Lecture 6.

Internet Transport Layer:

introduction to the

Transport Control Protocol (TCP)

RFC 793
(estensioni RFC 1122,1323,2018,2581,working group tsvwg)

G.Bianchi, G.Neglia, V.Mancuso

Università degli Studi di Palermo

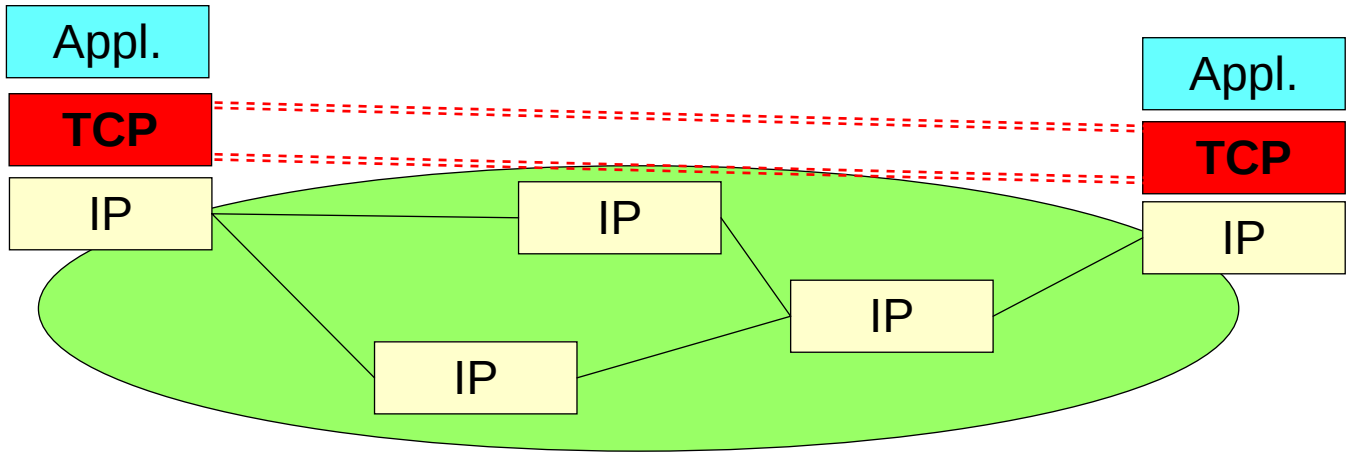
A MUCH more complex transport
for three main reasons

- **Connection oriented**
  - ⇨ implements mechanisms to setup and tear down a full duplex connection between end points
- **Reliable**
  - ⇨ implements mechanisms to guarantee error free and ordered delivery of information
- **Flow & Congestion controlled**
  - ⇨ implements mechanisms to control traffic

TCP services

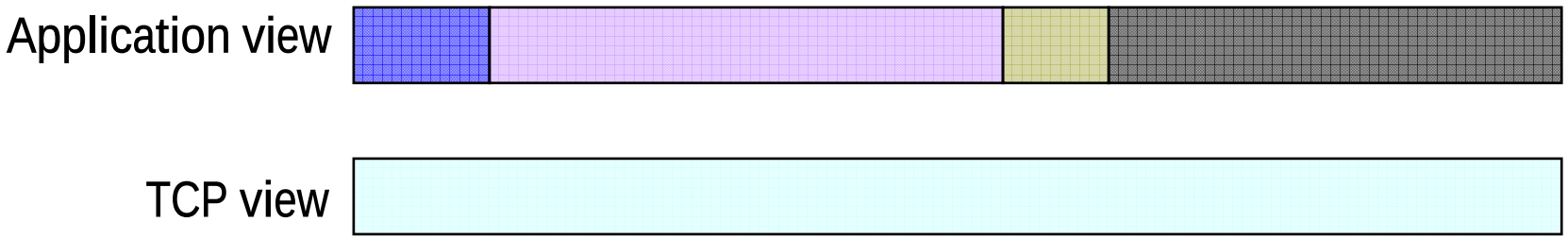
- **connection oriented**
  - ⇨ TCP connections
- ***reliable* transfer service**
  - ⇨ all bytes sent are received

- **TCP functions**
  - → application addressing (ports)
  - → error recovery (acks and retransmission)
  - → reordering (sequence numbers)
  - → flow control
  - → congestion control



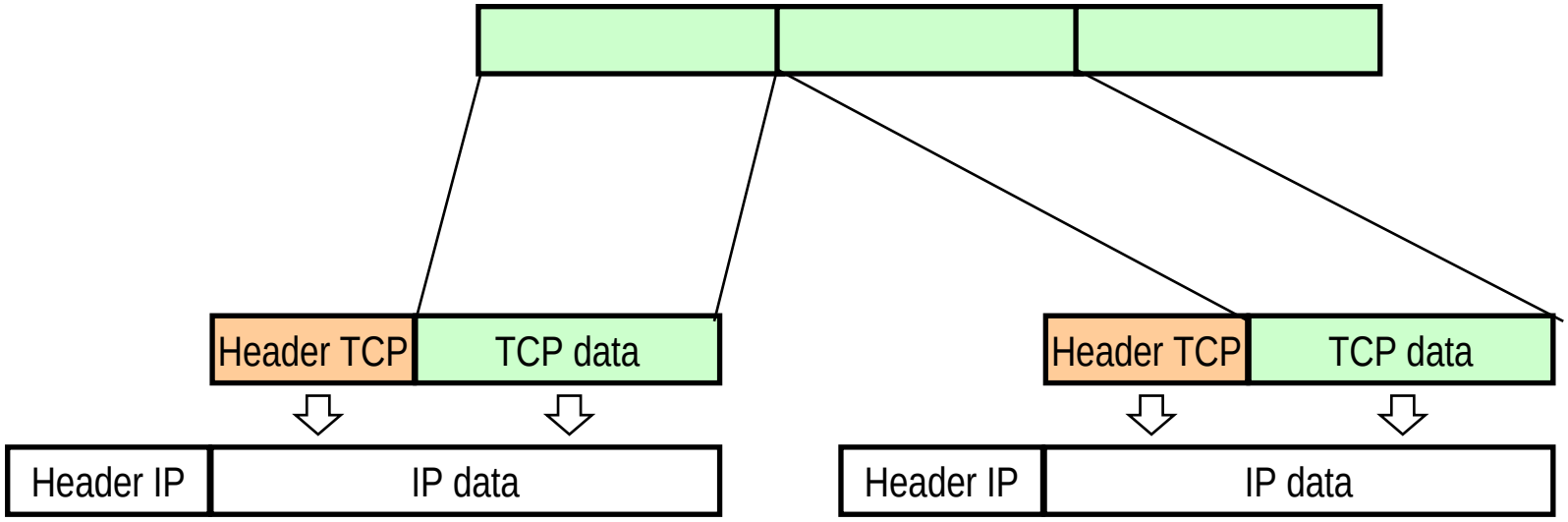
Byte stream service

- **TCP exchange data between applications as a stream of bytes**
- **It does not introduce any data delimiter (an application duty)**
  - ⇨ source application may enter 10 bytes followed by 1 and 40 (grouped with some semantics)
  - ⇨ data is buffered at source, and transmitted
  - ⇨ at receiver, may be read in the sequence 25 bytes, 22 bytes and 4 bytes...



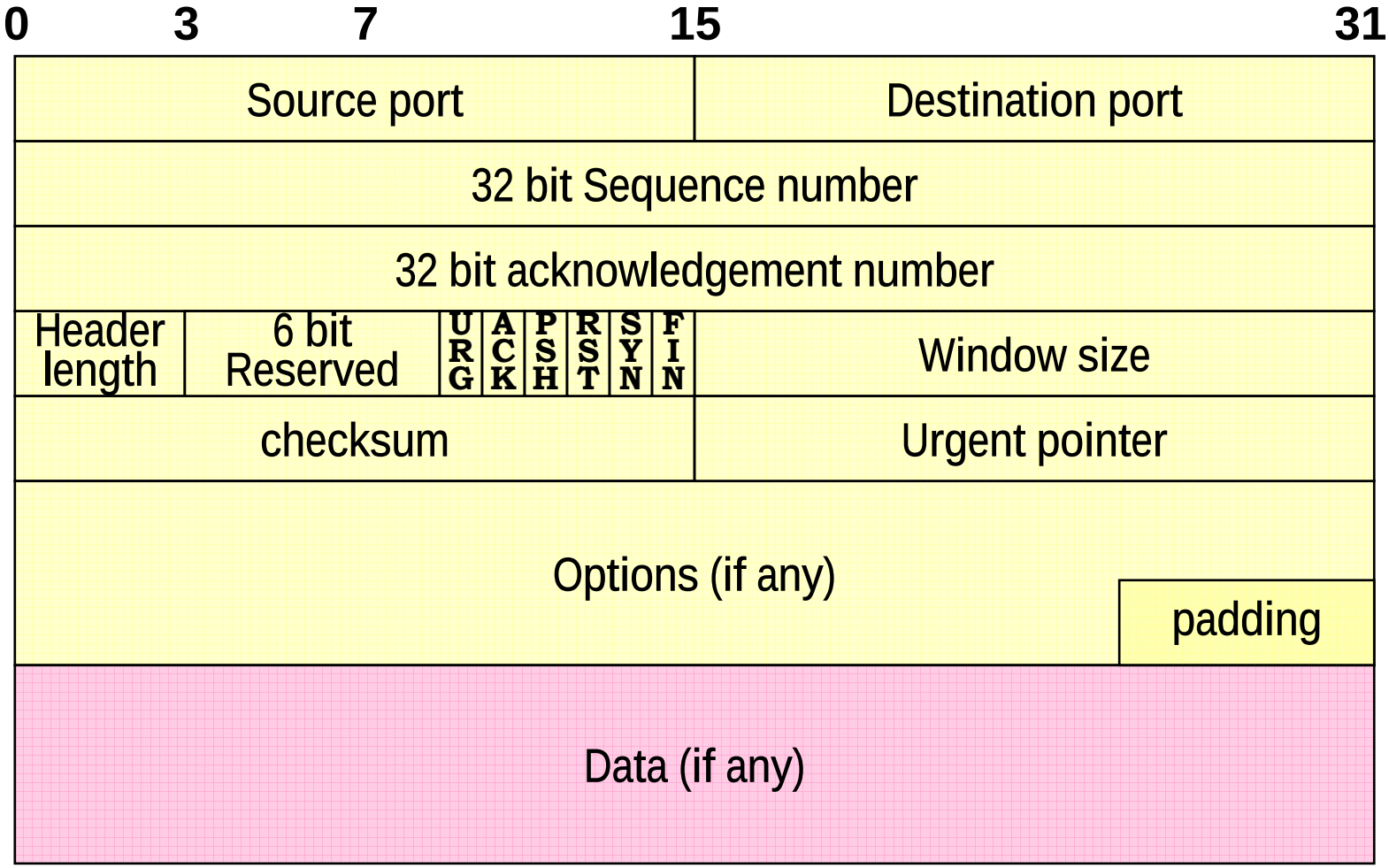
TCP segments

- ➔ Application data broken into segments for transmission
- ➔ segmentation totally up to TCP, according to what TCP considers being the best strategy
- ➔ each segment placed into an IP packet
- ➔ very different from UDP!!



