Chapter 3
Transport Layer

A note on the use of these PowerPoint slides:

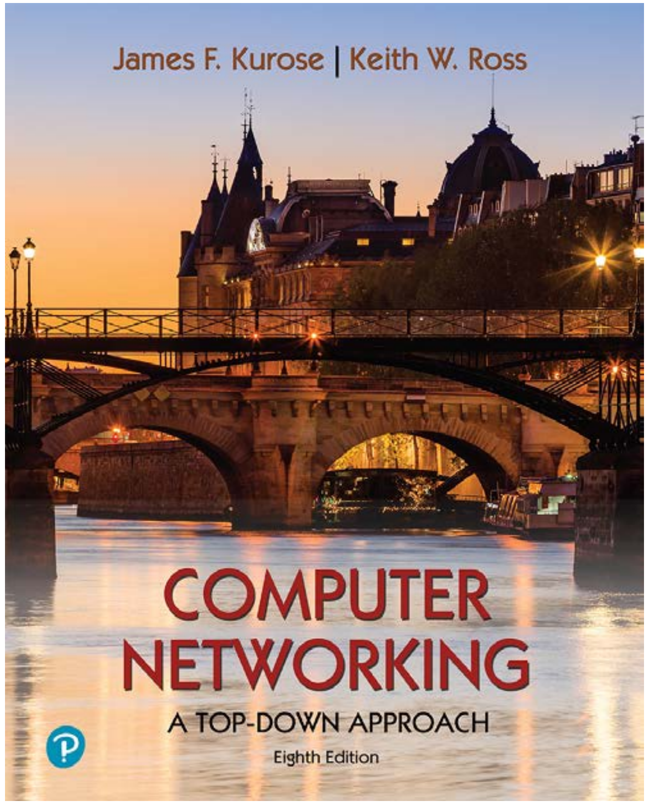
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

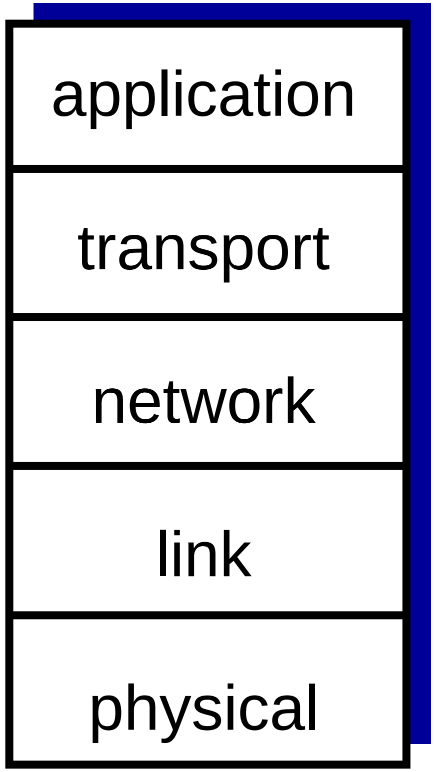
All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

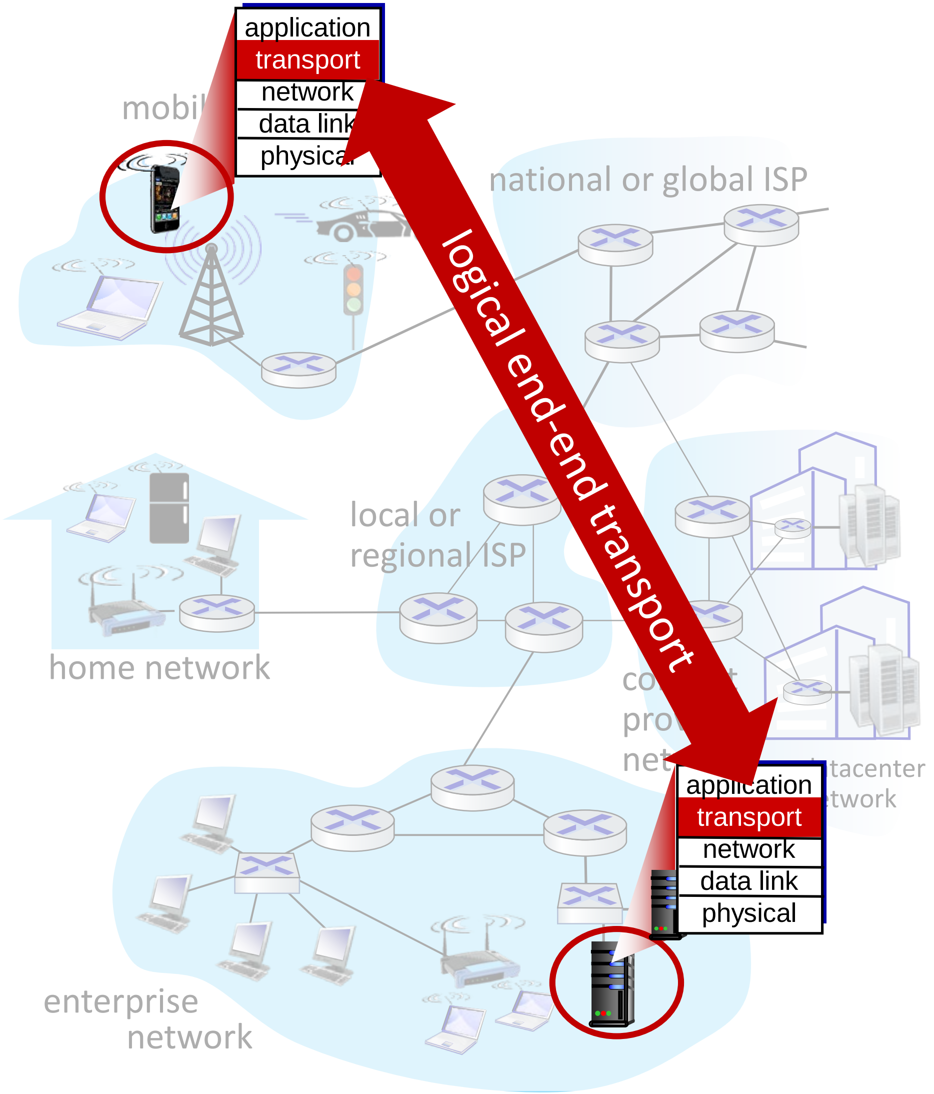
Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
  - TCP, UDP



-
برخلاف پروتکل لایه شبکه که وظیفه رسیدن اطلاعات از مبدا به مقصد از طریق نودهای توی شبکه رو داره پروتکل لایه انتقال یا لایه Transport فقط در نودهای انتهایی اجرا میشه و وظیفه اون رسیدن اطلاعات از مبدا هست به مقصد
لایه Transport اطلاعاتی که از لایه اپلیکیشن دریافت کرده که در قالب یک مسیج است در قالب سگمنت هایی به لایه شبکه میده و در لایه مقصد هم این سگمنت رو از لایه های شبکه دریافت میکنه و این ها رو کنار هم می ذاره و مسیج اپلیکیشن رو بازسازی می کنه و به لایه اپلیکیشن میده
در اینترنت ما دوتا پروتکل برای لایه Transport داریم: UDP , TCP

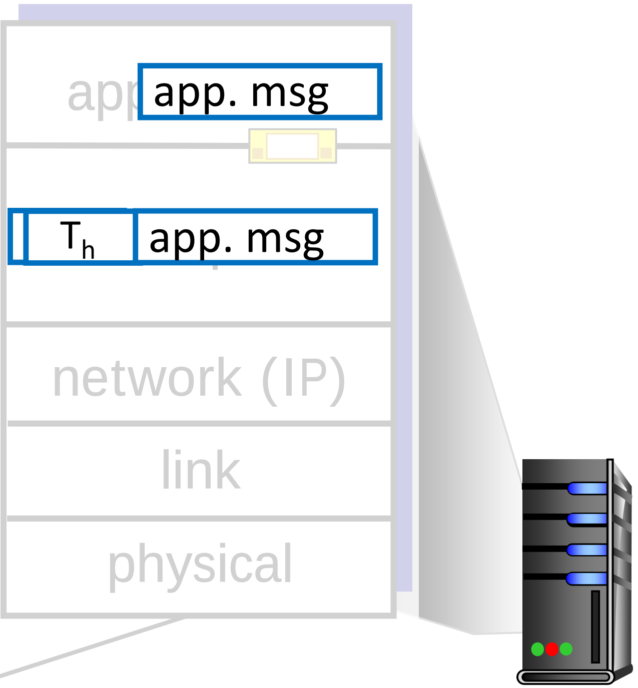
Transport vs. network layer services and protocols

- network layer: logical communication between *hosts*
- transport layer*:* logical communication between *processes*
  - relies on, enhances, network layer services

