Lecture 6: Securing Distributed and Networked Systems

CS 598: Network Security

Matthew Caesar

March 12, 2013

Today: Distributed Internet Services

- Previous cycle: how to build Internet services that run at a single location
- However, some modern services are built across many locations
  - Content distributed over the wide area, multiple sites
  - Need techniques to coordinate operations of distributed software running in the wide area
- Today: Overlay networks, 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)

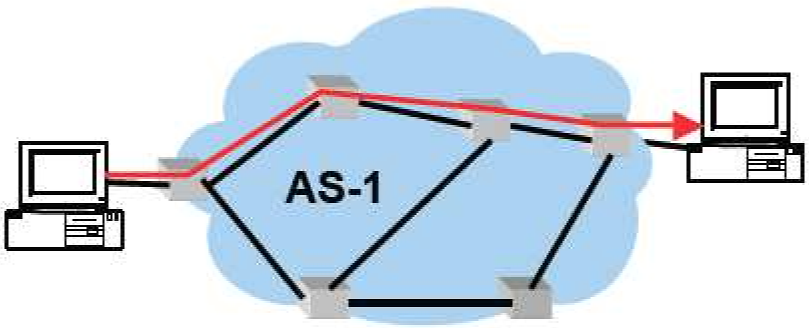
Overlay networks: Goals

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

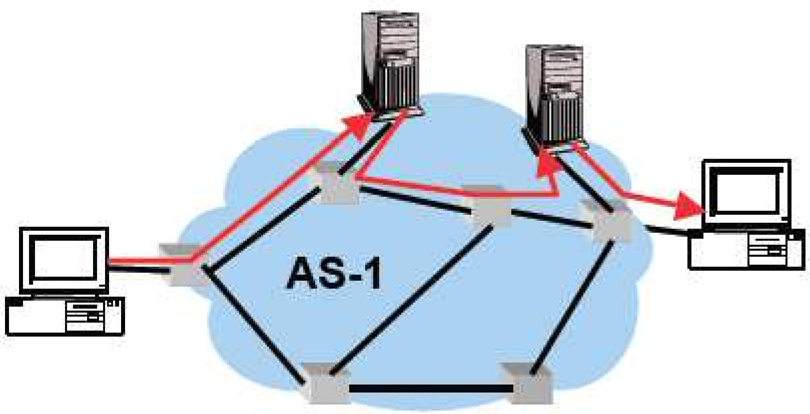
- Allow users to customize their service

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)



IP

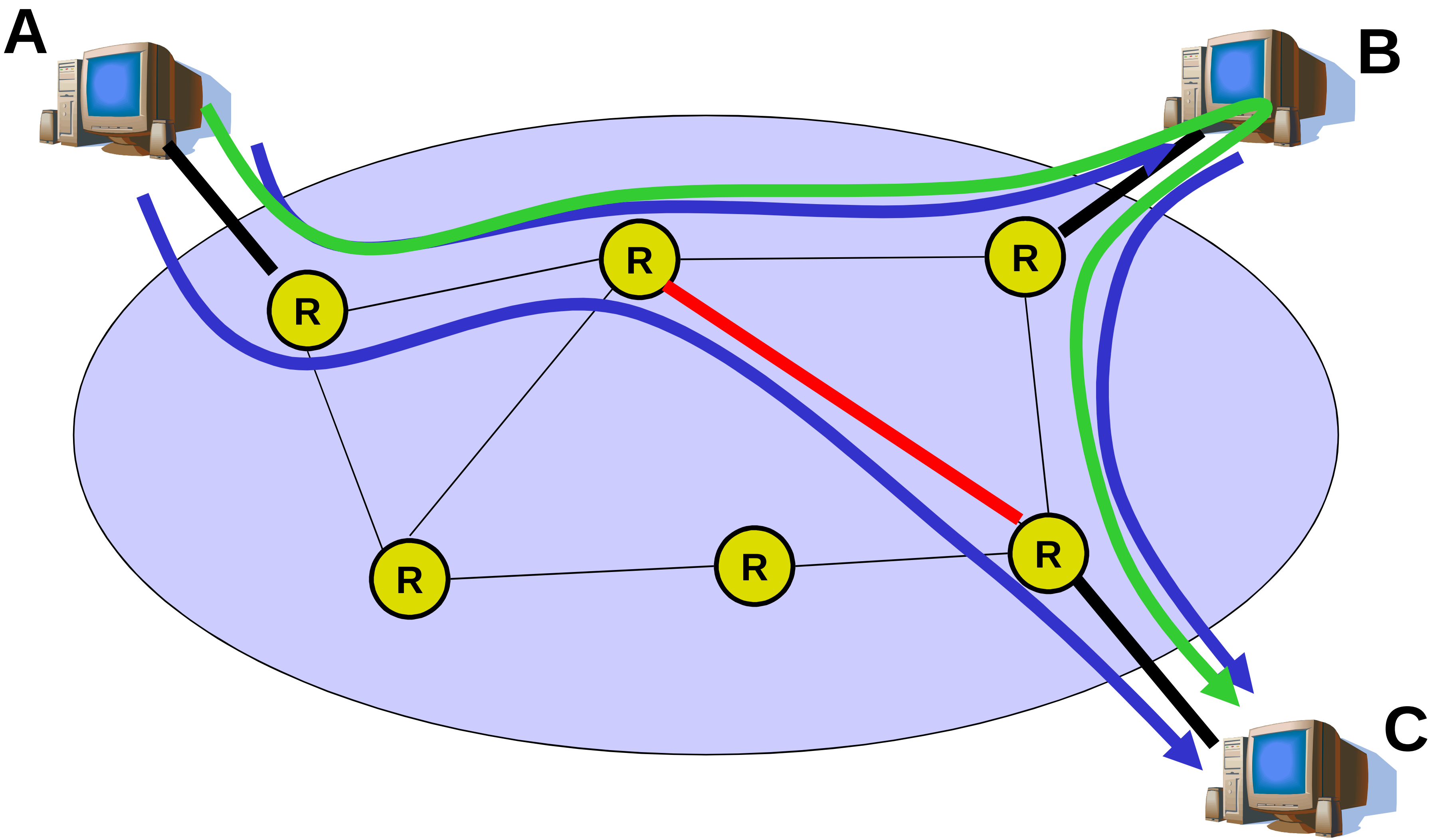


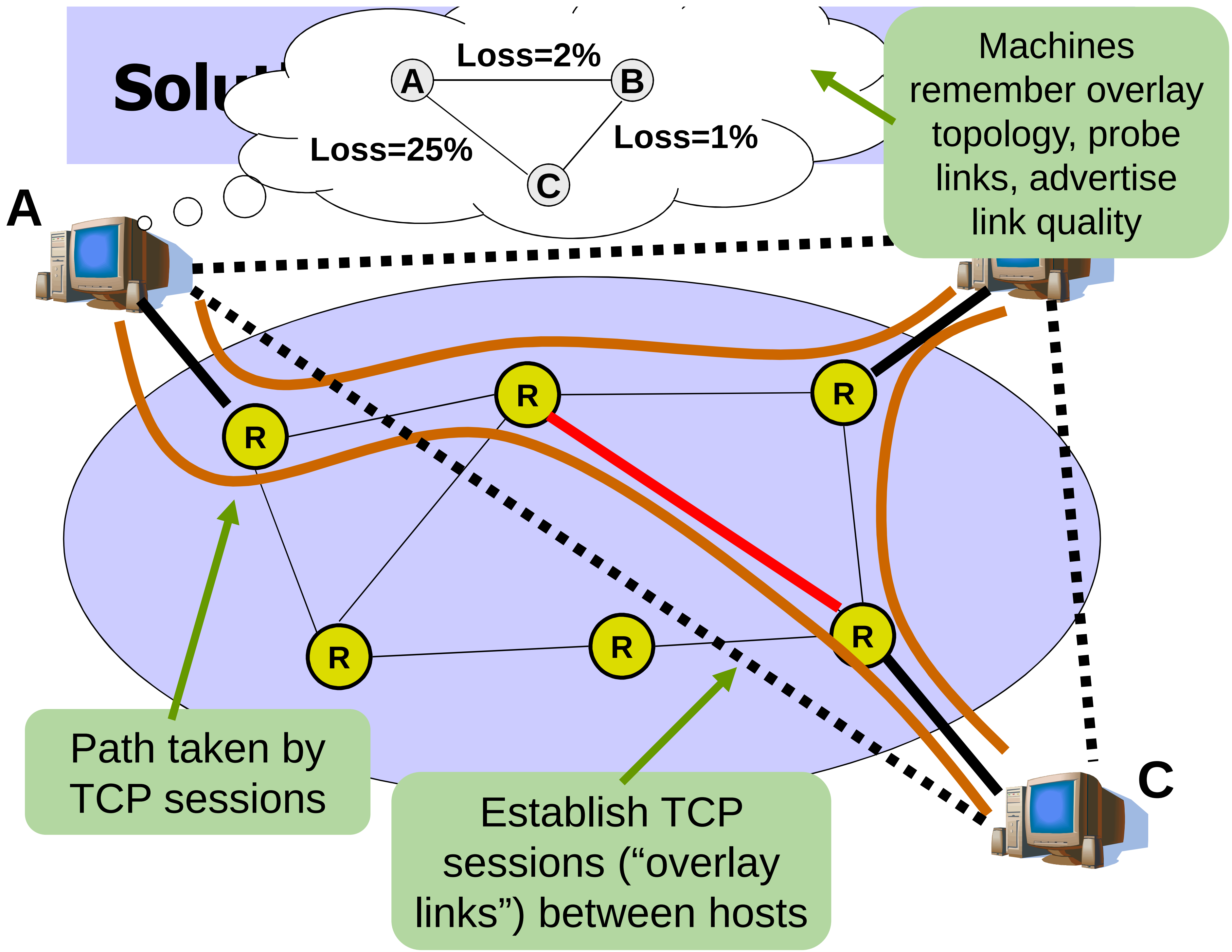
Overlay Network
(over IP)

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



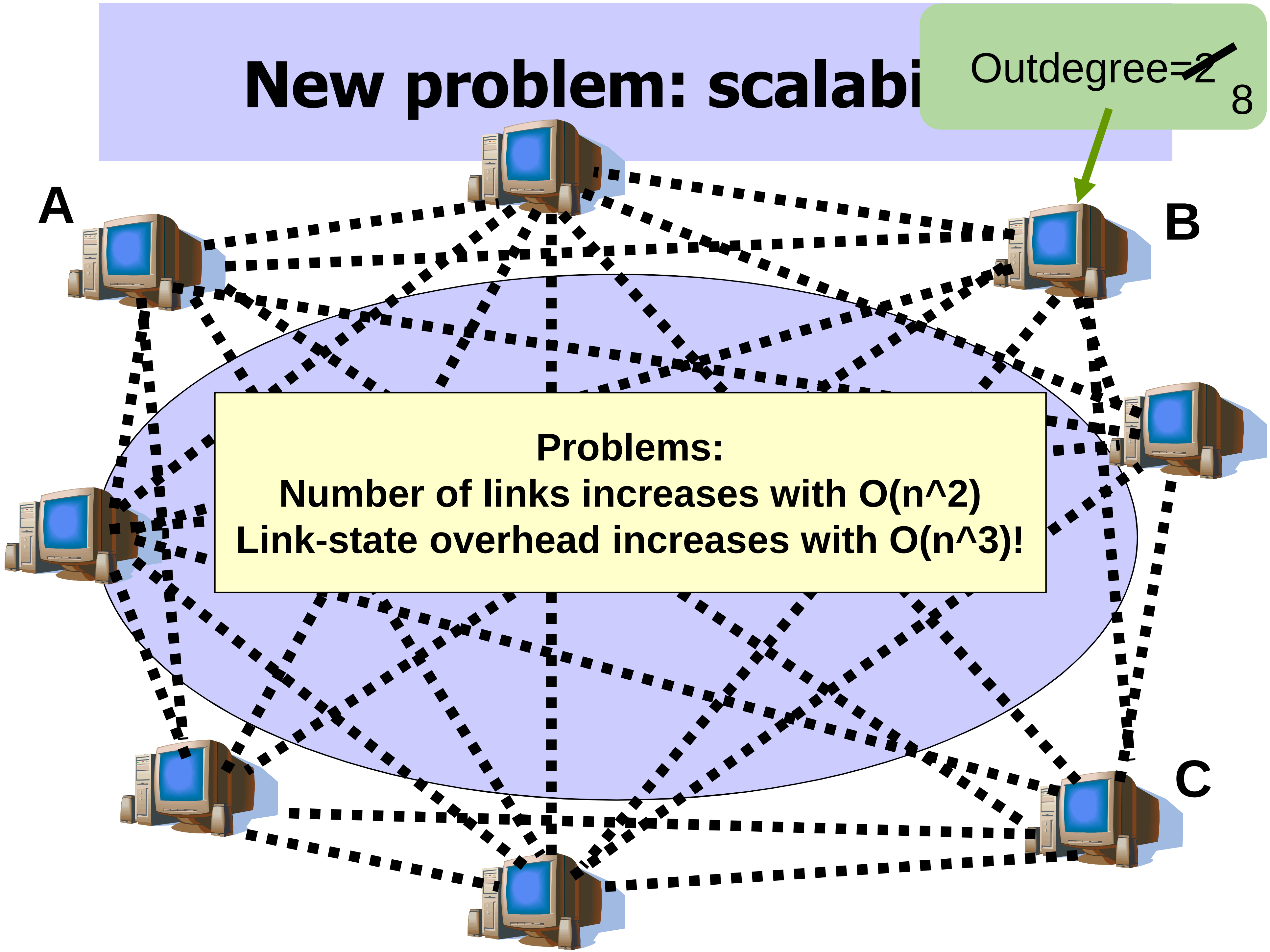

Solu
Loss=2%
Loss=25%
Loss=1%
A
B
C
Machines remember overlay topology, probe links, advertise link quality
R
Path taken by TCP sessions
Establish TCP sessions (“overlay links”) between hosts

Benefits of overlay networks

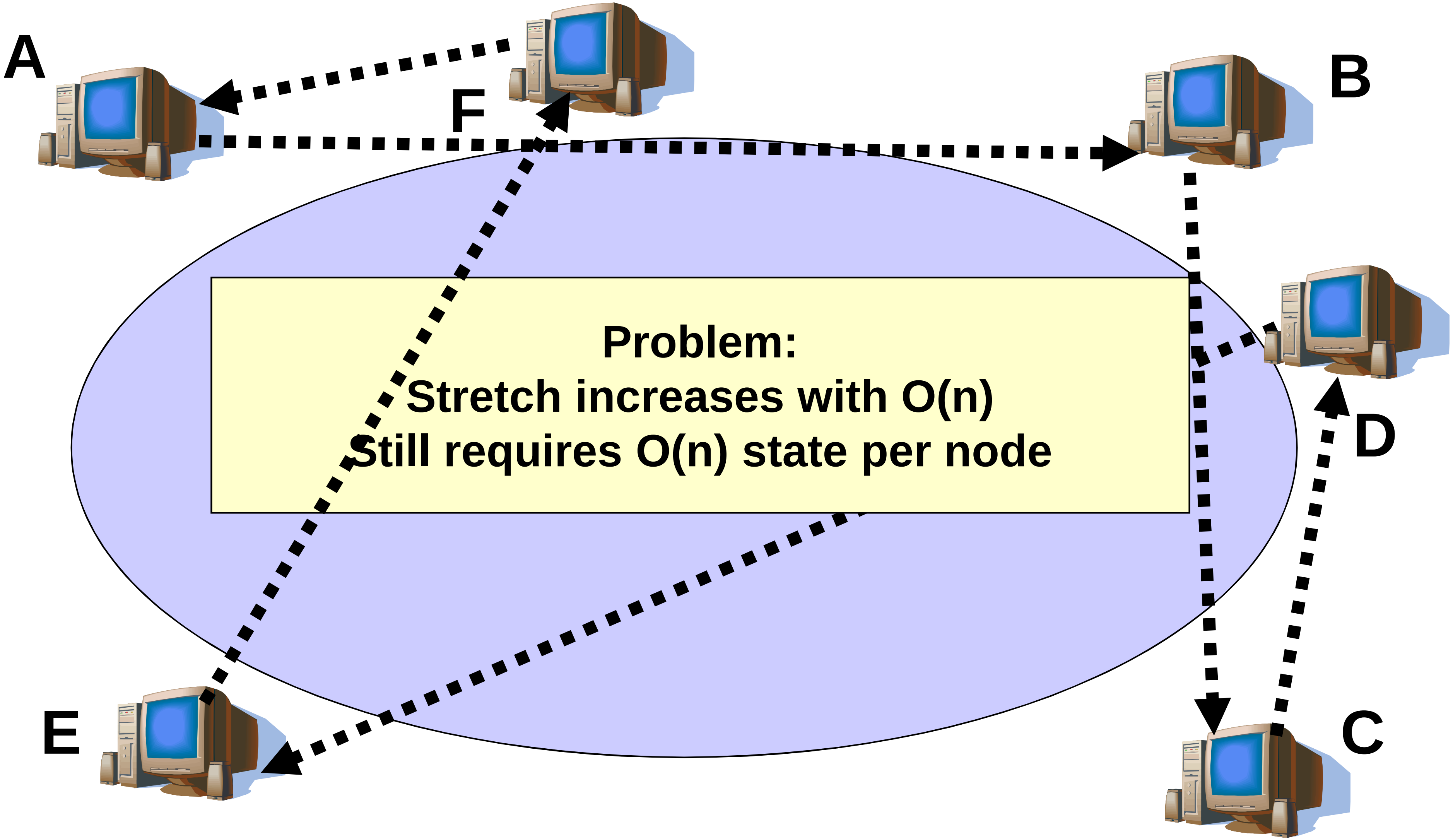
- Performance:
  - Difficult to provide QoS at network-layer due to deployment hurdles, lack of incentives, application-specific requirements
  - Overlays can probe faster, propagate more routes

