Peer-to-Peer Lookup Services

Jeng-Long Chiang

Jan. 22, 2003

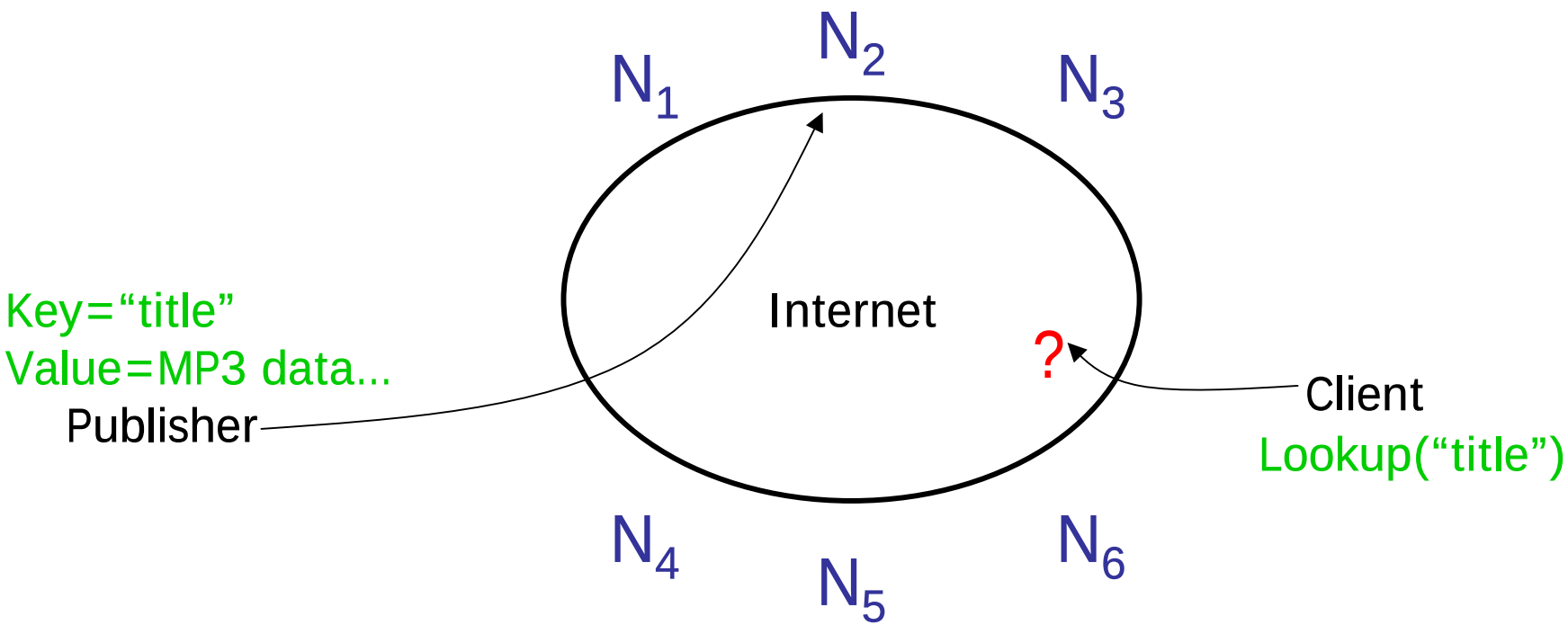
References

- K. Aberer et al., “Improving Data Access in P2P Systems”, *IEEE Internet Computing*, Jan/Feb 2002.
- S. Ratnasamy et al., “A Scalable Content-Addressable Network”, *SIGCOMM 2001*.
- I. Stoica et al., “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”, *SIGCOMM 2001*.
- H. C. Hsiao and C. T. King, “Tornado: Capability-Aware Peer-to-Peer Storage Networks”, to appear in *IPDPS 2003*.
- K. Aberer et al., “Peer-to-Peer Information System: Concepts, Models, State-of-the-Art, and Future Systems”, Tutorial, *ICDE 2002*.
- K. Aberer, “Data Access in Peer-to-Peer Information System”, Talk, 2001.
- R. Morris et al., “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, Talk, 2001.
- R. Morris et al., ” Building Peer-to-Peer Systems with Chord: a Distributed Lookup Service”, Talk, 2001.

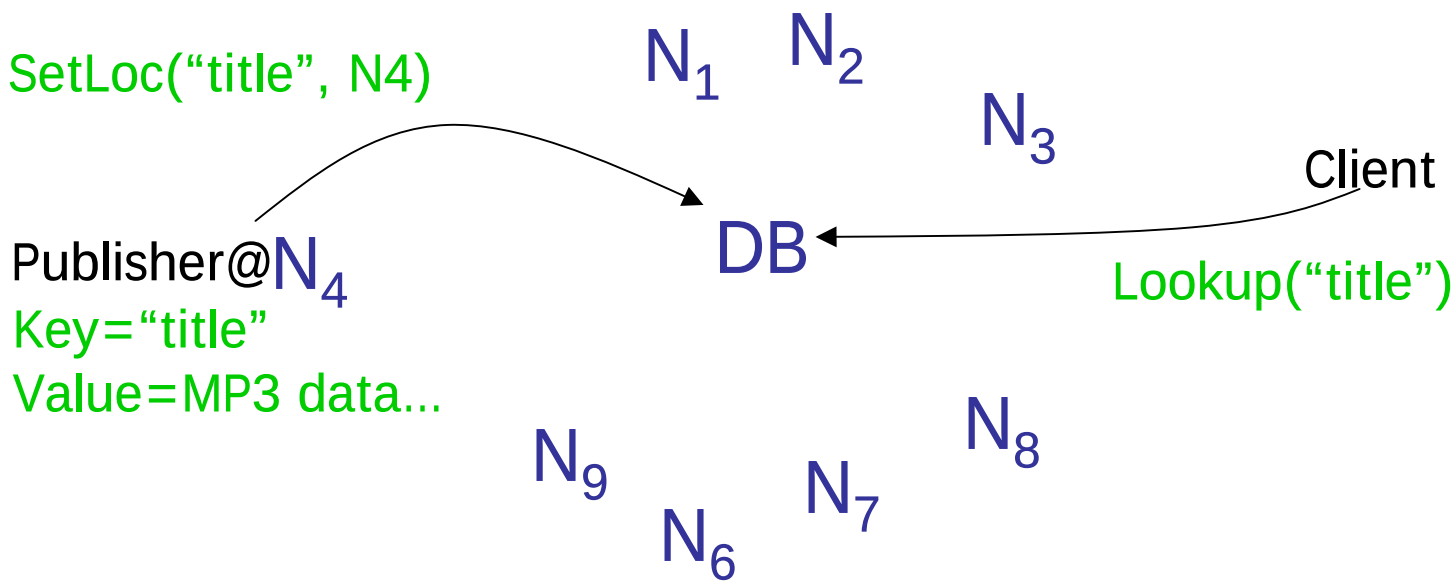
Peer-to-Peer(P2P)

- Properties:
  - No central coordination
  - No central database
  - No peer has a global view of the system
  - Global behavior emerges from local interactions
  - All existing data and services are accessible from any peer
  - Peers are autonomous
  - Peers and connections are unreliable
  - Peers are heterogeneous

