

Sistemi Operativi II

Corso di Laurea in Ingegneria Informatica

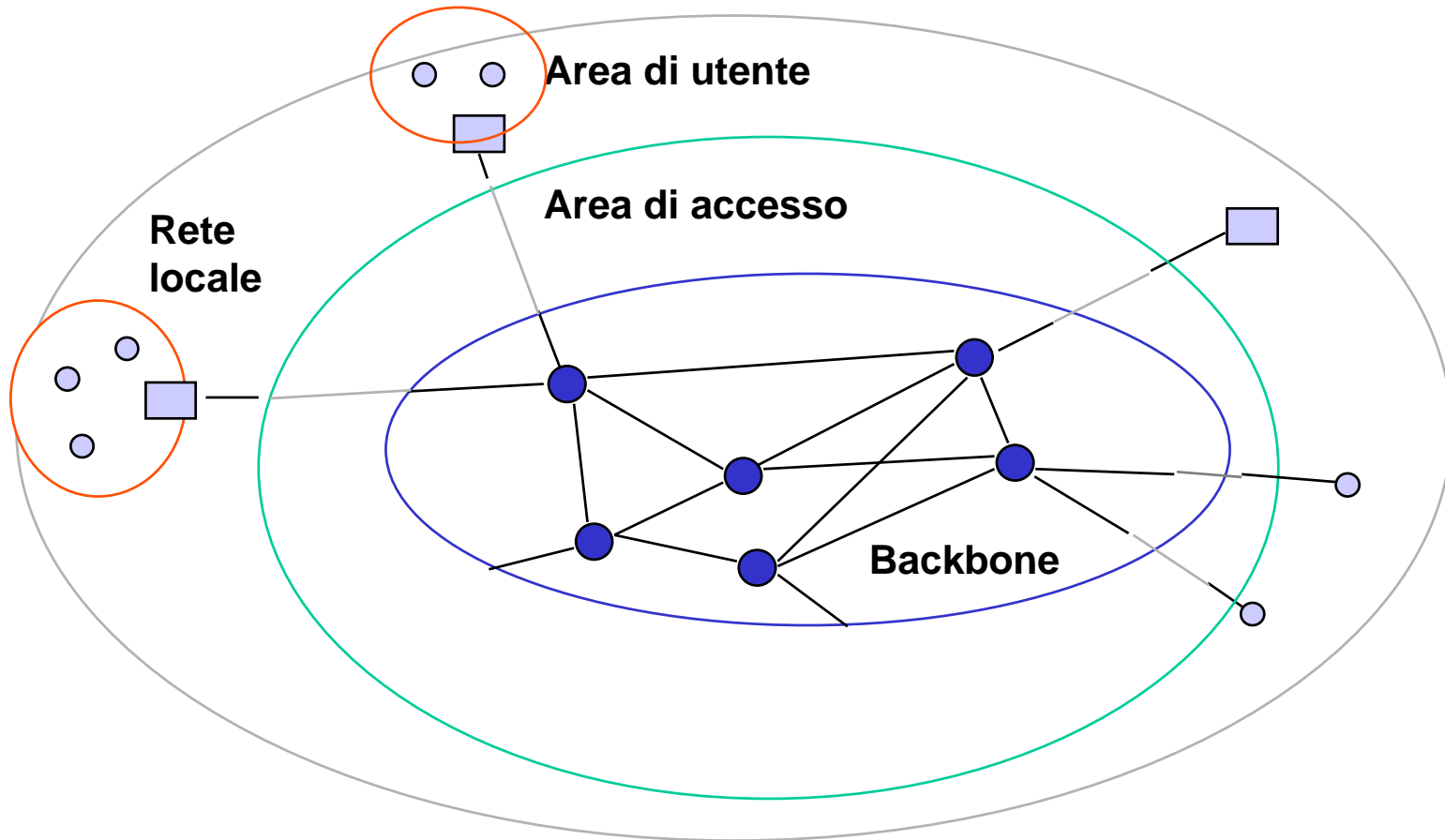
Facolta' di Ingegneria, Universita' "La Sapienza"

Docente: Francesco Quaglia

Programmazione di rete:

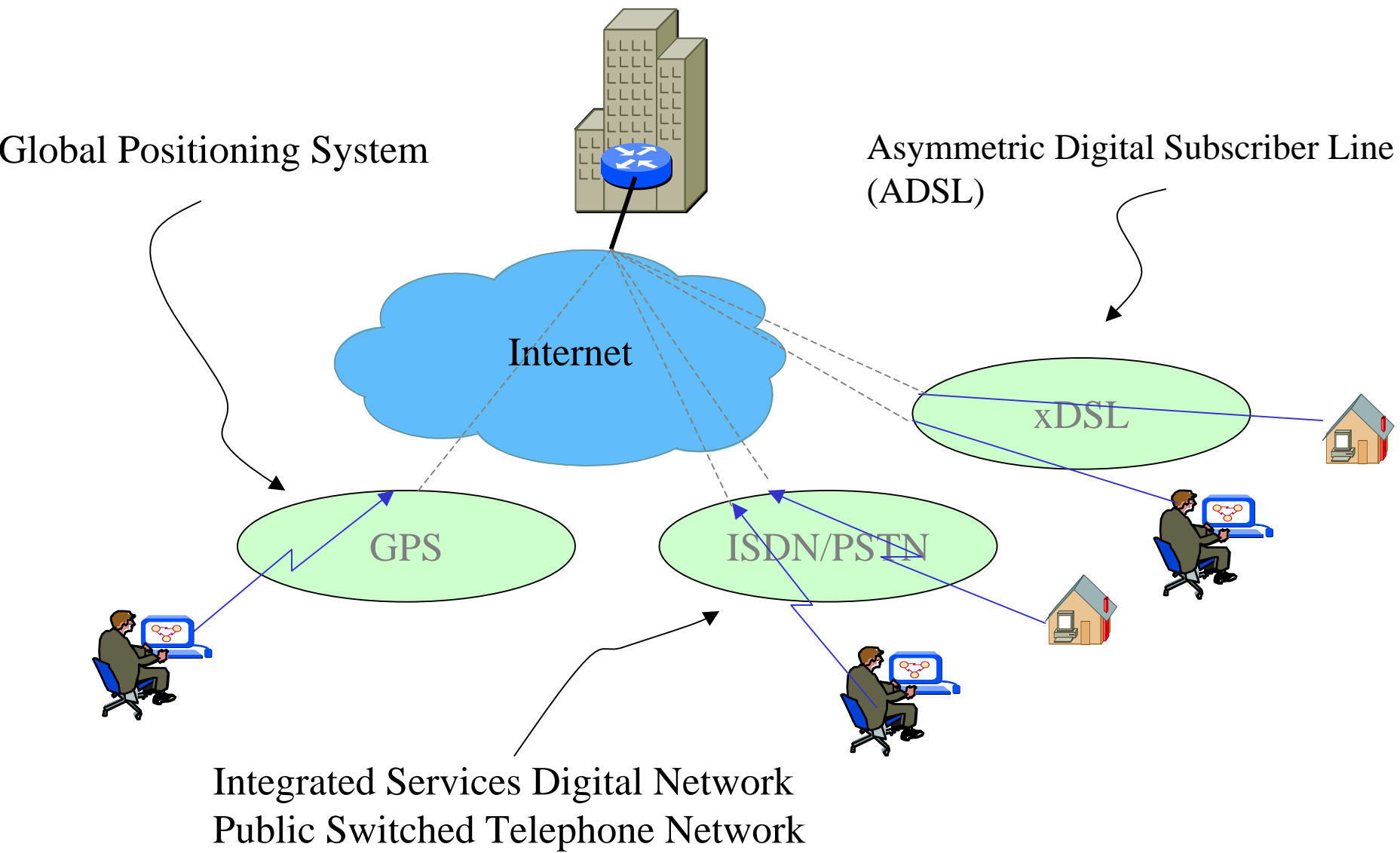
1. Architettura di Internet
2. Richiami di TCP/IP
3. Sockets in sistemi UNIX/Windows

Rete geografica per trasmissione dati

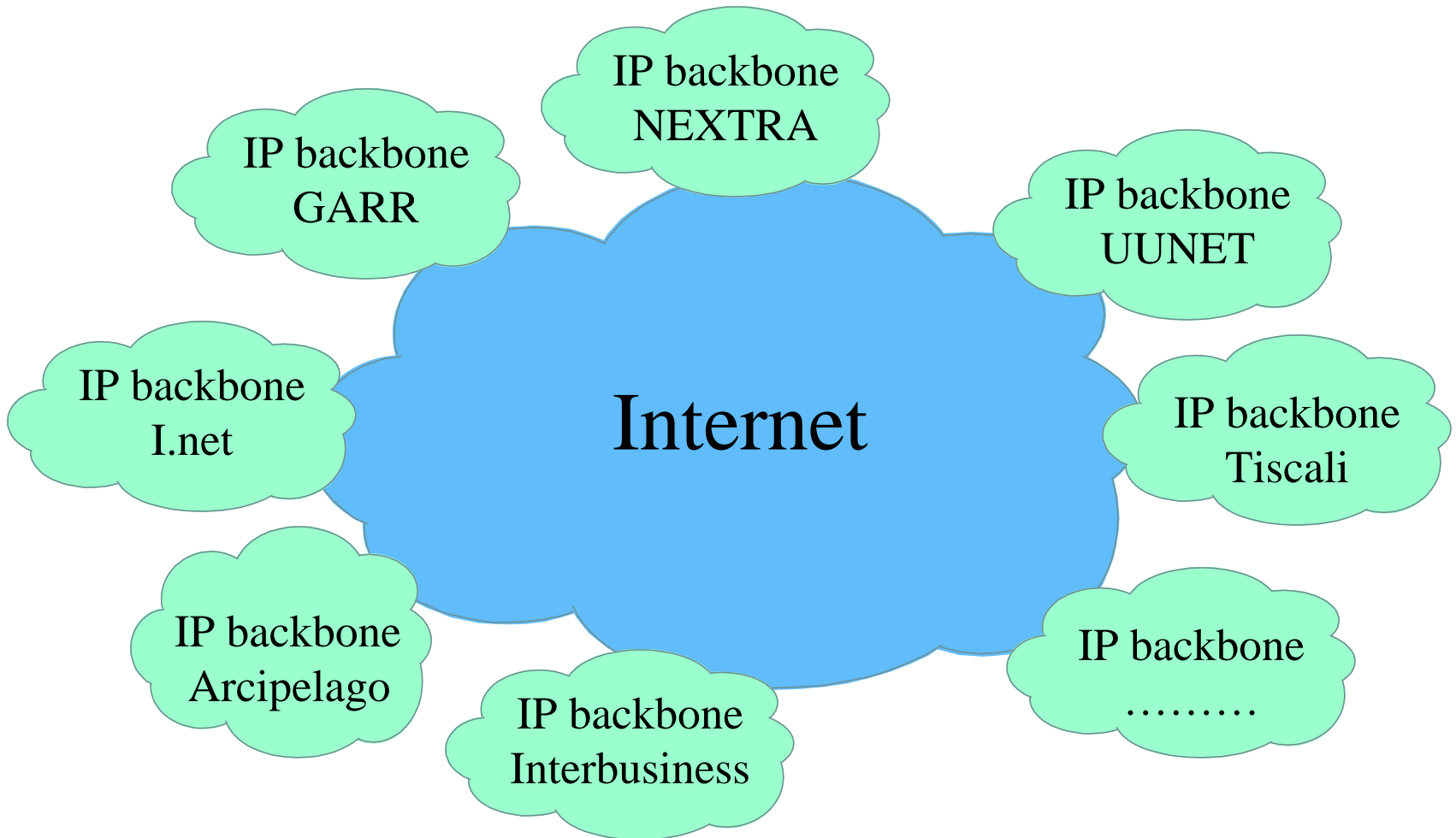


- = host
- = unità di accesso (gateway)
- = nodo del sottosistema di comunicazione (router)

