# NAMING

Distributed Systems

4. Sem BSc Informatics

IMC FH Krems

# LECTURE OVERVIEW

- Introduction and motivation
- Flat naming
- Structured naming
- Attribute-based naming

# NAMING

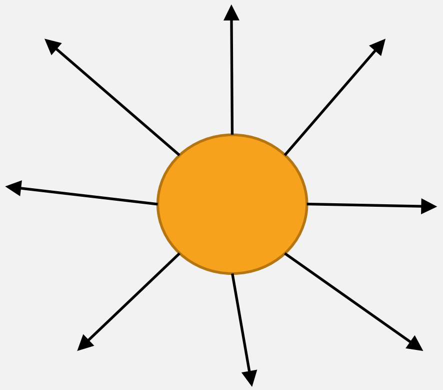
Why are names important in distributed systems?

# NAMING

**Types of names**

1. Address: point of access where the entity can be found.
   - Example: network address of a machine.
2. Identifier: unique name used for an entity.
   - 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 (will not be reused for another entity).
   - Example: key in a distributed hash table (DHT).
3. Human-friendly names: character strings readable by humans.
   - Example: domain names, file names.

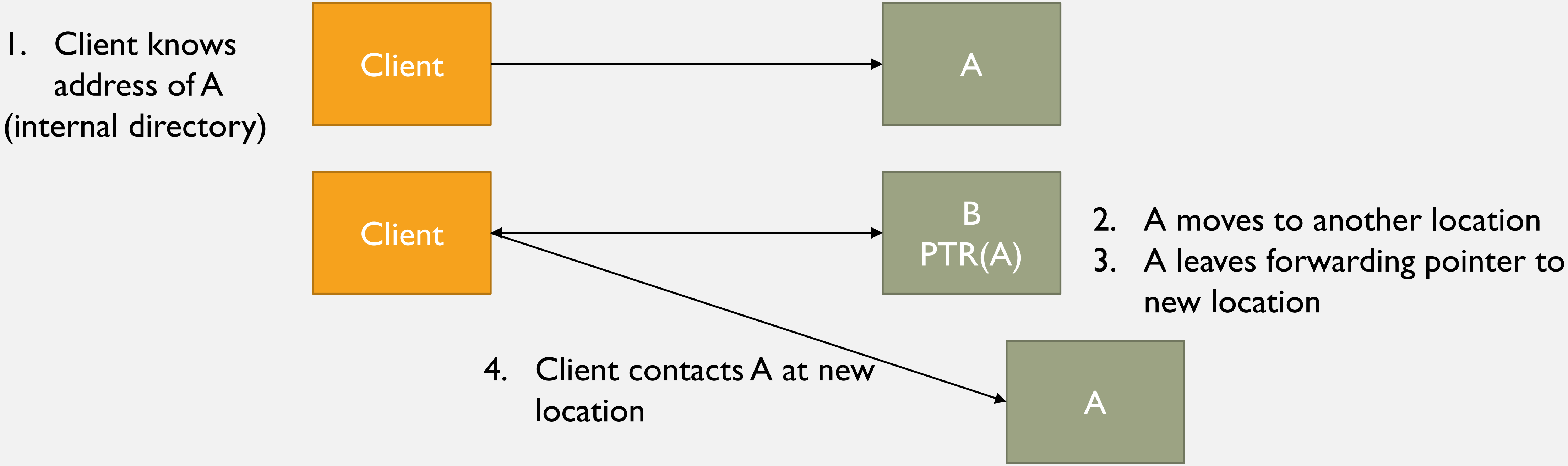
# FLAT NAMING



- Resolves the mapping identifier → address.
- First approach: broadcasting
  - Identifier is broadcast to each machine in a network.
  - Machines with an entry point for that entity send reply.
  - Example: **Address Resolution Protocol (ARP)**.
    - IP address is sent to all machines of a network via broadcast message.
    - Machine with that IP address replies with its network address.
  - Only efficient on moderate-size networks (e.g. LAN).

