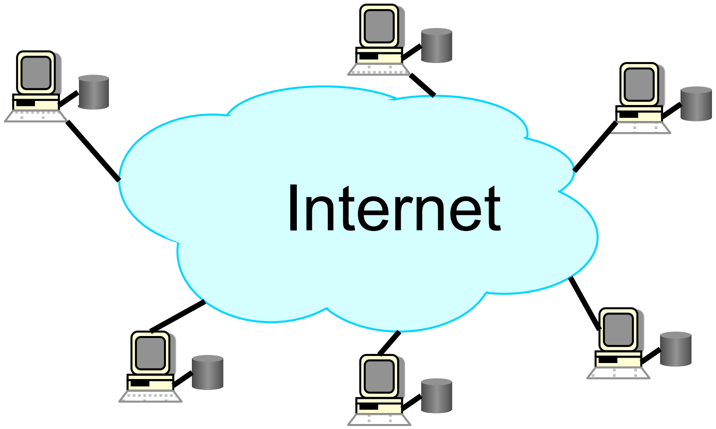
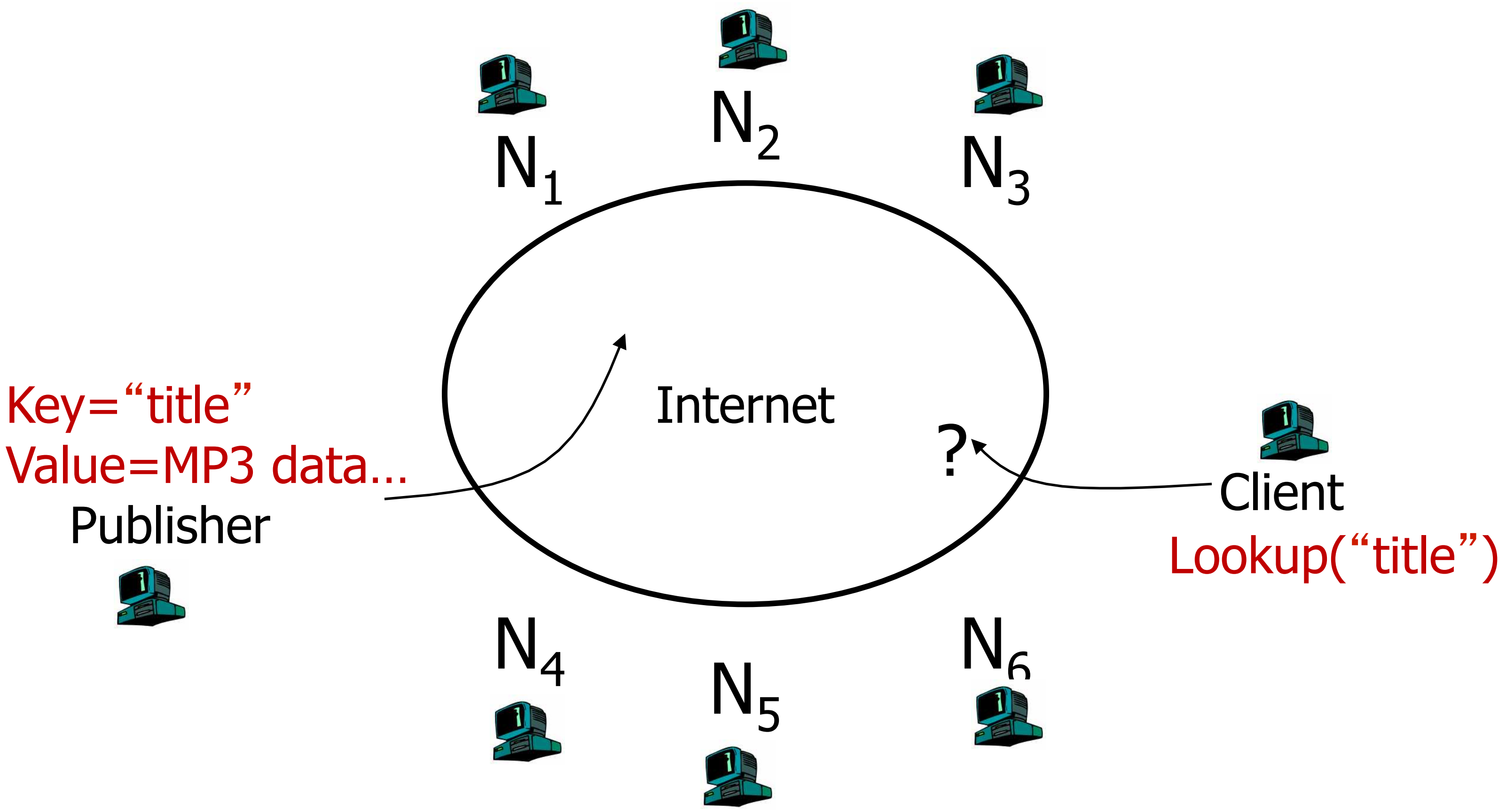
Network Applications: Structured P2P Applications

Recap: Objectives of P2P

- Bypass DNS to locate resources!
  - The lookup problem
- Share the storage *and* bandwidth of individual users
  - The bandwidth sharing problem



Recap: The Lookup Problem



find where a particular document is stored

Outline

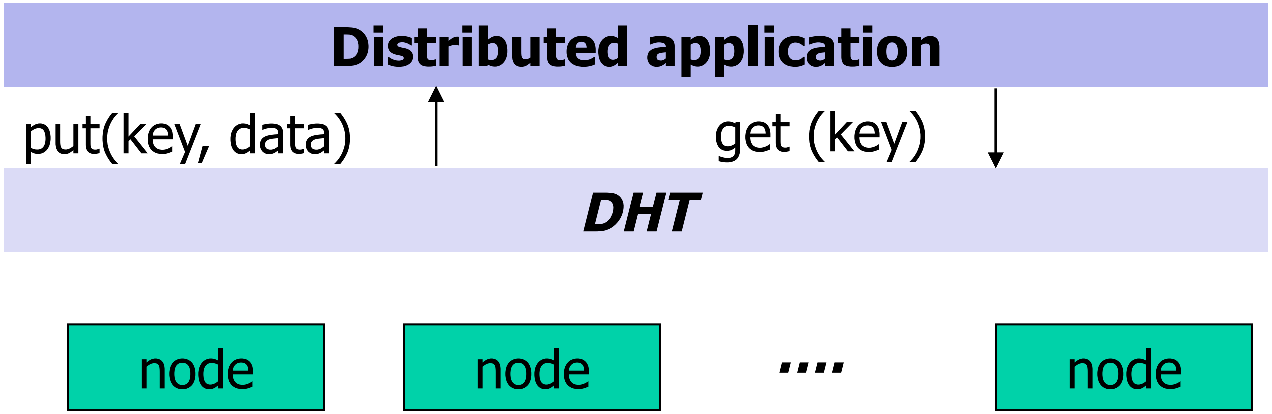
- Recap
- *P2P*
  - *The lookup problem*
    - Napster (central query server)
    - Gnutella (decentralized, flooding)
    - *Freenet*

Distributed Hash Tables (DHT): History

- In 2000-2001, academic researchers jumped on to the P2P bandwagon
- Motivation:
  - Frustrated by popularity of all these “half-baked” P2P apps. We can do better! (so they said)
  - Guaranteed lookup success for data in system
  - Provable bounds on search time
  - Provable scalability to millions of node
- Hot topic in networking ever since

DHT: Overview

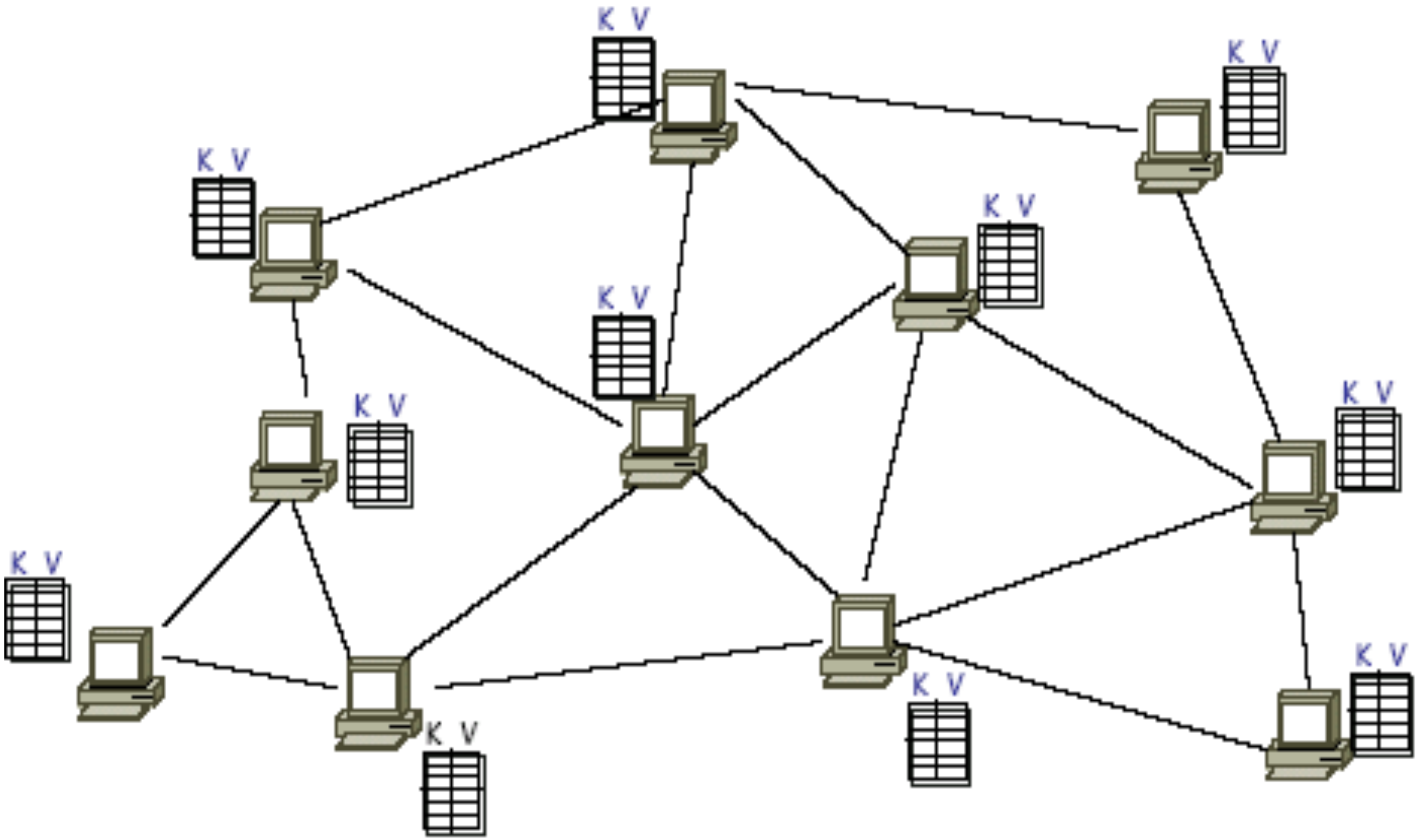
- Abstraction: a distributed "hash-table" (DHT) data structure
  - put(key, value) and get(key) → value
  - DHT imposes no structure/meaning on keys
  - One can build complex data structures using DHT
- Implementation:
  - Nodes in system form an interconnection network: ring, zone, tree, hypercube, butterfly network, ...



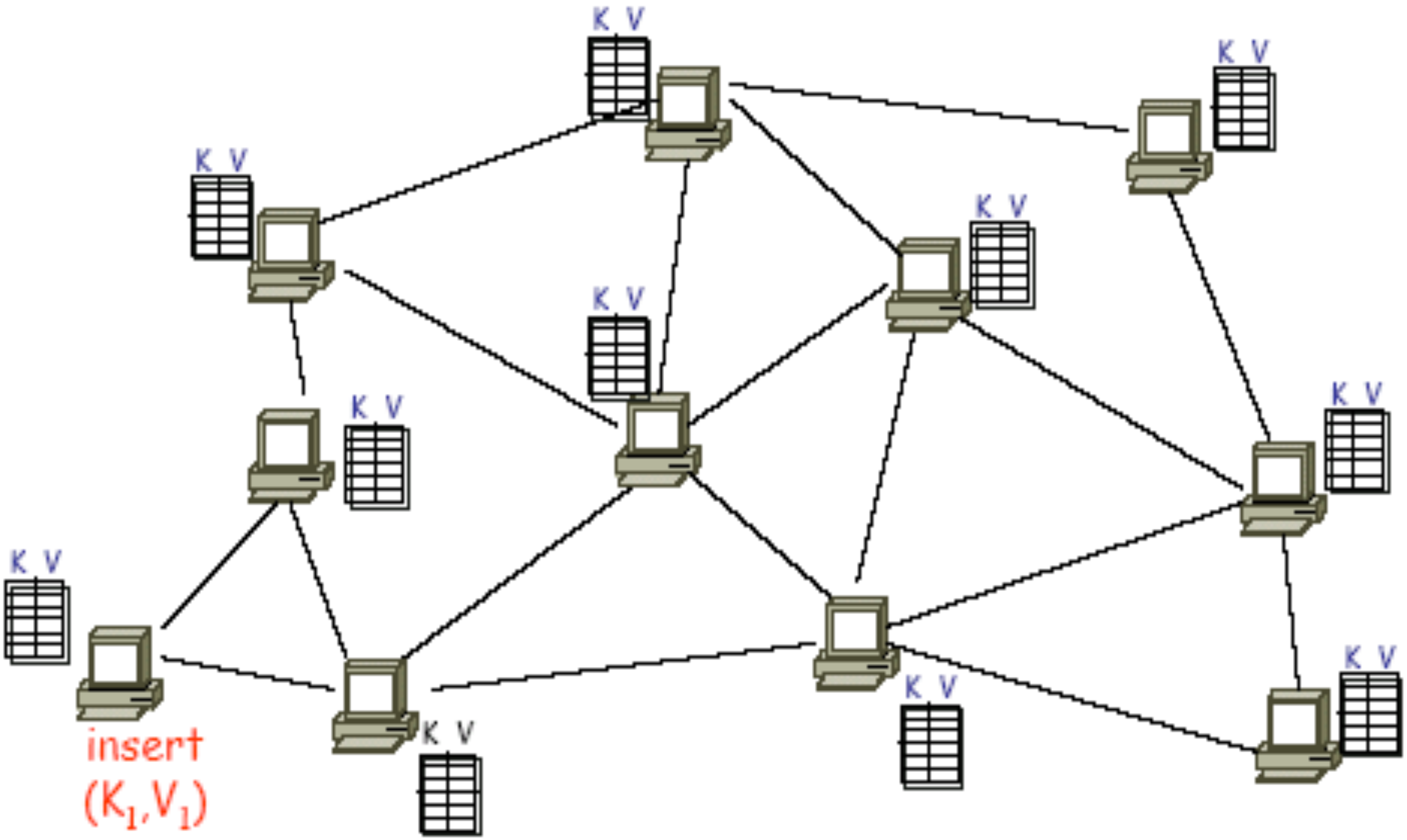
DHT Applications

- File sharing and backup [CFS, Ivy, OceanStore, PAST, Pastiche …]
- Web cache and replica [Squirrel, Croquet Media Player]
- Censor-resistant stores [Eternity]
- DB query and indexing [PIER, Place Lab, VPN Index]
- Event notification [Scribe]
- Naming systems [ChordDNS, Twine, INS, HIP]
- Communication primitives [I3, …]
- Host mobility [DTN Tetherless Architecture]

