Lecture 2-bis.

Internet Transport Protocols

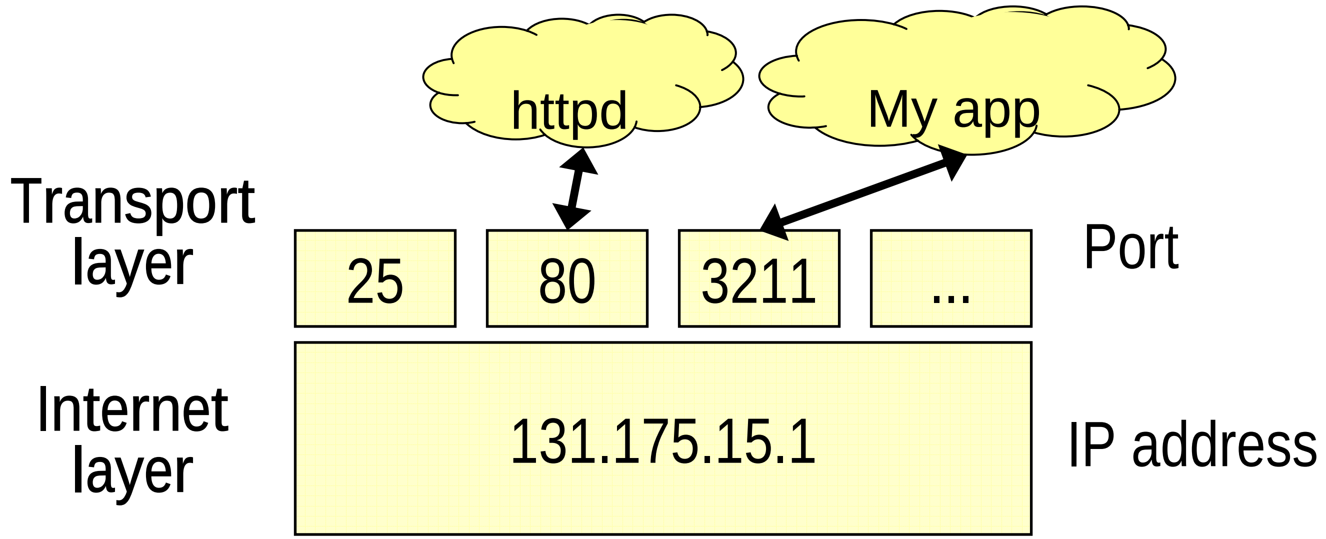
As seen by the application developer point of view

G.Bianchi, G.Neglia, V.Mancuso

Università degli Studi di Palermo

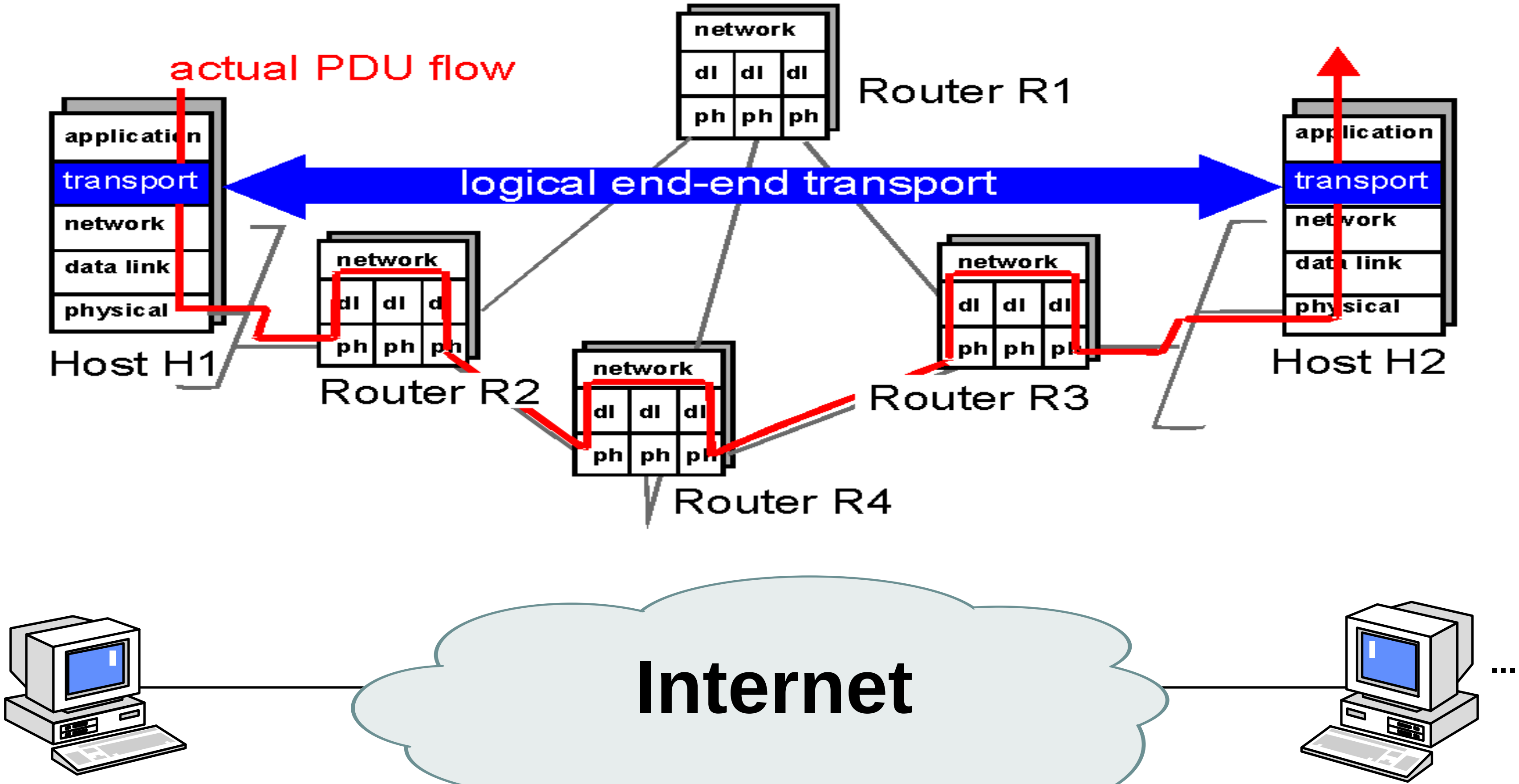
The primary (in principle unique) role of Internet transport protocols

- ➔ Extend IP's delivery service (between two end systems) to a deliver service between two APPLICATION PROCESSES running on the end systems
- ➔ MAPPING to OSI language:
  - ⇨ Port number = TSAP (Transport Service Access Point)
  - ⇨ IP address = NSAP (Network Service Access Point)

