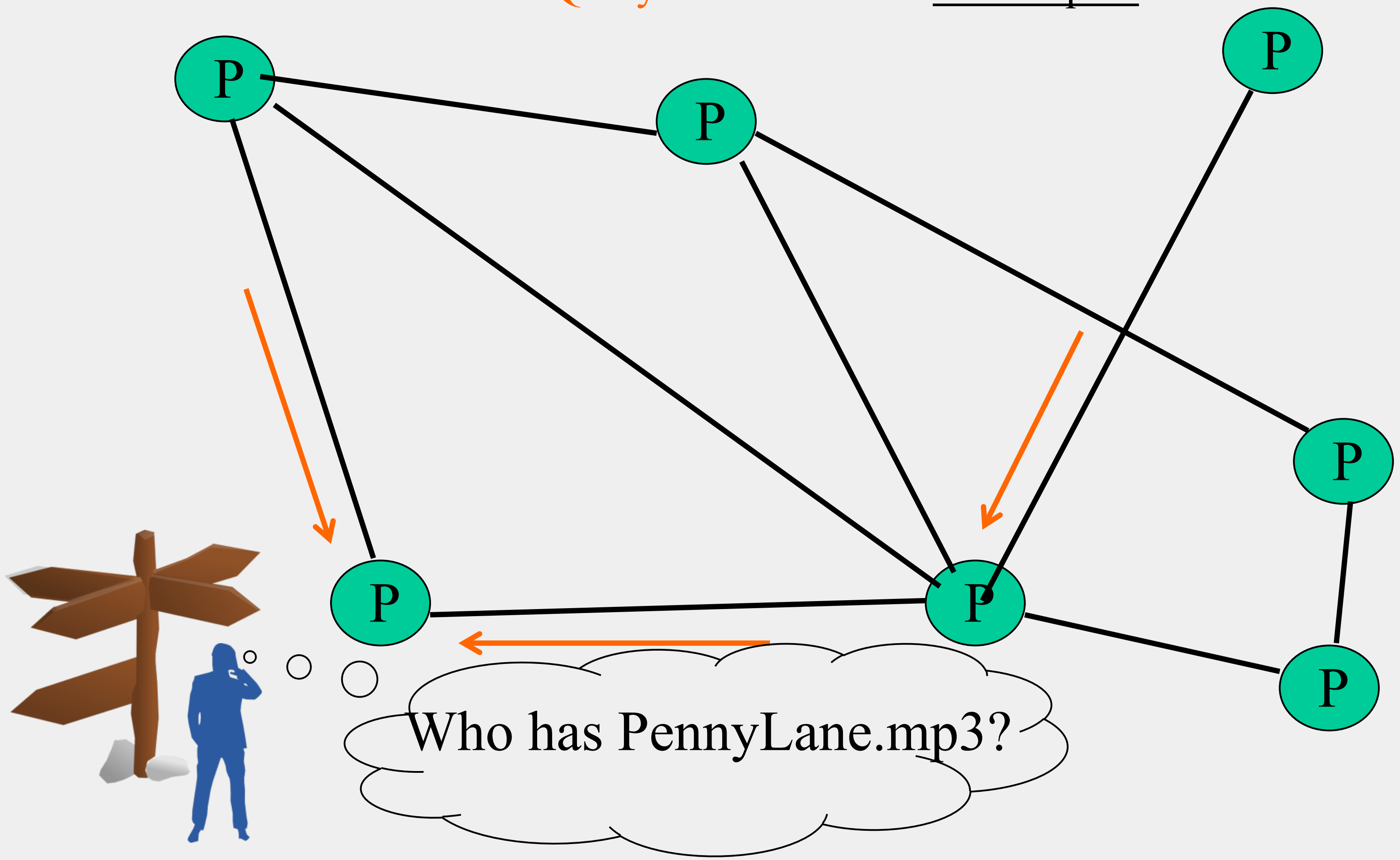
Gnutella Search

Query's flooded out, ttl-restricted, forwarded only once



Gnutella Search

Successful results QueryHit' s routed on reverse path



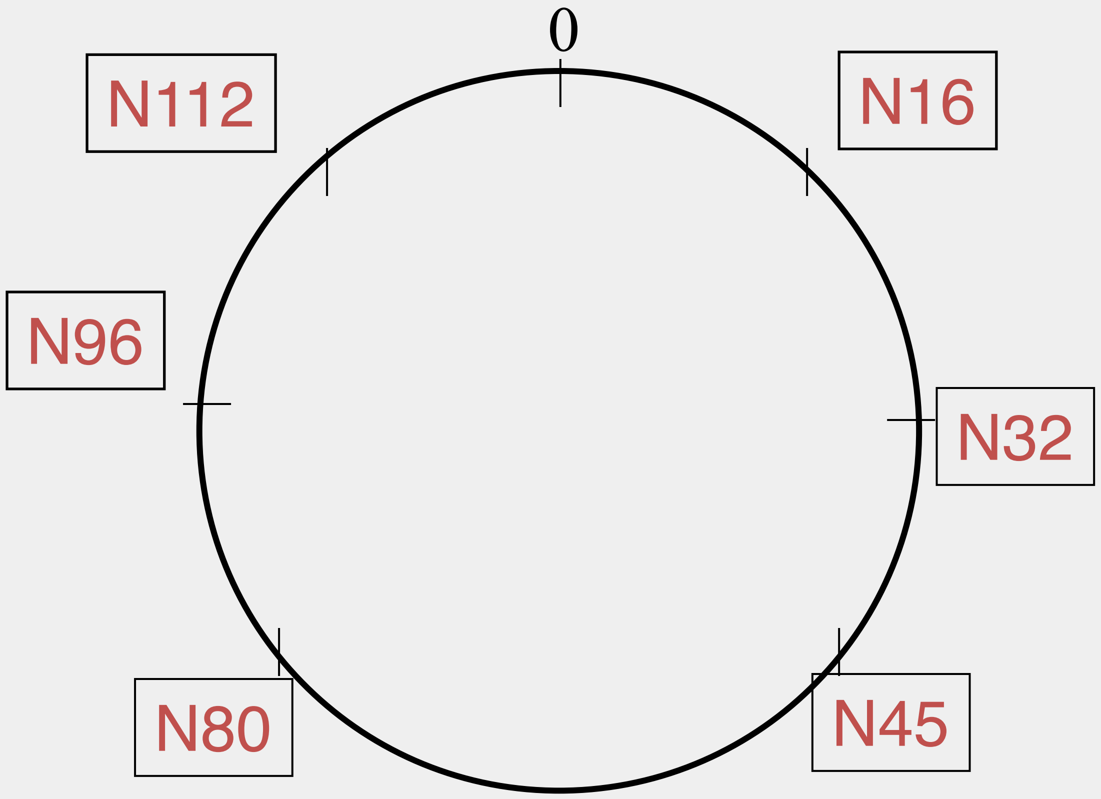
