Distributed Systems

CS425/ECE428

April 21 2021

Instructor: Radhika Mittal

Acknowledgements for some of the materials: Indy Gupta

Our agenda for the next 3-4 classes

- Brief overview of key-value stores

- Distributed Hash Tables
  - Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.

- Key-value stores in the cloud
  - How to run large-scale distributed computations over key-value stores?
    - Map-Reduce Programming Abstraction
  - How to design a large-scale distributed key-value store?
    - Case-study: Facebook's Cassandra

Quick Recap

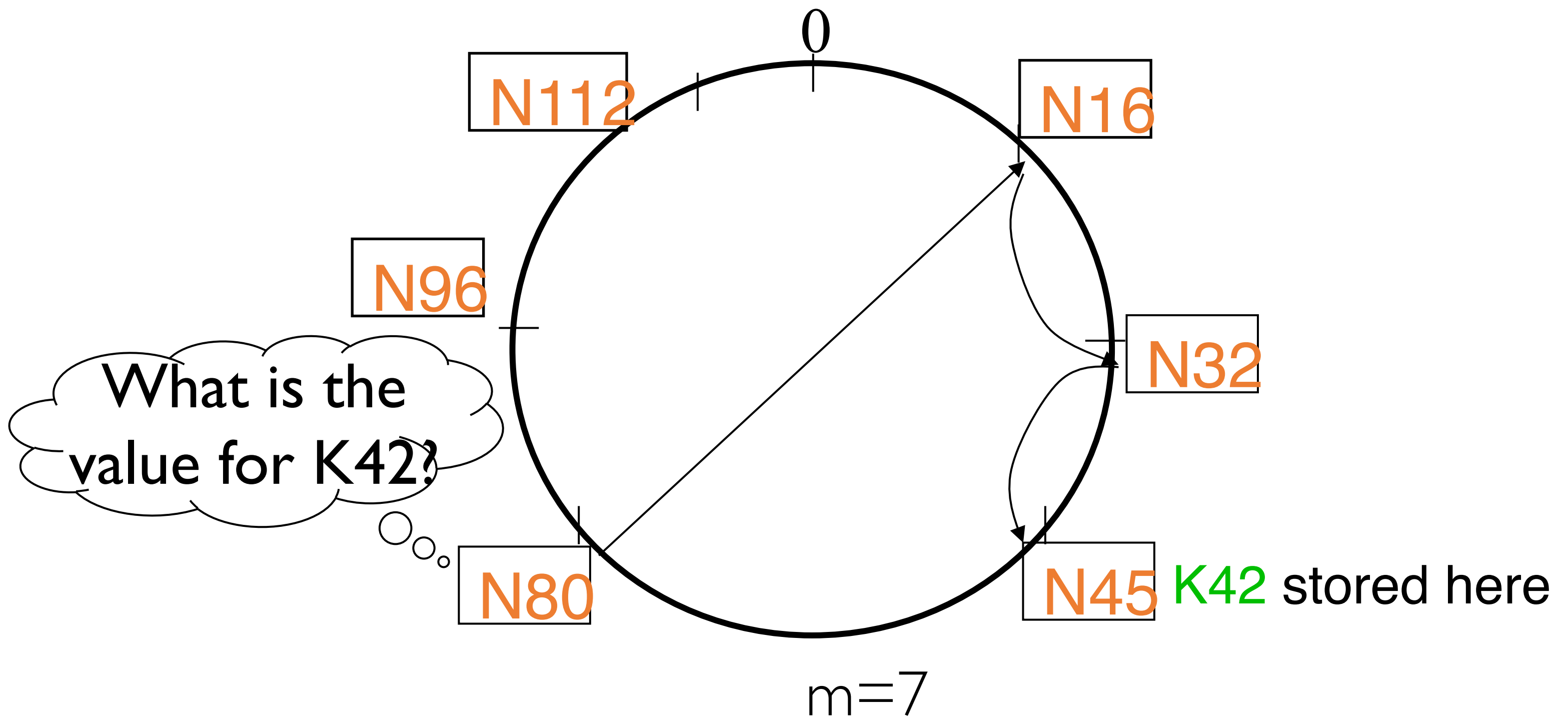
- Distributed Hash Tables
    - Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.
    - Other required properties: load balancing, fault tolerance.

