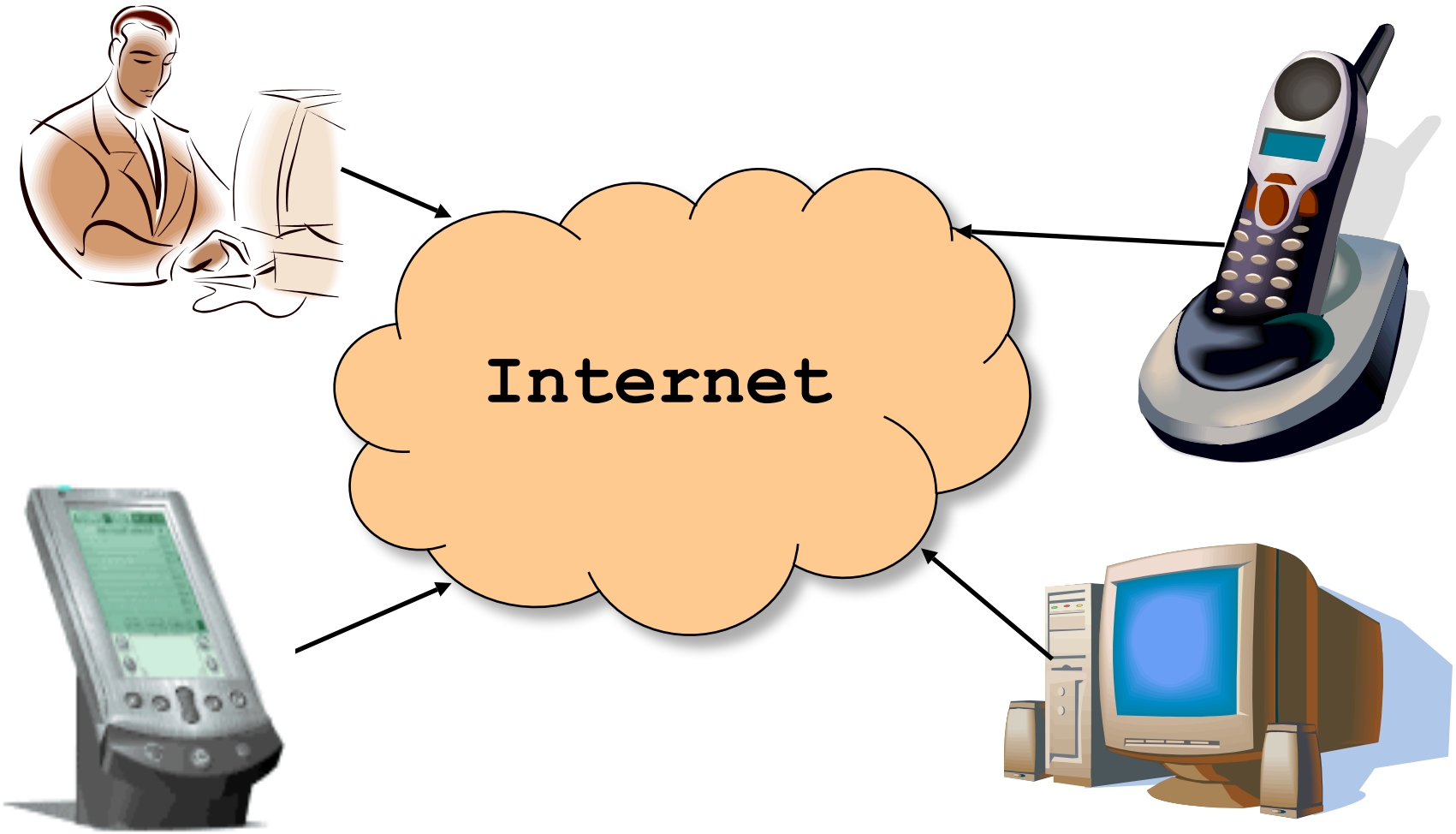
Συστήματα Πελάτη-Εξυπηρετητή στο Διαδίκτυο

