THE UNIVERSITY OF
MELBOURNE

COMP30023 – Computer Systems

Socket Programming and TCP flow control

Dr Lachlan Andrew

28/04/22

Recap

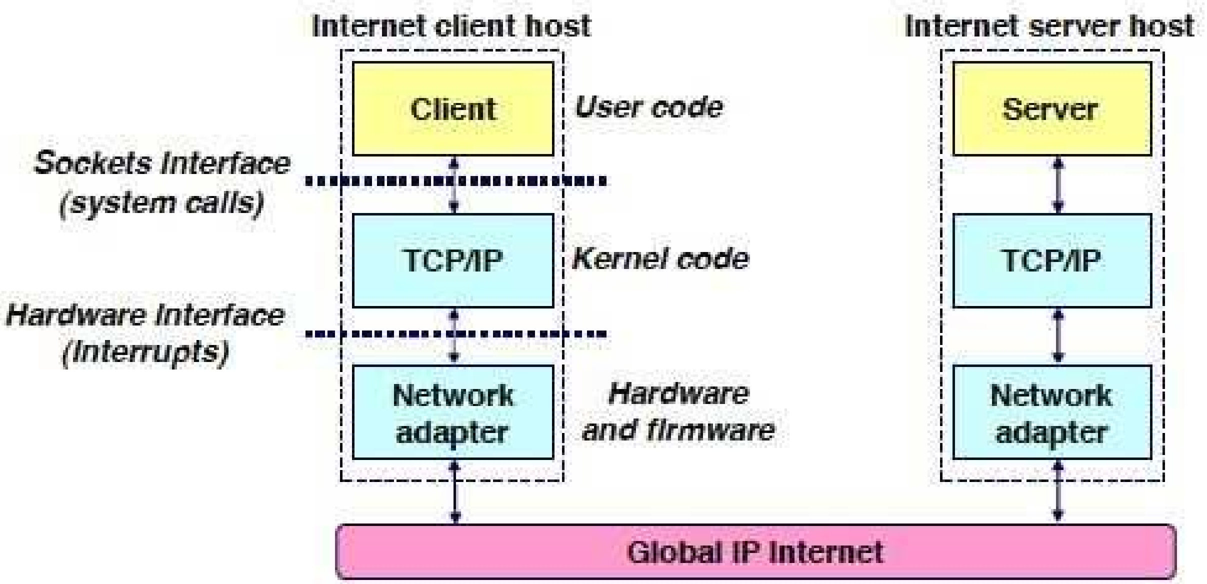
- TCP
  - Connection
  - Sending Data
  - Disconnection

Summary

- High level overview of socket programming
  - Relationship to TCP
  - Sockets in C

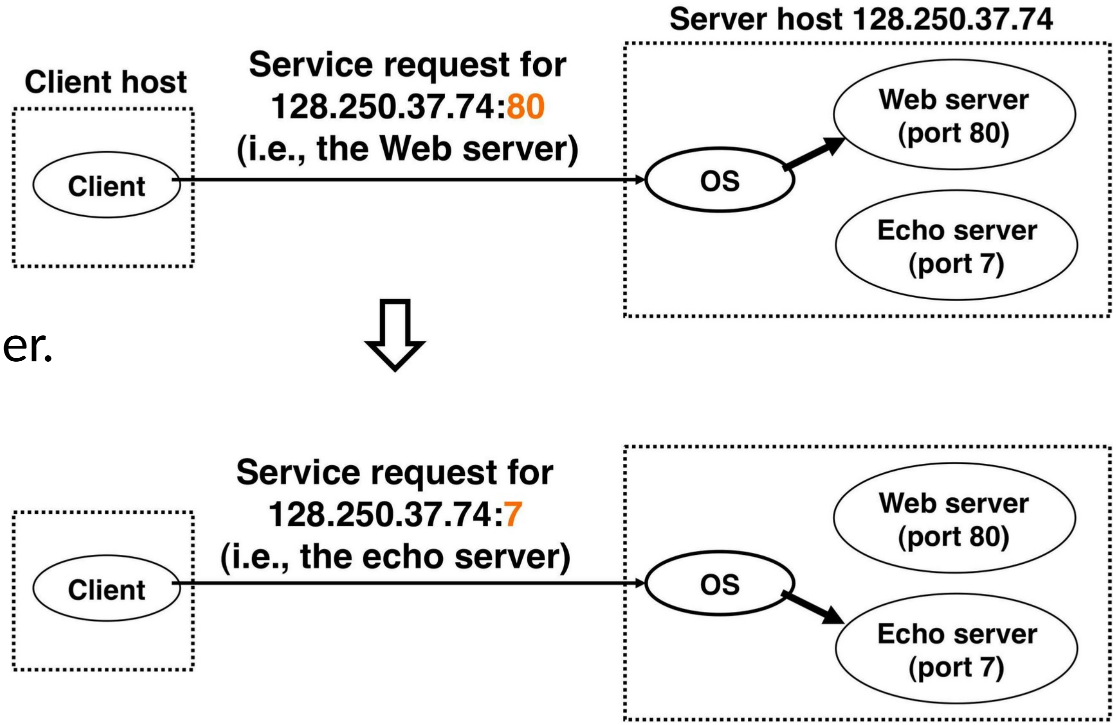
Sockets

- A message going from one host to another must cross the underlying network.
- A process sends and receives through a socket
  - the “doorway” leading in/out of the application



Using sockets

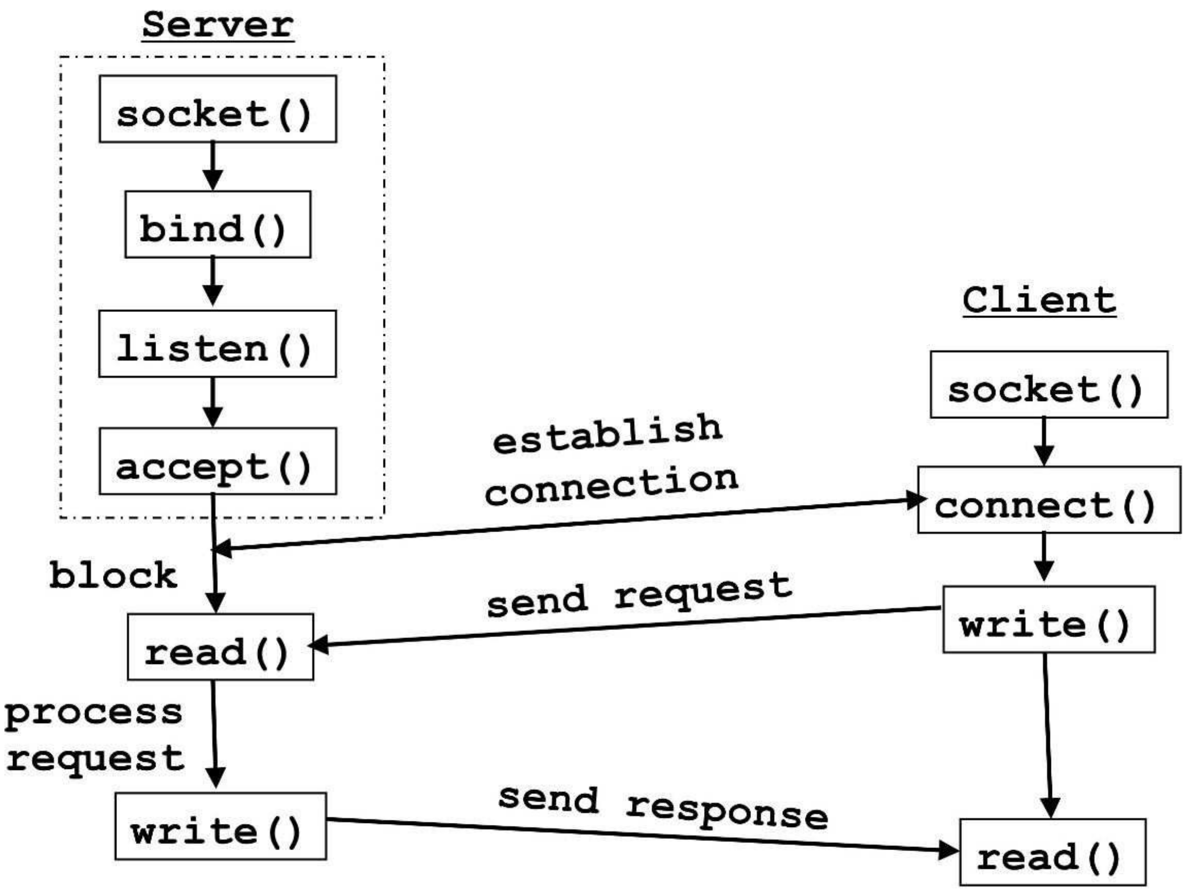
- The “address” of a socket is the 5-tuple:
  - Protocol
  - source-IP
  - source-port number
  - destination-IP
  - destination-port number.



