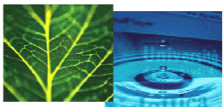


Structured vs unstructured systems

Alberto Montresor



Summary

Structured vs unstructured

Example, unstructured: the good(?) old Gnutella

Example, structured

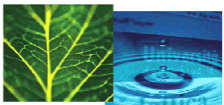
CAN

Chord

Kademlia

Comparisons

Topology bootstrap

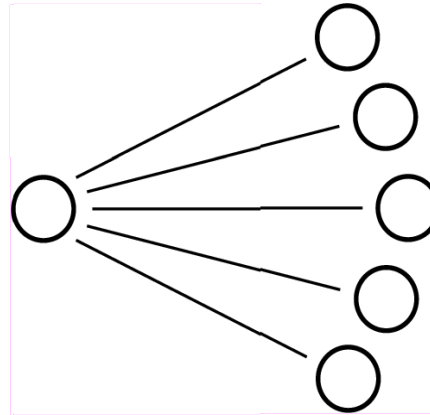


P2P Topologies

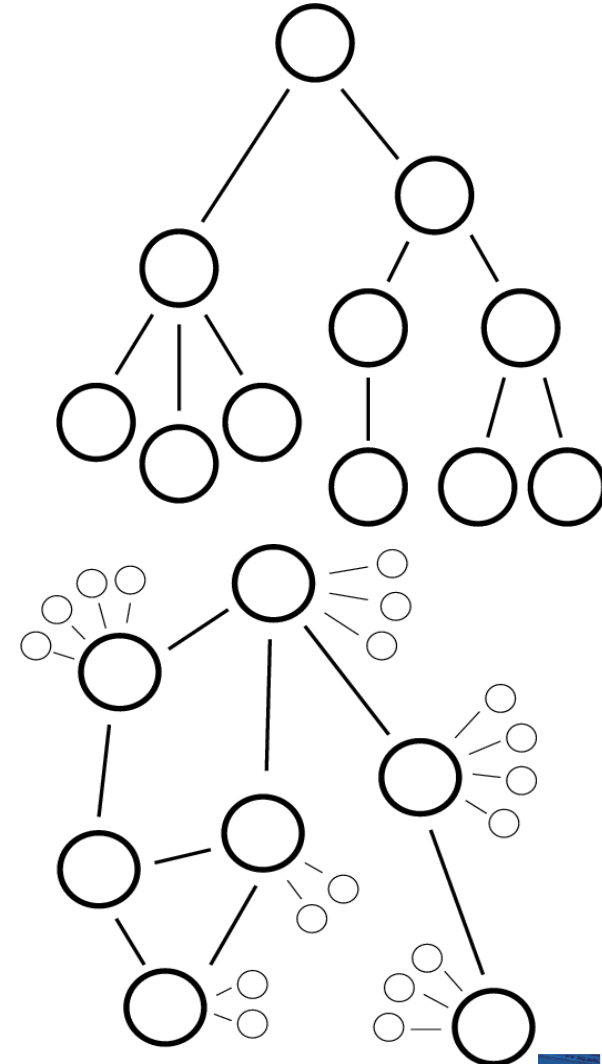
□ Unstructured

- No explicit topology
- *Observed rather than engineered*
- Example:
 - Gnutella, Freenet

Centralized

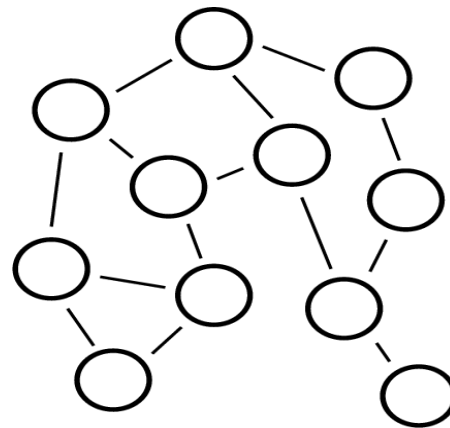


Hierarchical



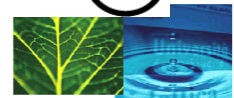
□ Structured

- *An explicit “shape” is maintained*
- Examples:
 - Rings, Trees, DHTs
 - But also random topologies are “structured”



Decentralized

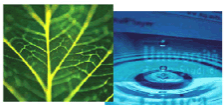
Hybrid



P2P Topologies

□ Criteria for topology selection

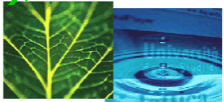
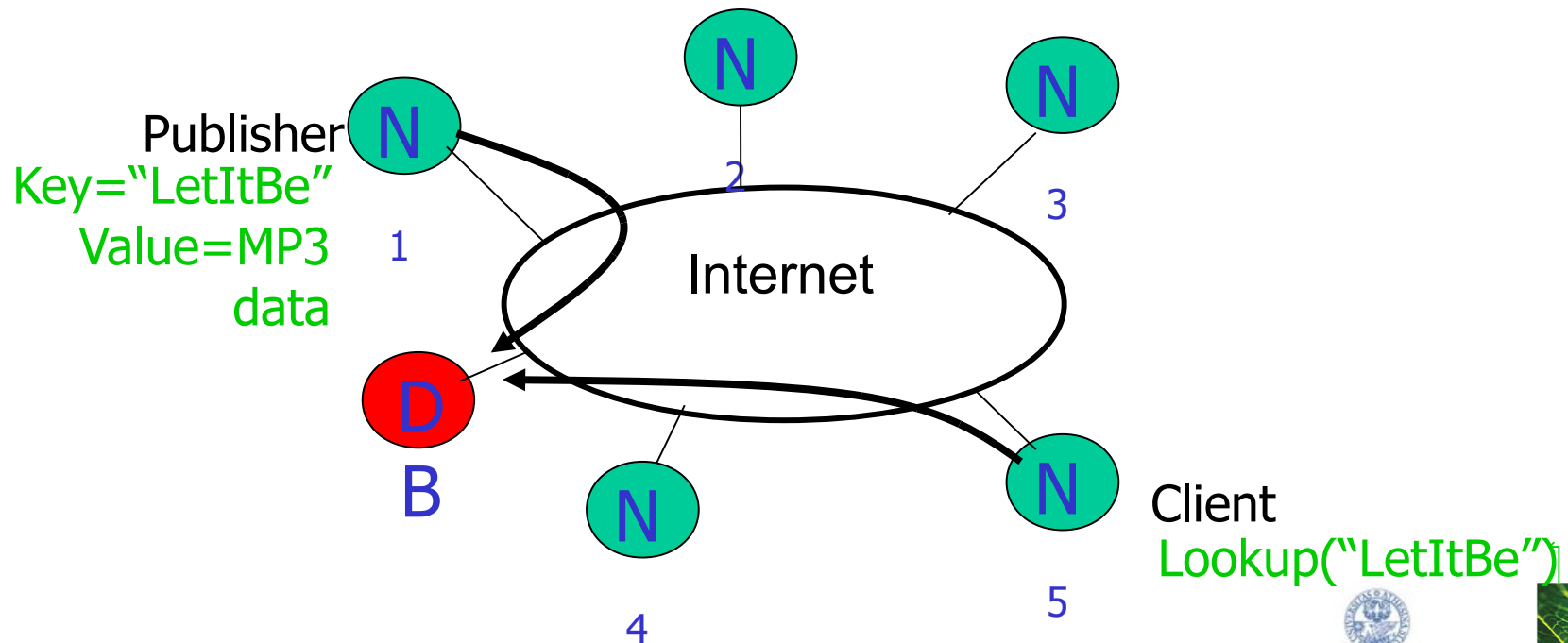
- Does it simplify location of data?
- Does it
 - balance the load, if nodes are equal?
 - exploit heterogeneity, otherwise?
- Is it robust?
 - Can it work if part of it is suddenly removed?
 - Can it be maintained in spite of churn?
- Has some correspondence with the underlying network topology?
 - Proximity (latency-based)
 - For example, Pastry, Kazaa, Skype



Lookup: Data location

❑ Centralized solution (e.g. Napster)

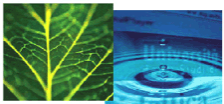
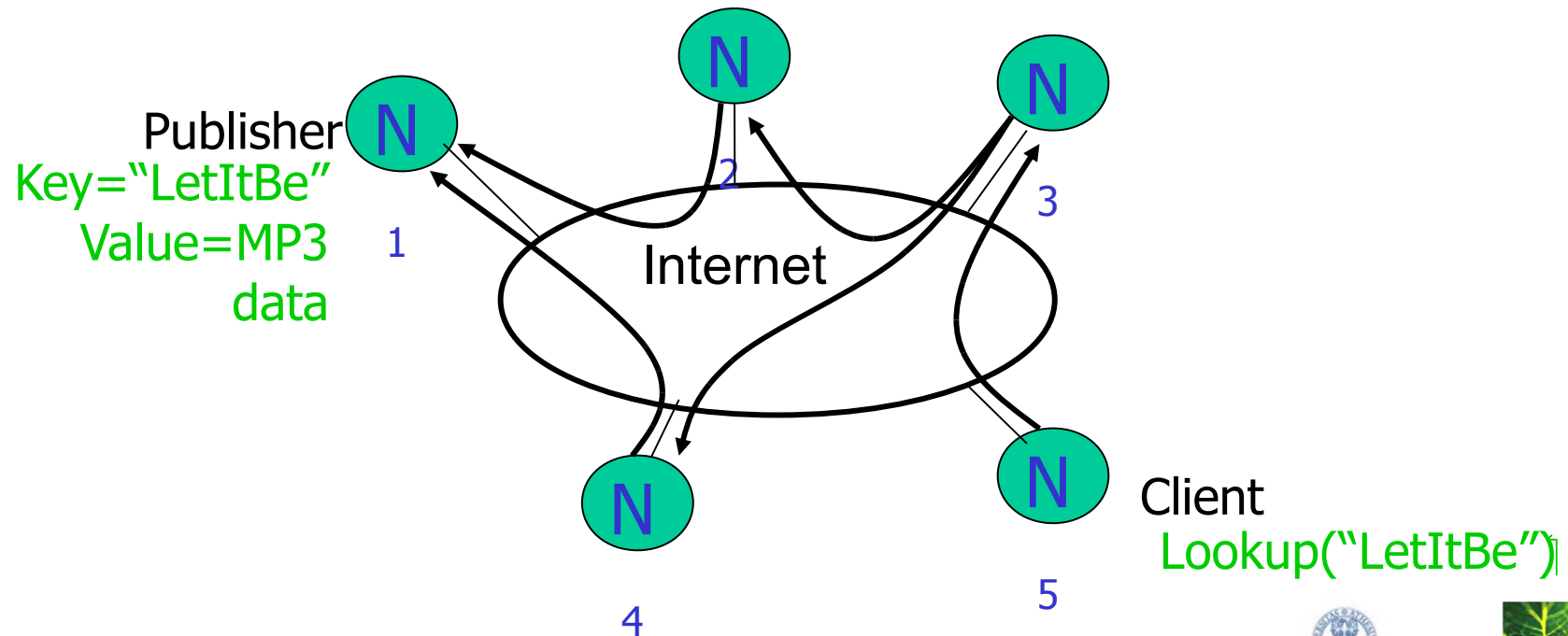
- Requires $O(N)$ state
- Single point of failure, bottleneck



Lookup: Data location

□ Flooding (e.g., Gnutella)

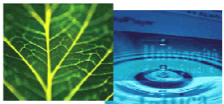
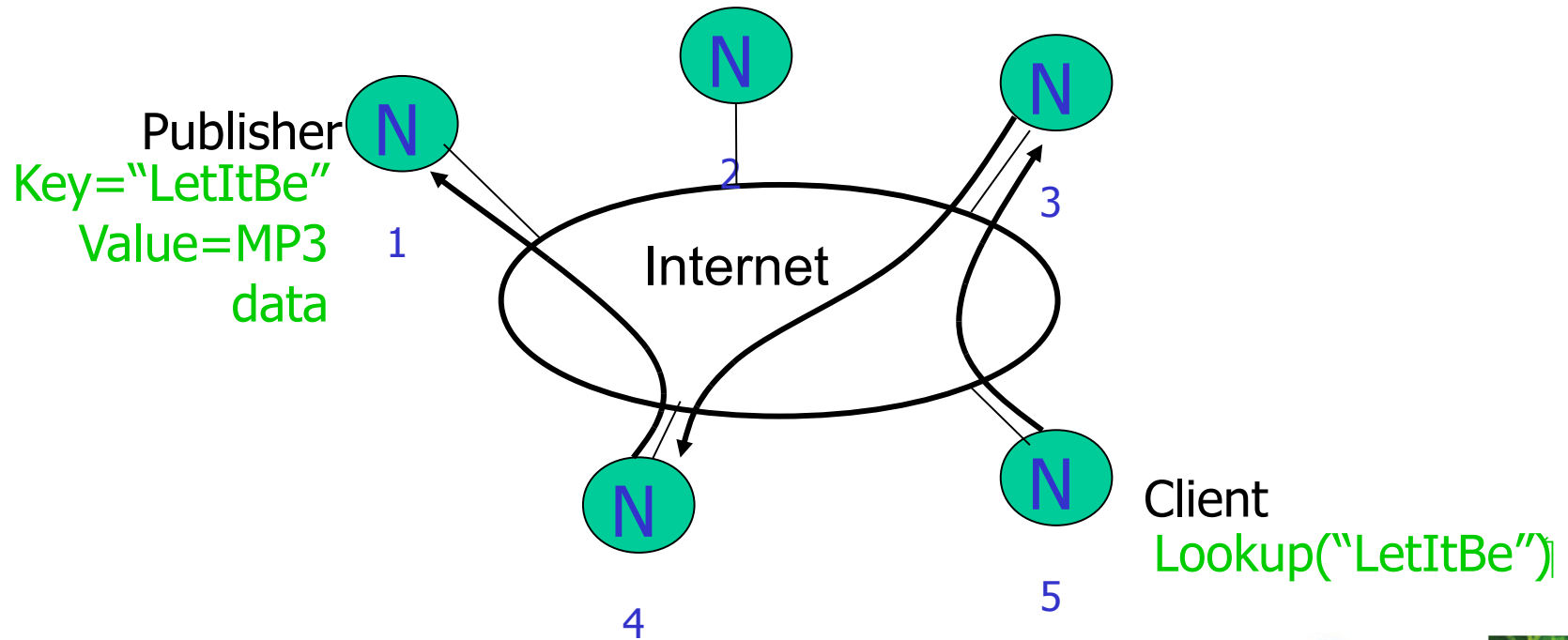
- Worst case $O(N)$ messages per lookup



Lookup: Data location

□ Routed messages (Freenet, Tapestry, Chord, CAN, etc.)

