CSC 525: Computer Networks

Sharing Information on the Internet

- The Internet contains a vast collection of information: documents, web pages, media, etc.

- One goal of the Internet is to make it easy to share this information.

- There are many different ways to do this.

Client/Server Model

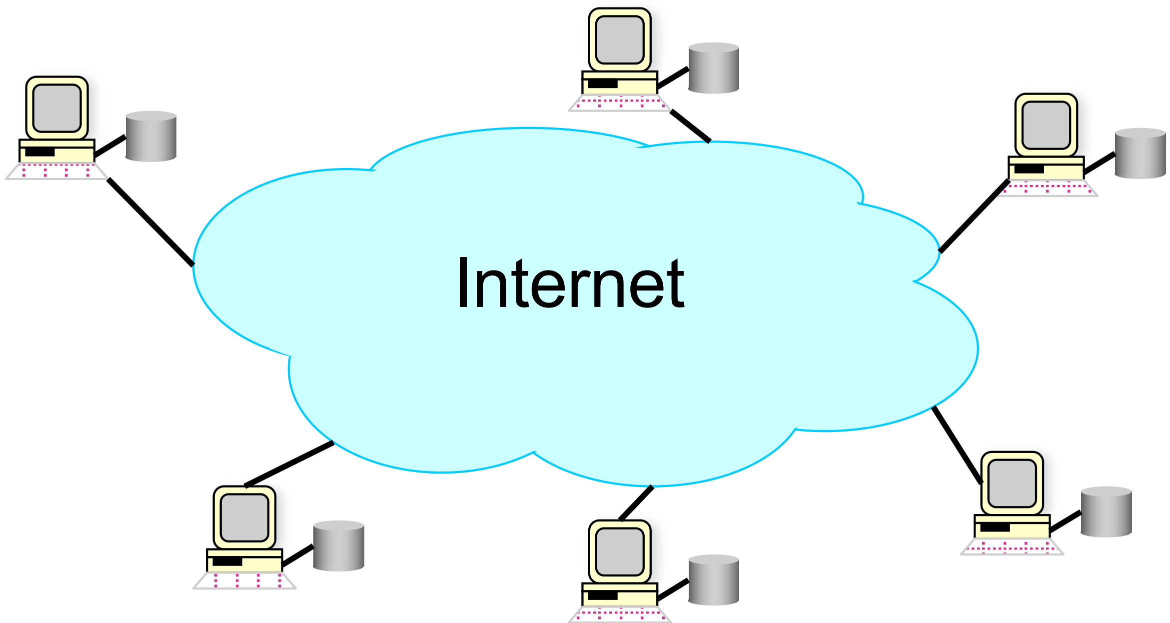
- In the beginning, there was FTP.
  - Need to know the server address and the file path.
- Then Web was invented
  - Follow hyperlinks
- Now we use Google
  - Search
  - The trend is that users don't need to know where the content is; they can focus on defining, identifying, and consuming the content.

Peer-to-Peer Model

- Not all contents are put on dedicated servers
  - Require technical expertise to set up and maintain
  - Need powerful machines as servers
  - Exposed to the world, attract traffic, …
- People want to share some contents, but don't want to commit to setting up and maintain a dedicated server.
- Or, we want all the participants to share the load (i.e., pool individual resource together to make the entire system works better).
- Peer-to-peer systems
  - A node can be a server and a client at the same time.

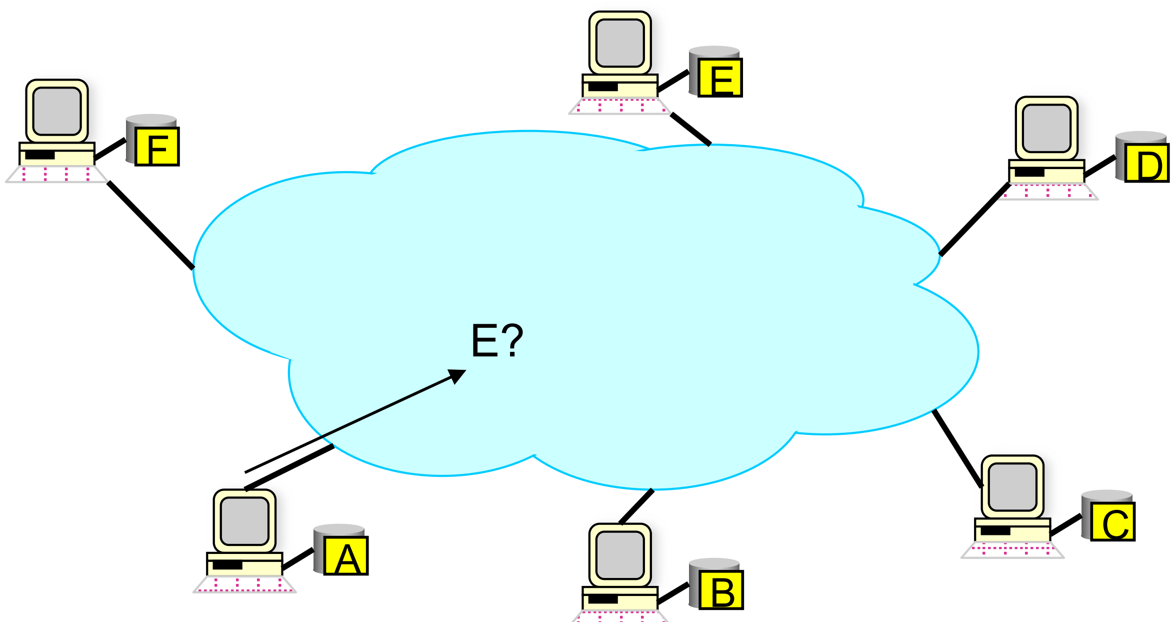
How Did It Start?

