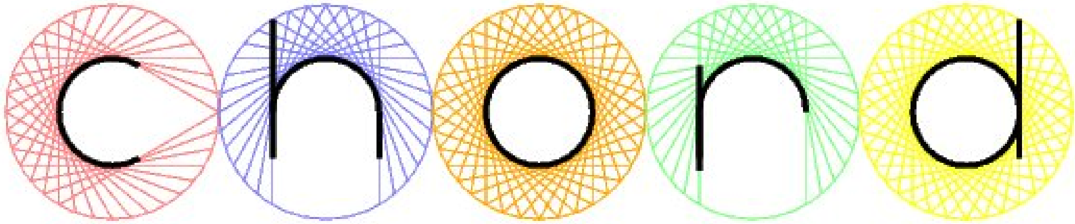


A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan

ACM SIGCOMM 2001

Presented by: Vassilis Lekakis

NOTE

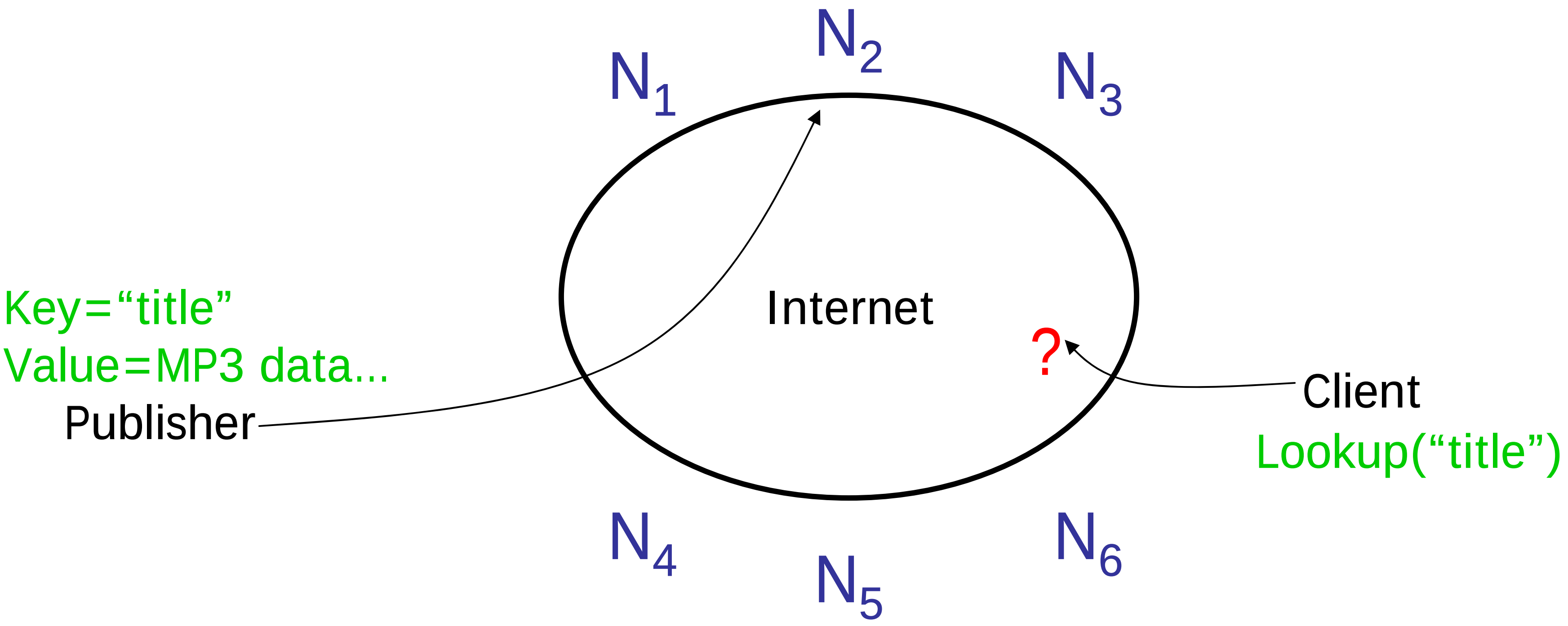
- (about Chord part)
  - Slides are based on a talk given by Robert Morris at Sigcomm 2001
  - Slide 28 is based on “A Survey and Comparison of

    Peer-to-Peer Overlay Network Schemes” , Jon Crowcroft et.al
- (about DDSN part)
  - Slides are based on a talk given by Russ Cox at IPTPS 2002

A peer-to-peer storage problem

- 1000 scattered music enthusiasts
- Willing to store and serve replicas
- How do you find the data?

