Reti di calcolatori

Modulo III vers. 4.0
Il livello applicativo

Claudio Covelli
claudio.covelli@gmail.com

Facoltà di Scienze Matematiche, Fisiche e Naturali
Università di Trento

Indice

- Introduzione ai protocolli applicativi
- Architetture: client-server e peer-to-peer (P2P)
- Principali protocolli applicativi client-server DNS, SMTP, HTTP
- Telnet ed FTP (cenni)
- Socket e trasmissione/ricezione su Internet dei messaggi applicativi

Architetture di rete dal punto di vista applicativo

- Due host, collegati in Internet (***), si scambiano fra loro dei messaggi (es. web client e web server)
- Si tratta di messaggi codificati secondo determinati protocolli standard detti applicativi, in quanto vengono gestiti da applicazioni sofware appositamente realizzate ed installate a bordo degli host (livello 7-6-5 dello stack ISO-OSI)
- Tali protocolli stabiliscono il formato dei messaggi e la loro sequenza di utilizzo
- Dal punto di vista dei protocolli applicativi, sono utilizzate in Internet due differenti architetture:
  - Client-server (il più diffuso modello, anche per ragioni storiche)
  - Peer to peer (nate dopo il 2000, in rapida crescita)

*** (vedremo nelle slides successive cosa significa esattamente questo termine)

Architettura client-server

- Le applicazioni client si collegano, in Internet , a server, in grado di fornire specifici servizi

- I server sono generalmente degli host ad elevata affidabilità (hardware ridondato e controllato) dislocati, solitamente, in apposite strutture (data center) e sui quali sono installate delle applicazioni in grado di fornire i servizi richiesti

- E' sempre l' applicazione client (es. browser) ad:
  - attivare una specifica connessione (socket) con il server e quindi
  - richiedere, mediante uno specifico protocollo applicativo, un determinato servizio (ad esempio richiesta di una pagina html ad un server web)

- Il server, ricevuta la richiesta:
  - crea, a sua volta, una connessione specifica con il client (socket)
  - fornisce a questo il servizio richiesto

Architettura client-server

```
// Esempio di socket lato client
protected String host;
protected int port;
protected DataInputStream in;
protected DataOutputStream out;

protected Socket connect () throws IOException {
  System.err.println ("Connessione a " + host + ":" + port + "...");
  Socket socket = new Socket (host, port);
  System.err.println ("Connessione avvenuta.");
  out = new DataOutputStream (socket.getOutputStream ());
  in = new DataInputStream (socket.getInputStream ());

  out.writeBytes ("GET " + file + " HTTP/1.0\r\n\r\n");
  ..................................................................
  return socket;
}
```

Architettura client-server

```
// Esempio di socket lato server
System.out.println ("Server in partenza sulla porta " + port);
ServerSocket server = new ServerSocket (port);
System.out.println ("Server partito sulla porta " +     server.getLocalPort() ) ;
System.out.println ("In attesa di connessioni...");
Socket client = server.accept ();
System.out.println ("Richiesta di connessione da " +  client.getInetAddress () ) ;
InputStream i = client.getInputStream ();
OutputStream o = client.getOutputStream ();
PrintStream p = new PrintStream (o) ;
p.println("BENVENUTI.")
```

Architettura client-server

client

Internet

Data center

server

Architettura client-server

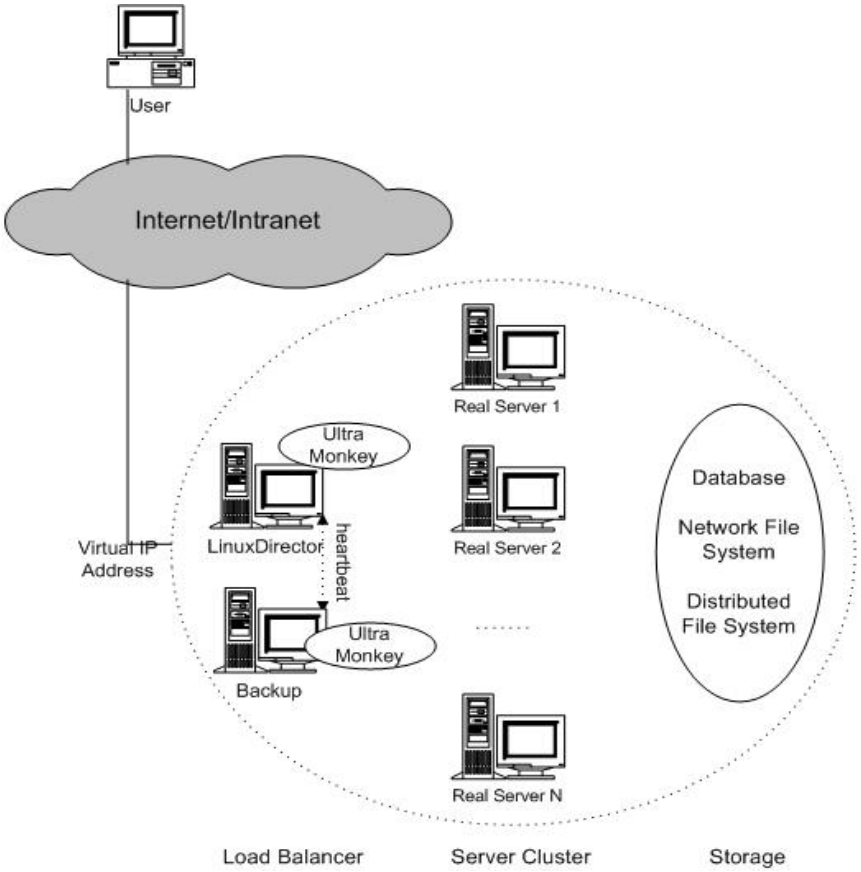
- Nell'architettura client-server, il server rappresenta il punto più vulnerabile (single point of failure), in quanto la sua compromissione, per svariate ragioni, (es. errore hardware/software, attacco DoS etc) implica l'interruzione del servizio

- Inoltre tutta la richiesta di banda trasmissiva è concentrata su di esso e quindi, in particolari situazioni, le performance del servizio possono diventare critiche

- Si ricorre pertanto a particolari tecniche (es. server ridondati, stesso servizio distribuito in modo trasparente su più server fisici) per far fronte a questo tipo di problema

- Load balancing ed high availability

Architettura client-server



Esempio di architettura client-server con load balancing ed in alta affidabilità

Per dettagli:

http://www.howtoforge.com/high_availability_loadbalanced_apache_cluster_p4

Architettura client-server

Note importanti

- I protocolli applicativi client-server che andremo ad esaminare presuppongono che client e server siano collegati in Internet
- Come vedremo meglio in seguito, i messaggi applicativi non vengono, infatti, trasmessi tali e quali ma sono suddivisi in pacchetti, ossia blocchi di bytes di lunghezza fissa, che contengono una parte del messaggio da trasmettere (payload) ed informazioni per la corretta consegna del pacchetto al destinatario (header)
- Internet è l'infrastruttura hardware e software che consente la trasmissione/ricezione di tali pacchetti, basandosi su protocolli specifici standard (TCP/UDP, IP)
- Quindi tutti i protocolli client e server applicativi presuppongono la disponibilità dei protocolli standard Internet sugli host client e server

Architettura client-server

Note importanti

- Questo spiega anche il motivo per i quali i protocolli Internet sono necessari anche nell'ambito di una singola LAN, dove la comunicazione potrebbe teoricamente avvenire facendo sì che i protocolli applicativi interfaccino direttamente i driver della scheda di rete ed usando, per la trasmissione, esclusivamente la tecnologia di rete disponibile (es. Ethernet o FrameRelay od ATM)

- In realtà i protocolli applicativi che andremo ad analizzare si basano necessariamente sui protocolli Internet per la trasmissione e ricezione dei messaggi, garantendo in tal modo l'internetworking (client e server su reti diverse, non solo geograficamente ma anche in termini di tecnologia utilizzata)

- Questo concetto si esprime con il termine di "incapsulamento" dei protocolli (encapsulation): i dati dei protocolli di livello superiore (es. messaggi) vengono consegnati ai protocolli di livello inferiore (TCP,UDP, IP) per la loro corretta trasmissione, come si vedrà meglio in seguito trattando le socket

Architettura peer-to-peer (P2P)

- Nuovo modello architetturale (nato verso gli anni 2000 con Napster) nel quale ogni host può essere, contemporaneamente, client e server, richiedente e fornitore di servizi (peer,servent)

- Internet stesso, secondo il progetto originale Arpanet, nasce come architettura che rende possibile lo scambio di messaggi e dati, libero e da pari a pari, fra host aventi caratteristiche hardware e software differenti (es. telnet, ftp)

- Poi, per i ben noti problemi di sicurezza, con il tempo Internet tende a divenire un sistema chiuso, con regole rigide per gli accessi (verso gli anni 90 nascono i firewall)

- Il modello peer-to-peer cerca di ritornare allo spirito originario di Internet, anche se, nelle sue applicazioni concrete, orientate soprattutto al file-sharing, sta creando non pochi problemi legati al rispetto del copyright

Architettura peer-to-peer (P2P)

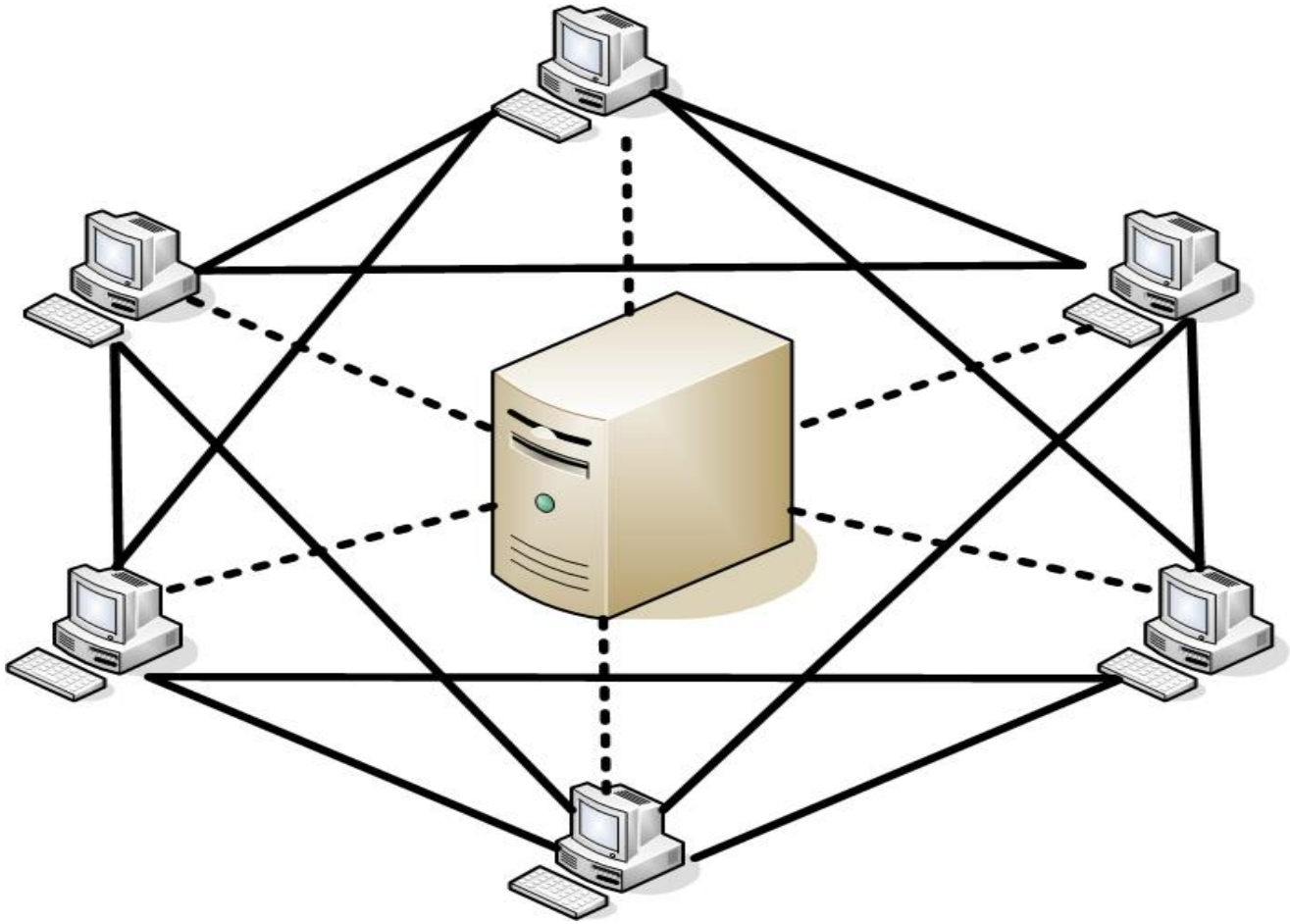
- Al di là di questo aspetto, è importante evidenziare alcuni punti di forza dell'architettura p2p, sulla quale si basano anche applicazioni diverse dal puro e semplice file-sharing (es. skype)

- Maggiore banda utilizzabile per il download, in quanto ogni peer mette a disposizione la propria (anche se con gli evidenti limiti posti dalla asimmetricità delle connessioni ADSL)

- Ridondanza dei dati, replicati su più host contemporaneamente

- Minori costi di gestione (non esiste più il concetto di data center), anche se questo vantaggio comporta una complicazione dei protocolli, in quanto occorre tener conto del fatto che un peer può disconnettersi, in qualsiasi momento, dalla rete

Architettura centralizzata P2P



Architettura centralizzata (P2P)

- Nell'architettura p2p di tipo centralizzato lo scambio di dati (files) avviene direttamente fra peers, ma vi è un server che ha il compito di gestire gli indici per la ricerca (esempio molto noto: Napster)

- Il server possiede l' elenco completo dei nomi dei files scaricabili e l'indirizzo Ip dei peers attivi, in grado di fornire ognuno di essi

- Ogni messaggio è nella forma <length> <type> <data>, ove length e type occupano ognuno 2 bytes

- Il peer apre una connessione con il server ed effettua il login (user + password + altri parametri opzionali quali la porta e la velocità di connessione)

- Il server controlla l'identità del peer ed invia un messaggio di tipo ACK al peer

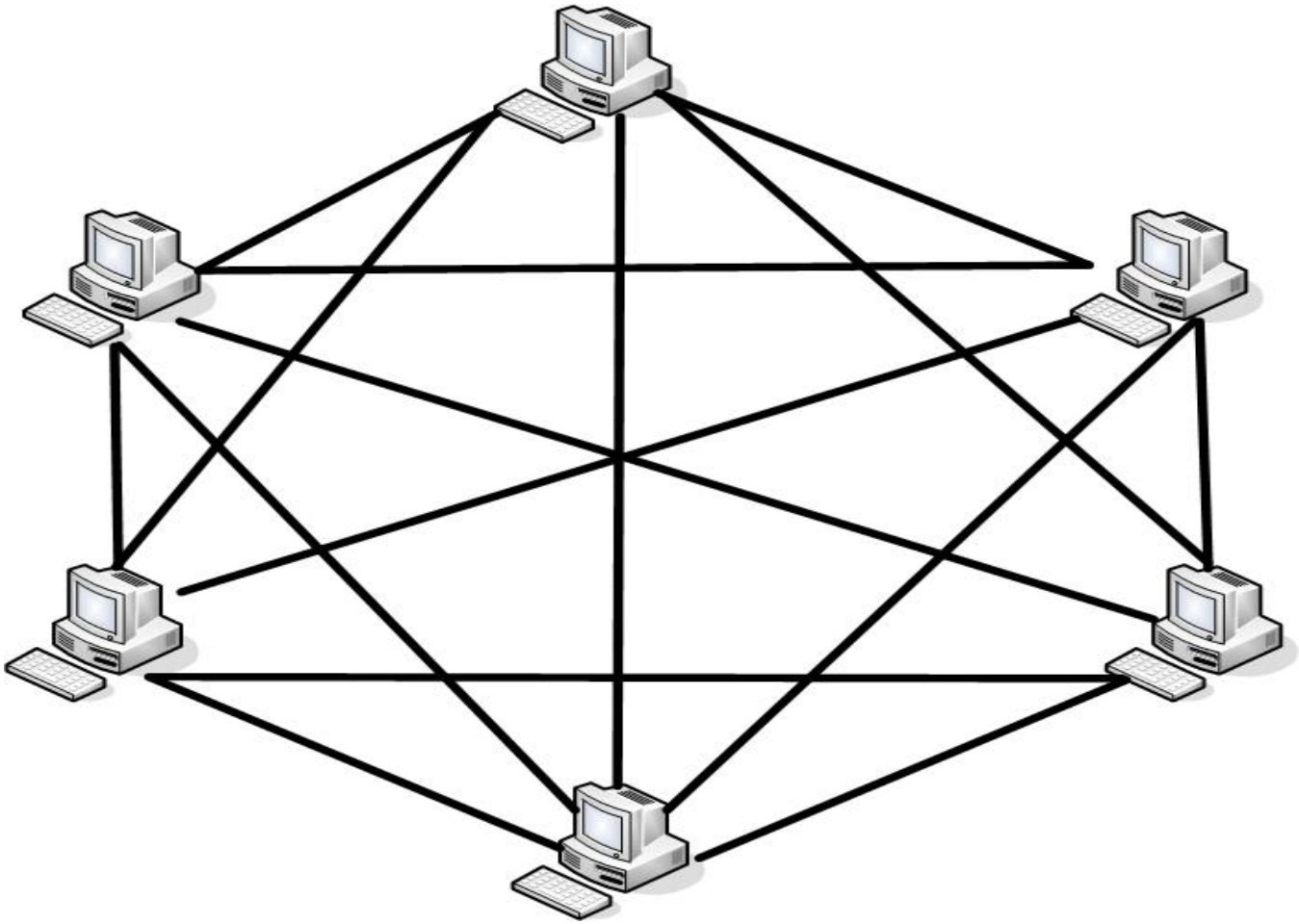
Architettura centralizzata (p2p)

- Il peer invia quindi al server la propria lista di files che vuole condividere con altri peers
- Quando il peer vuole effettuare il download di un certo file, invia un messaggio di tipo SEARCH al server , specificando delle parole di ricerca ed il numero massimo di risultati desiderati
- Il server verifica nel suo database la presenza di files che soddisfano il criterio di ricerca richiesto ed invia al peer richiedente dei messaggi di tipo RESPONSE
- Ognuno di questi messaggi di response contiene username ed indirizzo Internet (IP) del peer che possiede il file cercato ; il peer sceglie fra tutti uno dei peer e lo comunica al server che invia al peer richiedente informazioni addizionali (es. la porta per il download)
- Il peer contatta quindi il peer prescelto ed effettua il download

Napster: sintesi

- Napster, chiuso dall'autorità giudiziaria per problemi di copyright, rappresenta un esempio di architettura di transizione (client/server + p2p)
- Protocollo applicativo basato su protocolli Internet standard (TCP,IP)
- Vi è infatti ancora la presenza di un server centrale e quindi, anche in questo tipo di architettura, il server rappresenta un single point of failure
- Si richiede che ogni peer metta a disposizione banda trasmissiva e dati
- L'architettura p2p però dimostra chiaramente i suoi vantaggi: il download non richiede banda al server e la banda necessaria viene condivisa fra tutti i peer
- Scalabilità: la potenza della rete aumenta all'aumentare dei peer connessi

Architettura P2P pura (es. gnutella)



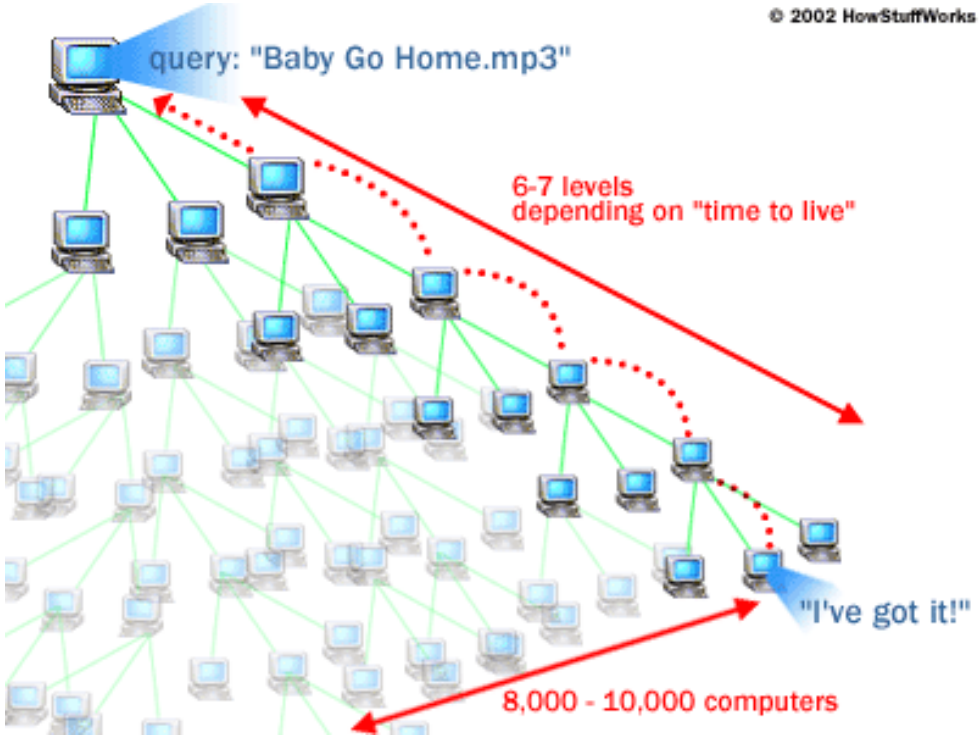
Pure P2P Networks

- Non esiste un server centrale (peer to peer puro)
- E' orientato al file sharing
- Peer to peer puro (modello decentralizzato). Ogni host, che si collega alla rete gnutella, può ricercare e mettere a disposizione dei files (servent)
- Per connettersi, occorre, inizialmente, disporre di almeno un peer già presente in rete (di solito tali peers sono preconfigurati sul client o reperiti sul web)
- Il client si connette a tale peer con il comando CONNECT e scambia con esso informazioni relative al tipo e numero di files in condivisione
- Specifici comandi del protocollo (ping, pong) consentono di individuare altri peer collegati al peer connesso ed il numero di files messi a disposizione da ognuno di essi. Ad ogni messaggio viene associato un valore di TTL (time to live)

Pure P2P Networks

- La ricerca di un file avviene inviando uno specifico messaggio (QUERY) al peer connesso, indicando il file da cercare ed il numero massimo di iterazioni di ricerca (TTL)
- Se uno dei peer contattati non dispone del file cercato, esso risponde con un messaggio (QUERYHIT) che contiene l'elenco dei files in sharing ed inoltra il messaggio agli altri peer conosciuti, decrementando di 1 il valore di TTL associato al messaggio (Flooding)
- Il processo di diffusione si interrompe solo quando il valore di TTL assume il valore 0: in tal caso l'ultimo peer ferma il processo di inoltro (forward)
- Ottenute le risposte, il richiedente contatta uno dei peer che possiede il file cercato e lo scarica usando il protocollo http

Pure P2P Networks



Ricerca dei peers in una rete P2P pura

P2P Networks: conclusioni

- Le reti P2P mostrano il loro punto di forza nell'utilizzo condiviso della banda; anche se viene generato molto traffico superfluo,viene meglio sfruttata la banda che ogni peer ha a disposizione
- Le informazioni sono ridondate e distribuite su più peer: nuove architetture (es. Bittorrent) consentono inoltre la suddivisione di files in blocchi ad il conseguente download in parallelo da più peers
- Si tratta peraltro di architetture adatte a particolari tipi di applicazioni (es. File sharing) anche se si stanno diffondendo applicazioni di altro tipo (es. skype)
- Peraltro le statistiche del traffico su Internet fanno pensare ad una loro forte crescita in un forte futuro

