

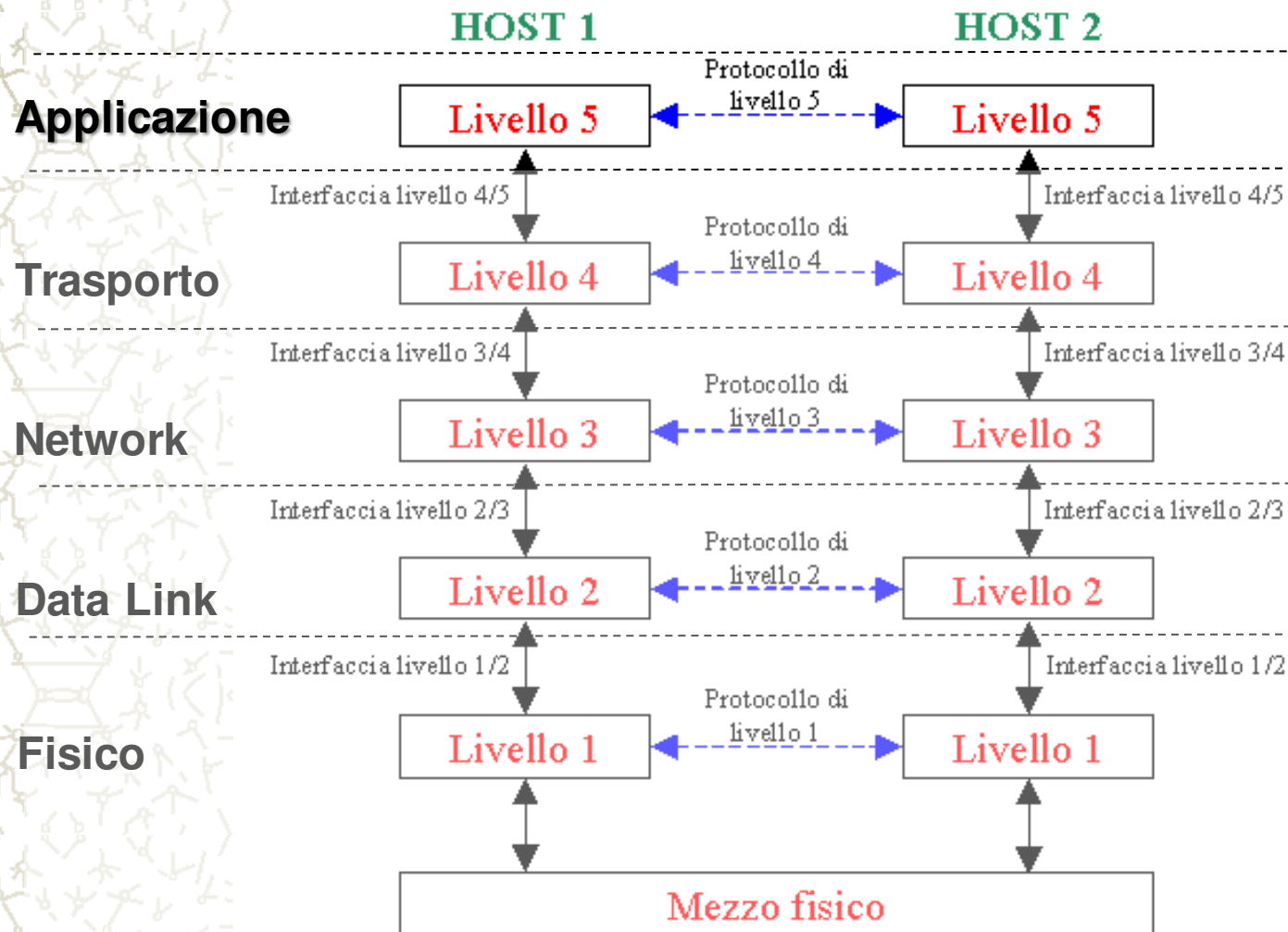
A decorative vertical strip on the left side of the slide, featuring a repeating pattern of network-related symbols such as nodes, arrows, and geometric shapes in a light gray color.

# **Reti di Calcolatori**

Application

A horizontal yellow bar at the bottom right of the slide, consisting of two adjacent rectangular segments of different shades of yellow.

# Modello ISO-OSI



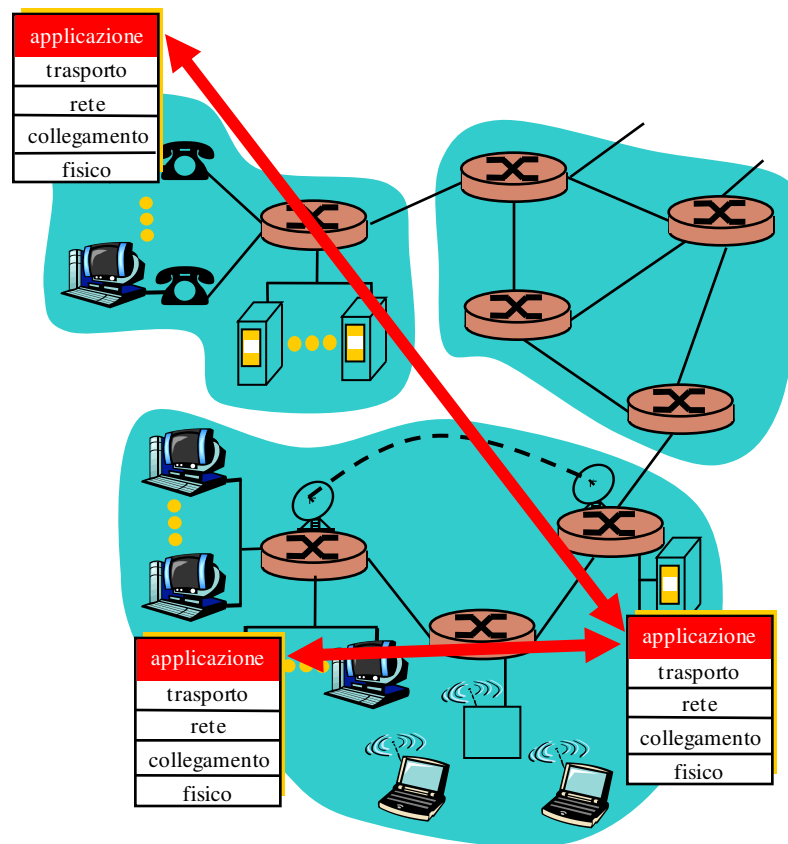
# Creare un'applicazione di rete

## Scrivere programmi che

- girano su sistemi terminali diversi
- comunicano attraverso la rete
- Ad es. il Web: il software di un server Web comunica con il software di un browser

## software in grado di funzionare su più macchine

- non occorre predisporre programmi per i dispositivi del nucleo della rete, quali router o commutatori Ethernet



# Alcune diffuse applicazioni di rete

- ☐ Posta elettronica
- ☐ Web
- ☐ Messaggistica istantanea
- ☐ Autenticazione in un calcolatore remoto (Telnet e SSH)
- ☐ Condivisione di file P2P
- ☐ Giochi multiutente via rete
- ☐ Streaming di video-clip memorizzati

☐ Telefonia via Internet

☐ Videoconferenza in tempo reale



# Alcuni Protocolli di rete

- Spesso basati sul modello client-server

➔ **DNS (Domain Name System)**  
Risoluzione dei nomi degli host

➔ **SMTP (Simple Mail Transfer Protocol)** e email  
sendmail o postfix

**POP3 (Post Office Protocol)** e **IMAP (Interactive Mail Access Protocol)**  
Accesso remoto alle caselle di posta elettronica

**FTP (File Transfer Protocol)**  
Trasferimento file remoti

**SNMP (Simple Network Management Protocol)**  
Gestione di apparati di rete

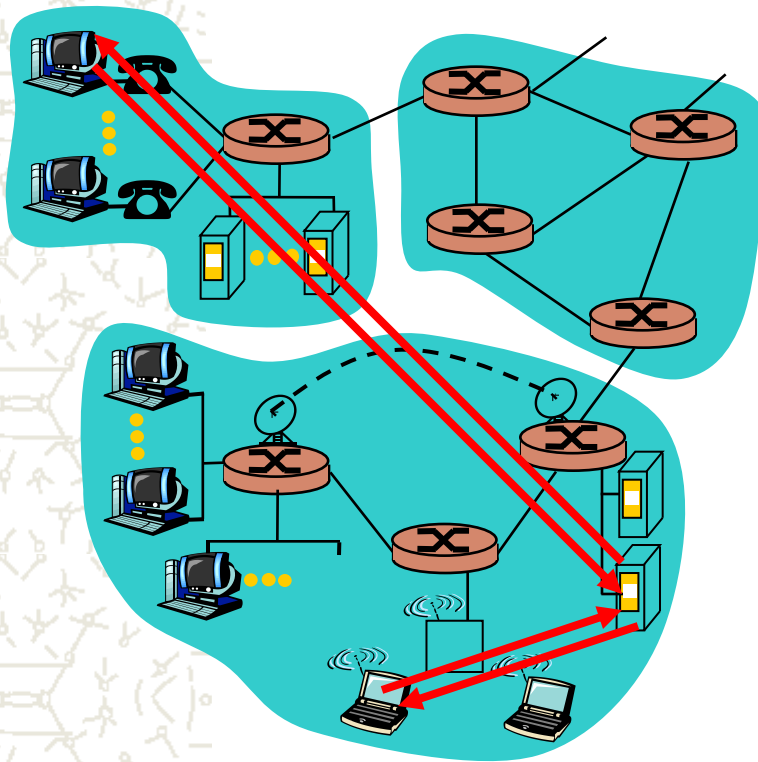
➔ **HTTP (HyperText Transfer Protocol)** e il WWW  
Server web e browsers

**Telnet, rlogin, rcp, ssh, sftp, nfs, lpd, ...**

# Architetture delle applicazioni di rete

- ☐ Client-server
- ☐ Peer-to-peer (P2P)
- ☐ Architetture ibride (client-server e P2P)

# Architettura client-server



## server:

- host sempre attivo
- indirizzo IP fisso
- server farm per creare un potente server virtuale

## client:

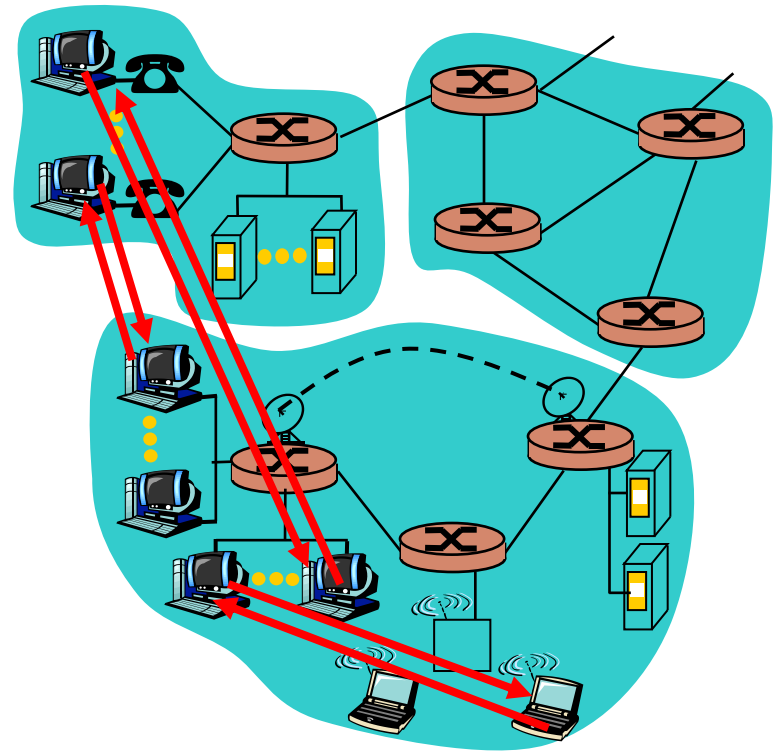
- comunica con il server
- può contattare il server in qualunque momento
- può avere indirizzi IP dinamici
- non comunica direttamente con gli altri client

# Architettura P2P pura

- ☐ non c'è un server sempre attivo
- ☐ coppie arbitrarie di host (peer) comunicano direttamente tra loro
- ☐ i peer non devono necessariamente essere sempre attivi, e cambiano indirizzo IP
- ☐ Un esempio: Gnutella

Facilmente scalabile

Difficile da gestire





# Ibridi (client-server e P2P)

## Napster

- Scambio di file secondo la logica P2P
- Ricerca di file centralizzata:
  - i peer registrano il loro contenuto presso un server centrale
  - i peer chiedono allo stesso server centrale di localizzare il contenuto

## Messaggistica istantanea

- La chat tra due utenti è del tipo P2P
- Individuazione della presenza/location centralizzata:
  - l'utente registra il suo indirizzo IP sul server centrale quando è disponibile online
  - l'utente contatta il server centrale per conoscere gli indirizzi IP dei suoi amici

# Remarking: Processi comunicanti

**Processo:** programma in esecuzione su di un host.

- All'interno dello stesso host, due processi comunicano utilizzando **scemi interprocesso** (definiti dal SO).
- processi su host differenti comunicano attraverso lo scambio di **messaggi**

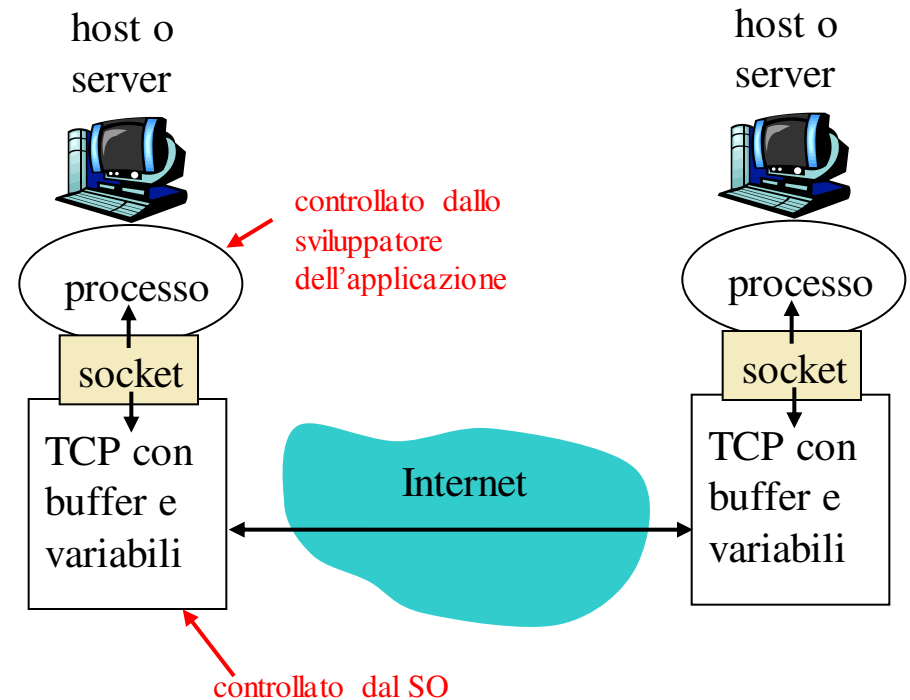
**Processo client:** processo che dà inizio alla comunicazione

**Processo server :** processo che attende di essere contattato

- Nota: le applicazioni con architetture P2P hanno processi client e processi server

# Remarking: Socket

- ❑ un processo invia/riceve messaggi a/dalla sua **socket**
- ❑ una socket è analoga a una porta
  - un processo che vuole inviare un messaggio, lo fa uscire dalla propria “porta” (socket)
  - il processo presuppone l’esistenza di un’infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla “porta” del processo di destinazione
- ❑ API: Application Program Interface, un set di protocolli e procedure per la creazione di applicazioni



# Remarking: Processi di indirizzamento

- ☐ Affinché un processo su un host invii un messaggio a un processo su un altro host, il mittente deve identificare il processo destinatario.
- ☐ Un host A ha un indirizzo IP univoco a 32 bit
- ☐ **D:** È sufficiente conoscere l'indirizzo IP dell'host su cui è in esecuzione il processo per identificare il processo stesso?
- ☐ **Risposta:** No, sullo stesso host possono essere in esecuzione molti processi.
- ☐ L'identificatore comprende sia l'indirizzo IP che i **numeri di porta** associati al processo in esecuzione su un host.
- ☐ Esempi di numeri di porta:
  - HTTP server: 80
  - Mail server: 25

# Quale servizio di trasporto richiede un'applicazione?

## Perdita di dati

- alcune applicazioni (ad esempio, audio) possono tollerare qualche perdita
- altre applicazioni (ad esempio, trasferimento di file, telnet) richiedono un trasferimento dati affidabile al 100%