The lookup problem

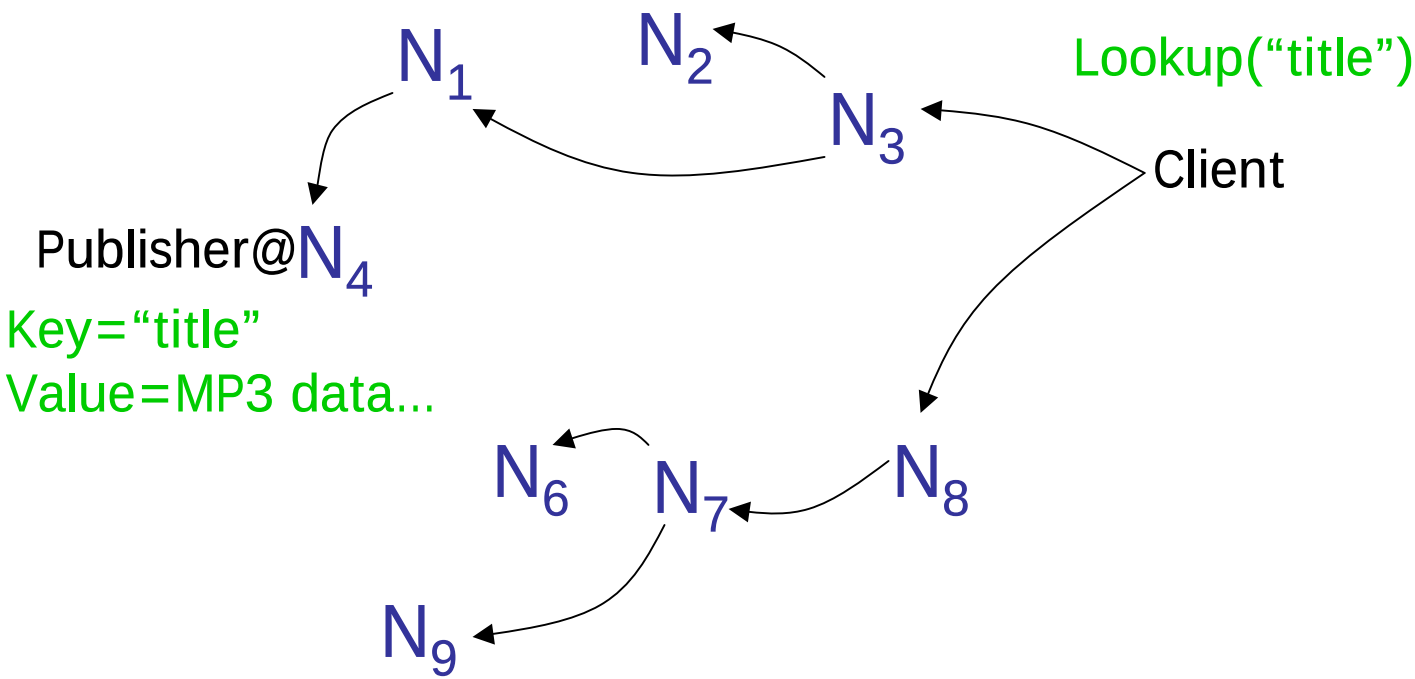


Centralized lookup (Napster)



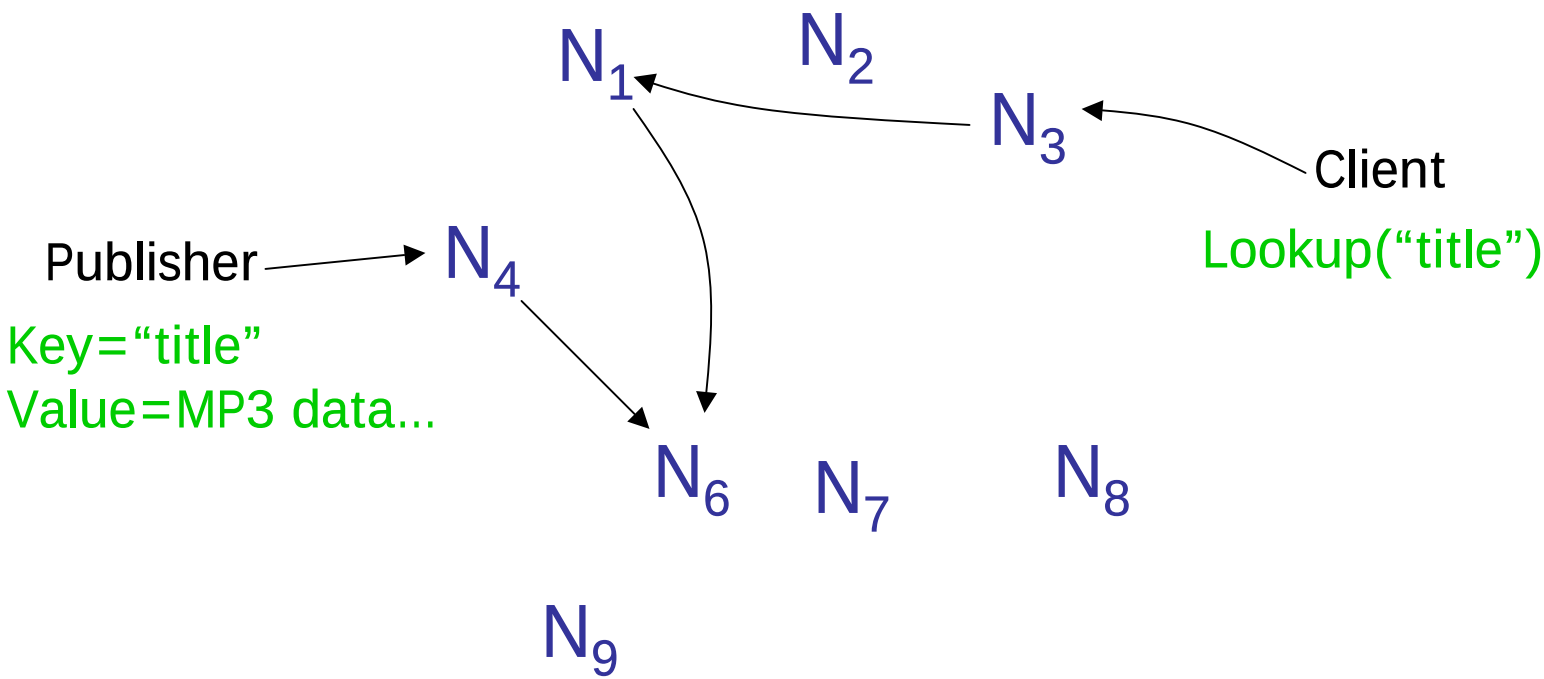
Simple, but O(N) state and a single point of failure

Flooded queries (Gnutella)