TCP segment format

20 bytes header (minimum)



<table>
<tr><td colspan="5">Source port</td><td colspan="3">Destination port</td></tr>
<tr><td colspan="8">32 bit Sequence number</td></tr>
<tr><td colspan="8">32 bit acknowledgement number</td></tr>
<tr><td>Header length</td><td colspan="2">6 bit Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Window size</td></tr>
<tr><td colspan="5">checksum</td><td colspan="3">Urgent pointer</td></tr>
</table>

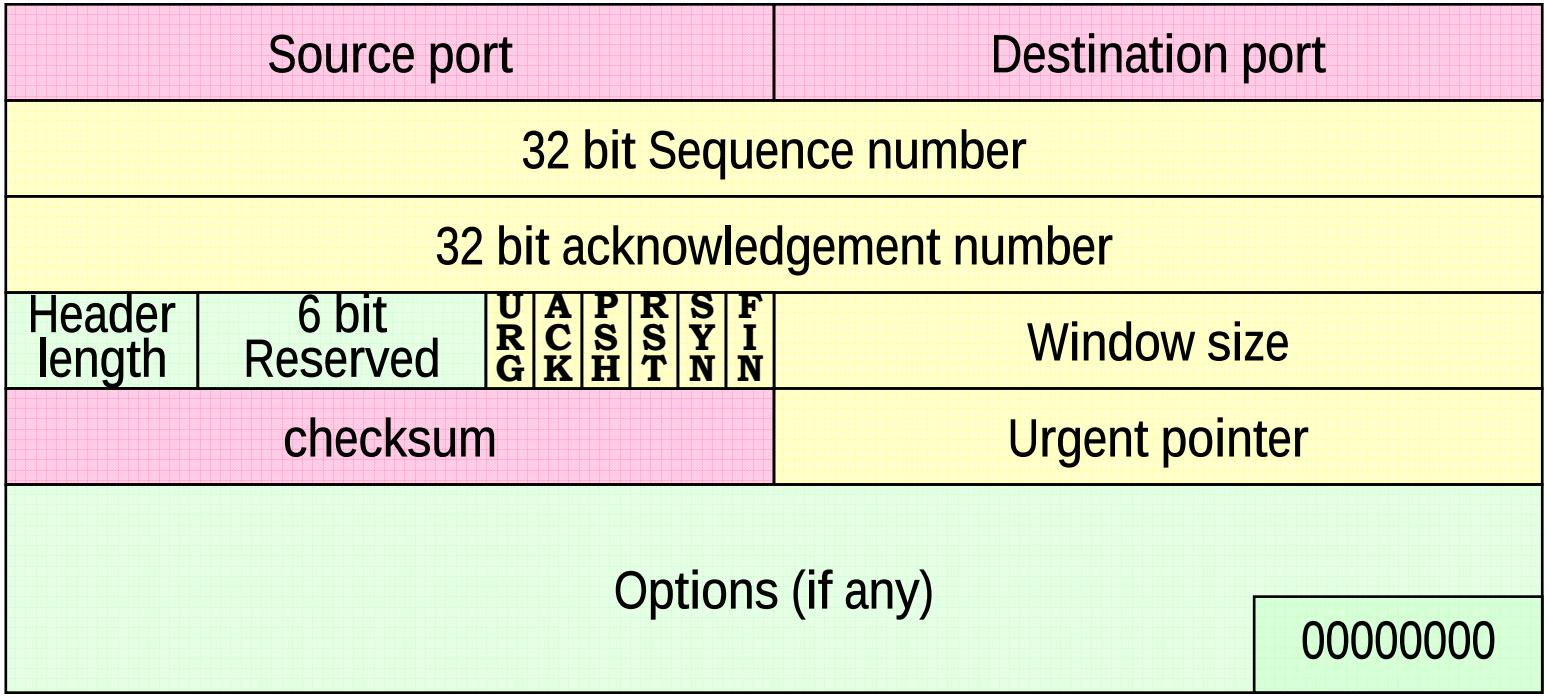
➔ **Source & destination port + source and destination IP addresses**

⇨ univocally determine TCP connection

➔ **checksum as in UDP**

⇨ same calculation including same pseudoheader

➔ **no explicit segment length specification**

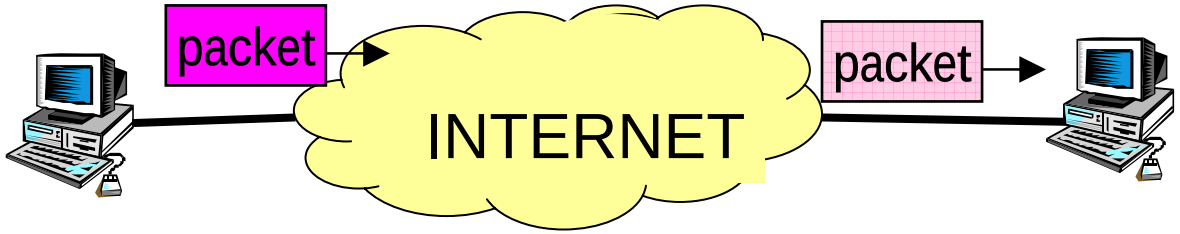


➔ **Header length: 4 bits**

- ⇨ specifies the header size (n*4byte words) for options
- ⇨ maximum header size: 60 (15*4)
- ⇨ option field size must be multiple of 32bits: zero padding when not.

➔ **Reserved: 000000 (still today!)**

Reliable data transfer: issues



PROBLEMS:

1) Packet received with errors
2) Packet not received at all

Same problem considered at DATA LINK LAYER
(although it is less likely that a whole packet is lost at data link)

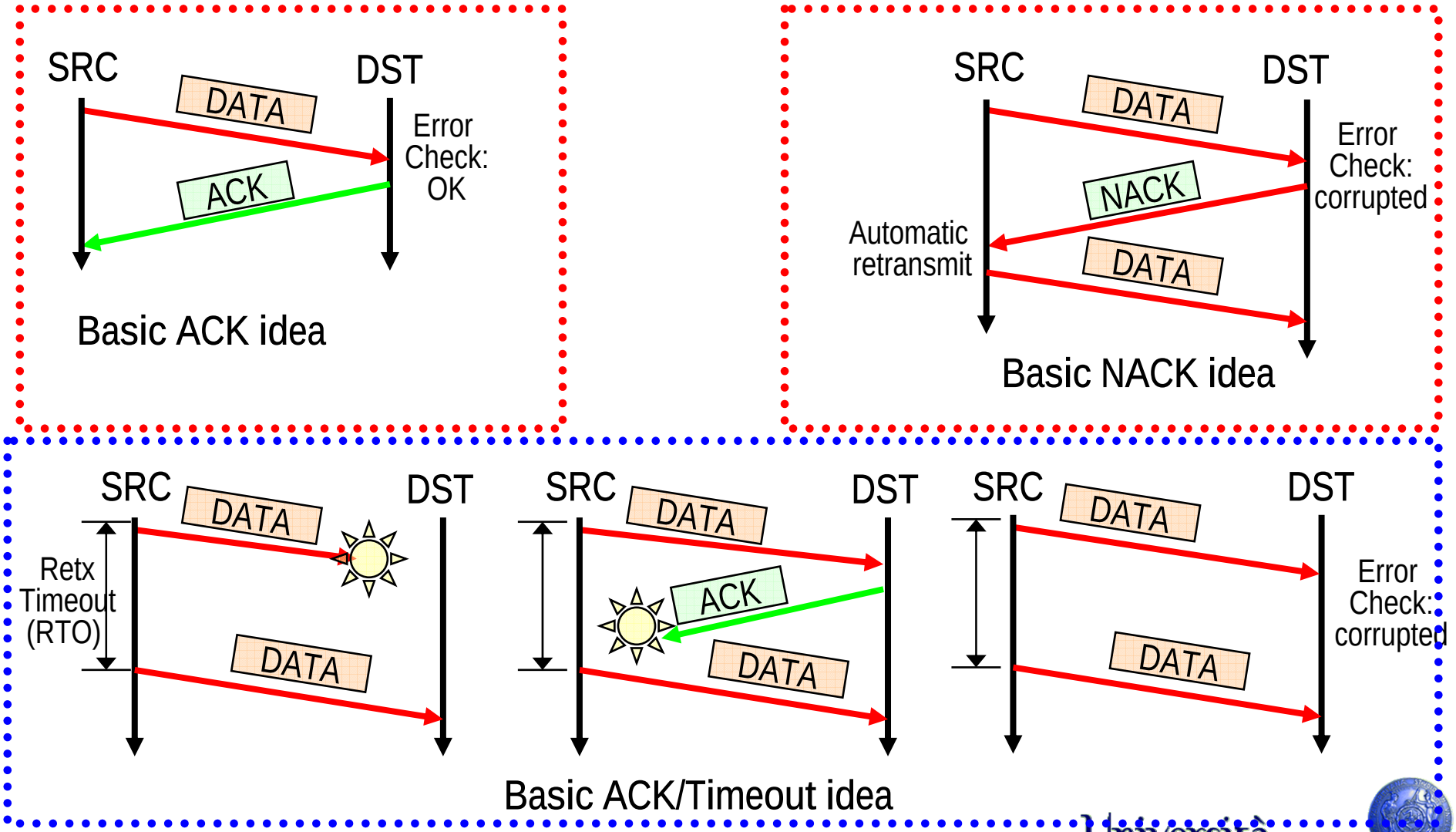
➔ mechanisms to guarantee correct reception:

- ⇨ Forward Error Correction (FEC) coding schemes
  - ➔ Powerful to correct bits affected by error, less effective in case of packet loss
  - ➔ Mostly used at link layer
- ⇨ Retransmission – issues:
  - ➔ ACK
  - ➔ NACK
  - ➔ TIMEOUT

Retransmission scenarios

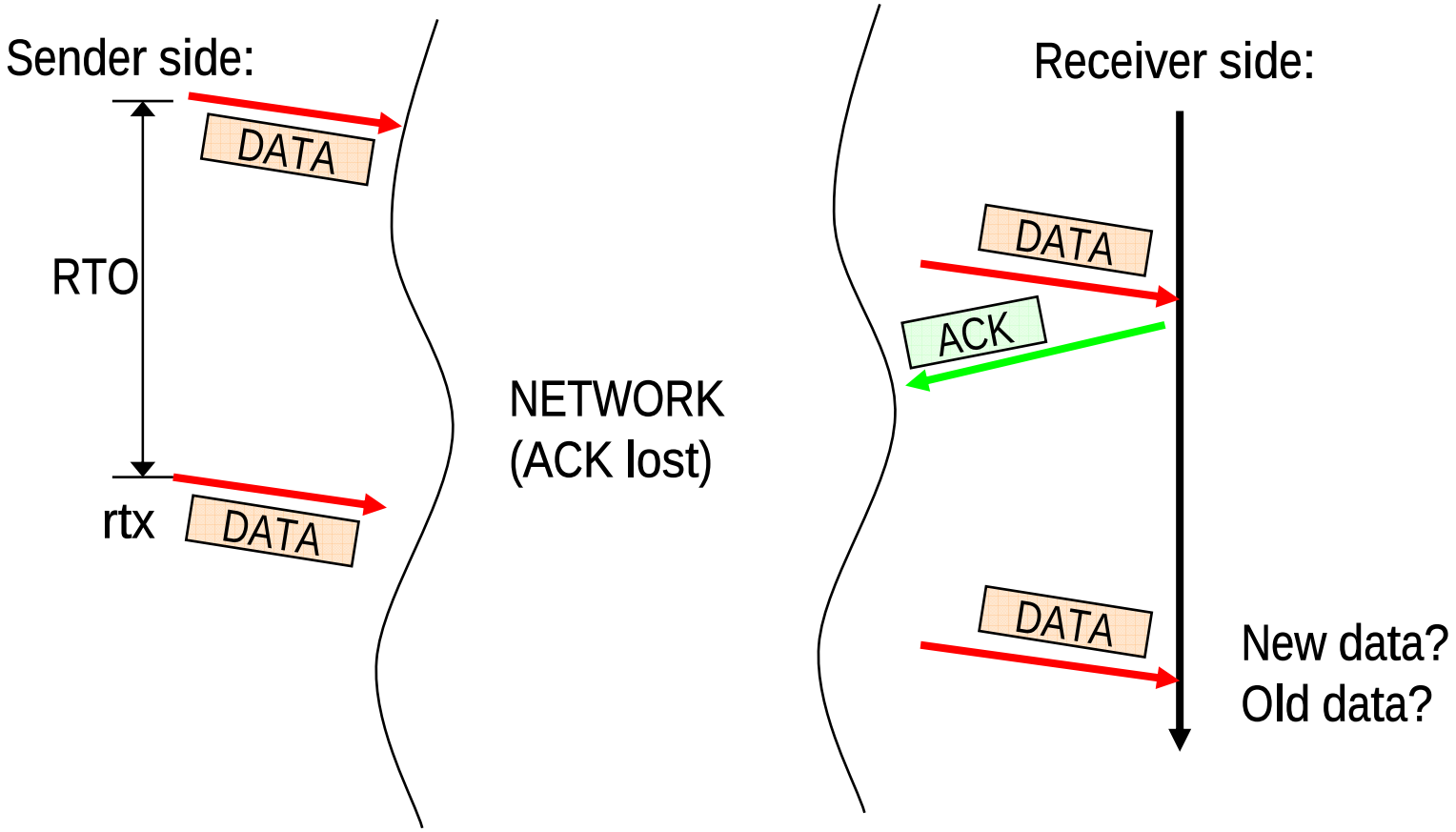
referred to as ARQ schemes (Automatic Repeat reQuest)

COMPONENTS: a) error checking at receiver; b) feedback to sender; c) retx



Why sequence numbers?

