Information Network

Lecture 6 : Transport Layer Introduction

Holger Thies

KYOTO UNIVERSITY

京都大学

Today's lecture

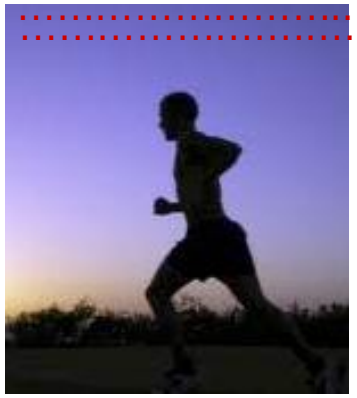
1. video streaming and content distribution networks (CDNs)
2. Summary of the application layer
3. Introduction to the transport layer

京都大学

Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease number of bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



frame *i*

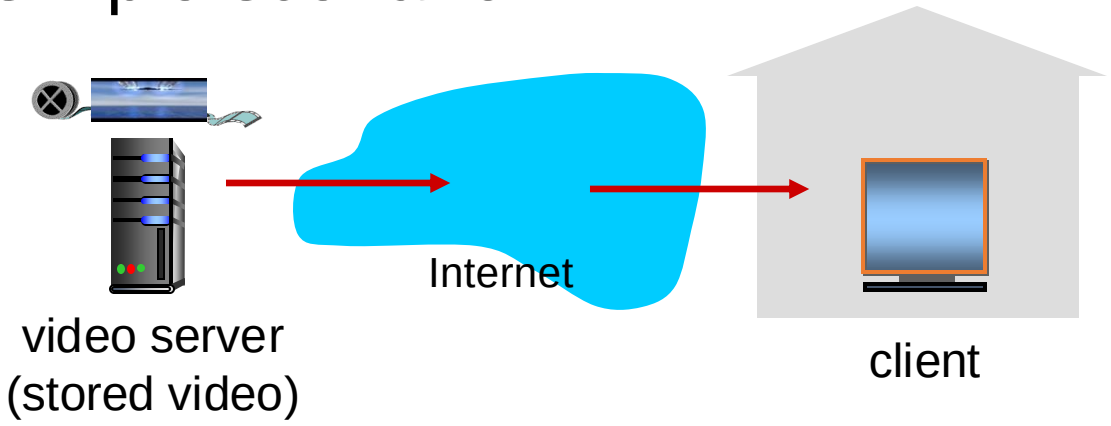
frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

京都大学

Streaming stored video

simple scenario:

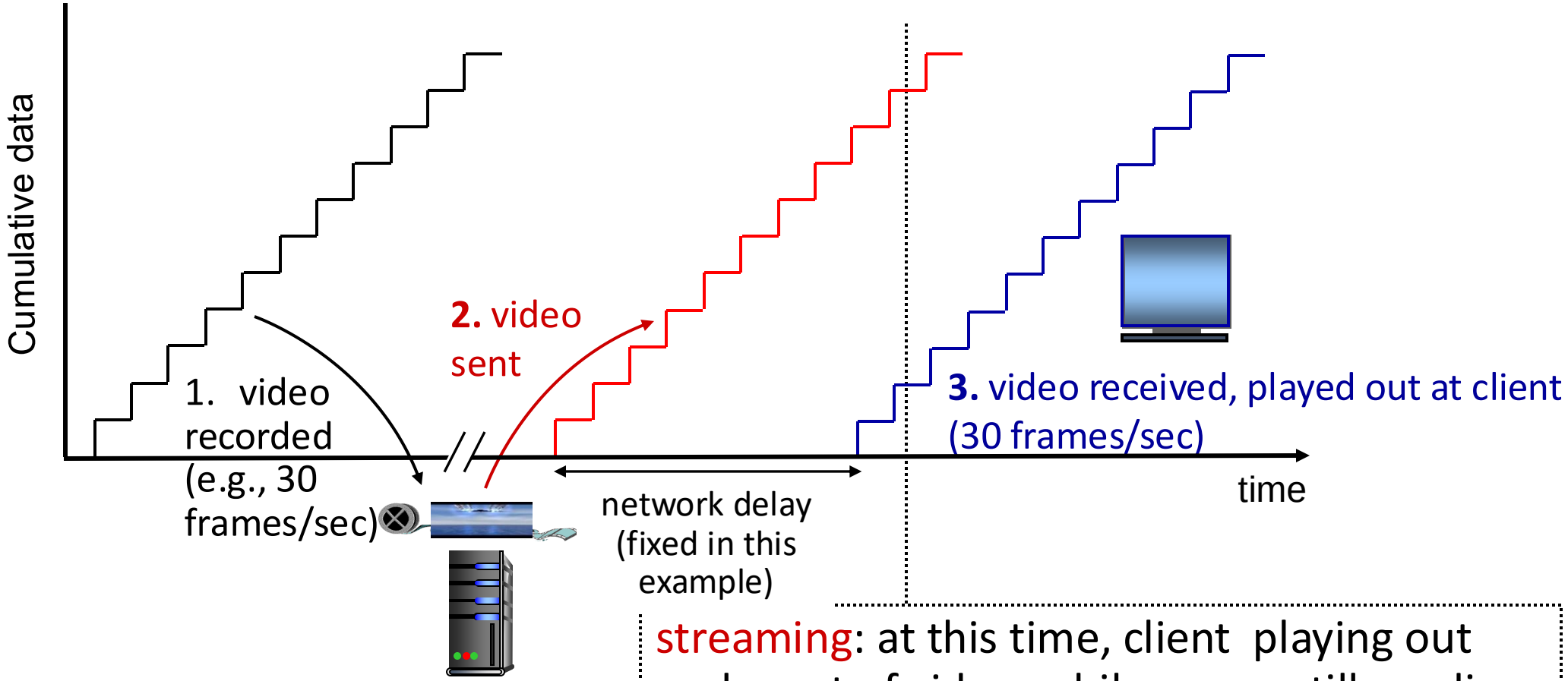


Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels
- packet loss, delay due to congestion will delay playout, or result in poor video quality

京都大学

Streaming stored video

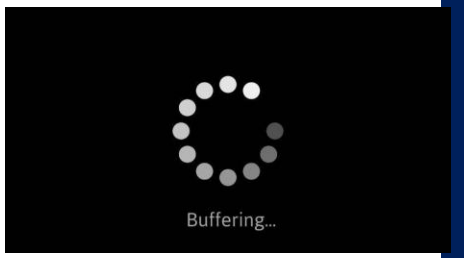


streaming: at this time, client playing out early part of video, while server still sending later part of video

京都大学

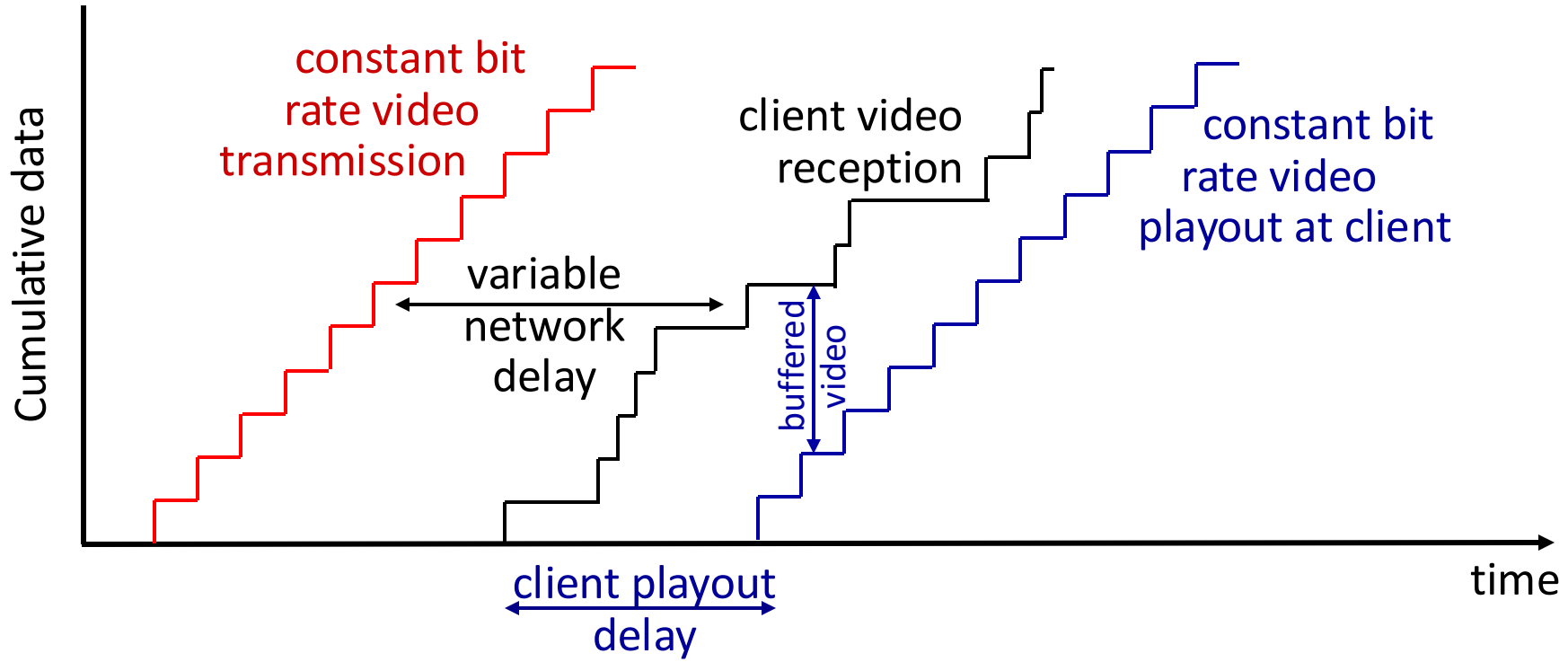
Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable, so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted (additional delay)



京都大学

Streaming stored video: playout buffering



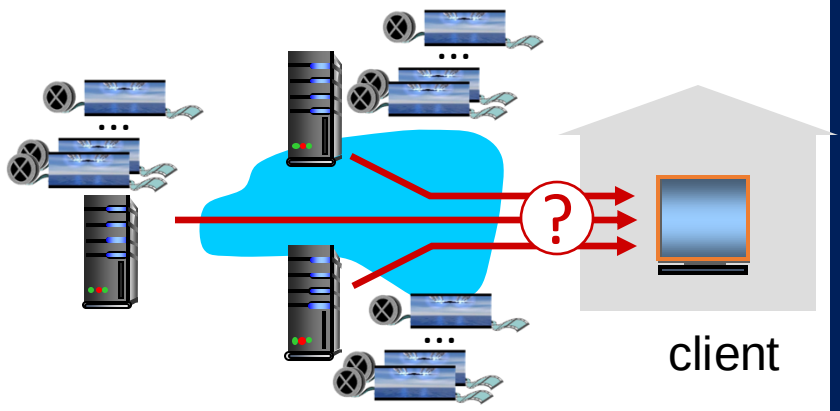
- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various locations
- *manifest file:* provides URLs for different chunks



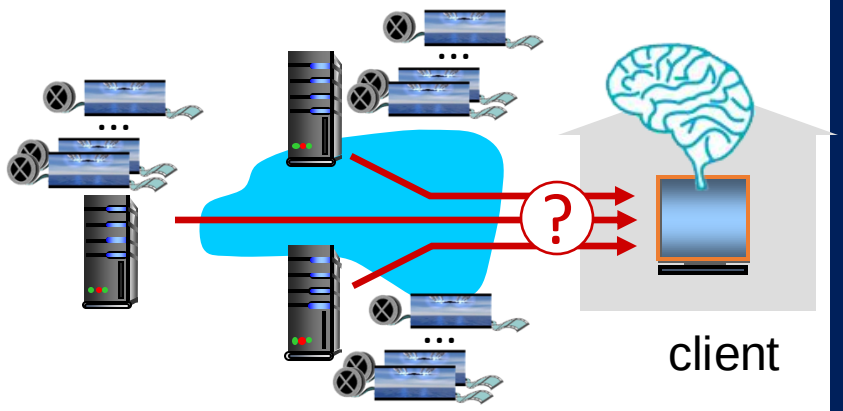
client:

- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

京都大学

Streaming multimedia: DASH

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)



Streaming video = encoding + DASH + playout buffering

京都大学

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long path to distant clients

This solution *doesn't scale*

京都大学

Content distribution networks

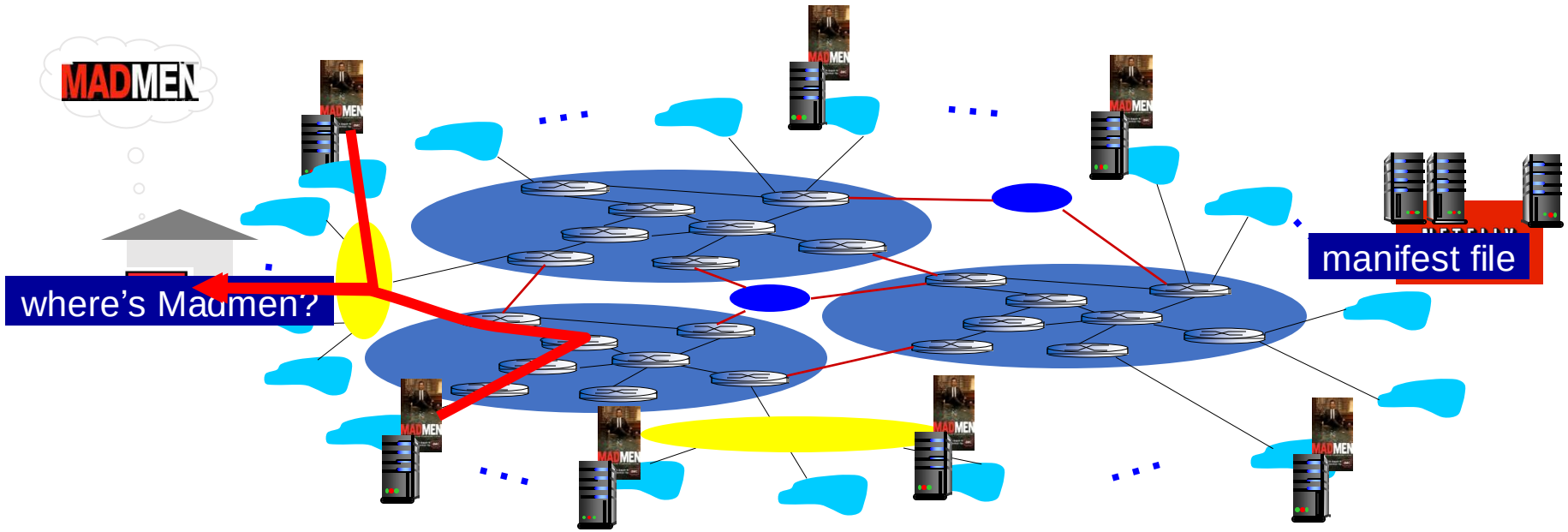
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*

京都大学

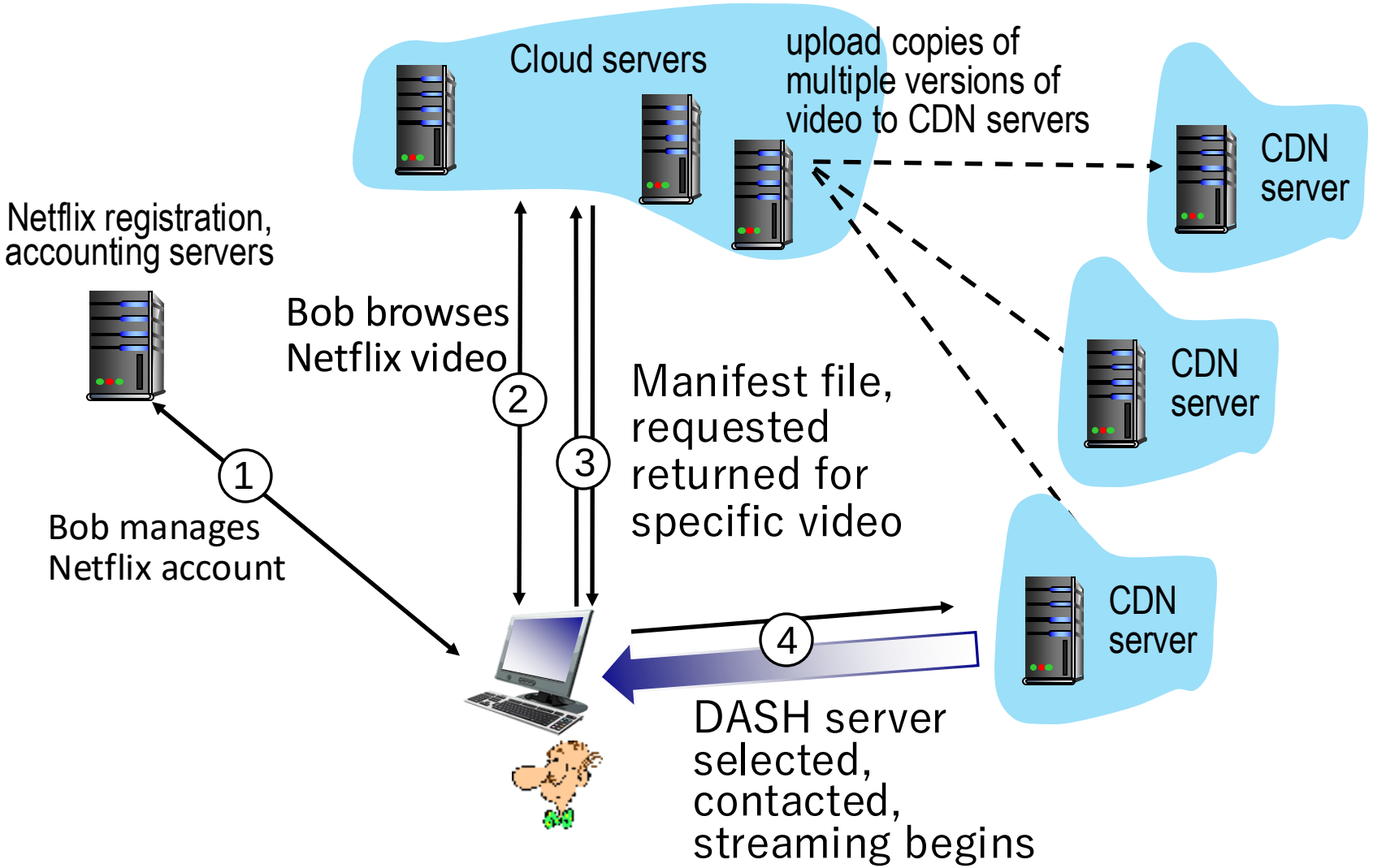
Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



京都大学

Case study: Netflix



京都大学

Application Layer Summary

our study of network application layer is now complete!

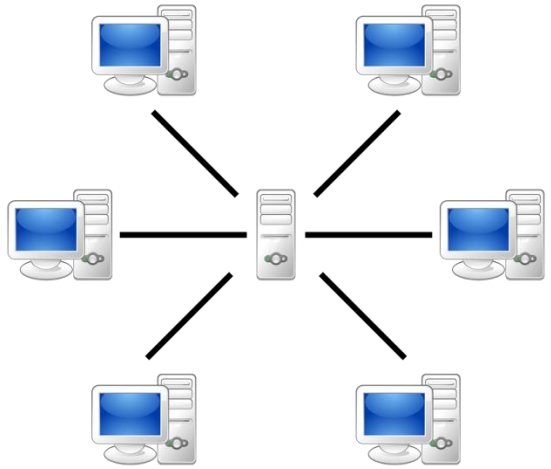
- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

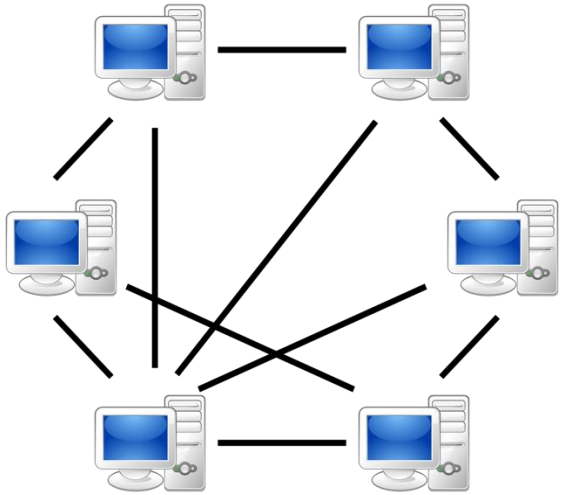
京都大学

Application architectures

Two main architectures for the application layer have been developed:



Client-server architecture



Peer-to-Peer (P2P) architecture

京都大学

Application Layer Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

京都大学

What transport service does an app need?

reliable data transfer

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

京都大学

Application Layer Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

京都大学

Internet transport protocols services

TCP (Transmission Control Protocol):

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

UDP (User Datagram Protocol):

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

京都大学

Application Layer Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

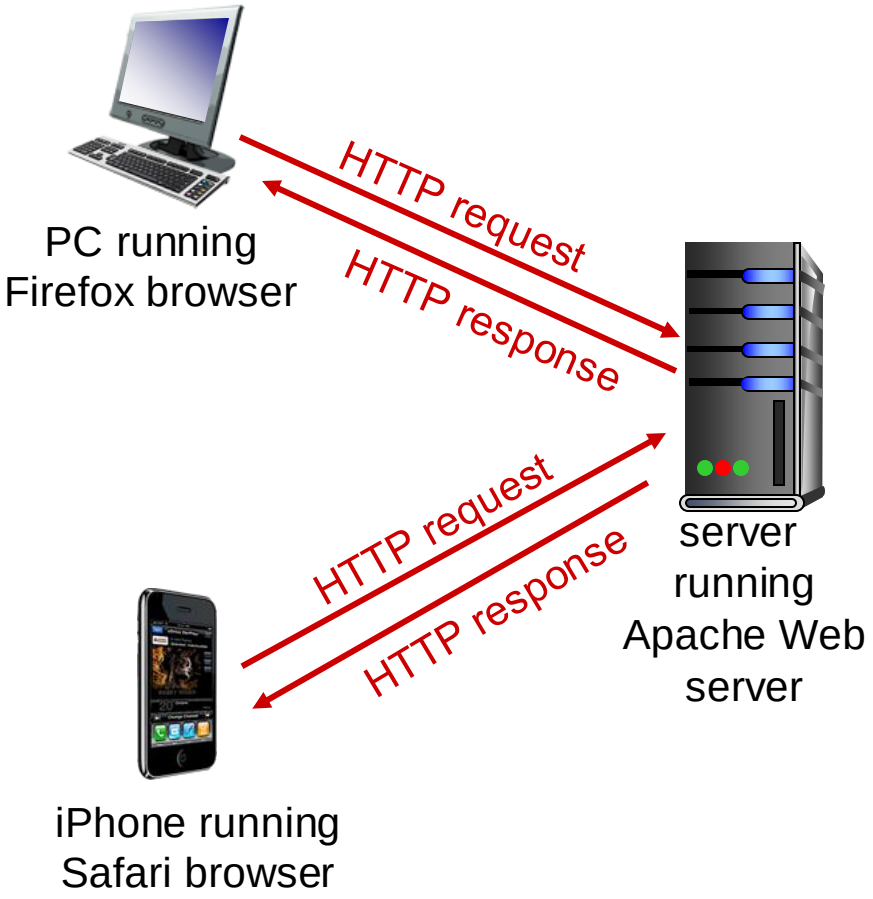
- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