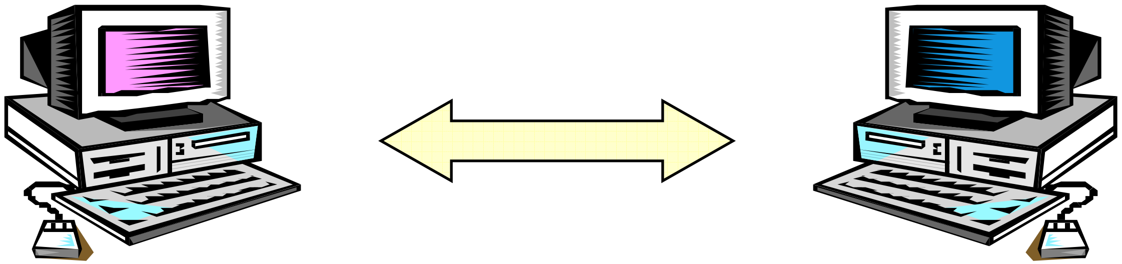


Transport Layer Protocols

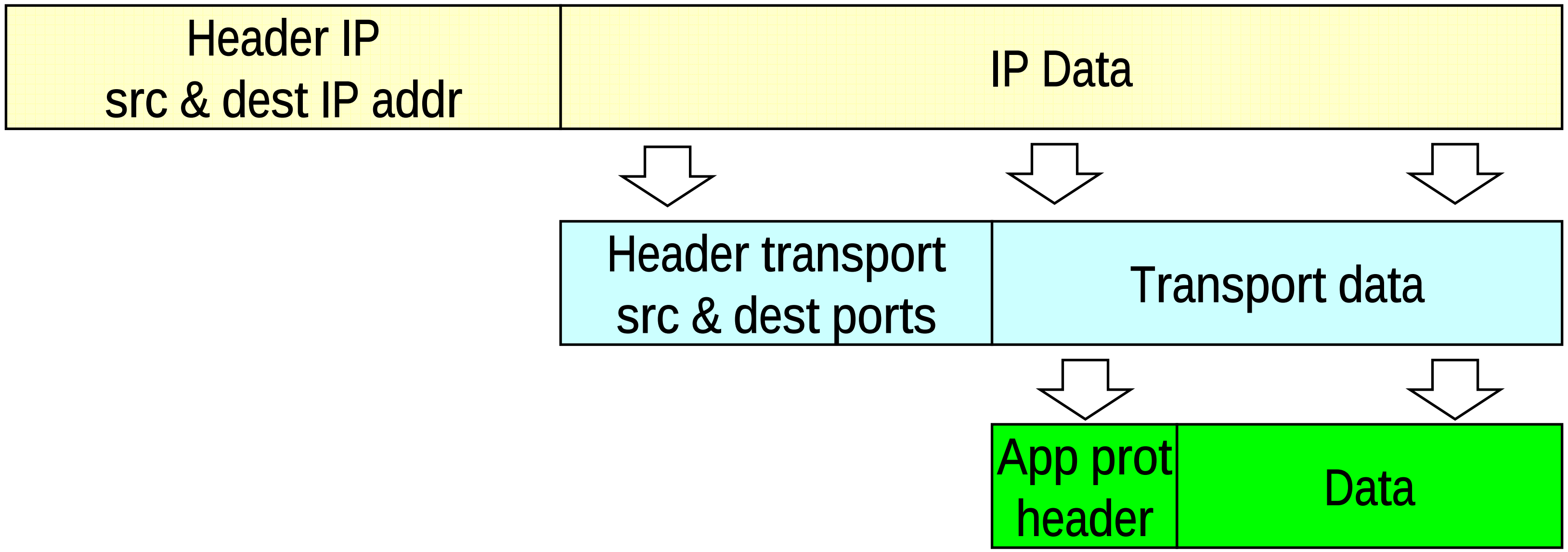
Entire network seen as a pipe



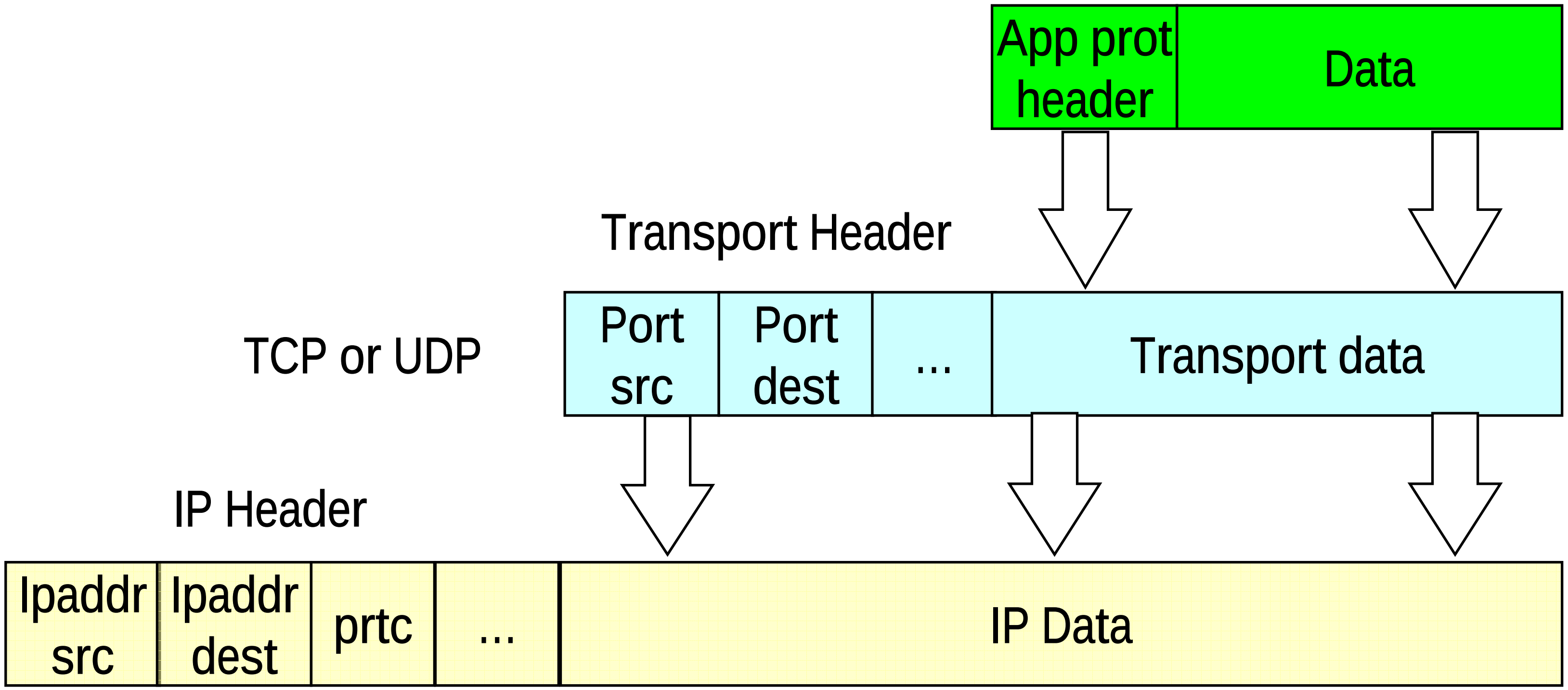
The Internet level view



Information units travelling in the network: IP packets

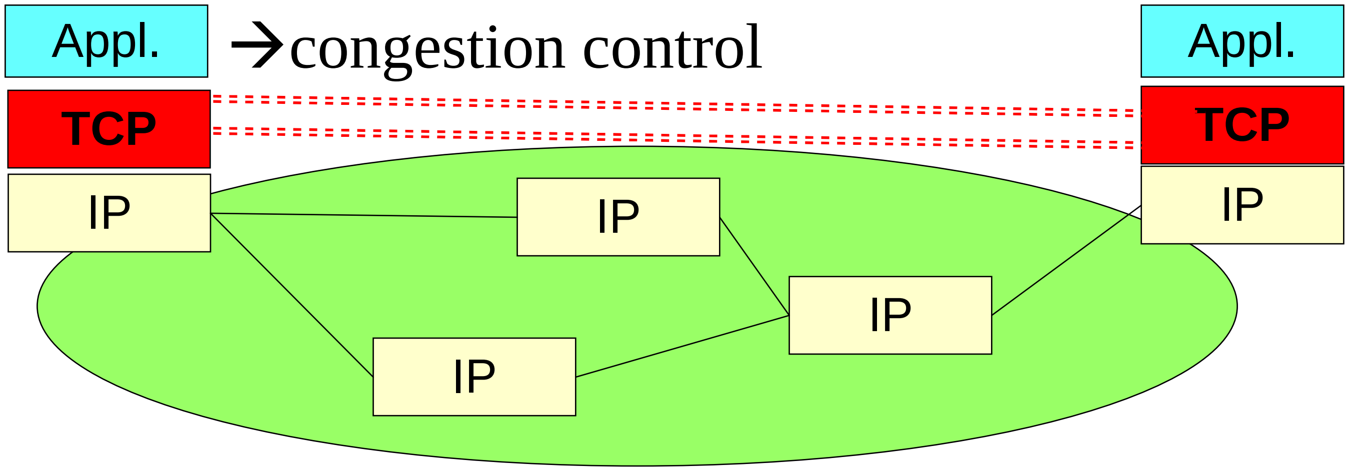


Where are port numbers?



Transport Control Protocol (TCP)

- ➔**connection oriented**
  - ⇨TCP connections
- ➔***reliable* transfer service.**
- ➔**TCP functions**
  - →application addressing (ports)
  - →error recovery (acks and retransmission)
  - →reordering (sequence numbers)
  - →flow control
  - →congestion control



Services Provided by TCP

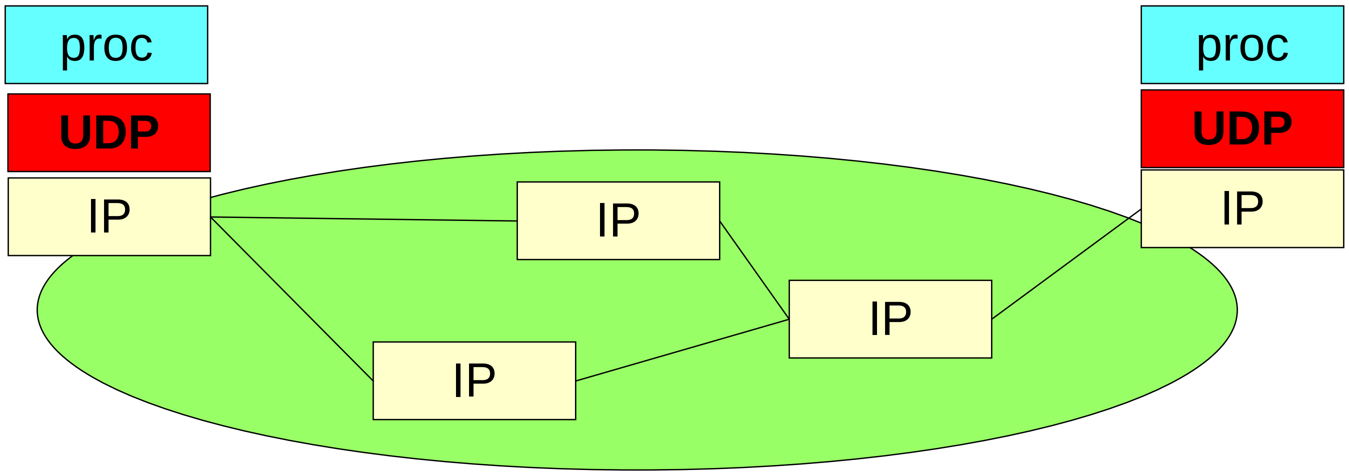
- ⇨Connection-oriented service: preliminary handshaking procedure creates a full duplex TCP connection
- ⇨Reliable transport service: communicating processes can rely on TCP to deliver all the messages sent *without error and in the proper order*.

