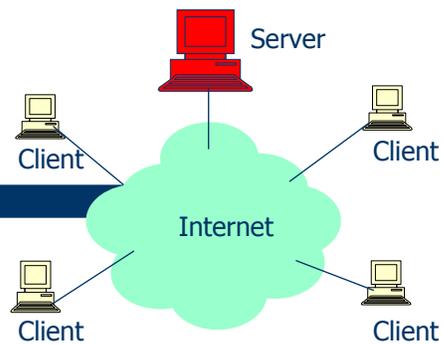


Peer-to-Peer Networking

1

History I



- Client-Server computing
- Well known and powerful
- Server provides services and resources
- Multiple clients can be supported by a single server
- 1-Many relationship => scalability with respect to the number of clients
- Model has dominated the architectural design of many applications
- Examples: HTTP, DNS, FTP

2

History II

Client-Server computing does have limitations

ISSUES

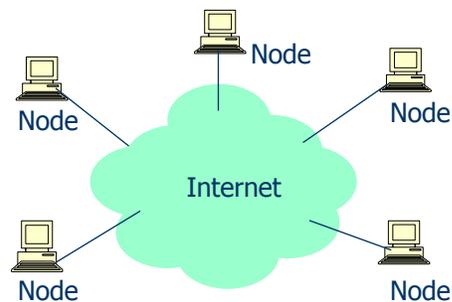
- Fault tolerance
- Central administration
- “Extreme” scalability
- Unused resources in “clients”

3

Peer-to-Peer Networks (P2P)

Peer-to-Peer computing

- **Computing paradigm where all the nodes have equivalent responsibilities and roles**
- *“neither introduces nor prohibits centralization”*
- *“sharing of resources through direct communication between consumers and providers”*
- *“a network architecture where all the available resources are located at the network edges”*
- *“the opposite of client-server”*



4

P2P Characteristics

- ✓ **Each node acts both as client and server**
- ✓ **Nodes are autonomous**
- ✓ **Network is dynamic**
- ✓ **There is no centralized authority** (*in theory*)
- ✓ **Network is large-scale**
- ✓ **Nodes have to co-operate in order to retrieve a resource or a service**

5

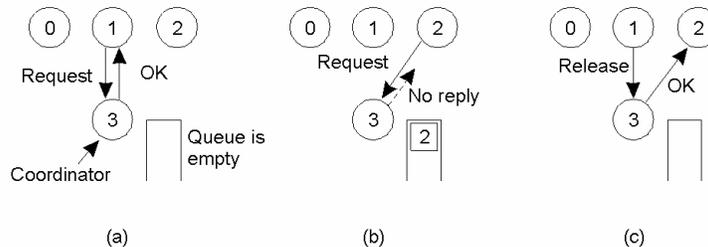
P2P Benefits

- **Efficient use of resources**
- **Scalability**
- **Reliability**
- **Ease of administration**

- **But: these benefits are not easy to achieve, see example of mutual exclusion algorithms on next few slides**

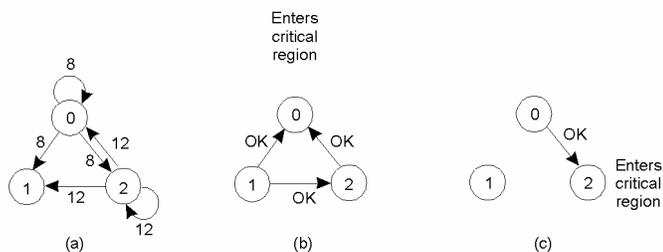
6

Mutual Exclusion: A Centralized Algorithm



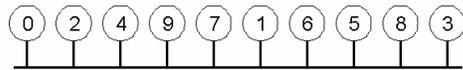
- 7
- Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
 - Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
 - When process 1 exits the critical region, it tells the coordinator, when then replies to 2

A Distributed Algorithm

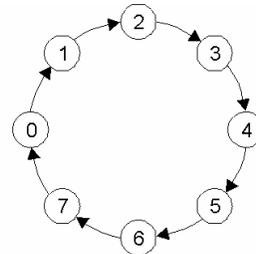


- 8
- Two processes want to enter the same critical region at the same moment.
 - Process 0 has the lowest timestamp, so it wins.
 - When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

A Token Ring Algorithm



(a)



(b)

- a) An unordered group of processes on a network.
- b) A logical ring constructed in software.

