Lecture 7.1

Application Layer: Domain Name System(DNS)

Dr. Vandana Kushwaha

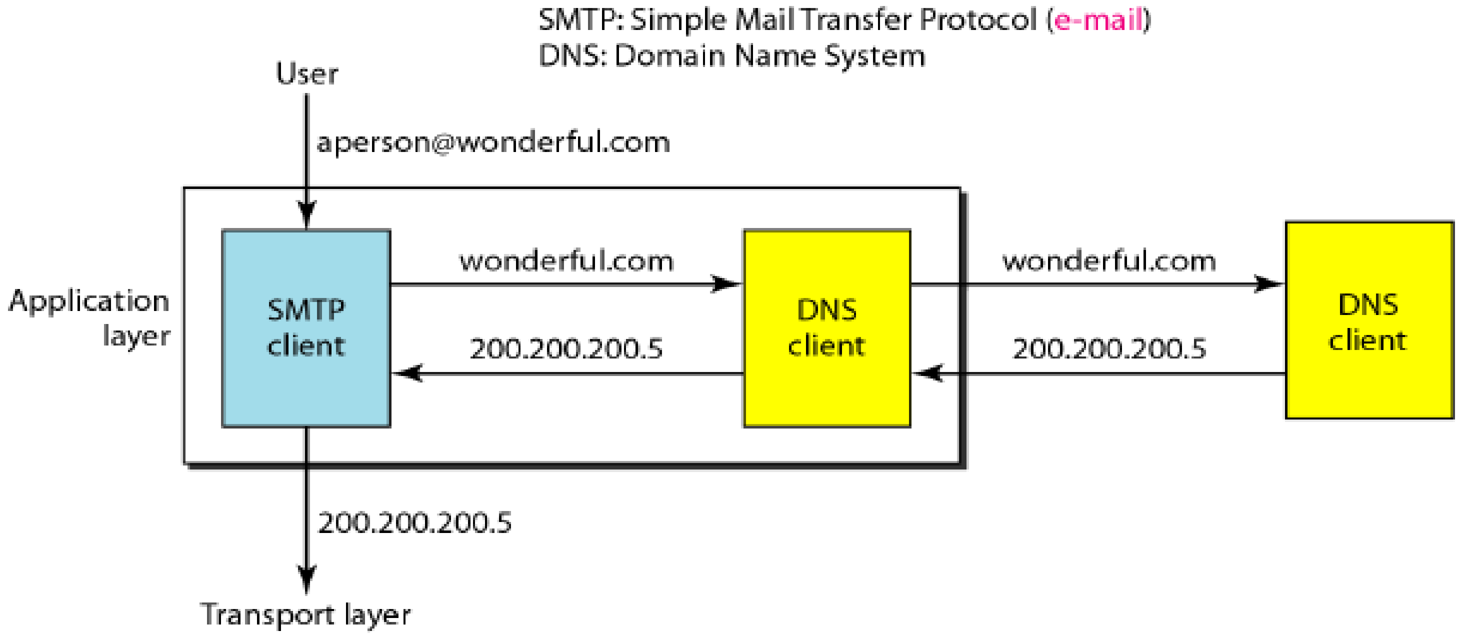
Department of Computer Science

Institute of Science, BHU, Varanasi

Introduction

- There are several **applications** in the **application layer** of the **Internet model** that follow the **client/server paradigm**.
- The **client/server programs** can be divided into **two categories**: those that can be **directly used** by the **user**, such as **e-mail**, and those that **support other application programs.**
- The **Domain Name System (DNS)** is a **supporting program** that is used by other programs such as **e-mail.**
- A user of an **e-mail** program may know the **e-mail address** of the **recipient;** however, the **IP protocol** needs the **IP address.**
- The **DNS client** program **sends** a **request** to a **DNS server** to **map** the **e-mail address** to the **corresponding IP address.**
- **Figure** on next slide shows an **example** of how a **DNS client/server** program can support an **e-mail program** to find the **IP address** of an **e-mail recipient**.