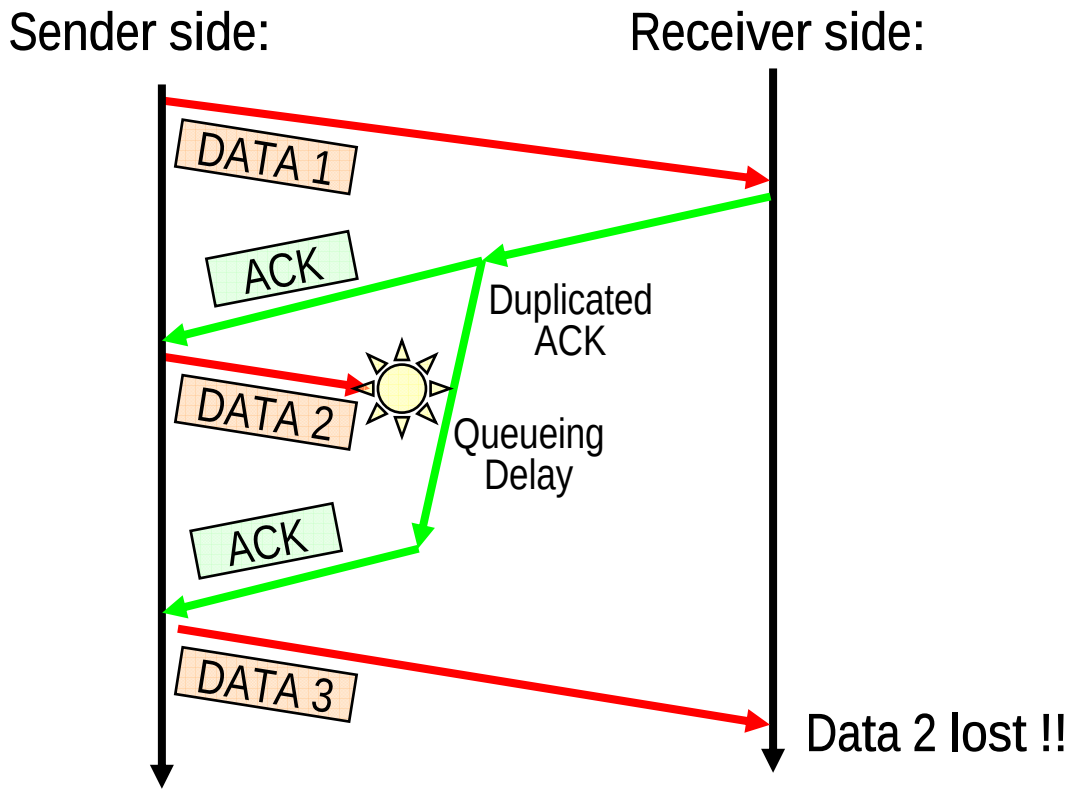
(on data)



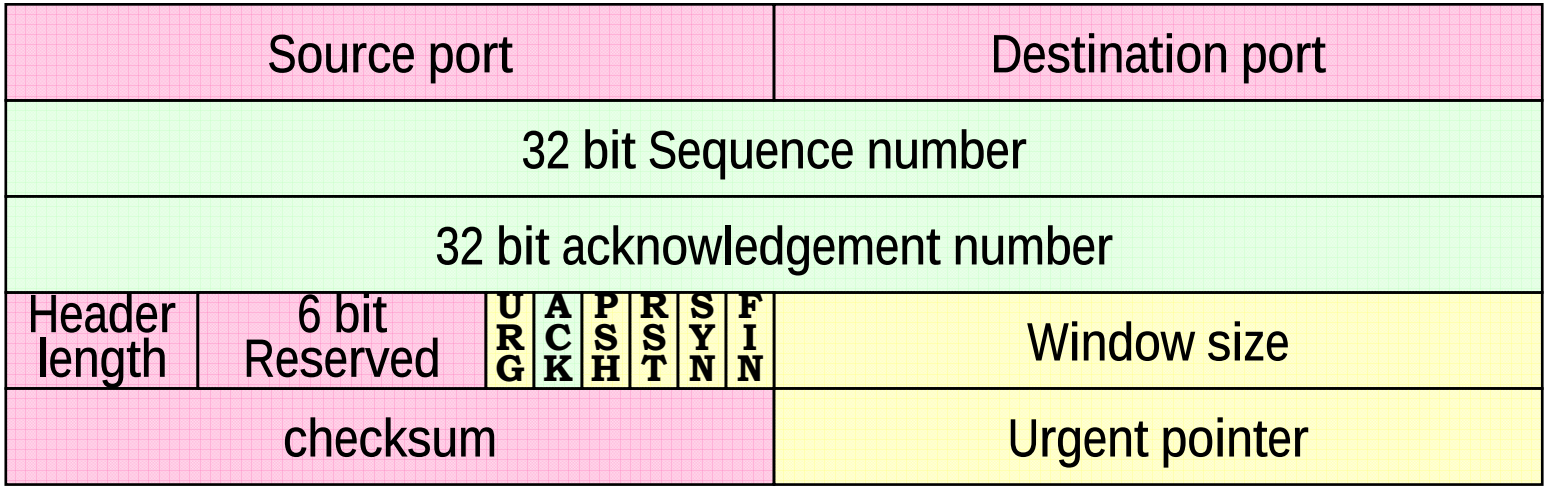
Need to univocally "label" all packets circulating in the network between two end points.
1 bit (0-1) enough for Stop-and-wait

Why sequence numbers?

(on ack)



With pathologically critical network (as the Internet!)
also need to univocally "label" all acks circulating
in the network between two end points.
1 bit (0-1) enough for Stop-and-wait



➔ **Sequence number:**

- ⇨ Sequence number of the *first* byte in the segment.
- ⇨ When reaches 2^{32}-1, next wraps back to 0

➔ **Acknowledgement number:**

- ⇨ valid only when ACK flag on
- ⇨ Contains the *next* byte sequence number that the host *expects* to receive (= last successfully received byte of data + 1)
- ⇨ grants successful reception for <u>all bytes</u> up to ack# - 1 (cumulative)

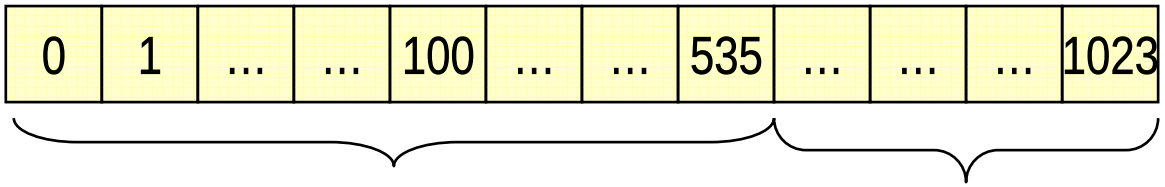
➔ **When seq/ack reach 2^{32}-1, next wrap back to 0**

TCP data transfer management

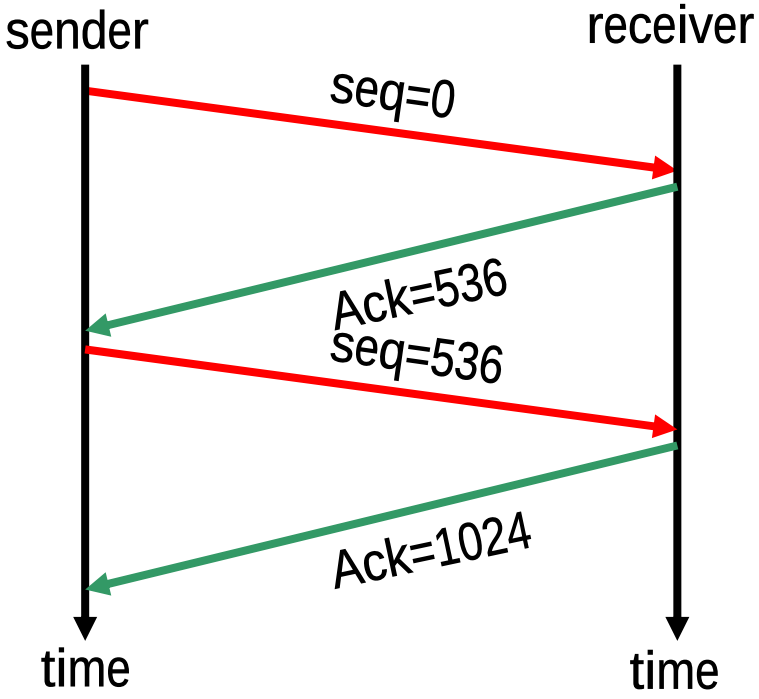
- **Full duplex connection**
  - data flows in both directions, independently
  - To the application program these appear as two unrelated data streams
  - Impossible to build multicast connection
- **each end point maintains a sequence number**
  - Independent sequence numbers at both ends
  - Measured in bytes
- **acks often carried on top of reverse flow data segments (piggybacking)**
  - But ack packets alone are possible

Byte-oriented

Example: 1 kbyte message – 1024 bytes



Example: segment size = 536 bytes → 2 segments: 0-535; 536-1023

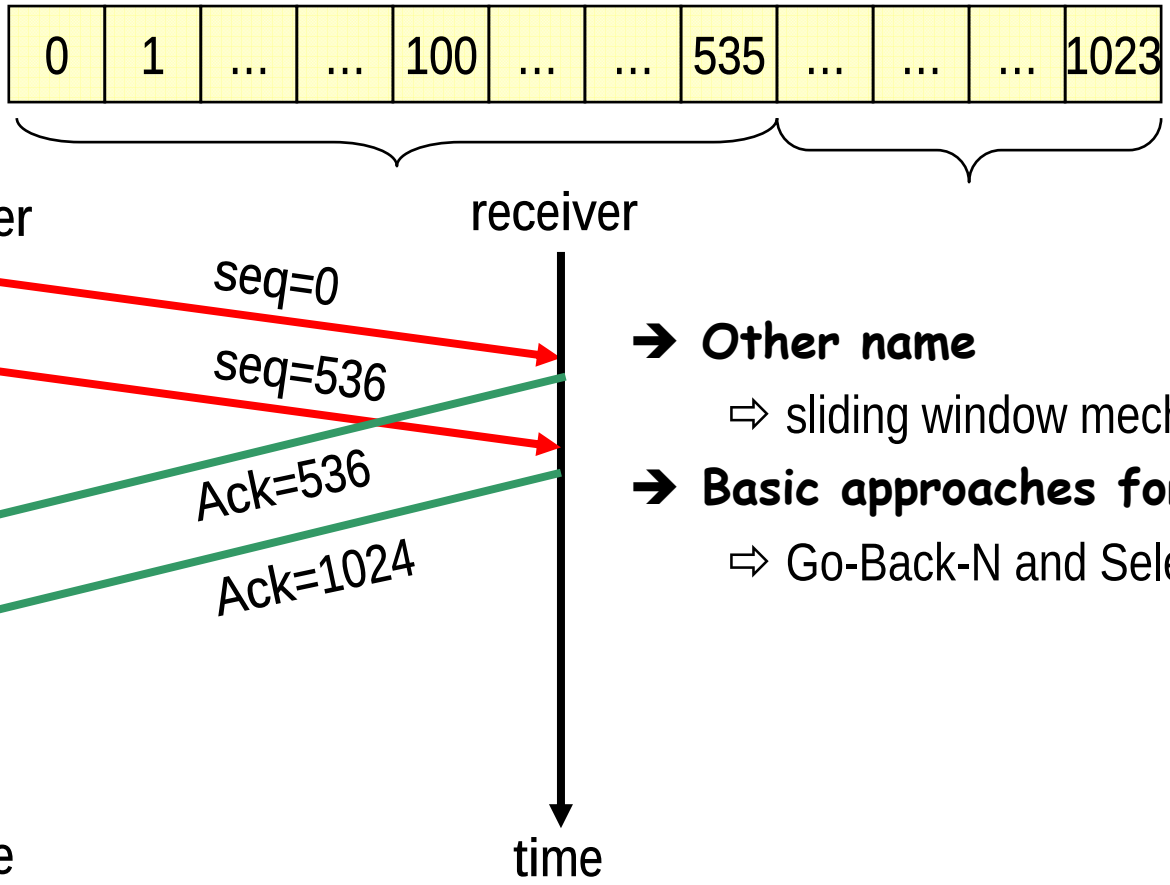


➔**No explicit segment size indication**

- ⇨ Seq = first byte number
- ⇨ Returning Ack = last byte number + 1
- ⇨ Segment size = Ack-seq#

Pipelining

Example: 1024 bytes msg; seg_size = 536 bytes → 2 segments: 0-535; 536-1023



➔ **Other name**

⇨ sliding window mechanisms

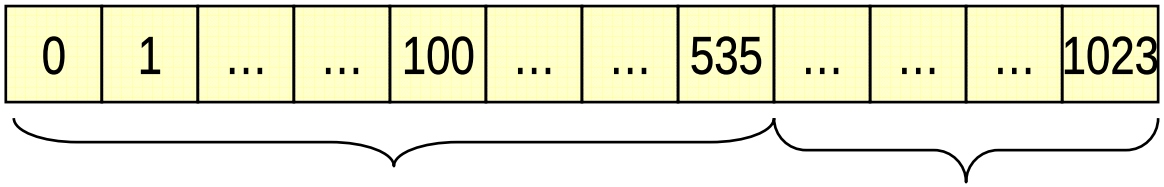
➔ **Basic approaches for error recovery**

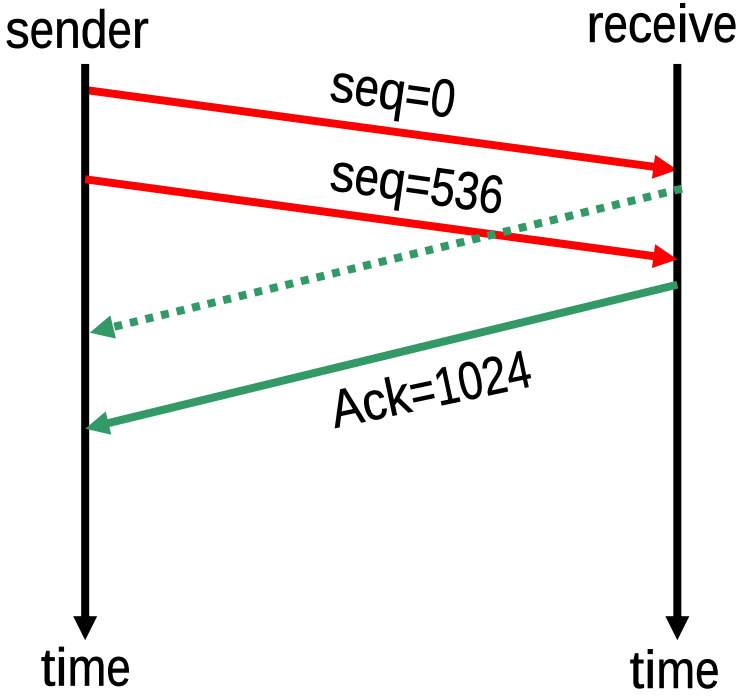
⇨ Go-Back-N and Selective Repeat

Why pipelining? Dramatic improvement in efficiency!

