

# CS5412: OVERLAY NETWORKS

Lecture IV

Ken Birman

# Overlay Networks

2

- We use the term *overlay network* when one network (or a network-like data structure) is superimposed upon an underlying network
- We saw this idea at the end of lecture III
- Today we'll explore some examples
  - ▣ The MIT “Resilient Overlay Network” (RON)
  - ▣ Content-sharing overlays (Napster, Gnutella, dc++)
  - ▣ Chord: An overlay for managing (key,value) pairs. Also known as a *distributed hash table* or DHT.

# Why create a overlay?

3

- Typically, we're trying to superimpose some form of routed behavior on a set of nodes
- The underlying network gives the nodes a way to talk to each other, e.g. over TCP or with IP packets
- But we may want a behavior that goes beyond just being able to send packets and reflects some kind of end-user "behavior" that we want to implement

# VPN overlays

4

- Many CS5412 students have experience with VPNs
  - ▣ A kind of remote login to your company or University
  - ▣ Allows you to access site securely through a firewall
  
- A VPN usually works by
  - ▣ Negotiating a security key (using saved credentials plus some form of password)
  - ▣ Making a TCP SSL (TLS) connection to a server
  - ▣ “Tunneling” traffic over that link; the IP address space of the VPN is available via this route

# Next example: RON

5

- Developed at MIT by a research group that
  - ▣ Noticed that Internet routing was surprisingly slow to adapt during overloads and other problems
  - ▣ Wanted to move data and files within a set of nodes
  - ▣ Realized that “indirect” routes often outperformed direct ones
- What do we mean by an indirect route?
  - ▣ Rather than send file F from A to B, A sends to C and C relays the file to B
  - ▣ If the A-B route is slow, perhaps A-C-B will be faster

# But doesn't Internet "route around" congestion?

6

- Early Internet adapted routing very frequently
  - ▣ Circumvent failed links or crashed routers
  - ▣ Cope with periodic connectivity, like dialup modems that are only connected now and then
  - ▣ Spread network traffic evenly by changing routing when loads change
- By 1979 a problem was noticed
  - ▣ Routing messages were creating a LOT of overhead
  - ▣ In fact the rate of growth of this overhead was faster than the rate of growth of the network size & load!

# How can overheads grow so fast?

7

- Think about the idea of algorithmic complexity
  - ▣ Like for sorting
- In a single machine, we know that sorting takes time  $O(n \log n)$  but that bubble sort is slow and takes time  $O(n^2)$ .
  - ▣ Both do the same thing
  - ▣ But bubble sort is just an inefficient way to do it
  - ▣ Leads to notion of asymptotic complexity

# Protocols have complexity too!

8

- Can be measured in many ways
  - ▣ How many messages are sent in total on the network?
  - ▣ How many do individual nodes send or receive?
  - ▣ How many “rounds” of the protocol are required
  - ▣ How many bytes of data are exchanged?
    - Of this how much is legitimate data and how much was added by the protocol?
    - Of the legitimate data, how many bytes are ones the receiver has never seen, and how many are duplicates?
  - ▣ How directly does data go from source to destination?



# Complexity of routing protocols

- Routing protocols vary widely in network complexity
- BGP, for example, is defined in terms of dialog between a BGP instance and its peers
  - ▣ At start, sends initialization messages that inform peers of the full routing table.
  - ▣ Subsequently, sends “incremental” update messages that announce new routes and withdraw old ones
- To understand the complexity of BGP we need to understand relationship between frequency of these packets size of network, and rate of network “events”

# BGP complexity study

10

- Can be evaluated using theory tools.
- Create a model... then present equations that predict costs in terms of event rates

[Bringing order to BGP: decreasing time and message complexity. Anat Bremler-barr, Nir Chen, Jussi Kangasharju, Osnat Mokryn, Yuval Shavitt. ACM Principles of Distributed Computing (PODC), Aug. 2007, pp. 368-369.]

# But more common to just use practical tools

- For example, back in 1979, Internet developers simply measured the percentage of network traffic that was due to network management protocols
- They discovered it was quite high and rising
- Concluded that steps were needed to reduce costs
  - ▣ Eliminated routing protocols that had higher overheads
  - ▣ Reduced rate of routing adaptations

# Today's Internet?

12

- There are *many* reasons routing adapts slowly
  - Old desire to keep overheads low
  - Modern need to route heavy traffic on economically efficient paths
  - Many policies and “cross-border” deals between ASs enter the picture
  - Best route is the cheapest route to operate not necessarily the route that makes the A-B file transfer move fastest!

# How RON approaches this

13

- They built an infrastructure that supports *IP tunneling*
  - ▣ Means that a packet from A to B might be treated as data and placed within a packet from A to C
  - ▣ Sometimes called “IP over IP”
- Now they can implement their own special routing layer that decides how to get data from A to B
  - ▣ A sends packet
  - ▣ RON intercepts it and “encapsulates” it for tunneling
  - ▣ Routes on its own routing infrastructure (still on the Internet)
  - ▣ On arrival, de-encapsulate and deliver

# How RON approaches this

14

- Build an all-to-all monitoring tool to track bandwidth and delay (latency)
  - Part of the trick was to estimate one-way costs
  - For brevity won't delve into those details
- This results in a table (we'll just show latency):

	<b>A</b>	<b>B</b>	<b>C</b>
<b>A</b>	-	17	9
<b>B</b>	5	-	22
<b>C</b>	14	2	-

- Note that A-B delay is 17ms, but A-C is 9 and C-B 2

# Source routing

15

- RON sender
  - ▣ Computes the best route considering direct and also one-hop indirect routes
  - ▣ Encapsulated packets
  - ▣ Specifies the desired routing in a special header: a form of “source routing”
- RON daemons relay the packet as instructed
- On arrival, extract inner packet and deliver it

# RON really works!

16

- MIT studies showed big performance speedups using this technique!
  - ▣ In fact the direct routes are almost *always* worse than the best indirect routes
  - ▣ And a single indirect hop is generally all they needed (double indirection adds too much delay)
- RON also adapts quickly
  - ▣ Internet routes much more slowly



# Learning from history...



17

- Concept: *Tragedy of the Commons* (or “Crisis”)
  - ▣ We share a really great resource (the “commons”)
  - ▣ But someone decides to use the commons for themselves in an unsustainable way and gains economic advantage
  - ▣ We need to be competitive, so all of us do the same
  - ▣ This denudes the commons... Everyone loses
- When we share a limited resource, sometimes the best shared policy isn't the best individual one

# What does this say about RON?

18

- For the individual user, RON makes things better
- But if we believe that economics has “shaped” the Internet, RON basically cheats!
  - ▣ In effect, the RON user is getting more network resource than he’s paying for by circumventing the normal sharing policy
  - ▣ If everyone did this, the RON approach would break down much as the commons ends up with no grass left

# Broader theory...

19

- The research community has been interested in what are called “Nash Equilibria”
- Idea is that a set of competitors each have a “utility” function (a measure of happiness) and sets of strategies that guide their action
  - ▣ Such as “decide to graze my cow on the commons”
- Goal is to find a configuration where if any player were to use some other strategy, they would lose utility
  - ▣ In principle we all see the logic of the optimal strategy
  - ▣ But assumes that players are logical and able to see big picture

# Other cases for overlays?

- A major use of overlays has been in *peer to peer file sharing services* such as Napster, Gnutella, dc++
  
- These generally have two aspects
  - ▣ A way to create a list of places that have the file you want (perhaps, a movie you want to download)
  - ▣ A way to connect to one of those places to pull the file from that machine to yours
    - Once you have the file, your system becomes a possible source for other users to download from
    - In practice, some users tend to run servers with better resources and others tend to be mostly downloaders

# A mix of technical and non-technical issues

21

- Non-technical: what is the “tragedy of the commons” scenario if everyone uses these sharing services?
- How should the law deal with digital IP ownership
- If a web search helps you find “inappropriate” content, or an ISP happens to carry that, were they legally responsible for doing so?

# Technical issue

22

- What's the very best way for a massive collection of computers in the wide-area Internet (the WAN) to implement these two aspects
  - ▣ Best way to do search?
  - ▣ Best way to implement peer-to-peer downloads?
  