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

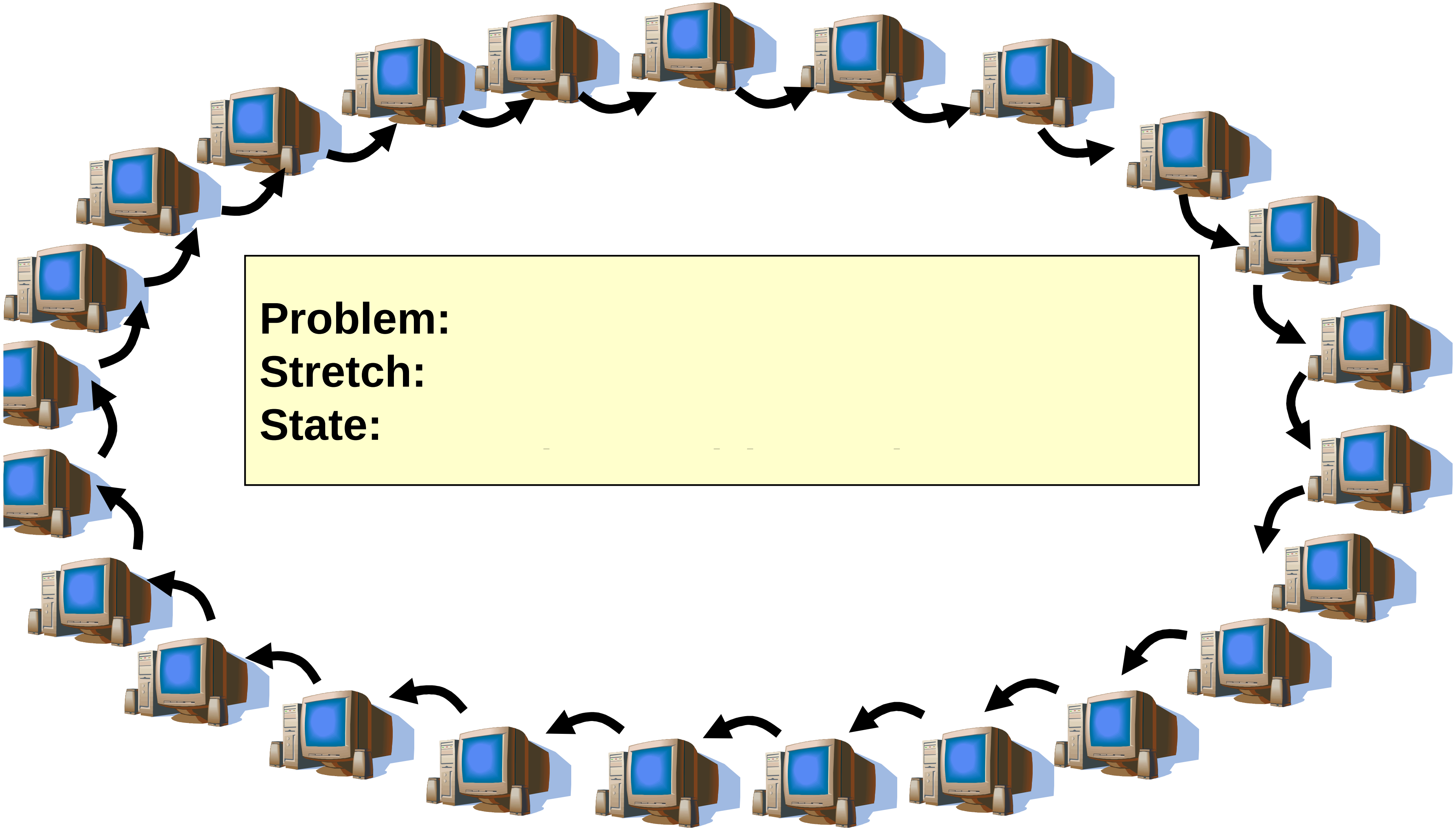
New problem: scalabi



Alternative:
replace full-mesh with logical ring



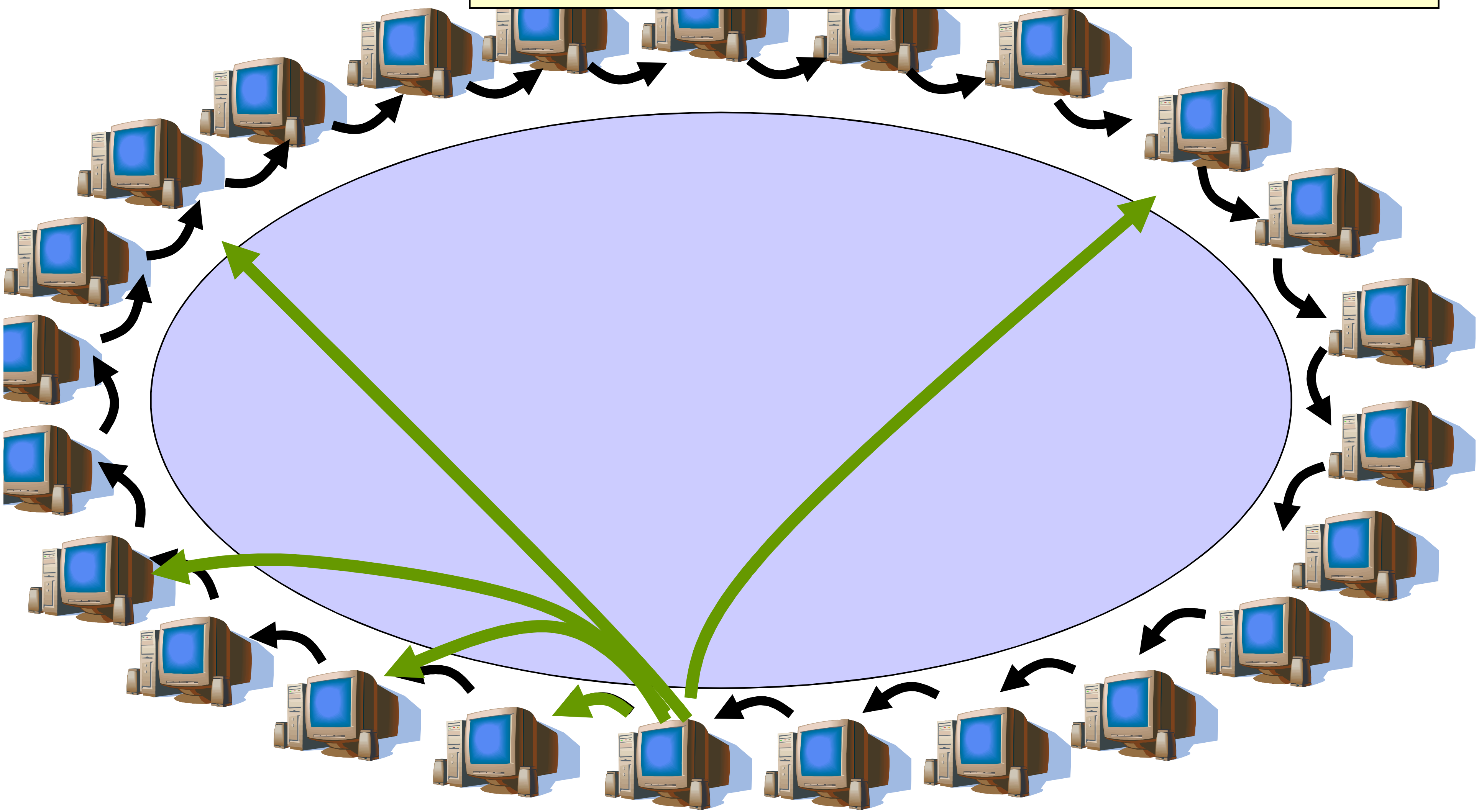
Alternative: replace full-mesh with ring



Problem:
Stretch:
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
- General concept: distributed data structures

Different kinds of DHTs

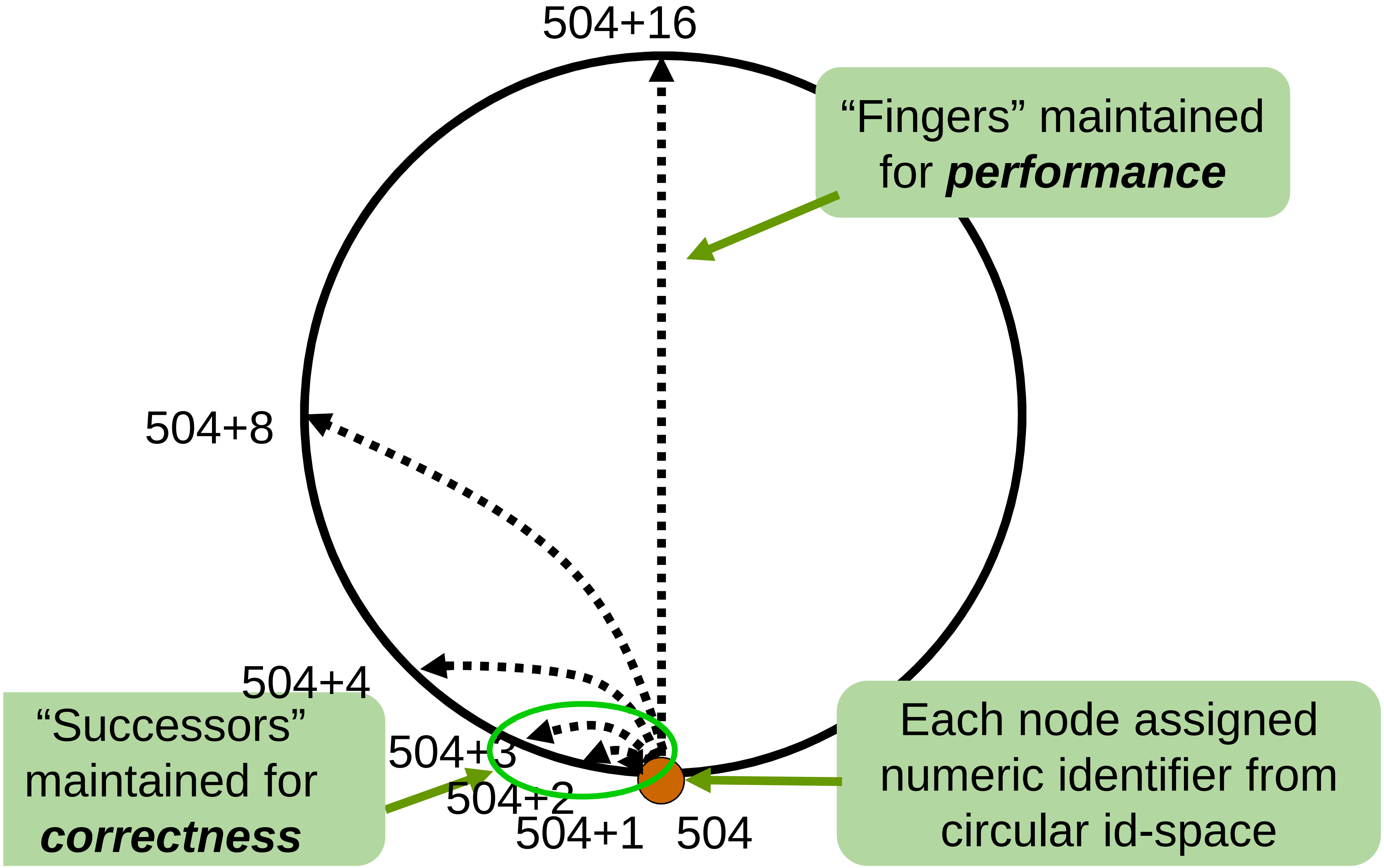
- Different topologies give different bounds on stretch (delay penalty)/state, different stability under churn, etc. Examples:

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

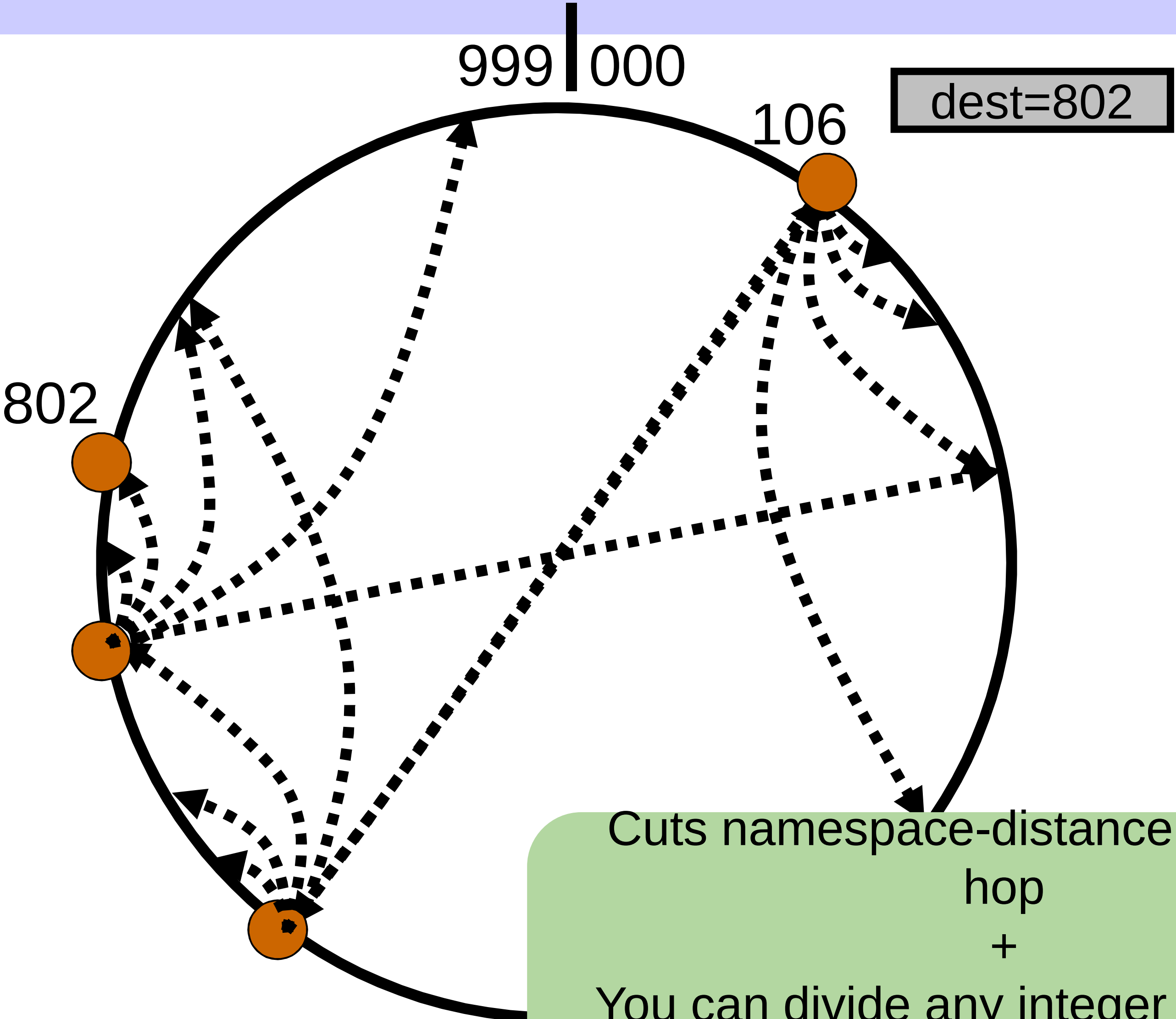
- Pastry
  - Pointers to nodes sharing varying prefix lengths with local node, plus pointer to immediate successor
  - Forward to numerically closest node

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

The Chord DHT



Chord Example: Forwarding a lookup



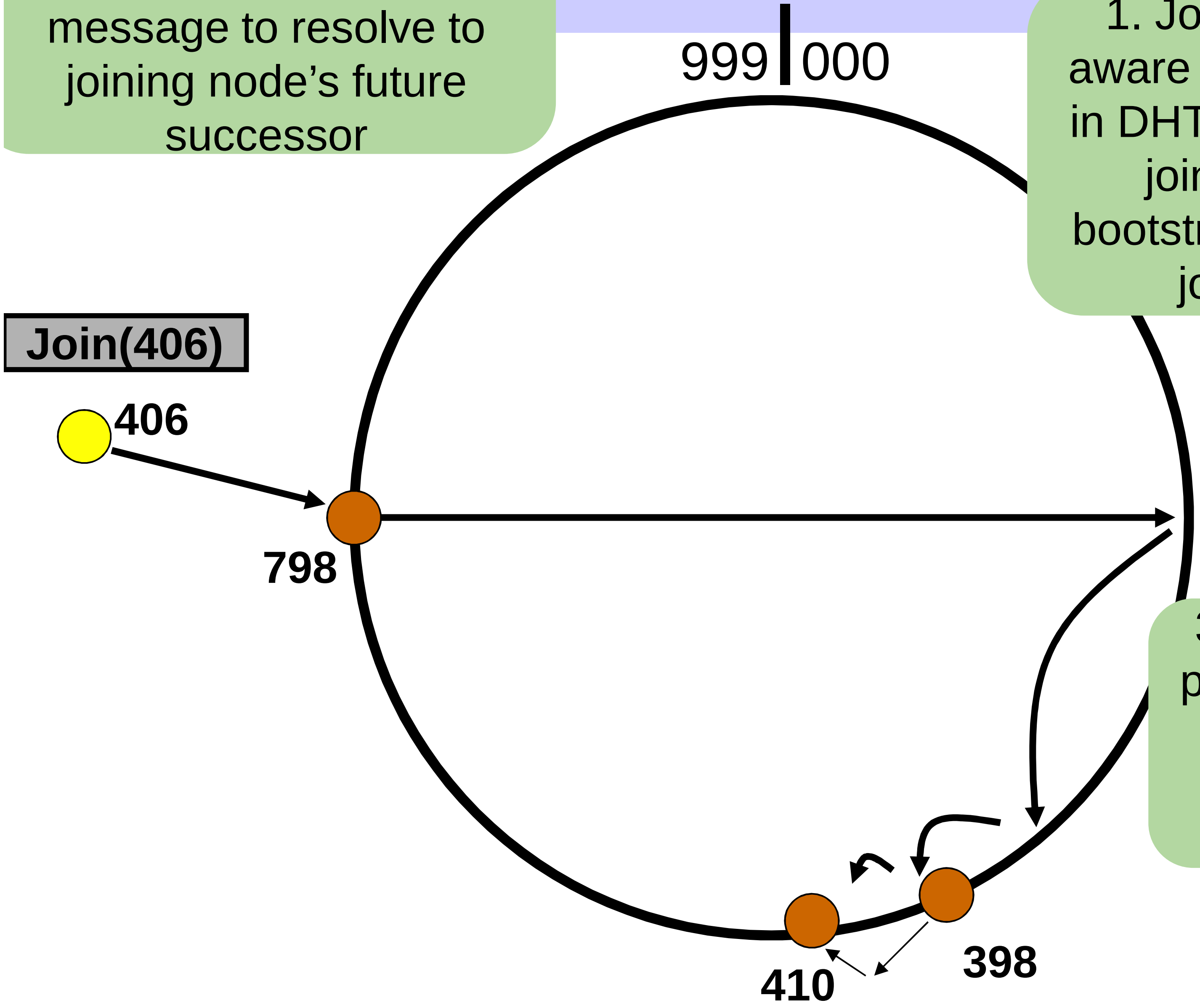
Cuts namespace-distance in half per hop
+
You can divide any integer N in half at most log(N) times
= logarithmic stretch

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

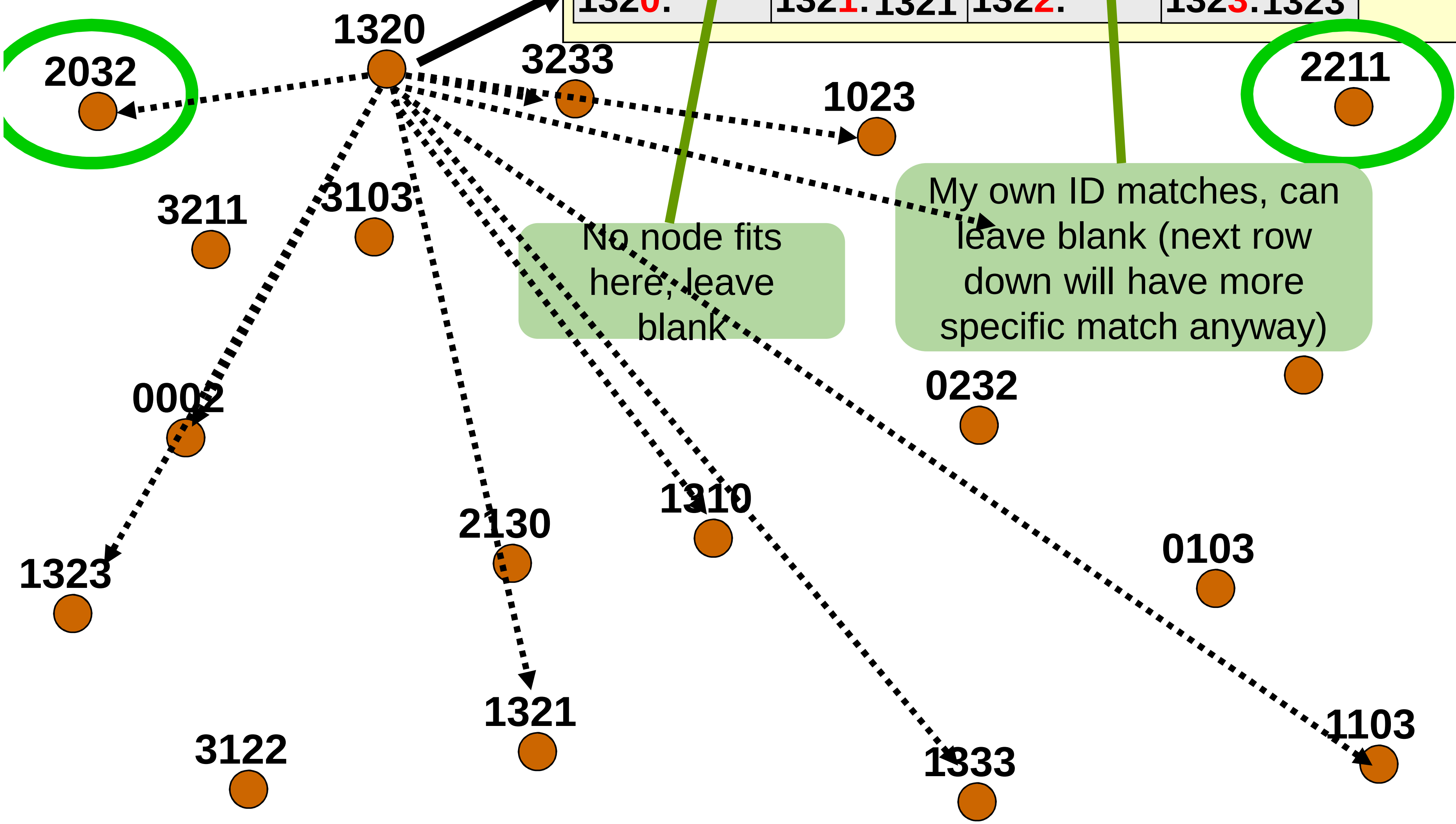
Chord: Discussion

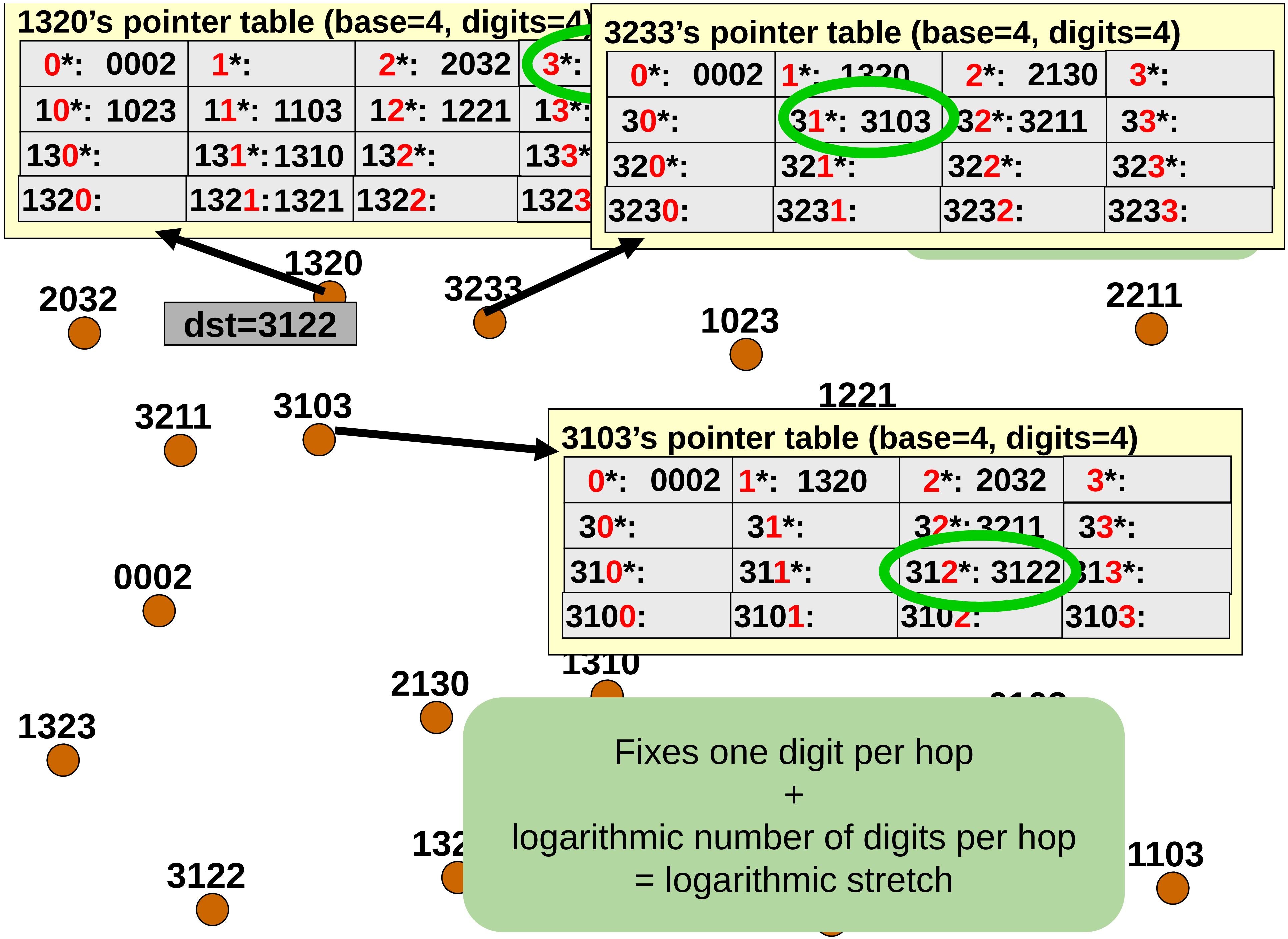
- Query can be implemented
  - Iteratively
  - Recursively
- Performance: routing in the overlay network can be more expensive than routing in the underlying network
  - Because usually **no** correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though 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