京都大学

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



京都大学

Application Layer Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

京都大学

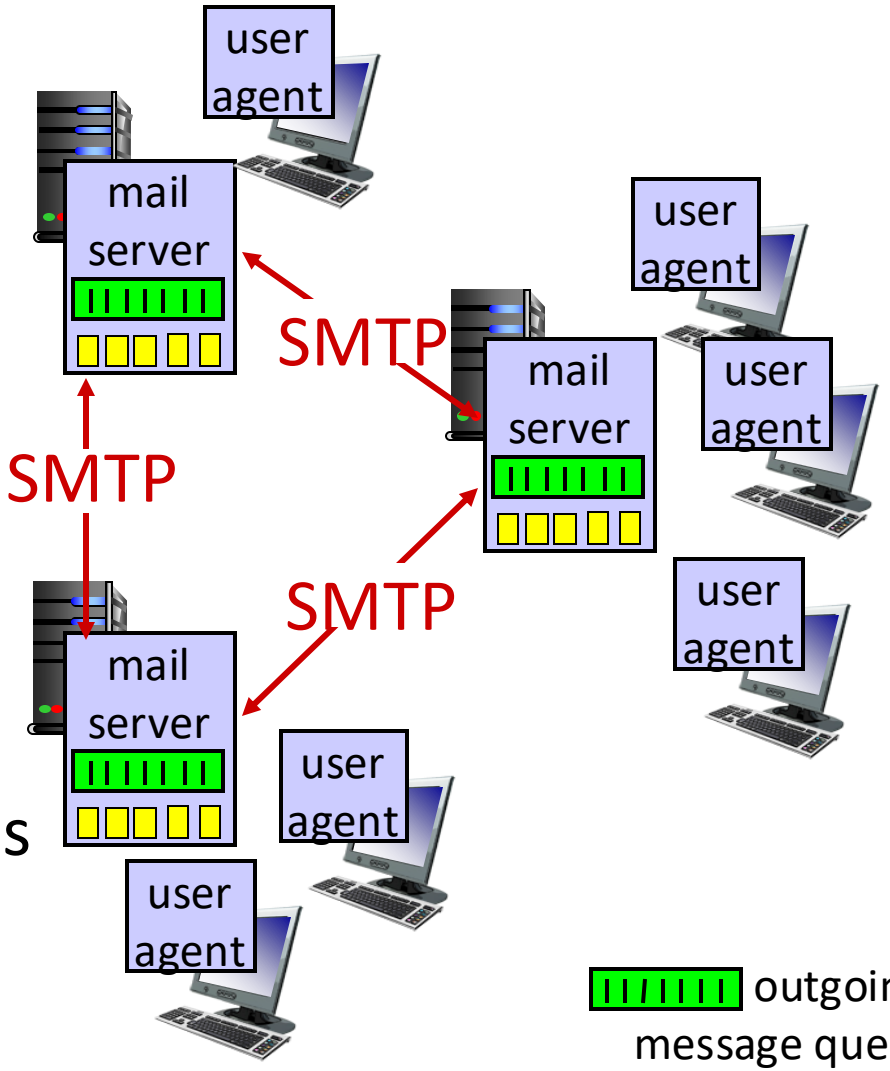
E-mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



京都大学

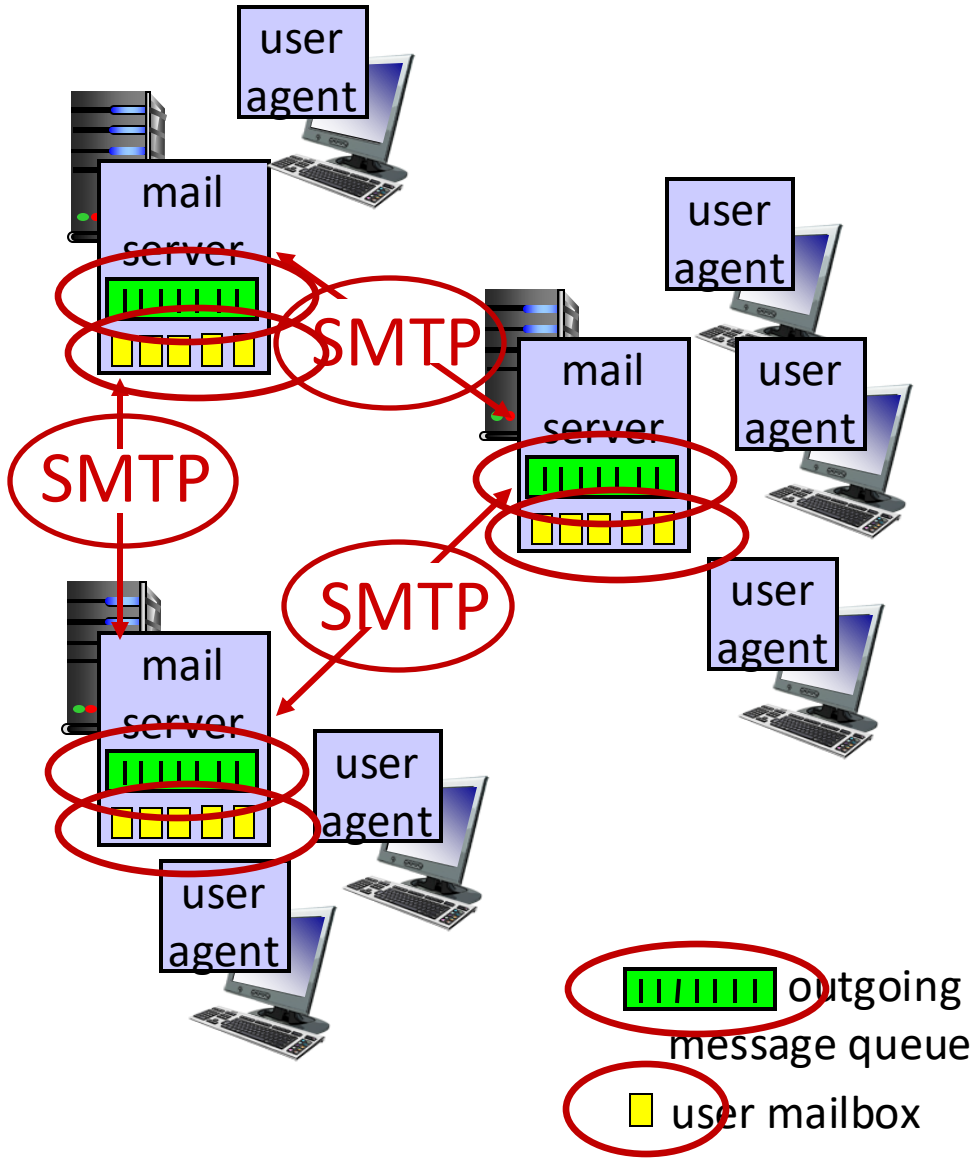
E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages

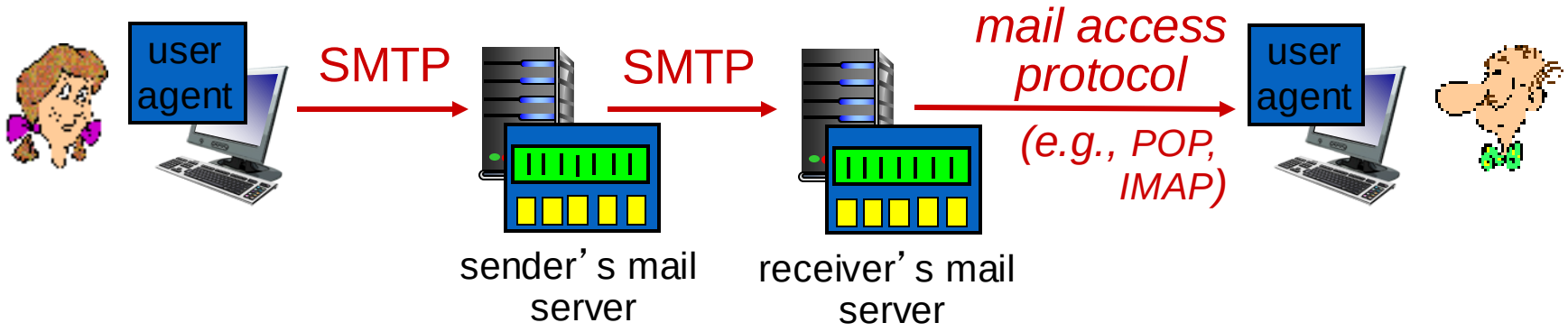
SMTP protocol between mail servers to send email messages

- client: sending mail server
- "server": receiving mail server



京都大学

Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol: authorization, download
  - IMAP: Internet Mail Access Protocol: more features, including manipulation of stored messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

京都大学

Application Layer Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - SMTP, POP3, IMAP
  - DNS

京都大学

DNS: domain name system

people: many identifiers:

- Name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing by routers
- IP address looks like 127.7.106.83
- Hierarchical: Going from left to right gives more and more specific information about the location
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

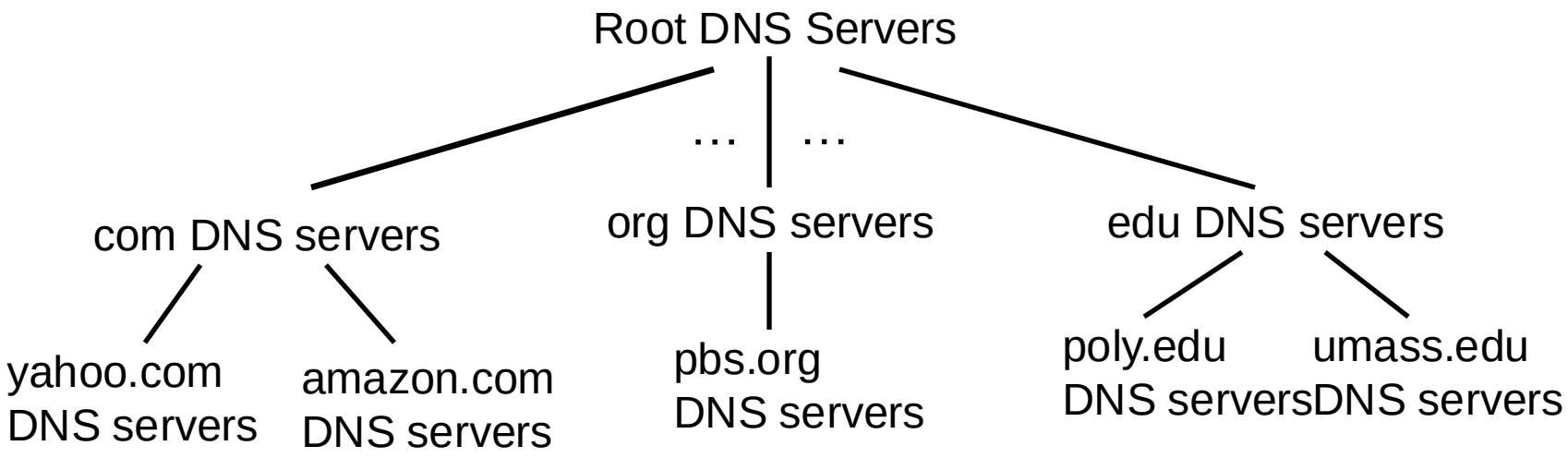
Domain Name System:

Service that translates hostnames to IP addresses

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)

京都大学

DNS: a distributed, hierarchical database



client wants IP for www.amazon.com; 1st approximation:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

京都大学

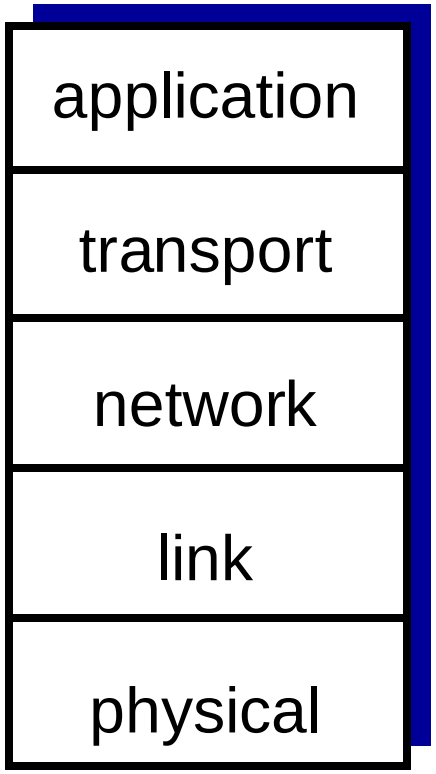
Today's lecture

1. video streaming and content distribution networks (CDNs)
2. Summary of the application layer
3. Introduction to the transport layer

京都大学

Internet protocol stack

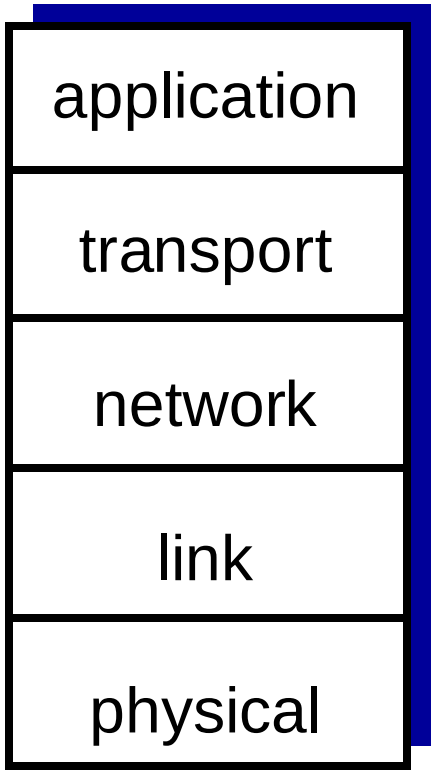
- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi),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



京都大学

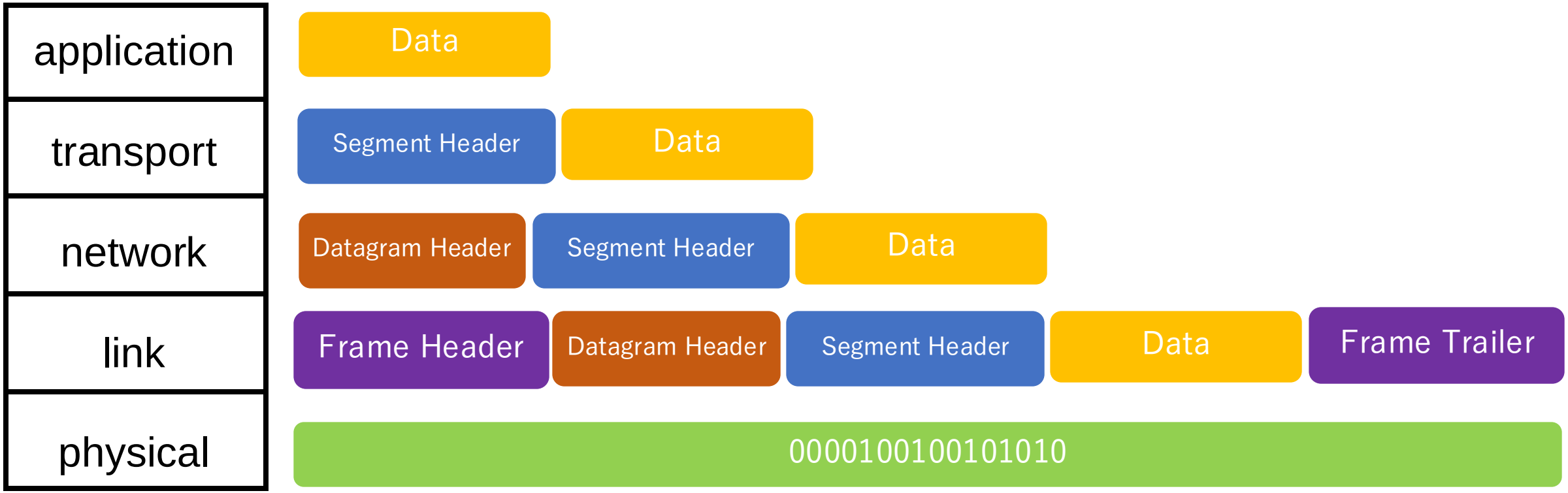
Packets, Segments, Datagrams, etc.

- The application layer gives data (of any size) to the underlying layers.
- To send the data it needs to be broken into smaller chunks of data (packets)
- Each layer adds additional header information
- We therefore use different terms for packets at different layers:
  - **Packet** is the general term for a chunk of data transmitted through the network
  - **Segment** is used for the data together with the header data from the transport layer
  - **Datagram** is used for the segment with additional information from the network layer.
  - **Frame** is used on the link layer
  - **Bits** are transferred on the physical layer



京都大学

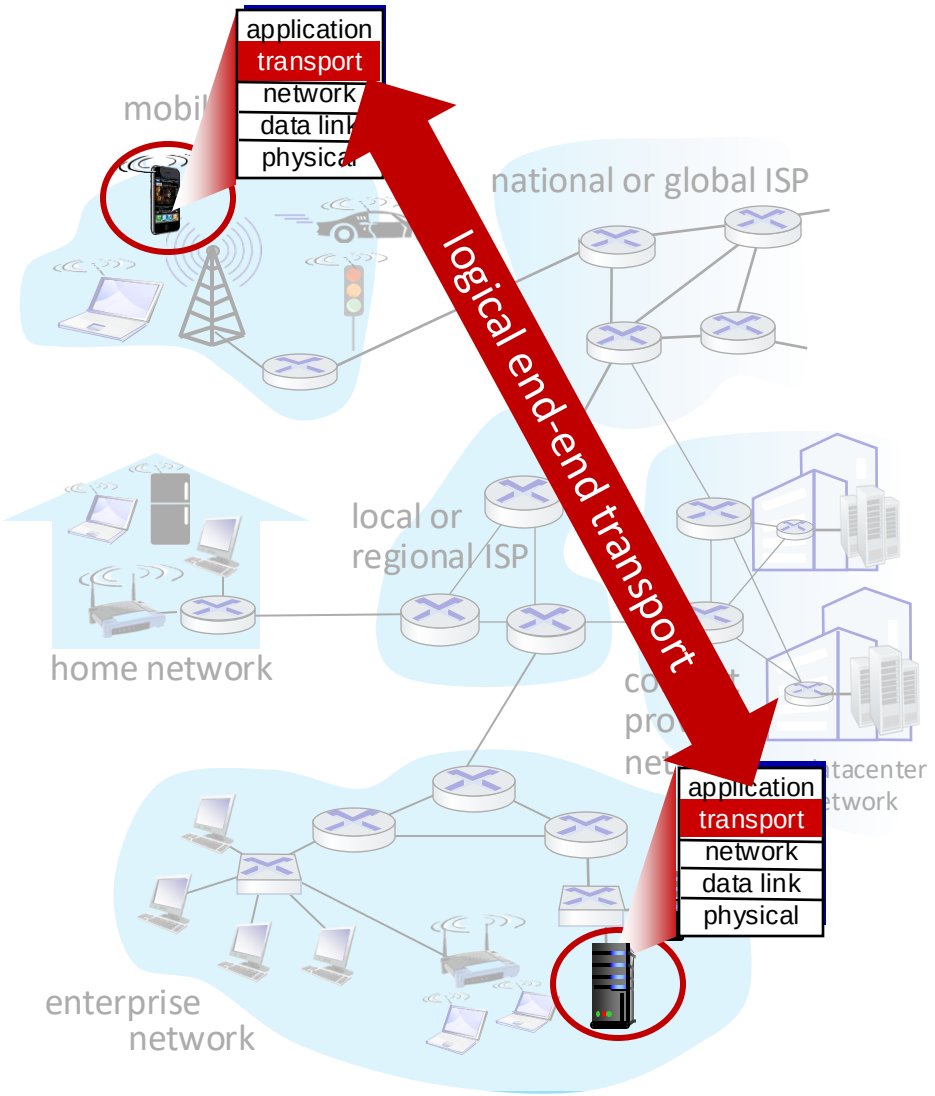
Packets, Segments, Datagrams, etc.



京都大学

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
  - extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



京都大学

Transport Layer

our goals:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

京都大学

Overview

3.1 transport-layer services

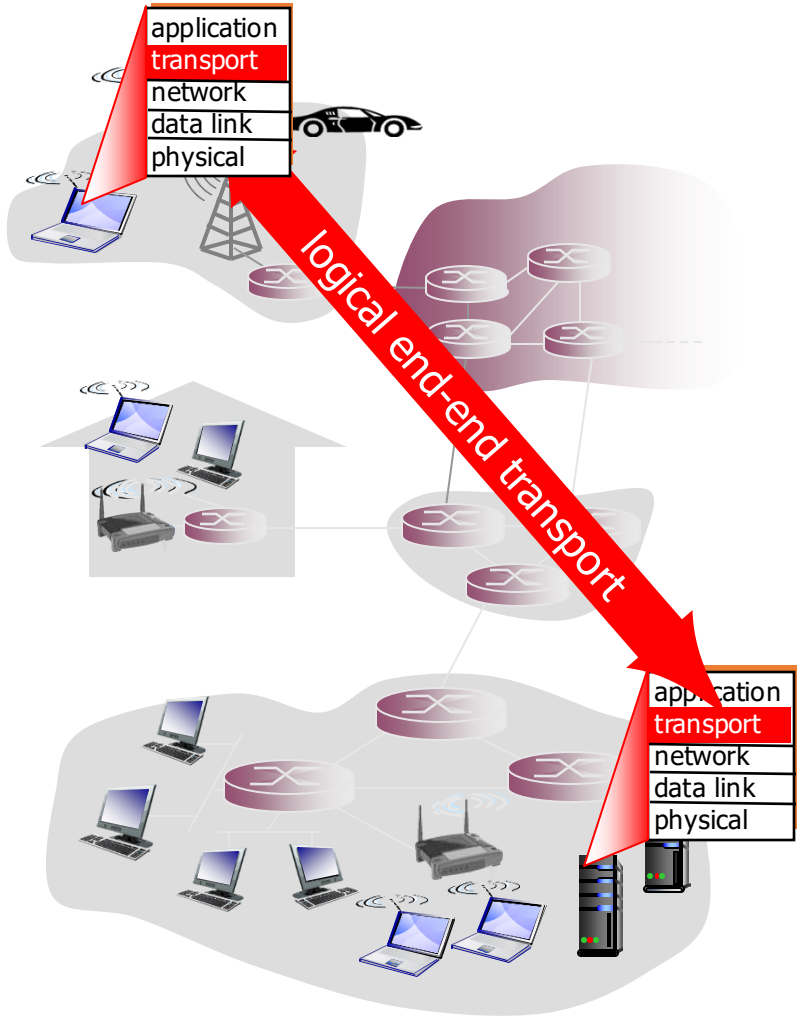
3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

京都大学

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - receive side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



京都大学

Transport vs. network layer

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

household analogy:

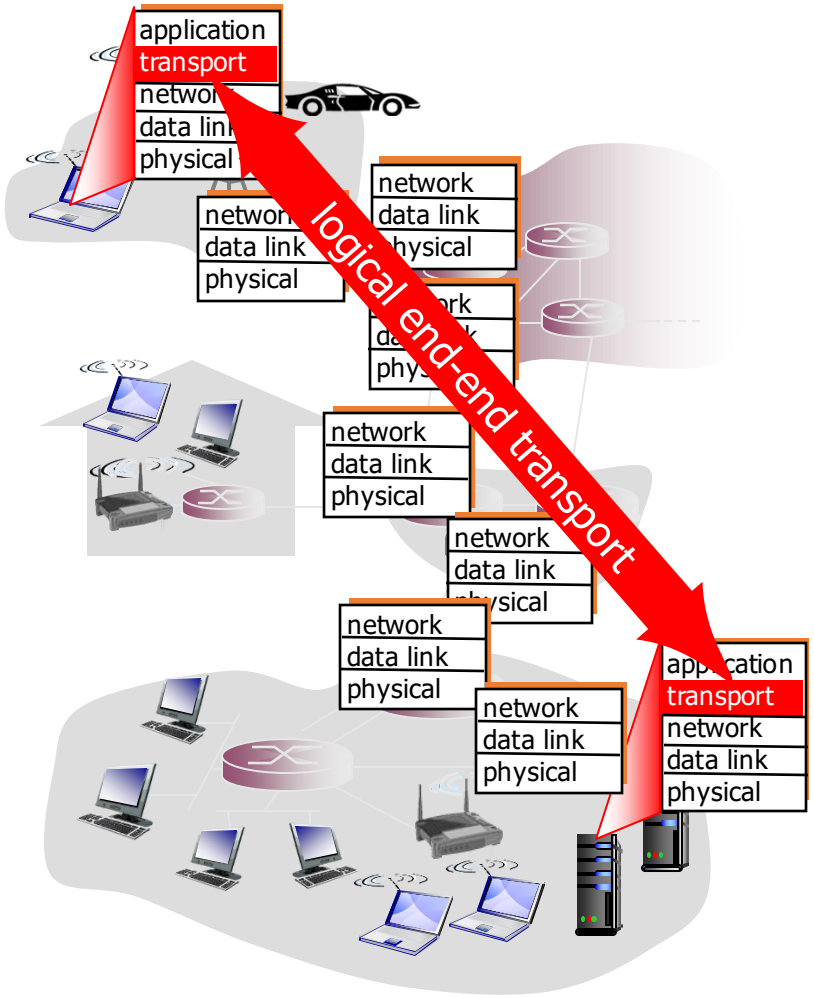
12 kids in house A sending letters to 12 kids in house B:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Kid in house A and kid in house B responsible for collecting and distributing letters
- network-layer protocol = postal service