Tecnologie per l'accesso

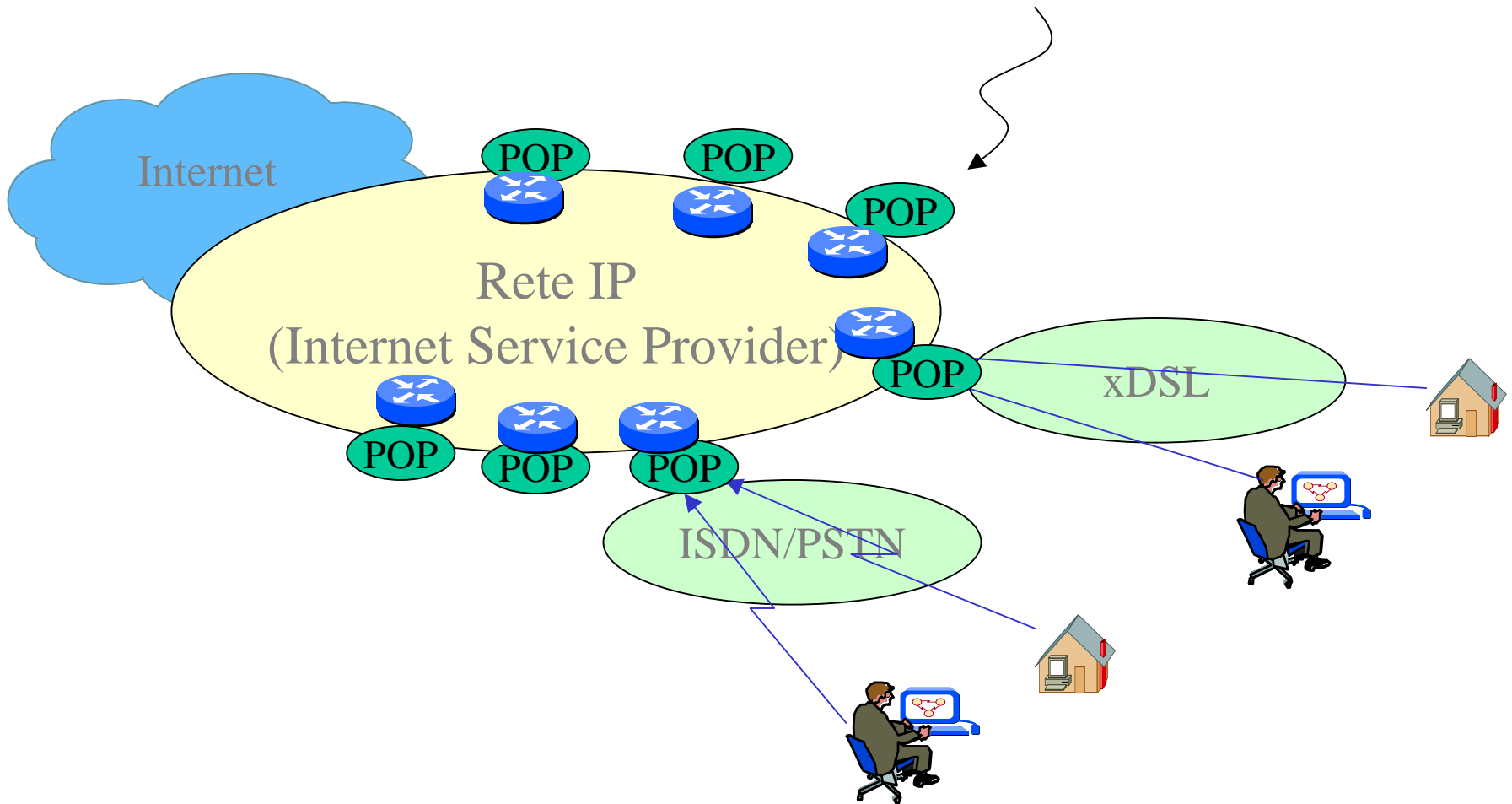


Architettura reale di Internet

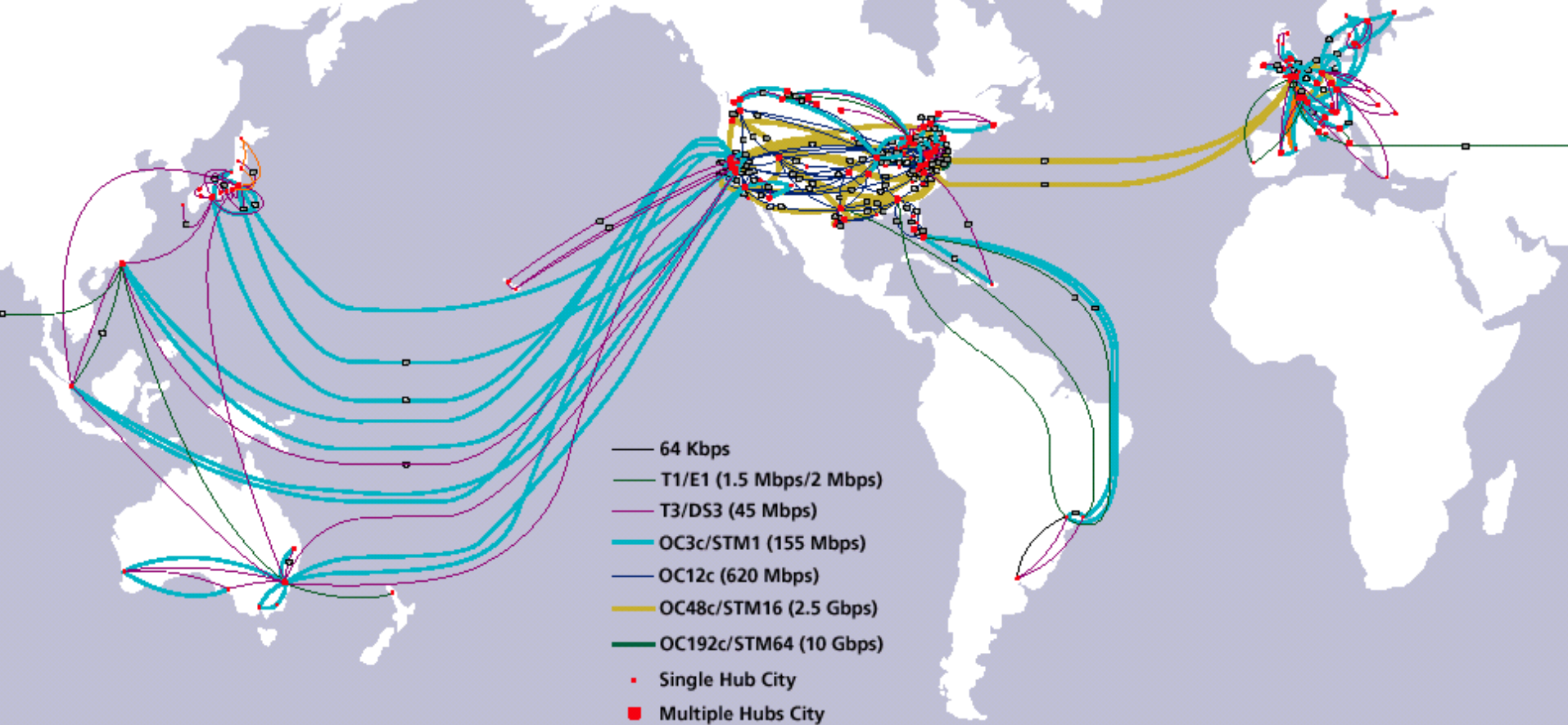


Internet service providers

Point of Presence



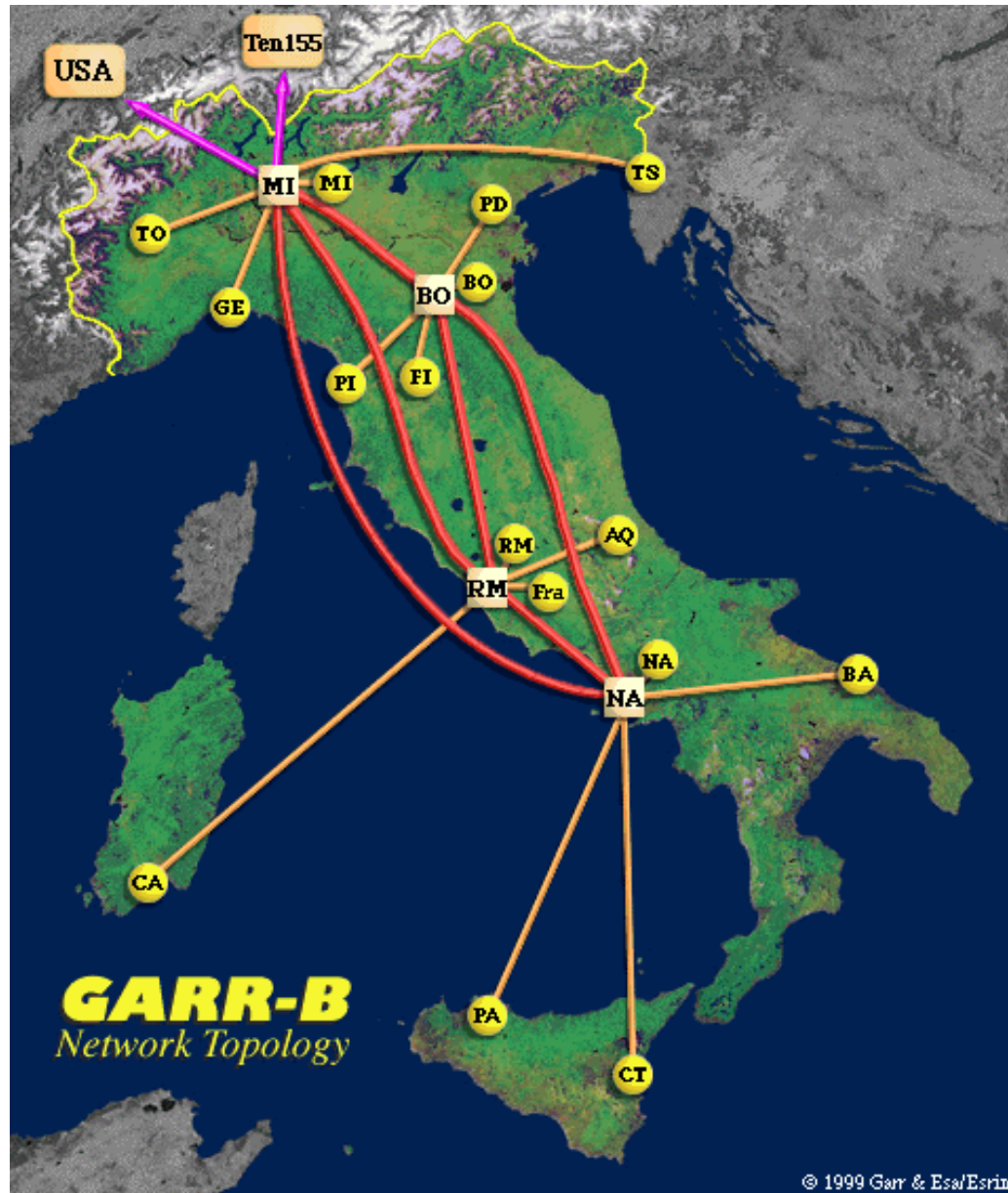
WorldCom's Global UUNET Internet network



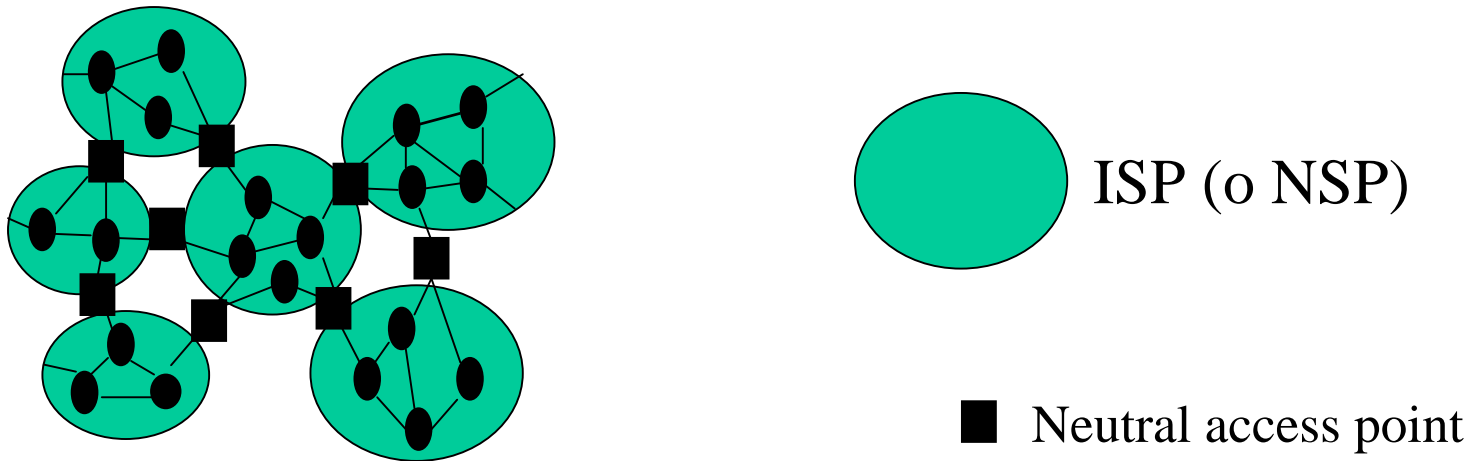
For more information see www.uu.net/network/maps
NB: UUNET also has infrastructure within individual countries, which is not shown on this map.
January 2001

WORLD.COM.

Backbone del GARR



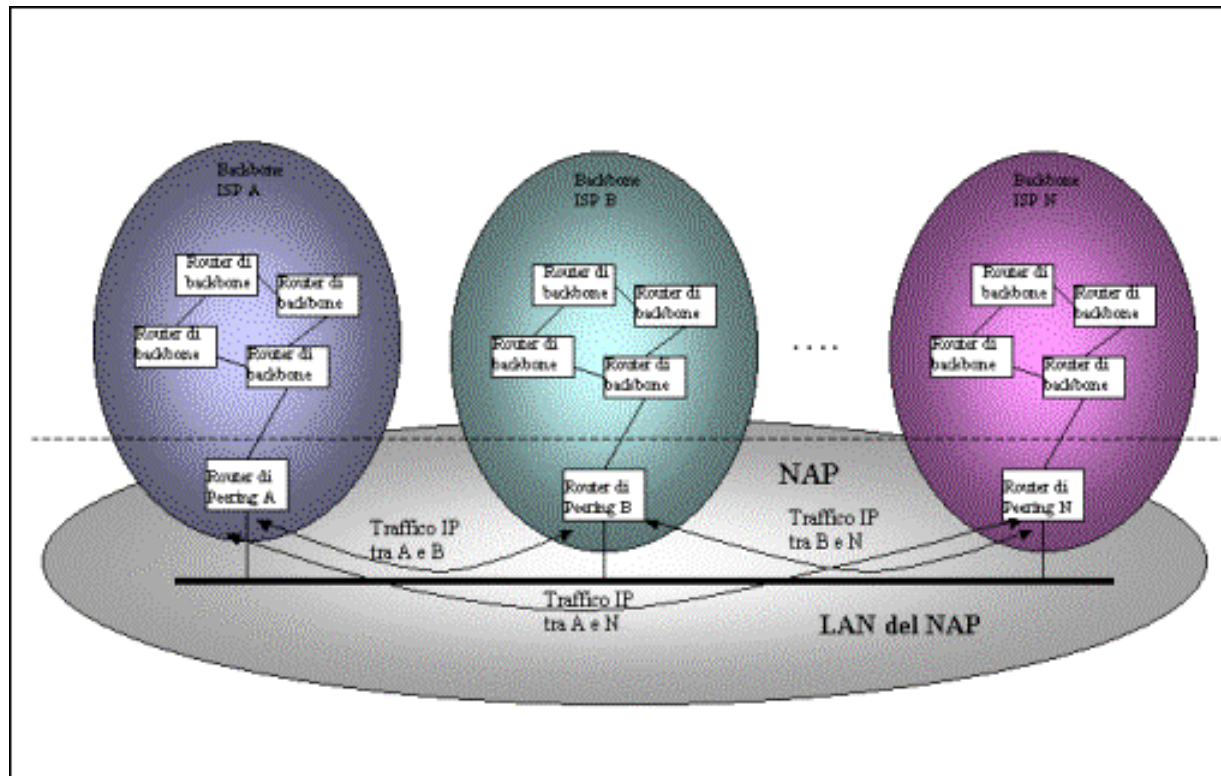
Neutral access points (NAP)



- punto “neutrale” di scambio dati tra ISPs (passaggio da un backbone ad un altro)
- un backbone attestato su di un NAP ha un gateway verso quel NAP
- il NAP è quindi un insieme di gateways verso un insieme di backbones

Passaggio dati al NAP

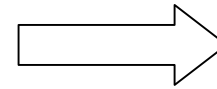
- lo scambio di dati tra diversi ISP avviene in base ai cosiddetti **accordi di peering**
- il passaggio di dati tra un backbone ed un altro al NAP è quindi **selettivo**



NAP Nautilus (Roma)

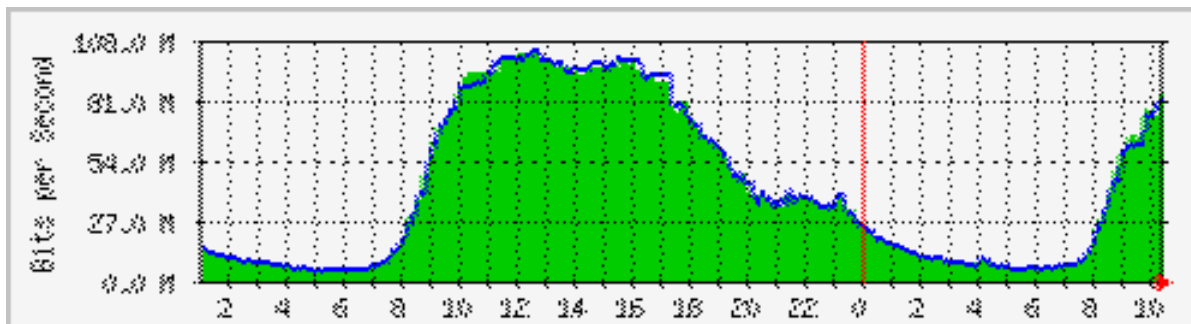
NAP gestito dal [CASPUR](#) (Consortium for the Applications of Supercomputation for University and Research) all'interno della Università "La Sapienza"

Network operator partecipanti a Nautilus

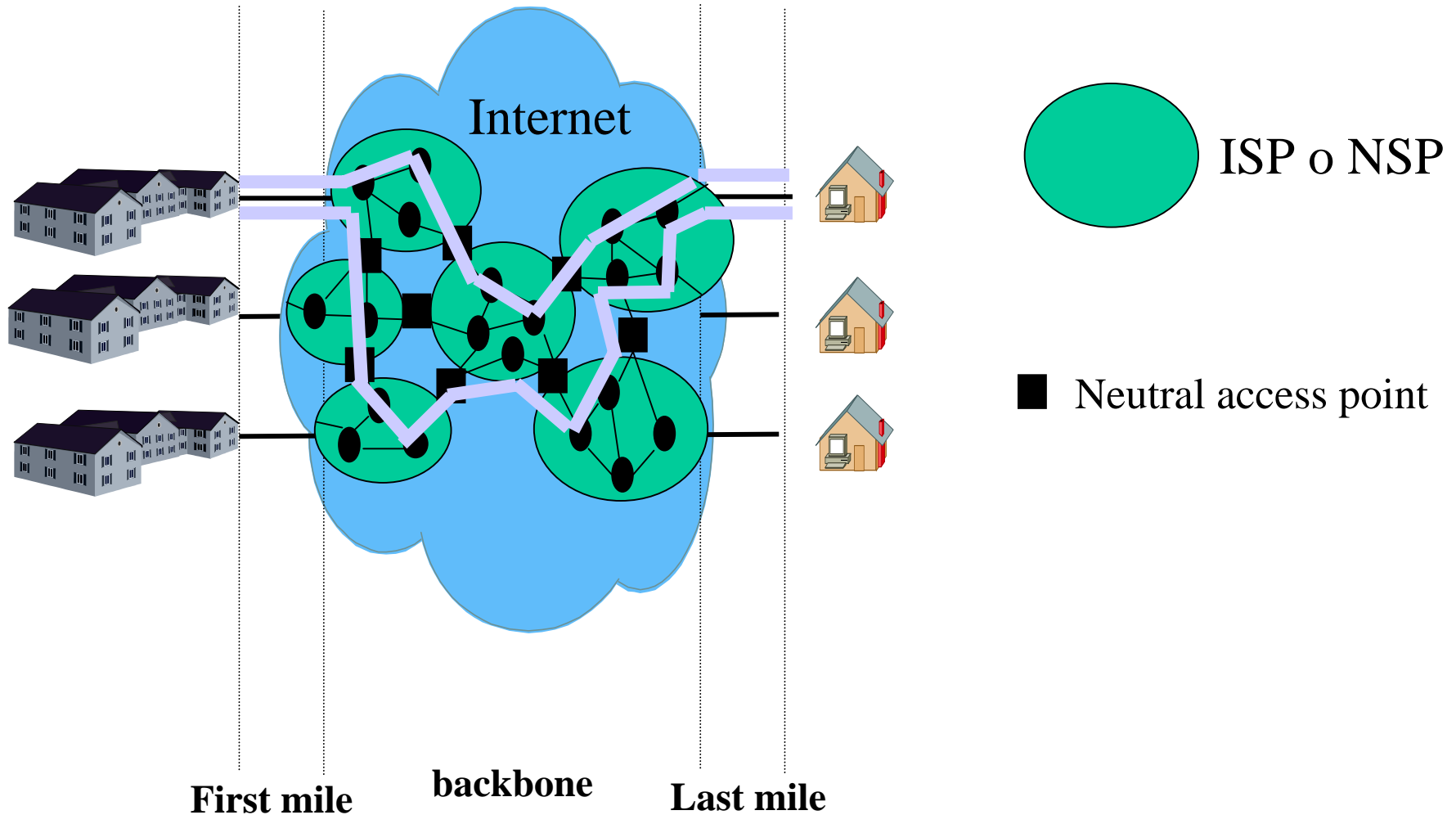


1. [Agora'](#)
2. [CASPUR](#)
3. [Cybernet](#)
4. [GARR](#)
5. [MClick](#)
6. [Unidata](#)
7. [InterBusiness](#)
8. [Unisource](#)
9. [Pronet](#)
10. [Infostrada](#)
11. [Wind](#)
12. [Tiscali](#)
13. [UUnet](#)
14. [Cubecom](#)
15. [Atlanet](#)
16. [Galactica](#)
17. [Postecom](#)
18. [Edisontel](#)

