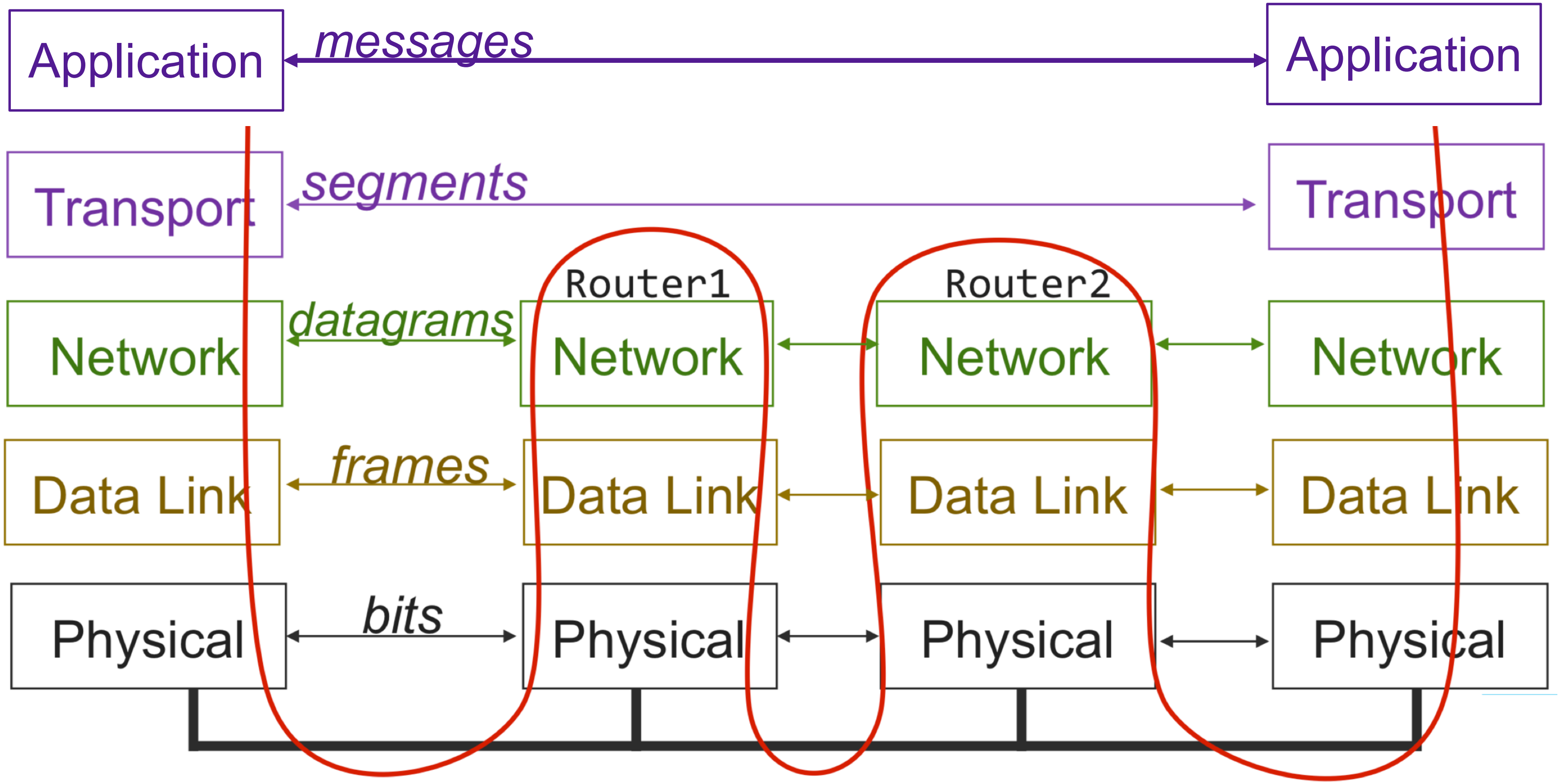
Lecture 20: Networking and the Internet

CS 105

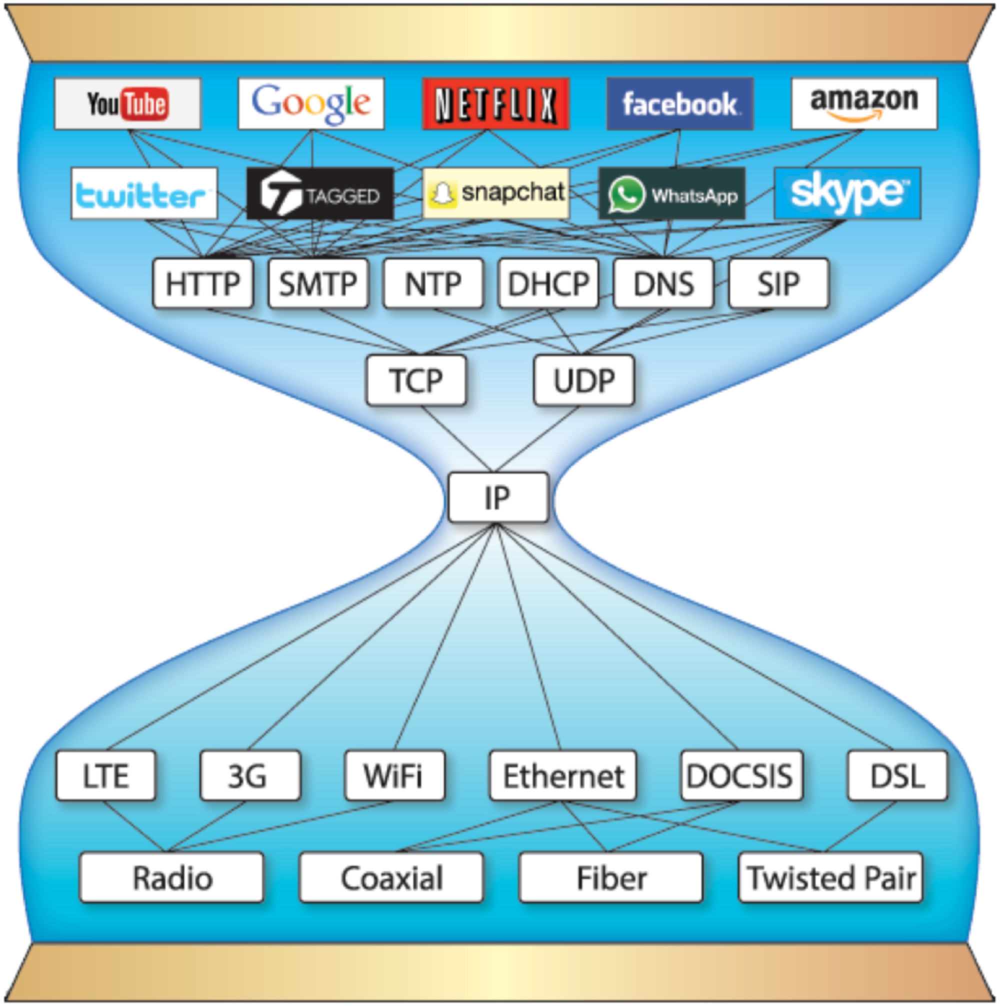
November 19, 2019

What is the Internet?

The Big Picture



Continuing up the Network Stack…



Domain Name System (DNS)

- Principals are identified by names
  - for web hosts, typically a domain name
  - e.g., www.cs.pomona.edu
- Internet hosts are identified by IP addresses
  - used by network layer to route packets between hosts
- The role of DNS is to translate between domain names and IP addresses

Properties of DNS Mappings

- Can explore properties of DNS mappings using `nslookup`

```
linux> nslookup www.cs.pomona.edu
Address: 134.173.71.56
```

- Each host has a locally defined domain name `localhost` which always maps to the **loopback address** `127.0.0.1`

```
linux> nslookup localhost
Address: 127.0.0.1
```

- Use `hostname` to determine real domain name of local host:

```
linux> hostname
pom-nat-84-6.pomona.edu
```

Properties of DNS Mappings (cont)

- Simple case: one-to-one mapping between domain name and IP address:

```
linux> nslookup little.cs.pomona.edu
Address: 134.173.66.223
```

- Multiple domain names mapped to the same IP address:

```
linux> nslookup cs.mit.edu
Address: 18.25.0.23
linux> nslookup eecs.mit.edu
Address: 18.25.0.23
```

Properties of DNS Mappings (cont)