The lookup problem



N_1 N_2 N_3

Key=“title”
Value=MP3 data...
Publisher

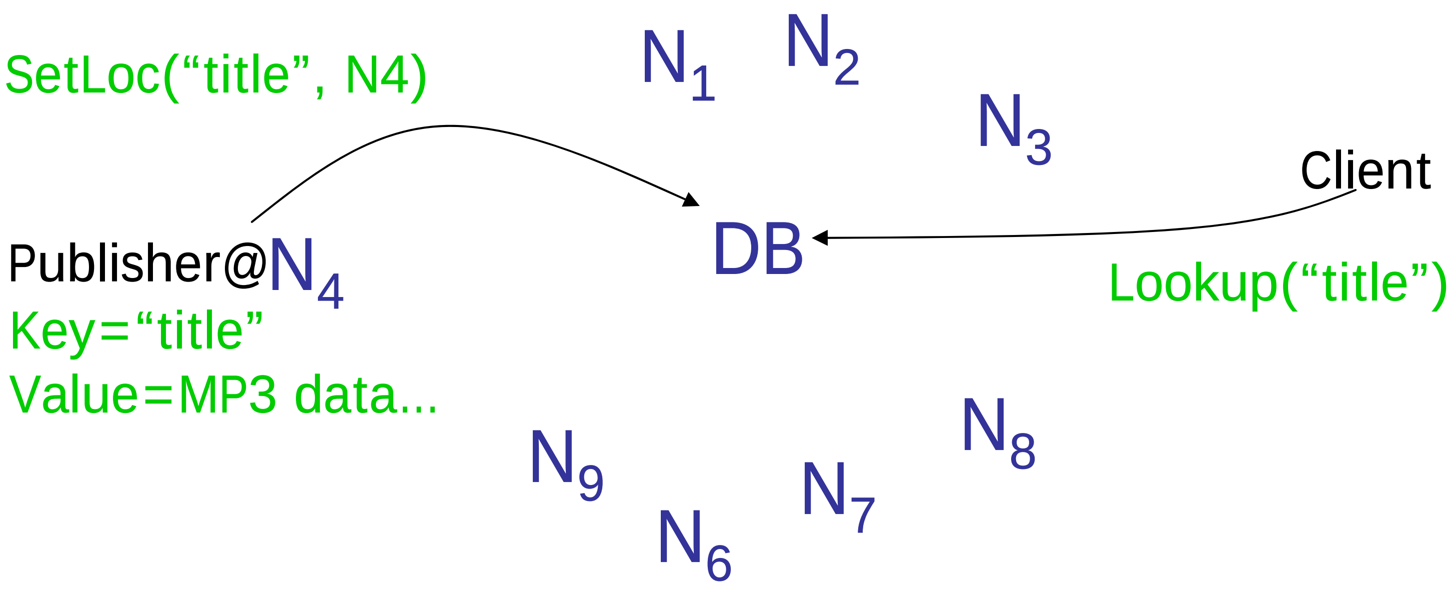
Internet

?

Client
Lookup(“title”)

N_4 N_5 N_6

Centralized lookup (Napster)



SetLoc(“title”, N4)

N_1 N_2

N_3

Client

Publisher@N_4

Key=“title”

Value=MP3 data...

DB

Lookup(“title”)

