Distributed Systems

ID2201

distributed hash tables

Johan Montelius

Distributed hash tables

- Large scale data bases
  - hundreds of servers
- High churn rate
  - servers will come and go
- Benefits
  - fault tolerant
  - high performance
  - self administrating

Routing overlay

- The problem of finding a node, object or resource in a network:
  - nodes can leave and join
  - nodes might fail
- Each object is described by a globally unique identifier (GUID).

Description, Identifier and Objects

Description

identify

Is the description unique?

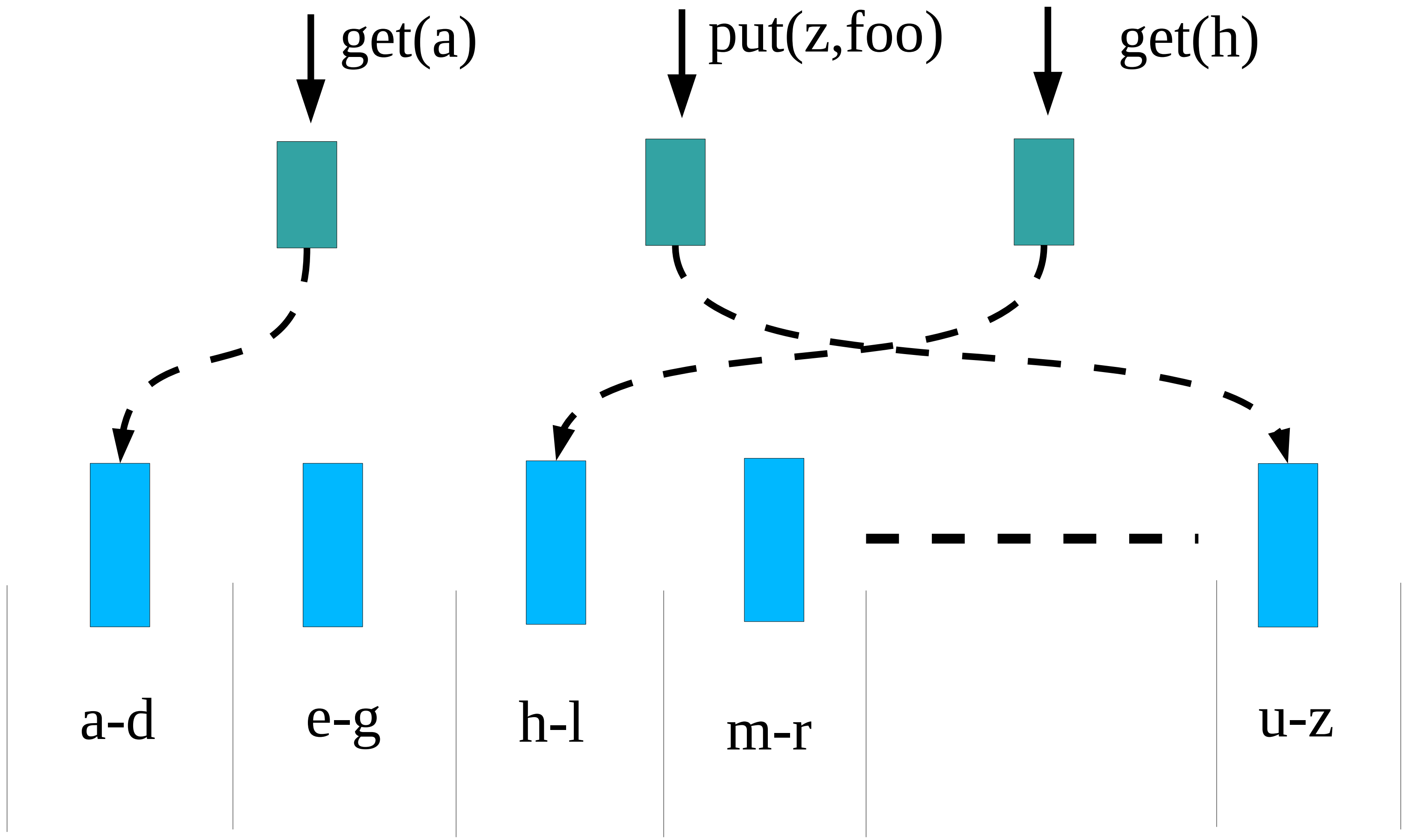
Identifier

How do we find a unique identifier?