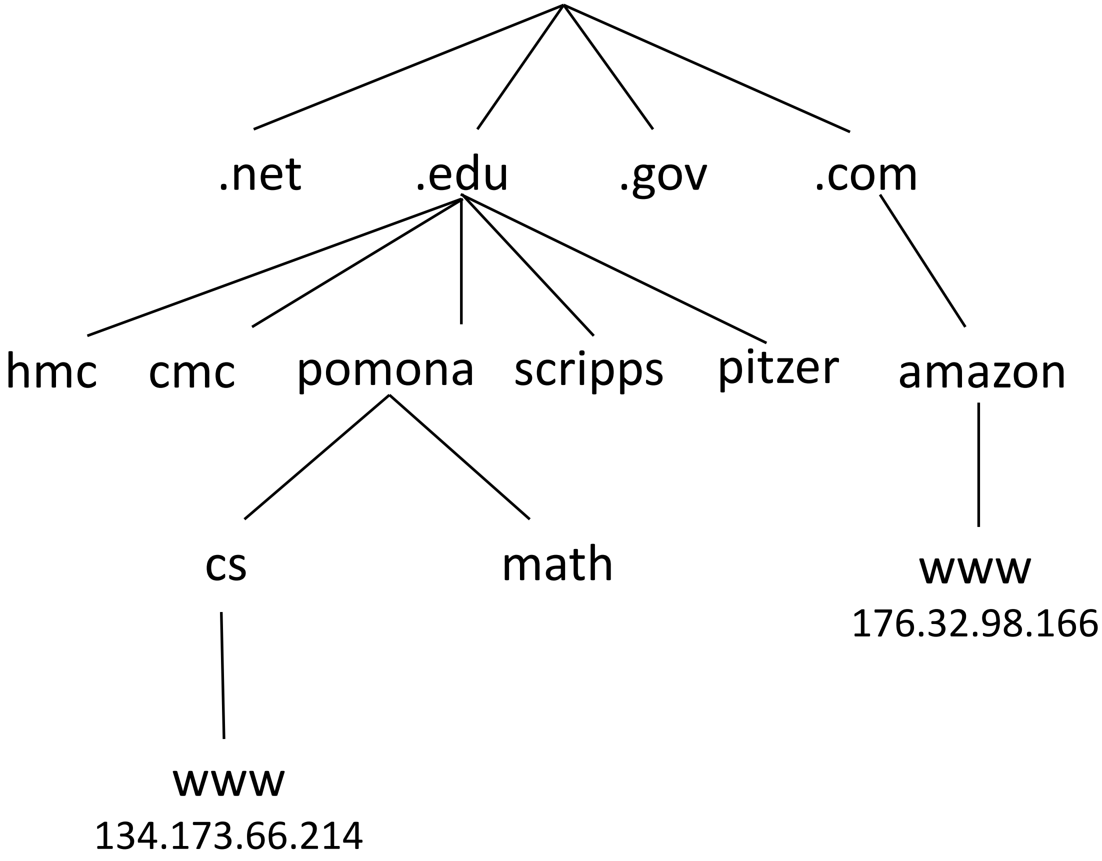
- Multiple domain names mapped to multiple IP addresses:

```
linux> nslookup www.twitter.com
Address: 199.16.156.6
Address: 199.16.156.70
Address: 199.16.156.102
Address: 199.16.156.230

linux> nslookup twitter.com
Address: 199.16.156.102
Address: 199.16.156.230
Address: 199.16.156.6
Address: 199.16.156.70
```

- Some valid domain names don’t map to any IP address:

Domain Name System (DNS)

- Distributed, hierarchical database
- Application-level protocol: hosts and DNS servers communicate to resolve names
- Names are separated into components by dots
- lookup occurs top down

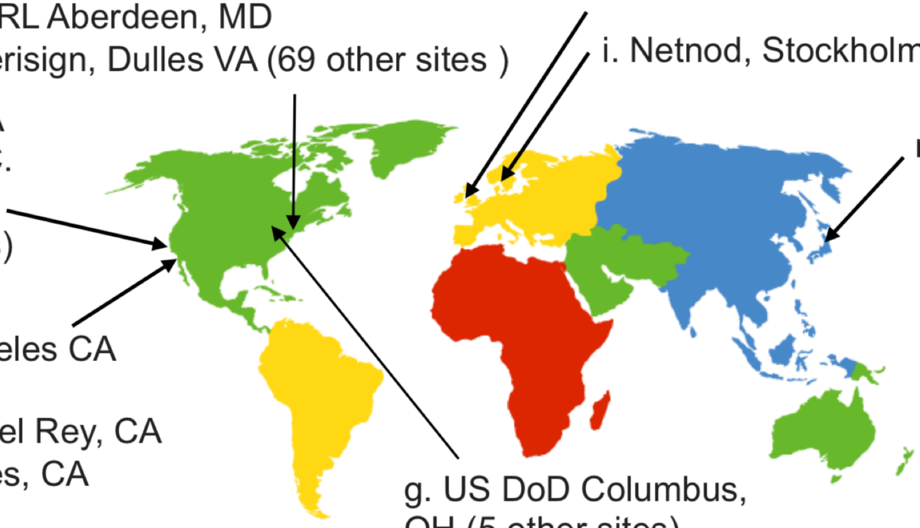


DNS Lookup

- the client asks its local nameserver
- the local nameserver asks one of the *root nameservers*

DNS Root Name Servers

- contacted by local name server that can't resolve name
- owned by Internet Corporation for Assigned Names & Numbers (ICANN)
- contacts authoritative name server if name mapping not known, gets mapping
- returns mapping to local name server



c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

e. NASA Mt View, CA
f. Internet Software C.
Palo Alto, CA
(and 48 other sites)

m. WIDE Tokyo (5 other sites)

13 root name "servers" worldwide

a. Verisign, Los Angeles CA
(5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
(41 other sites)

