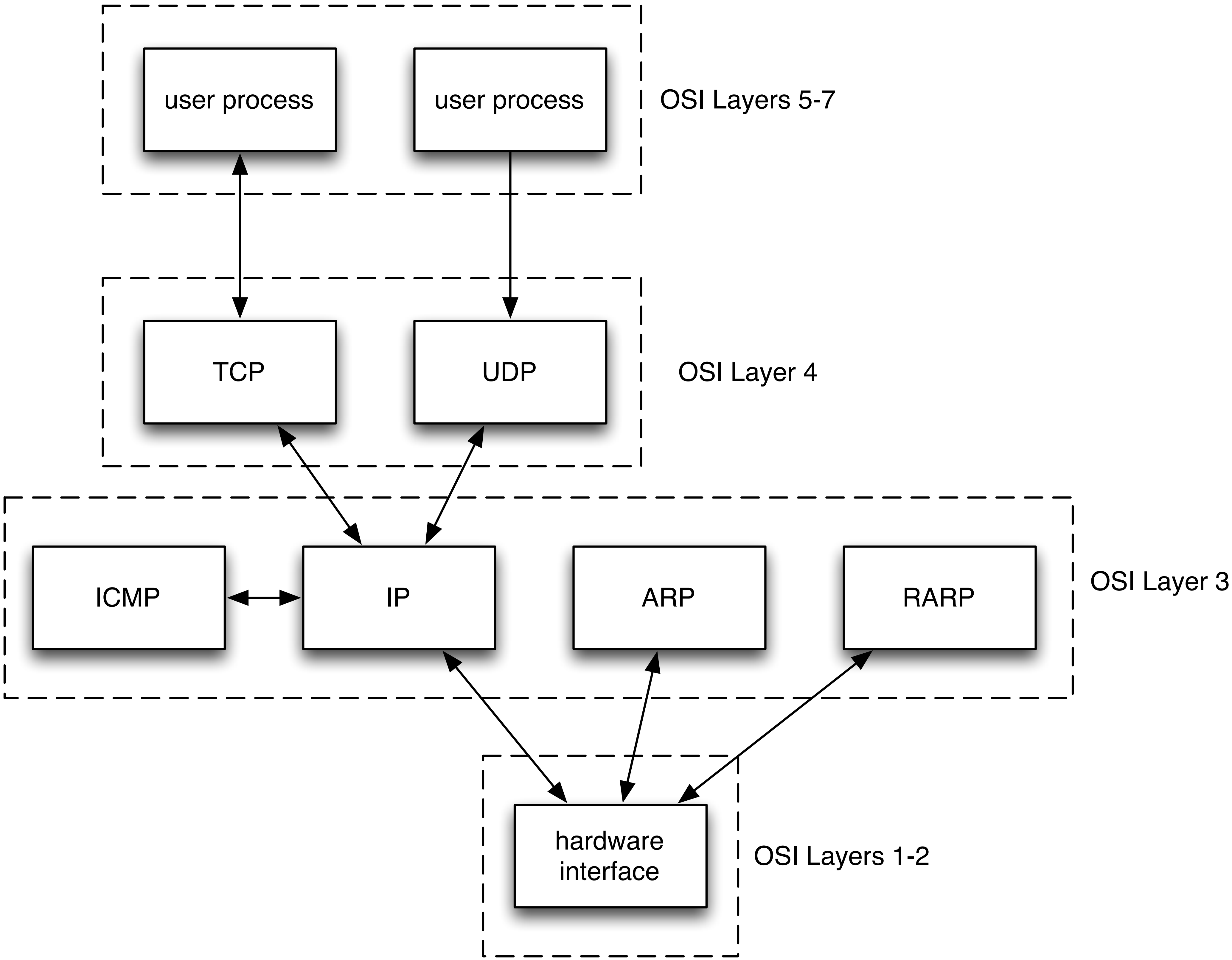
- Communication Protocols
  - TCP/IP protocol suite
    - the Internet protocols
  - Advanced Research Project Agency (ARPA) (1960s-1970s)
    - Department of Defense (DoD)
    - ARPANET
      - military, university and research sites
      - support computer science and military research projects

Communication protocols

- 1980s
  - DARPA Internet protocol suite
    - TCP/IP protocol suite
- ARPANET split in 1984
    - MILNET and ARPANET
- TCP/IP
  - not vendor specific
  - implemented in all range of computers
  - used for LAN’s and WAN’s
  - included in BSD Unix around 1982

Communication protocols

- Layering in the Internet protocol suite



Communication protocols

- *TCP: Transmission Control Protocol*
  - connection-oriented
  - reliable
  - full-duplex
- *UDP: User Datagram Protocol*
  - connectionless
  - unreliable
- *ICMP: Internet Control Message Protocol*
  - error and control information
  - gateways and hosts
  - usually generated by TCIP networking software

Communication protocols

- *IP: Internet Protocol*
  - packet delivery service for TCP, UDP and ICMP
- *ARP: Address Resolution Protocol*
  - maps an Internet address into a hardware address
- *RARP: Reverse Address Resolution Protocol*
  - maps a hardware address into an Internet address

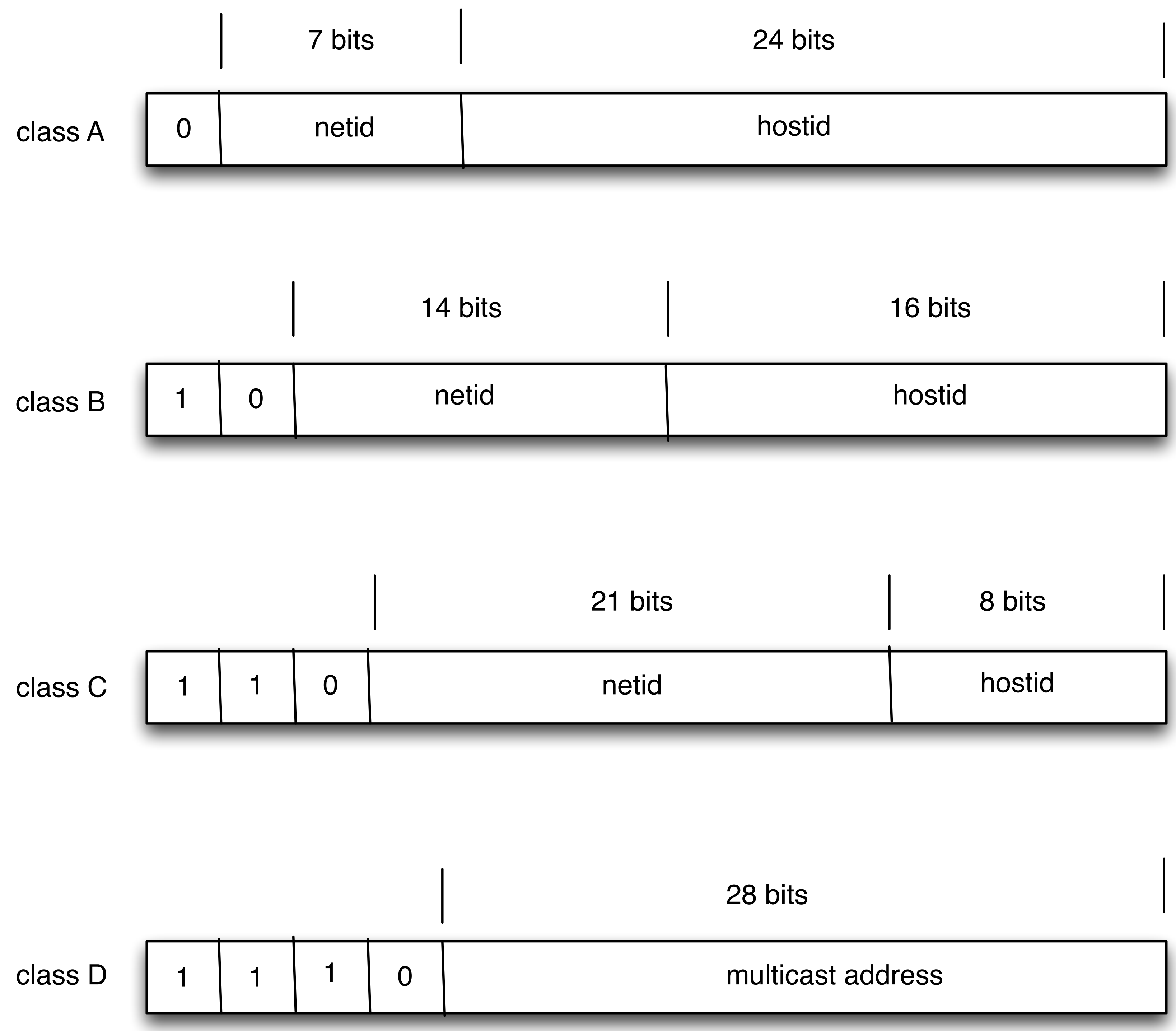
- Network Layer - IP
  - **IP Datagrams**
    - connectionless
      - considers each datagram independent of others
      - association must be provided at upper layers
      - each datagram contains source and destination address
    - unreliable delivery service
      - does not guarantee delivery
      - reliability must be provided at upper layers
      - computes and verifies checksum of header
      - discard datagrams with errors

