DHT-based P2P Architecture

No server to store game states

Not scalable to replicate states of every object in the game in every client

Idea: split responsibility of storing the states among the clients

Who store what?

Knutsson's Idea: divide game world into region and assign region coordinator to keep the states in the region.

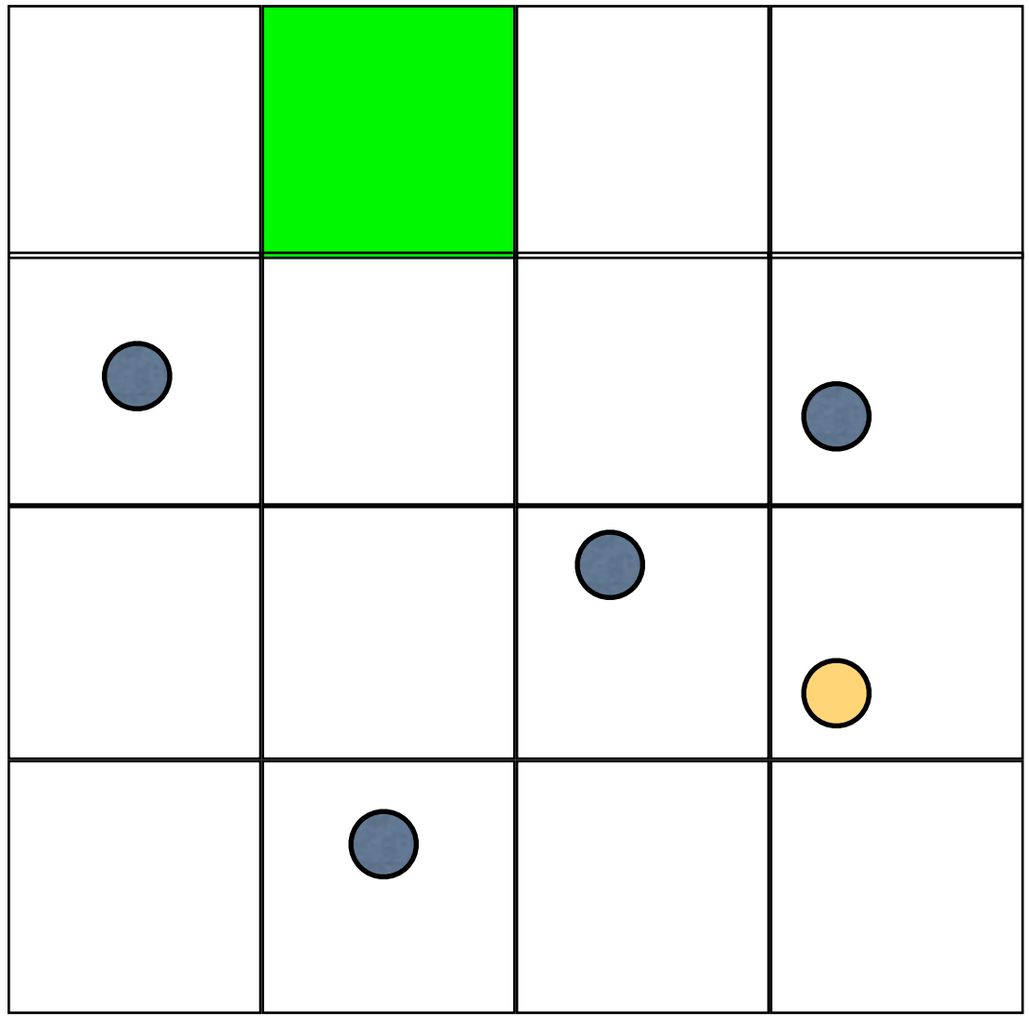
When a player needs to read/write the state of an object, it contacts the coordinator.

Either every client can maintain a directory of coordinators, or

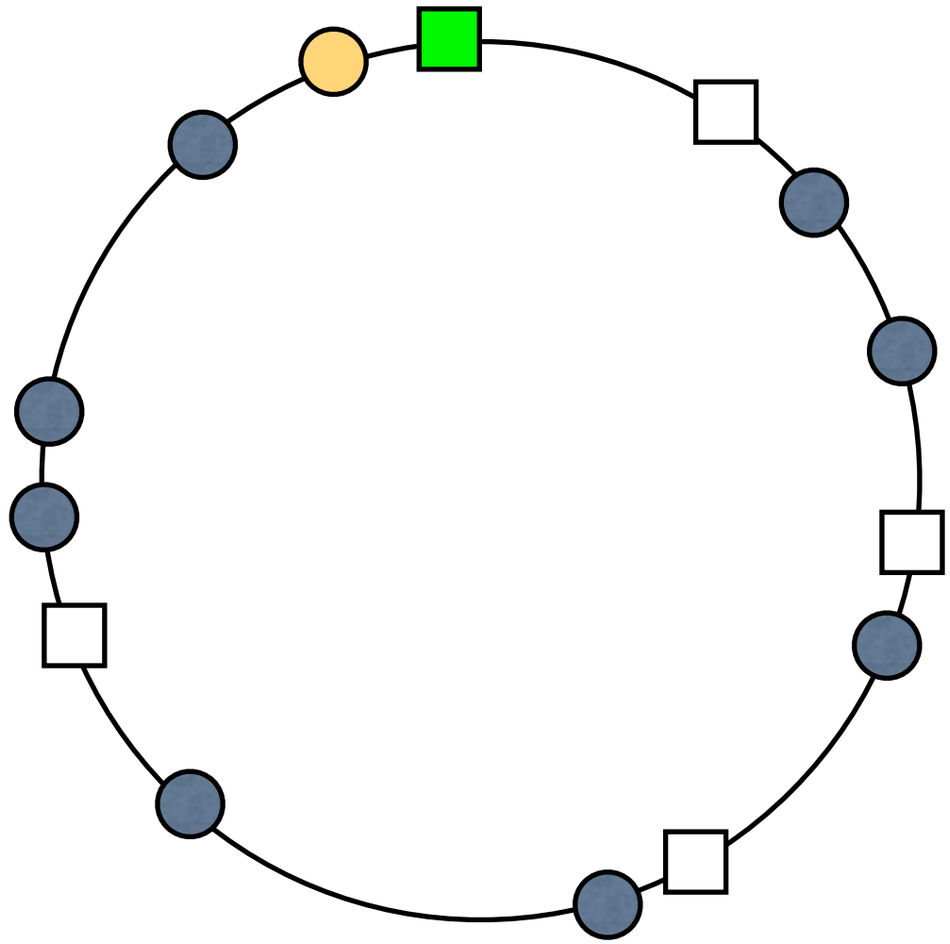
use DHT (Pastry)

Hash regions and nodes into the same ID space. The node whose ID is closest to the ID of a region becomes the coordinator.

