Distributed Hash Table

P2P Routing and Searching Algorithms

Ruixuan Li
College of Computer Science, HUST
rxli@public.wh.hb.cn
http://idc.hust.edu.cn/~rxli/

In Courtesy of Xiaodong Zhang, Ohio State Univ

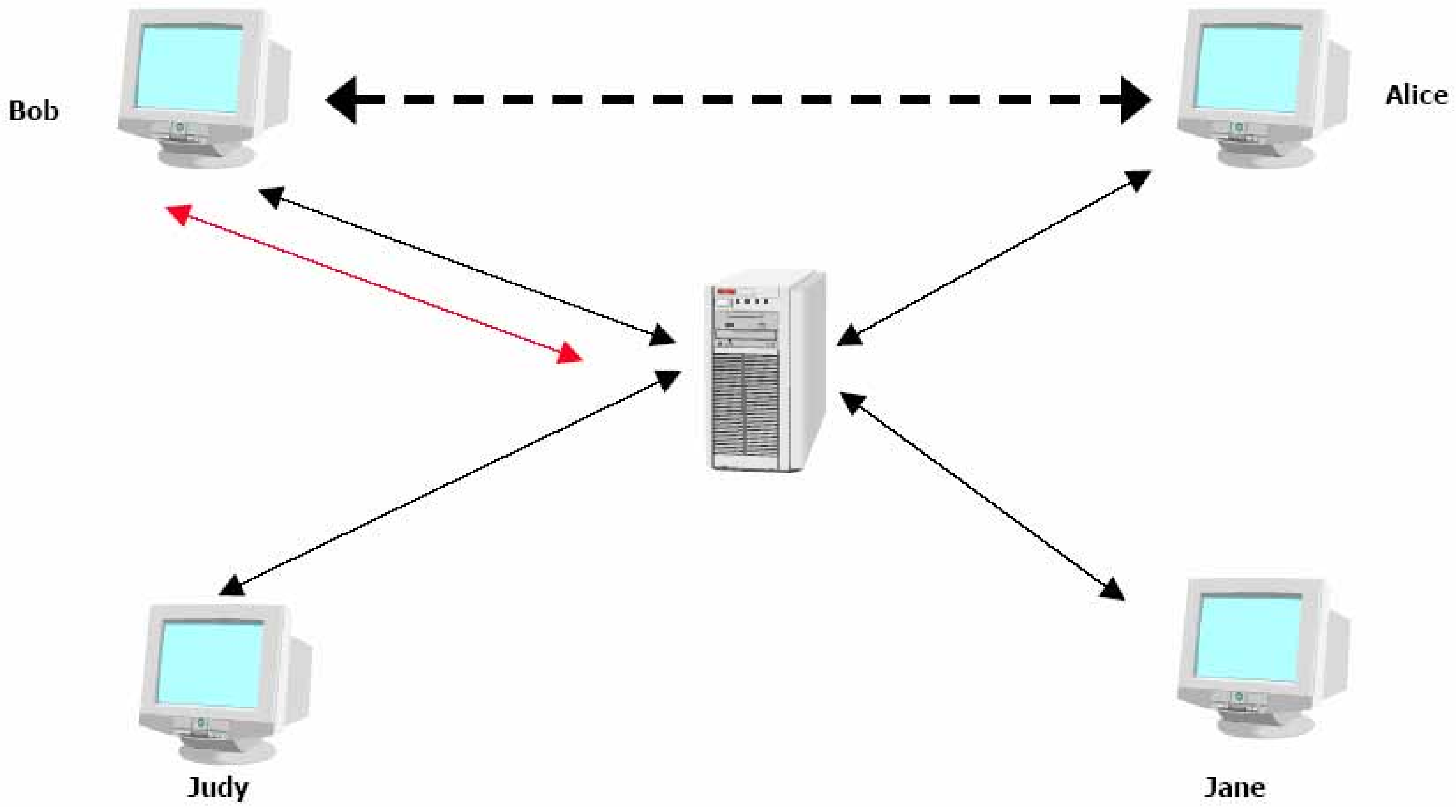
Outline

- Search Approaches in P2P
- Distributed Hash Table
- Case Study: CAN
- Open questions for DHT
- Summary

Search Approaches in P2P

- Centralized
- Flooding
- Document Routing

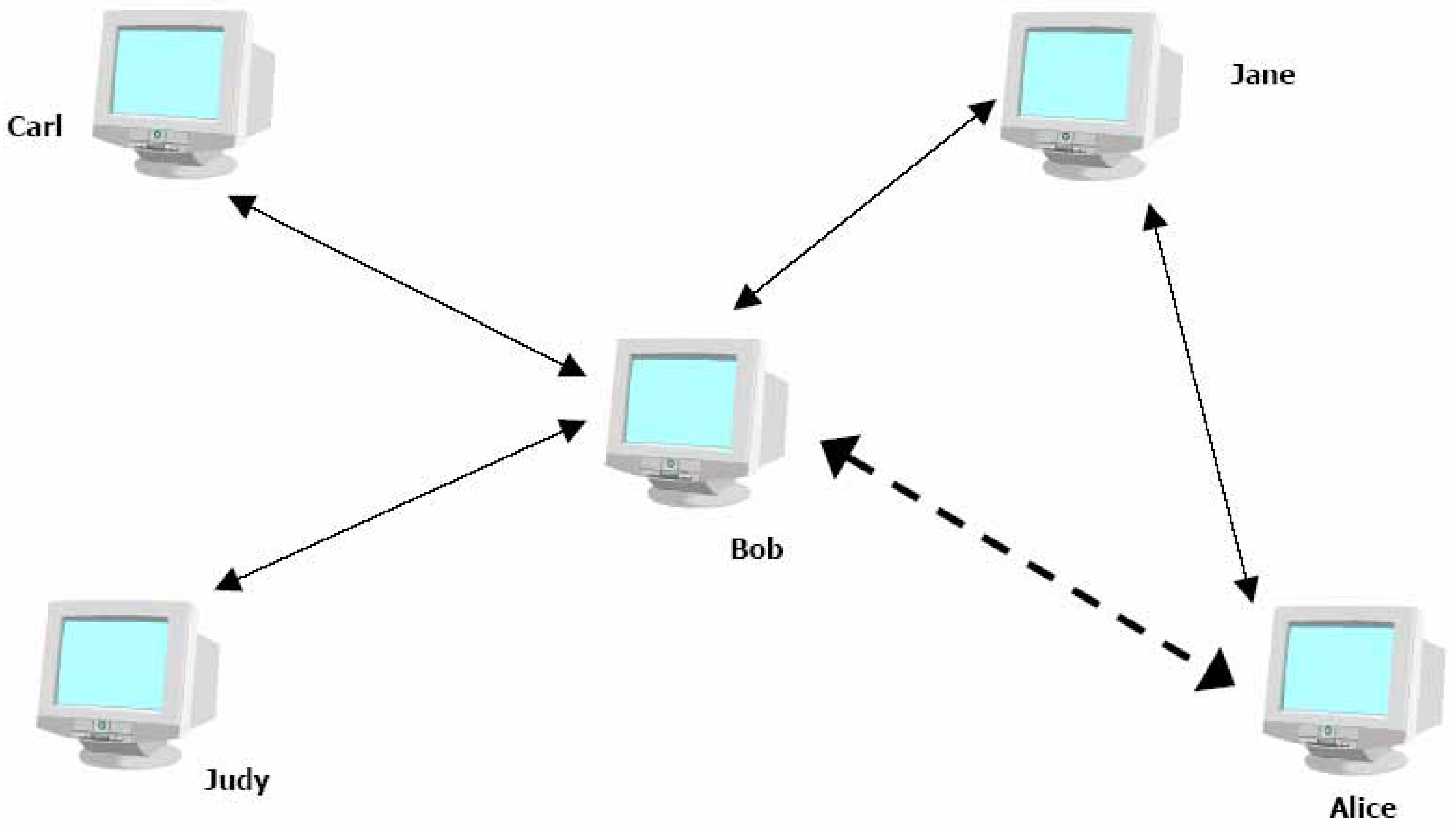
Centralized



Centralized

- **Benefits:**
  - Efficient search
  - Limited bandwidth usage
  - No per-node state
- **Drawbacks:**
  - Central point of failure
  - Limited scale

