CSx55: Distributed Systems [DHTs]

Routing in DHTs
So many, many
nodes and items
But the mapping's unambiguous
Deterministic to boot

Each node in the know only
about a few others
Messages relayed closer and closer
In a few bounded hops

Shrideep Pallickara
Computer Science
Colorado State University

1

Frequently asked questions from the previous class survey

- Is there ever a problem comparing such large numbers?
- N-fold replication of content in routers vs overlays
- Clockwise vs anti-clockwise traversals
- How can a node hierarchy be built up to optimize lookups based on text?

2

Topics covered in this lecture

- Distributed Hash Tables
- Chord

Distributed hash tables

- Few constraints on the structure of the keys
- REQUIREMENTS
  - Data identified using numeric **keys**
  - Nodes must be willing to store keys **for each other**

Storage and retrieval in distributed hash tables

- Data items are *inserted* and *found* by specifying a unique **key** for the data
- Underlying algorithm must determine ***which node*** is responsible for storing the data

Distributed Storage using DHTs: Publishing a file

- **Convert** file-name to numeric key
  - Using one-way hash functions like MD5 or SHA-1
- Call **`lookup(key)`**
  - Returns IP address of node responsible for key
- **Send file** to be stored at node returned by `lookup`

Distributed Storage using DHTs: Retrieving a file

1. Obtain name of file
2. Convert it to a key using one-way hash function
3. Call `lookup(key)`
4. Ask resulting node, from (3), for a copy of the file

7

8

Implementing DHTs: 3 core elements

- **Mapping** keys to nodes
- **Forwarding** a lookup for a key to the appropriate node
- Building **routing tables**

Implementing DHTs: **Mapping keys to nodes**

- Must be load balanced
- Done using one-way hash functions
  - MD5 (128-bit) or SHA-1 (160-bit)
- Ensures that content is distributed **uniformly**

Implementing DHTs
Forwarding lookups

- Any node that receives query for key
  - Must forward it to a node whose ID is ***closer*** to the key
- Above rule guarantees that query ***eventually arrives*** at the closest node
- For e.g.:
  - Node has ID 346, and key has ID 542
  - Forwarding to node 495 gets it numerically closer

Implementing DHTs:
Building routing tables

- Multiple nodes participate in locating content
- Each node must know about **some other** nodes
  - To forward lookup requests
  - SUCCESSOR
    - The node with the ***closest succeeding*** ID
  - Other nodes
    - For efficiency in routing

Distributed hash tables: Identifiers

- Data items are assigned an identifier from a large random space
  - 128-bit UUIDs or 160-bit SHA1 digests
- **Nodes are also assigned a number from the same identifier space**

Crux of the DHT problem

- Implement an efficient, **deterministic** scheme to
  - Map data items to node
- When you **look up** a data item
  - Network address of node holding the data is returned

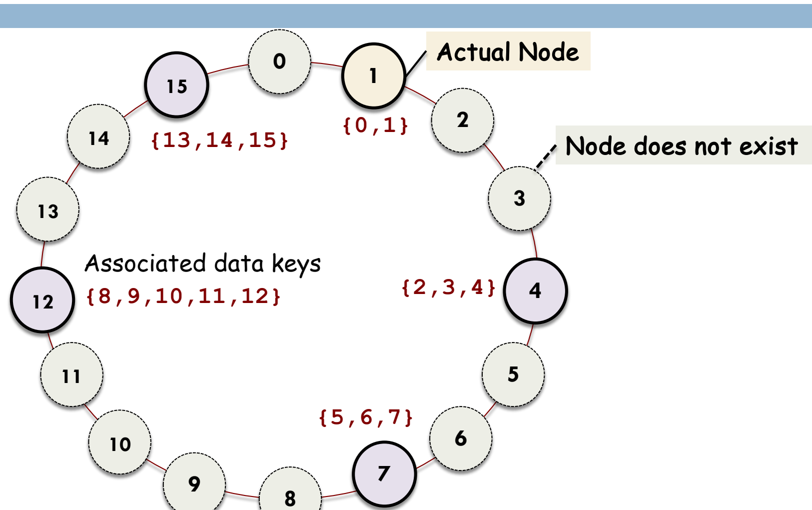
15

The Chord System

- Assigns IDs to keys and nodes from the same 1-dimensional ID space
- Nodes are organized into a **ring**
- Data item with key **`k`** is mapped to a node with the **smallest** **`id`** $\geq$ **`k`**
  - Also referred to as `successor(`**`k`**`)`

