UDP socket programming

Sigurd Eskeland

Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

Request-response

client process: process that initiates communication using a request

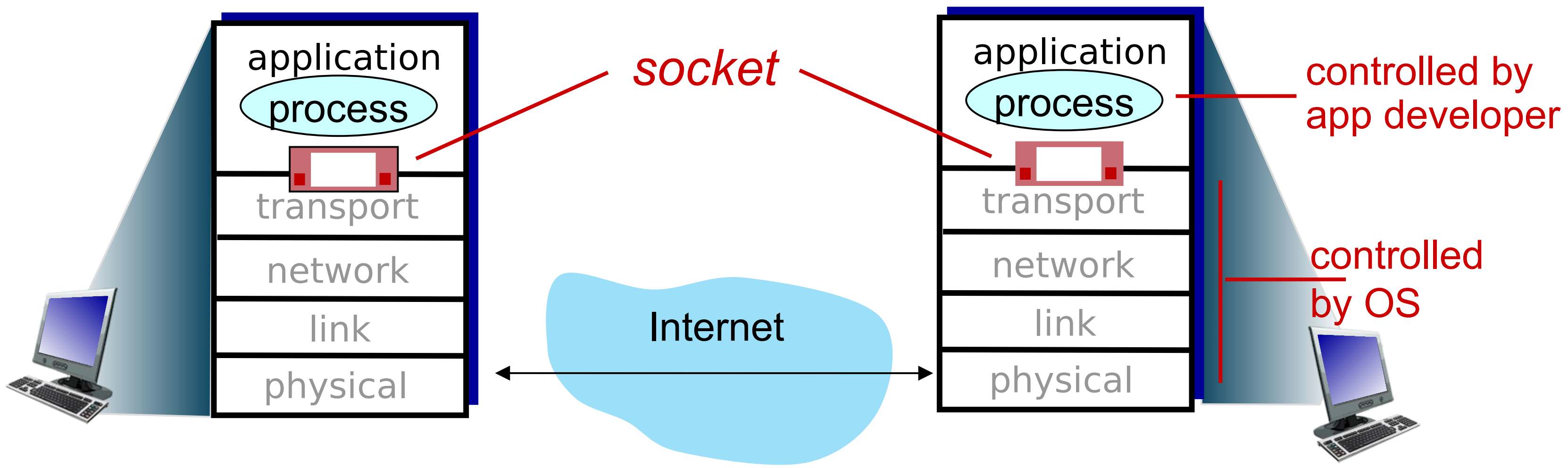
server process:

- process that waits to be contacted
- sends a response to the client

- note: applications with P2P architectures have client processes & server processes

Sockets

1. Process sends/receives messages to/from its socket
   - Two sockets involved: one on each side
2. A socket is analogous to a *door* between network application process and the transport protocol
   - The sending process shoves the message out the "door"
   - *Not the same as port number*; sockets *use* port numbers



