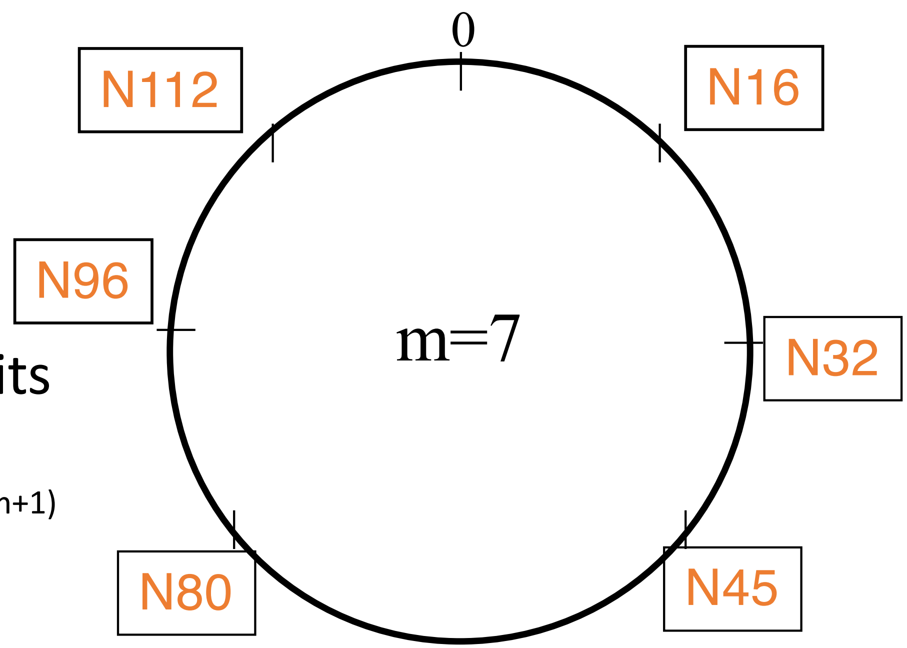
DHT (continued)

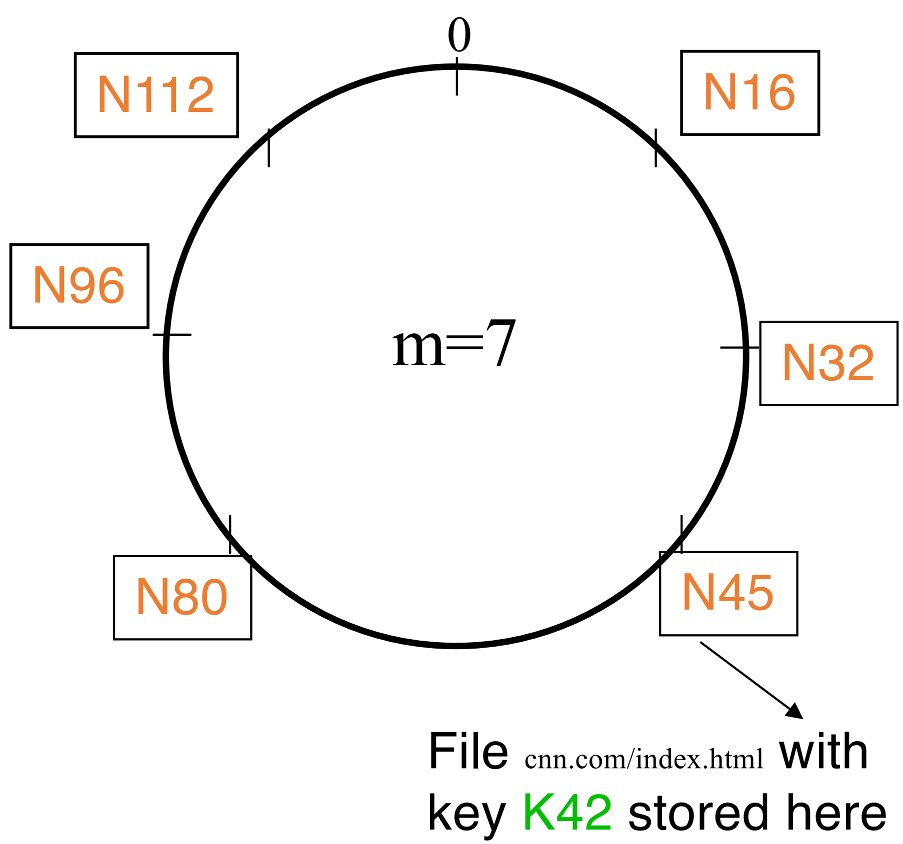
Recap

- Nodes arranged in a *ring* with positions labeled 0..2^m-1
  - m = 128, 160, 256
- A node's *position* is based on a hash of its identity
  - P(collision among N nodes) =~ $1-e^{-n(n-1)/2^(m+1)}$
  - For m=128, N=1,000,000,000, this is ~= 0.0000000000000000000001469…