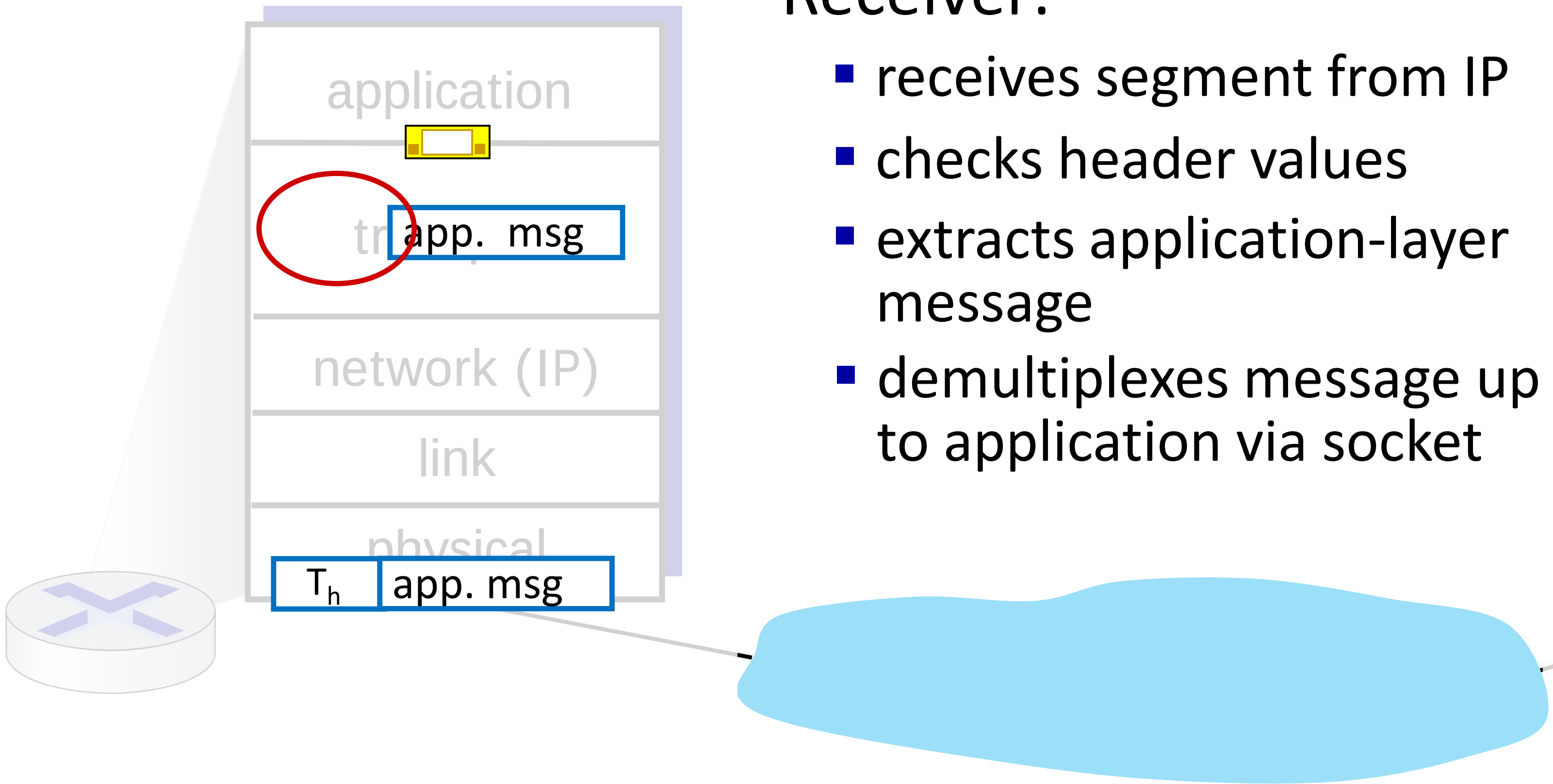
Transport Layer Actions

Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP

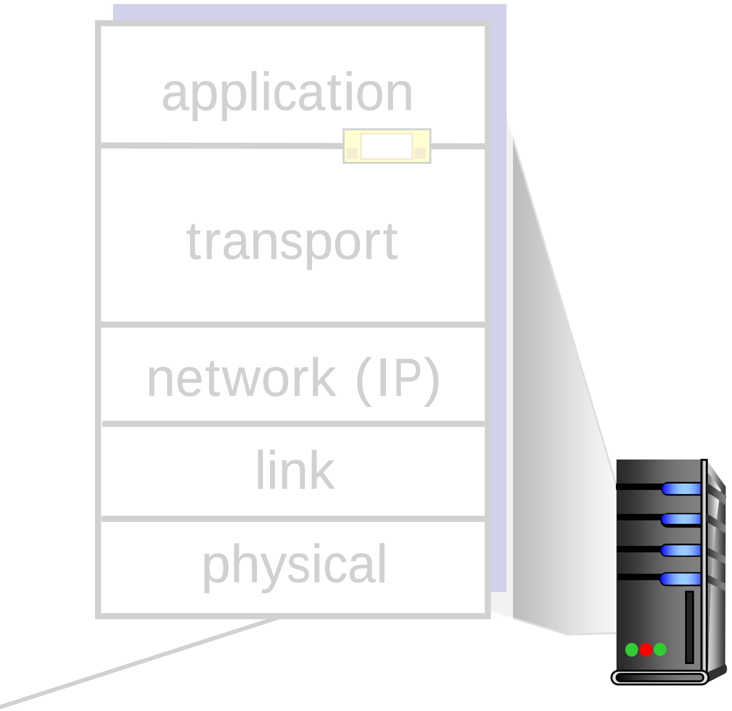


Transport Layer Actions



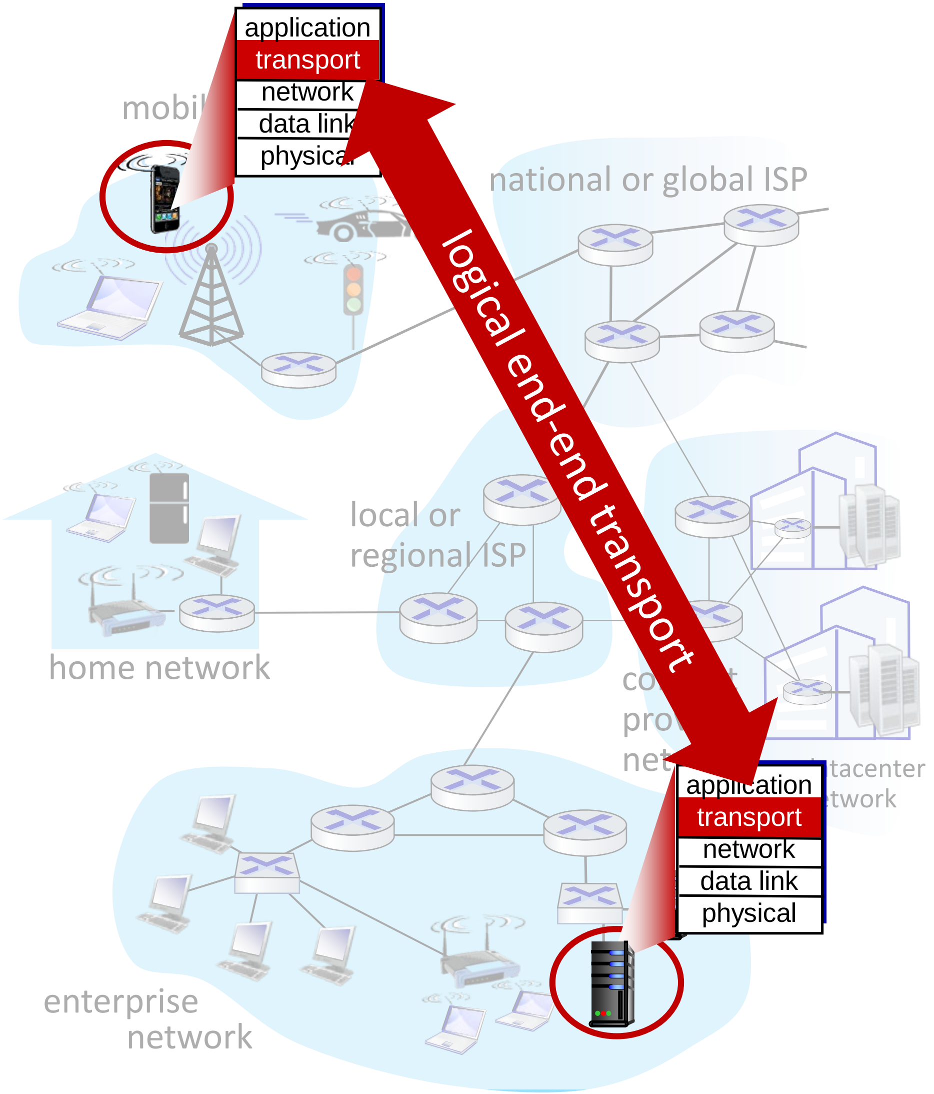
Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