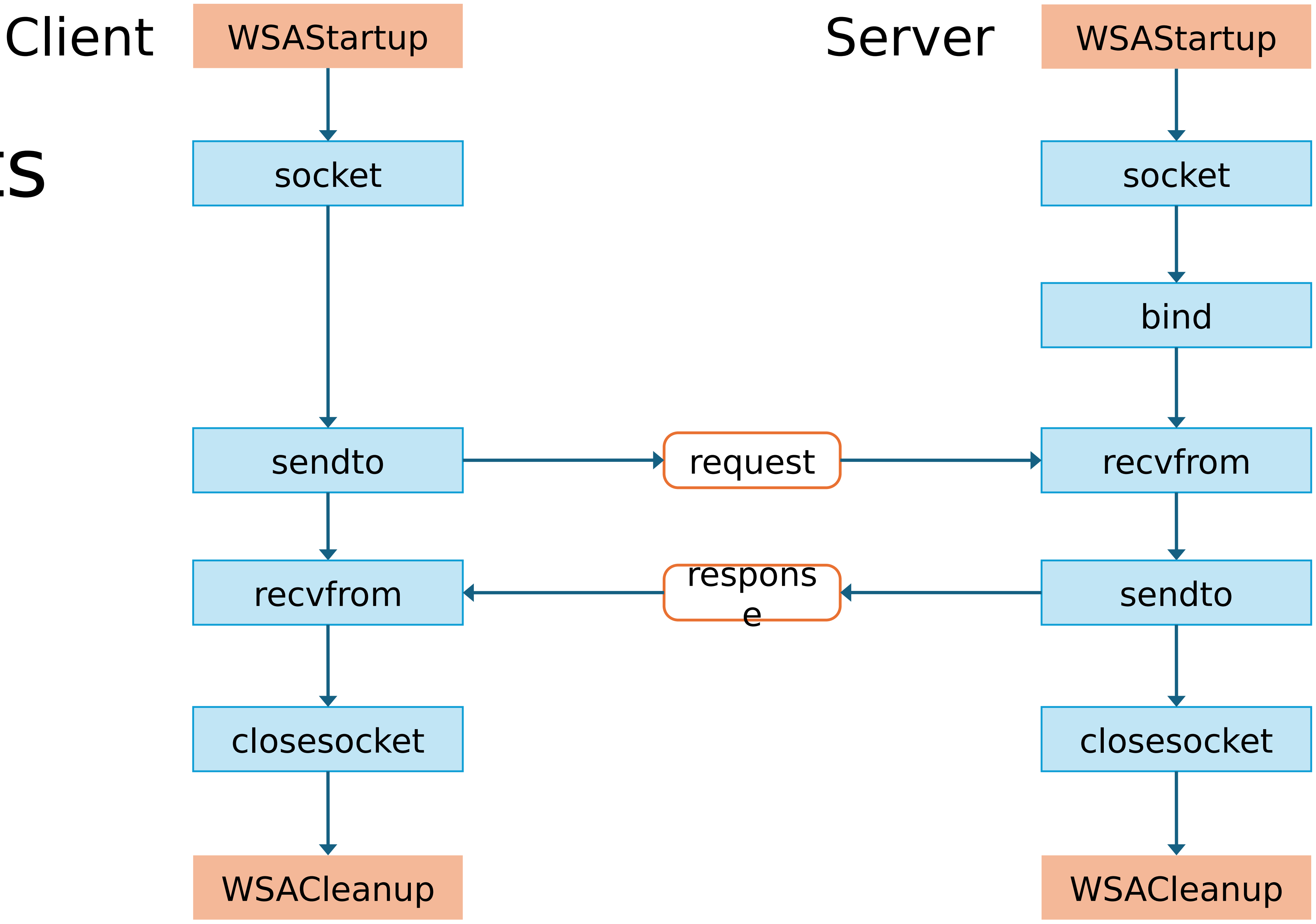
What is a socket?

1. A network socket is a *data structure* that serves as an endpoint for sending and receiving data across a network
2. Sockets are defined by an application programming interface (API)
3. A socket is accessed by a “handle” ☾ socket file descriptor
4. When creating a socket instance, the transport layer protocol (TCP or UDP) is specified

What is a socket?

5. If a program shall receive packets, the program must *bind* the socket to a specific port number
    - So that the OS can deliver incoming data packets to the right application process
6. A socket that has received a packet, then knows the sender’s IP address and port number
    - Needed for sending a response to the right sending process

UDP sockets


Client
WSAStartup
socket
sendto
recvfrom
closesocket
WSACleanup
request
respons
e
Server
WSAStartup
socket
bind
recvfrom
sendto
closesocket
WSACleanup

UDP server

```
#include <winsock2.h>

WSADATA wsa;
SOCKET socketServer;
```

Holds address of a received message →

```
struct sockaddr_in addressServer;
struct sockaddr_in addressReceived;
```

Start use of Winsock DLL (*Ws2_32.dll*) →

```
WSAStartup(MAKEWORD(2,2), &wsa);
```