Cumulative ack

Example: 1024 bytes msg; seg_size = 536 bytes → 2 segments: 0-535; 536-1023





➔ **Cumulative ack**

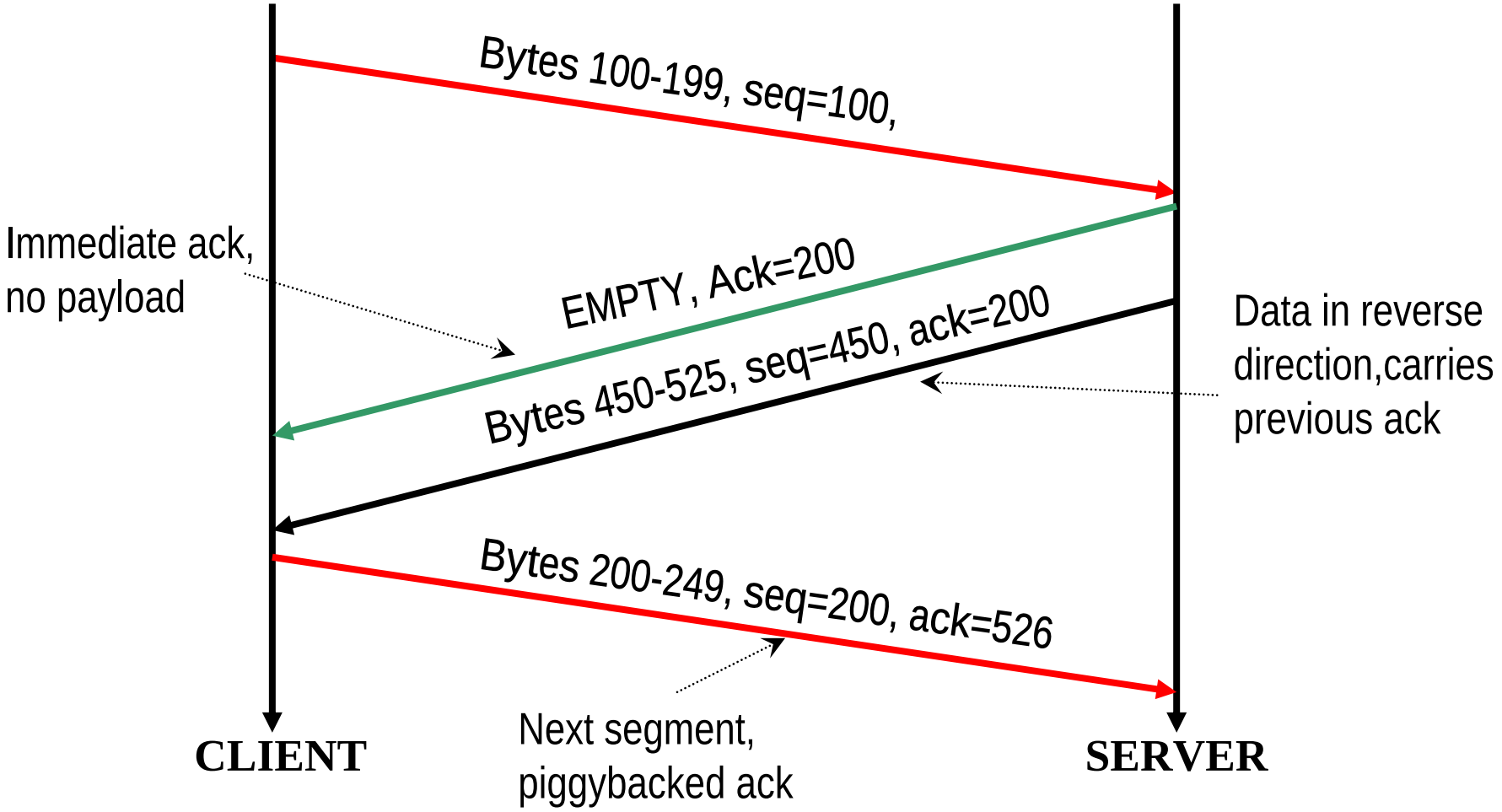
- ⇨ ACK = **all** previous bytes correctly received!
- ⇨ E.g. ACK=1024: all bytes 0-1023 received

➔ **Other names of pipelining:**

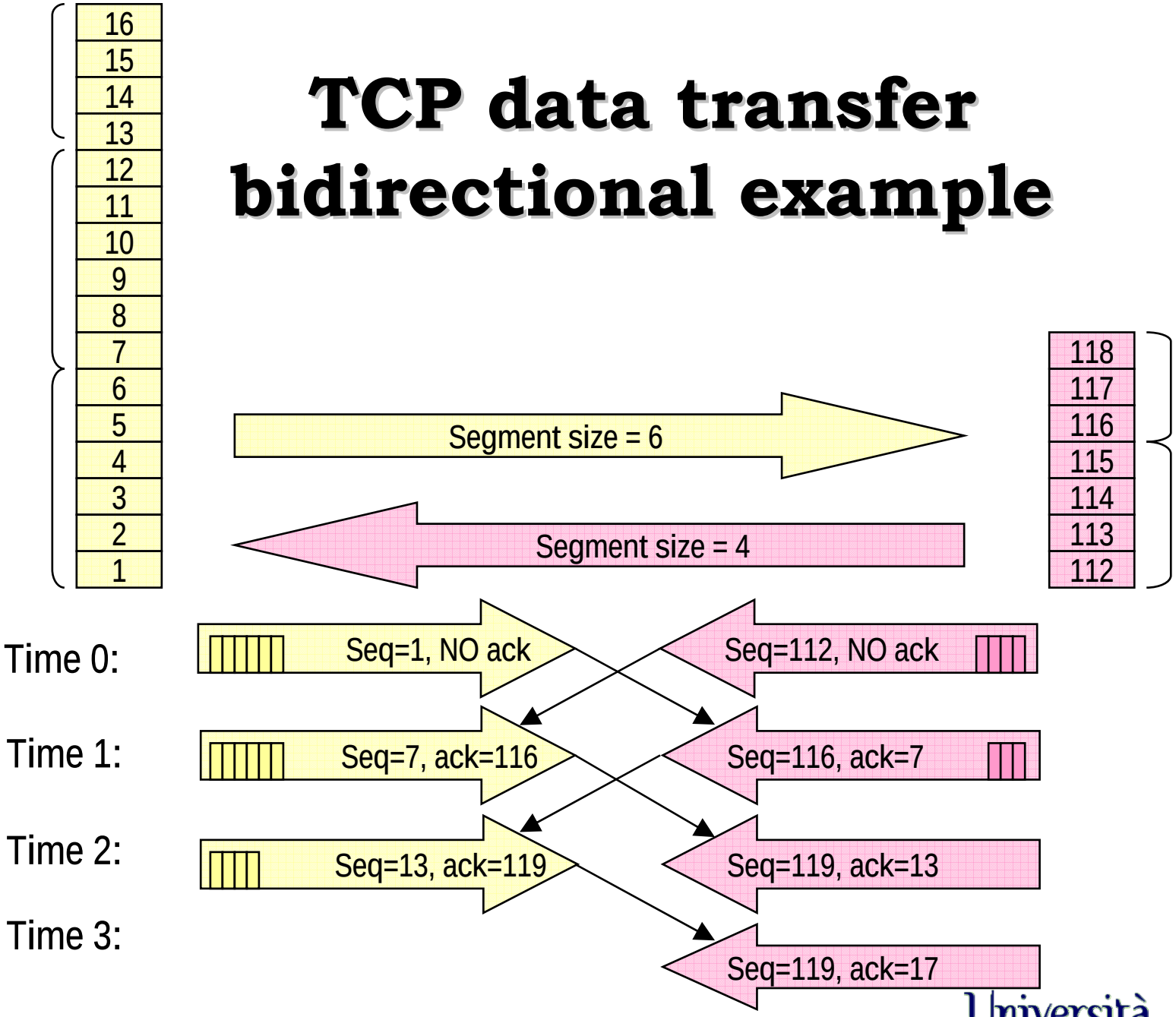
- ⇨ Go-Back-N ARQ mechanisms
- ⇨ Sliding window mechanisms

Why pipelining? Dramatic improvement in efficiency!

Multiple acks; Piggybacking

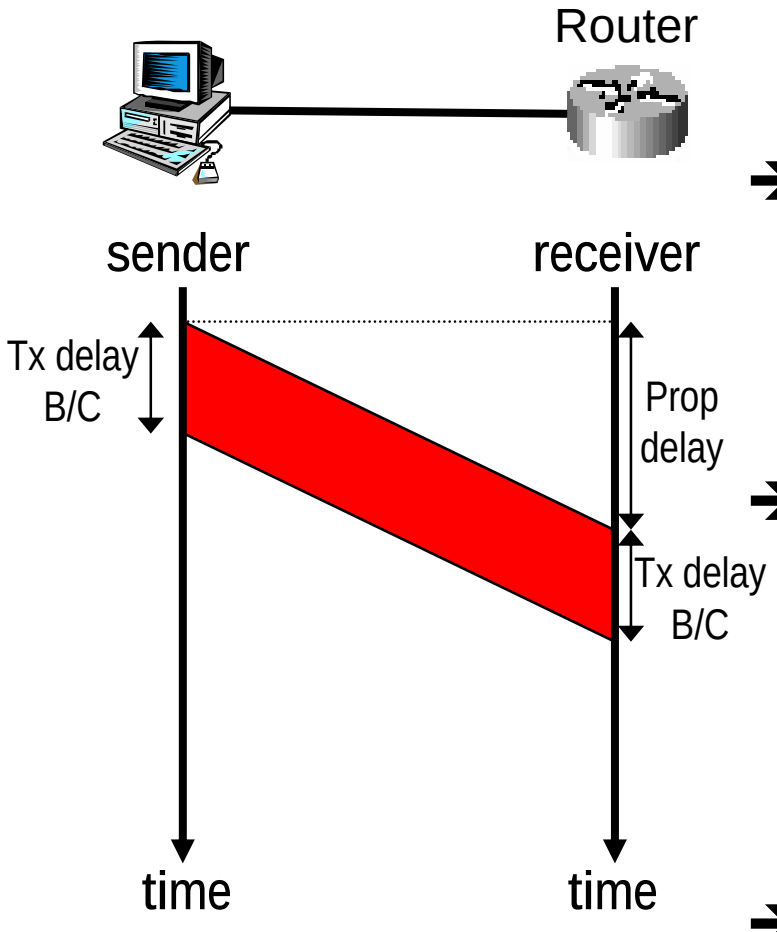


TCP data transfer bidirectional example



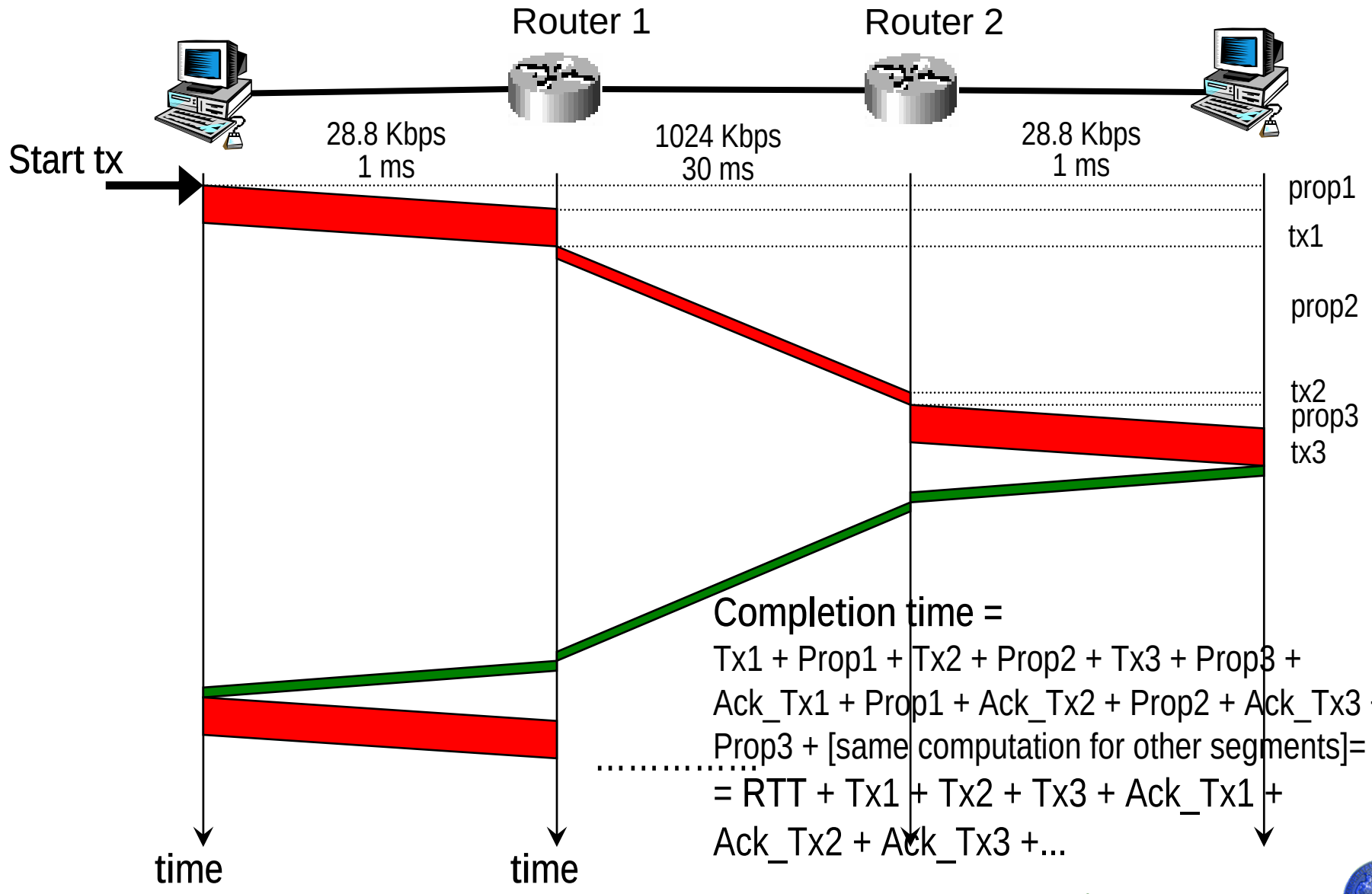
Performance issues with/without pipelining