- A killer application: Naptser: Free music over the Internet
- Key idea: share the content, storage and bandwidth of individual users
- Each node stores a subset of files
- Each node can access/download files from any other node in the system



Main Challenge

- Where to find a particular file?
  - Always a tradeoff between possible locations and search difficulty



Other Challenges

- Scaling: up to hundreds of thousands or millions of machines
- Adaptability: machines can come and go at any time
- Security and privacy
- …

Napster

- A centralized directory system that maps files (songs) to machines that store the files.
  - Napster app uploads information of local files to Napster server.
- How to find a file (song)
  - Query the directory system → return the address of a machine that has the file
    - Ideally this is the closest/least-loaded machine
  - ftp the file
- Advantages:
  - Simple network operations
  - Easy to implement sophisticated search engines on top of the centralized index system
- Disadvantages:
  - Robustness, scalability, legal liability

Gnutella

- Distributed file location, no single server that knows all files.
- Idea: flood the search request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively send the request to their neighbors
  - Eventually a machine that has the file receives the request, and it sends back the answer
- Which neighbors to connect (at overlay):
  - Initially randomly, may improve later on.
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped by request messages (to alleviate this problem, each request has a TTL)
  - Especially hard on nodes with slow links.

Kazza

- Modify Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for standard nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report their lists of files
  - Allow slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search messages across supernodes

Distributed Hash Table (DHT)

- Abstraction: a hash-table data structure implemented over a distributed system.
  - insert(key, value);
  - value = query(key) or lookup(key)
  - key can be a file name, the title of a song, etc.
  - value can be anything: a data object, file, IP address of the node that has the file, etc.
  - Only exact match, very limited expressiveness of queries. Applications are expected to build more functionality on top of DHT.
- Proposals
  - CAN, *Chord*, Pastry, Tapestry, and many others.

DHT Design Goals

- (key, value) pairs are stored distributedly at many hosts. Given a key, the DHT needs to locate which host stores the (key, value) pair.
- Make sure that an item is always found if it's in the DHT.
- Scales
  - Search takes O(logN) steps
- Handles rapid arrival and failure of nodes

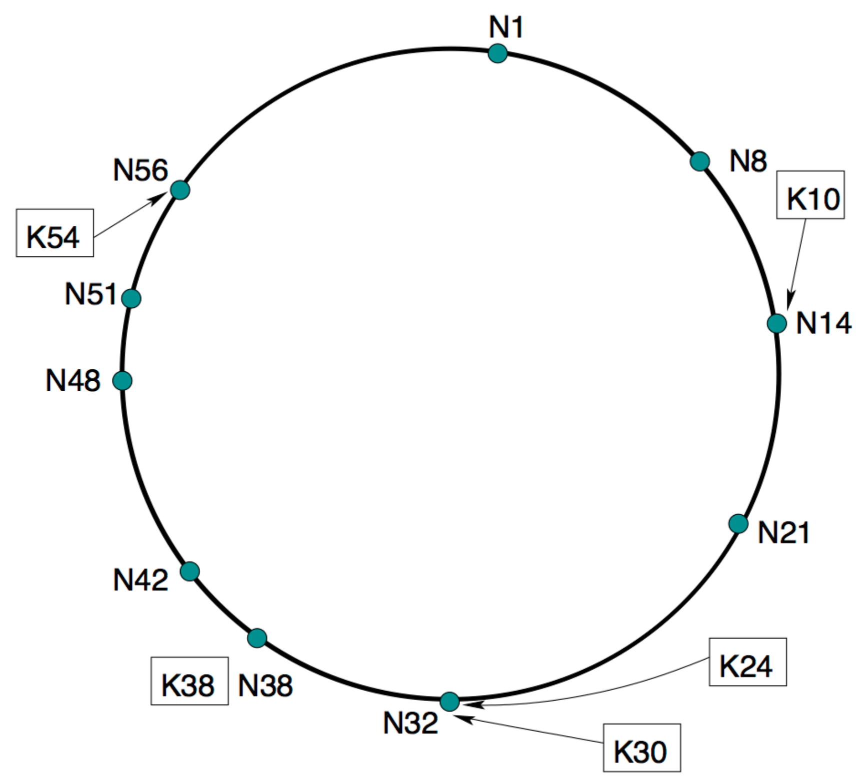
Chord

- Use a one-dimensional id space $0..2^m-1$
- Use the same hash function to hash keys to ids.
- Use the same hash function to hash nodes' IP addresses to ids.
- Organize the nodes into a structure to facilitate fast search of an id/key.

