Lecture 14:
Overlays

CS/ECE 438: Communication Networks
Prof. Matthew Caesar
May 1, 2010

Administrivia

- Upcoming deadlines
  - Presentation date signup due this Thursday Sept 17th
  - Track 1: MP1 due Sept 29th

Overlay networks

- Overlay networks
  - Improved flexibility and

- Distributed Hash Tables
  - Improved scalability, allow insertion of objects

- P2P, Bittorrent
  - Incentives for participation, lookup of local files

- Content distribution networks
  - Managed (provider-owned)

Overlay Networks and DHTs

Overlay networks: Motivations

- Protocol changes in the network happen very slowly

- Why?
  - Internet is shared infrastructure; need to achieve consensus
  - Many proposals require to change a large number of routers (e.g. IP Multicast, QoS); otherwise end-users won't benefit

- Proposed changes that haven't happened yet on large scale:
  - More addresses (IPv6, 1991)
  - Security (IPSEC, 1993); Multicast (IP multicast, 1990)

Overlay networks: Motivations

- Also, “one size does not fit all”

- Applications need different levels of
  - Reliability
  - Performance (latency
  - Security
  - Access control (e.g., who is allowed to join a multicast group)

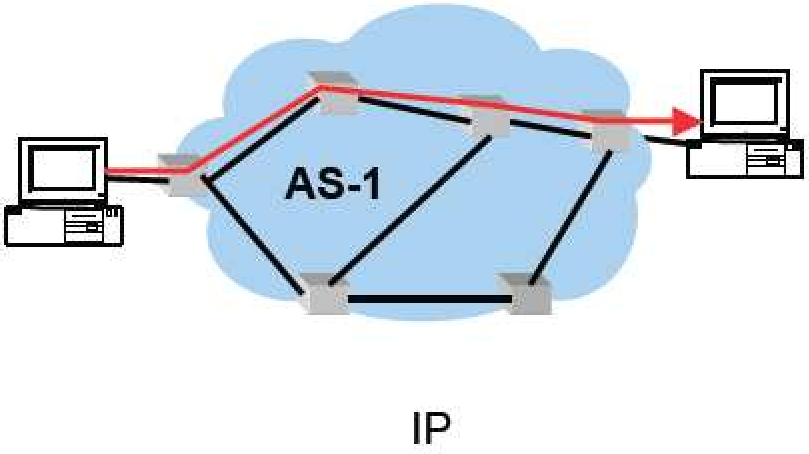
Overlay networks: Goals

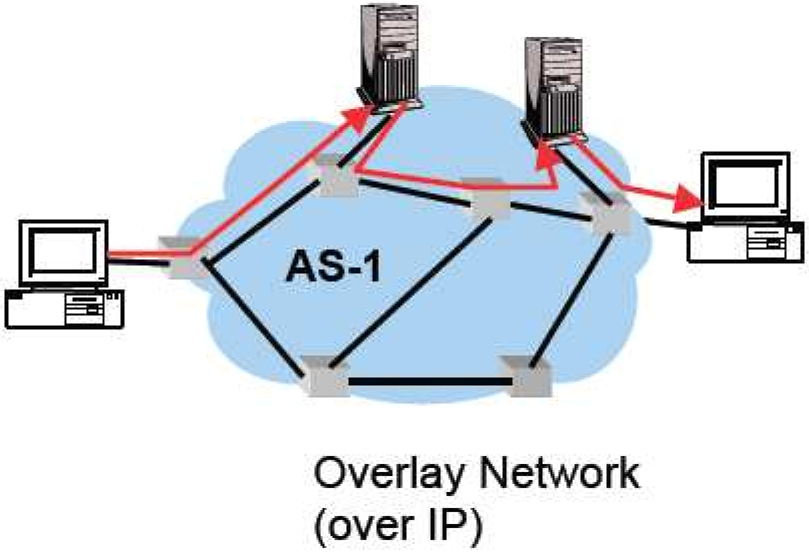
- Make it easy to deploy new functionalities in the network → Accelerate the pace of innovation

- Allow users to customize their service

Solution

- Build a computer network on top of another network
  - Individual hosts autonomously form a “virtual” network on top of IP
  - Virtual links correspond to inter-host connections (e.g., TCP sessions)

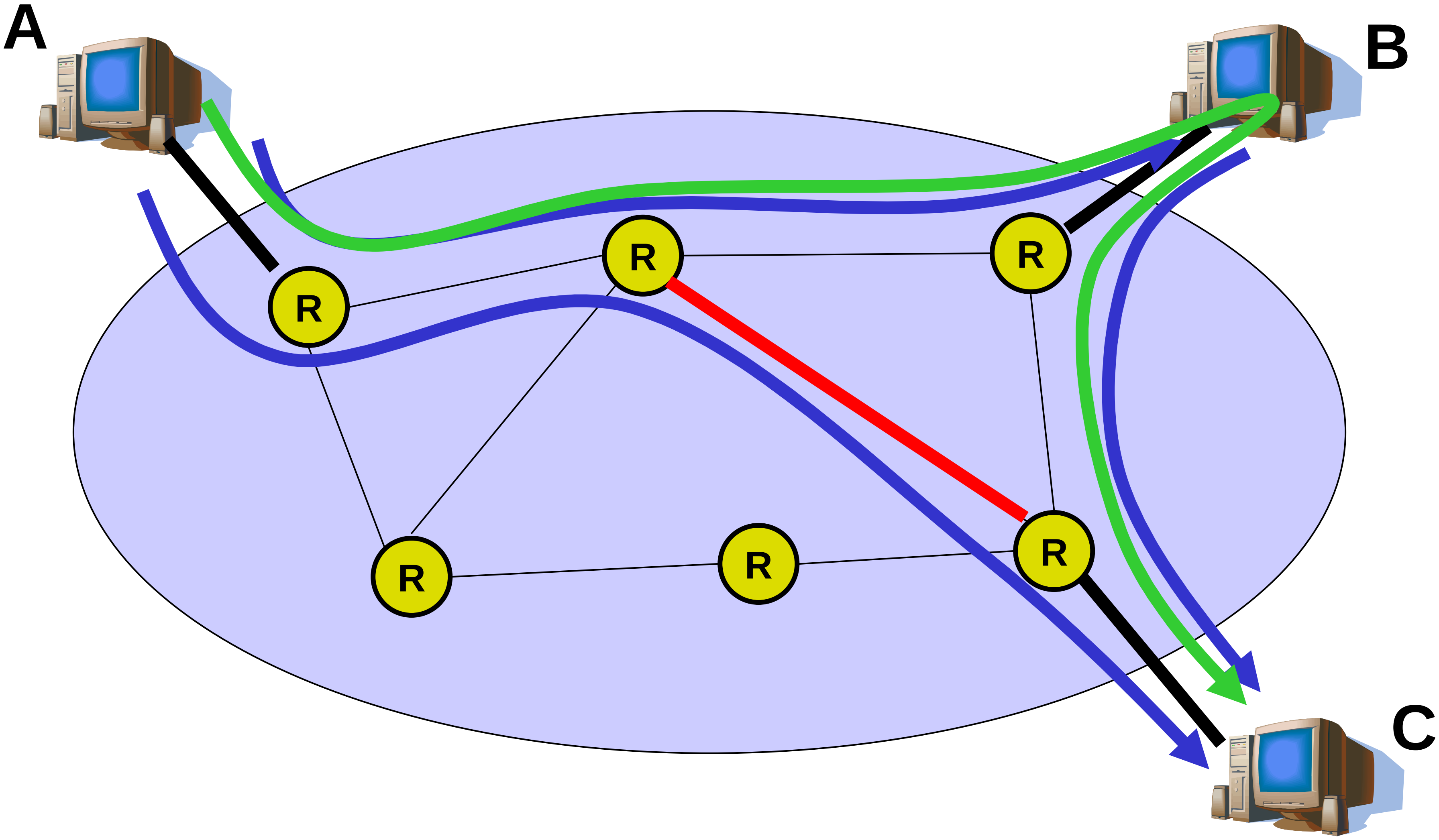




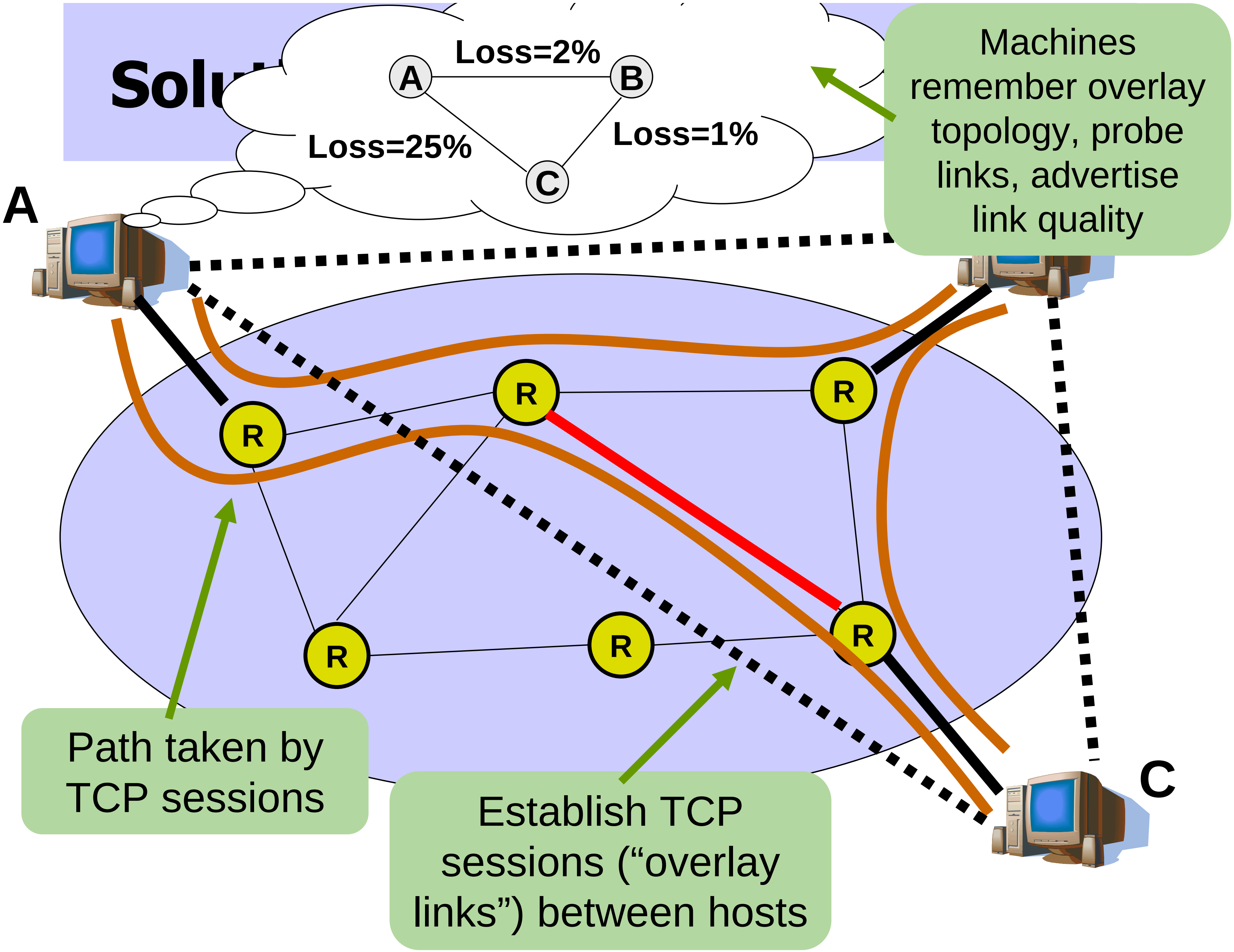
Example:
Resilient Overlay Networks

- Premise: by building an application-layer overlay network, can increase performance and reliability of routing
- Install N computers at different Internet locations
- Each computer acts like an overlay network router
  - Between each overlay router is an IP tunnel (logical link)
  - Logical overlay topology is all-to-all (N^2 total links)
- Run a link-state routing algorithm over the overlay topology
  - Computers measure each logical link in real time for packet loss rate, throughput, latency → these define link costs
  - Route overlay traffic based on measured characteristics

Motivating example: a congested network



Solu



Machines remember overlay topology, probe links, advertise link quality

Path taken by TCP sessions

Establish TCP sessions (“overlay links”) between hosts

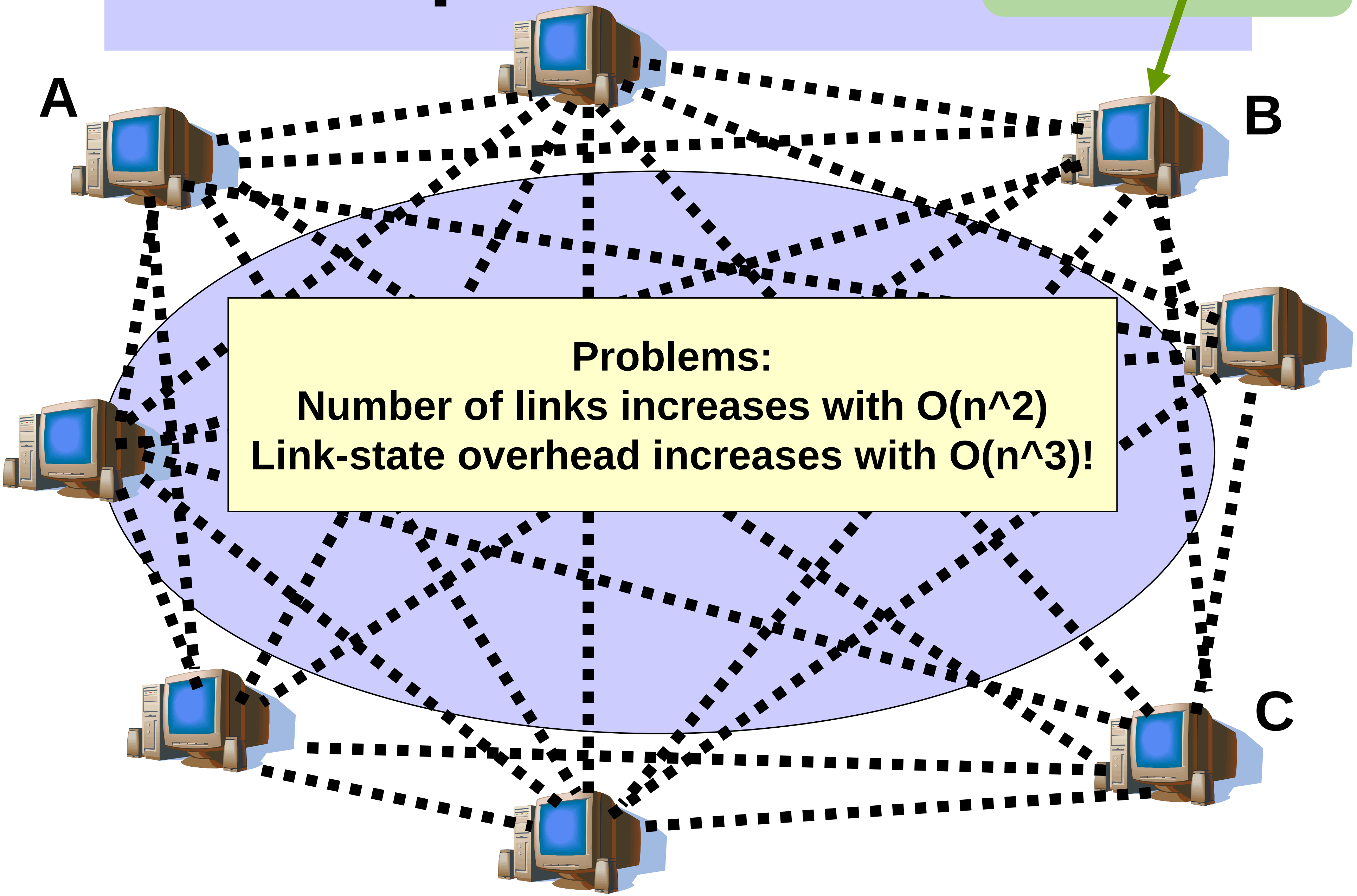
Benefits of overlay networks

- Performance:
  - Difficult to provide QoS at network-layer due to deployment hurdles, lack of incentives, application-specific requirements
  - Overlays can probe faster, propagate more routes

- Flexibility:
  - Difficult to deploy new functions at IP layer
  - Can perform multicast, anycast, QoS, security, etc