N_8

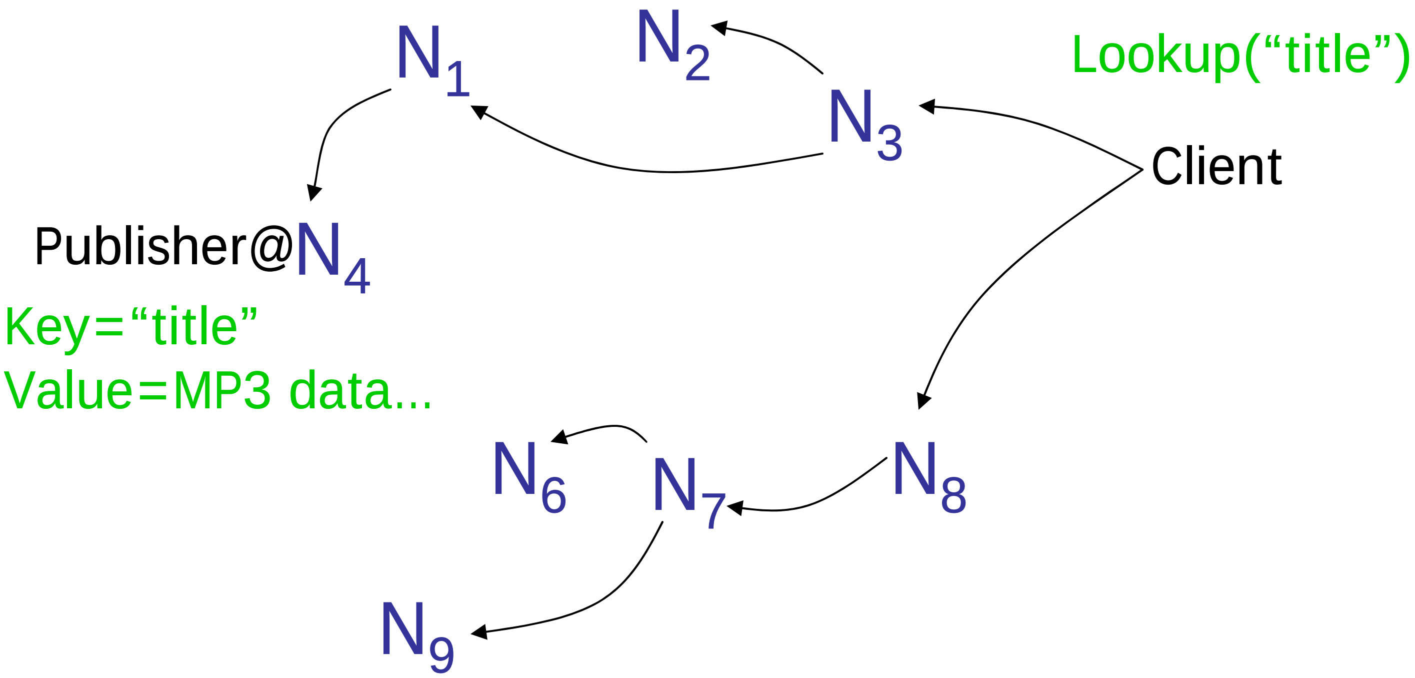
N_9

N_7

N_6

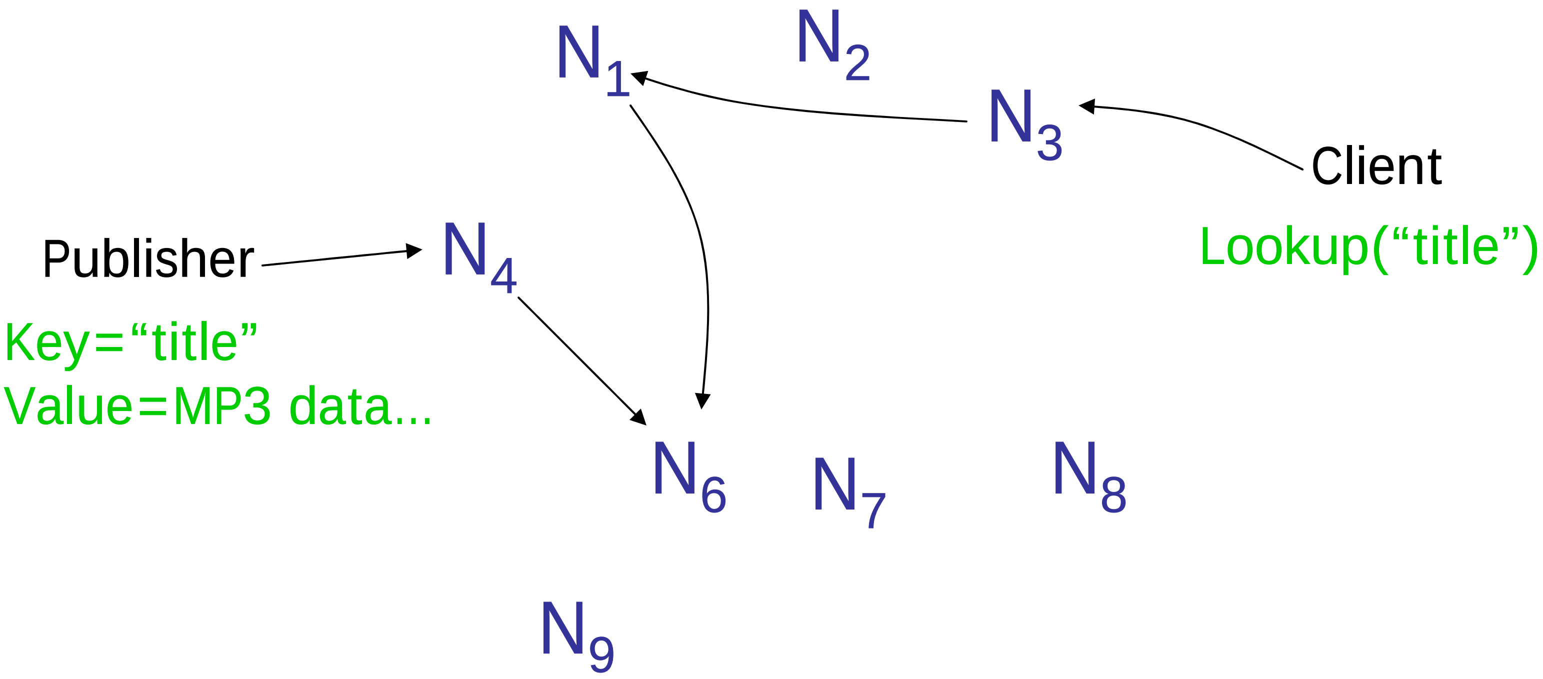
Simple, but O(N) state and a single point of failure

Flooded queries (Gnutella)



Robust, but worst case $O(N)$ messages per lookup

Routed queries
(Freenet, Chord, etc.)



N_1 N_2

N_3

Client

Publisher N_4

Lookup(“title”)

Key=“title”

Value=MP3 data...

N_6 N_7 N_8

N_9

Routing challenges

- Define a useful key nearness metric
- Keep the hop count small
- Keep the tables small
- Stay robust despite rapid change

- Freenet: emphasizes anonymity
- Chord: emphasizes efficiency and simplicity

Chord properties

- Efficient: O($log(N)$) messages per lookup
  - N is the total number of servers
- Scalable: O($log(N)$) state per node
- Robust: survives massive failures

- Proofs are in paper / tech report
  - Assuming no malicious participants

System Model

- Load Balance
- Decentralization
- Scalability
- Availability
- Flexible naming

- Runs as a service to high level sw
- App is responsible for
  - Authentication
  - Cashing
  - Replication
  - User friendly naming of data

Chord overview

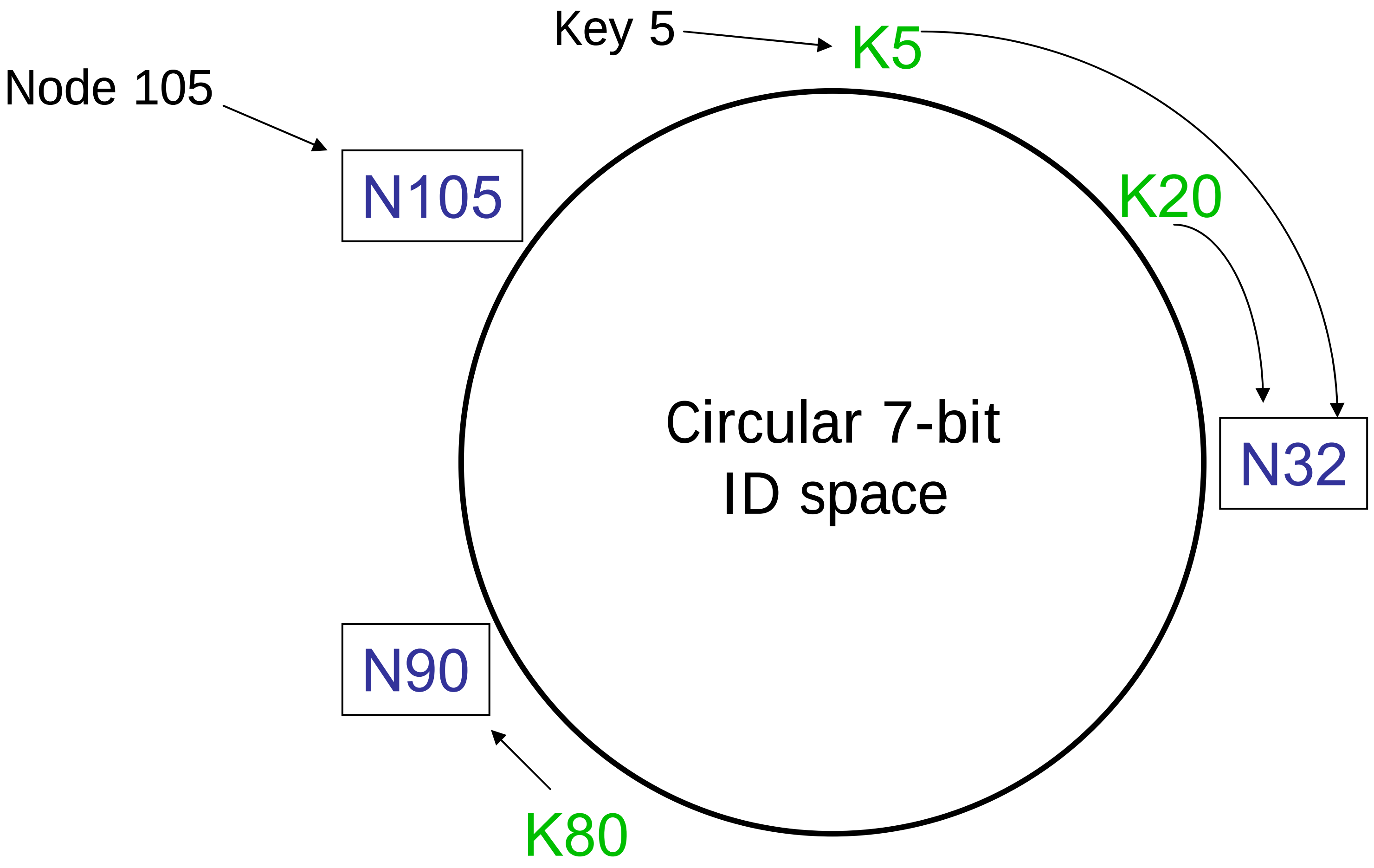
- Provides peer-to-peer hash lookup:
  - Lookup(key) → IP address
  - Chord does not store the data
- How does Chord route lookups?
- How does Chord maintain routing tables?

