# Chord

## Advanced issues

# Analysis

**Theorem. Search takes O (log N) time**

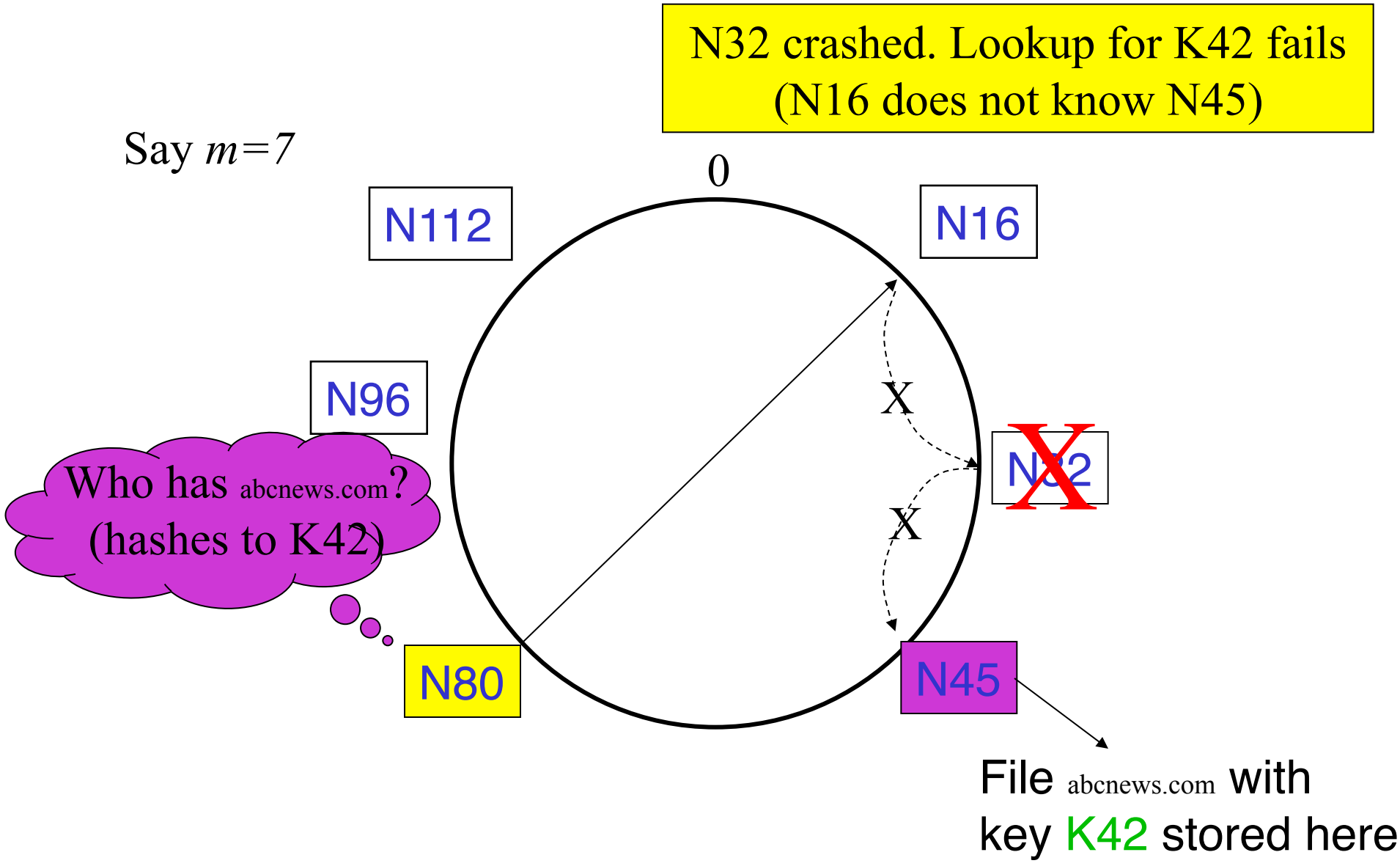
**(Note that in general, $2^m$ may be much larger than N)**

Proof. After *log N* forwarding steps, distance to key is at most $2^m / N$ (N= $2^{\log N}$). Number of nodes in the remaining range is O (log N*)* with high probability (this is property of consistent hashing). So by using *successor*s in that range, it will take at most an additional O (log N) forwarding steps.

# **Analysis** (contd.)

O(log N) search time  is true if finger and successor entries correct, But what if these entries are wrong (which is possible during join or leave operations, or process crash?

