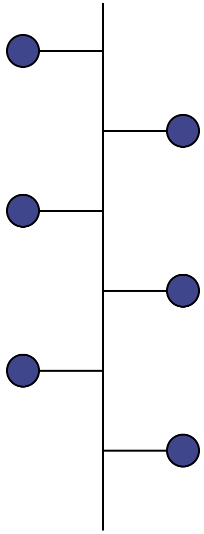
Introduction to Communication Networks and Distributed Systems



Unit 4a: DHTs

Prof. Dr.-Ing. Adam Wolisz

TKN Telecommunication Networks Group

Reminder:

- **We have discussed the mapping**

  Name ➔ *Address*

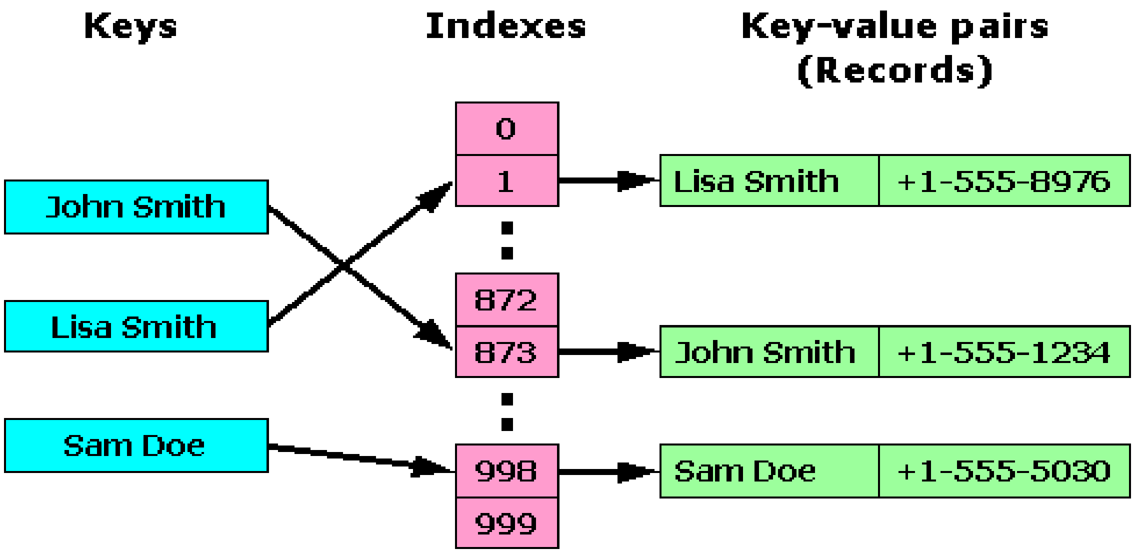
- **The solution has been DNS**
  - **Hierarchicall organization**
  - **Distributed**
  - **Using redundancy**

- **Challenge:**

  How to provide storage of pairs - like the above one – in a distributed way, even if there is no strong hierarchy?

Hash Tables (a classic)

- Items: [*Key*, *Value*] are stored
- The key is *hashed,* i.e. transformed (using a *hash function*) so that the result – ***the hash*** – can be used to locate a ***bucket*** in which the pair is stored. The bucket is identified by an ***index.***



In this example the index is simply the number of the record.

- The bucket might contain multiple such items (pairs)!

A Distributed Hash Table (DHT)

- Remember the mapping of NAMES to IP Addresses? Could we use Hash tables? Remember the scaling issue…
- Distributed Hash Tables spread the pairs across a number of computers (buckets) located arbitrarily across the world.

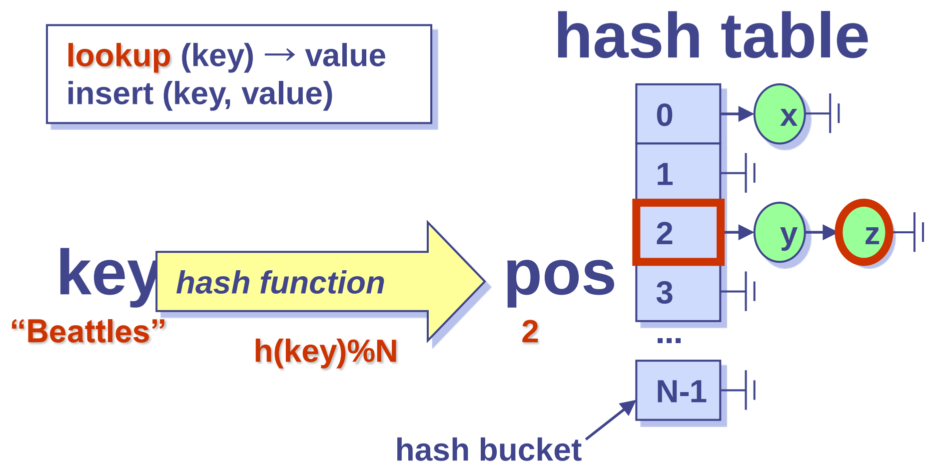
  Note: Copies of a single pair can be stored in one or in multiple locations!

- When a user queries the system, i.e. provides the key, the system uses the hash to find the pair from one of the computers where it's stored and returns the result.
- All the nodes are assumed to be reachable by some kind of unicast communication.
- DHT posses the features of :
  scaling, robustness, self-organization.

