CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

P2P SYSTEMS

Lecture E

CHORD

DHT=Distributed Hash Table

- A hash table allows you to insert, lookup, and delete objects with keys
- A *distributed* hash table allows you to do the same in a distributed setting (objects=files)
- Performance concerns:
  - Load balancing
  - Fault-tolerance
  - Efficiency of lookups and inserts
  - Locality
- Napster, Gnutella, FastTrack are all DHTs (sort of)
- So is Chord, a structured peer-to-peer system that we study next

Comparative Performance

	Memory	Lookup Latency	#Messages for a lookup	
Napster	*O(1)* (*O(N)*@server)	*O(1)*	*O(1)*	
Gnutella	*O(N)*	*O(N)*	*O(N)*	

Comparative Performance

	Memory	Lookup Latency	#Messages for a lookup	
Napster	$O(1)$ ($O(N)$@server)	$O(1)$	$O(1)$	
Gnutella	$O(N)$	$O(N)$	$O(N)$	
Chord	$O(log(N))$	$O(log(N))$	$O(log(N))$	

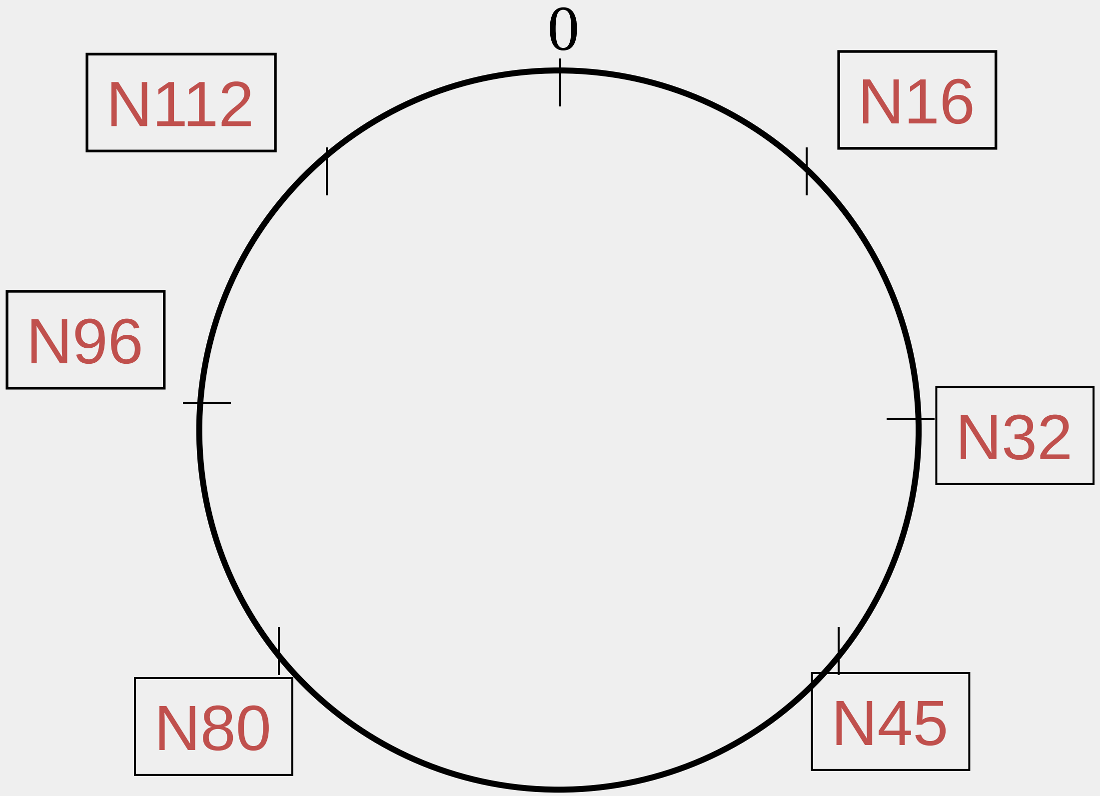
Chord

- 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

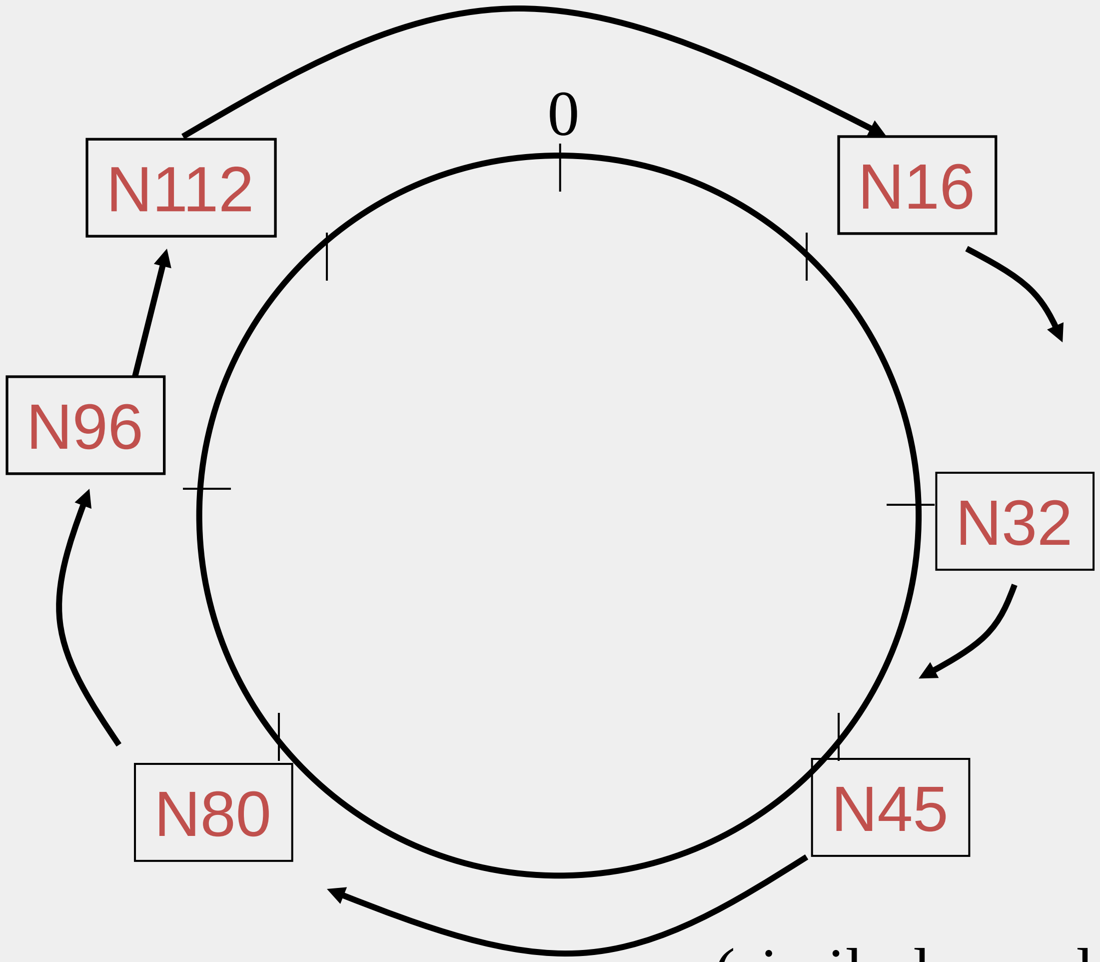
Say $m=7$

6 nodes



Peer pointers (1): successors

Say m=7



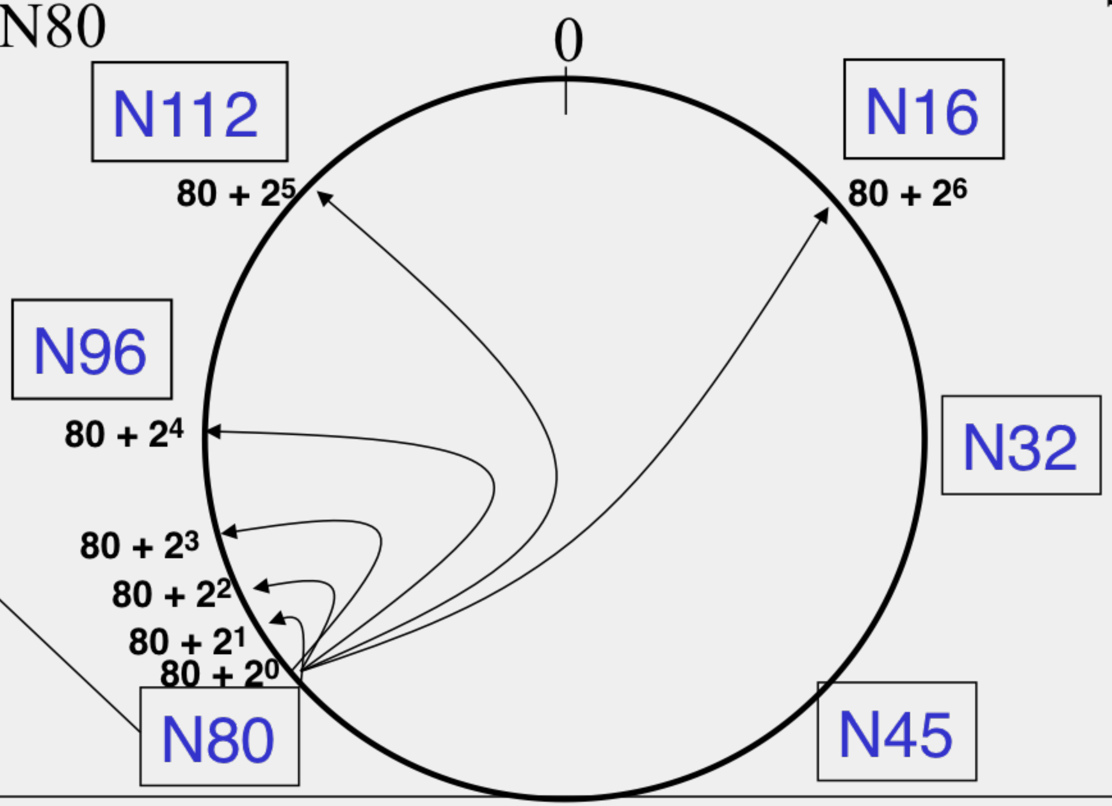
(similarly predecessors)