## Temporizzazione

- alcune applicazioni (ad esempio, telefonia Internet, giochi interattivi) per essere “realistiche” richiedono piccoli ritardi

## Ampiezza di banda

- ❑ alcune applicazioni (ad esempio, quelle multimediali) per essere “efficaci” richiedono un'ampiezza di banda minima
- ❑ altre applicazioni (“le applicazioni elastiche”) utilizzano l'ampiezza di banda che si rende disponibile

# Requisiti del servizio di trasporto di alcune applicazioni comuni

<b>Applicazione</b>	<b>Tolleranza alla perdita di dati</b>	<b>Ampiezza di banda</b>	<b>Sensibilità al tempo</b>
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/video in tempo reale	Sì	Audio: da 5 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi Kbps	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

# Remarking: Servizi dei protocolli di trasporto Internet

## Servizio di TCP:

- *orientato alla connessione*: è richiesto un setup fra i processi client e server
- *trasporto affidabile* fra i processi d'invio e di ricezione
- *controllo di flusso*: il mittente non vuole sovraccaricare il destinatario
- *controllo della congestione*: “strozza” il processo d'invio quando la rete è sovraccaricata
- *non offre*: temporizzazione, ampiezza di banda minima

## Servizio di UDP:

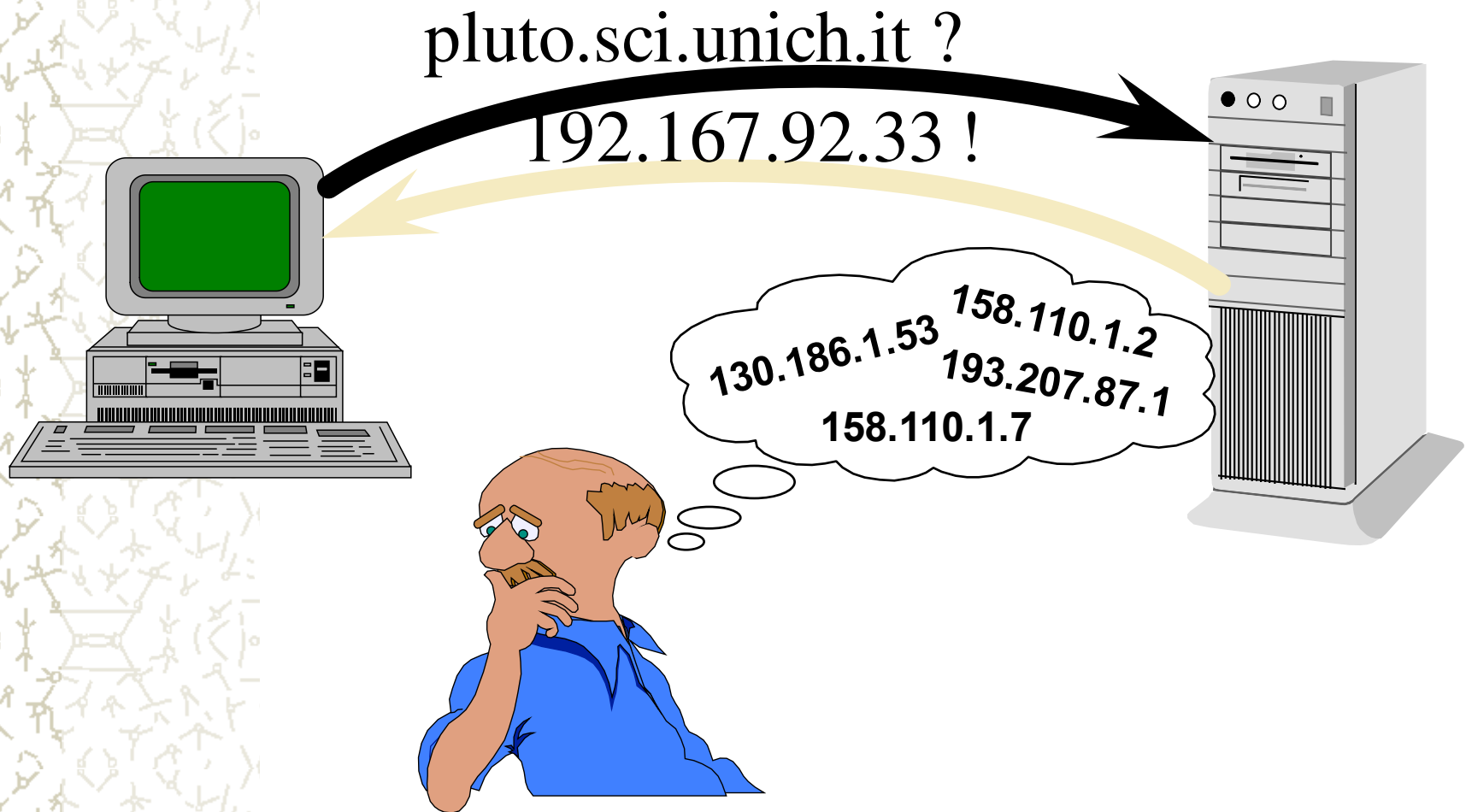
- trasferimento dati inaffidabile fra i processi d'invio e di ricezione
- non offre: setup della connessione, affidabilità, controllo di flusso, controllo della congestione, temporizzazione né ampiezza di banda minima

# Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

<b>Applicazione</b>	<b>Protocollo a livello applicazione</b>	<b>Protocollo di trasporto sottostante</b>
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	Proprietario (ad esempio, RealNetworks)	TCP o UDP
Telefonia Internet	Proprietario (ad esempio, Vonage, Dialpad)	Tipicamente UDP



# Domain Name Server (DNS)



# Domain Name System (DNS)

E' un database distribuito che permette di convertire i nomi simbolici degli host negli indirizzi IP numerici

E' più facile ricordare (e dà più informazioni)

[www.google.it](http://www.google.it)

che

64.233.167.99

**Per comunicare su rete serve l'indirizzo IP!**

Quando gli host sono pochi la tabella di conversione può essere memorizzata localmente ad ogni host (file /etc/hosts in Unix)

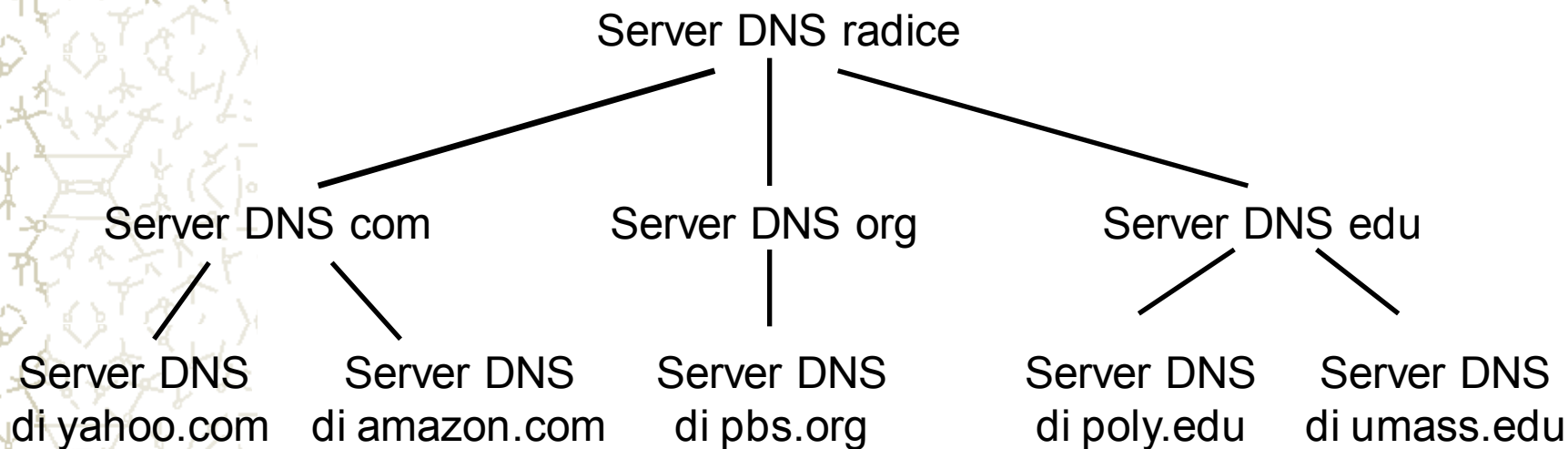
Il DNS è distribuito: ogni sito mantiene il suo database relativo agli host locali eseguendo su una macchina il **server DNS** che può essere interrogato da altri sistemi su Internet

# Funzionamento DNS

Prospettiva del client: DNS come scatola nera (servizio di traduzione).

1. Il cliente invia un messaggio di richiesta DNS specificando l'hostname che deve essere trasformato in indirizzo IP
2. Dopo un ritardo che va da msec a decine di secondi il client riceve un messaggio di risposta DNS che fornisce la correlazione desiderata.
3. La scatola nera è molto complessa e consiste di un gran numero di server dei nomi distribuiti sulla terra.

# Database distribuiti e gerarchici

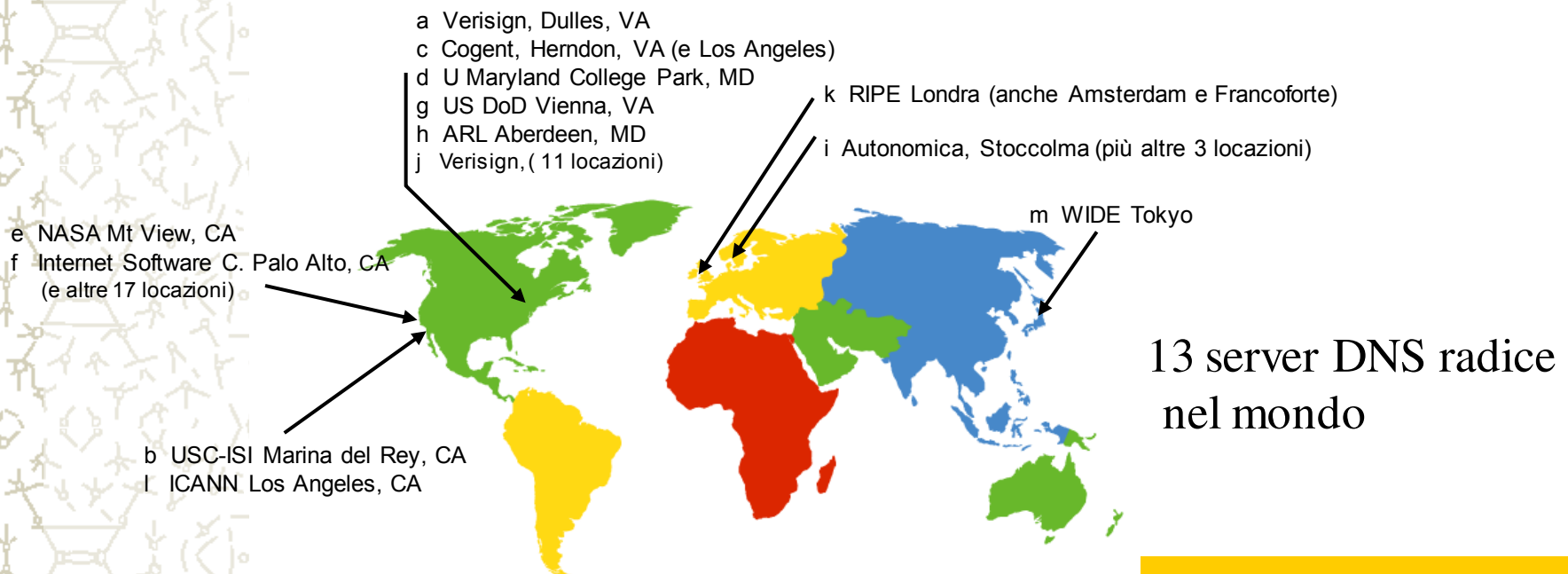


Il client vuole l'IP di [www.amazon.com](http://www.amazon.com)

- Il client interroga il server radice per trovare il server DNS com
- Il client interroga il server DNS com per ottenere il server DNS amazon.com
- Il client interroga il server DNS amazon.com per ottenere l'indirizzo IP di [www.amazon.com](http://www.amazon.com)

# DNS: server DNS radice

- contattato da un server DNS locale che non può tradurre il nome
- server DNS radice:
  - contatta un server DNS autorizzato se non conosce la mappatura
  - ottiene la mappatura
  - restituisce la mappatura al server DNS locale



# Server TLD e server di competenza

- **Server TLD (top-level domain):** si occupano dei domini com, org, net, edu, ecc. e di tutti i domini locali di alto livello, quali uk, fr, ca e jp.
  - Network Solutions gestisce i server TLD per il dominio com
  - Educause gestisce quelli per il dominio edu
- **Server di competenza (*authoritative server*):** ogni organizzazione dotata di host Internet pubblicamente accessibili (quali i server web e i server di posta) deve fornire i record DNS di pubblico dominio che mappano i nomi di tali host in indirizzi IP.
  - possono essere mantenuti dall'organizzazione o dal service provider

# DNS: caratteristiche principali

- database distribuito
- basato sul modello client/server
- tre componenti principali:
  - **Resolver** (client per l'interrogazione del name server)
  - **Spazio dei nomi** e informazioni associate  
(Resource Record - RR)
  - **Name server** (application server che mantiene i dati)
- accesso veloce ai dati (database in memoria centrale e meccanismo di caching)

# Il resolver

- Quando un'applicazione deve convertire un nome di host nell'indirizzo IP o viceversa, chiama una procedura resolver che contatta il DNS server del suo dominio
- In Java la procedura è incapsulata nella classe `InetAddress`
- In C sono disponibili le chiamate di libreria `gethostbyname` e `gethostbyaddr`
- Occorre specificare l'indirizzo IP del **DNS server** (o **Name Server**) a cui il resolver invia le sue interrogazioni (in Unix `/etc/resolv.conf`)
- Il protocollo di comunicazione fra resolver e Name server è del livello applicativo ed è basato su scambio di pacchetti UDP (porta server 53) o TCP
- `nslookup` è un programma client che permette di esplorare la tabella del Name Server



# Struttura dei nomi simbolici

Dal punto di vista sintattico un nome è una sequenza di segmenti alfanumerici separati da un punto.

Es: di.unisa.it

- La prima parte del nome rappresenta uno specifico calcolatore
- La seconda parte rappresenta il gruppo che lo possiede
- La terza parte (in generale, quella più a destra) rappresenta il top level domain (TLD)

# TLD (Top level domains)

Esistono 4 categorie principali:

1. Codici ISO 3165 identificativi delle singole nazioni (ccTLD → country code TLD)
2. Tre TLD che esistono solo negli Stati Uniti (usTLD): mil, edu, gov
3. TLD generici (gTLD) utilizzabili da soggetti in qualunque nazione: **org**, **com**, **net**,...
4. Suffisso int riservato alle organizzazioni nate in seguito a trattati internazionali