- Only exact matches
- $O(\log N)$, $O(N^{1/d})$ messages per lookup



Gnutella

The Gnutella protocol consists of:

A set of *message formats*

5 basic message types

A set of *rules* governing the inter-servent exchange of messages

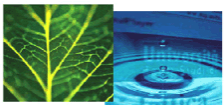
How to connect to a Gnutella network:

Handshaking:

A Gnutella servent connects to the network by establishing a connection with another servent currently on the network

The acquisition of another servent's address is not part of the protocol definition

Hostcache



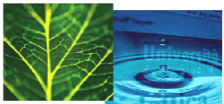
Gnutella: Two types of messages

Broadcast

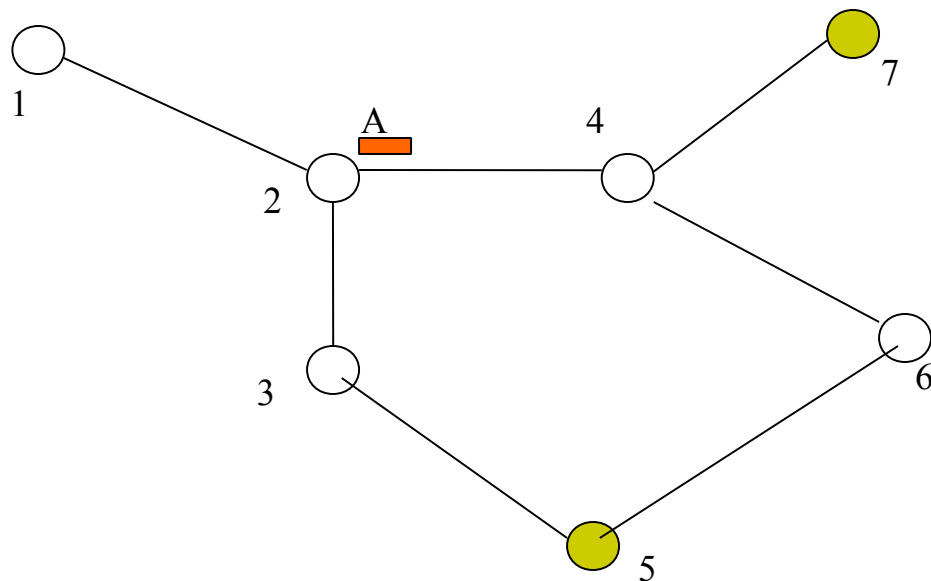
- Sent to all nodes with which the sender has open TCP connections
- Note: serious scalability problems

Back-propagate

- Sent on a specific connection on the reverse of the path taken by an initial broadcasted message

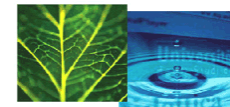


Gnutella search mechanism

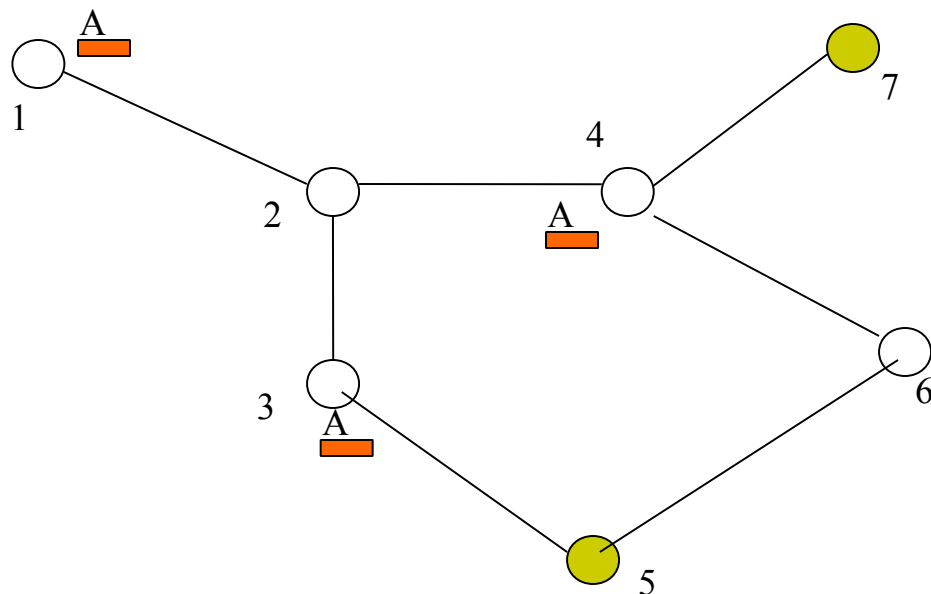


Steps:

- Node 2 initiates search for file A

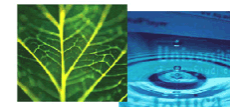


Gnutella search mechanism

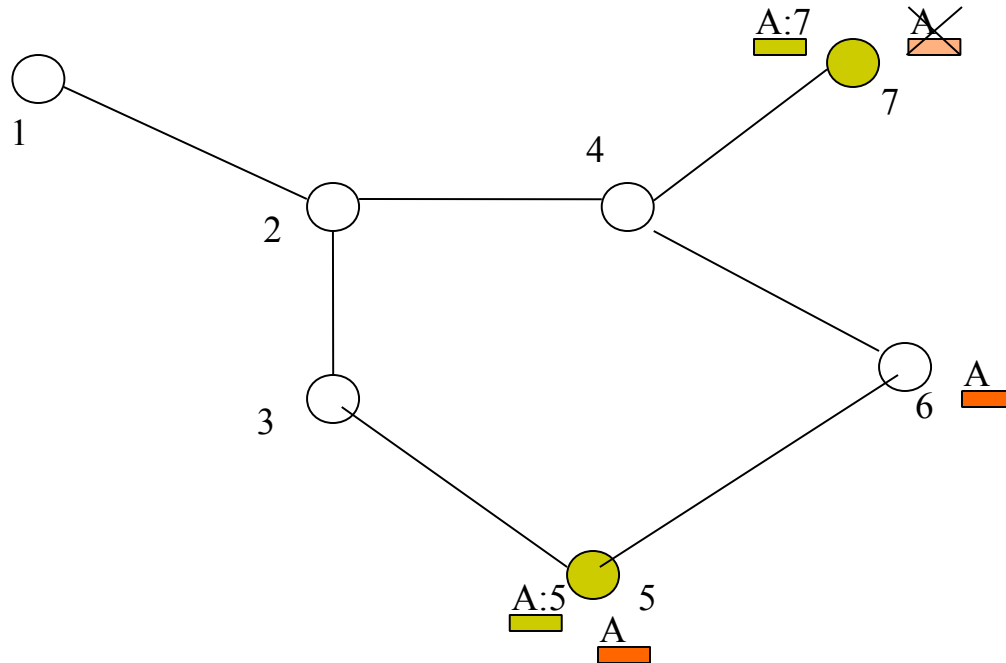


Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors

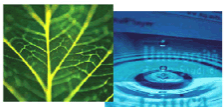


Gnutella search mechanism

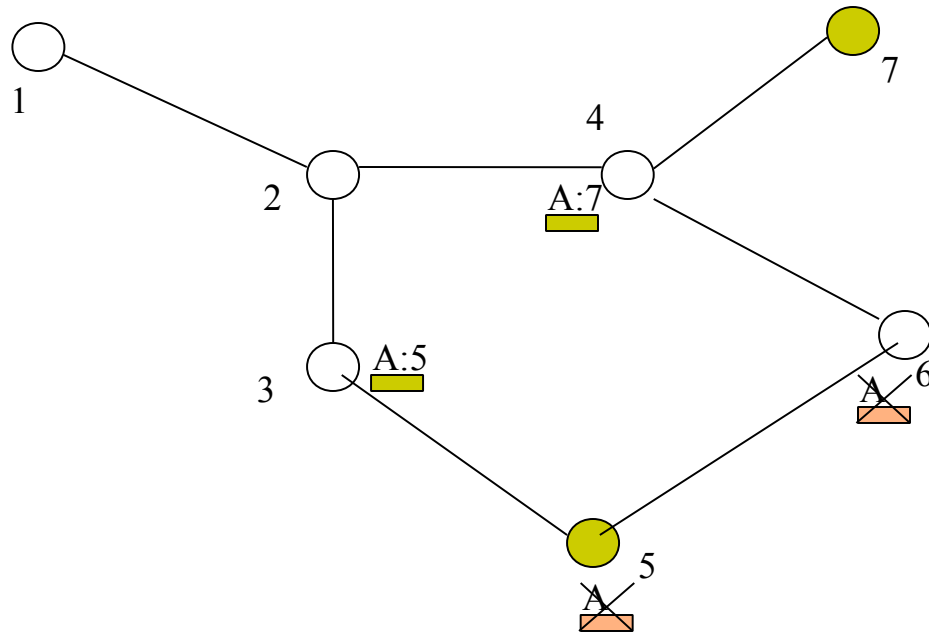


Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message

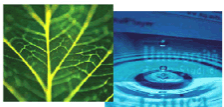


Gnutella search mechanism

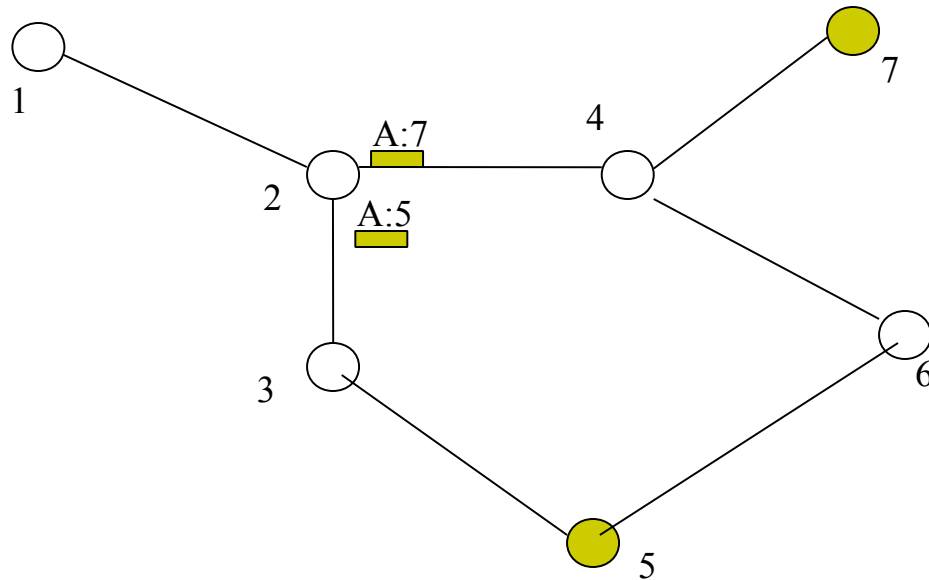


Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message
- Query reply message is back-propagated

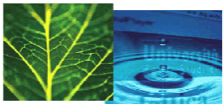


Gnutella search mechanism



Steps:

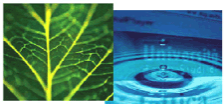
- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message
- Query reply message is back-propagated



Gnutella Messages

□ Each message is composed of:

- A 16-byte ID field uniquely identifying the message
 - randomly generated
 - not related to the address of the requester (anonymity)
 - used to detect duplicates and route back-propagate messages
- A message type field
 - PING, PONG
 - QUERY, QUERYHIT
 - PUSH (for firewalls)
- A Time-To-Live (TTL) Field
- Hops Field (inverse of TTL)
- Payload length



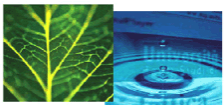
Gnutella Message Types

❑ PING (broadcast)

- Used to maintain information about the nodes currently in the network
- Originally, a "who's there" flooding message
- A server receiving a PING is expected to respond with a PONG message

❑ PONG (back-propagate)

- A PONG message has the same ID of the corresponding PING message
- Contains:
 - address of connected Gnutella server
 - total size and total number of files shared by this server



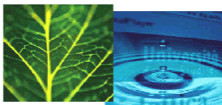
Gnutella Message Types

❑ QUERY (broadcast)

- The primary mechanism for searching the distributed network
- Contains the query string
- A server is expected to respond with a QUERYHIT message if a match is found against its local data set

❑ QUERYHIT (back-propagate)

- The response to a query
- Has the same ID of the corresponding QUERY message
- Contains enough information to acquire the data matching the corresponding query
 - IP Address + port number
 - List of file names



Gnutella Pros and Cons

Several problems in Gnutella 0.4 (the original one):

❑ What kind of topology is generated?

