第11章 网络编程
Network Programming: Part I

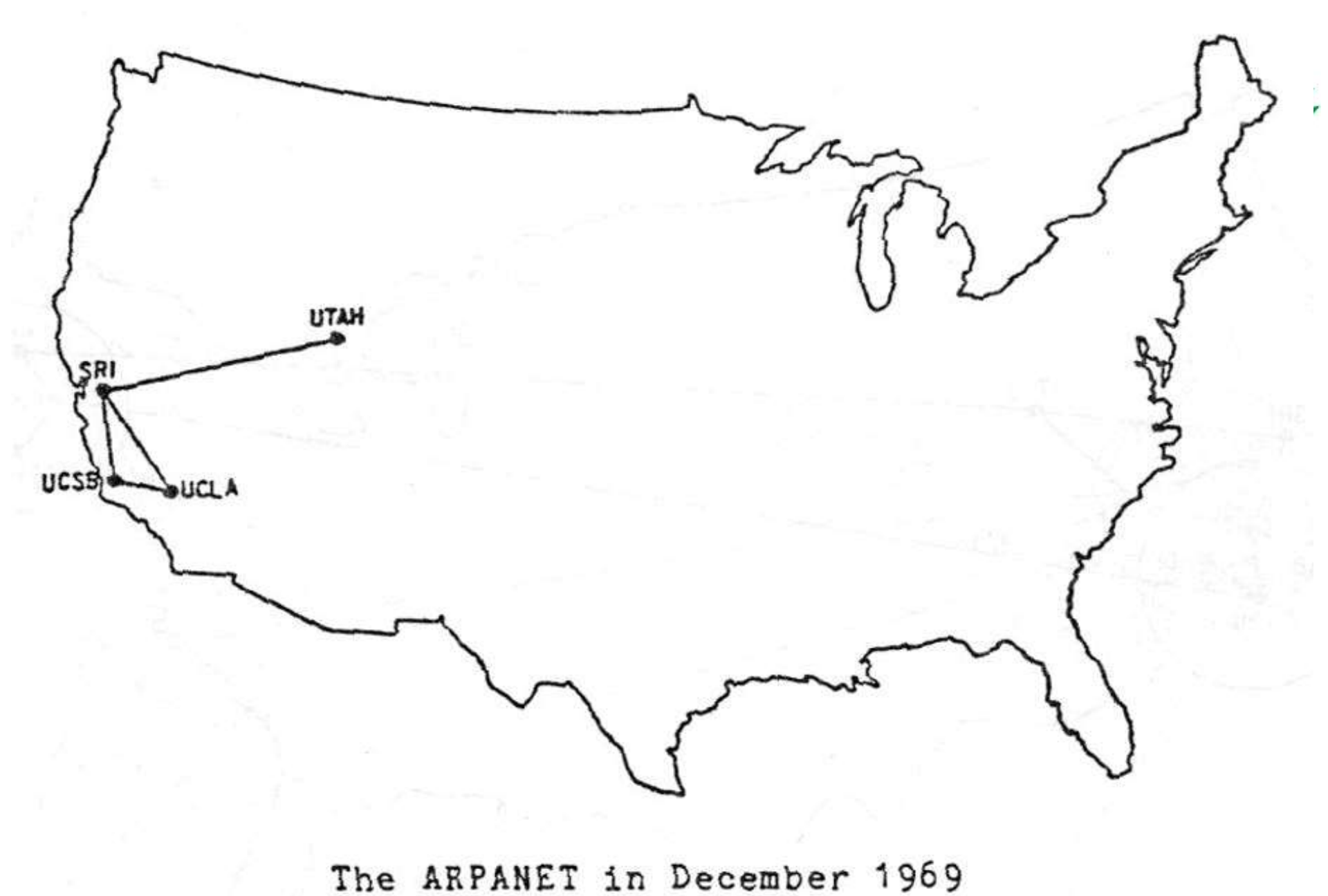
100076202： 计算机系统导论

任课教师：

宿红毅　　张艳　　黎有琦　　颜珂

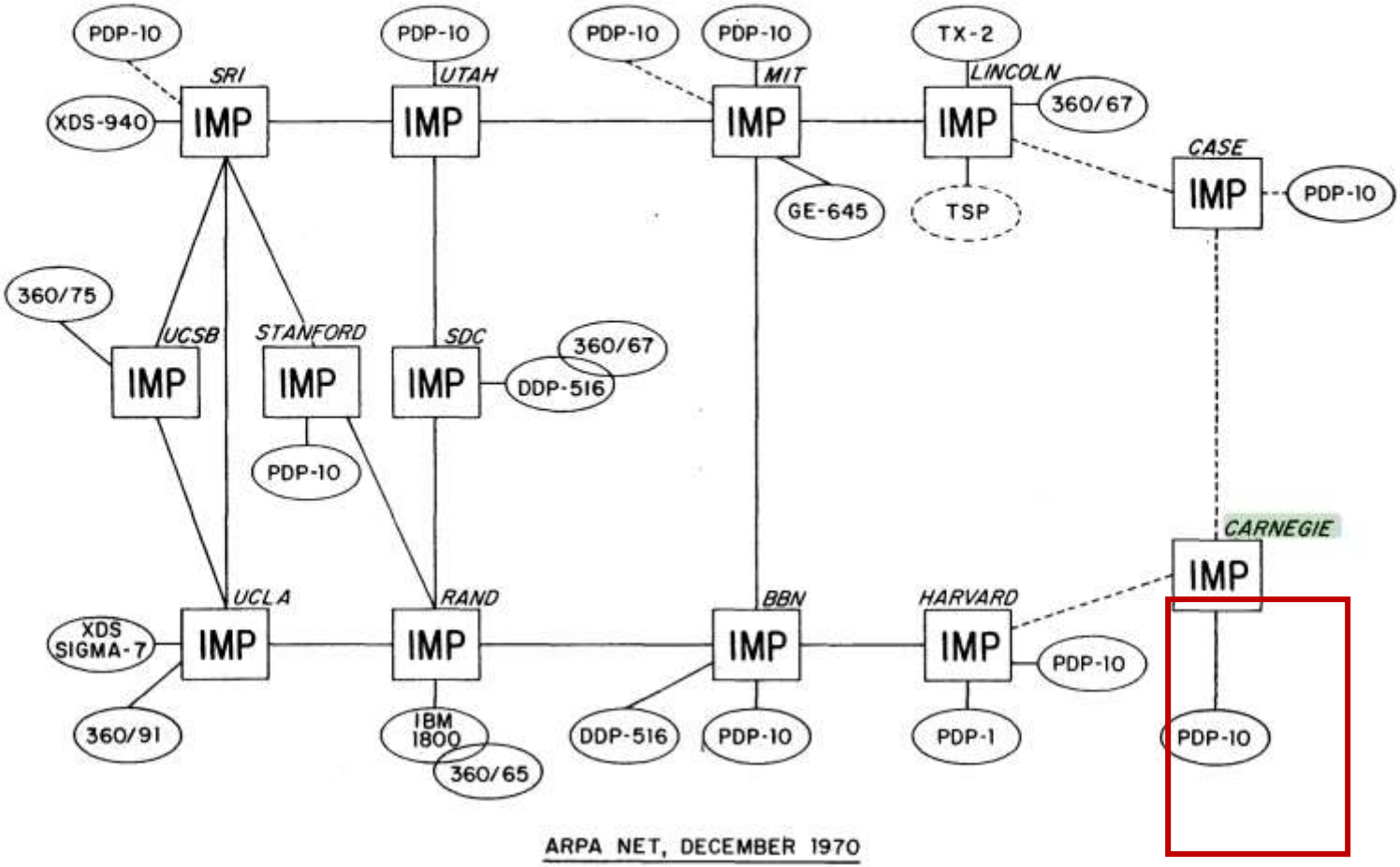
原作者：

Randal E. **Bryant and** David R. O'Hallaron



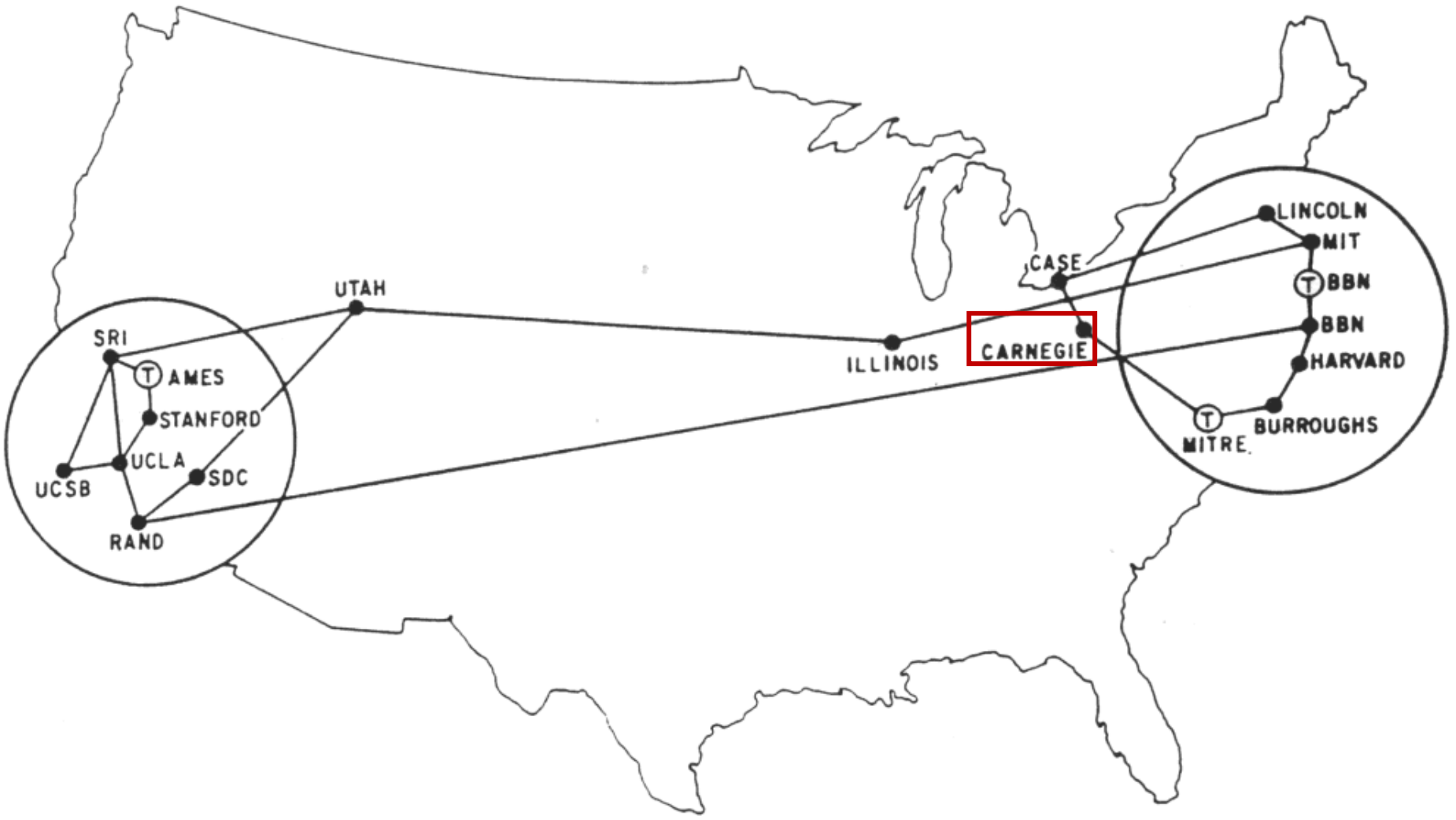
The ARPANET in December 1969

1969年12月ARPA网诞生（Internet前身）

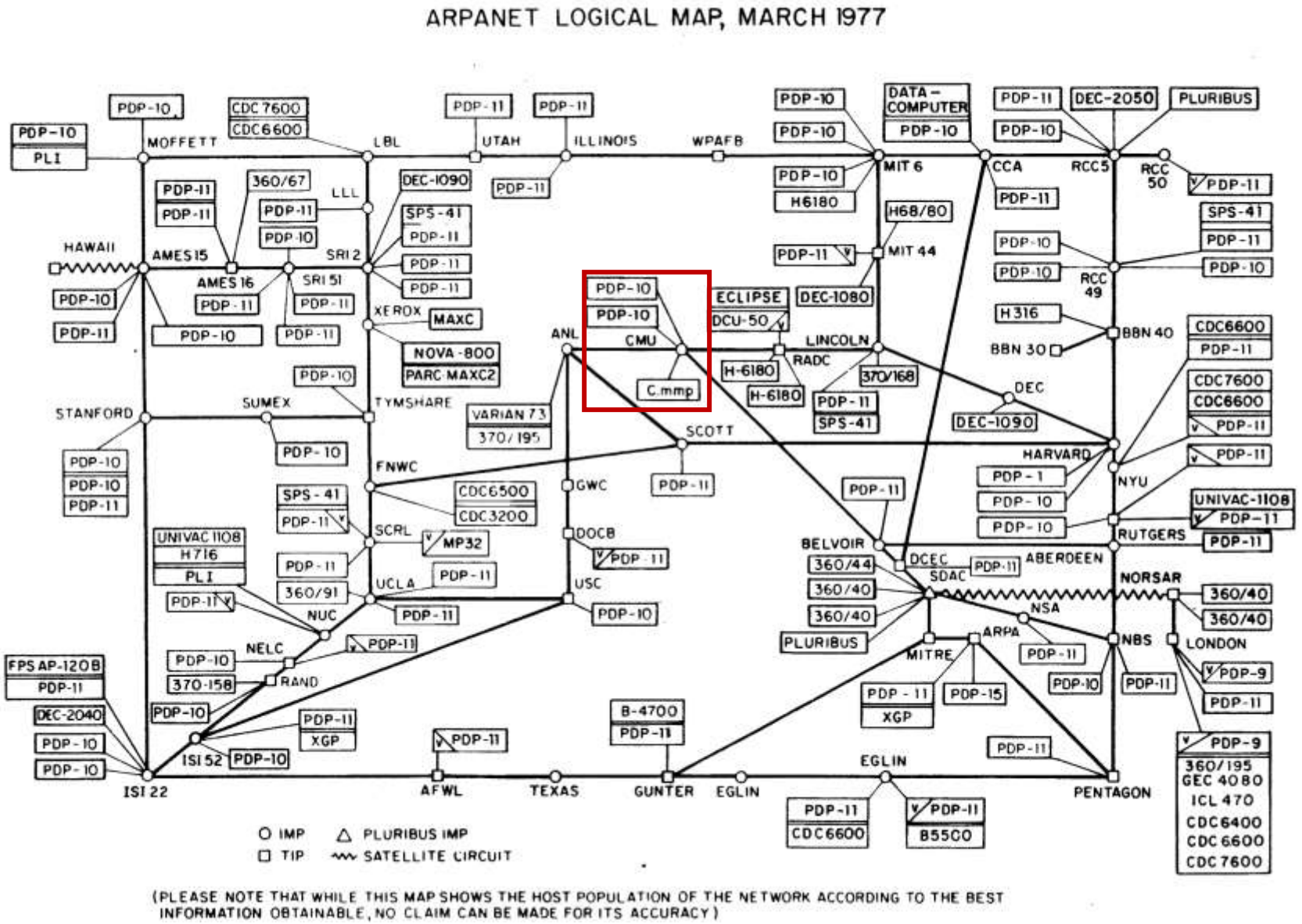


IMP： Interface Message Processor

接口报文处理机


SRI
AMES
STANFORD
UCLA
SDC
UCSB
RAND
UTAH
ILLINOIS
CASE
CARNEGIE
LINCOLN
MIT
BBN
BBN
HARVARD
BURROUGHS
MITRE

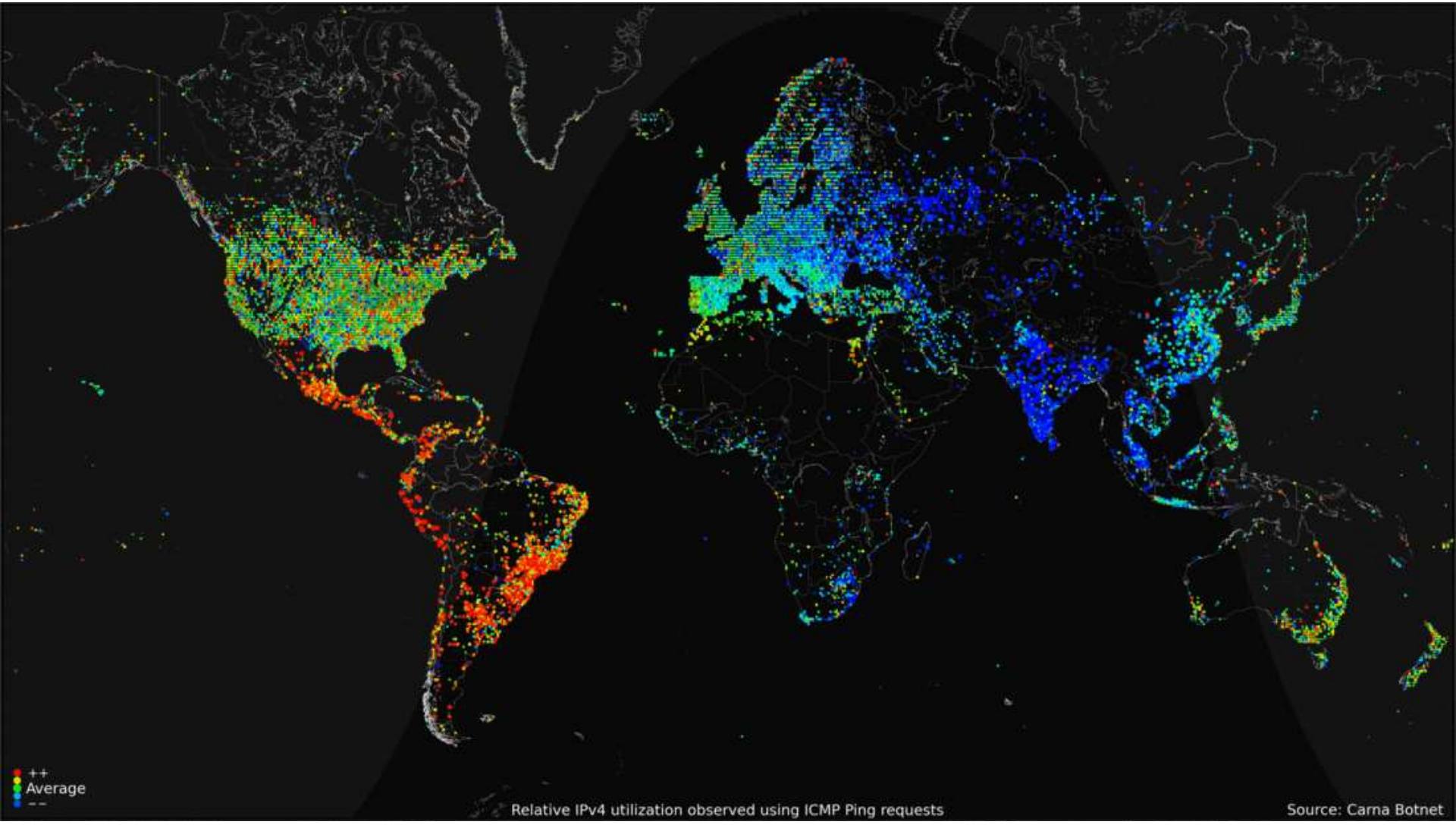
ARPA网逻辑地图，1977年3月



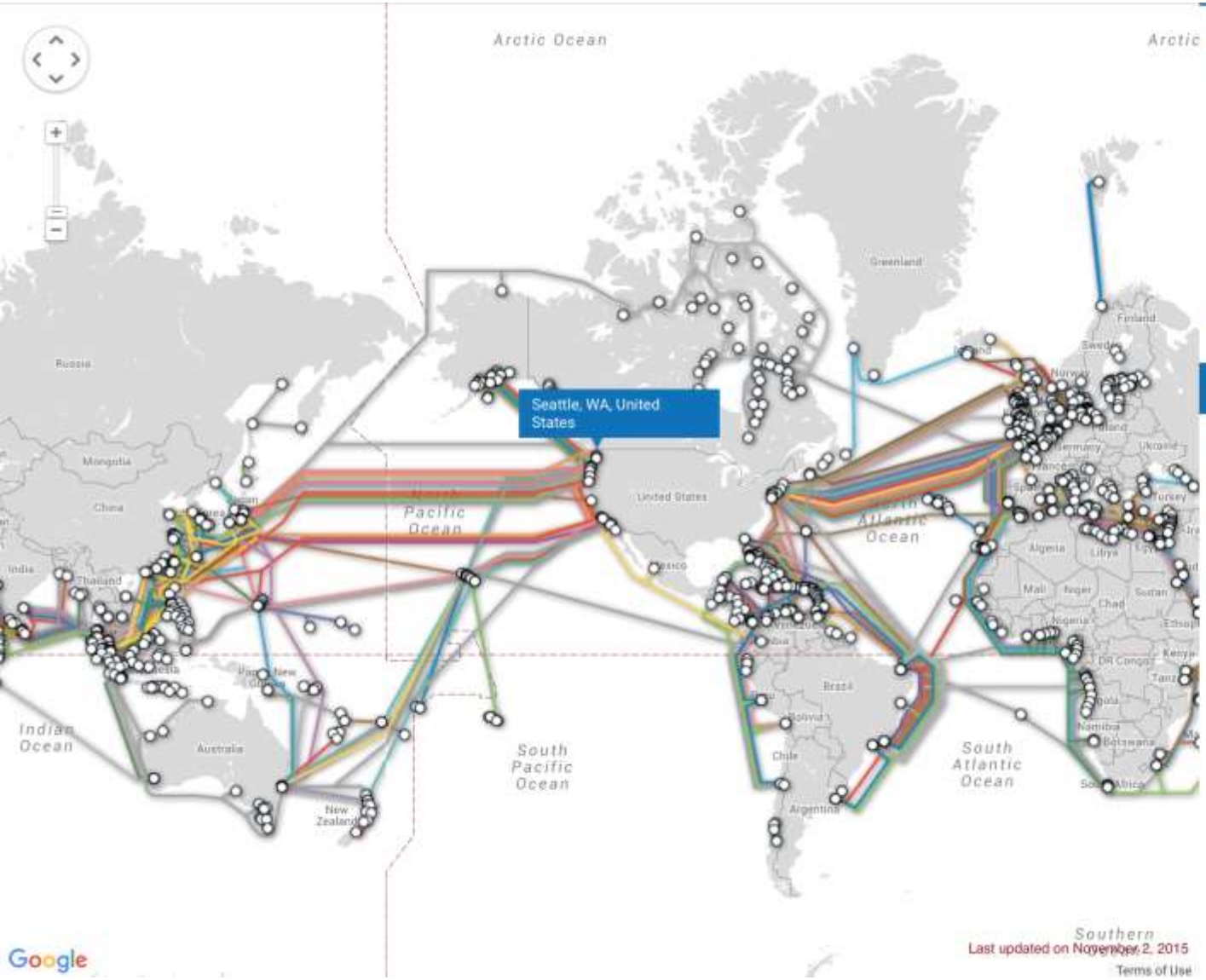
(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

Carna Botnet收集的4600亿设备连接互联网地图

A Map of 460 Billion Device Connections to the Internet collected by the Carna Botnet(僵尸网络)



海底电缆图 Submarine Cable Map



TeleGeography

Submarine Cable Map

The Submarine Cable Map is a free resource from TeleGeography. Data contained in this map is drawn from the Global Bandwidth Research Service and is updated on a regular basis.

To learn more about TeleGeography or this map please click here.