9

Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	$2(n - 1)$	$2(n - 1)$	Crash of any process
Token ring	1 to ∞	0 to $n - 1$	Lost token, process crash

10

Examples of P2P Systems

- Napster
- KaZaA
- Gnutella
- FreeNet
- NeuroGrid
- Chord, CAN, Tapestry
- JXTA

11

Primary P2P research question

How can we **efficiently** and **accurately** discover resources and services in a P2P network?

Solution 1: Introduce some centralization

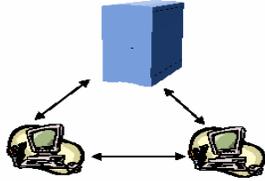
Solution 2: Introduce some structure

“centralization” and “structure” define two dimensions for classifying P2P networks

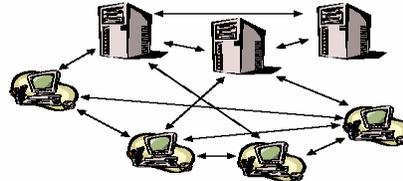
12

Types of P2P Systems I

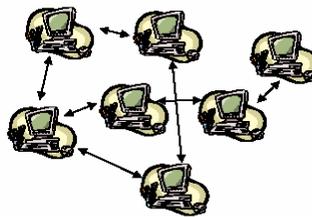
Centralized



Partially decentralized



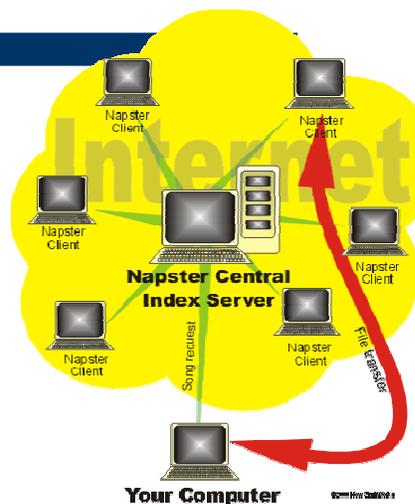
Totally decentralized



13

Napster I

- Sharing of music files
- Lists of files are uploaded to Napster server
- Queries contain various keywords of required file
- Server returns IP address of user machines having the file
- File transfer is direct



14

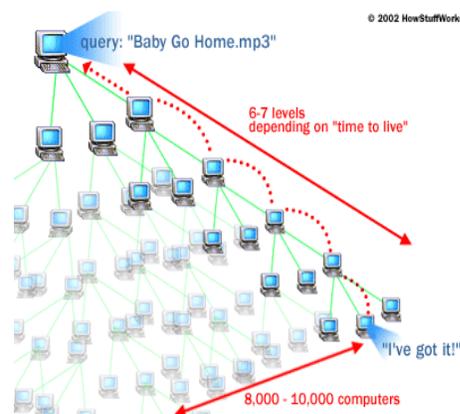
Napster II

- ✓ Centralised model
- ✓ Napster server ensures correct results
- ✓ Only used for finding the location of the files
- Scalability bottleneck
- Single point of failure
- Denial of Service attacks possible
- Lawsuits

15

Gnutella I

- Sharing of any type of files
- Decentralised search
- Queries are sent to the neighbour nodes
- Neighbours ask their own neighbours and so on
- Time To Live (TTL) field on queries
- File transfer is direct



16

Gnutella II

- ✓ Decentralised model
- ✓ No single point of failure
- ✓ Less susceptible to denial of service
- SCALABILITY (flooding)
- Cannot ensure correct results