# FLAT NAMING

- Second approach: forwarding pointers.



# FLAT NAMING

- Drawbacks of forwarding pointers
  1. Forwarding chains can become very long.
  2. Each node has to be maintained as long as necessary.
  3. Broken links.
- Chains have to be kept short and up to date → maintaining effort.

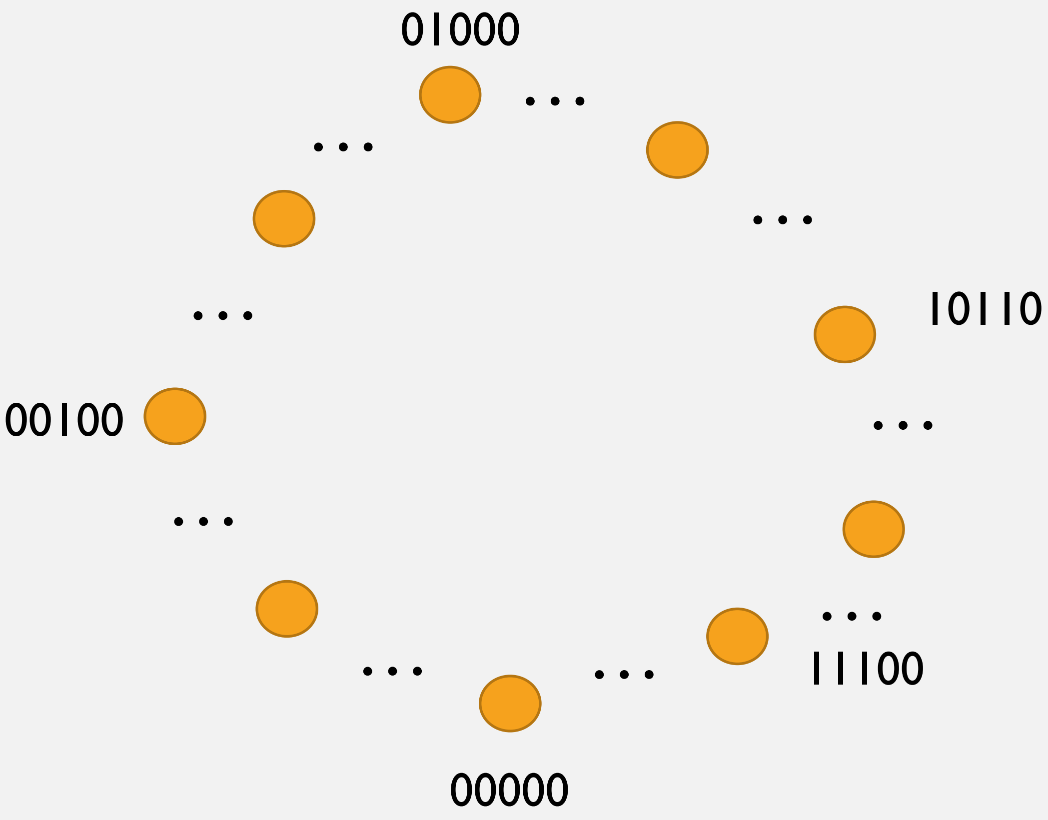
# FLAT NAMING

- Next approach: **Distributed Hash Tables (DHT).**
- (key, value) pairs are distributed among the nodes of the network.
- Value can be an address, or the entity itself (e.g. a document).
- Specially important in P2P networks.
  - Examples: BitTorrent, Freenet.
- Nodes store a part of the lookup table with a subset of entries.
- Nodes that join/leave the network are updated locally.
  - No central coordination.
- Extremely scalable approach.

# FLAT NAMING

**Distributed Hash Tables**

- Example: **Chord**.
- Identifier space in a *m*-bit space (e.g. 160 bits)
  - SHA-1 hash of a name is calculated as key.
  - Potentially $2^m$ nodes forming a circle.
- Both **nodes** and **entities** have identifiers in this space.



