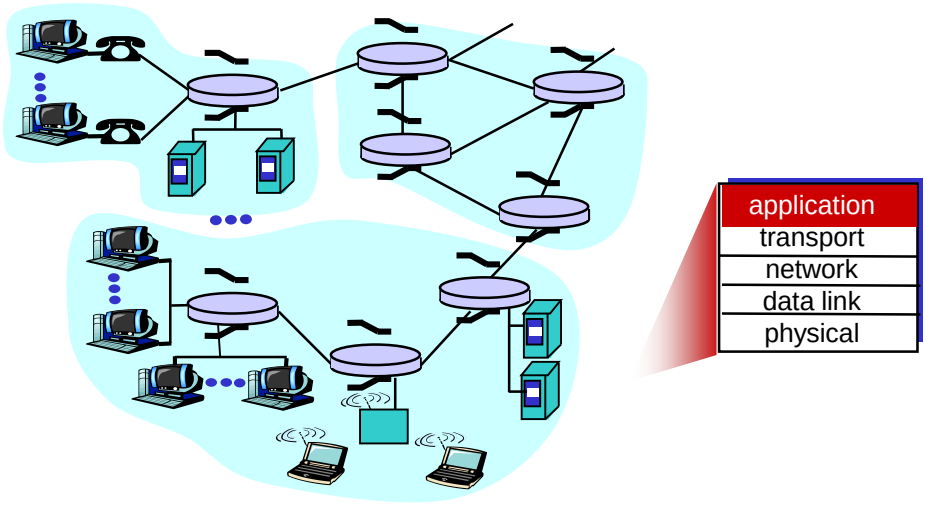
CS 4390
Computer Networks



Session 09

Transport Layer – Pipelined Protocols

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

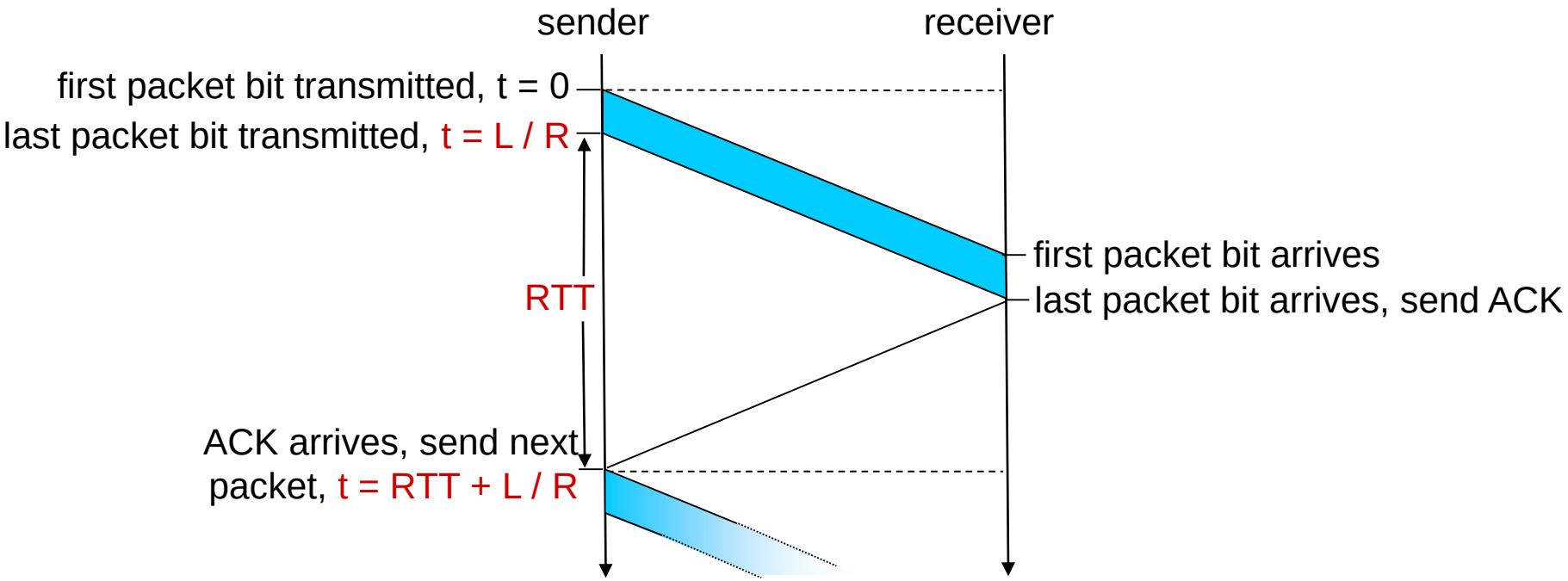
  - U $_{sender}$: *utilization* – fraction of time sender busy sending

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

  - *if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link!!!*

- network protocol limits use of physical resources
- Need to identify the cause and fix it: *why utilization is so low*?