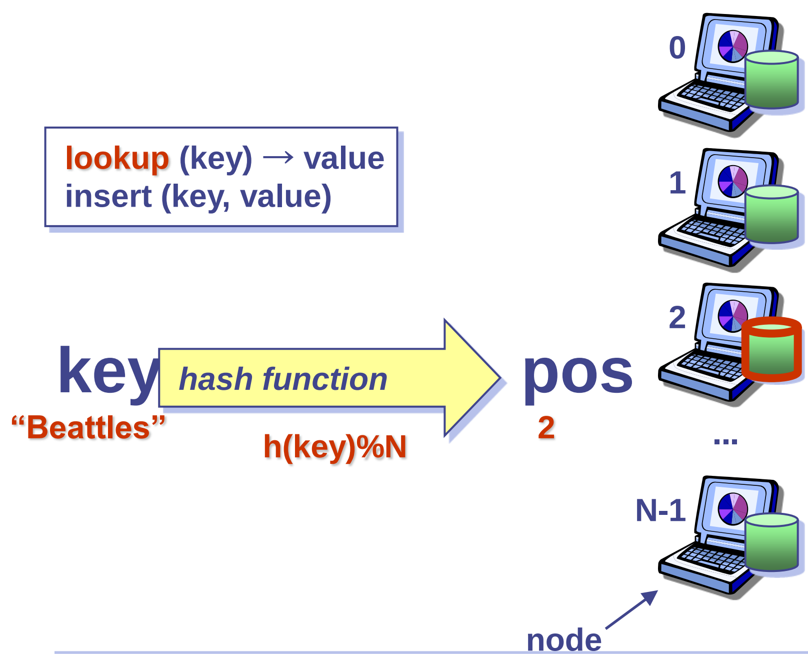
The hash table vs. DHT

[Ala Khalifeh, UCI]

- The key is hashed to find the proper bucket in a hash table



- In a Distributed Hash Table (DHT), nodes are the hash buckets
  - Key is hashed to find the responsible Node
  - Pairs are distributed among the nodes with respect to load balancing



DHT Interface

- Minimal interface (data-centric)

    Lookup(key) → value

    Insert(key, value)

    Delete (key)

- Supports a wide range of applications, because few restrictions
    - Value is application dependent
    - Keys have no semantic meaning

Note: *DHTs do not have to store data useful to end users,*
e.g. data files…
Data storage can be build on top of DHTs

DHTs: Problems

[Felber, Eurecom]

- Problem 1 (dynamicity): adding or removing nodes
  - With hash mod N, virtually every key will change its location!

$$h(k) \bmod m \neq h(k) \bmod (m+1) \neq h(k) \bmod (m-1)$$

- Solution: use consistent hashing
  - Define a fixed hash space
  - All hash values fall within that space and do not depend on the number of peers (hash bucket)
  - Each key goes to peer closest to its ID in hash space (according to some proximity metric)

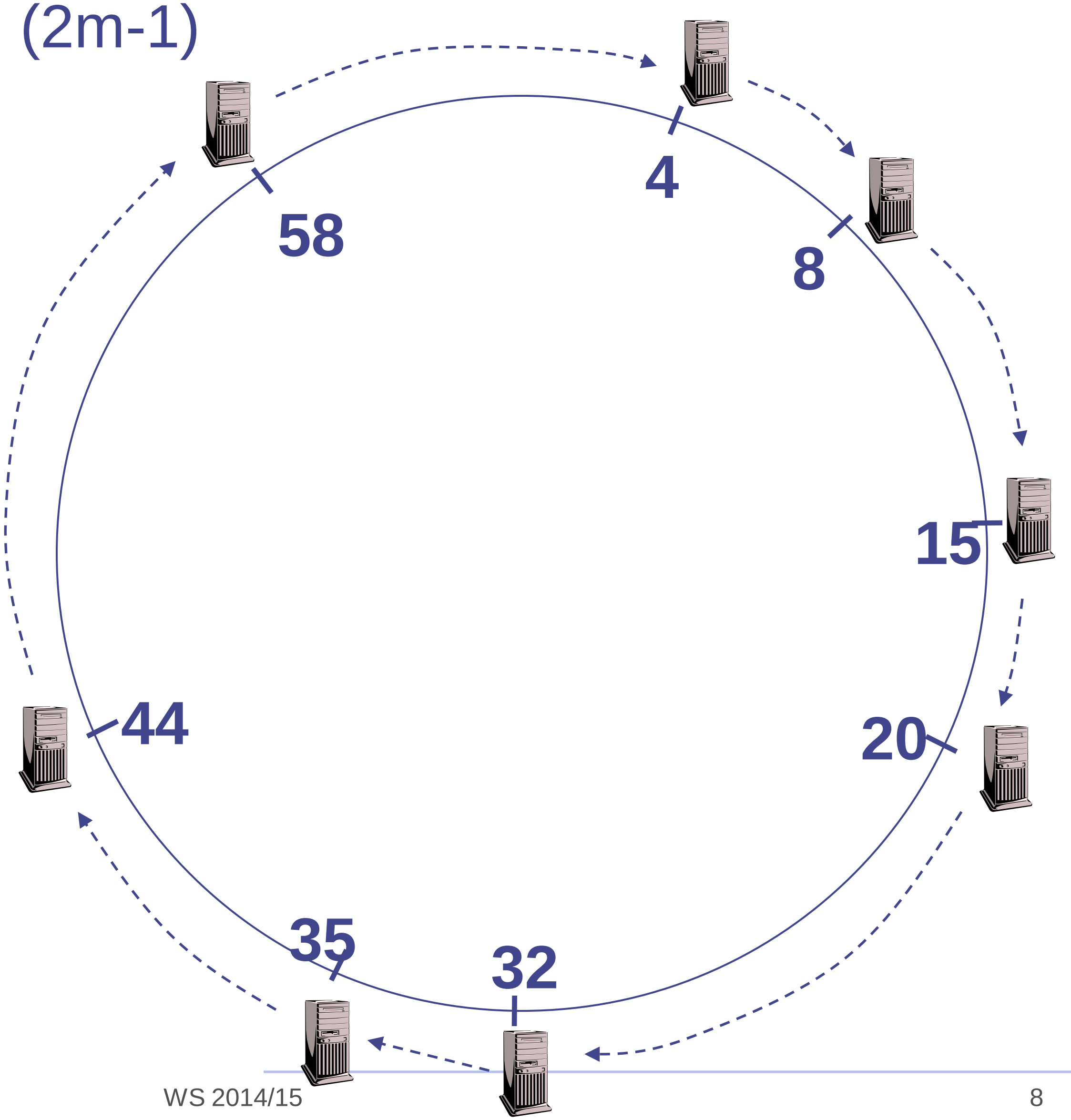
- Problem 2 (size): all nodes must be known (in order to insert or lookup items!)
  - Works with *small* and *static* server populations
- Solution: each peer knows of only **a few "neighbors**"
  - Messages are routed through neighbors via multiple hops