- Case-study: Chord

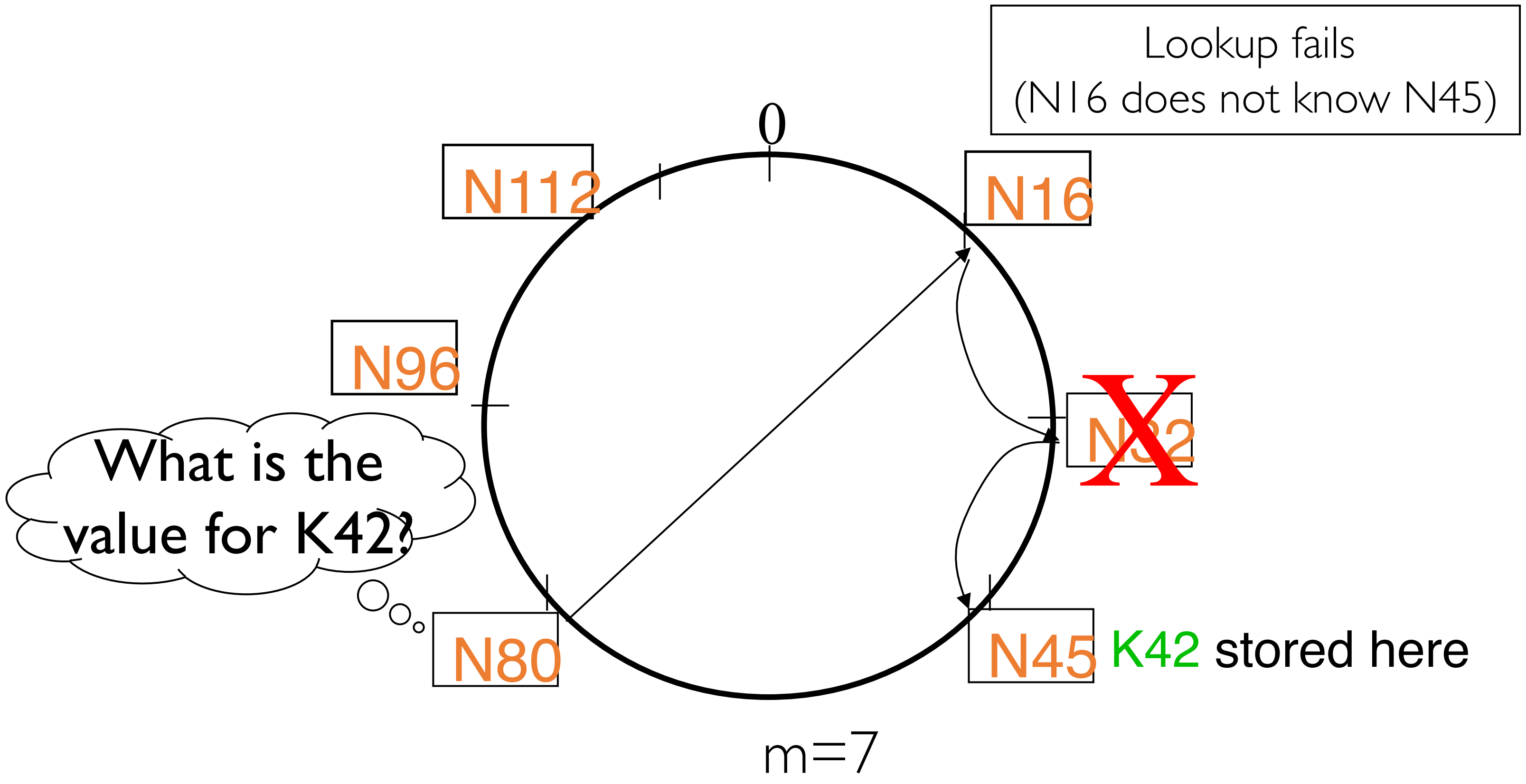
Quick Recap: Chord

- Uses **consistent hashing** to map nodes on a ring with m-bits identifiers.
- Uses consistent hashing to map a key to a node.
  - stored at **successor(key)**
- Each node maintains a **finger table** with m fingers.
  - With high probability, results in O(logN) hops for a look-up.
  - O(log(N)) hops true only if finger and successor entries correct.
    - What happens when nodes fails or when new nodes join in?
      - Our focus today.

