DIPARTIMENTO DI INFORMATICA
Via Salaria 113 - 00198, Roma

SAPIENZA UNIVERSITÀ DI ROMA

Chapter 2 Application Layer

Reti di Elaboratori

Corso di Laurea in Informatica

Università degli Studi di Roma "La Sapienza"

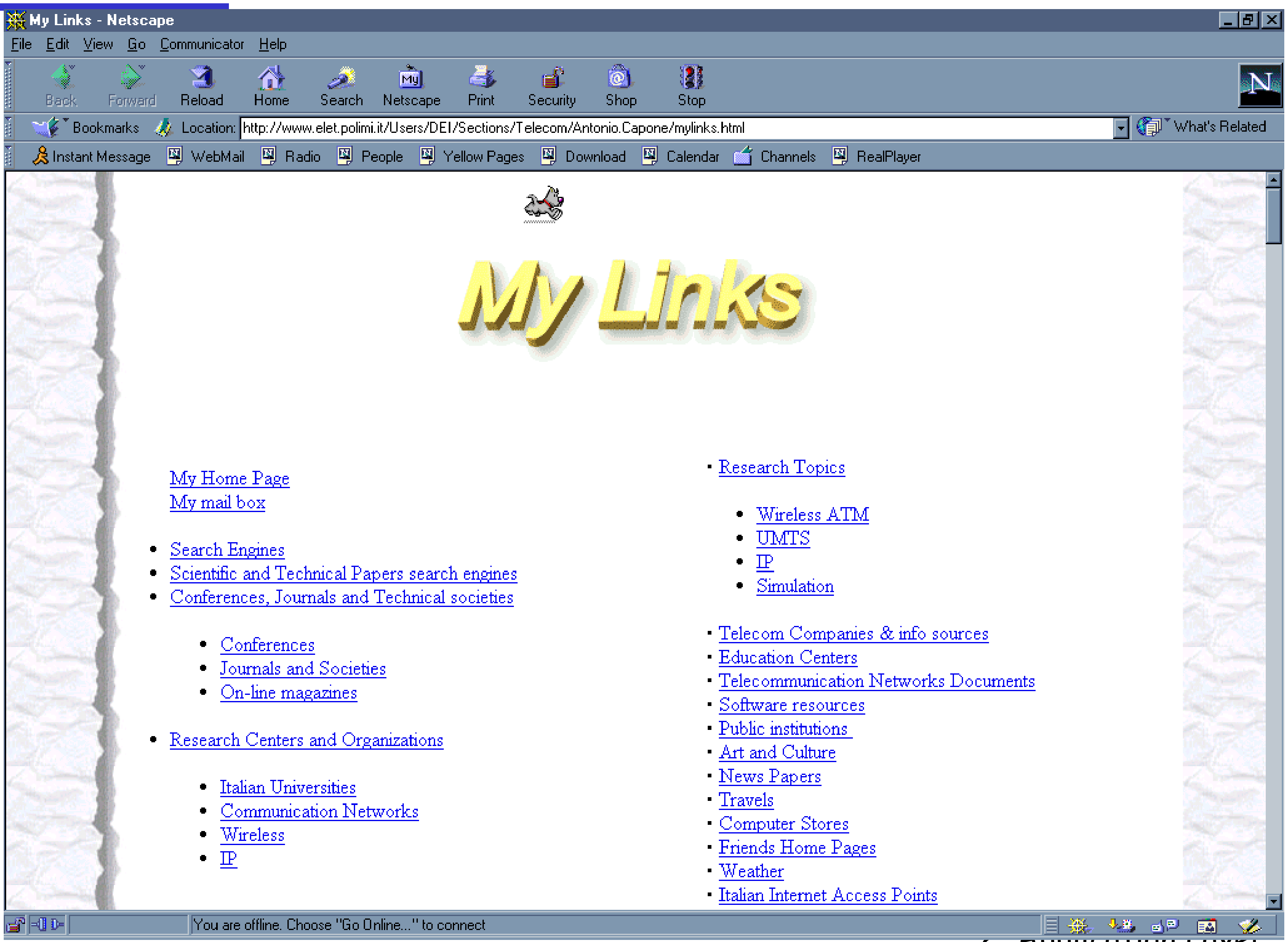
Canale A-L

Prof.ssa Chiara Petrioli

Chapter 2 outline

- 2.1 Principles of app layer protocols
  - clients and servers
  - app requirements
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.9 Content distribution
  - Network Web caching
  - Content distribution networks
  - P2P file sharing

I servizi di IP: esempio il WEB


My Links - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.elet.polimi.it/Users/DEI/Sections/Telecom/Antonio.Capone/mylinks.html
What's Related
Instant Message WebMail Radio People Yellow Pages Download Calendar Channels RealPlayer
You are offline. Choose "Go Online..." to connect

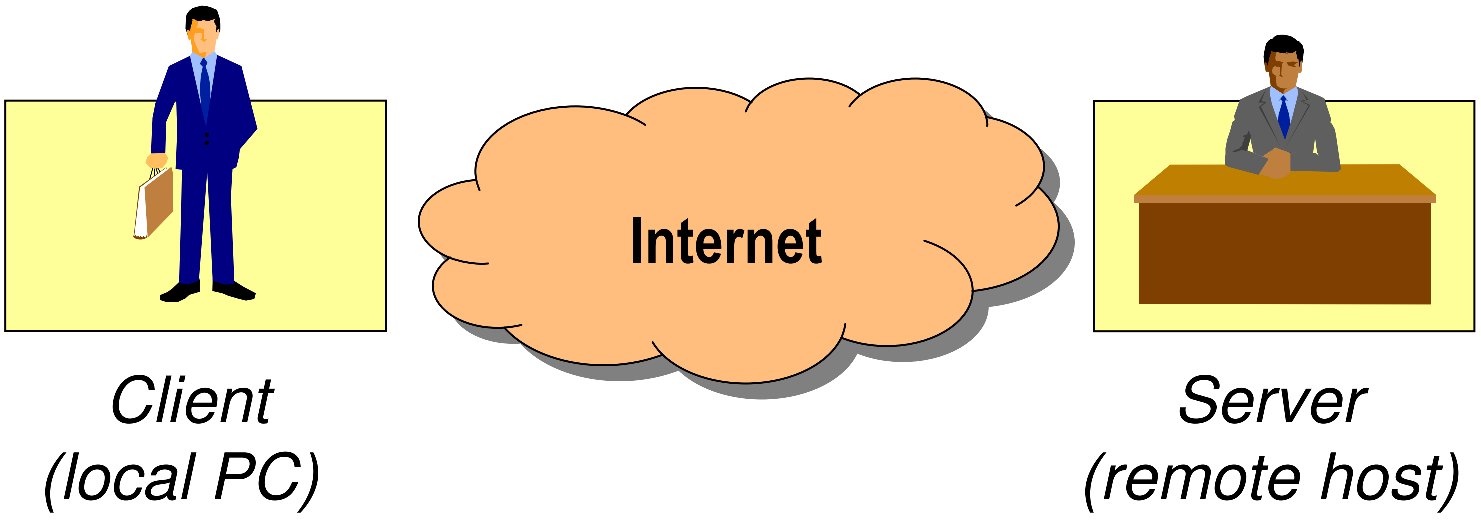
My Links

My Home Page
My mail box

- Search Engines
- Scientific and Technical Papers search engines
- Conferences, Journals and Technical societies
  - Conferences
  - Journals and Societies
  - On-line magazines
- Research Centers and Organizations
  - Italian Universities
  - Communication Networks
  - Wireless
  - IP

- Research Topics
  - Wireless ATM
  - UMTS
  - IP
  - Simulation
- Telecom Companies & info sources
- Education Centers
- Telecommunication Networks Documents
- Software resources
- Public institutions
- Art and Culture
- News Papers
- Travels
- Computer Stores
- Friends Home Pages
- Weather
- Italian Internet Access Points

Basic scenario



Client wants to retrieve a web page. What happens?

What is a “page” on the web?

a resource (i.e a file), specified by a
URL: Uniform Resource Locator.

e.g. my home page:

HTTP://cerbero.elet.polimi.it/people/bianchi/index.html

The three components of an URL

1. *Protocol (also called "scheme")*
   - *how can a page be accessed?* (**application protocol used**)
     - **http**://cerbero.elet.polimi.it/people/bianchi/index.html

2. *Host name*
   - *Where is the page located?* (**symbolic or numeric location**)
     - http://**cerbero.elet.polimi.it**/people/bianchi/index.html

3. *File (resource) name*
   - *What is the page called?* (**with full path**)
     - http://cerbero.elet.polimi.it**/people/bianchi/index.html**

Network applications: some jargon

Process: program running within a host.

- within same host, two processes communicate using interprocess communication (defined by OS).
- processes running in different hosts communicate with an application-layer protocol (defining message format, which message to exchange and in which order)

user agent: interfaces with user “above” and network “below”.

- implements user interface & application-level protocol
  - Web: browser
  - E-mail: mail reader
  - streaming audio/video: media player

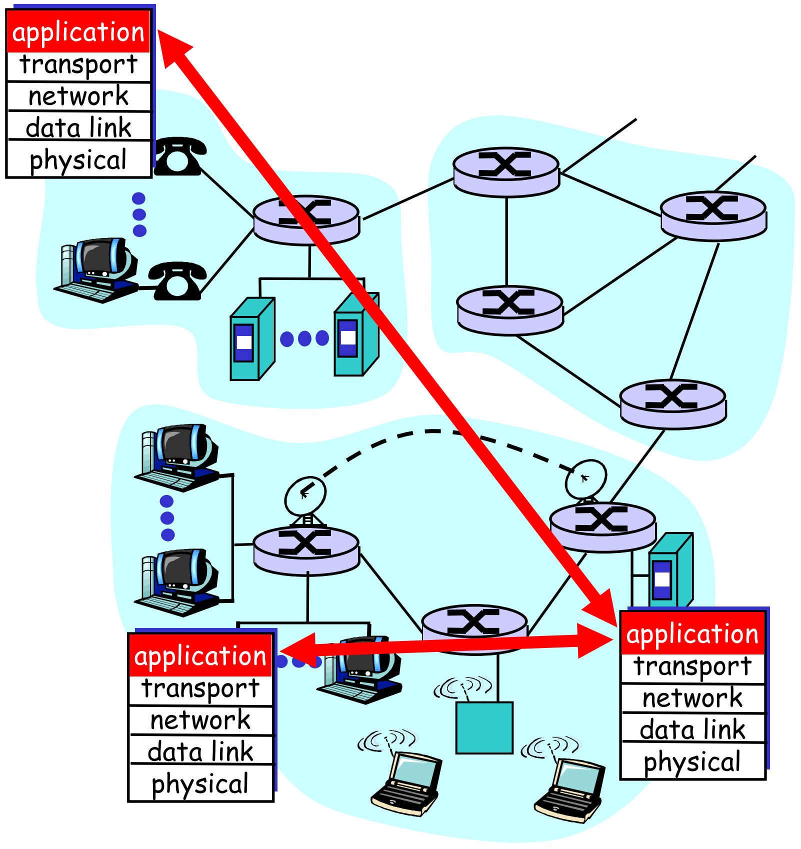
Applications and application-layer protocols

Application: communicating, distributed processes

- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- eg, KaZaA

Client-server paradigm

Typical network app has two pieces: *client* and *server*

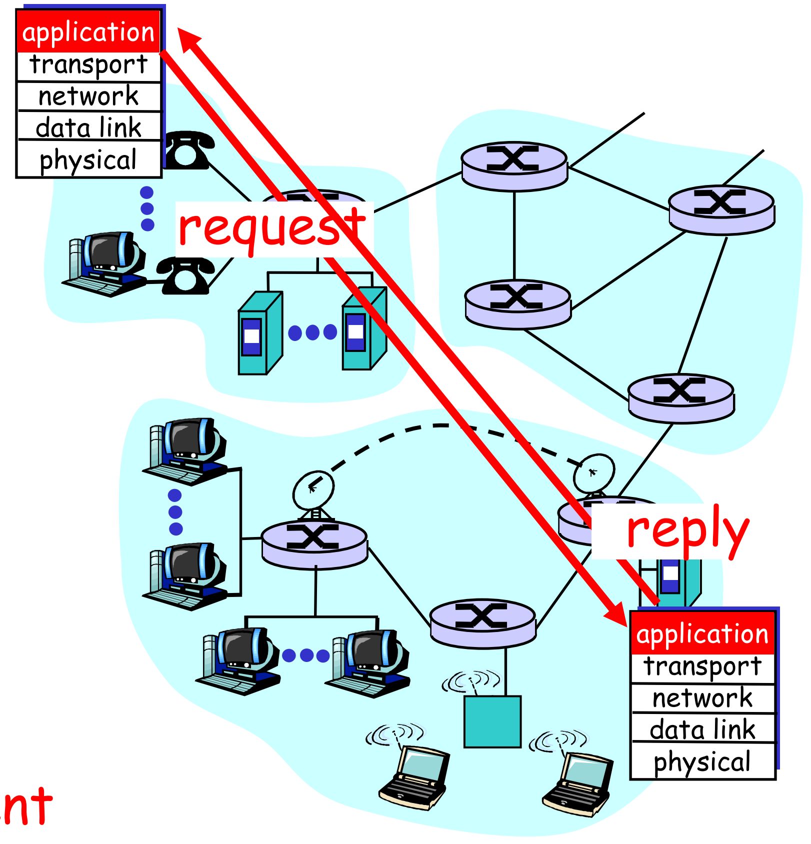
(NEXT SLIDE)

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- Web: client implemented in browser; e-mail: in mail reader

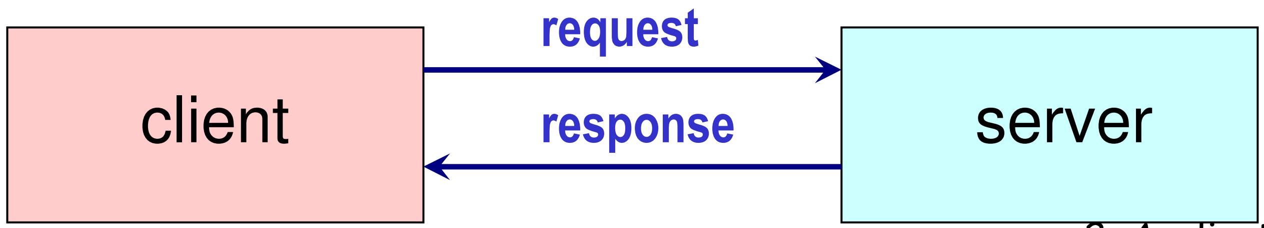
Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Architettura Client-Server

- Un processo client è solo in grado fare richieste di servizio (informazioni) e di interpretare le risposte
- Un processo server ha solo il compito di interpretare le richieste e fornire le risposte
- Se è necessario nello stesso host sia fare richieste che fornire risposte vengono usati due processi, client e server.
- Un protocollo applicativo per un'architettura client-server rispecchia questa divisione di ruoli e prevede messaggi di richiesta (request) generati dal lato client e messaggi di risposta (response) generati dal server