Communication protocols

- handles routing through Internet
- fragments large datagrams
  - duplicates source and destination address in each packet
- reassembles at final destination
- elementary form of flow control
  - very fast arrivals
  - send ICMP message to source
    - informs TCP layer

Communication protocols

- **Internet Addresses**
  - 32 bit address
  - encodes
    - network id
    - host id
  - dot notation

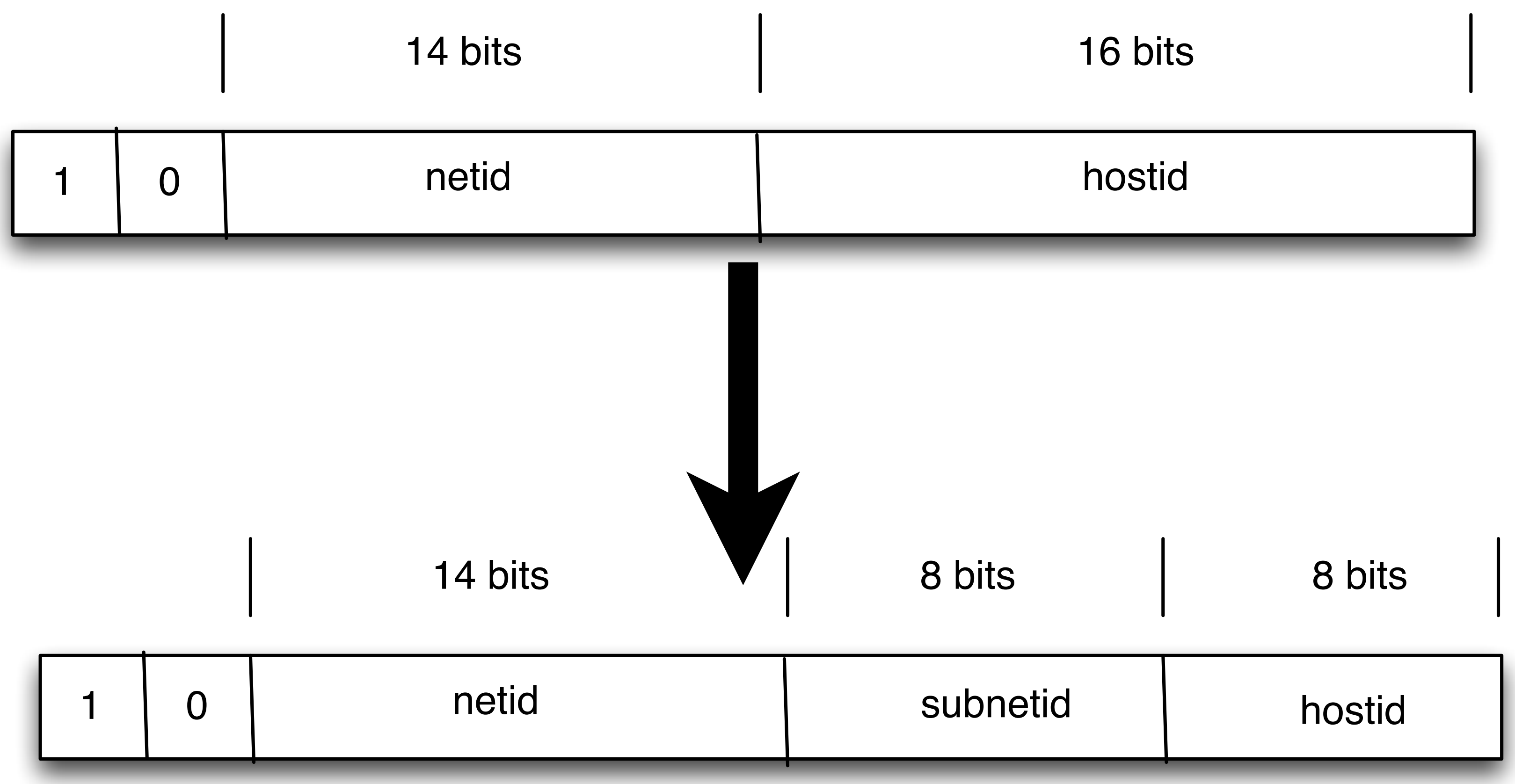


Communication protocols

- class A
  - lots of hosts
  - single network
- class C
  - more networks
  - fewer hosts per network
- every IP datagram contains source and destination address in header
  - network id used for routing
  - gateway needs no knowledge of host location

- **Subnet Addresses**
  - subdivide host address
    - example: 150 hosts in different networks
      - allocate host id 1 through 150 ignoring physical address
        - gateway must know which host is on which network
        - adding new hosts requires table updates
      - allocate some high-order bits from host id to subnets

Communication protocols



- subnets and mask

Mask: 255.255.255.0 (one network)
IP address: 192.168.1.20

11111111	11111111	11111111	00000000
192	168	1	00010100
192	168	1	00000000

network address 0

- subnets and mask

Mask: 255.255.255.248 (one network)
IP address: 192.168.1.20

11111111	11111111	11111111	11111000
192	168	1	00010100
192	168	1	00010000

network address 16

- CIDR notation
  - IP address and the prefix size
    - the number of leading 1 bits in the routing prefix mask

```
198.51.100.1/24  equivalent to subnet mask 255.255.255.0
has 8 bits for host field so 254 hosts per subnet

198.51.100.1/29  equivalent to subnet mask 255.255.255.248
has 3 bits for host field so 8 hosts per subnet
```

Communication protocols

CIDR	Host bits	Mask	Available addresses/ network	Number of subnetworks	Hosts per network
/24	8	255.255.255.0	256	1 subnet	254
/25	7	255.255.255.128	128	2 subnets	126
/26	6	255.255.255.192	64	4 subnets	62
/27	5	255.255.255.224	32	8 subnets	30
/28	4	255.255.255.240	16	16 subnets	14
/29	3	255.255.255.248	8	32 subnets	6
/30	2	255.255.255.252	4	64 subnets	2

- CIDR /26
  - 6 bits for host
  - mask 255.255.255.192
  - 64 IP addresses per sub net
  - 4 subnets
  - 62 hosts per network

Subnet range	Net address	Broadcast address	IP's for hosts
0-63	0	63	1-62
64-127	64	127	65-126
128-191	128	191	129-190
192-255	192	255	193-254

IP: 192.168.1.65
Calculate subnet net address

```
11111111 11111111 11111111 11000000
192          168          1     01000001
192          168          1     01000000 = 192.168.1.64 => subnet address
```

- **Address resolution**
  - types of addresses
    - 32 bit Internet address
      - dot notation address
      - unique per host
    - 48 bit Ethernet address
      - assigned by manufactures
      - unique for each interface card

