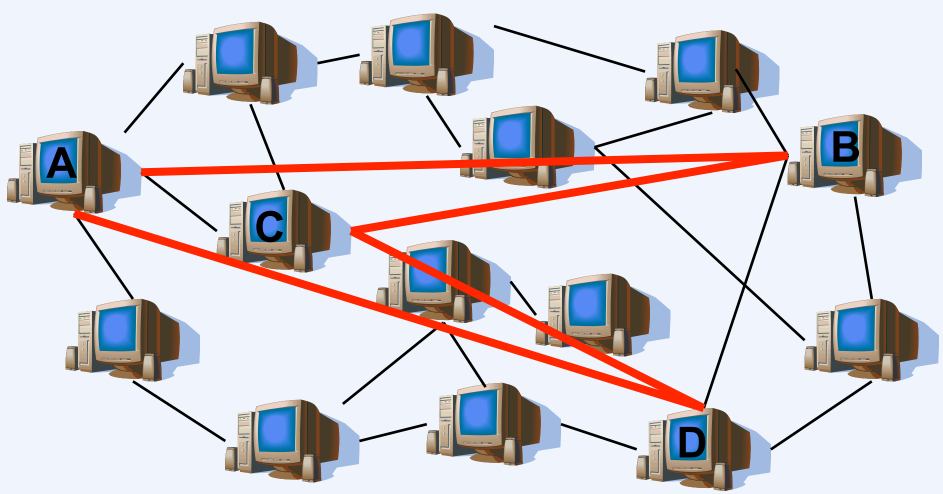
Peer to Peer II

Tapestry

Overlay Routing Concerns

- Stretch
  - routing delay penalty (RDP)
- Load balancing
  - popular object located on only one node



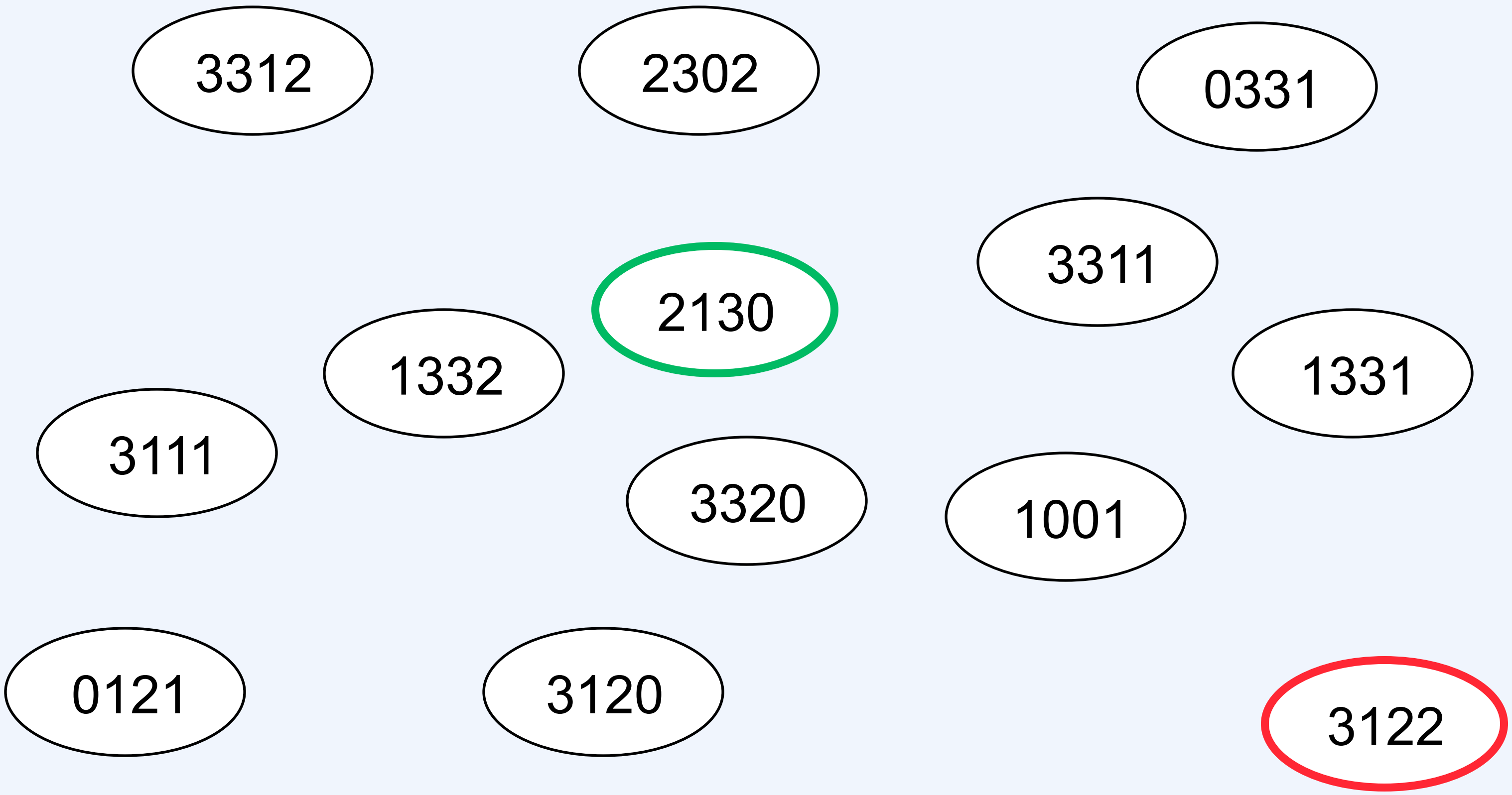
What to Do?