Chord IDs

- Key identifier = SHA-1(key)
- Node identifier = SHA-1(IP address)
- Both are uniformly distributed
- Both exist in the same ID space

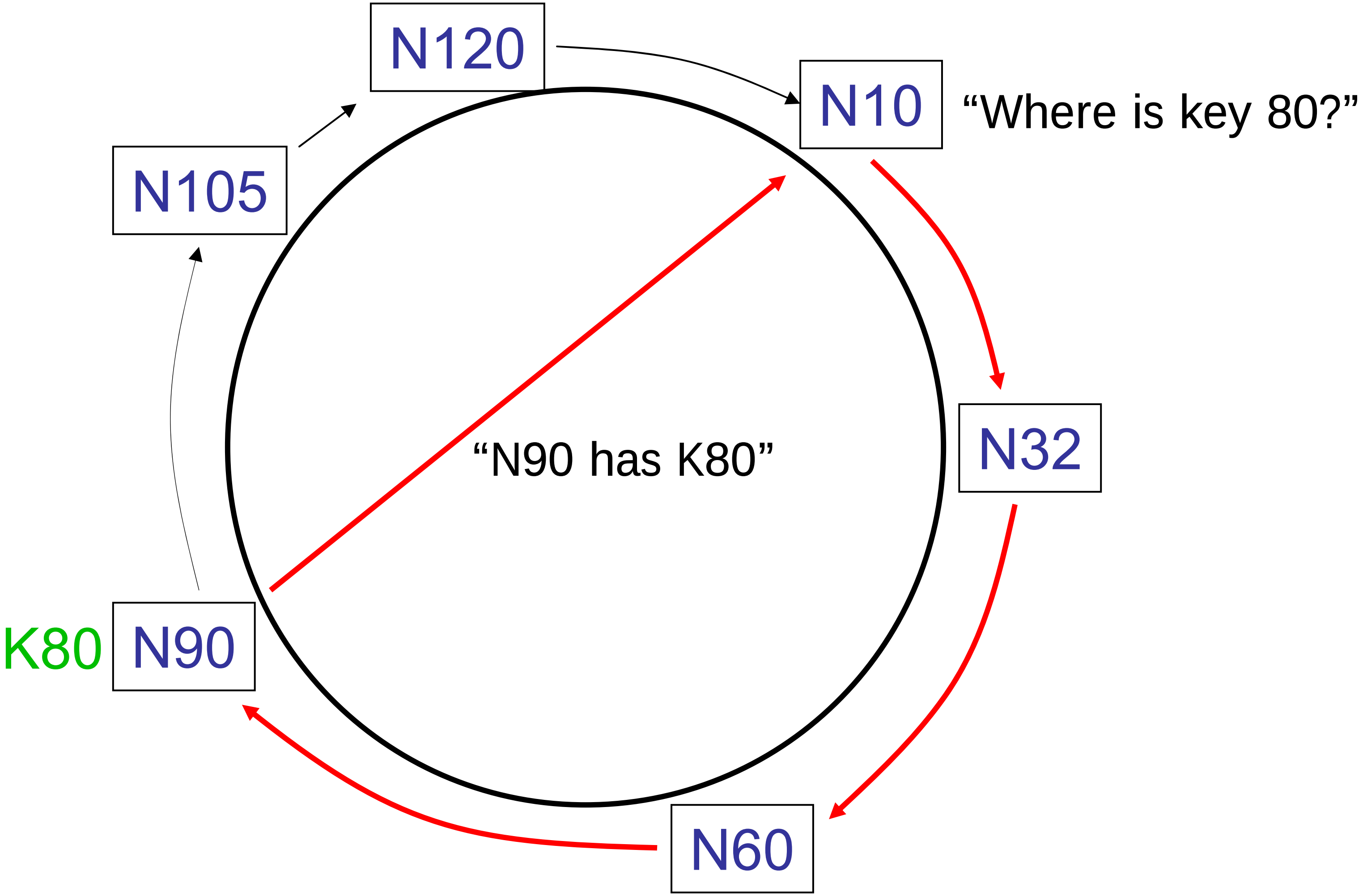
- How to map key IDs to node IDs?

Consistent hashing [Karger 97]



