Distributed Systems Design
COMP 6231

Naming: Chapter 5 Part 1

Lecture 6

Essam Mansour

Today…

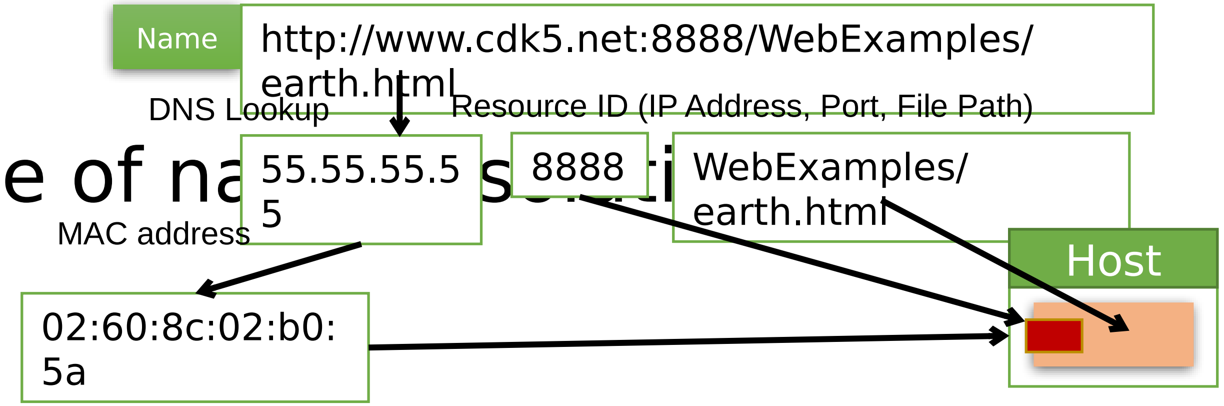
- Today’s Session:
  - Naming

Naming

- Names are used to uniquely identify entities in distributed systems
  - Entities may be processes, remote objects, newsgroups, etc.,

- Names are mapped to entities' locations using *name resolution*

- An example of na



Names, Addresses, and Identifiers

- An entity can be identified by three types of references

  a) Name
    - A name is a set of bits or characters that references an entity
    - Names can be human-friendly (or not)

  b) Address
    - Every entity resides on an access point, and access point has an address
    - Addresses may be location-dependent (or not)
    - E.g., IP Address + Port

  c) Identifier
    - Identifiers are names that *uniquely* identify entities
    - A *true identifier* is a name with the following properties:
      - An identifier refers to at-most one entity
      - Each entity is referred to by at-most one identifier
      - An identifier always refers to the same entity (i.e. it is never reused)

Naming Systems

- A *naming system* is simply a middleware that assists in name resolution

- Naming systems can be classified into three classes, based on the type of names used:
    a. Flat naming
    b. Structured naming
    c. Attribute-based naming

Classes of Naming

- Flat naming
- Structured naming
- Attribute-based naming

Flat Naming

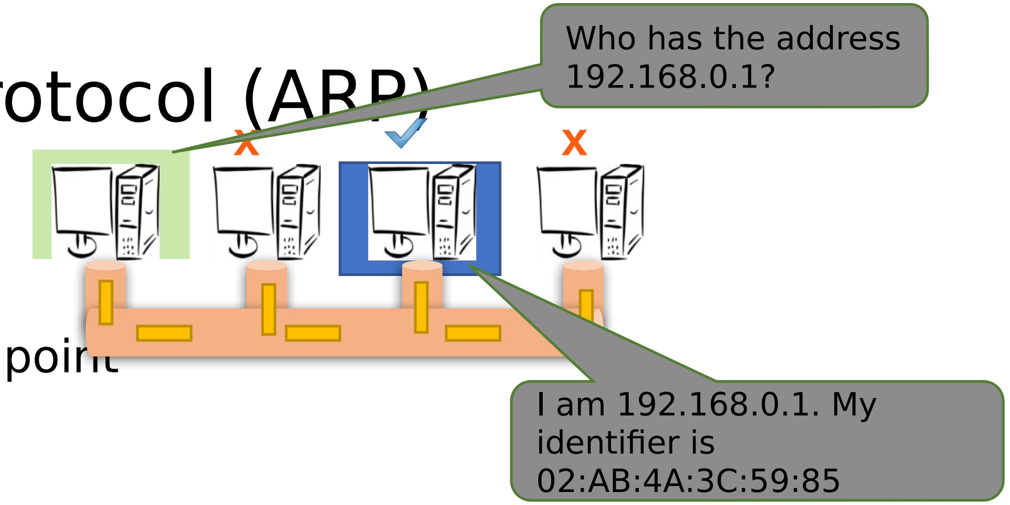
- In flat naming, identifiers are simply random bits of strings (known as *unstructured* or flat names)