routing

Object

Distributed Tables



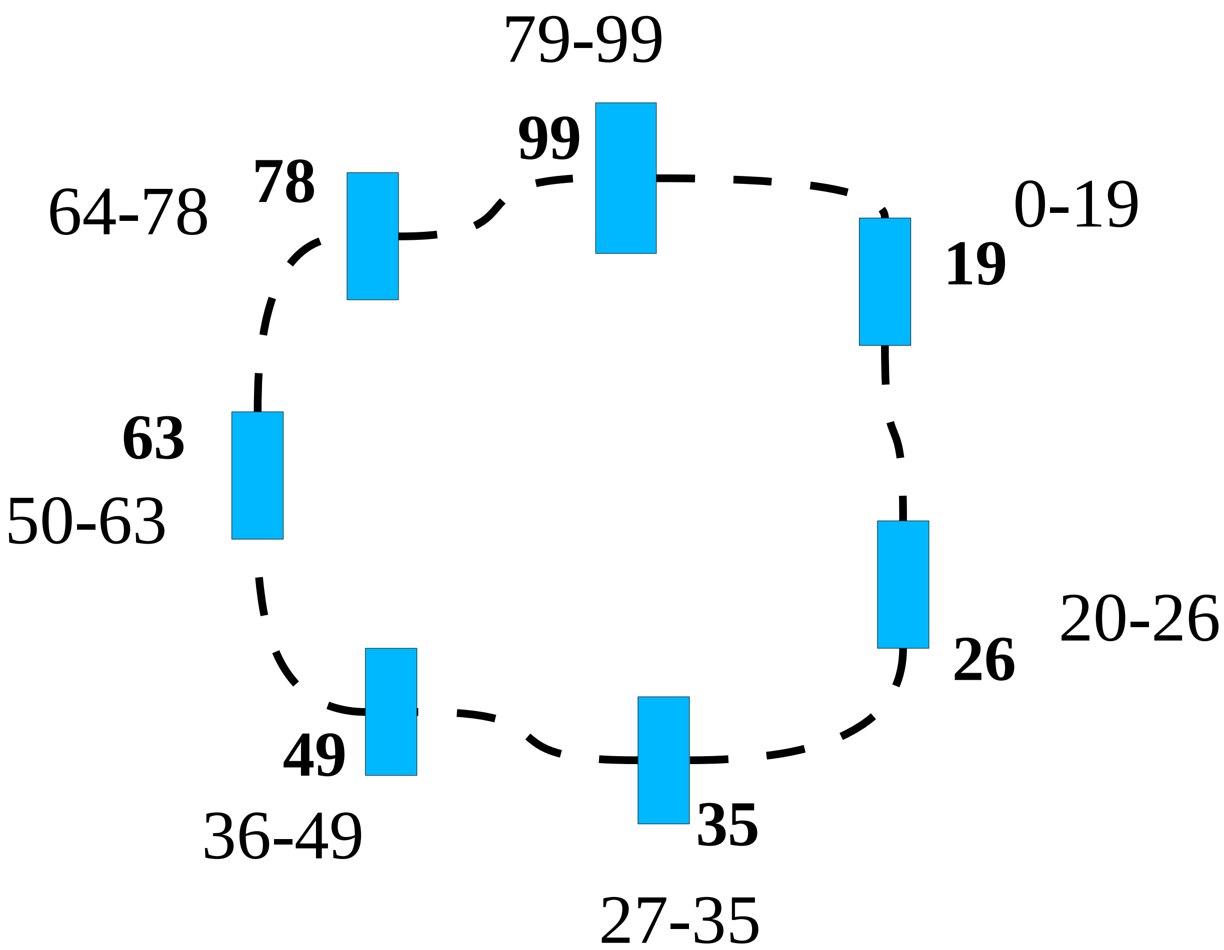
Problems

- How do we divide the table.
- What should we do when a node dies?
  - how is lookup changed
- What if we add more servers?
  - increase performance
- How does load balancing work?
  - hot spot

Distributed Hash Tables

- Use a hash value as key/identifier
  - uniform distribution
- Each node chooses a random hash value as its identifier.
- Nodes form a ring and can forward request in the ring.

