Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Acknowledgements for some of the materials: Indy Gupta

Logistics

- HW5 released.
  - you should be able to tackle:
    - first two questions right-away.
    - third question after today's class
    - last two questions after next week's classes.
- HW4 due today!

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 schedule jobs in the cloud?
  - How to design a large-scale distributed key-value store?
    - Case-study: Facebook's Cassandra

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

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.

Performing Lookups

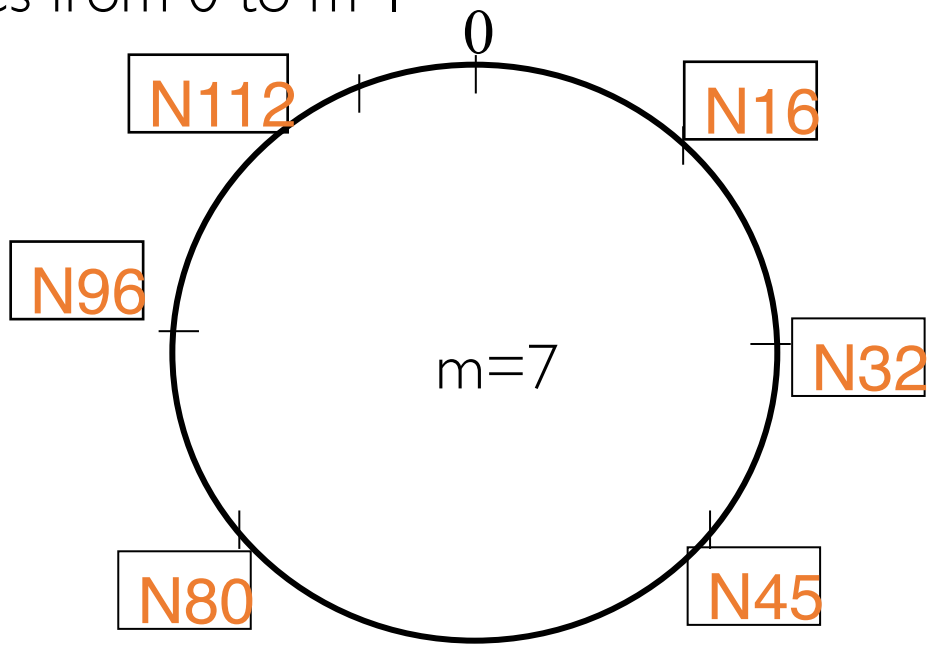
- Chord chooses a sweet middle-ground.
  - Each node is aware of ~**m** other nodes.
  - Maintains a *finger table* with **m** entries.
  - The ith entry of node n's finger table = successor($n + 2^i$)
    - i ranges from 0 to m-1
    - Recall: *successor(x) = first node with id greater than or equal to ($x \bmod 2^m$)*