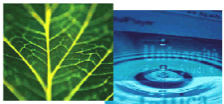
- Is it planned (“engineered”)?
- Is it good?

❑ Ping-pong traffic

- More than 50% of the traffic generated by Gnutella 0.4 is Ping/Pong related

❑ Scalability

- Each query generates a huge amount of traffic
- Potentially, each query is received multiple times from all neighbors



Gnutella Overlay Topology vs Underlying Network

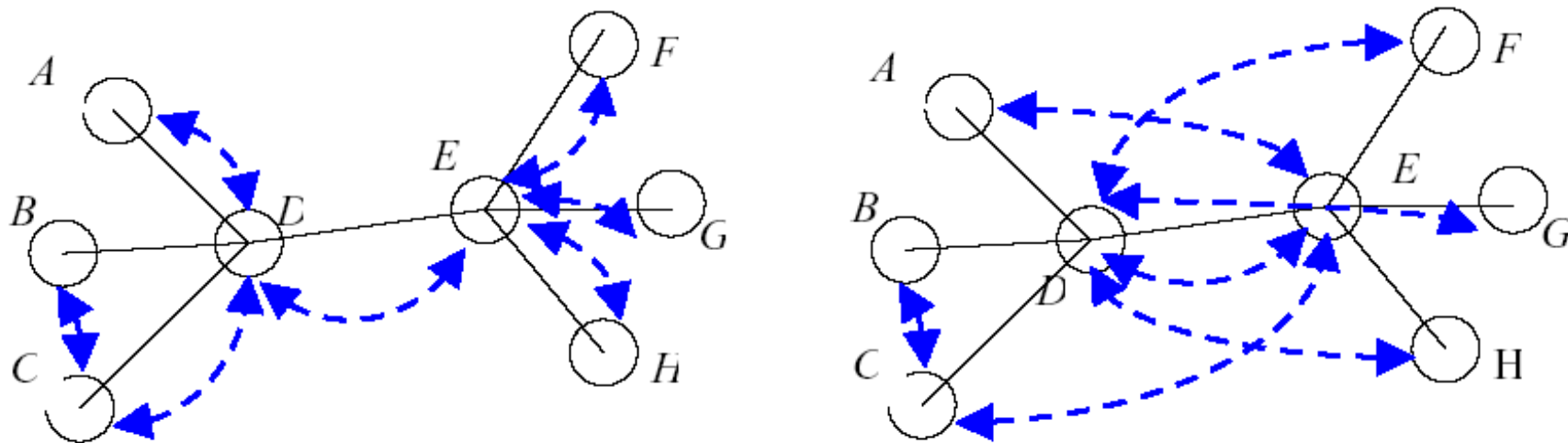
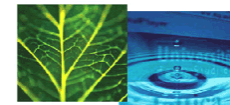


Figure 7: Two different mappings of Gnutella's virtual network topology (blue, dotted arrows) to the underlying network infrastructure (black, solid lines). Left picture: perfect mapping. A message inserted into the network by node A travels physical link D-E only once to reach all other nodes. Right picture: inefficient mapping. The same distribution requires that the message traverse physical link D-E six times.

figure 7 from Ripeanu, Iamnitchi & Foster,
IEEE IC, 6(1), 2002



Traffic

figure 2 from Ripeanu, Iamnitchi & Foster,
IEEE IC, 6(1), 2002

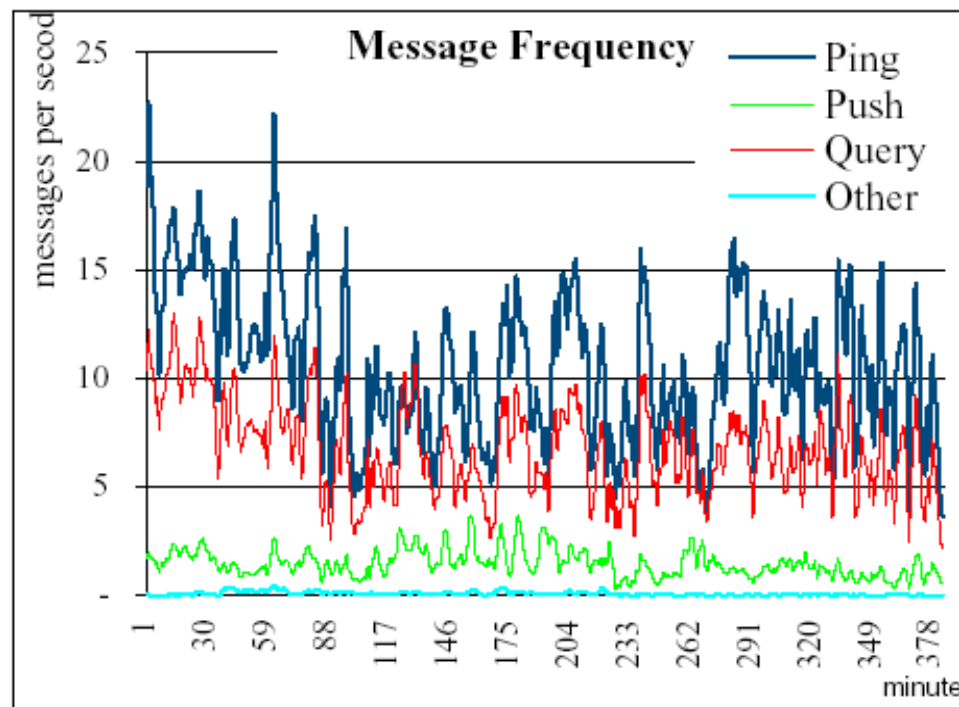
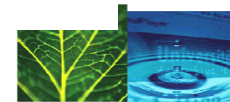


Figure 2: Generated traffic (messages/sec) in Nov. 2000 classified by message type over a 376 minute period. Note that overhead traffic (PING messages, that serve only to maintain network connectivity) formed more than 50% of the traffic. The only 'true' user traffic is QUERY messages. Overhead traffic has decreased by May 2001 to less than 10% of all generated traffic.



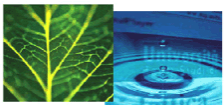
Gnutella: Conclusions

- ❑ A milestone in P2P computing

- Gnutella proved that “there was life after Napster”

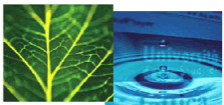
- ❑ But

- Gnutella is a patchwork of hacks
- The ping-pong mechanism, even with caching, is just “plain inefficient”



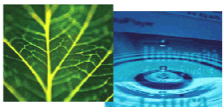
Distributed Hash Tables (DHTs)

- Abstraction: a distributed hash-table data structure
 - `put(key, item)`
 - `item = get(key)` (or `lookup(key)`)
 - Note: item can be anything: a data object, document, file, pointer to a file...
- Proposals
 - CAN, Chord, Kademlia, Pastry, Tapestry, etc



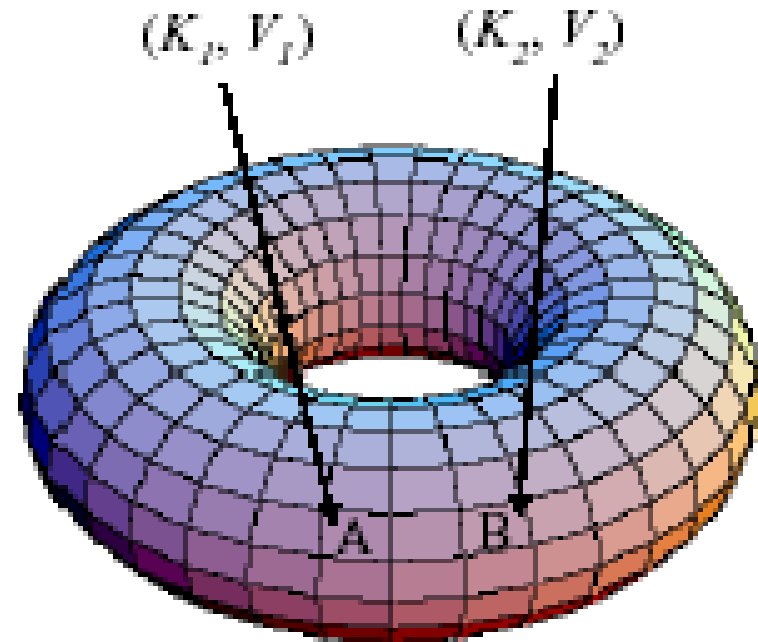
DHT Design Goals

- Make sure that an item (file) identified is always found
- Scales to hundreds of thousands of nodes
- Handles rapid arrival and failure of nodes

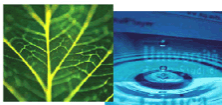


Content Addressable Network (CAN)

- Associate to each node and item a unique id in an d -dimensional Cartesian space on a d -torus
- Properties
 - Routing table size $O(d)$
 - Guarantees that a file is found in at most $d * n^{1/d}$ steps, where n is the total number of nodes

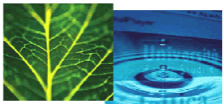
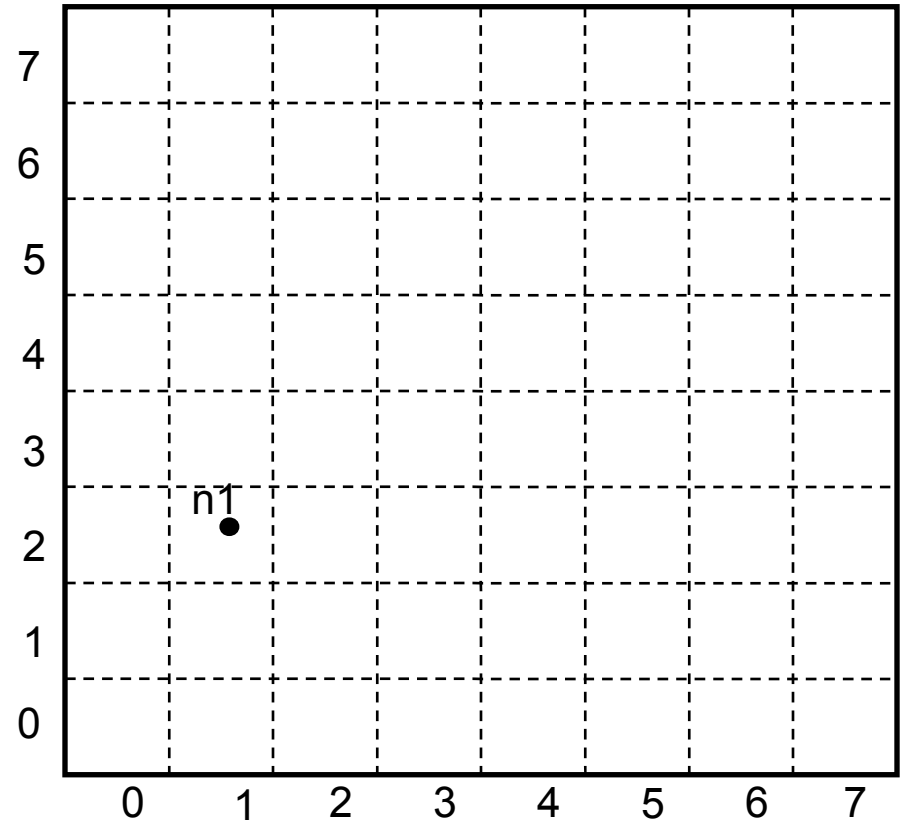