Example of using the DNS service



Domain Name System

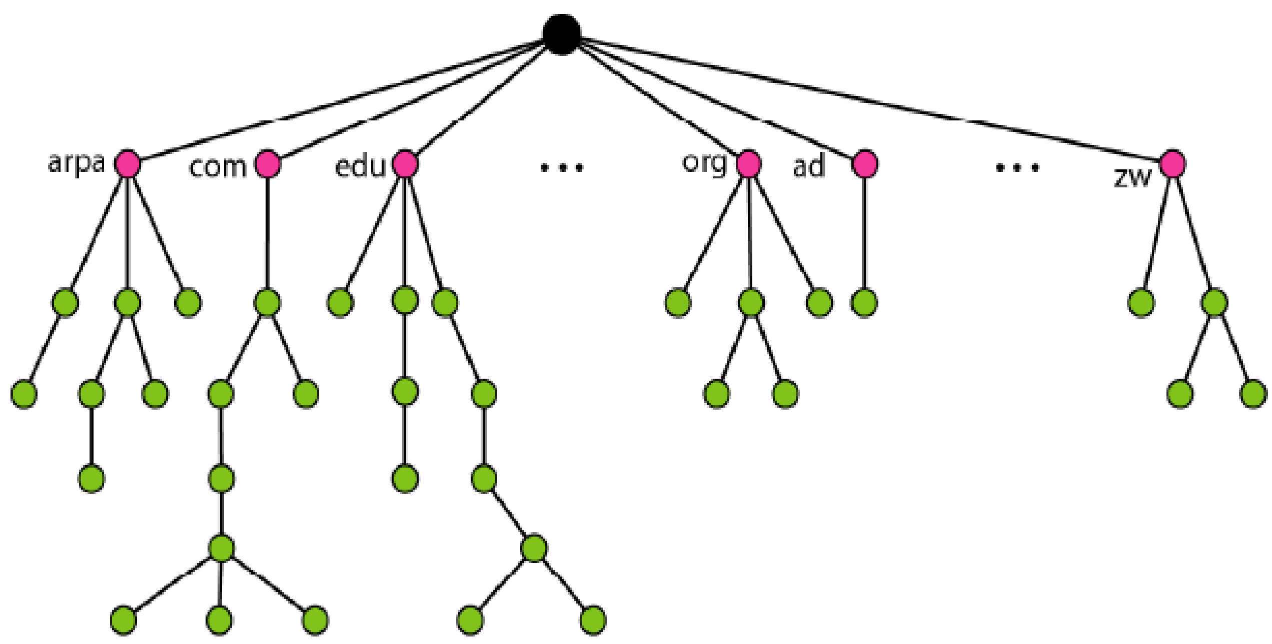
- To **identify** an **entity**, **TCP/IP protocols** use the **IP address**, which **uniquely identifies** the connection of a **host** to the **Internet.**
- However, people **prefer** to use **names instead of numeric addresses**.
- Therefore, we **need a system** that can **map a name to an address** or **an address to a name.**
- When the **Internet** was **small, mapping** was **done** by using a **host file.**
- The **host file** had only **two columns**: **name and address**.
- **Every host** could **store** the **host file** on its **disk** and **update** it **periodically** from a master host file.
- When a **program** or a **user** wanted to **map** a **name** to an **address**, the **host** consulted the **host file** and found the **mapping.**
- Today, however, it is **impossible** to have **one single host file** to **relate every address** with a **name** and **vice versa.**

Domain Name System

- The **host file** would be **too large to store in every host.**
- In addition, it would be **impossible** to **update all** the **host files every time** there was a **change.**
- One **solution** would be to **store** the **entire host file in a single computer** and allow **access** to this **centralized information** to **every computer** that needs **mapping.**
- But it would **create a huge amount of traffic** on the **Internet.**
- Another **solution**, the one used today, is to **divide this huge amount of information into smaller parts and store each part on a different computer.**
- In this **method**, the **host** that needs **mapping** can contact the **closest computer holding** the needed information.
- This **method** is used by the **Domain Name System (DNS)**.

DOMAIN NAME SPACE

- A **domain name space** was designed using **hierarchical name space.**
- In this **design** the **names are defined** in an **inverted-tree structure** with the **root** at the **top**.
- The **tree** can have only **128 levels**: **level 0 (root)** to **level 127.**