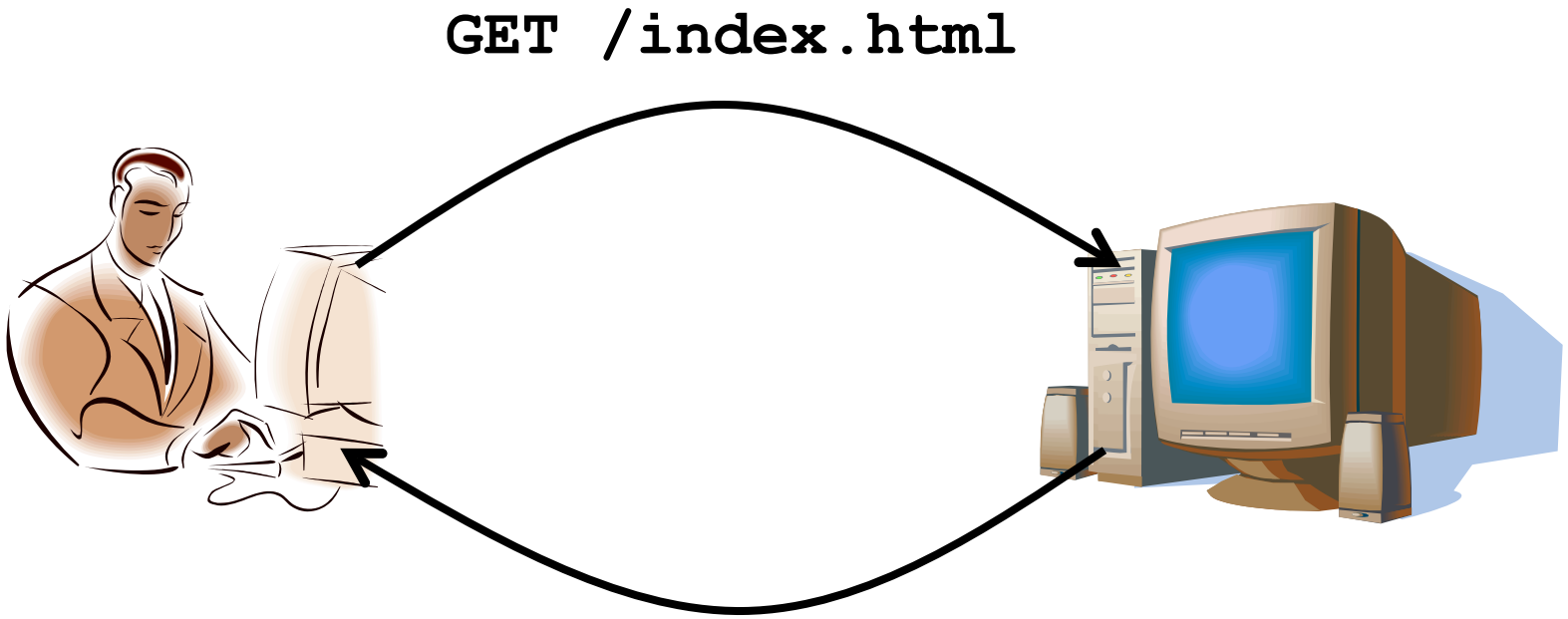
End System: Computer on the 'Net



Also known as a "host"…

Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server

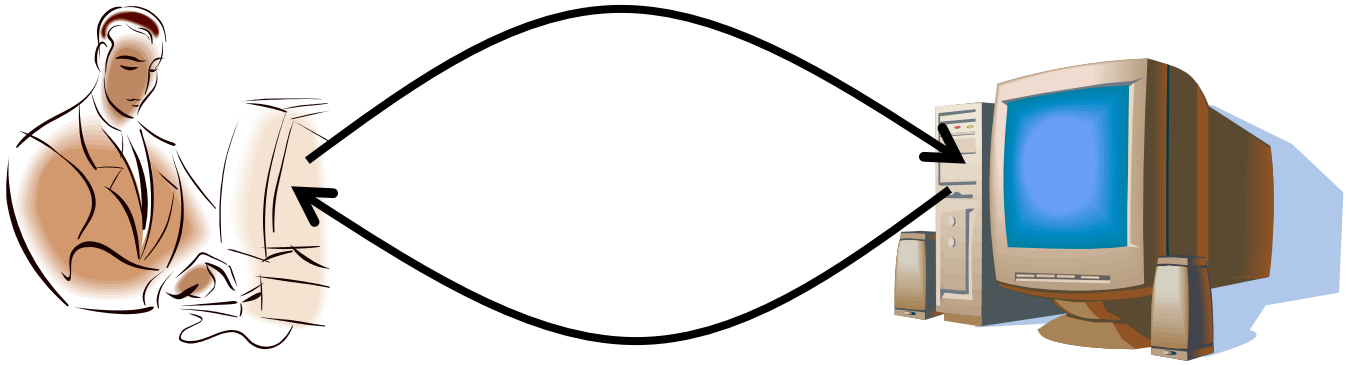


Clients Are Not Necessarily Human

- Example: Web crawler (or spider)
  - Automated client program
  - Tries to discover & download many Web pages
  - Forms the basis of search engines like Google
- Spider client
  - Start with a base list of popular Web sites
  - Download the Web pages
  - Parse the HTML files to extract hypertext links
  - Download these Web pages, too
  - And repeat, and repeat, and repeat…

Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address

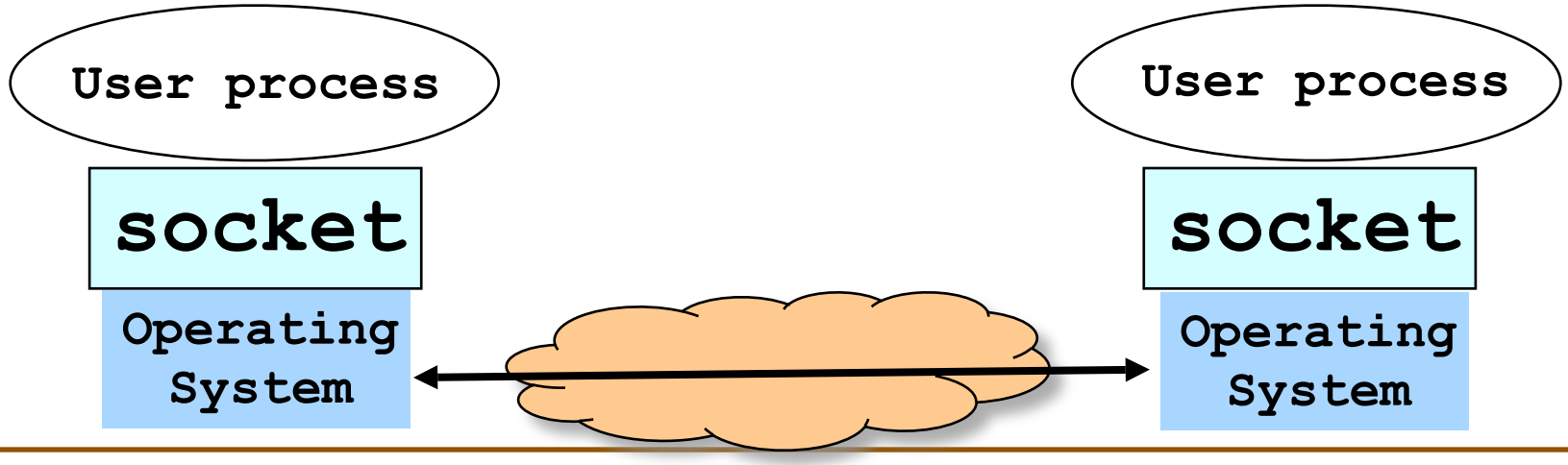


Client and Server Processes

- Program vs. process
  - Program: collection of code
  - Process: a running program on a host
- Communication between processes
  - Same end-host: inter-process communication
    - Governed by the operating system on the end host
  - Different end hosts: exchanging messages
    - Governed by the network protocols
- Client and server processes
  - Client process: process that initiates communication
  - Server process: process that waits to be contacted