HUAWEI MARINE NETWORKS

Sponsored in part by Huawei Marine

Feedback github

Search

Submarine Cable List

Seattle, WA, United States

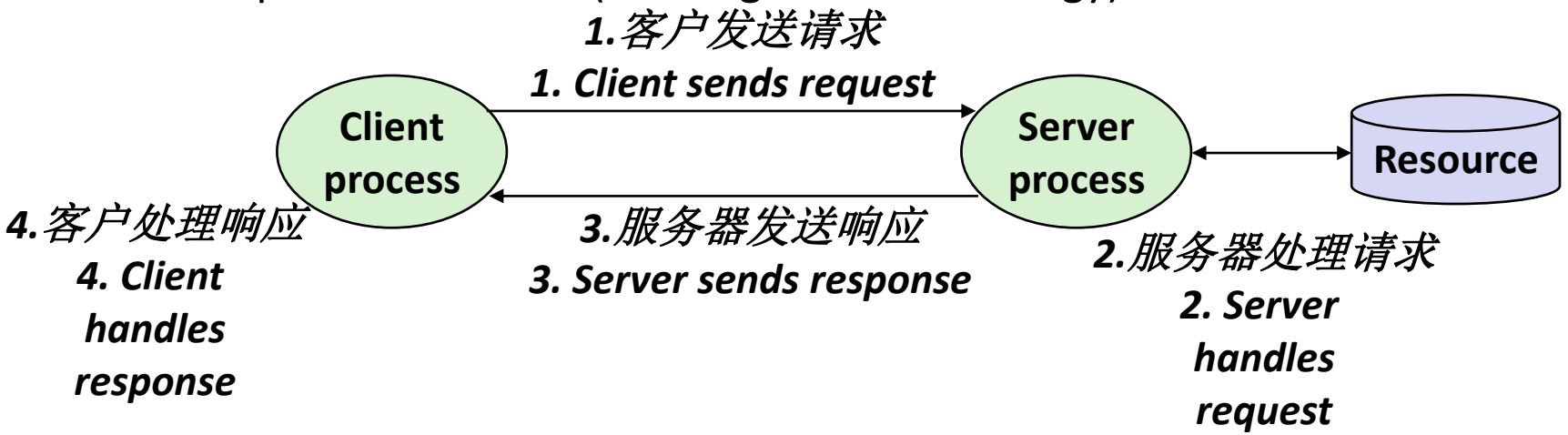
Email link

Cables

Arctic Fibre

客户-服务器事务 A Client-Server Transaction

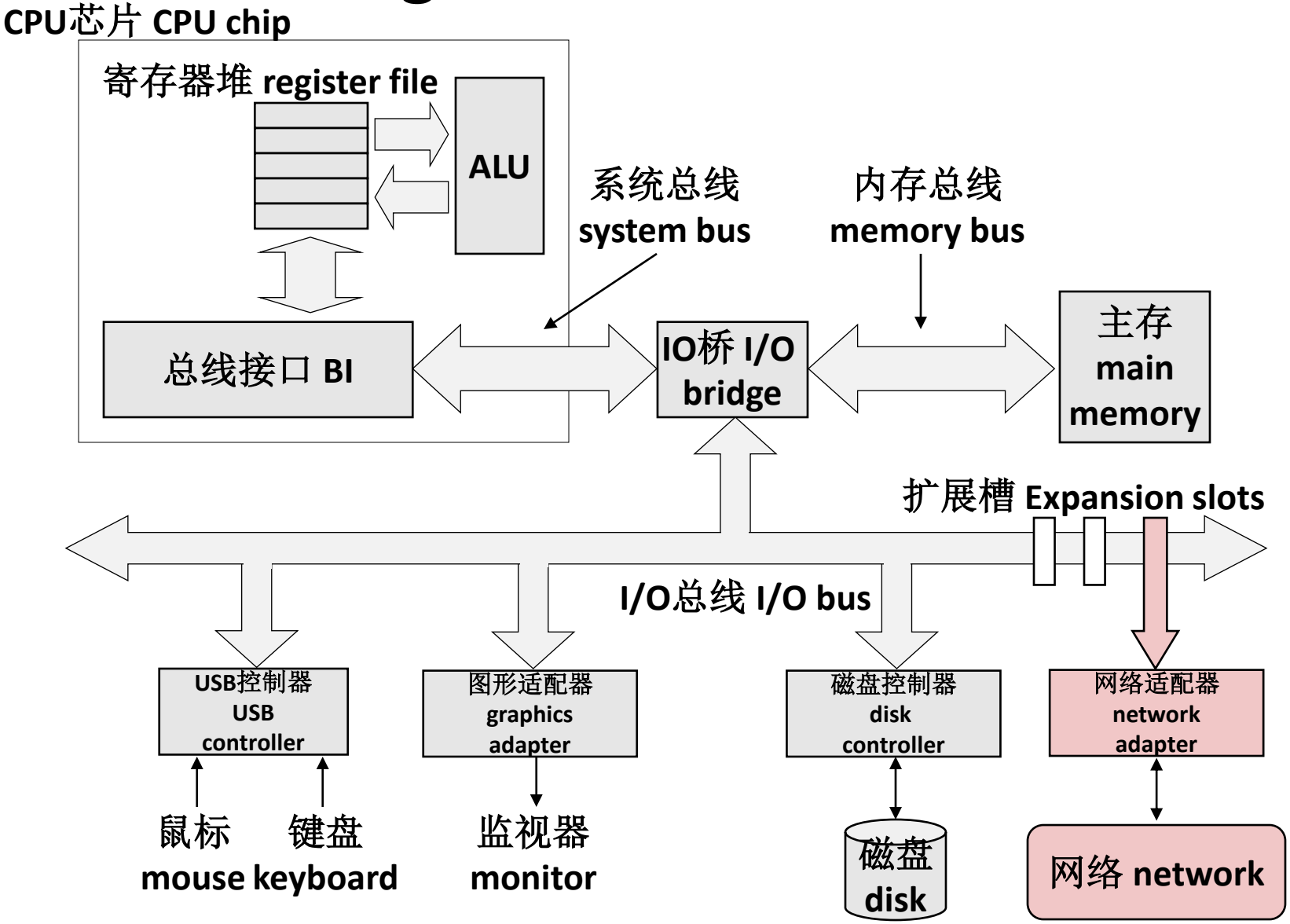
- 多数网络应用基于客户-服务器模型 **Most network applications are based on the client-server model:**
  - 一个服务器进程和一个或多个客户进程 A ***server*** process and one or more ***client*** processes
  - 服务器管理一些资源 Server manages some ***resource***
  - 服务器通过操作资源为客户提供服务 Server provides ***service*** by manipulating resource for clients
  - 服务器由来自客户的请求激活（自动售货机类比） Server activated by request from client (vending machine analogy)



注意：客户和服务器是运行在主机上的进程（可以是相同或不同的主机）
Note: clients and servers are processes running on hosts (can be the same or different hosts)

网络主机的硬件组织

Hardware Organization of a Network Host



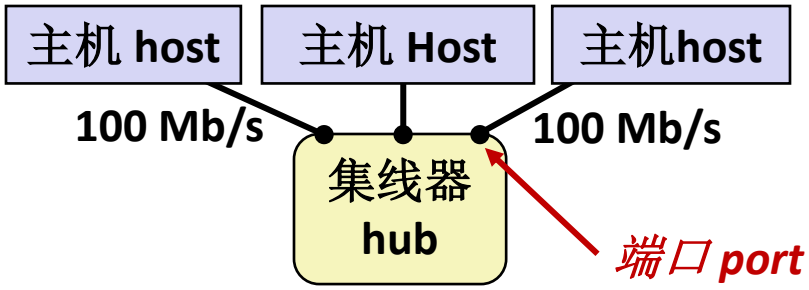
计算机网络
Computer Networks

- 网络是一个不同地理范围的设备和线路组成的分层系统
  A *network* is a hierarchical system of boxes and wires organized by geographical proximity
  - LAN（局域网）跨越建筑物或校园 LAN (Local Area Network) spans a building or campus
    - 以太网是最突出的例子 Ethernet is most prominent example
  - 广域网（WAN）遍布全国或世界 WAN (Wide Area Network) spans country or world
    - 通常是高速点对点（主要是光纤）链路 Typically high-speed point-to-point (mostly optical) links
  - 还有：SAN（存储区域网络）、MAN（城域网）等等 Also: SAN (Storage area network), MAN (Metropolitan), etc., etc.

计算机网络
Computer Networks

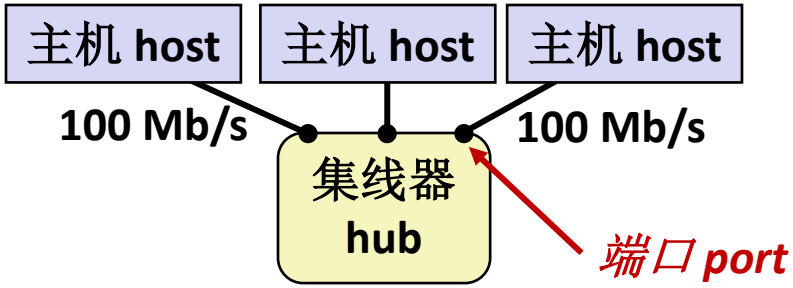
- 互联网络（**internet**）是互连的网络集合 **An *internetwork (internet)* is an interconnected set of networks**
  - 全球IP Internet（大写I）是internet（小写i）最著名例子 The Global IP Internet (uppercase “I”) is the most famous example of an internet (lowercase “i”)
- 让我们看看互联网络是如何从头开始构建的 **Let’s see how an internet is built from the ground up**

老式最低级：以太网网段
Old Lowest Level: Ethernet Segment



- 以太网段由一组通过电缆（双绞线）连接到集线器的主机组成 Ethernet segment consists of a collection of *hosts* connected by wires (twisted pairs) to a *hub*
- 跨越建筑物中的房间或楼层 Spans room or floor in a building

老式最低级：以太网网段
Old Lowest Level: Ethernet Segment

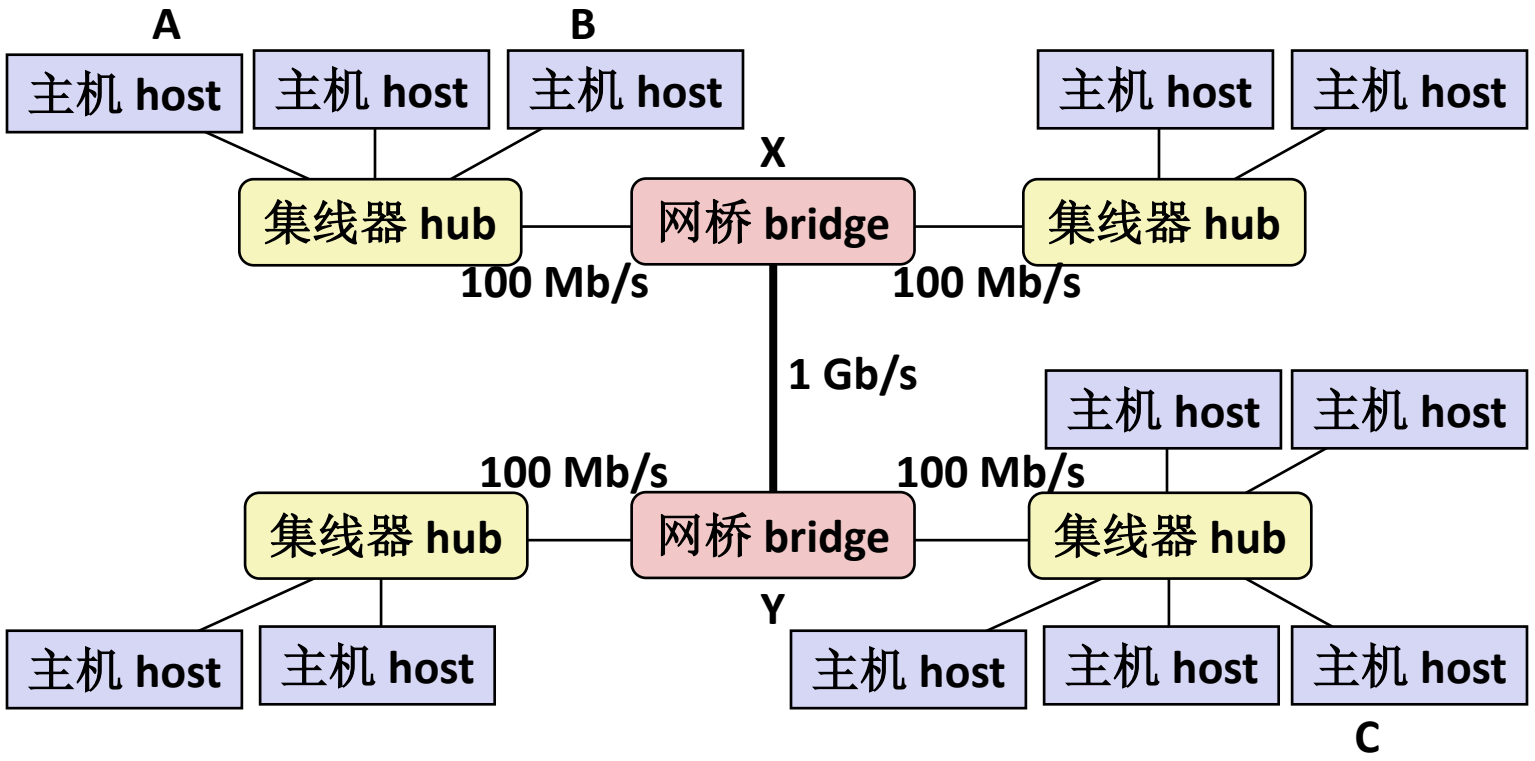


- 运营 Operation
  - 每个以太网适配器都有一个唯一的48位地址（MAC地址） Each Ethernet adapter has a unique 48-bit address (MAC address)
    - 例如，00:16:ea:e3:54:e6　　E.g., 00:16:ea:e3:54:e6
  - 主机以称为**帧**的块向任何其他主机发送比特 Hosts send bits to any other host in chunks called ***frames***
  - 集线器亦步亦趋地将每个位从每个端口复制到每个其他端口 Hub slavishly copies each bit from each port to every other port
    - 每个主机都能看到每一个比特位 Every host sees every bit

[注：集线器已过时。网桥（交换机、路由器）变得足够便宜，可以取代它们]

[Note: Hubs are obsolete. Bridges (switches, routers) became cheap enough to replace them]

下一级：网桥连接的以太网网段
Next Level: Bridged Ethernet Segment



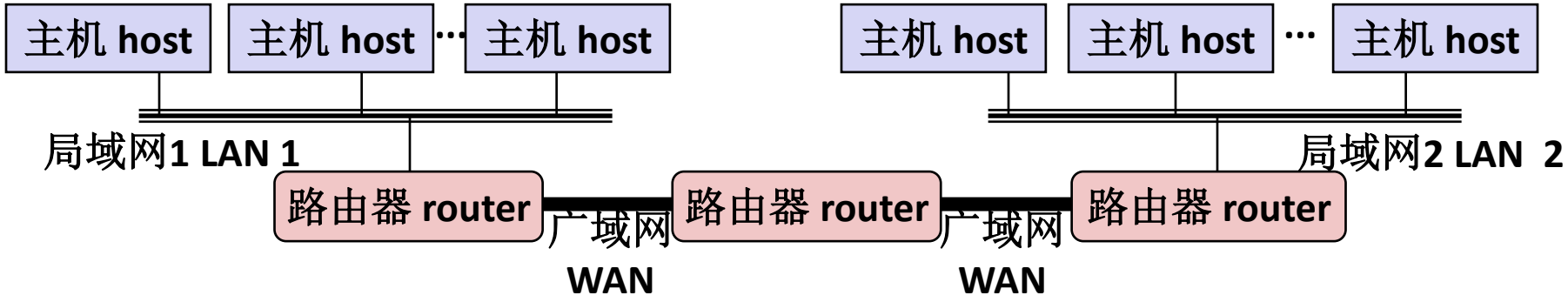
- 跨越建筑物或园区 Spans building or campus
- 网桥聪明地了解哪些主机可以从哪些端口访问，然后有选择地将帧从一个端口复制到另一个端口 Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port

局域网的概念视图
Conceptual View of LANs

- 为了简单起见，集线器、网桥和电缆通常显示为连接到单个电缆的主机集合：**For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:**

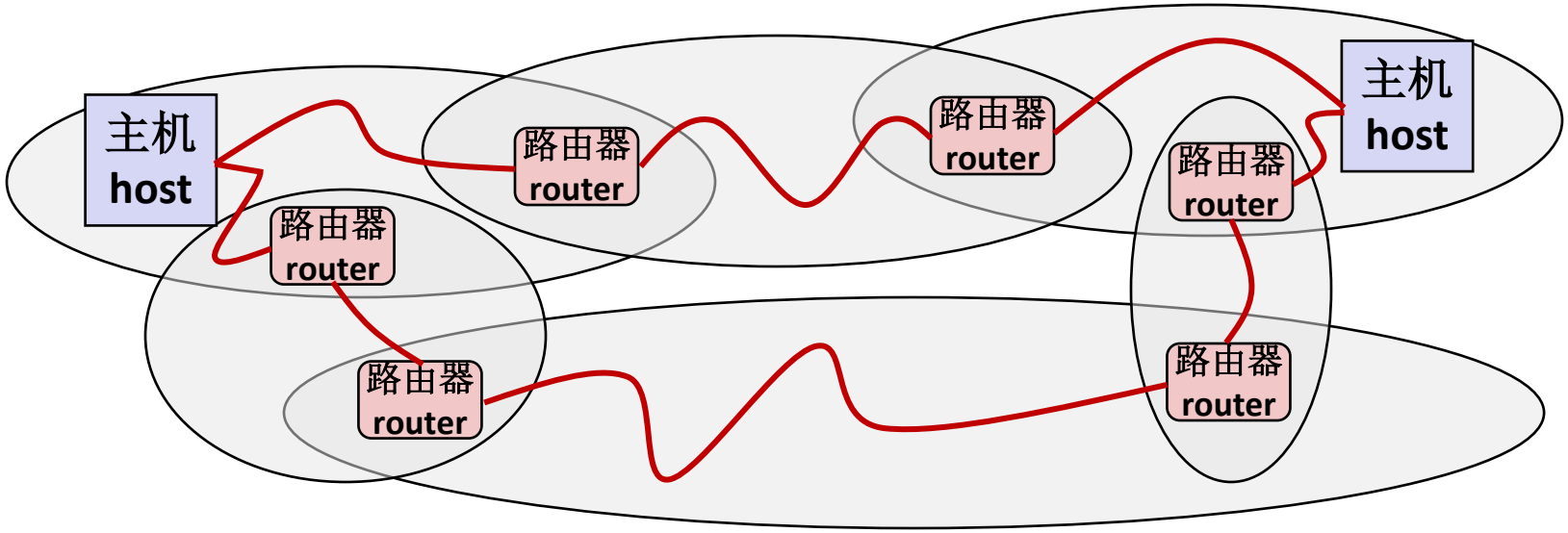
下一级：互联网络 Next Level: internets

- 多个不兼容的局域网可以通过称为路由器的专用计算机进行物理连接 Multiple incompatible LANs can be physically connected by specialized computers called *routers*
- 互连的网络称为互联网络（小写） The connected networks are called an *internet* (lower case)



局域网1和局域网2可能完全不同，完全不兼容
（例如，以太网、光纤通道、802.11、T1链路、DSL等）*
LAN 1 and LAN 2 might be completely different, totally incompatible
(e.g., Ethernet, Fibre Channel, 802.11, T1-links, DSL, …)*

互联网络的逻辑结构
Logical Structure of an internet



- 网络的自组织互连 **Ad hoc interconnection of networks**
  - 无特定拓扑 No particular topology
  - 非常不同的路由器和链路容量 Vastly different router & link capacities
- 通过网络跳步将数据包从源发送到目的地 **Send packets from source to destination by hopping through networks**
  - 路由器形成从一个网络到另一个网络的桥梁 Router forms bridge from one network to another
  - 不同的数据包可能采用不同的路由 Different packets may take different routes

互联网络协议（网际协议）的概念
The Notion of an internet Protocol

- 如何通过不兼容的局域网和广域网发送比特？ **How is it possible to send bits across incompatible LANs and WANs?**

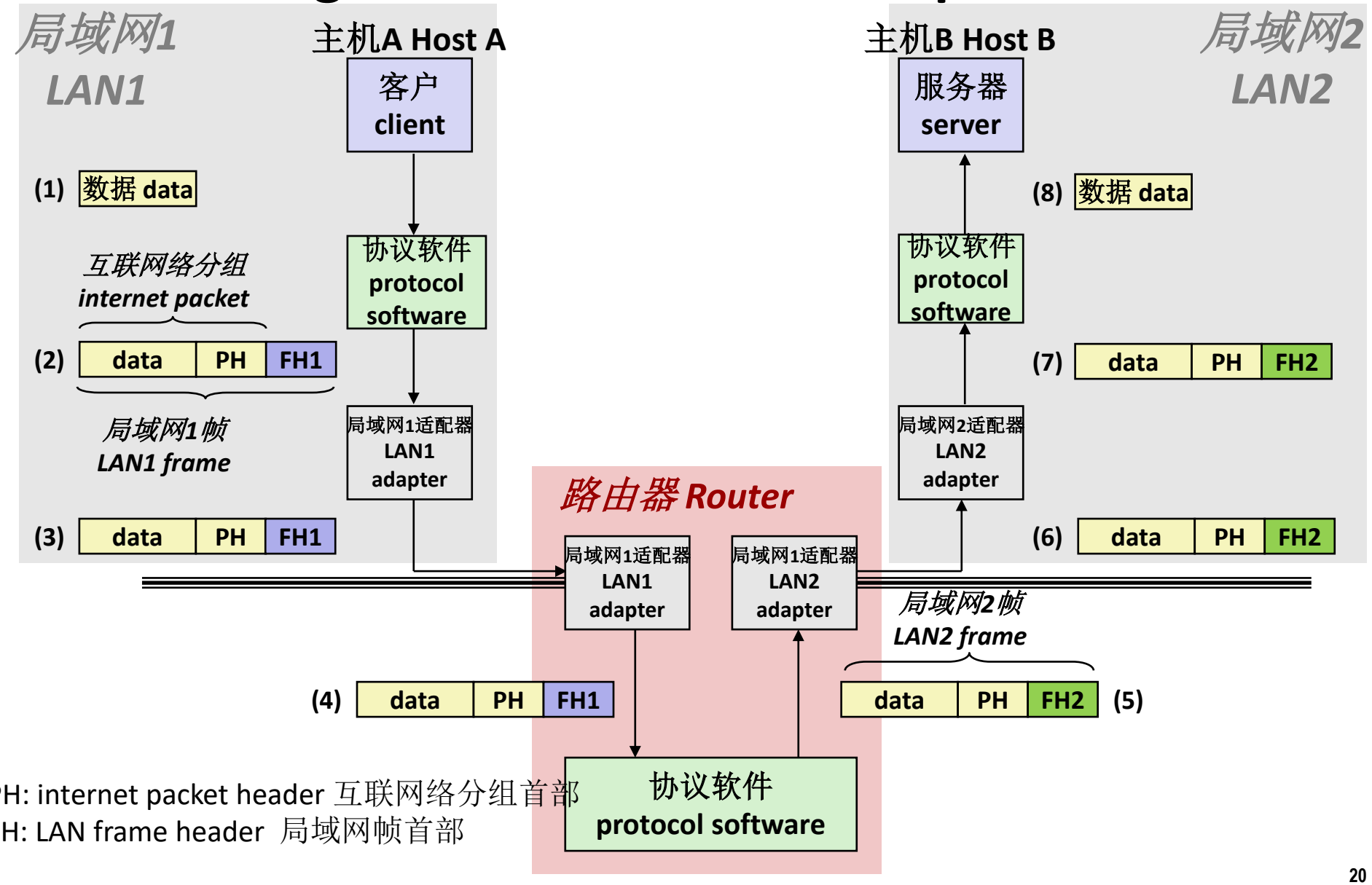
- 解决方案：在每个主机和路由器上运行协议软件 **Solution: *protocol* software running on each host and router**
  - 协议是一组规则，用于控制主机和路由器在网络间传输数据时应如何协作 Protocol is a set of rules that governs how hosts and routers should cooperate when they transfer data from network to network.
  - 消除不同网络之间的差异 Smooths out the differences between the different networks

互联网络协议（网际协议）做什么？
What Does an internet Protocol Do?

- 提供命名方案 **Provides a *naming scheme***
  - 互联网络协议（网际协议）定义了主机地址的统一格式 An internet protocol defines a uniform format for ***host addresses***
  - 为每个主机（和路由器）分配了至少一个唯一标识它的互联网络（网际）地址 Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it
- 提供传递机制 **Provides a *delivery mechanism***
  - 互联网络协议（网际协议）定义了标准传输单元（**分组**） An internet protocol defines a standard transfer unit (***packet***)
  - 分组由分组首部和有效载荷组成 Packet consists of ***header*** and ***payload***
    - 分组首部：包含分组大小、源地址和目标地址等 Header: contains info such as packet size, source and destination addresses
    - 信息有效负载：包含从源主机发送的数据位 Payload: contains data bits sent from source host

通过封装传输互联网络（网际）数据

Transferring internet Data Via Encapsulation



PH: internet packet header 互联网络分组首部

FH: LAN frame header 局域网帧首部

其它问题 Other Issues

- 我们忽略了一些重要问题： **We are glossing over a number of important questions:**
  - 如果不同的网络具有不同的最大帧大小，该怎么办？ (分段) What if different networks have different maximum frame sizes? (segmentation)
  - 路由器如何知道向哪里转发帧？ How do routers know where to forward frames?
  - 当网络拓扑发生变化时，如何通知路由器？ How are routers informed when the network topology changes?
  - 如果分组丢失了怎么办？ What if packets get lost?