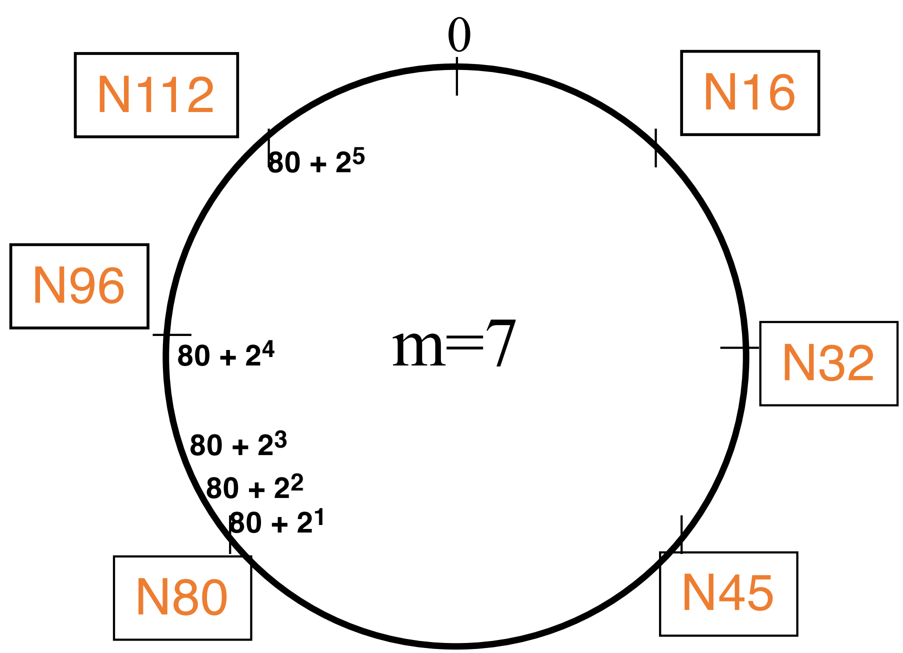
Recap

- Nodes arranged in a *ring* with positions labeled 0..2^m-1
- A node's *position* is based on a hash of its identity
- A key *x* is stored at node with first position greater than x on the ring
  - Each node stores approximately 1/N of all keys