- Cloud computing solutions often have a search requirement so we'll focus on that
  - ▣ Useful even within a single data center

# Context

23

- We have a vast number of machines (millions)
- Goal is to support (key,value) operations
  - ▣ Put(key,value) stores this value in association with key
  - ▣ Get(key) finds the value currently bound to this key
- Some systems allow updates, some allow multiple bindings for a single key. We won't worry about those kinds of detail today

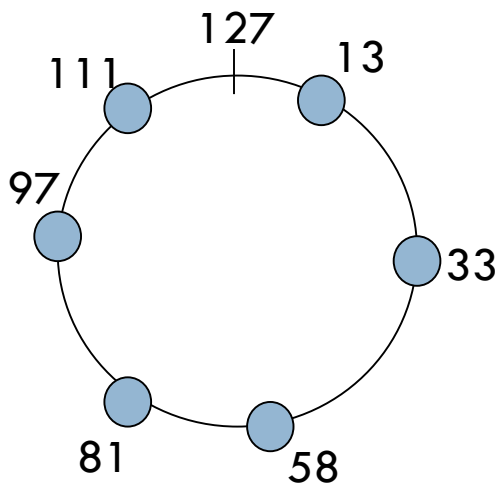
# P2P “environment”

24

- Nodes come and go at will (possibly quite frequently---a few minutes)
- Nodes have heterogeneous capacities
  - ▣ Bandwidth, processing, and storage
- Nodes may behave badly
  - ▣ Promise to do something (store a file) and not do it (free-loaders)
  - ▣ Attack the system



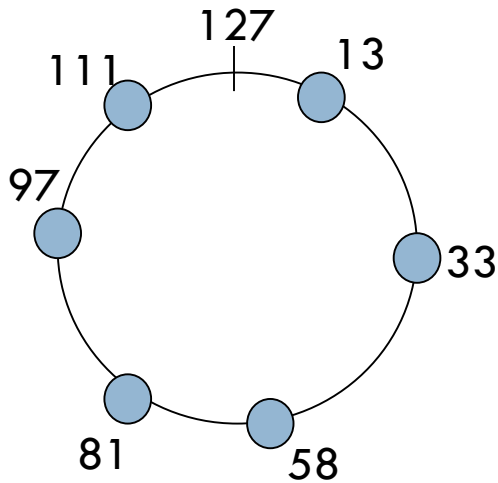
# Basics of all DHTs



- Goal is to build some “structured” overlay network with the following characteristics:
  - ▣ Node IDs can be mapped to the hash key space
  - ▣ Given a hash key as a “destination address”, you can route through the network to a given node
  - ▣ Always route to the same node no matter where you start from

# Simple example (doesn't scale)

26

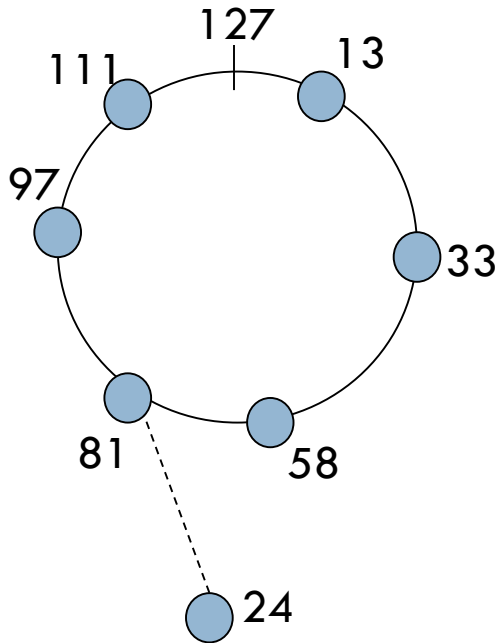


- Circular number space 0 to 127
- Routing rule is to move counter-clockwise until current node ID  $\geq$  key, and last hop node ID  $<$  key
- Example: key = 42
- Obviously you will route to node 58 from no matter where you start

# Building any DHT

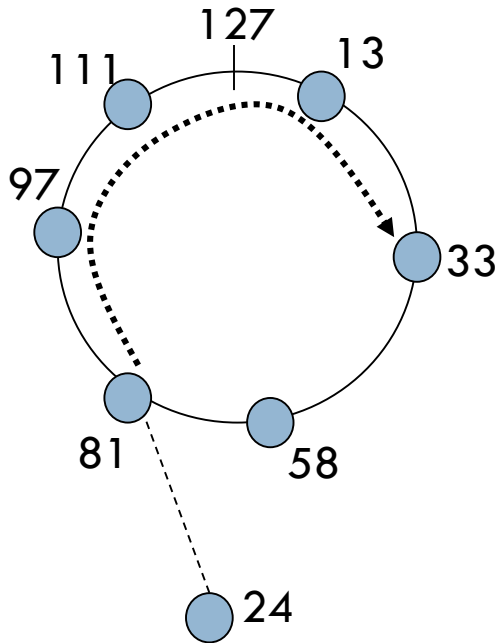
27

- Newcomer always starts with at least one known member



# Building any DHT

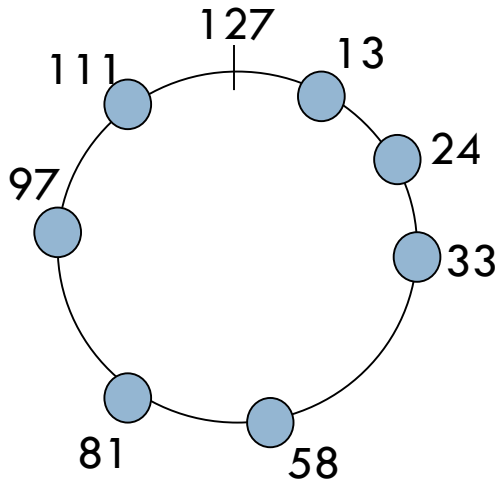
28



- Newcomer always starts with at least one known member
- Newcomer searches for “self” in the network
  - ▣ hash key = newcomer’s node ID
  - ▣ Search results in a node in the vicinity where newcomer needs to be

# Building any DHT

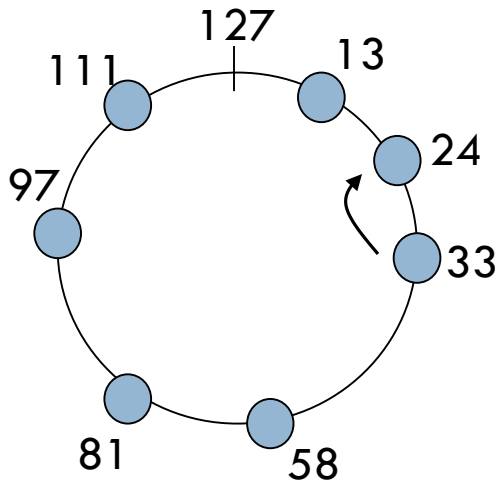
29



- Newcomer always starts with at least one known member
- Newcomer searches for “self” in the network
  - ▣ hash key = newcomer’s node ID
  - ▣ Search results in a node in the vicinity where newcomer needs to be
- Links are added/removed to satisfy properties of network

# Building any DHT

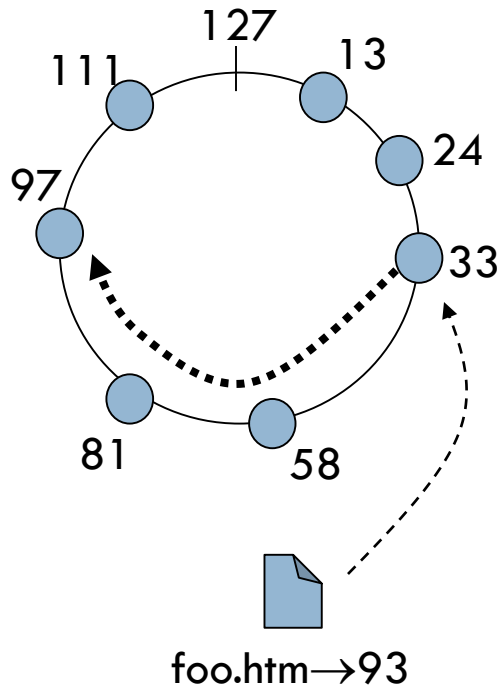
30



- Newcomer always starts with at least one known member
- Newcomer searches for “self” in the network
  - ▣ hash key = newcomer’s node ID
- Search results in a node in the vicinity where newcomer needs to be
- Links are added/removed to satisfy properties of network
- **Objects that now hash to new node are transferred to new node**

# Insertion/lookup for any DHT

31



- Hash name of object to produce key
  - Well-known way to do this
- Use key as destination address to route through network
  - Routes to the target node
- Insert object, or retrieve object, at the target node

# Properties of most DHTs

32

- Memory requirements grow (something like) logarithmically with  $N$
- Unlike our “any DHT”, where routing is linear in  $N$ , real DHTs have worst possible routing path length (something like) logarithmic with  $N$
- Cost of adding or removing a node grows (something like) logarithmically with  $N$
- Has caching, replication, etc...