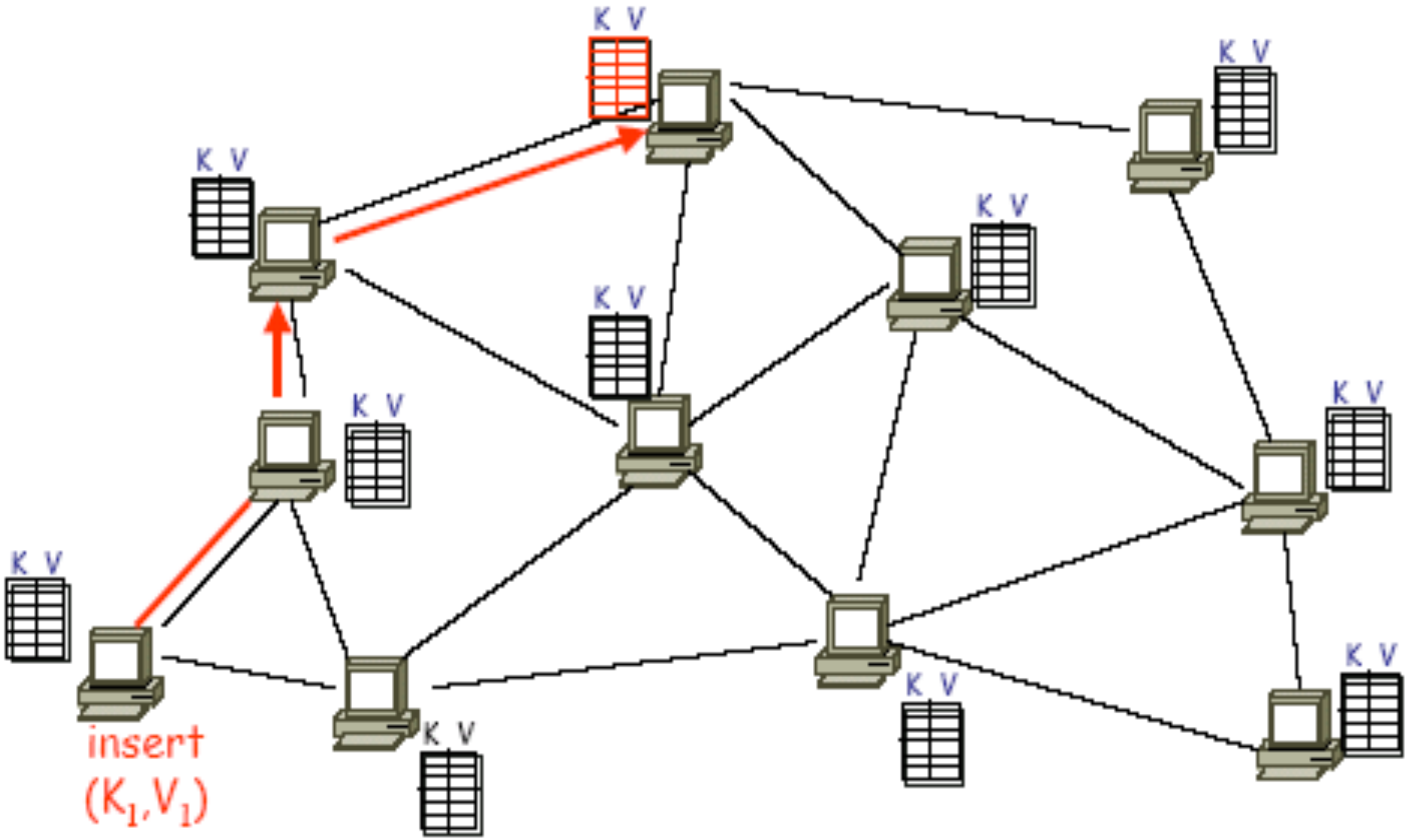
DHT: Basic Idea



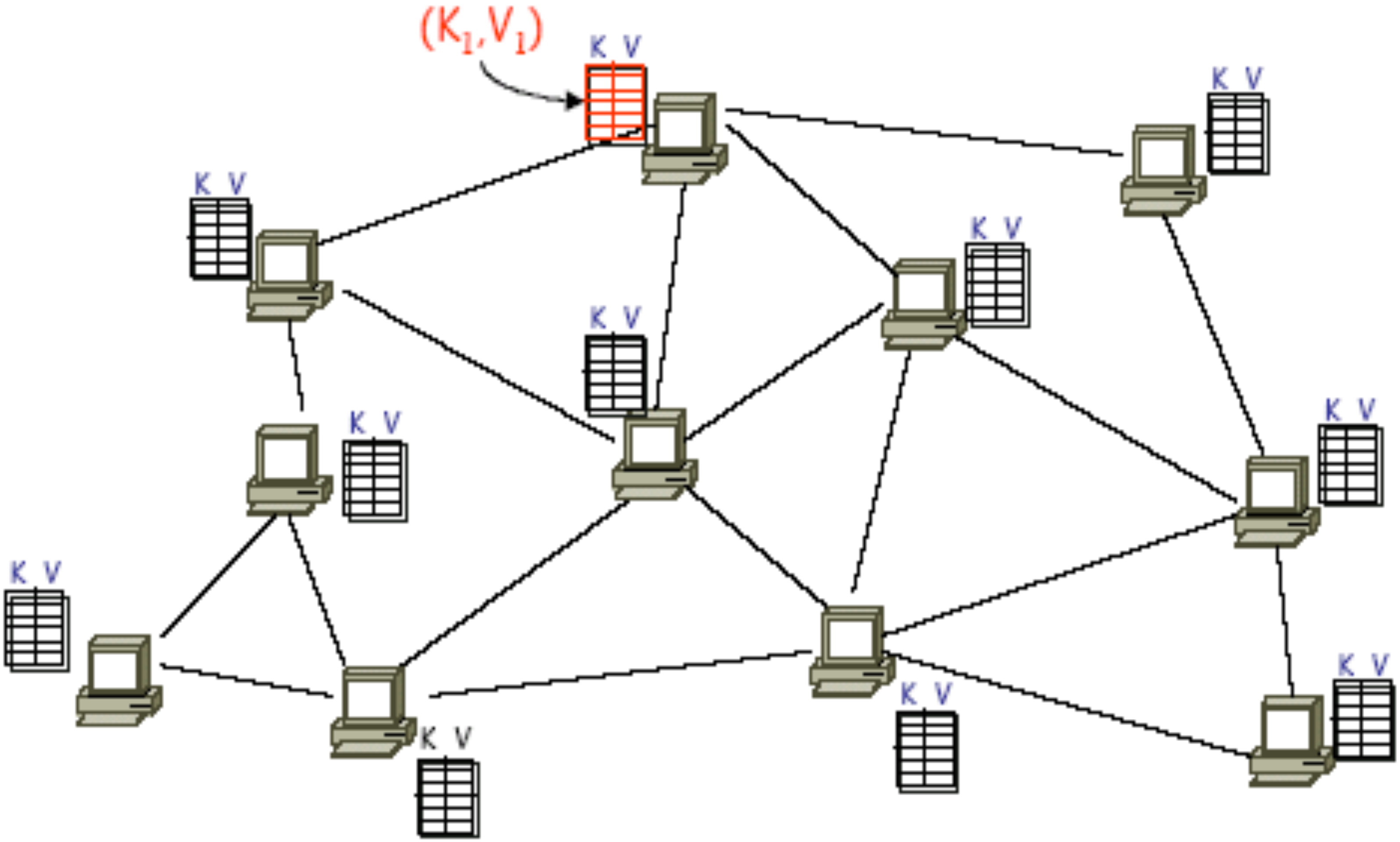
DHT: Basic Idea (2)



DHT: Basic Idea (3)

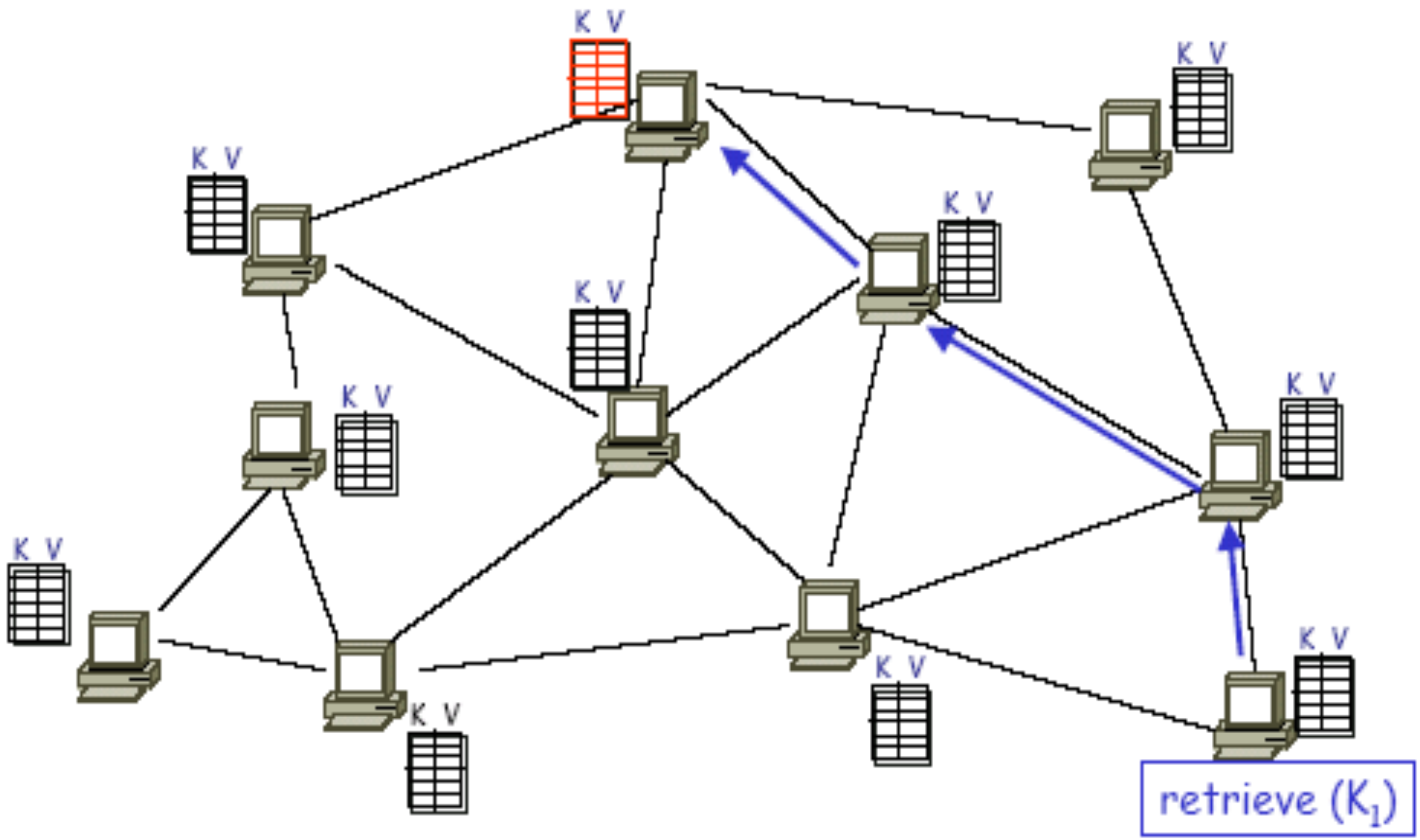


DHT: Basic Idea (4)



DHT: Basic Idea (5)

Two questions:
- how are the nodes connected?
- how the key space is partitioned?



Outline

- Admin. and review
- *P2P*
  - *the lookup problem*
    - Napster (central query server; distributed data server)
    - Gnutella (decentralized, flooding)
    - Freenet (search by routing)
    - *Chord*

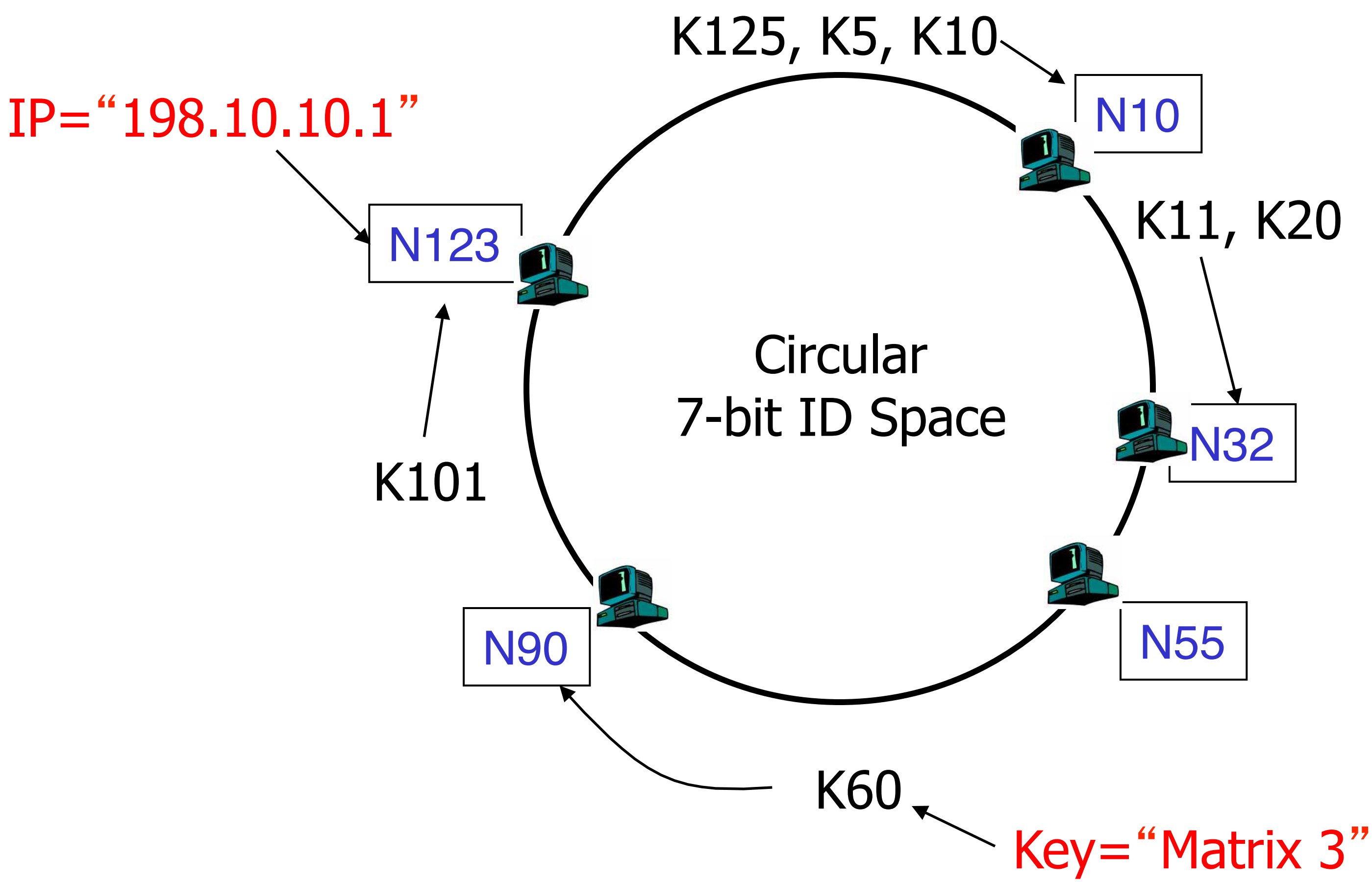
Chord

- Provides lookup service:
  - Lookup(key) → IP address
  - Chord does not store the data; after lookup, a node queries the IP address to store or retrieve data

- m bit identifier space for both keys and nodes
  - Key identifier = SHA-1(key), where SHA-1() is a popular hash function,

    Key="Matrix3" → ID=60
  - Node identifier = SHA-1(IP address)
    - IP="198.10.10.1" → ID=123