Finger Tables

Compute the finger table for N80

ith entry of node n's finger table = successor(n + 2^i),

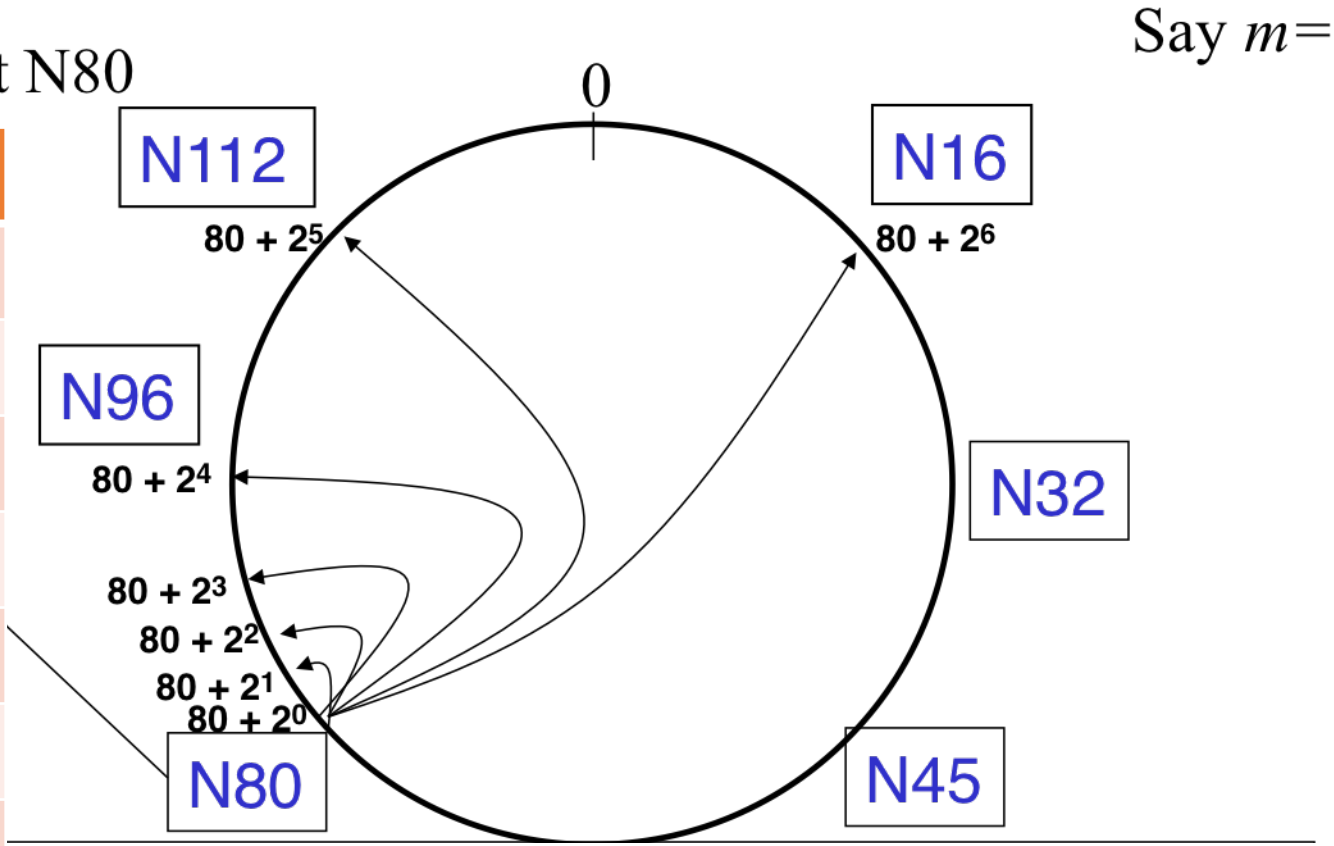
i ranges from 0 to m-1



Finger Tables

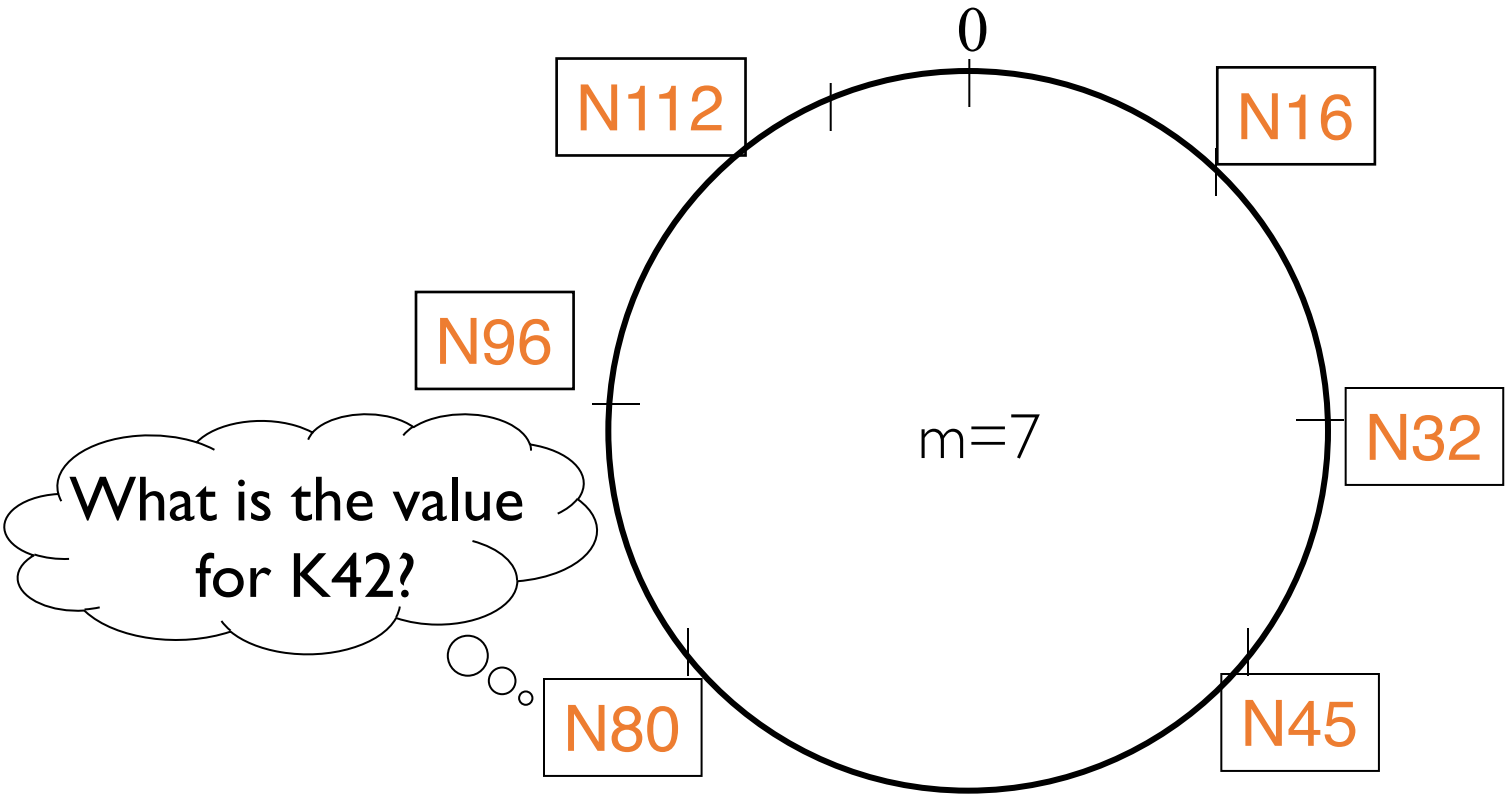
Finger Table at N80

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



Performing Lookups

Suppose N80 receives a request to lookup K42.

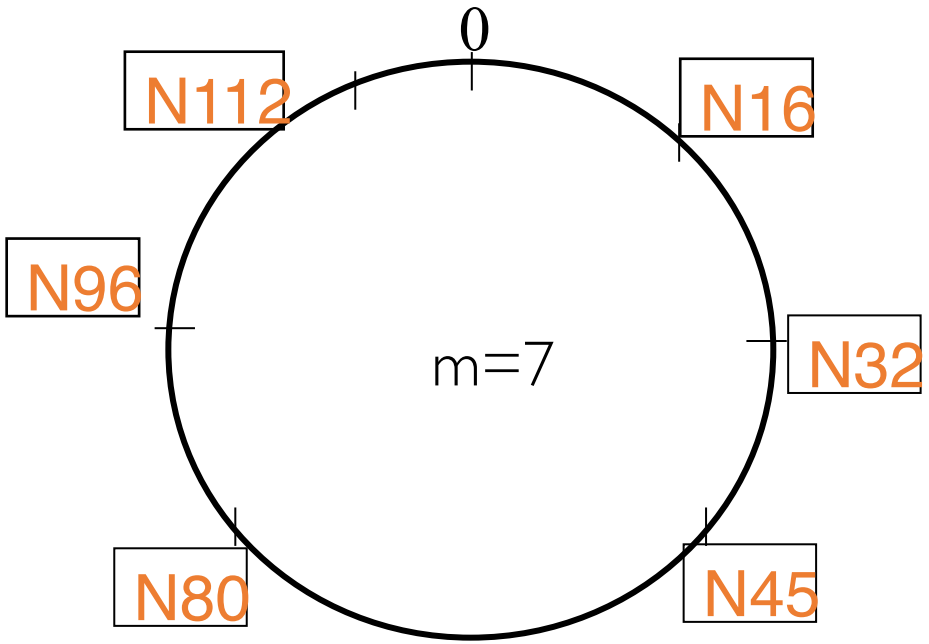


Need to locate successor of K42!

Which nodes is N80 aware of?

