TCP

Reliable delivery

all the good things from last time

Connection-oriented

Full duplex (= bidirectional)

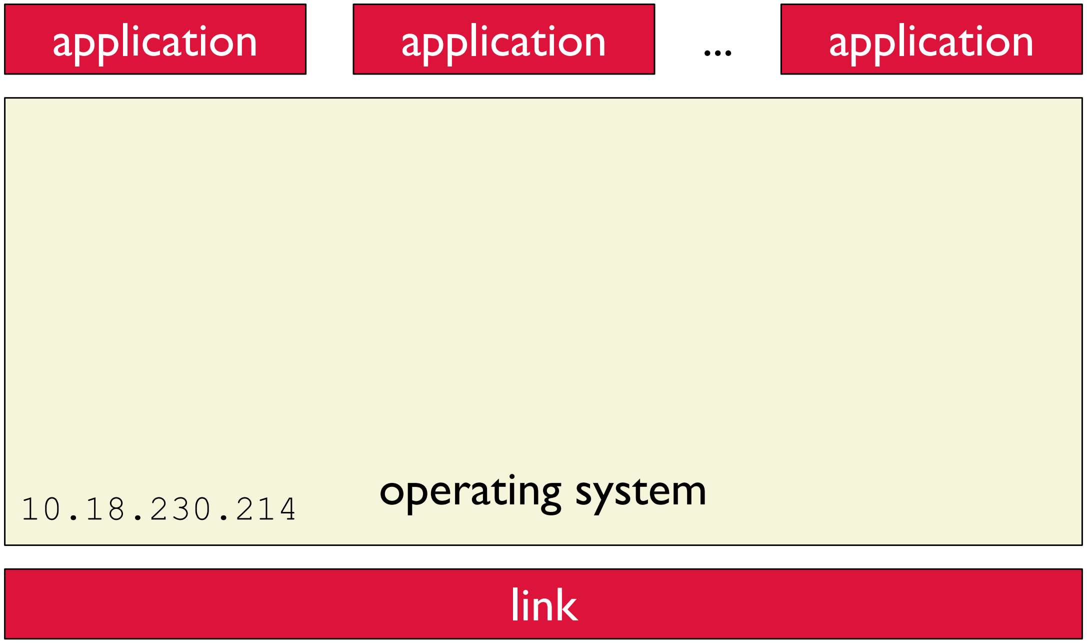
TCP Echo Server in Java

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        ServerSocket listener = new ServerSocket(server_port);
        System.out.println("Listening at " + server_port);

        for (int count = 1; true; count++) {
            Socket socket = listener.accept();
            InputStream input = socket.getInputStream();
            OutputStream output = socket.getOutputStream();
            byte[] buffer = new byte[5];

            int got = input.read(buffer);

            System.out.println(count + " Heard from " + socket.getInetAddress() + " " + socket.getPort());
            for (int i = 0; i < got; i++)
                System.out.printf(" %d", buffer[i]);
            System.out.print("\n");

            output.write(buffer, 0, got);

            socket.close();
        }
    }
}
```

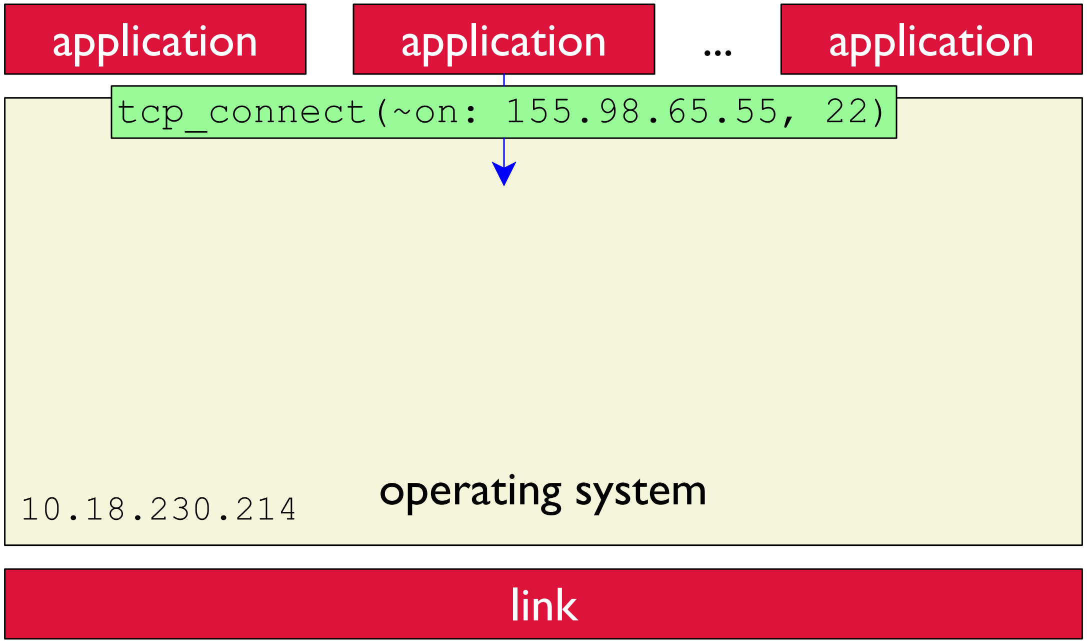
TCP Echo Client in Java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        Socket socket = new Socket("localhost", server_port);
        InputStream input = socket.getInputStream();
        OutputStream output = socket.getOutputStream();
        byte[] buf = new byte[3];

        buf[0] = 10;
        buf[1] = 20;
        buf[2] = 30;

        output.write(buf);
        int got = input.read(buf);
        for (int i = 0; i < got; i++)
            System.out.printf(" %d", buf[i]);
        System.out.print("\n");

        socket.close();
    }
}
```

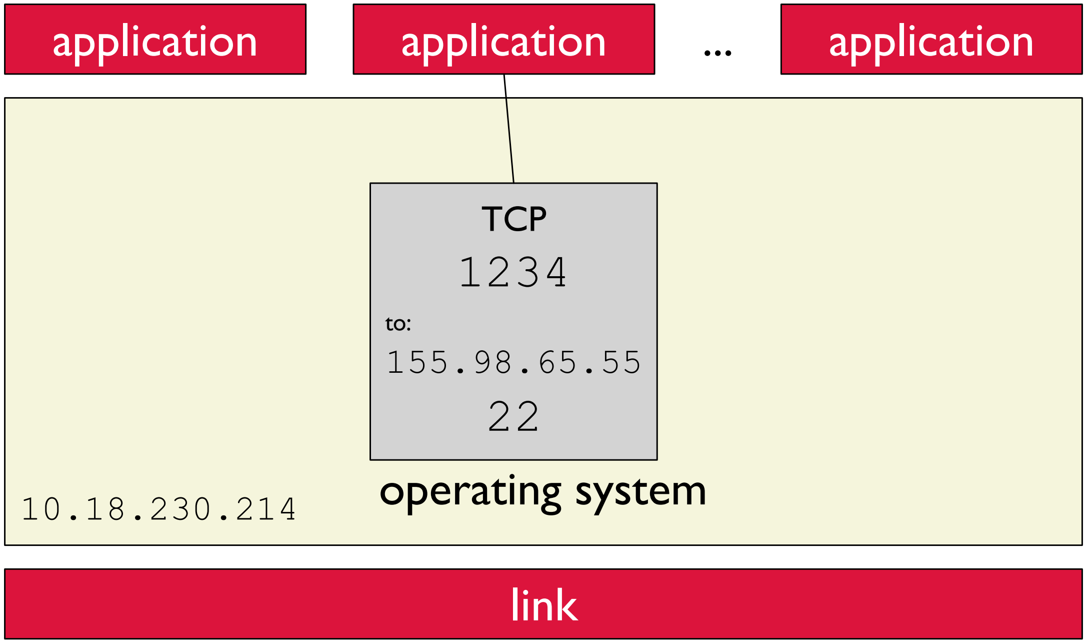
Client-Side TCP



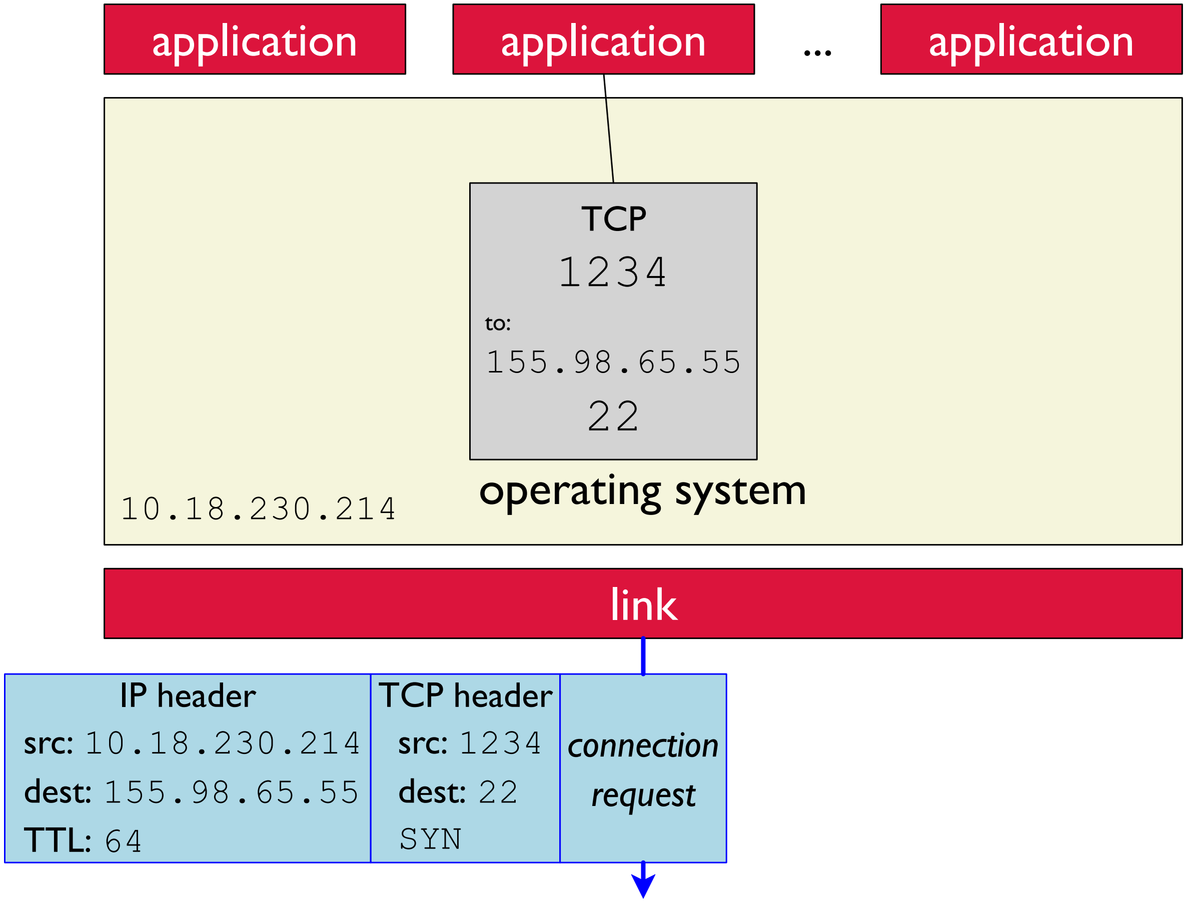
Client-Side TCP



Client-Side TCP



Client-Side TCP



application

application

...

application

TCP

1234

to:

155.98.65.55

22

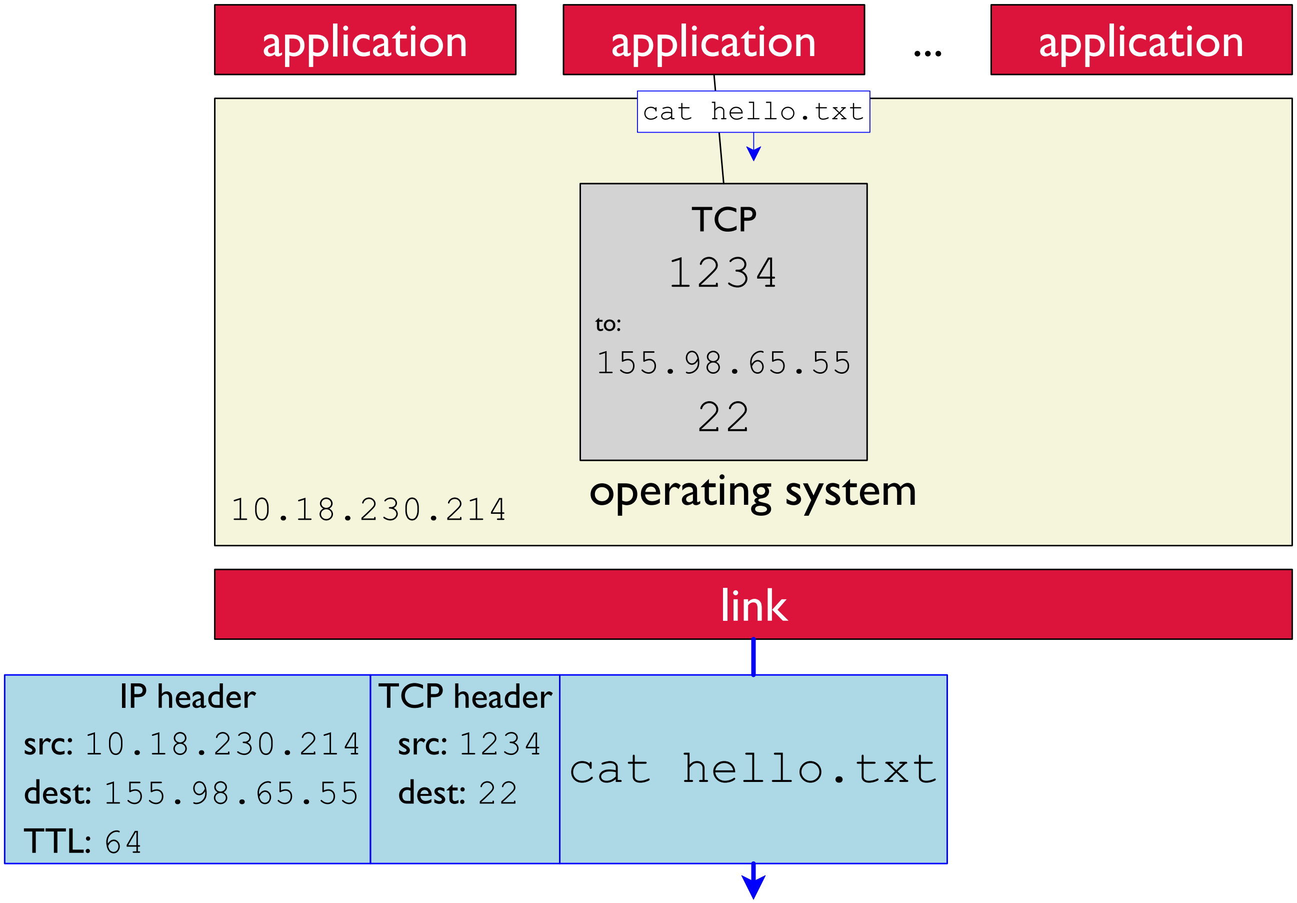
10.18.230.214

operating system

link

IP header	TCP header	
src: 10.18.230.214	src: 1234	*connection*
dest: 155.98.65.55	dest: 22	*request*
TTL: 64	SYN	

Client-Side TCP

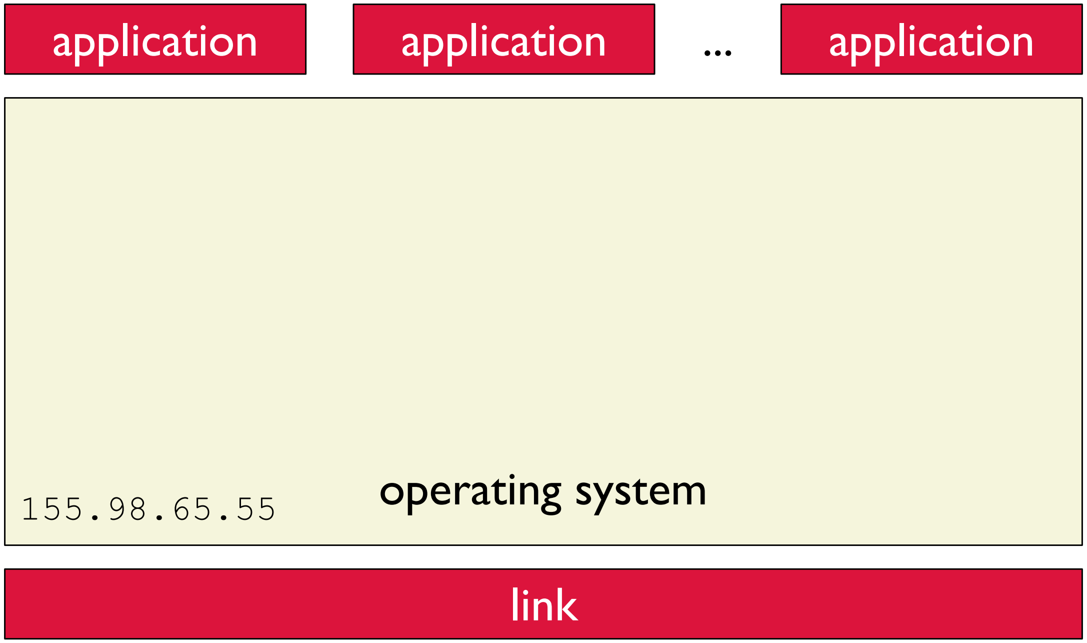


IP header
src: 10.18.230.214
dest: 155.98.65.55
TTL: 64

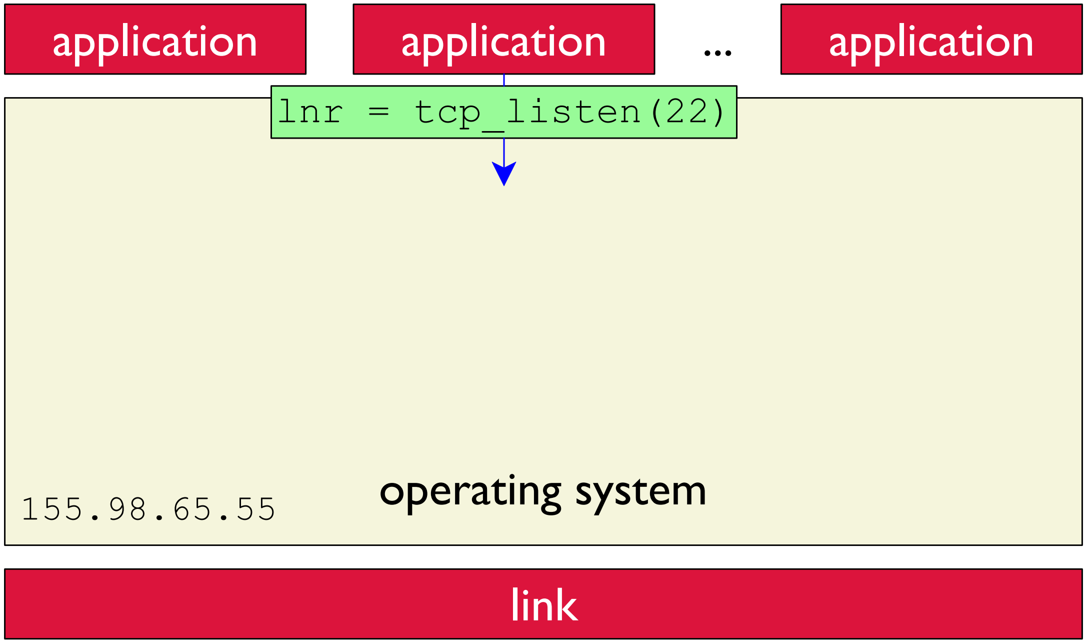
TCP header
src: 1234
dest: 22

```
cat hello.txt
```