- Have multiple copies of objects at well distributed nodes
  - How many?
  - If you have an object at all nodes, what is the cost of read, what is the cost of insert/delete?
  - What if you have one object?
- Take communication distance into account when setting up overlay networks

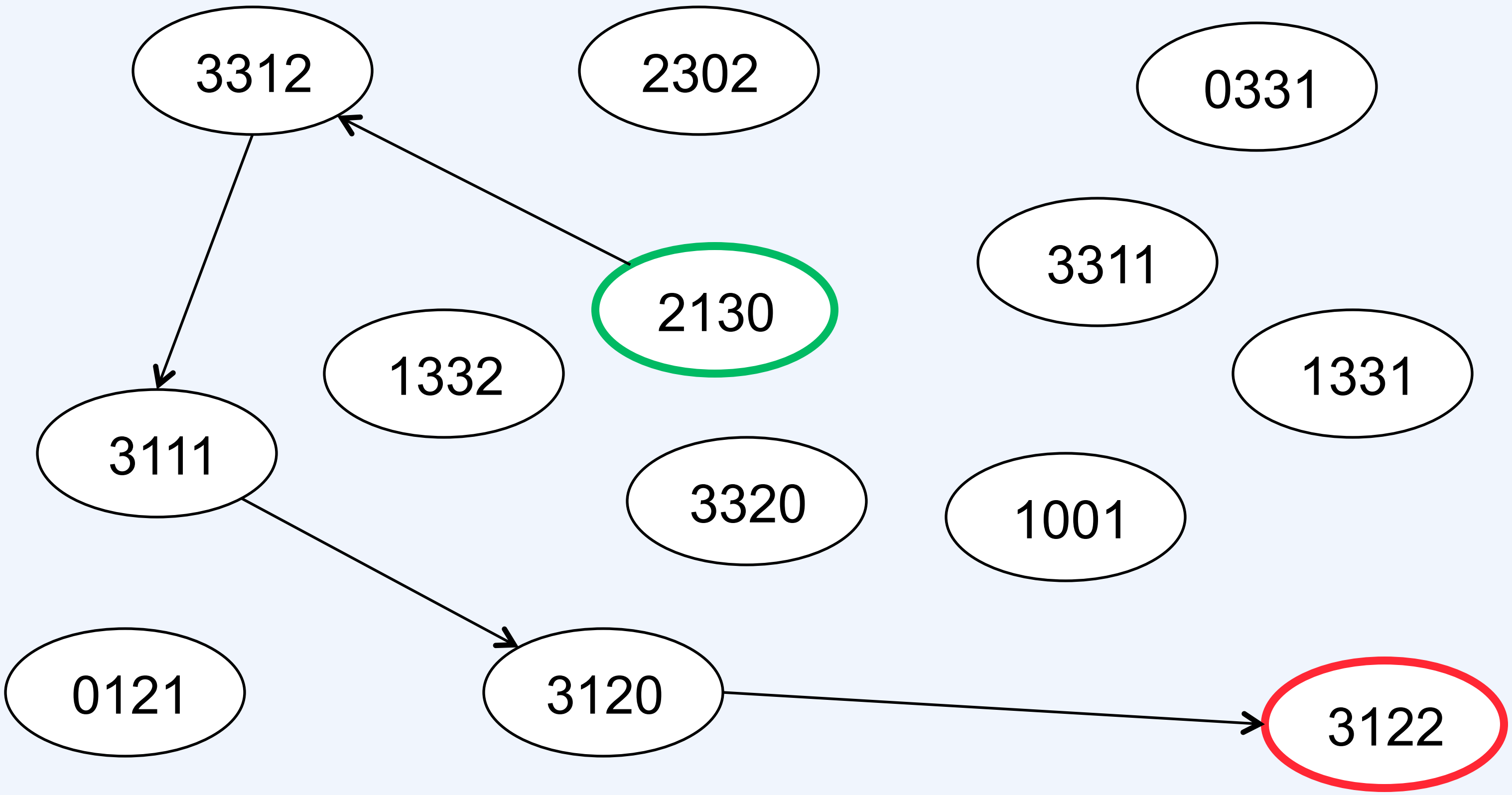
Tapestry

- Assign each block a unique m-bit ID
  - crypto hash of its contents
- Assign each computer a unique m-bit ID
- Store multiple copies of blocks each at a number of computers
- Store block addresses at computer that has closest ID
  - addresses are cached at other nodes
- Route requests for that block to that computer
  - request is redirected to nearest computer that has copy of block

How to Route?