Programmi Client e Server

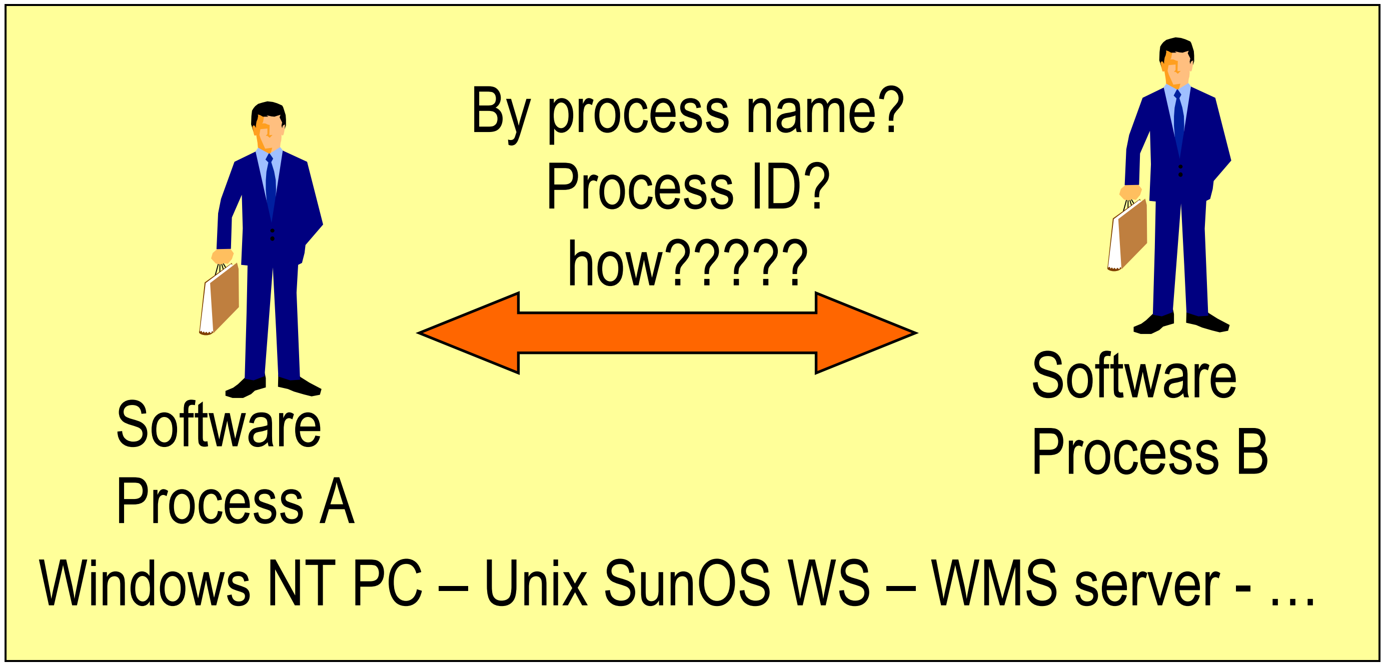
- Distinguiamo tra programma (software) e processo, istanza del programma in esecuzione su un host
- Un processo server è in esecuzione a tempo illimitato sul proprio host (daemon) e viene attivato mediante una *passive open*
- Un processo client viene attivato solo al momento di fare le richieste e viene attivato mediante una *active open* su richiesta dell'utente o di altro processo applicativo
- la passive open del server fa si che da quel momento il server accetti richieste dai client
- l'active open del client richiede l'indicazione dell'indirizzo e della porta del client, ed inizia la connessione TCP per mettere in comunicazione il client e il server

Addressing processes:

- For a process to receive messages, it must have an identifier
- Every host has a unique 32-bit IP address
- Q: does the IP address of the host on which the process runs suffice for identifying the process?
- Answer: No, many processes can be running on same host
- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

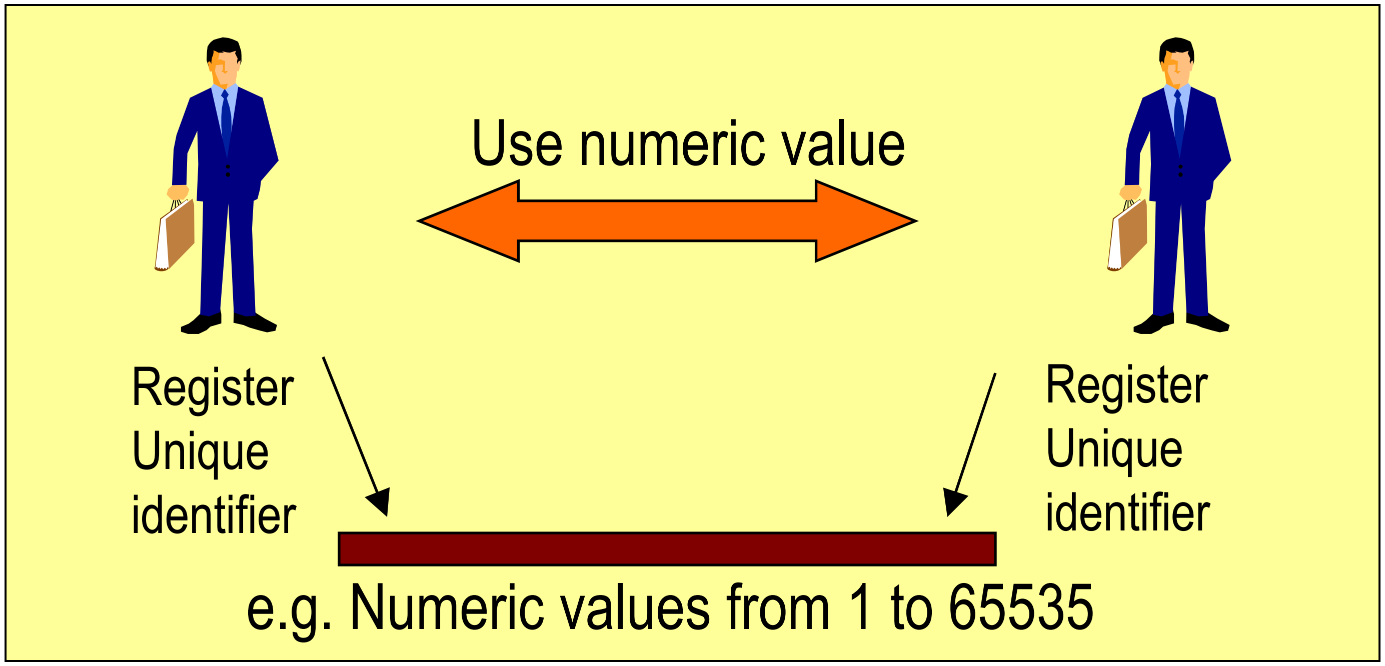
Addressing software processes

Inside the same machine: an Operating Systems issue



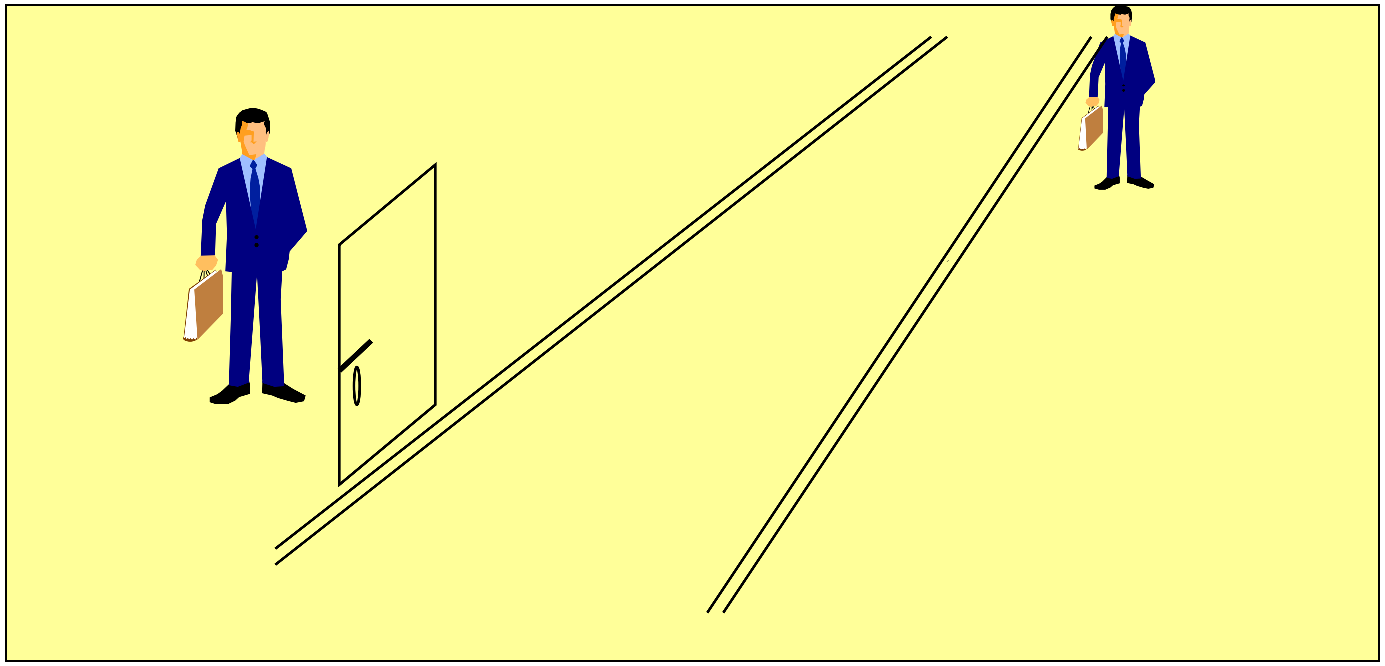
Different OSs = different process naming!

Addressing SW processes
very old solution adopted in most OS


Use numeric value
Register Unique identifier
Register Unique identifier
e.g. Numeric values from 1 to 65535

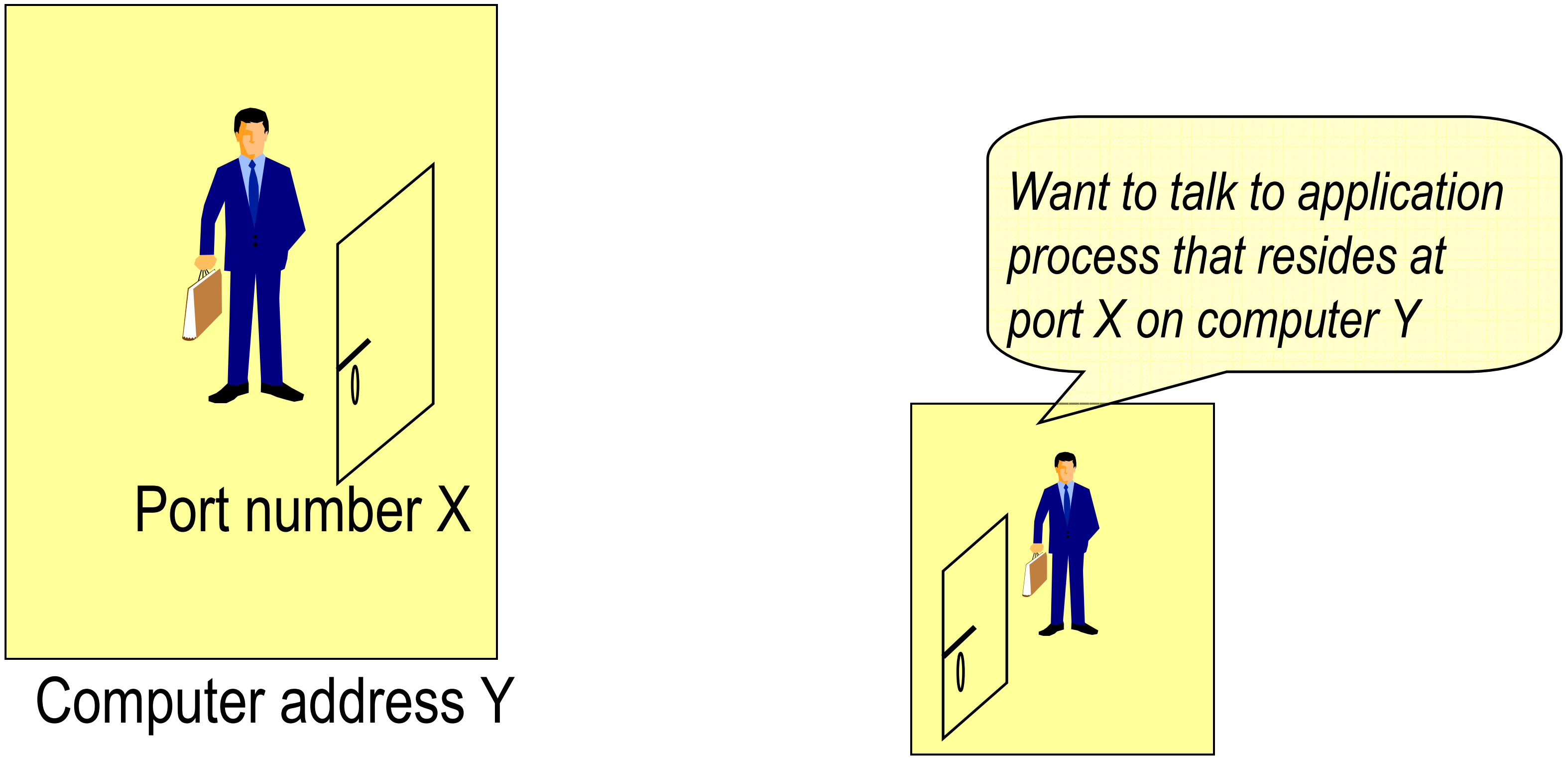
To talk each other, the SW processes need to know their numeric Values.

Port numbers



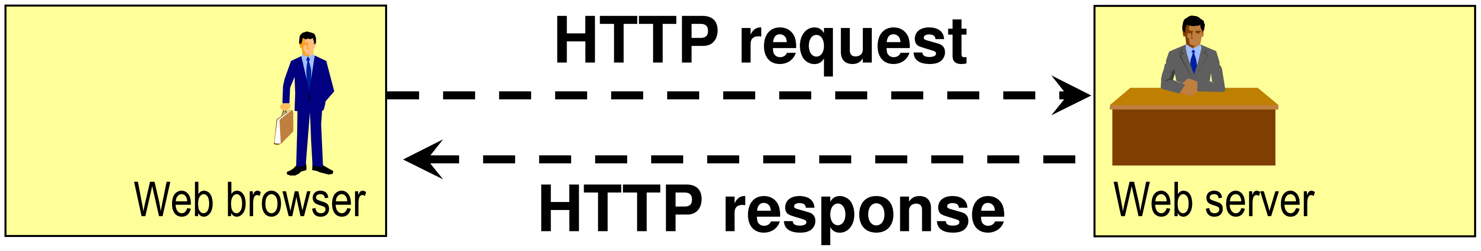
the “address” of the SW process inside the computer!