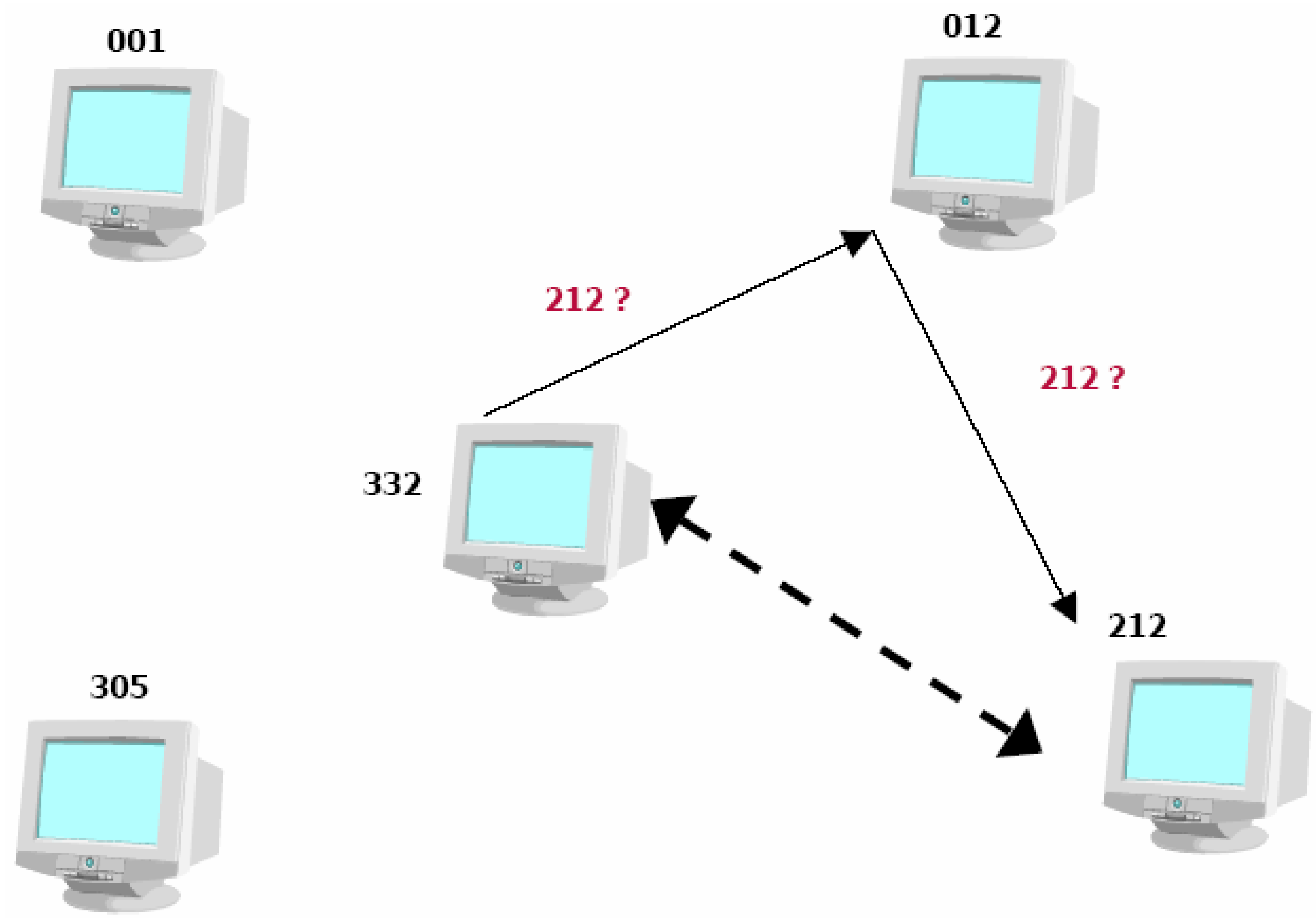
Flooding



Flooding

- **Benefits:**
  - No central point of failure
  - Limited per-node state
- **Drawbacks:**
  - Slow searches
  - Bandwidth intensive

Document Routing



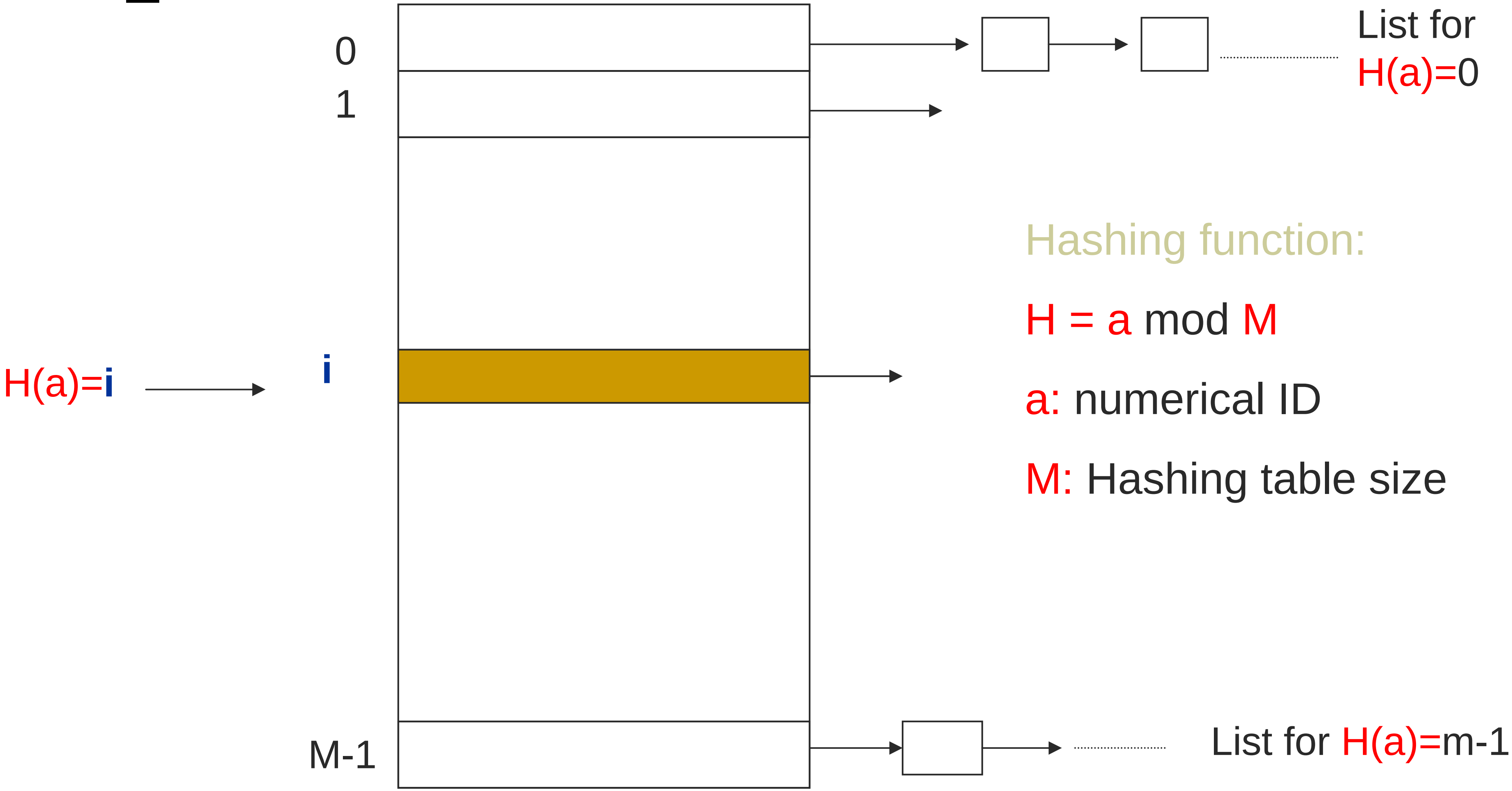
Document Routing

- **Benefits:**
  - More efficient searching
  - Limited per-node state
- **Drawbacks:**
  - Limited fault-tolerance vs redundancy