Chord

- 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

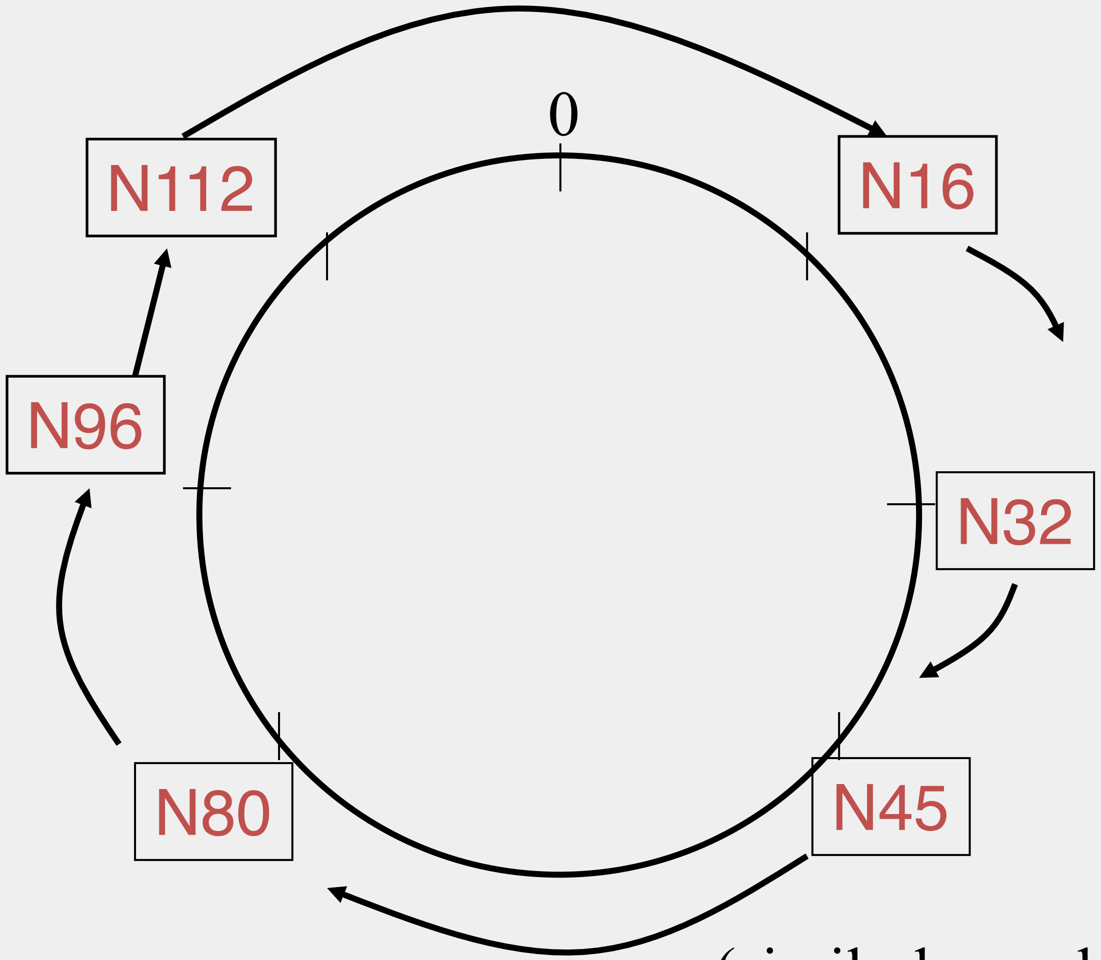
Say m=7

6 nodes



Peer pointers (1): successors

Say m=7



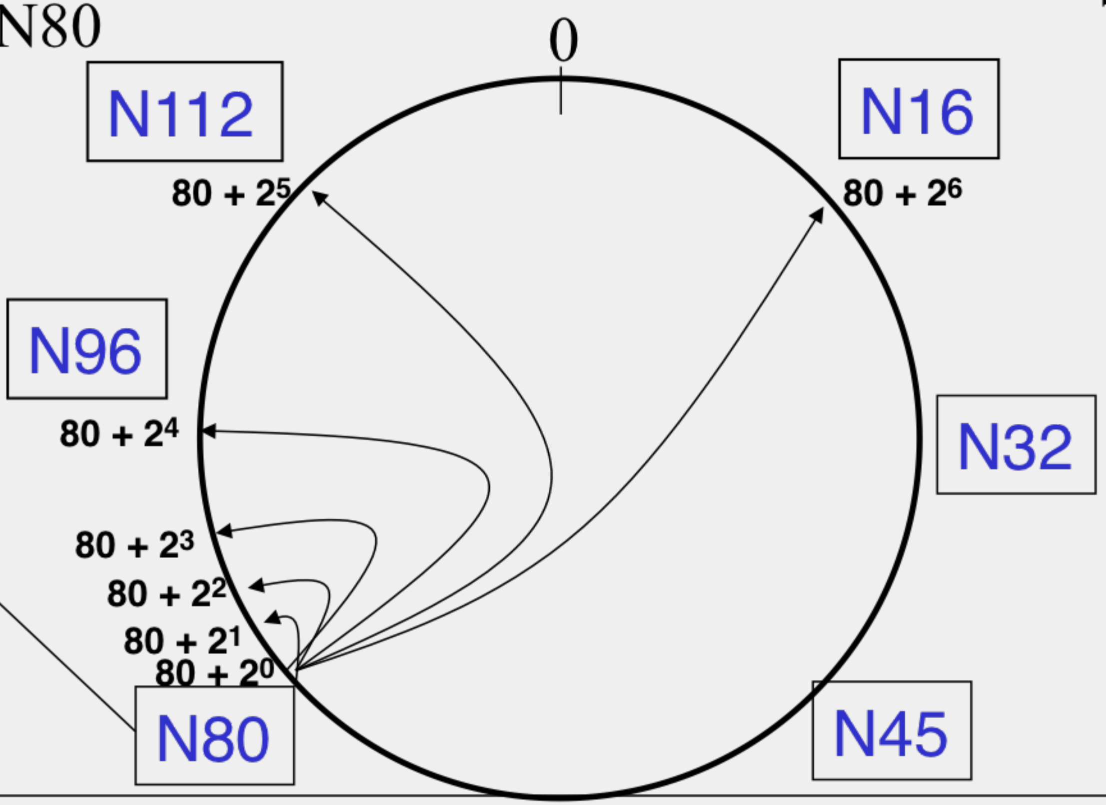
(similarly predecessors)