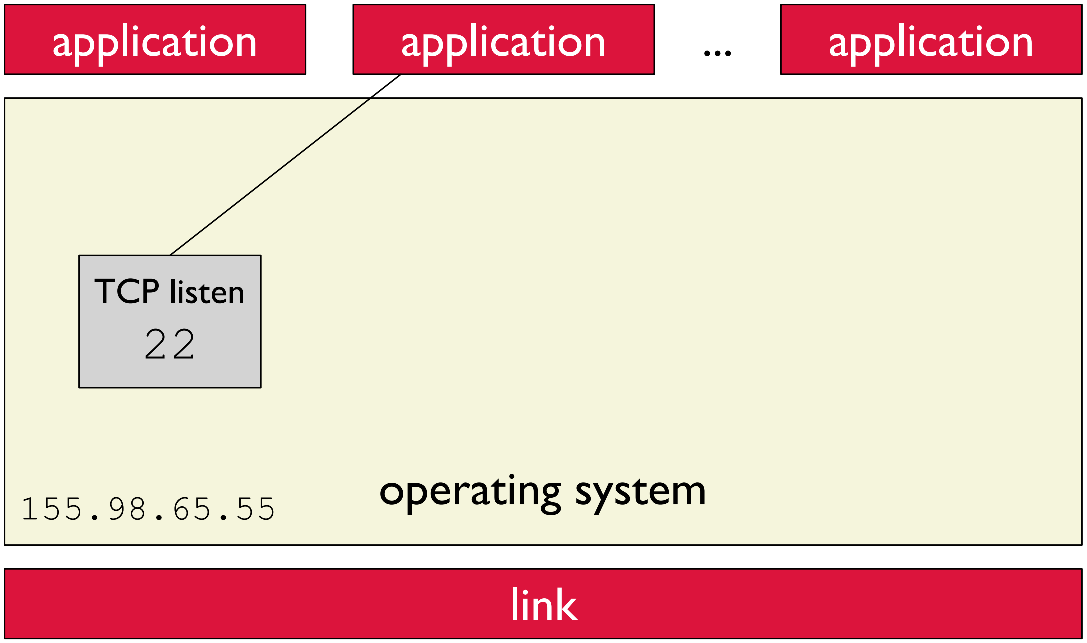
Server-Side TCP



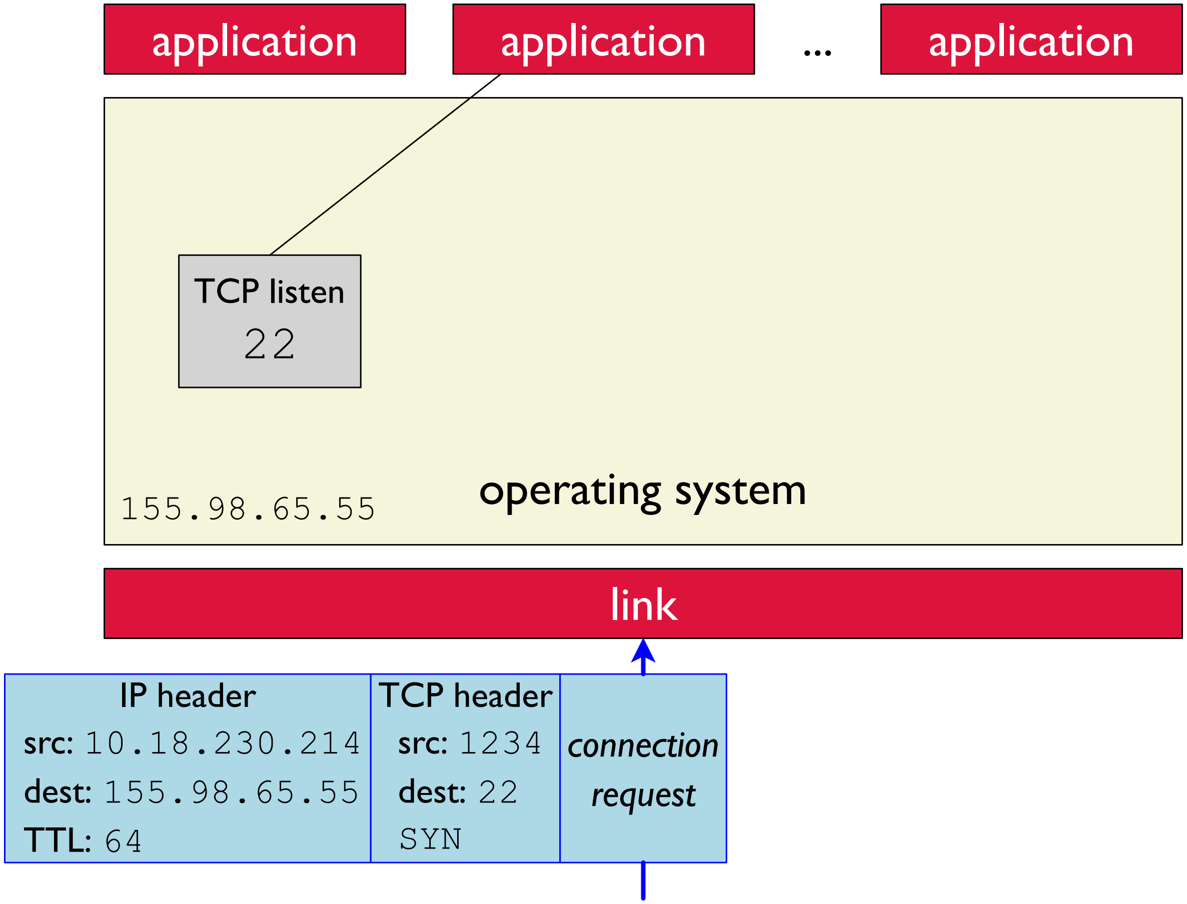
Server-Side TCP



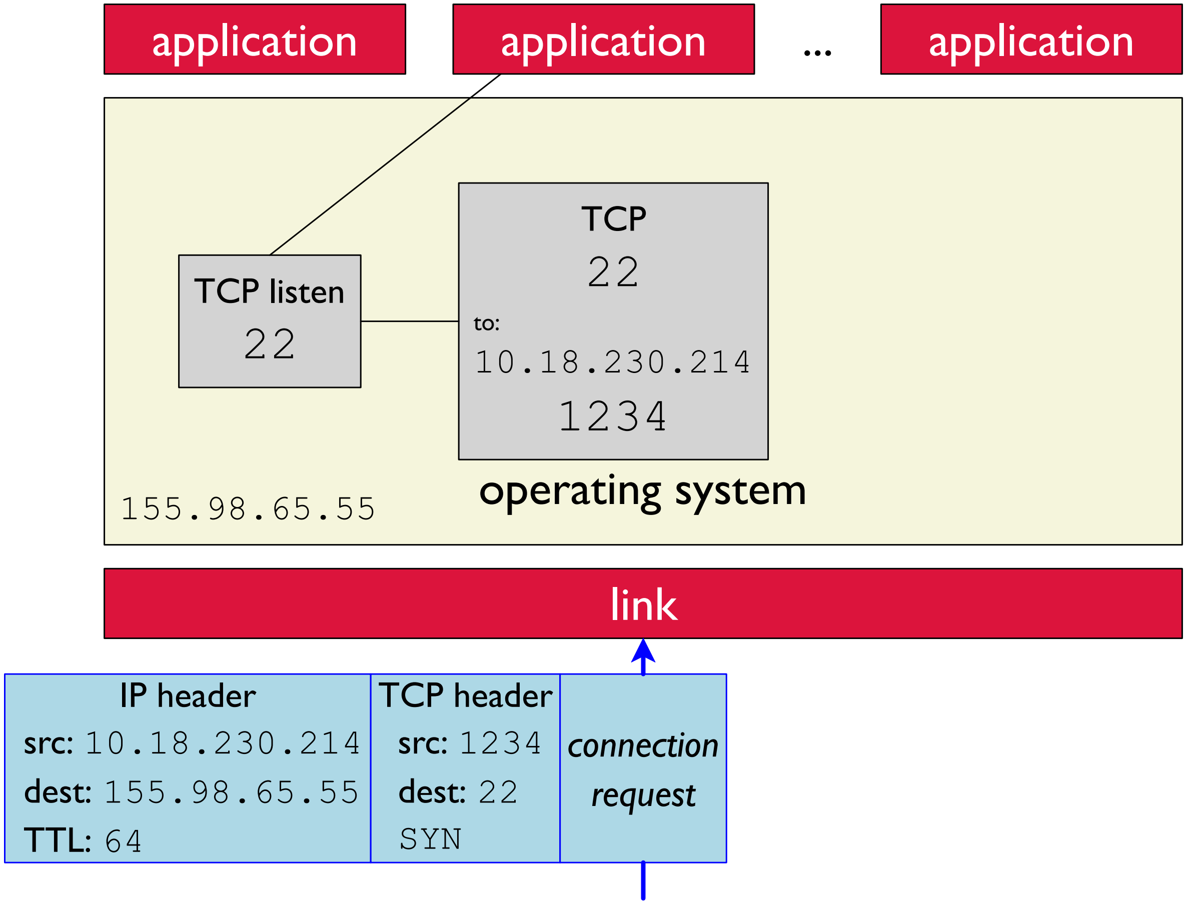
Server-Side TCP



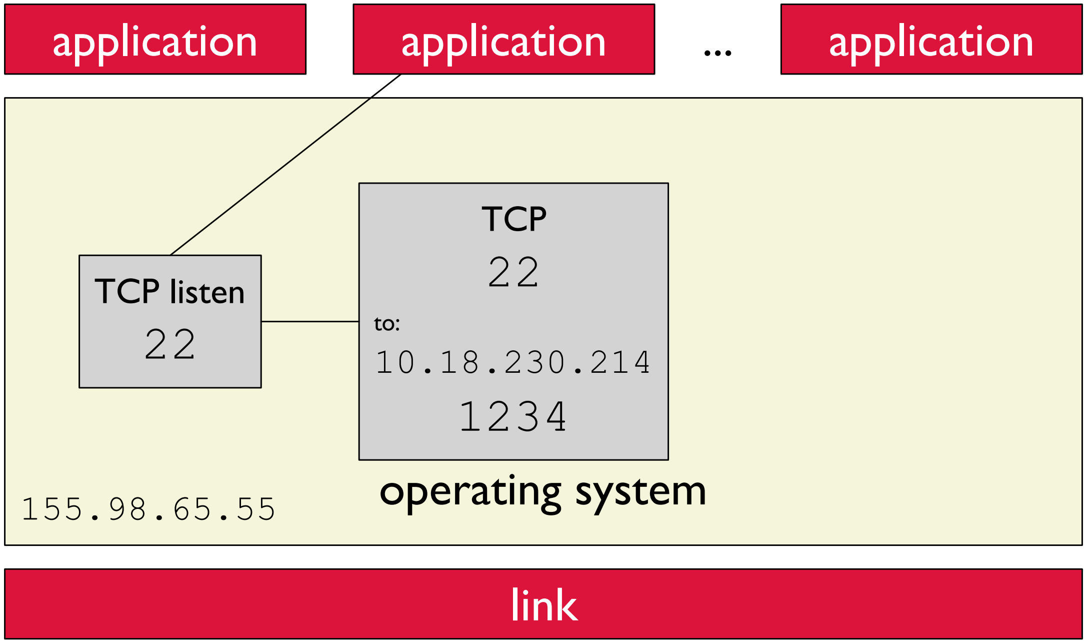
Server-Side TCP



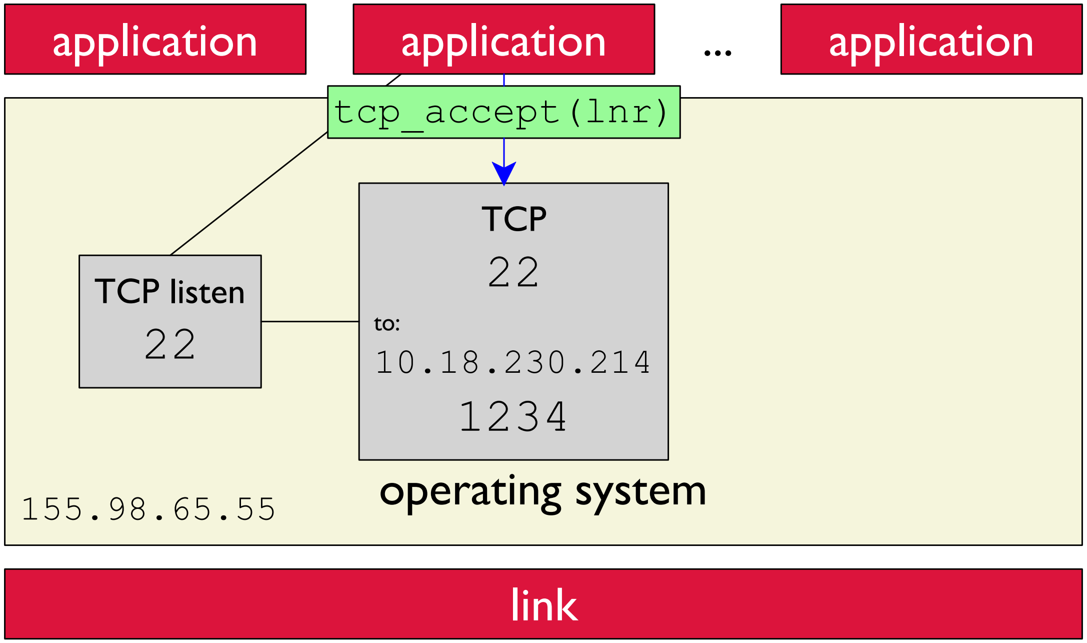
Server-Side TCP



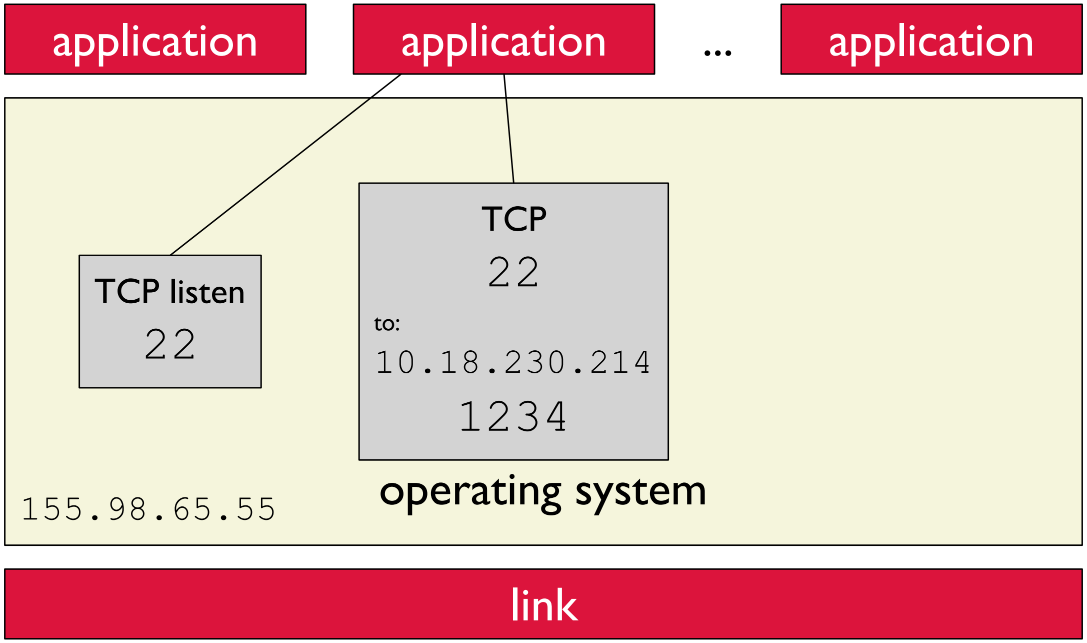
Server-Side TCP



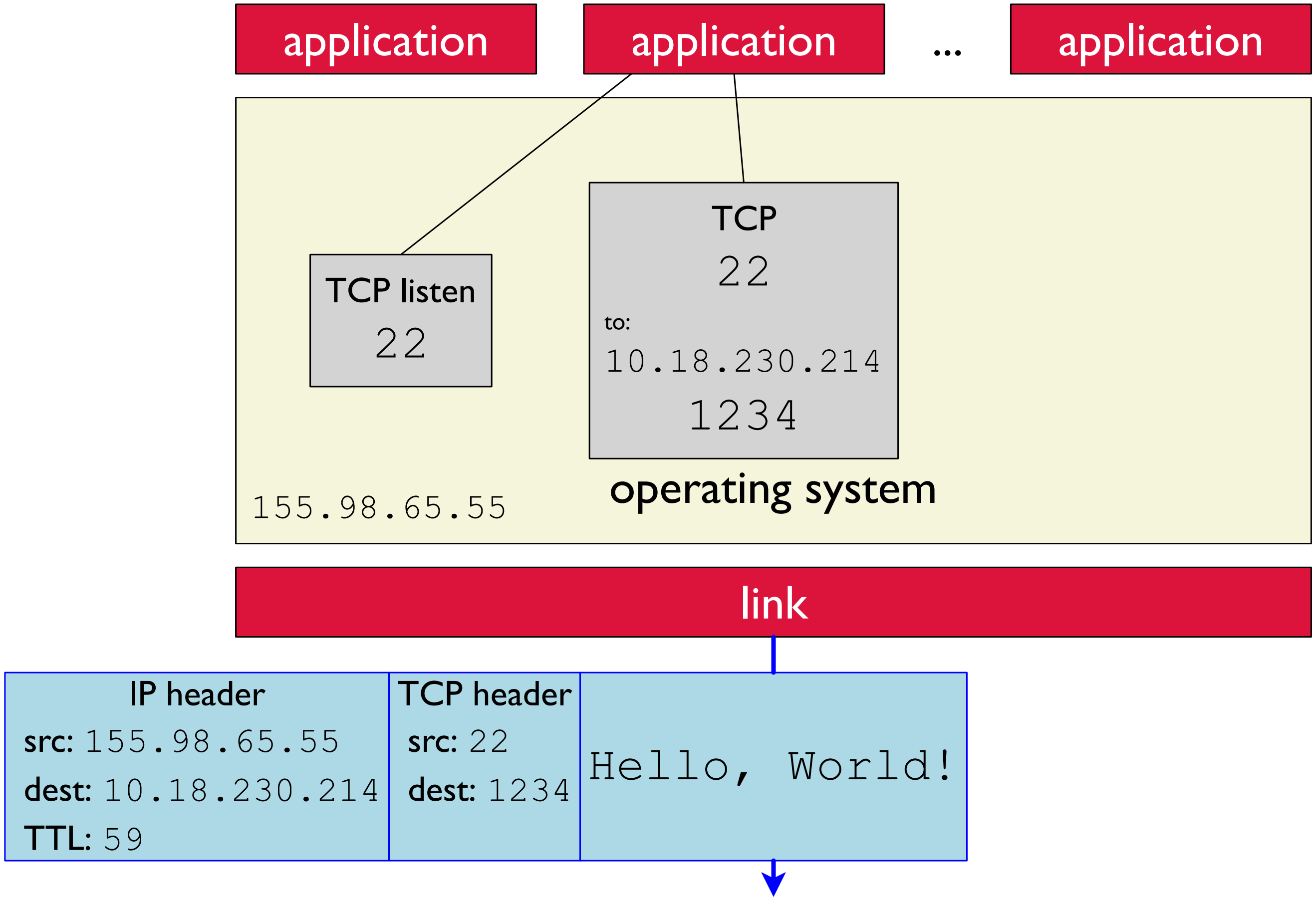
Server-Side TCP



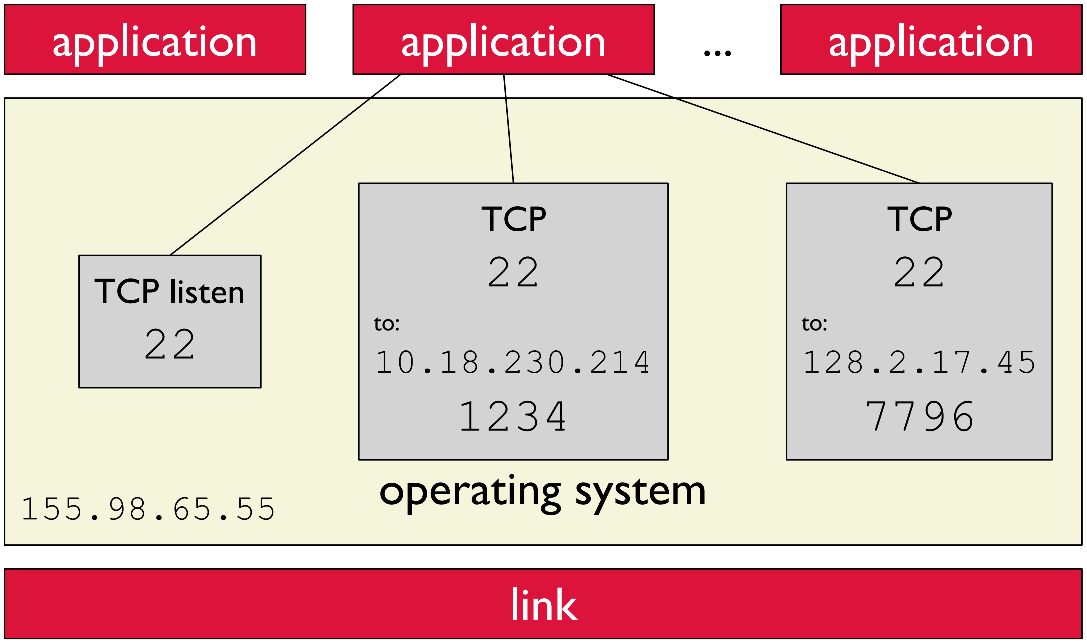
Server-Side TCP



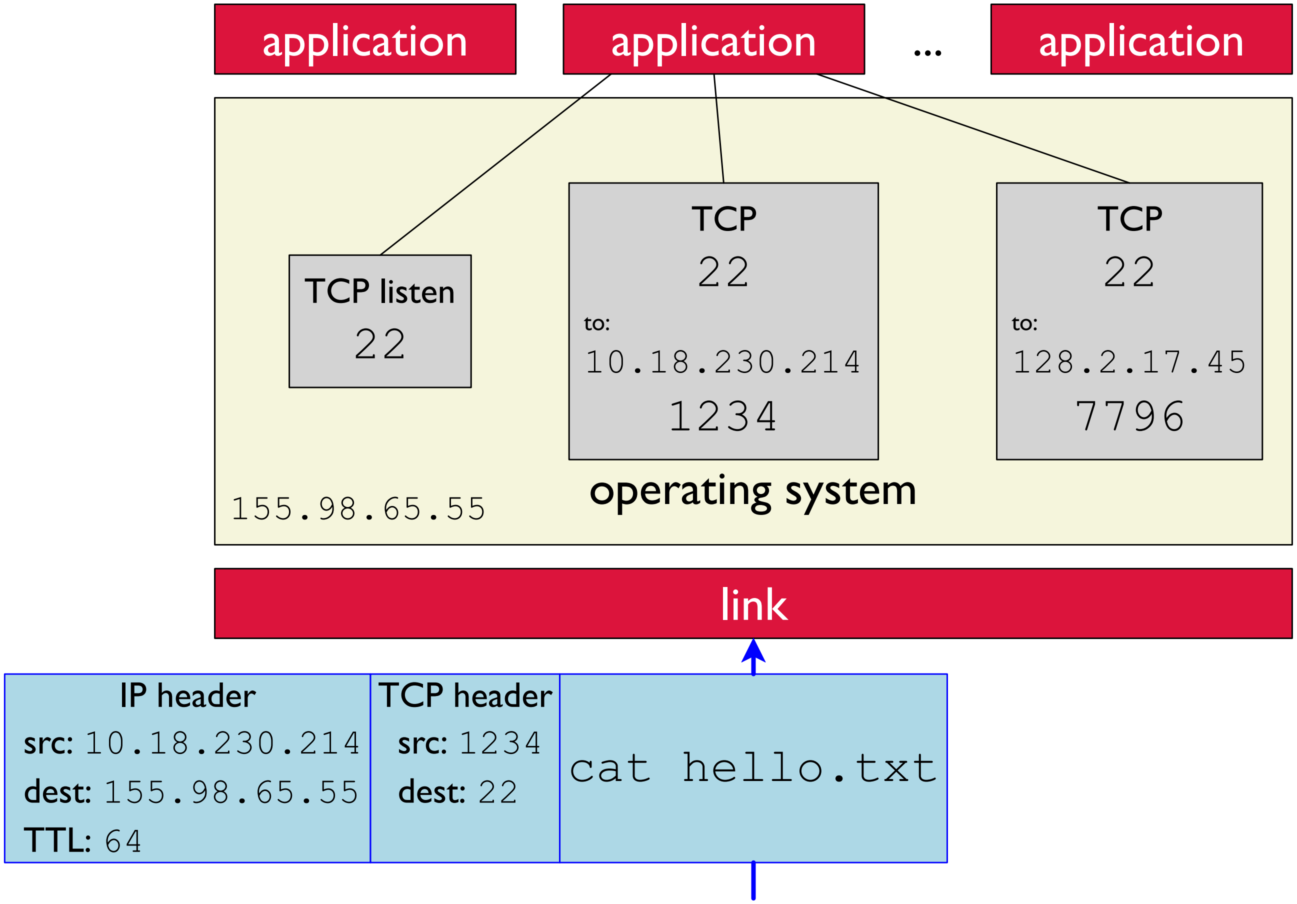
Server-Side TCP



Server-Side TCP



Server-Side TCP



application

application

…

application

TCP listen

22

TCP

22

to:

10.18.230.214

1234

TCP

22

to:

128.2.17.45

7796

155.98.65.55

operating system

link

IP header

src: 10.18.230.214

dest: 155.98.65.55

TTL: 64

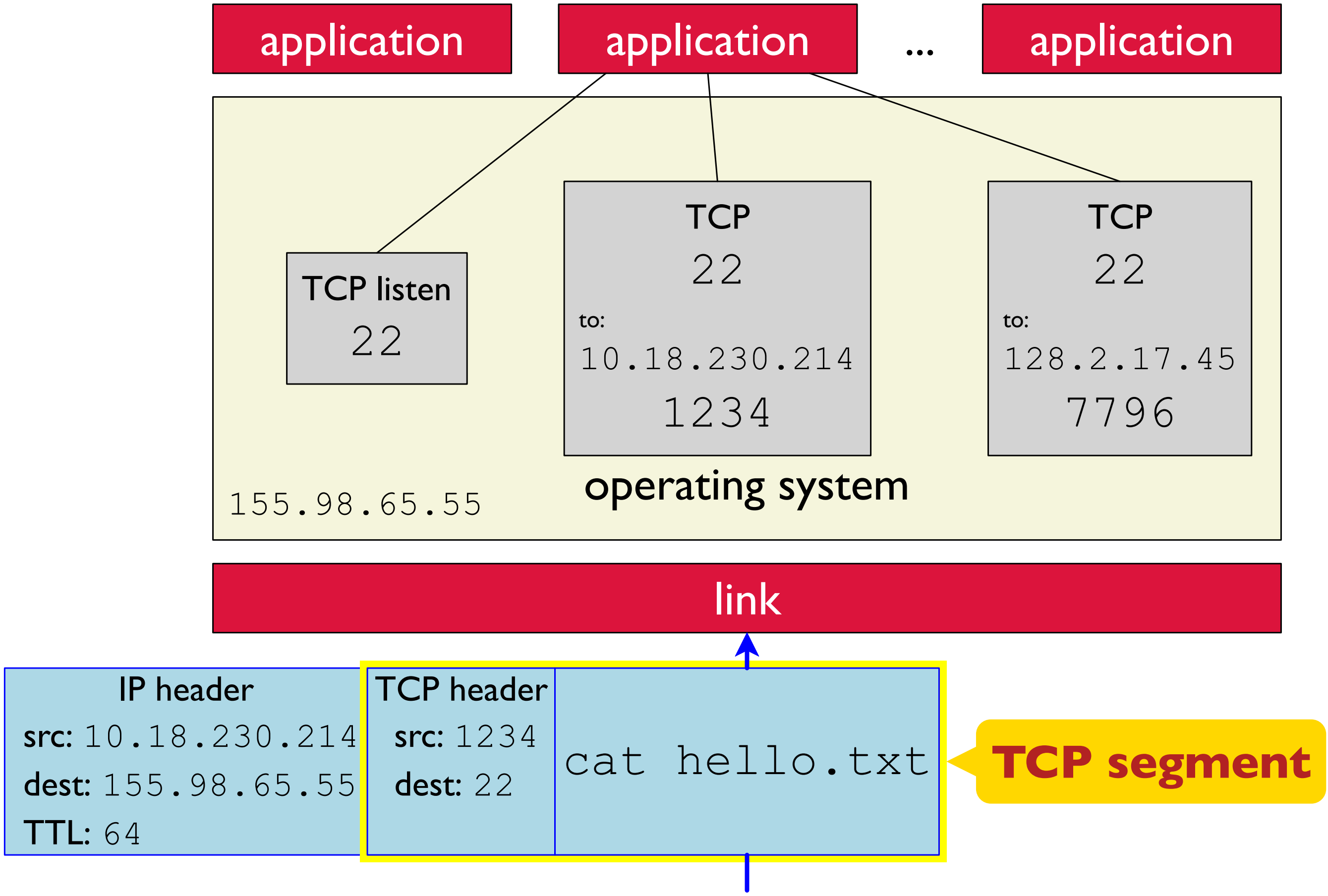
TCP header

src: 1234

