Lecture 2-ter.

A communication example

Managing a HTTP v1.0 connection

G.Bianchi, G.Neglia, V.Mancuso

Università degli Studi di Palermo

Managing a HTTP request

User digits URL and press return (or clicks…).
What happens (HTTP 1.0):

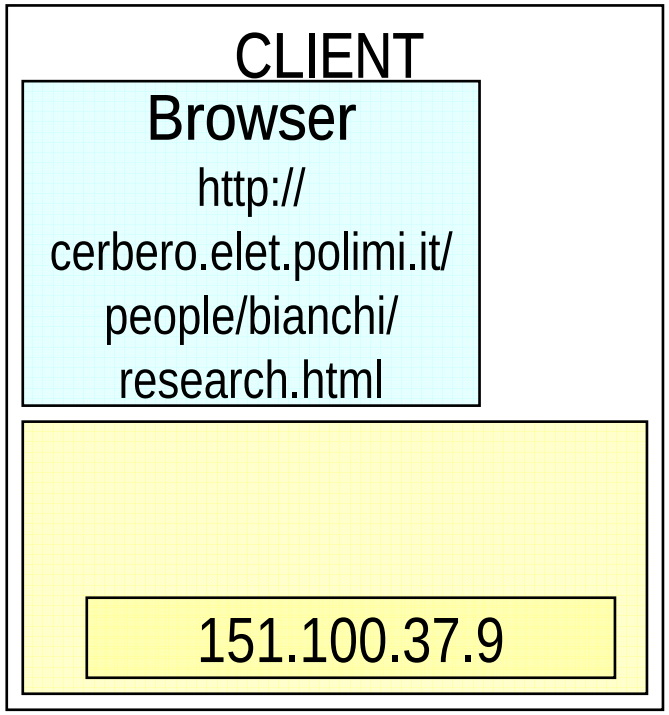
1. Browser opens a TCP transport session with server

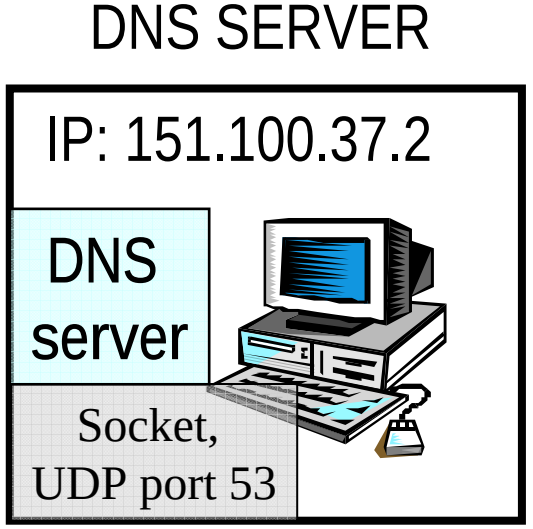
2. Within this transport session, Client sends HTTP request and receives HTTP response