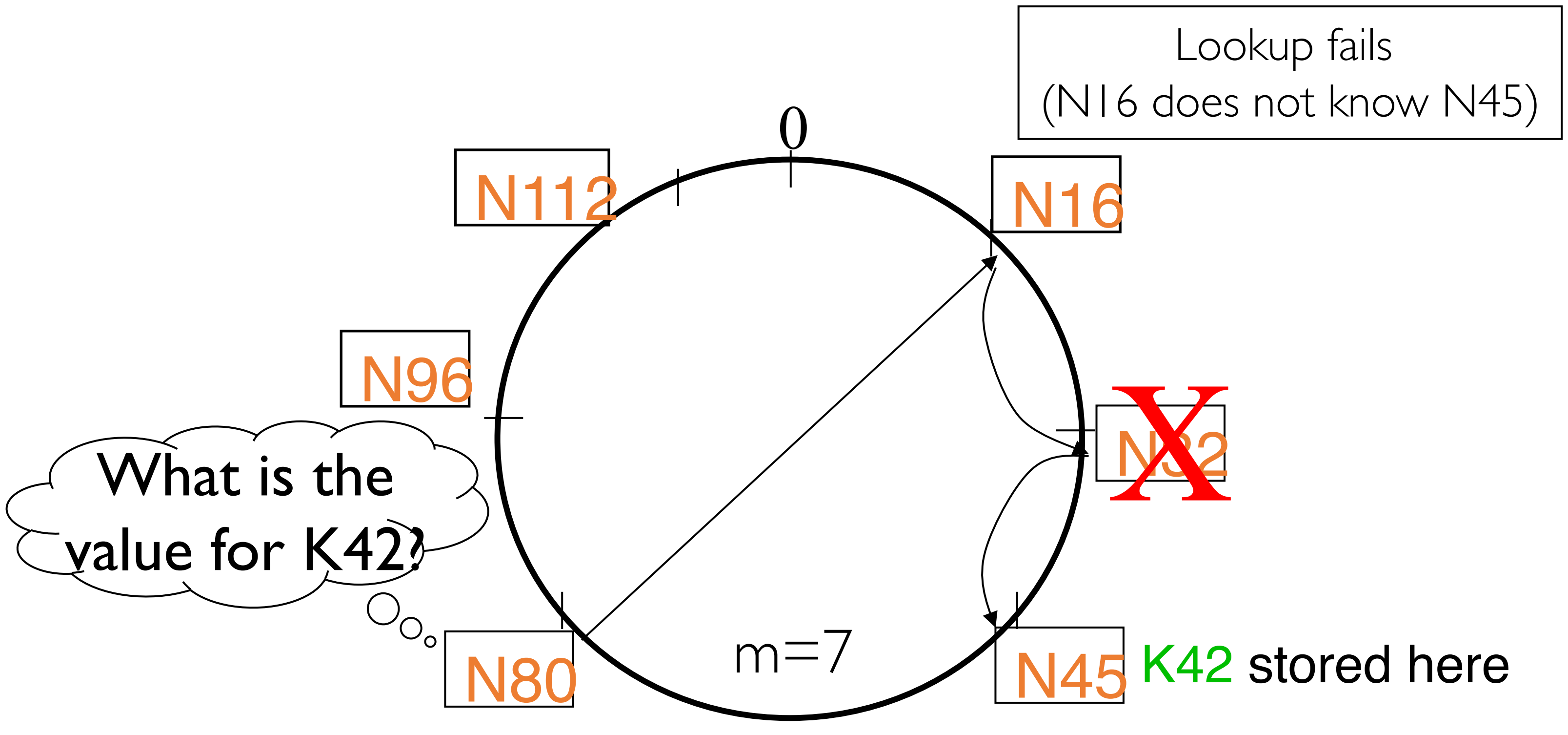
Search for key k at node n



If a node fails



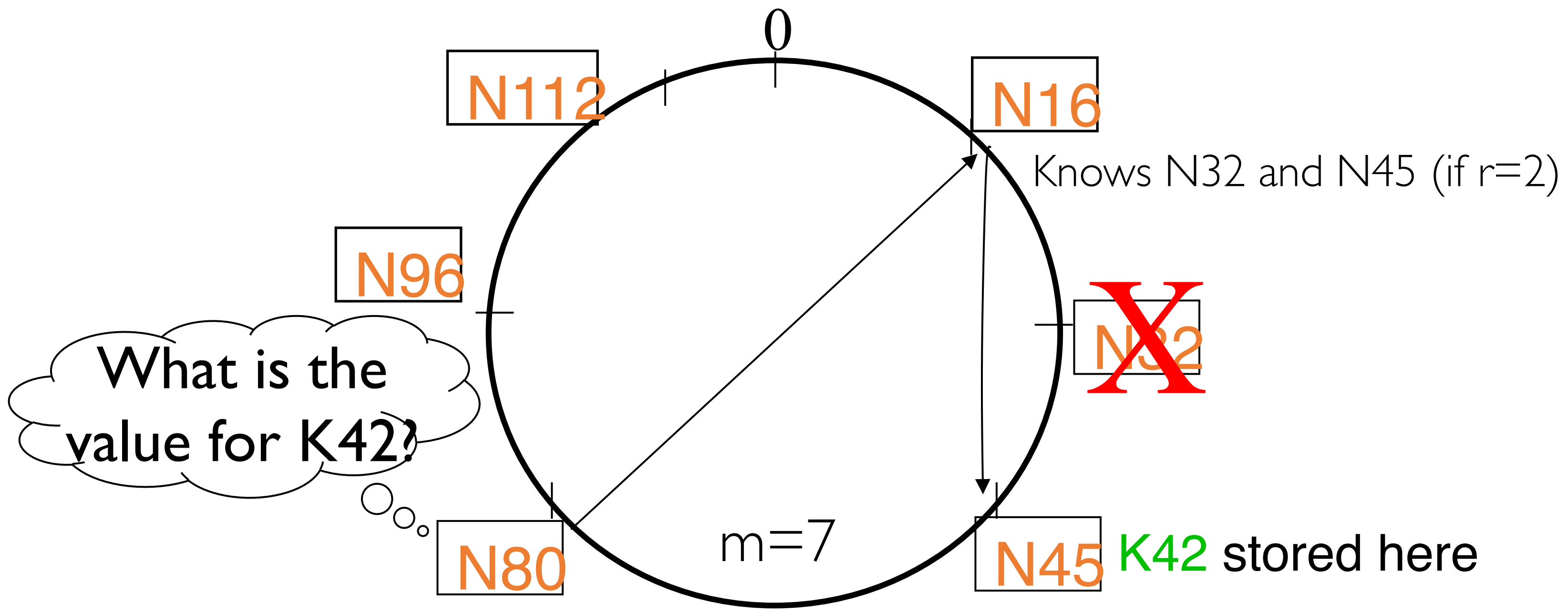
If a node fails



How do we handle this?

If a node fails

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



Search under node failures

