Domain Name System

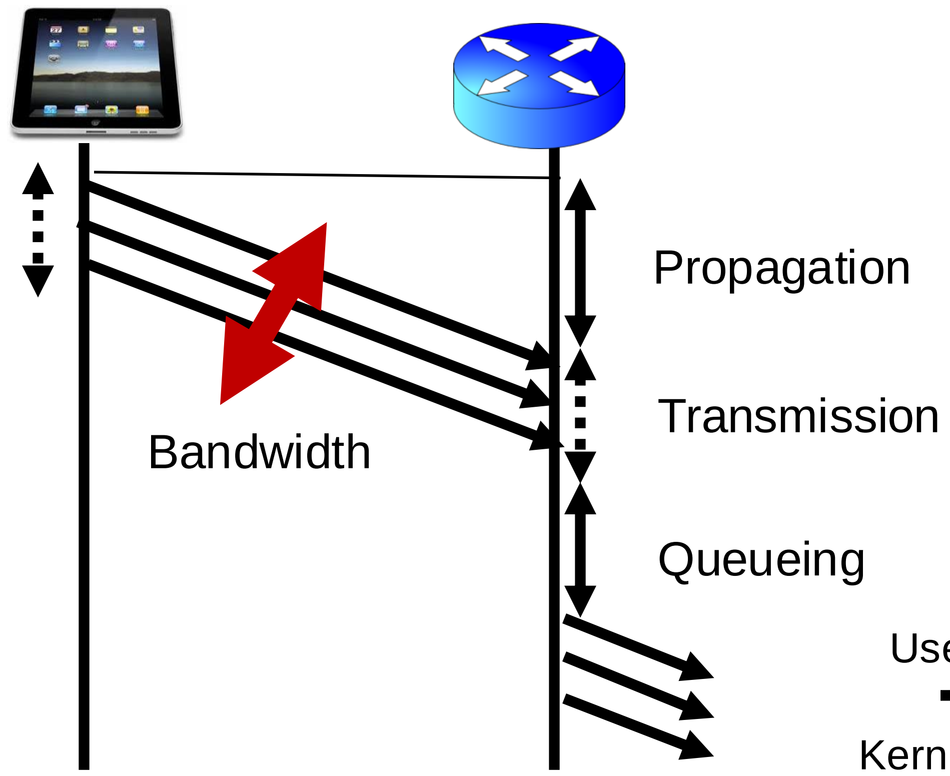
Lecture 4

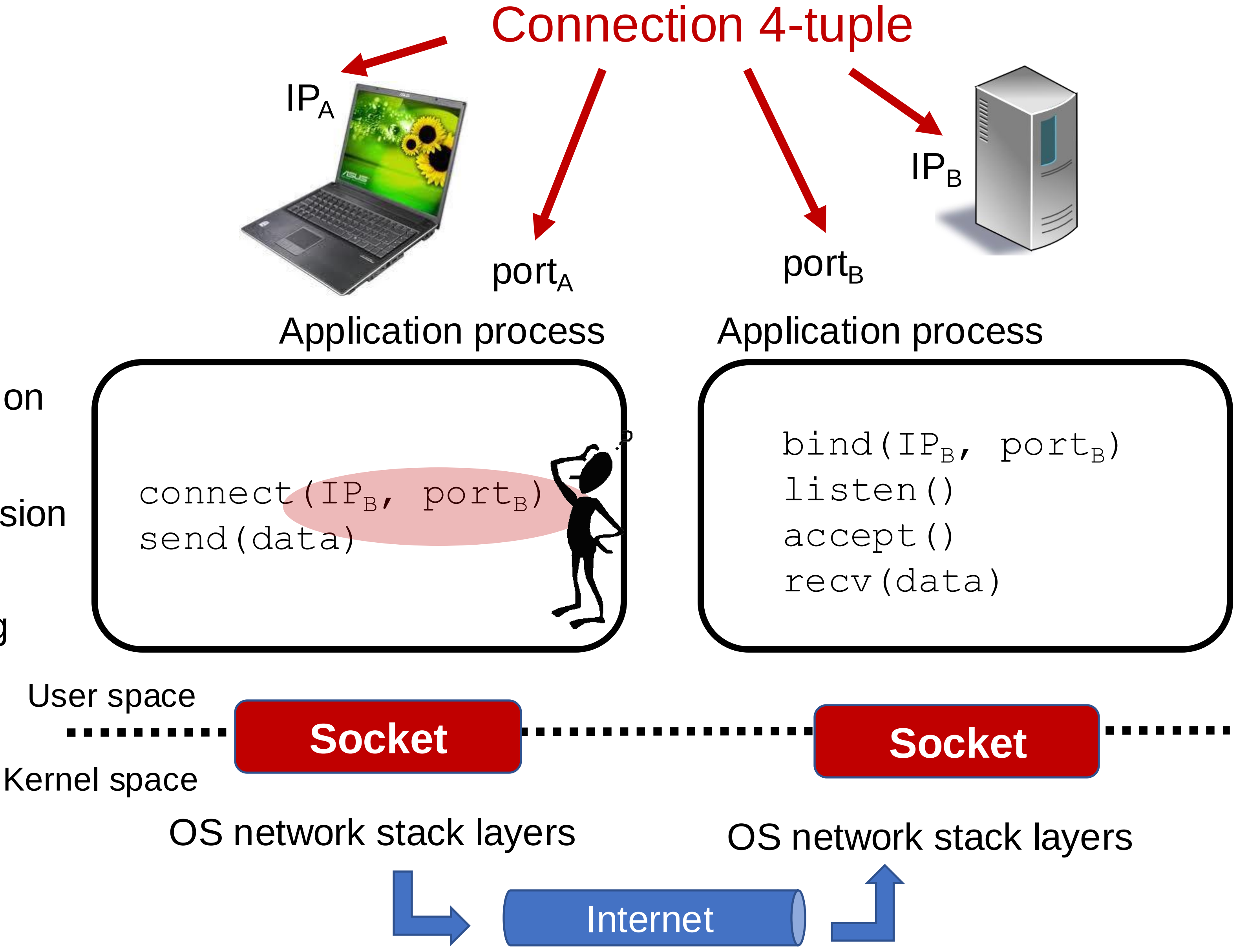
http://www.cs.rutgers.edu/~sn624/352-F24