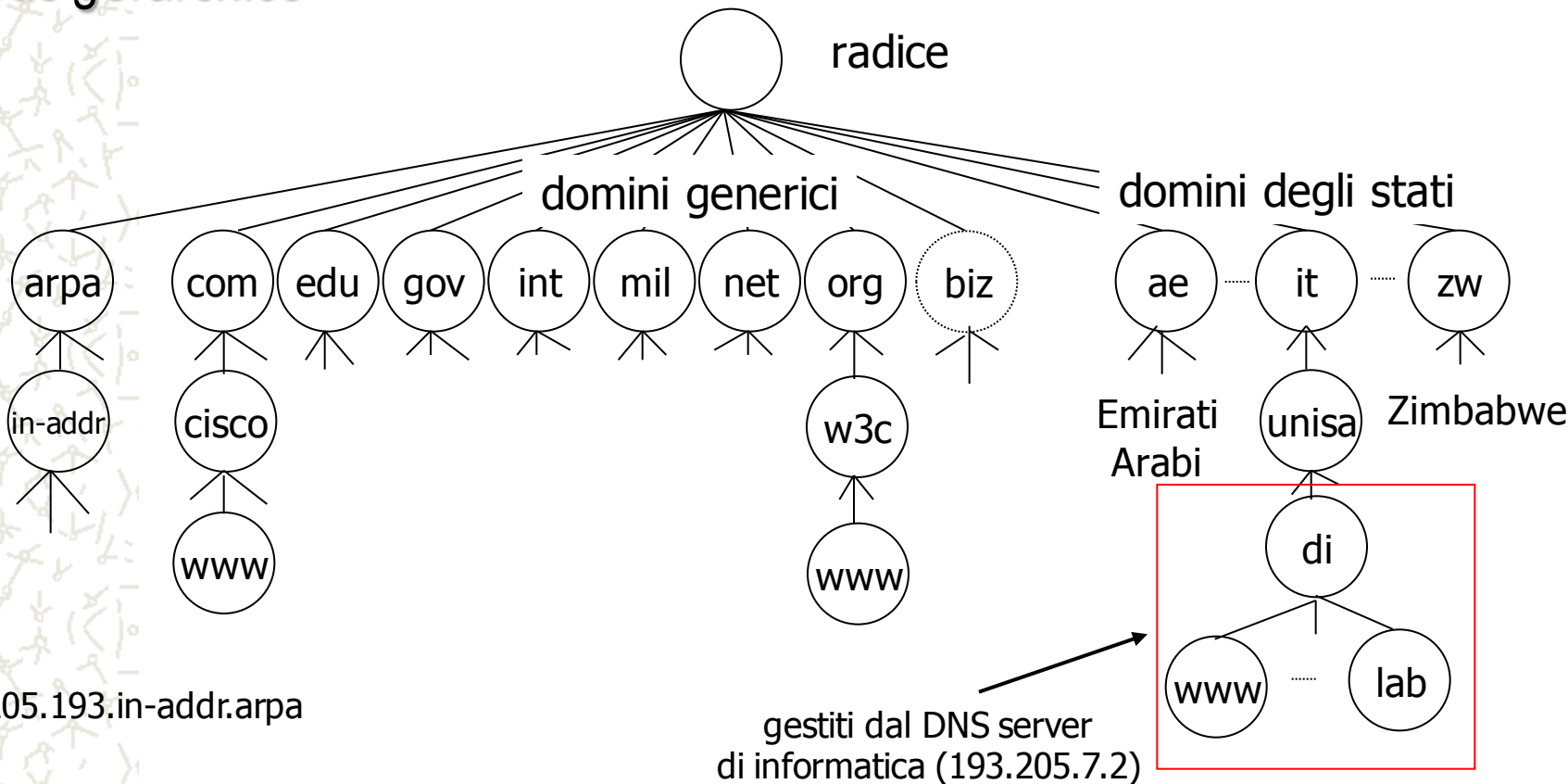
# gTLD

- edu: università ed enti di ricerca
- com: organizzazioni commerciali
- gov: enti governativi
- mil: enti militari
- net: organizzazioni di supporto e di gestione della rete
- org: enti privati non rientranti nelle precedenti categorie (es.:enti no-profit, associazioni non governative)

Altri nuovi TLD introdotti di recente:  
aero, biz, coop, info, museum, name,  
pro

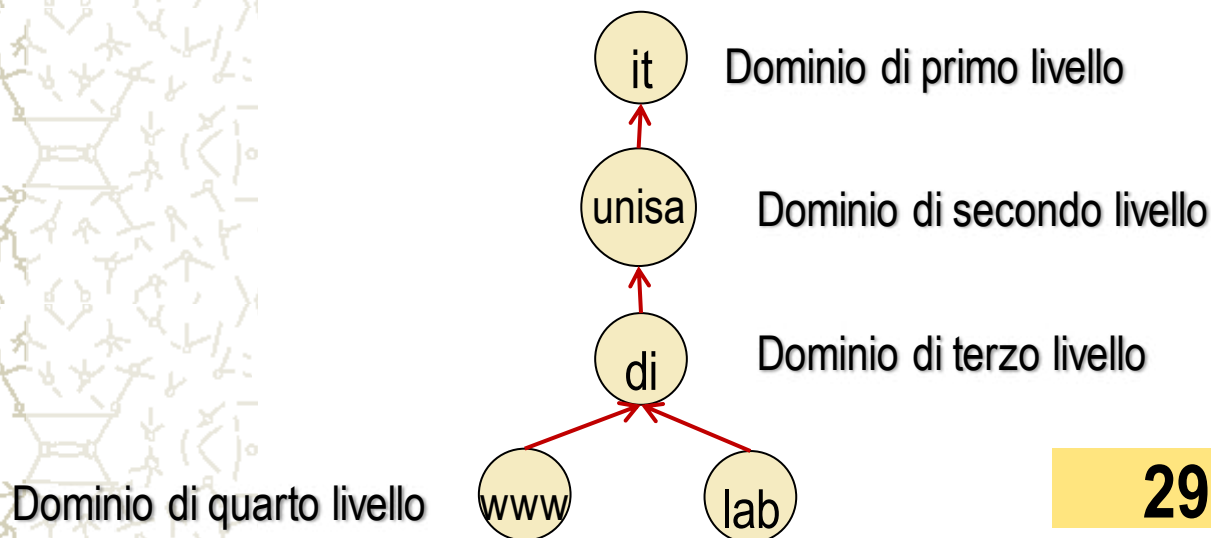
# Lo spazio dei nomi

I nomi dei domini Internet sono organizzati in domini e sottodomini in modo gerarchico



# Lo spazio dei nomi

- Il segmento più a destra rappresenta il dominio di primo livello che identifica la nazione di appartenenza dell'host cercato oppure la categoria cui appartiene la società proprietaria dell'host (edu,com,org..).
- Il secondo segmento indica un dominio di secondo livello che di solito individua una singola organizzazione (Università, Azienda, Ente).
- I segmenti successivi indicano in maniera sempre più specifica i sottodomini di terzo livello, quarto livello, fino ad individuare uno specifico host il cui nome è indicato all'estrema sinistra dell'indirizzo



# I nomi di dominio

- I nomi dei segmenti sono stringhe (case-insensitive) fino a 63 caratteri mentre il nome completo di un cammino non può superare 255 caratteri

FQDN (fully qualified domain name)

- FQDN indica un nome di dominio assoluto, ossia se il TDL A contiene un sottodominio B che contiene C, allora il nome completo è «C.B.A.» (termina con un punto)

PQDN (partially qualified domain name)

- IPQDN specifica solo parzialmente la locazione del dispositivo (non terminano con un punto). Es.: Se Z è un PQDN all'interno del contesto FQDN «Y.X.», allora l' FQDN di Z è «Z.Y.X.»

# DNS Server

Il server DNS è quella parte dell'applicazione che ha il compito di stabilire la corrispondenza tra un indirizzo logico e l'indirizzo IP di un host

## Tipi:

- **Server dei nomi locali:** Ciascun ISP (universitario, di un dipartimento accademico, di una società privata, etc.) ha un server dei nomi locale.
  - La richiesta inviata da un host viene prima mandata al server dei nomi locali dell'host.
  - Il server dei nomi locale è di solito “vicino” all'host (principio di località).
  - Il server dei nomi locale ha l'indirizzo del root server
- **Server dei nomi radice (root name server):** Quando il server dei nomi locali non può soddisfare immediatamente la richiesta di un host, si comporta come un client DNS ed invia la richiesta al root server

Esistono due casi:

# DNS Server

Tipi:

- .....
- **Server dei nomi radice (root name server):** Quando il server dei nomi locali non può soddisfare immediatamente la richiesta di un host, si comporta come un client DNS ed invia la richiesta al root server

Esistono due casi:

- Il root server ha la registrazione dell'hostname richiesto; la invia al server dei nomi locale che invia una risposta DNS all'host richiedente.
- Il root server non ha la registrazione, ma conosce l'indirizzo IP di un server dei nomi assoluto che contiene la correlazione di quel particolare hostname
- **Server dei nomi assoluto (authoritative name server):** Un server dei nomi si dice assoluto per un host se ha la corrispondenza tra il nome dell'host e IP

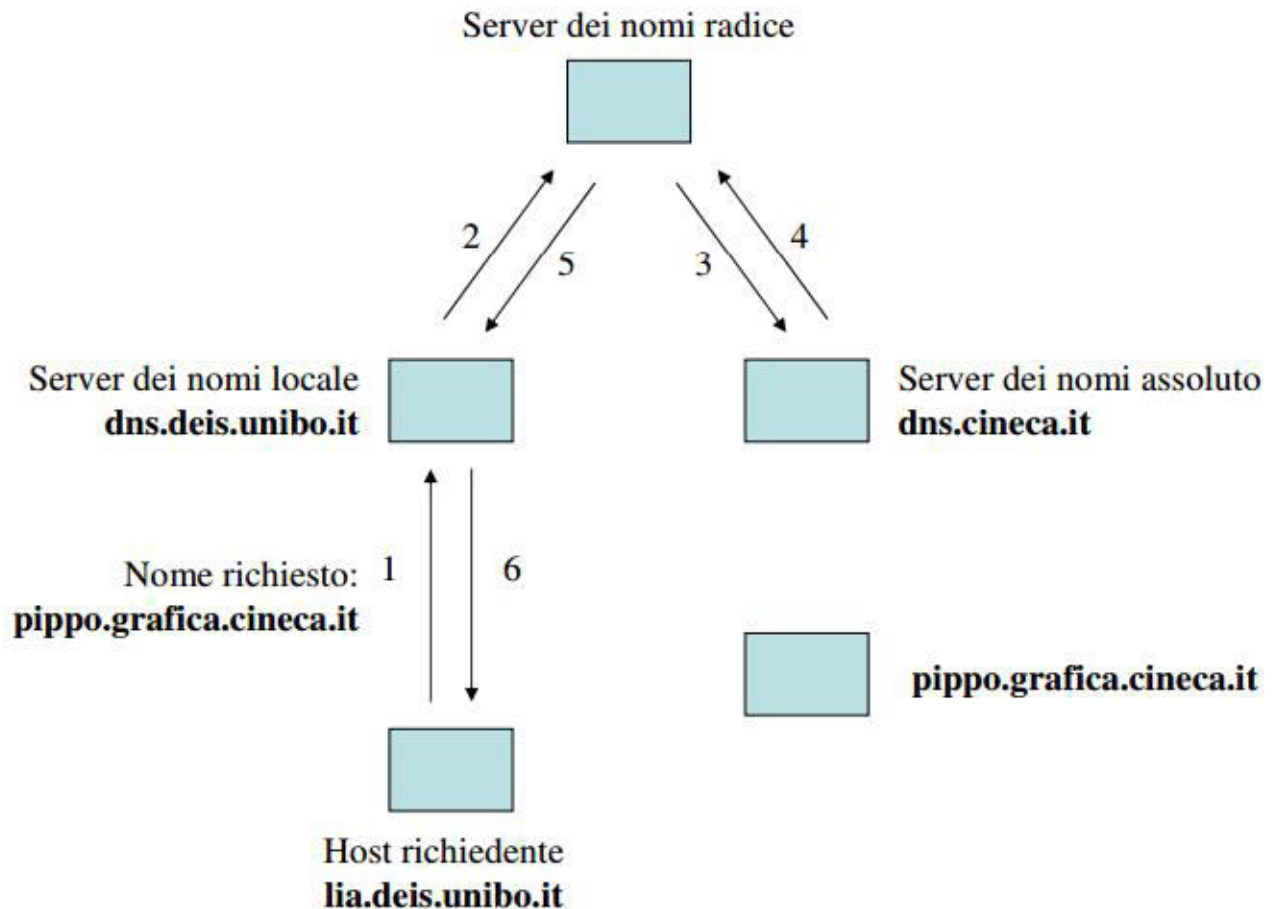


# Esempio

In questo esempio si suppone che il root server conosca l'indirizzo IP di un server di nomi assoluto per ciascun hostname.

Può essere che il root server conosca l'indirizzo IP di un server di nomi intermedio, che a sua volta conosce l'IP del server dei nomi assoluto per l'hostname.

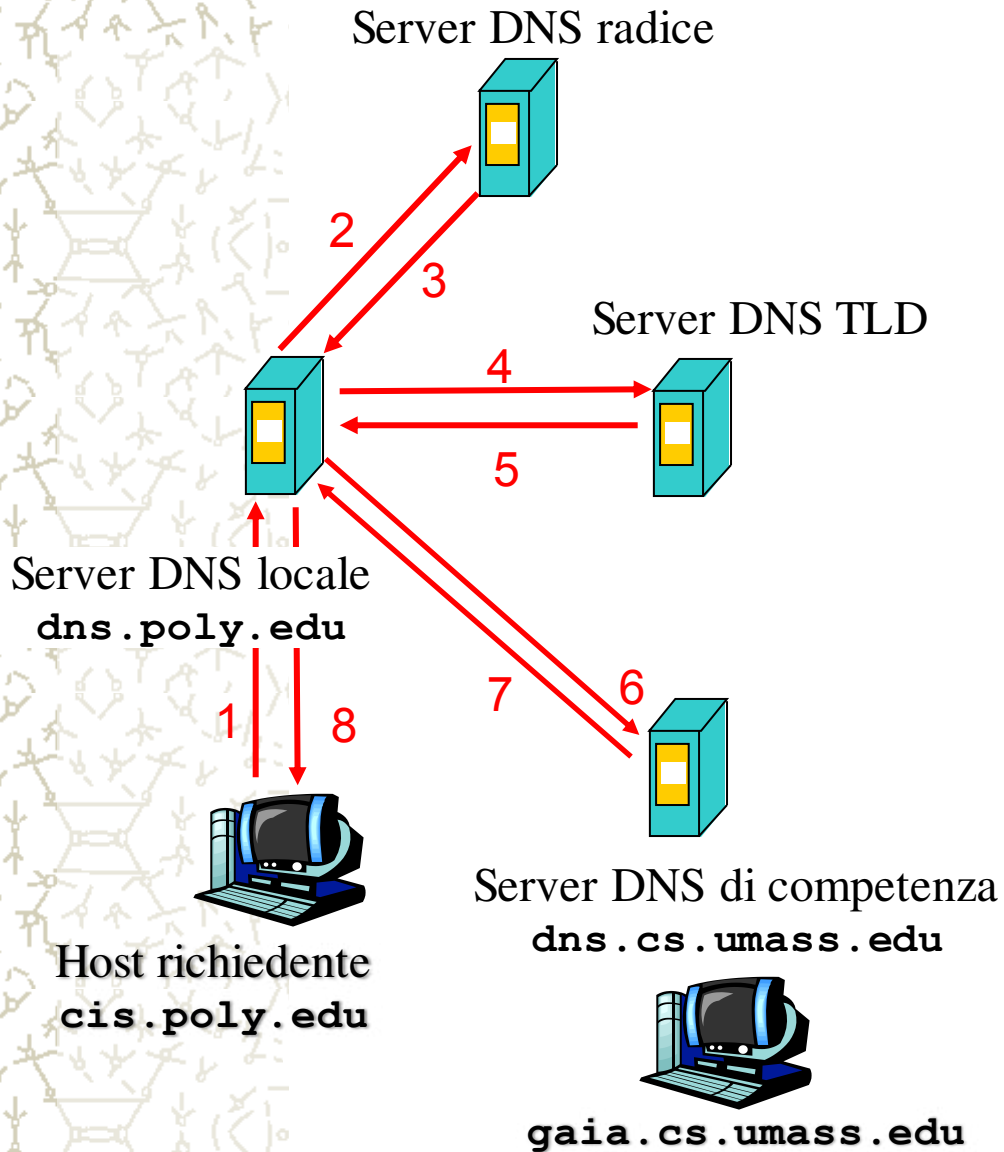
Ci possono essere anche due o più server intermedi nella catena tra il root server ed il server dei nomi radice.



