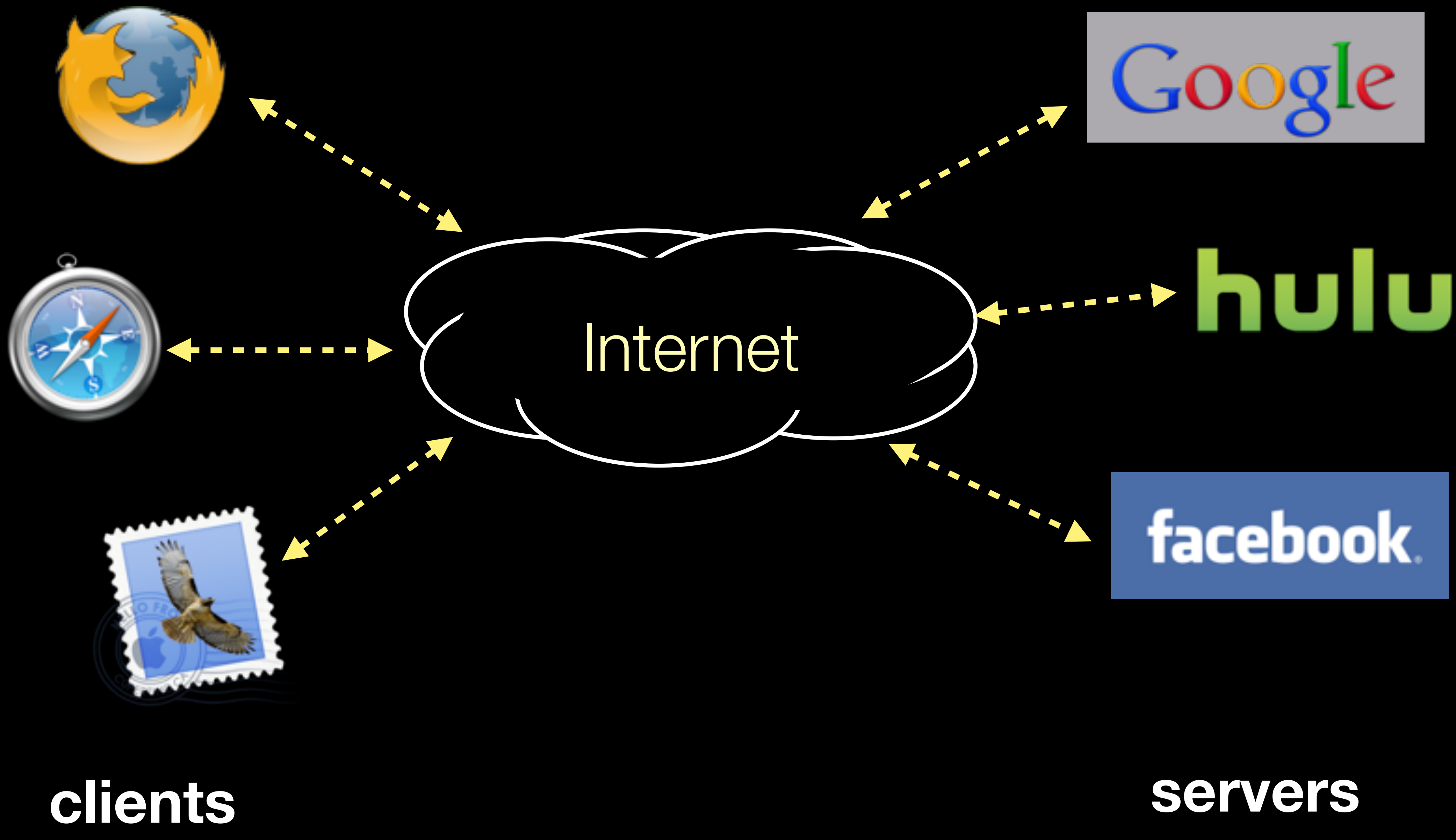
CSE 333 Section 3

Thursday 12 April 2012

Goals for Today

1. Overview IP addresses
2. Look at the IP address structures in C/C++
3. Overview DNS
4. Talk about how to use DNS to translate IP addresses
5. Write your own (short!) program to do this translation
6. Go over the solution