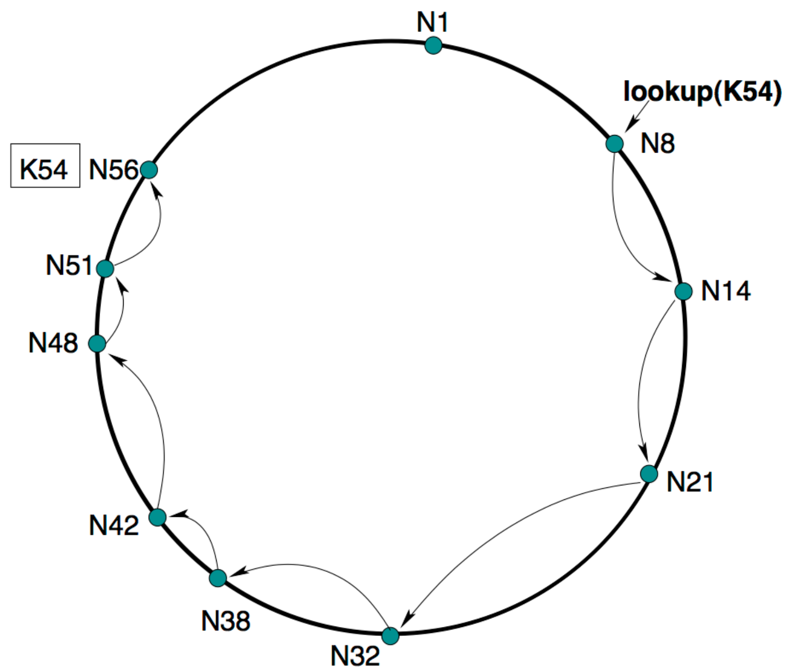
- Design decision
  - Decouple correctness from efficiency
- Properties
  - Routing table size O(log*N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log*N*) steps

The Ring

- Model the ID space as a ring.
  - Each node has an id = hash(ip).
- A node maintains its
  - Successor
  - Predecessor
  - Finger table
- *A key is stored at the node with next higher ID*.
  - E.g., a filename can be converted to a key = hash(filename)
  - Key 24 and 30 are stored in node 32.

