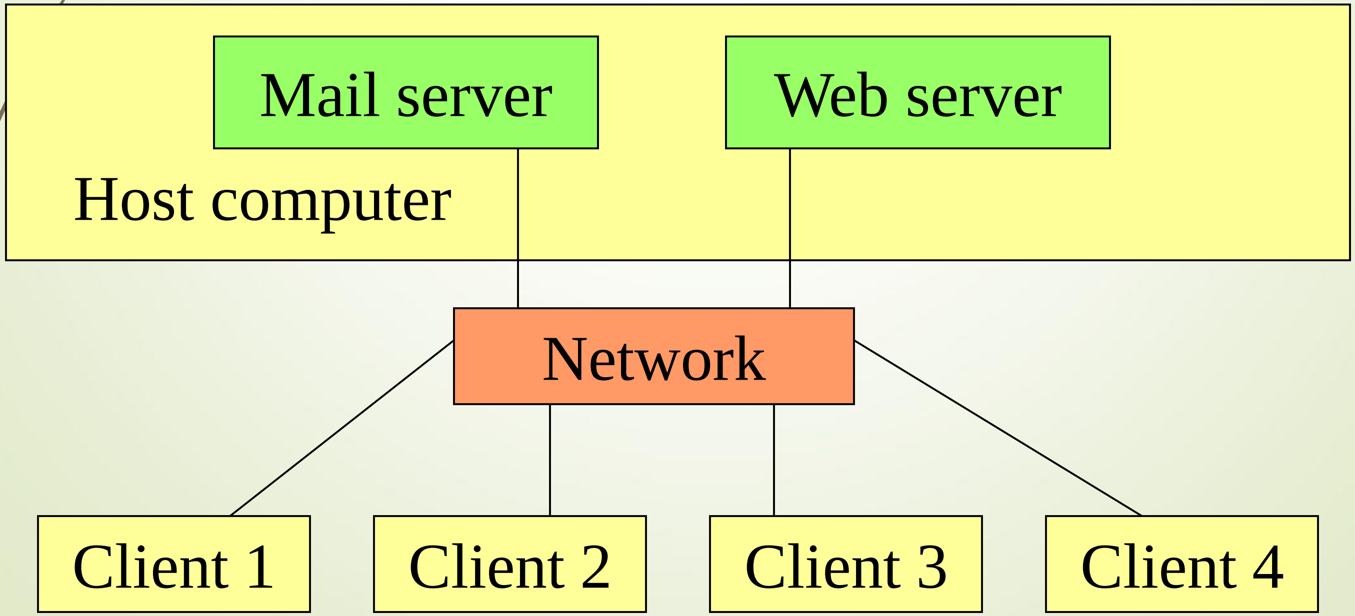
Networks and Client/Server Applications

Basics of Client/Server

- One host computer can have several servers
- Several clients can connect to a server



Network Addresses

- Every computer on a network has an address
- Every Internet address has two components:
  - an *IP name* (such as "lambert")
  - an *IP address* (such as "129.21.38.145")
- IP stands for Internet Protocol

Ports

- A *port* is a software abstraction of a physical space through which a client and a server can send messages

- Operating systems have several dedicated system ports and several free ports

Ports

- Ports are known by numbers
- For example, port 13 usually returns the day and time on the host computer
- Several processes can use the same port at the same time

Sockets

- A *socket* is a software abstraction that provides a communication link between a single server process and a single client process

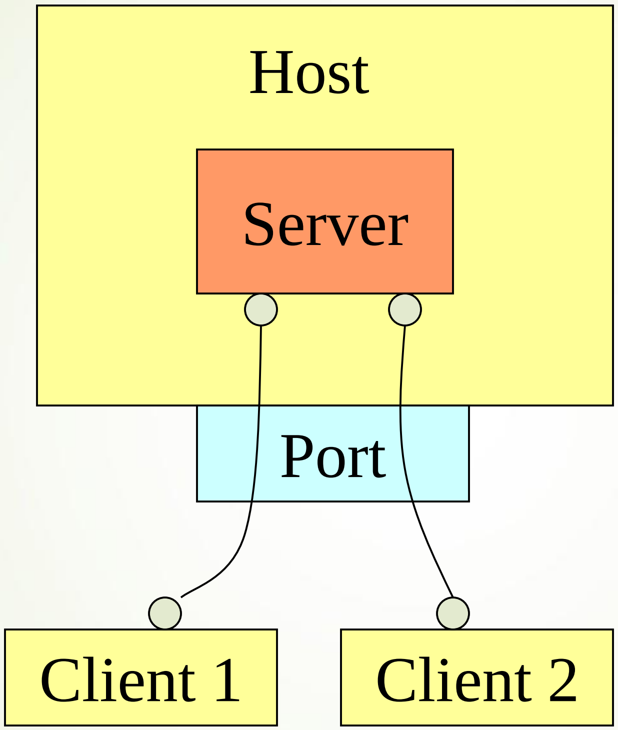
- Several sockets can be created on the same port

Sockets

- Two things are required to create a socket:
  - a valid IP address
  - a port number

- Client and server then use input and output operations to send messages through the socket

The Basic Setup

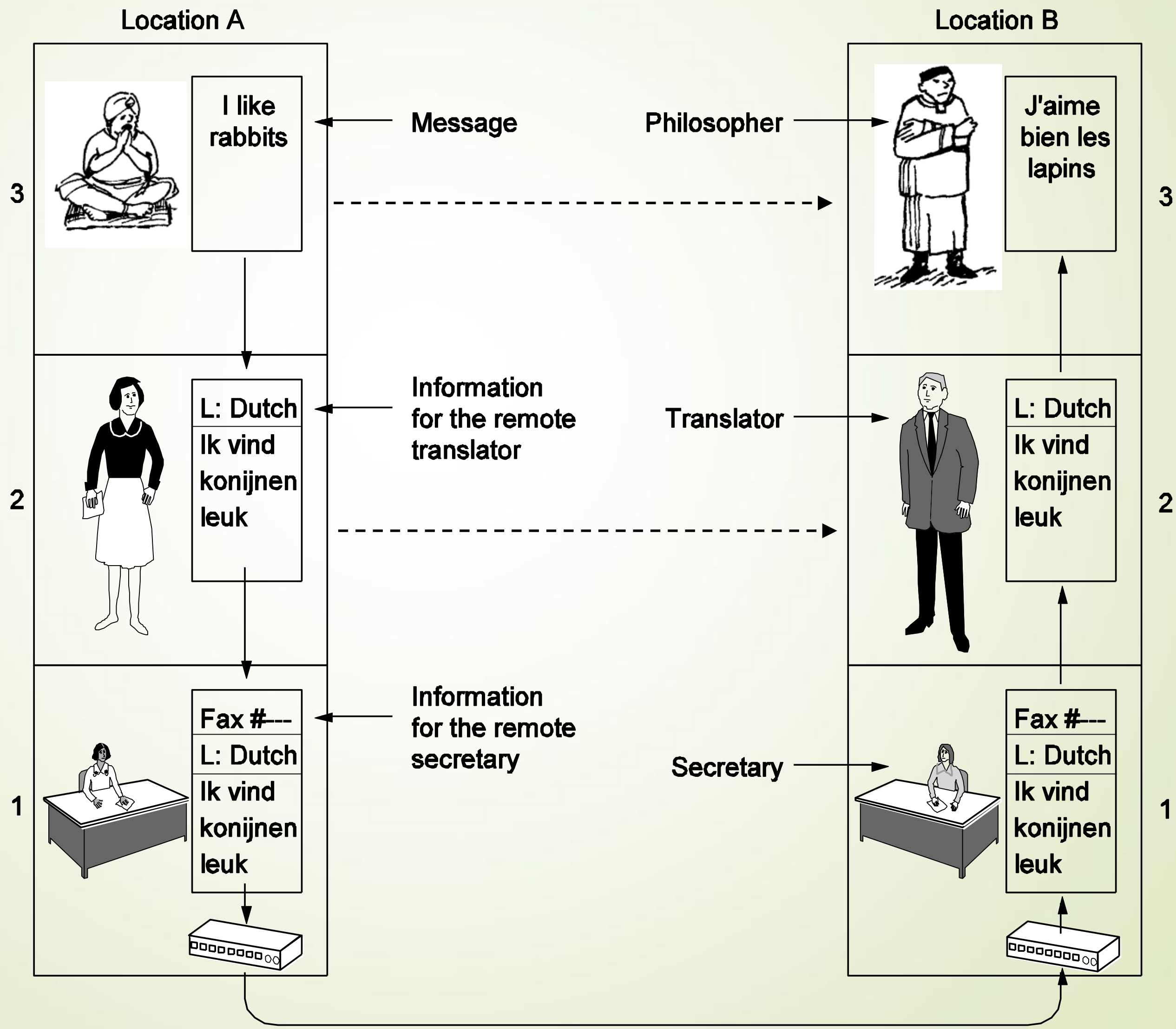


A server can be any application. A client can be any application.

A Real World Example to Protocol Architecture philosopher-translator-secretary architecture

Issues:

- peer-to-peer protocols are independent of each other
  - for example, secretaries may change the comm. medium to email
  - or the translators may agree on using another common language
- Each layer adds a header

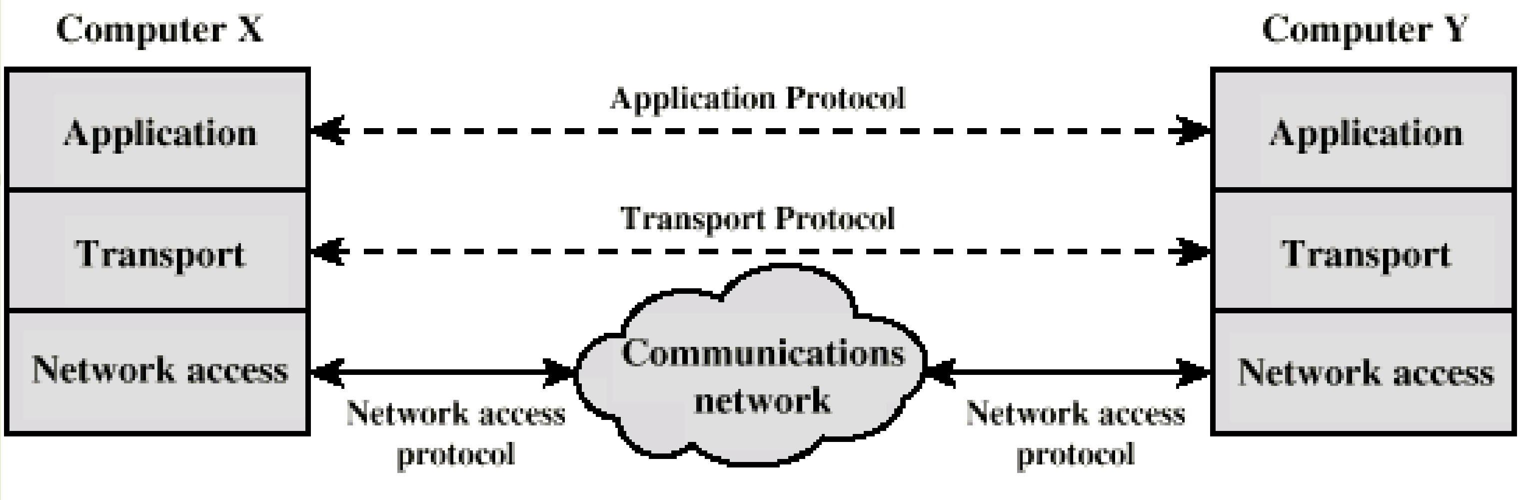


General protocol architecture principles that we have seen so far

- Layered structure
  - Protocol stack
- Each layer provides services to upper layer; expect services from lower one
  - Layer interfaces should be well-defined
- Peer entities communicate using their own protocol
  - peer-to-peer protocols
  - independent of protocols at other layers
  - if one protocol changes, other protocols should not get affected

A General Three Layer Model

- Generalize the previous example for a generic application
  - we can have different applications (e-mail, file transfer, …)



- Network Access Layer
- Transport Layer
- Application Layer

Network Access Layer

- Exchange of data between the computer and the network
- Sending computer provides address of destination
  - so that network can route
- Different switching and networking techniques
  - Circuit switching
  - Packet switching
  - LANs
  - etc.
- This layer may need specific drivers and interface equipment depending on type of network used.
- But upper layers do not see these details
  - independence property

Transport Layer

- Reliable data exchange
  - to make sure that all the data packets arrived in the same order in which they are sent out
  - Packets nor received or received in error are retransmitted
- Independent of network being used
- Independent of application

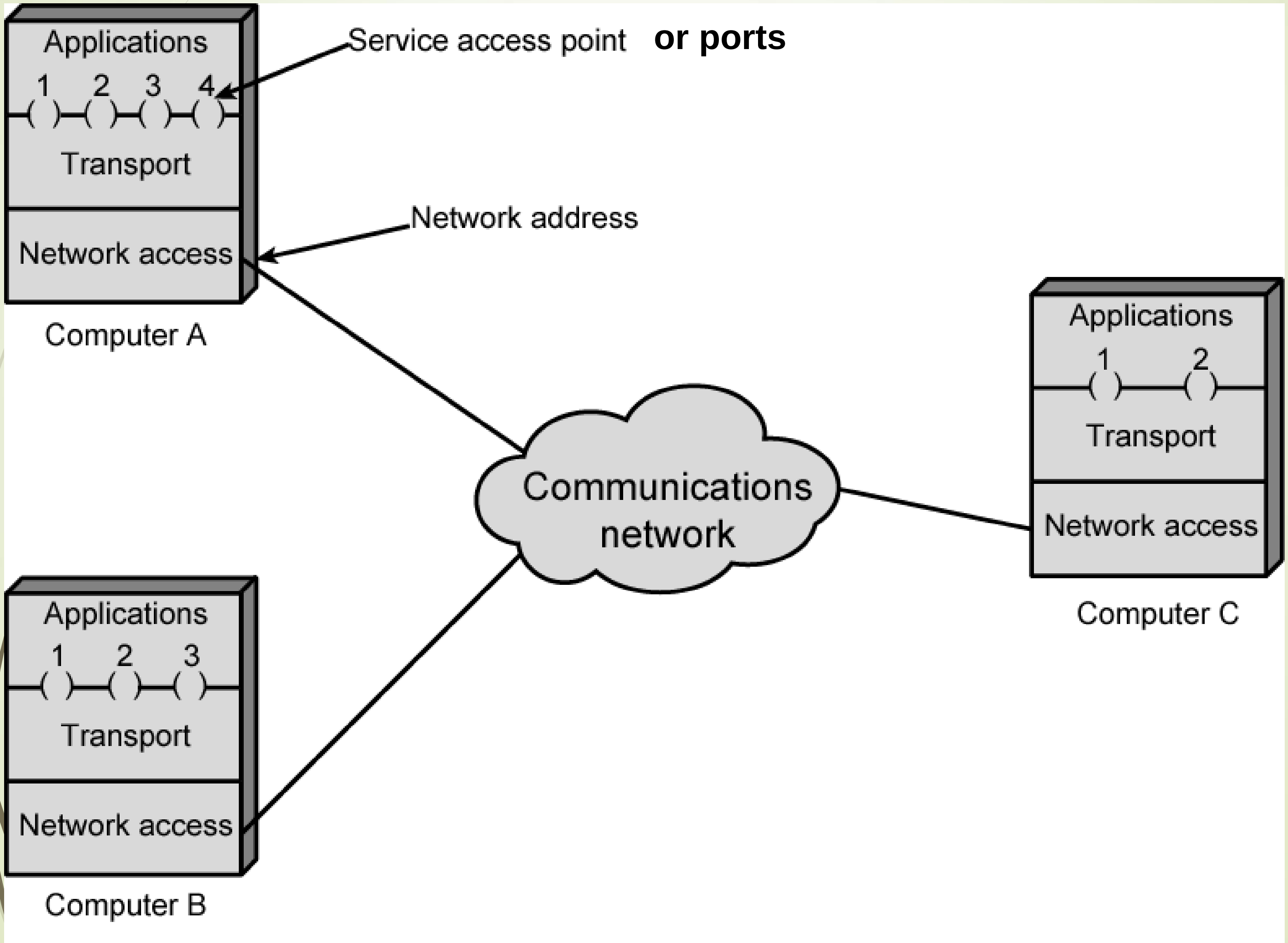
Application Layer

- Support for different user applications
- e.g. e-mail, file transfer

Addressing Requirements

- Two levels of addressing required
- Each computer needs unique network address
- Each application on a (multi-tasking) computer needs a unique address within the computer
  - The *service access point* or SAP
  - The *port number* in TCP/IP protocol stack

Protocol Architectures and Networks



Protocol Data Units (PDU)

- User data is passed from layer to layer
- Control information is added/removed to/from user data at each layer
  - Header (and sometimes trailer)
  - each layer has a different header/trailer
- Data + header + trailer = PDU (Protocol Data Unit)
  - This is basically what we call *packet*
  - each layer has a different PDU