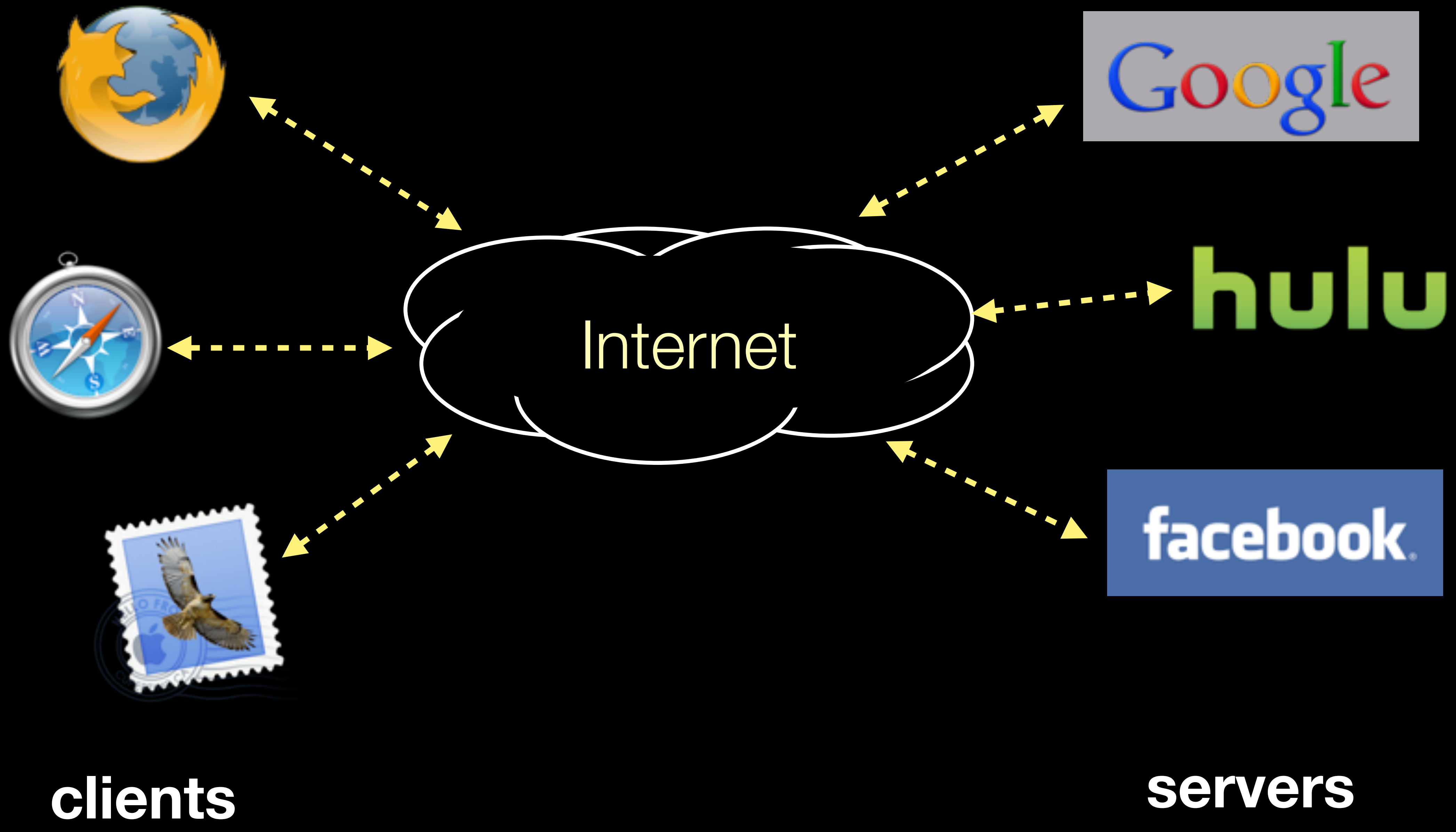
Networks from 10,000ft



Internet

clients

servers



Internet

clients

servers

- Clients talk to Servers
- Servers respond to Clients

... But how do they know how to reach each other?

... And how do we know if a response is for Firefox or Mail?

Network addresses

For IPv4, an IP address is a 4-byte tuple

- e.g., 128.95.4.1 (80:5f:04:01 in hex)

For IPv6, an IP address is a 16-byte tuple

- e.g., 2d01:0db8:f188:0000:0000:0000:0000:1f33
    - 2d01:0db8:f188::1f33 in shorthand

There are lots of structs coming up...

... we’ll walk through them one at a time.