# FLAT NAMING

**Chord**

- Entity with id $k$ is assigned the node with the smallest id such that $id \geq k$. This node is called the *successor* of entity $k$, succ($k$).
- Our problem is to find the address of succ($k$), given $k$.
- Each node stores a *finger table* (FT) with entries pointing to other nodes.
  - Entry $j$ points to the first node succeeding the current node by at least $2^{j-1}$
  - Example: FT[1] → points to first node $q$ succeeding the current node by at least 1
    - FT[2] → points to node $q'$ succeeding the current node by at least 2
    - FT[5] → points to node $q''$ succeeding the current node by at least 16
- Node $p$ forwards request to node $q$ in its finger table, so that it is the node with the smallest id that could store $k$.

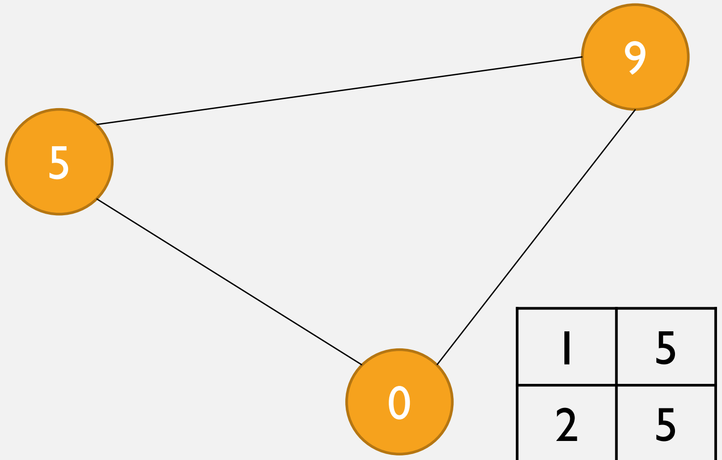
# FLAT NAMING

Chord: Example

- $m$ = 4, currently 3 nodes

keys(5)={1, 2, 3, 4, 5}

| | |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 0 |




| | |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 5 |

keys(9)={6, 7, 8, 9}

| | |
|---|---|
| 1 | 5 |
| 2 | 5 |
| 3 | 5 |