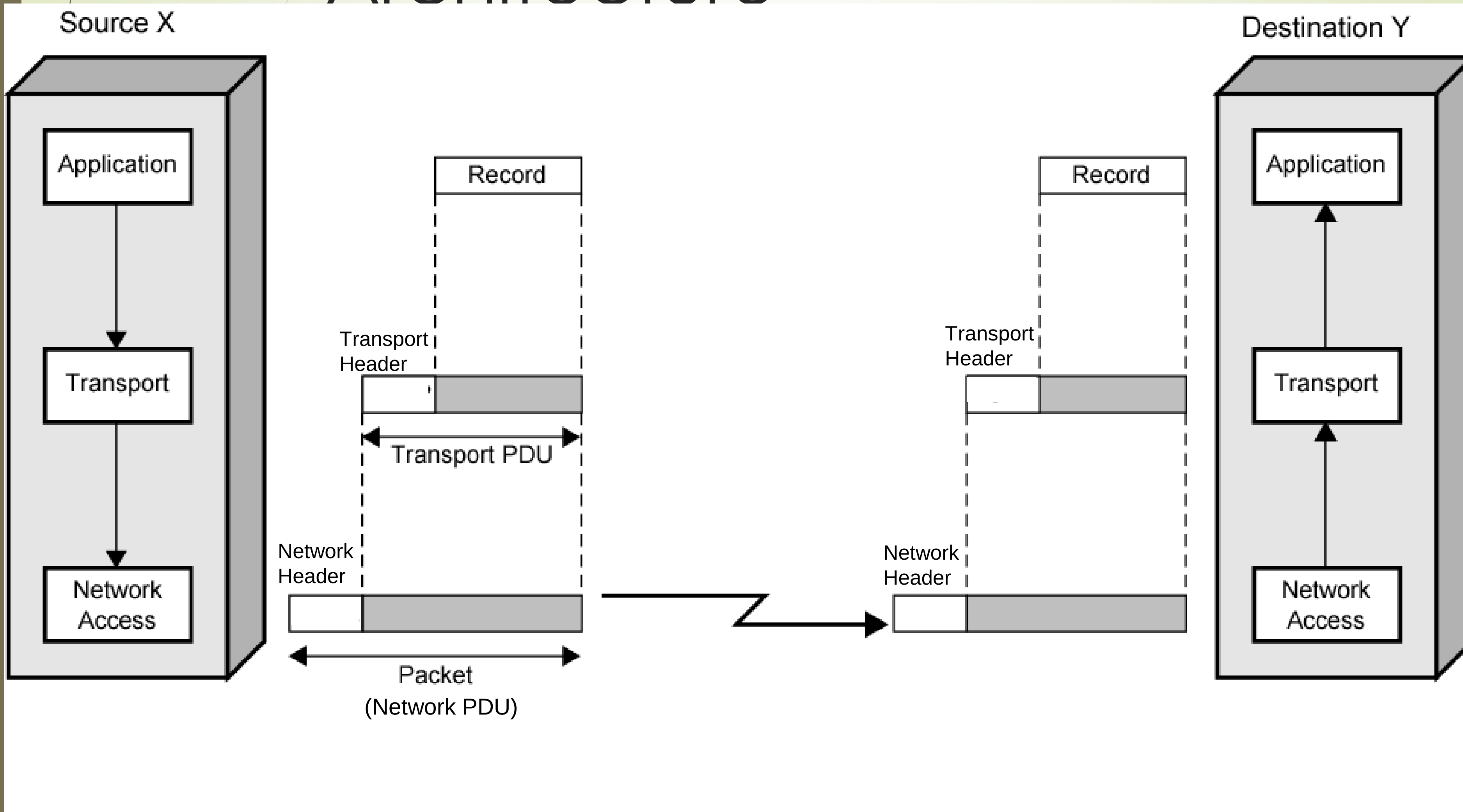
Transport PDU

- Transport layer may fragment user data
- Each fragment has a transport header added
  - Destination port
  - Sequence number
    - since the transport layer may split application data into smaller packets
  - Error detection code (generally at trailer)

Network PDU

- Adds network header
  - network address for destination computer
  - optional facilities from network (e.g. priority level)

Operation of a Protocol Architecture



Standard Protocol Architectures

- Common set of conventions
- Nonstandard vs. standard protocols
  - Nonstandard: K sources and L receivers lead to K*L different protocols
  - If common protocol used, we design only once
- Products from different vendors interoperate
  - If a common standard is not implemented in a product, then that product's market is limited; customers like standard products
  - Customers do not stick to a specific vendor

Standard Protocol Architectures

- Two approaches (standard)
  - OSI Reference model
    - never used widely
    - but well known
  - TCP/IP protocol suite
    - Most widely used
- Another approach (proprietary)
  - IBM's Systems Network Architecture (SNA)

OSI Reference Model

- Open Systems Interconnection
- Reference model
  - provides a general framework for standardization
  - defines a set of layers and services provided by each layer
  - one or more protocols can be developed for each layer
- Developed by the International Organization for Standardization (ISO)
  - also published by ITU-T (International Telecommunications Union)

OSI Reference Model

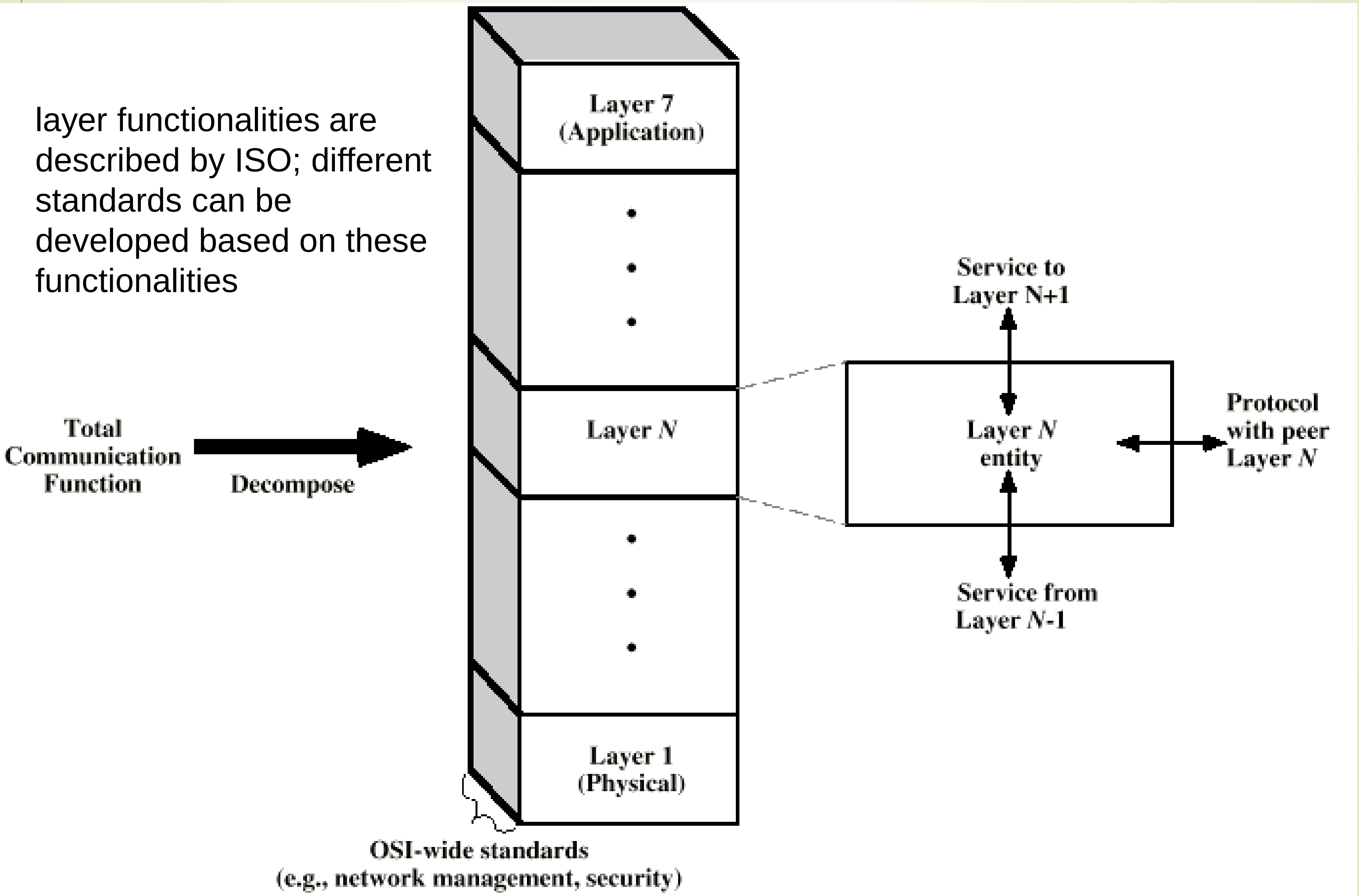
- A layered model
  - Seven layers – seven has been presented as the optimal number of layer
- Delivered too late (published in 1984)!
  - by that time TCP/IP started to become the de facto standard
- Although no OSI-based protocol survived, the model is still valid (in the textbooks)

OSI - The Layer Model

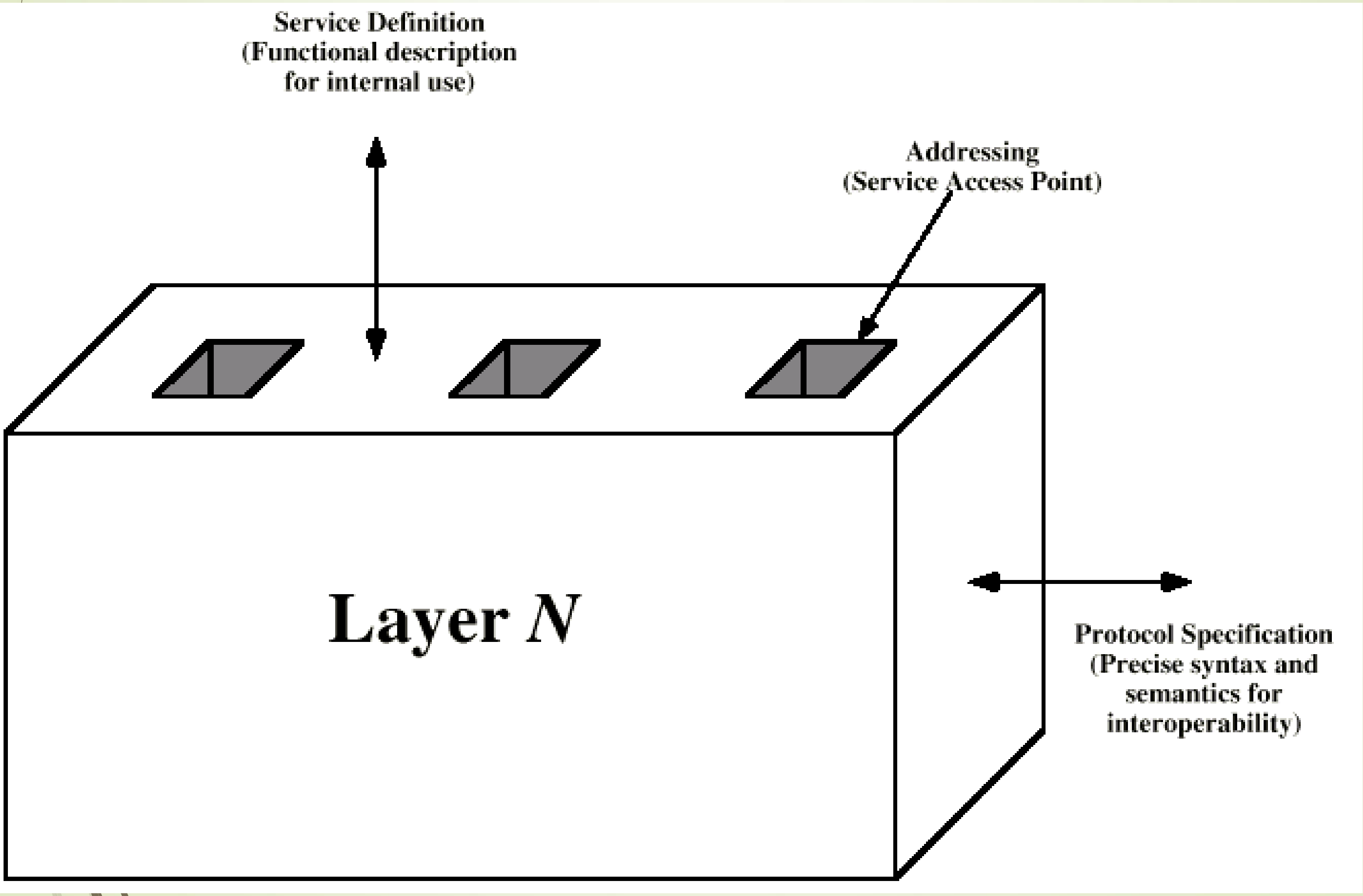
- Each layer performs a subset of the required communication functions
- Each layer relies on the next lower layer to perform more primitive functions
- Each layer provides services to the next higher layer
- Changes in one layer should not require changes in other layers

OSI as Framework for Standardization

layer functionalities are described by ISO; different standards can be developed based on these functionalities



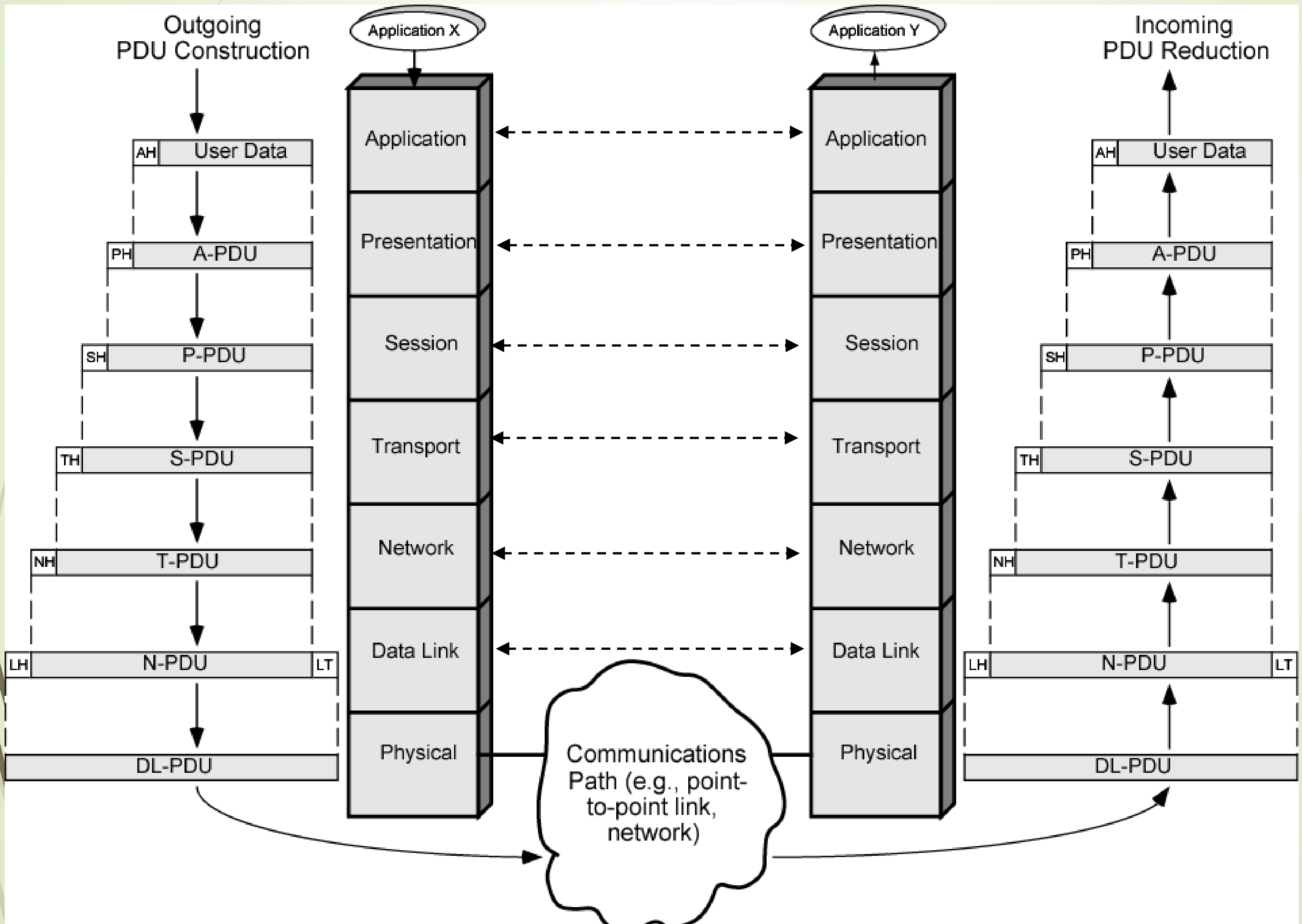
Layer Specific Standards



Elements of Standardization

- Protocol specification
  - Operates between the same layer on two systems
    - May involve different platforms
  - Protocol specification must be precise
    - Format of data units
    - Semantics of all fields
- Service definition
  - Functional description of what is provided to the next upper layer
- Addressing
  - Referenced by SAPs