Socket: End Point of Communication

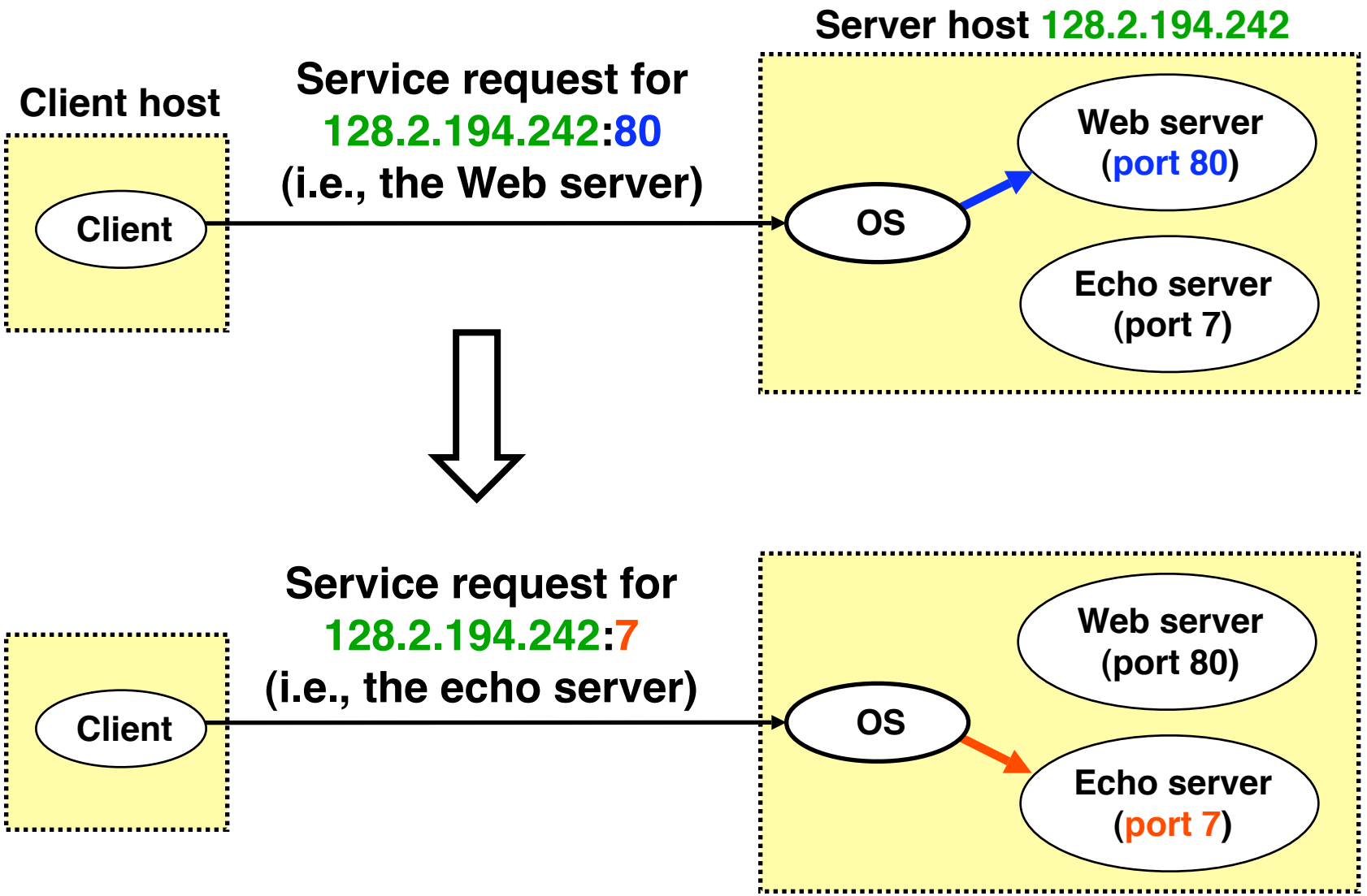
- Sending message from one process to another
  - Message must traverse the underlying network
- Process sends and receives through a "socket"
  - In essence, the doorway leading in/out of the house
- Socket as an Application Programming Interface
  - Supports the creation of network applications



Identifying the Receiving Process

- Sending process must identify the receiver
  - Name or address of the receiving end host
  - Identifier that specifies the receiving process
- Receiving host
  - Destination address that uniquely identifies the host
  - An IP address is a 32-bit quantity
- Receiving process
  - Host may be running many different processes
  - Destination port that uniquely identifies the socket
  - A port number is a 16-bit quantity

Using Ports to Identify Services



Knowing What Port Number To Use

- Popular applications have well-known ports
  - E.g., port 80 for Web and port 25 for e-mail
  - Well-known ports listed at http://www.iana.org
- Well-known vs. ephemeral ports
  - Server has a well-known port (e.g., port 80)
    - Between 0 and 1023
  - Client picks an unused ephemeral (i.e., temporary) port
    - Between 1024 and 65535
- Uniquely identifying the traffic between the hosts
  - Two IP addresses and two port numbers
  - Underlying transport protocol (e.g., TCP or UDP)

Delivering the Data: Division of Labor

- Network
  - Deliver data packet to the destination host
  - Based on the destination IP address
- Operating system
  - Deliver data to the destination socket
  - Based on the protocol and destination port #
- Application
  - Read data from the socket
  - Interpret the data (e.g., render a Web page)

UNIX Socket API

- 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 represented by an integer file descriptor
- System calls for sockets
  - Client: create, connect, write, read, close
  - Server: create, bind, listen, accept, read, write, close

Typical Client Program

- Prepare to communicate
  - Create a socket
  - Determine server address and port number
  - Initiate the connection to the server
- Exchange data with the server
  - Write data to the socket
  - Read data from the socket
  - Do stuff with the data (e.g., render a Web page)
- Close the socket

Creating a Socket: socket()

- Operation to create a socket
  - *int socket(int domain, int type, int protocol)*
  - Returns a descriptor (or handle) for the socket
  - Originally designed to support any protocol suite
- *domain*: protocol family
  - PF_INET for the Internet
  - PF_PACKET: direct access to network interface (bypass TPC/IP)
- *type*: semantics of the communication
  - SOCK_STREAM: reliable byte stream
  - SOCK_DGRAM: message-oriented service
- *protocol*: specific protocol
  - UNSPEC: unspecified
  - (PF_INET and SOCK_STREAM already implies TCP)

Connecting the Socket to the Server

- Translating the server's name to an address
  - *struct hostent *gethostbyname(char *name)*
  - Argument: the name of the host (e.g., "www.cnn.com")
  - Returns a structure that includes the host address
- Identifying the service's port number
  - *struct servent *getservbyname(char *name, char *proto)*
  - Arguments: service (e.g., "ftp") and protocol (e.g., "tcp")
- Establishing the connection
  - *int connect(int socket, struct sockaddr *server_address, int addrlen)*
  - Arguments: socket descriptor, server address, and address size
  - Returns 0 on success, and -1 if an error occurs

Sending and Receiving Data

- Sending data
  - *int write(int sockfd, void *buf, int len, int flags)*
  - Arguments: socket descriptor, pointer to buffer of data to send, length of the buffer, flags controlling details
  - Returns the number of characters written, and -1 on error
- Receiving data
  - *int read(int sockfd, void *buf, int len, int flags)*
  - Arguments: socket descriptor, pointer to buffer to place the data, size of the buffer, flags controlling details
  - Returns the number of characters read (where 0 implies "end of file"), and -1 on error
- Closing the socket
  - *int close(int sockfd)*

Byte Ordering: Little and Big Endian

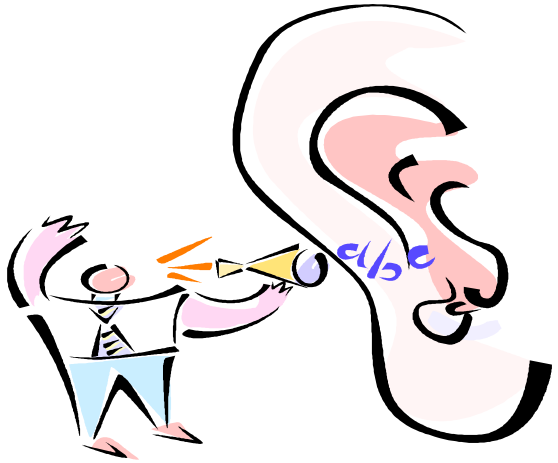
- Hosts differ in how they store data
  - E.g., four-byte number (byte3, byte2, byte1, byte0)
- Little endian ("little end comes first") ← Intel PCs!!!
  - Low-order byte stored at the lowest memory location
  - Byte0, byte1, byte2, byte3
- Big endian ("big end comes first")
  - High-order byte stored at lowest memory location
  - Byte3, byte2, byte1, byte 0
- IP is big endian (aka "network byte order")
  - Use htons() and htonl() to convert to network byte order
  - Use ntohs() and ntohl() to convert to host order

Why Can't Sockets Hide These Details?