| 4 | 9 |

keys(0)={10, 11, 12, 13, 14, 15, 0}

# FLAT NAMING

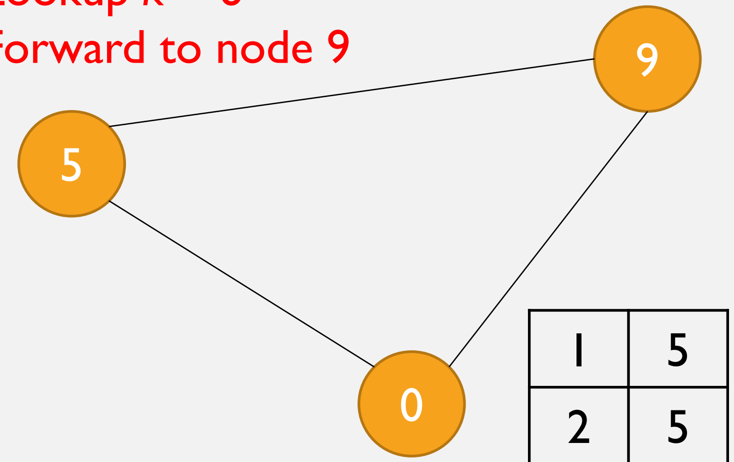
Chord: Example

- Resolve key = 6

3. Lookup $k$ = 6
4. Forward to node 9

keys(5)={1, 2, 3, 4, 5}

| | |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 0 |




| | |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 5 |

keys(9)={6, 7, 8, 9}

5. Lookup $k$ = 6
6. Found in node 9

| | |
|---|---|
| 1 | 5 |
| 2 | 5 |
| 3 | 5 |
| 4 | 9 |

keys(0)={10, 11, 12, 13, 14, 15, 0}

1. Lookup $k$ = 6
2. Forward to node 5

# FLAT NAMING

Chord

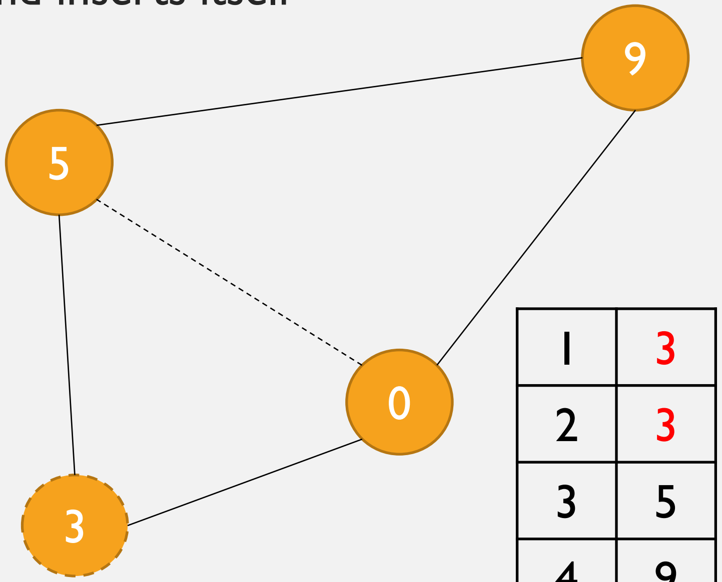
- Join the network: new node *p* looks up succ(*p*+1) and inserts itself

| | |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 3 |

keys(9)={6, 7, 8, 9}

keys(5)={~~1, 2, 3, 4~~, 5}

| | |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 0 |




3. Rest of nodes issue succ(*k*) requests regularly, updating their FT accordingly

| | |
|---|---|
| 1 | 5 |
| 2 | 5 |
| 3 | 9 |
| 4 | 0 |

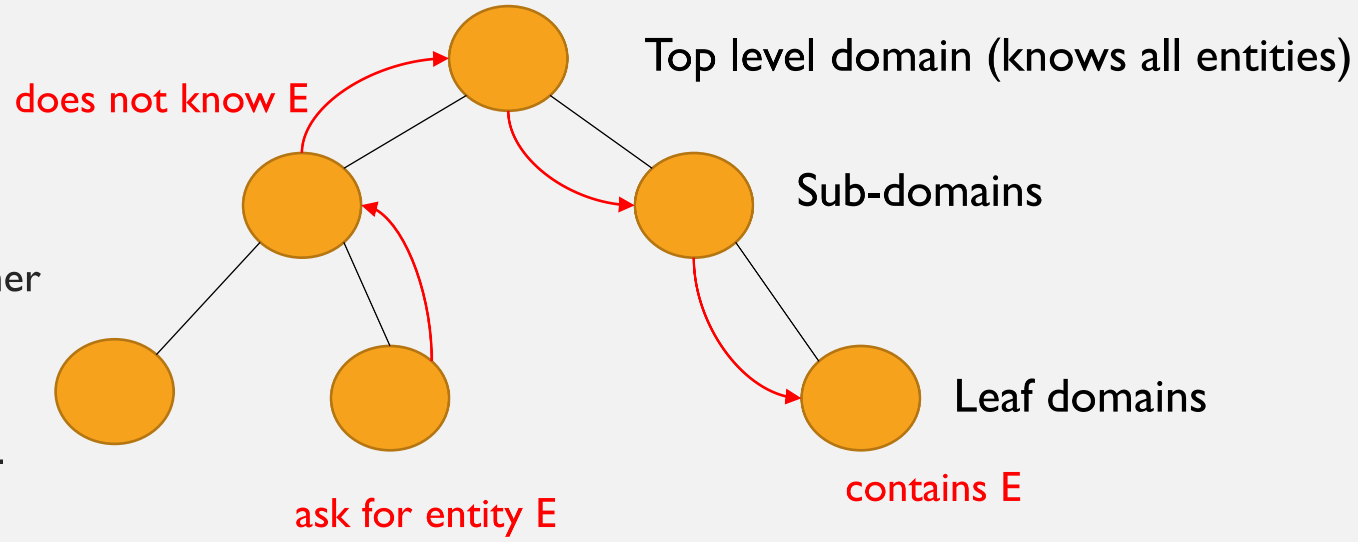
1. succ(4) = 5
2. Node 3 inserts itself
3. keys(3) = {1, 2, 3}