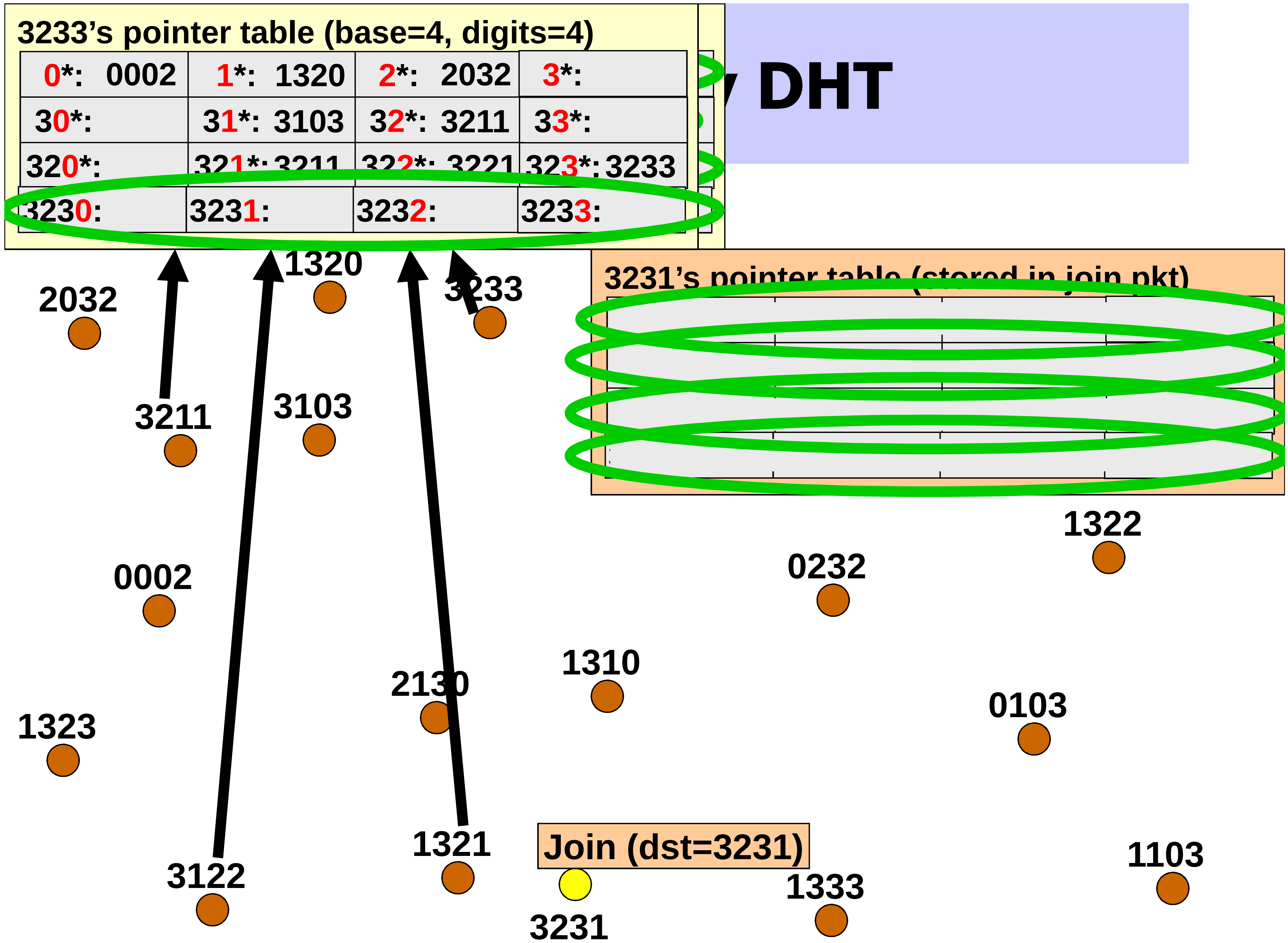
1320's pointer table (base=4, digits=4)

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




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:
1320
3233
2032
dst=3122
1023
2211
1221
3211
3103
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:
0002
1310
2130
1323
3122
1103
Fixes one digit per hop
+
logarithmic number of digits per hop
= logarithmic stretch

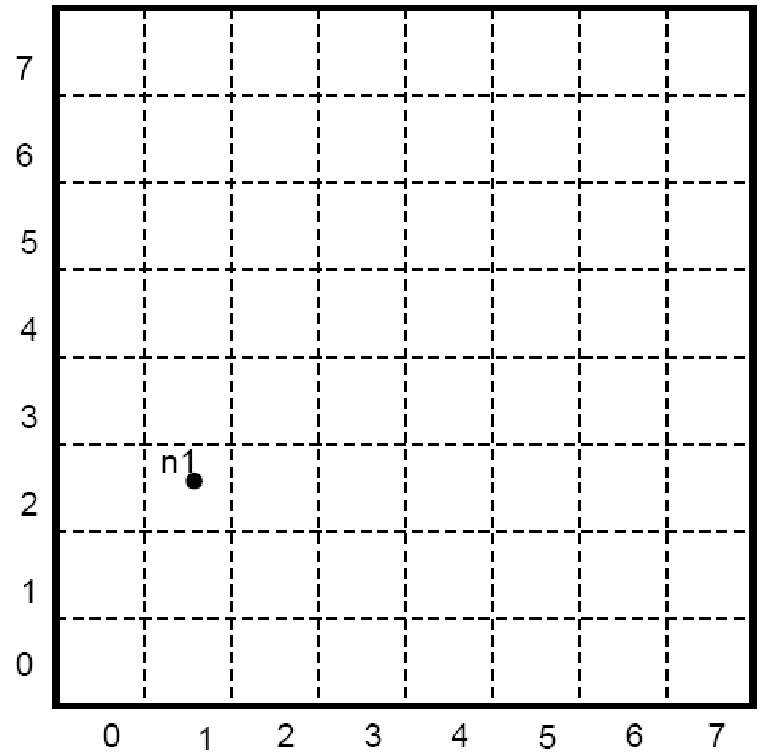

DHT
3233's pointer table (base=4, digits=4)
0*: 0002
1*: 1320
2*: 2032
3*:
30*:
31*: 3103
32*: 3211
33*:
320*:
321*: 3211
322*: 3221
323*: 3233
3230:
3231:
3232:
3233:
3231's pointer table (stored in join pkt)
Join (dst=3231)
2032
3211
1320
3233
3103
0002
1323
3122
2130
1321
3231
1310
0232
1322
0103
1333
1103

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

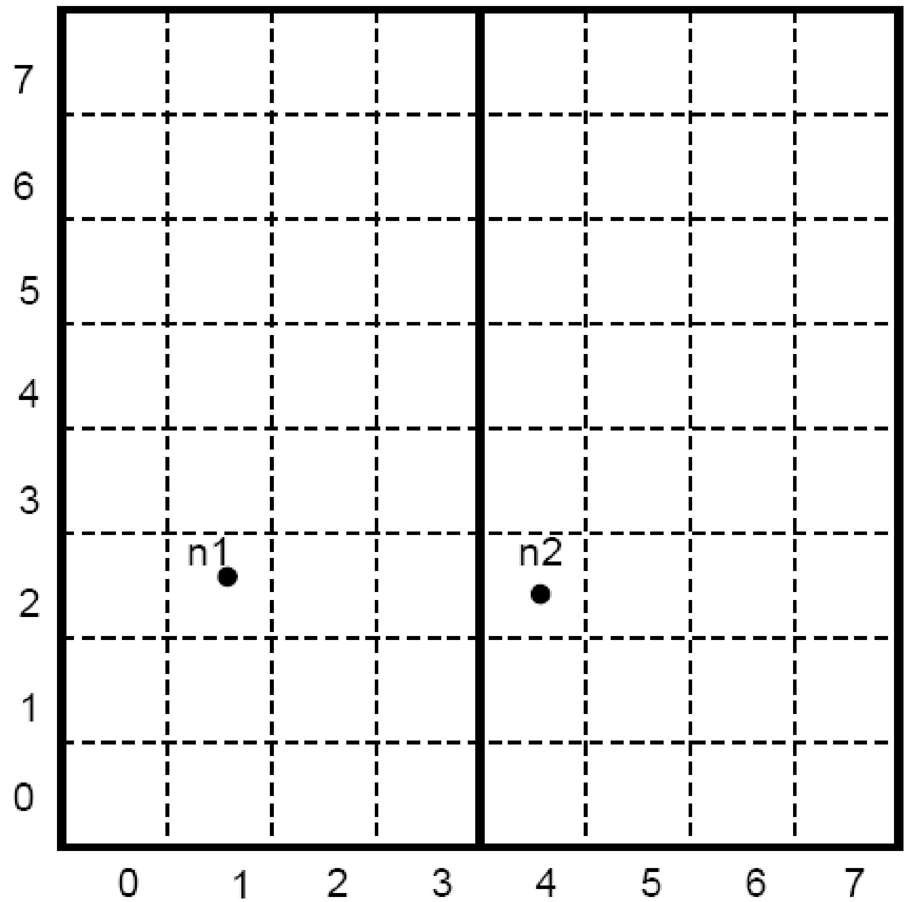
CAN Example: Two dimensional space

- 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



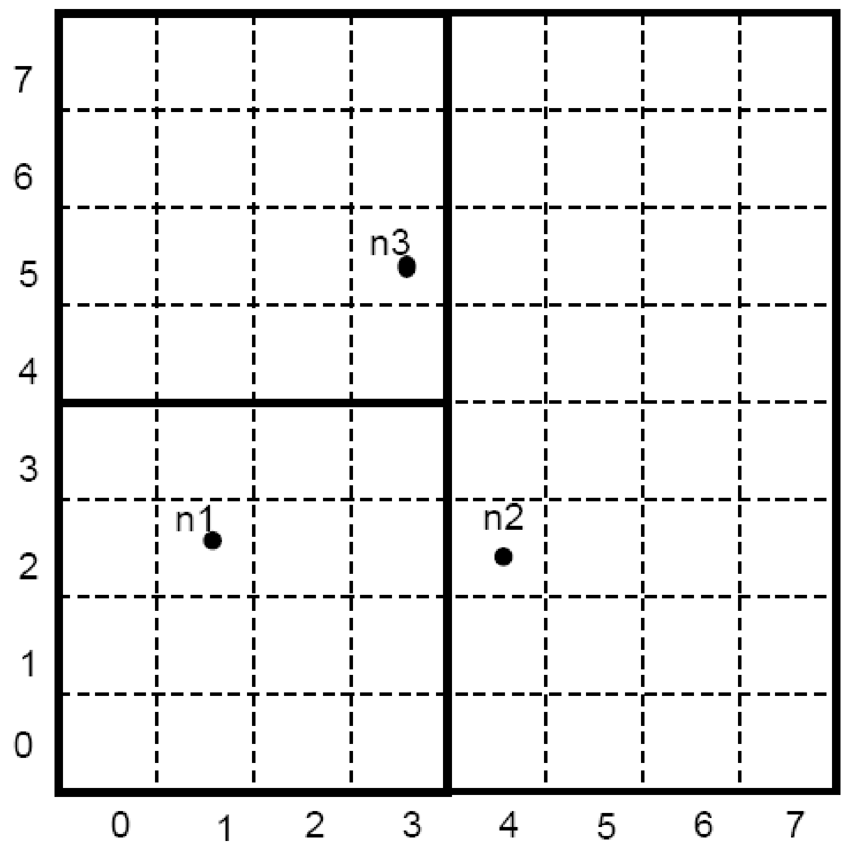
CAN Example: Two dimensional space

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



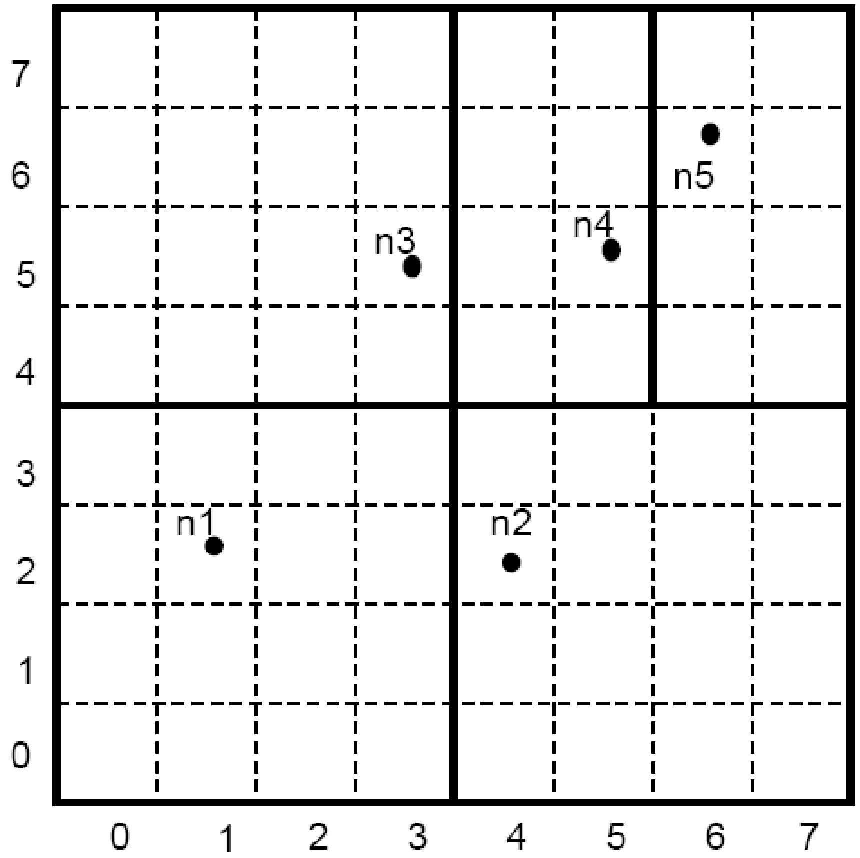
CAN Example:
Two dimensional space

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



CAN Example: Two dimensional space

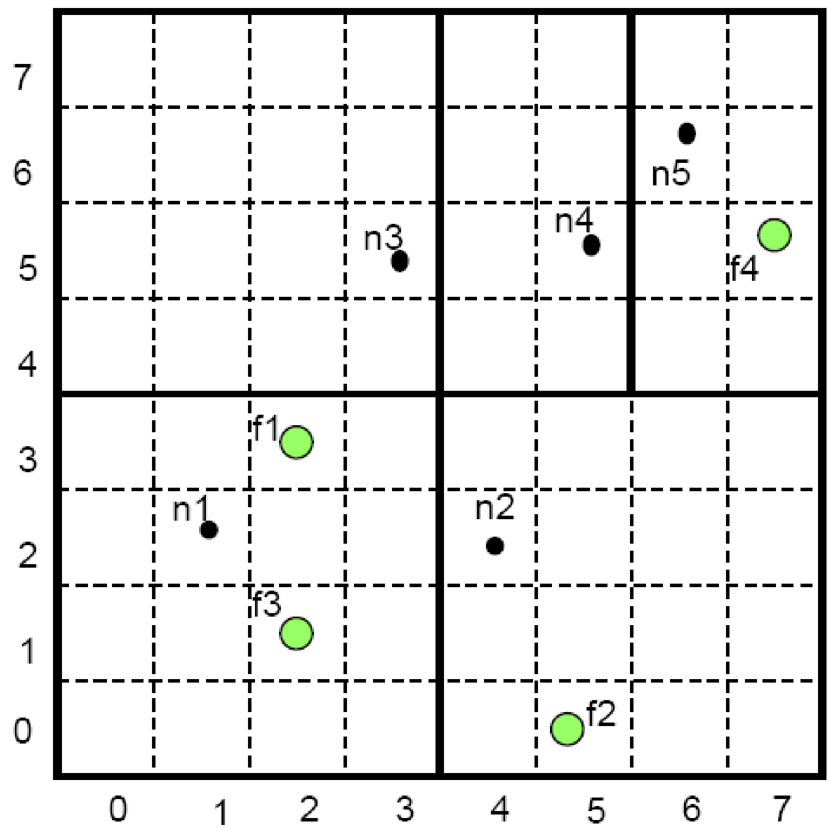
- Nodes n4:(5,5) and n5:(6,6) join



CAN Example:
Two dimensional space

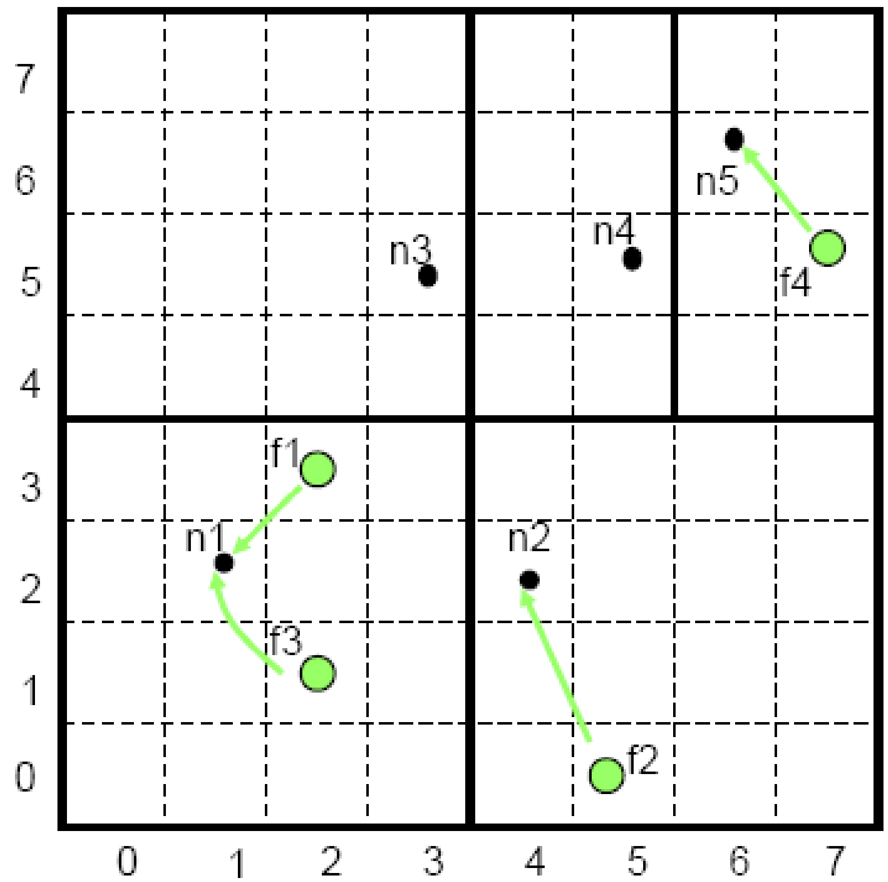
- Nodes:
  - n1:(1,2)
  - n2:(4,2)
  - n3:(3,5)
  - n4:(5,5)
  - n5:(6,6)