A key is stored at its successor: node with next higher ID

Basic lookup

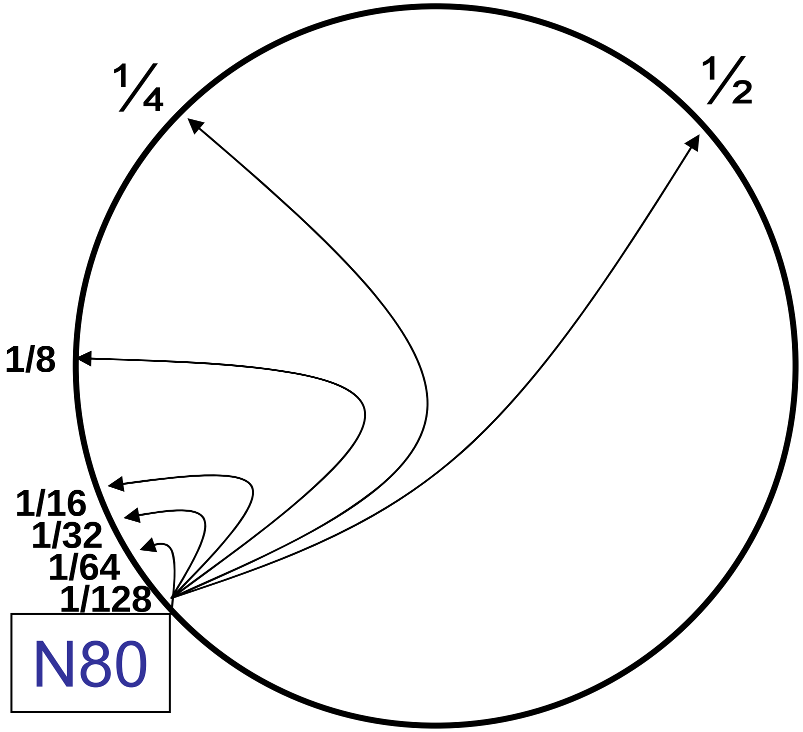


Simple lookup algorithm

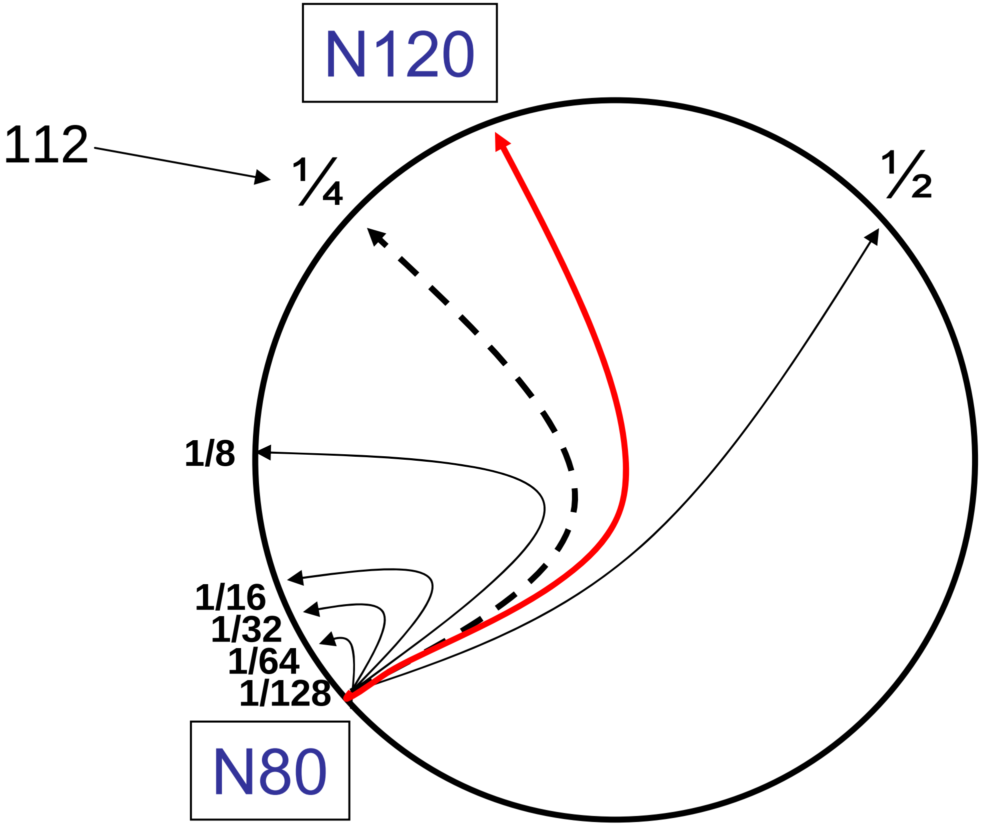
```
Lookup(my-id, key-id)
    n = my successor
    if my-id < n < key-id
        call Lookup(id) on node n   // next hop
    else
        return my successor                // done
```

- Correctness depends only on successors

“Finger table” allows log(N)-time lookups



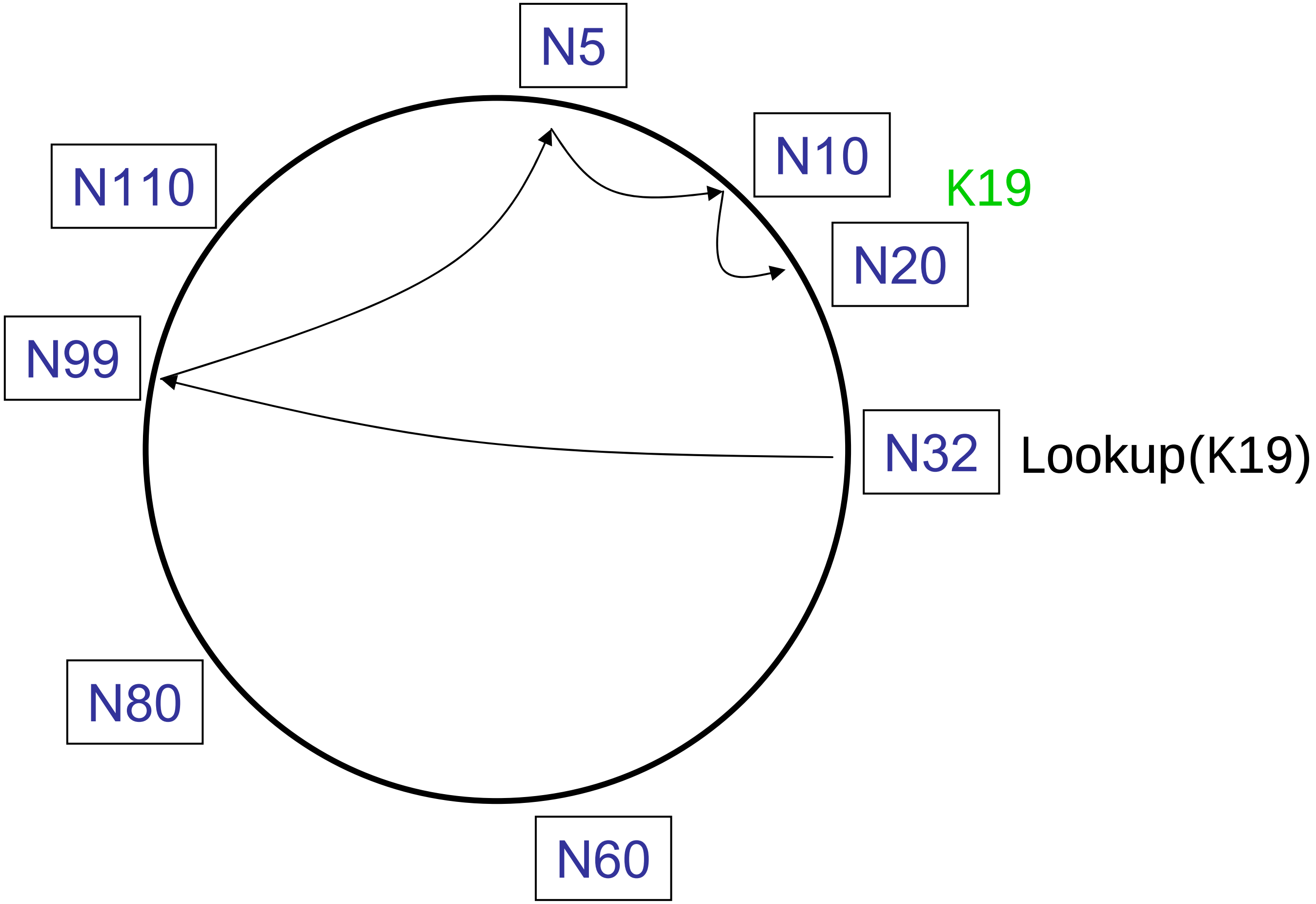
Finger *i* points to successor of $n+2^i$