- Dealing with endian differences is tedious
  - Couldn't the socket implementation deal with this
  - … by swapping the bytes as needed?
- No, swapping depends on the data type
  - Two-byte short int: (byte 1, byte 0) vs. (byte 0, byte 1)
  - Four-byte long int: (byte 3, byte 2, byte 1, byte 0) vs. (byte 0, byte 1, byte 2, byte 3)
  - String of one-byte charters: (char 0, char 1, char 2, …) in both cases
- Socket layer doesn't know the data types
  - Sees the data as simply a buffer pointer and a length
  - Doesn't have enough information to do the swapping

Servers Differ From Clients

- Passive open
  - Prepare to accept connections
  - … but don't actually establish one
  - … until hearing from a client



- Hearing from multiple clients
  - Allow a backlog of waiting clients
  - ... in case several try to start a connection at once
- Create a socket for each client
  - Upon accepting a new client
  - … create a *new* socket for the communication

Typical Server Program

- Prepare to communicate
  - Create a socket
  - Associate local address and port with the socket
- Wait to hear from a client (passive open)
  - Indicate how many clients-in-waiting to permit
  - Accept an incoming connection from a client
- Exchange data with the client over new socket
  - Receive data from the socket
  - Do stuff to handle the request (e.g., get a file)
  - Send data to the socket
  - Close the socket
- Repeat with the next connection request

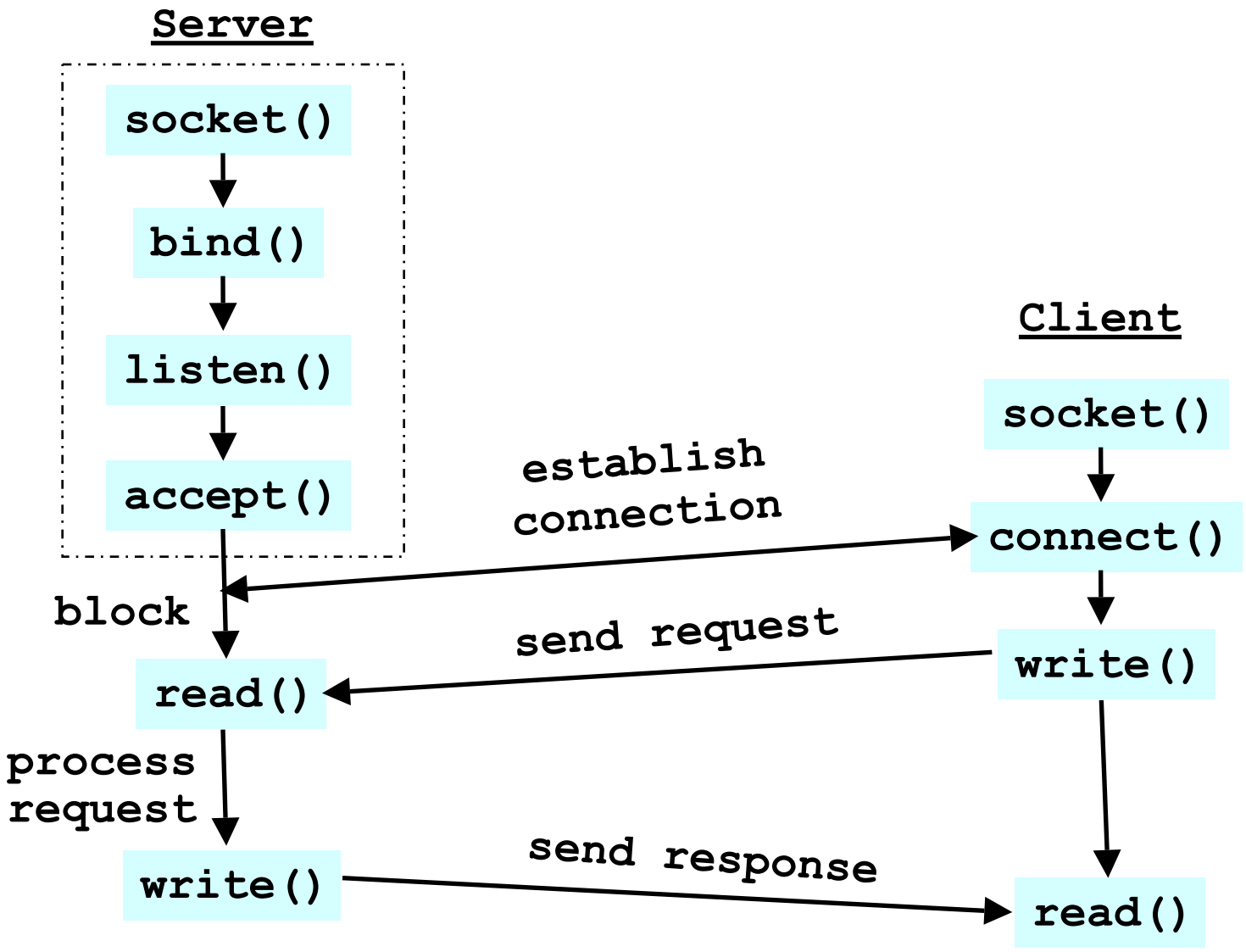
Server Preparing its Socket

- Bind socket to the local address and port number
  - *int bind (int sockfd, struct sockaddr *my_addr, int addrlen)*
  - Arguments: socket descriptor, server address, address length
  - Returns 0 on success, and -1 if an error occurs
- Define how many connections can be pending
  - *int listen(int sockfd, int backlog)*
  - Arguments: socket descriptor and acceptable backlog
  - Returns 0 on success, and -1 on error

Accepting a New Client Connection

- Accept a new connection from a client
  - *int accept(int sockfd, struct sockaddr *addr, int *addrlen)*
  - Arguments: socket descriptor, structure that will provide remote client address and port, and length of the structure
  - Returns *descriptor for a new socket for this connection* (the old socket still exists and corresponds to the passive open, to be used for future invocations of accept)
- Questions
  - What happens if no clients are around?
    - The *accept()* call *blocks* waiting for a client
  - What happens if too many clients are around?
    - Some connection requests don't get through
    - … But, that's okay, because the Internet makes no promises