京都大学

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
- services not available in both:
  - delay guarantees
  - bandwidth guarantees
  - These services are difficult to realize in packet switched networks



京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

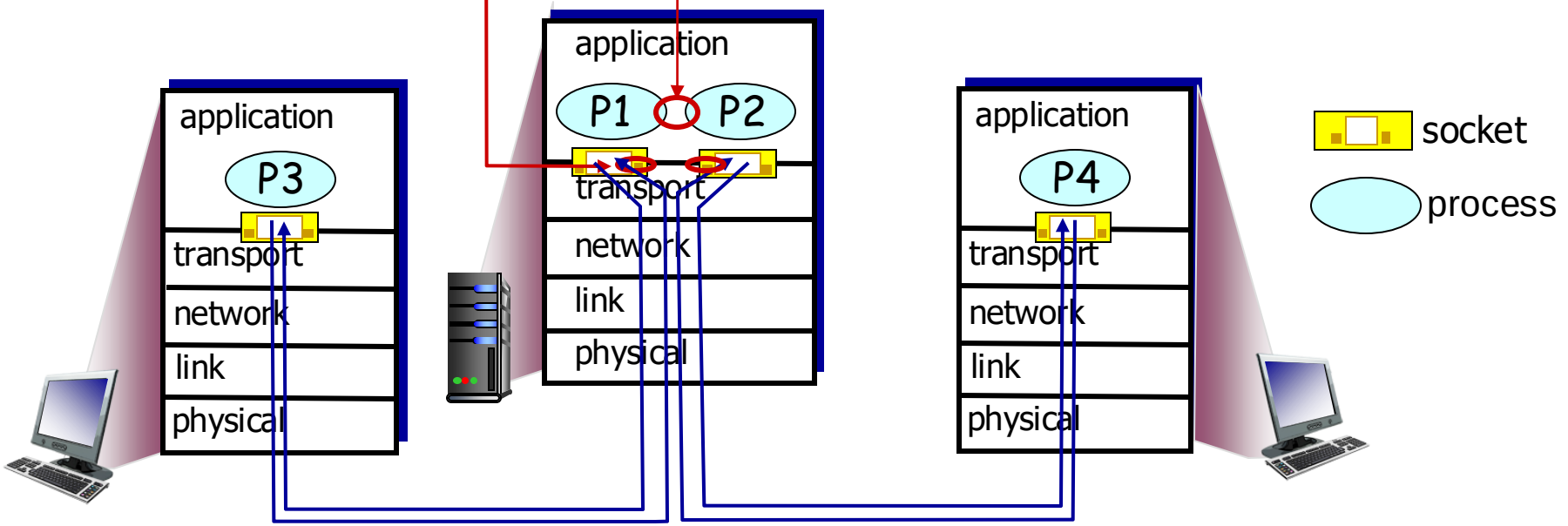
3.3 connectionless transport: UDP

京都大学

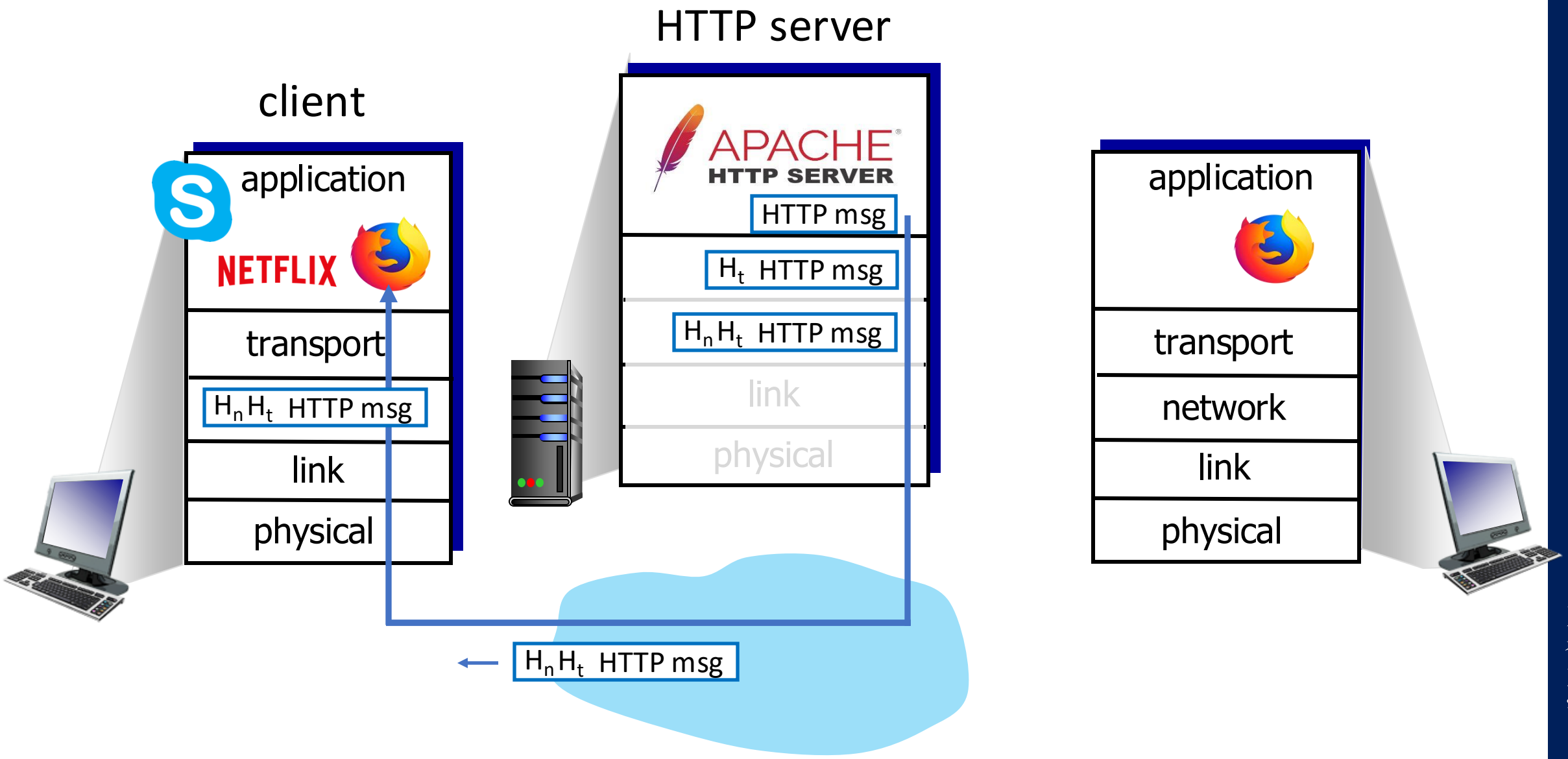
Multiplexing/demultiplexing

multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket

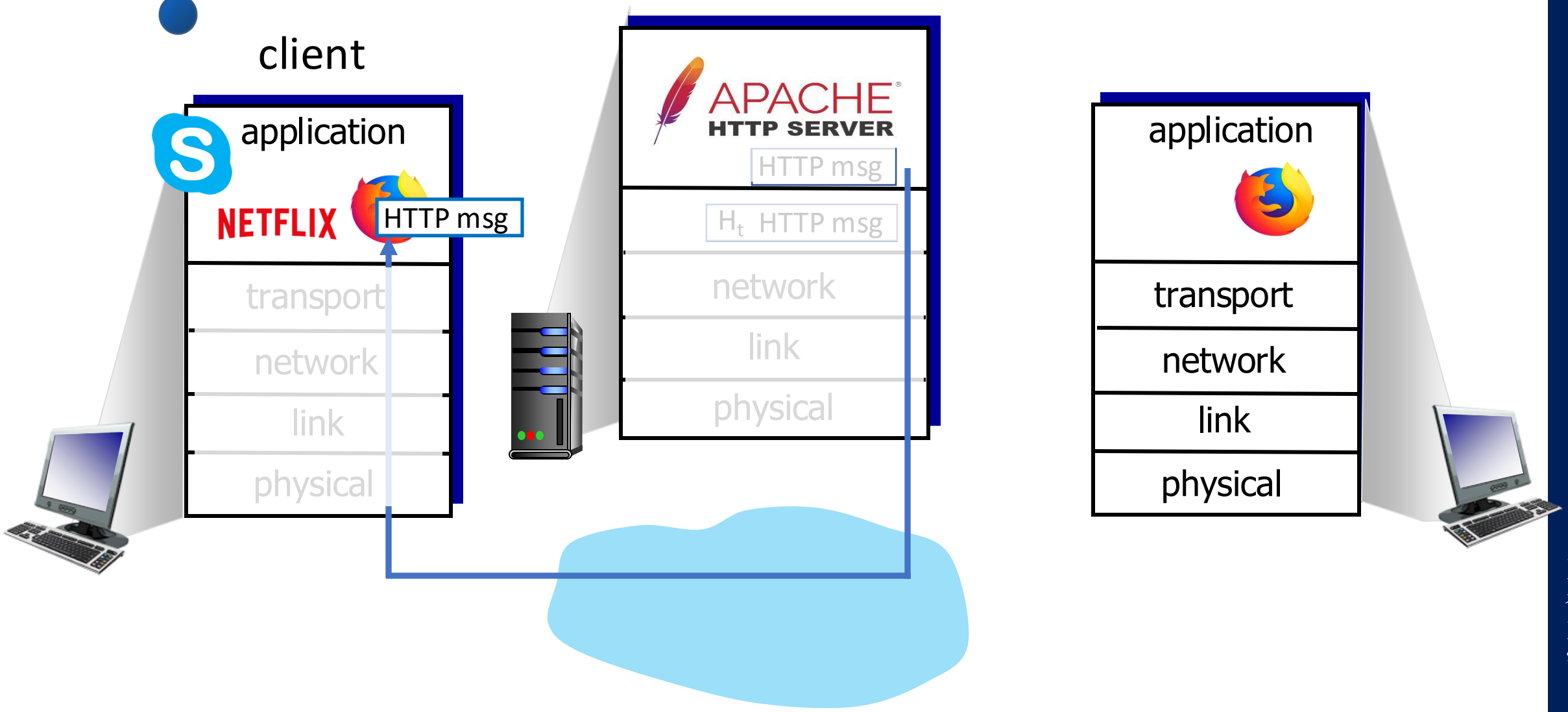


京都大学

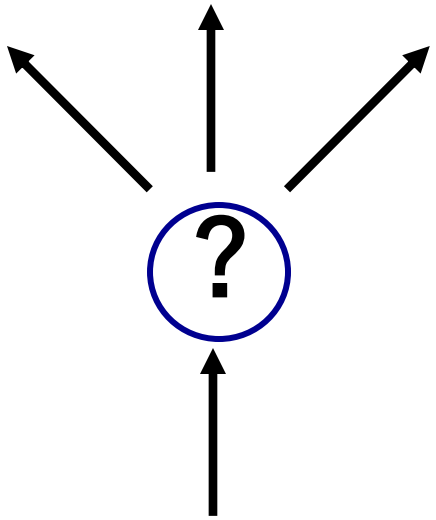

HTTP server
client
application
NETFLIX
transport
$H_n H_t$ HTTP msg
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
$H_n H_t$ HTTP msg
link
physical
application
transport
network
link
physical
$H_n H_t$ HTTP msg

京都大学

Q: how did transport layer know to deliver message to Firefox browser process rather then Netflix process or Skype process?

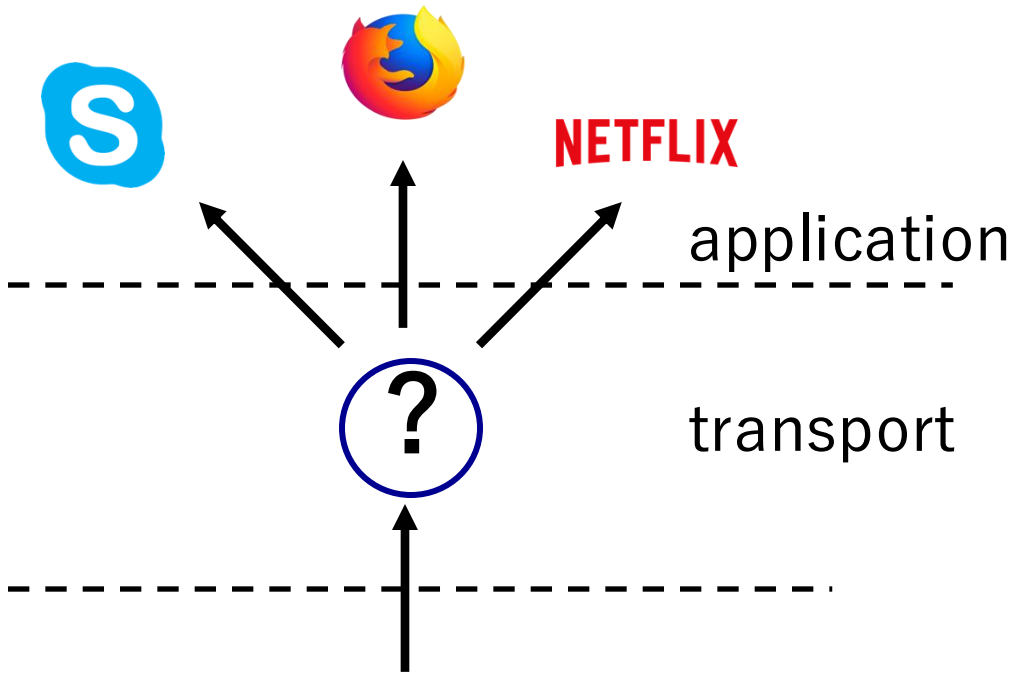


京都大学



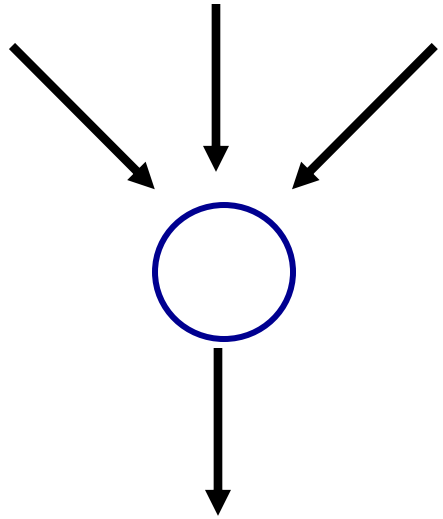
de-multiplexing

京都大学



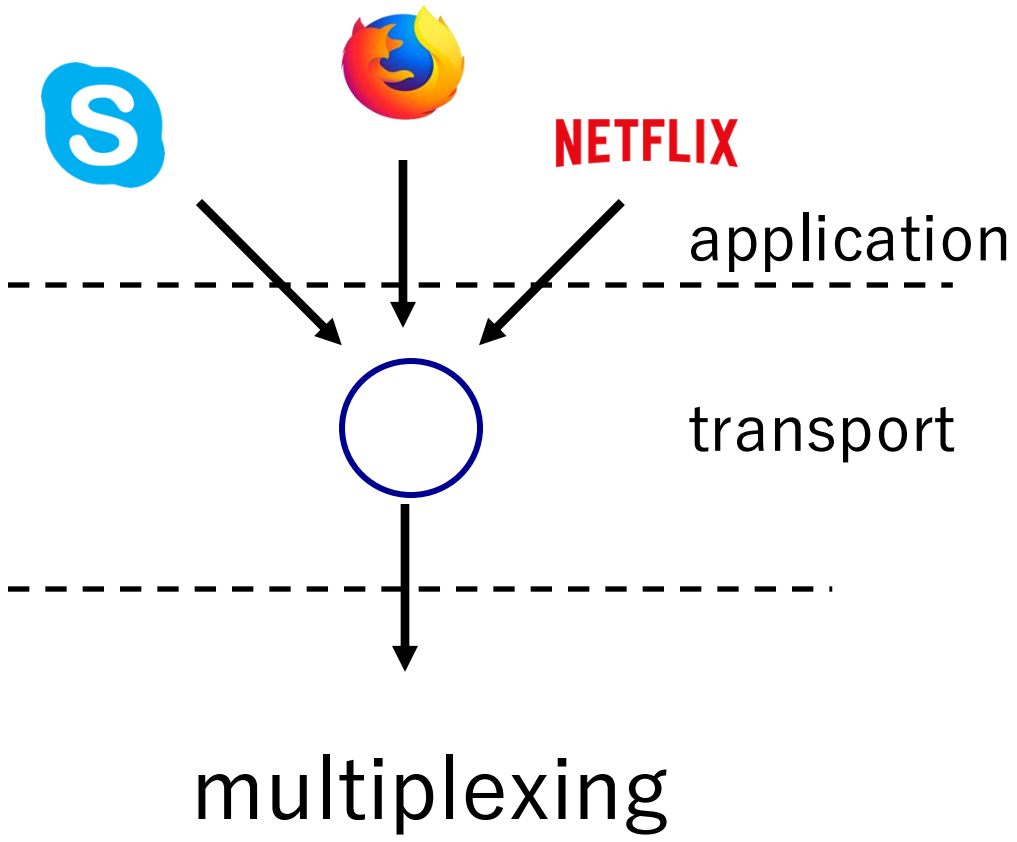
de-multiplexing

京都大学



multiplexing

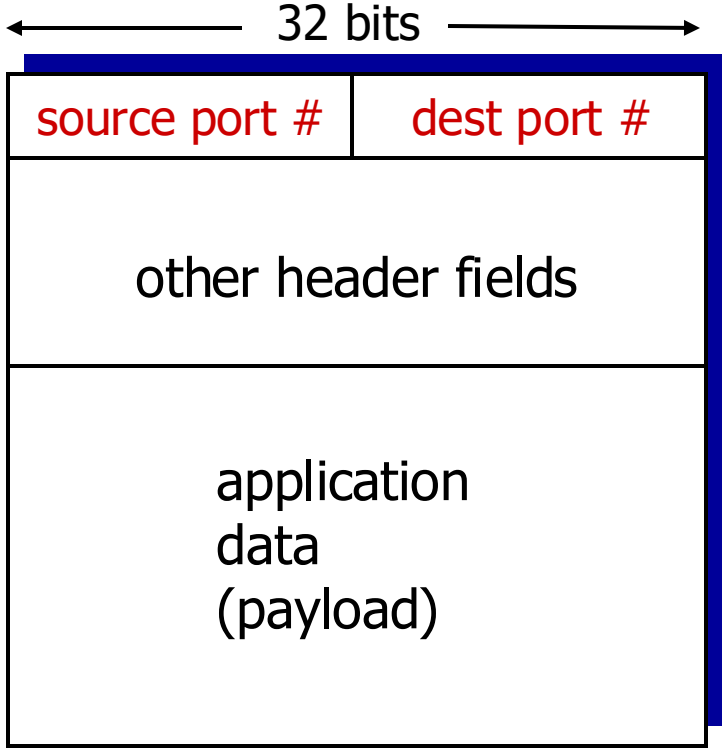
京都大学



京都大学

How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

京都大学

Connectionless demultiplexing

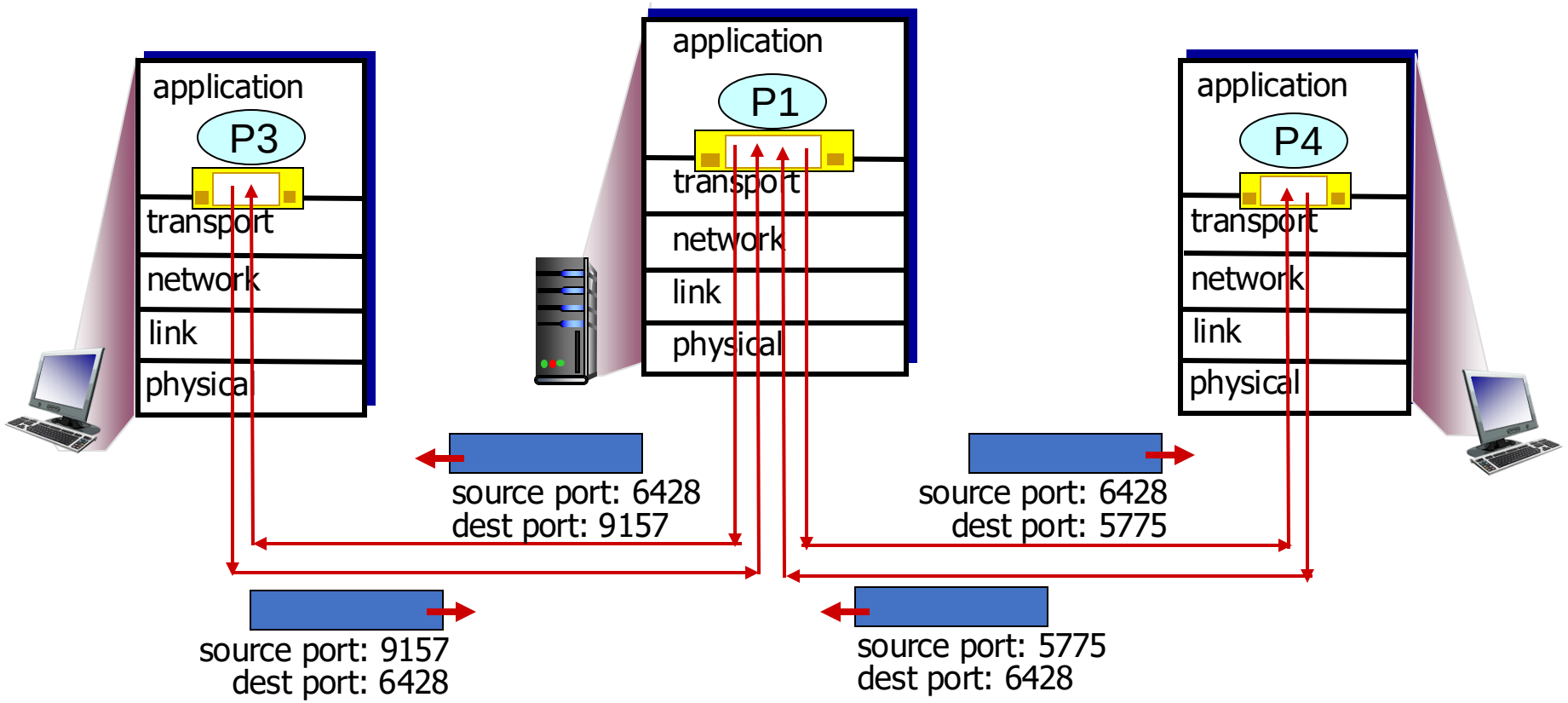
- created socket has host-local port number
- *recall:* when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port number

- when host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number

IP datagrams with *same dest. port number,* but different source IP addresses and/or source port numbers will be directed to *same socket* at destination

