CS 268:
Overlay Networks:
Distributed Hash Tables

Kevin Lai

May 1, 2001

From last time

- Overlay Benefits
  - Do not have to modify existing hardware and software
  - Do not have to deploy at every node
- Another Overlay Benefit
  - Free to ignore physical network topology
    - avoids complexity of worrying about physical topology
    - pitfall: usually results in poor latency

Motivation

- Many distributed applications need to map from an identifier to a host
  - overlay network
    - to route packets to hosts
    - e.g., application layer multicast, overlay QoS
  - persistent storage
    - to locate a data item
    - distributed file system, name system, distributed database, content distribution
    - e.g., Napster, Gnutella, FreeNet, DNS, Akamai
  - distributed computation
    - to exchange results
    - e.g., @Home

Existing Solutions

- Flat space routing
  - every node has a route to every other node
  - n^2 state and communication, constant distance
  - requires too much state and communication
  - e.g., Narada, RON cannot scale beyond ~100 nodes
- Hierarchical routing
  - every node routes through its parent
  - constant state and communication, log(n) distance
  - puts too much load on root
  - root is single point of failure
  - e.g., @Home is 1 level hierarchy, server is overloaded
  - e.g., Napster is 1 level hierarchy, vulnerable to legal action against root

Distributed Hash Tables

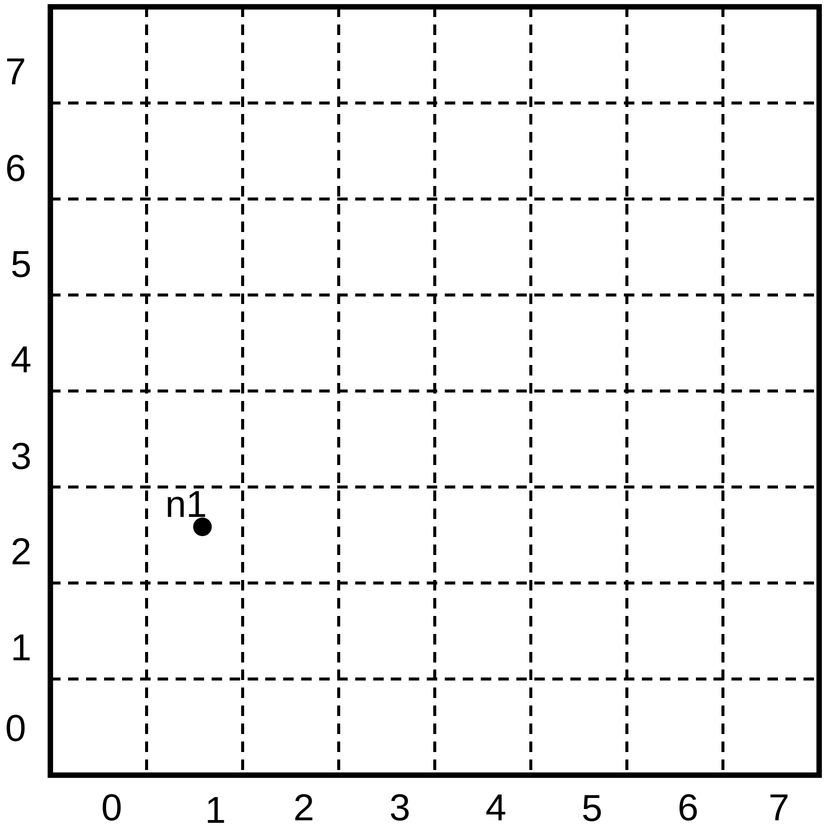
- Problem:
  - Given an id, map to a host
- Challenges
  - Scalability: hundreds of thousands or millions of machines
  - Instability
    - changes in routes, congestion, availability of machines
  - Heterogeneity
    - latency: 1ms to 1000ms
    - bandwidth: 32Kb/s to 100Mb/s
    - nodes stay in system from 10s to a year
  - Trust
    - selfish users
    - malicious users

Content Addressable Network (CAN)

- Associate to each node and item a unique *id* in an *d*-dimensional Cartesian space
- Goals
  - Scales to hundreds of thousands of nodes
  - Handles rapid arrival and failure of nodes
- 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