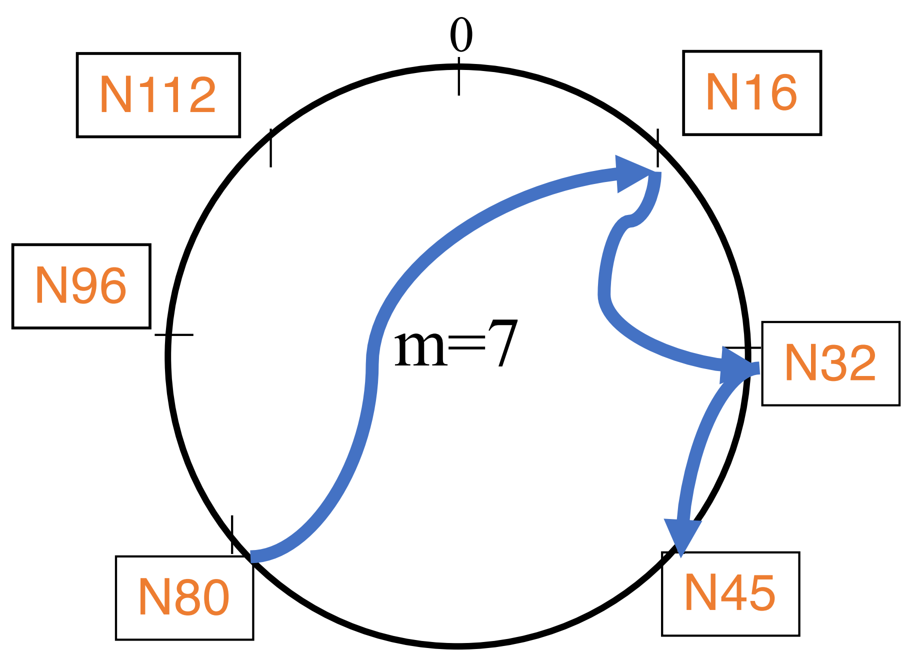
Recap

- Nodes arranged in a *ring* with positions labeled $0..2^m-1$
- A node's *position* is based on a hash of its identity
- A key *x* is stored at node with first position greater than x on the ring
- A node's fingers are based on $id + 2^i$ $(mod\ 2^m)$ for i = 0, ..., m-1
    - Up to m fingers, though only O(log N) distinct on average

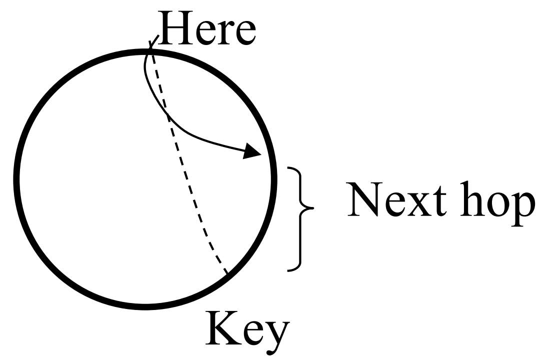


Recap

- Nodes arranged in a *ring* with positions labeled 0..2^m-1
- A node's *position* is based on a hash of its identity
- A key *x* is stored at node with first position greater than x on the ring