IPv4 address structures

```
// Port numbers and addresses are in *network order*.

// A mostly-protocol-independent address structure.
struct sockaddr {
    short int     sa_family;    // Address family; AF_INET, AF_INET6
    char          sa_data[14];  // 14 bytes of protocol address
};

// An IPv4 specific address structure.
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET == IPv4
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};

struct in_addr {
    uint32_t s_addr;  // IPv4 address
};
```

IPv6 address structures

```
// A structure big enough to hold either IPv4 or IPv6 structures.
struct sockaddr_storage {
    sa_family_t  ss_family;     // address family

    // a bunch of padding; safe to ignore it.
    char         __ss_pad1[_SS_PAD1SIZE];
    int64_t      __ss_align;
    char         __ss_pad2[_SS_PAD2SIZE];
};

// An IPv6 specific address structure.
struct sockaddr_in6 {
    u_int16_t       sin6_family;   // address family, AF_INET6
    u_int16_t       sin6_port;     // Port number
    u_int32_t       sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    u_int32_t       sin6_scope_id; // Scope ID
};

struct in6_addr {
    unsigned char   s6_addr[16];   // IPv6 address
};
```

Generating these structures

Often you have a string representation of an address

- how do you generate one of the address structures?

```
#include <stdlib.h>
#include <arpa/inet.h>

int main(int argc, char **argv) {
  struct sockaddr_in sa; // IPv4
  struct sockaddr_in6 sa6; // IPv6

  // IPv4 string to sockaddr_in.
  inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr));

  // IPv6 string to sockaddr_in6.
  inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));

  return EXIT_SUCCESS;
}
```

Generating these structures

How about going in reverse?

```
#include <stdlib.h>
#include <arpa/inet.h>

int main(int argc, char **argv) {
  struct sockaddr_in6 sa6;          // IPv6
  char astring[INET6_ADDRSTRLEN];   // IPv6

  // IPv6 string to sockaddr_in6.
  inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));

  // sockaddr_in6 to IPv6 string.
  inet_ntop(AF_INET6, &(sa6.sin6_addr), astring, INET6_ADDRSTRLEN);
  printf(“%s\n”, astring);

  return EXIT_SUCCESS;
}
```

DNS

People tend to use DNS names, not IP addresses

- the sockets API lets you convert between the two
- it’s a complicated process, though:
  - a given DNS name can have many IP addresses
  - many different DNS names can map to the same IP address
    - an IP address will reverse map into at most one DNS names, and maybe none
  - a DNS lookup may require interacting with many DNS servers

You can use the “dig” Linux program to explore DNS

- “man dig”

