DHT-based P2P Architecture

DHT:
Distributed Hash Table

Hash Table

insert (key, object)
delete (key)
obj = **lookup** (key)

Distributed Hash Table

insert (key, object)

delete (key)

obj = **lookup** (key)

DHT:

Objects can be stored in any node in the network.

Example:

Given a torrent file, find the list of peers seeding or downloading the file.

Implementation:
A centralized directory of which node stores which key (object).

How to do this in a fully distributed manner?

Idea: Have a set of established rules to decide which key (object) is stored in which node.

Rule: Assign IDs to nodes and objects. An object is stored in the node with closest ID.

How to assign ID?

Given a object, how to find the closest node?

Pastry

To assign ID, we can use a hash (e.g. into a 128 bit string).

e.g., hash IP address, URL, name etc.

To find the closest node to a given object, each node can store the list of all other nodes.

But this is not scalable.

More scalable solution: Each node only knows a small, constant number of nodes, in a routing table.

Suppose an ID is of the form

$$d_1\ d_2\ d_3\ \ldots\ d_m$$

with digit $d_i = \{0, 1, 2, \ldots n\}$

Suppose an ID is of the form

$$d_1\ d_2\ d_3\ ...\ d_m$$

with digit $d_i = \{0, 1, 2, ...\ n\text{-}1\}$

e.g., $n = 10$, $m = 4$, then ID looks like 0514, 2736, 4090 etc.

Suppose an ID is of the form

$$d_1\ d_2\ d_3\ \ldots\ d_m$$

with digit $d_i = \{0, 1, 2, \ldots n\text{-}1\}$

e.g., $n = 3$, $m = 4$, then ID looks like 1210, 1102, 2011 etc.

A node knows $(m)x(n\text{-}1)$ neighbors -- m groups, each group with $n\text{-}1$ entries.

Routing Table for Node 1201

0121	137.12.1.0
2001	22.31.90.9
1021	45.24.8.233
1121	:
1210	:
1222	:
1200	:
-	-

Each node *i* keeps a table

next(*k*,*d*) = address of node *j* such that

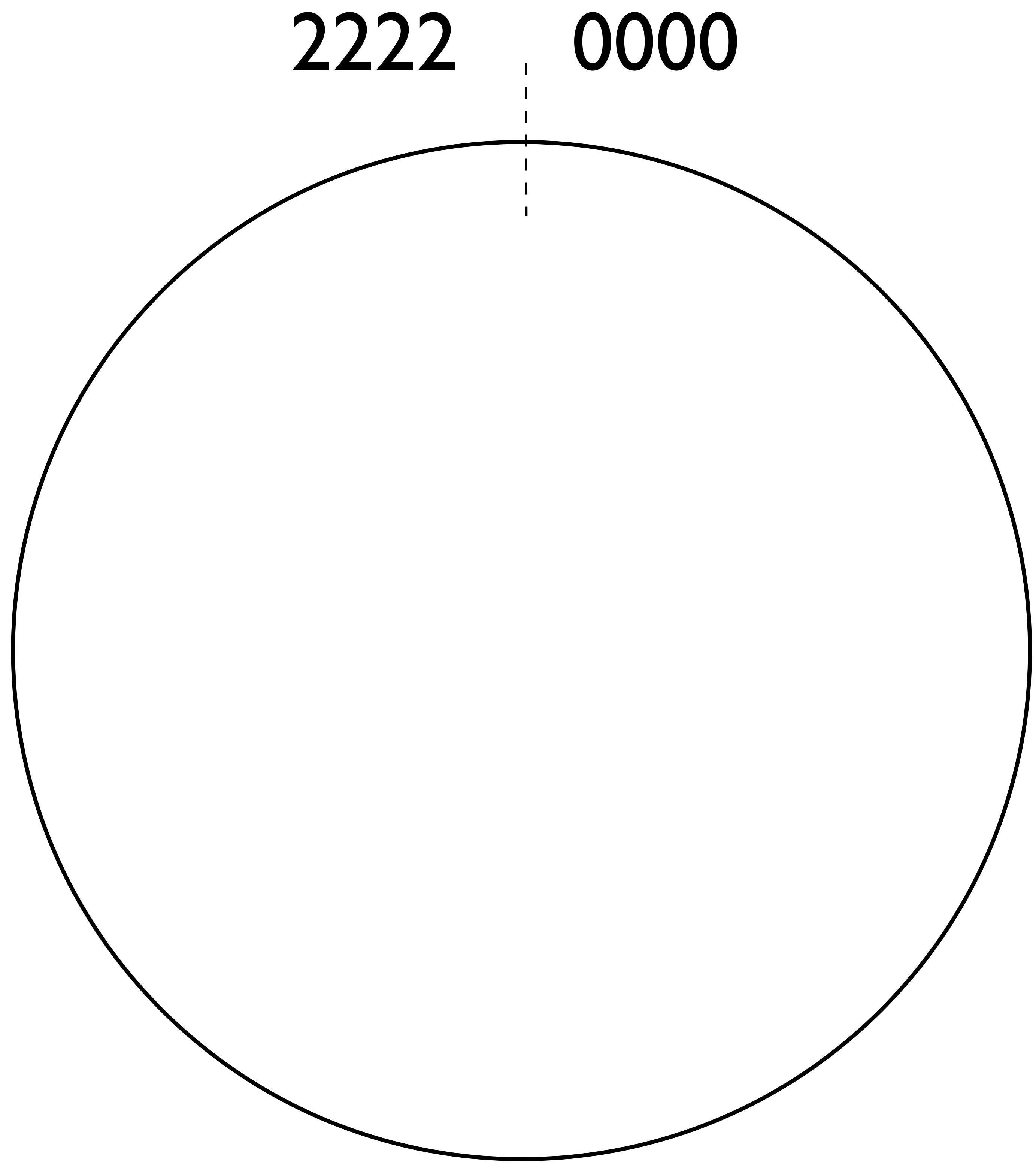
1. *i* and *j* share prefix of length *k*
2. *k+1* digit of *j* is *d*
3. node *j* is the "physically closest" match