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

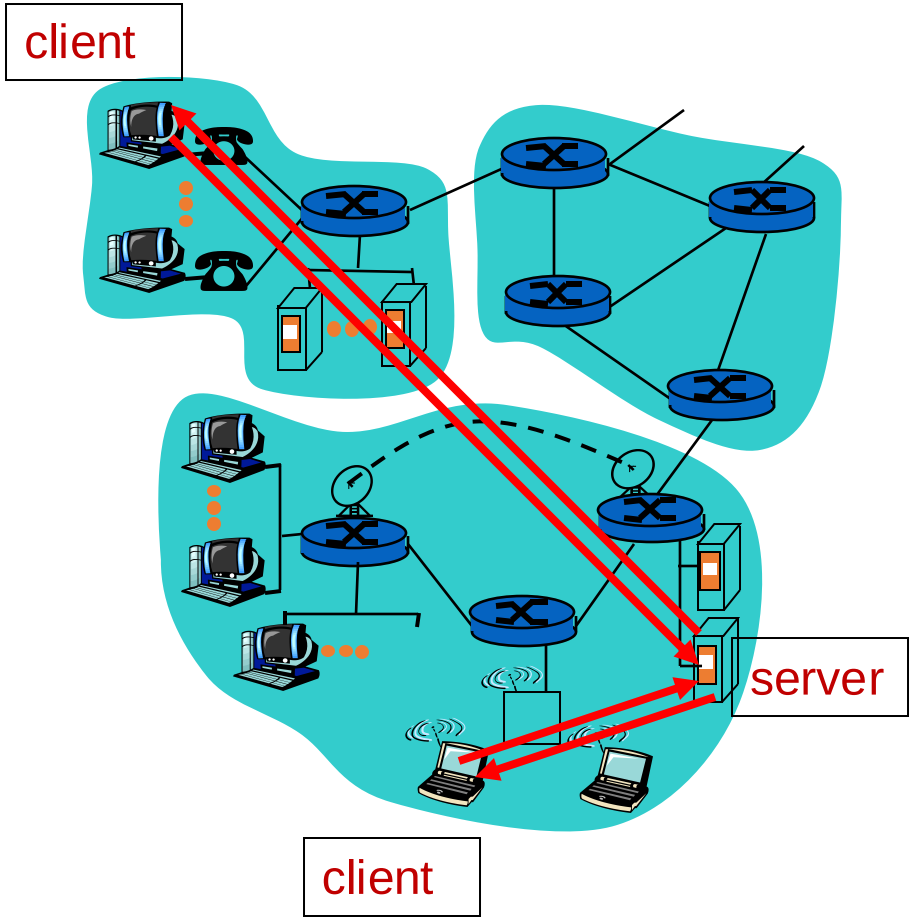
Review


Propagation
Transmission
Queueing
Bandwidth


Connection 4-tuple
IP_A
$port_A$
$port_B$
IP_B
Application process
Application process
connect(IP_B, $port_B$)
send(data)
bind(IP_B, $port_B$)
listen()
accept()
recv(data)
User space
Kernel space
Socket
Socket
OS network stack layers
OS network stack layers
Internet

Common Architectures of Applications

Client-server architecture



Server:

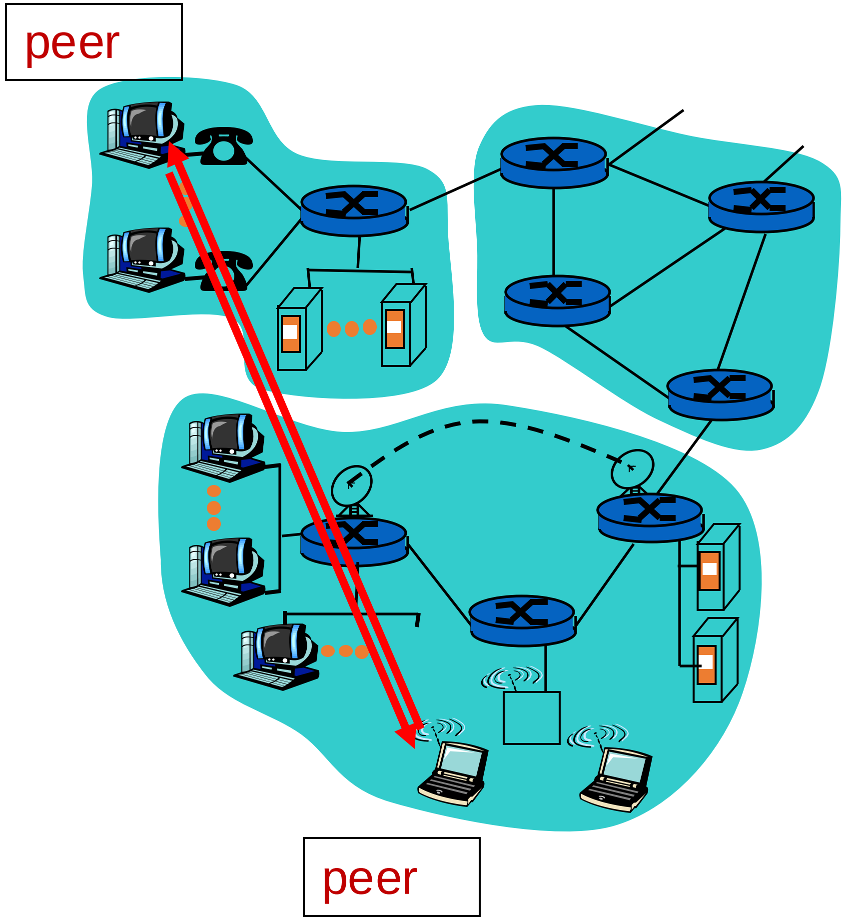
- Always-on endpoint
- Provides a “service” to the world
- Typically, a permanent IP address
- Hosted in clusters to scale to many users

Clients:

- A “customer” of the service
- Maybe intermittently connected
- May have dynamic IP addresses
- Typically, do not communicate directly with other clients

- The web and most mobile apps use a client-server architecture

Peer-to-peer (P2P) architecture



- Peers:
  - Intermittently connected hosts
  - Directly talking to each other
- Little to no reliance on always-up servers
  - Examples: BitTorrent
- Today, many applications use a hybrid model: servers to set up connectivity, communicate directly afterward
  - Example: (webRTC) Google meet

Going forward: A few app-layer protocols

- Domain Name System
- The web
- Streaming video

Domain Name System

Domain Name System (DNS)