Same addressing scheme works for different machines!!


Port number X
Computer address Y
Want to talk to application process that resides at port X on computer Y

Re-understanding URLs

$$\underbrace{\text{HTTP:}}_{\text{protocol}}//\underbrace{\text{cerbero.elet.polimi.it}}_{\text{location}}\underbrace{\text{/people/bianchi/index.html}}_{\text{Filename}}$$



Web browser (SW process) needs to send HTTP Request to Web Server.
Location not enough (it is just the computer address!)

Addressing web servers: (wrong) idea

$$\underbrace{\text{HTTP:}}_{\text{protocol}}\text{//}\underbrace{\text{cerbero.elet.polimi.it}}_{\text{location}}\underbrace{\text{/....}}_{\text{Filename}}$$

Location = computer address!
Protocol = SW process address (HTTP = goto Web Server)
…. No need for port numbers?? ***…***

What if more than one Web Server installed???

URL structure (corrected!)

$$\underbrace{\textbf{HTTP:/}}_{\text{protocol}}\underbrace{\textbf{/cerbero.elet.polimi.it : }\textbf{\textit{portnumber}}}_{\text{location}}\underbrace{\textbf{/....}}_{\text{Filename}}$$

Default value for HTTP: 80

HTTP://cerbero.elet.polimi.it/…

equal to

HTTP://cerbero.elet.polimi.it:80/…

different from

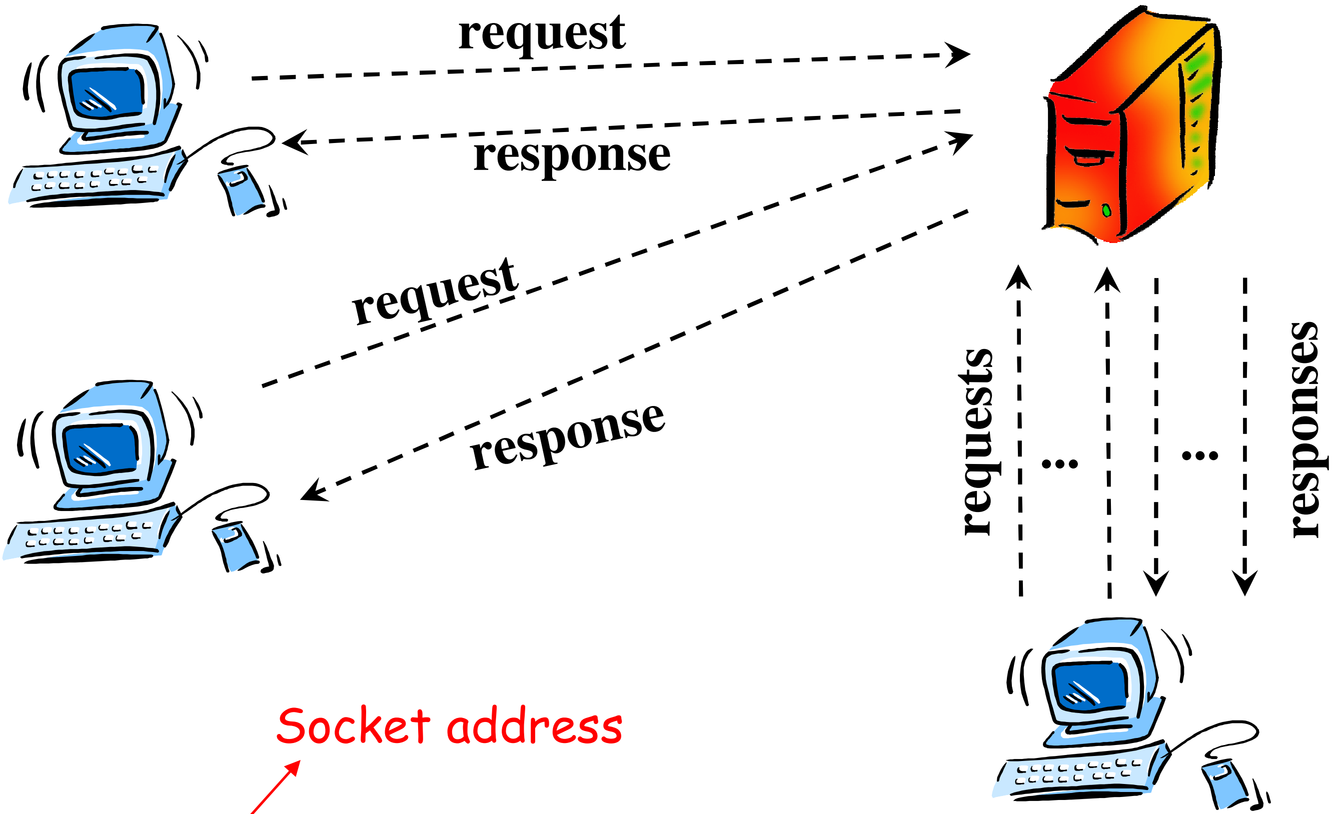
HTTP://cerbero.elet.polimi.it:8080/… (if exists!)

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: RFC 1700)
- number assignment (by IANA)
  - 0 not used
  - 1-255 reserved for well known processes
  - 256-1023 reserved for other processes
  - 1024-65535 dedicated to user apps

Programmi Client e Server

- Normalmente più client possono inviare richieste ad uno stesso server
- Un client può fare più richieste contemporanee



Socket address

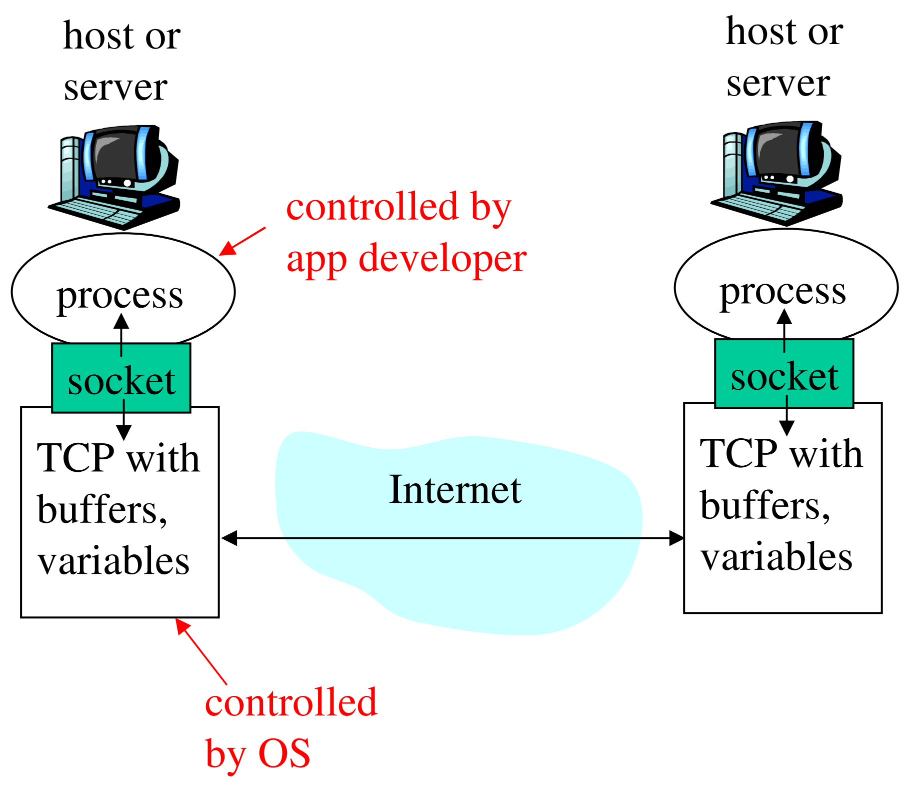
- Un flusso di informazioni tra due processi identificato da una quadrupla

(indirizzo IP sorgente, porta sorgente, indirizzo IP destinazione, porta destinazione)

- Di solito il client NON USA un well known port #
  - OS assegna un num. di porta disponibile

Processes communicating across network

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process assumes transport infrastructure on other side of door which brings message to socket at receiving process


host or server
host or server
controlled by app developer
process
process
socket
socket
TCP with buffers, variables
TCP with buffers, variables
Internet
controlled by OS

- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Socket API

- descriptor = socket(domain,type,protocol)

crea un socket e ne restituisce un descrittore

Protofamily: Type: tipo di comunicazione usata dal socket (SOCK_STREAM connection-oriented, SOCK_DGRAM connectionless)

- close(socket)
- bind(socket,localaddr,addrlen)

al server: associa ad un socket creato IP address e porta

- listen(socket,queuesize)

usata dal server per chiedere una passive open

- connect(socket,saddress,saddresslen)

Stabilisce una connessione con un server con address e protocol port number specificati in saddress attraverso il socket del client 'socket'

- newsock=accept(socket,caddress,caddresslen)

per accettazione lato server e creazione del socket per quella connessione

- send(socket,data,length,flags)
- recv(socket,buffer,length,flags)

Sequence of Socket Procedure Calls

SERVER

$$\begin{tikzcd}
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{bind} \arrow[d] \\
\text{listen} \arrow[d] \\
\text{accept} \arrow[d] \\
\text{send} \arrow[d] \\
\text{close} \arrow[uu, bend left=60]
\end{tikzcd}$$

CLIENT

$$\begin{tikzcd}
\text{gethostbyname} \arrow[d] \\
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{connect} \arrow[d] \\
\text{recv} \arrow[d] \\
\text{close}
\end{tikzcd}$$

Bind

- La funzione bind assegna un indirizzo locale ad un socket. È' usata cioè per specificare la prima parte dalla socket pair. Viene usata sul lato server per specificare la porta (e gli eventuali indirizzi locali) su cui poi ci si porrà in ascolto. Il prototipo della funzione è il seguente:
    - int bind(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen) Assegna un indirizzo ad un socket.
    - Con i socket TCP la chiamata bind permette di specificare l'indirizzo, la porta, entrambi o nessuno dei due

- Con bind si può assegnare un indirizzo IP specifico ad un socket, purché questo appartenga ad una interfaccia della macchina. Per un client TCP questo diventerà l'indirizzo sorgente usato per i tutti i pacchetti inviati sul socket, mentre per un server TCP questo restringerà l'accesso al socket solo alle connessioni che arrivano verso tale indirizzo.
- Normalmente un client non specifica mai l'indirizzo di un socket, ed il kernel sceglie l'indirizzo di origine quando viene effettuata la connessione, sulla base dell'interfaccia usata per trasmettere i pacchetti, (che dipenderà dalle regole di instradamento usate per raggiungere il server). Se un server non specifica il suo indirizzo locale il kernel userà come indirizzo di origine l'indirizzo di destinazione specificato dal SYN del client.

Sequence of Socket Procedure Calls

$$\begin{tikzcd}
\text{SERVER} & & \text{CLIENT} \\
\text{getprotobyname} \arrow[d] & & \text{gethostbyname} \arrow[d] \\
\text{socket} \arrow[d] & & \text{getprotobyname} \arrow[d] \\
\text{bind} \arrow[d] & & \text{socket} \arrow[d] \\
\text{listen} \arrow[d] & & \text{connect} \arrow[d] \\
\text{accept} \arrow[d] & & \text{recv} \arrow[d] \\
\text{send} \arrow[d] & & \text{close} \\
\text{close} \arrow[uu, bend left] & &
\end{tikzcd}$$

Listen

- int listen(int *s*, int *backlog*);
- The listen() function marks a connection-mode socket (for example, those of type SOCK_STREAM), specified by the socket argument *s*, as accepting connections, and limits the number of outstanding connections in the socket's listen queue to the value specified by the *backlog* argument. The socket *s* is put into 'passive' mode where incoming connection requests are acknowledged and queued pending acceptance by the process.
- The *backlog* parameter of this function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with the queue full, the client receives an error with an indication of ECONNREFUSED.

Sequence of Socket Procedure Calls

SERVER

$$\begin{tikzcd}
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{bind} \arrow[d] \\
\text{listen} \arrow[d] \\
\text{accept} \arrow[d] \\
\text{send} \arrow[d] \\
\text{close} \arrow[uu, bend left=60]
\end{tikzcd}$$

CLIENT

$$\begin{tikzcd}
\text{gethostbyname} \arrow[d] \\
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{connect} \arrow[d] \\
\text{recv} \arrow[d] \\
\text{close}
\end{tikzcd}$$

Accept

- int accept(int *s*, struct sockaddr **addr*, socklen_t **addrlen*);
- The accept() function accepts a connection on a socket. An incoming connection is acknowledged and associated with an immediately created socket. The original socket is returned to the listening state.
- The parameter *s* is a socket that has been created with socket() and bound to an address with bind() and that is listening for connections after a call to listen(). accept() extracts the first connection on the queue of pending connections, creates a new socket with the same properties of *s*, and returns a new handle to the socket.
- The accepted socket is used to read and write data to and from the socket that connected to it. However, it is not used to accept more connections. The original socket remains open to accept further connections.

Sequence of Socket Procedure Calls

SERVER

$$\begin{tikzcd}
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{bind} \arrow[d] \\
\text{listen} \arrow[d] \\
\text{accept} \arrow[d] \\
\text{send} \arrow[d] \\
\text{close} \arrow[uu, bend left]
\end{tikzcd}$$

CLIENT

$$\begin{tikzcd}
\text{gethostbyname} \arrow[d] \\
\text{getprotobyname} \arrow[d] \\
\text{socket} \arrow[d] \\
\text{connect} \arrow[d] \\
\text{recv} \arrow[d] \\
\text{close}
\end{tikzcd}$$

Connect

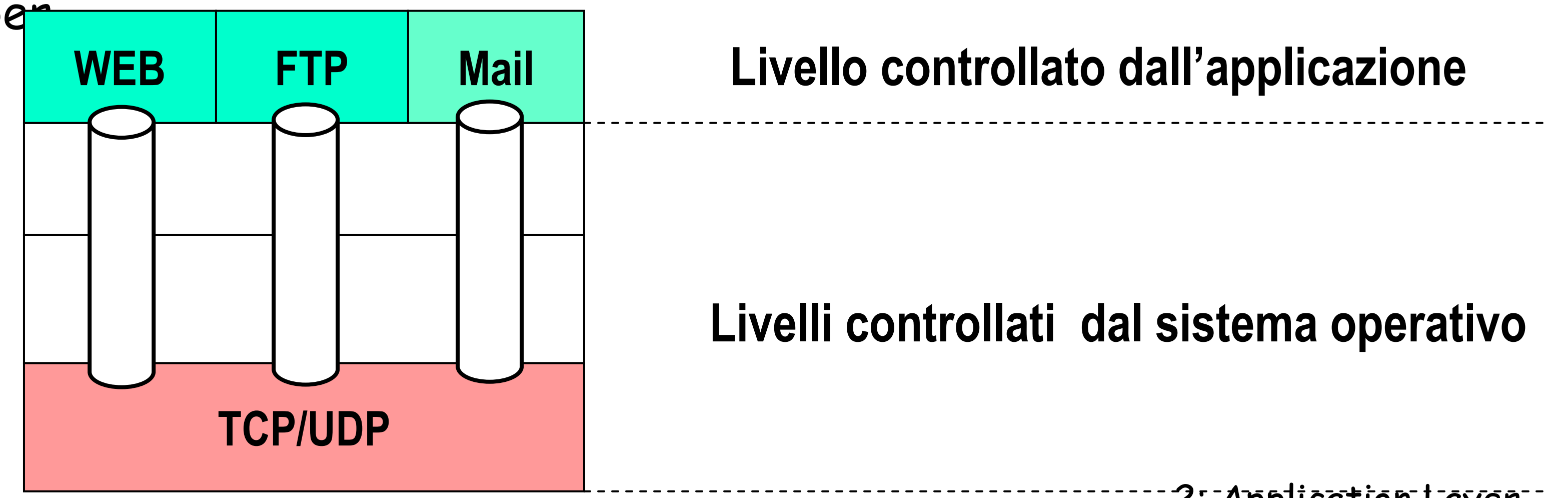
- int connect(int *s*, const struct sockaddr **name*, socklen_t *namelen*);
- The connect() system call initiates a connection on a socket.
  - If the parameter *s* (a socket) is of type SOCK_DGRAM, then connect() permanently specifies the peer to which datagrams are to be sent.
  - For stream sockets (type SOCK_STREAM), an active connection is initiated to the foreign host using *name* (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send/receive data.