17

KaZaA

- Hybrid of Napster and Gnutella
- Super-peers act as local search hubs
 - Each super-peer is like a constrained Napster server
 - Automatically chosen based on capacity and availability
- Lists of files are uploaded to a super-peer
- Super-peers periodically exchange file lists
- Queries are sent to super-peers

18

Distributed Hash Tables (DHT)

- Distributed version of a hash table data structure
- Stores (key, value) pairs
 - The key is like a filename
 - The value can be file contents
- Goal: Efficiently insert/lookup/delete (key, value) pairs
- Each peer stores a subset of (key, value) pairs in the system
- Core operation: Find node responsible for a key
 - Map key to node
 - Efficiently route insert/lookup/delete request to this node

21

DHT Generic Interface

- **Node id**: m-bit identifier (similar to an IP address)
- **Key**: sequence of bytes
- **Value**: sequence of bytes

put(key, value)

- Store (key,value) at the node responsible for the key

value = get(key)

- Retrieve value associated with key (from the appropriate node)

22

DHT Applications

- File sharing
- Databases
- Service discovery
- Chat service
- Publish/subscribe networks

23

DHT Desirable Properties

- Keys mapped evenly to all nodes in the network
- Each node maintains information about only a few other nodes
- Efficient routing of messages to nodes
- Node insertion/deletion only affects a few nodes

24

Chord API

Node id: m-bit identifier (similar to an IP address)

Key: m-bit identifier (hash of a sequence of bytes)

Value: sequence of bytes

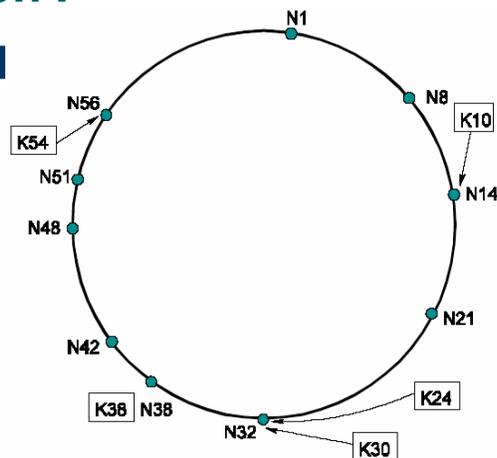
API

- `insert(key, value)`
- `lookup(key)`
- `update(key, newval)`
- `join(n)`
- `leave()`

25

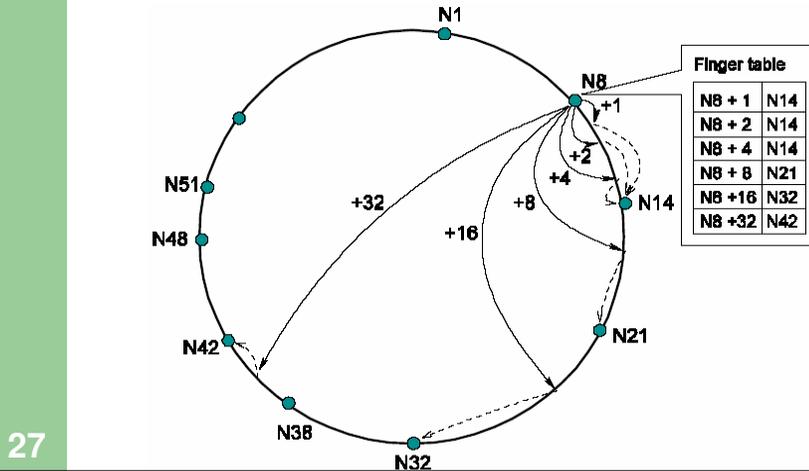
Chord Operation I

- Nodes form a circle based on node identifiers
- Each node is responsible in storing a portion of the keys
- Hash function ensures even distribution of keys and nodes in the circle