- A node's fingers are based on *id + 2^i (mod 2^m)* for i = 0, ..., m-1
- Search for key x proceeds by using largest finger that makes progress towards key



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* (why?)

  Takes at most *m* steps: 2^m is at most a constant multiplicative factor above *N,* lookup is *O(log(N))*
- (intuition): after *log(N)* forwardings, distance to key is at most $2^m / N$ (why?)

  Number of node identifiers in a range of $2^m / N$

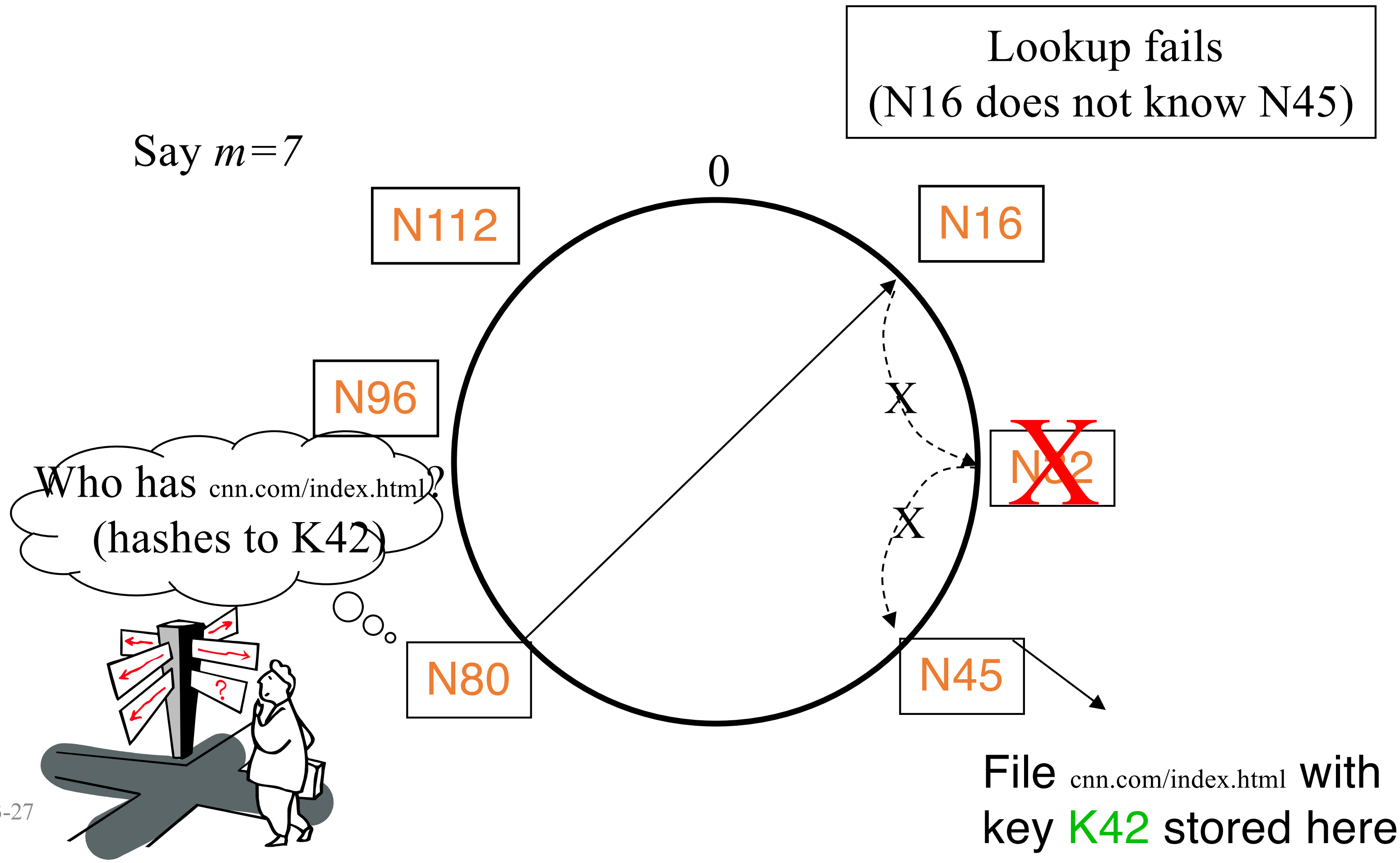
  is *O(log(N))* with high probability (why?)

