CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

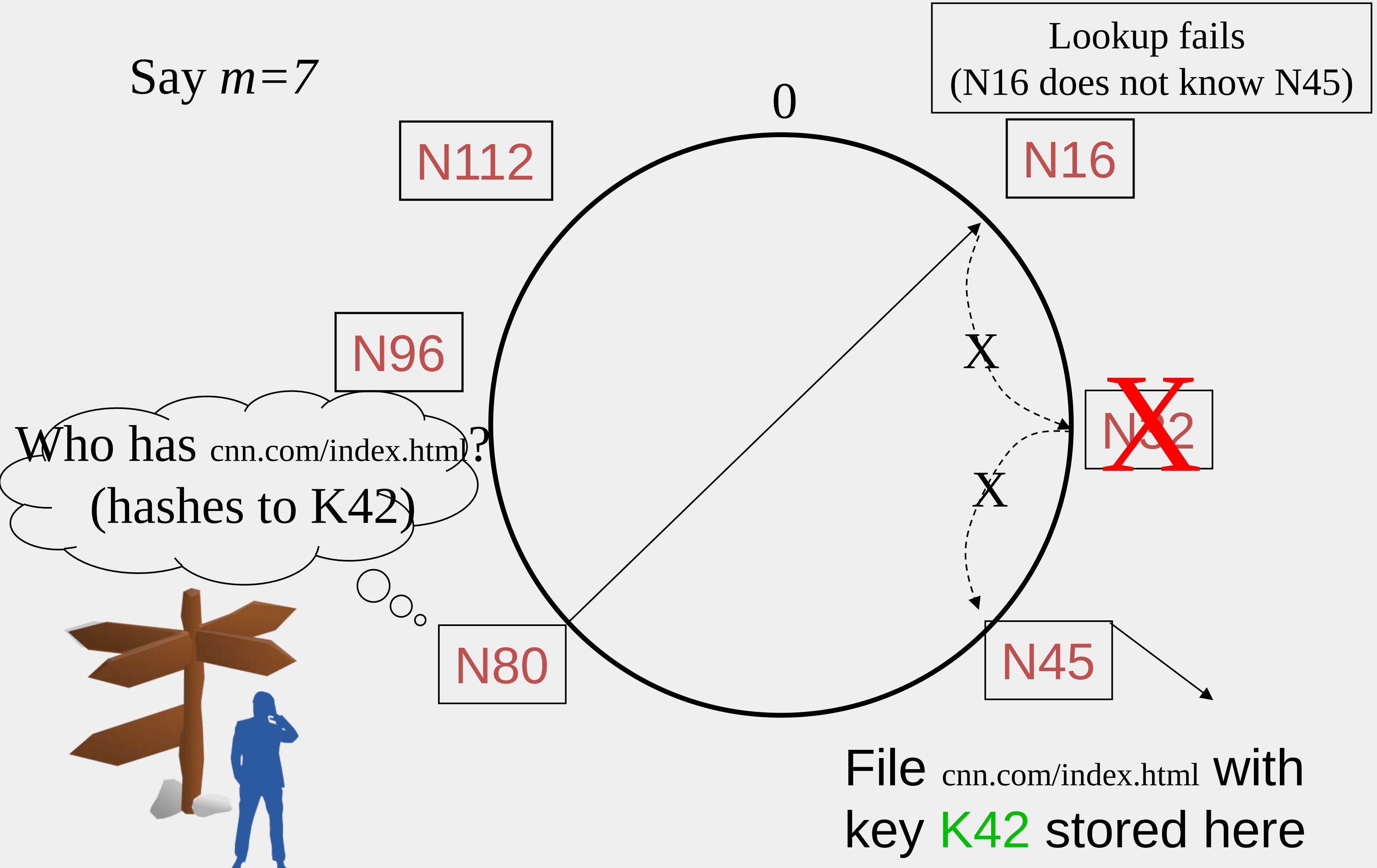
P2P SYSTEMS

Lecture F

FAILURES IN CHORD

Search under peer failures

Say m=7

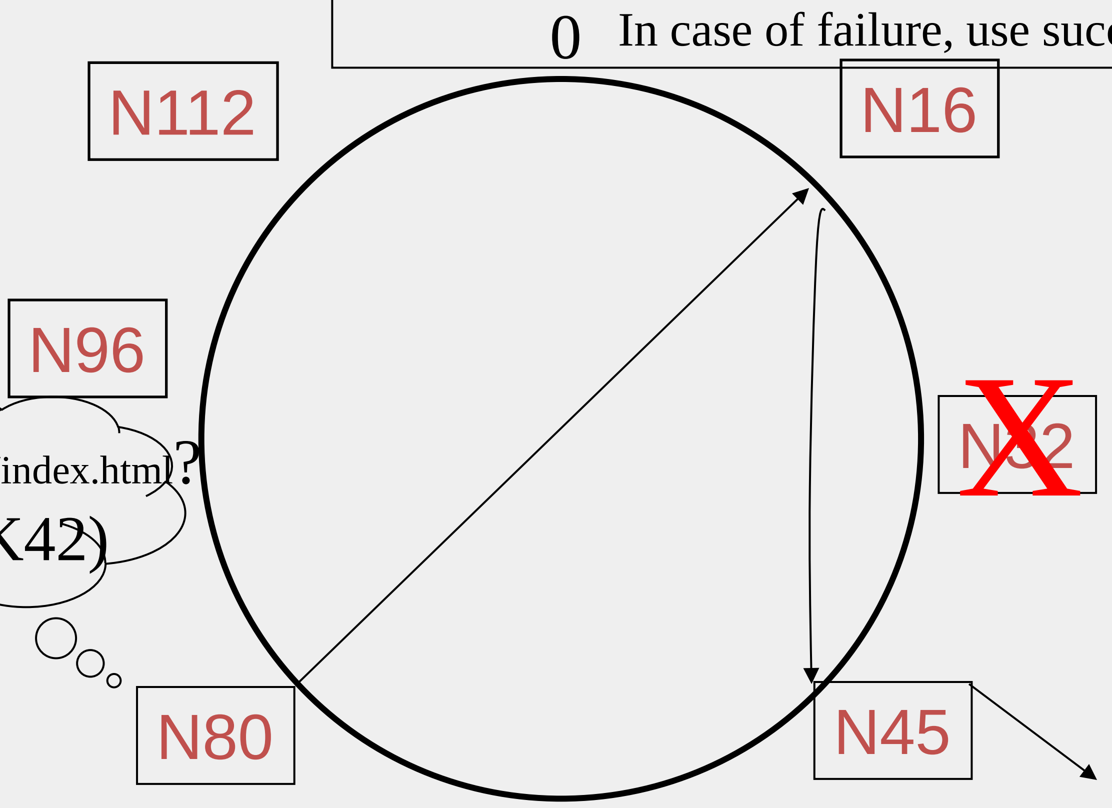