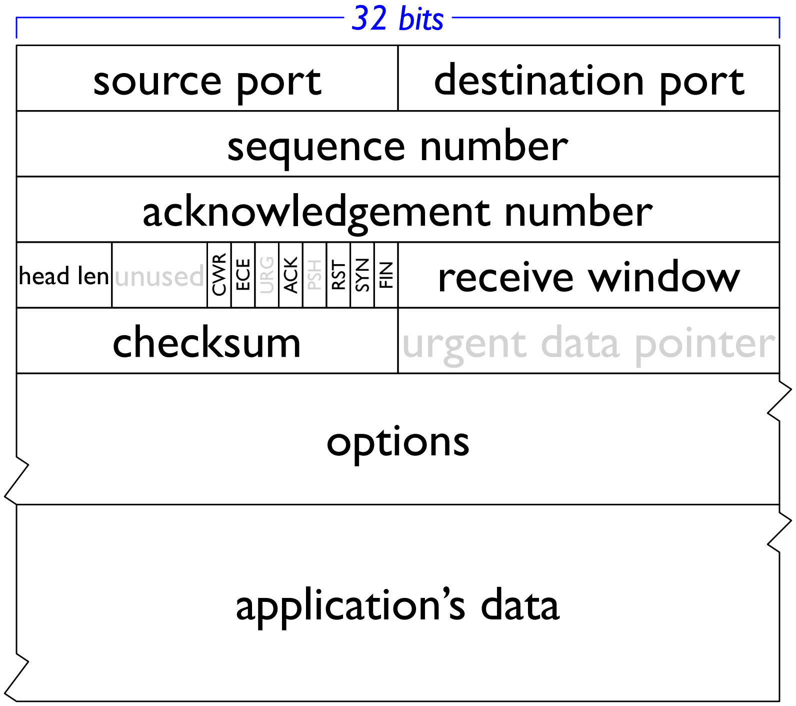
dest: 22

cat hello.txt

Server-Side TCP



TCP Segment Details



32 bits

source port | destination port

sequence number

acknowledgement number

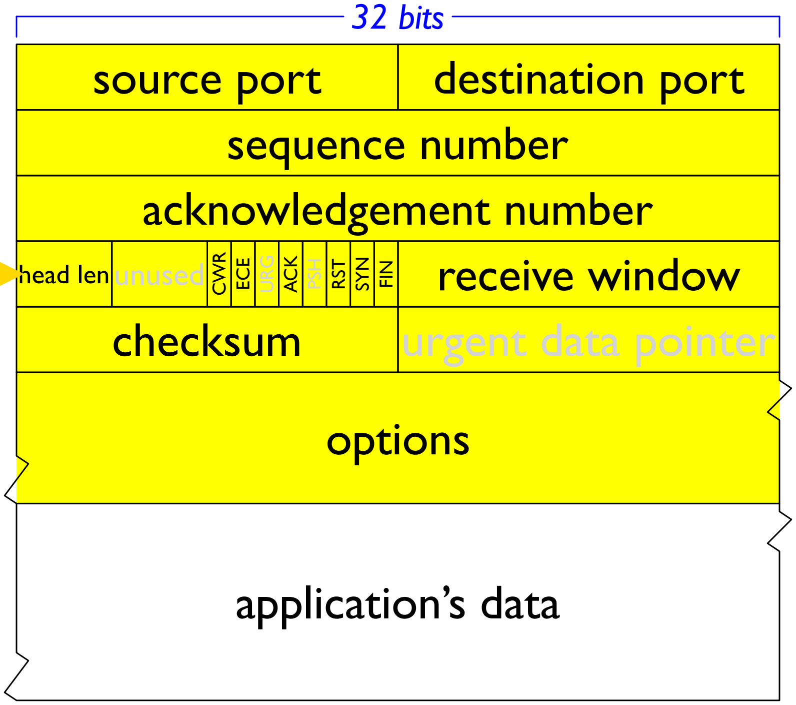
head len | unused | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | receive window

checksum | urgent data pointer

options

application’s data

TCP Segment Details



32 bits

source port | destination port

sequence number

acknowledgement number

head len | unused | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | receive window

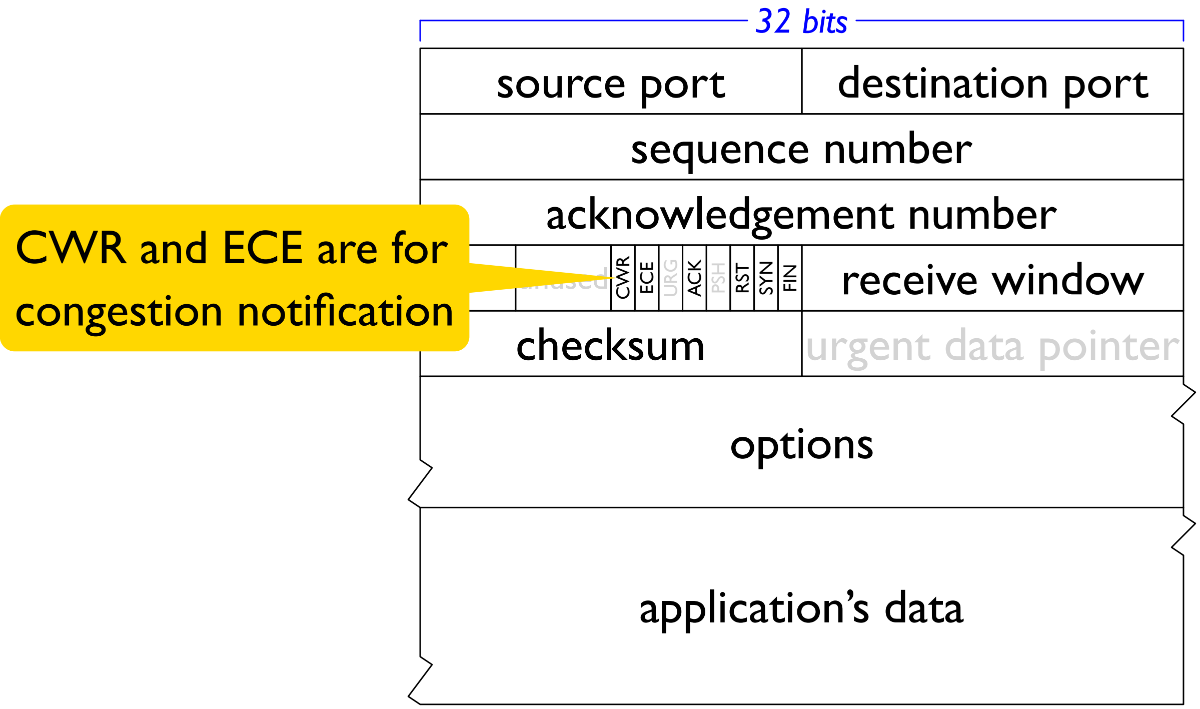
checksum | urgent data pointer

options

application’s data

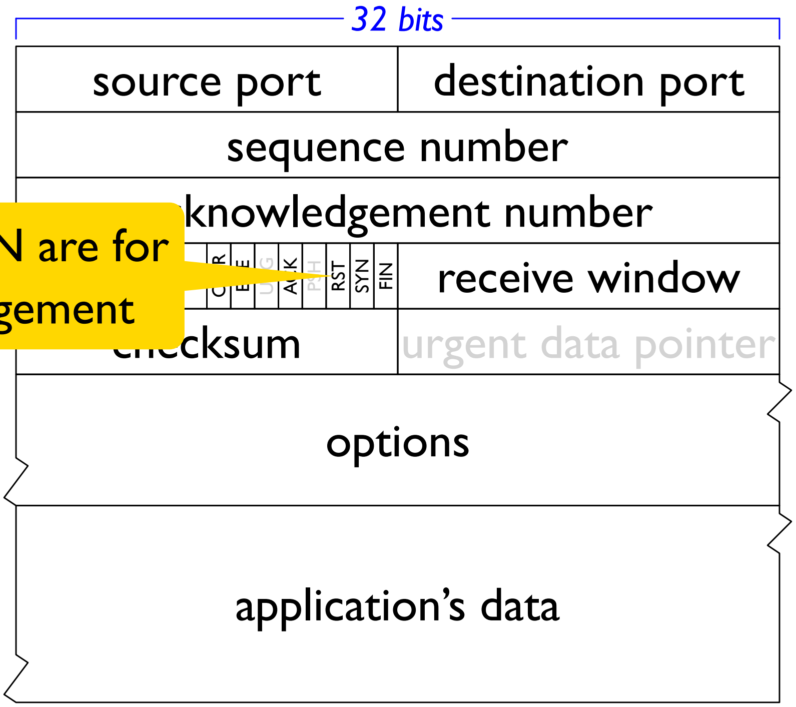
header size in 32-bit words

TCP Segment Details



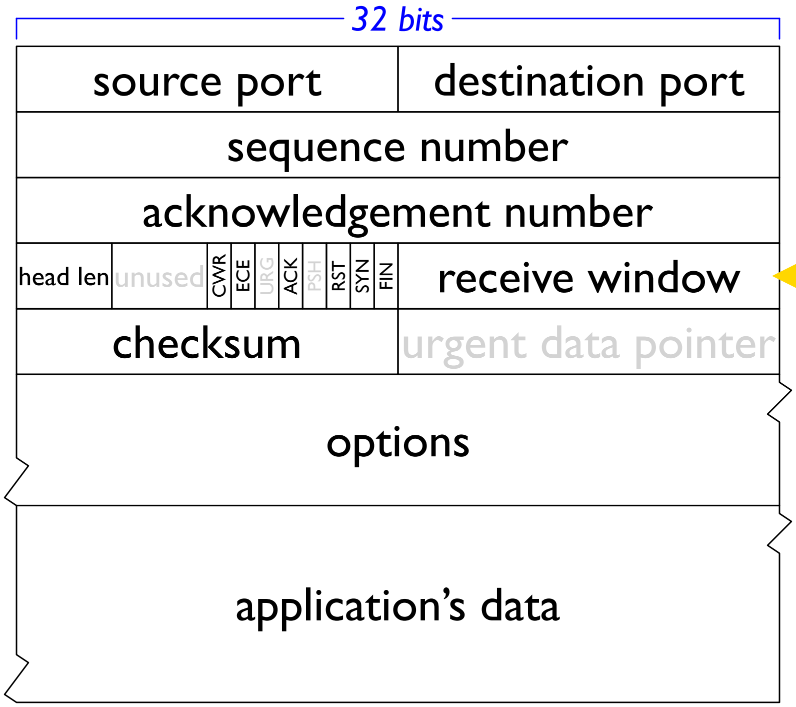
CWR and ECE are for congestion notification