Identifier to Node Mapping Example [S.Shenker and I.Stoica, UCB]

Associate to each node and item a unique *id* in an *uni*-dimensional space 0.. (2m-1)

- Node 8 maps [5,8]
- Node 15 maps [9,15]
- Node 20 maps [16, 20]
- …
- Node 4 maps [59, 4]

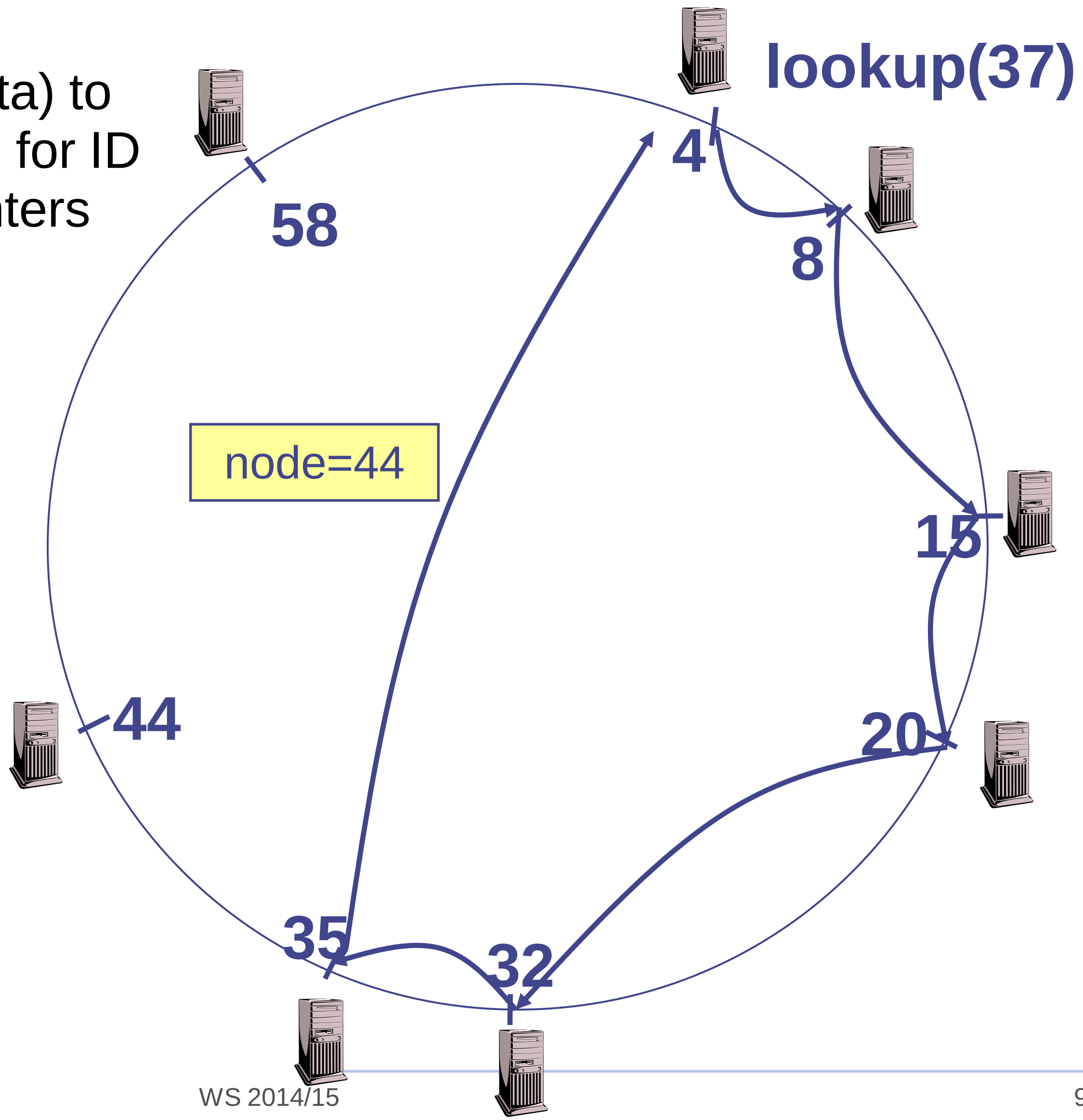
- Each node maintains a pointer to its successor



Chord Lookup

[Scott Shenker and Ion Stoica, UCB]

- Each node maintains its successor
- Route packet (ID, data) to the node responsible for ID using successor pointers