- address resolution problems
  - know IP address
    - how does IP layer determine Ethernet address
      - *address resolution problem*
- diskless workstation at boot
  - OS can determine the Ethernet address
    - don’t want to embed IP address in code
  - how can the IP address be determined
      - *reverse address resolution problem*

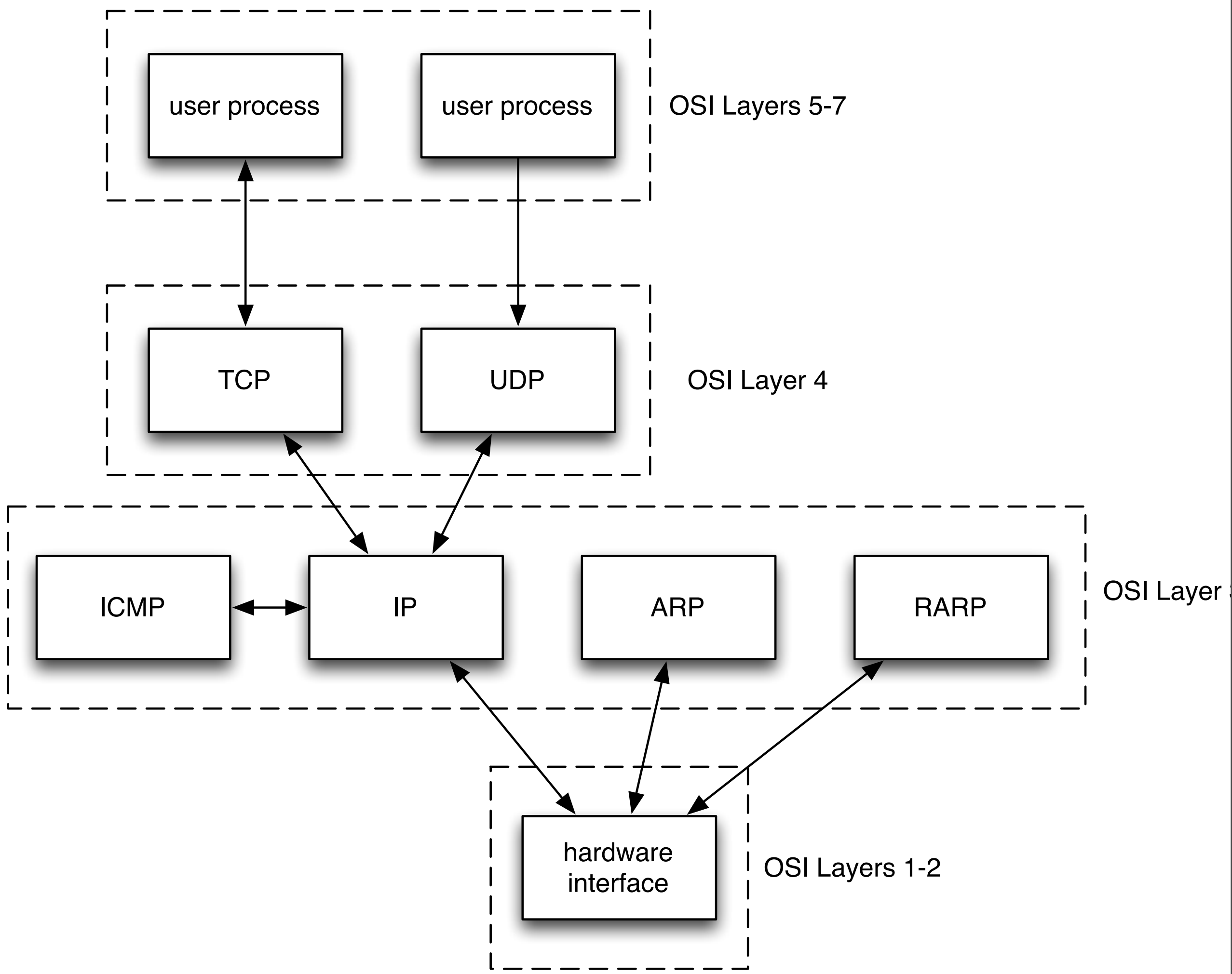
Communication protocols

- *address resolution problem*
  - Internet Address Resolution Protocol (ARP)
  - ask host with specific IP address to send Ethernet address
  - packet is broadcasted to the network
  - one host will respond
  - maintain a map of IP-Ethernet addresses

Communication protocols

- *reverse address resolution problem*
  - Reverse Internet Address Resolution Protocol (RARP)
  - RARP server
    - maintains a map of IP-Ethernet addresses
  - workstation broadcasts a message
    - contains Ethernet address
    - requests IP address

- Transport Layer - UDP and TCP
  - user process interacts with TCP/IP protocol
    - sending and receiving
      - TCP data
        - connection-oriented
        - reliable
        - full duplex
      - UDP data
        - connectionless
        - unreliable



Communication protocols

- comparison

	IP	UDP	TCP
connection-oriented ?	no	no	yes
message boundaries ?	yes	yes	no
data checksum ?	no	opt.	yes
positive ack. ?	no	no	yes
timeout and rexmit ?	no	no	yes
duplicate detection ?	no	no	yes
sequencing ?	no	no	yes
flow control ?	no	no	yes

Communication protocols

- TCP layer provides
  - reliability
  - establishment and termination of connections
  - sequencing of data
    - out of order
  - end-to-end reliability
    - checksum
    - positive acknowledgement
    - timeouts
  - end-to-end flow control

Communication protocols

- Port numbers
  - multiple processes using either UDP or TCP   FIGURE 4.14
  - need to associate data with process
  - ports
    - 16 bit integer port number
    - identifies server (service)
      - *well defined ports*
        - ftp port 21
        - ssh port 22
        - ...

Communication protocols

- file /etc/services

```
daytime         13/tcp      # Daytime (RFC 867)
...

ftp-data        20/udp      # File Transfer [Default Data]
ftp-data        20/tcp      # File Transfer [Default Data]
ftp             21/udp      # File Transfer [Control]
ftp             21/tcp      # File Transfer [Control]
#
ssh             22/udp      # SSH Remote Login Protocol
ssh             22/tcp      # SSH Remote Login Protocol
#
telnet          23/udp      # Telnet
telnet          23/tcp      # Telnet
#
                24/udp      # any private mail system
                24/tcp      # any private mail system
#
smtp            25/udp      # Simple Mail Transfer
smtp            25/tcp      # Simple Mail Transfer
...
```