TCP Segment Details



RST, SYN, and FIN are for connection management

TCP Segment Details



32 bits

source port

destination port

sequence number

acknowledgement number

head len

unused

CWR

ECE

URG

ACK

PSH

RST

SYN

FIN

receive window

checksum

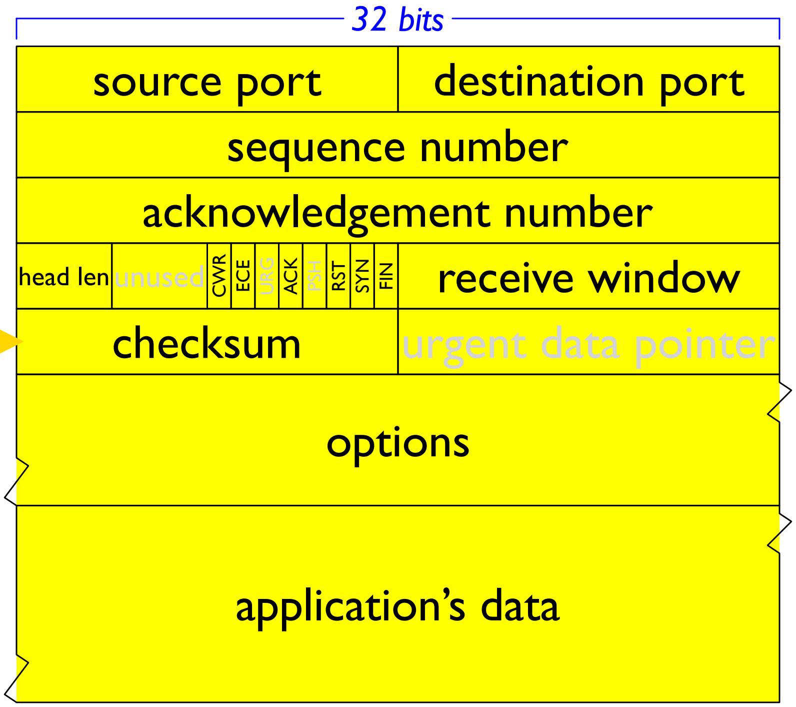
urgent data pointer

options

application’s data

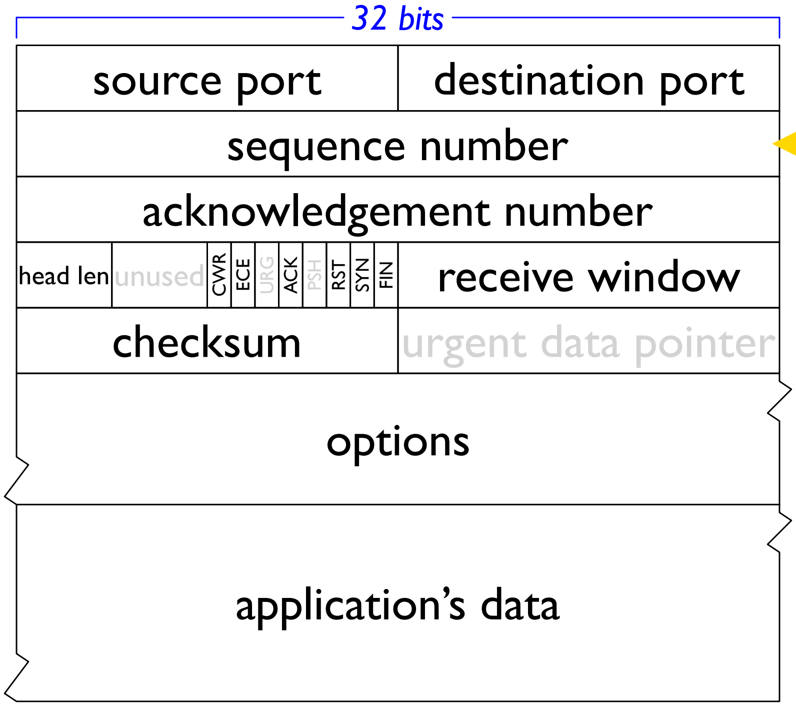
number of bytes the receiver is ready to accept

TCP Segment Details



1's complement of sum of 16-bit words

TCP Segment Details



32 bits

source port | destination port

sequence number

acknowledgement number

head len | unused | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | receive window

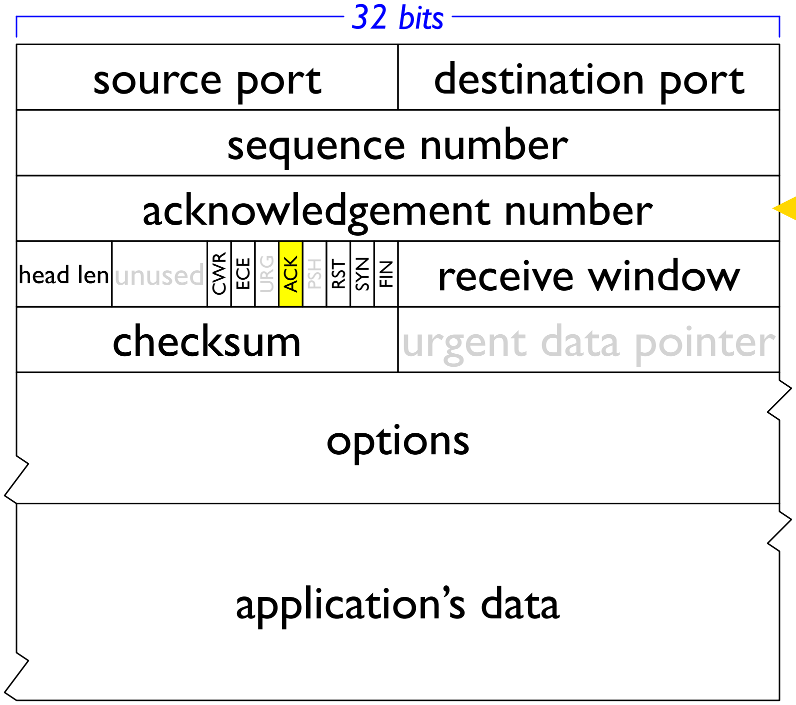
checksum | urgent data pointer

options

application's data

corresponds to bytes *sent previously*, not counting new data here — and counts bytes, not packets

TCP Segment Details



32 bits

source port | destination port

sequence number

acknowledgement number

head len | unused | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | receive window

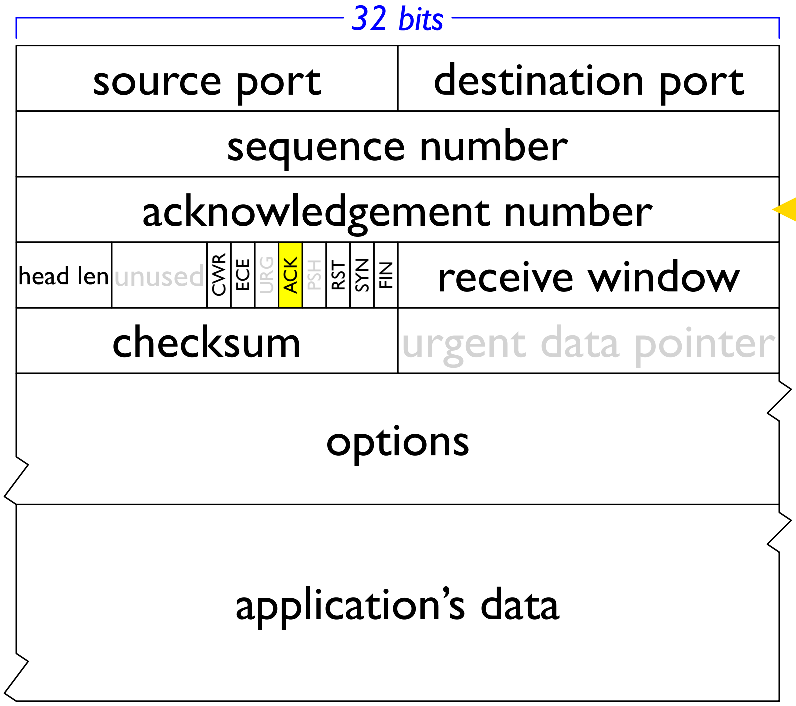
checksum | urgent data pointer

options

application’s data

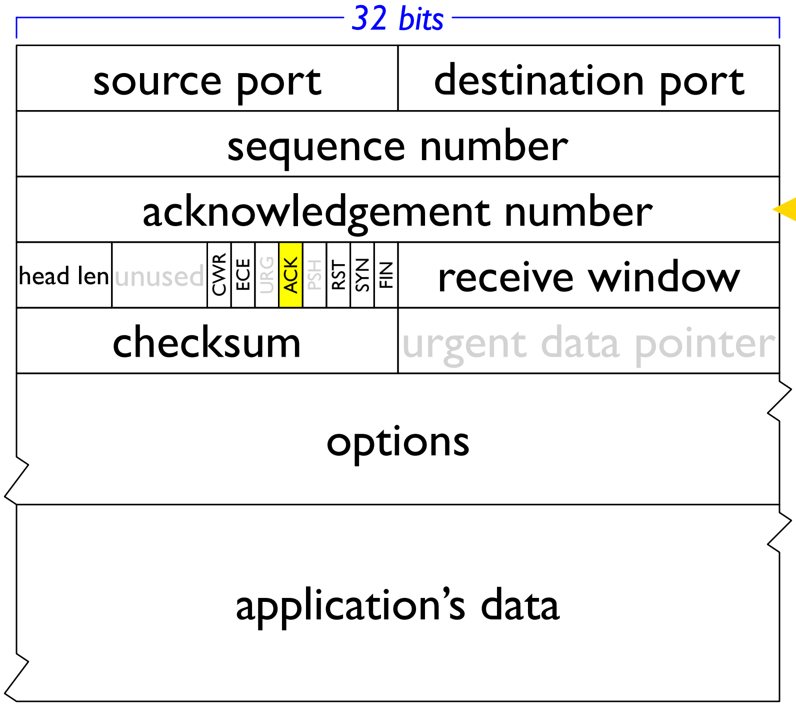
valid when ACK flag is set

TCP Segment Details



corresponds to all bytes *received*, so indicates next expected byte number

TCP Segment Details



32 bits

source port

destination port

sequence number

acknowledgement number

head len

unused

CWR

ECE

URG

ACK

PSH

RST

SYN

FIN

receive window

checksum

urgent data pointer

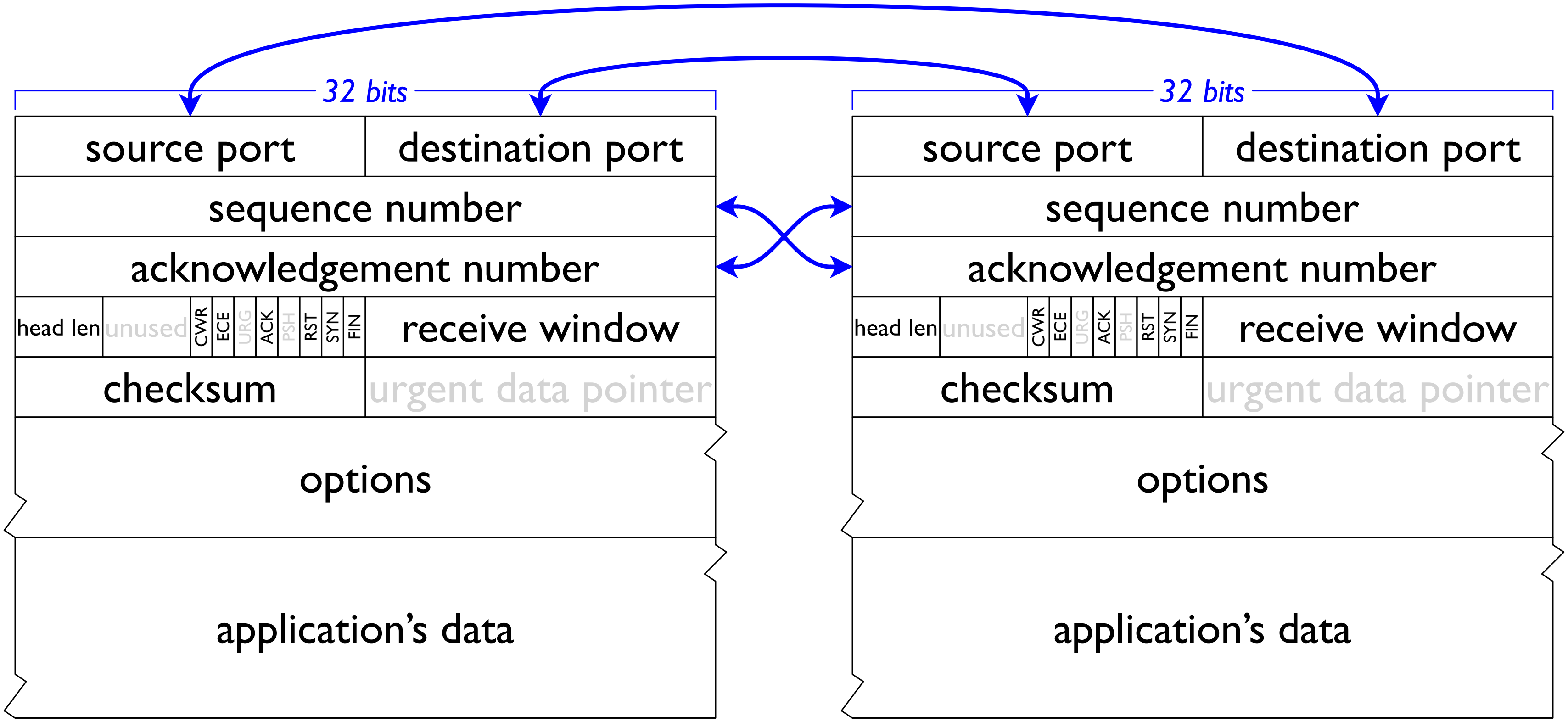
options

application's data

corresponds to all bytes *received*, so indicates next expected byte number

cumulative acknowledgements

Sender and Receiver Fields



32 bits

source port
destination port
sequence number
acknowledgement number
head len
unused
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
receive window
checksum
urgent data pointer
options
application’s data

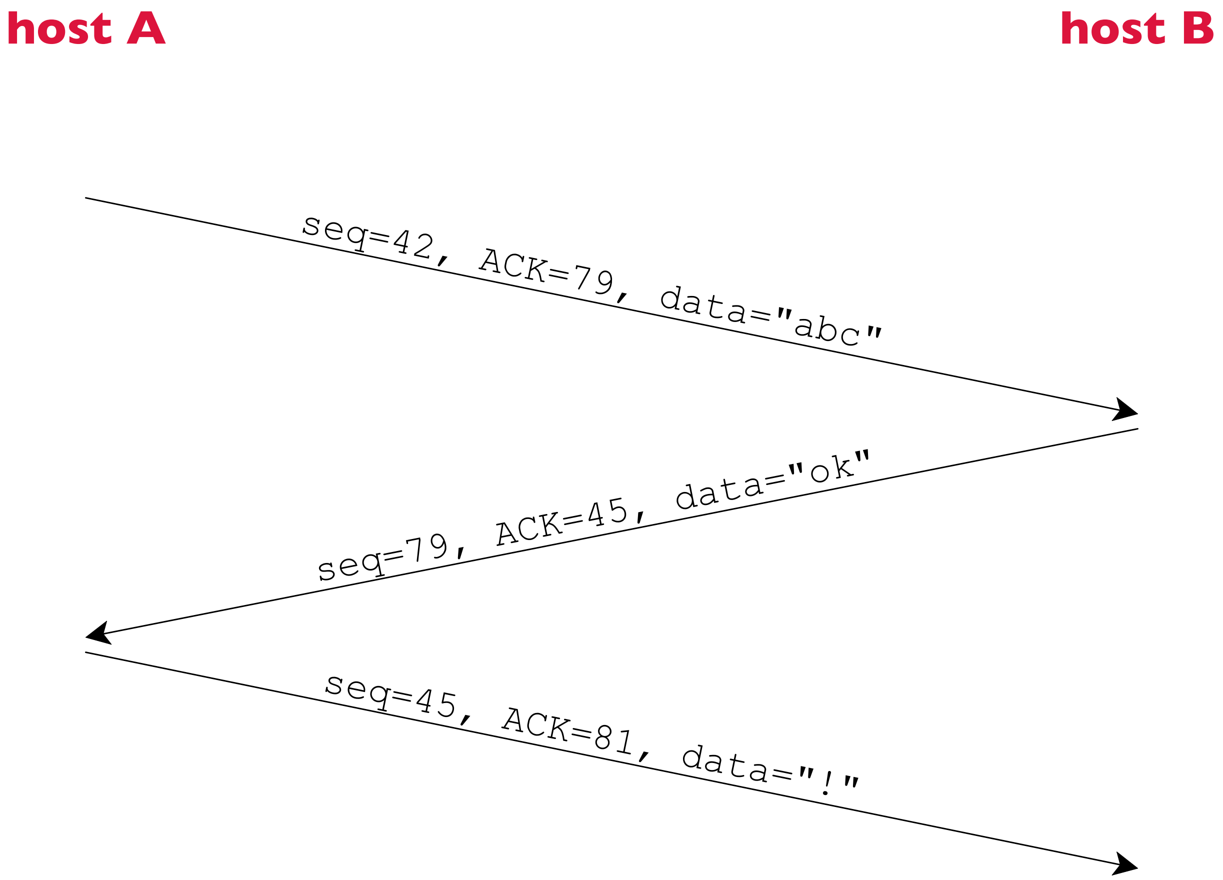
packets sent by host A

32 bits

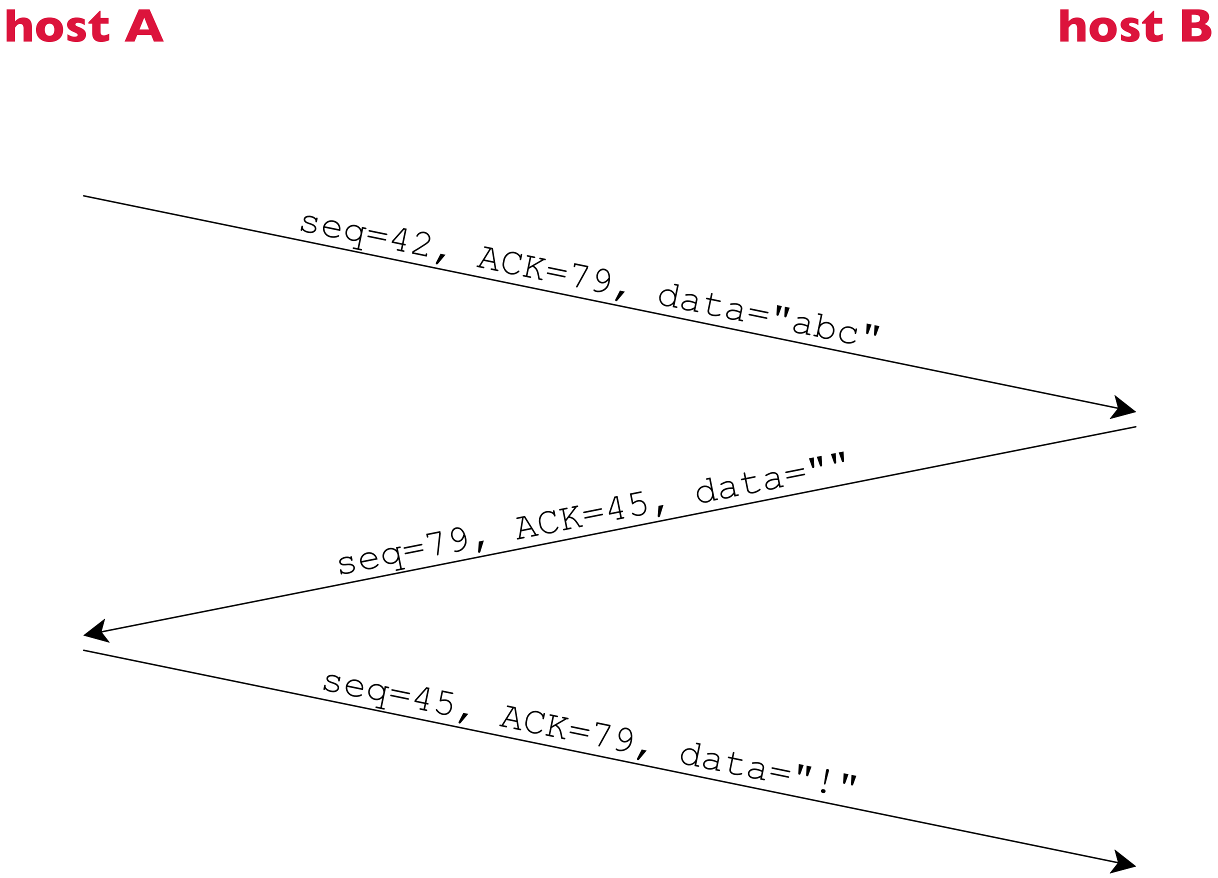
source port
destination port
sequence number
acknowledgement number
head len
unused
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
receive window
checksum
urgent data pointer
options
application’s data