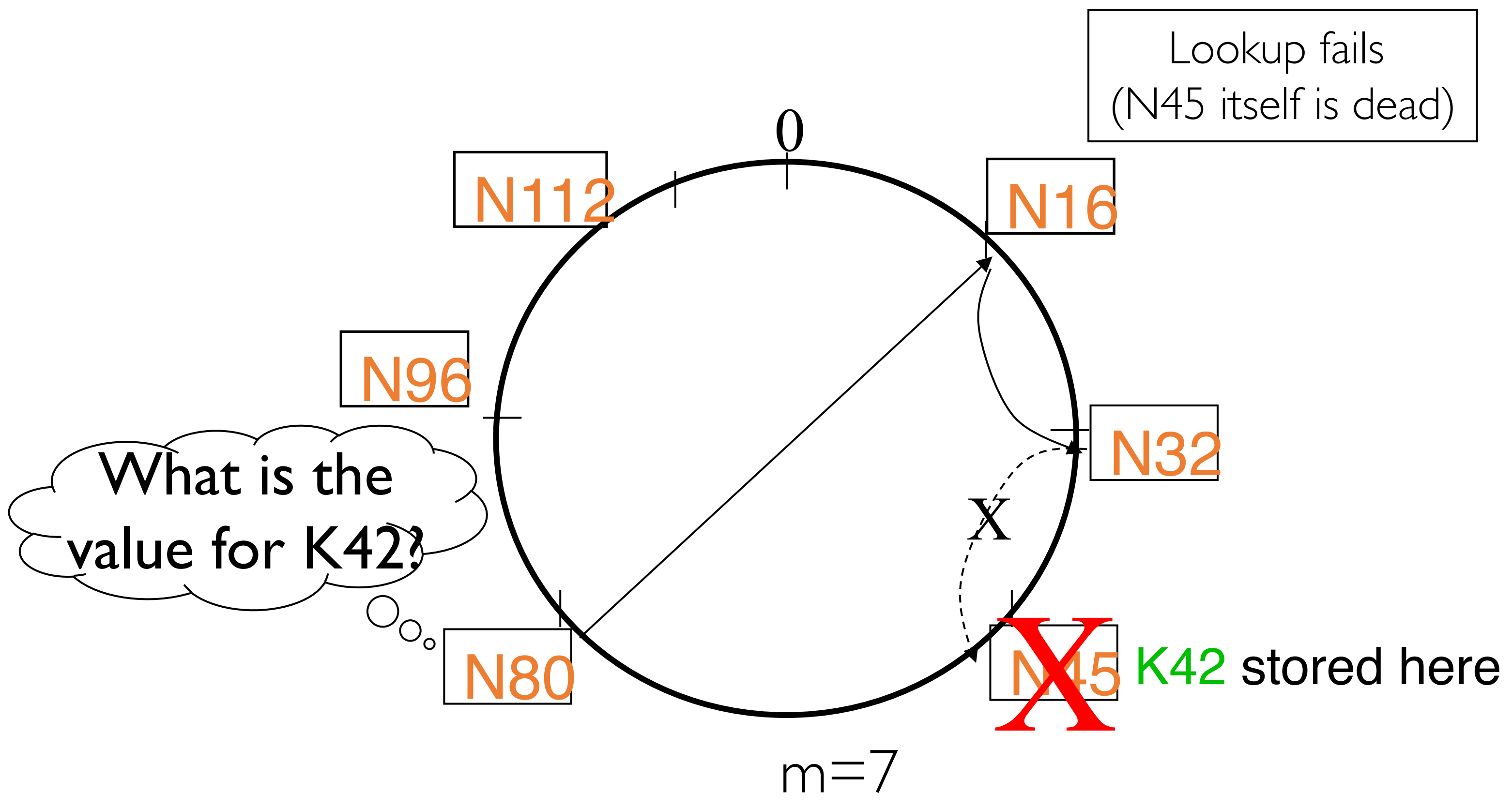
- If every node fails with probability 0.5, choosing r=2log(N) suffices to maintain lookup correctness (i.e. keep the ring connected) with high probability.
  - Intuition:
    - Pr(at given node, at least one successor alive)=