| | |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 5 |
| 4 | 9 |

keys(0)={10, 11, 12, 13, 14, 15, 0}

# FLAT NAMING

- Last approach: hierarchical naming
- Based on *domains.*
- Each domain contains entities.
- When looking up an entity
  a. Entity is not there → ask higher level domain.
  b. Entity is there → address of entity itself or pointer to sub-domain.

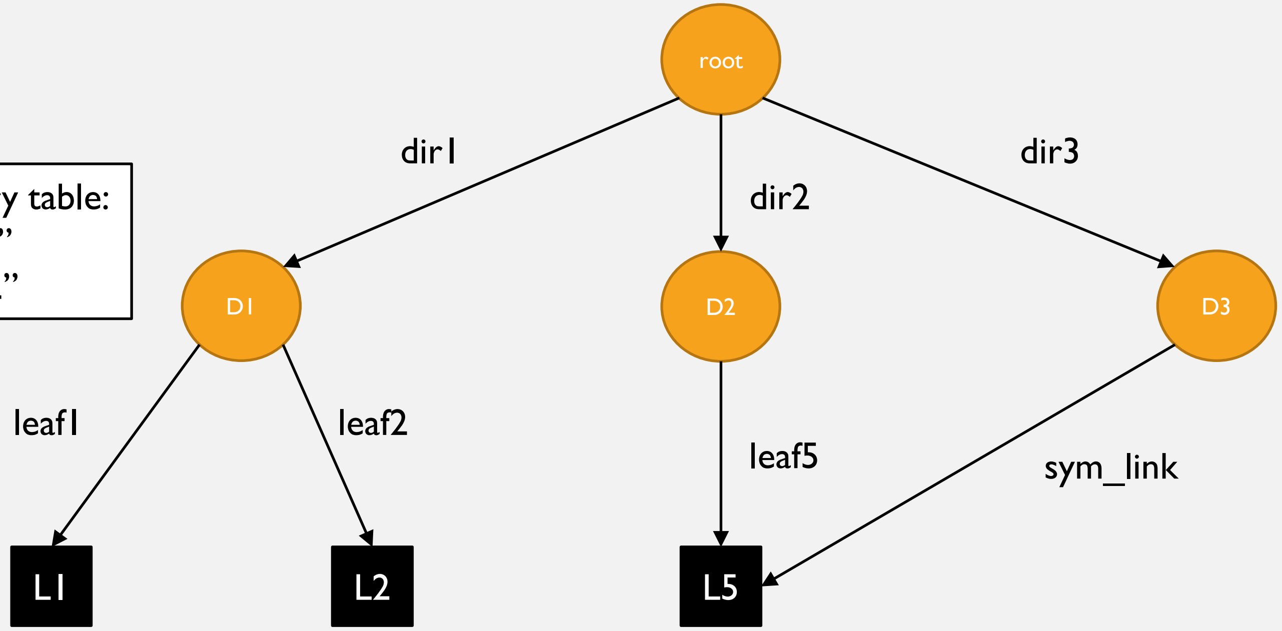


# STRUCTURED NAMING

- Flat names useful for machines, but normally not meaningful for humans.
- How to generate unique names that are human readable?
- Solution: use name spaces.
- Name spaces can be visualized as name graphs (labeled directed graphs) with:
  - Root node: node without ingoing edges.
  - Directory nodes: nodes with outgoing edges → stores directory tables.
  - Leaf nodes: nodes without outgoing edges → entities.
    - Value can be the address of the entity or the entity itself (e.g. a file).