packets sent by host B

Example Sequence



Example Sequence



host A

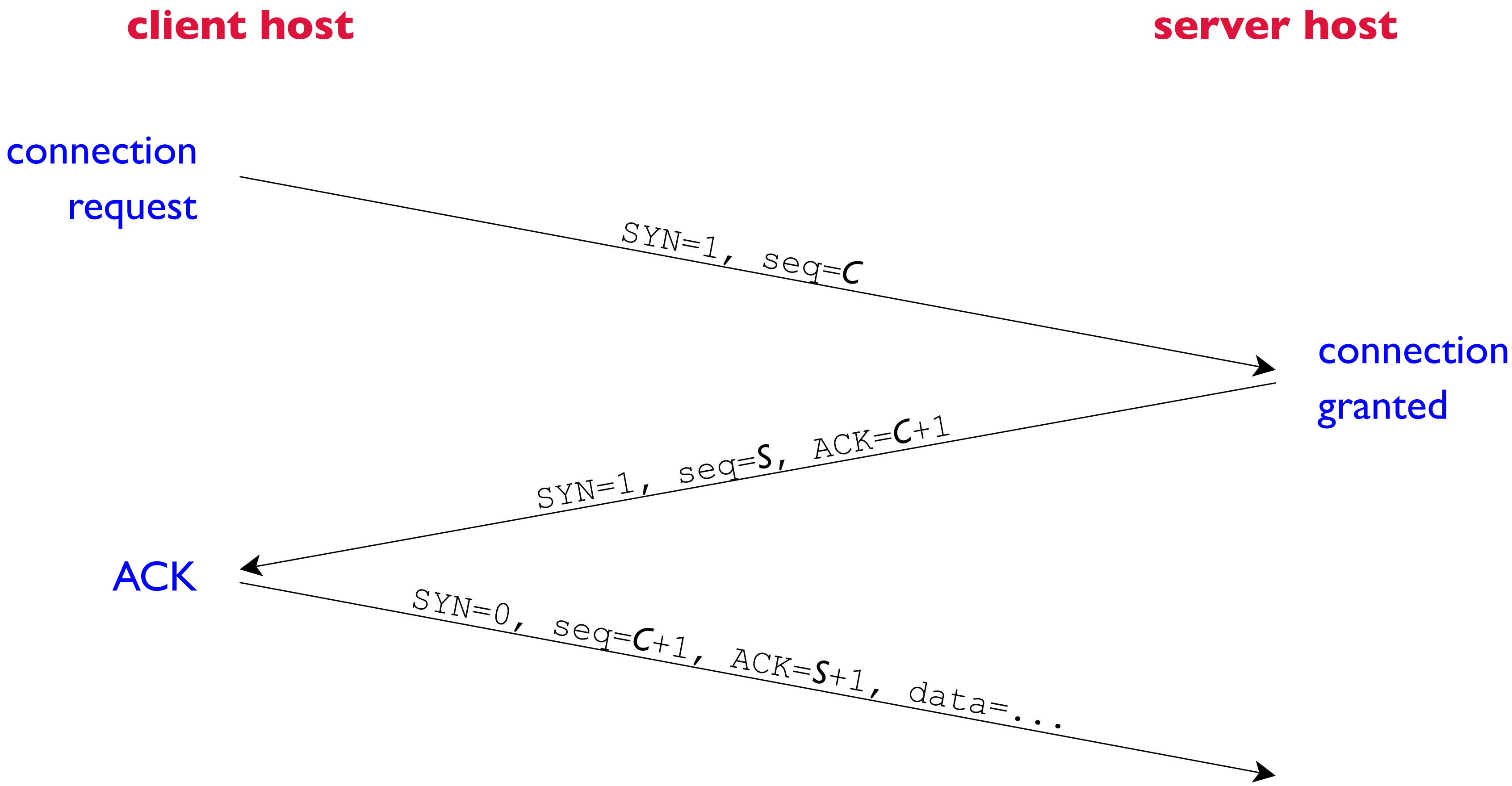
host B

seq=42, ACK=79, data="abc"

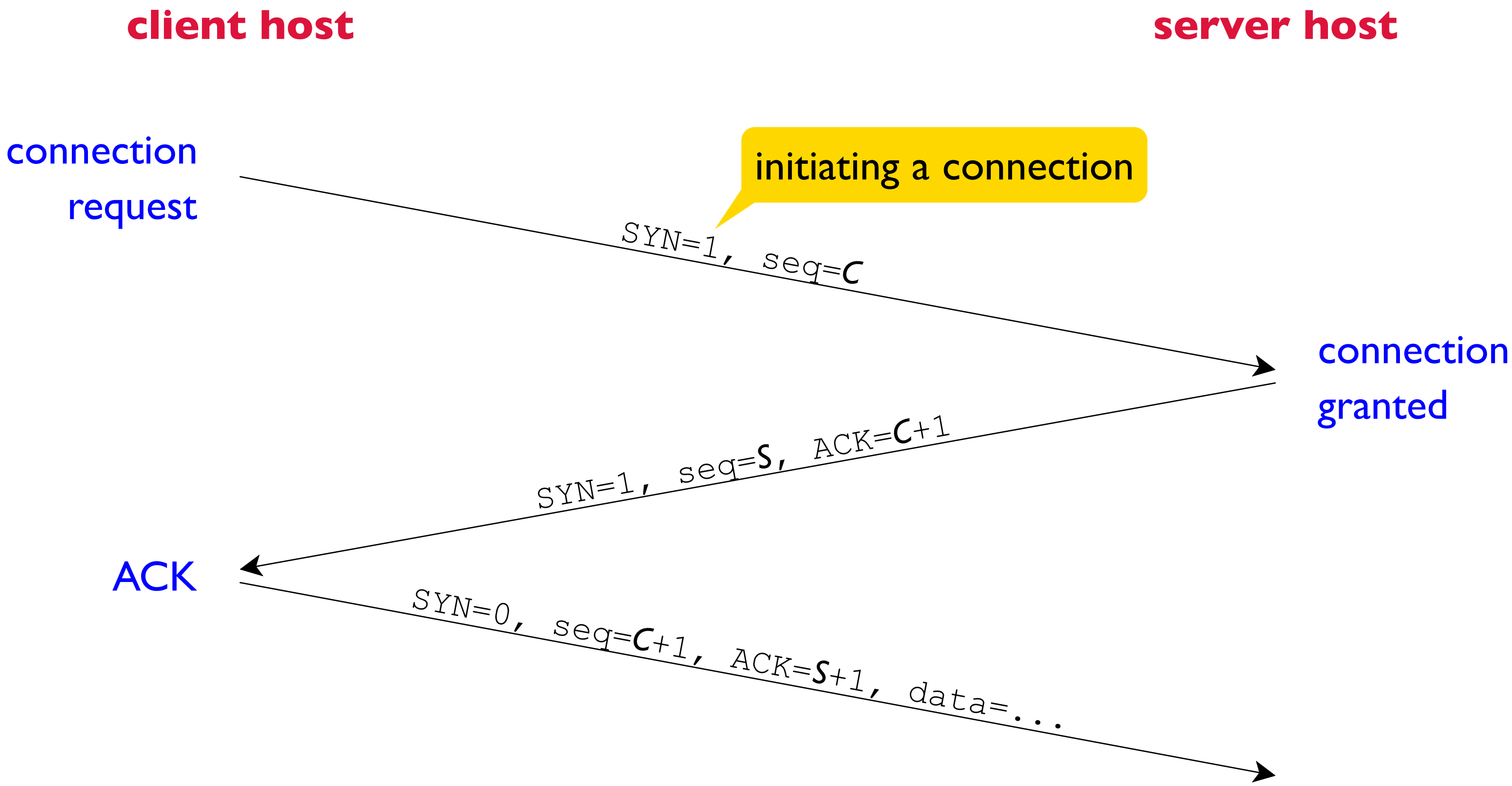
seq=79, ACK=45, data=""

seq=45, ACK=79, data="!"