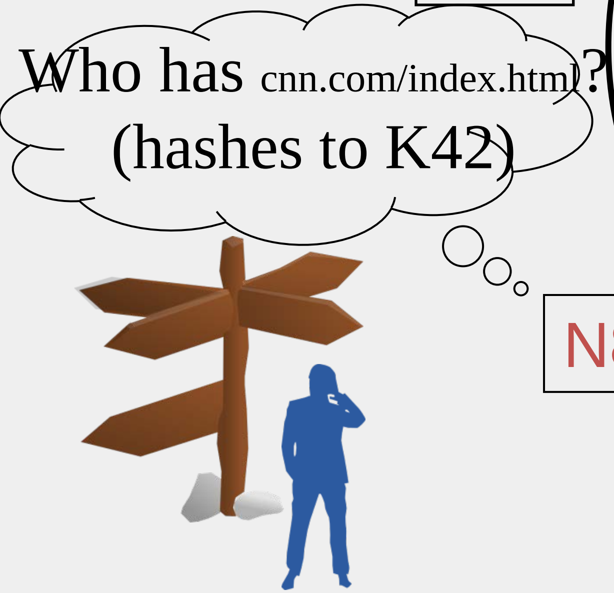
Search under peer failures

Say $m=7$

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

In case of failure, use successor entries