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



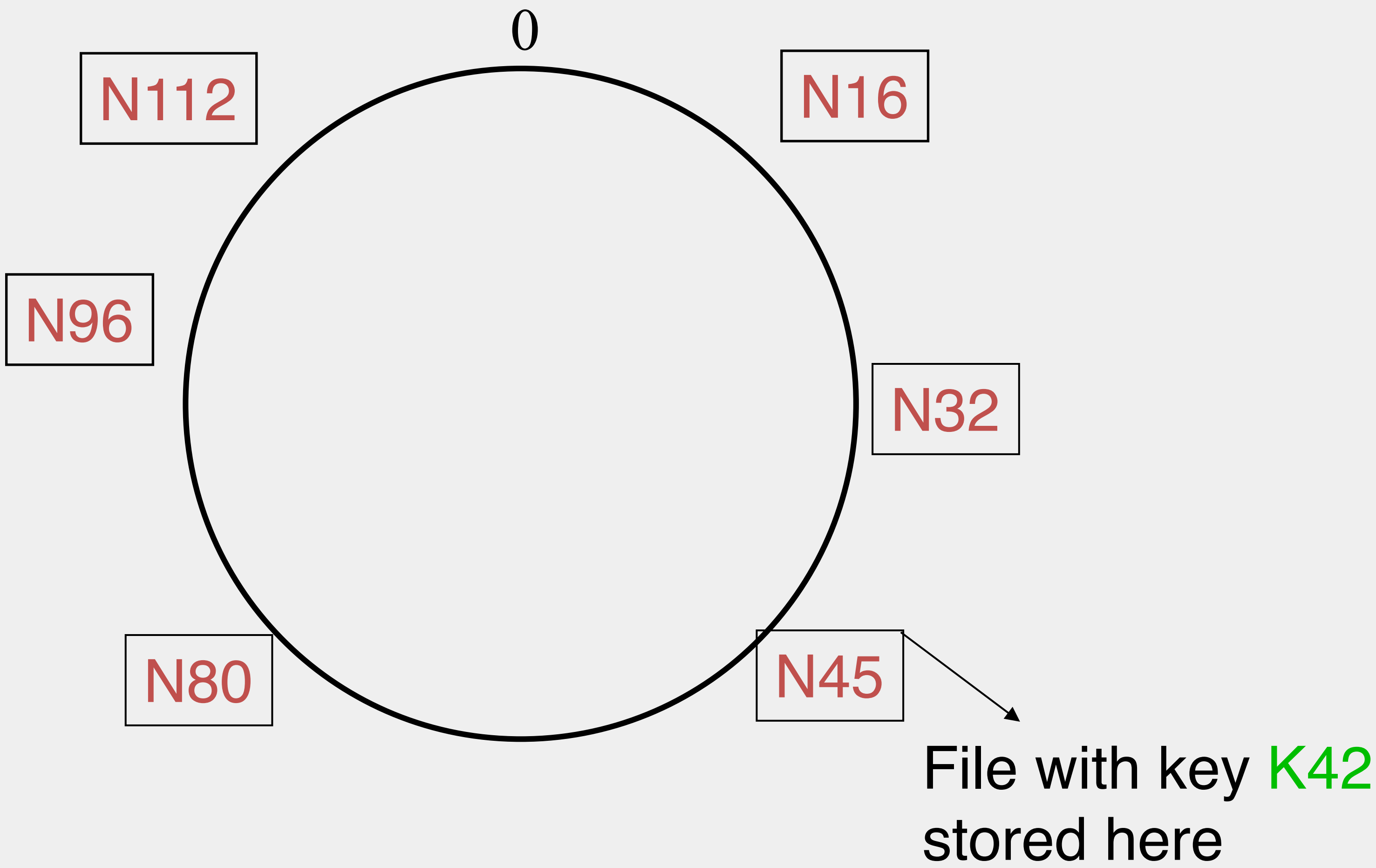
ith entry at peer with id n is first peer with id >= $n + 2^i (\text{mod} 2^m)$

What about the files?

- 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*)