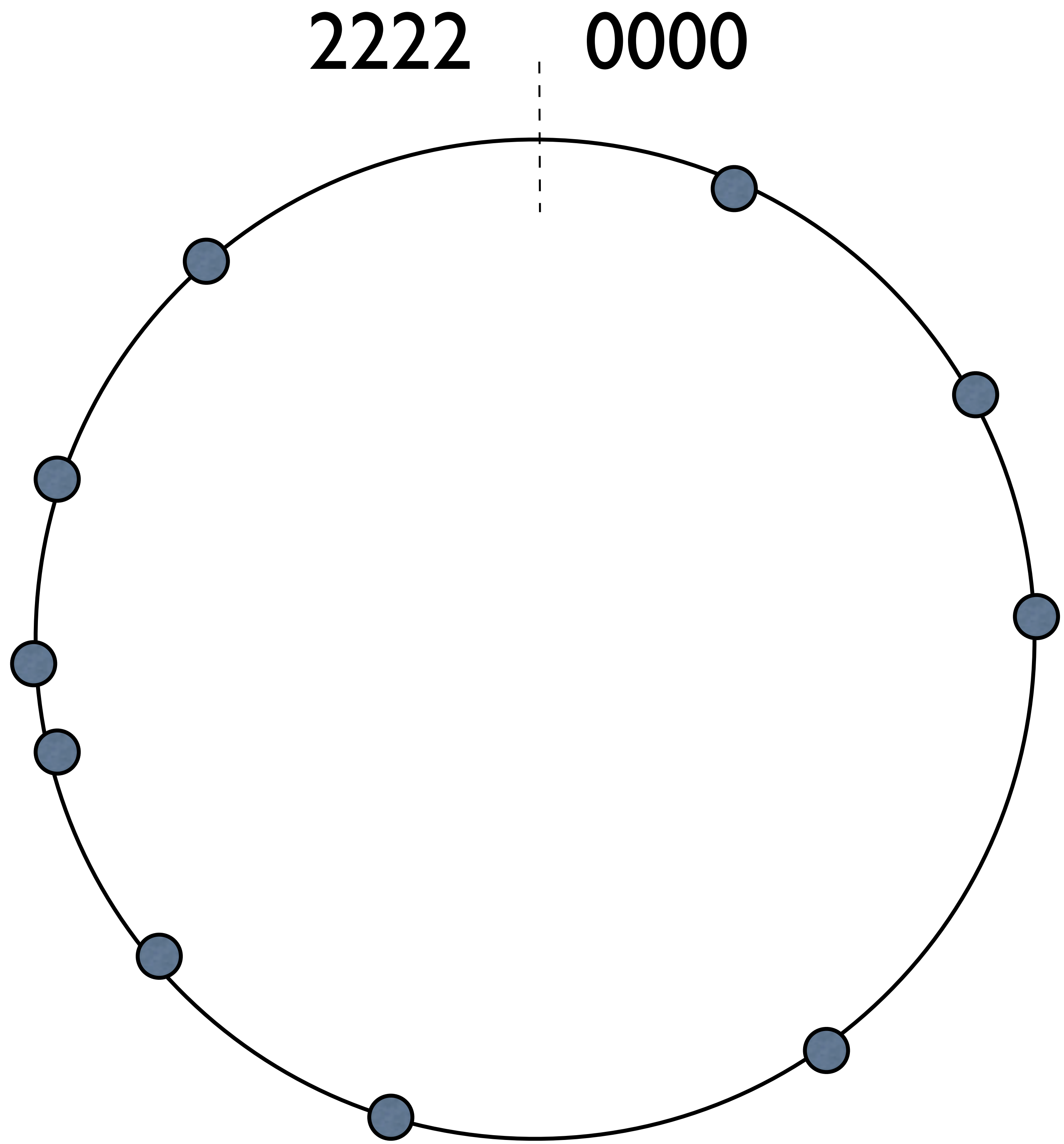
In addition, each node knows *L* other nodes with closest ID. (*L*/2 above, *L*/2 below)