Applicazioni client-server

- Dopo questa necessaria analisi dei due modelli architetturali più diffusi, torniamo alle applicazioni client-server per un loro approfondimento
- Si andranno ad analizzare i principali protocolli, con un particolare riguardo al loro scopo ed ai messaggi scambiati fra client e server
- Quello che si evidenzierà in modo chiaro è che i protocolli applicativi si basano sempre su una sequenza standard e predefinita di messaggi
- Alla fine di questo modulo si andrà infine ad approfondire come i messaggi dei protocolli applicativi vengono trasmessi nell'ambito della rete Internet
- Il primo protocollo applicativo che viene analizzato è quello utilizzato per la conversione di nomi mnemonici in indirizzi IP ossia il protocollo utilizzato da DNS (Domain Name System)

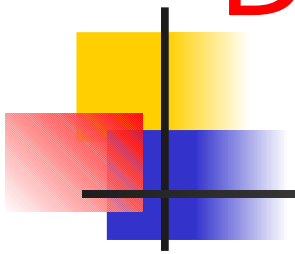
DNS

- Ogni host visibile su Internet, sia esso client o server o peer, viene identificato mediante uno specifico indirizzo, univoco a livello mondiale
- Questo indirizzo prende il nome di indirizzo IP (es. 8.1.2.3)
- L'indirizzo IP è un numero binario di 32 bits e viene utilizzato da tutti i protocolli applicativi che utilizzano Internet per la consegna dei messaggi
- Nella sua rappresentazione decimale esso viene suddiviso in 4 bytes e di ognuno di questi bytes viene fornita la rappresentazione in formato decimale, separata dalle rimanenti con punti decimali
- L'indirizzo IP viene assegnato da determinate Authorities (per l'utente finale di solito è un ISP Internet Service Provider) in modo da garantirne l'univocità e viene associato alla scheda di rete dell'host in fase di configurazione (es.comando ifconfig od ip)

DNS

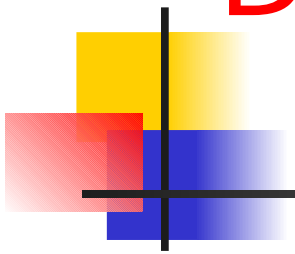
- Essendo univoco, l'indirizzo IP è uno dei parametri associati al messaggio che consente, in fase di trasmissione , di indicare in modo preciso a quale degli host tale messaggio vada inviato
- Prendiamo ad esempio il caso di un client applicativo (es. web browser, client di posta) che invii un certa richiesta ad un server
- Tale richiesta deve corrispondere ad uno dei messaggi codificati nel protocollo utilizzato (es. “GET index.html http 1.1” per web browser)
- L'applicazione software client, deve associare a tale messaggio l'indirizzo IP del server di destinazione
- Questo indirizzo può essere indicato dalla persona che usa il client, inserendo, nella apposita casella di testo del browser, l'indirizzo IP dell'host da contattare

DNS



- Ad essere precisi, di per sé questa informazione non è sufficiente ed occorre anche associare al messaggio il numero di porta (port), che indica a quale delle applicazioni dell'host di destinazione il messaggio deve essere consegnato
- Infatti su tale host potrebbero essere presenti più applicazioni (es. un server web, un server smtp etc) e quindi il solo indirizzo IP non è sufficiente per la corretta consegna
- Tali concetti verranno comunque ripresi ed approfonditi fra poco, quando parleremo di socket
- Ora quasi mai la persona che usa l'applicazione client (web browser, client di posta elettronica etc), è in grado di sapere l'esatto indirizzo IP del server

DNS



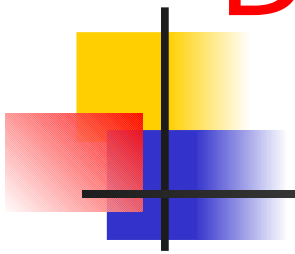
- Occorre quindi un meccanismo che semplifichi tale operazione; esso consiste nell'associare, ad un indirizzo IP dell'host, un nome mnemonico (es. www.unitn.it al posto di 193.205.206.17)

- L'associazione fra nome ed indirizzo IP viene registrata, come meglio vedremo in seguito, in un particolare database (nameserver) che ha la caratteristica di essere distribuito

- Infatti, ogni organizzazione che gestisce alcuni host, visibili su Internet, deve farsi carico di gestire una porzione di questo database, in modo da fornire a tutti il mapping fra nome mnemonico ed indirizzo IP di tali host

- E' comunque importante evidenziare che tutti i protocolli applicativi usano, per la trasmissione dei messaggi, il solo indirizzo IP. Il nome mnemonico viene utilizzato solo dall'utente dell'applicazione

DNS



- Esiste pertanto un particolare protocollo applicativo, richiamato a sua volta da tutti i protocolli applicativi, che consente la trasformazione del nome mnemonico in indirizzo IP (mapping)e viceversa (reverse mapping)

- Questo protocollo di risoluzione dei nomi è incluso in uno standard più ampio, detto DNS (domain name system), che definisce anche tutti gli elementi corollari e la struttura organizzativa che lo rendono possibile

- Nelle slides successive andremo ad analizzare in dettaglio il DNS, la sua organizzazione ed architettura e si forniranno informazioni sui tipi di messaggi utilizzati per risolvere un nome mnemonico in indirizzo IP

DNS

- Il DNS (Domain Name System) è:
  - Un insieme di nomi, organizzati in modo gerarchico, che consente di individuare in modo univoco gli host di Internet (name space) mediante un opportuno nome mnemonico
  - Un insieme di authorities, gerarchicamente collegate, responsabili dell'assegnazione univoca di tali nomi nel name space
  - Un insieme di server ed un protocollo software, per la gestione e l'utilizzo di tali nomi, in modo da rendere possibile la conversione automatica da nome mnemonico ad indirizzo IP e viceversa (reverse mapping)

DNS

Un po' di storia

- Inizialmente i client gestivano il mapping, memorizzando un file testuale (HOST.TXT) riportante, per ogni indirizzo IP, il relativo nome mnemonico
- La versione originale di tale file era detenuta presso l'Università di Stanford (SRI NIC) che gestiva tutti gli indirizzi IP mondiali
- Periodicamente ogni client scaricava, via FTP, tale file sul proprio hard disk in modo da aggiornare le definizioni
- Con il crescere di Internet, questo sistema si rivelò del tutto inefficace

DNS

Un po' di storia

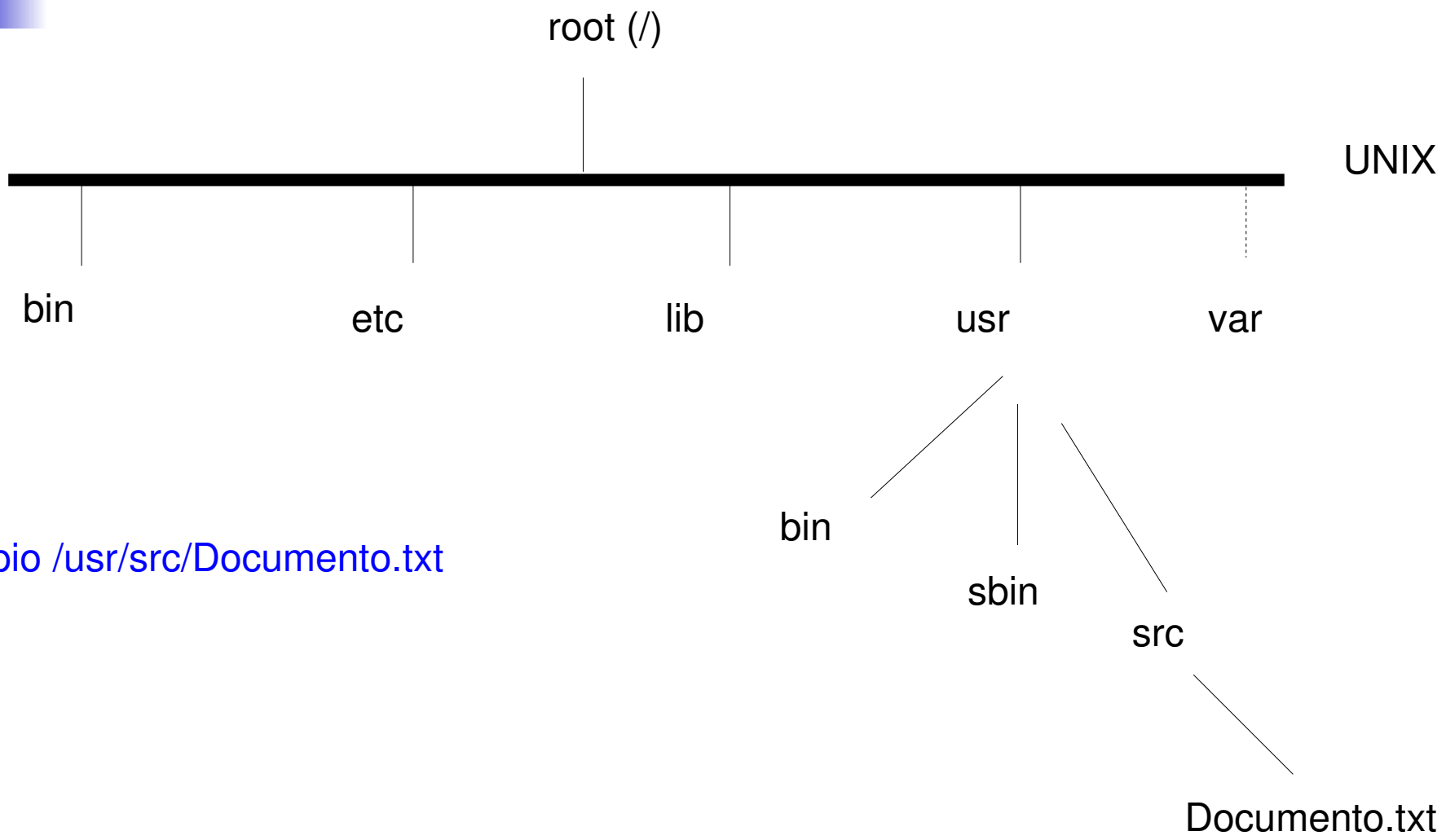
- Banda: il file host.txt assumeva, di giorno in giorno, dimensioni sempre maggiori e questo comportava elevato impegno di banda nel download
- Univocità dei nomi: SRI NIC assegnava gli indirizzi ma non aveva alcuna autorità sui nome. Ciò comportava potenziali problemi di duplicazione (clashing) con effetti ben evidenti
- Inefficienza: ogni giorno sempre più host si collegavano ad Internet ed il sistema basato sulla distribuzione di un singolo file non riusciva più a garantire una rapida sincronizzazione

DNS

- Si progettò quindi una nuova architettura (DNS) che:
  - consentisse una gestione distribuita del mapping
  - ne garantisse, nel contempo, una visione integrata ed unitaria

- I nomi degli host appartenti ad Internet vengono definiti, in analogia ai nomi delle directory di un file system Unix, mediante una struttura ad albero rovesciato (inverted tree) in cui:
  - la radice rappresenta il cosiddetto dominio di root (analoga alla root di un file system unix)
  - ogni nodo costituisce, a sua volta, la radice di un sottoalbero (subtree)
  - ogni subtree rappresenta un dominio (directory in Unix)
  - ogni foglia (nodo senza ulteriori diramazioni) rappresenta un host

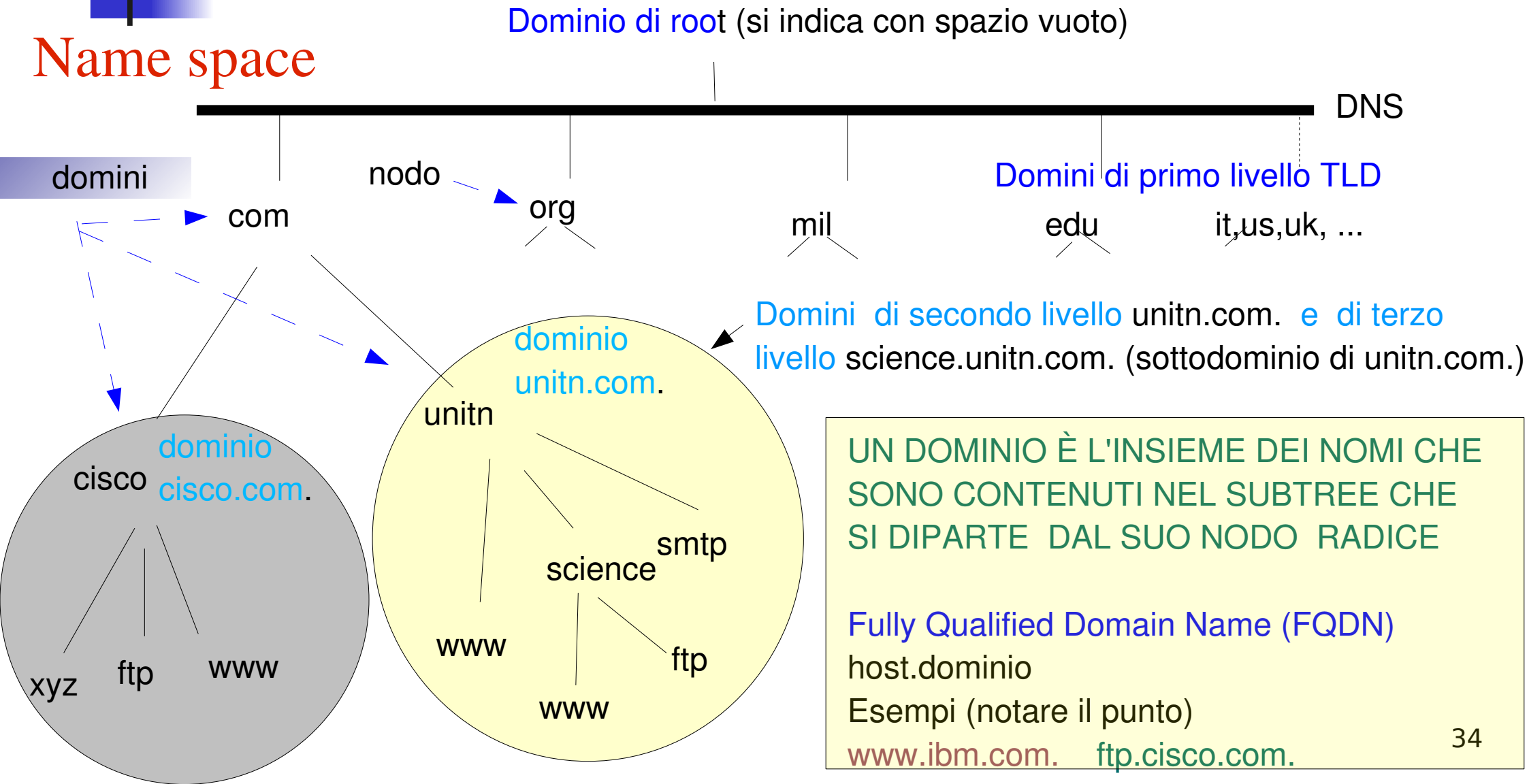
DNS



esempio /usr/src/Documento.txt

DNS

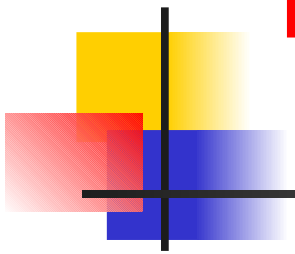
Name space



UN DOMINIO È L'INSIEME DEI NOMI CHE SONO CONTENUTI NEL SUBTREE CHE SI DIPARTE DAL SUO NODO RADICE

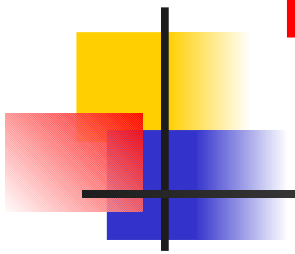
Fully Qualified Domain Name (FQDN)
host.dominio
Esempi (notare il punto)
www.ibm.com. ftp.cisco.com.

DNS



- Il dominio indica quindi un sottoinsieme di nomi , identificato dal subtree che si diparte dal nodo che rappresenta la sua radice
- Ogni nodo ed ogni foglia hanno una label, analogamente ai files ed alle directories di Unix
- Il nome di dominio (FQDN) è formato dai nomi, separati da punti, di tutte le label incontrate partendo dal nodo che ne rappresenta la radice, salendo verso la root (up the tree)
- Per la root non si usa alcuna label
- Un host si indica con il nome della foglia seguito dal punto e dal nome del dominio di appartenenza (es. www.ibm.com.)

DNS



- Il DNS prevede anche un'organizzazione di authorities e server (nameserver) per la gestione e risoluzione dei nomi inseriti nel name space
- Anche le authorities sono organizzate gerarchicamente; esse agiscono per delega
- L'authority principale (ICANN), non profit con sede in California, assegna i nomi ai domini di primo livello generici (gTLD) e country-code (ccTLD)
- ICANN gestisce anche i nuovi TLD biz. , coop. , info. etc
- ICANN delega ad altre authorities (registry) la gestione dei domini di primo livello (ad es. it. è delegato al NIC di Pisa)

DNS

- I registry , a loro volta, delegano la gestione delle procedure di registrazione di un loro sottodominio di secondo livello (es. unitn.it.) ad organizzazioni dette registrar o maintainer, che registrano tale sottodominio presso il registry su richiesta del cliente richiedente (registrant)

- Solo dopo il completamento delle procedure di registrazione, gli host del dominio saranno accessibili via Internet

- Si noti che la responsabilità di un certo sottodominio è in carico al registrant; il registrar/maintainer è solamente un intermediario

- Ogni registrant gestisce e controlla, in sintesi, un certo sottoinsieme del name space; tale sottoinsieme viene definito zona

DNS

- L'authority alla quale viene delegata una zona (registrant) mette, solitamente, a disposizione almeno due nameserver (primario e secondario) per la risoluzione dei nomi in essa contenuti (name server autoritativi per la zona)

- In alternativa la zona può essere, per comodità, gestita tramite i nameserver del registrar; un nameserver può infatti gestire più zone contemporaneamente. La responsabilità della zona e l' aggiornamento dei relativi nomi è comunque sempre a carico del registrant.

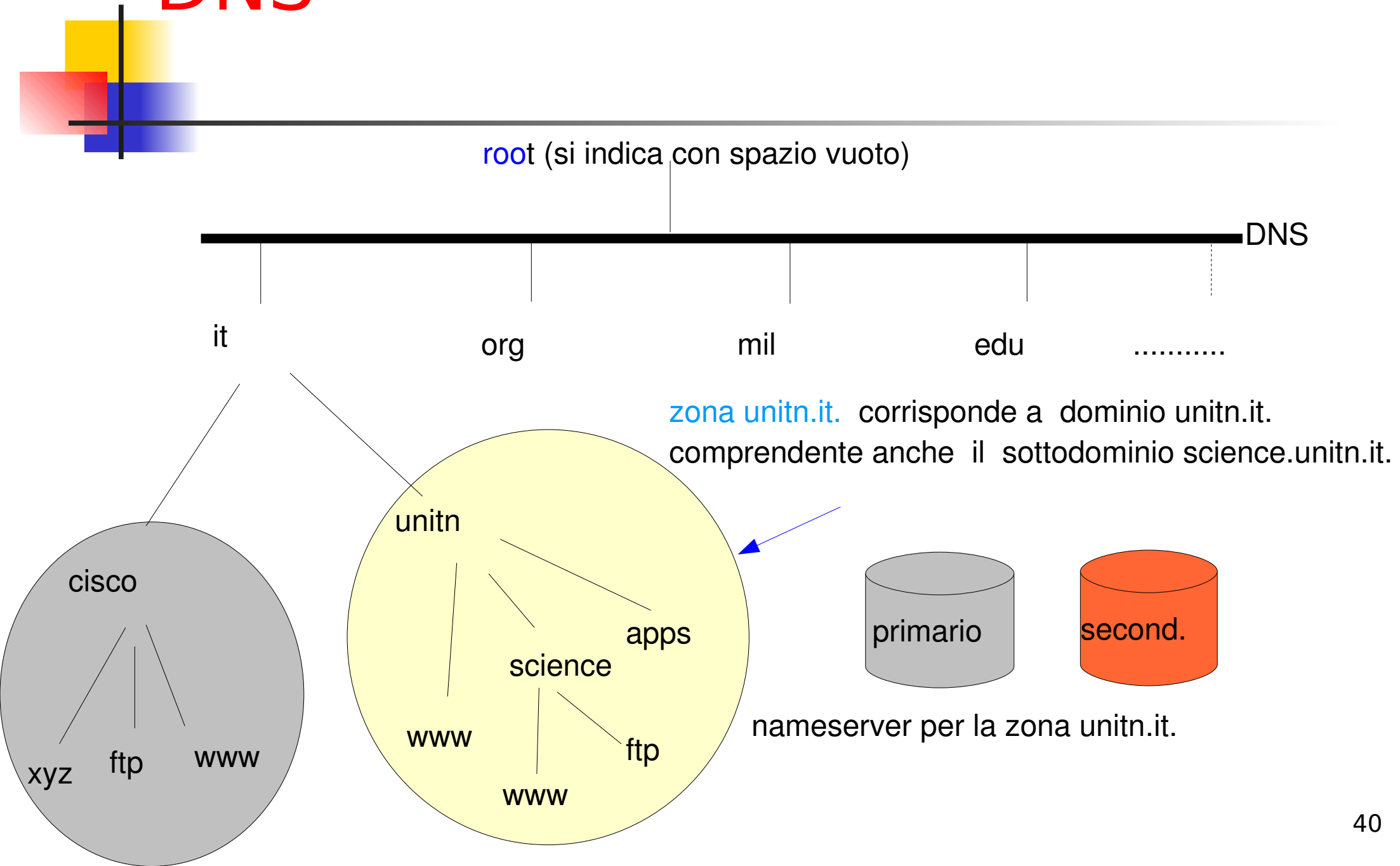
- La zona può coincidere con un dominio; se però uno o più sottoinsiemi di nomi di tali domini vengono delegati ad altre authorities si è in presenza di un dominio suddiviso in più zone (ossia ad un dominio corrispondono n zone; ad es. al dominio unitn.it. potrebbero corrispondere le zone unitn.it. e science.unitn.it.)