- 这些（和其他）问题由*计算机网络*系统解决 **These (and other) questions are addressed by the area of systems known as *computer networking***

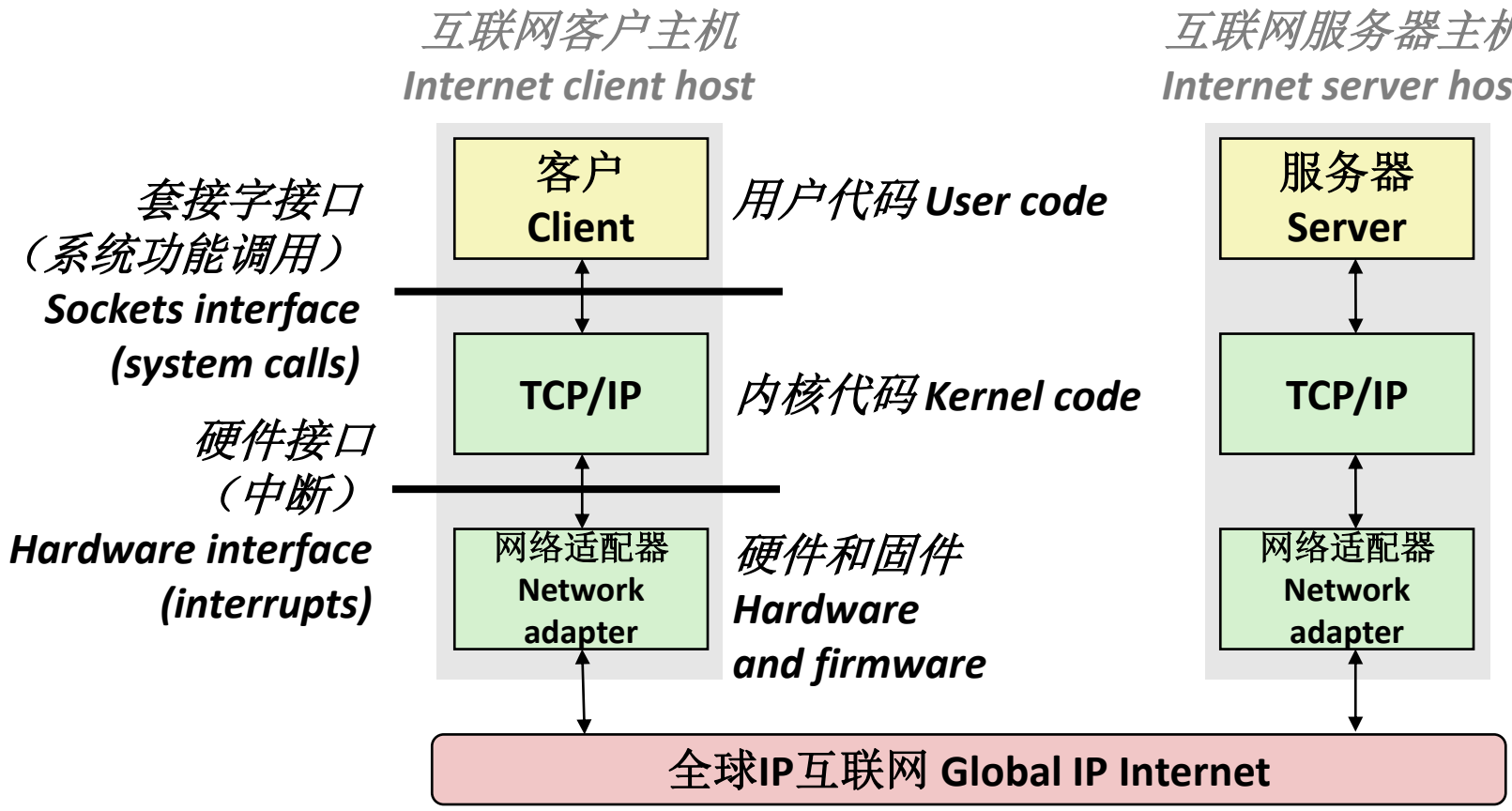
全球IP互联网（大写）
Global IP Internet (upper case)

- 互联网络的最著名例子 **Most famous example of an internet**

- 基于**TCP/IP**协议簇 **Based on the TCP/IP protocol family**
  - IP (网际协议 Internet Protocol)
    - 提供基本命名方案和主机之间不可靠分组（数据报）传输能力 Provides *basic naming scheme* and unreliable *delivery capability* of packets (datagrams) from *host-to-host*
  - UDP (用户数据报协议 User Datagram Protocol)
    - 使用IP提供进程之间不可靠数据报传输 Uses IP to provide *unreliable* datagram delivery from *process-to-process*
  - TCP (传输控制协议 Transmission Control Protocol)
    - 使用IP提供在连接上进程之间可靠的字节流 Uses IP to provide *reliable* byte streams from *process-to-process* over *connections*

- 通过混合**Unix**文件**I/O**和*套接字接口*函数来访问 **Accessed via a mix of Unix file I/O and functions from the *sockets interface***

互联网应用的硬件和软件组织
Hardware and Software Organization of an Internet Application



互联网的程序员视图
A Programmer's View of the Internet

1.主机映射为一组32位*IP地址* 1. Hosts are mapped to a set of 32-bit *IP addresses*

- 128.2.203.179
- 127.0.0.1 (总是localhost主机 always *localhost*)

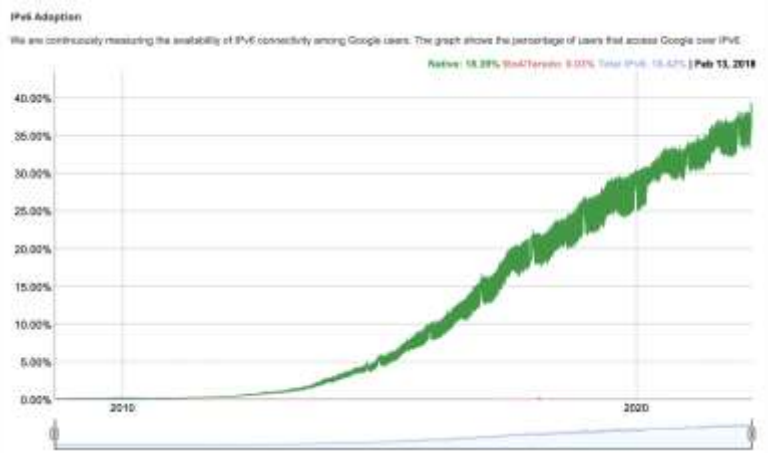
2.为了方便人使用，域名系统将一组称为互联网域名的标识符映射成IP地址 2. As a convenience for humans, the Domain Name System maps a set of identifiers called Internet *domain names* to IP addresses:

- www.cs.cmu.edu "解析成 resolves to" 128.2.217.3

3. 一台互联网主机上的进程可以通过连接和另一台互联网主机上的进程通信 A process on one Internet host can communicate with a process on another Internet host over a *connection*

旁注：IPv4和IPv6 Aside: IPv4 and IPv6

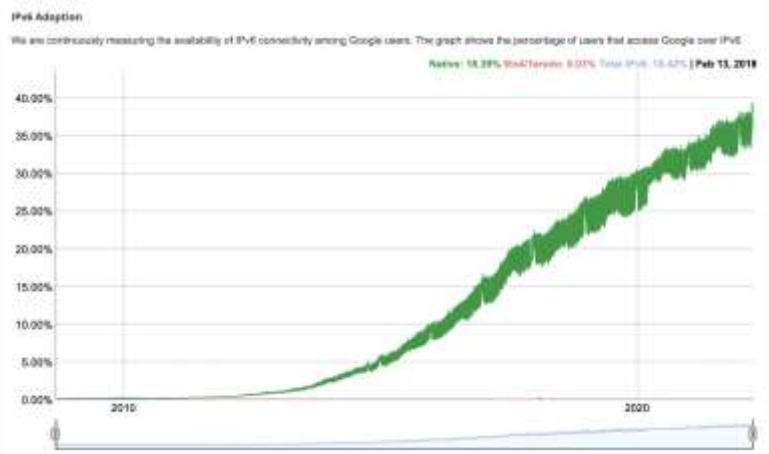
- 最初的互联网协议（32位地址）被称为互联网协议版本4（IPv4） **The original Internet Protocol, with its 32-bit addresses, is known as *Internet Protocol Version 4* (IPv4)**
- **1996年：互联网工程任务组（IETF）引入了具有128位地址的互联网协议版本6（IPv6） 1996: Internet Engineering Task Force (IETF) introduced *Internet Protocol Version 6* (IPv6) with 128-bit addresses**
  - 计划作为IPv4的继任者 Intended as the successor to IPv4



Google公司的IPv6流量
IPv6 traffic at Google

旁注：IPv4和IPv6 Aside: IPv4 and IPv6

- 大部分互联网流量仍由IPv4承载 **Majority of Internet traffic still carried by IPv4**
- 我们将关注IPv4，但将向您展示如何编写独立于协议的网络代码 **We will focus on IPv4, but will show you how to write networking code that is protocol-independent.**



Google公司的IPv6流量
IPv6 traffic at Google

(1) IP地址 IP Addresses

- **32位IP地址存储在IP地址结构中 32-bit IP addresses are stored in an *IP address struct***
  - IP地址始终以网络字节顺序（大端字节顺序）存储在内存中 IP addresses are always stored in memory in *network byte order* (big-endian byte order)
  - 实际上是分组首部中的字段（整数）从一台计算机传输到另一台计算机 True True in general for any integer transferred in a packet header from one machine to another.
    - 例如，用于标识互联网连接的端口号 E.g., the port number used to identify an Internet connection.

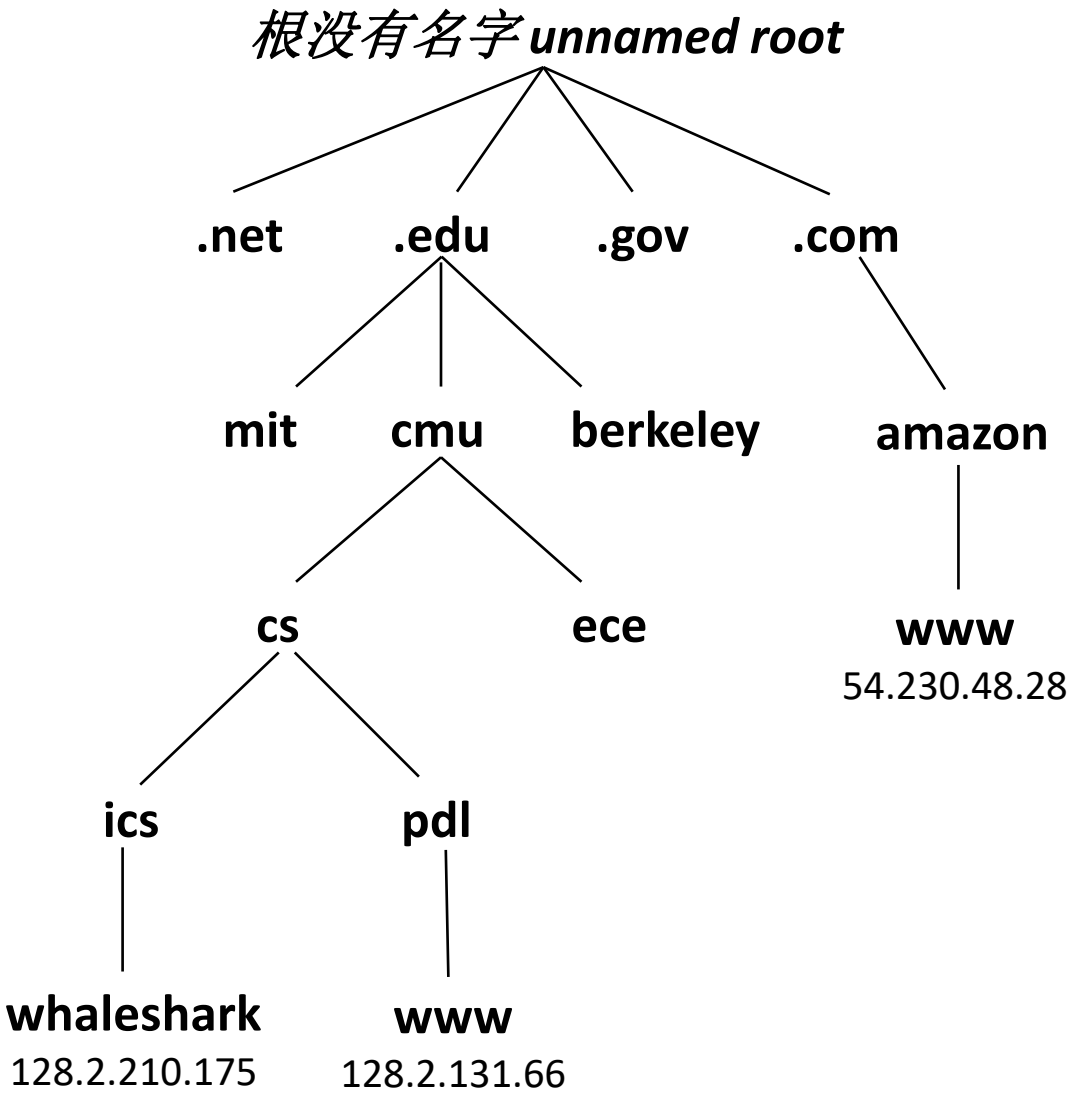
```
/* Internet address structure */
struct in_addr {
    uint32_t  s_addr; /* network byte order (big-endian) */
};
```

点分十进制记法 Dotted Decimal Notation

- 按照惯例，32位IP地址中的每个字节由其十进制值表示，并用句点分隔 **By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period**
  - IP地址： IP address: `0x8002C2F2 = 128.2.194.242`

- 使用**getaddrinfo**和**getnameinfo**函数（稍后介绍）在IP地址和点分十进制格式之间进行转换 **Use `getaddrinfo` and `getnameinfo` functions (described later) to convert between IP addresses and dotted decimal format.**

(2) 互联网域名 Internet Domain Names



第一级域名
First-level domain names

第二级域名
Second-level domain names

第三级域名
Third-level domain names

.space	.store	.stream	.studio
.study	.style	.supplies	.supply
.support	.surf	.surgery	.sydney
.systems	.taipei	.tattoo	.tax
.taxi	.team	.tech	.technology
.tennis	.theater	.theatre	.tienda
.tips	.tires	.tirol	.today
.tokyo	.tools	.top	.tours
.town	.toys	.trade	.trading
.training	.tube	.university	.uno
.vacations	.vegas	.ventures	.versicherung
.vet	.viajes	.video	.villas
.vin	.vip	.vision	.vlaanderen
.vodka	.vote	.voting	.voto
.voyage	.wales	.wang	.watch
.webcam	.website	.wed	.wedding
.whoswho	.wien	.wiki	.win
.wine	.work	.works	.world
.wtf	.在线	.移动	.онлайн
.сайт	.קום	.орг	.中文网
.संगठन	.机构	.みんな	.游戏
.企业	.xyz	.yoga	.yokohama
.zone			

域名系统 Domain Naming System (DNS)

- 互联网在一个巨大的全球分布式数据库***DNS***中维护IP地址和域名之间的映射 **The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called *DNS***

- 从概念上讲，程序员可以将DNS数据库视为数百万个*主机条目*的集合 **Conceptually, programmers can view the DNS database as a collection of millions of *host entries.***
  - 每个主机条目定义一组域名和IP地址之间的映射 Each host entry defines the mapping between a set of domain names and IP addresses.
  - 在数学意义上，主机条目是域名和IP地址的等价类 In a mathematical sense, a host entry is an equivalence class of domain names and IP addresses.

DNS映射的属性 Properties of DNS Mappings

- 使用nslookup可以浏览DNS映射的属性 Can explore properties of DNS mappings using `nslookup`
  - (为简洁起见，编辑了输出 Output edited for brevity)

- 每个主机都有一个本地定义的域名localhost，它始终映射为回环地址127.0.0.1 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
```

- 使用hostname确定本地主机的真实域名 Use `hostname` to determine real domain name of local host:

```
linux> hostname
whaleshark.ics.cs.cmu.edu
```

DNS映射的属性 （续）
Properties of DNS Mappings (cont)

- 简单情况：域名和IP地址之间一对一的映射 **Simple case: one-to-one mapping between domain name and IP address:**

```
linux> nslookup whaleshark.ics.cs.cmu.edu
Address: 128.2.210.175
```

- 多个域名映射到同样的IP地址 **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
```

- 以及逆向解析 **And backwards:**

```
linux> nslookup 18.25.0.23
23.0.25.18.in-addr.arpa    name = eecs.mit.edu.
```

DNS映射的属性 （续）
Properties of DNS Mappings (cont)

- 多个域名映射到多个IP地址 Multiple domain names mapped to multiple IP addresses:

```
linux> nslookup www.twitter.com
Address: 104.244.42.65
Address: 104.244.42.129
Address: 104.244.42.193
Address: 104.244.42.1

linux> nslookup twitter.com
Address: 104.244.42.129
Address: 104.244.42.65
Address: 104.244.42.193
Address: 104.244.42.1
```

- 有些合法域名没有映射到任何IP地址 Some valid domain names don’t map to any IP address:

```
linux> nslookup ics.cs.cmu.edu
(No Address given)
```

(3) 互联网连接 Internet Connections

- 客户端和服务器通常通过TCP*连接*发送字节流进行通信。每个连接是： **Clients and servers most often communicate by sending streams of bytes over TCP *connections*. Each connection is:**
  - *点对点*：连接一对进程 *Point-to-point*: connects a pair of processes.
  - *全双工*：数据可以同时在两个方向上流动 *Full-duplex*: data can flow in both directions at the same time,
  - *可靠*：源发送的字节流最终会按照发送的顺序被目标接收。*Reliable*: stream of bytes sent by the source is eventually received by the destination in the same order it was sent.

- *套接字*是连接的端点 ***A socket* is an endpoint of a connection**
  - *套接字地址*是IP地址：端口对 *Socket address* is an `IPaddress:port` pair

(3) 互联网连接 Internet Connections

- *端口*是一个16位整数，用于标识进程： **A *port* is a 16-bit integer that identifies a process:**
  - *临时端口*: 当客户端发出连接请求时，由客户端内核自动分配。 ***Ephemeral port*: Assigned automatically by client kernel when client makes a connection request.**
  - *熟知端口*: 与服务器提供的某些*服务*相关（例如，端口80与Web服务器相关） ***Well-known port:* Associated with some *service* provided by a server (e.g., port 80 is associated with Web servers)**

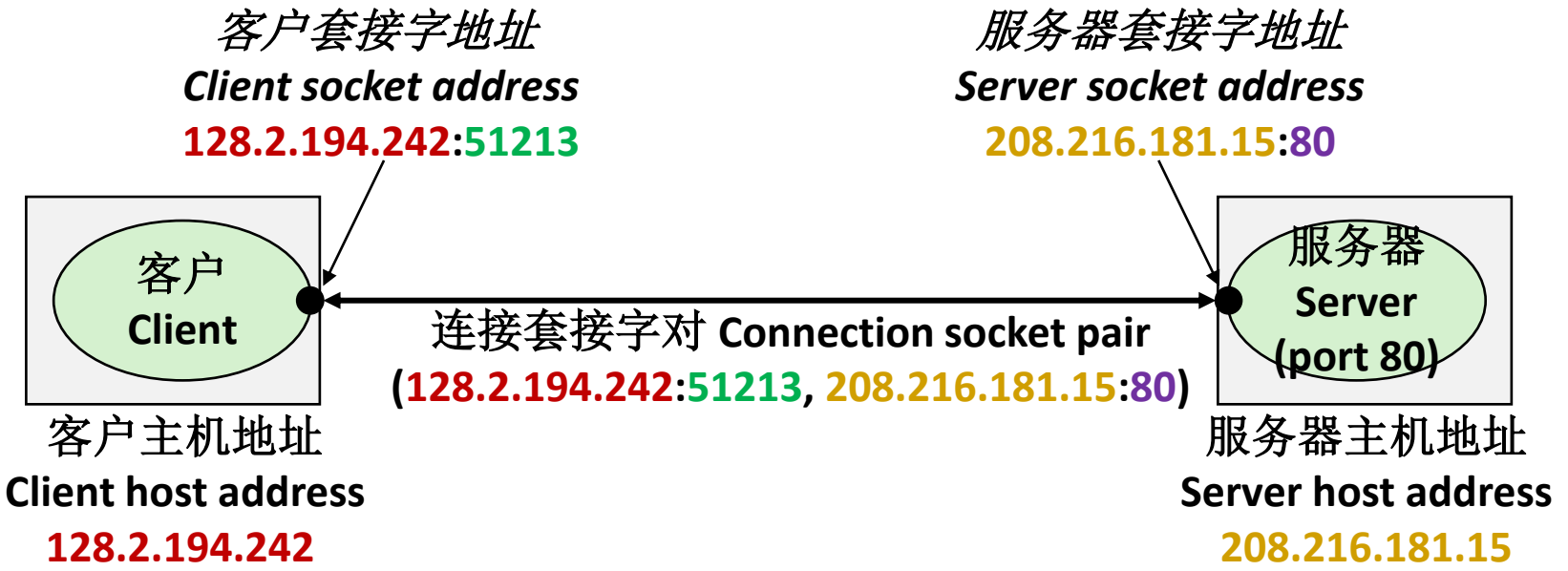
熟知的服务名称和端口
Well-known Service Names and Ports

- 流行服务已永久分配了*熟知端口*和相应的*熟知服务名称*:
  Popular services have permanently assigned *well-known ports* *and* corresponding *well-known service names*:
  - echo服务器： echo servers: echo 7
  - ftp服务器：ftp servers: ftp 21
  - ssh服务器：ssh servers: ssh 22
  - 电子邮件服务器： email servers: smtp 25
  - 未加密的Web服务器： Unencrypted Web servers: http 80
  - SSL/TLS加密Web：SSL/TLS encrypted Web: https 443

- 熟知端口和服务名称之间的映射包含在每个Linux机器上的/etc/services文件中 **Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.**

连接的剖析 Anatomy of a Connection

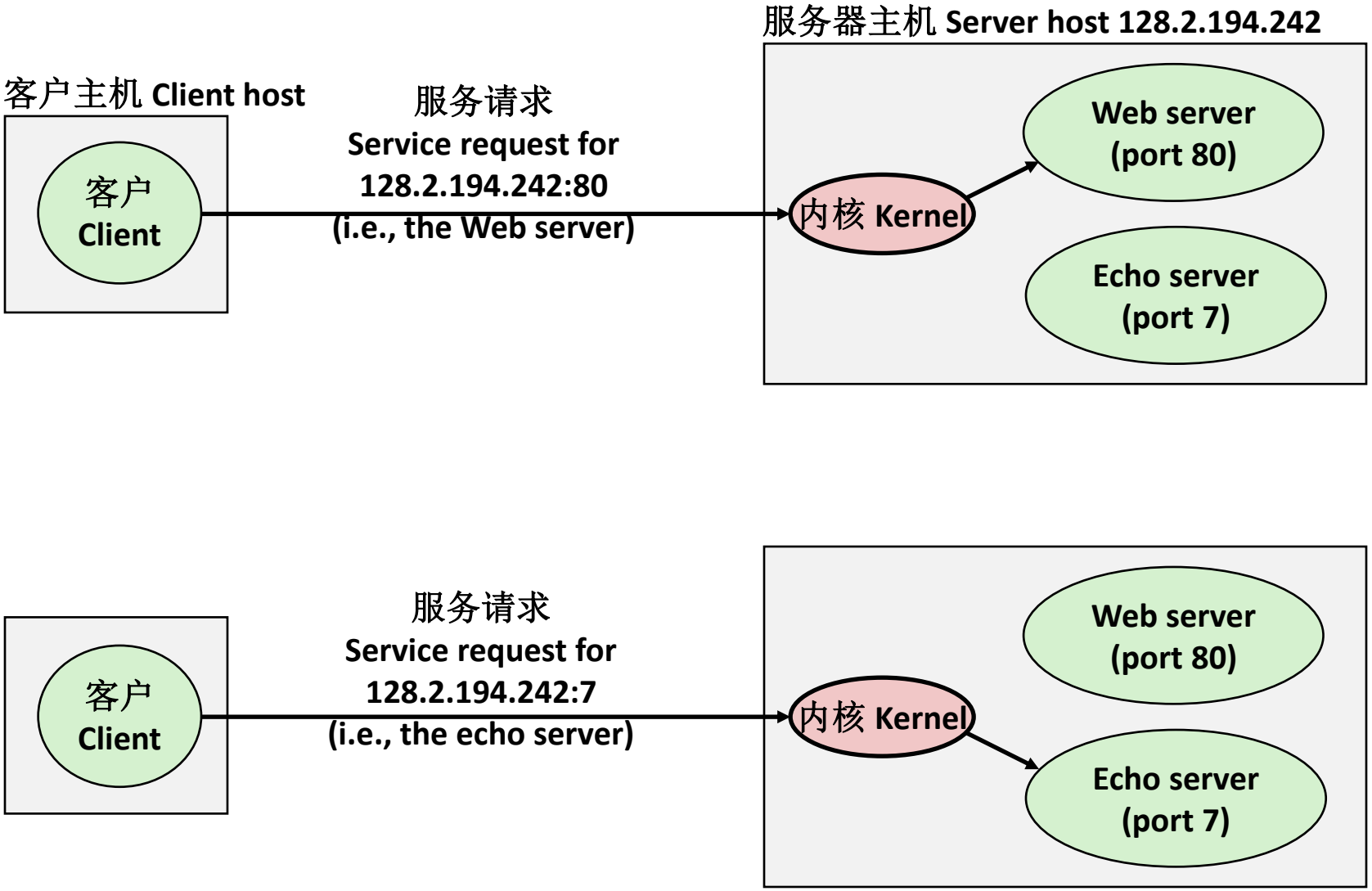
- 连接由其端点（套接字对）的套接字地址唯一标识 A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - `(cliaddr:cliport, servaddr:servport)`



51213是由内核分配的临时端口
51213 is an ephemeral port allocated by the kernel

80是与Web服务器关联的熟知端口
80 is a well-known port associated with Web servers

使用端口标识服务
Using Ports to Identify Services



套接字接口 Sockets Interface

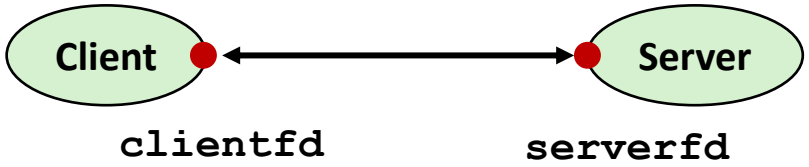
- 与Unix I/O结合使用的一组系统级函数，用于构建网络应用程序 **Set of system-level functions used in conjunction with Unix I/O to build network applications.**

- 创建于80年代早期，是最初的Berkeley Unix发行版的一部分，其中包含早期版本的互联网协议 **Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.**

- 适用于所有现代系统 **Available on all modern systems**
  - Unix variants变体, Windows, OS X, IOS, Android, ARM

套接字 Sockets

- 什么是套接字？ **What is a socket?**
  - 对于内核来说，套接字是通信的端点 To the kernel, a socket is an endpoint of communication
  - 对于应用程序，套接字是一个文件描述符，它允许应用程序从网络读取/向网络写入 To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - 使用FD抽象可以重用代码和接口 Using the FD abstraction lets you reuse code & interfaces
- 客户端和服务器通过读取和写入套接字描述符来相互通信 **Clients and servers communicate with each other by reading from and writing to socket descriptors**



- 常规文件I/O和套接字I/O之间的主要区别是应用程序如何“打开”套接字描述符 **The main distinction between regular file I/O and socket I/O is how the application “opens” the socket descriptors**

套接字编程示例 Socket Programming Example

- 回声服务器和客户 **Echo server and client**
- 服务器 **Server**
  - 接受连接请求 Accepts connection request
  - 重复发送回输入行 Repeats back lines as they are typed
- 客户 **Client**
  - 请求连接到服务器 Requests connection to server
  - 重复 Repeatedly:
    - 从终端读一行 Read line from terminal
    - 发送给服务器 Send to server
    - 从服务器读响应 Read reply from server
    - 打印行到终端 Print line to terminal