Sequence of Socket Procedure Calls

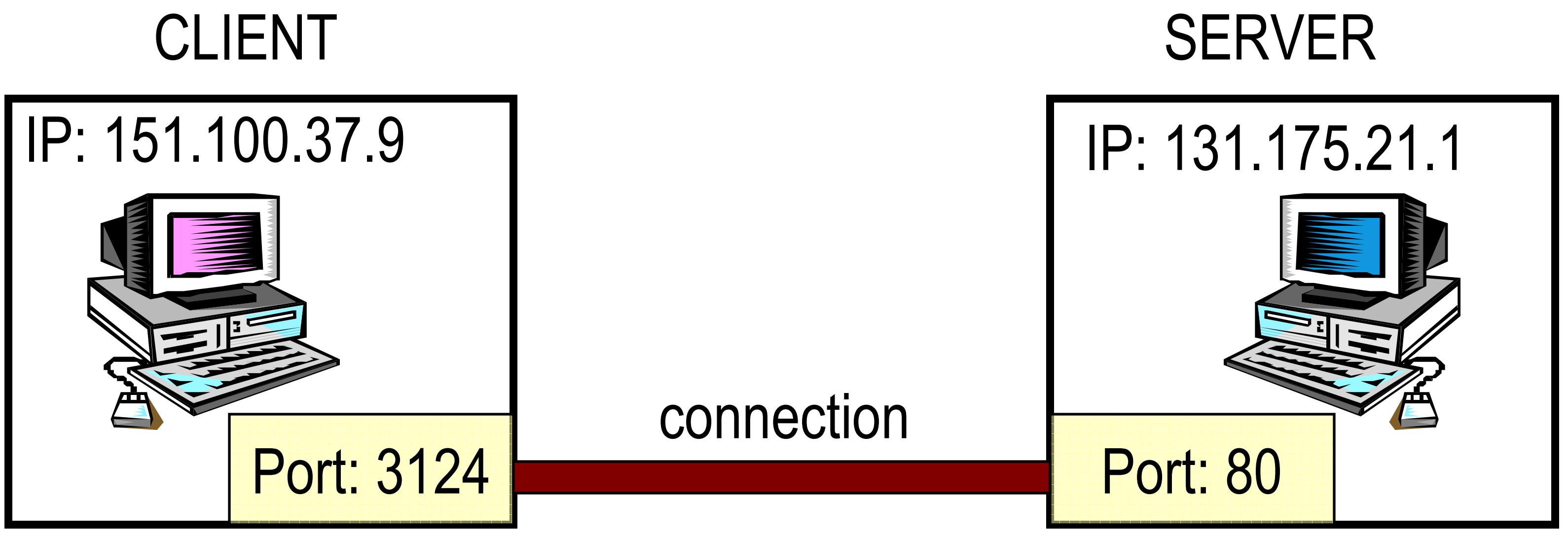
$$\begin{tikzcd}
\text{SERVER} & & \text{CLIENT} \\
\text{getprotobyname} \arrow[d] & & \text{gethostbyname} \arrow[d] \\
\text{socket} \arrow[d] & & \text{getprotobyname} \arrow[d] \\
\text{bind} \arrow[d] & & \text{socket} \arrow[d] \\
\text{listen} \arrow[d] & & \text{connect} \arrow[d] \\
\text{accept} \arrow[d] & & \text{recv} \arrow[d] \\
\text{send} \arrow[d] & & \text{close} \\
\text{close} \arrow[uu, bend left] & &
\end{tikzcd}$$

Interazione coi livelli inferiori: architettura TCP/IP

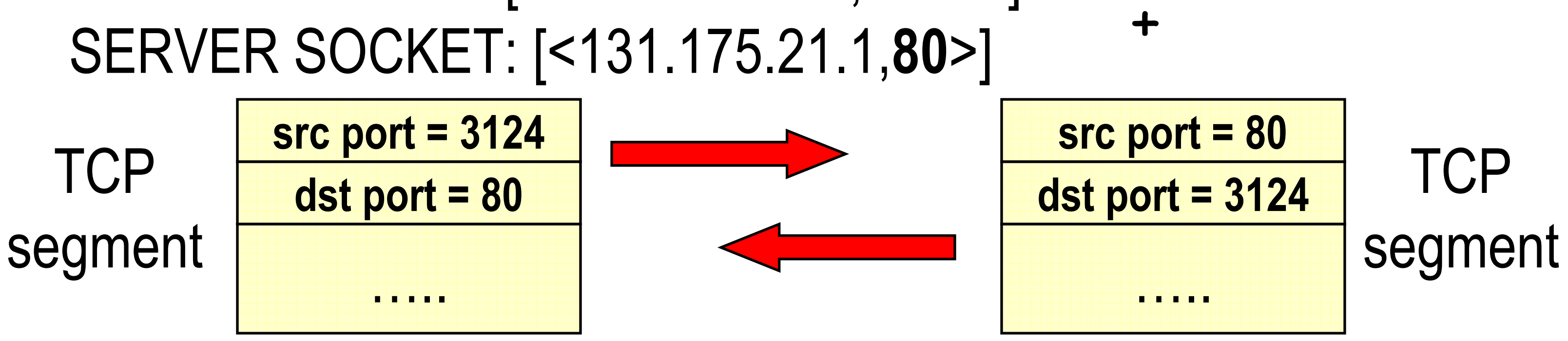
- I protocolli applicativi che si appoggiano sull' Internet Protocol si appoggiano direttamente sul protocollo di trasporto TCP/UDP
- Lo scambio di messaggi fra i processi applicativi avviene utilizzando i servizi dei livelli inferiori attraverso i SAP (Service Access Point)
- I protocolli di trasporto estendono il servizio di trasporto in rete dell'info tra due end systems offerto da IP consentendo il trasporto dell'info tra due PROCESSI APPLICATIVI in esecuzione sui due end systems
- Ogni processo è associato ad un SAP. I SAP tra livello applicativo e di trasporto sono le *porte* e l'insieme porta TCP/UDP e indirizzo IP viene chiamato *socket*
- Come si distingue a quale protocollo di trasporto passare il segmento? Protocol number


WEB
FTP
Mail
TCP/UDP
Livello controllato dall'applicazione
Livelli controllati dal sistema operativo

Connections
identified by socket addresses at its ends

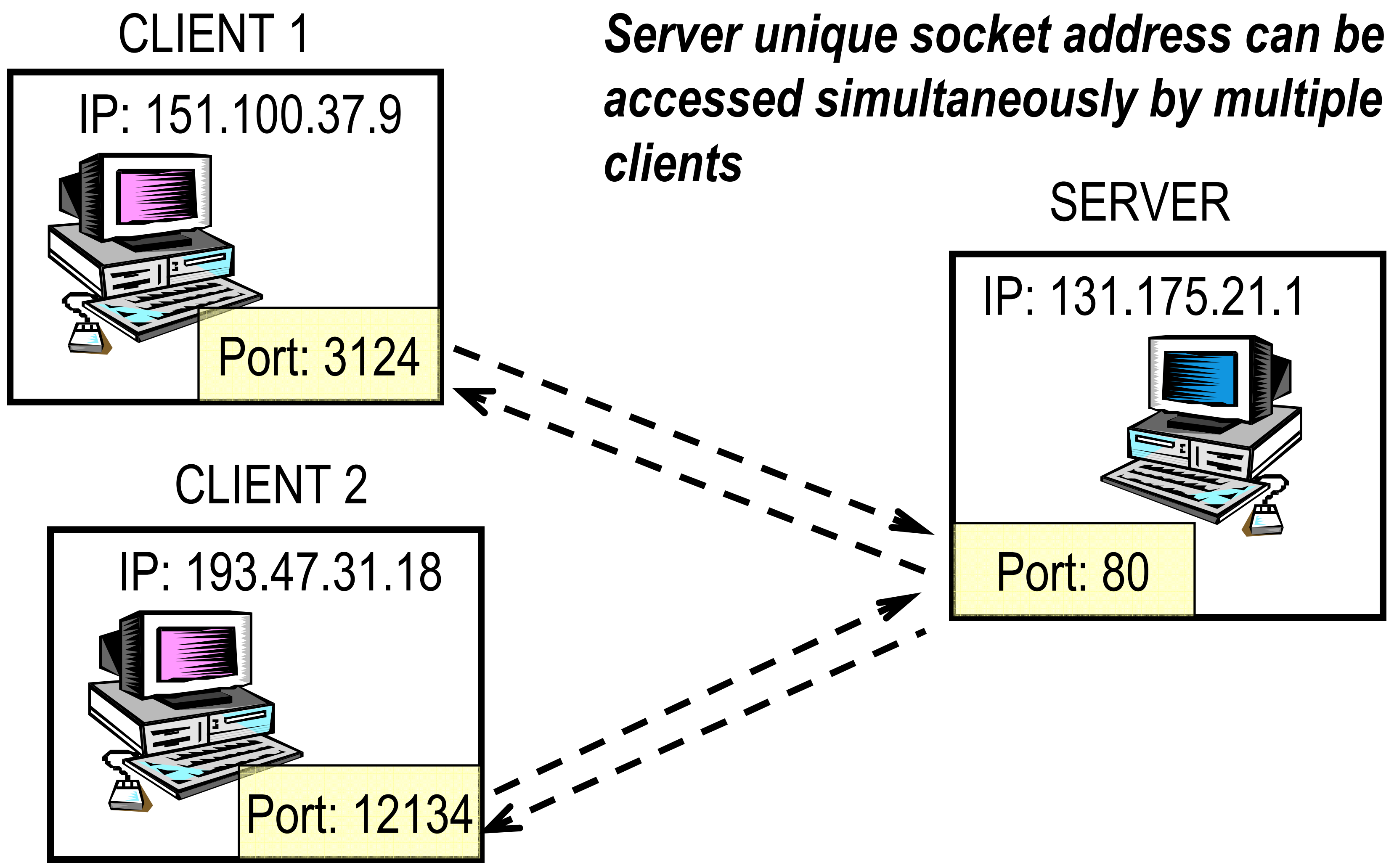


CLIENT SOCKET: [<151.100.37.9,**3124**>]
SERVER SOCKET: [<131.175.21.1,**80**>]



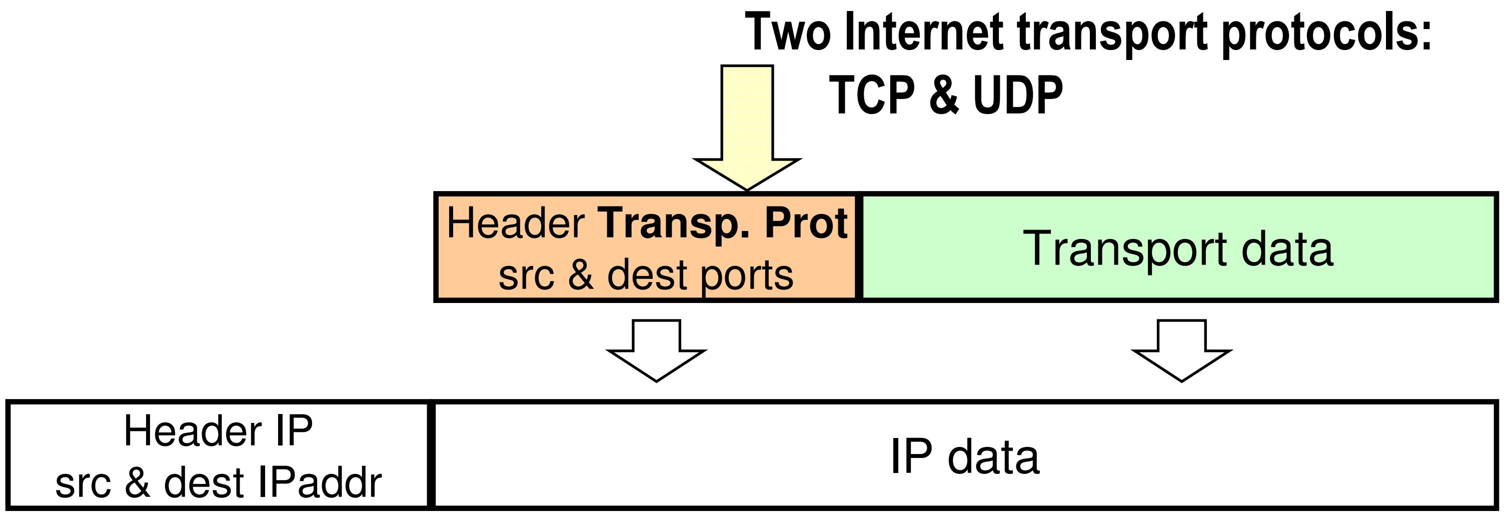
Managing multiple connections

Server unique socket address can be accessed simultaneously by multiple clients



Socket addresses: where?