Link delay computation



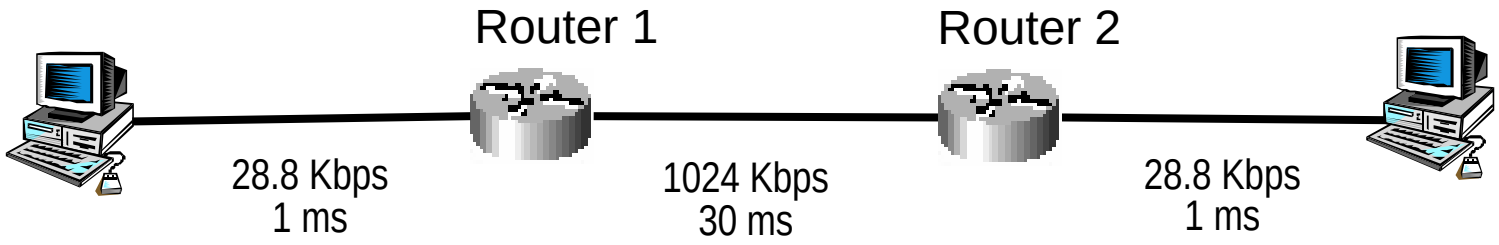
- Transmission delay:
  - C [bit/s] = link rate
  - B [bit] = packet size
  - transmission delay = B/C [sec]
- Example:
  - 512 bytes packet
  - 64 kbps link
  - transmission delay = 512*8/64000 = 64ms
- Propagation delay – constant depending on
  - Link length
  - Electromagnetig waves propagation speed in considered media
    - 200 km/s for copper links
    - 300 km/s in air
- Hypothesis: other delays neglected
  - queueing
  - processing

Stop-and-wait performance



Stop-and-wait performance

Numerical example



➔ Message:
- ⇨ 1024 bytes;
- ⇨ 2 segments: 536+488 bytes
- ⇨ Overhead: 20 bytes TCP + 20 bytes IP
- ⇨ ACK = 40 bytes (header only)

➔ Segment 1:
- ⇨ Tx1 = 576*8/28,8 = 160ms
- ⇨ Tx3 = Tx1
- ⇨ Tx2 = 576*8/1024 = 4,5 ms

➔ Segment 2:
- ⇨ Tx1 = 528*8/28,8 = 146,7ms
- ⇨ Tx3 = Tx1
- ⇨ Tx2 = 528*8/1024 = 4,1 ms

➔ Acks:
- ⇨ Tx1 = Tx3 = 40*8/28,8 = 11,1ms
- ⇨ Tx2 = 40*8/1024 = 0,3 ms

RESULT:

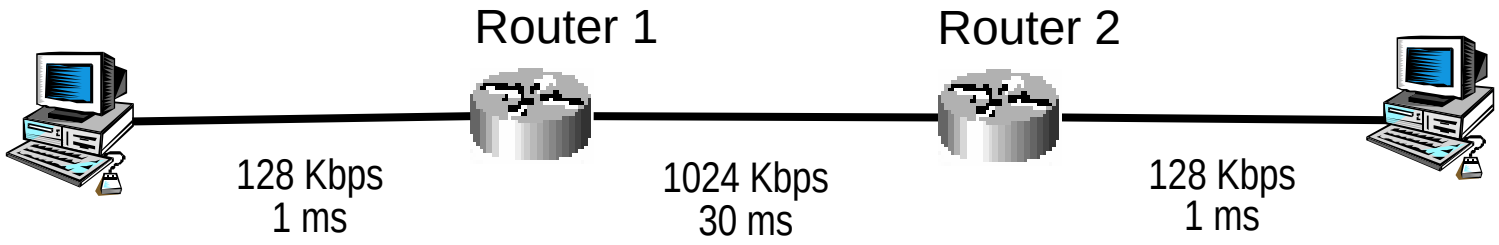
D = 667 (tx total) + 2*RTT =
= 795 ms

THR = 1024*8/795 =
= 10,3 kbps

Stop-and-wait performance

Numerical example

With ISDN?



➔ Segment 1:

- ⇨ Tx1 = Tx3 = 576*8/128 = 36ms
- ⇨ Tx2 = 576*8/1024 = 4,5 ms

➔ Segment 2:

- ⇨ Tx1 = Tx3 = 528*8/128 = 33 ms
- ⇨ Tx2 = 528*8/1024 = 4,1 ms

➔ Acks:

- ⇨ Tx1 = Tx3 = 40*8/128 = 2,5 ms
- ⇨ Tx2 = 40*8/1024 = 0,3 ms

RESULT:

D = 157,2 (tx total) + 2*RTT =
= 279,9 ms

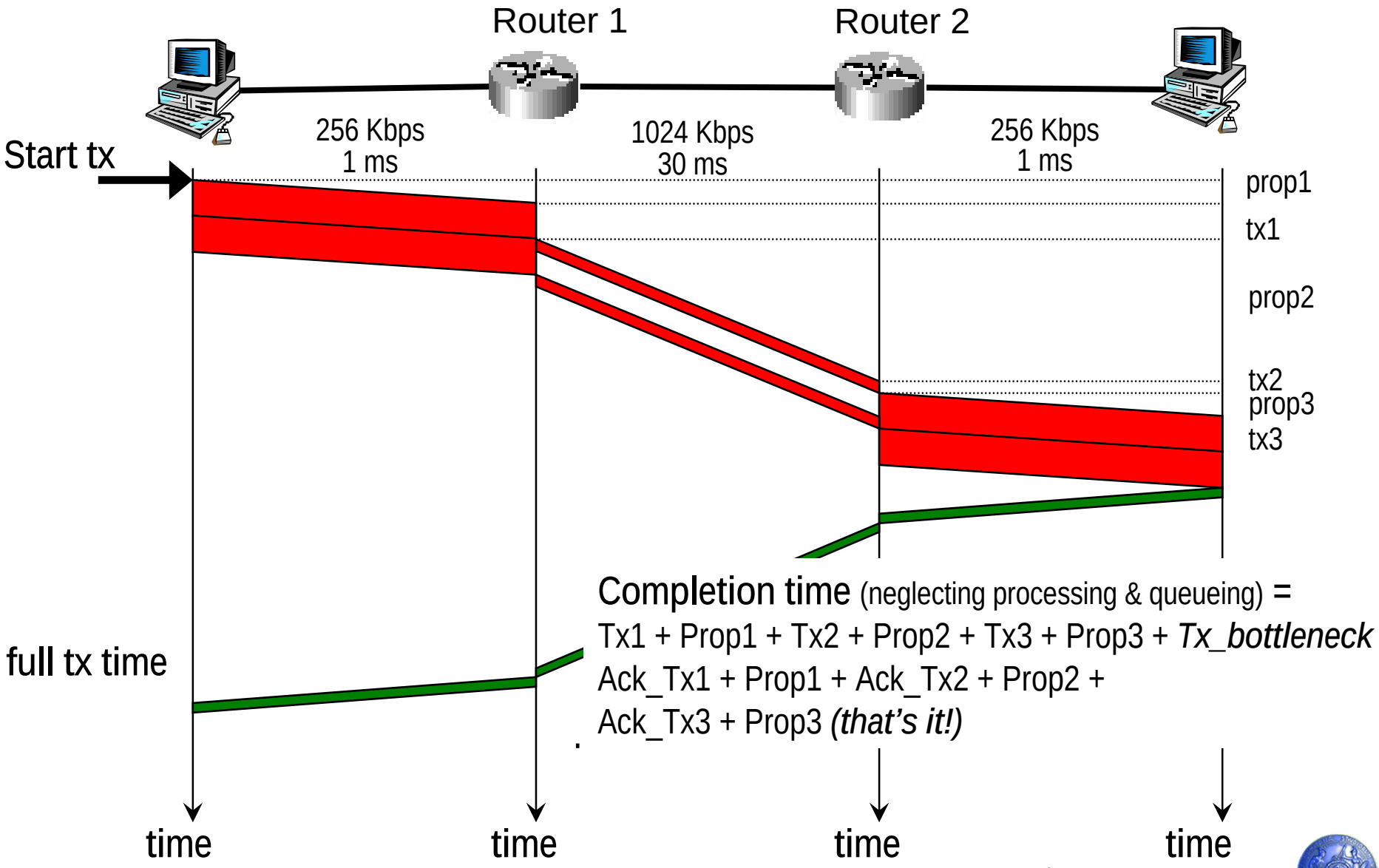
THR = 1024*8/279,9 =
= 29,3 kbps

on Gbps fiber optics?

D = negligible + 2*RTT =
= 128 ms

THR = 1024*8/128 =
= 64 kbps

Pipelining performance



Completion time (neglecting processing & queueing) =

Tx1 + Prop1 + Tx2 + Prop2 + Tx3 + Prop3 + *Tx_bottleneck*

Ack_Tx1 + Prop1 + Ack_Tx2 + Prop2 +

Ack_Tx3 + Prop3 *(that's it!)*

Pipelining performance
numerical example

On 28,8 kbps links

D = 347 (tx segm1+ack) + RTT +
+ 160 (segm2 bottleneck) =
= 571 ms

THR = 1024*8/571 =
= 14,3 kbps

On 128 kbps ISDN links

D = 81,8 (tx segm1+ack) + RTT +
+ 33 (segm2 bottleneck) =
= 178,8 ms

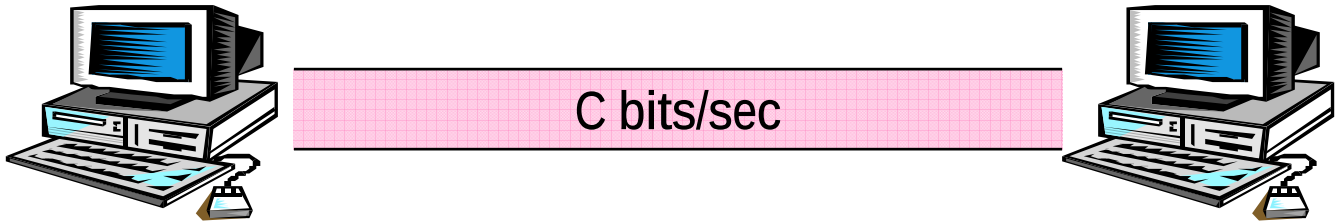
THR = 1024*8/178,8 =
= 45,8 kbps

on Gbps fiber optics?