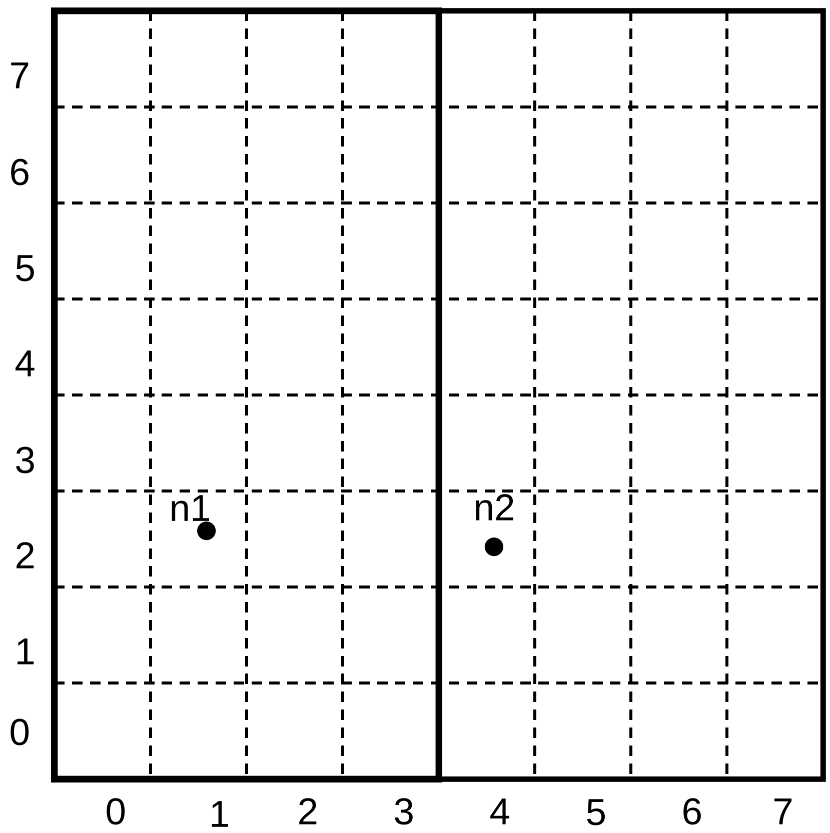
CAN Example: Two Dimensional Space

- 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



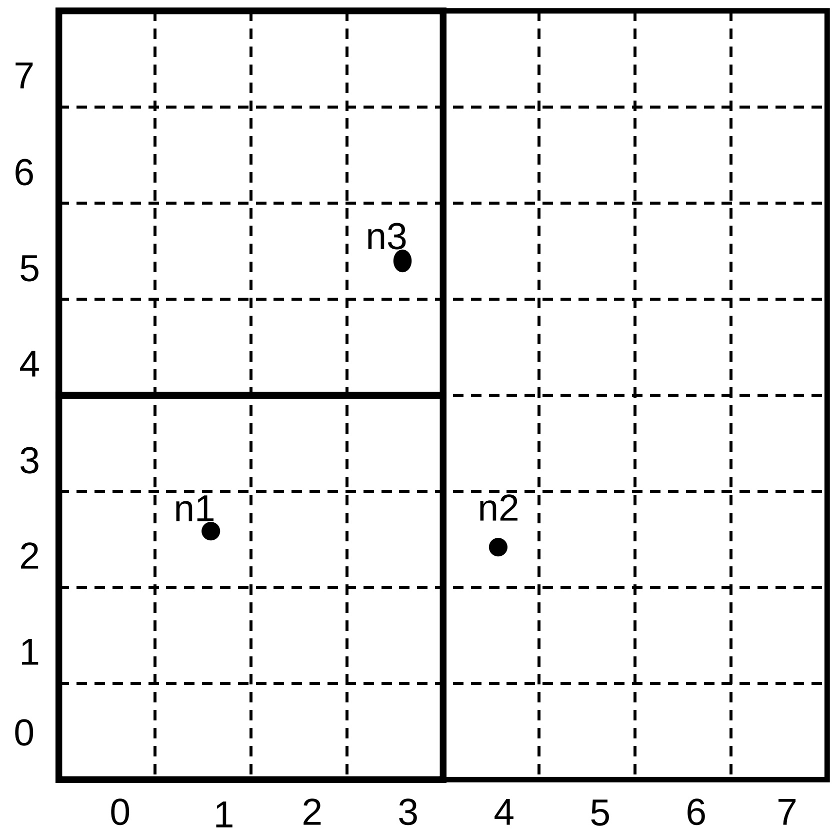
CAN Example: Two Dimensional Space

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



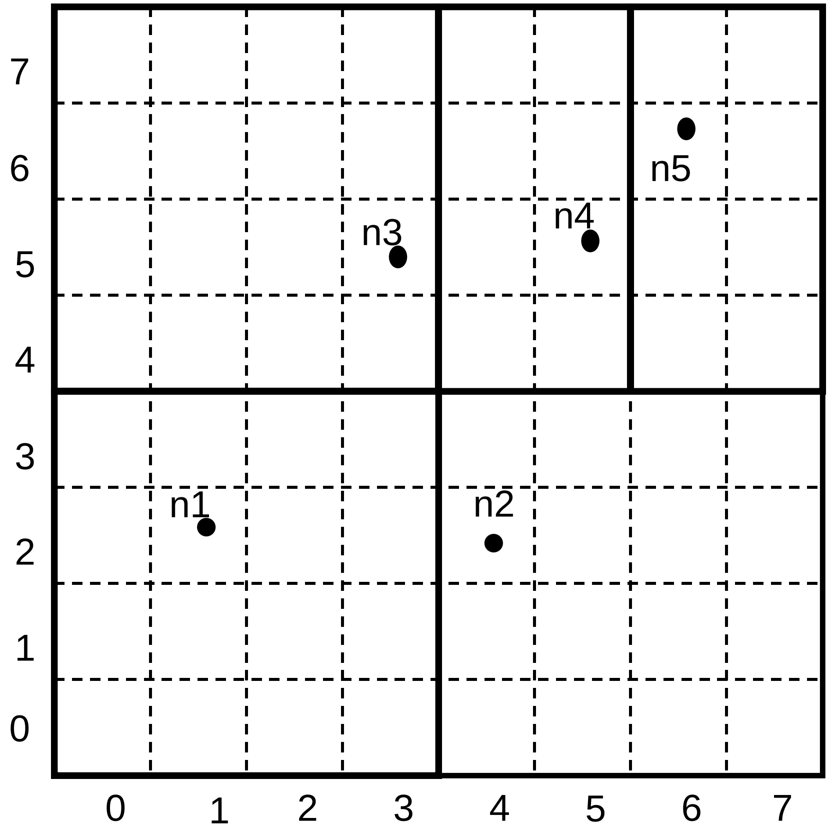
CAN Example: Two Dimensional Space

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



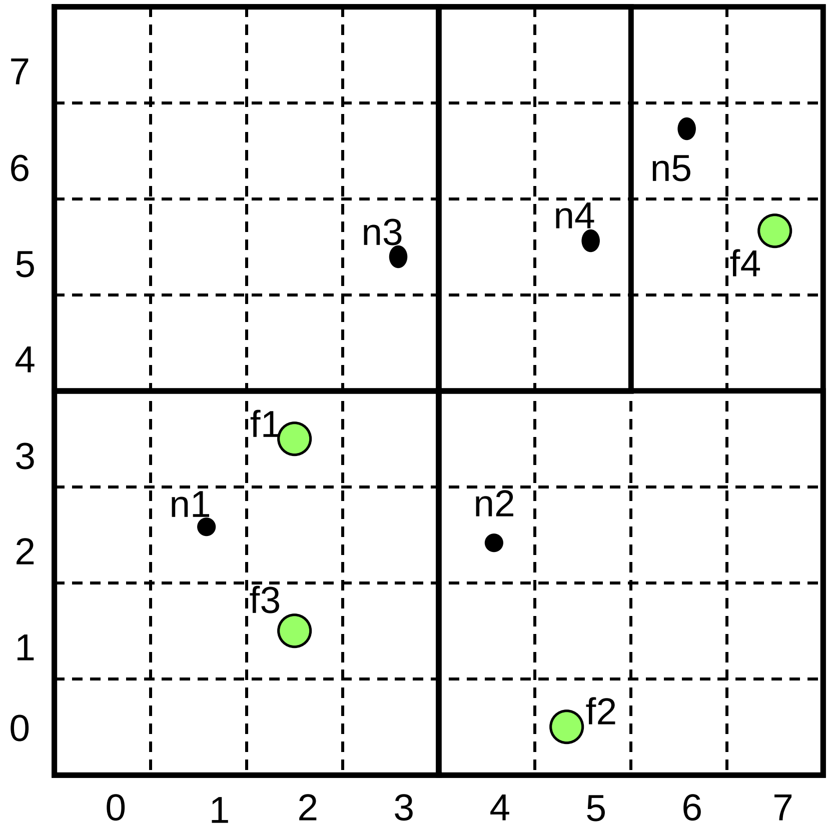
CAN Example: Two Dimensional Space

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