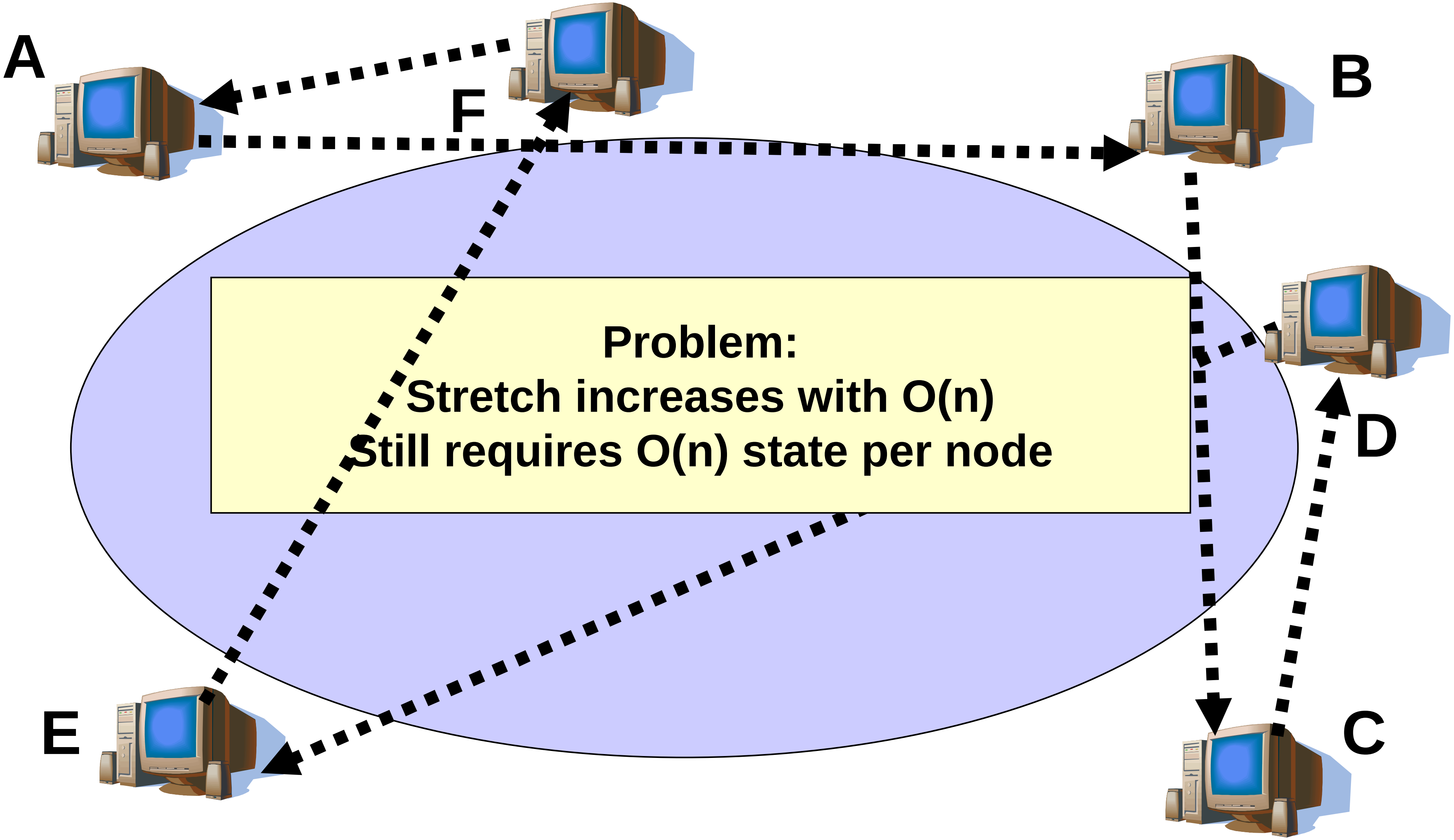
New problem: scalabi

Outdegree=~~2~~ 8



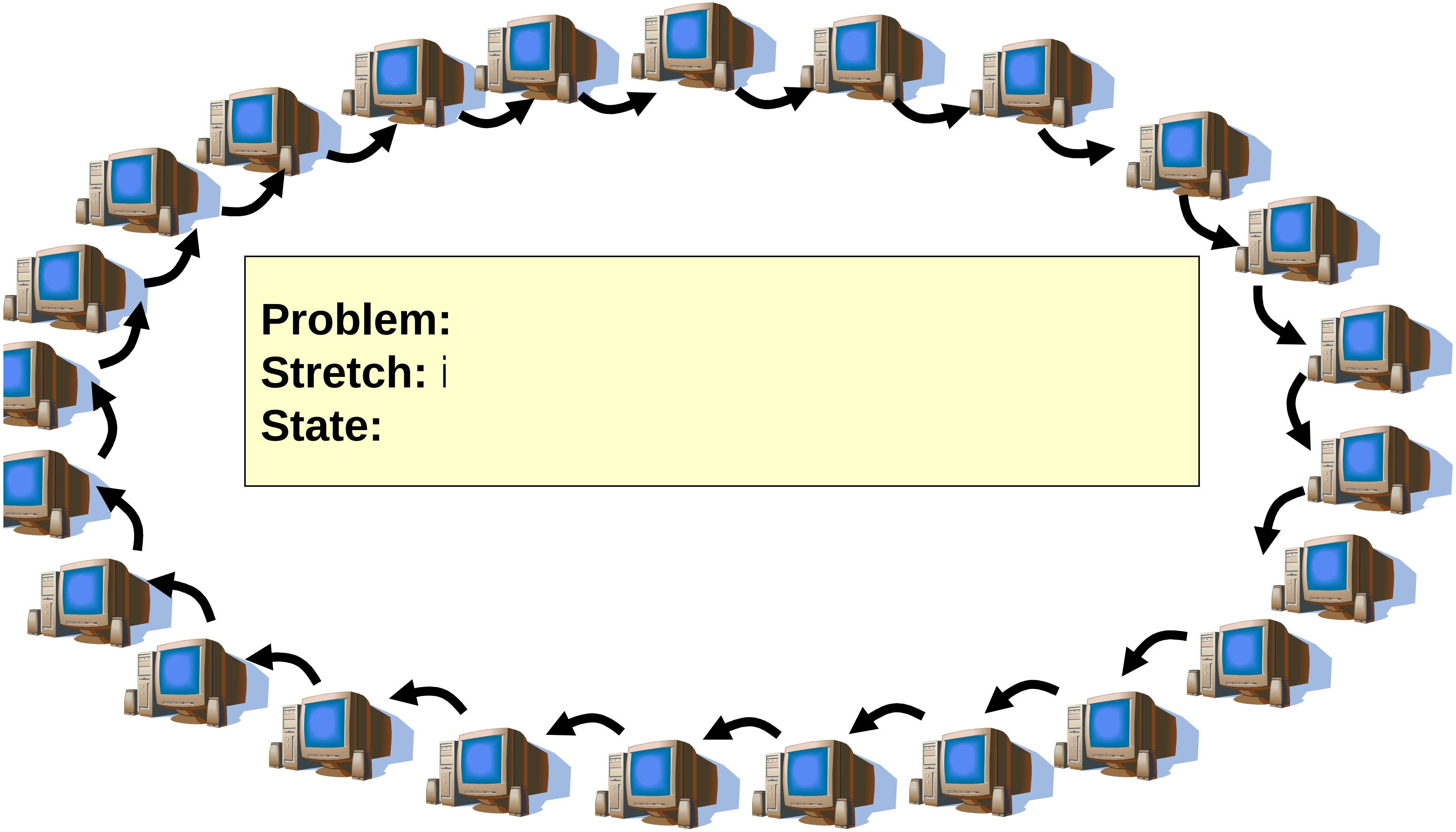
Problems:
Number of links increases with O(n^2)
Link-state overhead increases with O(n^3)!

Alternative:
replace full-mesh with logical ring



Problem:
Stretch increases with O(n)
Still requires O(n) state per node

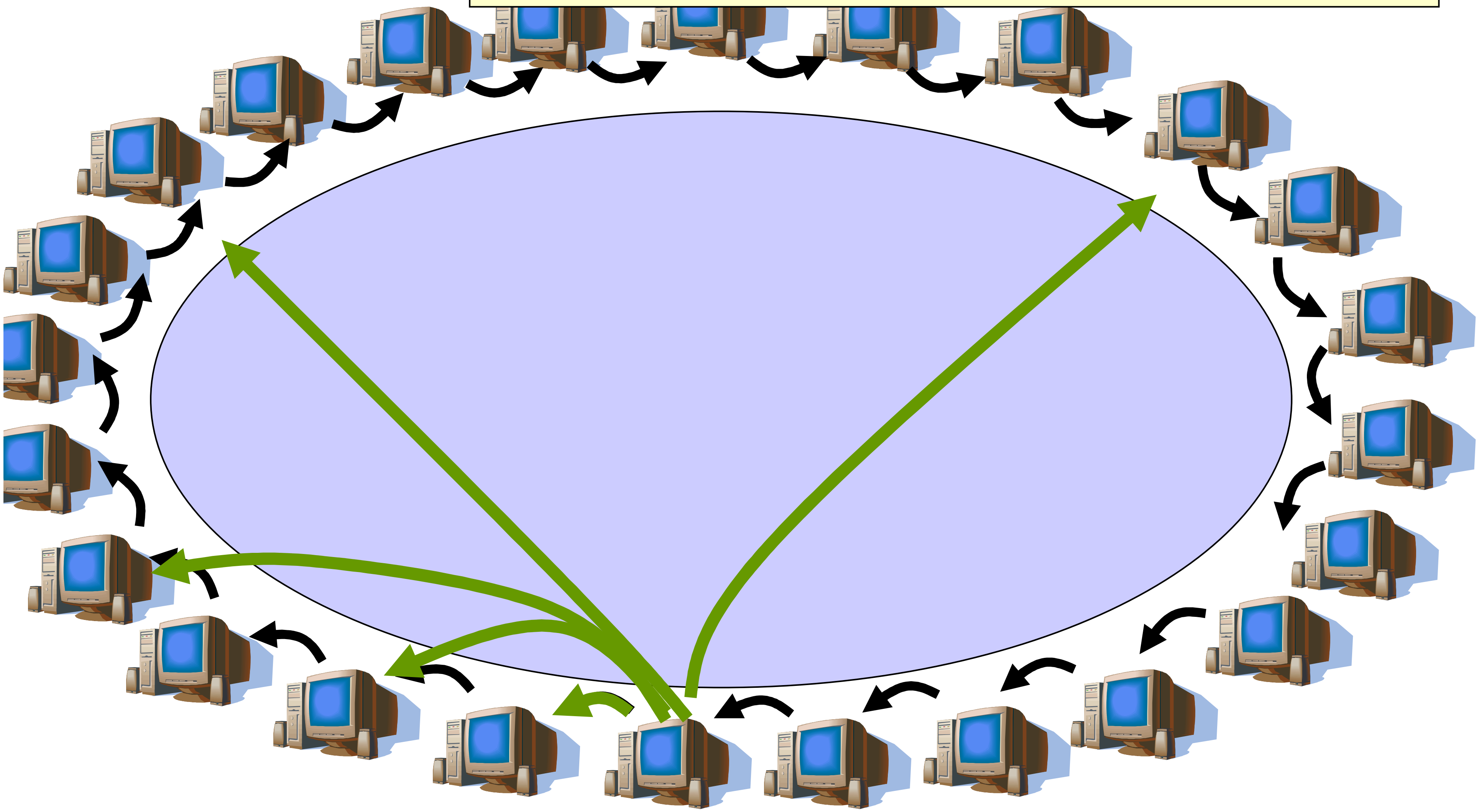
Alternative:
replace full-mesh with ring



Problem:
Stretch: i
State:

keep son

Improvement:
Stretch:
State:



Scaling overlay networks with Distributed Hash Tables (DHTs)

- Assign each host a numeric identifier
  - Randomly chosen, hash of node name, public key, etc
- Keep pointers (fingers) to other nodes
  - Goal: maintain pointers so that you can reach any destination in few overlay hops
  - Choosing pointers smartly can give low delay, while retaining low state
- Can also store objects
  - Insert objects by "consistently" hashing onto id space
- Forward by making progress in id space

Different kinds of DHTs

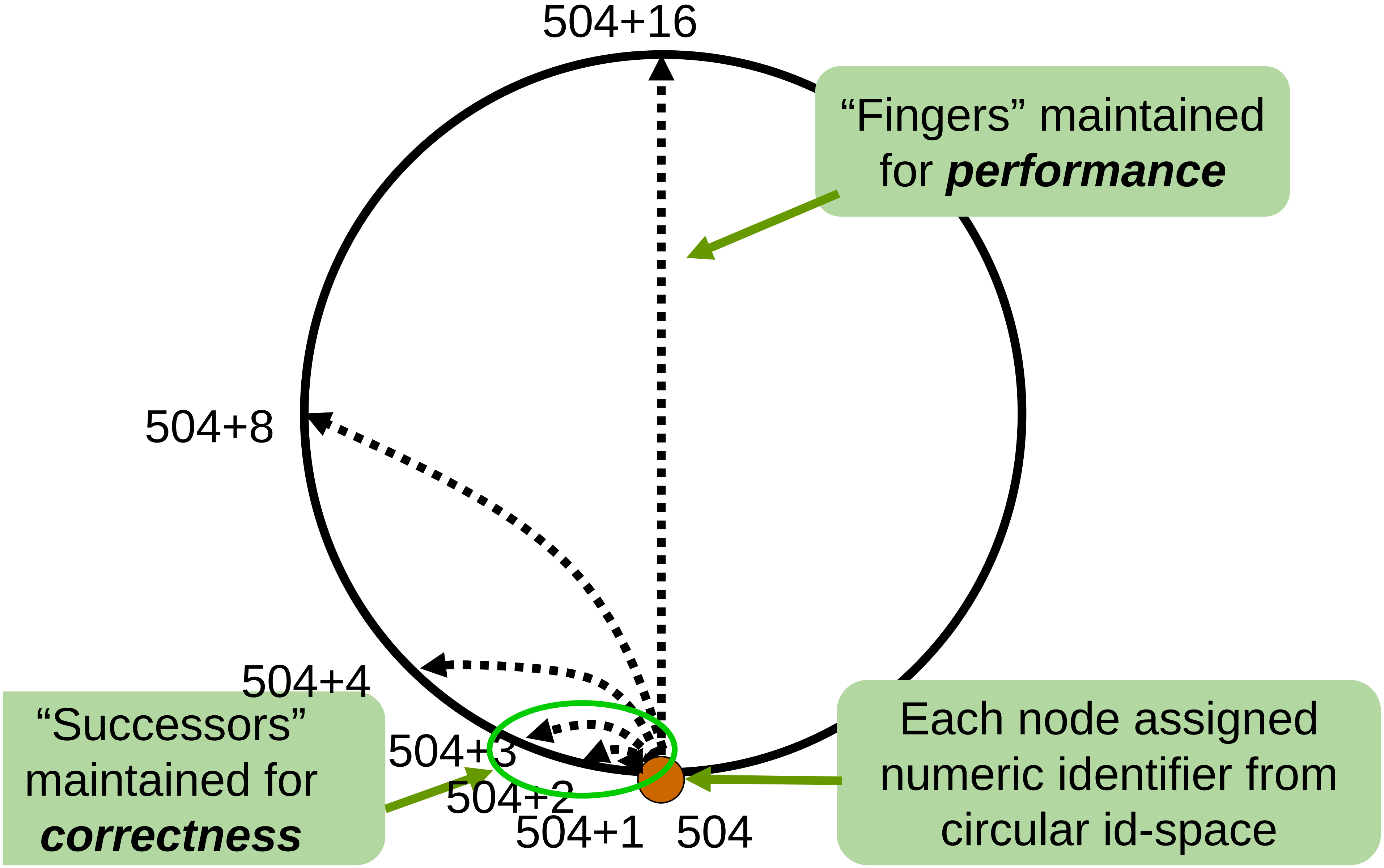
- Different topologies give different bounds on stretch (delay penalty)/state, different stability under churn, etc. Examples:

- Chord
  - Pointers to immediate successor on ring, nodes spaced 2^k around ring
  - Forward to numerically closest node without overshooting

- Pastry
  - Pointers to nodes sharing varying prefix lengths with local node, plus pointer to immediate successor
  - Forward to numerically closest node

- Others: Tapestry (like Pastry, but no successor pointers), CAN (like Chord, but torus namespace instead of ring)