2-torus



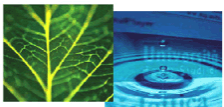
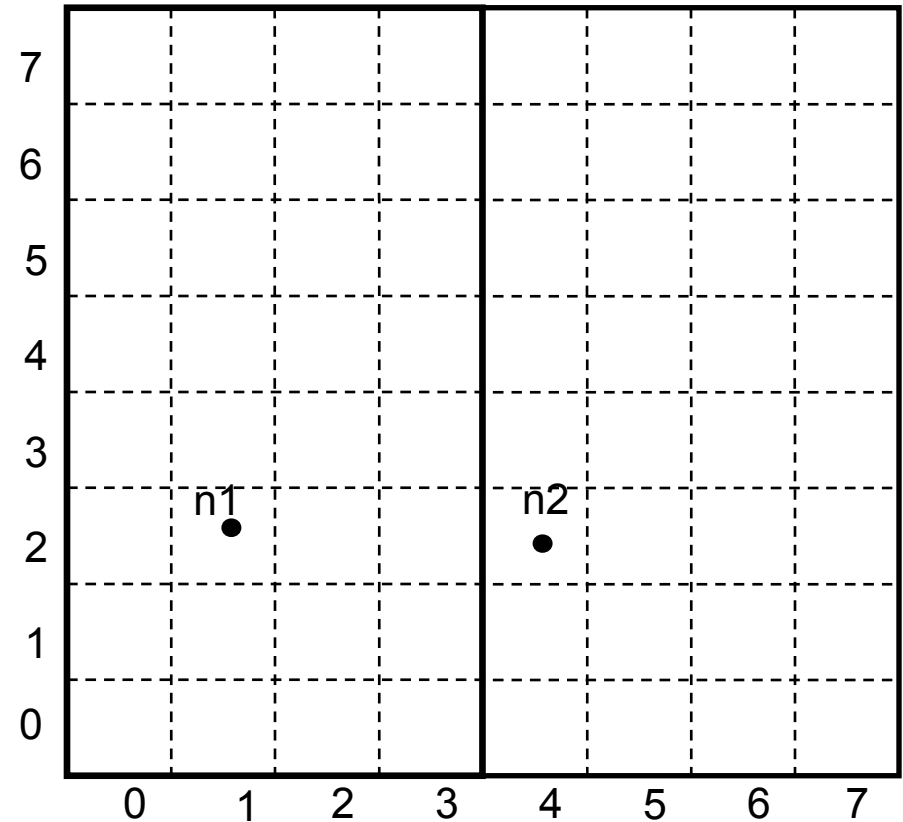
CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
 - Node n1:(1, 2) first node that joins
→ cover the entire space



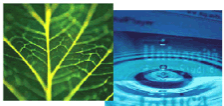
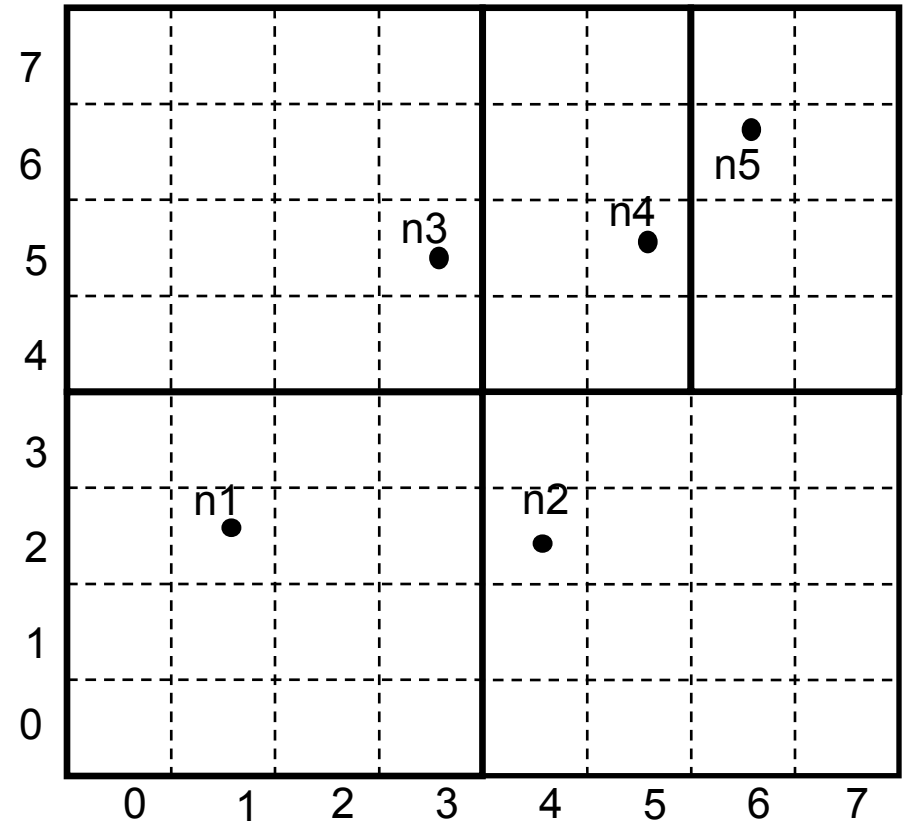
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins \rightarrow space is divided between n1 and n2



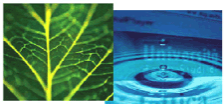
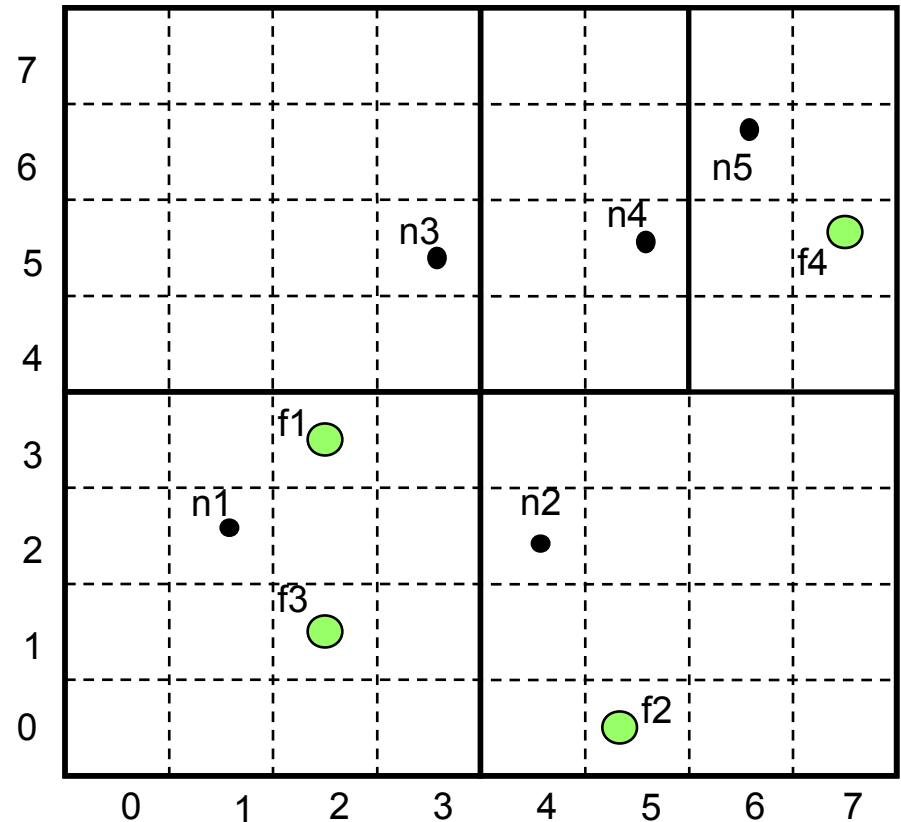
CAN Example: Two Dimensional Space

- Nodes $n4:(5, 5)$ and $n5:(6,6)$ join



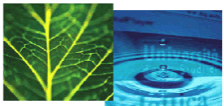
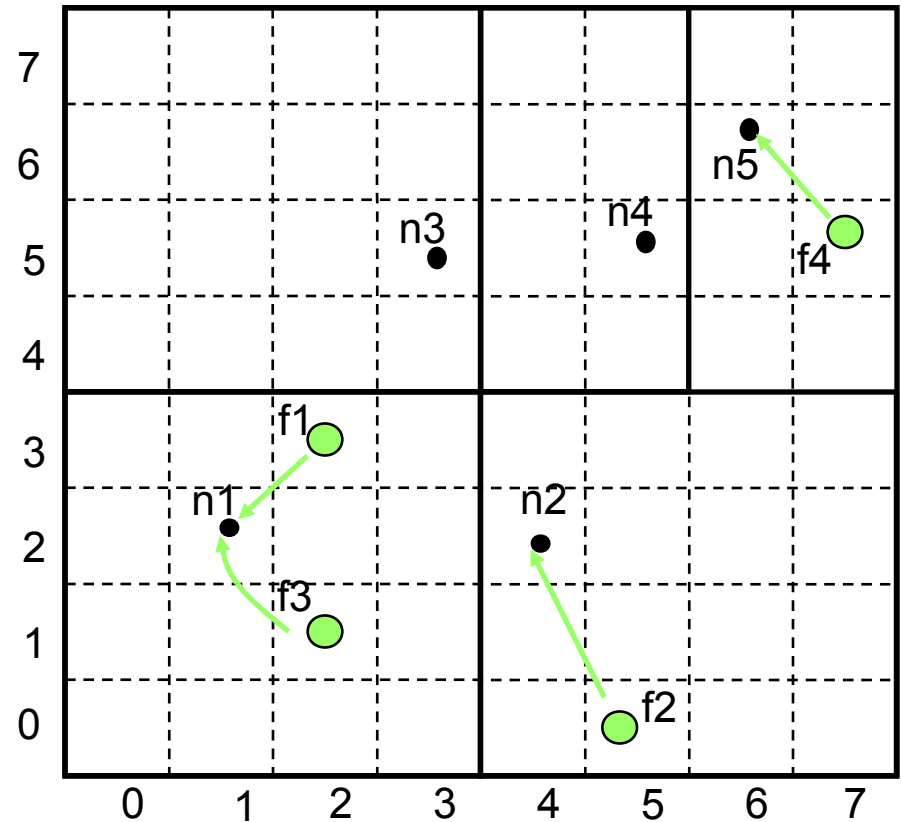
CAN Example: Two Dimensional Space

- Nodes: $n1:(1, 2)$; $n2:(4,2)$; $n3:(3, 5)$; $n4:(5,5)$; $n5:(6,6)$
- Items: $f1:(2,3)$; $f2:(5,1)$; $f3:(2,1)$; $f4:(7,5)$