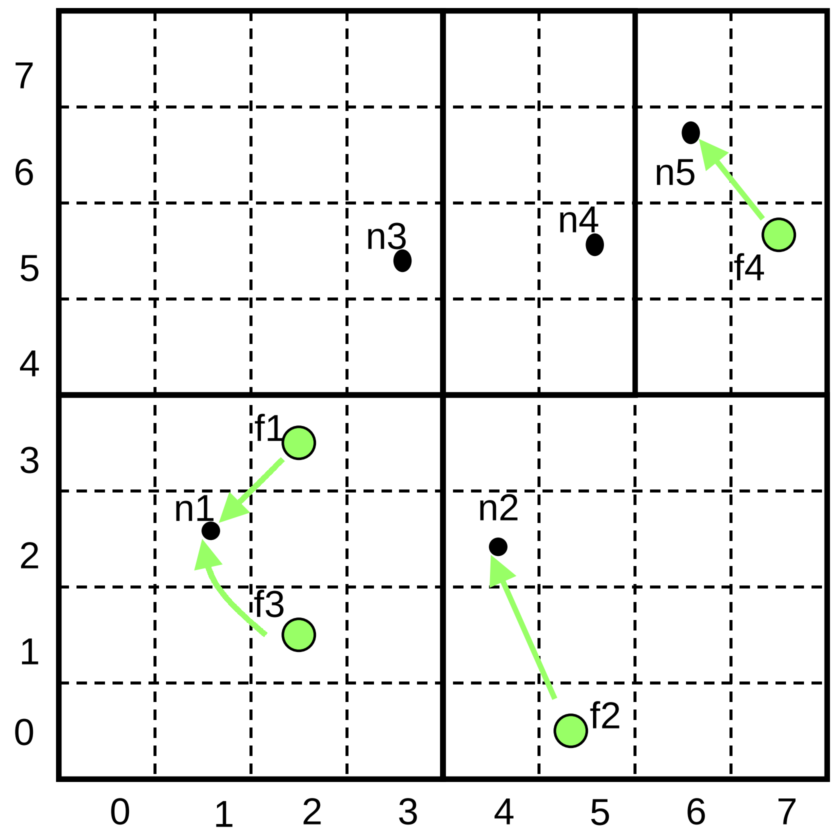
CAN Example: Two Dimensional Space

- 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);



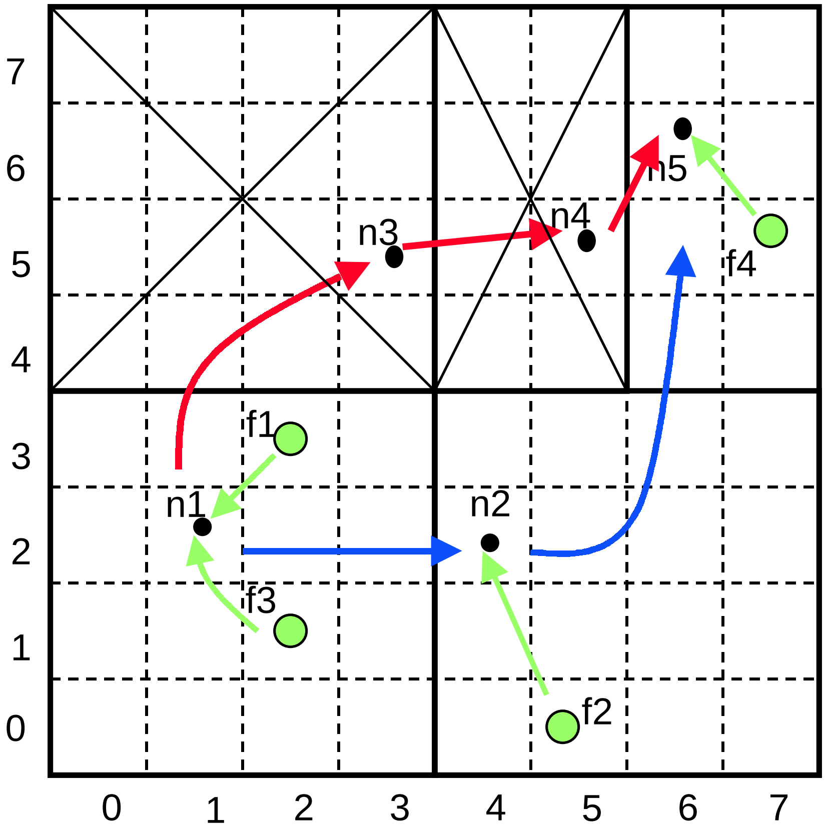
CAN Example: Two Dimensional Space

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



CAN: Query Example

- Each node knows its neighbors in the *d*-space
- Forward query to the neighbor that is closest to the query *id*
- Example: assume n1 queries f4
- Can route around some failures
  - some failures require local flooding