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

Routed queries



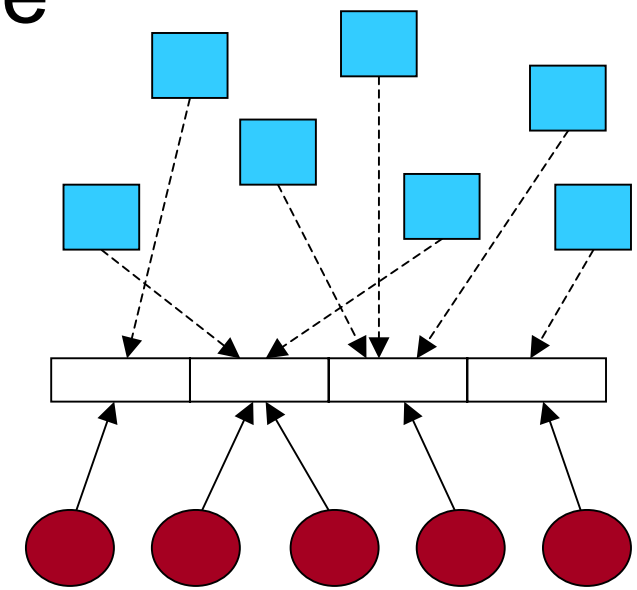
e.g. P-Grid, CAN, Chord, Tornado, …

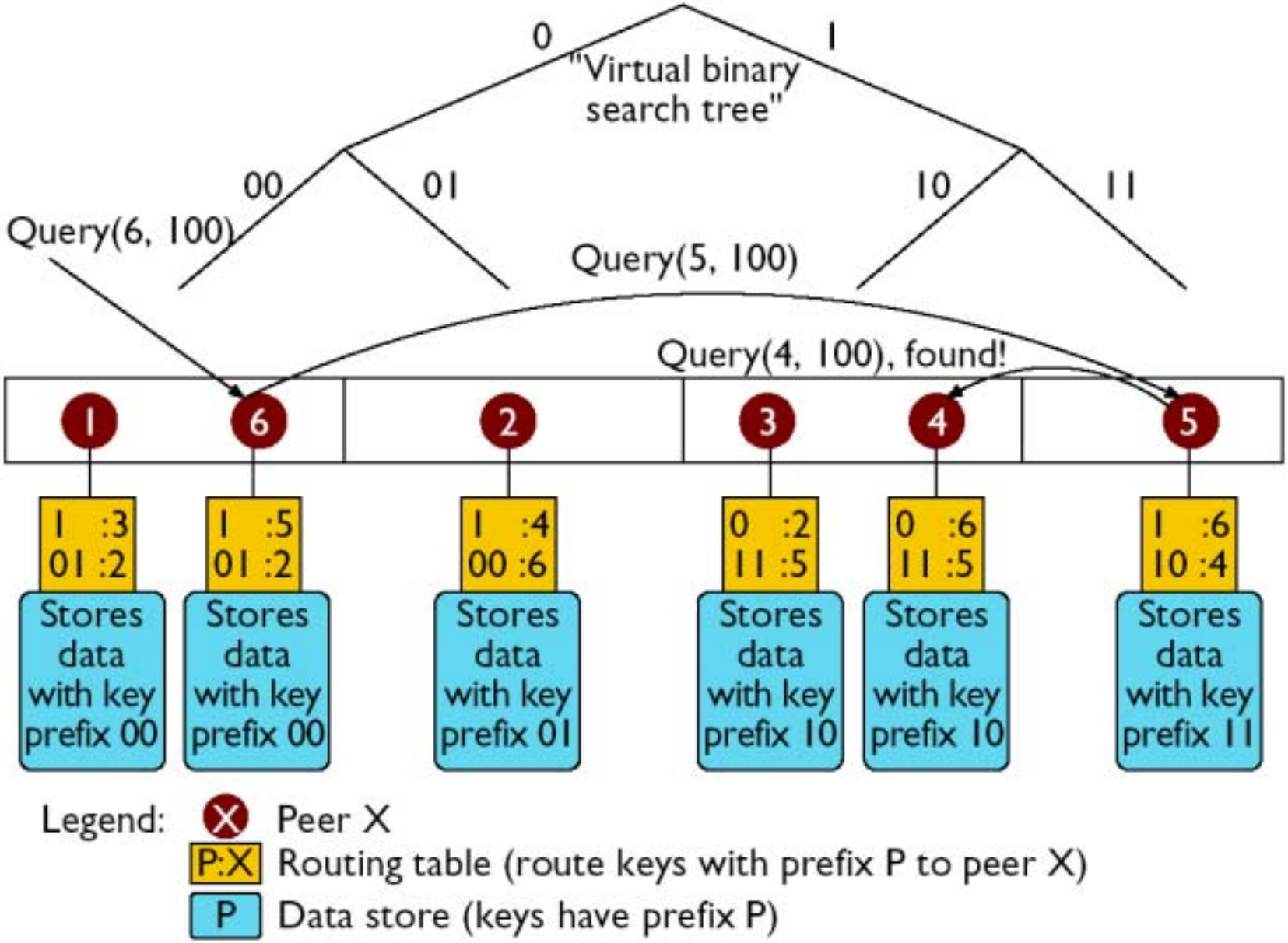
Comparison Criteria

- Infrastructure
  - Tree, space, …
- Routing
  - Hashing, …
- Searching
  - Equality, prefix, …
- Performance
  - Search, update, node join/leave, …
- Robustness

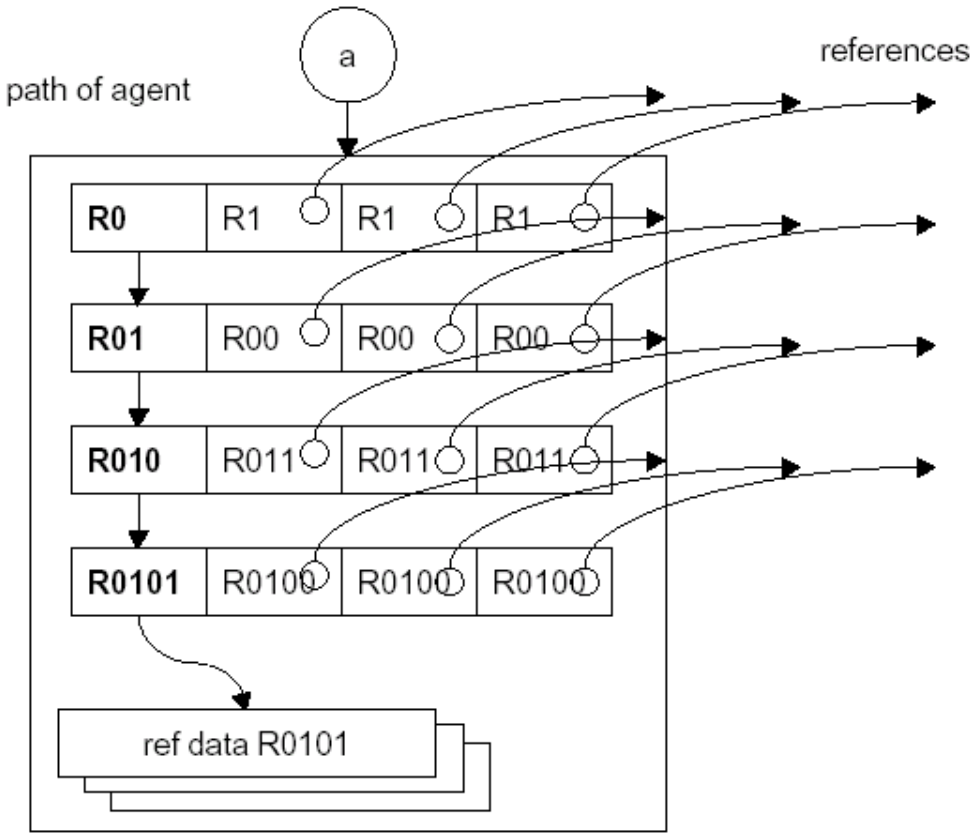
P-Grid

- Virtual binary search tree
- Hashing (data)
  - “title”->0110xx…
- Prefix (hashed keys)
- Robustness
  - Replication
- Data items stored at peer if node identifier is prefix of data key