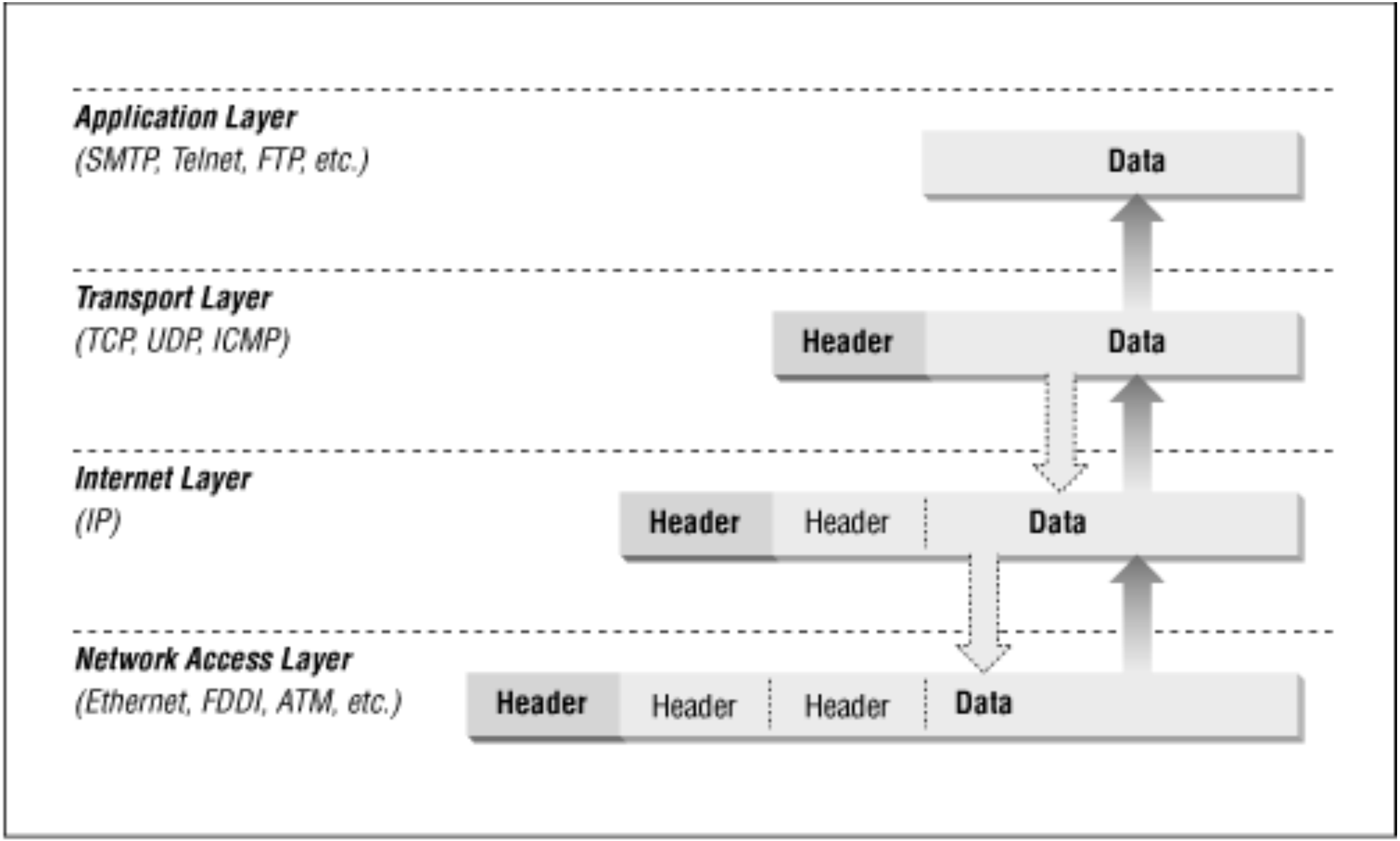
Communication protocols

- ftp example
  - client requests an unused port from TCP
    - *`ephemeral port number`*
      - 1-255 are reserved
      - BSD reserves 1-1023 (privileged processes)
  - client connects to server via port 21
  - server knows
    - IP address of client
    - port used by client

Communication protocols

- Ethernet header
  - 48-bit source address
  - 48-bit destination address
- IP datagram
  - source and destination IP address
  - protocol identifier
- UDP and TCP header
  - source port number
  - destination port number
  - UDP and TCP ports are independent

Communication protocols



Tuesday, October 30, 12

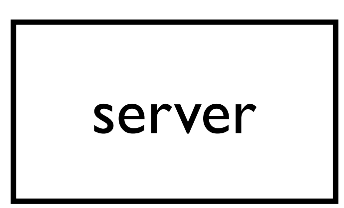
Communication protocols

- 5-tuple association
  - the protocol (TCP or UDP)
  - local host’s IP address
  - local port number
  - foreign host’s IP number
  - foreign port number

```
{ tcp, 128.10.0.3, 1500, 128.10.0.7, 21 }
```

Communication protocols

- Concurrent servers
  - spawned child continues to use the well-known port
  - sequence of actions
    - server is started on host orange
    - opens well-known port 21
    - waits for client request

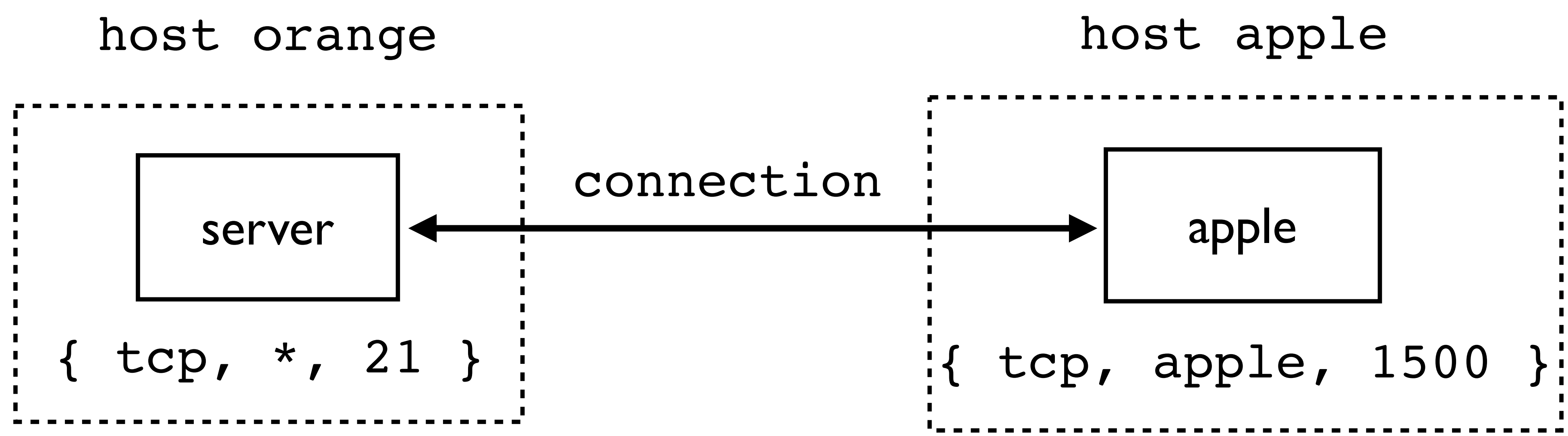


`{tcp, *, 21 }`

server waiting for TCP connection on any network, on port 21

Communication protocols

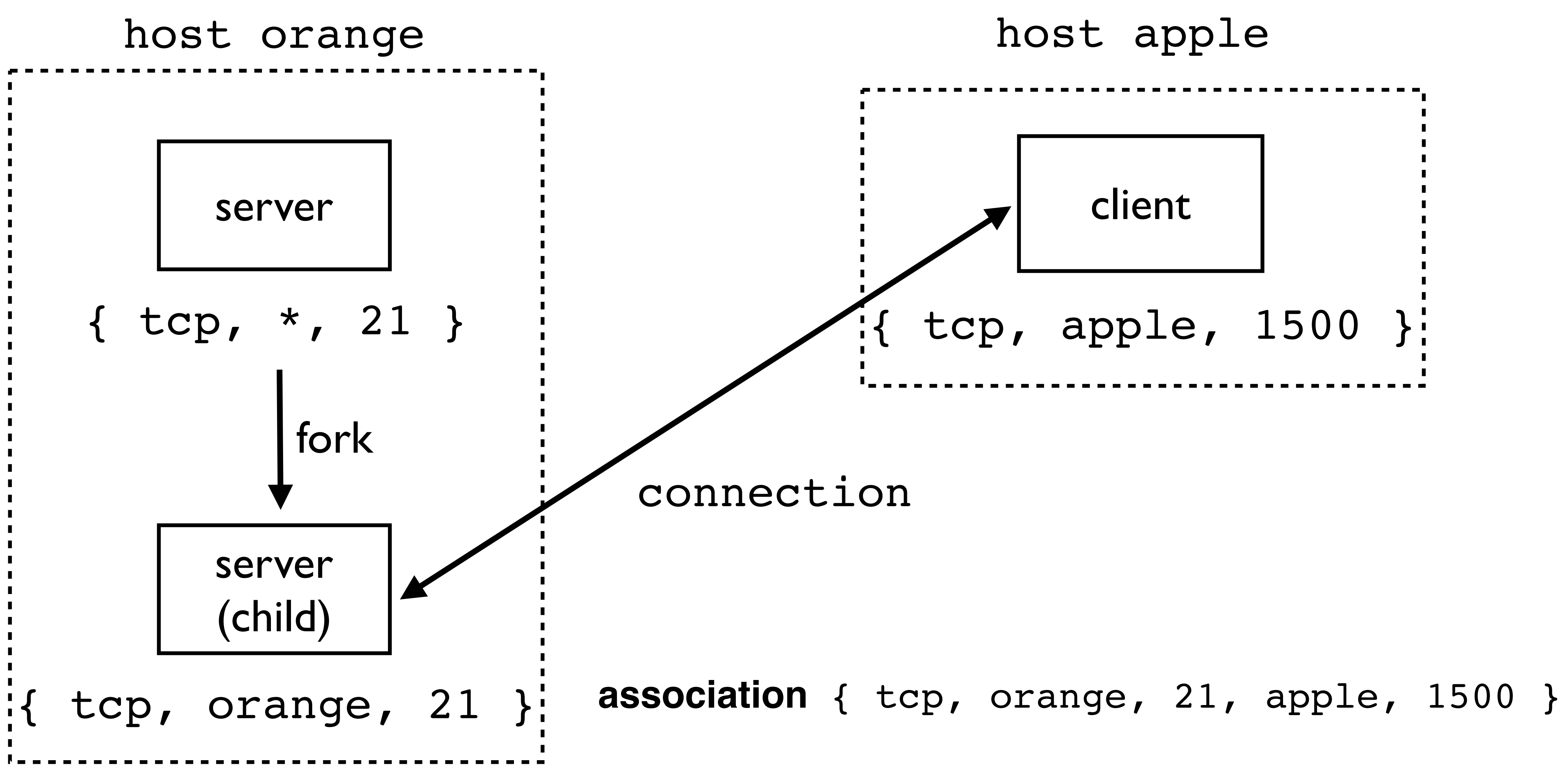
- Client `apple` starts up
  - executes an active open to the server
  - `ephemeral` port number 1500 assigned by TCP