# Search under peer failures

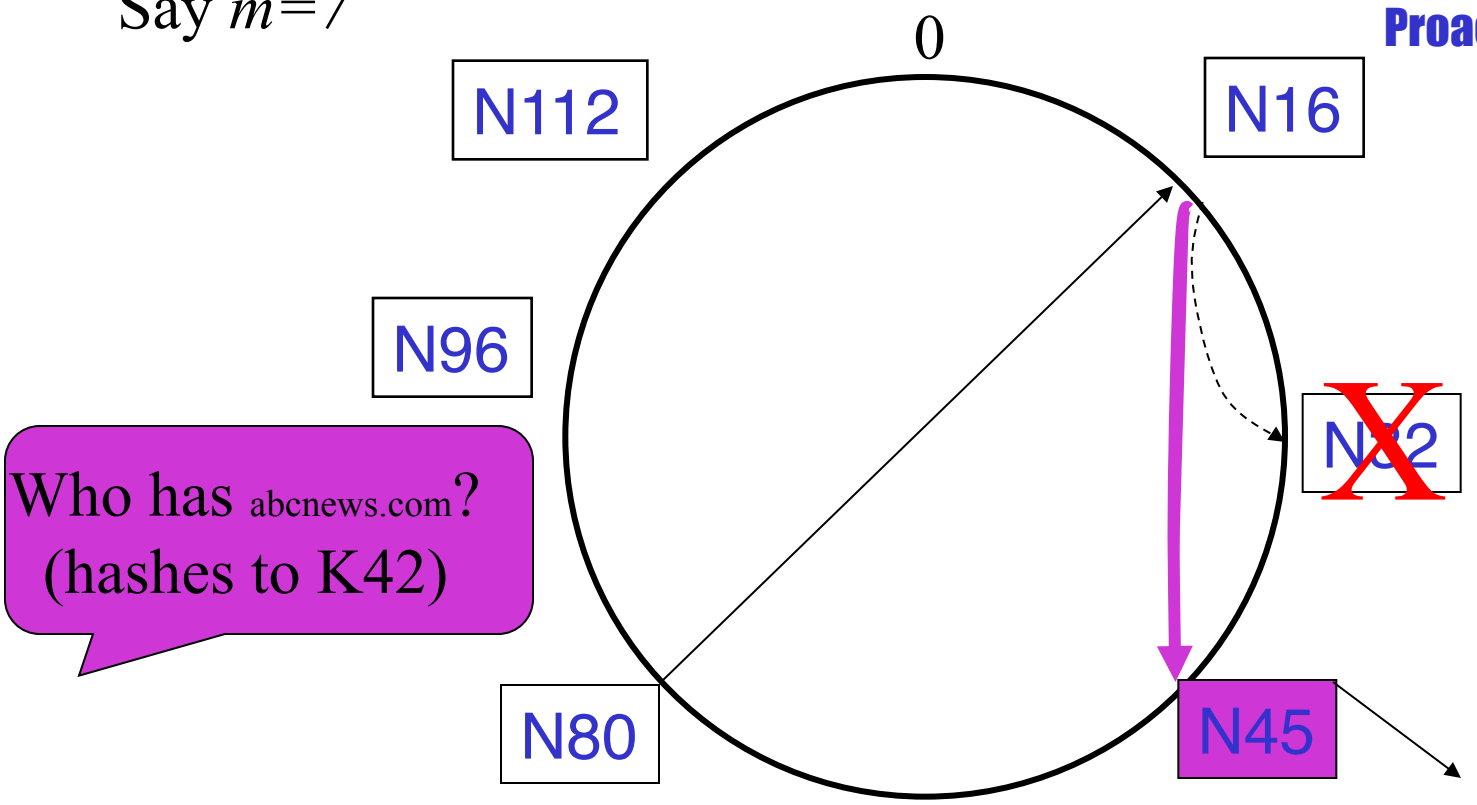


# Search under peer failures

One solution: maintain ***r* multiple *successor* entries**
in case of a failure, use other successor entries.