DNS

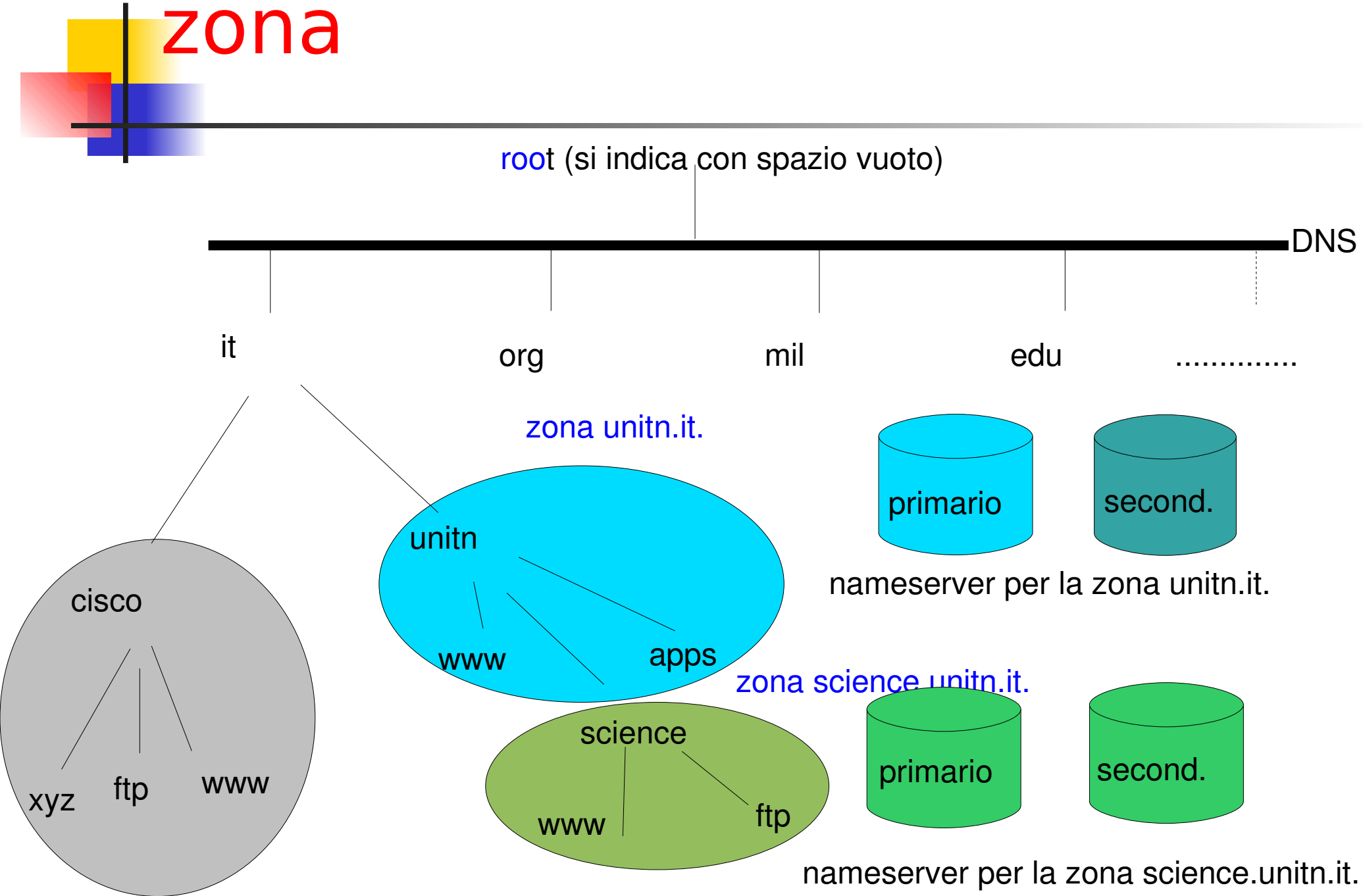
- Gerarchia di authorities: ICANN -> Registry -> Registrant (la zona può essere gestita tramite i nameserver del registrar)

- Se un sottodominio, in carico ad un certo registrant, viene ulteriormente delegato, la relativa zona può essere gestita mediante nameserver autonomi o tramite i nameserver del registrant delegante (vedi esempi slides seguenti)

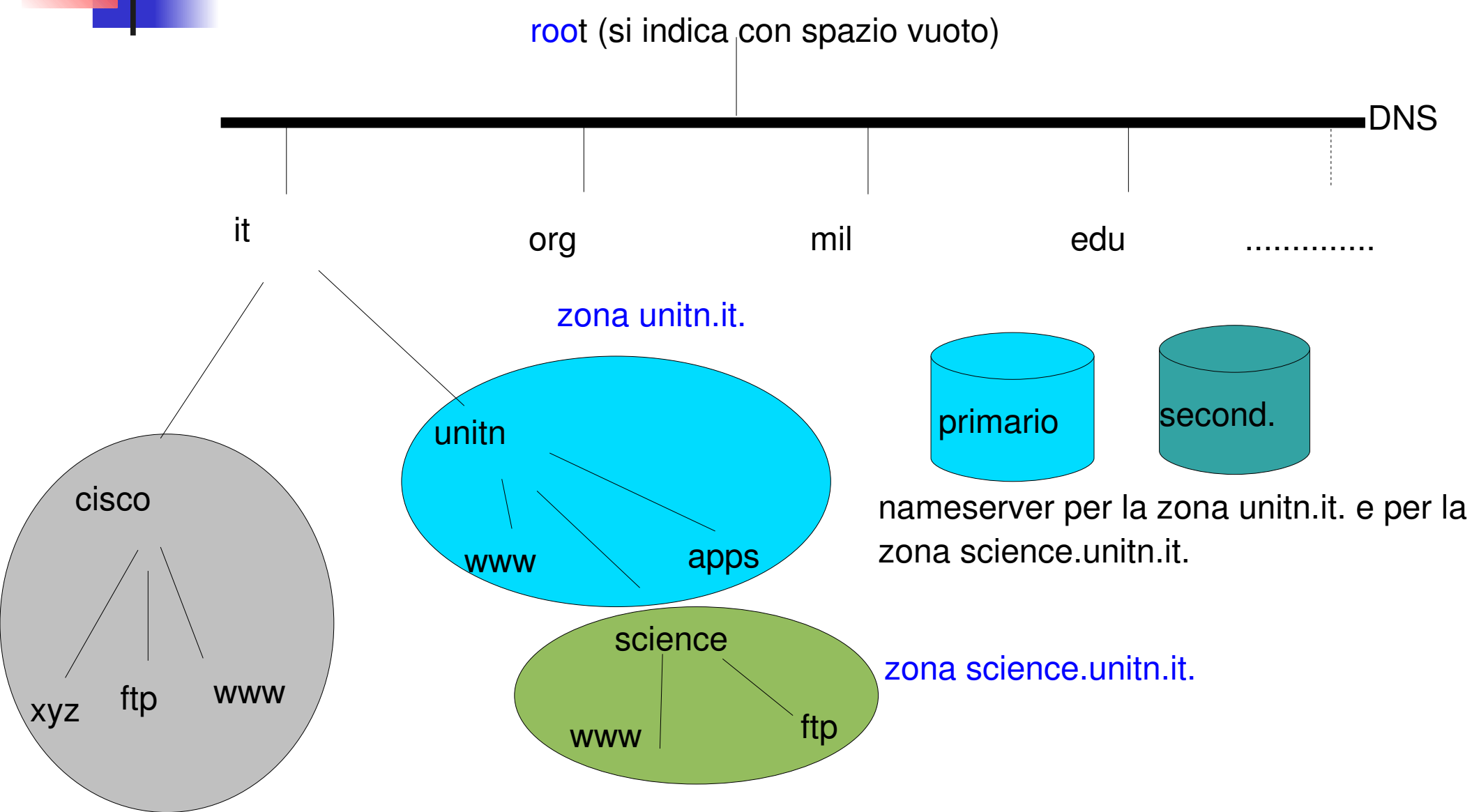
DNS


root (si indica con spazio vuoto)
DNS
it
org
mil
edu
.....
zona unitn.it. corrisponde a dominio unitn.it.
comprendente anche il sottodominio science.unitn.it.
unitn
cisco
apps
science
www
ftp
www
xyz
ftp
www
primario
second.
nameserver per la zona unitn.it.

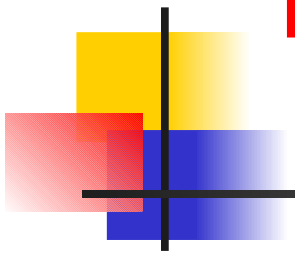
DNS: nameserver per ogni zona



DNS: si usano i nameserver del delegante



DNS

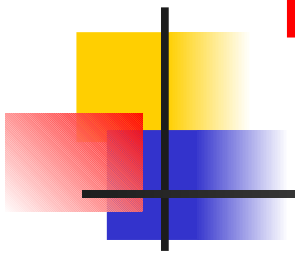


- In sintesi, mentre il dominio è un concetto che si riferisce al contesto del name space, il termine “zona” si riferisce al contesto della gerarchia delle authorities

- Una zona può coincidere o meno con un dominio; si differenzia solo quando alcuni sottodomini di un certo dominio sono delegati ad altre authorities; in questo caso il dominio è formato da tante zone quanti sono i suoi sottodomini gestiti da authorities differenti

- Ad esempio, nelle slide precedenti, il dominio unitn.it è formato dalla zona unitn.it, che comprende i nomi degli host direttamente collegati al nodo unitn.it., e dalla zona science.unitn.it. che include tutti i nomi collegati al nodo omonimo

DNS



- Si dice, in gergo tecnico, che un nameserver è autoritativo per una certa zona, ad indicare che è responsabile dei nomi di tutti gli host appartenenti a quella zona

- Il nameserver gestisce delle specifiche informazioni (RR resource record) delle quali le più importanti sono:

  - SOA (Start of Authority): ne esiste uno per zona e riporta dei parametri di gestione

  - NS (name server): indica il nome dei nameserver (primario e secondari) dove reperire le informazioni di una zona (un record per host)

  - A (address): indica l'indirizzo IP (un record per host)

DNS

Esempio di zona definita in un nameserver

```
; zone fragment for example.com

example.com      IN    SOA   ns1.example.com.     hostmaster.example.com. (
                 2003080800 ; serial number
                 2h         ; slave refresh =  2 hours
                 15M        ; slave update retry = 15 minutes
                 3W12h      ; slave expiry = 3 weeks + 12 hours
                 2h20M      ; minimum = zone default TTL
                 )
; main domain name servers
                 IN    NS    ns1.example.com.
                 IN    NS    ns2.example.com.
; main domain mail servers
                 IN    MX    mail.example.com.
; A records for name servers above
ns1.example.com.        IN    A     194.207.0.3
ns2.example.com.        IN    A     194.207.0.4
; A record for mail server above
mail.example.com.       IN    A     194.207.0.5
www.example.com.        IN    A     194.207.0.6
....
```

Nome zona

equivale a @

SOA: definisce il nameserver principale, l'emal della persona di riferimento ed i parametri temporali di gestione

NS: si definiscono i nameserver principale e secondario

MX: si definisce il mailserver del dominio

A: si definiscono gli indirizzi IP dei due nameserver

A: si definiscono gli indirizzi IP degli host del dominio

DNS

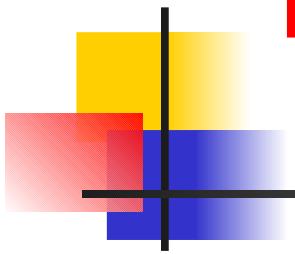
- Nel caso di zona delegata, sul nameserver del delegante dovranno essere presenti delle informazioni (glue records) che consentano di puntare, per la zona delegata, ai nameserver di competenza

- In effetti tutto il DNS si basa sul concetto di delega:

  - il nameserver di root delega ai nameserver che gestiscono i nomi dei domini di primo livello (es. i domini di root delegano la zona it. ai nameserver gestiti dal NIC di Pisa)

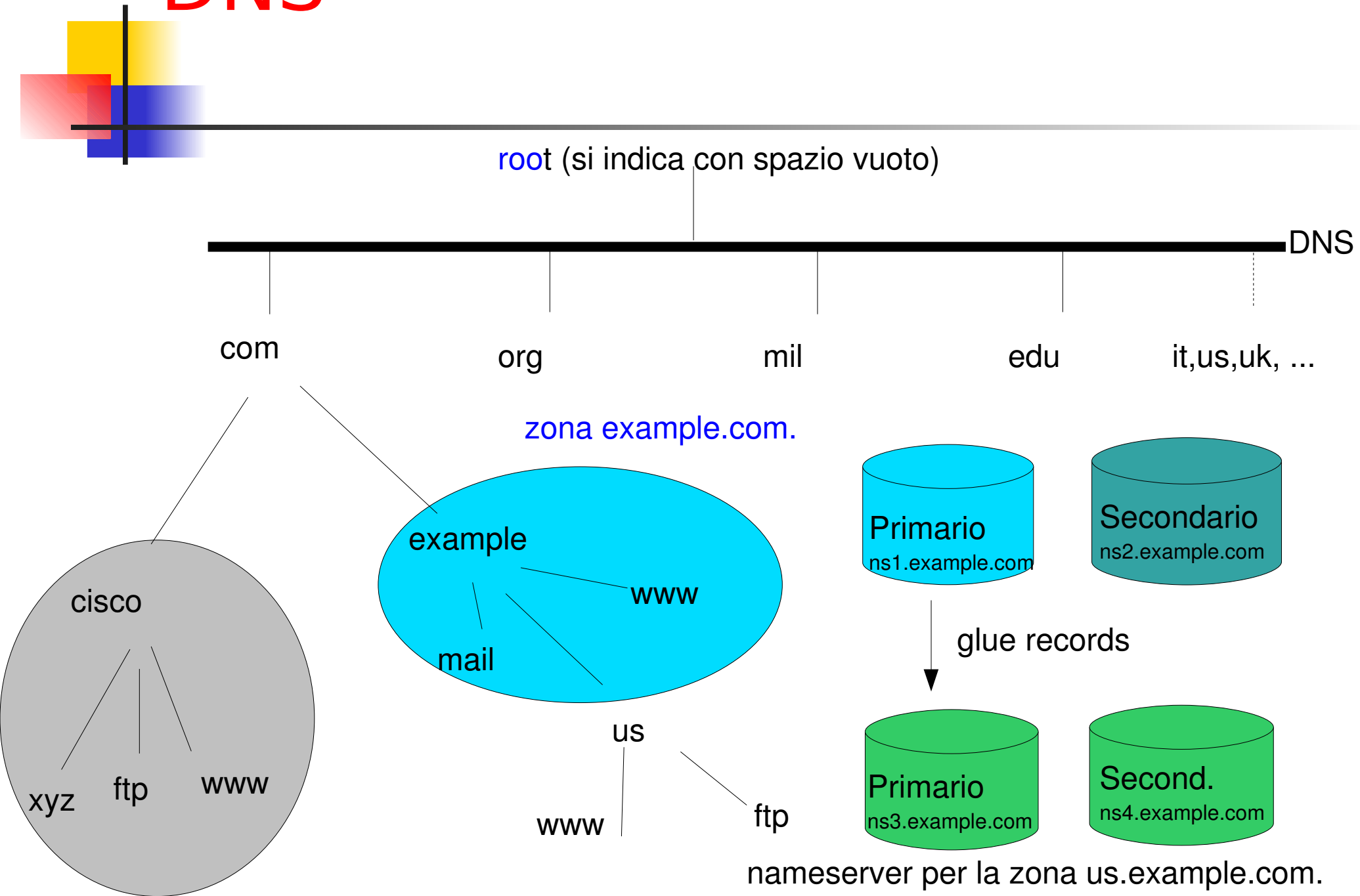
  - I nameserver che gestiscono i nomi dei domini di primo livello delegano ai nameserver che gestiscono i nomi dei domini di secondo livello e così via (es. i nameserver di it. delegano la zona unitn.it. ai nameserver dell'Ente che gestisce tale zona)

DNS



- Il nameserver del delegante deve quindi riportare gli indirizzi IP dei nameserver della zona delegata (glue records) in modo che, in fase di risoluzione degli indirizzi, si possa, scorrendo la sequenza di deleghe che porta dai nameserver di root a quelli autoritativi, accedere direttamente a questi ultimi per reperire gli indirizzi IP degli host appartenenti alla zona cercata

DNS


root (si indica con spazio vuoto)
DNS
com
org
mil
edu
it,us,uk, ...
zona example.com.
example
www
mail
cisco
xyz
ftp
www
us
www
ftp
Primario
ns1.example.com
Secondario
ns2.example.com
glue records
Primario
ns3.example.com
Second.
ns4.example.com
nameserver per la zona us.example.com.

DNS

```
; zone fragment for example.com

example.com.        IN     SOA   ns1.example.com. hostmaster.example.com. (
            2003080800 ; serial number
            2h         ; refresh =  2 hours
            15M        ; update retry = 15 minutes
            3W12h      ; expiry = 3 weeks + 12 hours
            2h20M      ; minimum = 2 hours + 20 minutes
            )
; main domain name servers
            IN     NS    ns1.example.com.
            IN     NS    ns2.example.com.
; main domain mail servers
            IN     MX     mail.example.com.
; A records for name servers above
ns1.example.com.        IN     A     194.207.0.3
ns2.example.com.        IN     A     194.207.0.4
; A record for mail server above
mail .example.com.      IN     A     194.207.0.5
....
```

Configurazione del name server autoritativo per la zona example.com.

DNS

```
; sub-domain definitions

; we define two name servers for the sub-domain
us.example.com          IN      NS      ns3.us.example.com.

; the  second name server
us.example.com.         IN      NS      ns4.us.example.com.
; sub-domain address records for name server only - glue record
ns3.us.example.com      IN      A       11.10.0.24 ; 'glue' record
ns4.us.example.com      IN      A       11.10.0.25; 'glue' record
```

Glue records che definiscono gli indirizzi IP dei nameserver che gestiscono la zona delegata us.example.com

Delega della zona us.example.com da parte del nameserver autoritativo per la zona example.com

DNS

Vantaggi

- Distribuzione del carico di lavoro e migliori performance
- Ridondanza e quindi affidabilità
- Ogni organizzazione è direttamente responsabile dei suoi dati
- Impossibilità di nomi duplicati

DNS

Modalità di risoluzione dei nomi

- Il client (ad esempio il browser) contatta il nameserver associato all'host sul quale è installato (parametro di configurazione) e chiede che venga risolto un certo nome (supponiamo www.unitn.it.) in modalità ricorsiva (ossia il nameserver dovrà effettuare tutte le ricerche necessarie restituendo al client solo il risultato finale)

- Il nameserver locale, se non ha già disponibile l'indirizzo richiesto in cache, contatta, in modalità iterativa, uno dei rootserver richiedendo la risoluzione del nome cercato

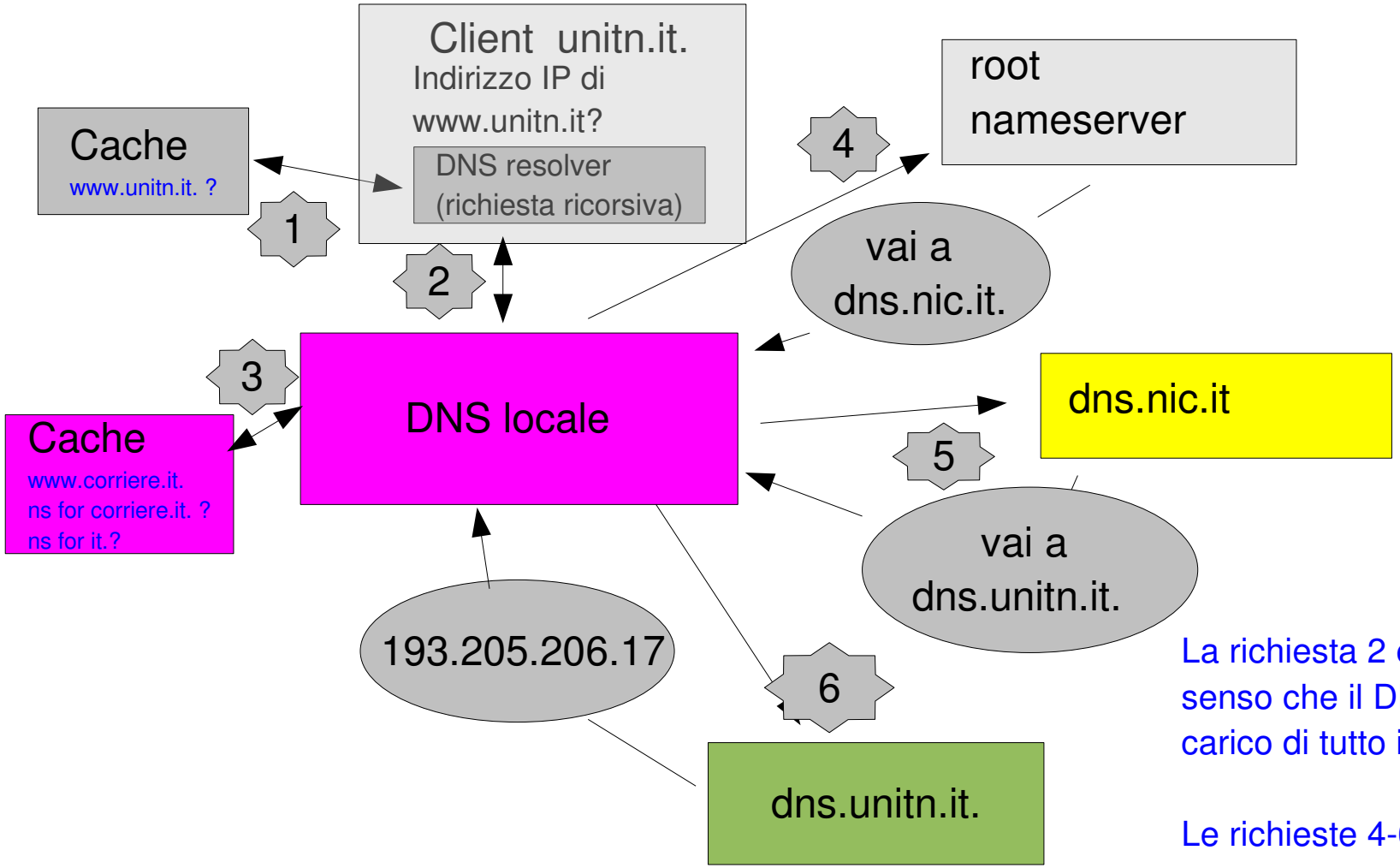
- Il rootserver risponde fornendo gli indirizzi IP dei nameserver autoritativi per il dominio di primo livello presente nel nome cercato (nel nostro caso it.)