Stop-and-wait Operation of rdt3.0



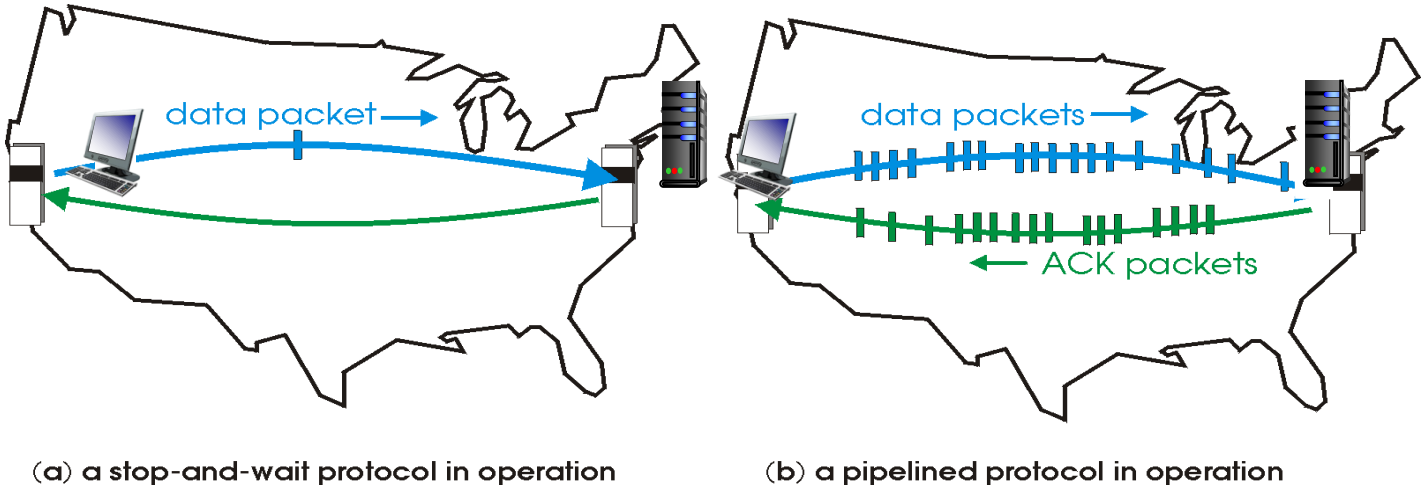
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

How to improve channel utilization?

Pipelined Protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver



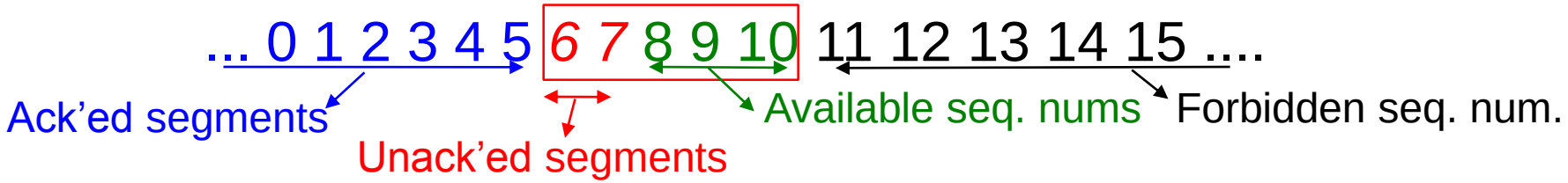
(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- No more 'stop-and-wait'!
- Generic form of '*sliding window*' protocols