N80's finger table

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

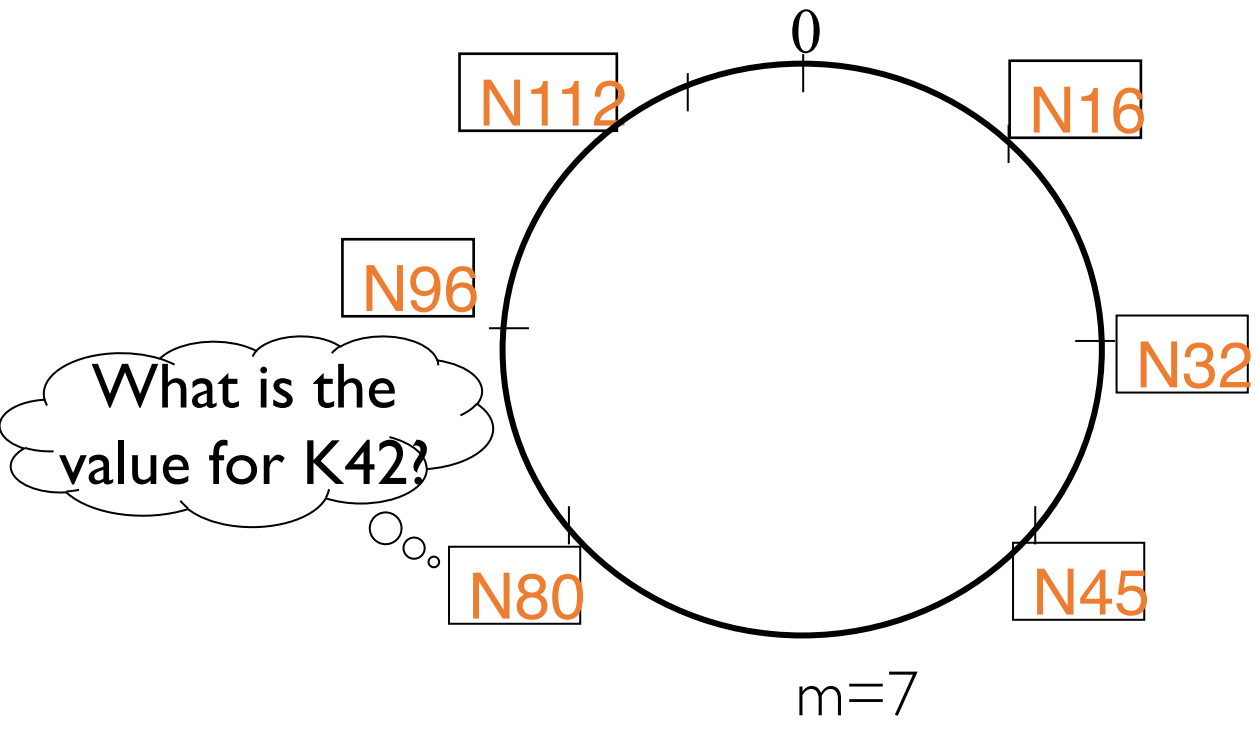


Need to locate successor of K42!

Forward the query to the most promising node you know of.

Search for key k at node n

At node n, if k lies in range (n, next(n)], where next(n) is n's *ring successor* then next(n) = successor(key). Send query to next(n)
Else, send query for k to largest finger entry <= k

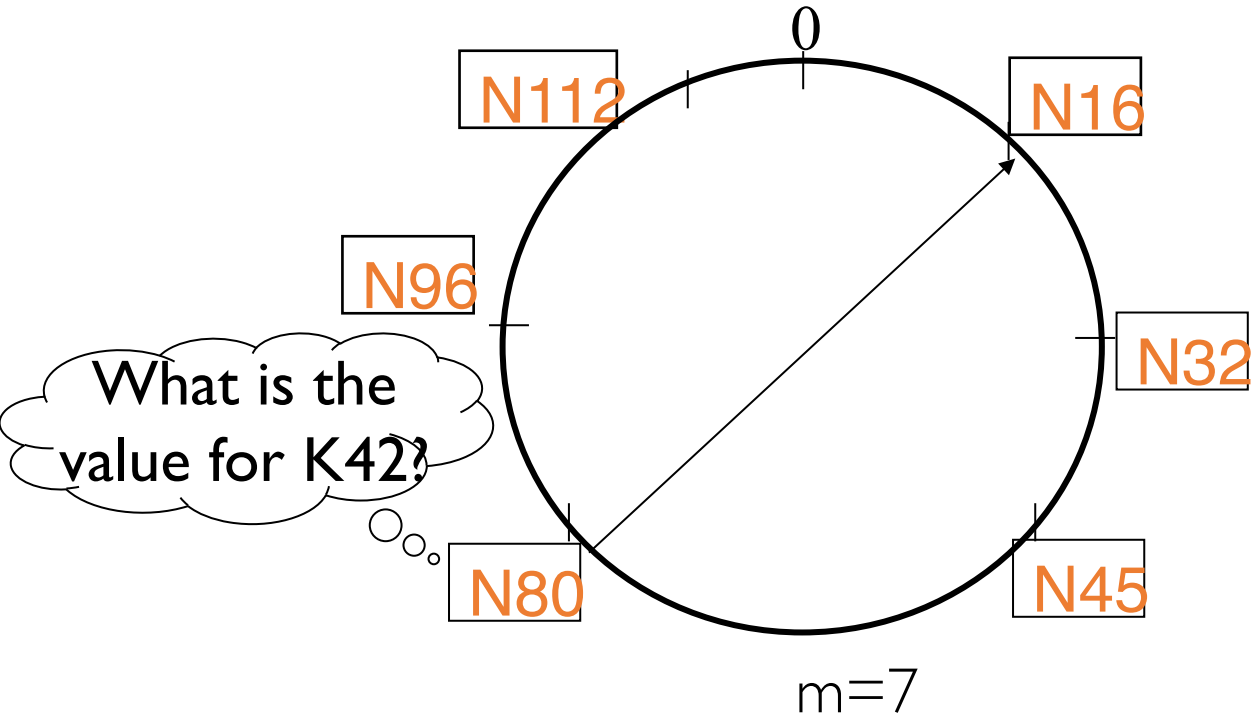


N80's finger table

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

Search for key k at node n

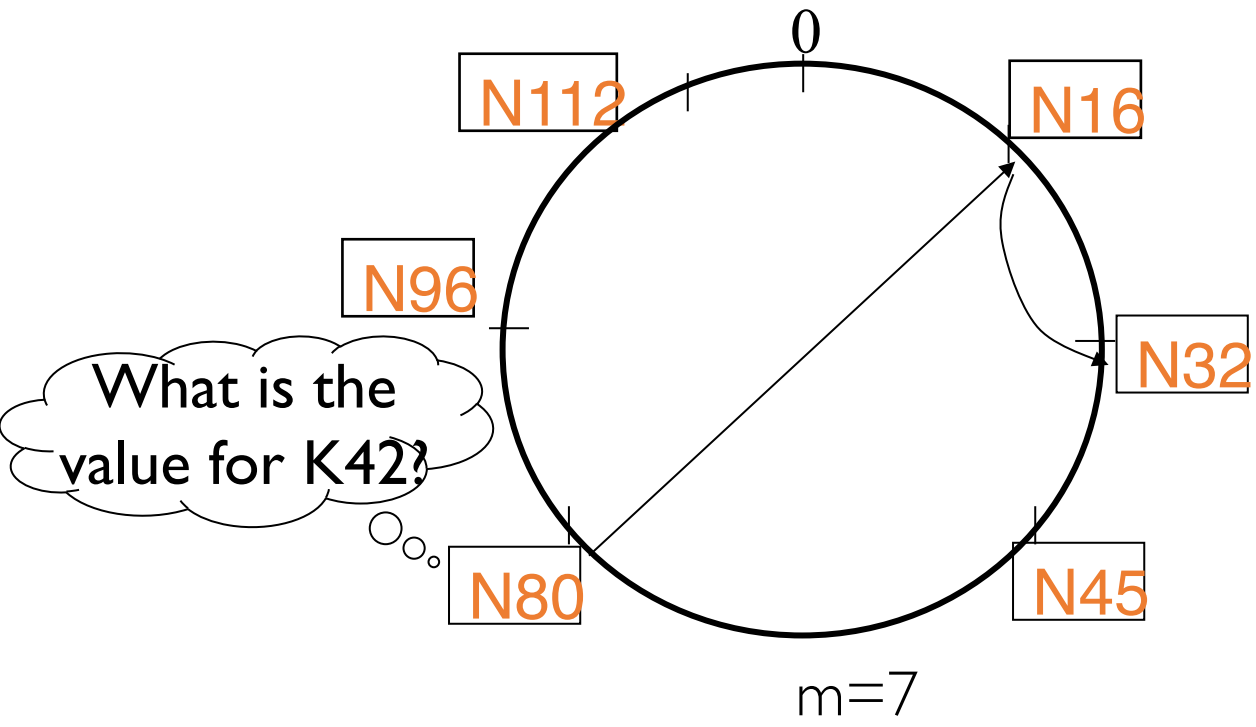
At node n, if k lies in range (n, next(n)], where next(n) is n's *ring successor* then next(n) = successor(key). Send query to next(n)
Else, send query for k to largest finger entry <= k



N16's finger table?