# DHT Issues

33

- Resilience to failures
- Load Balance
  - ▣ Heterogeneity
  - ▣ Number of objects at each node
  - ▣ Routing hot spots
  - ▣ Lookup hot spots
- Locality (performance issue)
- Churn (performance and correctness issue)
- Security

# We're going to look at four DHTs

34

- At varying levels of detail...
  - CAN (Content Addressable Network)
    - ACIRI (now ICIR)
  - Chord
    - MIT
  - Kelips
    - Cornell
  - Pastry
    - Rice/Microsoft Cambridge

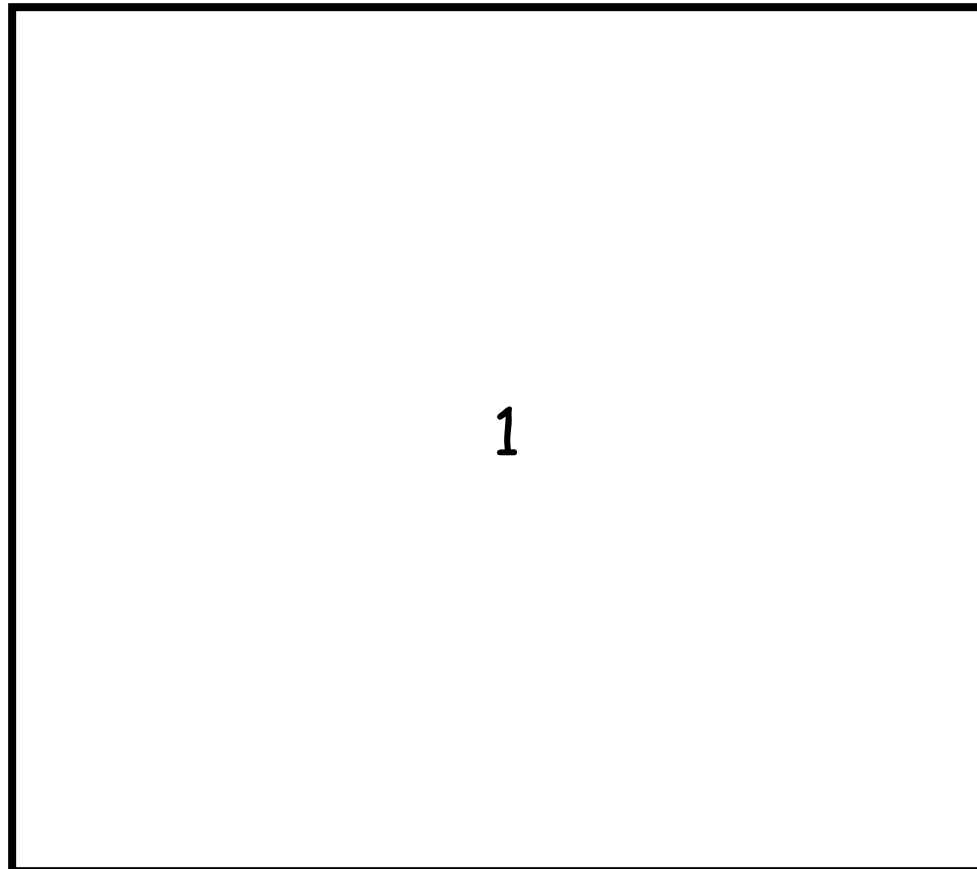
# Things we're going to look at

35

- What is the structure?
- How does routing work in the structure?
- How does it deal with node departures?
- How does it scale?
- How does it deal with locality?
- What are the security issues?

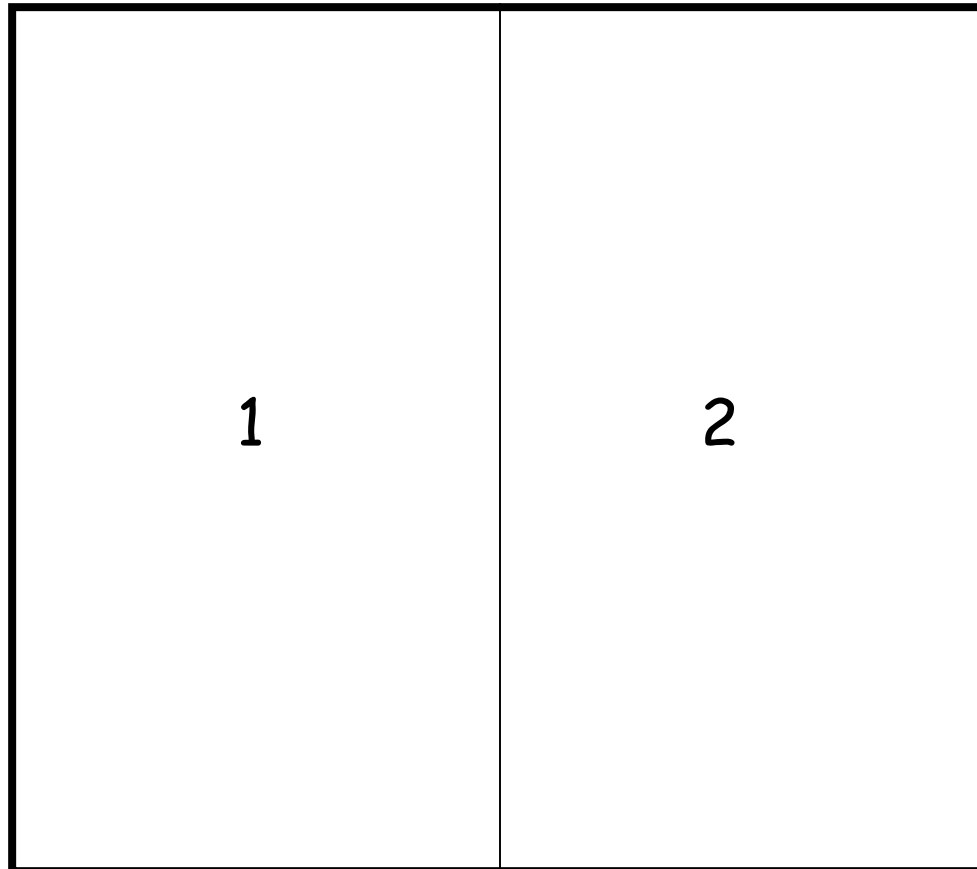
# CAN structure is a cartesian coordinate space in a D dimensional torus