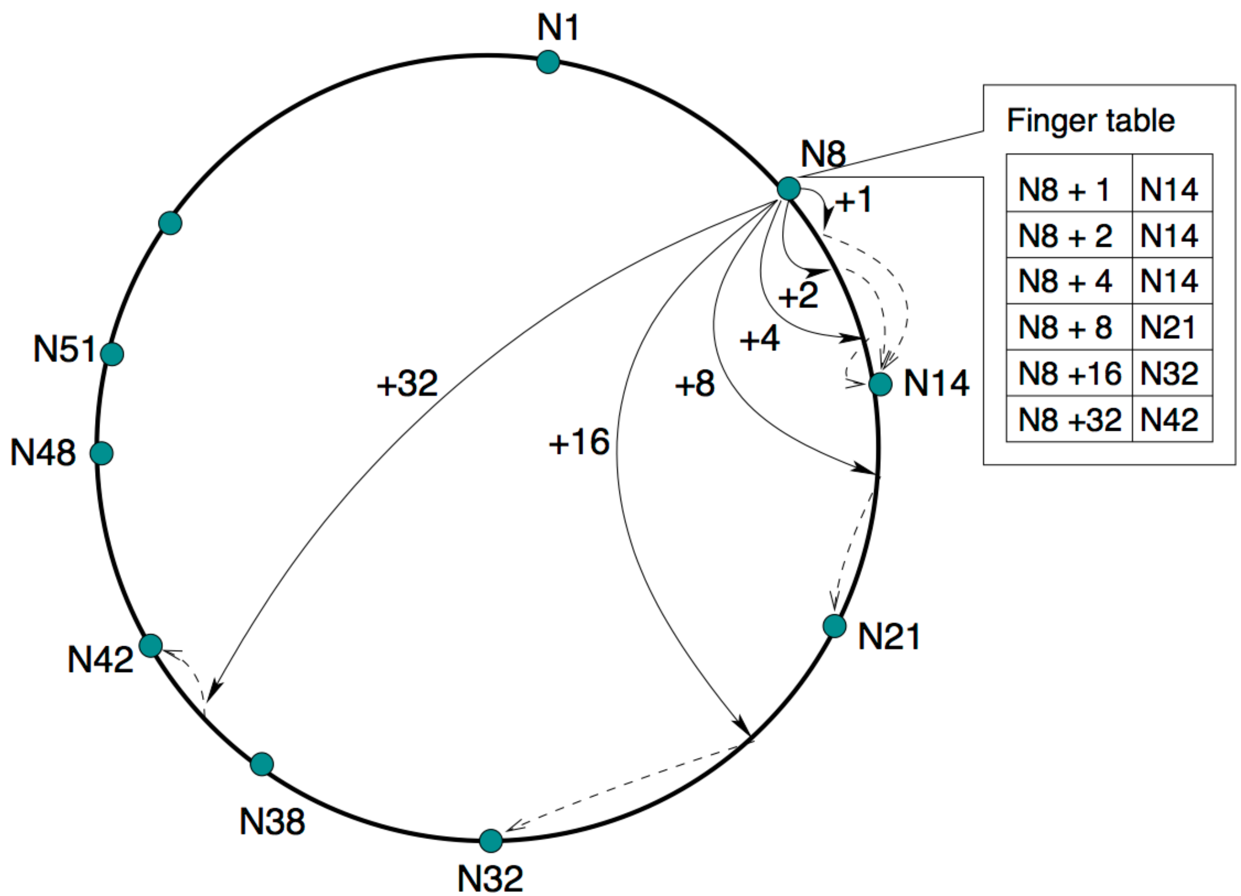


Basic Lookup



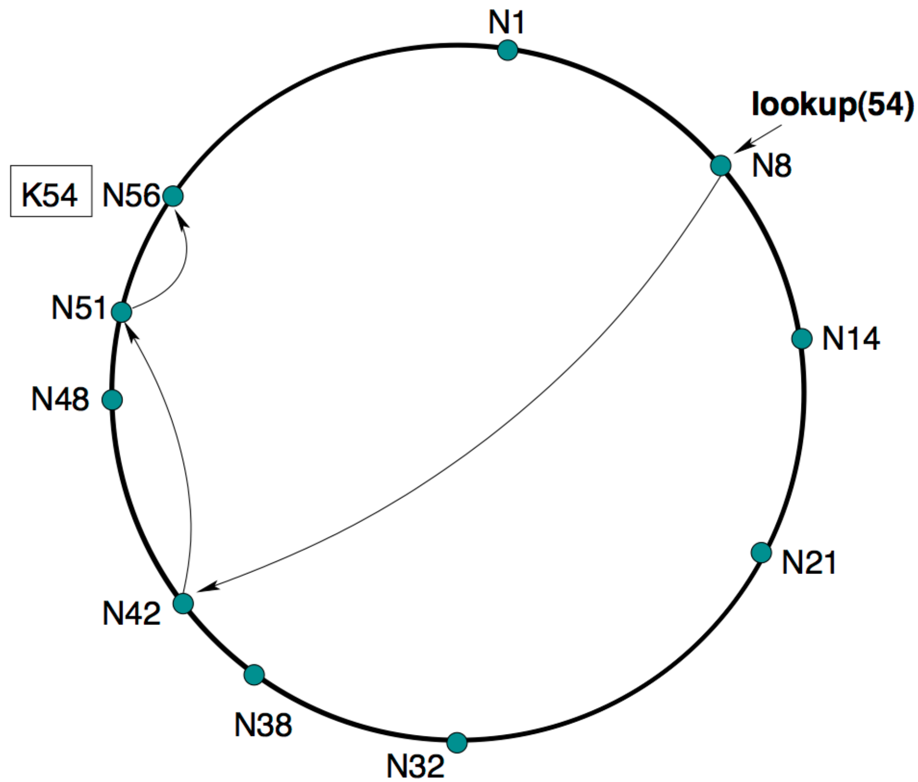
- Linear search
  - Search the ring clockwise one node at a step.
  - O(N) hops, with only one routing entry (successor) per node.
- Direct search
  - If every node maintains a list of every other nodes, the search can be done in 1 step, but the routing table size is N.
- Anything in the middle?

Finger Table



- Some hints to speed up the lookup
- Finger[k]: successor to (n+2^(k-1)) mod 2^m

Scalable Lookup



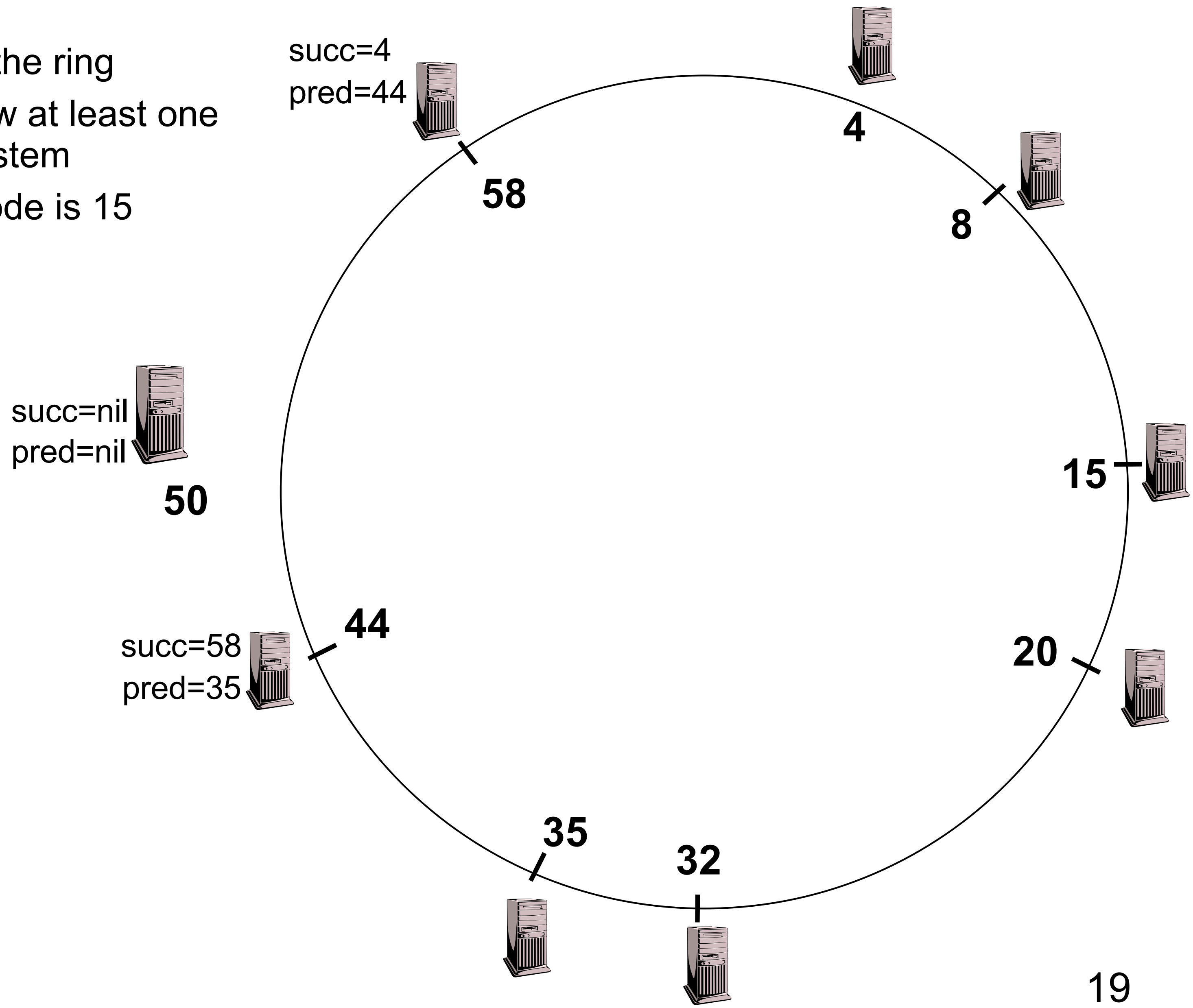
- Binary search using
  - Route to the node according to the finger table.
  - Each hop cuts the distance to target by half.
  - O(logN) hops with O(logN) routing table entries.