3. Once finished HTTP message exchange, Server closes transport session

4. Client parses page, and iterates the above process for additional requested objects within the page

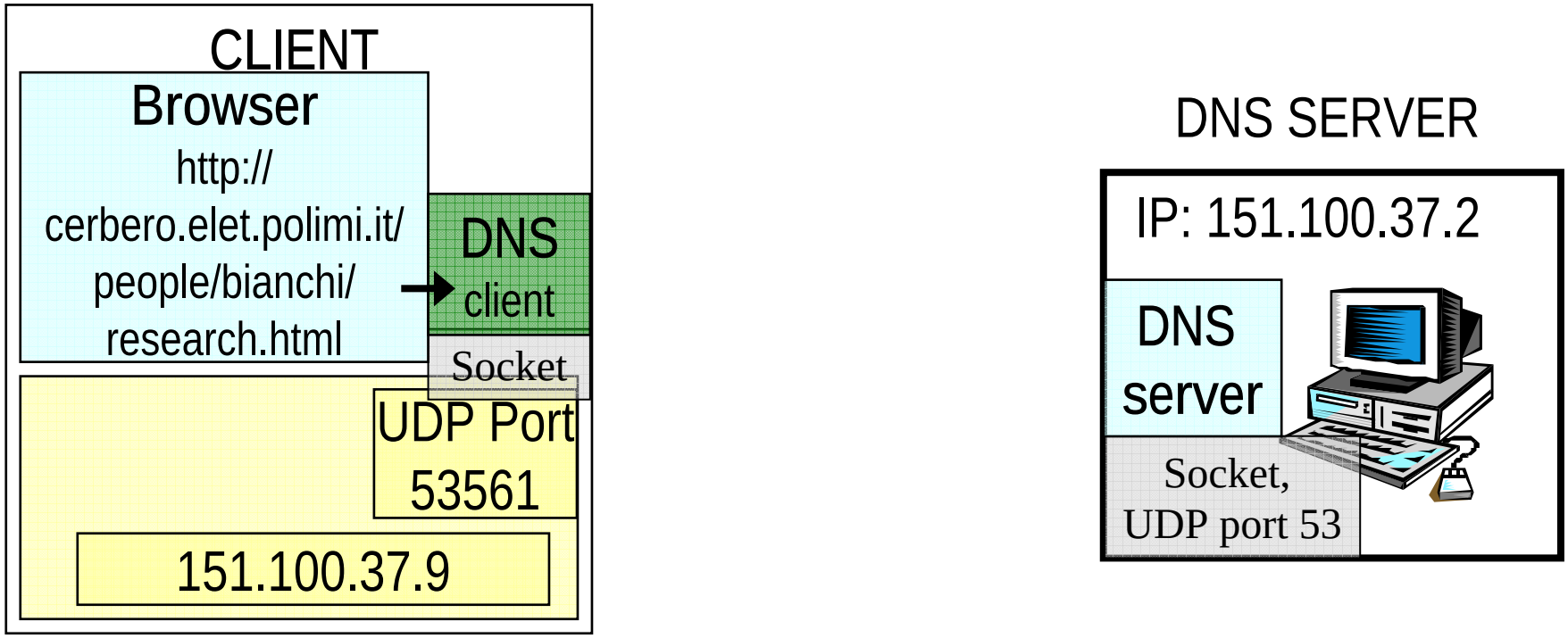
Step 1 - opening transport session: client side, step a





DNS server has a socket to accept new requests, the socket corresponds to the well known UDP port 53 (a **listen port**)

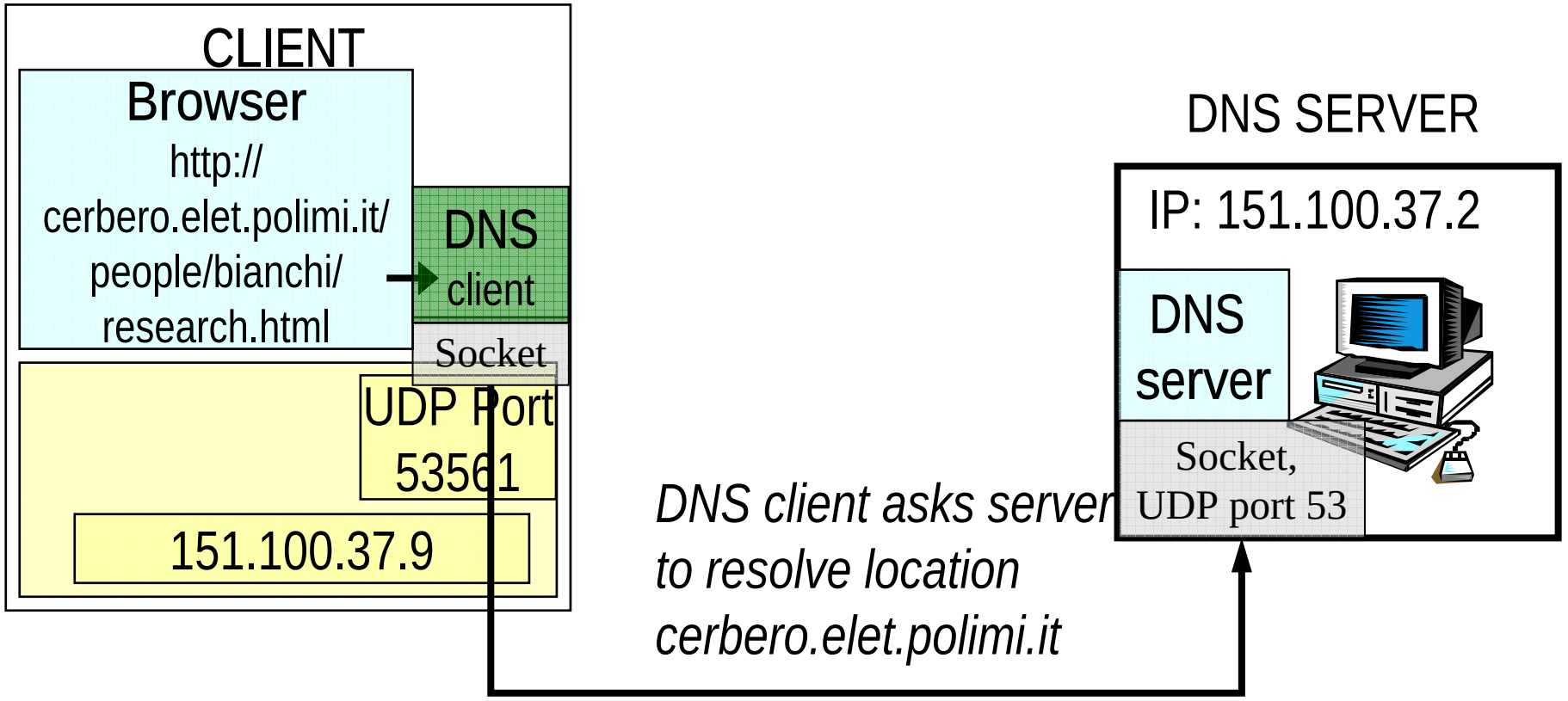
Step 1 - opening transport session: client side, step a