The OSI Environment



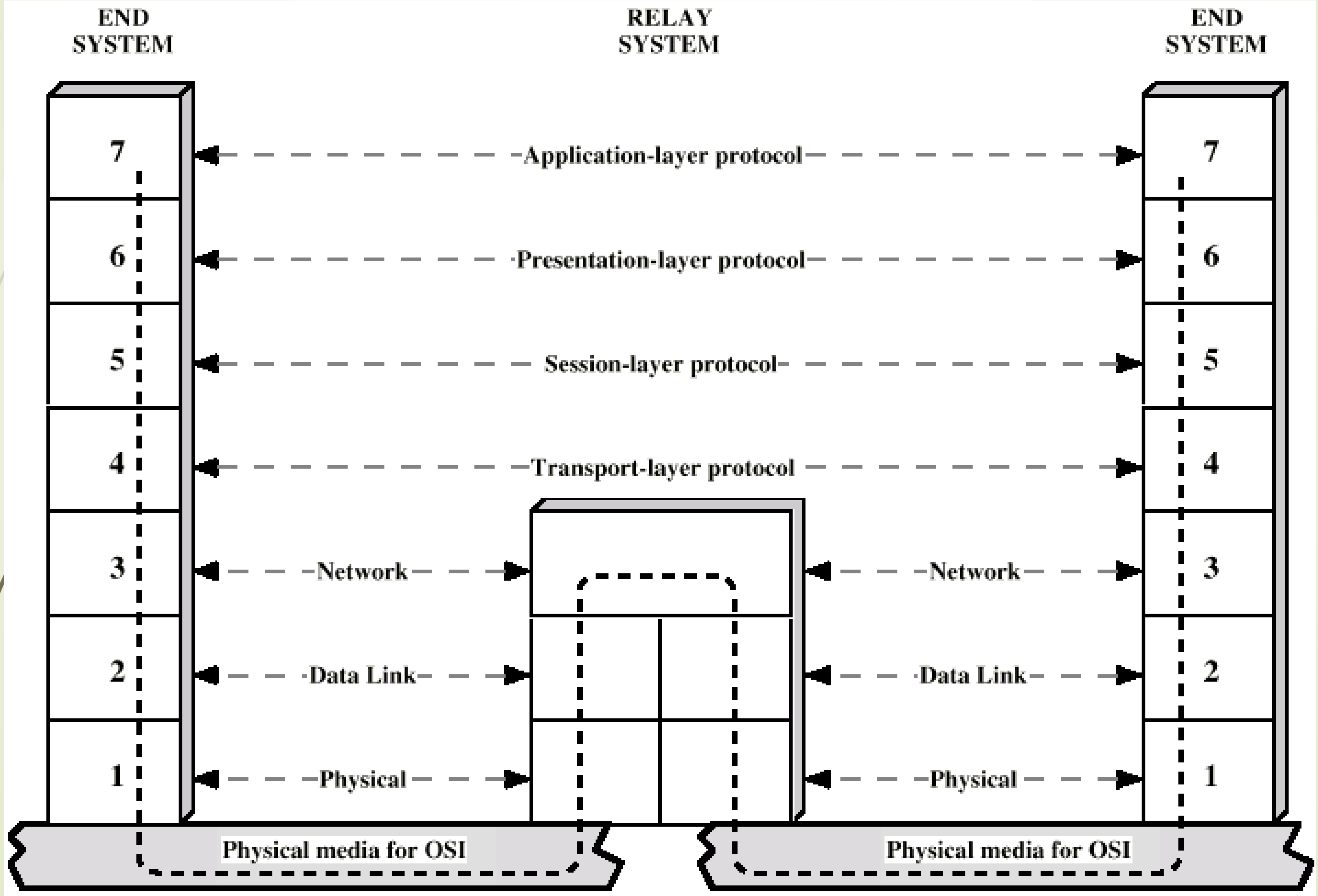
OSI Layers (1)

- Physical
  - Physical interface between devices
  - Characteristics
    - Mechanical - interface specs
    - Electrical - voltage levels for bits, transmission rate, coding, etc.
- Data Link
  - Basic services: error detection and control, flow control at the link level (point to point)
    - Higher layers may assume error free transmission
  - Later a sublayer is added to Data Link Layer
    - MAC (Medium Access Control) sublayer
    - to deal with broadcast networks

OSI Layers (2)

- Network
  - Transfer of information through communication network
    - network related issues
  - Network nodes (relays/routers) should perform switching and routing functions
  - QoS (Quality of Service) and congestion control are also addressed in this layer
  - Several other internetworking issues
    - e.g. differences in addressing, max. data length, etc.
  - Higher layers do not need to know about underlying networking technology
  - Not needed on direct links

Use of a Relay/Router



OSI Layers (3)

- Transport
  - End to end exchange of data
  - In sequence, no losses, no duplicates
  - If needed, upper layer data are split into smaller units
- Session
  - Control of dialogues
    - whose turn to talk?
    - Dialogue discipline (full-duplex, half-duplex)
  - Checkpointing and recovery

OSI Layers (4)

- Presentation
  - Data formats
  - Data compression
  - Encryption
- Application
  - Support for various applications

TCP/IP Protocol

- Most widely used interoperable network protocol architecture
- Specified and extensively used before OSI
  - OSI was slow to take place in the market
- Funded by the US Defense Advanced Research Project Agency (DARPA) for its packet switched network (ARPANET)
  - DoD automatically created an enormous market for TCP/IP
- Used by the Internet and WWW

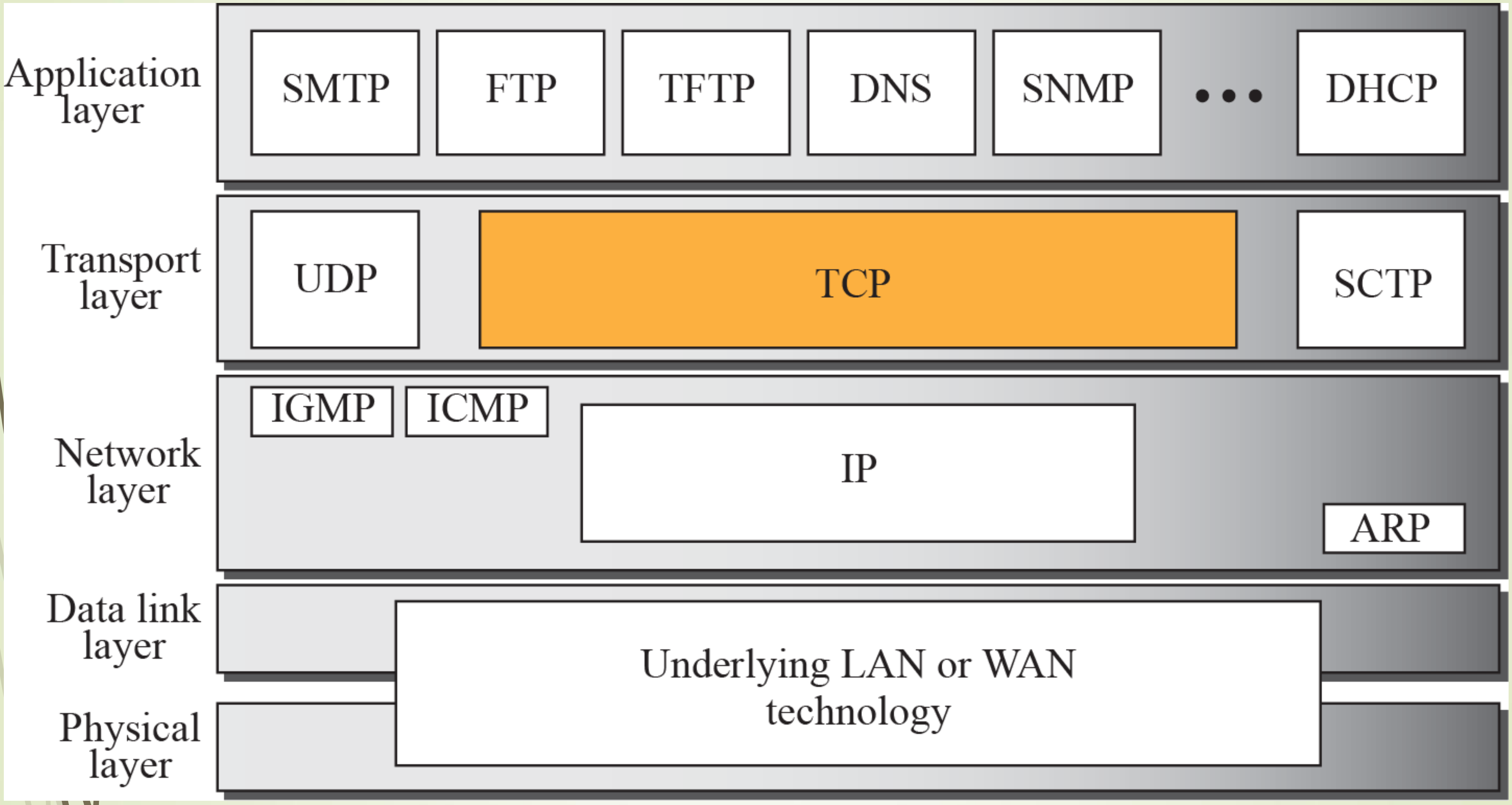
Common Protocols in TCP/IP Protocol Stack

- ARP: Address Resolution Protocol
- IP: Internet Protocol (RFC 791)
- UDP: User Datagram Protocol (RFC 768)
- TCP: Transmission Control Protocol (RFC 793)

TCP/IP Protocol Suite

- TCP/IP does not have an official layer structure
- But protocols imply one
  - Application layer
  - Transport (host to host) layer
  - Internet layer
  - Network access layer
  - Physical layer
- Actually TCP/IP reference model has been built on its protocols
  - That is why that reference model is only for TCP/IP protocol suite
  - and this is why it is not so important to assign roles to each layer in TCP/IP; understanding TCP, IP and the application protocols would be enough

TCP/IP protocol



OSI vs. TCP/IP

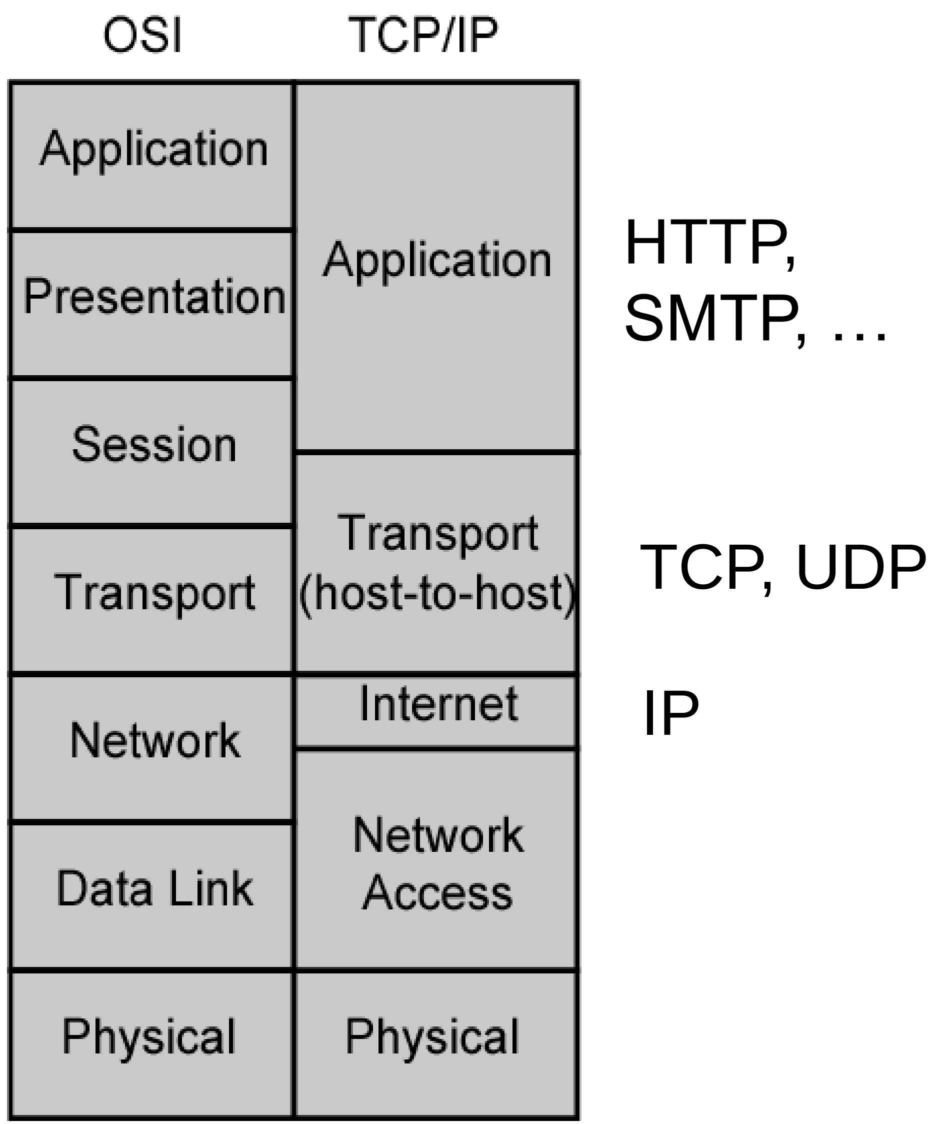

OSI
TCP/IP
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Transport
(host-to-host)
Internet
Network
Access
Physical
HTTP,
SMTP, …
TCP, UDP
IP

Table 15.1 *Well-known Ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

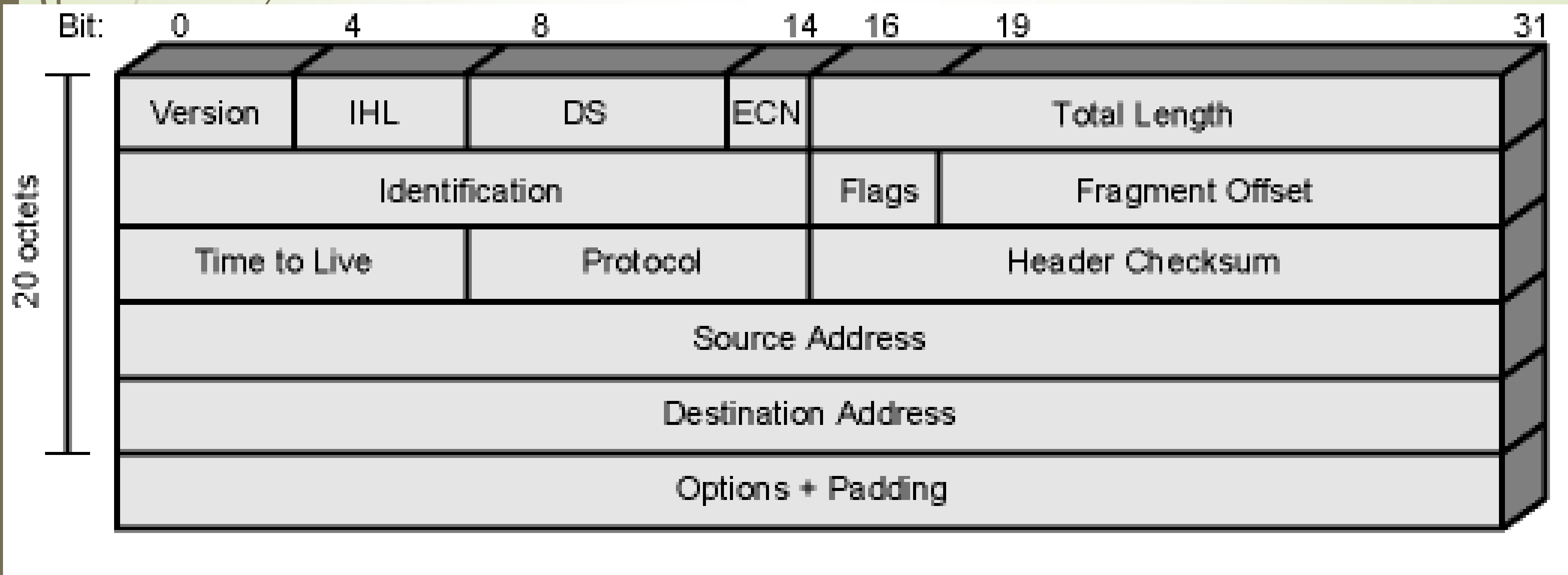
Network Access and Physical Layers

- TCP/IP reference model does not discuss these layers too much
  - the node should connect to the network with a protocol such that it can send IP packets
  - this protocol is not defined by TCP/IP
  - mostly in hardware
  - a well known example is Ethernet

IP (Internet Protocol)

The core of the TCP/IP protocol suite

- Two versions co-exist
  - v4 – the widely used IP protocol
  - v6 – has been standardized in 1996, but still not widely deployed
- IP (v4) header minimum 20 octets (160 bits)