DNS

Modalità di risoluzione dei nomi

- Il nameserver locale contatta quindi, sempre in modalità iterativa, uno di questi due nameserver che risponderà con gli indirizzi IP dei name server autoritativi per la zona unitn.it
- Il nameserver locale contatta, in modalità iterativa, uno dei due nameserver autoritativi per la zona unitn.it. ed otterrà finalmente l'indirizzo IP dell' host cercato (www.unitn.it)
- L'indirizzo così ottenuto viene restituito al client
- Parte di questo processo può essere semplificato e ridotto ricorrendo a meccanismi di caching delle informazioni
- Si noti che questa navigazione è resa possibile dai glue records

DNS



La richiesta 2 è di tipo ricorsivo, nel senso che il DNS contattato si fa carico di tutto il processo di risoluzione

Le richieste 4-6 sono iterative

DNS

Laboratorio

Verifica del processo ricorsivo di risoluzione di un nome

- Si userà il comando dig:

dig @<NomeNameServer> <nome cercato> +norec

Non effettuare ricerca in modo ricorsivo

Esempio

dig @dns.nic.it www.unitn.it +norec

DNS

Laboratorio

Esempio di configurazione ed attivazione di un nameserver

```
 sudo /etc/init.d/bind9 start
dig @192.168.20.3  www.example.com.
dig @192.168.20.3  ftp:example.com.
dig @192.168.20.3  www.example.com.   +norec
```

DNS

TLD (Top Level Domains)

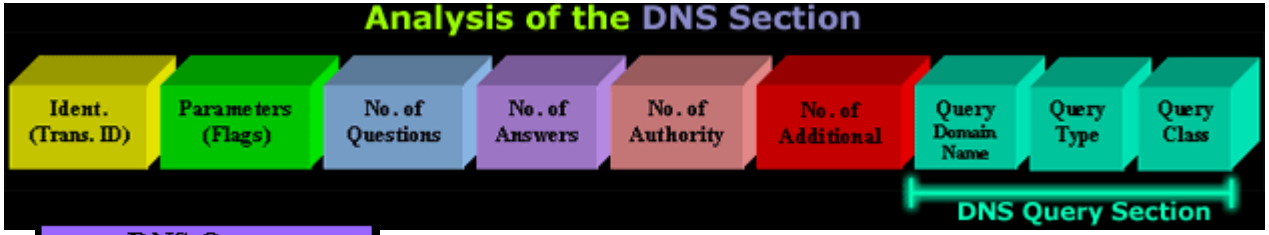
- com (commercial)
- edu (educationa)
- gov (government; es. nasa.gov)
- mil (military)
- net (network infrastructure)
- org (not commercial organizatios)
- int (international)
- arpa (usato per reverse mapping)
- country code (it,us,uk, de etc). ISO 3166
- new (es. aero, biz, coop, info, museum, eu etc)

DNS

I 13 Root Servers

1)	A.ROOT-SERVERS.NET.	198.41.0.4
2)	B.ROOT-SERVERS.NET.	192.228.79.201
3)	C.ROOT-SERVERS.NET.	192.33.4.12
4)	D.ROOT-SERVERS.NET.	128.8.10.90
5)	E.ROOT-SERVERS.NET.	192.203.230.10
6)	F.ROOT-SERVERS.NET.	192.5.5.241
7)	G.ROOT-SERVERS.NET.	192.112.36.4
8)	H.ROOT-SERVERS.NET.	128.63.2.53
9)	I.ROOT-SERVERS.NET.	192.36.148.17
10)	J.ROOT-SERVERS.NET.	192.58.128.30
11)	K.ROOT-SERVERS.NET.	193.0.14.129
12)	L.ROOT-SERVERS.NET.	198.32.64.12
13)	M.ROOT-SERVERS.NET.	202.12.27.33

Protocollo DNS (query)



DNS Query

2 Trans. ID: The client uses this field to match responses to queries.

2 Parameters: Specifies the operation requested and a response code.

2 Number of Questions: Count of entries that appear in the *Question Section*.

2 Number of Answers: Count of entries that appear in the *Answer Section*. (Always set to zero in a Query)

2 Number of Authority: Count of entries that appear in the *Authority Section*. (Always set to zero in a Query)

2 Number of Additional: Count of entries that appear in the *Additional Section*. (Always set to zero in a Query)

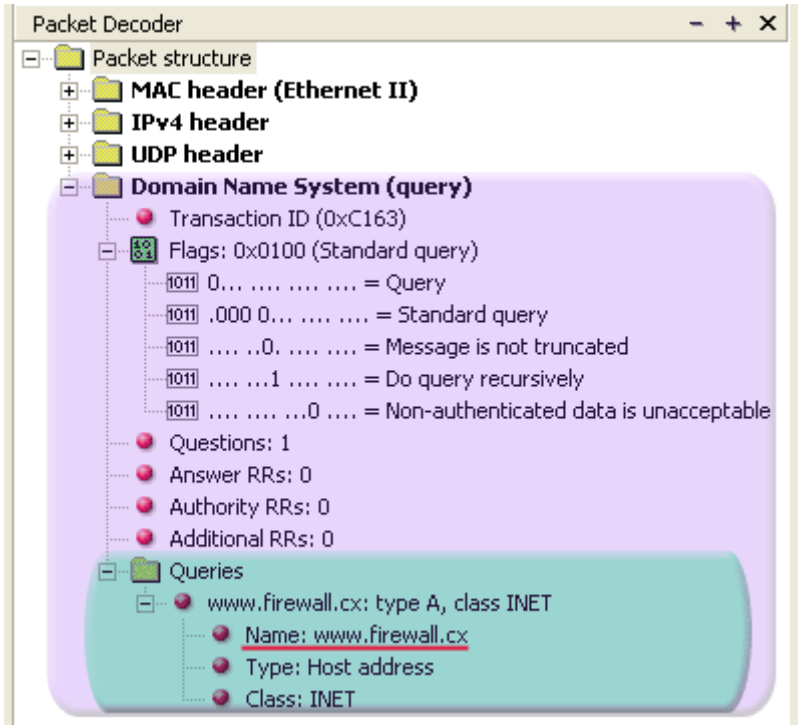
DNS Query Section

Query Domain Name: The domain for which the query is sent. This field has a variable length.

2 Query Type: Encodes the type of question. e.g whether the questions refers to a host address (A type) or mail address (MX type).

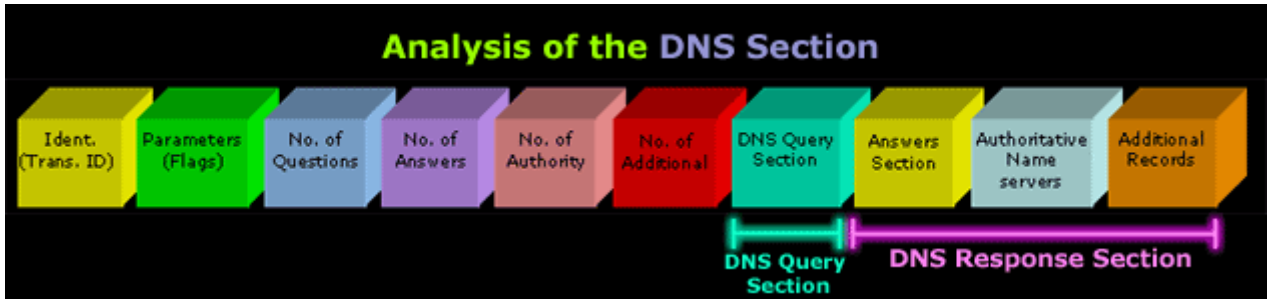
2 Query Class: Allows domain names to be used for arbitrary objects.

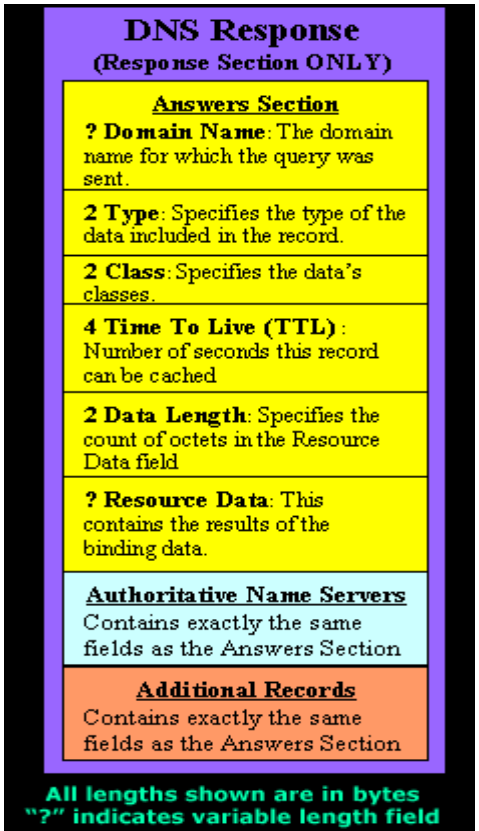
All lengths shown are in bytes

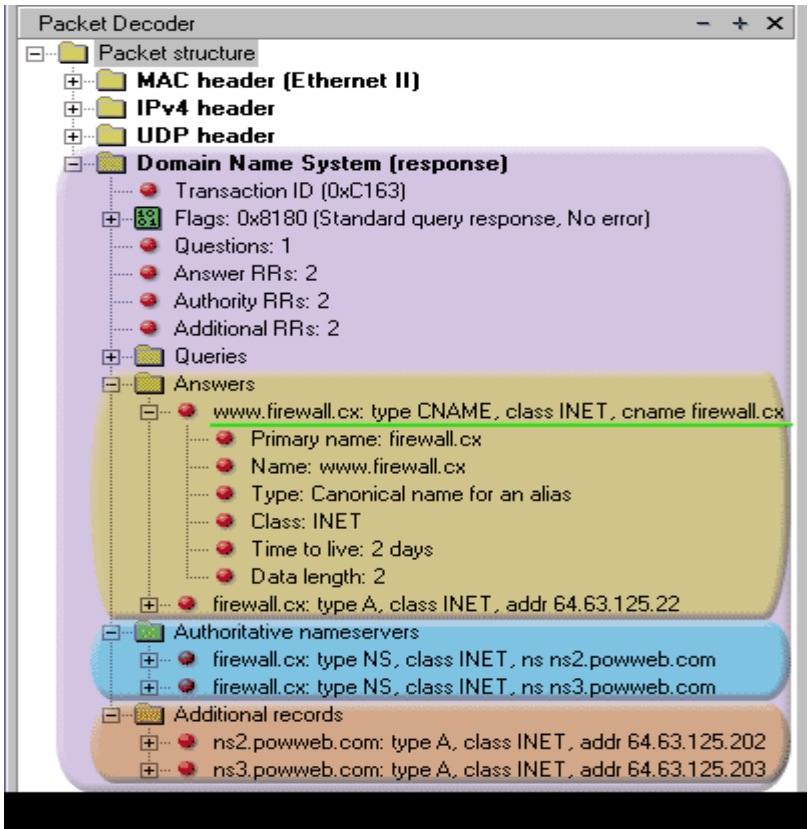


Protocollo DNS (answer)

http://www.firewall.cx/dns-query-format.php







Procotocollo SMTP

- Breve storia
- Componenti architetturali
- Principi di funzionamento
- Formato del messaggio
- Protocollo POP3

Posta elettronica

Breve storia

- Diffusa inizialmente (1975) a livello accademico e concepita come un semplice servizio di invio di un file testuale nella directory (mailbox) del destinatario, è ora l'applicazione più utilizzata su Internet
- Crescita esponenziale a partire dagli anni 90, grazie anche alla diffusione di client ad interfaccia grafica
- Si basa su un protocollo client-server standard (Simple Mail Transfer Protocol), molto semplice e costituito da pochi comandi
- La semplicità comporta però effetti collaterali negativi (es. posta indesiderata o spamming, informazioni inviate in chiaro, mancanza di un meccanismo di autenticazione etc)

Posta elettronica

Struttura indirizzo di posta

- <user name>@<domain name>
- Esempio: mario.rossi@example.com
- case insensitive

Nota bene:

Nell'indirizzo di posta compare solo il dominio dell'utente e non il nome del mailserver del dominio, in quanto è il DNS che consente, grazie al record di tipo MX, di associare, al nome del dominio, il nome del/dei server di posta

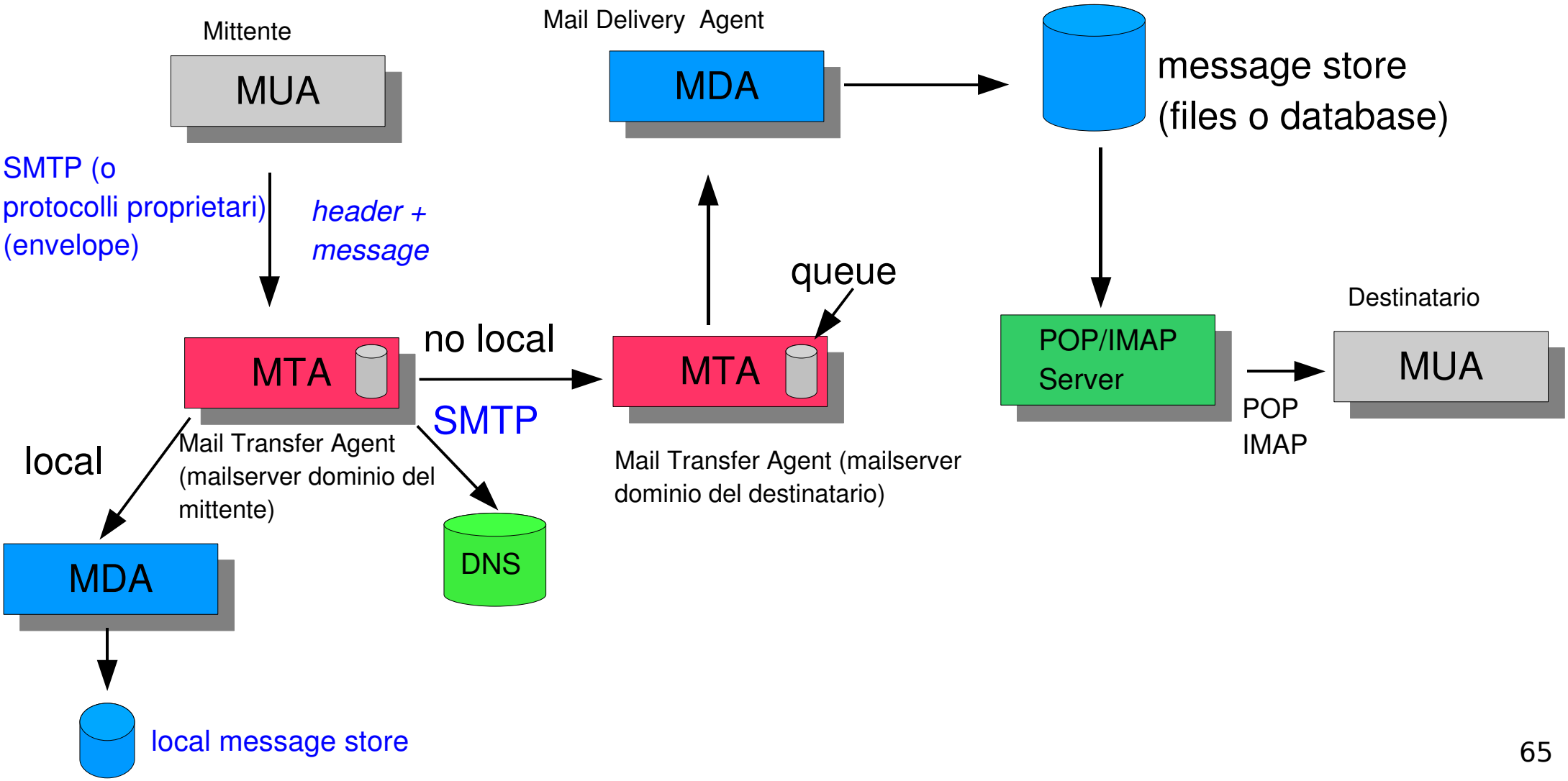
Posta elettronica

```
example.com. IN SOA ns.example.com. kdent.example.com. (
    1049310513
    10800
    3600
    604800
    900 )
example.com.   IN NS ns.example.com.
server1.example.com.  IN A 192.168.100.220
ns.example.com.      IN A 192.168.100.5
mail1.example.com.    IN A 192.168.100.50
mail2.example.com.    IN A 192.168.100.54
mail3.example.com.    IN A 192.168.100.123
;
; Mail Exchangers
example.com.    IN MX   10    mail1.example.com.
example.com.    IN MX   20    mail2.example.com.
example.com.    IN MX   30    mail3.example.com.
;
; CNAME Records
;
pop.example.com.     IN CNAME mail1.example.com.
www.example.com.     IN CNAME server1.example.com.
```

Definizione dei mail server nel file di configurazione del name server

preference number (indica la priorità di scelta del server smtp)

Componenti Architetturali



Protocollo SMTP

Mail User Agent (client di posta)

- E' il client che consente di gestire i propri messaggi di posta
- Esempi di user agent diffusi: MS Outlook, IBM Lotus Notes, Eudora, Evolution. Ora sempre più sostituiti da client web
- Per ora considereremo solo gli aspetti legati all'invio di email; la ricezione sarà analizzata più avanti
- Alla richiesta di send da parte dell'utente, MUA si collega ad MTA del dominio ed invia l'email usando il protocollo SMTP

Protocollo SMTP

MTA Mail Transfer Agent

- L' MTA è il processo software che gestisce l'invio e la ricezione dei messaggi di posta
- L' MTA del dominio mittente è server nei confronti dell'user agent e client nei confronti del/i mailserver destinatario/i
- Lo scambio di informazioni fra MTA avviene con protocollo SMTP
- I termini MTA e mailserver sono usati come sinonimi
- Esempi: MS Exchange, IBM Lotus Domino, Sendmail, Postfix etc

Protocollo SMTP

MDA Mail Delivery Agent

- E' la componente software che riceve da MTA il messaggio di posta e lo memorizza nello spazio (message store) dove sono configurate le caselle degli utenti
- Le caselle degli utenti possono avere forma di file oppure di database relazionale
- Dispone normalmente di funzioni di controllo del contenuto (content filtering, antivirus etc)
- Alcuni prodotti (es. Postfix) includono entrambe le componenti MDA ed MTA, ma quest'ultima può, se necessario, utilizzare un MDA indipendente

Protocollo SMTP

POP/IMAP Server

- E' la componente software che interagisce con i client di posta degli utenti
- Accede al message store e trasmette ai client le email contenute nelle caselle di posta
- Si basa su protocolli alternativi:
  - POP3
  - IMAP

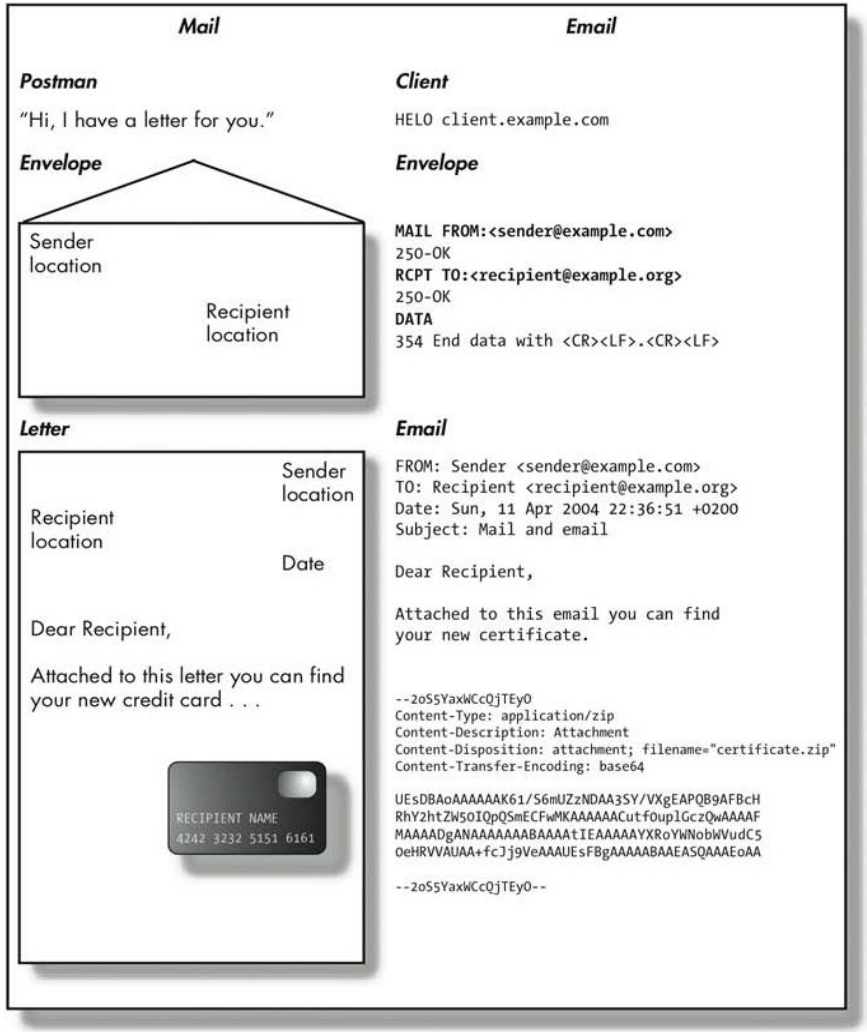
Posta elettronica

Struttura del messaggio di posta elettronica

- Un'email è costituita da:
  - Envelope: serie di comandi, in protocollo SMTP, scambiati fra MUA ed MTA e fra i vari MTA per la corretta consegna del messaggio.
    - Helo: comando che segnala l'inizio dell'envelope
    - Mail From, Rcpt To: identificano essenzialmente gli indirizzi di email del mittente (envelope sender) e dei destinatari(envelope recipient). Tali indirizzi possono coincidere o meno con quelle presenti nell'header. Ad esempio i vari BCC sono presenti in envelope ma non nell'header
    - Data: riporta il testo dell'email da inviare (header + body)
  - Header: metadati del messaggio (vedi slides seguenti)
  - Body: messaggio vero e proprio ed allegati (attachments)

Posta elettronica

Struttura del messaggio di posta elettronica



Protocollo SMTP

Email header

Header Field	Description
`Received:`	Every email server through which the message passes adds this header field to the beginning of an email message. Thus, all the `Received` header lines create a block of header fields on top of the message header. Therefore, if we read this block from bottom to top, we will discover the whole route, and through which email servers the message has passed. This header field may contain words such as: `from` (sending server). `by` (receiving server). `via` (physical path). `with` (network or mail protocol). `id` (receiver message identification). `for` (for whom the message is intended). For example, if the recipient is set as a distribution list, then the original recipient will be maintained, i.e., the distribution list).
`From:`	The sender who sends the email.
`Sender:`	Handled by, e.g.., secretary. This field contains, for example, information about a conference through which the message was received.
`Date:`	Posting date (date, time, and time zone).
`Reply to:`	To whom you send your replies.
`In-Reply-To:`	Replying to your message. Identification of the previous correspondence to which this message is an answer.
`To:`	Recipients (primary recipients of the message).
`Cc:`	Carbon copy (secondary recipients of the message).

Protocollo SMTP

Email header

Header Field	Description
`Bcc:`	Blind carbon copy (additional recipients of the message). This header field is erased before sending.
`Message Id:`	Message identification (unique identifier that refers to this version of a particular message).
`Keywords:`	Keywords or phrases describing the content separated by commas.
`References:`	Identify other correspondence which this message references.
`Subject:`	Subject (short summary of the message content).
`Comments:`	Comments about the message.
`Encrypted:`	Encryption (obsolete).
`X-:`	All header fields beginning with the string `X-` are user defined (here the word *user* means software developer, not software user). For example, X-Mailer is often used to identify the software used by the sender of the message.
`Resent:`	When automatic message forwarding is used (for example, return of an undelivered message by an intermediate mail server), the `Resent-` string will be add at the front of the original header field (for example, `Resent-From` or `Resent-CC`. etc.).