Putting it All Together

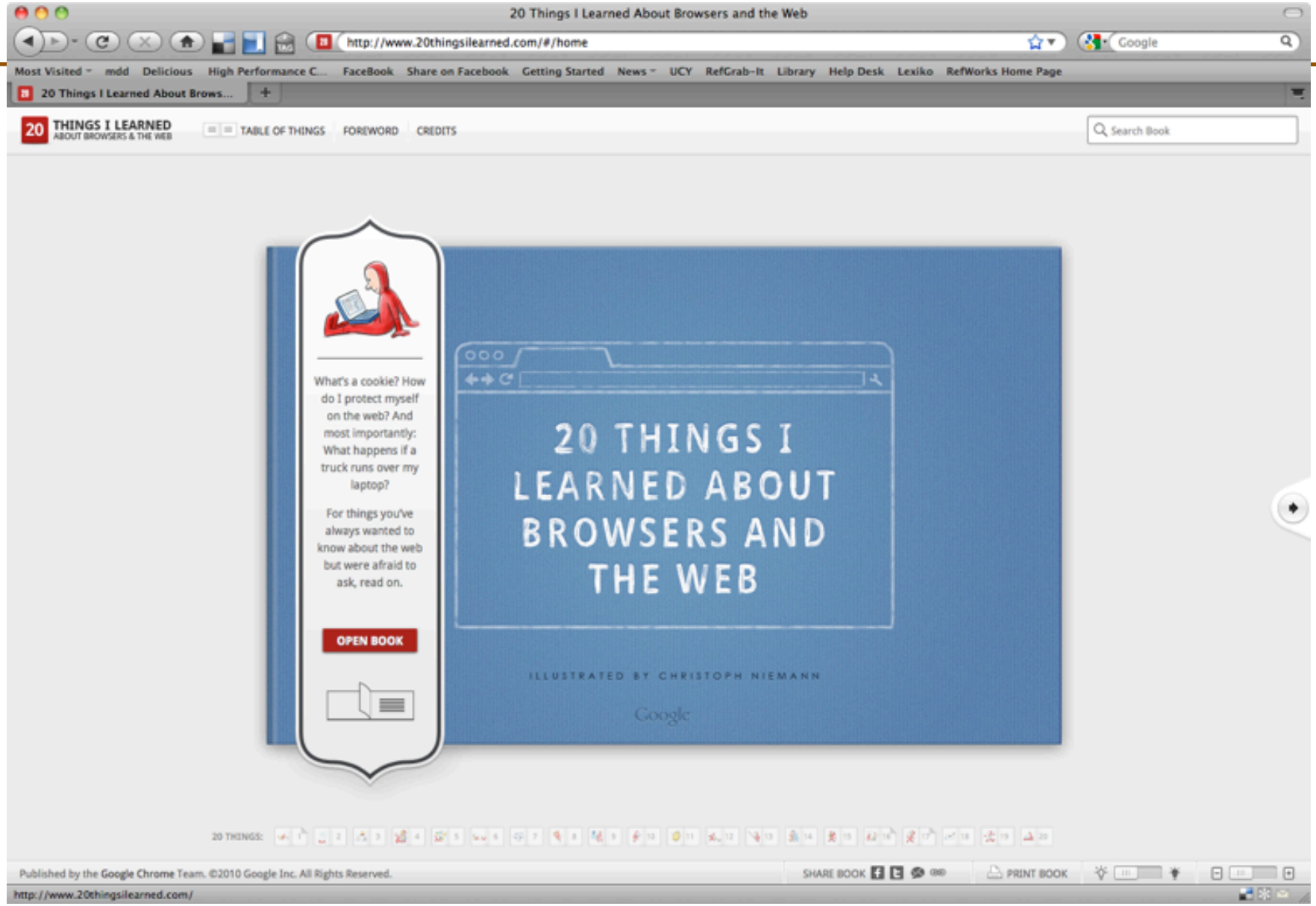


Serving One Request at a Time?

- Serializing requests is inefficient
  - Server can process just one request at a time
  - All other clients must wait until previous one is done
- Need to time share the server machine
  - Alternate between servicing different requests
    - Do a little work on one request, then switch to another
    - Small tasks, like reading HTTP request, locating the associated file, reading the disk, transmitting parts of the response, etc.
  - Or, start a new process to handle each request
    - Allow the operating system to share the CPU across processes
  - Or, some hybrid of these two approaches

Wanna See Real Clients and Servers?

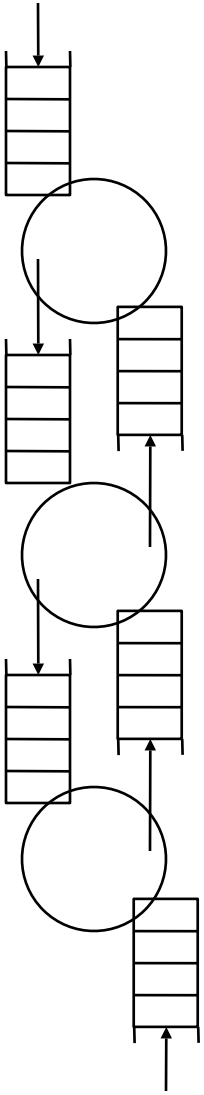
- Apache Web server
  - Open source server first released in 1995
  - Name derives from “a patchy server” ;-)
  - Software available online at http://www.apache.org
- Mozilla Web browser
  - http://www.mozilla.org/developer/
- Sendmail
  - http://www.sendmail.org/
- BIND Domain Name System
  - Client resolver and DNS server
  - http://www.isc.org/index.pl?/sw/bind/
- …


20 Things I Learned About Browsers and the Web
http://www.20thingsilearned.com/#/home
Google
Most Visited
mdd
Delicious
High Performance C...
FaceBook
Share on Facebook
Getting Started
News
UCY
RefGrab-It
Library
Help Desk
Lexiko
RefWorks Home Page
20 Things I Learned About Brows...
20 THINGS I LEARNED ABOUT BROWSERS & THE WEB
TABLE OF THINGS
FOREWORD
CREDITS
Search Book
What's a cookie? How do I protect myself on the web? And most importantly: What happens if a truck runs over my laptop?
For things you've always wanted to know about the web but were afraid to ask, read on.
OPEN BOOK
20 THINGS I LEARNED ABOUT BROWSERS AND THE WEB
ILLUSTRATED BY CHRISTOPH NIEMANN
Google
20 THINGS:
Published by the Google Chrome Team. ©2010 Google Inc. All Rights Reserved.
SHARE BOOK
PRINT BOOK
http://www.20thingsilearned.com/