Berkeley Sockets

- Socket interface
  - originally provided in Berkeley UNIX
  - later adopted by all popular operating systems
  - simplifies porting applications to different OSes
- In UNIX, everything is like a file
  - all input is like reading a file
  - all output is like writing a file
  - file is “addressed” by an integer file descriptor
- API implemented as system calls:
  - examples include connect(), read(), write(), close()

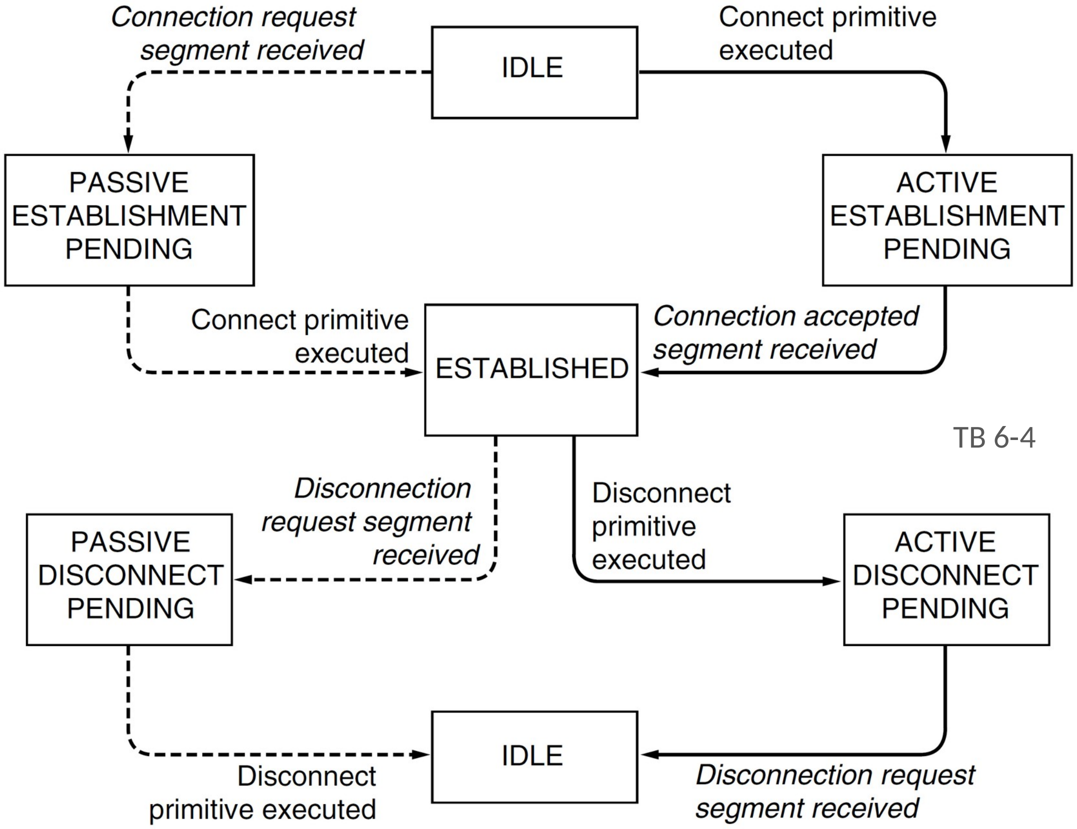
Using sockets



Socket Primitives

State	Description
SOCKET	Creates a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection (block until then)
CONNECT	Actively attempt to establish a connection
SEND	Send some data over a connection (write())
RECEIVE	Receive *some* data from the connection (read())
CLOSE	Release the connection

Simplified (6-)State diagram for Connection Management


IDLE
Connection request segment received
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted segment received
Disconnection request segment received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
Disconnect primitive executed
IDLE
Disconnection request segment received

TB 6-4

Complete list of Socket States

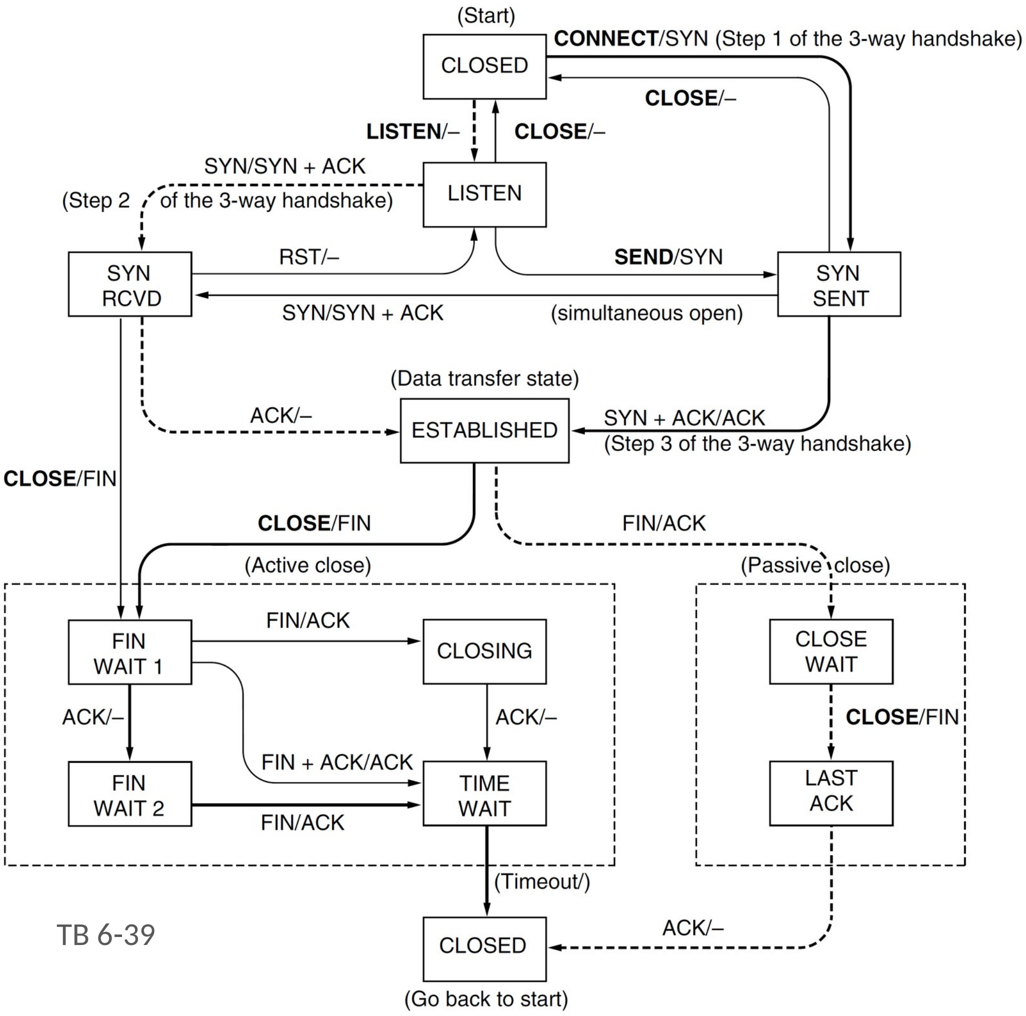
State	Simplified name	Description
CLOSED	Idle	No connection is active or pending
LISTEN	Pass. est.	The server is waiting for an incoming call
SYN RCVD	Pass. est.	A connection request has arrived; wait for ACK
SYN SENT	Act. est.	The application has started to open a connection
ESTABLISHED	Established	The normal data transfer state
FIN WAIT 1	Act. disc.	The application has said it is finished
FIN WAIT 2	Act. disc.	The other side has agreed to release
TIME WAIT	Act. disc.	Wait for all packets to die off
CLOSING	Act. disc.	Both sides have tried to close simultaneously
CLOSE WAIT	Pass. disc.	The other side has initiated a release
LAST ACK	Pass. disc.	Wait for all packets to die off