26

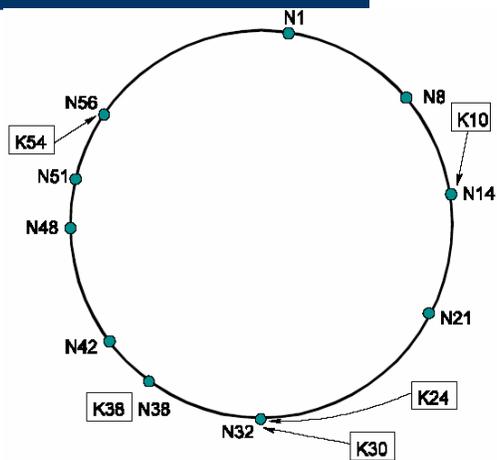
Chord Operation II



27

Chord Operation III

- Lookup the furthest node that precedes the key
- Query reaches target node in $O(\log N)$ hops



28

Chord Properties

In a system with N nodes and K keys:

- ✓ Each node manages at most K/N keys
- ✓ Bound information stored in every node
- ✓ Lookups resolved with $O(\log N)$ hops
- No delivery guarantees
- Poor network locality

29

Network Locality

Nodes close on ring can be far in the network



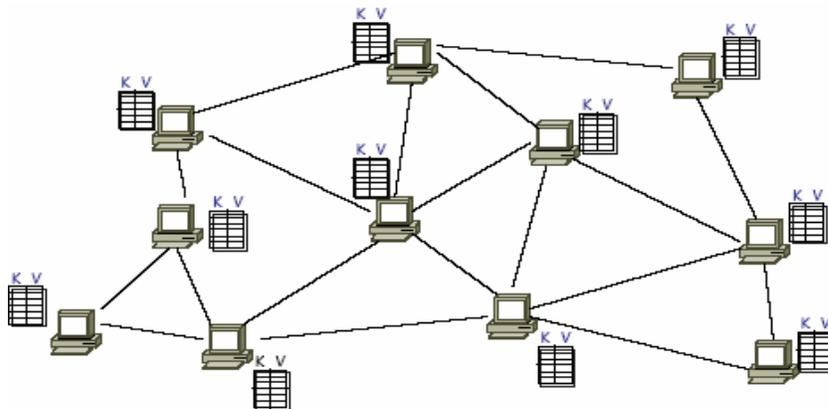
30

Content Addressable Network (CAN)

- CAN: Internet Scale Hash table
- Interface
 - `insert(key,value)`
 - `value = retrieve(key)`
- Idea: associate to each node and item a unique coordinate in an d-dimensional Cartesian space.
- Desired properties
 - scalable
 - operationally simple
 - good performance