Sliding Window Protocol

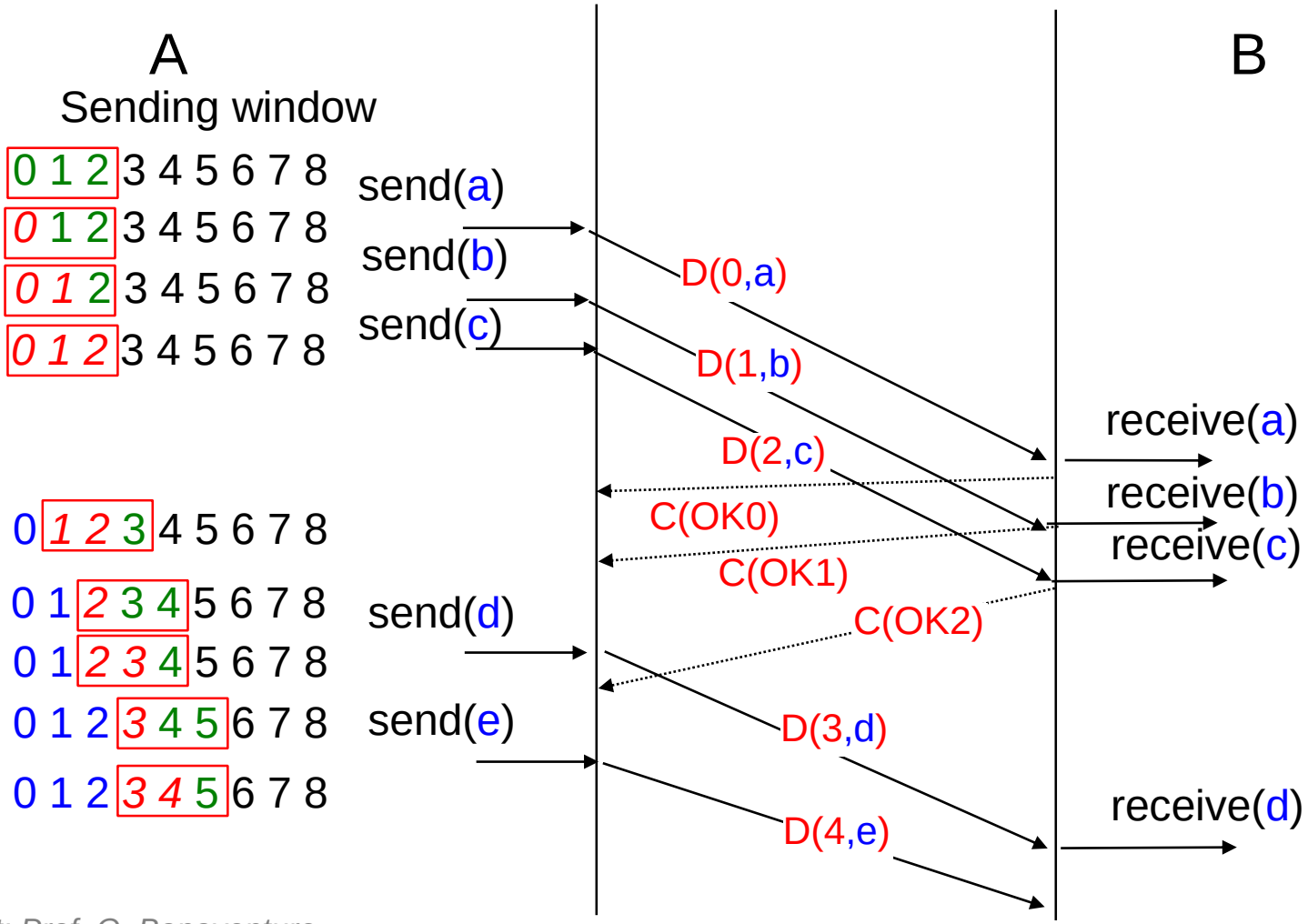
- To avoid 'stop-and-wait' behavior
  - Sender keeps a list of all the segments that it is allowed to send: *sending_window*
  - Receiver also maintains a receiving window with the list of acceptable sequence numbers: *receiving_window*
  - During data transfer the windows appear to be sliding across segment sequential numbers
- Sender and receiver must use compatible windows (e.g. negotiated during connection establishment phase)