Ogni messaggio vero e proprio è preceduto da un header riportante metadati di vario tipo, utilizzati da MUA mittente per invio ed MUA destinatario per presentazione e ricerca

Protocollo SMTP

Email header

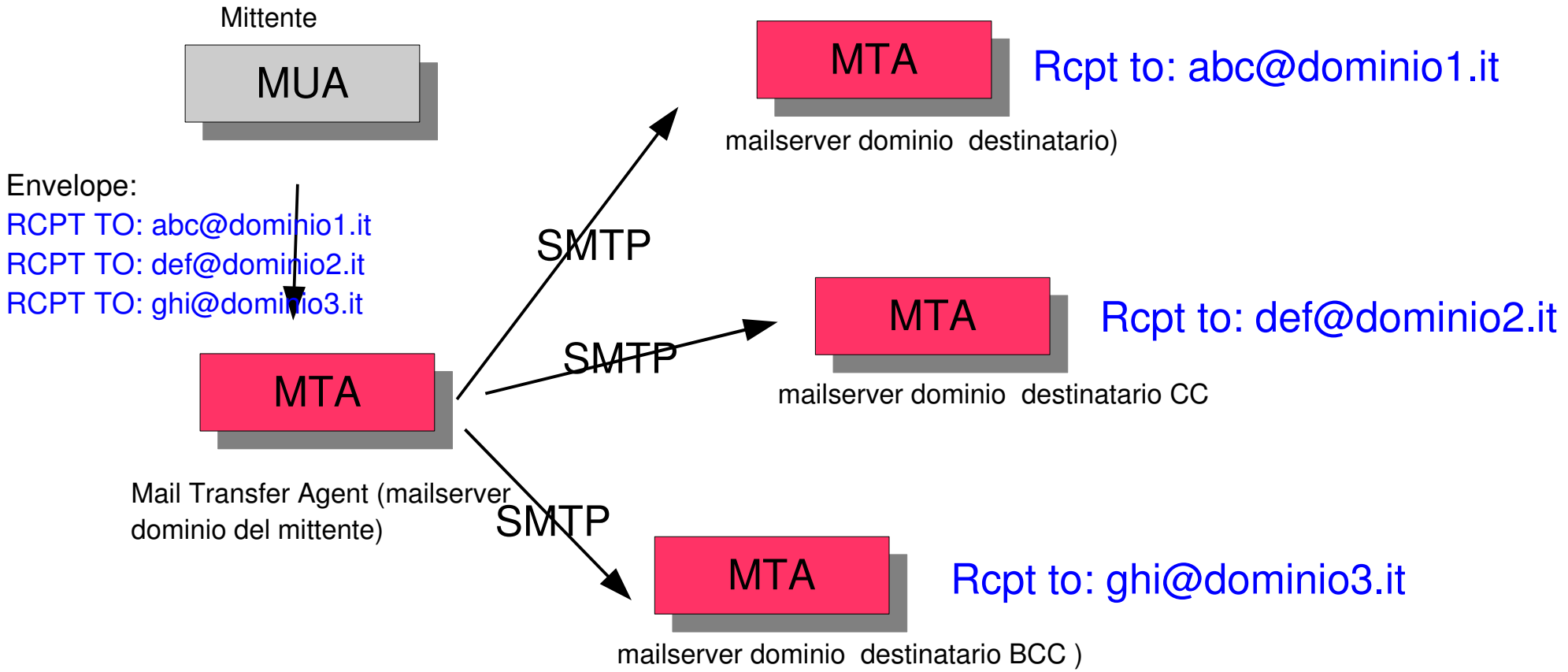
```
Received: from amphissa.erlm.siemens.de ([146.254.164.8])
          by cz.siemens.net
          with Microsoft SMTPSVC;
          Thu, 29 Dec 2005 08:14:58 +0100
Received: from tegea.erlm.siemens.de
          by amphissa.erlm.siemens.de
          with ESMTP
          id 17862158526
          for <libor.dostalek@siemens.com>;
          Thu, 29 Dec 2005 08:14:58 +0100 (CET)
Received: from zetes.siemens.com (zetes.siemens.com [217.194.34.75])
          by tegea.erlm.siemens.de
          with ESMTP
          id CA10D1774B8
          for <libor.dostalek@siemens.com>;
          Thu, 29 Dec 2005 08:14:56 +0100 (CET)
Received: from imap.packtpub.com (unknown [217.207.125.60])
          by zetes.siemens.com (Postfix) with ESMTP
          for <libor.dostalek@siemens.com>;
          Thu, 29 Dec 2005 08:14:55 +0100 (CET)
Received: from paramita (unknown [203.122.53.88])
          by imap.packtpub.com (Postfix)
          with ESMTP
          id B6D24970B36
          for <libor.dostalek@siemens.com>;
          Thu, 29 Dec 2005 07:22:12 +0000 (GMT)
From: "Abhishek" <abhisheks@packtpub.com>
To: "'Dostalek Libor'" <libor.dostalek@siemens.com>
Subject: RE: TCP/IP DNS_Chapter 14
Date: Thu, 29 Dec 2005 12:44:44 +0530
Message-ID: <000001c60c47$8ce00220$0d00a8c0@paramita>
X-Mailer: Microsoft Office Outlook 11
Return-Path: abhisheks@packtpub.com

Message text
```

l'header viene modificato nei vari passaggi fra MTA, in modo da disporre del tracciamento del percorso effettuato dall'email

Componenti Architetturali

Un messaggio corrisponde ad n invii

TO: abc@dominio1.it
CC: def@dominio2.it
BCC: ghi@dominio3.it



Protocollo SMTP

Un messaggio corrisponde ad n invii

- L'utente specifica al client il mittente (FROM) ed i destinatari del messaggio
- Il client consente di indicare vari destinatari:
  - diretti (TO)
  - CC (Carbon Copy: per conoscenza)
  - BCC (Blind Carbon Copy)
- L'indirizzo del mittente (FROM) origina un comando SMTP di tipo MAIL FROM

Protocollo SMTP

Un messaggio corrisponde ad n invii

- Gli indirizzi dei destinatari (TO, CC, BCC) originano un comando SMTP di tipo RCPT TO, uno per ogni destinatario
- Infine con il comando SMTP DATA il client invia header e body del messaggio. Nell'header sono riportati i destinatari diretti (TO) ed in CC.
- Una prima classificazione dei comandi client e dei corrispondenti comandi SMTP

Comando client	Comando SMTP	Note
FROM:	MAIL FROM	Unico comando; riportato anche in header
TO, CC, BCC	RCPT TO	Un comando RCPT per ogni destinatario; TO e CC riportati in header
Subject	DATA	Header messaggio
Altri parametri	DATA	Header messaggio
Messaggio	DATA	Body messaggio

Protocollo SMTP

SMTP protocol: passi fondamentali

1) Il client MUA risolve, via DNS, l'indirizzo IP del mailserver del dominio del mittente (parametro di configurazione del client)

2) Il client si connette alla porta TCP 25 del mailserver, che risponde con un messaggio 220 <ready>

3) Il client segnala l'inizio dell'evelope con un comando HELO, seguito opzionalmente dal proprio nome completo di dominio (FQDN). Il server risponde con 250 <OK>

4) Il client specifica il mittente (envelope sender) con MAIL FROM: <indirizzo>; il server risponde con: 250 <OK>

Protocollo SMTP

SMTP protocol: passi fondamentali

5) Il client precisa l'indirizzo/i del destinatario/i (envelope recipients) con RCPT TO:<indirizzo>, la risposta è ancora 250 <OK>

6) Il client dichiara di essere pronto a trasmettere il vero messaggio con: DATA Il server risponde con: “354 End data with <CR><LF>.<CR><LF>”

7) Il client inserisce l'header ed il body del messaggio e termina con una riga contenente il solo carattere “.”

8) Il client chiude la sessione con il comando QUIT

Protocollo SMTP

SMTP protocol: passi fondamentali

9) Il processo continua in modo analogo fra MTA rispettivamente dei domini mittente e destinatario.

10) Una significativa differenza fra MUA e MTA è data dal fatto che, a fronte dell'unica connessione da MUA contenente n richieste RCPT TO ricevute dal client, MTA mittente innesca n connessioni con i vari mailserver dei domini destinatari (relay)

11) Inoltre MTA, nel processo di relay, lascia inalterati header e body del messaggio, ad eccezione del campo Received che indica i vari mailserver attraverso i quali il messaggio è transitato

Protocollo SMTP

Laboratorio

- Simulazione della connessione MUA/MTA, via SMTP, con invio di un'email ad un server locale sendmail, usando telnet
  - sudo /etc/init.d/sendmail start
  - netstat -tan (per controllare che la porta 25 sia attiva)
  - telnet localhost 25
  - invio comandi smtp helo, mail from, rcpt to, data etc
  - in data specificare header + body

- Analisi con Ethereal (laboratorio successivo)

Protocollo SMTP

Setcode e gestione errori

- La comunicazione avviene con setcode NVT ASCII (7 bits per carattere)
- Qualora MTA mittente non sia in grado di comunicare con quello/quelli destinatario, il messaggio da inviare viene tenuto in coda e si ritenta la trasmissione dopo un certo numero di secondi (es. 1000) e per un certo numero di giorni (es. 5)

Protocollo SMTP

Protocollo POP3

- Ogni mailserver riceve la posta destinata agli utenti del proprio dominio e la deposita nelle relative caselle
- E' stato standardizzato un protocollo (POP3) che consente agli user agent (client di posta) di accedere alle caselle di posta e di effettuare il download delle relative email
- Il servizio è di norma disponibile sulla porta 110 del mailserver

Protocollo POP3

Protocollo POP3

- Effettuata l'autenticazione il client può usare i seguenti comandi:
  - STAT permette di conoscere il numero di email presenti nella casella utente e lo spazio occupato
  - RETR <message number>: consente il download l'email avente il numero specificato
  - DELE <message number>: cancellazione
  - TOP <numero di email> <numero di righe> : consente di visualizzare le righe specificate nel secondo parametro per il numero di email specificato nel primo
- Esiste un protocollo alternativo (IMAP) che consente di mantenere le proprie email sul server e quindi poterle visualizzare da più postazioni di lavoro

Protocollo Http

Tim Berners-Lee

- Laurea in Physics – Oxford University
- Ricercatore al Cern di Ginevra
- Approfondisce l'uso degli ipertesti, ossia di link fra documenti elettronici
- Progetta un metodo per ottenere documenti fra loro collegati, fisicamente localizzati su computer diversi connessi via Internet
- Progetta il protocollo HTTP ed il linguaggio HTML
- Scrive il primo Web Server, denominato "httpd" nel 1989

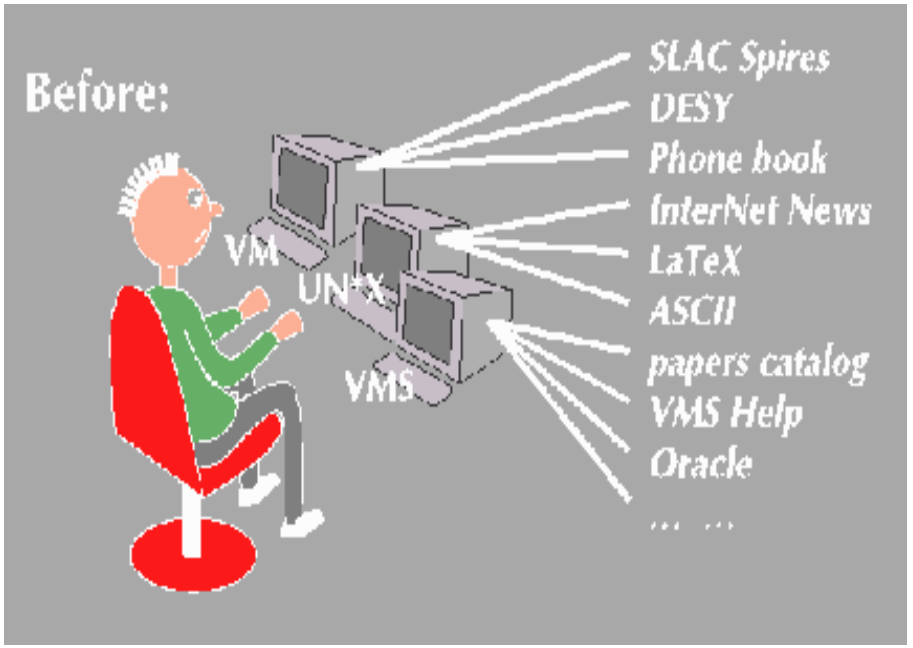
Protocollo Http

- Progetta e realizza il primo client, chiamato "WorldWideWeb“ (1991). Il client riceve dal server un documento in un ben preciso formato (HTML) e lo visualizza in forma di testo, immagini e link
- Diversi client sperimentali (viola,cello etc.)
- In 1993 Marc Andreessen di NCSA vede Viola e scrive Mosaic in ambiente X-Windows; successive versioni per PC e MAC
- Idea, allora rivoluzionaria, di distribuire Mosaic gratuitamente
- Da Mosaic nasce Netscape.....

Protocollo Http

La Babele dei sistemi

To find some information at CERN, one had to have one of a **number of different terminals** connected to a **number of different computers**, and one had to learn a **number of different programs** to access that data



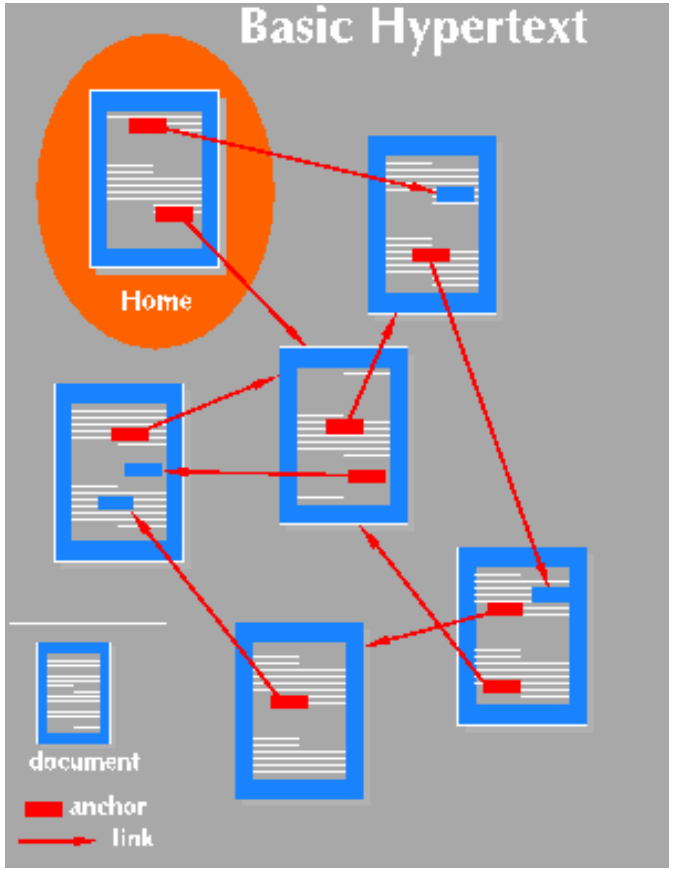
Protocollo Http

La Babele dei sistemi

Once information is available, it should be accessible from any type of computer, in any country, and an (authorized) person should only have to use one simple program to access it

Protocollo Http

Hypertext is text with links

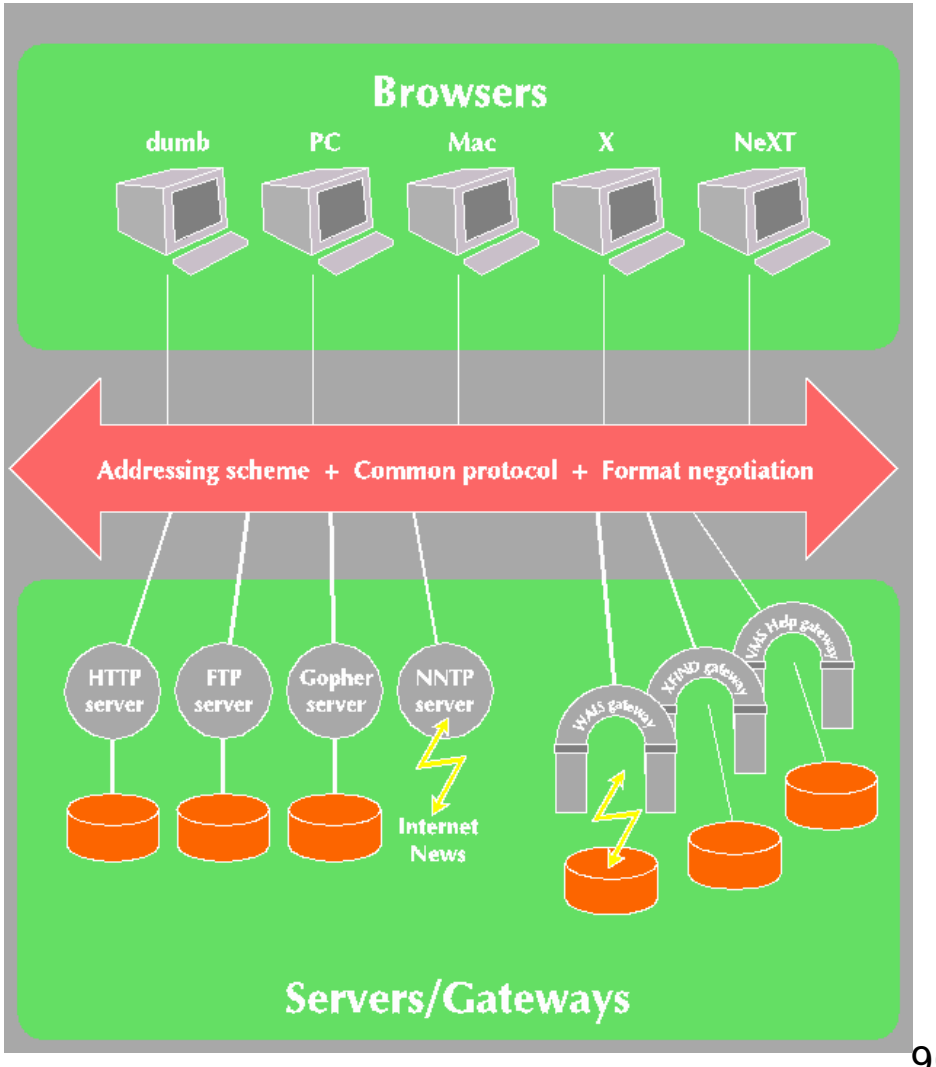


Hypertext is not a new idea: in fact, when you read a book there are links between index and the text.
If you include bibliographies which refer to other books and papers, text is in fact already full of references.

Protocollo Http

Architettura client-server

- To allow the web to scale, it was designed without any centralized facility.
- Anyone can publish information, and anyone (authorized) can read it
- There is no central control.
- To publish data you run a server, and to read data you run a client.
- All the clients and all the servers are connected to each other by the Internet. The W3 protocols and other standard protocols allow all clients to communicate with all servers.



Protocollo Http

URL (Uniform Resource Locator; termine più generico URI)

- È un **indirizzo** (locator) che permette di identificare in modo **standard** (uniform) ed **univoco** una **risorsa** (**resource**) sul web
- Per risorsa si intende generalmente **un file** (ad esempio una pagina statica html, oppure una pagina dinamica asp oppure jsp, un file pdf etc.)
- La risorsa si trova su di un server. Il browser chiede al server tale risorsa (specificando appunto il suo URL) ed ottiene in risposta il documento (**generalmente una pagina HTML**) corrispondente all'URL specificato

Protocollo Http

URL

- L' URL ha un formato standard ed è composto da tre elementi:

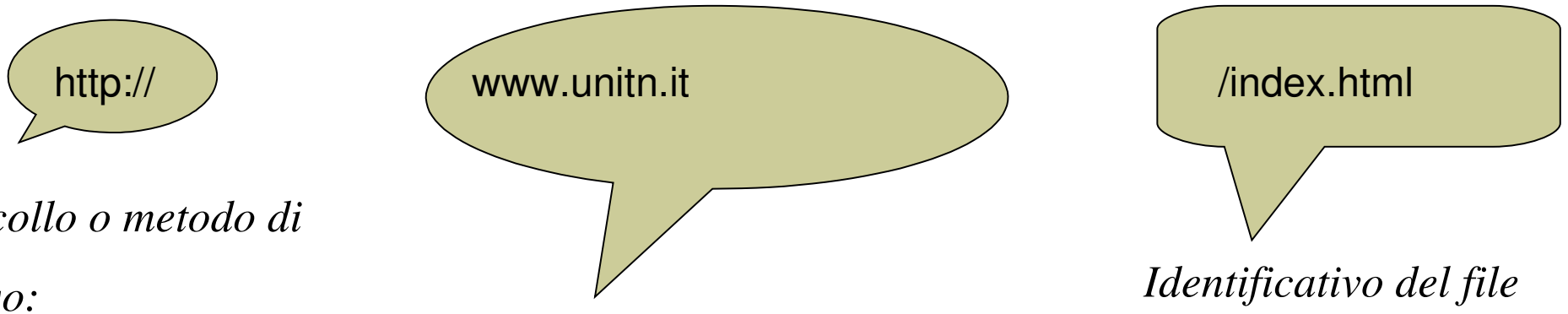
  protocollo **ServerName** **Filepath**

- Esempio http://www.unitn.it/index.html

- Si può quindi dire che l'URL è un' **estensione del file name**, nel senso che include il nome DNS del server dove il file si trova ed il protocollo di accesso

Protocollo Http

URL (Uniform Resource Locator)



Protocollo o metodo di accesso:
Indica la sequenza di comandi che verranno utilizzati nel colloquio client-server

Indirizzo del server

Indica il nome univoco con il quale il server è conosciuto nel Web.
Viene decodificato dai DNS nel relativo indirizzo IP ufficiale (es 193.205.206.17)

Identificativo del file nell'ambito del server (indica il posizionamento della risorsa cercata rispetto alla directory di base del Web Server)

Protocollo Http

HTML (HyperText Markup Language)

- È il linguaggio con il quale viene rappresentato il documento inviato dal Web server al client
- Consente di indicare al browser il contenuto di un documento, in forma di testo + riferimenti ad immagini + link
- Consente inoltre di indicare al browser le modalità di visualizzazione del contenuto di un documento

Protocollo Http

HTML (HyperText Markup Language)

- Annidati nel contenuto, sono infatti presenti dei tag ossia delle parole riservate comprese fra i simboli <> che indicano al browser come rappresentare il contenuto
- I tag esprimono comandi di rappresentazione ad alto livello; ogni browser traduce opportunamente (ed anche arbitrariamente) questi comandi
- Visualizzare esempio (simple.html)

Protocollo Http

Esempio HTML

```
<HTML>
<HEAD>
<TITLE>Esempio di documento HTML</TITLE>
<HEAD>
<BODY>
<H1>Titolo di livello 1</H1>
<P>Gli elementi di un documento HTML </P>
<H2>Titolo di livello 2</H2>
<P>Questa è una lista di elementi ordinati</P>
<OL>
  <LI>Primo elemento</LI>
  <LI>Secondo Elemento</LI>
</OL>
</BODY>
</HTML>
```

Protocollo Http

HTTP (HyperText Transport Protocol)

- Protocollo di livello 7 (applicativo)
- Nasce per l'interscambio di risorse fra un client ed un server
- Per risorsa si intende un qualsiasi gruppo di informazioni associate ad un URL
- Si tratta di norma di un documento HTML, statico o dinamico, ma può essere anche un file di qualsiasi tipo ad es. PDF, RA, MP3 etc

Protocollo Http

- Quando il client (browser) chiede al Web server una risorsa, specifica nell'URL il protocollo
- Nel caso che questo corrisponda ad Http, il browser traduce la richiesta in una sequenza di comandi, che vengono inviati al server attraverso una socket basata su protocollo TCP
- La porta well-known del server assume di norma il valore 80

Protocollo Http

Struttura request/response dei comandi Http

- Linea iniziale, differente per richiesta/risposta
- Una serie di parametri ognuno rappresentato nella seguente forma:

  <nome parametro>: <valore parametro><CR><LF>

  es. Accept_Language: en-us,en;q=0.5
- Una linea vuota contenente i caratteri CarriageReturn+LineFeed
- Un corpo del messaggio (body) opzionale e contenente il messaggio da trasmettere (es. documento html o file)

Protocollo Http

Linea iniziale request

<metodo> <path e nome della risorsa> <http version>

- GET /path/to/file/index.html HTTP/1.0
  - Chiede al server l'invio di una certa risorsa presente nella directory locale del server al path specificato
- HEAD /path/to/file/index.html HTTP/1.0
  - Analoga al comando GET ma prevede solo l'invio della risposta da parte del server e non della risorsa richiesta

Protocollo Http

Linea iniziale request

- POST /path/to/file/index.html <http version>
  - Si usa al posto di GET quando occorre inviare un blocco di dati al server (ad esempio dei parametri specifici di una richiesta senza che essi siano visti in modo esplicito a livello di URL)
  - I dati sono trasmessi nel body
  - Di norma vi sono due header (ContentType e ContentLength) ad indicare il tipo e la lunghezza di dati

Protocollo Http

Linea iniziale request

- Esempio di POST:

POST /path/script.cgi HTTP/1.0
From: frog@jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies

header

linea di separazione header-body

body con dati da trasmettere al server

Protocollo Http

- Il server, ricevuta la richiesta HTTP del client, individua il documento cercato sul suo file system ed invia la risposta sulla socket di connessione al client

- Tale risposta è rappresentata da:
  - una linea iniziale di stato,
  - una sequenza di informazioni di controllo, nella forma:

    <nome parametro>:<valore parametro>

    terminate da una linea vuota (solo CR e LF)
  - il documento richiesto in formato HTML

