



DNS and the Web

EE122 Fall 2012

Scott Shenker

<http://inst.eecs.berkeley.edu/~ee122/>

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson
and other colleagues at Princeton and UC Berkeley

Announcements

- Midterm on Thursday
 - Closed book
 - Crib sheet: 2-sided, 8pt font minimum.
- Midterm will **not** cover details from today's lecture
 - **You should know role basic concepts and roles**
 - o DNS resolves names to addresses
 - o Differences between names and addresses
 - o HTTP retrieves content, is app-layer protocol
 - o Content is named relative to hosts
 - o
- **Review on Tuesday**

Today: DNS and Web

- How many people already know how they work?
- Today is not about the details of these mechanisms, but about what roles they fulfill
 - Section will fill in the details....

DNS

Naming

- Internet has one global system of addressing: IP
 - By explicit design
- And one global system of naming: DNS
 - Almost by accident, naming was an afterthought
- At the time, only items worth naming were hosts
 - A mistake that causes many painful workarounds
- Everything is now named relative to a host
 - Content is most notable example (URL structure)

Logical Steps in Using Internet

- Person has name of entity she wants to access
 - Content, host, etc.
- Invokes an application to perform relevant task
 - Using that name (e.g., `www.cnn.com`)
- App invokes DNS to translate name to address
 - E.g. `157.166.255.18`
- App invokes transport protocol to contact host
 - Using address as destination

Addresses vs Names

- Scope of relevance:
 - App/user is primarily concerned with names
 - Network is primarily concerned with addresses
- Frequency:
 - Name → address lookup once (or get from cache)
 - Address → physical port lookup on each packet
- When moving a host to a different subnet:
 - The address changes
 - The name does not change
- When moving content to a differently named host
 - Name and address both change! (**should it?**)

Relationship Betw'n Names/Addresses

- Addresses can **change** underneath
 - Move www.cnn.com to 4.125.91.21
 - Humans/Apps should be unaffected
- Name could map to **multiple** IP addresses
 - www.cnn.com to multiple replicas of the Web site
 - Enables
 - o Load-balancing
 - o Reducing latency by picking nearby servers
- **Multiple names** for the same address
 - E.g., aliases like www.cnn.com and cnn.com
 - Mnemonic stable name, and dynamic canonical name
 - o Canonical name = actual name of host

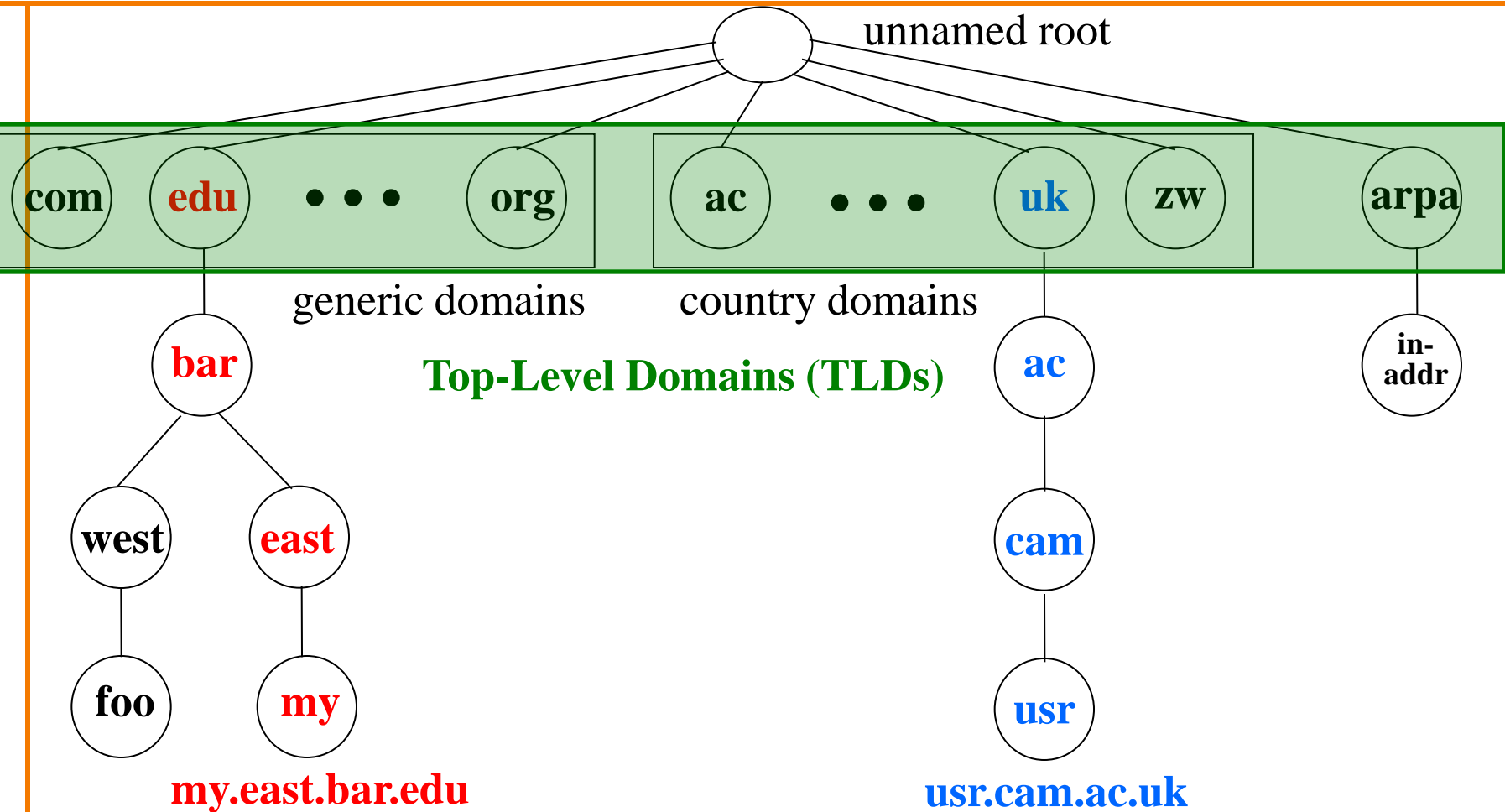
Mapping from Names to Addresses

- Originally: per-host file /etc/hosts
 - SRI (Menlo Park) kept master copy
 - Downloaded regularly
 - Flat namespace
- Single server not resilient, doesn't scale
 - Adopted a distributed hierarchical system
- Two intertwined hierarchies:
 - Infrastructure: hierarchy of DNS servers
 - Naming structure: www.cnn.com

Domain Name System (DNS)

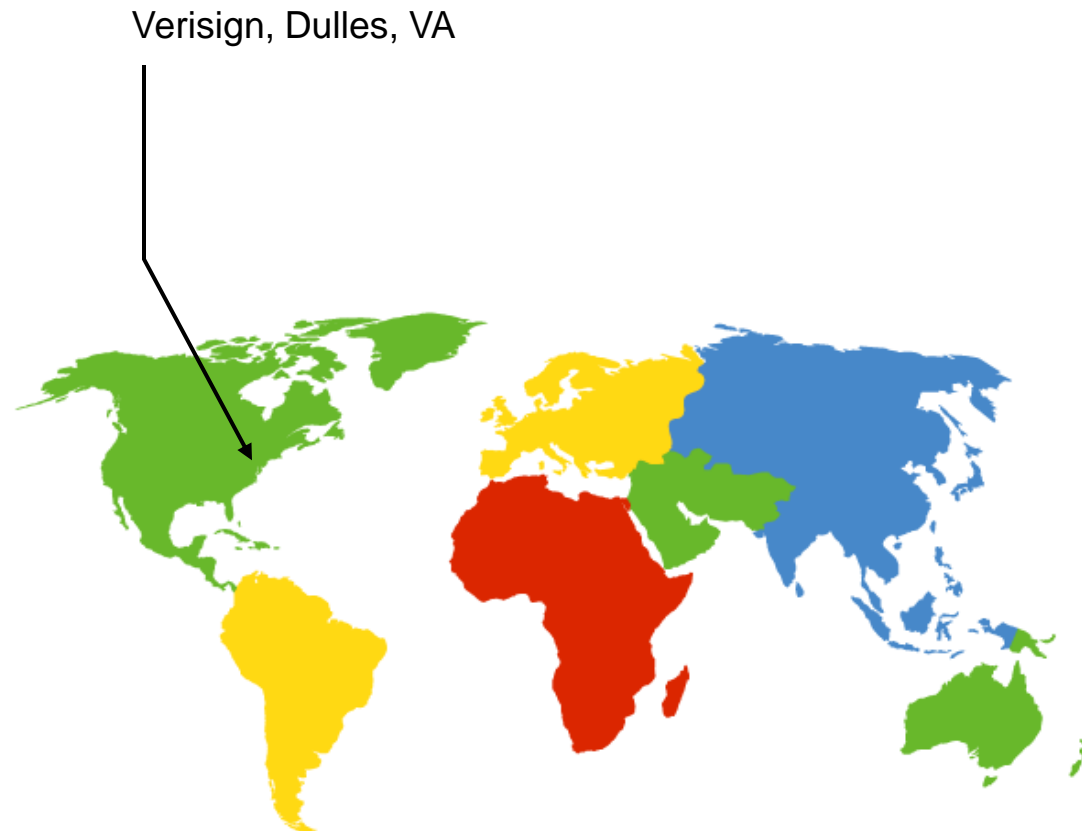
- Top of hierarchy: Root
 - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
 - .com, .edu, etc.
 - Managed professionally
- Bottom Level: Authoritative DNS servers
 - Actually do the mapping
 - Can be maintained locally or by a service provider