IPv6

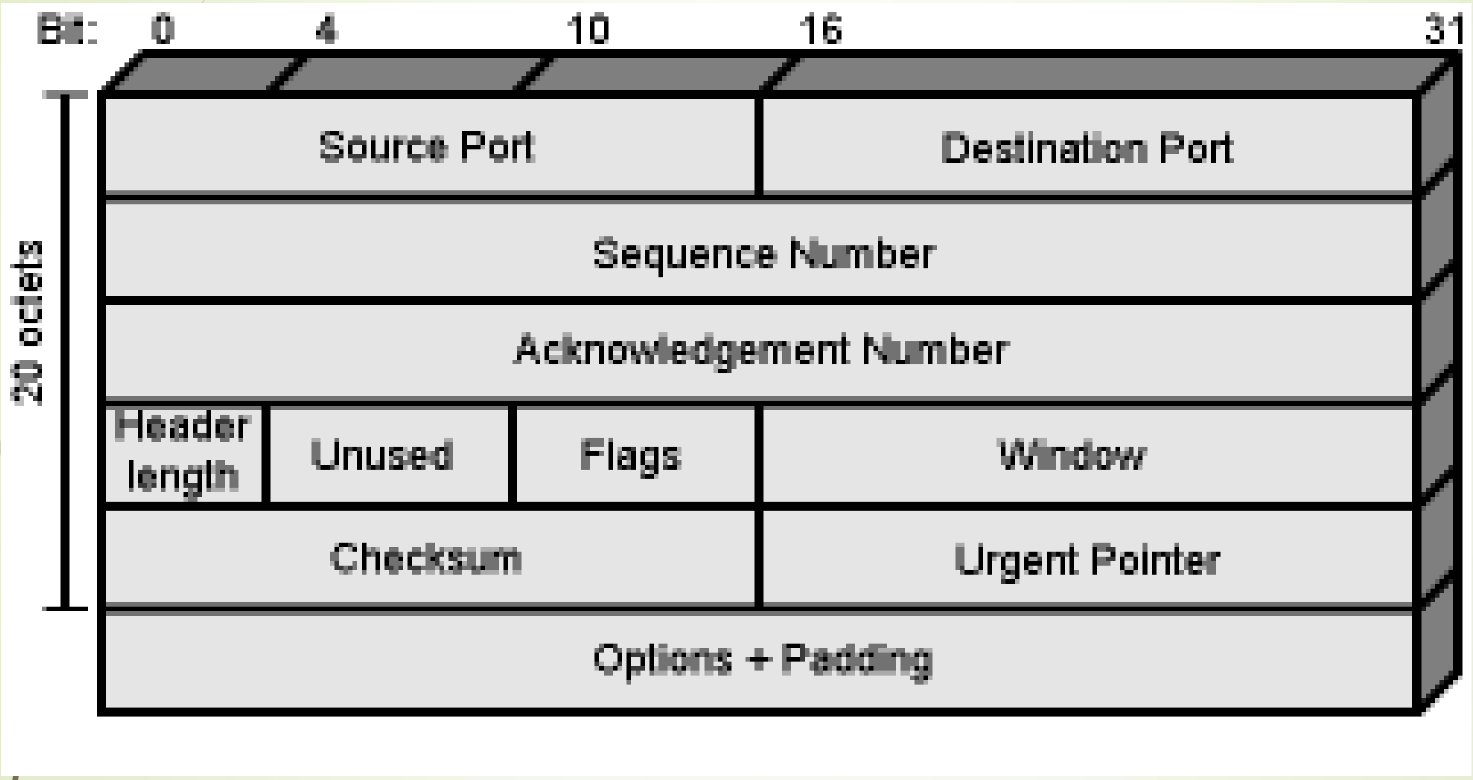
- IPv6
  - Enhancements over IPv4 for modern high speed networks
  - Support for multimedia data streams
- But the driving force behind v6 was to increase address space
  - 128-bit as compared to 32-bit of v4
- Not backward compatible
  - all equipment and software must change

TCP

- Transmission Control Protocol
  - end to end protocol
  - Reliable connection = provides flow and error control
- In TCP terms, a *connection* is a

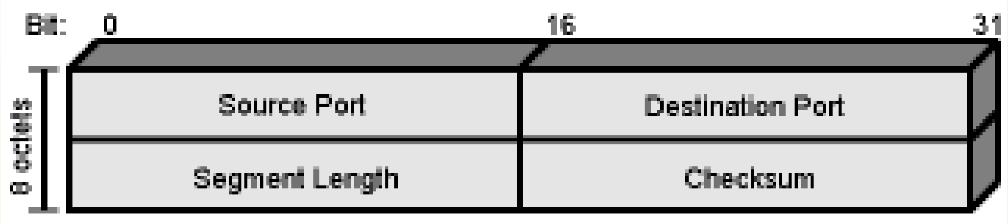
  temporary association between entities in different systems
- TCP PDU
  - Called "TCP segment"
  - Includes source and destination port
    - Identify respective users (applications)
    - pair of ports (together with the IP addresses) uniquely identify a connection; such an identification is necessary in order TCP to track segments between entities.

TCP Header


Bit: 0 4 10 16 31
20 octets
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Header length
Unused
Flags
Window
Checksum
Urgent Pointer
Options + Padding

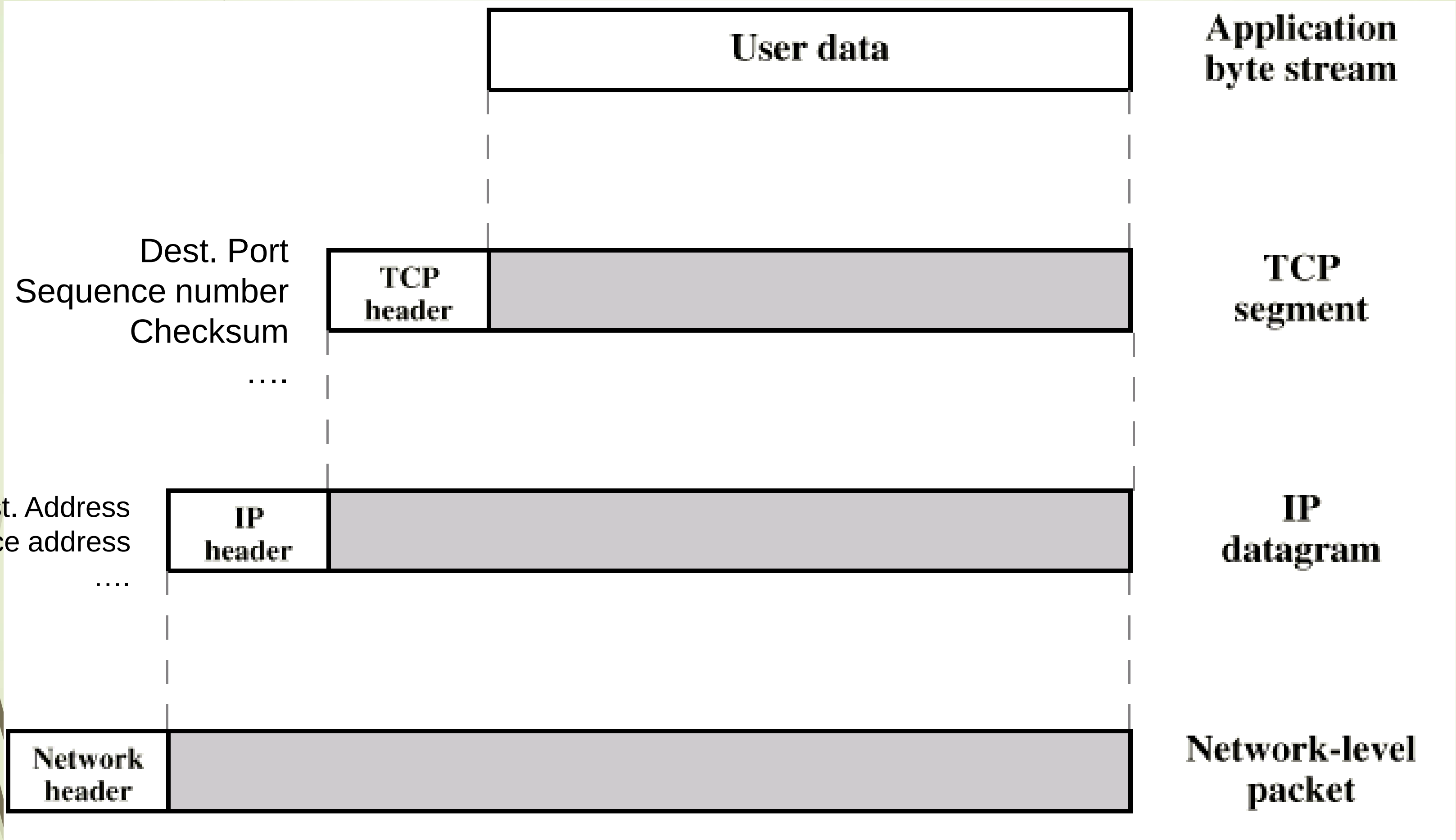
UDP

- User Datagram Protocol
- Alternative to TCP
  - end-to-end protocol
- Not guaranteed delivery
- No preservation of sequence
- No protection against duplication
- Minimum overhead



(b) UDP Header

PDUs in TCP/IP

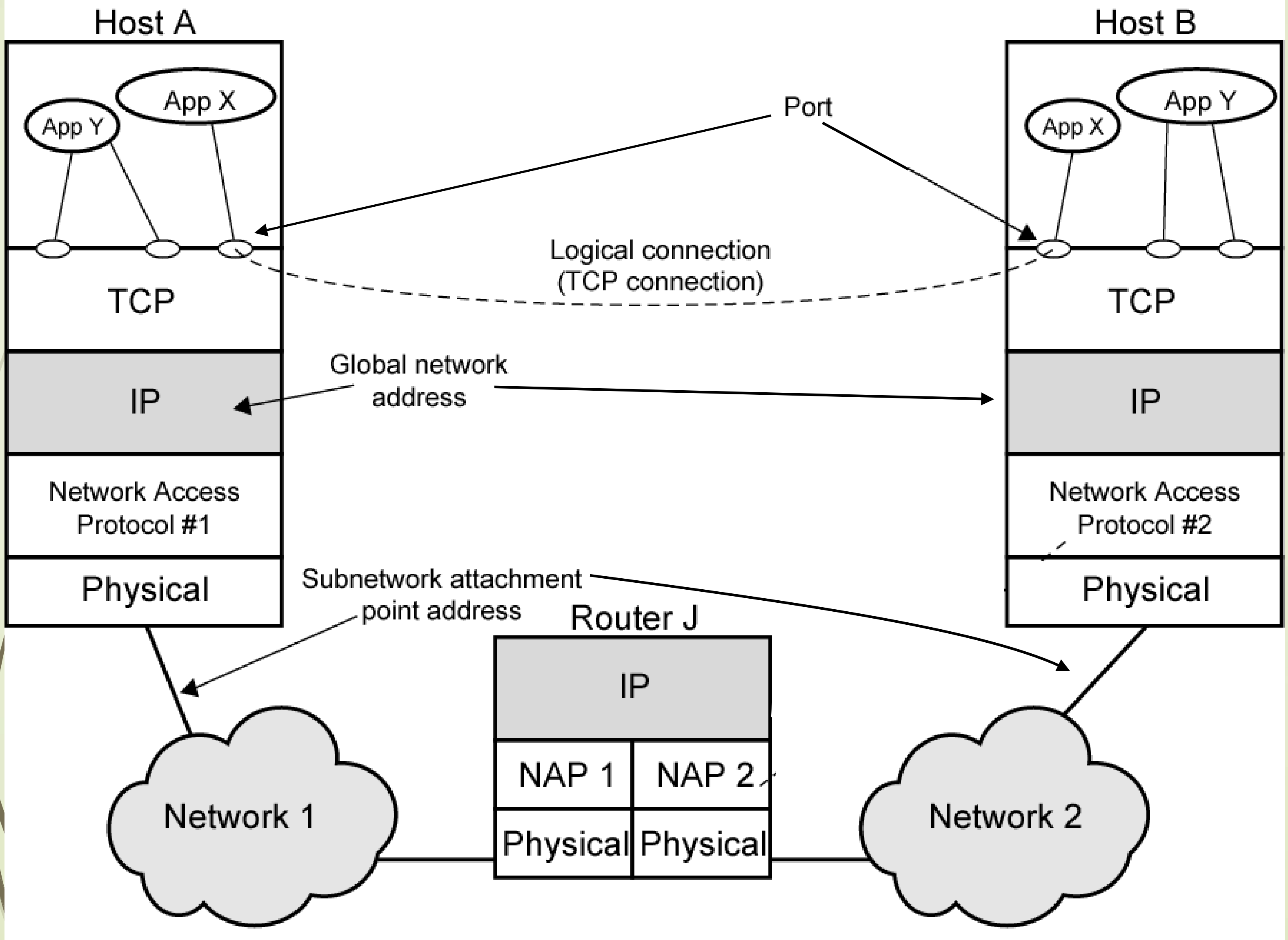


Dest. Port
Sequence number
Checksum
….

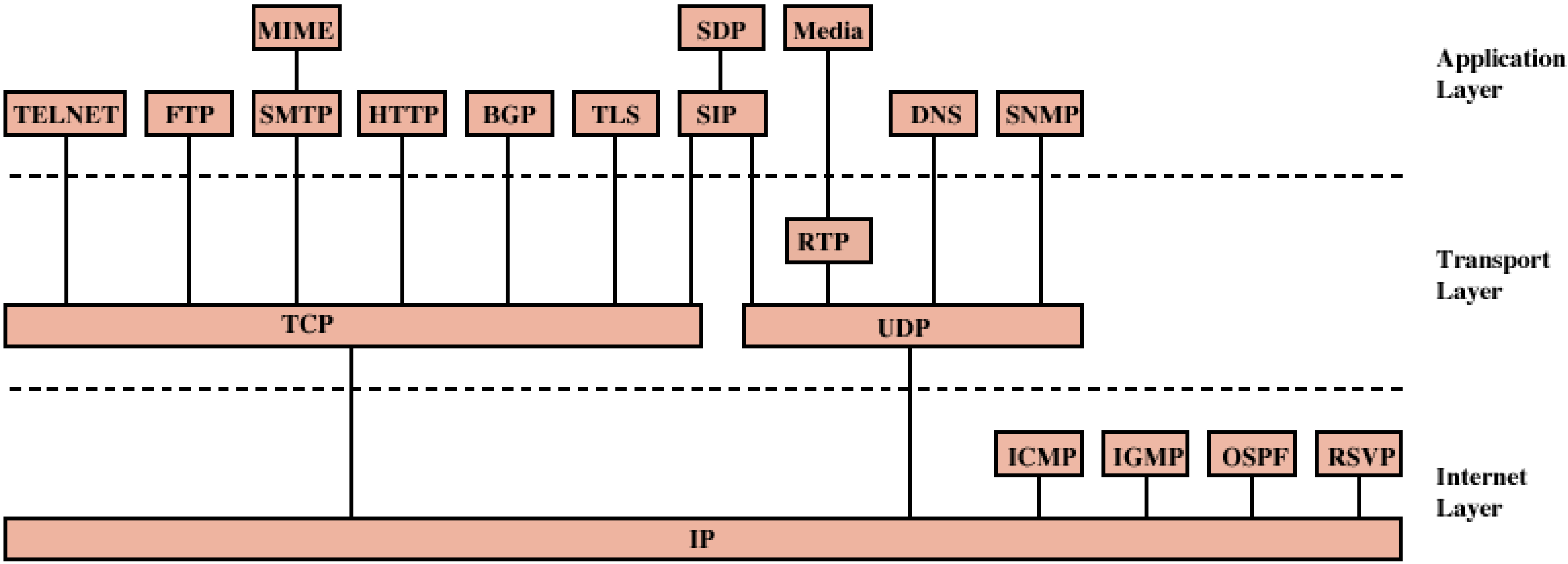
Dest. Address
Source address
….

Dest. Network Address
Priority info

Operation of TCP and IP



Some Protocols in TCP/IP Suite



BGP	=	Border Gateway Protocol	RSVP	=	Resource ReSerVation Protocol
DNS	=	Domain Name System	RTP	=	Real-Time Transport Protocol
FTP	=	File Transfer Protocol	SDP	=	Session Description Protocol
HTTP	=	Hypertext Transfer Protocol	SIP	=	Session Initiation Protocol
ICMP	=	Internet Control Message Protocol	SMTP	=	Simple Mail Transfer Protocol
IGMP	=	Internet Group Management Protocol	SNMP	=	Simple Network Management Protocol
IP	=	Internet Protocol	TCP	=	Transmission Control Protocol
MIME	=	Multi-Purpose Internet Mail Extension	TLS	=	Transport Layer Security
OSPF	=	Open Shortest Path First	UDP	=	User Datagram Protocol

Internetworking

- Interconnected set of networks
  - May be seemed as a large network
- Each constituent network is a *subnetwork*
- Entire configuration referred to as an *internet*
  - not *the Internet*
    - conceptually the same, but by “internet” we do not mean a specific network
    - the Internet is the most important example of an internet