回声服务器/客户会话示例
Echo Server/Client Session Example

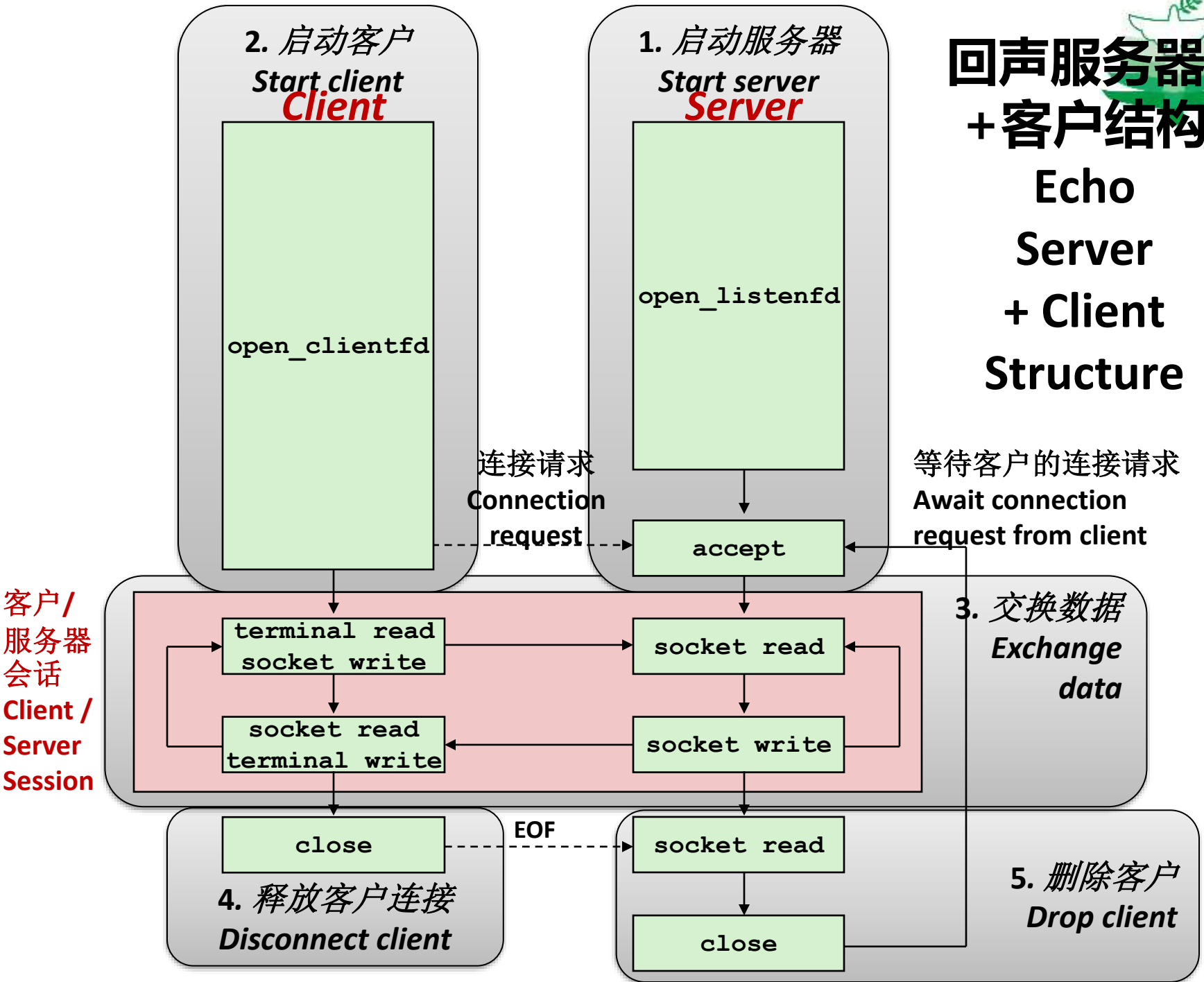
客户 Client

```
bambooshark: ./echoclient whaleshark.ics.cs.cmu.edu 6616        (A)
This line is being echoed                                       (B)
This line is being echoed
This one is, too                                                (C)
This one is, too
^D
bambooshark: ./echoclient whaleshark.ics.cs.cmu.edu 6616        (D)
This one is a new connection                                    (E)
This one is a new connection
^D
```

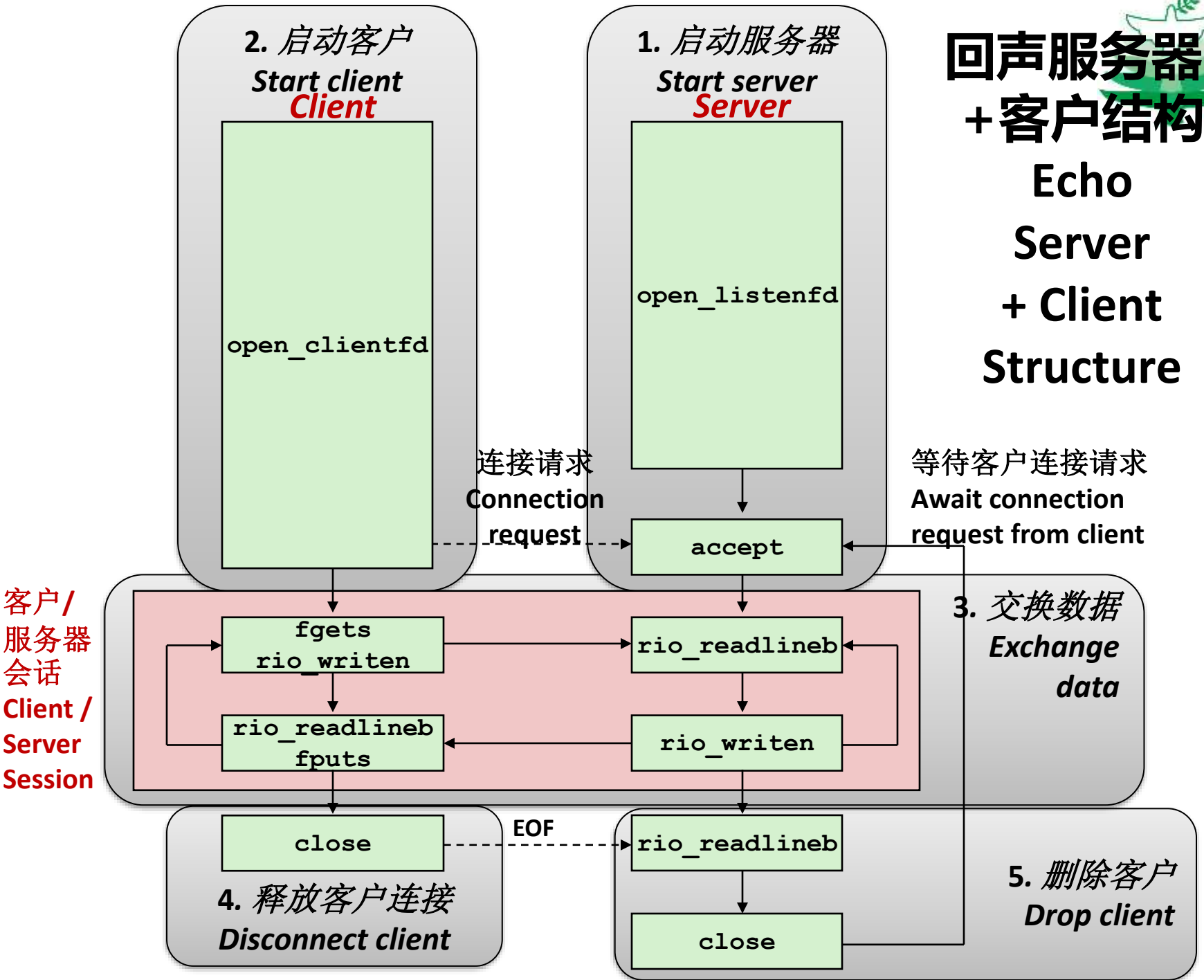
服务器 Server

```
whaleshark: ./echoserveri 6616
Connected to (BAMBOOSHARK.ICS.CS.CMU.EDU, 33707)        (A)
server received 26 bytes                                (B)
server received 17 bytes                                (C)
Connected to (BAMBOOSHARK.ICS.CS.CMU.EDU, 33708)        (D)
server received 29 bytes                                (E)
```

回声服务器+客户结构
Echo Server + Client Structure



回声服务器+客户结构
Echo Server + Client Structure



回忆：无缓冲RIO输入/输出
Recall: Unbuffered RIO Input/Output

- 与Unix读写接口相同 Same interface as Unix `read` and `write`
- 特别适用于在网络套接字上传输数据 Especially useful for transferring data on network sockets

```
#include "csapp.h"

ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

Return: num. bytes transferred if OK, 0 on EOF (`rio_readn` only), -1 on error

- **`rio_readn`**仅在遇到EOF时返回不足值 **`rio_readn`** returns short count only if it encounters EOF
  - 仅当您知道要读取多少字节时才使用它 Only use it when you know how many bytes to read
- **`rio_writen`**从不返回不足值 **`rio_writen`** never returns a short count
- 对**`rio_readn`**和**`rio_writen`**的调用可以在同一描述符上任意交错 Calls to **`rio_readn`** and **`rio_writen`** can be interleaved arbitrarily on the same descriptor

回忆：带缓冲RIO输入/输出
Recall: Buffered RIO Input Functions

- 高效地从部分缓存在内部内存缓冲区中的文件中读取文本行和二进制数据 **Efficiently read text lines and binary data from a file partially cached in an internal memory buffer**

```
#include "csapp.h"

void rio_readinitb(rio_t *rp, int fd);

ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
```

Return: num. bytes read if OK, 0 on EOF, -1 on error

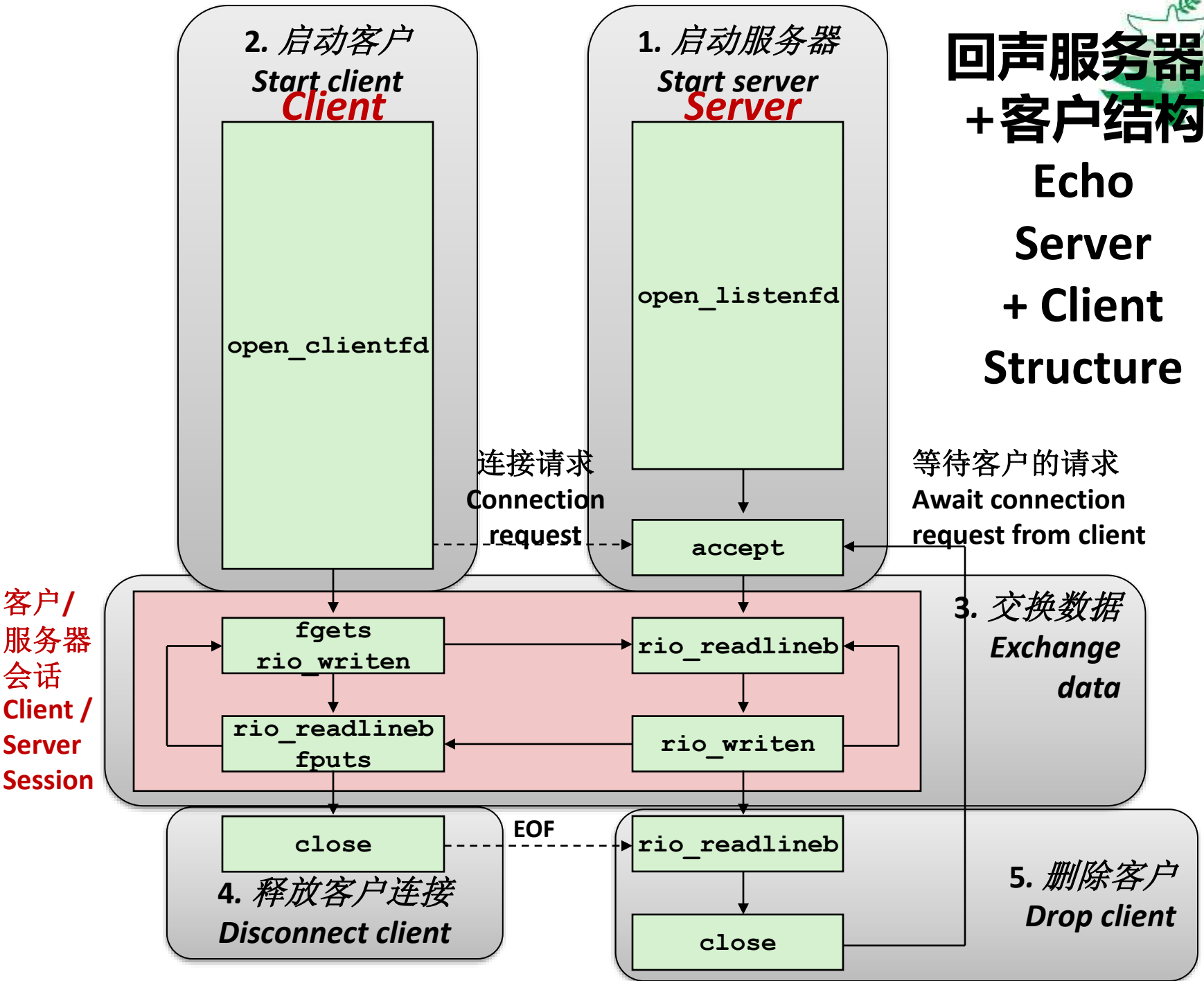
  - **rio_readlineb**从文件fd读最多**maxlen**字节文本行，并存储该行到**usrbuf** **rio_readlineb** reads a ***text line*** of up to **maxlen** bytes from file **fd** and stores the line in **usrbuf**
    - 特别适合从网络套接字读文本行 Especially useful for reading text lines from network sockets
  - 停止条件 Stopping conditions
    - 读取了最大字节 **maxlen** bytes read
    - 遇到文件结束符 EOF encountered
    - 遇到换行符（'\n'） Newline ('\n') encountered

回声客户：主例程 Echo Client: Main Routine

```c
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd;
    char *host, *port, buf[MAXLINE];
    rio_t rio;

    host = argv[1];
    port = argv[2];

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

echoclient.c

回声服务器+客户结构
Echo Server + Client Structure



迭代回声服务器：主例程
Iterative Echo Server: Main Routine

```
#include "csapp.h"
void echo(int connfd);

int main(int argc, char **argv)
{
    int listenfd, connfd;
    socklen_t clientlen;
    struct sockaddr_storage clientaddr; /* Enough room for any addr */
    char client_hostname[MAXLINE], client_port[MAXLINE];

    listenfd = Open_listenfd(argv[1]);
    while (1) {
        clientlen = sizeof(struct sockaddr_storage); /* Important! */
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
        Getnameinfo((SA *) &clientaddr, clientlen,
                    client_hostname, MAXLINE, client_port, MAXLINE, 0);
        printf("Connected to (%s, %s)\n", client_hostname, client_port);
        echo(connfd);
        Close(connfd);
    }
    exit(0);
}
```

echoserveri.c

回声服务器：echo函数
Echo Server: echo function

- 服务器使用RIO读取和回显文本行，直到遇到EOF（文件结束）条件 **The server uses RIO to read and echo text lines until EOF (end-of-file) condition is encountered.**
  - 客户端调用close(clientfd)导致的EOF条件 EOF condition caused by client calling `close(clientfd)`

```
void echo(int connfd)
{
    size_t n;
    char buf[MAXLINE];
    rio_t rio;

    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
        printf("server received %d bytes\n", (int)n);
        Rio_writen(connfd, buf, n);
    }
}
```

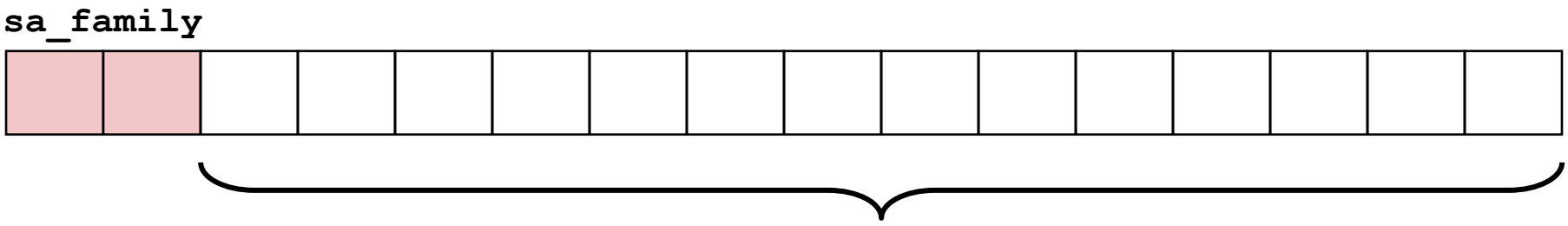
echo.c

套接字地址结构 Socket Address Structures

- 通用套接字地址 **Generic socket address:**
  - 函数的地址参数 For address arguments to **`connect`**, **`bind`**, and **`accept`** *(next lecture)*
  - 之所以如此仅仅是因为在设计套接字接口时，C没有泛型（void*）指针 Necessary only because C did not have generic (**`void *`**) pointers when the sockets interface was designed
  - 为了强制转换方便，我们采用Stevens惯例： For casting convenience, we adopt the Stevens convention:

```
typedef struct sockaddr SA;
```

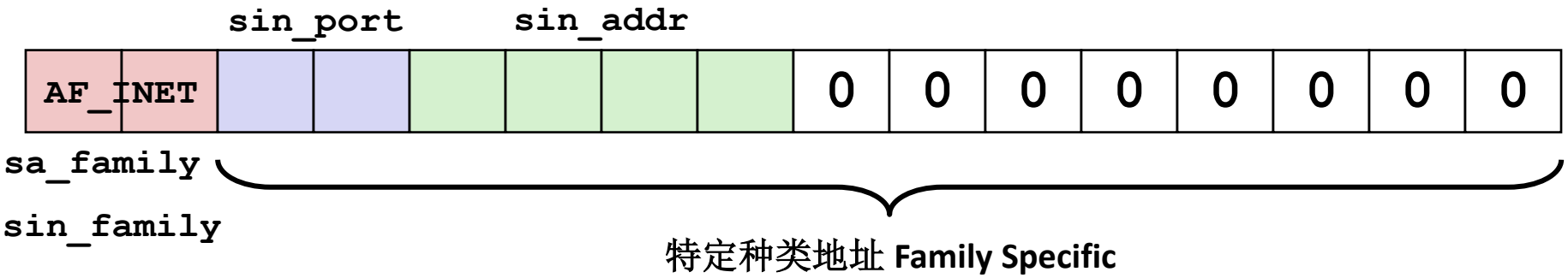
```
struct sockaddr {
  uint16_t  sa_family;    /* Protocol family */
  char      sa_data[14];  /* Address data  */
};
```



套接字地址结构 Socket Address Structures

- 互联网（IPv4）特定套接字地址 **Internet (IPv4) specific socket address:**
  - 必须强制转换，在函数使用套接字地址参数的时候 Must cast (`struct sockaddr_in *`) to (`struct sockaddr *`) for functions that take socket address arguments.

```
struct sockaddr_in  {
  uint16_t        sin_family;  /* Protocol family (always AF_INET) */
  uint16_t        sin_port;    /* Port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```



主机和服务转换：getaddrinfo
Host and Service Conversion: getaddrinfo

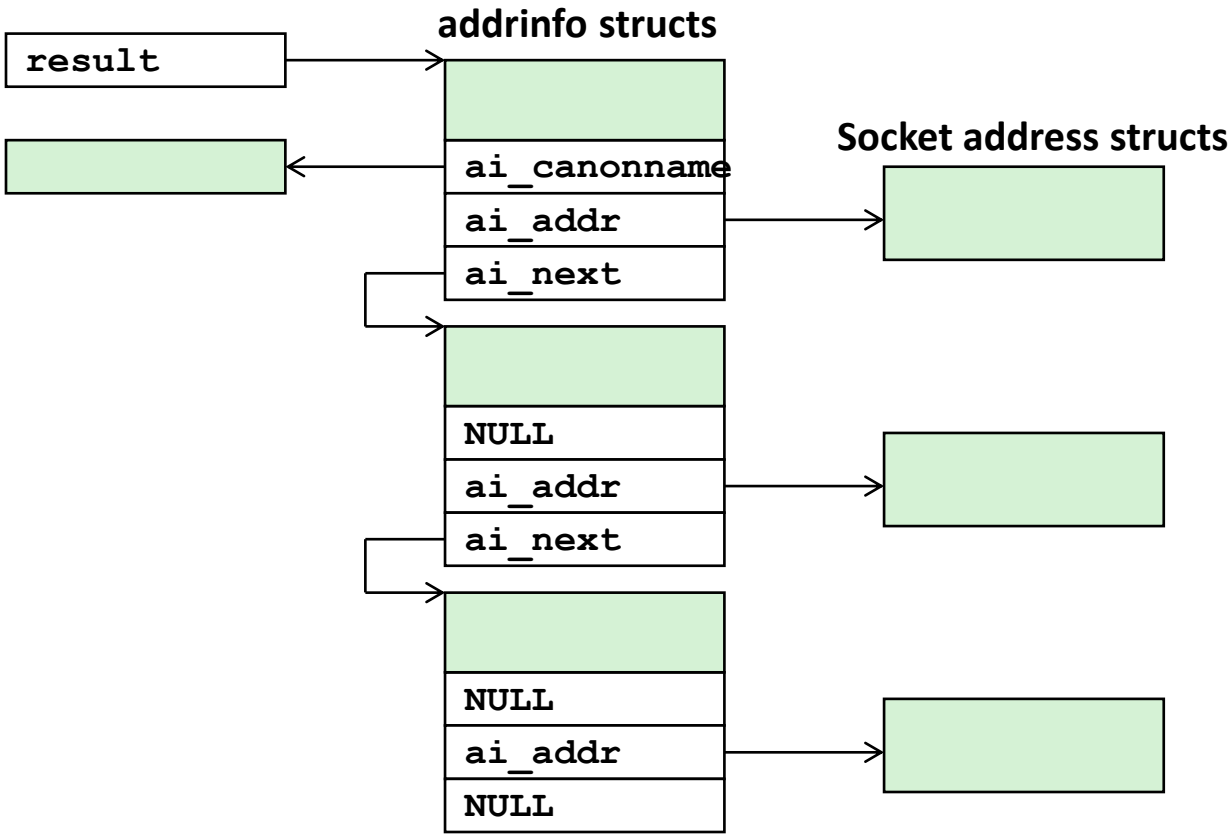
- getaddrinfo是将主机名、主机地址、端口和服务名的字符串表示转换为套接字地址结构的现代方法 **getaddrinfo is the modern way to convert string representations of hostnames, host addresses, ports, and service names to socket address structures.**
  - 替换过时的gethostbyname和getservbyname函数 Replaces obsolete gethostbyname and getservbyname funcs.
- 优势：**Advantages:**
  - 重入（可由线程程序安全使用） Reentrant (can be safely used by threaded programs).
  - 允许我们编写可移植的协议无关代码 Allows us to write portable protocol-independent code
    - 同时适用于IPv4和IPv6 Works with both IPv4 and IPv6
- 缺点 **Disadvantages**
  - 有点复杂 Somewhat complex
  - 幸运的是，在大多数情况下，少量的使用模式就足够了 Fortunately, a small number of usage patterns suffice in most cases.

主机和服务转换：getaddrinfo
Host and Service Conversion: getaddrinfo

```
int getaddrinfo(const char *host,              /* Hostname or address */
                const char *service,           /* Port or service name */
                const struct addrinfo *hints,/* Input parameters */
                struct addrinfo **result);    /* Output linked list */

void freeaddrinfo(struct addrinfo *result);   /* Free linked list */

const char *gai_strerror(int errcode);        /* Return error msg */
```

- 给定主机和服务，getaddrinfo返回结果为指向addrinfo结构的链表，每个addrinfo结构指向对应的套接字地址结构，并且包含套接字接口函数的参数 Given `host` and `service`, `getaddrinfo` returns `result` that points to a linked list of `addrinfo` structs, each of which points to a corresponding socket address struct, and which contains arguments for the sockets interface functions.
- 助手程序函数： Helper functions:
  - `freeadderinfo` frees the entire linked list. 释放整个链表
  - `gai_strerror` converts error code to an error message.将错误代码转换为错误消息

getaddrinfo返回的链表
Linked List Returned by `getaddrinfo`



addrinfo结构 addrinfo Struct

```
struct addrinfo {
    int              ai_flags;      /* Hints argument flags */
    int              ai_family;     /* First arg to socket function */
    int              ai_socktype;   /* Second arg to socket function */
    int              ai_protocol;   /* Third arg to socket function  */
    char            *ai_canonname;  /* Canonical host name */
    size_t           ai_addrlen;    /* Size of ai_addr struct */
    struct sockaddr *ai_addr;       /* Ptr to socket address structure */
    struct addrinfo *ai_next;       /* Ptr to next item in linked list */
};
```

- 由getaddrinfo返回的每个addrinfo结构包含可以直接传递给套接字函数的参数 Each addrinfo struct returned by getaddrinfo contains arguments that can be passed directly to `socket` function.
- 它也会指向一个套接字地址结构，该结构可以直接传递给connect和bind函数 Also points to a socket address struct that can be passed directly to `connect` and `bind` functions .

(`socket`, `connect`, `bind` to be discussed next lecture下次课讨论这些函数)

主机和服务转换：getnameinfo
Host and Service Conversion: getnameinfo

- getnameinfo的功能与getaddrinfo的功能相反，它将一个套接字地址转换成对应的主机和服务 **getnameinfo is the inverse of getaddrinfo, converting a socket address to the corresponding host and service.**
  - 替代过时的gethostbyaddr和getservbyport函数 Replaces obsolete `gethostbyaddr` and `getservbyport` funcs.
  - 可重入和协议无关的 Reentrant and protocol independent.