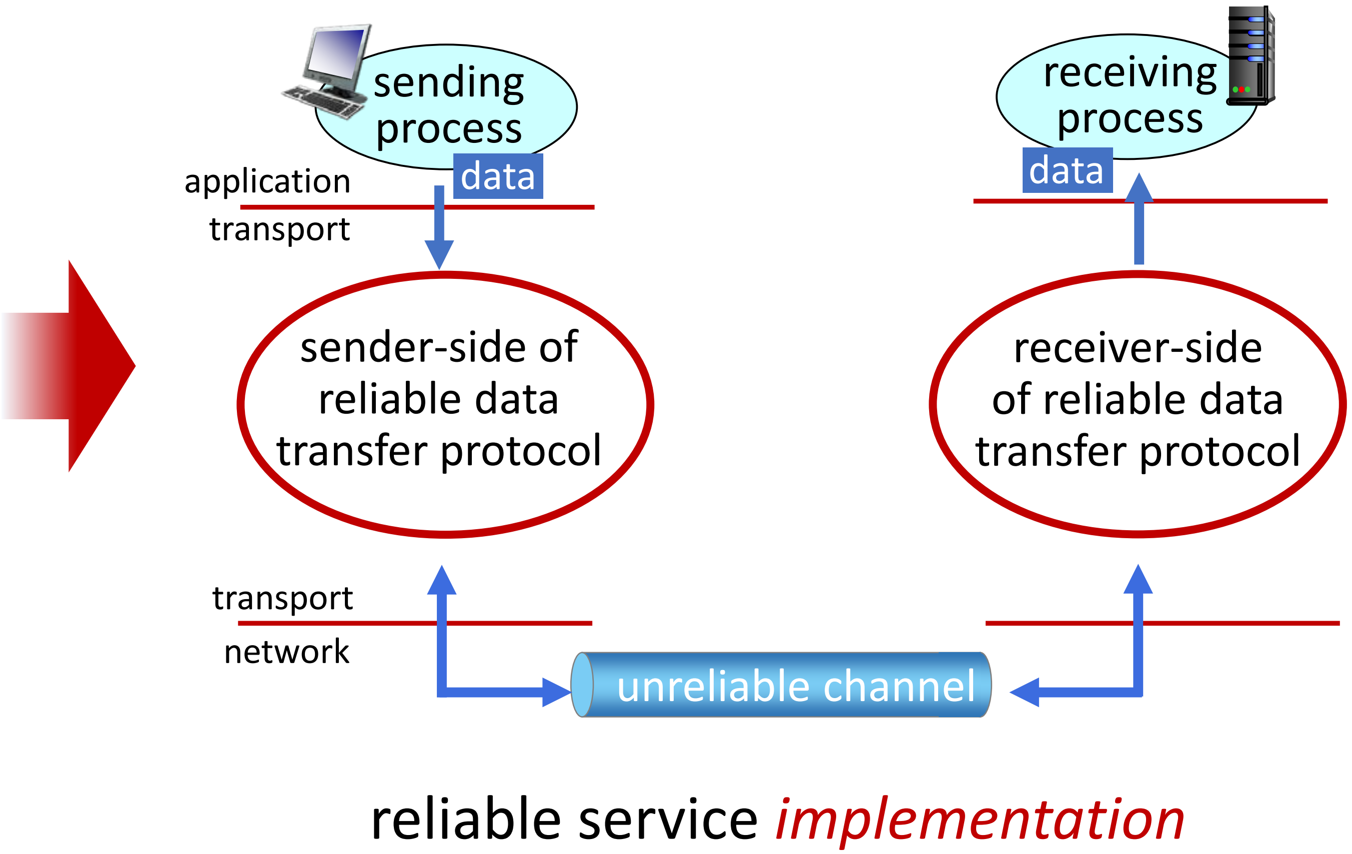


-
TCP: برای انتقال مطمئن و با ترتیب درست سگمنت های اطلاعات از مبدا به مقصد استفاده میشه برای این منظور پروتکل TCP از مکانیزم هایی استفاده میکنه از جمله اونها بحث connection setup است که برای مدیریت ارسال بسته ها استفاده میشه و flow control که برای مدیریت بافر سمت گیرنده استفاده میشه و congestion control که برای مدیریت ازدحام در شبکه استفاده میشه
UDP: ورژن ساده تری از این پروتکل است که صرفا انتقال بسته ها از مبدا تا مقصد رو مدیریت می کنه و اطمینان از رسیدن بسته ها و تاخیر درست اون ها رو تضمین نمیکنه
پروتکل UDP برای اپلیکیشن هایی استفاده میشه که چنین تضمینی رو لازم ندارند برخلاف پروتکل TCP که برای اپلیکیشن هایی استفاده میشه که رسیدن مطمئن و با ترتیب درست اطلاعات باید در اونجا تضمین بشه