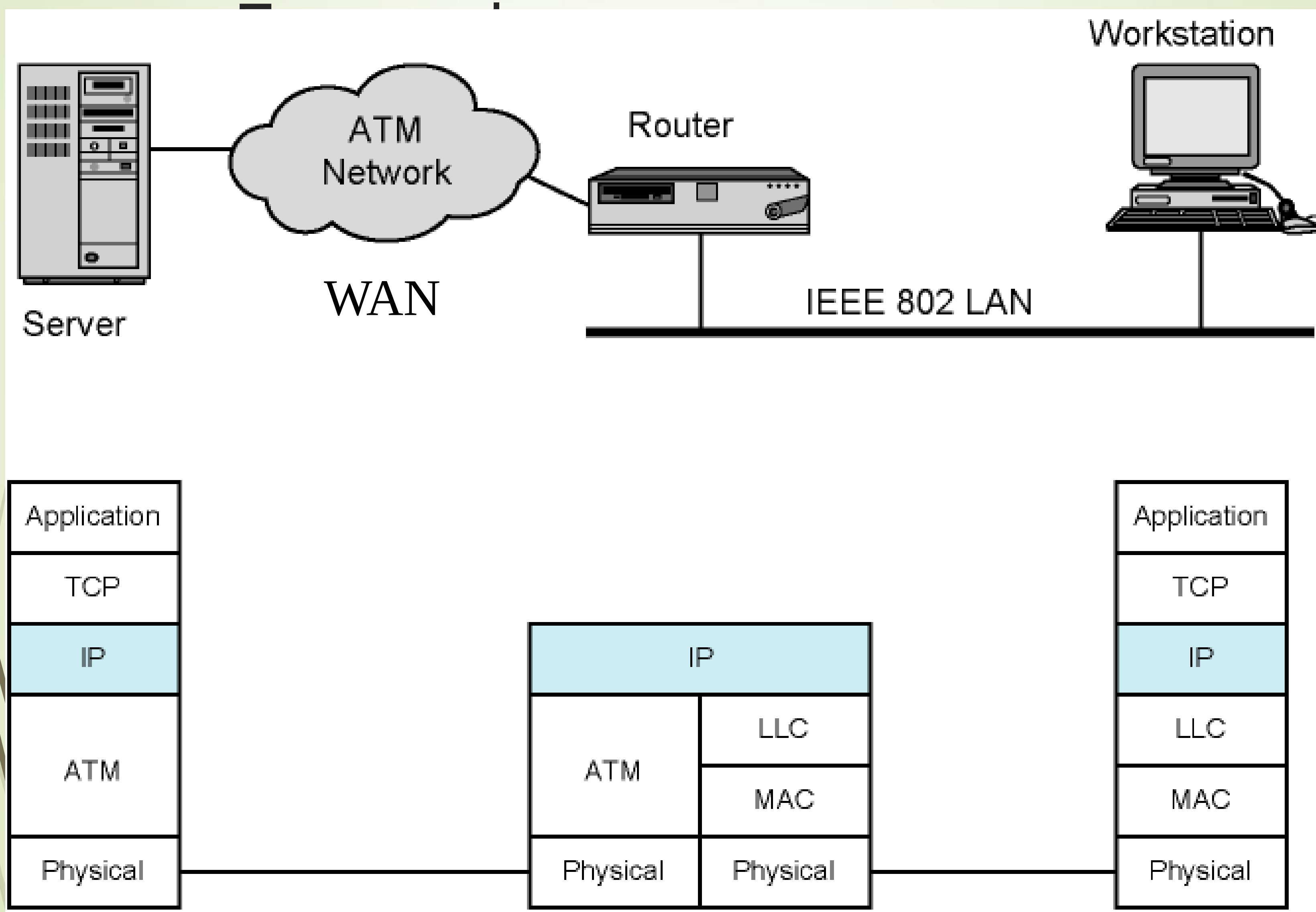
Internetworking Devices

- Each subnetwork supports communication among the devices attached to that subnetwork
  - End systems (ESs)
- Subnetworks connected by intermediate systems (ISs)
  - In practice, ISs are *routers* that are used to relay and route packets between different subnetworks
  - If subnetworks use different Network Access Protocols, router should support all of the protocols
  - In OSI terminology, a router works at layer 3 (network layer)

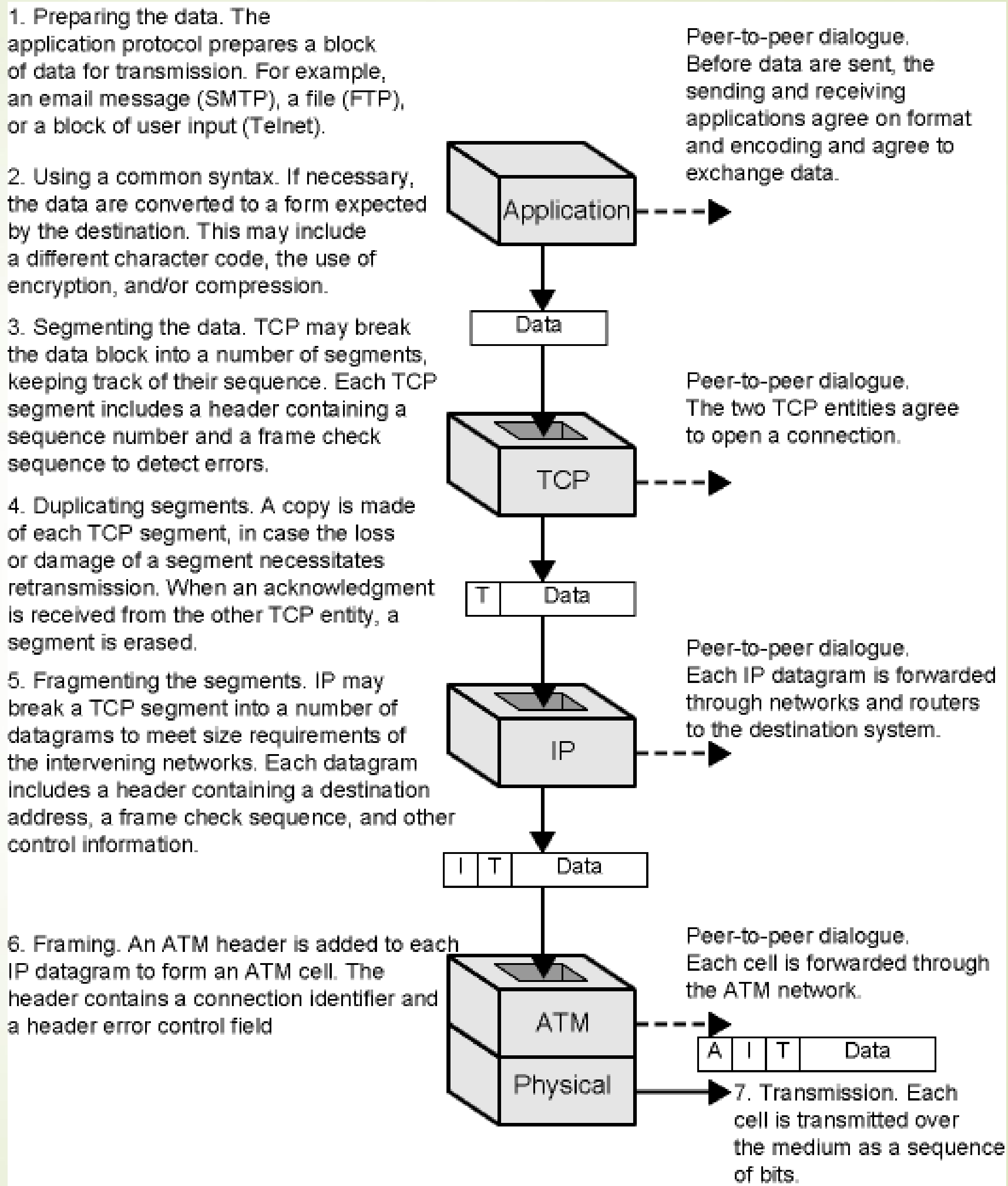
Routers

- Interconnect dissimilar subnetworks without any modifications on architecture of subnetworks
- Must accommodate differences among networks, such as
  - Addressing schemes
    - network addresses may need to be translated
  - Maximum packet sizes
    - if two subnetworks have different limits for max. packet sizes, then router may need fragment/reassemble the packets
- We have seen that subnetworks may have different network access and physical layers, but they have to speak the same (inter)network protocol implemented in all end systems and routers
  - The most important internetwork protocol is the IP protocol

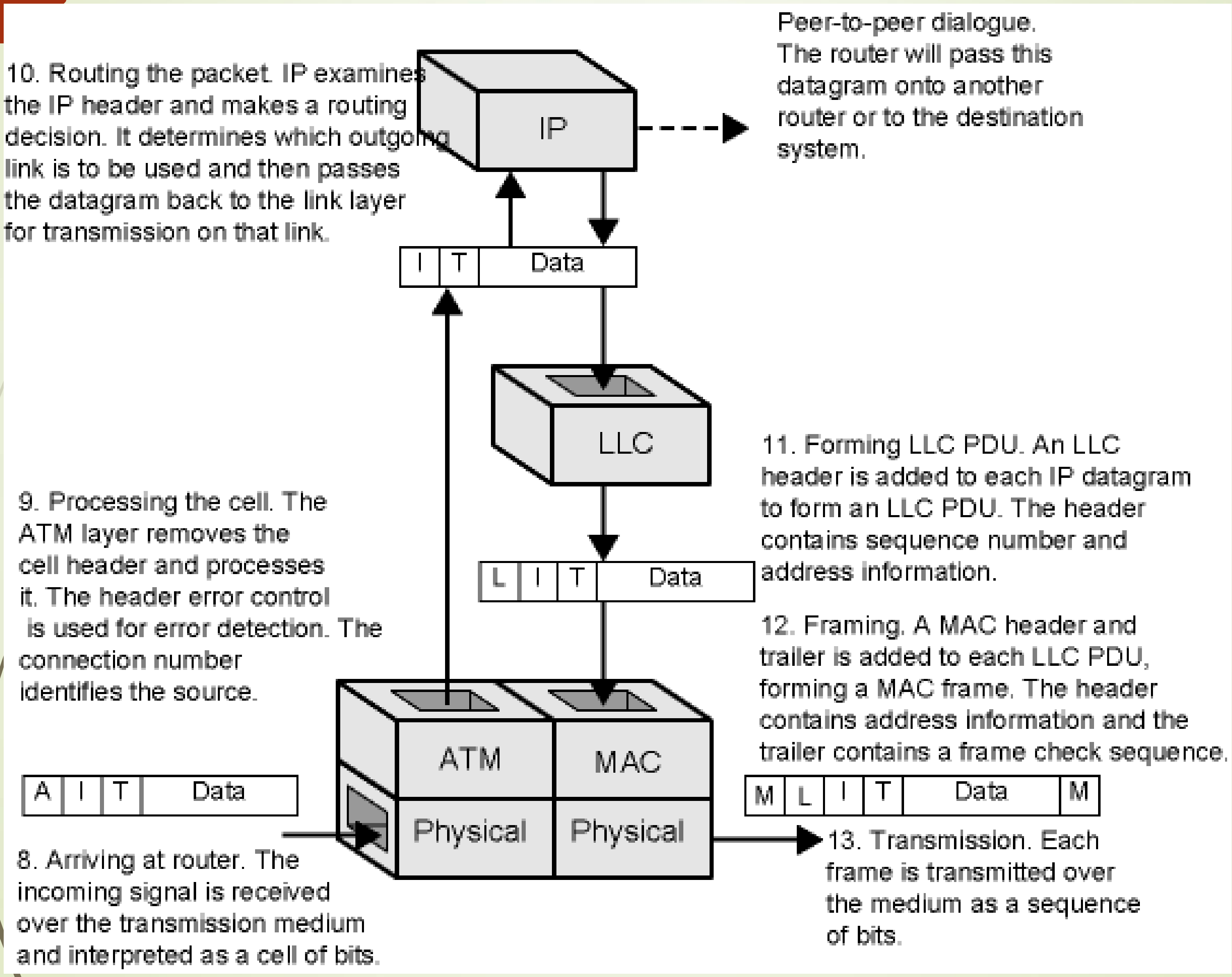
Configuration for TCP/IP



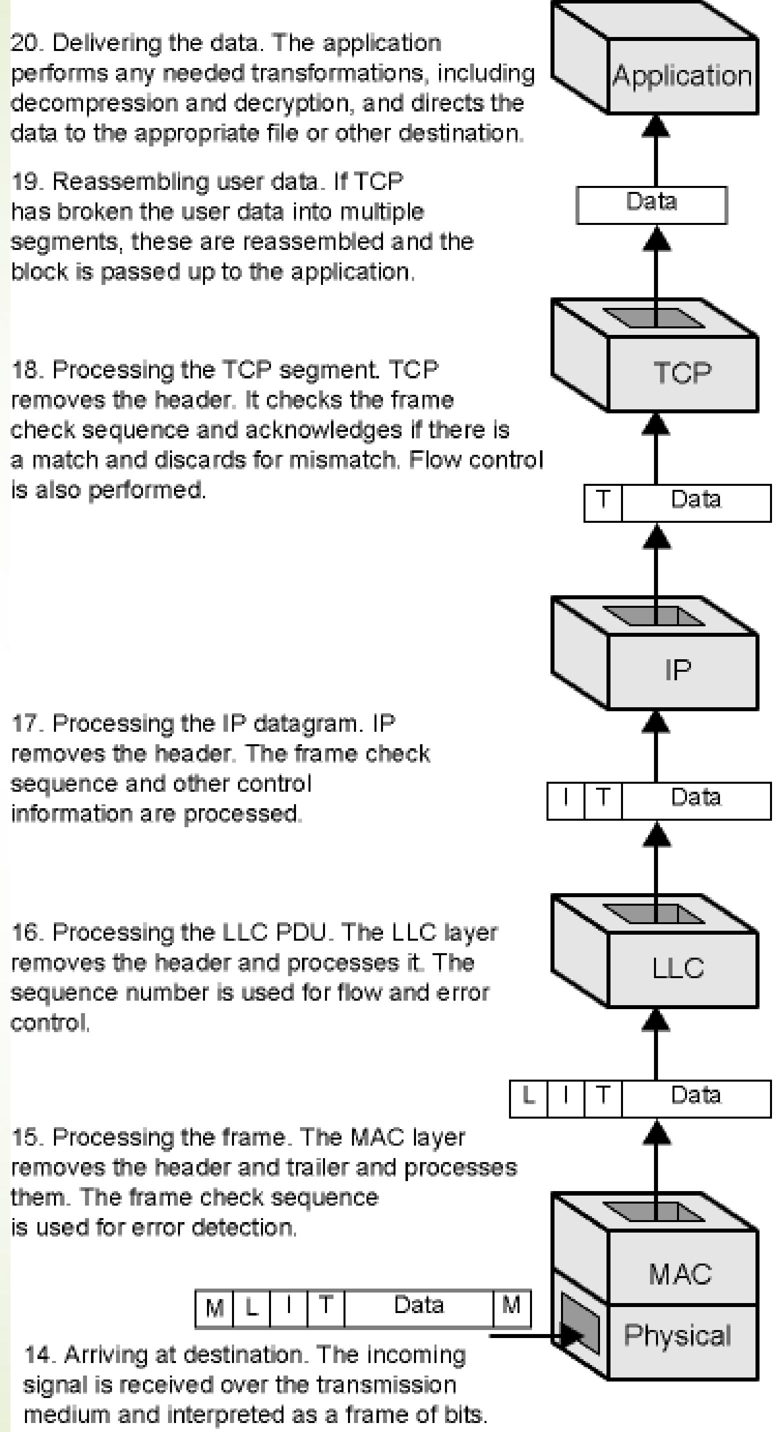
Action of Sender



Action of Router



Action of Receiver



Standards

- Required to allow for interoperability among equipments
- Advantages
  - Ensures a large market for equipment and software
  - Allows products from different vendors to communicate
- Disadvantage
  - Freeze technology (???)

Standards Organizations in Networking

- Internet Society
- ISO (International Organization for Standardization)
  - more formal
  - NGO, but most members are from governments
- ITU-T (formerly CCITT)
  - International Telecommunications Union
  - UN agency
  - governmental

Internet Society (ISOC)

- Internet development and standardization
- 3 suborganizations
  - IAB (Internet Architecture Board)
    - overall Internet architecture
  - IETF (Internet Engineering Task Force)
    - protocol engineering and development
  - IESG (Internet Engineering Steering Group)
    - monitors IETF standardization efforts

IETF Organization

- Grouped in areas
  - e.g. applications, security, routing, etc.
  - each area has an Area Director, who is also member of IESG
- Each area has several working groups
  - working groups actually contribute to standards/protocols, etc.
- Voluntary participation in IETF working groups
- For detail see
  - www.ietf.org or
  - RFC 3160 - The Tao of IETF - A Novice's Guide to the Internet Engineering Task Force

Internet Drafts and RFCs

- Internet Draft
  - Draft and temporary documents
  - expires in 6 months, if IESG does not approve it as an RFC
  - can be resubmitted
  - published online
  - comments are welcome
- RFC (Request for Comments)
  - final version
  - can obsolete previous RFCs about the same topic
  - actually an RFC can be of any type of document
    - not necessarily a standard
    - Best Current Practice, Experimental, Informational RFCs
    - April 1st RFCs (http://en.wikipedia.org/wiki/April_1_RFC)
      - My favorite is *IP over Avian Carriers* (RFC 1149)

Internet Standards Track

- Steps involve increasing amount of scrutiny and testing
- Step 1: Internet Draft
- Step 2: Proposed standard
  - Internet Draft approved as an RFC by IESG
  - must remain at least six months to advance
- Step 3: Draft standard
  - at least two independent and interoperable implementations
  - must remain at least 4 months
- Step 4: Internet standard
  - Significant operational experience
    - key difference between ISOC and other standardization organizations
  - Consensus needed