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

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)=

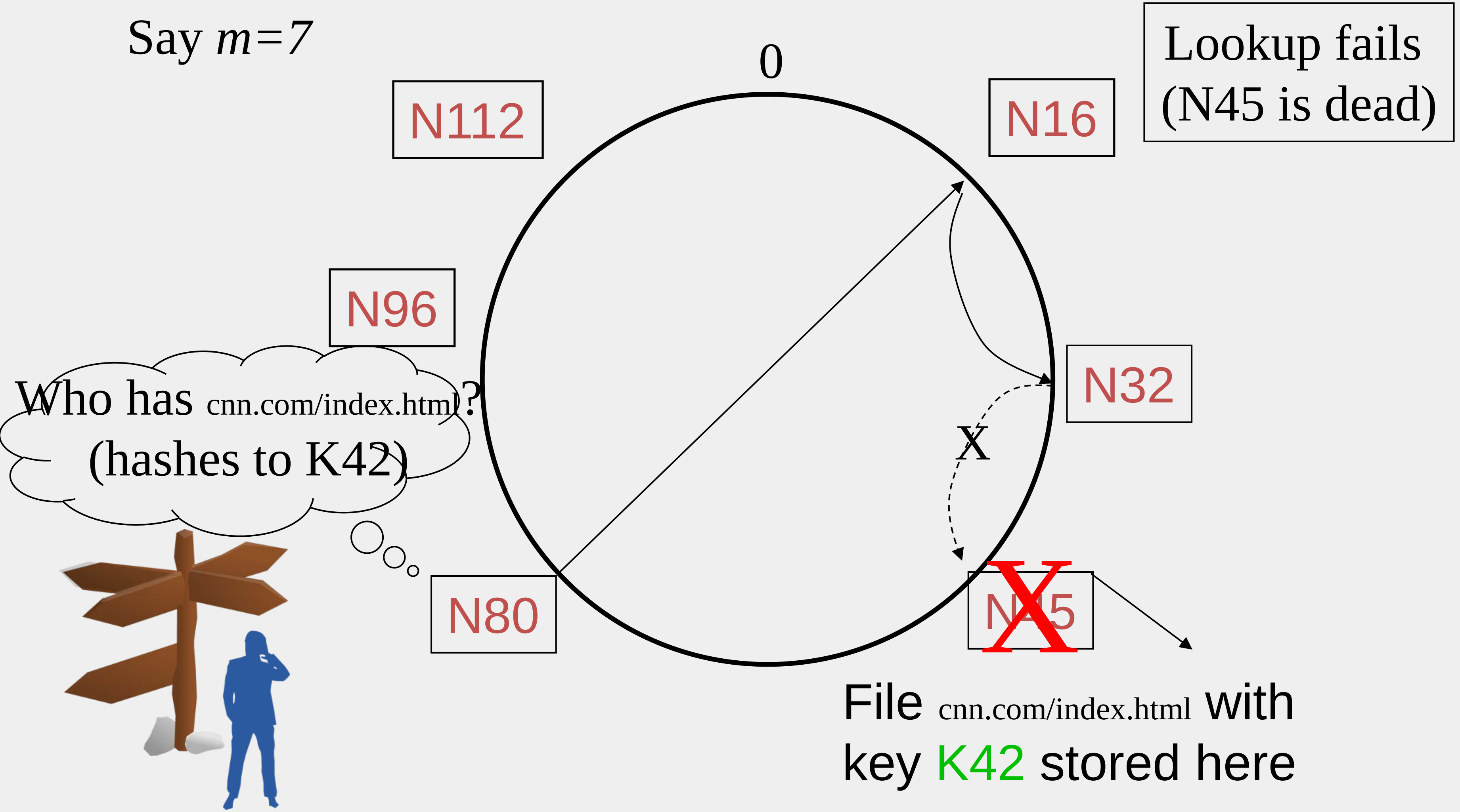
$$1-\left(\frac{1}{2}\right)^{2\log N} = 1-\frac{1}{N^2}$$