The Chord DHT

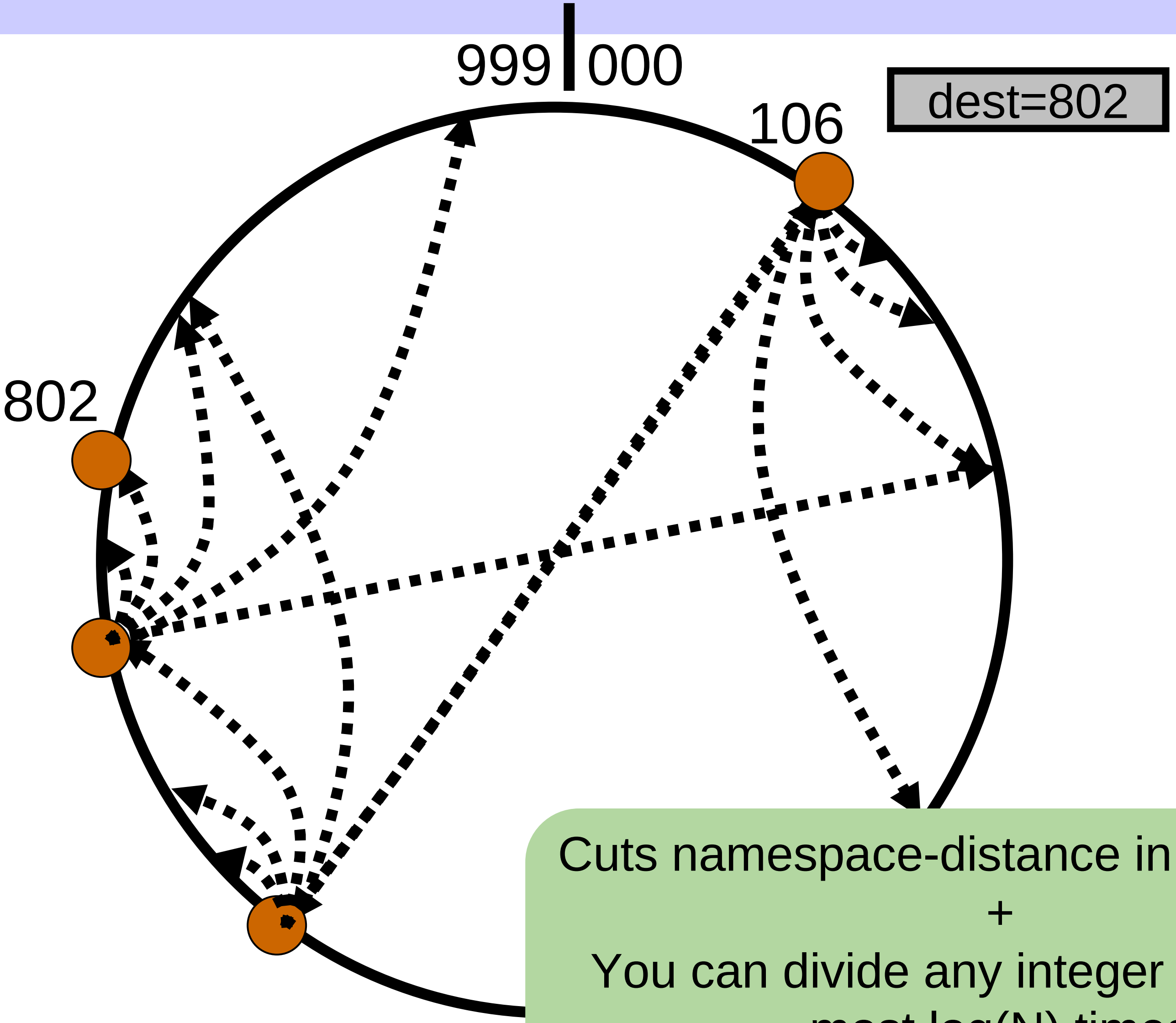


"Fingers" maintained for ***performance***

"Successors" maintained for ***correctness***

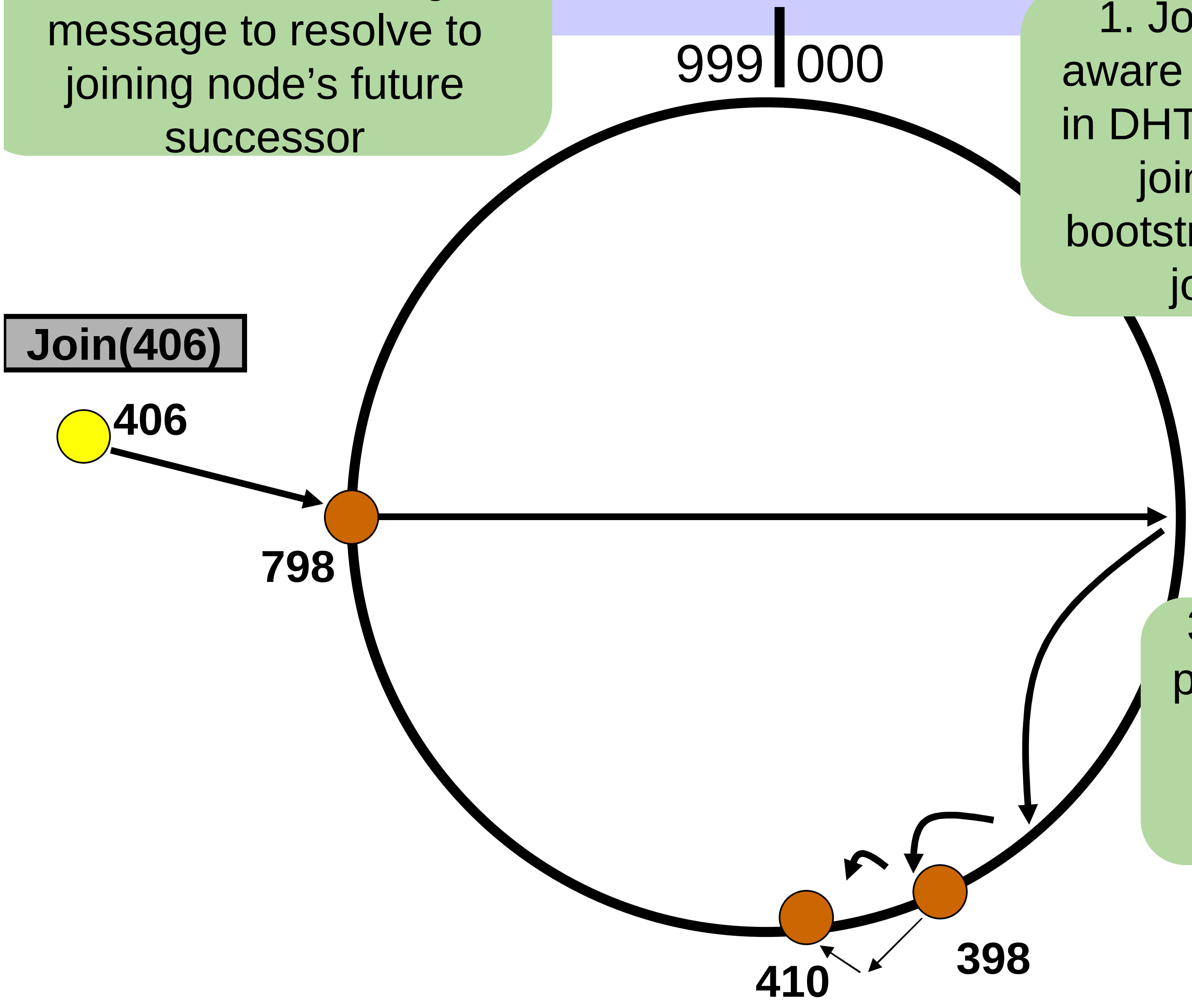
Each node assigned numeric identifier from circular id-space

Chord Example: Forwarding a lookup



Cuts namespace-distance in half per hop
+
You can divide any integer N in half at most log(N) times
= logarithmic stretch

mple: Joining a new node



1. Joining node must be aware of a “bootstrap” node in DHT. Joining node sends join request through bootstrap node towards the joining node’s ID

2. Bootstrap forwards message towards joining node’s ID, causing message to resolve to joining node’s future successor

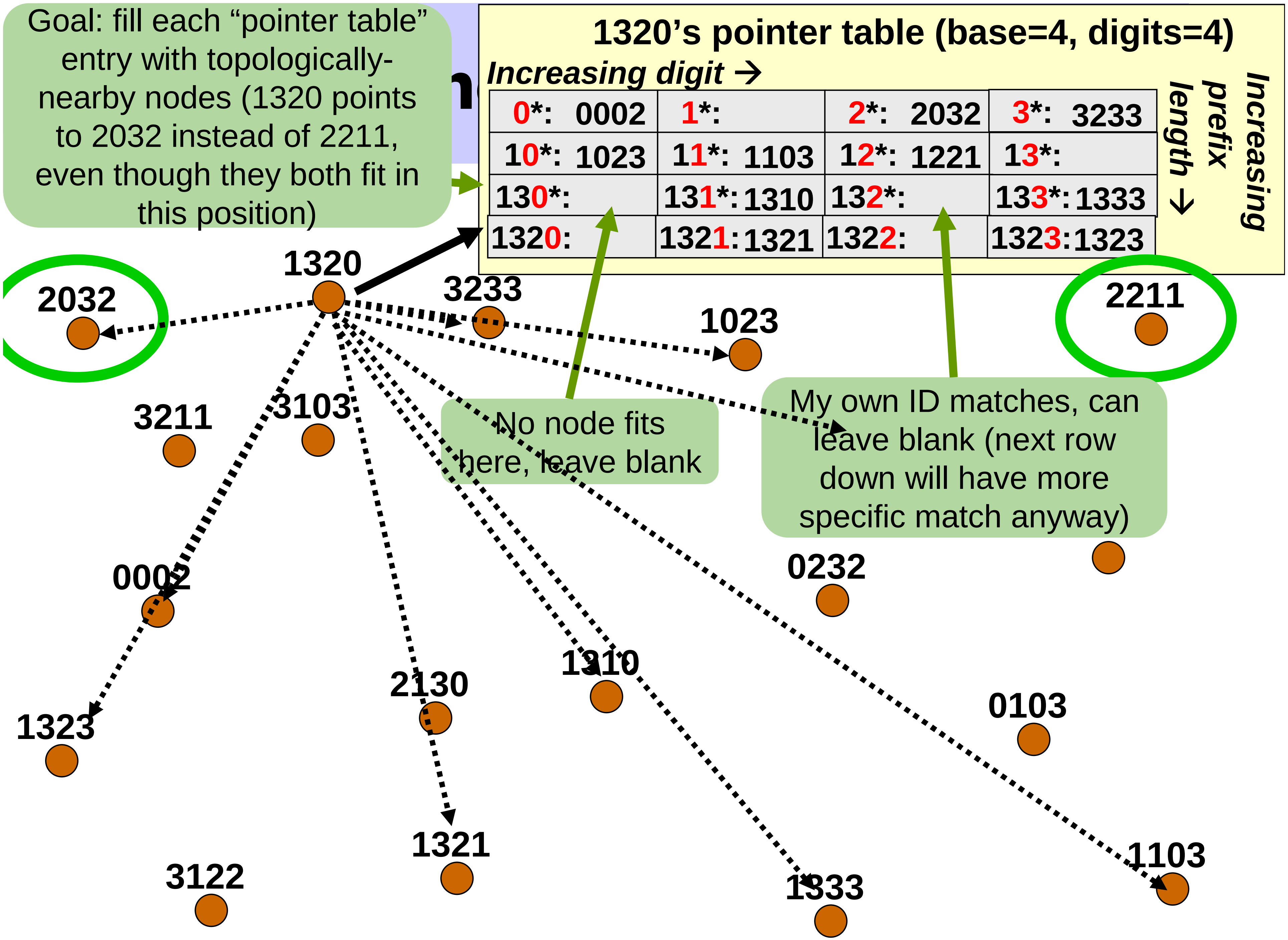
3. Successor informs predecessor of its new successor, adds joining node as new predecessor

Chord: Improving robustness

- To improve robustness, each node can maintain more than one successor
  - E.g., maintain the K>1 successors immediately adjacent to the node
- In the notify() message, node A can send its k-1 successors to its predecessor B
- Upon receiving the notify() message, B can update its successor list by concatenating the successor list received from A with A itself

Chord: Discussion

- Query can be implemented
  - Iteratively
  - Recursively
- Performance: routing in the overlay network can be more expensive than routing in the underlying network
  - Because usually **no** correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though both the initiator and the node that store them are in Europe!
  - Solutions: can maintain multiple copies of each entry in their finger table, choose closest in terms of network distance



Goal: fill each “pointer table” entry with topologically-nearby nodes (1320 points to 2032 instead of 2211, even though they both fit in this position)

1320’s pointer table (base=4, digits=4)

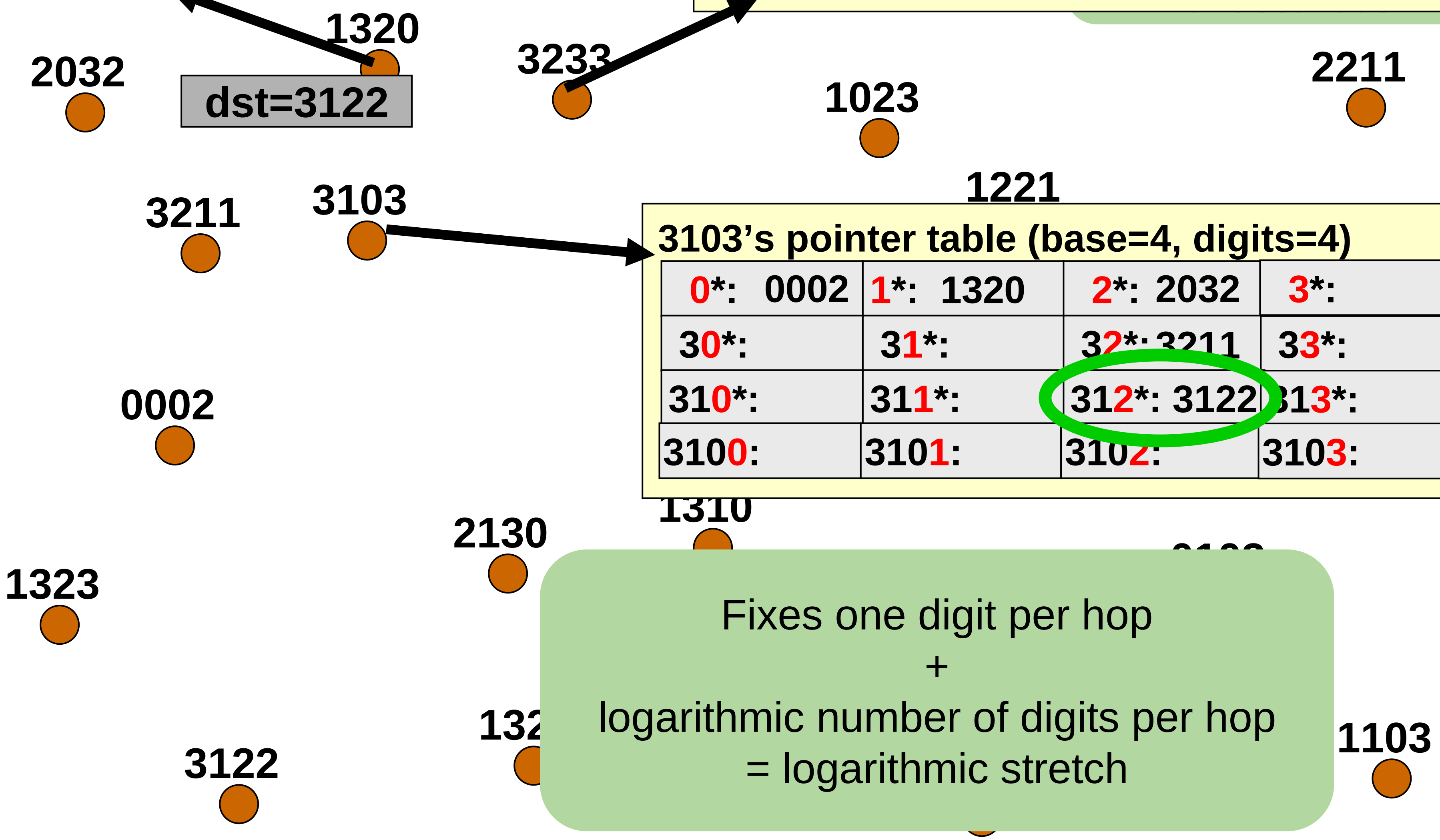
Increasing digit →

Increasing prefix length ↓

0*: 0002	1*:	2*: 2032	3*: 3233
10*: 1023	11*: 1103	12*: 1221	13*:
130*:	131*: 1310	132*:	133*: 1333
1320:	1321: 1321	1322:	1323: 1323

No node fits here, leave blank

My own ID matches, can leave blank (next row down will have more specific match anyway)



1320’s pointer table (base=4, digits=4)

0*: 0002	1*:	2*: 2032	3*:
10*: 1023	11*: 1103	12*: 1221	13*:
130*:	131*: 1310	132*:	133*
1320:	1321: 1321	1322:	1323

3233’s pointer table (base=4, digits=4)

0*: 0002	1*: 1320	2*: 2130	3*:
30*:	31*: 3103	32*: 3211	33*:
320*:	321*:	322*:	323*:
3230:	3231:	3232:	3233:

3103’s pointer table (base=4, digits=4)

0*: 0002	1*: 1320	2*: 2032	3*:
30*:	31*:	32*: 3211	33*:
310*:	311*:	312*: 3122	313*:
3100:	3101:	3102:	3103:

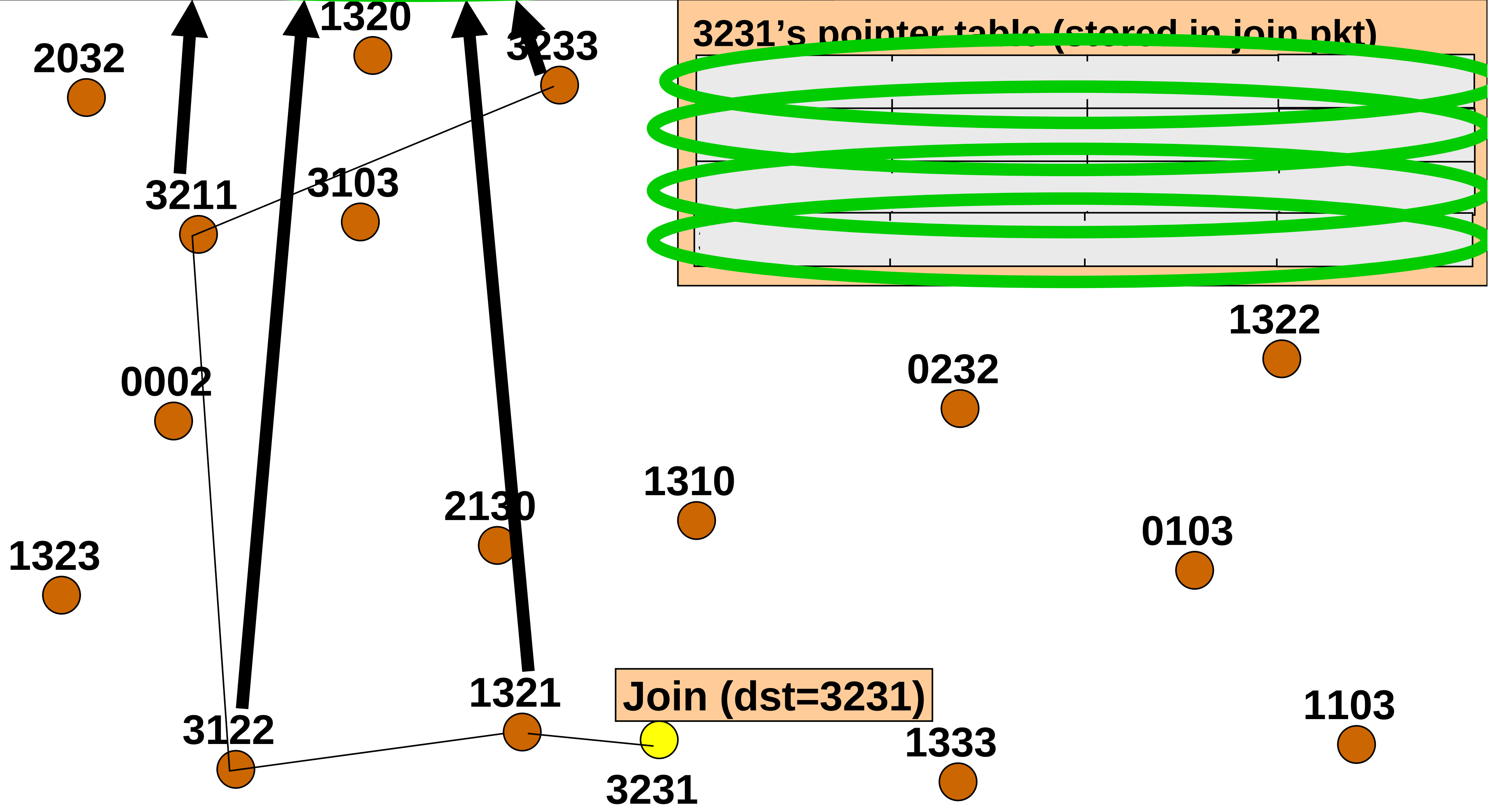
Fixes one digit per hop
+
logarithmic number of digits per hop
= logarithmic stretch

DHT

3233's pointer table (base=4, digits=4)

0*: 0002	1*: 1320	2*: 2032	3*:
30*:	31*: 3103	32*: 3211	33*:
320*:	321*: 3211	322*: 3221	323*: 3233
3230:	3231:	3232:	3233:

3231's pointer table (stored in join pkt)



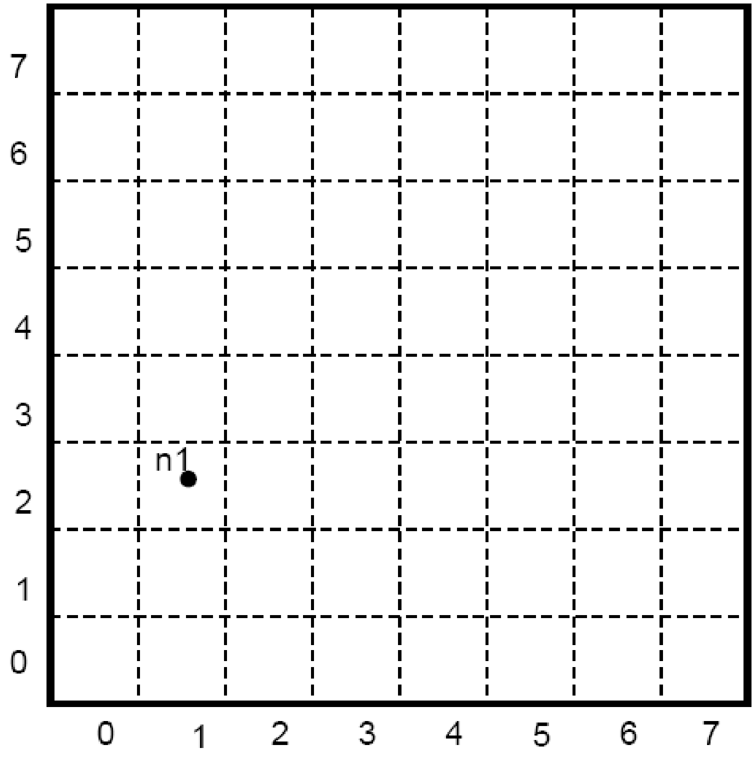
Join (dst=3231)

Content Addressable Network (CAN)

- Associate to each node and item a unique id in a d-dimensional space
- 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

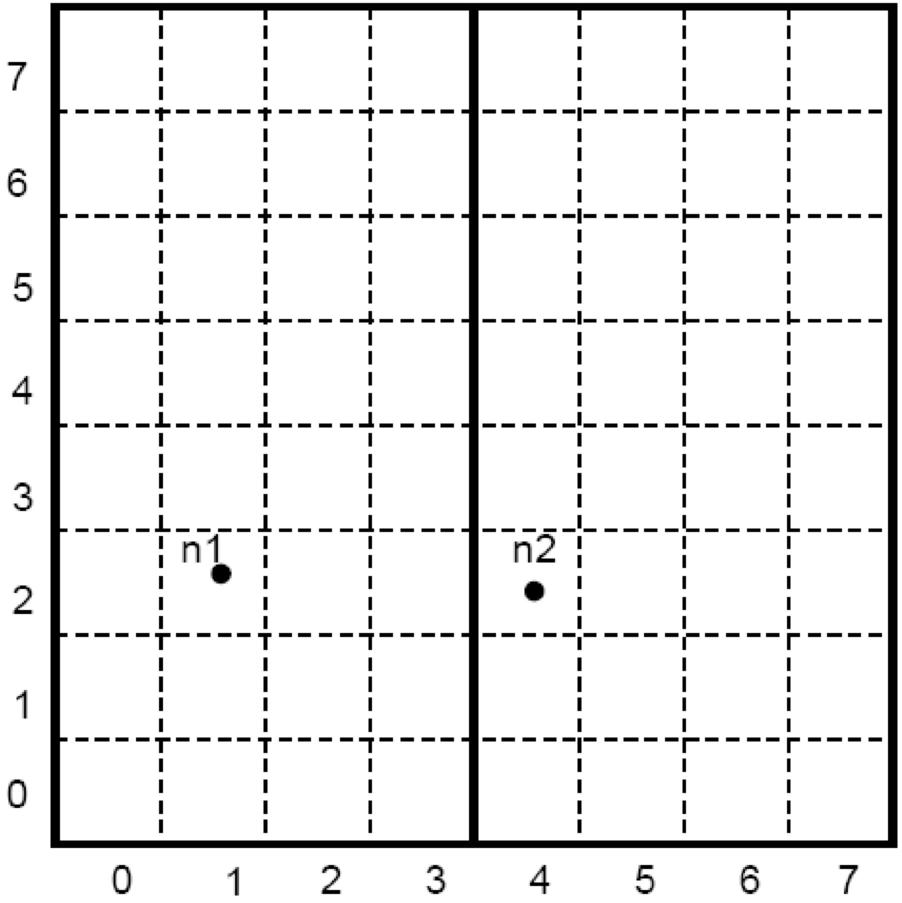
CAN Example:
Two dimensional space

- 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:
  - Assume space size (8x8)
  - Node n1:(1,2) first node that joins
    - Cover the entire space



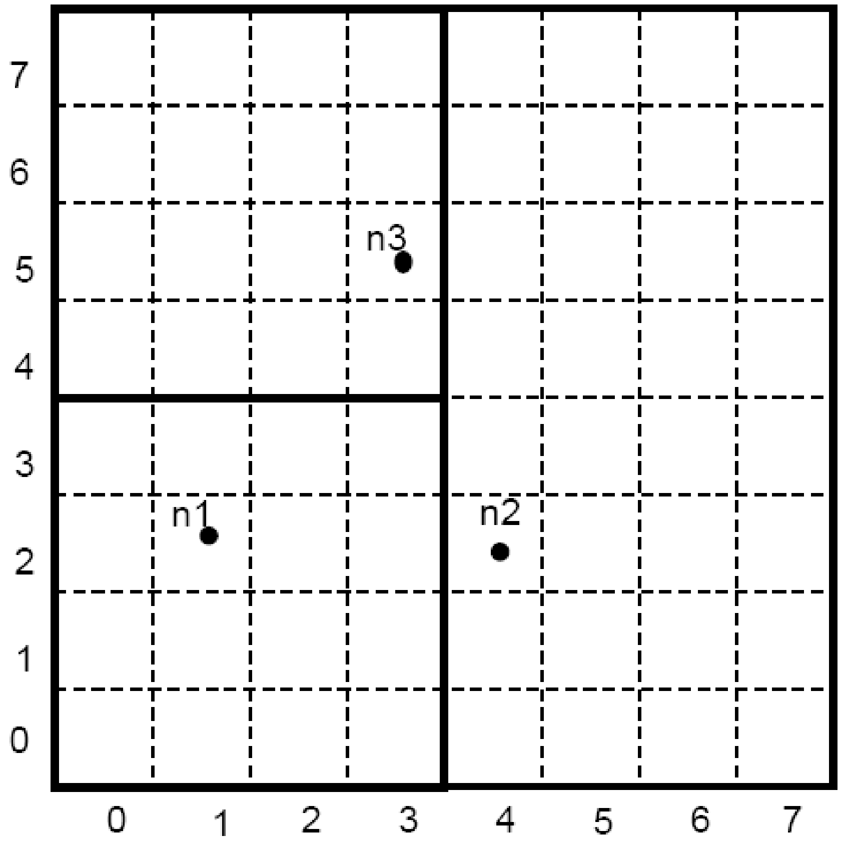
CAN Example: Two dimensional space

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



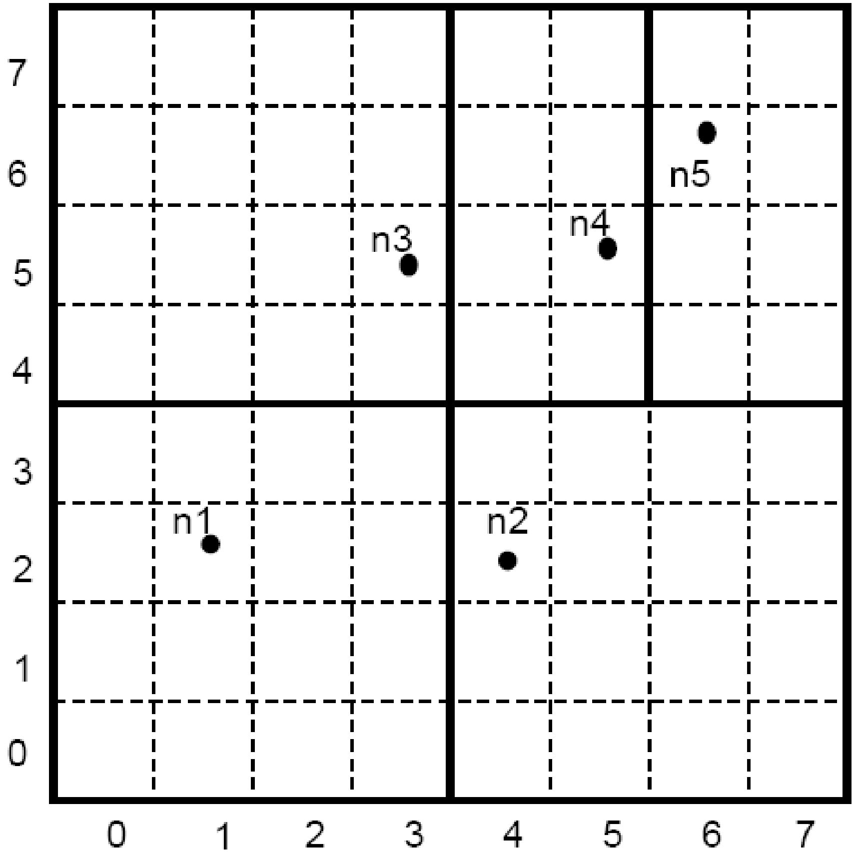
CAN Example:
Two dimensional space

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



CAN Example:
Two dimensional space

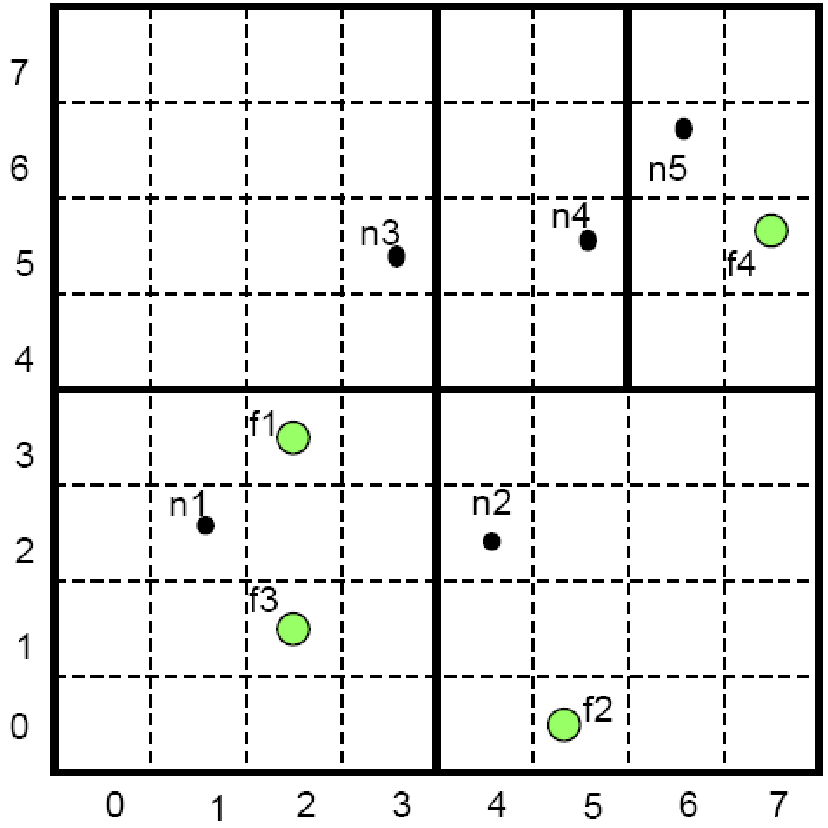
- Nodes n4:(5,5) and n5:(6,6) join



CAN Example: Two dimensional space

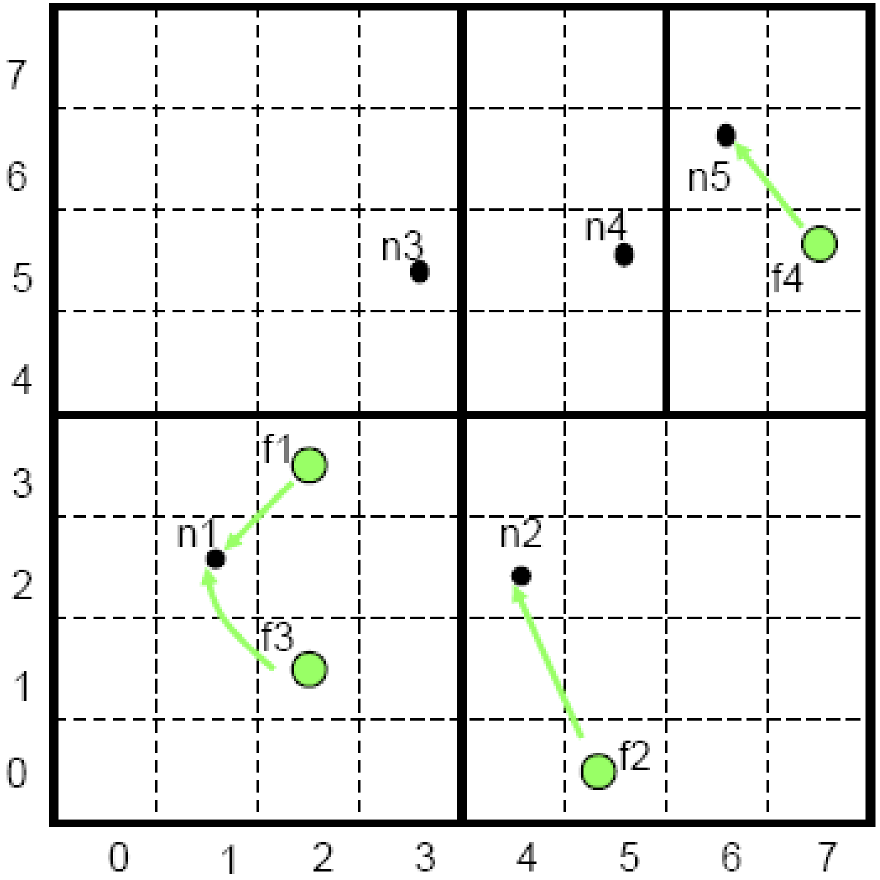
- Nodes:
  - n1:(1,2)
  - n2:(4,2)
  - n3:(3,5)
  - n4:(5,5)
  - n5:(6,6)