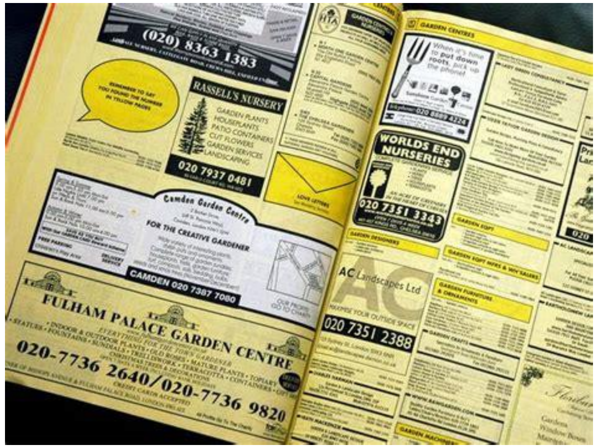
- Problem: Humans cannot remember Internet (IP) addresses
  - The average human brain can remember 7 digits for a few names
  - On average, IP addresses have 12 digits

- Solution: Use human-friendly names to refer to endpoints
  - Alphanumeric names (e.g. www.cs.rutgers.edu)
  - Called host names or domain names

- A new problem! We need a directory (address book) to translate human-friendly names to IPs

You have a name. Can you lookup an address?

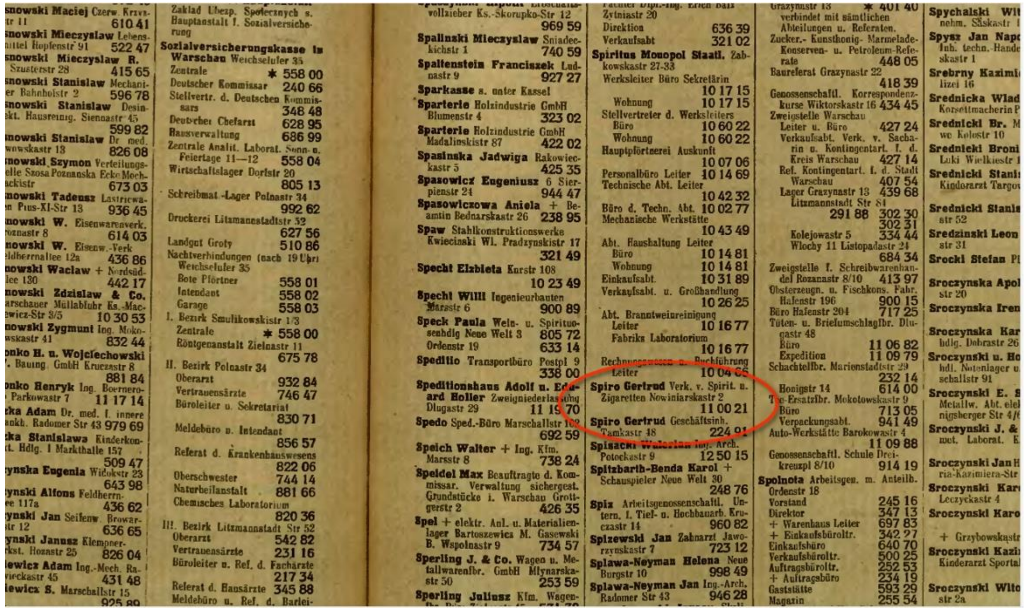
Directories



- Directories map a *name* to an *address*
- We call this process Address Resolution
- Simplistic designs
  - Central directory
  - Ask everyone (flooding)
  - Tell everyone (e.g., push to a file like /etc/hosts)
- Scalable distributed designs
  - Hierarchical namespace (e.g., Domain Name System (DNS))
  - Flat namespace (e.g., Distributed Hash Table)