Protocollo Http

Linea iniziale response

- La linea iniziale della risposta indica:
  - la versione HTTP usata dal server
  - lo status ossia un codice che può assumere diversi valori quale ad esempio:
    - HTTP/1.1 200 OK
    - HTTP/1.1 304 not found
    - HTTP/1.1 504 server error
    - etc etc
- A seguire sono presenti parametri informativi del server

Protocollo Http

Parametri informativi del server

- Esempio di risposta:
  - *HTTP/1.1 200 OK*
  - *Connection: close*
  - *Date: Thu, 06 Aug 1998 12:00:05 GMT*
  - *Server: Apache/1.3.0 (Unix)*
  - *Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT*
  - *Content-Length:6821*
  - *Content-Type: text/html*

Protocollo Http

Laboratorio: analisi della risposta da parte del server HTTP

- sudo /etc/init.d/apache2 start
- netstat -tanp
- copia di un file html di esempio in /var/www
- test di visualizzazione con Ethereal (laboratorio successivo)

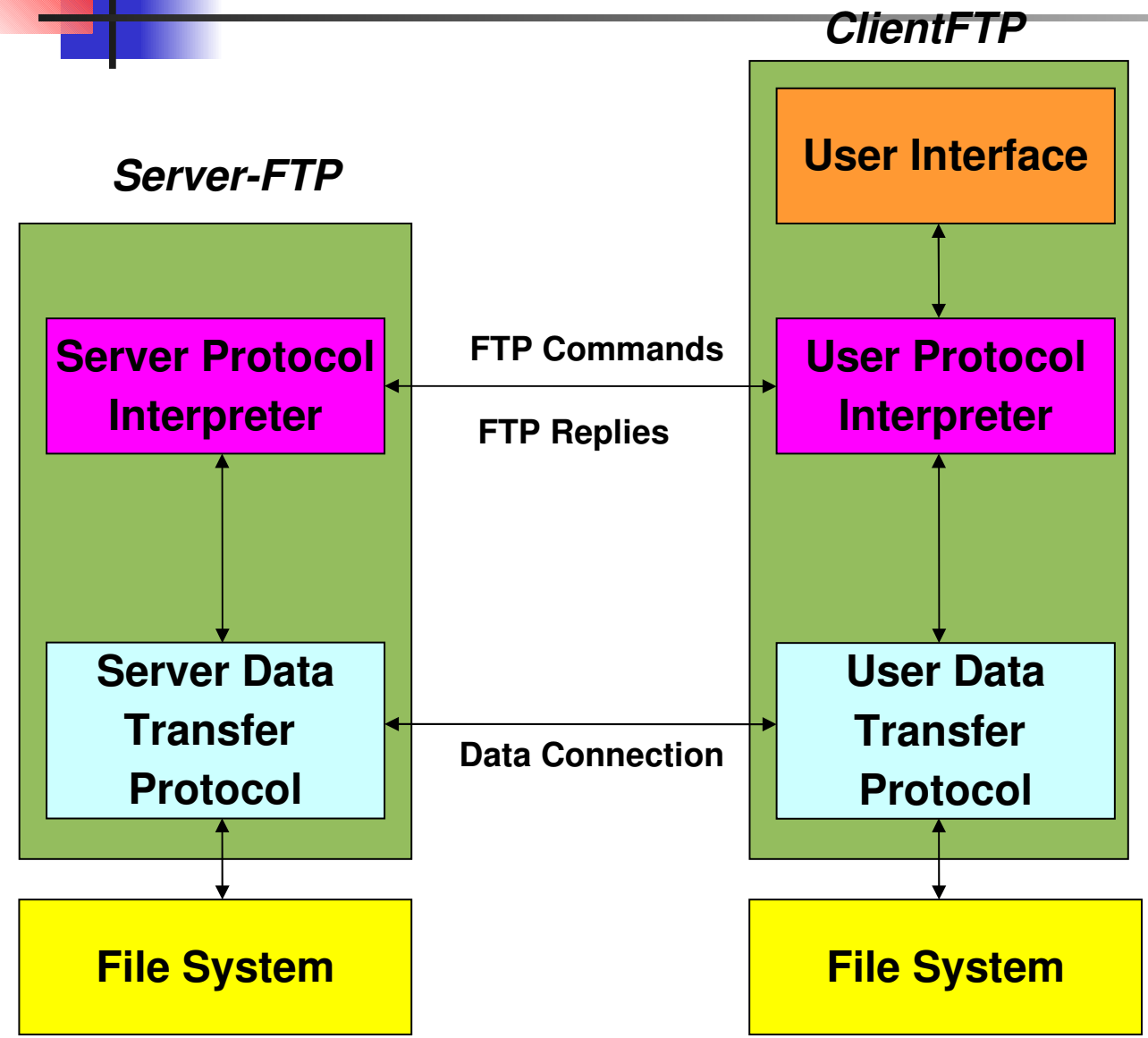
Protocollo Http

- HTTP è un protocollo stateless nel senso che ogni connessione è indipendente dalle altre
- Qualora sia necessario che il server sia in grado di associare una richiesta ad altre, inviate in precedenza, si ricorre alla tecnica del cookie
- Il cookie non è altro che un numero di riconoscimento, inviato da un server al client all'atto della prima connessione, memorizzato ed inviato successivamente dal client per farsi riconoscere
- Tramite i cookie è possibile realizzare delle applicazioni interattive basate su sessioni (es. carrello elettronico)

Protocollo Ftp

- File Transfer Protocol: protocollo per accedere ad una specifica directory di un server ed eseguirvi operazioni di download, upload di files.
- Apparentemente semplice, in realtà deve tenere in considerazione l'esigenza di fornire dei comandi indipendenti dal tipo di sistema operativo sottostante (es. file system diversi, nomi diversi ed altre differenze nella memorizzazione dei files)
- Protocollo "storico" di Internet (1971), anche se ora è sempre più sostituito da altri (http, scp) per i suoi evidenti limiti, soprattutto in termini di sicurezza (es. la pssword di login viaggia in chiaro)
- Fornisce una sequenza di comandi per:
  - Collegarsi ad un server ed effettuare il login (open,user, password)
  - Navigare nelle directories del server (cd, ls, size, cdup, dir etc)

Protocollo Ftp



Il protocollo FTP prevede due diverse connessioni (socket):
1) trasferimento comandi FTP
2) trasferimento dati

La seconda connessione può essere attiva o passiva a seconda che sia il server ad attivarla nei confronti del client o viceversa

Protocollo Ftp

- Navigare nelle directory del client (lcd)
- Navigare nelle directories del server (cd, ls, size, cdup, dir etc)
- Settare le modalità di trasferimento (ascii, binary)
- Trasferire files (get, put, mget, mput etc)
- Chiudere la connessione (close, disconnect)

- Per esigenze particolarmente semplici, si può usare un protocollo alternativo (Trivial File Transfer Protocol). Viene ad esempio usato per aggiornare il firmware di apparati di rete (telefoni SIP, switch, router) ed in tutti i casi nei quali non è necessaria interazione fra utente e server

Protocollo Telnet

- Un altro protocollo “storico” di Internet (1969), ora, in genere, sostituito dal più sicuro ssh

- Consente l'accesso ad un server da remoto, fornendo al client una finestra interattiva (shell) per eseguire comandi in formato testo, come se l'utente remoto fosse connesso localmente

- In sintesi quindi il suo scopo è:
  - Da parte del client, l'invio al server dei caratteri digitati a tastiera come se fossero stati digitati direttamente in una shell del server
  - Da parte del server, la trasmissione al client dell'output testuale prodotto dai vari comandi
- Come per FTP, la relativa semplicità del protocollo deve però tener conto dell'eterogeneità dei sistemi e delle differenti modalità di rappresentazione dei dati (es. ASCII, EBCDIC)

Protocollo Telnet

- Per risolvere questo problema, Telnet definisce un terminale virtuale standard (Network Virtual Terminal), in modo da disaccoppiare il protocollo dai sistemi operativi sottostanti

- NVT adotta dei caratteri standard e comuni per la rappresentazione dell'input e dell'output (logical keyboard, logical printer), basata su 95 caratteri ASCII stampabili (NTV ASCII)

- I caratteri digitati a tastiera dal client, vengono intercettati dall'applicazione telnet e trasformati nel formato previsto da NVT. Essi vengono trasmessi al server, dove l'applicazione telnet trasforma i caratteri NVT in quelli specifici del server. Analogo passaggio avviene per l'output.

Socket

- Abbiamo analizzato i principali protocolli client-server e visto che essi si basano sulla trasmissione bidirezionale di messaggi (da client a server e viceversa)
- Nelle successive slides andremo ad approfondire quali sono le tecniche di programmazione che si devono adottare per gestire protocolli di questo tipo, a livello applicativo (socket)
- Prima di affrontare questo argomento, è però importante analizzare cosa succede al messaggio applicativo quando esso viene trasmesso in rete
- Andremo pertanto ad analizzare lo stack TCP/IP, ossia la sequenza di layer software che si fanno carico di gestire il corretto invio/ricezione dei messaggi applicativi

Stack TCP/IP

- Il client (mittente), preparato un messaggio in memoria (es. “Get index.html http/1.0 ”), lo invia al server (ricevente)

- Per far questo definisce una specifica struttura in memoria (socket), che approfondiremo nelle prossime slides, e connette la socket al server, specificandone il numero di porta (ossia il numero well-known che identifica il tipo di processo sul server) ed il suo indirizzo IP ossia l'indirizzo che identifica, in modo univoco, la scheda di rete del server su Internet

- La socket rappresenta, per il client applicativo, un canale di comunicazione client-server molto semplificato, come se esso fosse un file virtuale sul quale vanno scritti i messaggi da trasmettere e letti i messaggi di risposta ricevuti

Stack TCP/IP

- La socket costituisce pertanto un meccanismo di astrazione, che consente alle applicazioni di ignorare tutte le problematiche connesse alla comunicazione e di inviare/ricevere i vari messaggi come flusso indifferenziato di bytes

- Il server rimane sempre in attesa di nuove richieste di connessione (listening) su una socket specifica: alla ricezione della richiesta di connessione da parte del client, il server crea a sua volta una socket specifica sulla quale verranno ricevuti i messaggi in partenza dal client ed inviati i messaggi di risposta (simmetricità della connessione)

- Delle problematiche, connesse alla trasmissione corretta dei messaggi, si fanno carico altri strati software che ricevono i dati inseriti nella socket e ne gestiscono la consegna al server, verificando nel contempo la correttezza di ricezione

- Si fa notare che non si tratta di un unico strato software, in quanto le problematiche sono molteplici e complesse e quindi si preferisce far gestire il processo di trasmissione (ed analogamente quello di ricezione) a strati software diversi (stack)

TCP

- Un primo layer protocollare che troviamo è TCP (Transport Control Protocol)

- TCP riceve i messaggi dalla socket ed ha i seguenti principali compiti:
  - Suddividere il messaggio in pacchetti di dimensioni fisse (TCP segment)

  - Aggiungere ad ogni pacchetto così ottenuto, un'intestazione con particolari informazioni (TCP header), in modo da garantire che tali pacchetti non vadano persi e vengano ricomposti dal ricevente secondo l'ordine con il quale essi sono stati inviati dal mittente

  - Ad ogni TCP segment viene infatti associato, nell'header, il valore di un particolare contatore (Sequence number) che viene inizializzato in modo random in ogni nuova socket ed associato al primo byte del primo segmento trasmesso.

TCP

- Si può pertanto considerare che ogni byte del flusso TCP sia numerato e tale numero corrisponda al sequence number del TCP segment di appartenza, incrementato del valore corrispondente alla posizione del byte all'intero del TCP segment

- Il sequence number di ogni TCP segment inviato viene a costituire pertanto una sequenza di numeri crescenti

- Questo permette a TCP lato destinatario di ricostruire la corretta sequenza di invio e di verificare l'eventuale perdita di segmenti

TCP

- Quando il destinatario (server nel nostro esempio) riceve il pacchetto ed invia a sua volta pacchetti TCP di risposta, viene memorizzato, in un particolare campo (acknowledgment number) dei segmenti di risposta, il valore corrispondente al numero dell'ultimo byte ricevuto + 1, che corrisponde quindi al prossimo sequence number atteso

- Un altro importante campo presente nell'header TCP è il numero di porta (destination port) ossia un numero che identifica in modo univoco il processo ricevente (well-know port per processi server). Tale valore viene dedotto dalla socket.

- Nell'header TCP viene specificato anche il numero di porta del mittente (source port), in modo da consentire la corretta consegna al processo client mittente dei TCP segment di risposta. Si dice che tale numero rappresenta un valore efemerale, in quanto è associato a processi client, per loro natura di durata limitata

TCP

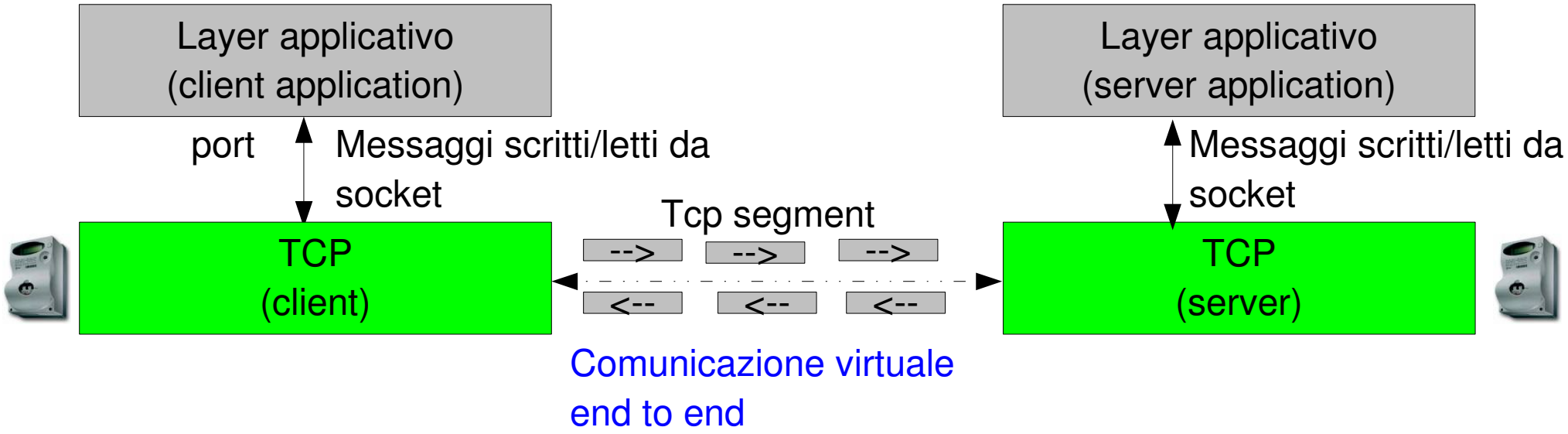
- Infine un altro campo significativo dell'header TCP è la window size che informa il ricevente del numero massimo di byte che il mittente può ricevere in risposta. Se per esempio tale valore è a zero significa che il mittente non è più in grado di ricevere bytes e quindi il ricevente deve bloccare ogni risposta fino a quando non riceva un nuovo segmento con window size maggiore di zero

- TCP è in sintesi lo strato software che si fa carico della corretta consegna dei messaggi inviati e del controllo di flusso fra mittente e destinatario

- Esso deve quindi essere presente sia presso il mittente che presso il destinatario (controllo end to end)

- **Tutto quanto detto per l'invio di messaggi da client a server, vale simmetricamente per il processo inverso, ossia per i segmenti di risposta**

TCP: sintesi



- I messaggi scritti dal mittente (client) su socket, vengono suddivisi da TCP in segmenti sui quali sono presenti delle informazioni di controllo
- Tali segmenti arrivano a TCP destinatario (server) dove vengono controllati e passati alla socket del server, individuata dal parametro destination port dell'header TCP
- Il processo applicativo lato server riceve i messaggi ed invia il messaggio di risposta che segue un percorso simmetrico. In ogni pacchetto di risposta è presente un valore (acknowledgment number) che informa la controparte sull'ultimo segmento ricevuto

UDP

- In alternativa a TCP, i messaggi possono essere consegnati ad un layer software denominato UDP (User Datagram Protocol)

- UDP non suddivide i messaggi in segmenti ma si limita ad inglobarli in pacchetti (UDP datagram), ognuno composto dal messaggio da trasmettere preceduto dall'header UDP (messaggi di lunghezza massima di 65000 bytes circa)

- L'header UDP è molto più semplice rispetto a quello TCP: i campi più significativi presenti sono solo la porta mittente e destinatario. A differenza di TCP non esiste numerazione dei pacchetti ed acknowledgement

- UDP non si fa quindi carico alcuno della corretta consegna corretta dei pacchetti (unreliable protocol)

UDP

- UDP fornisce un meccanismo più snello ed agevole, anche se non affidabile, di consegna dei messaggio; simple and fast
- L'eventuale gestione degli errori è quindi lasciata ai protocolli applicativi
- UDP viene usato infatti in alternativa a TCP quando:
  - si devono privilegiare le prestazioni rispetto alla perdita di pacchetti (es. applicazioni multimediali) oppure
  - si è in presenza di protocolli basati su un semplice meccanismo di messaggi domanda/risposta (es. DNS), per i quali il controllo di flusso non rappresenta un problema e la perdita eventuale di un messaggio viene risolta, in modo molto semplice, ritrasmettendolo

IP

- IP (Internet Protocol) rappresenta il secondo layer software (dopo TCP/UDP), che si fa carico dell'ulteriore elaborazione dei messaggi inviati da un client ad un server

- IP prende in carico, da TCP/UDP, i messaggi - in forma di TCP segment od UDP datagram- e crea, per ognuno di essi, un nuovo pacchetto (IP datagram), aggiungendovi uno suo header (IP header) contenente, fra le informazioni più significative:
  - Indirizzo IP mittente (source address)
  - Indirizzo IP di destinazione (destination address)

- Quindi il nuovo pacchetto che viene a determinarsi è formato da IP header + TCP/UDP header + dati originari (messaggi applicativi od una loro parte)

- Questo processo che vede i messaggi inglobati prima in pacchetto TCP/UDP e questi, a loro volta in pacchetti IP, prende il nome di incapsulamento (encapsulation)

IP

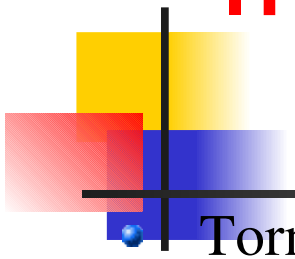
- E' importante notare che ogni layer software, che gestisce il processo di invio, prende i dati dal layer immediatamente superiore ed aggiunge sempre un suo specifico header, contenente le informazioni necessarie al trattamento ad esso associato

- Le informazioni registrate nell'header vengono poi esaminate dal medesimo layer presente nel destinatario, per realizzare i dovuti controlli previsti nello specifico layer, secondo il protocollo standard del layer

- Ad esempio il sequence number viene esaminato dal TCP lato destinatario per controllare la corretta sequenza dei pacchetti ricevuti dal mittente

- Effettuati i controlli necessari il layer, lato destinatario, elimina il proprio header e passa i dati ricevuti al layer superiore (es. IP passa i pacchetti ricevuti a TCP oppure UDP)

IP



- Tornando ad IP, abbiamo visto che nel suo header le informazioni più importanti sono gli indirizzi IP mittente e destinatario

- Il trattamento effettuato da IP consiste infatti, nell'individuare il percorso necessario (route) per la consegna del pacchetto al corretto destinatario

- IP usa appunto l'indirizzo IP di destinazione (e solo quello) per decidere dove inoltrare il pacchetto

- Come vedremo infatti più avanti quando approfondiremo gli indirizzi IP, ognuno di tali indirizzi ha una struttura che individua la LAN di appartenza (net-id) e lo specifico host all'interno della LAN (host-id)

IP

- Si distinguono quindi due casi:
  1. Destinatario sulla stessa LAN del mittente (stesso net-id)
  2. Destinatario su LAN differenti (diverso net-id)

- Nel primo caso la consegna al destinatario avviene nell'ambito della stessa LAN e quindi IP consegna il pacchetto al layer software sottostante(data link) per la trasmissione vera e propria dei dati al destinatario

- Nel secondo caso, IP consulta delle proprie tabelle (tabelle di routing) che indicano, per ogni IP di destinazione, a quale indirizzo IP il pacchetto vada consegnato (next hop)

- Nel caso del client mittente il next hop è rappresentato dall'indirizzo IP del router che mette in comunicazione la rete di appartenza con le altre.

IP

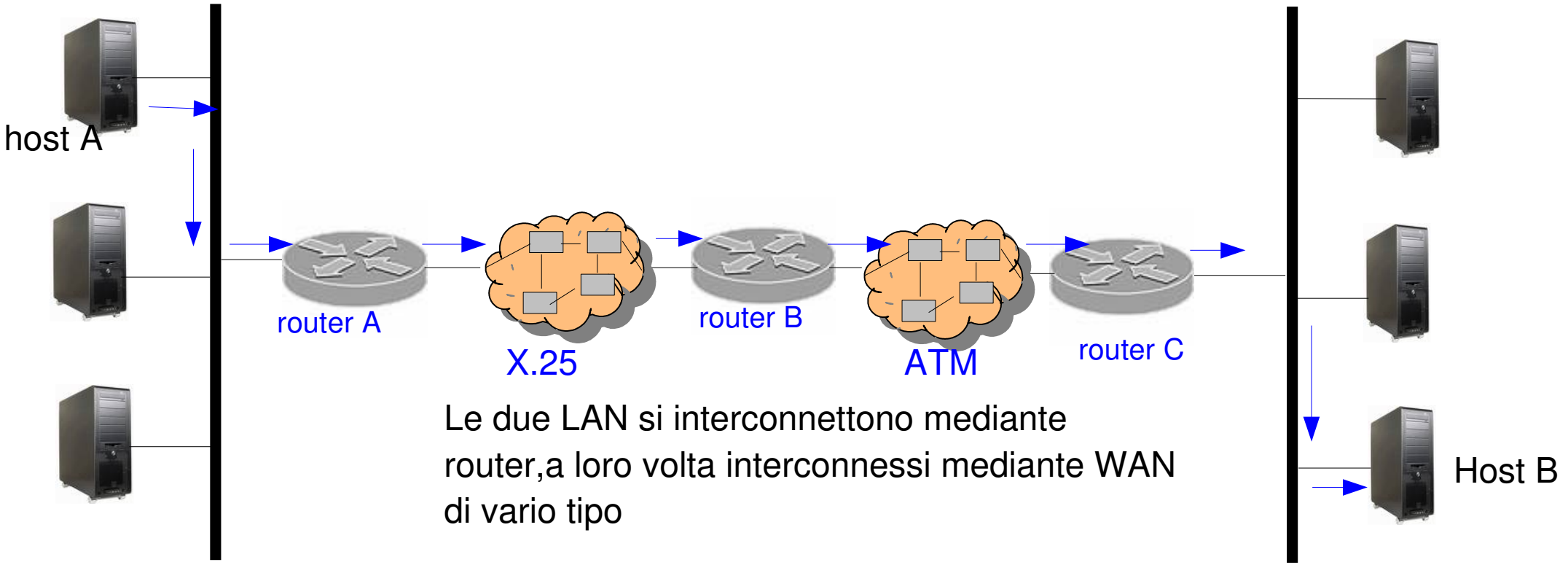
- Anche in questo secondo caso, una volta individuato l'indirizzo IP del next hop, IP consegna il pacchetto al layer immediatamente inferiore (data link)

- Poiché fra reti diverse (diverso net-id) sia possibile l'interscambio di messaggi previsti dai vari protocolli di Internet, occorre che esse siano fra loro collegate da specifici dispositivi detti router (Internetworking).