Chord Joining Operation

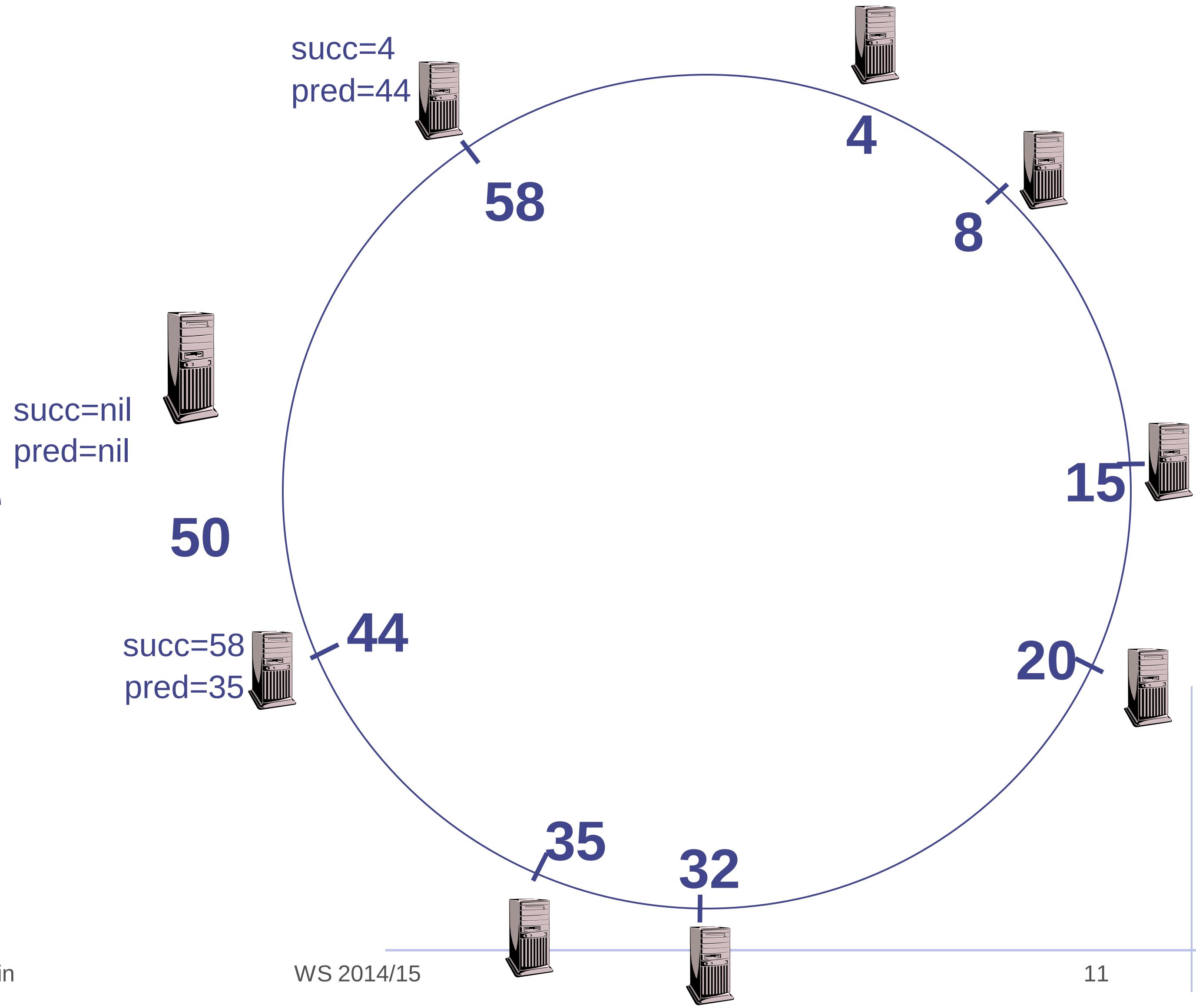
[Scott Shenker and Ion Stoica, UCB]

- 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'
  - A doesn't do anything, otherwise

Joining Operation

[Scott Shenker and Ion Stoica, UCB]

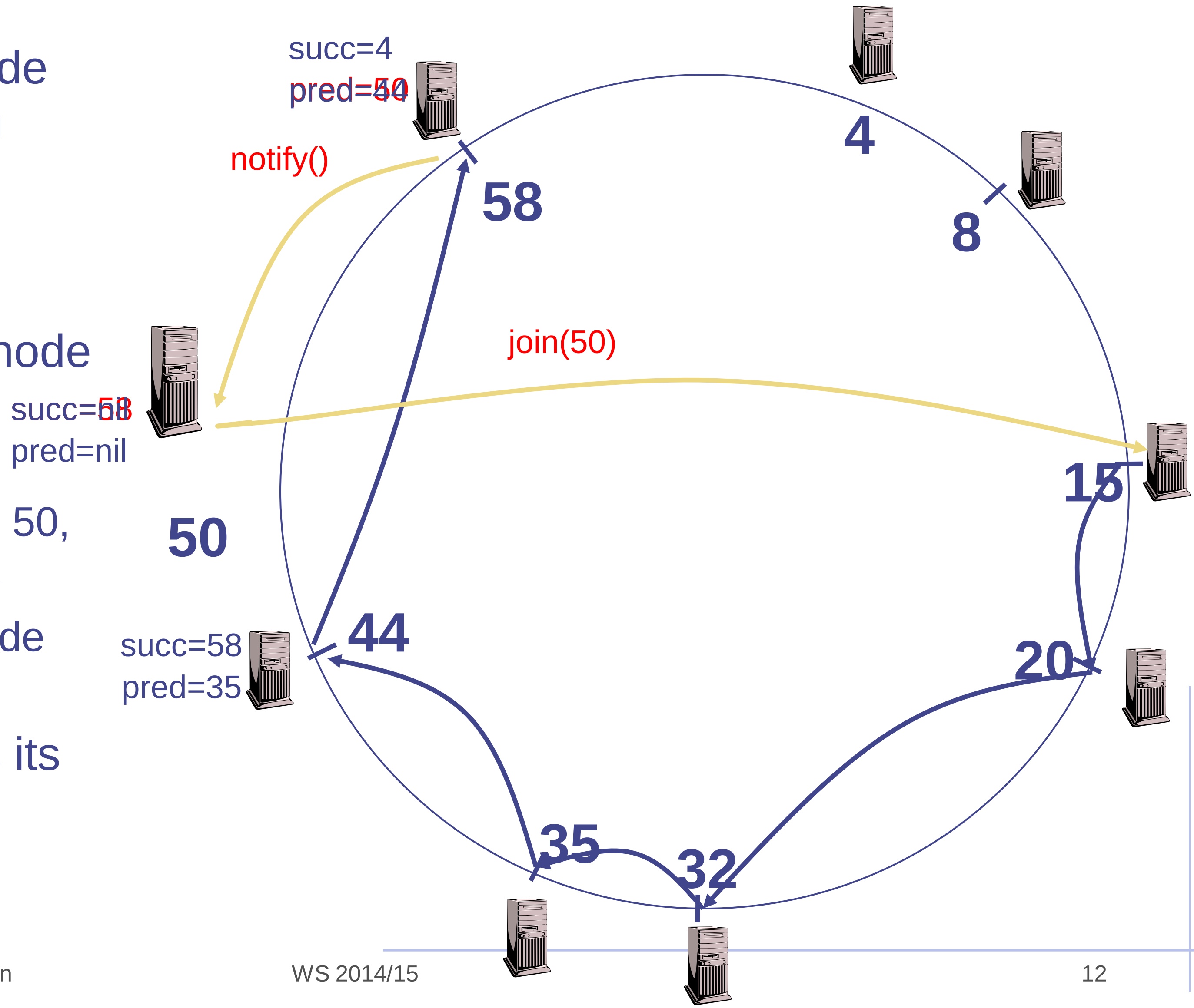
- Node with id=50 joins the ring
- Node 50 needs to know at least one node already in the system
  - Assume known node is 15