➔**TCP *does not* provide:**

- ⇨a minimum transmission rate guaranteed (sending rate is regulated by TCP congestion control)
- ⇨any sort of delay guarantees (the World Wide Wait …)

User Datagram Protocol (UDP)

- ➔ **Connectionless**
  - ⇨ UDP packets
- ➔ **offers *unreliable* transfer service (*send and pray*).**
- ➔ **UDP functions**
  - → application addressing (ports)
  - → error checking



Services Provided by UDP

- ⇨connectionless (no handshaking)
- ⇨arbitrary sending rate service
  - » no congestion control mechanism present

➔**UDP minimalist lightweight service model *does not provide:***

- ⇨any guarantee of reception, any guarantee of order
- ⇨any guarantee on delay

UDP

- **Connectionless**
  - UDP packets
- ***unreliable* transfer service**
  - send and pray
- **UDP functions**
  - application addressing (ports)
  - error checking

TCP

- **connection oriented**
  - TCP connections
- ***reliable* transfer service**
  - all bytes sent are recv
- **TCP functions**
  - application addressing (ports)
  - error recovery (acks and retransmission)
  - reordering (sequence numbers)
  - flow control
  - congestion control

Service Requirements

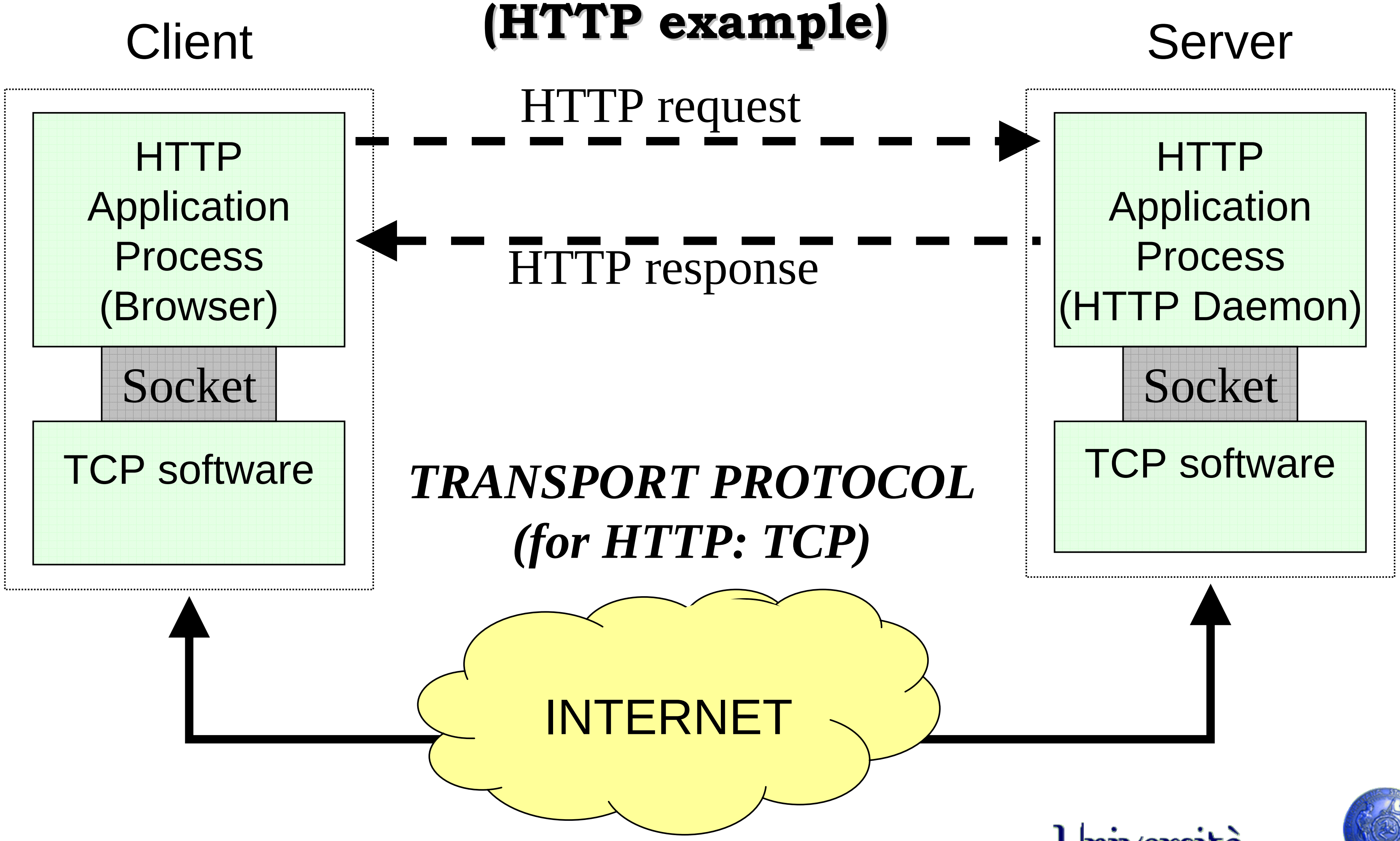
Application	Data Loss	Bandwidth	Time sensitive?
file transfer	no loss	elastic	no
electronic mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: few Kbps to 1Mbps video: 10's Kbps to 5 Mbps	yes: 100's of msec
stored audio/video	loss-tolerant	same as interactive audio/video	yes: few seconds
interactive games	loss-tolerant	few Kbps to 10's Kbps	yes: 100's msecs
financial applications	no loss	elastic	yes and no

Common Applications and related transport

Application	Application-layer protocol	Underlying Transport Protocol
electronic mail	SMTP (RFC 821)	TCP
remote terminal access	Telnet (RTC 854)	TCP
Web	HTTP (RFC 2068)	TCP
file transfer	FTP (RFC 959)	TCP
remote file server	NFS	UDP or TCP
streaming multimedia	Proprietary (e.g., Real Networks)	UDP or TCP
Internet telephony	proprietary (e.g. Vocaltec)	typically UDP

A closer look at applications: The Socket Interface

(HTTP example)