0
1
"Virtual binary search tree"
00
01
10
11
Query(6, 100)
Query(5, 100)
Query(4, 100), found!
1
6
2
3
4
5
1 :3
01 :2
1 :5
01 :2
1 :4
00 :6
0 :2
11 :5
0 :6
11 :5
1 :6
10 :4
Stores data with key prefix 00
Stores data with key prefix 00
Stores data with key prefix 01
Stores data with key prefix 10
Stores data with key prefix 10
Stores data with key prefix 11
Legend:
Peer X
P:X
Routing table (route keys with prefix P to peer X)
P
Data store (keys have prefix P)

Data Structure of an Agent



P-Grid Construction Algorithm (Bootstrap)

- When agents meet (randomly)
  - Compare the current search paths p and q
- Case 1: p and q are the same
  - If the maximal path length is not reached extend the paths and split search space, i.e. to p0 and q1
- Case 2: p is a subpath of q, i.e. q = p0…
  - Extend p by the complement of q, i.e. p1
- Case 3: only a common prefix exists
  - Forward to one of the referenced peers
  - Limit forwarding by *recmax*

0,0 -> 00,01

0,01 -> 00,01

00,011 -> r01,011 -> 010,011

4. Search and Update

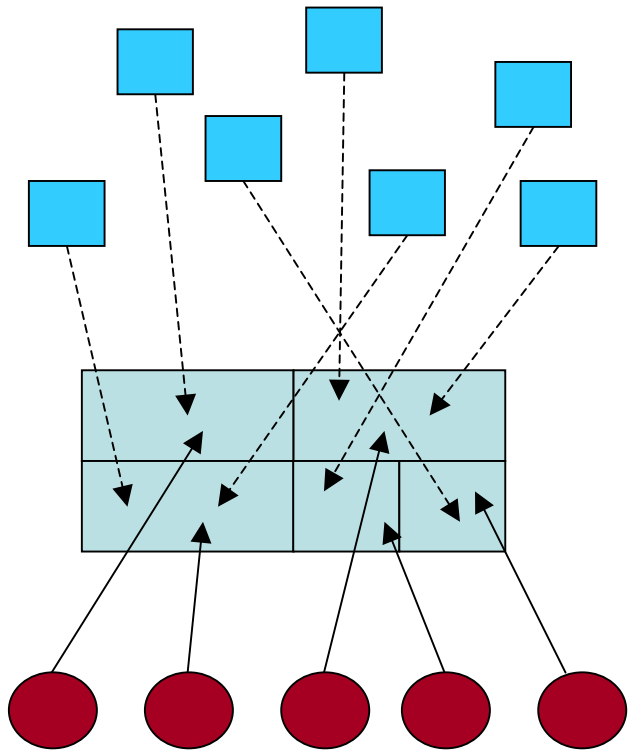
- Search straightforward **<O(logN)**
  - Follow own path or references
  - At most k steps k: key(path) length, k<logN
  - If multiple references are online, select randomly
- Updates **O(N/2^k)**
  - All replicas need to be found
  - Repeated searches
    - Breadth first (limited recursion breadth)
    - Depth first
    - Depth first and contact buddies with same key

Node Join and Leave

- M: total data items
- Data items in a node: **O(M/2^k)**
- Node join: **O(M/2^(i+1))**, 0<=i<=k-1
  - Select an identifier and split data items
  - Select an identifier and replicate data items
- Node leave: **O(M/2^k)**
  - If there are other nodes in an identifier, just leave
  - If it is the last node in an identifier, merge it's data items with a node on the other sub-tree
- Involve node: **O(1)**

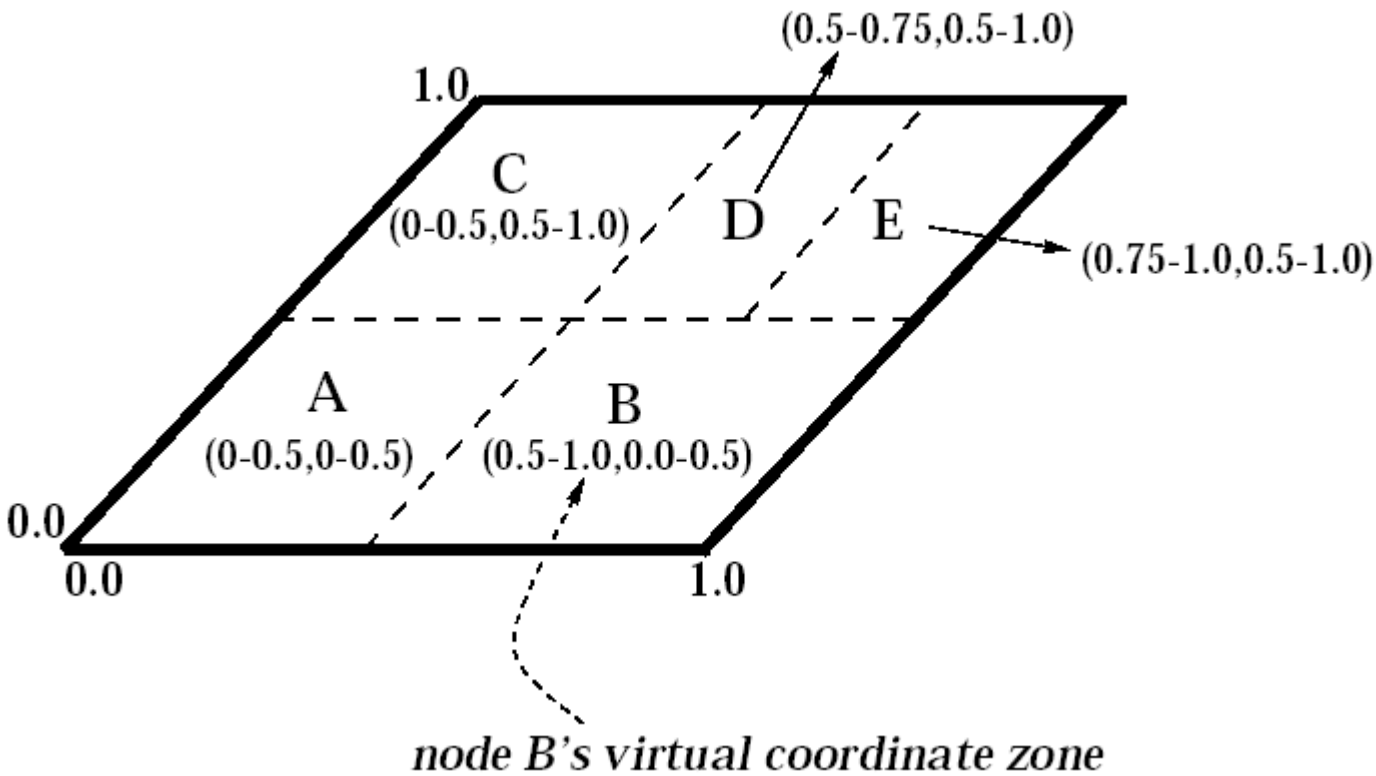
CAN

- Multi-dimensional space
- Hashing (data)
  - E.g. “title”->(x, y, z, …)
- Equality
- Robustness
  - Multiple hashing space



- Requests for a particular key are routed to the CAN node whose zone contains that key.