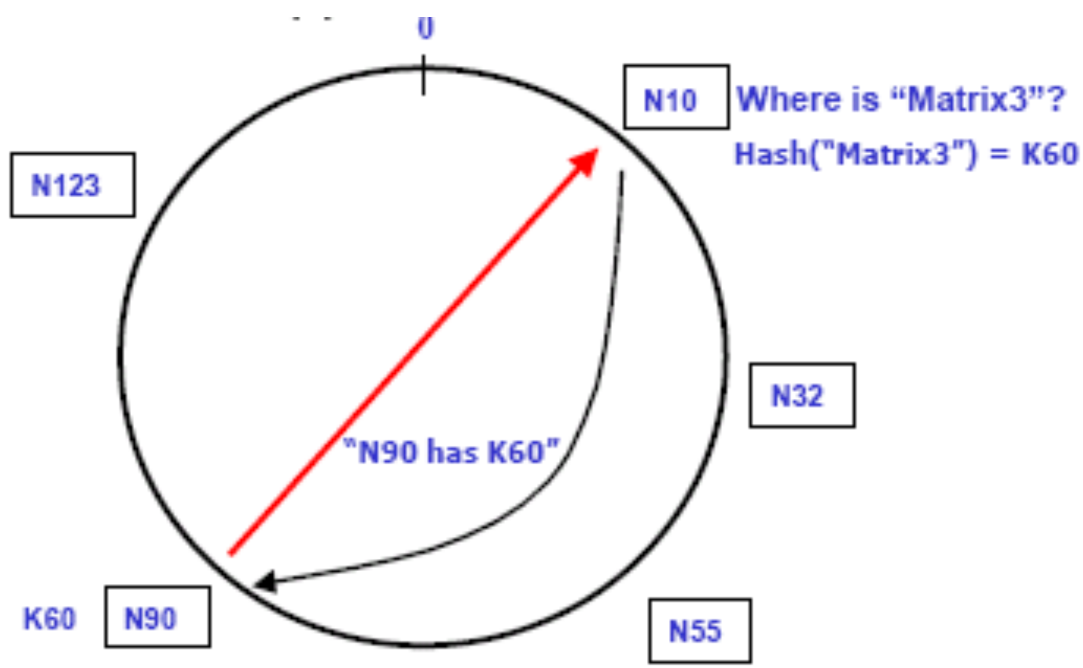
Chord: Storage using a Ring



- A key is stored at its successor: node with next higher ID

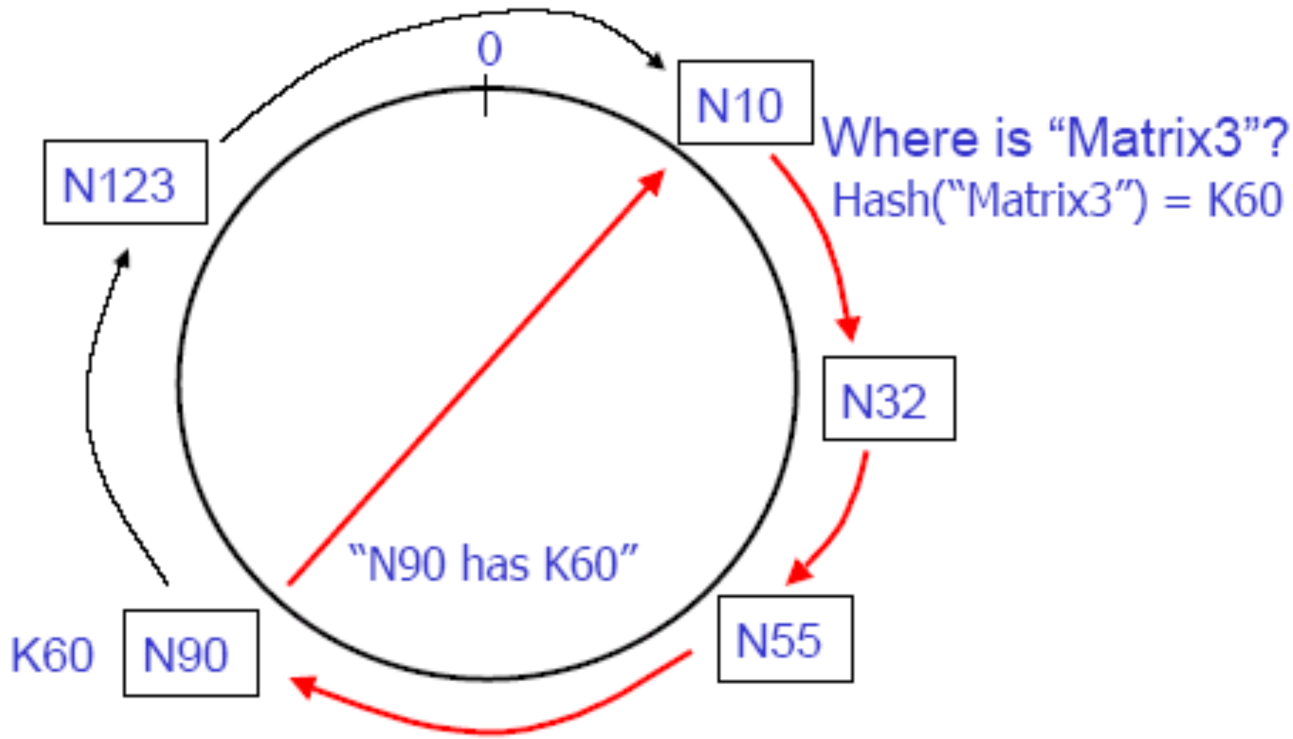
How to Search: One Extreme

- Every node knows of every other node
- Routing tables are large O(N)
- Lookups are fast O(1)



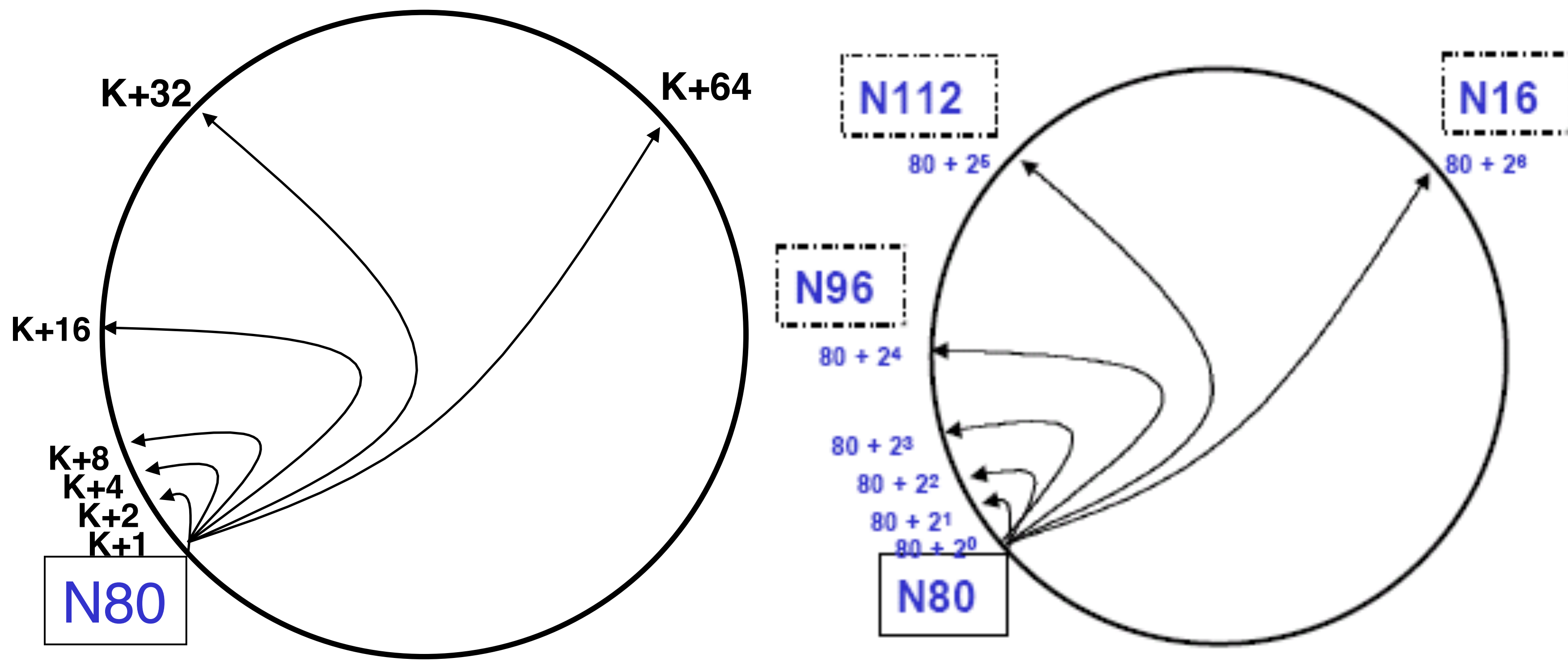
How to Search: the Other Extreme

- Every node knows its successor in the ring
- Requires O(N) lookups



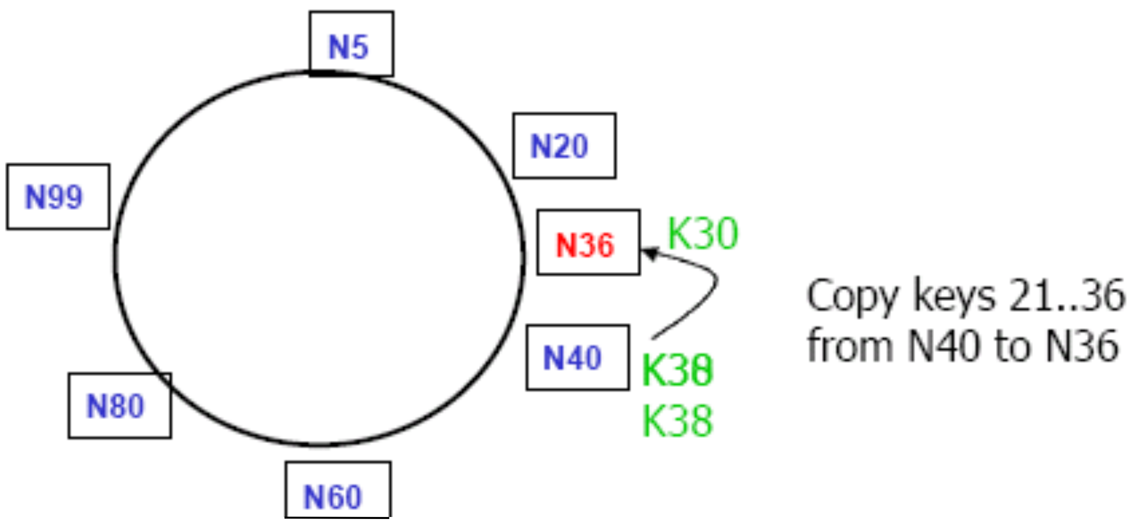
Chord Solution: “finger tables”

- Node K knows the node that is maintaining $K + 2^i$, where K is mapped id of current node
  - Increase distance exponentially (small world?)



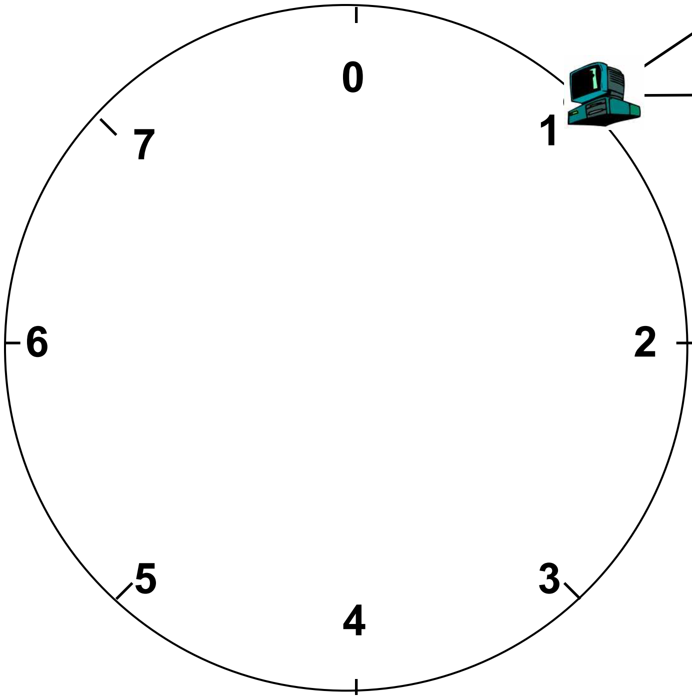
Joining the Ring

- Use a contact node to obtain info
- Transfer keys from successor node to new node
- Updating fingers of existing nodes



DHT: Chord Join

- Assume an identifier space [0..8]
- Node n1 joins

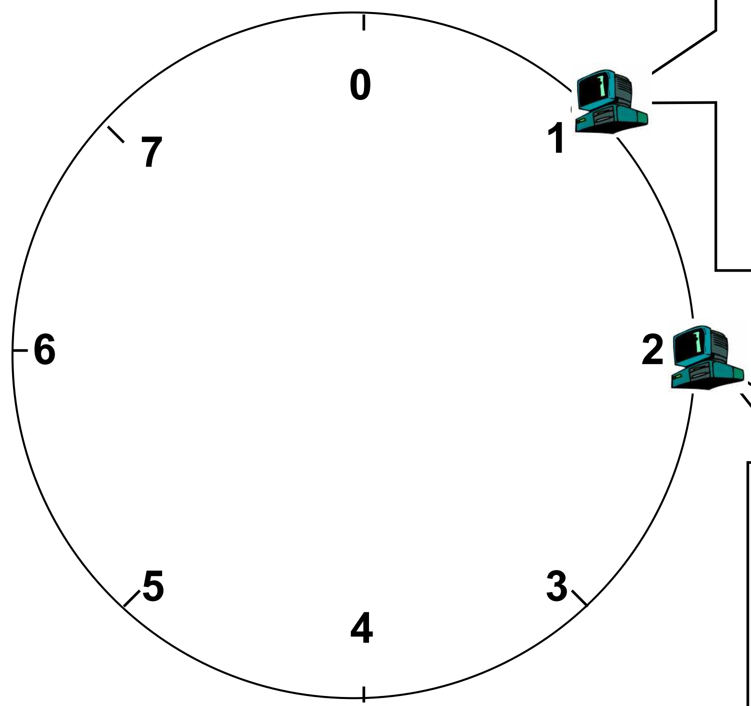


Succ. Table

i	$id+2^i$	succ
0	2	1
1	3	1
2	5	1

DHT: Chord Join

- Node n2 joins



Succ. Table

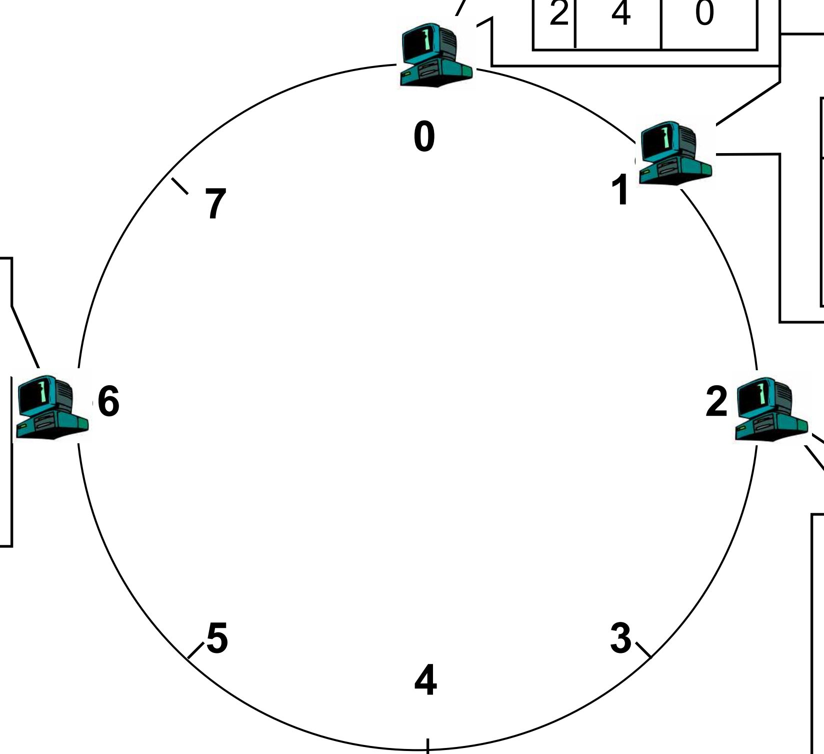
i	$id+2^i$	succ
0	2	2
1	3	1
2	5	1

Succ. Table

i	$id+2^i$	succ
0	3	1
1	4	1
2	6	1

DHT: Chord Join

- Nodes n0, n6 join



Succ. Table (node 0)