# Server DNS locale

- Non appartiene strettamente alla gerarchia dei server
- Ciascun ISP (università, società, ISP residenziale) ha un server DNS locale.
  - detto anche “default name server”
- Quando un host effettua una richiesta DNS, la query viene inviata al suo server DNS locale
  - il server DNS locale opera da proxy e inoltra la query in una gerarchia di server DNS

# Esempio



L'host  
`cis.poly.edu`  
vuole l'indirizzo IP di  
`gaia.cs.umass.edu`

# Name servers: altre info

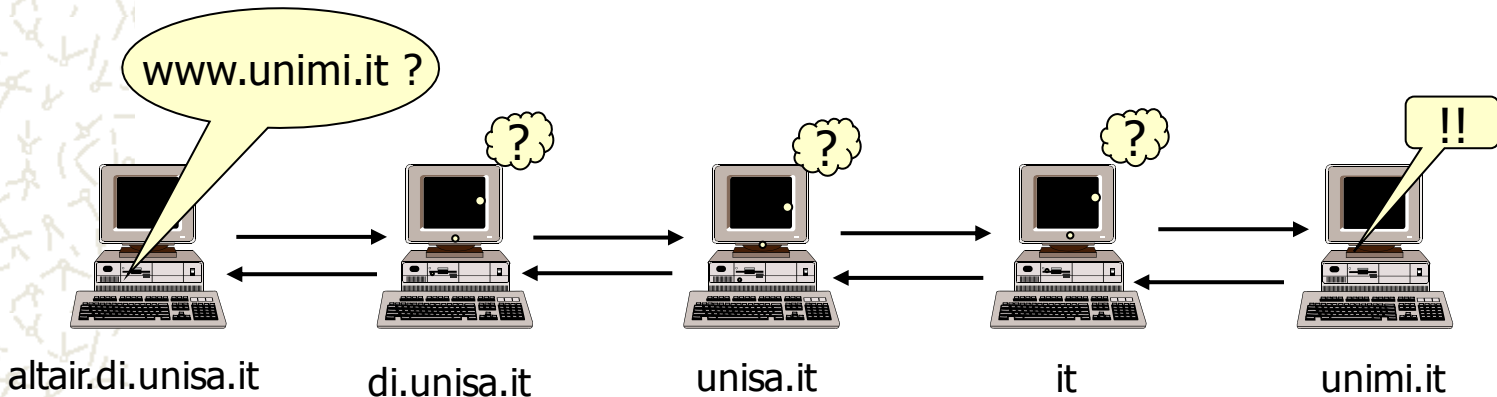
- Un Name Server ha autorità per una o più zone
- In una zona esiste un **name server primario** ma possono esistere anche **name servers secondari** (per ridondanza)
- I name server secondari ottengono le informazioni dal name server primario (zone transfer) interrogandolo periodicamente mentre i name server primari leggono le loro informazioni da un file
- Quando si aggiunge un nuovo host nella zona si aggiunge una riga alla tabella presente sul name server primario
- Il resolver chiede di risolvere il nome al name server locale

Il nome dell'host è **locale** -> viene restituito il record di autorità

Il nome è relativo ad un host di un' **altra zona** -> **interrogazione ricorsiva** (a meno che non sia nella cache)

# Interrogazione ricorsiva

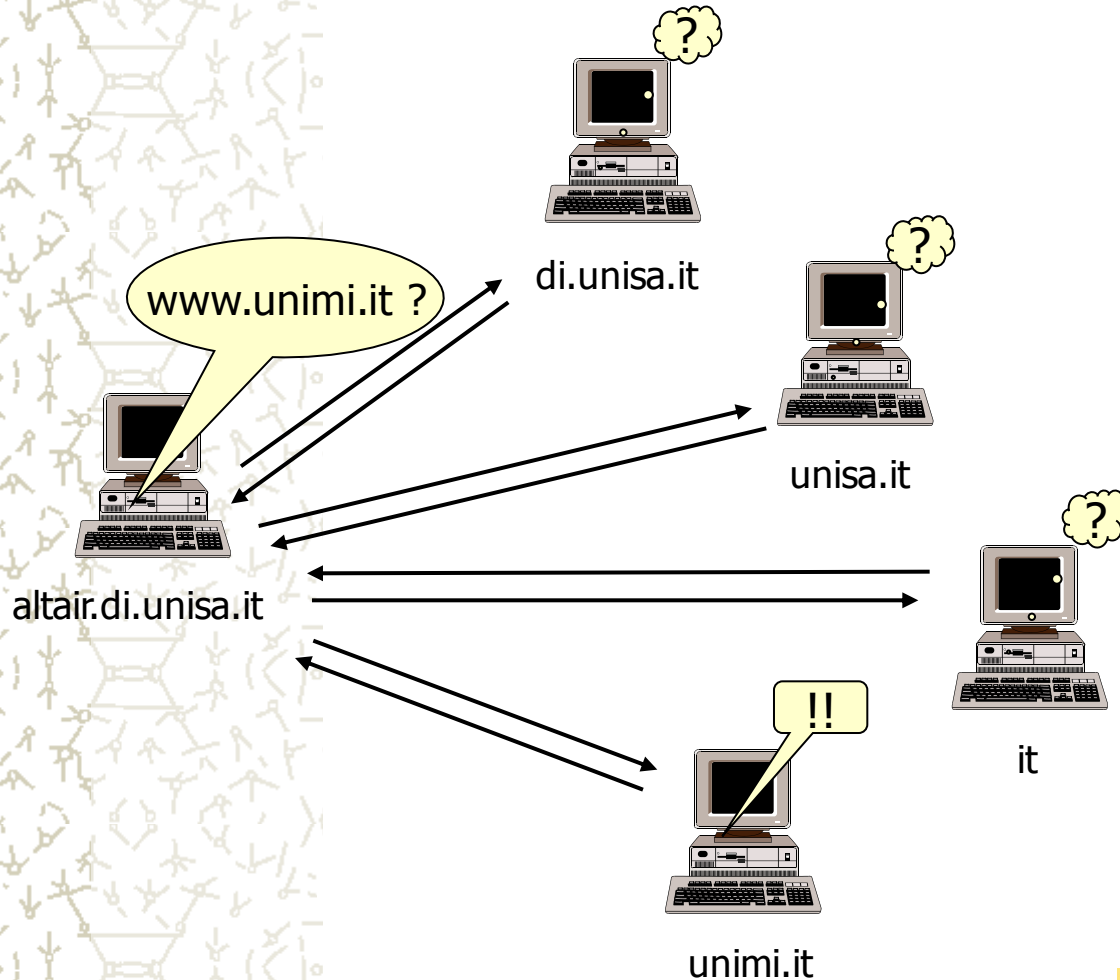
- Nella query ricorsiva il resolver propaga la richiesta al name server di livello superiore che gestisce la stessa negoziando con gli altri name server nella gerarchia.
- La risposta viene propagata all'indietro



- La richiesta non si propaga ai livelli superiori se un name server ha la risposta nella cache
- Le risposte nella cache non sono autoritative. Potrebbe esserci incoerenza con il vero valore
- I record sono inseriti nella cache con un time-to-live

# Interrogazione iterativa

- Nella query iterativa la negoziazione con i vari name server autoritativi per le zone interessate è gestita direttamente dal resolver



- Quando un host o un server dei nomi A fa una richiesta iterativa ad un server dei nomi B,
- se questo non ha la correlazione richiesta,
- esso invia immediatamente una risposta DNS ad A
- che indica l'indirizzo IP del server successivo nella catena, cioè del server dei nomi C.
- Il server A invia allora una richiesta a C.

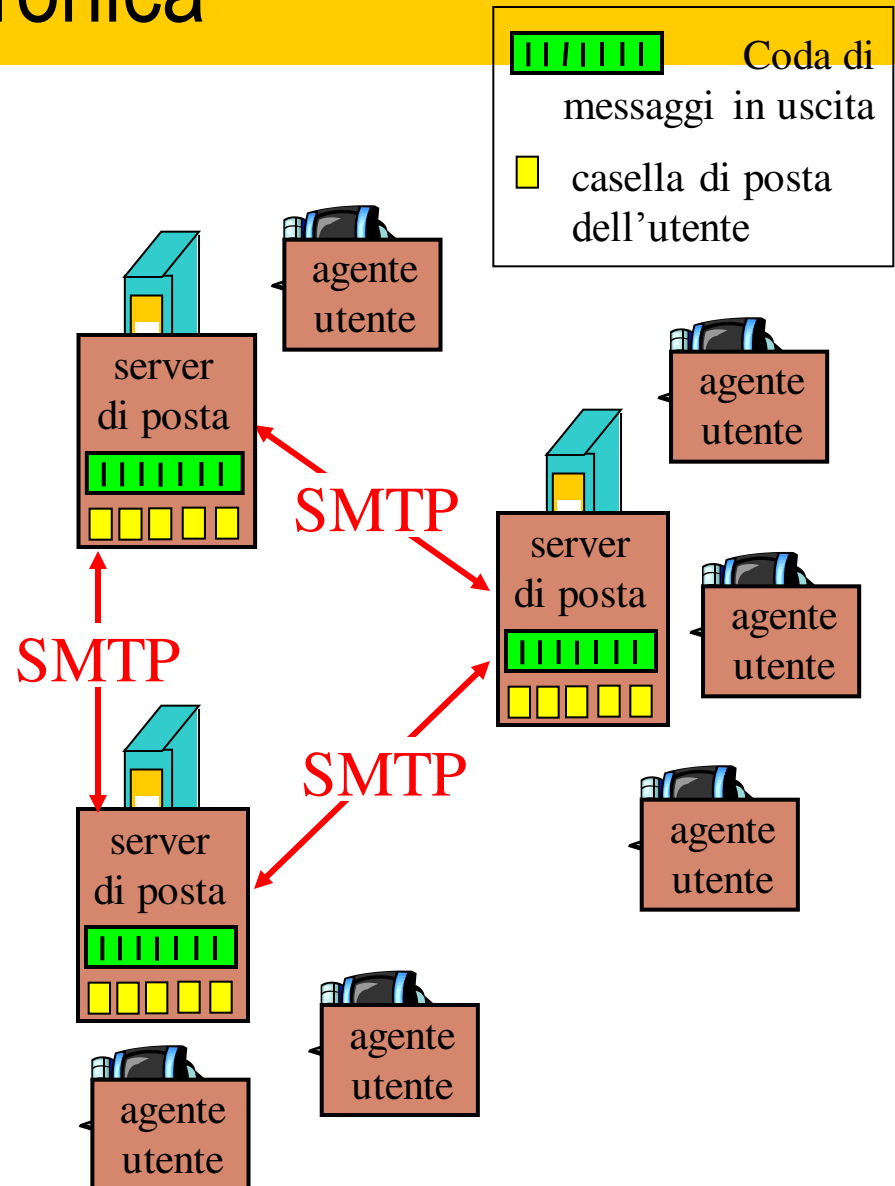
# Posta elettronica

## Tre componenti principali:

- agente utente
- server di posta
- simple mail transfer protocol: SMTP

### Agente utente

- detto anche "mail reader"
- programmi per leggere e gestire la posta e le mailboxes
- esempi: Eudora, Outlook, elm, Netscape Messenger
- Possono utilizzare protocolli per la gestione di mailboxes remote (pop3, imap).
- i messaggi in uscita o in arrivo sono memorizzati sul server





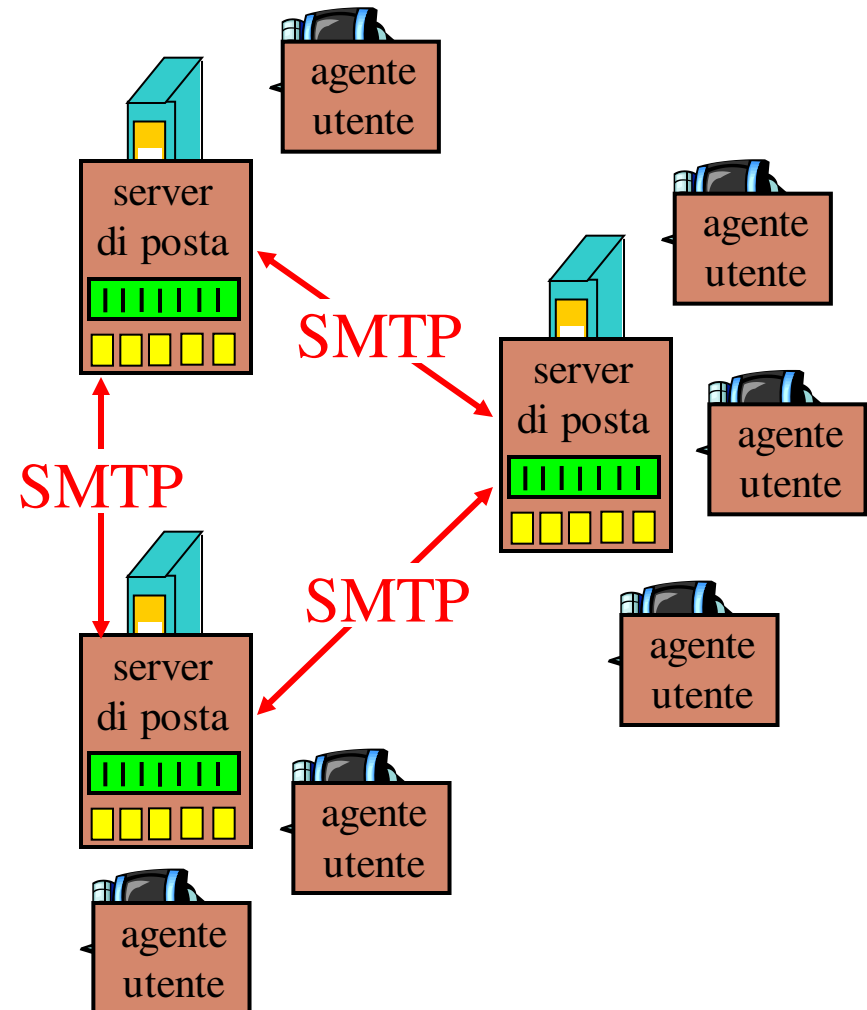
# Posta elettronica: server di posta

**Server di posta** include:

- **Casella di posta** (*mailbox*) contiene i messaggi in arrivo per l'utente
- **Coda di messaggi** da trasmettere

Usa il

- **Protocollo SMTP** tra server di posta per inviare messaggi di posta elettronica
  - client: server di posta trasmittente
  - “server”: server di posta ricevente





# Posta elettronica: SMTP [RFC 2821]

Inviare un messaggio corrisponde a trasferire un file

Per questo obiettivo esistono gli agenti di trasferimento di messaggi per

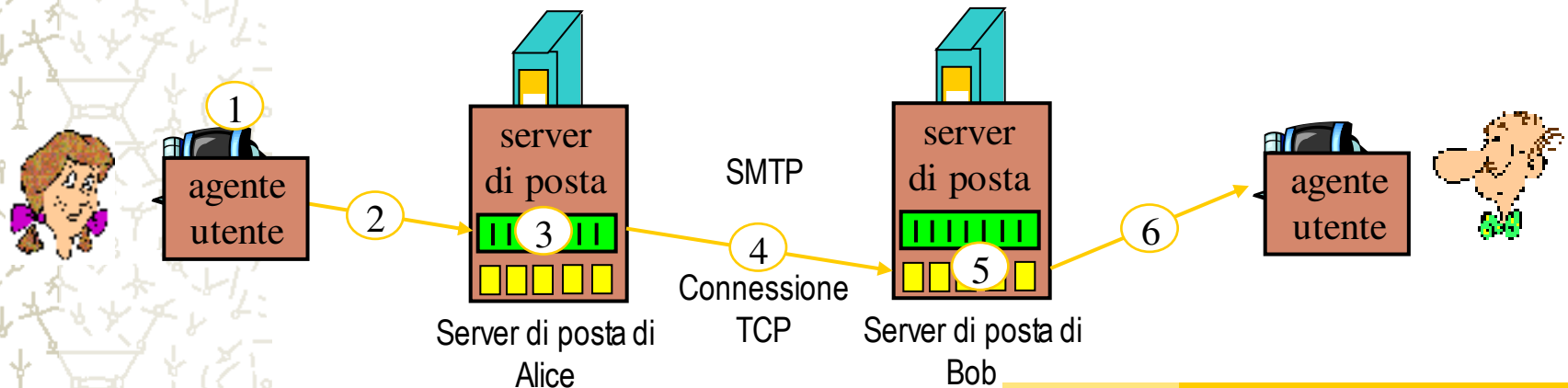
- gestire il trasferimento dei messaggi dalla sorgente alla destinazione e la ricezione dei messaggi sui server di posta
- Essi usano il protocollo SMTP (simple mail transfer protocol)

SMTP (simple mail transfer protocol) usa

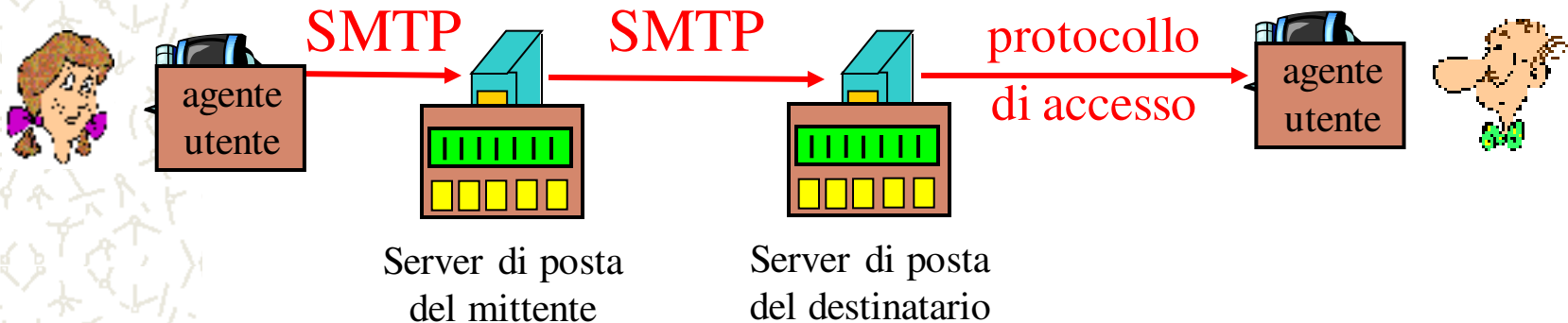
- TCP per trasferire in modo affidabile i messaggi di posta elettronica dal client al server (porta 25)
- tre espressioni per il trasferimento
  - handshaking (saluto)
  - trasferimento di messaggi
  - chiusura
- interazione comando/risposta
  - **comandi**: testo ASCII
  - **risposta**: codice di stato ed espressione
- i messaggi devono essere nel formato ASCII a 7 bit