京都大学

Connectionless demux: example



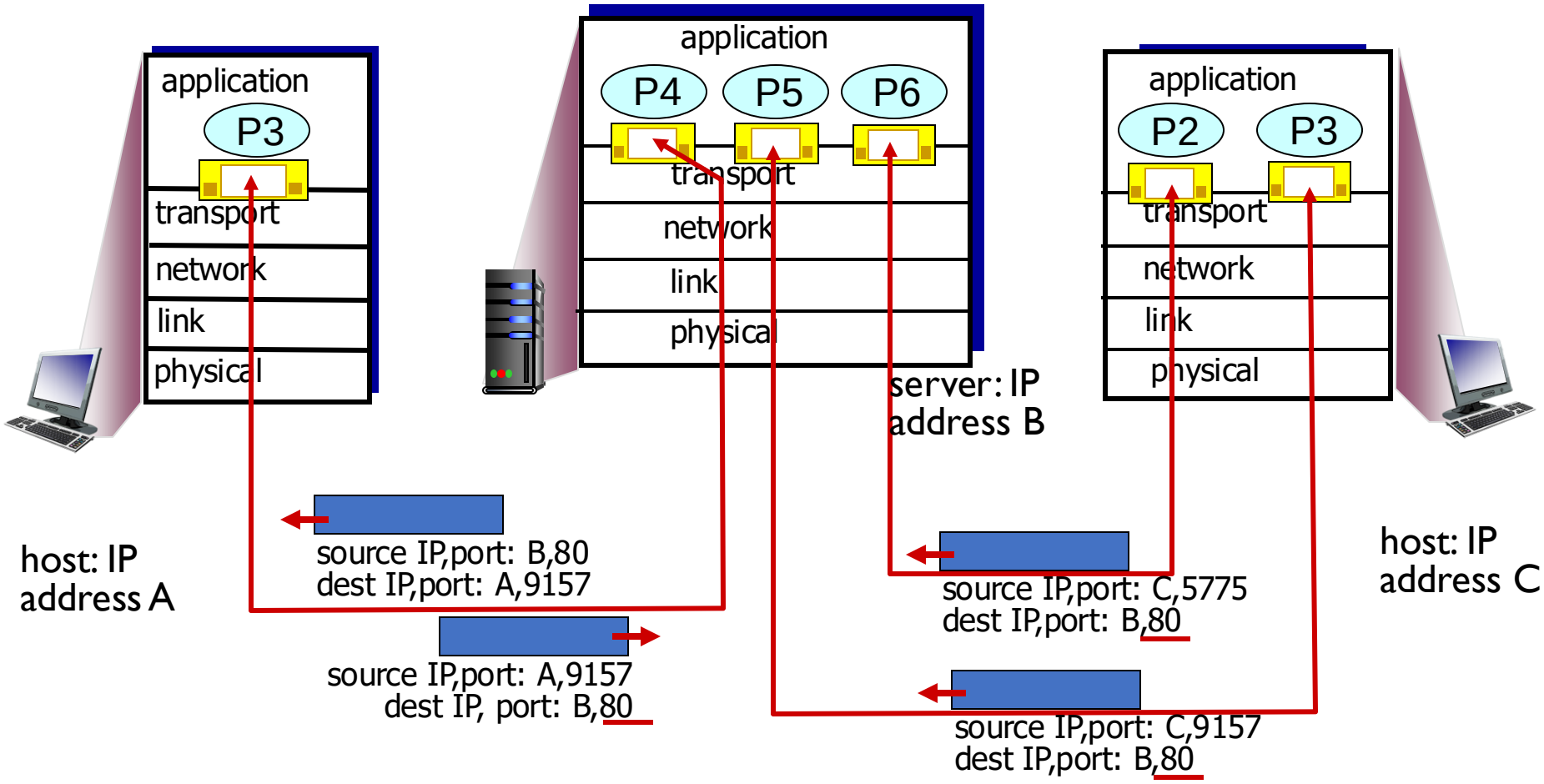
京都大学

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

京都大学

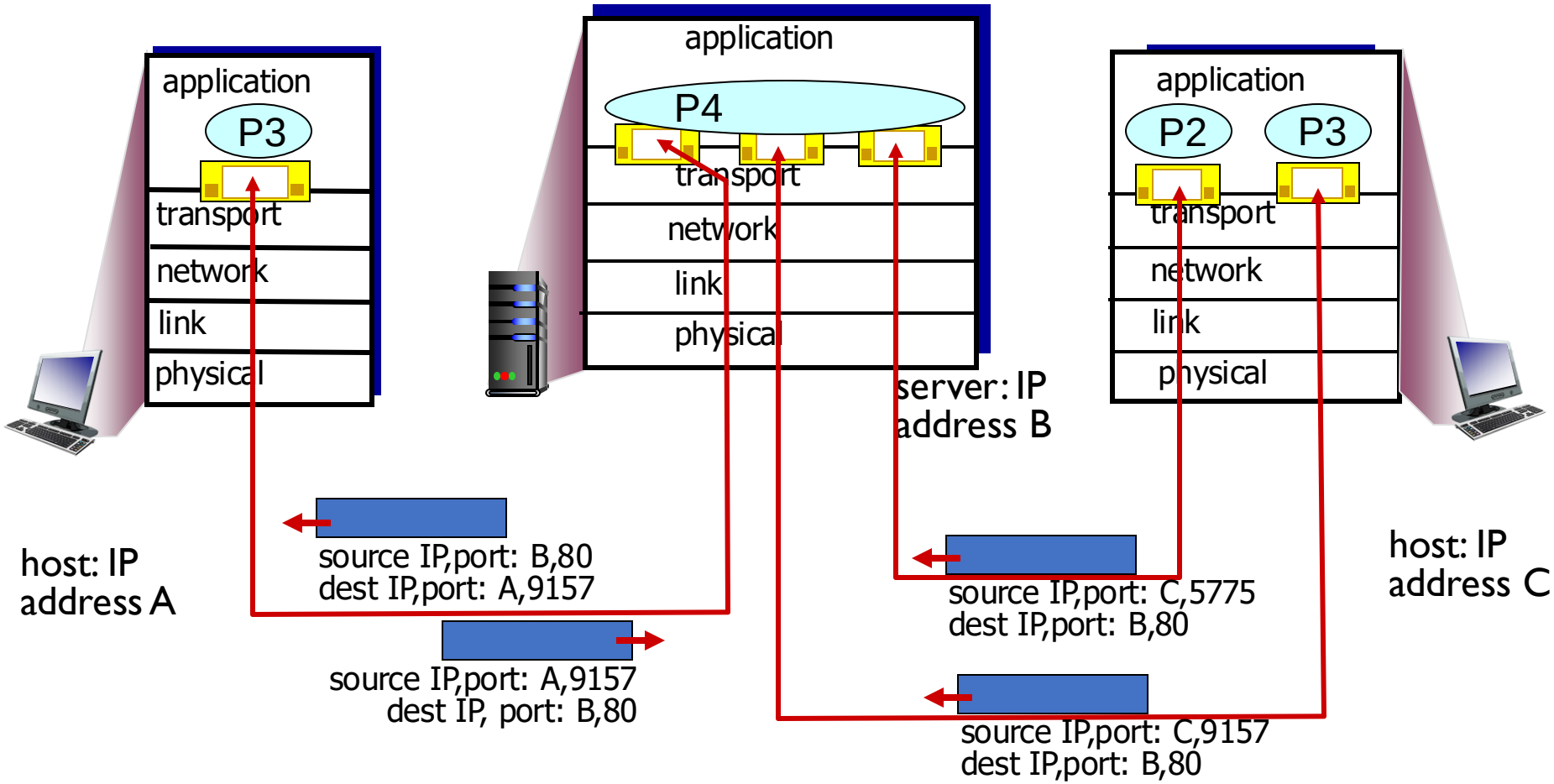
Connection-oriented demux: example



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

京都大学

Connection-oriented demux: example



京都大学

Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

京都大学

Today's lecture

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

京都大学

UDP: User Datagram Protocol

- Very simple Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP (used to manage network printers etc.)
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

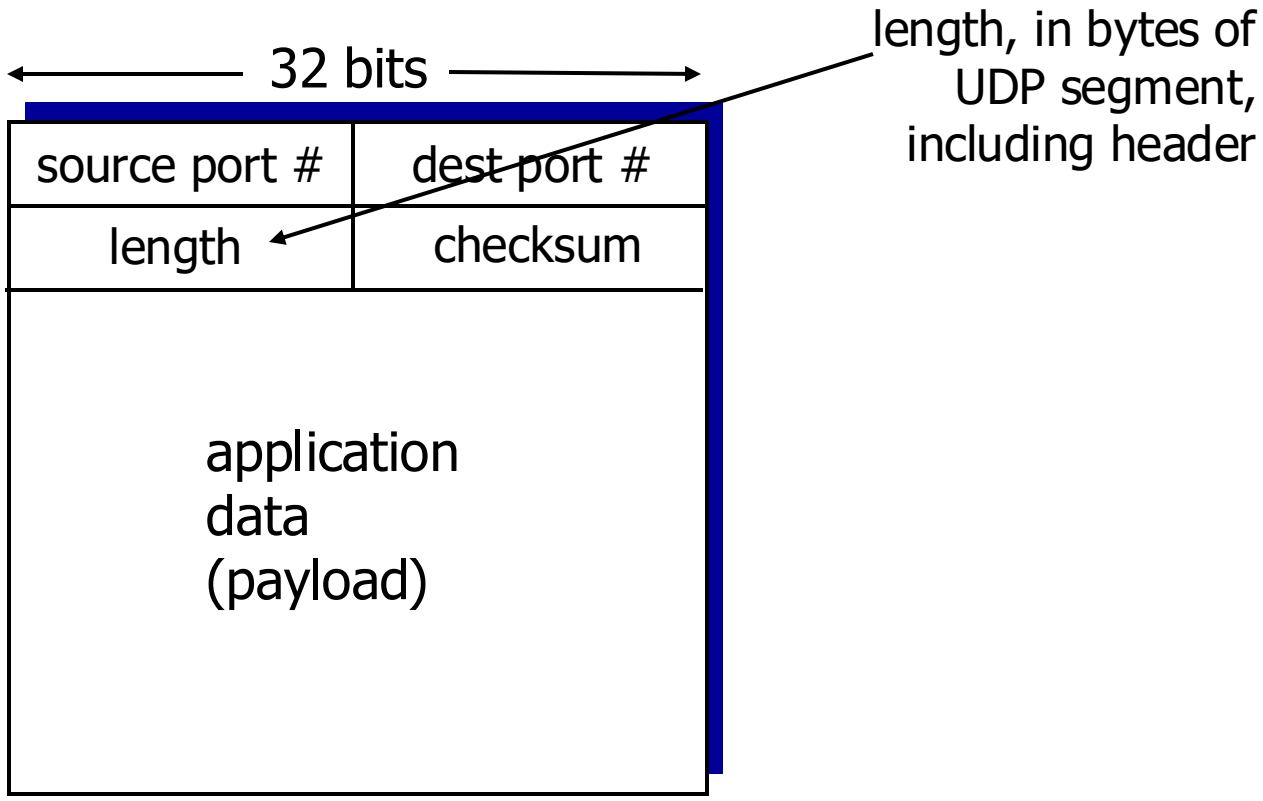
京都大学

Advantages of UDP over TCP

- Finer application-level control over what data is sent and when.
  - Packet will be immediately passed to network layer
  - Real-time applications often require a minimum sending rate.
  - TCP not well-suited for these applications.
- No connection establishment
  - No delay to establish connection.
- No connection state
  - A server can support many more active clients when the application runs over UDP rather than TCP.
- Small packet overhead
  - TCP segment has 20 bytes of header overhead, UDP only 8 bytes.

京都大学

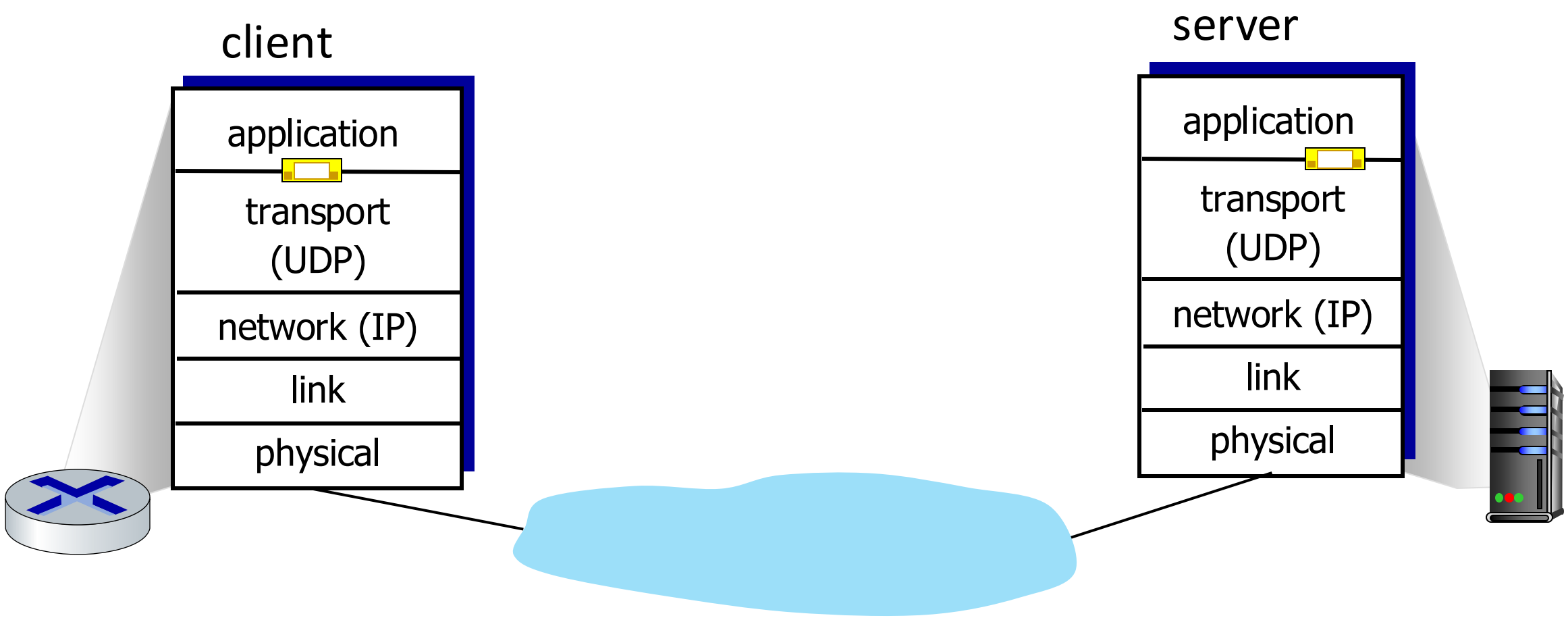
UDP: segment header



UDP segment format

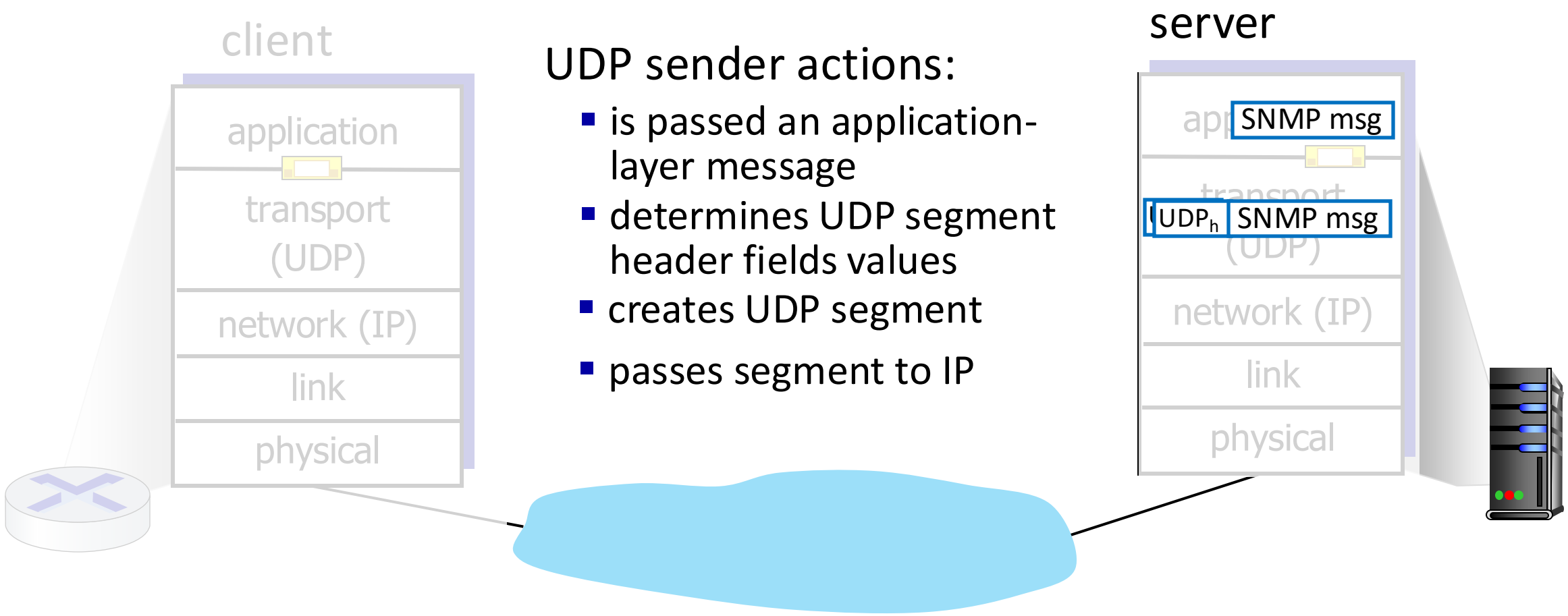
京都大学

UDP: Transport Layer Actions



京都大学

UDP: Transport Layer Actions

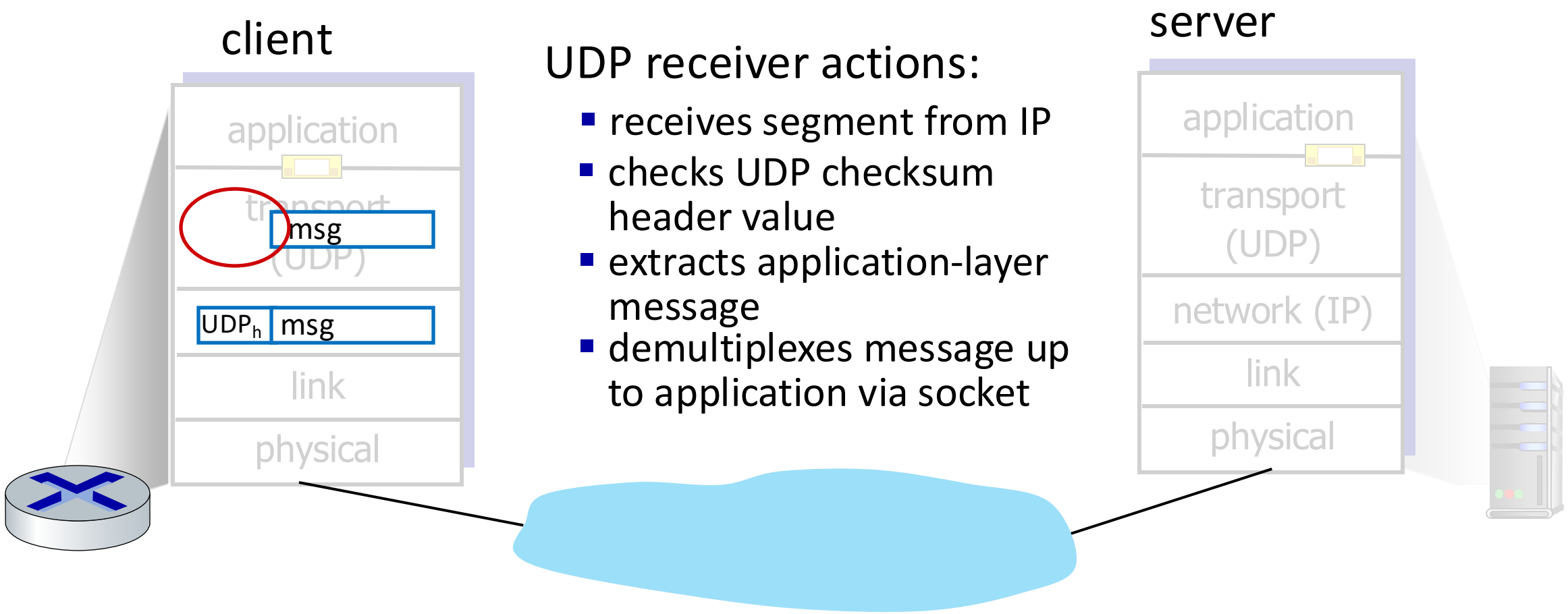


UDP sender actions:

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

京都大学

UDP: Transport Layer Actions



UDP receiver actions:

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

京都大学

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

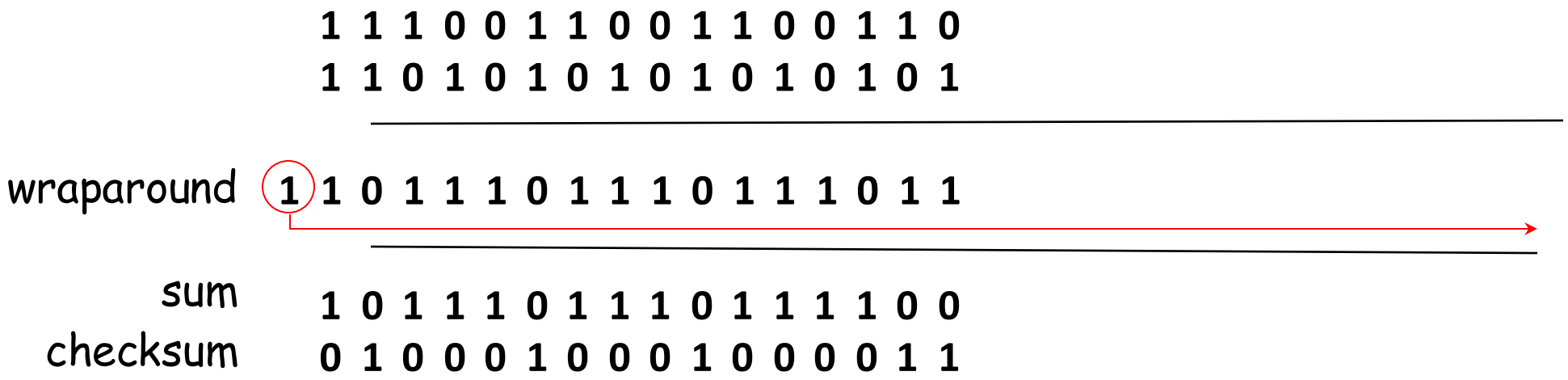
receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

京都大学

Internet checksum: example

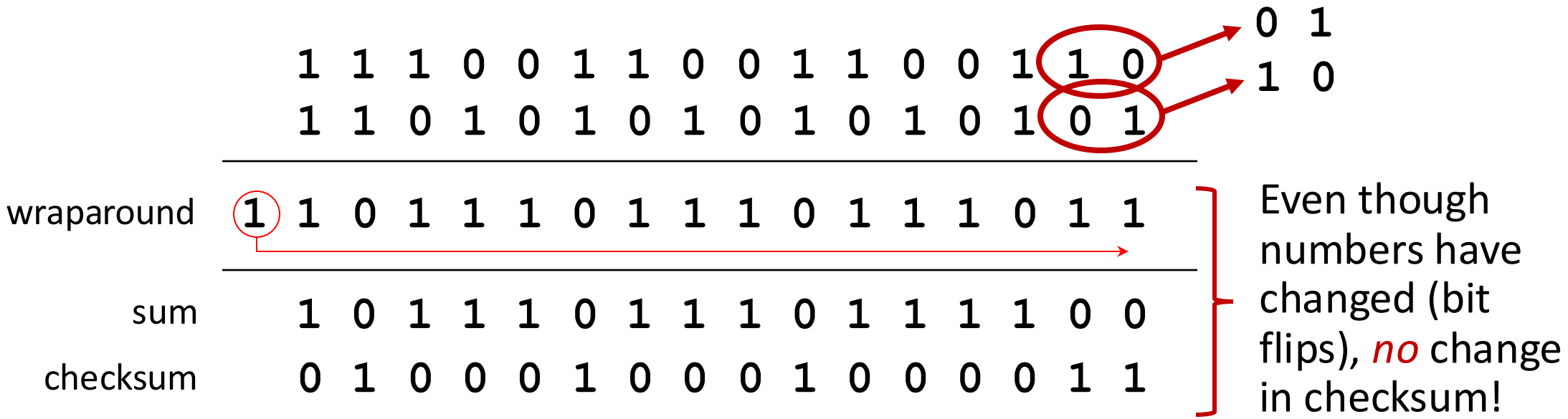
example: add two 16-bit integers



京都大学

Internet checksum: weak protection!