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

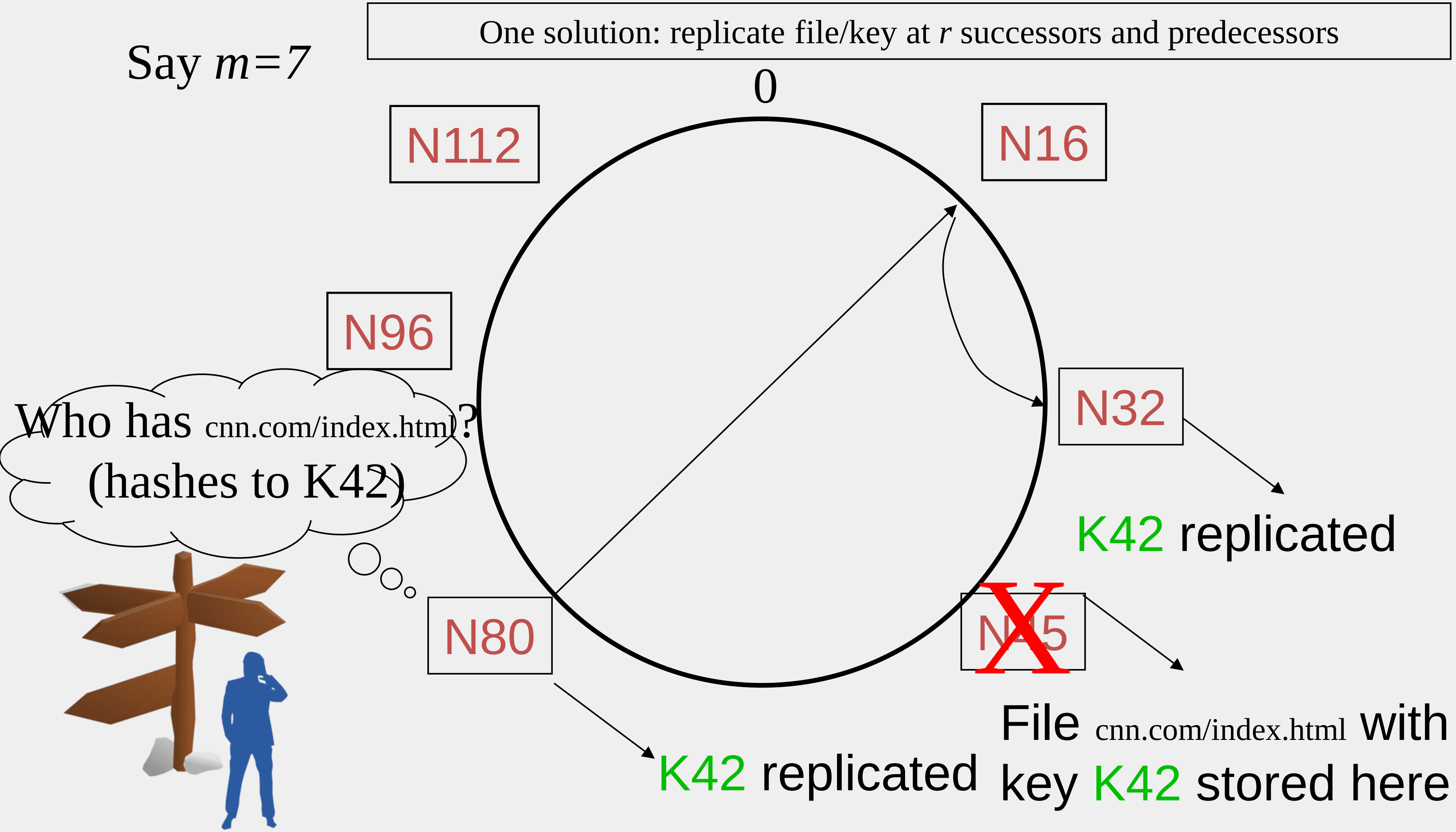
$$\left(1-\frac{1}{N^2}\right)^{N/2} = e^{-\frac{1}{2N}} \approx 1$$