Document Routing

- Approach:
  - Distributed Hash Table (DHT)

Standard Hashing



Hashing function:

H = a mod M

a: numerical ID

M: Hashing table size

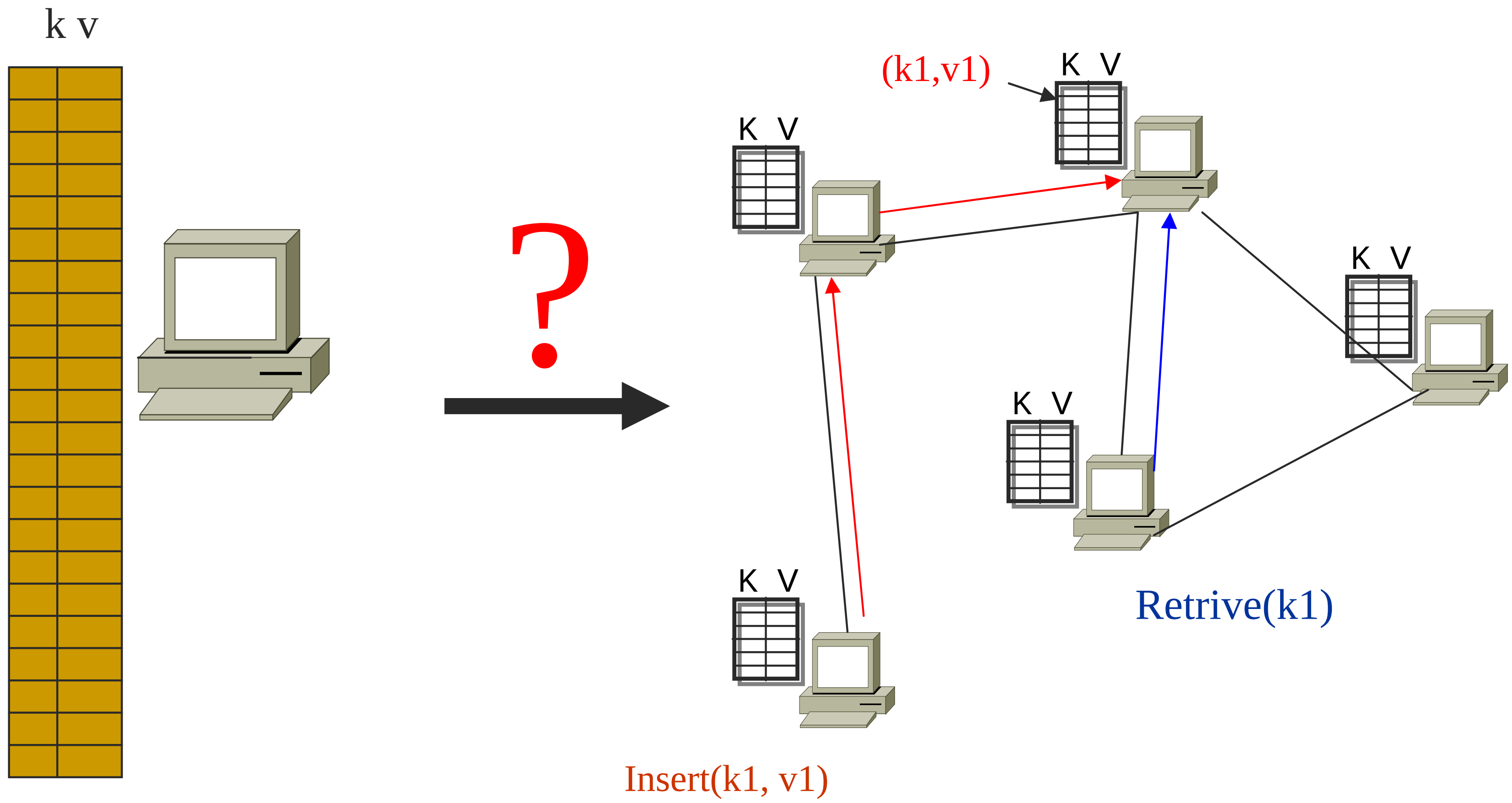
Basic Hashing Operations

- Insert (a, S): insert object a to Set S.
  - Compute h(a);
  - Search the list pointed by Table[h(a)]; if a is not on the list, it is appended in the list.
- Delete (a, S): delete object a from set S.
  - Search the list pointed by Table[h(a)] and delete object a in the list;
- Find (a, S): find object a in Set S.
  - Search the list pointed by Table [h(a)]; if a is on the list, returns its location, otherwise returns Null.

Distributed Hash Table (DHT)

- Problem: given an object stored in a node or multiple nodes, find it.
- The Lookup problem (Find (a, S)): S is distributed and stored in many nodes.
  - Returns the network location of the node currently responsible for the given key.
- Take a 2-d CAN as an example (a Ph.D. dissertation at Berkeley)

From Hash Table to DHT



a. Hash table

b. Distributed hash table

From Hash table to DHT (cont)

"Core" questions when introducing "distributed":

- How to divide a whole hash table to multiple distributed hash tables?
- How to reach the hash table who has the key I want, if I cannot find it from the local hash table?

Requirements:

- Data should be identified using unique numeric keys

  using hash function such as SHA-1 (Secure Hash Algorithm)
- Nodes should be willing to store keys for each other.

Content Addressable Network

- The overlay nodes are built on a 2-D coordinate space.
- Join: a new peer node
  - Chooses a random point P in the 2-D space;
  - Asks a node in P2P to find node n in P;
  - Node n splits the zone into two, assigns ½ to the new nodes;
- Insert: a key is hashed on to a point in the 2-D space, and is stored at the node whose zone contains the point's space.
- Routing Table: each node contains the logic locations of all its neighbors in the 2-D space.

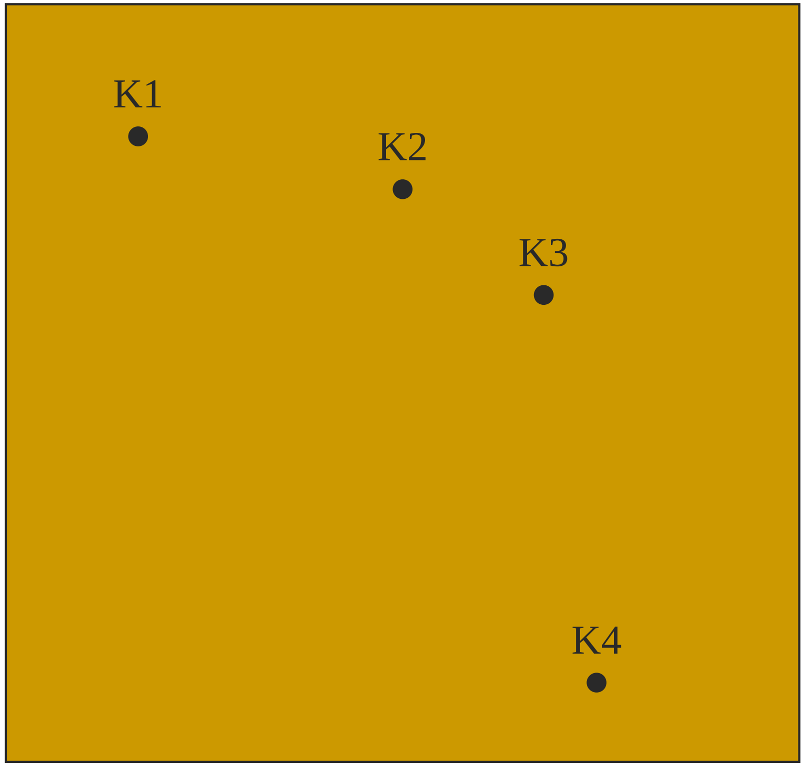
2-D CAN (continued)

- Lookup: after a peer joins, it forwards the request (a hashed location) along a routing path to the node storing the key.
  - a move instruction is made based on the routing table.
- Each node maintains O(d) states, lookup cost is O(dN^(1/d)), where d = dimension, N = # of nodes.