example: add two 16-bit integers



京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

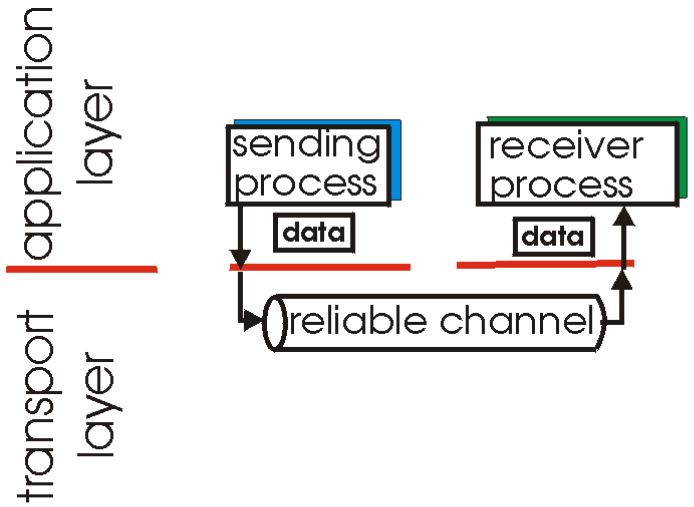
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

京都大学

Principles of reliable data transfer

- important in application, transport, link layers

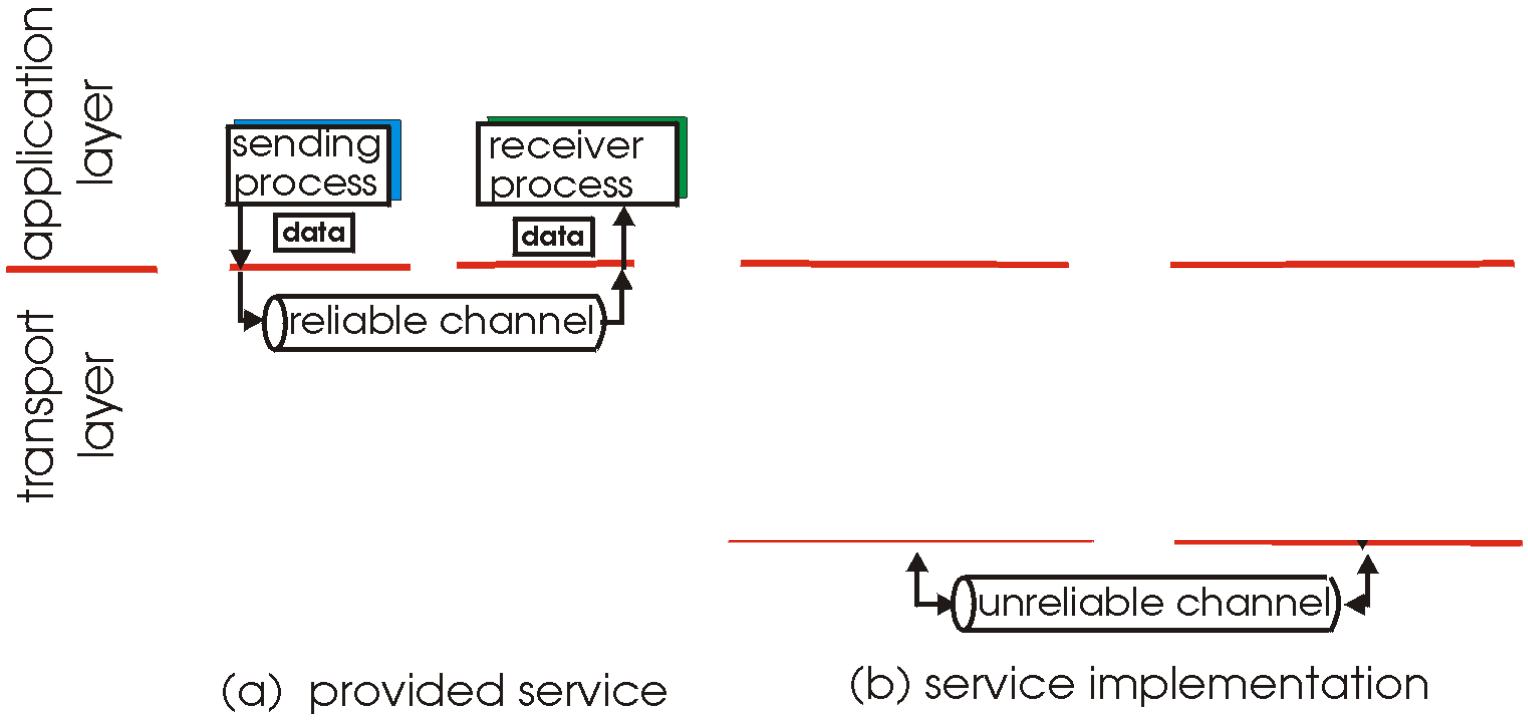


(a) provided service

京都大学

Principles of reliable data transfer

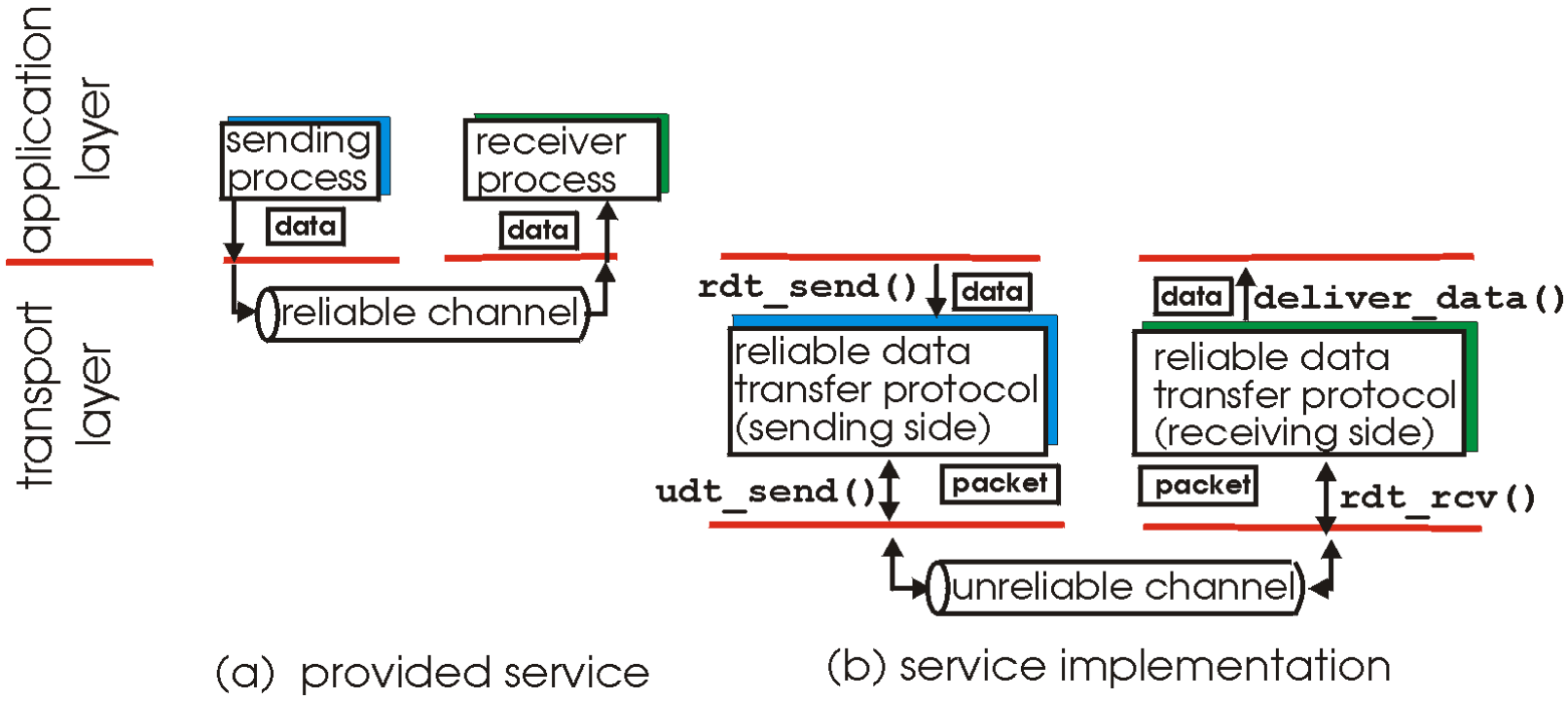
- important in application, transport, link layers
  - top-10 list of important networking topics!



京都大学

Principles of reliable data transfer

- important in application, transport, link layers
  - top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol
- We assume that unreliable channel can lose packets and that packets can have errors.

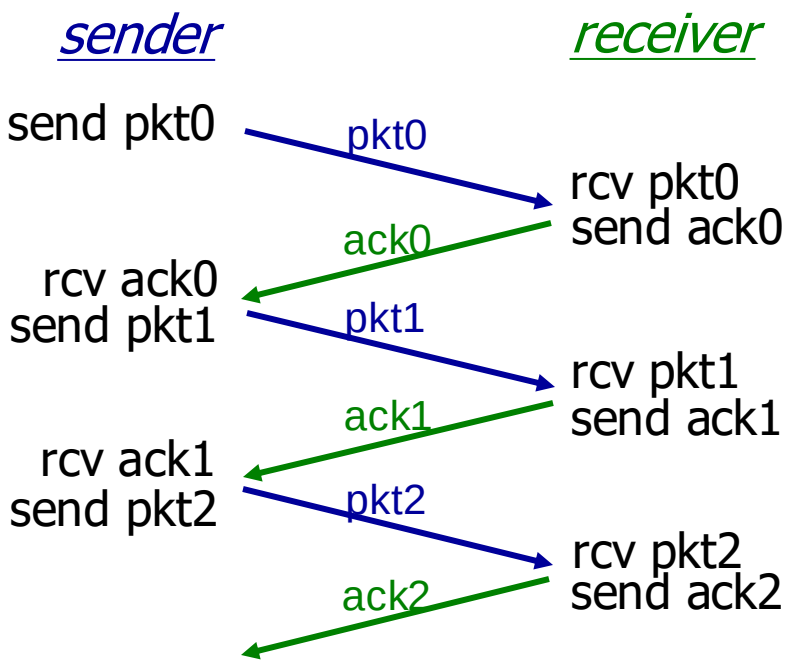
京都大学

Acknowledgment message

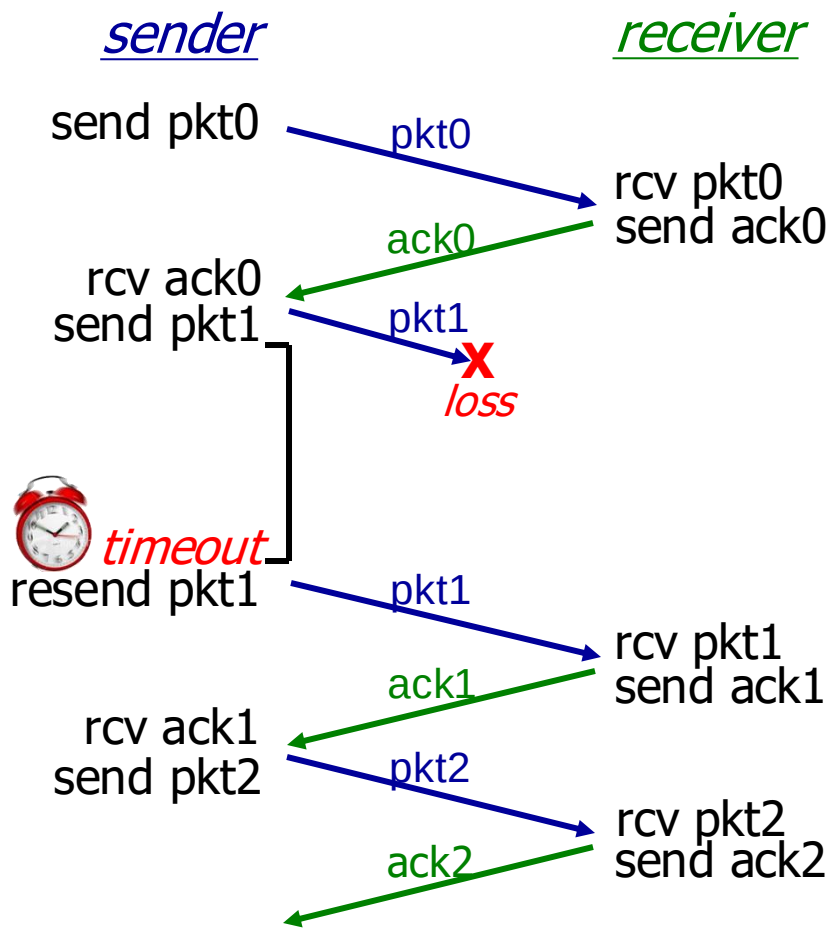
- Packets are sent one by one and are numbered (sequence numbers).
- When a packet is sent, its sequence number is contained in the header information.
- When the receiver receives a packet and it does not contain any errors an acknowledgment message (ACK) is sent for this packet and the receiver remembers the sequence number.
- The sender waits for the acknowledgment packet until it sends the next packet.
- If the sender does not receive an acknowledgment for some time, the same packet is sent again.
- If the receiver receives a packet with a sequence number other than expected or the packet has errors, the packet is discarded, and an acknowledgment message for the last valid packet is send.

京都大学

Reliable data transfer using ack messages



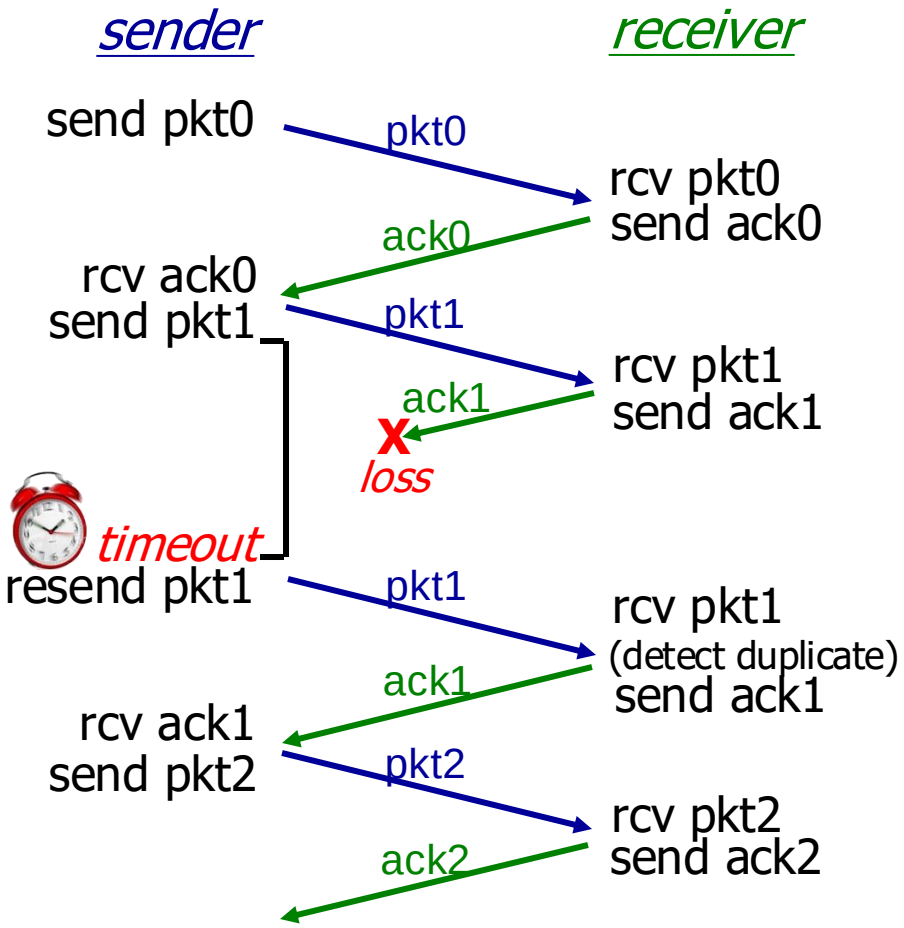
(a) no loss



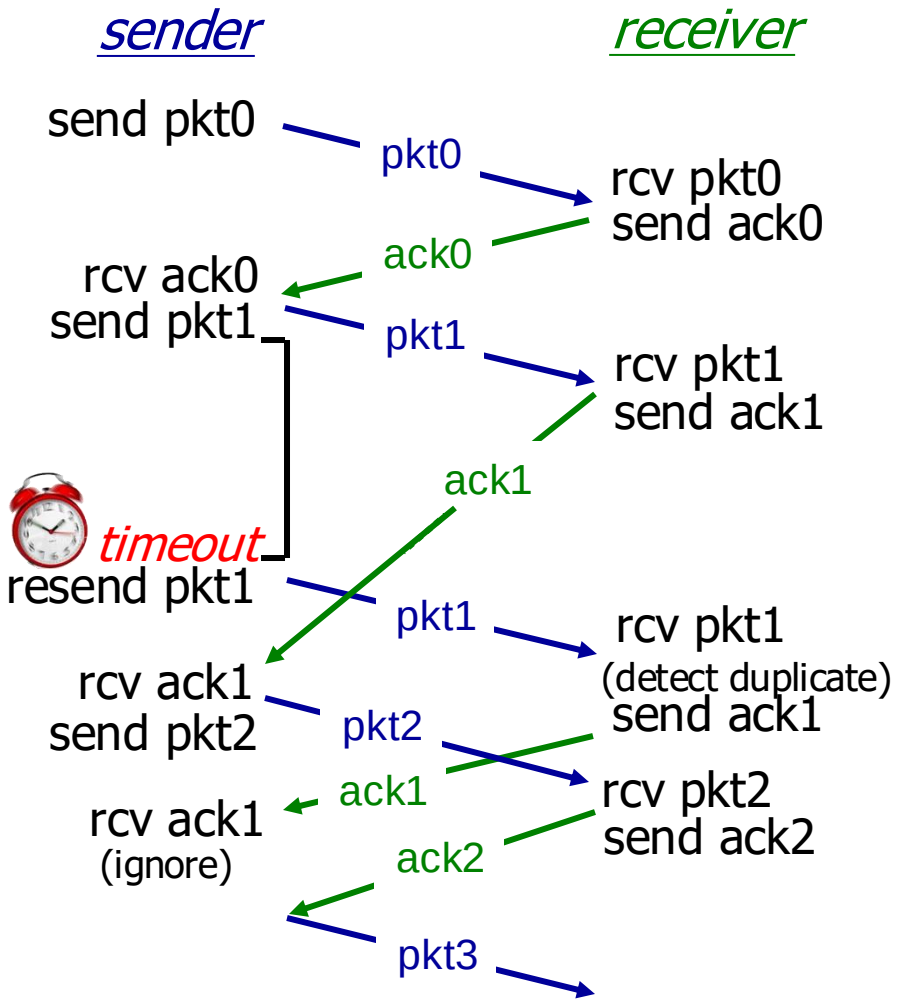
(b) packet loss

京都大学

Reliable data transfer using ack messages



(c) ACK loss



(d) premature timeout/ delayed ACK

Note: 1 bit sequence number (0 or 1) suffices

京都大学

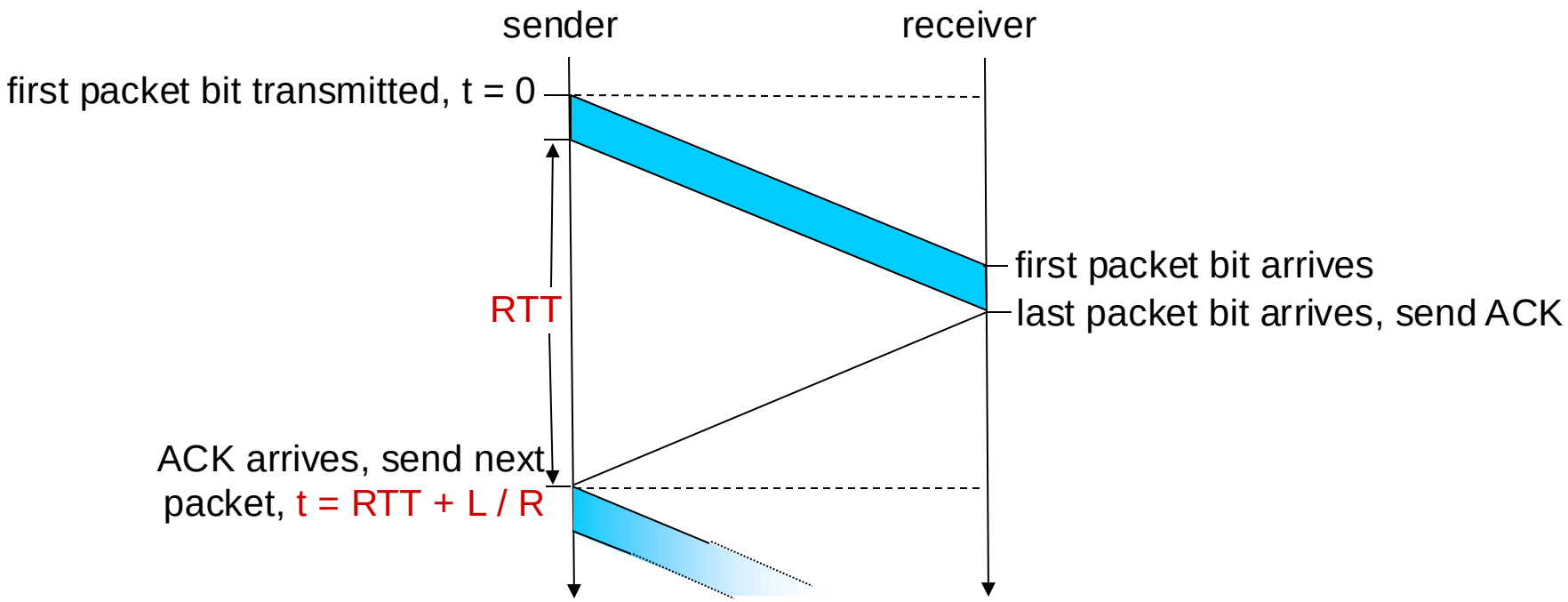
Performance

- Protocol is correct, but performance is not good
- We consider the utilization of the sender, i.e., the fraction of time the sender is busy sending data
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
- Time to transmit data into channel:

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

京都大学

Stop-and-wait operation



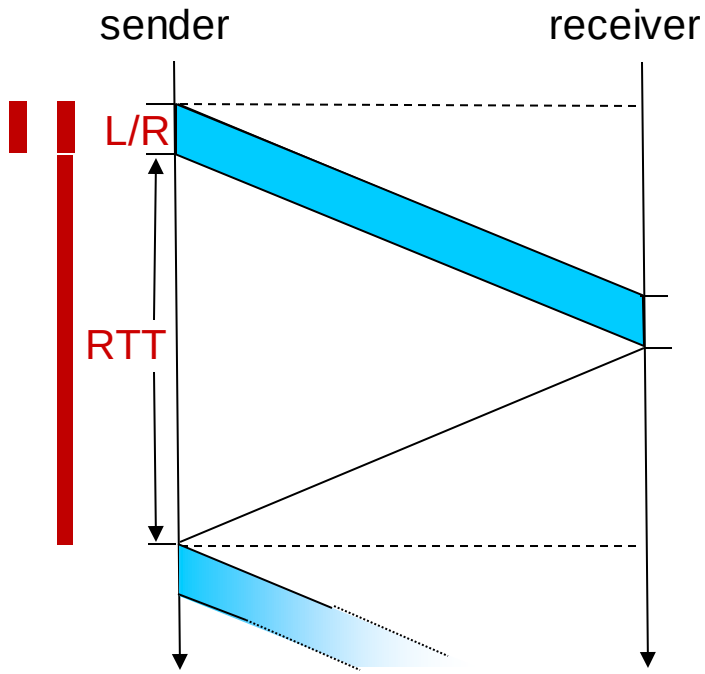
京都大学

Stop-and-wait operation

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

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

$$= 0.00027$$

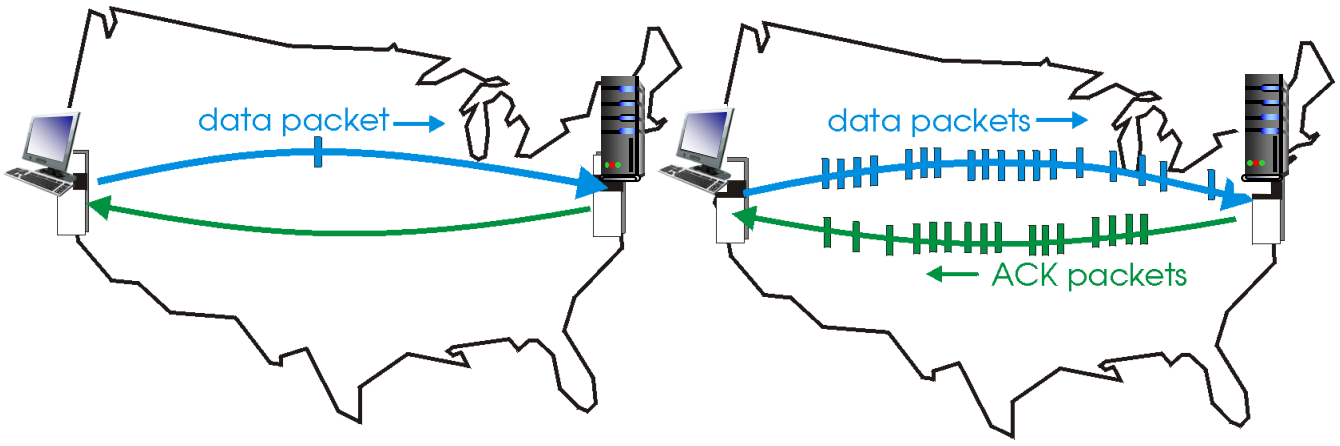


- if RTT=30 msec, 1KB packet every 30 msec: 33kB/sec throughput over 1 Gbps link
- Protocol limits performance of underlying infrastructure (channel)