Game Map



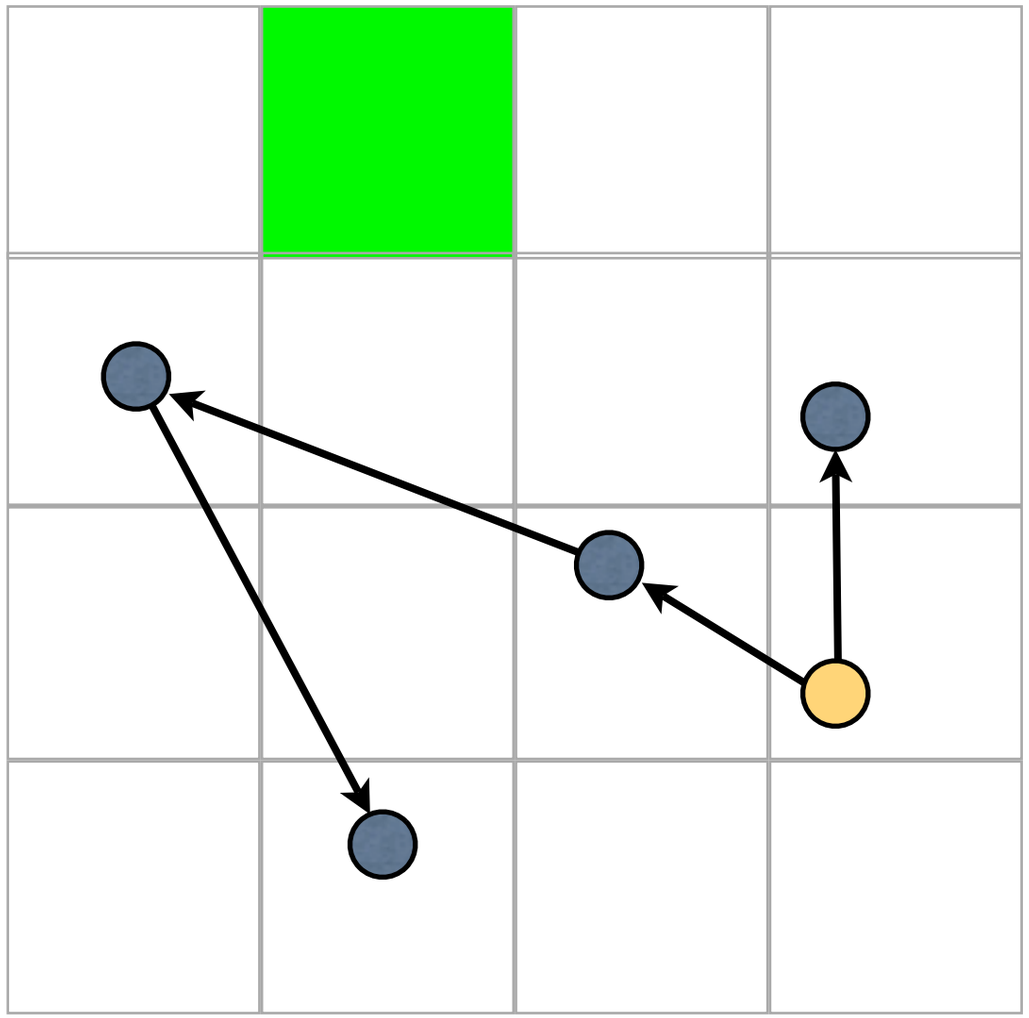
DHT ID space



The coordinator is likely to be not from the same region it is coordinating, reducing the possibility of cheating.

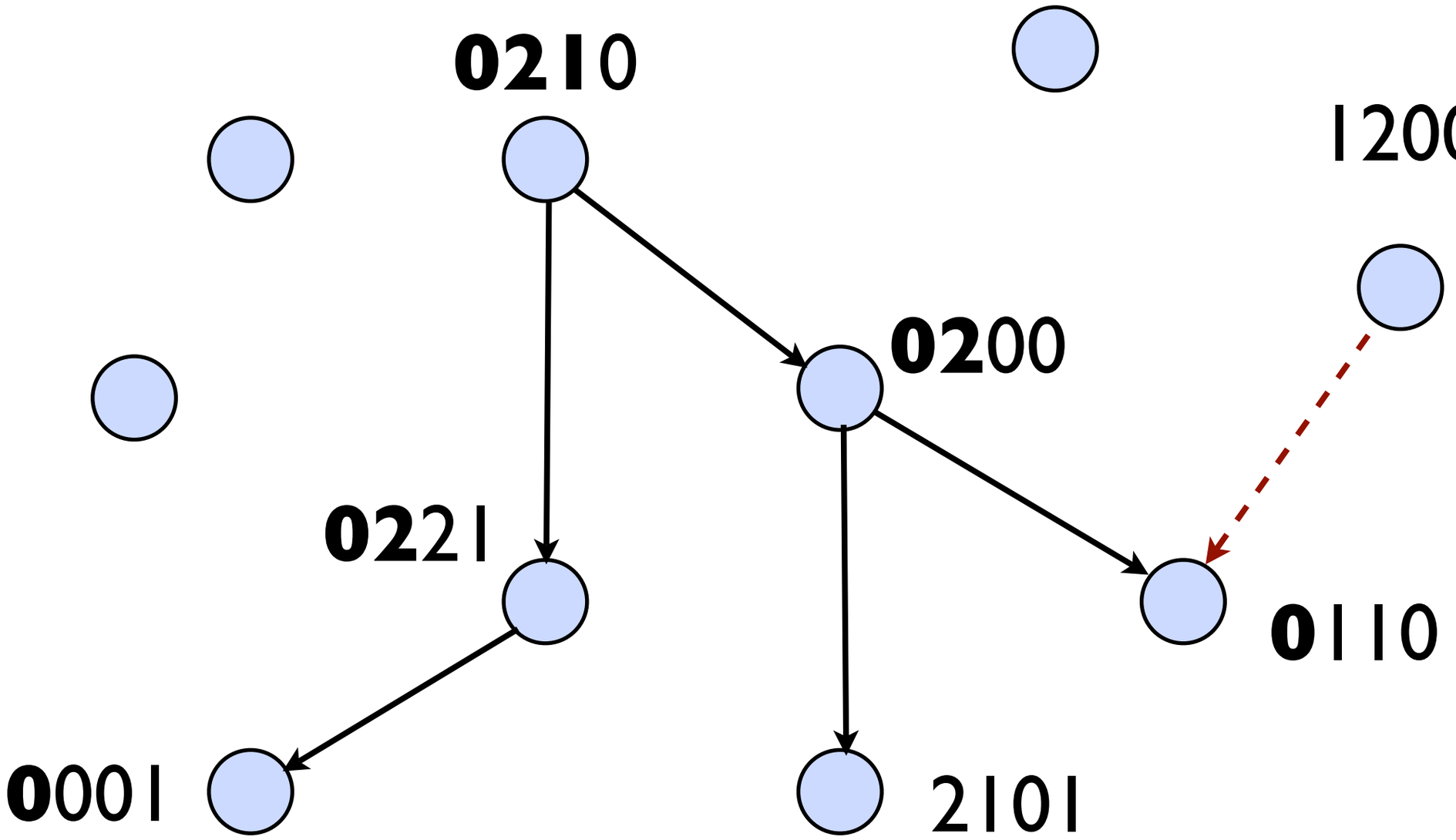
The coordinator is the “server” of the region and serves as the root of multicast tree as well.