Case Study: CAN

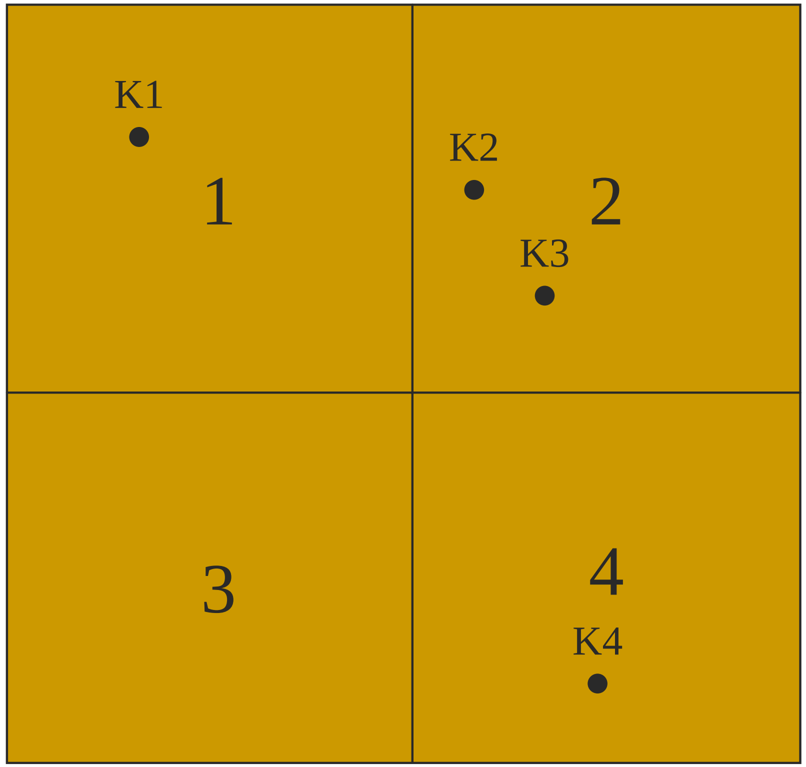
- CAN: Content-Addressable Network
- Basic Data Structure: d-dimensional Cartesian coordinate space
- Every key (k) is mapped to a point (x,y) in the coordinate space

  x = h1(k), y = h2(k);
- The coordinate space = key space



Zone: answer to question 1

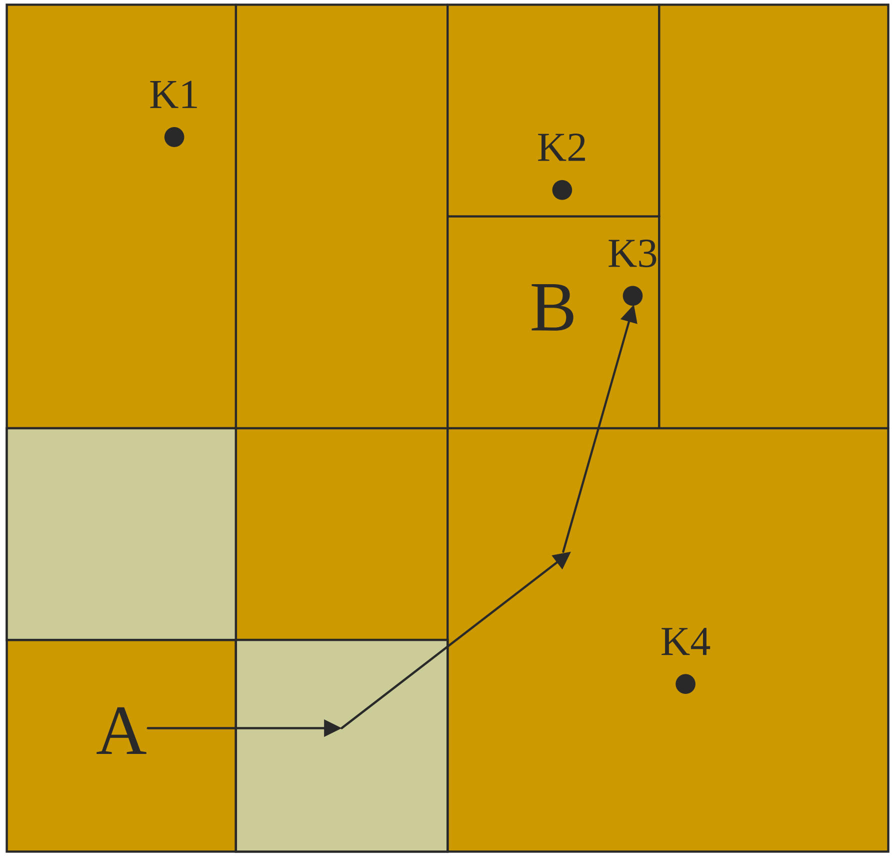
- This coordinate space is partitioned into distinct zones.
- Every node holds a distinct zone
- A node should store all keys that fall into the zone it owns



Routing: answer to question 2

- Every node only maintains the states of its neighbors
- Forward lookup request to a neighbor closer to the key in the coordinate space

Node A wants to lookup k3



Insertion & Retrieval in CAN

1. Node A inserts (k3, v3)

2. x3=h1(k3), y3=h2(k3)