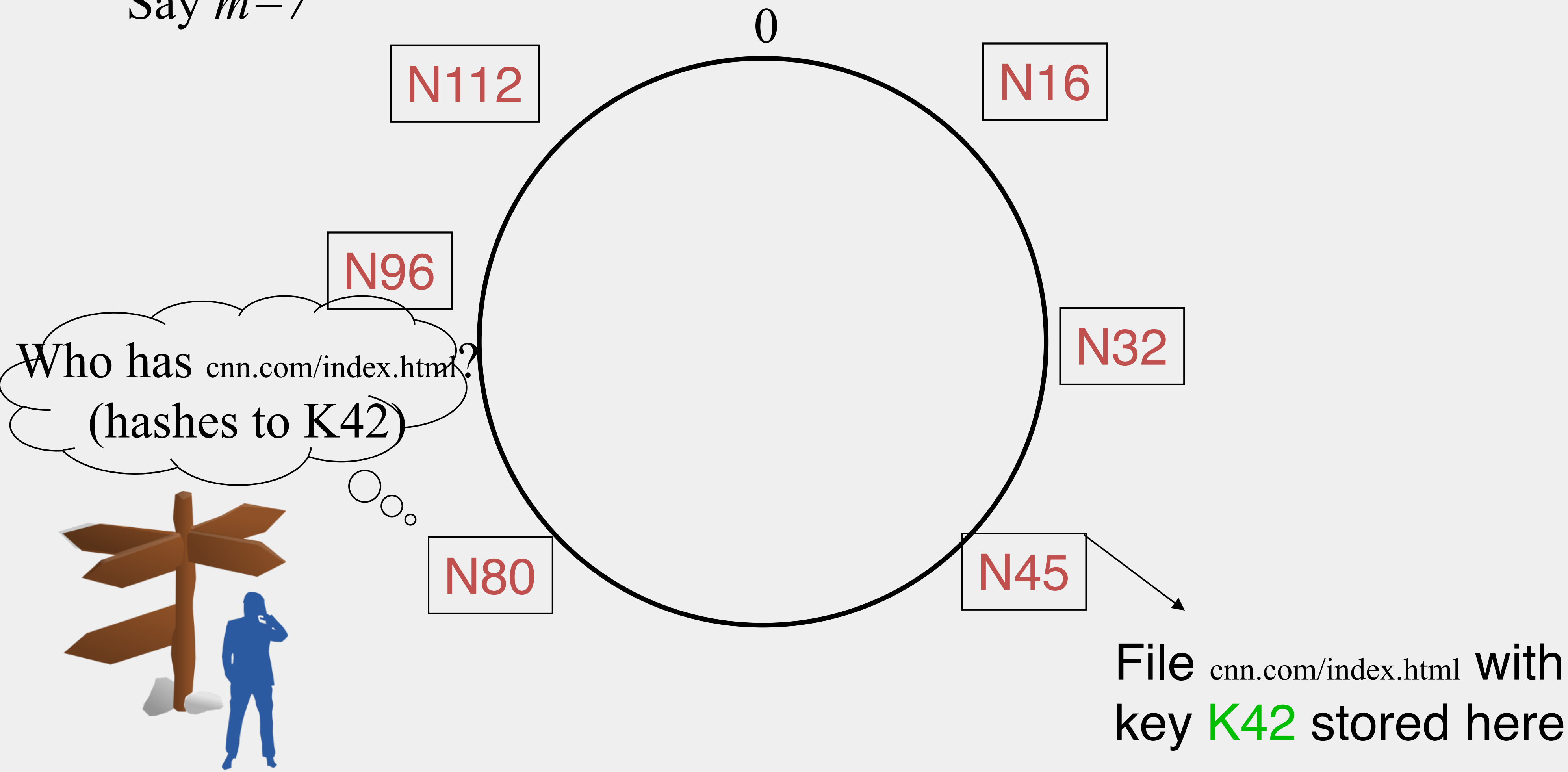
Mapping Files

Say m=7



Search

Say $m=7$

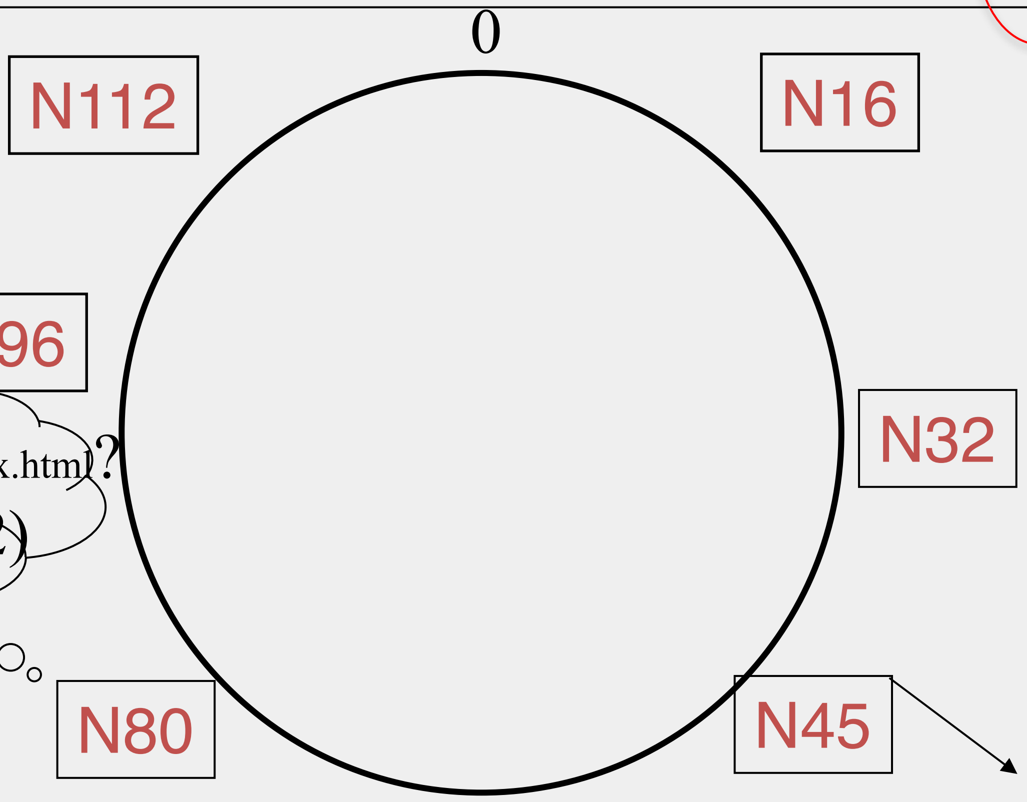


Search

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

At or to the anti-clockwise of *k*
(it wraps around the ring)

Say $m=7$



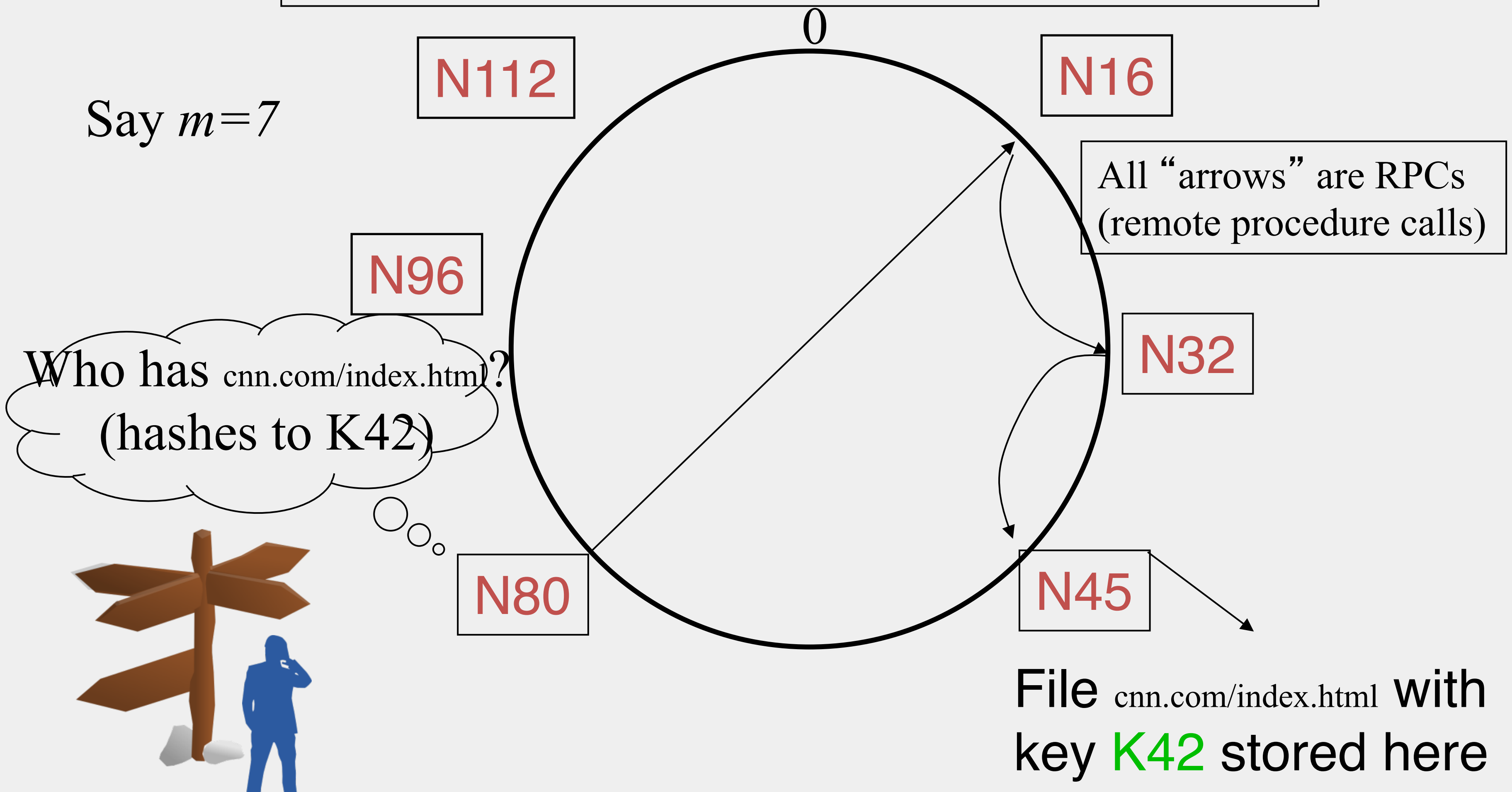
Who has cnn.com/index.html?
(hashes to K42)