Node Failure Recovery

- 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

Chord

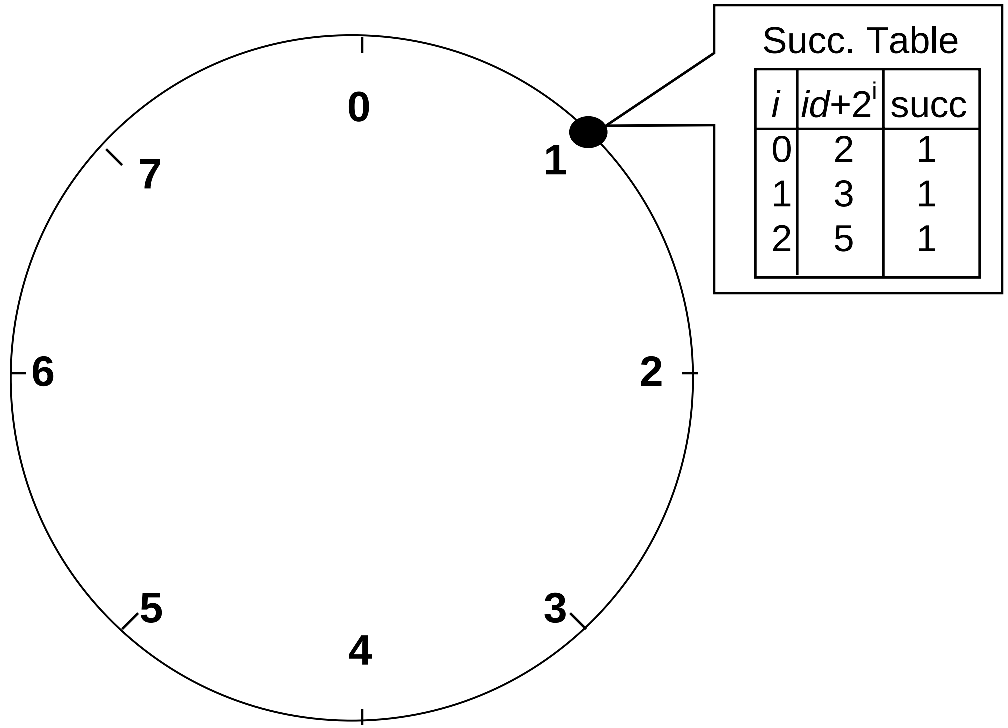
- Associate to each node and item a unique *id* in an *uni*-dimensional space
- Goals
  - Scales to hundreds of thousands of nodes
  - Handles rapid arrival and failure of nodes
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Data Structure

- Assume identifier space is $0..2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the succesor node of *id*

Chord Example

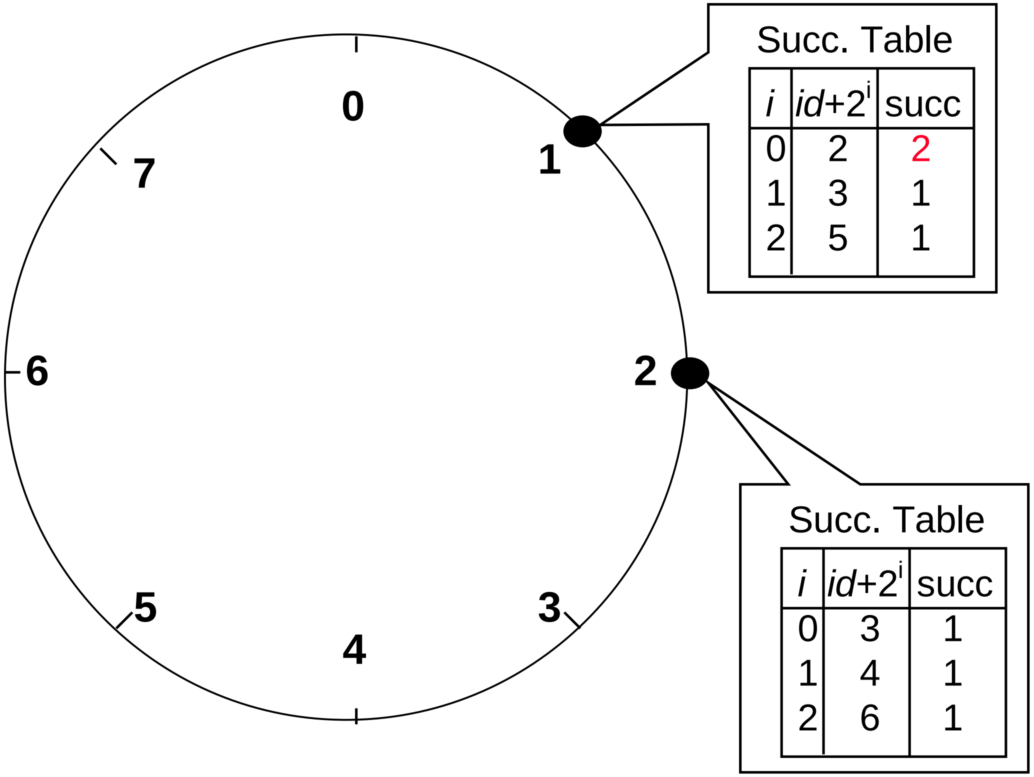
- Assume an identifier space 0..8
- Node n1:(1) joins→all entries in its finger table are initialized to itself



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

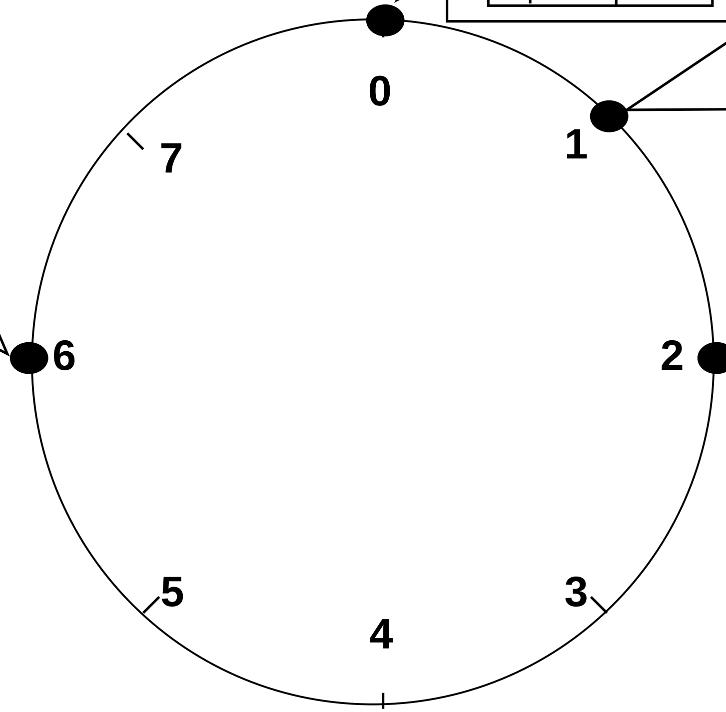
Chord Example

- Node n2:(3) joins



Chord Example

- Nodes n3:(0), n4:(6) 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)