3. Route Insertion request to (x3,y3)

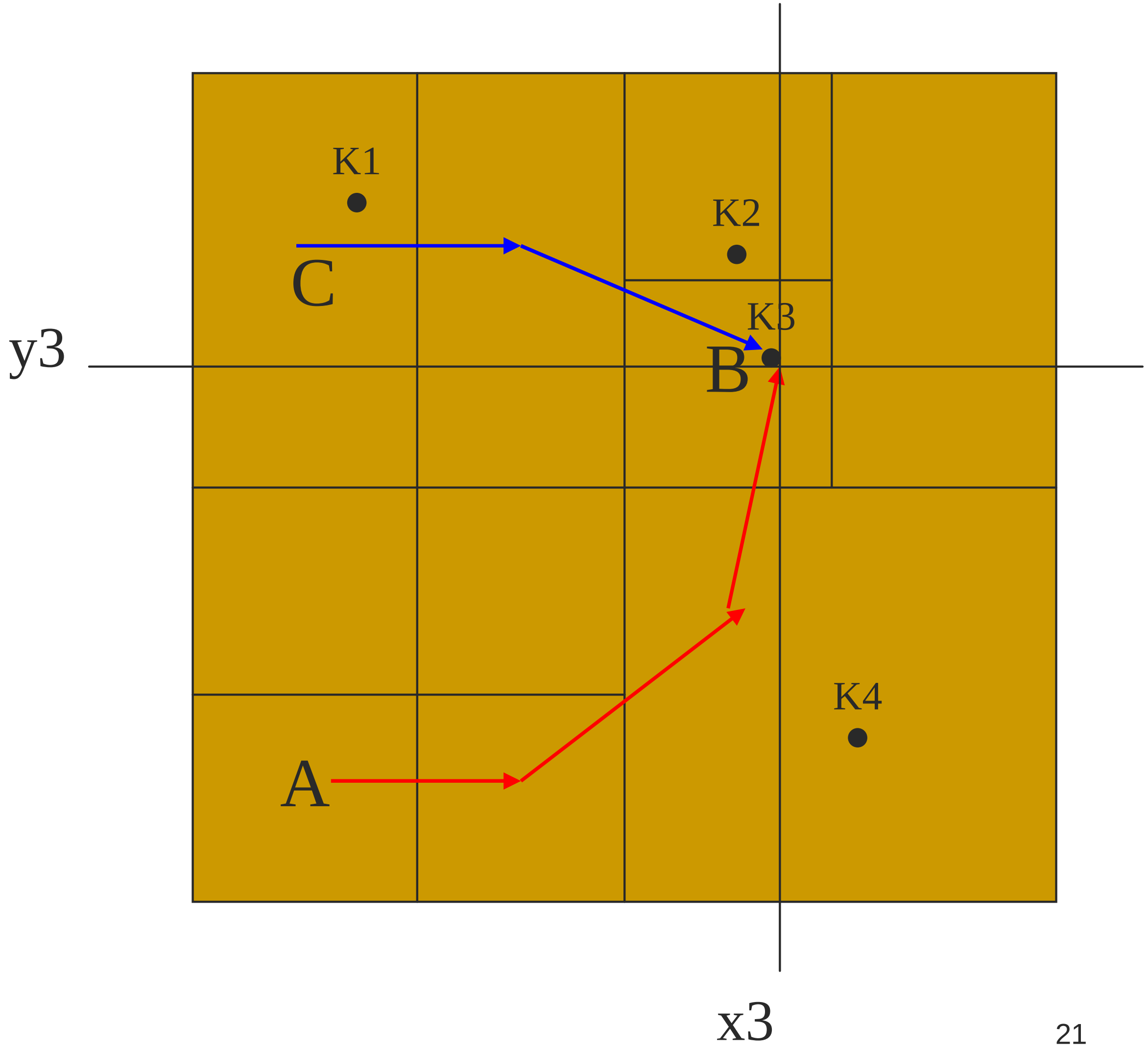
4. (x3, y3) is in the zone of node B, so node B should store (k3,v3) in its hash table

5. Node C retrieves k3

6. Computes x3, y3 like A does

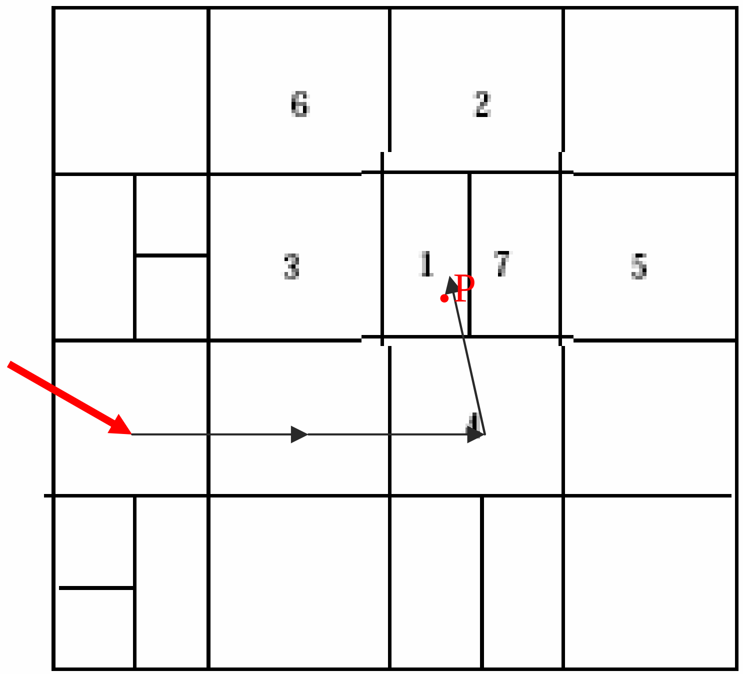
7. Route lookup request to (x3,y3)

8. Node B receives lookup request, and retrieves (k3, v3) from its hash table



How does a new node join the CAN?

- Bootstrap
  - The new node find a node already in the CAN
- Finding a zone
  - Find a node randomly whose zone will be split
    - JOIN request message
    - Splitting
    - Hand over part of (key, value) pairs
- Joining the routing
  - The neighbors of the split zone is notified so that routing can include the new node