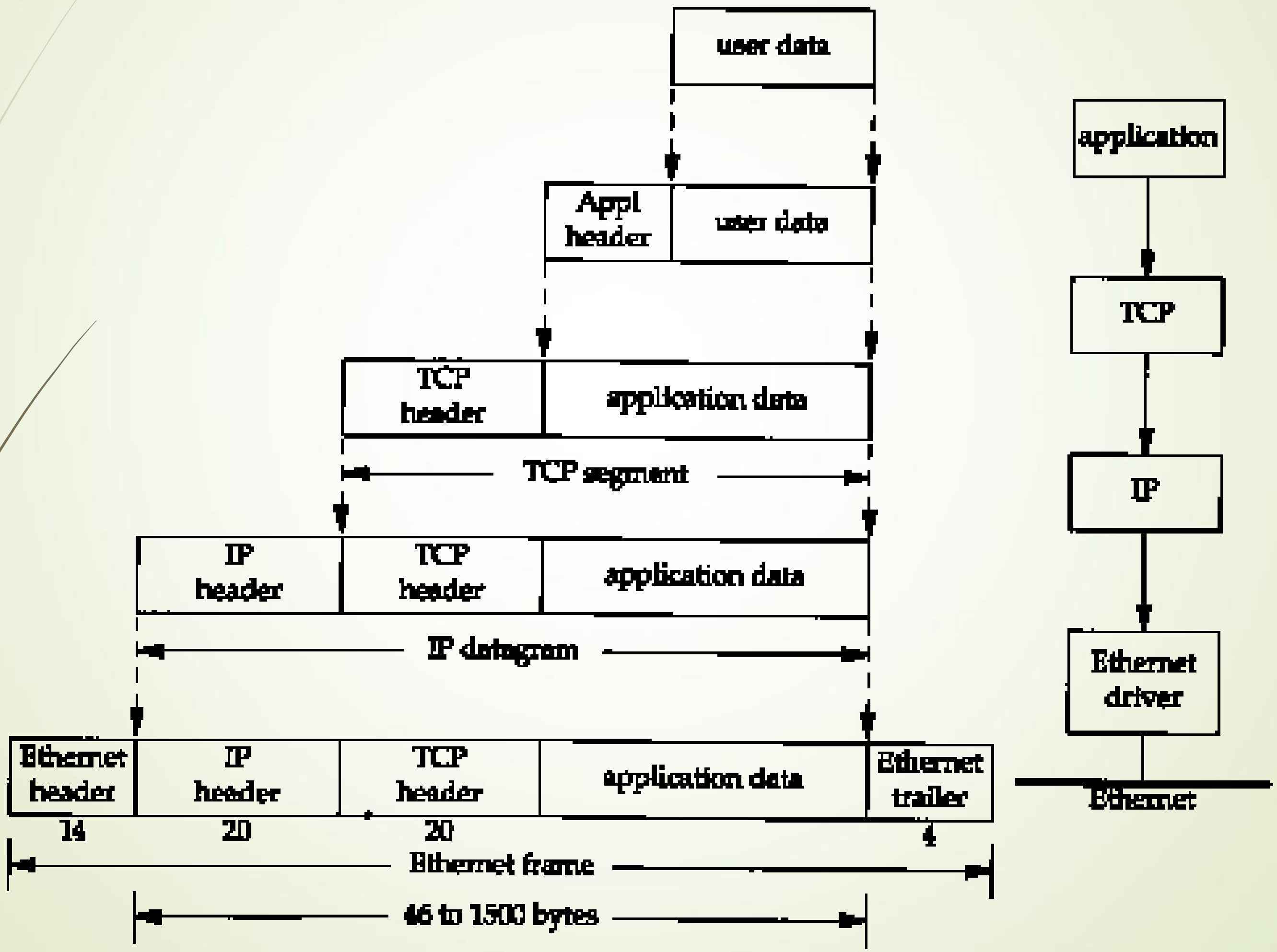
Internet Assigned Numbers Authority (IANA)

- An ISOC entity responsible for all “unique numbers” on the Internet
  - including IP addresses
- Almost all protocols work with numeric parameters
  - e.g. port numbers, error codes, status codes, message types, options, etc.
  - the meanings of all numeric codes are mostly specified in RFCs, but number assignment is formalized by IANA

Networking

- Layering
  - ISO OSI 7-layer model
    - Physical, data link, network, transport, session, presentation, application
  - TCP/IP model
    - Link, network, transport, application

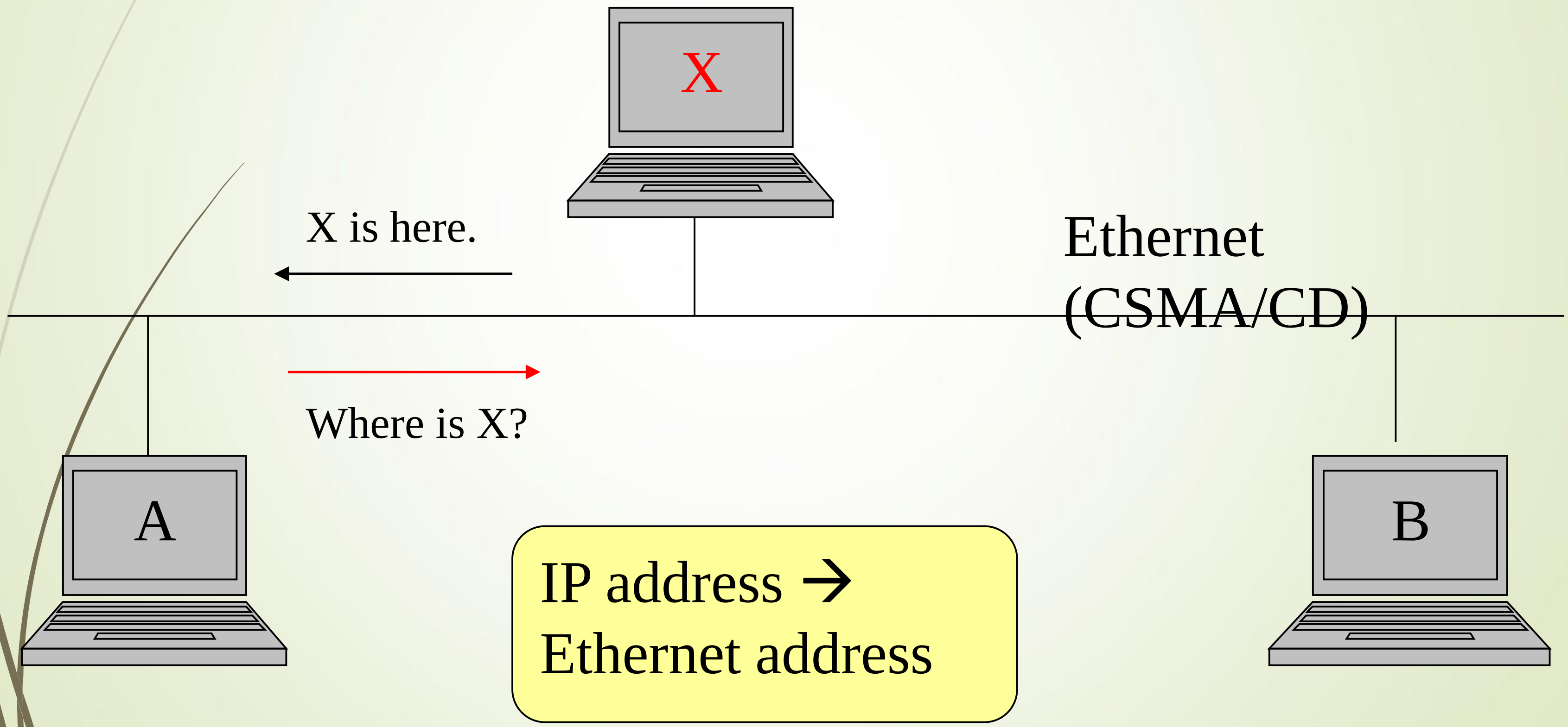
Encapsulation



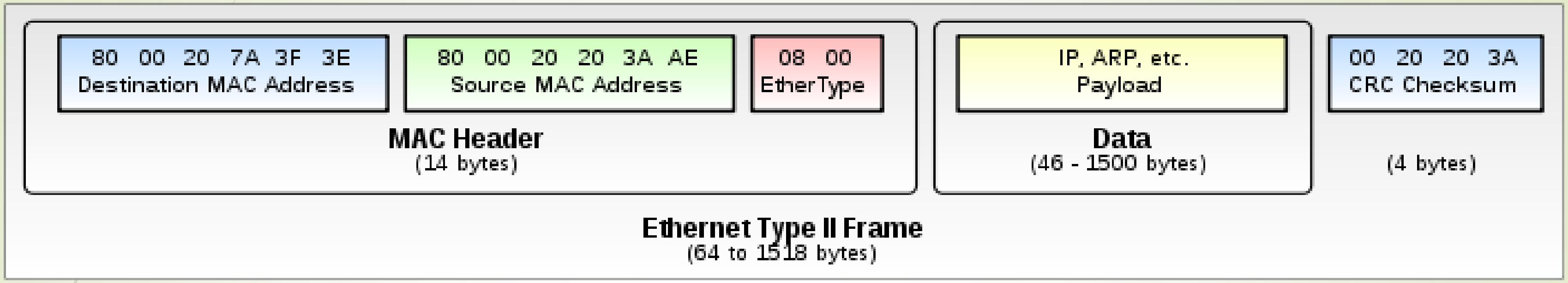
Protocol Headers

- Ethernet header
  - MAC (Ethernet) addresses
- IP header
  - IP addresses, protocol
- TCP/UDP header
  - Port numbers

ARP


X
X is here.
Ethernet
(CSMA/CD)
Where is X?
A
B
IP address →
Ethernet address

Ethernet Header



Payload

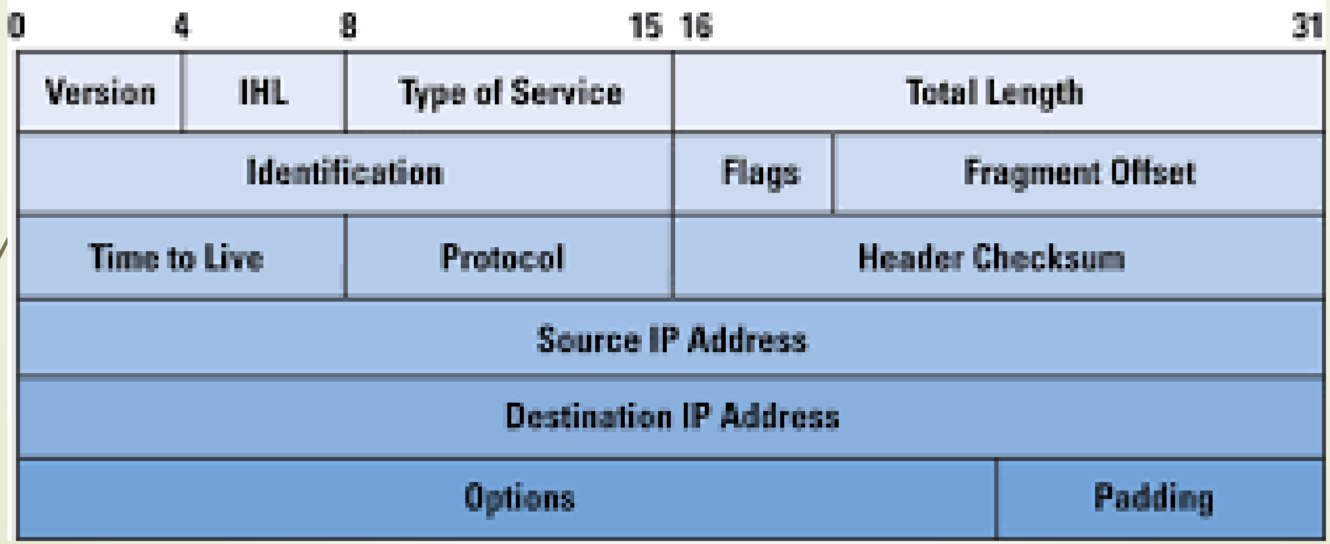
The minimum payload is 42 octets when an 802.1Q tag is present and 46 octets when absent. The maximum payload is 1500 octets.

Frame check sequence

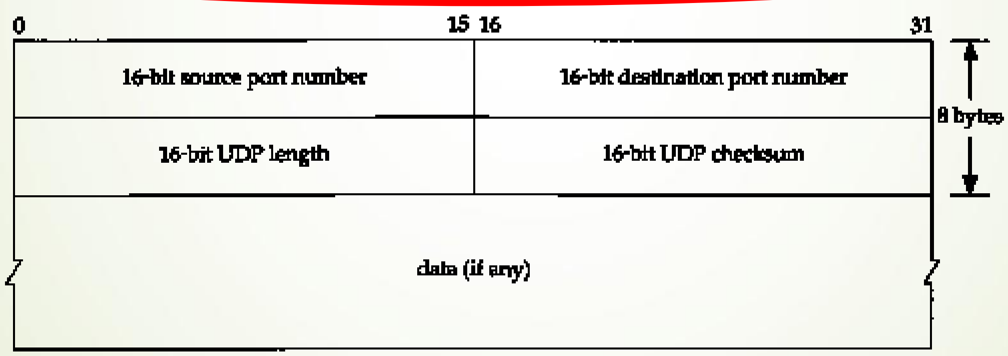
The frame check sequence (FCS) is a four-octet cyclic redundancy check (CRC) that allows detection of corrupted data within the entire frame as received on the receiver side.

IP Header

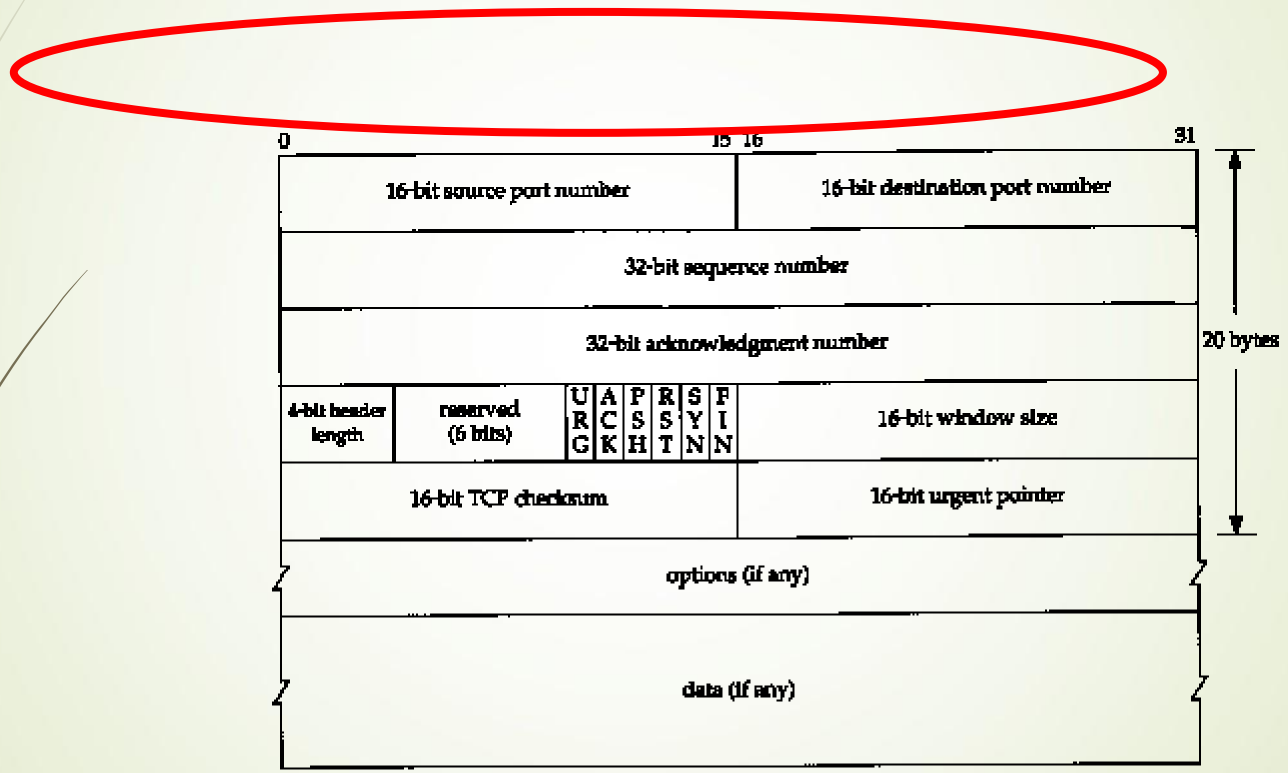
Figure 1: The IP Header



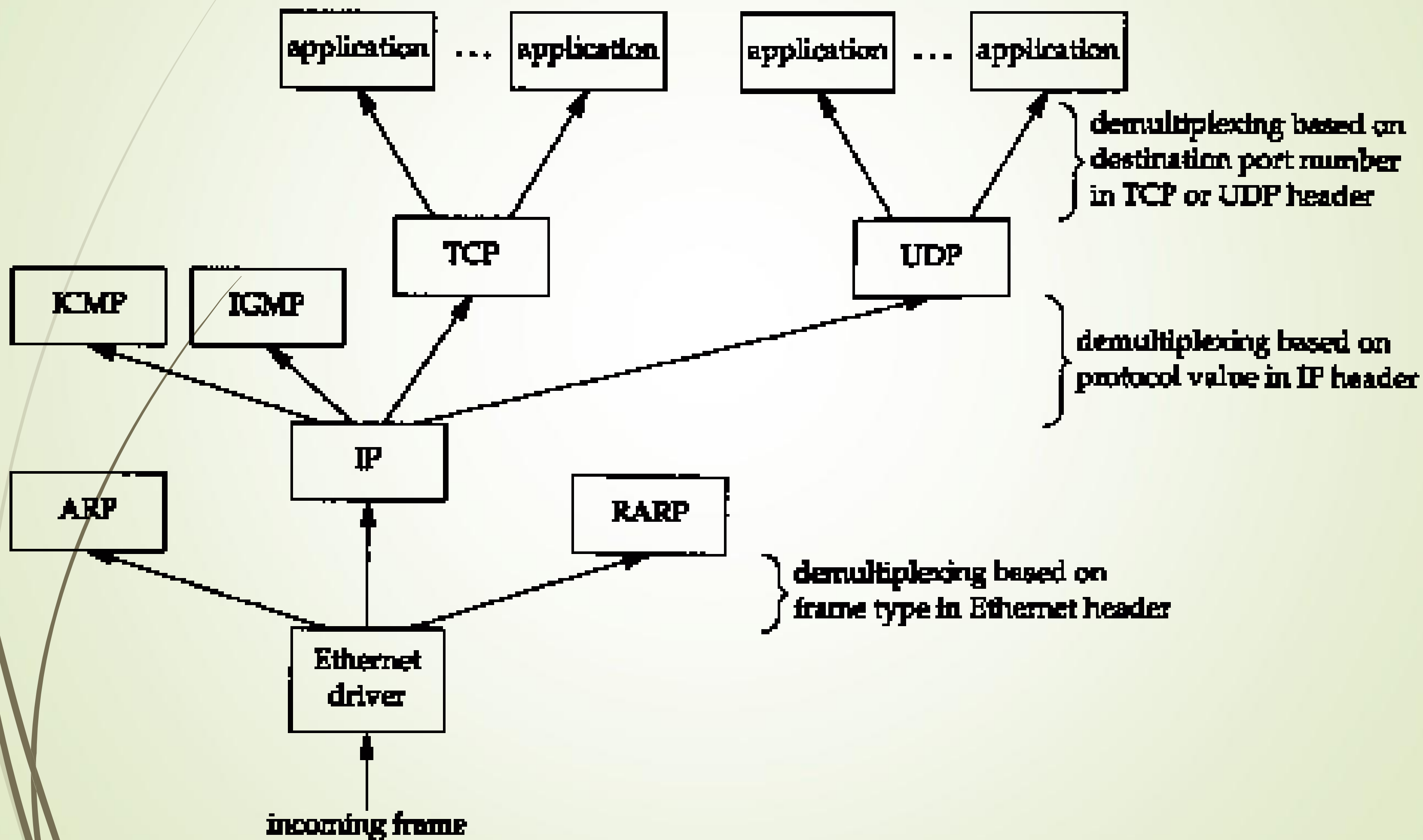
UDP Header


0
15 16
31
16-bit source port number
16-bit destination port number
16-bit UDP length
16-bit UDP checksum
8 bytes
data (if any)

