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

DHT == distributed "dict", where value is file
chord == improved consistent hashing where no files need to transfer to node joins/leave, just update distributed routing table

diff versions of DHTs == diff ways to deal w these 4 issues:

1. LB == ring

2. fault == replicate files at multiple successors + 1 node indexing multiple successor's IP

3. eff CRUD == mod 2 eqn

4. local == bad in chord since file transfer everytime a new node joins, tapestry is invented to solve that

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

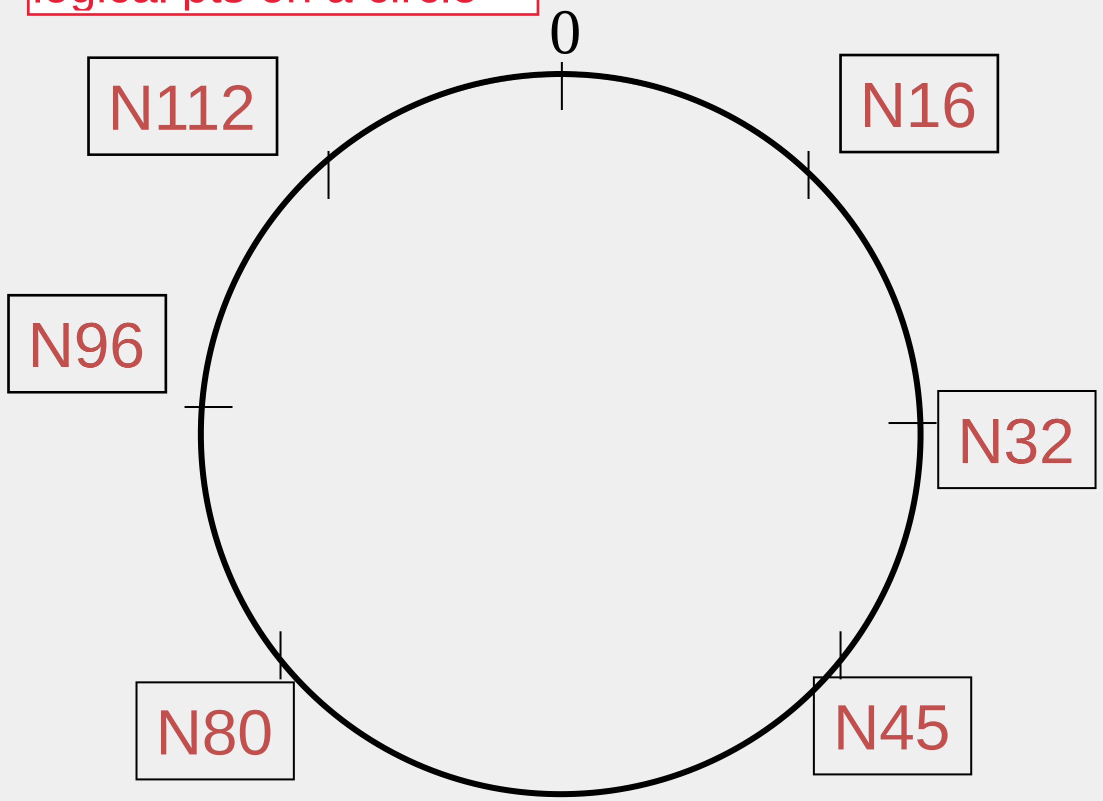
- Developers: I. Stoica, D. Karger, F. Kaashoek, H. Balakrishnan, R. Morris, Berkeley, and MIT
- Intelligent choice of neighbors to reduce latency and message cost of routing (lookups/inserts)
- Uses *Consistent Hashing* on node’ s (peer’ s) address
  - SHA-1(ip_address,port) →160 bit string
  - Truncated 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