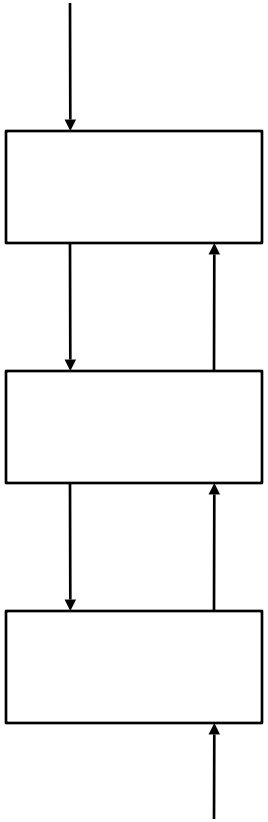
Θέματα Υλοποίησης Πρωτοκόλλων

- Μοντέλο Διεργασίας (process model)
- Διεργασία ανά πρωτόκολλο (process-per-protocol):
  - Κάθε πρωτόκολλο υλοποιείται σαν διαφορετική διεργασία
  - Ένα μήνυμα που διατρέχει την στοίβα των πρωτοκόλλων, περνάει από μια διεργασία πρωτοκόλλου στην άλλη.
- Διεργασία ανά μήνυμα (process-per-message):
  - Κάθε πρωτόκολλο θεωρείται σαν “στατικός” κώδικας
  - Κάθε μήνυμα αντιστοιχίζεται σε μια διαφορετική διεργασία.
  - Η “διαδρομή” της στοίβας των πρωτοκόλλων γίνεται με κλήσεις διαδικασιών.

Θέματα Υλοποίησης Πρωτοκόλλων



Process-per-protocol



Process-per-message

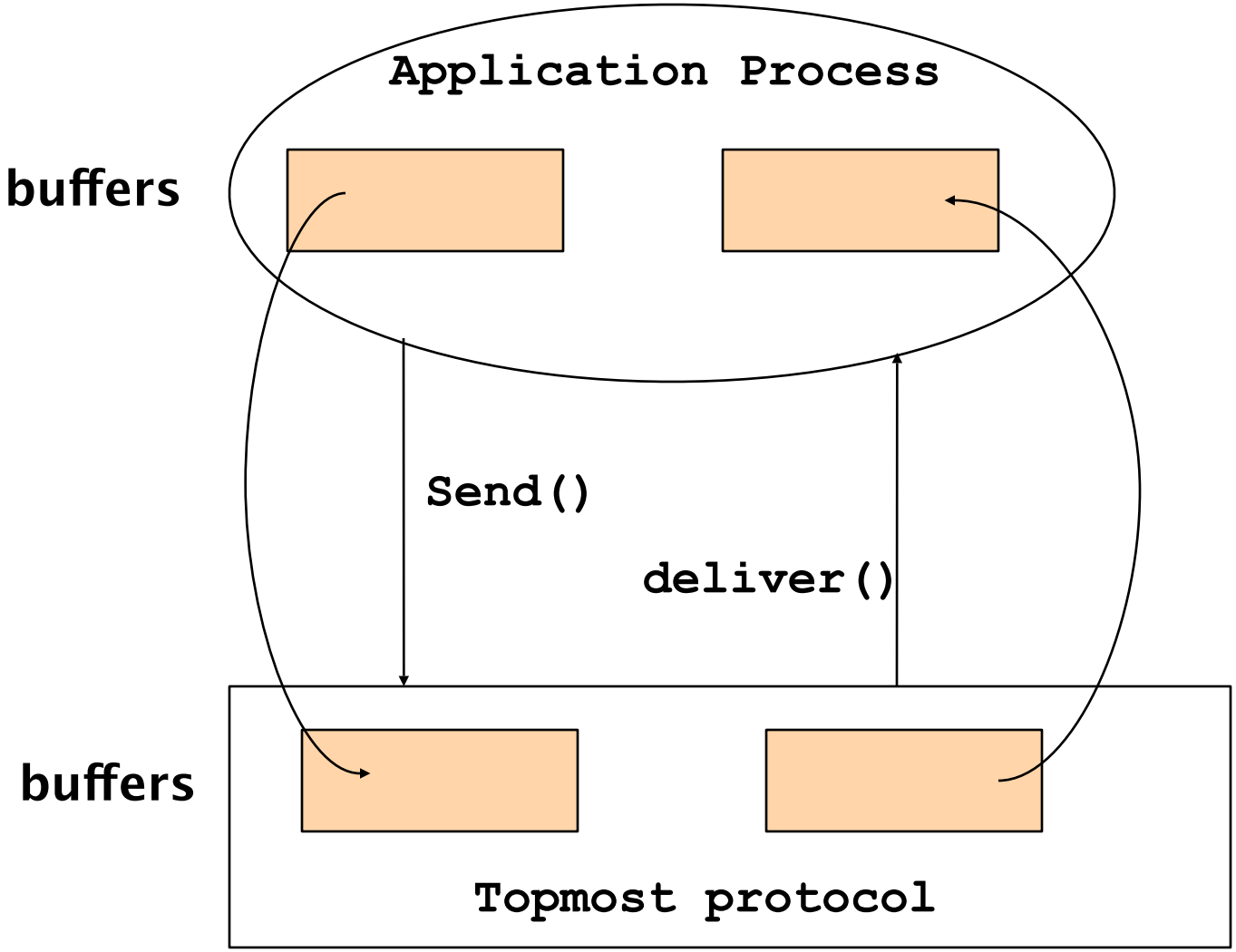
Θέματα Υλοποίησης Πρωτοκόλλων

- Process-per-protocol model:
  - Ακούγεται ευκολότερο στην υλοποίηση
  - Λιγότερο αποδοτικό από το process-per-message model: η κλήση διαδικασίας είναι μια τάξη μεγέθους πιο γρήγορη από το context switch
- Στην πράξη, τα μηνύματα “κατεβαίνουν” την στοίβα των πρωτοκόλλων με μια σειρά κλήσεων διαδικασίας send και “ανεβαίνουν” τη στοίβα των πρωτοκόλλων με μια σειρά κλήσεων διαδικασίας deliver.
- Η παραλαβή μηνυμάτων από τις εφαρμογές γίνεται με την κλήση λειτουργίας receive, η οποία προκαλεί context switch.