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

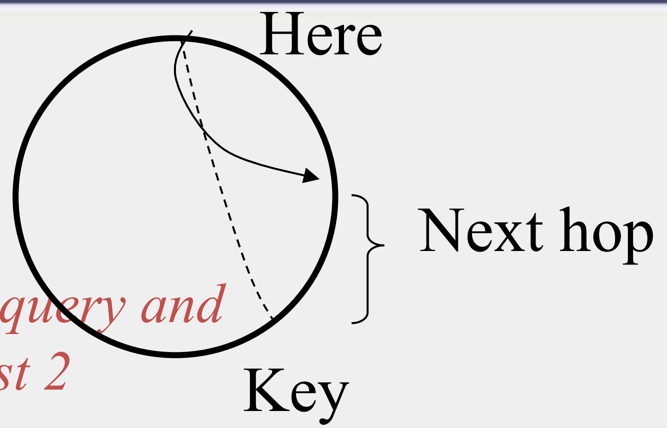
Search

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

Say $m=7$



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! and "Balls and Bins")

  So using *successors* in that range will be ok, using another *O(log(N))* hops

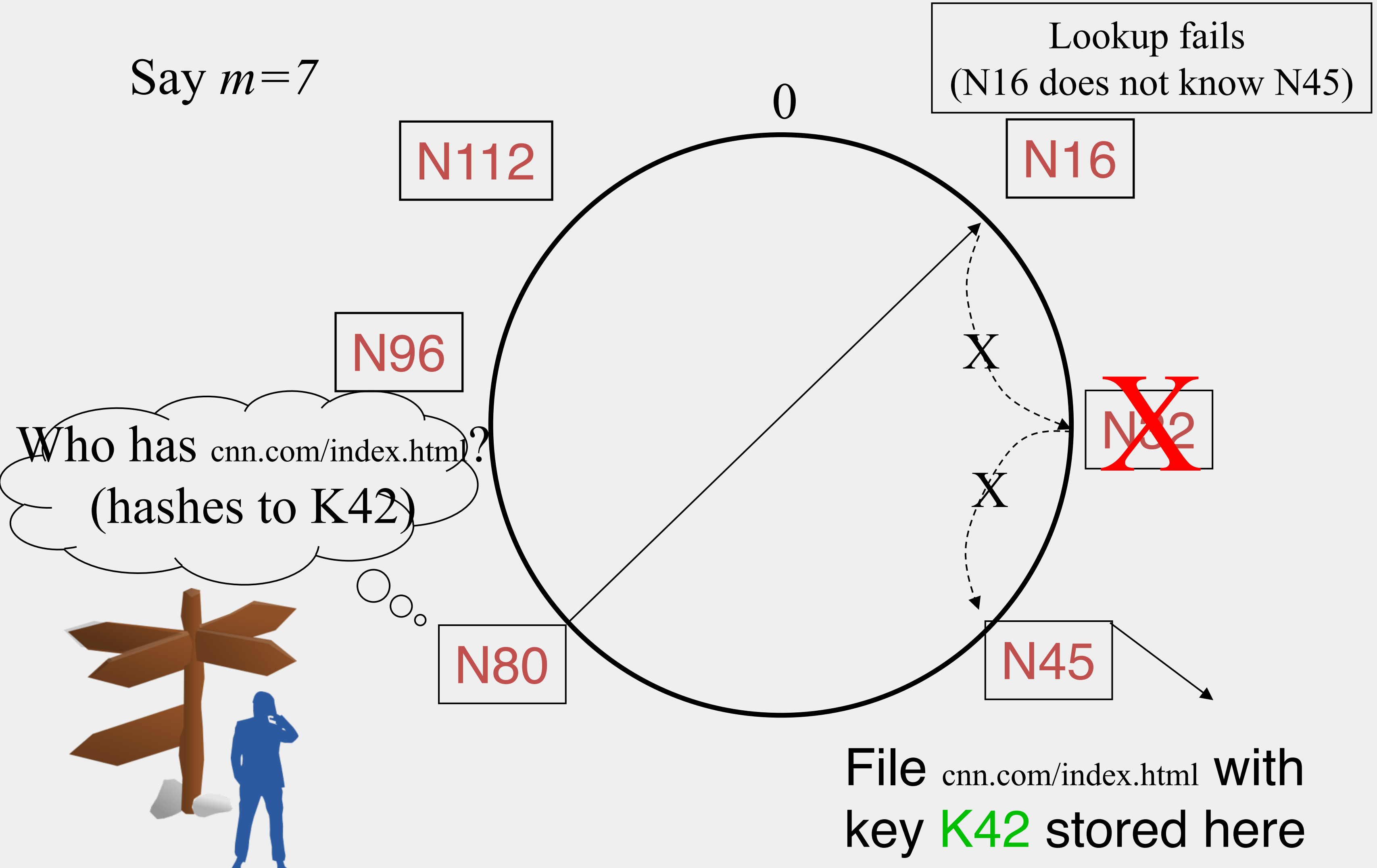
Analysis (contd.)