Search for key k at node n

At node n, if k lies in range (n, next(n)], where next(n) is n's *ring successor* then next(n) = successor(key). Send query to next(n)
Else, send query for k to largest finger entry <= k

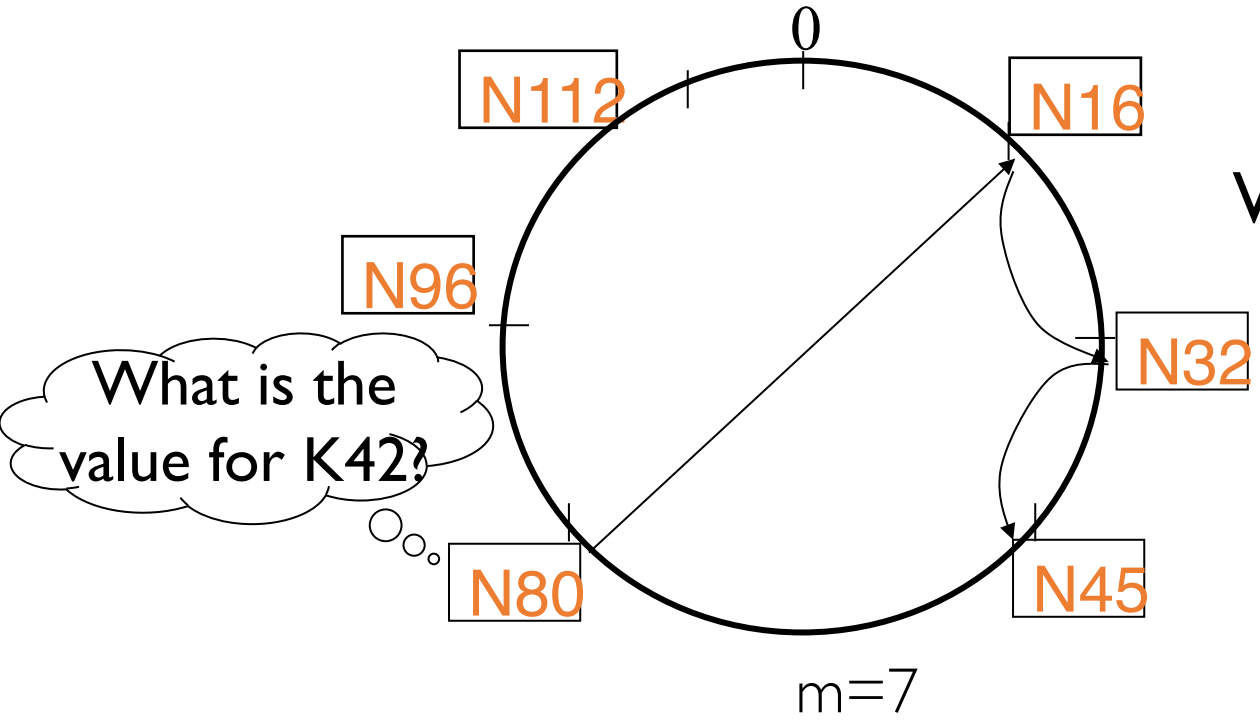


N16's finger table

i	ft[i]
0	32
1	32
2	32
3	32
4	32
5	80
6	80

Search for key k at node n

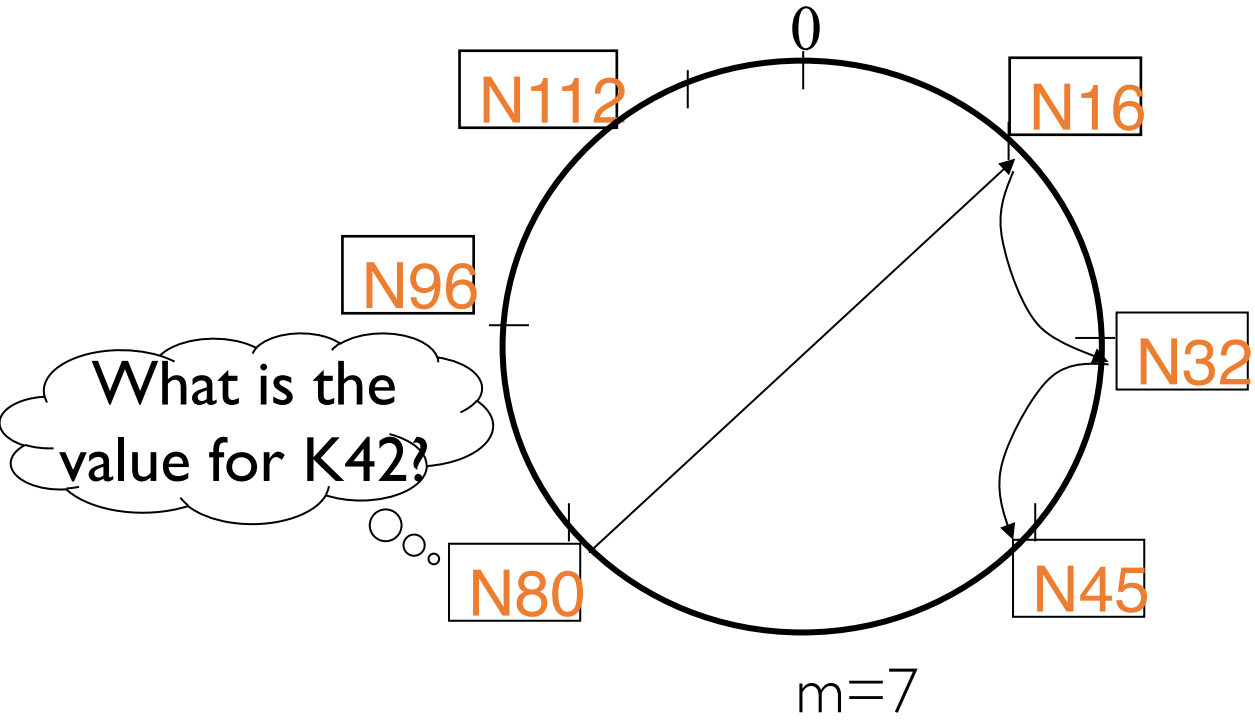
At node n, if k lies in range (n, next(n)], where next(n) is n's *ring successor* then next(n) = successor(key). Send query to next(n)
Else, send query for k to largest finger entry <= k



What will N32 do?