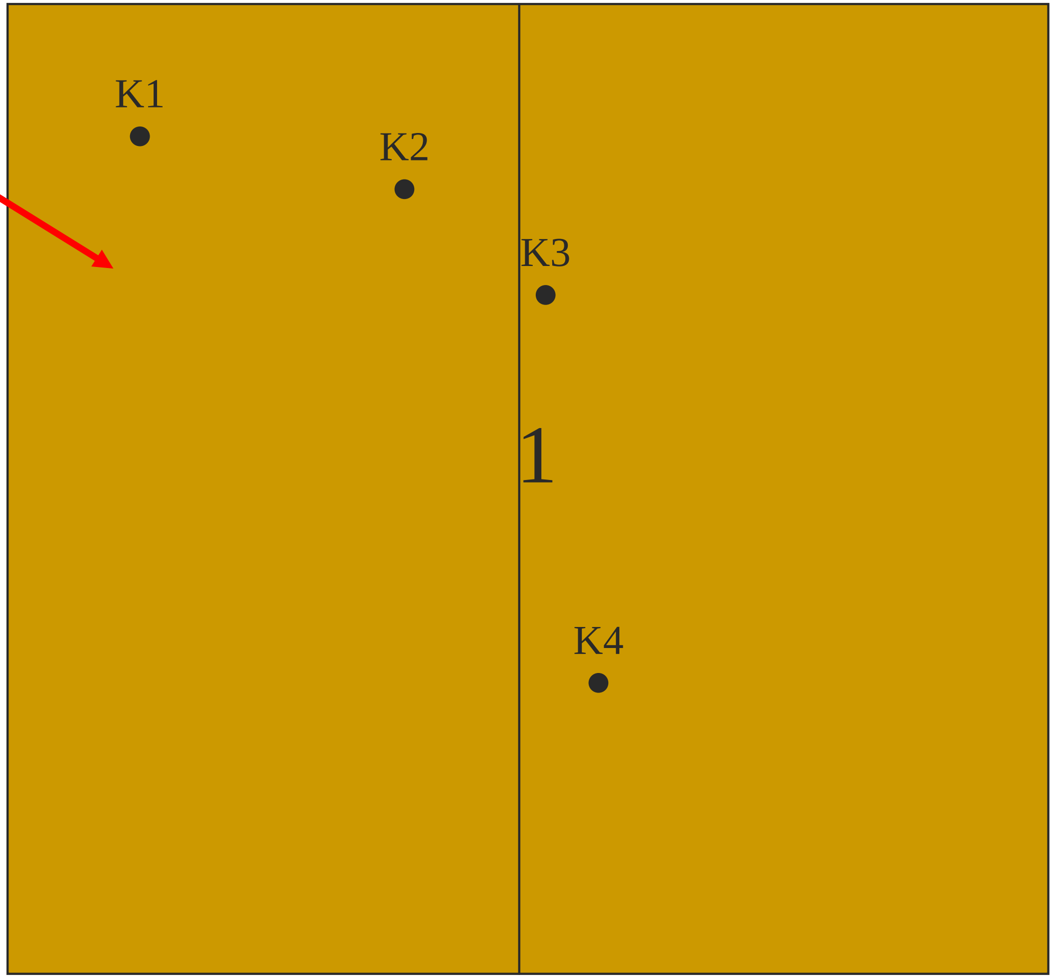


1’s coordinate neighbor set = {2, 3, 4, 7}

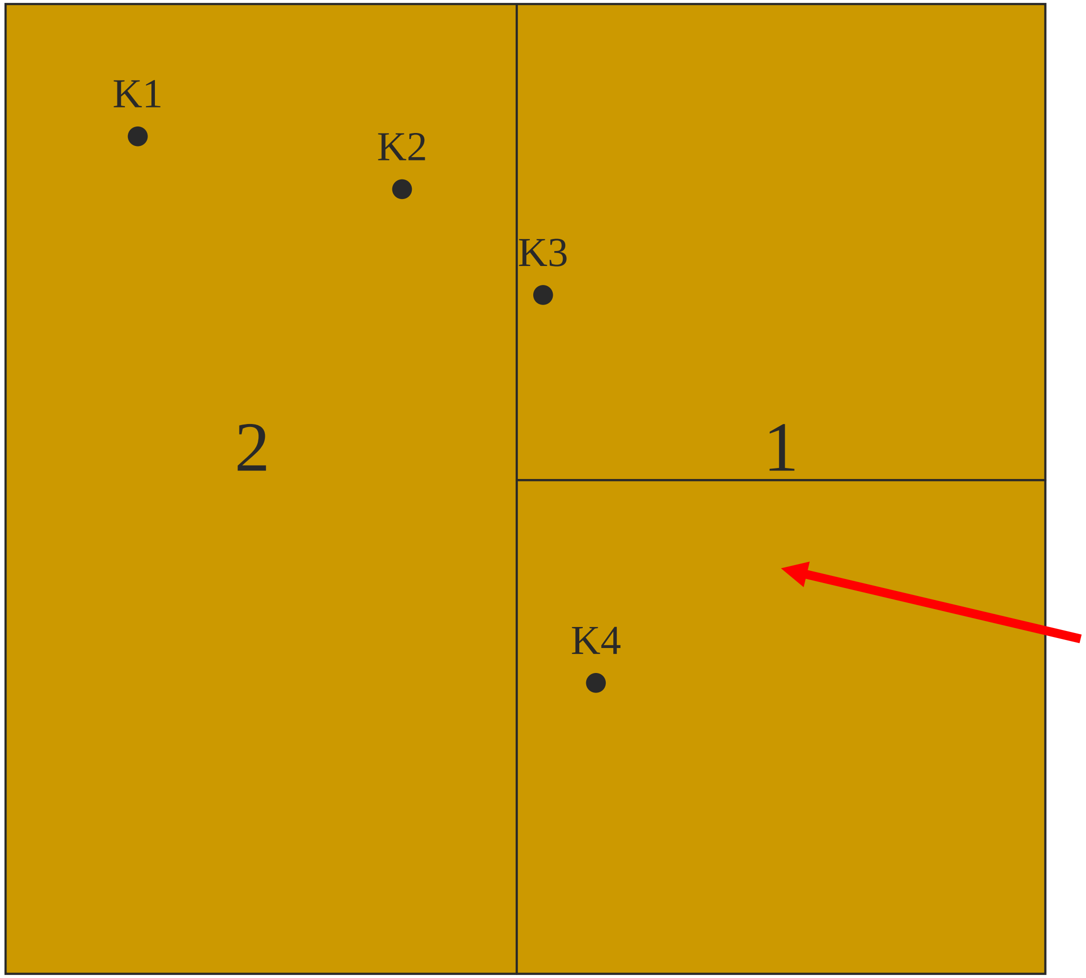
7’s coordinate neighbor set = {1, 2, 4, 5}