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



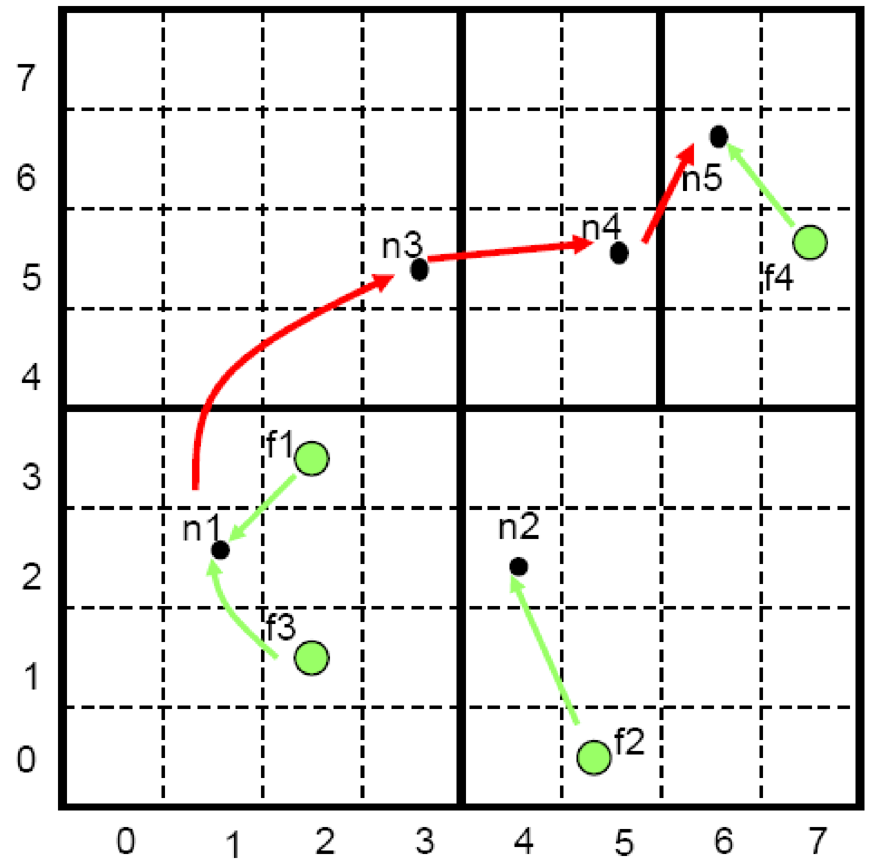
CAN Example:
Two dimensional space

- 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



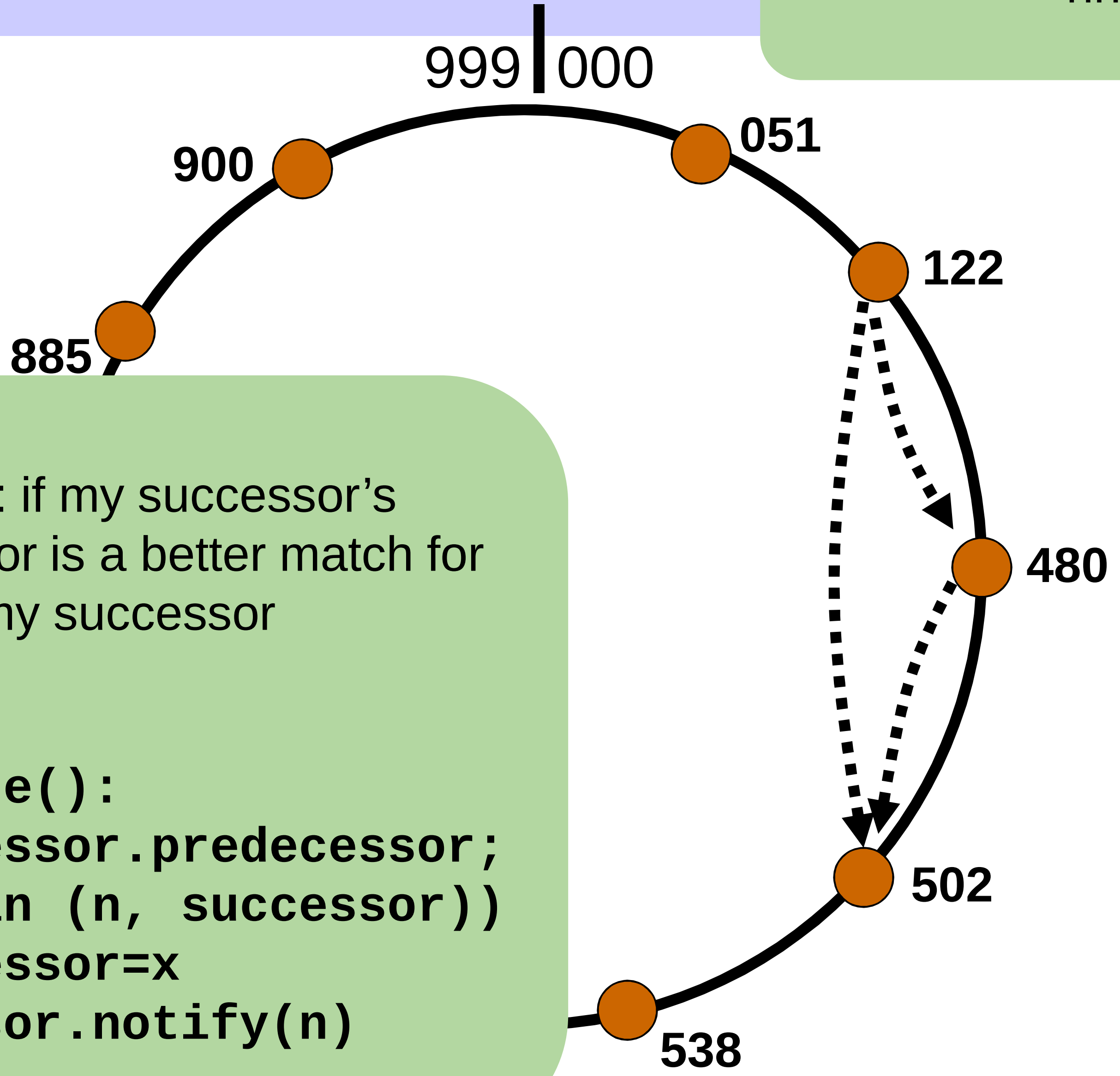
Preserving consistency

- 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

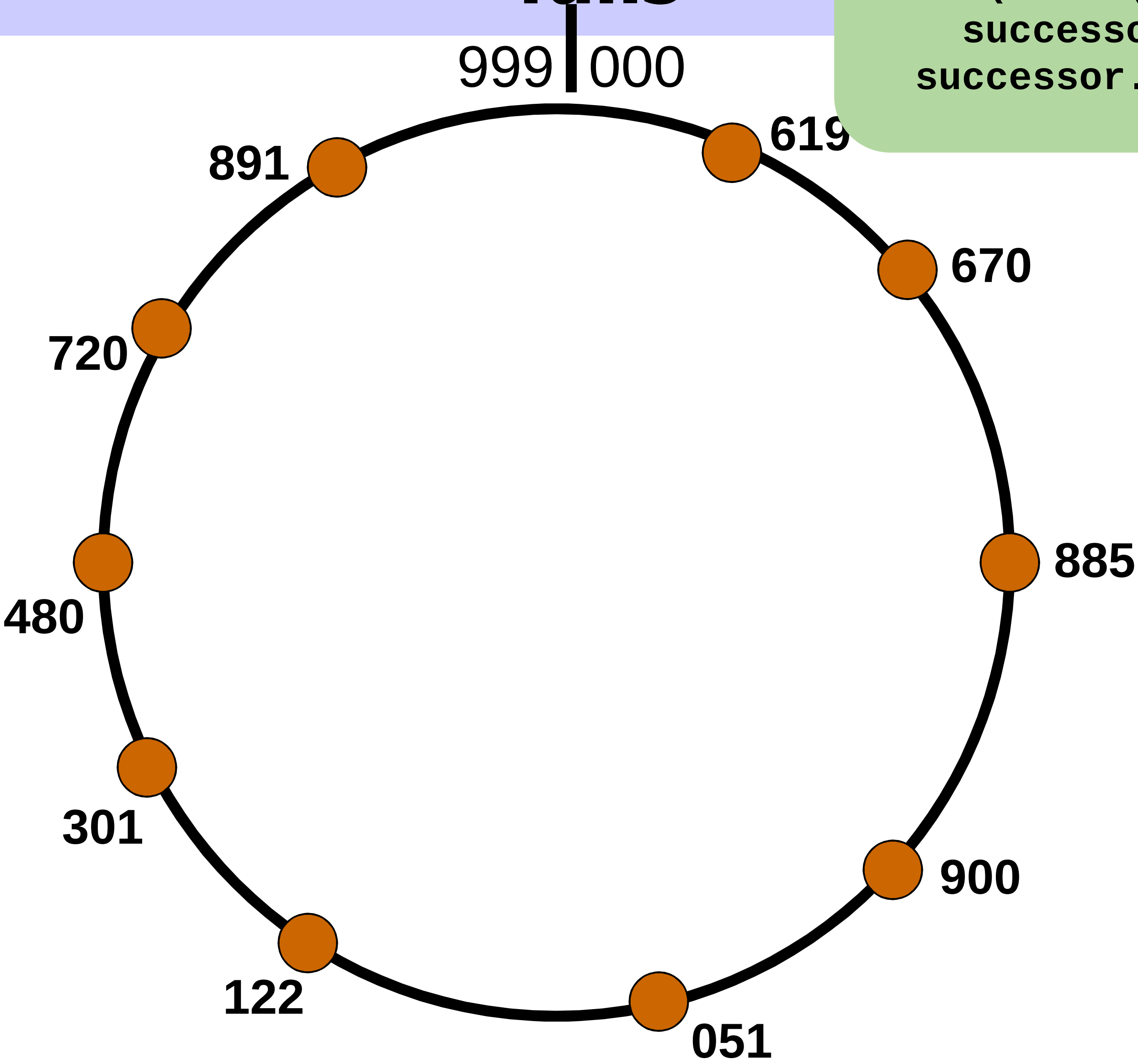


Check: if my successor's predecessor is a better match for my successor

```
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
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

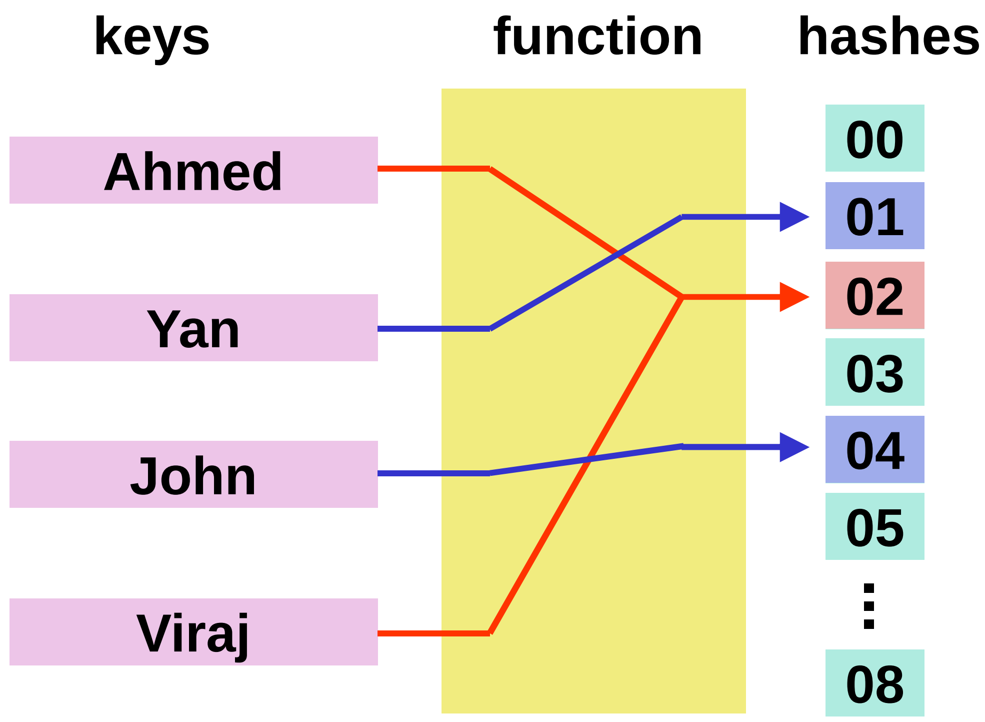
Security issues

- 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

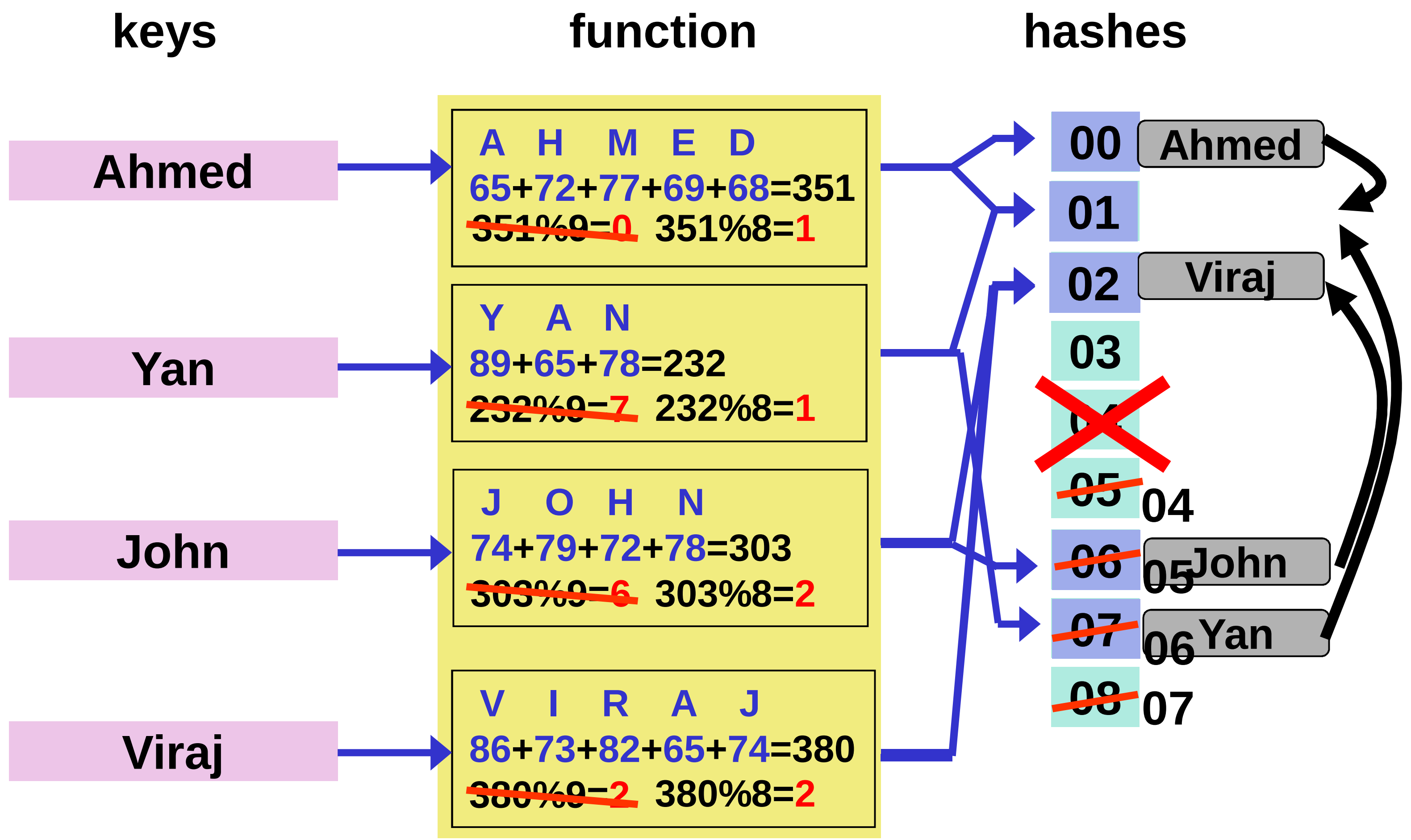
Hashing in networked software

- 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

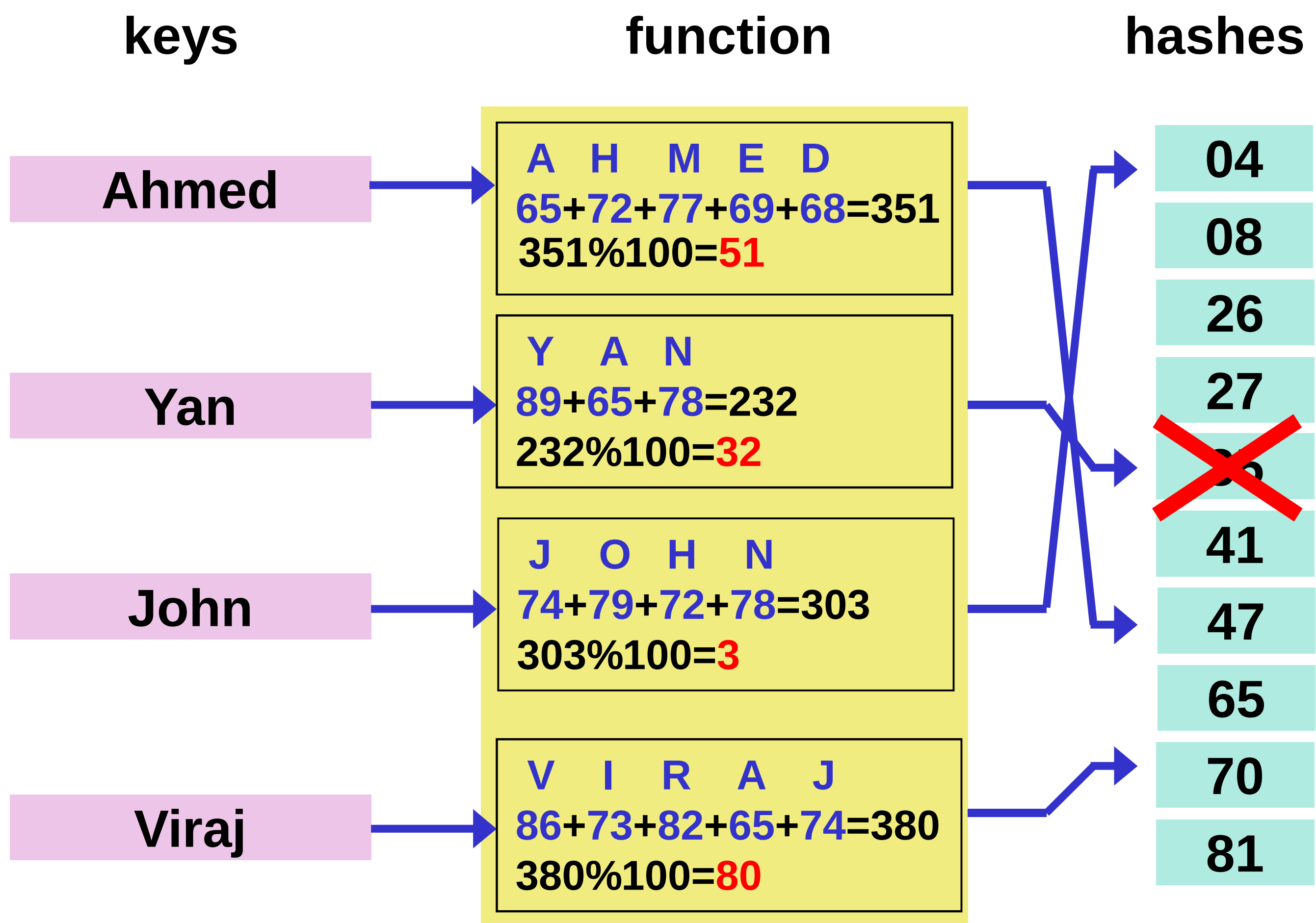


- Example: Sum ASCII digits, mod number of bins
- Problem: ____________________

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

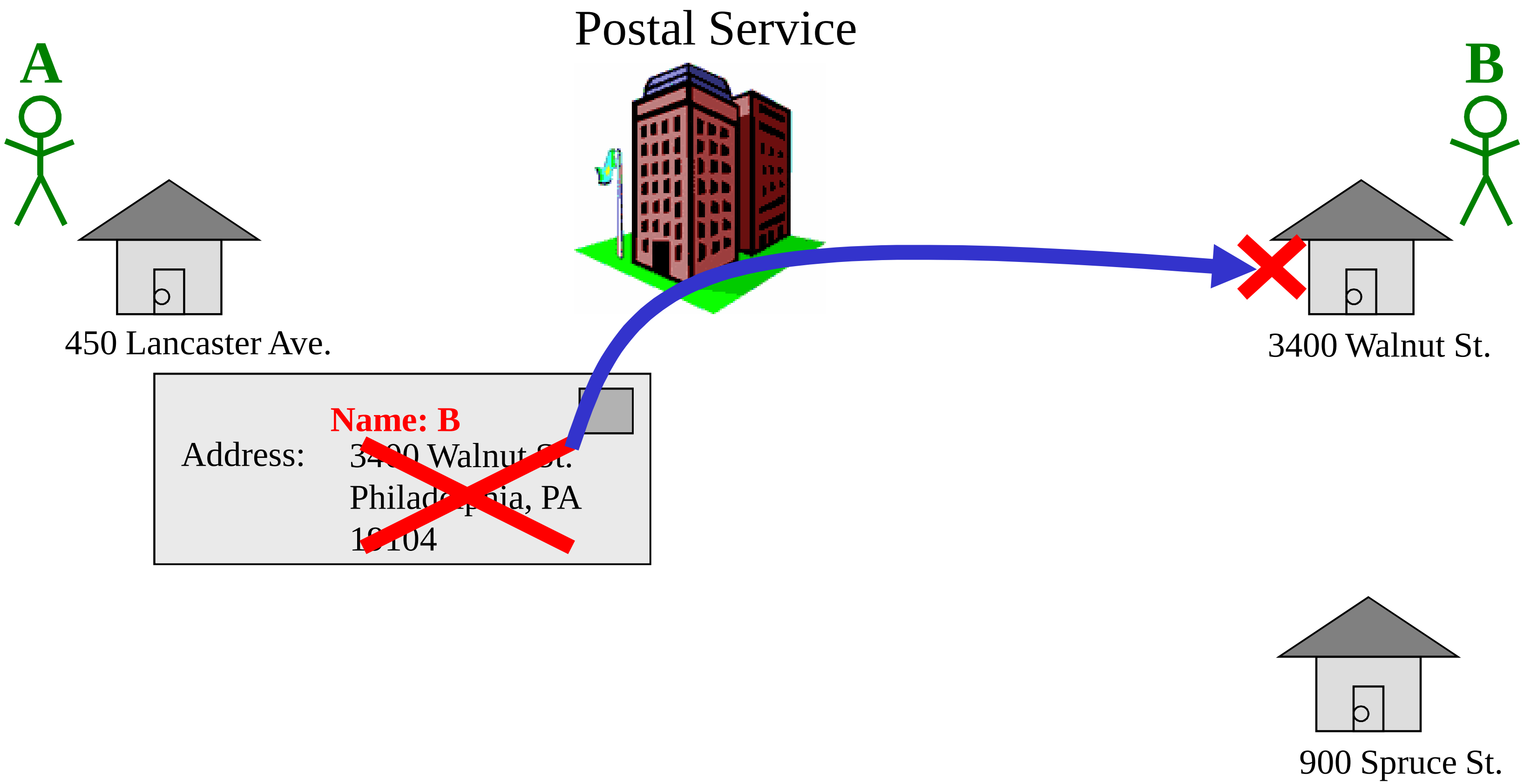
Solution: Consistent Hashing



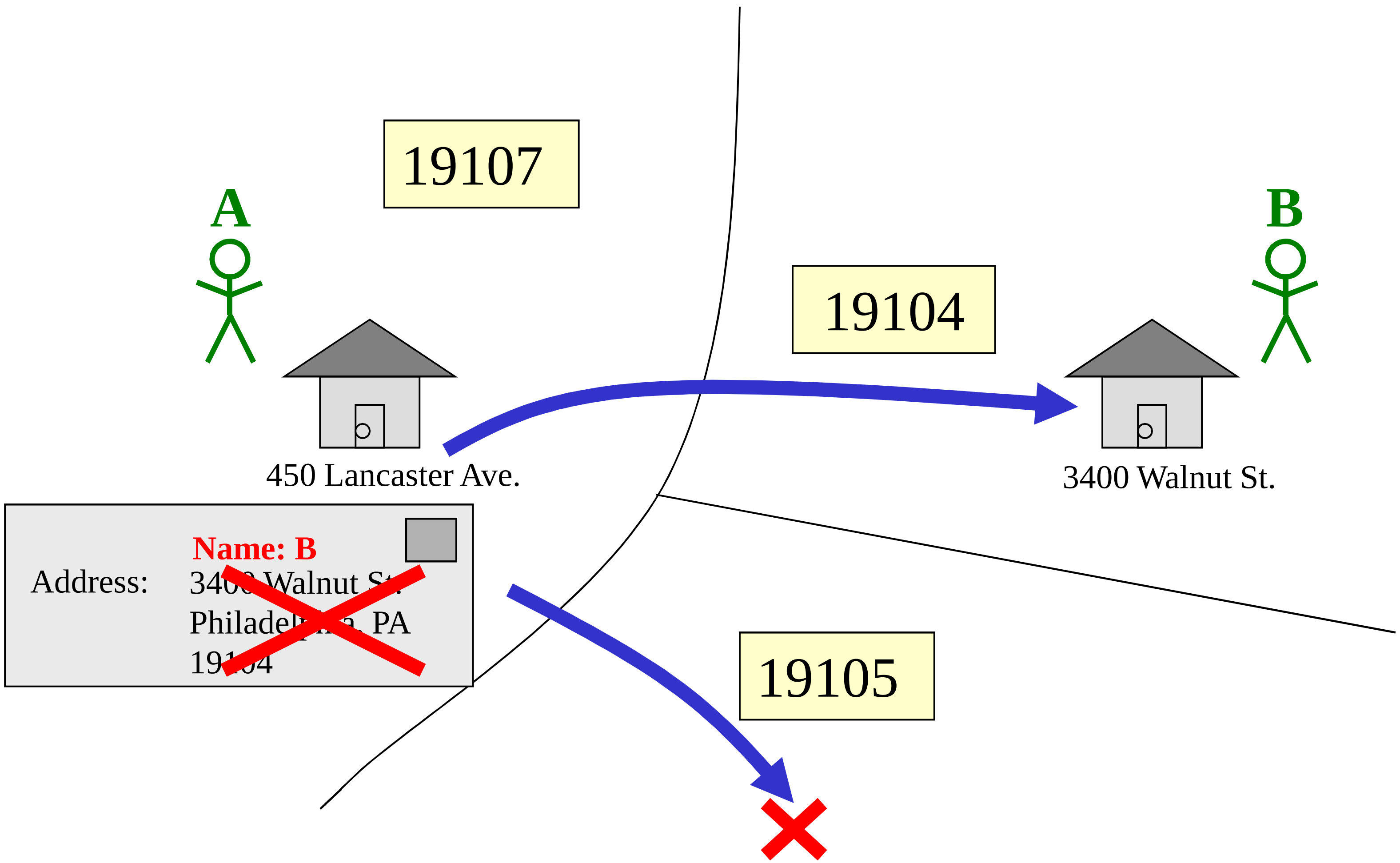
- Slots have IDs selected randomly from [0,100]
- Hash keys onto same space, map key to closest bin
- Less churn on failure → more stable system

Network layer DHTs

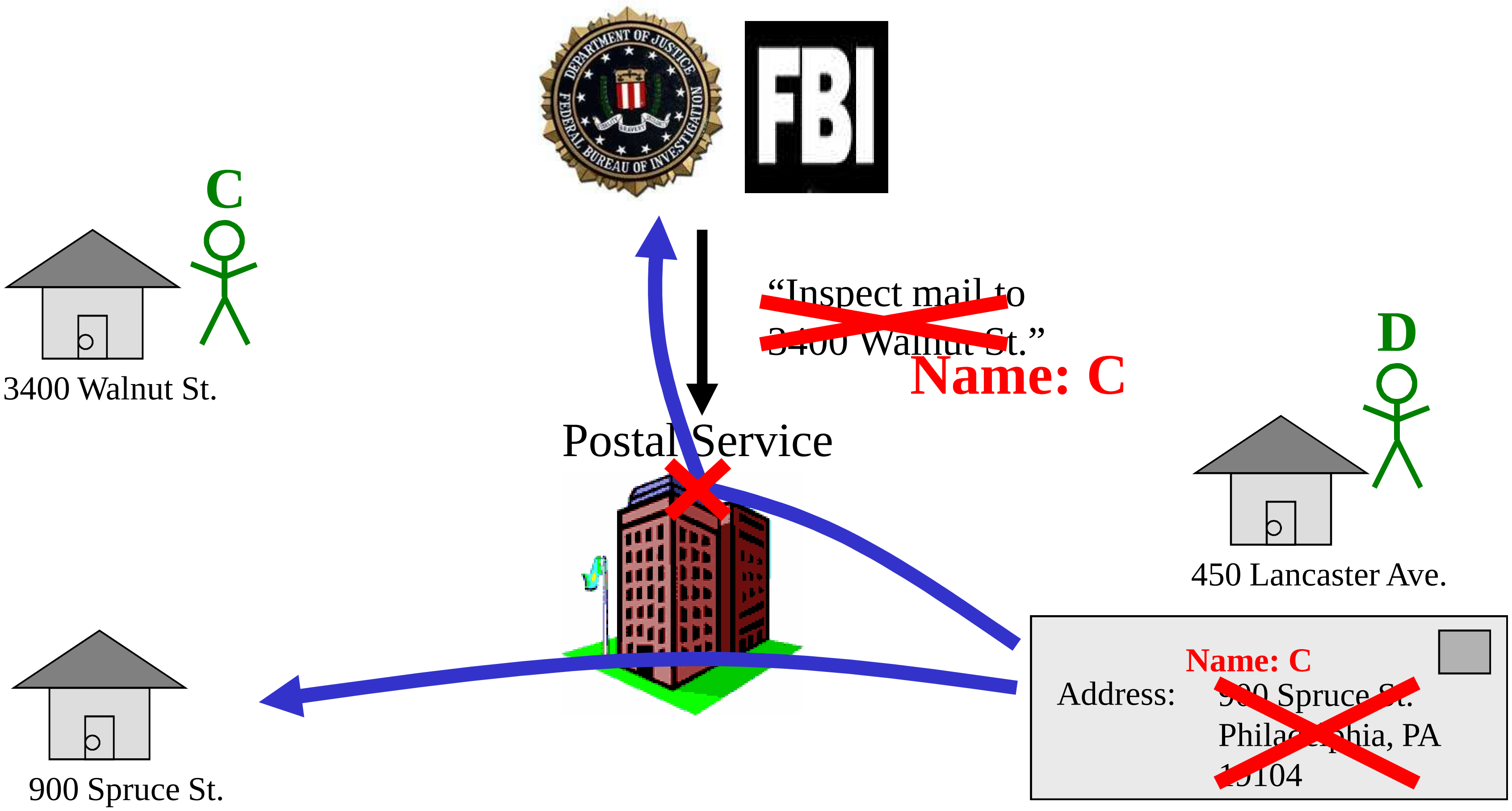
Scenario: Sending a Letter



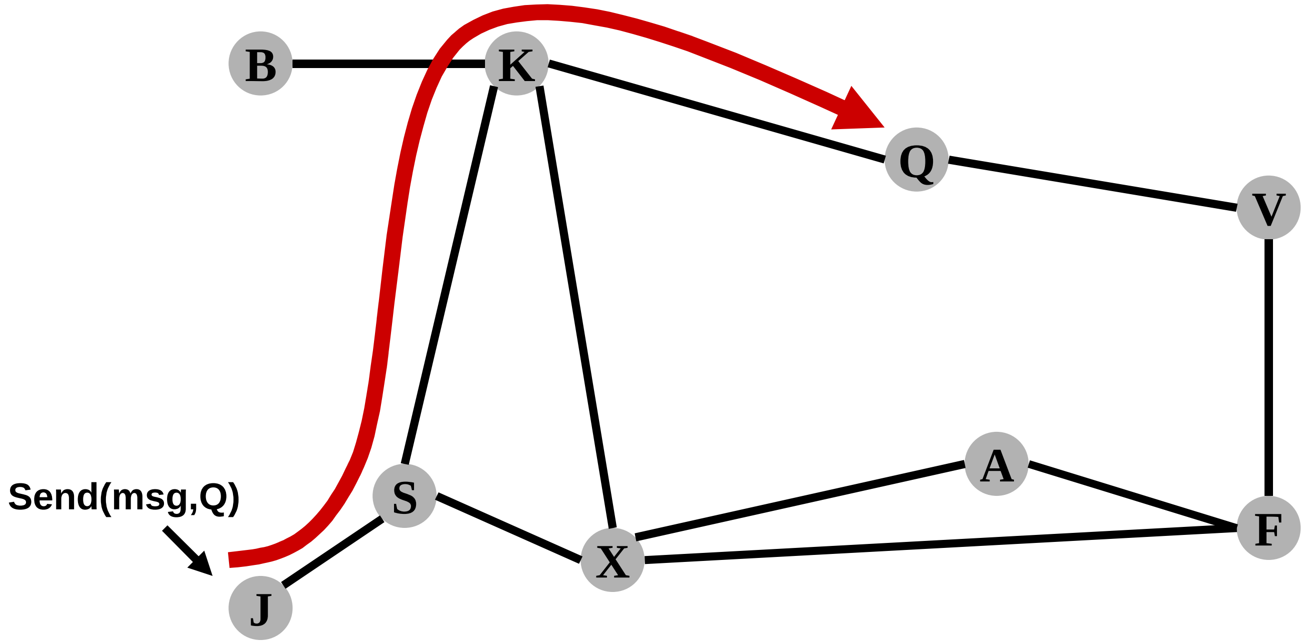
Scenario: Address Allocation



Scenario: Access Control

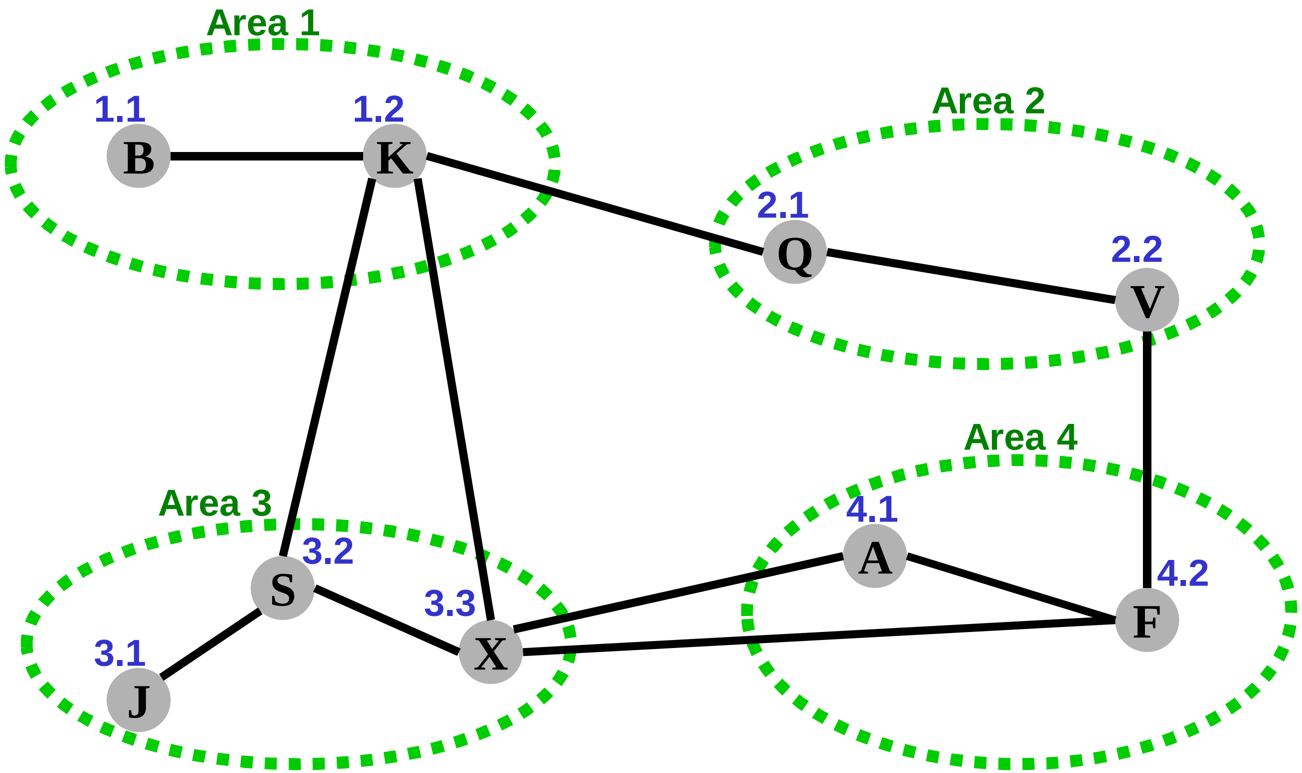


How Routing Works Today



- Each node has an identity
- Goal: find path to destination

Scaling Requires Aggregation



- Pick addresses that depend on location
- Aggregation provides excellent scaling properties