# Scenario: Alice invia un messaggio a Bob

- 1) Alice usa il suo agente utente per comporre il messaggio da inviare "a" bob@some school.edu
- 2) L'agente utente di Alice invia un messaggio al server di posta di Alice; il messaggio è posto nella coda di messaggi
- 3) Il lato client di SMTP apre una connessione TCP con il server di posta di Bob
- 4) Il client SMTP invia il messaggio di Alice sulla connessione TCP
- 5) Il server di posta di Bob pone il messaggio nella casella di posta di Bob
- 6) Bob invoca il suo agente utente per leggere il messaggio



# Protocolli di accesso alla posta

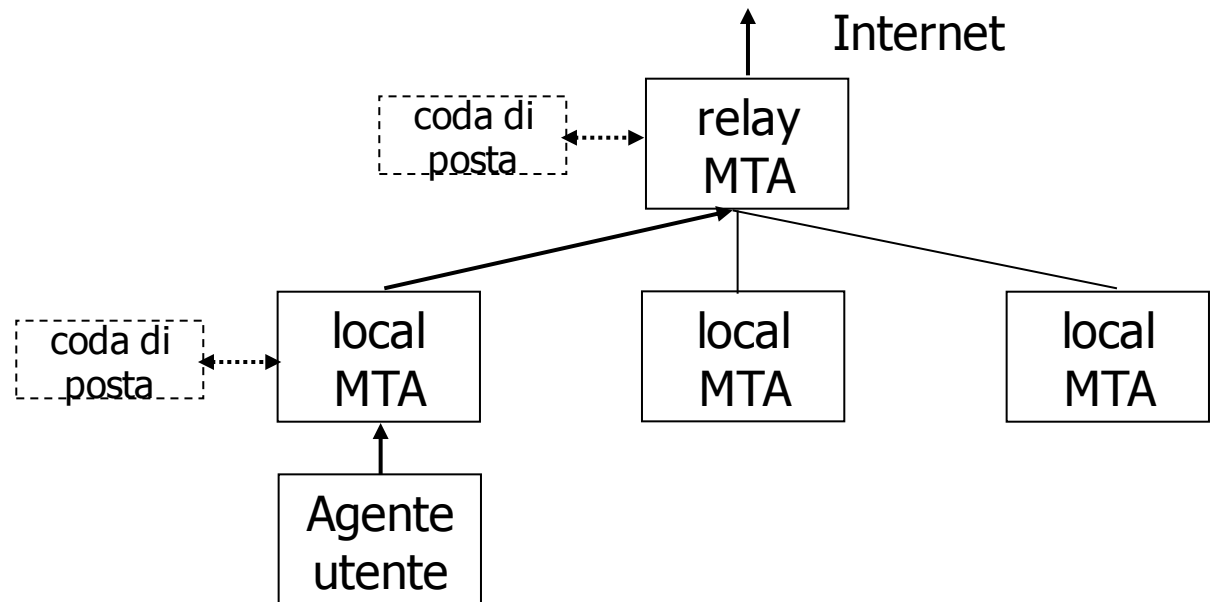


- SMTP: consegna/memorizzazione sul server del destinatario
- Protocollo di accesso alla posta: ottenere i messaggi dal server
  - POP: Post Office Protocol [RFC 1939]
    - autorizzazione (agente <--> server) e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - più funzioni (più complesse)
    - manipolazione di messaggi memorizzati sul server
  - HTTP: Hotmail , Yahoo! Mail, ecc.

# Relay Agents

- Server di riferimento per l'invio della posta
- Tutti i client inviano la posta al relay che la invia al destinatario
  - Semplifica la configurazione
  - Sono sempre connessi (possono ritentare in caso di insuccesso)

MTA (mail transfer agent)  
è un programma che si occupa della ricezione e smistamento da un computer all'altro di messaggi di posta elettronica.

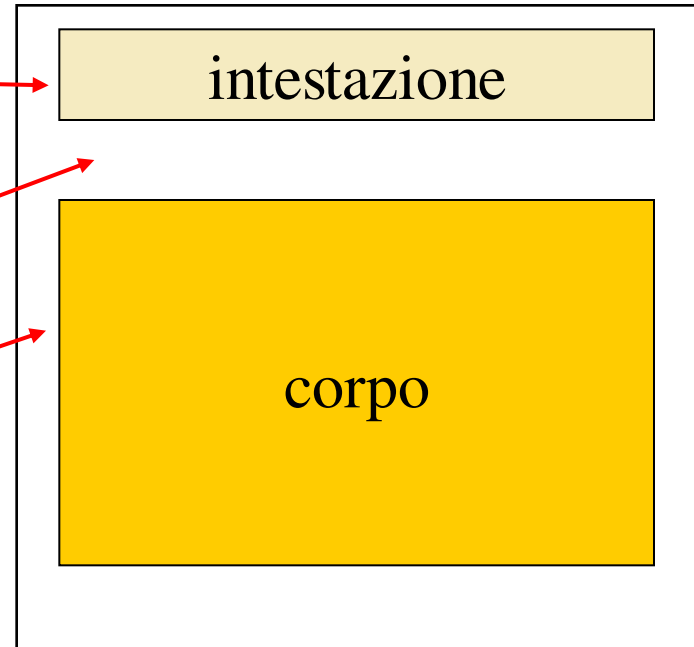


# Indirizzi di posta elettronica

- Hanno il formato `utente@host.dominio`
- Gli indirizzi sono risolti dal DNS che individua il server a cui inviare il messaggio (eventualmente usa una richiesta MX)
- Il server di posta riceve i messaggi e li accoda nella mailbox dell'utente
- La mailbox è un file di testo in una directory specifica nel server (es. in Unix è normalmente `/var/spool/mail/utente`)
- L'utente può accedere alla posta localmente leggendo il file mailbox. Il client per la lettura della posta provvede a individuare i singoli messaggi nella mailbox

# Formato dei messaggi

- E' definito in RFC 821 - RFC 822
- **Righe di intestazione**, per esempio
  - To:
  - From:
  - Subject:
- **Riga vuota**
- **Corpo**
  - il “messaggio”, soltanto caratteri ASCII



I campi di intestazione consistono in linee di testo ASCII contenenti il nome del campo seguito dal carattere `:` e poi da un valore

L'agente utente costruisce il messaggio e lo passa all'agente di trasferimento che utilizza alcuni campi dell'intestazione per l'invio

# Esempio di intestazione

Return-Path: <BWerner@computer.org>  
Delivered-To: fp@di.unisa.it  
Received: from firewall.di.unisa.it (firewall.di.unisa.it [10.0.0.1])  
by alpha.di.unisa.it (Postfix) with ESMTP id B763218337  
for <fp@di.unisa.it>; Sat, 19 May 2001 03:02:57 +0200 (CEST)  
Received: from sendmailout.computer.org (unknown [206.99.234.2])  
by firewall.di.unisa.it (Postfix) with ESMTP id 6A9053C0BC  
for <fp@di.unisa.it>; Sat, 19 May 2001 02:48:10 +0200 (CEST)  
Received: from cray.computer.org ([63.84.223.121])  
by sendmailout.computer.org (Build 101 8.9.3/NT-8.9.3) with ESMTP  
id UAA01113; Fri, 18 May 2001 20:03:57 -0400  
Subject: ICDAR'01: Your Author Kit  
To: ld47@csgz.edu.cn, fp@di.unisa.it  
Cc: Karl.Tombre@loria.fr, haralick@gc.cuny.edu  
X-Mailer: Lotus Notes Release 5.0.4 June 8, 2000  
Message-ID: <OF1A96792E.5B9B2F1A-ON88256A50.00814155@computer.org>  
From: BWerner@computer.org  
Date: Fri, 18 May 2001 17:03:57 -0700  
X-MIMETrack: Serialize by Router on Cray/IEEE Computer Society(Release  
5.0.6a |January 17, 2001) at 05/18/2001 05:11:59 PM  
MIME-Version: 1.0  
Content-type: multipart/mixed;  
Boundary="0\_\_=88256A50008141558f9e8a93df938690918c88256A5000814155"  
Content-Disposition: inline

# MIME

- **Multipurpose Internet Mail Extensions** (RFC 1521)
- Aggiunge dei campi di intestazione per definire la struttura del corpo del messaggio (è gestita dall'agente utente)

**Mime-Version:**

**Content-Type:**

**Content-Transfer-Encoding:**

**Content-ID:**

**Content-Description:**

I tipi di contenuto sono organizzati in categorie predefinite  
es. text/plain, text/html, image/gif, multipart/mixed,  
application/octet-stream, .....



# Formato del messaggio: estensioni di messaggi multimediali

- MIME: estensioni di messaggi di posta multimediali, RFC 2045, 2056
- Alcune righe aggiuntive nell'intestazione dei messaggi dichiarano il tipo di contenuto MIME

Versione MIME

metodo usato  
per codificare i dati

Tipo di dati  
multimediali, sottotipo,  
dichiarazione  
dei parametri

Dati codificati

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
```

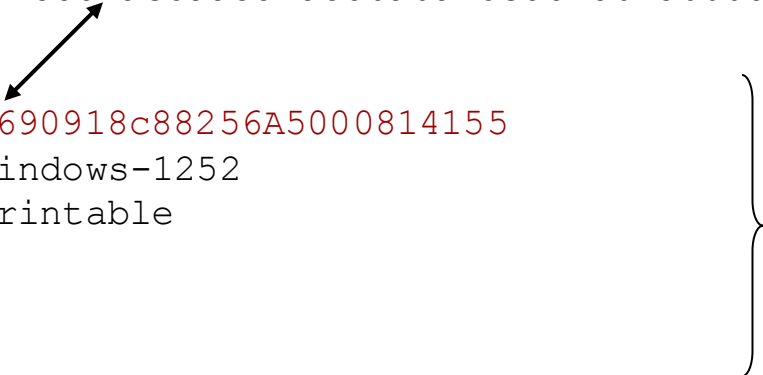
# Esempio MIME

```
MIME-Version: 1.0
Content-type: multipart/mixed;
    Boundary="0__=88256A50008141558f9e8a93df938690918c88256A5000814155"
Content-Disposition: inline

--0__=88256A50008141558f9e8a93df938690918c88256A5000814155
Content-type: text/plain; charset=Windows-1252
Content-transfer-encoding: quoted-printable
Dear Author:
Congratulations! We have been notified that your paper has been accepted
d

--0__=88256A50008141558f9e8a93df938690918c88256A5000814155
Content-type: application/pdf;
    name="=?Windows-1252?Q?icdar01=5Fcopyright=5Fform.pdf?="
Content-Disposition: attachment; filename="=?Windows-
1252?Q?icdar01=5Fcopyright=5Fform.pdf?="
Content-transfer-encoding: base64

JVBERi0xLjIgDSXi48/TDQogDTEwIDAgb2JqDTw8DS9MZW5ndGggMTEgMCBSDS9GaWx0ZXIgL0Zs
YXRlRGVjb2RlIA0+Pg1zdHJlYW0NCkiJrFfZctvIFf0C/UPXvESuUBAaO+YplEx5WDNjuSi6nFTp
```



Parte 1

# SMTP

- Il protocollo SMTP definisce la sequenza di comandi (inviati in ASCII) necessaria per il trasferimento dei messaggi
- Insieme minimale di comandi usati dal client

**HELO** <host>

“Saluta” il server

**MAIL From:** <indirizzo>

indica il mittente del messaggio

**RCPT To:** <indirizzo>

Indica il destinatario (recipient)

} “busta”

**DATA**

Invio corpo del messaggio terminato da un . su una linea

**QUIT**

Chiude la connessione

# Esempio SMTP