Say $m=7$

**Reactive vs.**
**Proactive approach**



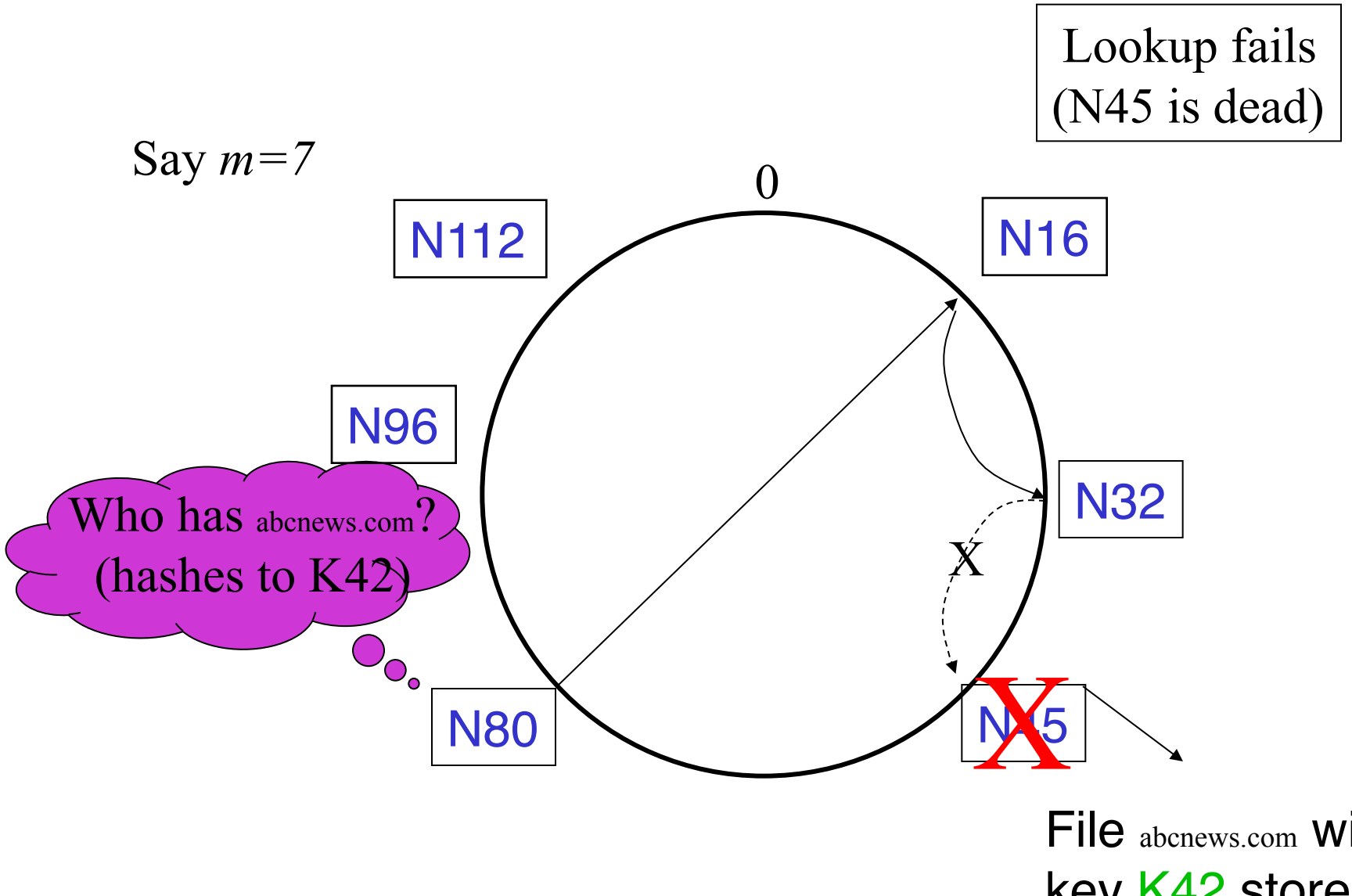

File abcnews.com with
key K42 stored here

# Search under peer failures

Choosing r=2log(N) suffices to maintain the correctness "with high probability." Say 50% of nodes fail (i.e prob of failure = ½). For a given node, Probability (at least one successor alive) =

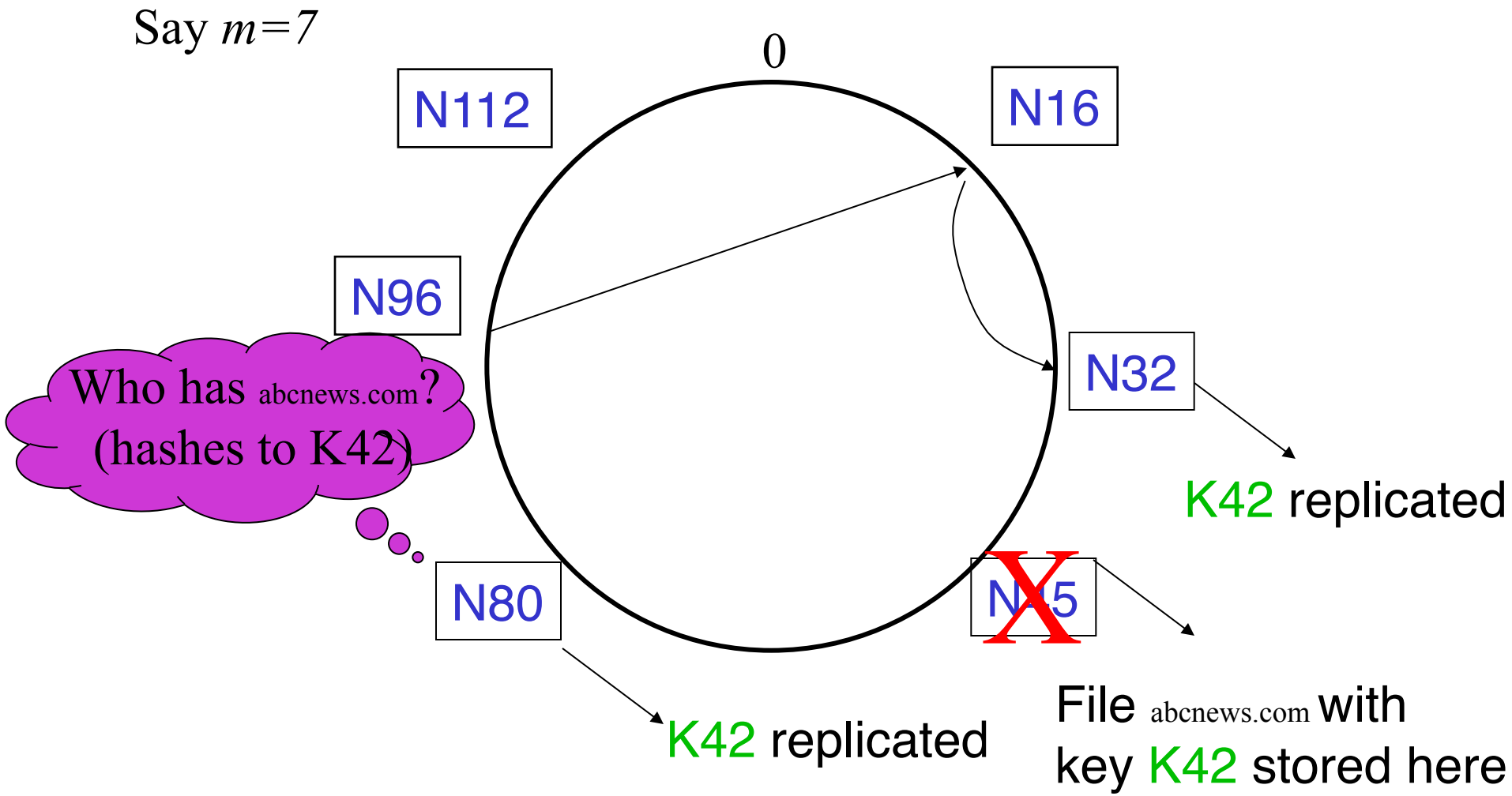
$$1-(\frac{1}{2})^{2\log N} = 1-\frac{1}{N^2}$$