36



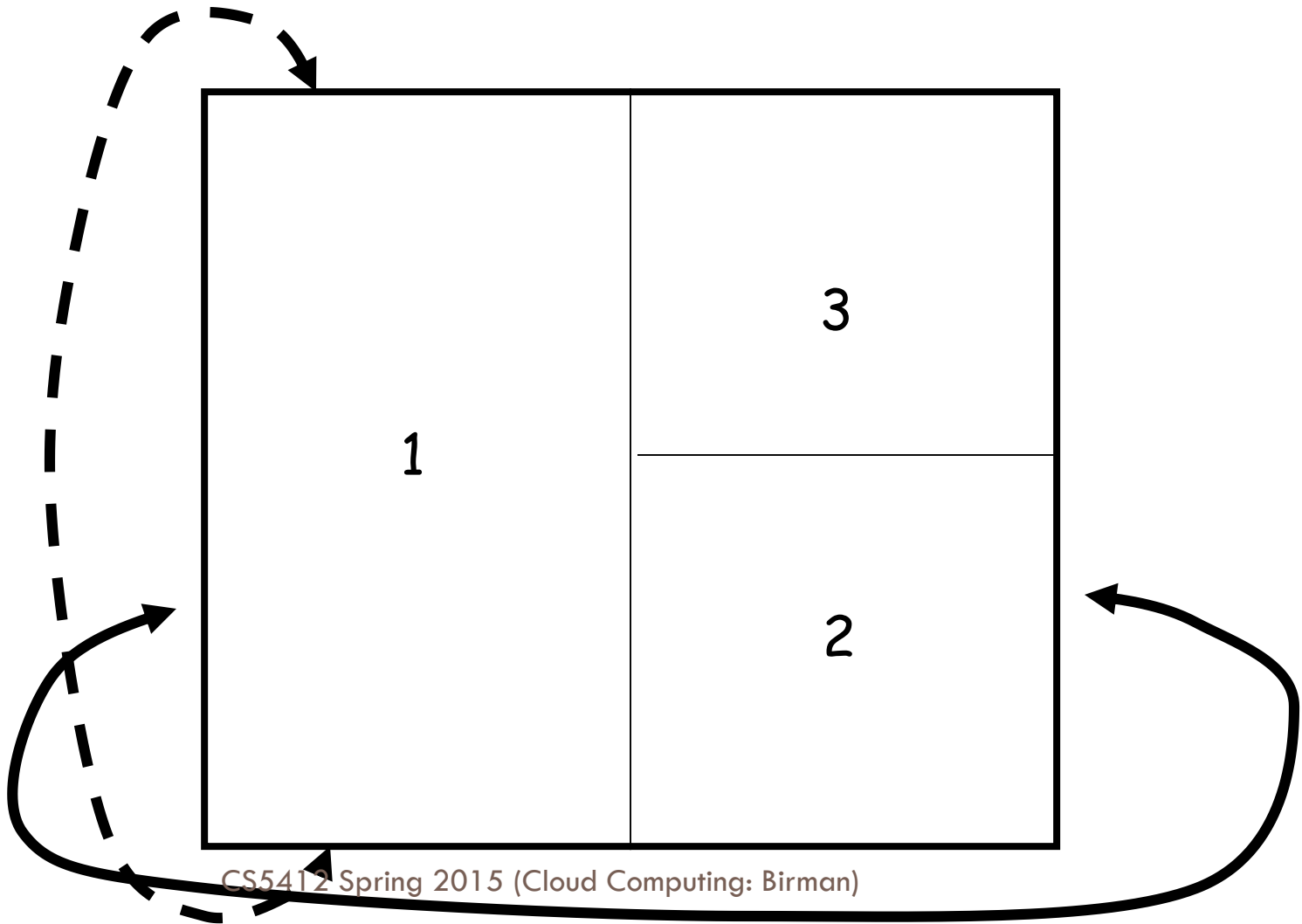
# Simple example in two dimensions

37



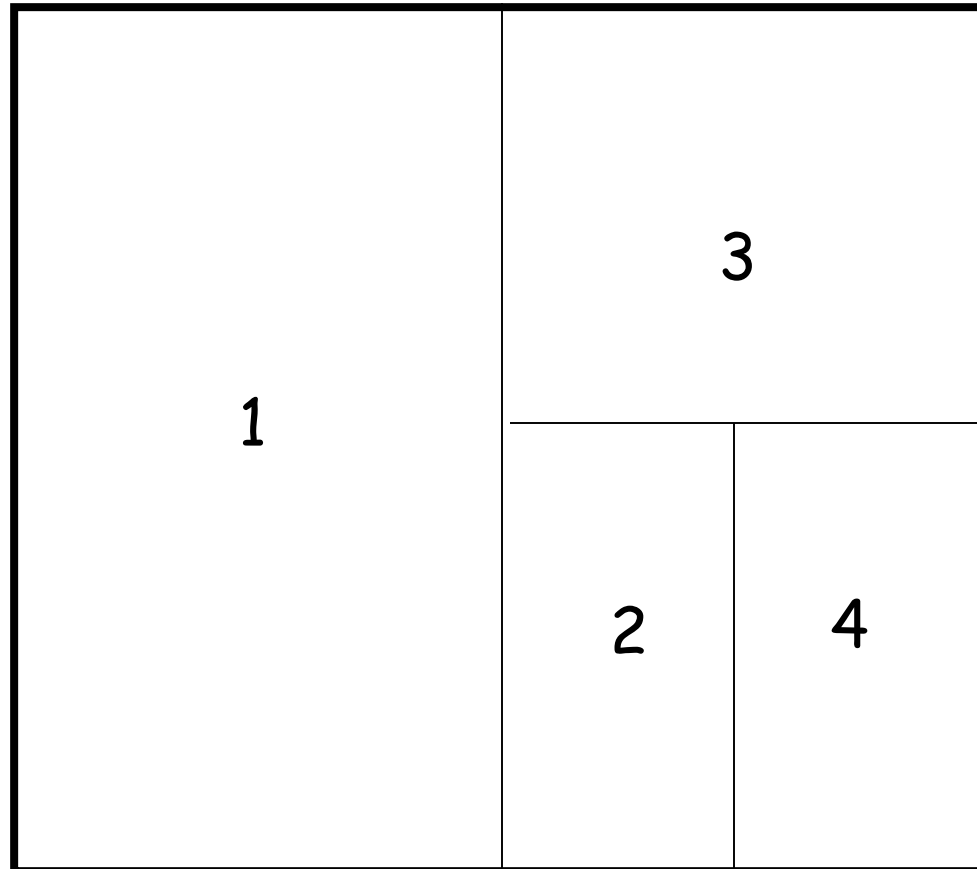
Note: torus wraps on “top” and “sides”

38



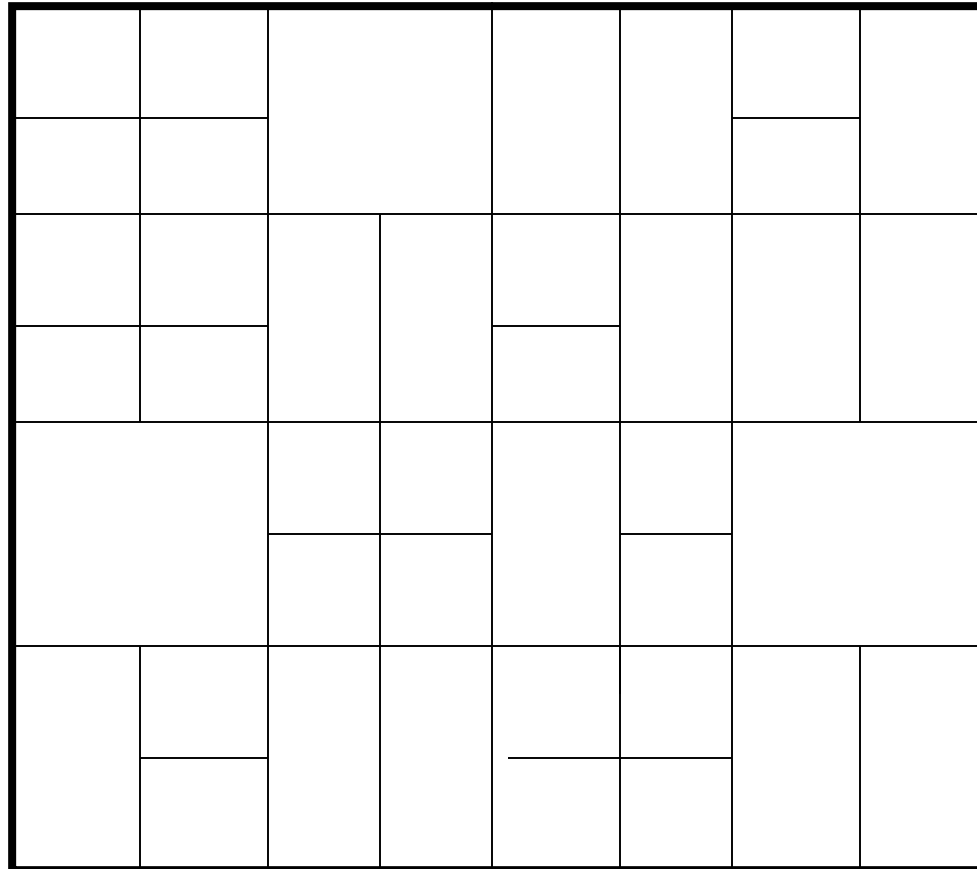
# Each node in CAN network occupies a “square” in the space