  So using *successor*s in that range will be ok

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

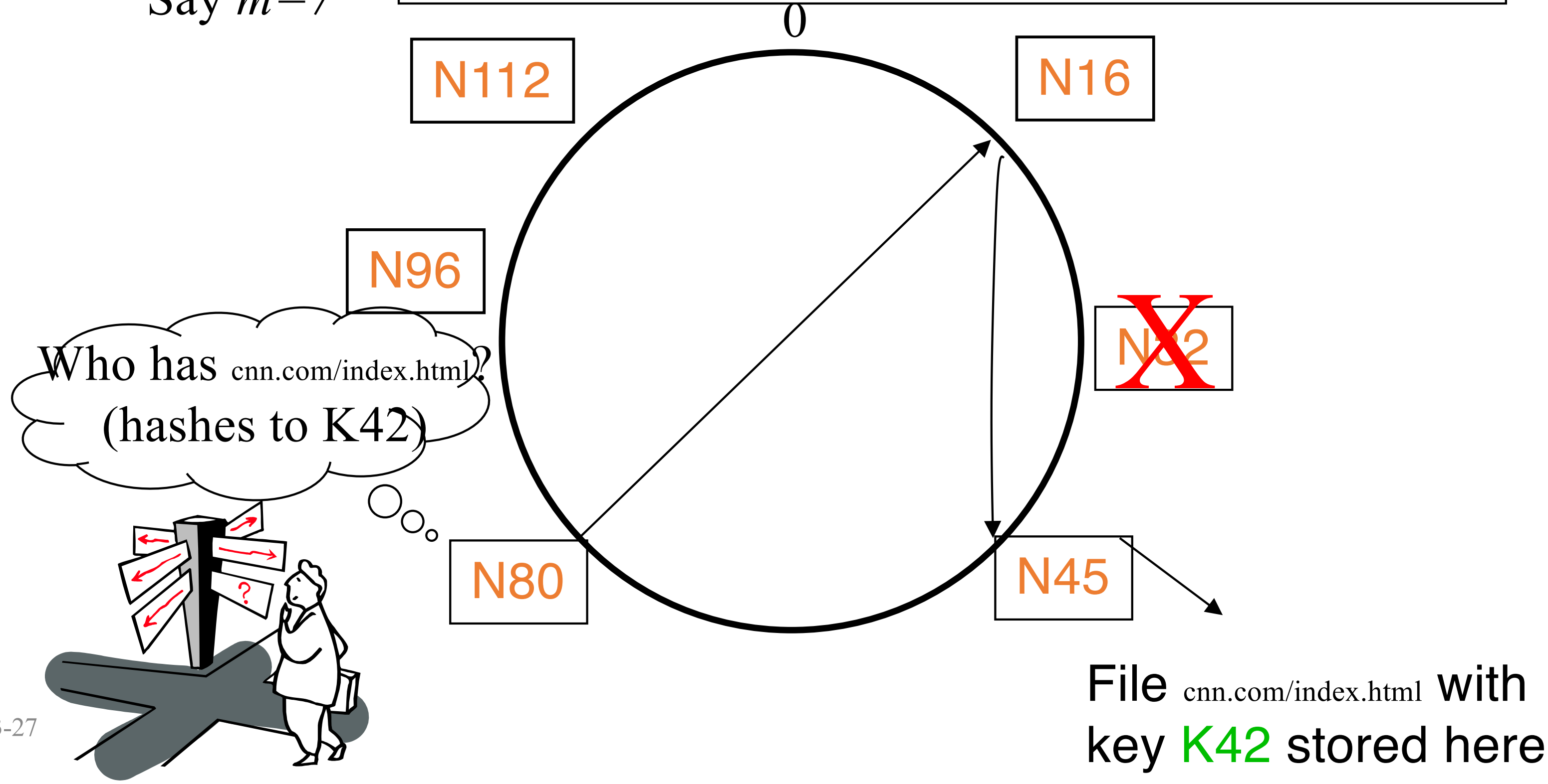
Search under peer failures