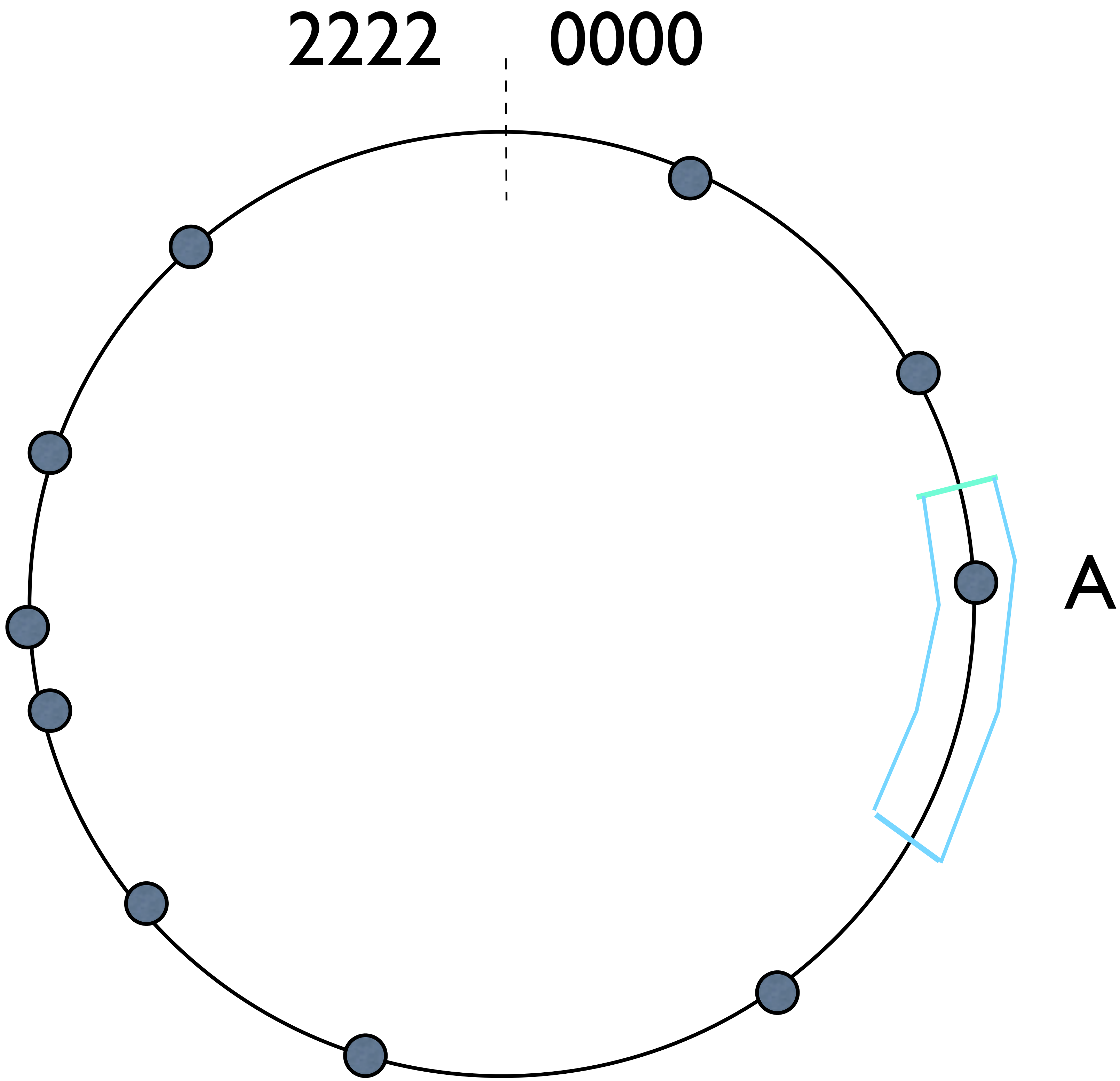
Leaf Set for Node 1201

1122	2.12.1.0
1200	12.30.99.90
1202	78.8.73.231

Visualizing Pastry

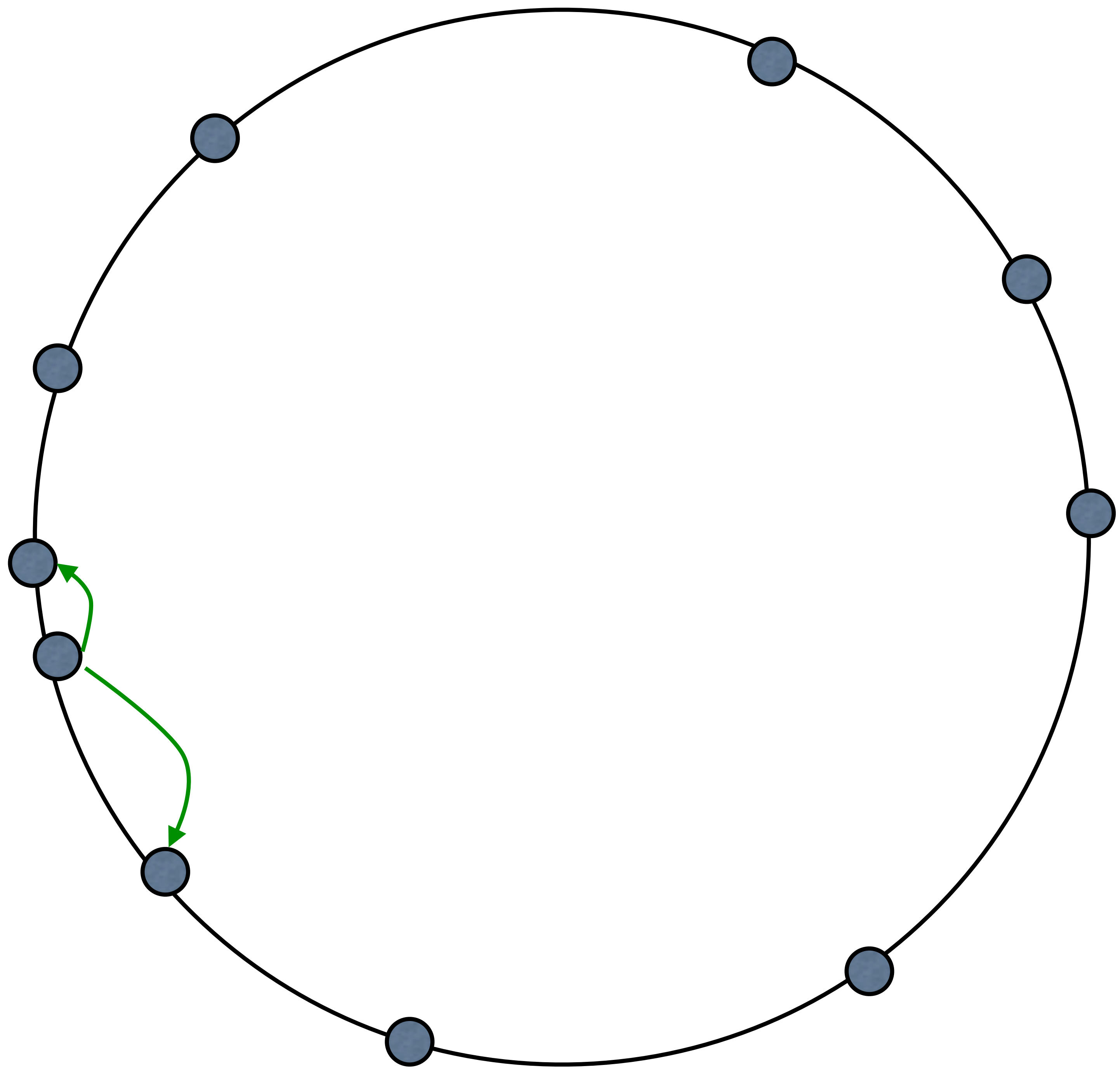