Joining Operation

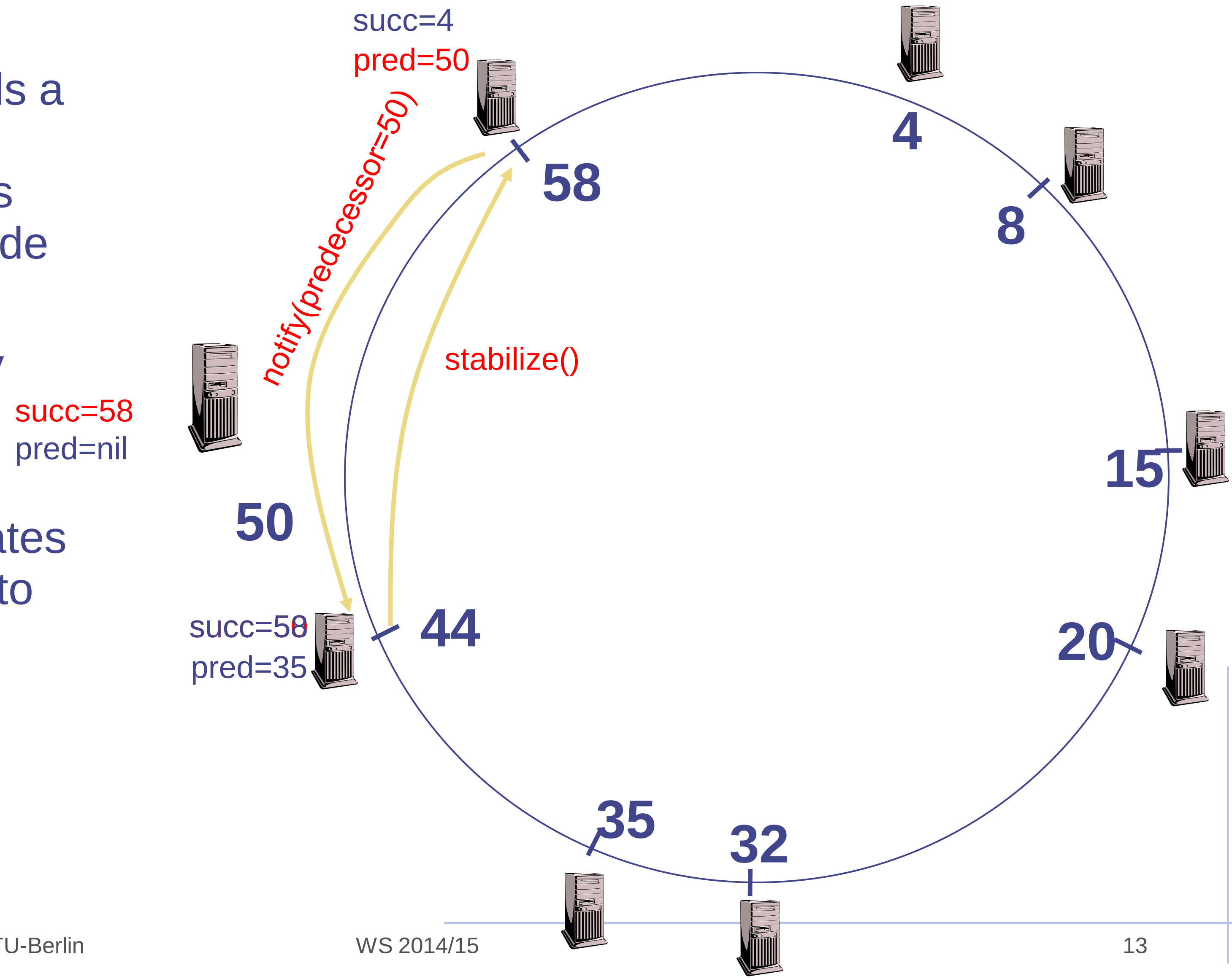
[Scott Shenker and Ion Stoica, UCB]