3312
2302
0331
3311
2130
1332
1331
3111
3320
1001
0121
3120
3122

Prefix Routing



Neighbor Table for 3312

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
33xx	–	3311 128.12.236.81	3320 128.248.192.76	–
331x	–	3311 128.12.236.81	3312 128.213.97.6	–

Routing Algorithm

```
// executed at each node in route to destination
NextHop(targetHash, step) {
   nextDigit = digit(targetHash, step)
   return(table[step, nextDigit])
}

digit(num, pos) {
   return ((num/base^(d-pos))%base)
}
```

Surrogate Routing

- Store object's location list at unique computer whose hash is "closest" to the object's hash
  - unique computer known as the *root*
  - all routes to object's hash reach the root regardless of the starting point
  - the path to this root goes through various "surrogate" nodes
    - if there is a hole in the neighbor table corresponding to the "next digit", then choose the first non-empty entry in the row that's greater (mod base) than the one desired
    - the node routed to is the *surrogate*

How?

- If no next hop exists, try the next larger digit, mod *base*
  - each neighbor-table row must have at least one entry
    - why?
  - if any two neighbor-table rows (of different nodes) share the same prefix, they must agree on which entries are null
    - why?

Neighbor Tables for 3312 and 3320

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
33xx	–	3312 128.213.97.6	3320 128.248.192.76	–
331x	–	3311 128.12.236.81	3312 128.213.97.6	–

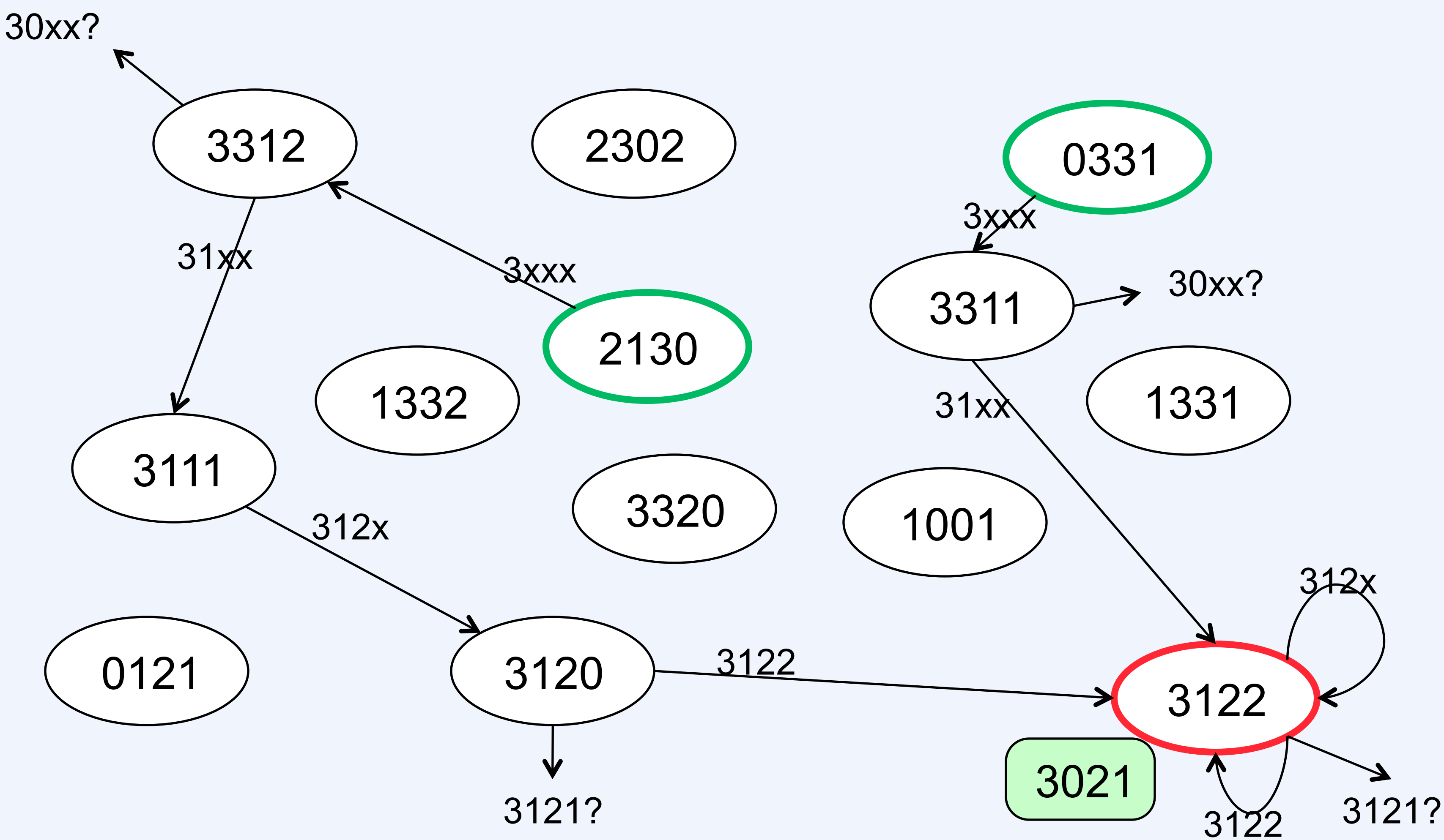
	0	1	2	3
xxxx	0121 128.148.158.13	1001 128.18.11.192	2130 128.113.225. 127	3311 128.12.236.81
3xxx	–	3120 128.162.253.247	–	3320 128.248.192.76
33xx	–	3311 128.12.236.81	3320 128.248.192.76	–
332x	3320 128.248.192.76	–	–	–