- A flat name does not contain any information on how to locate an entity

- We will study four types of name resolution mechanisms for flat names:
  1. Broadcasting
  2. Forwarding pointers
  3. Home-based approaches
  4. Distributed Hash Tables (DHTs)

1. Broadcasting

- Approach: Broadcast the name/address to the whole network; the entity associated with the name responds with its current identifier
- Example: Address Resolution Protocol (ARP)
  - Resolve an IP address to a MAC address
  - In this system,
    - IP address is the *address* of the entity
    - MAC address is the *identifier* of the access point

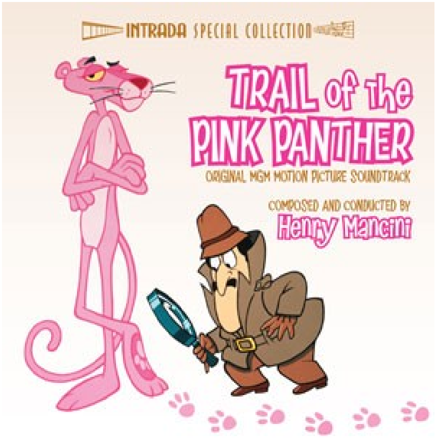


- Challenges:
  - Not scalable in large networks
    - This technique leads to flooding the network with broadcast messages
  - Requires all entities to *listen* (or *snoop*) to all requests

2. Forwarding Pointers

- Forwarding pointers enable locating *mobile* entities
  - Mobile entities move from one access point to another

- When an entity moves from location A to location B, it leaves behind (at A) a reference to its new location at B



- Name resolution mechanism:
  - Follow the *chain of pointers* to reach the entity
  - Update the entity’s reference when the present location is found

- Challenges:
  - Long chains lead to longer resolution delays
  - Long chains are prone to failures due to broken links

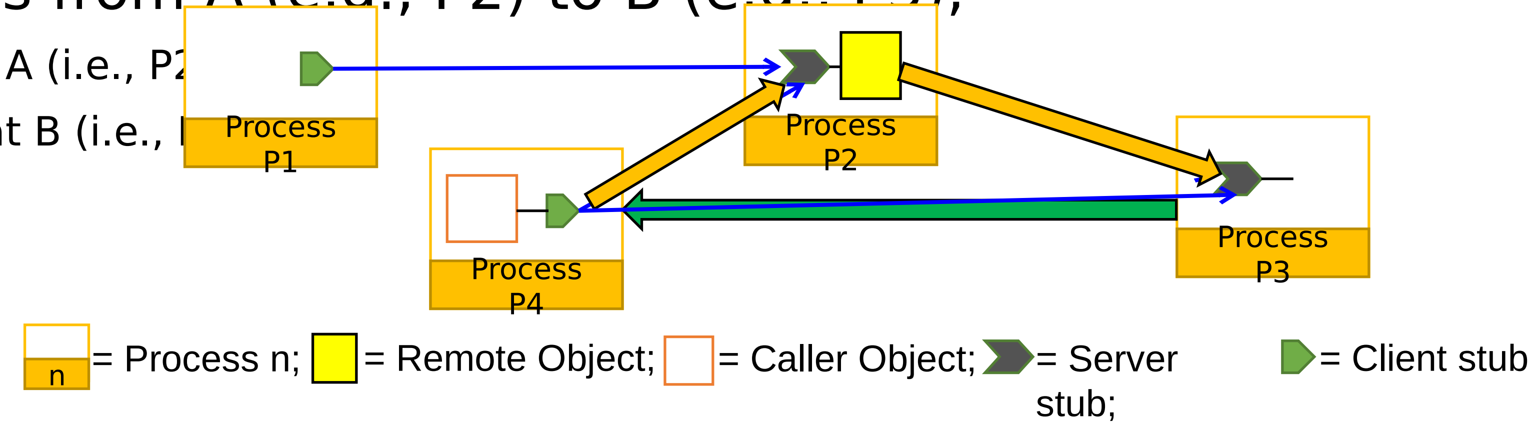
Forwarding Pointers – An Example

- Stub-Scion Pair (SSP) chains implement remote invocations for mobile entities using *forwarding pointers*
  - Server stub is referred to as <u>*Scion*</u> in the original paper
- Each forwarding pointer is implemented as a pair:

```
(client stub, server stub)
```