- Node 50 asks node 15 to forward join message
- When join(50) reaches the destination (i.e., node 58), node 58
  1) updates its predecessor to 50,
  2) returns a notify message to node 50
- Node 50 updates its successor to 58



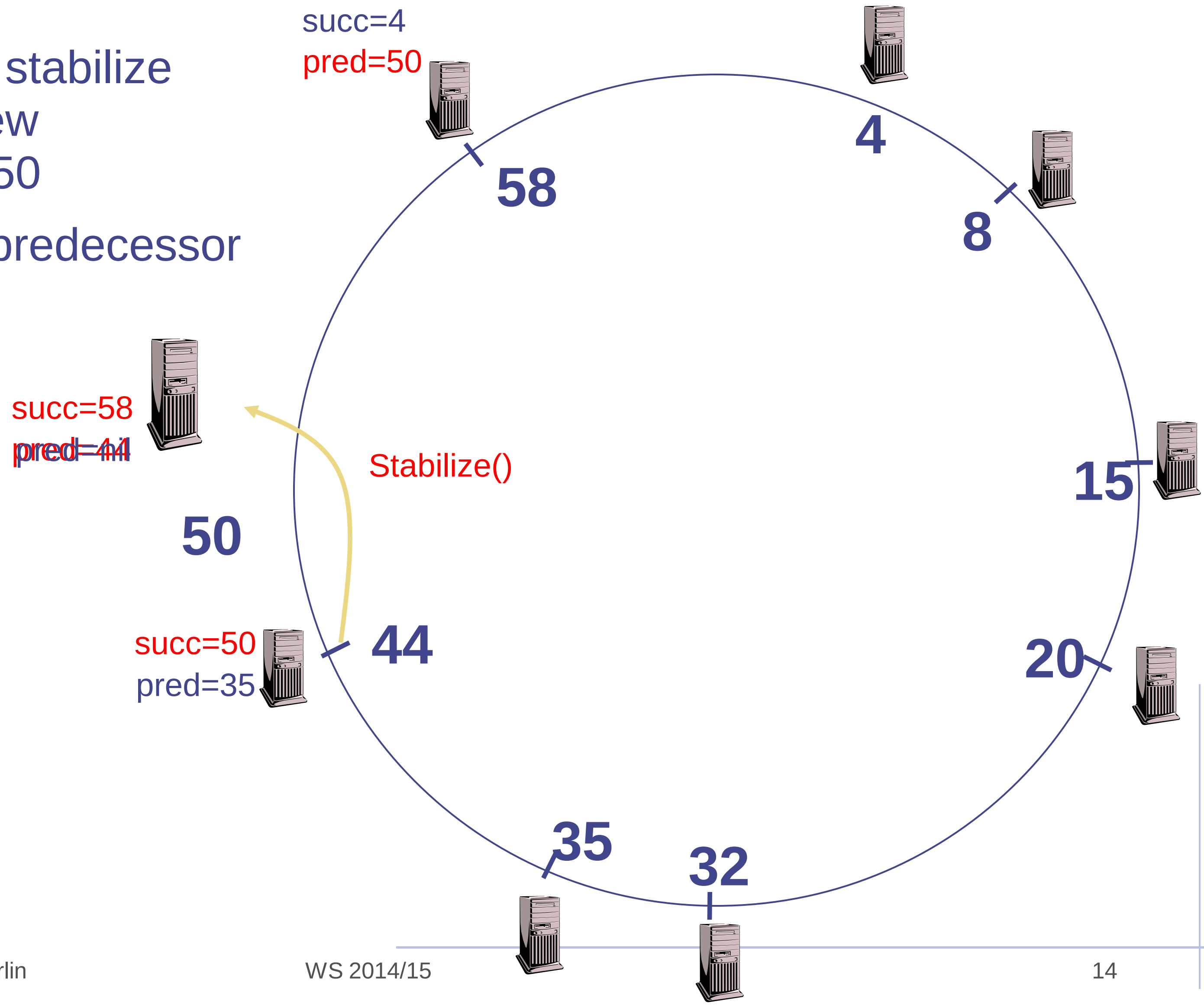
Joining Operation (cont'd) [Scott Shenker and Ion Stoica, UCB]