Search for key k at node n

At node n, if k lies in range (n, next(n)], where next(n) is n's *ring successor* then next(n) = successor(key). Send query to next(n)
Else, send query for k to largest finger entry <= k



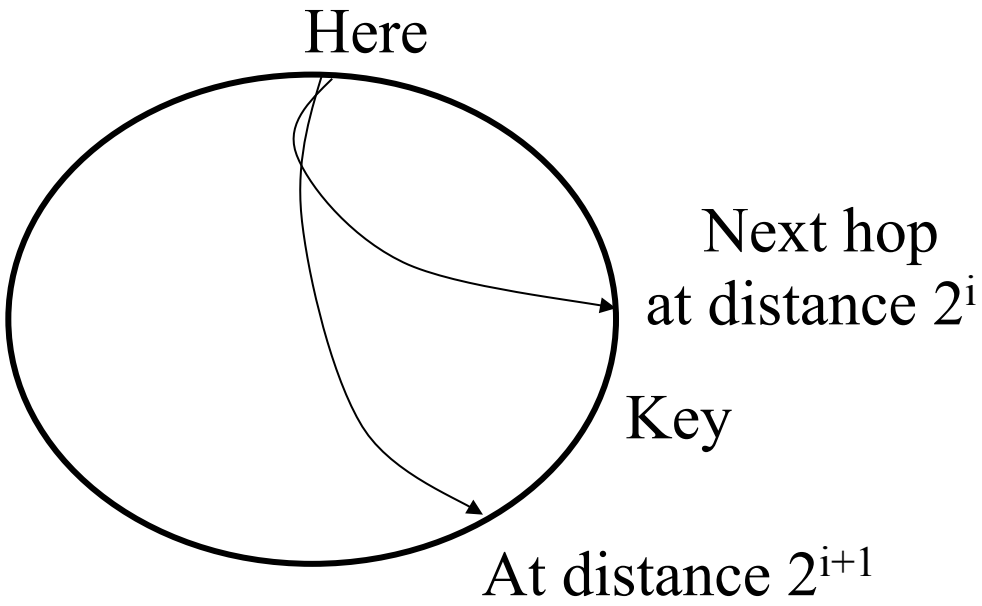
Key is found!

Analysis

Search takes O(log(N)) time

Proof Intuition:

- (intuition): at each step, distance between query and node-with-file reduces by a factor of at least 2



Analysis

Search takes O(log(N)) time

Proof Intuition:

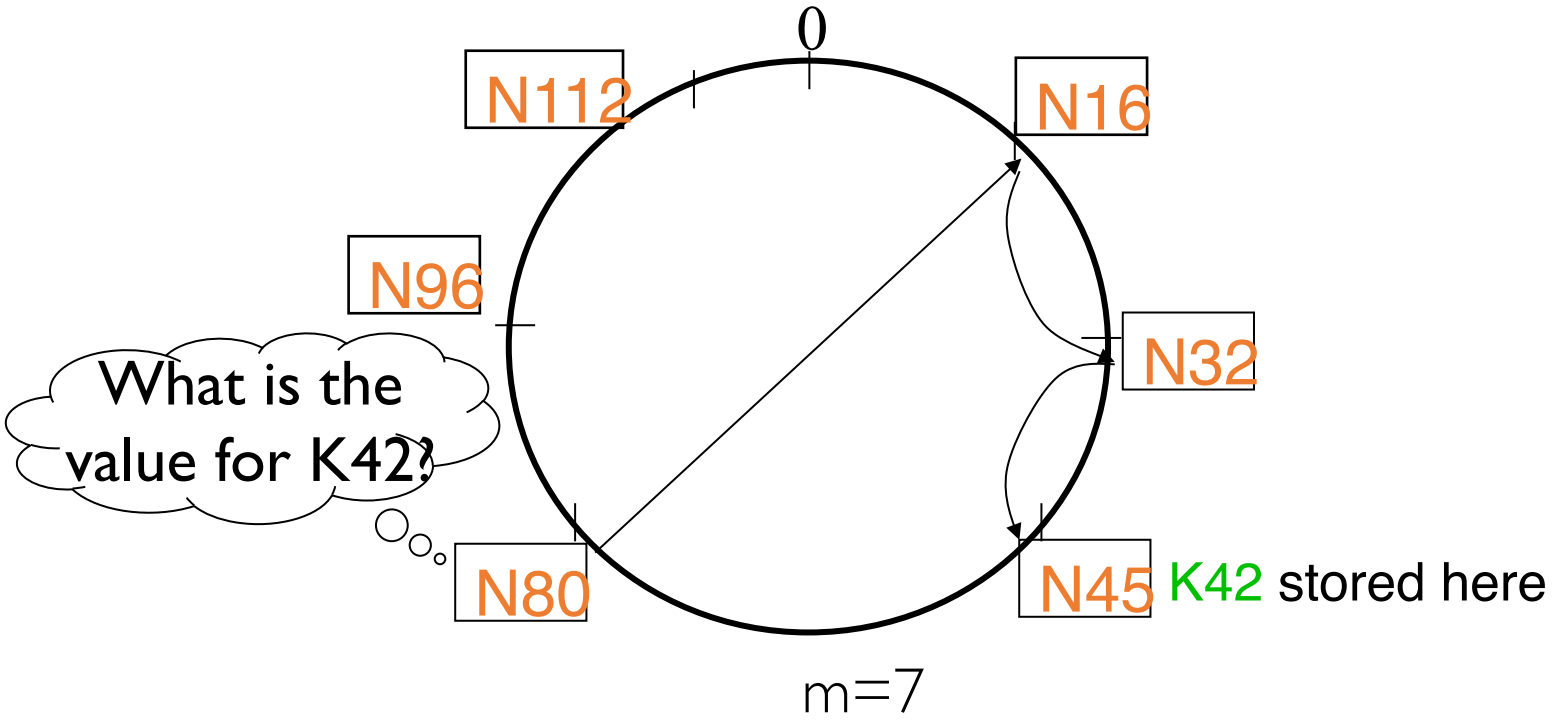
- (intuition): at each step, distance between query and node-with-file reduces by a factor of at least 2
- (intuition): after log(N) forwardings, distance to key is at most $2^m/2^{\log(N)} = 2^m / N$
- Expected number of node identifiers in a range of $2^m / N$:
  - ideally one
  - O(log(N)) with high probability (by properties of consistent hashing)

So using ring successors in that range will use another O(log(N)) hops. Overall lookup time stays O(log(N)).