  - The server stub contains a local reference to the actual object or a local reference to another client stub

- When object moves from A (e.g., P2) to B (e.g., P3),
  - It leaves a client stub at A (i.e., P2
  - It installs a server stub at B (i.e.,



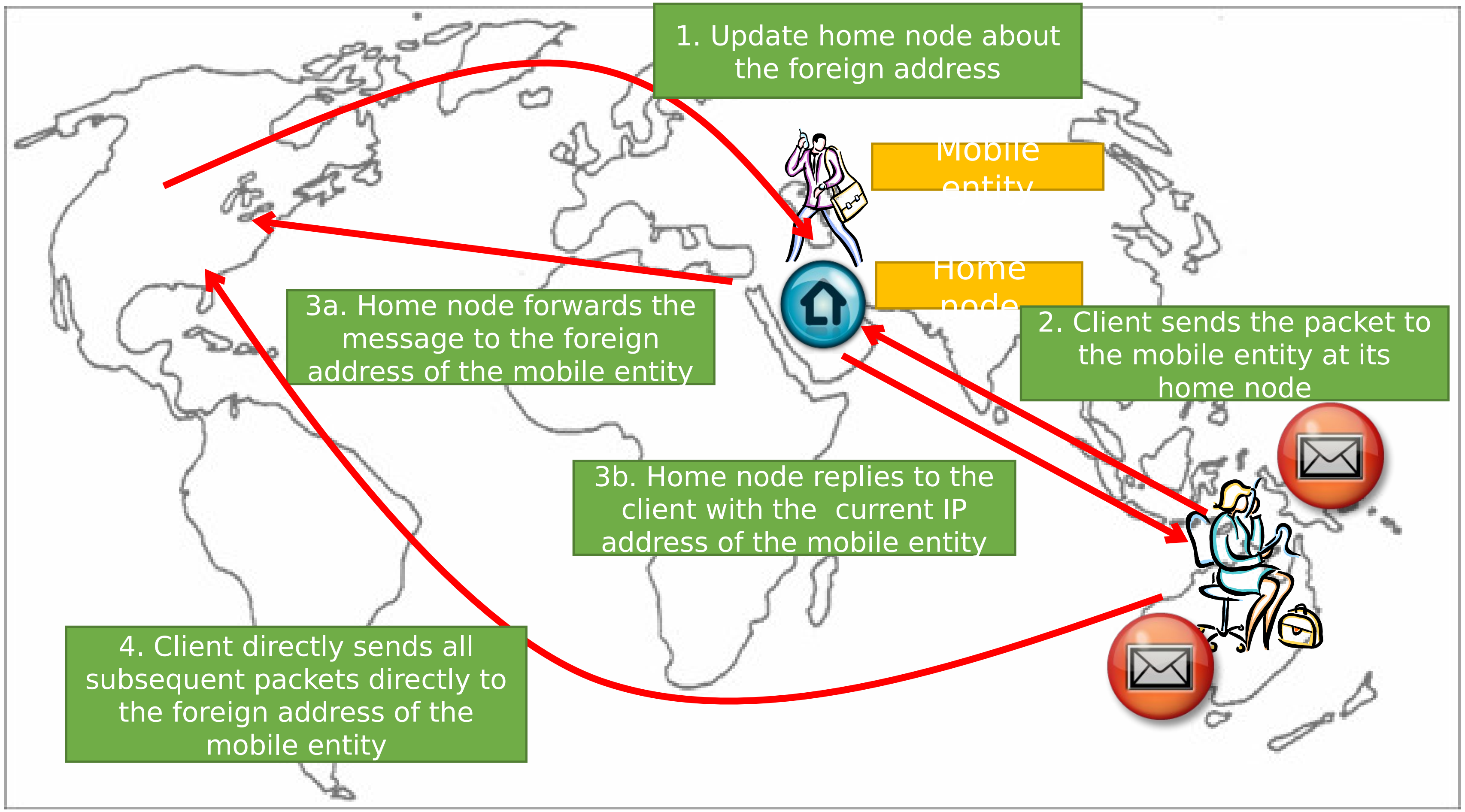
3. Home-Based Approaches

- Each entity is assigned a home node
  - The home node is typically *static* (has fixed access point and address)
  - It keeps track of the *current* address of the entity