2-space with 5 nodes

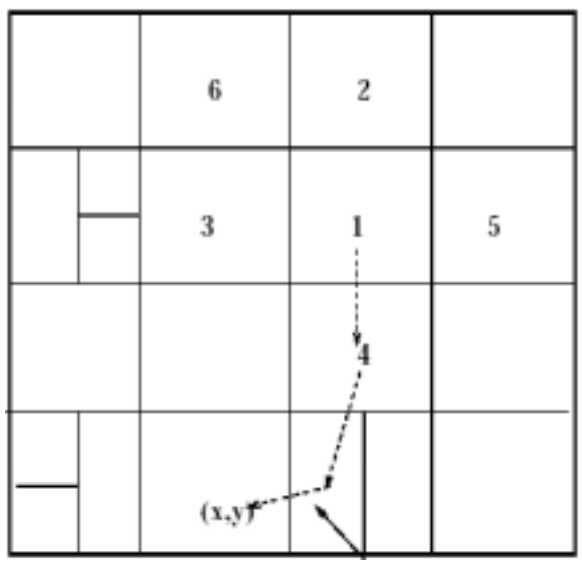


Routing

- Straight line from source to destination
- Coordinate routing table
  - IP addresses and coordinate zone of neighbors

Neighbors in d-dimensional coordinate space:
•Overlap along d-1 D
•Abut along one D

Search: **O(dN^1/d)**



CAN Construction

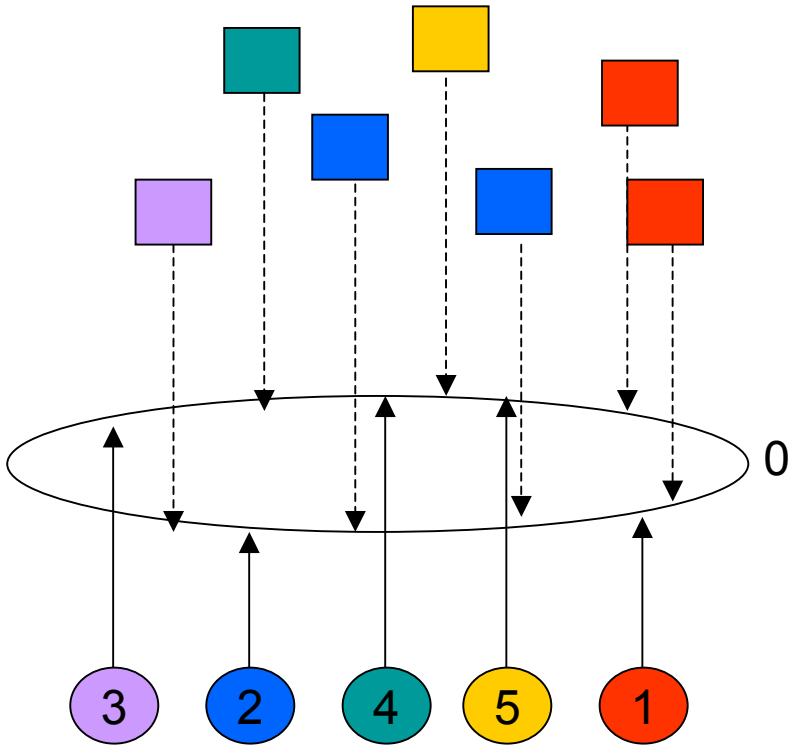
- When a new node joins: **O(M/2N)**
  - Bootstrap
    - DNS->bootstrap node->CAN nodes
  - Find a zone
    - random point->JOIN->zone splitting->pairs transfer
  - Join the routing
    - Neighbor adjustment->UPDATE
  - Involved nodes: **O(2d)**

CAN Maintenance

- When a node leaves: **O(M/N)**
  - Either neighboring node or node with least-size zone takes over the left zone.
- When failures occur:
  - Periodic UPDATE messages
  - Takeover algorithm
    - TAKEOVER message
    - Takeover timer
- Update: **O(H)**