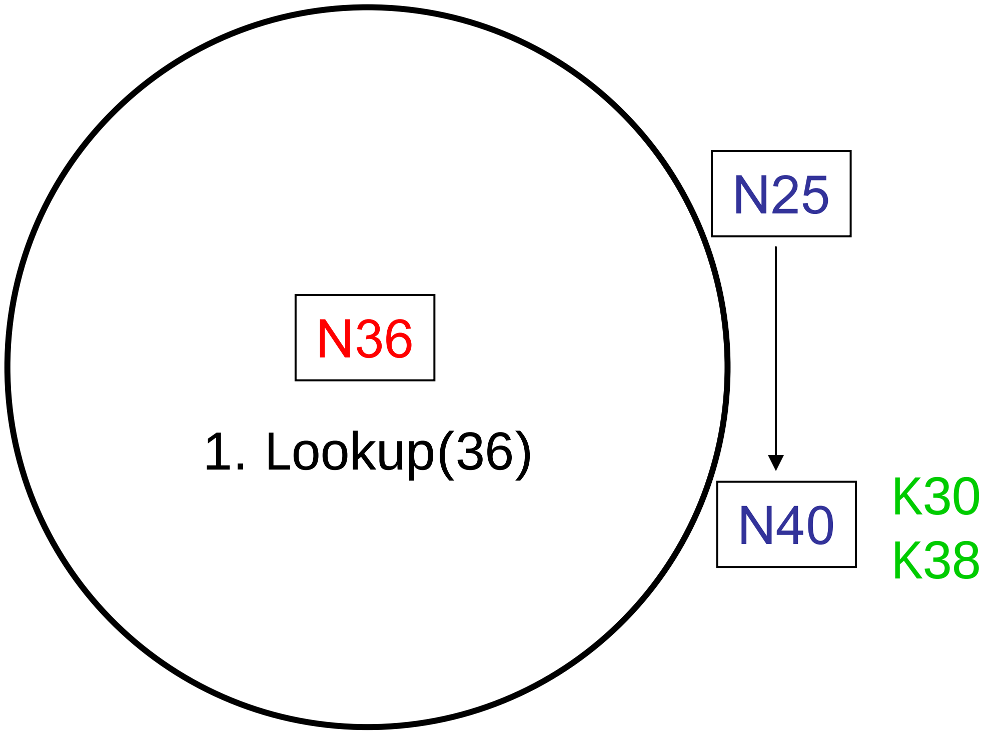
Lookup with fingers

```
Lookup(my-id, key-id)
    look in local finger table for
        highest node n s.t. my-id < n < key-id
    if n exists
        call Lookup(id) on node n        // next hop
    else
        return my successor              // done
```