```
int getnameinfo(const SA *sa, socklen_t salen, /* In: socket addr */
                char *host, size_t hostlen,    /* Out: host */
                char *serv, size_t servlen,    /* Out: service */
                int flags);                    /* optional flags */
```

转换示例 Conversion Example

```c
#include "csapp.h"

int main(int argc, char **argv)
{
    struct addrinfo *p, *listp, hints;
    char buf[MAXLINE];
    int rc, flags;

    /* Get a list of addrinfo records */
    memset(&hints, 0, sizeof(struct addrinfo));
   // hints.ai_family = AF_INET;       /* IPv4 only */
    hints.ai_socktype = SOCK_STREAM; /* Connections only */
    if ((rc = getaddrinfo(argv[1], NULL, &hints, &listp)) != 0) {
        fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(rc));
        exit(1);
    }
```

hostinfo.c

转换示例(续) Conversion Example (cont)

```
    /* Walk the list and display each IP address */
    flags = NI_NUMERICHOST; /* Display address instead of name */
    for (p = listp; p; p = p->ai_next) {
        Getnameinfo(p->ai_addr, p->ai_addrlen,
                    buf, MAXLINE, NULL, 0, flags);
        printf("%s\n", buf);
    }

    /* Clean up */
    Freeaddrinfo(listp);

    exit(0);
}
```

hostinfo.c

运行hostinfo Running hostinfo

```
whaleshark> ./hostinfo localhost
127.0.0.1

whaleshark> ./hostinfo whaleshark.ics.cs.cmu.edu
128.2.210.175

whaleshark> ./hostinfo twitter.com
199.16.156.230
199.16.156.38
199.16.156.102
199.16.156.198

whaleshark> ./hostinfo google.com
172.217.15.110
2607:f8b0:4004:802::200e
```

下次课 Next time

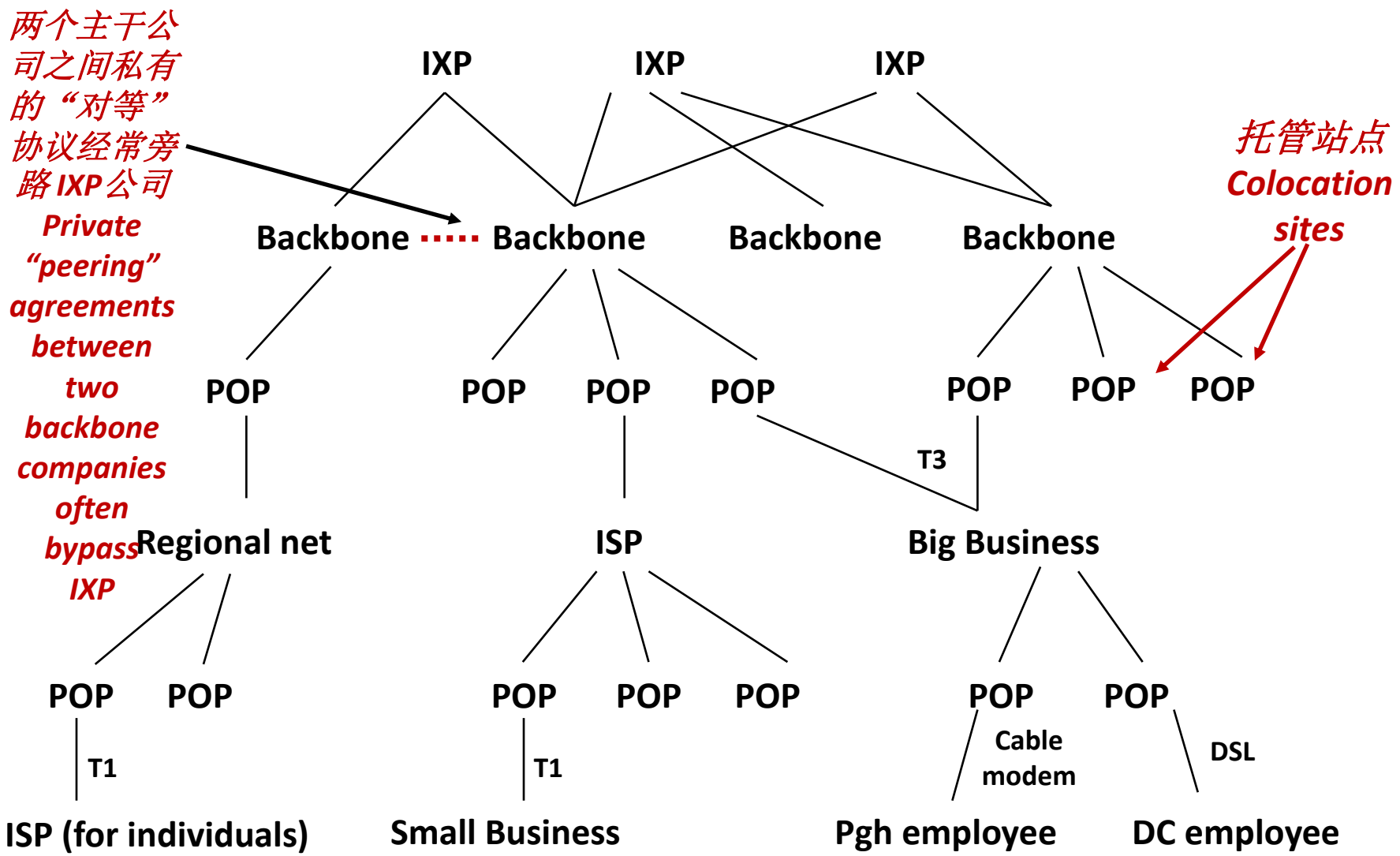
- 使用getaddrinfo实现主机和服务转换 Using `getaddrinfo` for host and service conversion
- 编写客户和服务器 Writing clients and servers
- 编写Web服务器！ Writing Web servers!

附加的幻灯片 Additional slides

基本互联网组件 Basic Internet Components

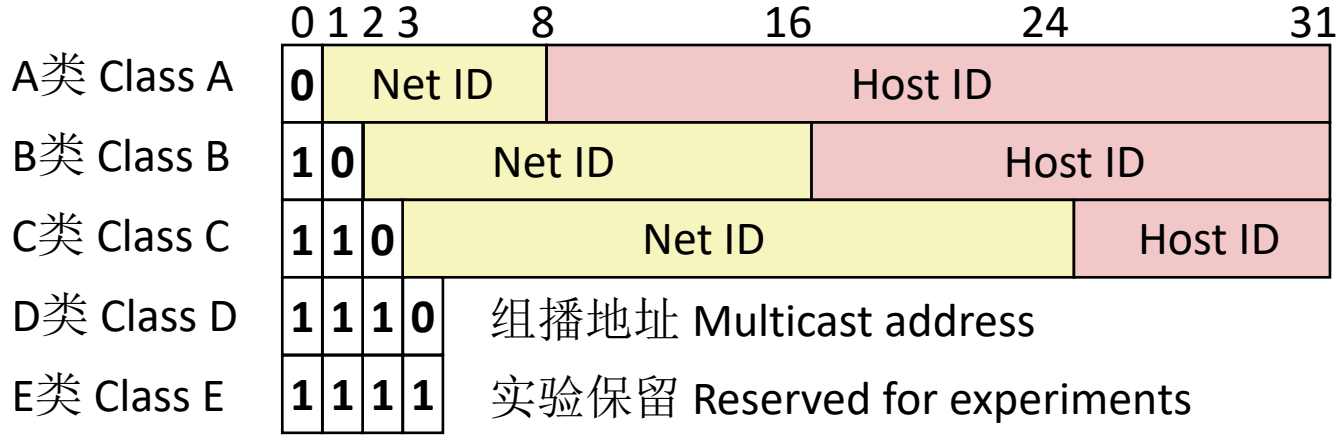
- 互联网主干 **Internet backbone:**
  - 由高速点到点网络连接的路由器（国家或世界范围）集合collection of routers (nationwide or worldwide) connected by high-speed point-to-point networks
- 互联网交换点 **Internet Exchange Points (IXP):**
  - 连接多个主干的路由器（通常称为对等体）router that connects multiple backbones (often referred to as peers)
  - 又称网络接入点（NAP）Also called Network Access Points (NAP)
- 区域网络 **Regional networks:**
  - 覆盖较小地理区域的小型主干（例如城市或州） smaller backbones that cover smaller geographical areas (e.g., cities or states)
- 现场点 **Point of presence (POP):**
  - 连接到互联网的机器 machine that is connected to the Internet
- 互联网服务提供商 **Internet Service Providers (ISPs):**
  - 提供拨号或直接接入到POP provide dial-up or direct access to POPs

互联网连接层次结构 Internet Connection Hierarchy



IP地址结构 IP Address Structure

- **IP(V4)地址空间分成类：IP (V4) Address space divided into classes:**



- **网络号记为w.x.y.z/n形式 Network ID Written in form w.x.y.z/n**
  - n为主机地址部分的位数 n = number of bits in host address
  - 例如CMU记为： 128.2.0.0/16　　E.g., CMU written as 128.2.0.0/16
    - B类地址 Class B address
- **不能路由的（私有）IP地址 Unrouted (private) IP addresses:**

  10.0.0.0/8　172.16.0.0/12　192.168.0.0/16

互联网的演变 Evolution of Internet

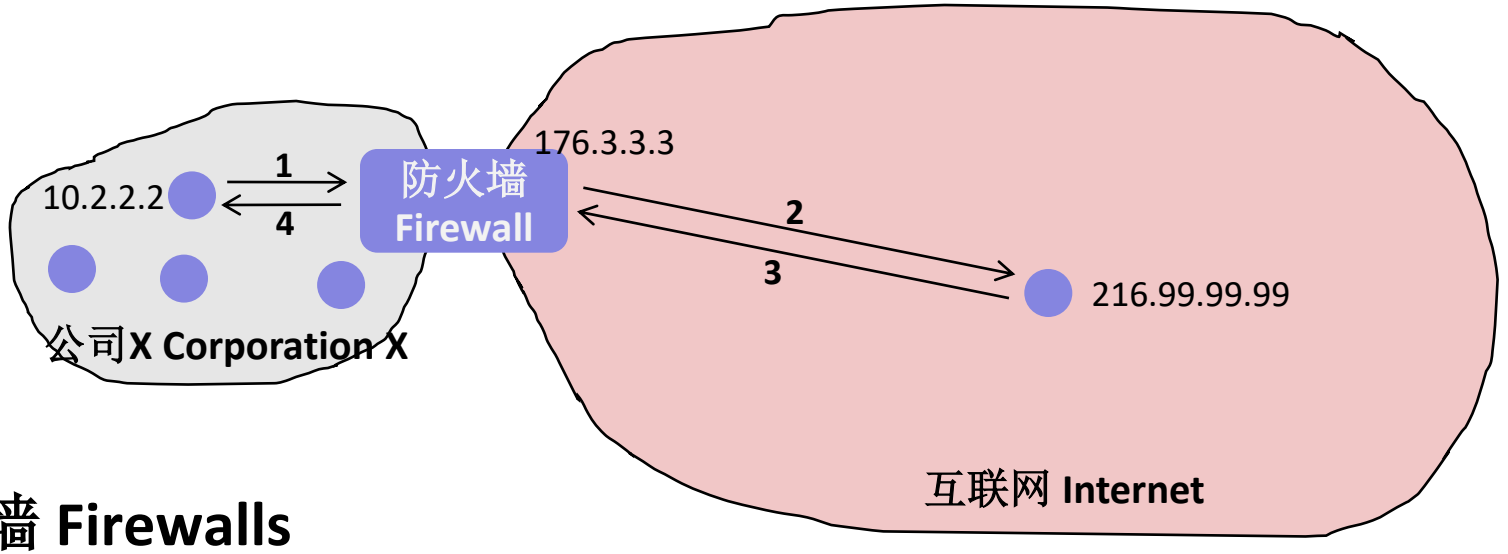
- 原始思想 **Original Idea**
  - 互联网上的每个节点都有唯一的IP地址 Every node on Internet would have unique IP address
    - 每个人都可以直接与每个人交谈 Everyone would be able to talk directly to everyone
  - 无保密或认证 No secrecy or authentication
    - 消息对同一局域网上的路由器和主机均可见 Messages visible to routers and hosts on same LAN
    - 可能伪造分组首部中的源地址字段 Possible to forge source field in packet header

- 缺点 **Shortcomings**
  - 没有足够的IP地址可用 There aren't enough IP addresses available
  - 不希望每个人都能访问或了解所有其他主机 Don't want everyone to have access or knowledge of all other hosts
  - 安全问题要求保密和身份验证 Security issues mandate secrecy & authentication

互联网的演变：命名
Evolution of Internet: Naming

- 动态分配地址 **Dynamic address assignment**
  - 大多数主机不需要静态地址 Most hosts don't need to have known address
    - 仅用作服务器才需要 Only those functioning as servers
  - DHCP（动态主机配置协议） DHCP (Dynamic Host Configuration Protocol)
    - 本地ISP为临时使用分配地址 Local ISP assigns address for temporary use
- 例子： **Example:**
  - CMU的笔记本电脑（有线连接） Laptop at CMU (wired connection)
    - IP地址128.2.213.29（bryant-tp4.cs.cmu.edu） IP address 128.2.213.29 (**`bryant-tp4.cs.cmu.edu`**)
    - 静态分配 Assigned statically
  - 家里的笔记本电脑 Laptop at home
    - IP地址192.168.1.5 IP address 192.168.1.5
    - 仅在家庭网络内有效 Only valid within home network

互联网的演变：防火墙
Evolution of Internet: Firewalls



- 防火墙 Firewalls
  - 对互联网的其他部分隐藏组织节点 Hides organizations nodes from rest of Internet
  - 在组织内使用本地IP地址 Use local IP addresses within organization
  - 对于外部服务，提供代理服务 For external service, provides proxy service
    1. 客户端请求： Client request: src=10.2.2.2, dest=216.99.99.99
    2. 防火墙转发： Firewall forwards: src=176.3.3.3, dest=216.99.99.99
    3. 服务器响应： Server responds: src=216.99.99.99, dest=176.3.3.3
    4. 防火墙转发响应： Firewall forwards response: src=216.99.99.99, dest=10.2.2.2

第11章 网络编程

Network Programming: Part II

100076202： 计算机系统导论

任课教师：

宿红毅　　张艳　　黎有琦　　颜珂

原作者：

Randal E. **Bryant and** David R. O'Hallaron

Carnegie Mellon University

议题

- 上次的问题 Questions from yesterday
- 上次没有讨论的内容 Material we didn't get to yesterday
  - 使用套接字发送数据 Transmitting data using sockets
  - 套接字地址 Socket addresses
  - `getaddrinfo`
- 建立连接 Setting up connections
- 应用层协议示例：HTTP　Application protocol example: HTTP

协议栈 Protocol Stacks

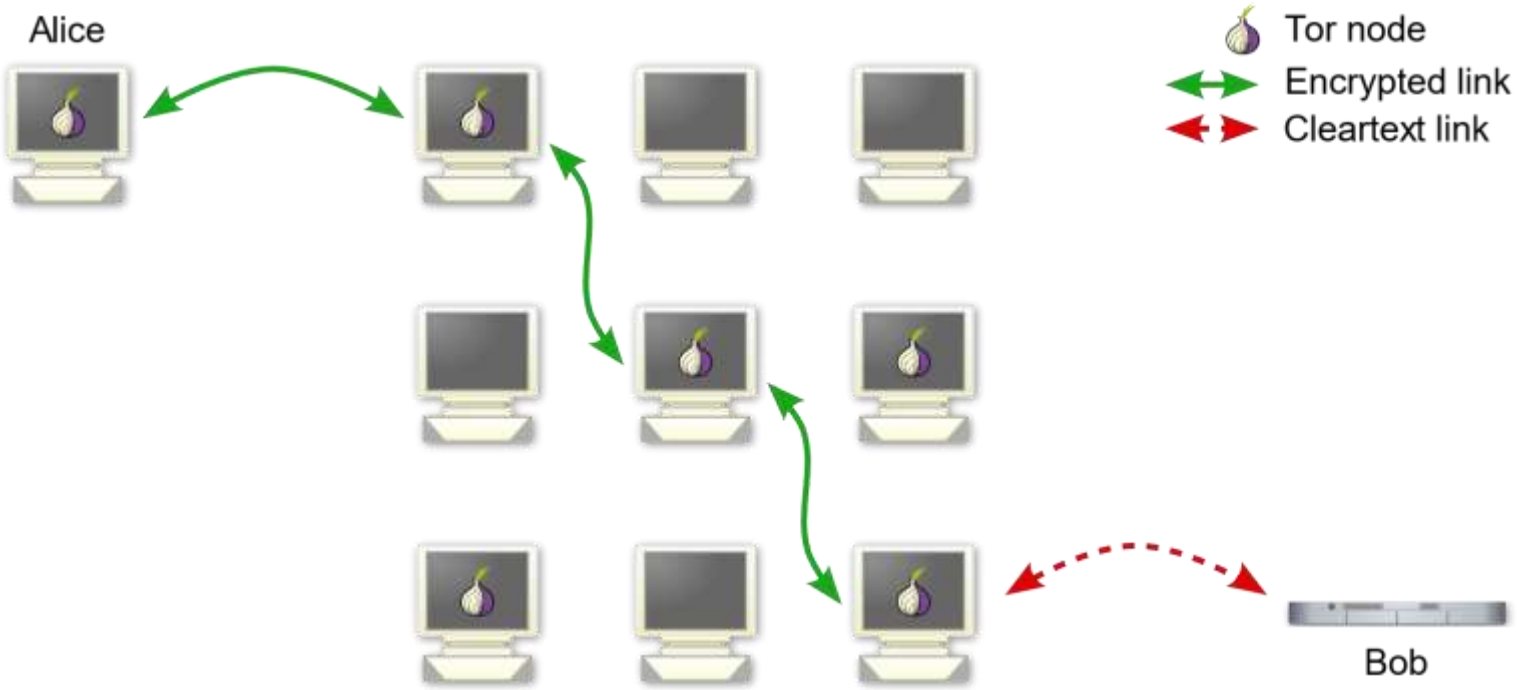
OSI模型 OSI Model

应用层 Application
表示层 Presentation
会话层 Session
运输层 Transport
网络层 Network
数据链路层 Data Link
物理层 Physical

互联网模型 Internet Model

<table>
<tr><td>应用层
Application</td><td>HTTP</td><td>SMTP</td><td rowspan="2">SSH</td><td>DNS</td></tr>
<tr><td>安全
Security</td><td colspan="2">TLS</td><td></td></tr>
<tr><td>运输层
Transport</td><td colspan="3">TCP</td><td>UDP</td></tr>
<tr><td>寻址
Addressing</td><td colspan="4">IP</td></tr>
<tr><td>物理层
Physical
Link</td><td colspan="2">Ethernet</td><td>WiFi</td><td>SDH</td></tr>
</table>

洋葱网站又名“暗网”
Onion sites aka “the dark web”

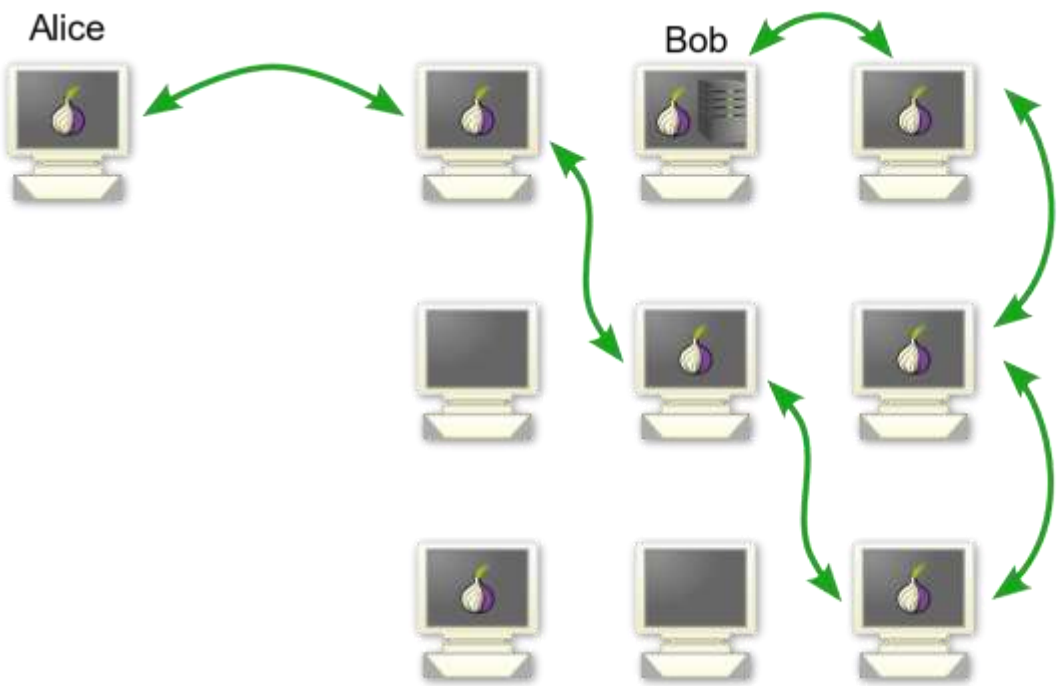


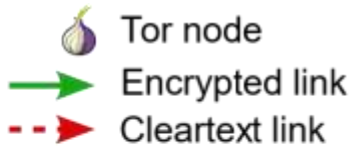
这不是暗网。这只是使用Tor（洋葱路由器 The Onion Router）来保护“网络”的正常使用。

This isn’t the dark web.

This is just using Tor to protect regular use of the ‘net.

洋葱网站又名“暗网”
Onion sites aka “the dark web”





“暗网”由只有在使用Tor和类似协议时才能访问的网站和其他服务组成。

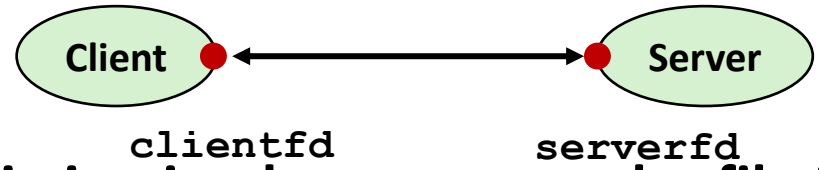
The “dark web” consists of web sites and other services that are accessible *only* when using Tor and similar protocols.

议题

- 上次的问题 Questions from yesterday
- **上次没有讨论的内容 Material we didn't get to yesterday**
  - 使用套接字发送数据 Transmitting data using sockets
  - 套接字地址 Socket addresses
  - **`getaddrinfo`**
- 建立连接 Setting up connections
- 应用层协议示例：HTTP　Application protocol example: HTTP

Sockets

- **What is a socket?**
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - Using the FD abstraction lets you reuse code & interfaces
- **Clients and servers communicate with each other by reading from and writing to socket descriptors**

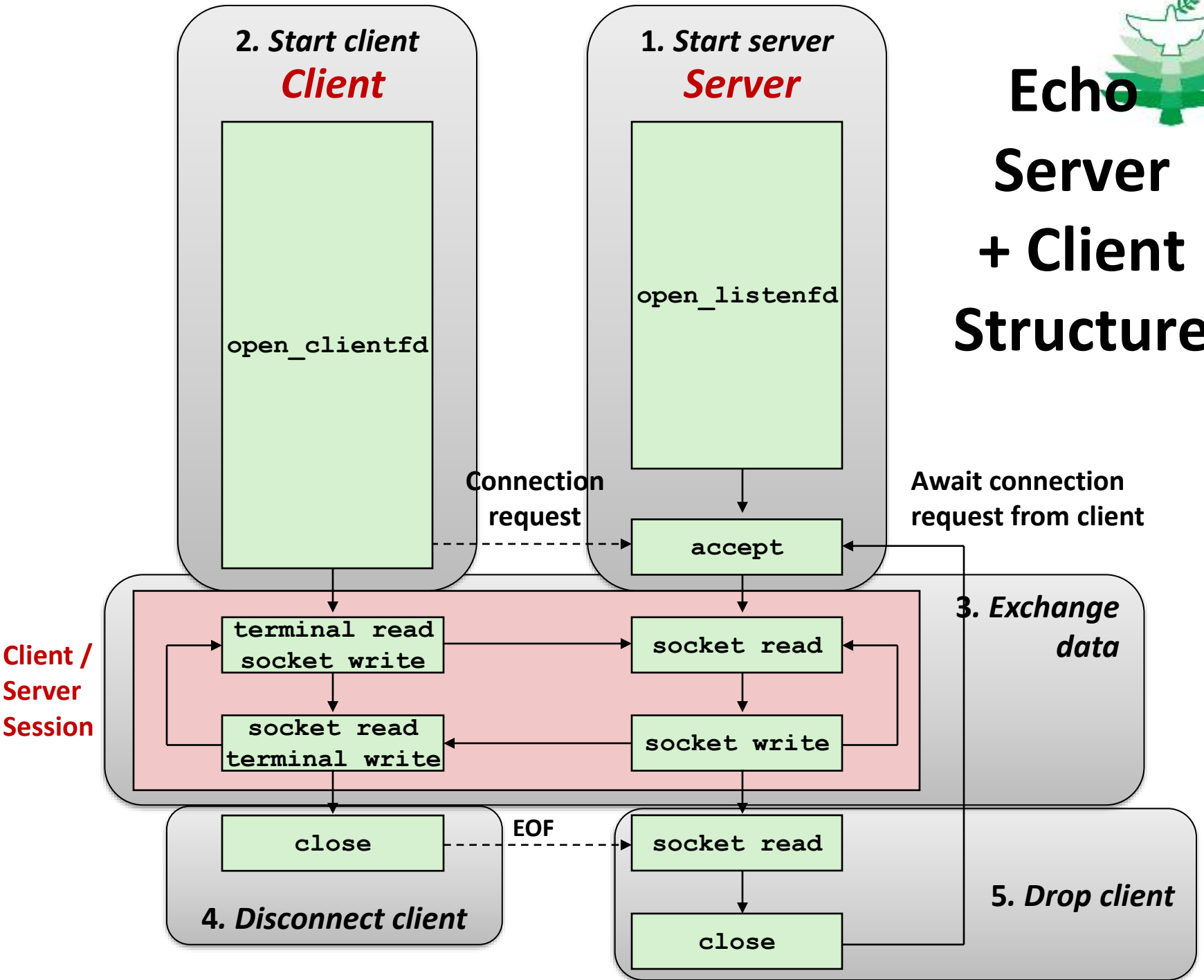


- **The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors**

Socket Programming Example

- **Echo server and client**
- **Server**
  - Accepts connection request
  - Repeats back lines as they are typed
- **Client**
  - Requests connection to server
  - Repeatedly:
    - Read line from terminal
    - Send to server
    - Read reply from server
    - Print line to terminal

Echo Server + Client Structure



Recall: Unbuffered RIO Input/Output

- **Same interface as Unix `read` and `write`**
- **Especially useful for transferring data on network sockets**

```
#include "csapp.h"

ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

Return: num. bytes transferred if OK, 0 on EOF (`rio_readn` only), -1 on error

- **`rio_readn`** returns short count only if it encounters EOF
  - Only use it when you know how many bytes to read
- **`rio_writen`** never returns a short count
- Calls to **`rio_readn`** and **`rio_writen`** can be interleaved arbitrarily on the same descriptor

Recall: Buffered RIO Input Functions

- **Efficiently read text lines and binary data from a file partially cached in an internal memory buffer**

```
#include "csapp.h"

void rio_readinitb(rio_t *rp, int fd);

ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
```

Return: num. bytes read if OK, 0 on EOF, -1 on error

  - `rio_readlineb` reads a ***text line*** of up to **`maxlen`** bytes from file **`fd`** and stores the line in **`usrbuf`**
    - Especially useful for reading text lines from network sockets
  - Stopping conditions
    - **`maxlen`** bytes read
    - EOF encountered
    - Newline (‘**`\n`**’) encountered

Echo Client: Main Routine

```
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd;
    char *host, *port, buf[MAXLINE];
    rio_t rio;

    host = argv[1];
    port = argv[2];

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

echoclient.c

Echo Server: `echo` function

- **The server uses RIO to read and echo text lines until EOF (end-of-file) condition is encountered.**
  - EOF condition caused by client calling **`close(clientfd)`**

```
void echo(int connfd)
{
    size_t n;
    char buf[MAXLINE];
    rio_t rio;

    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
        printf("server received %d bytes\n", (int)n);
        Rio_writen(connfd, buf, n);
    }
}
```

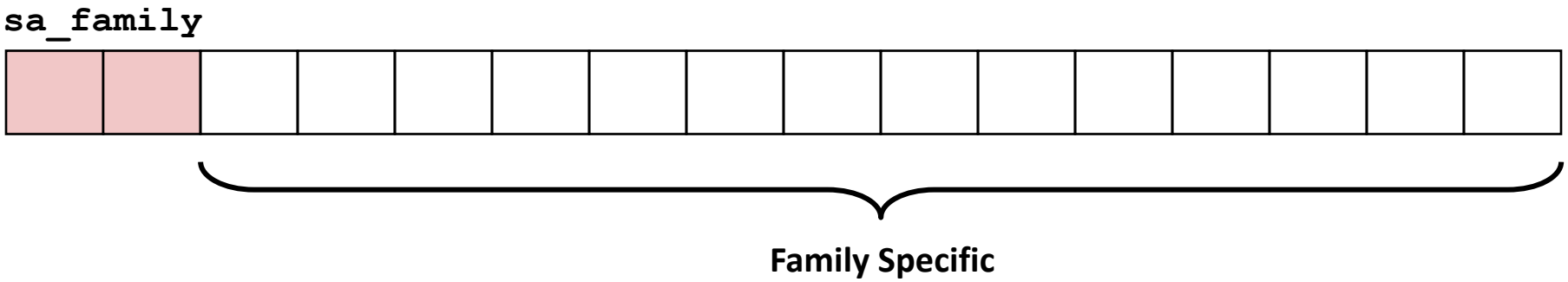
echo.c

Socket Address Structures

- **Generic socket address:**
  - For address arguments to **`connect`**, **`bind`**, and **`accept`** *(next lecture)*
  - In C++ this would be an abstract base class
  - For casting convenience, we adopt the Stevens convention:

    `typedef struct sockaddr SA;`

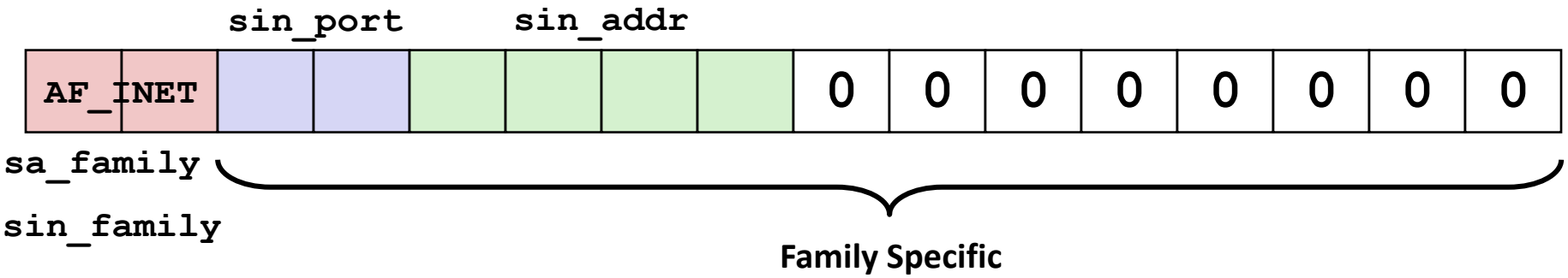
```
struct sockaddr {
  uint16_t  sa_family;    /* Protocol family */
  char      sa_data[14];  /* Address data  */
};
```



Socket Address Structures

- **Internet (IPv4) specific socket address:**
  - Must cast (`struct sockaddr_in *`) to (`struct sockaddr *`) for functions that take socket address arguments.