Join Operation

- Each node A periodically sends a stabilize() message to its successor B
- Upon receiving a stabilize() message, node B
  - returns its predecessor B'=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'
  - Otherwise (B' == A), A doesn't do anything.

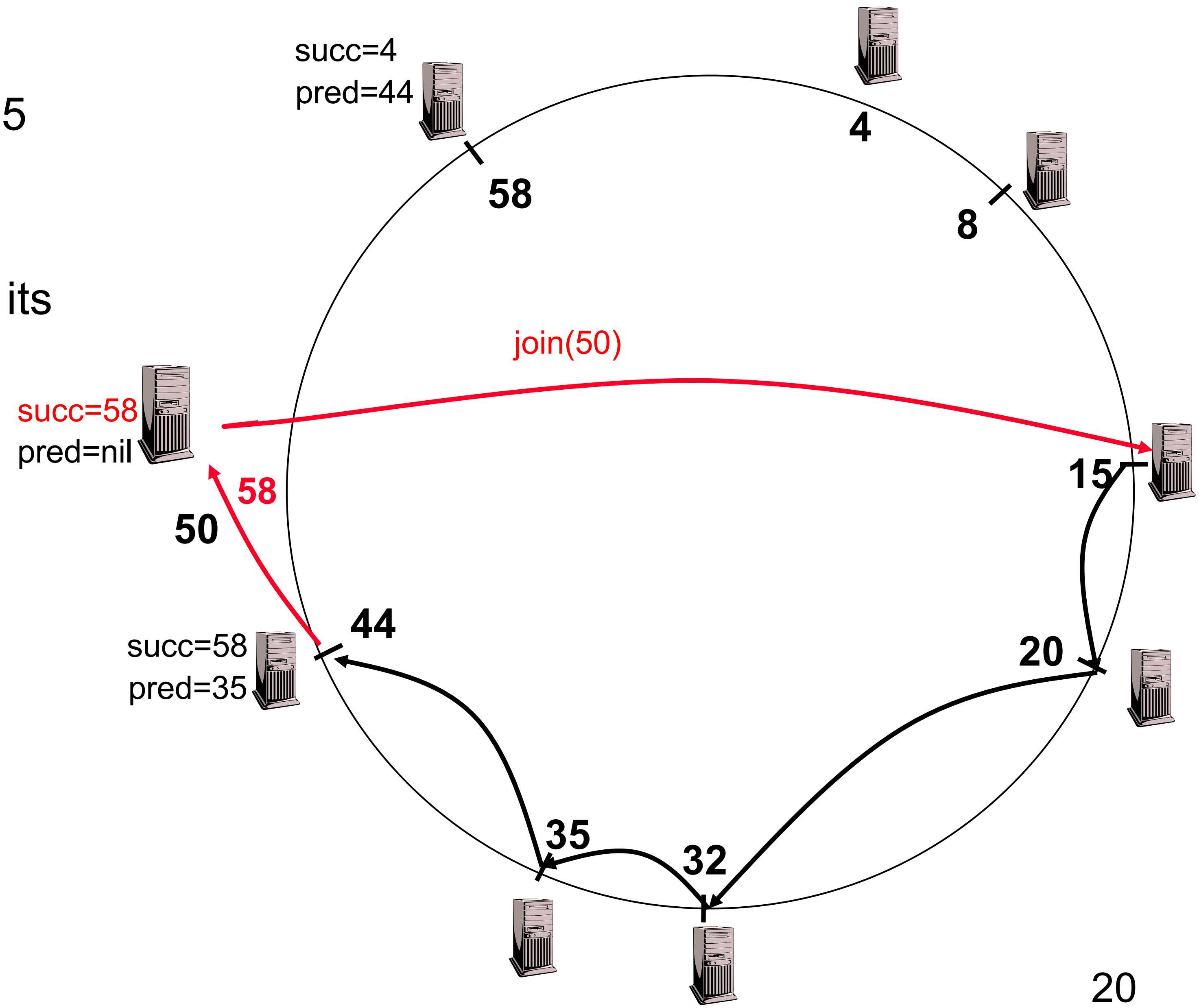
Joining Operation

- 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



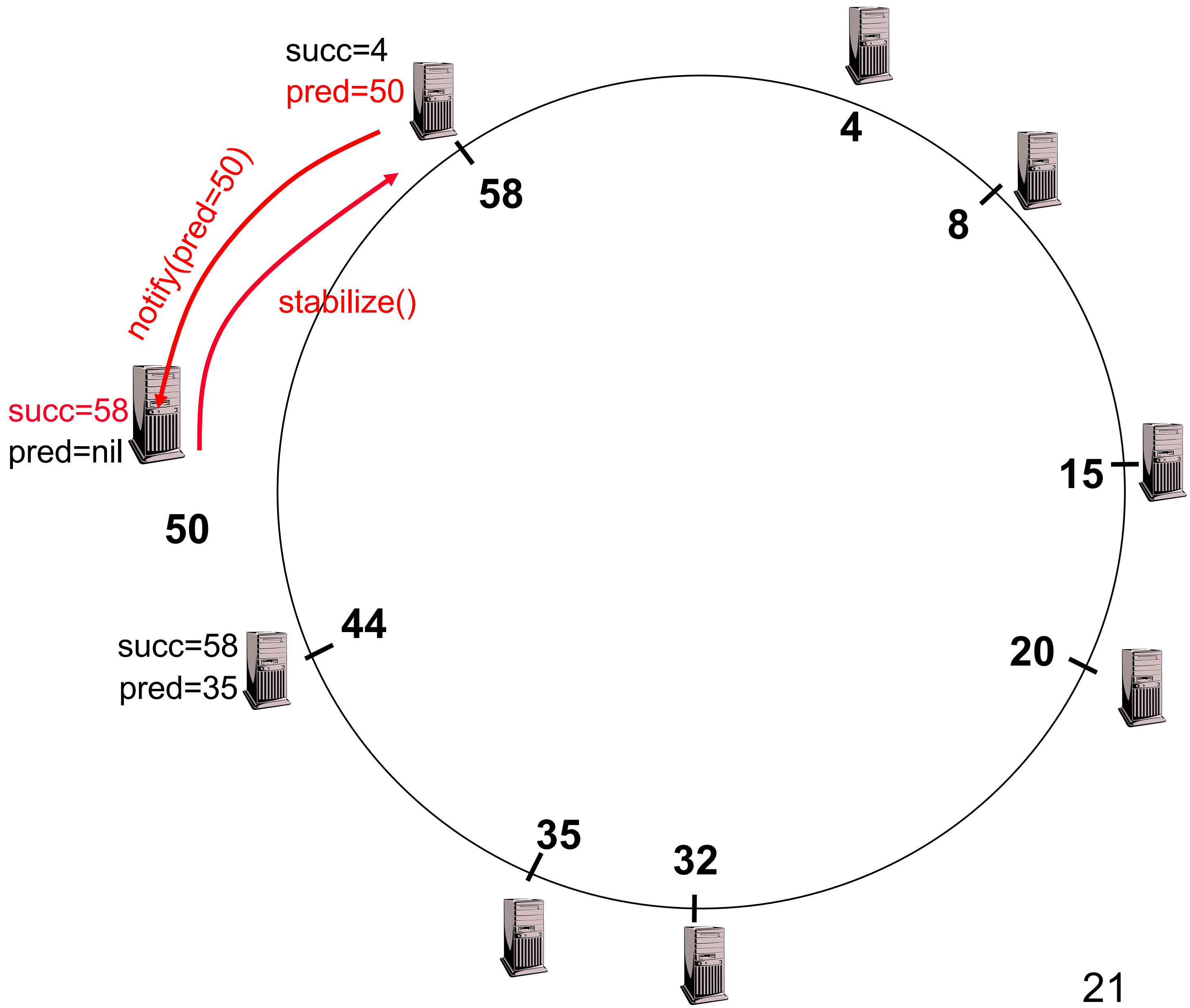
Joining Operation

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