2222
0000


2222
0000

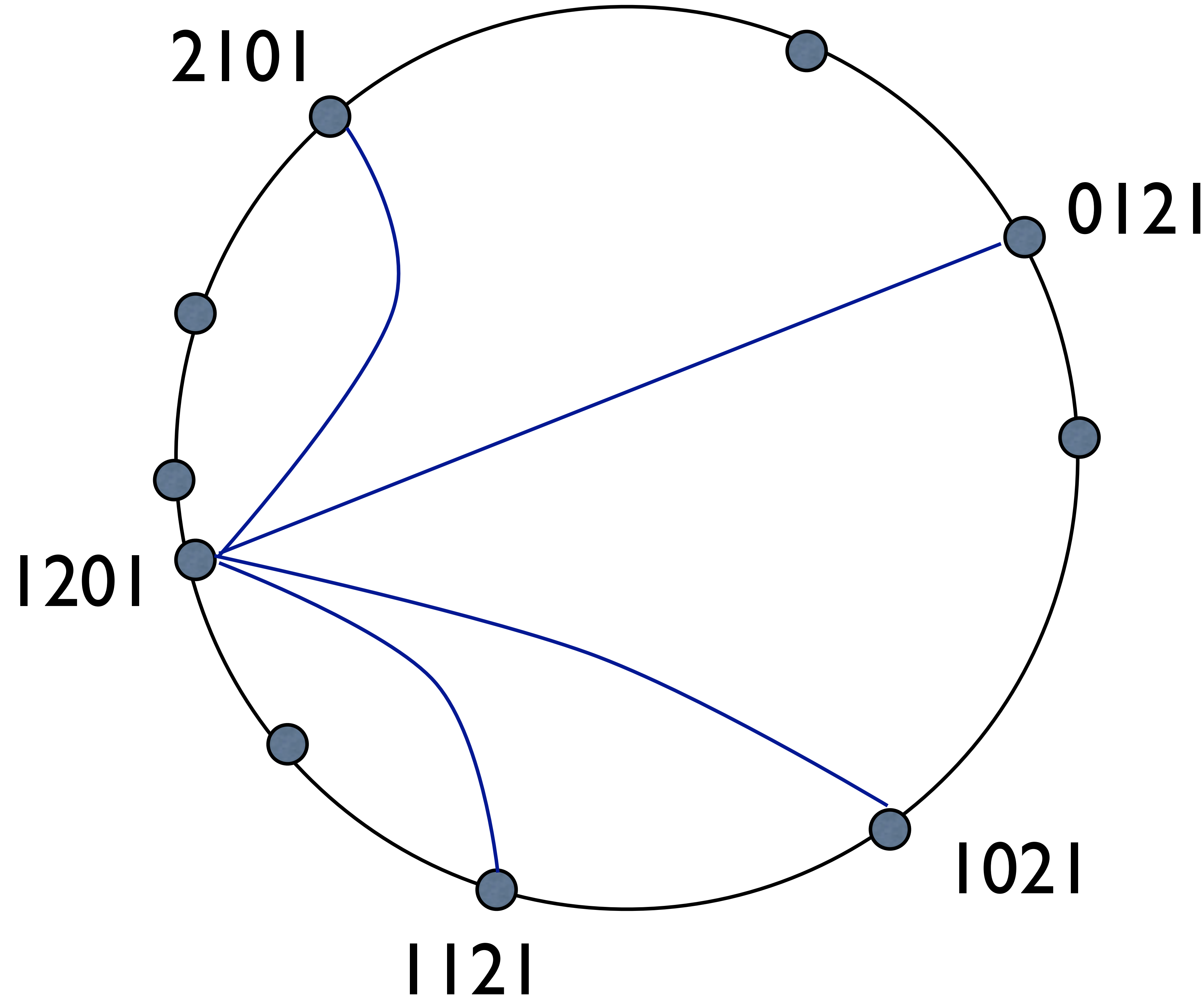


Any object whose IDs that falls within the blue region is stored in node A.