DNS hierarchy

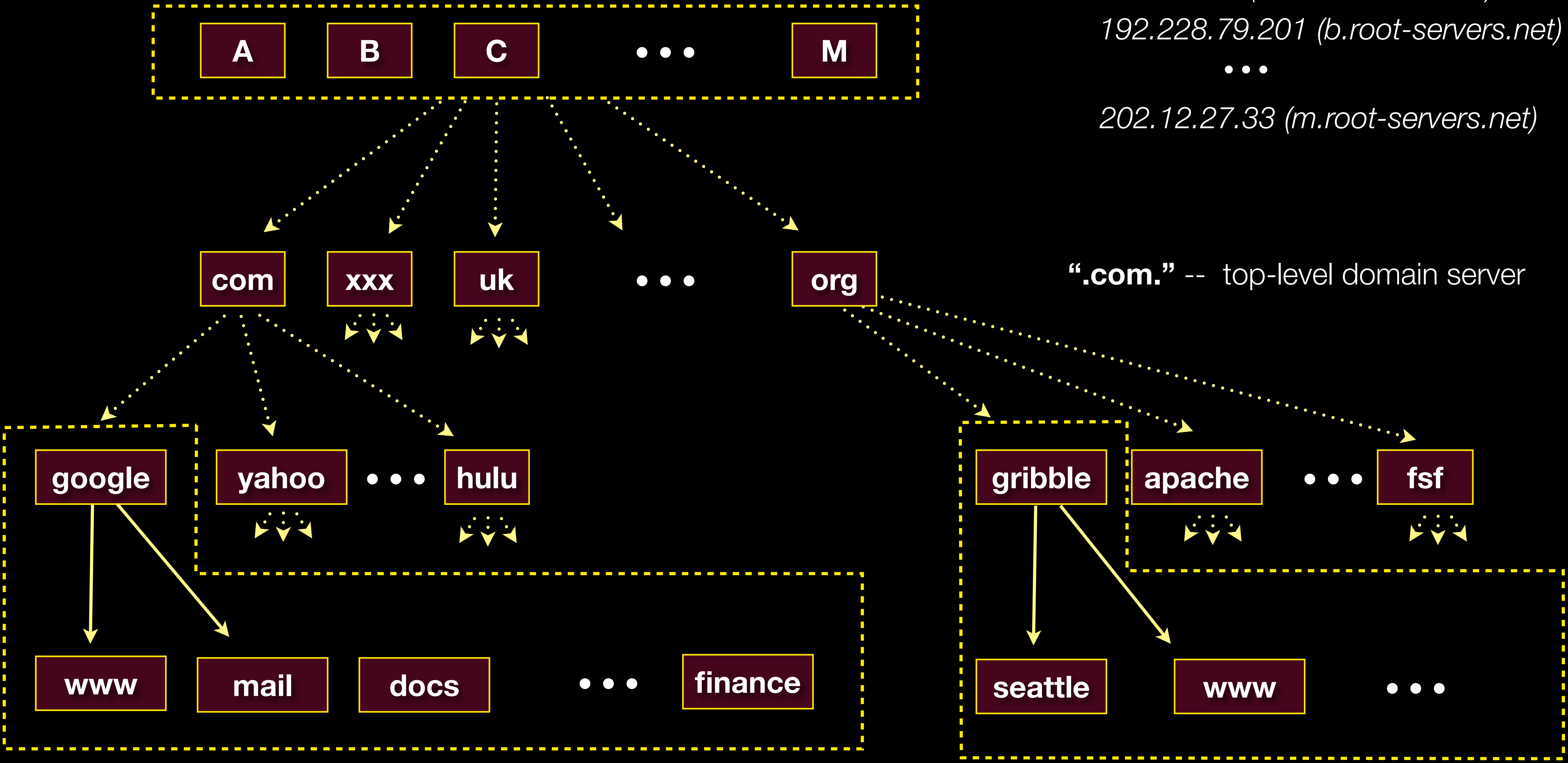
“.” -- root name servers

198.41.0.4 (a.root-servers.net)

192.228.79.201 (b.root-servers.net)

• • •

202.12.27.33 (m.root-servers.net)



A B C • • • M

com xxx uk • • • org

google yahoo • • • hulu

www mail docs • • • finance

gribble apache • • • fsf

seattle www • • •

“.com.” -- top-level domain server

Resolving DNS names

The POSIX way is to use **getaddrinfo( )**

- a pretty complicated system call; the basic idea...
  - set up a “hints” structure with constraints you want respected
    - e.g., IPv6, IPv4, or either
  - tell getaddrinfo( ) which host and port you want resolved
    - host: a string representation; DNS name or IP address
  - getaddrinfo( ) gives you a list of results packet in an “addrinfo” struct
  - free the addrinfo structure using freeaddrinfo( )

getaddrinfo() and structures

```
int getaddrinfo(const char *hostname,         // hostname to look up
                const char *servname,         // service name
                const struct addrinfo *hints, //desired output type
                struct addrinfo **res);       //result structure

// Hints and results take the same form. Hints are optional.
struct addrinfo {
    int             ai_flags;     // Indicate options to the function
    int             ai_family;    // AF_INET, AF_INET6, or AF_UNSPEC
    int             ai_socktype;  // Socket type, (use SOCK_STREAM)
    int             ai_protocol;  // Protocol type
    size_t          ai_addrlen;   // INET_ADDRSTRLEN, INET6_ADDRSTRLEN
    char            *ai_cananname;// canonical name for the host
    struct sockaddr *ai_addr;     // Address (input to inet_ntop)
    struct addrinfo *ai_next;     // Next element (It’s a linked list)
};

// Converts an address from network format to presentation format
const char *inet_ntop(int af,                    // family (see above)
                      const void * restrict src, // sockaddr
                      char * restrict dest,      // return buffer
                      socklen_t size);           // length of buffer
```

DNS lookup program

- Take in an argument to translate to ip (e.g. “google.com”)
- If you don’t want to take in an argument look up my CSE machine:
  - “cerise.cs.washington.edu” ---> 128.208.6.34
- Setup/initialize your hints addrinfo struct (remember to free it later!)
  - zero out everything except ai_family and ai_socktype
- Use getaddrinfo() to ask DNS for the IP
  - you can use gai_strerror() to translate error codes
- Cycle through returned addresses, printing results
  - use inet_ntop() to get a nice string out
  - you need to distinguish between IPv4 and IPv6

Don’t worry about getting it perfect, we just want you to work with the structures and be familiar with them.

Let’s go over the solution...