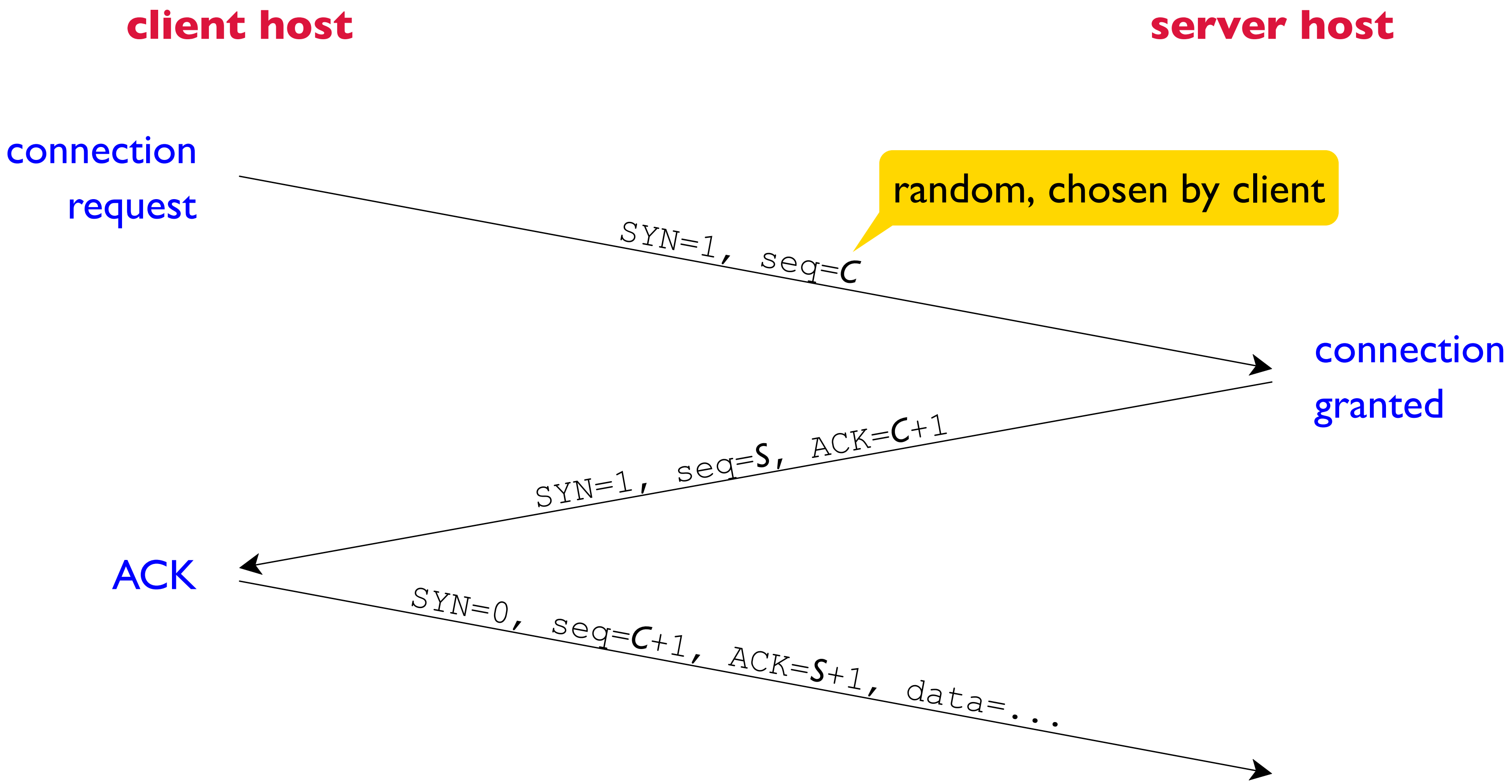
TCP Handshake: Initiating a Connection



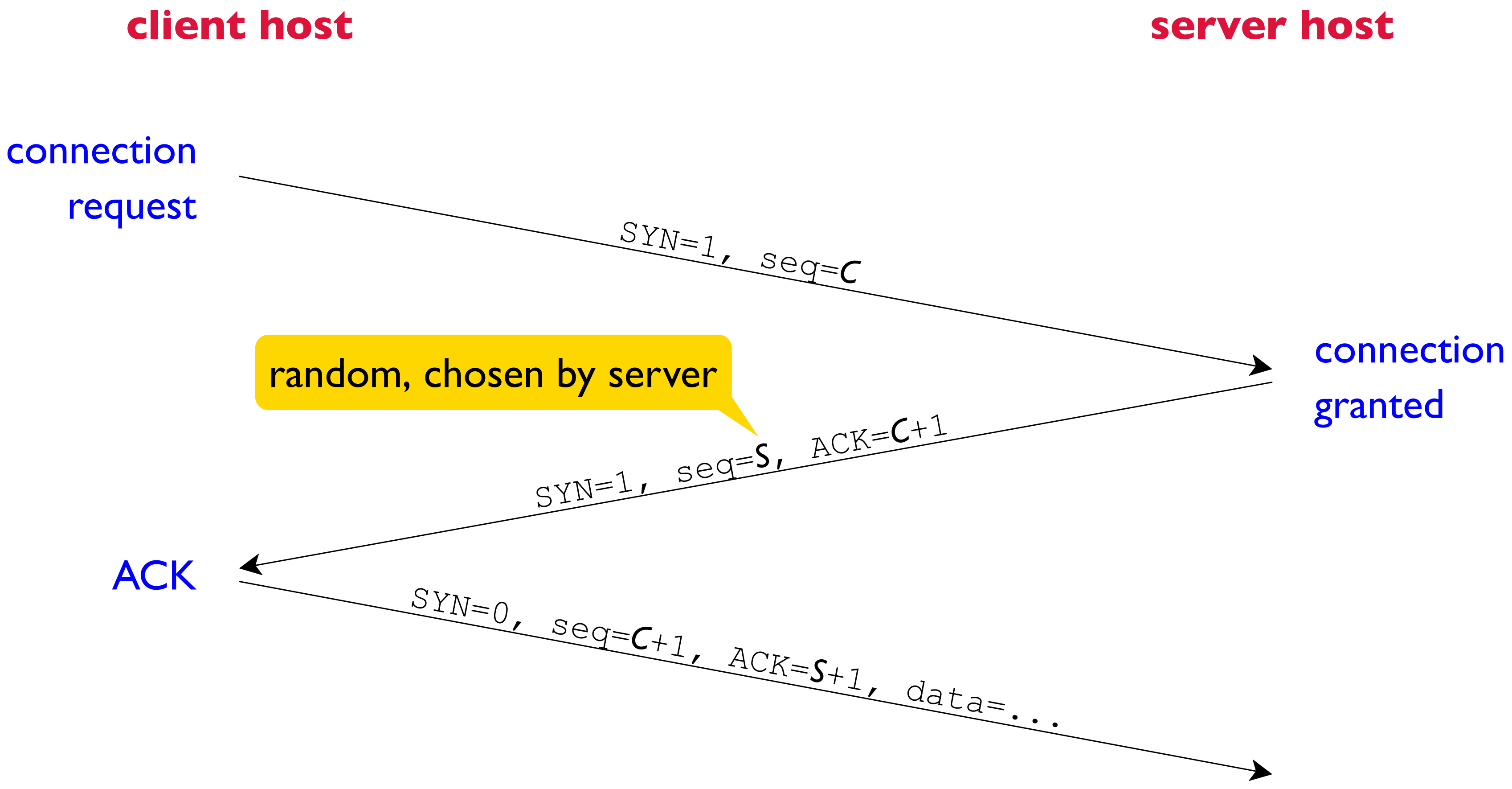
TCP Handshake: Initiating a Connection



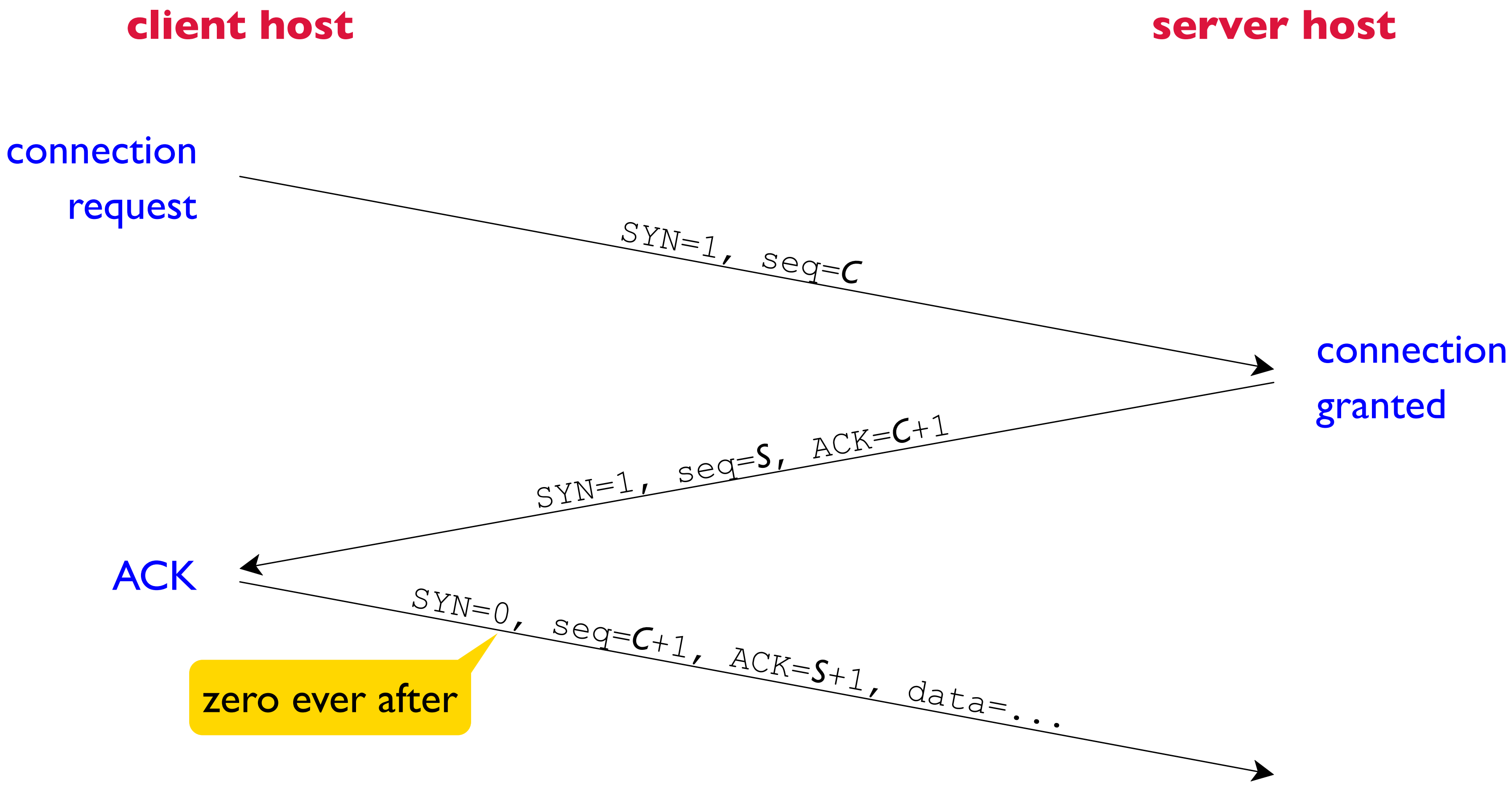
TCP Handshake: Initiating a Connection



TCP Handshake: Initiating a Connection



TCP Handshake: Initiating a Connection



client host

server host

connection request

SYN=1, seq=*C*

connection granted

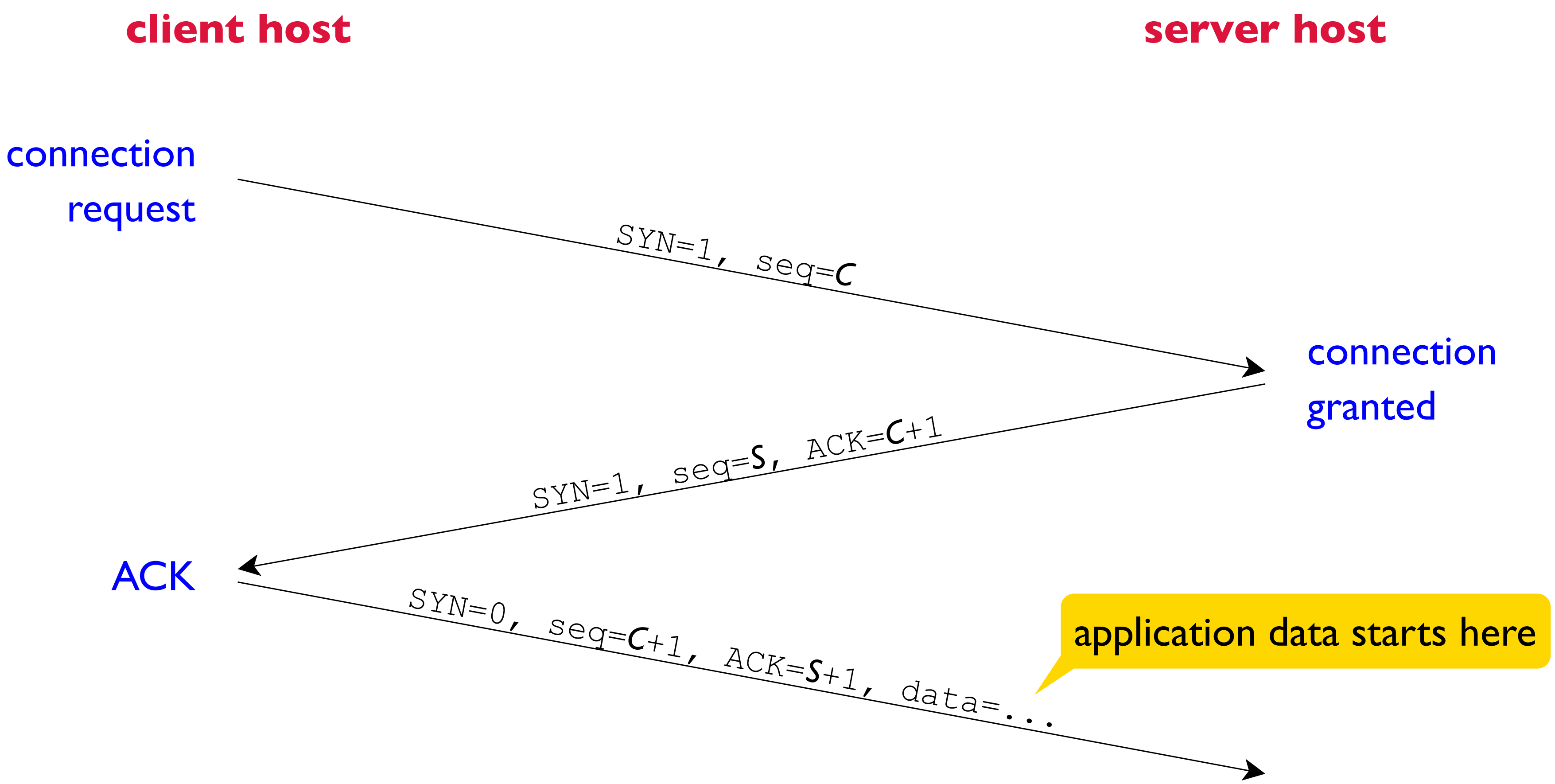
SYN=1, seq=*S*, ACK=*C*+1

ACK

SYN=0, seq=*C*+1, ACK=*S*+1, data=...

zero ever after

TCP Handshake: Initiating a Connection



client host

server host

connection request

SYN=1, seq=*C*

connection granted

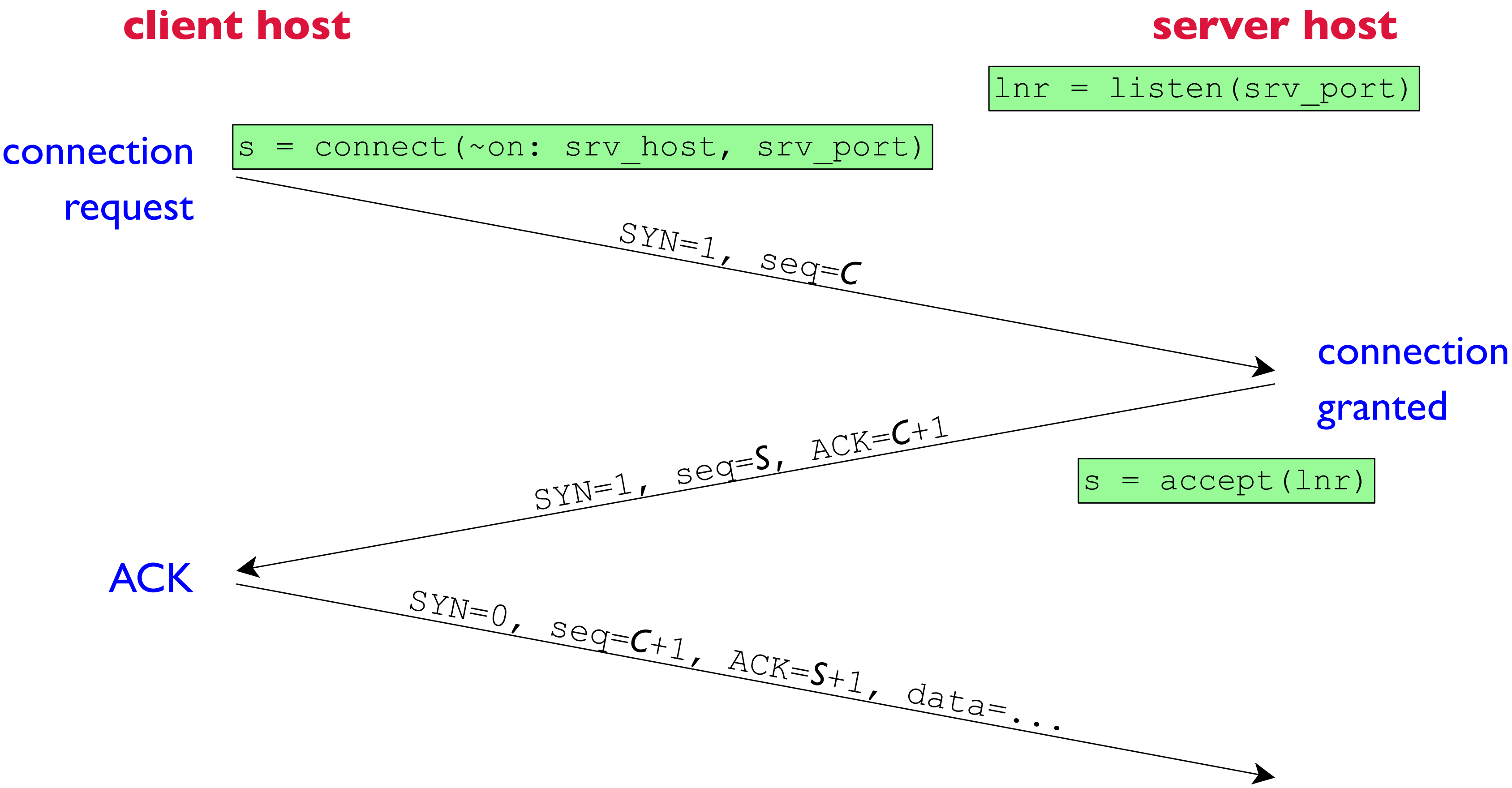
SYN=1, seq=*S*, ACK=*C*+1

ACK

SYN=0, seq=*C*+1, ACK=*S*+1, data=...

application data starts here

TCP Handshake: Initiating a Connection



client host

server host

```
lnr = listen(srv_port)
```

connection request

```
s = connect(~on: srv_host, srv_port)
```

SYN=1, seq=***C***

connection granted

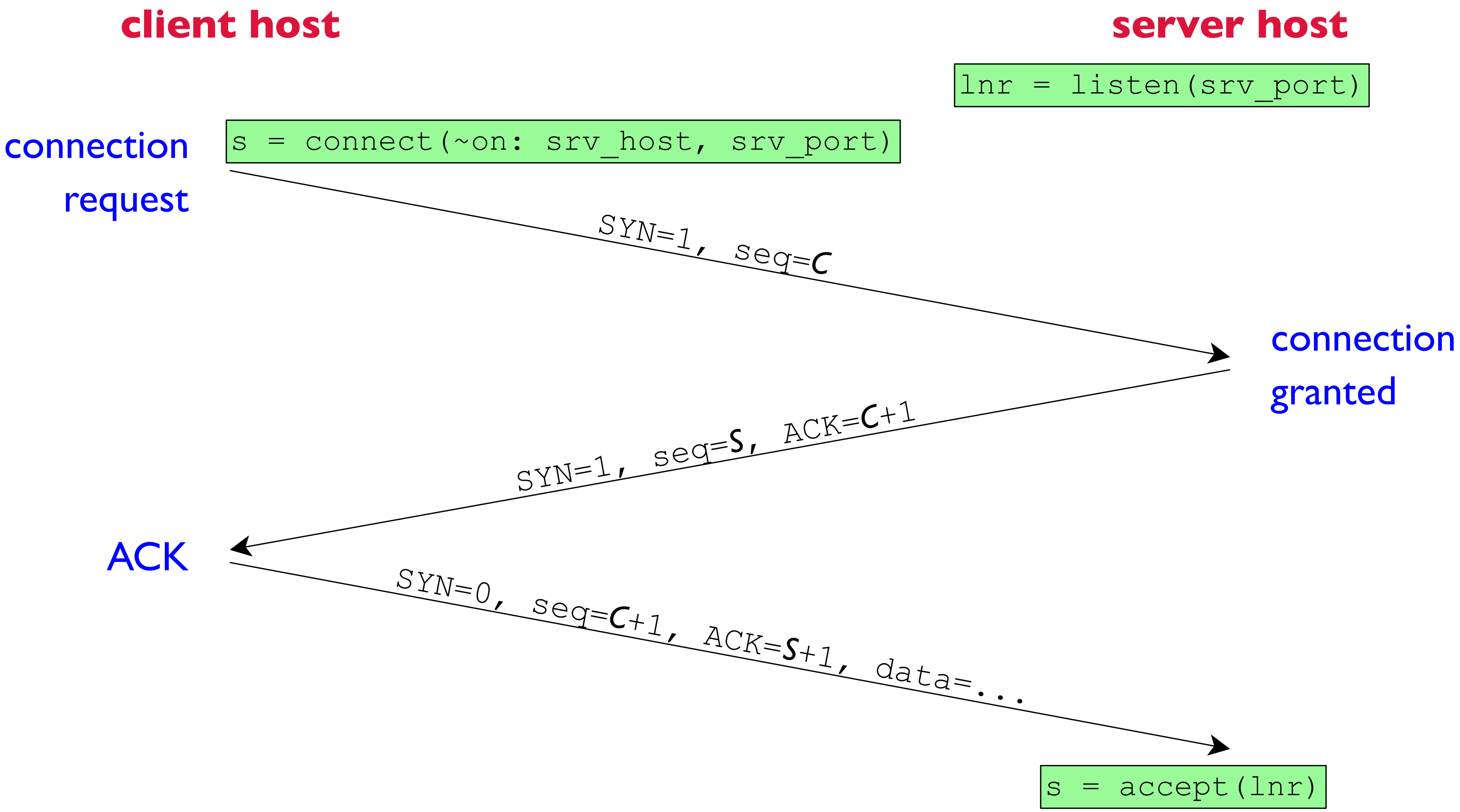
SYN=1, seq=***S***, ACK=***C***+1

```
s = accept(lnr)
```

ACK

SYN=0, seq=***C***+1, ACK=***S***+1, data=...

TCP Handshake: Initiating a Connection



client host

server host

```
lnr = listen(srv_port)
```

connection request

```
s = connect(~on: srv_host, srv_port)
```

SYN=1, seq=*C*

connection granted

SYN=1, seq=*S*, ACK=*C*+1

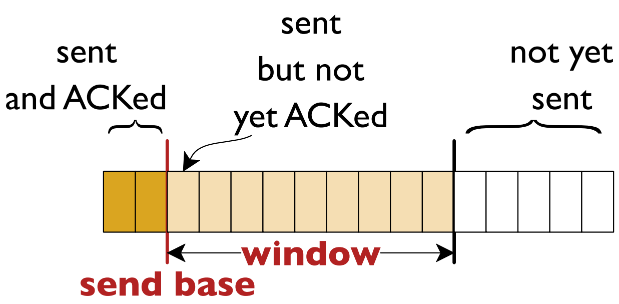
ACK

SYN=0, seq=*C*+1, ACK=*S*+1, data=...