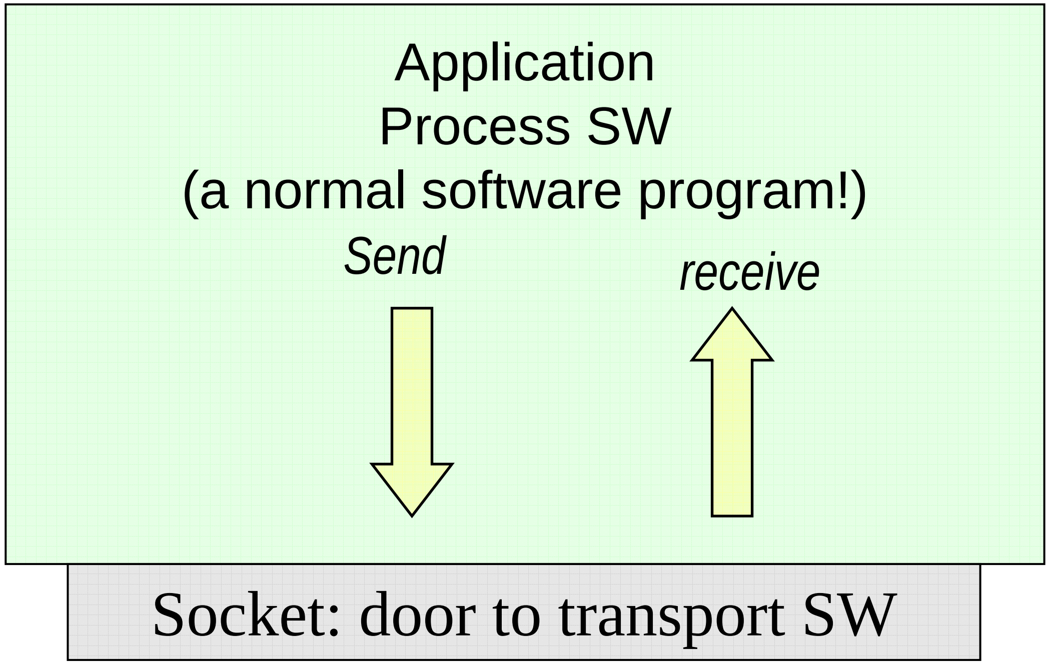


Sockets in Unix OSs

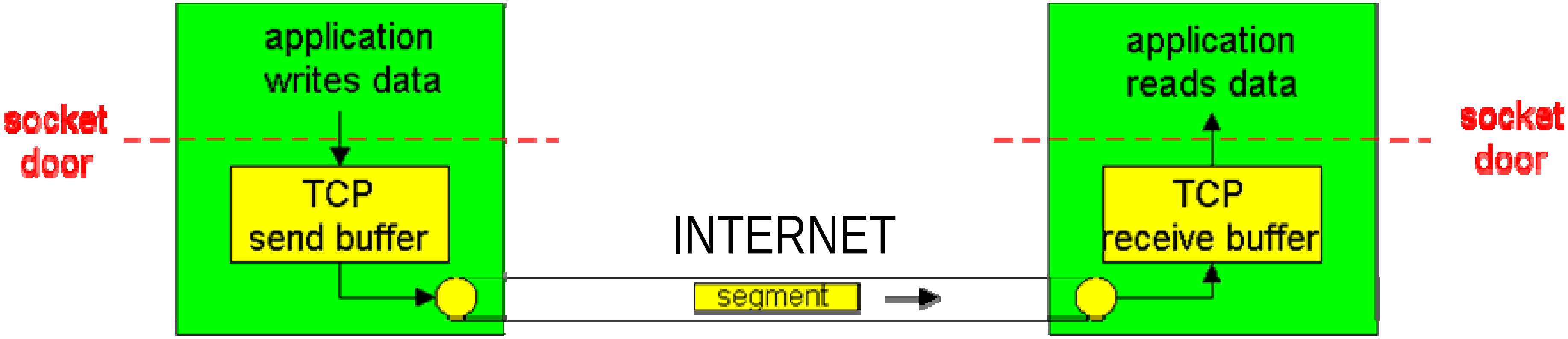
- ➔**Just file descriptors (everything is a file in Unix!)**
- ➔**"stream sockets" using TCP**
- ➔**"datagram sockets" using UDP**
- ➔**Common I/O file functions:** read(), write()
- ➔**More powerful I/O functions:** send(), receive()
- ➔**Other specific function:** socket(), bind(), connect(), listen(), accept()

The application developer view

- ➔the only mean for apps to send/receive messages is through sockets
- ➔"doors" that hide transportation infrastructure to processes
- ➔Very limited control on transport protocol (buffer sizing, variables)



Why it is trivial (!) to write networking apps?



- ➔ **Application software duties:**
  - ⇨ open socket (e.g. C, C++, JAVA function call, OS call, external library primitive)
  - ⇨ Injects message in its own socket
  - ⇨ being confident message is received on the other side
- ➔ **TCP software: in charge of managing segments!**
  - ⇨ reliable message transport when TCP used
  - ⇨ Segmentation performed by TCP transmitter
  - ⇨ Receive buffer necessary to ensure proper packet's order & reassembly

An open question

- ➔ **Socket: OS interface between the application and the transport level**
- ➔ **Ports: numbers in the transport header to identify the specific application**
- ➔ **Which is the relation?**
- ➔ **We focus on the server**

A first hypothesis

➔ **one-to-one mapping**

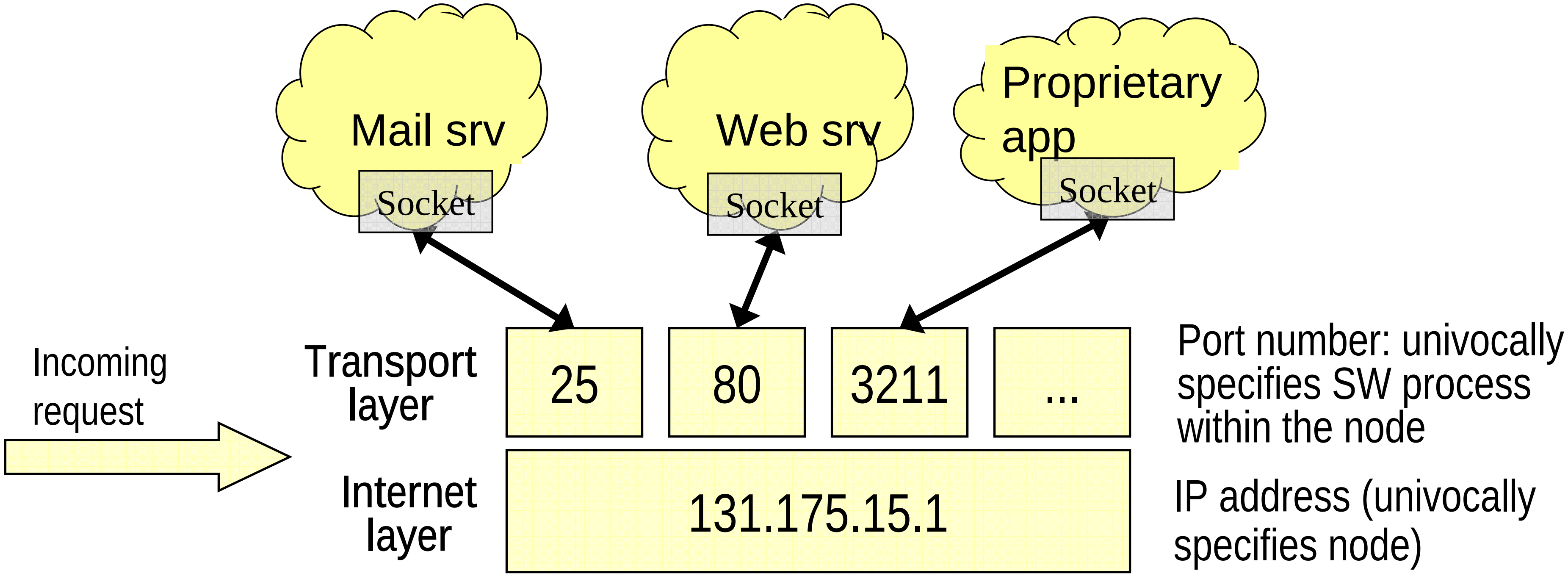
socket ⇄ port #

Trivial refinement

- ➔**The socket is on a specific host,**
- ➔**i.e. port# has a local meaning**

socket ⇄ (IP address,port #)

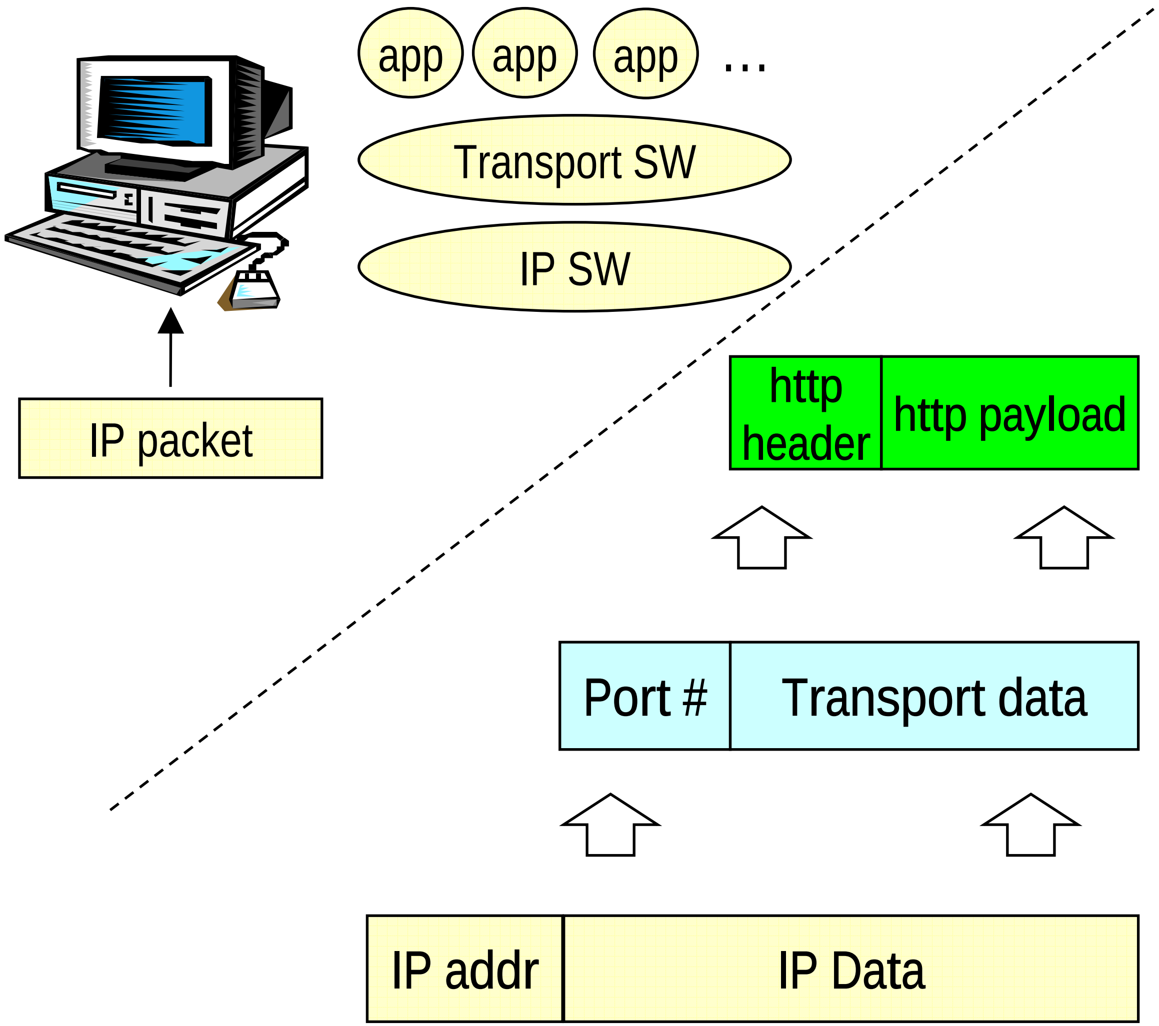
How to reach server socket:
pair of IP Address and Port Number