Distributed Hierarchical Database



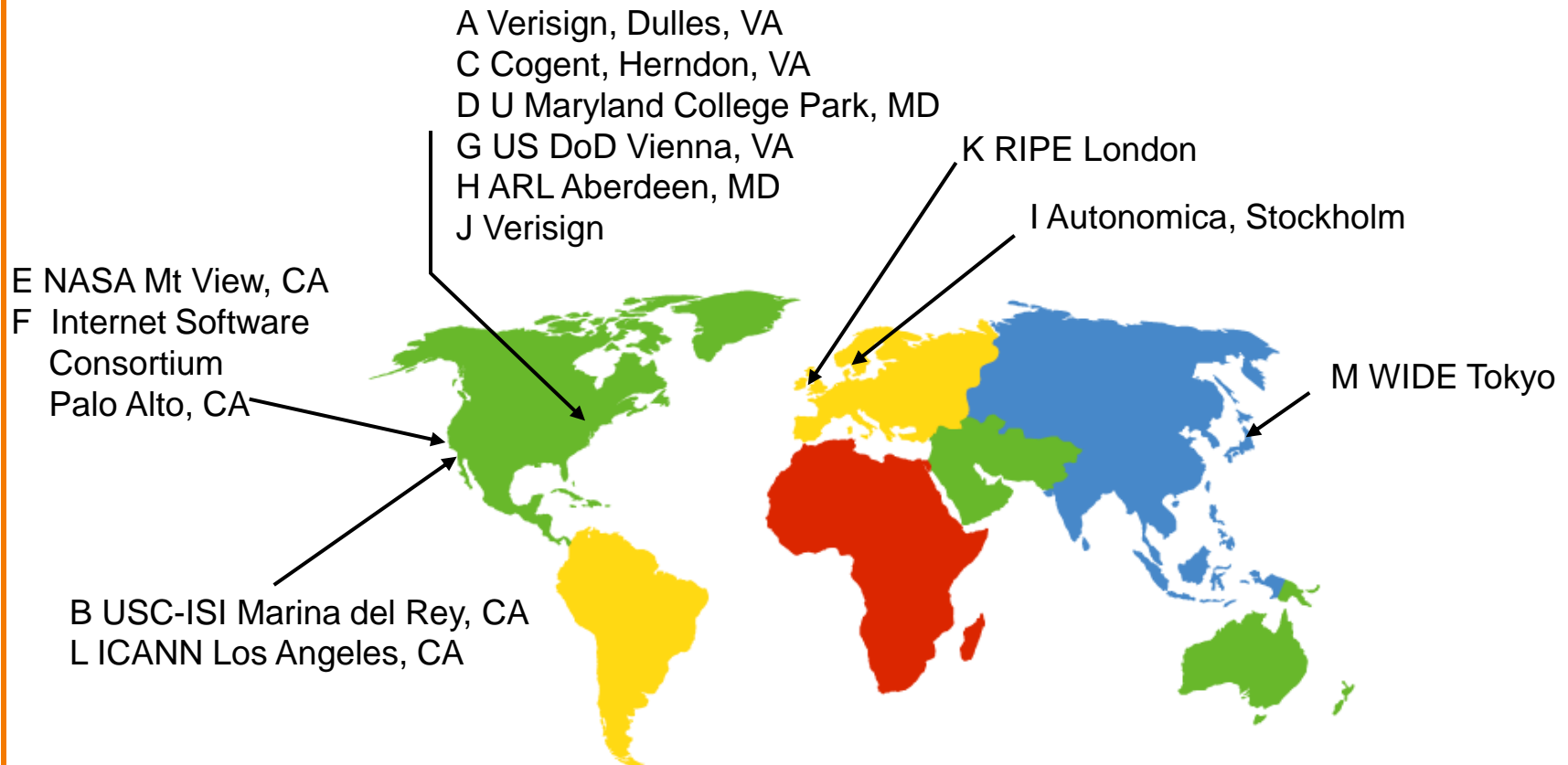
DNS Root

- Located in Virginia, USA
- How do we make the root scale?



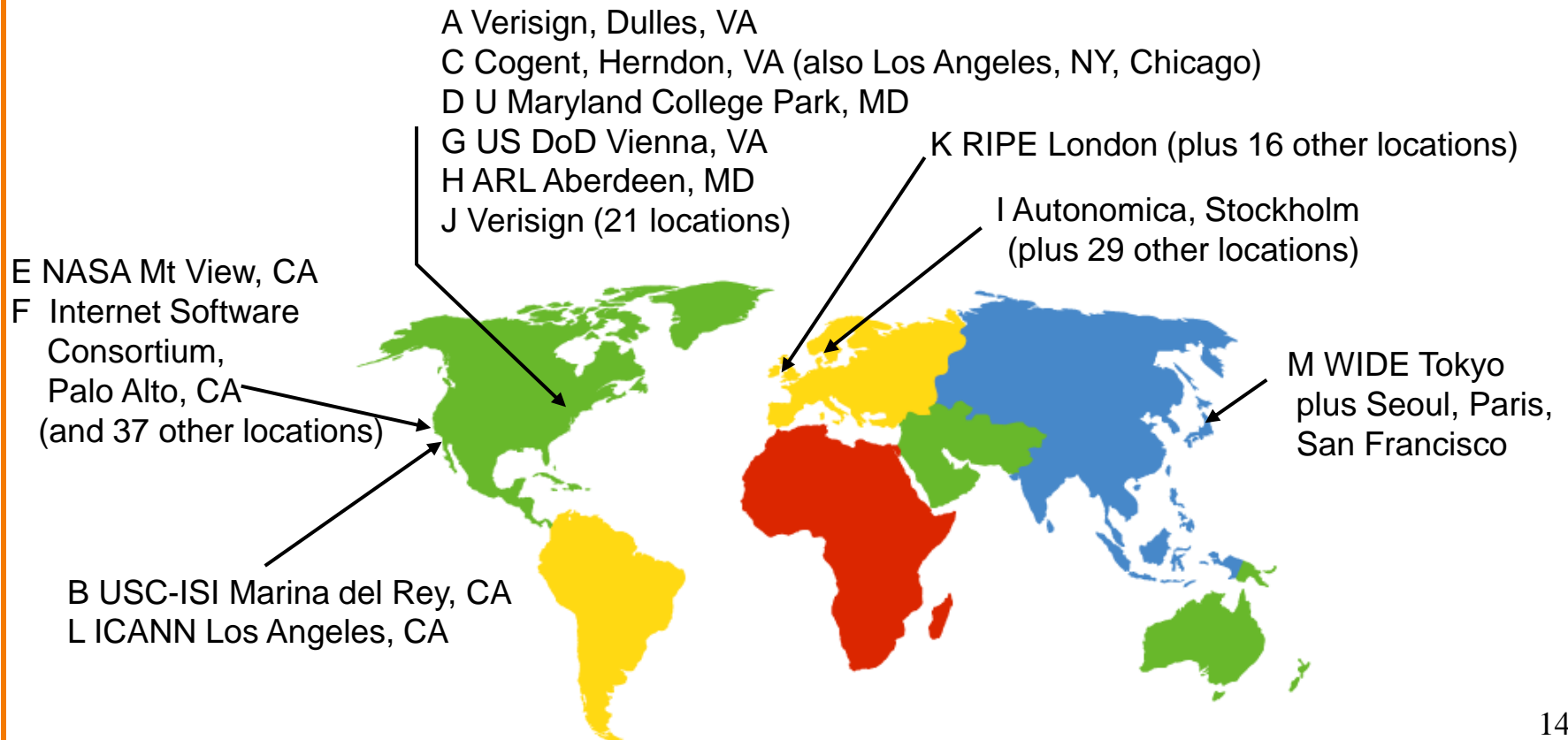
DNS Root Servers

- 13 root servers (see <http://www.root-servers.org/>)
 - Labeled A through M
- Does **this** scale?



DNS Root Servers

- 13 root servers (see <http://www.root-servers.org/>)
 - Labeled A through M
- Replication via **any-casting** (localized routing for addresses)



Refresher course on anycast

- Routing finds shortest paths to destination
- If several locations are given the same address, then the network will deliver the packet to the closest location with that address
- This is called “anycast”
 - But no modification of routing is needed for this....

Was Hierarchy Necessary?