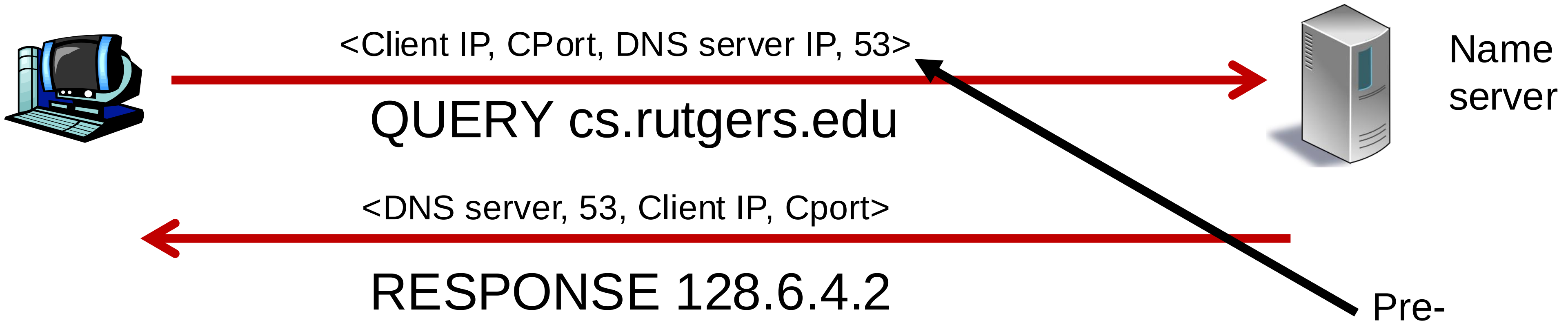
Simple DNS ("tell everyone")

- What if every endpoint has a local directory?
- `/etc/hosts.txt:` how DNS worked in the early days of the Internet
- Q: What if endpoints changed addresses? How do you keep this up to date?



Simple DNS

DOMAIN NAME	IP ADDRESS
spotify.com	98.138.253.109
cs.rutgers.edu	128.6.4.2
www.google.com	74.125.225.243
www.princeton.edu	128.112.132.86



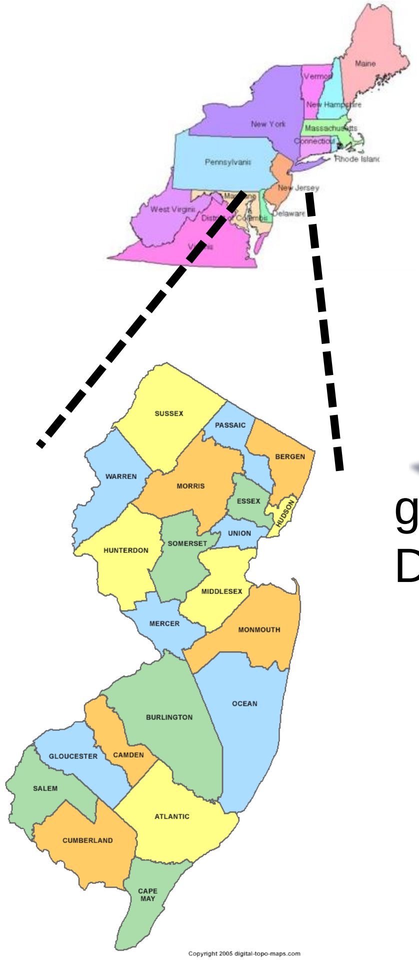
- Key idea: Implement a server that looks up a table.
- Would this work?
  - Every new (changed) host needs to be entered in this table
  - Performance: can the server serve billions of Internet users
  - Failure: what if the server or the database crashes?
  - How to secure this server?