```
struct sockaddr_in  {
  uint16_t         sin_family;  /* Protocol family (always AF_INET) */
  uint16_t         sin_port;    /* Port num in network byte order */
  struct in_addr   sin_addr;    /* IP addr in network byte order */
  unsigned char    sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```



Host and Service Conversion: `getaddrinfo`

- **`getaddrinfo` is the modern way to convert string representations of hostnames, host addresses, ports, and service names to socket address structures.**
  - Replaces obsolete `gethostbyname` and `getservbyname` funcs.

- **Advantages:**
  - Reentrant (can be safely used by threaded programs).
  - Allows us to write portable protocol-independent code
    - Works with both IPv4 and IPv6

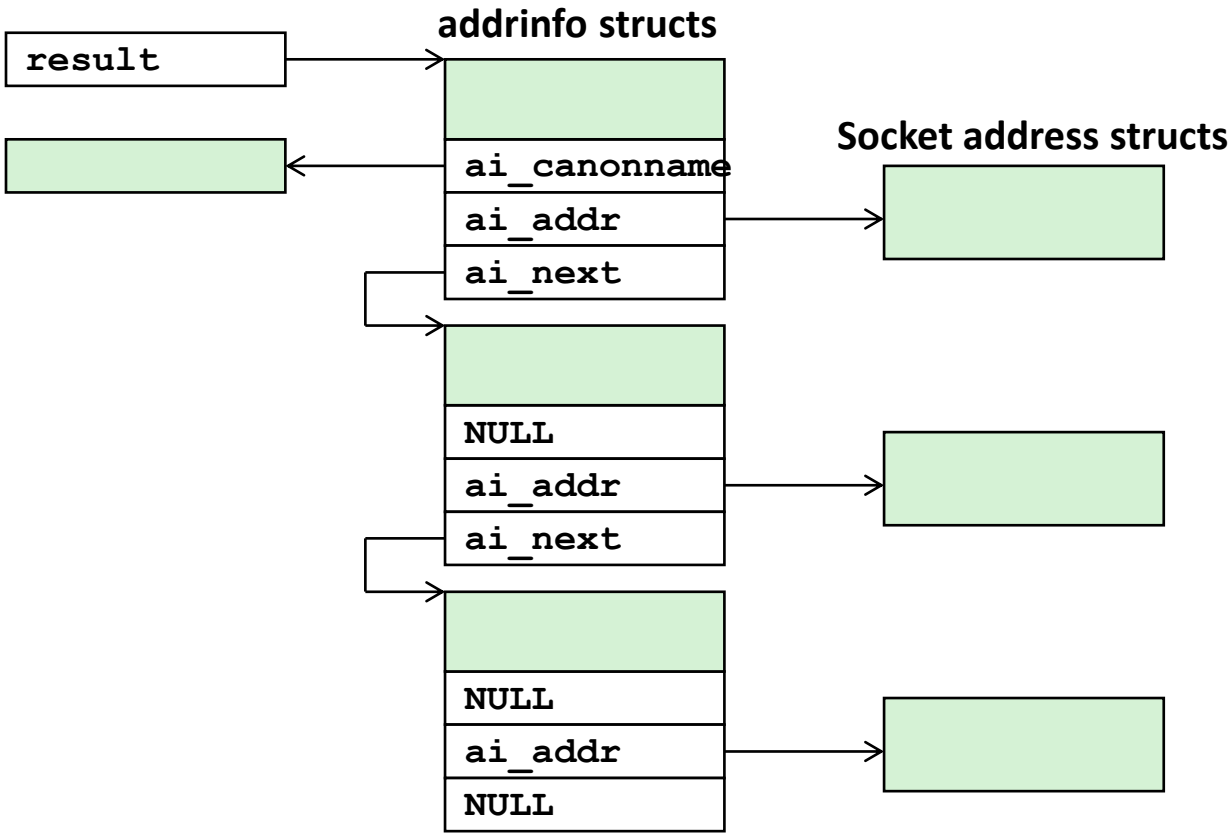
- **Disadvantages**
  - Somewhat complex
  - Fortunately, a small number of usage patterns suffice in most cases.

Host and Service Conversion: `getaddrinfo`

```
int getaddrinfo(const char *host,              /* Hostname or address */
                const char *service,           /* Port or service name */
                const struct addrinfo *hints,/* Input parameters */
                struct addrinfo **result);    /* Output linked list */

void freeaddrinfo(struct addrinfo *result);   /* Free linked list */

const char *gai_strerror(int errcode);        /* Return error msg */
```

- Given `host` and `service`, `getaddrinfo` returns `result` that points to a linked list of `addrinfo` structs, each of which points to a corresponding socket address struct, and which contains arguments for the sockets interface functions.
- Helper functions:
  - `freeadderinfo` frees the entire linked list.
  - `gai_strerror` converts error code to an error message.

Linked List Returned by `getaddrinfo`



addrinfo Struct

```
struct addrinfo {
    int              ai_flags;      /* Hints argument flags */
    int              ai_family;     /* First arg to socket function */
    int              ai_socktype;   /* Second arg to socket function */
    int              ai_protocol;   /* Third arg to socket function  */
    char            *ai_canonname;  /* Canonical host name */
    size_t           ai_addrlen;    /* Size of ai_addr struct */
    struct sockaddr *ai_addr;       /* Ptr to socket address structure */
    struct addrinfo *ai_next;       /* Ptr to next item in linked list */
};
```

- **Each addrinfo struct returned by getaddrinfo contains arguments that can be passed directly to `socket` function.**
- **Also points to a socket address struct that can be passed directly to `connect` and `bind` functions .**

(**`socket`**, **`connect`**, **`bind`** to be discussed next)

Host and Service Conversion: `getnameinfo`

- **`getnameinfo` is the inverse of getaddrinfo, converting a socket address to the corresponding host and service.**
  - Replaces obsolete `gethostbyaddr` and `getservbyport` funcs.
  - Reentrant and protocol independent.

```
int getnameinfo(const SA *sa, socklen_t salen, /* In: socket addr */
                char *host, size_t hostlen,    /* Out: host */
                char *serv, size_t servlen,    /* Out: service */
                int flags);                    /* optional flags */
```

Conversion Example

```c
#include "csapp.h"

int main(int argc, char **argv)
{
    struct addrinfo *p, *listp, hints;
    char buf[MAXLINE];
    int rc, flags;

    /* Get a list of addrinfo records */
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 only */
    hints.ai_socktype = SOCK_STREAM; /* Connections only */
    if ((rc = getaddrinfo(argv[1], NULL, &hints, &listp)) != 0) {
        fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(rc));
        exit(1);
    }
```

hostinfo.c

Conversion Example (cont)

```
    /* Walk the list and display each IP address */
    flags = NI_NUMERICHOST; /* Display address instead of name */
    for (p = listp; p; p = p->ai_next) {
        Getnameinfo(p->ai_addr, p->ai_addrlen,
                    buf, MAXLINE, NULL, 0, flags);
        printf("%s\n", buf);
    }

    /* Clean up */
    Freeaddrinfo(listp);

    exit(0);
}
```

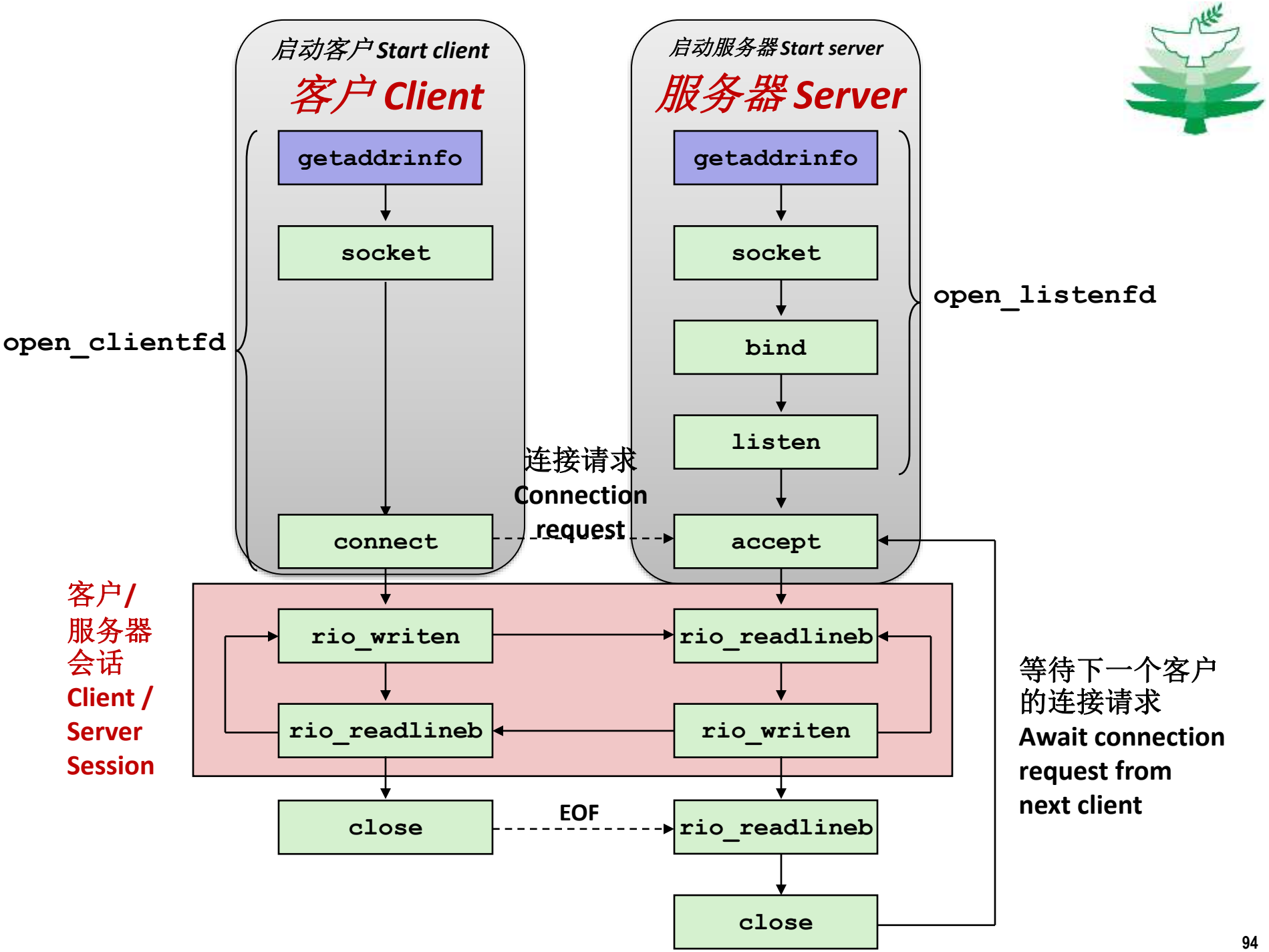
hostinfo.c

Running hostinfo

```
whaleshark> ./hostinfo localhost
127.0.0.1

whaleshark> ./hostinfo whaleshark.ics.cs.cmu.edu
128.2.210.175

whaleshark> ./hostinfo twitter.com
199.16.156.230
199.16.156.38
199.16.156.102
199.16.156.198

whaleshark> ./hostinfo google.com
172.217.15.110
2607:f8b0:4004:802::200e
```

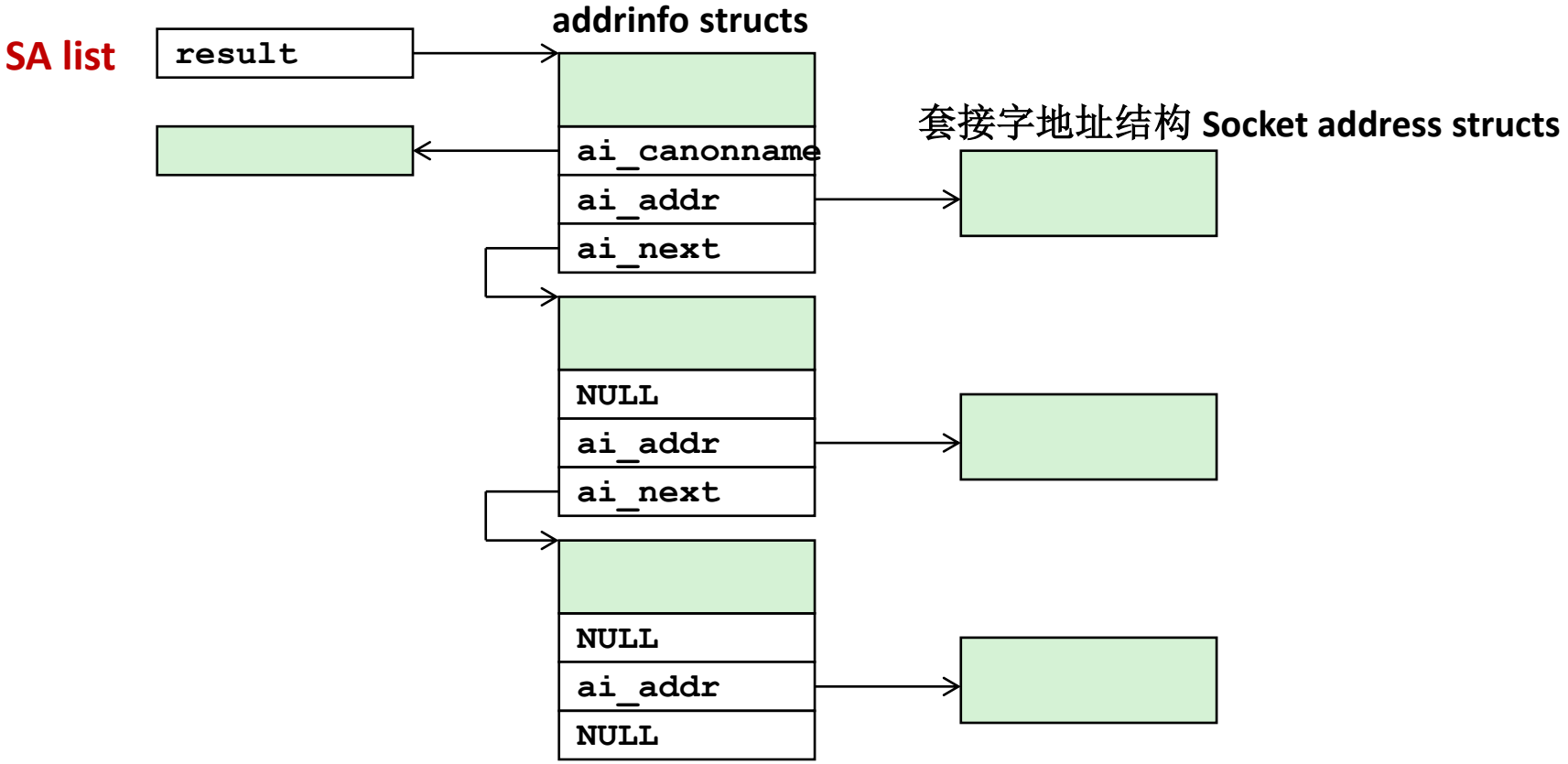
议题

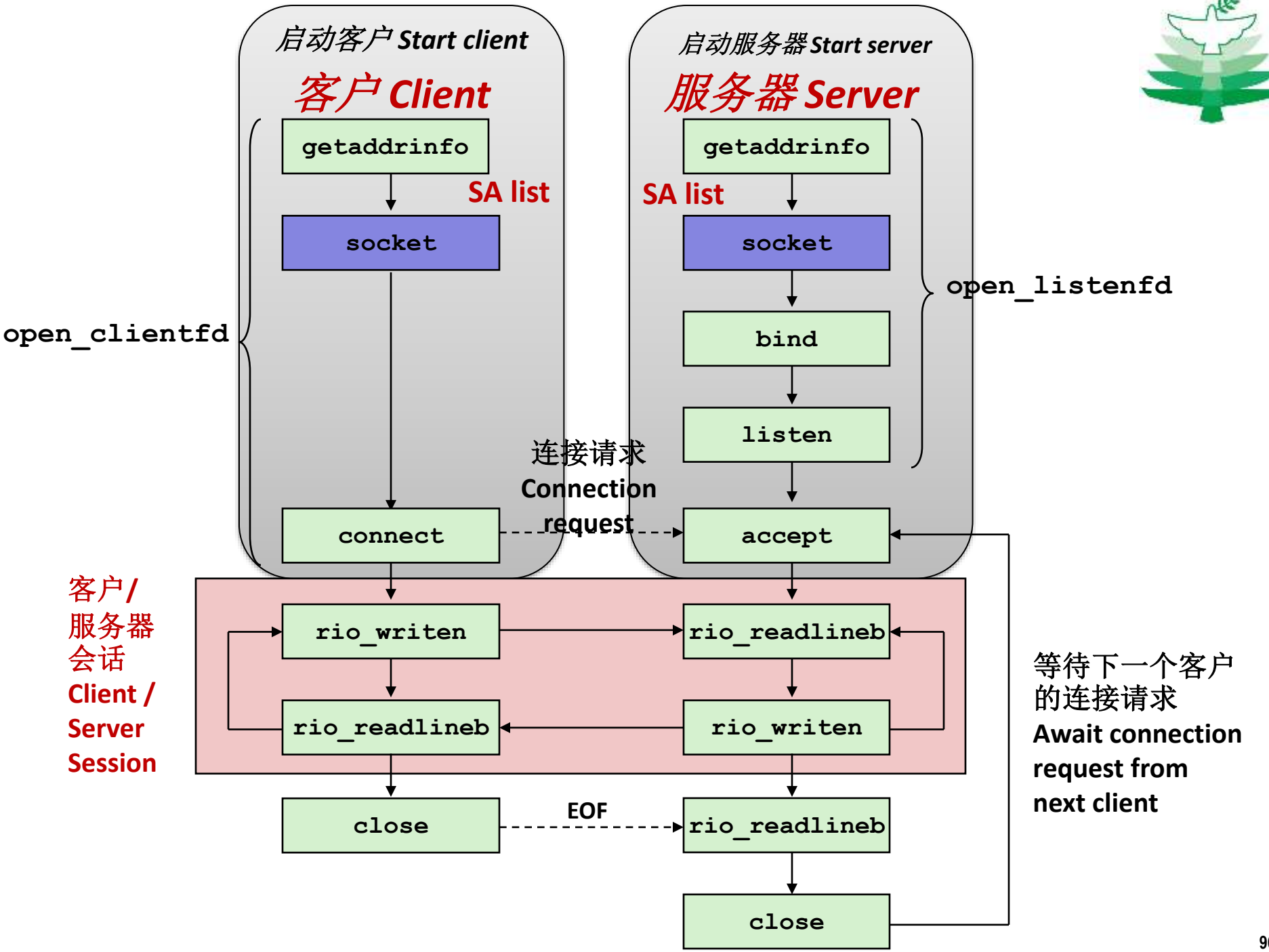
- 上次的问题 Questions from yesterday
- 上次没有讨论的内容 Material we didn't get to yesterday
  - 使用套接字发送数据 Transmitting data using sockets
  - 套接字地址 Socket addresses
  - `getaddrinfo`
- **建立连接 Setting up connections**
- 应用层协议示例：HTTP　Application protocol example: HTTP


启动客户 Start client
客户 Client
启动服务器 Start server
服务器 Server
getaddrinfo
socket
connect
getaddrinfo
socket
bind
listen
accept
open_clientfd
open_listenfd
连接请求
Connection
request
客户/
服务器
会话
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close
等待下一个客户
的连接请求
Await connection
request from
next client

复习：getaddrinfo Review: getaddrinfo

- Getaddrinfo将主机名、主机地址、端口和服务名的字符串表示转换成套接字地址结构 getaddrinfo converts string representations of hostnames, host addresses, ports, service names to socket address structures




启动客户 Start client
客户 Client
getaddrinfo
SA list
socket
open_clientfd
connect
启动服务器 Start server
服务器 Server
getaddrinfo
SA list
socket
bind
listen
open_listenfd
accept
连接请求
Connection
request
客户/
服务器
会话
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close
等待下一个客户
的连接请求
Await connection
request from
next client

套接字接口：socket
Sockets Interface: socket

- 客户和服务器使用socket函数创建套接字描述符 Clients and servers use the `socket` function to create a *socket descriptor*:

```
int socket(int domain, int type, int protocol)
```

- 示例： Example:

```
int clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

特定协议！
Protocol specific!

指示正在使用32位IPv4地址
Indicates that we are using 32-bit IPV4 addresses

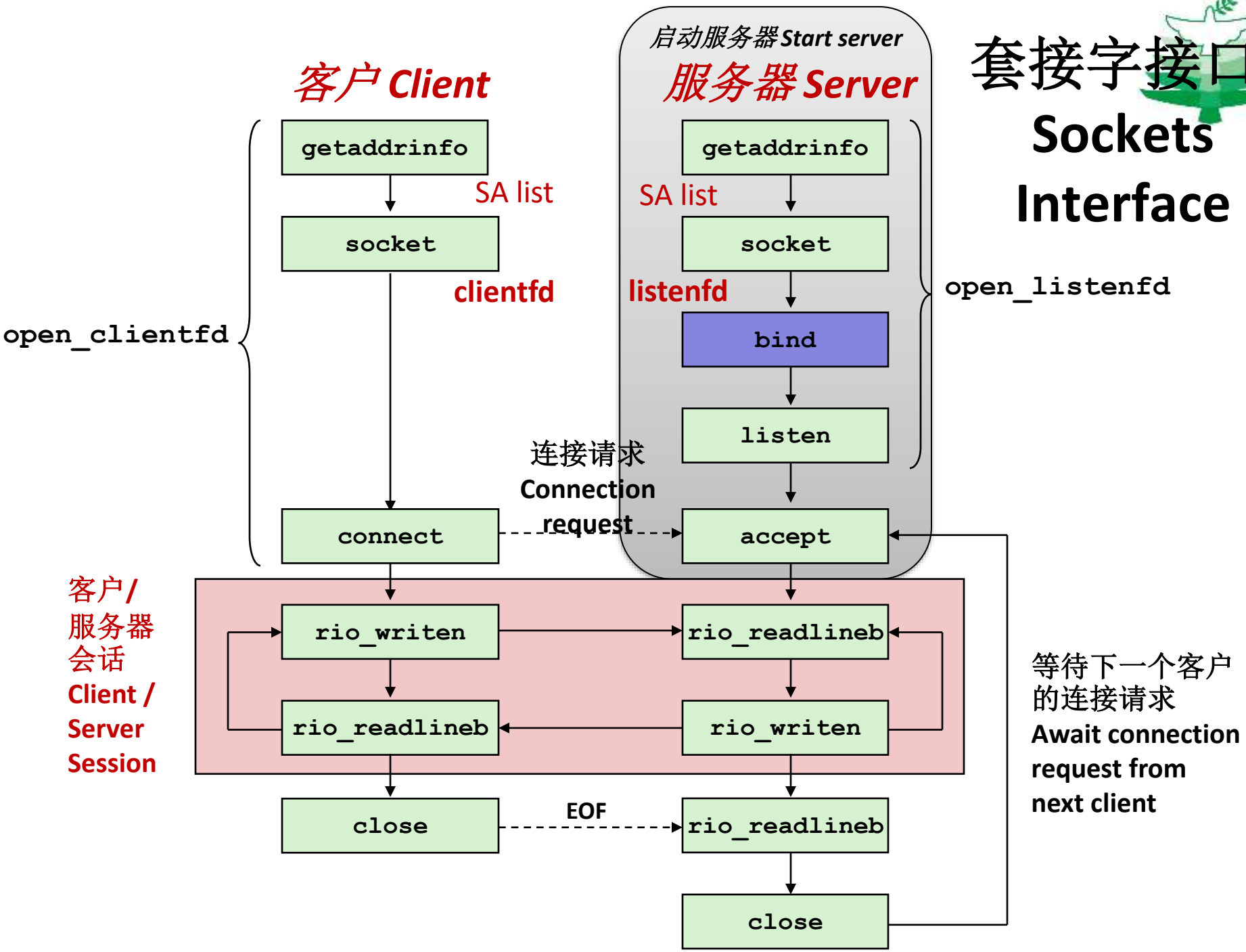
指示套接字为可靠（TCP）连接端点
Indicates that the socket will be the end point of a reliable (TCP) connection

- 示例： Example:

```
int clientfd = socket(ai->ai_family, ai->ai_socktype,
                      ai->ai_protocol);
```

使用getaddrinfo而且不必知道或关心用哪个协议
Use getaddrinfo and you don't have to know or care which protocol!

套接字接口
Sockets Interface



套接字接口：bind
Sockets Interface: bind

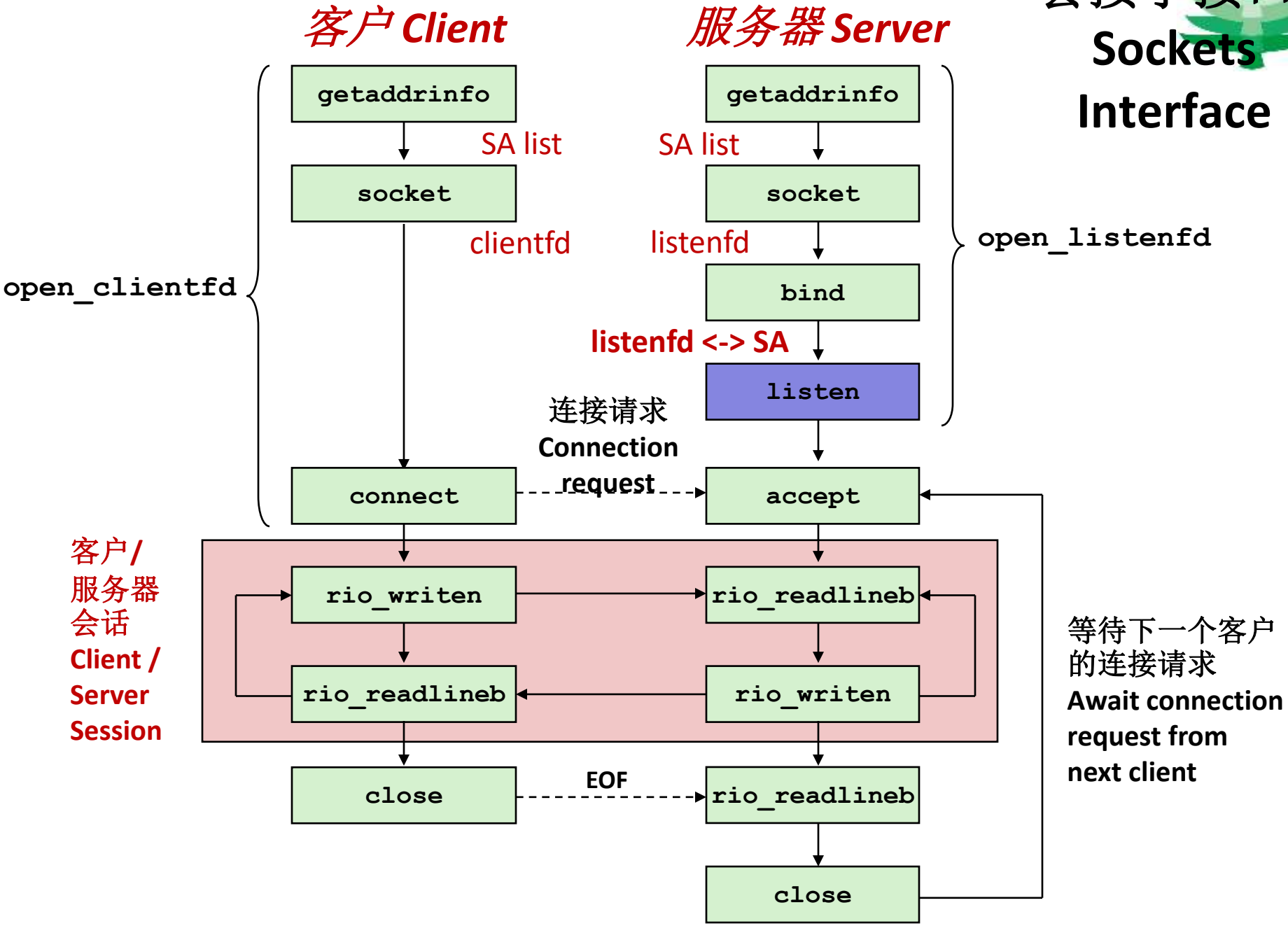
- 服务器使用bind要求内核将服务器套接字地址和套接字描述符相关联 A server uses `bind` to ask the kernel to associate the server's socket address with a socket descriptor:

```
int bind(int sockfd, SA *addr, socklen_t addrlen);
```

Our convention: `typedef struct sockaddr SA;`

- 进程可以通过读取描述符sockfd来读取连接端点为addr到达的字节 Process can read bytes that arrive on the connection whose endpoint is `addr` by reading from descriptor `sockfd`
- 类似地，对sockfd的写入是沿着连接端点addr进行传输 Similarly, writes to `sockfd` are transferred along connection whose endpoint is `addr`
- 最佳实践是使用getaddrinfo提供参数addr和addrlen Best practice is to use `getaddrinfo` to supply the arguments `addr` and `addrlen`.

套接字接口 Sockets Interface



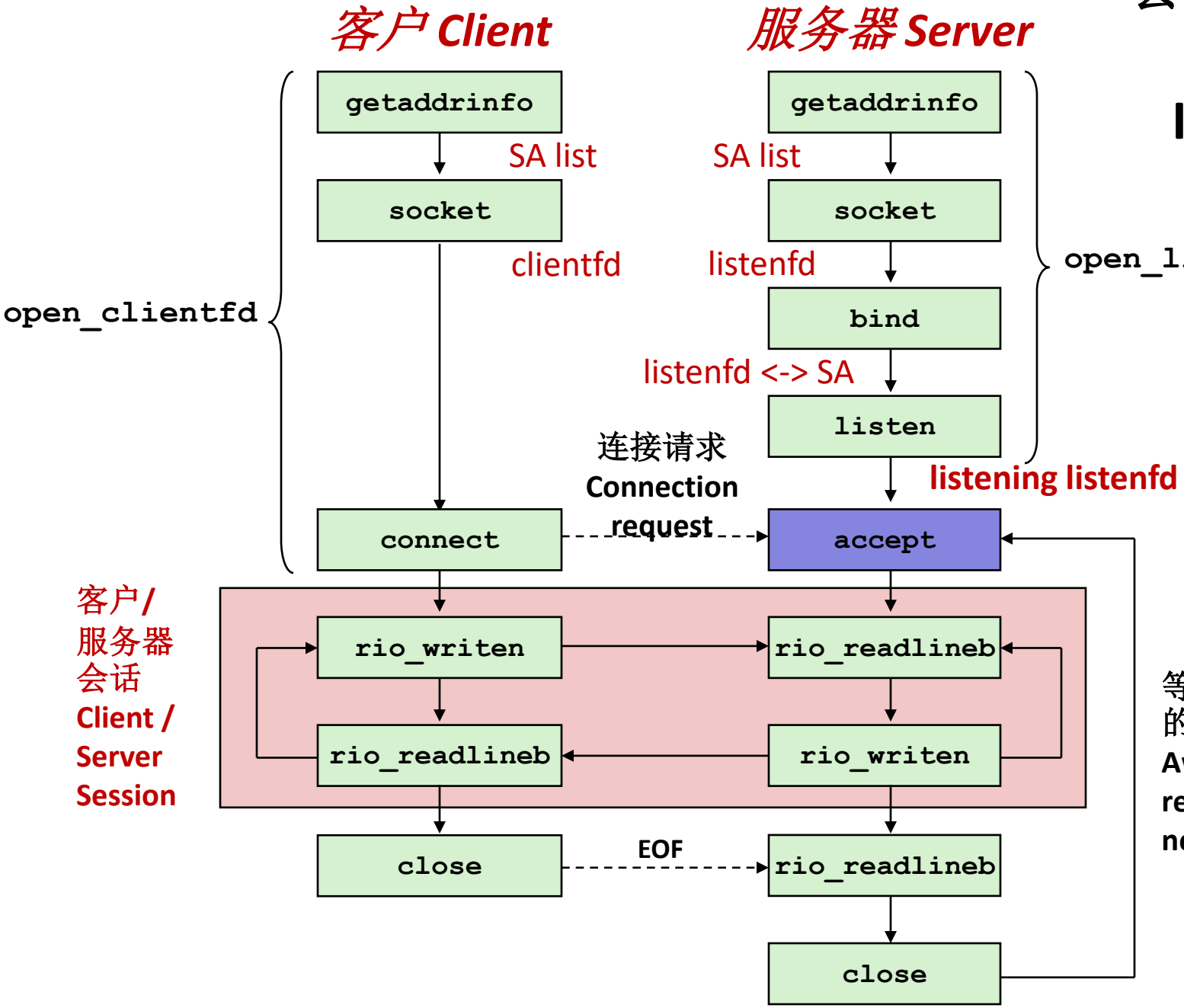
套接字接口：listen
Sockets Interface: `listen`

- 内核假设来自socket函数的描述符是一个活动套接字，该套接字处于客户端 Kernel assumes that descriptor from socket function is an *active socket* that will be on the client end
- 服务器调用listen函数，告诉内核描述符将由服务器而不是客户端使用： A server calls the listen function to tell the kernel that a descriptor will be used by a server rather than a client:

```
int listen(int sockfd, int backlog);
```

- 将sockfd从活动套接字转换为可接受客户端连接请求的侦听套接字 Converts `sockfd` from an active socket to a *listening socket* that can accept connection requests from clients.
- backlog是关于内核在开始拒绝请求之前应该排队的未完成连接请求数量的提示（默认情况下为128） `backlog` is a hint about the number of outstanding connection requests that the kernel should queue up before starting to refuse requests (128-ish by default)

套接字接口 Sockets Interface



套接字接口：accept
Sockets Interface: accept

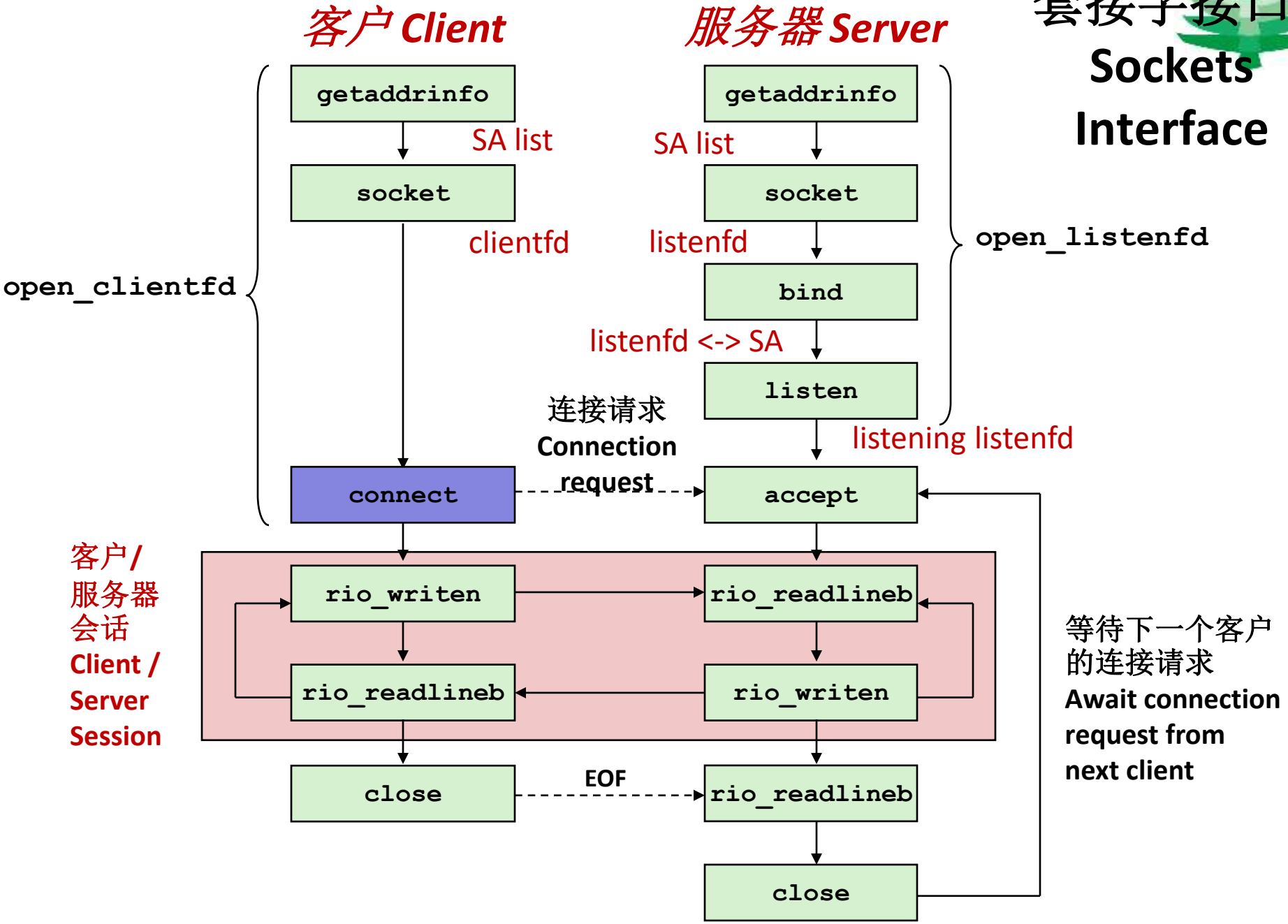
- 服务器通过调用accept等待来自客户端的连接请求：
  Servers wait for connection requests from clients by calling `accept`:

```
int accept(int listenfd, SA *addr, int *addrlen);
```

- 等待连接请求到达绑定到listenfd的连接，然后在addr中填写客户端的套接字地址，在addrlen中填写套接字地址的大小 Waits for connection request to arrive on the connection bound to `listenfd`, then fills in client's socket address in `addr` and size of the socket address in `addrlen`.

- 返回可用于通过Unix I/O例程与客户端通信的连接描述符 connfd Returns a *connected descriptor* `connfd` that can be used to communicate with the client via Unix I/O routines.

套接字接口 Sockets Interface



套接字接口：connect
Sockets Interface: connect

- 客户端通过调用connect建立与服务器的连接： A client establishes a connection with a server by calling `connect`:

```
int connect(int clientfd, SA *addr, socklen_t addrlen);
```

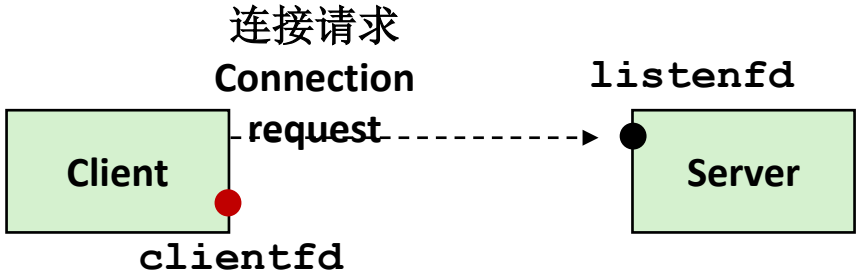
- 尝试在套接字地址addr处与服务器建立连接 Attempts to establish a connection with server at socket address `addr`
  - 如果成功，那么clientfd现在就可以读写了。 If successful, then **`clientfd`** is now ready for reading and writing.
  - 生成的连接以套接字对为特征 Resulting connection is characterized by socket pair

    （**`x:y, addr.sin_addr:addr.sin_port`**）
    - x是客户端地址 **`x`** is client address
    - y是唯一标识客户端主机上的客户端进程的临时端口 **`y`** is ephemeral port that uniquely identifies client process on client host
- 最佳实践是使用getaddrinfo提供参数addr和addrlen Best practice is to use `getaddrinfo` to supply the arguments

connect/accept Illustrated-说明

1.服务器阻塞在accept函数，等待侦听描述符listenfd上的连接请求

1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`



2.客户端通过阻塞式调用connect来发出连接请求

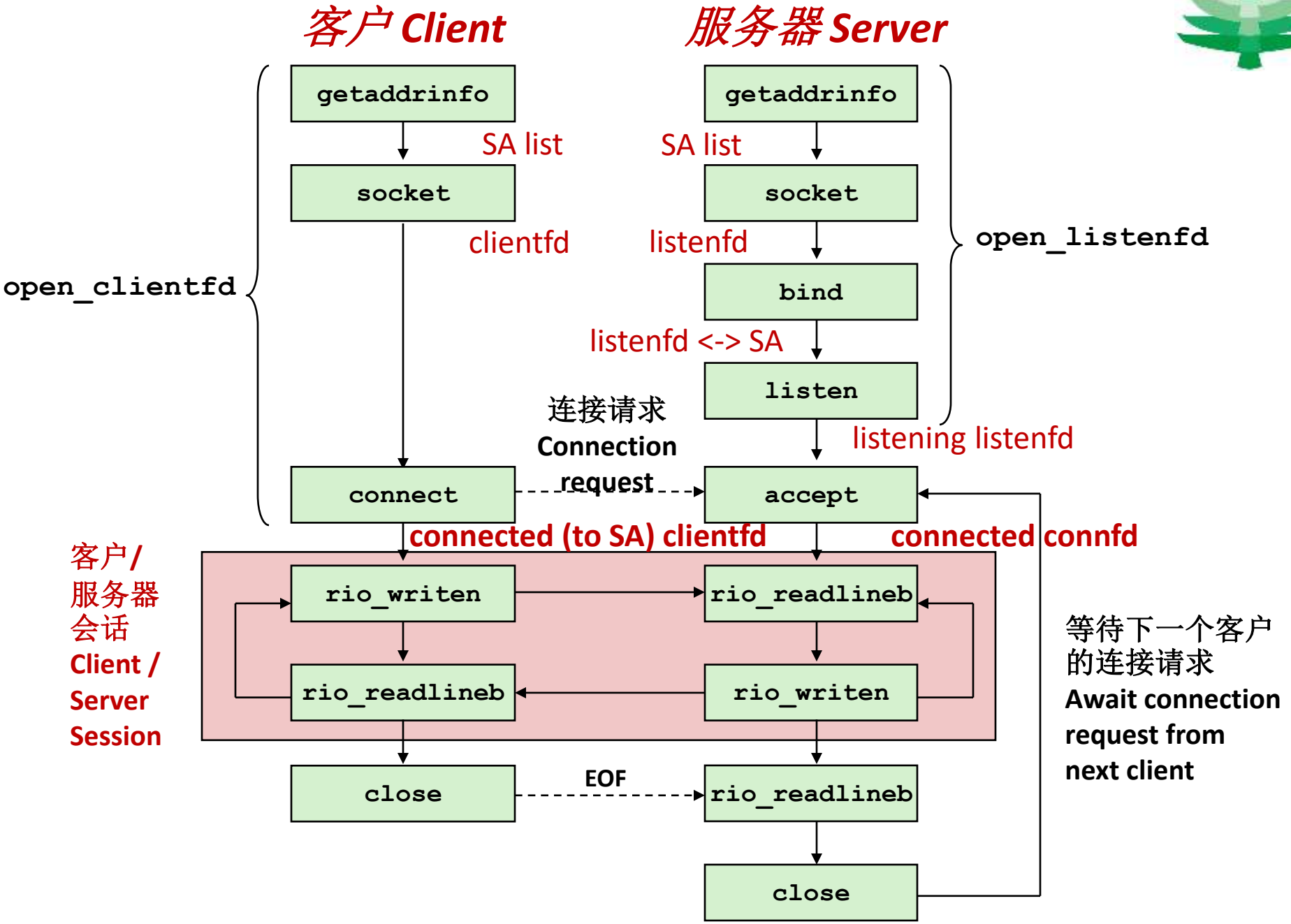
2. Client makes connection request by calling and blocking in `connect`

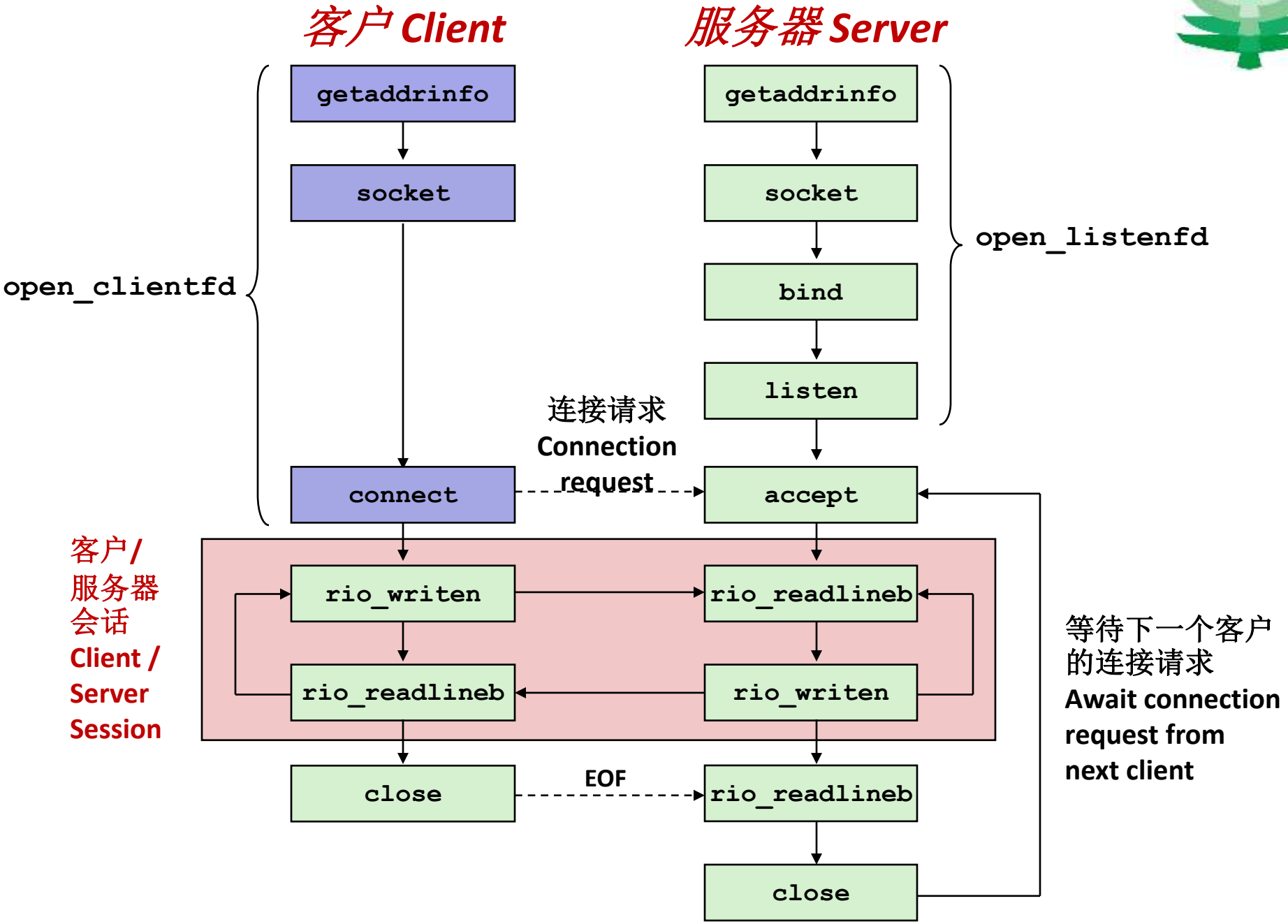
3.服务器从accept返回connfd，客户端从connect返回。现在已在clientfd和connfd之间建立连接

3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`

连接与侦听描述符对比
Connected vs. Listening Descriptors

- 侦听描述符 **Listening descriptor**
  - 客户端连接请求的端点 End point for client connection requests
  - 创建一次并在服务器的生命周期内存在 Created once and exists for lifetime of the server
- 连接描述符 **Connected descriptor**
  - 客户端和服务器之间连接的端点 End point of the connection between client and server
  - 每当服务器接受来自客户端的连接请求时，都会创建一个新的描述符 A new descriptor is created each time the server accepts a connection request from a client
  - 仅在服务客户端所需的时间内存在 Exists only as long as it takes to service client
- 为什么有区别？ **Why the distinction?**
  - 允许同时通过多个客户端连接进行通信的并发服务器 Allows for concurrent servers that can communicate over many client connections simultaneously
    - 例如每次我们收到一个新的请求时，我们都会创建一个子进程来处理这个请求 E.g., Each time we receive a new request, we fork a


客户 Client
服务器 Server
getaddrinfo
getaddrinfo
SA list
SA list
socket
socket
clientfd
listenfd
open_clientfd
open_listenfd
bind
listenfd <-> SA
listen
listening listenfd
连接请求
Connection
request
connect
accept
connected (to SA) clientfd
connected connfd
客户/
服务器
会话
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close
等待下一个客户
的连接请求
Await connection
request from
next client


客户 Client
服务器 Server
getaddrinfo
socket
connect
getaddrinfo
socket
bind
listen
accept
open_clientfd
open_listenfd
连接请求
Connection
request
客户/
服务器
会话
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close
等待下一个客户
的连接请求
Await connection
request from
next client

套接字助手：open_clientfd
Sockets Helper: open_clientfd

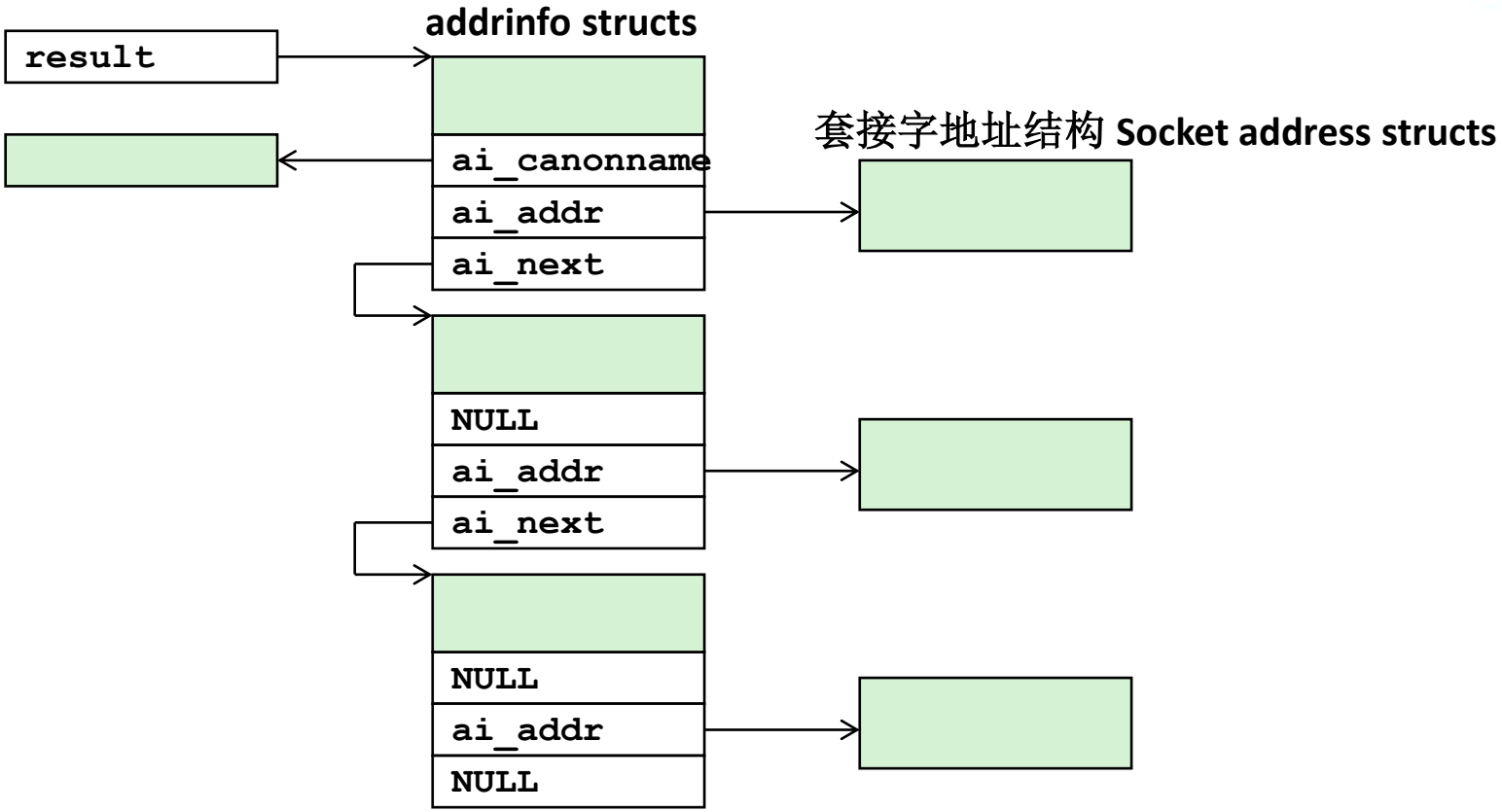
- 与服务器建立一个连接 Establish a connection with a server