Traffico giornaliero



Architettura a tre livelli di Internet



- le tre zone sono potenziali colli di bottiglia
- possibile impatto sulla scelta dell'ISP

Trattamento dei colli di bottiglia

Soluzioni hardware

- first mile, last mile: aumentare la banda che connette al provider
- backbone: dipende dal miglioramento delle infrastrutture di rete dei singoli ISP (non controllabile dagli utenti finali)

Soluzione software

- caching trasparente di pagine vicino a dove risiede l'utente (e.g. AKAMAI)
- in questo modo si spera che l'utente possa accedervi con larga banda
- si tenta anche di diminuire il traffico

Nota: idea di soluzione simile a quella della gerarchia di caching delle memorie nei sistemi operativi

Modello OSI (Open System Interconnection)

ESEMPIO DI PROFILO DEI PROTOCOLLI PER IL PIANO UTENTE (commutazione di pacchetto)



ESEMPIO DI PROFILO DEI PROTOCOLLI PER IL PIANO UTENTE (commutazione di circuito)



Funzionalità dei vari livelli

Livello 1 - Fisico

- gestisce la trasmissione dei segnali e si occupa della sincronizzazione tra componenti elettronici

Livello 2 - Data Link

- gestisce la correttezza della trasmissione (utilizzo di codici a rivelazione e correzione d'errore)

Livello 3 - Rete

- gestisce l'instradamento delle informazioni

Livello 4 - Trasporto

- gestisce lo smistamento delle informazioni ed, eventualmente, l'affidabilità nella trasmissione

Livello 5 - Sessione

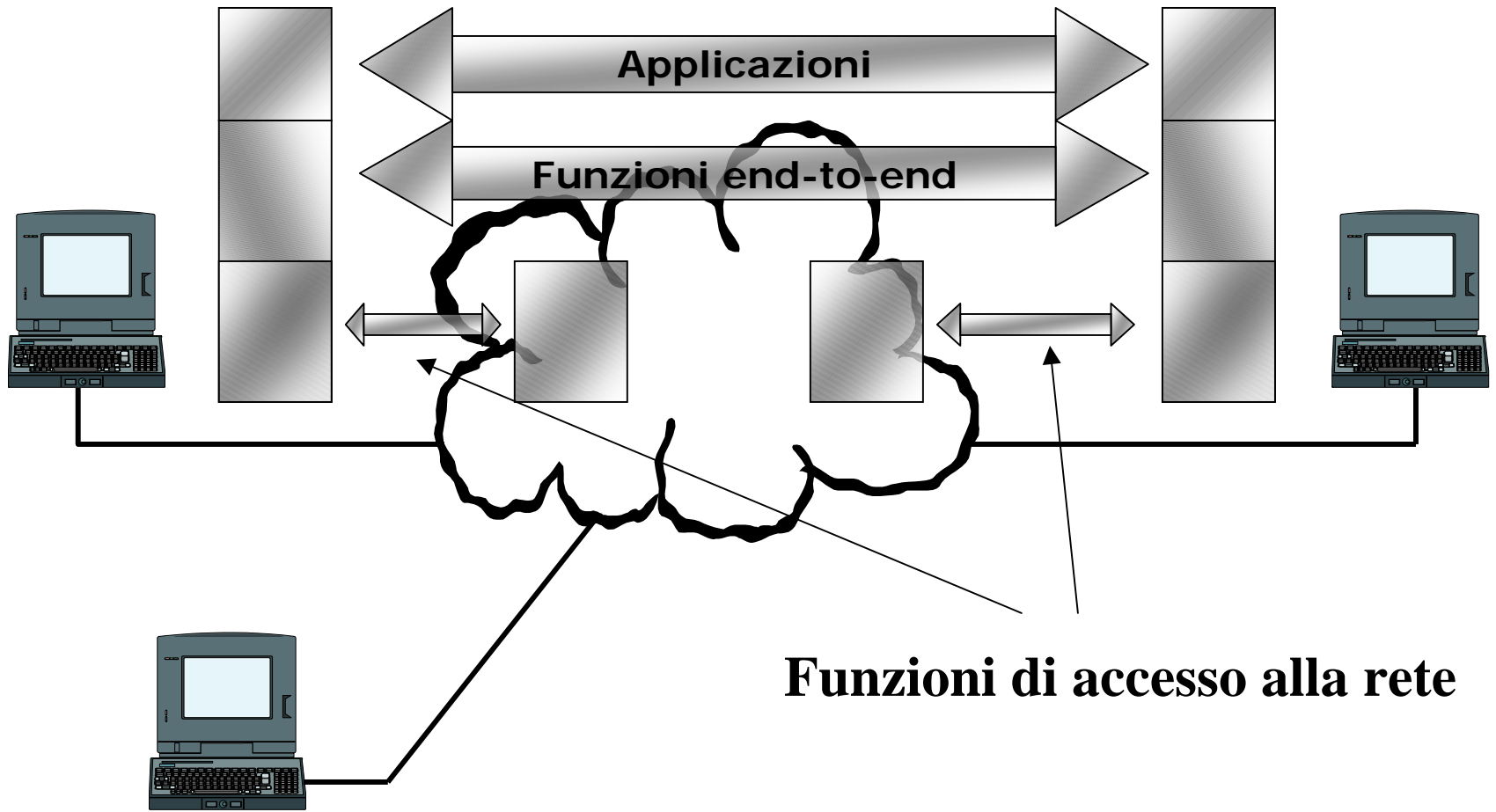
- gestisce una semantica di sessione tra le parti coinvolte nello scambio di informazioni