39



# With relatively uniform square sizes

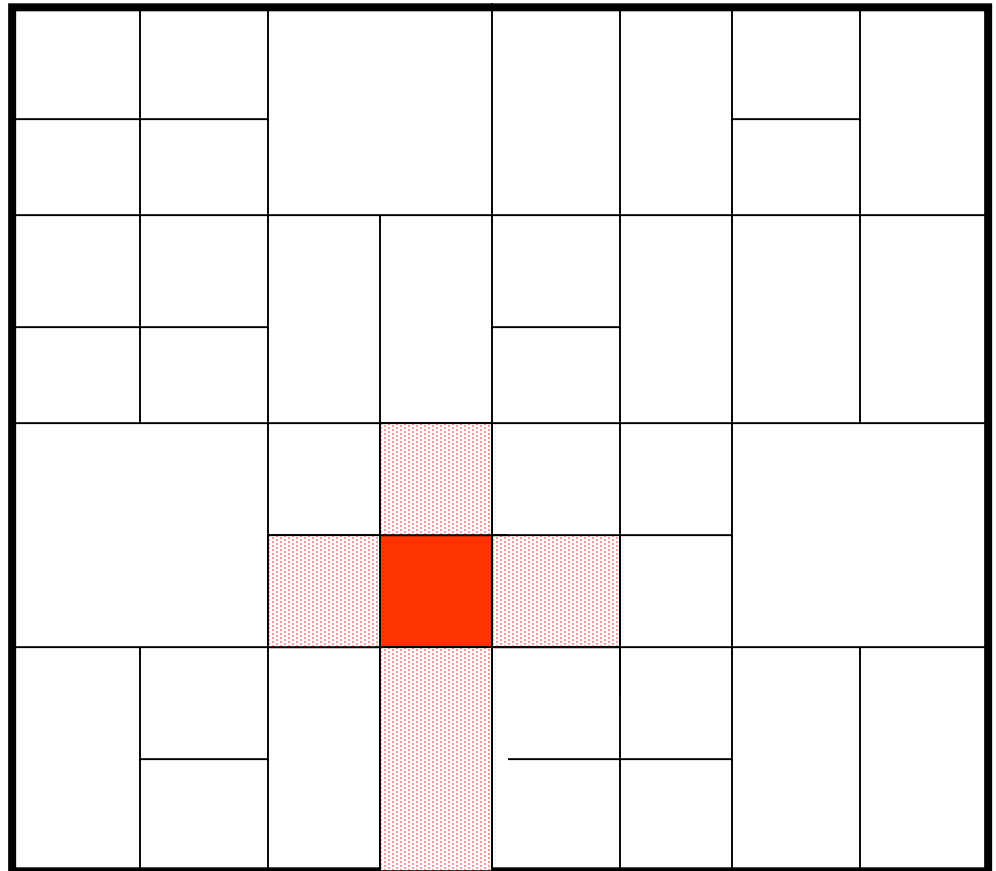
40





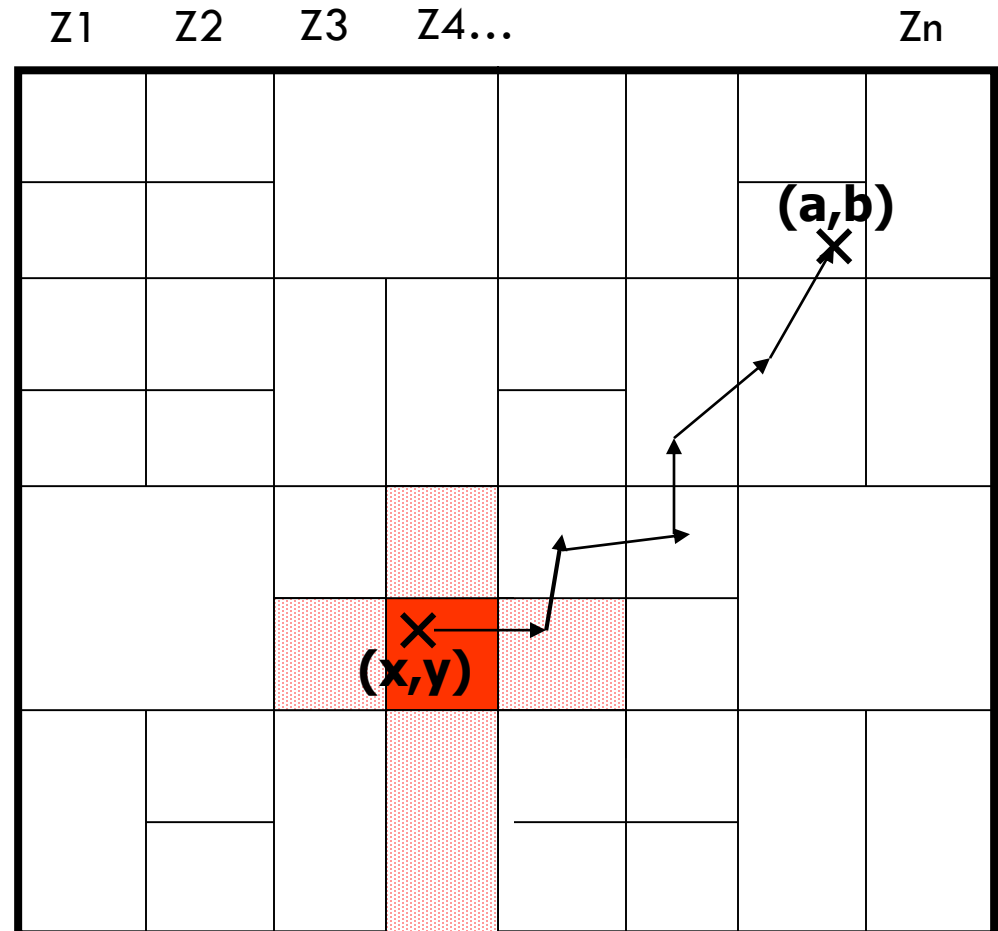
# Neighbors in CAN network

- Neighbor is a node that:
- Overlaps  $d-1$  dimensions
- Abuts along one dimension



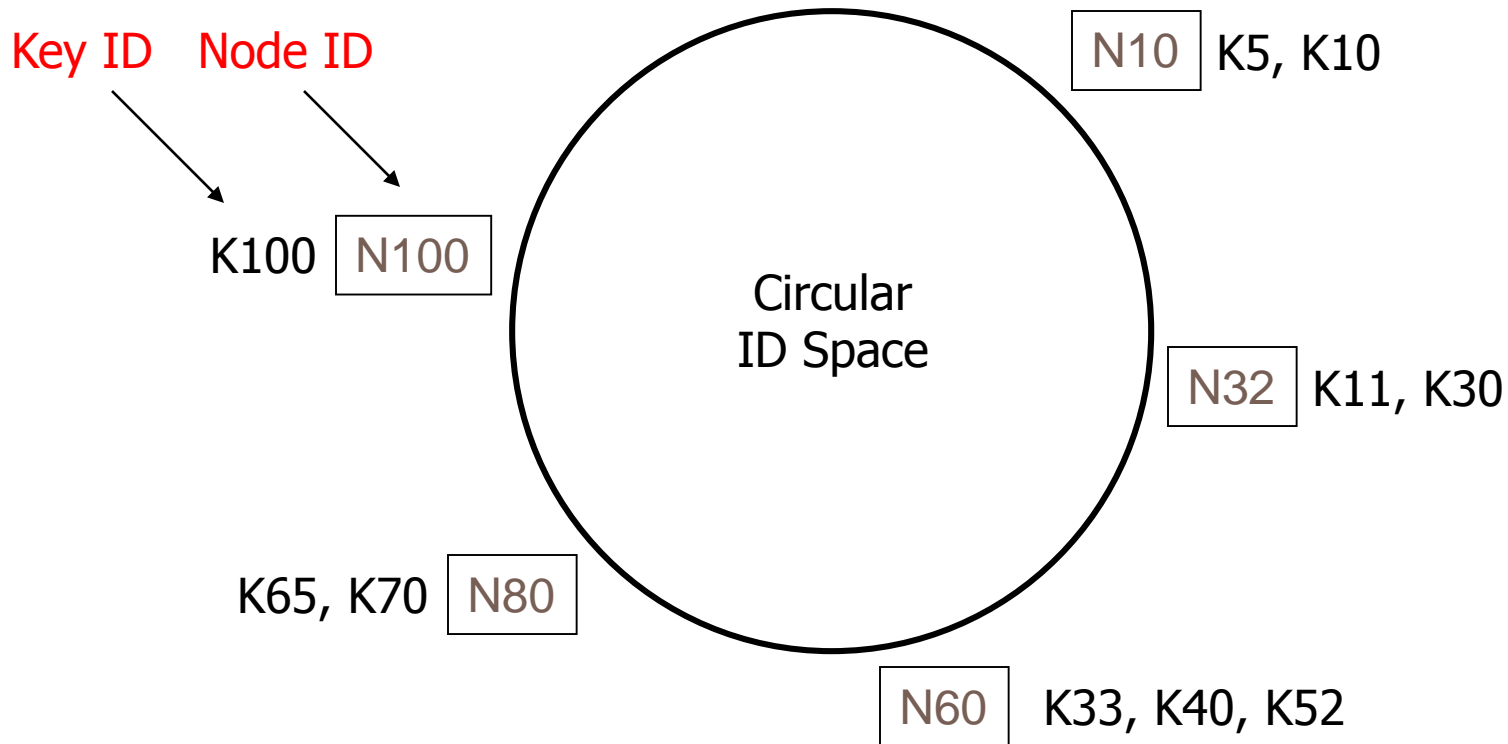
# Route to neighbors closer to target

- d-dimensional space
- n zones
  - ▣ Zone is space occupied by a “square” in one dimension
- Avg. route path length
  - ▣  $(d/4)(n^{1/d})$
- Number neighbors =  $O(d)$
- Tunable (vary d or n)
- Can factor proximity into route decision