Lookups take O($log(N)$) hops



Joining: linked list insert



N25

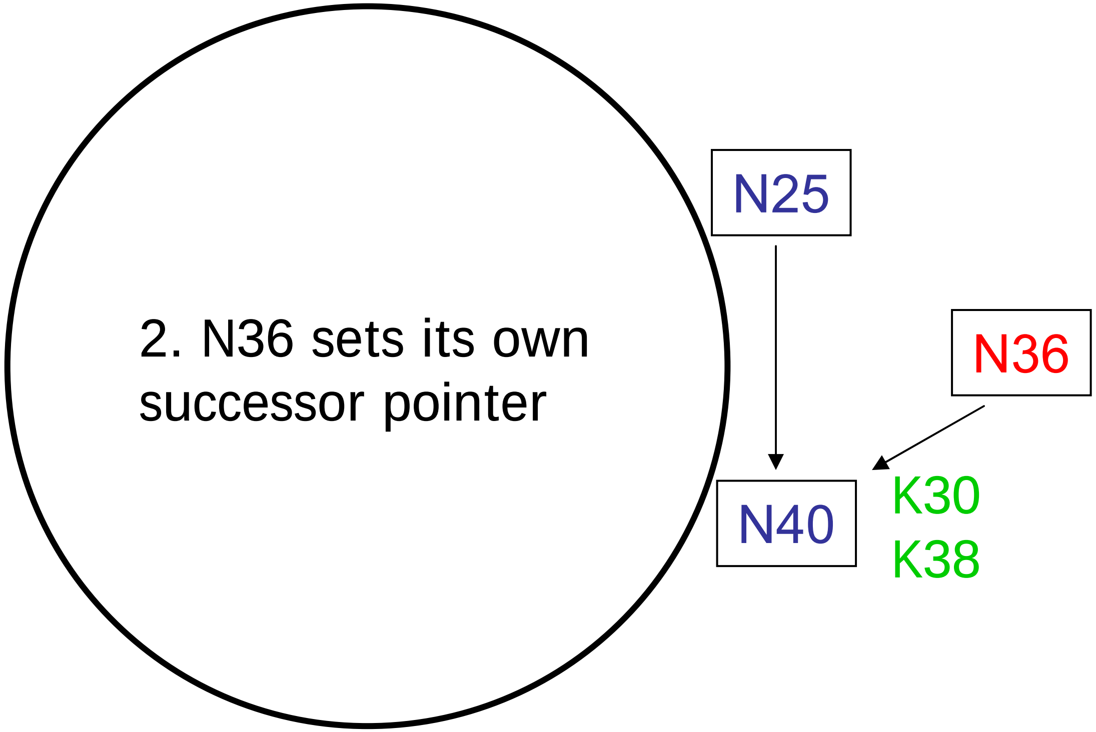
N36

1. Lookup(36)

N40

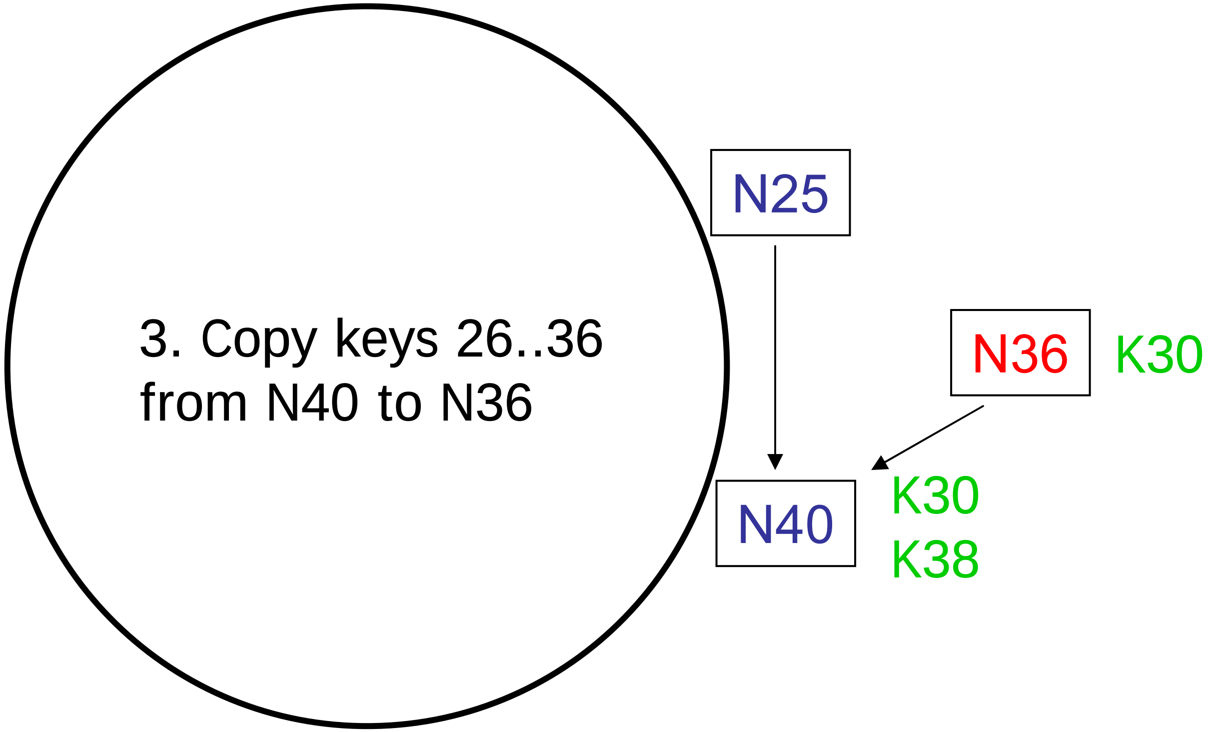
K30
K38

Join (2)



2. N36 sets its own successor pointer

Join (3)