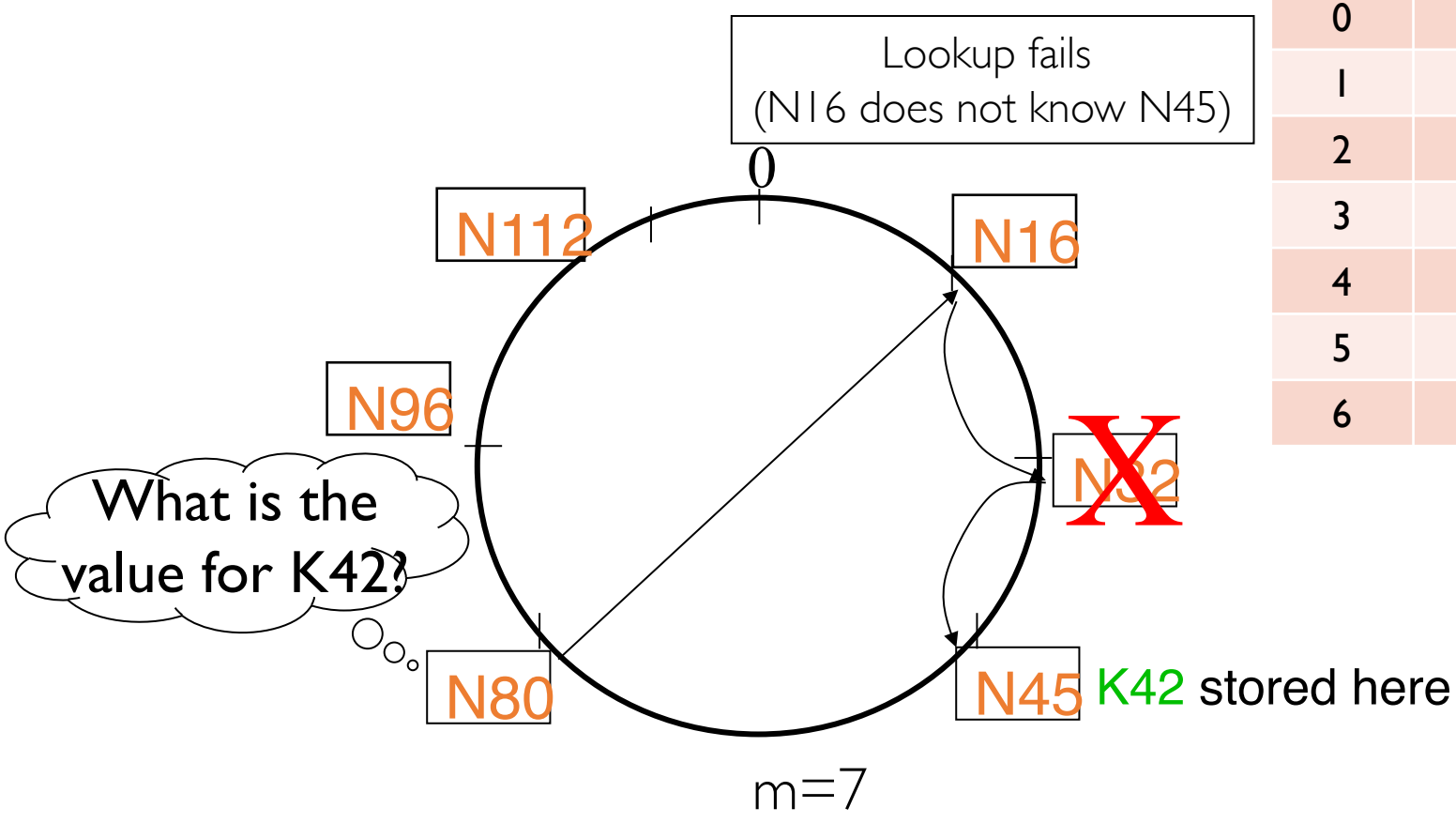
Analysis

- 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 and churn.

Search for key k at node n



If a node fails



N16's finger table

i	ft[i]
0	32
1	32
2	32
3	32
4	32
5	80
6	80

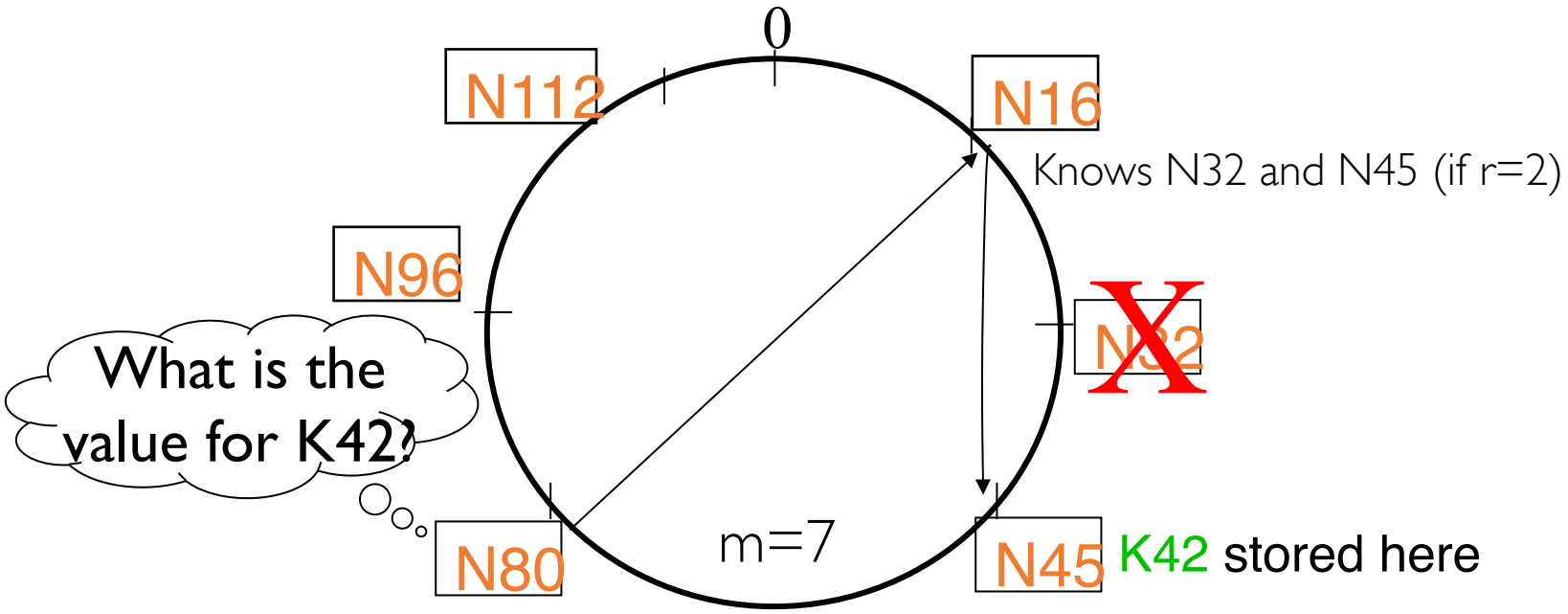
How do we handle this?

Handling failures

- When a node fails:
  - Need to update finger table (we will discuss how)
  - Might take some time.
- Lookups can fail or timeout in that duration.
  - The client can issue the query again.
- How can we increase robustness of the system until the finger tables get updated?
  - How do we minimize failure of look-ups?