DOMAIN NAME SPACE

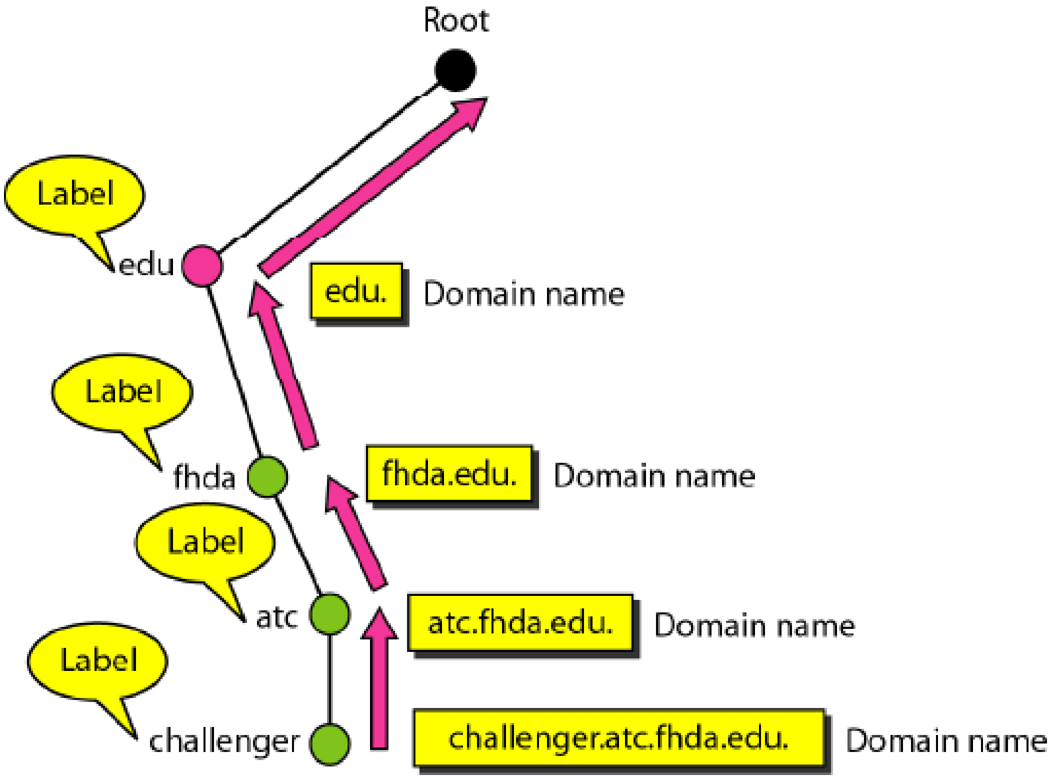
Label

- Each **node** in the **tree** has a **label**, which is a **string** with a maximum of **63 characters.**
- The **root label** is a **null string** (empty string).
- **DNS** requires that **children of a node** (nodes that branch from the same node) have **different labels**, which guarantees the **uniqueness** of the **domain names.**

Domain Name

- Each **node** in the **tree** has a **domain name.**
- A **full domain name** is a **sequence of labels separated by dots (.)**
- The **domain names** are always **read from** the **node up to the root**.
- The **last label** is the **label of the root** (null).
- This means that a **full domain name** always **ends in a null label**, which means the **last character** is a **dot** because the **null string** is nothing.

Domain Name


Root
Label
edu
edu.
Domain name
Label
fhda
fhda.edu.
Domain name
Label
atc
atc.fhda.edu.
Domain name
Label
challenger
challenger.atc.fhda.edu.
Domain name

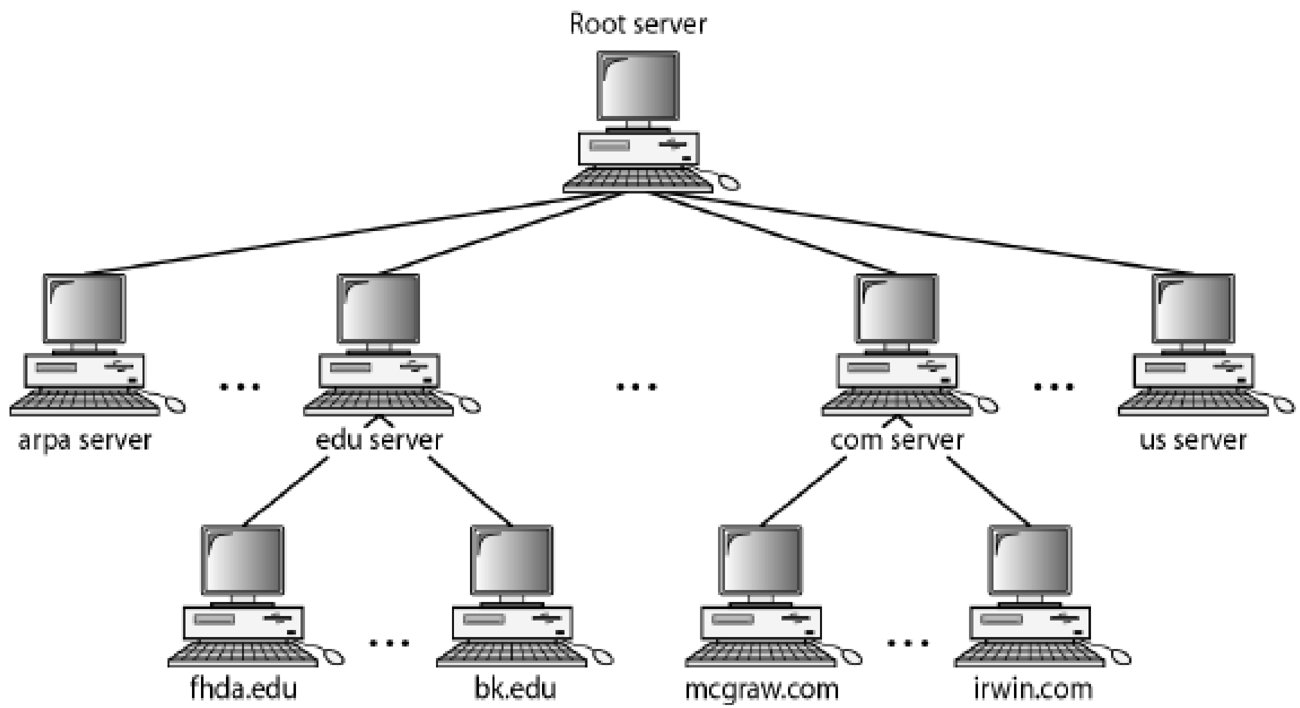
Distribution of Name Space

- The **information** contained in the **domain name space** must be **stored somewhere.**
- However, it is very **inefficient** and also **unreliable** to have just **one computer** store such a **huge amount of information**.
- It is **inefficient** because responding to requests from **all over the world** places a **heavy load** on the system.
- It is **unreliable** because any **failure** makes the **data inaccessible**.