Socket Finite State Machine

Bold before slash:
System call
e.g., connect

Non-bold before slash:
Packet received
e.g., SYN

After slash:
Packet sent
e.g., SYN



TB 6-39

Sockets in C

- Headers

```
#define _POSIX_C_SOURCE 200112L // Required for VSCode
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
```

- Variables

```
int listenfd = 0, connfd = 0, re = 1, s, n;
char sendBuff[1024];
struct addrinfo hints, *res, *rp;
```

- Create a socket

```
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
s = getaddrinfo(NULL, "5000", &hints, &res);
listenfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &re, sizeof(re));
```

AF_INET6

Create IPv6 socket

- getaddrinfo() can return multiple addresses as a linked list
- If you want the IPv6 address in particular, you may have to loop over the responses:

```
for (p = res; p!= NULL; p = p->ai_next) {
   if (p->ai_family == AF_INET6
          && (sock = socket(p->ai_family,
                            p->ai_socktype,
                            p->ai_protocol)) < 0 )
   {
      // socket creation was attempted but failed
   }
}
// Use  sock  here if it is >= 0.
```

Sockets in C - server

- Bind and listen

```
bind(listenfd, res->ai_addr, res->ai_addrlen);
// maximum number of client connections to queue
listen(listenfd, 10);
```

p of loop

Put TCP state machine in LISTEN state

- Accept, write/send, close

```
struct sockaddr_storage client_addr;
socklen_t client_addr_size = sizeof client_addr;
connfd = accept(listenfd, (struct sockaddr*)&client_addr,
&client_addr_size);
snprintf(sendBuff, sizeof(sendBuff), "Hello World!\n");
n = write(connfd, sendBuff, strlen(sendBuff));
close(connfd);
```

Block

- Wait until one of several files is ready to read / write
  - `select(), pselect(), poll()`

Sockets in C - client

- Connect

```
// Same as server, without hints.ai_flags
s = getaddrinfo("127.0.0.1", "5000", &hints, &res);
for (rp = res; rp != NULL; rp = rp->ai_next) {
    connfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
    if (connfd == -1) continue;
if (connect(connfd, rp->ai_addr, rp->ai_addrlen) != -1) break;
    close(connfd);
}
```

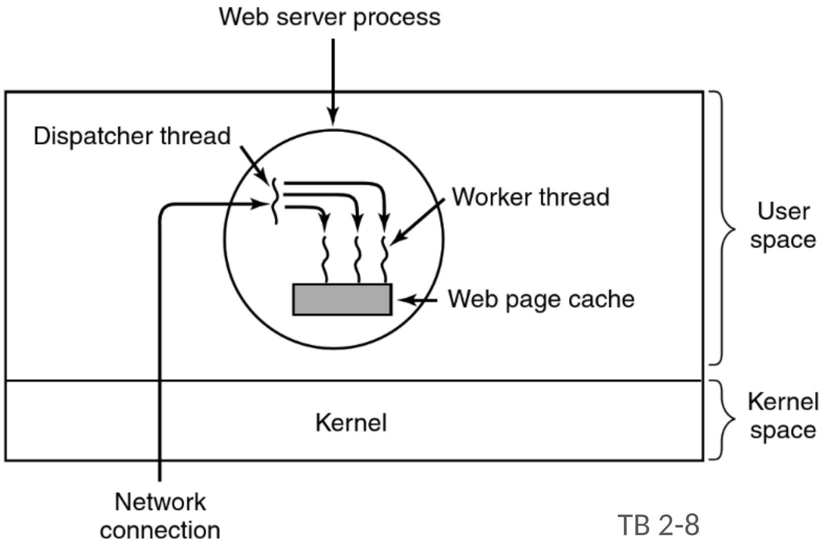
- Receive

```
while ((n = read(connfd, recvBuff, sizeof(recvBuff)-1)) > 0) {
    // process received buffer
}
```

  - If the socket is blocking (see fcntl and O_NONBLOCK), it waits until there is data
    - This loop reads the whole connection
  - If non-blocking, this just reads data that has arrived
    - More may come after a delay

Multi-threaded Web Server

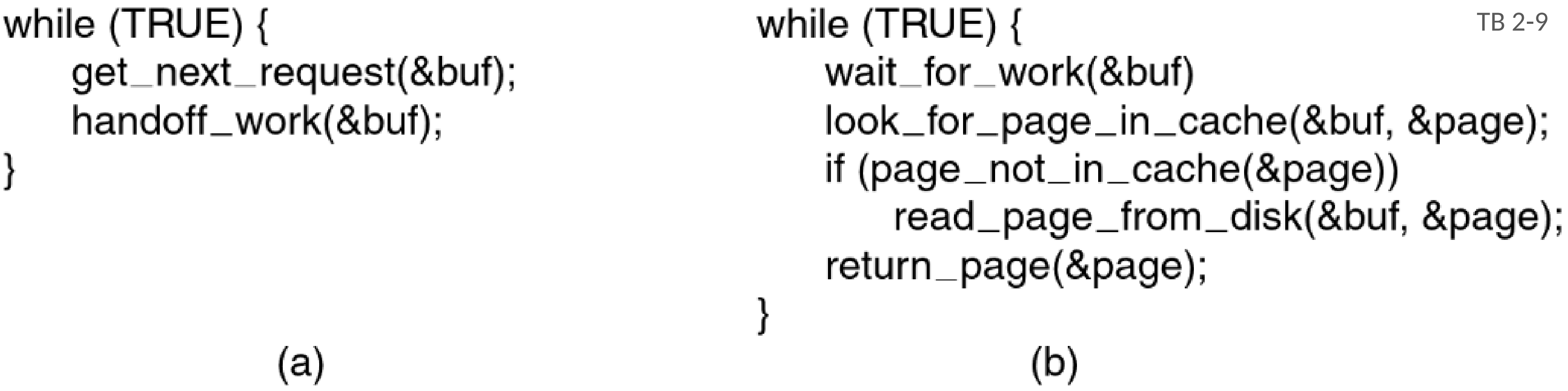
- Clearly a web server needs to be able to handle concurrent connections from multiple clients
- This can be achieved through the usage of a multi-threaded web server