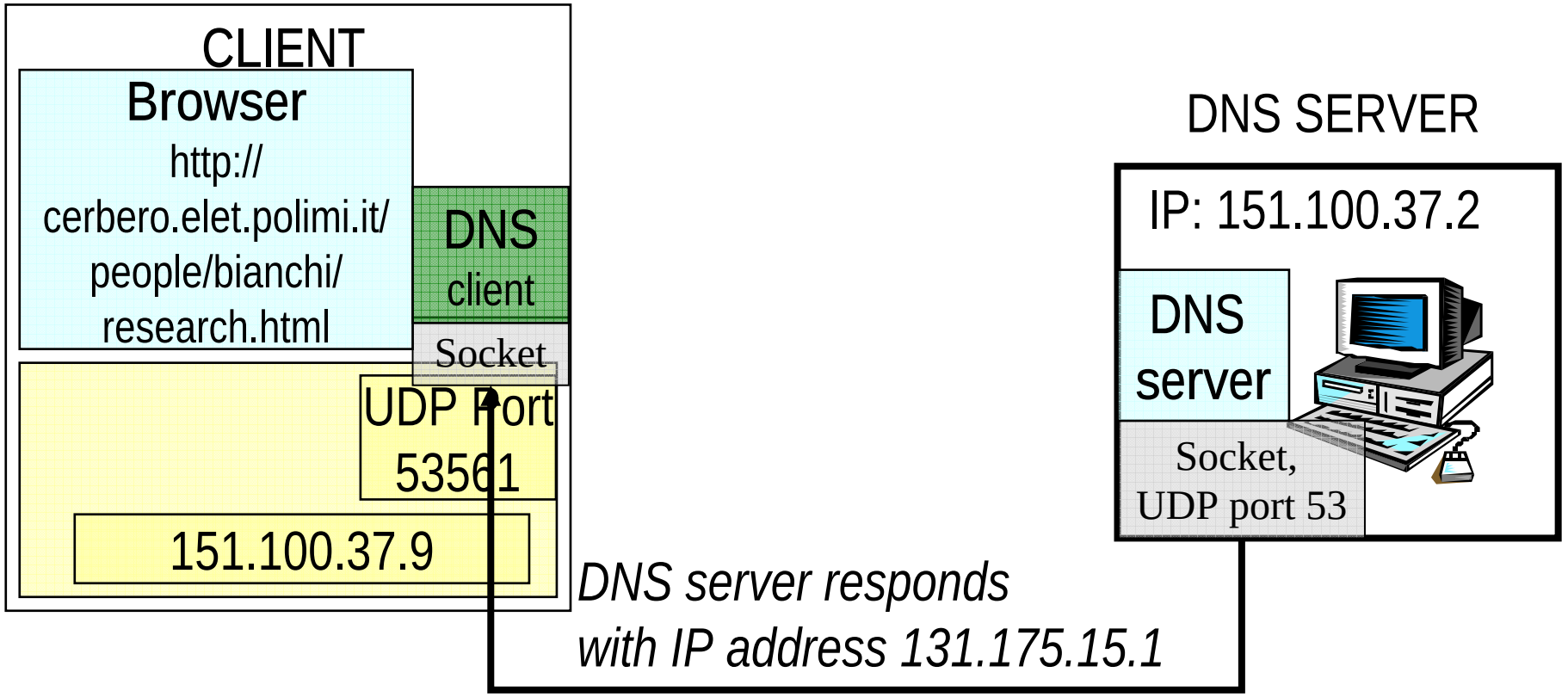
Browser asks DNS client to resolve cerbero.elet.polimi.it