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	0

Succ. Table (node 1)

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Succ. Table (node 2)

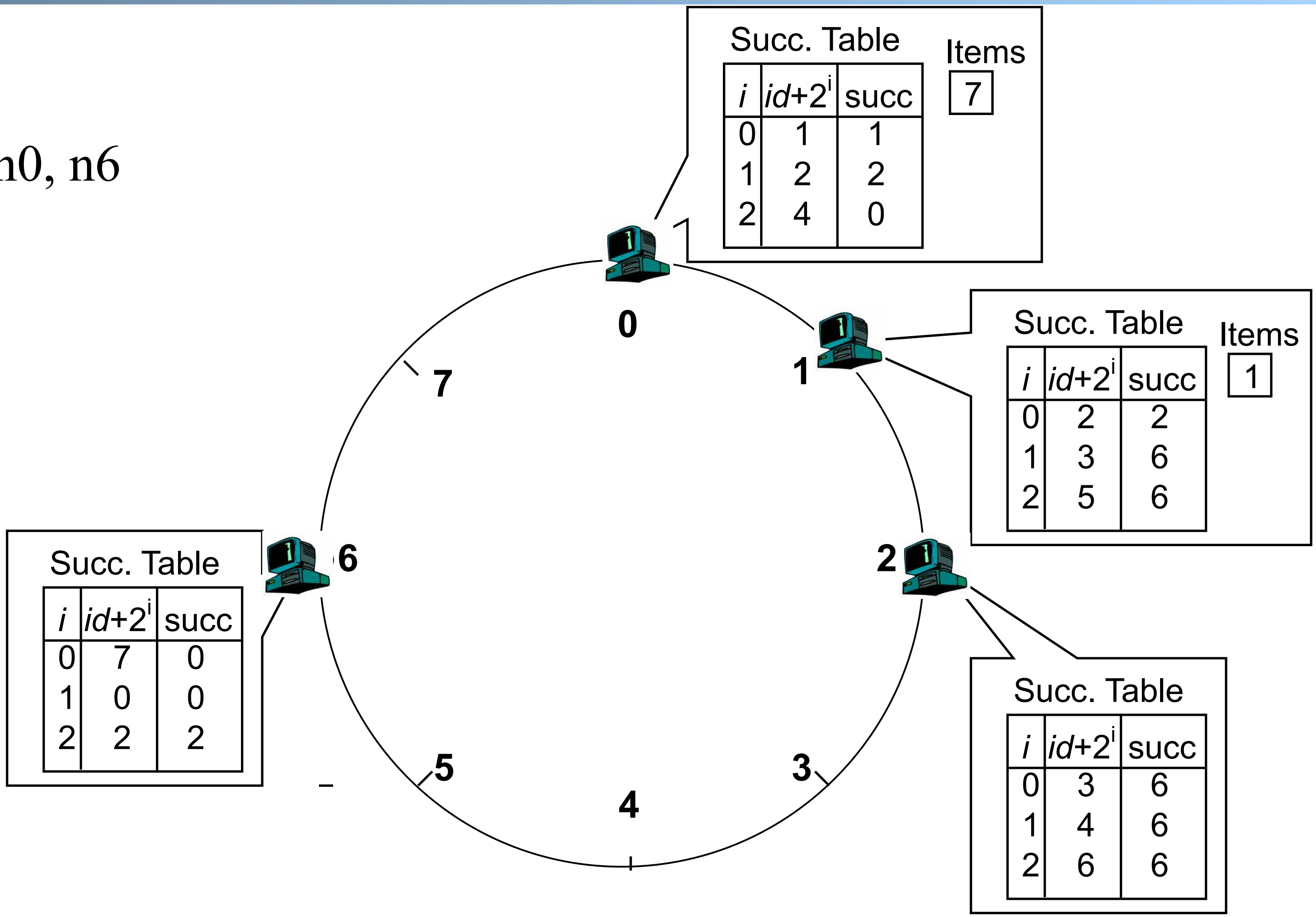
i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Succ. Table (node 6)

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

DHT: Chord Join

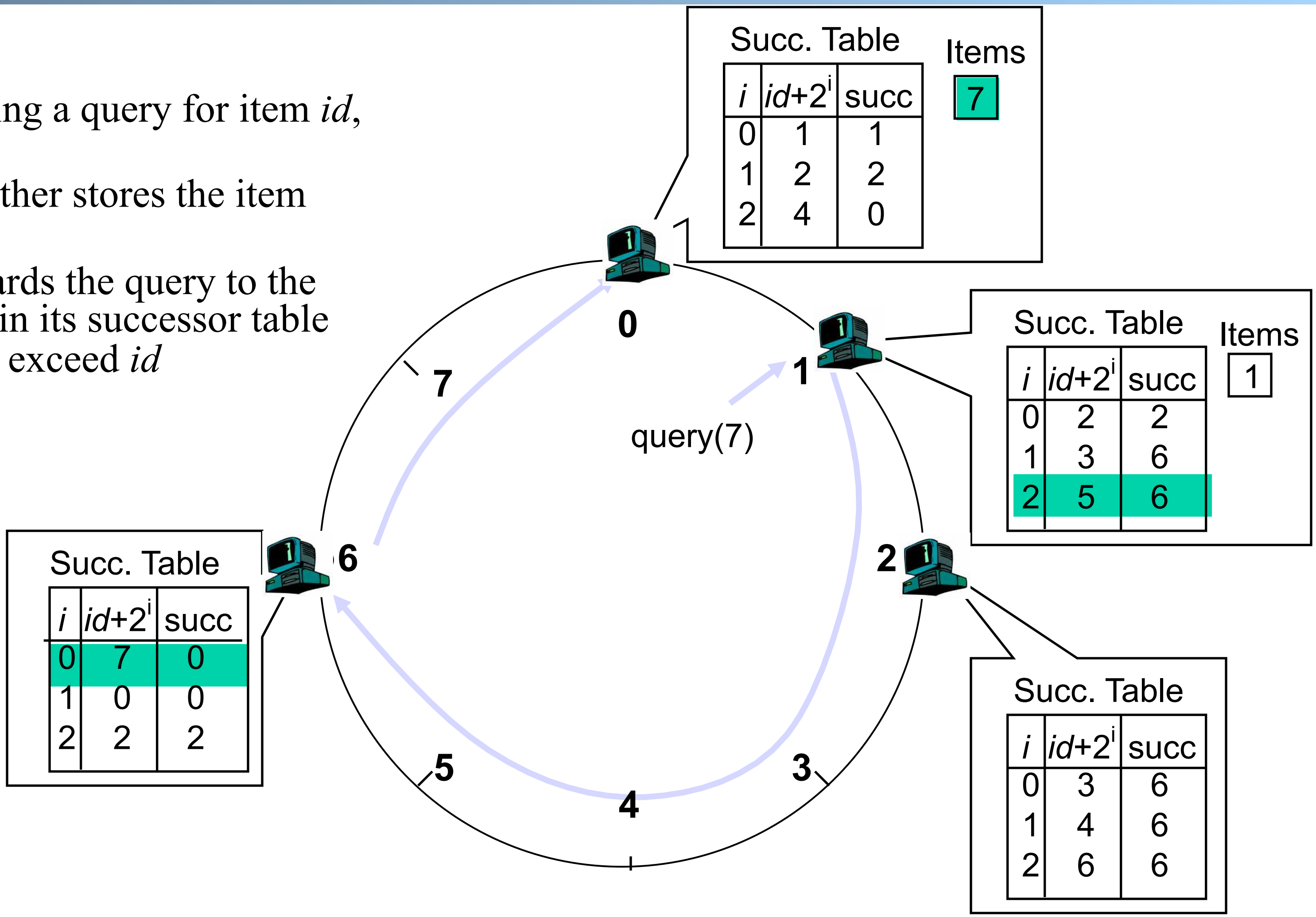
- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f1



DHT: Chord Routing

Two questions:
- how are the nodes connected?
- How the key space is partitioned?

- Upon receiving a query for item *id*, a node:
- Checks whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



Outline

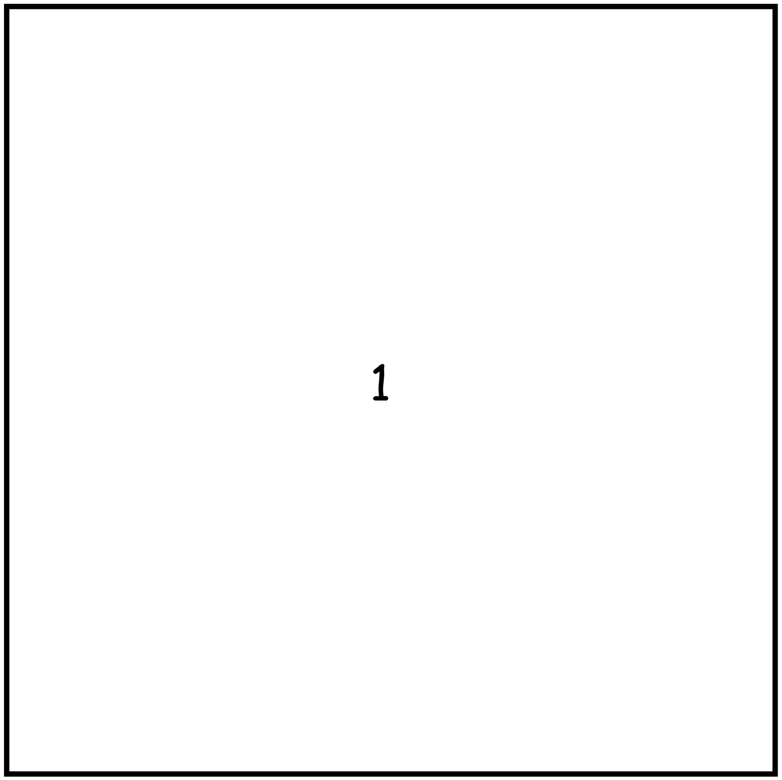
- Admin. and review
- *P2P*
  - *The lookup problem*
    - Napster (central query server; distributed data server)
    - Gnutella (decentralized, flooding)
    - Freenet (search by routing)
    - Chord (search by routing on a virtual ring)
  - *Content Addressable Network*

Content Addressable Network (CAN)

- Key space is an (virtual) d-dimensional Cartesian space
  - Associate to each item a unique coordinate in the space
  - Partition the space amongst all of the nodes

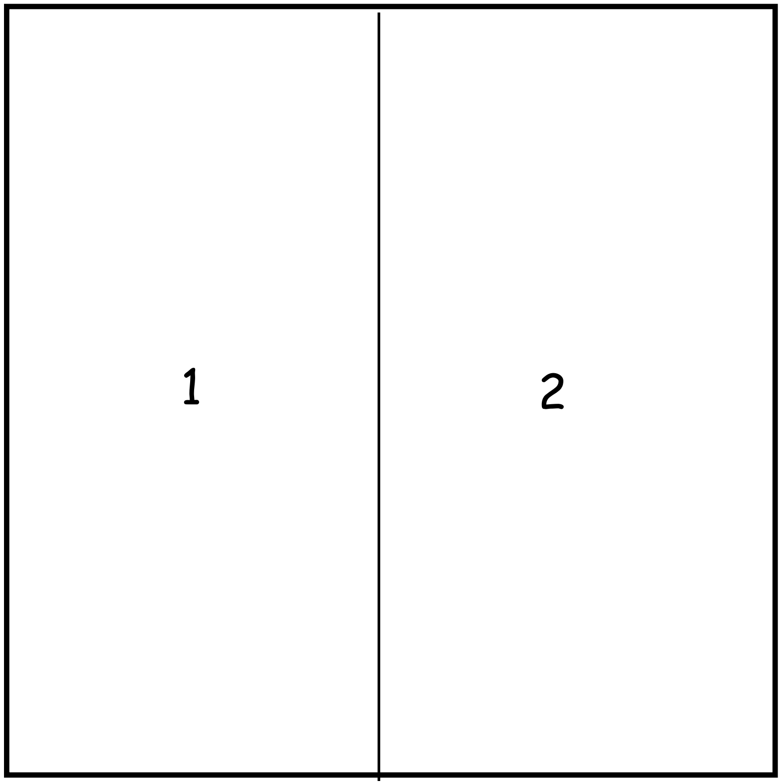
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



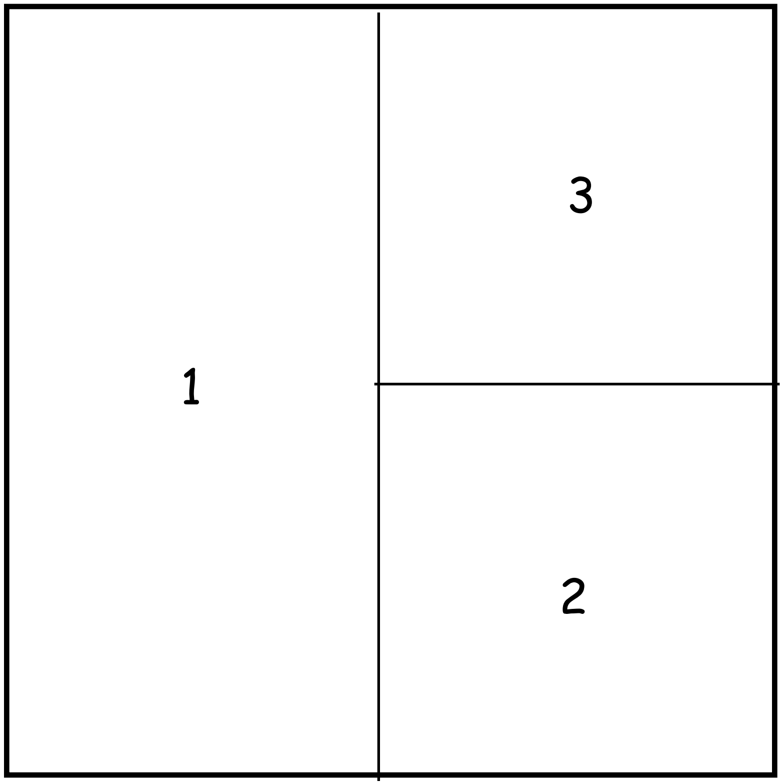
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



CAN Example: Two Dimensional Space

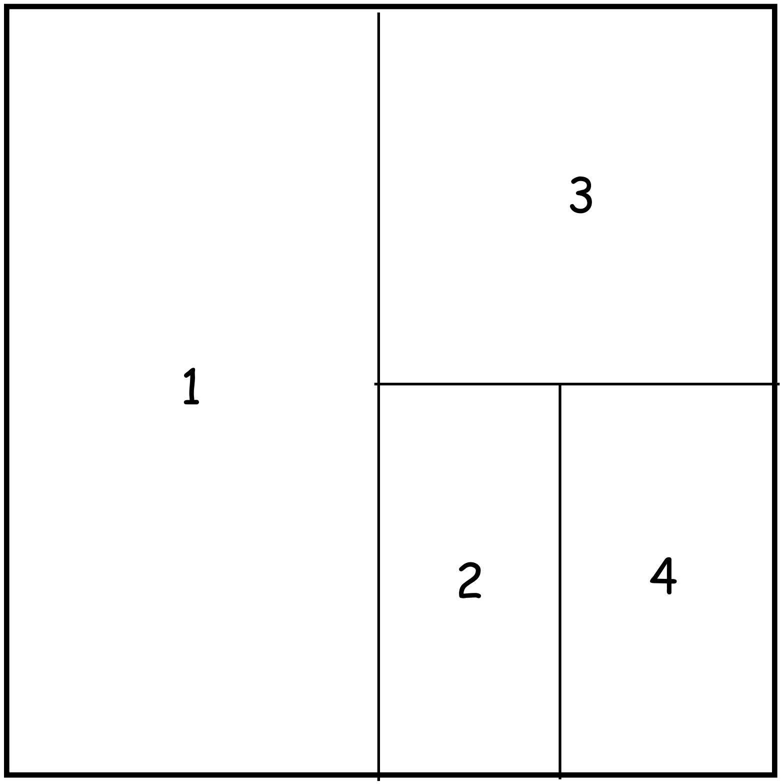
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