- Ports: in the Internet Transport protocol header (TCP or UDP)
- IP addresses: In the IP packet header



IP packets travel in the network based on IP address;

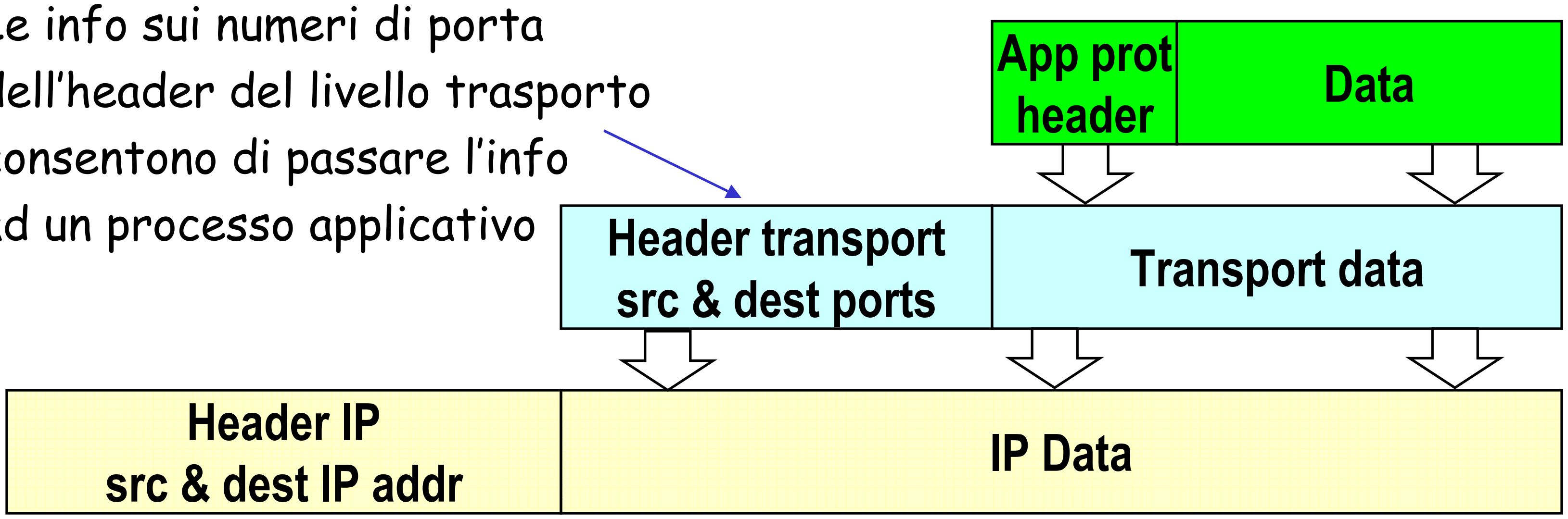
Once they reach destination, they are delivered to app based on port #

Sockets and Internet transport protocols

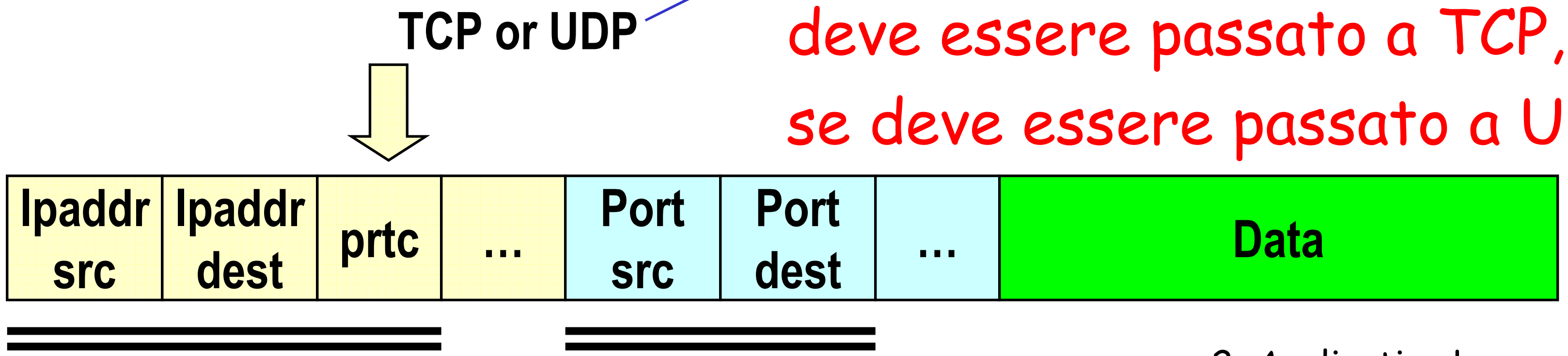
- Two transport protocols in Internet:
  - TCP (reliable, connection-oriented)
  - UDP (unreliable, connectionless)
- When opening socket, needs to specify which transport to use!

Socket addresses (refined)

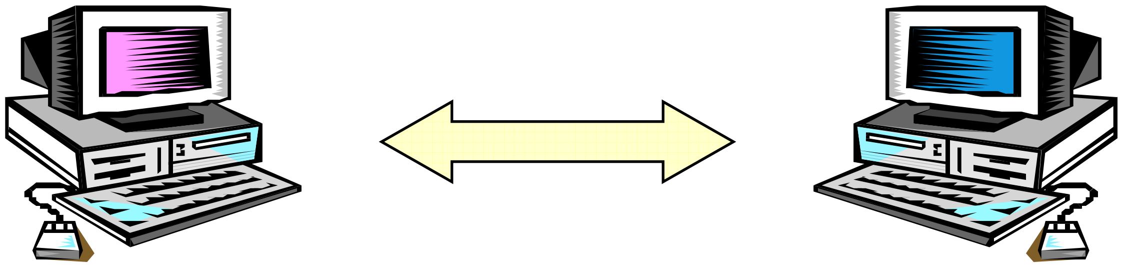
Le info sui numeri di porta dell'header del livello trasporto consentono di passare l'info ad un processo applicativo


App prot header
Data
Header transport src & dest ports
Transport data
Header IP src & dest IP addr
IP Data

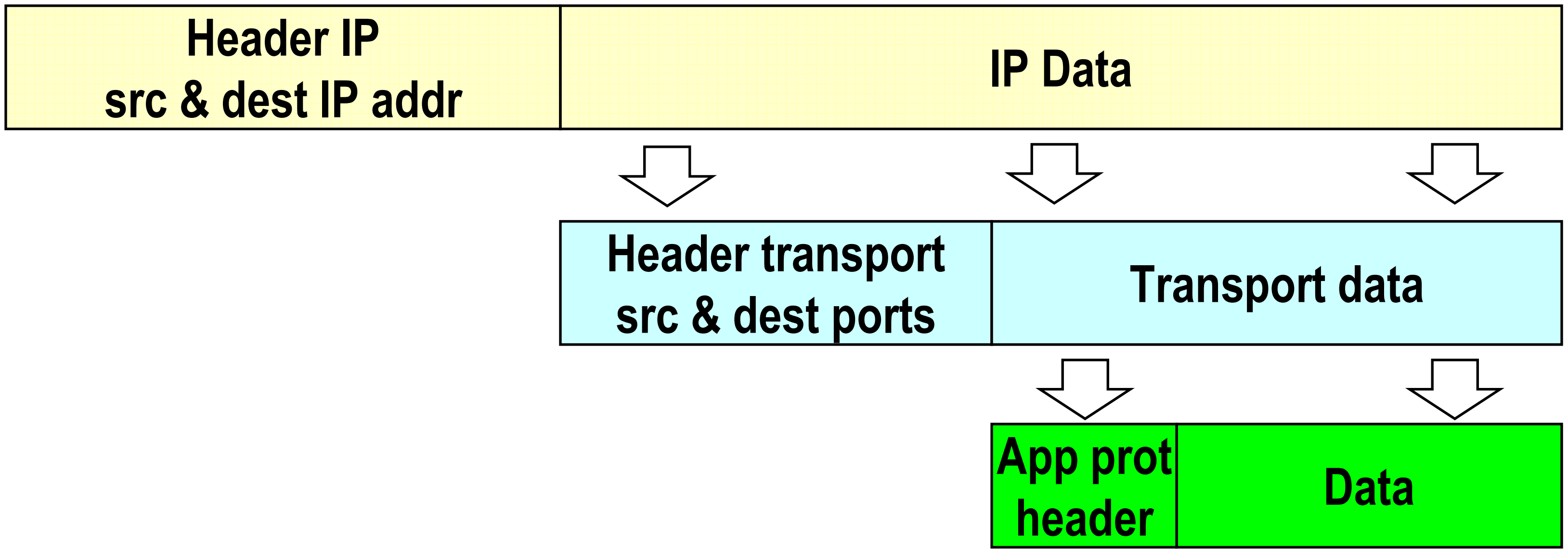
Nell'header IP il campo Protocol vale 6 se il pacchetto deve essere passato a TCP, 17 se deve essere passato a UDP



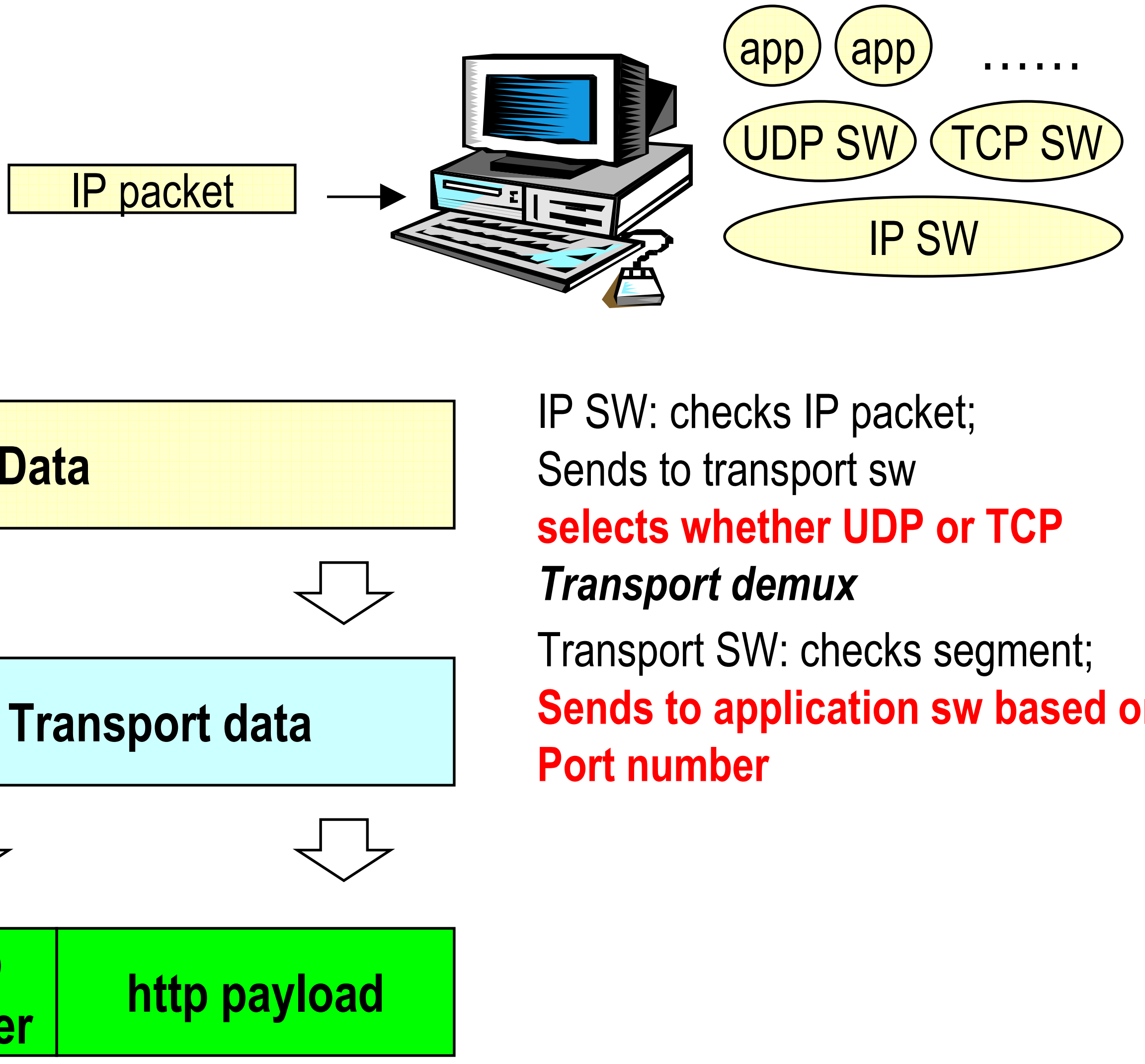
The Internet level view



Information units travelling in the network: IP packets



Demultiplexing at receiver



IP SW: checks IP packet;
Sends to transport sw
selects whether UDP or TCP
Transport demux
Transport SW: checks segment;
Sends to application sw based on Port number

Programmi Client e Server

- Un client può essere eseguito in modalità parallela o seriale
  - esempio: invia più richieste in parallelo per i file che compongono una pagina web
- Anche un server può essere eseguito in modalità parallela o seriale
- Normalmente i server che usano TCP vengono eseguiti in modalità parallela e sono dunque in grado di rispondere a più richieste contemporaneamente
- Con ognuno dei client viene aperta una connessione TCP che viene mantenuta per il tempo necessario a scambiare richieste e risposte
- La gestione delle procedure per ciascun client collegato avviene mediante la generazione di processi figli (per clonazione del processo o uso di processi multi-thread)

What transport service does an app need?

Data loss

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

Timing

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

Bandwidth

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

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP? Which service does it add to IP?