DNS client opens a UDP socket, Port # is given by OS

Step 1 - opening transport session: client side, step a



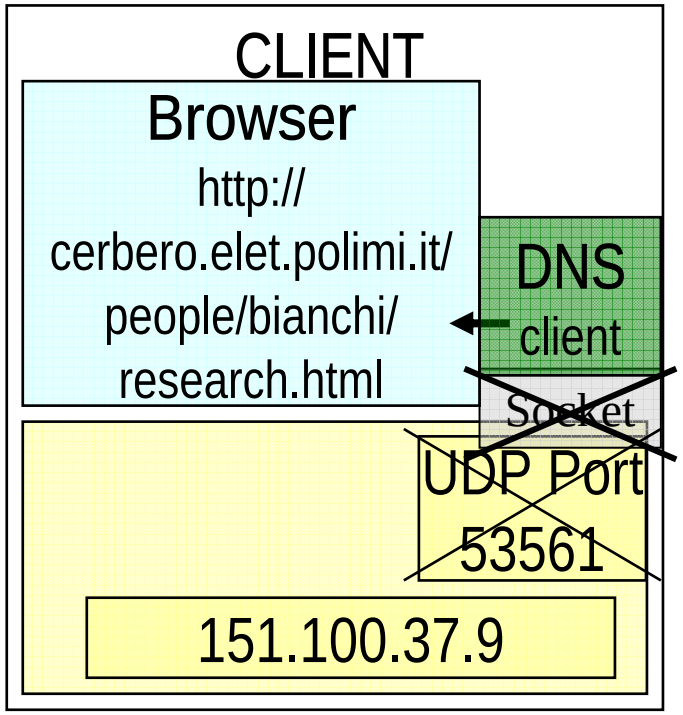
Step 1 - opening transport session: client side, step a

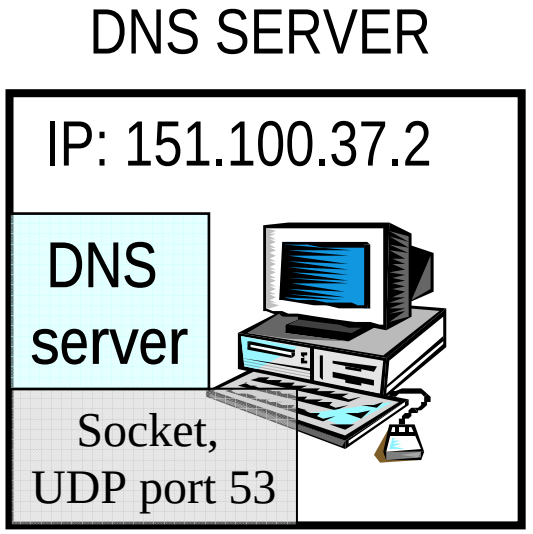


Uses UDP packet

SRC <151.100.37.2, port 53>, DEST <151.100.37.9,port=53561>

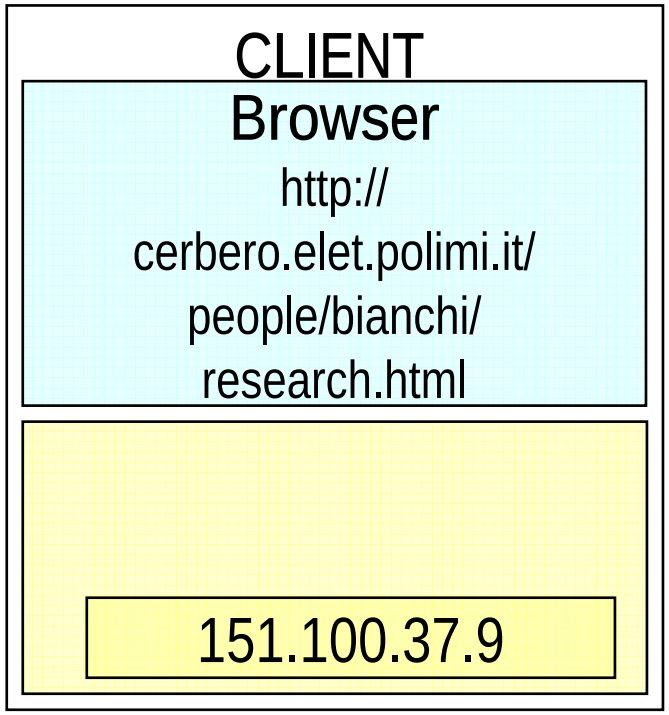
Step 1 - opening transport session: client side, step a