`{ tcp, apple, 1500 }` clients half association or **`socket`**

Communication protocols

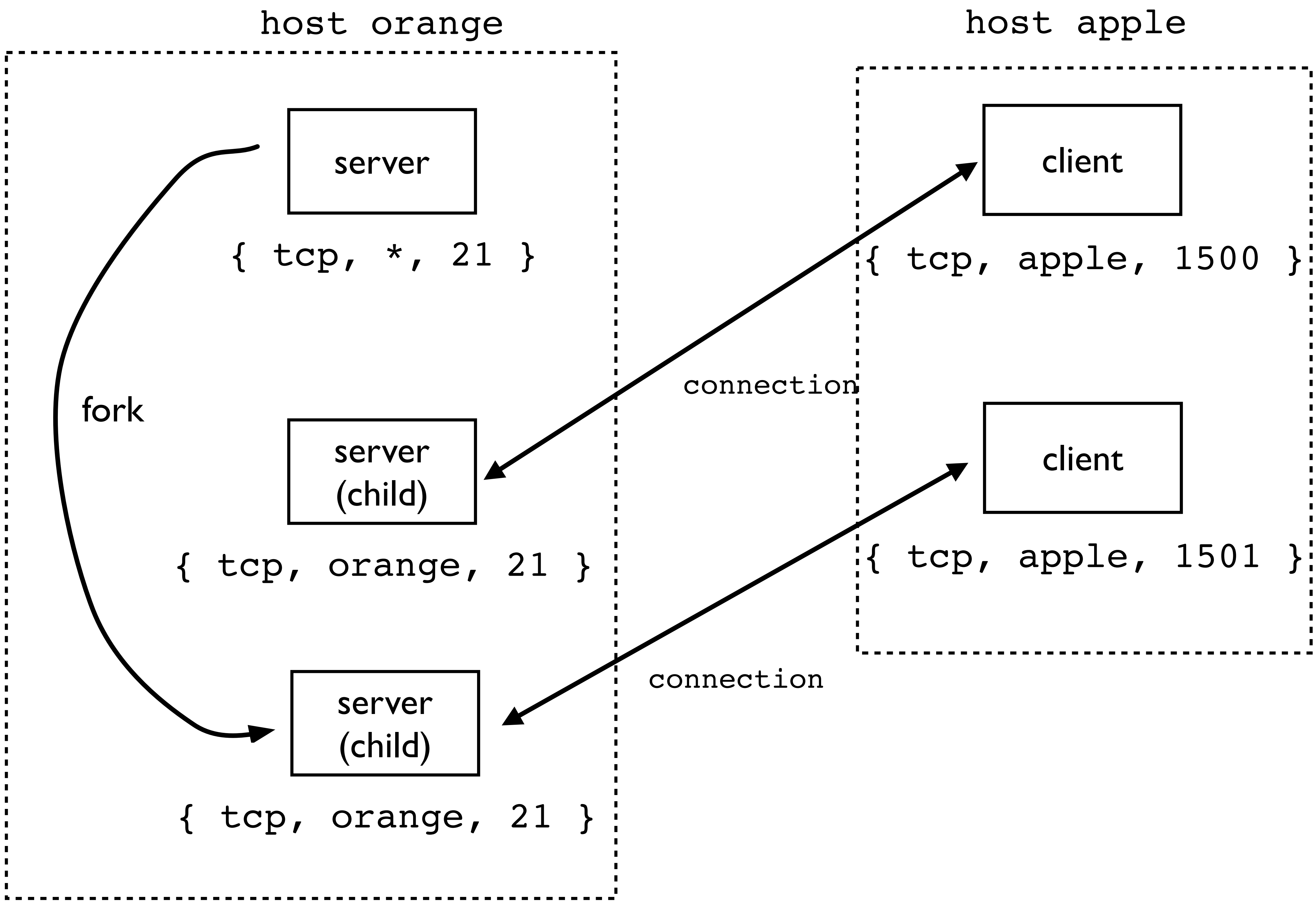
- server receives clients connection request
- it forks a process
- passes the connection to the child process
- server returns to its wait loop



- chile handles request

Communication protocols

- another client on `apple` requests a connection
- ephemeral port 1501



- two identical half associations
- two complete unique associations

associations

```
{ tcp, orange, 21, apple, 1500 }
{ tcp, orange, 21, apple, 1501 }
```

- TCP module on orange is able to determine which server child is to receive data message
  - based on source IP and source port number

- Ethernet header

Ethernet 802.3 Packet format.

<table>
<tr><th>00</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="16">Destination Address</td></tr>
<tr><td colspan="16">Source Address</td></tr>
<tr><td colspan="16">EtherType</td></tr>
<tr><td colspan="16">Data
:::</td></tr>
</table>

Destination Address. 6 bytes.
The address(es) are specified for a unicast, multicast (subgroup), or broadcast (an entire group).

Source Address. 6 bytes.
The address is for a unicast (single computer or device).

EtherType. 16 bits.
Which upper layer protocol will utilized the Ethernet frame.

Data. variable, 46-1500 bytes.

- IP header

MAC header	IP header	Data :::

IP header:

<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="8">Differentiated Services</td><td colspan="16">Total length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment offset</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">Protocol</td><td colspan="16">Header checksum</td></tr>
<tr><td colspan="32">Source IP address</td></tr>
<tr><td colspan="32">Destination IP address</td></tr>
<tr><td colspan="32">Options and padding :::</td></tr>
</table>

TTL, Time to Live. 8 bits.
A timer field used to track the lifetime of the datagram. When the TTL field is decremented down to zero, the datagram is discarded.

Protocol. 8 bits.
This field specifies the next encapsulated protocol.

Header checksum. 16 bits.
A 16 bit one's complement checksum of the IP header and IP options.

Source IP address. 32 bits.
IP address of the sender.

Destination IP address. 32 bits.
IP address of the intended receiver.

- TCP header

MAC header	IP header	TCP header	Data :::

TCP header:

<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td colspan="32">Sequence Number</td></tr>
<tr><td colspan="32">Acknowledgment Number</td></tr>
<tr><td colspan="4">Data Offset</td><td colspan="3">reserved</td><td colspan="3">ECN</td><td colspan="6">Control Bits</td><td colspan="16">Window</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">Urgent Pointer</td></tr>
<tr><td colspan="32">Options and padding :::</td></tr>
<tr><td colspan="32">Data :::</td></tr>
</table>

Source Port. 16 bits.