CAN Example: Two Dimensional Space

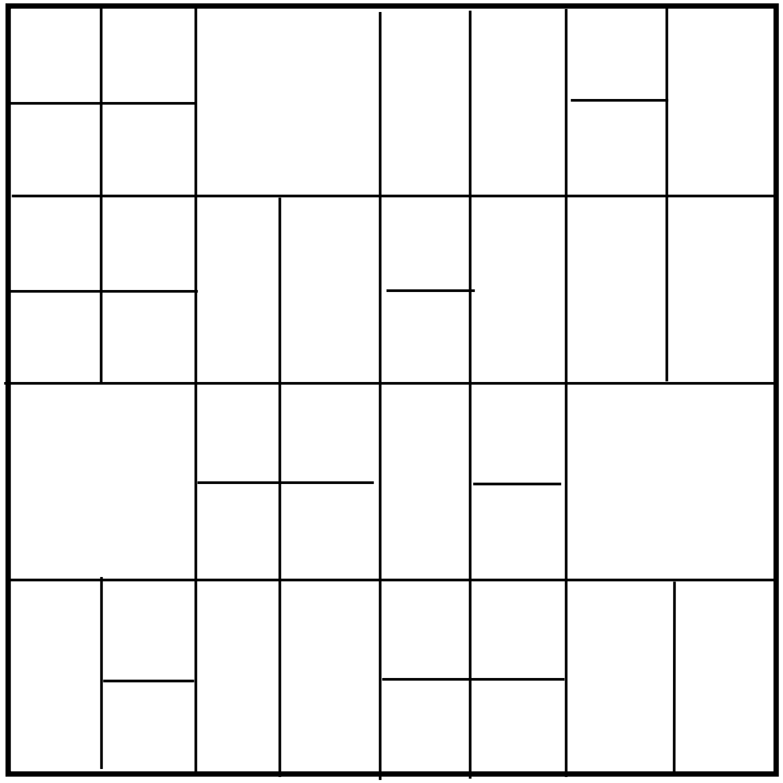
- Space divided among nodes

- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



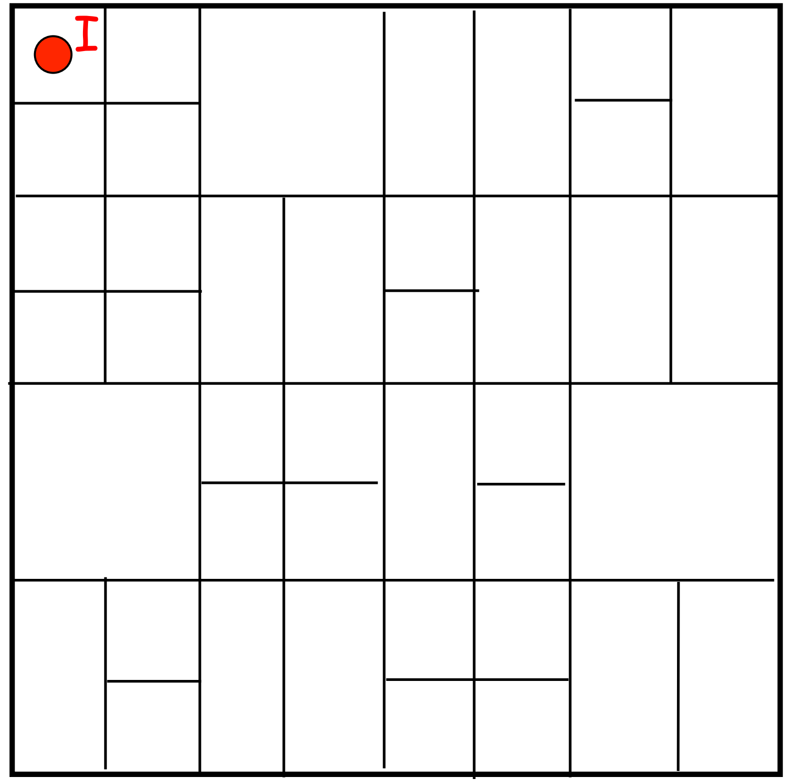
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



CAN Insert: Example (1)

node I::insert(K,V)

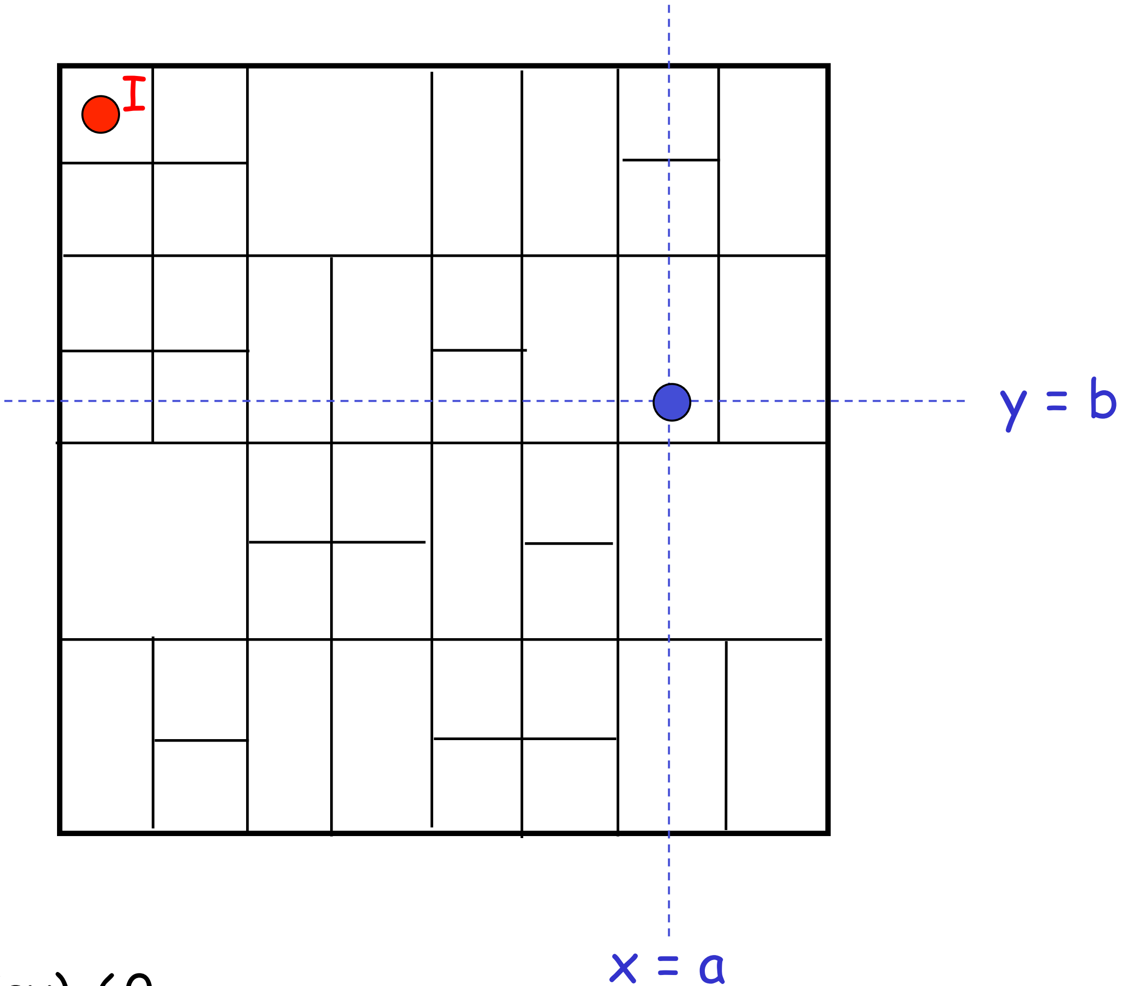


CAN Insert: Example (2)

node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$



Example: Key="Matrix3" h(Key)=60

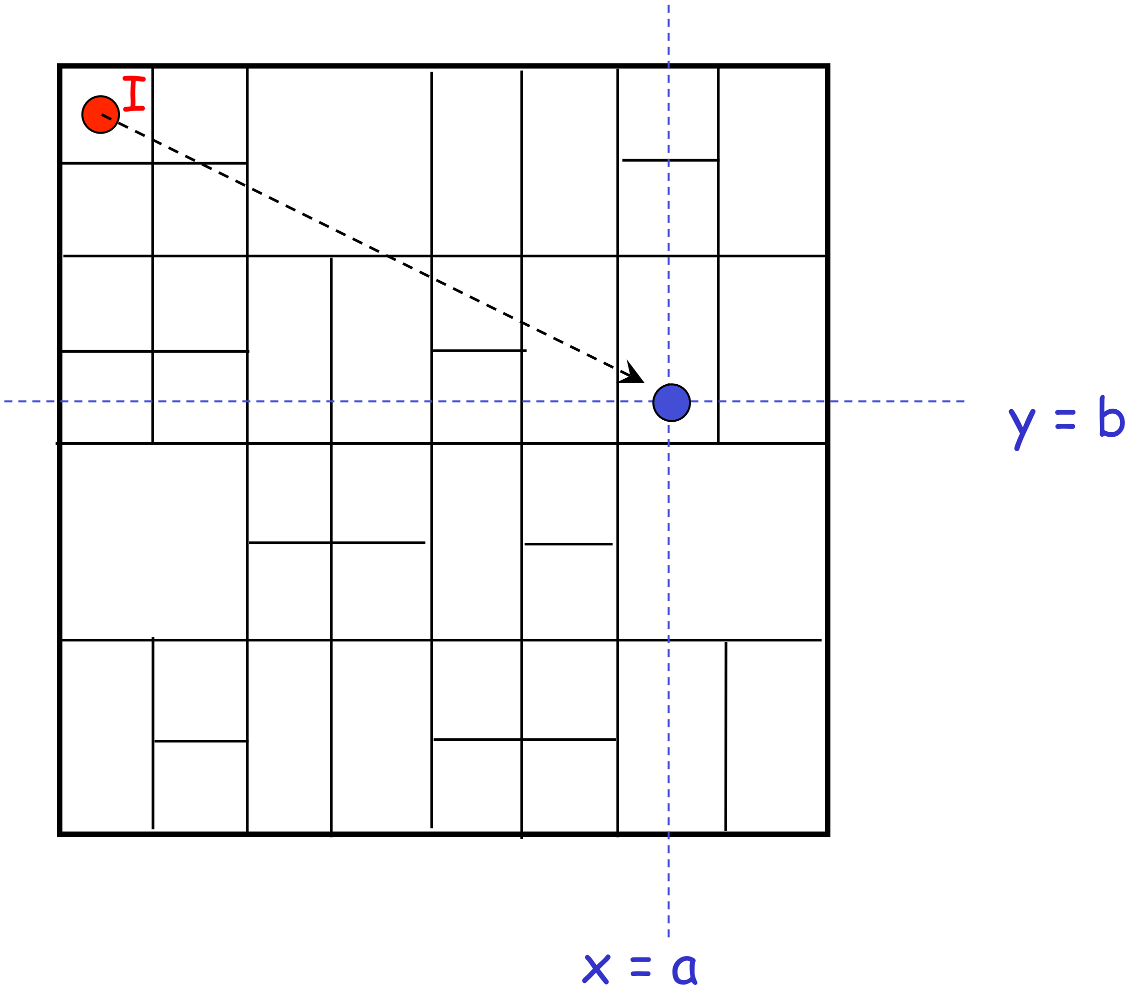
CAN Insert: Example (3)

node I::insert(K,V)

(1) $a = h_x(K)$

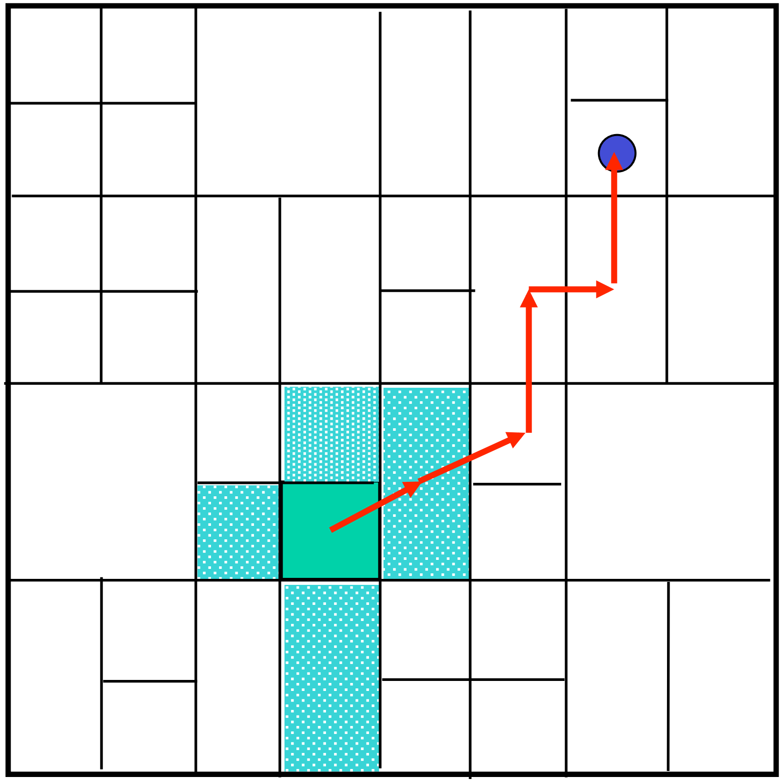
$b = h_y(K)$

(2) route(K,V) -> (a,b)



CAN Insert: Routing

- A node maintains state only for its immediate neighboring nodes
- Forward to neighbor which is closest to the target point
  - a type of greedy, local routing scheme



CAN Insert: Example (4)

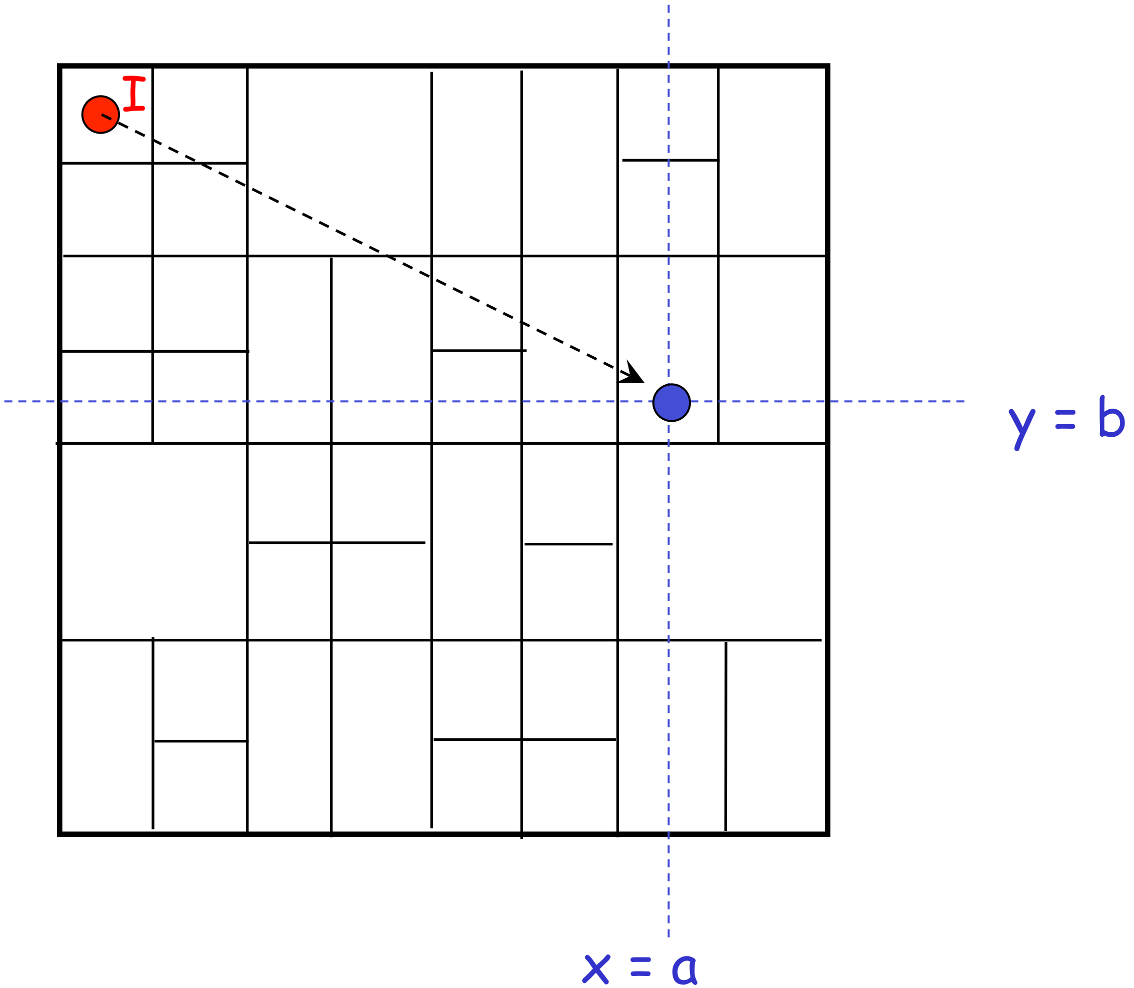
node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$