m == 2^m -1 no. of logical pts on a circle

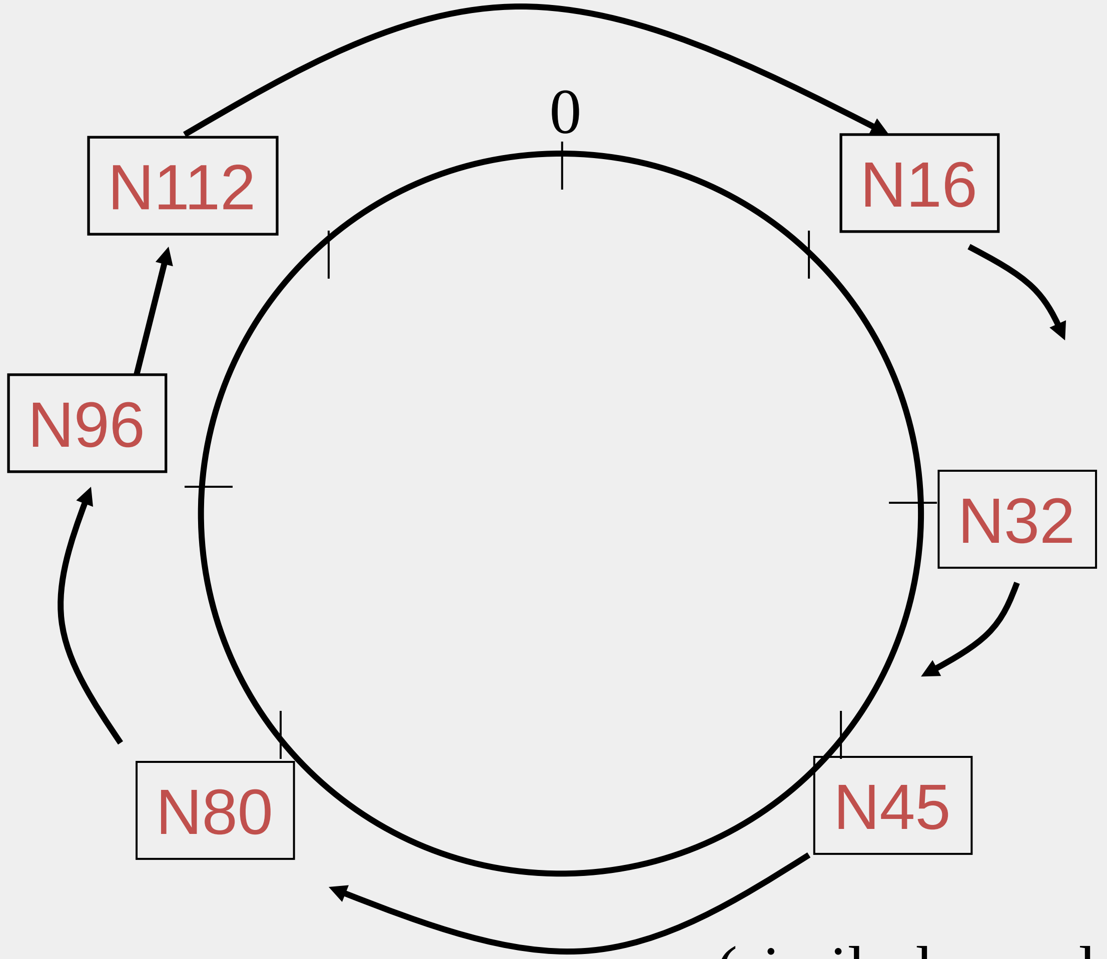
Say *m=7*

6 nodes



Peer pointers (1): successors

Say *m*=*7*

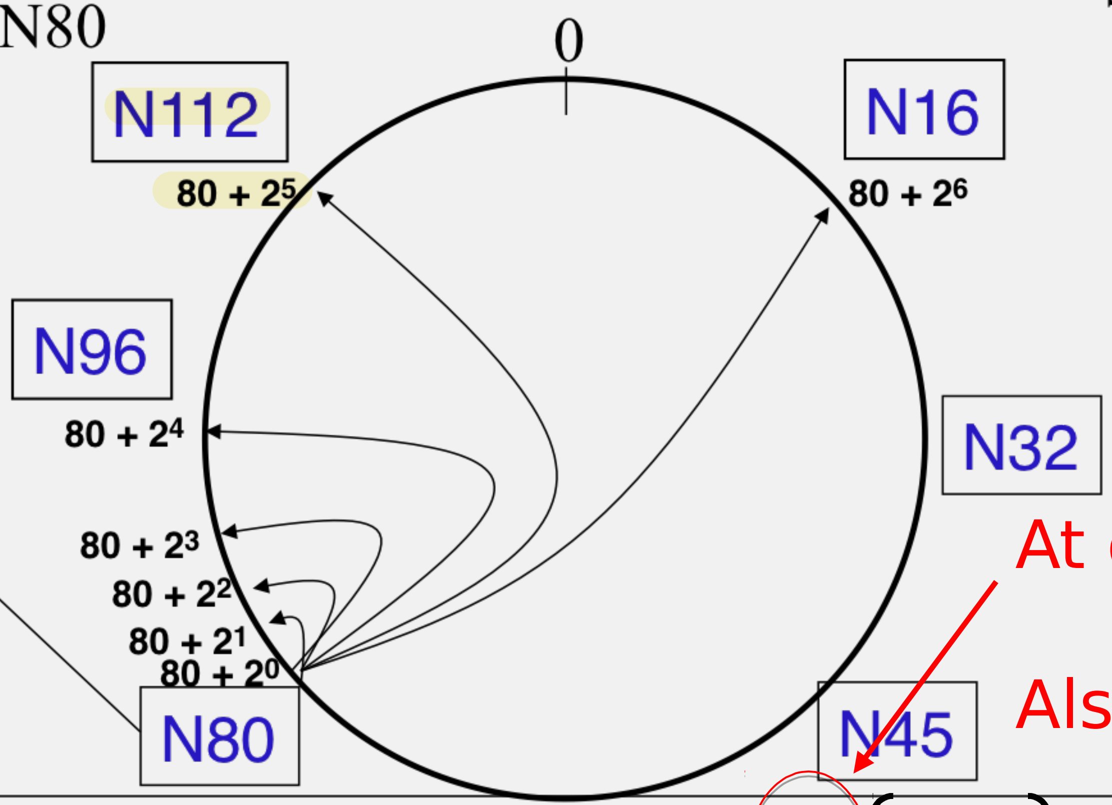


(similarly predecessors)

Peer pointers (2): finger tables

Finger Table at N80

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



Say m=7

1st 5 entries all mapped to node 96

finger table == index successo's IP

but why duplicate 1st 5 entries ??

At or to the clockwise of.

key eqn of finding successor !!!

Also, use (n+2^i) mod 2^m.

ith entry at peer with id n is first peer with id >= $\left[n + 2^i\right](\bmod 2^m)$