- Key is topology-dependent addressing!

Topology-Dependent Addresses Aren't Always Possible

- Networks can't use topology-dependent addresses because topology changes so rapidly

- Decades-long search for scalable routing algorithms for ad hoc networks

Topology-Dependent Addresses Aren't Always Desirable

- Using topology-based addresses in the Internet complicates access controls, mobility, and multihoming

- Would like to embed persistent identities into network-layer addresses

Can We Scale without Topology-Dependent Addresses?

- Is it possible to scale without aggregation?
- Distributed Hash Tables don't solve this problem

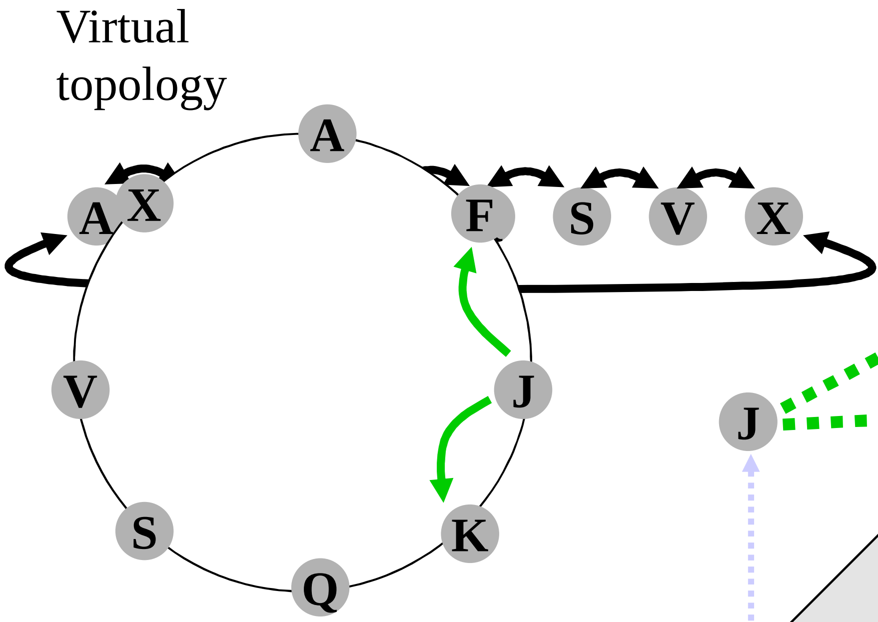
This Talk

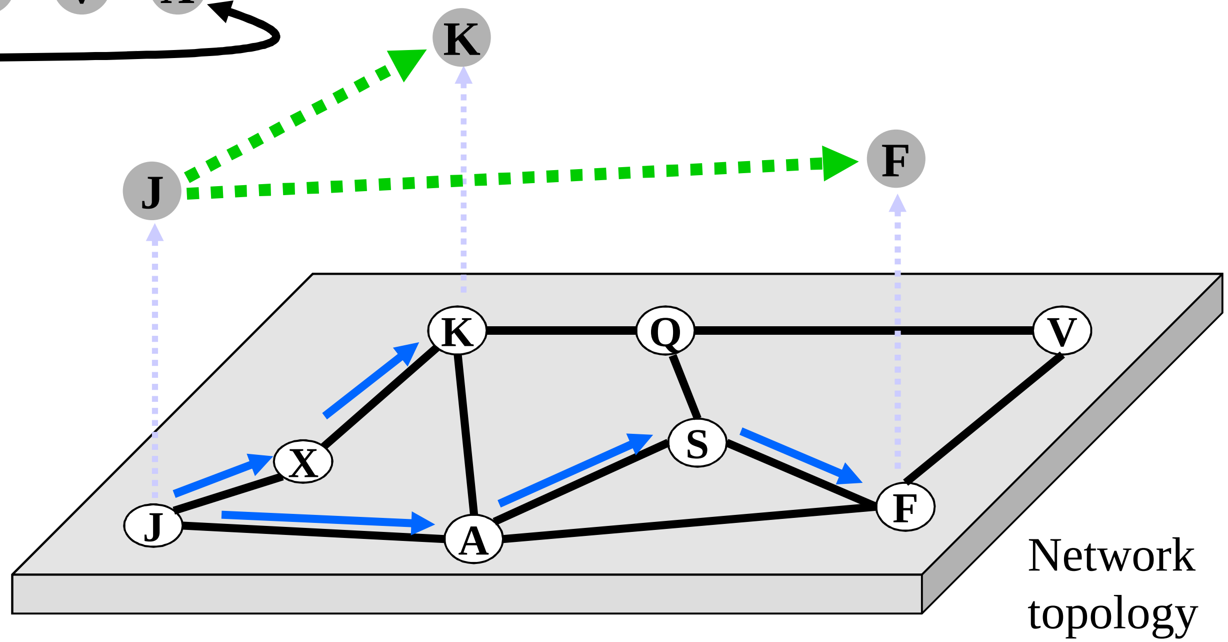
- Will describe how to route scalably on flat identifiers that applies to both:
- Wireless networks:
  - Challenge is dynamics
- Wired networks:
  - Challenge is scale, policies, and dynamics

Outline

- Routing on an abstract graph
  - What state is maintained
  - How to route using that state
  - How to correctly maintain state
- Wireless sensornet implementation
- Evaluation for Internet routing
- Conclusions

State maintained at each node

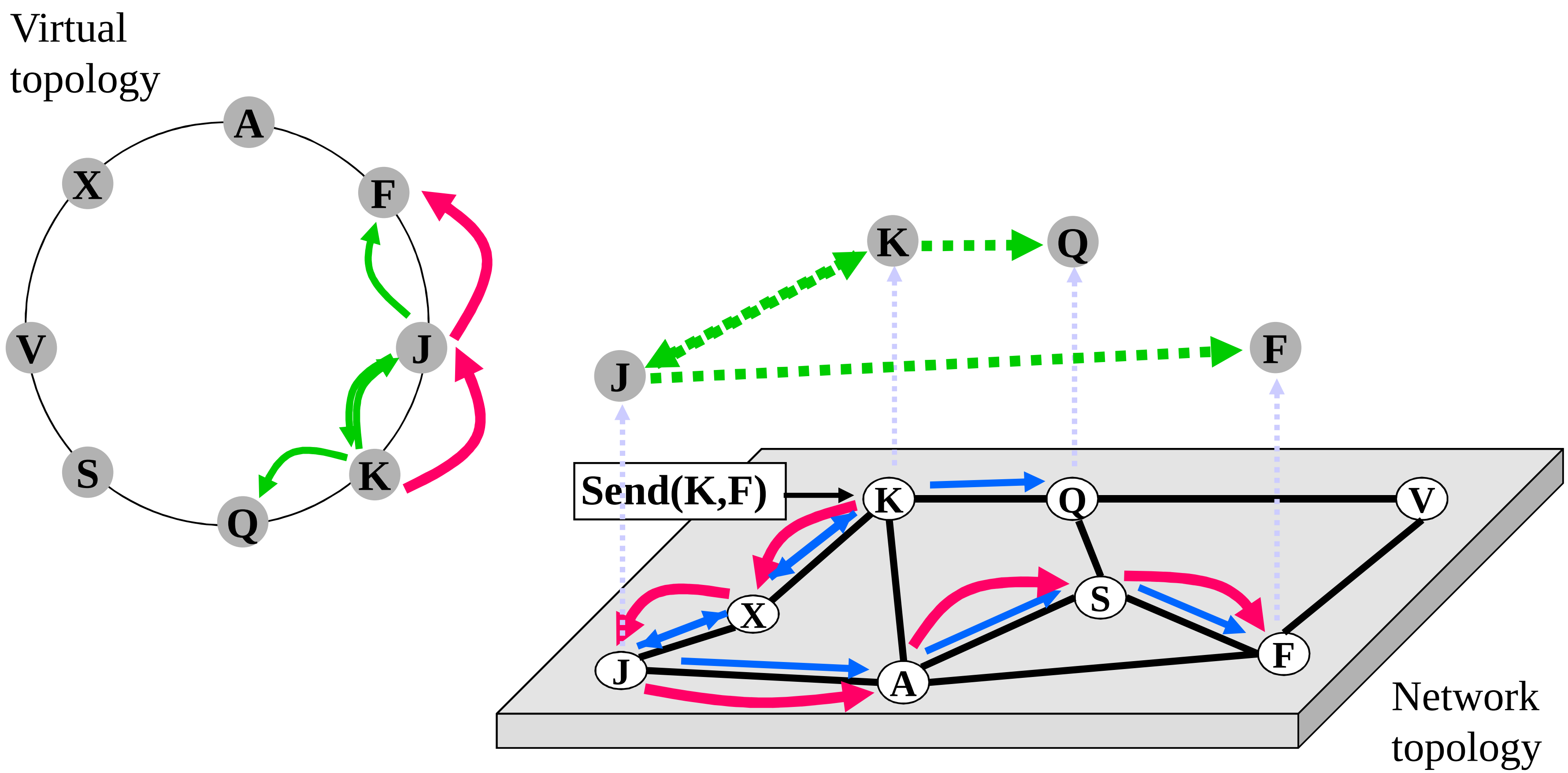




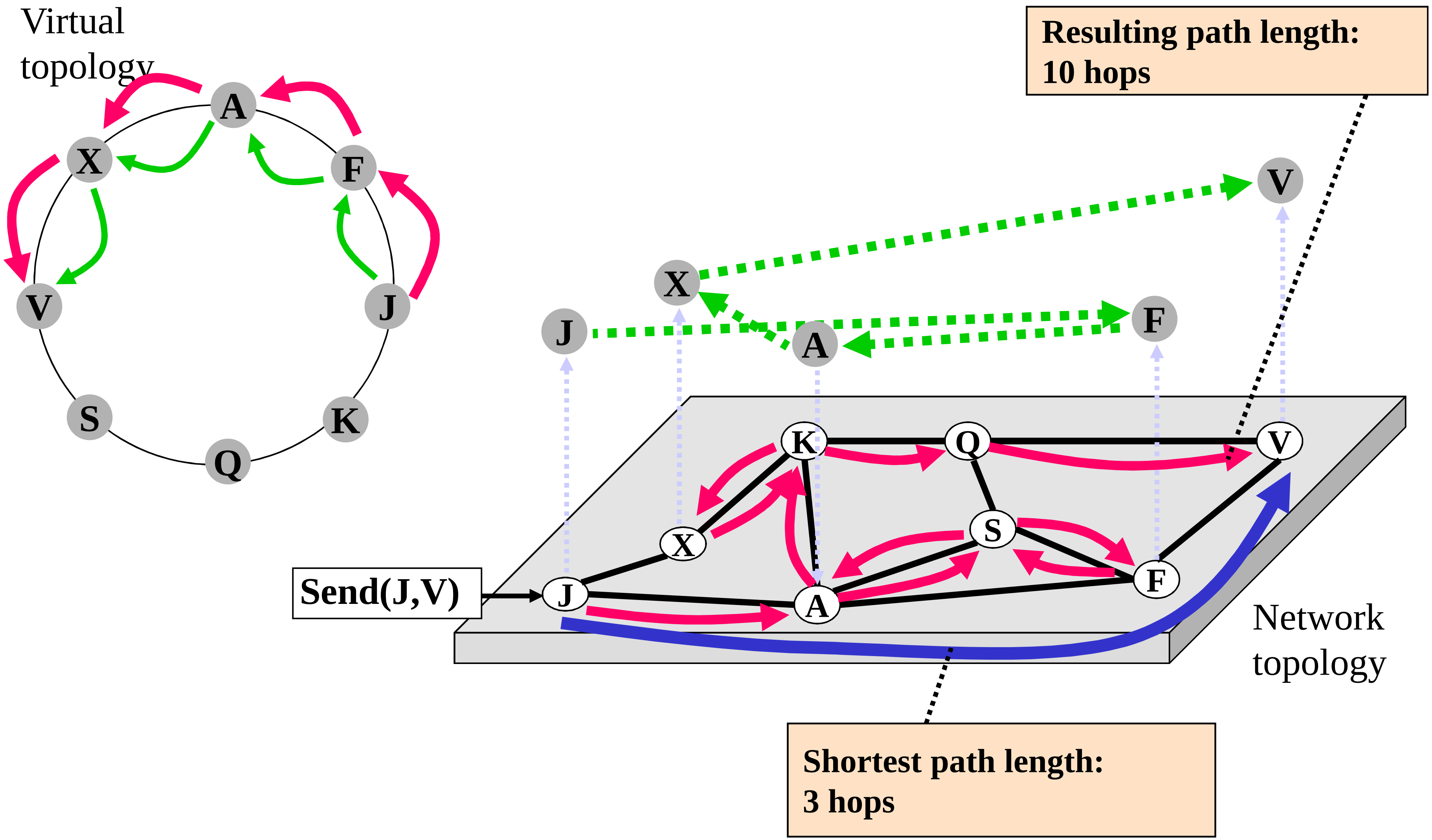
1. Write down sorted list of IDs

2. Build paths between neighbors in list

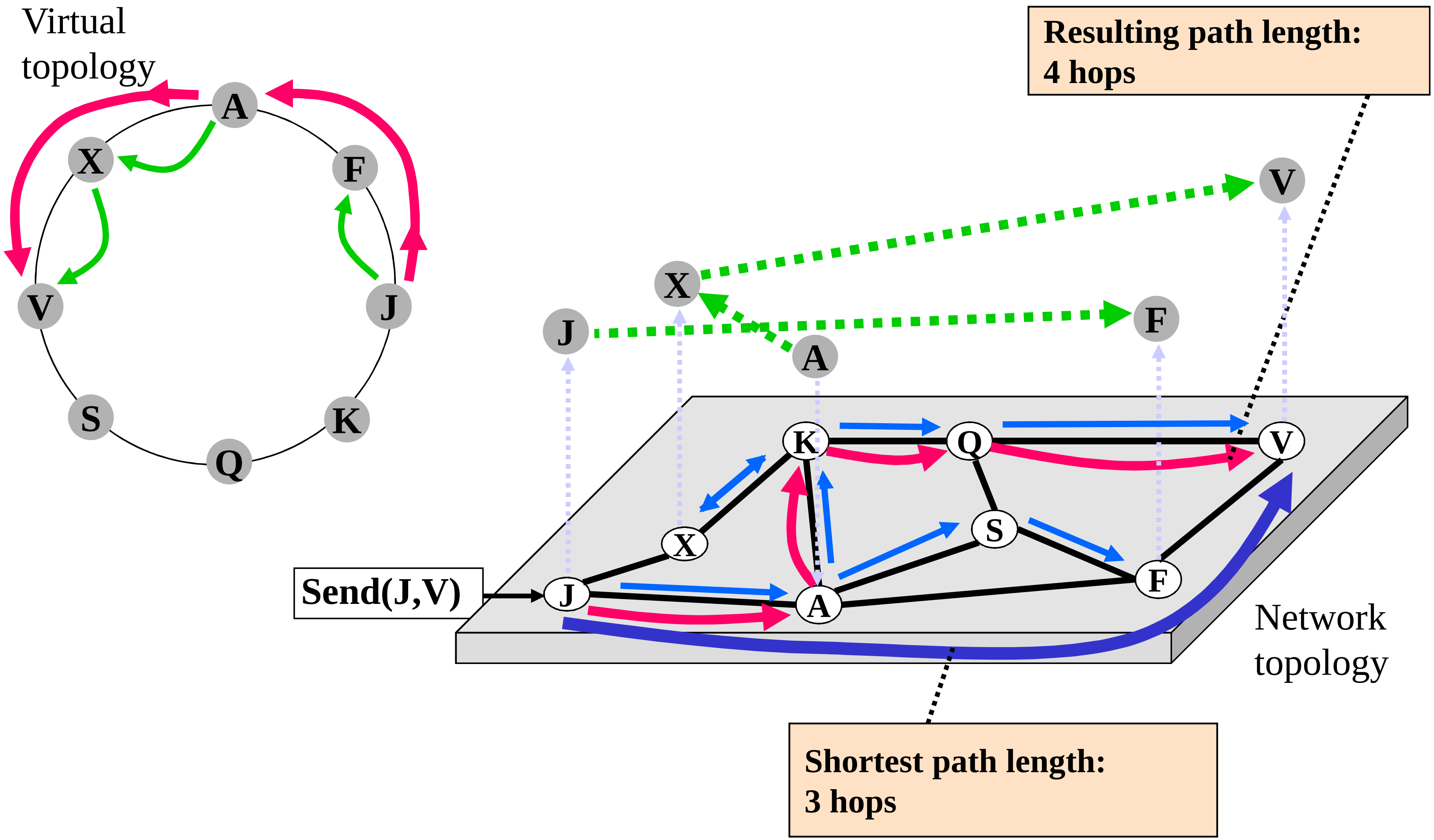
How to forward packets



The stretch problem



Optimization: shortcutting

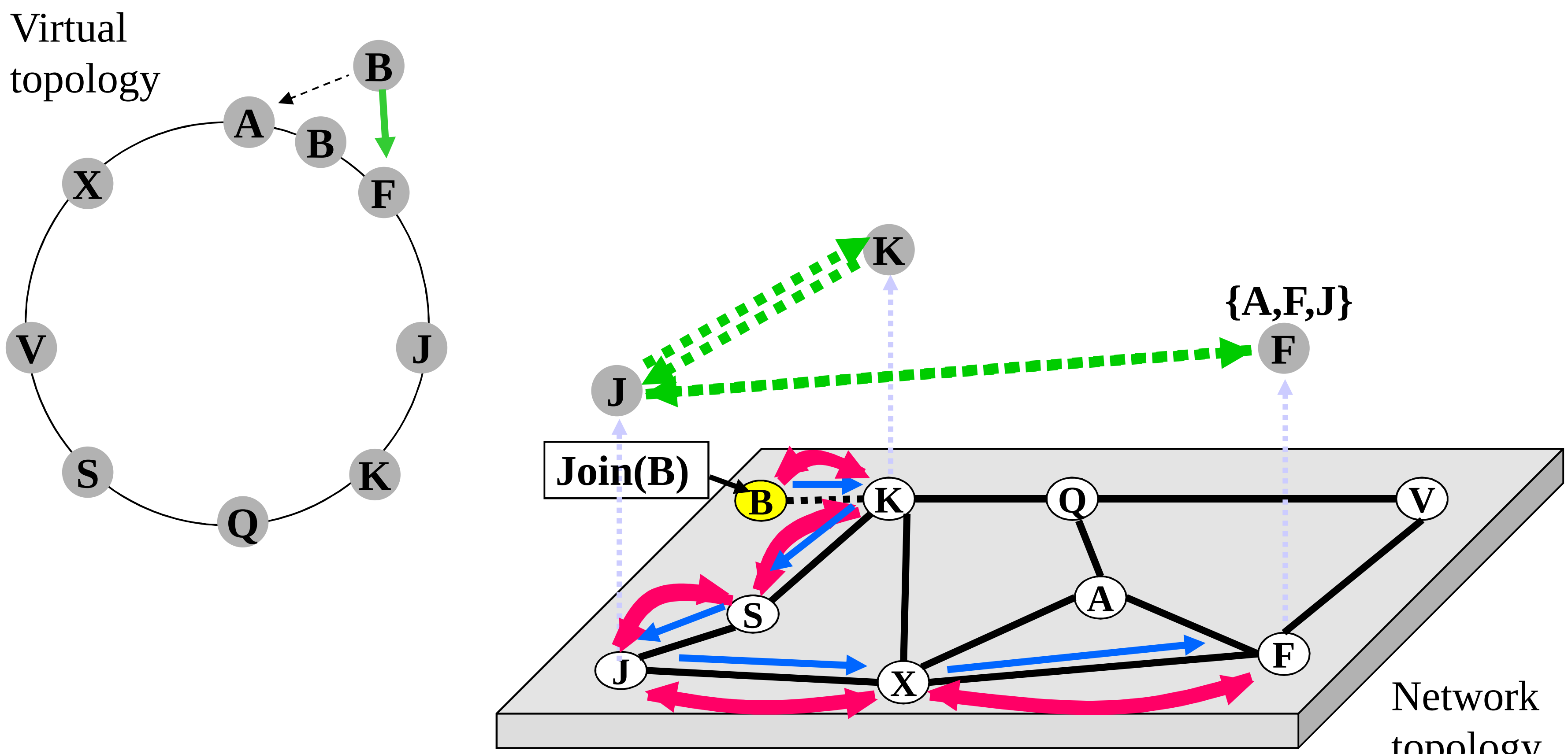


A summary so far…

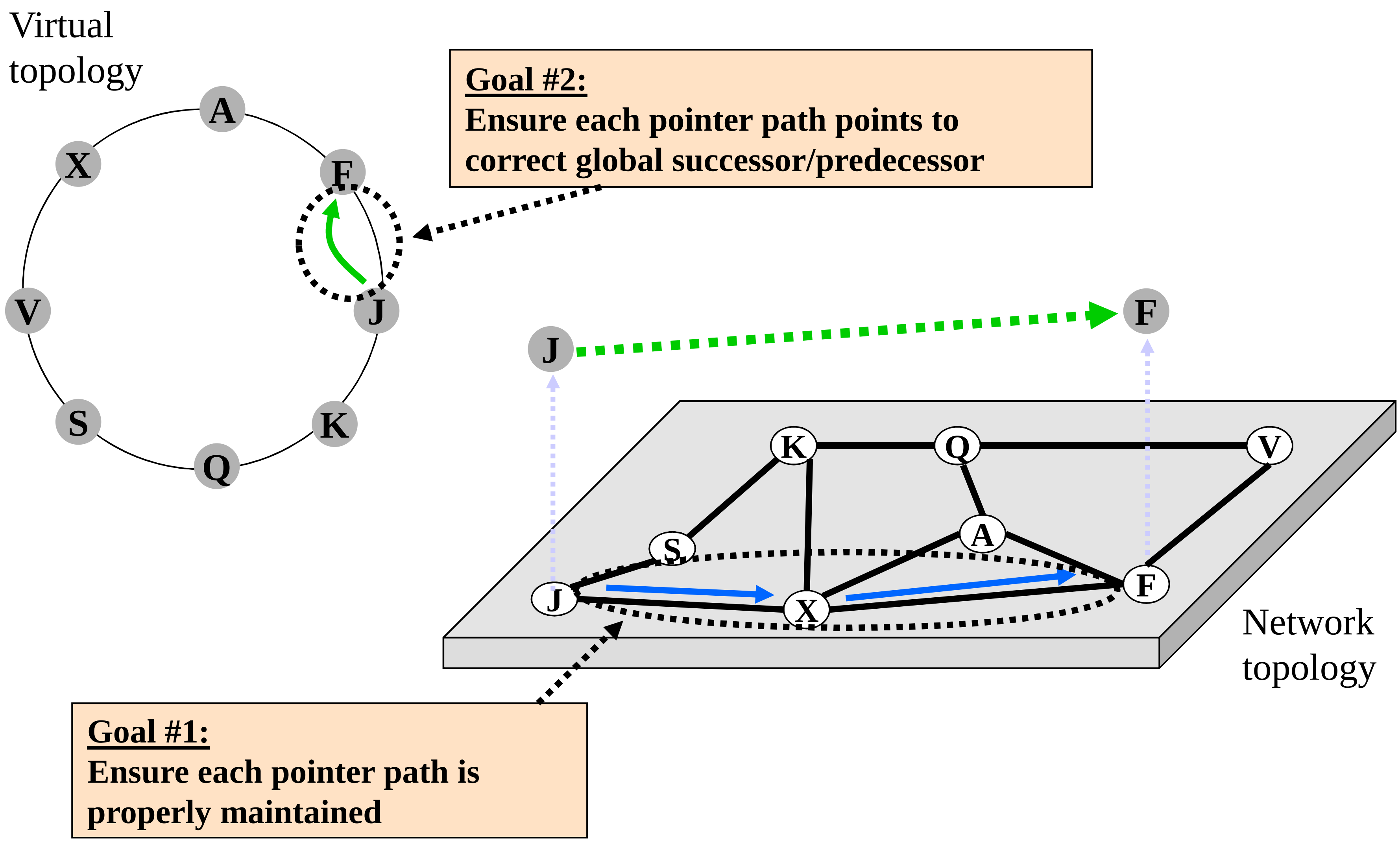
- The algorithm has two parts
  - Route linearly around the ring
  - Shortcut when possible