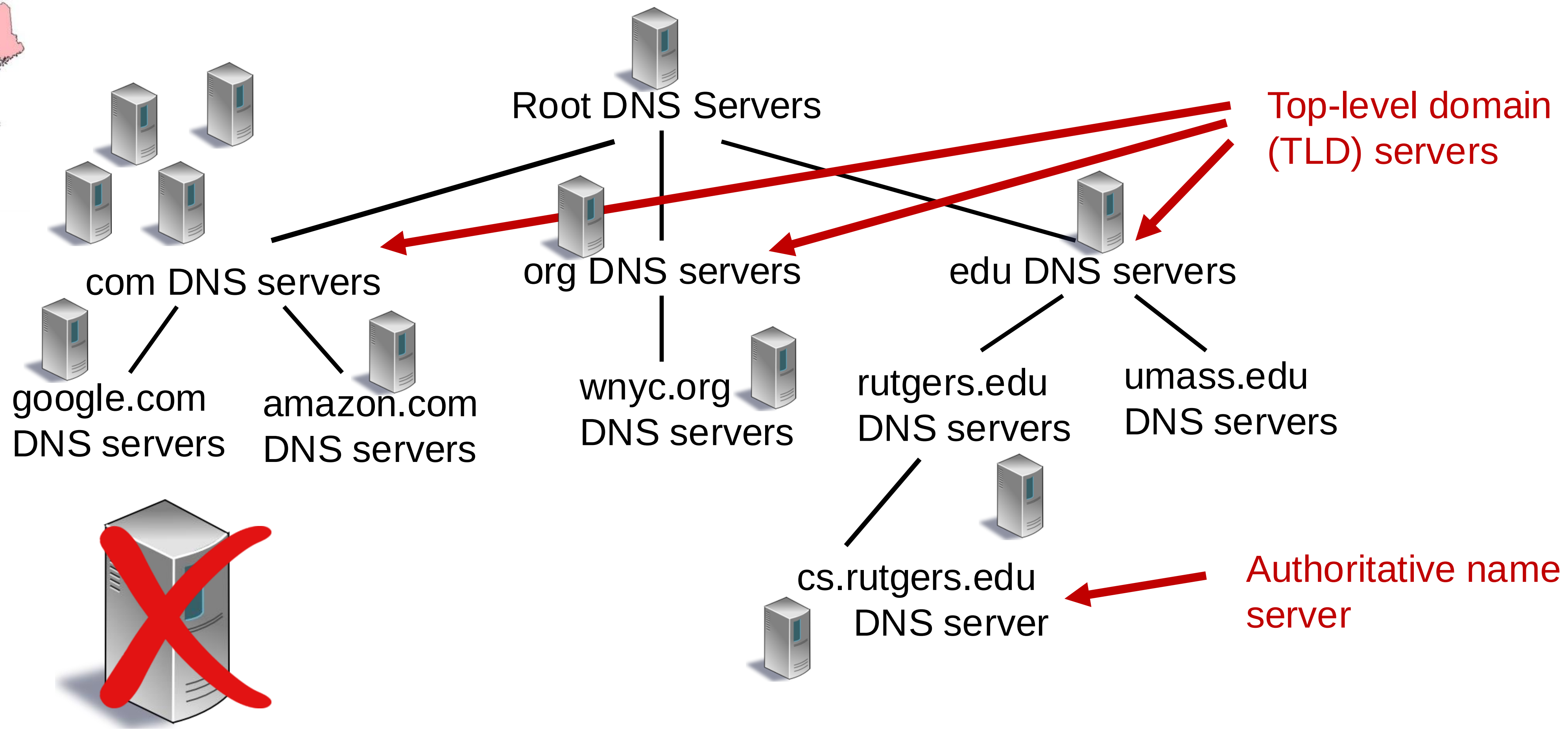
Ideas to make DNS work for the Internet

- Idea #1. Hierarchy
  - Organize names hierarchically so we can divide the work of resolution
  - Internet: some names under “.com”, others with “.org”, “.edu”, …
  - Called top-level domains (TLD)
  - TLDs may contain sub-domains, sub-sub-domains, …
  - Lowest level: fully qualified domain name (e.g. people.cs.rutgers.edu)

- Idea #2. Distribution
  - Each node in the hierarchy served separately (name servers)
  - Lowest level: Manage changes in IP addresses of endpoints
    - Authoritative name server

Distributed and Hierarchical database





RFC 1034

Hierarchy

Replication

DNS Protocol

- Client-server application
- Client connects to (known) port 53 on server
- For now, assume the DNS server IP is known
- Two types of messages
  - Queries
  - Responses
- Type of Query (OPCODE)
  - Standard query (0x0)
    - e.g., Request IP address for a given domain name
  - Updates (0x5)
    - Provide a binding of IP address to domain name

DNS in action

- `dig <domain-name>`
- `dig +trace <domain-name>`
- `dig @<dns-server> <domain-name>`