Principles of reliable data transfer

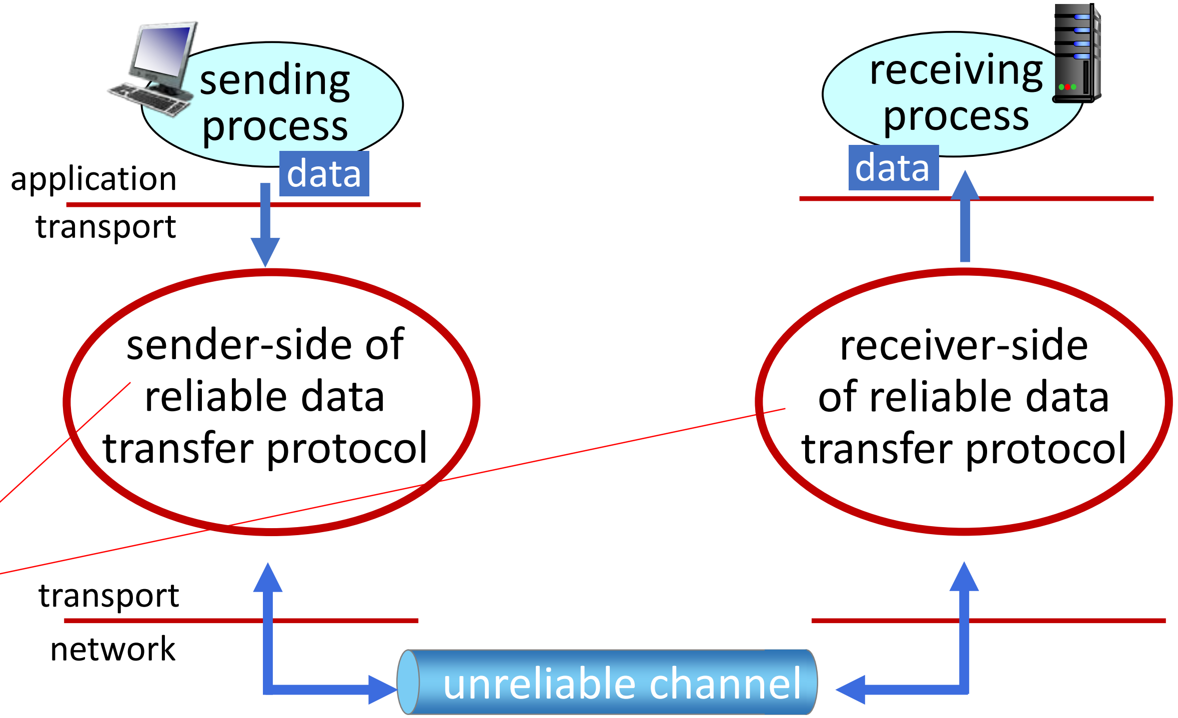
reliable service *abstraction*

Principles of reliable data transfer



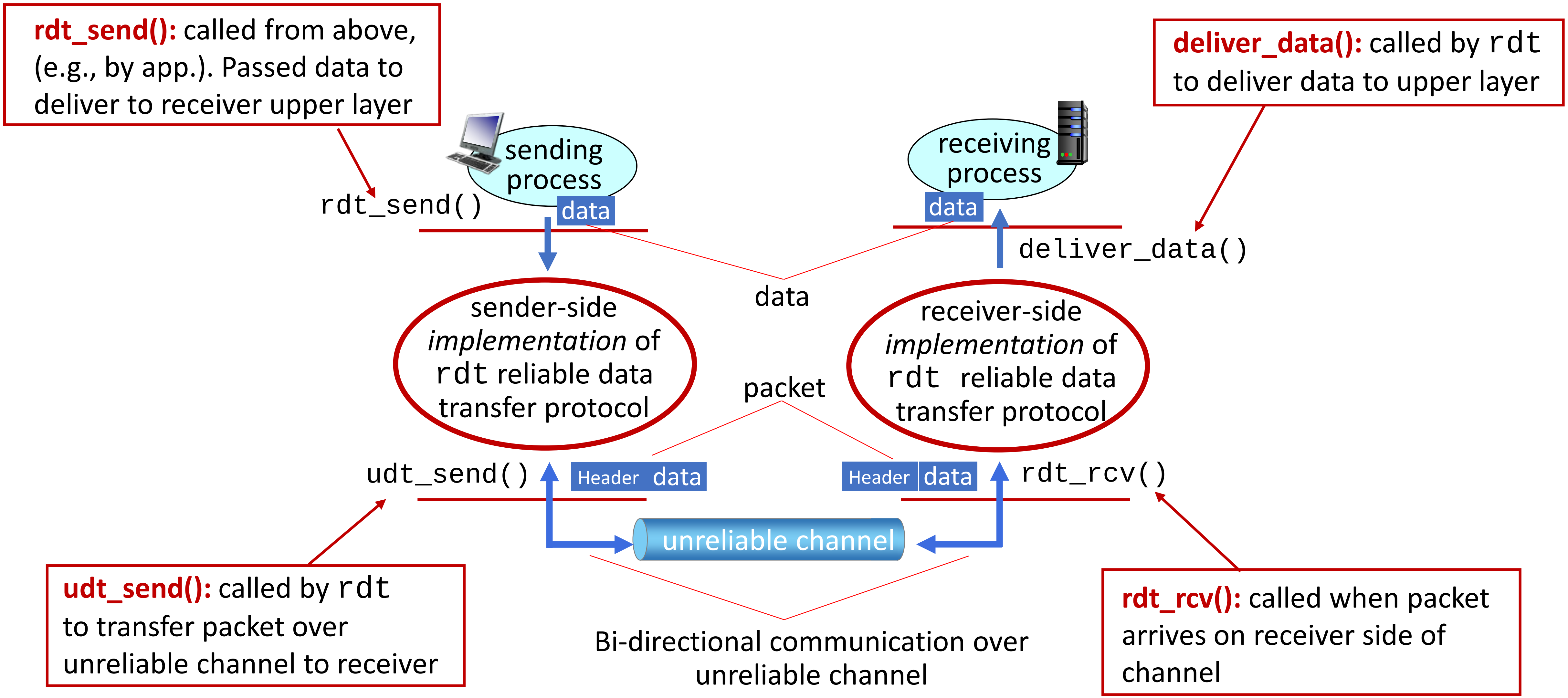
Principles of reliable data transfer

Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



reliable service *implementation*

Reliable data transfer protocol (rdt): interfaces

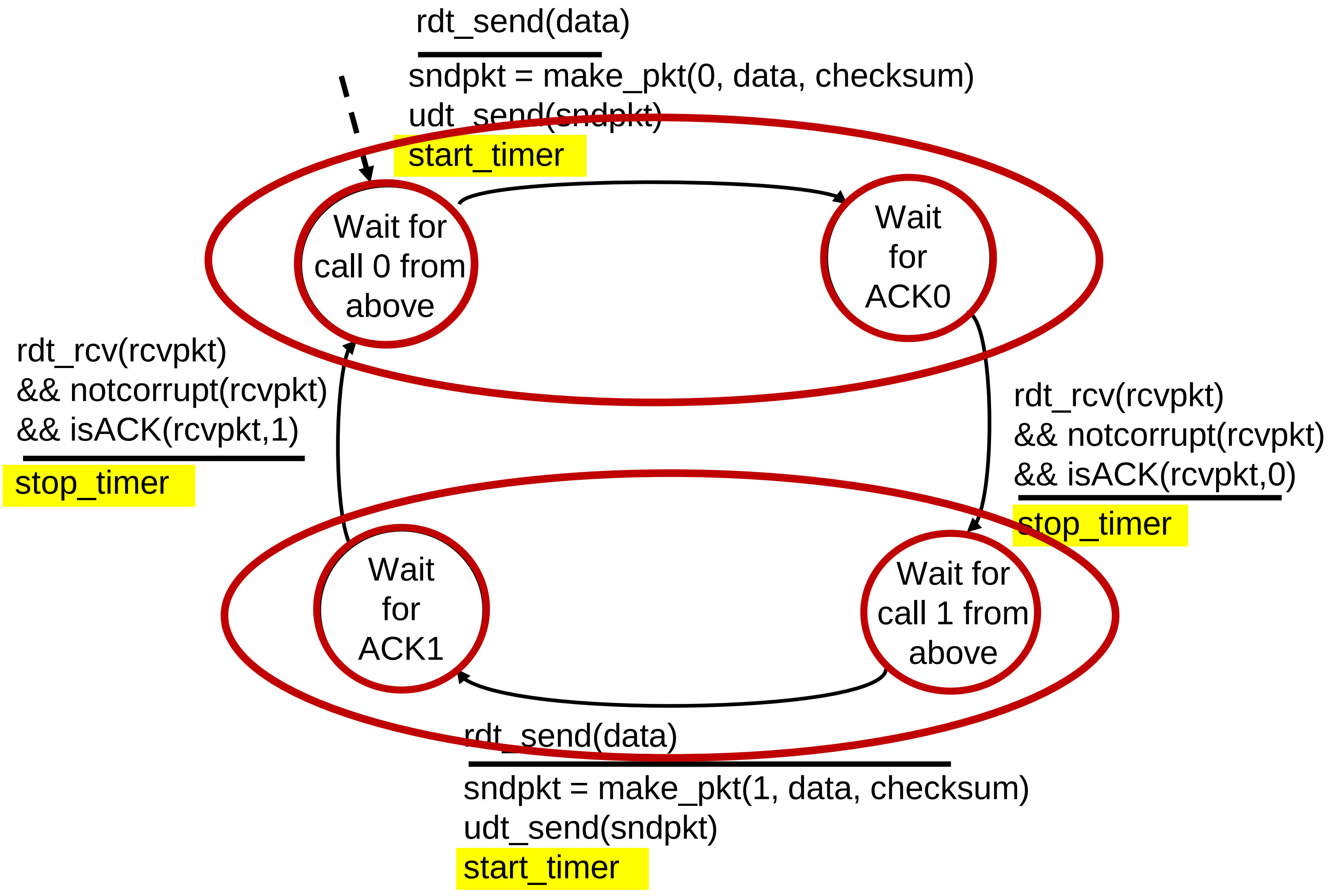


rdt3.0: channels with errors *and* loss

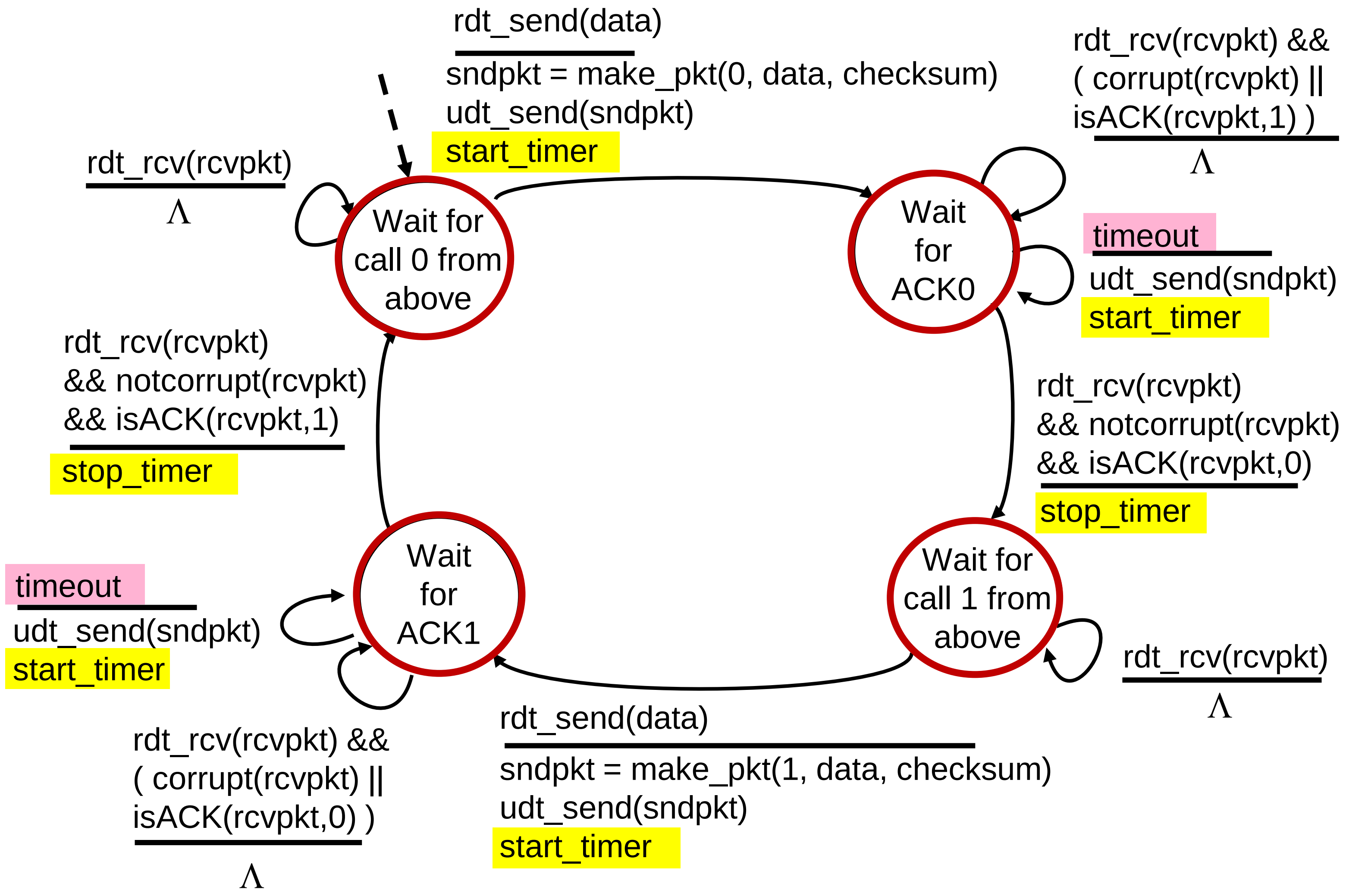
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after "reasonable" amount of time