Hierarchy of Name Servers

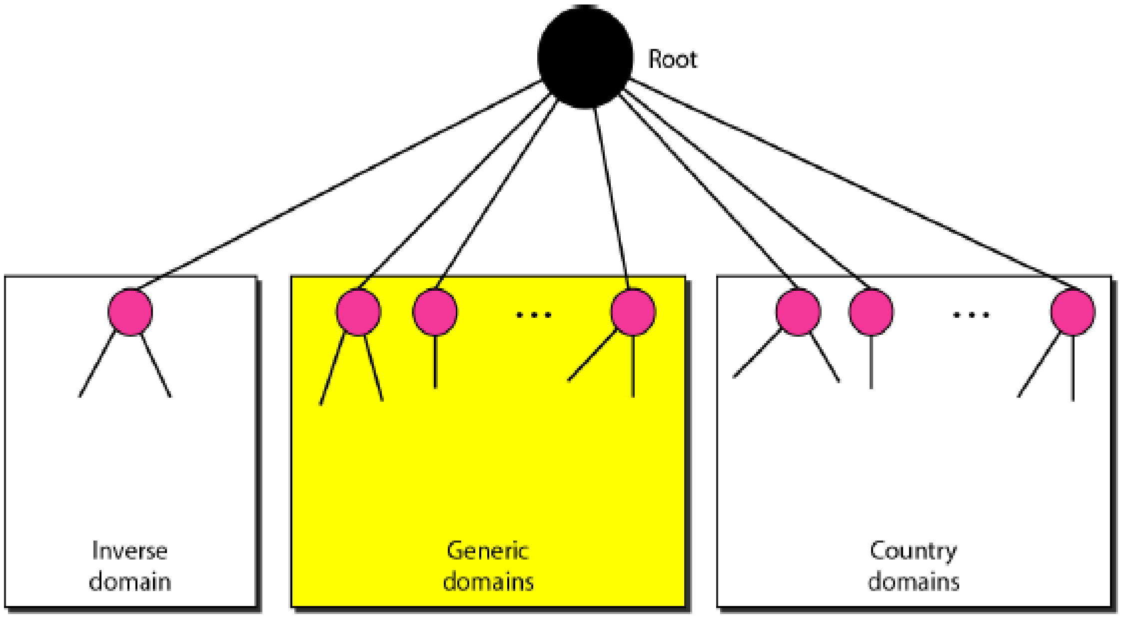
- The **solution** to these **problems** is to **distribute the information** among **many computers** called **DNS servers**.
- One way to do this is to **divide the whole space** into **many domains** based on the **first level.**
- In other words, we let the **root** stand alone and create as **many domains (subtrees)** as there are **first-level nodes.**
- Because a **domain** created in this way could be **very large**, **DNS** allows domains to be **divided further** into **smaller domains (subdomains).**
- Each **server** can be **responsible** (authoritative) for a **small domain.**
- In other words, we have a **hierarchy of servers** in the same way that we have a **hierarchy of names.**