Livello 6 - Presentazione

- gestisce la modalità di presentazione dei dati verso il livello sovrastante

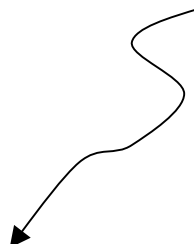
Livello 7 - Applicazioni

Rete geografica di calcolatori



Struttura a tre livelli di una rete di calcolatori

API



Area Applicativa

Interoperabilità trasporto dell'informazione

TRASPORTO

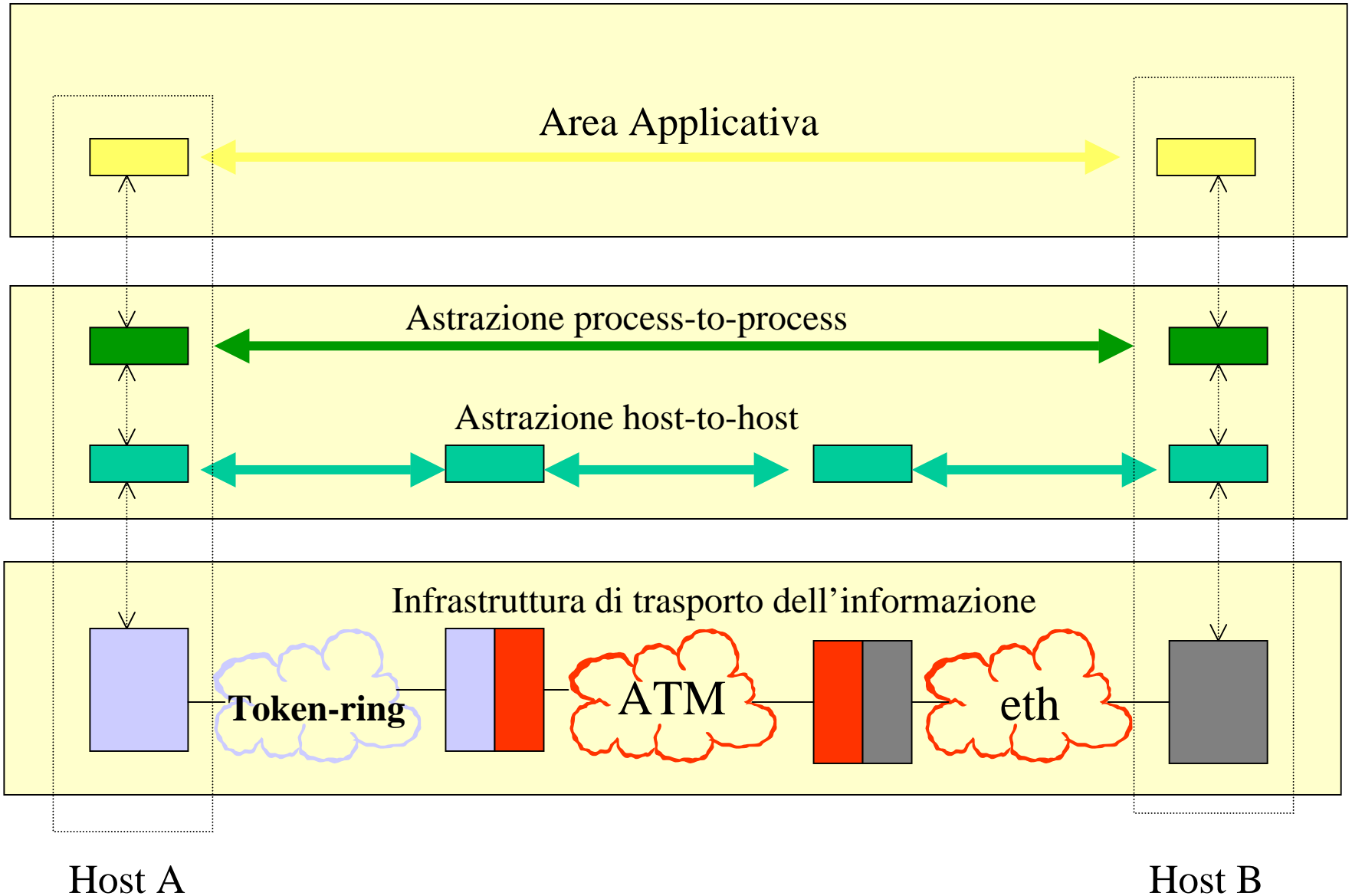
RETE

Infrastruttura di trasporto dell'informazione

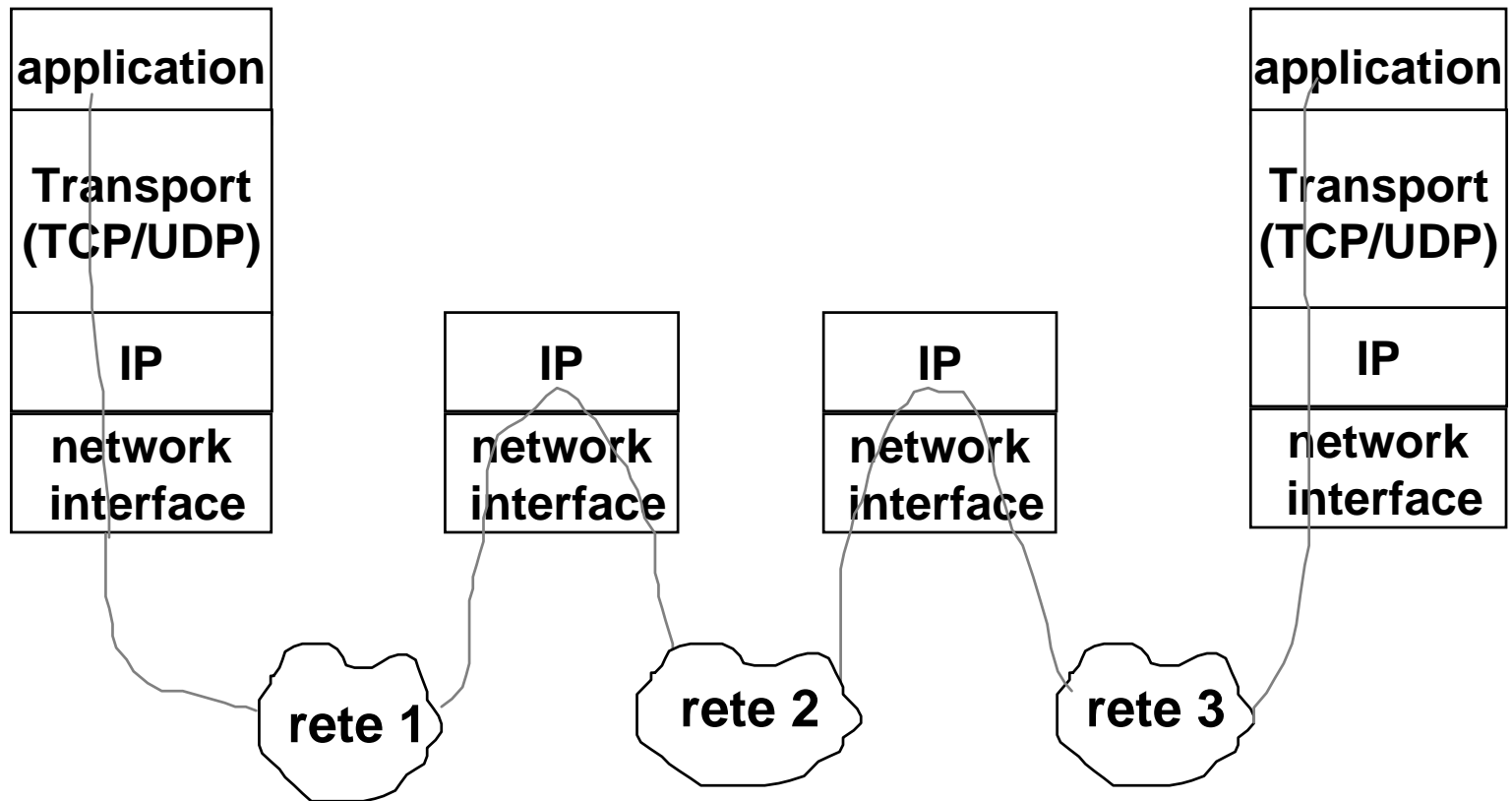
LINK

FISICO

Rete geografica di calcolatori



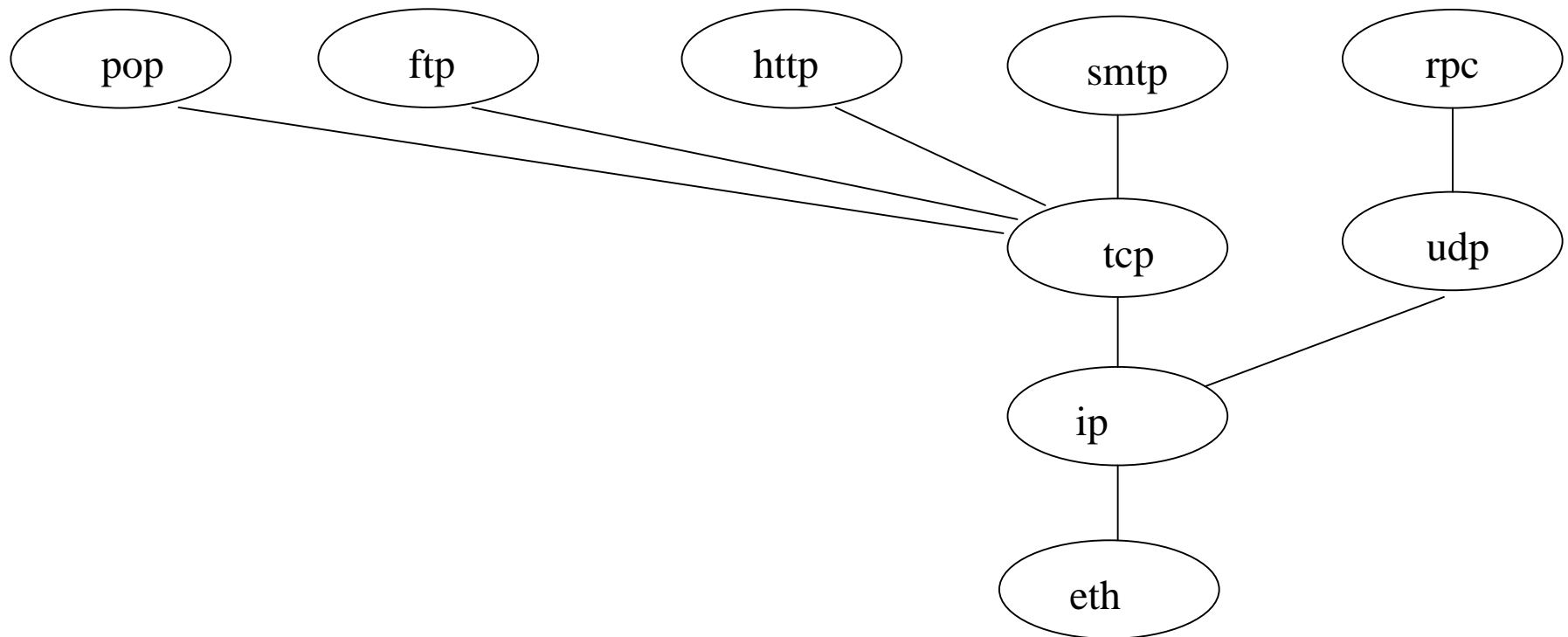
Architettura TCP/IP



- IP virtualizza omogeneità di gestione delle informazioni tra reti distinte che compongono Internet

Protocol Stack: esempi

- http= hyper text tranfer protocol
- smtp= simple mail transfer protocol
- rpc= remote procedure call
- ftp = file transfer protocol
- pop = post office protocol



Indirizzamento

Area Applicativa

Indirizzamento DNS "www.uniroma1.it"

Interoperabilità trasporto
dell'informazione

Indirizzamento IP "151.100.16.1"

Infrastruttura di trasporto
dell'informazione

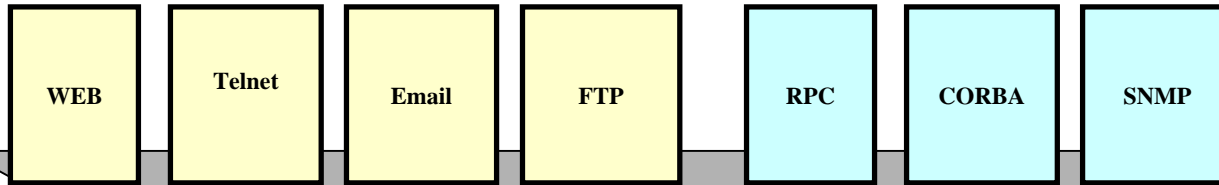
Indirizzamento MAC "ABC123578ABB"

- l'indirizzo MAC è il **nome unico** che contraddistingue ogni scheda di rete prodotta nel mondo
- la standardizzazione fa sì che non possano esistere due schede di rete con lo stesso indirizzo MAC

Architettura completa

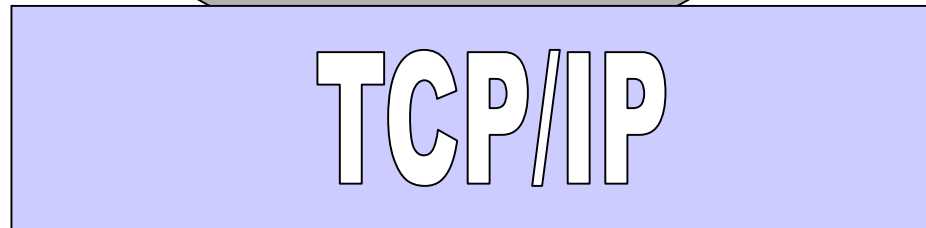
Applicazioni di base

Supporto per interoperabilità applicativa



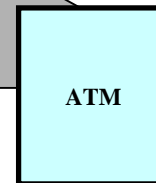
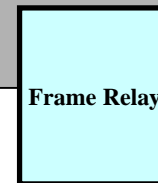
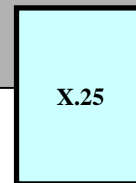
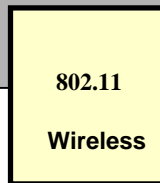
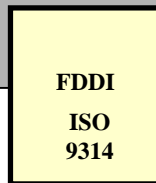
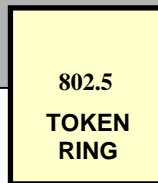
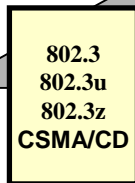
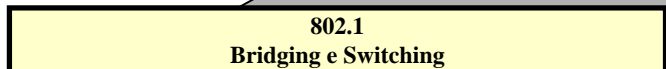
Area delle applicazioni

Interoperabilità di trasporto dell'informazione



Process-to-process

Host-to-host



Reti Locali

Backbone

Il protocollo IP

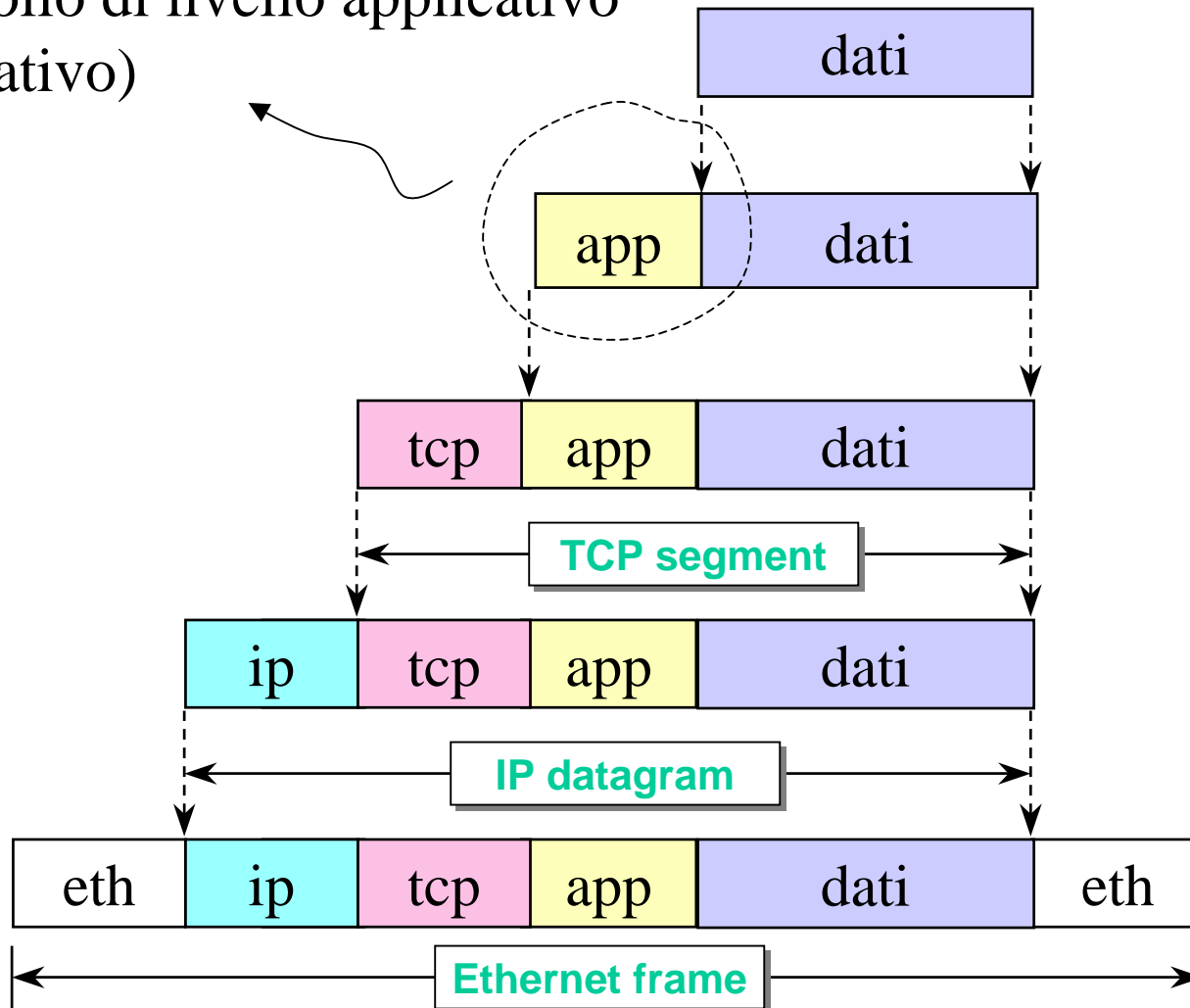
- IP e' una grande coperta che nasconde ai protocolli sovrastanti tutte le disomogeneità dell'infrastruttura di trasporto dell'informazione
- per far questo necessita di due funzionalità di base:
 - indirizzamento di rete (indirizzi omogenei a dispetto della rete fisica sottostante)
 - instradamento dei pacchetti (Routing) - capacità di inviare pacchetti da un host ad un altro utilizzando gli indirizzi definiti al punto precedente

Proprietà di IP

- senza connessione (datagram)
- consegna best effort
 - I pacchetti possono perdersi
 - I pacchetti possono essere consegnati fuori sequenza
 - I pacchetti possono essere duplicati
 - I pacchetti possono subire ritardi arbitrari

Pacchetti IP

protocollo di livello applicativo
(facoltativo)



Trasmissione IP

In Trasmissione, IP

- riceve il segmento dati dal livello di trasporto

Segmento dati

- inserisce header e crea datagram

IP

Segmento dati

- applica l'algoritmo di routing
- invia i dati verso l'opportuna interfaccia di rete

In Ricezione, IP

- consegna il segmento al protocollo di trasporto individuato

Segmento dati

- se sono dati locali, individua il protocollo di trasporto, elimina l'intestazione

IP

Segmento dati

- verifica la validità del datagram e l'indirizzo IP
- riceve i dati dalla interfaccia di rete

Indirizzamento IP



numero che identifica
la rete locale di destinazione
su Internet

numero che identifica
l'host di destinazione
sulla rete locale di destinazione

IPv6 lavorerà con indirizzi a 128 bit

Classi di indirizzi

Classe A

(0.0.0.0 - 127.255.255.255)

127.0.0.0 riservato



Classe B

(128.0.0.0 - 191.255.255.255)



Classe C

(192.0.0.0 - 223.255.255.255)



Classe D

(224.0.0.0 - 239.255.255.255)



Classe E

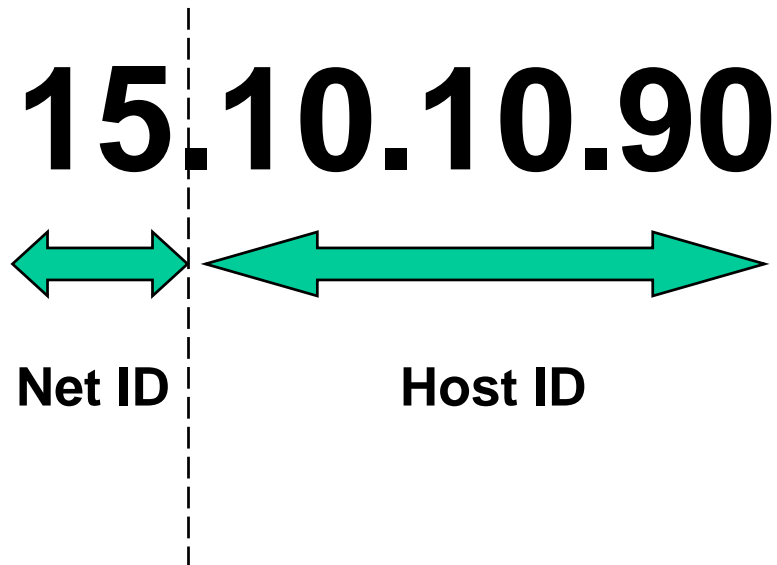
(240.0.0.0 - 255.255.255.254)



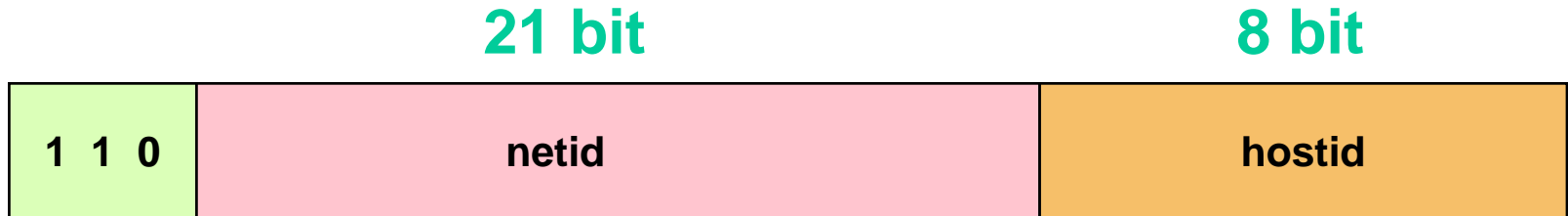
Indirizzi di classe A



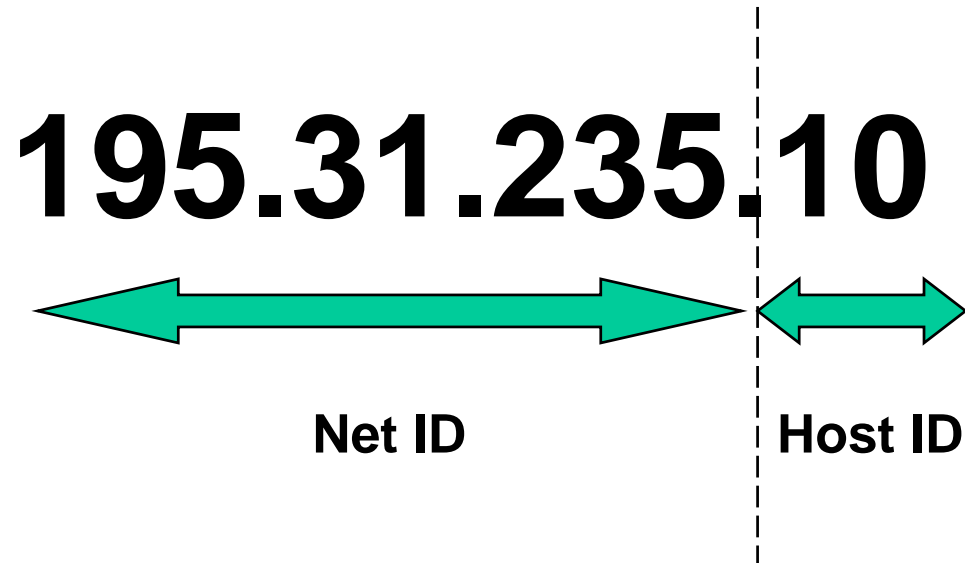
- esempio di indirizzo di classe A:



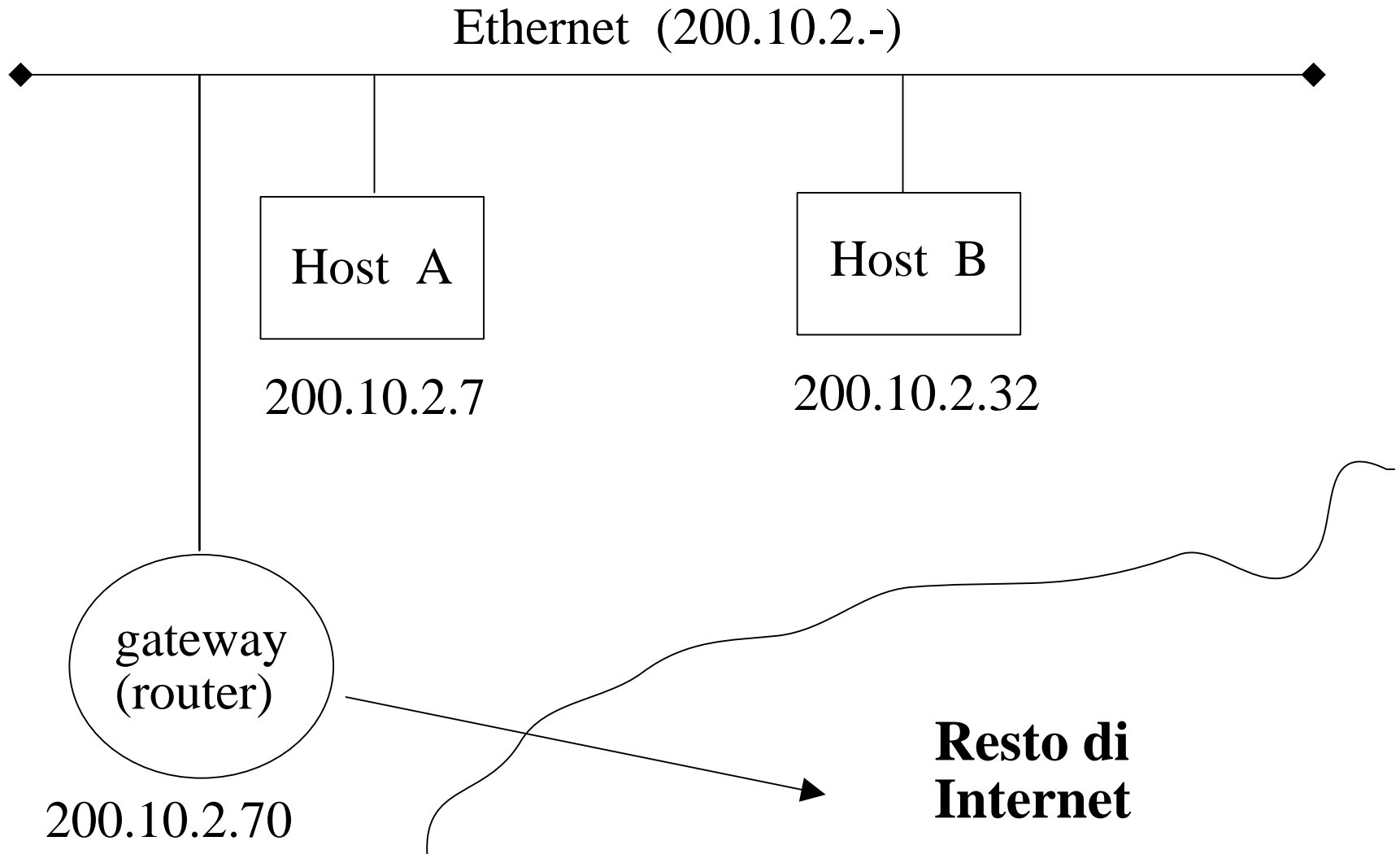
Indirizzi di classe C



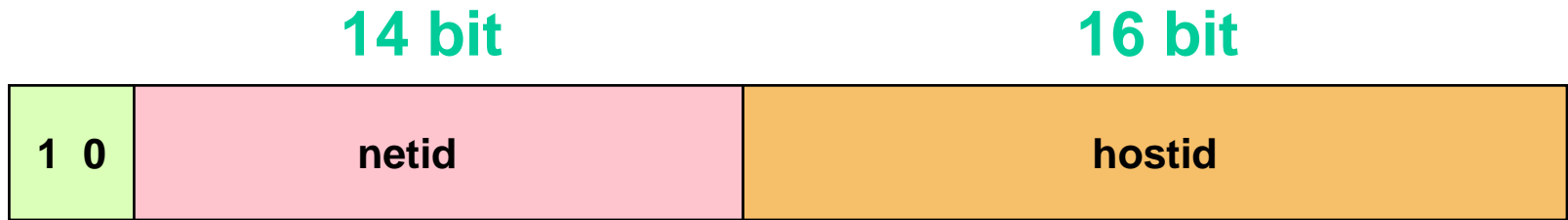
- esempio di indirizzo di classe C:



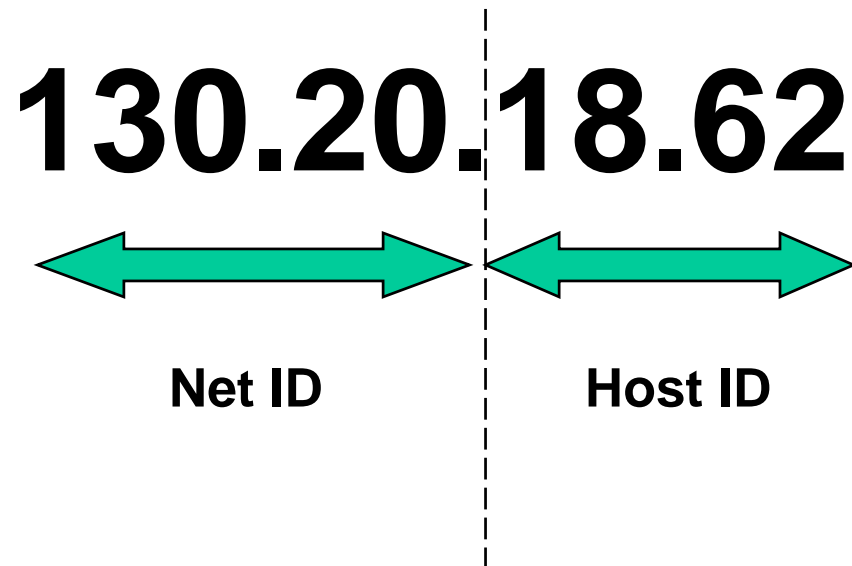
Distinzione degli host in classe C



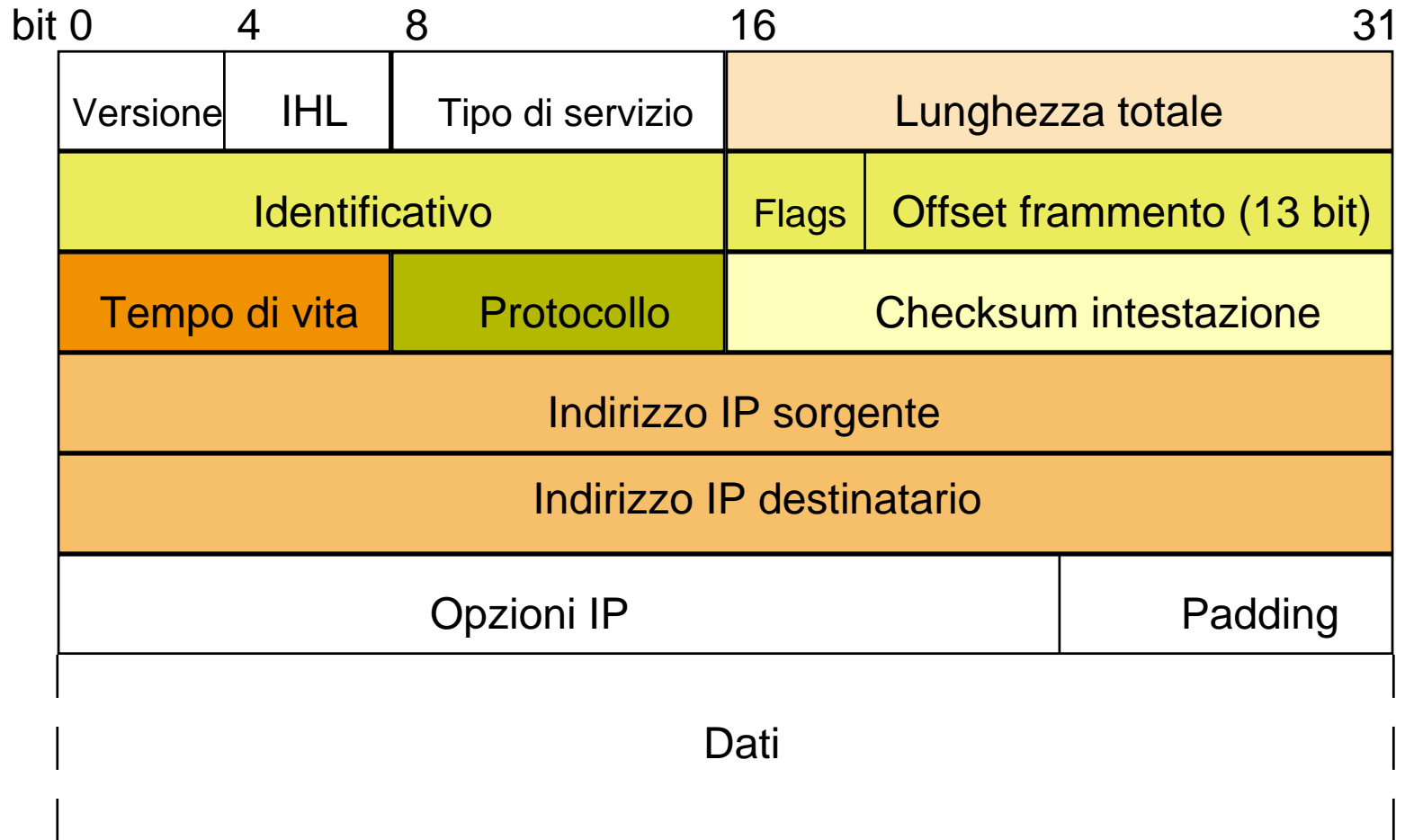
Indirizzi di classe B



- esempio di indirizzo di classe B:



Struttura del pacchetto IP



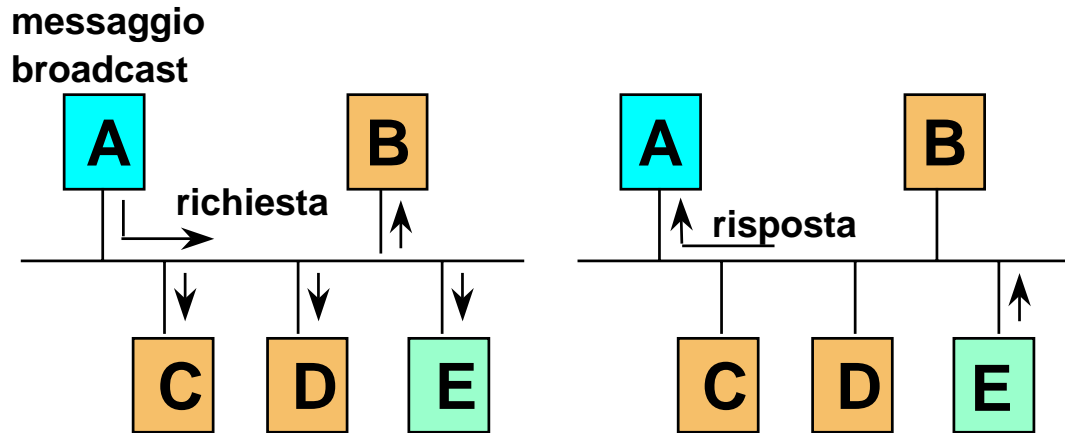
IHL = Internet Header Length

Identificativo/flags/offset di frammento = frammentazione di pacchetto

Tempo di vita = TTL (accoppiato con ICMP e traceroute)