```
int open_clientfd(char *hostname, char *port) {
  int clientfd;
  struct addrinfo hints, *listp, *p;

  /* Get a list of potential server addresses */
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_socktype = SOCK_STREAM;  /* Open a connection */
  hints.ai_flags = AI_NUMERICSERV;  /* …using numeric port arg. */
  hints.ai_flags |= AI_ADDRCONFIG;  /* Recommended for connections */
  Getaddrinfo(hostname, port, &hints, &listp);
```

csapp.c

AI_ADDRCONFIG表示“在此计算机上使用IPv4和IPv6中的任何一种”。这是客户机的良好实践，而不是服务器。
AI_ADDRCONFIG means “use whichever of IPv4 and IPv6 works on this computer”. Good practice for clients, not for servers.

getaddrinfo

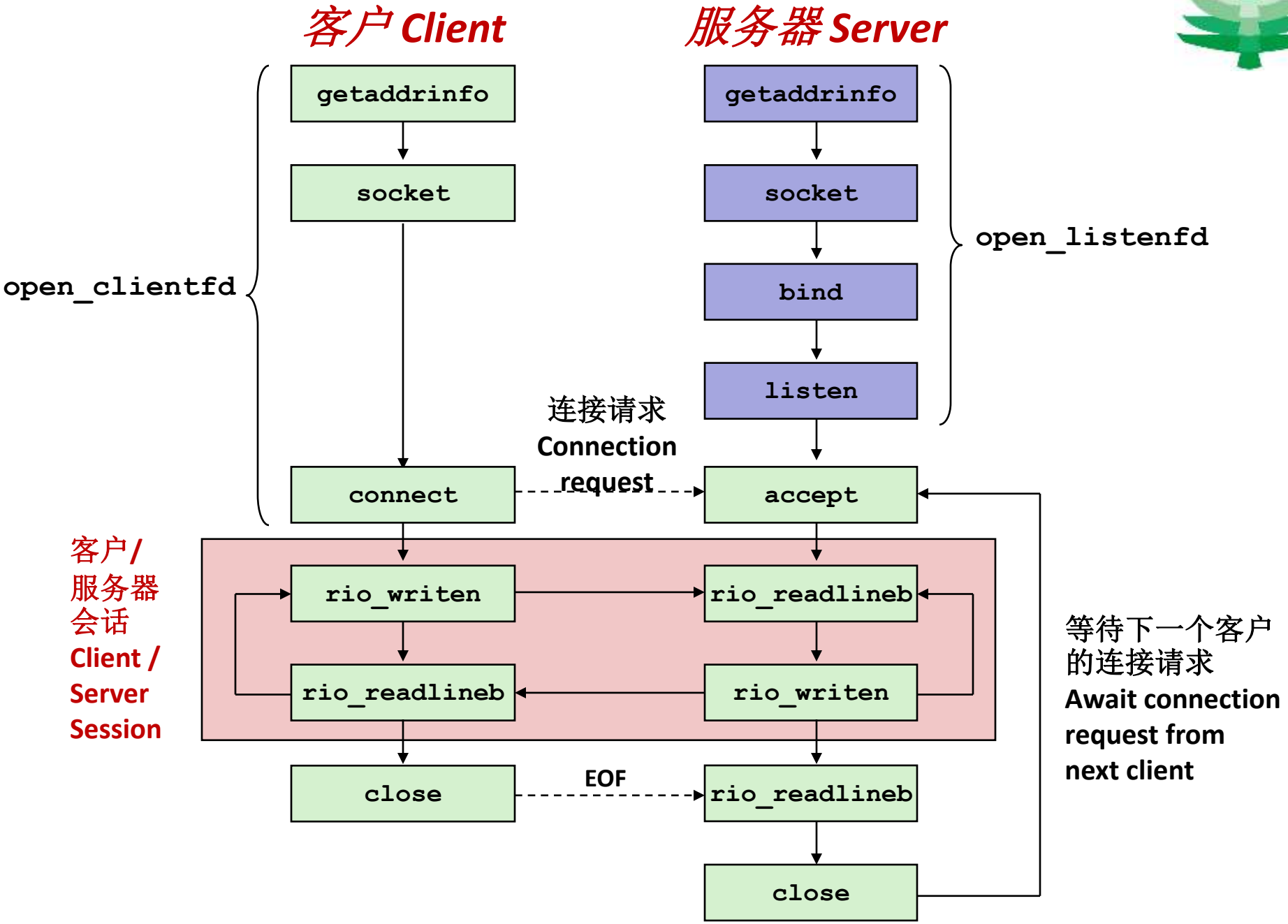


- 客户端：遍历此列表，依次尝试每个套接字地址，直到socket和connect的调用成功 Clients: walk this list, trying each socket address in turn, until the calls to `socket` and `connect` succeed.
- 服务器：遍历列表对所有地址调用socket、listen和bind，然后使用select接受其中任何一个地址上的连接（超出我们的范围） Servers: walk the list calling `socket`, `listen`, `bind` for *all* addresses, then use `select` to accept connections on any of them (beyond our scope)

套接字助手：open_clientfd（续）
Sockets Helper: open_clientfd (cont)

```
    /* Walk the list for one that we can successfully connect to */
    for (p = listp; p; p = p->ai_next) {
        /* Create a socket descriptor */
        if ((clientfd = socket(p->ai_family, p->ai_socktype,
                               p->ai_protocol)) < 0)
            continue; /* Socket failed, try the next */

        /* Connect to the server */
        if (connect(clientfd, p->ai_addr, p->ai_addrlen) != -1)
            break; /* Success */
        Close(clientfd); /* Connect failed, try another */
    }

    /* Clean up */
    Freeaddrinfo(listp);
    if (!p) /* All connects failed */
        return -1;
    else    /* The last connect succeeded */
        return clientfd;
}
```

csapp.c


客户 Client
服务器 Server
getaddrinfo
socket
open_clientfd
getaddrinfo
socket
bind
listen
open_listenfd
连接请求
Connection
request
connect
accept
客户/
服务器
会话
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close
等待下一个客户
的连接请求
Await connection
request from
next client

套接字助手：open_listenfd
Sockets Helper: open_listenfd

- 创建侦听描述符，用于接受客户端的连接请求 Create a listening descriptor that can be used to accept connection requests from clients.

```
int open_listenfd(char *port)
{
    struct addrinfo hints, *listp, *p;
    int listenfd, optval=1;

    /* Get a list of potential server addresses */
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_socktype = SOCK_STREAM;              /* Accept connect. */
    hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG; /* …on any IP addr */
    hints.ai_flags |= AI_NUMERICSERV;             /* …using port no. */
    Getaddrinfo(NULL, port, &hints, &listp);
```

csapp.c

AI_PASSIVE意味着“我计划侦听这个套接字” AI_PASSIVE means “I plan to listen on this socket.”

AI_ADDRCONFIG正常情况下不用于服务器，但是出于方便使用它

AI_ADDRCONFIG normally not used for servers, but we use it for convenience

套接字助手：open_listenfd（续）
Sockets Helper: open_listenfd (cont)

```
/* Walk the list for one that we can bind to */
for (p = listp; p; p = p->ai_next) {
    /* Create a socket descriptor */
    if ((listenfd = socket(p->ai_family, p->ai_socktype,
                           p->ai_protocol)) < 0)
        continue;  /* Socket failed, try the next */

    /* Eliminates "Address already in use" error from bind */
    Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
               (const void *)&optval , sizeof(int));

    /* Bind the descriptor to the address */
    if (bind(listenfd, p->ai_addr, p->ai_addrlen) == 0)
        break; /* Success */
    Close(listenfd); /* Bind failed, try the next */
}
```

csapp.c

生产服务器不会在第一次成功时跳出循环，我们这样做只是为了简单 A production server would not break out of the loop on the first success. We do that for simplicity only.

套接字助手：open_listenfd（续）
Sockets Helper: open_listenfd (cont)

```
    /* Clean up */
    Freeaddrinfo(listp);
    if (!p) /* No address worked */
        return -1;

    /* Make it a listening socket ready to accept conn. requests */
    if (listen(listenfd, LISTENQ) < 0) {
        Close(listenfd);
        return -1;
    }
    return listenfd;
}
```

csapp.c

- 关键点：open_clientfd和open_listenfd两个都是与任何特定IP版本无关的 Key point: open_clientfd and open_listenfd are both independent of any particular version of IP.

使用telnet测试服务器
Testing Servers Using `telnet`

- **telnet程序对于测试通过互联网连接传输ASCII字符串的服务器非常有用 The `telnet` program is invaluable for testing servers that transmit ASCII strings over Internet connections**
  - 我们的简单回声服务器 Our simple echo server
  - Web服务器 Web servers
  - 邮件服务器 Mail servers

- 用法： **Usage:**
  - linux>telnet <主机> <端口号> **`linux>` *`telnet <host> <portnumber>`***
  - 创建与服务器的连接，该服务器运行在<host>上并侦听端口<portnumber> Creates a connection with a server running on ***`<host>`*** and listening on port ***`<portnumber>`***

用telnet测试回声服务器
Testing the Echo Server With `telnet`

```
whaleshark> ./echoserveri 15213
Connected to (MAKOSHARK.ICS.CS.CMU.EDU, 50280)
server received 11 bytes
server received 8 bytes



makoshark> telnet whaleshark.ics.cs.cmu.edu 15213
Trying 128.2.210.175...
Connected to whaleshark.ics.cs.cmu.edu (128.2.210.175).
Escape character is '^]'.
Hi there!
Hi there!
Howdy!
Howdy!
^]
telnet> quit
Connection closed.
makoshark>
```

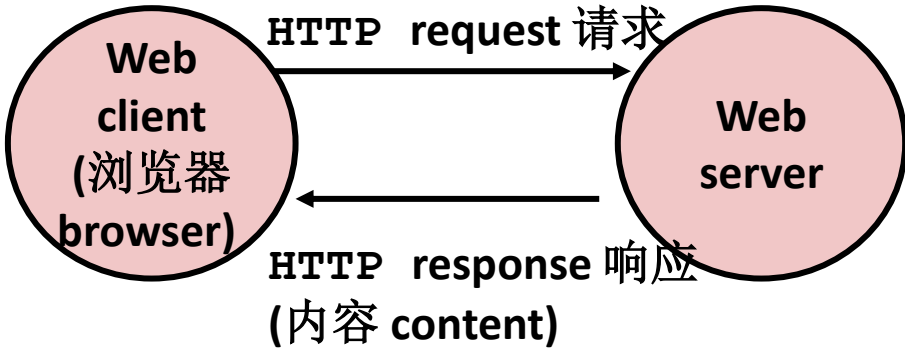
议题

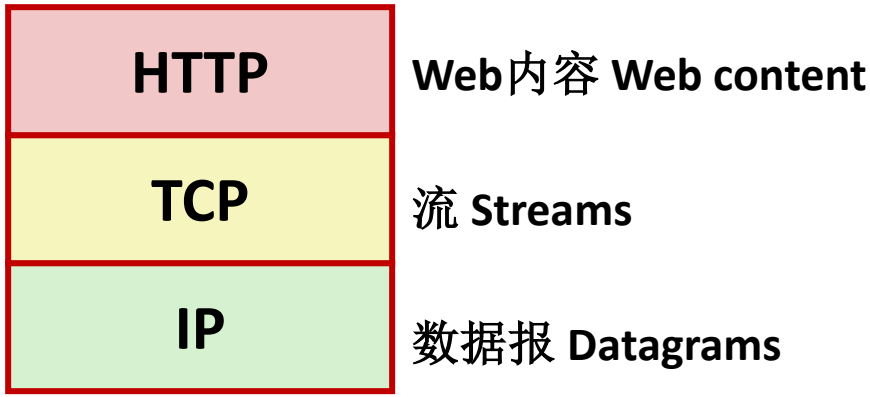
- 上次的问题 Questions from yesterday
- 上次没有讨论的内容 Material we didn't get to yesterday
  - 使用套接字发送数据 Transmitting data using sockets
  - 套接字地址 Socket addresses
  - `getaddrinfo`
- 建立连接 Setting up connections
- **应用层协议示例：HTTP　Application protocol example: HTTP**

Web服务器基础 Web Server Basics

- 客户和服务器使用超文本传送协议（HTTP）通信 **Clients and servers communicate using the HyperText Transfer Protocol (HTTP)**
  - 客户和服务器建立TCP连接 Client and server establish TCP connection
  - 客户请求内容 Client requests content
  - 服务器对请求内容进行响应 Server responds with requested content
  - 客户和服务器关闭连接（最终） Client and server close connection (eventually)
- **当前版本是HTTP/2.0，但是HTTP/1.1仍在广泛使用 Current version is HTTP/2.0 but HTTP/1.1 widely used still**
  - RFC 2616, June, 1999.

  http://www.w3.org/Protocols/rfc2616/rfc2616.html





Web内容 Web Content

- **Web服务器返回内容给客户 Web servers return *content* to clients**
  - *内容：*字节序列，具有相关联的MIME（多用途互联网邮件扩展）类型
    content: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

- **MIME类型示例 Example MIME types**
  - **`text/html`** HTML文档 HTML document
  - **`text/plain`** 无格式文本 Unformatted text
  - **`image/gif`** GIF格式编码的二进制图像 Binary image encoded in GIF format
  - **`image/png`** PNG格式编码的二进制图像 Binary image encoded in PNG format
  - **`image/jpeg`** JPG格式编码的二进制图像 Binary image encoded in JPEG format

完整的MIME类型列表网址：You can find the complete list of MIME types at:
`http://www.iana.org/assignments/media-types/media-types.xhtml`

静态和动态内容 Static and Dynamic Content

- **HTTP响应中返回的内容可以是*静态*的，也可以是*动态*的 The content returned in HTTP responses can be either *static* or *dynamic***
  - 静态内容：存储在文件中并响应HTTP请求检索的内容 *Static content*: content stored in files and retrieved in response to an HTTP request
    - 示例：HTML文件、图像、音频剪辑、Javascript程序 Examples: HTML files, images, audio clips, Javascript programs
    - 请求标识哪个内容文件 Request identifies which content file
  - 动态内容：响应HTTP请求而动态生成的内容 *Dynamic content*: content produced on-the-fly in response to an HTTP request
    - 示例：由服务器代表客户端执行的程序生成的内容 Example: content produced by a program executed by the server on behalf of the client
    - 请求标识包含可执行代码的文件 Request identifies file containing executable code
- **Web内容关联一个文件，该文件由服务器管理 *Web content associated with a file that is managed by the server***

统一资源定位符和客户及服务器如何使用 URLs and how clients and servers use them

- 文件的惟一名字：URL（统一资源定位符） Unique name for a file: URL (Universal Resource Locator)
- 示例URL： Example URL:
  `http://www.cmu.edu:80/index.html`
- 客户使用前缀(`http://www.cmu.edu:80`) 进行推理：
  Clients use *prefix* (`http://www.cmu.edu:80`) to infer:
  - 要联系的服务器（协议）类型 What kind (protocol) of server to contact (HTTP)
  - 服务器所在位置 Where the server is (**`www.cmu.edu`**)
  - 它正在侦听哪个端口 What port it is listening on (80)

统一资源定位符和客户及服务器如何使用 URLs and how clients and servers use them

- 服务器使用后缀（/index.html）： **Servers use *suffix* (`/index.html`) to:**
  - 确定请求是针对静态内容还是动态内容 Determine if request is for static or dynamic content.
    - 对此没有硬性规定 No hard and fast rules for this
    - 一种约定：可执行文件位于cgi-bin目录中 One convention: executables reside in **`cgi-bin`** directory
  - 在文件系统上查找文件 Find file on file system
    - 后缀中的首字母“/”表示所请求内容的主目录 Initial “**/**” in suffix denotes home directory for requested content.
    - 最小后缀为“/”，服务器扩展为配置的默认文件名（通常为index.html） Minimal suffix is “**/**”, which server expands to configured default filename (usually, **`index.html`**)

HTTP请求示例 HTTP Request Example

```
GET / HTTP/1.1
Host: www.cmu.edu
```

```
Client: request line
Client: required HTTP/1.1 header
Client: blank line terminates headers
```

- HTTP标准要求每个文本行以“\r\n”（回车换行）结尾 HTTP standard requires that each text line end with “\r\n”
- 空行（“\r\n”）终止请求和响应首部 Blank line (“\r\n”) terminates request and response headers

HTTP请求 HTTP Requests

- **HTTP请求是一个*请求行*，后跟零个或多个*请求首部* HTTP request is a *request line*, followed by zero or more *request headers***
- **请求行：方法 uri 版本 Request line: `<method> <uri> <version>`**
  - 方法是GET、POST、OPTIONS、HEAD、PUT、DELETE或TRACE之一 **`<method>`** is one of **`GET`**, **`POST`**, **`OPTIONS`**, **`HEAD`**, **`PUT`**, **`DELETE`**, or **`TRACE`**
  - uri通常是代理的URL，服务器的URL后缀 **`<uri>`** is typically URL for proxies, URL suffix for servers
    - URL是URI（统一资源标识符）的一种类型 A URL is a type of URI (Uniform Resource Identifier)
    - 参见 See http://www.ietf.org/rfc/rfc2396.txt
  - 版本是请求的HTTP版本（HTTP/1.0或HTTP/1.1） **`<version>`** is HTTP version of request (**`HTTP/1.0`** or **`HTTP/1.1`**)

HTTP请求 HTTP Requests

- **HTTP请求是一个*请求行*，后跟零个或多个*请求首部* HTTP request is a *request line*, followed by zero or more *request headers***
- **请求首部：首部名：首部数据 Request headers: `<header name>: <header data>`**
  - 向服务器提供其他信息 Provide additional information to the server

HTTP响应 HTTP Responses

- **HTTP响应是一个*响应行*，后跟零个或多个*响应首部*，可能后跟*内容*，并用空行（“\r\n”）分隔首部和内容 HTTP response is a *response line* followed by zero or more *response headers*, possibly followed by *content*, with blank line (“\r\n”) separating headers from content.**
- 响应行：**Response line:**
  版本 状态代码 状态消息 **<version> <status code> <status msg>**
  - 版本是响应的HTTP版本 <version> is HTTP version of the response
  - 状态码是数字状态 <status code> is numeric status
  - 状态消息是对应的英文文本 <status msg> is corresponding English text
    - **200 OK** 请求已正确处理 Request was handled without error
    - **301 Moved** 提供备用URL Provide alternate URL
    - **404 Not found** 服务器找不到文件 Server couldn’t find the file

HTTP响应 HTTP Responses

- **HTTP响应是一个*响应行*，后跟零个或多个*响应首部*，可能后跟*内容*，并用空行（“\r\n”）分隔首部和内容 HTTP response is a *response line* followed by zero or more *response headers*, possibly followed by *content*, with blank line (“\r\n”) separating headers from content.**
- **响应首部：首部名：首部数据 Response headers: <header name>: <header data>**
  - 提供有关响应的其他信息 Provide additional information about response
  - **Content-Type**：响应主体中内容的MIME类型 MIME type of content in response body
  - **Content-Length**：响应主体中内容的长度 Length of content in response body

示例HTTP事务 Example HTTP Transaction

```
whaleshark> telnet www.cmu.edu 80          Client: open connection to server
Trying 128.2.42.52...                      Telnet prints 3 lines to terminal
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET / HTTP/1.1                             Client: request line
Host: www.cmu.edu                          Client: required HTTP/1.1 header
                                           Client: blank line terminates headers
HTTP/1.1 301 Moved Permanently             Server: response line
Date: Wed, 05 Nov 2014 17:05:11 GMT        Server: followed by 5 response headers
Server: Apache/1.3.42 (Unix)               Server: this is an Apache server
Location: http://www.cmu.edu/index.shtml   Server: page has moved here
Transfer-Encoding: chunked                 Server: response body will be chunked
Content-Type: text/html; charset=...       Server: expect HTML in response body
                                           Server: empty line terminates headers
15c                                        Server: first line in response body
<HTML><HEAD>                               Server: start of HTML content
…
</BODY></HTML>                             Server: end of HTML content
0                                          Server: last line in response body
Connection closed by foreign host.         Server: closes connection
```

- HTTP标准要求每个文本行以“\r\n”（回车换行）结尾 HTTP standard requires that each text line end with “\r\n”
- 空行（“\r\n”）终止请求和响应首部 Blank line (“\r\n”) terminates request and response headers

示例HTTP事务，例2
Example HTTP Transaction, Take 2

```
whaleshark> telnet www.cmu.edu 80          Client: open connection to server
Trying 128.2.42.52...                      Telnet prints 3 lines to terminal
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET /index.shtml HTTP/1.1                  Client: request line
Host: www.cmu.edu                          Client: required HTTP/1.1 header
                                           Client: blank line terminates headers
HTTP/1.1 200 OK                            Server: response line
Date: Wed, 05 Nov 2014 17:37:26 GMT        Server: followed by 4 response headers
Server: Apache/1.3.42 (Unix)
Transfer-Encoding: chunked
Content-Type: text/html; charset=...
                                           Server: empty line terminates headers
1000                                       Server: begin response body
<html ..>                                  Server: first line of HTML content
…
</html>
0                                          Server: end response body
Connection closed by foreign host.         Server: close connection
```

示例HTTP(S)事务，例3 Example HTTP(S) Transaction, Take 3

```
whaleshark> openssl s_client www.cs.cmu.edu:443
CONNECTED(00000005)
…
Certificate chain
…
-
Server certificate
-----BEGIN CERTIFICATE-----
MIIGDjCCBPagAwIBAgIRAMiF7LBPDoySilnNoU+mp+gwDQYJKoZIhvcNAQELBQAw
djELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk1JMRIwEAYDVQQHEwlBbm4gQXJib3Ix
EjAQBgNVBAoTCUludGVybmV0MjERMA8GA1UECxMISW5Db21tb24xHzAdBgNVBAMT
wkWkvDVBBCwKXrShVxQNsj6J
…
-----END CERTIFICATE-----
subject=/C=US/postalCode=15213/ST=PA/L=Pittsburgh/street=5000 Forbes
Ave/O=Carnegie Mellon University/OU=School of Computer
Science/CN=www.cs.cmu.edu          issuer=/C=US/ST=MI/L=Ann
Arbor/O=Internet2/OU=InCommon/CN=InCommon RSA Server CA
SSL handshake has read 6274 bytes and written 483 bytes
…
>GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 12 Nov 2019 04:22:15 GMT
Server: Apache/2.4.10 (Ubuntu)
Set-Cookie: SHIBLOCATION=scsweb; path=/; domain=.cs.cmu.edu
... HTML Content Continues Below ...
```