adapted from slides by Prof. O. Bonaventure

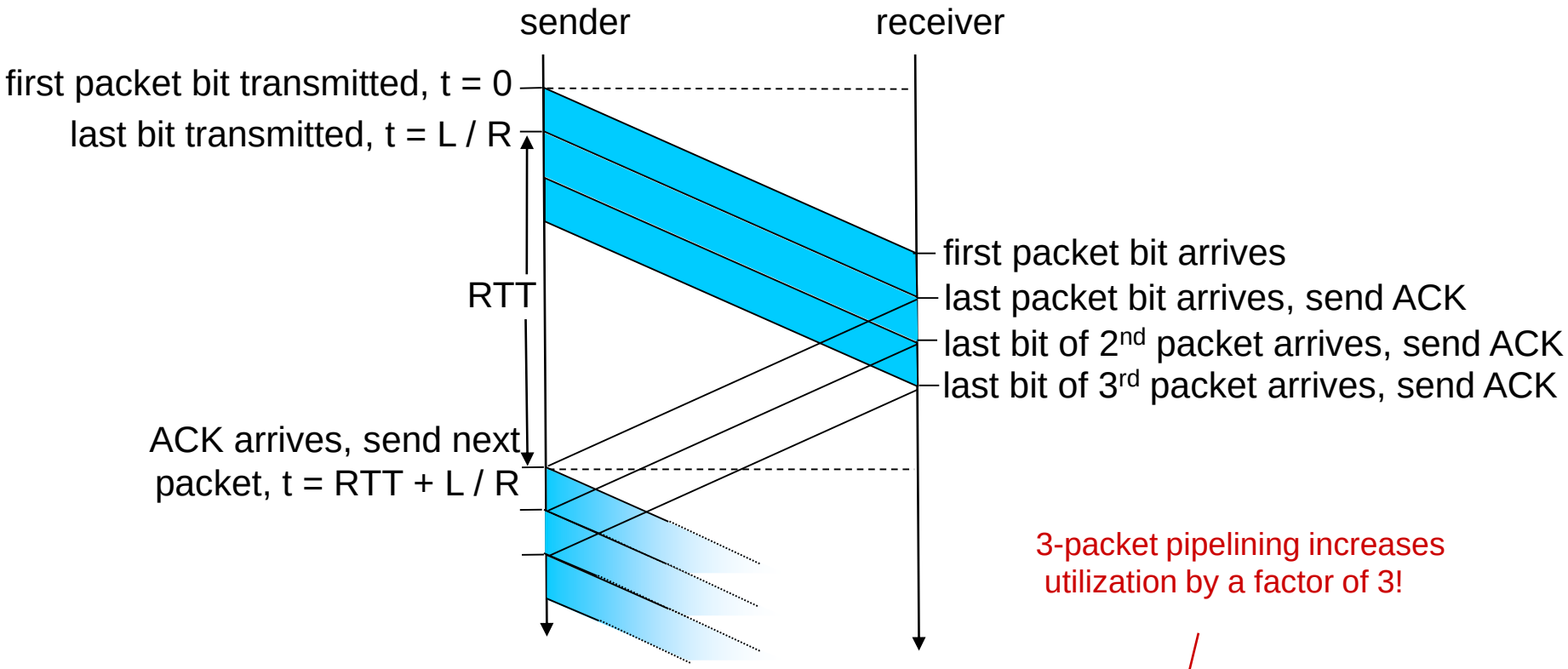
Sliding Windows Example

- Sending and receiving window size : 3 segments



credit: Prof. O. Bonaventure

Pipelining: Increased Utilization



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

rdt with Pipelined Protocol

- *Problem:* how to provide reliable data transfer with a pipelined protocol?
- Basic solutions:
  1. Go-Back-N
     - simple implementation (particularly on receiving side)
     - throughput will be limited when losses occur
  2. Selective Repeat
     - more difficult from an implementation viewpoint
     - throughput can remain high when limited losses occur

Pipelined Protocols: Overview

Go-back-N:

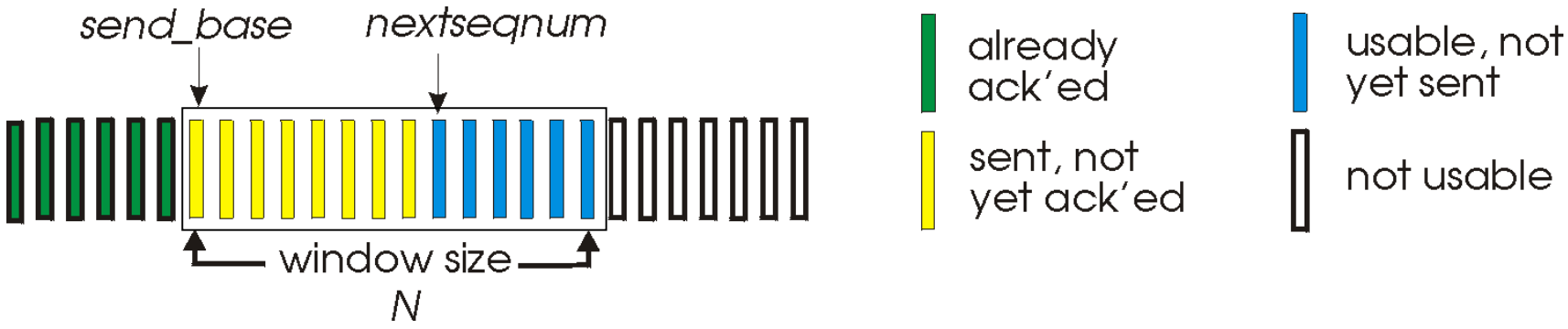
- sender can have up to N unack'ed packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unack'ed packet
  - when timer expires, retransmit *all* unack'ed packets

Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unack'ed packet
  - when timer expires, retransmit only that unack'ed packet