# Chord uses a circular ID space

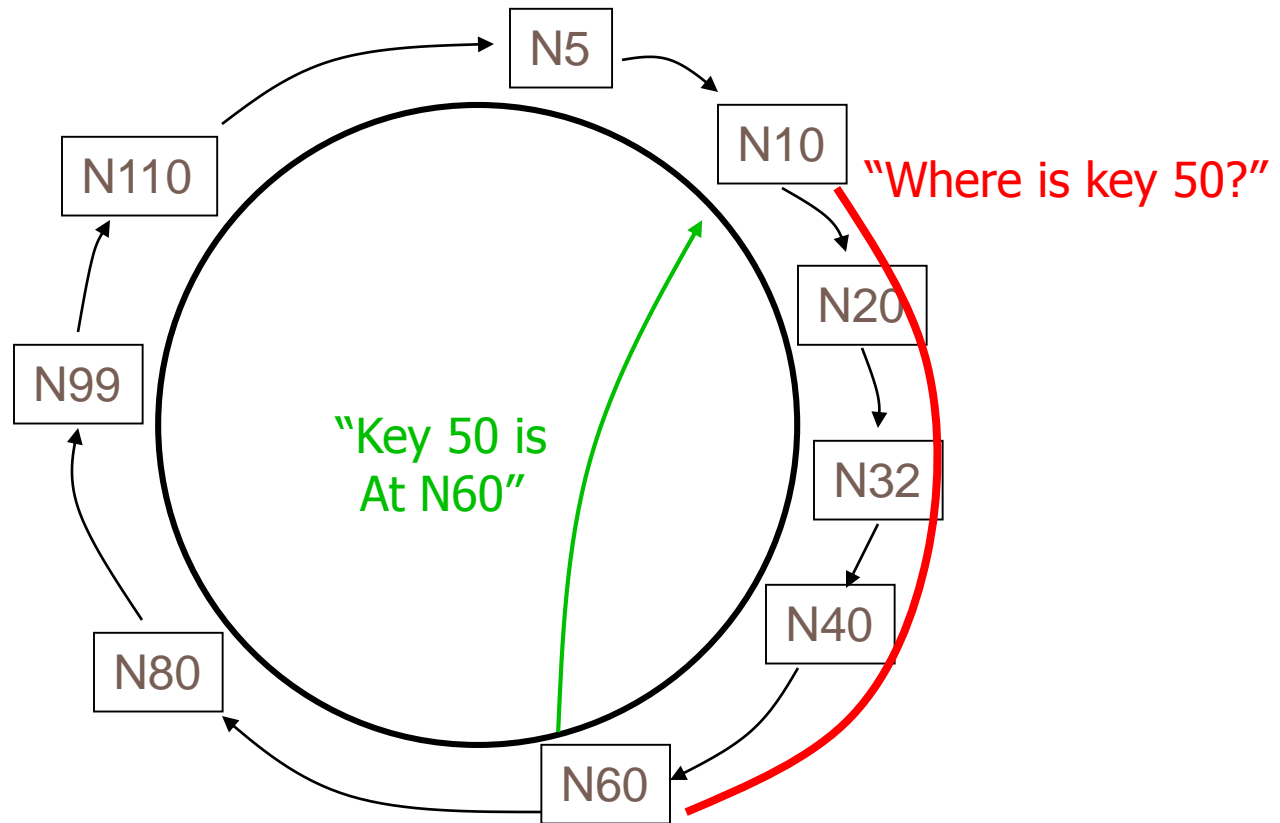
43



- **Successor: node with next highest ID**

# Basic Lookup

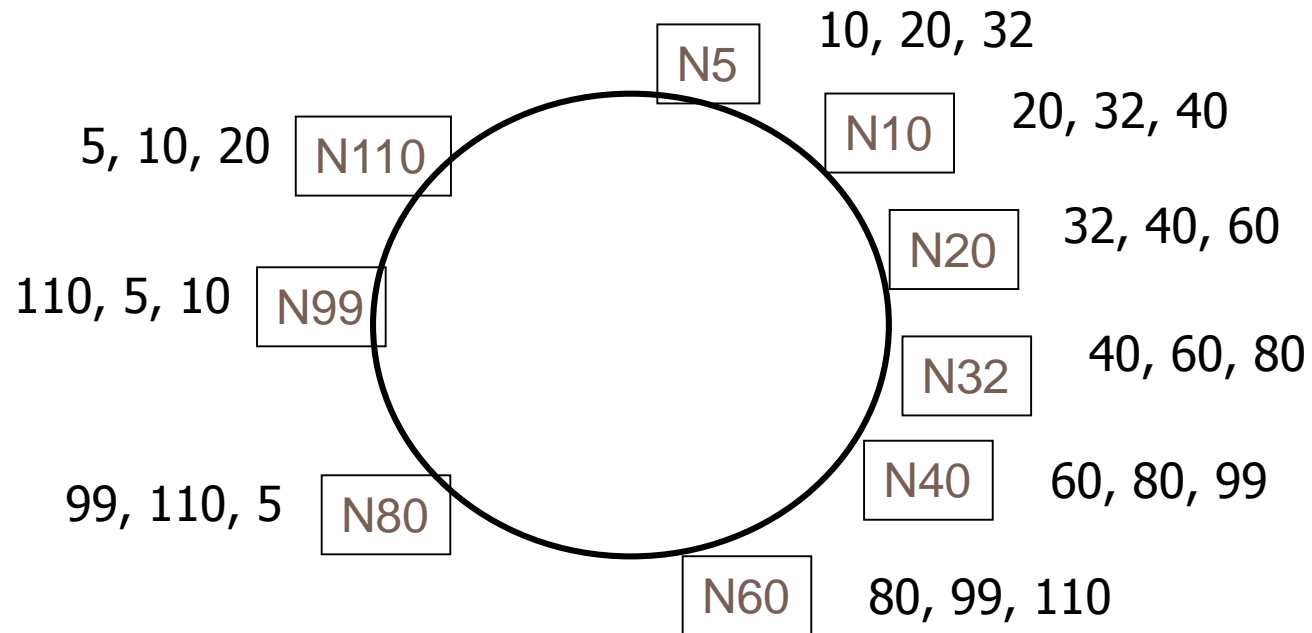
44



- Lookups find the ID's predecessor
- Correct if successors are correct

# Successor Lists Ensure Robust Lookup

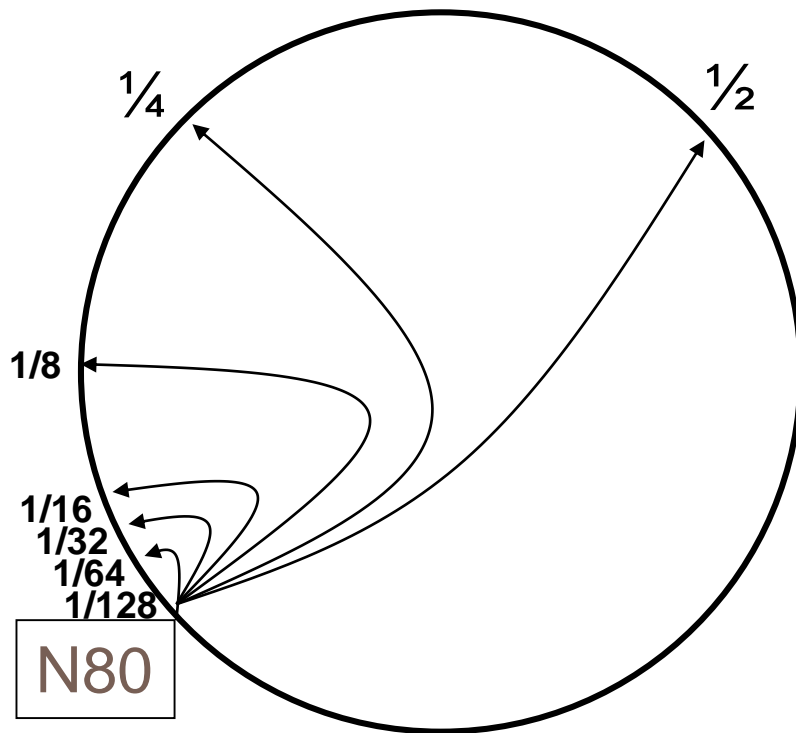
45



- Each node remembers  $r$  successors
- Lookup can skip over dead nodes to find blocks
- Periodic check of successor and predecessor links