Destination Port. 16 bits.

Sequence Number. 32 bits.
The sequence number of the first data byte in this segment. If the SYN bit is set, the sequence number is the initial sequence number and the first data byte is initial sequence number + 1.

Acknowledgment Number. 32 bits.
If the ACK bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

- Berkeley Sockets (Internet Sockets)
  - API's
    - Berkeley sockets
    - System V Transport Layer Interface (TLI)
  - Berkeley sockets
    - network I/O differs from file I/O
      - file I/O
        - `open creat, close, read, write` and `lseek`
    - same principal
    - different semantics
    - network I/o is more involved

Communication protocols

- network I/O
  - client-server relation is not symmetrical
    - client and server have different roles
  - different types of connections
    - connection-oriented
      - more like I/O
      - once connection is established, the network I/O on that connection is with the same peer
    - connectionless
      - no connection is open
      - every network I/O can be with different process

Communication protocols

- more parameters are required
  - association

```
{ protocol, local-addr, local-process,
                    foreign-addr, foreign-process }
```

  - parameters may differ for different protocols
- network interface must support multiple communications protocols
  - different address scheme

Communication protocols

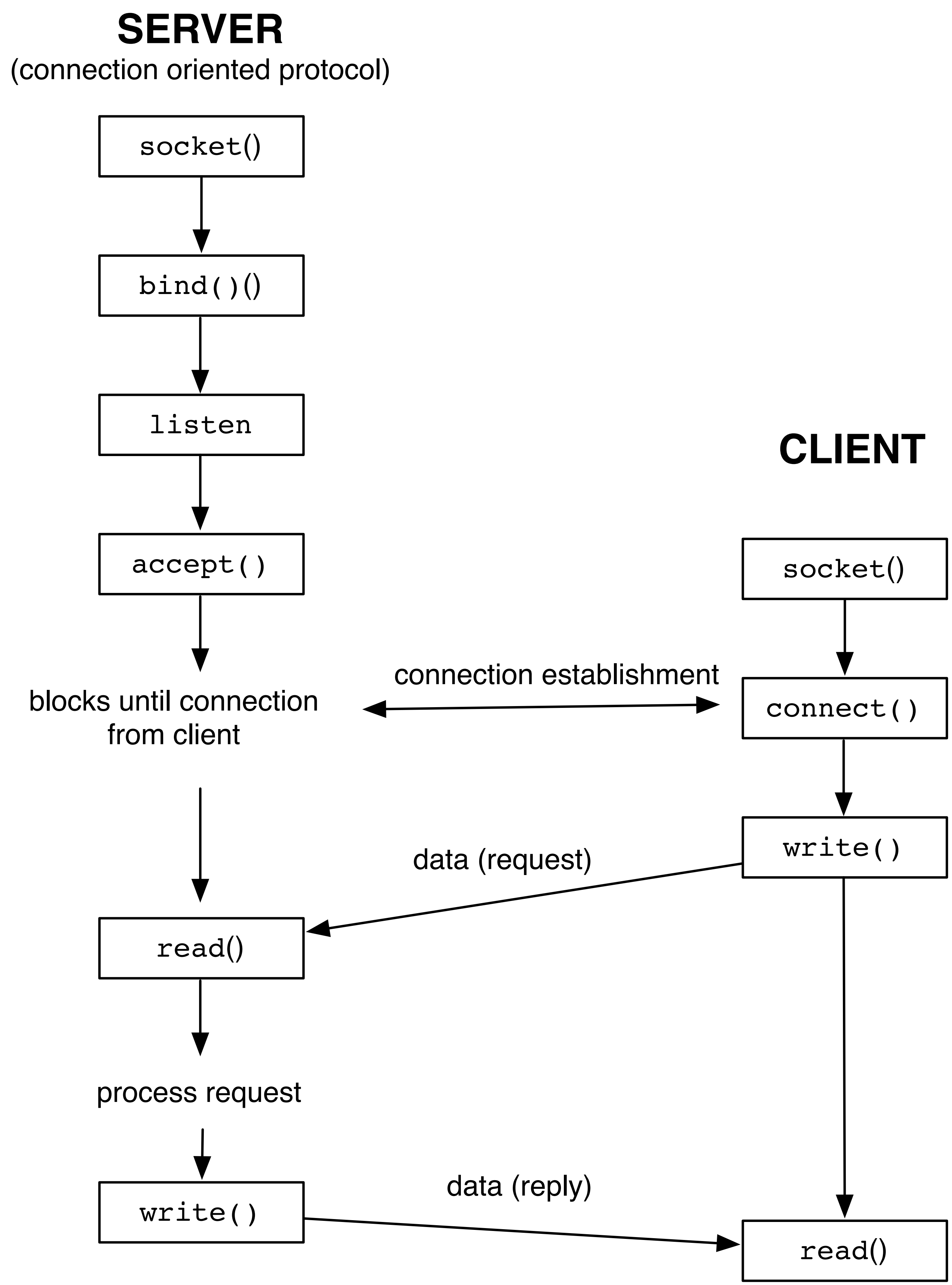
- specify type of process
  - server
- type of protocol
  - connection-oriented
  - connectionless
- type of server
  - concurrent server
  - iterative server

	iterative server	concurrent server
connection-oriented	infrequent	typical
connectionless-oriented	typical	infrequent