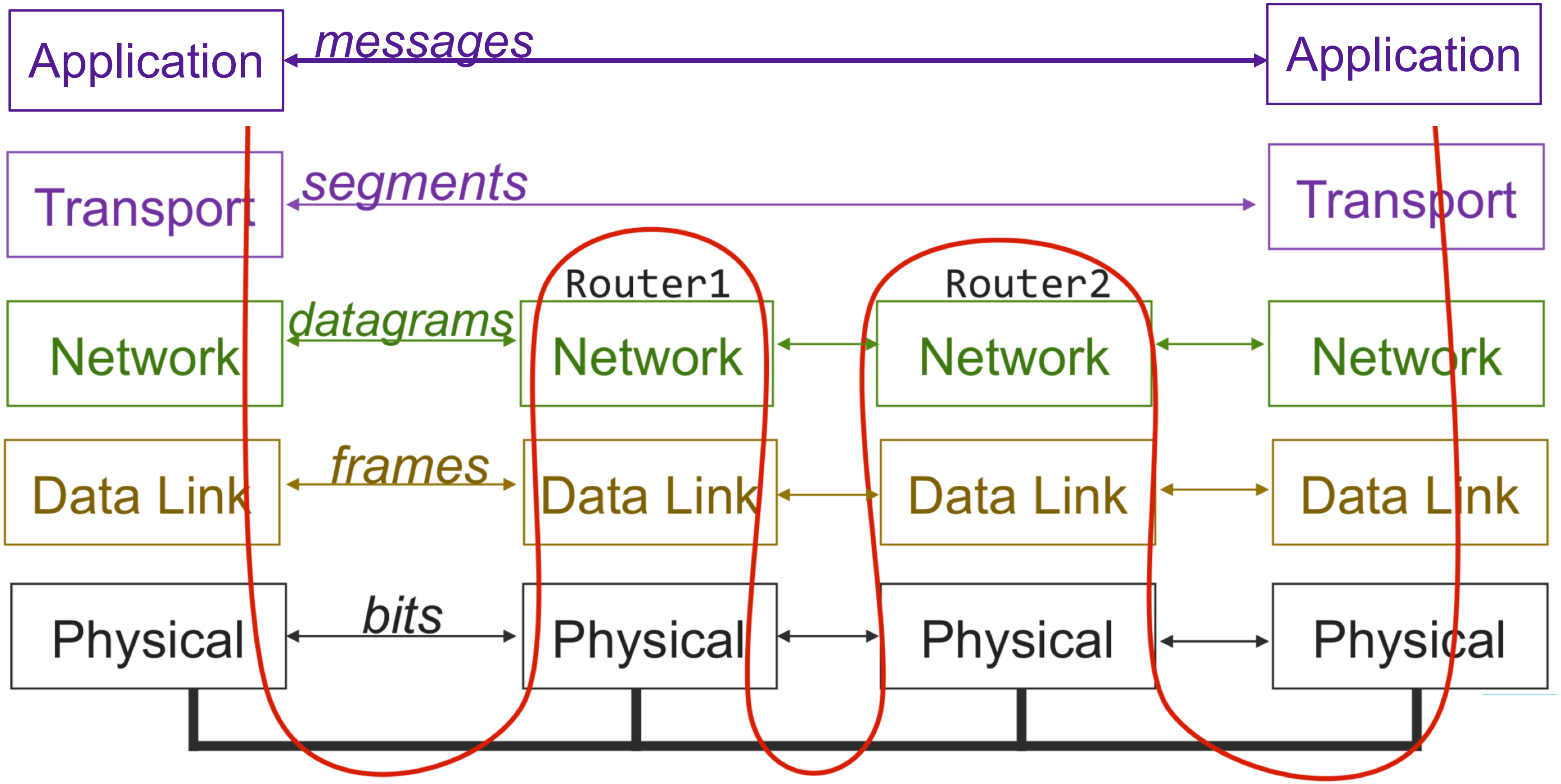
g. US DoD Columbus, OH (5 other sites)

DNS Lookup

- the client asks its local nameserver
- the local nameserver asks one of the *root nameservers*
- the root nameserver replies with the address of the top level nameserver
- the server then queries that nameserver
- the top level nameserver replies with the address of the authoritative nameserver
- the server then queries that nameserver
- repeat until host is reached, cache result.

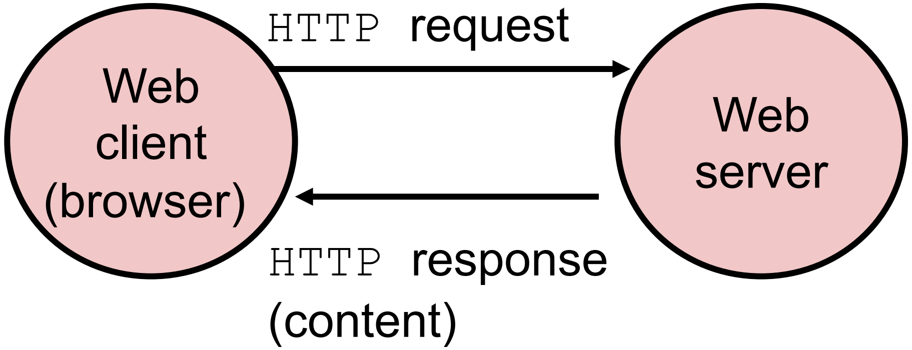
- Example: Client wants IP addr of www.amazon.com
  1. Queries root server to find com DNS server
  2. Queries .com DNS server to get amazon.com DNS server
  3. Queries amazon.com DNS server to get IP address for www.amazon.com