(2) route(K,V) -> (a,b)

(3) (K,V) is stored at (a,b)



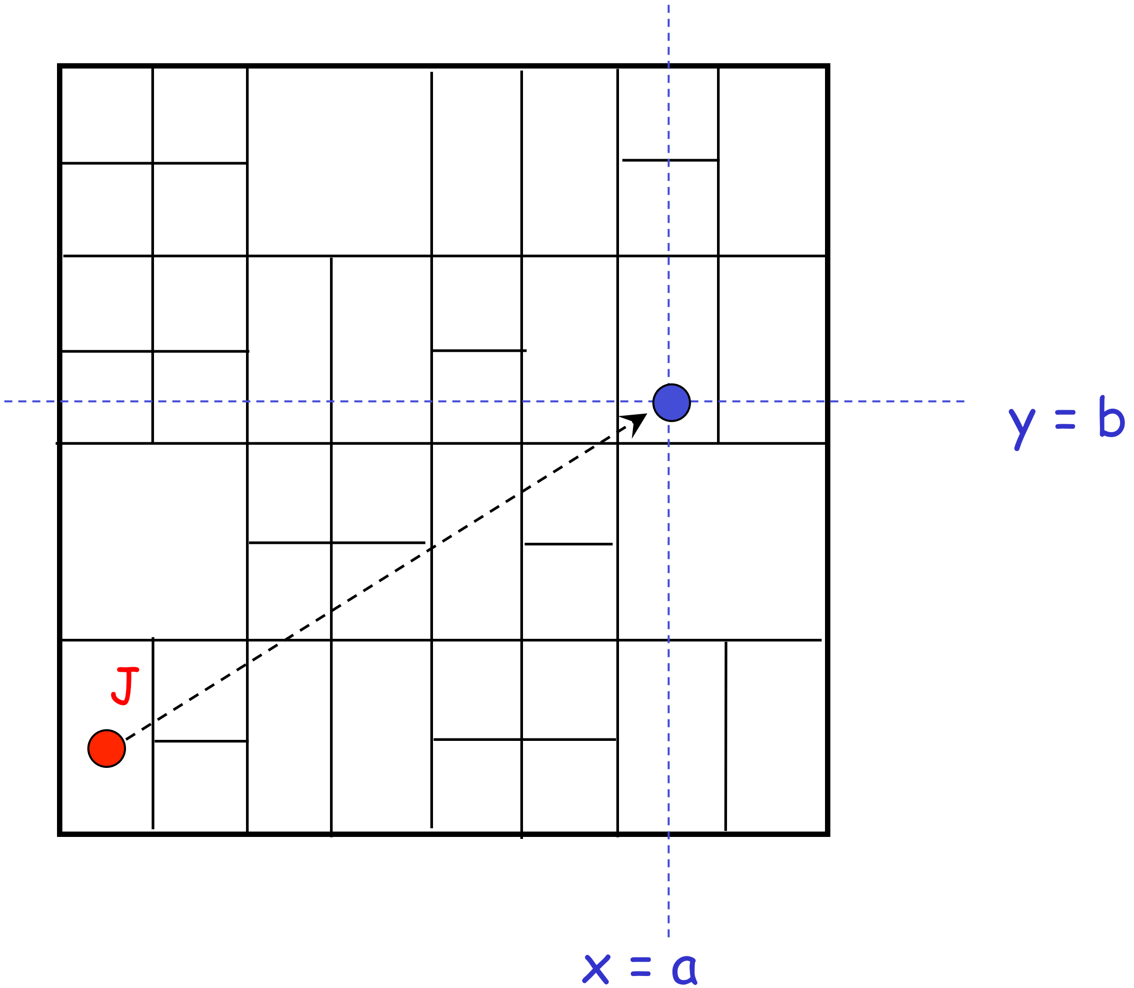
CAN Retrieve: Example

node J::retrieve(K)

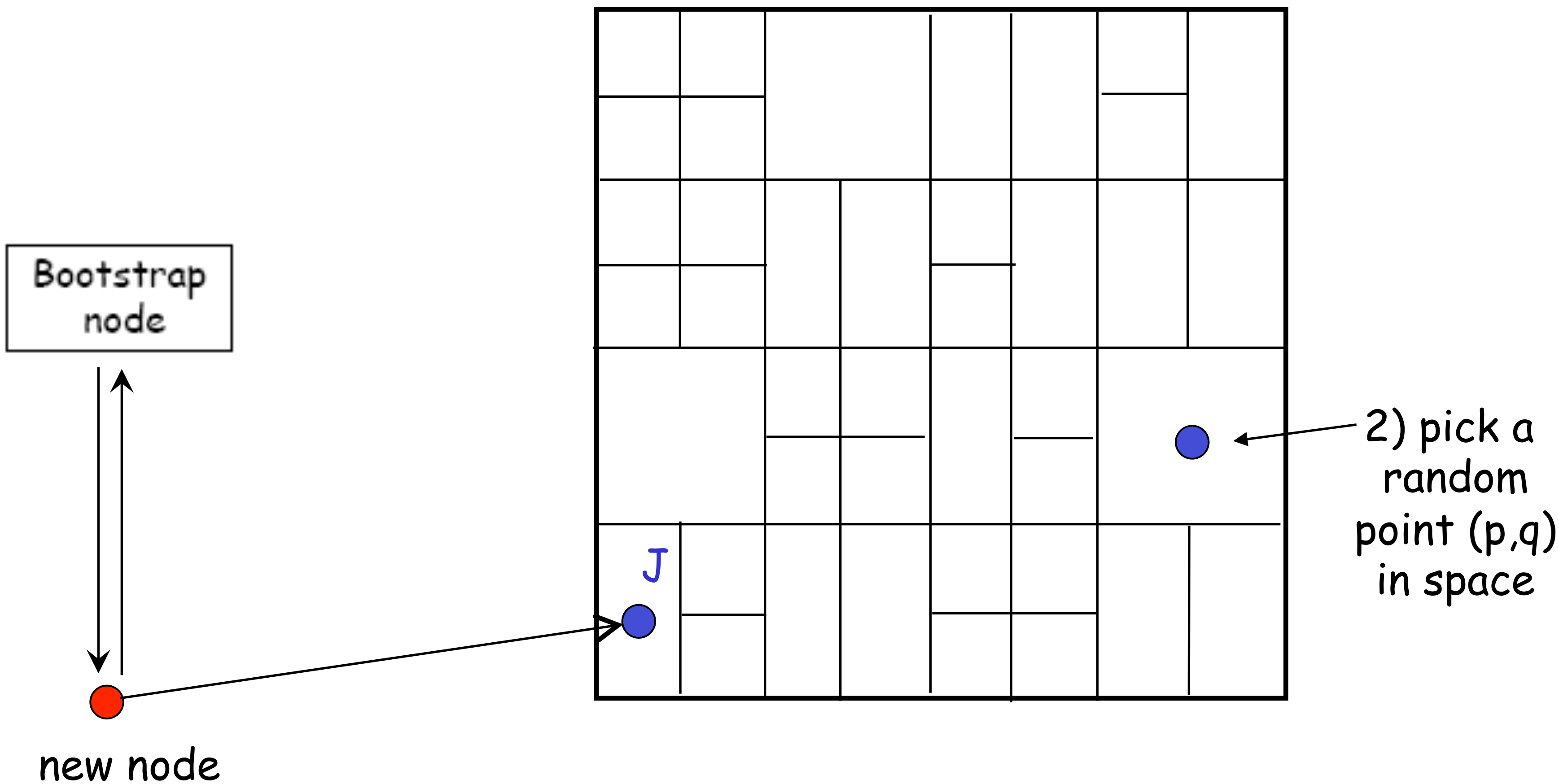
(1) $a = h_x(K)$

$b = h_y(K)$

(2) route "retrieve(K)" to (a,b)

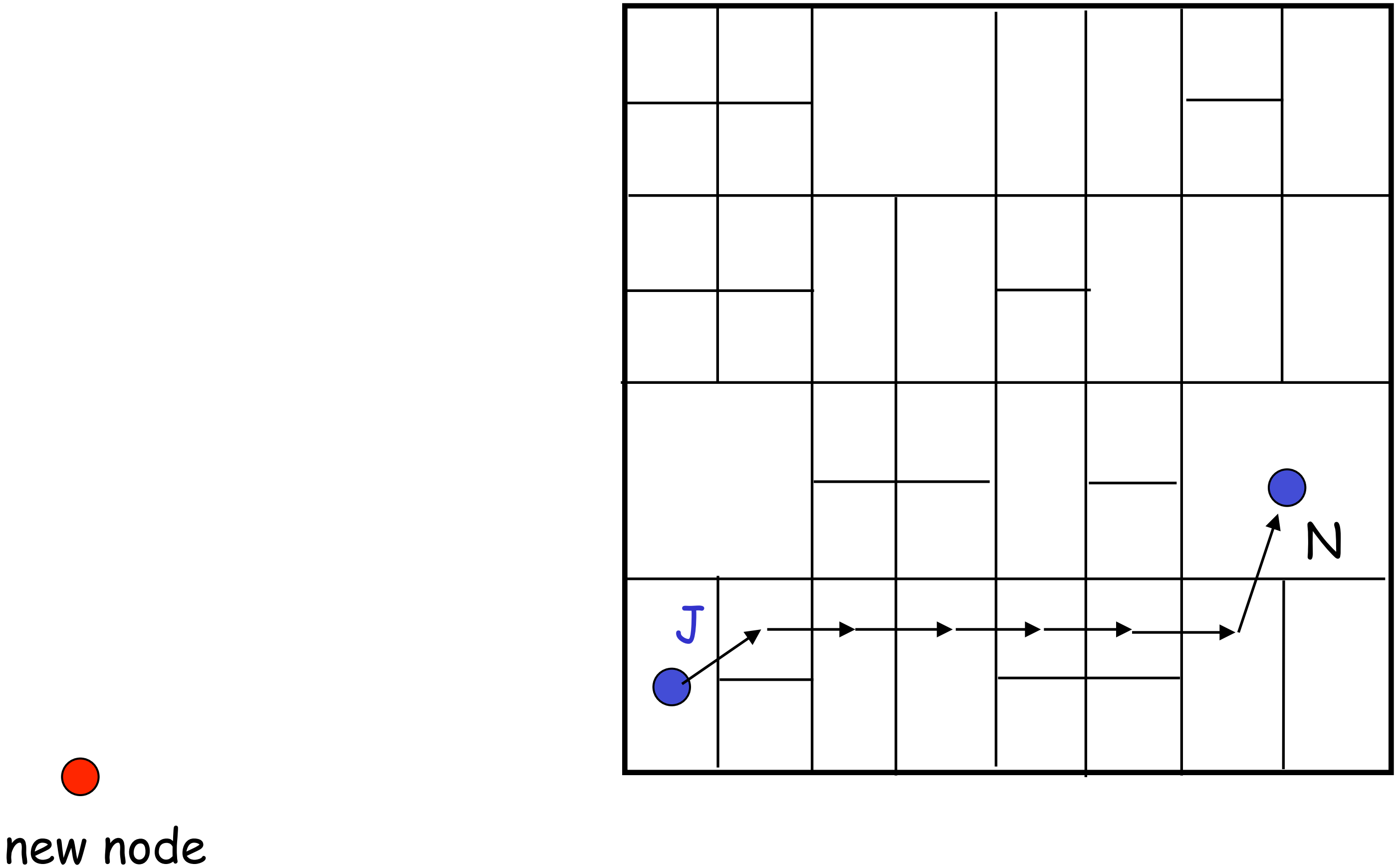


CAN Insert: Join (1)



1) Discover some node “J” already in CAN

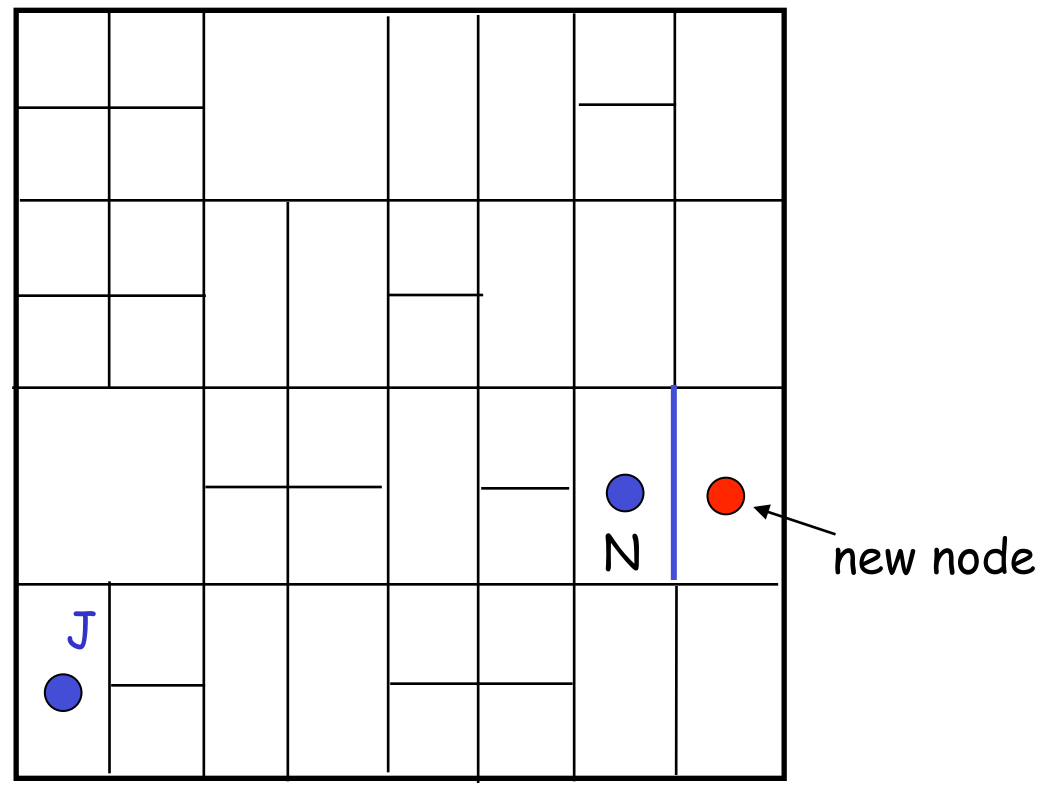
CAN Insert: Join (2)



3) J routes to (p,q), discovers node N

CAN Insert: Join (3)

Inserting a new node affects only a single other node and its immediate neighbors



4) split N' s zone in half… new node owns one half

CAN Complexity

- Guaranteed to find an item if in the network
- Scalability
  - For a uniform (regularly) partitioned space with **n** nodes and **d** dimensions
    - Per node, number of neighbors is 2d
    - Routing path length is $dn^{1/d}$
    - Average routing path is $(dn^{1/d})/3$ hops (due to Manhattan distance routing, expected hops in each dimension is dimension length * 1/3)
  - A fixed d can scale the network without increasing per-node state
- Load balancing
  - Hashing achieves some load balancing
  - Overloaded node replicates popular entries at neighbors
- Robustness
  - No single point of failure
  - Can route around trouble