TB 2-8

Multi-threaded web server

TB 2-9



```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

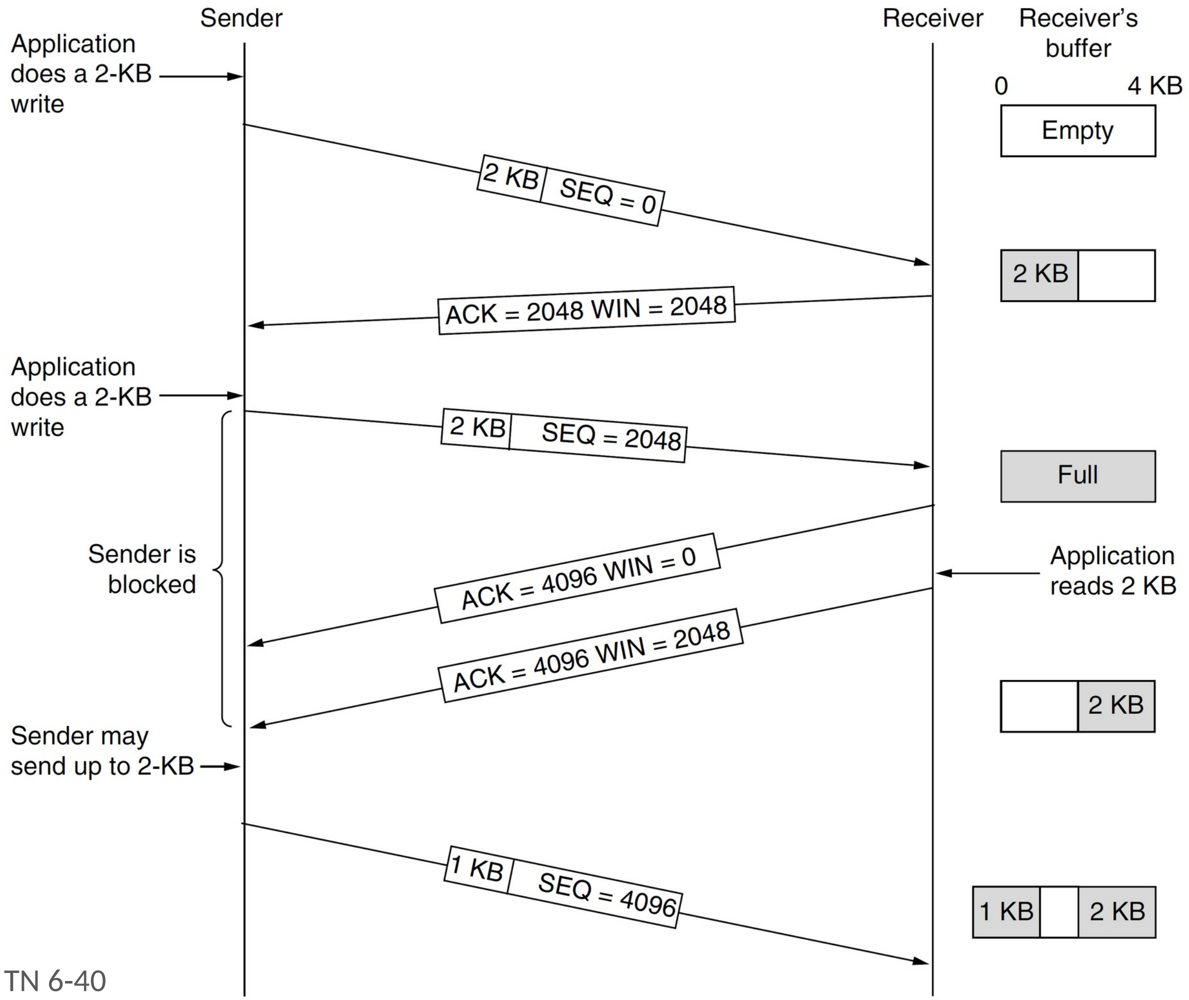
(b)

- High level outline of code for previous slide:
  - Dispatcher thread
  - Worker thread

TCP Sliding Window

- Sliding window is controlled by receiver
- Determines amount of data the receiver is able to accept
  - Sender and receiver maintain buffers to send and receive data independently of the application
  - No guarantee that data is immediately sent or read from the respective buffers

TCP Sliding Window



TN 6-40

TCP Sliding Window

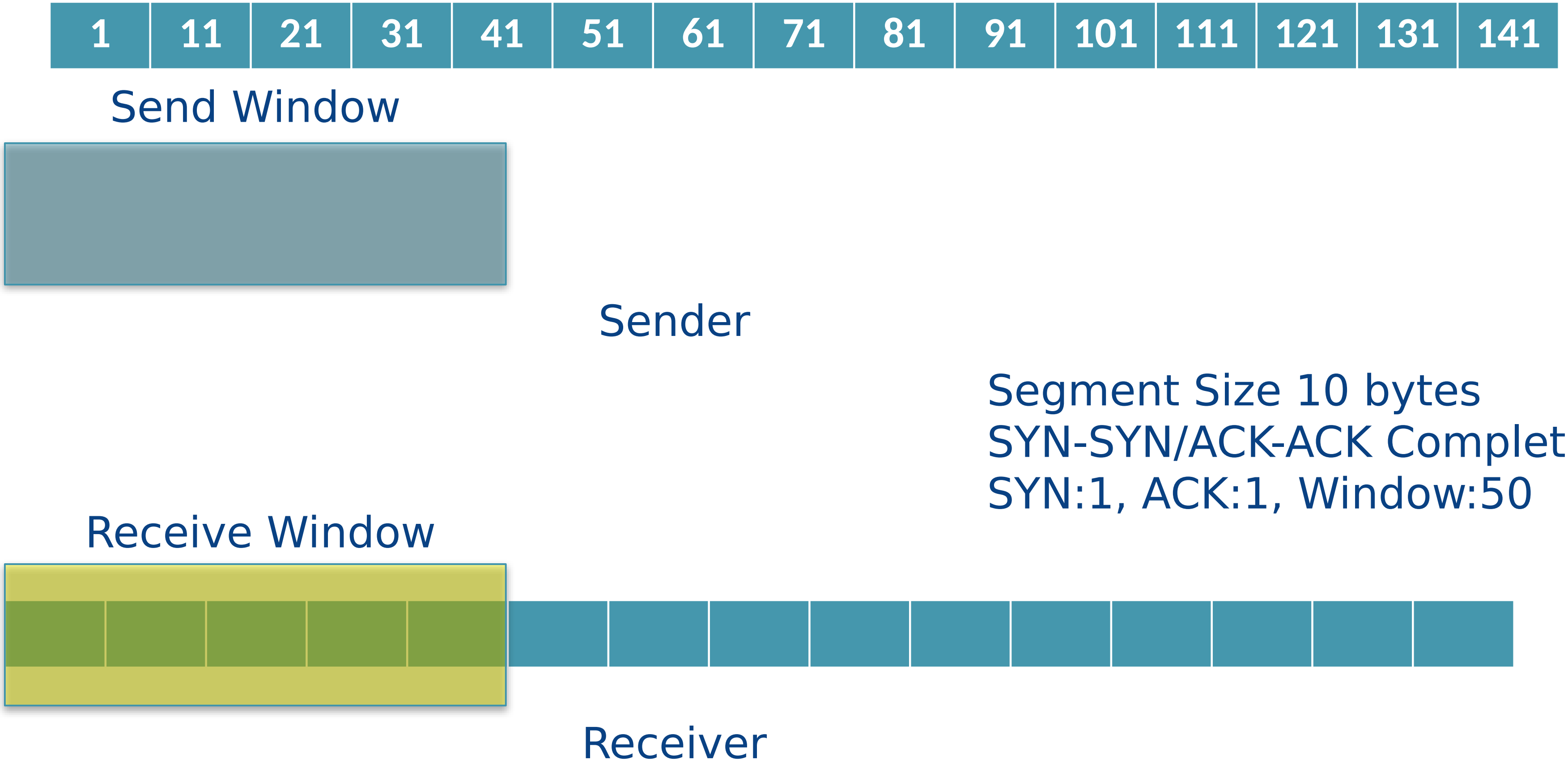