Leaf Sets



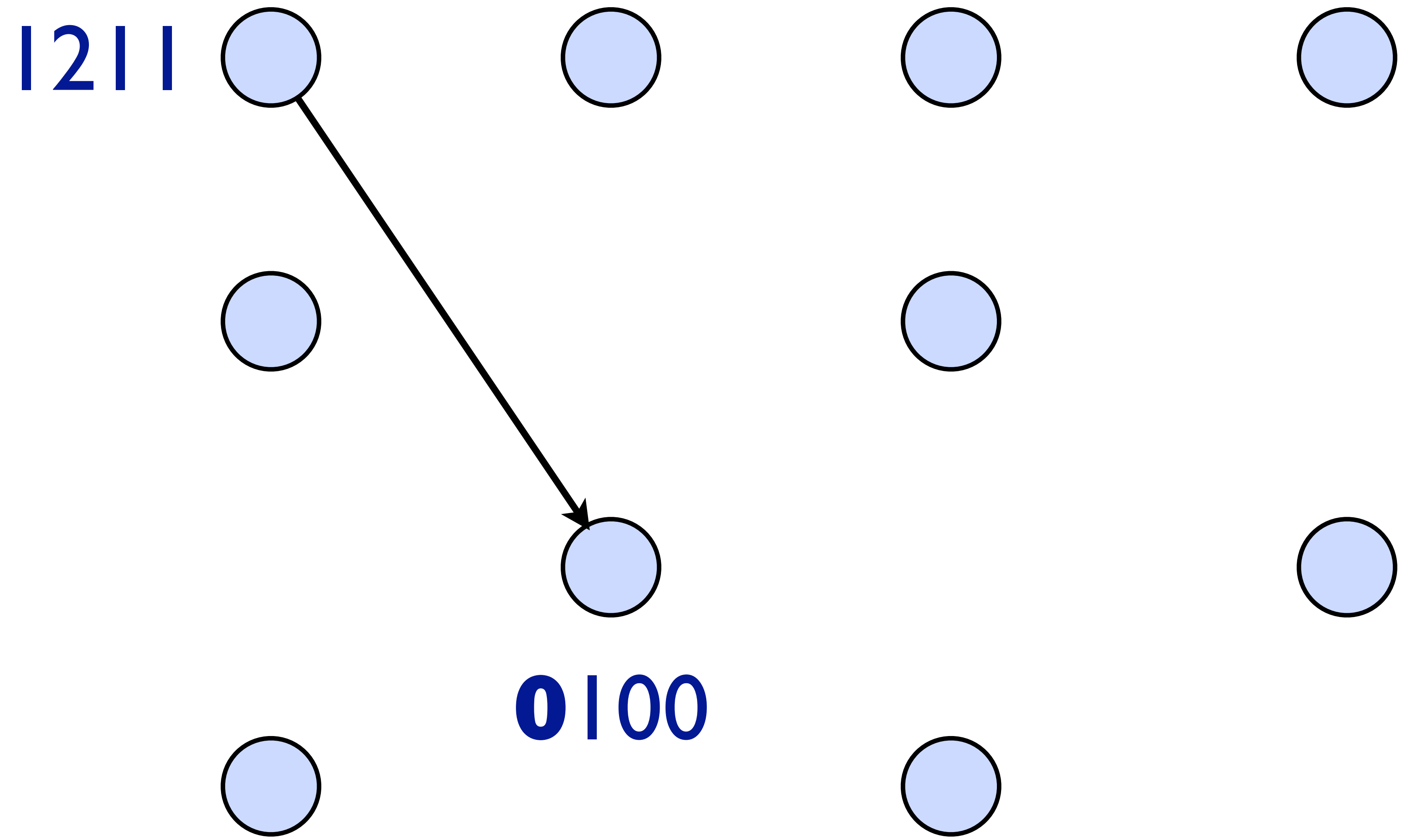
Example Routing Table



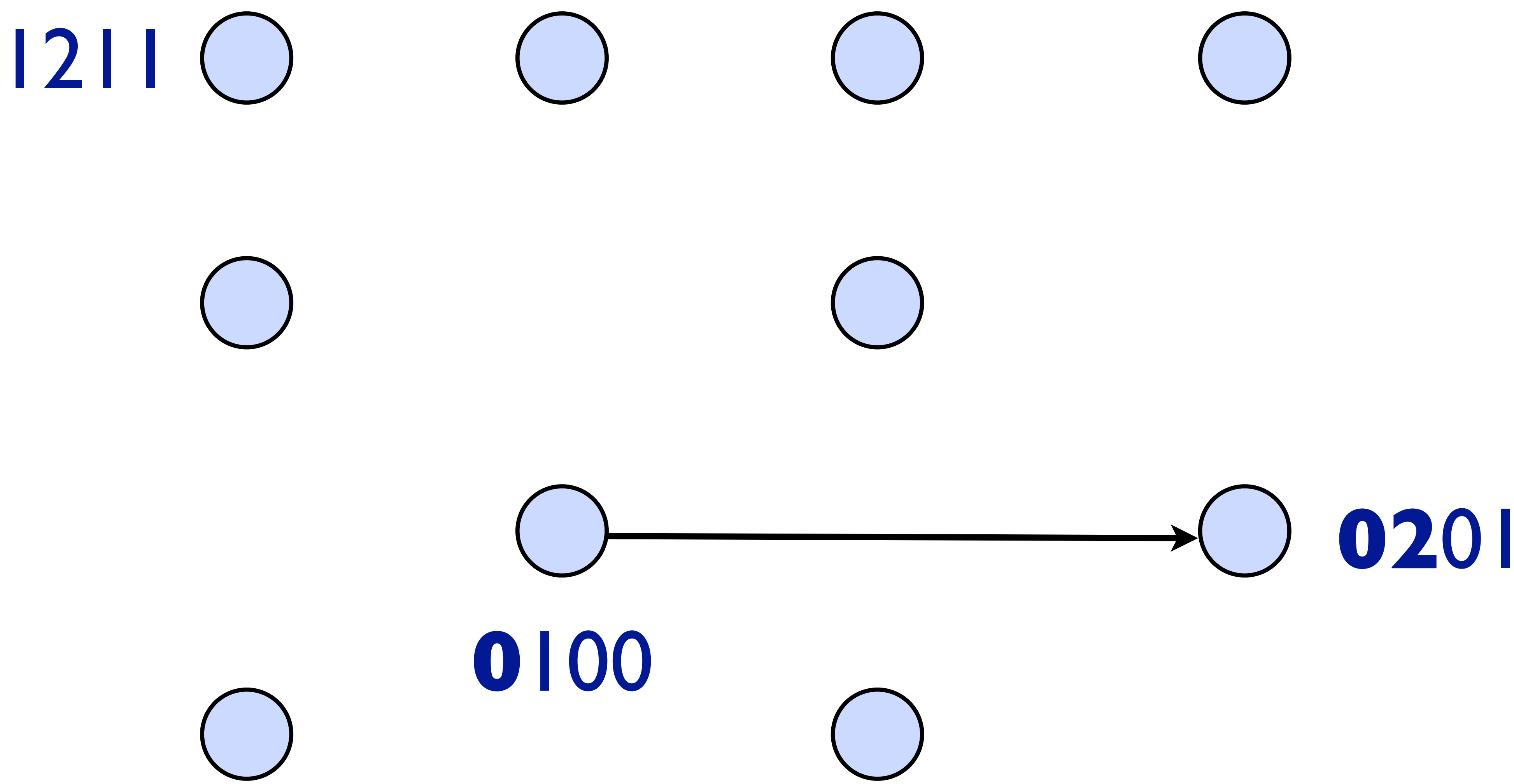
Recall that we want to find the node with ID closest to the ID of a given object.

node = route(object_id)