# Search under peer failures (2)



# Search under peer failures (2)

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



# Dealing with dynamic issues

Peers fail

New peers join

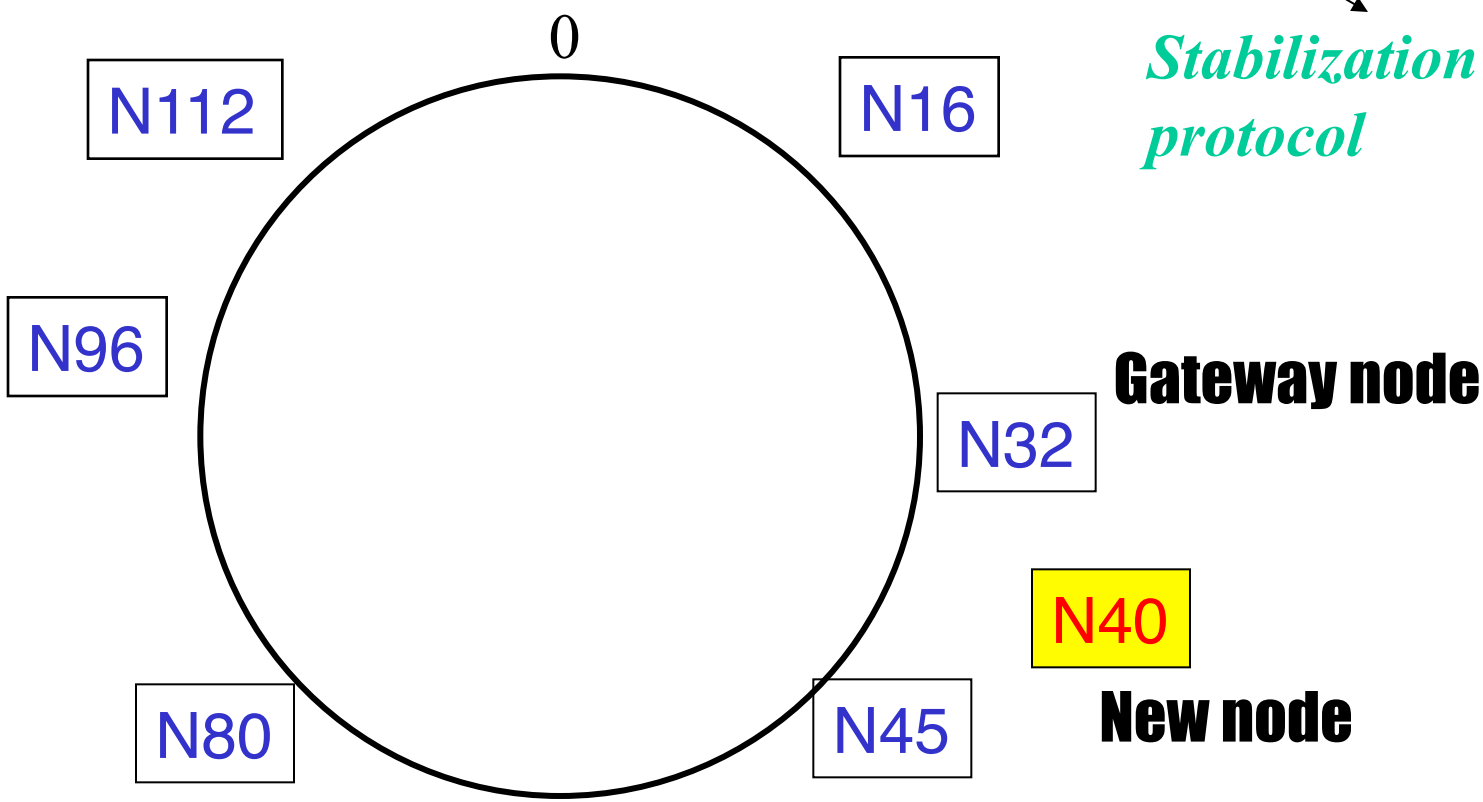
Peers leave

Need to update *successor*s and *finger*s, and ensure keys reside in the right places

# New peers joining

Some gateway node directs **N40** to its successor N45
N32 updates successor to N40
N40 initializes successor to N45, and obtains fingers from it
*N40 periodically talks to neighbors to update finger table*