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



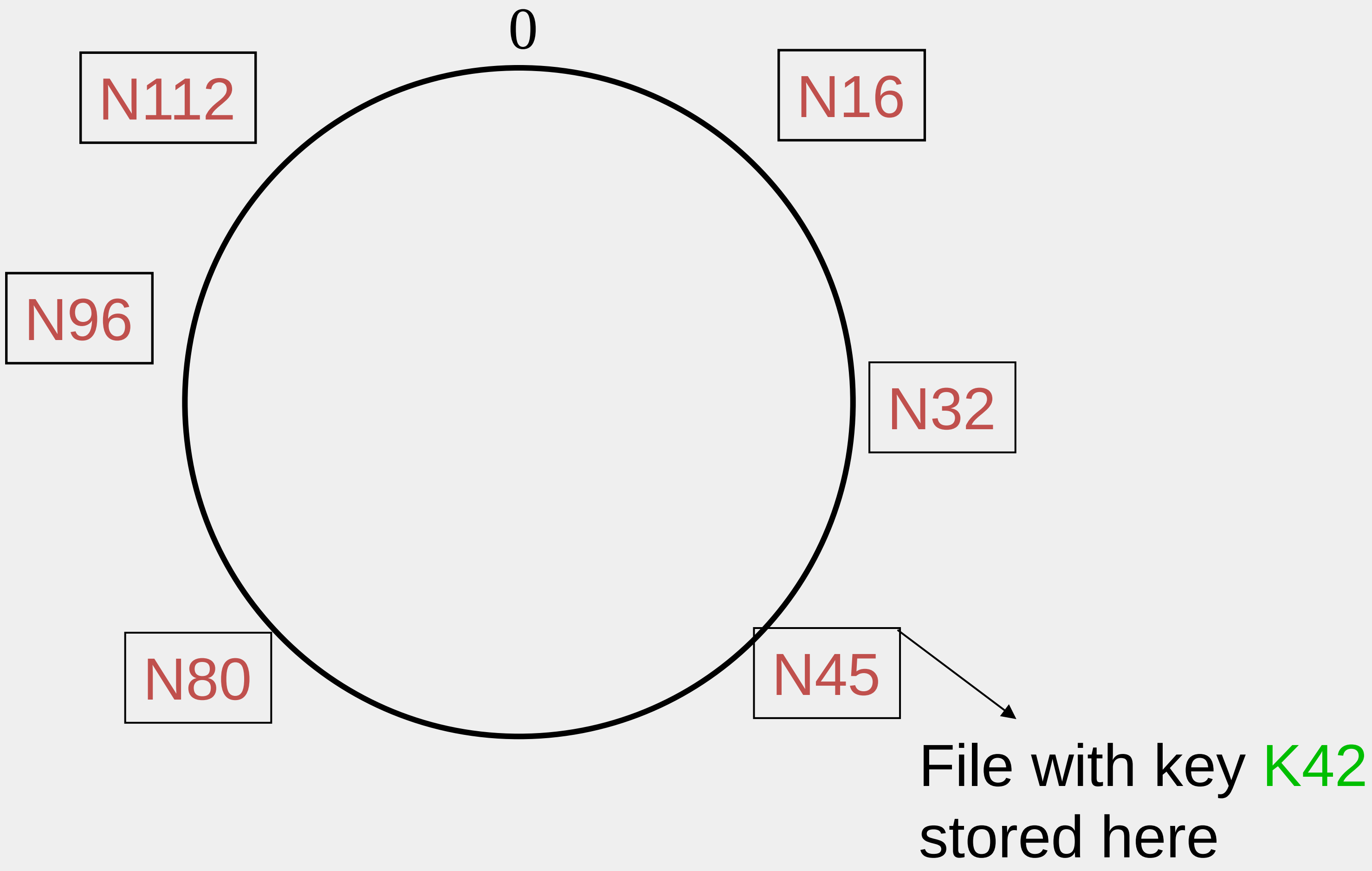
ith entry at peer with id n is first peer with id >= $n + 2^i (\mathrm{mod}\, 2^m)$

What about the 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 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*)