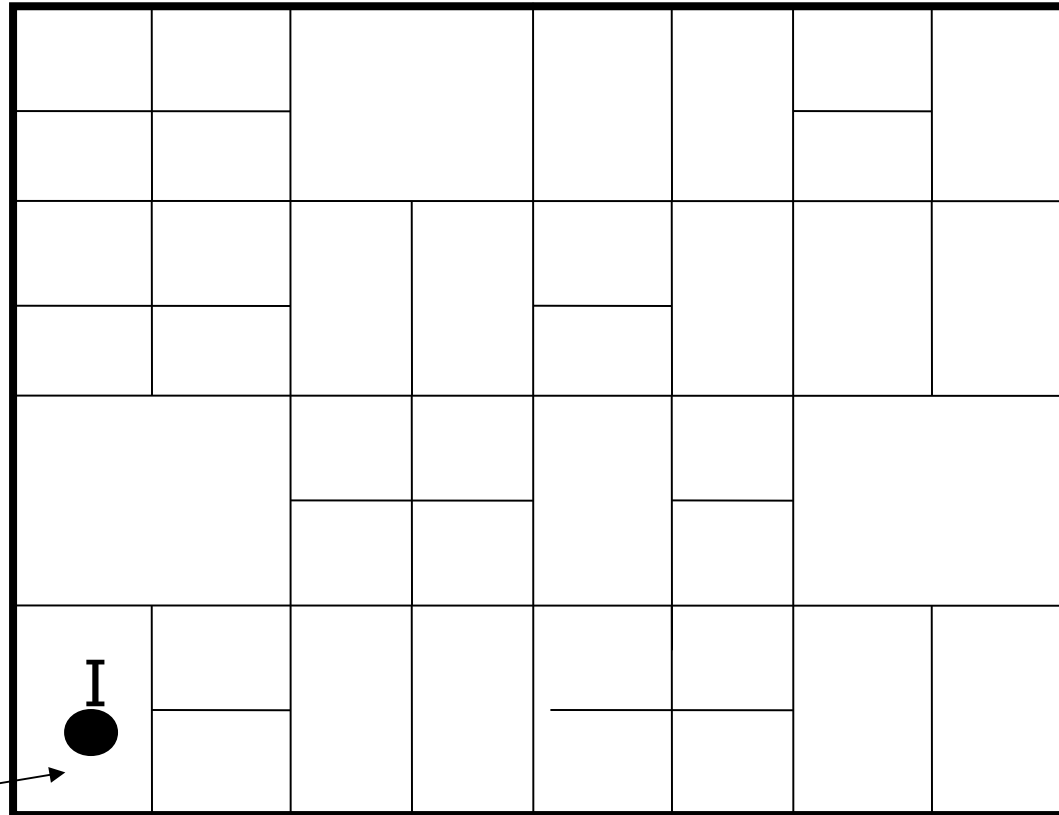


CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space

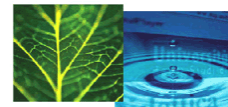


CAN: Node Joining

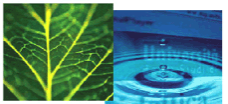
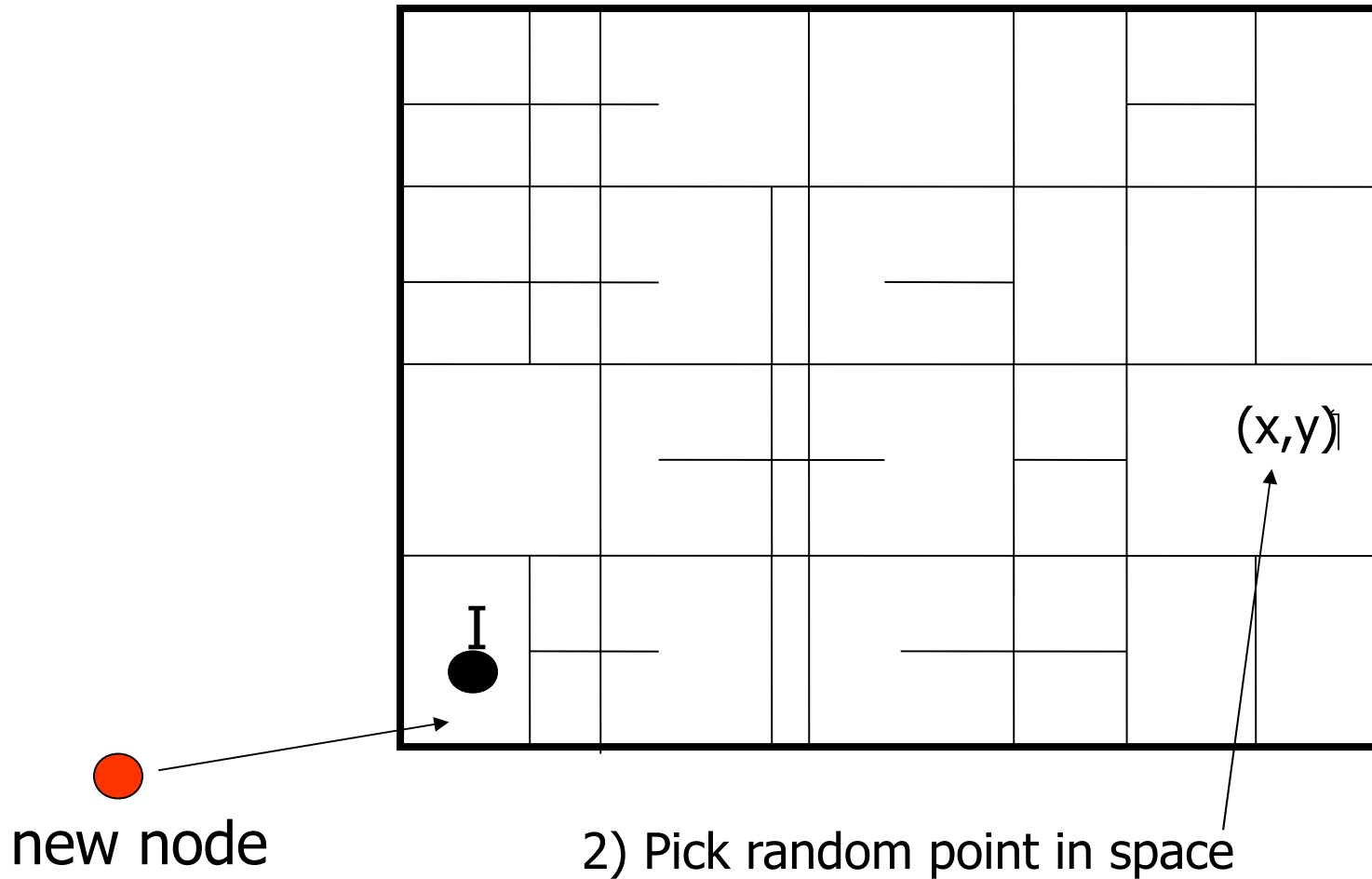


new node

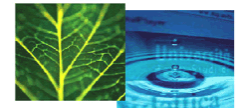
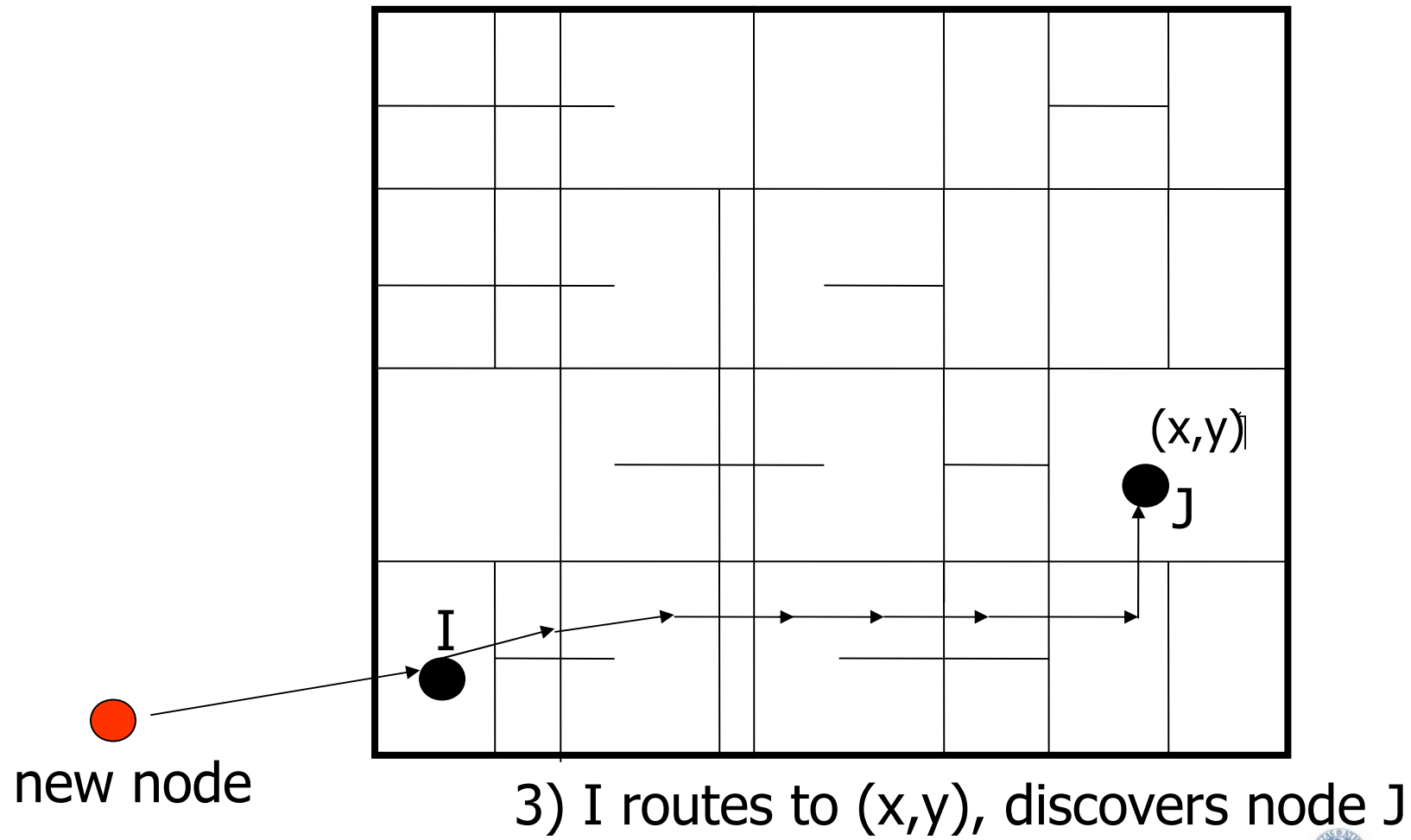
1) Discover some node "I" already in CAN



CAN: Node Joining

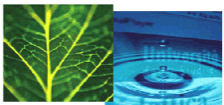


CAN: Node Joining



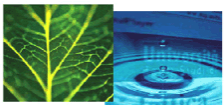
Node departure

- Node explicitly hands over its zone and the associated (key,value) database to one of its neighbors
- In case of network failure this is handled by a take-over algorithm
- Problem : take over mechanism does not provide regeneration of data
- Solution:
every node has a backup of its neighbours



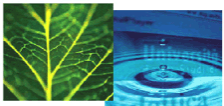
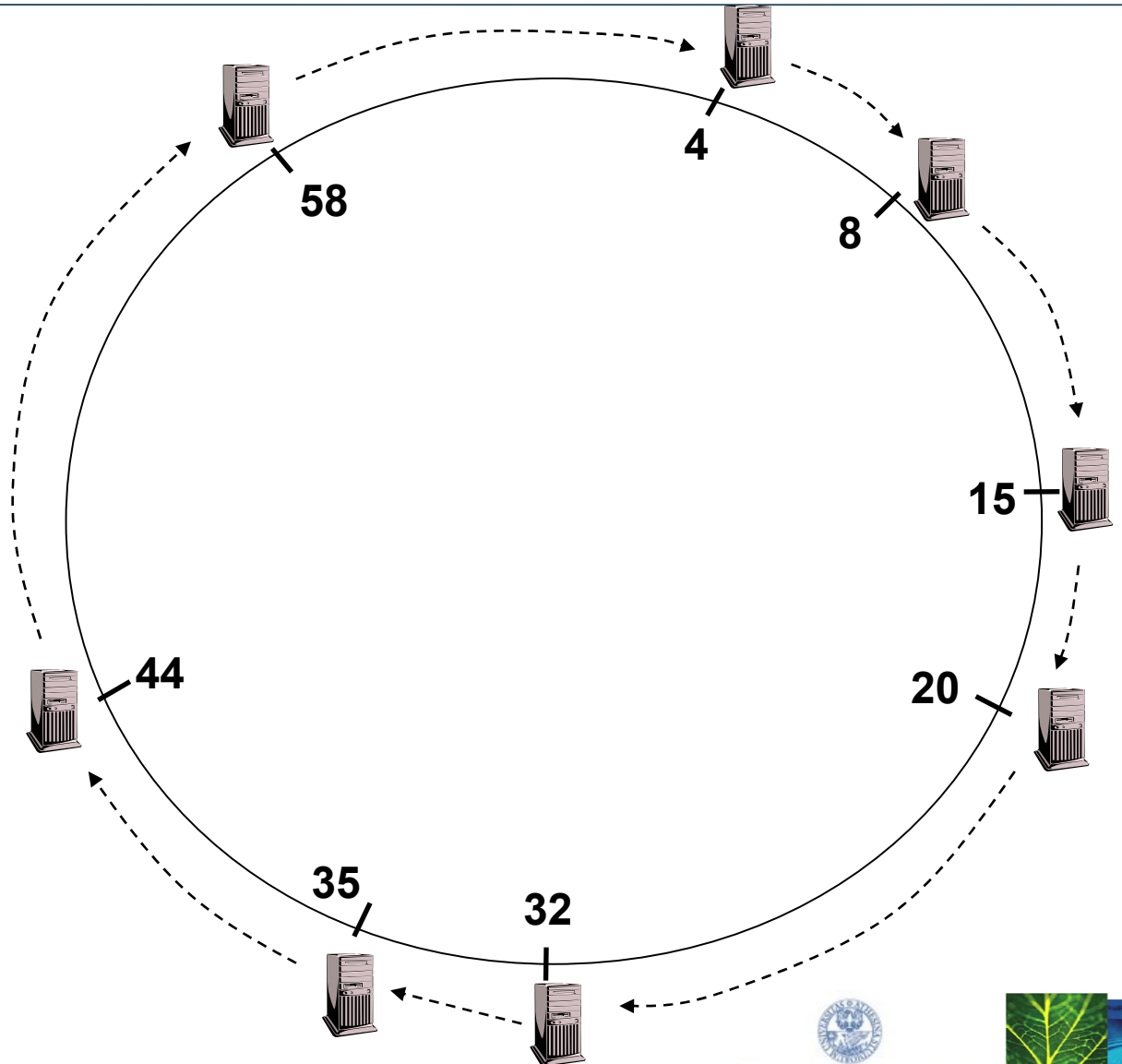
Chord

- Associate to each node and item a unique *id* in an *uni*-dimensional space $0..2^m-1$
- Key design decision
 - Decouple correctness from efficiency
- Properties
 - Routing table size $O(\log(N))$, where N is the total number of nodes
 - Guarantees that a file is found in $O(\log(N))$ steps