Game Map

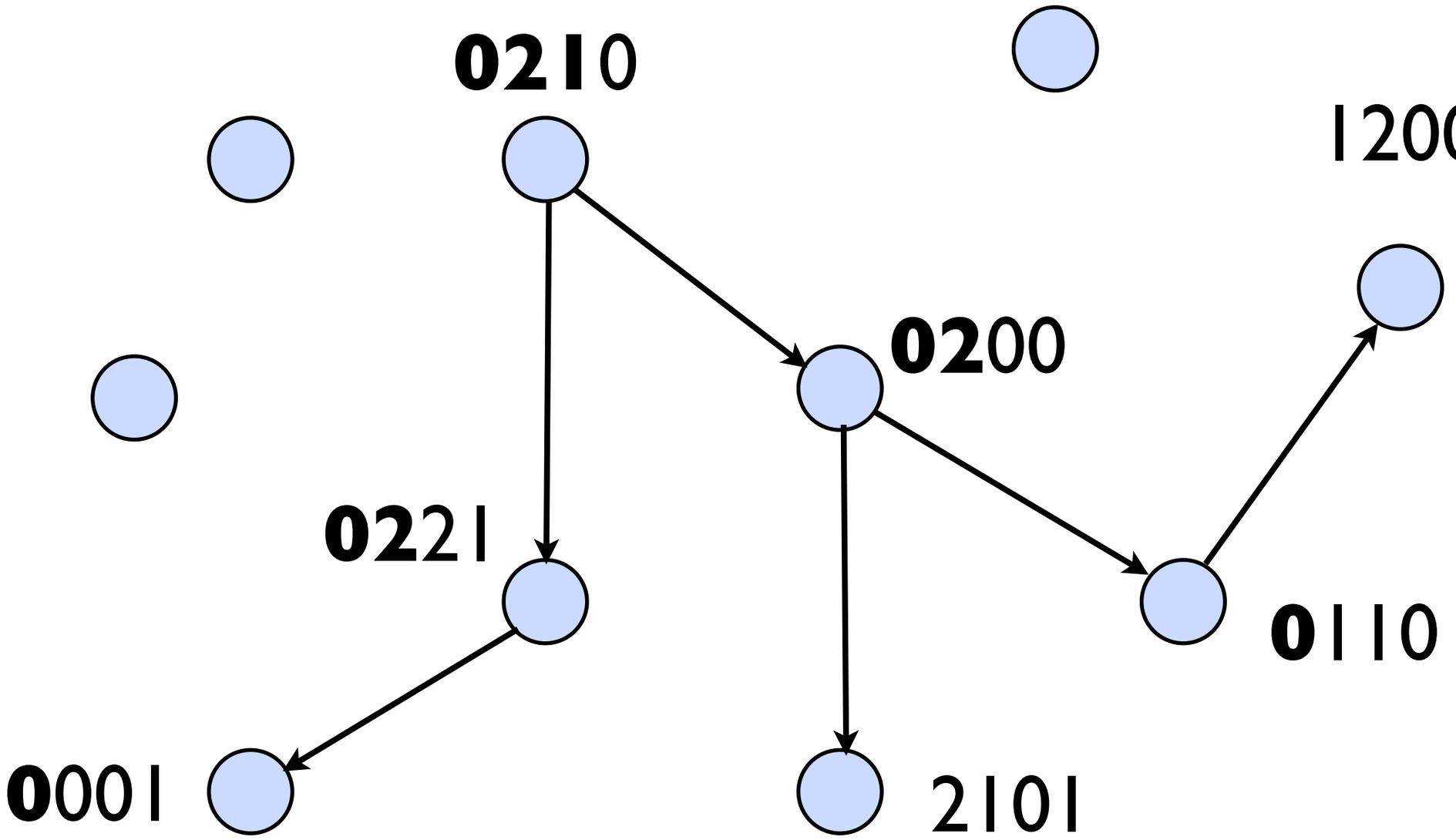


To subscribe to a region with ID r, route a JOIN message to r.

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



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



To update a state of an object, route an update message to the region of that object.

The message will reach the coordinator. The coordinator then forward the updates along the multicast trees to subscribers.

what if coordinator fails?

Pastry would route messages to the next closest node.

Use the next closest node to the region as the backup coordinator.

The primary coordinator knows the backup from its leave set and replicate the states to the backup coordinator.

If the backup receives messages for a region, it knows that the primary has failed and takes over the responsibility.

Issues with Knutsson's scheme

1. No defense against cheating

2. Large latency when
(i) look for objects in a region
(ii) creating new objects
(iii) update state of objects

3. Extra load on coordinators

4. Frequent changes of coordinators for fast moving players.

Knutsson's design is for MMORPG game (slow pace, tolerate latency)

Can similar architecture be used for FPS games?

Colyseus

1. Distribute the states to all nodes, not just coordinator.

2. Support multi-dimensional interest management

3. Reduce latency by prefetching

1. Distribute the states to all nodes, not just coordinator.

Each object is stored in exactly one node ("primary")

Other nodes might store copies of the object ("replica")

State of an avatar should be stored primarily in the node of the corresponding players.

State of objects within a player's AoI should be stored in the node of that player as well.

But movement of players leads to migration of primary.

Best way to place of primary states remains open.

Interest Management: what are the objects within my AoI?