Chord/CAN Summary

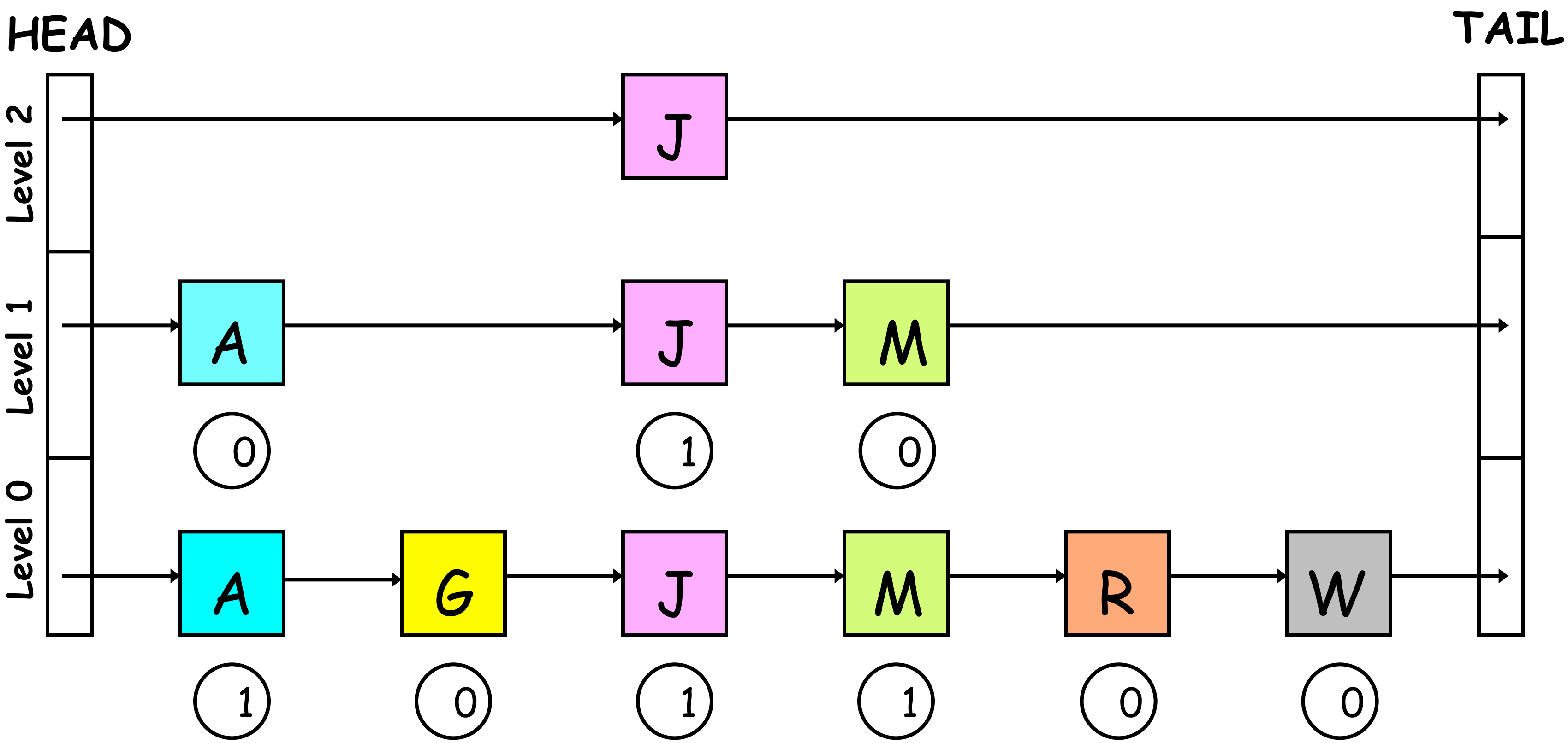
- Each node "owns" some portion of the key-space
  - In CAN, it is a multi-dimensional "zone"
  - In Chord, it is the key-id-space between two nodes in 1-D ring
- Files and nodes are assigned random locations in key-space
  - Provides some load balancing
  - Probabilistically equal division of keys to nodes
- Routing/search is local (distributed) and greedy
  - Node X does not know of a path to a key Z
  - But if node Y appears to be closest to Z among all of the nodes known to X
  - So route to Y

There are Other DHT Algorithms

- Tapestry (Zhao et al)
  - Keys interpreted as a sequence of digits
  - Incremental suffix routing
    - Source to target route is accomplished by correcting one digit at a time
    - For instance: (to route from 0312 → 1643)
    - 0312 → 2173 → 3243 → 2643 → 1643
  - Each node has a routing table

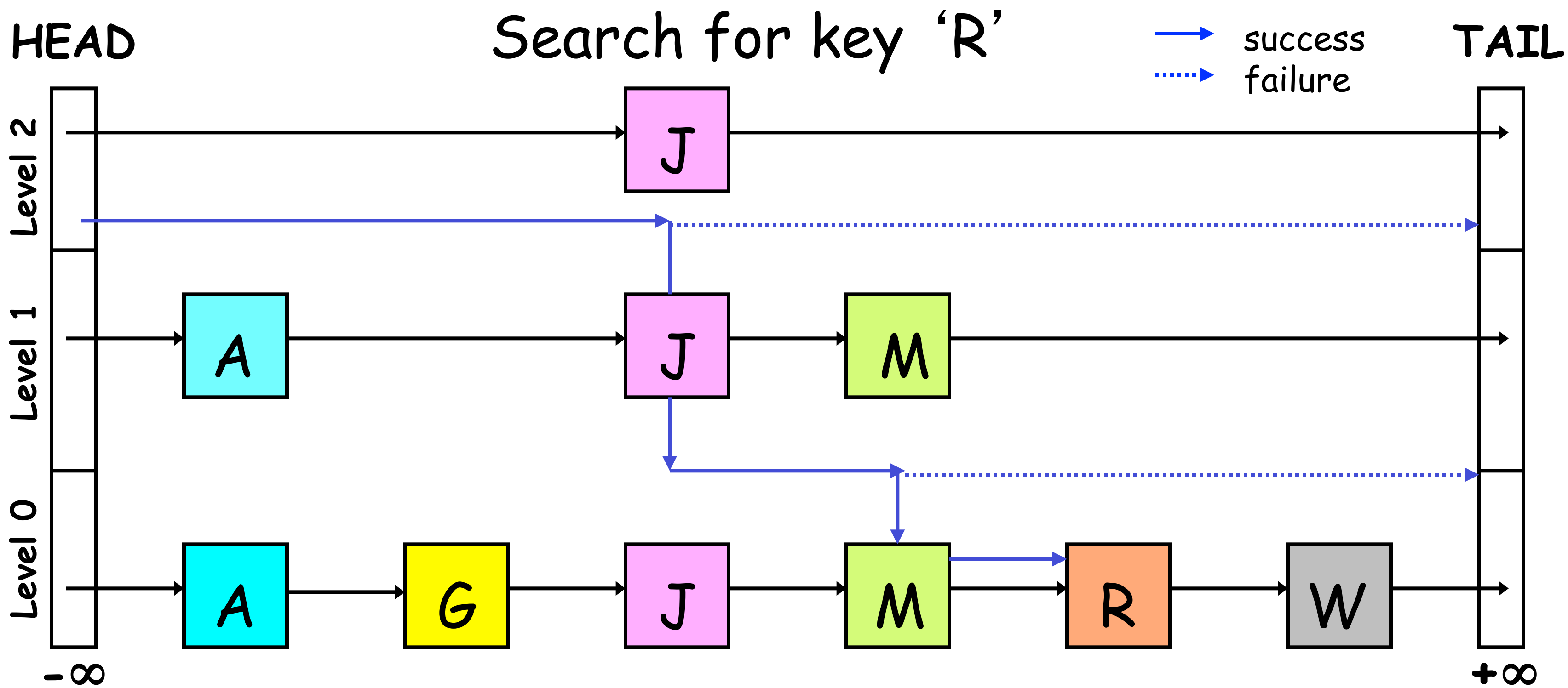
- Skip Graphs (Aspnes and Shah)

Skip List



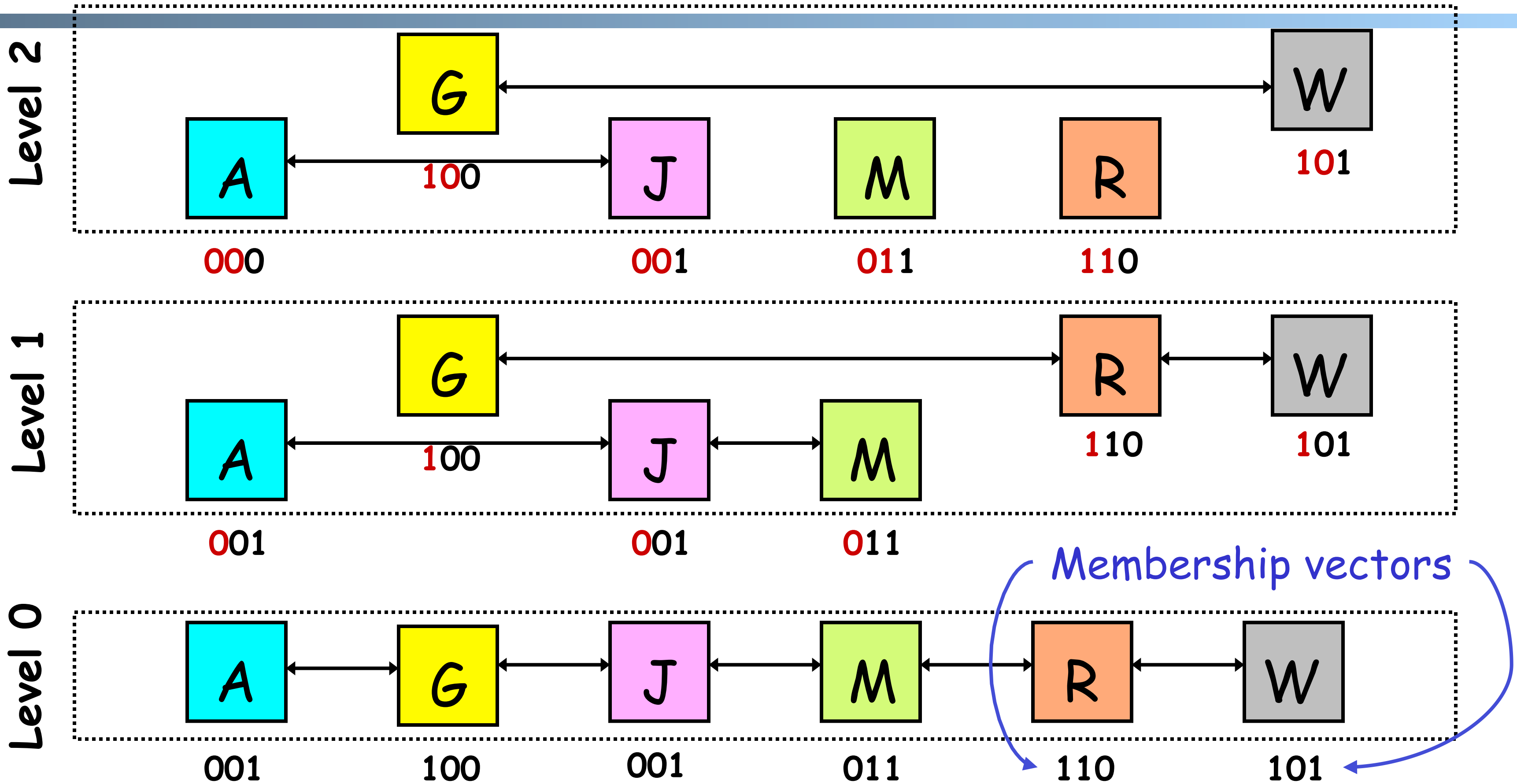
Each node linked at higher level with probability 1/2.

Searching in a skip list



Time for search: O(log n) on average.
On average, constant number of pointers per node.

A Skip Graph



Link at level i to nodes with matching prefix of length i.
Think of a tree of skip lists that share lower layers.

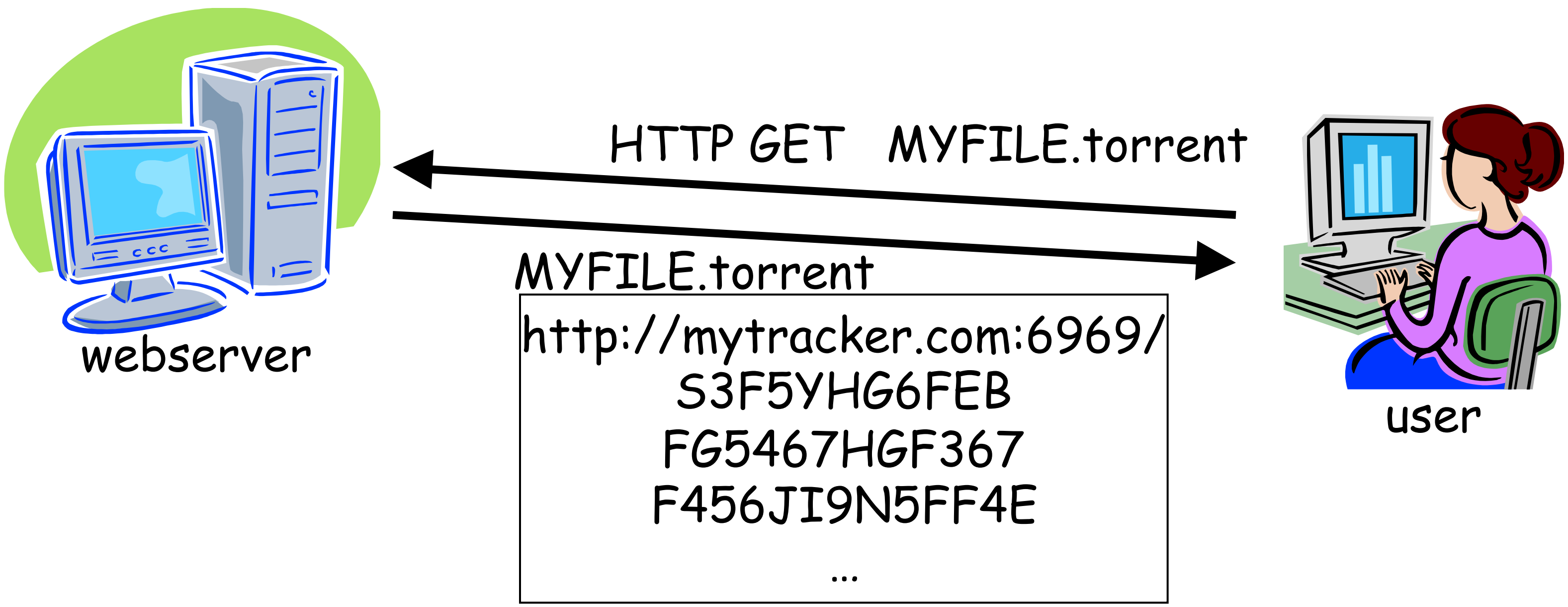
Summary: DHT

- Underlying metric space.
- Nodes embedded in metric space
- Location determined by key
- Hashing to balance load
- Greedy routing
- Typically
  - $O(\log n)$ space at each node
  - $O(\log n)$ routing time

Outline

- Recap
- *P2P*
  - *the lookup problem*
    - Napster (central query server; distributed data server)
    - Gnutella (decentralized, flooding)
    - Freenet (search by routing)
    - Chord (search by routing on a virtual ring)
    - Content Addressable Network (virtual zones)
  - *The scalability problem*