16

Mapping of data items to nodes in Chord



Chord lookups

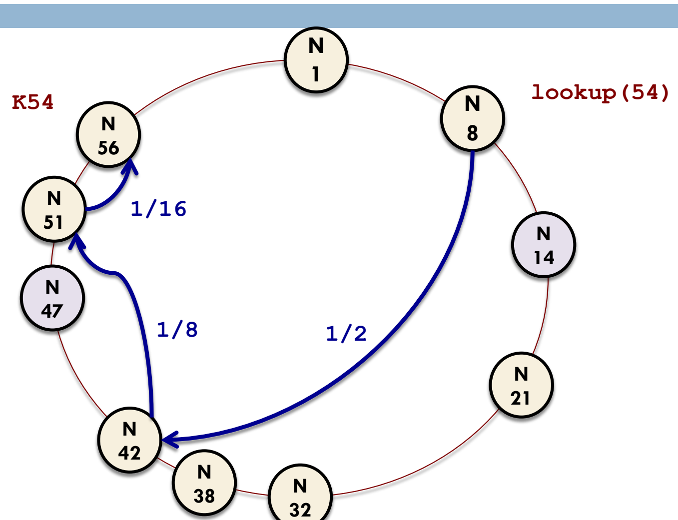
- **N** is the number of possible nodes in the system
- Each node maintains a **finger table**
  - With log N entries
  - Entries contains IP addresses of nodes
    - Half-way around the ID space from it
    - 1/4th, 1/8th, … **in powers of two**
  - Ensures node can forward lookup query to at least ½ of the remaining ID-space distance to key
    - Lookups in *O(log N)*

Storing keys and forwarding lookups

- An entity with key k falls under the ***jurisdiction*** of node with the *smallest identifier* id
  - $id >= k$
  - Referred to as the successor of k or $succ(k)$
- A node ***forwards*** query for key k to node (in its FT) with highest ID $\leq k$
  - The exception is ONLY when the first entry is greater than k
    - In this case, that node is responsible for storing that element

19

Chord lookup example for k=54



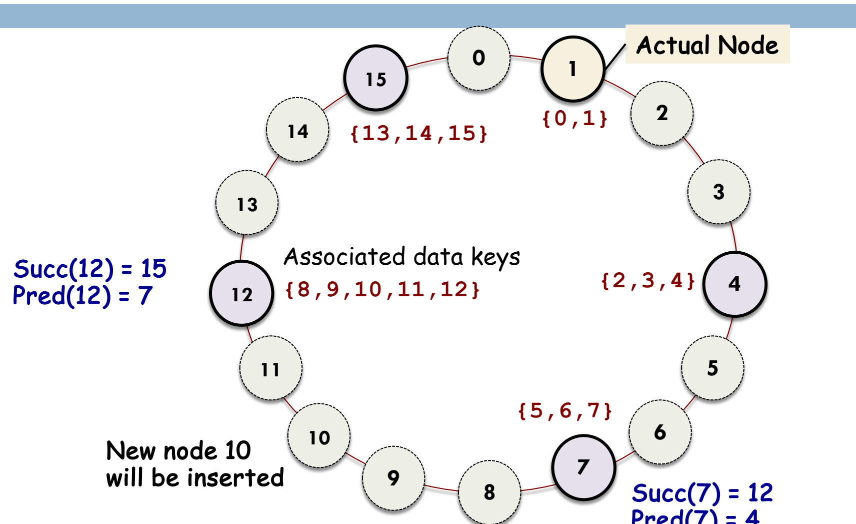
20

When a node wants to join

- Generate a random id
  - Probability of collisions is low
- **`lookup(id)`**
  - Will return successor(id)
- Contact *successor(id)* and its predecessor
  - Insert self in the ring
  - **Transfer** data items
    - All keys must be fetched from the new node's successor