➔*(IP,port): a unique identification of an application layer service to which requests need to be sent*

➔*The first contact needs well known port #*

Demultiplexing at receiver (1)

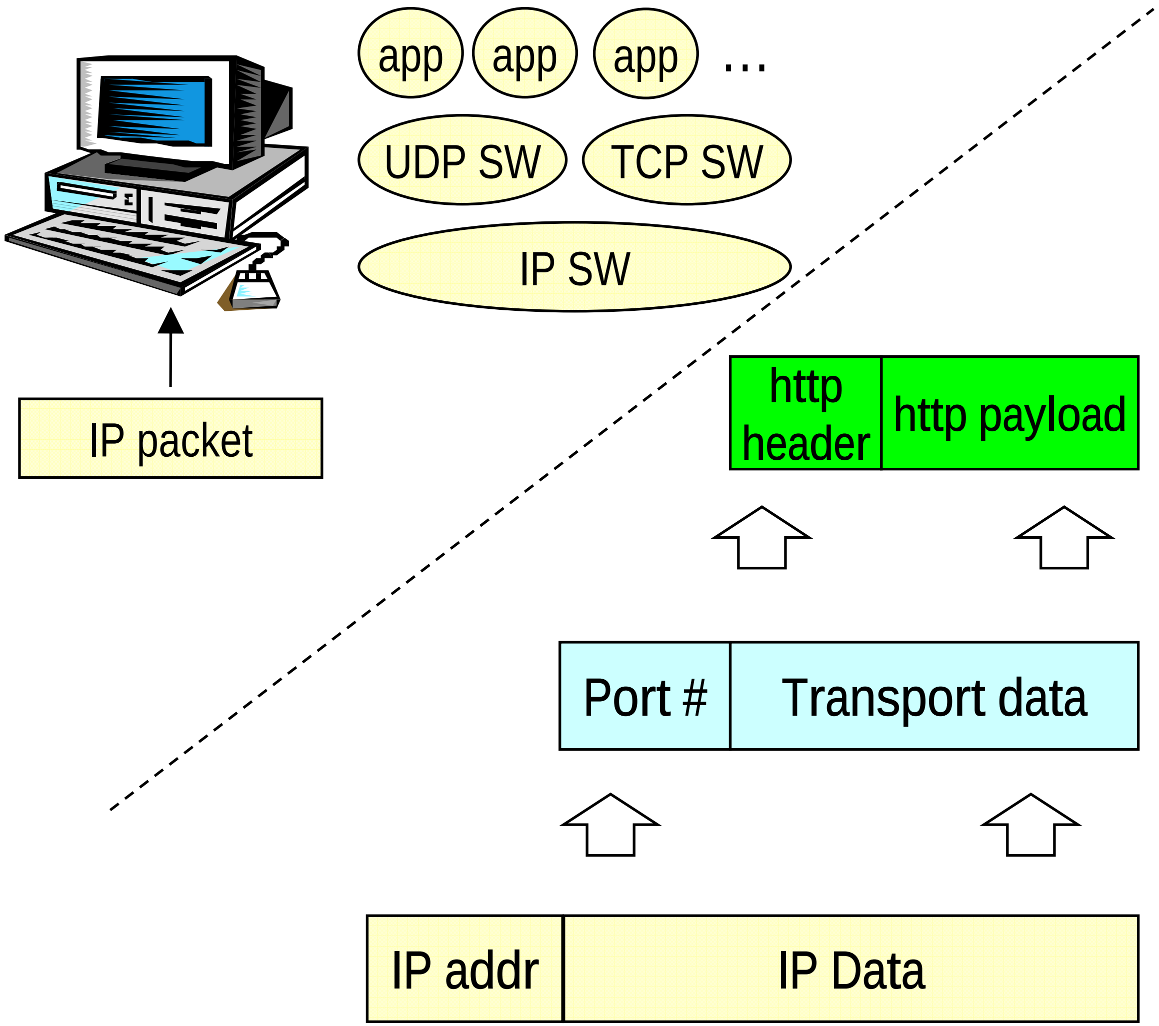


Information entering app Software (managed by app Developer)

Transport SW: checks segment; Sends to application sw based on Port number
Application demux

IP SW: checks IP packet; Sends to transport sw
Transport demux

Demultiplexing at receiver (2)

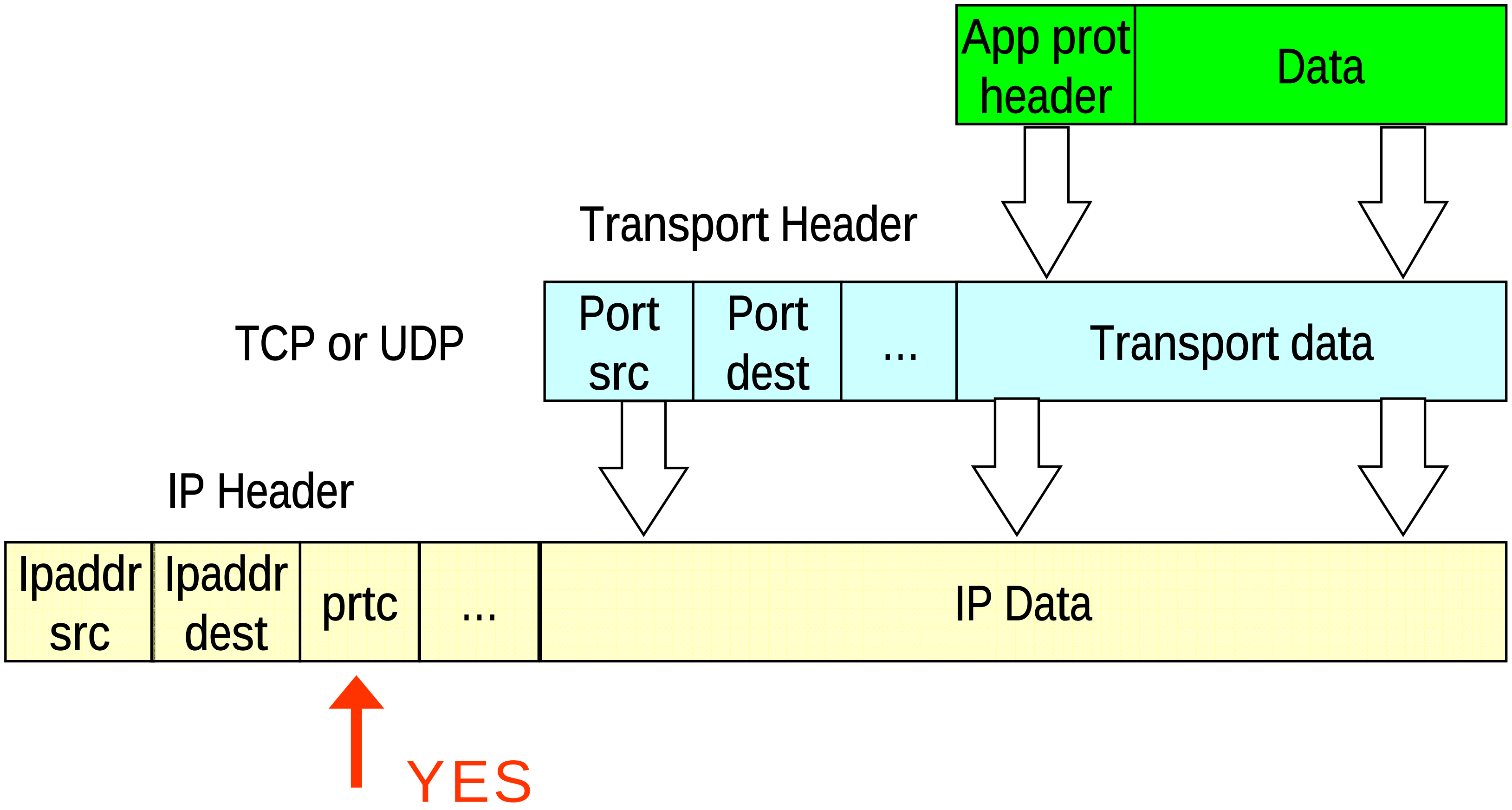


Information entering app Software (managed by app Developer)

Transport SW: checks segment; Sends to application sw based on Port number
Application demux

IP SW: checks IP packet; Sends to transport sw
selects whether UDP or TCP
Transport demux

Does IP software know about transport protocol?