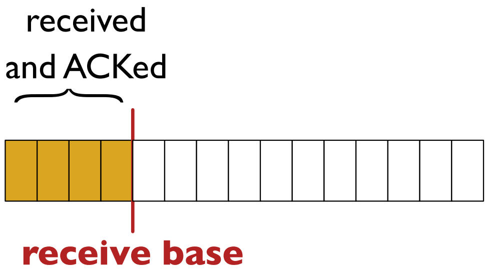
```
s = accept(lnr)
```

Buffers and Flow Control

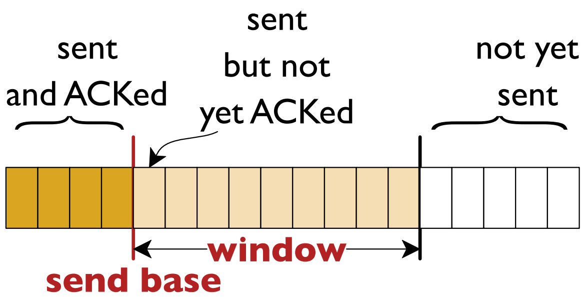
sending side of client



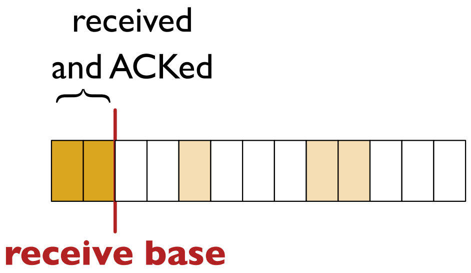
receiving side of client



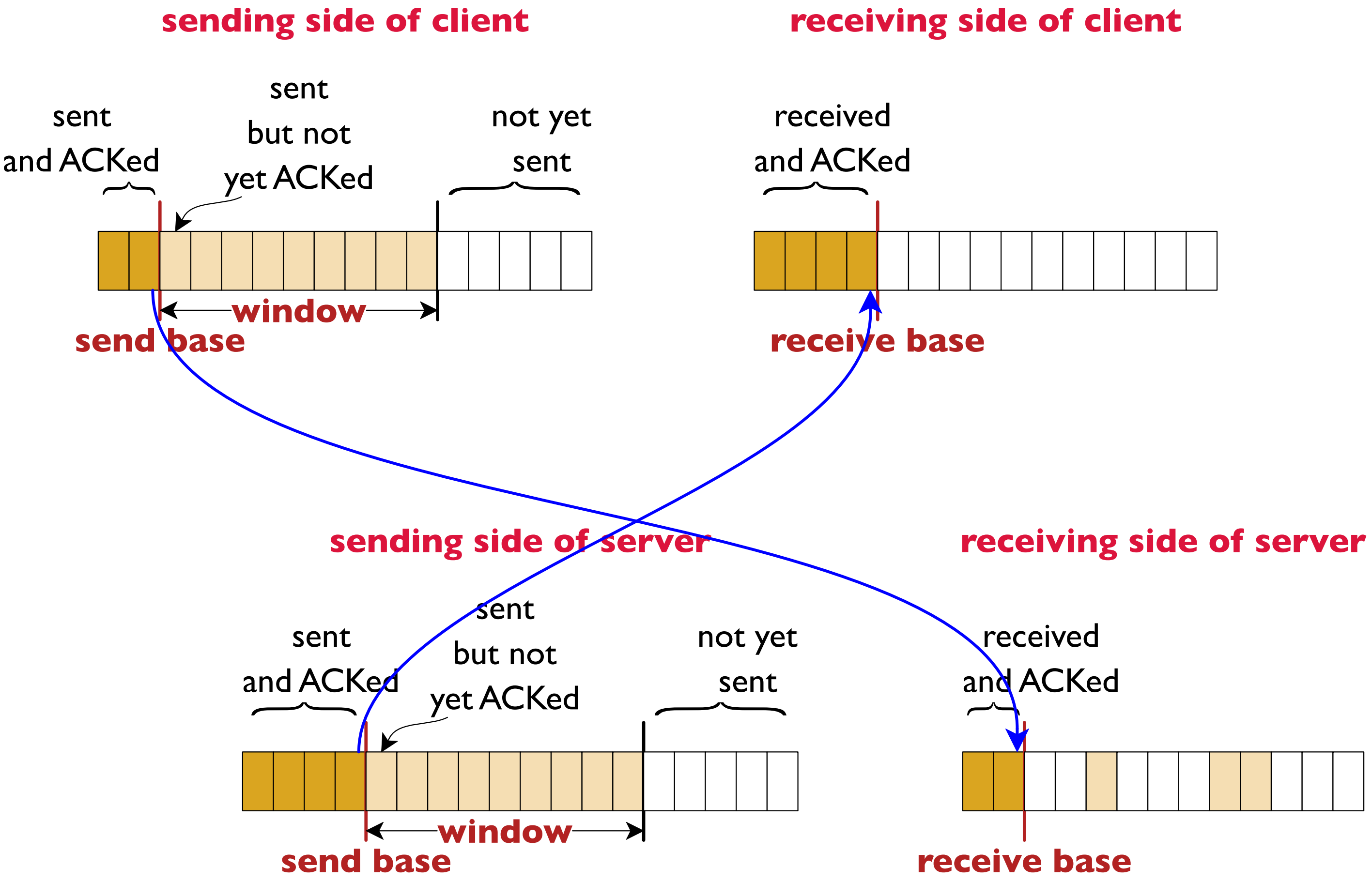
sending side of server



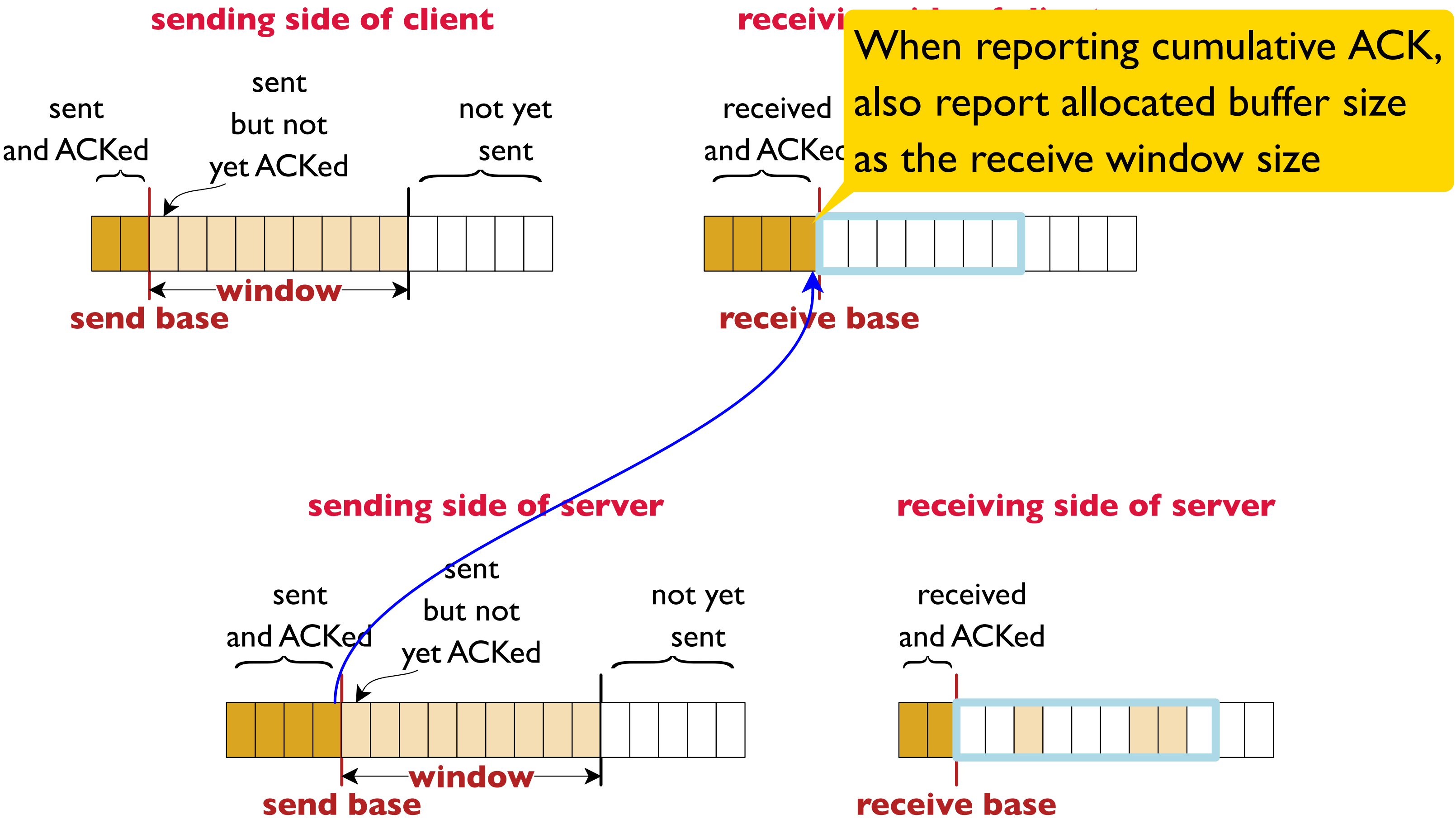
receiving side of server



Buffers and Flow Control

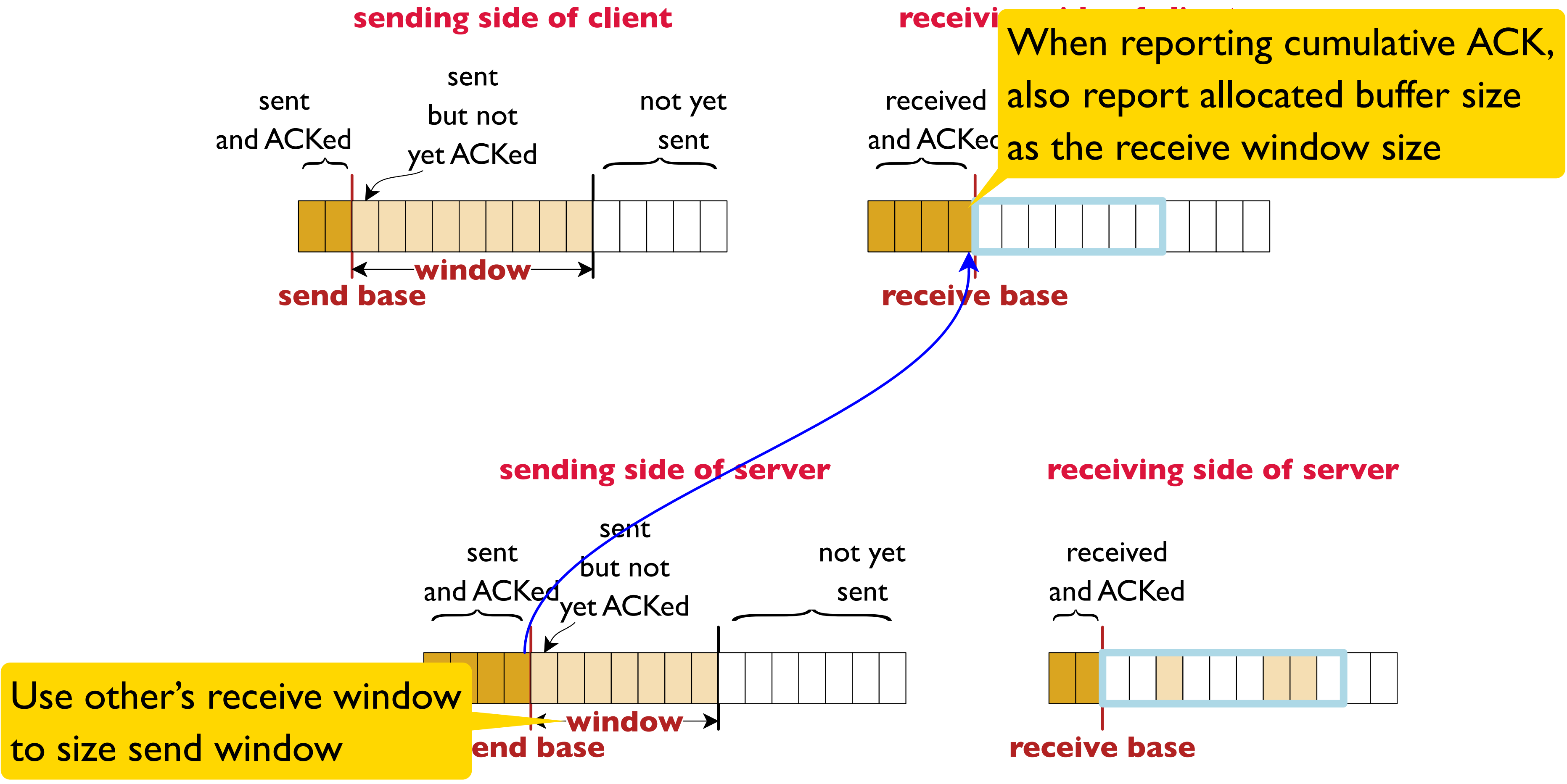


Buffers and Flow Control



When reporting cumulative ACK, also report allocated buffer size as the receive window size

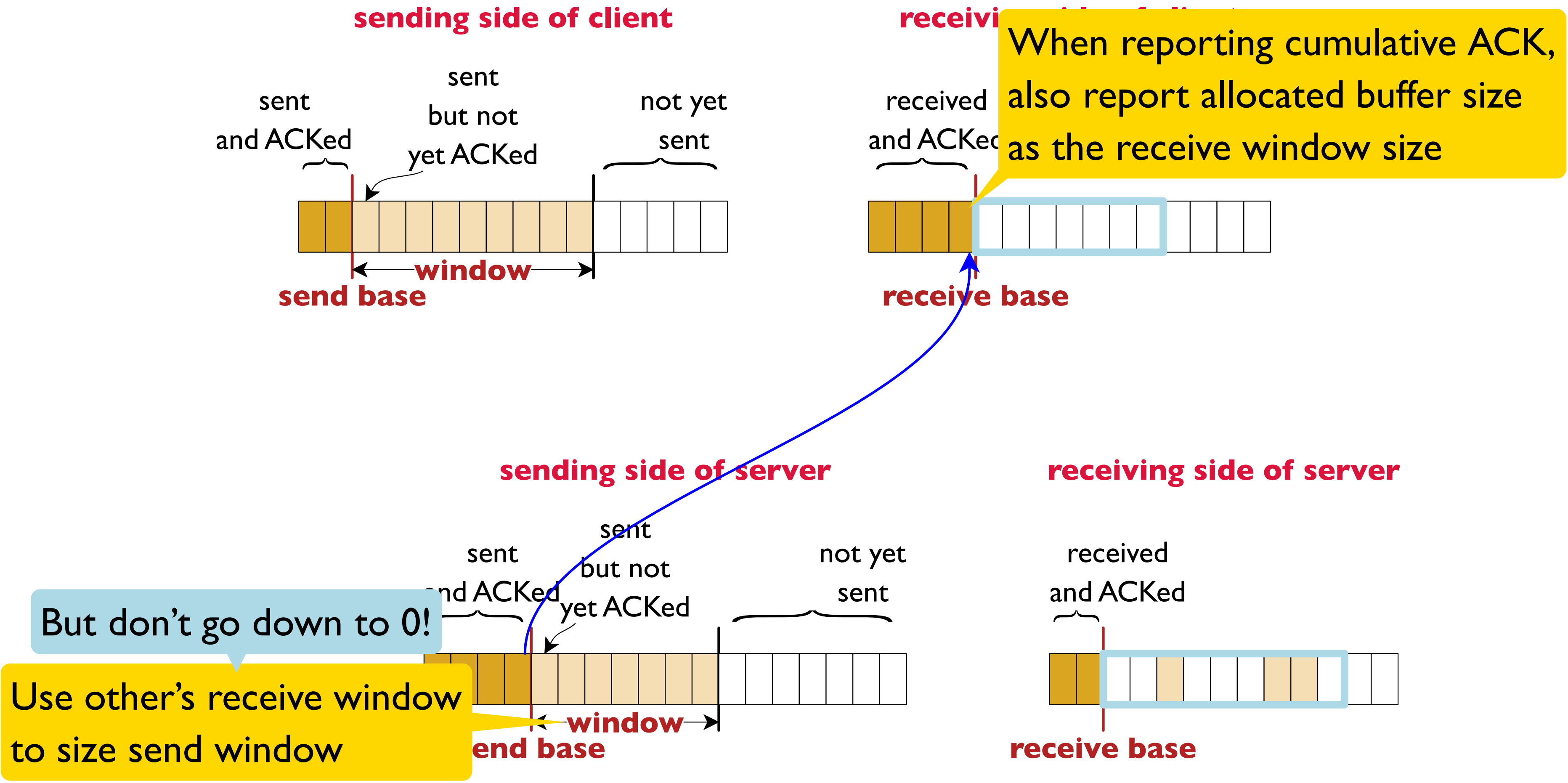
Buffers and Flow Control



When reporting cumulative ACK, also report allocated buffer size as the receive window size

Use other's receive window to size send window

Buffers and Flow Control



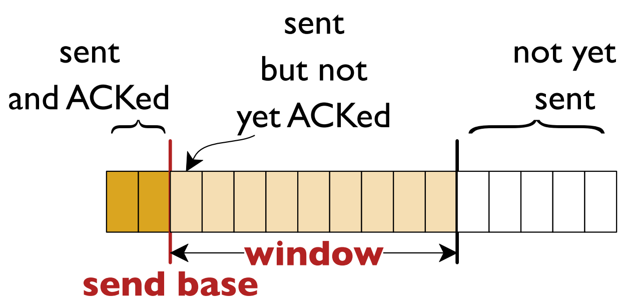
When reporting cumulative ACK, also report allocated buffer size as the receive window size

But don’t go down to 0!

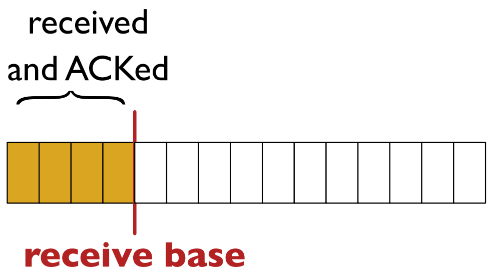
Use other’s receive window to size send window

Out-of-Order ACK Policy

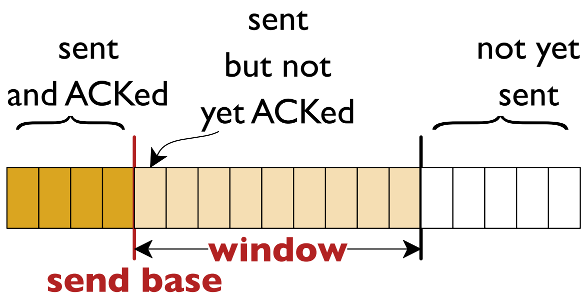
sending side of client



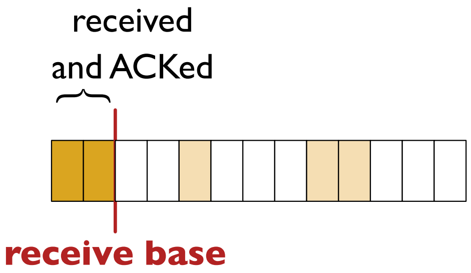
receiving side of client



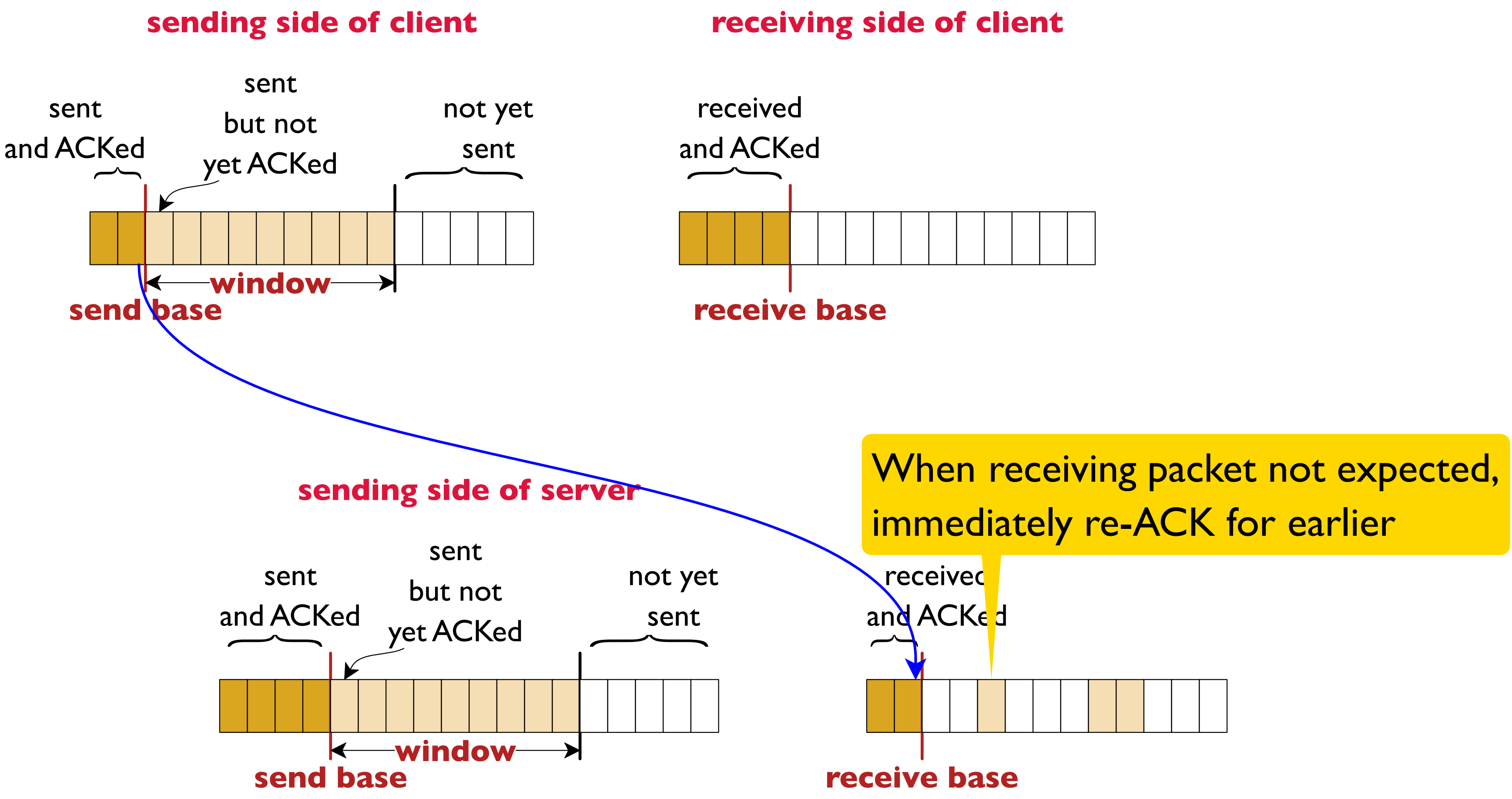
sending side of server



receiving side of server

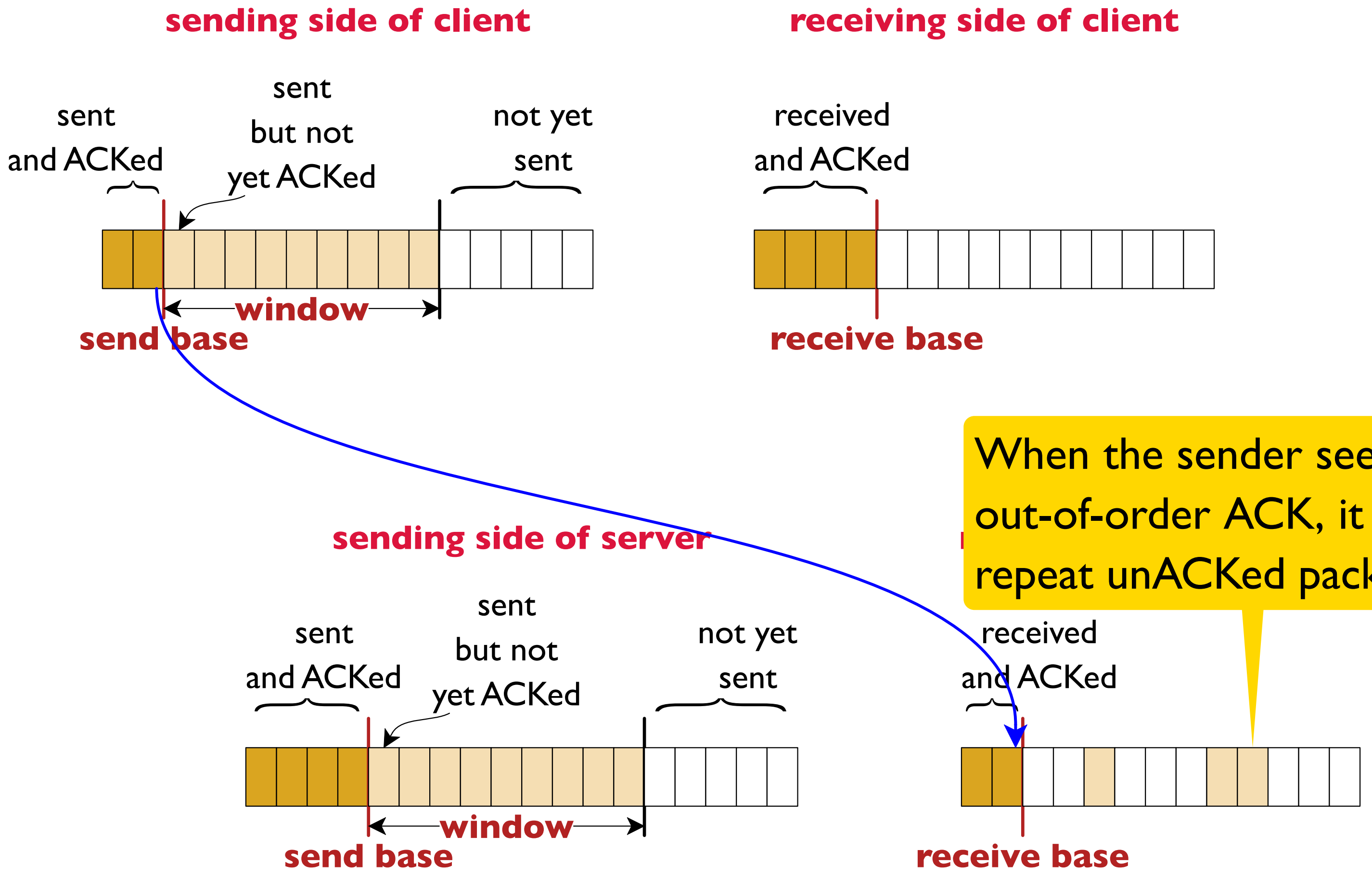


Out-of-Order ACK Policy



When receiving packet not expected, immediately re-ACK for earlier

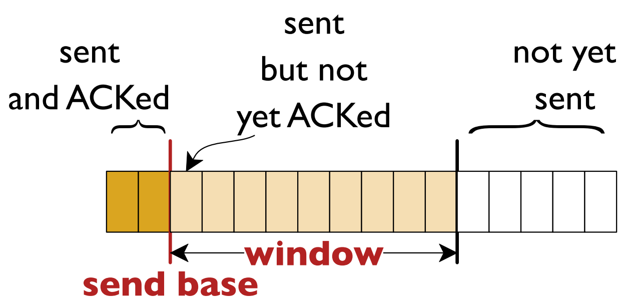
Out-of-Order ACK Policy



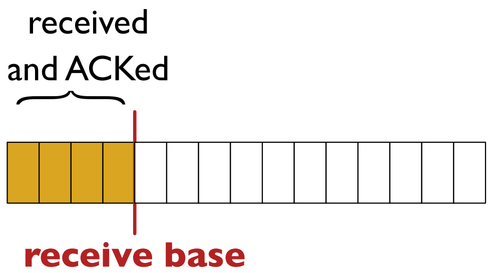
When the sender see this third, out-of-order ACK, it will immediately repeat unACKed packets