- Two aspects of hierarchy:
 - Name resolution: walk up/down hierarchy
 - Name allocation: control over namespace partitioned
- How to handle both without hierarchy?
 - **Any ideas?**
- Resolution: Google
 - scalable key-value store
- Allocation:
 - Statistically unique names (random)

Using DNS

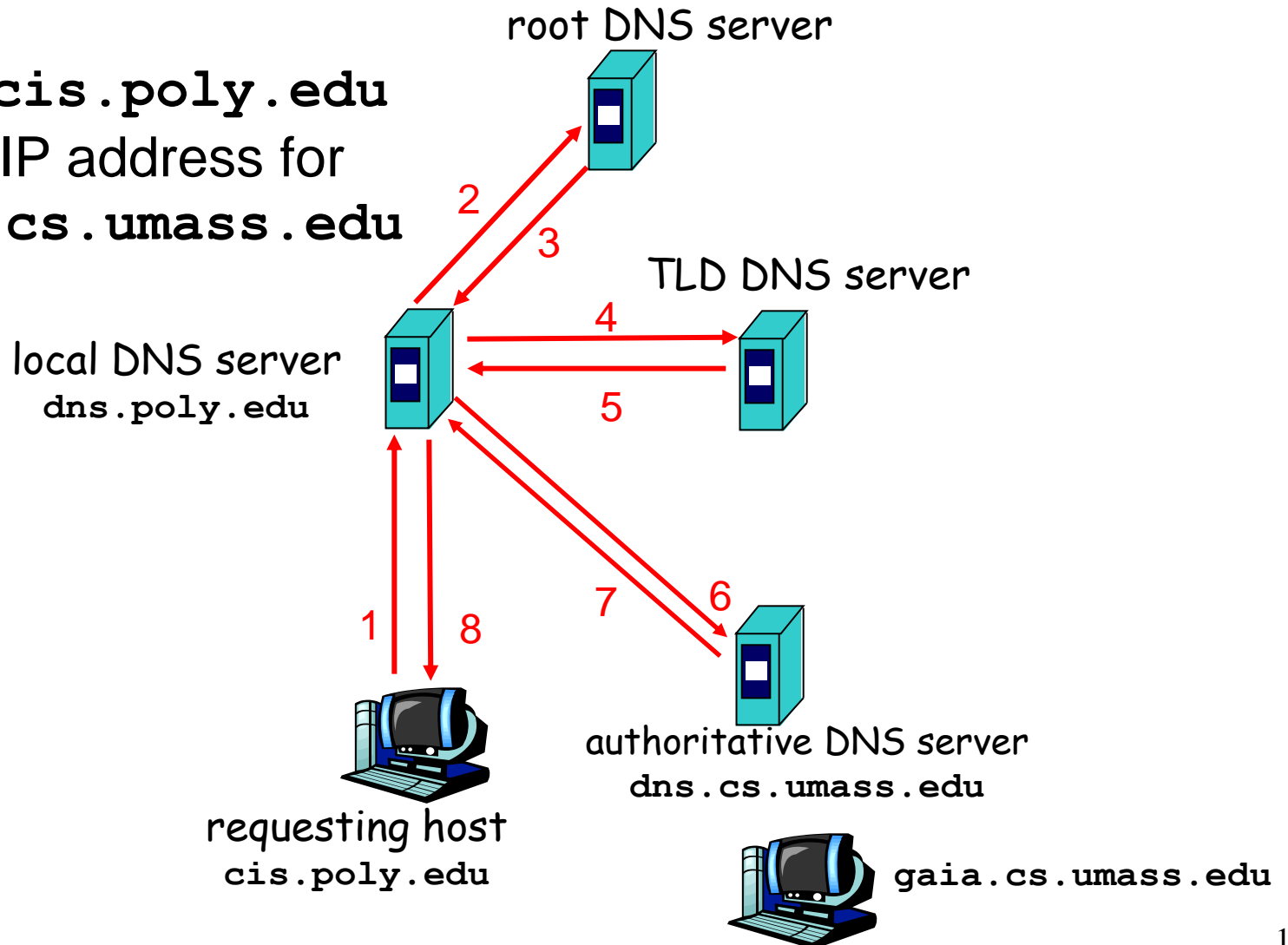
- Two components
 - Local DNS servers
 - Resolver software on hosts
- Local DNS server (“default name server”)
 - Usually near the endhosts that use it
 - Local hosts configured with local server (e.g., `/etc/resolv.conf`) or learn server via DHCP
- Client application
 - Extract server name (e.g., from the URL)
 - Do **`gethostbyname()`** to trigger resolver code

Many of you have complained....

- ...that applications knowing addresses is a violation of layering
- What do people think?
- My opinion:
 - Layers are highly modular abstractions
 - Implementations are not very modular
 - o Violate modularity in several places. This is one of them.
- *Applications handling addresses as bags of bits is ok, but “understanding addresses” is not*

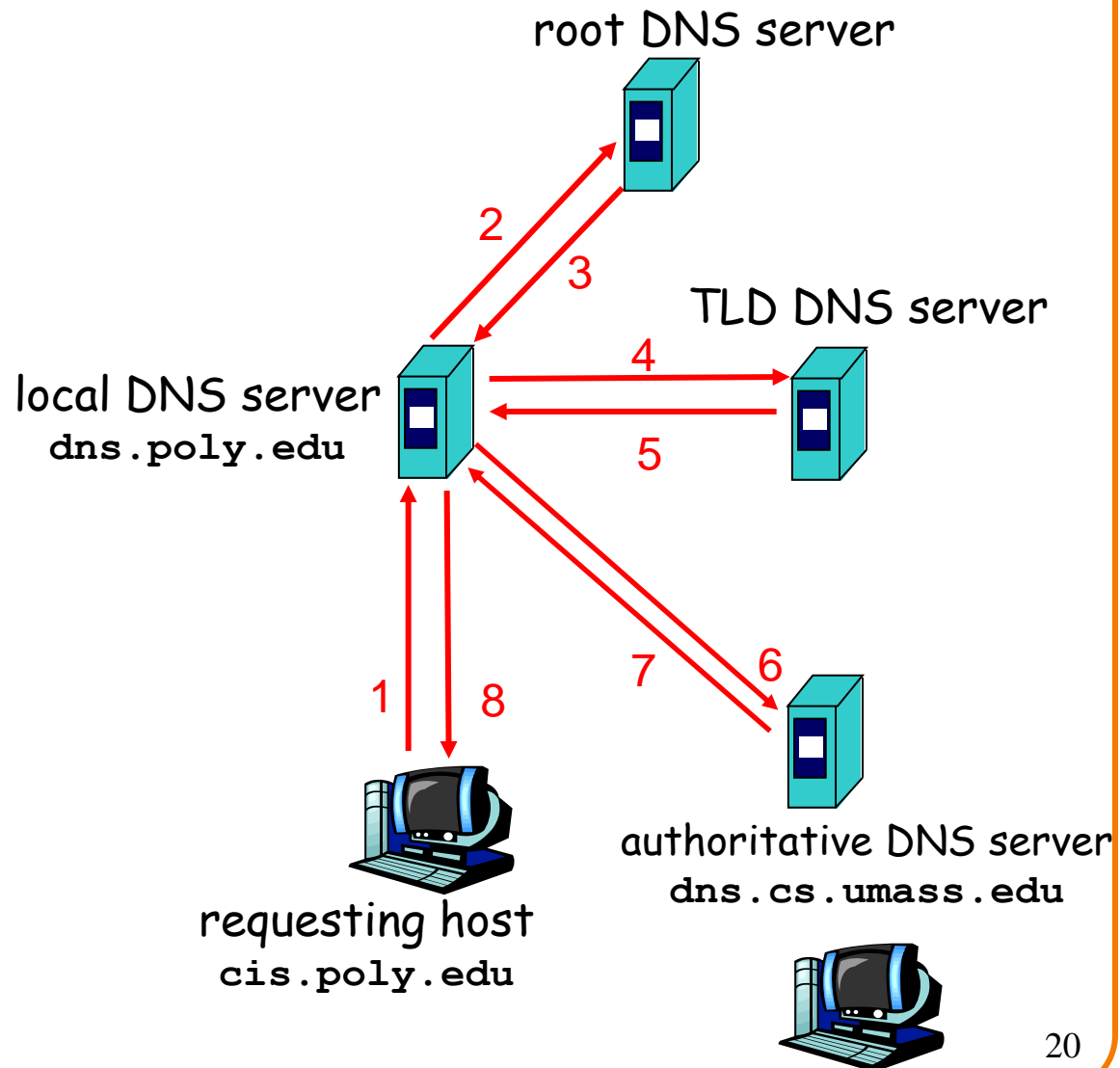
How Does Resolution Happen?

Host at `cis.poly.edu`
wants IP address for
`gaia.cs.umass.edu`



Recursive vs. Iterative Queries

- **Recursive** query
 - Ask server to get answer for you
 - E.g., request 1 and response 8
- **Iterative** query
 - Ask server who to ask next
 - E.g., all other request-response pairs



DNS Caching

- Performing all these queries takes time
 - And all this **before** actual communication takes place
 - E.g., 1-second latency before starting Web download
- **Caching** can greatly reduce overhead
 - The top-level servers very rarely change
 - Popular sites (e.g., www.cnn.com) visited often
 - Local DNS server often has the information cached
- How DNS caching works
 - DNS servers cache responses to queries
 - Responses include a “**time to live**” (TTL) field
 - Server deletes cached entry after TTL expires

Negative Caching

- Remember things that don't work
 - Misspellings like *www.cnn.comm* and *www.cnnn.com*
 - These can take a long time to fail the first time
 - Good to remember that they don't work
 - ... so the failure takes less time the next time around
- But: negative caching is **optional**
 - And not widely implemented

DNS Resource Records

DNS: distributed DB storing resource records (RR)

RR format: (name, value, type, ttl)

- Type=A
 - **name** is hostname
 - **value** is IP address
- Type=NS
 - **name** is domain (e.g. foo.com)
 - **value** is hostname of authoritative name server for this domain
- Type=PTR
 - **name** is reversed IP quads
 - o E.g. 78.56.34.12.in-addr.arpa
 - **value** is corresponding hostname
- Type=CNAME
 - **name** is alias name for some “**canonical**” name
 - E.g., `www.cs.mit.edu` is really `eeecsweb.mit.edu`
 - **value** is canonical name
- Type=MX
 - **value** is name of mailserver associated with **name**
 - Also includes a weight/preference

DNS Protocol

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

Message header:

- **Identification:** 16 bit # for query, reply to query uses same #
- **Flags:**
 - Query or reply
 - Recursion desired
 - Recursion available
 - Reply is authoritative
- Plus fields indicating **size** (0 or more) of optional header elements