# STRUCTURED NAMING

Directory table:
L1: "leaf1"
L2: "leaf2"




Name = {dir1, leaf1}   Name = {dir2, leaf2}   Name = {dir2, leaf5}, {dir3, sym_link}

# STRUCTURED NAMING

- Name resolution
  - Begin with root node.
  - Lookup directory table to find the node $N_1$ the first label refers to.
  - Contact node $N_1$ and look up next label in directory table to find node $N_2$.
  - Contact node $N_2$ and look up the next label in directory …
  - … until leaf node reached or entity non-existent

# STRUCTURED NAMING

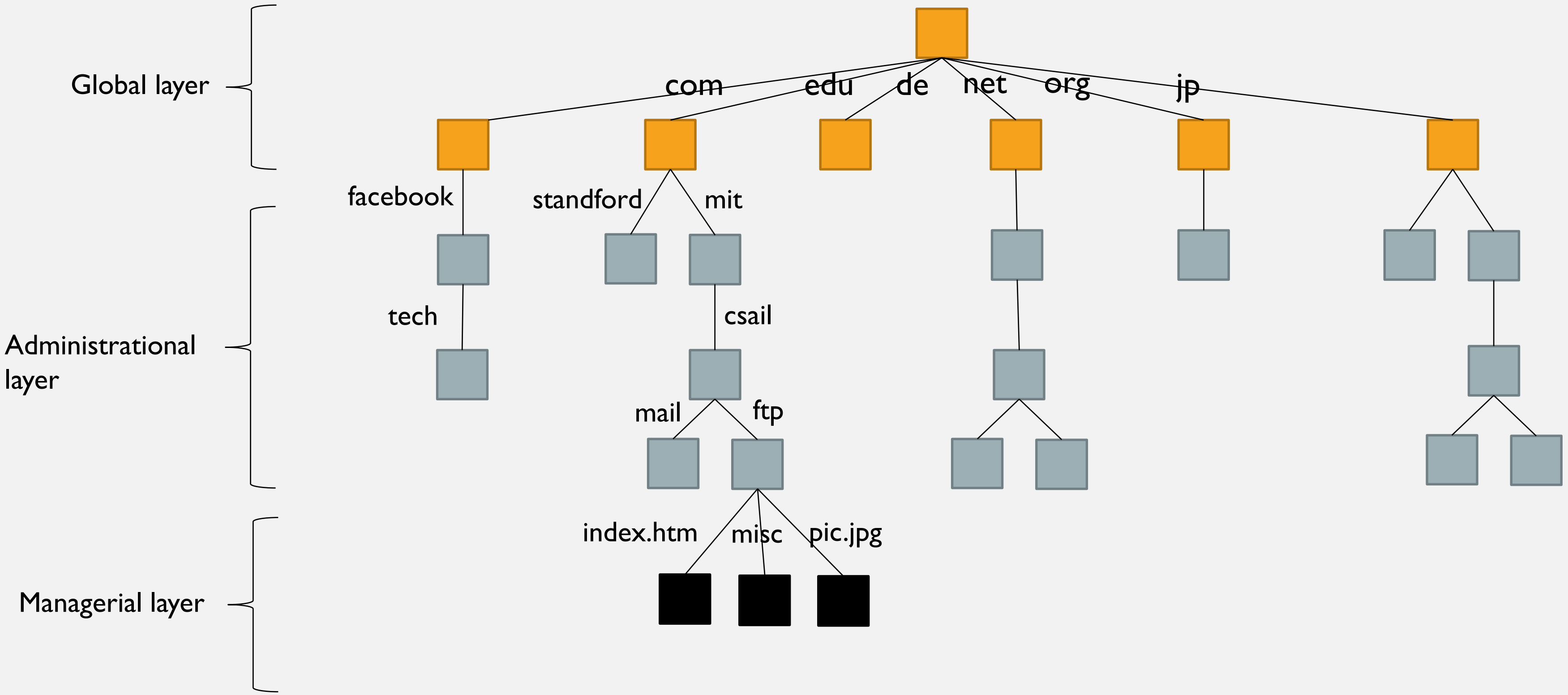
- Naming implemented in distributed systems by **name servers.**
- Number of name servers depending on geographical spread of the system.
- Distributed name spaces are organized into logical layers:
  - **Global layer:** root node and its direct children. Most stable layer (directory tables are rarely changed).
  - **Administrational layer:** directory nodes maintained by a single organization (i.e. a directory node per department inside an organization). Changes more than global layer.
  - **Managerial layer:** Nodes represent entities that regularly change (i.e. shared files or directories). Managed also by end-users.