Remarks

- ➔ **When opening socket, needs to specify which transport to use!**
- ➔ **UDP port numbers are independent from TCP ones!**
  - ⇨ This means that TCP looks at TCP ports, while UDP looks at UDP ones
- ➔ **Normally (for pure convenience) port number = same meaning for TCP and UDP**
  - ⇨ if a well known service is offered by both TCP and UDP, the port number is the same
  - ⇨ if a well known (low port number) service is offered for one protocol only, the corresponding port for the other protocol is generally unused
- ➔ **BUT possibly the same port number has different meaning for TCP and UDP....**
  - ⇨ Details in RFC1700 or http://www.iana.org/assignments/port-numbers

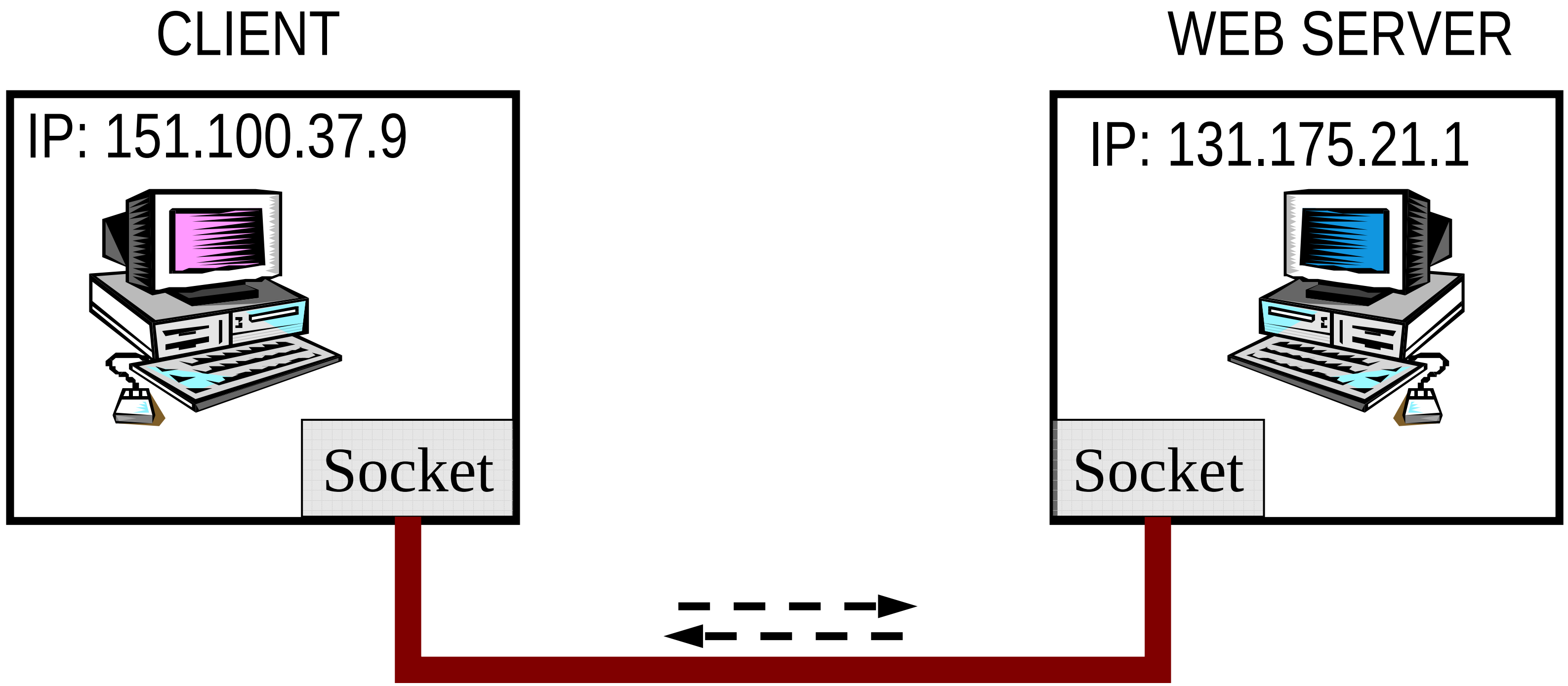
Consequence

- ➔ **If two applications employ different protocols, they can employ the same port #**
- ➔ **Mapping refinement**

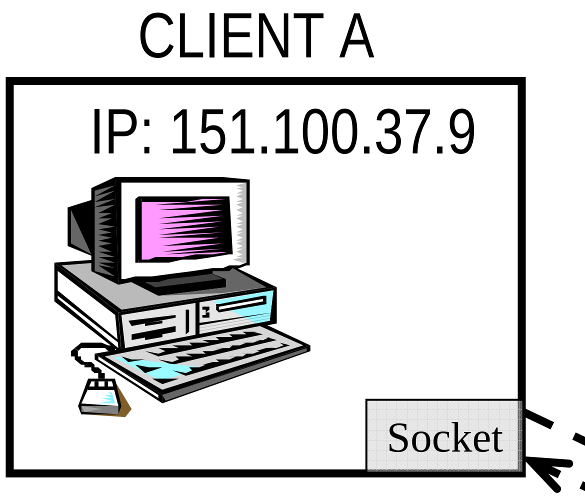
socket ⇄ (protocol,IP addr., port #)

- ➔ **Is it enough? Not always**

(TCP) Connections
identified by sockets at its ends

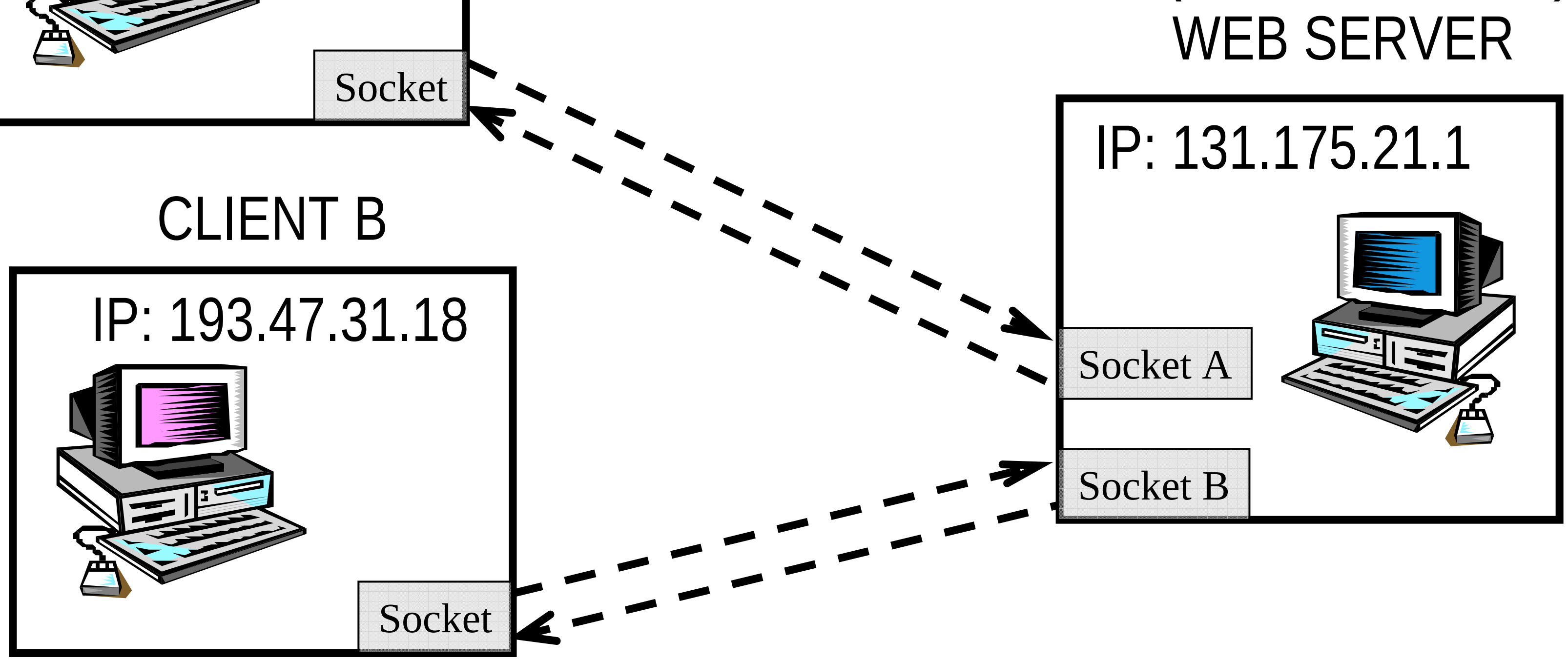


Managing multiple connections



➔*A socket for each connection is needed, because each connection has its own status*