Hierarchy of name servers



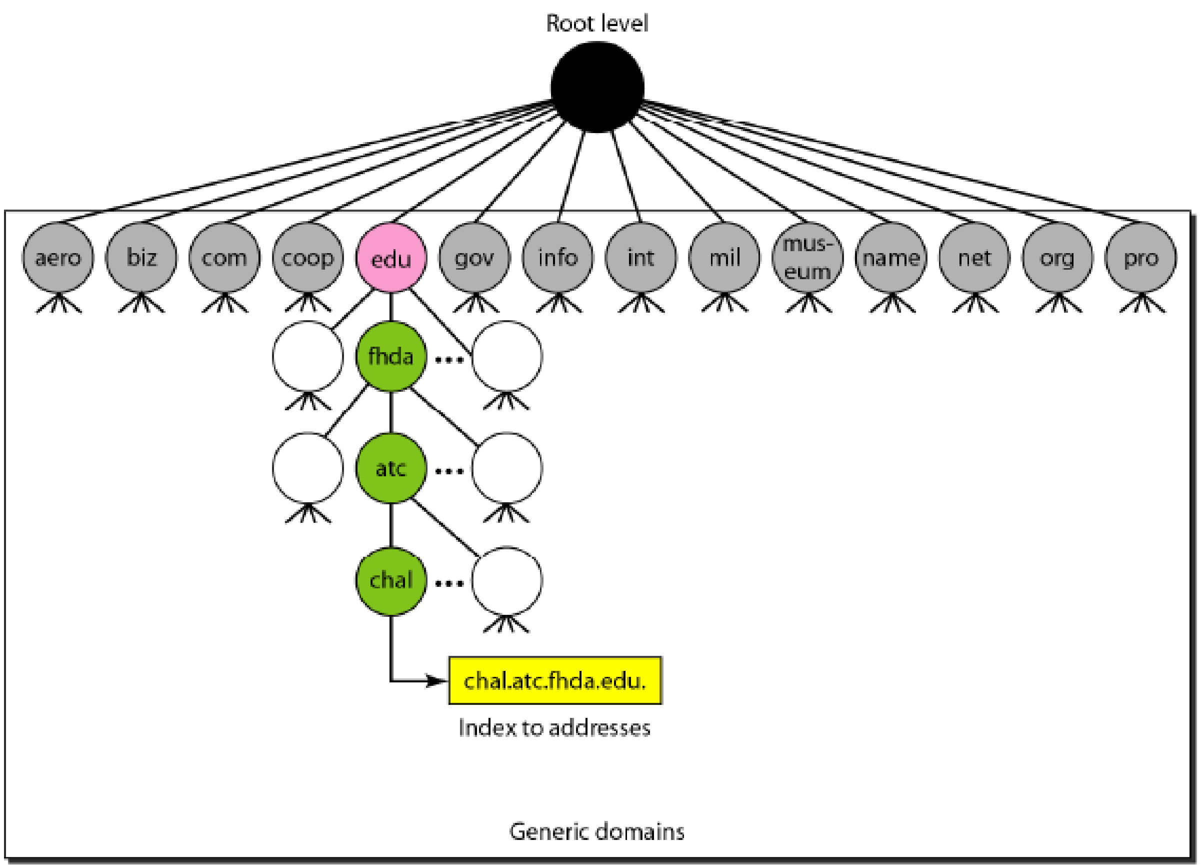
DNS IN THE INTERNET

- **DNS is a protocol** that can be used in **different platforms.**
- In the Internet, the **domain name space (tree)** is divided into **three different sections:** ***Generic domains, Country domains***, and the ***Inverse domain.***



Generic Domains

- The **Generic domains** define **registered hosts** according to their **generic behavior**.
- **Each node** in the **tree** defines a **domain**, which is an **index** to the **domain name space database.**



Generic Domains

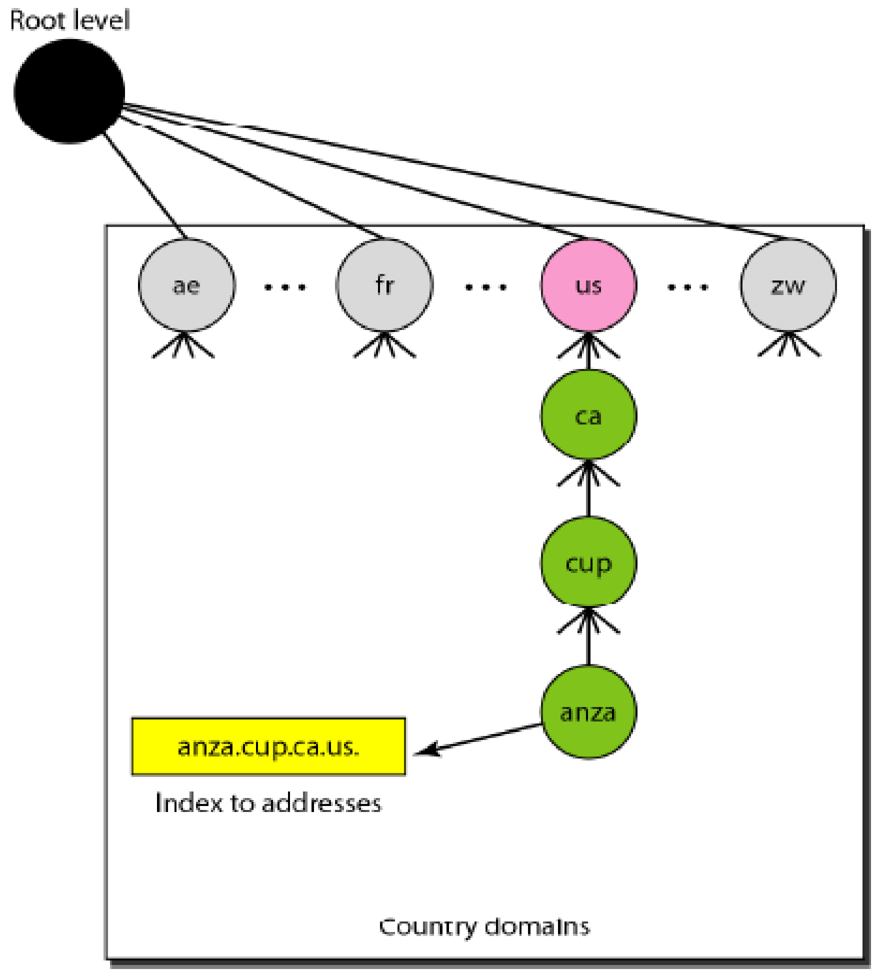
- **First level** in the **generic domains** section allows **14 possible labels**.
- These **labels** describe the **organization types** as listed in **Table** below.