Protocollo = TCP/UDP

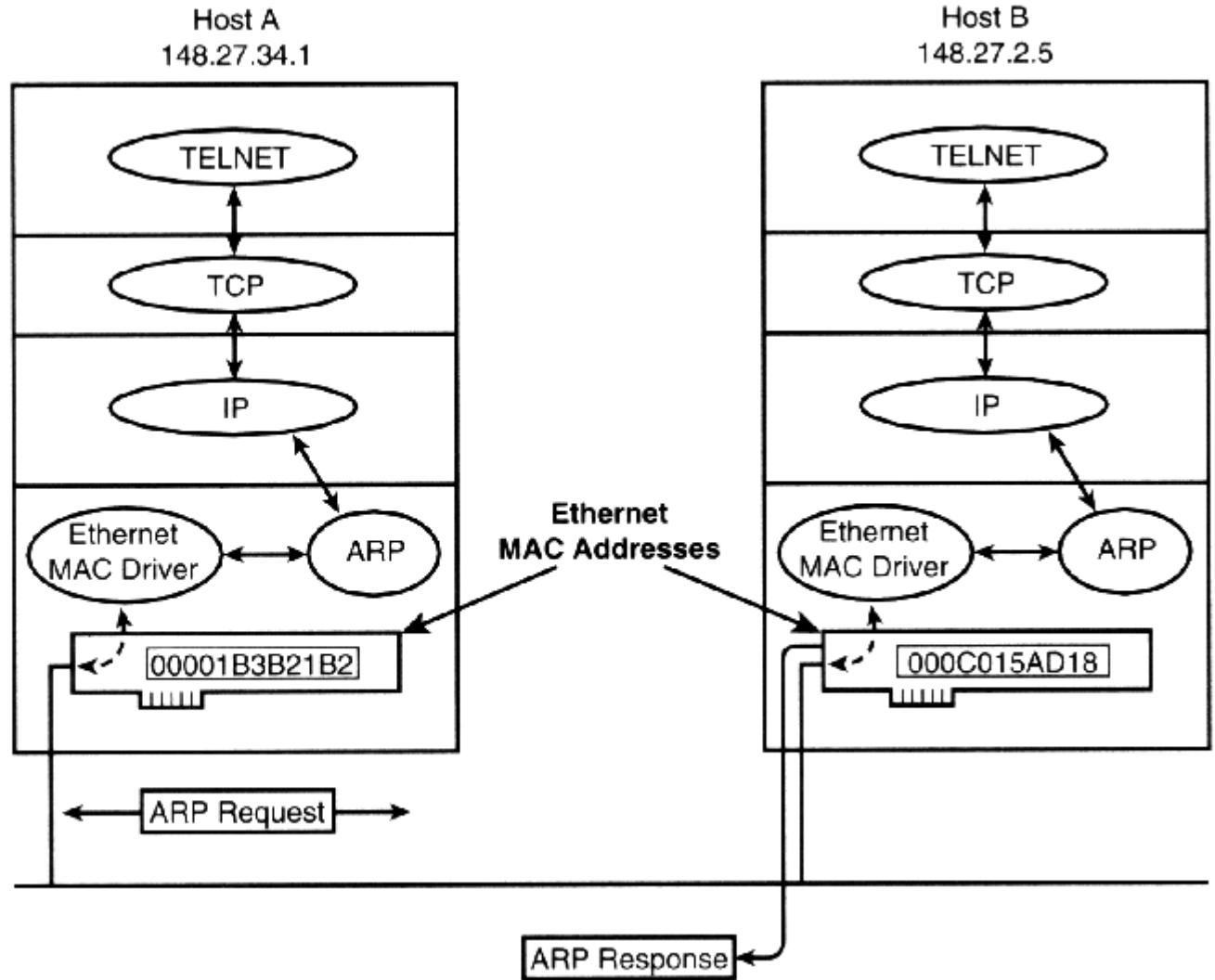
Address Resolution Protocol: ARP



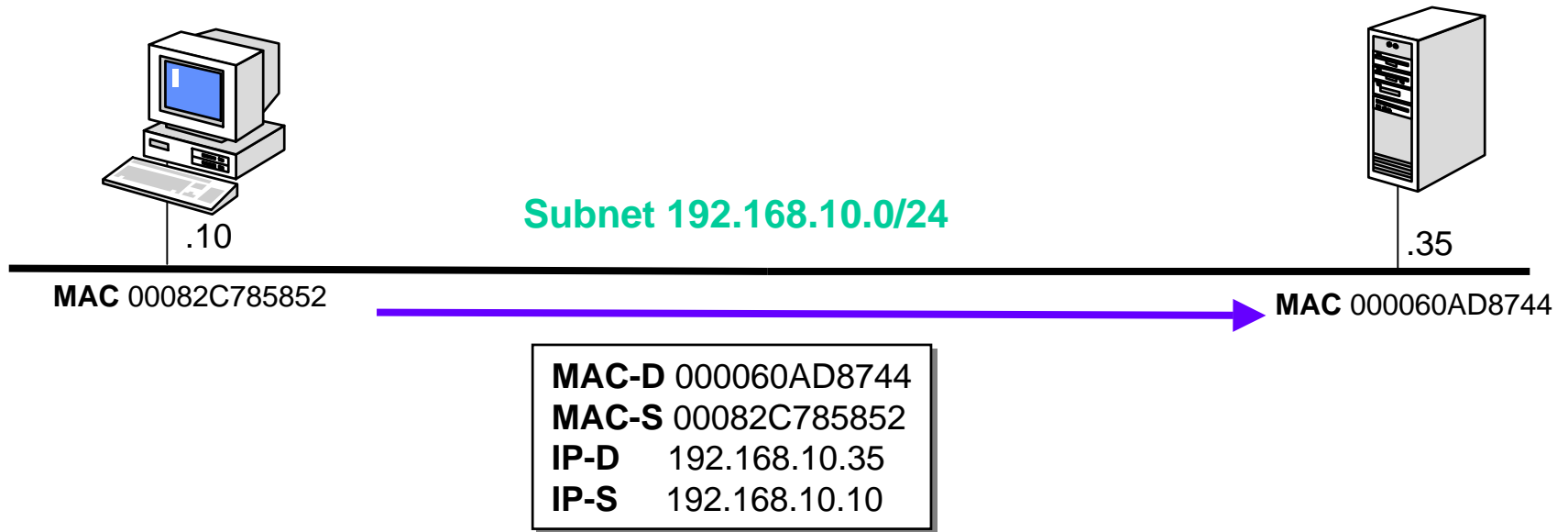
- serve a risolvere un indirizzo IP (che di per sè non significa nulla) in un indirizzo fisico (che è l'unico dato che l'host può capire davvero)
- mittente e il destinatario sulla stessa sottorete:
 1. Il mittente verifica la presenza nella cache ARP della coppia IP/MAC. Se la trova può inviare le informazioni
 2. In caso negativo diffonde un messaggio broadcast la cui intestazione comprende l'indirizzo Ip del destinatario
 3. L'host con tale indirizzo IP risponde notificando il suo indirizzo MAC
- mittente e destinatario su reti diverse: il mittente non cerca l'indirizzo Mac del destinatario, ma solo quello del suo gateway di default
- solo l'ultimo gateway dovrà scoprire l'indirizzo fisico dell'host di destinazione per potergli recapitare le informazioni.

FIGURE 4.9.

Resolution of an IP address into its MAC address using ARP.

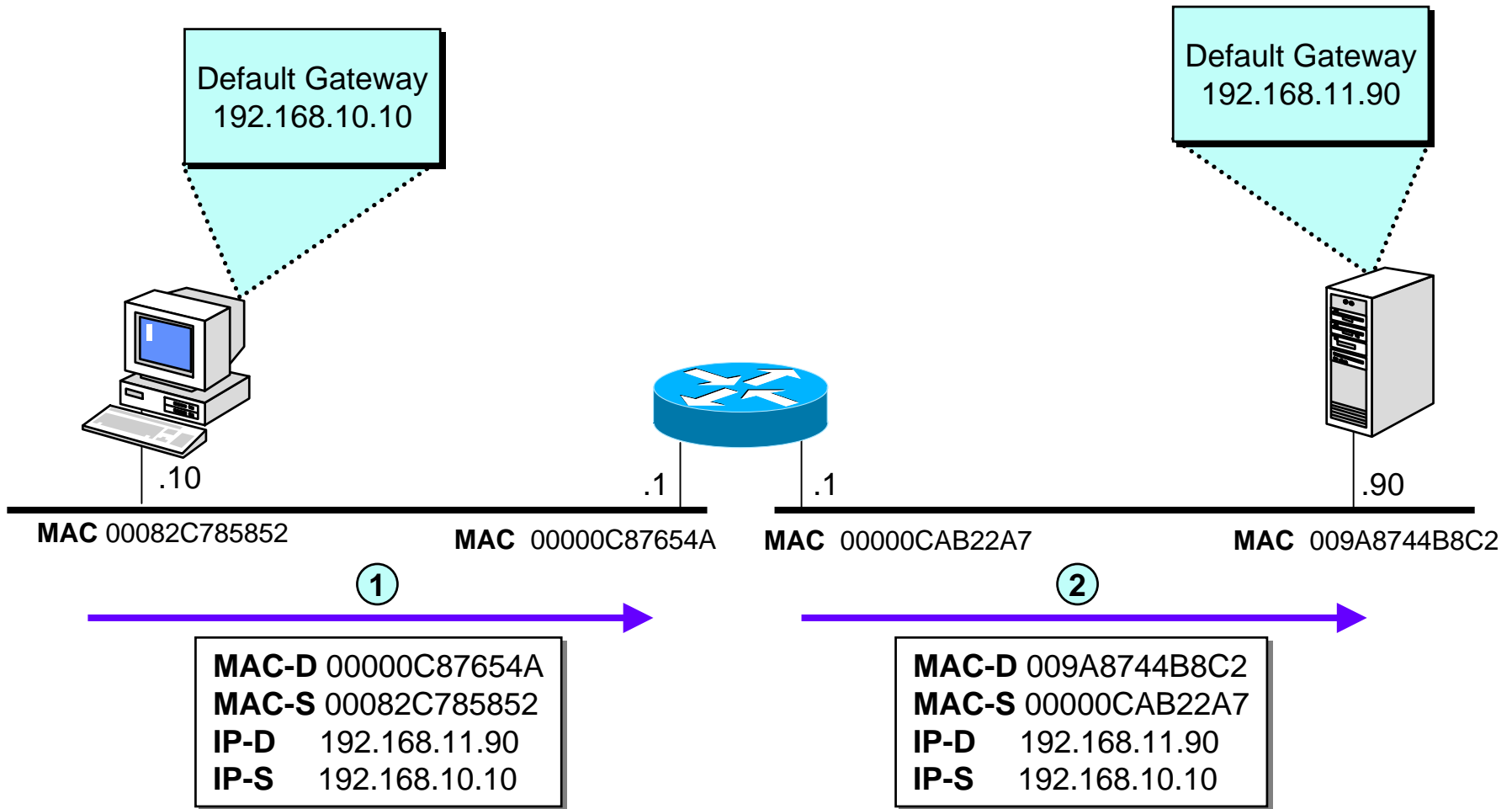


Forwarding diretto: esempio



- il mittente genera un frame contenente l'indirizzo MAC dell'host destinazione

Forwarding indiretto: esempio



Il livello del Trasporto

Supporta realmente trasferimento di dati tra processi (il livello rete supporta solo trasferimento tra host)

Protocolli standard

TCP (Transmission Control Protocol)

- orientato alla connessione (instaurazione e chiusura esplicita)
- connessione full duplex (e' possibile trasferimento contemporaneo nelle due direzioni della connessione)
- consegna affidabile ed ordinata delle informazioni

UDP (User Datagram Protocol)

- non orientato alla connessione
- consegna non affidabile
- consegna non ordinata

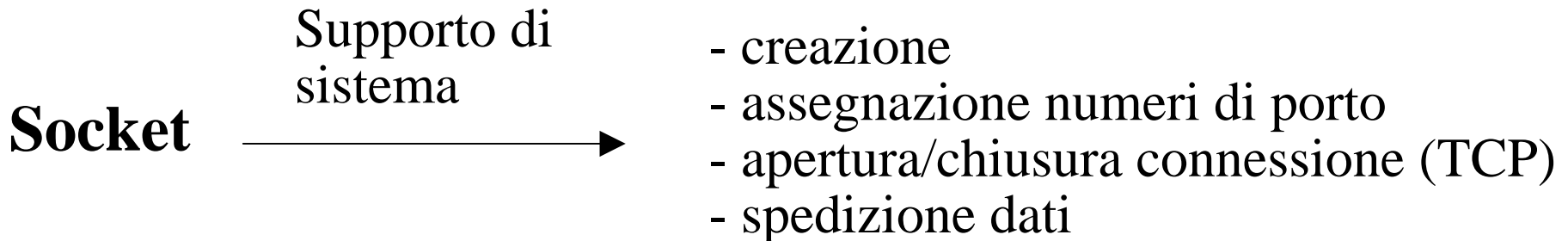
Indirizzamento al livello del trasporto

Composizione di un indirizzo

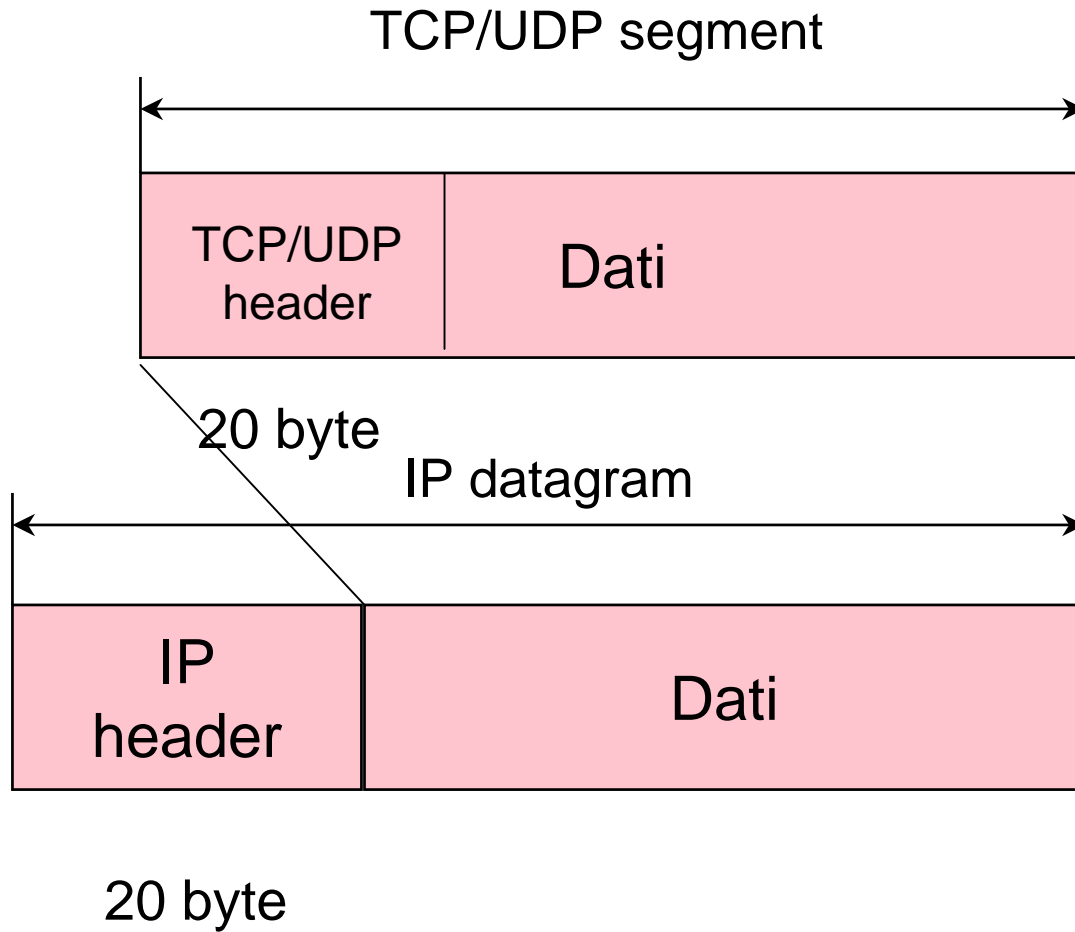
<numero IP, numero di porto>

- l'utilizzo dei numeri di porto permette lo smistamento del traffico di dati in ingresso ad un host
 - quindi la possibilita' di supporto alla ricezione per applicazioni (processi) distinti
-

Costrutti di sistema per il supporto del livello del Trasporto



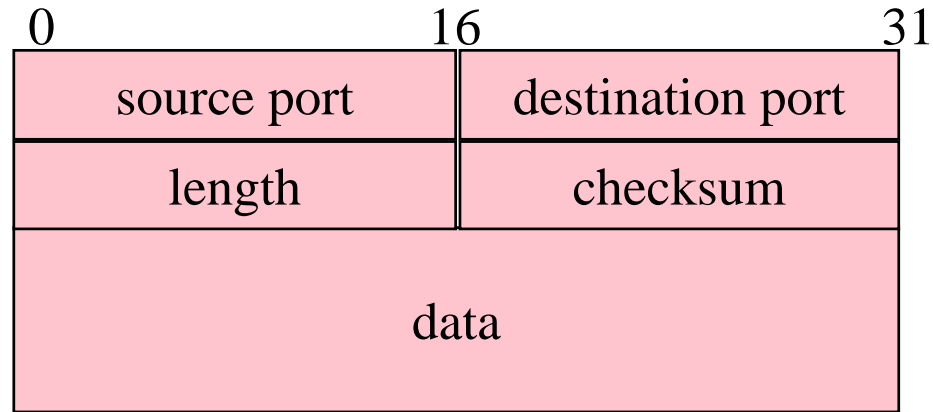
Mecchansismo di Trasporto



Trasporto UDP

Il pacchetto UDP viene imbustato in IP ed indirizzato con il campo protocol pari a 17

L'intestazione di UDP è lunga 8 byte

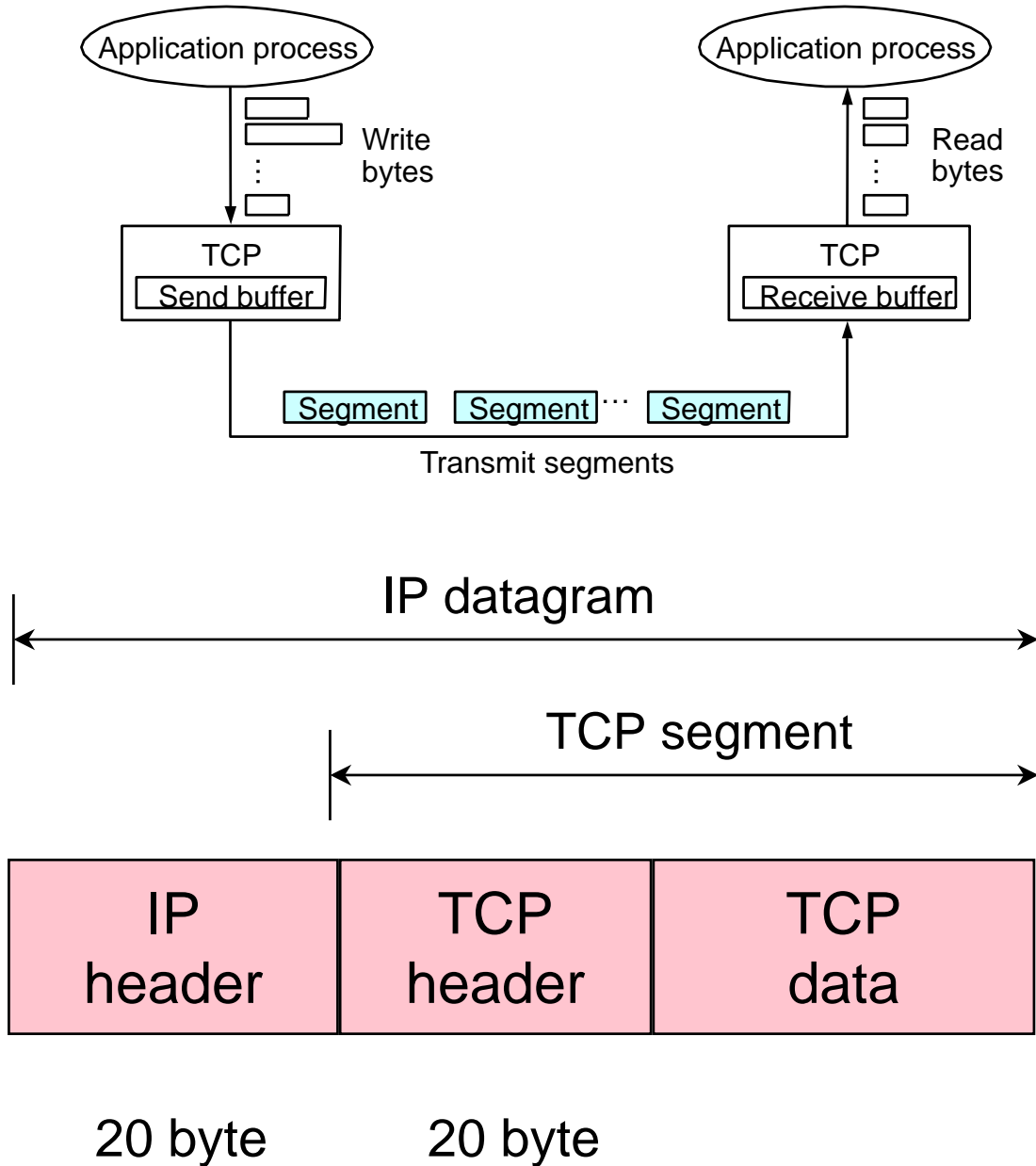


port number: funzioni di smistamento dei dati

length, è la lunghezza in byte del pacchetto UDP, header e dati; il minimo valore per questo campo è di 8 byte, quando la parte dati è vuota; questa informazione è ridondante perché nell'intestazione IP è presente il campo length, relativo alla lunghezza di tutto il pacchetto IP; visto che l'intestazione UDP ha una lunghezza fissa di 8 byte, la lunghezza della parte dati potrebbe essere ricavata sottraendo al contenuto del campo length dell'header IP 8 byte;