- Il router ha tante schede di rete quante sono le reti direttamente collegate (vedi esempio seguente)

- IP è installato a bordo di ogni router ed il suo compito è proprio quello di esaminare i pacchetti da inoltrare decidendo, in base all'indirizzo IP di destinazione, l'indirizzo IP del router successivo (next hop) al quale il router va inviato

Internetworking

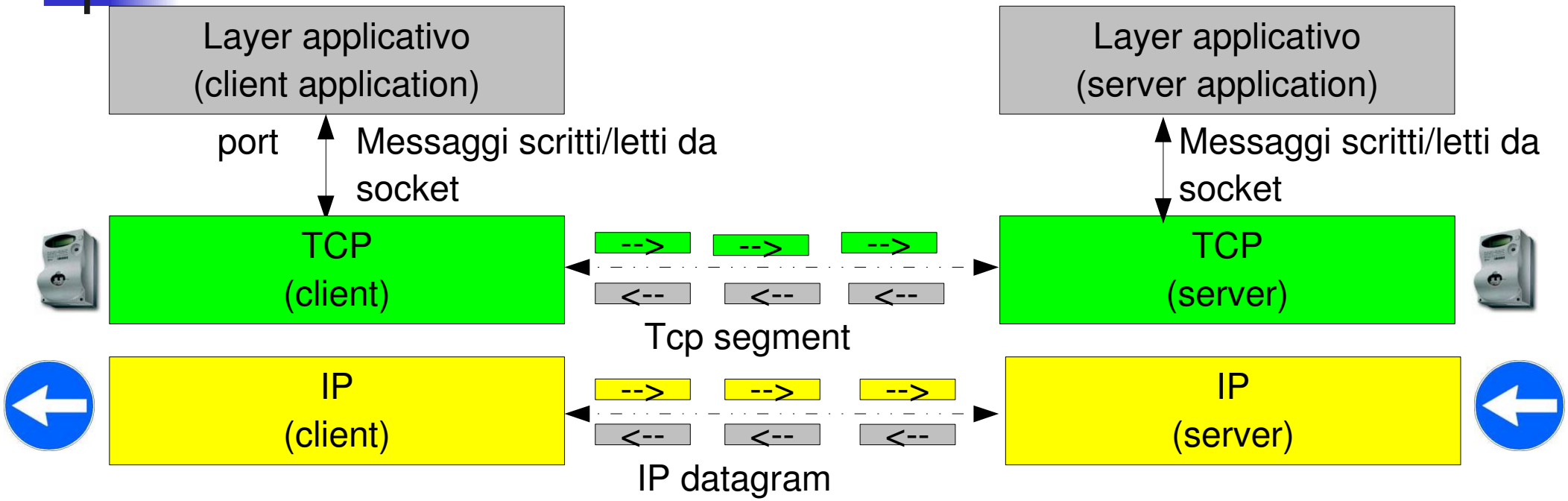


Le due LAN si interconnettono mediante router,a loro volta interconnessi mediante WAN di vario tipo

LAN Ethernet host A (mittente)

LAN Ethernet host B (destinatario)

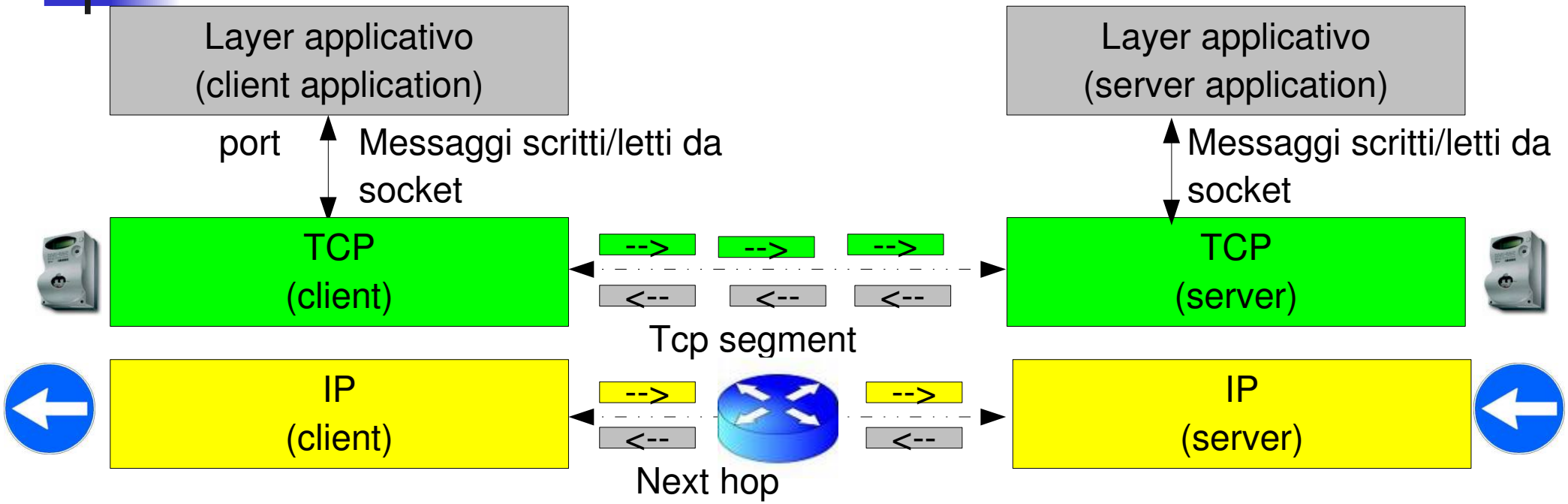
IP (stessa LAN)



- IP aggiunge ad ogni TCP segment (UDP datagram) un header riportante indirizzi IP mittente e destinatario assieme ad altre informazioni di controllo

- In base all'IP destinatario IP mittente decide se consegnare al layer direttamente inferiore oppure ad un router intermedio (next hop)

IP (se LAN destinatario diversa)



- Nel caso di destinatario su altra LAN IP mittente consegna al next hop ossia alla scheda di rete del router collegato alla LAN (default gateway)
- IP, installato a bordo del router, è in grado di capire se il destinatario appartiene ad una delle sue reti direttamente collegate oppure va inoltrato ad altro router , ricadendo in un caso analogo al punto precedente
- Nel caso di destinatario collegato ad una delle sue reti, IP del router consegna il pacchetto al livello immediatamente inferiore (data-link)

Link

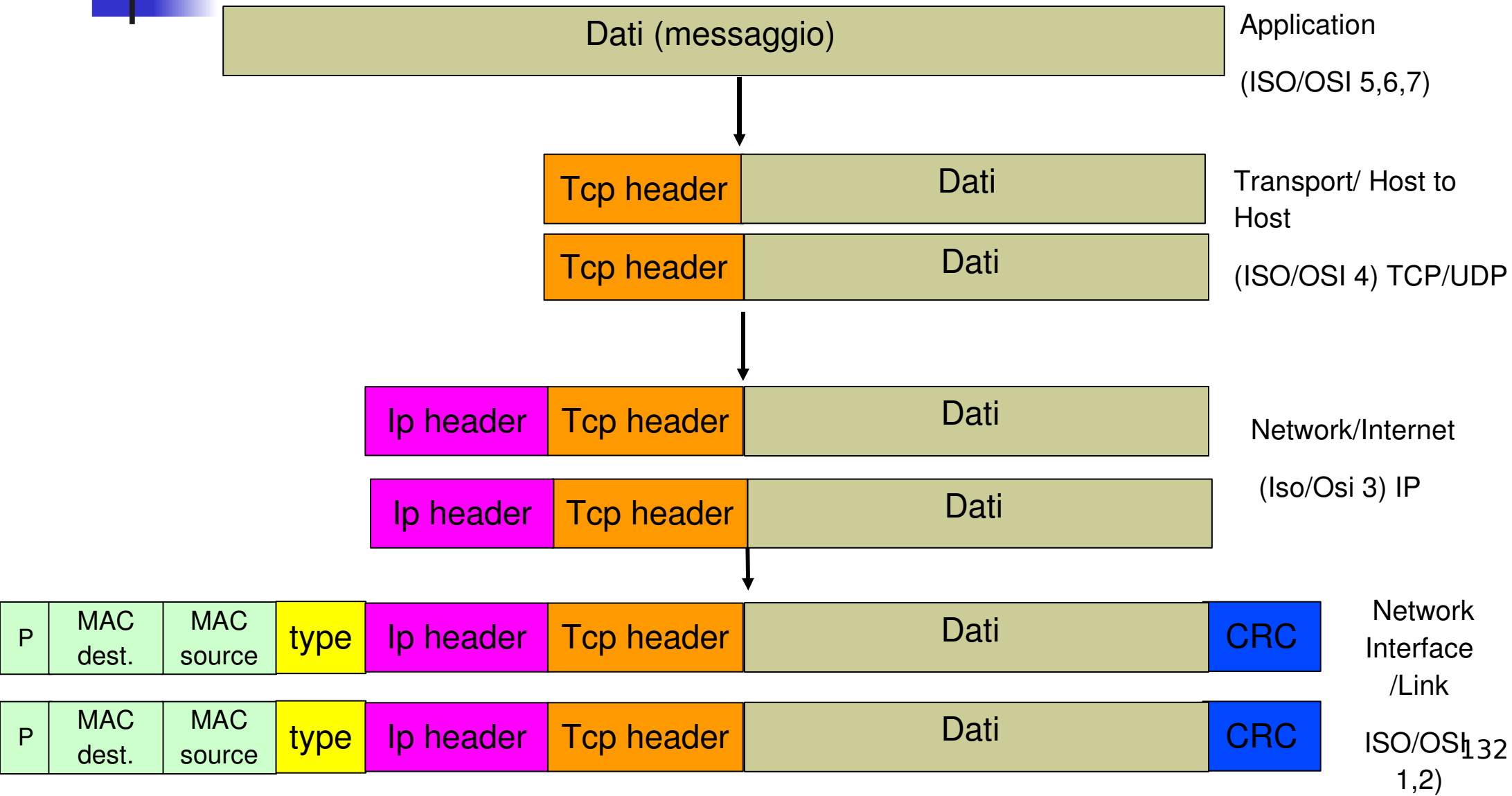
- L'ultimo layer software (link/network interface) prende in carico i pacchetti IP dal livello immediatamente superiore (IP) e lo incapusula in un pacchetto, funzione della tecnologia utilizzata (Ethernet, ATM etc)

- Nel caso di Ethernet il pacchetto IP viene inserito in un frame che contiene, fra le informazioni più significative, gli indirizzi fisici della scheda mittente e di quella destinataria (i cosiddetti MAC address)

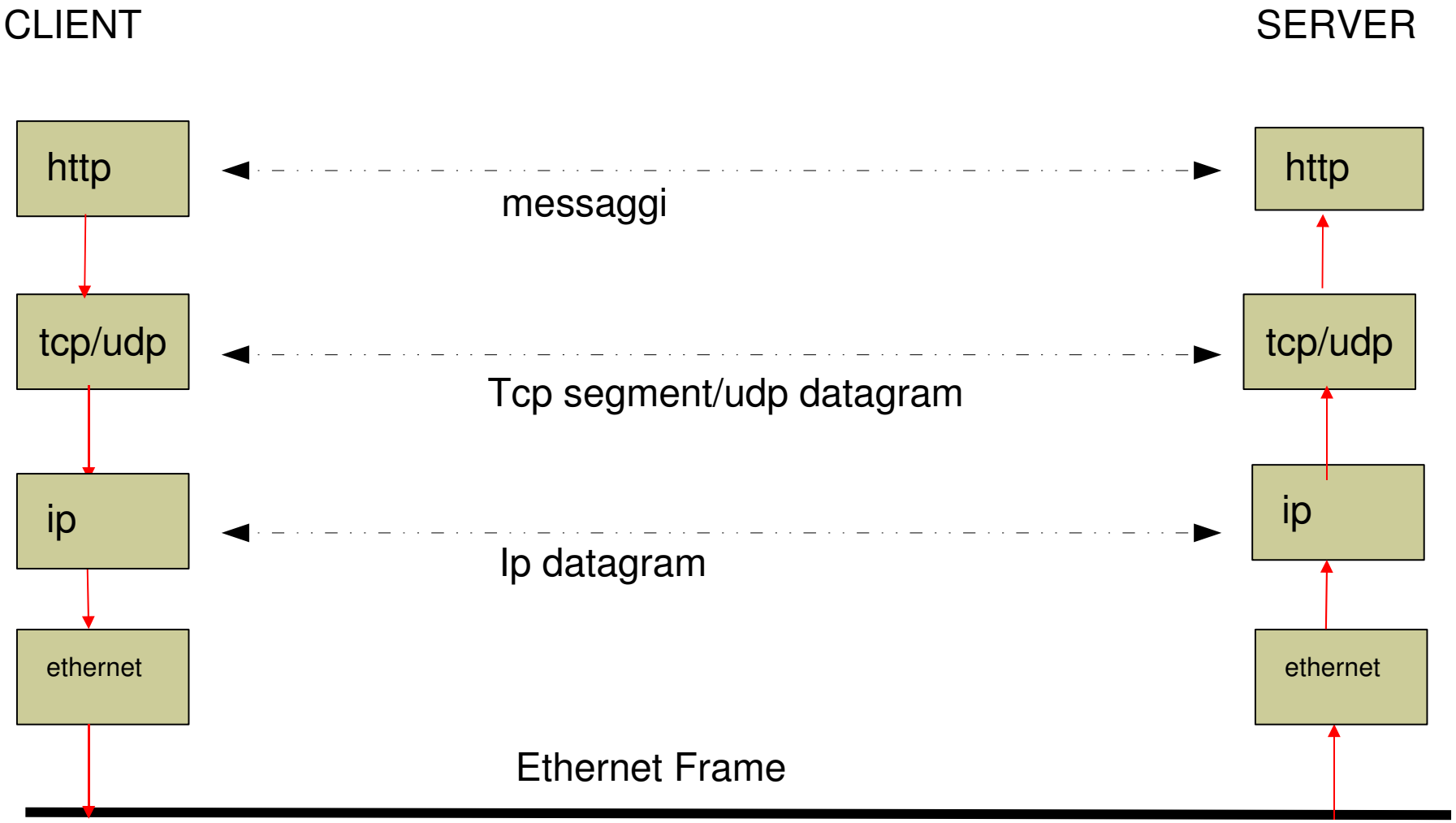
- Ethernet invia in broadcast il frame sulla LAN; solo la scheda di rete avente MAC eguale a quella del destinatario, tratterà il frame

- Ethernet si fa anche carico di codificare i bits dei frame in opportuni segnali elettrici, ottici, radio per l'invio fisico dei dati al destinatario

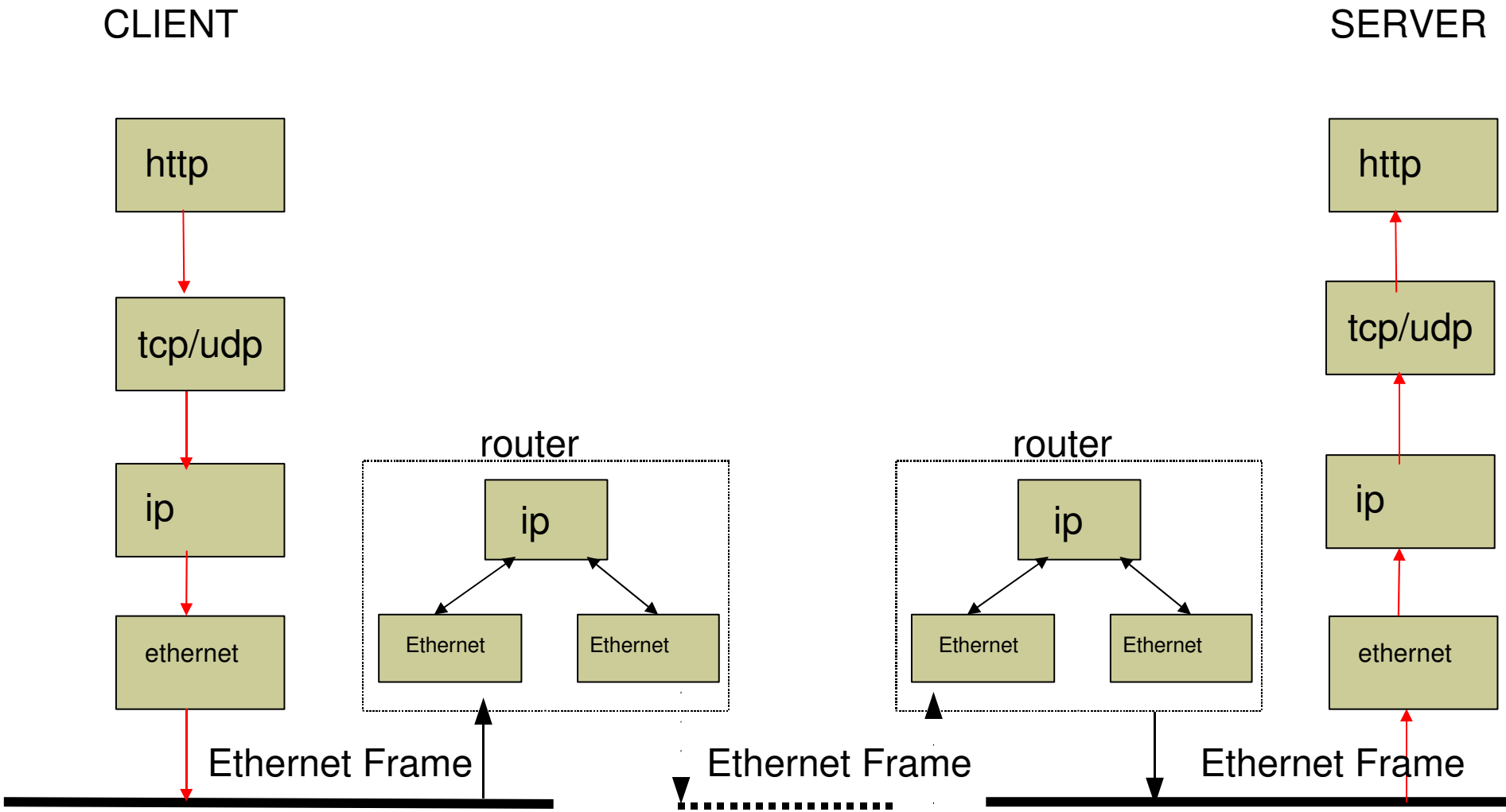
Stack TCP/IP


Dati (messaggio)
Application
(ISO/OSI 5,6,7)
Tcp header
Dati
Transport/ Host to Host
Tcp header
Dati
(ISO/OSI 4) TCP/UDP
Ip header
Tcp header
Dati
Network/Internet
Ip header
Tcp header
Dati
(Iso/Osi 3) IP
P
MAC dest.
MAC source
type
Ip header
Tcp header
Dati
CRC
Network Interface /Link
P
MAC dest.
MAC source
type
Ip header
Tcp header
Dati
CRC
ISO/OSI 1,2)

Stack TCP/IP (se server su stessa LAN)



Stack TCP/IP (se server su LAN diversa)



Socket

- Quando un'applicazione client deve inviare messaggi ad un server, come prima cosa crea una socket, ossia definisce un'area di memoria alla quale vengono associati 5 parametri
  - Protocollo immediatamente inferiore da utilizzare (TCP/UDP)
  - Indirizzo IP mittente
  - Indirizzo IP destinatario (risolto eventulmente tramite DNS)
  - Porta mittente (efemerale)
  - Porta destinataria (well-known)

- Il client, creata la socket, si connette quindi al server con un particolare comando messo a disposizione dalla libreria delle socket (connect); da questo punto in poi il client è in grado di inviare/ricevere, sulla socket, messaggi allo/dallo specifico server

- Il server, al momento della connect da parte del client, deve già disporre di una socket in grado di ricevere richieste di connessione (server in listening)

Socket

- Se la connessione viene accettata, viene creata una specifica socket lato server per la ricezione dei messaggi inviati dal client e l'invio, a questo, delle risposte
- La socket è un canale di comunicazione che consente lo sviluppo di applicazioni client/server
- Allo stesso server si collegano contemporaneamente, via socket, più client
- Introdurremo ora la programmazione delle socket prima lato server e poi lato client, analizzando i principali comandi da utilizzare
- Infine compileremo ed eseguiremo un'applicazione client/server che effettua l'eco del messaggio inviato dal client (Echo Server)

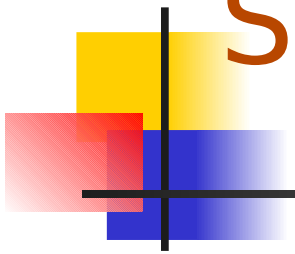
Socket

- Come primo passo, un'applicazione server crea una socket generica, in grado di ricevere messaggi (richieste di connessione) da parte di qualsiasi client, indipendentemente dal suo indirizzo IP

- Come secondo passo (bind), il server associa alla socket il suo indirizzo IP e la sua porta well-known (socket TCP/UDP)

- Poi, terzo step, il server si mette in attesa di richieste di connessione da parte del client; esse vengono poste in una specifica coda (listen)

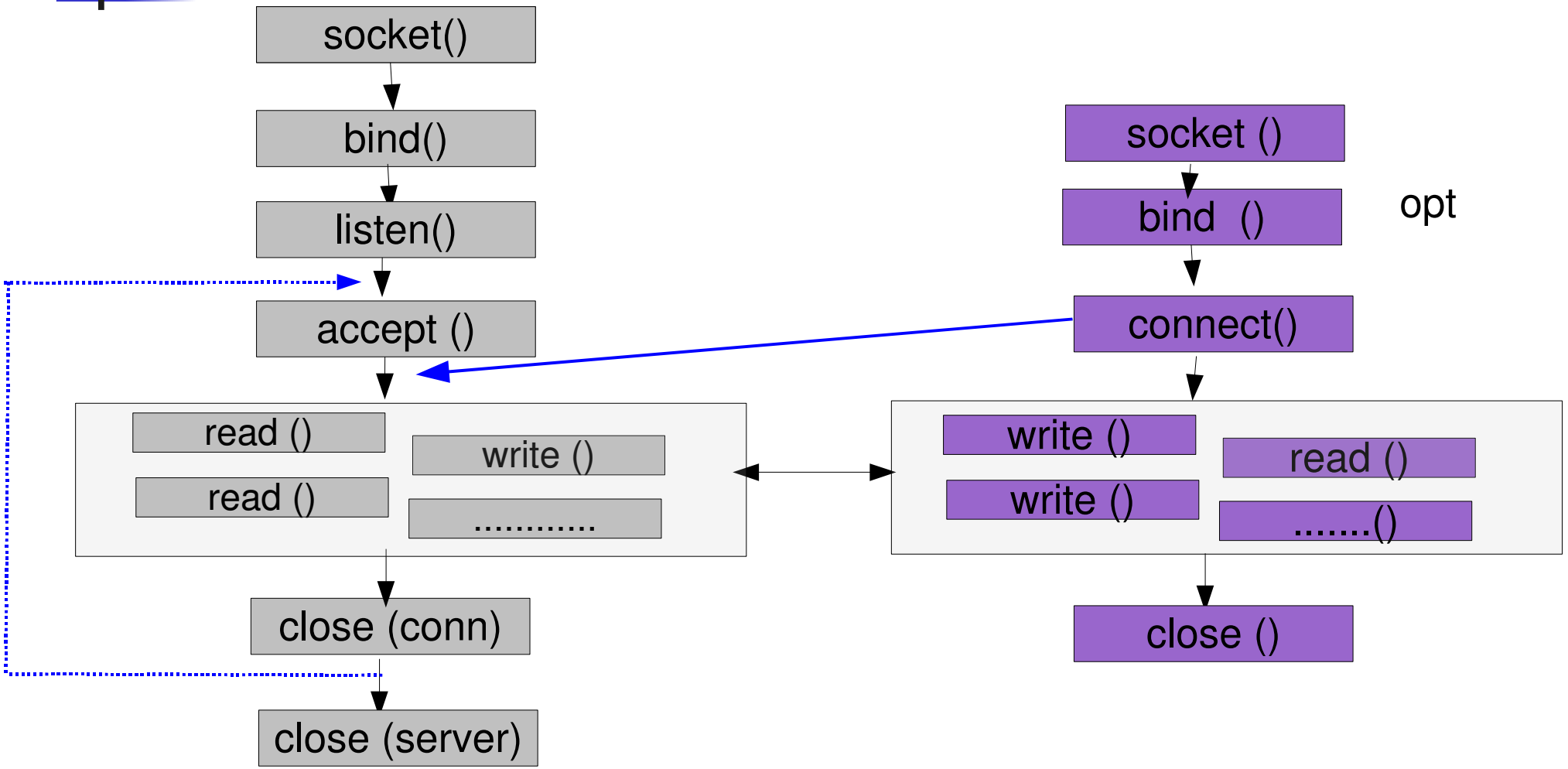
Socket



- Ad ogni nuova richiesta di connessione da parte di un client, il server l'accetta (accept), creando una nuova socket, distinta dalla precedente (che deve rimanere sempre disponibile per le altre richieste) e specifica per il client