TCP Header


0
15 16
31
16-bit source port number
16-bit destination port number
32-bit sequence number
32-bit acknowledgment number
4-bit header length
reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
16-bit window size
16-bit TCP checksum
16-bit urgent pointer
options (if any)
data (if any)
20 bytes

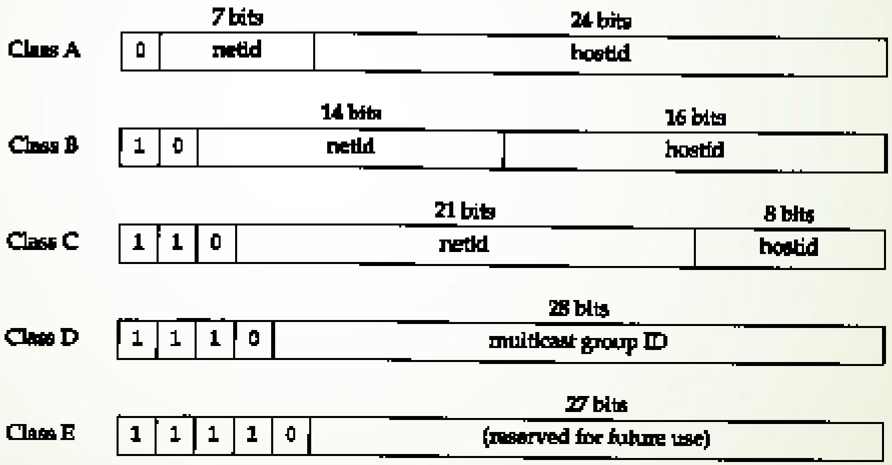
Demultiplexing



IP Addresses

- IPv4 address
  - Dotted decimal: 140.112.8.130
- Unicast, broadcast, and multicast
- Private address space
  - 10.0.0.0 - 10.255.255.255 (10/8 prefix)
  - 172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
  - 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)
- Class A, B, C, D, E

IP Addresses (cont.)



Port Numbers

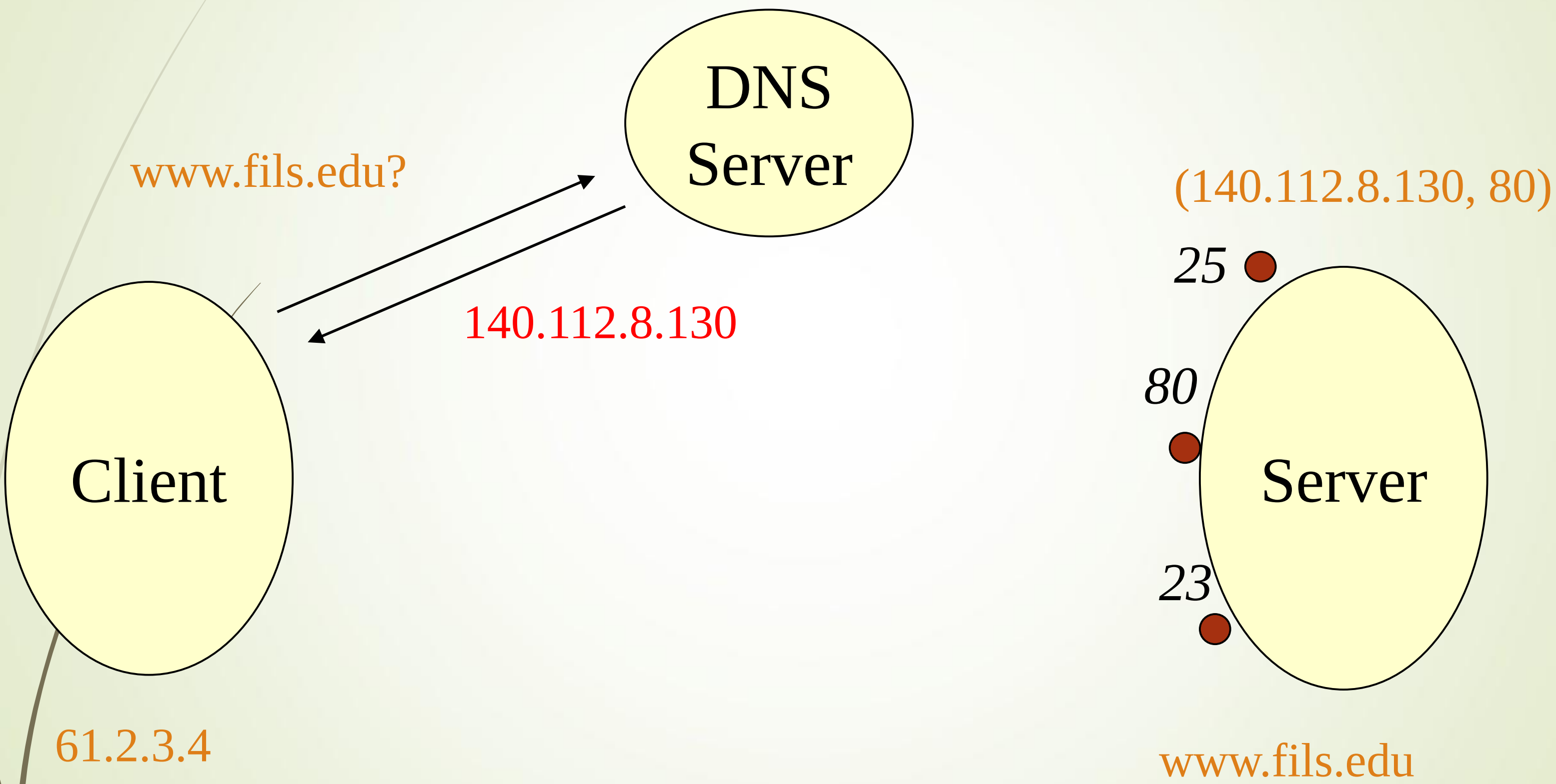
- *Well-known* ports: 1-1023
  - HTTP: 80
  - SMTP: 25
  - Telnet: 23
  - FTP: 21 (control), 20 (data)
- Others
  - Gnutella: 6346, 6347
- Client vs. server ports

Useful Tools

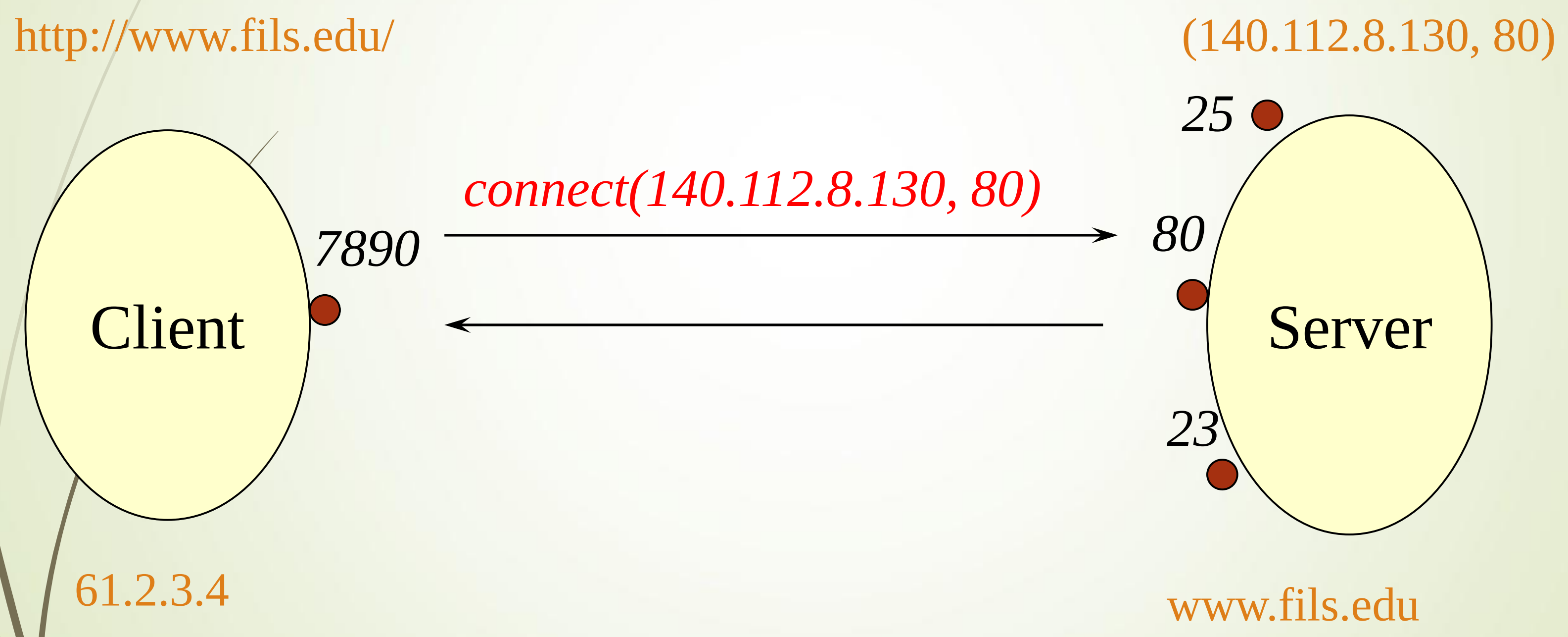
- Packet sniffer or analyzer
  - Tcpdump
  - Ethereal
  - NetXRay
- Packet generator
  - Socket programming
- Packet capture libraries
  - Libpcap & WinPcap

Example Scenario: Web Browsing

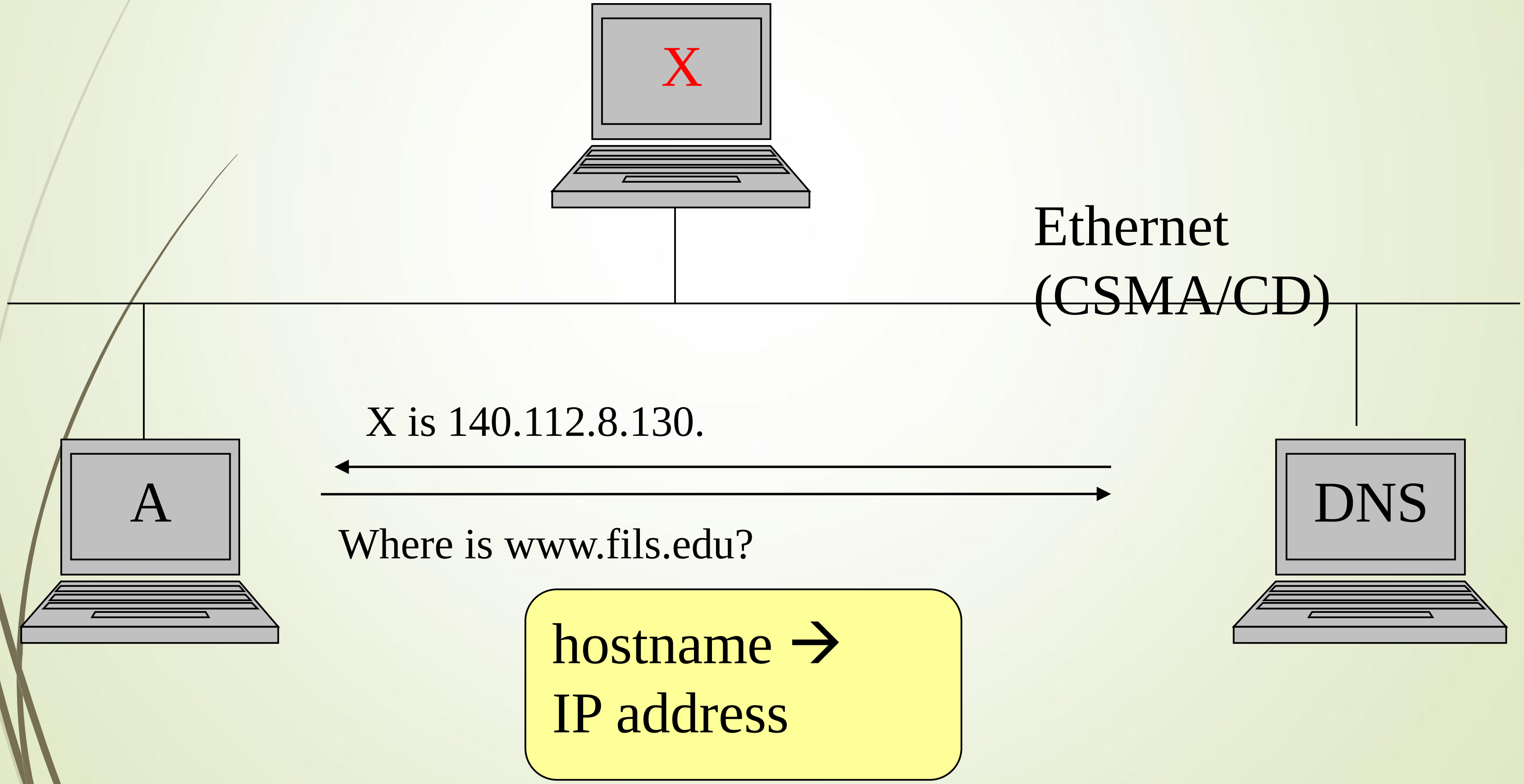
Example Scenario: Web Browsing



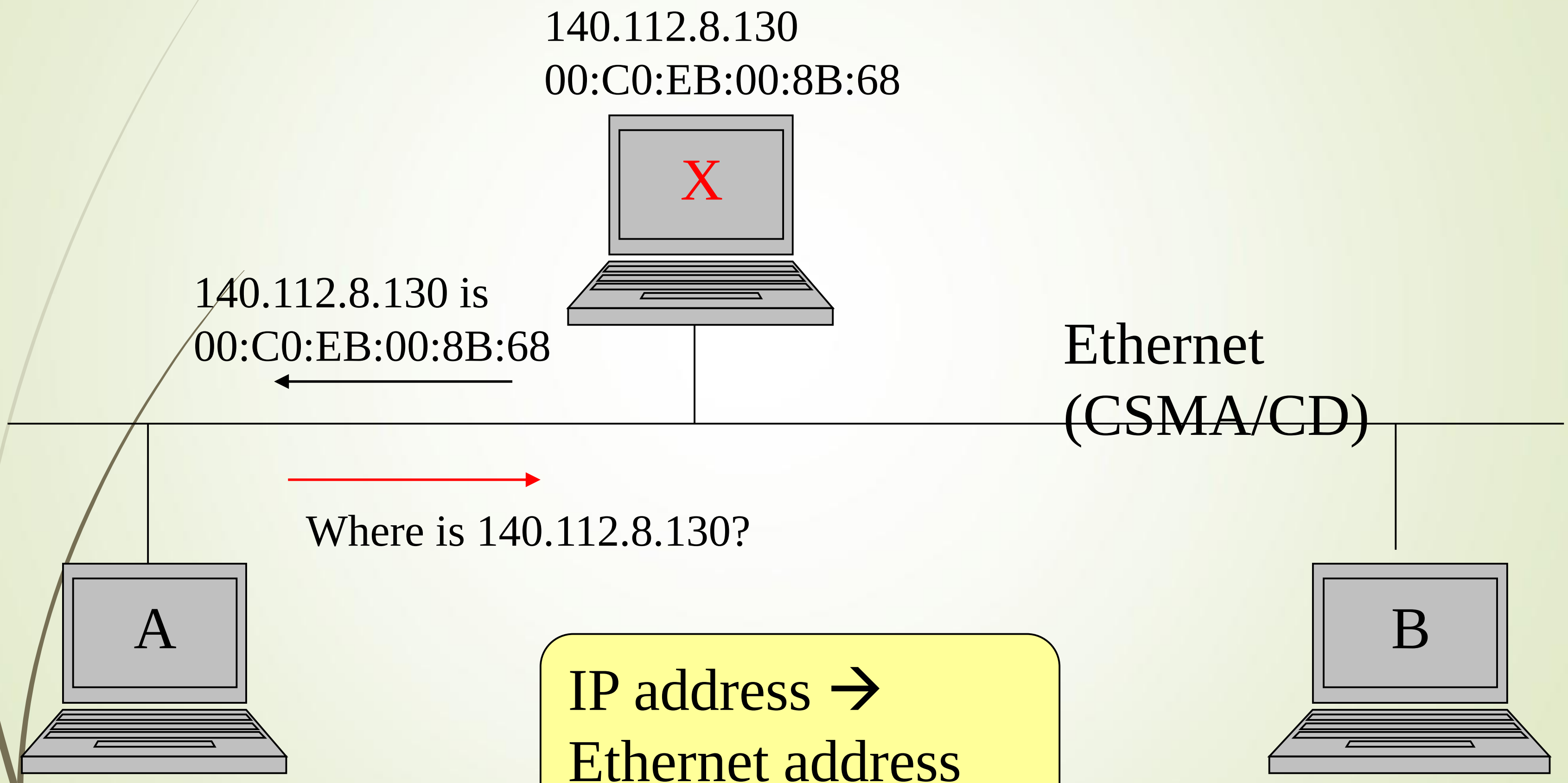
Example Scenario: Web Browsing



DNS Name Resolution



ARP (Revisited)