Create UDP-socket (IPv4) →

```
socketServer = socket(AF_INET,  SOCK_DGRAM, 0 );
```

Any address →

Address is IPv4 →

Specify port number →

```
// Initialize address structure
addressServer.sin_addr.s_addr = INADDR_ANY;
addressServer.sin_family = AF_INET;
addressServer.sin_port = htons(8888);   // LSB -> MSB
```

assign addressServer to the socket →

```
bind(socketServer, (struct sockaddr*)& addressServer,
                                        sizeof(addressServer));
```

get size of address structure →

```
int nFromlen = sizeof(addressReceived);
```

Wait for a message: capture string and →

```
recvfrom(socketServer, sReceivedString, STR_SIZE, 0,
                  (struct sockaddr*)& addressReceived, &
```

UDP server

	`WSAStartup(MAKEWORD(2,2), &wsa);`
	`socketServer = socket(AF_INET , SOCK_DGRAM , 0 );`
	`// Initialize address structure`
Any address	`addressServer.sin_addr.s_addr = INADDR_ANY;`
Address is IPv4	`addressServer.sin_family = AF_INET;`
Specify port number	`addressServer.sin_port = htons(8888); );   // LSB -> MSB`
assign addressServer to the socket	`bind(socketServer, (struct sockaddr*)& addressServer, sizeof(addressServer));`
get size of address structure	`int nFromlen = sizeof(addressReceived);`
Wait for a message: capture string and address	`recvfrom(socketServer, sReceivedString, STR_SIZE, 0, (struct sockaddr*)& addressReceived, & nFromlen);`

```
sendto( … );

closesocket(socketServer);

WSACleanup();
```

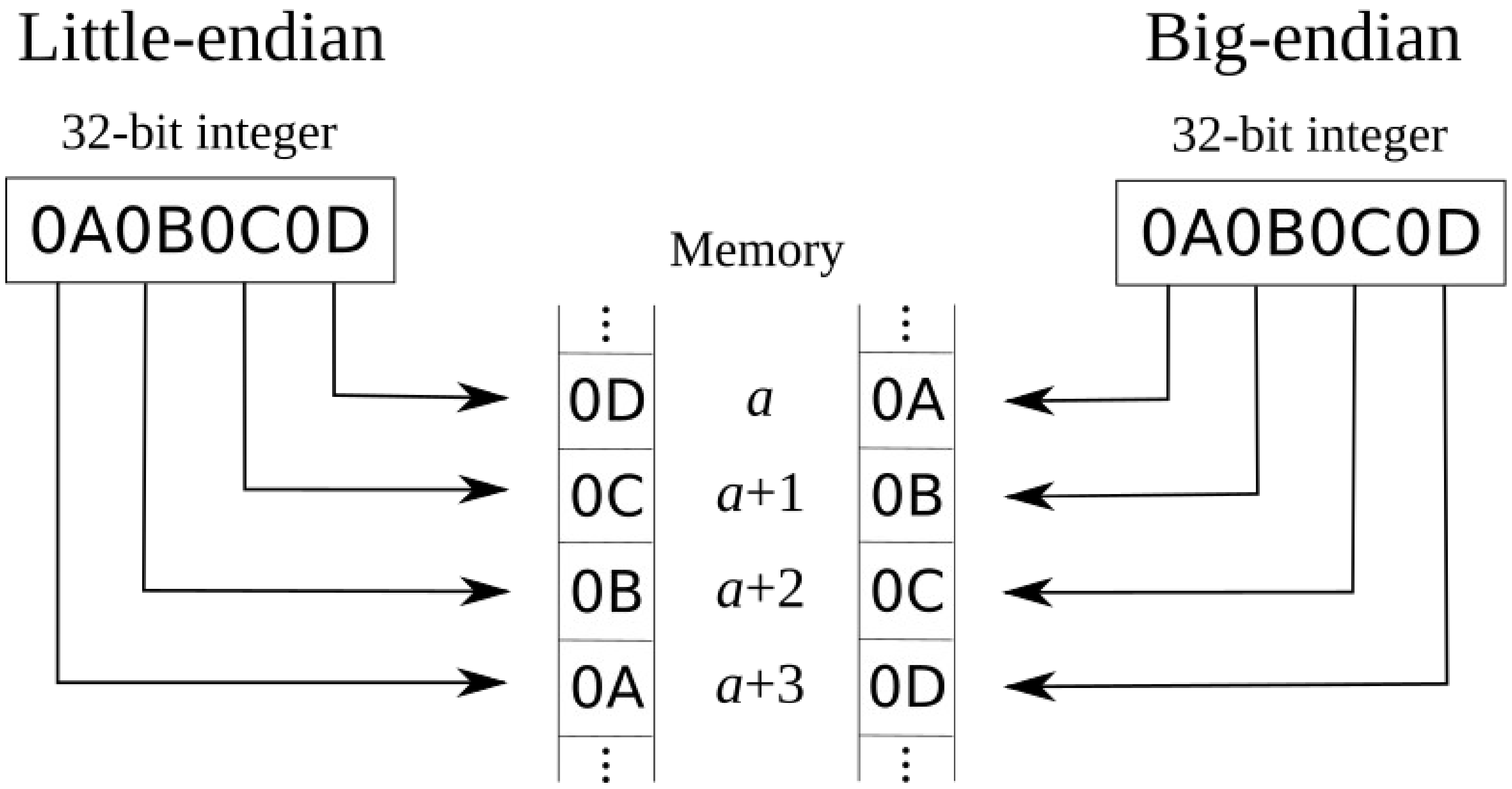
Endianness - htons()