Colyseus supports multi-dimensional interest management

Mercury

Normally DHT supports exact match query only

We need range query to support generalized interest management

$$100 < x < 200$$

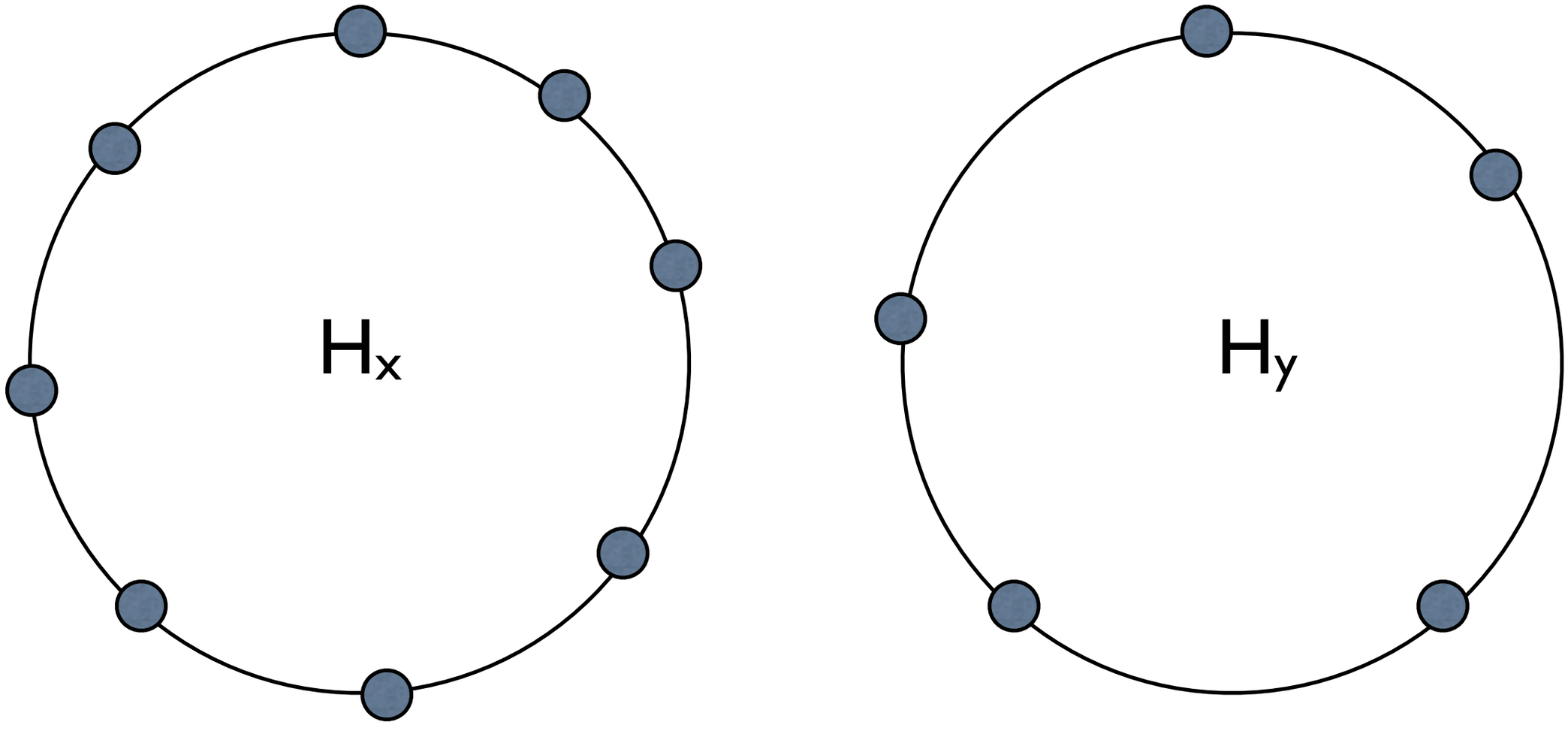
$$600 < y < 700$$

DHT does not support this efficiently because hashing distribute the keys randomly

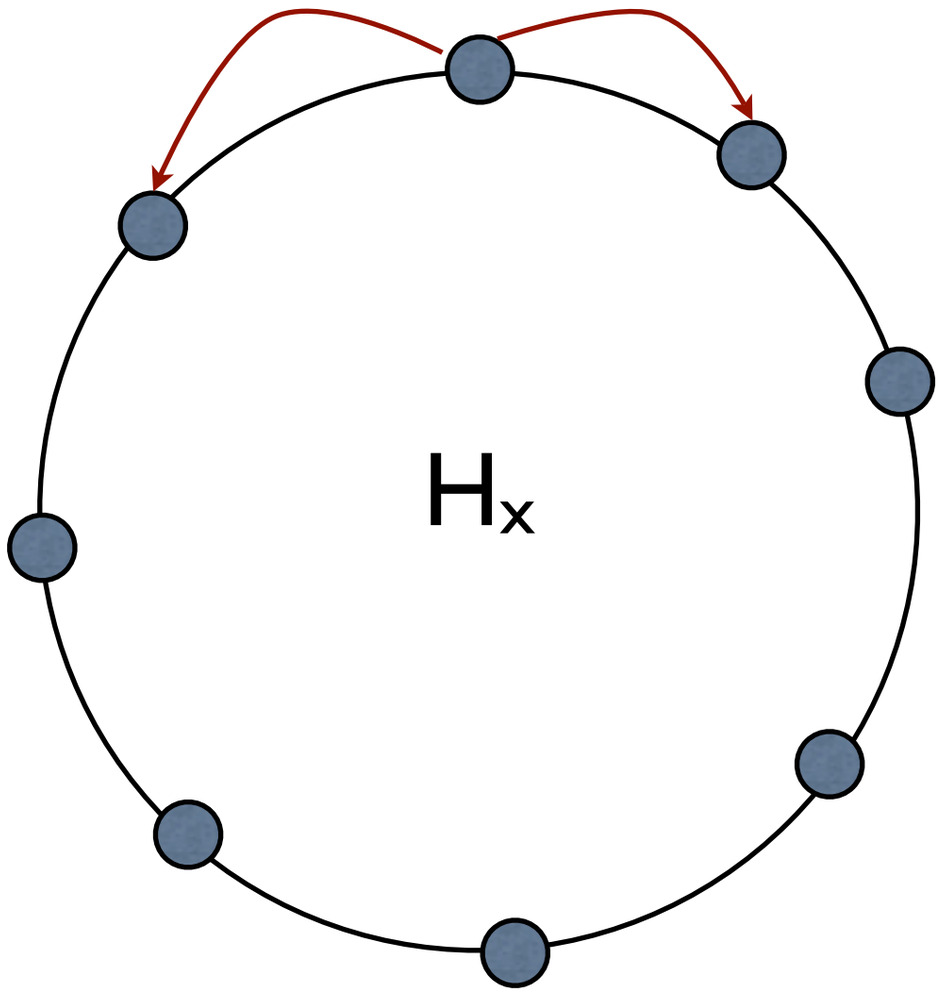
A simple but inefficient solution is to query each values within the range.