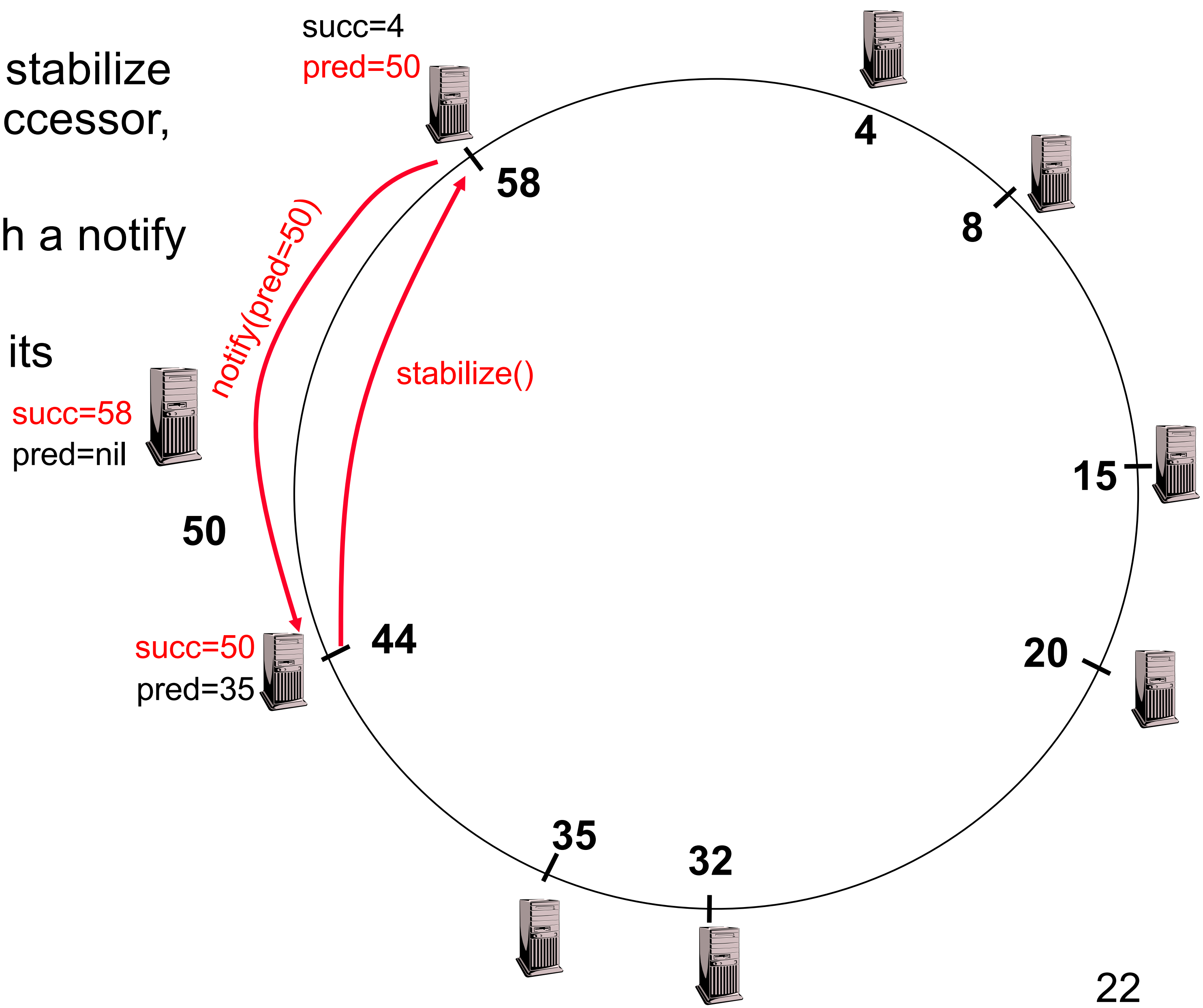
Joining Operation

- Node 50: send stabilize() to node 58
- Node 58:
  - update predecessor to 50
  - send notify() back



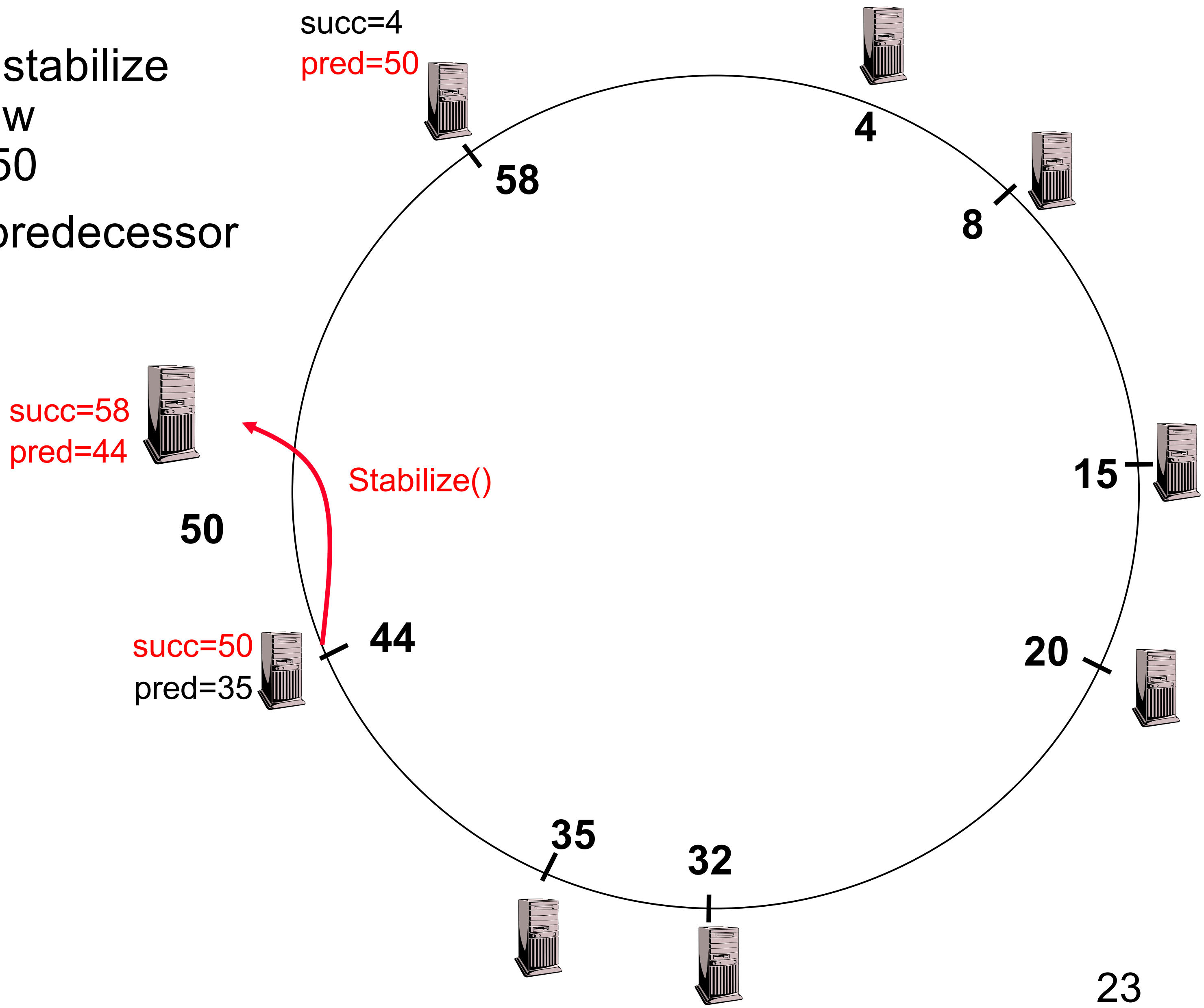
Joining Operation (cont' d)