Say $m=7$

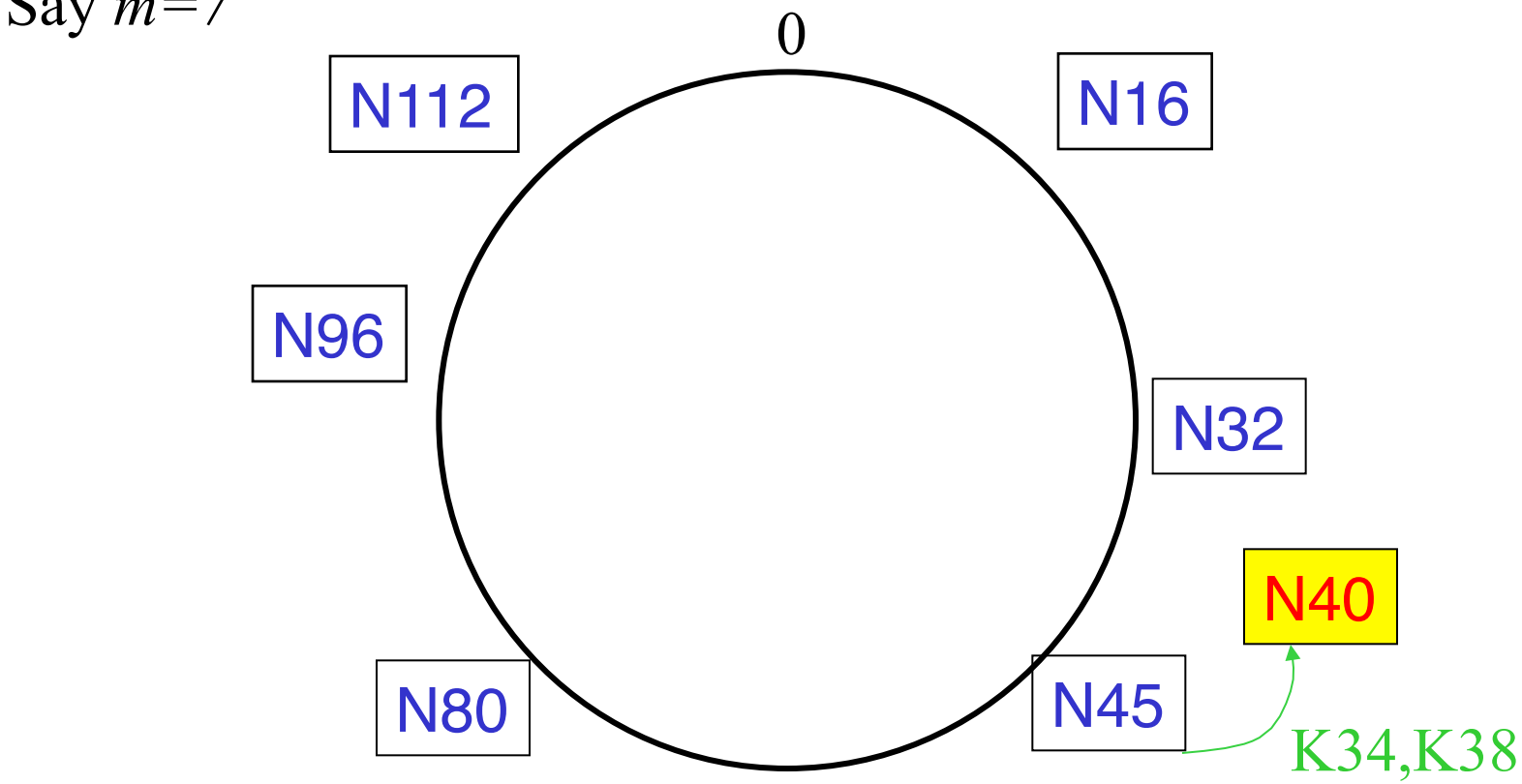



*Stabilization protocol*

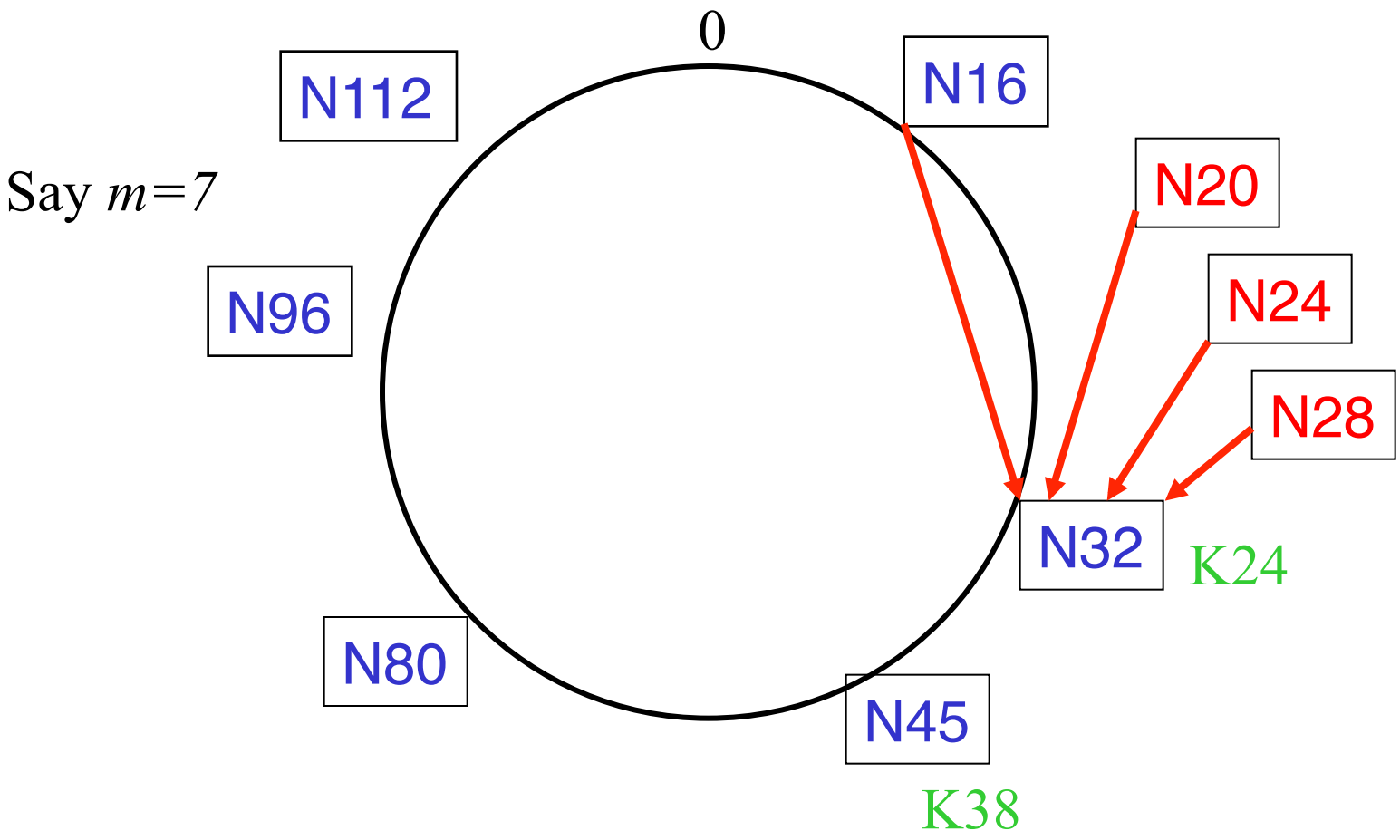
# New peers joining (2)