- Up next, the technical details…

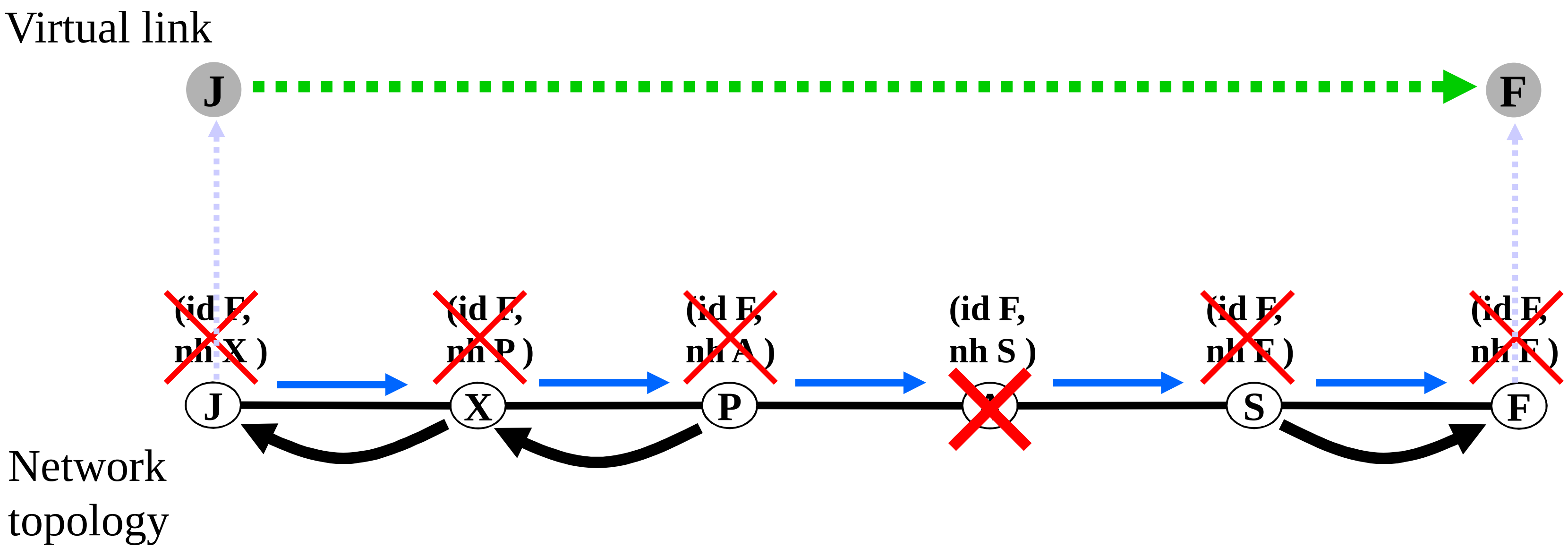
Joining a new node



How to maintain state

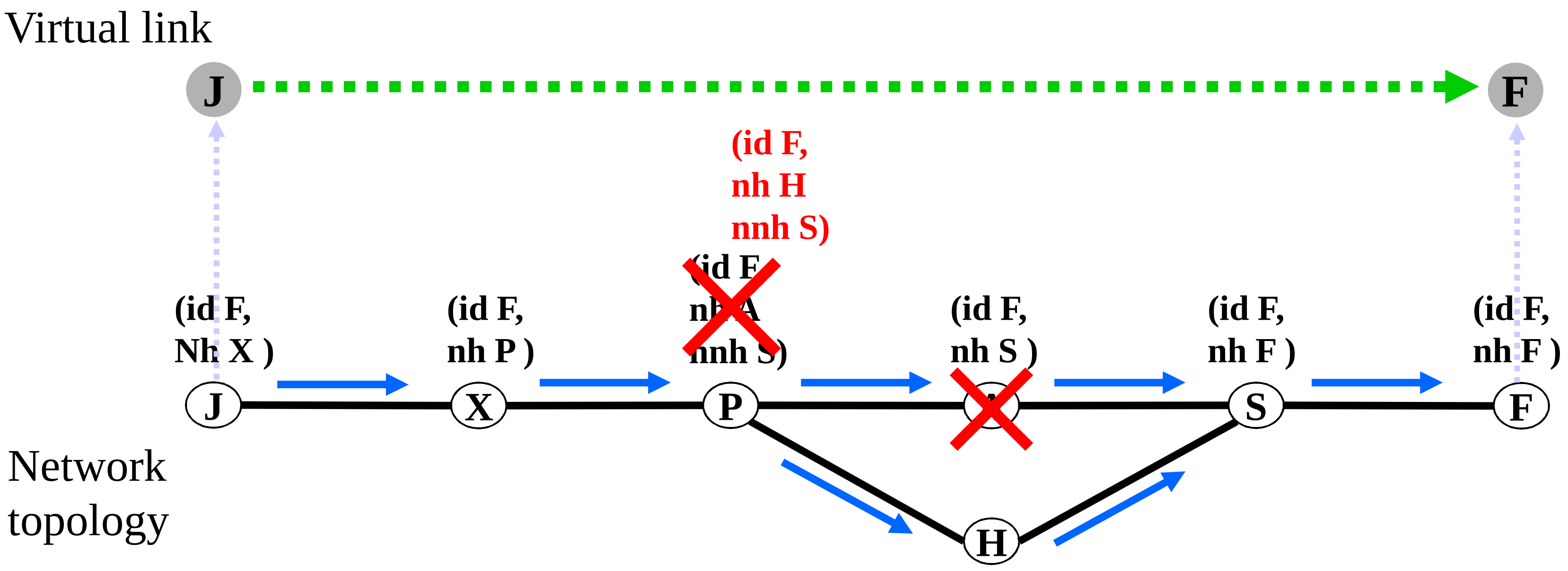


Path maintenance



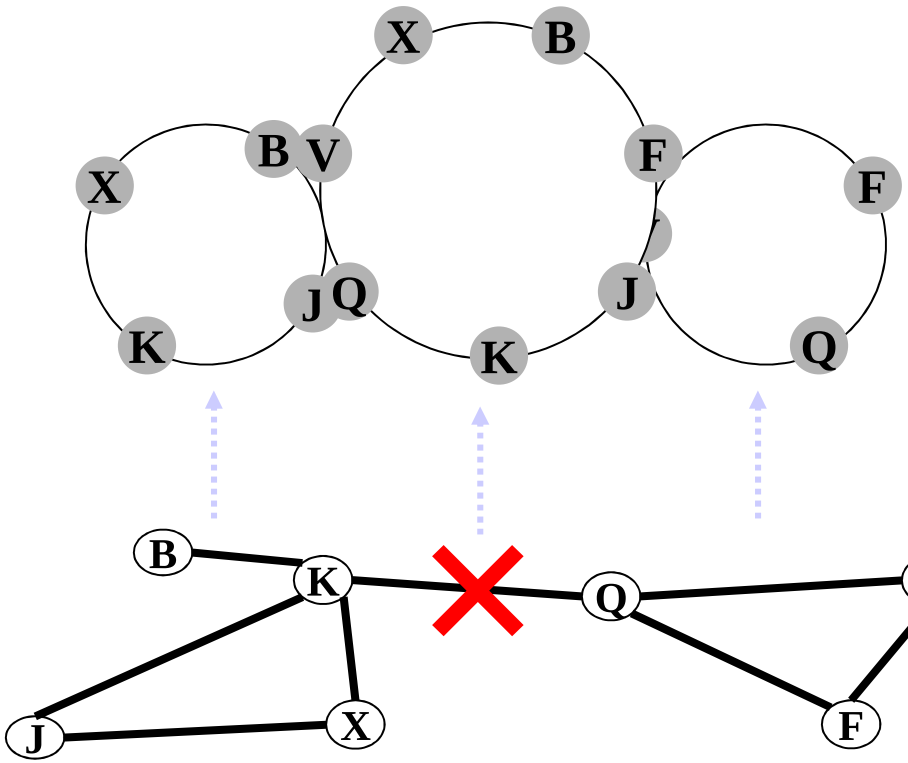
- Nodes maintain (endpoint ID, next hop) pairs per-path
- Local fault detection, teardowns remove path state
- Local repair sometimes possible

Path maintenance



- Nodes maintain (endpoint ID, next hop) pairs per-path
- Local fault detection, teardowns remove path state
- Local repair sometimes possible

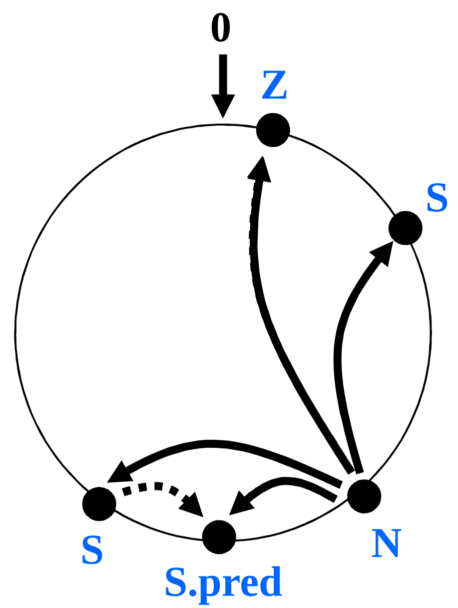
Challenges of ring maintenance



- Need to ensure network-level events don’t cause ring partitions, misconvergence

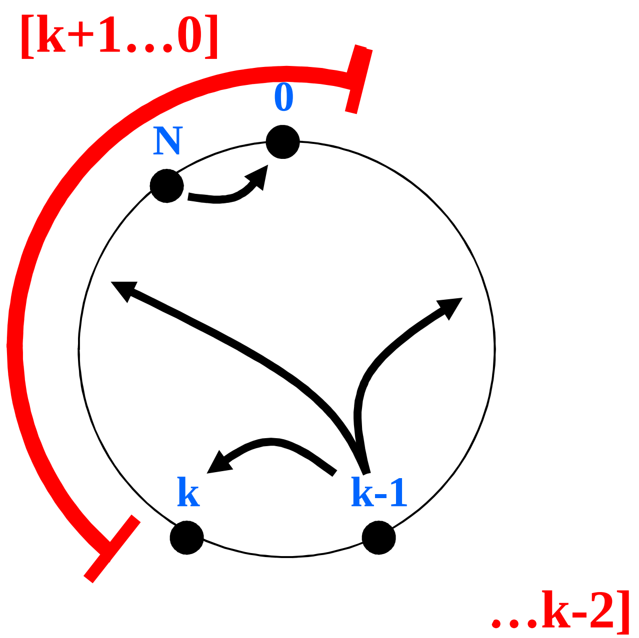
Ring maintenance

- Base mechanism:
  - Discover node Z closest to zero position, distributes Z's ID throughout partition
- Inductive mechanism:
  - Set N's successor to be the closest among:
    - N's current successor
    - N's successor's predecessor
    - The zero node Z



Ring maintenance: proof sketch

- Consider ring with nodes **{0…N}**, assume routing has converged
- Base case: **N**'s successor must point to **0**
- Inductive step: **k-1** must point to **k**
  - if **k-1** points to **S** in **[k+1…0]**, **S** would inform **k-1** about **S-1** → not converged
  - if **k-1** points to **S** in **[1…k-2]**, then **k-1** would change to point to zero node **0** → not converged



Reachability property: If there is a network level path between two nodes A and B, A can route to B via the ring

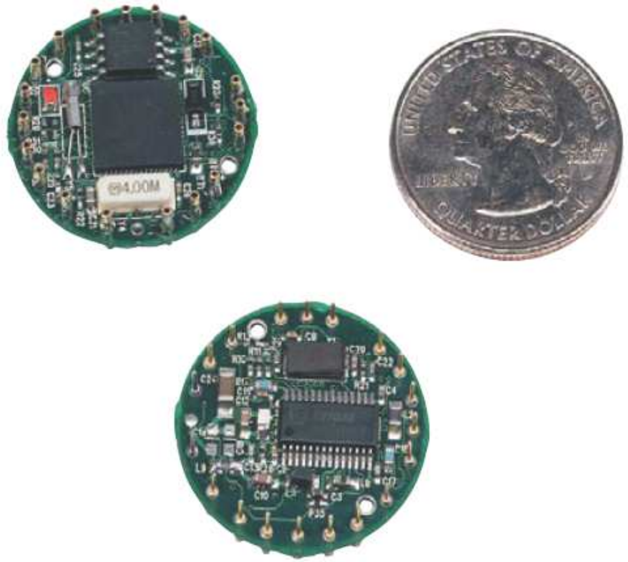
Outline

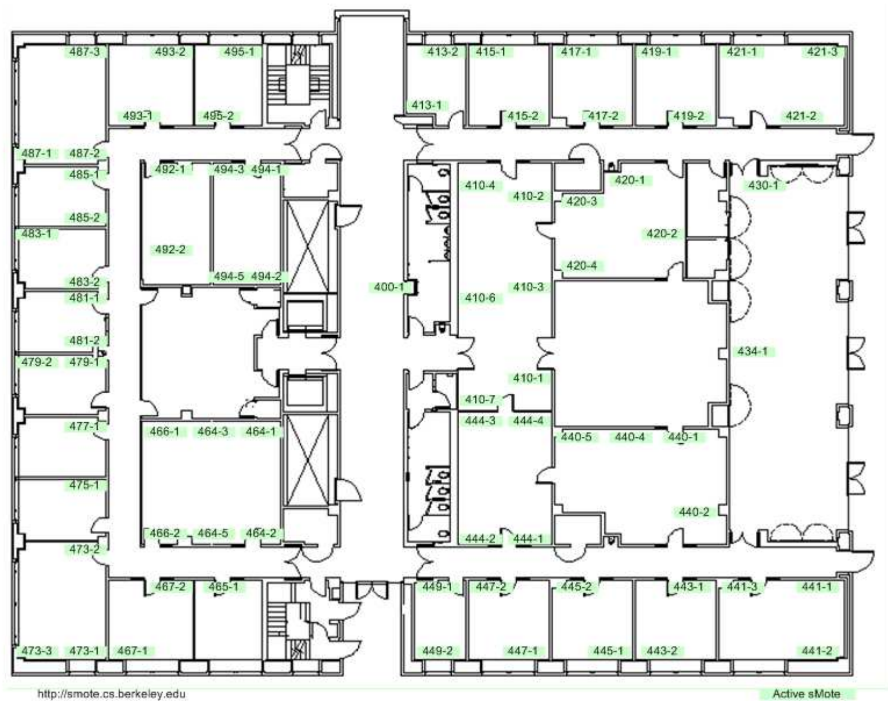
- Introduction
- Routing on an abstract graph