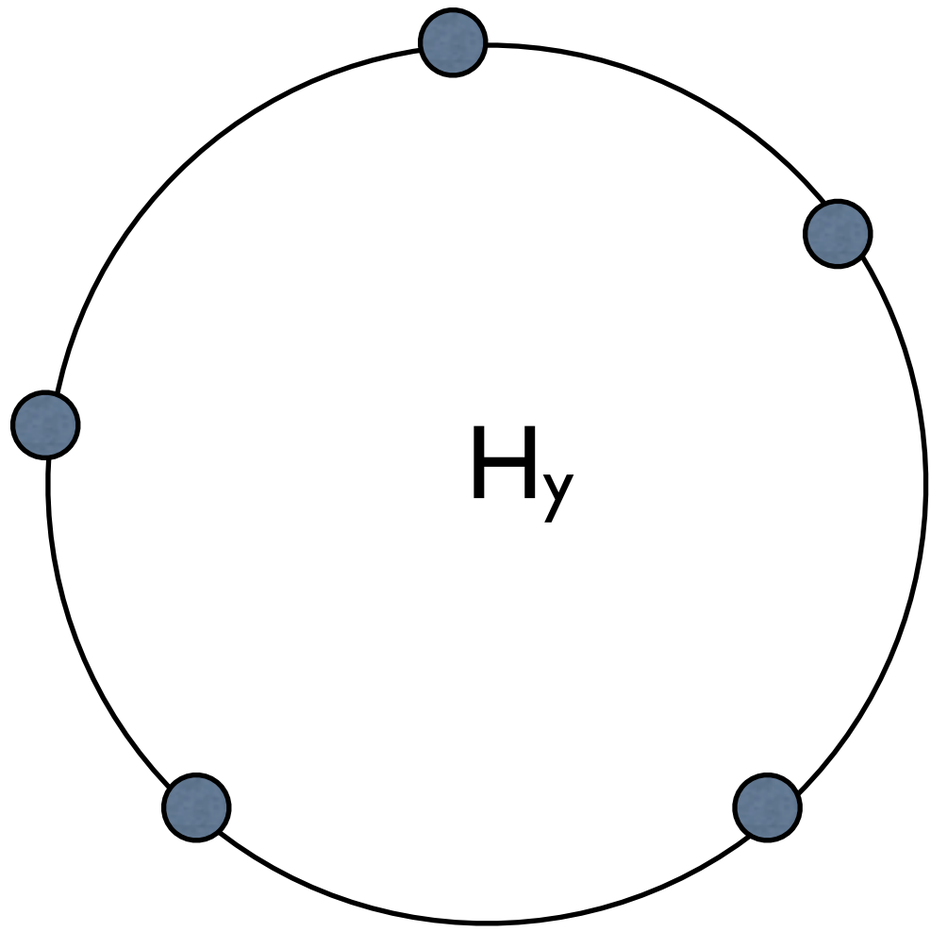
Must not use hashed IDs

Each node is in charge of a range of values in one of the dimension.
Each ring is called a hub.

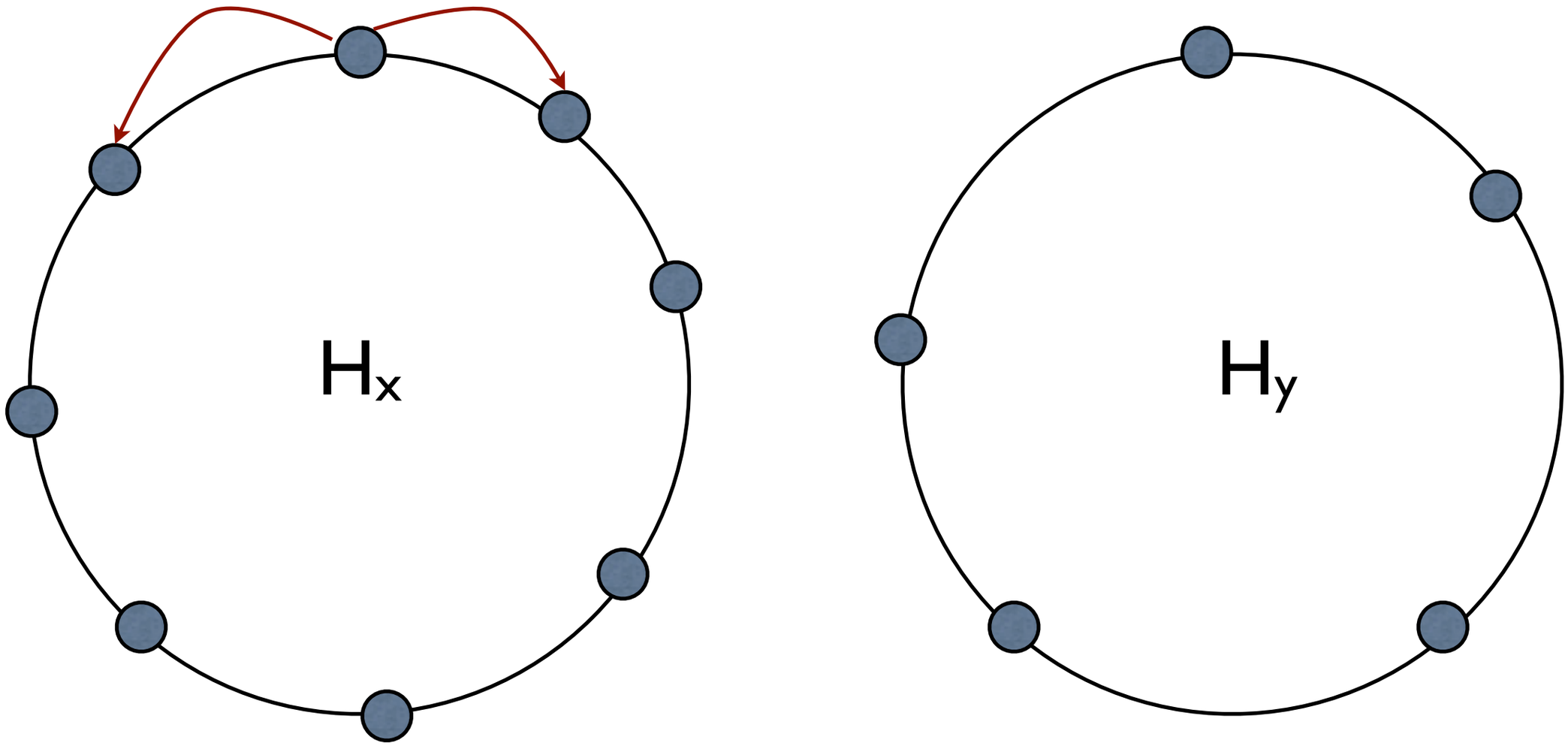


Each node keep tracks of its predecessors and successors.