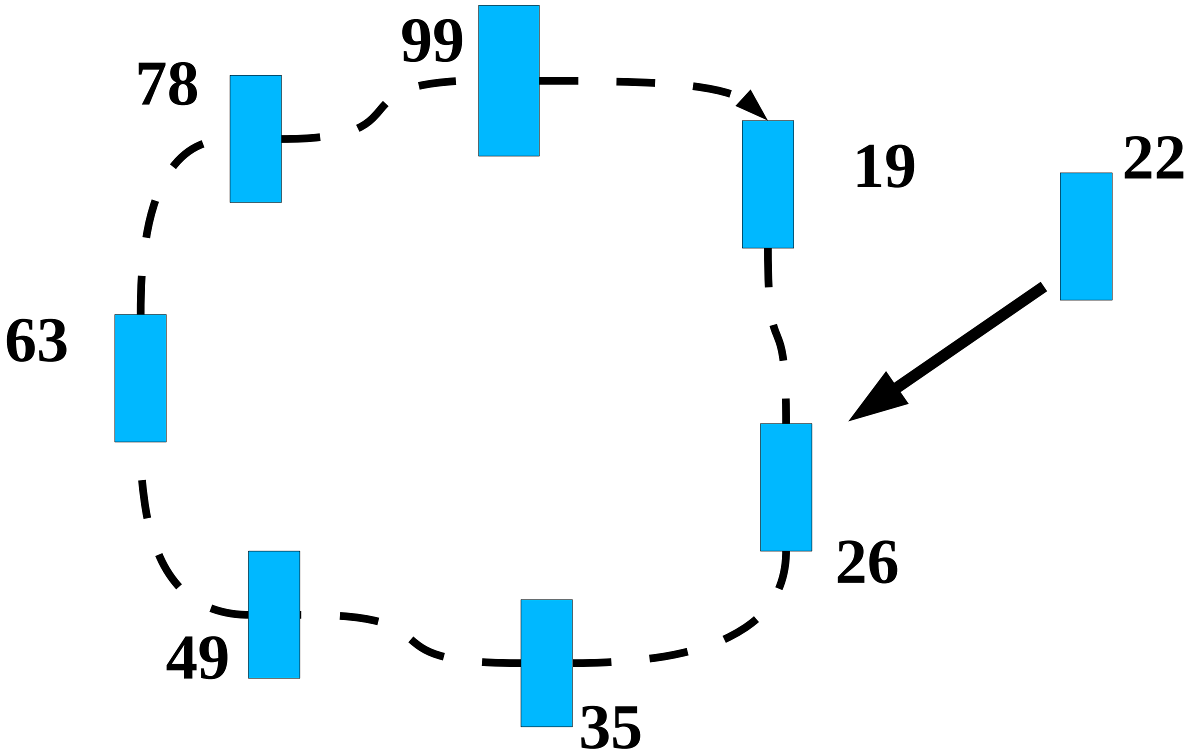
Responsibility



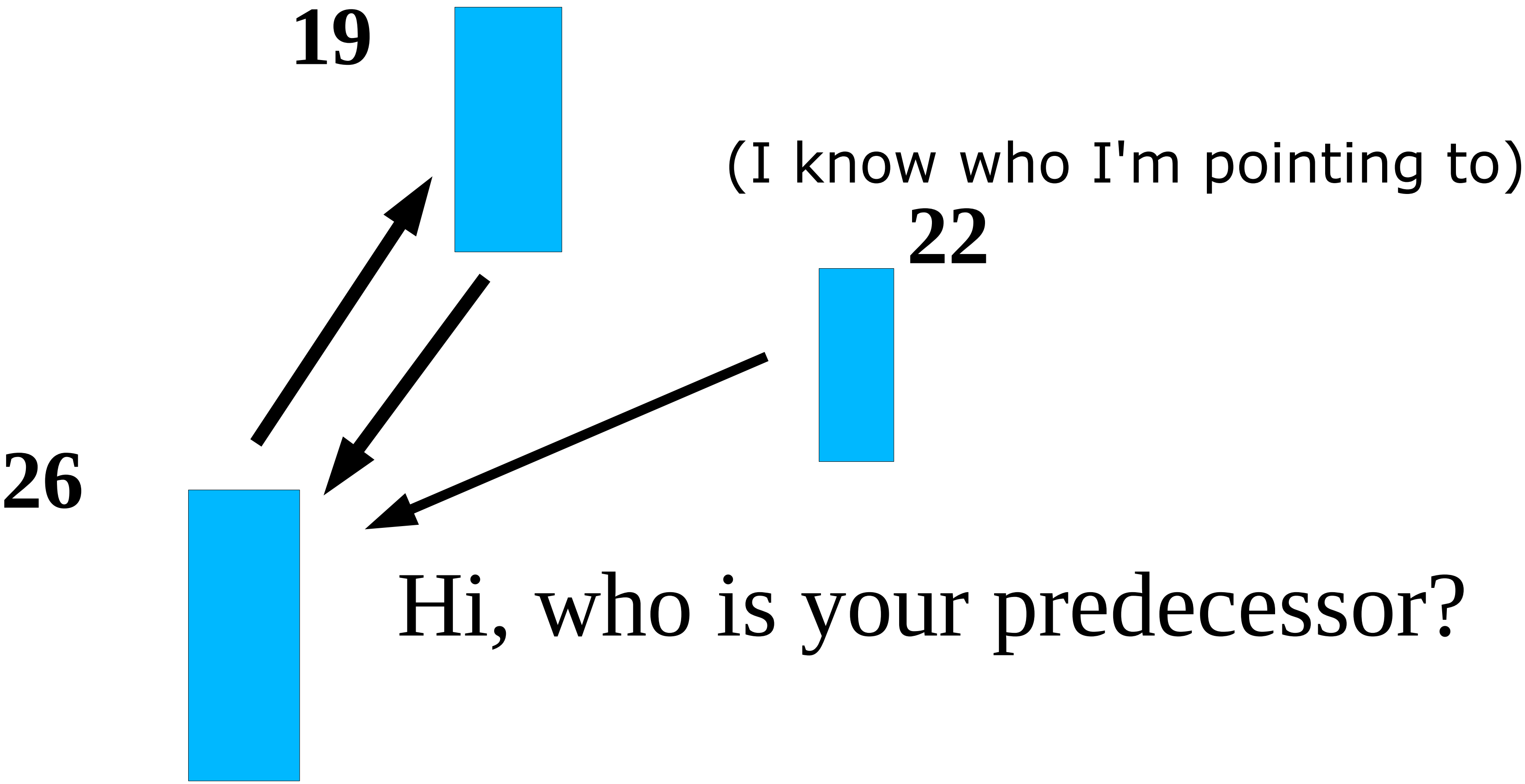
Will this work?

- If nodes choose an identifier at random, can it not become uneven distribution?

Adding a node



Stabilization



Stabilization

- Hi, who is your predecessor?
  - This is my predecessor, it has id 19.
  - I have id 22, why not point to me?
- Hi, who is your predecessor?
  - This is my predecessor, it has id 24.
  - Hmm, that should be my successor.
- Let's play a game!

Adding a store

- If we have a ring it is simple to add a store:
  - add key-value pair
  - lookup value given key
- Need to take over part of store when entering the ring.

Does it pay off

- Set up a ring with one one node.
- Have several client doing add and lookup operations.
- Increase the number of nodes in the ring.
- Does it pay off?

Handling failures

- We should survive one failure and maintain the ring (forget a bout the store for a while).
- How do we do?

0.99 uptime probability

- Assume that a the risk of a node crashing during a stabilization period is 1/10.
- How many successors should we keep track of?