- Wireless sensornet implementation
  - Motivation behind using flat IDs
  - Methodology: sensornet implementation
  - Results from deployment
- Evaluation for Internet routing
- Conclusions

Why flat IDs for wireless?

- Multihop wireless networks on the horizon
  - Rooftop networks, sensornets, ad-hoc networks
- Flat IDs scale in dynamic networks
  - No location service needed
  - Flood-free maintenance reduces state, control traffic
- Developed and deployed prototype implementation for wireless sensornets
  - Extensions: failure detection, link-estimation

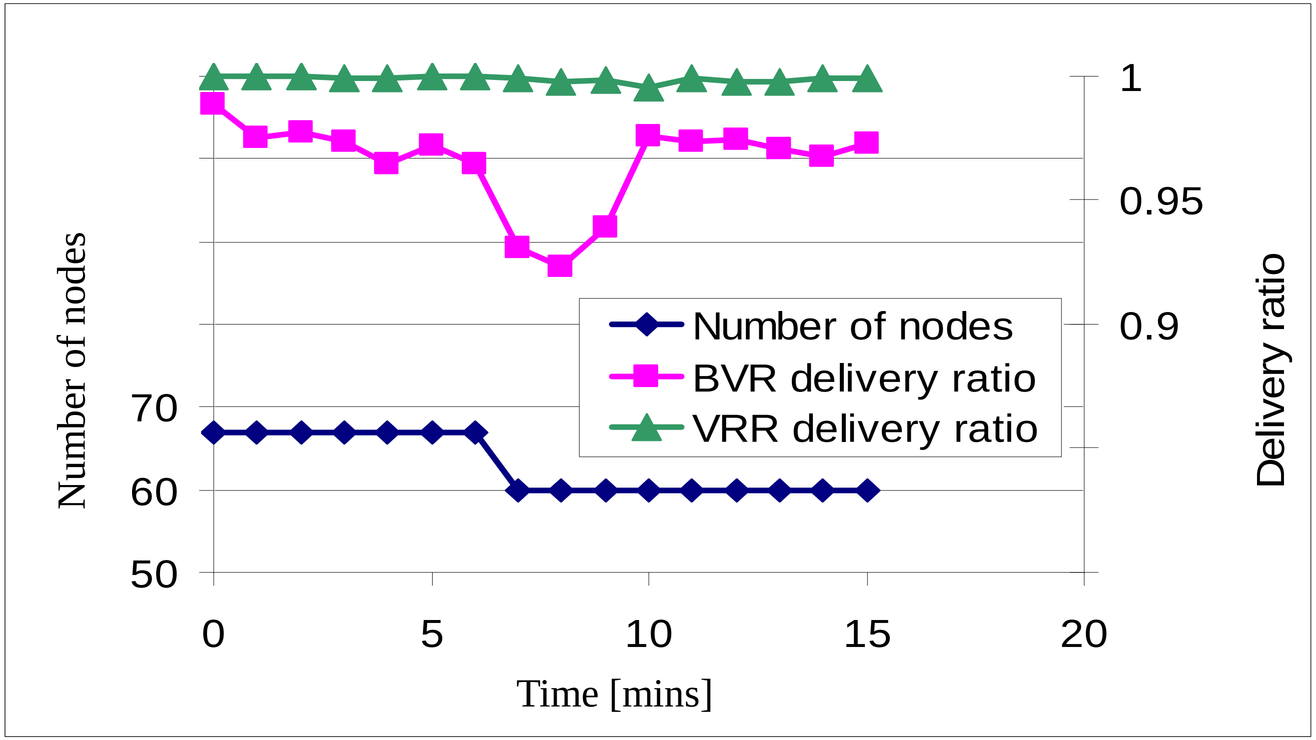
Methodology





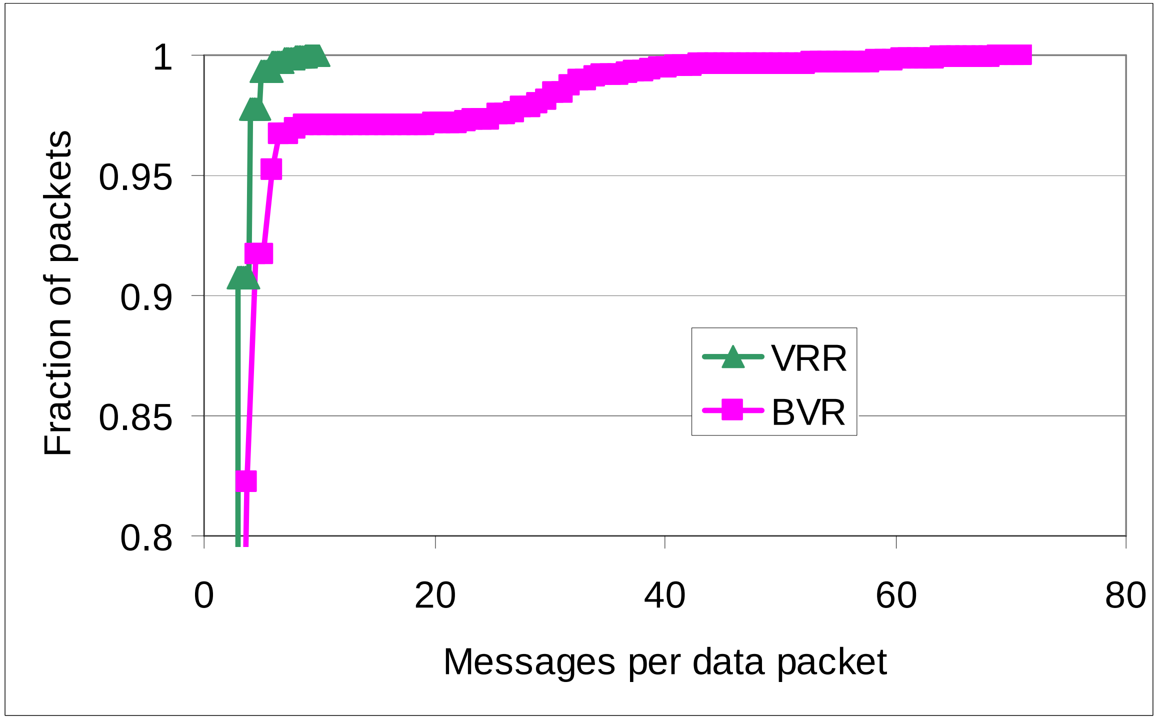
- TinyOS implementation: Virtual Ring Routing (VRR)
  - Deployment on testbed: 67 mica2dot motes (4KB memory, 19.2kbps radio)
  - Compared with Beacon Vector Routing (BVR), AODV, DSR
- Metrics: Delivery ratio, control overhead

Effect of node failure



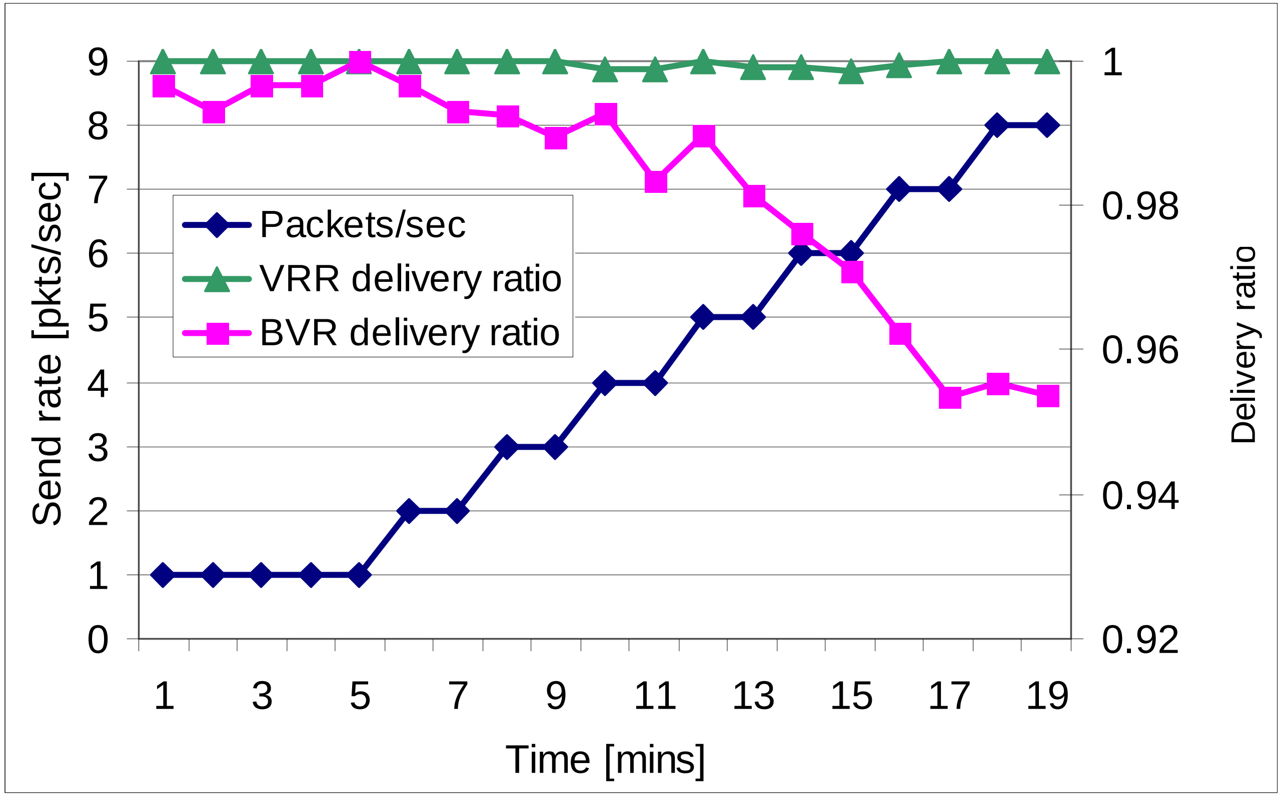
- Both VRR and BVR perform well
- BVR's performance degrades because of coordinate instability and overhead to recover from failures

Transmission overhead



- Flat routing requires no scoped flooding, which reduces transmission overhead

Effect of congestion



- Flat-routing resilient to congestion losses, since identifiers topology-independent

Outline

- Introduction
- Routing on an abstract graph
- Wireless sensornet implementation
- Evaluation for Internet routing
  - Motivation behind using flat IDs
  - Extensions to support policies, improve scaling
  - Performance evaluation on Internet-size graphs
- Conclusions

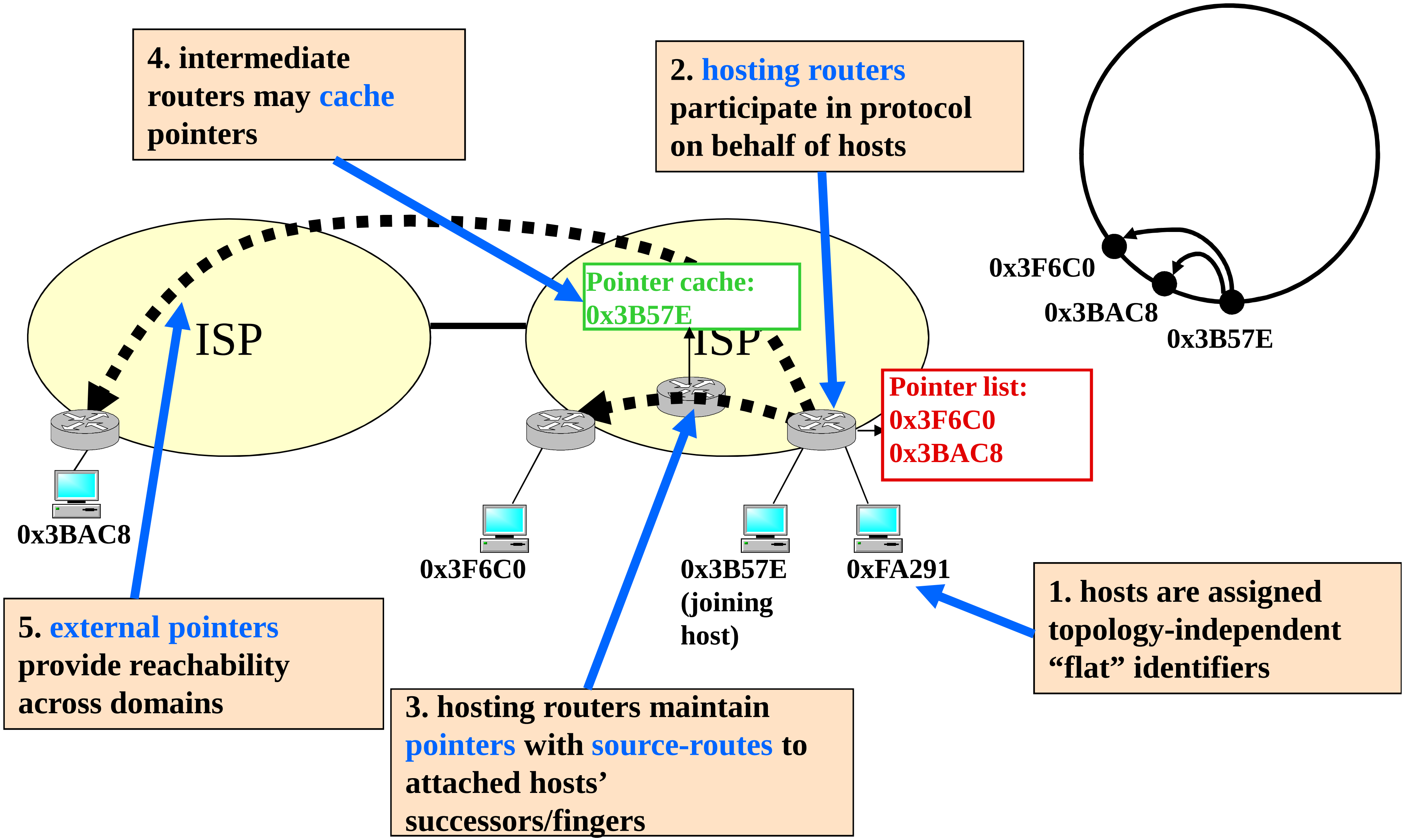
Why flat IDs for the Internet?