- Nel caso di server iterativo (no fork), il server comunica con il client attraverso la nuova socket e, terminata la connessione, riprende in considerazione la successiva richiesta di connessione presenta nella coda di listening

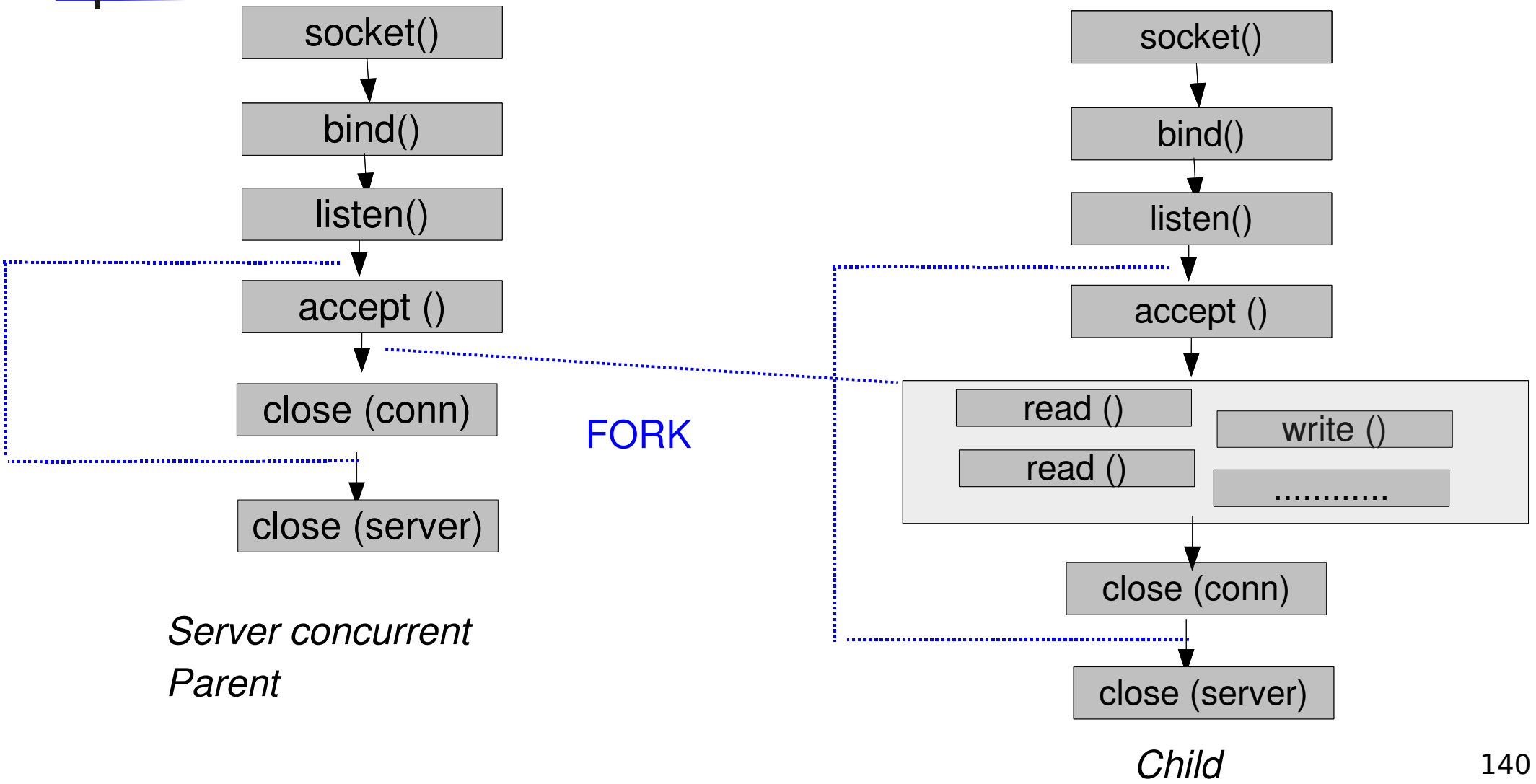
TCP Socket



Server iterativo

Client

TCP Socket


socket()
bind()
listen()
accept ()
close (conn)
close (server)
FORK
socket()
bind()
listen()
accept ()
read ()
read ()
write ()
.....
close (conn)
close (server)
Server concurrent
Parent
Child

Socket

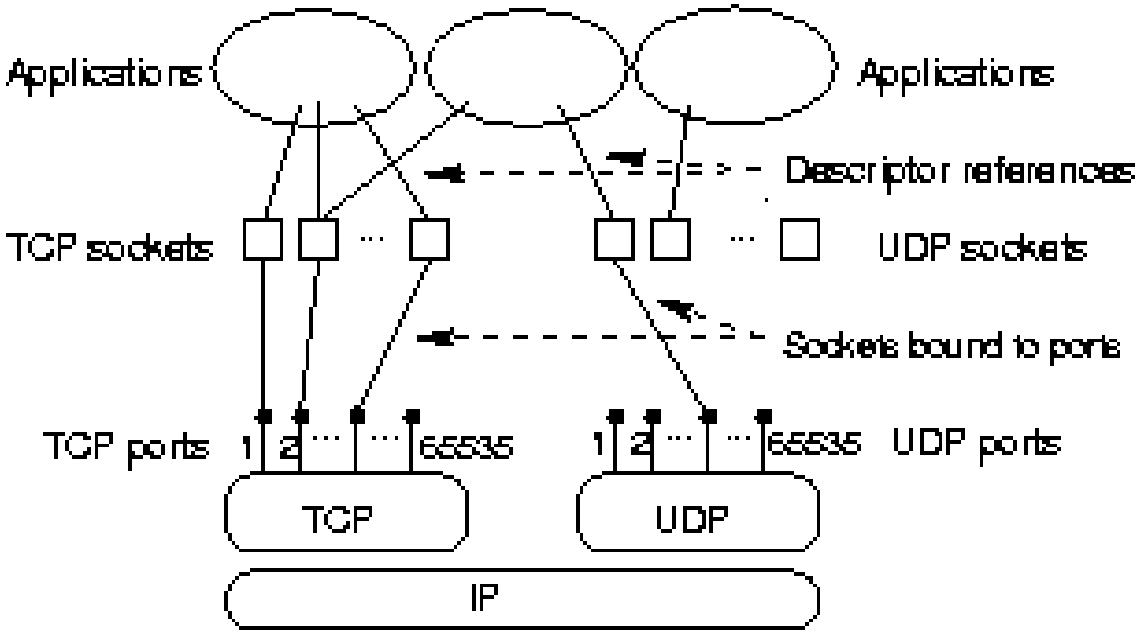
- Nel caso di server concurrent, il processo server crea, con fork, un nuovo processo (oppure un nuovo thread) che gestirà la comunicazione con il client specifico attraverso la nuova socket generata dall'accept; il processo padre continuerà invece la gestione della coda di listen

- Il processo client, molto più semplicemente, si limita a creare una socket, ad associare eventualmente ad essa una specifica porta ed uno specifico indirizzo IP (operazione generalmente effettuata automaticamente) ed a connettere tale socket al server (connect)

- Vediamo ora in dettaglio le socket TCP ed UDP

Socket

- Ogni socket TCP/UDP è identificata da tipo di protocollo, porta ed indirizzo IP mittente, porta ed indirizzo IP destinataria



Socket

- Ogni operazione di creazione di socket restituisce un numero, detto socket descriptor ed analogo ad un file descriptor, che viene usato come riferimento per le successive operazione di lettura scrittura (recvfrom, sendto, recv, etc)

- Il valore di socket descriptor, in Unix, è il primo numero di file descriptor disponibile; in altri sistemi operativi esiste invece differenza fra socket descriptor e file descriptor

Socket

- A differenza delle pipes tale canale è sempre bidirezionale
- E' importante ricordare che i dati vengono trasmessi in forma di stream ossia di flusso non formattato di dati; non esiste in altre parole, nelle socket, il concetto di record
- Vediamo ora in dettaglio come vengono utilizzate con il linguaggio C socket che utilizzano, per il trasporto il protocollo TCP

TCP Socket

- L'applicazione server parte e crea una socket TCP pronta a ricevere la connessione da parte dei vari client su una specifica well-known port.

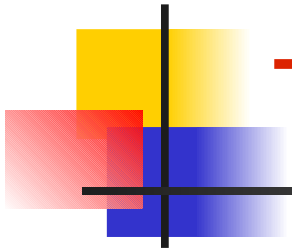
- Tale socket dovà avere un indirizzo IP ed una porta specifici lato server e qualsiasi indirizzo IP e porta lato client (cfr. comando bind)

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

Server

1. Crea una TCP socket
2. Assegna address/port alla socket (bind)
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta una nuova connessione
   b. Comunica con il client
   c. Chiude la connessione

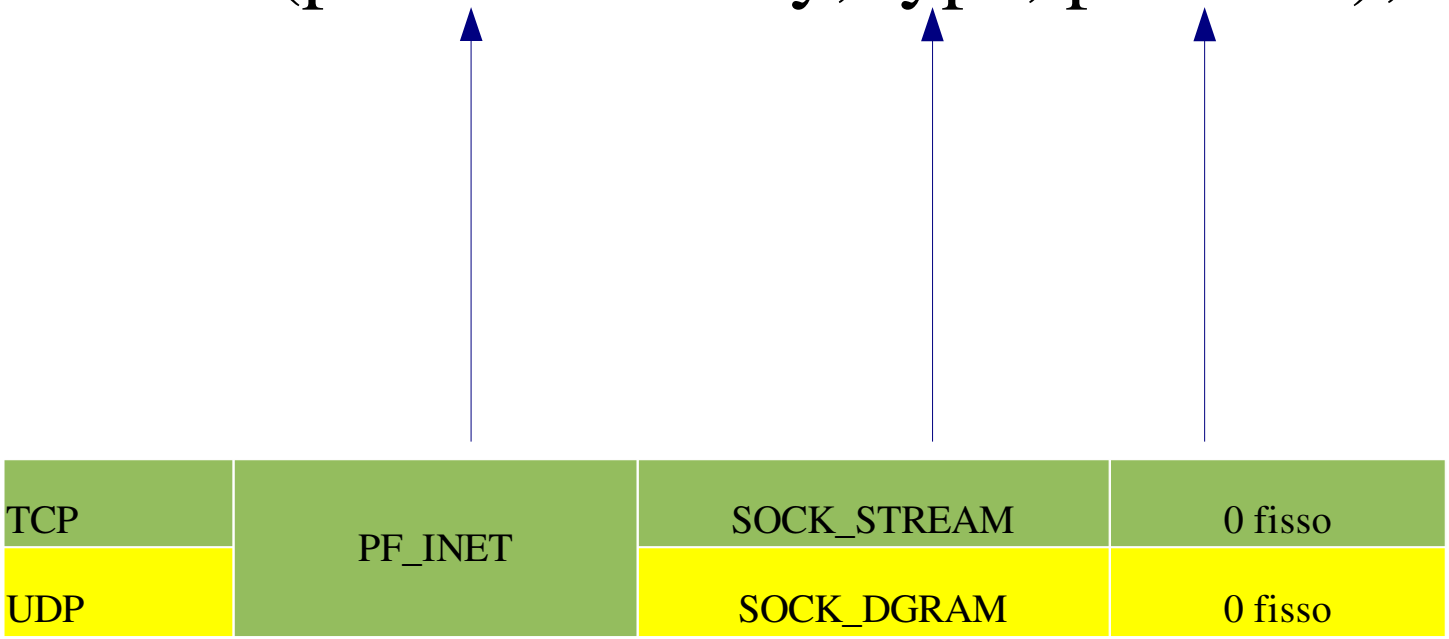


TCP Socket

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <errno.h>
.....
int sd;   /*definisce il  socket descriptor */

/*  crea una socket TCP memorizzando  nella variabile sd il socket descriptor */

if ( ( sd = socket(PF_INET, SOCK_STREAM, 0)  < 0)
            {  perror("Socket");
               exit(errno);  }
.....
```

TCP Socket

sd= socket(protocol family, type, protocol);



Sintassi dell'istruzione socket

TCP Socket

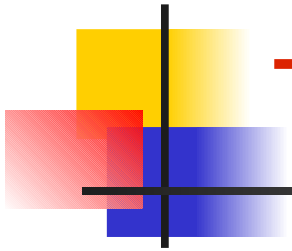
- Il server, con operazione di bind, assegna quindi, alla socket creata:
  - il suo indirizzo IP (ANY nel caso di host multihomed)
  - la porta well-known sulla quale rimarrà in ascolto di connessioni
- L'istruzione bind, in generale, configura i valori "locali" della socket (indirizzo , porta)

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

Server

1. Crea una TCP socket
2. Assegna address/port alla socket (bind)
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta una nuova connessione
   b. Communica con il client
   c. Chiude la connessione



TCP Socket

```
#define MY_PORT=9999

struct    sockaddr_in    addr;                          /* struct definition */
bzero   (&addr,sizeof addr));                           /* zero filling */
addr.sin_family = AF_INET;                              /* Internet address family */
addr.sin_port = htons(MY_PORT);                         /* request for  specific port  9999 */
addr.sin_addr.s_addr = htonl( INADDR_ANY);              /* any IP interface */

if ( bind(sd, (struct sockaddr *)&addr, sizeof(struct sockaddr)) != 0 )
    { perror("socket--bind");
      exit(errno); }
```

TCP Socket

Struct Generica

- **struct sockaddr**

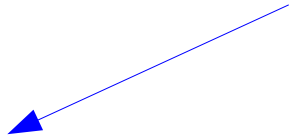
```
{
        unsigned short sa_family;      /* Address  Family (e.g., AF_INET) */
        char sa_data[14];              /* Protocol-specific address information */
};
```

Struct specifica per IP

- **struct sockaddr_in**

```
{
        unsigned short sin_family;     /* Internet protocol (AF_INET) */
        unsigned short sin_port;       /* Port (16-bits) */
        struct in_addr sin_addr;       /* Internet address (32-bits) */
        char sin_zero[8];              /* Not used */
};
```

- **struct in_addr**



```
{
        unsigned long s_addr;          /* Internet address (32-bits) */
};
```

TCP Socket

- A cosa servono le funzioni htonl, htons ?

- Occorre ricordare che CPU di tipo differente usano due modi diversi per memorizzare i valori interi (16,32 bit ed oltre) :
  - Big Endian (es. Motorola)
  - Little Endian (es. Intel)

TCP Socket

BIG ENDIAN

Most Significant Byte

Less Significant Byte

LITTLE ENDIAN

Less Significant Byte

Most Significant Byte

n = indirizzo in memoria

TCP Socket

- Supponiamo ad esempio di dover memorizzare il valore esadecimale (0x04030201)
- Nelle Cpu di tipo Big Endian il valore 04 (Most Significant Byte) viene assegnato al byte con indirizzo di memoria inferiore, ossia n
- Nell Cpu di tipo Little Endian il valore 04 viene assegnato al byte di indirizzo di memoria superiore

TCP Socket

BIG ENDIAN

memory address 00

Most Significant Byte

Less Significant Byte

LITTLE ENDIAN

memory address 00

Less Significant Byte

Most Significant Byte

n = indirizzo in memoria

TCP Socket

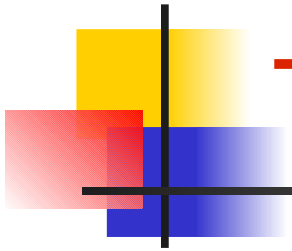
- Le socket usano il metodo Big Endian per memorizzare i dati nello stream di comunicazione e quindi vengono messe a disposizione delle funzioni opportune per effettuare la conversione da little endian a big endian, in modo da assicurare la portabilità

- Si dice quindi che tali funzioni riportano i dati in network byte order

TCP Socket

- Quindi:
  - htonl (host to network long): converte un integer long (32 bits) in big endian qualunque sia la modalità di rappresentazione nell'host
  - htons (host to network small): idem per small integer (16 bits)

- Per dettagli usare il comando shell: man byteorder



TCP Socket

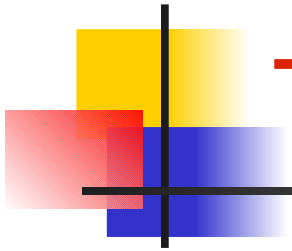
- Il server associa la socket così creata alla coda di listening
- Con l'istruzione listen, il server si dichiara pronto a ricevere le varie richieste di connessione da parte dei client; queste verranno inserite in una specifica coda, la cui dimensione costituisce un parametro dell'istruzione
- Le varie richieste in coda verranno poi trattate dalla seguente istruzione di accept

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta una nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
if (listen(sd, 20) != 0)
   {  perror("socket--listen");
      exit(errno);}

// 20 è eguale al numero massimo di  richieste in coda
```

TCP Socket

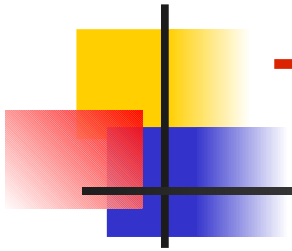
- Il server quindi entra in un ciclo infinito
- Il processo server si ferma
- Quando arriva una nuova connessione, essa entra nella coda di listening e, non appena il server è in grado di elaborarla, viene eseguita l'istruzione accept che crea una nuova socket specifica fra server e client
- La nuova socket ha i valori locali del server e quelli remoti del client connesso

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
while(1)     /* ciclo senza fine */
{
   int  clientfd;
   int  msglen;
   struct sockaddr_in client_addr;
   int addrlen=sizeof(client_addr);

    /* riceve dal client  una richiesta di connessione e crea una nuova socket */
    if ((clientfd=accept(sd,(struct sockaddr *)&client_addr,&addrlen)) > 0)
        printf("%s:%d connected\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
    .......
}
```

TCP Socket

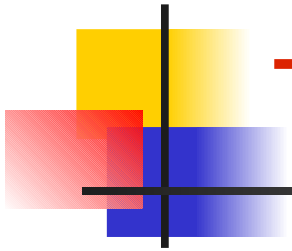
- Il server, dopo l'accept, scambia le informazioni con il client sulla nuova socket, creata dall'istruzione accept, mediante istruzioni di tipo recv,send.
- Il server può lavorare in modo iterativo (una connessione per volta), oppure , usualmente, in modo concurrent (viene creato un processo figlio).
- Qualora la lunghezza del messaggio superi quella del buffer previsto per la ricezione, sono necessarie più istruzioni di lettura

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessionecret

Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
#define MAXBUF          1024
char    buffer[MAXBUF];

 /* Riceve il messaggio dal  client */

        msglen = recv(clientfd, buffer, MAXBUF,0);

/* Invia il messaggio di risposta al client */
        send(clientfd, buffer, msglen, 0);
```

TCP Socket

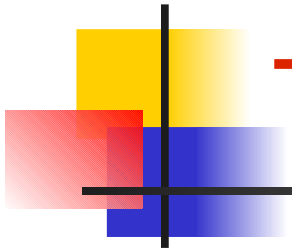
- Si chiude infine la socket di connessione con il client e si ritorna (nel caso di server iterativo) ad esaminare la successiva richiesta di connessione

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

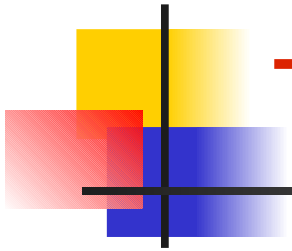
Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

close(clientfd)



TCP Socket

Laboratorio

- Creiamo il processo server di tipo iterativo secondo le modalità appena viste

TCP Socket

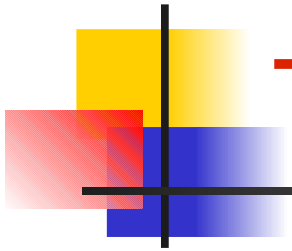
- Vediamo ora l'implementazione del client
- Innanzitutto viene creata una socket di connessione

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

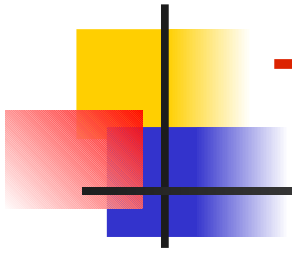
Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
int  clientfd;

/* Crea una socket di connessione  al server TCP */

if ((clientfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      printf("errore in socket\n");
      exit(1);
      }
```



TCP Socket

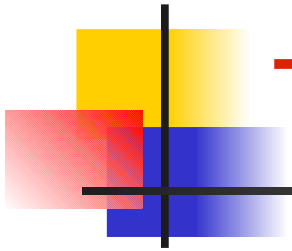
- Si attiva quindi la connessione

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

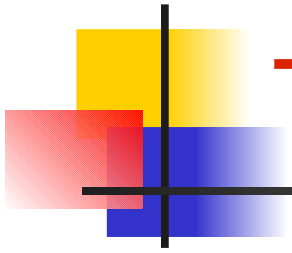
Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
struct sockaddr_in    servaddr;

memset((char )&servaddr, 0, sizeof(servaddr));          /* struct initialization */
servaddr.sin_family =  AF_INET;                         /* Internet Family*/
servaddr.sin_port = htons(9999);                        /* server port */
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");      /* Server address */

if (connect(clientfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
    printf("errore in connect\n");
```



TCP Socket

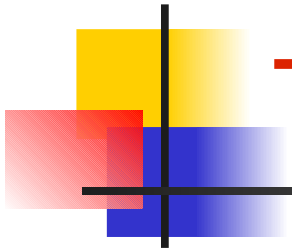
- Si esegue il colloquio con il server

Client

1. Crea una TCP socket
2. Apre una connessione
3. Comunica con il server
4. Chiude la connessione

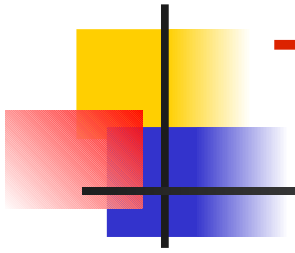
Server

1. Crea una TCP socket
2. Assegna address/port locali alla socket
3. Si mette in ascolto sulla socket
4. Per ogni nuova richiesta:
   a. Accetta la nuova connessione
   b. Comunica con il client
   c. Chiude la connessione



TCP Socket

```
 strcpy(buf,"stringa di eco");  /* null padded*/

/* Invia  la stringa al  server */
if (send(clientfd, buf, sizeof(buf), 0) !=  sizeof(buf))
       printf("send() error");
printf("client sent to server: %s\n",buf);

/* riceve echo */
 if ((recvMsgSize = recv(clientfd, buf,sizeof(buf), 0)) < 0)
                        printf("recv() failed");
 printf("client received from server: %s\n" ,buf);
```



TCP Socket

- Esercizio SimpleClient.c)
- Test funzionamento applicazione
- Homework
  - Echo server in modalità concurrent
  - Time server (daytime_server.c, daytime_client.c)