Big-endian system

- stores the most significant byte (MSB) of a word at the smallest memory address

Little-endian system

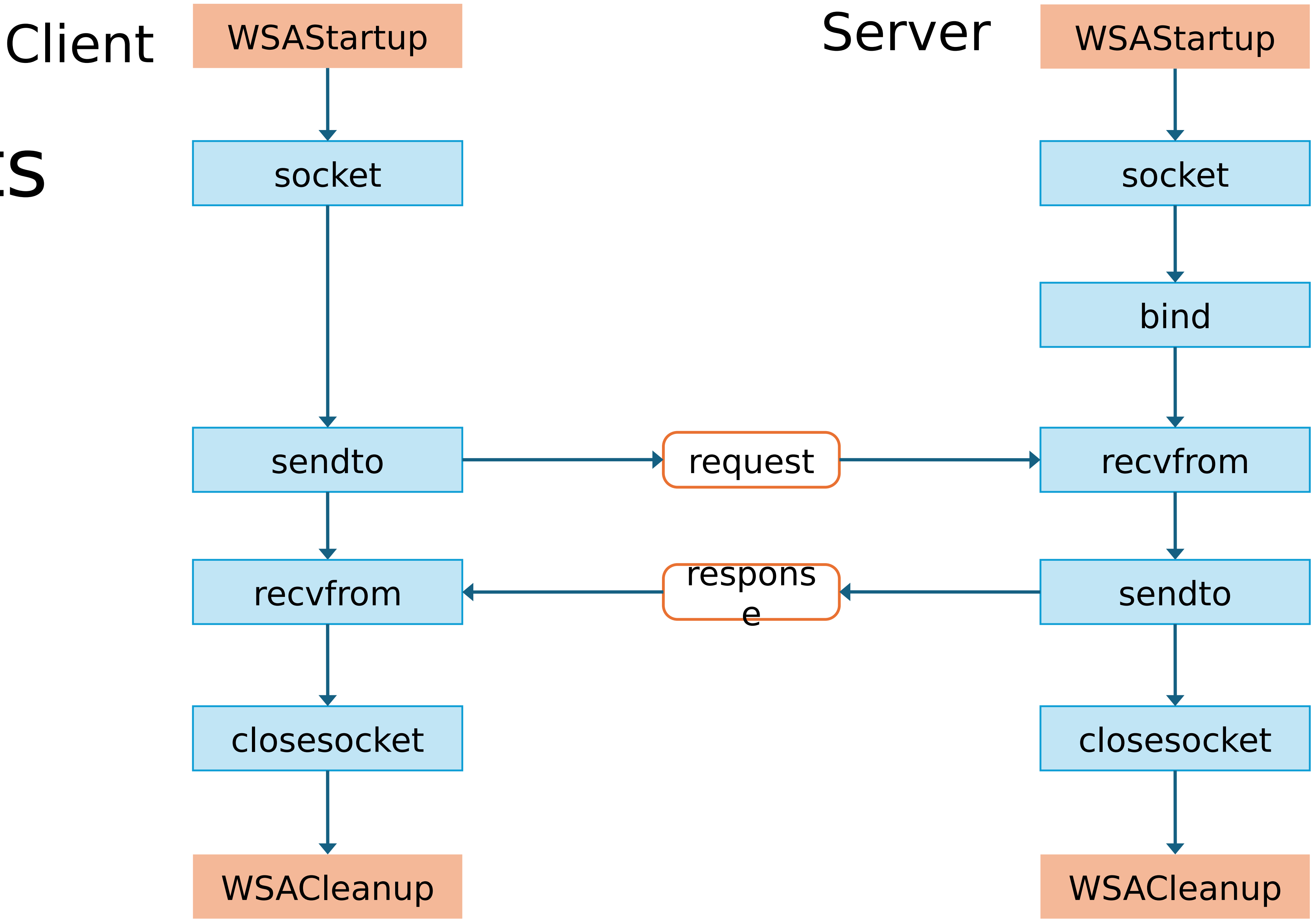
- stores the least-significant byte (LSB) at the smallest address