- Today's Internet conflates addressing with identity
- Flat IDs sidestep this problem completely
  - Provides network routing without any mention of location
  - Benefits: no need for name resolution service, simpler configuration, simpler access controls

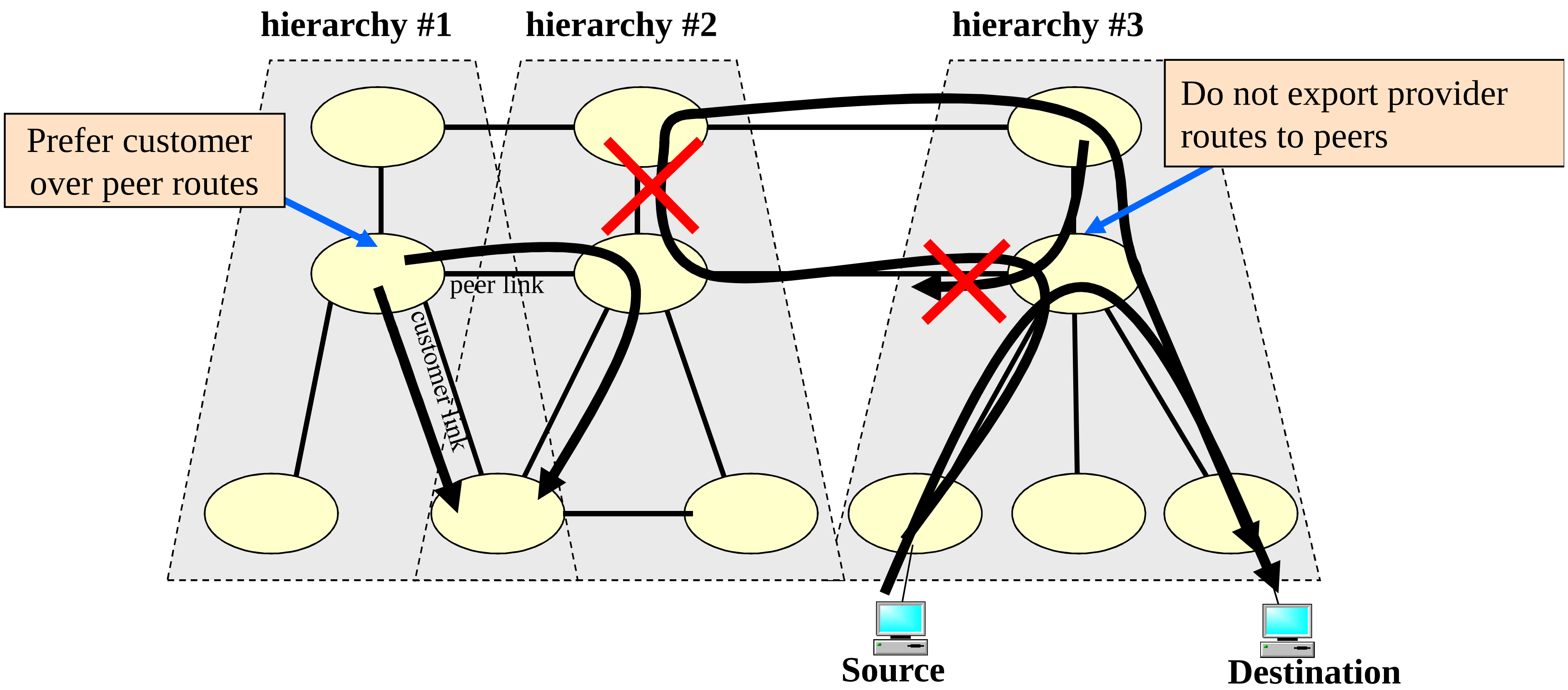
Challenges of Internet routing

- Internet routing is very different from wireless routing
  - Challenges: policies, scaling
- Need new mechanisms to deal with these challenges
  - Policy-safe successor paths
  - Locality-based pointer caching

Flat IDs for Internet routing

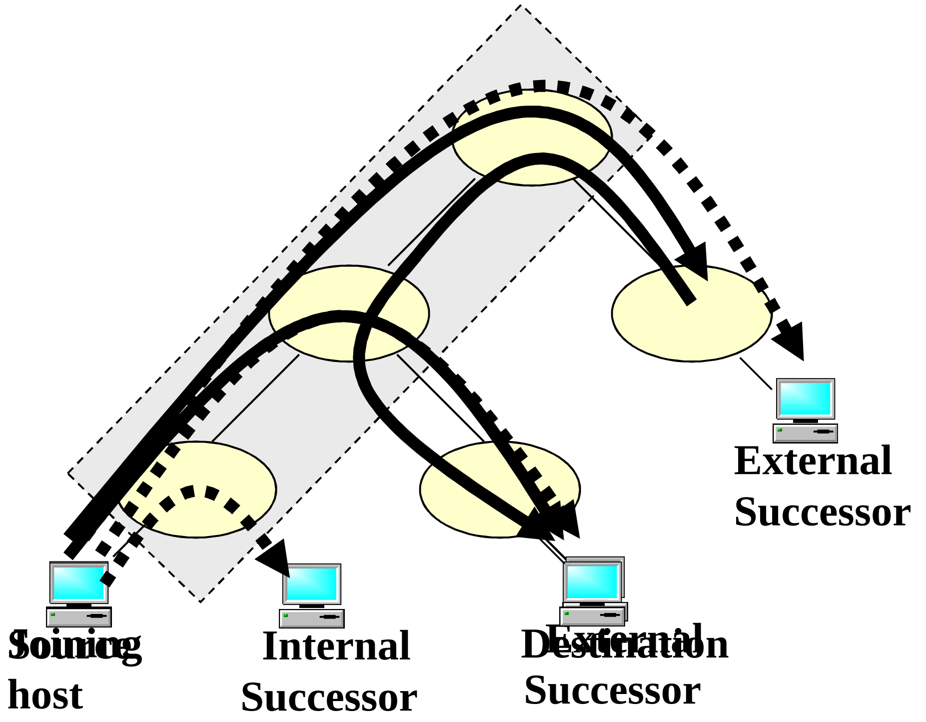


Internet policies today



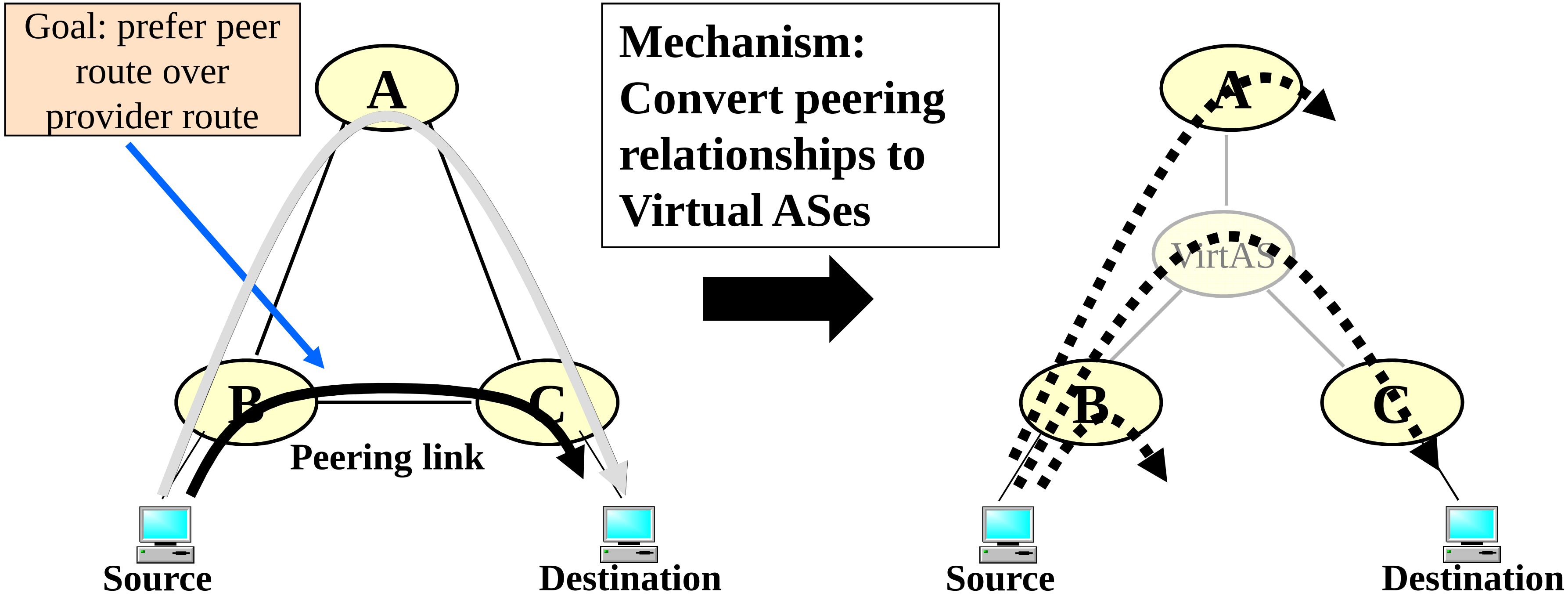
- **Economic relationships:** peer, provider/customer
- **Isolation:** routing contained within hierarchy

Isolation



Isolation property: traffic between two hosts traverses no higher than their lowest common provider in the ISP hierarchy

Policy support



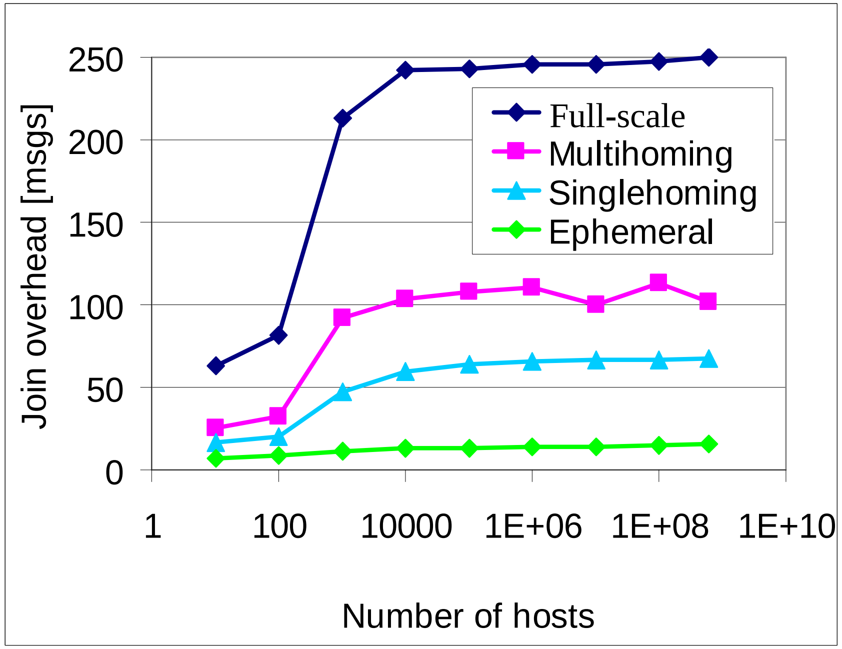
- Peering
- Provider-customer
- Backup

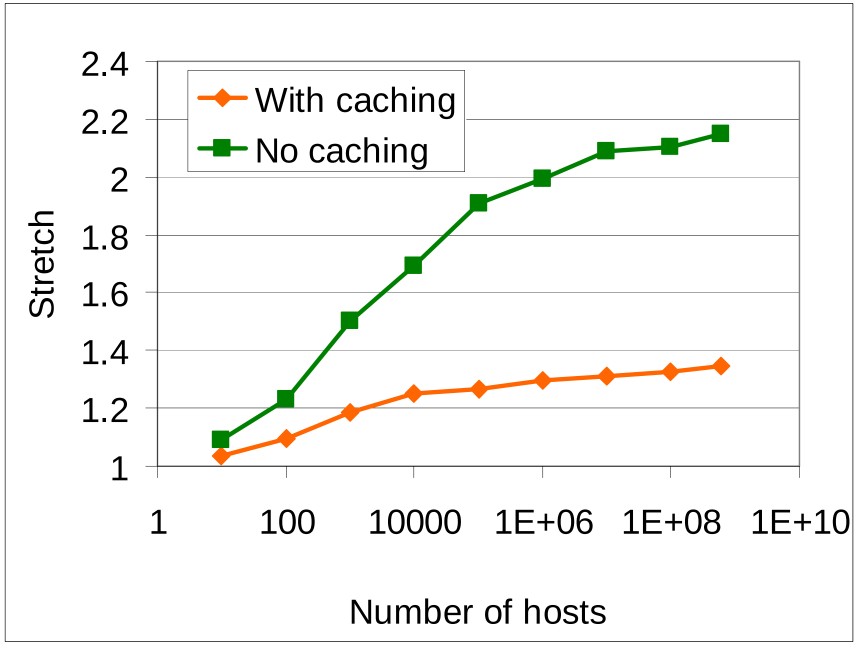
Traffic respects peering, backup, and provider-customer relationships

Evaluation

- Distributed packet-level simulations
  - Deployed on cluster across 62 machines, scaled to 300 million hosts
  - Inferred Internet topology from Routeviews, Rocketfuel, CAIDA skitter traces
- Implementation
  - Ran on Planetlab as overlay, covering 82 ASes
  - Configured inter-ISP policies from Routeviews traces
- Metrics: stretch, control overhead

Internet-scale simulations





- Join overhead <300 msgs, stretch < 1.4
- Root-server lookups inflate latency from 54ms to 134ms, Flat IDs has no penalty