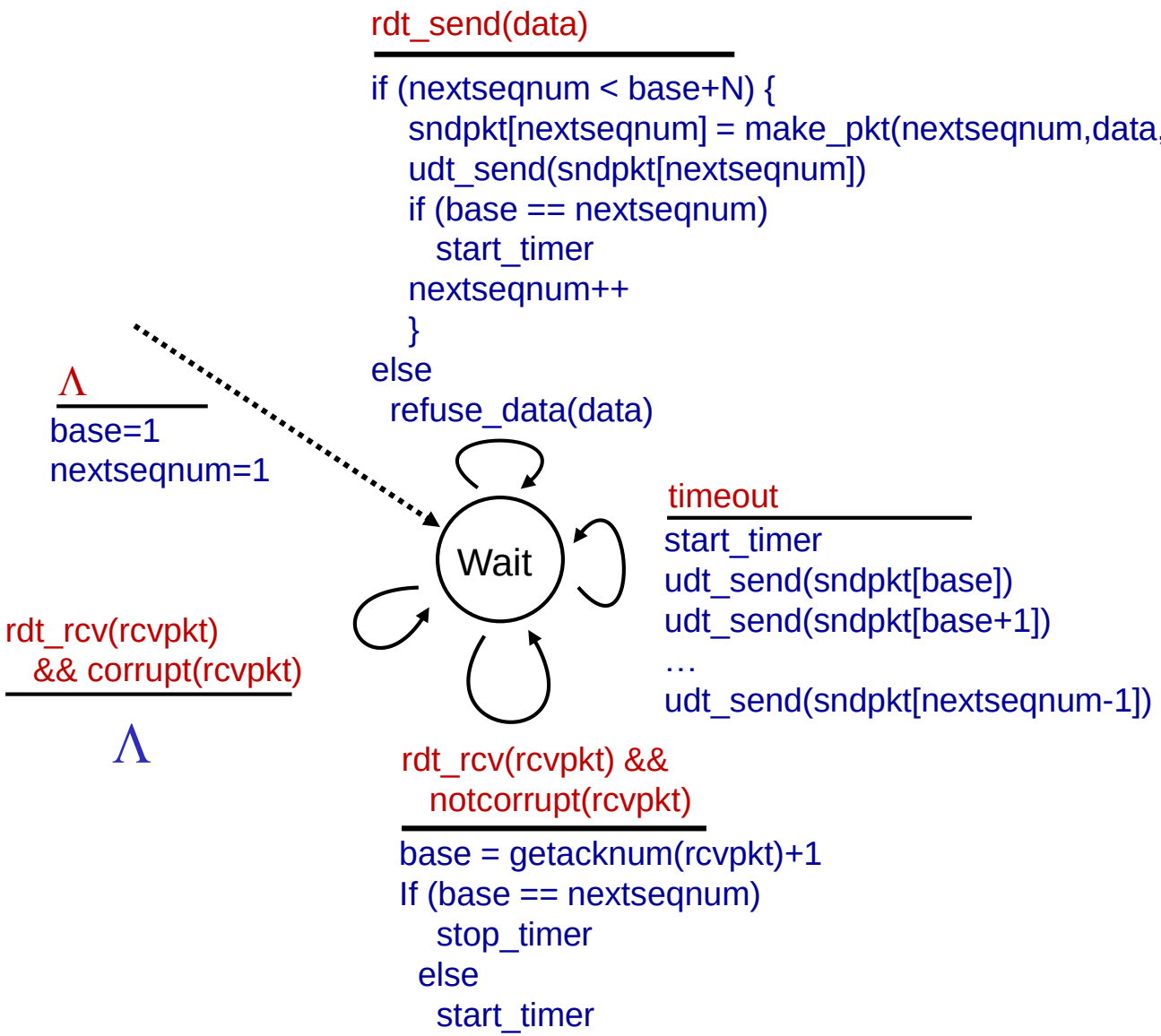
Go-Back-N: Sender

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

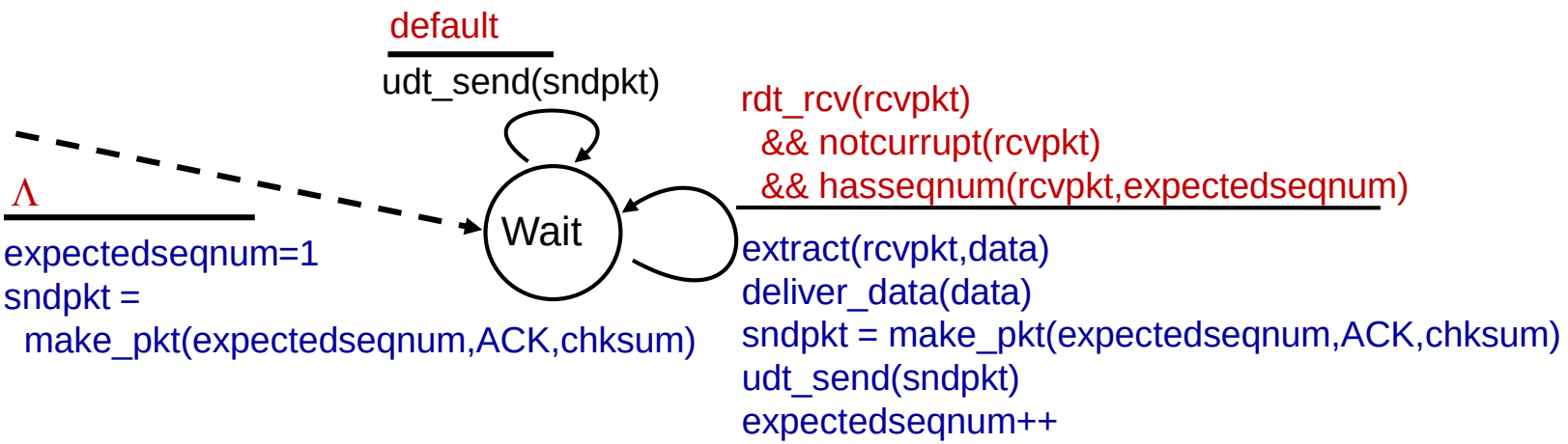


- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

GBN: Extended FSM for Sender



GBN: Extended FSM for Receiver

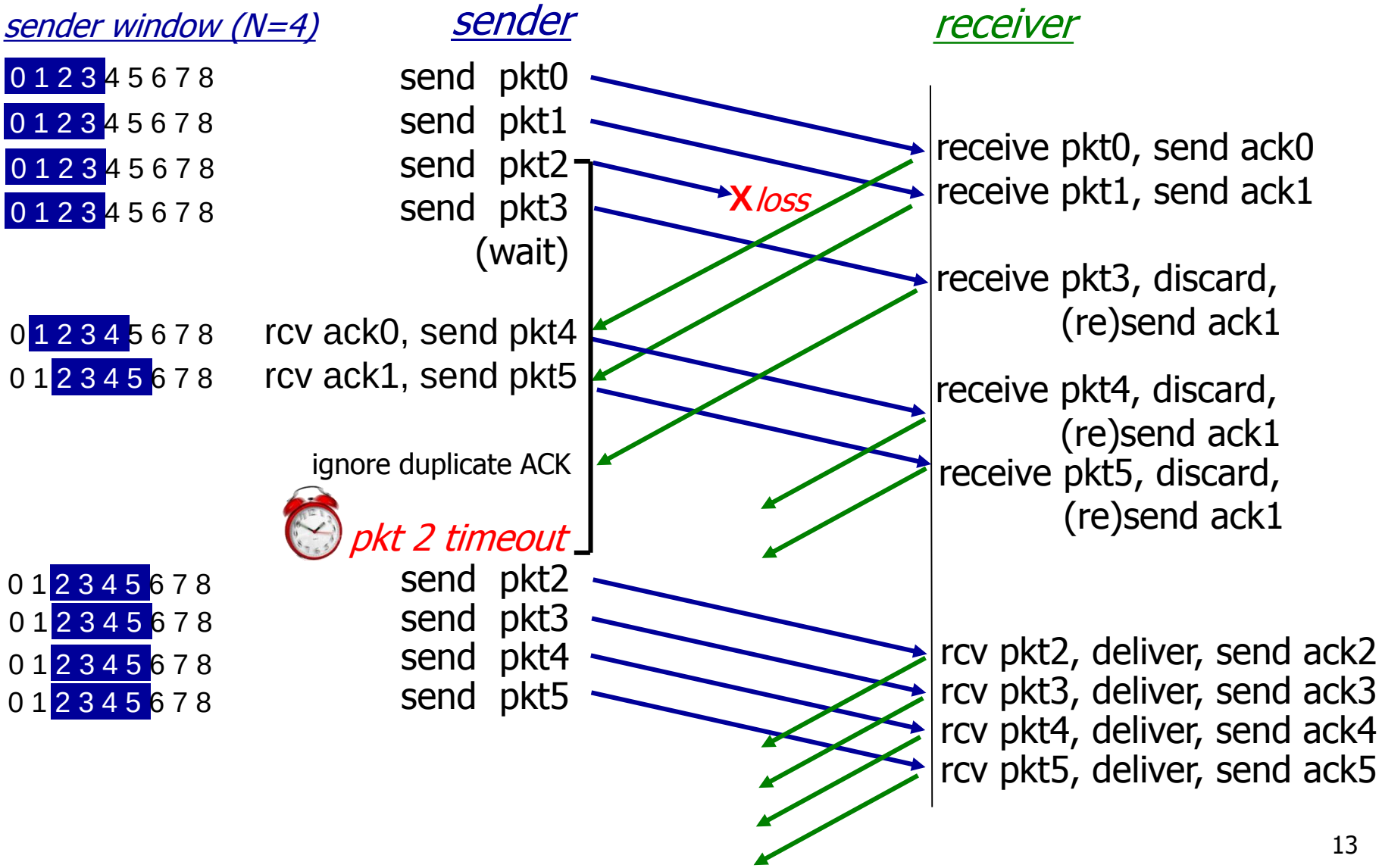


ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

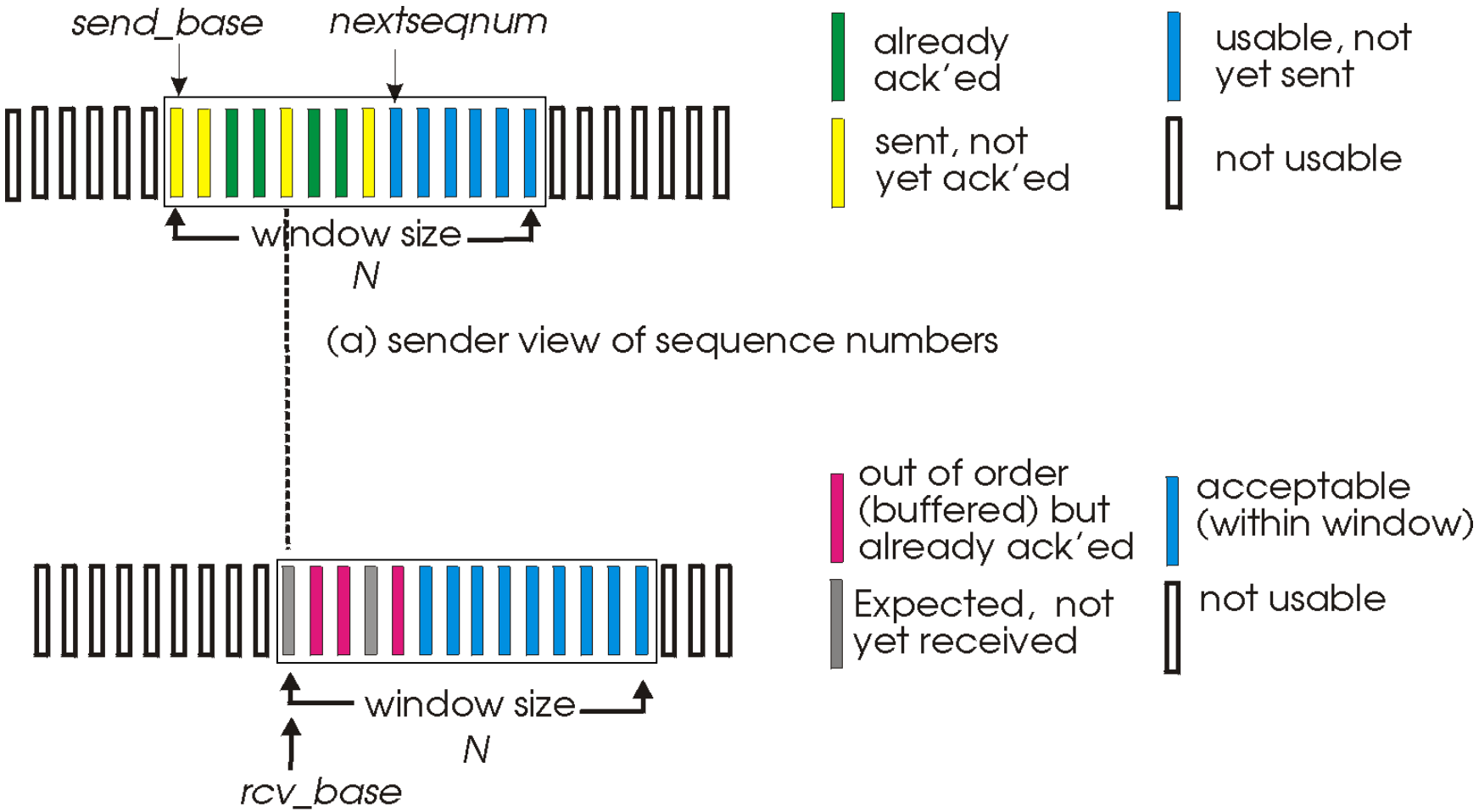
GBN in Action



Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACK'ed pkt
- sender window
  - *N* consecutive seq #'s
  - limits seq #s of sent, unACK'ed pkts

Selective Repeat: sender, receiver Windows



Checkpoint 9-1

What about the receiver window in Go Back N?