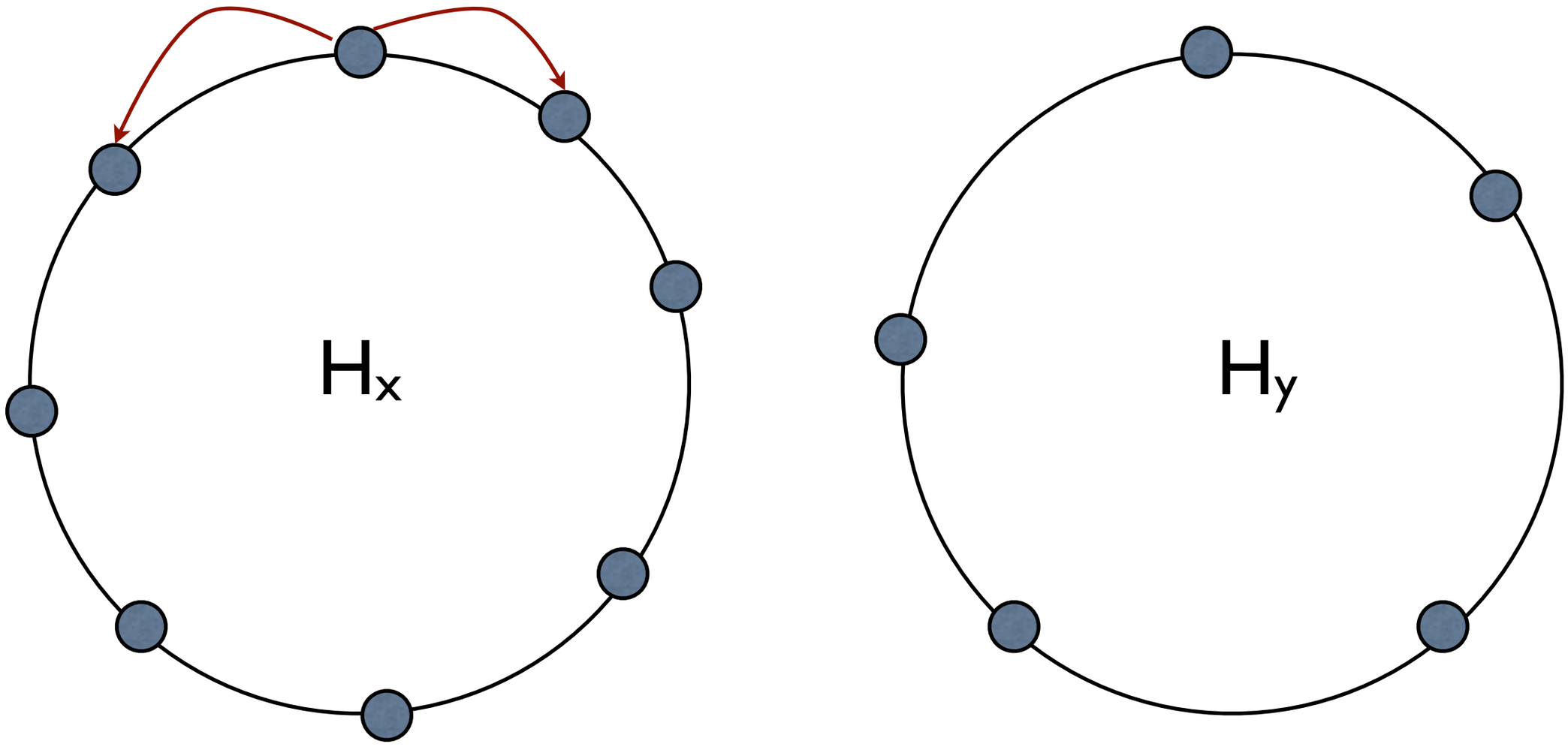




To do an exact lookup, we perform linear search along the ring



To do a range lookup, we perform linear search along the ring and follow the succ/pred links until we find all values in range.