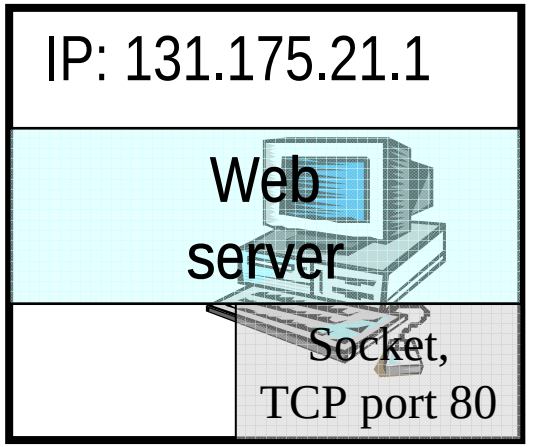




DNS solver gives the answer and closes the UDP socket

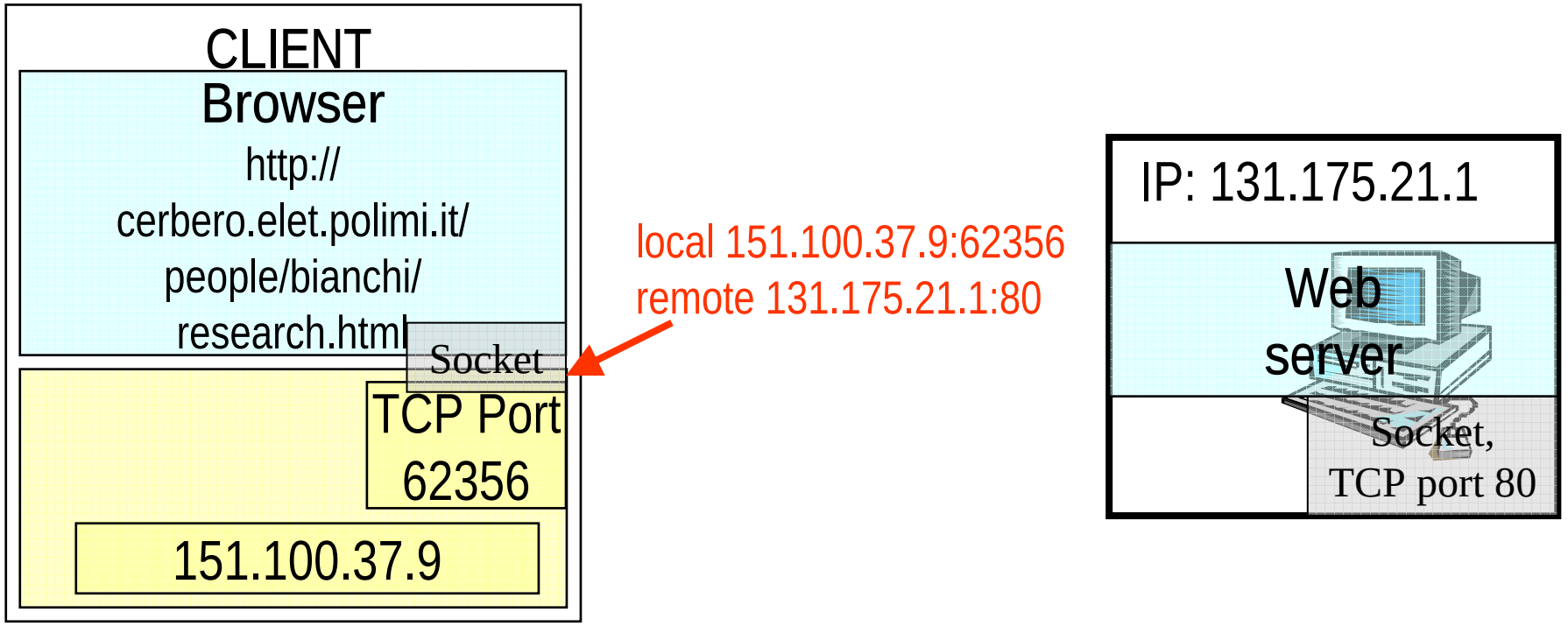
Step 1 - opening transport session: client side, step b