➔ *It is not true for UDP (connectionless)*

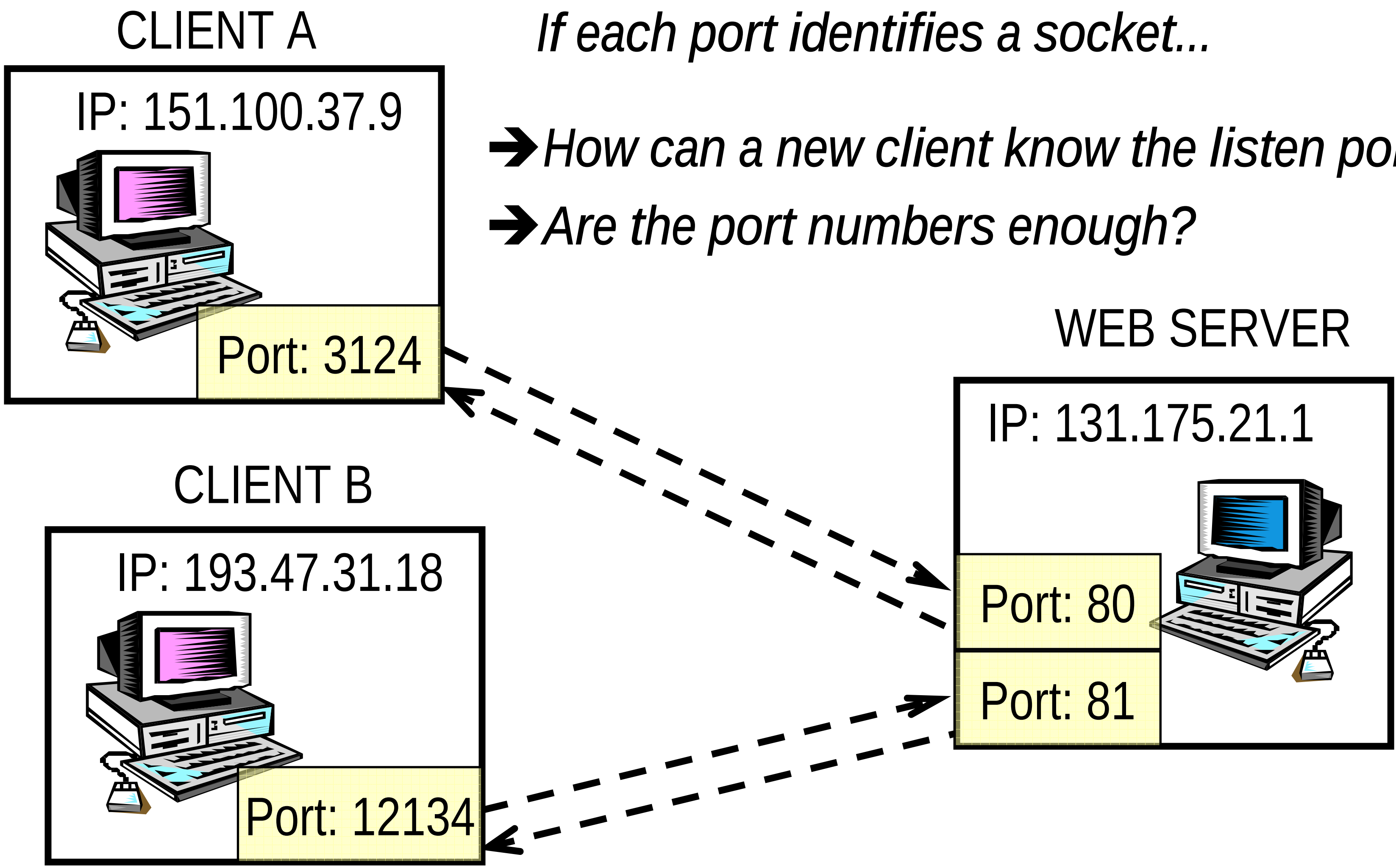


Managing multiple connections

If each port identifies a socket...

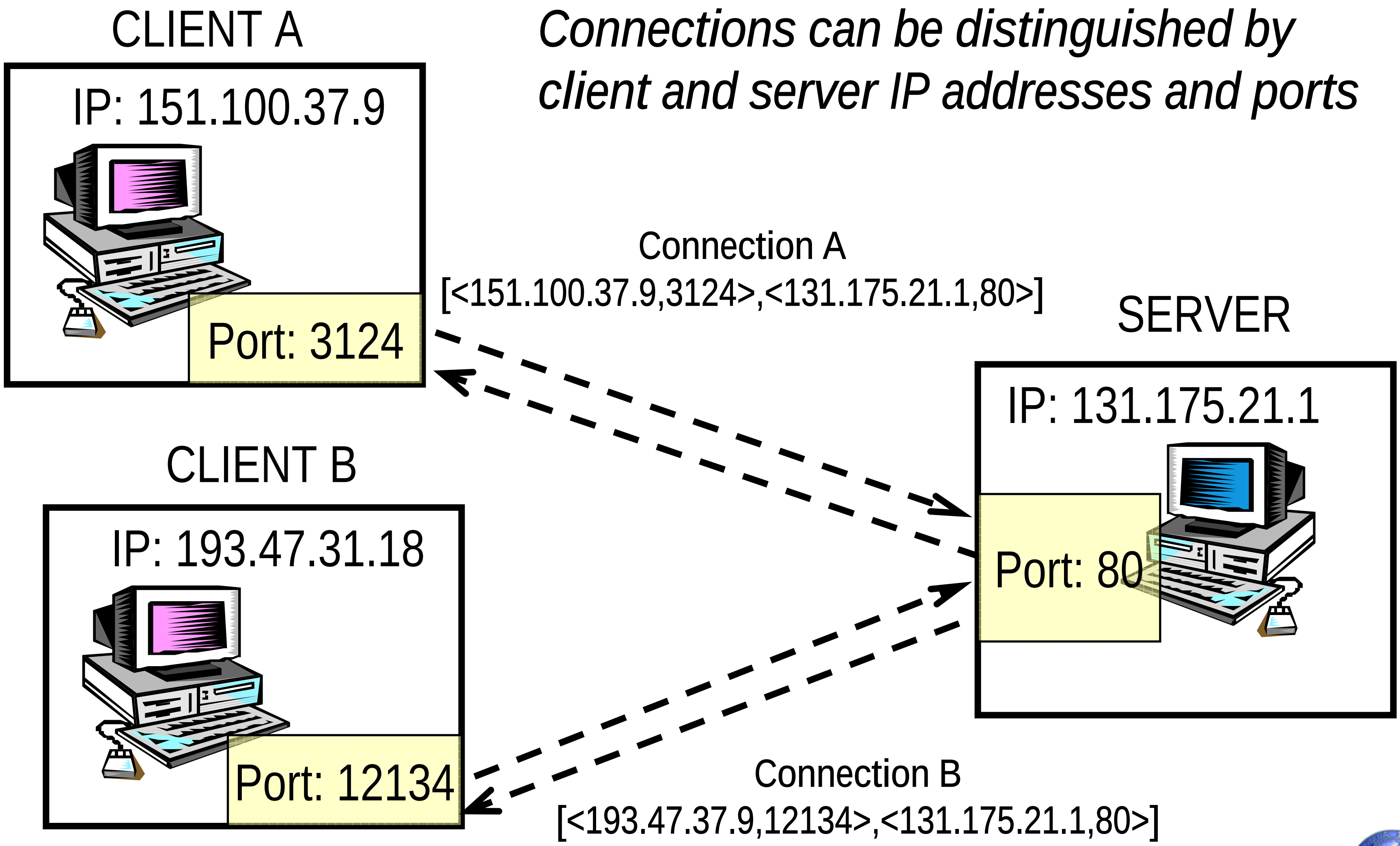
➔ *How can a new client know the listen port?*