D = negligible + RTT = 64 ms

THR = 1024*8/64 = 128 kbps

Simplified performance model

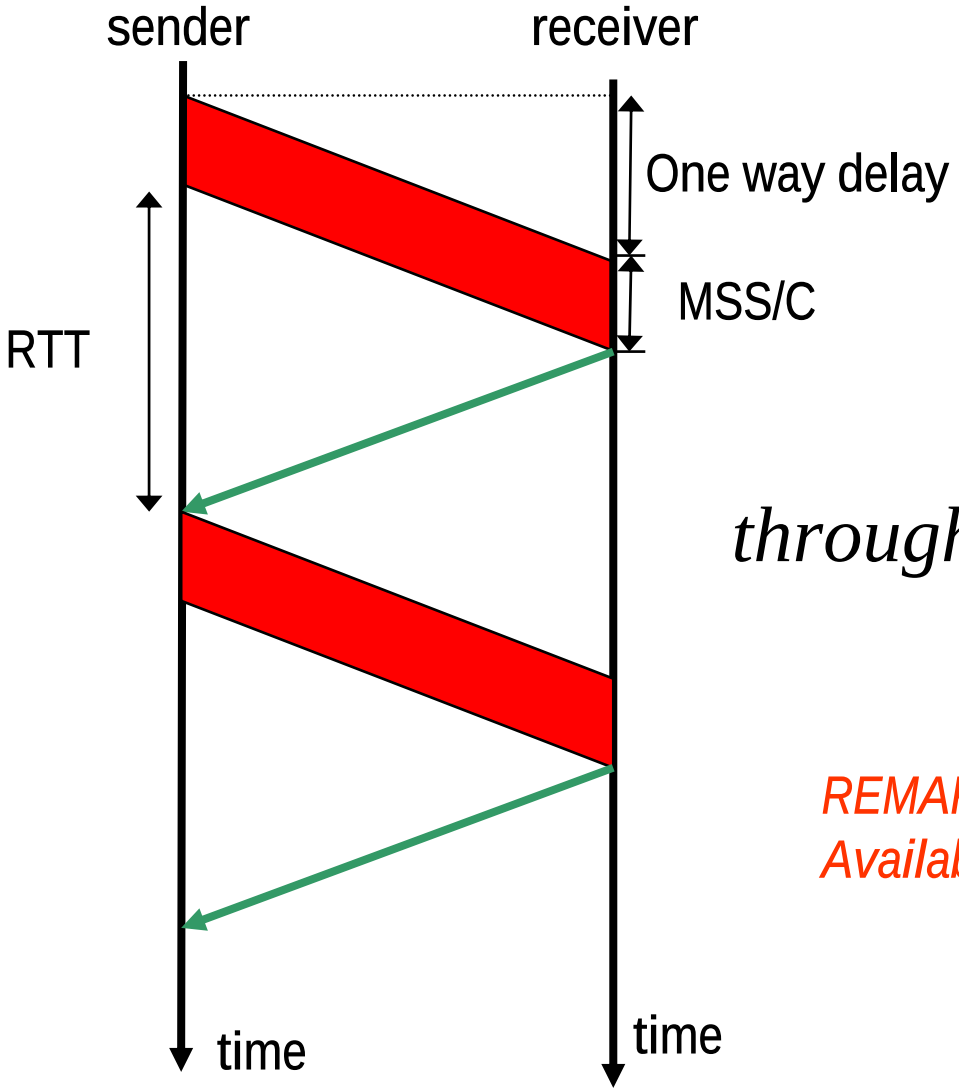


Approximate analysis, much simpler than multi-hop
Typically, C = bottleneck link rate

MSS = segment size
MSIZE = message size
Ignore overhead
Ignore ACK transmission time
No loss of segments
W = number of outstanding segments

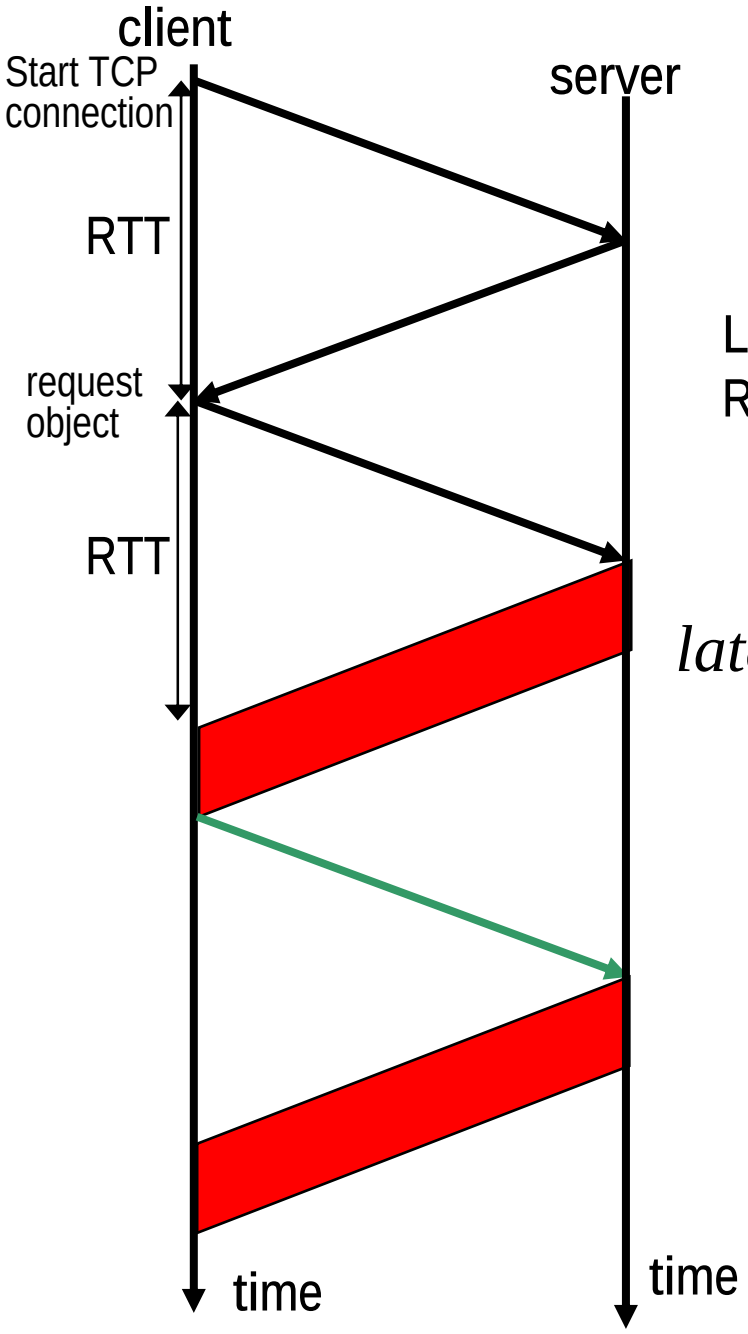
- W=1: stop-and-wait
- W>1: sliding window
- *This is a highly dynamic parameter in TCP!!*
- *For now, consider W fixed*

W=1 case (stop-and-wait)



$$throughput = \frac{MSS}{RTT + MSS / C}$$

REMARK: throughput always lower than Available link rate!



Latency in TCP retrieval model

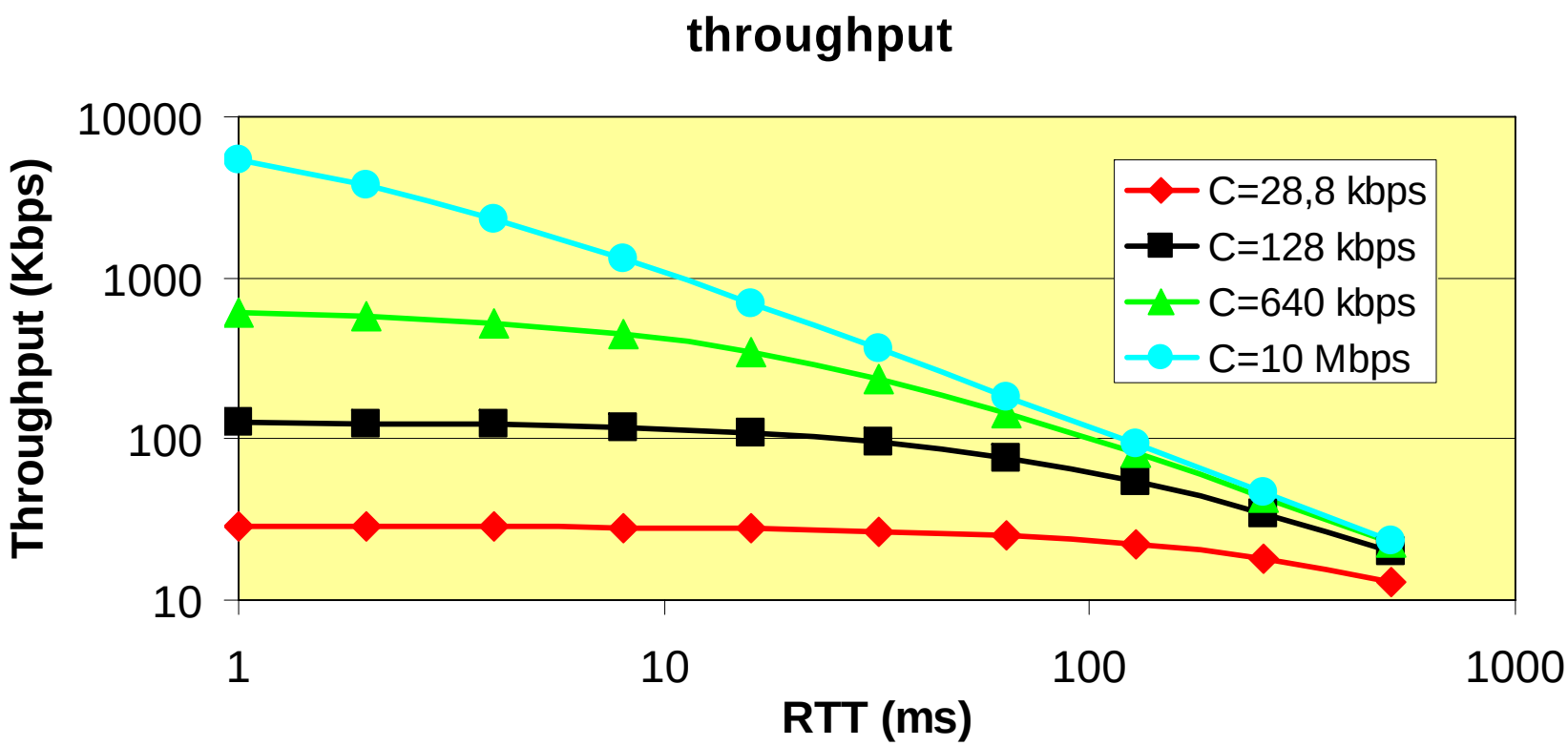
Latency: time elapsing between TCP connection Request, and last bit received at client

$$latency = 2RTT + \frac{MSIZE}{C} + \left\lceil \frac{MSIZE}{MSS} - 1 \right\rceil RTT$$

Number of segments In which message is split

W=1 case (stop-and-wait)

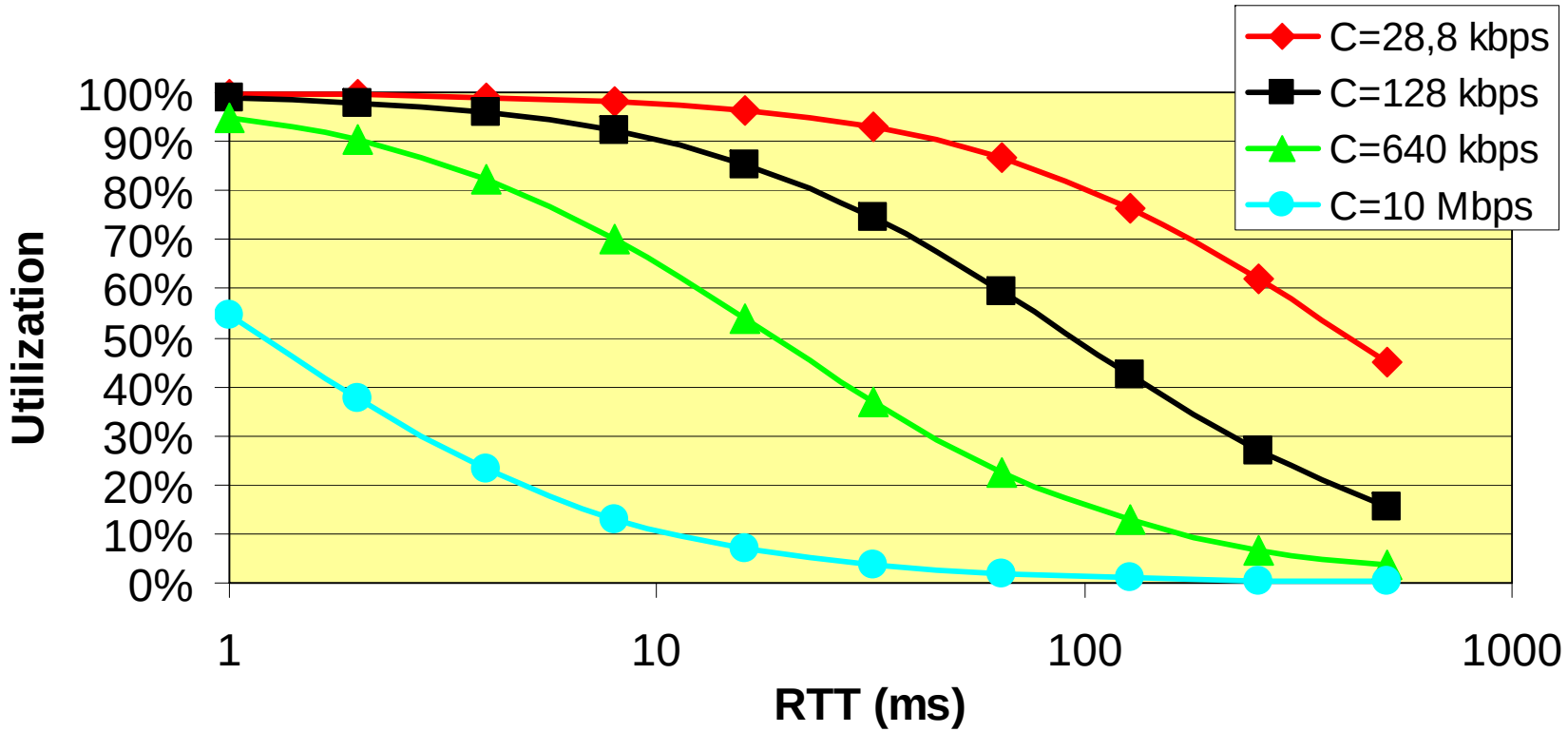
MSS = 1500 bytes



Under-utilization with: 1) high capacity links, 2) large RTT links

W=1 case (stop-and-wait)