Search under peer failures (2)

Say m=7



Search under peer failures (2)



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:

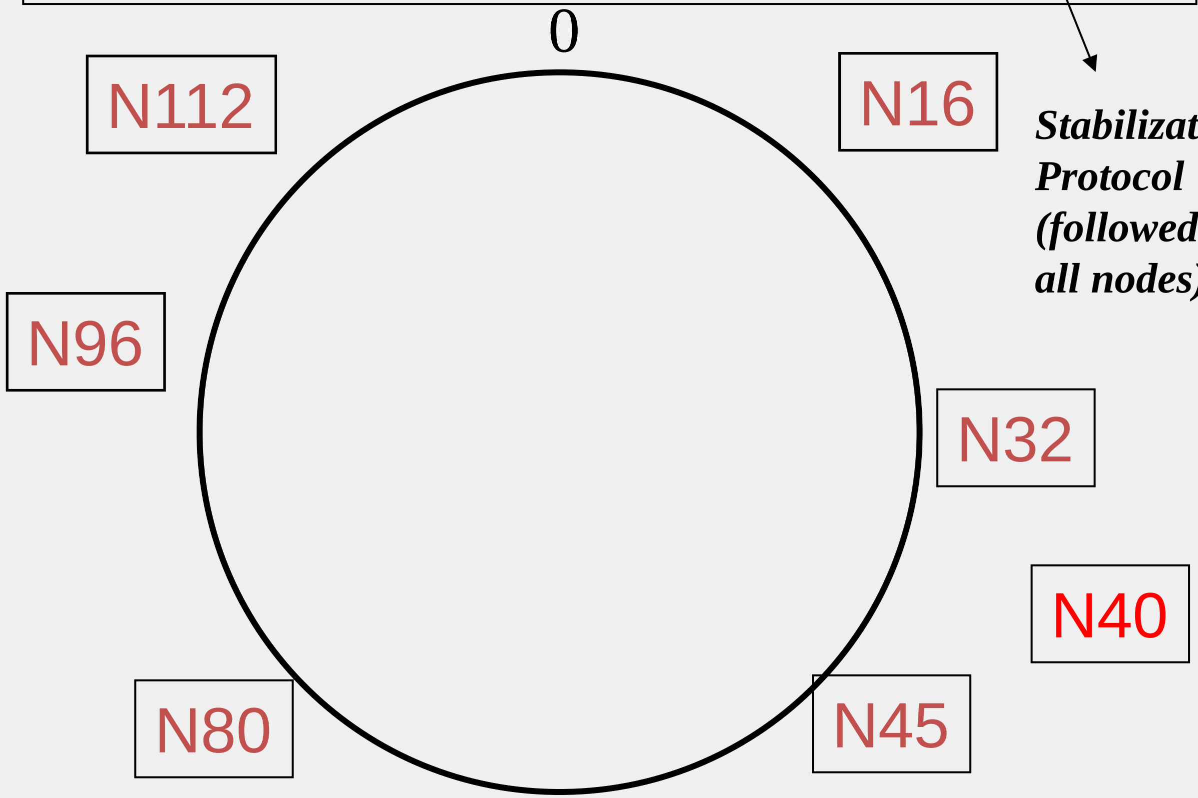
→ update *successor*s and *finger*s, and copy keys

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

Stabilization Protocol (followed by all nodes)