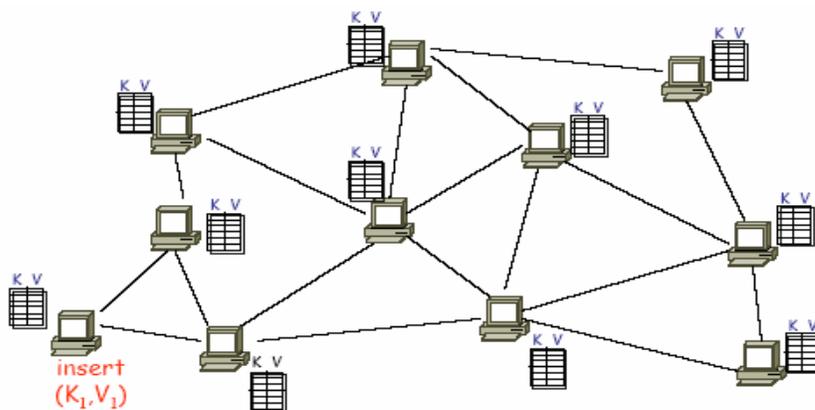
31

CAN: Basic Idea



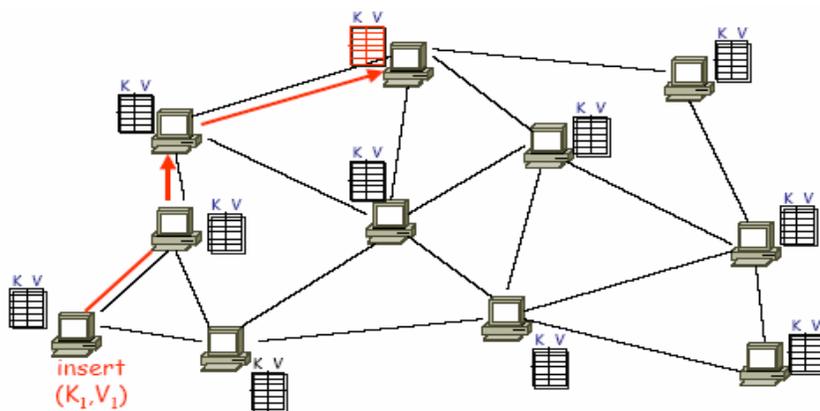
32

CAN: Basic Idea (2)



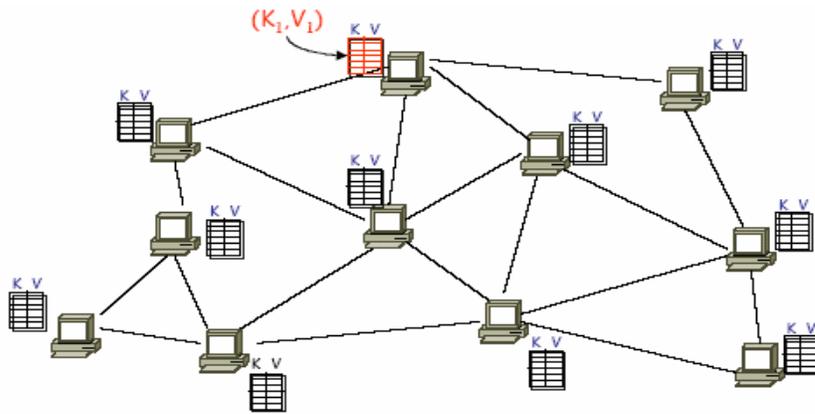
33

CAN: Basic Idea (3)



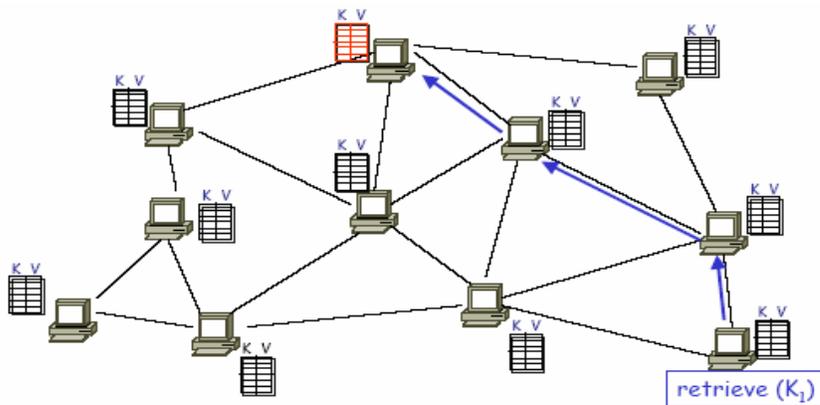
34

CAN: Basic Idea (4)



35

CAN: Basic Idea (5)



36

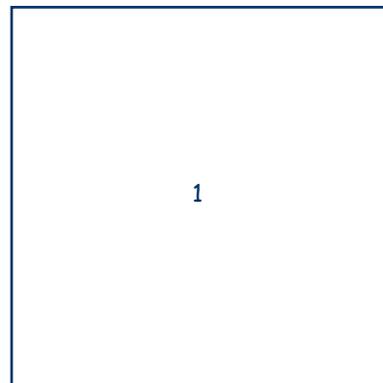
CAN Solution: Detail

- Entire space is partitioned amongst all the nodes
 - Virtual Cartesian coordinate space
- Every node “owns” a zone in the overall space
- Abstraction
 - store data at “points” in the space
 - route from one “point” to another
 - point = node that owns the enclosing zone