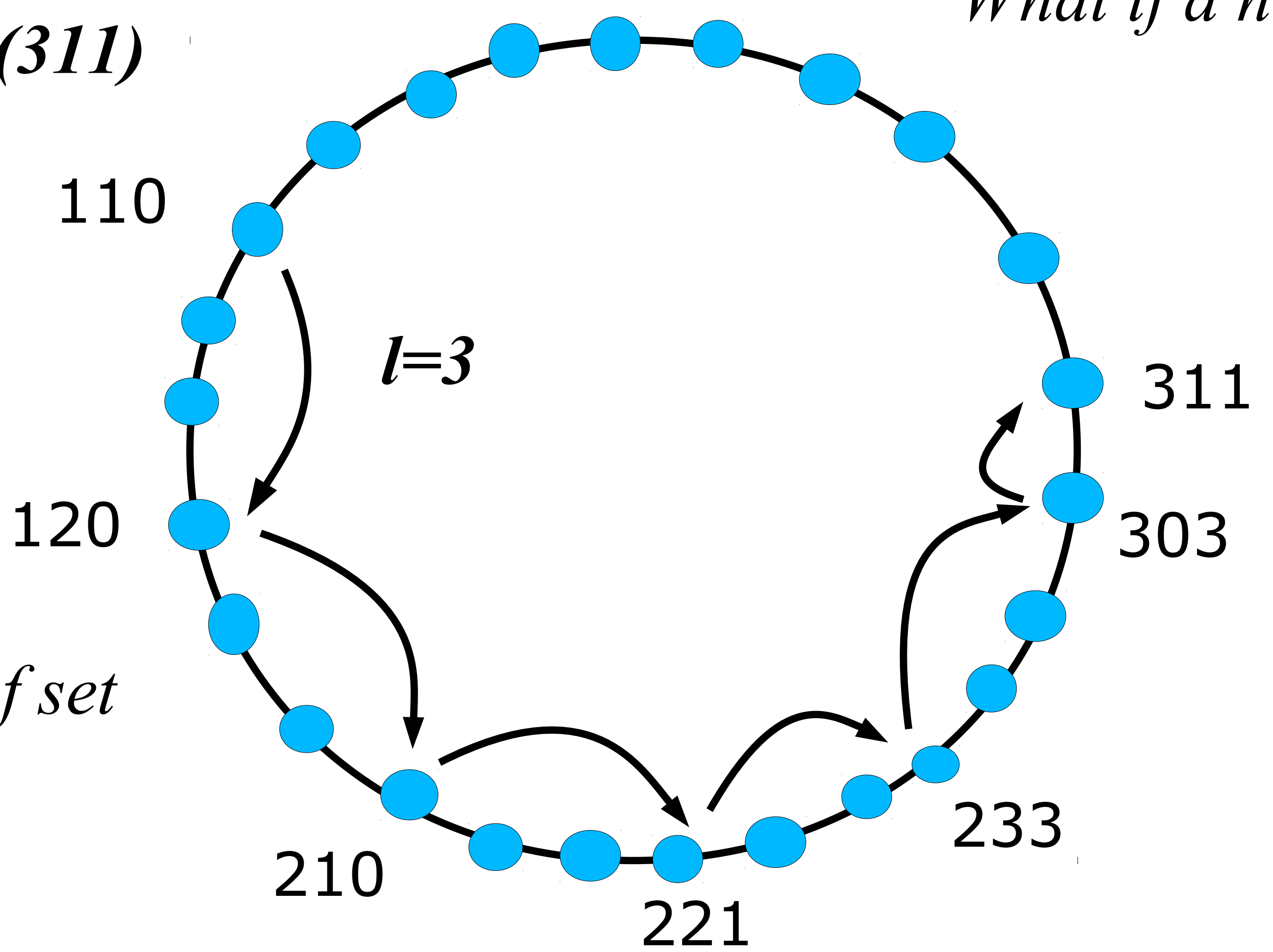
Distributed Hash Tables (Pastry)

- Compute a hash of the value to store.
- The computed value determines which node that is responsible for the data.
- API:
  - put(guid, data): send the data to the responsible node
  - remove(guid): fin the responsible node and remove the data
  - get(guid): find the node and locate the data

circular routing (simple case)

What if a node dies?

get(311)