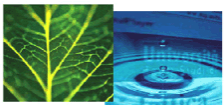
Identifier to Node Mapping Example

- Node 8 maps [5,8]
 - Node 15 maps [9,15]
 - Node 20 maps [16, 20]
 - ...
 - Node 4 maps [59, 4]
-
- Each node maintains a pointer to its successor



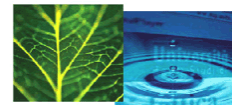
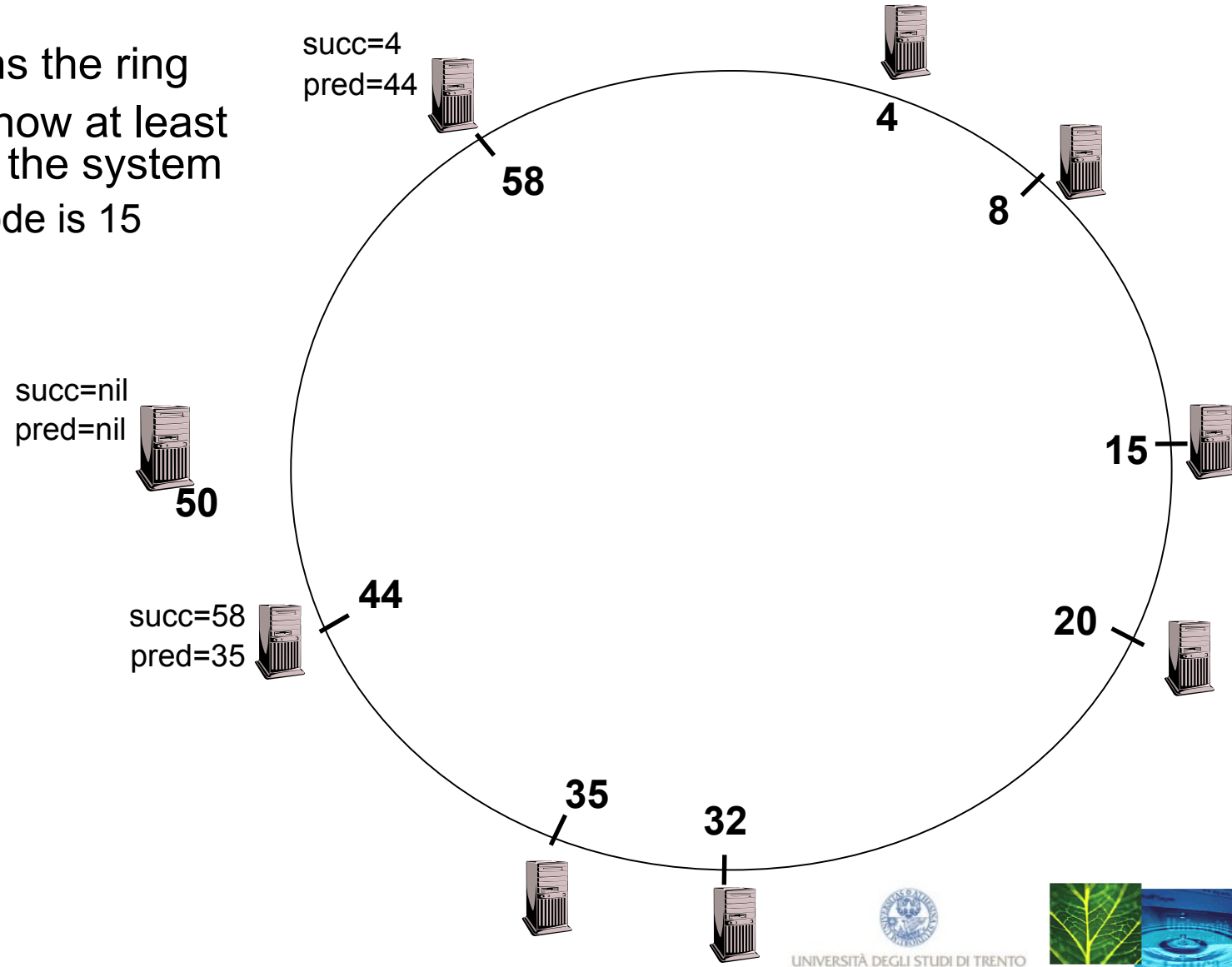
Joining Operation

- Each node A periodically sends a **stabilize()** message to its successor B
- Upon receiving a **stabilize()** message, node B
 - returns its predecessor $B' = \text{pred}(B)$ to A by sending a **notify(B')** message
- Upon receiving **notify(B')** from B,
 - if B' is between A and B, **A updates its successor to B'**
 - A doesn't do anything, otherwise



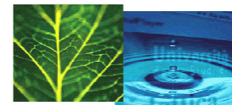
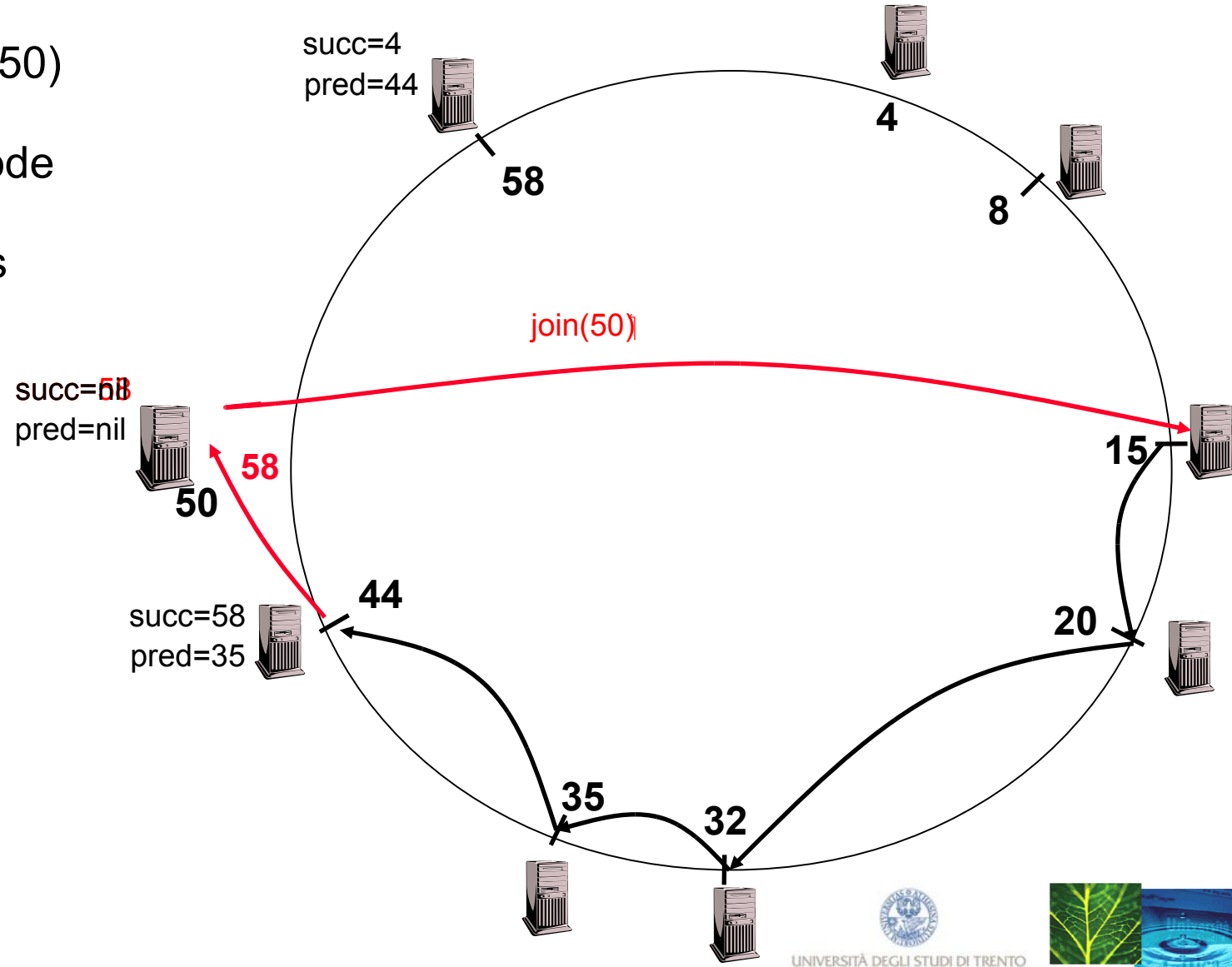
Joining Operation

- Node with id=50 joins the ring
- Node 50 needs to know at least one node already in the system
 - Assume known node is 15



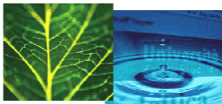
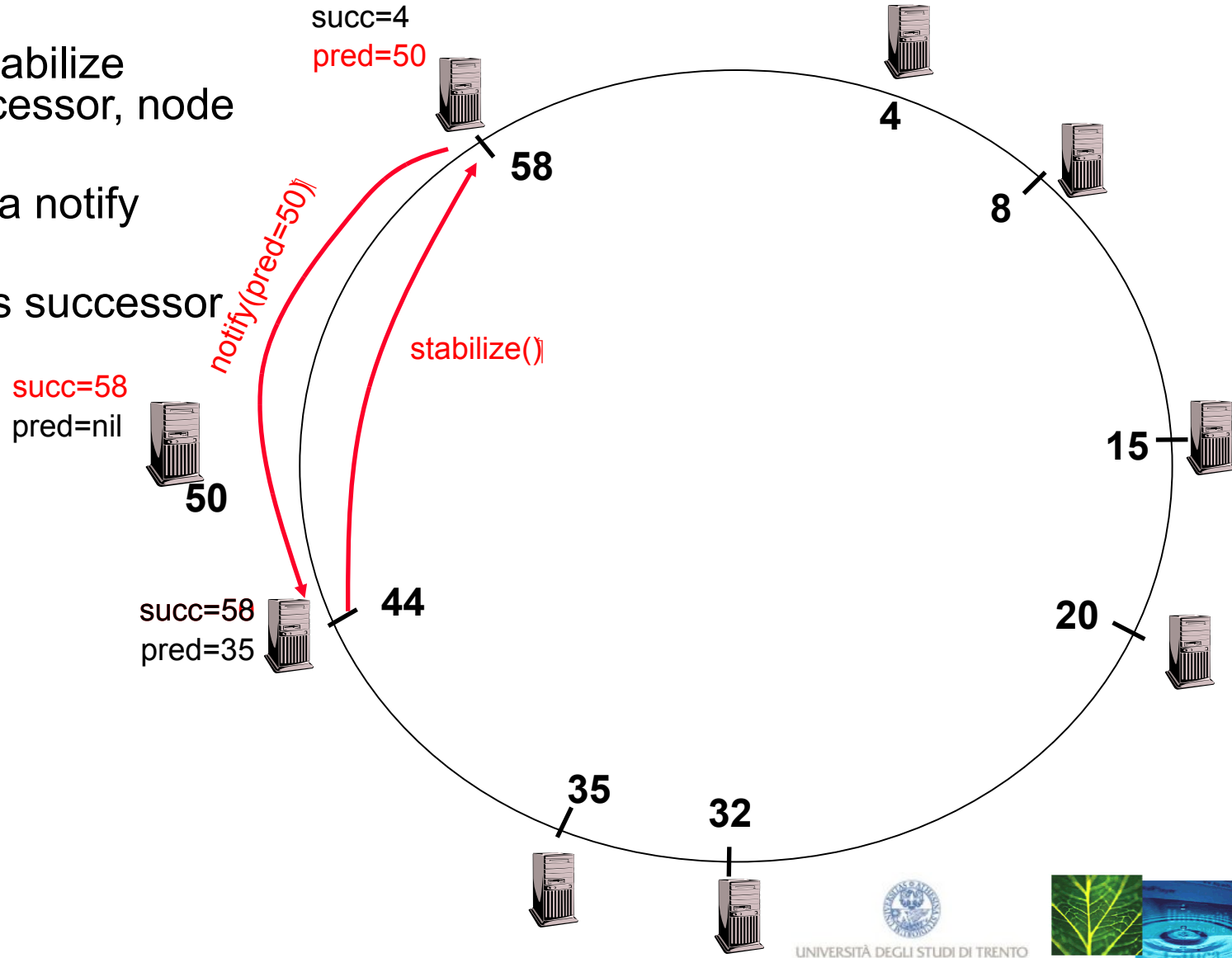
Joining Operation

- Node 50: send join(50) to node 15
- Node 44: returns node 58
- Node 50 updates its successor to 58