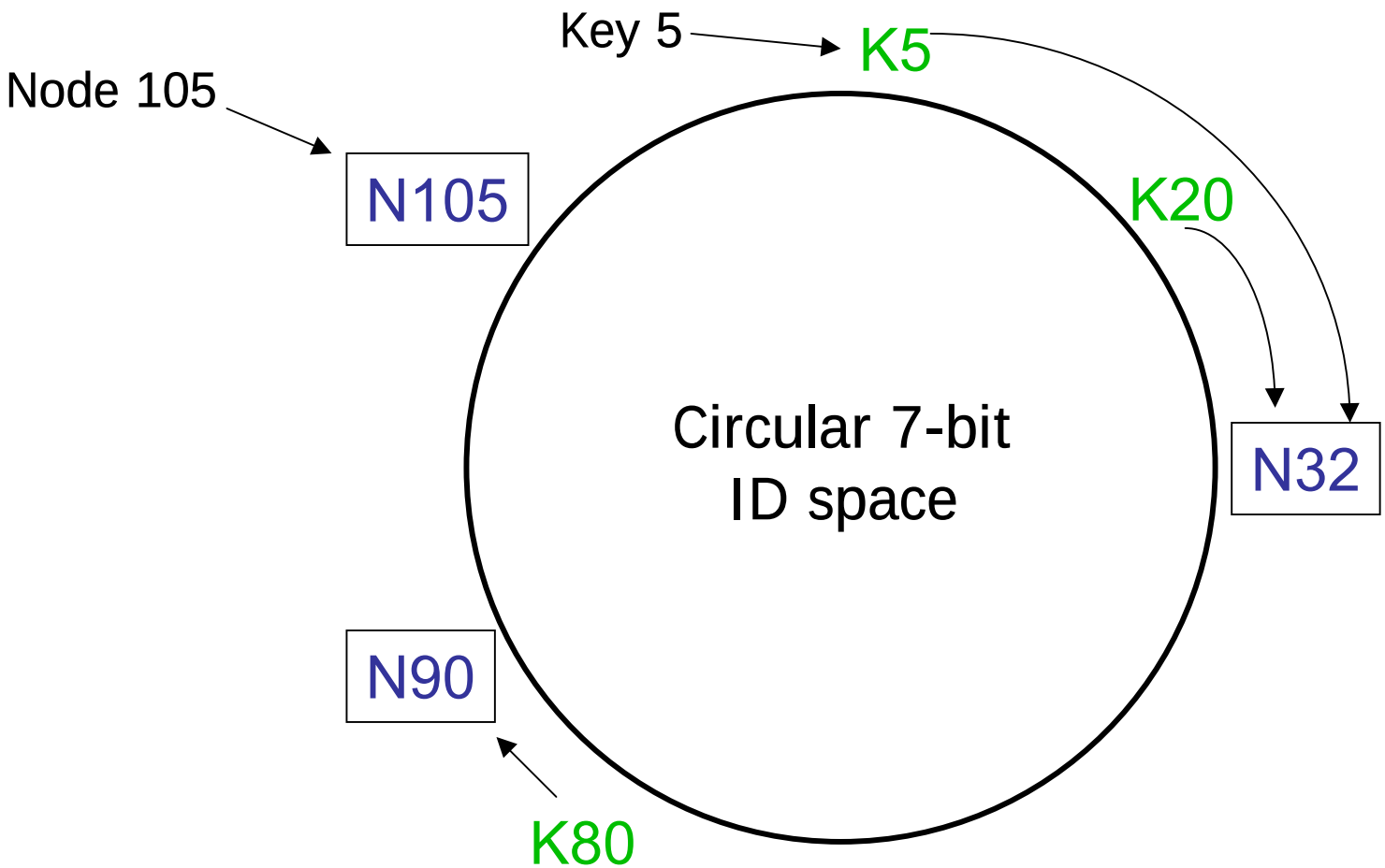
CHORD

- Implicit binary tree
- Hashing (data and node)
  - “title”/addr.->m-bit string
- Equality
- Robustness
  - Successor-list



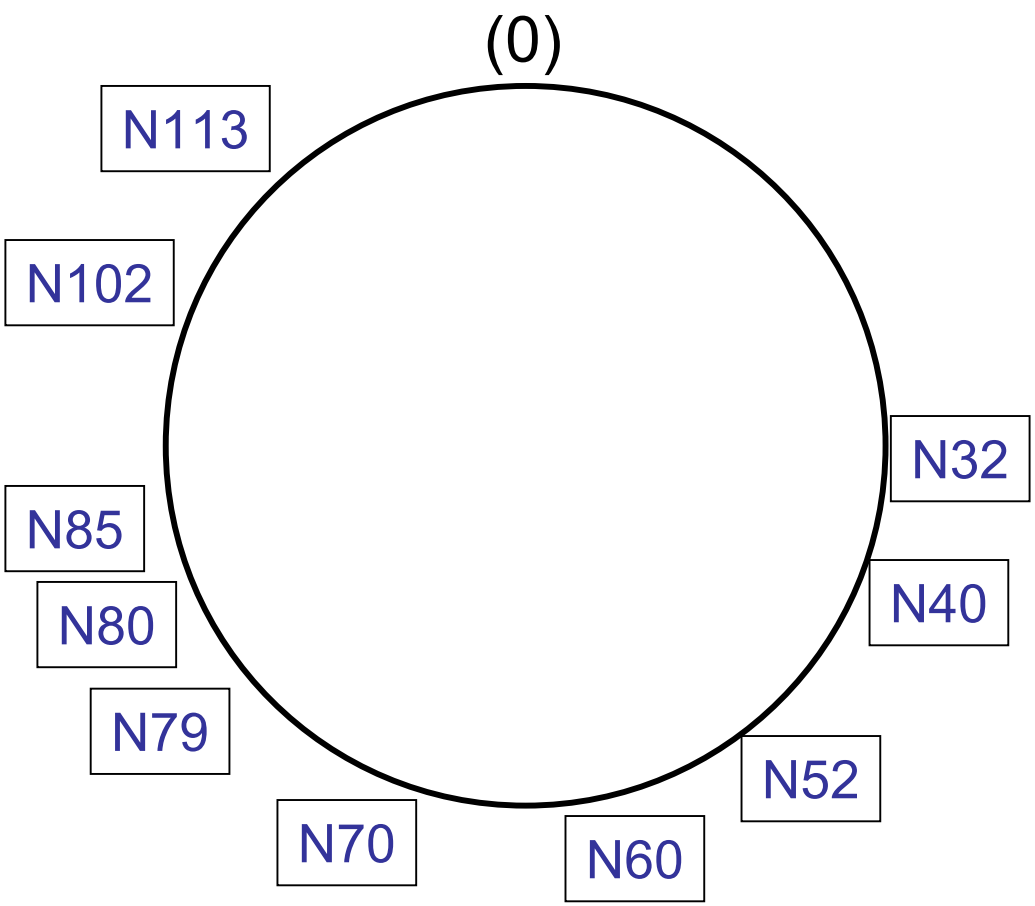
- Each peer with hashed identifier p is responsible for all hashed keys k such that k in] predecessor(p), p]

Consistent hashing [Karger 97]



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

Chord Finger Table



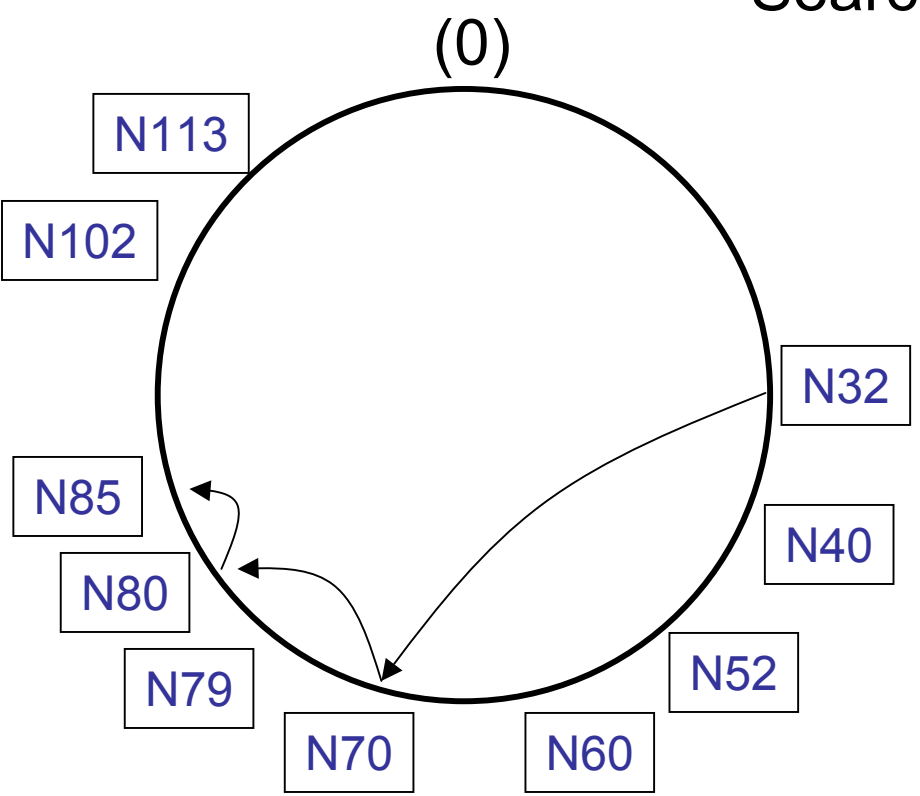
N32's Finger Table

33..33	N40
34..35	N40
36..39	N40
40..47	N40
48..63	N52
64..95	N70
96..31	N102

Node n's i-th entry: first node $\geq n + 2^{i-1}$

Chord Lookup

Search: O(logN)