- Items:
  - f1(2,3)
  - f2(5,1)
  - f3:(2,1)
  - f4(7,5)



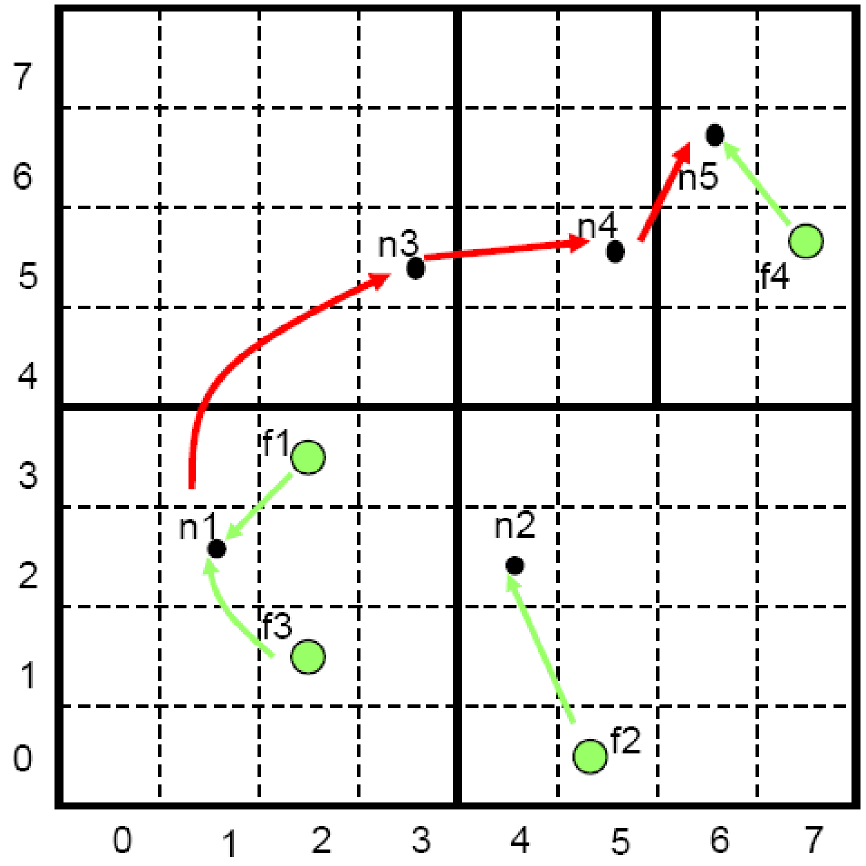
CAN Example:
Two dimensional space

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



CAN Example: Two dimensional space

- 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

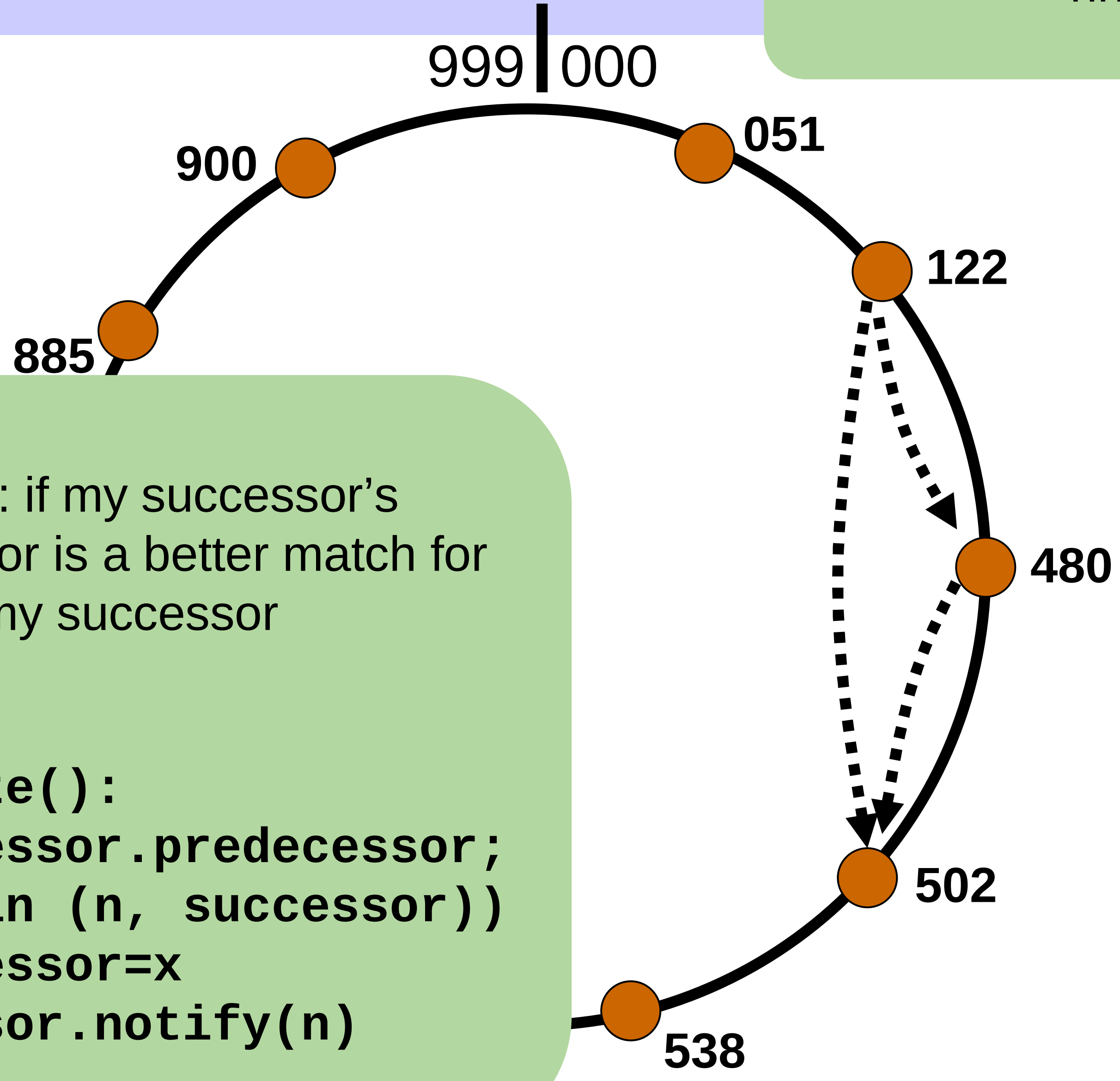


Preserving consistency

- What if a node fails?
  - Solution: probe neighbors to make sure alive, proactively replicate objects
- What if node joins in wrong position?
  - Solution: nodes check to make sure they are in the right order
  - Two flavors: *weak* stabilization, and *strong* stabilization

Chord Example: weak

Tricky case: zero position on ring

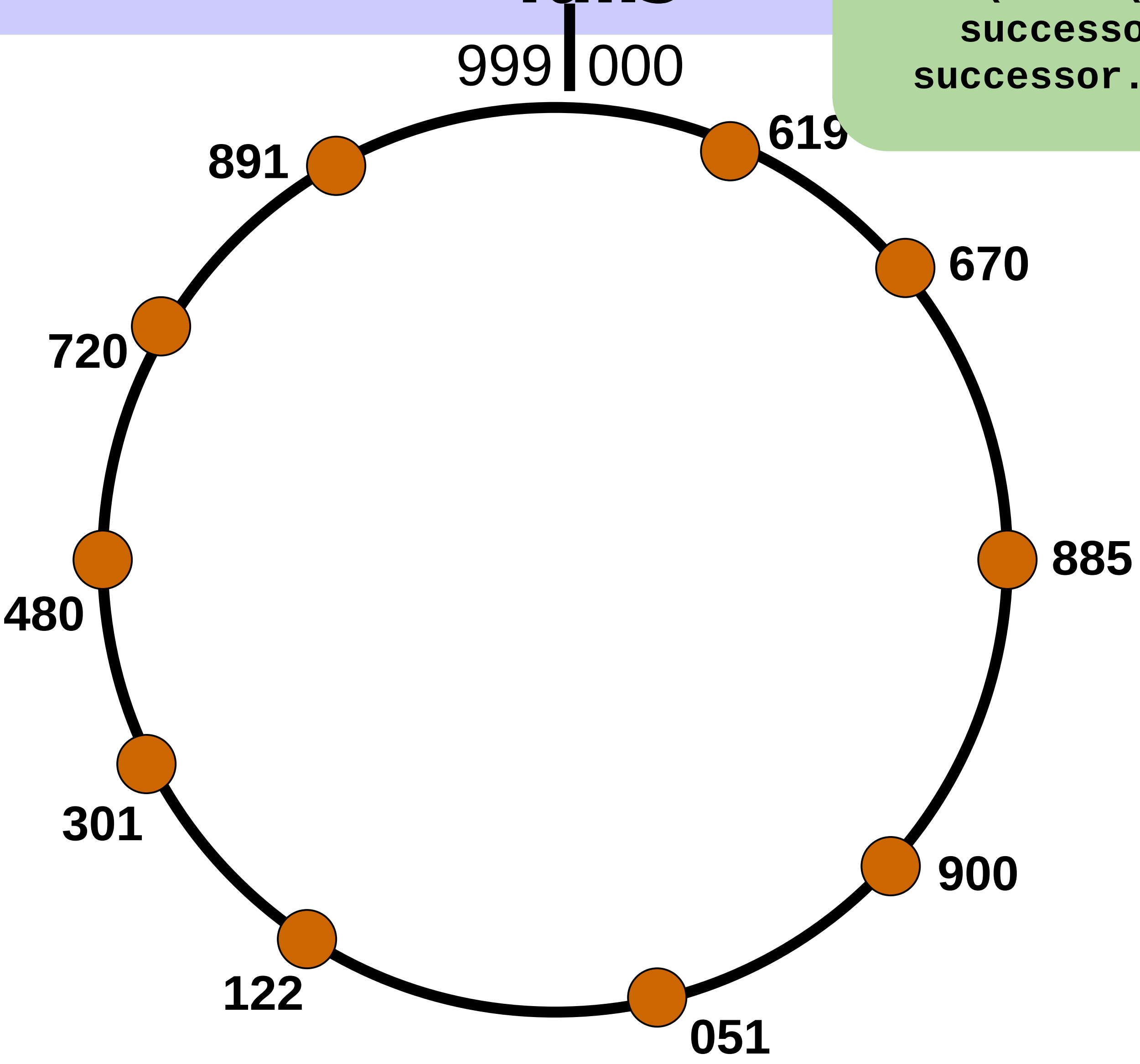


Check: if my successor's predecessor is a better match for my successor

```
n.stablize():
  x=successor.predecessor;
  if (x in (n, successor))
    successor=x
  successor.notify(n)
```

Example where weak st
fails

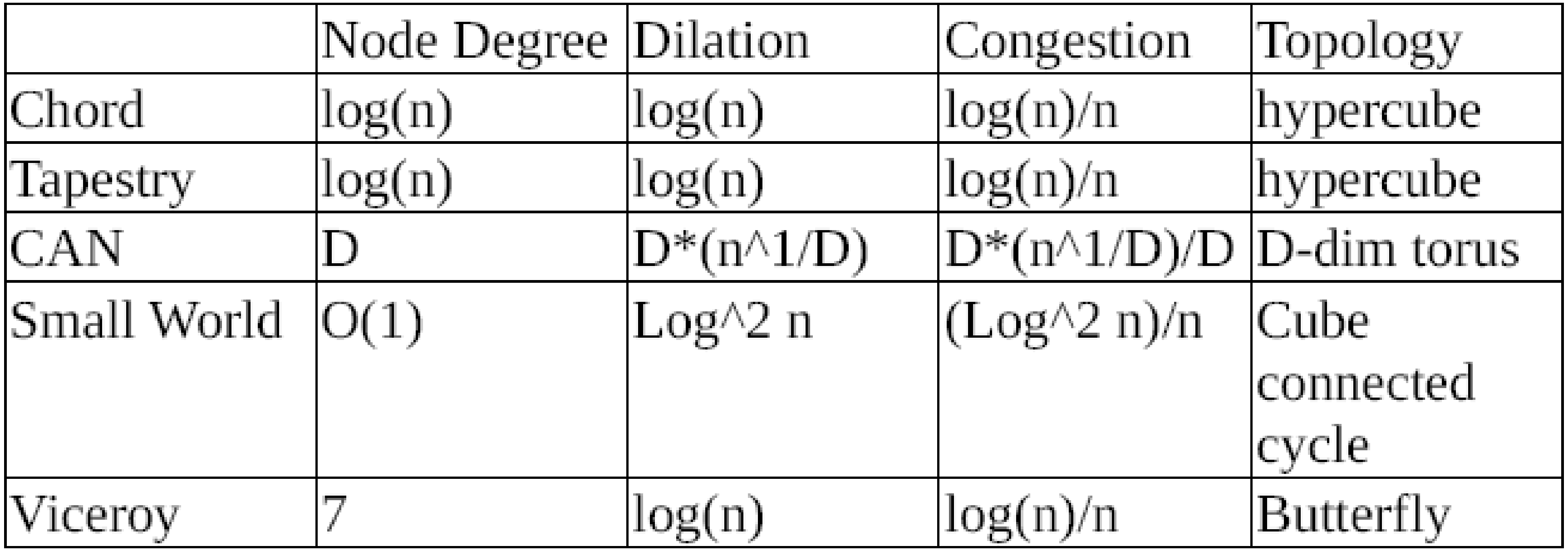
```
n.stablize():
  x=successor.predecessor;
  if (x in (n, successor))
    successor=x
  successor.notify(n)
```



Comparison of DHT geometries

Geometry	**Algorithm**
Ring	Chord, Symphony
Hypercube	CAN
Tree	Plaxton
Hybrid = Tree + Ring	Tapestry, Pastry
XOR d(id1, id2) = id1 XOR id2	Kademlia

Comparison of DHT algorithms



	Node Degree	Dilation	Congestion	Topology
Chord	log(n)	log(n)	log(n)/n	hypercube
Tapestry	log(n)	log(n)	log(n)/n	hypercube
CAN	D	D*(n^1/D)	D*(n^1/D)/D	D-dim torus
Small World	O(1)	Log^2 n	(Log^2 n)/n	Cube connected cycle
Viceroy	7	log(n)	log(n)/n	Butterfly

- Node degree: The number of neighbors per node
- Dilation: Length of longest path that any packet traverses in the network
  - Stretch: Ratio of longest path to shortest path through the underlying topology
- Congestion: maximum number of paths that use the same link

Security issues

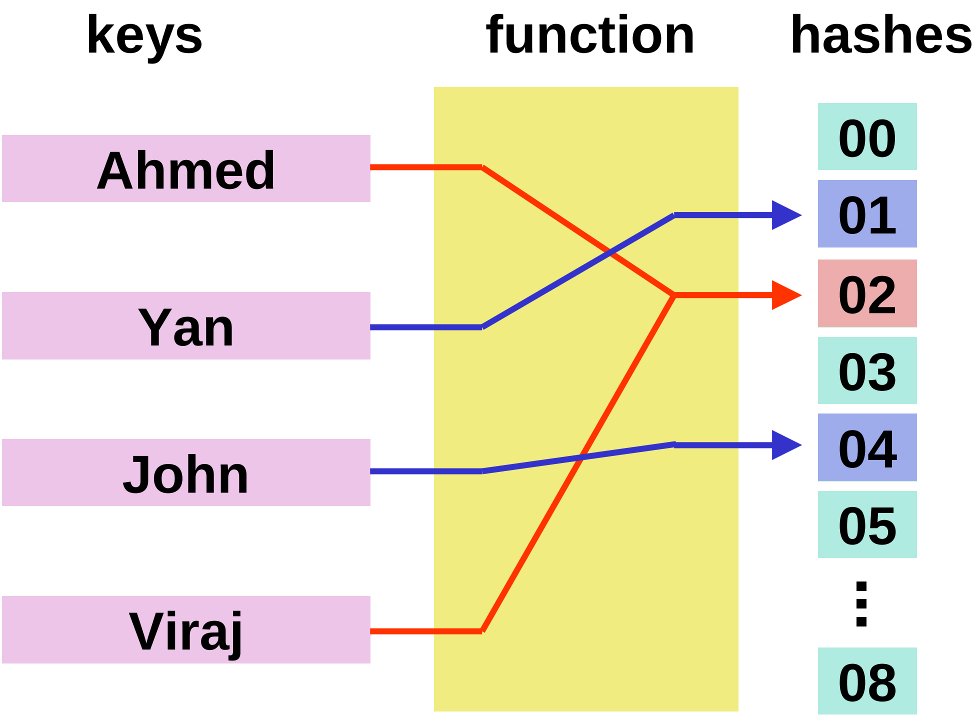
- Sybil attacks
  - Malicious node pretends to be many nodes
  - Can take over large fraction of ID space, files
- Eclipse attacks
  - Malicious node intercepts join requests, replies with its cohorts as joining node's fingers
- Solutions:
  - Perform several joins over diverse paths, PKI, leverage social network relationships, audit by sharing records with neighbors