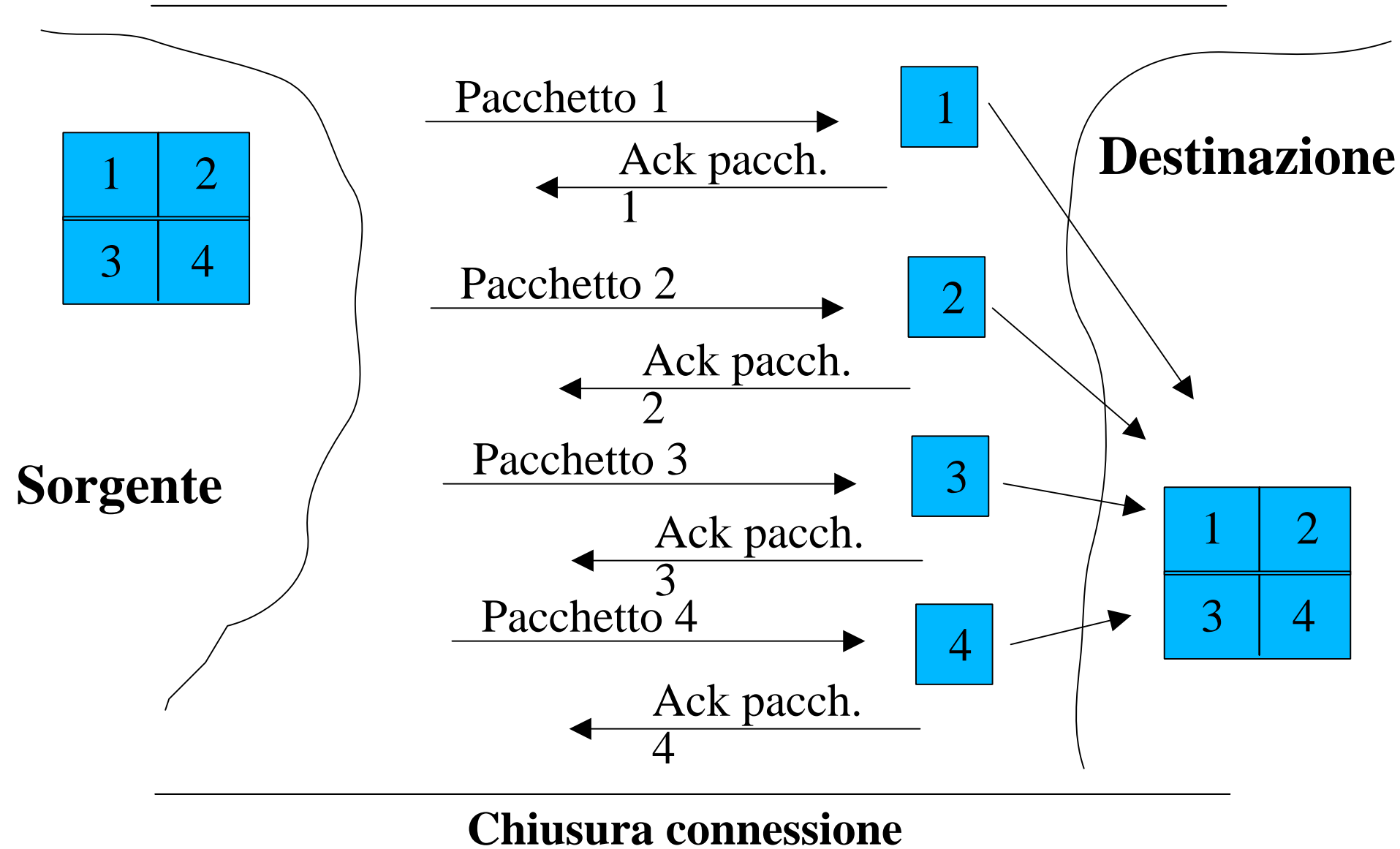
checksum, campo per il controllo di errore, che copre tutto il pacchetto UDP, header e dati; in realtà oltre al pacchetto UDP, il checksum è applicato anche ad una parte dell'intestazione IP, composta tra l'altro dagli indirizzi IP sorgente e destinazione e dal campo protocol, detta UDP-pseudo-header

Trasporto TCP

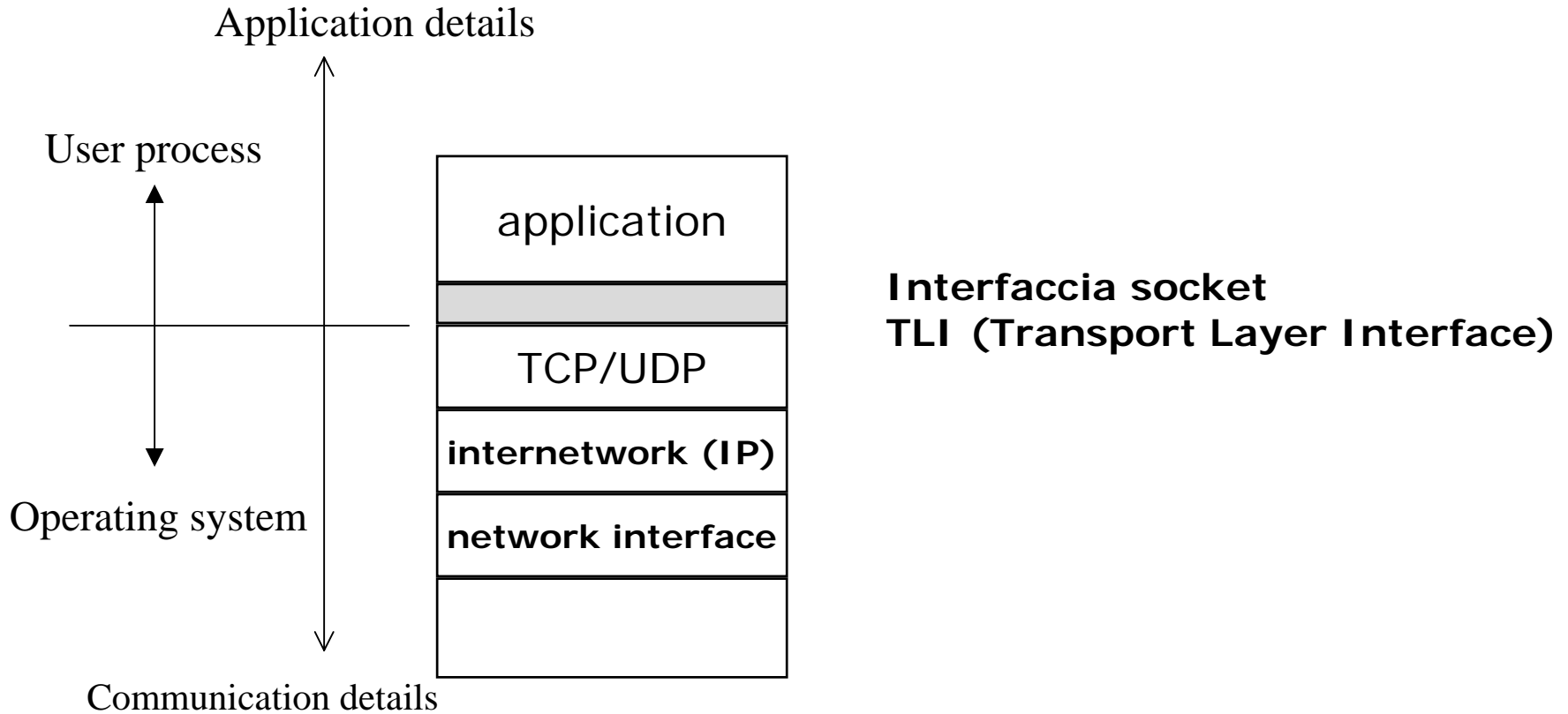


Flusso del traffico TCP: un esempio

Apertura connessione



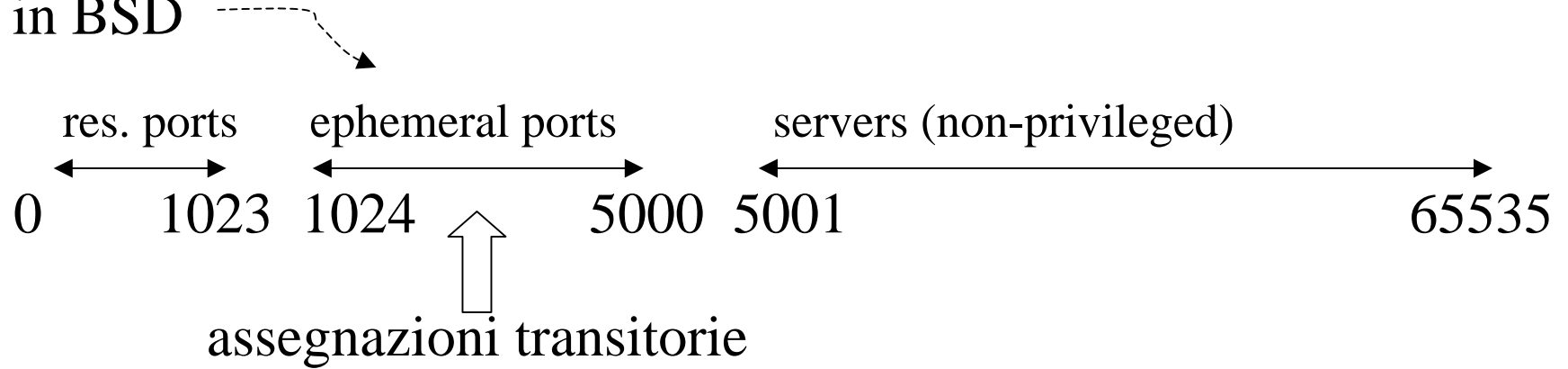
Sockets



- end-point determinati da:
 - indirizzo dell'host (IP): livello rete
 - Port number: livello di trasporto

Port numbers

- l'utilizzo dei numeri di porto da parte del sistema operativo varia con la versione del sistema
- in BSD



- servizi ben noti lavorano sui seguenti port numbers:
 - ftp 21/tcp
 - telnet 23/tcp
 - finger 79/tcp
 - snmp 161/udp

UNIX sockets

```
int socket(int domain, int type, int protocol)
```

Descrizione invoca la creazione di un socket

Argomenti

- 1) domain: specifica del dominio di comunicazione relativamente al quale può operare il socket
- 2) type: specifica la semantica della comunicazione associata al socket
- 3) protocol: specifica il particolare protocollo di comunicazione per il socket

Restituzione un intero positivo (descrittore di socket) in caso di successo; -1 in caso di fallimento

Domini

- AF_INET Internet protocols
- AF_UNIX Unix internal protocols
(not really communication, but IPC)
- AF_NS Xerox NS protocols
- AF_IMPLINK IMP link layer (Interface Message Processor)

dove AF = address family

Notazioni equivalenti

- PF_INET
- PF_UNIX
- PF_NS
- PF_IMPLINK

dove PF = protocol family

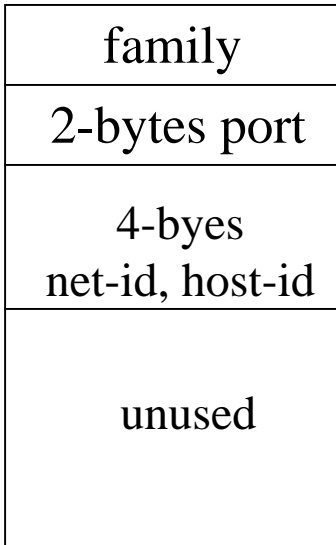
Formato degli indirizzi

in `<sys/socket.h>`

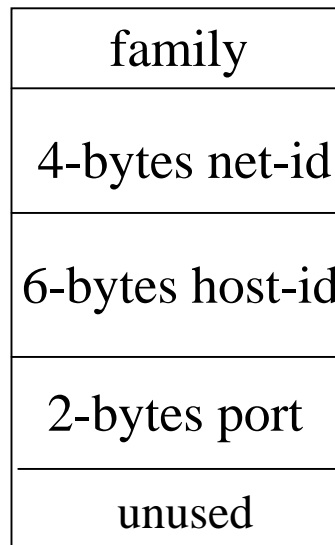
```
struct sockaddr{
    u_short  sa_family; /* address family */
    char     sa_data[14]; /* up to 14 bytes of
                           protocol specific address */
}
```

adeguato per `AF_INET` e `AF_NS`

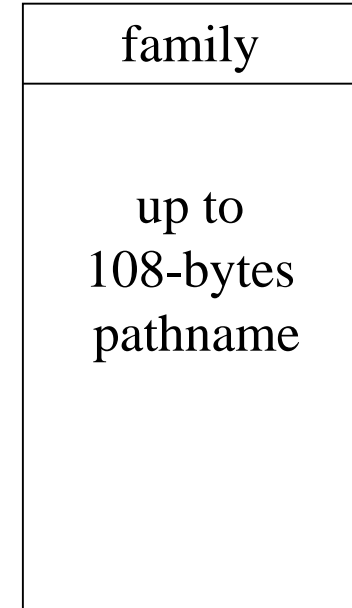
struct sockaddr_in



struct sockaddr_ns



struct sockaddr_un



Tipo di comunicazione

- SOCK_STREAM streaming
- SOCK_DGRAM datagram
- SOCK_RAW raw data
- SOCK_SEQPACKET sequenced packet
- SOCK_RDM reliable delivery of messages (non implementato)

Protocolli di default

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	yes	TCP	SPP
SOCK_DGRAM	yes	UDP	IDP
SOCK_RAW		IP	yes
SOCK_SEQPACKET			SPP

nessun acronimo

Sequenced packet protocol

Selezione del protocollo

- alcune volte, fissata la coppia (domain,type), è possibile scegliere tra più protocolli di comunicazione
- altre volte invece fissata tale coppia esiste un solo protocollo di comunicazione valido
- il parametro protocol specifica quale protocollo si vuole effettivamente usare una volta fissata tale coppia qualora esista una possibilità di scelta
- il valore 0 per il parametro protocol indica che si vuole utilizzare il protocollo di default, o eventualmente l'unico disponibile per quella coppia (domain,type)

Combinazioni ammissibili per AF_INET

domain	type	protocol	actual protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)

ICMP = Internet Control Management Protocol
ha funzioni di monitoring/gestione del livello IP

La definizione delle macro IPPROTO_XXX è nell'header
<netinet/in.h>

Coppie di sockets

```
int socketpair(int domain, int type, int protocol, int sockvec[2])
```

Descrizione invoca la creazione di un socket

Argomenti

- 1) domain: specifica del dominio di comunicazione relativamente al quale può operare il socket
- 2) type: specifica la semantica della comunicazione associata alla socket
- 3) protocol: specifica il particolare protocollo di comunicazione per il socket
- 4) sockvec[2]: coppia di descrittori di socket restituiti

Restituzione -1 in caso di fallimento

- opera sia su `SOCK_STREAM` che `SOCK_DGRAM`, in ogni caso solo sul domino `AF_UNIX`

Assegnazione di un indirizzo

```
int bind(int ds_sock, struct sockaddr *my_addr, int addrlen)
```