```
fp@ing.unina.it... Connecting to alpha.di.unisa.it. via relay...
220 alpha.di.unisa.it ESMTP Postfix (Postfix-19991231) (Linux-
Mandrake)
>>> EHLO ultra3.di.unisa.it
250-alpha.di.unisa.it
250-PIPELINING
250-SIZE 10240000
250-ETRN
250 8BITMIME
>>> MAIL From:<fp@ultra3.di.unisa.it> SIZE=20
250 Ok
>>> RCPT To:<fp@ing.unina.it>
250 Ok
>>> DATA
354 End data with <CR><LF>.<CR><LF>
>>> .
250 Ok: queued as BE37A18337
fp@ing.unina.it... Sent (Ok: queued as BE37A18337)
Closing connection to alpha.di.unisa.it.
>>> QUIT
221 Bye
```



Extended SMTP

# II WEB

- Nato nel 1989 al CERN di Ginevra come mezzo per scambiare informazioni
- **w3c - Consorzio World Wide Web** (1994)
  - Standardizzazione dei protocolli su Web
  - Interoperabilità fra i siti
  - <http://www.w3c.org>
- ⌘ Il **WEB** è una collezione di documenti ipertestuali distribuita su server collegati alla rete Internet
- ⌘ La ragnatela (web) è un grafo i cui nodi sono i singoli documenti collegati fra loro da puntatori (**hyperlink**)
- ⌘ La maggior parte dei documenti su web è in formato **HTML** (**HyperText Markup Language**) ma si trovano anche altri formati (Macromedia Flash, pdf, postscript, ecc..)

# World Wide Web (1991): WWW

- Nel 1991 presso il CERN di Ginevra il ricercatore Tim Berners-Lee definì il protocollo **HTTP** (*HyperText Transfer Protocol*).
- Un sistema che permette una lettura ipertestuale, *non-sequenziale* dei documenti.
- La lettura si effettua saltando da un punto all'altro mediante l'utilizzo di rimandi (link o, più propriamente, *hyperlink*).
- L'applicazione del protocollo HTTP sulla rete Internet ha visto nascere il **World Wide Web:**

***“Grande Ragnatela Mondiale”***

Il servizio WWW permette di navigare ed usufruire di un insieme vastissimo di contenuti (multimediali e non) e di ulteriori **servizi** accessibili a tutti.

# Le pagine WEB e i siti

- Una **pagina web** è il modo in cui vengono rese disponibili all'utente finale le informazioni del World Wide Web della rete Internet.
- Un insieme di pagine web, tra loro relazionate secondo una gerarchica e una struttura ipertestuale costituiscono un **sito web**.
- Una pagina web ha uno o più **URL** (*Universal Resource Locator*), un **link** permanente che ne permette l'individuazione.
- Le pagine web vengono lette dai **Browser**.

# Categorie siti WEB

I siti web vengono divisi in categorie per inquadrarne il settore di operatività o i servizi offerti:

- **sito personale:** contiene informazioni prevalentemente autobiografiche o focalizzate sui propri interessi personali.
- **sito aziendale:** promuove un'azienda o un servizio.
- **sito di commercio elettronico (o "e-commerce"):** specializzato nella vendita di beni e/o servizi via internet.
- **forum:** luogo in cui discutere tramite la pubblicazione e la lettura di messaggi, organizzati per discussioni (*thread*) e messaggi (*post*).
- **blog:** i contenuti vengono visualizzati in forma cronologica.
- **Social network:** servizio di rete basato su relazioni sociali.
- **motore di ricerca:** registra i contenuti degli altri siti e li rende disponibili per la ricerca.



# WWW: I Browser

- **I Browser** permettendo di visualizzare i contenuti delle pagine dei siti web e di interagire con essi.
- **I Browser** sono in grado di interpretare l'HTML:
  - HTML è il codice con il quale sono scritte la maggior parte delle pagine web.
  - I Browser visualizzano la pagina in forma di ipertesto.
- Alcuni browser:
  - *Internet Explorer*: sviluppato da Microsoft e incluso in Windows a partire dal 1995.
  - *Firefox*: open source prodotto da Mozilla Foundation.
  - *Safari*: sviluppato da Apple Inc. per il sistema operativo Mac OS X, iOS e successivamente anche per Windows.
  - *Opera*: prodotto da Opera Software, disponibile per i sistemi operativi Windows, Macintosh, Linux e molti altri.
  - *Google Chrome*: sviluppato da Google. La ricetta di Google Chrome è: velocità, sicurezza e manualità. Attualmente il più diffuso.

# URL: Uniform Resource Locator (1)

L'*URL* è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet.

*Esempio*: un documento, un'immagine, un video, diventa accessibile ad un *client* che ne fa richiesta attraverso l'utilizzo di un *web browser*

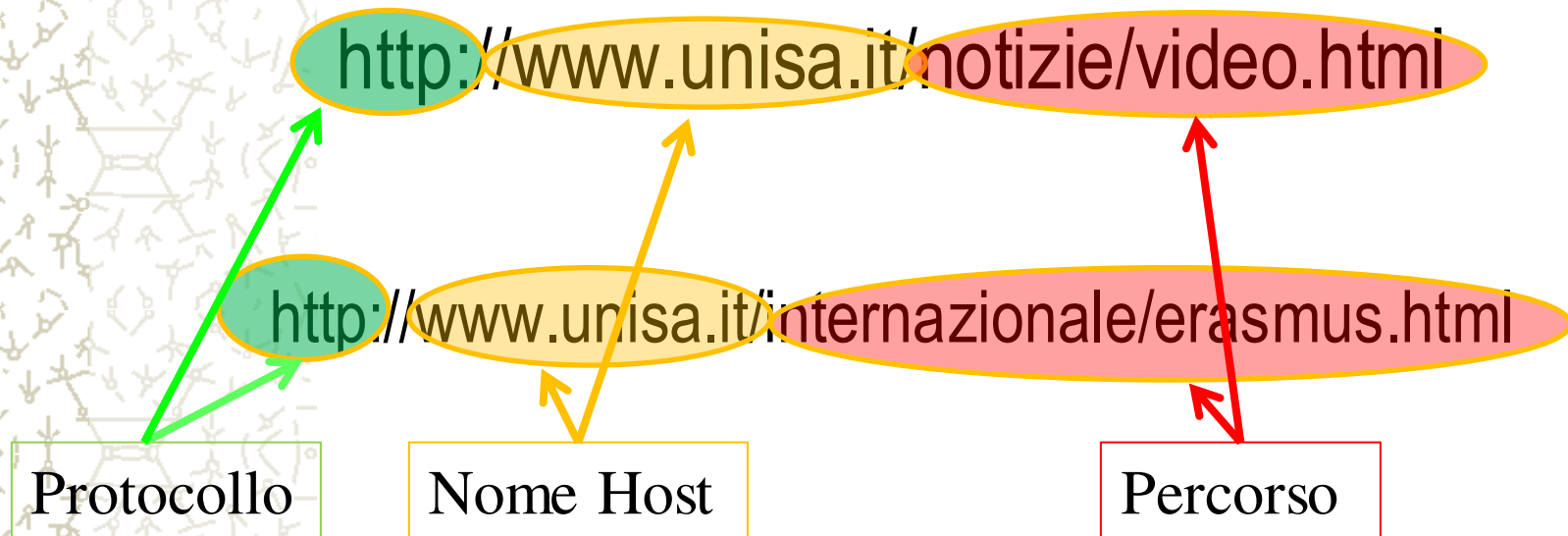
- [http://it.wikipedia.org/wiki/Uniform\\_Resource\\_Locator](http://it.wikipedia.org/wiki/Uniform_Resource_Locator)
- <http://it.wikipedia.org/wiki/Informatica>
- <http://www.borsainside.com/finanzainside/pil.shtm>

# URL: Uniform Resource Locator (2)

Ogni URL si compone normalmente di sei parti, alcune delle quali opzionali:

1. **Protocollo**: Descrive il protocollo di comunicazione da utilizzare per accedere al server. I più comuni sono: http, https, ftp.
2. **username:password@** (opzionale): è possibile specificare l'autenticazione per l'accesso alla risorsa. Attenzione è in chiaro.
3. **nomehost**: Rappresenta l'indirizzo fisico del server su cui risiede la risorsa.
4. **porta** (opzionale): Vi è necessità di indicare questo parametro quando il processo server è in ascolto su una porta non conforme allo standard.
5. **percorso** (opzionale): Percorso (pathname) nel file system del server che identifica la risorsa (generalmente una pagina web, una immagine o un file multimediale).
6. **querystring** (opzionale): al termine dell'url è possibile aggiungere una query string separandola utilizzando il simbolo "?". La query string è una stringa di caratteri che consente di passare al server uno o più parametri.

# URL: Uniform Resource Locator (3)



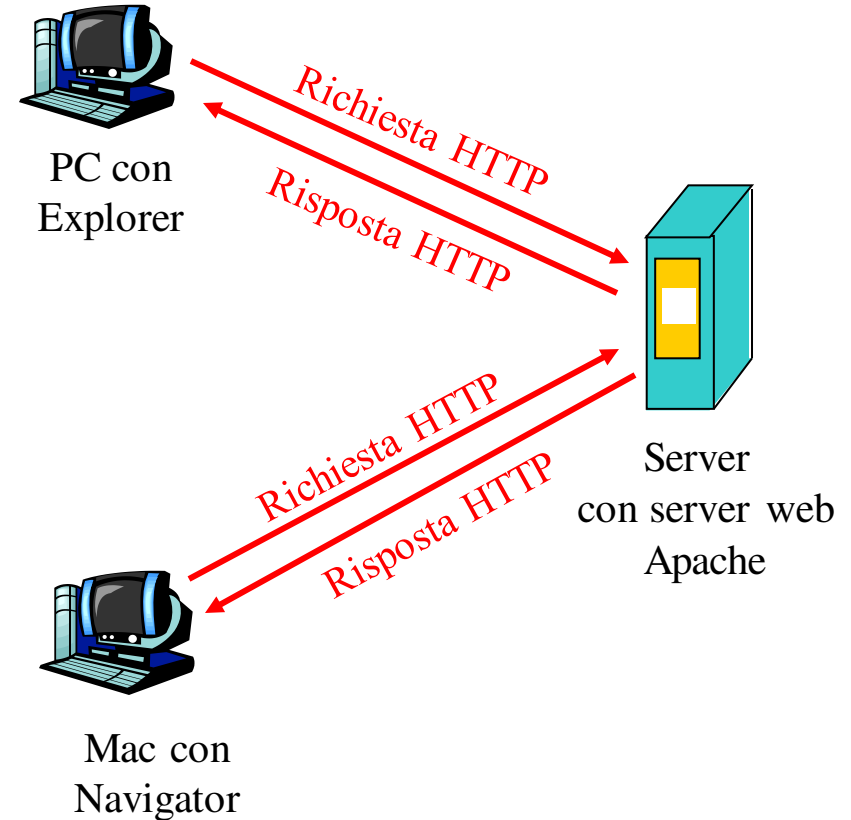
`http://www.unisa.it/studenti/avvisi/viscom.php?id=318`

**Query String**

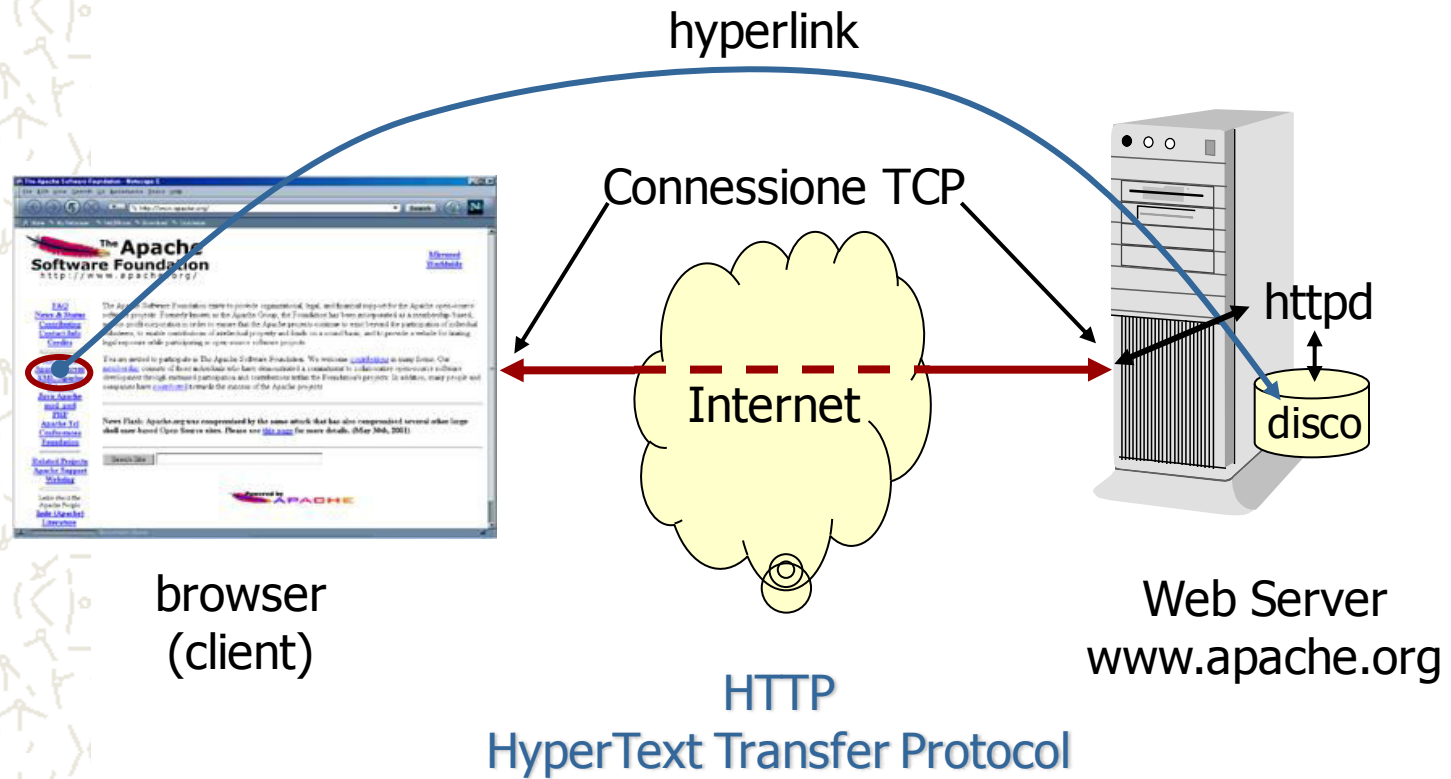
# HTTP

## HTTP: hypertext transfer protocol

- Protocollo a livello di applicazione del Web
- Modello client/server
  - *client*: il browser che richiede, riceve, “visualizza” gli oggetti del Web
  - *server*: il server web invia oggetti in risposta a una richiesta
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



# Web servers



# HTTP

- Ogni sito Web ha un processo server in ascolto su una porta TCP
- La porta standard è la numero 80
- HTTP definisce il protocollo di comunicazione fra il client e il server
- E' utilizzato per trasferire ogni tipo di risorsa (file) su WWW
  - Una risorsa è un'entità individuata da un URL (Uniform Resource Locator)
  - file di qualsiasi formato
- Utilizza un modello client-server
  - Il client HTTP apre la connessione e invia un messaggio di richiesta al server HTTP
  - Il server invia una risposta che in genere contiene la risorsa richiesta e poi chiude la connessione
- Il protocollo è senza stato (non c'è memoria delle transazioni)

# Formato dei messaggi HTTP

- I messaggi di richiesta e risposta hanno formato simile

Linea Iniziale (diversa per richiesta e risposta)

Header1: valore1

Header2: valore2

Header3: valore3

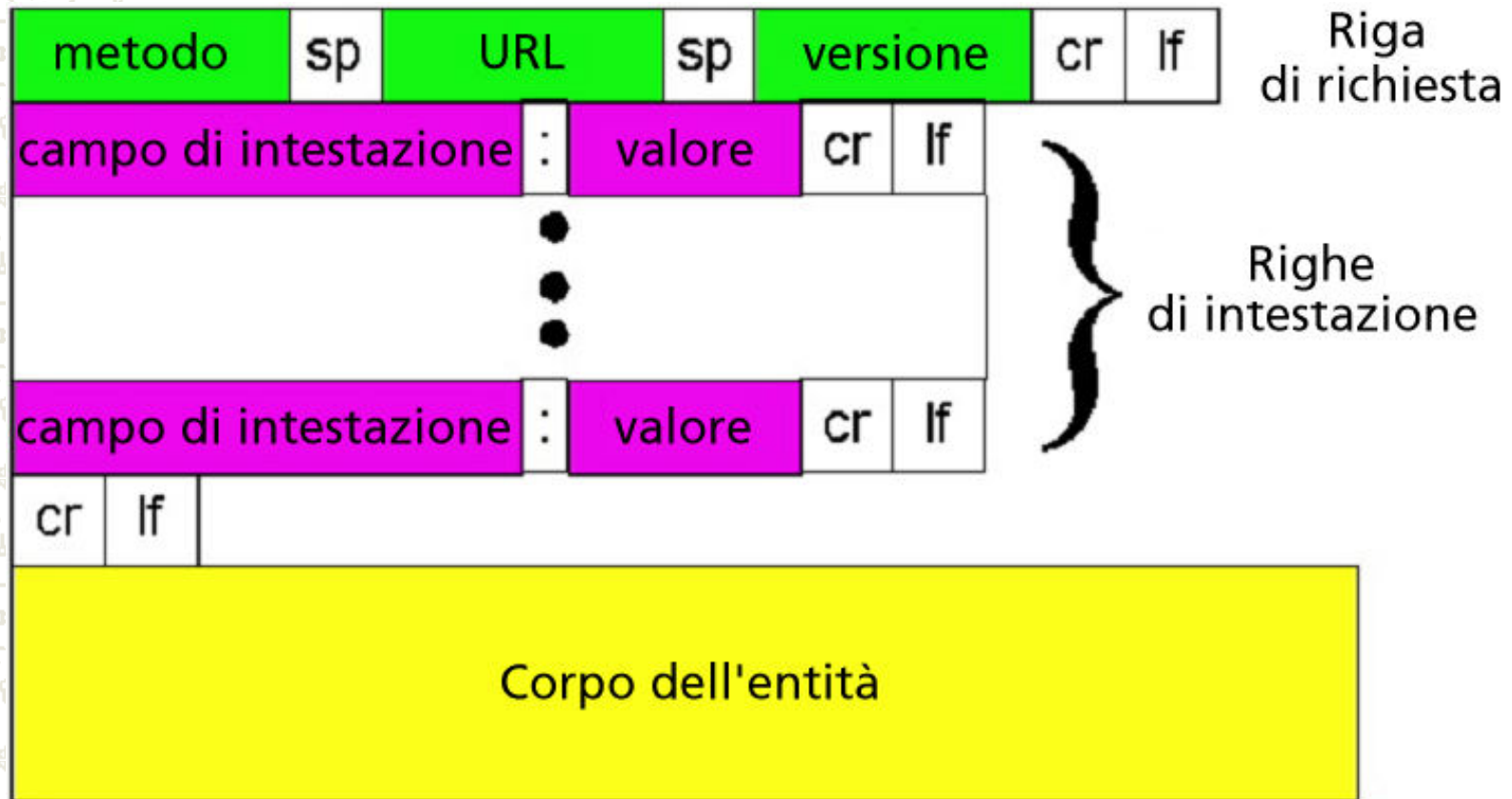
Linea vuota!

Corpo del messaggio (contenuto del file, risultato di una interrogazione; può essere lungo molte linee e essere binario)

⌘ L'intestazione è separata dal corpo da una linea vuota (CR LF)



# Messaggio di richiesta HTTP: formato generale



# Richieste HTTP

- La linea iniziale di una richiesta è formata da 3 parti
  - Metodo
  - Percorso locale della risorsa richiesta
  - Versione di HTTP usata

## ⌘ Esempio di richiesta

```
GET /index.html HTTP/1.1
```

← Linea Iniziale

```
Host: univac.di.unisa.it
```

```
User-Agent: Mozilla/5.0 (Windows; U; Win98; en-US; m18)  
Gecko/20010131 Netscape6/6.01
```