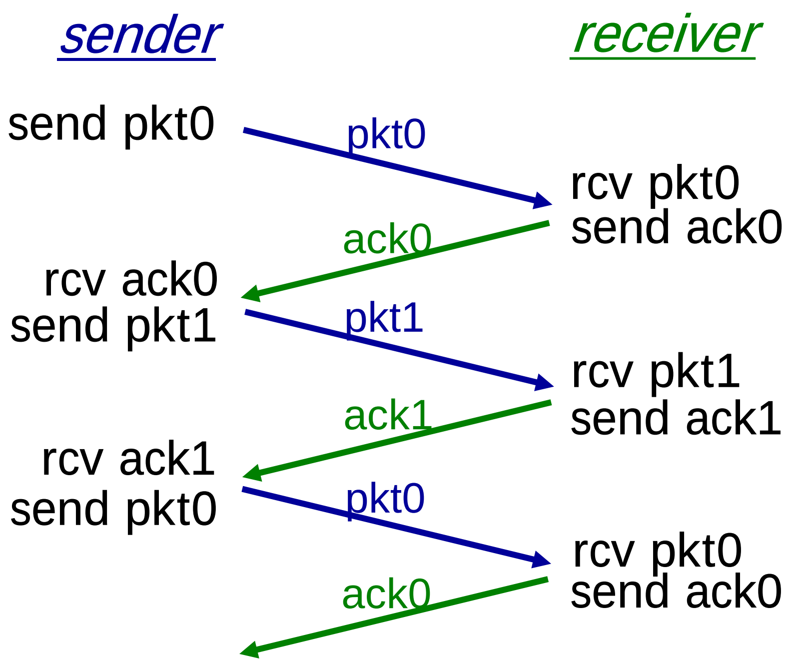
rdt3.0 sender



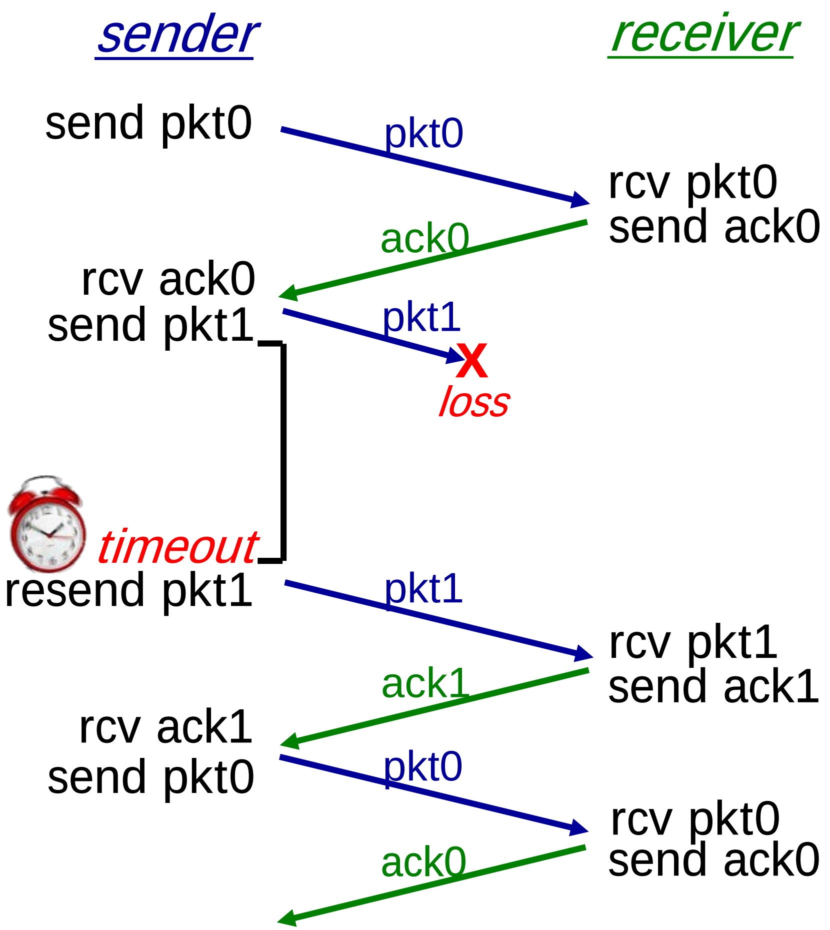
rdt3.0 sender



rdt3.0 in action

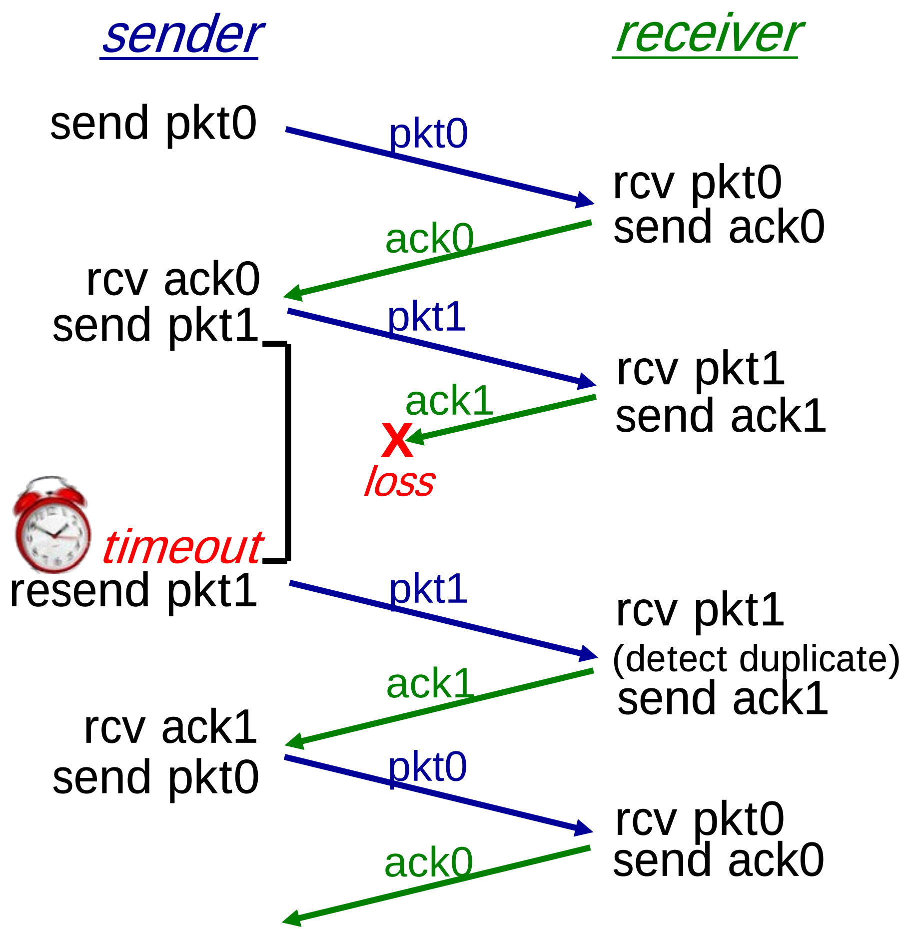


(a) no loss

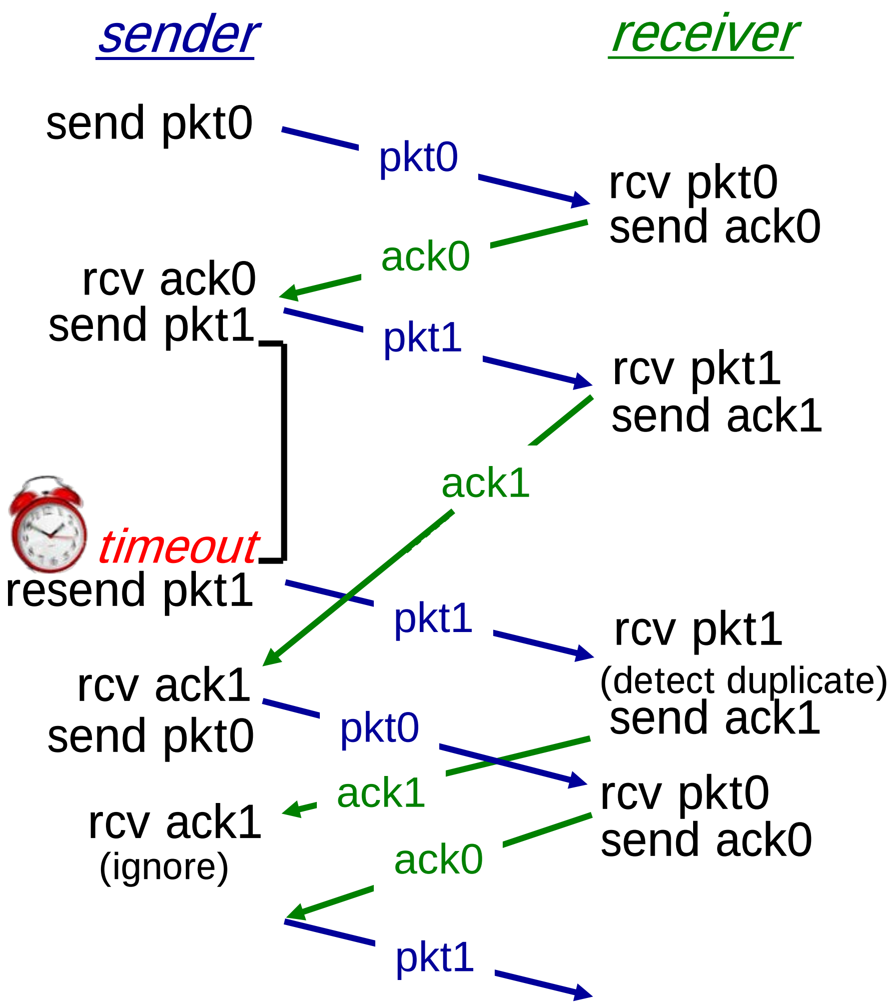


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

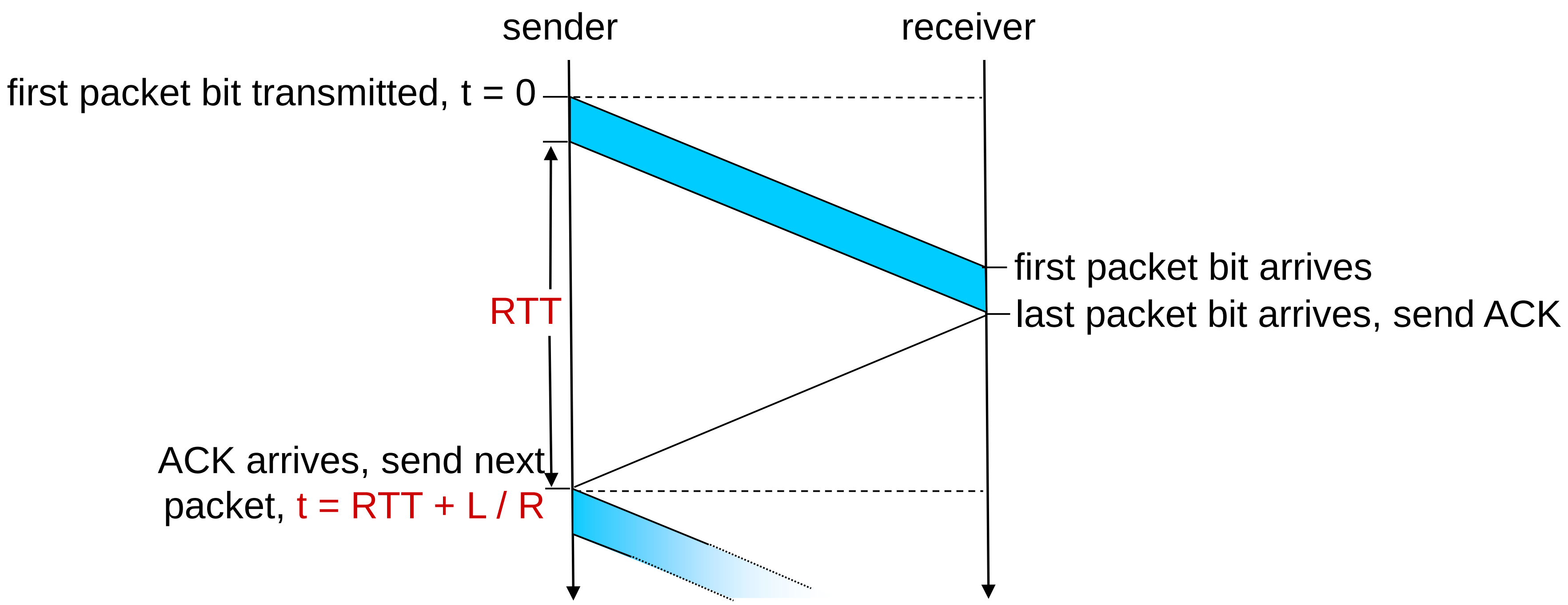
Performance of rdt3.0 (stop-and-wait)

- U_{sender}: *utilization* – fraction of time sender busy sending

- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - time to transmit packet into channel:

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

rdt3.0: stop-and-wait operation

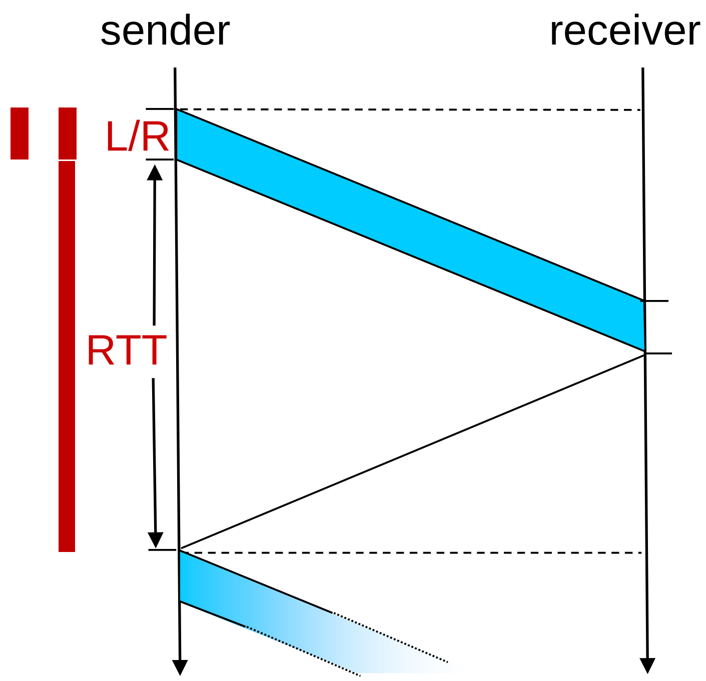


rdt3.0: stop-and-wait operation

$$U_{sender} = \frac{L / R}{RTT + L / R}$$

$$= \frac{.008}{30.008}$$

$$= 0.00027$$

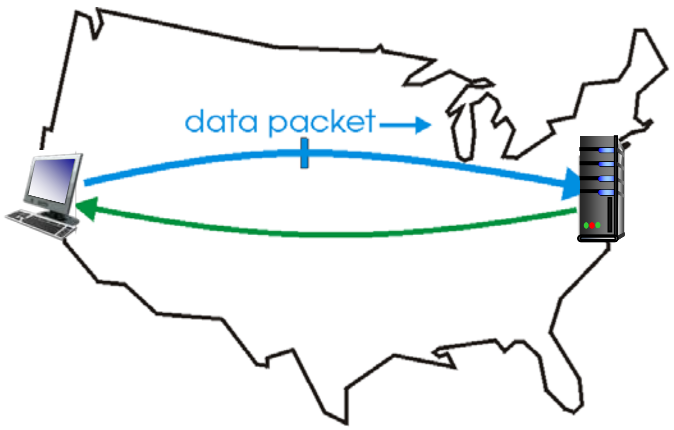


- rdt 3.0 protocol performance stinks!
- Protocol limits performance of underlying infrastructure (channel)