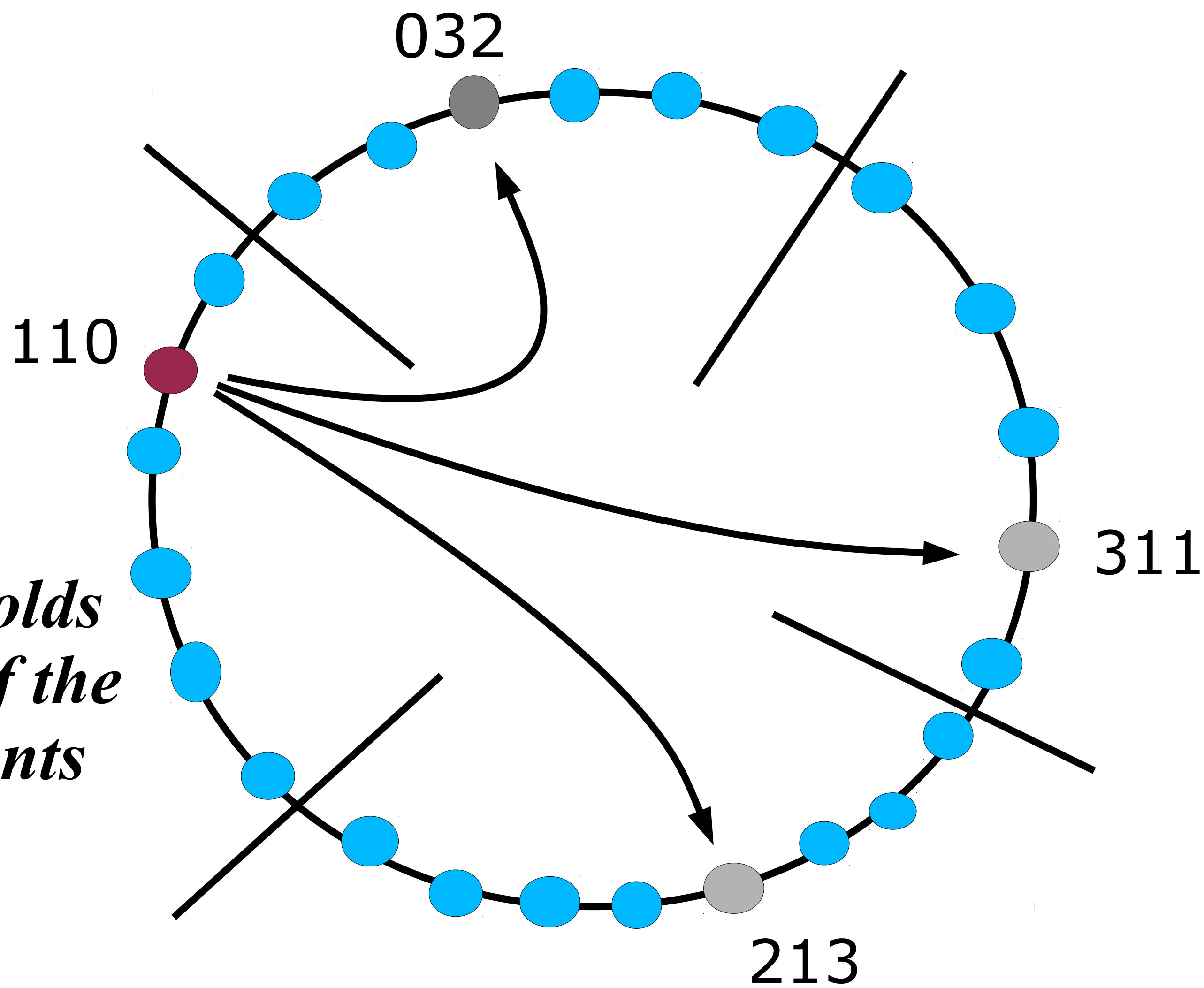


nodes holds leaf set of neighbors +/- l key value

Improvement

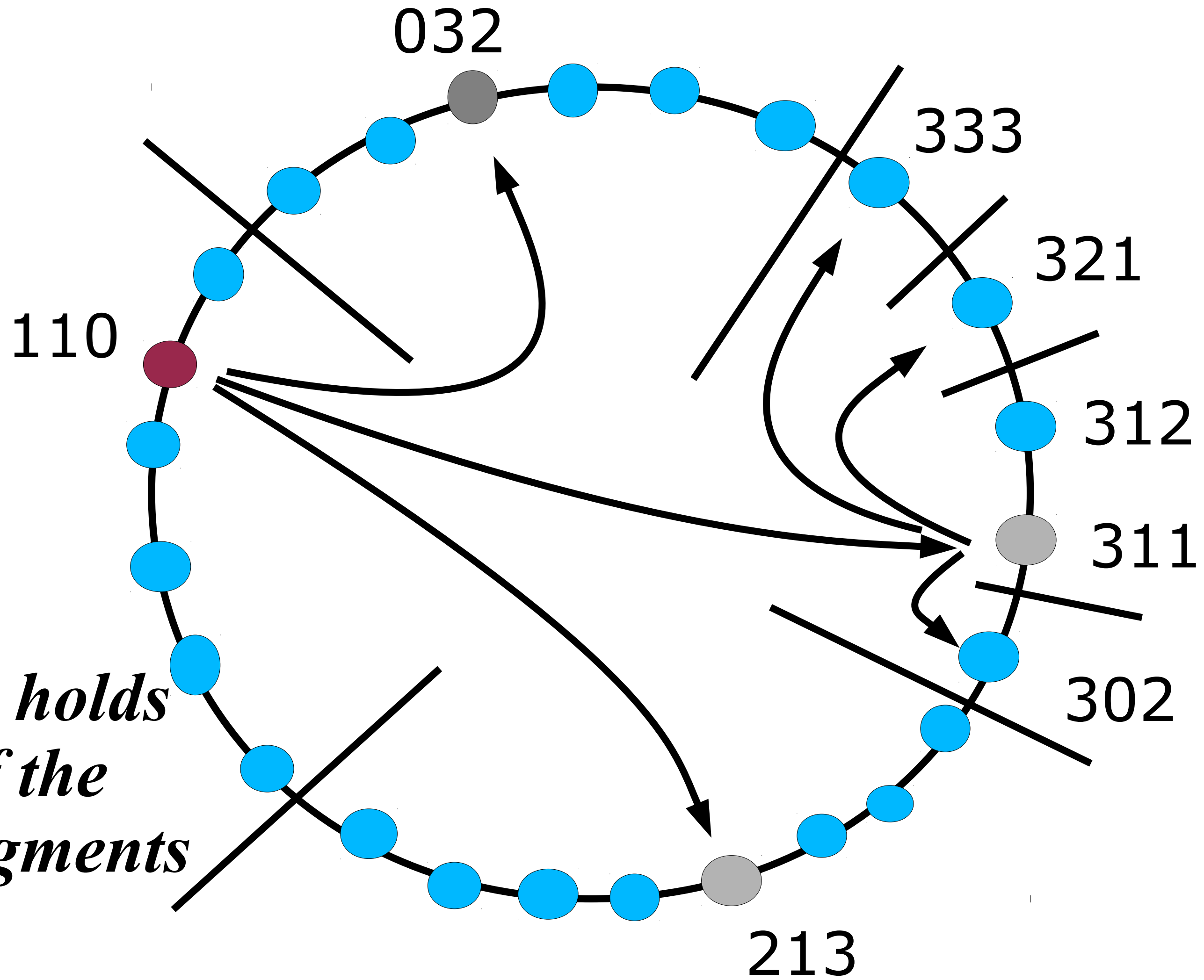
- A routing table that with r rows, each row has routing entries of k nodes.
  - $r = \log_k(n)$
- Several nodes are candidates for each entry.
- Pastry
  - 32 rows
  - 16 entries per row
  - Any node found in 32 hops.
  - GUID space is 16^{32} or 2^{128}

Pastry routing (example with k=4 not 16)