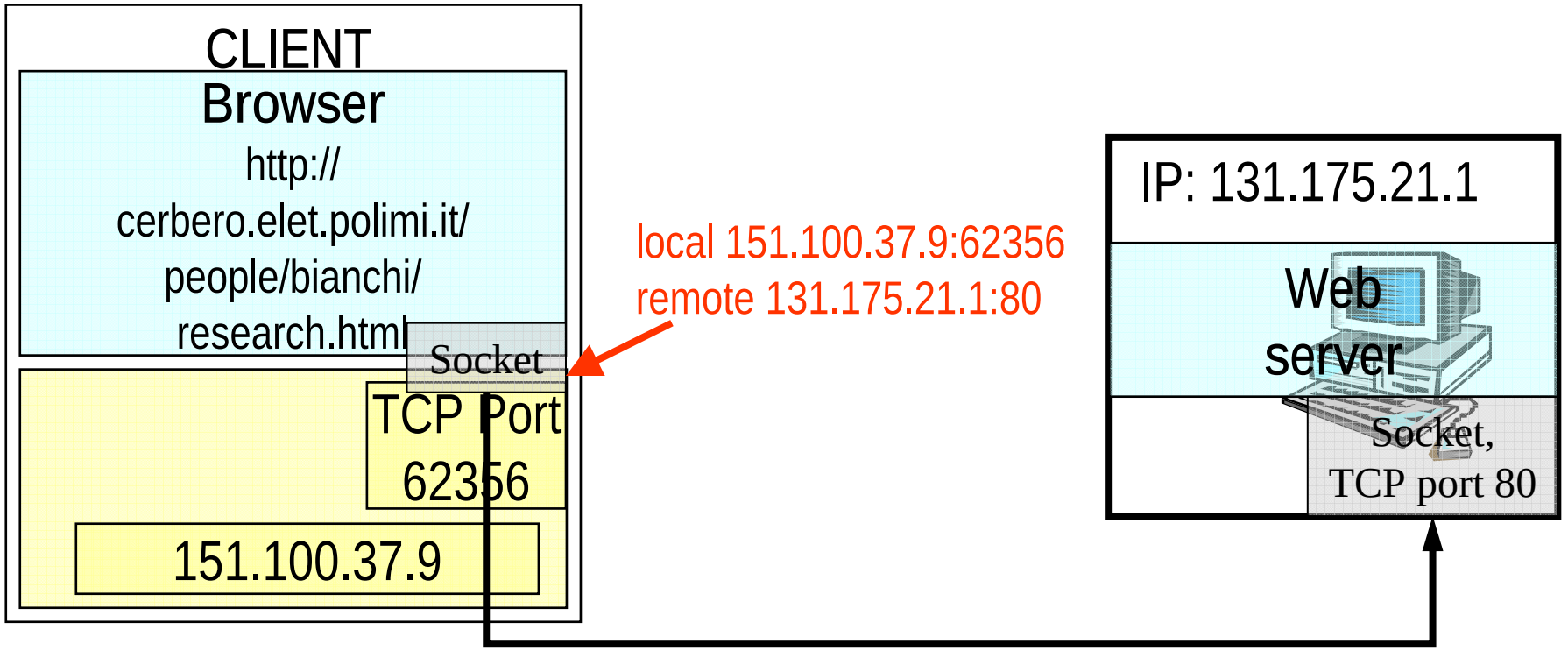
Web server has a socket to accept new requests, the socket corresponds to the well known TCP port 80 (a listen port)

Step 1 - opening transport session: client side, step b



The browser open a TCP socket towards <131.175.21.1,80>,
the sender port is given by the OS,
the socket is identified by the quintuple, it is a connection port

Step 1 - opening transport session: client side, step b



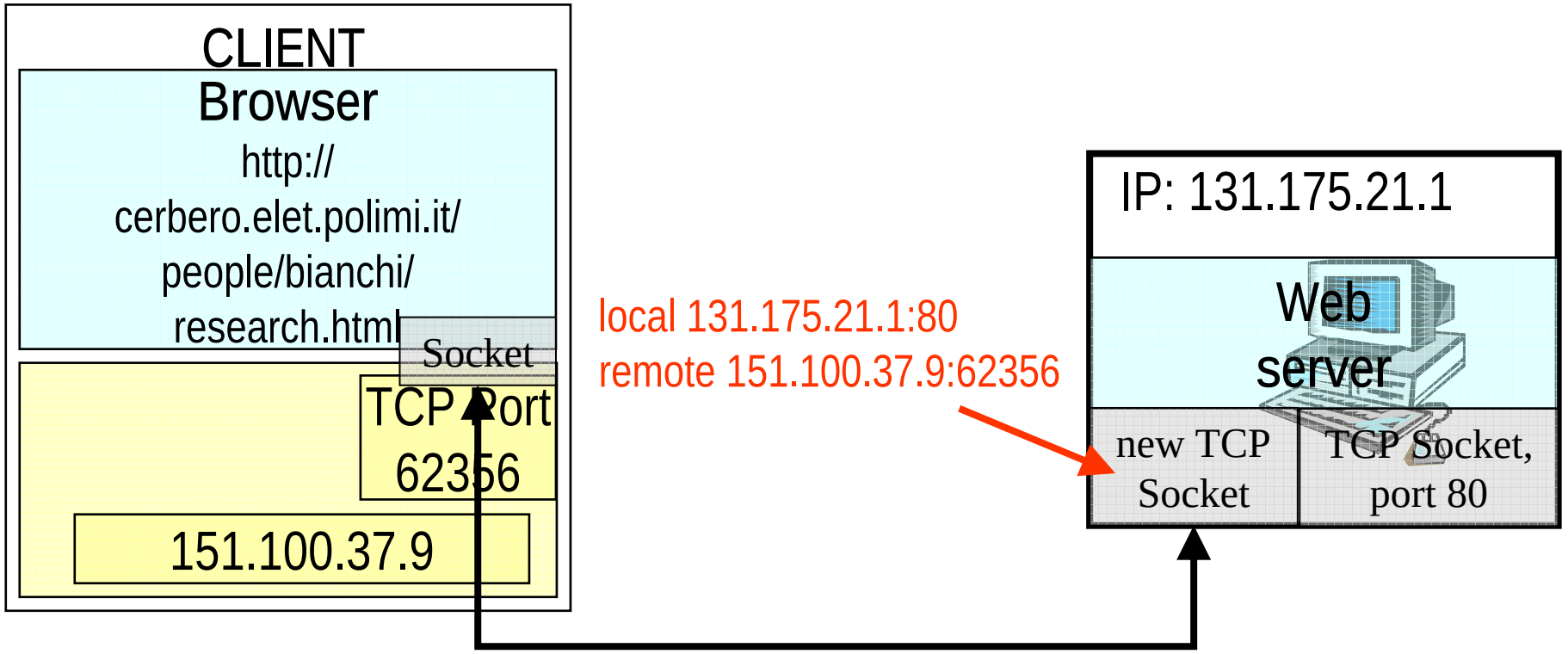
Browser sends TCP conn req to server 131.175.21.1 port 80