- Entity-home interaction:
  - Entity’s home address is registered at a naming service
  - The entity updates the home about its current address (foreign address) whenever it moves

- Name resolution:
  - Client contacts the home to obtain the foreign address
  - Client then contacts the entity at the foreign location

3. Home-Based Approaches – An Example



3. Home-Based Approaches – Challenges

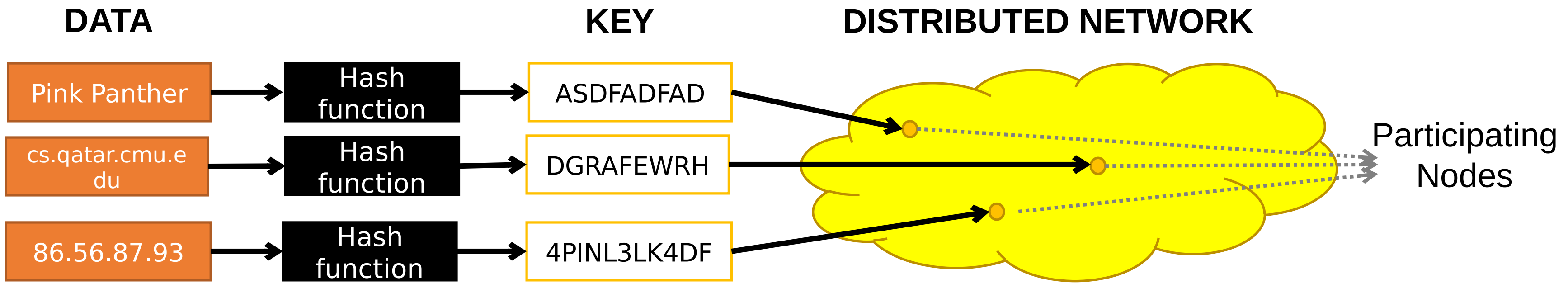
- The static home address is permanent for an entity’s lifetime
  - If the entity permanently moves, then a *simple* home-based approach incurs higher communication overhead

- Connection set-up overheads due to communication between the client and the home can be excessive
  - Consider the scenario where the clients are nearer to the mobile entity than the

4. Distributed Hash Table (DHT)

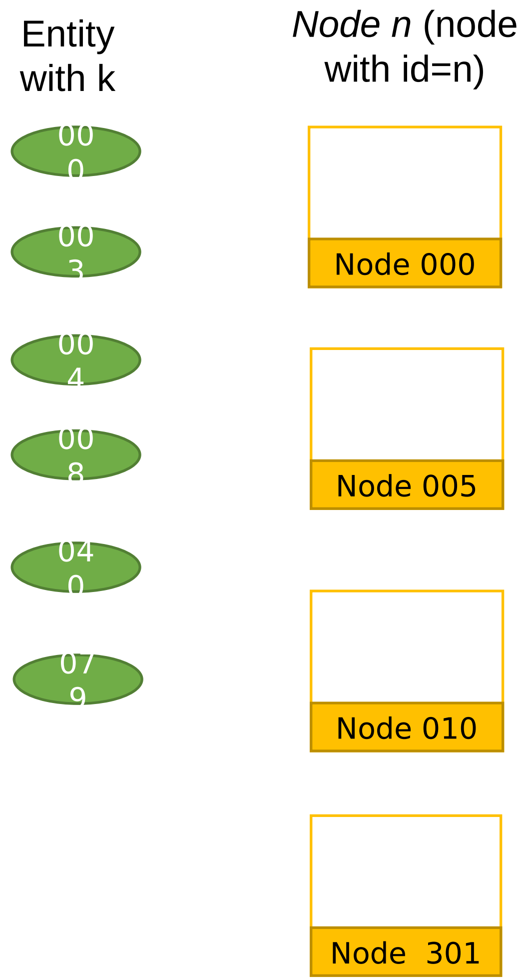
- DHT is a distributed system that provides a lookup service similar to a hash table
  - *(key, value)* pair is stored in the nodes participating in the DHT
  - The responsibility for maintaining the mapping from keys to values is distributed among the nodes
  - Any participating node can serve in retrieving the value for a given key