京都大学

Pipelined protocols

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

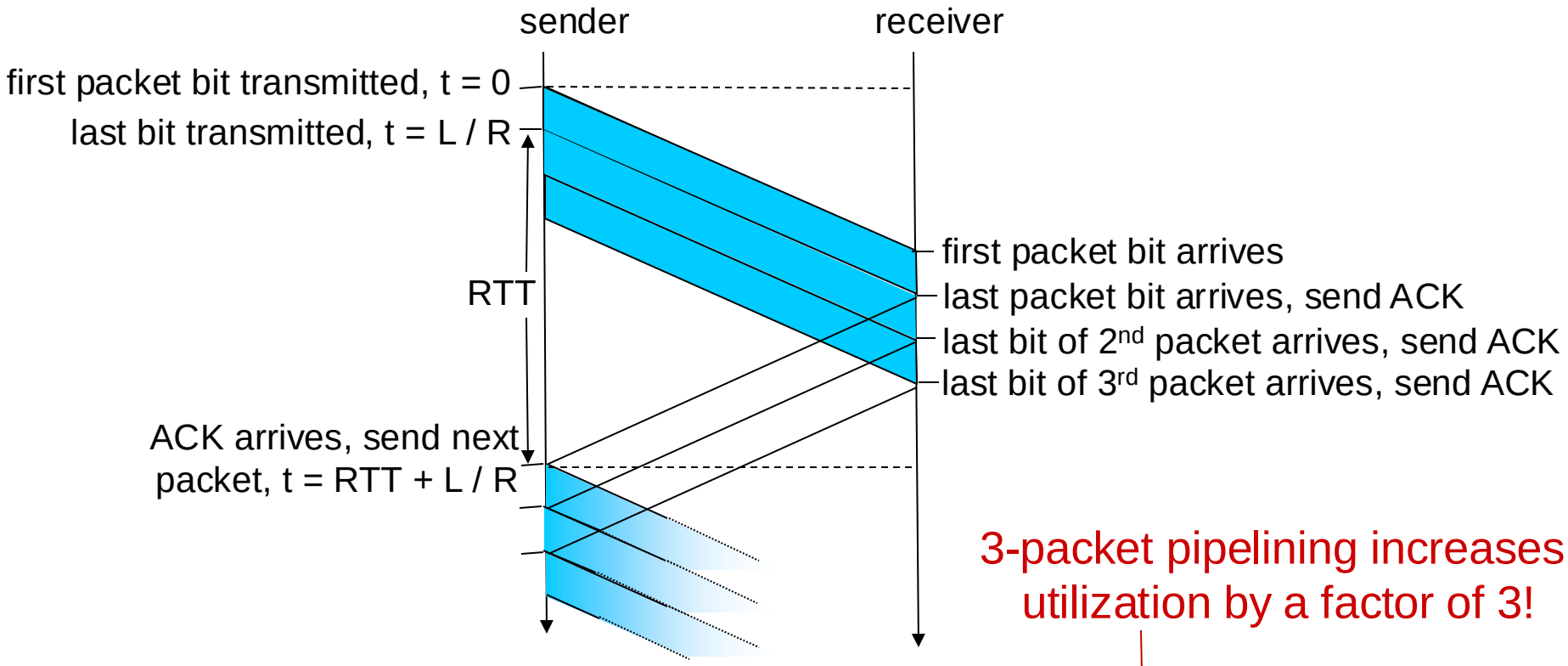


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

京都大学

Pipelining: increased utilization



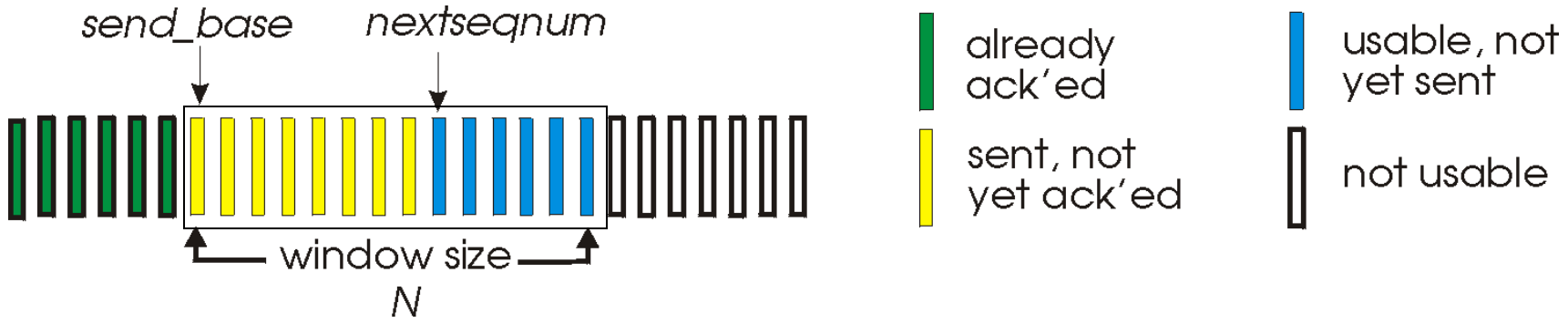
3-packet pipelining increases utilization by a factor of 3!

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

京都大学

Go-Back-N: sender

- k-bit sequence number in packet header
- "window" of up to N, consecutive unacknowledged packets allowed



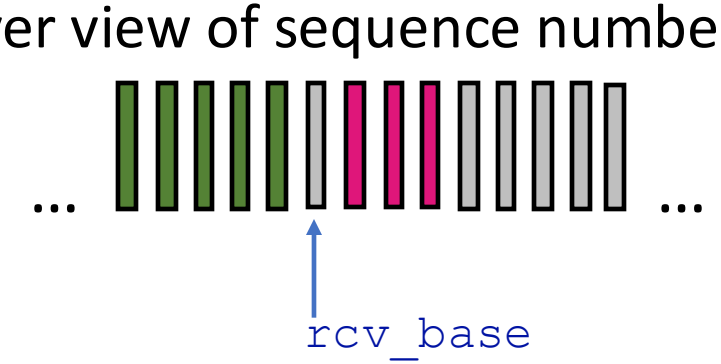
- ACK(n): ACKs all pkts up to, including sequence number n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher sequence number packets in window

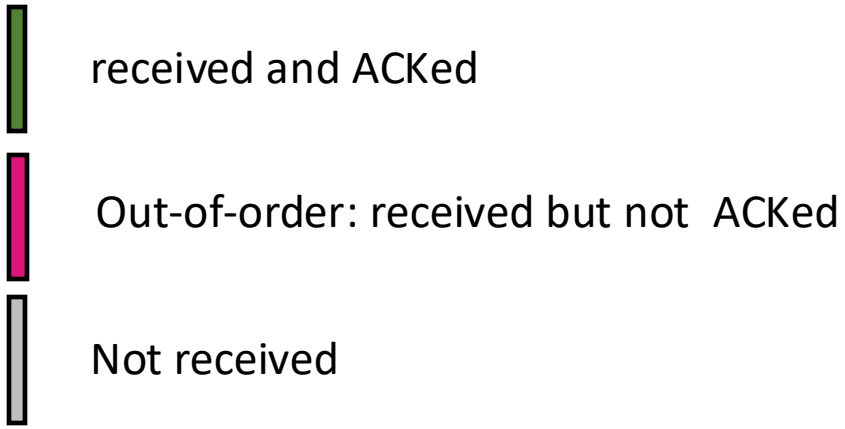
京都大学

Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq number
  - 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 packet with highest in-order sequence number

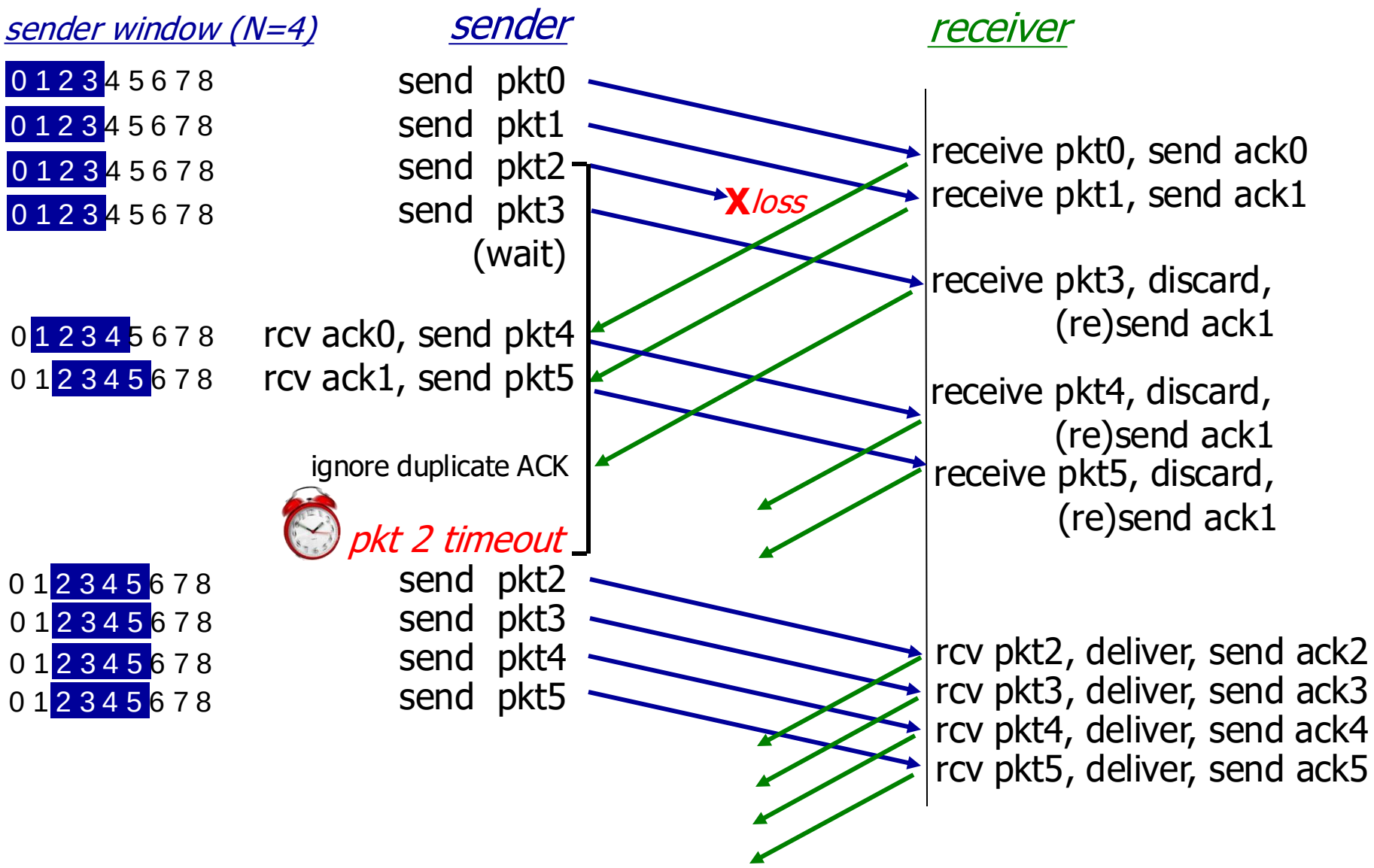
Receiver view of sequence number space:





京都大学

GBN in action



京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

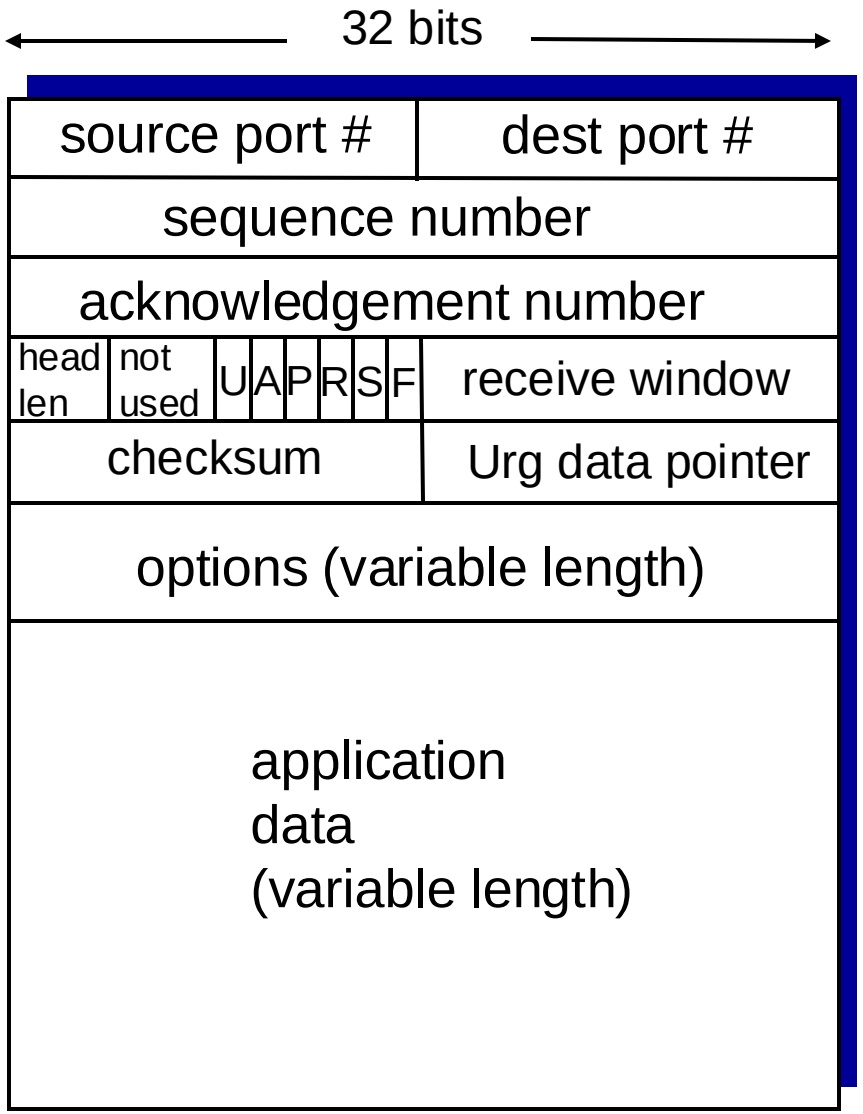
京都大学

TCP: Overview

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no “message boundaries”
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

京都大学

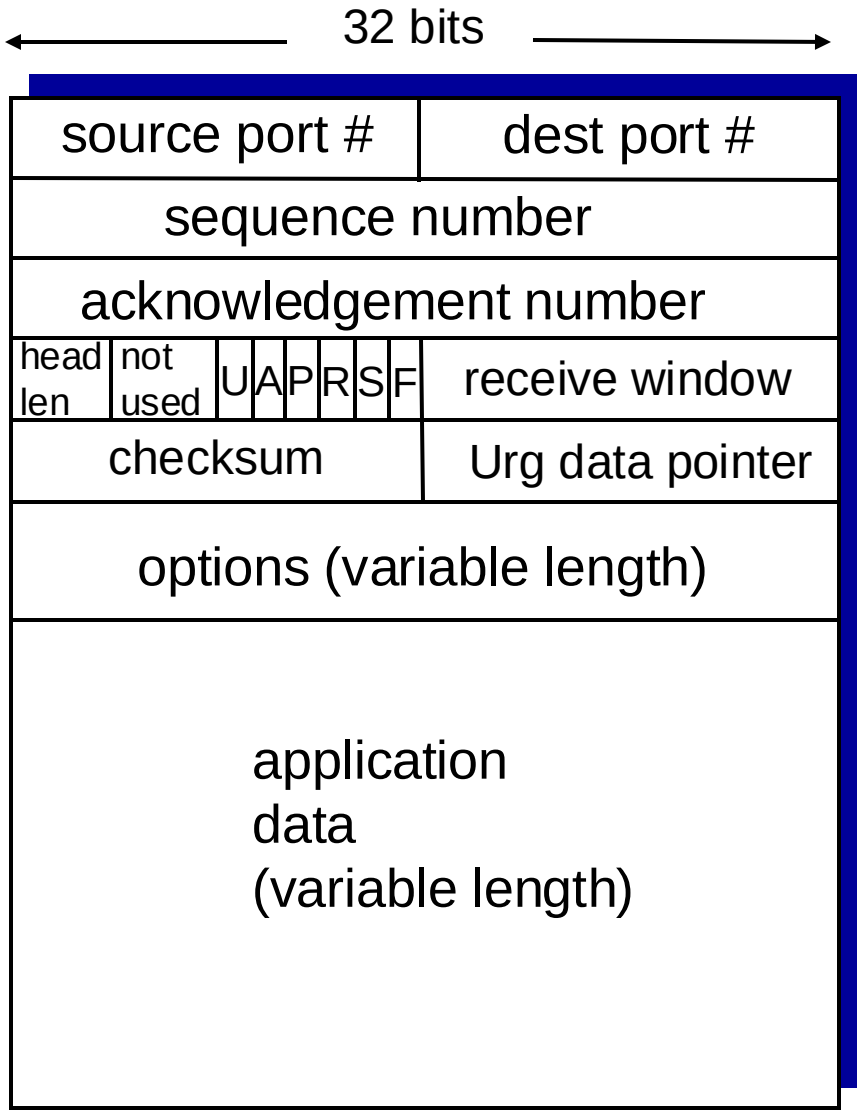
TCP segment structure



京都大学

TCP Segment Header

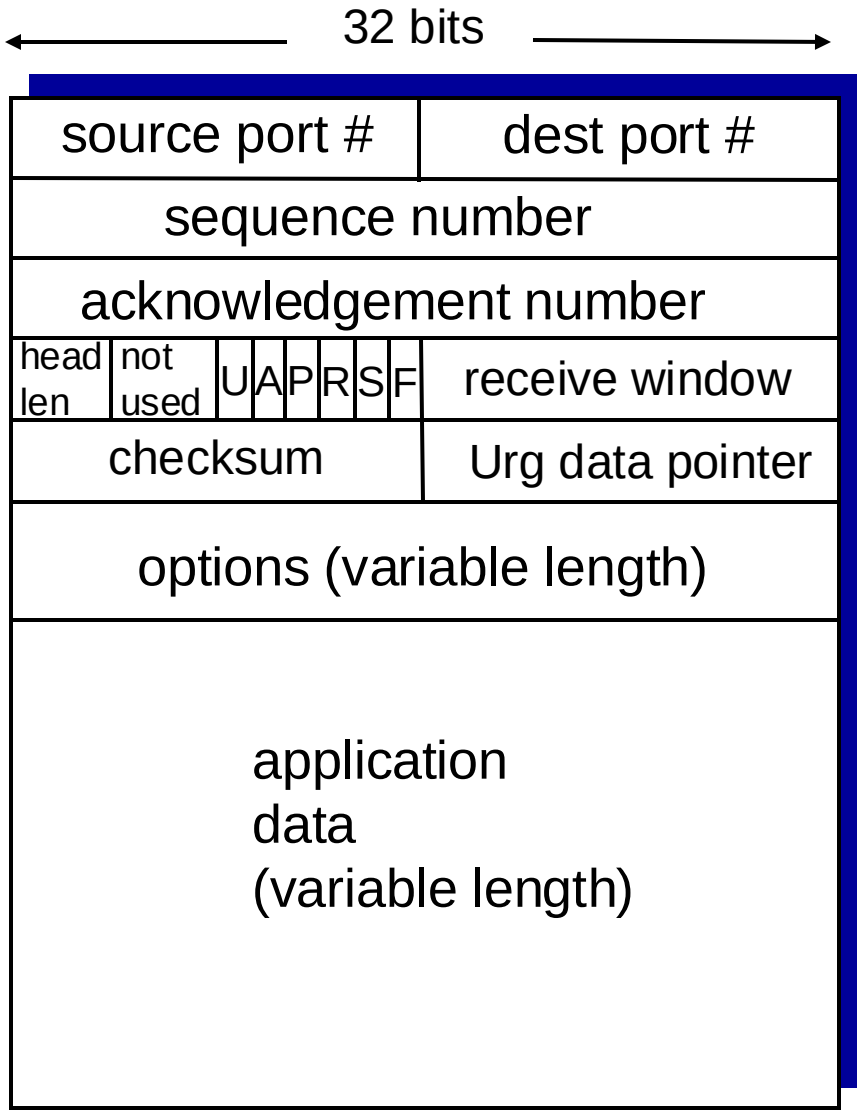
- Source Port, Destination Port
- Sequence Number
  - At the transport layer application data is split into several smaller segments.
  - Sequence Number is used to keep track of the position of the current segment in the sequence.
- Acknowledgment Number: Number of the next expected segment
  - Used for reliable data transfer.