If a node fails

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



Search under node failures

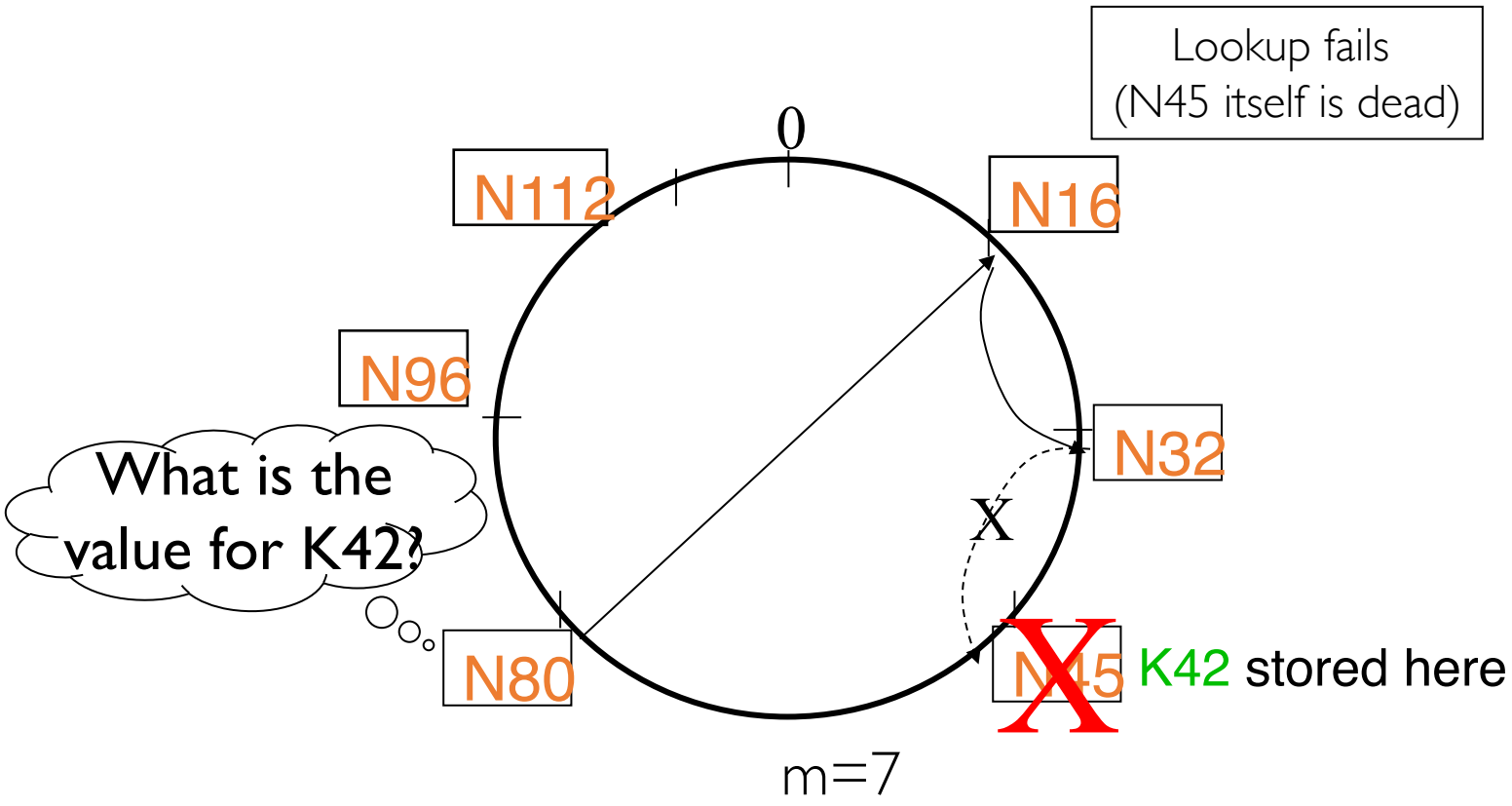
- If every node fails with probability 0.5, choosing r=2log(N) suffices to maintain lookup correctness (i.e. keep the ring connected) with high probability.
  - Intuition:
    - Pr(at given node, at least one predecessor alive)=

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

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

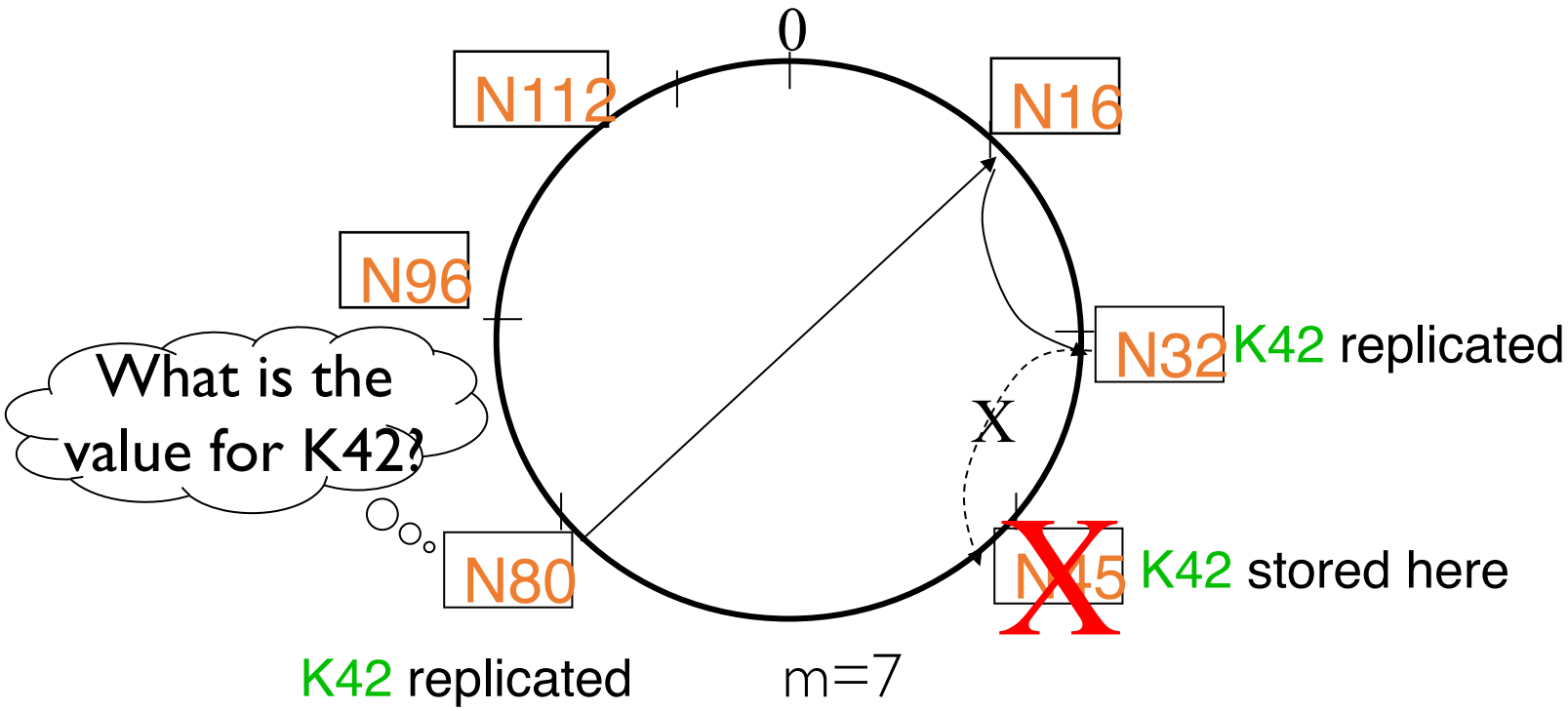
$$(1-\frac{1}{N^2})^{N/2}=e^{-\frac{1}{2N}}\approx 1$$