Hashing in networked software

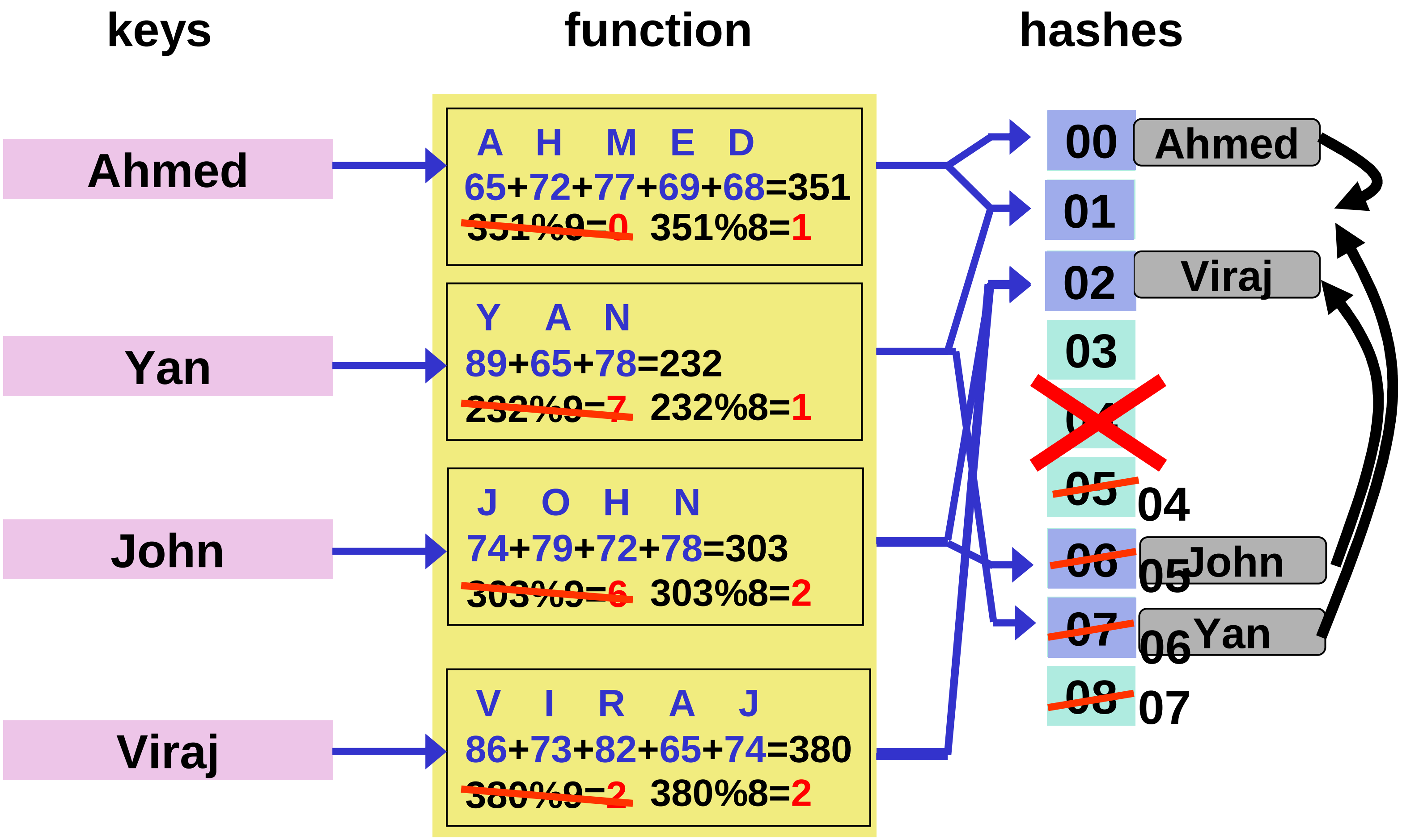
- Hash table: maps identifiers to keys
  - Hash function used to transform key to index (slot)
  - To balance load, should ideally map each key to different index

- Distributed hash tables
  - Stores values (e.g., by mapping keys and values to servers)
  - Used in distributed storage, load balancing, peer-to-peer, content distribution, multicast, anycast, botnets, BitTorrent's tracker, etc.

Background: hashing



Example



keys

function

hashes

Ahmed

Yan

John

Viraj

A H M E D
65+72+77+69+68=351
~~351%9=0~~ 351%8=1

Y A N
89+65+78=232
~~232%9=7~~ 232%8=1

J O H N
74+79+72+78=303
~~303%9=6~~ 303%8=2

V I R A J
86+73+82+65+74=380
~~380%9=2~~ 380%8=2

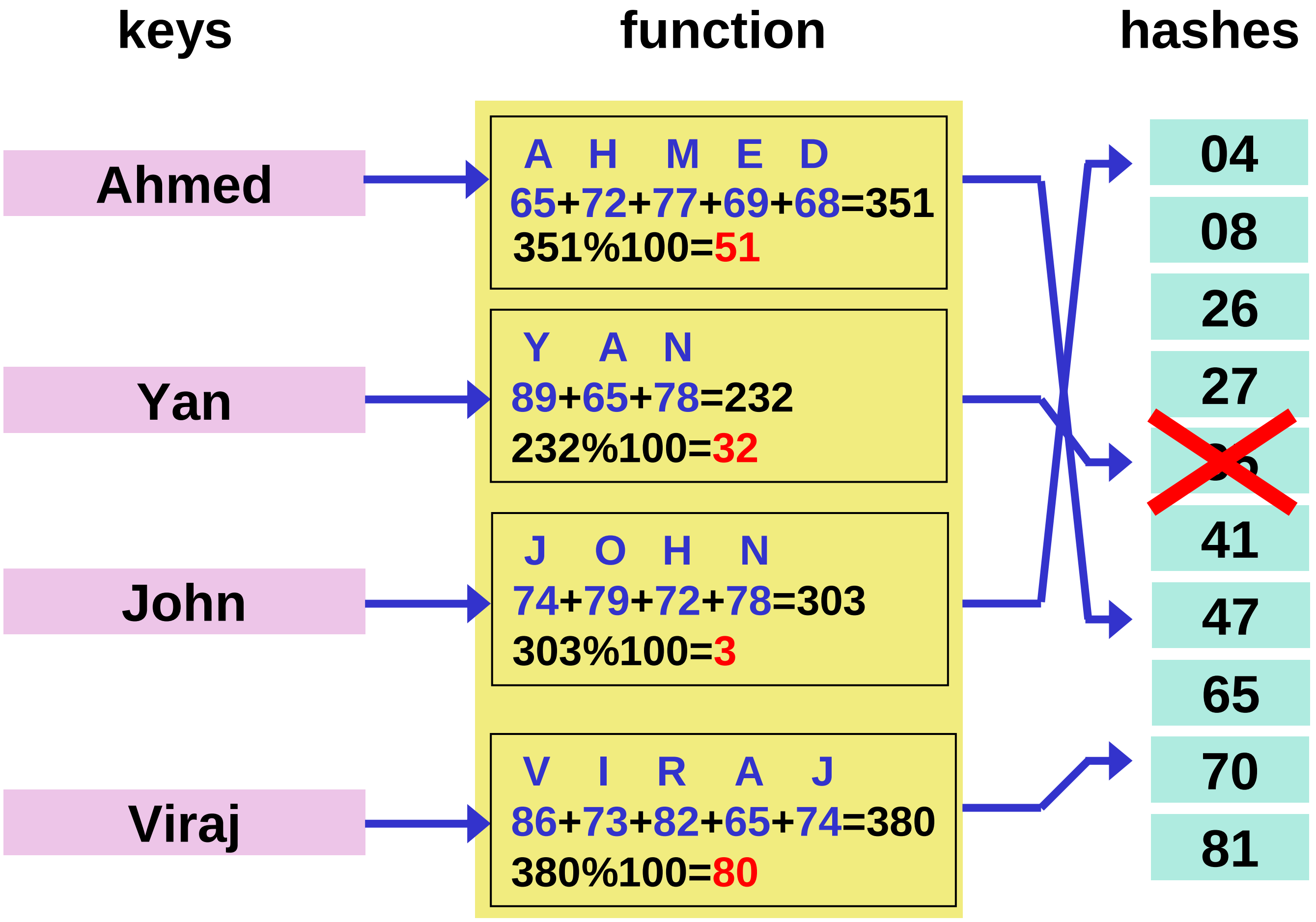
00 Ahmed
01
02 Viraj
03
~~05~~ 04
~~06~~ 05 John
~~07~~ 06 Yan
~~08~~ 07

- Example: Sum ASCII digits, mod number of bins
- Problem: ____________________

Solution: Consistent Hashing

- Hashing function that reduces churn
- Addition or removal of one slot does not significantly change mapping of keys to slots
- Good consistent hashing schemes change mapping of K/N entries on single slot addition
  - K: number of keys
  - N: number of slots
- E.g., map keys and slots to positions on circle
  - Assign keys to closest slot on circle

Solution: Consistent Hashing



keys

function

hashes

Ahmed

Yan

John

Viraj

A H M E D
65+72+77+69+68=351
351%100=51

Y A N
89+65+78=232
232%100=32

J O H N
74+79+72+78=303
303%100=3

V I R A J
86+73+82+65+74=380
380%100=80

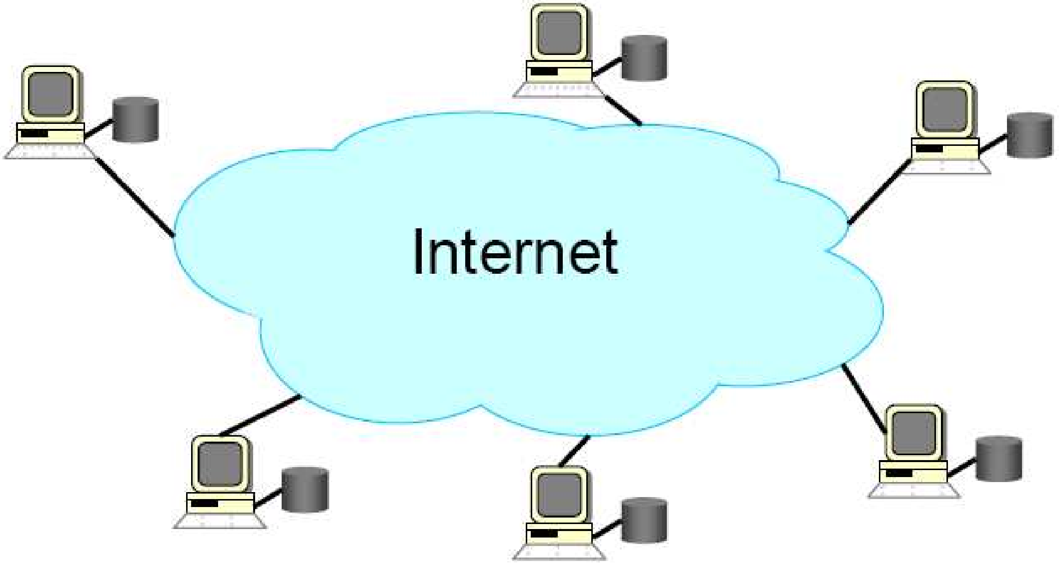
04
08
26
27
41
47
65
70
81

- Slots have IDs selected randomly from [0,100]
- Hash keys onto same space, map key to closest bin
- Less churn on failure → more stable system

Peer-to-peer networking

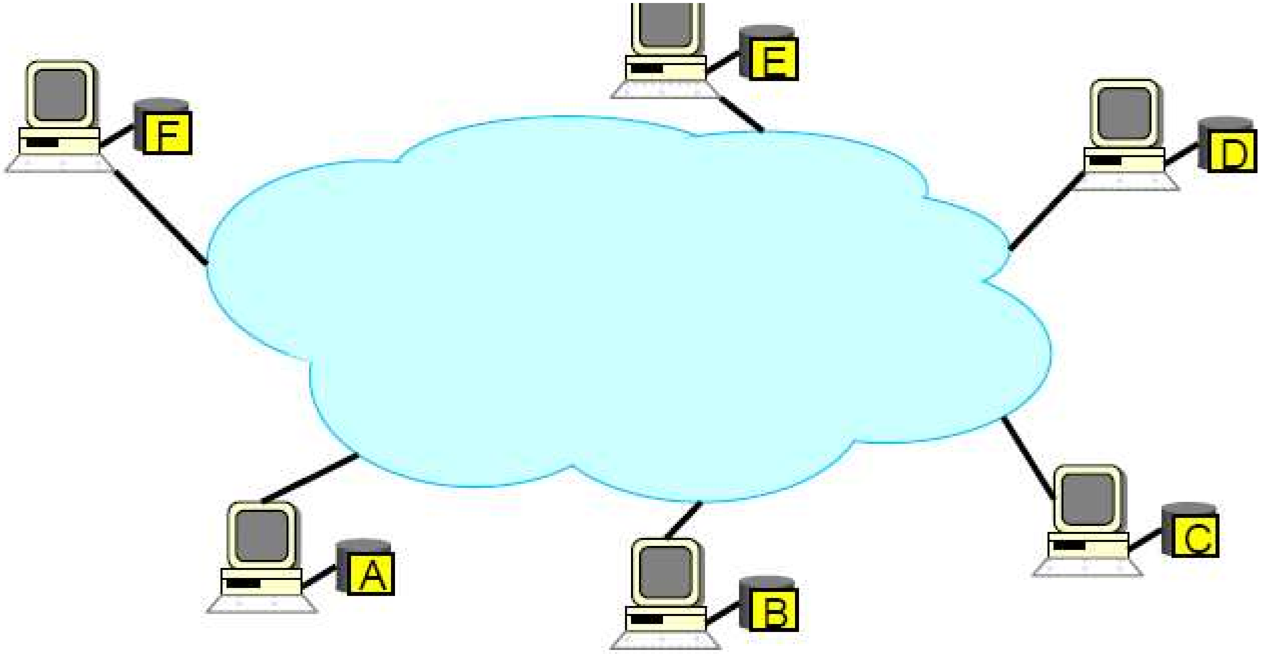
How did it start?

- A killer application: Napster (1999)
  - Free music over the Internet
- Key idea: share storage and bandwidth of individual (home) users



Model

- Each user stores a subset of files
- Each user has access (can download) files from all users in the system



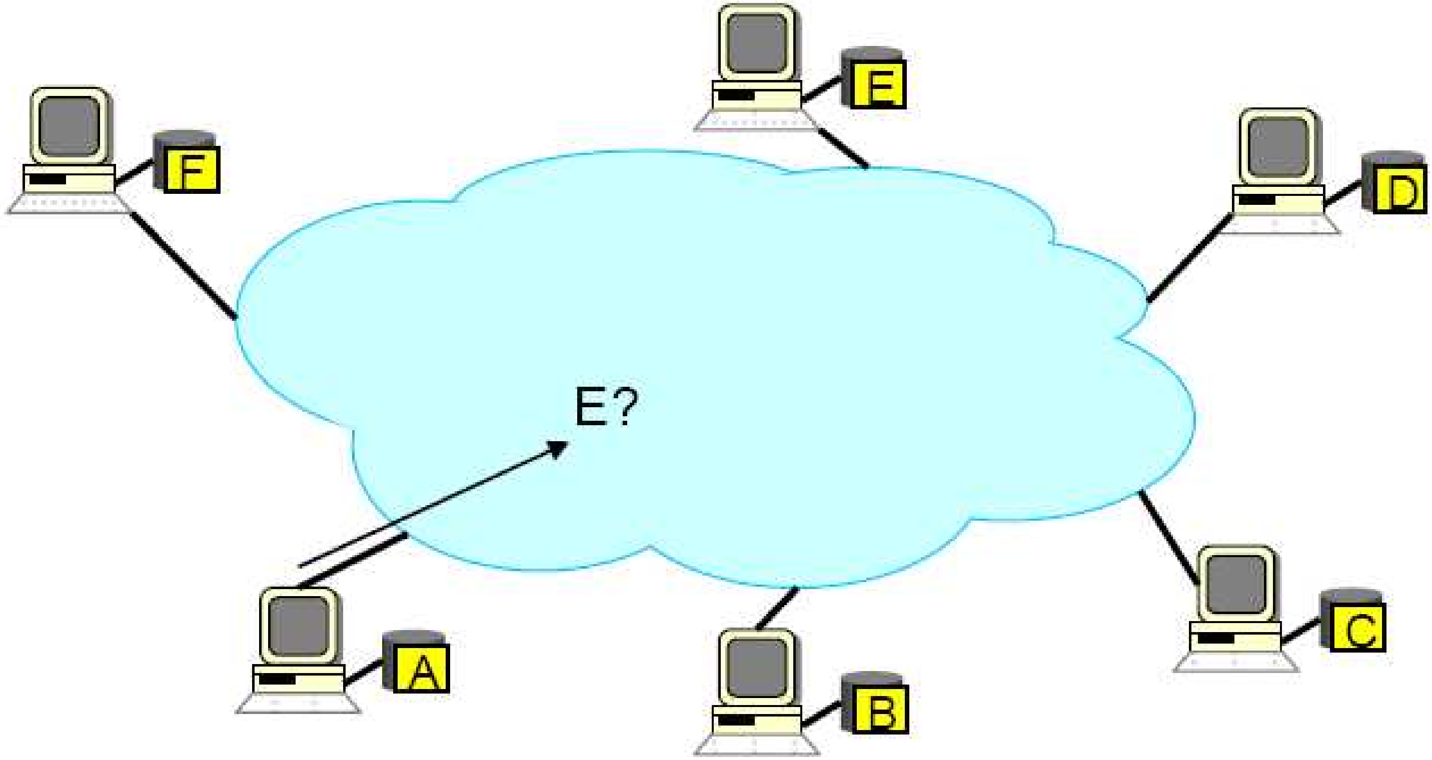
Relationship to DHTs and Overlays

- DHTs like Chord allow distributed object storage
  - Hosts can “put” and “get” objects
  - Objects referenced by well-known key (e.g., hash of file contents)