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

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

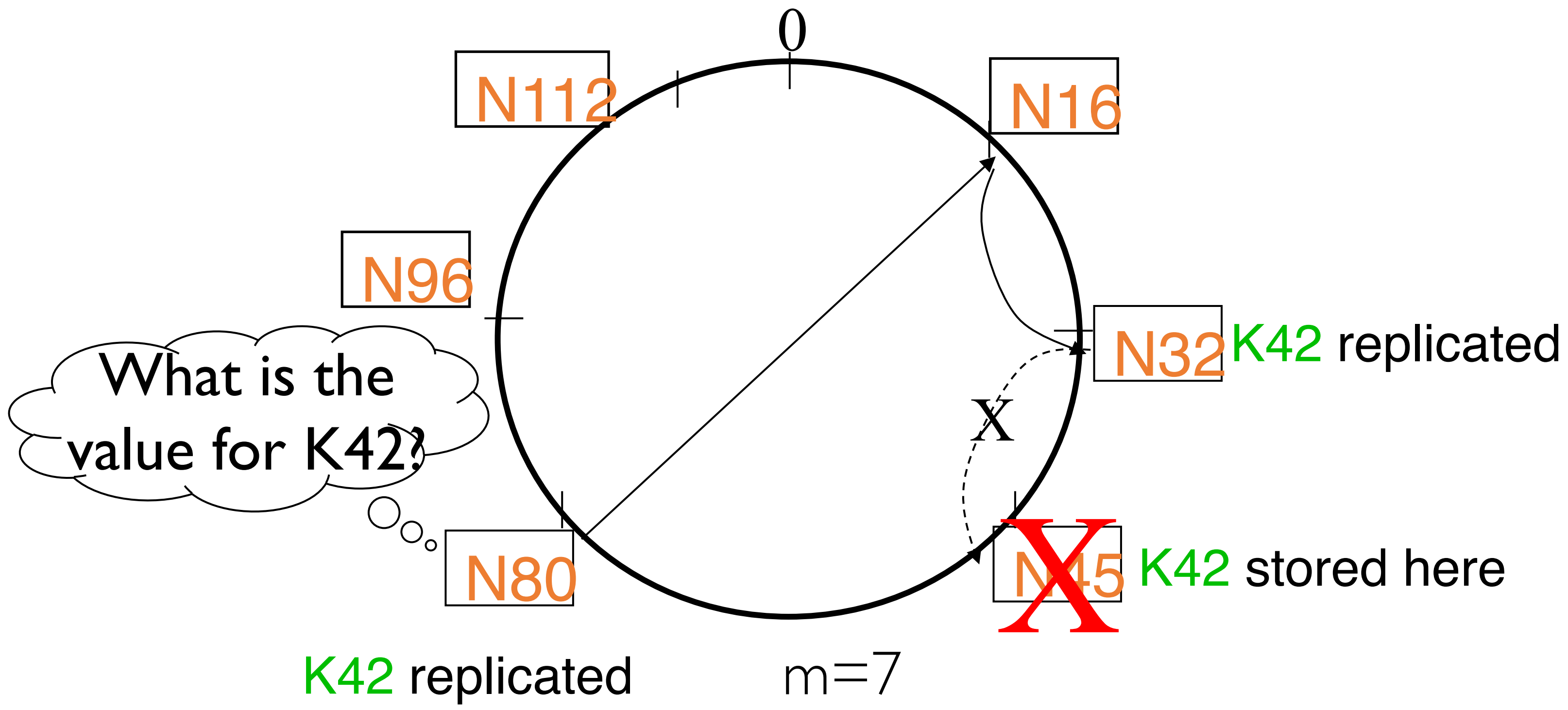
$$(1-\frac{1}{N^2})^{N/2}=e^{-\frac{1}{2N}}\approx 1$$