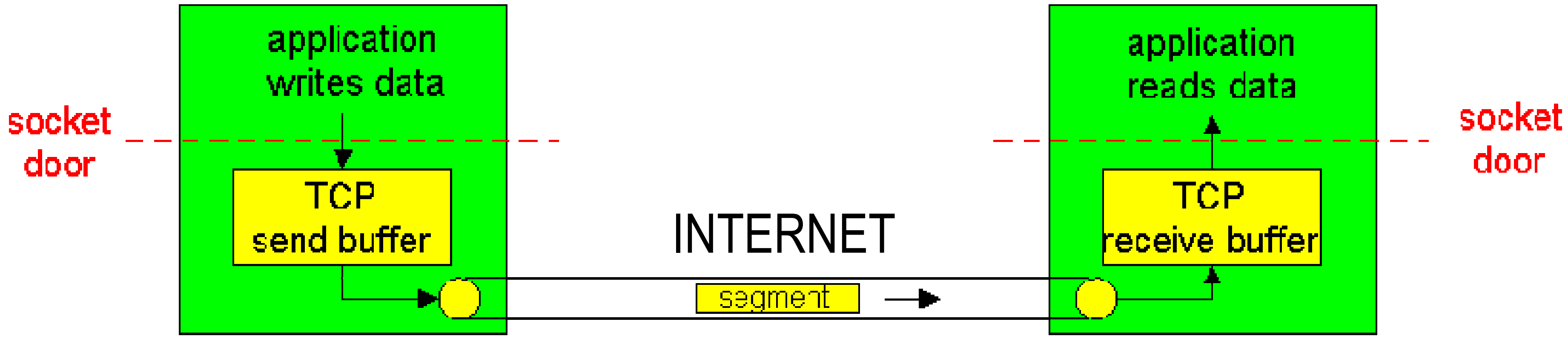
Multiplexing/Demultiplexing

Error Control

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

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

Chapter 2 outline

Breve storia del Web

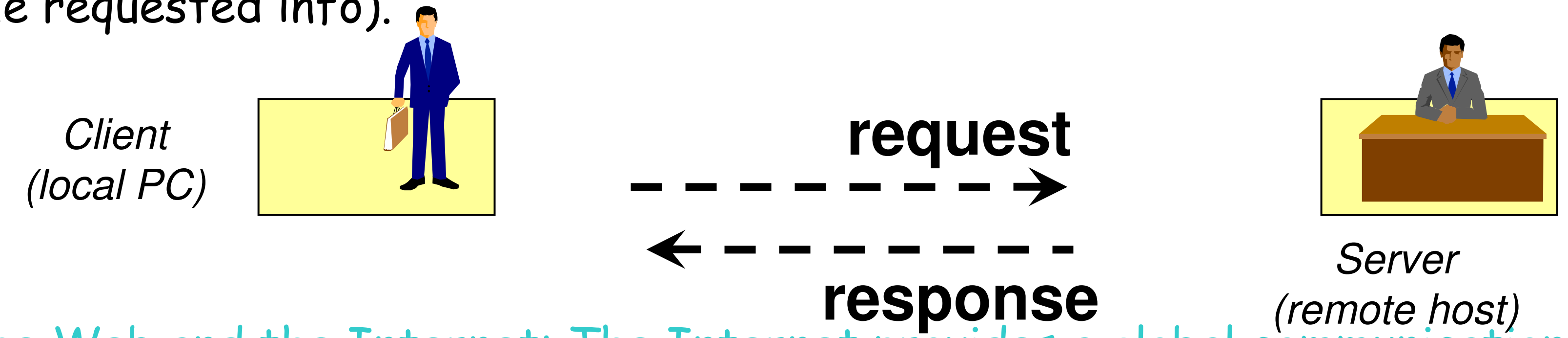
- 1945: Vannevar Bush in 'As we may think' proposes Memex to extend human memory via mechanical means. V. Bush wanted to make the existing store of knowledge (fastly growing) more accessible to mankind. 'A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it maybe consulted with exceeding speed and flexibility. It is an enlarged supplement to his memory. …Wholly new forms of encyclopedias will appear ready made with a mesh of **associative trails** running through them'
- 1965 Ted Nelson coins the term 'hypertext' to describe nonsequential writing that presents information as a collection of linked nodes. 'Readers can pursue the information in a variety of ways by navigating from one node to another'
- 1970s scientific community communicate through ARPANET (exchanging e-mails, file transfers..)
- 1989 Tim Berners-Lee creates the World Wide Web (networked application that links users and services distributed across computers around the world. Users can view pages, search for information, send and receive e-mails, initiate business transactions etc. Tim Berners Lee proposed linking info presented on various machines at CERN (European Laboratory for Particle Physics, Geneva). Other systems had been created to download files (FTP)from remote computers, search for information (Gopher), WAIS (Wide Area Information Servers) etc. Why did the Web succed? 1)SIMPLE INTERFACE 2) ENHANCED FEATURES 3) USE OF HYPERTEXT

Breve storia del Web

- 1991 First browser and server introduced
- 1993 The Web consisted of around 50 servers
- 1993 First release of Mosaic browser for windows (written by Marc Andreesen and Eric Bina). The web accounted for 1% of the traffic of the Internet
- Late 1990s → Web responsible for over 75% of the Internet traffic!! Hundreds millions of users; millions of web sites. Reasons for success: graphical user interface, ease of publishing new content
- Web used for e-business, business to business, interactions between users (chatrooms and games). Web increasingly used also to access private or proprietary data.
    - Web traffic: small number of Web pages accounts for the majority of web accesses
    - from text to images, animations, video files (increased storage needed at the Web servers, increased time for downloading, increased load on the network)
    - Dynamically generated content (e.g. search engines generates pointers based on keywords)
    - Web used for banking, e-business;important personal info (e.g. credit card info transferred)→ security, authentication needed

Basic 'bricks' of the Web

- Uniform Resource Identifier (URI)
  - Tipicamnete si usa un Uniform Resource Location (URL) –allows to identify a web resource
- Hypertext Markup Language (HTML) –provides a standard representation for hypertext documents in ASCII format. Allows authors to format text, reference images, embed hypertext links. Software tools have been developed for generating HTML files (e.g. Netscape Composer, FrontPage) so that editing a web page is straightforward and does not require HTML knowledge.
- Hypertext Transfer Protocol (HTTP)—it is the protocol web components use to communicate. Defines the format and meaning of messages exchanged between Web components such as clients and servers. Request-response protocol: the client issues a request. If the request can be satisfied the server replies (with the requested info).



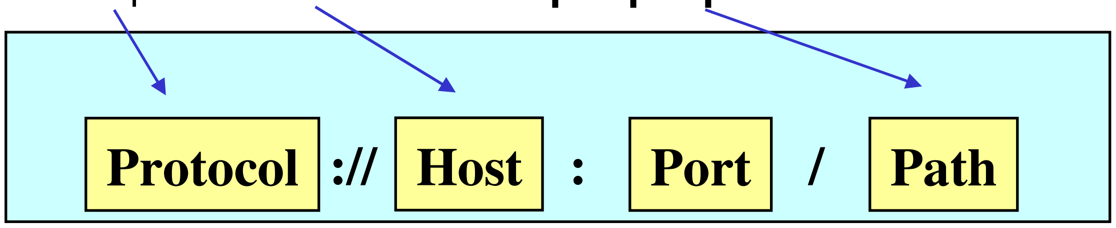
- The Web and the Internet: The Internet provides a global communication infrastructure allowing Web clients to access a wide variety of Web Servers throughout the world

The three components of an URL

1. *Protocol (also called "scheme")*
   - *how can a page be accessed?* **(application protocol used)**
     - **http**://www.dsi.uniroma1.it/people/petrioli/index.html

2. *Host name*
   - *Where is the page located?* **(symbolic or numeric location)**
     - http://www.**dsi.uniroma1.it**/people/petrioli/index.html

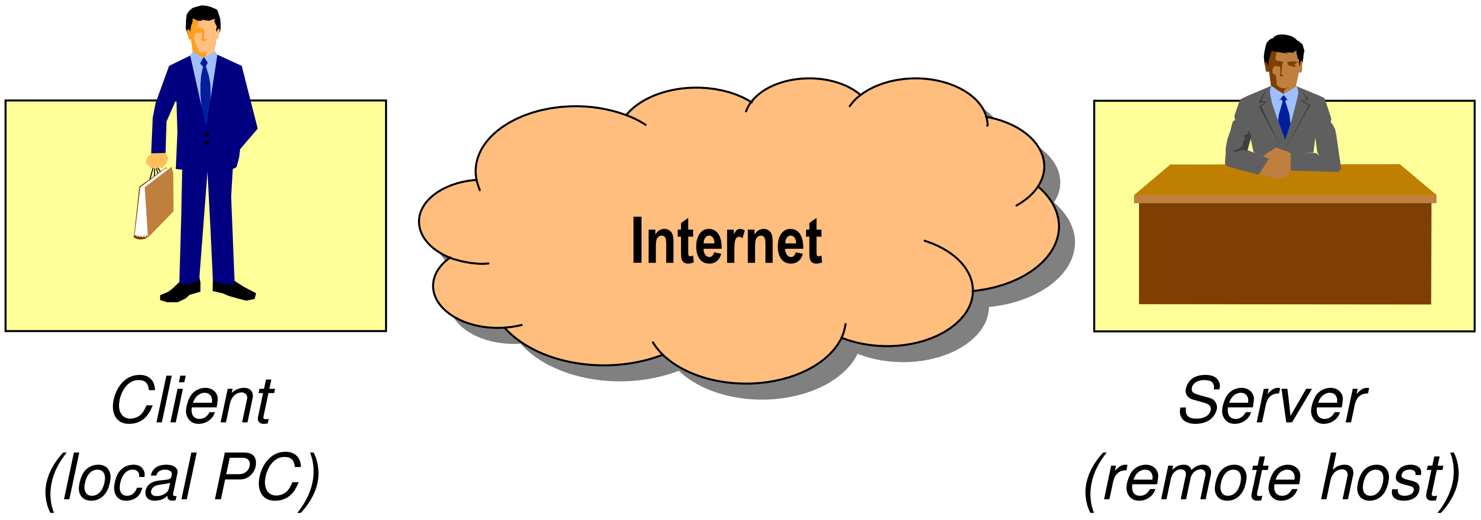
3. *File (resource) name*
   - *What is the page called?* **(with full path)**
     - http://www.dsi.uniroma1.it/**people/petrioli/index.html**



Web Content

- The Web is a collection of resources or objects distributed throughout the Internet. **Each object is a network- accessible document or service**, which maybe available in different formats (e.g. HTML, poscript, PDF). A resource maybe a static file on a machine or maybe dynamically generated upon request.
- **A web page consists of a container resource such as HTML file which may include links to one or more embedded resources such as images and animations.**
- HTTP 1.0
  - Each HTTP transfer involves a single object
  - Downloading a web page involves separate HTTP transfer (1 for the container resource, one for each of the embedded objects)
- Request-response protocol. A user agent (e.g. a browser) initiates an HTTP request and handles the response. The program that actually sends the HTTP request and receive the response is called **client**. The **origin server** is the program that (possibly generates and) provides a Web resource. It receives an HTTP request and sends a response.
- The client may send its HTTP request to an intermediary on the client-server path called proxy (e.g. caching frequently accessed web pages). A proxy acts both as client and server.

Basic scenario



Client wants to retrieve a web page. What happens?

What is a “page” on the web?

a resource (i.e a file), specified by a
URL: Uniform Resource Locator.

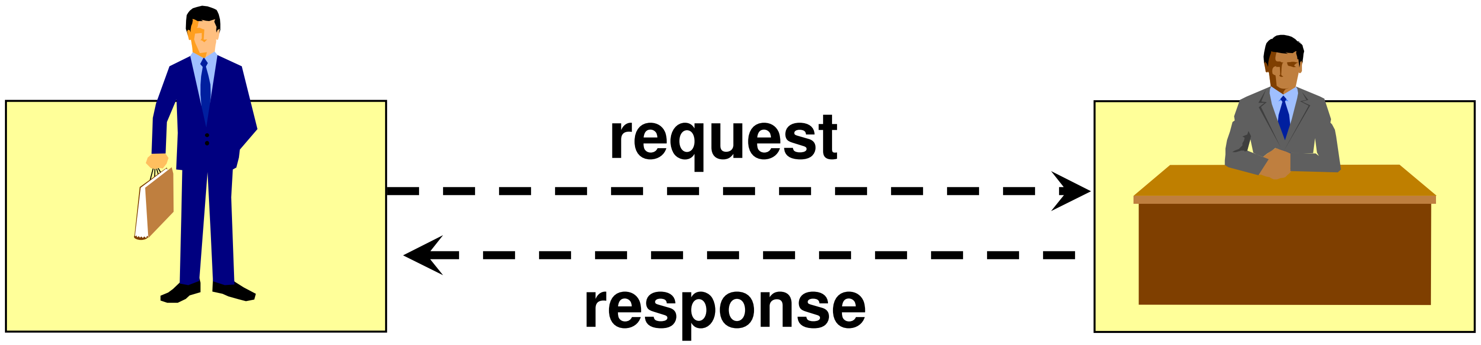
HTTP://cerbero.elet.polimi.it/people/bianchi/index.html

1. Protocol

- HTTP: the protocol of the WWW
  - version 1.0, RFC 1945, may 1996
  - version 1.1, RFC 2068 (jan97), RFC 2616 (jun99)

(but also **FTP**: file transfer protocol, **TELNET**: opens a telnet window, **FILE**: access local file etc.)

- HTTP: Request-response protocol. A request message is sent from the client to a recepient server. The recipient server send a response message back.

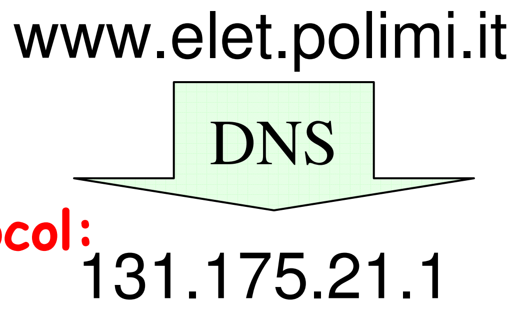


- Different requests for the same URL can result in different responses due to the time of request, changes in the resource, the request header fields.

2. Location - host name

Specifies where is the page located:

- **on which host**
  - **Humans understand names;**
  - **Machines prefer numbers!**
  - **Domain Name System (DNS) protocol:**
    - **translates names in numbers**



3. File names
(several shortcuts handled server side)

page name non mandatory

http://cerbero.elet.polimi.it/people/bianchi/
http://www.yahoo.com
http://www.sun.com/products

Unix style user id shortcuts

http://www.cs.columbia.edu/~hgs

Il server e' in grado di estrapolare il path completo verso la risorsa

Various page extensions (with server side meaning: no client side interpretation!)

Pagine dinamiche

*.htm *.html *.asp *.jsp *.php ……..