N32's Finger Table

33..33	N40
34..35	N40
36..39	N40
40..47	N40
48..63	N52
64..95	N70
96..31	N102

N70's Finger Table

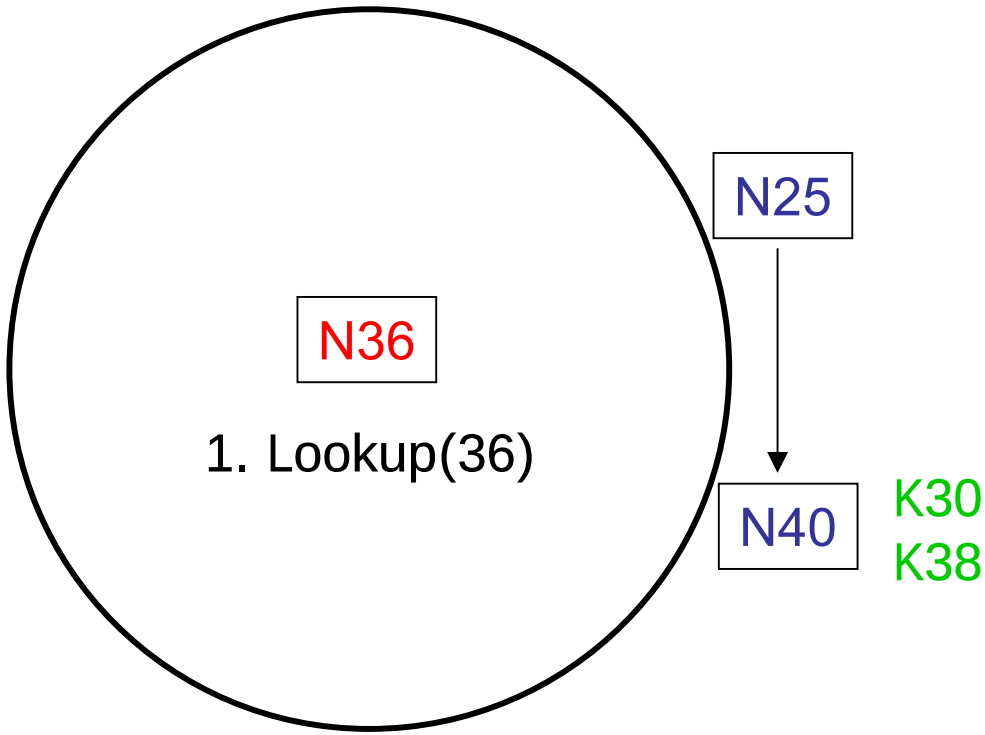
71..71	N79
72..73	N79
74..77	N79
78..85	N80
86..101	N102
102..5	N102
6..69	N32

N80's Finger Table

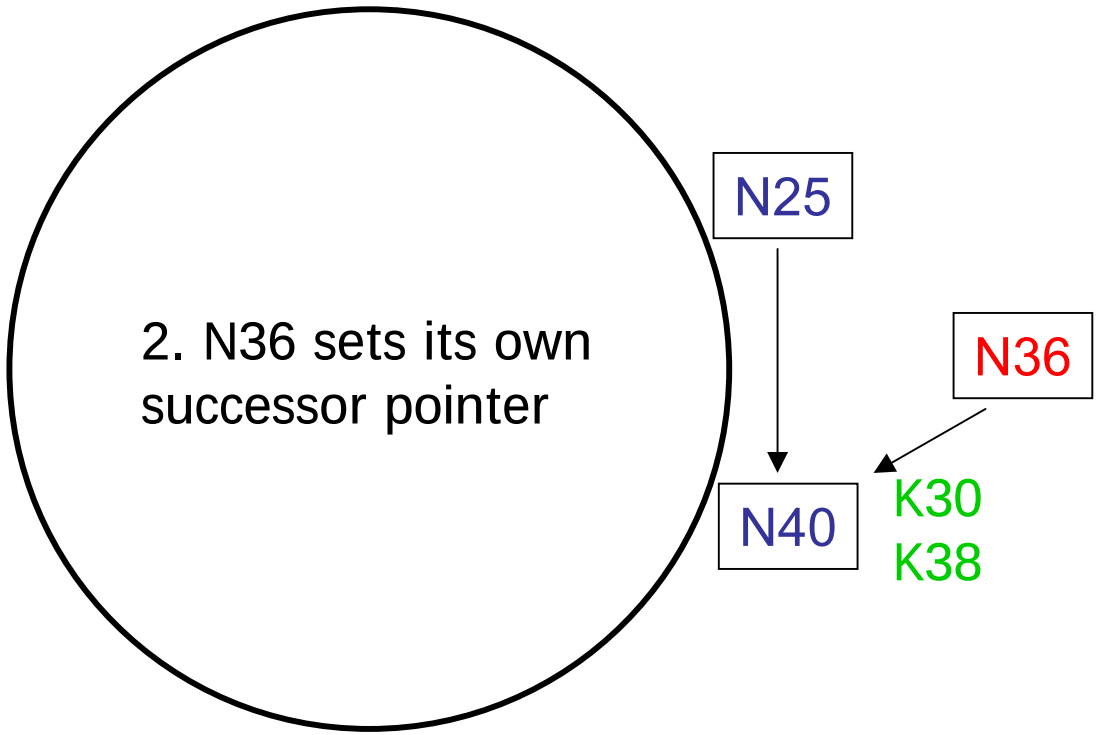
81..81	N85
82..83	N85
84..87	N85
88..95	N102
96..111	N102
112..15	N113
16..79	N32

Node 32, lookup(82): 32 → 70 → 80 → 85.