Surrogate Routing Algorithm

```
// executed at each node in route to destination
NextHop(targetHash, step) {
   nextDigit = digit(targetHash, step)
   while ((next = table[step, nextDigit]) == NULL)
         nextDigit += 1 mod base
   return next
}

digit(num, pos) {
   return ((num/base^(d-pos))%base)
}
```

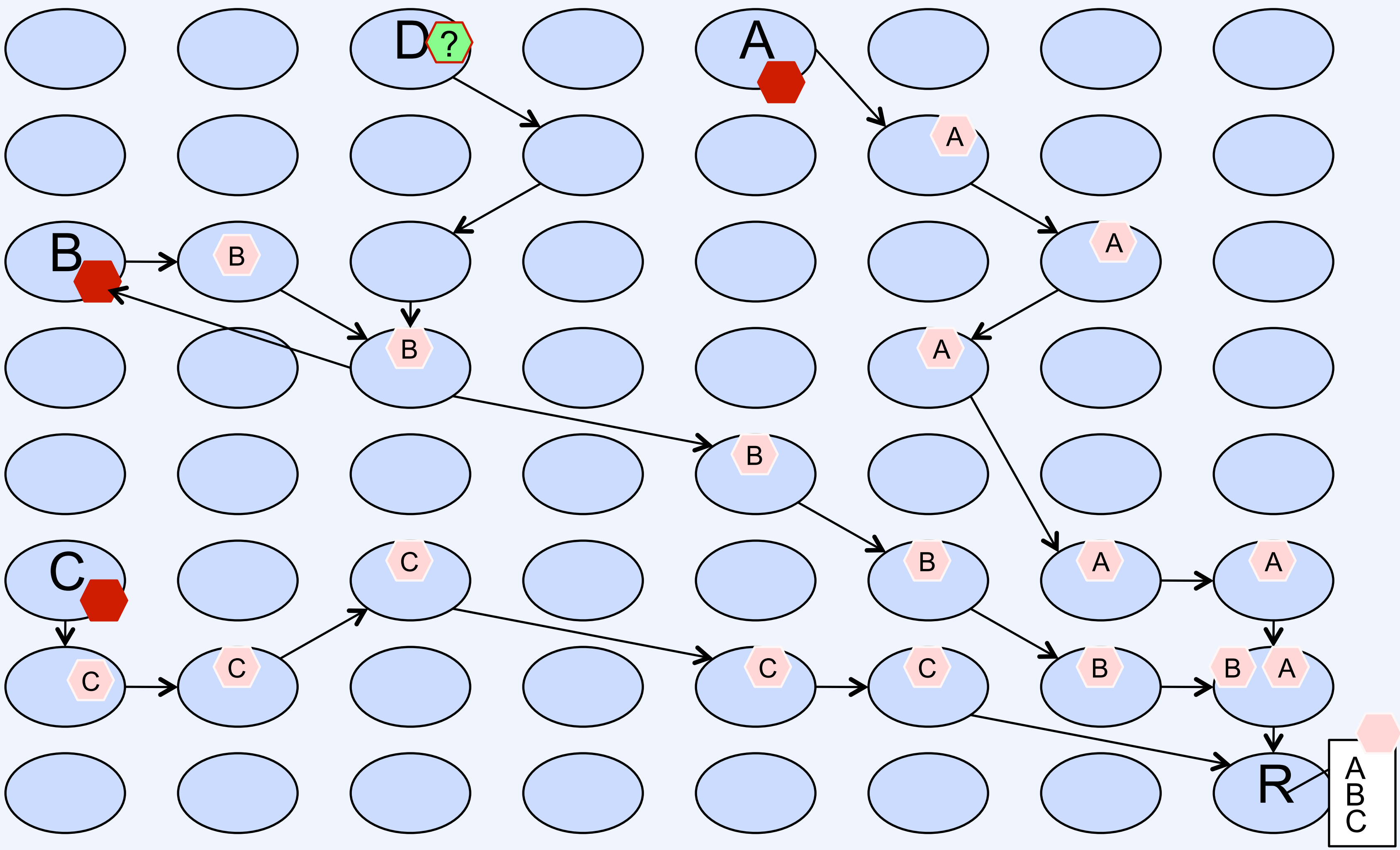
Surrogate Routing for 3021