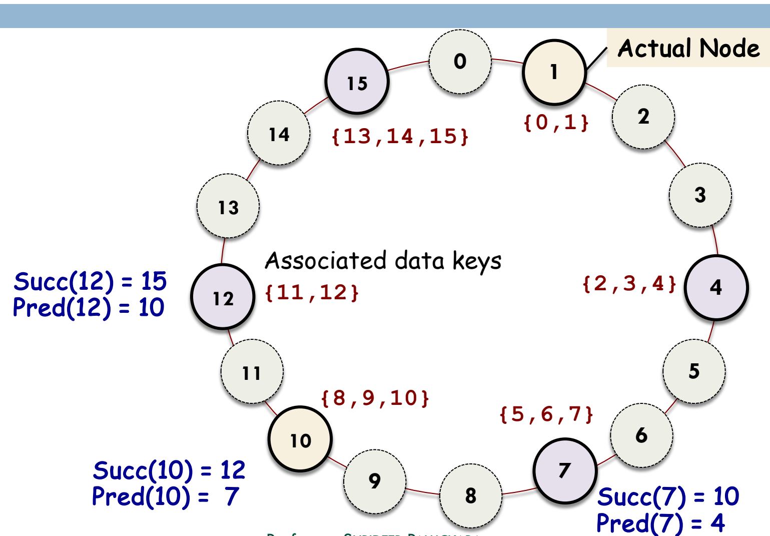
21

An example of inserting a new node



22

An example of inserting a new node

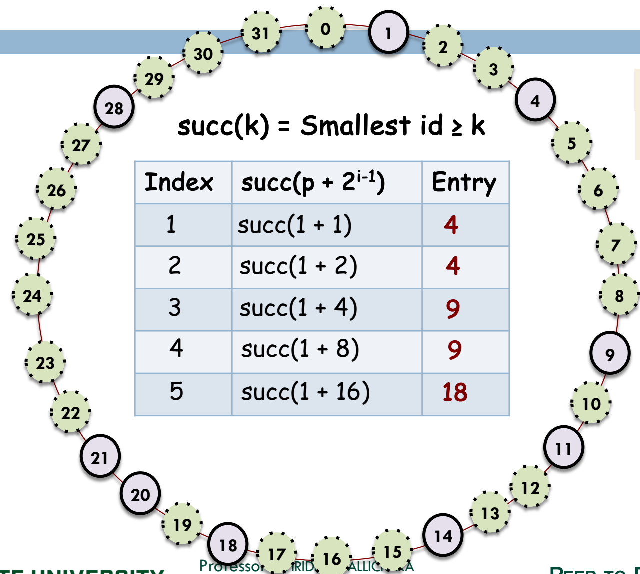


Finger Table in Chord

- Chord uses an m-bit identifier space
  - 2^m possible peers
- Each node, p, in Chord maintains a Finger Table with m-entries
  - **$FT_p[i] = succ(p + 2^{i-1})$**

Note: This is when you count your indices from 1.
When you code, and we are counting from 0 this would be
$FT_p[i] = succ(p + 2^i)$

Constructing the Finger Table: Node 1



succ(k) = Smallest id ≥ k

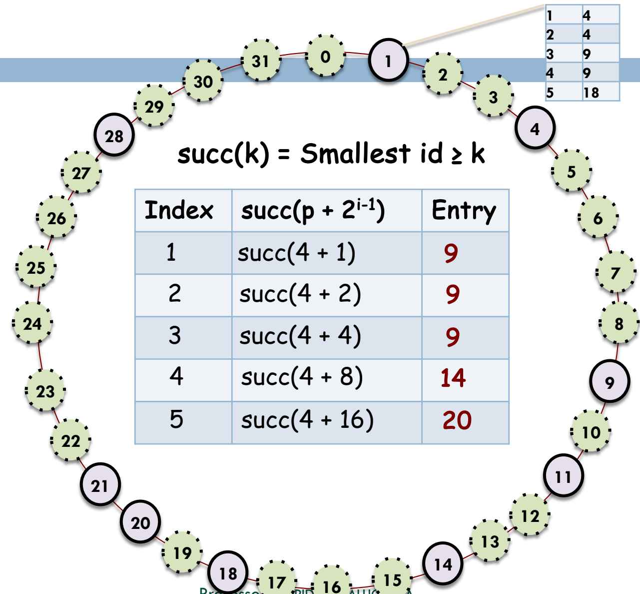
Index	succ(p + 2^{i-1})	Entry
1	succ(1 + 1)	4
2	succ(1 + 2)	4
3	succ(1 + 4)	9
4	succ(1 + 8)	9
5	succ(1 + 16)	18

We are looking at a 5-bit ID space.