- We will study a representative DHT known as Chord



Chord

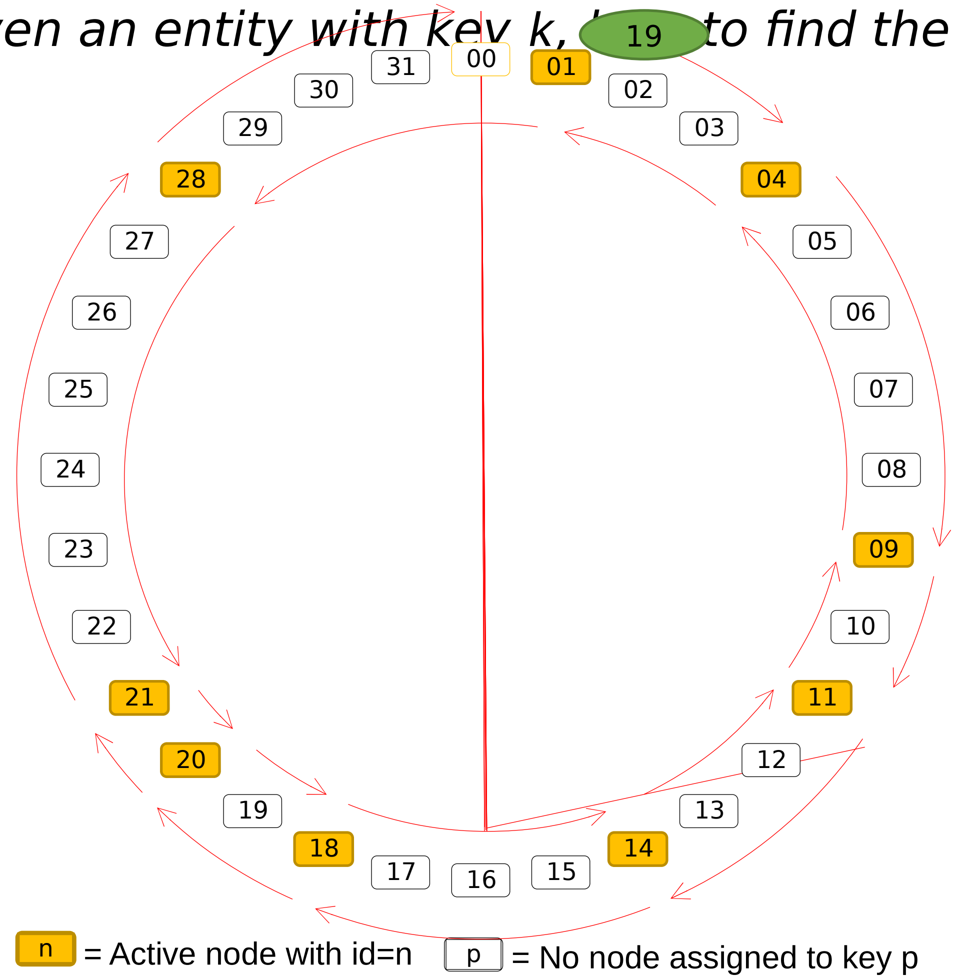
- Chord assigns an *m-bit identifier* (randomly chosen) to each node
  - A node can be contacted through its network address
- Alongside, it maps each entity to a node
  - Entities can be processes, files, etc.,
- Mapping of entities to nodes
  - Each node is responsible for a set of entities
  - An entity with key k falls under the jurisdiction of the node with the smallest identifier `id >= k`. This node is known as the *successor of k*, and is denoted by `succ(k)`



Map each entity with key k to node succ(k)

A Naïve Key Resolution Algorithm

- The main issue in DHT is to efficiently resolve a key *k* to the network location of `succ(k)`
  - *Given an entity with key k, how to find the node `succ(k)`?*