Label	*Description*
aero	Airlines and aerospace companies
biz	Businesses or firms (similar to "com")
com	Commercial organizations
coop	Cooperative business organizations
edu	Educational institutions
gov	Government institutions
info	Information service providers
int	International organizations
mil	Military groups
museum	Museums and other nonprofit organizations
name	Personal names (individuals)
net	Network support centers
org	Nonprofit organizations

Country Domains

- The **country domains** section uses **two-character country abbreviations** (e.g., **us** for United States).
- **Second labels** can be **organizational**, or they can be more specific, national designations.
- The **United States**, for **example,** uses **state abbreviations** as a **subdivision** of us (e.g., **ca.us.**).
- The address ***anza.cup.ca.us*** can be translated to **De Anza College in Cupertino, California,** in the **United States.**

Country Domains



Inverse Domain

- The **inverse domain** is used to **map** an **address** to a **name**.
- This may happen, for **example**, when a **server** has received **a request** from a **client** to do a **task.**
- Although the **server** has a **file** that contains a **name list of authorized clients.**
- The **server** asks its **resolver** to **send** a **query** to the **DNS server** to **map an address to a name** to **determine** if the **client is on the authorized list.**

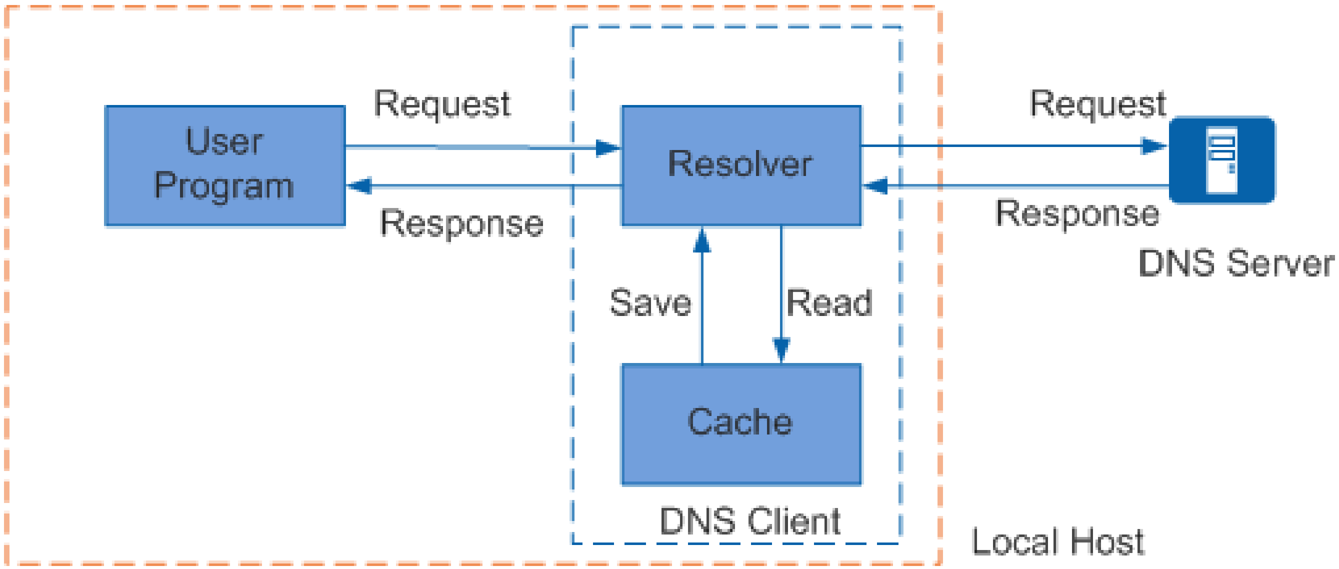
Address Resolution

- **Mapping** a **name** to an **address** or an **address** to a **name** is called ***name-address resolution.***

Resolver

- **DNS** is designed as a **client/server application**.
- A **host** that needs to **map** an **address** to a **name** or a **name** to an **address** calls a **DNS client called a resolver**.
- The **resolver** accesses the **closest DNS server** with a **mapping request**.
- If the **server** has the **information**, it **satisfies** the **resolver**; otherwise, it either **refers** the **resolver** to **other servers** or asks **other servers** to provide the information.
- After the **resolver receives** the **mapping**, it **interprets** the **response** to see if it is a **real resolution** or an **error,** and **finally delivers** the **result** to the **process** that **requested it.**