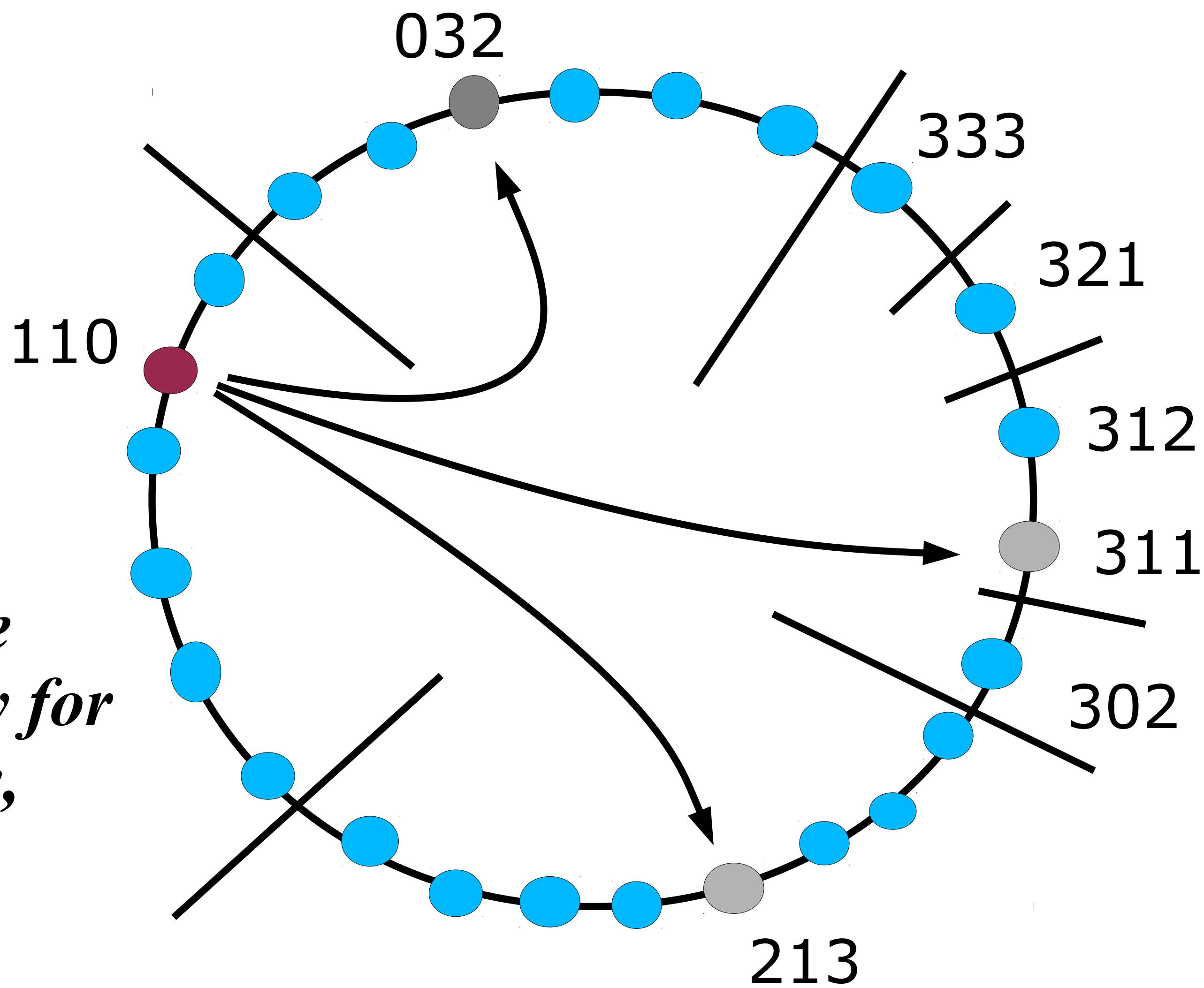
first row of 110 holds entries for each of the three other segments

Pastry routing (example with k=4 not 16)



second row of 311 holds entries for each of the three other sub segments

Pastry routing (example with k=4 not 16)



why did we choose 311 to be the entry for the 300-333 range, why not 321?