37

CAN Example: Two Dimensional Space

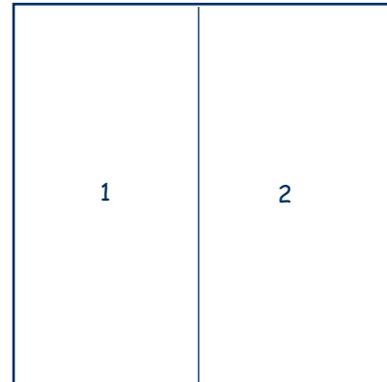
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



38

CAN Example: Two Dimensional Space

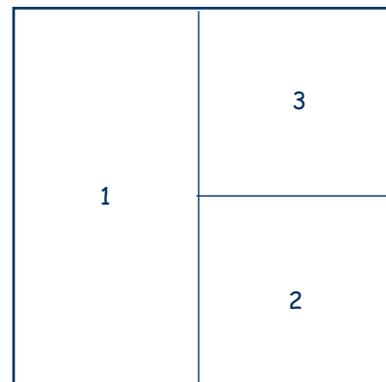
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



39

CAN Example: Two Dimensional Space

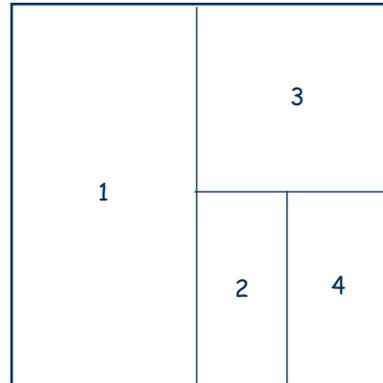
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



40

CAN Example: Two Dimensional Space

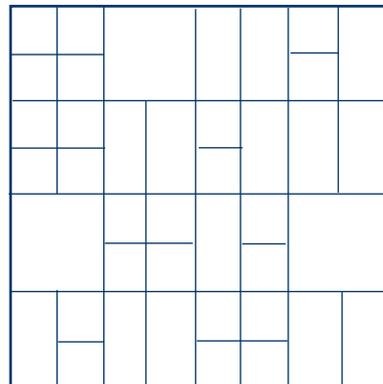
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



41

CAN Example: Two Dimensional Space

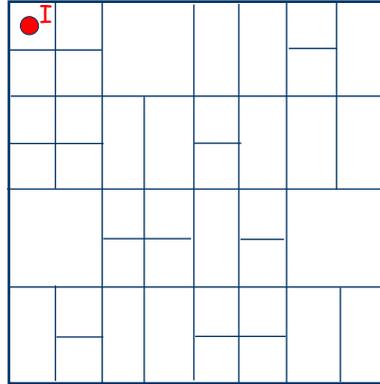
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



42

CAN Insert: Example (1)

node I::insert(K,V)

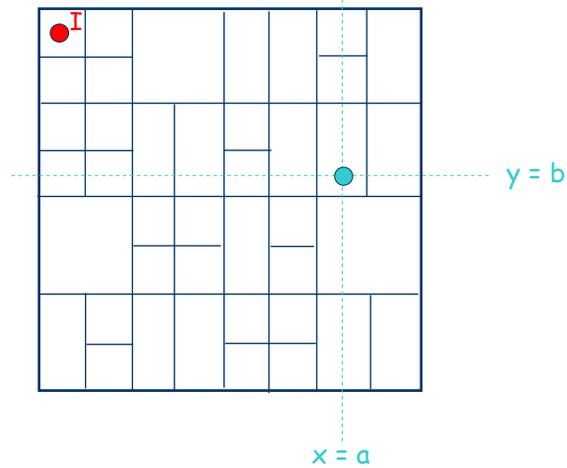


43

CAN Insert: Example (2)

node I::insert(K,V)