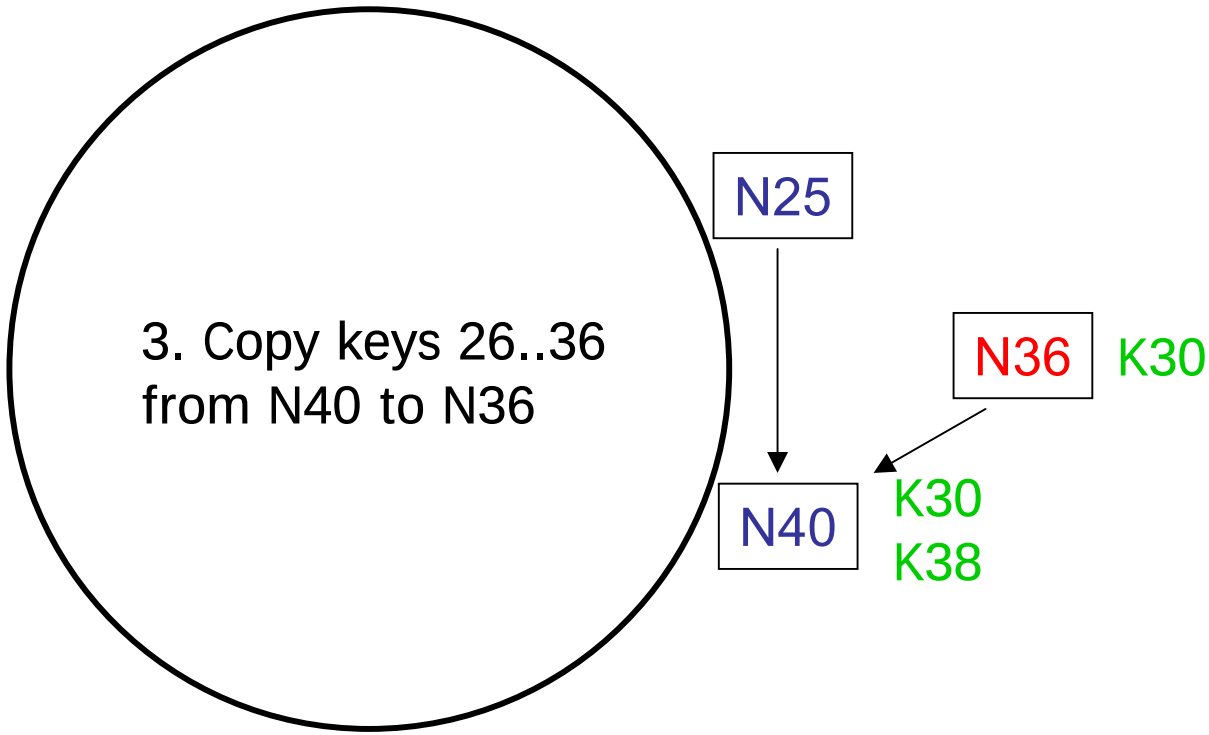
Joining: linked list insert



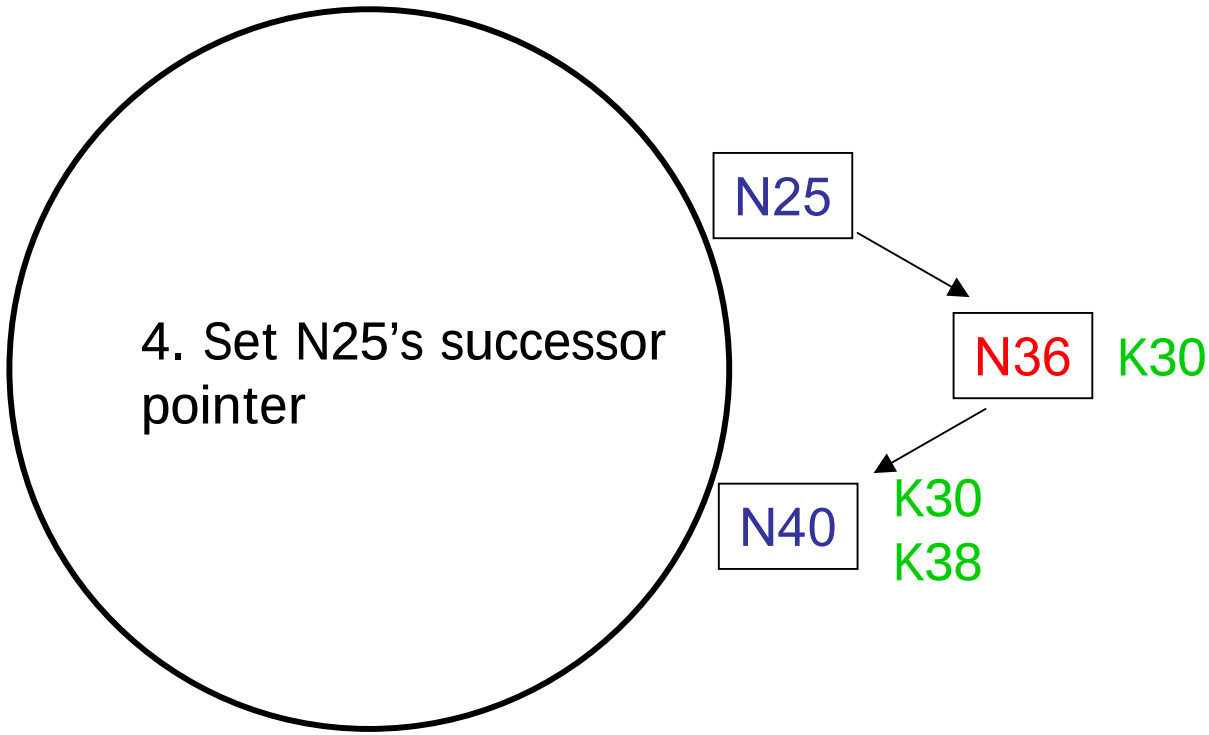
Join (2)



Join (3)



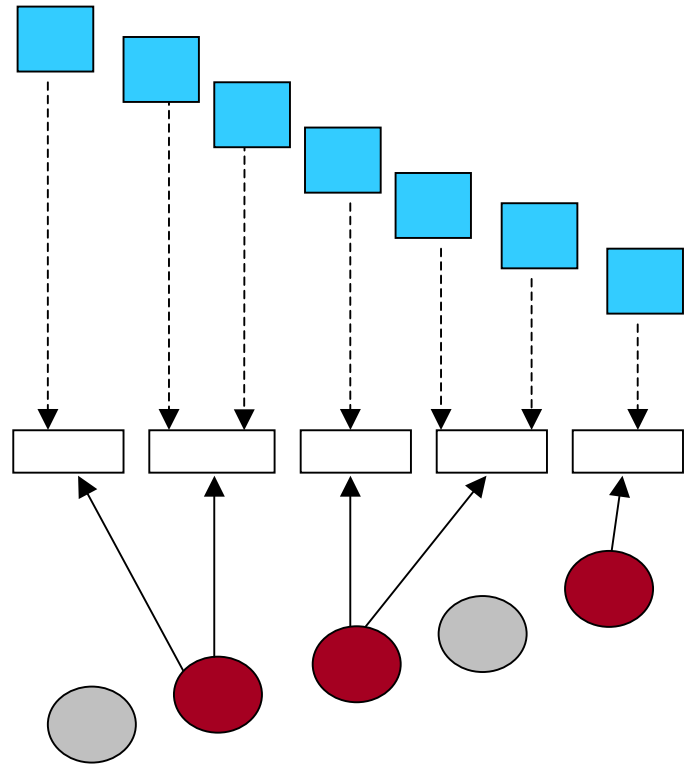
Join (4)



Update finger pointers in the background
Correct successors produce correct lookups
Message: **O(M/2N)**, involved node: **O(m)**, Update: **O(1)**

Tornado

- Binary search tree
- Hashing (data and node)
  - “title”/addr.-> m-bit ID
- Equality
- Robustness
  - Replicas by limited vectors



- Tornado considers node heterogeneity to dynamically allocate virtual homes

Performance of Tornado

- Search (same as CHORD):
  - Li=(x+R/2^i) mod R, R=2^m
  - **O(logN)** at virtual home level
- Update: **O(replicas)**
- Node join:
  - Involved node: **O(m)**
  - Message: **O(1)**
- Node leave: **>O(M/N)**

Comparison

	Routing	Search	Hashing	Search Performance	Replication
Napster	No	String comparison	No?	O(N)	Yes?
Gnutella	Breadth-first-search	String comparison	No	O(N) $2\sum^{TTL} C*(C-1)i$	Yes?
P-Grid	Binary prefix tree	Prefix	Yes, data	O(logN)	Yes
CAN	Multi-dimension grid	Equality	Yes, data	O(dN^(1/d))	Yes? Multi-hash
CHORD	Implicit binary tree	Equality	Yes, Data & node	O(logN)	No? Multi-hash
Tornado	Binary search tree	Equality	Yes, Date & node	O(logN)	Yes, Limited vectors

What is P2P Good for?

- Underutilization of
  - Information
  - Bandwidth
  - Computing resources

- P2P E-mail system
  - reduces dependency on mail servers
  - increases reliability and security