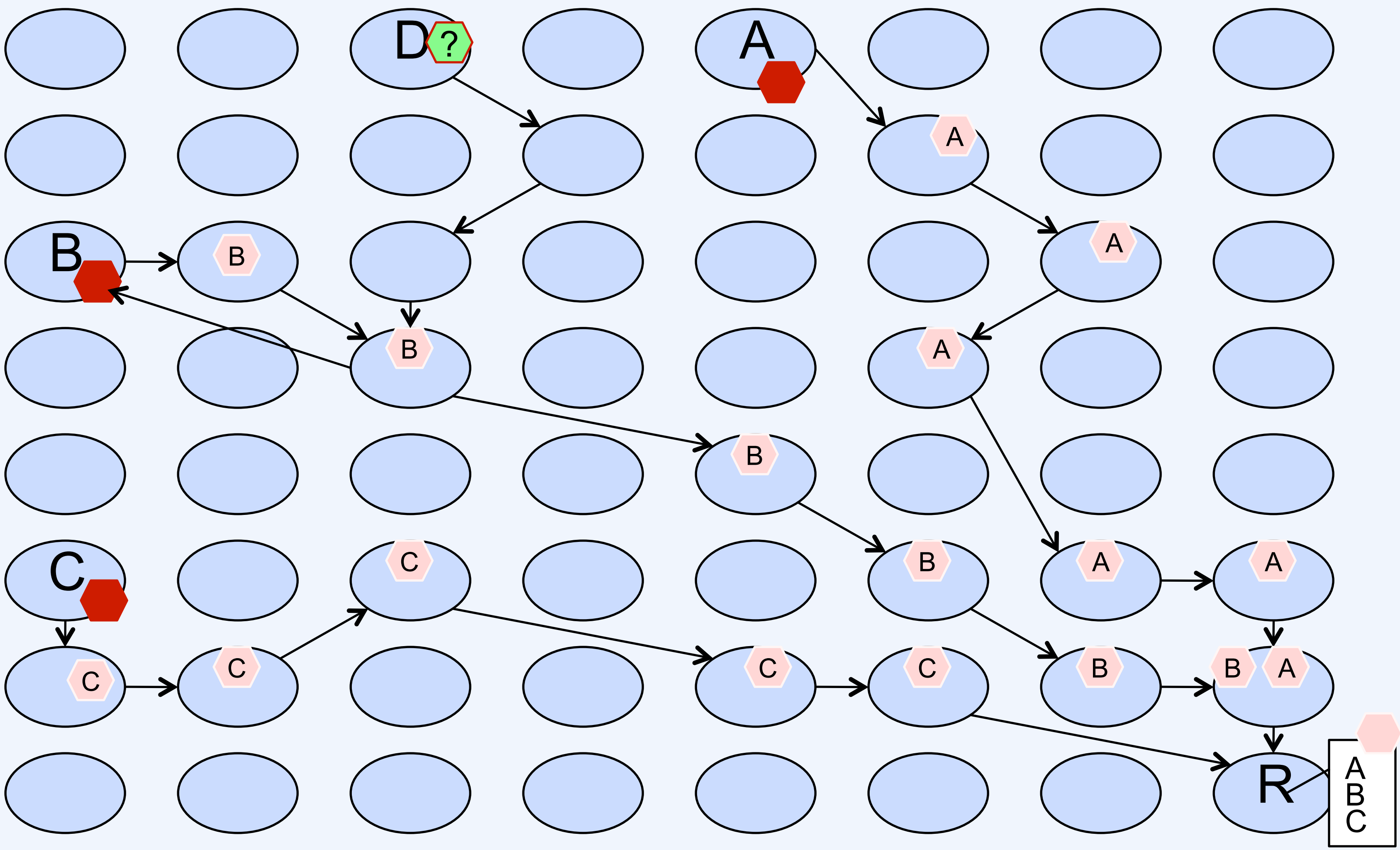
Performance and Redundancy

- For any particular neighbor-table entry, there may be a number of possible valid next hops
  - all of them work
  - choose the one that's "closest"
    - communication delay makes sense for this
  - if a next hop can't be reached
    - use one of the other possible next hops
    - store some number of them in table
      - "secondary entries"