- When the window is 0 the sender should not send any data
  - Can send URGENT data
  - Can send “zero window probe”: 0 byte segment that causes the receiver to re-announce the next expected byte and window size (window probe) this is designed to prevent deadlock
- Senders may delay sending data, e.g., instead of sending the 2kiB immediately, could wait for a further 2kiB to fill the 4kiB receive window

TCP Sliding Window

- Send window
  - What data the sender is able to send – unacknowledged segments and unsent data that will fit into the receive window
- Receive window
  - Amount of data the receiver is willing to receive – window size in ACK

- Other windows are maintained for congestion control

TCP Sliding Window



1	11	21	31	41	51	61	71	81	91	101	111	121	131	141

Send Window

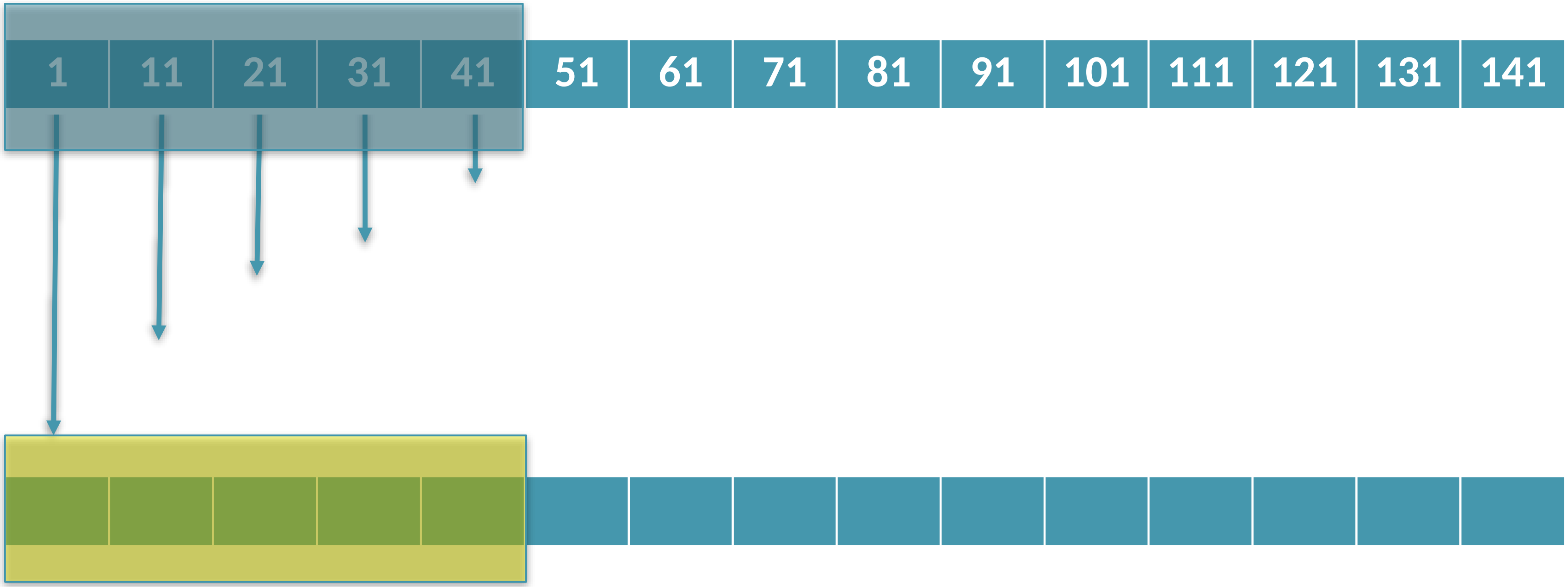
Sender

Segment Size 10 bytes
SYN-SYN/ACK-ACK Completed
SYN:1, ACK:1, Window:50

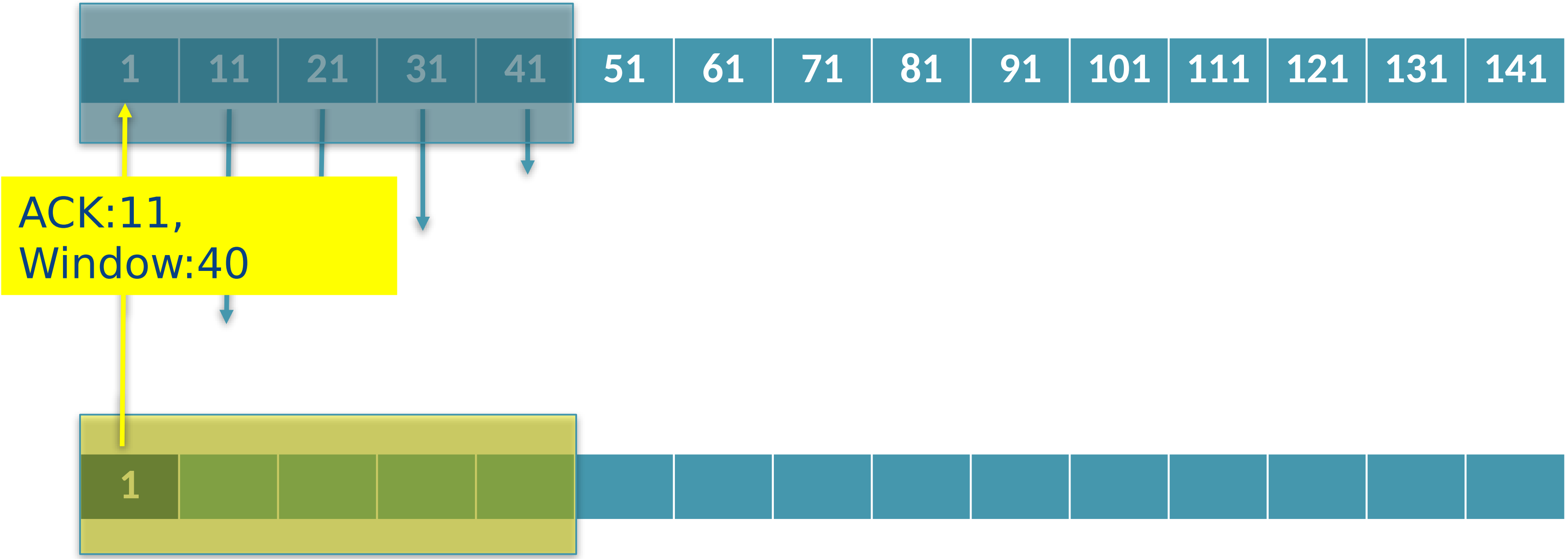
Receive Window

Receiver

TCP Sliding Window

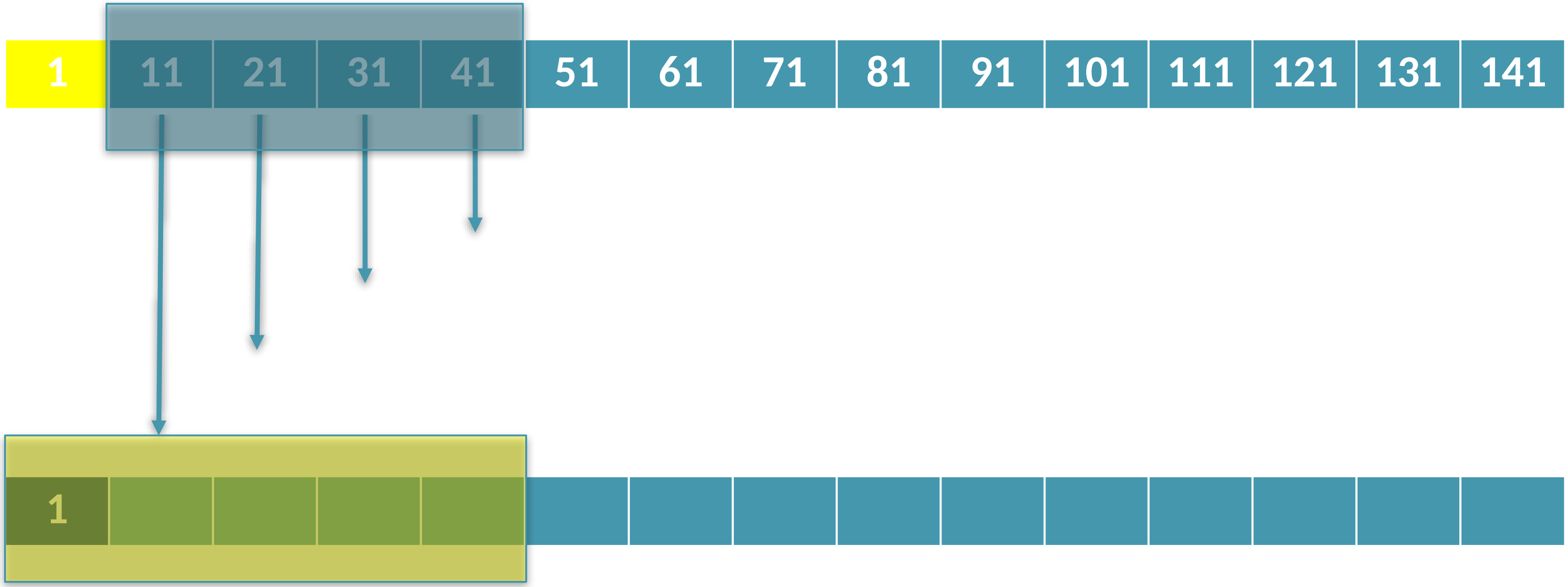


TCP Sliding Window

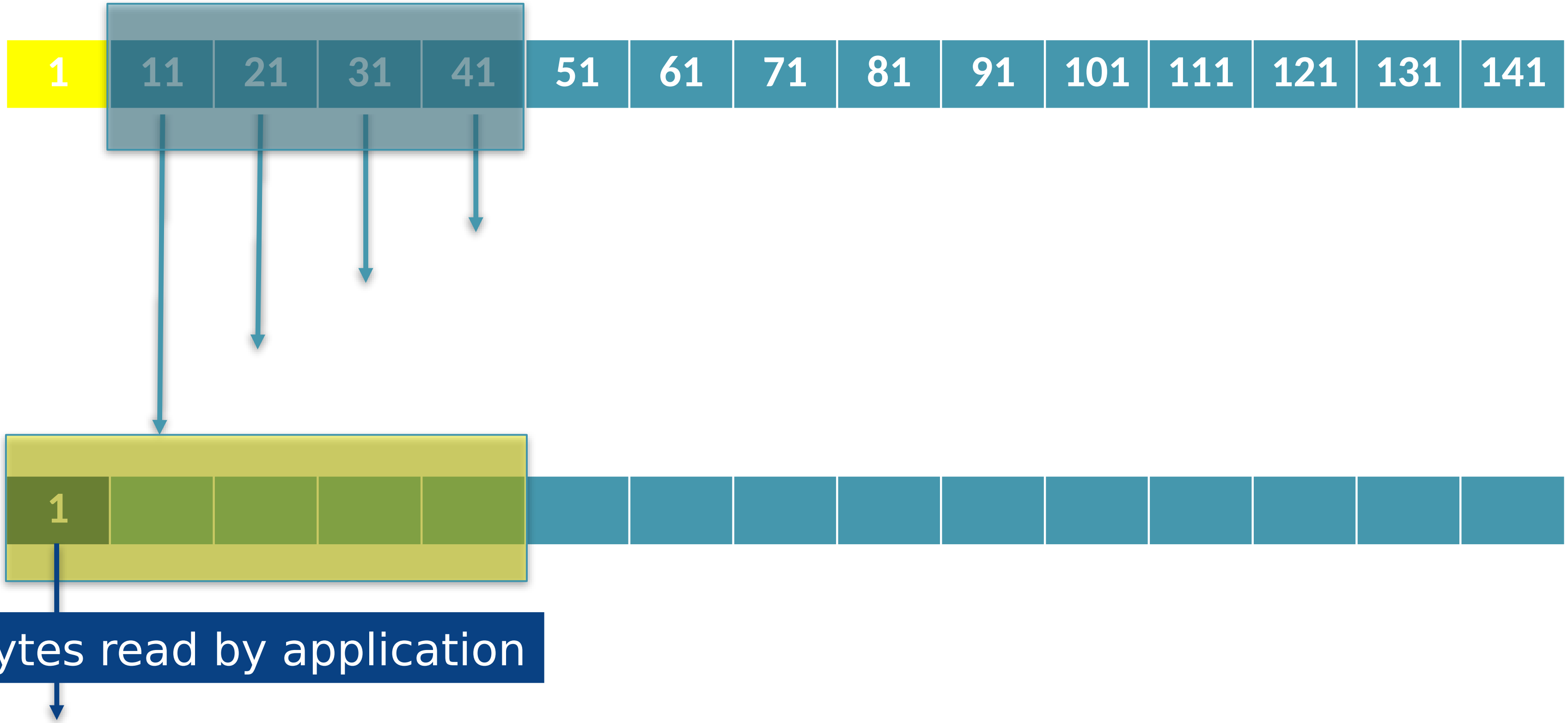


ACK:11,
Window:40

TCP Sliding Window

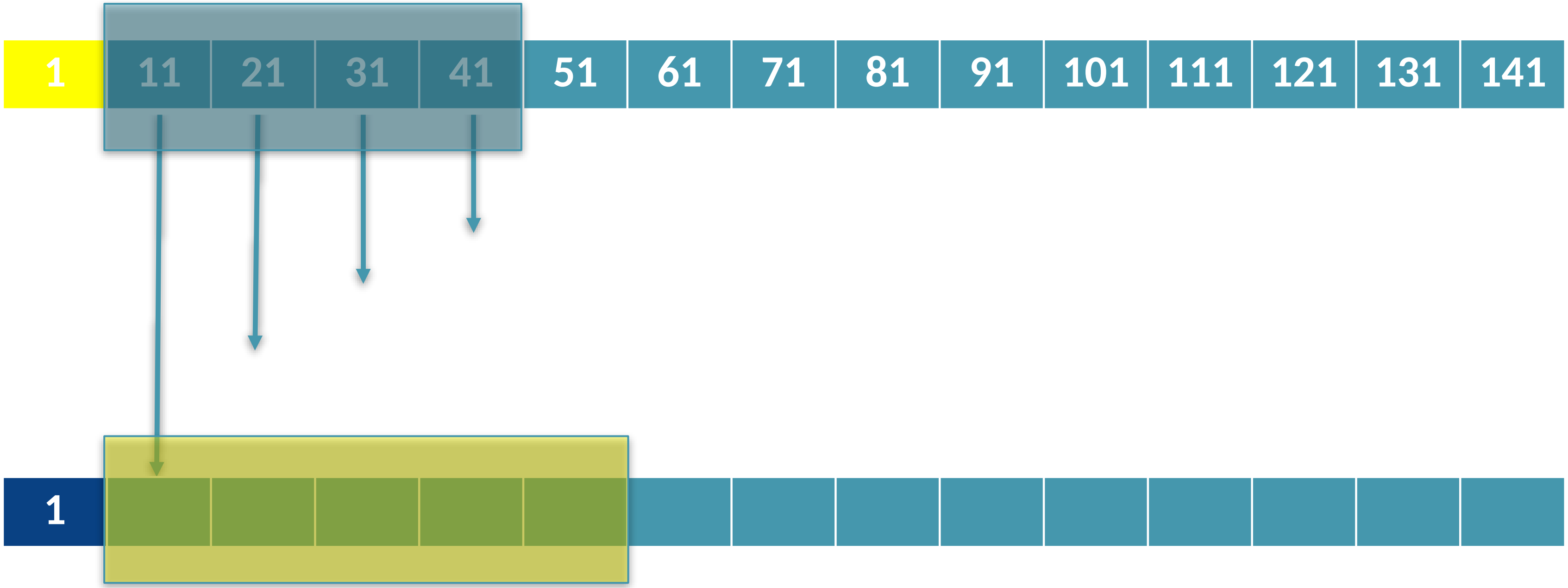


TCP Sliding Window

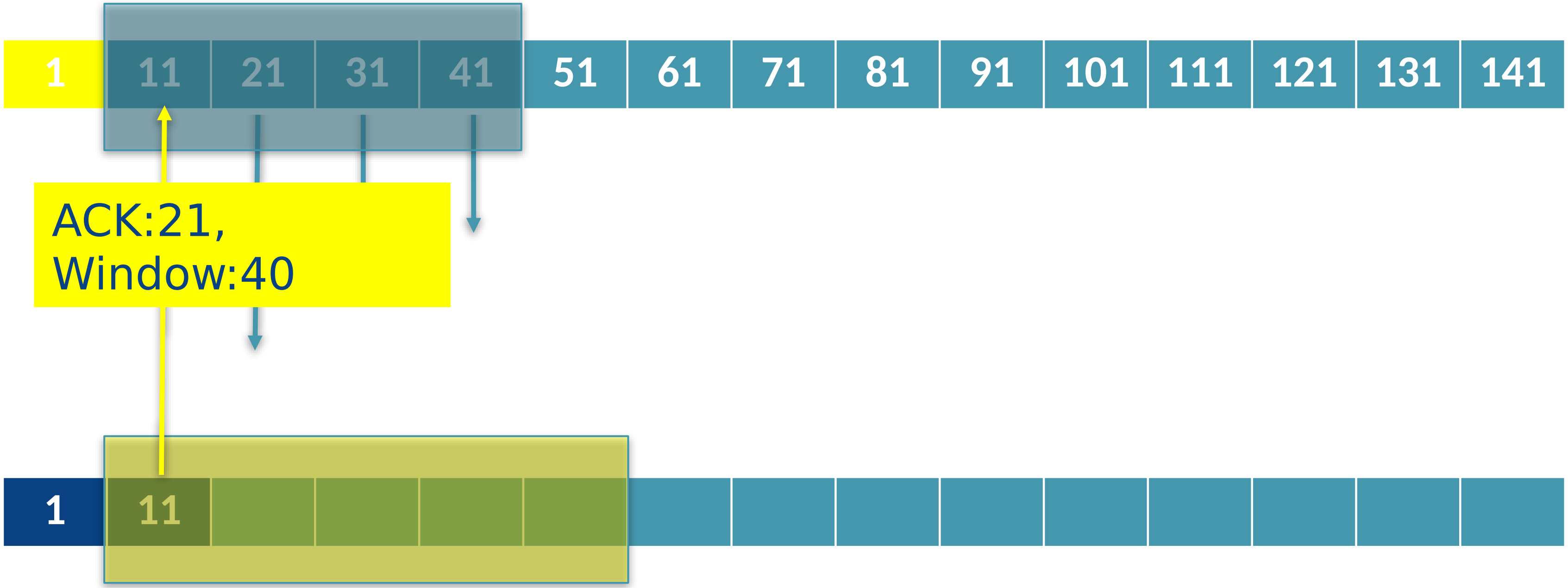


0 bytes read by application

TCP Sliding Window

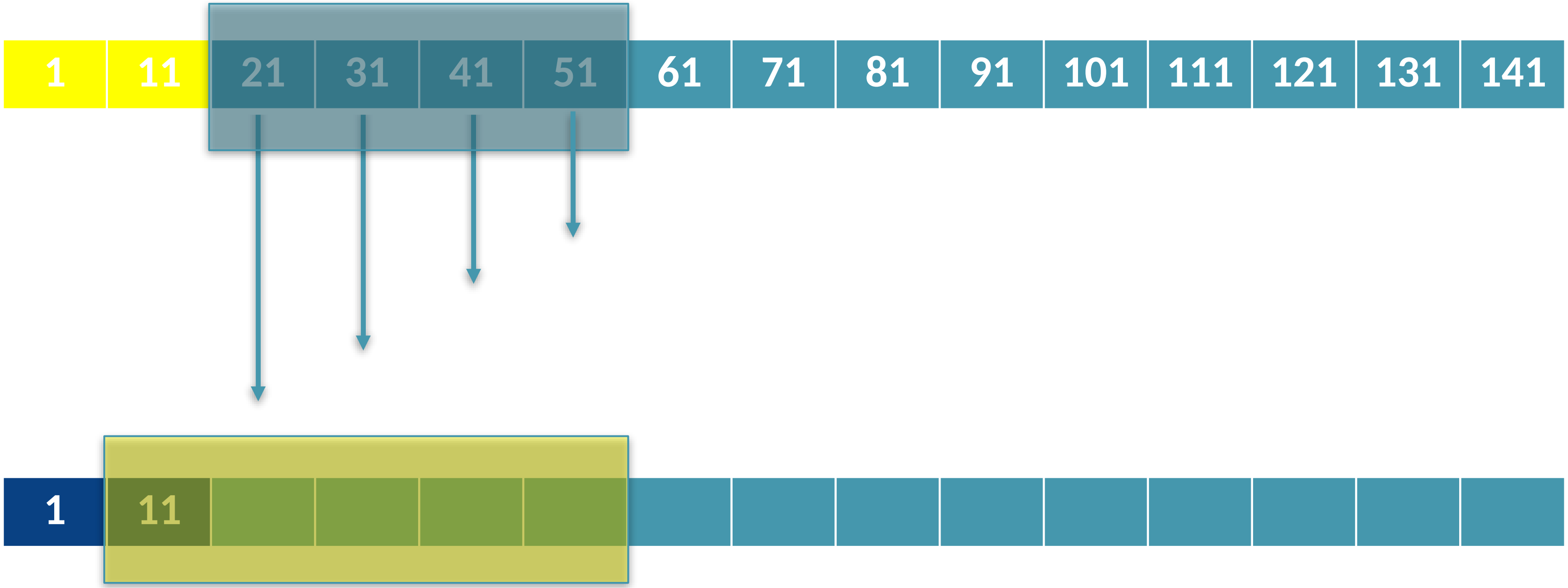