N40 may need to copy some files/keys from N45
(files with fileid between 32 and 40)

Say $m=7$



# Concurrent join

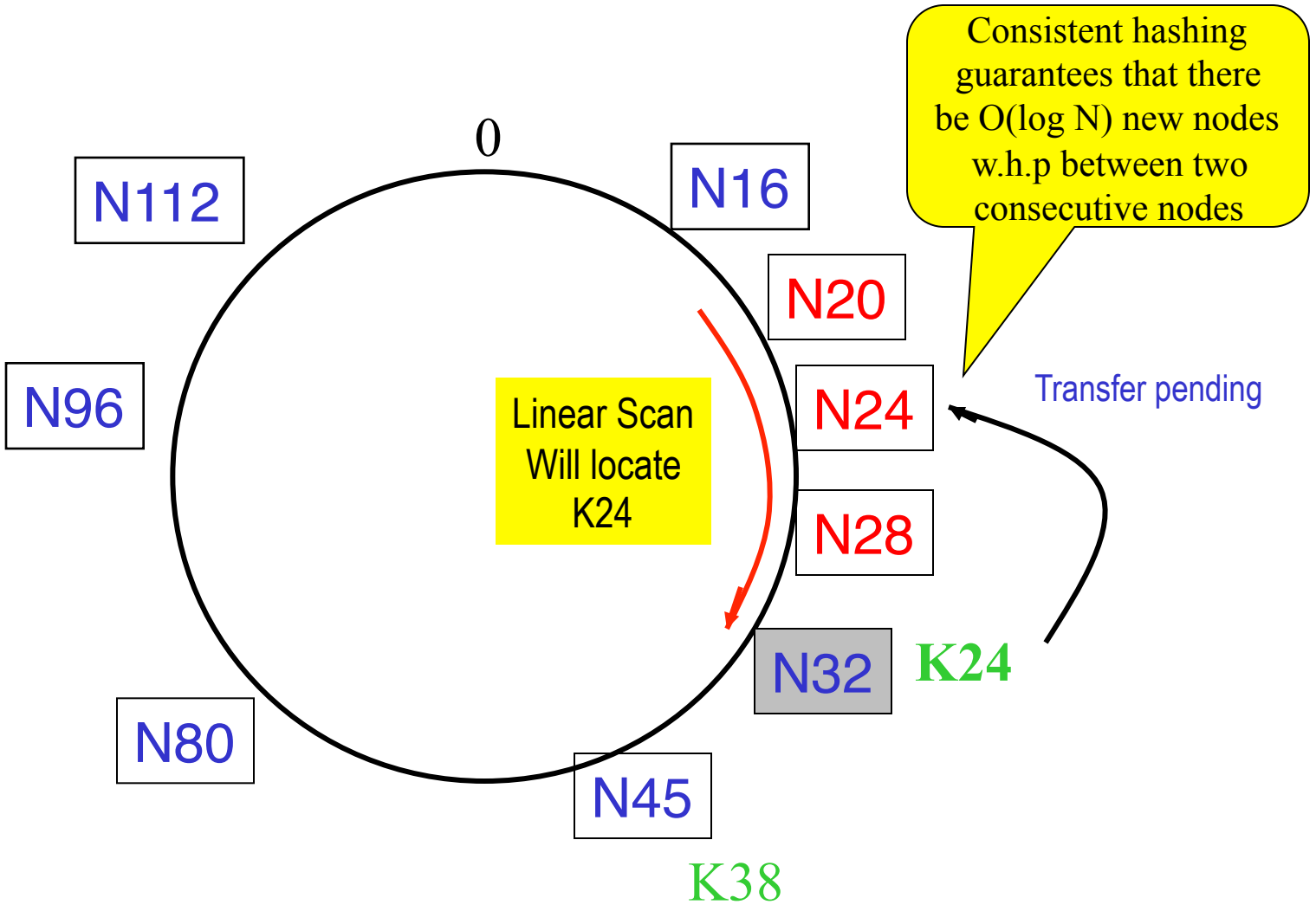



Argue that each node will eventually be reachable

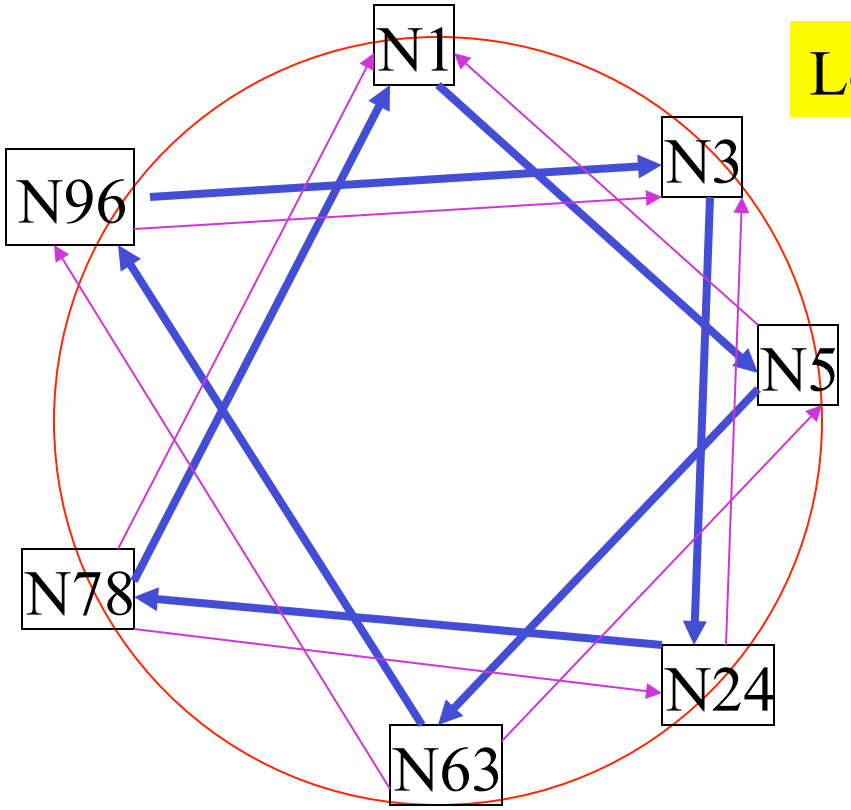
# Effect of join on lookup

If in a stable network with N nodes, another set of N nodes joins the network, and the join protocol correctly sets their successors, then lookups will take O(log N) steps w.h.p

# Effect of join on lookup



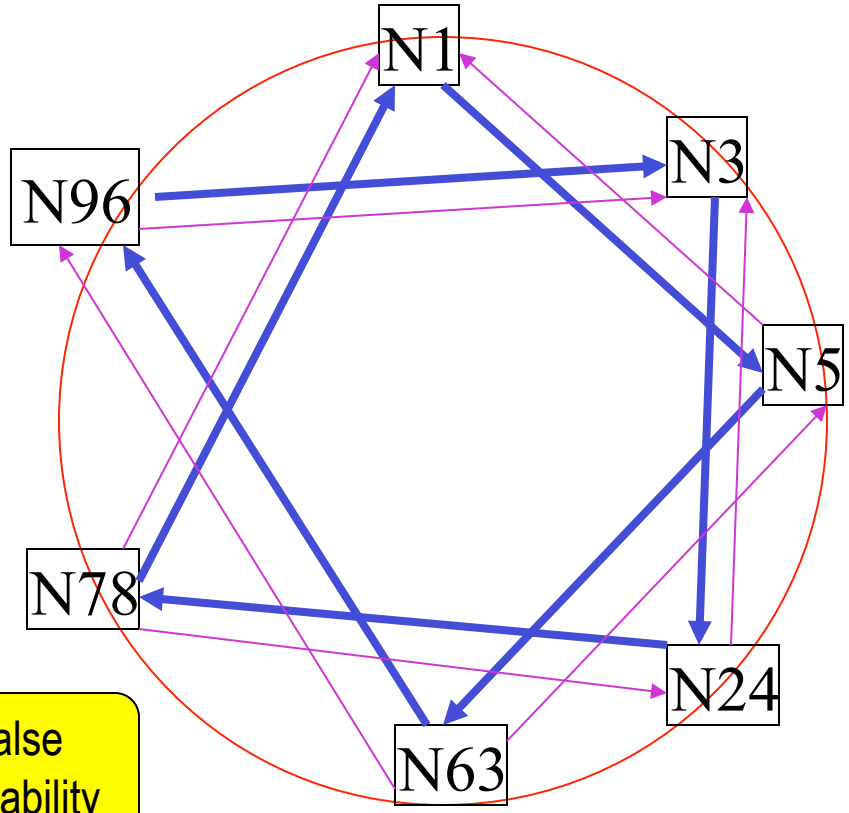
# Weak and Strong Stabilization




Loopy network

For all u: (successor (predecessor (u))) = u. Still it is **weakly stable** but not **strongly stable**. Why?

# Loopy network




(succ (pred (u))) = u

stable

Must be false
for strong stability

What is funny / awkward about this?

**There is a v: u < v < successor (u)** **(Weakly stable)**

# Strong stabilization

The key idea of recovery from loopiness is: Let each node u ask its successor to walk around the ring until it reaches a node v : u <v ≤ successor(u). If

**There exists a v: u <v < successor(u)**

then loopiness exists, and reset successor(u):=v

Takes $O(N^2)$ steps. But loopiness is a rare event.

No protocol for recovery exists from a split ring.

# New peers joining (3)

- A new peer affects *O(log N)* other finger entries in the system. So, the number of messages per peer join= *O(log(N)*log(N))*