The Big Picture



HTTP

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (eventually)
- Current version is HTTP/2.0
  - RFC 7540, 2015
  - Includes protocol negotiation
  - HTTP/1.1 still in use (RFC 2616, 1999)
  - HTTP/3 proposed



Web client (browser)

`HTTP` request

Web server

`HTTP` response (content)

HTTP	Web content
TCP	Streams
IP	Datagrams

URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: http://www.cs.pomona.edu:80classes/cs105/index.html

- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
  - What kind (protocol) of server to contact (HTTP)
  - Where the server is (`www.cs.pomona.edu`)
  - What port it is listening on (80)

- Servers use *suffix* (`/classes/cs105/index.html`) to:
  - Determine if request is for static or dynamic content.
    - No hard and fast rules for this
    - One convention: executables reside in `cgi-bin` directory
  - Find file on file system
    - Initial "`/`" in suffix denotes home directory for requested content.
    - Minimal suffix is "`/`", which server expands to configured default filename (usually, `index.html`)

HTTP Requests

- HTTP request is a **request line**, followed by zero or more **request headers**

- Request line: `<method> <uri> <version>`
  - `<method>` is one of `GET, POST, OPTIONS, HEAD, PUT, DELETE,` or `TRACE`
  - `<uri>` is typically URL for proxies, URL suffix for servers
    - A URL is a type of URI (Uniform Resource Identifier)
    - See http://www.ietf.org/rfc/rfc2396.txt
  - `<version>` is HTTP version of request (`HTTP/1.0` or `HTTP/1.1`)

- Request headers: `<header name>: <header data`>
  - Provide additional information to the server

HTTP Responses

- HTTP response is a **response line** followed by zero or more **response headers**, possibly followed by **content**
  - a blank line (“`\r\n`”) separates headers from content.

- Response line: `<version> <status code> <status msg>`
  - <version> is HTTP version of the response
  - <status code> is numeric status
  - <status msg> is corresponding English text

200	OK	Request was handled without error
301	Moved	Provide alternate URL
404	Not found	Server couldn’t find the file

- Response headers: `<header name>: <header data>`
  - Provide additional information about response
  - `Content-Type:` MIME type of content in response body
  - `Content-Length:` Length of content in response body

Web Content

- Web servers return content to clients
  - *content:* a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

- Example MIME types

• `text/html`	HTML document
• `text/plain`	Unformatted text
• `image/gif` format	Binary image encoded in GIF
• `image/png` format	Binar image encoded in PNG
• `image/jpeg` format	Binary image encoded in JPEG

You can find the complete list of MIME types at:
`http://www.iana.org/assignments/media-types/media-types.xhtml`

Static and Dynamic Content

- The content returned in HTTP responses can be either **static** or **dynamic**
  - *Static content*: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
    - Request identifies which content file
  - *Dynamic content*: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
    - Request identifies file containing executable code
- Bottom line: *Web content is associated with a file that is managed by the server*

Tiny Web Server

- Tiny Web server described in text
  - Tiny is a sequential Web server
  - Serves static and dynamic content to real browsers
    - text files, HTML files, GIF, PNG, and JPEG images
  - 239 lines of commented C code
  - Not as complete or robust as a real Web server
    - You can break it with poorly-formed HTTP requests (e.g., terminate lines with “\n” instead of “\r\n”)

Tiny Operation

- Accept connection from client
- Read request from client (via connected socket)
- Split into <method> <uri> <version>
  - If method not GET, then return error
- If URI contains "`cgi-bin`" then serve dynamic content
  - (Would do wrong thing if had file "`abcgi-bingo.html`")
  - Fork process to execute program
- Otherwise serve static content
  - Copy file to output

Tiny Serving Static Content

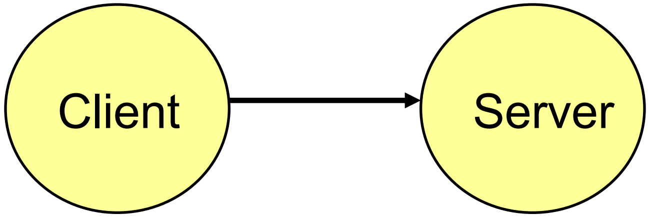
```
void serve_static(int fd, char *filename, int filesize)
{
    int srcfd;
    char *srcp, filetype[MAXLINE], buf[MAXBUF];

    /* Send response headers to client */
    get_filetype(filename, filetype);
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);
    sprintf(buf, "%sConnection: close\r\n", buf);
    sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);
    sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);
    Rio_writen(fd, buf, strlen(buf));

    /* Send response body to client */
    srcfd = Open(filename, O_RDONLY, 0);
    srcp = Mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0);
    Close(srcfd);
    Rio_writen(fd, srcp, filesize);
    Munmap(srcp, filesize);
}
```