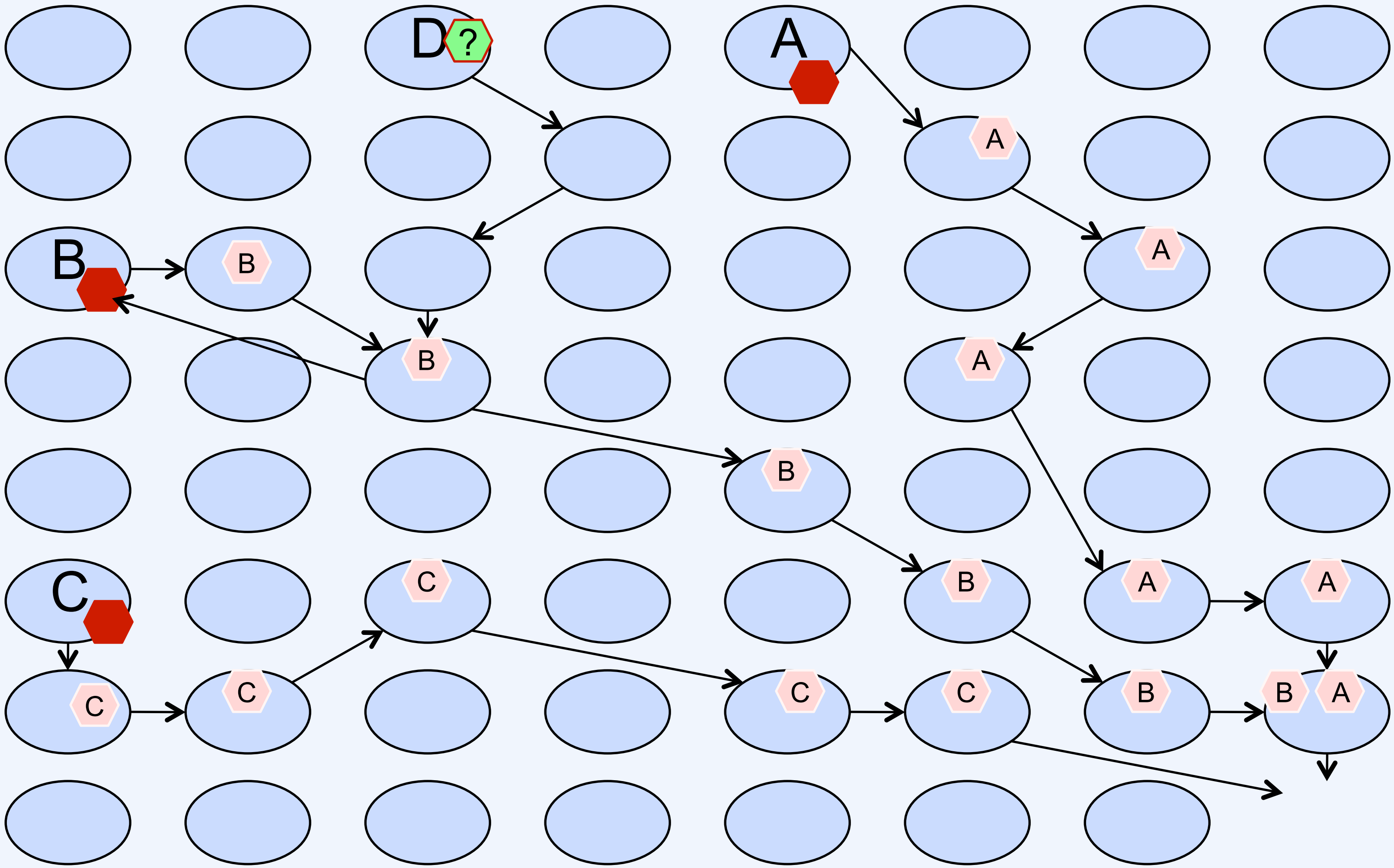
Publishing



Failure



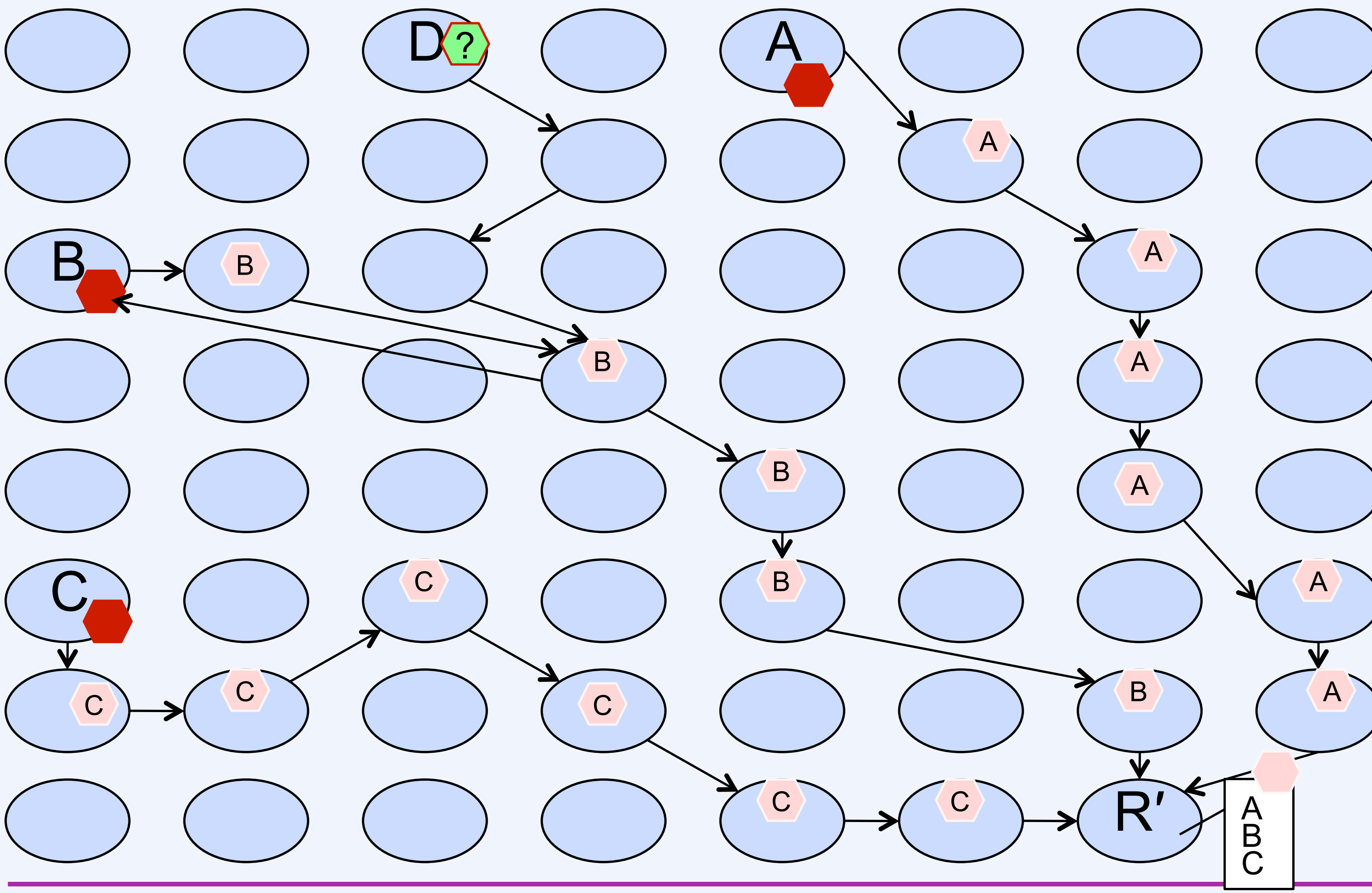
Failure



Soft State

- State information times out
  - e.g., reference to node holding an object
- Must be periodically reestablished
  - nodes must periodically republish their objects