- Similar set of operations for dealing with peers leaving

# Bidirectional Chord

Each node u has fingers to

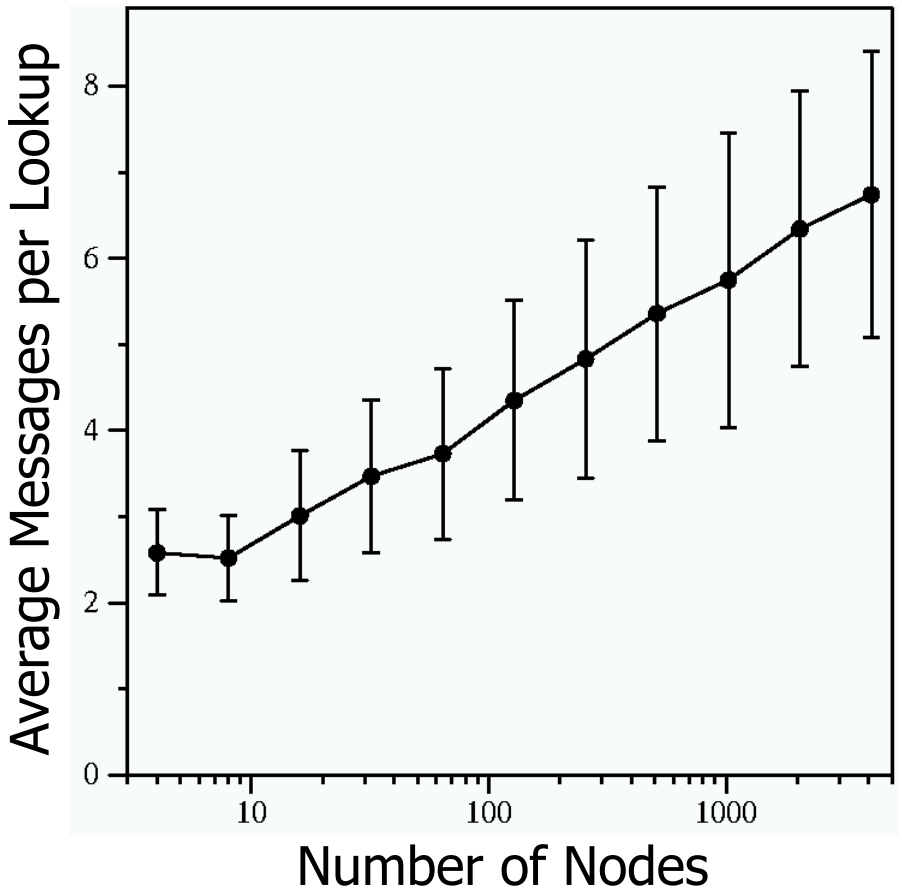
u+1, u+2, u+4, u+8 …  as well as

u-1, u-2, u-4, u-8 …

How does it help?

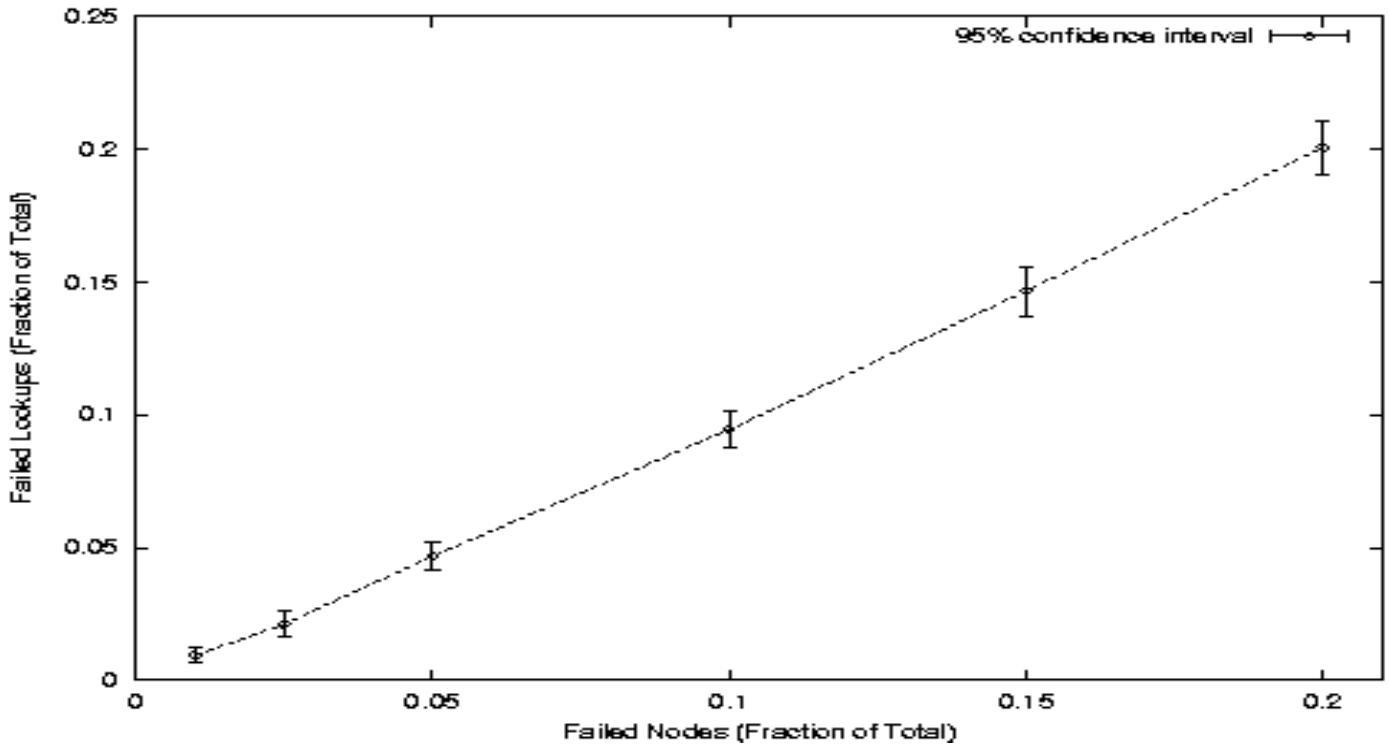
# Cost of lookup

- Cost is O(Log N) as predicted by theory
- constant is 1/2



# Robustness

- Simulation results: static scenario
- Failed lookup means original node with key failed (no replica of keys)




- Result implies good balance of keys among nodes!

# Strengths

- Consistent hashing guarantees balance
- Proven performance in many different aspects
  - "with high probability" proofs
- Good tolerance to random node failures

# Weakness

- Network proximity not addressed
- Protocol security
  - Malicious data insertion
  - Malicious Chord table information
- Keyword search