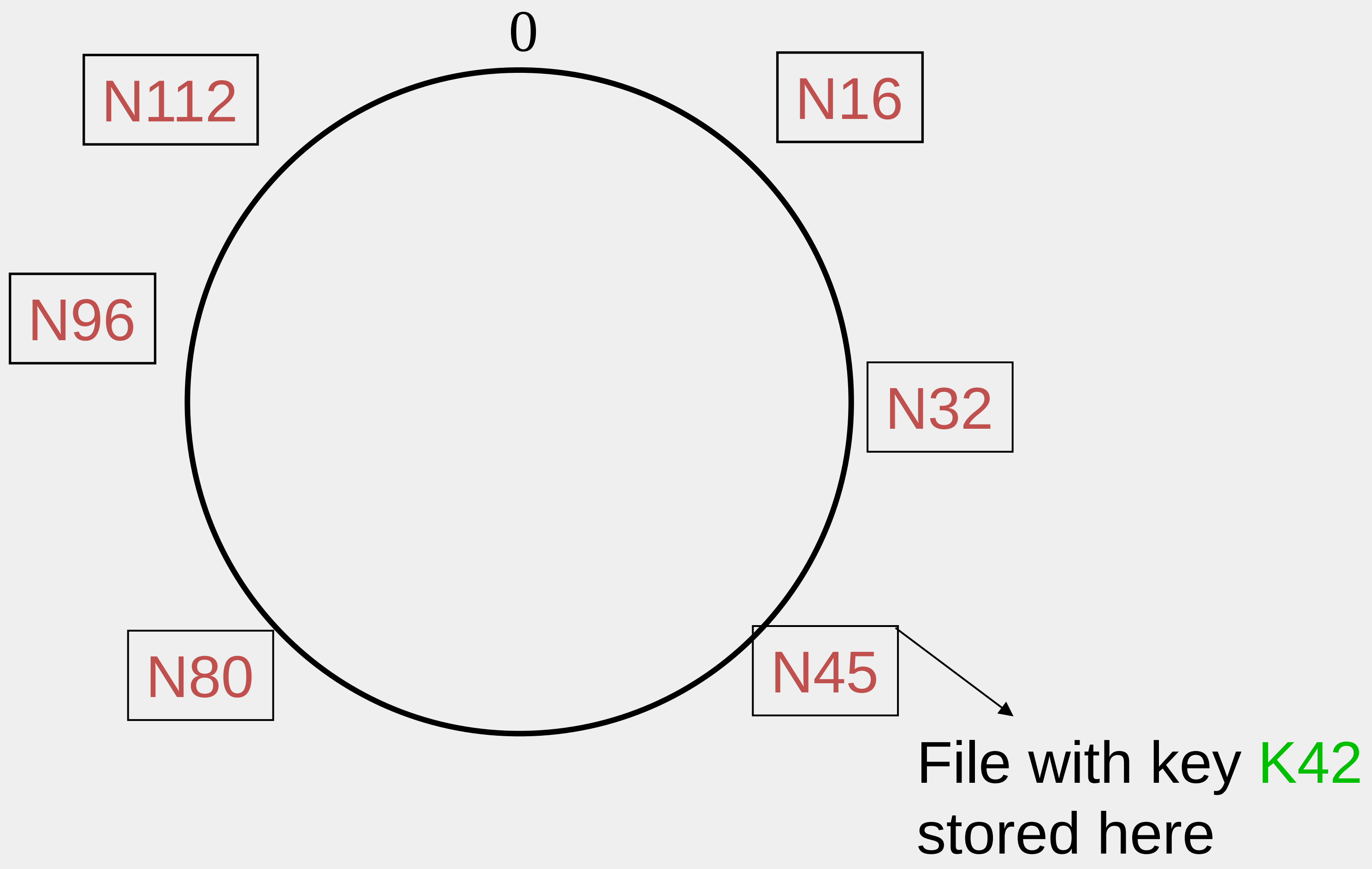
What about the files?

- 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 at or to the clockwise of 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*)