# Chord “Finger Table” Accelerates Lookups

46

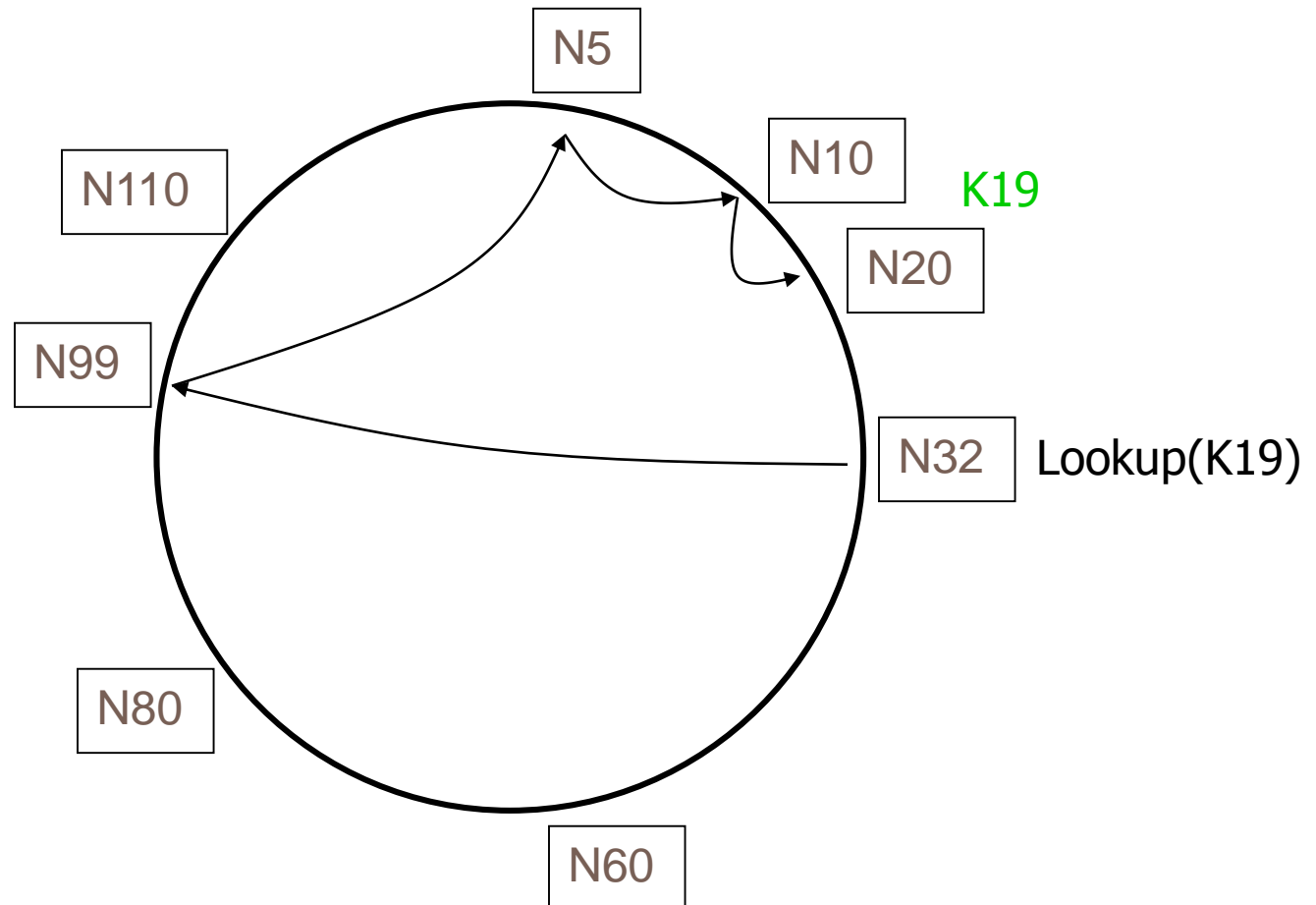


To build finger tables, new node searches for the key values for each finger

To do it efficiently, new nodes obtain successor's finger table, and use as a hint to optimize the search

# Chord lookups take $O(\log N)$ hops

47



# Drill down on Chord reliability

48

- Interested in maintaining a correct routing table (successors, predecessors, and fingers)
- Primary invariant: correctness of successor pointers
  - ▣ Fingers, while important for performance, do not have to be exactly correct for routing to work
  - ▣ Algorithm is to “get closer” to the target
  - ▣ Successor nodes always do this



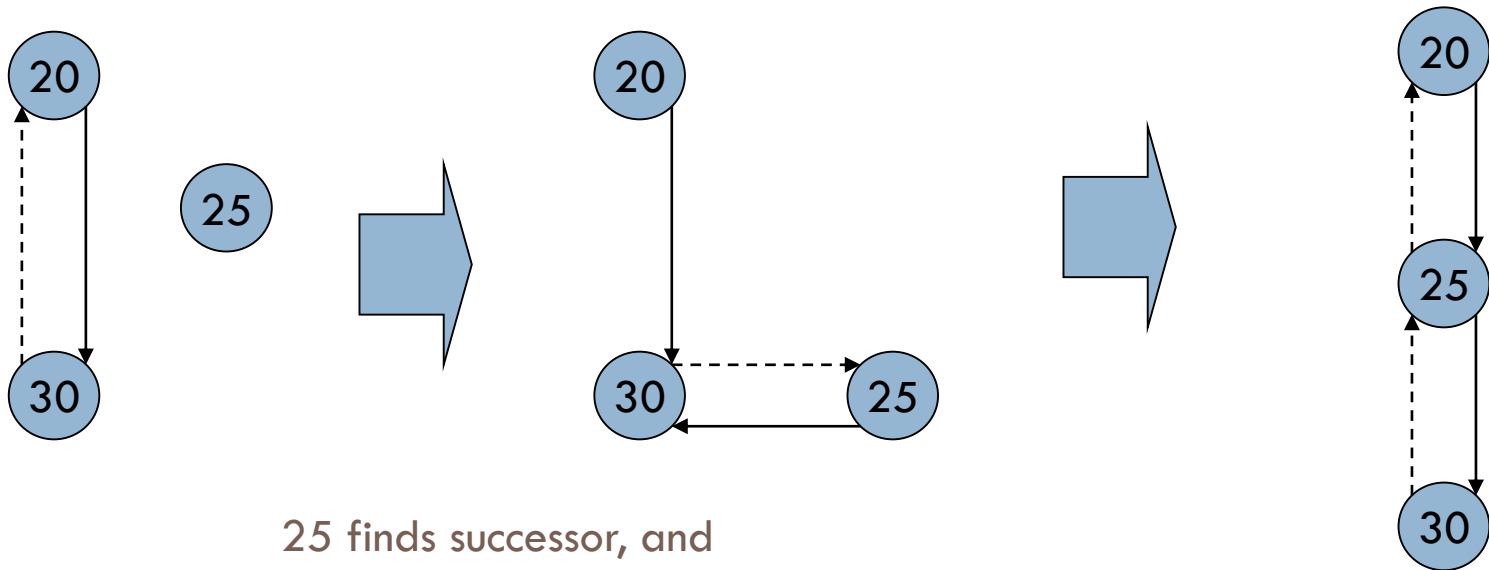
# Maintaining successor pointers

49

- Periodically run “stabilize” algorithm
  - ▣ Finds successor’s predecessor
  - ▣ Repair if this isn’t self
- This algorithm is also run at join
- Eventually routing will repair itself
- Fix\_finger also periodically run
  - ▣ For randomly selected finger

# Initial: 25 wants to join correct ring (between 20 and 30)

50

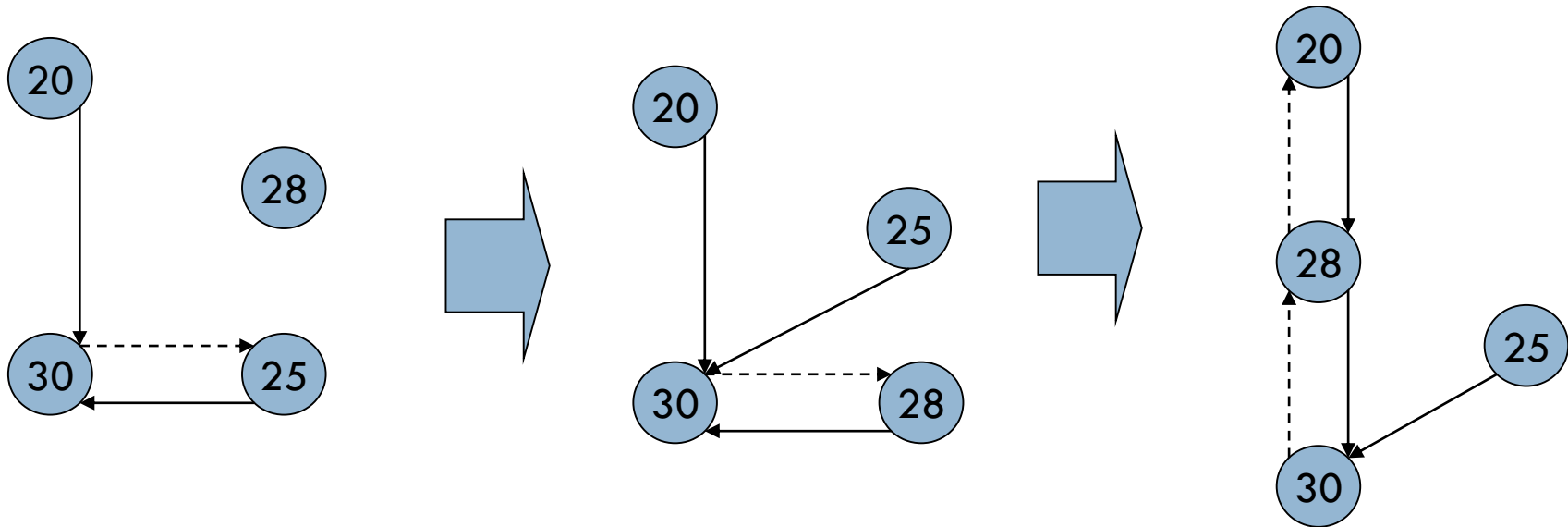


25 finds successor, and  
tells successor (30) of  
itself

20 runs "stabilize":  
20 asks 30 for 30's predecessor  
30 returns 25  
20 tells 25 of itself