a) GBN has no receiver window

b) GBN has a window of size 1

c) GBN has the same receiver window as Selective Repeat

d) It depends

Selective Repeat

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

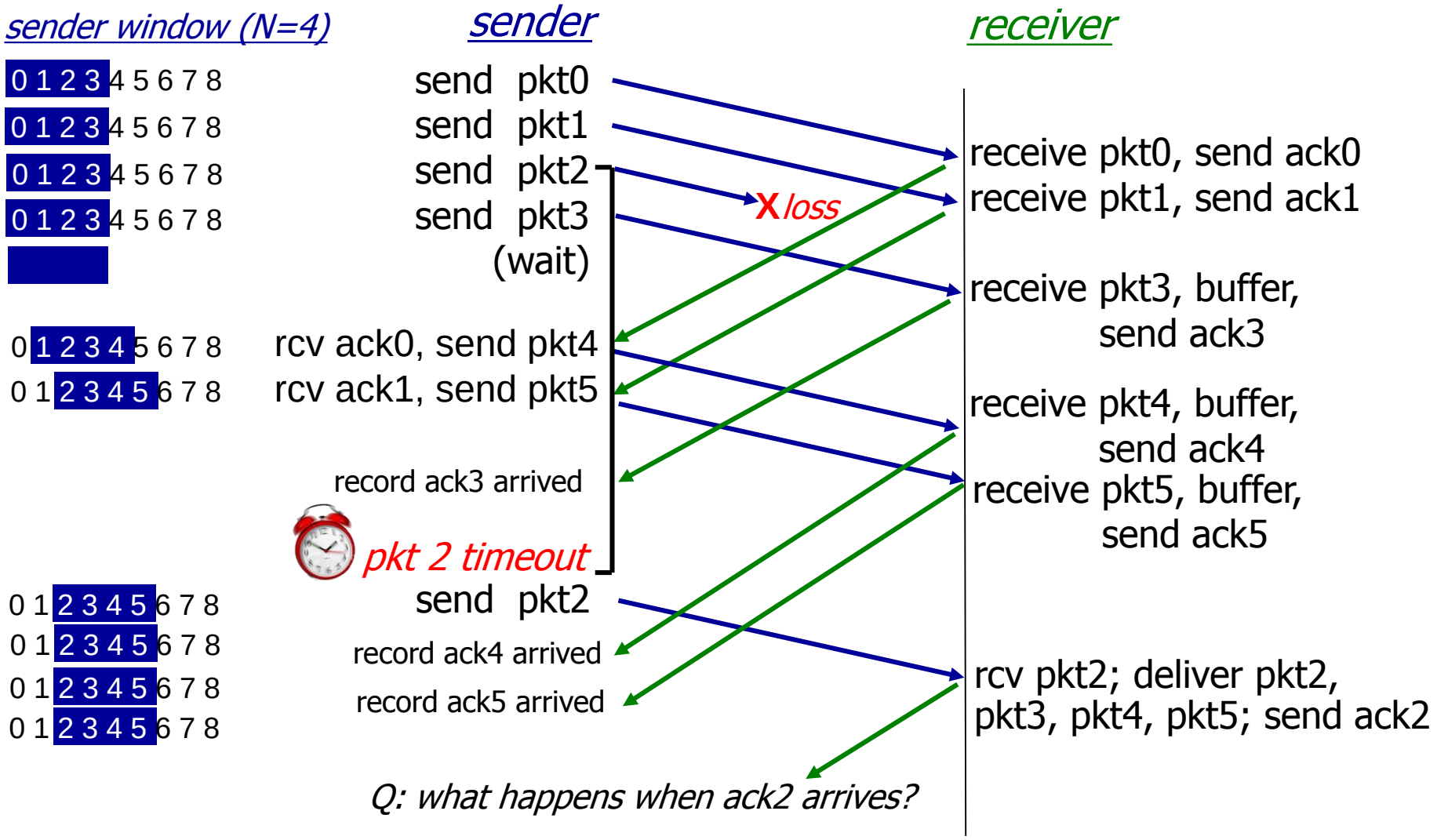
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective Repeat in Action