- 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



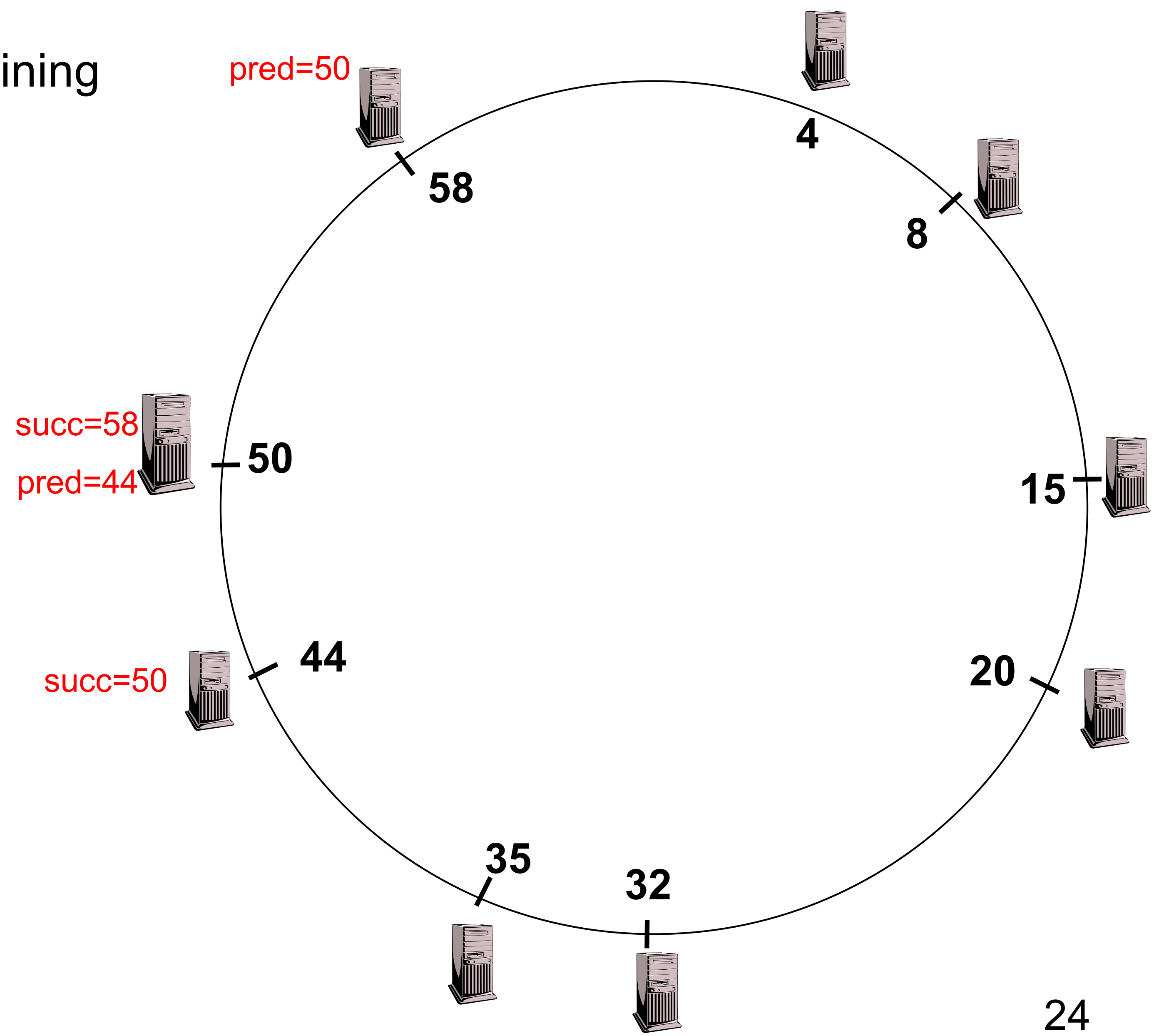
Joining Operation (cont’d)

- 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

Improve Performance

- Overlay hops ≠ Network 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
  - Better load balancing, but more complex.

Conclusions

- Distributed Hash Tables are a key component of scalable and robust overlay networks
  - Chord: O(log n) state, O(log n) distance
- A single function: find an item given a key (exact match)
- What can DHT do for us?
  - Distributed object lookup
  - Decentralized file system
  - Application layer multicast
  - Persistent storage
  - …