- *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

Rest of the slides are for recommended reading

Search under peer failures

Say m=7



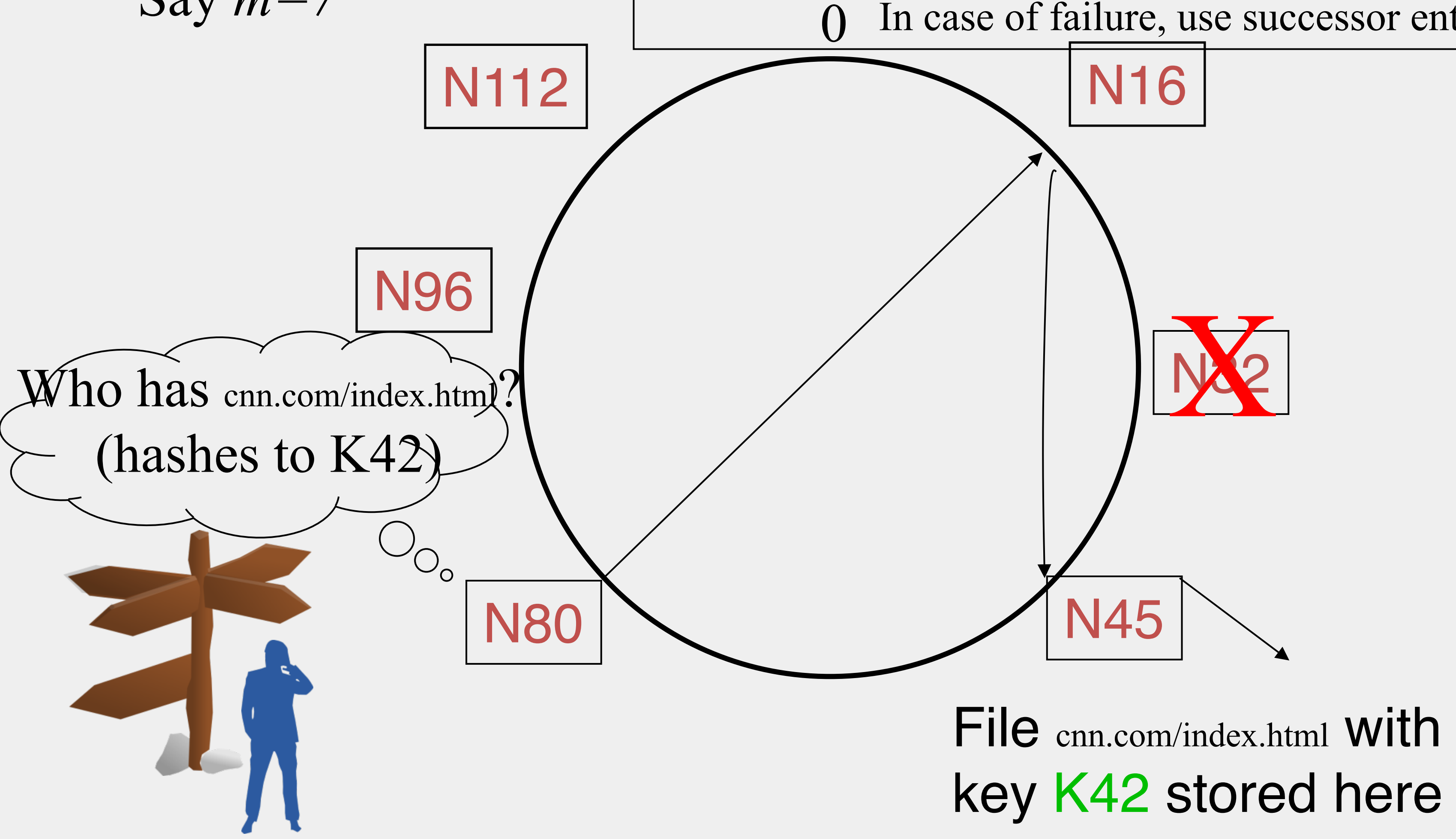
File cnn.com/index.html with key K42 stored here

Search under peer failures

Say $m=7$

One solution: maintain *r* multiple *successor* entries

In case of failure, use successor entries



Search under peer failures

- Choosing *r=2log(N)* suffices to maintain *lookup correctness* w.h.p.(i.e., ring connected)
  - Say 50% of nodes fail
  - Pr(at given node, at least one successor alive)=