filename is mapped to K42 but stored at N45

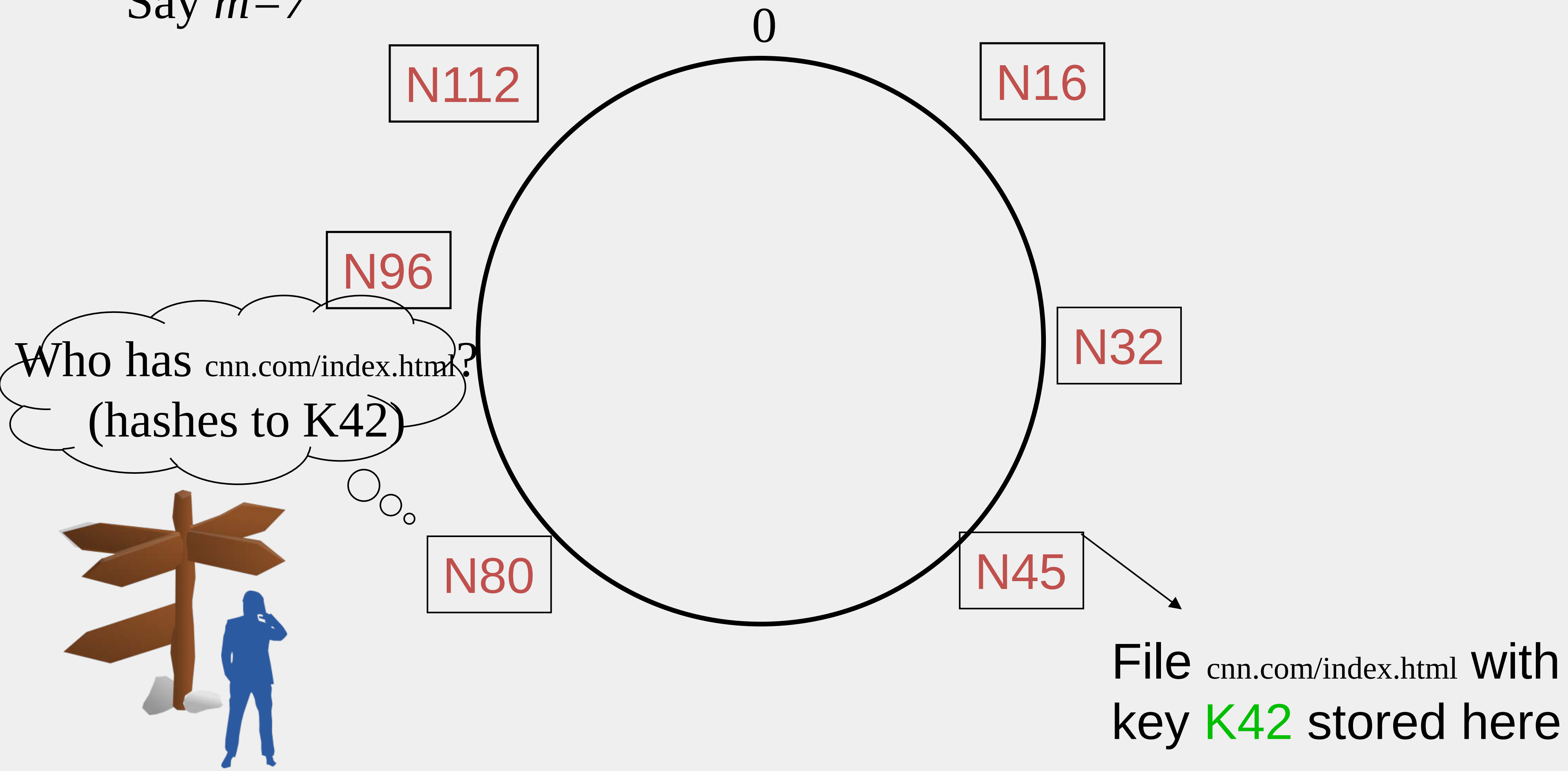
Mapping Files

Say *m=7*



Search

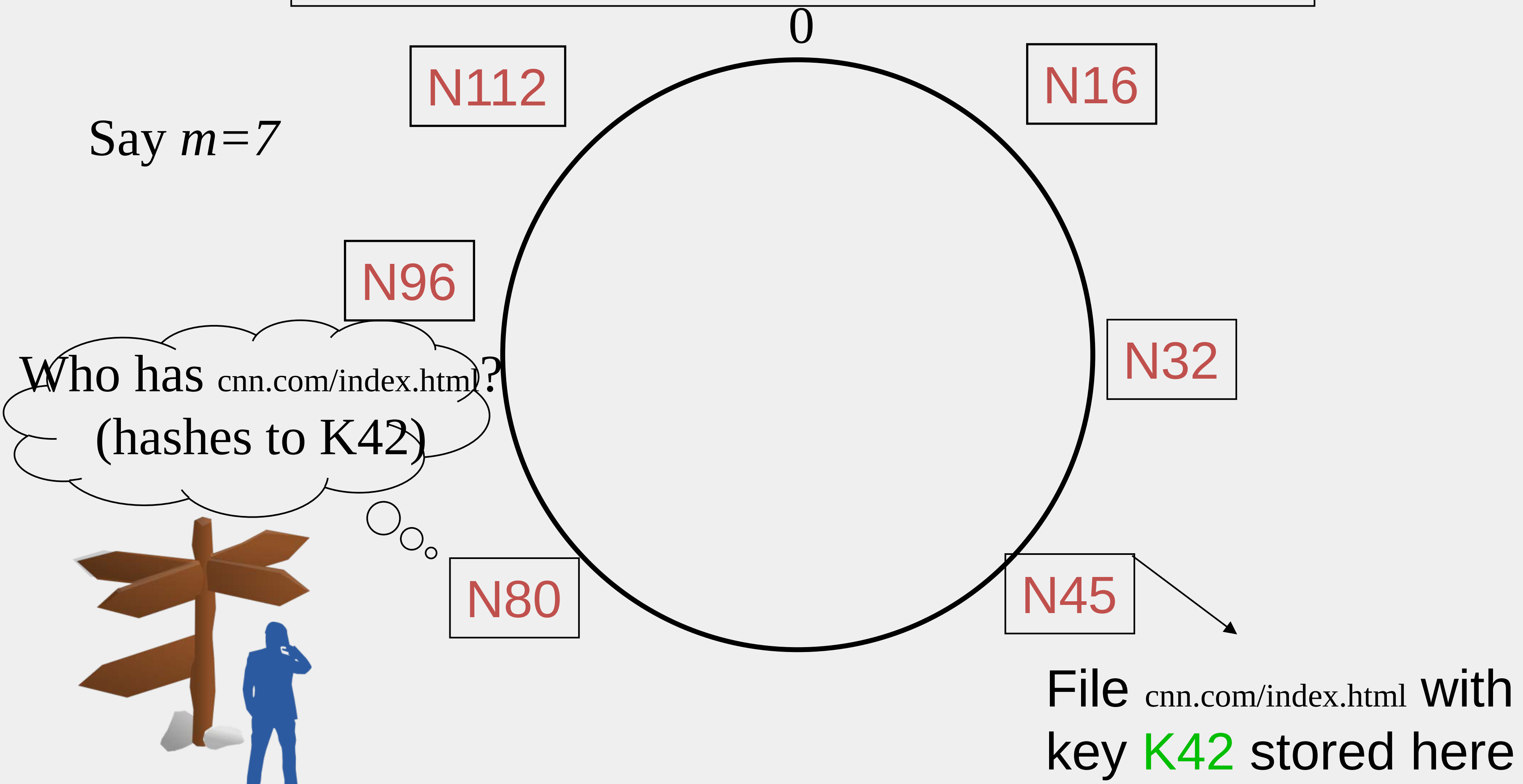
Say *m=7*



Search

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

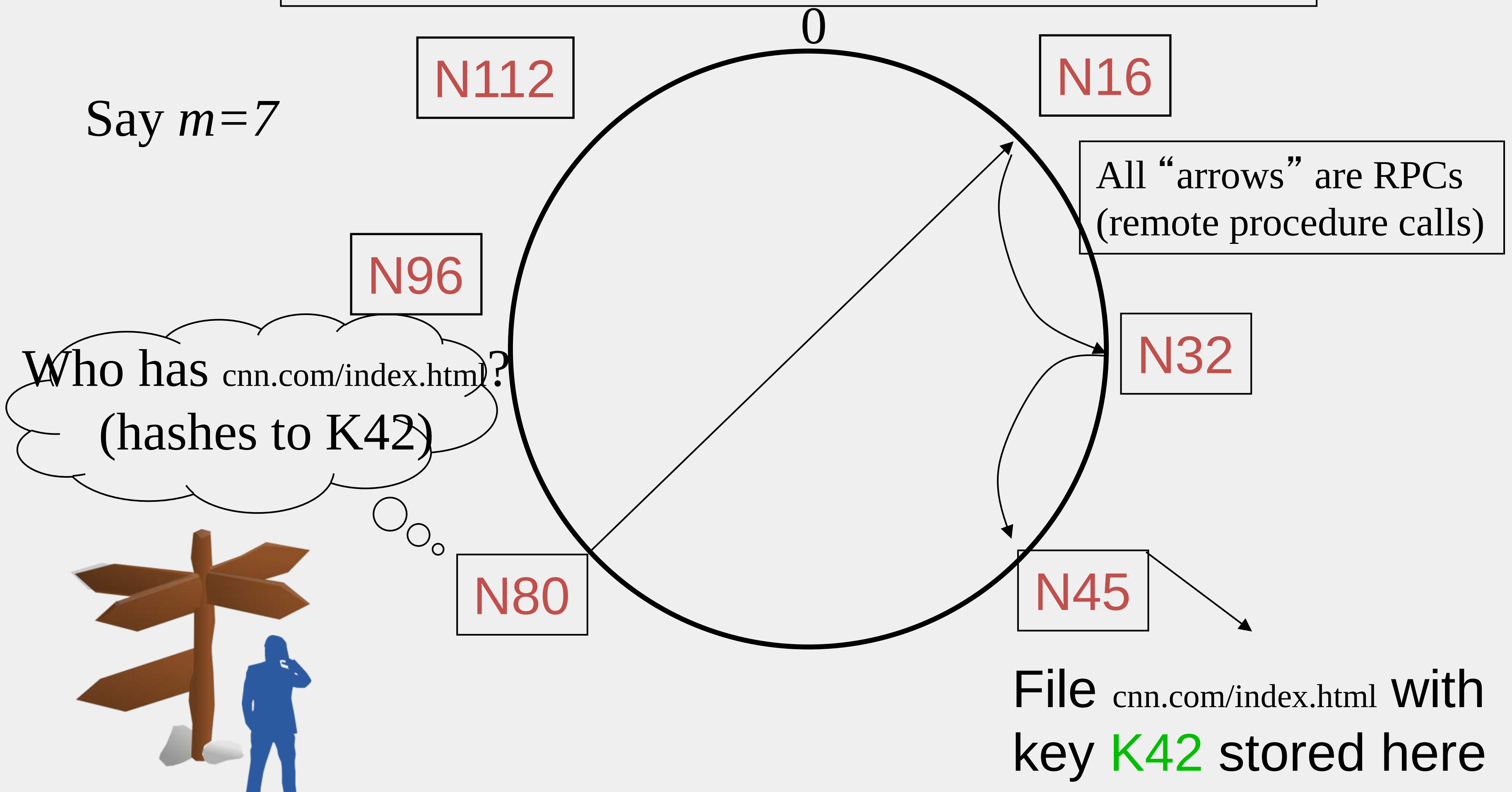
Say *m=7*



Search

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

Say $m=7$



0
N112
N16
N96
N32
N80
N45