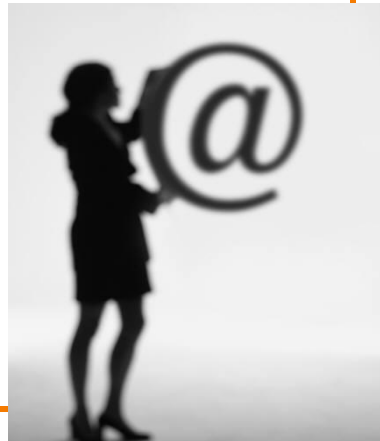
<i>16 bits</i>	<i>16 bits</i>
Identification	Flags
# Questions	# Answer RRs
# Authority RRs	# Additional RRs
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Reliability

- DNS servers are **replicated** (primary/secondary)
 - Name service available if at least one replica is up
 - Queries can be load-balanced between replicas
- Usually, UDP used for queries (**why???**)
 - Need reliability: must implement this on top of UDP
 - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
 - **Exponential backoff** when retrying same server
- Same identifier for all queries
 - Don't care which server responds

Inserting Resource Records into DNS

- Example: just created startup “FooBar”
- Get a block of address space from ISP
 - Say 212.44.9.128/25
- Register **foobar.com** at Network Solutions (say)
 - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
 - Registrar inserts RR pairs into the **com** TLD server:
 - o (**foobar.com**, **dns1.foobar.com**, NS)
 - o (**dns1.foobar.com**, **212.44.9.129**, A)
- Put in your (authoritative) server **dns1.foobar.com**:
 - Type A record for **www.foobar.com**
 - Type MX record for **foobar.com**



DNS Measurements (MIT data from 2000)

- What is being looked up?
 - ~60% requests for A records
 - ~25% for PTR records
 - ~5% for MX records
 - ~6% for ANY records
- How long does it take?
 - Median ~100msec (but 90th percentile ~500msec)
 - 80% have no referrals; 99.9% have fewer than four
- Query packets per lookup: ~2.4
 - But this is misleading....

DNS Measurements (MIT data from 2000)

- Does DNS give answers?
 - ~23% of lookups fail to elicit an answer!
 - ~13% of lookups result in NXDOMAIN (or similar)
 - o Mostly reverse lookups
 - Only ~64% of queries are successful!
 - o *How come the web seems to work so well?*
- ~ 63% of DNS packets in unanswered queries!
 - Failing queries are frequently retransmitted
 - 99.9% successful queries have ≤ 2 retransmissions

Moral of the Story

- If you design a highly resilient system, many things can be going wrong without you noticing it!

DNS Measurements (MIT data from 2000)

- Top 10% of names accounted for ~70% of lookups
 - Caching should really help!
- 9% of lookups are unique
 - Cache hit rate can never exceed 91%
- Cache hit rates ~ 75%
 - But caching for more than 10 hosts doesn't add much

A Common Pattern.....

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
 - Large fraction of total metric in the top 10%
 - Sizable fraction (~10%) of total fraction in low values
- In an exponential distribution
 - Large fraction is in top 10%
 - But low values have very little of overall total
- Lesson: have to pay attention to both ends of dist.
- Here: caching helps, but not a panacea

DNS and Security

- No way to verify answers
 - Opens up DNS to many potential attacks
 - DNSSEC fixes this
- Most obvious vulnerability: recursive resolution
 - Using recursive resolution, host must trust DNS server
 - When at Starbucks, server is under their control
 - And can return whatever values it wants
- More subtle attack: Cache poisoning
 - Those “additional” records can be anything!

Cache Poisoning

- Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and reply:

```
;; QUESTION SECTION:
;www.foobar.com.          IN      A

;; ANSWER SECTION:
www.foobar.com.          300     IN      A      212.44.9.144

;; AUTHORITY SECTION:
foobar.com.              600     IN      NS      dns1.foobar.com.
foobar.com.              600     IN      NS      google.com.

;; ADDITIONAL SECTION:
google.com.              5       IN      A      212.44.9.155
```

Evidence of the attack disappears 5 seconds later!

A foobar.com machine, *not* google.com

The Web

The Web – Precursor



Ted Nelson

- **1967**, Ted Nelson, Xanadu:
 - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
 - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term “Hypertext”
 - Influenced research community
 - o Who then missed the web.....

The Web – History



Tim Berners-Lee

- Physicist trying to solve real problem
 - Distributed access to data
- World Wide Web (WWW): a distributed database of “pages” linked through **Hypertext Transport Protocol (HTTP)**
 - First HTTP implementation - 1990
 - o Tim Berners-Lee at CERN
 - HTTP/0.9 – 1991
 - o Simple GET command for the Web
 - HTTP/1.0 – 1992
 - o Client/Server information, simple caching
 - HTTP/1.1 - 1996

Why Didn't CS Research Invent Web?

HTML is precisely what we were trying to PREVENT— ever-breaking links, links going outward only, quotes you can't follow to their origins, no version management, no rights management.

– Ted Nelson

**Academics get paid for being clever,
not for being right.**

–Don Norman

Why So Successful?

- What do the web, youtube, fb have in common?
 - The ability to self-publish
- Self-publishing that is easy, independent, free
- No interest in collaborative and idealistic endeavor
 - People aren't looking for Nirvana (or even Xanadu)
 - People also aren't looking for technical perfection
- Want to make their mark, and find something neat
 - Two sides of the same coin, creates synergy
 - “Performance” more important than dialogue....

Web Components

- Infrastructure:
 - Clients
 - Servers
 - Proxies
- Content:
 - Individual objects (files, etc.)
 - Web sites (coherent collection of objects)
- Implementation
 - HTML: formatting content
 - URL: naming content
 - HTTP: protocol for exchanging content

HTML: HyperText Markup Language

- A *Web page* has:
 - Base HTML file
 - Referenced objects (*e.g.*, images)
- HTML has several functions:
 - Format text
 - Reference images
 - Embed *hyperlinks* (HREF)

URL Syntax

protocol : // *hostname* [: *port*] / *directory path* / *resource*

protocol http, ftp, https, smtp, rtsp, etc.

hostname DNS name, IP address

port Defaults to protocol's standard port
e.g. http: 80 https: 443

directory path Hierarchical, reflecting file system

resource Identifies the desired resource

Can also extend to program executions:

http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format

```
% telnet www.icir.org 80
GET /jdoe/ HTTP/1.0
<blank line, i.e., CRLF>
```

Steps in HTTP Request

- HTTP Client initiates TCP connection to server
 - SYN
 - SYNACK
 - ACK
- Client sends HTTP request to server
 - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

How many RTTs for a single request?

Round trips for an exchange

- TCP SYN →
- ← TCP SYN-ACK

First RTT (To Get TCP Started)

- TCP ACK →
- HTTP REQUEST →
- ← HTTP RESPONSE

Second RTT (To Get HTTP Response)

- TCP FIN →
- ← TCP FIN-ACK
- TCP ACK →

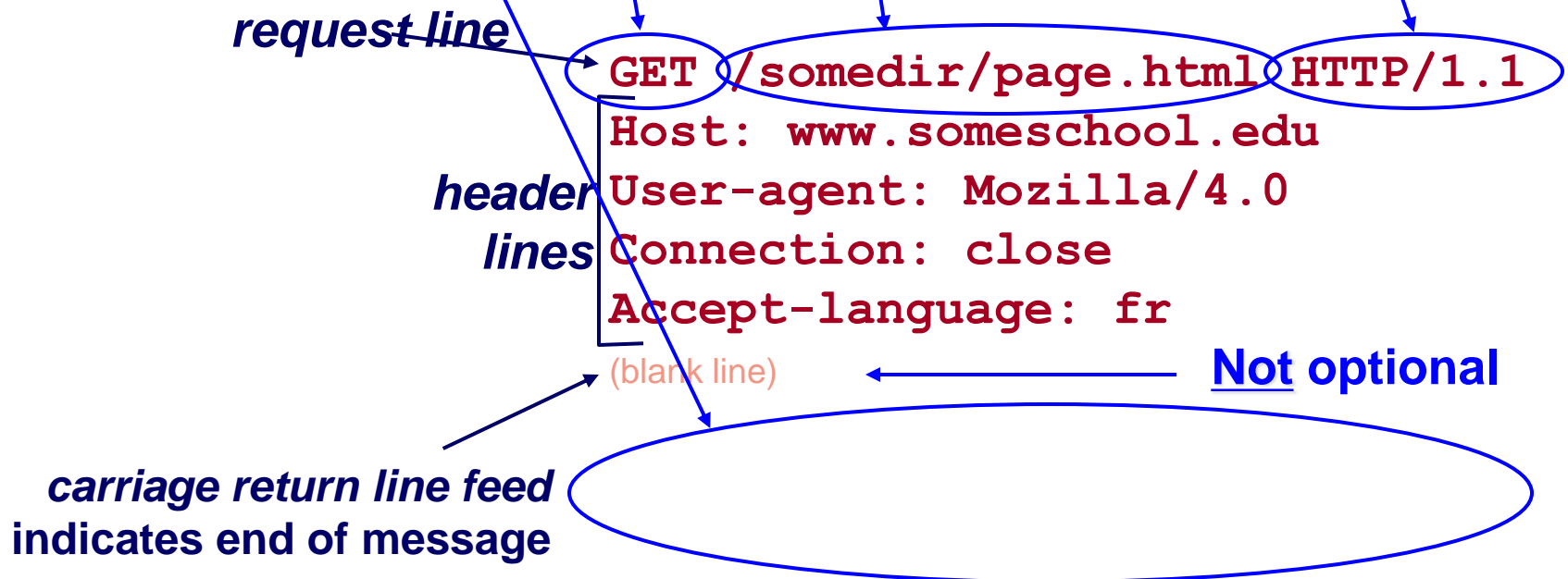
Third (and a half) RTT (To Close Down TCP Connection)

*Typically this third RTT doesn't matter (because data is delivered)*⁴⁴

Client-to-Server Communication

- HTTP Request Message

- Request line: **method, resource, and protocol version**
- Request headers: provide information or modify request
- Body: **optional data** (e.g., to “POST” data to the server)