In-Order ACK Policy

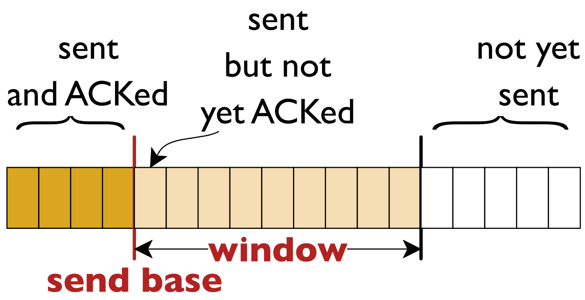
sending side of client



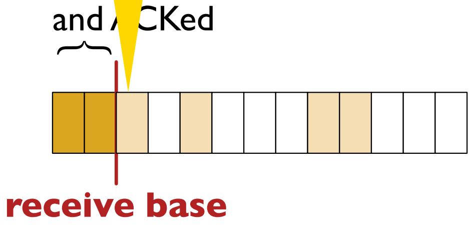
receiving side of client



sending side of server

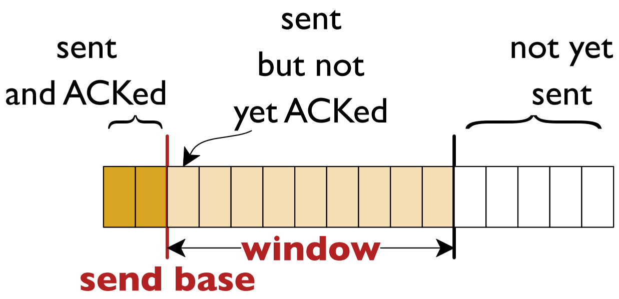


When an in-order packet is received, wait a little while, in case the ACK can cover more

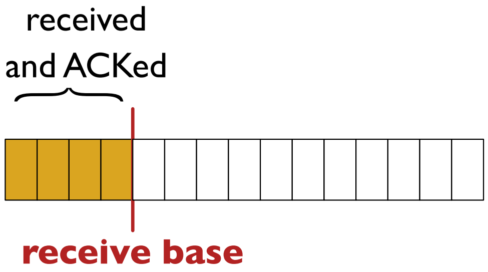


Timeout Policy

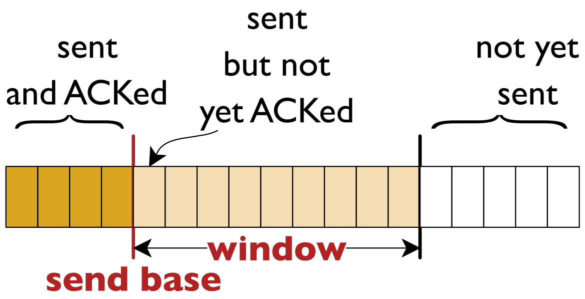
sending side of client



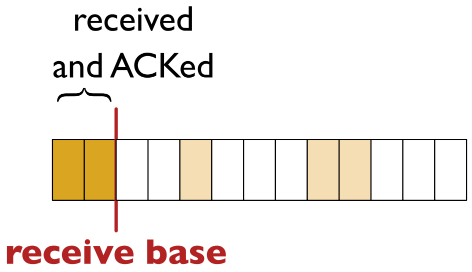
receiving side of client



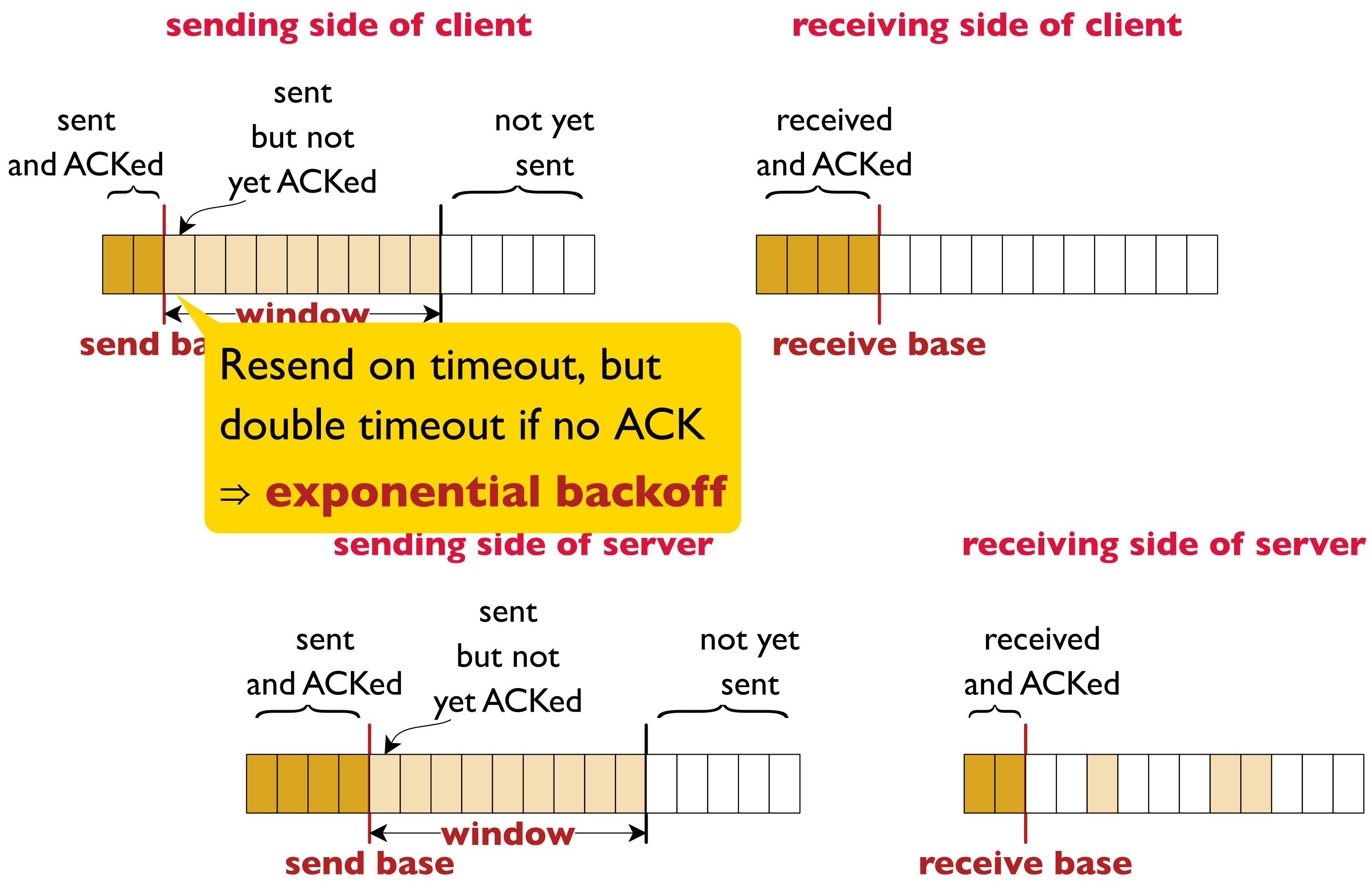
sending side of server



receiving side of server



Timeout Policy



Resend on timeout, but double timeout if no ACK

⇒ **exponential backoff**

Closing TCP Connections

Each send end of a connection can be closed separately

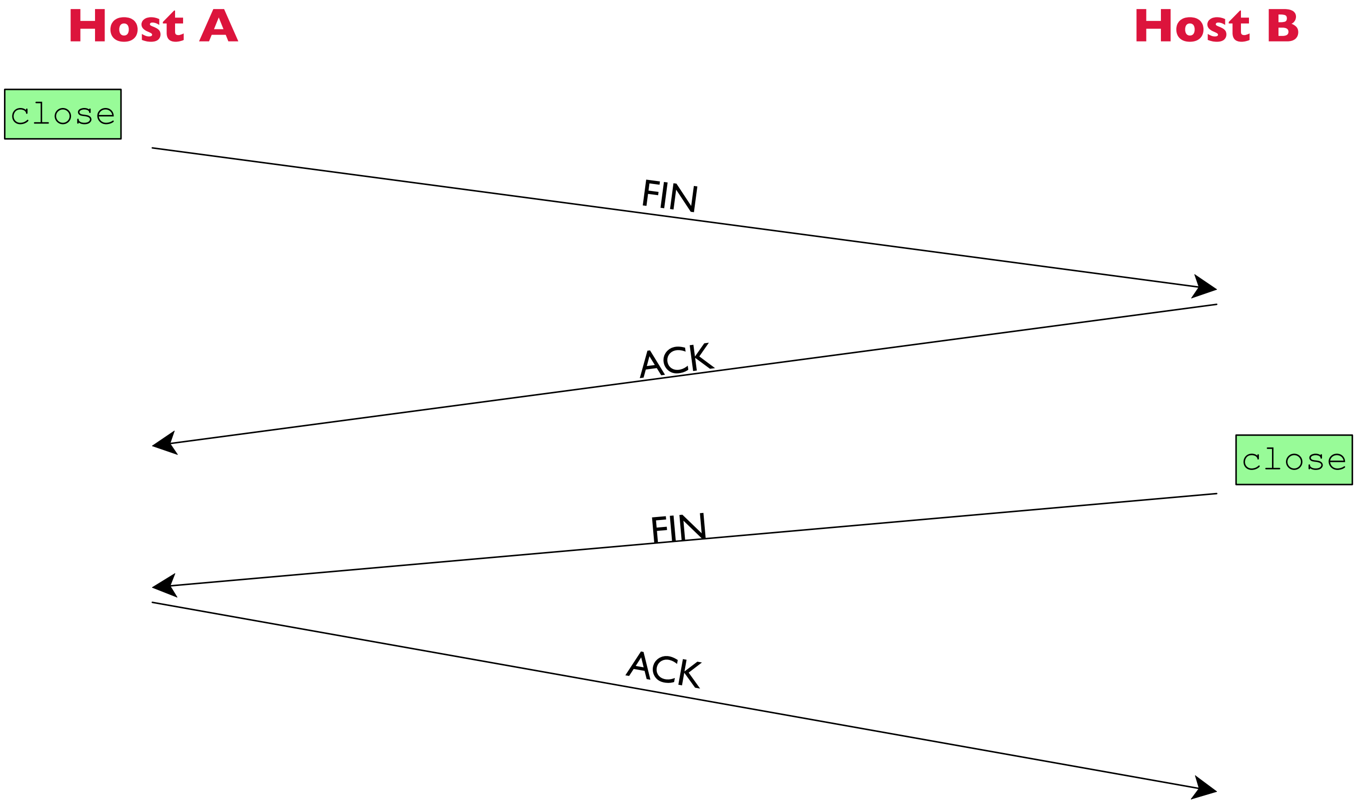
The `shutdown` sustem call can close only one direction of a socket, while `close` closes both

When a sending end is closed, the other host's receive end produces EOF — but new data still can be sent the other way

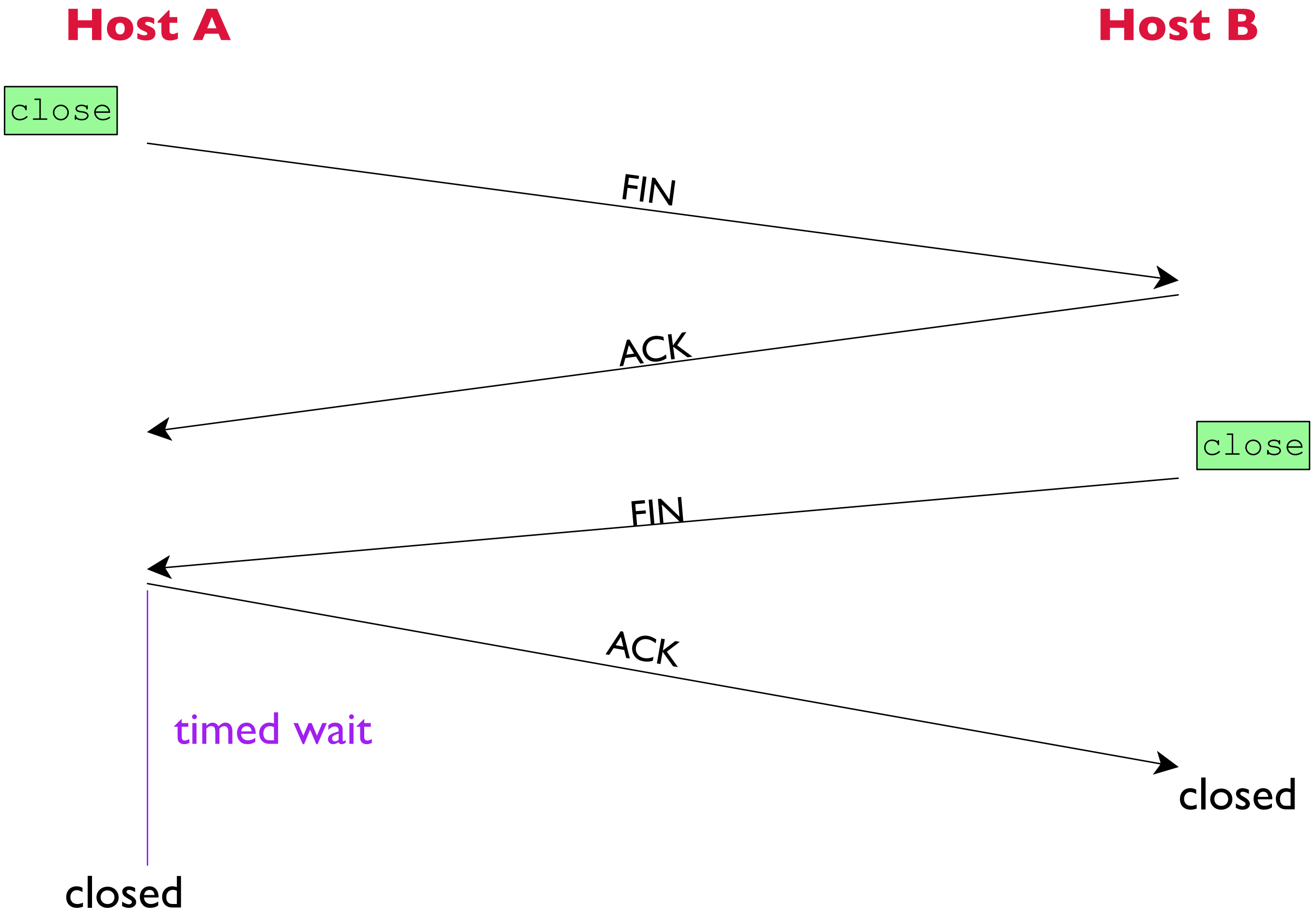
The connection terminates only after all send ends are closed

The OS socket representing a connection stays allocated until both the send and receive parts are closed

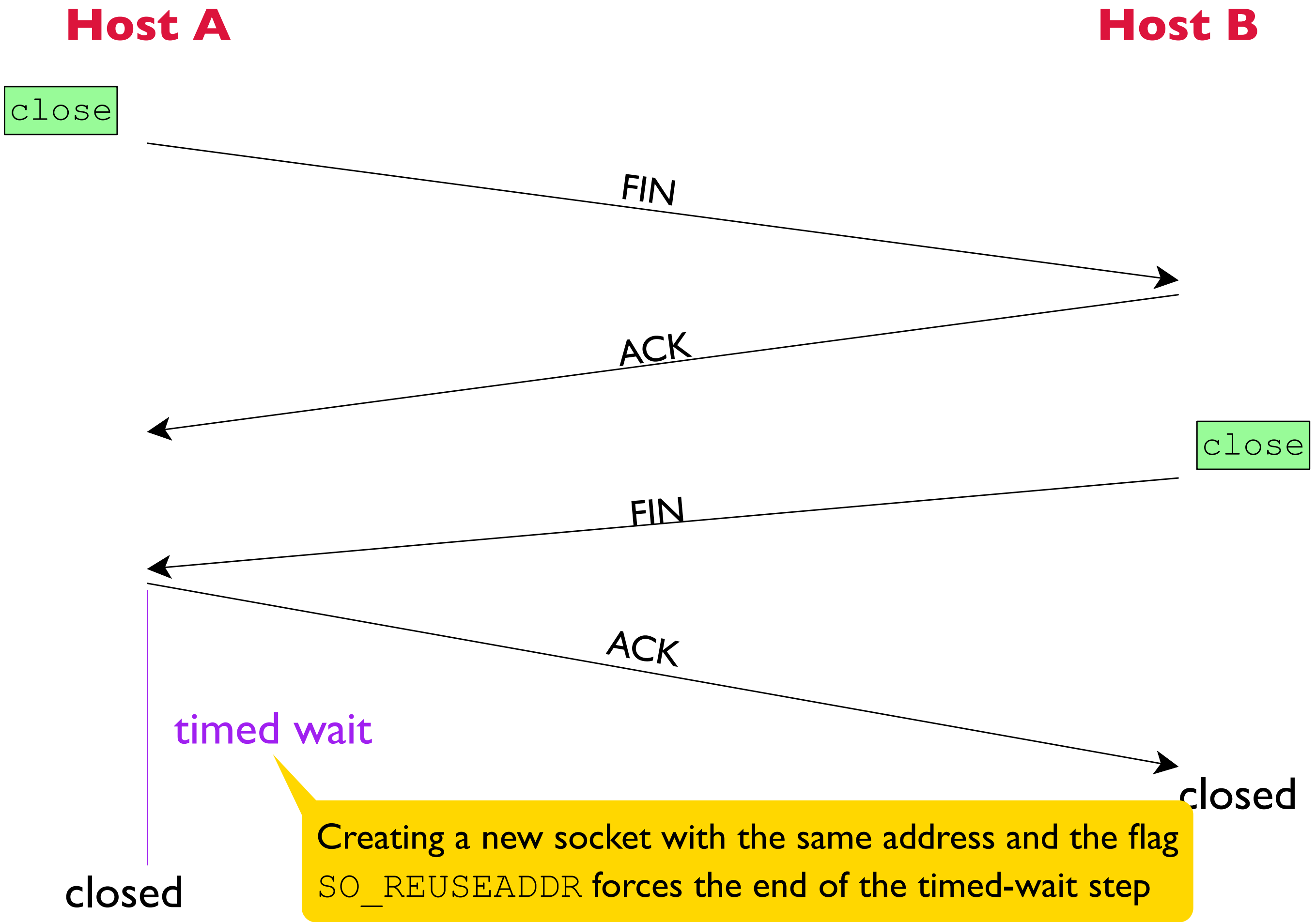
Closing TCP Connections



Closing TCP Connections



Closing TCP Connections



Creating a new socket with the same address and the flag `SO_REUSEADDR` forces the end of the timed-wait step

Summary

TCP: **connection-oriented** and **full duplex**

- server **listens** at a port number
- client **connects** a socket to that a port number
- server **accepts** a socket from the listener

In a TCP packet:

- **Sequence numbers** and **acknowledgment numbers** implement cumulative acknowledgments
- **Window** sizes enable flow control

Setup with `SYN` `ACK`, teardown with `FIN` `ACK`