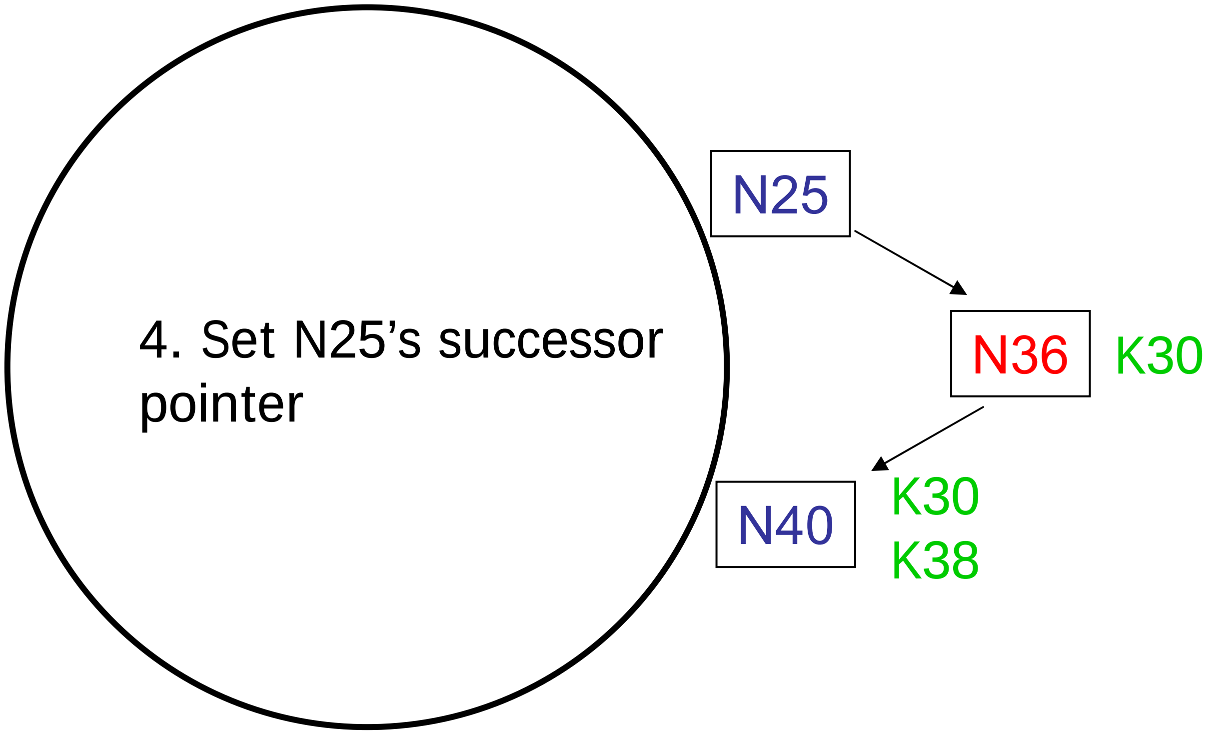
3. Copy keys 26..36
from N40 to N36

N25

N36 K30

N40 K30
K38

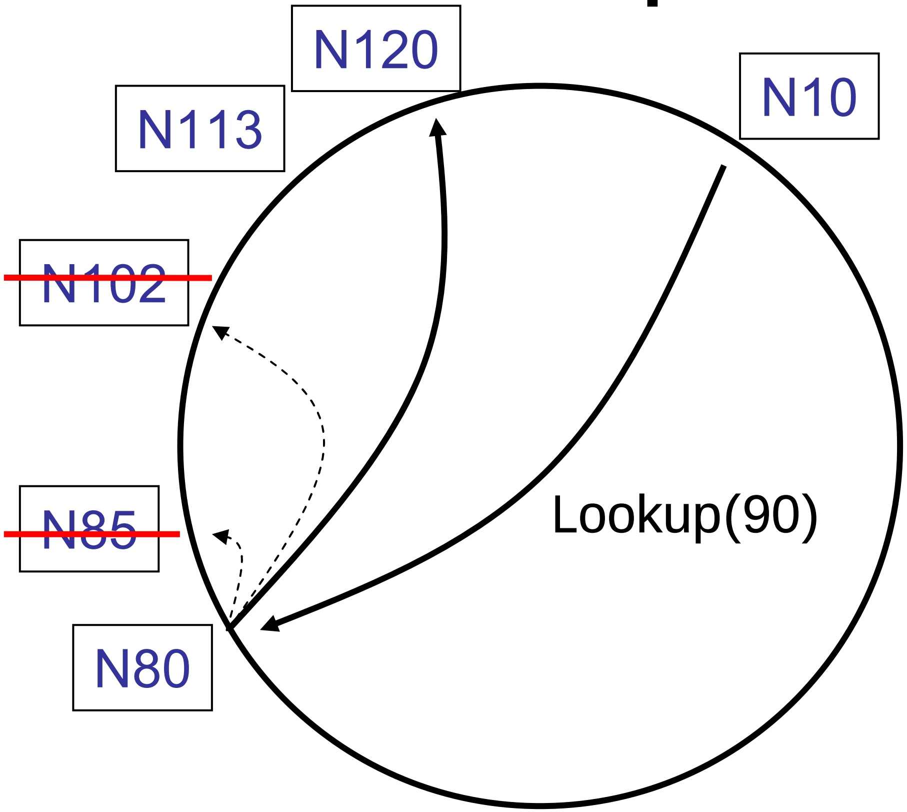
Join (4)



4. Set N25's successor pointer

Update finger pointers in the background
Correct successors produce correct lookups

Failures might cause incorrect lookup



N80 doesn’t know correct successor, so incorrect lookup

Solution: successor lists

- Each node knows r immediate successors
- After failure, will know first live successor
- Correct successors guarantee correct lookups

- Guarantee is with some probability

Lookup with fault tolerance

```
Lookup(my-id, key-id)
  look in local finger table and successor-list
      for highest node n s.t. my-id < n < key-id
  if n exists
      call Lookup(id) on node n      // next hop
      if call failed,
          remove n from finger table
          return Lookup(my-id, key-id)
  else return my successor           // done
```

Misc

- Working implementation as part of CFS
- Chord library: 3,000 lines of C++
- Has been used in:
  - Cooperative File System (CFS) for distributed read-only storage (SOSP ’01)
  - Ivy, a p2p file system (OSDI ’02) (read/write)
  - DDNS, a p2p DNS (IPTPS 02)

System	***CAN***	***Chord***
Unit	DHT	
Architecture	Multi-dimensional ID coordinate space	Unidirectional and circular id space
Lookup	Key, value pairs to map a point P in the coordinate spac	Matching key and NodeID
System parameters	N - #peers D-#dimensions	N - #peers
Routing Performance	$O(d.N^{1/d})$	$O(\log N)$
Routing State	2d	$\log N$
Join/Leave	2d	$(\log N)^2$
Security	Low level – both suffer from man-in-the-middle attacks	
Reliability/ Fault Tolerance	Failure of peers will not cause network-wide failures	
Where	?	As a service/linked lib to high level sw

Serving DNS using a p2p lookup service

Russ Cox, Athicha Muthitacharoen, Robert Morris

Presented by: Vassilis Lekakis

Overview

- The experiment: redo DNS in a peer−to−peer manner.
- The result: not as good as conventional DNS.
- The talk: what we expected, what we learned.
  - *Draw general conclusions about peer−to−peer systems.*
  - *Directions for future research.*
  - *Or guidelines for selecting peer−to−peer apps*

Motivation

- Before DNS there was a global *hosts.txt*.
- **DNS is an attempt to distribute *hosts.txt*, but:**
  - *Everyone has to be a DNS admin.*
  - *I can't have a domain without a 24/7 DNS server.*
  - *Locally correct, globally wrong configurations.*
- ***P2P lookup systems might fix these:***
  - *Organization, replication, much configuration handled by the P2P layer.*
  - *I don't need to keep a 24/7 server up.*
  - *Lack of hierarchy avoids half−broken configs (?)*

DNS & DNS SEC

- Original DNS uses IP based authentication
- DNSSEC uses crypto based authentication
- DNS SEC separates serving from authedication

- Can we explore alternate lookup methods?

  (p2p dynamic hash Tables)

DNS using P2P Hash Table

- Look up SHA1(*name*, *query type*).
- Answers RRSets like DNS
- It works just like a distributed host.txt
  - Prototype implemented in Chord
- Stores fixed number of replicas

Evaluation: Latency

- Uncached latency is too big O(log n) RPCs
  - Chord : log base = 2
  - Pastry, Kademlia: log base = 16
  - DNS, log base ?? (>> 1,000,000)

Evaluation: Robustness

- DDNS: Inherited from Chord
- DNS: fairly robust already
  - Root servers are highly replicated
  - DOS attack to old anymore

Evaluation: Loss of network Connectivity

- Suppose UOC gets cut off from Internet
  - In DNS
    - UOC can still connect to UOC hosts
    - UOC cannot connect to Internet hosts
    - Internet cannot lookup nor connect to UOC hosts
  - In P2P DNS
    - UOC can't look up but can connect to UOC hosts (??)
    - UOC can look up but can't connect to Internet hosts.
    - Internet can look up but can't connect to UOC hosts

Evaluation: Functionality

- DDNS : functionality of a distributed host.txt

- BUT
  - *No dynamically generated records*
  - *No support for "ANY" queries*
  - *No server side load balancing*

Evaluation: Administration

- DNS
  - requires significant expertise to administer
  - Bad configurations
- DDNS
  - Ease of deployment
  - 24/7 servers uptime?
  - Why trust servers run by others?
  - Users need incentives in order to run servers

Conclusions

- P2P systems have fundamental limitations and simply aren’t appropriate for apps that need
  - Lower latency.
  - Protection against insertion DoS.
  - Choice of functionality for network outages.
  - More than just distributed hash tables.
  - High confidence in the network.
  - Generic incentives for people to run servers