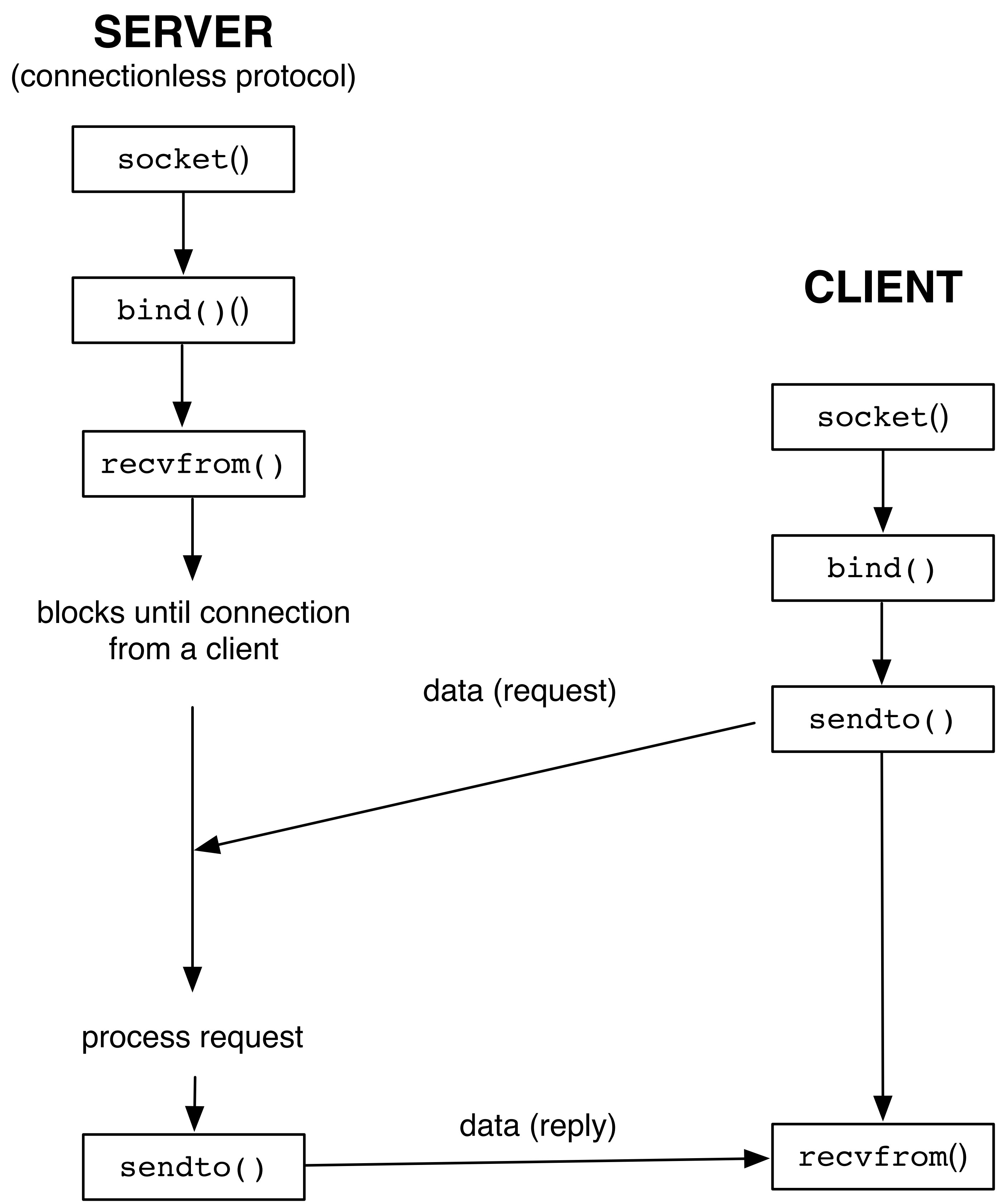
Communication protocols

- Overview
  - 4.1cBSD on VAX circa 1982
    - Unix domain
    - Internet domain (TCP/IP)
    - Xerox NS domain

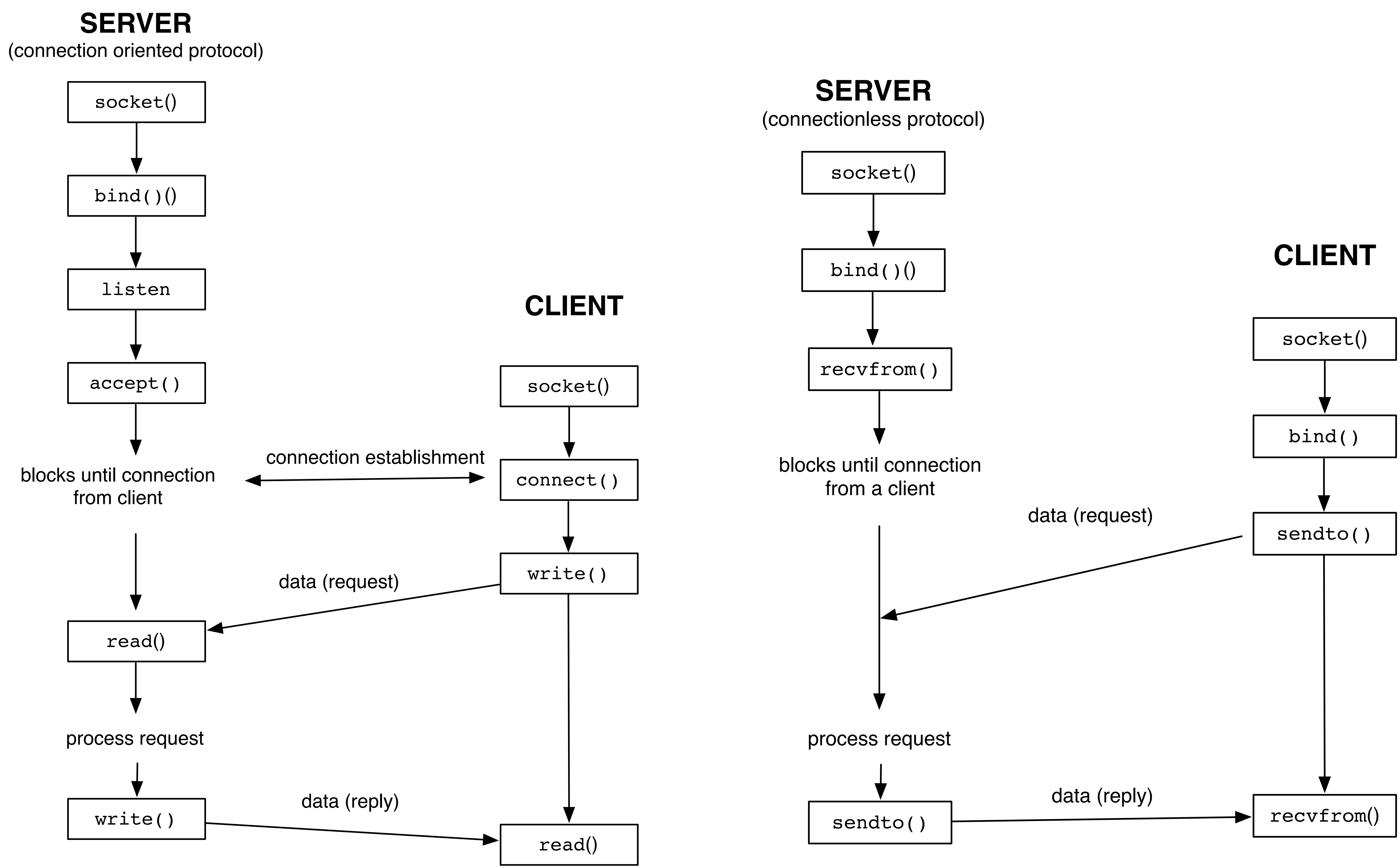
Communication protocols



Communication protocols



Communication protocols



- Unix Domain Protocol
  - communicate among processes on the same Unix system
  - a form of IPC built into networking system
  - provides
    - connection-oriented
    - connectionless
  - reliable
    - exists within the kernel
    - no transmission over external facilities

Communication protocols

- name space uses pathnames
- sample association

```
{unixstr, 0, /tmp/log.1, 0, /dev/logfile}
```

- used for testing
  - test locally before deploying to network

- Socket Addresses
- many system calls require a pointer to a socket address structure
- defined in `<sys/socket.h>`

```
struct sockaddr (
   u_short   sa_family          /*  address family:  AF_xxx value  */
   char      sa_data[14];       /*  up to 14 bytes of protocol-specific address  */
};
```

- 14 bytes protocol-specific address are interpreted according to the type of address

- 14 bytes protocol-specific address for Internet family

```
struct in_addr (
   u_long   s_addr            /*  32 bit netid/hostid   */
                              /*  network byte ordered  */
};

struct sockaddr_in {
    short     sin_family;               /*  AF_INET  */
    u_short   sin_port;                 /*  16-bit port number */
    struct    in_addr     sin_addr      /*  32-bit netid/hostid   */
                                        /*  network byte ordered  */
    char      sin_zero[8];              /*  unused  */
};
```