One more example

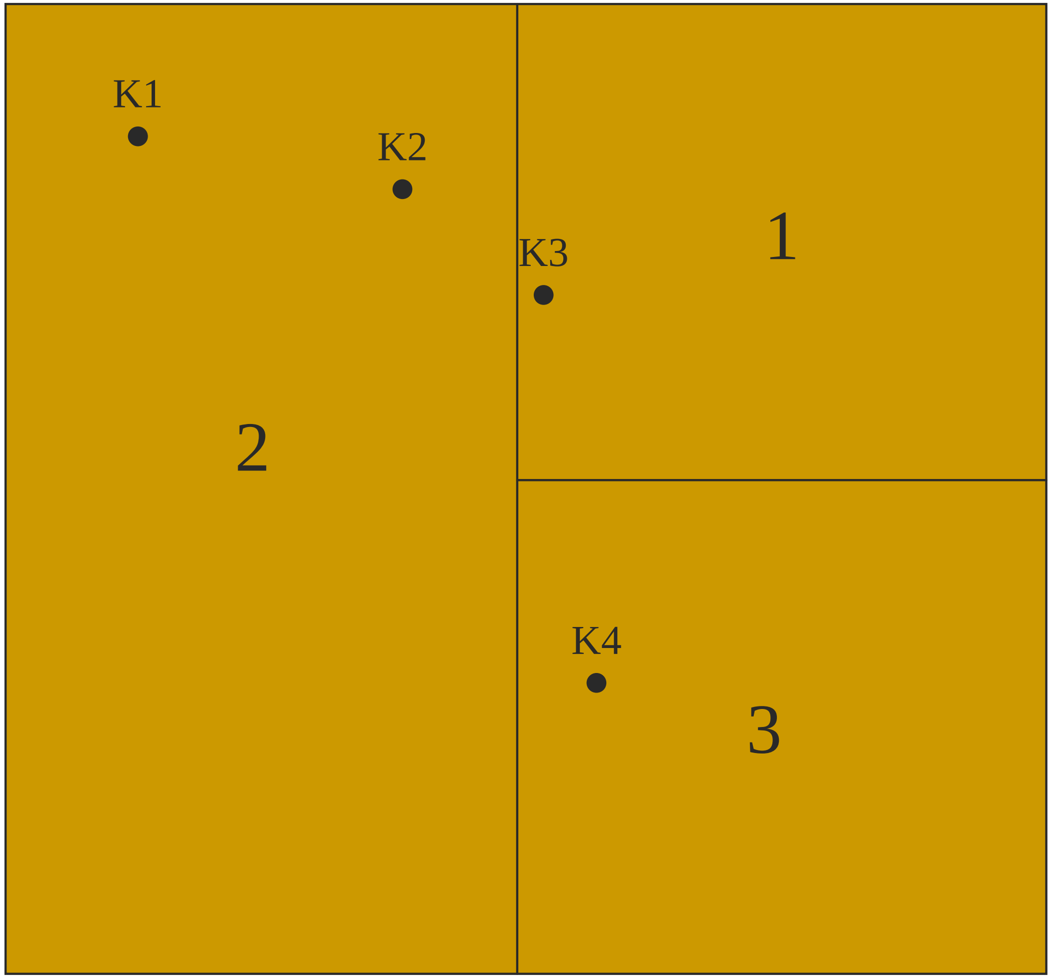
pick a random point in space



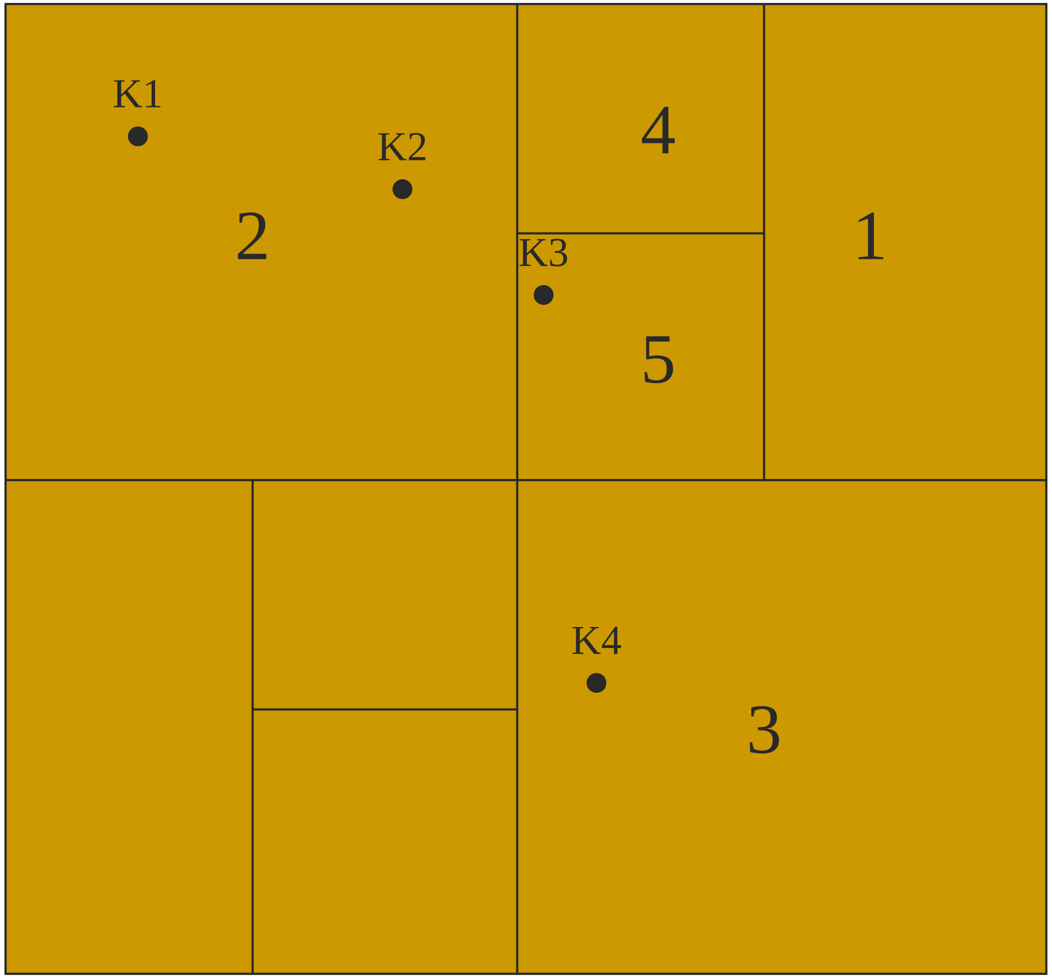
One more example



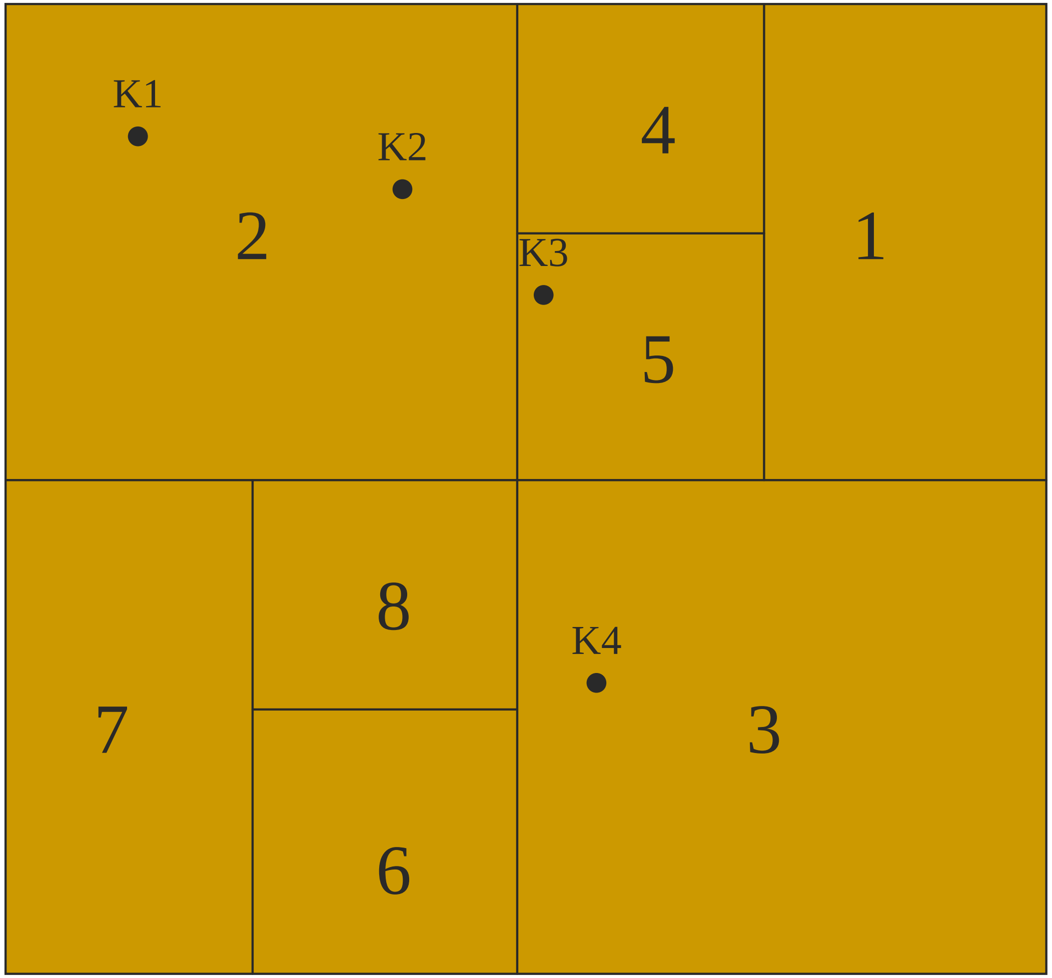
One more example



One more example

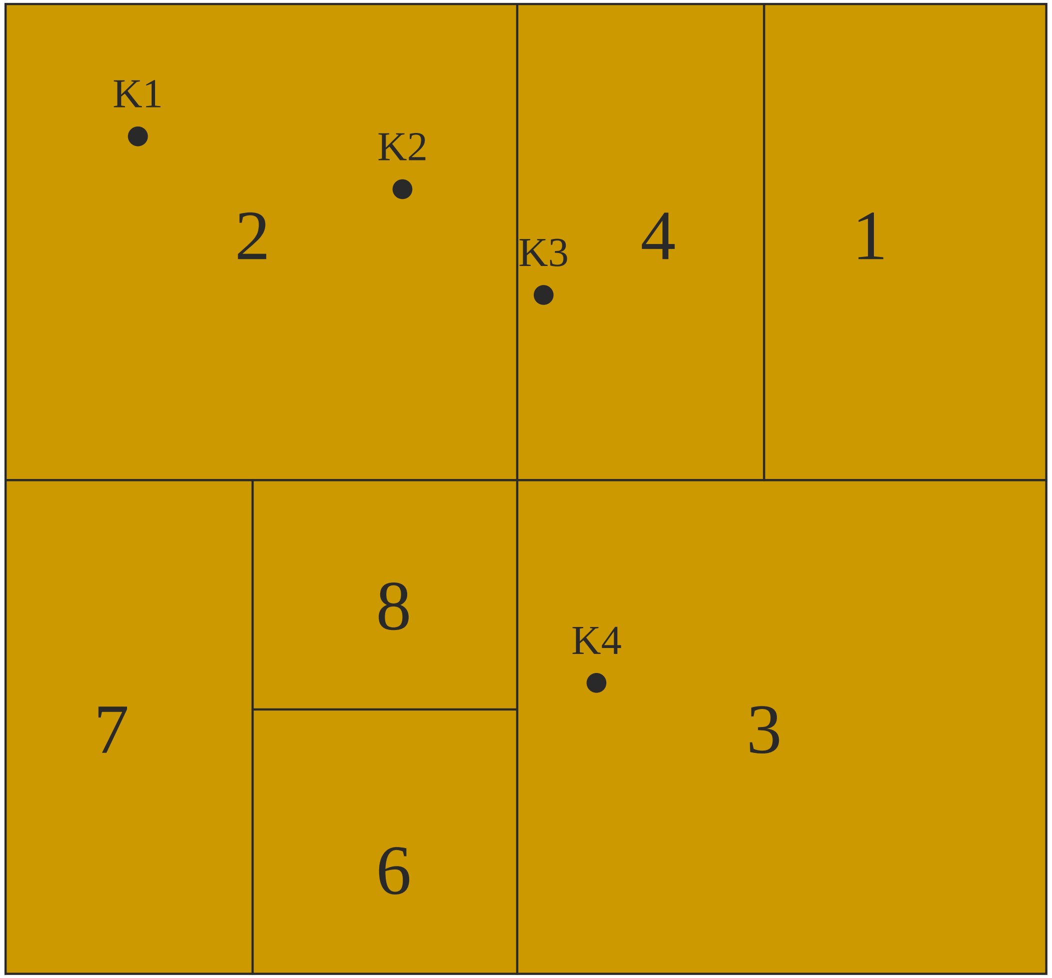


How does a node depart?