Recovery



Redundant Redundancy

- When "root nodes" of objects disappear, it may take some time before re-publication is effective
  - solution: extra root nodes
  - how?
    - "salt" the hashes
      - append a small integer before hashing
      - multiple hashes for one object: multiple roots for the object

Adding a Node

- Steps for adding node n
  1) find existing node G
  2) search for n's hash starting at G
  3) at each step i, fill in row i of n's table with row i of table of node being visited
  4) stop when empty table entry is encountered
  5) fill in remainder of table with self entries
  6) notify other nodes to update their tables

Initializing a Neighbor Table Row (1)

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	–	–	–
30xx	–	–	–	–
300x	–	–	–	–

n's (3001's) Table

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
33xx	–	3311 128.12.236.81	3320 128.248.192.76	–
331x	–	3311 128.12.236.81	3312 128.213.97.6	–

G's (3312's) Table

Initializing a Neighbor Table Row (2)

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	3001 128.250.19.172	3111 128.172.53.237	–	3312 128.213.97.6
30xx	–	–	–	–
300x	–	–	–	–

n’s (3001’s) Table

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
33xx	–	3311 128.12.236.81	3320 128.248.192.76	–
331x	–	3311 128.12.236.81	3312 128.213.97.6	–

step 2 (3312’s) Table

Initializing a Neighbor Table Row (3)

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	3001 128.250.19.172	3111 128.172.53.237	–	3312 128.213.97.6
30xx	3001 128.250.19.172	–	–	–
300x	–	–	–	–

n's (3001's) Table

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
31xx	–	3111 128.172.53.237	3120 128.162.253.247	–
311x	–	3111 128.172.53.237	–	–

step 3 (3111's) Table

Initializing a Neighbor Table Row (4)

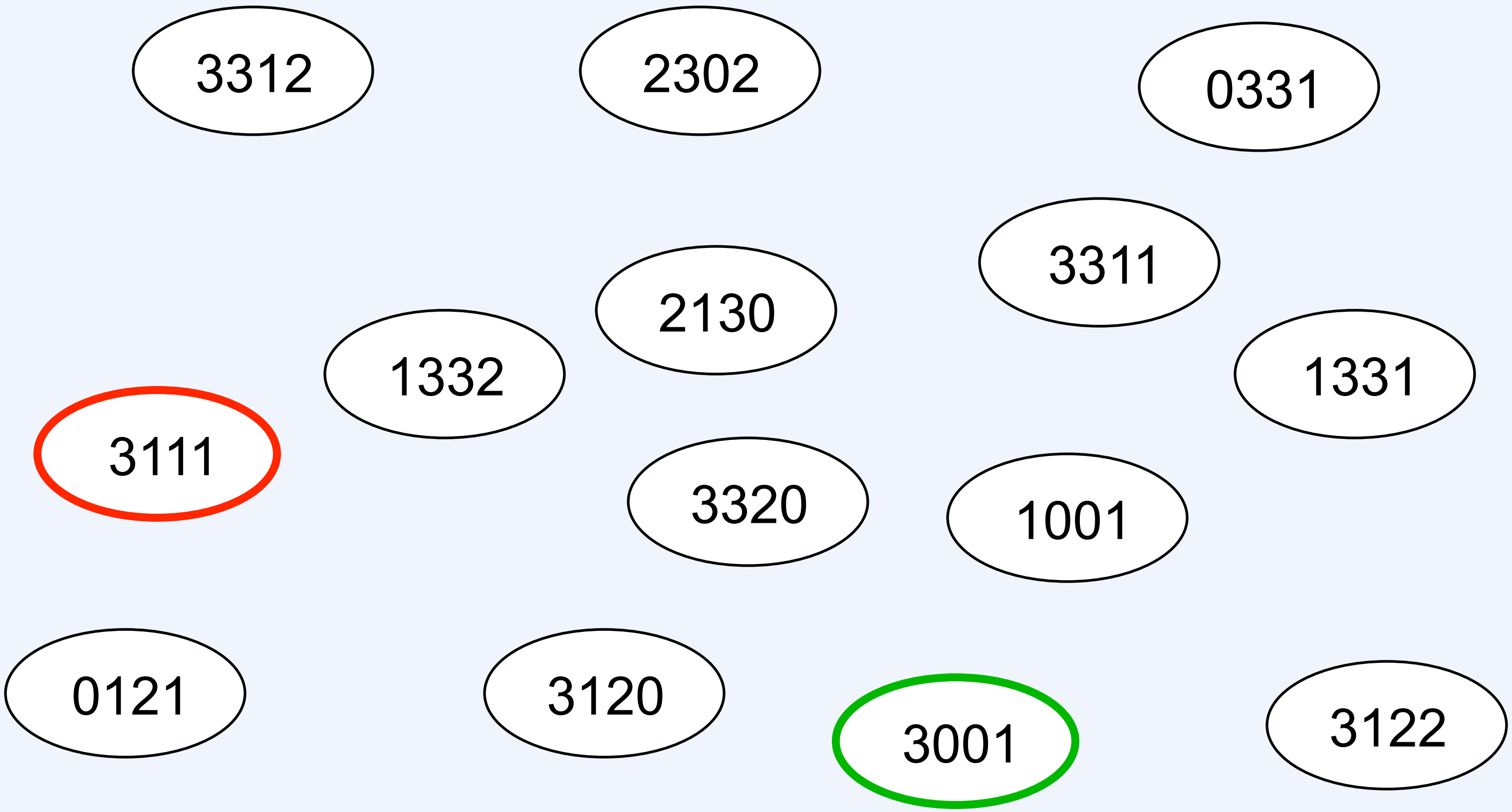
	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	3001 128.250.19.172	3111 128.172.53.237	–	3312 128.213.97.6
30xx	3001 128.250.19.172	–	–	–
300x	–	3001 128.250.19.172	–	–