IDs go from 0 through ($2^5 - 1$)

25

Constructing the Finger Table: Node 4



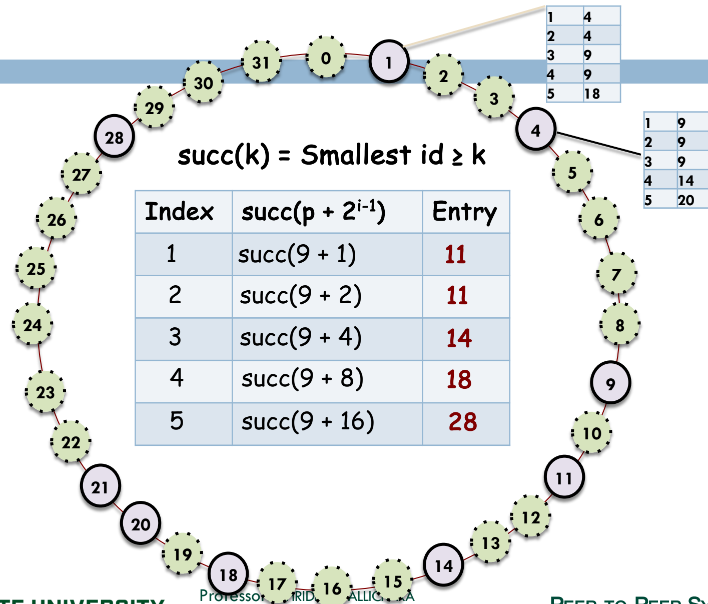
1	4
2	4
3	9
4	9
5	18

succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(4 + 1)	9
2	succ(4 + 2)	9
3	succ(4 + 4)	9
4	succ(4 + 8)	14
5	succ(4 + 16)	20

26

Constructing the Finger Table: Node 9

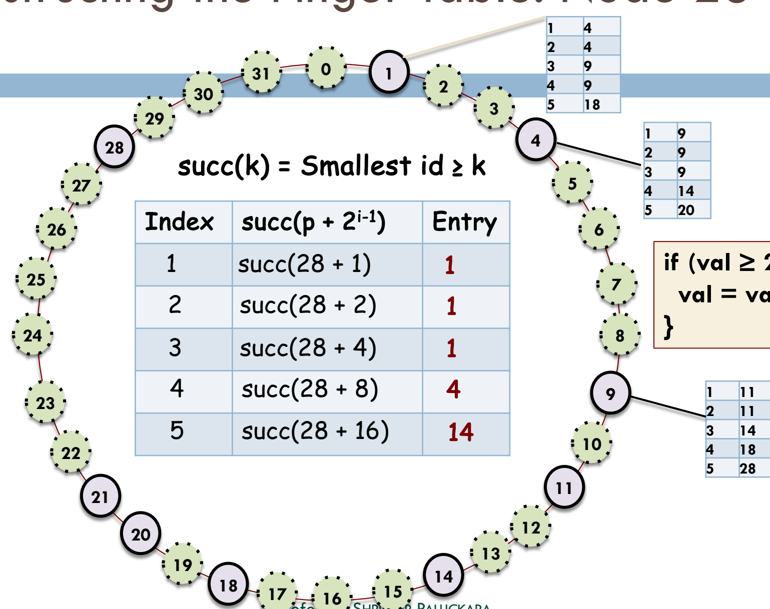


succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(9 + 1)	11
2	succ(9 + 2)	11
3	succ(9 + 4)	14
4	succ(9 + 8)	18
5	succ(9 + 16)	28