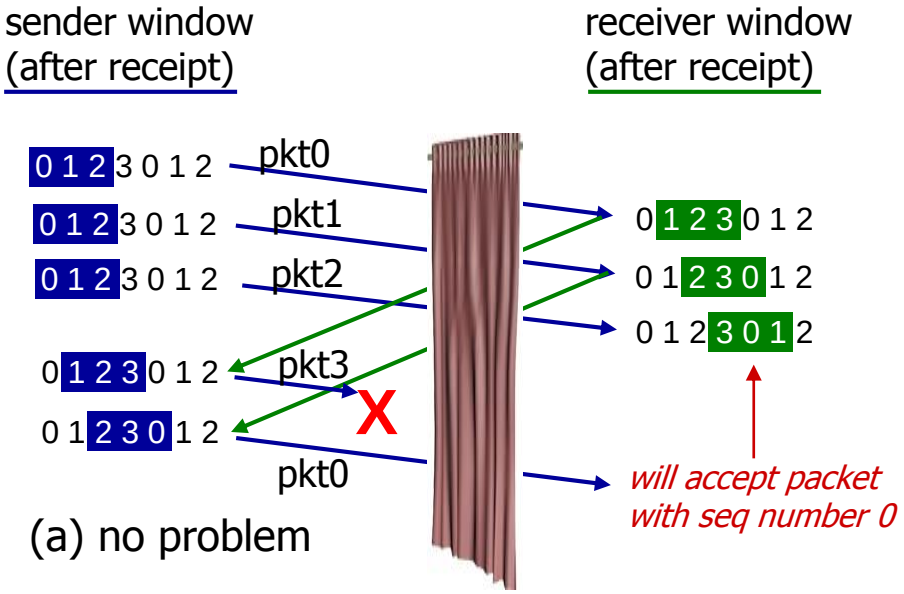
Selective Repeat: Dilemma

example:

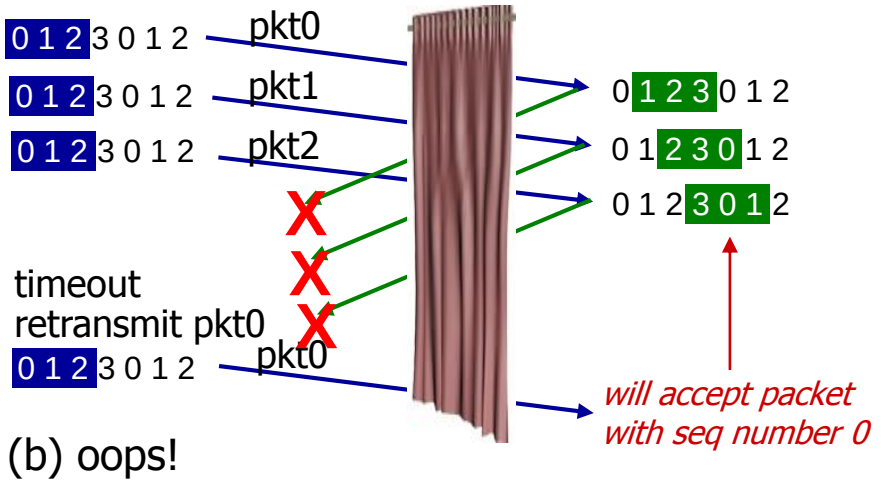
- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?



receiver can't see sender side.
receiver behavior identical in both cases!
something is (very) wrong!



Pointers to Corresponding Sections of Textbook

6th edition

Sections 3.4.2, 3.4.3, 3.4.4

7th edition

Sections 3.4.2, 3.4.3, 3.4.4