If a node fails



If a node fails

One solution: replicate key-value at r successors and predecessors



Need to deal with dynamic changes

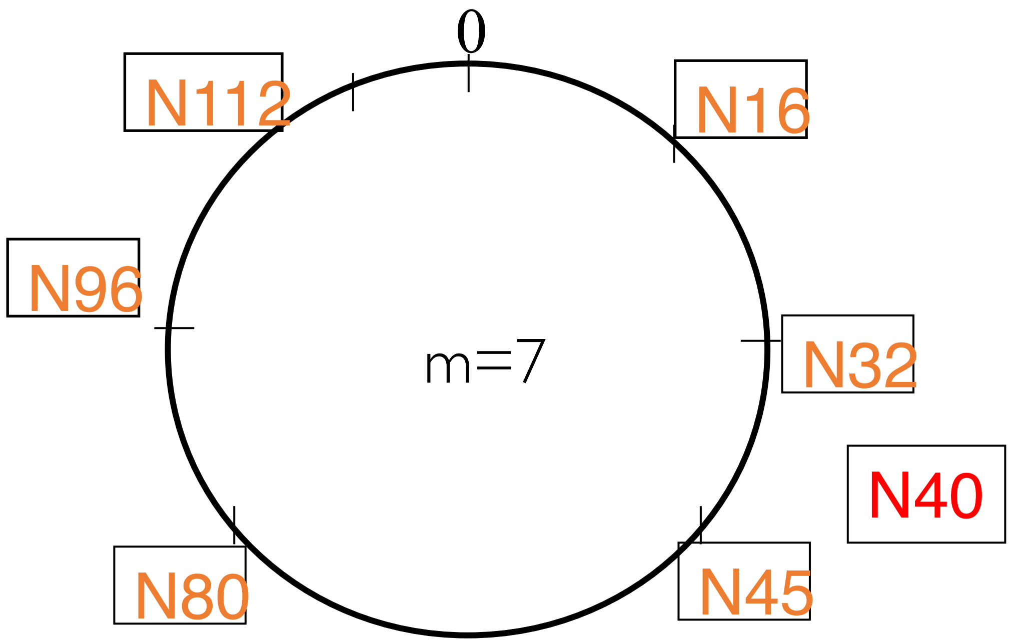
- ✓ Nodes fail
- New nodes join
- Nodes leave

So, all the time, need to:

→ Need to update successors and fingers, and copy keys

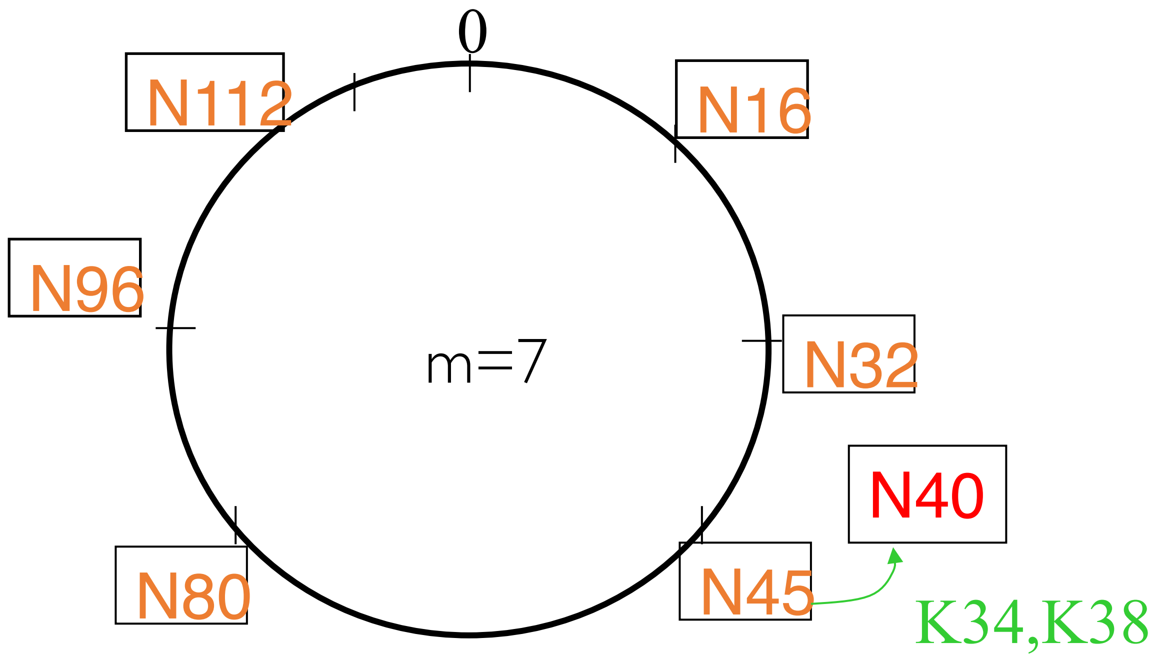
New node joins

New node contacts an existing Chord node (introducer).
Introducer directs N40 to N45 (and N32).
N32 updates its ring successor to N40.
N40 initializes its ring successor to N45, and initializes its finger table.
Other nodes also update their finger table.



New node joins

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



New node joins

- A new peer affects O(log(N)) other finger entries in the system, on average.

- Number of messages per node join (to initialize the new node's finger table) = O(log(N)*log(N))

- Proof in Chord's extended TechReport.

Concurrent Joins

- Aggressively maintaining and updating finger tables each time a node join can be difficult under high *churn*.
    - E.g. when new nodes are concurrently added.
- Correctness of lookup does not require all nodes to have fully "correct" finger table entries.
- Need two invariants:
    - Each node, n, correctly maintains its ring successor (*next(n)*)
        - First entry in the finger table.
    - The node, successor(k), is responsible for key k.

Stabilization Protocol

- When a node n joins (via an introducer)
    - initialize next(n), i.e. the ring successor
    - notify next(n).

- When node n gets notified by node n’:
    - // update prev(n), i.e. the ring predecessor of n
    - if (prev(n) == nil or n’ is in (prev(n), n))
        - prev(n) = n’.

Stabilization Protocol (contd)

- Each node n will periodically run stabilization:
    - x = prev(next(n))
    - if x in (n, next(n)), then next(n) = x.
    - notify next(n).

- Each node n periodically updates a random finger entry.
    - Pick a random i in [0, m-1]
    - Lookup successor($n + 2^i$)

New node joins

New node contacts an existing Chord node (introducer).

Introducer informs N40 of N45.