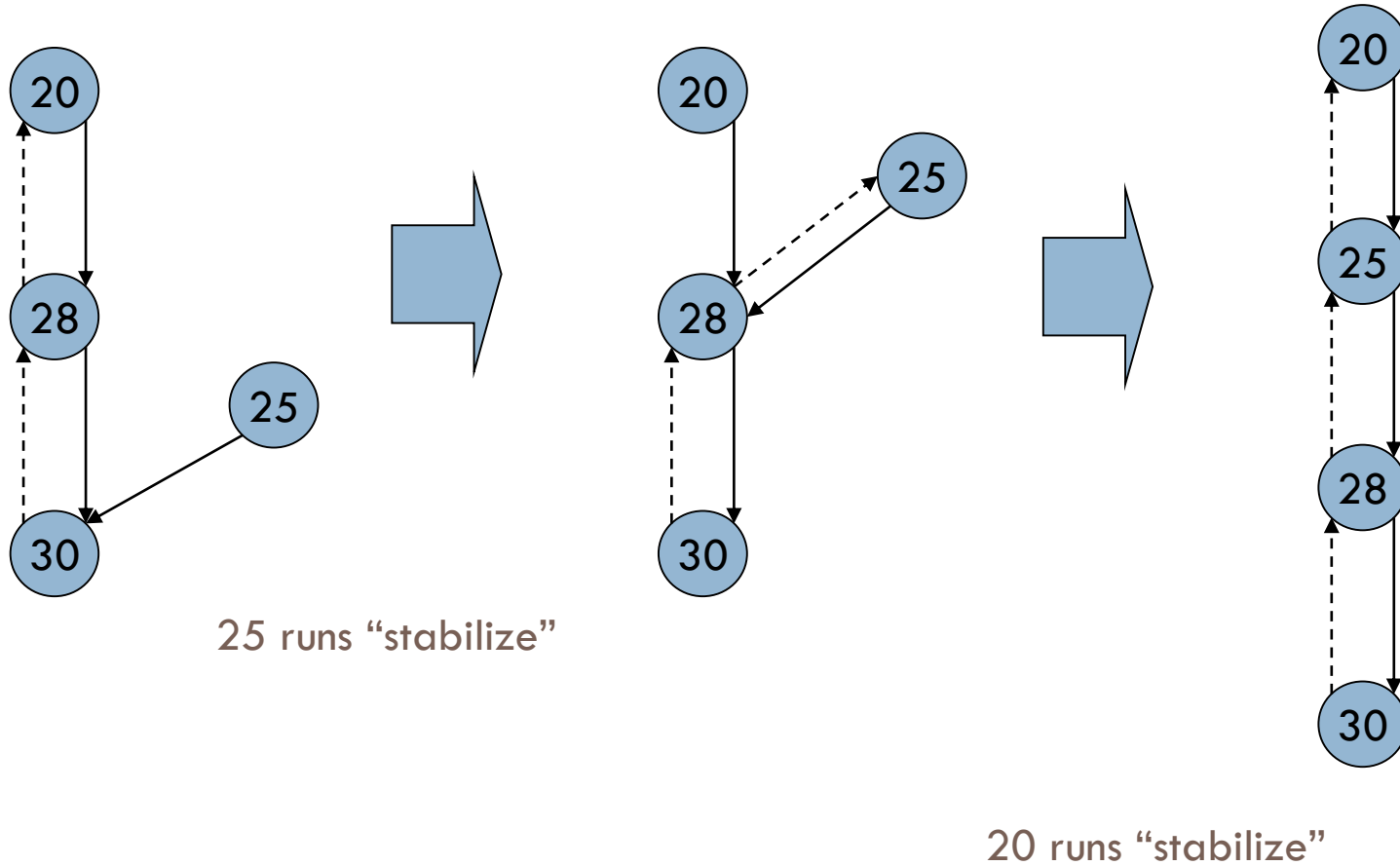
# This time, 28 joins before 20 runs “stabilize”

51



28 finds successor, and  
tells successor (30) of  
itself

20 runs “stabilize”:  
20 asks 30 for 30’s predecessor  
30 returns 28  
20 tells 28 of itself



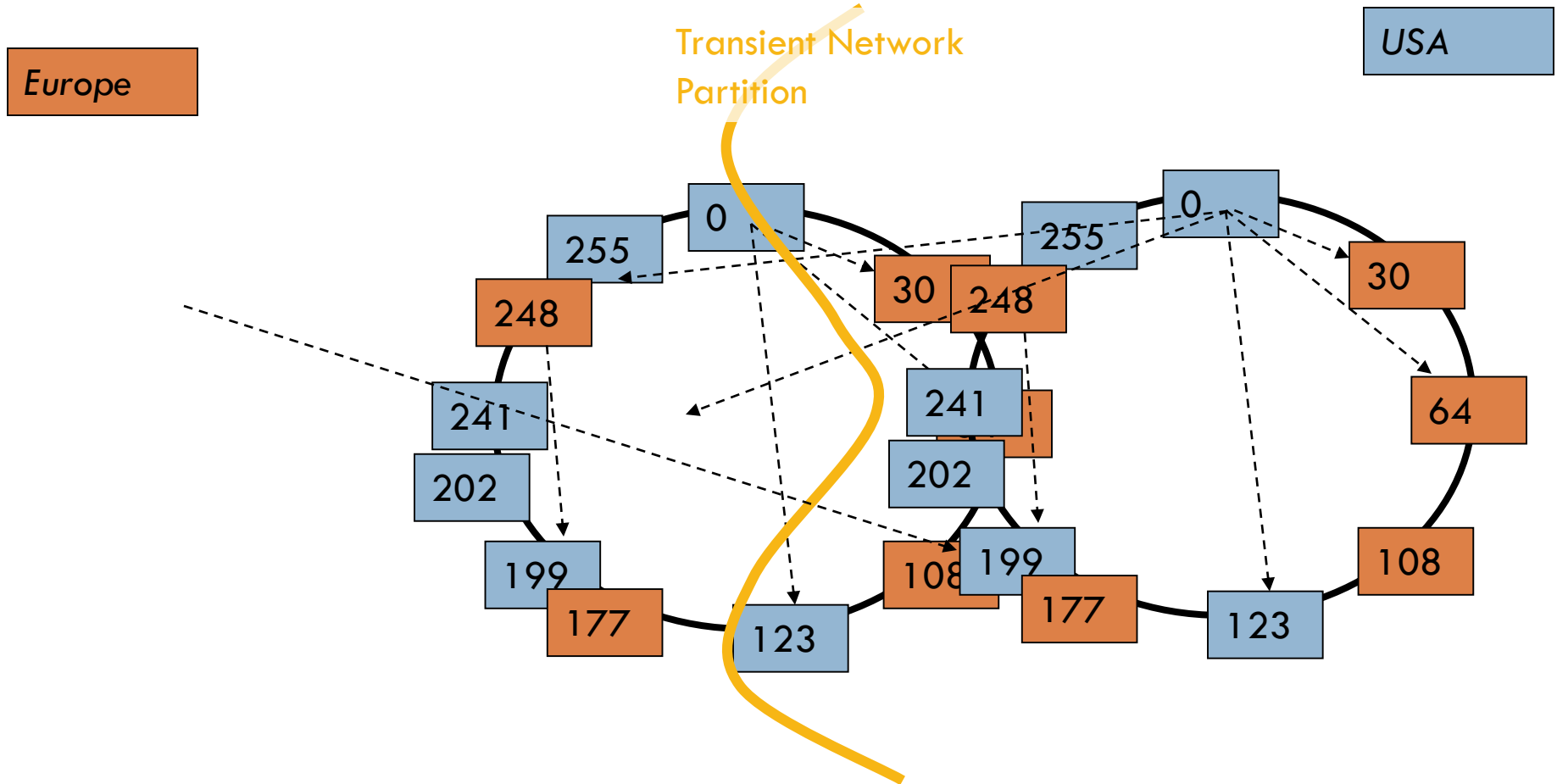
# Chord summary

53

- Ring with a kind of binary-search
- Self-repairing and self-organizing
- Depends on having a “good” hash function; otherwise some nodes might end up with many (key,value) pairs and others with few of them

# Chord can malfunction if the network partitions...

54



# Chord has no sense of “integrity”

55

- The system doesn't know it should be a ring... so it won't detect that it isn't a ring!
- MIT solution is to make this very unlikely using various tricks, and they work
- But an attacker might be able to force Chord into a partitioned state and if so, it would endure

# ... so, who cares?

56

- Chord lookups can fail... and it suffers from high overheads when nodes churn
  - ▣ Loads surge just when things are already disrupted... quite often, because of
  - ▣ And can't predict when Chord might remain disrupted once that way
- Worst case scenario: Chord can become inconsistent and stay that way

## The Fine Print

The scenario you have been shown is of low probability. In all likelihood, Chord would repair itself after any partitioning failure that might really arise. Caveat emptor and all that.



# More issues

57

- Suppose my machine has a (key,value) pair and your machine, right in this room, needs it.
- Search could still take you to Zimbabwe, Lima, Moscow and Paris first!
- Chord paths lack “locality” hence can be very long, and failures that occur, if any, will disrupt the system

# Impact?

58

- Other researchers began to look at Chord and ask if they could design similar structures that
  - ▣ Implement the DHT interface
  - ▣ But have better locality and are better at self-healing after disruptive events
- We'll examine some of them in the next lecture