Case sensitivity

- Protocol: non case-sensitive
- Location: non case-sensitive
- File name: case-sensitive

http://cerbero.elet.polimi.it/people/bianchi/index.html

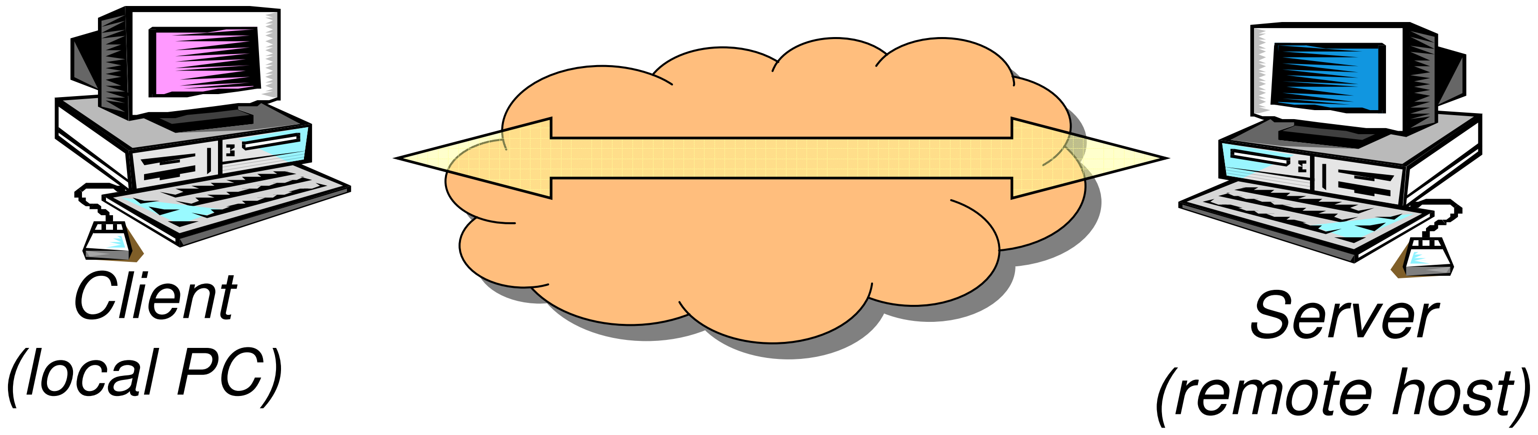
equal to

Http://Cerbero.Elet.PoliMi.it/people/bianchi/index.html

different from

http://cerbero.elet.polimi.it/people/bianchi/Index.html

Refined scenario



Client wants to retrieve a remote resource:

/people/bianchi/index.html

On the server

cerbero.elet.polimi.it

By using the application layer protocol

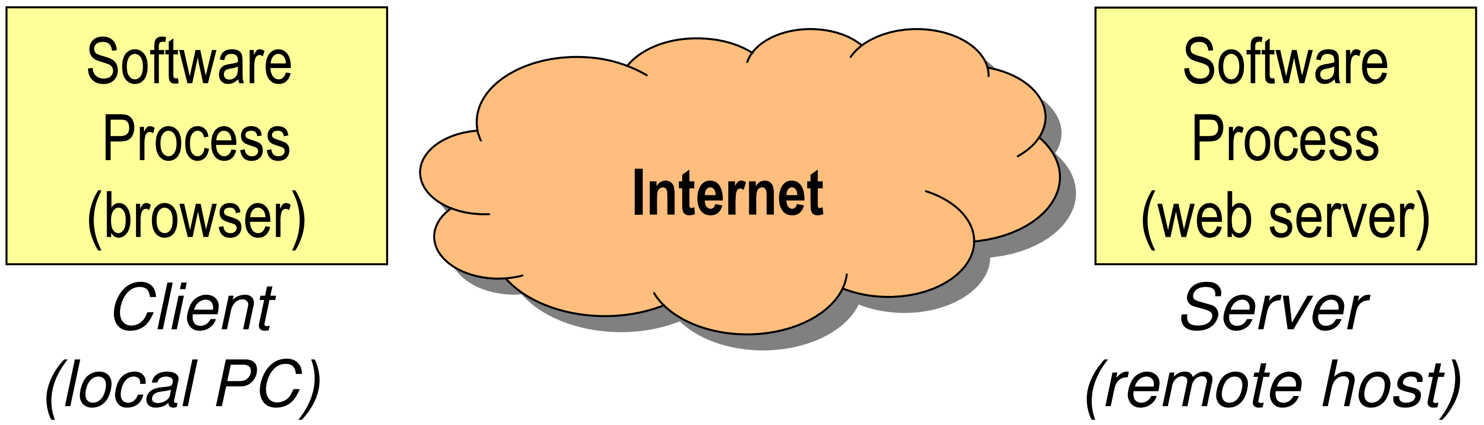
http

What happens?

Being more precise

- Who is the "client"?
  - The web browser
- Who is the server?
  - The web server
- What is the Networking application?
  - The WWW
- What is the application layer protocol?
  - HTTP

A more precise communication model:



Networking Application vs Application Layer Protocol

- Networking application
  - software processes
  - on possibly different end-systems
  - that communicate each other by exchanging messages
- Application Layer Protocol
  - define format of messages
  - define order of messages exchanged
  - define actions taken on receipt of a message

An application-layer protocol: *is only one piece (although a big piece!!) of a network application*

Example: the WWW application

many components, including:

- standard for document formats
  - HTML & HTML interpreters
- Web browsers
  - Netscape Navigator, Internet Explorer,…
- Web servers
  - Apache, Microsoft and Netscape servers, …
- Back-end Data Base DB connectivity, programming/scripting languages
  - Public domain (e.g. MYSQL) or commercial (Oracle, …)
  - ASP (active Server Pages), JSP (Java Server Pages), PHP (Perl Helper Pages) scripting embedded in html; e.g. CGI to connect to external program
- An application-layer protocol
  - the HyperText Transfer Protocol (HTTP)

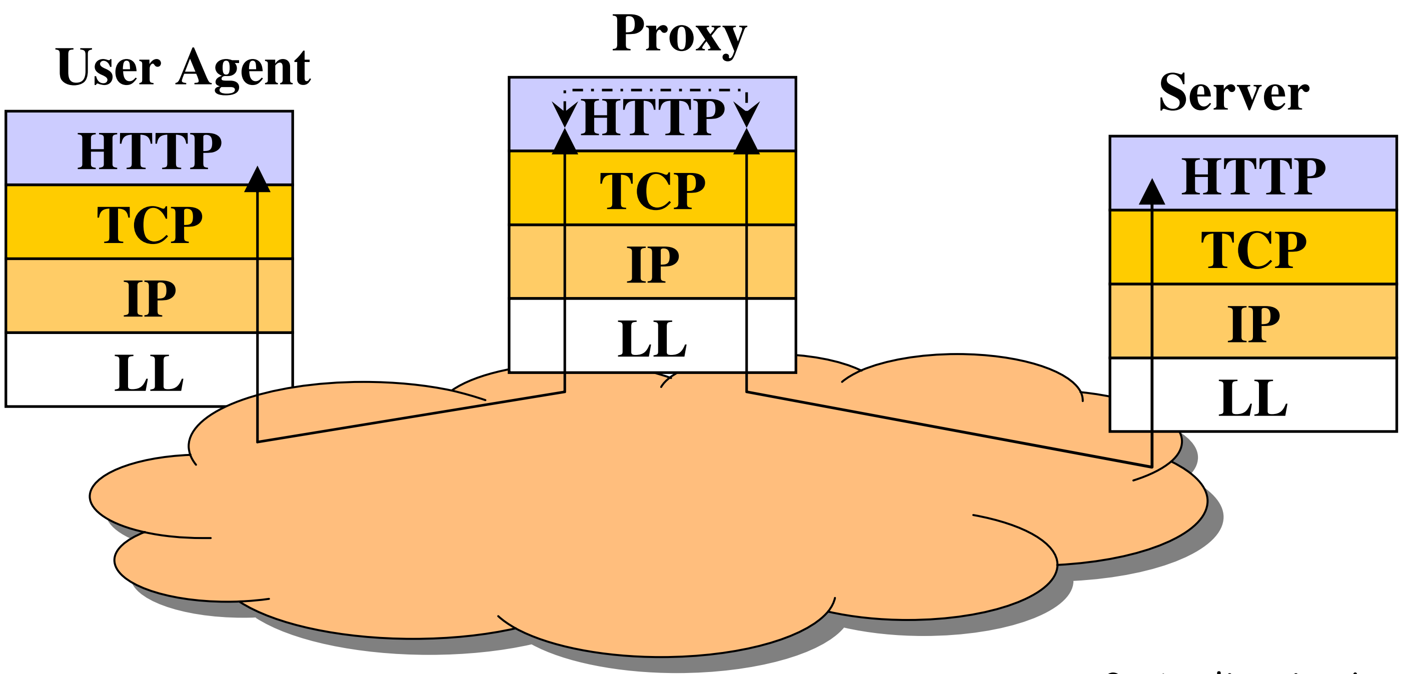
Chi gestisce l'evoluzione del Web?

- Internet Engineering Task Force
  - http e sua evoluzione→ il protocollo di comunicazione e' un RFC pubblico
  - MUST; SHOULD; MAY→ diversi livelli di compliance richiesti
- Evoluzione del Web promossa dal W3C (World Wide Web Consortium)
  - si concentra non sugli aspetti di rete ma di rappresentazione dei contenuti

Proxy

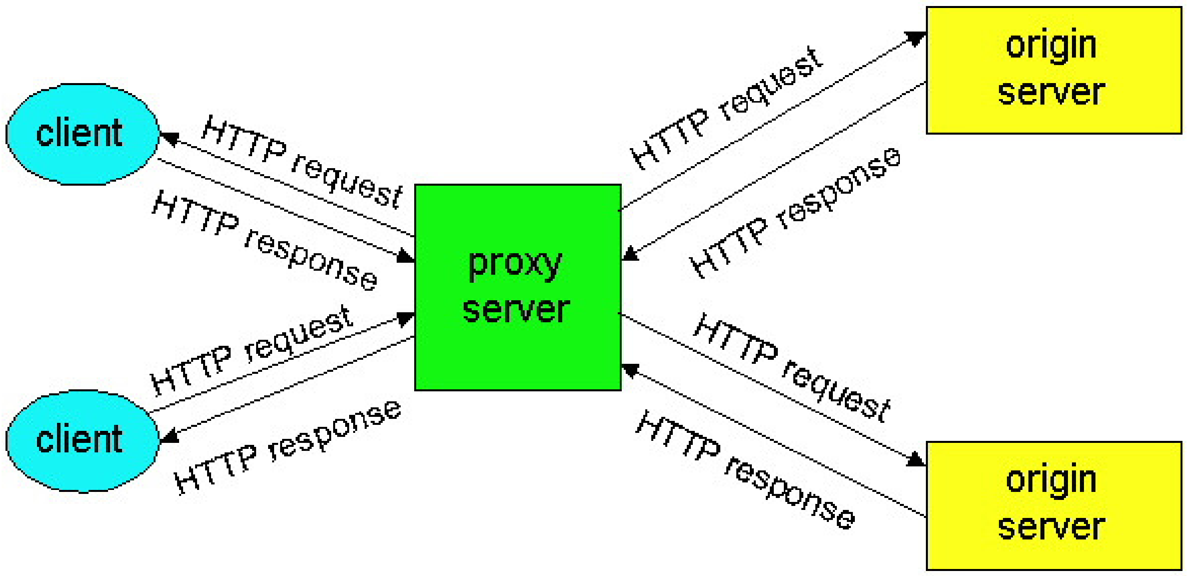
An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

- I proxy sono degli instradatori di messaggi di livello applicativo
- Devono essere sia client che server
- Il server vede arrivare tutte le richieste dal proxy (mascheramento degli utenti del proxy)



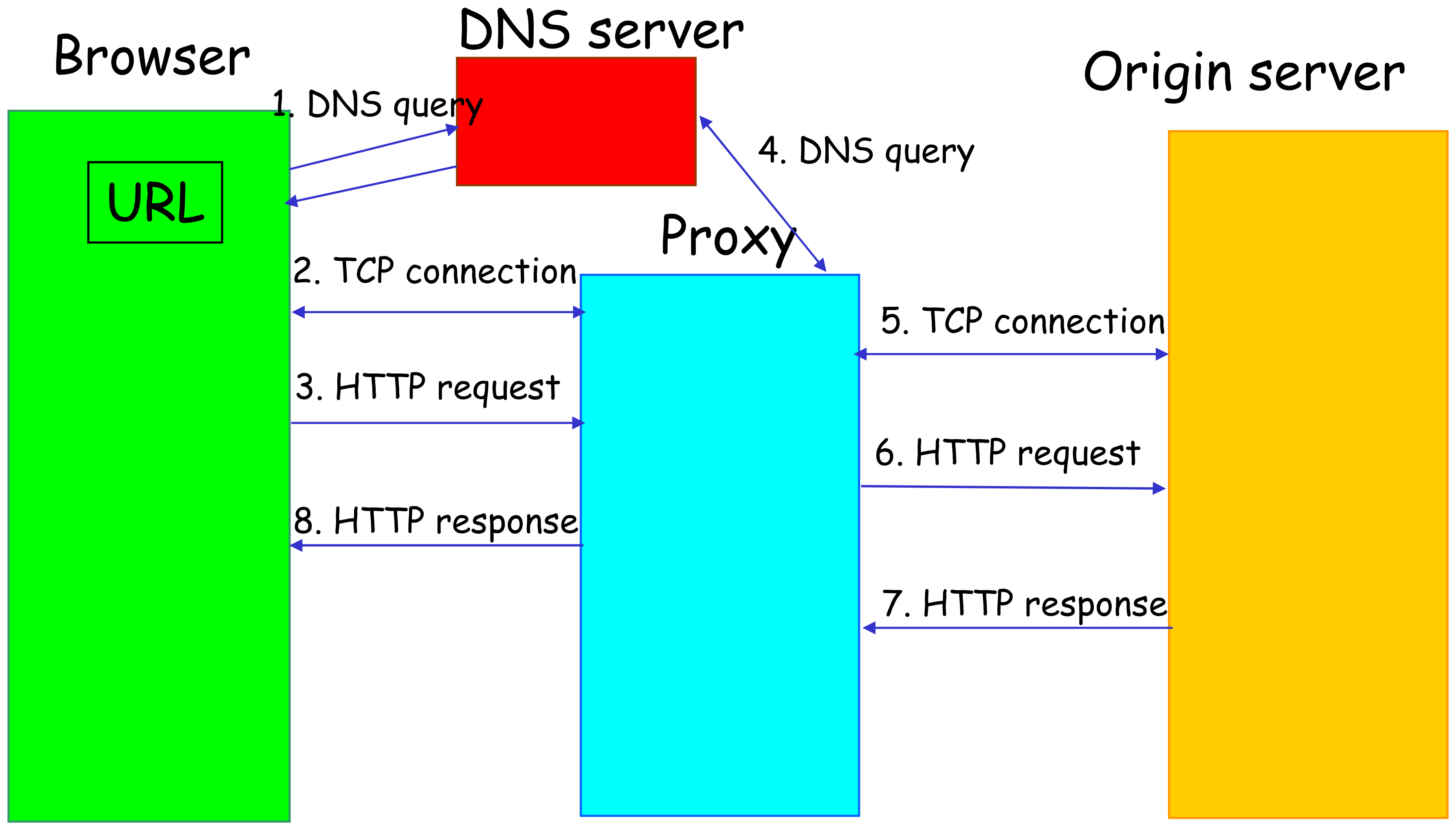
Cache di rete: uso dei proxy

- Compito principale dei proxy è fornire una grande memoria di cache
- Se un documento è contenuto nella cache viene scaricato più velocemente sul client
- Razionale: i server forniscono le stesse info o info fortemente ridondanti a comunita' locali di utenti