27

Constructing the Finger Table: Node 28



succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(28 + 1)	1
2	succ(28 + 2)	1
3	succ(28 + 4)	1
4	succ(28 + 8)	4
5	succ(28 + 16)	14

```
if (val ≥ 2^m) {
  val = val (mod 2^m)
}
```


28

Using the finger table to route queries: Make sure you don't overshoot

- To lookup a key k, node p will forward query to node q with index j in p's FT where:

Node with greatest ID less than or equal to k

$$q = FT_p[j] \leq k < FT_p[j+1]$$

OR

$$q = FT_p[1] \text{ when } p < k < FT_p[1]$$

First entry ONLY if its ID is greater than k

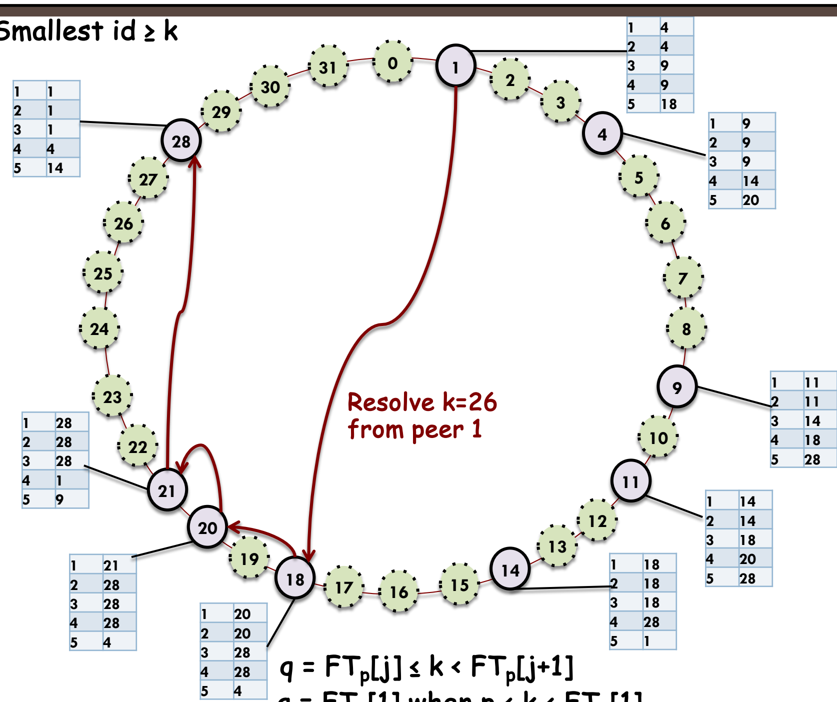
Stop forwarding the query when you are the target node