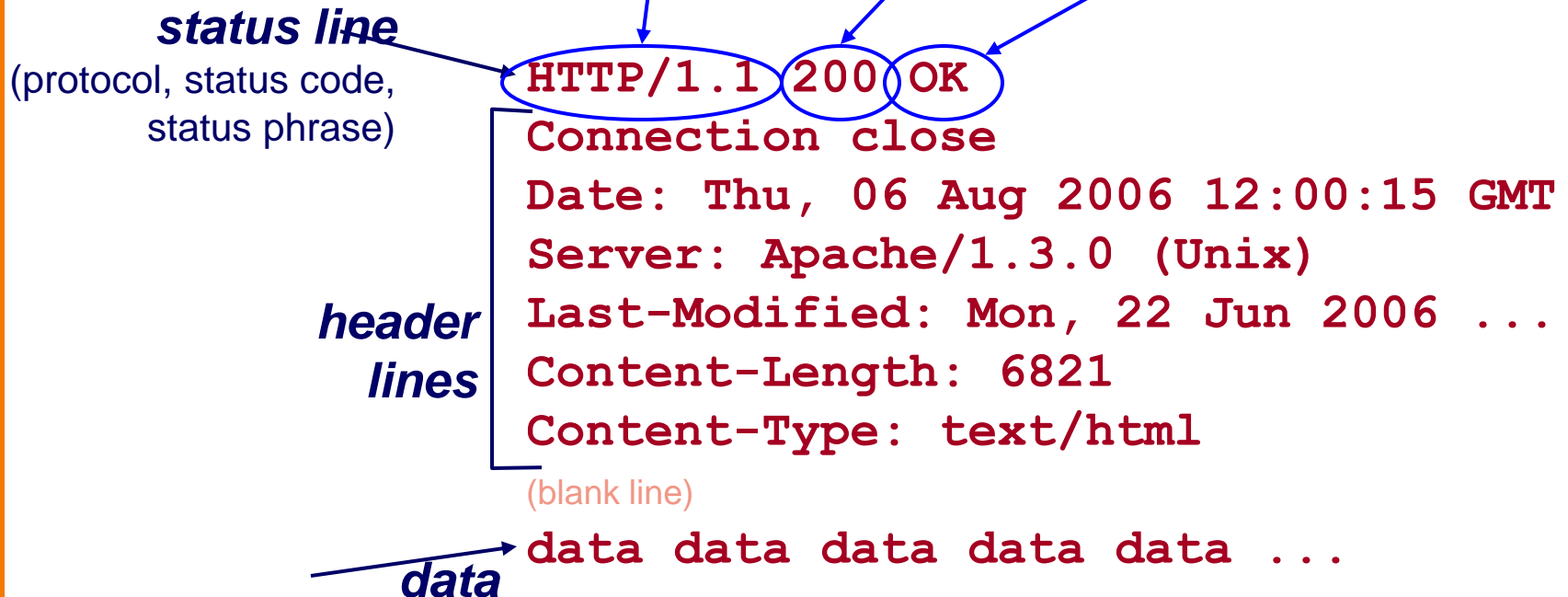
Client-to-Server Communication

- Request *methods* include:
 - **GET**: Return current value of resource, run program, ...
 - **HEAD**: Return the meta-data associated with a resource
 - **POST**: Update resource, provide input to a program, ...
- *Headers* include:
 - Useful info for the server
 - o e.g. desired language

Server-to-Client Communication

- HTTP Response Message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional data



e.g., requested HTML file

Server-to-Client Communication

- Response code classes
 - Similar to other ASCII app. protocols like SMTP

<u>Code</u>	<u>Class</u>	<u>Example</u>
1xx	Informational	100 Continue
2xx	Success	200 OK
3xx	Redirection	304 Not Modified
4xx	Client error	404 Not Found
5xx	Server error	503 Service Unavailable

Different Forms of Server Response

- Return a file
 - URL matches a file (*e.g.*, /www/index.html)
 - Server returns file as the response
 - Server generates appropriate response header
- Generate response dynamically
 - URL triggers a program on the server
 - Server runs program and sends output to client
- Return meta-data with no body

HTTP Resource Meta-Data

- Meta-data
 - Info *about* a resource, stored as a separate entity
- Examples:
 - Size of resource, last modification time, type of content
- Usage example: Conditional GET Request
 - Client requests object “**If-modified-since**”
 - If unchanged, “**HTTP/1.1 304 Not Modified**”
 - No body in the server’s response, only a header

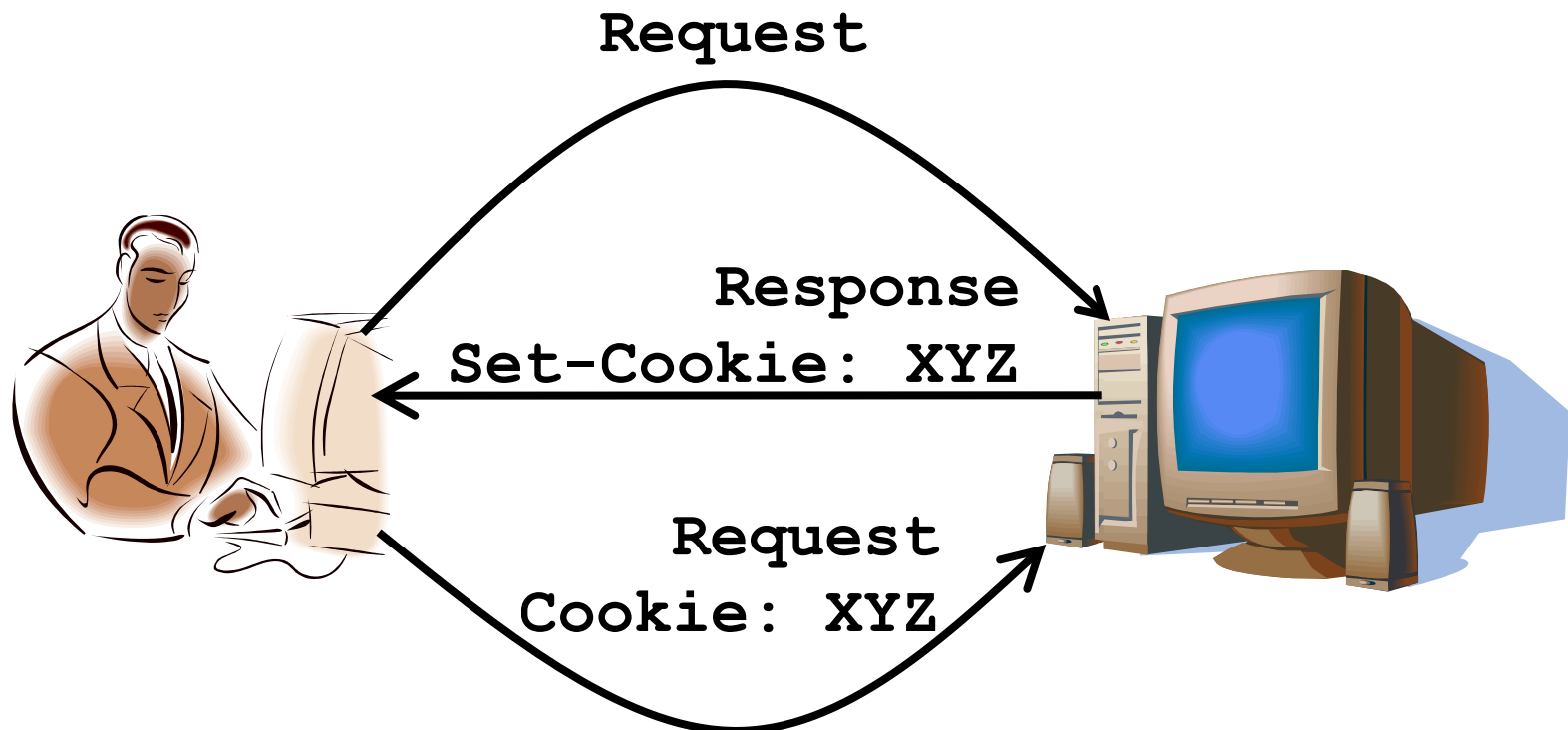
HTTP is *Stateless*

- Each request-response treated independently
 - Servers *not* required to retain state
- **Good:** Improves scalability on the server-side
 - Failure handling is easier
 - Can handle higher rate of requests
 - Order of requests doesn't matter
- **Bad:** Some applications **need** persistent state
 - Need to uniquely identify user or store temporary info
 - *e.g.*, Shopping cart, user profiles, usage tracking, ...