TCP Sliding Window

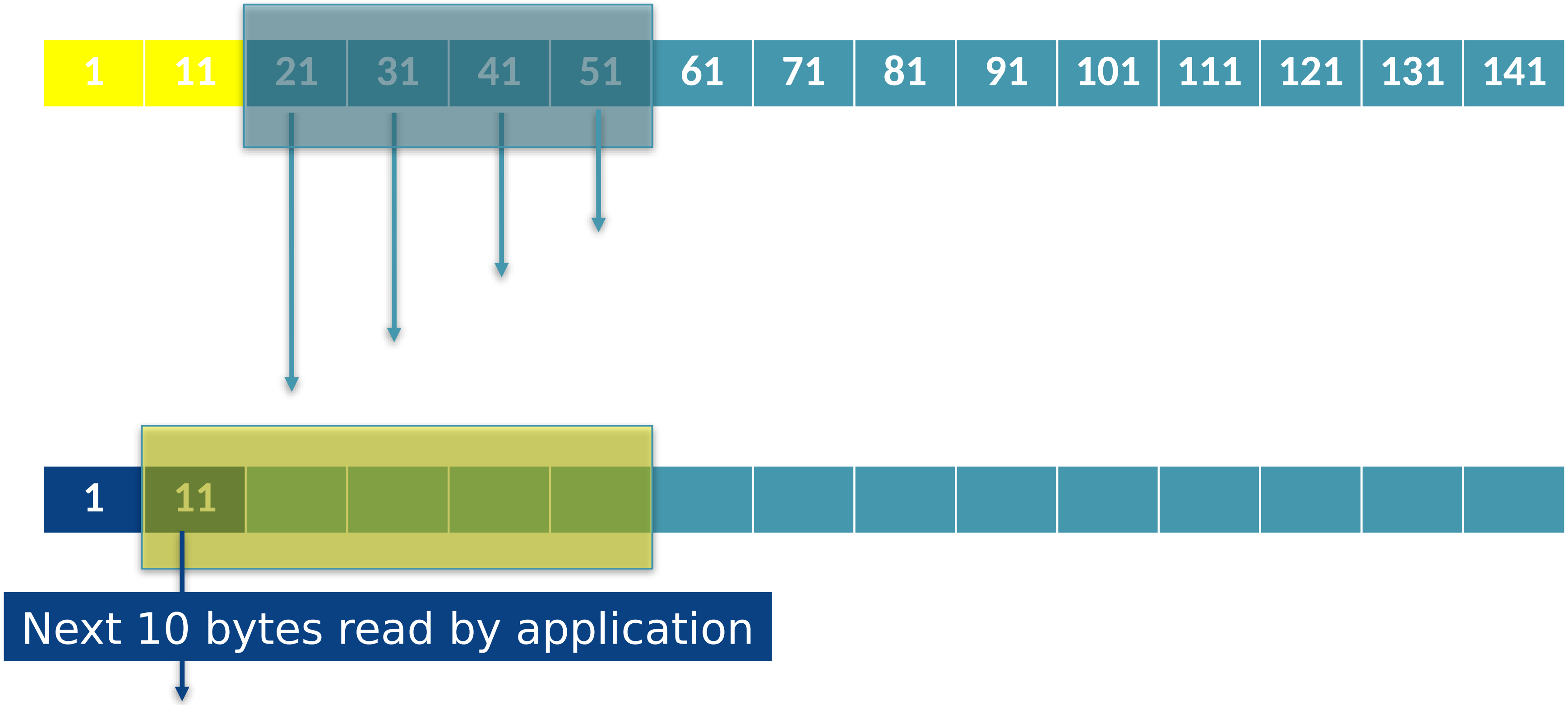


ACK:21,
Window:40

TCP Sliding Window

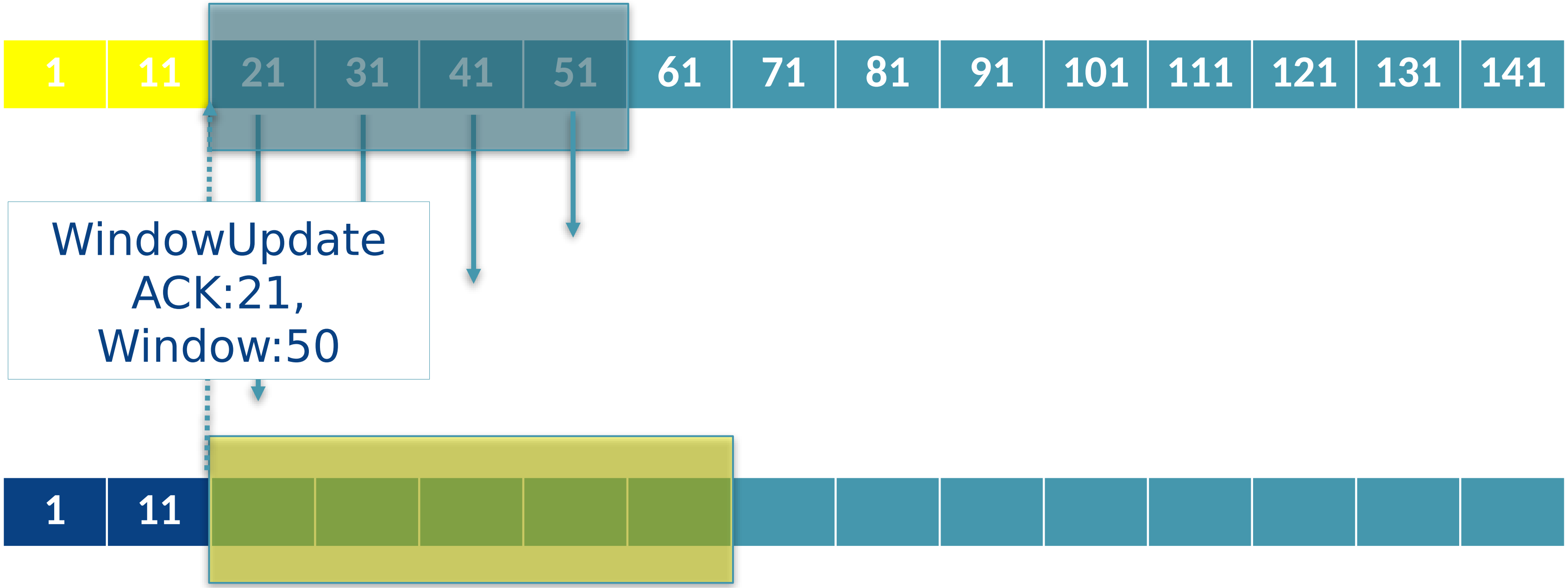


TCP Sliding Window



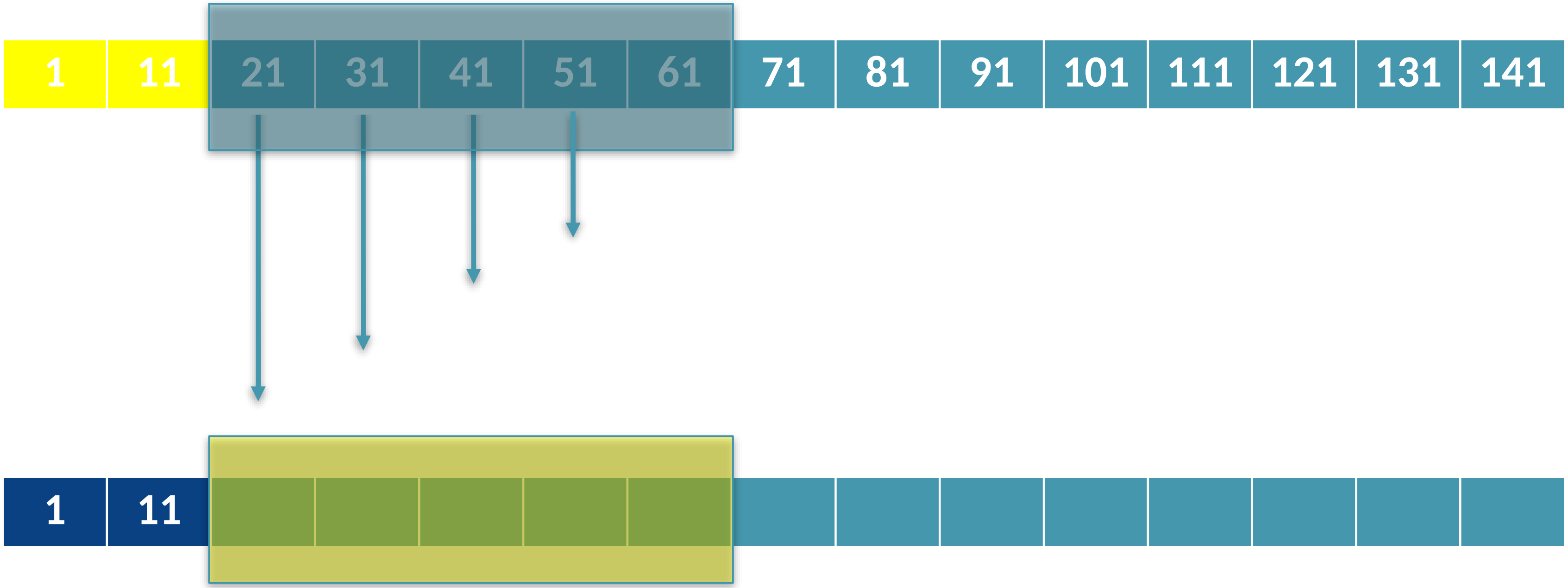
Next 10 bytes read by application

TCP Sliding Window – Window Update

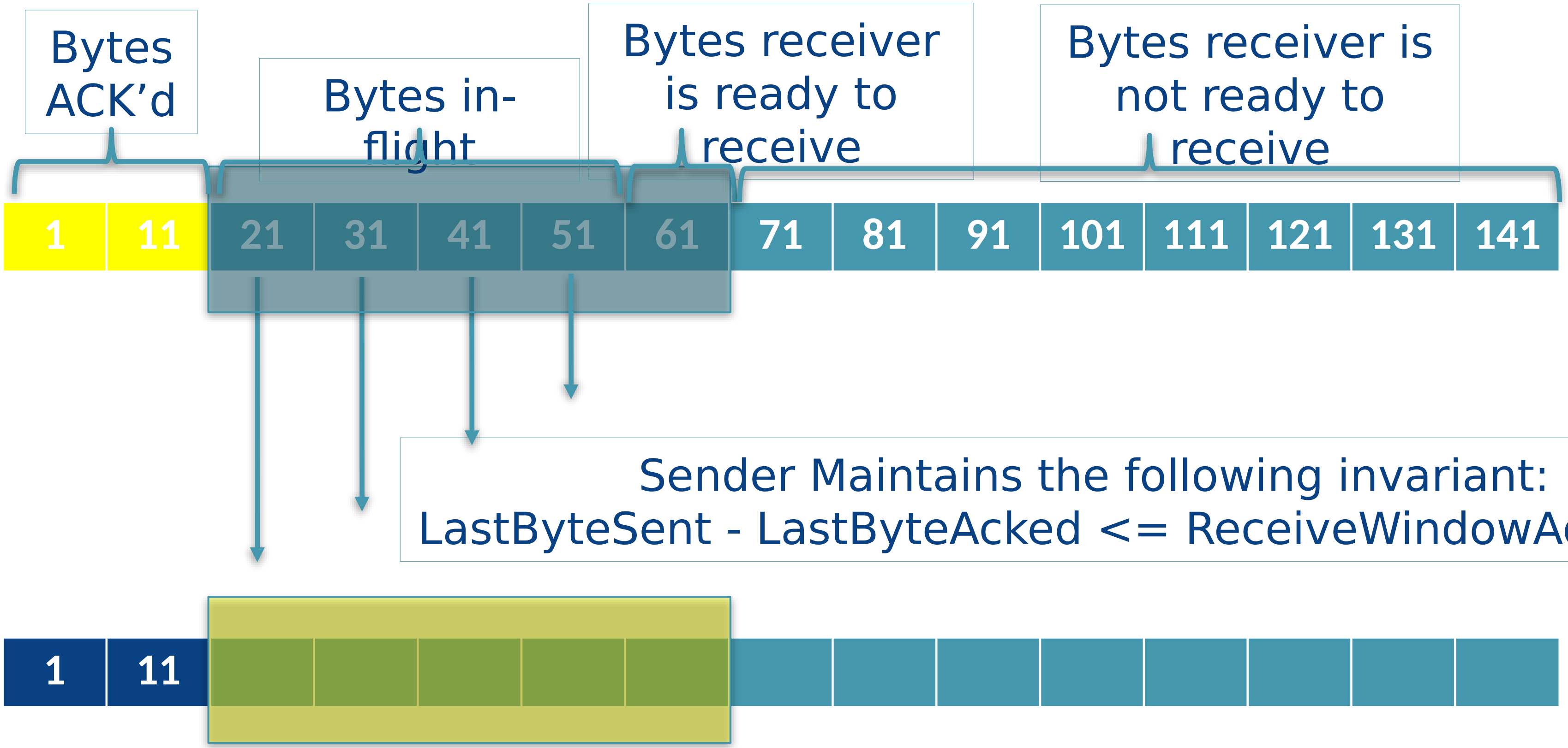


WindowUpdate ACK:21, Window:50

TCP Sliding Window



TCP Sliding Window


Bytes ACK'd
Bytes in-flight
Bytes receiver is ready to receive
Bytes receiver is not ready to receive
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
1
11

Sender Maintains the following invariant:
LastByteSent - LastByteAcked <= ReceiveWindowAdverti

Acknowledgement

- The slides are based on slides prepared by Chris Culnane based on material developed previously by: Michael Kirley, Zoltan Somogyi, Rao Kotagiri, James Bailey and Chris Leckie.
- Some of the images included in the notes were supplied as part of the teaching resources accompanying the text books listed in lecture 1.
  - (And also) Computer Networks, 6th Edition, Tanenbaum A., Wetherall. D. https://ebookcentral.proquest.com/lib/unimelb/detail.action?docID=6481879
- Textbook Reference: 3.5.4, 3.5.5, 3.4, bits of 3.2. The text doesn't cover sockets in C, but many books do; Google's first pick is: TCP/IP Sockets in C, by Donahoo. M, Calvert. K