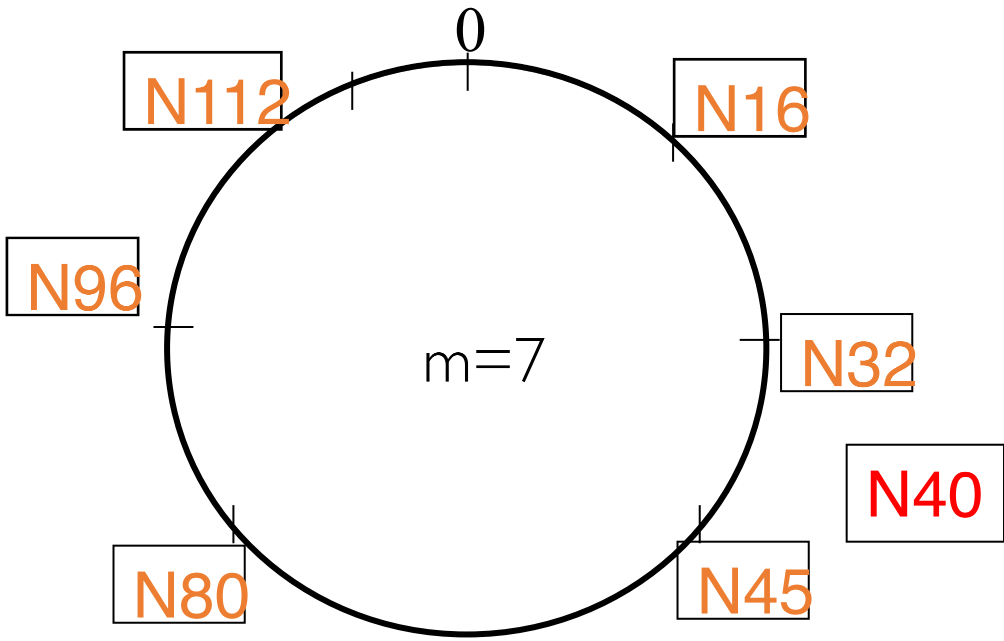
N40 initializes its ring successor to N45.

N40 notifies N45, and N45 initializes its ring predecessor to N40.

N32 realizes its new successor is N40 when it runs stabilization.

N32 notifies N40, and N40 initializes its ring predecessor to N32.

Periodically and eventually, each node update their finger table entries.



Stabilization Protocol (contd)

- Failures can be handled in a similar way.
  - *Also need failure detectors (you've seen them!)*
  - Maintain knowledge of **r** ring successors.
  - Update ring successor when it fails, and notify.

Stabilization Protocol (contd)

- Look-ups may fail while the Chord system is getting stabilized.
  - Such failures are transient.
    - Eventually ring successors and finger-table entries will get updated.
    - Application can then try again after a timeout.
  - Such failures are also unlikely in practice
    - Multiple key-value replicas and ring successors.

Chord Summary

- Consistent hashing for load balancing.
- O(logn) lookups via correct finger tables.
- *Correctness* of lookups requires correctly maintaining ring successors.
- As nodes join and leave a Chord network, runs a stabilization protocol to periodically update ring successors and finger table entries.
- Fault tolerance: Maintain **r** ring successors and **r** key replicas.

Our agenda for the next 3-4 classes

- Brief overview of key-value stores

- Distributed Hash Tables
    - Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.

- Key-value stores in the cloud
    - How to run large-scale distributed computations over key-value stores?
        - Map-Reduce Programming Abstraction
    - How to design a large-scale distributed key-value store?
        - Case-study: Facebook's Cassandra

Cloud Computing

Many Cloud Providers

- AWS: Amazon Web Services
  - EC2: Elastic Compute Cloud
  - S3: Simple Storage Service
- Microsoft Azure
- Google Cloud/Compute Engine/AppEngine
- Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera
- And many many more!

What is a cloud?

- Cloud = Lots of storage + compute cycles nearby

- Cloud services provide:
  - managed *clusters* for distributed computing.
  - managed *distributed datastores*.

What is a cloud?

- A single cloud-site (aka "Datacenter") consists of
  - Compute nodes (grouped into racks) (2)
  - Switches, connecting the racks in a hierarchical network topology.
  - Storage (backend) nodes connected to the network (3)
  - Front-end for submitting jobs and receiving client requests (1)
  - (1-3: Often called "three-tier architecture")

- A geographically distributed cloud consists of
  - Multiple such sites
  - Each site perhaps with a different structure and services

Features of cloud

I. Massive scale.
   - Tens of thousands of servers and cloud tenants, and hundreds of thousands of VMs.

II. On-demand access:
   - Pay-as-you-go, no upfront commitment, access to anyone.

III. Data-intensive nature:
   - What was MBs has now become TBs, PBs and XBs.
     - Daily logs, forensics, Web data, etc.

Must deal with immense complexity!

- Fault-tolerance and failure-handling
- Replication and consensus
- Cluster scheduling

- How would a cloud user deal with such complexity?
  - **Powerful abstractions and frameworks**
  - Provide **easy-to-use** API to users.
  - Deal with the complexity of distributed computing under the hood.

MapReduce
is one such powerful
abstraction.

MapReduce

- To be continued in next class.
  - Overview
  - Examples
  - Execution and scheduling