State in a Stateless Protocol:

Cookies

- *Client-side* state maintenance
 - Client stores small(?) state on behalf of server
 - Client sends state in future requests to the server
- Can provide authentication

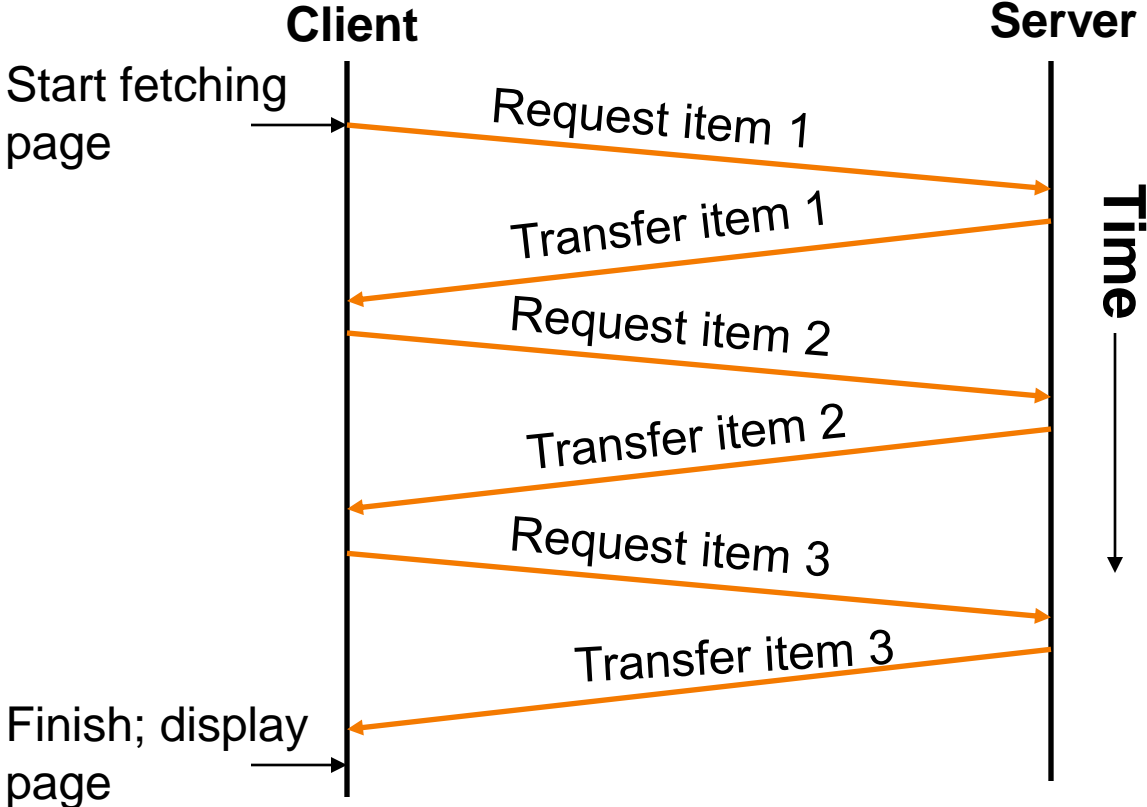


Performance Issues

HTTP Performance

- Most Web pages have multiple objects
 - e.g., HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
 - *One item at a time*

Fetch HTTP Items: Stop & Wait

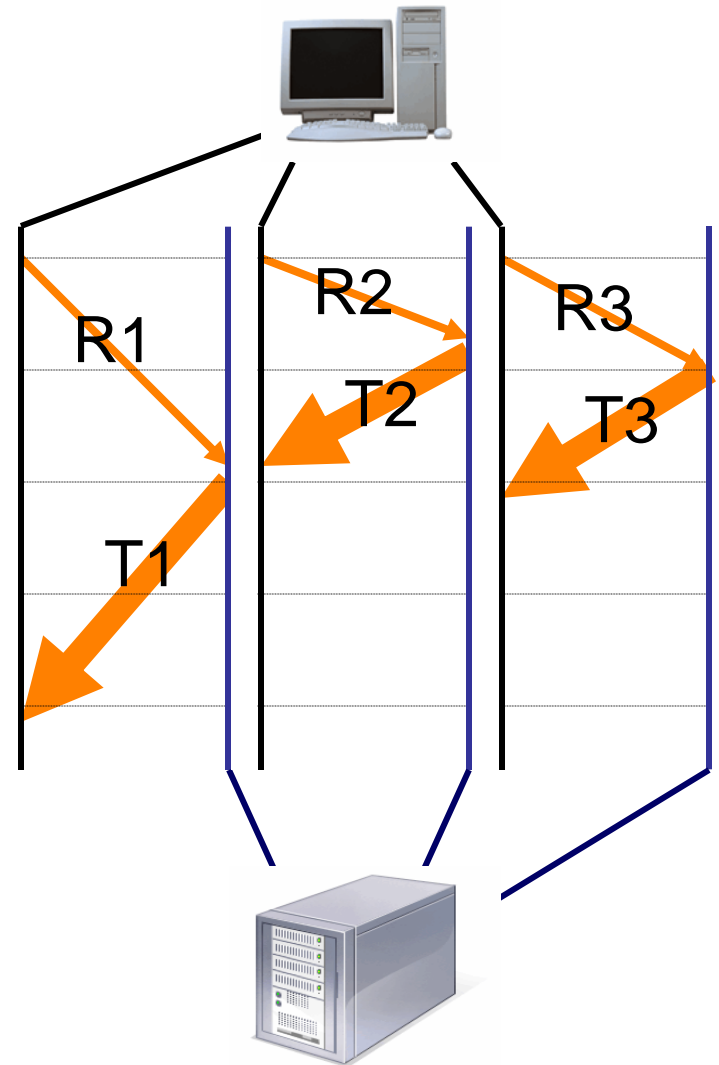


**≥2 RTTs
per
object**

Improving HTTP Performance:

Concurrent Requests & Responses

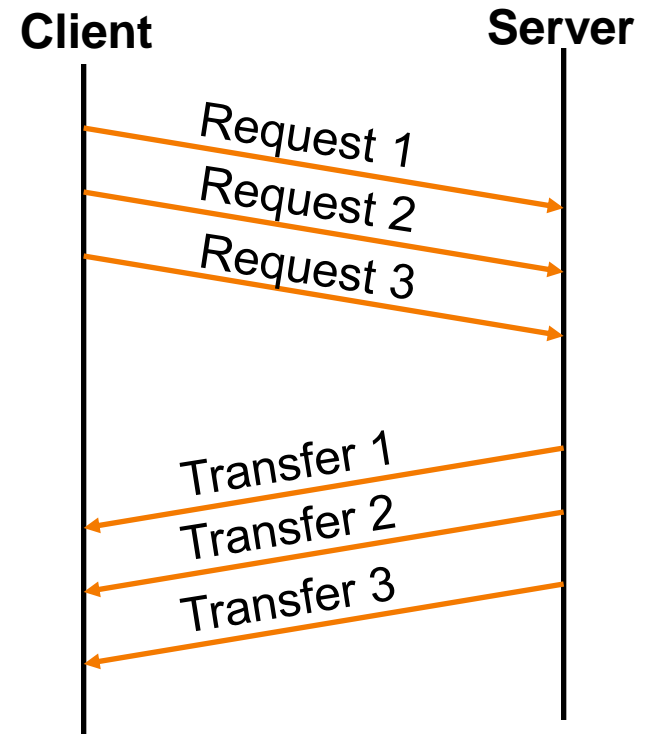
- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
- Client = 😊
- Server = 😊
- Network = 😞 Why?



Improving HTTP Performance:

Pipelined Requests & Responses

- *Batch* requests and responses
 - Reduce connection overhead
 - Multiple requests sent in a single batch
 - Maintains order of responses
 - Item 1 always arrives before item 2
- How is this different from concurrent requests/responses?
 - Single TCP connection



Improving HTTP Performance:

Persistent Connections

- Enables multiple transfers per connection
 - Maintain TCP connection across multiple requests
 - Including transfers subsequent to current page
 - Client or server can tear down connection
- Performance advantages:
 - Avoid overhead of connection set-up and tear-down
 - Allow TCP to learn more accurate RTT estimate
 - Allow TCP congestion window to increase
 - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time: $\sim 2n$ RTT
- Persistent: $\sim (n+1)$ RTT
- M concurrent: $\sim 2\lceil n/m \rceil$ RTT
- Pipelined: ~ 2 RTT
- Pipelined/Persistent: ~ 2 RTT first time, RTT later