If a node fails



If a node fails

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



Need to deal with dynamic changes

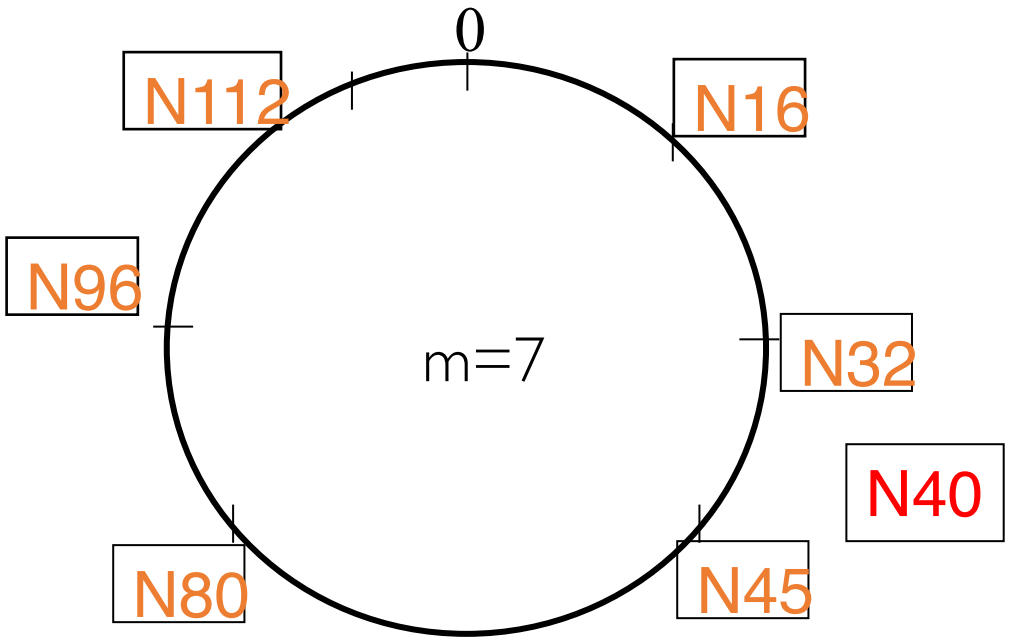
- Nodes fail
- New nodes join
- Nodes leave

So, all the time, need to:

→ Need to update successors and fingers, and copy keys

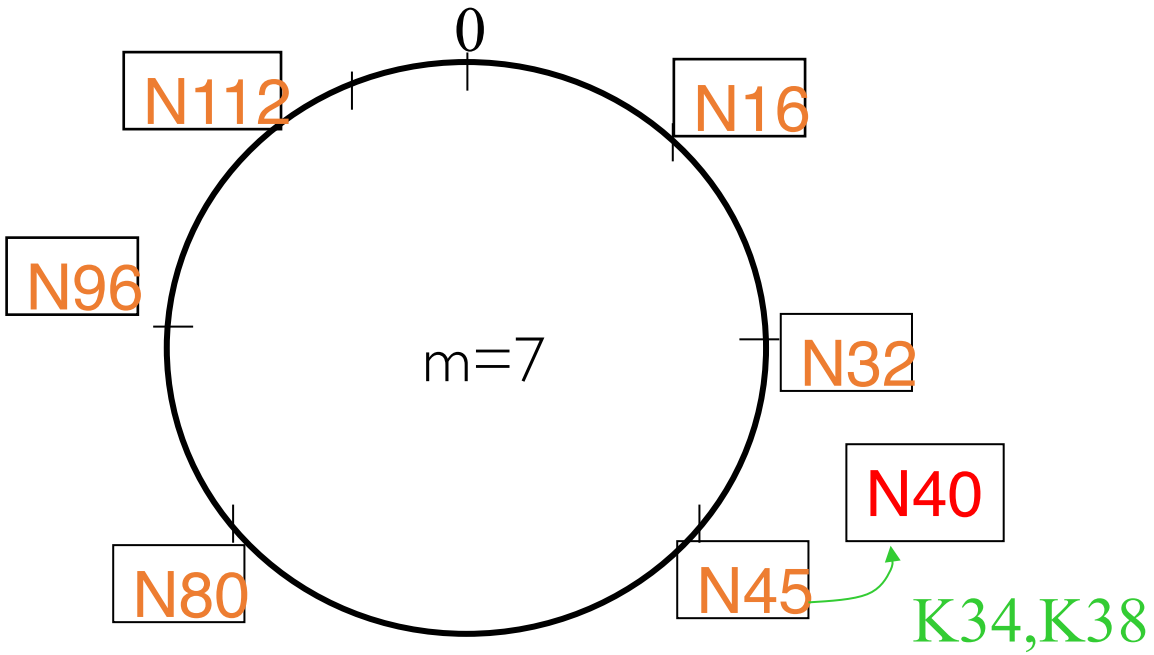
New node joins

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)



Concurrent Joins

- Aggressively maintaining and updating finger tables each time a node joins 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.

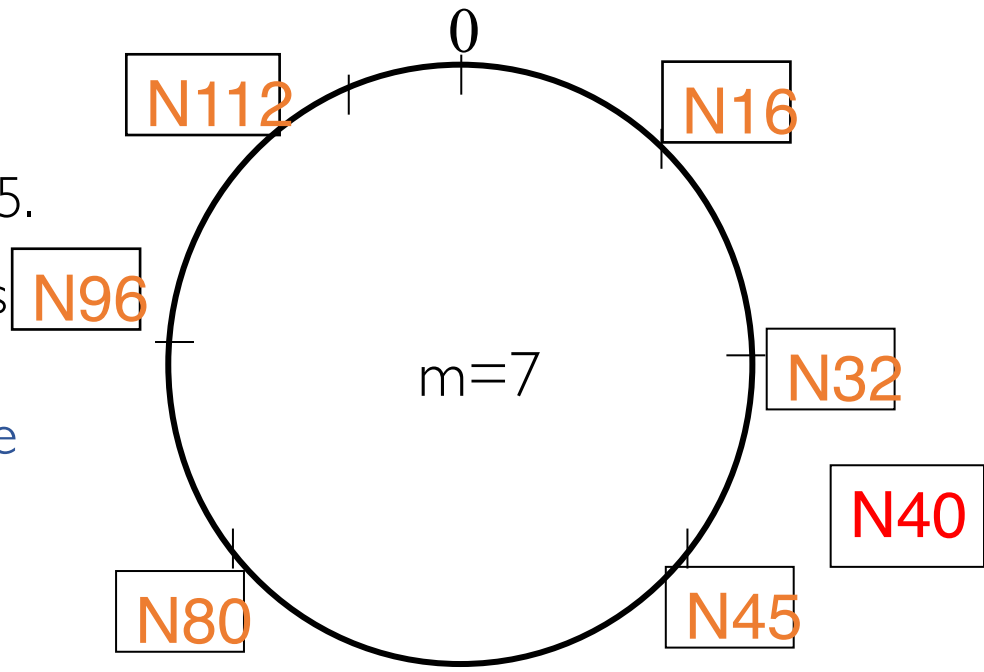
N40 initializes its ring successor to N45.

N40 notifies N45, and N45 updates its ring predecessor to N40.
N45 can begin transferring appropriate key-value pairs 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.
  - The predecessor of a failed node update's its ring successor, and notifies it.

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.