# STRUCTURED NAMING

- Example: **Domain Name System (DNS).**
- Translation name (URL) → IP address.
- One of the backbone services of the Internet.
- Current 13 logical root name servers
  - Each one operates redundantly (high availability).
- Normally uses UDP on port 53.
- **Domain:** subtree of the DNS naming graph.
- **DNS Zone:** contiguous portion of the domain space which is administrated by a single entity (with a name server). A domain can be portioned into zones.

# STRUCTURED NAMING

- Example: **Domain Name System (DNS)**




- High availability (replication)
- Client-side caching (top level domains do not change often)
- Update performance not critical

- Availability requirements depend on organization
- Update performance important

- Availability not critical
- Update performance critical

# STRUCTURED NAMING

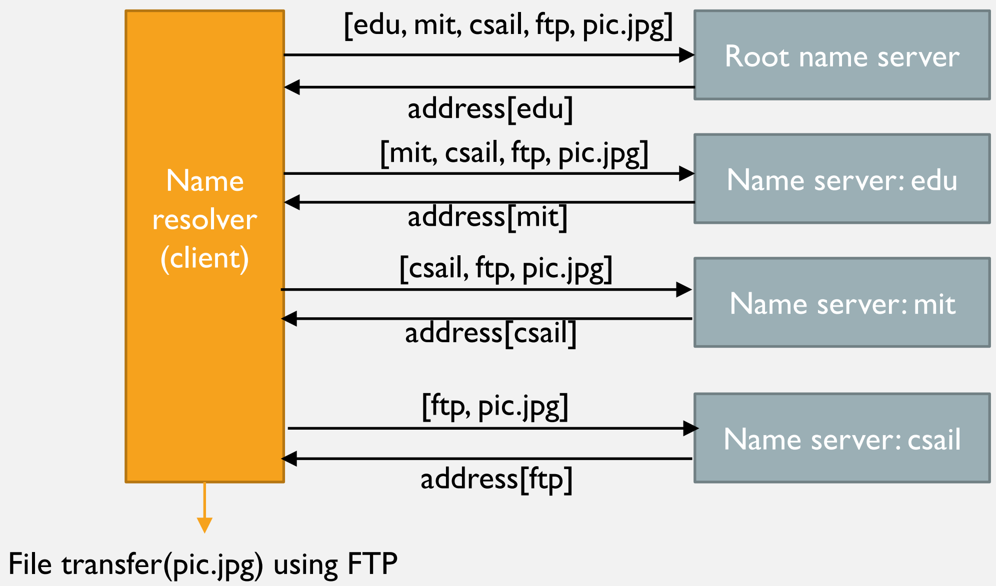
**Domain Name Service (DNS)**

- Each zone is implemented by a name server.
- A name server contains a zone file with records for all nodes in a zone.
- DNS record types: relate a domain name with certain data
  - SOA (Start of authority): metadata about the zone (administration).
  - A (address): the IP address of a particular host.
  - NS (name server): the IP address of a name server implementing the zone.
  - MX (mail exchange): the IP address of a mail server.
  - SRV (service): record for a specific service.

# STRUCTURED NAMING

DNS name resolution: [edu, mit, csail, ftp, pic.jpg] with ftp (ftp://ftp.csail.mit.edu/pic.jpg)

1. Iterative name resolution



# STRUCTURED NAMING

DNS name resolution: [edu, mit, csail, ftp, pic.jpg] with ftp (ftp://ftp.csail.mit.edu/pic.jpg)

2. Recursive name resolution