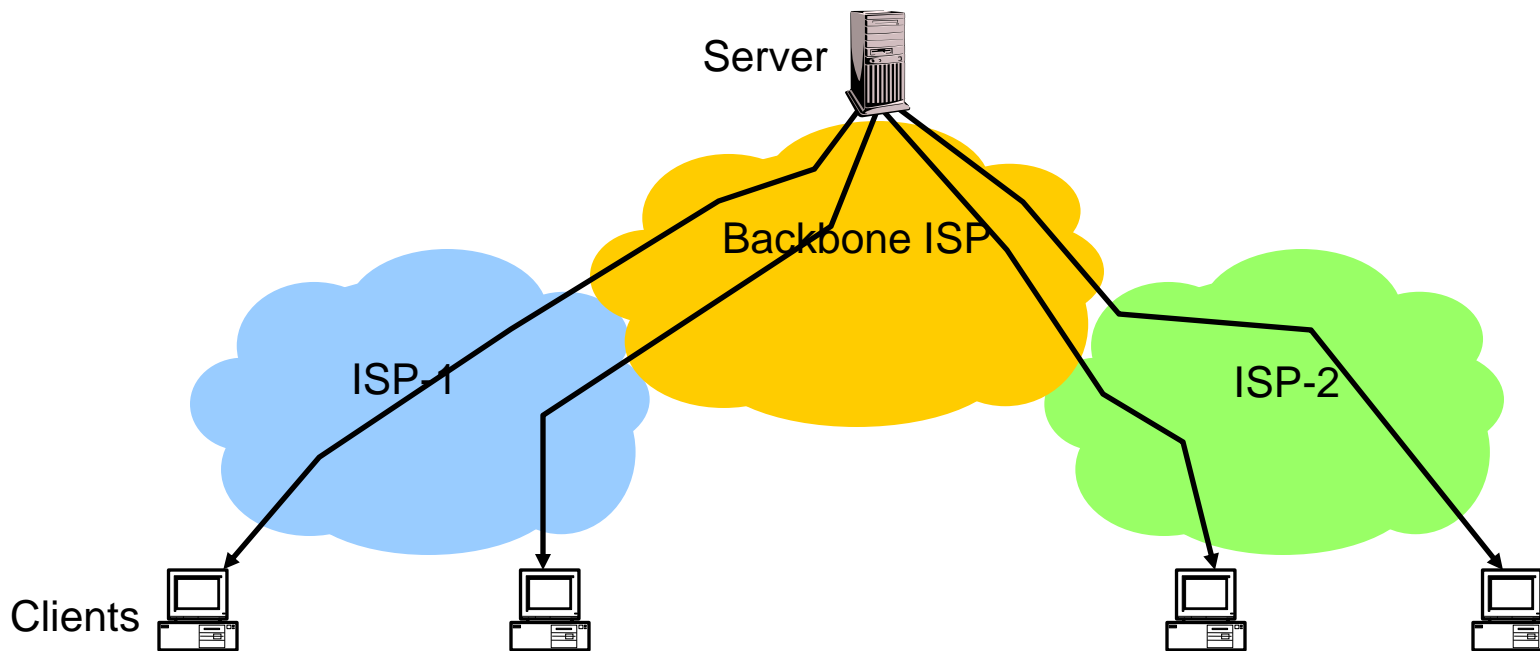
Scorecard: Getting n Large Objects

Time dominated by bandwidth

- One-at-a-time: $\sim nF/B$
- M concurrent: $\sim [n/m] F/B$
 - assuming shared with large population of users
 - and each TCP connection gets the same bandwidth
- Pipelined and/or persistent: $\sim nF/B$
 - The only thing that helps is getting more bandwidth..

Improving HTTP Performance: Caching

- Many clients transfer **same information**
 - Generates **redundant** server and network load
 - Clients experience **unnecessary** latency



Caching: How

- Modifier to GET requests:
 - **If-modified-since** – returns “not modified” if resource not modified since specified time
- Response header:
 - **Expires** – how long it’s safe to cache the resource
 - **No-cache** – ignore all caches; always get resource directly from server

Caching: Why

- Motive for placing content closer to client:
 - User gets better response time
 - Content providers get happier users
 - o Time is money, really!
 - Network gets reduced load
- Why does caching work?
 - Exploits *locality of reference*
- How well does caching work?
 - Very well, up to a limit
 - Large overlap in content
 - But many unique requests

Improving HTTP Performance:

Caching on the Client

Example: Conditional GET Request

- Return resource only if it has changed at the server
 - Save server resources!

Request from client to server:

```
GET /~ee122/fa07/ HTTP/1.1
Host: inst.eecs.berkeley.edu
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
<CRLF>
```

- How?
 - Client specifies “if-modified-since” time in request
 - Server compares this against “last modified” time of desired resource
 - Server returns “304 Not Modified” if resource has not changed
 - or a “200 OK” with the latest version otherwise

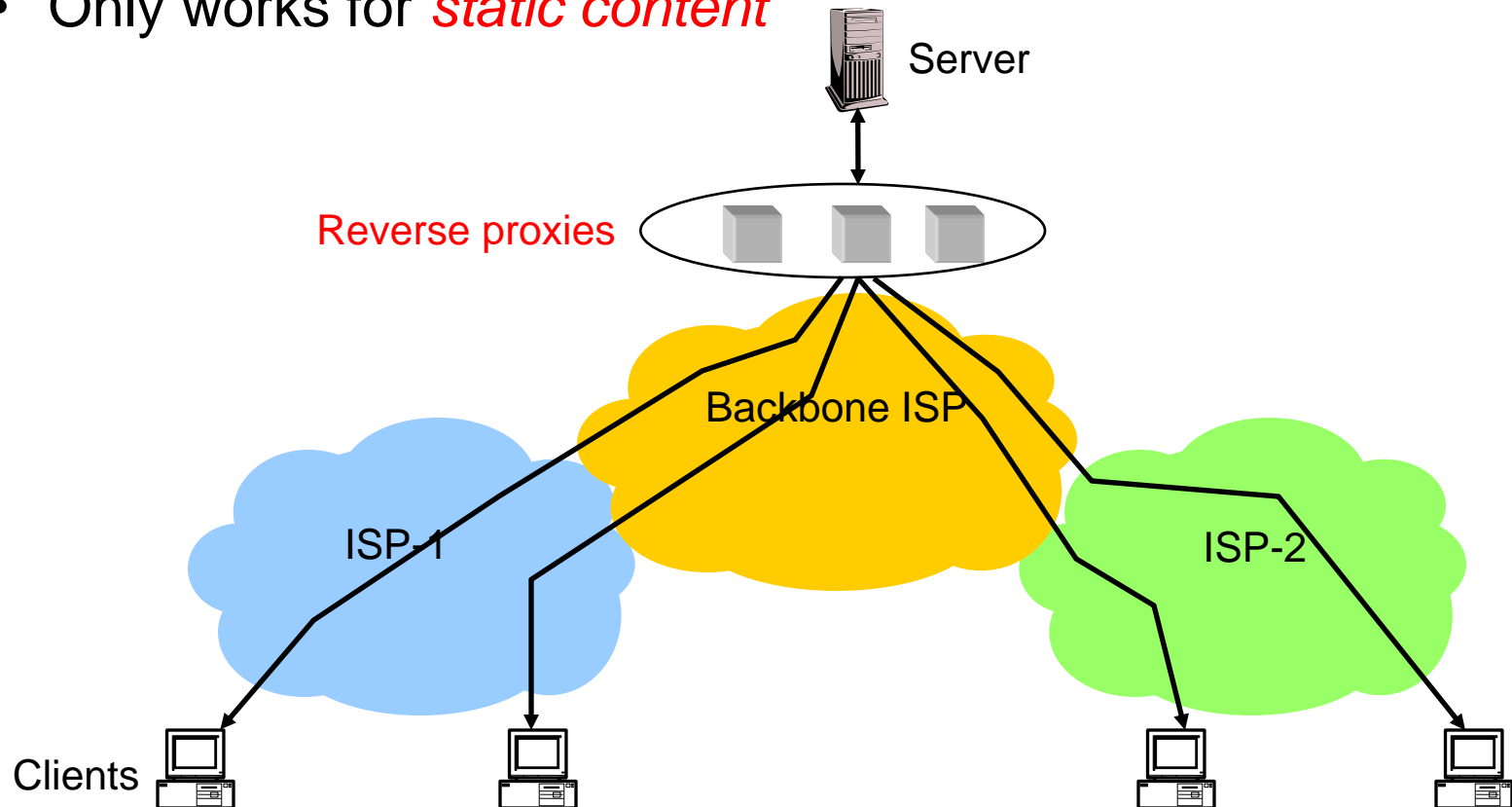
Improving HTTP Performance:

Caching with Reverse Proxies

Cache documents close to **server**

→ decrease server load

- Typically done by content providers
- Only works for *static content*



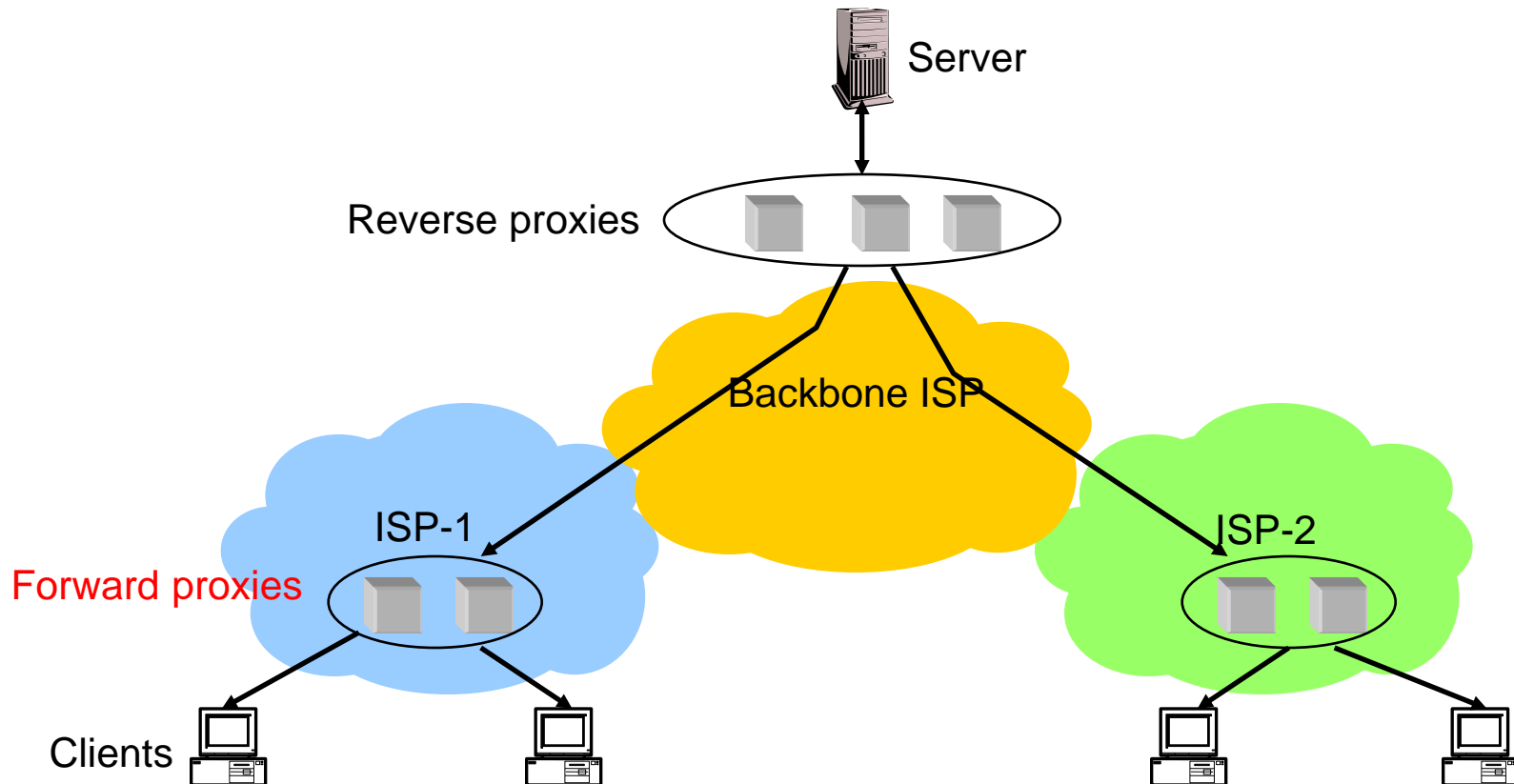
Improving HTTP Performance:

Caching with Forward Proxies

Cache documents close to **clients**

→ reduce network traffic and decrease latency

- Typically done by ISPs or corporate LANs

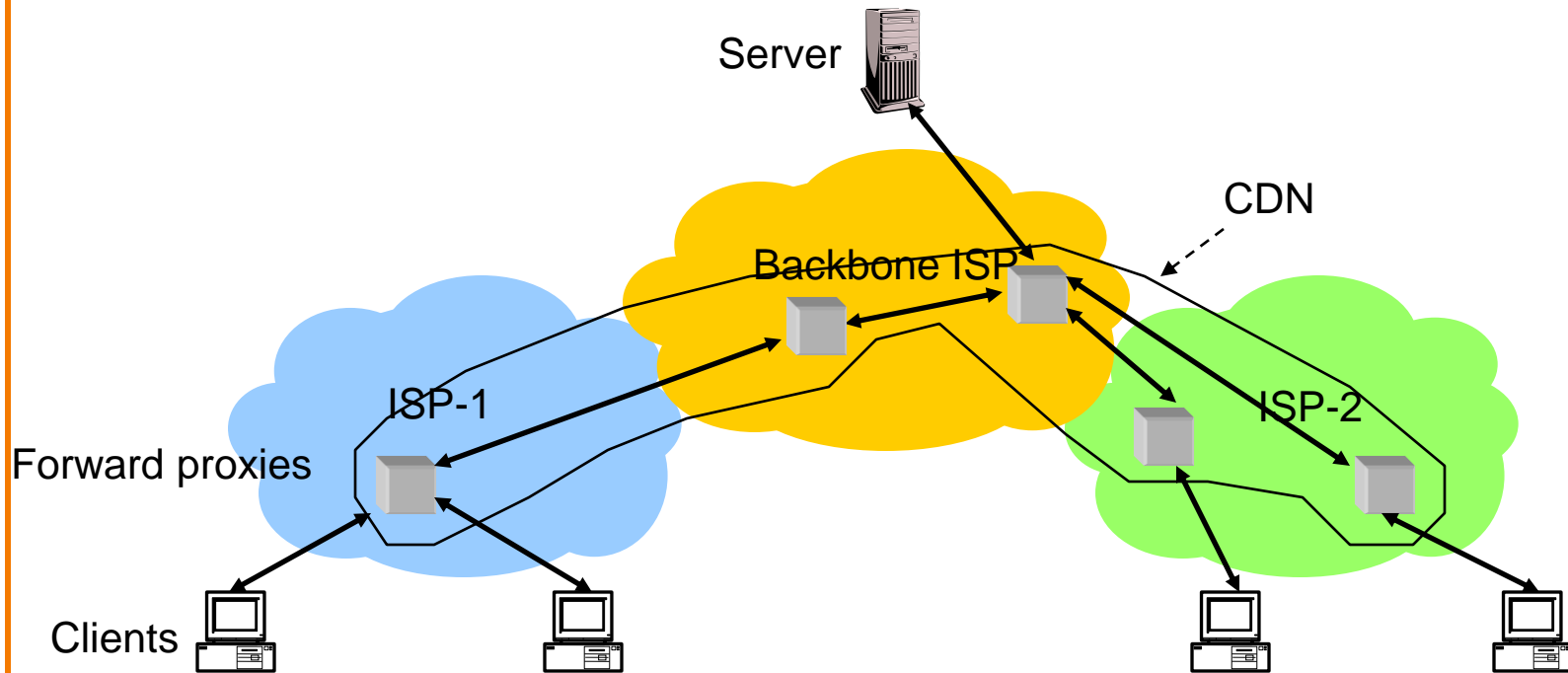


Improving HTTP Performance:

Caching w/ Content Distribution Networks

- Integrate forward and reverse caching functionality
 - One overlay network (usually) administered by one entity
 - *e.g.*, Akamai
- Provide document caching
 - **Pull**: Direct result of clients' requests
 - **Push**: Expectation of high access rate
- Also do some processing
 - Handle *dynamic* web pages
 - *Transcoding*

Improving HTTP Performance: Caching with CDNs (cont.)



CDN Example – Akamai

- Akamai creates new domain names for each client content provider.
 - e.g., a128.g.akamai.net
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
 - “Akamaize” content
 - e.g.: <http://www.cnn.com/image-of-the-day.gif> becomes <http://a128.g.akamai.net/image-of-the-day.gif>
- *Requests now sent to CDN's infrastructure...*

Hosting: Multiple Sites Per Machine

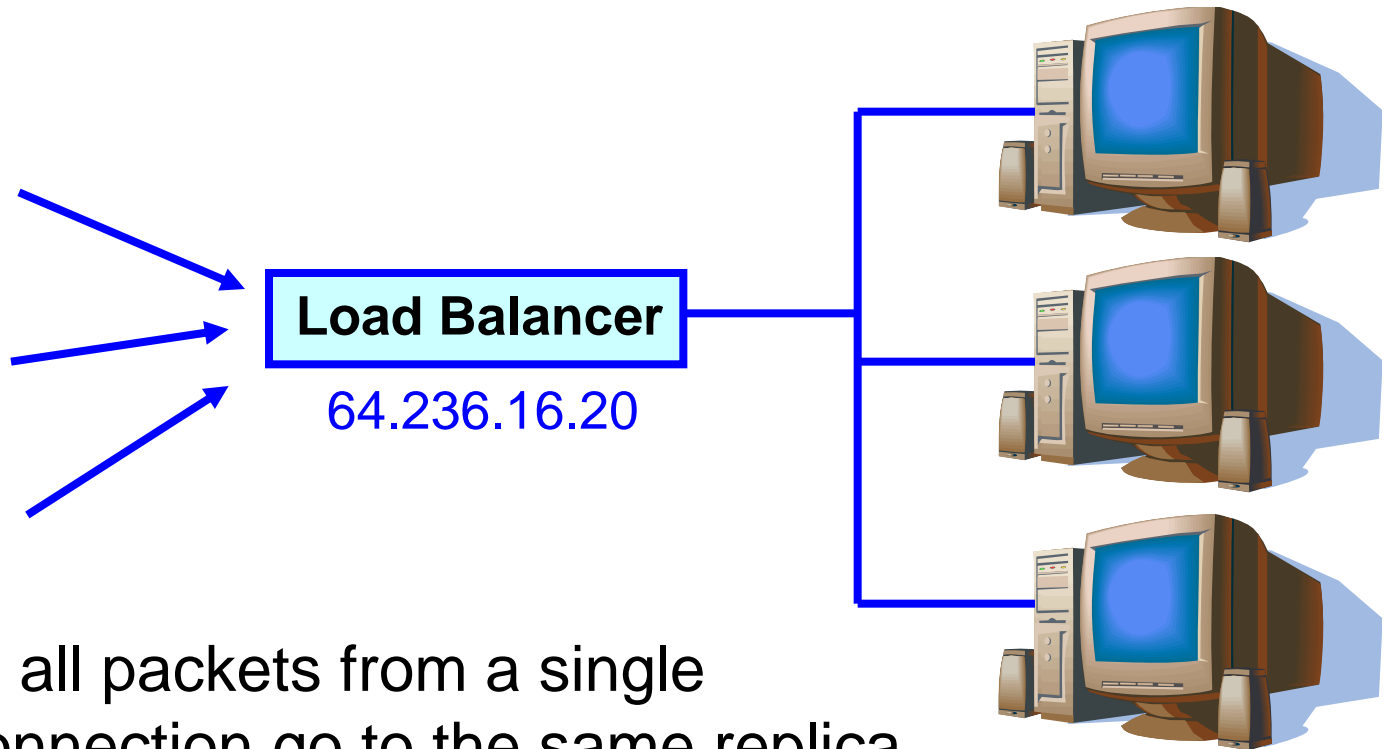
- Multiple Web sites on a single machine
 - Hosting company runs the Web server on behalf of multiple sites (e.g., www.foo.com and www.bar.com)
- Problem: GET /index.html
 - www.foo.com/index.html OR www.bar.com/index.html?
- Solutions:
 - Multiple server processes on the same machine
 - o Have a separate IP address (or port) for each server
 - Include site name in HTTP request
 - o Single Web server process with a single IP address
 - o Client includes “Host” header (e.g., Host: www.foo.com)
 - o *Required header* with HTTP/1.1

Hosting: Multiple Machines Per Site

- Replicate popular Web site across many machines
 - Helps to handle the load
 - Places content closer to clients
- Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
 - Balance load across server replicas
 - Pair clients with nearby servers

Multi-Hosting at Single Location

- Single IP address, multiple machines
 - Run multiple machines behind a single IP address



- Ensure all packets from a single TCP connection go to the same replica

Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
 - Same name but different addresses for all of the replicas
 - Configure DNS server to return different addresses