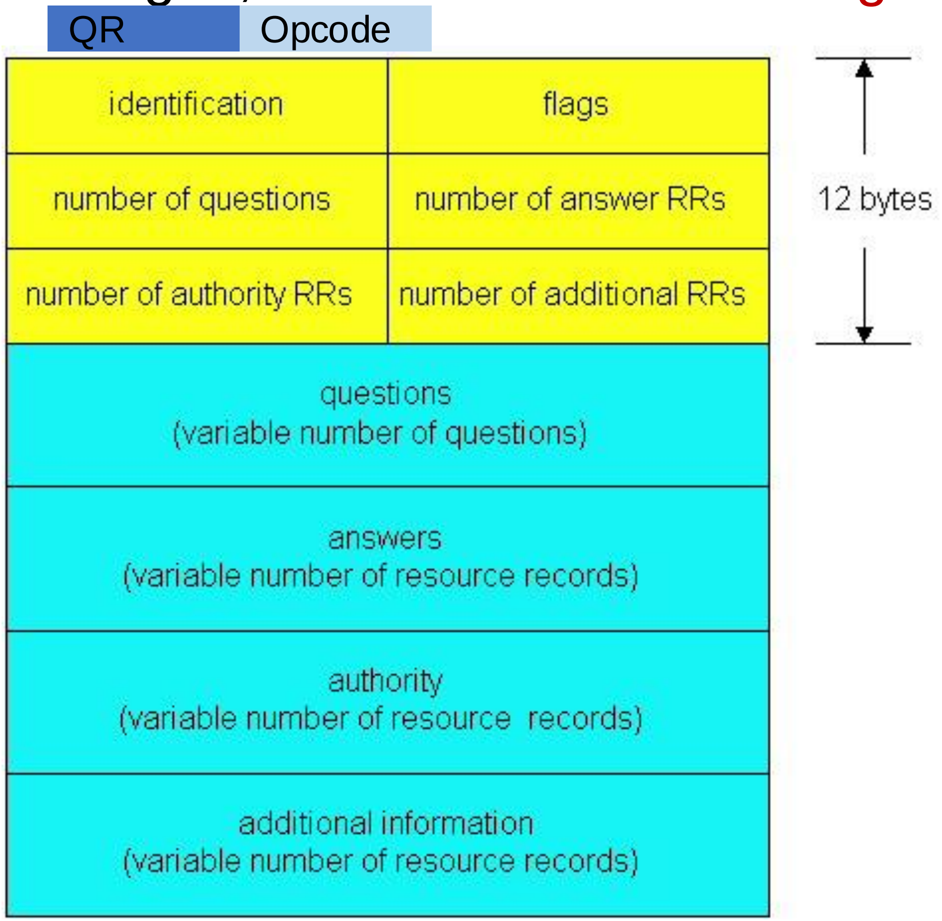
- Don’t just watch; try it!

DNS protocol: Message format

DNS protocol : query and reply messages, both with same message format

Message header

- QR = 0 for Query, 1 for response
- Opcode= 0 standard
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - Authoritative answer
  - recursion desired
  - recursion available
  - reply is authoritative



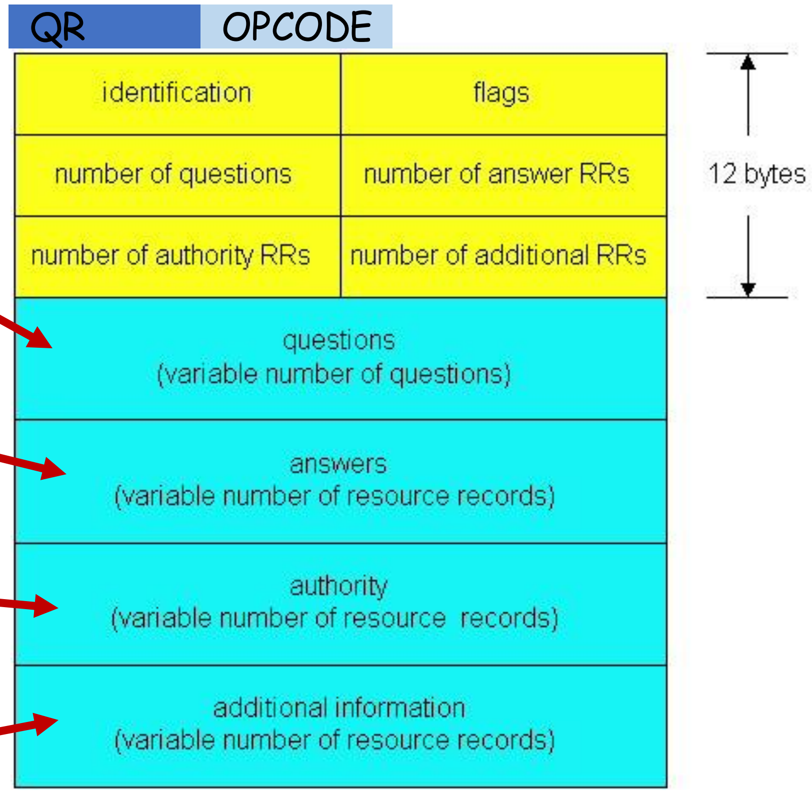
DNS protocol: Message format

Name, class, type fields
for a query

Resource records in
response to query

records for
authoritative servers

additional helpful
info that relates to the query
(but not the answer)