- A node is ***responsible*** for keys that fall in the range

  key > predecessor

  key <= self

Smallest id ≥ k

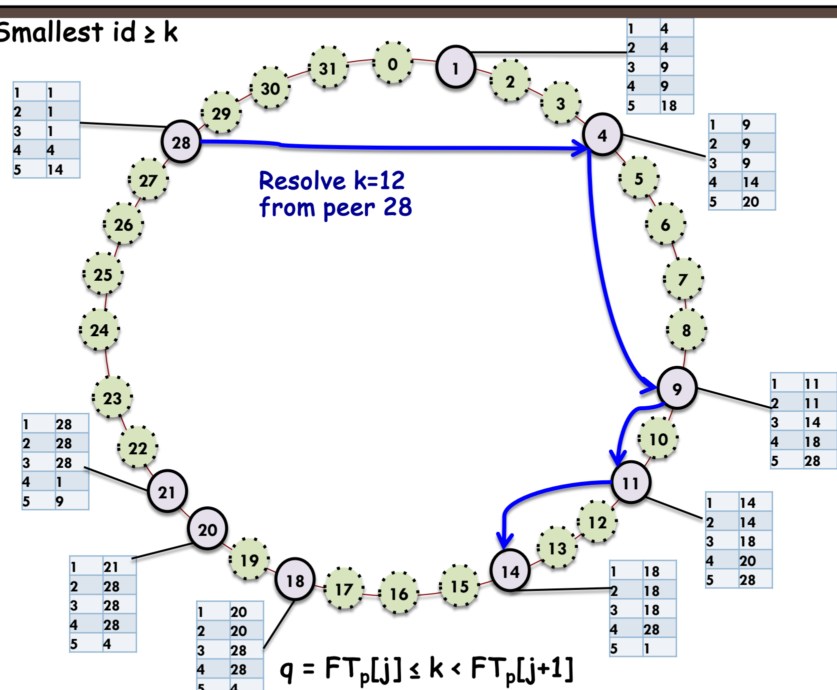


$q = FT_p[j] \leq k < FT_p[j+1]$

$q = FT_p[1]$ when $p < k < FT_p[1]$

31

Smallest id ≥ k



$q = FT_p[j] \leq k < FT_p[j+1]$

$q = FT_p[1]$ when $p < k < FT_p[1]$

32

Keeping the finger table up-to-date: At node q, $FT_q[1]$ must be accurate

① Contact *succ(q+1)* {This is $FT_q[1]$}
- Have it return its predecessor

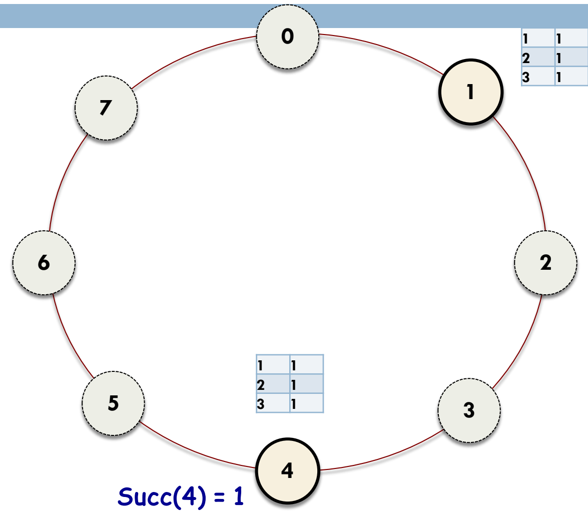
② If $q = pred(\ succ(q+1)\ )$
- Everything is fine

③ Otherwise:
- There is a new node p such that $q < p \leq succ(q+1)$
- $FT_q[1] = p$
- Check if p has recorded q as its predecessor
  No? Go to step (1)

33

34

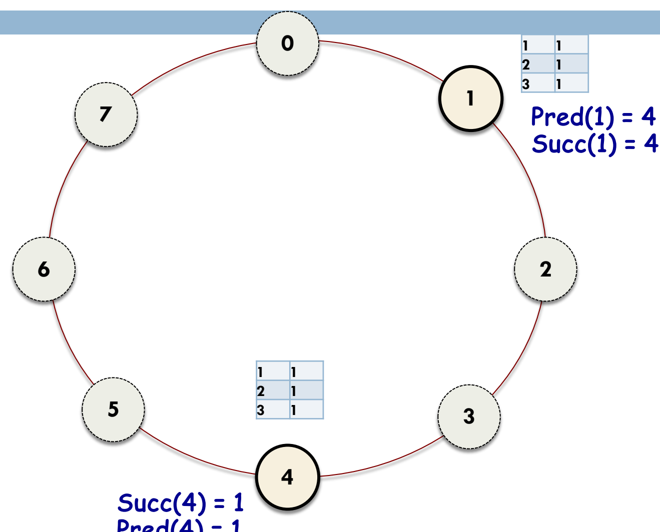
An example of inserting a new node N-4: Node-4 comes in and contacts Node-1



COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L15.35

35

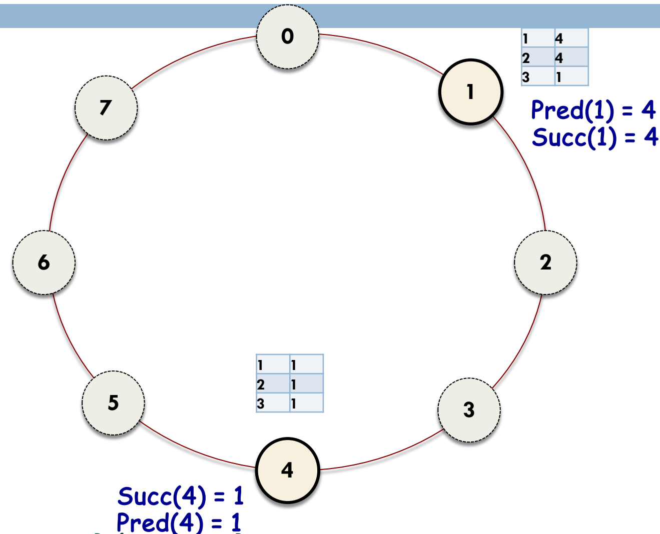
Installing successor at Node-1



COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L15.36

36

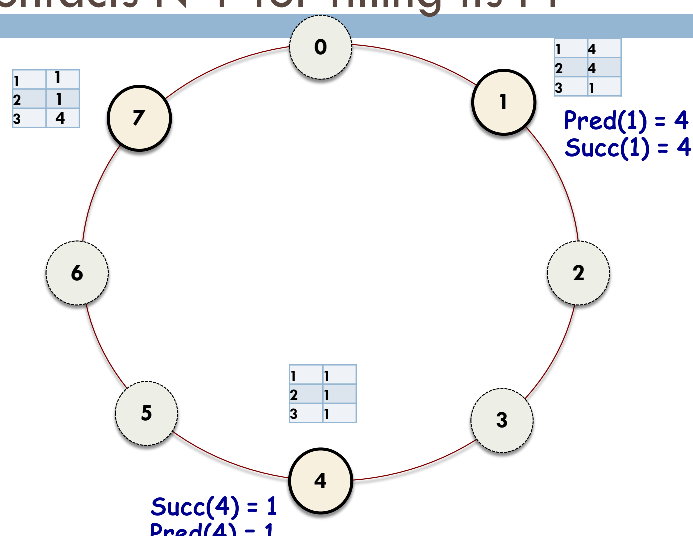
Updating the FT at N-1



COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L15.37

37

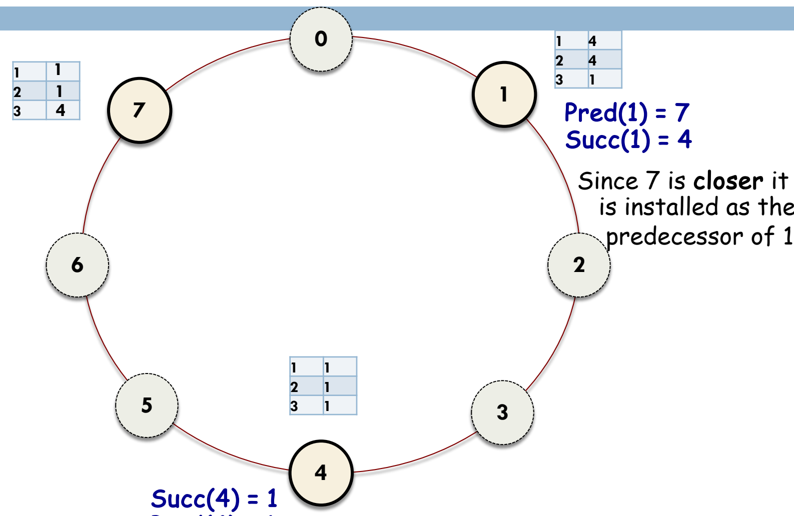
An example of inserting a new node N-7: N-7 contacts N-1 for filling its FT