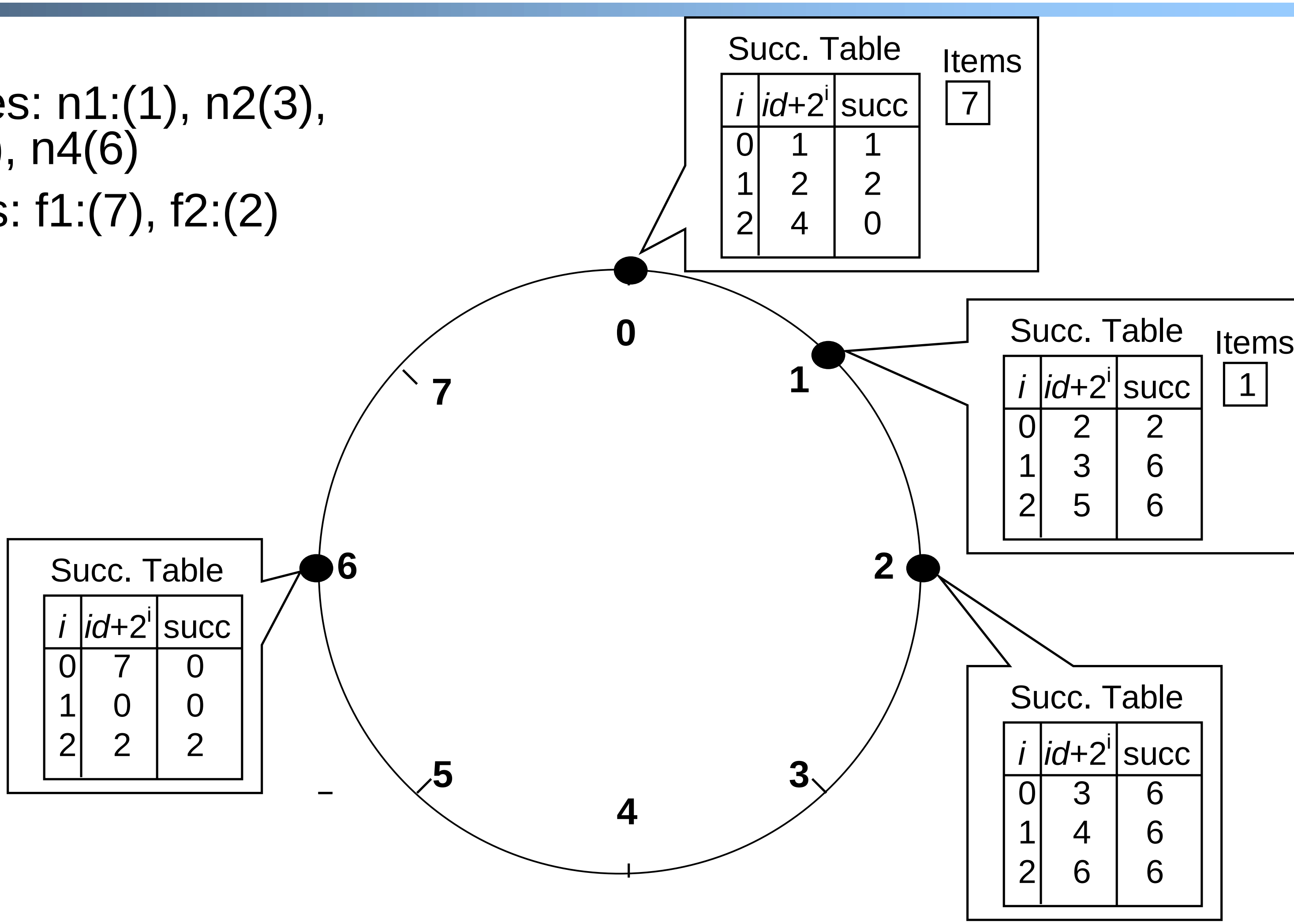
i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Succ. Table (node 6)

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

Chord Examples

- Nodes: n1:(1), n2(3), n3(0), n4(6)
- Items: f1:(7), f2:(2)



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)