TCP connection
<151.100.37.9, 2345>,<131.175.21.1,80>

opening transport session: server side

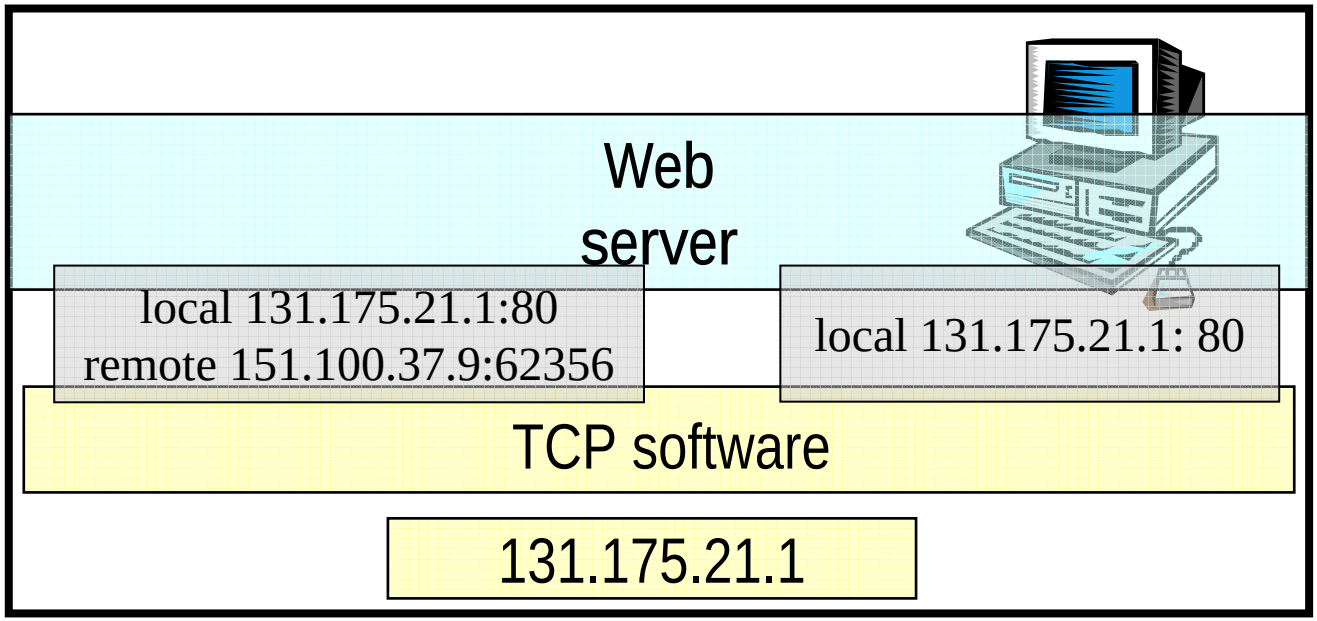
- ⇨ httpd (http daemon) process listens for arrival of connection requests from port 80.
- ⇨ Upon connection request arrival, server decides whether to accept it, and send back a TCP connection accept
- ⇨ This opens a TCP connection, uniquely identified by client address+port and server address+port 80

Step 1 - opening transport session: server side, step b



If the server accepts, it creates a new **connection socket**, dedicated to this connection and identified by the quintet

Step 1 - opening transport session: server side, step b



All the packets to the web server are addressed to 131.175.21.1:80
The transport payload is delivered to a specific connection socket, if any (e.g. when they come from 151.100.37.9:62356),
otherwise they are delivered to the listen port (but they have to be connection requests)

Remark

- **In all this described operation, clients NEED to have an IP address….**
- **Although obvious, this is not "automatic"**
  - does your home PC have an IP address?
  - And your laptop??
  - And your WAP GSM???