DNS Resolver

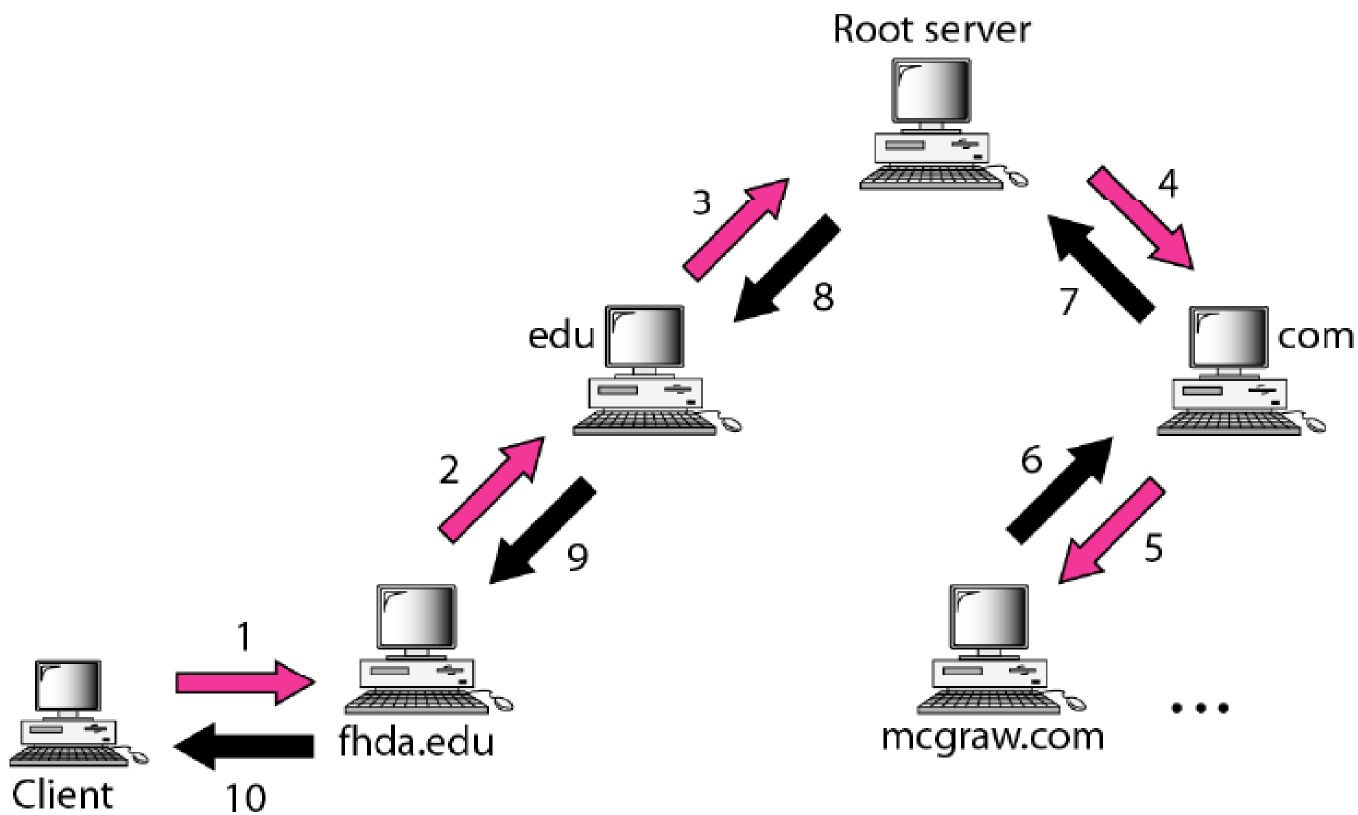


Address Resolution

Recursive Resolution

- The **client (resolver)** can ask for a **recursive answer** from a **name server.**
- This means that the **resolver expects** the **server** to **supply** the **final answer.**
- If the **server** is the authority for the **domain name**, it **checks** its **database** and **responds.**
- If the **server** is **not** the authority, it **sends** the **request** to **another server** (the parent usually) and **waits** for the **response.**
- If the **parent** is the **authority**, it **responds**; otherwise, it **sends** the **query** to yet **another server.**
- When the **query** is **finally resolved**, the **response travels back** until it **finally reaches** the **requesting client.**