Endianness

A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest. A little-endian system, in contrast, stores the least-significant byte at the smallest addres

UDP sockets


Client
WSAStartup
socket
sendto
recvfrom
closesocket
WSACleanup
request
respons
e
Server
WSAStartup
socket
bind
recvfrom
sendto
closesocket
WSACleanup

UDP client

```
#include <winsock2.h>
#include <ws2tcpip.h> // InetPton


WSADATA wsa;
SOCKET socketClient;
```

Holds target address

```
struct sockaddr_in addressServer;
char sTarget[] = "127.0.0.1";    // loopback address
```

Start use of Winsock DLL (*Ws2_32.dll*)

```
WSAStartup(MAKEWORD(2,2), &wsa);
```

Create UDP-socket (IPv4)

```
socketClient = socket(AF_INET,  SOCK_DGRAM, 0 );
```

Receiver's IP address – convert text-address to binary form

```
// Initialize address structure wrt. the target
InetPton(AF_INET, __TEXT(sTarget), &
addressServer.sin_addr);
```

Address is IPv4

```
addressServer.sin_family = AF_INET;
```

Specify **target's** port number

```
addressServer.sin_port = htons(8888);   // LSB -> MSB
```

Send message to target address using socketClient

```
sendto(socketClient, sMessage, sizeof(message), 0,
            (struct sockaddr*)& addressServer,
sizeof(addressServer) );
```

UDP client

Receiver’s IP address – convert text-address to binary form

Address is IPv4

Specify target’s port number

Send message to address using socketClient

Receive message from client

Close the socket

```
struct sockaddr_in addressServer;

WSAStartup(MAKEWORD(2,2), &wsa);

socketClient = socket(AF_INET , SOCK_DGRAM , 0 );

// Initialize address structure wrt. the target
InetPton(AF_INET, __TEXT("127.0.0.1"), &
addressServer.sin_addr);
addressServer.sin_family = AF_INET;
addressServer.sin_port = htons(8888);   // LSB -> MSB

sendto(socketClient, sSendText, strlen(sSendText), 0,
          (struct sockaddr*)& addressServer,
sizeof(addressServer) );

recvfrom( ... );

closesocket(socketClient);

WSACleanup();
```