- 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



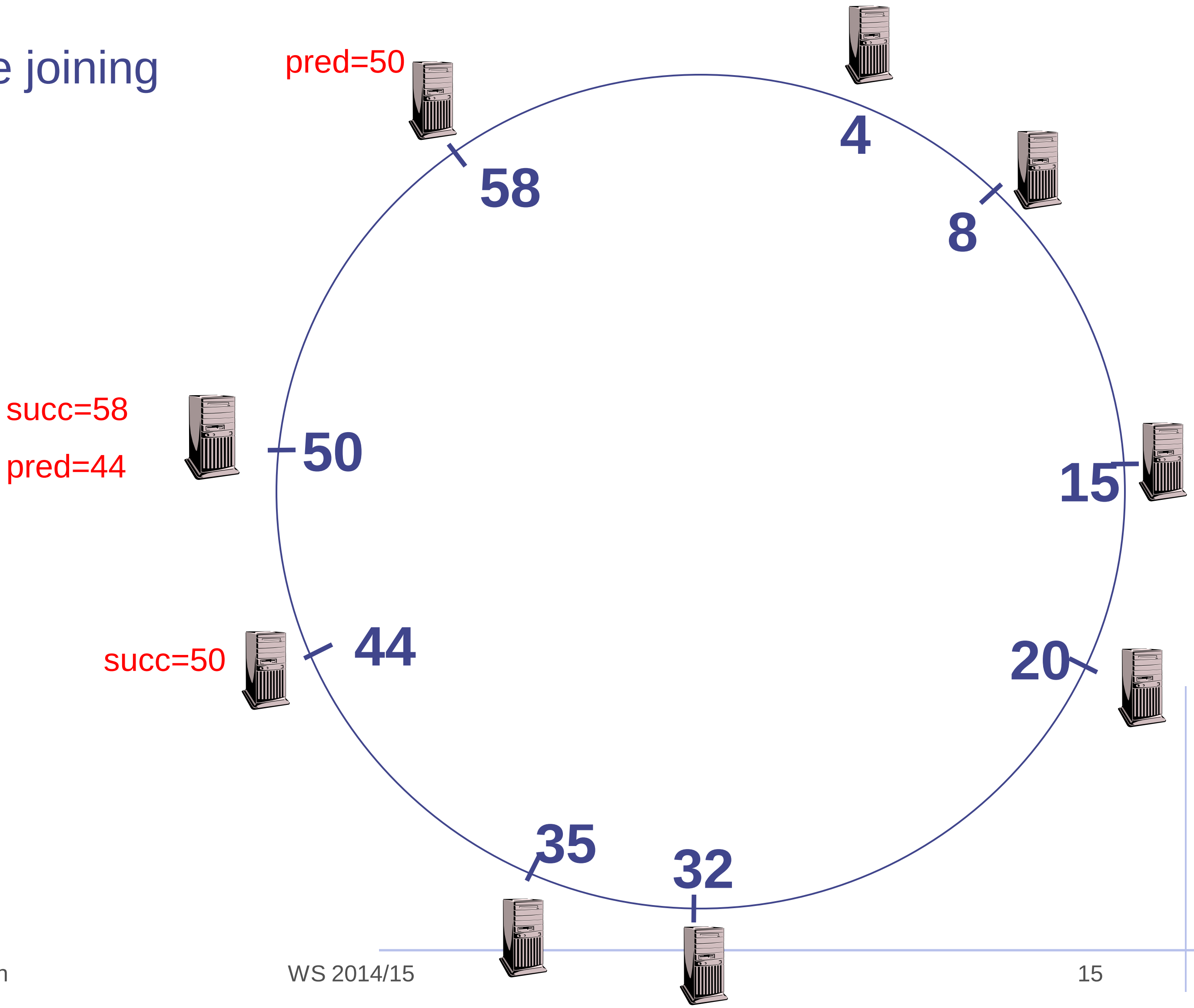
Joining Operation (cont'd)

- 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!

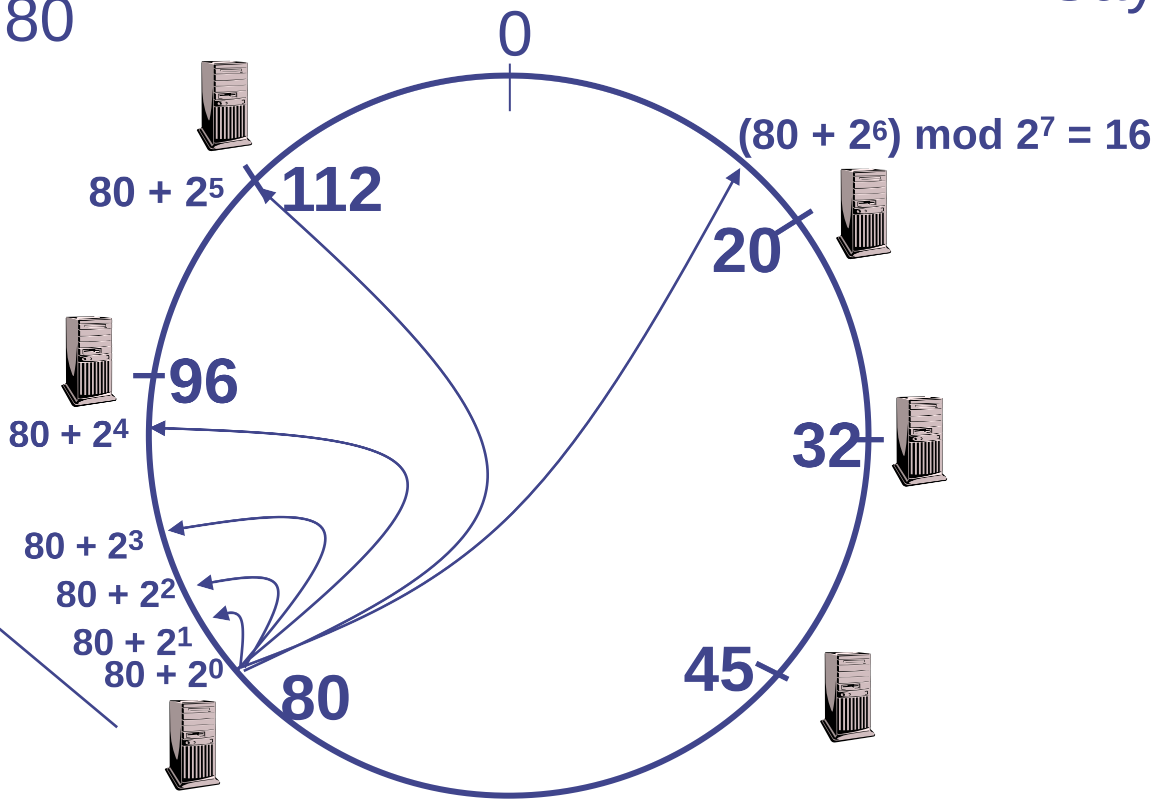


Achieving Efficiency: *finger tables*

Say $m=7$

Finger Table at 80

i	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	112
6	20



*i*th entry at peer with id *n* is first peer with id >= $n + 2^i \pmod{2^m}$

Chord Performance (improvements)