tiny.c

Serving Dynamic Content

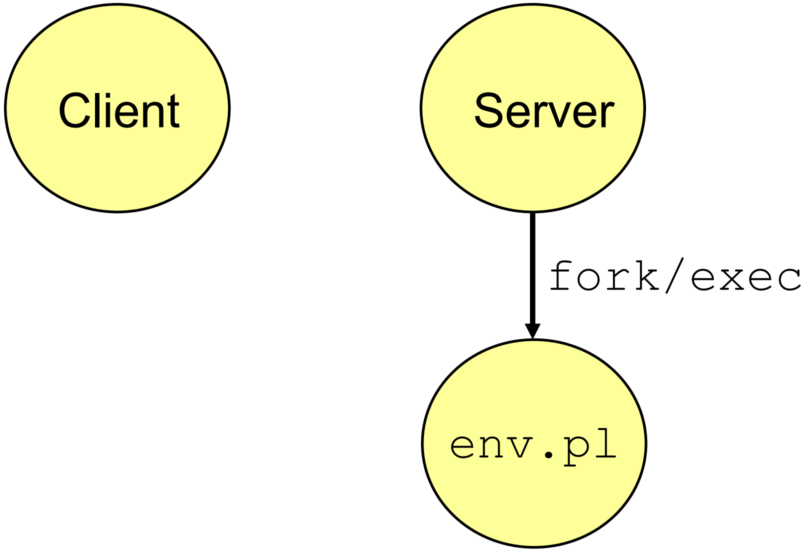
- Client sends request to server
- If request URI contains the string "`/cgi-bin`", the Tiny server assumes that the request is for dynamic content

```
GET /cgi-bin/env.pl HTTP/1.1
```



Serving Dynamic Content (cont)

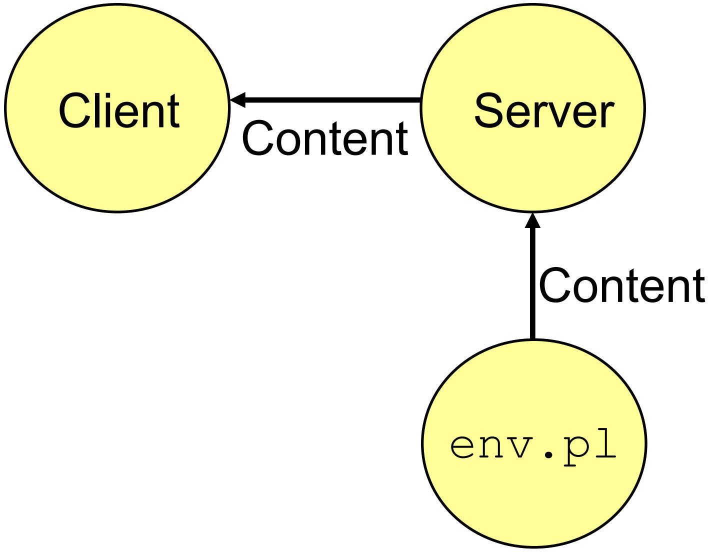
- The server creates a child process and runs the program identified by the URI in that process



Serving Dynamic Content (cont)

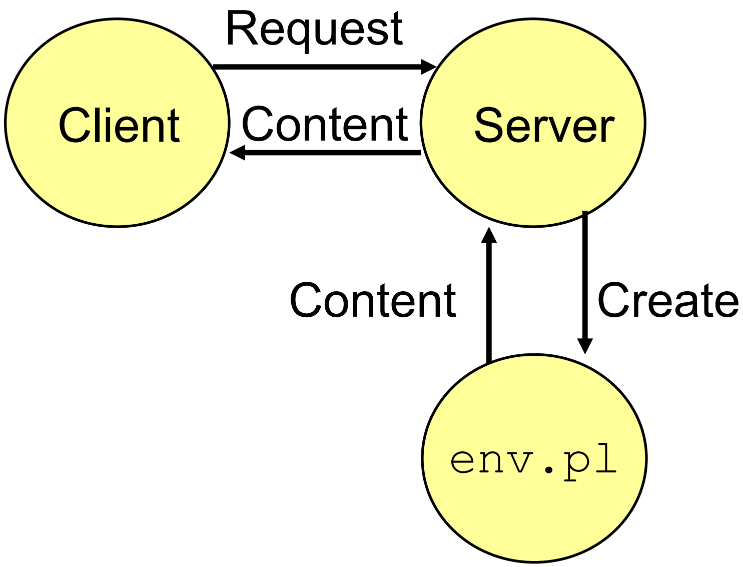
- The child runs and generates the dynamic content

- The server captures the content of the child and forwards it without modification to the client



Issues in Serving Dynamic Content

- How does the client pass program arguments to the server?
- How does the server pass these arguments to the child?
- How does the server pass other info relevant to the request to the child?
- How does the server capture the content produced by the child?
- These issues are addressed by the Common Gateway Interface (CGI) specification.