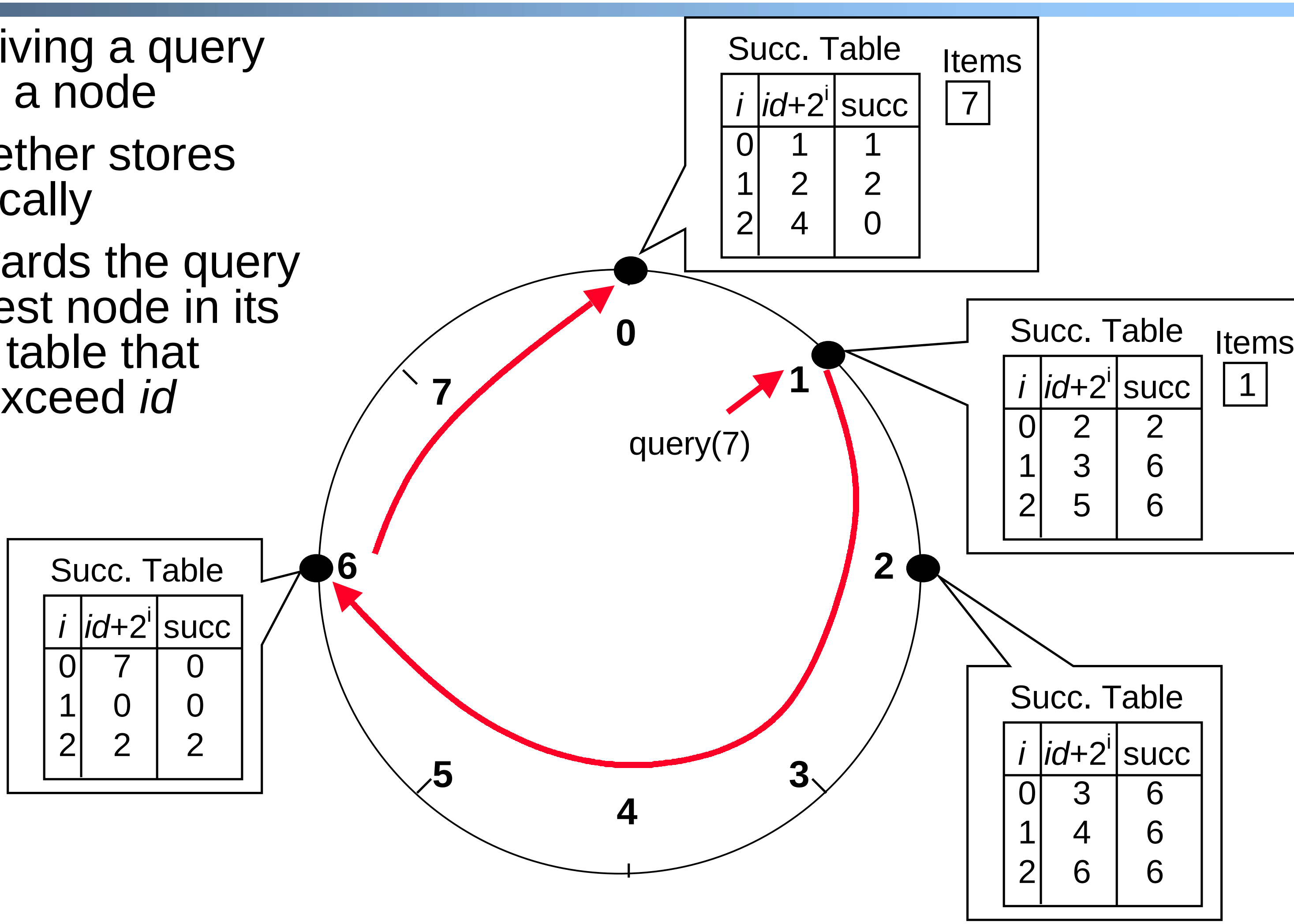
i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Succ. Table (node 6)

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

Query

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



CAN/Chord Optimizations

- Weight neighbor nodes by RTT
  - when routing, choose neighbor who is closer to destination with lowest RTT from me
  - reduces path latency
- Multiple physical nodes per virtual node
  - reduces path length (fewer virtual nodes)
  - reduces path latency (can choose physical node from virtual node with lowest RTT)
  - improved fault tolerance (only one node per zone needs to survive to allow routing through the zone)
- Several others

Discussion

- Queries
  - Iteratively or recursively
- Heterogeneity?
- Trust?

Conclusions

- Distributed Hash Tables are a key component of scalable and robust overlay networks
- CAN: O(d) state, O(d*n1/d) distance
- Chord: O(log n) state, O(log n) distance
- Both achieve stretch < 2
- Simplicity is key
- Services built on top of distributed hash tables
  - p2p file storage, i3 (chord)
  - multicast (CAN)
  - persistent storage (OceanStore using Tapestry)

I3 Motivation

- Today's Internet is built around a point-to-point communication abstraction:
  - Send packet "p" from host "A" to host "B"
- This abstraction allows Internet to be highly scalable and efficient, but…
- … not appropriate for applications that require:
  - Multicast
  - Anycast
  - Mobility
  - …

Why?

- Point-to-point communication abstraction implicitly assumes that there is one sender and one receiver, and that they are placed at fixed and well-known locations
  - E.g., a host identified by the IP address 128.32.xxx.xxx is most likely located in the Berkeley area

Existing Solutions

- Change IP to support new services, e.g.,
  - mobile IP
  - IP multicast
- Disadvantages:
  - Difficult to implement while maintaining Internet’s scalability
  - Even if they are implemented, ISPs might not have incentive to enable them

Existing Solutions (cont'd)

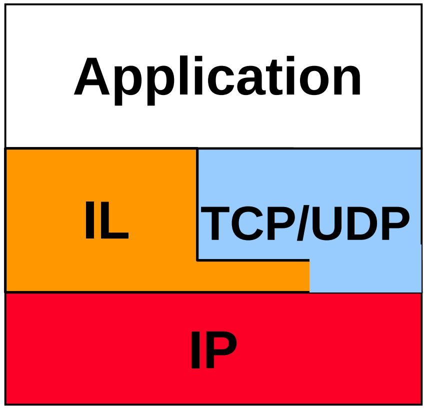
- Implement the required functionality at the application level, e.g.,
  - Application level multicast (Narada, Overcast, Scattercast,…)
  - Application level mobility via DNS
- Disadvantages:
  - Efficient routing is hard
  - Each application implements the same functionality over and over again
  - No synergy in deployment
    - might have n nodes deploy overlay A, and m nodes deploy overlay B instead of n+m nodes deploying i3 for both A and B
  - May have redundant overhead
    - e.g., probing for closest nodes

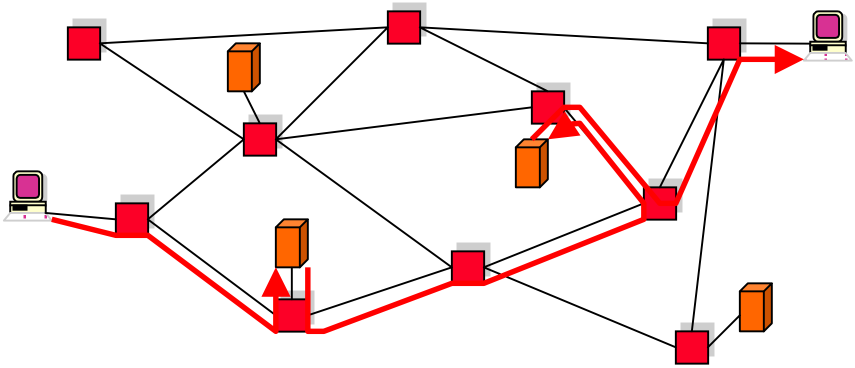
Key Observation

- All previous solutions use a simple but powerful technique: indirection
  - Assume a logical or physical indirection point interposed between sender(s) and receiver(s)
- Examples:
  - IP multicast assumes a logical indirection point: the IP multicast address
  - Mobile IP assumes a physical indirection point: the home agent