Step 2: talking HTTP

`http://cerbero.elet.polimi.it/people/bianchi/research.html`

➔Simplest http command: GET

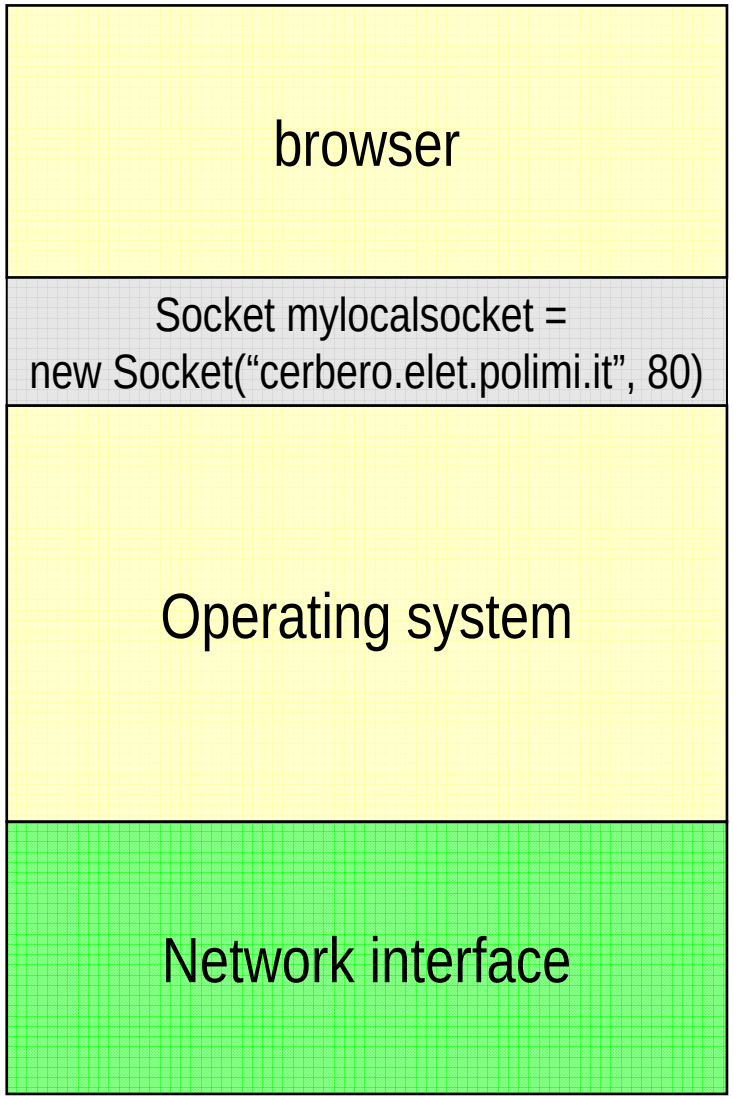
- ⇨client request:

  GET /people/bianchi/research.html HTTP/1.0

- ⇨server response
  - ➔status line (basic information)
  - ➔headers (additional information);
  - ➔blank line (nothing before CRLF)
  - ➔requested page

    more complex commands available

App. message deliver over TCP
(after TCP connection has been opened)



Prepares ASCII request message
GET /people/bianchi/research.html HTTP/1.0

Sends into output stream of opened socket

- TCP breaks message into TCP segments, each with destination port, sequence number, and checksum
- TCP hands segment to IP with dst address (not port)
- IP adds dst address to segment, to form IP datagram
- IP hands datagram to network access protocol, with address of intermediate router on the same network
- Network access protocol adds information for transportation over subnetwork & sends packet to intermediate router or host

Step 3: closing TCP conn

- ➔Server controls delivery of full page
- ➔and closes TCP connection when it is sure all the page has been received by client

Step 4: further objects

- **Client parses page**
- **if additional objects required** ***(pages, images, on the same server or on different servers):***
  - ⇨ for each, open new TCP connection
  - ⇨ and repeat described operation

This operation creates significant performance drawbacks!
is typical of HTTP/1.0 without Keep_Alive
pipelining considered in HTTP/1.1

TCP level analysis