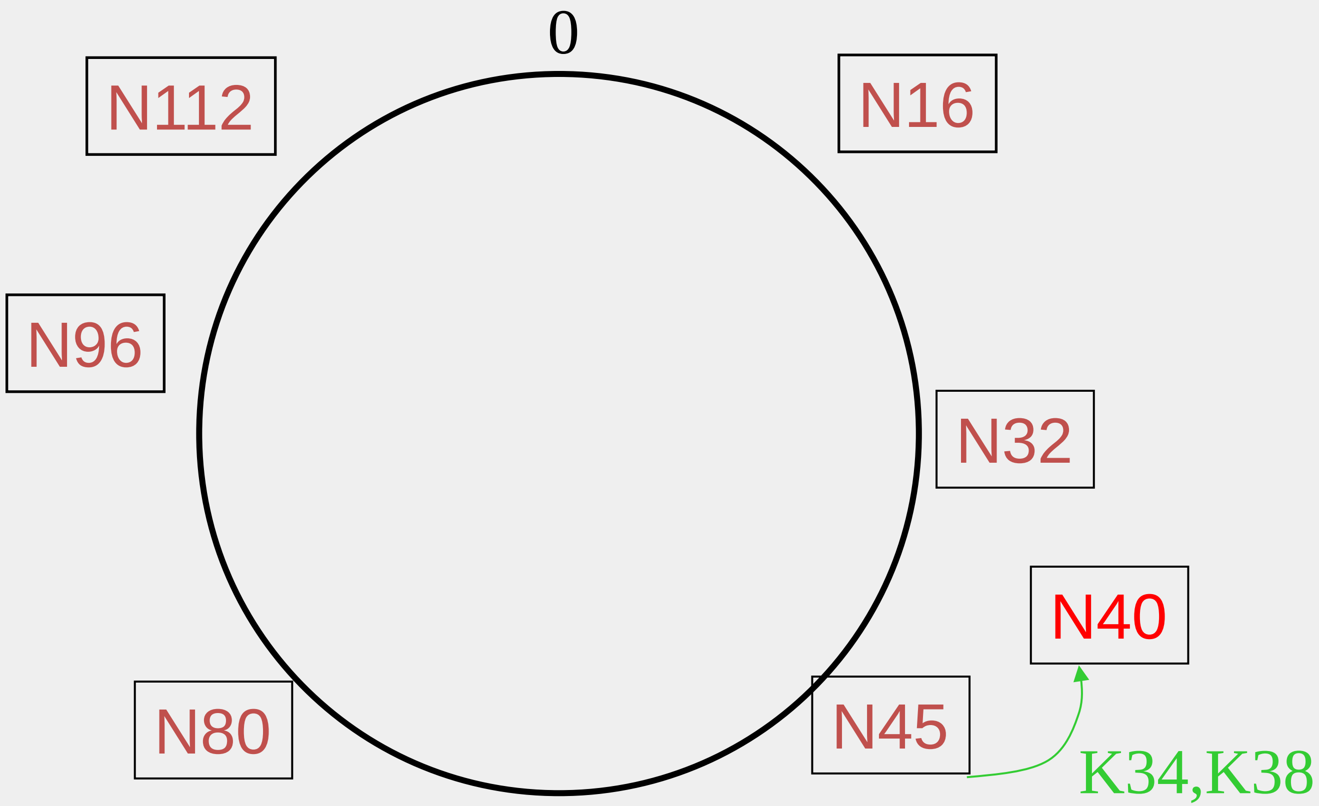
Say m=*7*



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* (we' ll see these later in the course!)

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]

Churn

- When nodes are constantly joining, leaving, failing
  - Significant effect to consider: traces from the Overnet system show *hourly* peer turnover rates (***churn***) could be *25–100%* of total number of nodes in system
  - Leads to excessive (unnecessary) key copying (remember that keys are replicated)
  - Stabilization algorithm may need to consume more bandwidth to keep up
  - Main issue is that files are replicated, while it might be sufficient to replicate only meta information about files
  - Alternatives
    - Introduce a level of indirection (any p2p system)
    - Replicate metadata more, e.g., Kelips (later in this lecture series)

Virtual Nodes

- Hash can get non-uniform ➔ Bad load balancing
  - Treat each node as multiple virtual nodes behaving independently
  - Each joins the system
  - Reduces variance of load imbalance

Wrap-up Notes

- Virtual Ring and Consistent Hashing used in Cassandra, Riak, Voldemort, DynamoDB, and other key-value stores

- Current status of Chord project:
  - File systems (CFS, Ivy) built on top of Chord
  - DNS lookup service built on top of Chord
  - Internet Indirection Infrastructure (I3) project at UC Berkeley
  - Spawned research on many interesting issues about p2p systems

http://www.pdos.lcs.mit.edu/chord/