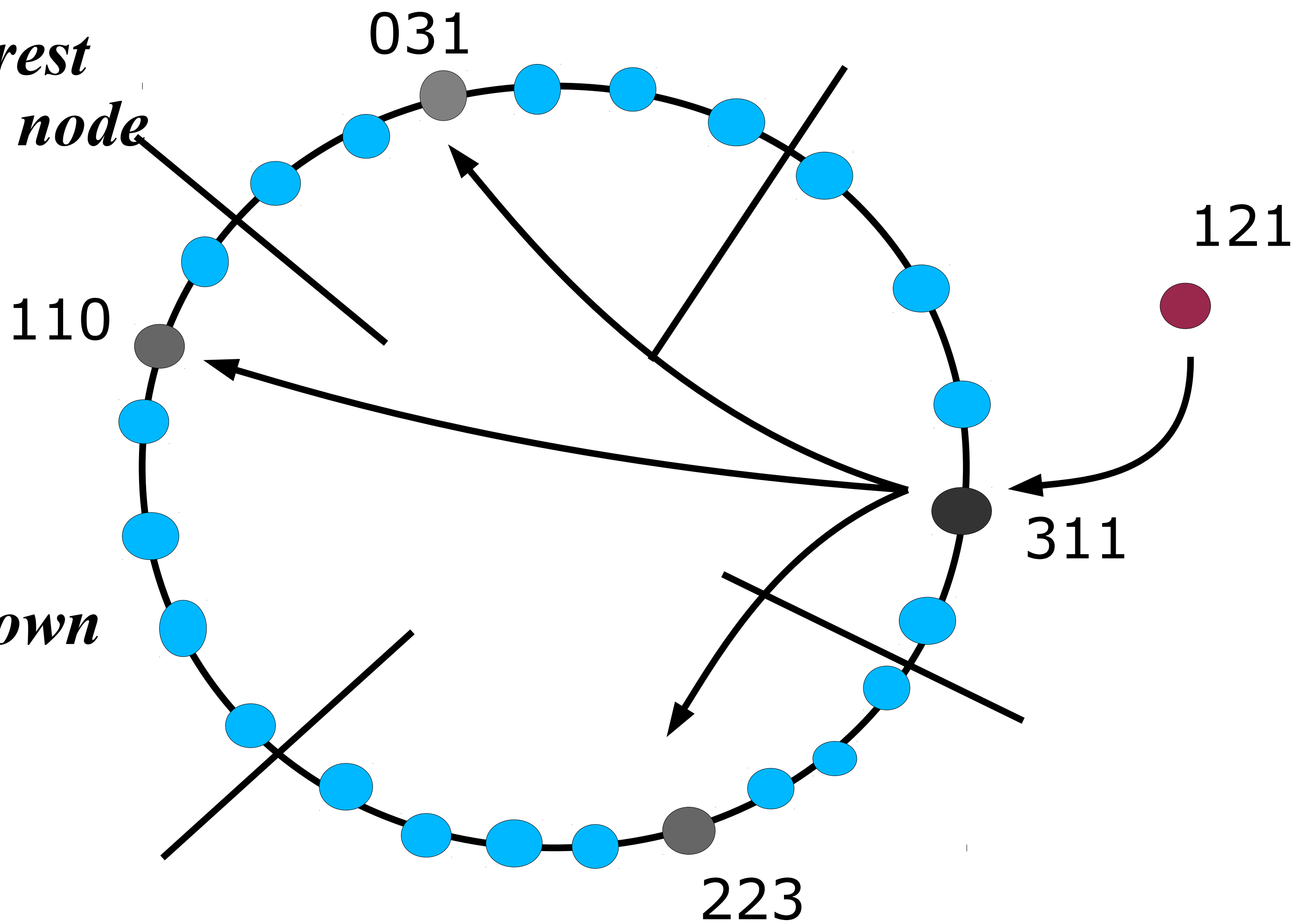
Improvement

- Entries in the routing table should give priority to nodes that are *network wise nearby*.
- How do we detect this?

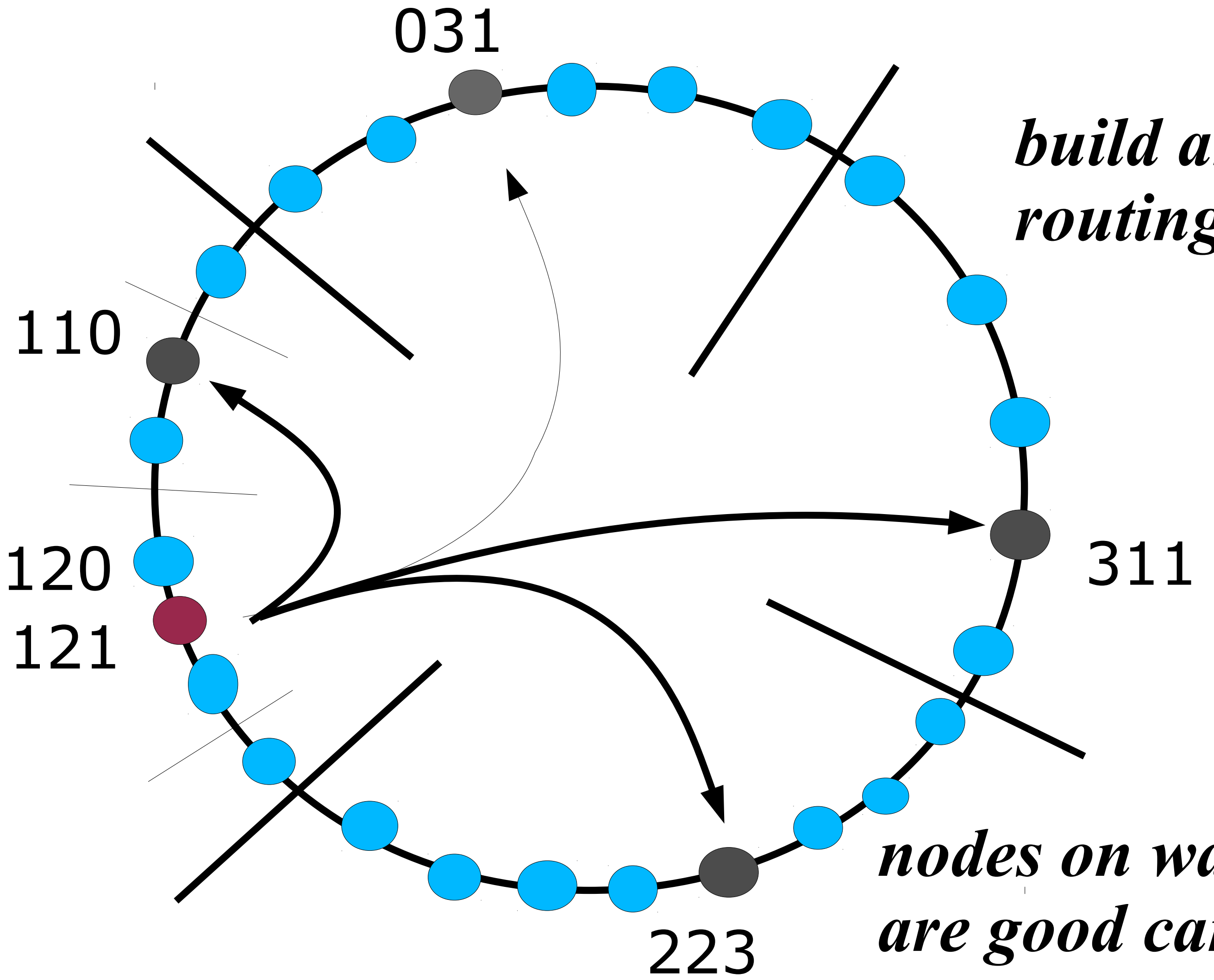
Pastry joining (example with k=4 not 16)

talk to the nearest (network wise) node

route your way down



Pastry joining (example with k=4 not 16)