Θέματα Υλοποίησης Πρωτοκόλλων

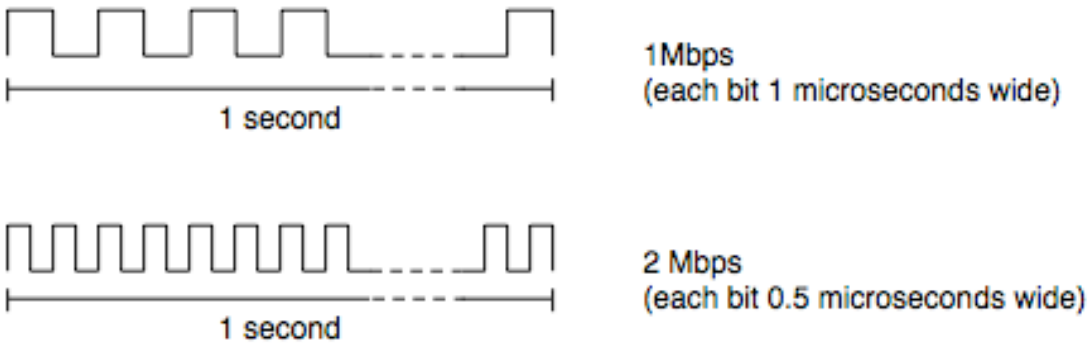
- Σε εξερχόμενα μηνύματα, το πρωτόκολλο μιας διαστρωμάτωσης εκτελεί λειτουργία send προς το πρωτόκολλο της χαμηλότερης διαστρωμάτωσης:
  - Η λειτουργία αυτή μπορεί εύκολα να υλοποιηθεί με κλήση διαδικασίας, οπότε δεν χρειάζεται αλλαγή συγκειμένου (context switch)
- Για τη διαχείριση εισερχομένων μηνυμάτων, το πρωτόκολλο μιας διαστρωμάτωσης εκτελεί λειτουργία receive και μπλοκάρει, περιμένοντας από το πρωτόκολλο της χαμηλότερης διαστρωμάτωσης να παραδόσει κάποιο εισερχόμενο μήνυμα
  - Το μπλοκάρισμα συνεπάγεται αλλαγή συγκειμένου
  - Για την αποφυγή του κόστους αυτού στις υλοποιήσεις των πρωτοκόλλων, αντί της receive γίνεται χρήση λειτουργίας *deliver* (με κλήση διαδικασίας από πρωτόκολλο χαμηλότερης σε πρωτόκολλο υψηλότερης διαστρωμάτωσης)

Καταχωρητές μηνυμάτων



Μετρικές Επίδοσης - performance metrics

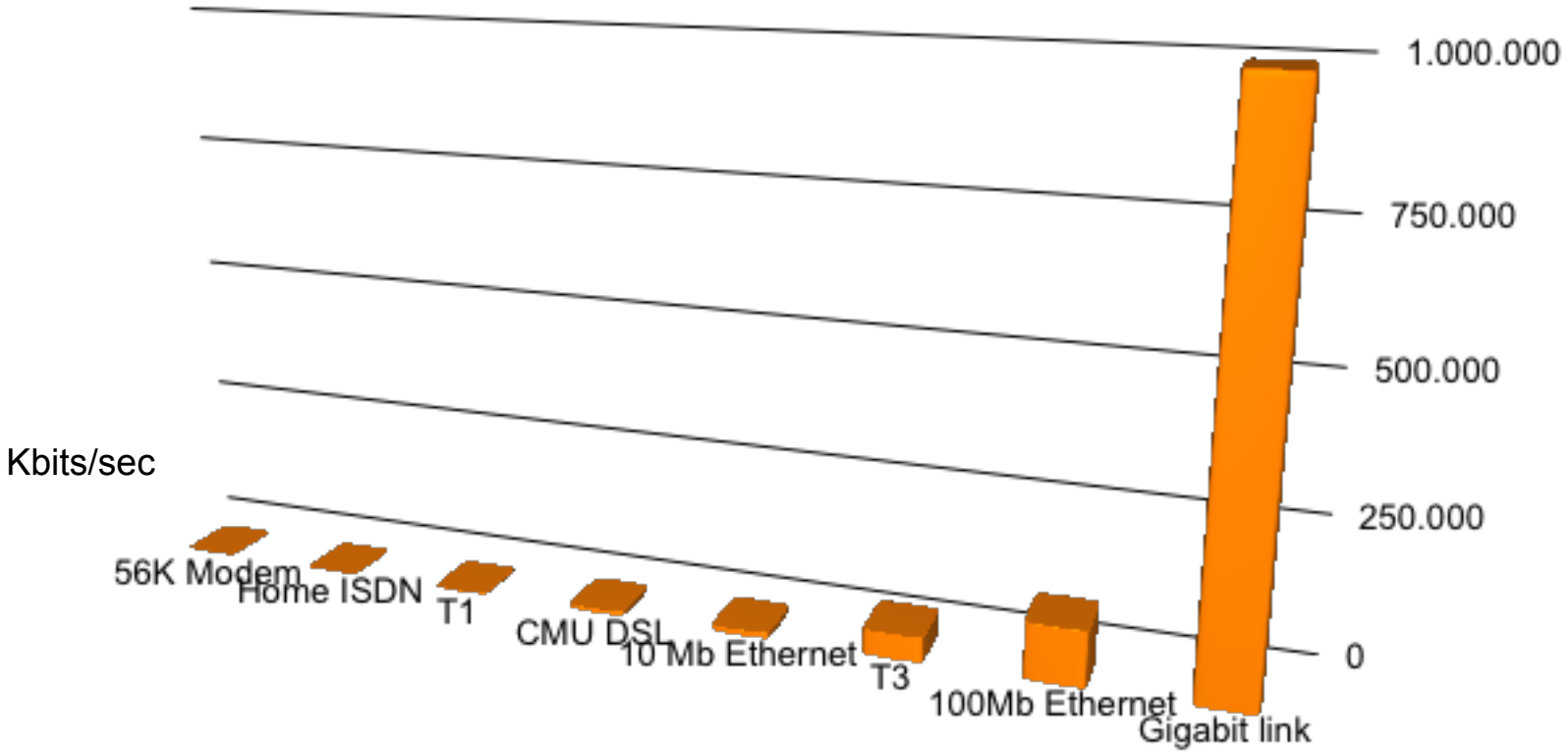
- Bandwidth (throughput) - ζωνικό εύρος/εύρος ζώνης
  - Ποσότητα μεταφερόμενων δεδομένων ανά χρονική στιγμή
  - Link vs end-to-end
  - KB = 2^{10} bytes
  - Mbps = 10^6 bits per second



Bandwidth of a Truck

- Semi Tractor-Trailer 30'L x 10'H x 8'W ≈ 2500 ft^3
- **DVDs (Digital Videodisks)**
  - @5 GB each, 2000 GB (2 terabytes)/ ft^3
  - Semi holds 5 million GB = 5 petabytes (enough to store every book ever published)
- Pittsburgh - San Francisco ≈ 3000 miles
  - @ 50 miles/hour = 60 hours ≈ 200,000 seconds
  - Bandwidth ≈ 25 GB / second ≈ 200 gigabits/sec
    200 times the bandwidth of gigabit Ethernet!
- Problem: latency = 60 hours