- Chord 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

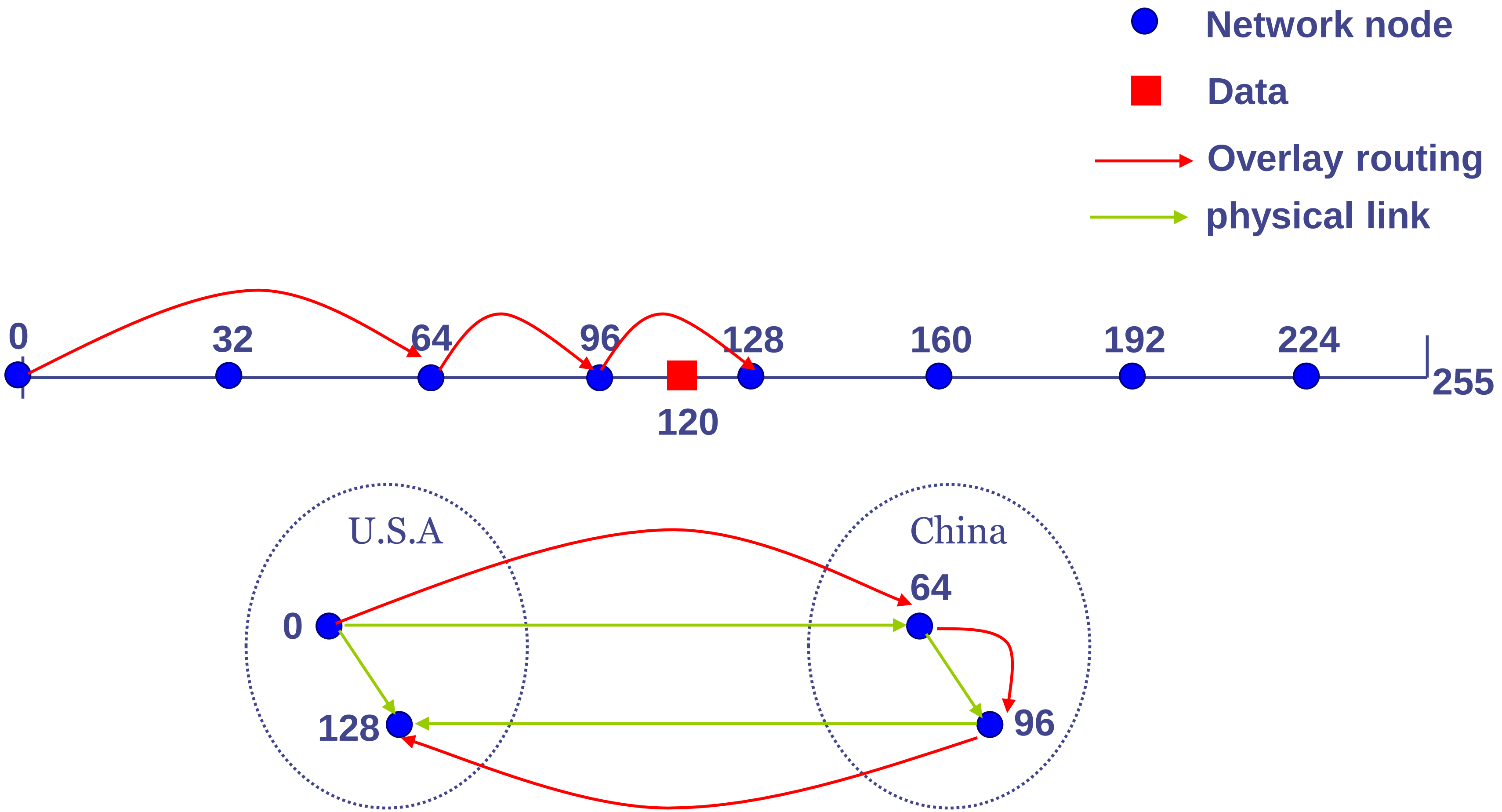
- Reducing latency
  - Chose finger that reduces expected time to reach destination
  - Chose the closest node from range $[N+2^{i-1}, N+2^{i})$ as successor

- Stretch is another parameter….

$$= \frac{\text{latency for each lookup on the overlay topology}}{\text{average latency on the underlying topology}}$$

  - Nodes <u>close</u> on ring, but <u>far away</u> in Internet
  - Goal: put nodes in routing table that result in few hops <u>and</u> low latency

Latency stretch in Chord [Ratnasamy et al. 2001]



A Chord network with N(=8) nodes and m(=8)-bit key space

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

The Problem of Membership Churn [Sean C. Rhea, Intel Res]

- In a system with 1,000s of machines, some machines failing / recovering at all times
- This process is called *churn*
- Without repair, quality of overlay network degrades over time
- A significant problem deployed DHTs systems

Observation: in >50 % cases, MTBF in order of minutes..

What Makes a Good DHT Design [Felber, Eurecom]

- The number of neighbors for each node should remain “reasonable” (small degree)
- DHT routing mechanisms should be decentralized (no single point of failure or bottleneck)
- Should gracefully handle nodes joining and leaving
  - Repartition the affected keys over existing nodes
  - Reorganize the neighbor sets
  - Bootstrap mechanisms to connect new nodes into the DHT
- DHT must provide low stretch
  - Minimize ratio of DHT routing vs. unicast latency between two nodes

Multiple solutions…

- Chord
- Tapestry
- Pastry
- CAN
- ….

DHTs Support Many Applications

[Felber, Eurecom]

- File sharing [CFS, OceanStore, PAST, …]
- Web cache [Squirrel, …]
- Censor-resistant stores [Eternity, FreeNet, …]
- Application-layer multicast [Narada, …]
- Event notification [Scribe]
- Naming systems [ChordDNS, INS, …]
- Query and indexing [Kademlia, …]
- Communication primitives [I3, …]
- Backup store [HiveNet]
- Web archive [Herodotus]