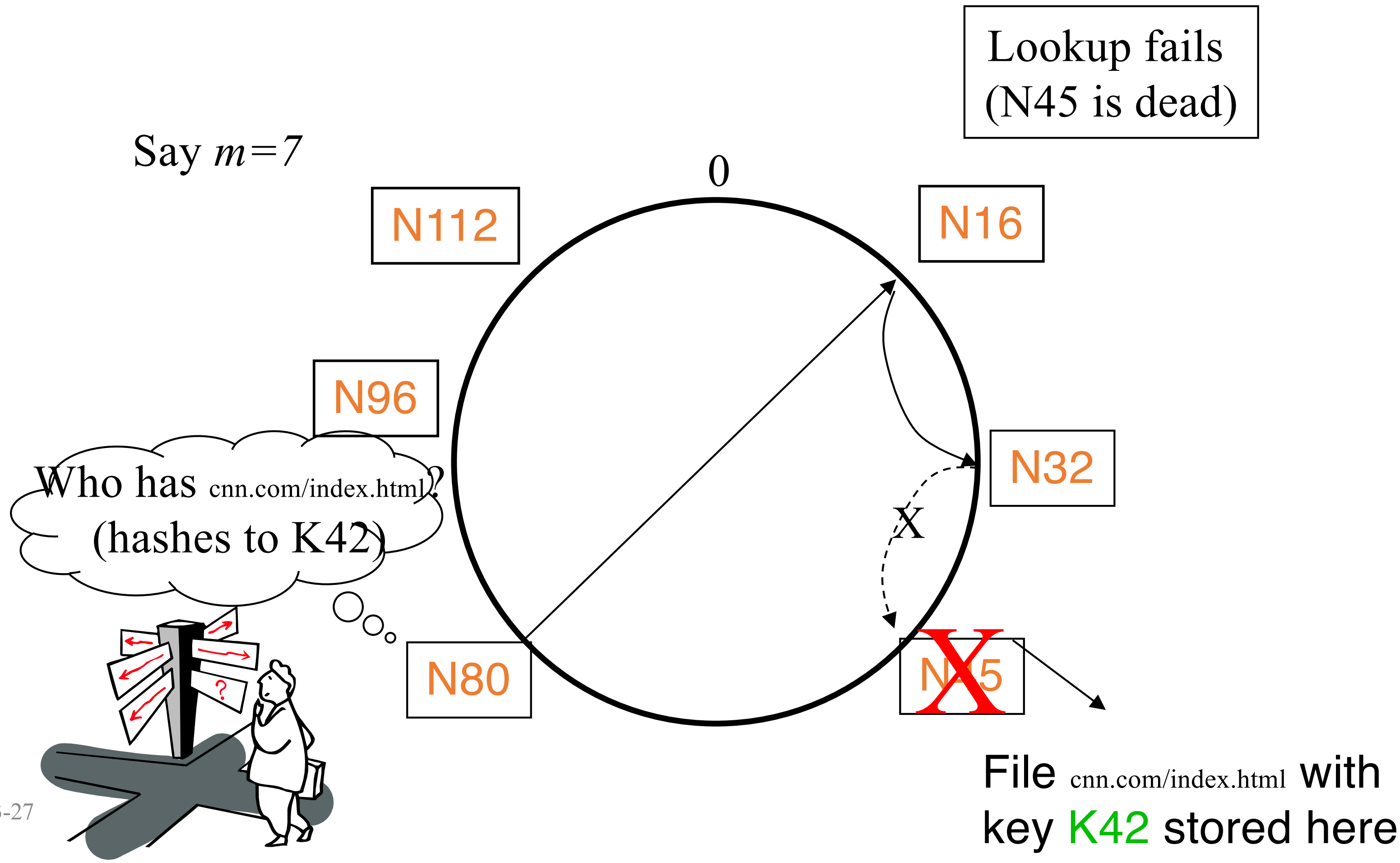
Search under peer failures

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

Say m=7



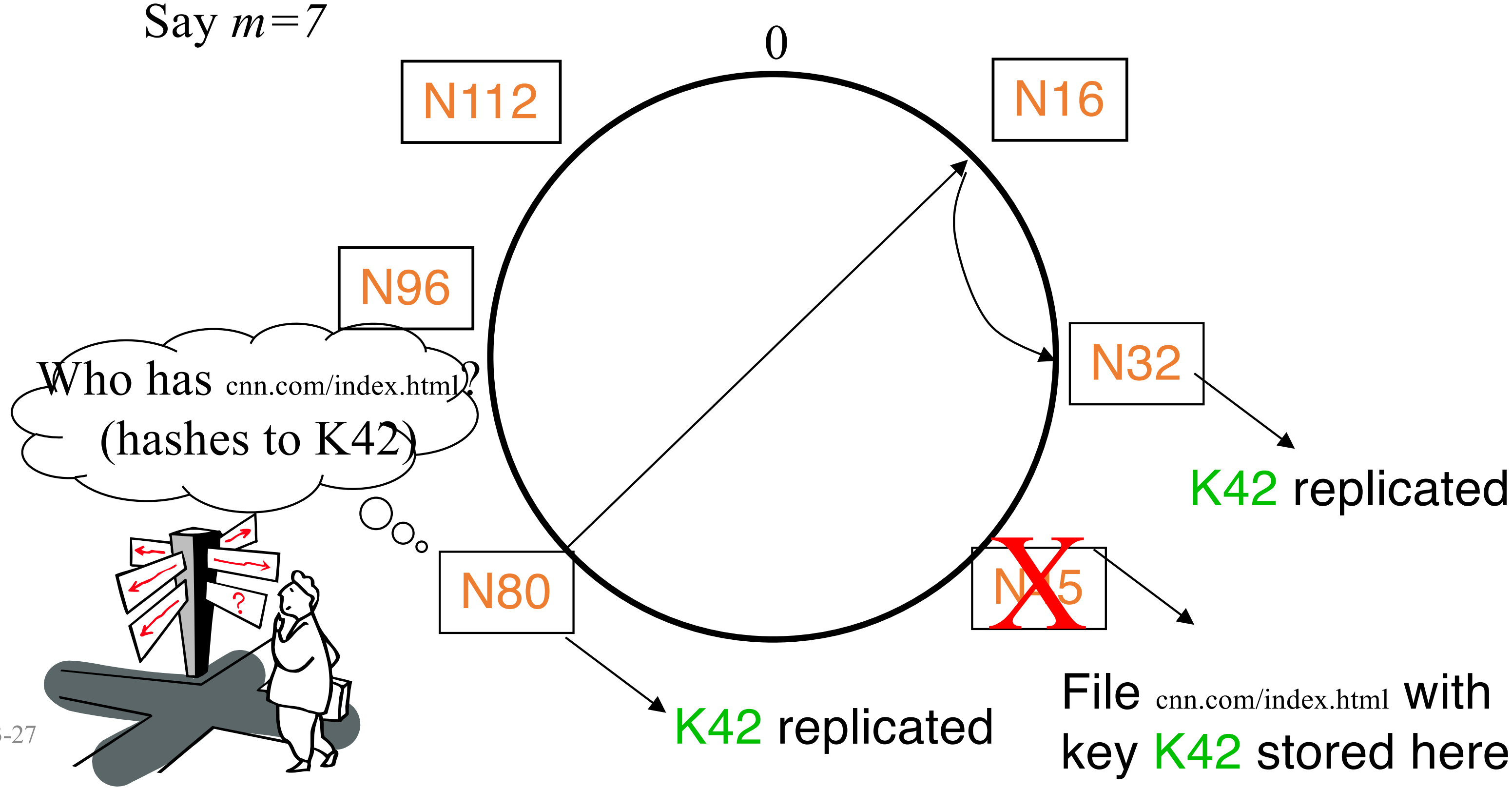
Search under peer failures (2)