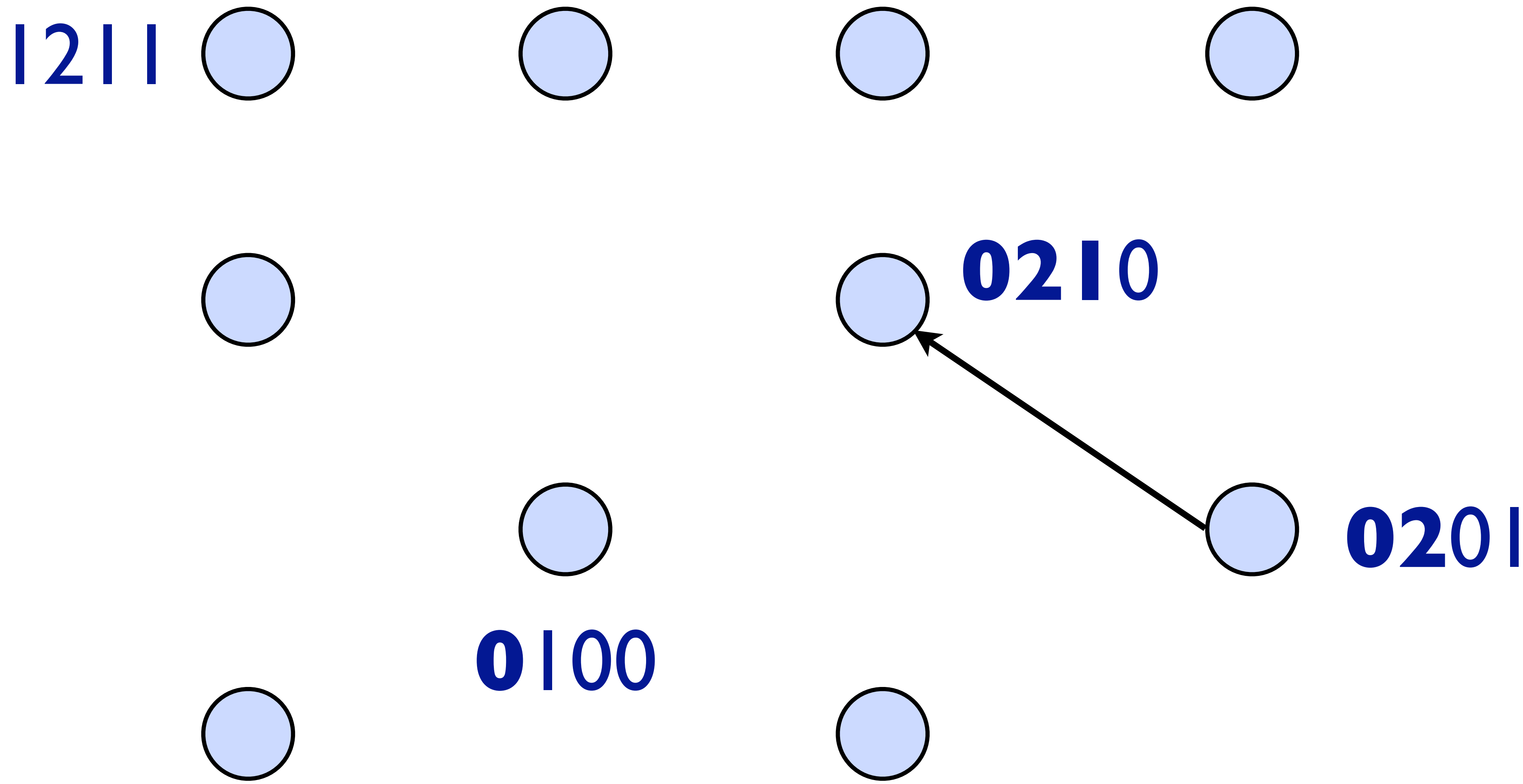
route(0212) issued at node 1211.
1211 forward the request to next(0,0).



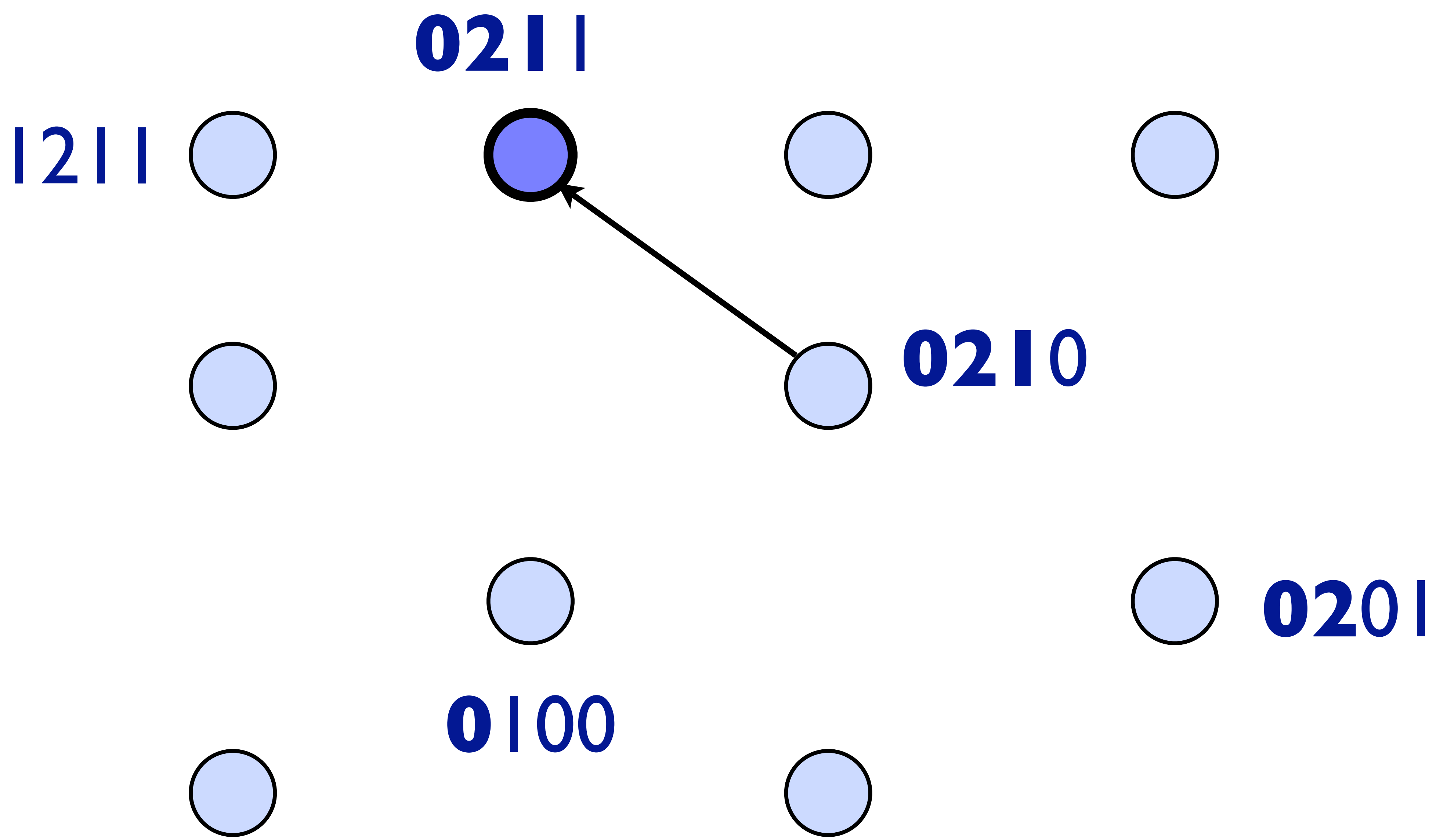
route(0212) received at 0100.
0100 forward the request to next(1,2).



route(0212) received at 0201.
0201 forward the request to next(2,1).