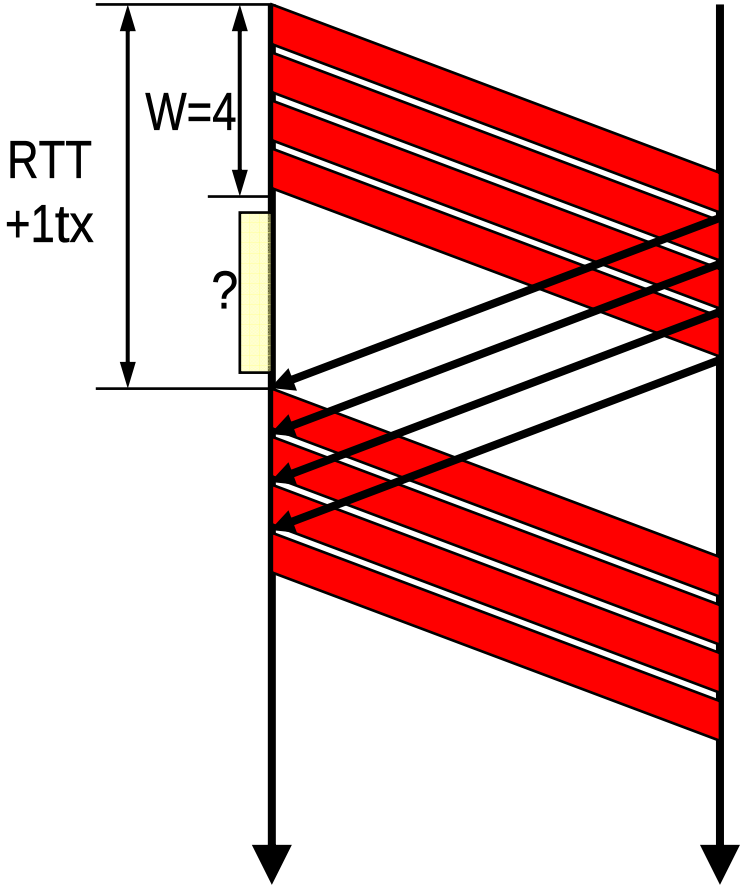
MSS = 1500 bytes



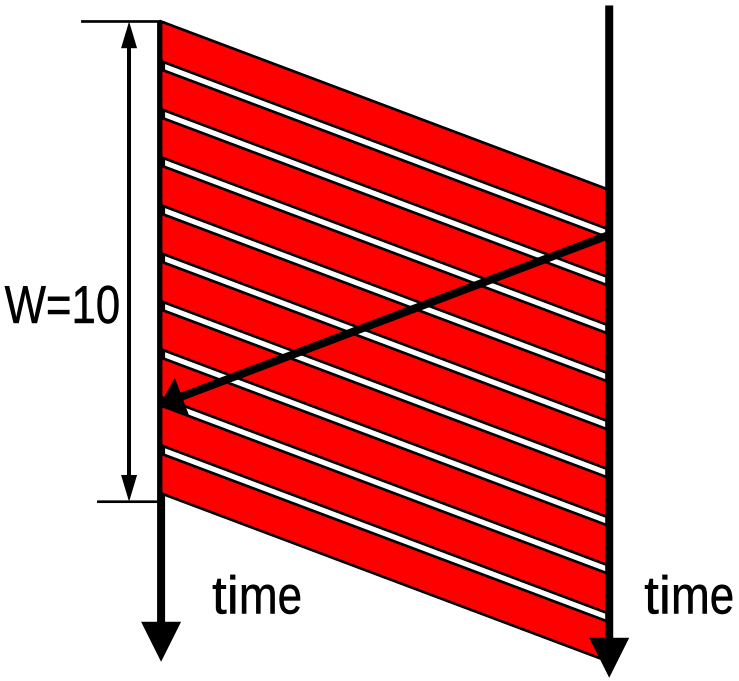
Under-utilization with: 1) high capacity links, 2) large RTT links

Pipelining (W>1) analysis

two cases



UNDER-SIZED WINDOW:
THROUGHPUT INEFFICIENCY



WINDOW SIZE that allows
CONTINUOUS TRANSMISSION

Continuous transmission

Condition in which link rate is fully utilized

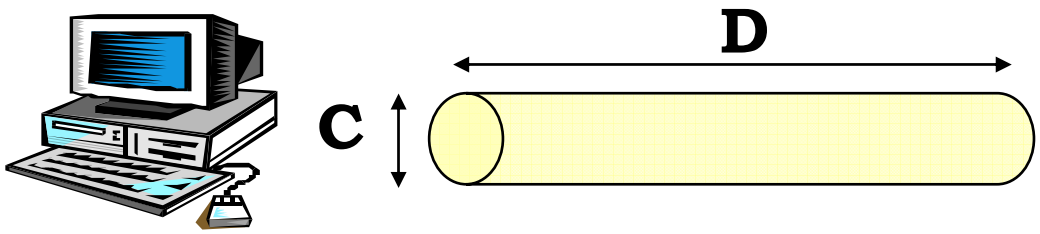
$$\underbrace{W \cdot \frac{MSS}{C}}_{\text{Time to transmit W segments}} > \underbrace{RTT + \frac{MSS}{C}}_{\text{Time to receive Ack of first segment}}$$

We may elaborate:

$$W \cdot MSS > RTT \cdot C + MSS \approx RTT \cdot C$$

This means that full link utilization is possible when window size (in bits) is Greater than the bandwidth (C bit/s) delay (RTT s) product!

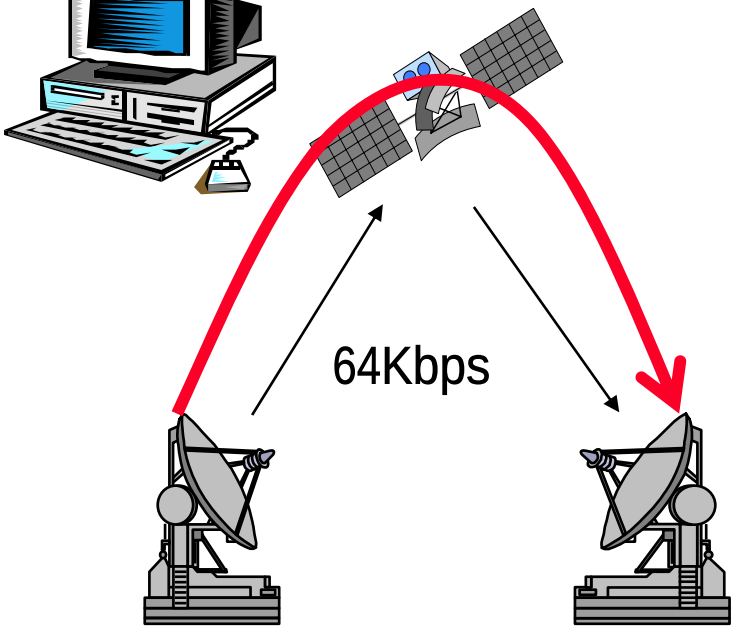
Bandwidth-delay product



➔**Network: like a pipe**

➔**C [bit/s] x *D* [s]**

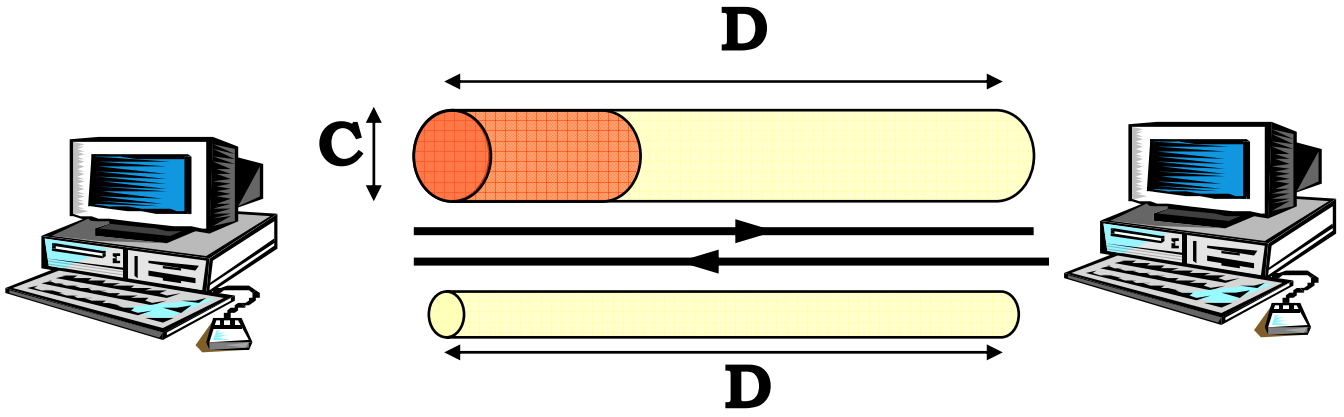
- ⇨ number of bits “flying” in the network
- ⇨ number of bits injected in the network by the tx, before that the first bit is received at distance D s



A 15360 (64000x0.240) bits “worm” in the air!!

bandwidth-delay product = bytes # that saturates network pipe

Bandwidth-delay product (usually considered)



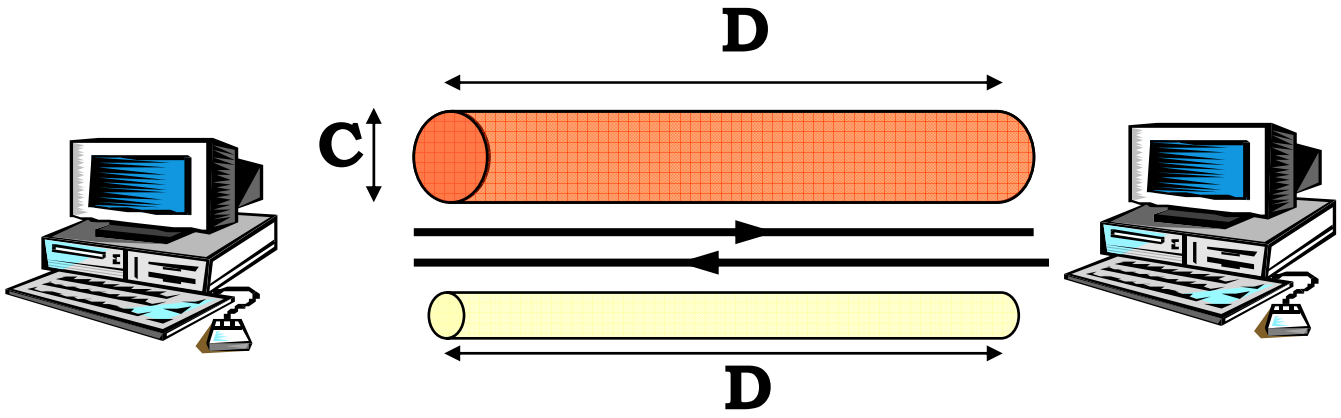
RTT=2*D

After D/4 s

➔ **C [bit/s] x *RTT* [s]**

- ⇨ number of bits “flying” in the network (partly received)
- ⇨ number of bits injected in the network by the tx, before that the ack of the first bit is received

Bandwidth-delay product (usually considered)

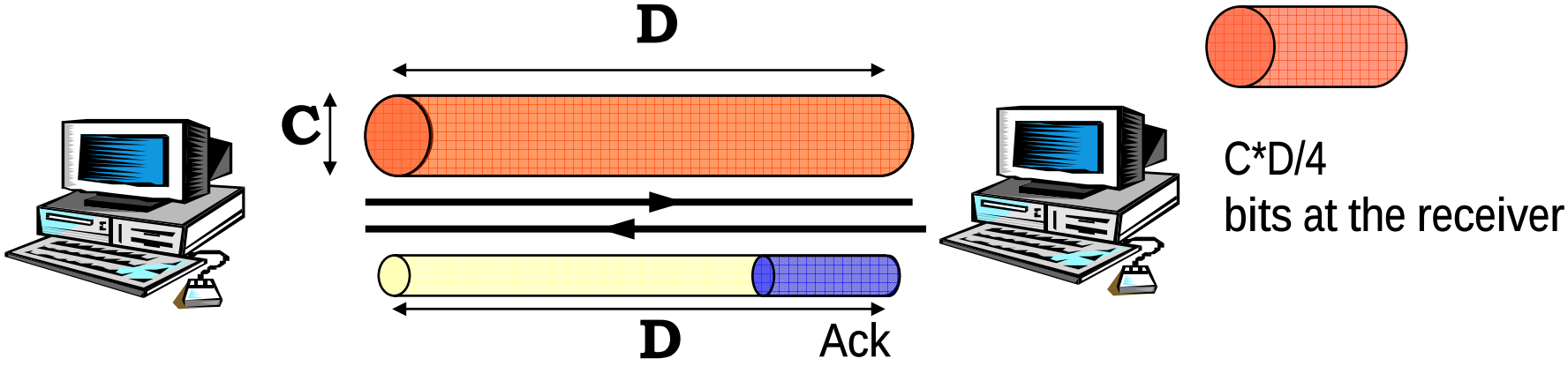


After D s

➔ **C [bit/s] x *RTT* [s]**

- ⇨ number of bits “flying” in the network (partly received)
- ⇨ number of bits injected in the network by the tx, before that the ack of the first bit is received