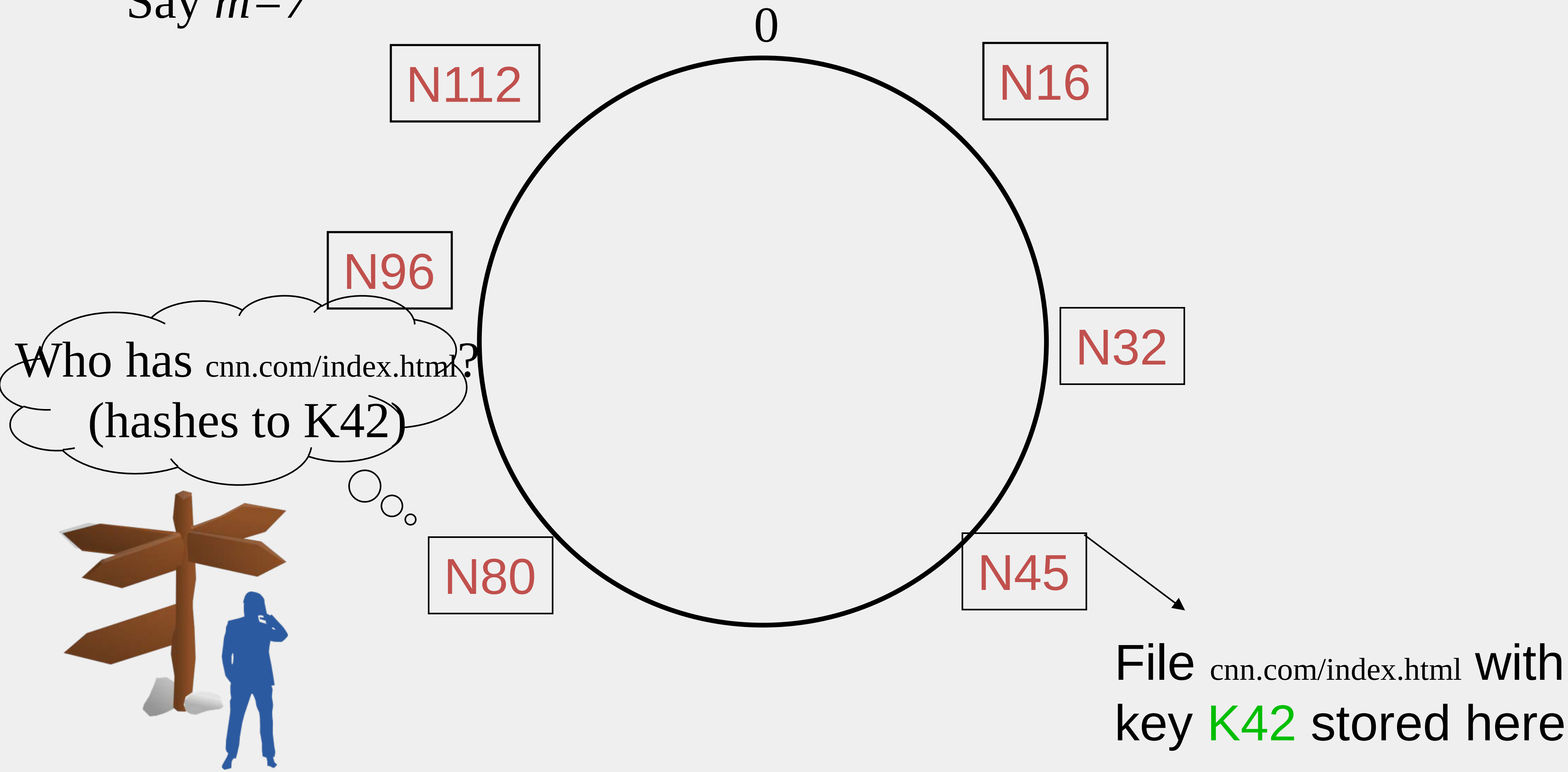
Mapping Files

Say $m=7$



Search

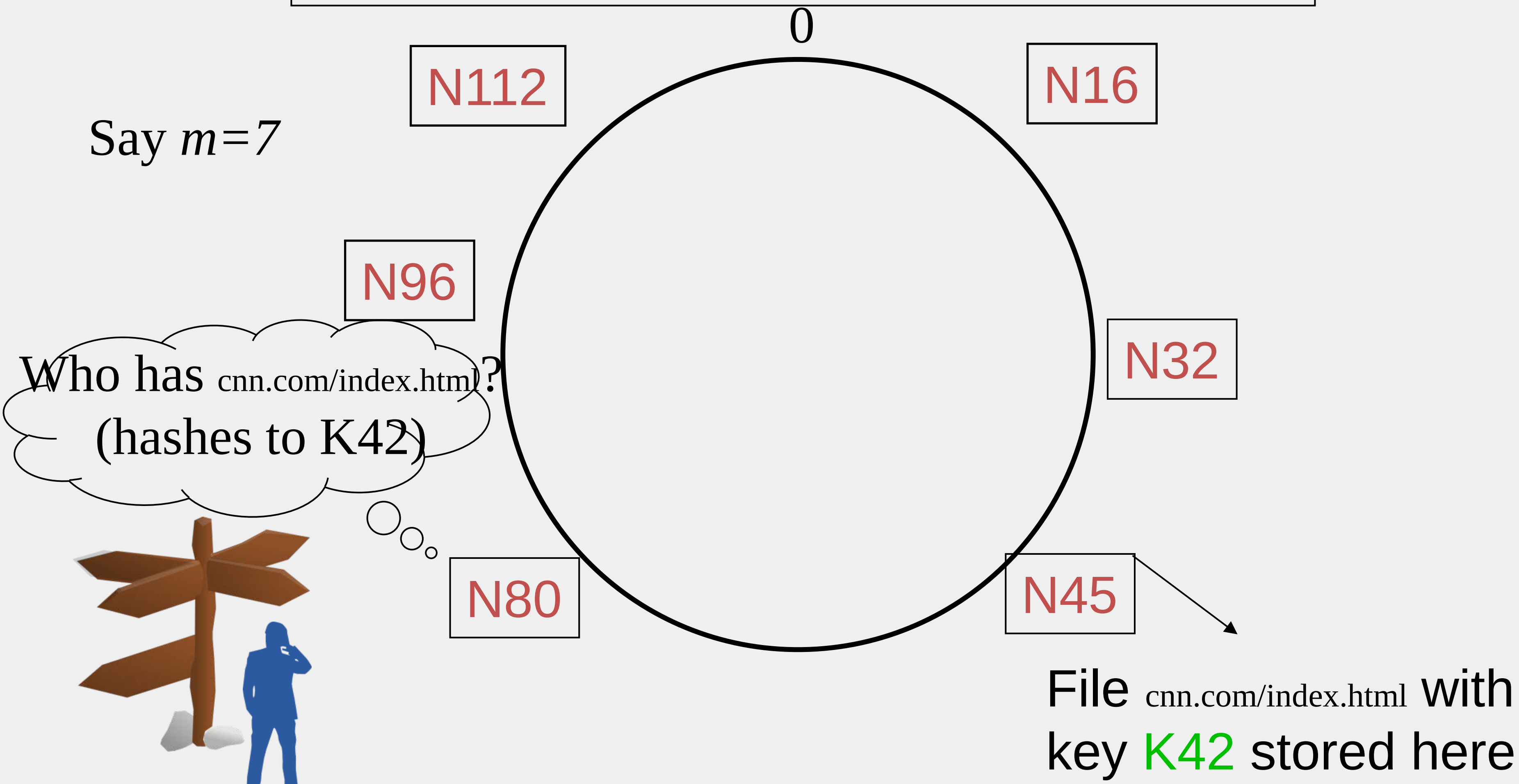
Say m=7



Search

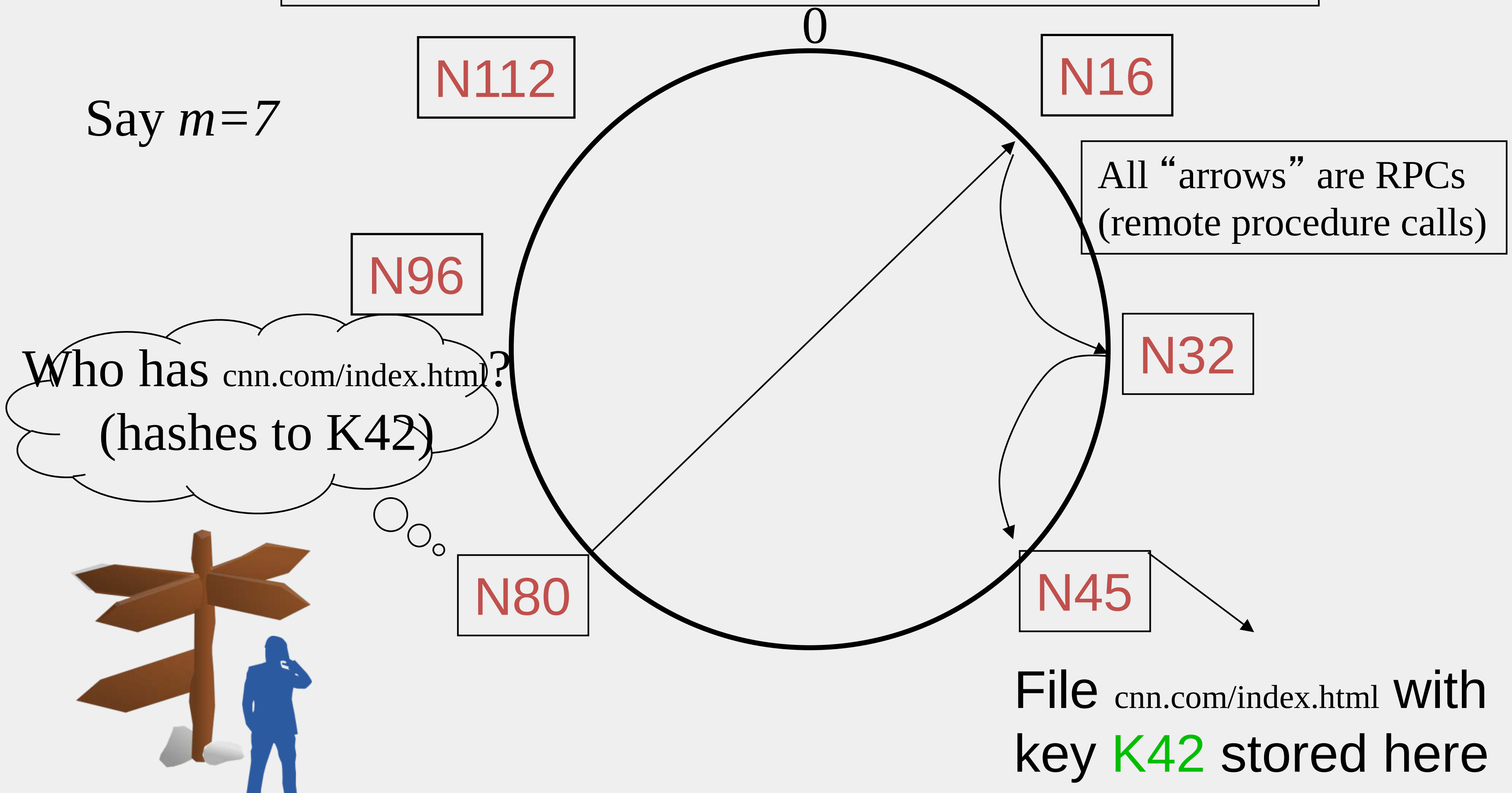
At node *n*, send query for key *k* to largest successor/finger entry <= *k*
if none exist, send query to *successor(n)*

Say $m=7$

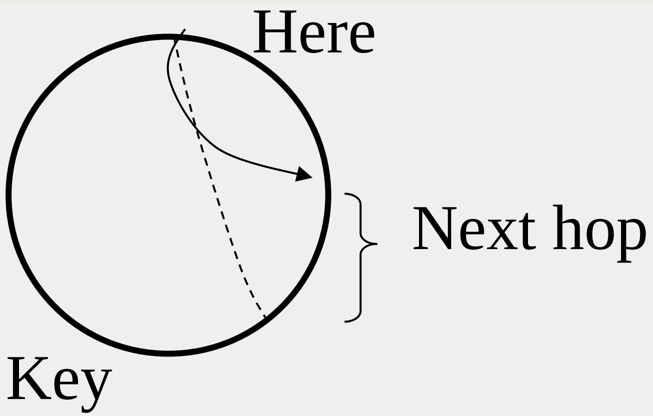


Search

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



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 $2^m / N$

  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