0201 found that it is within the range of its leaf set, and forward it to the closest node.



After 4 lookups, we found the node closest to 0212 is 0211.

We can now implement the following using route()

insert (key, object)

delete (key)

obj = **lookup** (key)

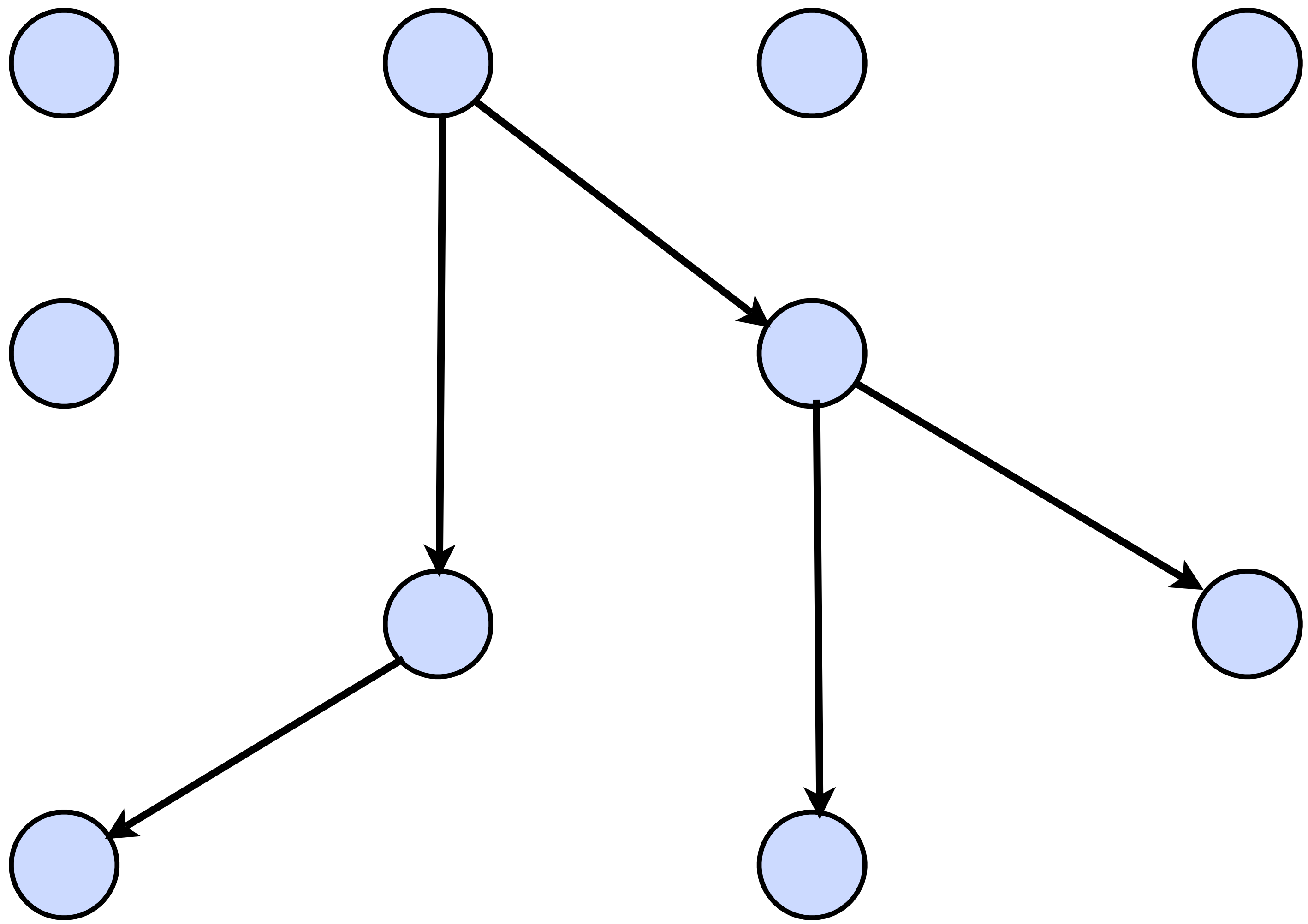
Scribe

Application-Level Multicast over Pastry

Recall: IP multicast is not deployed. We therefore need an alternative multicast solution.

In Application-Layer Multicast, nodes duplicate and forward messages at the application layer.

Nodes need not be a subscriber of a group to forward messages for the group.