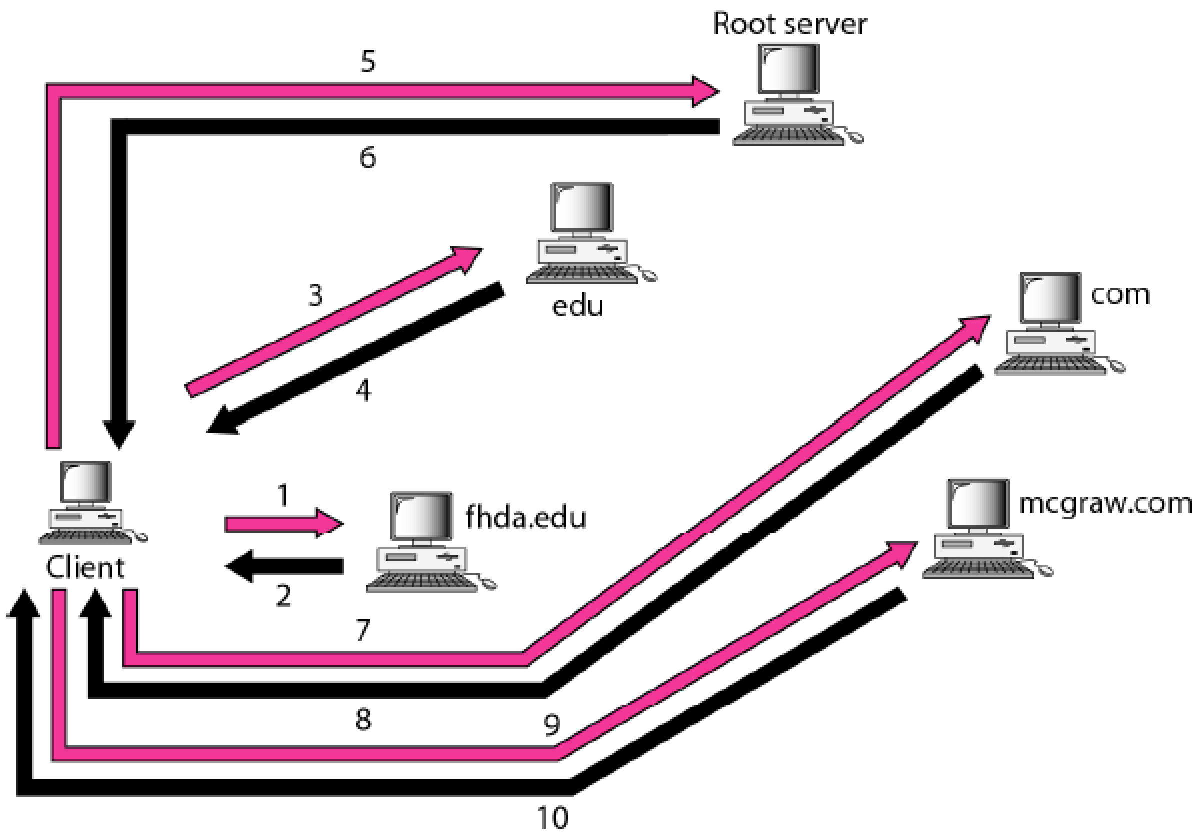
Recursive Resolution



Iterative Resolution

- If the **client** does **not ask** for a **recursive answer**, the **mapping** can be done **iteratively.**
- If the **server** is an **authority** for the **name**, it **sends** the **answer.**
- If it is **not**, it **returns** (to the client) the **IP address** of the **server that it thinks can resolve the query.**
- The **client** is **responsible** for **repeating** the **query** to this **second server.**
- If the **newly addressed server** can **resolve** the **problem**, it **answers** the **query** with the **IP address**; otherwise, it **returns** the **IP address** of a **new server** to the **client.**
- Now the **client** must **repeat the query** to the **third server.**
- This **process** is called **iterative resolution** because the client **repeats the same query** to **multiple servers.**

Iterative Resolution



Caching

- **Each time** a **server receives** a **query** for a **name** that is **not** in **its domain**, it **needs** to **search** its **database** for a server IP address.
- **Reduction** of this **search time** would **increase efficiency.**
- **DNS** handles this with a **mechanism** called **caching.**
- When a **server** asks for a **mapping** from **another server** and **receives** the **response,** it **stores this information** in its **cache memory** before **sending** it to the **client.**
- If the **same** or **another client** asks for the **same mapping**, it can **check** its **cache memory** and **solve** the **problem.**
- **Caching speeds up resolution**, but it can also be **problematic.**
- If a **server caches** a **mapping** for a **long time**, it may send an **outdated mapping** to the **client.**
- To **counter** this, **two techniques** are used.

Caching

- **First,** the **authoritative server** always adds information to the **mapping** called ***time-to-live*** **(TTL).**
- It **defines** the **time** in **seconds** that the **receiving server** can **cache** the **information.**
- **After that time**, the **mapping** is **invalid** and any **query must** be **sent again** to the **authoritative server.**
- **Second, DNS requires** that **each server** keep a **TTL counter** for **each mapping** it **caches.**
- The **cache memory** must be **searched periodically**, and those **mappings** with an **expired TTL** must be **purged.**

ENCAPSULATION

- **DNS** can use either **UDP or TCP.** In both cases the **well-known port** used by the server is **port 53.**
- **UDP** is used when the **size** of the **response message** is **less than 512 bytes** because most **UDP packages** have a **512-byte packet size limit**.
- If the **size** of the **response message** is **more than 512 bytes**, a **TCP** connection is used.
- In that case, one of two scenarios can occur:

Case 1:

- If the **resolver** has **prior knowledge** that the **size** of the **response message** is **more than 512 bytes**, it uses the **TCP connection.**

ENCAPSULATION

Case 2:

- If the **resolver does not know** the **size** of the **response message**, it can use the **UDP port.**
- However, if the **size** of the **response message** is **more than 512 bytes**, the **server truncates** the **message** and **turns on** the **TC bit.**
- The **resolver** now **opens a TCP connection** and **repeats** the **request** to **get** a **full response** from the **server.**
- **Note: TC** *Truncated* : **1 bit** in **DNS header**, response **too large for UDP (1).**