京都大学

TCP Segment Header

- head len: Length of Header
  - For the receiver to know where the header ends and the application data begins
- 6 TCP control flags
- receive window
  - Number of bytes the receiver is willing to accept until an acknowledgment will be sent
- Checksum
  - Same as in UDP
- Urg data pointer
  - Point out segments that are urgent
  - Rarely used
- Options
  - Additional options such as more complicated flow control
  - rarely used



京都大学

TCP seq. numbers, ACKs

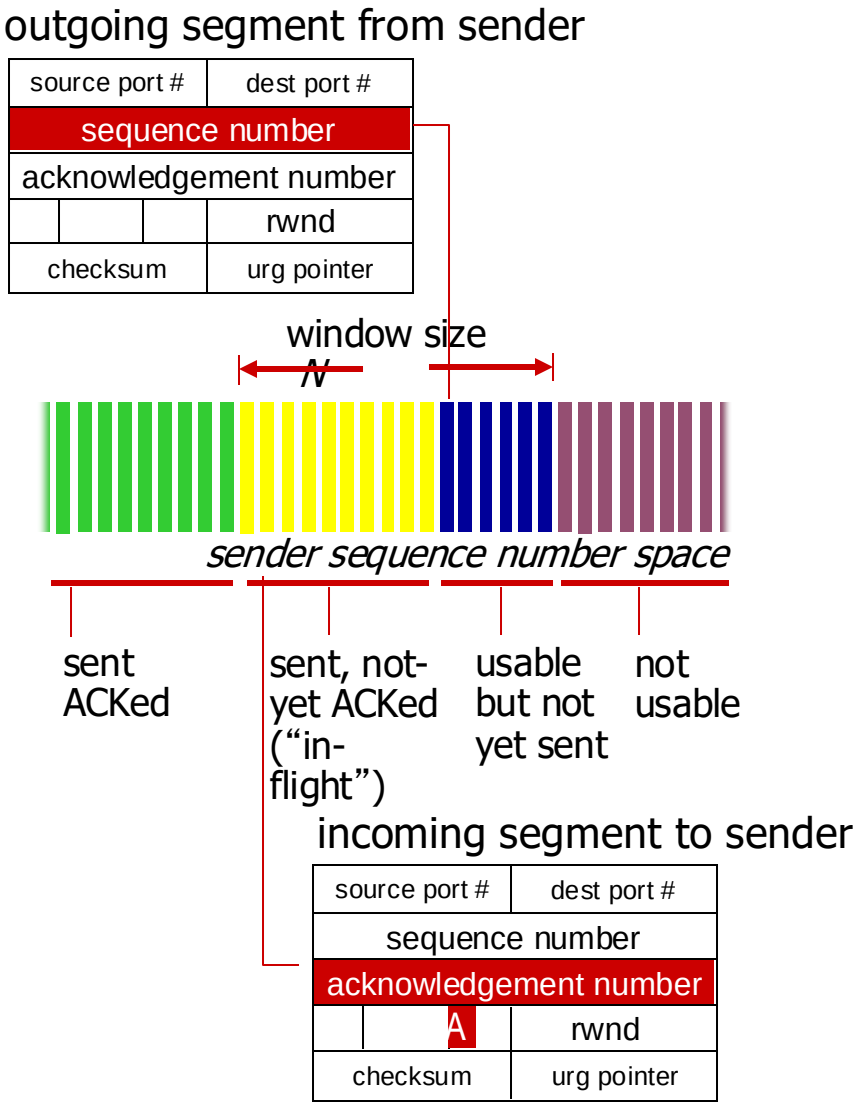
sequence numbers:

- byte stream "number" of first byte in segment's data

acknowledgements:

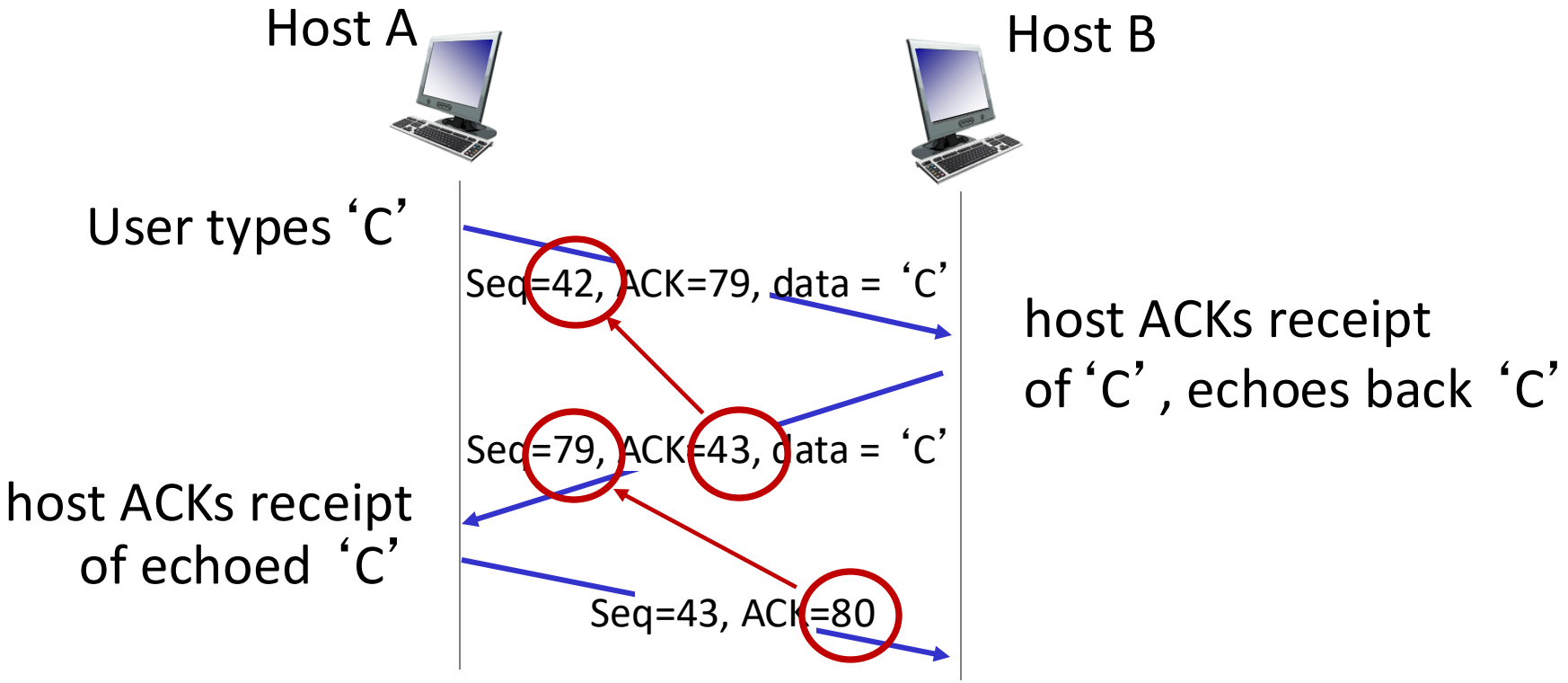
- Sequence number of next byte expected from other side
- cumulative ACK

- TCP does not specify how the receiver handles out-of-order segments, it is up to implementor



京都大学

TCP sequence numbers, ACKs



simple telnet scenario

京都大学

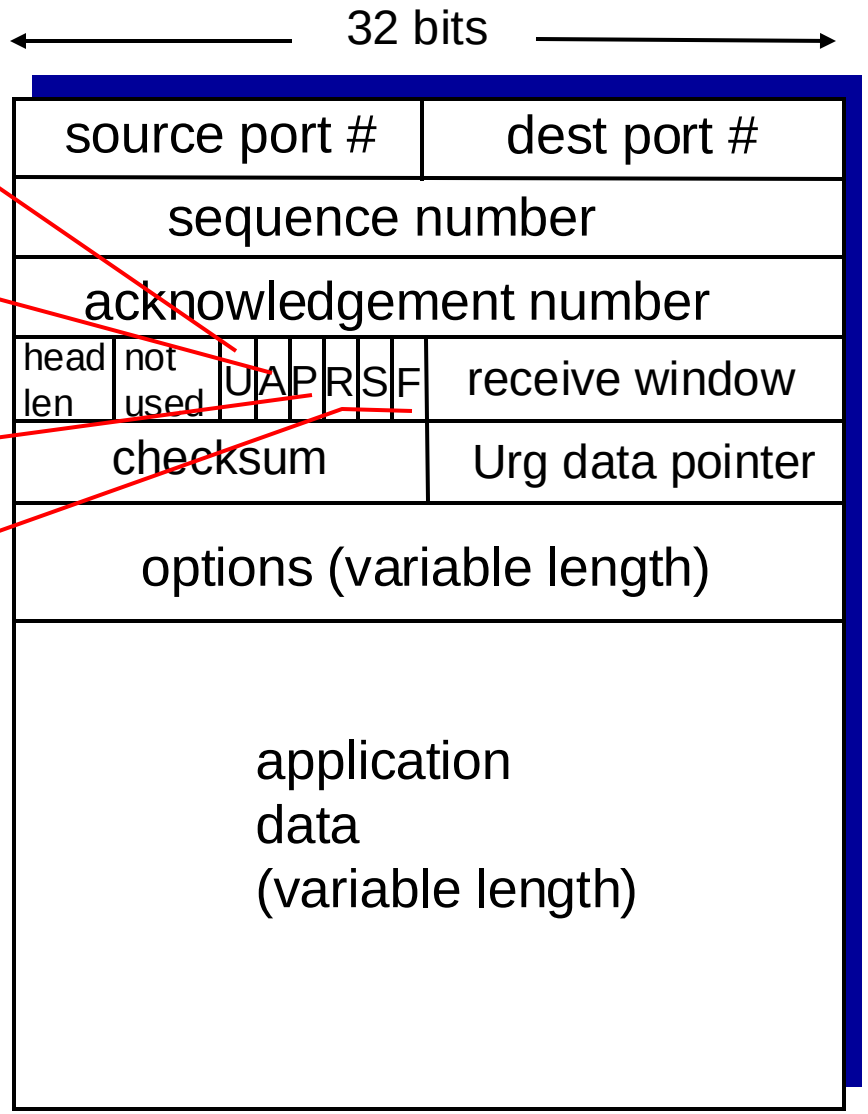
TCP control flags

URG: urgent data
(generally not used)

ACK: ACK #
valid

PSH: push data now
(generally not used)

RST, SYN, FIN:
connection estab
(setup, teardown
commands)



京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

京都大学

TCP reliable data transfer

- TCP creates reliable data transfer service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

京都大学

TCP Sender (simplified)

event: data received from application

- create segment with seq number
- seq number is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **`TimeOutInterval`**

event: timeout

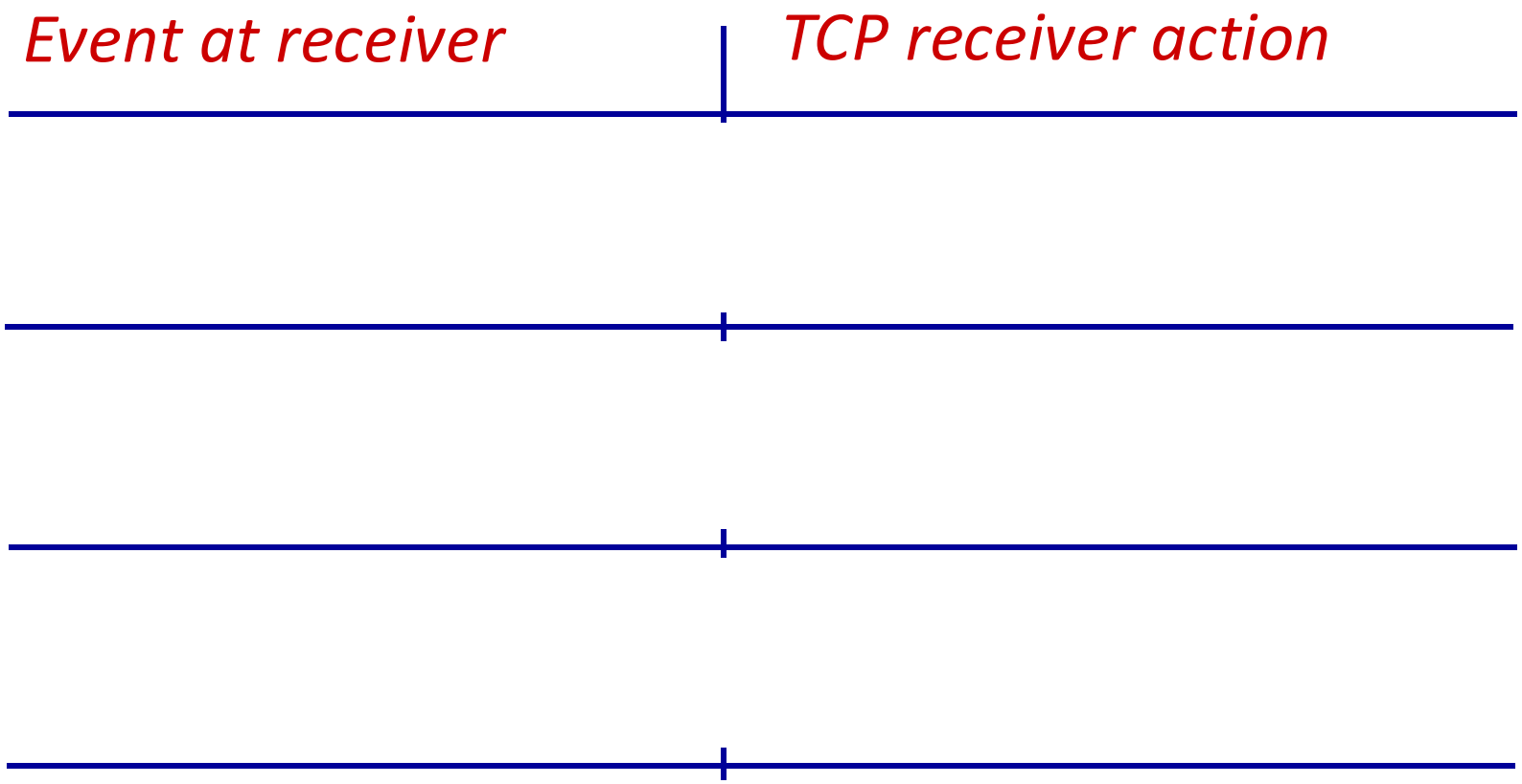
- retransmit segment that caused timeout
- restart timer

event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

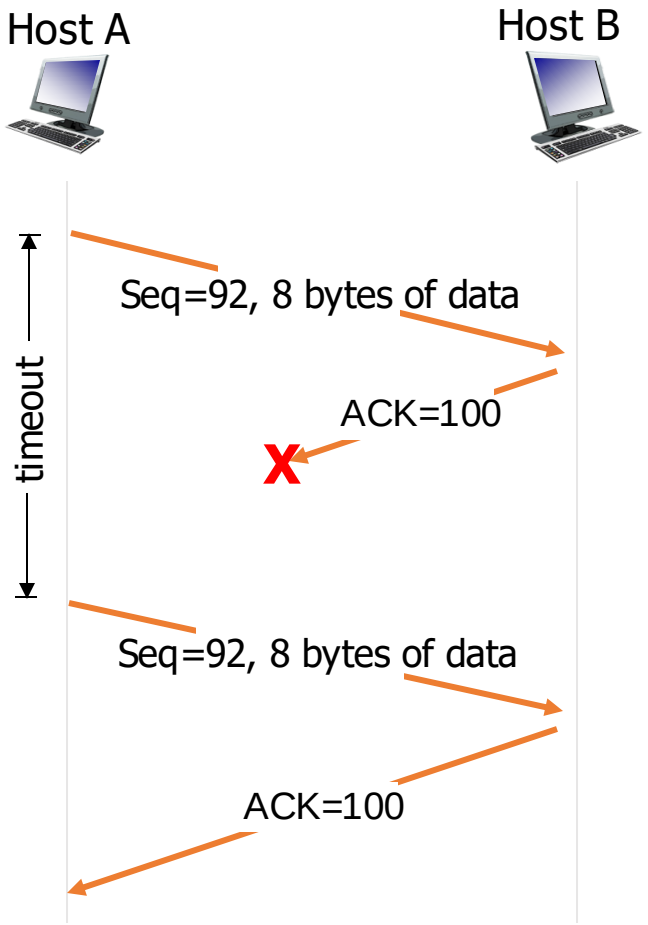
京都大学

TCP Receiver: ACK generation [RFC 5681]

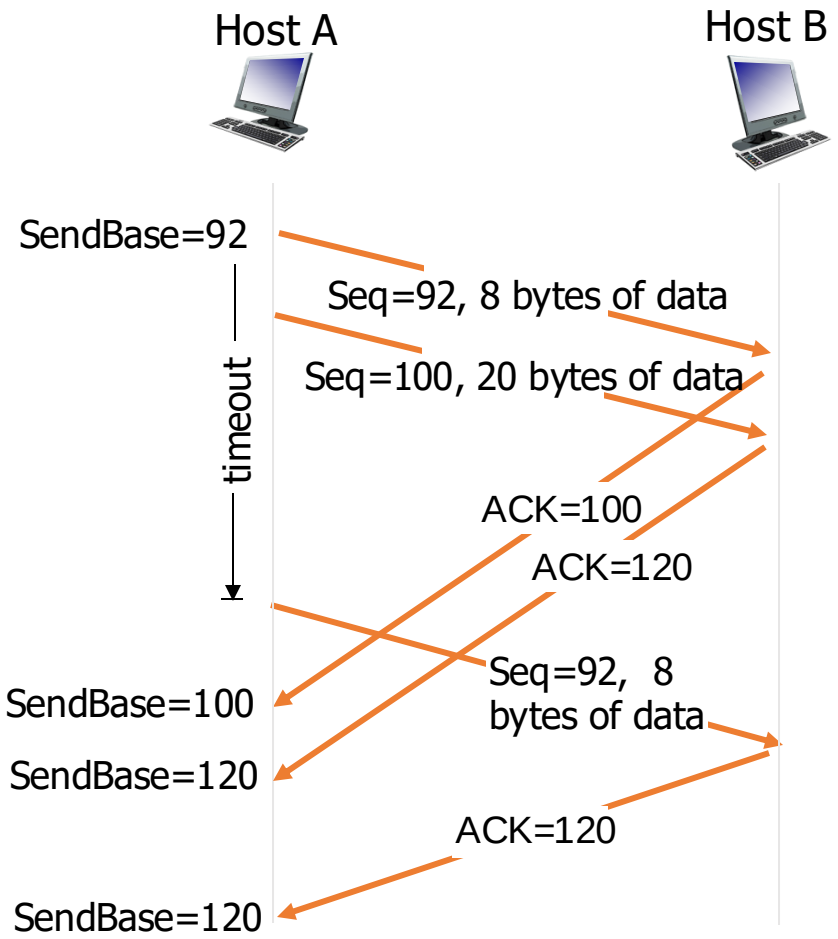


京都大学

TCP: retransmission scenarios



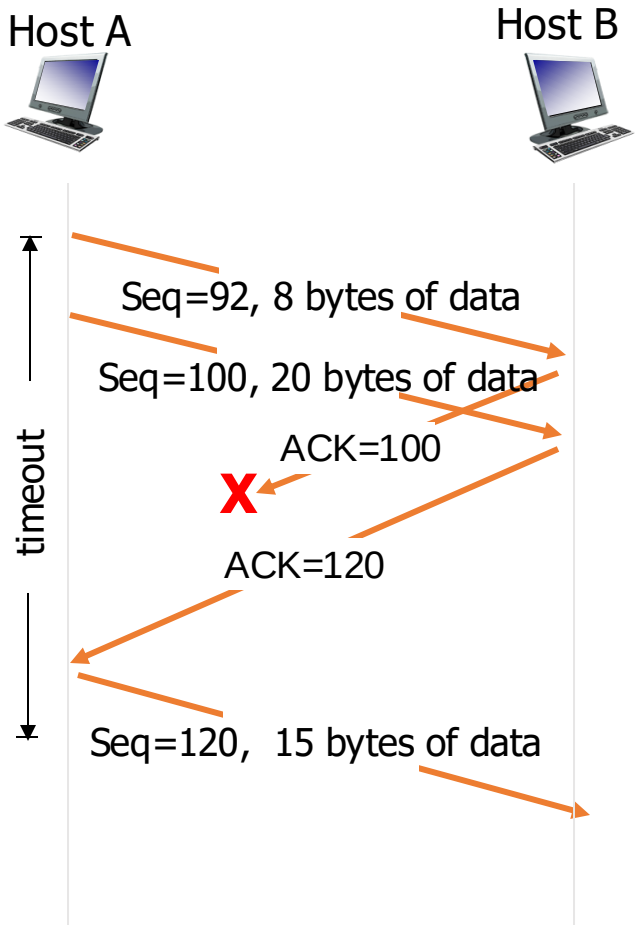
lost ACK scenario



premature timeout

京都大学

TCP: retransmission scenarios



cumulative ACK

京都大学

TCP fast retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

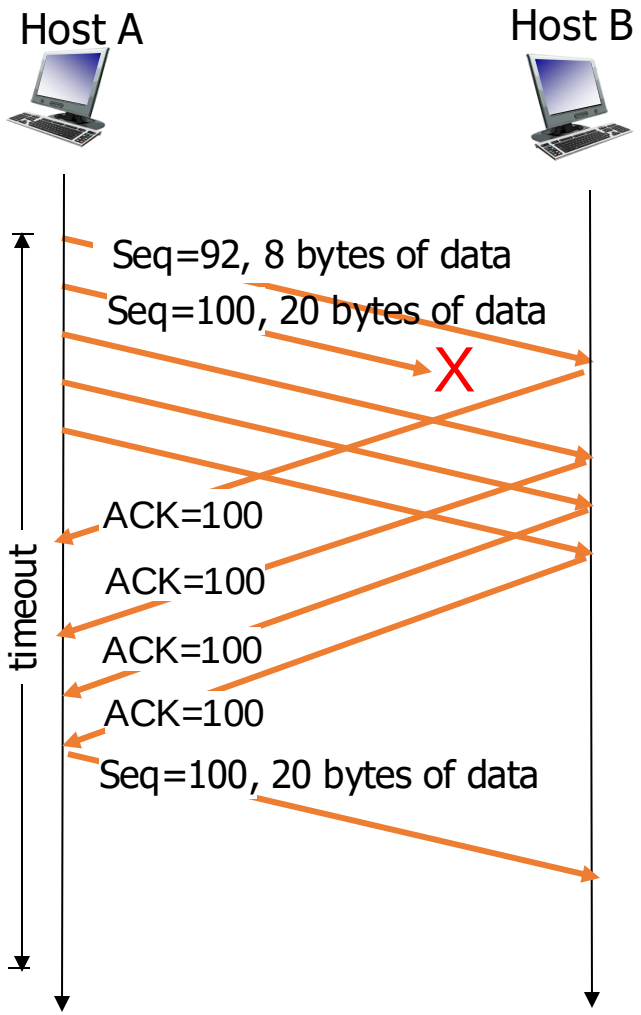
TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq number

- likely that unacked segment lost, so don't wait for timeout

京都大学

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

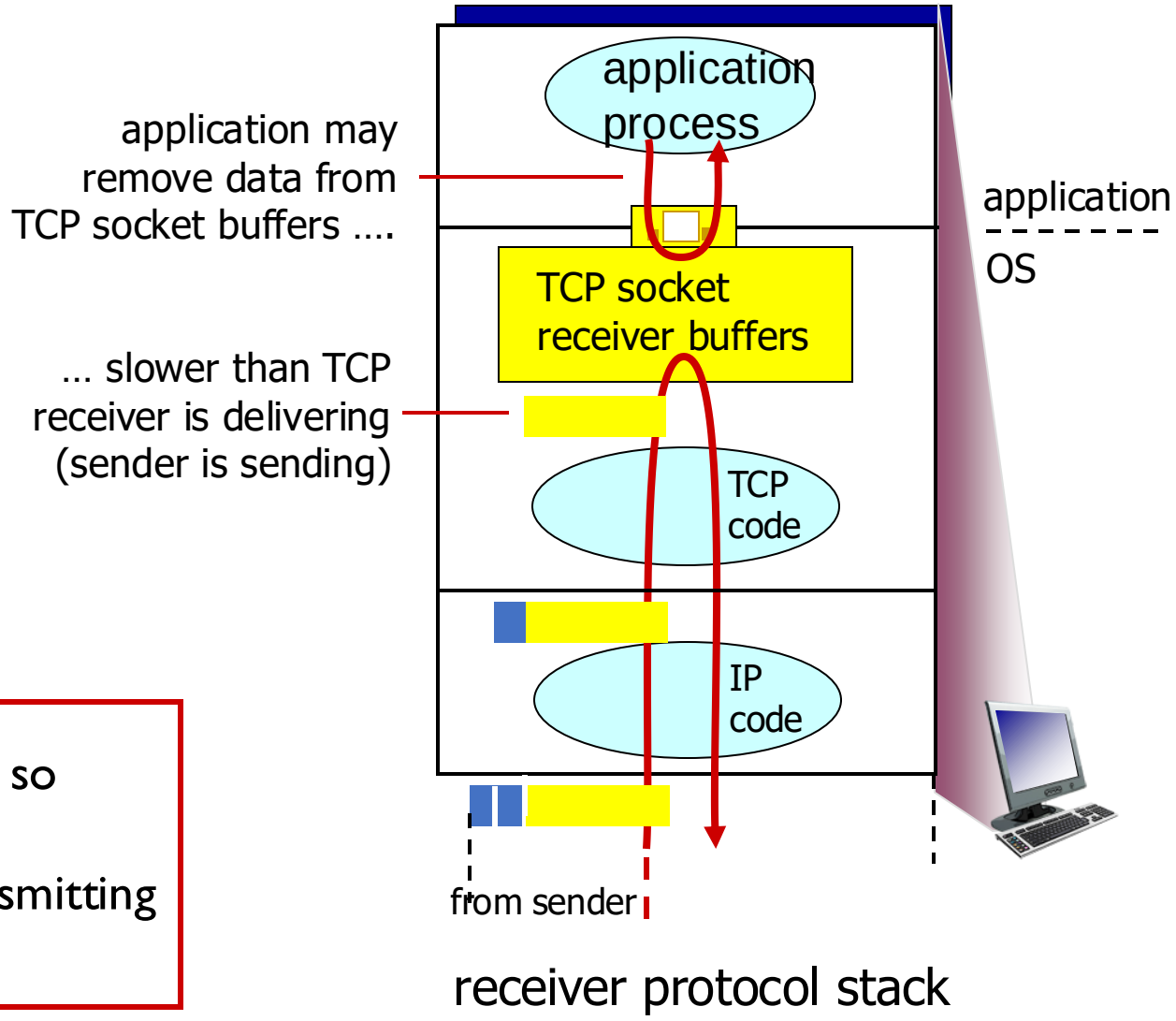
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