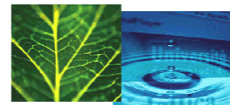
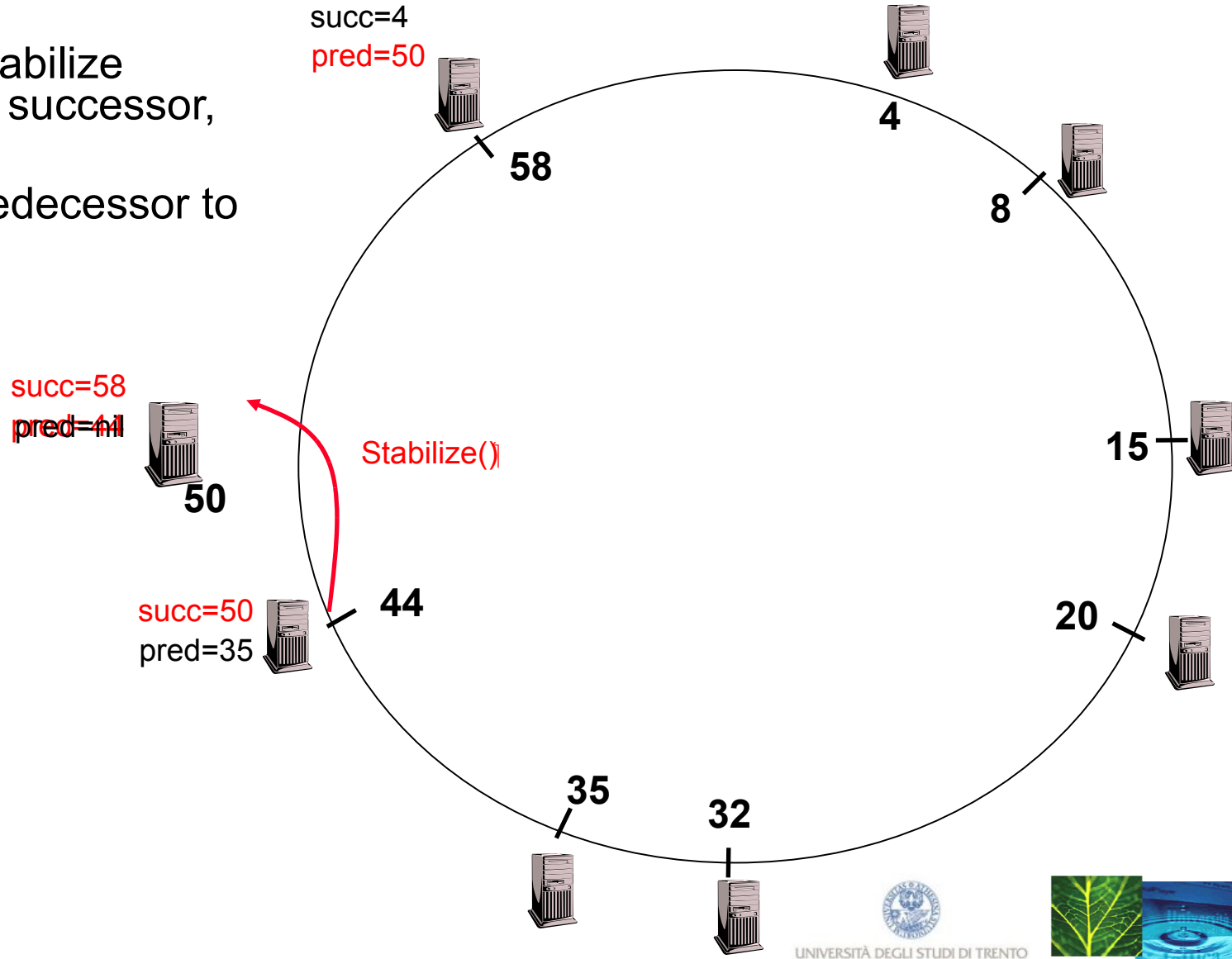
Joining Operation (cont'd)

- Node 44 sends a stabilize message to its successor, node 58
- Node 58 reply with a notify message
- Node 44 updates its successor to 50



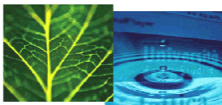
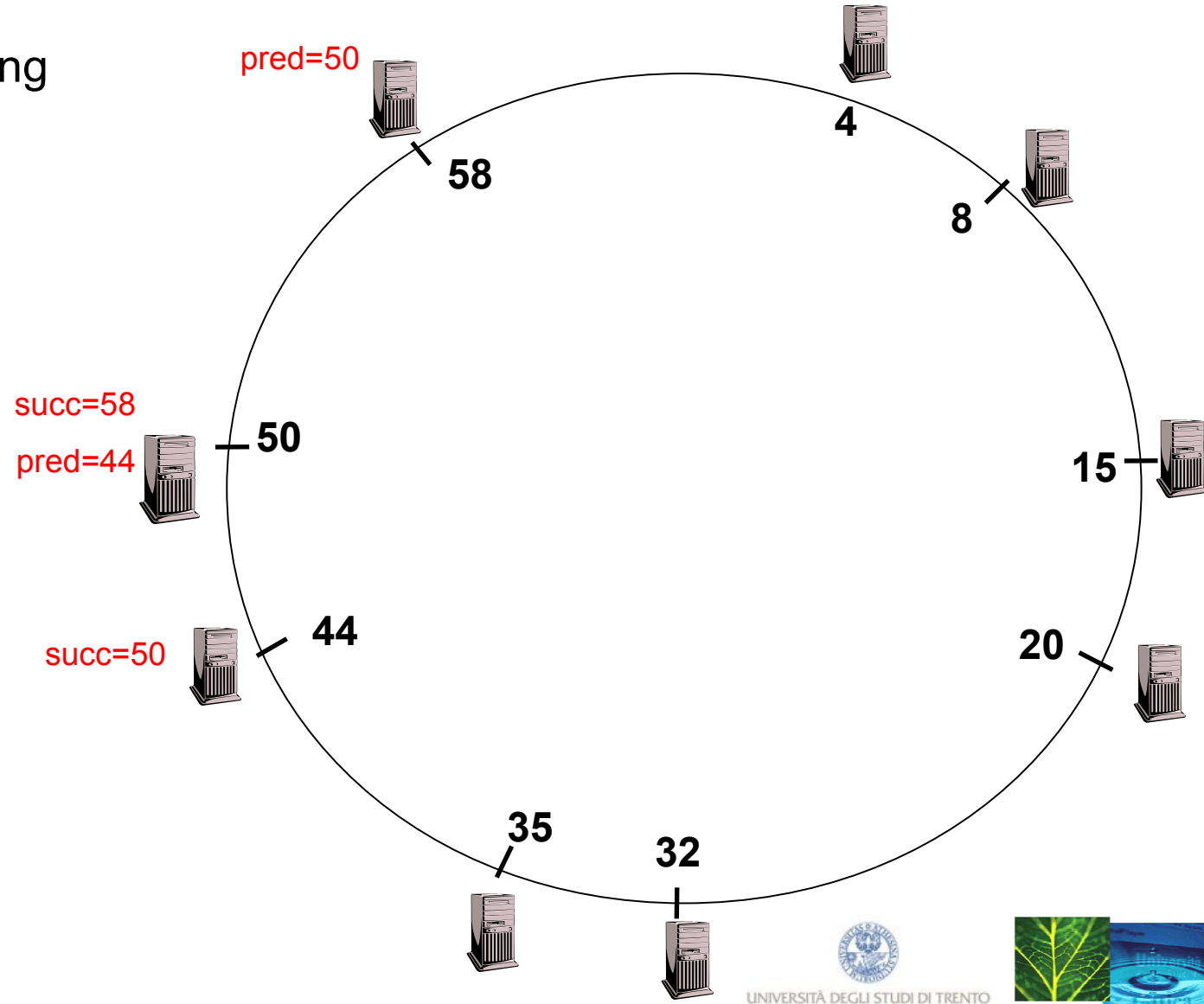
Joining Operation (cont'd)

- Node 44 sends a stabilize message to its new successor, node 50
- Node 50 sets its predecessor to node 44



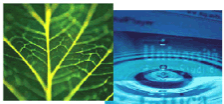
Joining Operation (cont'd)

This completes the joining operation!



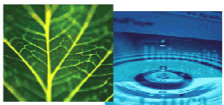
Achieving Robustness

- To improve robustness each node maintains the k (> 1) immediate successors instead of only one successor
- In the `notify()` message, node A can send its $k-1$ successors to its predecessor B
- Upon receiving `notify()` message, B can update its successor list by concatenating the successor list received from A with A itself



Chord Optimizations

- Reduce latency
 - Chose finger that reduces expected time to reach destination
 - Chose the closest node from range $[N+2^{i-1}, N+2^i)$ as successor
- Accommodate heterogeneous systems
 - Multiple virtual nodes per physical node

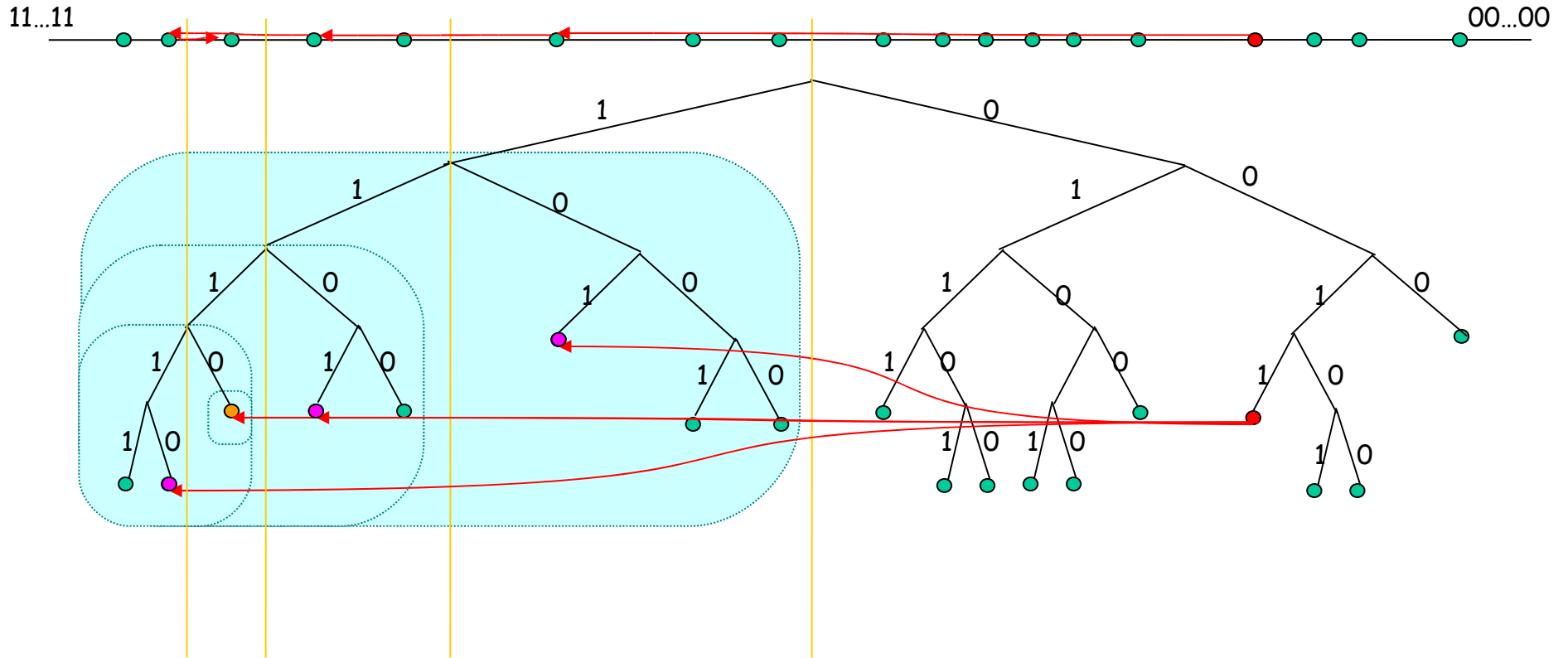


Kademlia Overview

➤ Applications

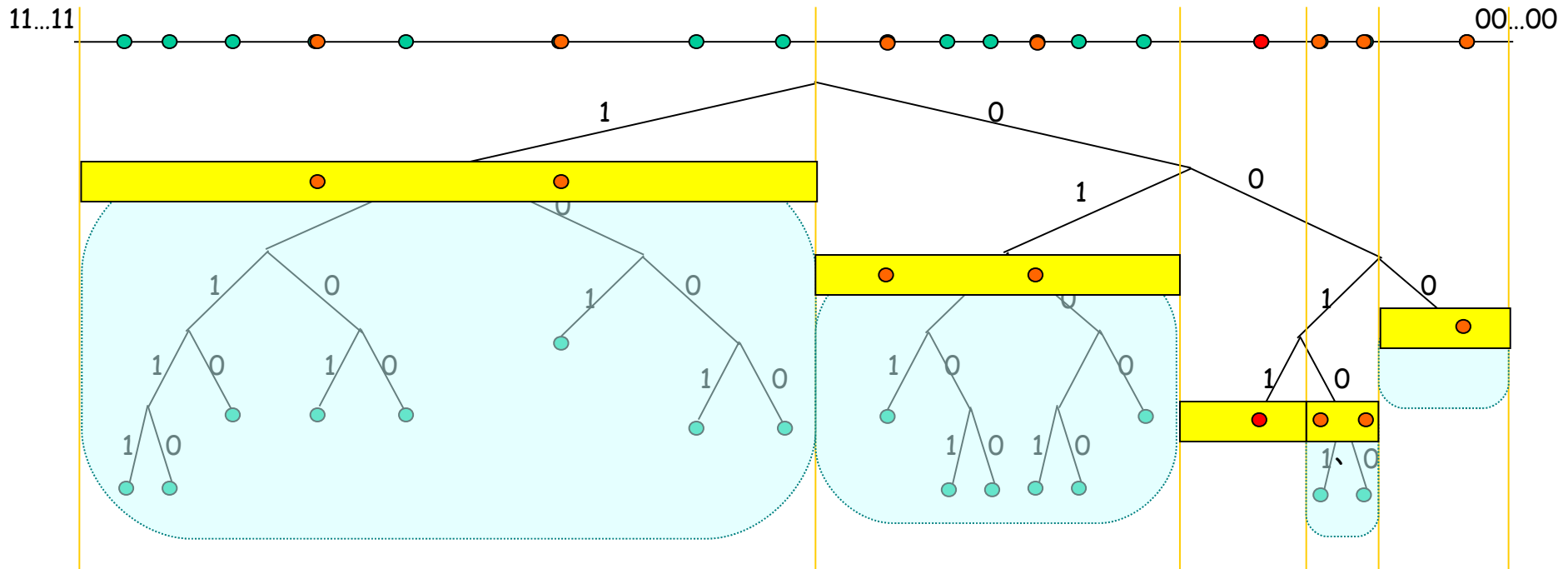
- Overnet network is based on Kademlia concepts
- Azureus (BitTorrent client)