Bandwidth-delay product (usually considered)

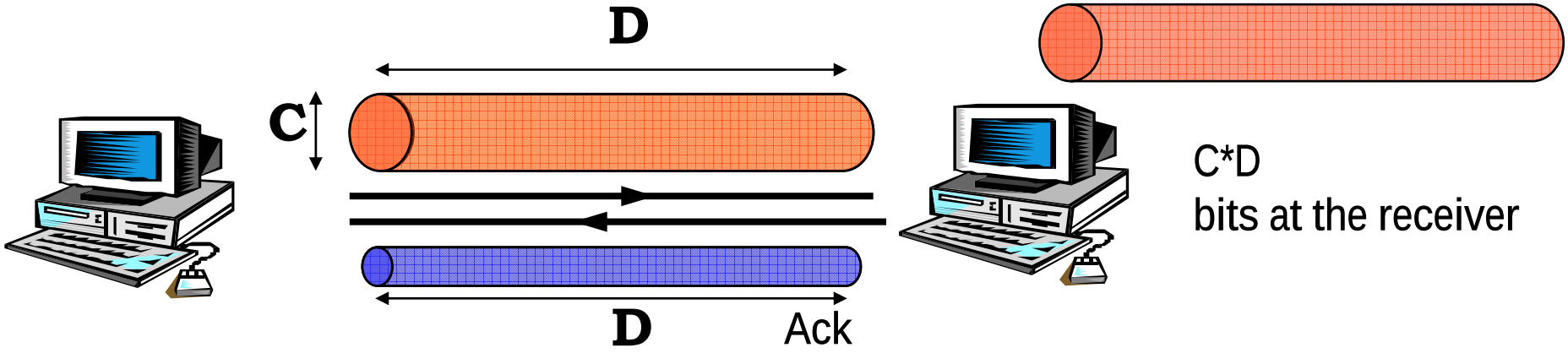


After 5/4*D s

➔ **C [bit/s] x *RTT* [s]**

- ⇨ number of bits “flying” in the network (partly received)
- ⇨ number of bits injected in the network by the tx, before that the ack of the first bit is received

Bandwidth-delay product (usually considered)

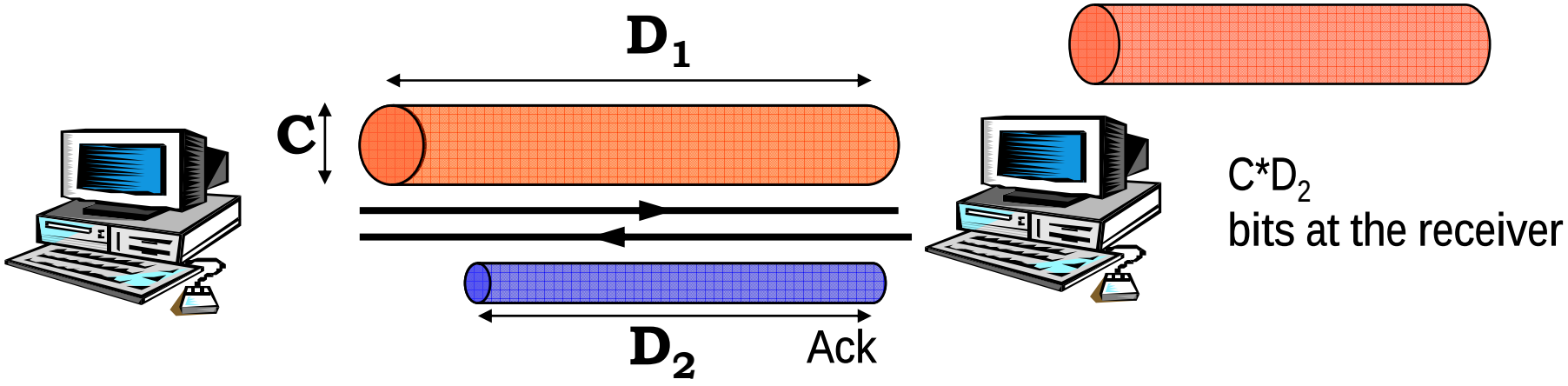


After 2D s

➔ **C [bit/s] x *RTT* [s]**

- ⇨ number of bits “flying” in the network (partly received)
- ⇨ number of bits injected in the network by the tx, before that the ack of the first bit is received

Bandwidth-delay product (usually considered)



After RTT=D_1+D_2 s

➔ **C [bit/s] x *RTT* [s]**

- ⇨ number of bits “flying” in the network (partly received)
- ⇨ number of bits injected in the network by the tx, before that the ack of the first bit is received

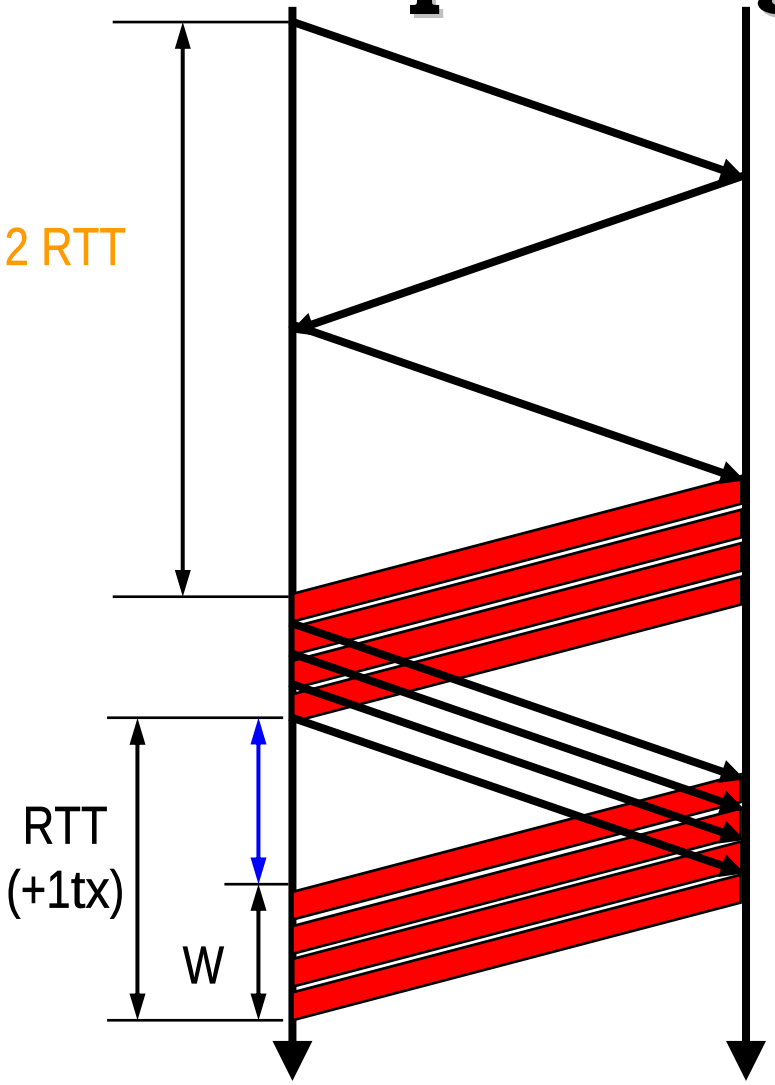
Long Fat Networks

LFNs (el-ef-an(t)s): large bandwidth-delay product

NETWORK	RTT (ms)	rate (kbps)	BxD (bytes)
Ethernet	3	10.000	3.750
T1, transUS	60	1.544	11.580
T1 satellite	480	1.544	92.640
T3 transUS	60	45.000	337.500
Gigabit transUS	60	1.000.000	7.500.000

The 65535 (16 bit field in TCP header) maximum window size W may be a limiting factor!

Pipelining (W>1) analysis



$$thr = \min\left(C, \frac{W \cdot MSS}{RTT + MSS/C}\right)$$

Delay analysis (for TCP object retrieval) – Non continuous transmission

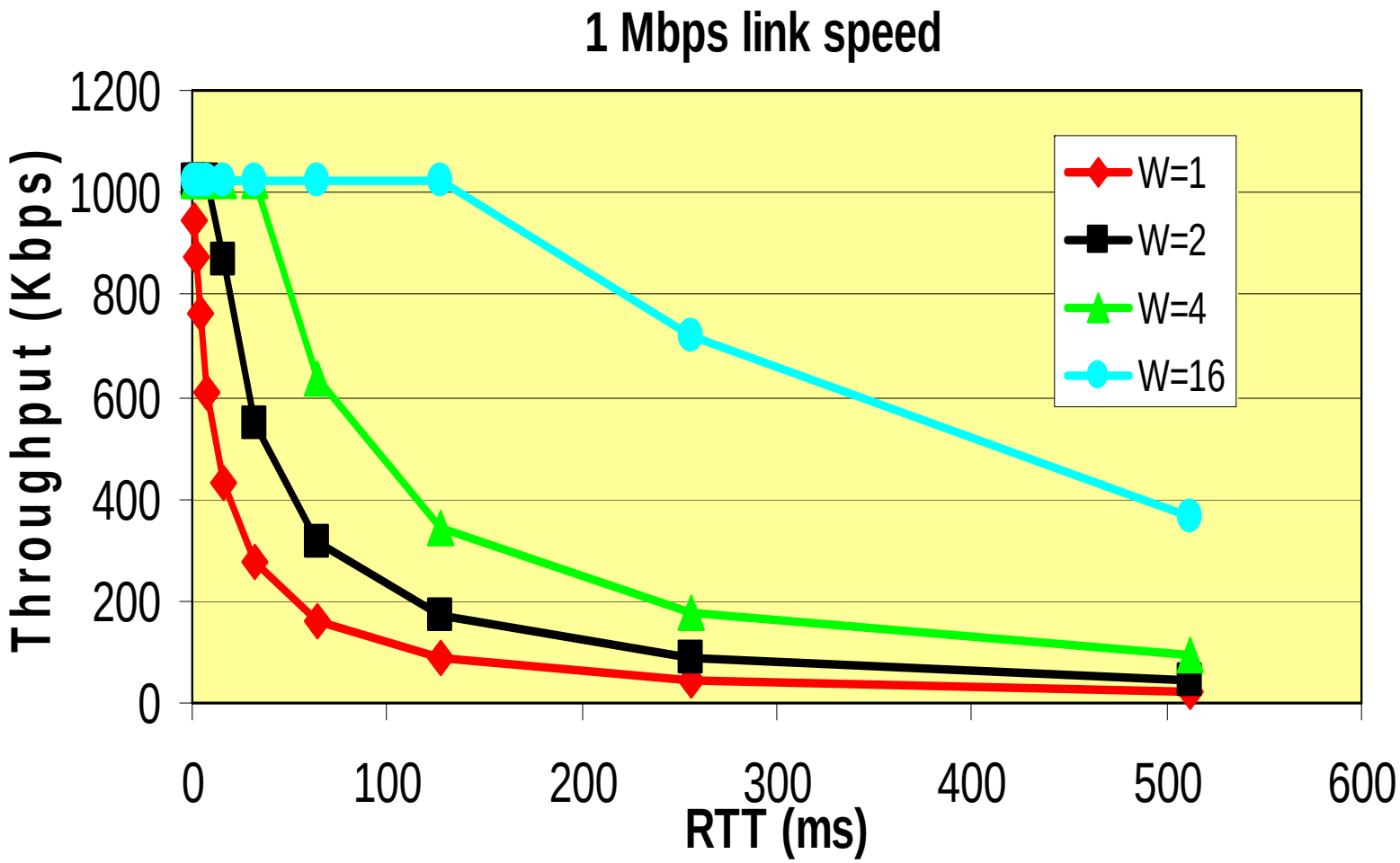
$$latency = 2RTT + \frac{MSIZE}{C} + \left\lceil \frac{MSIZE}{W \cdot MSS} - 1 \right\rceil \left(RTT - \frac{(W-1)MSS}{C} \right)$$

Continuous transmission case:

$$latency = 2RTT + \frac{MSIZE}{C}$$

Throughput for pipelining

MSS = 1500 bytes



Università degli Studi di Palermo

Maximum achievable throughput
(assuming infinite speed line...)