DNS Protocol: Actions

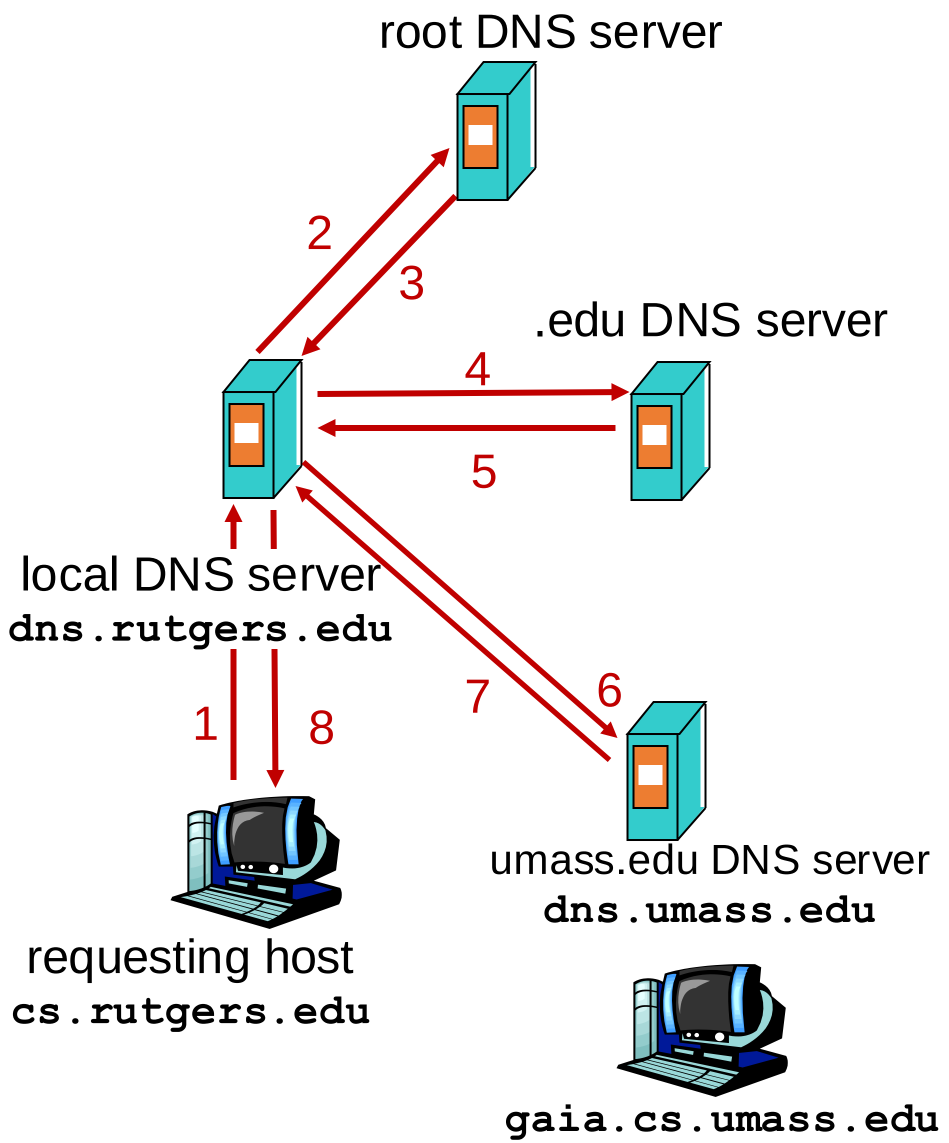
- When client wants to know an IP address for a host name

  - Client sends a DNS query to the “local” name server in its network
  - If name server contains the mapping, it returns the IP address to the client
  - Otherwise, the name server forwards the request to the root name server
  - The request works its way down the DNS hierarchy until it reaches a name server with a mapping for the requested name

Example

- Host at cs.rutgers.edu wants IP address for gaia.cs.umass.edu

- Local DNS server
- Root DNS server
- TLD DNS server
- Authoritative DNS server



Query type

- Iterative query

- Contacted server replies with name of server to contact

- “I don’t know this name, but ask this other server”

- Queries 2,4,6 are iterative from point of view of the local DNS server