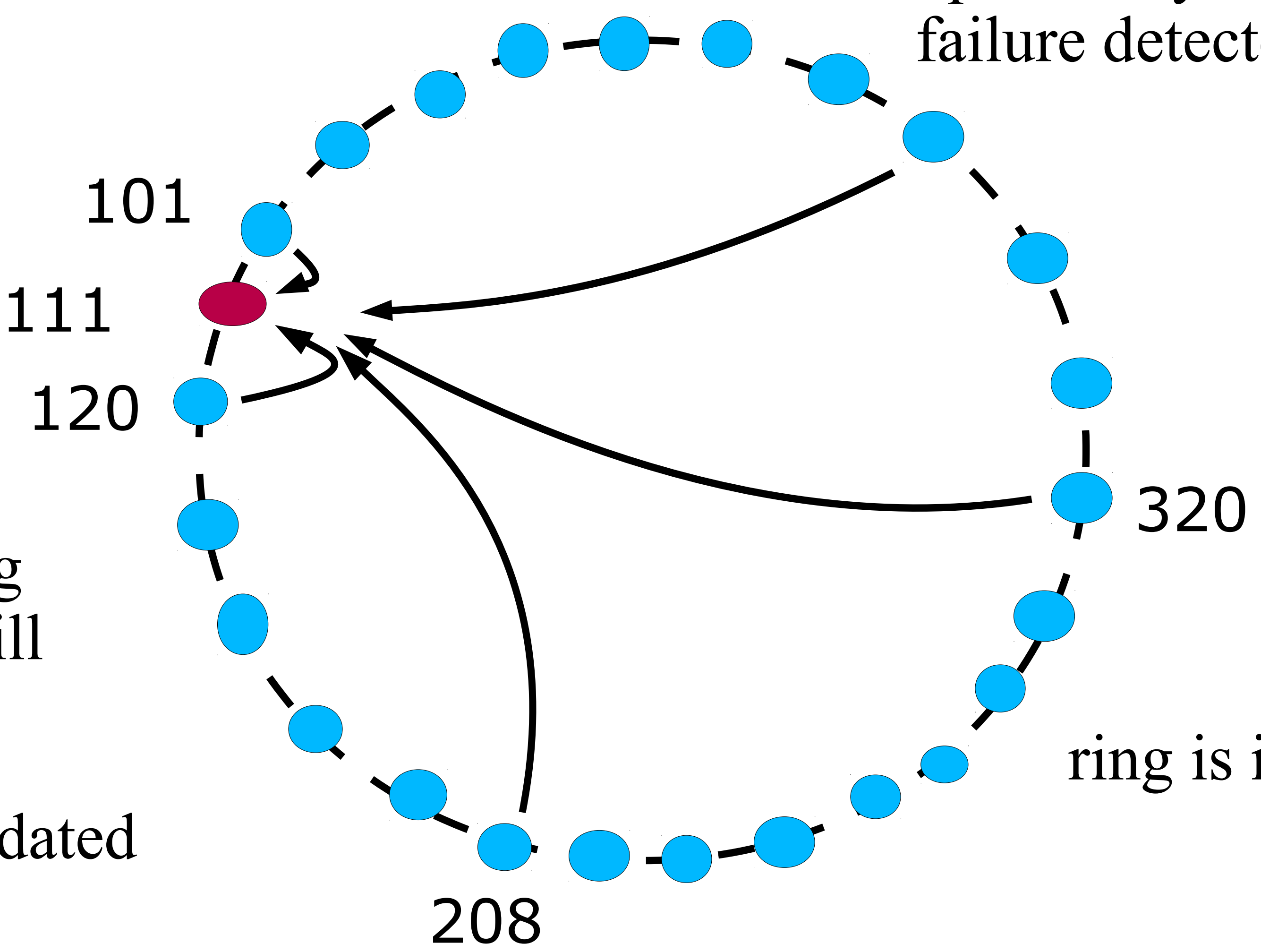


build an initial routing table

squeeze in

nodes on way down are good candidates

leaving



routing tables updated by when failure detected

nodes with leaving node in leaf set will detect it

leaf set updated

ring is imaginary

robustness

- Routing tables can have multiple nodes in each entry, giving priority to the closest but any one will work.
- If nodes can fail, objects need to be replicated at neighboring nodes.
  - how to coordinate updates
  - versioning
  - R/W set

Usage

- Distributed web caching: Squirrel
  - Each client is part of a DHT and keeps cached pages that can be access by all clients.
- File store: OceanStore/Pond, Ivy
  - Large scale file storage with mutable files.
  - Keeps versions of files to keep track of changes.
  - Can not compete with NFS for local are networks nor with AFS for wide are networks.

Media distribution

- How can we make use of a peer-to-peer network for distribution of files:
  - distributed hash table to locate content holder
  - request parts of the file from each holder
  - why?

Summary

- Distributed Hash Tables (DHT) used to store objects.
  - routing,
  - how to join and leave
  - replication
  - mutable objects