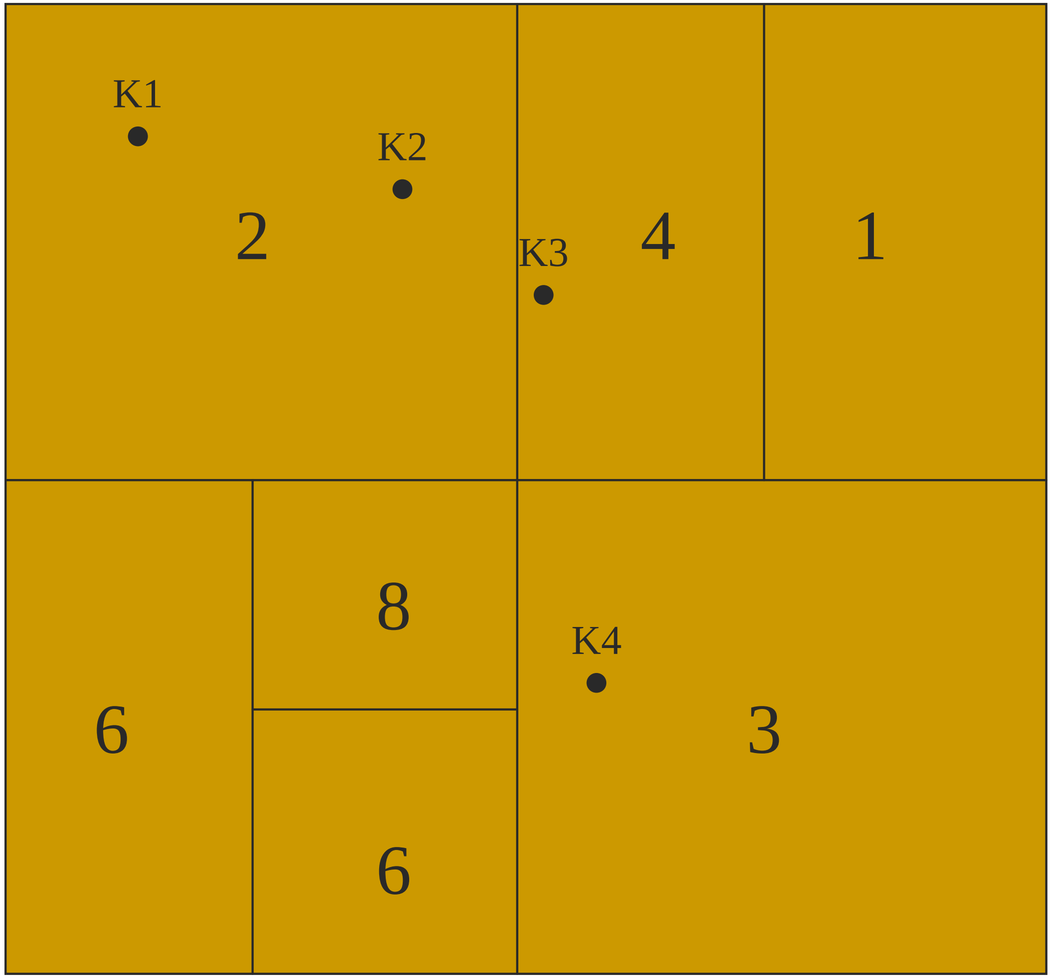
Node 5 is leaving

How does a node depart?



Node 7 is leaving

How does a node depart?



CAN: node failures

- Detect failures
  – Send periodic update message to neighbors

- Need to repair the space
  – recover database
    • soft-state updates
    • use replication, rebuild database from replicas
  – repair routing
    • takeover algorithm

CAN: takeover algorithm

- Simple failures
  - know your neighbor's neighbors
  - when a node fails, one of its neighbors takes over its zone
- More complex failure modes
  - simultaneous failure of multiple adjacent nodes
  - scoped flooding to discover neighbors
  - hopefully, a rare event

Why Unstructured P2P Co-exists?

- When peers are highly dynamic and transient, maintenance and updating of DHT will be too expensive to afford. Little effect to U-P2P.
- DHT only provides information of ”needles”, not ”hails”, which can only provided by U-P2P.
- DHT only provides ”key word” search. The search in U-P2P can be very vague, leave a large space for a wide range of development, such as semantic Web.

Operation Cost of CAN

- States of neighbors
  - $2d$
- Average path length
  - $(d/4)(n^{1/d})$

Note that other algorithms like CHORD, TAPSTRY and PASTRY route in O(log n) hops with each node maintaining O(log n) neighbors.

If we select d=(log n)/2, we could achieve the same scaling properties.

Design Improvements

Goals:

- Reduce the latency of CAN routing
- Improve CAN robustness in routing and data availability
- Load balancing

Techniques:

- Multi-dimensioned coordinate spaces
- Realities: multiple coordinate spaces
- Better CAN routing metrics
- Overloading coordinate zones
- Multiple hash functions
- Topologically-sensitive construction of the CAN overlay network
- More Uniform Partitioning
- Caching and Replication for “hot spot”

Caching and Replication

- **Caching:**
  - Cache the data keys it recently accessed
- **Replication:**
  - Overloaded node can replicate the data key at its neighbors

Open questions for DHT

- Operation costs
  - Path lengths:O(log$n$) vs. O($dn^{1/d}$) hops   (Others vs. CAN)
  - Neighbors : O(log$n$) vs. O(d) (Others vs. CAN)

    Can one achieve O(log$n$) path lengths (or better) with O(1) neighbors? (Answer: Koorde)
- Fault tolerance and concurrent changes
  - high cost for simultaneous failures
- Proximity routing
  - More efficient algorithm?
- Security
  - Malicious nodes and false routes
- Indexing and keyword search

Discussion: merits & limits of DHT

Merits:

- Decentralized management:

  relieve managing burden

  avoid a single point of failure
- A common interface:

  make implementation of distributed apps much easier
- Scalability:

  lookup cost: $O(\log N)$, $O(dN^{1/d})$
- Fault tolerance:

  routing and data availability

Limits:

- How to implement keyword lookup based on DHT?
- Requirements on participants: memory and storage size, CPU speed
- Incompatibility between DHTs
- Your opinions…

Bibliography

- Sylvia Ratnasamy, Paul Francis, Mark Handley, and Richard Karp, A Scalable Content-Addressable Network, ACM SIGCOMM 2001
- Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001
- Antony Rowstron and Peter Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, Middleware 2001
- Brad Karp, Sylvia Ratnasamy, Sean Rhea, and Scott Shenker. *Spurring Adoption of DHTs with OpenHash, a Public DHT Service*, IPTPS 2004
- “Koorde: A simple degree-optimal distributed hash table”

THANK YOU !!!

For not falling asleep : -)