- However in unmanaged networks, hosts don’t have incentive to store other’s objects
  - I download files I want on my local host
  - May be willing to share my files

Main challenge

- Find where file is stored



Other challenges

- Scale: up to millions of machines
- Dynamicity: machines can come and go at any time

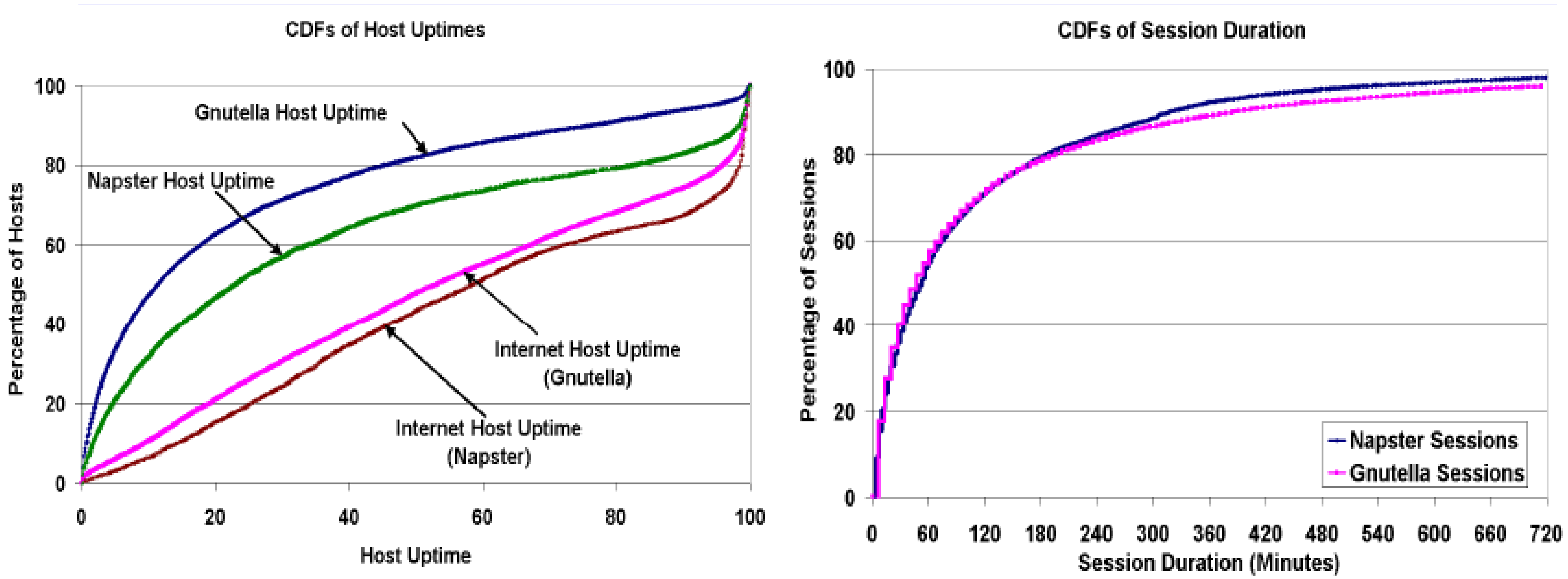
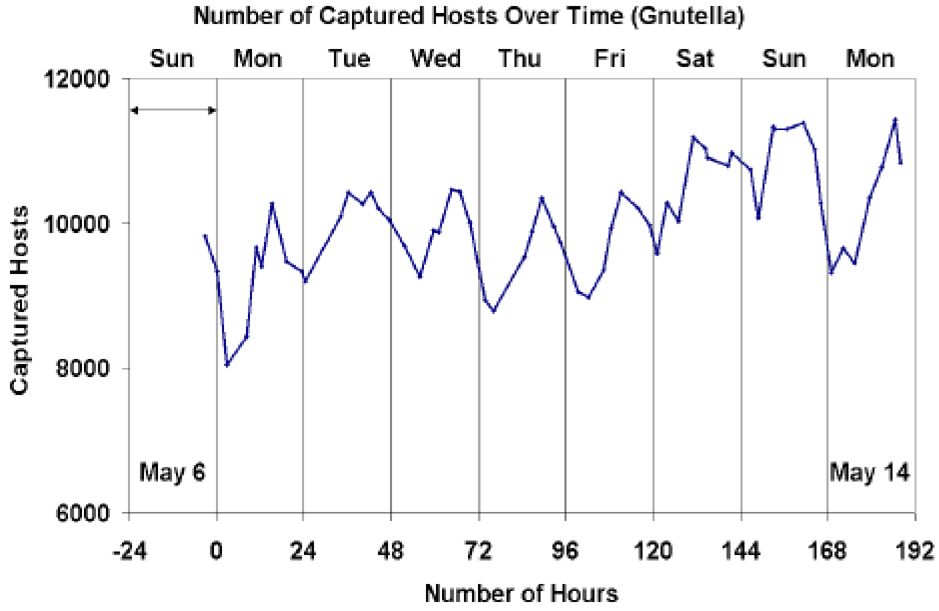


Fig. 9 Left: IP-level uptime of peers ("Internet Host Uptime"), and application-level uptime of peers ("Gnutella/Napster Host Uptime") in both Napster and Gnutella, as measured by the percentage of time the peers are reachable; Right: The distribution of Napster/Gnutella session durations

- P2P networks are <u>dynamic</u>



Other challenges

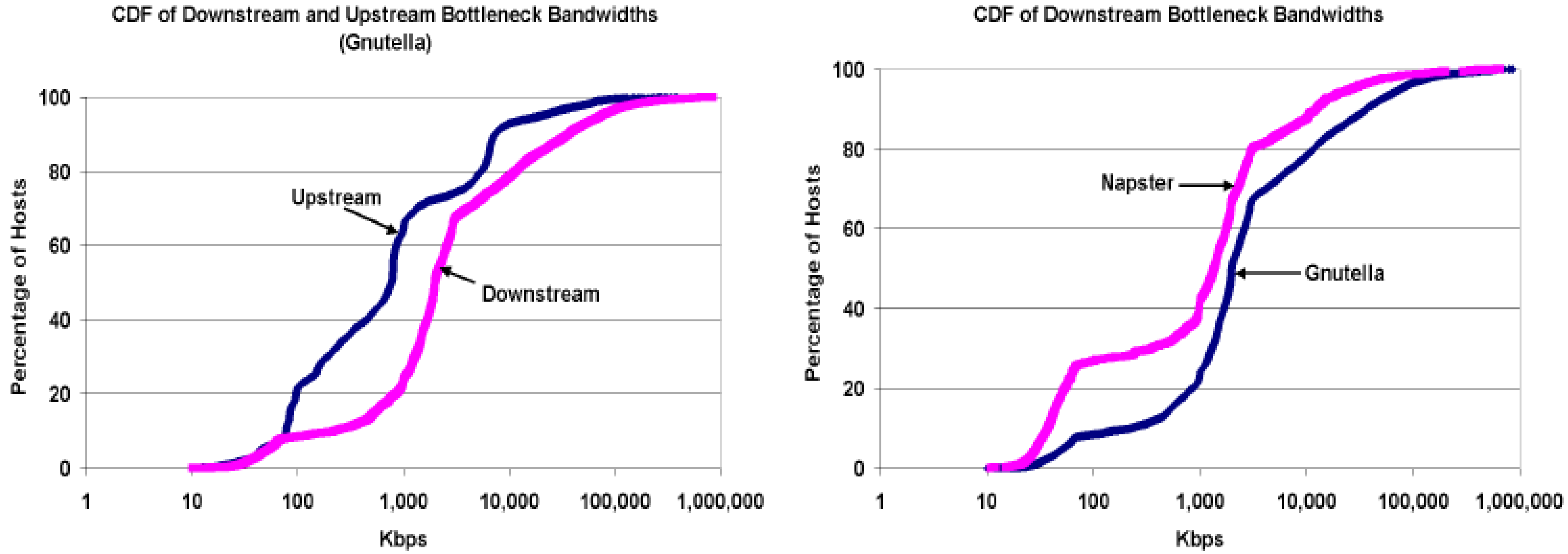


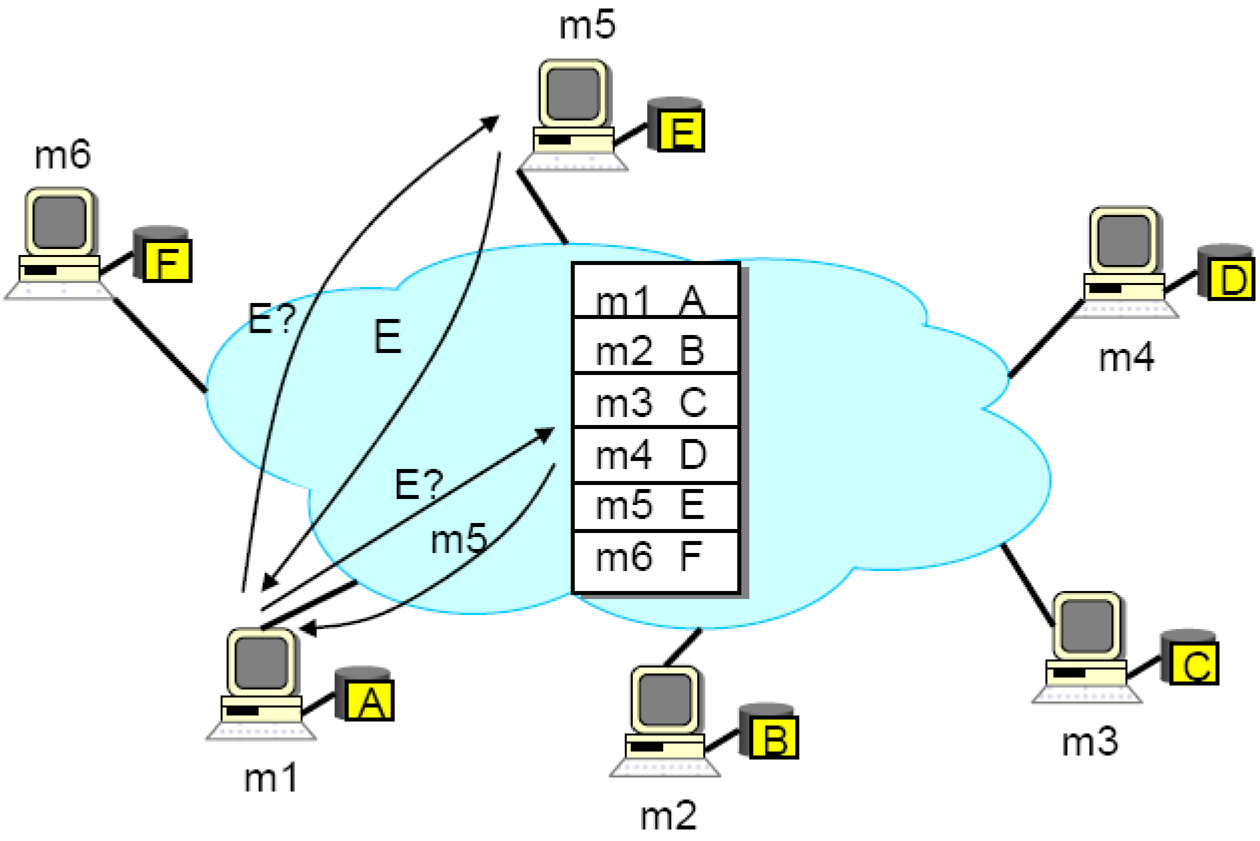
Fig. 6 Left: CDFs of upstream and downstream bottleneck bandwidths for Gnutella peers; Right: CDFs of downstream bottleneck bandwidths for Napster and Gnutella peers.

- P2P networks are <u>heterogeneous</u>

Napster

- Assume a centralized index system that maps files (songs) to machines that are alive
- How to find a file (song)
  - Query the index system --> return a machine that stores the required file
    - Ideally this is the closest/least-loaded machine
  - FTP the file
- Advantages
  - Simplicity, easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness, scalability (?)

Napster example



The aftermath

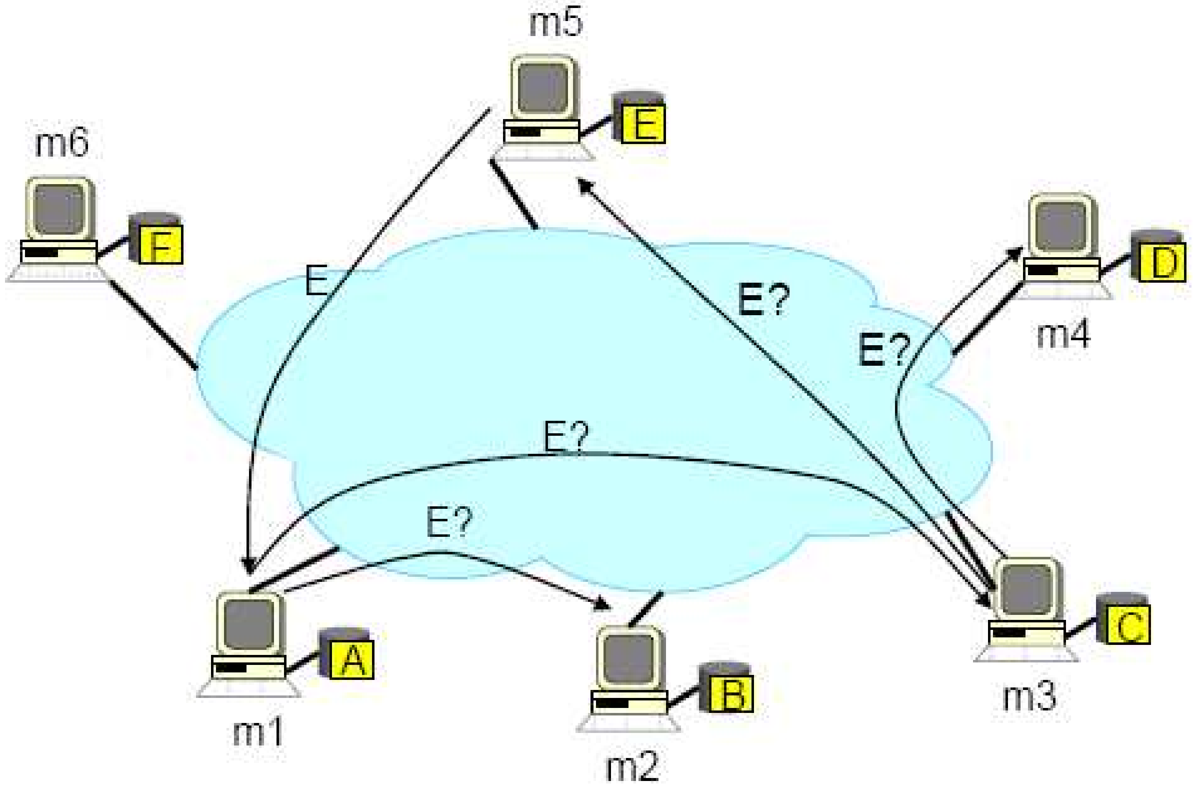
- **“Recording industry association of America (RIAA) sues music startup Napster for $20 Billion”** – December 1999
- **“Napster ordered to remove copyrighted material”** – March 2001
- Main legal argument:
  - Napster owns the index system, so it is directly responsible for disseminating copyrighted material

Gnutella (2000)

- Distribute file location
- Idea: broadcast the request
- How to find a file?
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machines that has the file receives the request, and it sends back the answer
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with requests (to alleviate this problem, each request has a TTL)