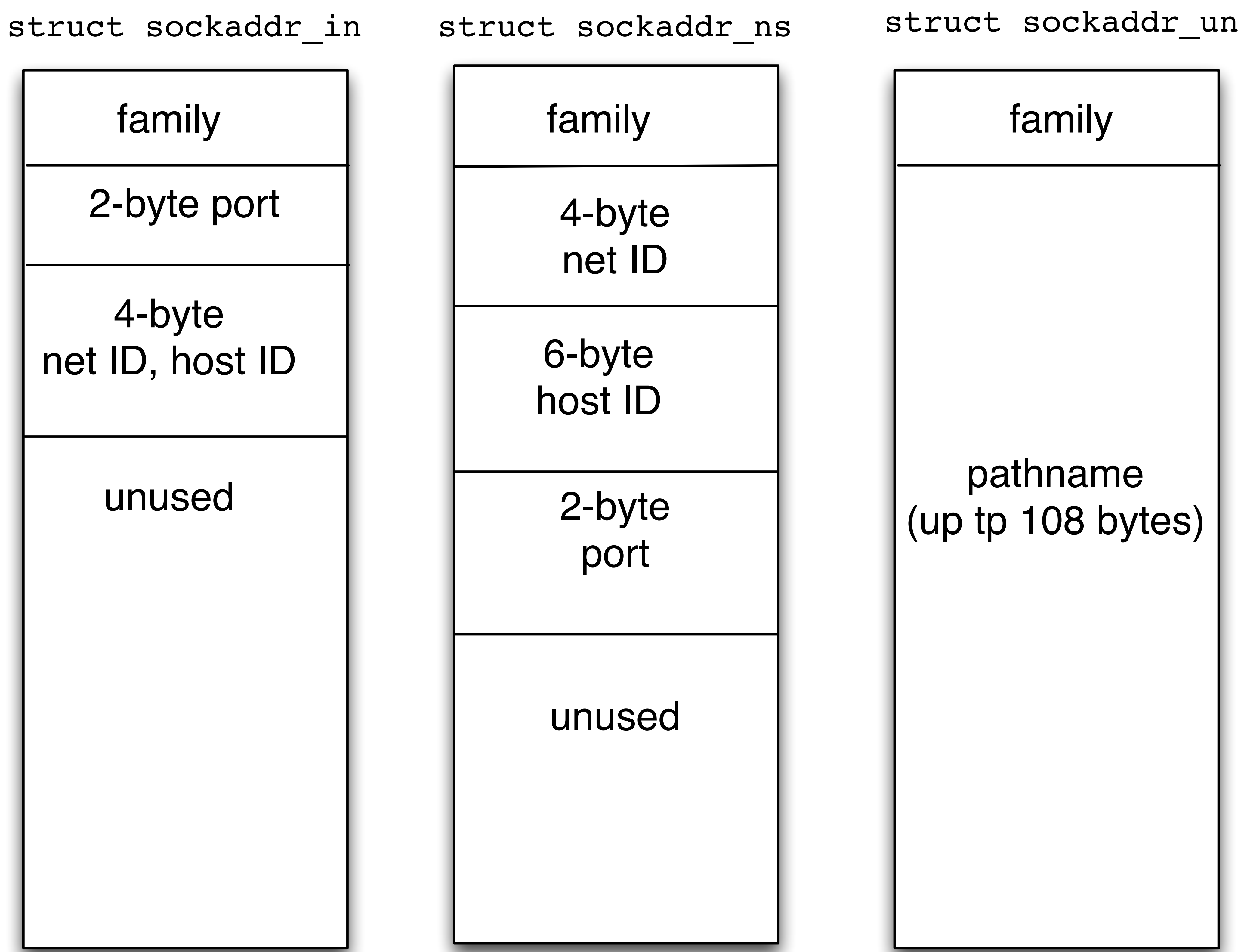
Communication protocols

- unsigned data types

C Data type	BSD	System V
`unsigned char`	`u_char`	`unchar`
`unsigned short`	`u_short`	`ushort`
`unsigned int`	`u_int`	`uint`
`unsigned long`	`u_long`	`ulong`

- defined in `<sys/types.h>`

- handle different size of address structure



Communication protocols

- calls like connect and bind work with any of supported domains
- must pass any socket address structure
  - `sockaddr_ns`
  - `sockaddr_ns`
  - `sockaddr_ns`
- pass address of protocol-specific structure as an argument casting the pointer into a pointer to a generic `sockaddr` structure

```
struct sockaddr_in  server_addr;  /* Internet-specific addr struct */
...
(fill in Internet-specific information )
...
connect ( sockfd, (struct sockaddr * ) &serv_addr, sizeof(serv_addr));
```

- Elementary Socket System Calls
  - `socket`
  - `socketpair`
  - `bind`
  - `connect`
  - `listen`
  - `accept`
  - `send, sendto, recv, recvfrom`
  - `close`

Communication protocols

- `socket` System Call
  - specifies type of communication protocol
    - Internet TCP
    - Internet UDP
    - XNS
    - ...

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

Communication protocols

```
int socket(int family, int type, int protocol);
```

- family:

`AF_UNIX`	Unix internal protocols
`AF_INET`	Internet protocols
`AF_NS`	Xerox NS protocols
`AF_IMPLINK`	IMP link layer *(Interface Message Processor)*

- type:

`SOCK_STREAM`	stream socket
`SOCK_DGRAM`	datagram socket
`SOCK_RAW`	raw socket
`SOCK_SEQPACKET`	sequenced packet socket

- socket family and type combinations

	AF_UNIX	AF_INET	AF_NS
`SOCK_STREAM`	Yes	TCP	SPP
`SOCK_DGRAM`	Yes	UDP	IDP
`SOCK_RAW`		IP	Yes
`SOCK_SEQPACKET`			SPP

- protocol
  - typically 0
  - specialized applications may require a specific protocol value

Communication protocols

```
int socket(int family, int type, int protocol);
```

- returns an integer
  - `sockfd`
- similar to a file descriptor
- specifies one element of the association

```
{ protocol, local-addr, local-process,
                    foreign-addr, foreign-process }
```

- other elements must be filled

Communication protocols

- system calls and association elements

	protocol	local-addr, local-process	foreign-addr, foreign-process
connection-oriented server	socket()	bind()	listen(), accept()
connection-oriented client	socket()	connect()	
connectionless server	socket()	bind()	recvfrom()
connectionless client	socket()	bind()	sendto()

- `socketpair` System Call
  - only for Unix domain

```
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int family, int type, int protocol, int sockvec[2]);
```

- returns 2 socket descriptors
  - `sockvec[0]` and `sockvec[1]`
  - bidirectional
  - stream pipes
- similar to pipes

Communication protocols

- `bind` System Call
  - assigns a name to an unnamed socket

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

- there are 3 uses of `bind`
  - server registers well-know address with the system
    - “this is my address, messages received for this address are mine”
    - for connection-oriented and connectionless
  - a client registers a specific address for itself
  - a connectionless client needs to assure that the system assigns it some unique address
    - at other end server has a valid return address

    - `bind` fills *local-addr* and *local-process* of association 5-tuple

Communication protocols

- `connect` System Call
  - establish a connection with a server

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
```

- conversions might take place
  - buffer size
  - amount of data to exchange between acknowledges
- does not return until connection is established or error occurs
- if client does not call `bind`
  - `connect` fills *local-addr* , *local-process, foreign-addr* , *foreign-process,* of association 5-tuple

Communication protocols

- connectionless clients can call connect
  - will only store *servaddr* for suture sends
    - no need to specify destination adress on every datagram
    - can use `read, write, recv` and `send` system calls
  - will return immediately
  - no exchange of messages

- `accept` System Call
  - connection oriented server
  - accepts connections

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *peer, int *addrlen);
```

- blocks until a requests arrives
- takes first connection request and creates another socket with same properties as *sockfd*
- *peer* and *addrlen* store address and size of address of the client

- typical scenario - concurrent server

```
int sockfd, newsockfd;

if ( ( sockfd = socket( ... ) ) < 0 )
  err_sys("socket error");
if ( bind(sockfd, ... )  < 0 )
  err_sys("bind error");
if ( listen(sockfd, 5 ) < 0 )
  err_sys(""listen error");

for ( ; ; ) {
    newsockfd = accept(sockfd, ...)  /* blocks  */
    if (newsockfd < 0 )
      err_sys("accept error");

    if ( fork() == 0 ) {
      close(sockfd);  /* child */
      doit(newsockfd);  /* process request */
      exit(0);
   }
close(newsockfd);
}
```

Communication protocols

- all elements of 5-tuple association have been filled for newsockfd after return from `accept`
- `sockfd` argument passed to accept has only thre elements filled
  - `foreign-addr` and `foreign-process` are unspecified
    - original process can `accept` another connection using same `sockfd`

- typical scenario - iterative server

```
int sockfd, newsockfd;

if ( ( sockfd = socket( ... ) ) < 0 )
  err_sys(“socket error”);
if ( bind(sockfd, ... )  < 0 )
  err_sys(“bind error”);
if ( listen(sockfd, 5 ) < 0 )
  err_sys(“”listen error”);

for ( ; ; ) {
    newsockfd = accept(sockfd, ...)  /* blocks  */
    if (newsockfd < 0 )
      err_sys(“accept error”);

    doit(newsockfd);  /* process request */
    close(newsockfd);
}
```