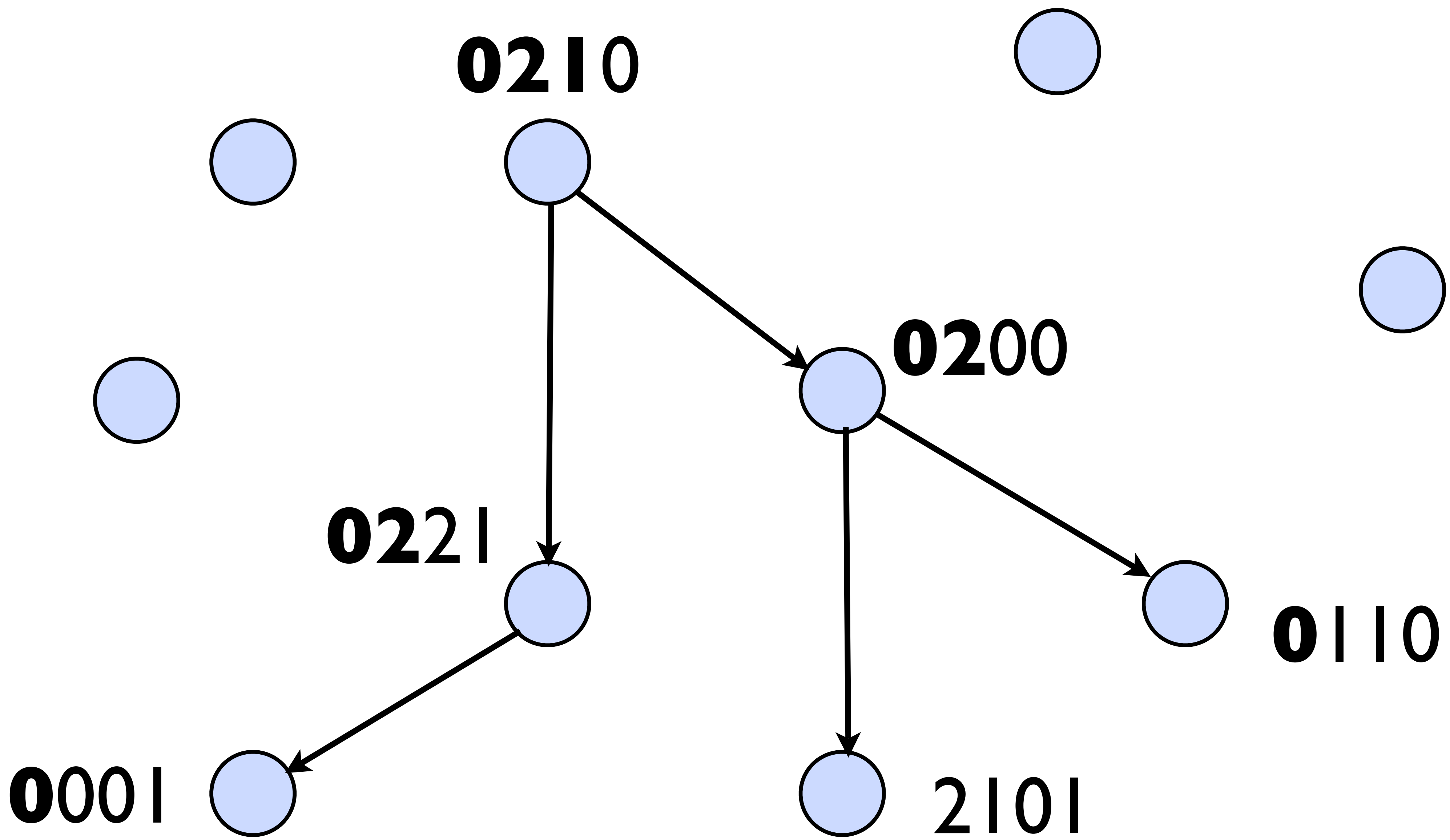


A major component of application-layer multicast is construction of multicast tree (or, who should forward to who?)

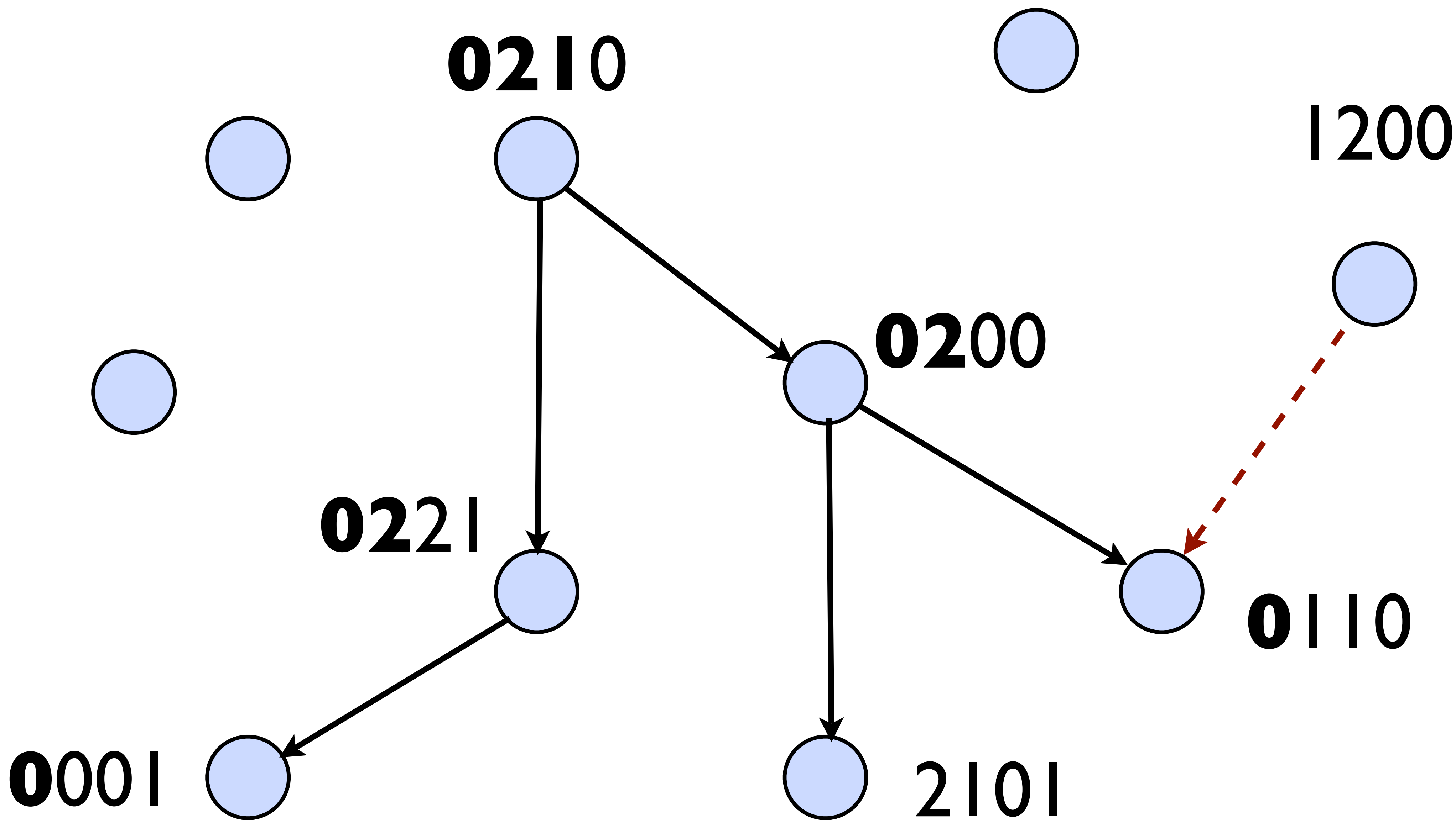
Scribe uses Pastry to construct the tree. Each multicast group is assign a random ID from the same ID space as nodes and objects.

The node with the closest ID to the group ID serves as a "rendezvous point" for the group.

Tree for group 0212



1200 join the group by routing a join message to group ID.



DHT-based P2P Architecture