Bandwidth by Technology

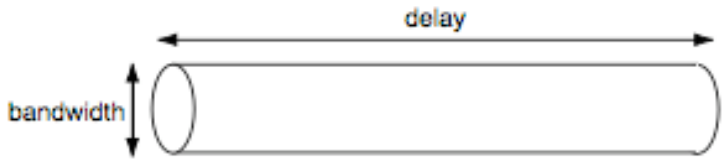


Μετρικές Επίδοσης - performance metrics

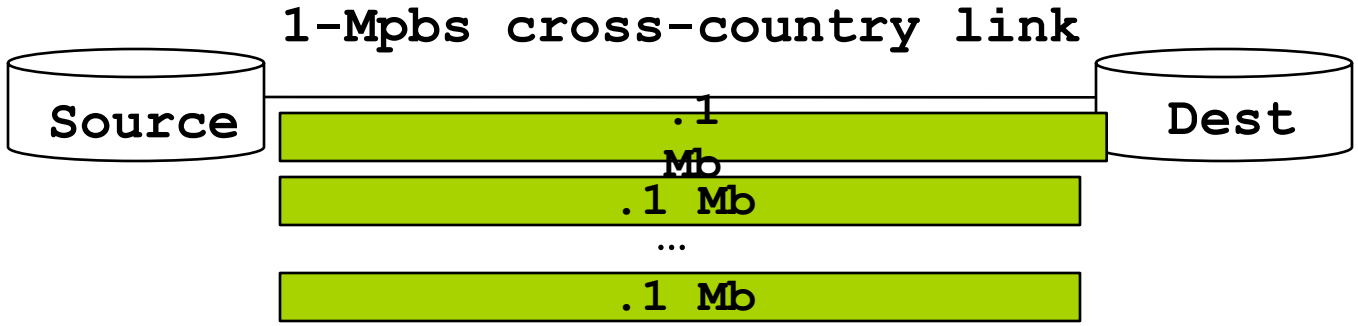
- Latency - χρονική υστέρηση
  - Χρόνος που απαιτείται για αποστολή μηνύματος (transmission time) από σημείο A σε σημείο B στο δίκτυο
  - Μονής κατεύθυνσης ή μετ' επιστροφή (RTT)
  - Υπολογισμός:
    - Latency = Propagation + Transmit + Queue
    - Propagation = Distance / SpeedOfLight
    - Transmit = Size / Bandwidth
  - Speed of light:
    - 3.0 x 10^8 meters/second στο κενό
    - 2.3 x 10^8 meters/second σε καλώδιο
    - 2.0 x 10^8 meters/second σε οπτική ίνα
- Ο χρόνος μετάδοσης ενός μικρού μηνύματος σε τοπικό δίκτυο είναι συνήθως κάτω του 1ms – 1000 πιό αργός από τον χρόνο κλήσης μιας λειτουργίας κάποιας διεργασίας που βρίσκεται φορτωμένη στην κεντρική μνήμη ενός Η/Υ.
- Ωστόσο, ο χρόνος αυτός είναι συγκρίσιμος (ή και πιό μικρός) από τον χρόνο ανάγνωσης του μηνύματος από έναν σκληρό δίκο (λόγω υστέρησης και ρυθμού μετάδοσης).

Υστέρηση vs Εύρος Ζώνης

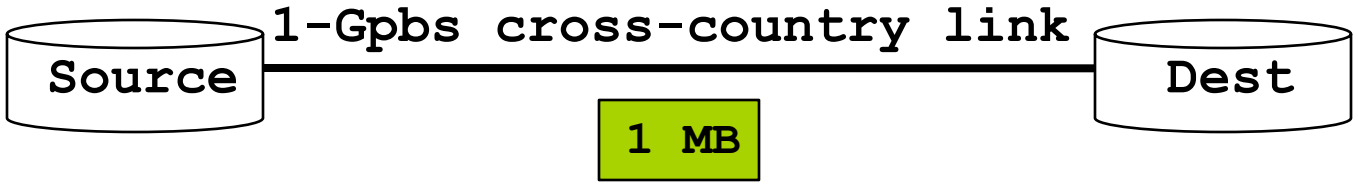
- Σχετική σημασία εύρους ζώνης και υστέρησης
  - Σε μικρά μηνύματα (1 byte): 1ms vs 100ms dominates 1Mbps vs 100 Mbps
  - Σε μεγάλα μηνύματα (25 MB): Mbps vs 100 Mbps dominates 1ms vs 100 ms
- Delay x Bandwidth = πόσα bits πρέπει να στείλει ο αποστολέας πριν την άφιξη του πρώτου bit στον παραλήπτη



Υστέρηση vs Εύρος Ζώνης



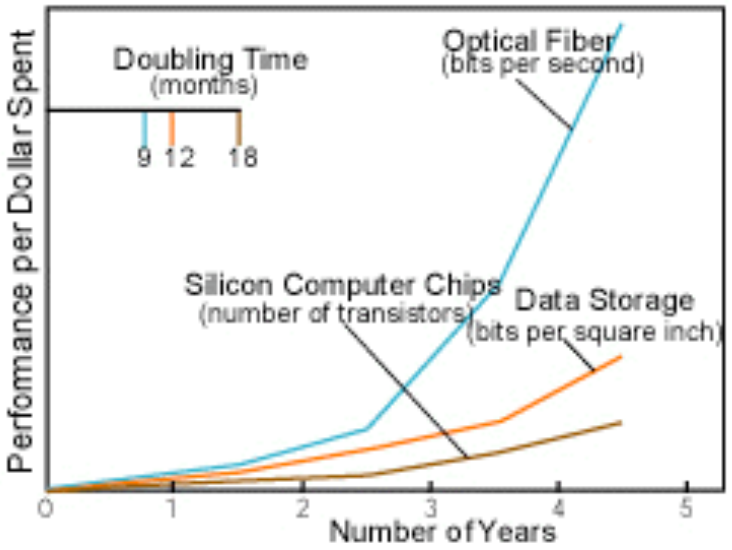
Takes 80 RTT to transmit 1 MB file;
1.25% of the file is sent during each RTT



A 1 MB file does not even fill 1RTT worth
1 RTT worth of the Gbps link

Network Exponentials

- Network vs. computer performance
  - Computer speed doubles every 18 months
  - Network speed doubles every 9 months
  - Difference = order of magnitude per 5 years
- 1986 to 2000
  - Computers: x 500
  - Networks: x 340,000
- 2001 to 2010
  - Computers: x 60
  - Networks: x 4000



Moore's Law vs. storage improvements vs. optical improvements. Graph from **Scientific American** (Jan-2001) by Cleo Vilett, source Vined Khoslan, Kleiner, Caufield and Perkins.