```
Accept: */*
```

```
Accept-Language: en
```

```
Accept-Encoding: gzip, deflate, compress, identity
```

```
Keep-Alive: 300
```

```
Connection: keep-alive
```

header

# Metodi HTTP

- I metodi definiscono le operazioni possibili su una risorsa

## ○ GET

chiede il trasferimento di una risorsa. Se è seguita dall'intestazione *If-Modified-Since* il server invia i dati solo se sono stati modificati dopo la data specificata (gestione cache del browser)

## ○ HEAD

Richiede solo le intestazioni relative alla risorsa. Serve per verificare le caratteristiche della risorsa senza trasferirla

## ○ POST

Utilizzato per inviare dati da elaborare al server. L'intestazione è seguita da un corpo della richiesta che contiene i dati. Il tipo e la dimensione dei dati è indicata dagli header MIME **Content-Type:** e **Content-Length:**

# Risposta HTTP

- La linea iniziale di una risposta costituisce una linea di stato

HTTP/1.1 200 OK

headers

Date: Wed, 06 Jun 2001 22:44:40 GMT  
Server: Apache/1.3.20 (Win32)  
Last-Modified: Wed, 06 Jun 2001 22:32:26 GMT  
ETag: "0-64-3b1eaf7a"  
Accept-Ranges: bytes  
Content-Length: 100  
Connection: close  
Content-Type: text/html

← Linea vuota

risorsa

<HTML>  
<HEAD>  
<TITLE>Prova</TITLE>  
</HEAD>  
<BODY>  
<H2>File HTML di Prova</H1>  
</BODY>

# Codici di stato

- La linea di stato riporta un codice di stato e la sua “spiegazione”

HTTP/1.1 200 OK

⌘ I codici più comuni sono:

**200 OK**

Richiesta con successo. La risorsa è nel corpo della risposta

**404 Not Found**

La risorsa richiesta non esiste

**301 Moved Permanently**

**302 Moved Temporarily**

**303 See Other** (in HTTP 1.1)

La risorsa è stata spostata ad un altro URL specificato nel campo **Location:** dell'intestazione. Il client dovrebbe saltare a tale locazione (redirect).

**500 Server Error**

# Intestazioni

- Seguono il formato specificato in RFC 822 anche per l'email
- HTTP 1.0 definisce 16 header (tutti opzionali)
- HTTP 1.1 definisce 46 header (obbligatorio [Host:](#))
- Alcuni esempi....

**User-Agent:**

Identifica il programma client che effettua la richiesta. Individua webots, spiders, ecc..

**Server:**

Identifica il server

**Last-Modified:**

Indica la data di modifica della risorsa. E' usata per gestire le cache

**Content-Type:**

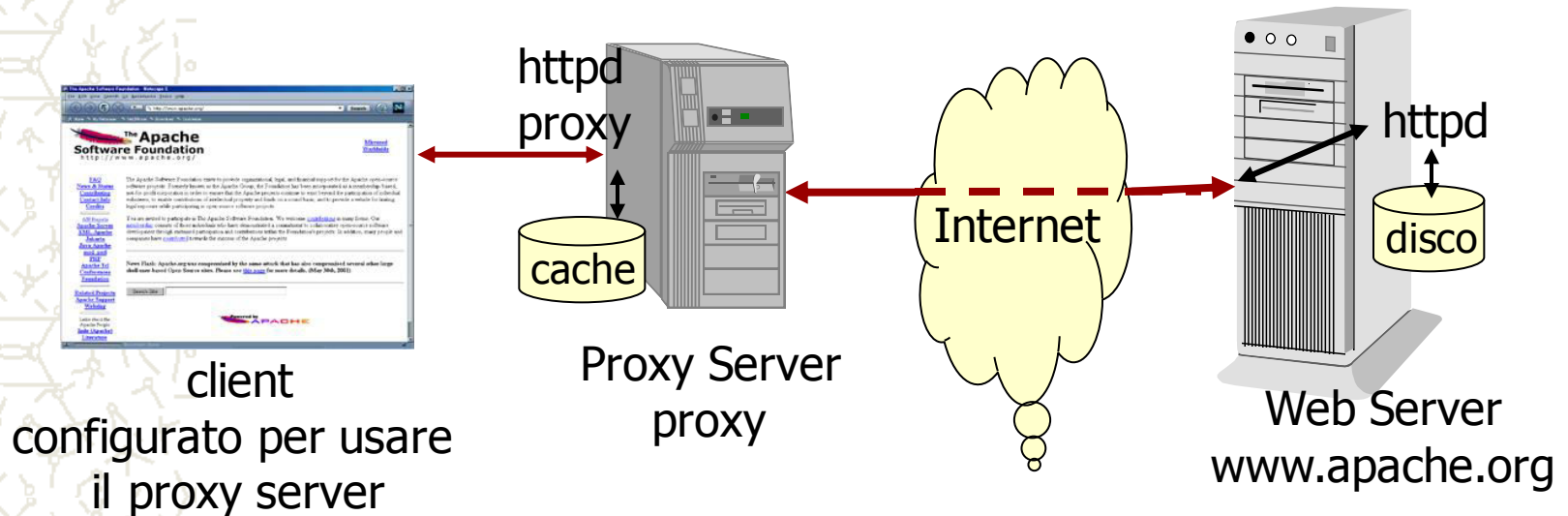
Tipo MIME del corpo del messaggio

**Content-Length:**

Lunghezza in byte del corpo del messaggio

# Proxy HTTP

- Un proxy HTTP agisce da intermediario fra il client e il server
  - ❑ Riceve le richieste dal client
  - ❑ Propaga la richiesta al server corretto
- Sono usati su LAN per caching o accesso ad Internet tramite firewall



# Connessioni HTTP

## Connessioni non persistenti

- Almeno un oggetto viene trasmesso su una connessione TCP
- HTTP/1.0 usa connessioni non persistenti

## Connessioni persistenti

- Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server
- HTTP/1.1 usa connessioni persistenti nella modalità di default



# Connessioni non persistenti

Supponiamo che l'utente immetta l'URL

`www.someSchool.edu/someDepartment/home.index`

(contiene testo,  
riferimenti a 10  
immagini jpeg)

1a. Il client HTTP inizializza una connessione TCP con il server HTTP (processo) a `www.someSchool.edu` sulla porta 80

2. Il client HTTP trasmette un *messaggio di richiesta* (con l'URL) nella socket della connessione TCP. Il messaggio indica che il client vuole l'oggetto `someDepartment/home.index`

1b. Il server HTTP all'host `www.someSchool.edu` in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

3. Il server HTTP riceve il messaggio di richiesta, forma il *messaggio di risposta* che contiene l'oggetto richiesto e invia il messaggio nella sua socket

tempo

# Connessioni non persistenti (continua)

4. Il server HTTP chiude la connessione TCP

5. Il client HTTP riceve il messaggio di risposta che contiene il file html e visualizza il documento html. Esamina il file html, trova i riferimenti a 10 oggetti jpeg

6. I passi 1-5 sono ripetuti per ciascuno dei 10 oggetti jpeg

tempo

# Connessioni persistenti

## Connessioni persistenti

- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta

### Connessione persistente **senza** pipelining:

- il client invia una nuova richiesta solo quando ha ricevuto la risposta precedente

### Connessione persistente **con** pipelining:

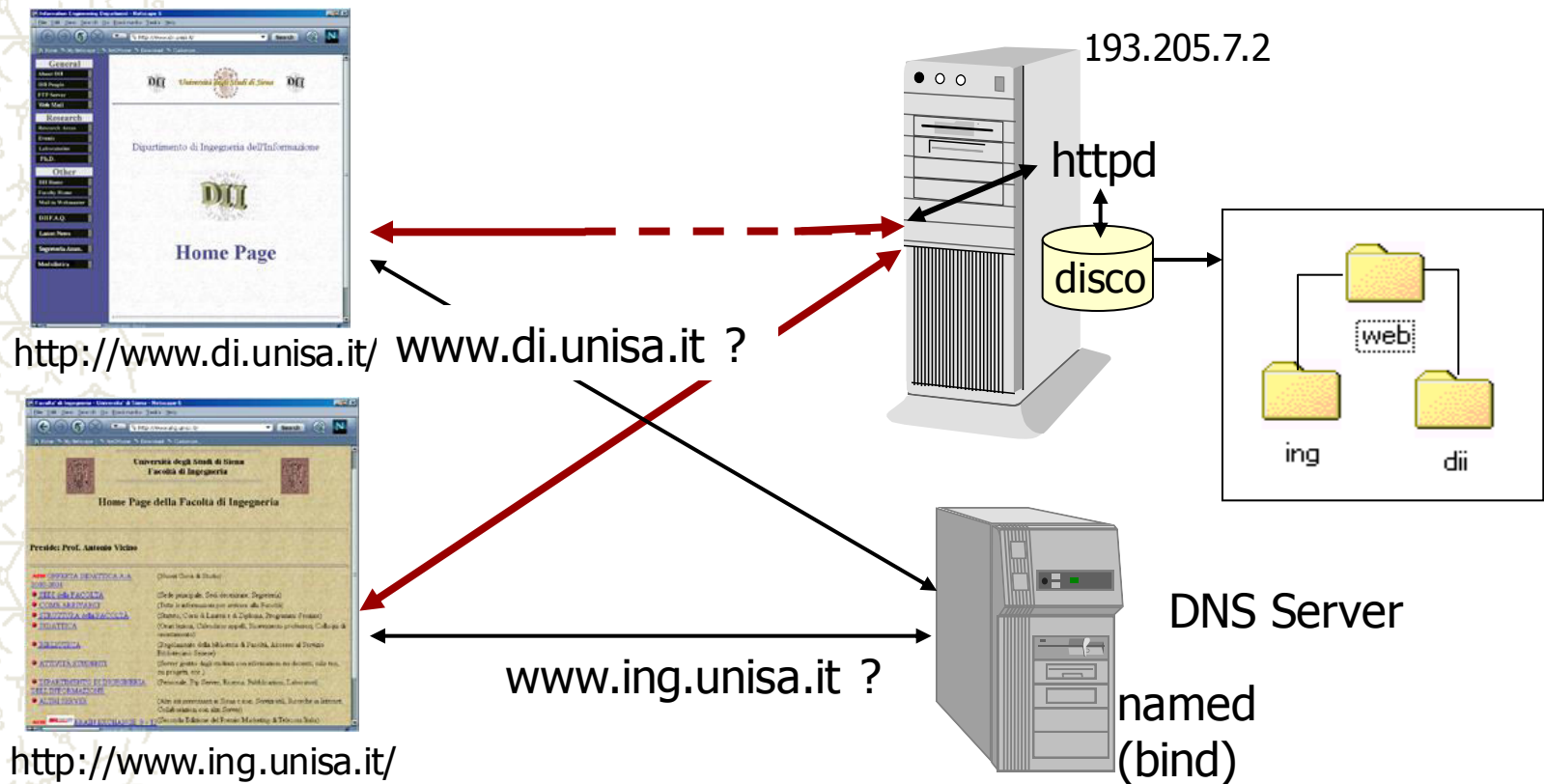
- è la modalità di default in HTTP/1.1
- il client invia le richieste non appena incontra un oggetto referenziato

# HTTP 1.1

- Permette più transazioni su una stessa connessione **persistente**
  - ▣ Attivo di default. Si inviano le richieste in pipelining e si recuperano le risposte nello stesso ordine
- Ha introdotto il supporto per le **cache** (If-Modified-Since:)
- Risposta più veloce per pagine generate dinamicamente utilizzando la codifica **chunked**
  - ▣ Non si deve specificare la lunghezza del messaggio nell'intestazione
  - ▣ Il messaggio è scomposto in blocchi (**chunks**)
- Uso migliore degli indirizzi IP permettendo di ospitare più **siti virtuali** su uno stesso server con un solo IP
  - ▣ Uso dell'intestazione **Host:** -> specifica il nome del sito a cui è indirizzata la richiesta

# Multi-homed IP

- Un server allo stesso indirizzo IP può gestire più domini
- Richiede alias nel DNS



# Linguaggio per siti web

Il linguaggio più diffuso con cui i siti web sono realizzati è:

***l'HTML***

***(Hyper Text Markup Language)***

- non è un linguaggio di programmazione.
- è un *linguaggio di markup*.

Un linguaggio di ***markup***:

- è un insieme di regole che descrivono le modalità di:
  - impaginazione,
  - formattazione,
  - visualizzazione grafica (layout) del contenuto di una pagina web.

L'HTML viene interpretato dai web browser.

# Tag HTML

## *L'HTML fa uso di tag (marcatori)*

- Hanno differenti nomi a seconda della loro funzione.
- I tag vanno inseriti tra parentesi angolari (**<TAG>**).
- Esiste un apertura di un tag: <nomeTag>
- Ed una chiusura attraverso "/": </nomeTag>

**Es.:**