➔ *Are the port numbers enough?*

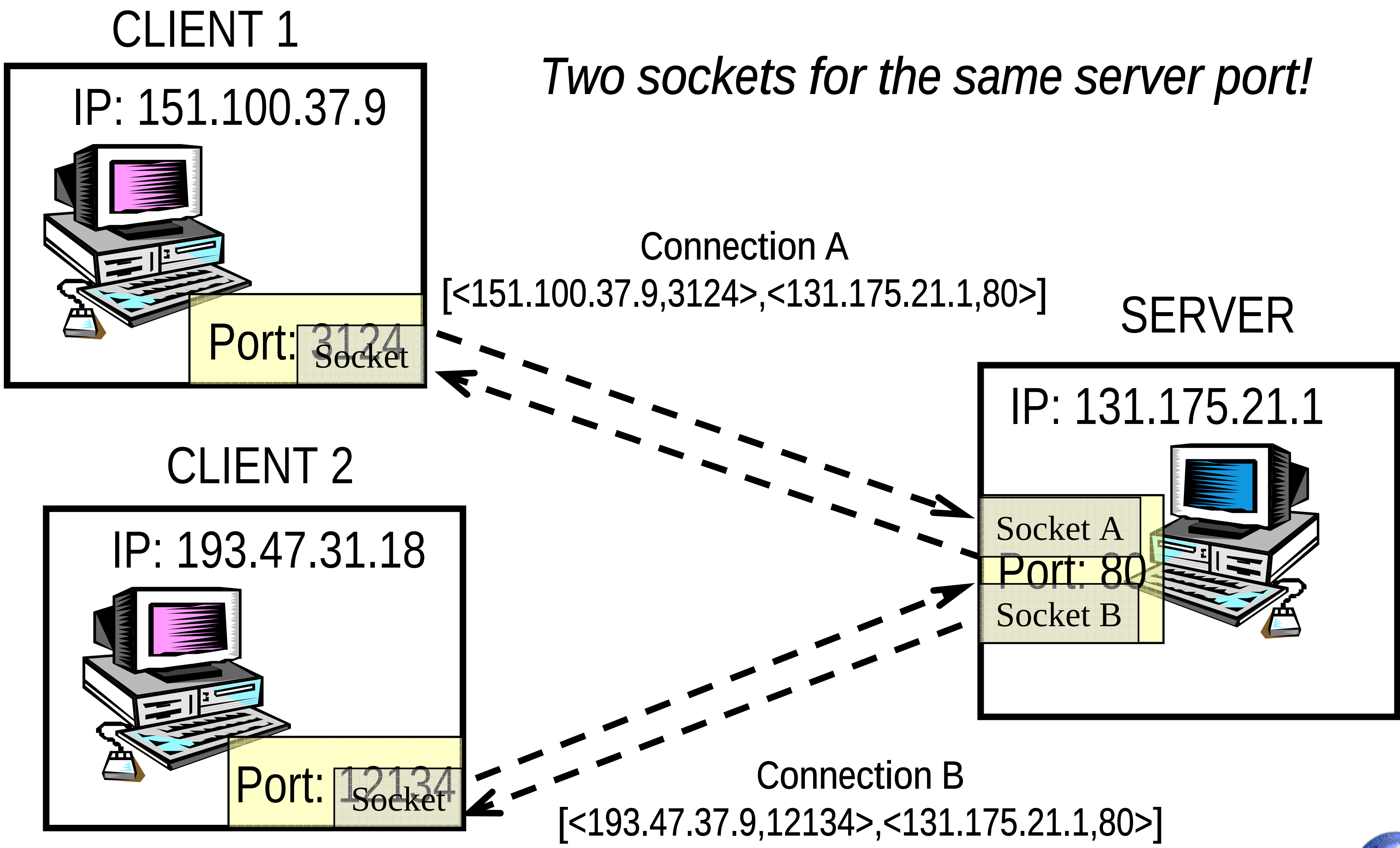


Managing multiple connections

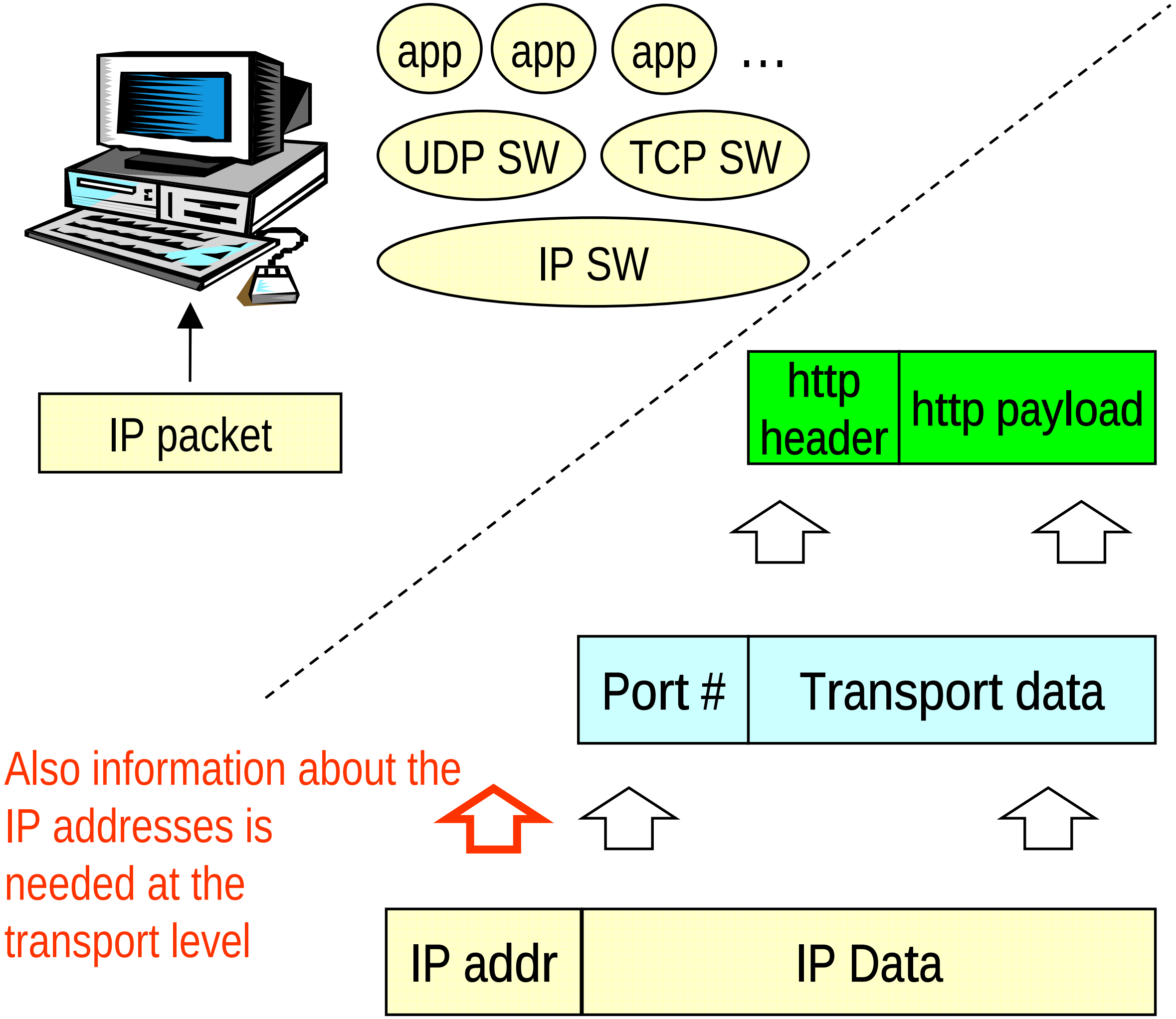
Connections can be distinguished by client and server IP addresses and ports



Managing multiple connections



Demultiplexing at receiver (3)



Also information about the IP addresses is needed at the transport level

Information entering app Software (managed by app Developer)

Transport SW: checks segment; Sends to application sw based on IP addresses and Port numbers
Application demux

IP SW: checks IP packet; Sends to transport sw selects whether UDP or TCP
Transport demux

Conclusions

- ➔**A socket always identifies unique protocol and port**

socket ⟶ (protocol,IP addr, port #)

- ➔**It <u>can</u> identify also address and port of the remote application**

socket ⟶ (prot, src IP addr, src port, dest IP addr, dest port)

Conclusions

➔**Protocol and port <u>can</u> identify a unique socket**

socket ⟵ (protocol,IP addr, port #)

<u>listen ports/sockets</u>

➔**but in general more information is required**

socket ⟵ (prot., src IP addr, src port, dest IP addr, dest port)

<u>connection ports/sockets</u>

How to reach client socket

another pair of IP Address and Port Number

- ➔ **The server needs to know:**
  - ⇨ The host to which send a response
    - → src IP address
  - ⇨ The application software process at client side capable of correctly interpret the response
    - → src port #
- ➔ **Generally client DOES NOT use a well known port #**
  - ⇨ It is not needed (the client starts talking)
  - ⇨ OS just assigns one available (Ephemeral ports)

Typical question: WHY every PC needs an IP address?
More complex issue: HOW your home PC gets an IP address?

Port numbers

- **16 bit address (0-65535)**
- **well known port numbers for common servers**
  - FTP 20, TELNET 23, SMTP 25, HTTP 80, POP3 110, … (full list: http://www.iana.org/assignments/port-numbers)
- **number assignment**
  - 0-1023 (system) well known ports: service contact ports assigned by IANA, on most systems they can only be used by system (or root) processes or by programs executed by privileged users.
  - 1024-49151 (user) registered ports: service contact ports listed by IANA, on most systems they can be used by ordinary user processes or programs executed by ordinary users.
  - 49152-65535 dynamic/private ports.

Last remark about terminology

➔ **Sometimes socket is considered synonym of the quintet:**

(prot., src IP addr, src port,

dest IP addr, dest port)

Ipaddr src	Ipaddr dest	prtc	…	Port src	Port dest	…	Data