Solution

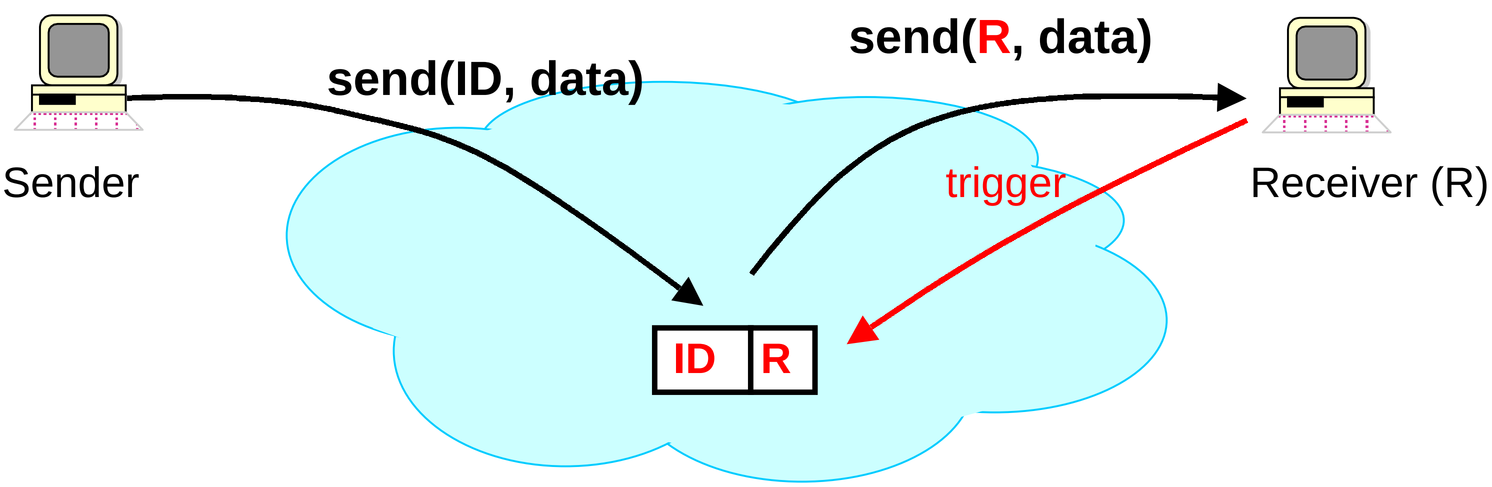
- Add an efficient indirection layer (IL) on top of IP
  - Transparent for legacy applications
- Use an overlay network to implement IL
  - Incrementally deployable; don't need to change IP





Internet Indirection Infrastructure (i3)

- Change communication abstraction: instead of point-to-point, exchange data by name
  - Each packet is associated an identifier ID
  - To receive a packet with identifier ID, receiver R maintains a trigger (ID, R) into the overlay network



Service Model

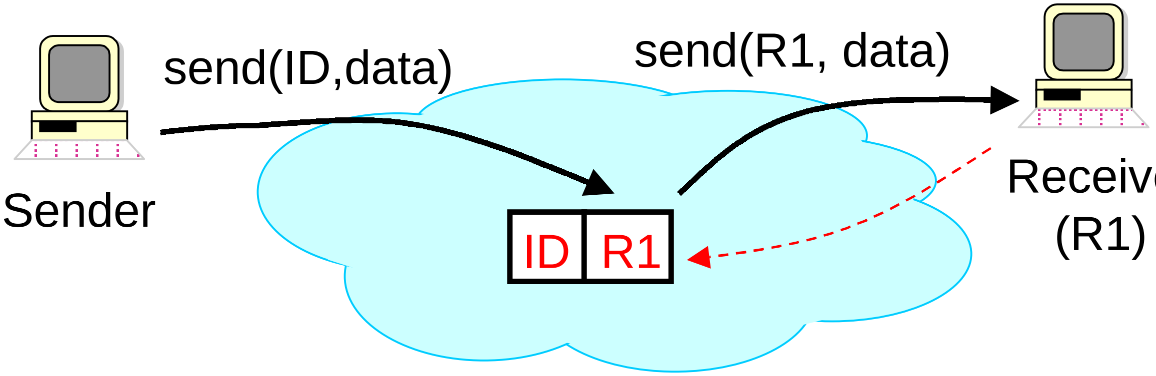
- Best-effort service model (like IP)
- Triggers are periodically refreshed by end-hosts
- Reliability, congestion control, and flow-control implemented at end-hosts

The Promise

- Provide support for
  - Mobility
  - Multicast
  - Anycast
  - Composable services

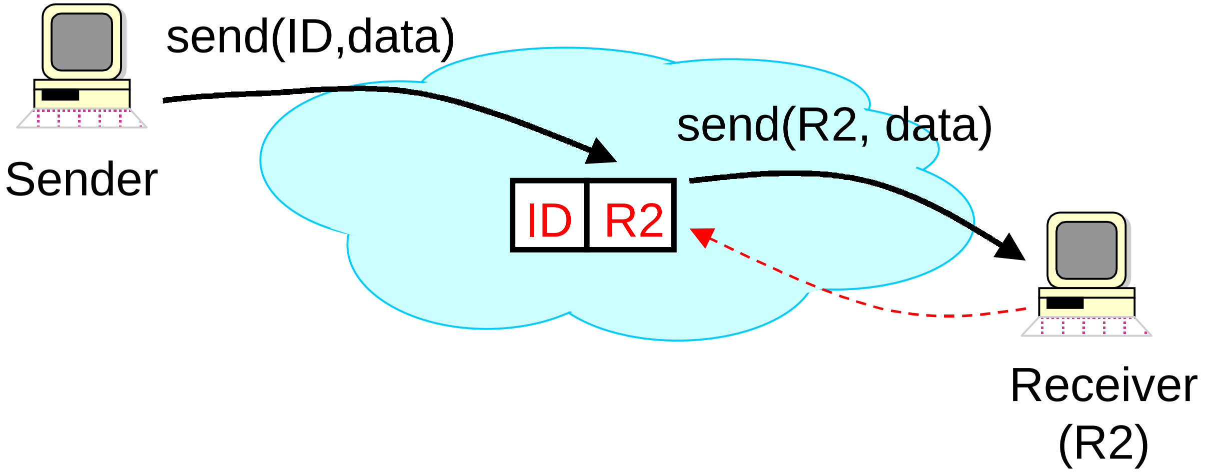
Mobility

- Host just needs to update its trigger as moves from one subnet to another
  - Robust
  - Support simultaneous mobility
  - Can eliminate the “triangle routing problem”
  - Location privacy