```
<P align="right">testo</P>
```

- Il tag P indica un paragrafo
- Align="right" indica di spostare il contenuto a destra.

# Lista Tag (1)

## ***Elementi di base:***

Tipo	Tag	Descrizione
Tipo documento	<HTML></HTML>	Segnano l'inizio e la fine del file.
Titolo	<TITLE></TITLE>	Titolo della pagina, da inserire all'interno della testata.
Testata	<HEAD></HEAD>	informazioni descrittive; come il titolo.
Corpo	<BODY></BODY>	contenuto della pagina



# Lista Tag (2)

## ***Proprietà della pagina: sfondi e colori***

Tipo	Tag	Descrizione
Immagine di sfondo	<BODY BACKGROUND="URL">	Imposta l'imm. di sfondo.
Colore di sfondo	<BODY BGCOLOR="#*****">	Imposta il colore di sfondo.
Colore del testo	<BODY TEXT="#*****">	Imposta il colore del testo.
Colore dei collegamenti	<BODY LINK="#*****">	Imposta il colore dei collegamenti.
Colore dei collegamenti visitati	<BODY VLINK="#*****">	Imposta il colore dei colleg. visitati.
Colore del colleg. selezionato	<BODY ALINK="#*****">	Imposta il colore dei colleg. selez.

# Lista Tag (3)

## ***Formattazione del testo***

Tipo	Tag	Descrizione
Neretto	<code>&lt;B&gt;&lt;/B&gt;</code>	Imposta il neretto per il testo tra i tag.
Corsivo	<code>&lt;I&gt;&lt;/I&gt;</code>	Imposta il corsivo per il testo tra i tag.
Sottolineato	<code>&lt;U&gt;&lt;/U&gt;</code>	Sottolinea il testo tra i tag.
Testo scorrevole	<code>&lt;MARQUEE&gt;&lt;/MARQUEE&gt;</code>	Rende scorrevole il testo tra i tag.
Font: dimensioni	<code>&lt;FONT SIZE=?&gt;&lt;/FONT&gt;</code>	Setta la dimensione del testo tra i tag.
Font: colore	<code>&lt;FONT COLOR="#*****"&gt;&lt;/FONT&gt;</code>	Setta il colore del testo tra i tag.
Font: tipo	<code>&lt;FONT FACE="Verdana"&gt;&lt;/FONT&gt;</code>	Setta lo stile del testo tra i tag.

# Lista Tag (4)

## ***Collegamenti (link)***

Tipo	Tag	Descrizione
Link a un documento	<code>&lt;A HREF="URL"&gt;&lt;/A&gt;</code>	Collegamento ad un documento di indirizzo URL.
Definizione di un'ancora	<code>&lt;A NAME="nomeàncora"&gt;&lt;/A&gt;</code>	Setta un riferimento all'interno di una pagina.
Link a un'ancora dello stessa pagina	<code>&lt;A HREF="#nomeàncora"&gt;&lt;/A&gt;</code>	Collegamento all'ancora.
Link a un'ancora di un'altra pagina	<code>&lt;A HREF="URL#nomeàncora"&gt;&lt;/A&gt;</code>	Collegamento all'ancora che si trova in un'altra pagina.
Link a indirizzo e-mail	<code>&lt;A HREF="mailto:indirizzo@posta.it"&gt;&lt;/A&gt;</code>	Invia una email.

# Lista Tag (5)

## *Altro*

Tipo	Tag	Descrizione
Visualizzazione immagine	<code>&lt;IMG SRC="URL"&gt;</code>	Importa l'immagine contenuta nell'URL.
Separazione	<code>&lt;DIV&gt;&lt;/DIV&gt;</code>	Usato per raggruppare blocchi di elementi.
Paragrafo: allineamento	<code>&lt;P ALIGN=LEFT CENTER RIGHT JUSTIFY&gt; &lt;/P&gt;</code>	Allinea il paragrafo a sx, centro, dx, giustificato.
Elenco puntato	<code>&lt;UL&gt;&lt;LI&gt;&lt;/UL&gt;</code>	Crea un elenco puntato.
Inizio e fine tabella	<code>&lt;TABLE&gt;&lt;/TABLE&gt;</code>	Tag principali per la creazione di una tabella.
Riga	<code>&lt;TR&gt;&lt;/TR&gt;</code>	Inserisce una riga nella tabella.
Cella/Colonna	<code>&lt;TD&gt;&lt;/TD&gt;</code>	Inserisce una cella in una riga.

# Struttura file html

**<html>**

**<head>**

.....

**</head>**

**<body>**

....

....

....

**</body>**

**</html>**

Contiene informazioni non immediatamente percepibili, ma che riguardano il modo in cui il documento deve essere letto e interpretato. Questo è il luogo dove scrivere – ad esempio – i meta-tag, script JavaScript o VbScript, fogli di stile, ecc.

Qui è racchiuso il contenuto vero e proprio del documento.

# Editor

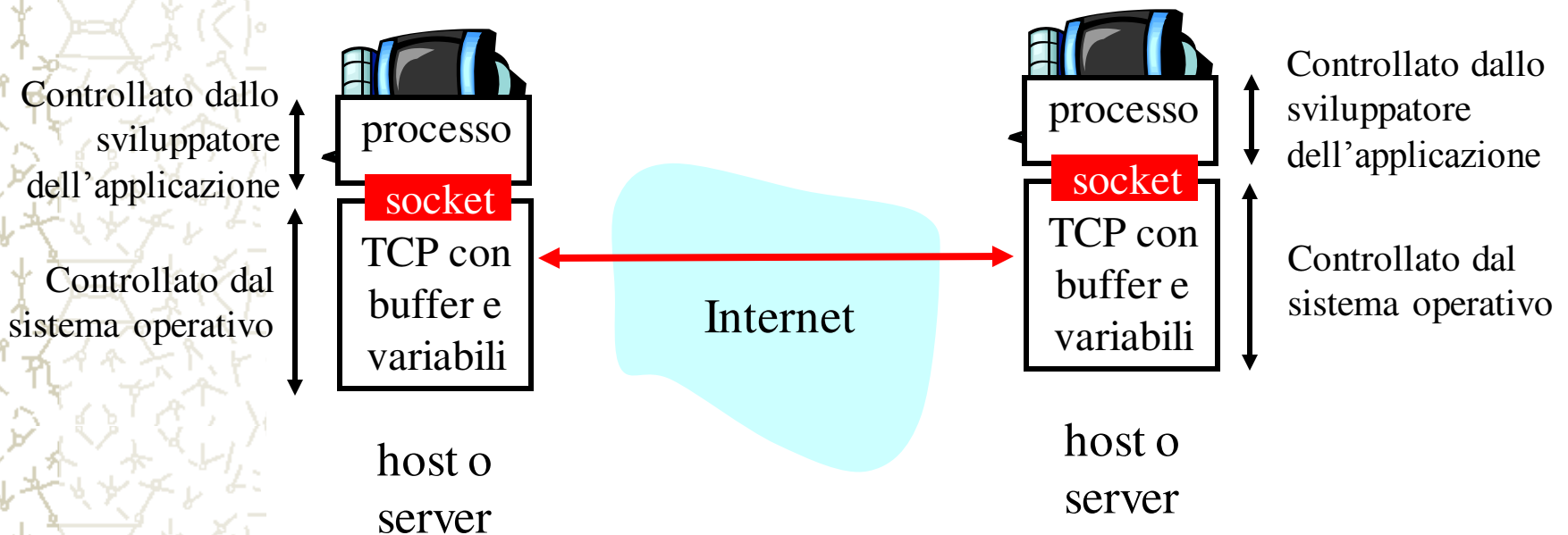
Esistono almeno 4 modi per creare una pagina html:

1. Con un programma WYSIWYG (What You See Is What You Get).
2. Usando un programma di videoscrittura (tipo Word) e salvare con l'opzione SALVA COME HTML.
3. Con un qualsiasi text-editor (Blocco Note, WordPad, WORD, PFE, ecc.). L'importante è che il file creato abbia come estensione ".html".
4. Usare dei text-editor pensati per scrivere in html. Questi infatti permettono di lavorare sul file sorgente ma contengono menu e bottoni che suggeriscono i comandi.

# Programmazione delle socket con TCP

Socket: una porta tra il processo di un'applicazione e il protocollo di trasporto end-end (UCP o TCP)

Servizio TCP: trasferimento affidabile di **byte** da un processo all'altro



# Programmazione delle socket con TCP

## Il client deve contattare il server

- Il processo server deve essere in corso di esecuzione
- Il server deve avere creato una socket (porta) che dà il benvenuto al contatto con il client

## Il client contatta il server:

- Creando una socket TCP
- Specificando l'indirizzo IP, il numero di porta del processo server
- Quando il **client crea la socket**: il client TCP stabilisce una connessione con il server TCP

- Quando viene contattato dal client, il **server TCP crea una nuova socket** per il processo server per comunicare con il client
  - consente al server di comunicare con più client
  - numeri di porta origine usati per distinguere i client

## Punto di vista dell'applicazione

*TCP fornisce un trasferimento di byte affidabile e ordinato (“pipe”) tra client e server*



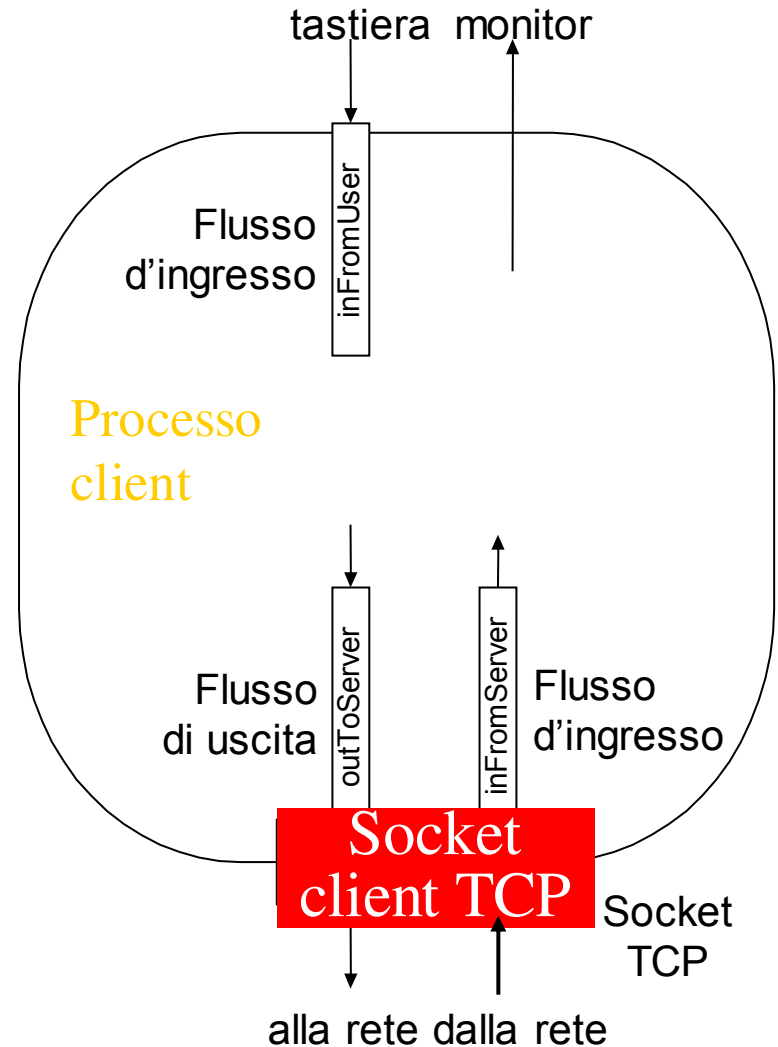
# Remarking: Termini

- Un **flusso** (*stream*) è una sequenza di caratteri che fluisce verso/da un processo.
- Un **flusso d'ingresso** (*input stream*) è collegato a un'origine di input per il processo, ad esempio la tastiera o la socket.
- Un **flusso di uscita** (*output stream*) è collegato a un'uscita per il processo, ad esempio il monitor o la socket.

# Programmazione delle socket con TCP

## Esempio di applicazione client-server:

- 1) Il client legge una riga dall'input standard (flusso **inFromUser**) e la invia al server tramite la socket (flusso **outToServer**)
- 2) Il server legge la riga dalla socket
- 3) Il server converte la riga in lettere maiuscole e la invia al client
- 4) Il client legge nella sua socket la riga modificata e la visualizza (flusso **inFromServer**)



# Interazione delle socket client/server: TCP

## Server (gira su `hostid`)

crea la socket  
port=`x` per la richiesta  
in arrivo:

`welcomeSocket =  
ServerSocket()`

attende la richiesta di  
connessione in ingresso  
`connectionSocket =  
welcomeSocket.accept()`

legge la richiesta da  
`connectionSocket`

scrive la risposta a  
`connectionSocket`

chiude  
`connectionSocket`

## Client

crea la socket  
connessa a `hostid`, port=`x`  
`clientSocket =  
Socket()`

invia la richiesta usando  
`clientSocket`

legge la risposta da  
`clientSocket`

chiude  
`clientSocket`

Setup della  
connessione TCP

# Esempio: client Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {
```

```
    public static void main(String argv[]) throws Exception
    {
```

```
        String sentence;
        String modifiedSentence;
```

Crea un  
flusso d'ingresso

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

Crea una  
socket client,  
connessa al server

```
        Socket clientSocket = new Socket("hostname", 6789);
```

Crea un  
flusso di uscita  
collegato alla socket

```
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
```

# Esempio: client Java (TCP), continua

Crea  
un flusso d'ingresso  
collegato alla socket

```
BufferedReader inFromServer =  
    new BufferedReader(new  
        InputStreamReader(clientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

Invia una riga  
al server

```
outToServer.writeBytes(sentence + '\n');
```

Legge la riga  
dal server

```
modifiedSentence = inFromServer.readLine();
```

```
System.out.println("FROM SERVER: " + modifiedSentence);
```

```
clientSocket.close();
```

```
}
```

```
}
```

# Esempio: server Java (TCP)

```
import java.io.*;  
import java.net.*;
```

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String clientSentence;  
        String capitalizedSentence;
```

Crea una socket  
di benvenuto  
sulla porta 6789

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

Attende, sulla socket  
di benvenuto,  
un contatto dal client

```
        while(true) {
```

```
            Socket connectionSocket = welcomeSocket.accept();
```

Crea un  
flusso d'ingresso  
collegato alla socket

```
            BufferedReader inFromClient =  
                new BufferedReader(new  
                    InputStreamReader(connectionSocket.getInputStream()));
```

# Esempio: server Java (TCP), continua

Crea un flusso di  
uscita collegato alla  
socket

```
DataOutputStream outToClient =  
    new DataOutputStream(connectionSocket.getOutputStream());
```

Legge la riga  
dalla socket

```
clientSentence = inFromClient.readLine();
```

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Scrive la riga  
sulla socket

```
outToClient.writeBytes(capitalizedSentence);
```

```
}  
}  
}
```

Fine del ciclo while,  
ricomincia il ciclo e attende  
un'altra connessione con il client

# Programmazione delle socket *con UDP*

UDP: non c'è "connessione" tra client e server

- Non c'è handshaking
- Il mittente allega esplicitamente a ogni pacchetto l'indirizzo IP e la porta di destinazione
- Il server deve estrarre l'indirizzo IP e la porta del mittente dal pacchetto ricevuto

UDP: i dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio

Punto di vista dell'applicazione

*UDP fornisce un trasferimento  
inaffidabile di gruppi di  
byte ("datagrammi")  
tra cliente e server*



# Interazione delle socket client/server: UDP

## Server (gira su `hostid`)

crea la socket  
`port=x` per la  
richiesta in arrivo:  
`serverSocket =`  
`DatagramSocket()`

legge la richiesta da  
`serverSocket`

scrive la risposta a  
`serverSocket`  
specificando l'indirizzo  
dell'host client e  
il numero di porta

## Client

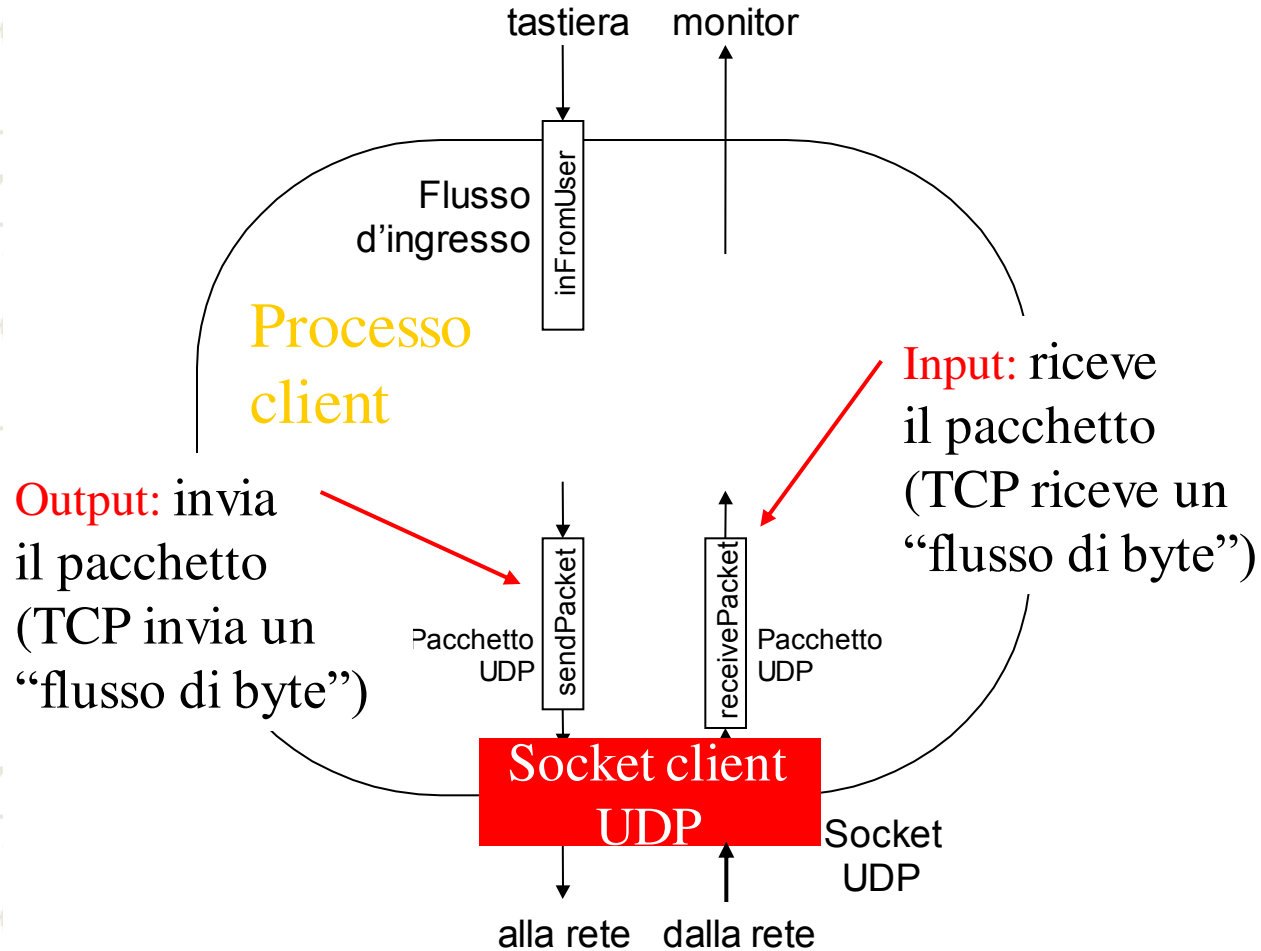
crea la socket  
`clientSocket =`  
`DatagramSocket()`

crea l'indirizzo (`hostid`, `port=x`)  
e invia la richiesta di datagrammi  
usando `clientSocket`

legge la risposta da  
`clientSocket`

chiude  
`clientSocket`

# Esempio: client Java (UDP)



# Esempio: client Java (UDP)

```
import java.io.*;
import java.net.*;
```

```
class UDPClient {
    public static void main(String args[]) throws Exception
    {
```

Crea un  
flusso d'ingresso

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

Crea una  
socket client

```
        DatagramSocket clientSocket = new DatagramSocket();
```

Traduce il  
nome dell'host  
nell'indirizzo IP  
usando DNS

```
        InetAddress IPAddress = InetAddress.getByName("hostname");
```

```
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
```

```
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

# Esempio: client Java (UDP), continua

Crea il datagramma con  
i dati da trasmettere,  
lunghezza,  
indirizzo IP, porta

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

Invia  
il datagramma  
al server

```
clientSocket.send(sendPacket);
```

Legge  
il datagramma  
dal server

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);
```

```
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());
```

```
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}
```

```
}
```

# Esempio: server Java (UDP)

```
import java.io.*;  
import java.net.*;
```

```
class UDPServer {  
    public static void main(String args[]) throws Exception  
    {
```

Crea una socket per  
datagrammi  
sulla porta 9876

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

```
        byte[] receiveData = new byte[1024];  
        byte[] sendData = new byte[1024];
```

```
        while(true)  
        {
```

Crea lo spazio per  
i datagrammi

```
            DatagramPacket receivePacket =  
                new DatagramPacket(receiveData, receiveData.length);
```

Riceve i  
datagrammi

```
            serverSocket.receive(receivePacket);
```

# Esempio: server Java (UDP), continua

```
String sentence = new String(receivePacket.getData());
```

Ottiene  
l'indirizzo IP e  
il numero di porta  
del mittente

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

Crea  
il datagramma  
da inviare  
al client

```
sendData = capitalizedSentence.getBytes();
```

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress,  
                        port);
```

Scrive  
il datagramma  
sulla socket

```
serverSocket.send(sendPacket);
```

```
}  
}  
}
```

Fine del ciclo while,  
ricomincia il ciclo e attende  
un altro datagramma