n's (3001's) Table

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
31xx	–	3111 128.172.53.237	3120 128.162.253.247	–
311x	–	3111 128.172.53.237	–	–

step 4 (3111's) Table

Updated View



Notifying Others

- Need to insert n in all neighbor-table entries that are empty where n should go
- Before n was added, any search for n ended up at its root
- Proceed backwards from root
  - – each neighbor table includes back pointers to all nodes that route to it
  - – *flooding* procedure:
    - - on receipt of notification
      - if routing table contains hole where n goes
        - –insert n
        - –notify neighbors via back pointers

A Problem

	0	1	2	3
xxxx	0121 128.148.158.13	1001 128.18.11.192	2130 128.113.225. 127	3311 128.12.236.81
3xxx	–	3120 128.162.253.247	–	3320 128.248.192.76
33xx	–	3311 128.12.236.81	3320 128.248.192.76	–
332x	3320 128.248.192.76	–	–	–

- **Node 3322 is added**
- **Object 3321's surrogate was 3320**
  - **now it's 3322**
  - **what about all the location info that was stored assuming 3320?**

Doing a Better Job …

- Find all nodes for which new node fills holes in neighbor tables
  - propagate new node on spanning tree of just the relevant nodes
- Handle “re-rooted” objects
  - efficiently …
- Build new neighbor tables
  - optimizing for closeness

Observation

- Let α be longest common prefix of new node and its root
- All nodes whose neighbor tables contain holes to be filled by new node have this prefix

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	3001 128.250.19.172	3111 128.172.53.237	–	3312 128.213.97.6
30xx	3001 128.250.19.172	–	–	–
300x	–	3001 128.250.19.172	–	–

n's (3001's) Table

	0	1	2	3
xxxx	0331 128.148.128.173	1332 128.138.117.92	2302 128.118.165. 27	3312 128.213.97.6
3xxx	–	3111 128.172.53.237	–	3312 128.213.97.6
31xx	–	3111 128.172.53.237	3120 128.162.253.247	–
311x	–	3111 128.172.53.237	–	–

root's (3111's) Table

Application

- Send new node's info to all nodes with prefix α
  - hash and IP address
- How?
  - via spanning tree that reaches all such nodes
  - use "acknowledged multicast"

Acknowledged Multicast

```
n.acknowledgedMulticast(α, function) {
  if (notOnlyNodeWithPrefix(α))
    for i = 0 to b-1
      neighbor = neighborWithPrefix(α·i)
      if neighbor exists
        S = neighbor.acknowledgedMulticast(α·i, function)
  else
    apply function
  wait S
  SendAcknowledgement()
}
```

Using Acknowledged Multicast

- To fill holes in neighbor tables with new node
  - supply function that does this
- To “re-root” object references (move them to the new node)
  - supply function that does this
- To get list of all nodes with given prefix
  - supply function that returns node IDs

Races

- Suppose a search for object x occurs while new node N is being added
  - N becomes the new root for x
- Potential problems:
  1) search arrives at N before object references are transferred to N
  2) search arrives at old root after object references are transferred to N

Problem 1 Solution

- **Mark N as “being inserted”**
  - **on “object not found”**
    - **forward request to original root**

Problem 2 Solution

- Include path with search request
  - on “object not found”
    - check neighbor table to see if path would still be taken
      - i.e., has hole subsequently been filled (by new root)?
      - if so, reroute to new root

Optimizing the Neighbor Table

- Want primary node for each neighbor-table entry to be the one that's closest
  - secondary nodes should be the next closest
- How can this be constructed for a new node?

Sketch

- Use acknowledged multicast to find all nodes with prefix α (longest prefix in common with root)
- From this set of nodes, construct table row $|\alpha|$
  - determine which are closest to the new node
- From this set of nodes, find all nodes with prefix one shorter than α
  - construct table row $|\alpha|$-1, using closest nodes
- etc.