(1) $a = hx(K)$
 $b = hy(K)$



44

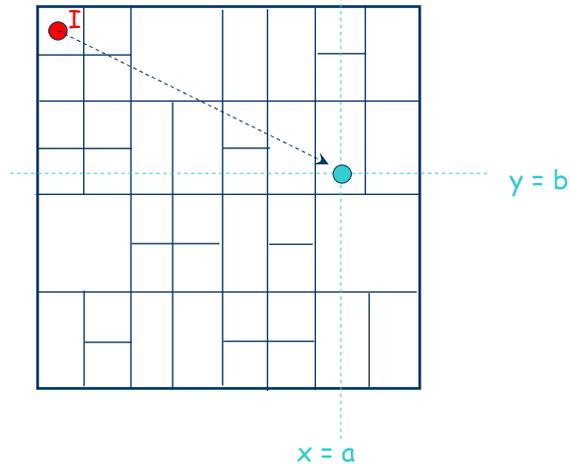
CAN Insert: Example (3)

node I::insert(K,V)

(1) $a = hx(K)$

$b = hy(K)$

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



45

CAN Insert: Example (4)

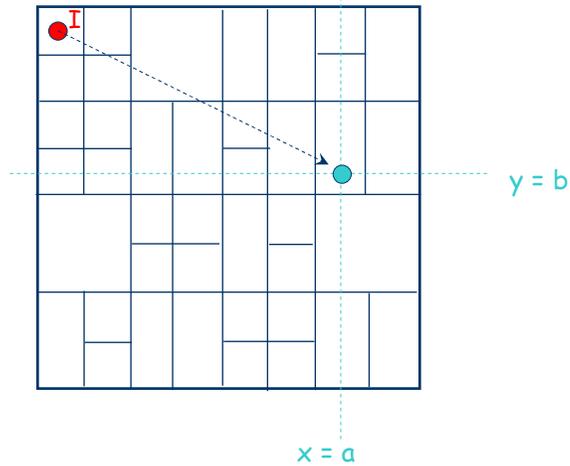
node I::insert(K,V)

(1) $a = hx(K)$

$b = hy(K)$

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

(3) (a,b) stores (K,V)



46

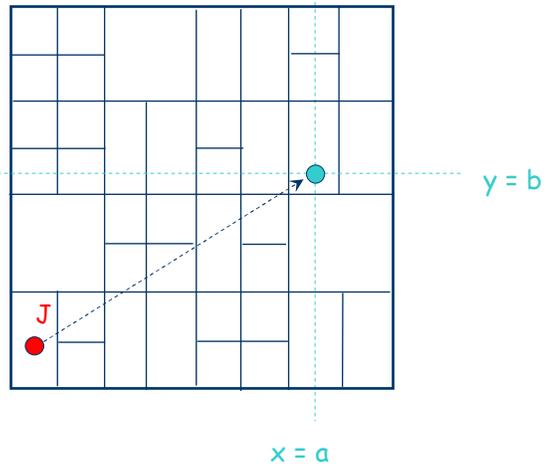
CAN Retrieve: Example

node J::retrieve(K)

(1) $a = hx(K)$

$b = hy(K)$

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



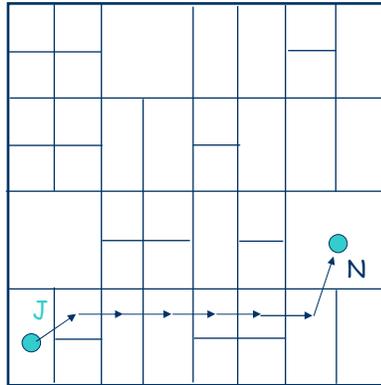
47

CAN Feature: Summary

- Data stored in the CAN is addressed by name (i.e. key), not location (i.e. IP address)
- Question: What is missing in the procedure?