Search under peer failures (2)

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

Say m=7



Need to deal with dynamic changes

✓Peers fail

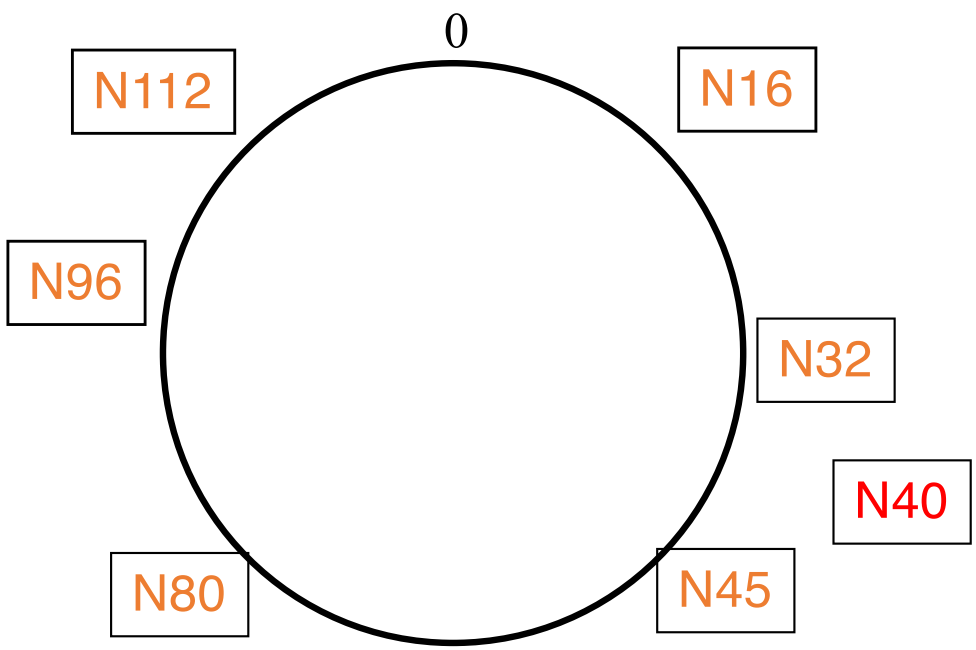
- New peers join
- Peers leave
  - P2P systems have a high rate of *churn* (node join, leave and failure)