All "arrows" are RPCs (remote procedure calls)
Who has cnn.com/index.html? (hashes to K42)
File cnn.com/index.html with key K42 stored here

if client is at N80, filenames to hashed to key K42, it looks up its finger table for closest entry to node N42. thats why happened at N16 n N32. motivation is very quickly get close to target file node.

but at N32 its stuck since its immediate successor N45 is already bigger than K42. if you are lucky, N45 contains K42, then passing query to successor is a bet that works. if NOT, then overshoots, K42 could be in N42/43/44 but N32 doesn't have it in finger table.

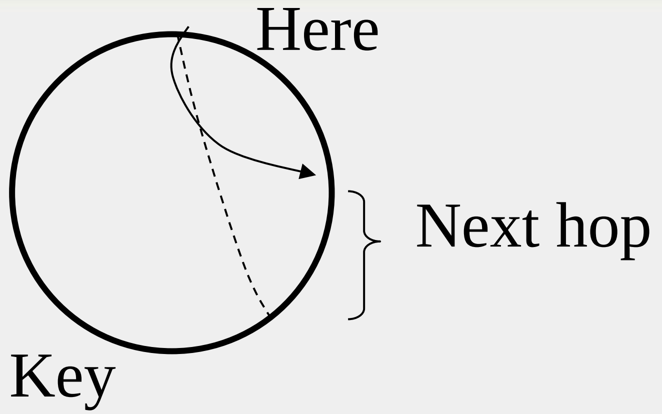
if overshoots, N45 just shoot back to its closest N3x/4x n restart step by step immediate successor again.

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