Linking in CLion

- In Windows, the WinSock library is implemented by WS2_32.dll

- In CLion, this DLL is specified in CMakeLists.txt by the line

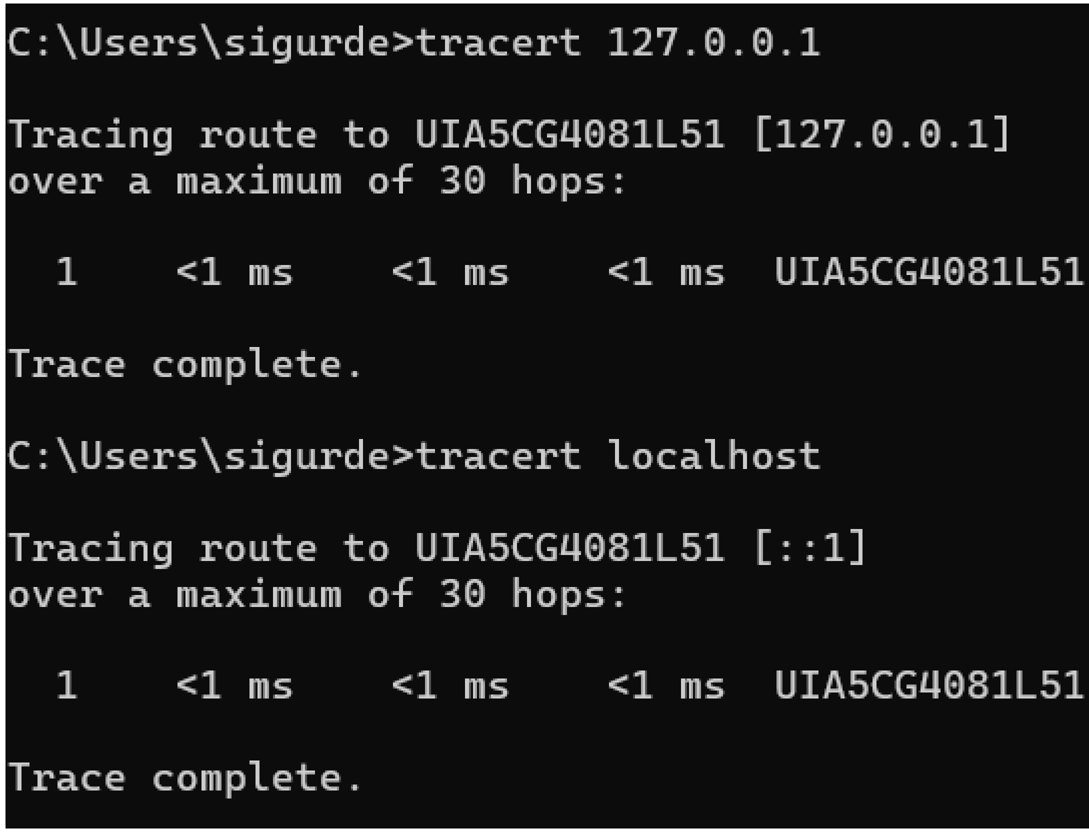
```
target_link_libraries(${CMAKE_PROJECT_NAME} Ws2_32)
```

Programming exercise

1. Client reads a line of characters (data) from its keyboard and sends data to server
2. Server receives the data and converts characters to uppercase
3. Server sends modified data to client
4. Client receives modified data and displays line on its screen

IP loopback address

- The special network address, 127.0.0.1, is defined as a local loopback address
- Hosts use local loopback addresses to send messages to themselves
- In Windows, the name **localhost** is an alias for 127.0.0.1



```
C:\Users\sigurde>tracert 127.0.0.1

Tracing route to UIA5CG4081L51 [127.0.0.1]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51

Trace complete.

C:\Users\sigurde>tracert localhost

Tracing route to UIA5CG4081L51 [::1]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51

Trace complete.
```