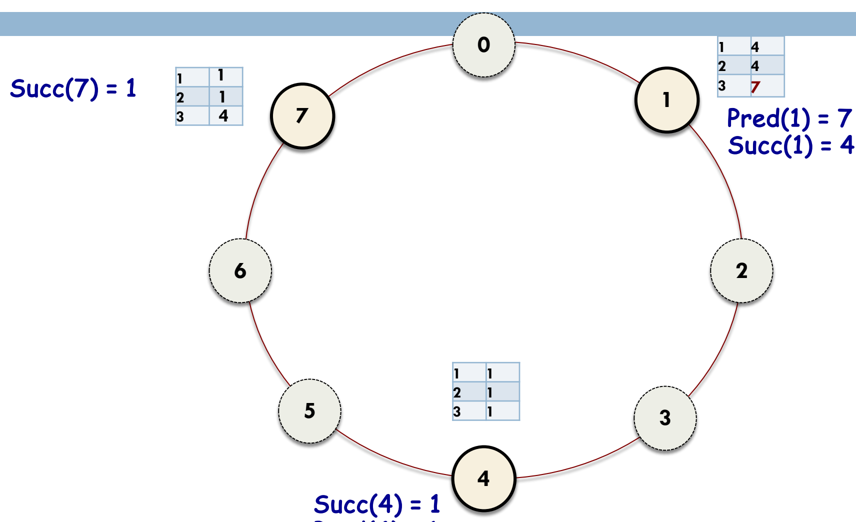
COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L15.38

38

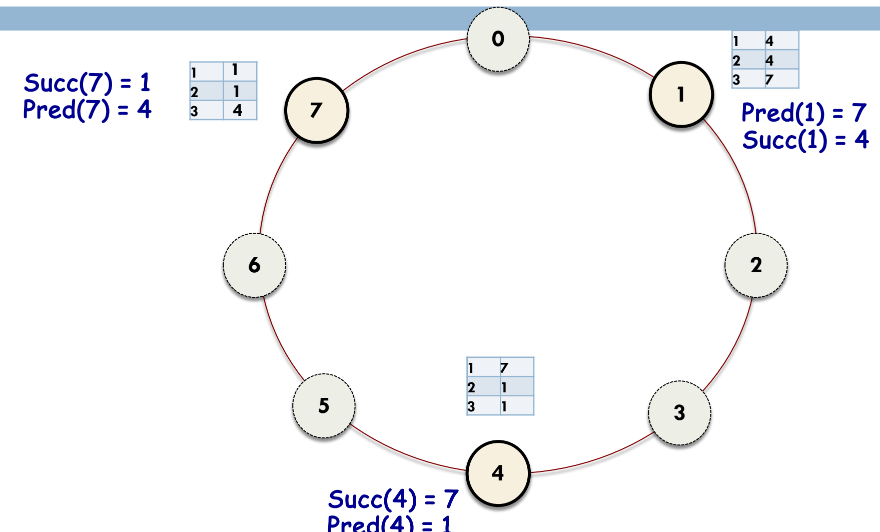
N-7 informs N-1 that it (N-7) is now N-1's predecessor



When N-1 updates its FT later on …

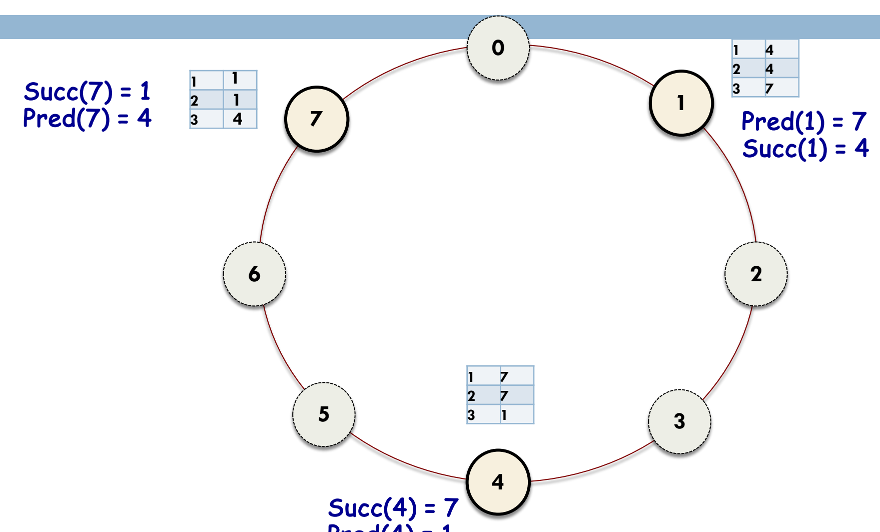


N-4 contacts N-1 to see if it is still its predecessor … and installs N-7 as its successor



41

When the FT at N-4 is updated …



42

The contents of this slide-set are based on the following references

- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van der Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 5]
- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]