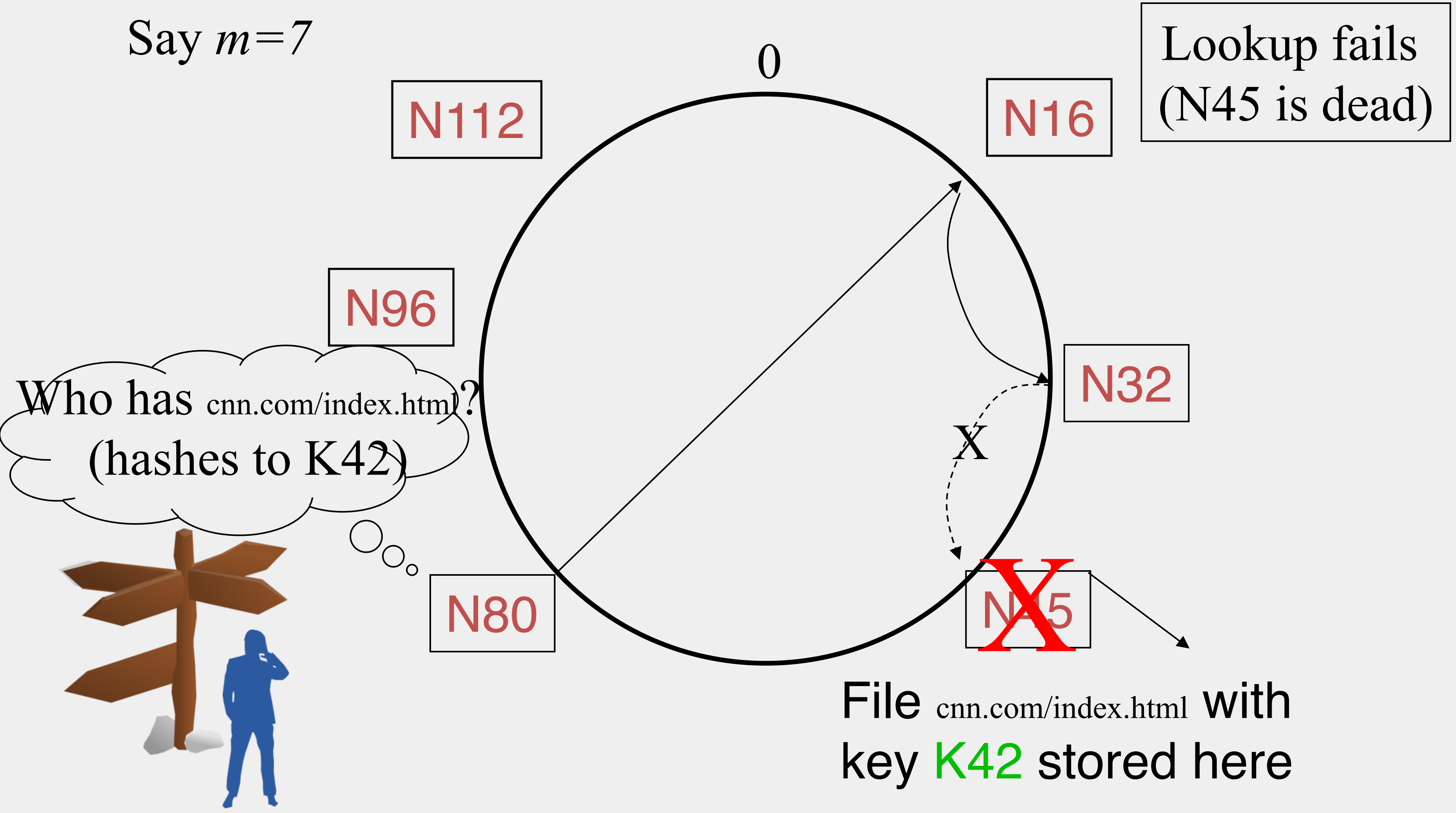
$$1-\left(\frac{1}{2}\right)^{2\log N} = 1-\frac{1}{N^2}$$

  - Pr(above is true at all alive nodes)=

$$\left(1-\frac{1}{N^2}\right)^{N/2} = e^{-\frac{1}{2N}} \approx 1$$

Search under peer failures (2)

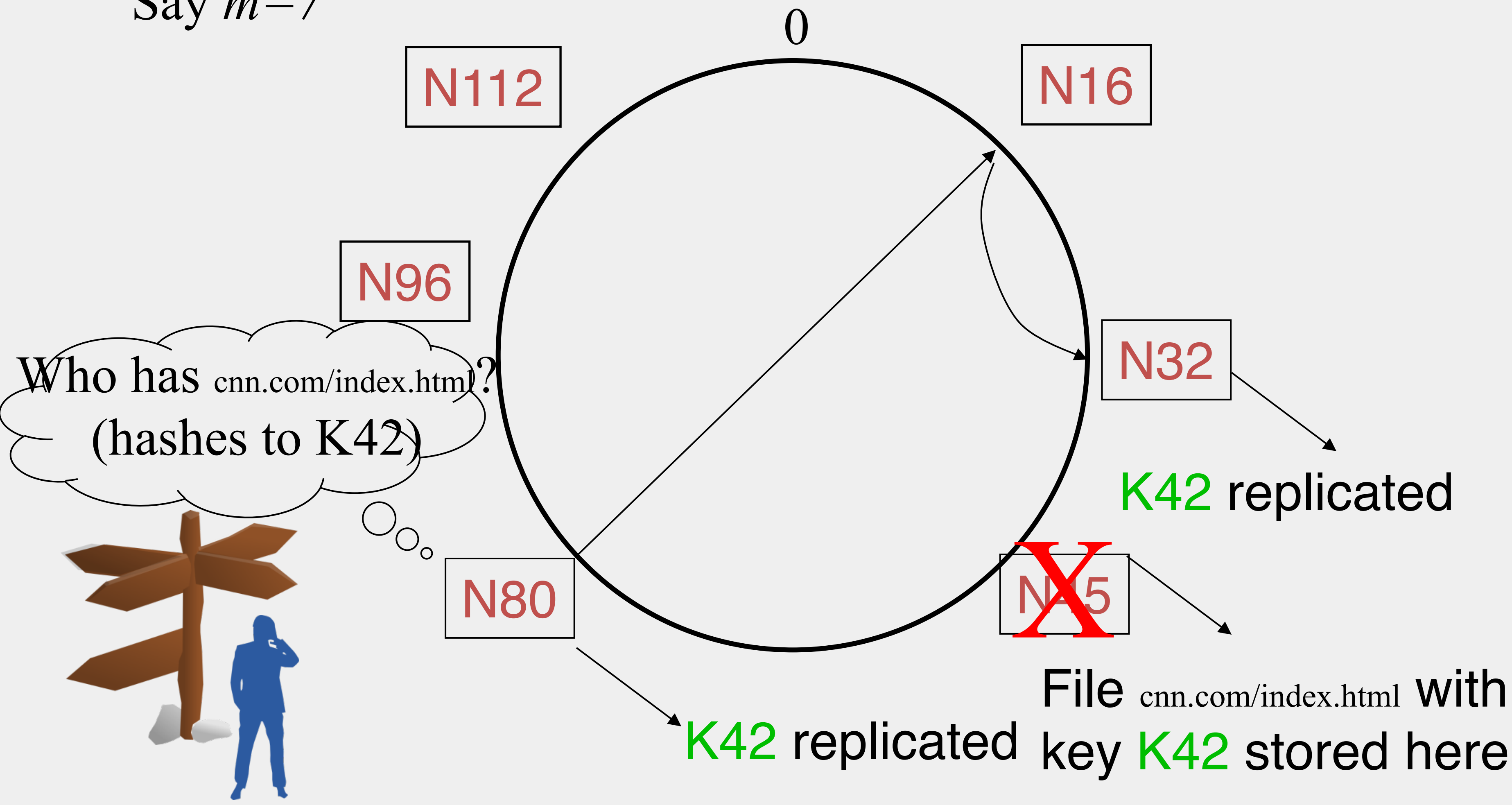
Say m=7



Search under peer failures (2)

Say $m=7$

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



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.