48

CAN Insert: Join (2)



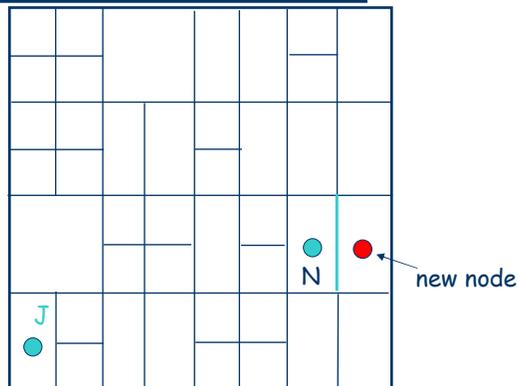
51

new node

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



52

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

CAN Node Failure

- Need to repair the space
 - takeover algorithm
 - when a node fails, one of its neighbors takes over its zone

Only the failed node's immediate neighbors are required for recovery

53

CAN Evaluations

- Guarantee 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 $2d$
 - 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 (more than one neighbour may lead to target, if this greedy strategy fails, resort to localized flooding)
 - can route around trouble

54

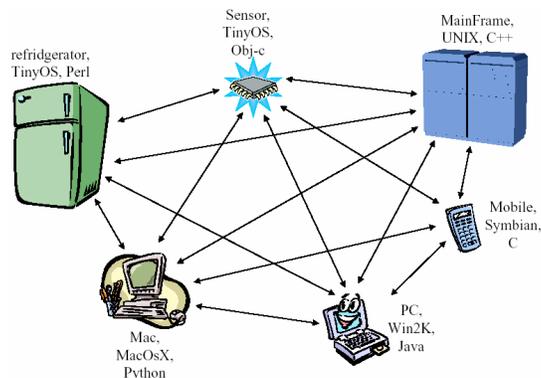
JXTA

- **Abbreviation of juxtaposition:**
 - “putting things next to each other”
- **Concept of Bill Joy (Chief Scientist, Sun)**
- **Fairly new concept**
 - First talk in 15.02.2001
 - Web-site (www.jxta.org) opened the 24.04.2001
- **Specification defining the basic concepts for creating P2P applications [sJxta02]**
- **An open-source project developed around the specification (Project JXTA)**

55

Aims of JXTA

- **Interoperability**
- **Platform independence**
- **Ubiquity**



56

JXTA Layers

- Core layer: monitoring communication, security, membership
- Service layer
- Application layer

APPLICATION

SERVICE

CORE

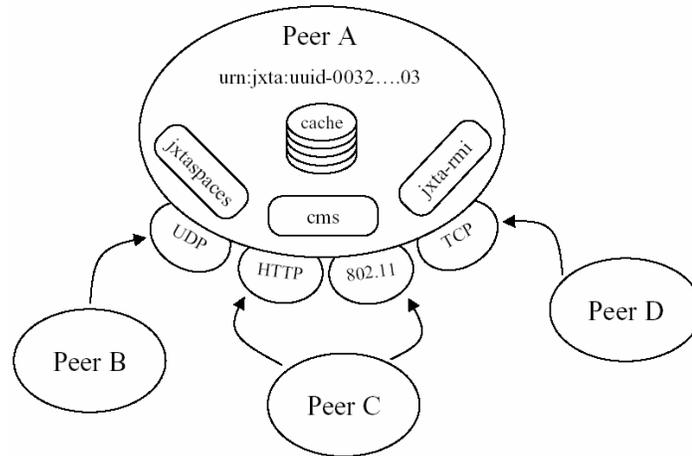
57

JXTA Peer I

- Peer
 - Any networked device using JXTA
 - Has a unique ID, allowing to be addressed independently of its physical location
 - Interact with each other spontaneously
 - May provide network services
 - May cache information
 - No assumption about availability
 - May have multiple network interfaces (for sending/receiving data)
 - Interact with a small number of peers

58

JXTA Peer II



59

JXTA Peer Groups

- Collection of peers sharing a common interest
- Create a secure, scoped and monitored environment
- Provide a set of core services
 - Discovery service
 - Membership service
 - Pipe service
 - Resolver service
 - Monitoring service
- A peer may join several peer groups
- All peers belong to the « world peer group »

60

Summary

- Peer-to-Peer networks are dynamic environments that facilitate resource sharing on a large-scale
- Main research question is how to organize and retrieve information efficiently and accurately
- Current systems use two methods: centralization and/or structure
- Focus gradually moves towards the coordinated use of versatile, distributed computing resources

BUT

Isn't this what GRIDs are all about ???