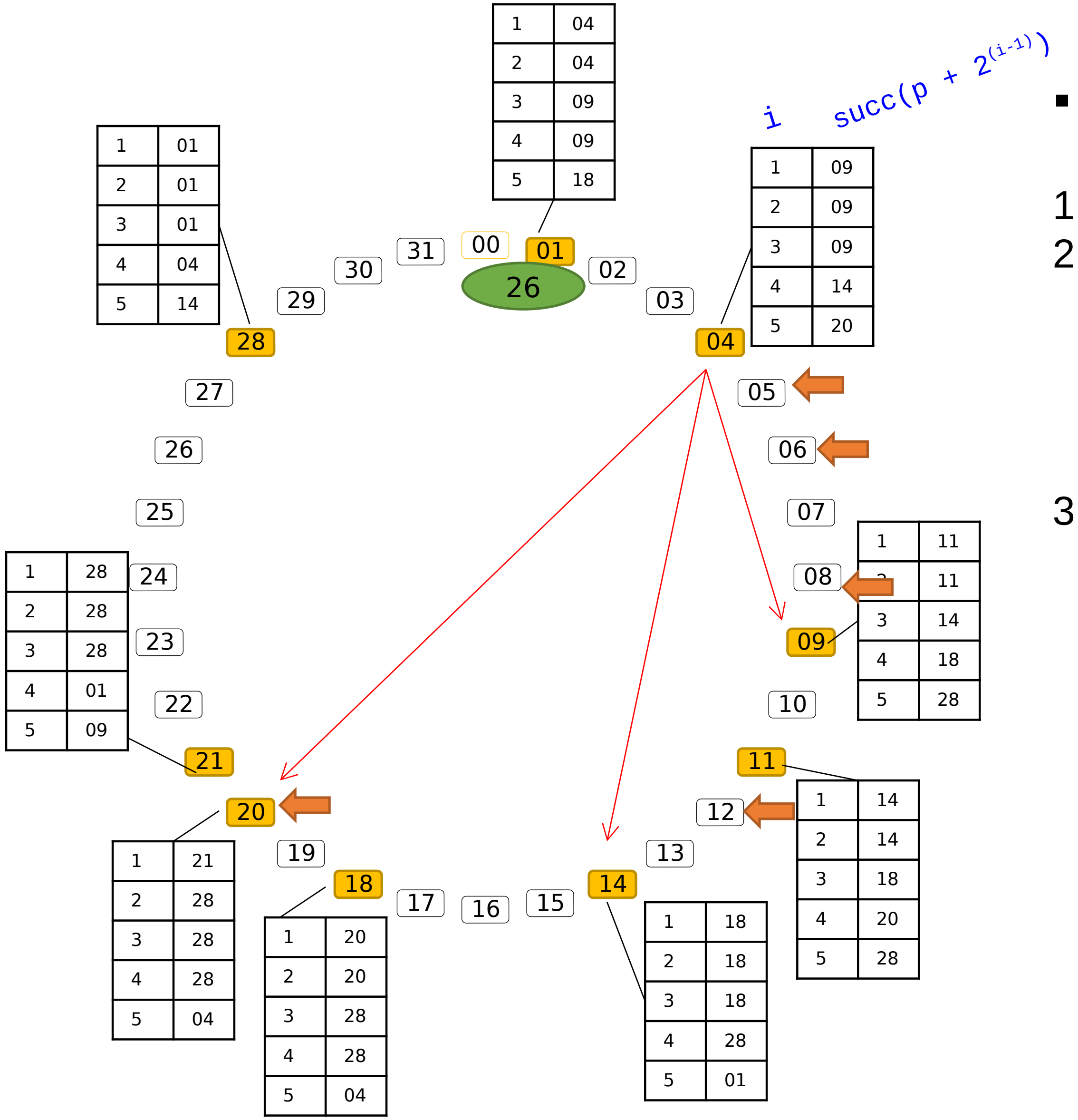


1. All nodes are arranged in a logical ring according to their IDs
2. Each node ‘p’ keeps track of its immediate neighbors: `succ(p)` and `pred(p)`
3. If ‘p’ receives a request to resolve key ‘k’:
   - If `pred(p) < k <=p`, node p will handle it
   - Else it will forward it to `succ(n)` or `pred(n)`

Solution is not scalable:

- As the network grows, forwarding delays increase
  - Key resolution has a time complexity of `O(n)`

Key Resolution in Chord



- Chord improves key resolution by reducing the time complexity to `O(log n)`

1. All nodes are arranged in a logical ring according to their IDs
2. Each node ‘p’ keeps a table FT_p of at-most *m* entries. This table is called Finger Table

   $$FT_p[i] = succ(p + 2^{(i-1)})$$

   NOTE: $FT_p[i]$ increases exponentially

3. If node ‘p’ receives a request to resolve key ‘k’:
   - Node p will forward it to node q with index j in F_p where

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

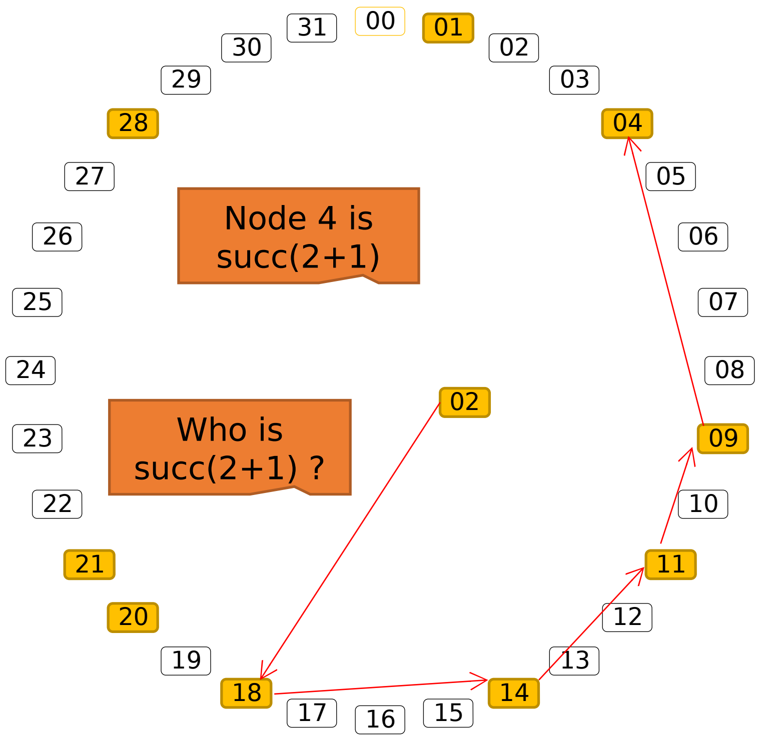
     - If $k > FT_p[m]$, then node p will forward it to $FT_p[m]$
     - If $k < FT_p[1]$, then node p will forward it to $FT_p[1]$

Chord – Join and Leave Protocol

- In large-scale distributed Systems, nodes dynamically *join* and *leave* (voluntarily or due to failures)

- If a node p wants to join:
  - It contacts arbitrary node, looks up for `succ(p+1)`, and inserts itself into the ring

- If node p wants to leave:
  - It contacts `pred(p)` and `succ(p+1)` and updates them



Chord – Finger Table Update Protocol

- For any node q, $FT_q[1]$ should be up-to-date
  - It refers to the next node in the ring
  - Protocol:
    - Periodically, request `succ(q+1)` to return `pred(succ(q+1))`
    - If `q = pred(succ(q+1))`, then information is up-to-date
    - Otherwise, a new node p has been added to the ring such that `q < p < succ(q+1)`
      - $FT_q[1] = p$
      - Request p to update `pred(p) = q`
    - Similarly, node p updates each entry `i` by finding $succ(p + 2^{(i-1)})$

Exploiting Network Proximity in Chord

- The logical organization of nodes in the overlay network may lead to inefficient message transfers
  - Node `k` and node `succ(k +1)` may be far apart

- Chord can be optimized by considering the network location of nodes
  1. Topology-Aware Node Assignment
     - Two nearby nodes get identifiers that are close to each other

  2. Proximity Routing
     - Each node `q` maintains '`r`' successors for i^{th} entry in the finger table
     - $FT_q[i]$ now refers to r successor nodes in the range

       $$[p + 2^{(i-1)}, p + 2^i - 1]$$

     - To forward the lookup request, pick one of the r successors closest to the node q

A To-Do List

- Read Chapters 5