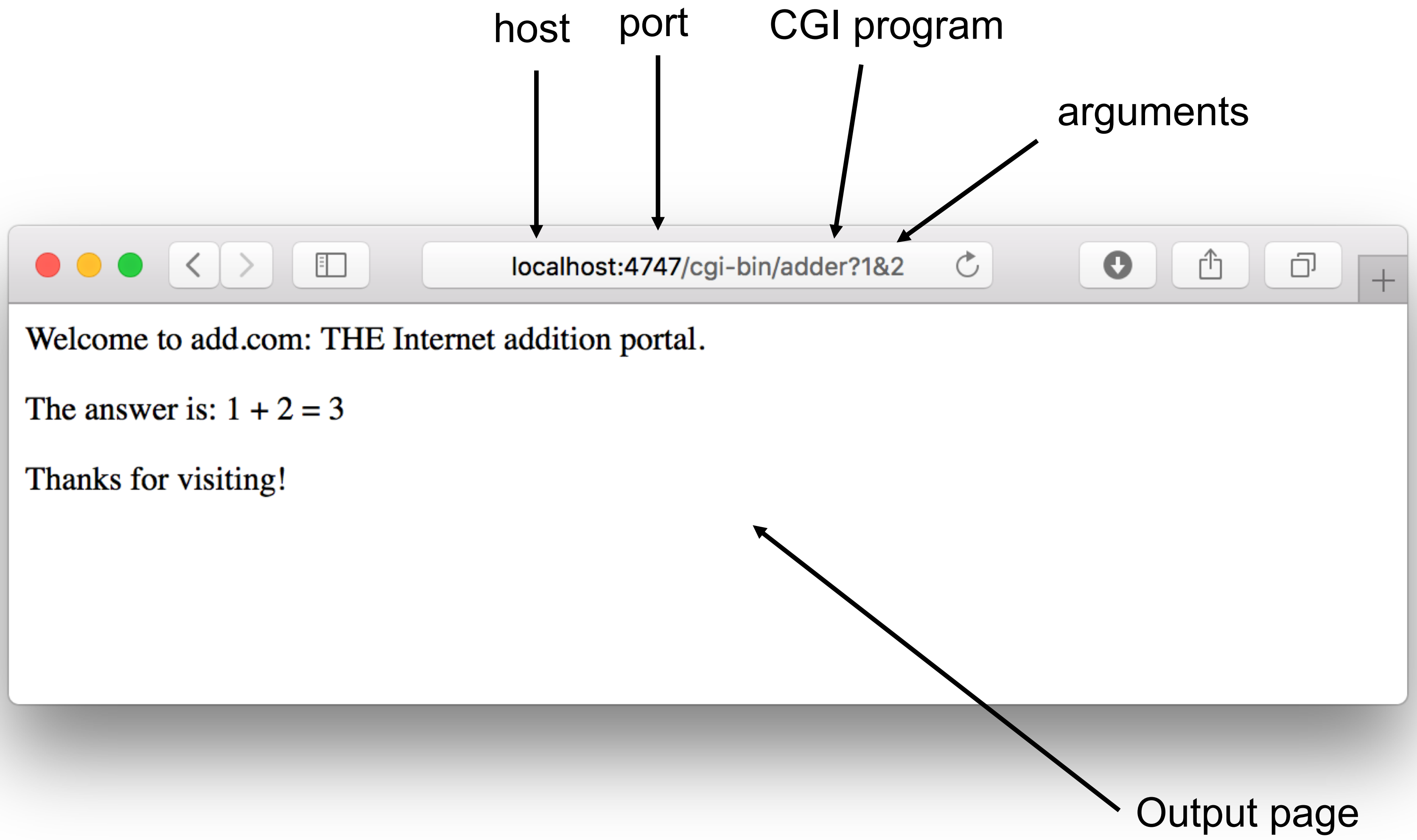
CGI

- Because the children are written according to the CGI spec, they are often called *CGI programs.*

- However, CGI really defines a simple standard for transferring information between the client (browser), the server, and the child process.

- CGI is the original standard for generating dynamic content. Has been largely replaced by other, faster techniques:
  - E.g., fastCGI, Apache modules, Java servlets, Rails controllers
  - Avoid having to create process on the fly (expensive and slow).

The add.com Experience



host

port

CGI program

arguments

localhost:4747/cgi-bin/adder?1&2

Welcome to add.com: THE Internet addition portal.

The answer is: 1 + 2 = 3

Thanks for visiting!

Output page

Serving Dynamic Content With GET

- Question: How does the client pass arguments to the server?
- Answer: The arguments are appended to the URI

- Can be encoded directly in a URL typed to a browser or a URL in an HTML link
  - `http://add.com/cgi-bin/adder?15213&18213`
  - `adder` is the CGI program on the server that will do the addition.
  - argument list starts with `“?”`
  - arguments separated by `“&”`
  - spaces represented by `“+” or “%20”`