京都大学

TCP flow control



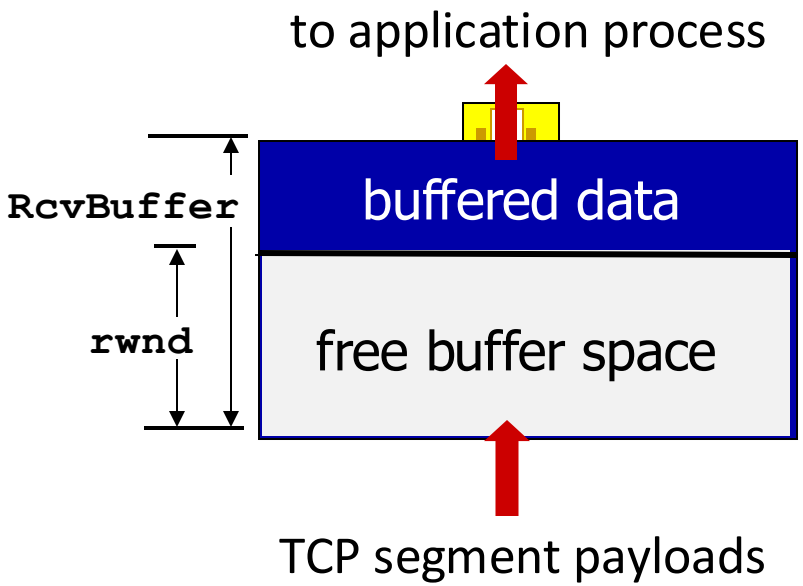
flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

京都大学

TCP flow control

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow

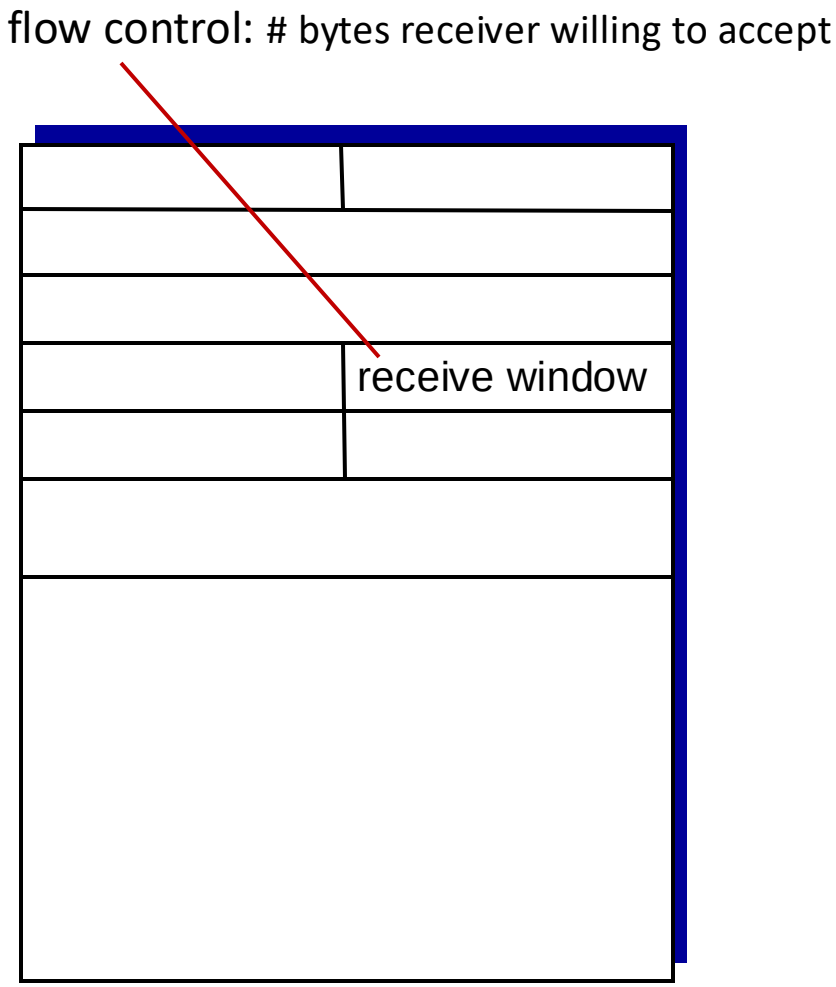


TCP receiver-side buffering

京都大学

TCP flow control

- TCP receiver “advertises” free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed (“in-flight”) data to received **`rwnd`**
- guarantees receive buffer will not overflow



TCP segment format

京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

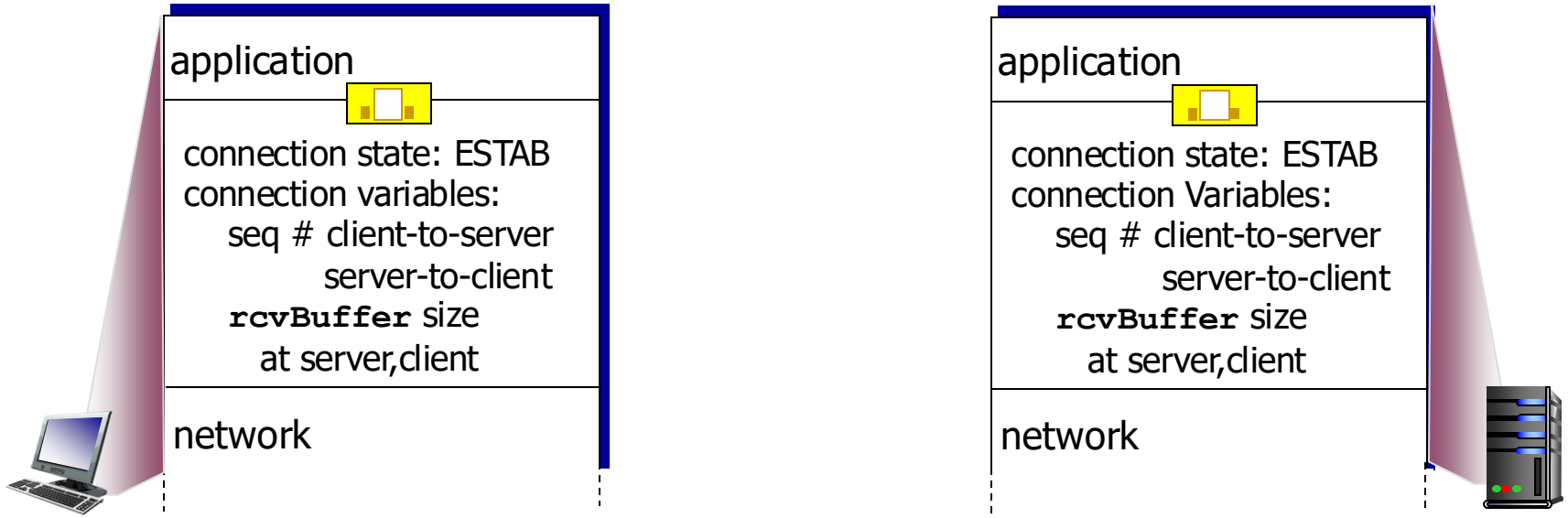
3.7 TCP congestion control

京都大学

Connection Management

before exchanging data, sender/receiver “handshake”:

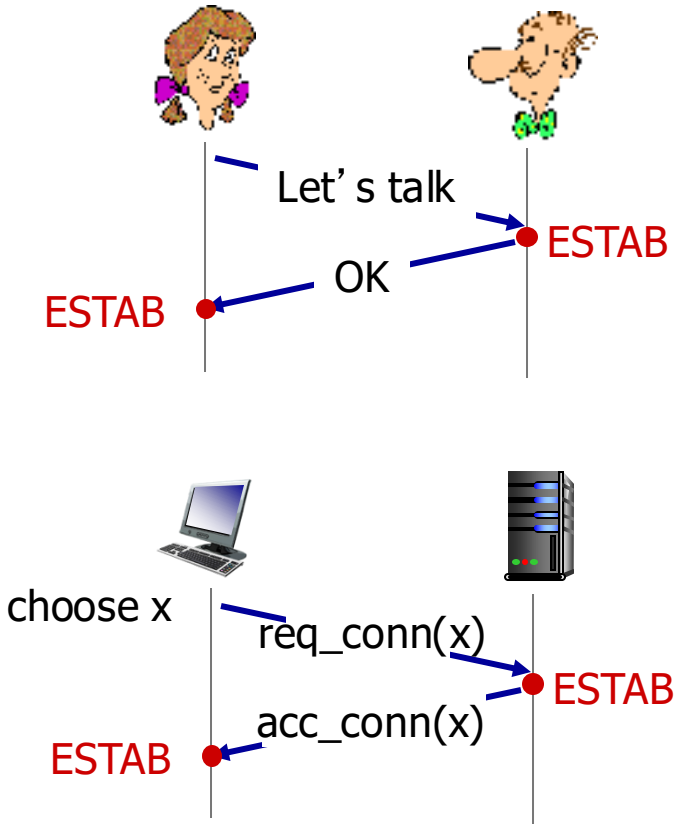
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters



京都大学

Agreeing to establish a connection

2-way handshake:

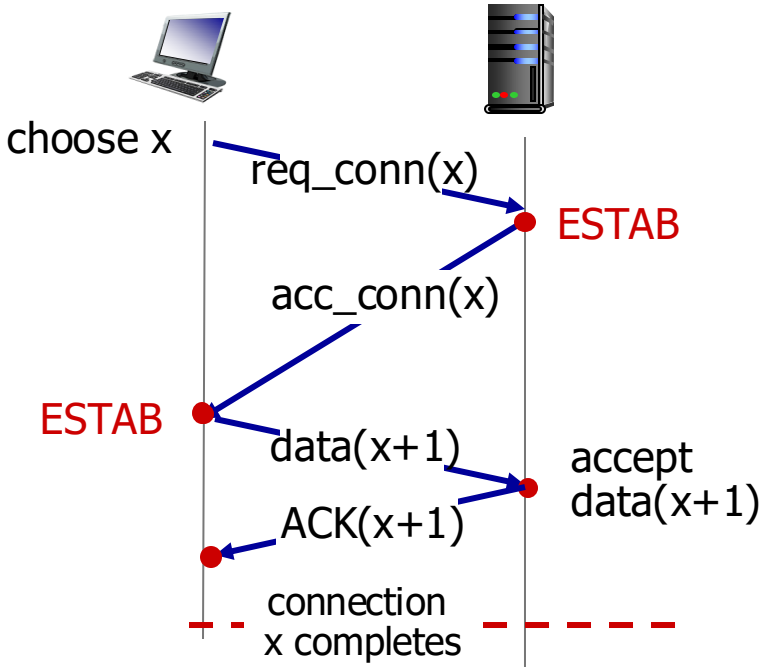


Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

京都大学

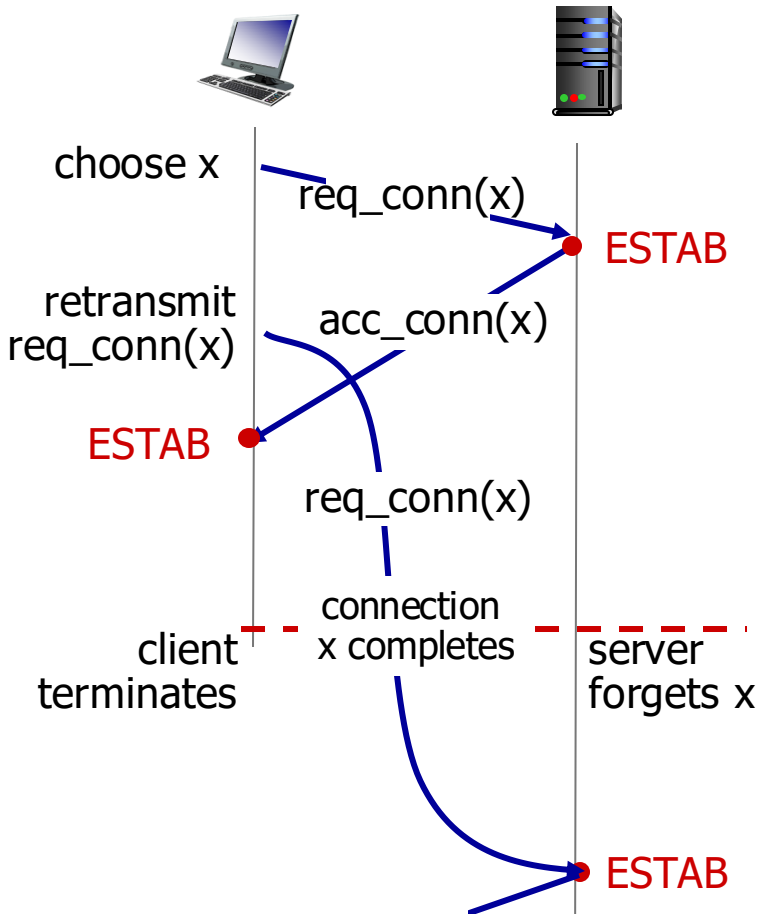
2-way handshake scenarios



No problem!

京都大学

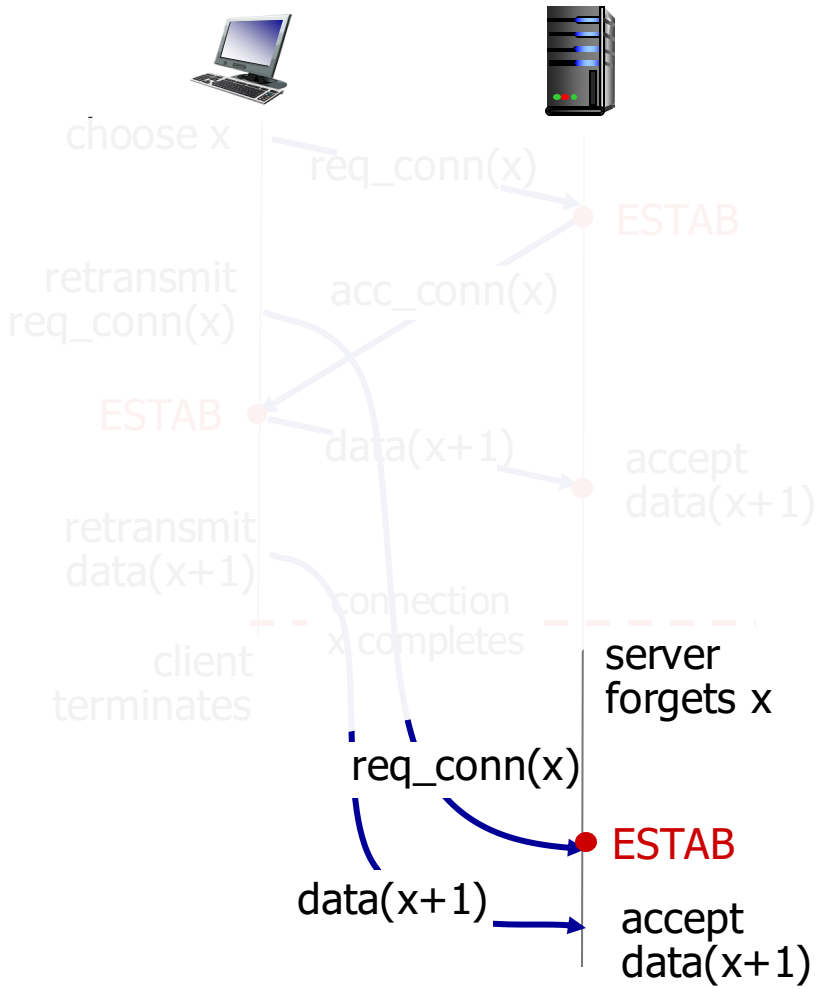
2-way handshake scenarios



Problem: half open connection! (no client)

京都大学

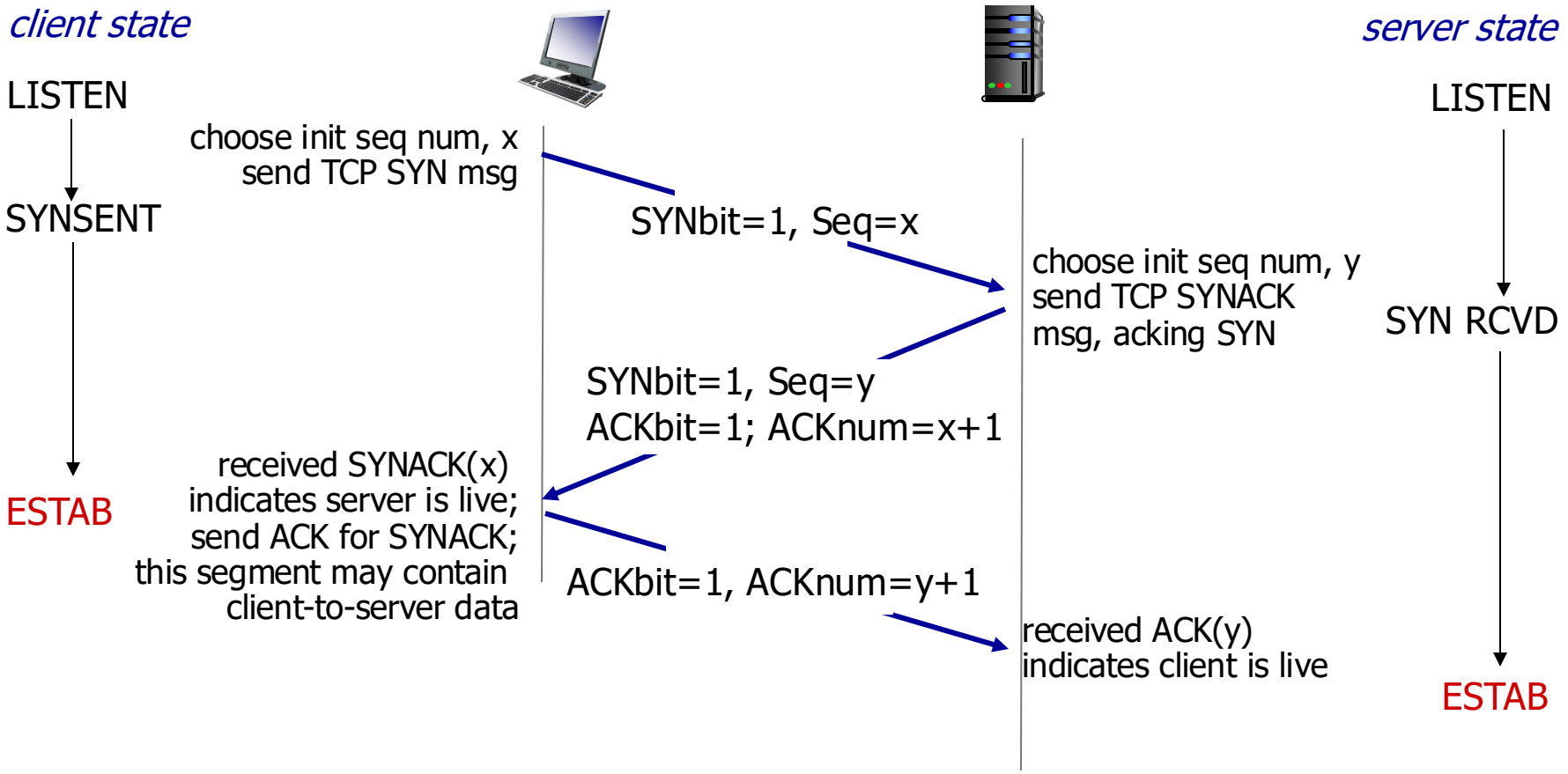
2-way handshake scenarios



Problem: duplicate data accepted!

京都大学

TCP 3-way handshake



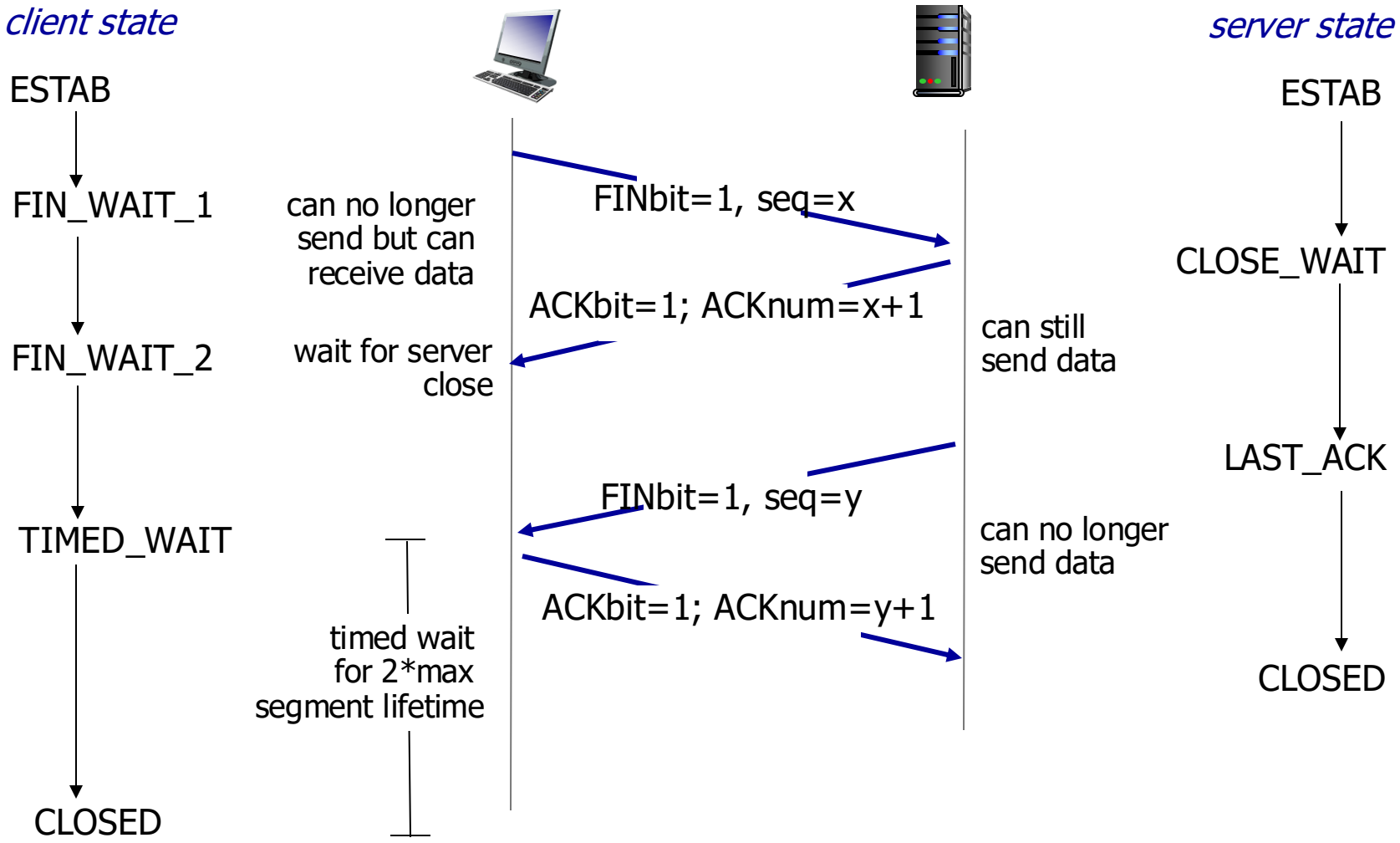
京都大学

TCP: closing a connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN

京都大学

TCP: closing a connection



京都大学