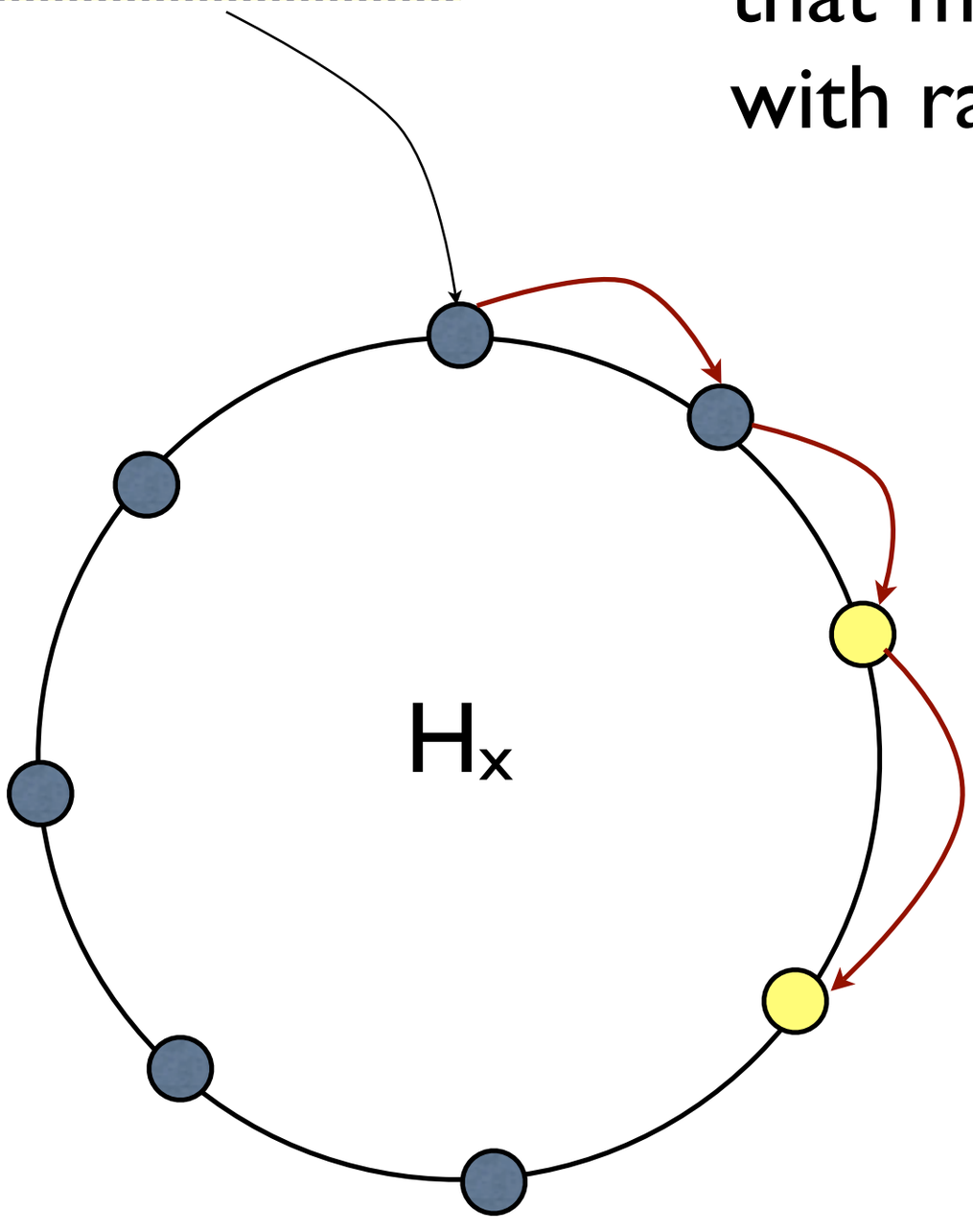


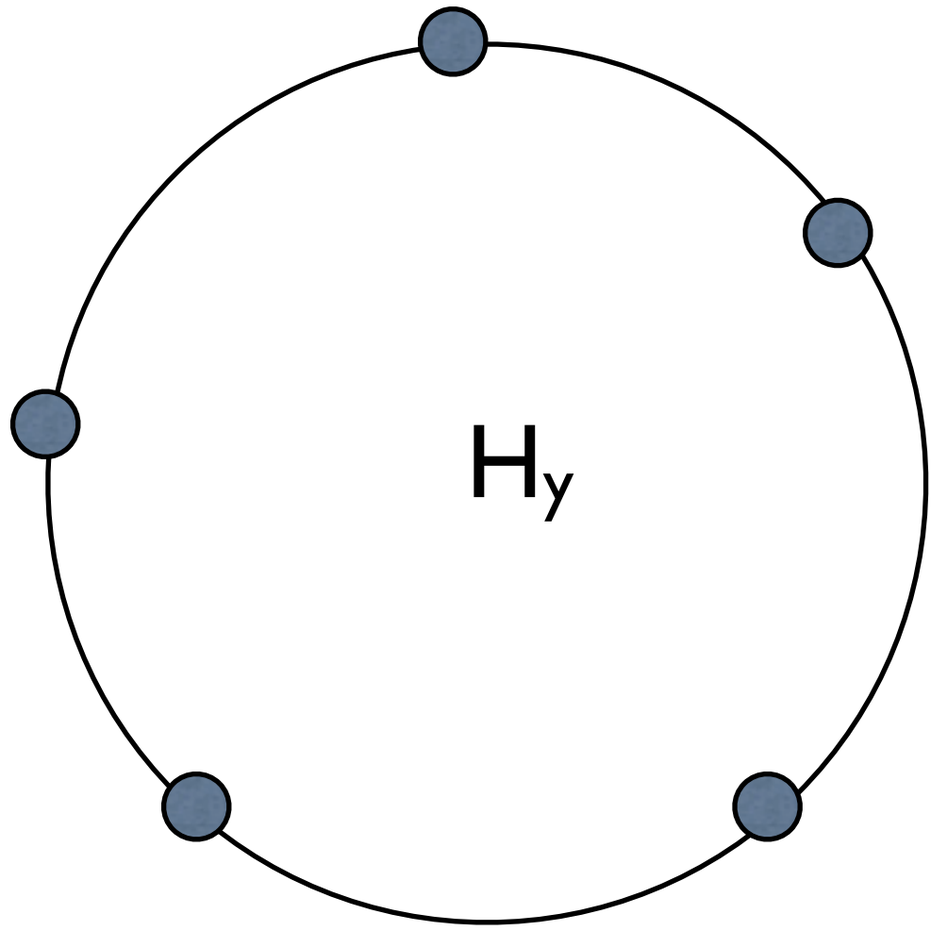
O(n) hops is needed for linear search but we can reduce it to O(log n) hops