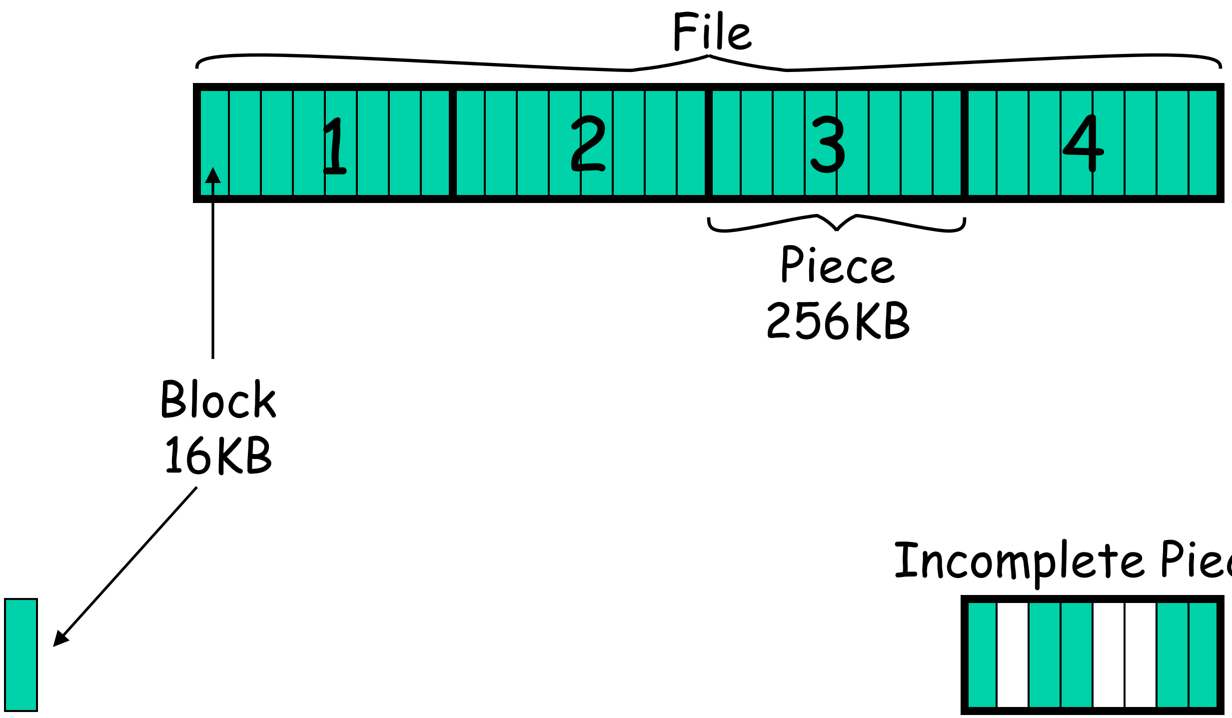
BitTorrent: Initialization



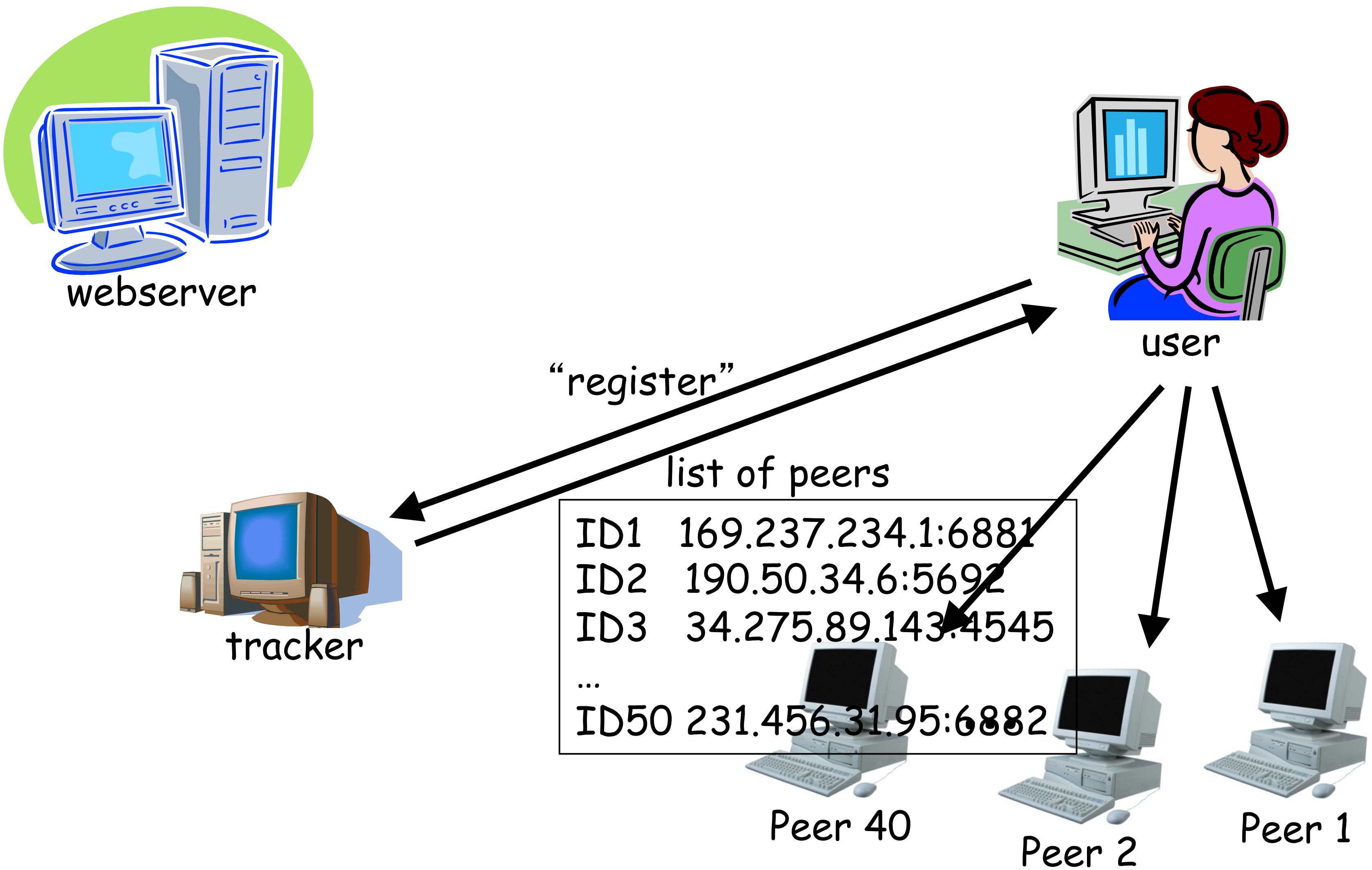
Metadata File Structure

- Contains information necessary to contact the tracker and describes the files in the torrent
  - Announce URL of tracker
  - File name
  - File length
  - Piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - Also creation date, comment, creator, …

File Organization



Tracker Protocol



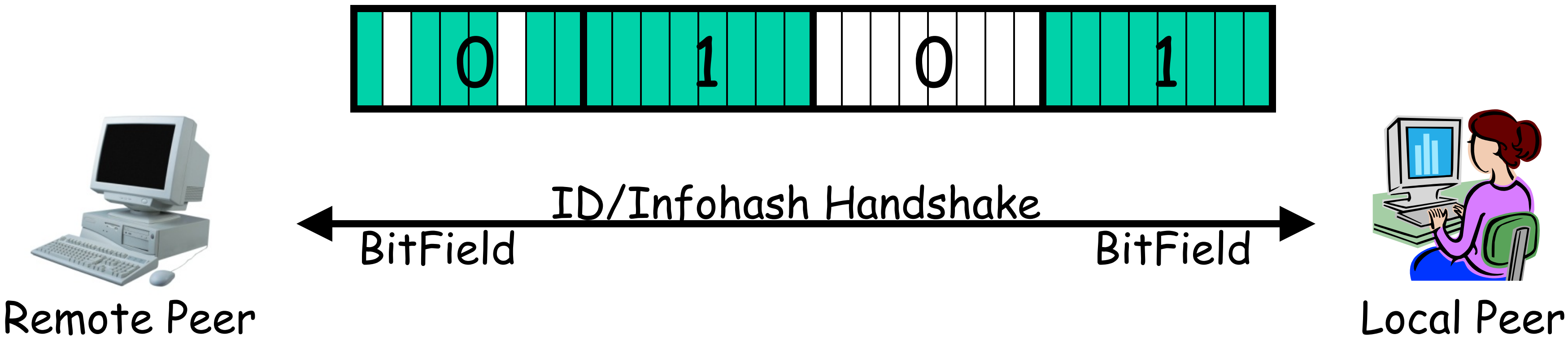
Tracker Protocol

- Communicates with clients via HTTP/HTTPS
- Client GET request
  - Info_hash: uniquely identifies the file
  - Peer_id: chosen by and uniquely identifies the client
  - Client IP and port
  - Numwant: how many peers to return (defaults to 50)
  - Stats: bytes uploaded, downloaded, left
- Tracker response
  - Interval: how often to contact the tracker
  - List of peers, containing peer id, IP and port
  - Stats: complete, incomplete

- Tracker-less mode; based on the Kademlia DHT

“On the Wire” Protocol

(Over TCP)

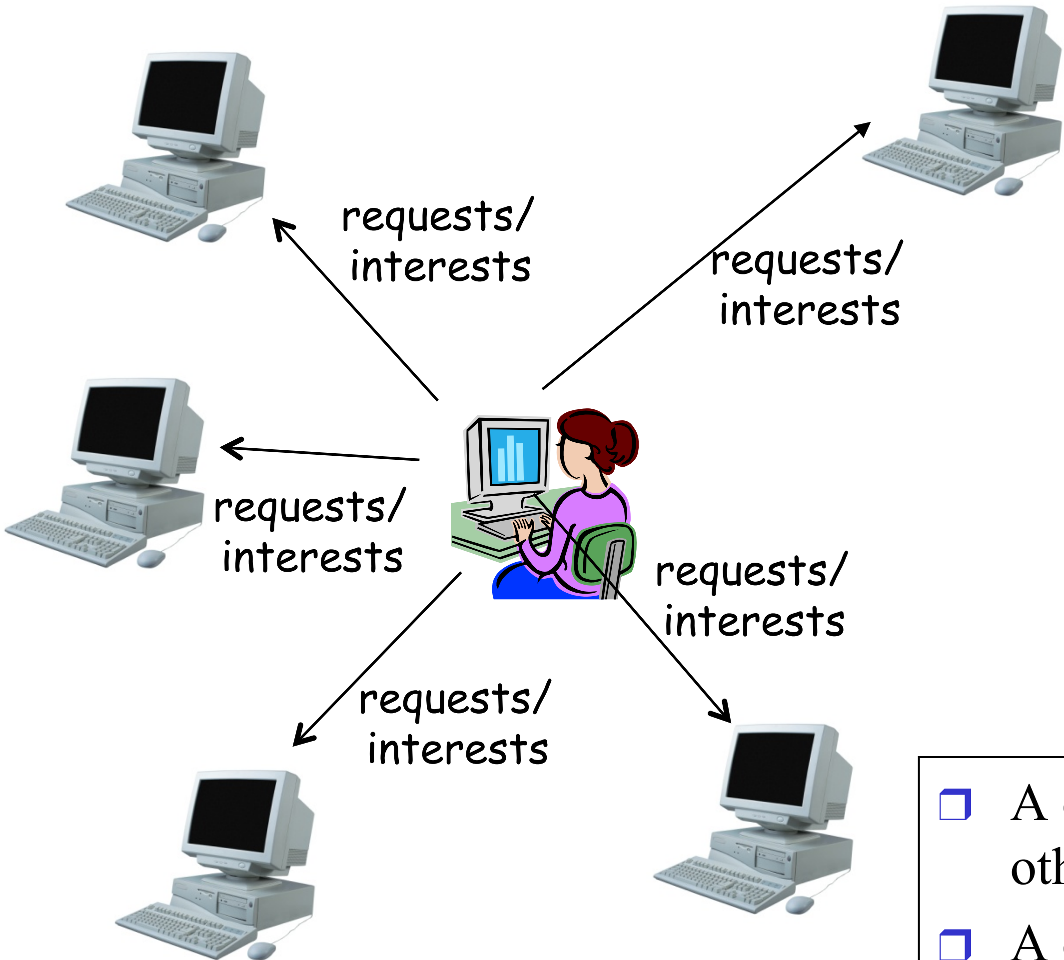


Questions
- How to efficiently utilize the peer bandwidth?
- How to deal with the incentive issue?

Piece Selection: Requests/ Interests

- When downloading starts: choose at random and request them from the peers
  - Get pieces as quickly as possible
  - Obtain something to offer to others
- After 4 pieces: request (local) rarest first
  - Achieves the fastest replication of rare pieces
  - Obtain something of value
- Endgame mode
  - Defense against the "*last-block problem*": cannot finish because missing a few last pieces
  - Send requests for missing sub-pieces to all peers in our peer list
  - Send cancel messages upon receipt of a sub-piece

Piece Selection: Requests/ Interests

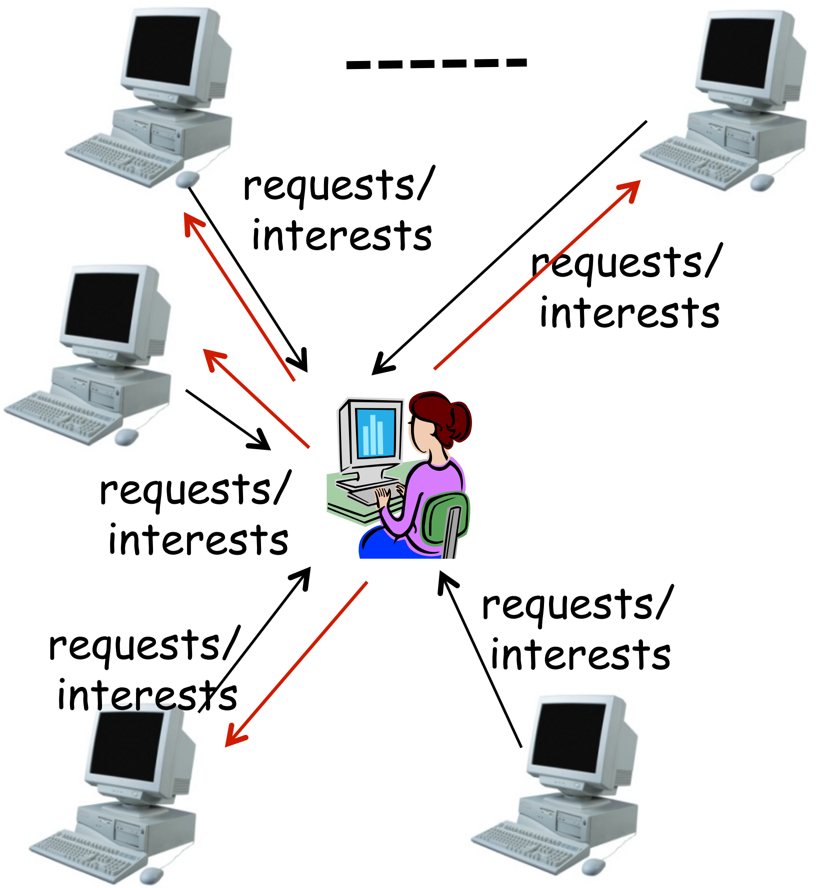


- A client has content that others need
- A client is willing to upload

Peer Selection – Response/ Unchoking

- Periodically (typically every 10 seconds) calculate data-receiving rates from all peers
- Upload to (*unchoke*) the fastest

  - constant number (4) of unchoking slots
  - partition upload *bandwidth* equally among unchoked



commonly referred to as "tit-for-tat" strategy

Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - Typically every 3 unchoking rounds (30 seconds)
- Multi-purpose mechanism
  - Allow bootstrapping of new clients
  - Continuously look for the fastest peers (exploitation vs exploration)

BitTorrent: Summary

- Very widely used
  - Mainline: written in Python
  - Azureus: the most popular, written in Java
  - Other popular clients: ABC, BitComet, BitLord, BitTornado, μTorrent, Opera browser
- Many explorations, e.g.,
  - BitThief
  - BitTyrant
- Better understanding is needed