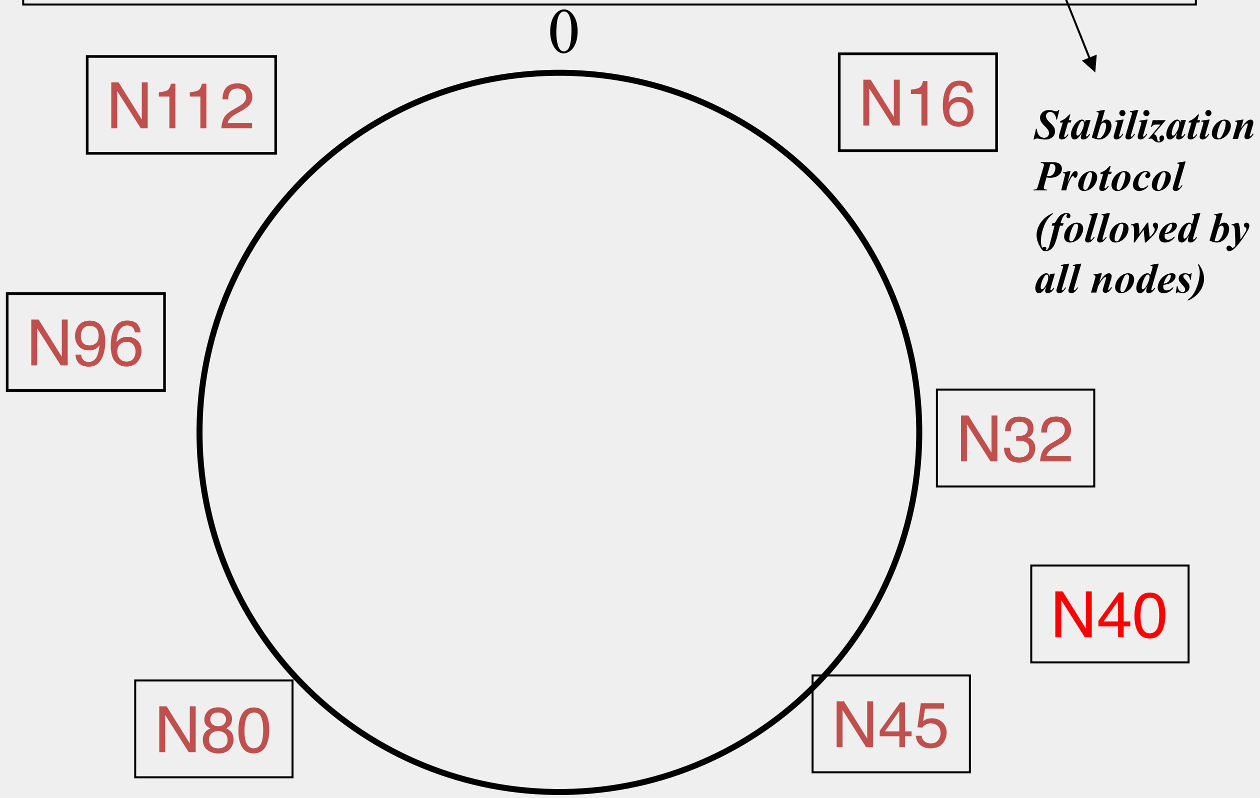
So, all the time, need to:

→ Need to update *successor*s and *finger*s, 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

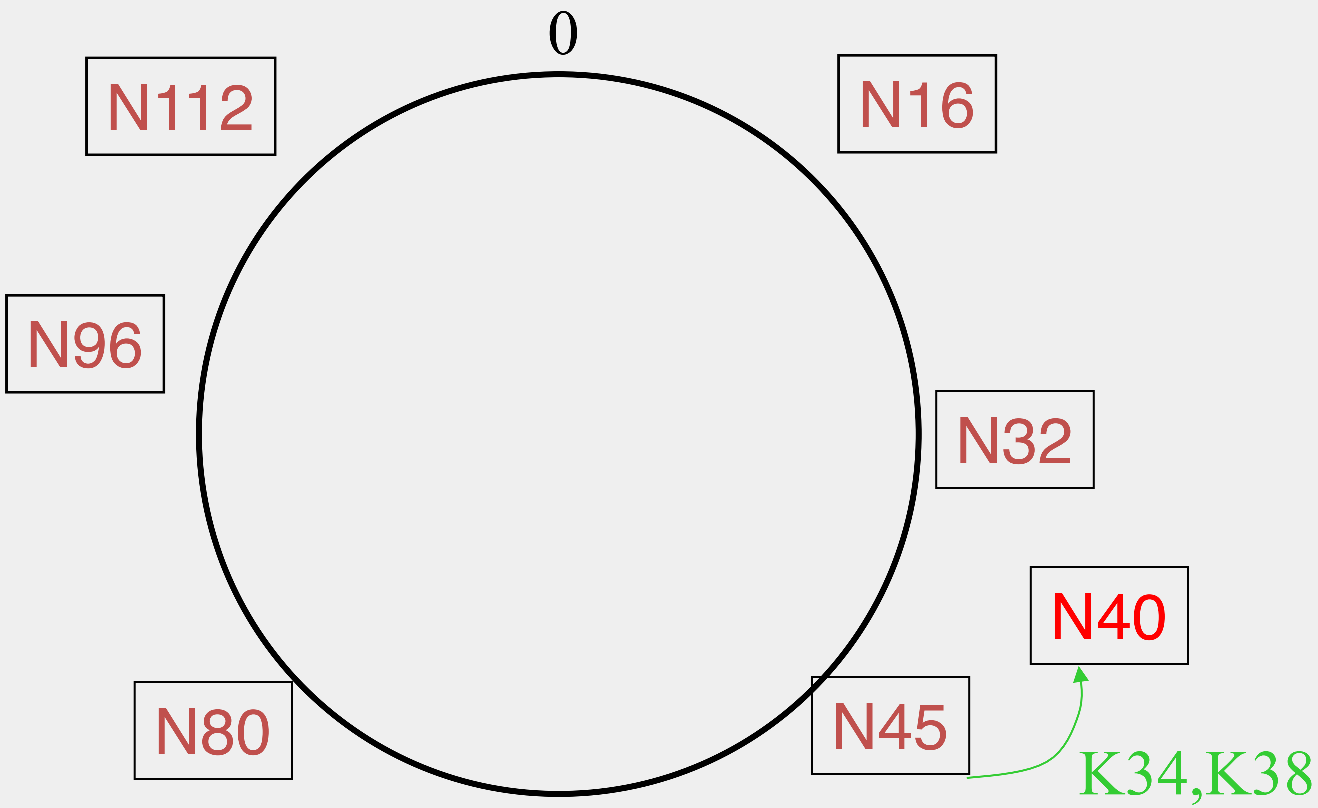
Say $m=7$



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 [Why?]
- 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* (you've seen them!)

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]