Gnutella: Example

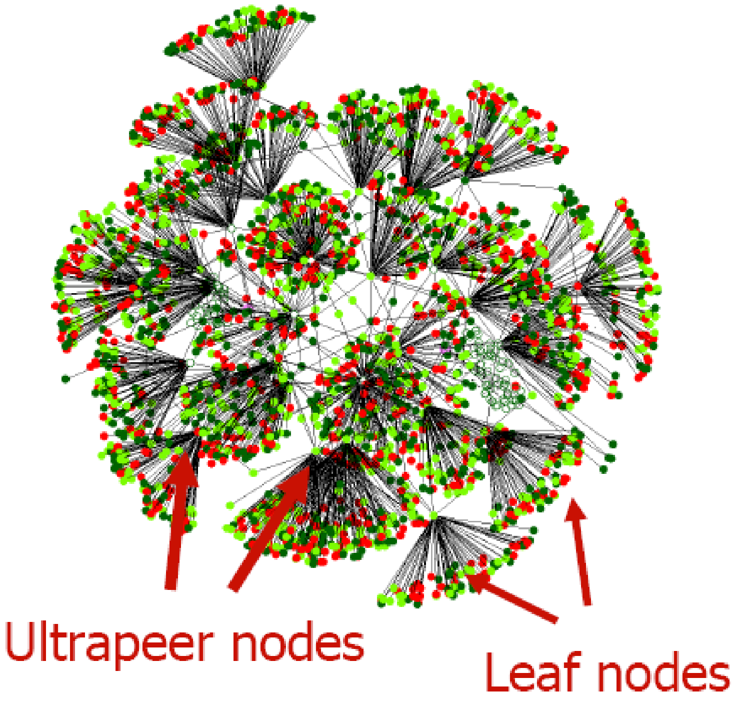
- Assume: m1's neighbors are m2 and m3; m3's neighbors are m4 and m5;…



Two-level hierarchy

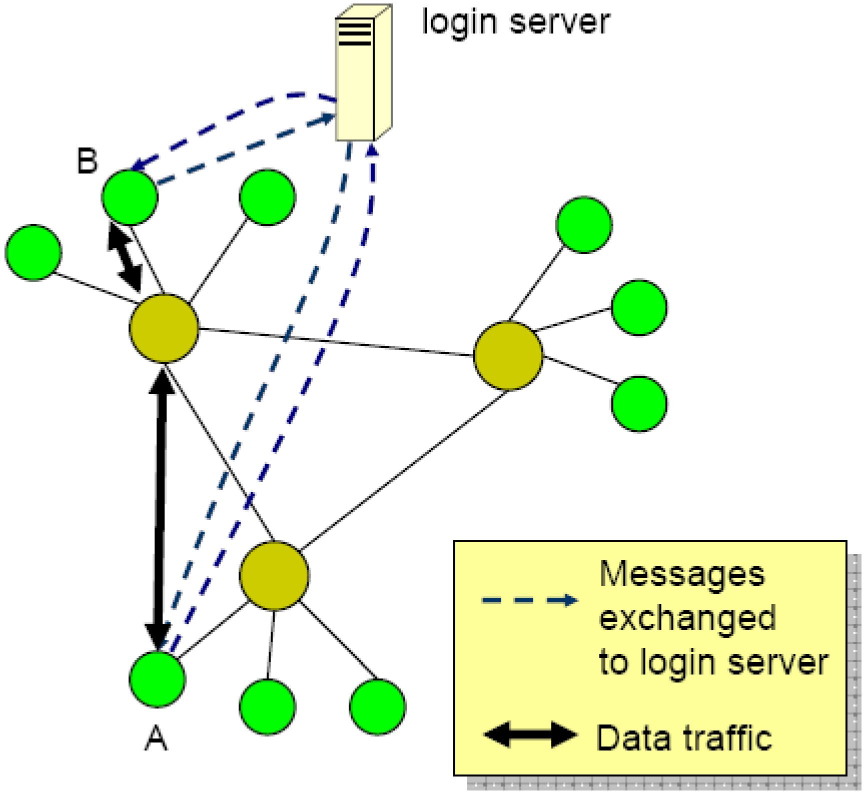
- Current Gnutella implementation, KaZaa
- Leaf nodes are connected to a small number of ultrapeers (supernodes)
- Query
  - A leaf sends query to its ultrapeers
  - If ultrapeers don't know the answer, they flood the query to other ultrapeers
- More scalable:
  - Flooding only among ultrapeers

Oct 2003 crawl
Of Gnutella



Skype (2003)

- Peer-to-peer Internet telephony
- Two-level hierarchy like KaZaa
- Ultrapeers used to route traffic between NATed end-hosts
- Plus a login server to
  - Authenticate users
  - Ensure that names are unique across network



(Note*: probable protocol; Skype protocol is not published)

BitTorrent (2001)

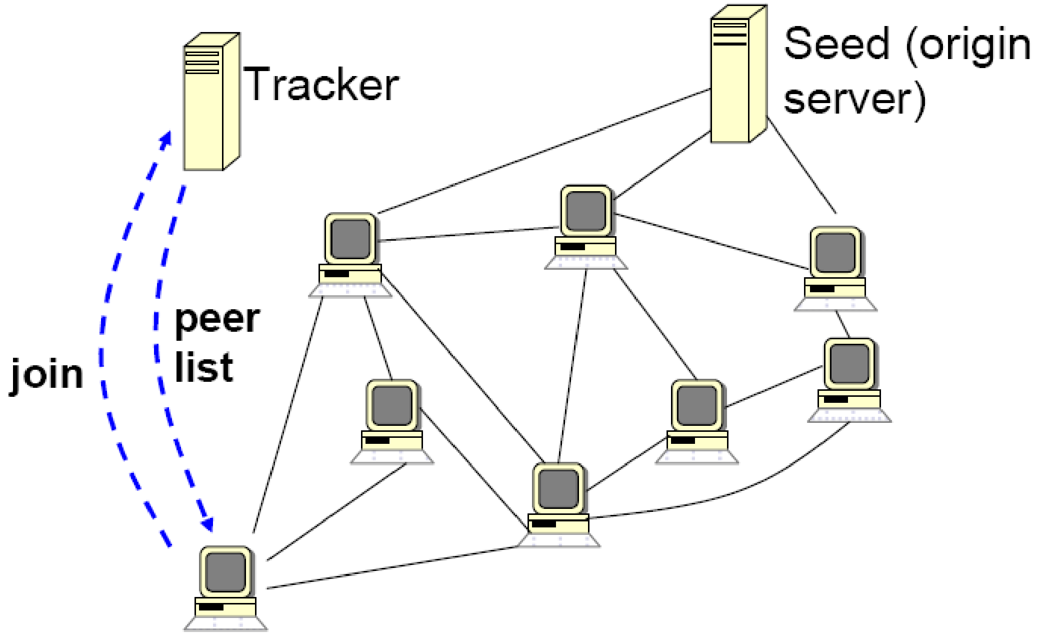
- Has become most common protocol for transferring large files
  - 27-55% of all Internet traffic
  - Estimated 1.7 petabytes source content shared in 2008
- Model:
  - Origin server wishes to distribute file (seed) to other hosts (peers)
  - Once multiple hosts have multiple pieces of the file, they may become source for that part of the file
  - Once a host downloads the entire file, it may become a new seed

BitTorrent (2001)

- Goal: allow fast downloads even when sources have low up-link capacity
- How does it work?
  - Seed (origin) – site storing the file to be downloaded
  - Tracker – server maintaining list of peers in system
  - Split each file into pieces (~256 KB each), and each piece into sub-pieces (~16 KB each)
  - The loader loads one piece at a time
  - Within one piece, the loader can load up to five sub-pieces in parallel

BitTorrent: Join Procedure

1. Peer contacts tracker responsible for file it wants to download
2. Tracker returns a list of peers (20-50) downloading the same file
3. Peer connects to peer in the list



BitTorrent: Download Algorithm

- Download consists of three phases
- Start: get a piece as soon as possible
  - Select a random piece
- Middle: spread all pieces as soon as possible
  - Select rarest piece next
- End: avoid getting stuck with a slow source when downloading the last sub-pieces
  - Request in parallel the same sub-piece
  - Cancel slowest downloads once a sub-piece has been received

- (For details see: http://bittorrent.org/bittorrentecon.pdf)

BitTorrent

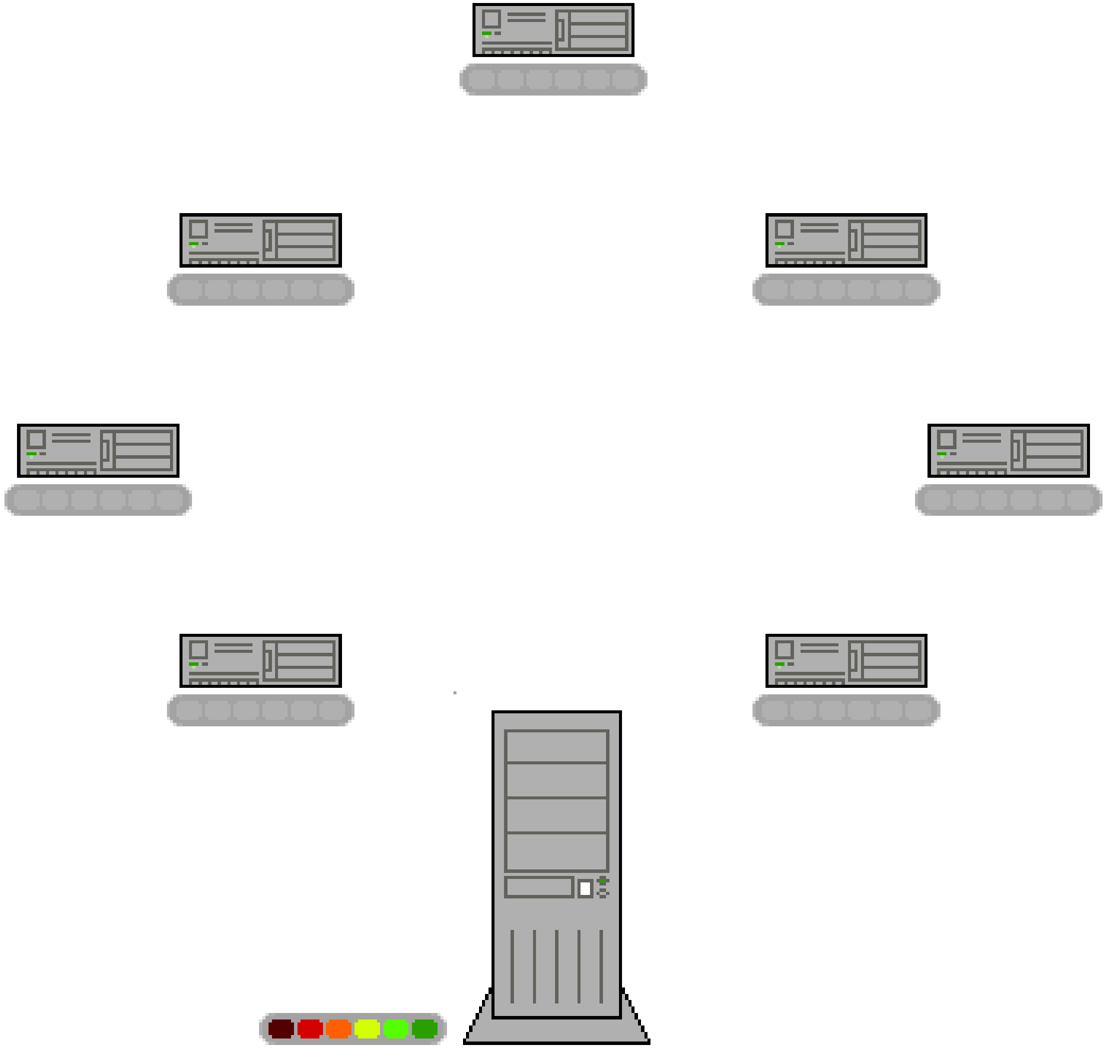
- Benefits:
  - Significant reduction in origin’s hardware and bandwidth requirements
    - Don’t need a big server farm to handle a flash crowd
  - Provides redundancy against outages
  - Provides a temporary source, which is harder to trace

BitTorrent Protocol

- To share a file, peer first creates a **torrent file**, containing metadata about files to be shared
  - Checksum for each file “chunk” (which are typically between 64KB and 4MB)
  - URL of the tracker
  - Names of files, their lengths, chunk length used

- Torrent files are then registered with a **tracker**
  - Maintains list of clients currently participating in torrent
  - Alternatively, some clients use DHT in place of tracker

BitTorrent Protocol



BitTorrent

- Users browse the web to find a torrent of interest, download and open with a BitTorrent client
  - Client then connects to trackers specified in torrent file
  - Receives list of peers currently transferring file chunks
  - Client then connects to peers to receive the chunks it needs

Smart selection of chunks speeds download

- Downloading in random order increases opportunity to exchange data
- **“tit-for-tat”**, where clients prefer to send data to clients that send data back to them
  - Problem: two clients don’t share data because neither takes the initiative
  - Problem: when node first joins it may take some time to gain a strong enough reputation to get data from peers
- **“optimistic unchoking”**, where client reserves part of its bandwidth to send chunks to random peers

The BitTorrent Controversy

- Some groups object to bittorrent
  - Content owners: significant number of torrents host copyrighted material
  - ISP networks: significant rise in BitTorrent network increases congestion, harms performance for delay sensitive traffic
  - Enterprise networks: BitTorrent often contacts 300-500 servers per second! Rapidly fills up NAT tables

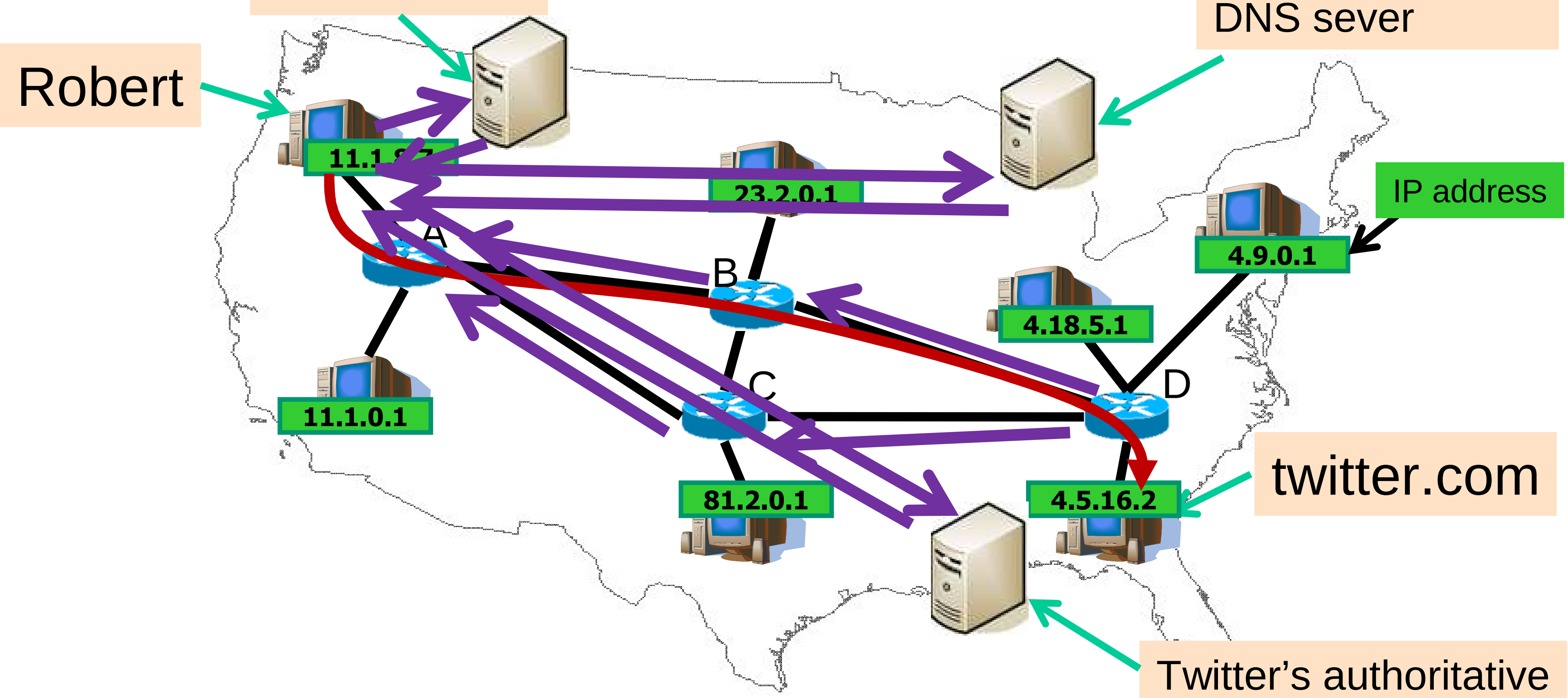
- ISPs have begun rate-limiting BitTorrent
  - So, BitTorrent clients began using headerencryption
  - So, ISPs began to use “deep packet inspection” to look past header
  - → Arms race

Limitations of BitTorrent

- Lack of anonymity
  - Possible to obtain IP addresses of clients from the tracker
- Leeching
  - User may leave swarm after downloading without seeding
  - Can block users that don’t upload much, but this harms dial-up and asymmetric broadband users
- Speed
  - Download speed limited by bandwidth of peers. Problem if many peers are on asymmetric connections

- → Future clients and ongoing development may rectify these limitations

Review: Content distribution



Robert’s local DNS server

.com authoritative DNS sever

Robert

IP address

twitter.com

Twitter’s authoritative DNS server

Review: Content distribution