→ 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

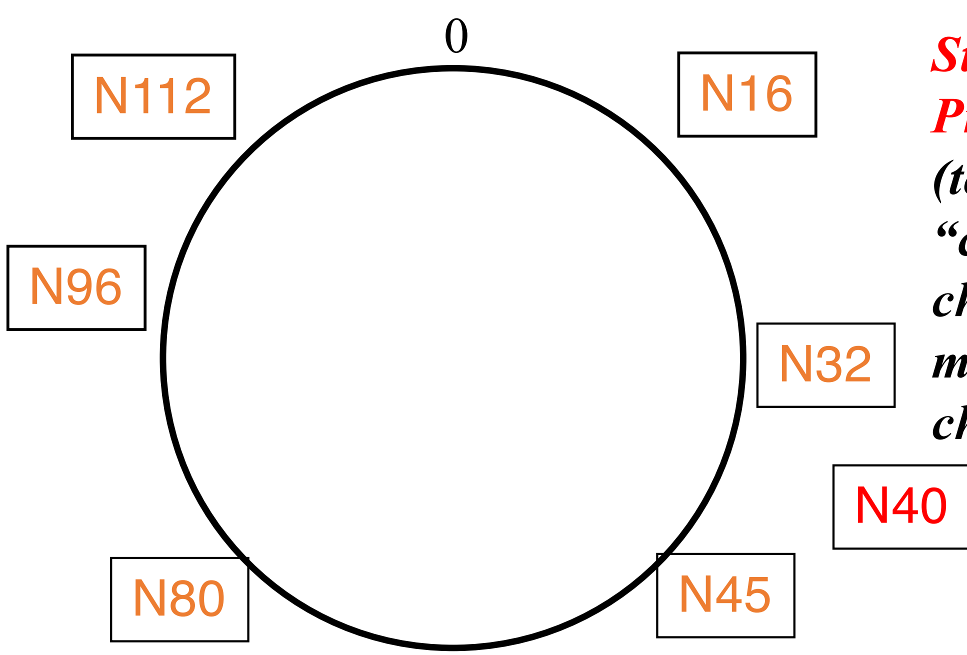
Say *m=7*



New peers joining

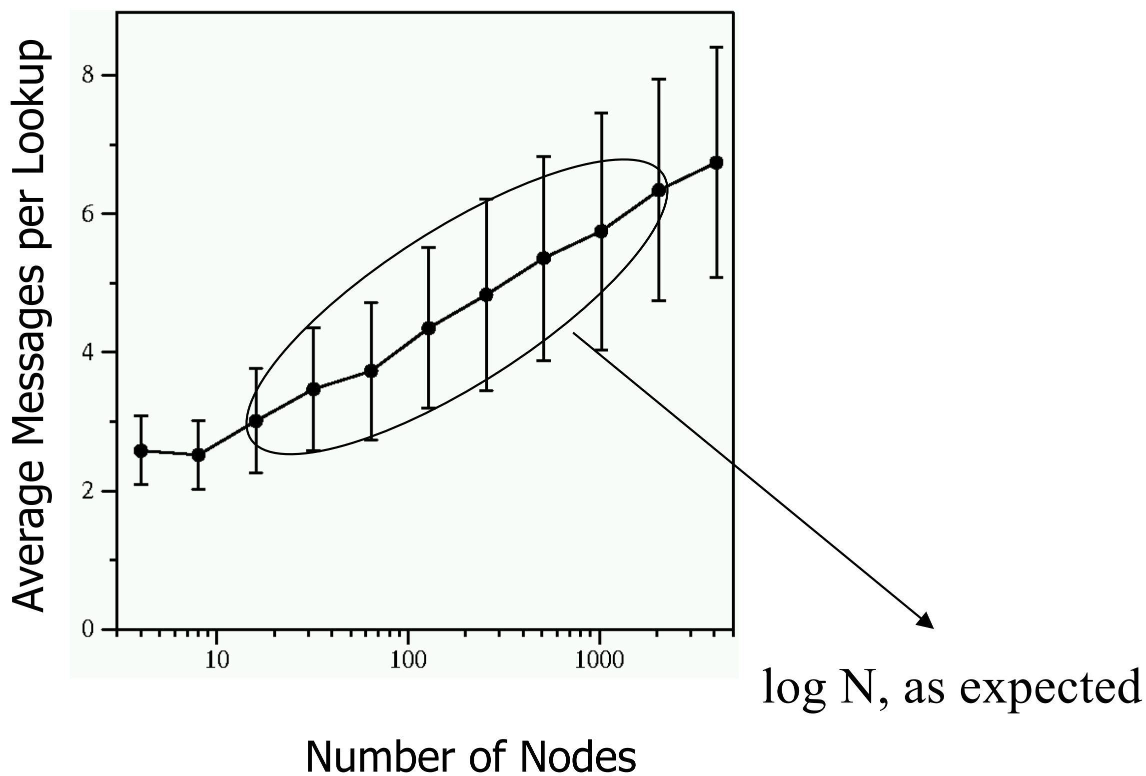
Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it
N40 periodically talks to its neighbors to update finger table

Say $m=7$



Stabilization Protocol
(to allow for "continuous" churn, multiple changes)

Lookups


Average Messages per Lookup
0
2
4
6
8
10
100
1000
Number of Nodes
log N, as expected

Chord Protocol: Summary

- $O(log(N))$ memory and lookup costs
- Hashing to distribute filenames uniformly across key/address space
- Allows dynamic addition/deletion of nodes

DHT Deployment

- Many DHT designs
  - Chord, Pastry, Tapestry, Koorde, CAN, Viceroy, Kelips, Kademlia, …
- Slow adoption in real world
  - Most real-world P2P systems unstructured
    - No guarantees
    - Controlled flooding for routing
  - Kademlia slowly made inroads, now used in many file sharing networks
- Distributed key-value stores adopt some of the ideas of DHTs
  - Dynamo, Cassandra, etc.