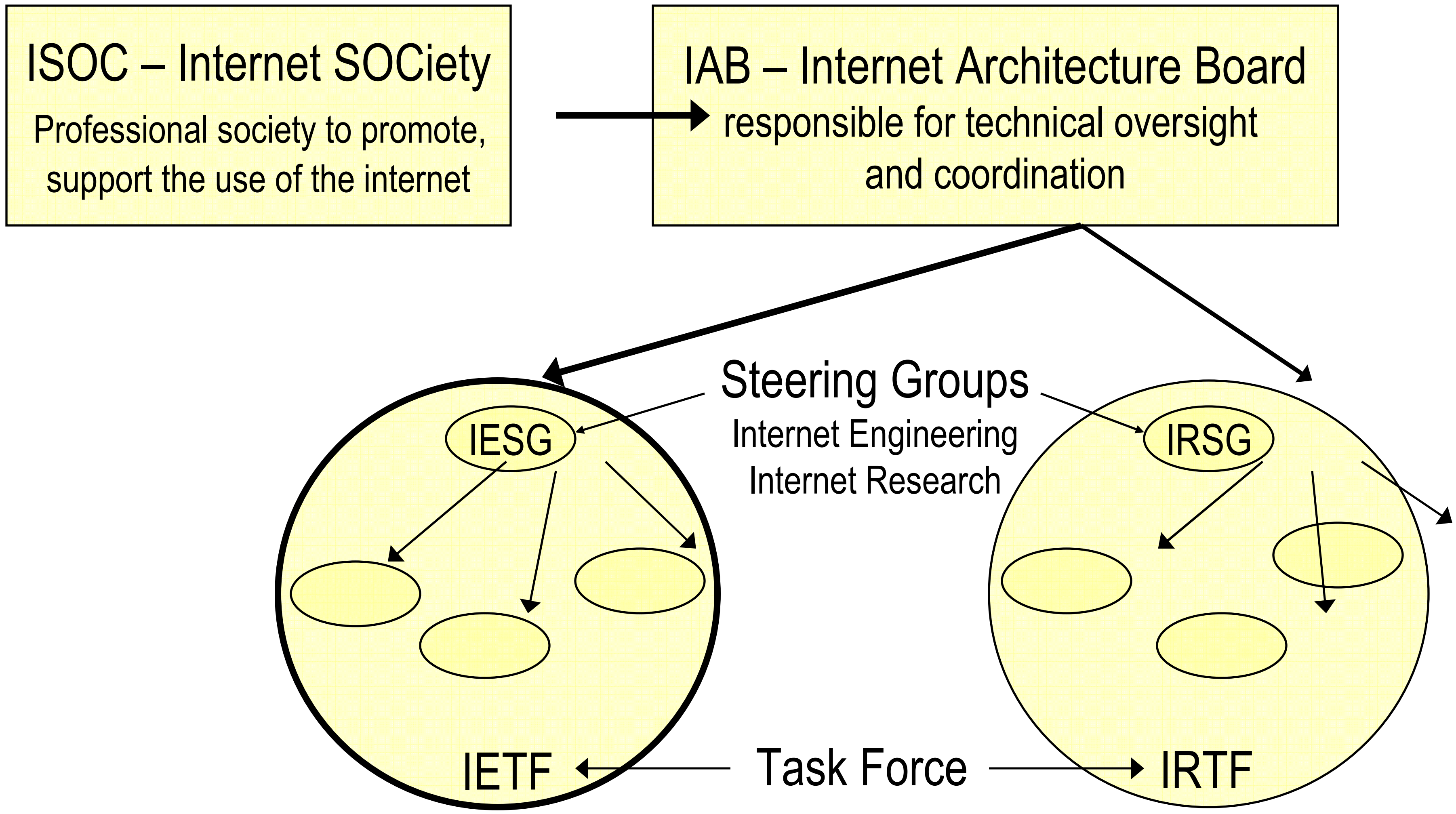


- Consistency:cached pages might not be the most recent...
- Ad clickthrough counts:how does Yahoo know how many times you accessed their pages, or ***more importantly***, their ads?

Download atraverso un proxy…



Technical Bodies Structure

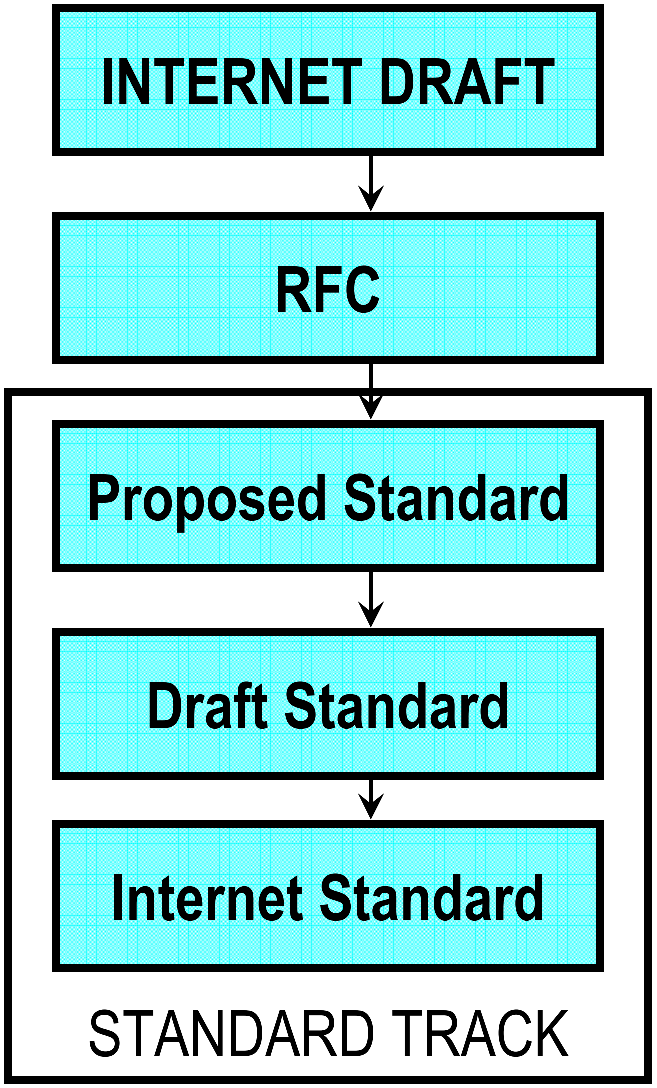

ISOC – Internet SOCiety
Professional society to promote, support the use of the internet
IAB – Internet Architecture Board
responsible for technical oversight and coordination
Steering Groups
Internet Engineering
Internet Research
IESG
IRSG
IETF
Task Force
IRTF

IETF credo

We reject kings, presidents and voting.
We believe in rough consensus
and running code

David Clark (MIT), 1992

Internet Standard Process



Draft version for information review and comments. 6 months lifetime

Official Internet publication: never expires

Entry level - protocol specification should be stable technically

At least 2 independent & interoperable implementations testing all spec. fcts

Have had significant field use and clear community interest in production use

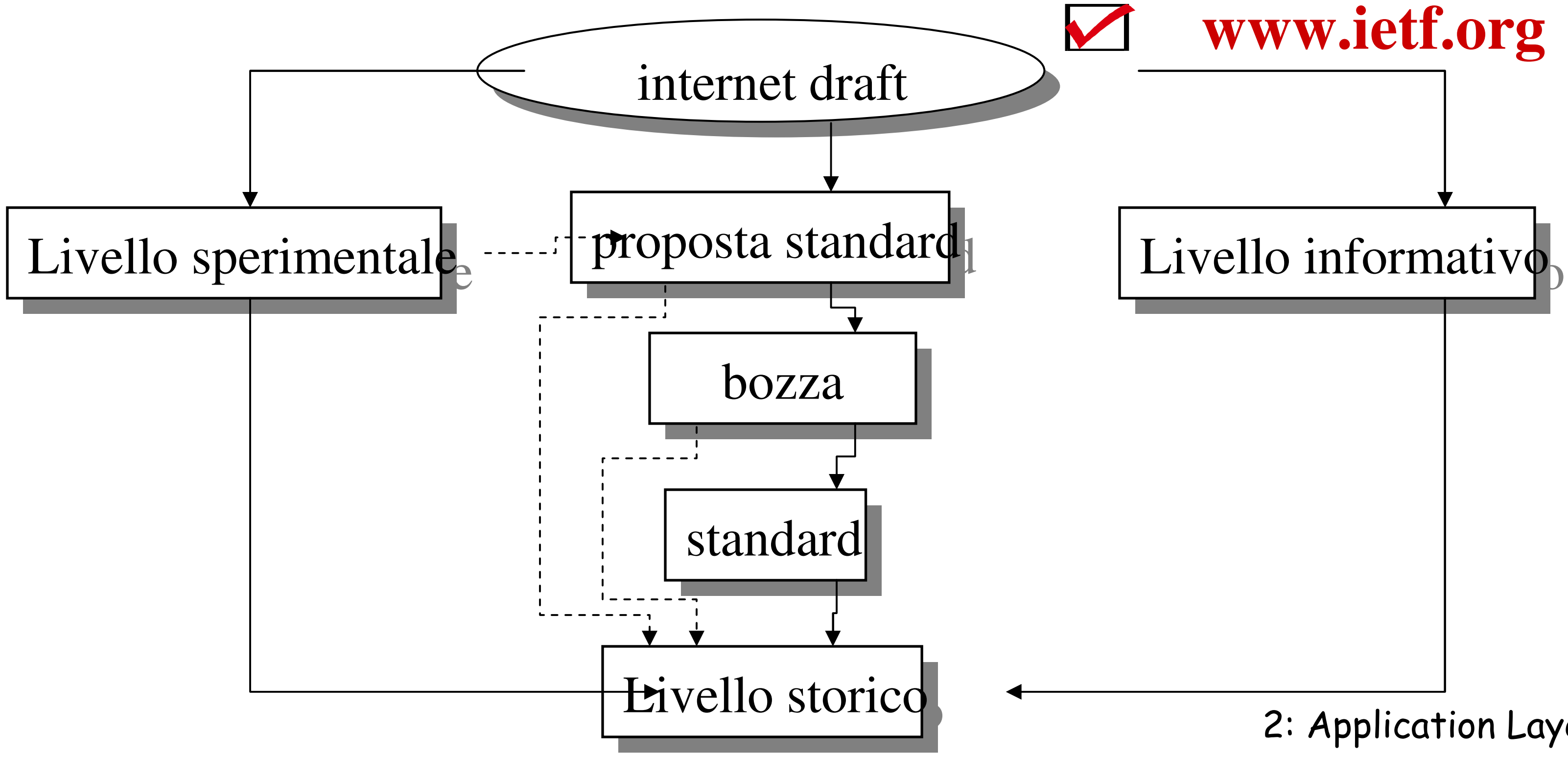
Non-Standard Track (the most common track!!)

- Specifications may not be intended to be an Internet standard
- Three labels
  - Informational—es. general info on IETF operation and how standardization progresses
  - Experimental—interesting, may become a standard but needs a larger community and more experiments
  - Historic – were in the standard track but have been removed due to lack of current use

Gli standard di Internet

- Gli standard di Internet sono documenti pubblici denominati RFC (Request For Comments)
- L'organismo che coordina la stesura degli RFC è l'IETF (Internet Engineering Task Force)

✓ **www.ietf.org**



Internet Documents

- RFC - Request For Comments
  - RFC3000 in Nov 2000
  - Updated RFCs published with new numbers
  - Not all describe protocols
  - Not all used!
- BCP - Best Current Practice
- FYI - For Your Information
  - RFC subseries: FYI = no protocol specs (es. RFC1718: the Tao of the Internet)
- STD - STanDard
  - official Internet Standard

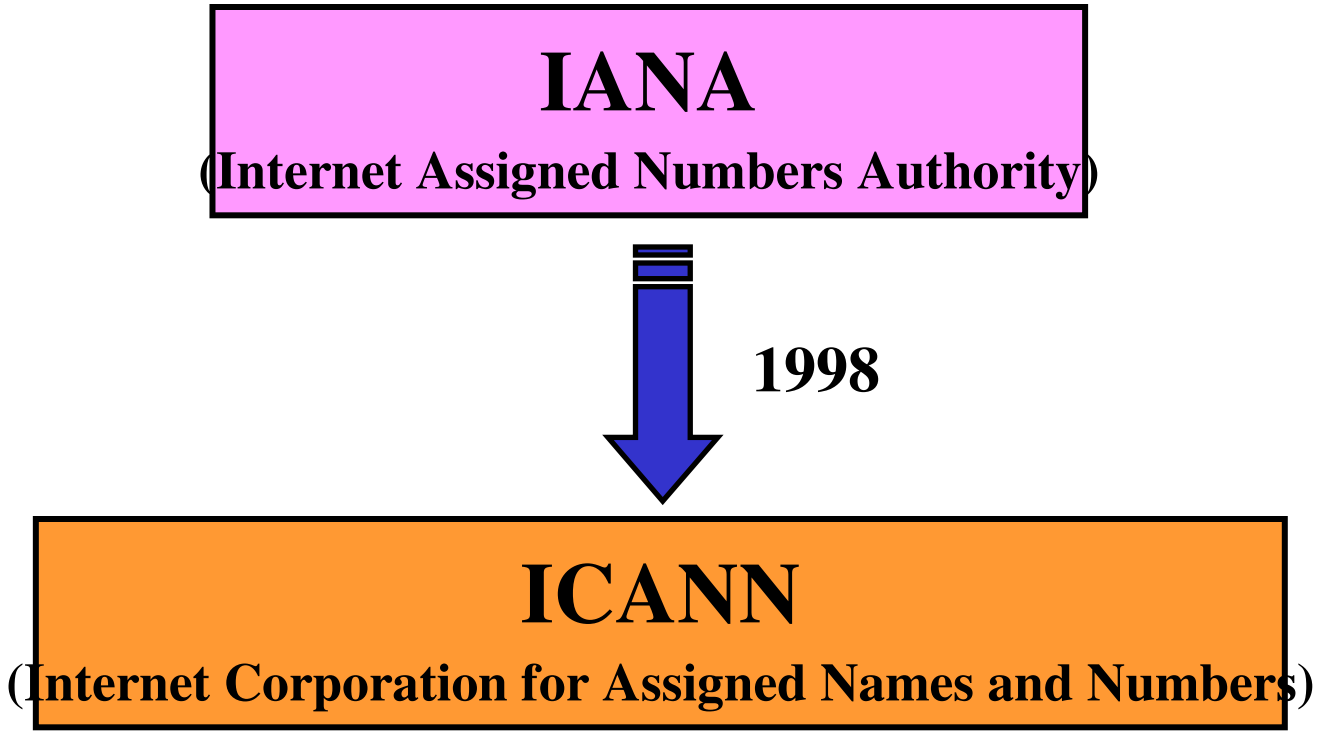
Important Documents

all RFCs from ftp://ds.internic.net/rfc
RFCs + IDs + WG: http://www.ietf.org

- RFC2300 (STD0001): Internet Official Protocol Standards (standardization process description)
- RFC1340 (STD0002): Assigned Numbers
- RFC1122 + RFC1123 (STD0003) Requirement for Internet hosts - communication layer (1122), Application and support (1123) (descri[tion of the TCP/IP architecture)

Indirizzi e nomi

- Gli indirizzi IP sono assegnati su base globale
- Internet fa uso anche di nomi simbolici che sono anch'essi assegnati su base globale


IANA
(Internet Assigned Numbers Authority)
1998
ICANN
(Internet Corporation for Assigned Names and Numbers)