Basic Idea



- Consider a query for ID 111010... initiated by node 0011100...

Routing Table



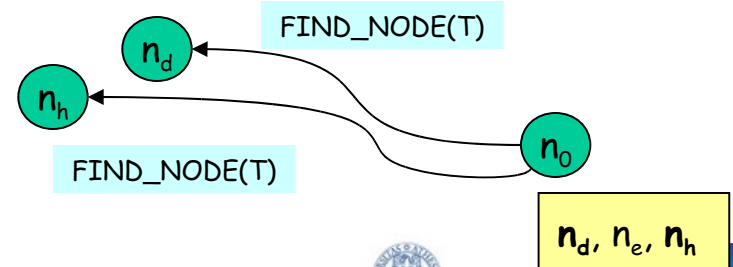
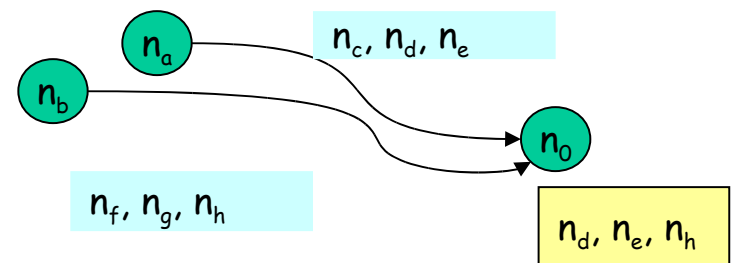
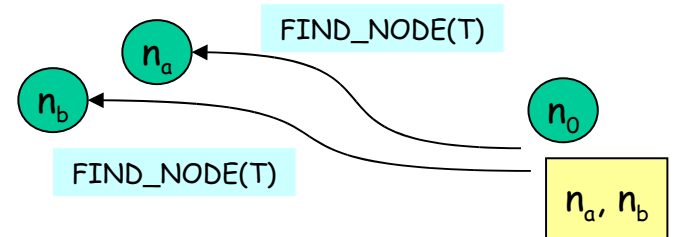
- Consider routing table for a node with prefix 0011
- Its binary tree is divided into a series of subtrees
- The routing table is composed of a series of **k-buckets** corresponding to each of these subtrees
 - Consider a 2-bucket example, each bucket will have at least 2 contacts for its key range
 - A contact consist of $\langle \text{IP:Port}, \text{NodeID} \rangle$

Query Routing Algorithm

- **Goal:** Find k nodes closest to ID T
- **Initial Phase:**
 - Select α nodes closest to T from n_o 's routing table
 - Send `FIND_NODE(T)` to each of the α nodes in parallel
- **Iteration:**
 - Select α nodes closest to T from the results of previous RPC
 - Send `FIND_NODE(T)` to each of the α nodes in parallel
 - Terminate when a round of `FIND_NODE(T)` fails to return any *closer* nodes
- **Final Phase:**
 - Send `FIND_NODE(T)` to all of k closest nodes not already queried
 - Return when have results from all the k -*closest* nodes.

$\alpha = 2$

$k = 3$

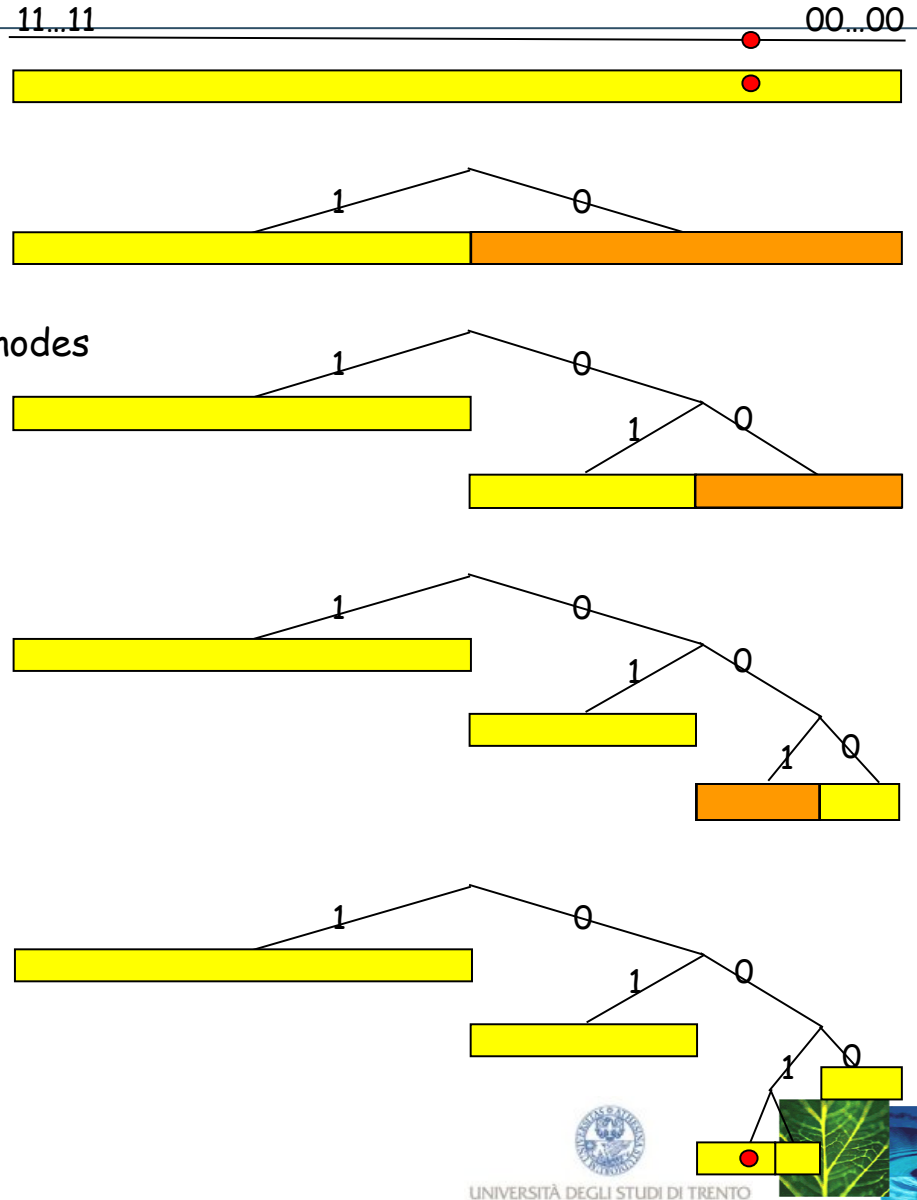
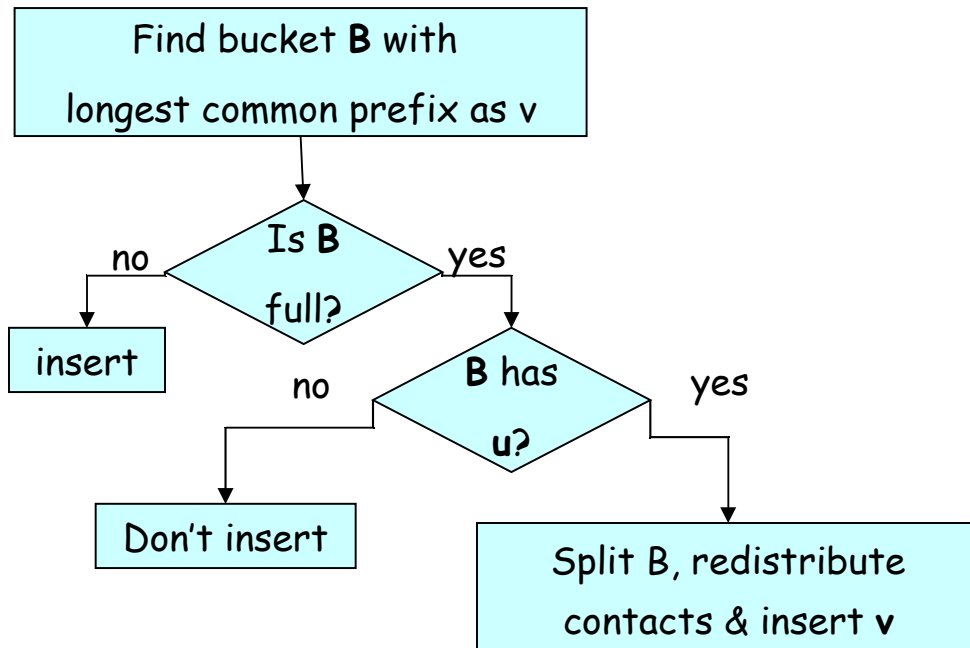


Node Joining & Routing Table Evolution

➤ Joining Node (u):

- ✓ Borrow an alive node's ID (w) off-line
- ✓ Initial routing table has a single k-bucket containing u and w.
- ✓ u performs FIND_NODE(u) to learn about other nodes

➤ Inserting new entry (v)

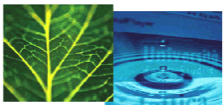


Summary

- Strengths
 - Low control message overhead
 - Tolerance to node failure and leave
 - Capable of selecting low-latency path for query routing
 - Provable performance bounds
- Weaknesses
 - Non-uniform distribution of nodes in ID-space results into imbalanced routing table and inefficient routing
 - Balancing of storage load is not truly solved
 - No experimental results provided

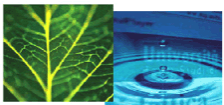
Conclusions

- Distributed Hash Tables are a key component of scalable and robust overlay networks
 - CAN: $O(d)$ state, $O(d*n^{1/d})$ distance
 - Chord: $O(\log n)$ state, $O(\log n)$ distance
 - Kademlia: $O(\log n)$ state, $O(\log n)$ distance
- All of them can achieve stretch < 2
- Simplicity is key
- Services built on top of distributed hash tables
 - persistent storage (OpenDHT, Oceanstore)
 - p2p file storage, i3 (Chord)
 - multicast (CAN, Tapestry)



Three aspects of a DHT design

- **Geometry:** smallest network graph that ensures correct routing/lookup in the DHT
 - Tree, Hypercube, Ring, Butterfly, Debruijn
- **Distance function:** captures a geometric structure
 - $d(id1, id2)$ for any two node identifiers
- **Algorithm:** rules for selecting neighbors and routes using the distance function

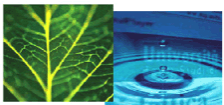


Analysis of *Overlay Path Latency*

- Goal: Minimize end-to-end overlay path latency
 - not just the number of hops
- Both FNS and FRS can reduce latency
 - Tree has FNS, Hypercube has FRS, Ring & XOR have both

Evaluation:

- Using Internet latency distributions (see paper)

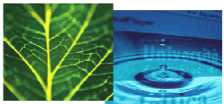


Which is more effective, FNS or FRS?

FNS+FRS Ring



Plain << FRS << FNS FNS+FRS
Neighbor Selection is much better than Route Selection



Conclusions

- Routing Geometry is a fundamental design choice
 - Geometry determines flexibility
 - Flexibility improves resilience and proximity
- Ring has the greatest flexibility