Publish/Subscribe Using Mercury

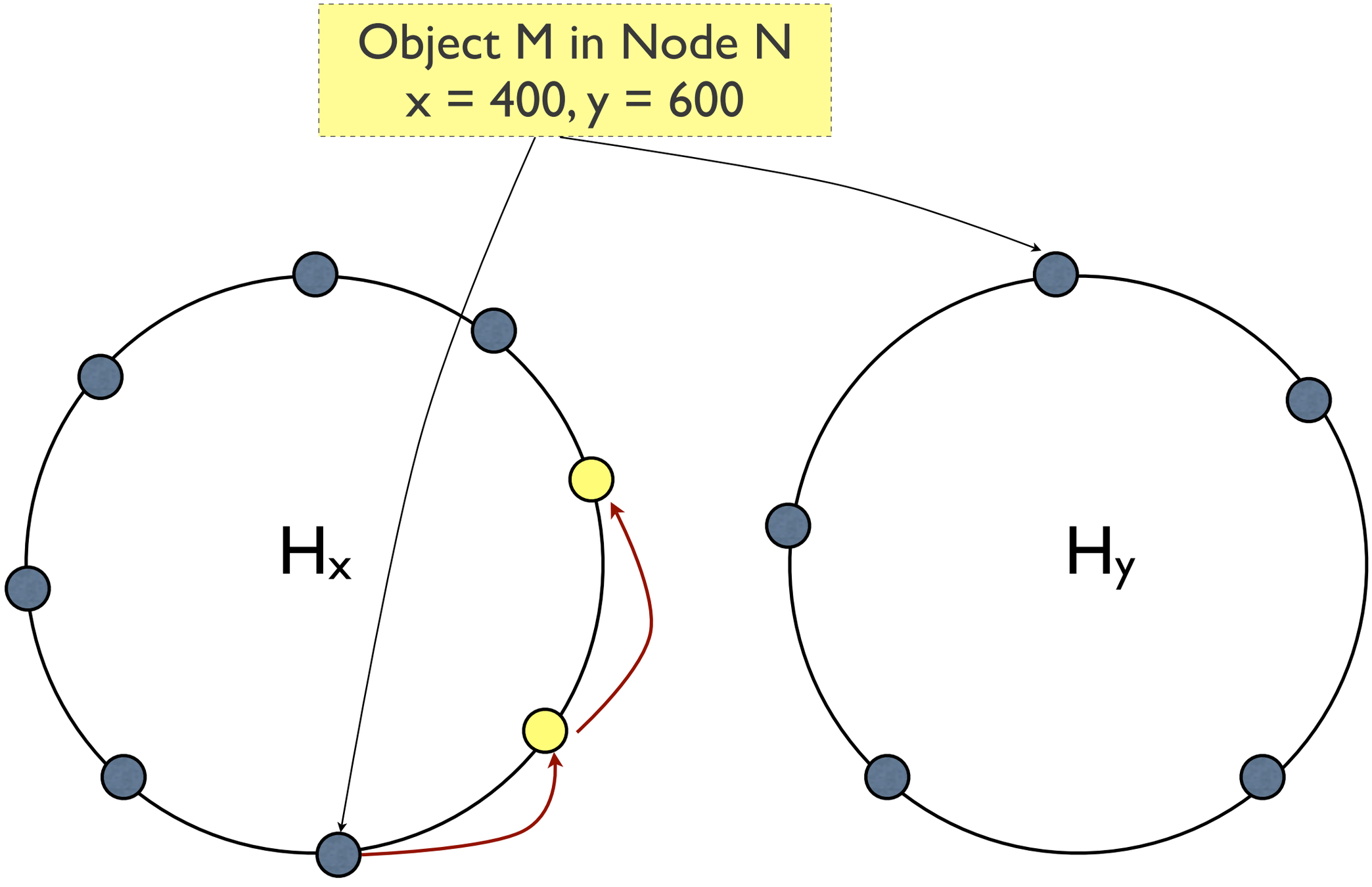
Node A:
100 < x < 500
600 < y < 700

A new subscription is sent to the hub that corresponds to one of its dimension and is stored in nodes that maintain the values overlapped with range.





An update (publication) is sent to all hubs.



Mercury interface:

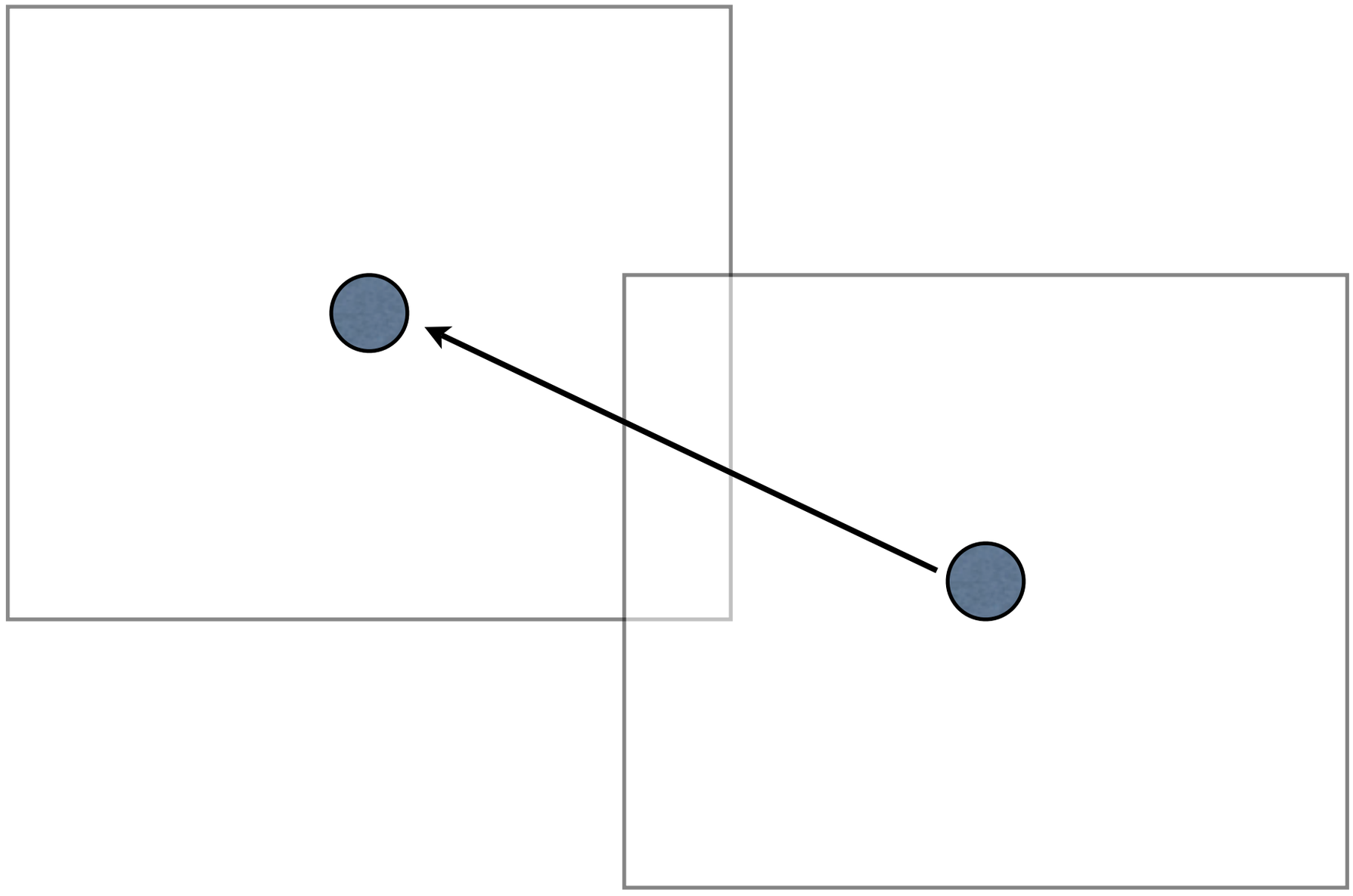
given an AoI, returns the list of objects within the AoI and the nodes that store their primary copy.

The node that store matching publication and subscriptions is responsible for informing the subscribers of new publications and new subscribers of existing publications.

With this list of objects, the player contacts the corresponding nodes to read/write the state of the objects directly.

Latency is a concern only during discovery of new objects. We can further reduce this latency by prefetching.

Let t be the time needed to send query and get reply back from Mercury. A player can predict its position and AoI after time t.



Increase t to be more conservative -- less misses, but more unnecessary object information.

The authors showed that Colyseus is scalable and give latency small enough for Quake II/III.

Recap

Without a trusted central server:

1. how to order events?
2. how to prevent cheat?
3. how to do interest management?
4. who should store the states?

Many interesting proposals, but no perfect solution.

1. Increase message overhead
2. Increase latency
3. No conflict resolution
4. Cheating
5. Robustness is hard

Many tricks we learnt from pure P2P architecture is useful if we have a cluster of servers for games

"P2P among servers"

Part III of CS4344

Hybrid Architecture