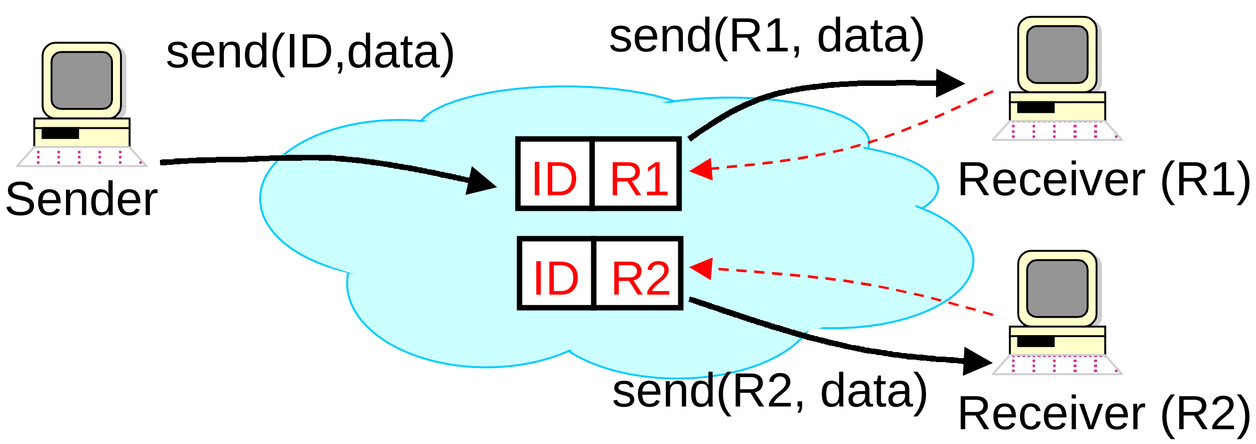
Mobility

- Host just needs to update its trigger as moves from one subnet to another
  - Robust
  - Support simultaneous mobility
  - Can eliminate the “triangle routing problem”
  - Location privacy



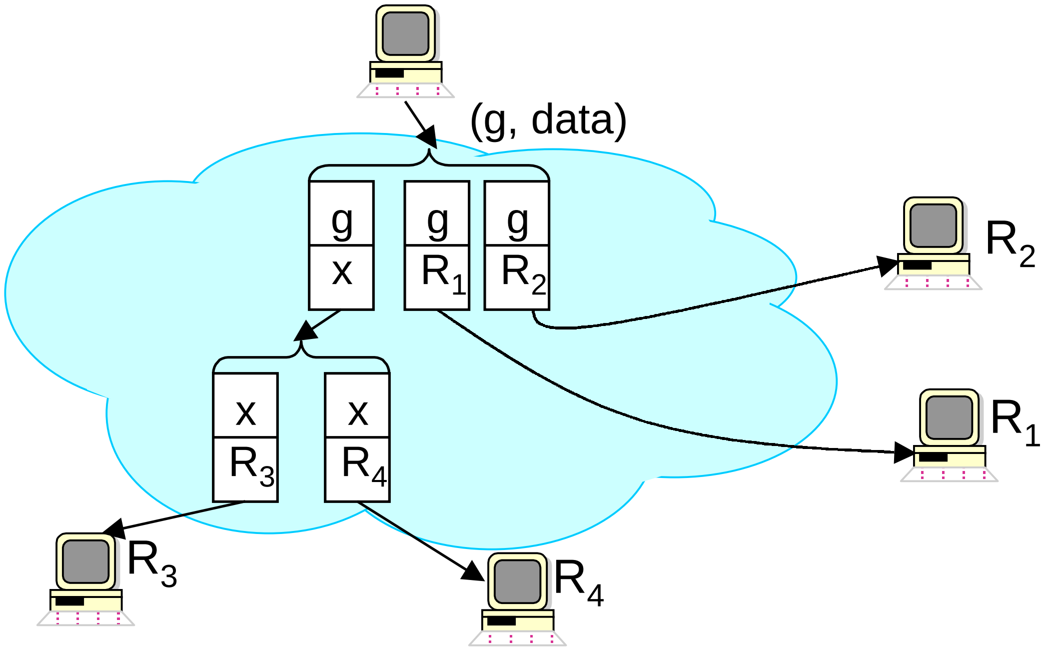
Multicast

- Unifies multicast and unicast abstraction
  - Multicast: receivers insert triggers with the same identifier
- An application can dynamically switch between multicast and unicast



Discussion: I3 vs. IP Multicast

- I3 doesn't provide direct support for scalable multicast
  - Simple to implement
- Give end-hosts the ability to control "routing"
  - End-hosts can build their own multicast tree if needed!



Implementation

- I3 is implemented on top of Chord
  - But can easily use CAN, Pastry, or Tapestry
- Each trigger (id, …) is stored on the server responsible for id
- Use Chord routing to find best matching trigger for a packet (id, data)
  - Path length O(log N), where N is the number of nodes in the system

Achieving Efficient Routing

- Source caches the I3 server that stores the matching trigger
  - Only first packet(s) of a flow experience O(log N) path length
  - Subsequent packets are forwarded via IP to the server S storing the matching trigger and then via IP to the destination
- Problem: if server S is far away → triangle routing problem

Avoid Triangle Routing Problem

- Each end-host picks the private triggers close to itself
- How: sampling the identifier space
- Sampling can be done off-line
  - I3 is an infrastructure → expect that mapping to be quite stable

Conclusions

- Indirection – key technique to implement basic communication abstractions and services
  - e.g., Multicast, Mobility
- Possible to build efficient indirection Layer on top of IP
- Shows the power of a simple, efficient primitive