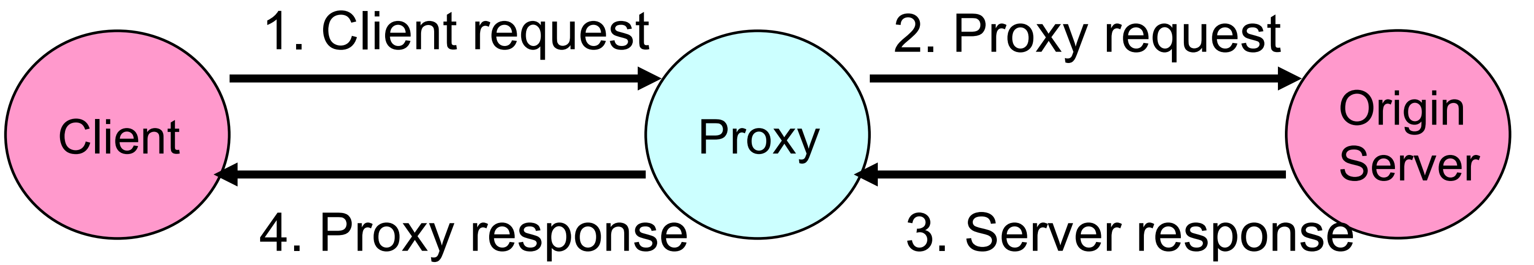
Testing Servers Using `telnet`

- The `telnet` program is invaluable for testing servers that transmit ASCII strings over Internet connections
  - Our simple echo server
  - Web servers
  - Mail servers

- Usage:
  - **`linux>`** ***`telnet <host> <portnumber>`***
  - Creates a connection with a server running on ***`<host>`*** and listening on port ***`<portnumber>`***

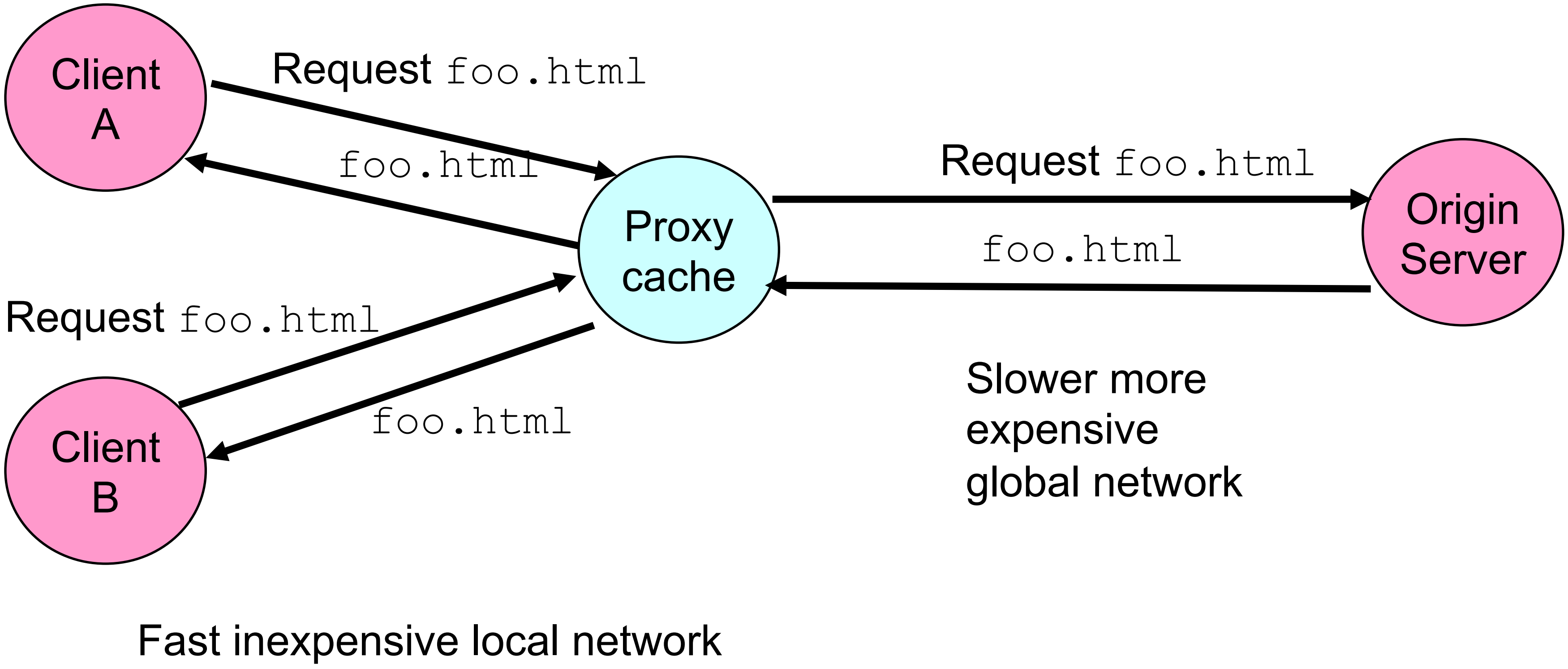
Proxies

- A **proxy** is an intermediary between a client and a server
  - To the client, the proxy acts like a server
  - To the server, the proxy acts like a client



Why Proxies?

- Can perform useful functions as requests and responses pass by
  - Examples: Caching, logging, anonymization, filtering



Client A
Request foo.html
foo.html
Proxy cache
Request foo.html
Origin Server
foo.html
Request foo.html
Client B
foo.html
Slower more expensive global network
Fast inexpensive local network