rdt3.0: pipelined protocols operation

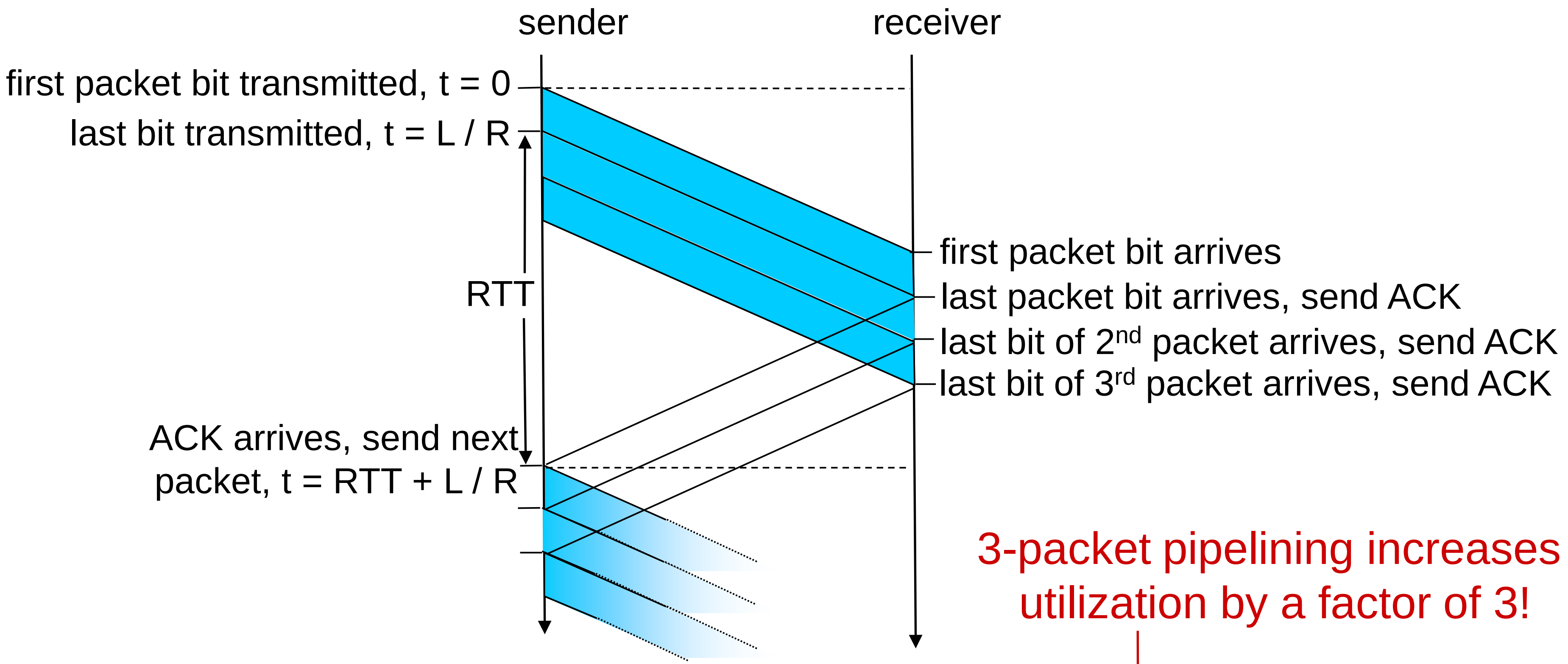
pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets

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



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

Pipelining: increased utilization

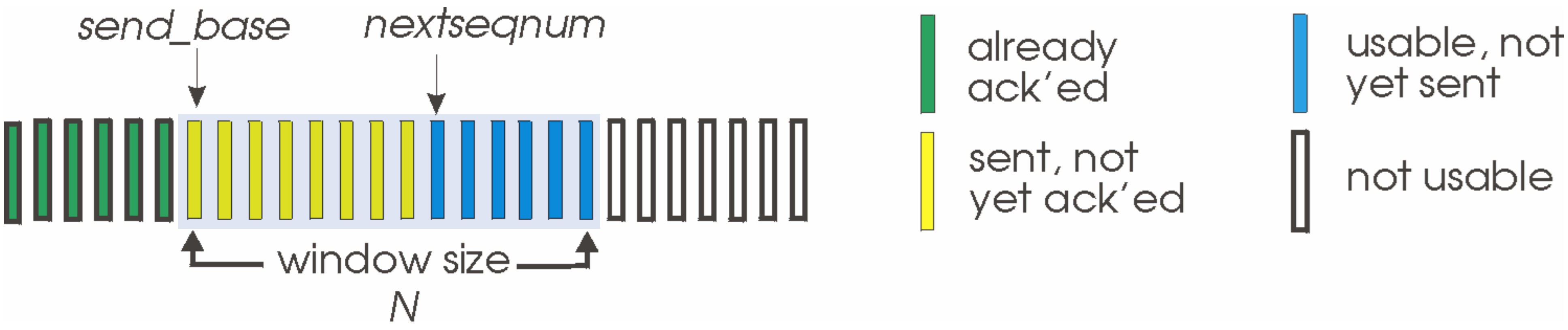


3-packet pipelining increases utilization by a factor of 3!

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

Go-Back-N: sender

- sender: “window” of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header

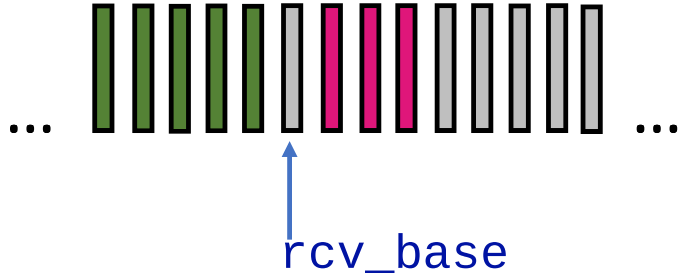


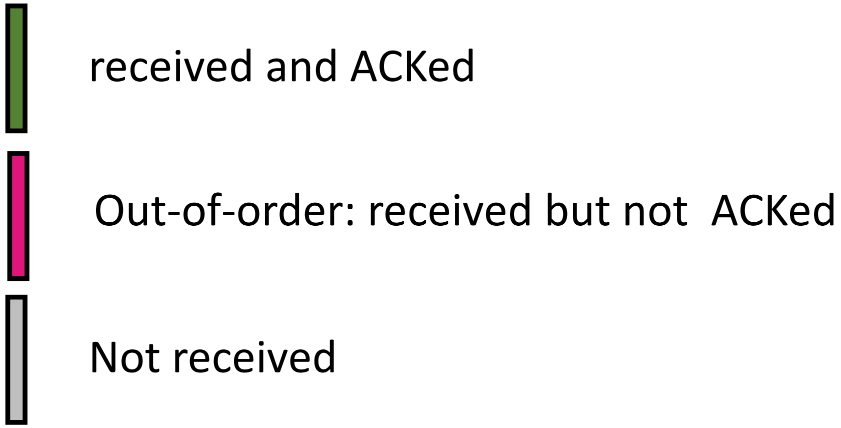
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver

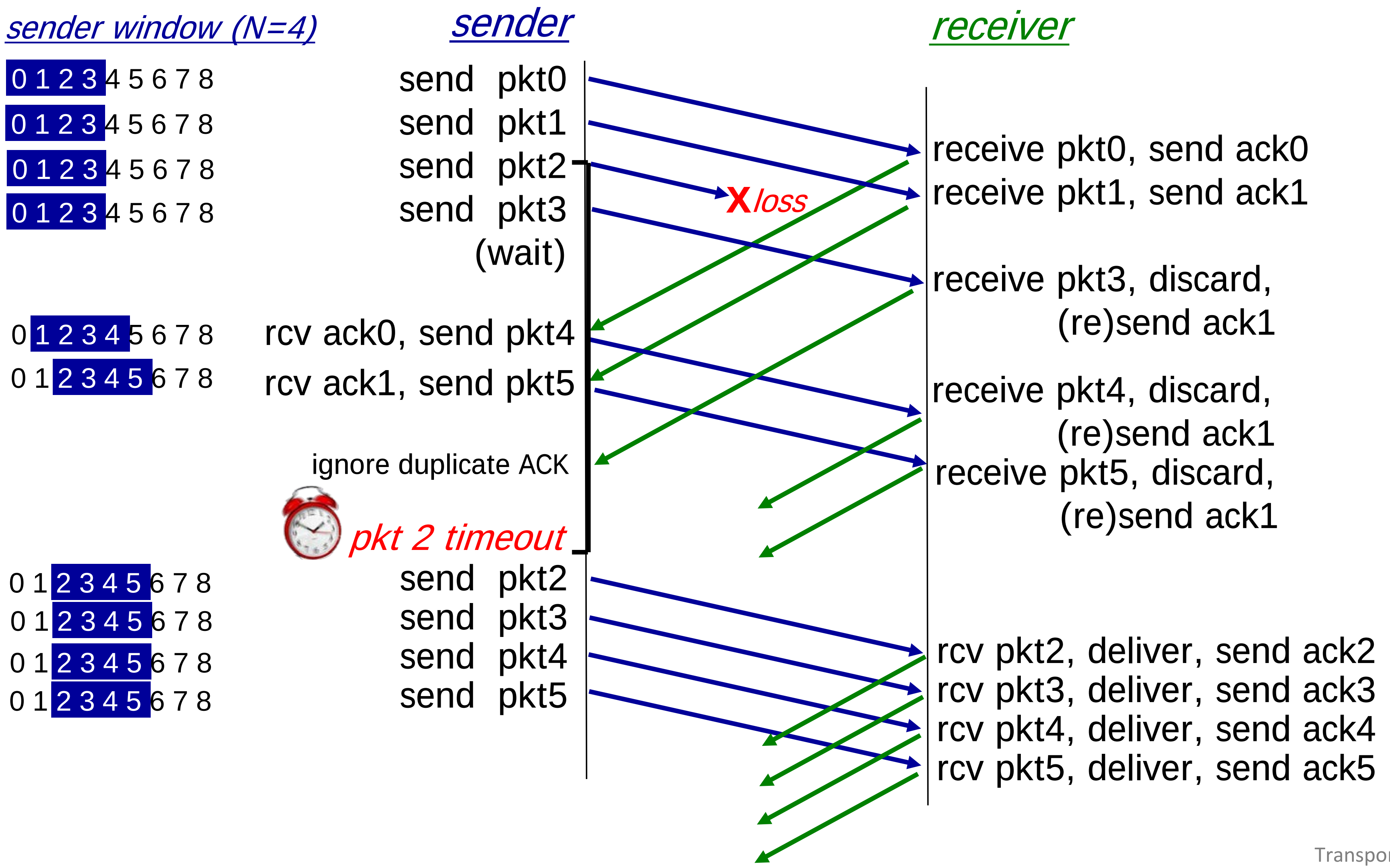
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don’t buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:





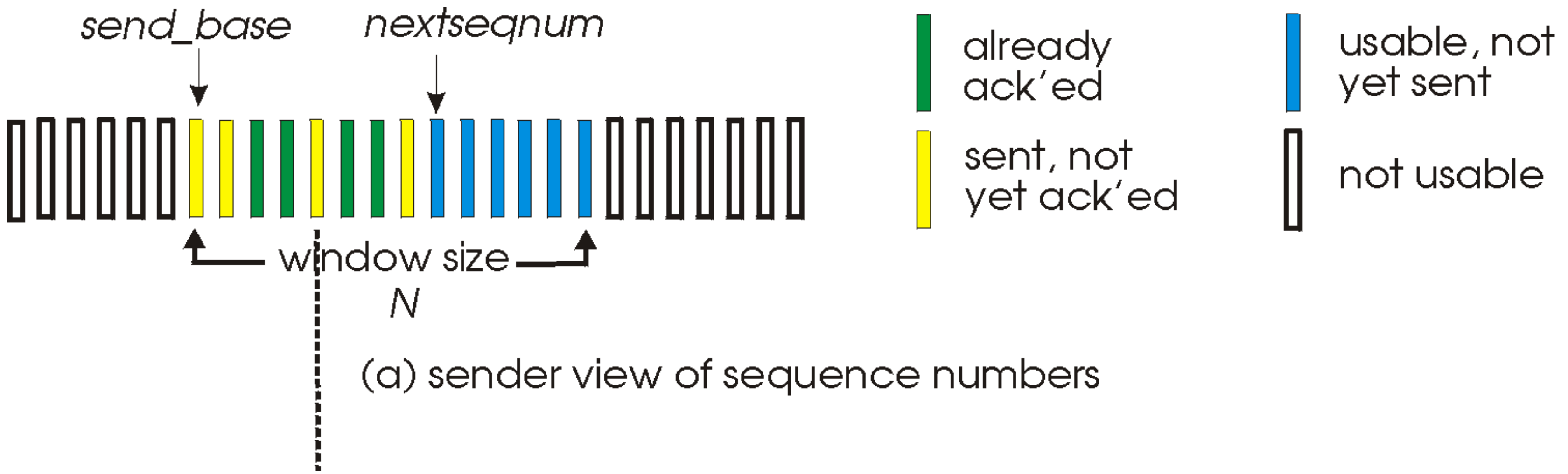
Go-Back-N in action



Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

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

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

receiver

packet *n* in [rcvbase, rcvbase+N-1]

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

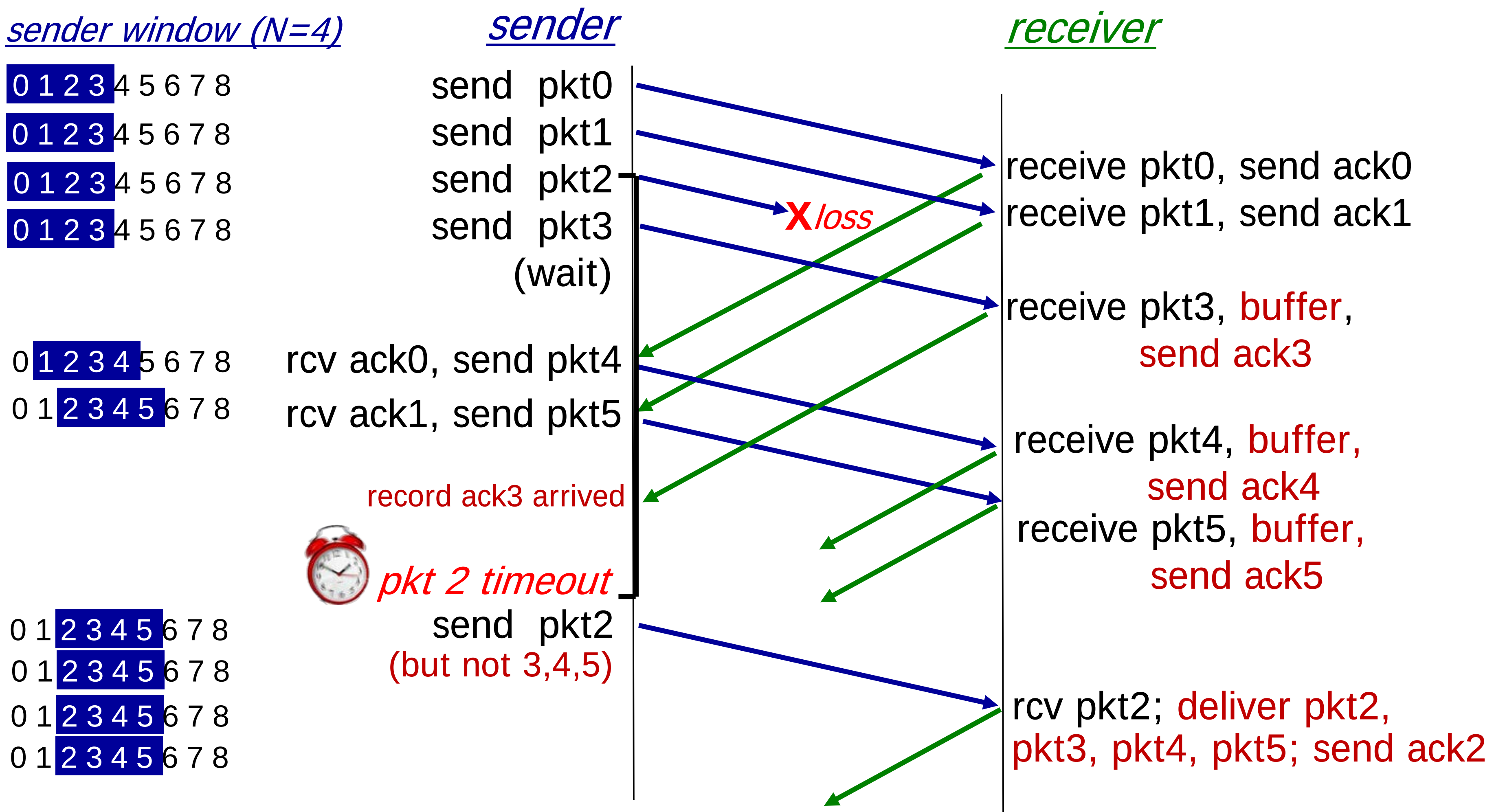
packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

otherwise:

- ignore

Selective Repeat in action



Q: what happens when ack2 arrives?