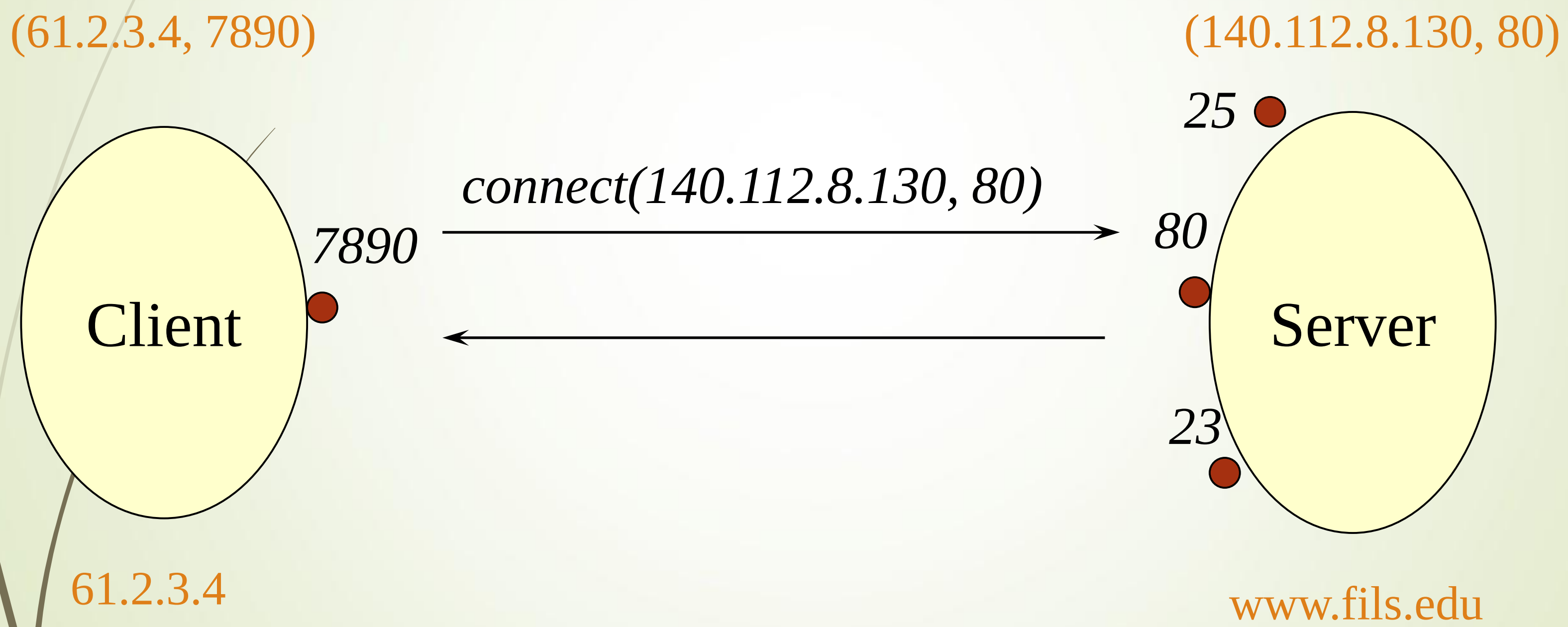


IP address → Ethernet address

Sockets

- ARP: Ethernet (hardware, MAC) address
- IP: IP address
- TCP/UDP: port number
- Port vs. service
- Sockets: $\{IP_{src}, port_{src}, IP_{dest}, port_{dest}\}$

Socket Connection



Socket Programming

- UNIX: BSD Socket API (in C)
  - socket(), bind(), listen(), accept(), connect(), send(), recv(), sendto(), recvfrom(), select(), …
- Java Socket API
  - java.net.Socket
- Perl, Python, …

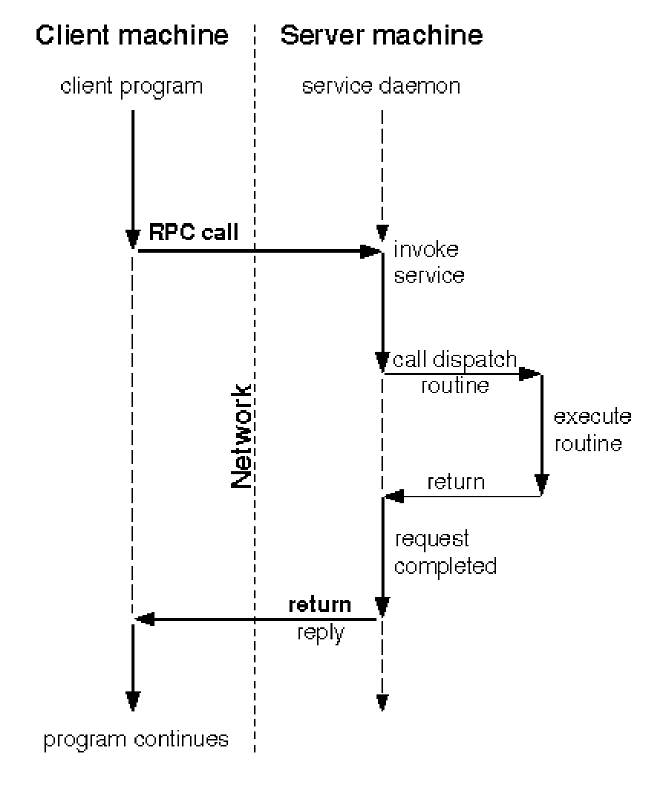
Remote Procedure Call

- RFC 1831 – RPC v2
- RFC 1832 -- XDR: External Data Representation Standard
  - A machine-independent representation of data
- Local vs. remote procedure calls

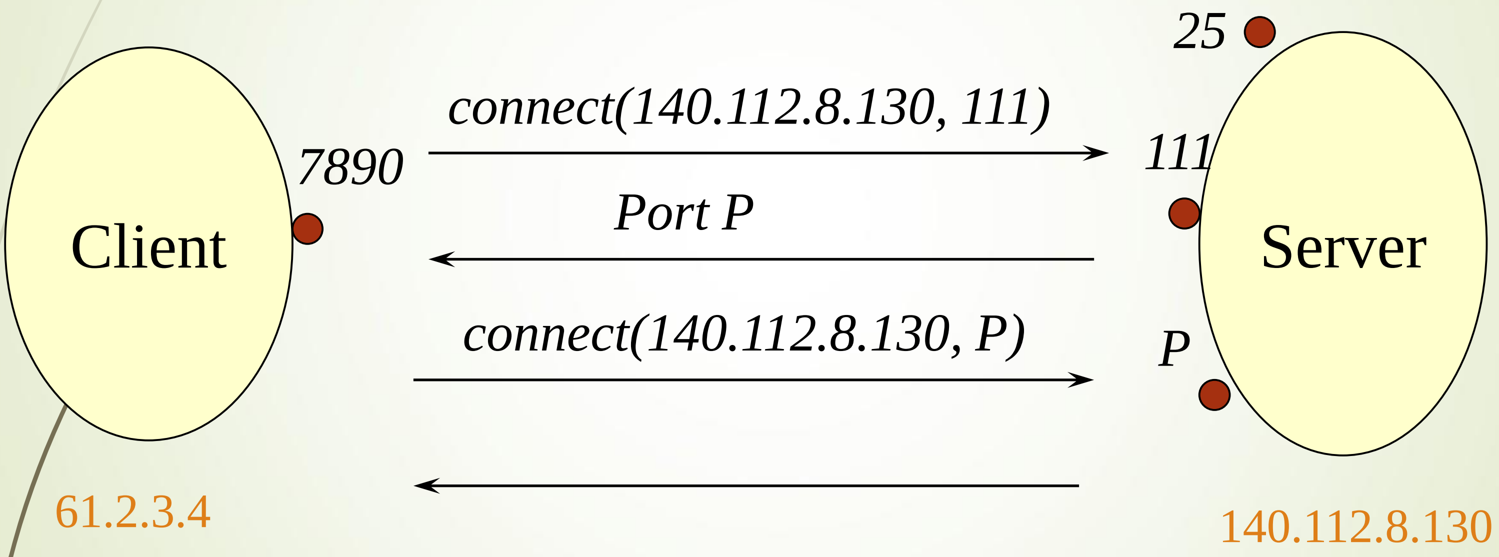
RPC

- UDP/TCP transport
  - RPC/UDP: connectionless, fast
  - RPC/TCP: connection-oriented, slower
- Portmap service (or *portmapper*)
  - Port 111
  - RFC 1833

RPC


Client machine
Server machine
client program
service daemon
RPC call
invoke service
call dispatch routine
execute routine
return
Network
request completed
return
reply
program continues

RPC Portmapping



RPC Programming

- *rpcgen*
- Applications: NFS (Network File System), …

Internet Layer

- Connectionless, point to point internetworking protocol (uses the datagram approach)
  - takes care of routing across multiple networks
  - each packet travels in the network independently of each other
    - they may not arrive (if there is a problem in the network)
    - they may arrive out of order
  - a design decision enforced by DoD to make the system more flexible and responsive to loss of some subnet devices
- Implemented in end systems and routers as the Internet Protocol (IP)

Transport Layer

- End-to-end data transfer
- Transmission Control Protocol (TCP)
  - connection oriented
  - reliable delivery of data
  - ordering of delivery
- User Datagram Protocol (UDP)
  - connectionless service
  - delivery is not guaranteed
- Can you give example applications that use TCP and UDP?

Application Layer

- Support for user applications
- A separate module for each different application
  - e.g. HTTP, SMTP, telnet

Python Tools: Sockets

- The **`socket`** module includes functions classes for implementing network connections via sockets
- The client and sever each create their own sockets and run methods to talk to each other

Getting the Host Name and IP

```
>>> import socket

>>> socket.gethostname()
'smalltalk'

>>> socket.gethostbyname(socket.gethostname())
'134.432.111.34'
```

Python Tools: Codecs

- Strings are transmitted as bytes, so they must be encoded before and decoded after transmission
- Strings are encoded and decoded using a ***codec***, as defined in the **`codecs`** module

Encoding and Decoding Strings

```
bytes(string, codec) -> an array of bytes

codecs.decode(byteArray, codec) -> a string
```

Consult the **codecs** doc for info on the possible codecs

```
>>> from codecs import decode

>>> data = bytes('Good luck on the final exam', 'ascii')

>>> print(decode(data, 'ascii')
Good luck on the exam!
```

The Role of the Server

- The server creates a socket and listens for requests from clients
- When a client request comes in, the server sends the appropriate response via the socket
- When the client disconnects, the server continues to listen for more requests

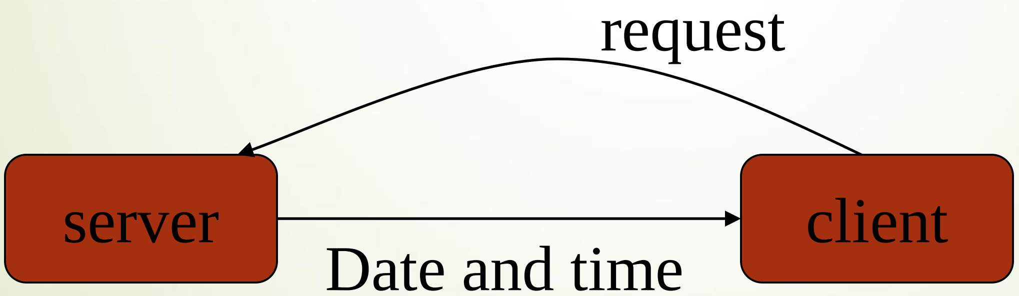
The Structure of a Server

```
Import resources

Set up and connect the server to the net

While True:
    Accept a connection from a client
    Process the request for service
```

A server runs forever, unless an exception is raised

Example: A Date/Time Server

- When a client connects, the server sends the current date and time
- When the client receives this information, it is displayed in the terminal



Example: A Day/Time Server

```
from socket import *
from time import ctime
```

The **`socket`** module includes resources for sockets

The **`ctime`** function returns the date and time

Example: A Day/Time Server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 21566
ADDRESS = (HOST, PORT)
```

A socket is associated with the host computer's IP address and a port number

These data are organized in a tuple

`localhost` supports a server and a client running on the same computer

Example: A Day/Time Server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 21566
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)
```

`socket` returns a socket object of the type specified by its arguments

`bind` and **`listen`** establish the socket's connection to the net and listen for client requests

Example: A Day/Time Server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 21566
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
```

`accept` pauses until a client connects

`accept` returns the client's socket and address information

Example: A Day/Time Server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 21566
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    client.send(bytes(ctime() + '\nHave a nice day!', 'ascii'))
    client.close()
```

`send` sends an encoded string to the client and **`close`** ends the connection

Example: A Day/Time Server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 21566
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    client.send(bytes(ctime() + '\nHave a nice day!', 'ascii'))
    client.close()

server.close()          # Never reached here, but useful if exception
                        # handling is added
```

Example: A Day/Time Client

```
from socket import *

HOST = 'localhost'
PORT = 21566
BUFSIZE = 1024
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
```

Setup code for a client socket is very similar to the code for a server socket

`BUFSIZE` (1 kilobyte here) indicates the number of bytes allowed for each input operation

Example: A Day/Time Client

```
from socket import *

HOST = 'localhost'
PORT = 21566
BUFSIZE = 1024
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.connect(ADDRESS)
```

`connect` connects this socket to the server at the specified address

Example: A Day/Time Client

```
from socket import *
from codecs import decode

HOST = 'localhost'
PORT = 21566
BUFSIZE = 1024
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.connect(ADDRESS)

dayAndTime = decode(server.recv(BUFSIZE), 'ascii')

print(dayAndTime)
server.close()
```

`recv` inputs an encoded string from the server (the date and time)

A One-on-One Chat Server

- When a client connects, send a greeting and wait for a reply
- When the reply is received, send another message
- An empty string/reply should disconnect the client

A One-on-One Chat Server

```
while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    client.send(bytes('Welcome to my chat room!', 'ascii'))

    while True:
        message = decode(client.recv(BUFSIZE), 'ascii')
        if not message:
            print('Client disconnected')
            client.close()
            break
        else:
            print(message)
            client.send(bytes(input('> '), 'ascii'))
```

Service includes a nested loop for carrying on the conversation

A One-on-One Chat Client

```
server = socket(AF_INET, SOCK_STREAM)
server.connect(ADDRESS)
print(decode(server.recv(BUFSIZE), 'ascii'))  # Displays server's
                                              # greeting

while True:
    message = input('> ')
    if not message:
        break
    server.send(bytes(message, 'ascii'))
    reply = decode(server.recv(BUFSIZE), 'ascii')
    if not reply:
        break
    print(reply)
server.close()
```

Client now has a loop to carry on the conversation

Loop ends when the client sends or receives `''`

Putting the Doctor Online

- Very similar to a one-on-one chat, but the server responds by using a **`Doctor`** object's reply instead of a human being's input

- Minor changes to the chat server, but no changes at all to the chat client!

A One-on-One Chat Server

```
while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    client.send(bytes('Welcome to my chat room!', 'ascii'))

    while True:
        message = decode(client.recv(BUFSIZE), 'ascii')
        if not message:
            print('Client disconnected')
            client.close()
            break
        else:
            print(message)
            client.send(bytes(input('> '), 'ascii'))
```

Service includes a nested loop for carrying on the conversation

A One-on-One Therapy Server

```
while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    dr = Doctor()
    client.send(bytes(dr.greeting(), 'ascii'))

    while True:
        message = decode(client.recv(BUFSIZE), 'ascii')
        if not message:
            print('Client disconnected')
            client.close()
            break
        else:
            client.send(bytes(dr.reply(message), 'ascii'))
```

Create the appropriate “bot” for carrying out the server’s side of the conversation

Going “Live”: the Server

```
from socket import *
from time import ctime

HOST = gethostbyname(gethostname())
PORT = 21566
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

while True:
    print('Waiting for connection . . . ')
    client, address = server.accept()
    print('... connected from:', address)
    client.send(bytes(ctime() + '\nHave a nice day!', 'ascii'))
    client.close()
```

Can deploy this server on any machine with an IP address

Going “Live”: the Client

```
from socket import *
from codecs import decode

HOST = input('Enter the server name: ')
PORT = 21566
BUFSIZE = 1024
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.connect(ADDRESS)

dayAndTime = decode(server.recv(BUFSIZE), 'ascii')

print(dayAndTime)
server.close()
```

The HOST must be the name or IP of the server