Descrizione invoca l'assegnazione di un indirizzo al socket

Argomenti

- 1) ds_sock: descrittore di socket
- 2) *my_addr: puntatore al buffer che specifica l'indirizzo
- 3) addrlen: lunghezza (in byte) dell'indirizzo

Restituzione -1 in caso di fallimento

- il terzo parametro serve per specificare la **taglia esatta** dell'indirizzo rispetto al dominio di interesse
- il buffer strutturato di tipo **sockaddr** è dimensionato in modo da poter contenere indirizzi appartenenti al dominio per cui la loro specifica richiede il massimo numer di byte (unica eccezione è il dominio AF_UNIX)

Buffer strutturato per AF_INET

in <netinet/in.h>

```
struct in_addr {
    u_long    s_addr;    /* 32net-id/host-id */
                    /* network byte ordered */
}

struct sockaddr_in {
    short      sin_family; /* domain */
    u_short    sin_port;  /* 2-bytes port number */
    struct in_addr sin_addr; /* 4-bytes host-id */
    char       sin_zero[8]; /* unused */
}
```

- se **sin_port** è pari a 0, il sistema assegna un ephemeral port (non adeguato in caso di server)
- usare **bzero()** per evitare comportamenti non deterministici

Un esempio di assegnazione di indirizzo in AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

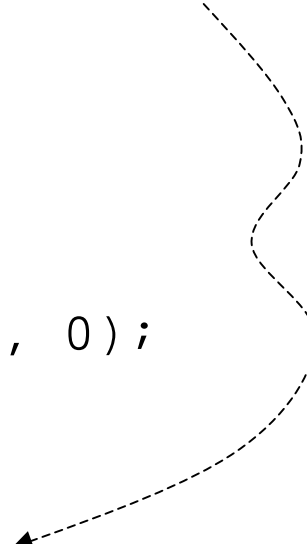
qualsiasi indirizzo
esterno è ammesso

```
void main() {
    int ds_sock;
    struct sockaddr_in my_addr;

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port       = 25000;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));
}
```



Un esempio di assegnazione di indirizzo in AF_UNIX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
```

**campi sun_family
e sun_path**



```
void main() {
    int ds_sock; int len;
    struct sockaddr_un my_addr;

    ds_sock = socket(AF_UNIX, SOCK_STREAM, 0);

    my_addr.sun_family = AF_UNIX;
    strcpy(my_addr.sun_path, "my_name");
    len = sizeof(my_addr.sun_path) +
          sizeof(my_addr.sun_family);

    bind(ds_sock, &my_addr, len);
}
```

Chiusura di un socket

- quando un processo non ha più bisogno di un dato socket per la comunicazione può chiudere tramite la chiamata `close()`
- il parametro della chiamata sarà il descrittore del socket che si vuole chiudere
- è da notare che quando un processo chiude un socket, il socket stesso viene rimosso solo qualora non vi sia alcun altro processo che possieda un descrittore valido per quel socket
- i descrittori di socket vengono trattati alla stregua di descrittori di file (tabella locale per processo)
- descrittori validi multipli possono essere originati per effetto della system call `fork()`

Un esempio

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```
void main() {
    int ds_sock;  char c;
    struct sockaddr_in my_addr;

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = 2500;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));

    if ( fork()!=0 ) close(ds_sock)
    else {
        while ( read(0,&c,1) == -1 );
        close(ds_sock)
    }
}
```

socket ancora
attivo



Attesa di connessioni (SOCK_STREAM)

```
int accept(int ds_sock, struct sockaddr *addr, int *addrlen)
```

Descrizione invoca l'accettazione di una connessione su un socket

Argomenti

- 1) ds_sock: descrittore di socket
- 2) *addr: puntatore al buffer su cui si copierà l'indirizzo del chiamante
- 3) *addrlen: puntatore al buffer su cui si scriverà la taglia dell'indirizzo del chiamante (compatibilità per domini)

Restituzione un intero positivo indicante il descrittore di un nuovo socket in caso di successo; -1 in caso di errore

- l'accettazione effettua lo switch della connessione su un nuovo socket
- il port number per il nuovo socket e' lo stesso del socket originale

Un esempio nel dominio AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```
void main() {
    int ds_sock, ds_sock_acc;
    struct sockaddr_in my_addr;
    struct sockaddr addr;
    int addrlen;

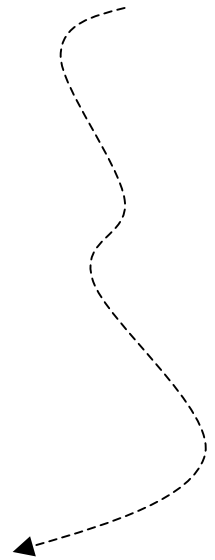
    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = 25000;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));
    ds_sock_acc = accept(ds_sock, &addr, &addrlen);

    close(ds_sock);
    close(ds_sock_acc);
}
```

switch della connessione



Backlog di connessioni

```
int listen(int ds_sock, int backlog)
```

Descrizione invoca l'impostazione orientativa del numero di connessioni pendenti

Argomenti

- 1) sock_ds: descrittore di socket
- 2) backlog: numero di connessioni da mantenere sospese

Restituzione -1 in caso di errore

- una connessione è pendente quando non può essere associata ad un socket destinazione che però esiste
- il backlog specificato tramite questa system call è orientativo nel senso che il sistema operativo potrebbe decidere di mantenere pendenti un numero più alto di connessioni rispetto al richiesto
- su alcune versioni di UNIX è necessario impostare un backlog prima di attendere qualsiasi connessione

Un esempio nel dominio AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#define BACKLOG 10

void main() {
    int ds_sock, ds_sock_acc, addrlen;
    struct sockaddr_in my_addr; struct sockaddr addr;

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = 25000;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));
    listen(ds_sock, BACKLOG);

    while(1) {
        ds_sock_acc = accept(ds_sock, &addr, &addrlen);
        close(ds_sock_acc);
    }
}
```

Connessione di un socket

```
int connect(int ds_socks, struct sockaddr *addr, int addrlen)
```

Descrizione invoca la connessione di un socket su un indirizzo

Argomenti

- 1) ds_sock: descrittore del socket da connettere
- 2) *addr: puntatore al buffer contenente l'indirizzo al quale connettere il socket
- 3) addrlen: la taglia dell'indirizzo al quale ci si vuole connettere

Restituzione -1 in caso di errore

- necessaria in caso di tipo di comunicazione SOCK_STREAM
- può essere usata anche in caso di comunicazione connectionless, ovvero SOCK_DGRAM, SOCK_RAW

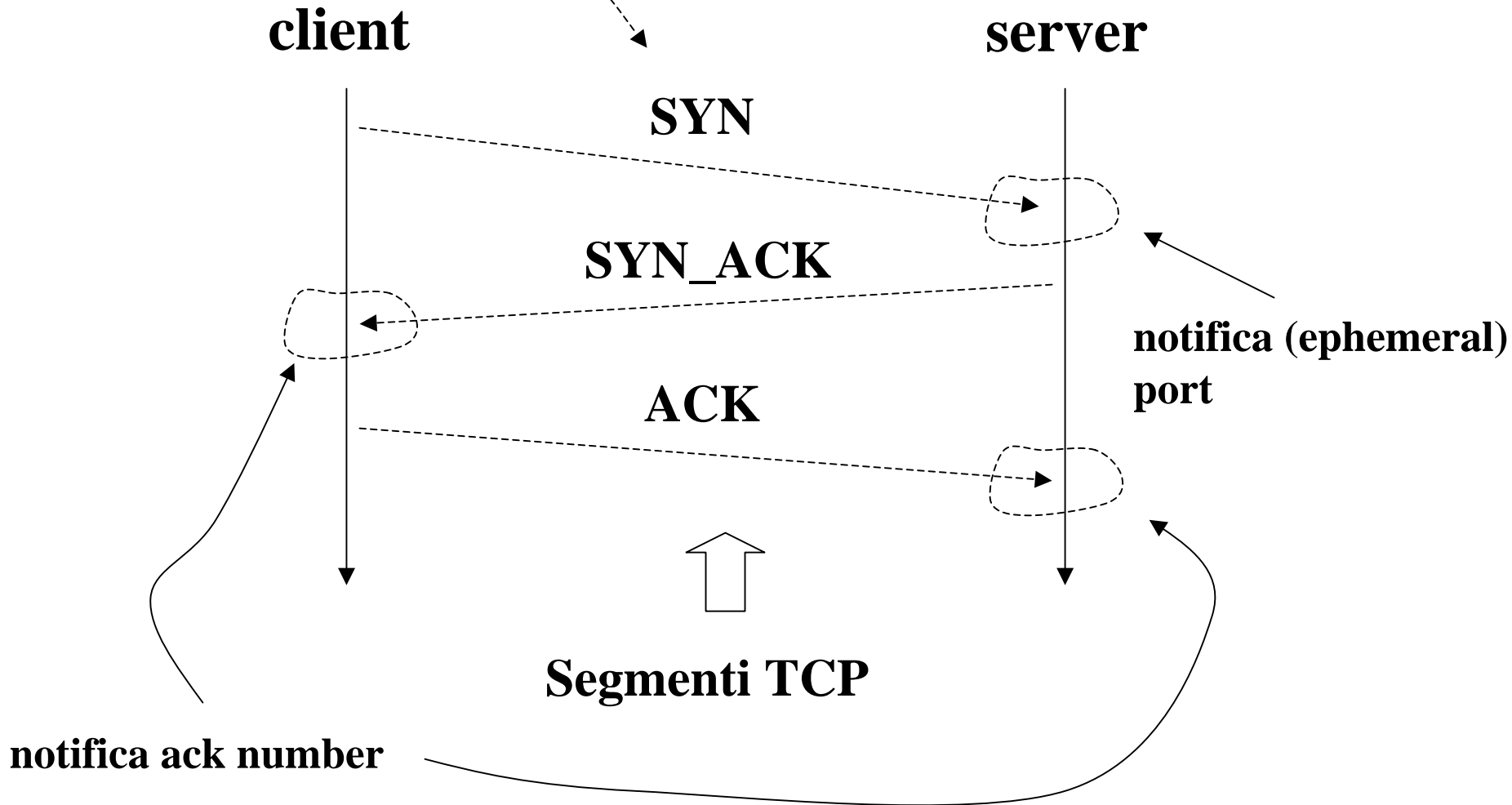
Connessione su comunicazione DGRAM

Vantaggi

- non c'è necessità di reimpostare ogni volta l'indirizzo del destinatario nello spedire un nuovo datagram
- le system call per la spedizione avranno quindi bisogno di identificare solo il punto di uscita dal sistema
- otteniamo come una “post box” univocamente associata ad una destinazione
- se il protocollo datagram usato supporta notifica di indirizzi invalidi allora la connessione permette di riportare indirizzi invalidi al mittente
- i messaggi di errore (ad esempio “port unreachable”) sono riportati tramite appositi pacchetti ICMP

Protocollo di connessione STREAM

A tre fasi



Associazione completa:

`<source-port, source-IP, destination-port, destination-IP, protocol>`

Un esempio nel dominio AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```
void main() {
    int ds_sock, length, ret; struct sockaddr_in addr;
    struct hostent *hp; /* utilizzato per la restituzione
                        della chiamata gethostbyname() */

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    addr.sin_family = AF_INET;
    addr.sin_port   = 25000;

    hp = gethostbyname("hermes.dis.uniroma1.it");
    memcpy(&addr.sin_addr, hp->h_addr, 4);

    ret = connect(ds_sock, &addr, sizeof(addr));
    if ( ret == -1 ) printf("Errore nella chiamata connect\n");

    close(ds_sock);
}
```

Il buffer struct hostent

```
#define h_addr h_addr_list[0]; /* indirizzo del buffer
                                di specifica del
                                numero IP */

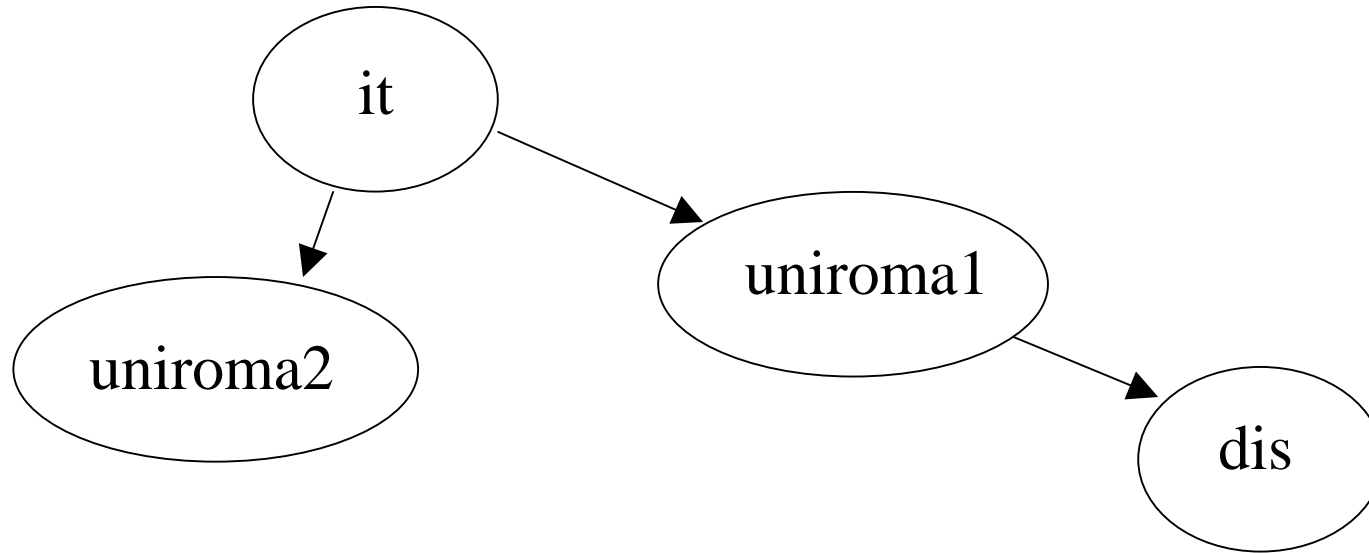
struct hostent {
    char    *h_name;           /* nome ufficiale dell'host */
    char    **h_aliases;      /* lista degli alias */
    int     h_addrtype;       /* tipo di indirizzo dell'host */
    int     h_length;         /* lunghezza (in byte)
                                dell'indirizzo */
    char    **h_addr_list;    /* lista di indirizzi dal
                                name server */
}
```

- gethostbyname() è non completamente adeguata per il multithread poichè non è rientrante
- in tal caso usare gethostbyname_r(), che invece è rientrante

Servizio DNS

- ad ogni host viene associato un nome, definito come stringhe separate da punti
- la prima stringa identifica il nome dell'host vero e proprio
- le stringhe rimanenti identificano la rete di appartenenza, detta anche dominio
- esistono host dislocati in rete che ospitano dei name server, ovvero dei server che implementano un meccanismo distribuito su scala geografica per risalire all'indirizzo IP di un host a partire dal nome (e viceversa)
- l'organizzazione e' gerarchica, basata su ripartizione per domini

Gerarchia di domini



Ogni livello gerarchico ha almeno un NS autoritativo

Alcuni domini di massimo livello

- com -> organizzazioni commerciali
- edu -> istituzioni USA per l'istruzione
- gov -> istituzioni governative USA
- mil -> istituzioni militari USA
- net -> maggiori organizzazioni di supporto ad Internet
- org -> organizzazioni senza scopo di lucro diverse dalle precedenti
- it,fr,... -> domini nazionali

Spedizione e ricezione dati

Previo uso di connect()

- si possono utilizzare le system call read() write()
- alternativamente si possono utilizzare le seguenti system call

```
int send(int sock_ds, const void *buff, int size, int flag)
```

```
int recv(int sock_ds, const void *buff, int size, int flag)
```

Descrizione invocano spedizione/ricezione di dati tramite socket

Argomenti

- 1) sock_ds: descrittore di socket locale
- 2) *buff: puntatore al buffer destinato ai dati
- 3) size: taglia dei dati
- 4) flag: specifica delle opzioni di spedizione

Restituzione -1 in caso di errore

Spedizione e ricezione dati

Modalità sconnessa

- si possono utilizzare le seguenti system call

```
int sendto(int sock_ds, const void *buff, int size, int flag, struct sockaddr *addr,  
           int addrlen)
```

```
int recvfrom(int sock_ds, const void *buff, int size, int flag, struct sockaddr *addr,  
            int *addrlen)
```

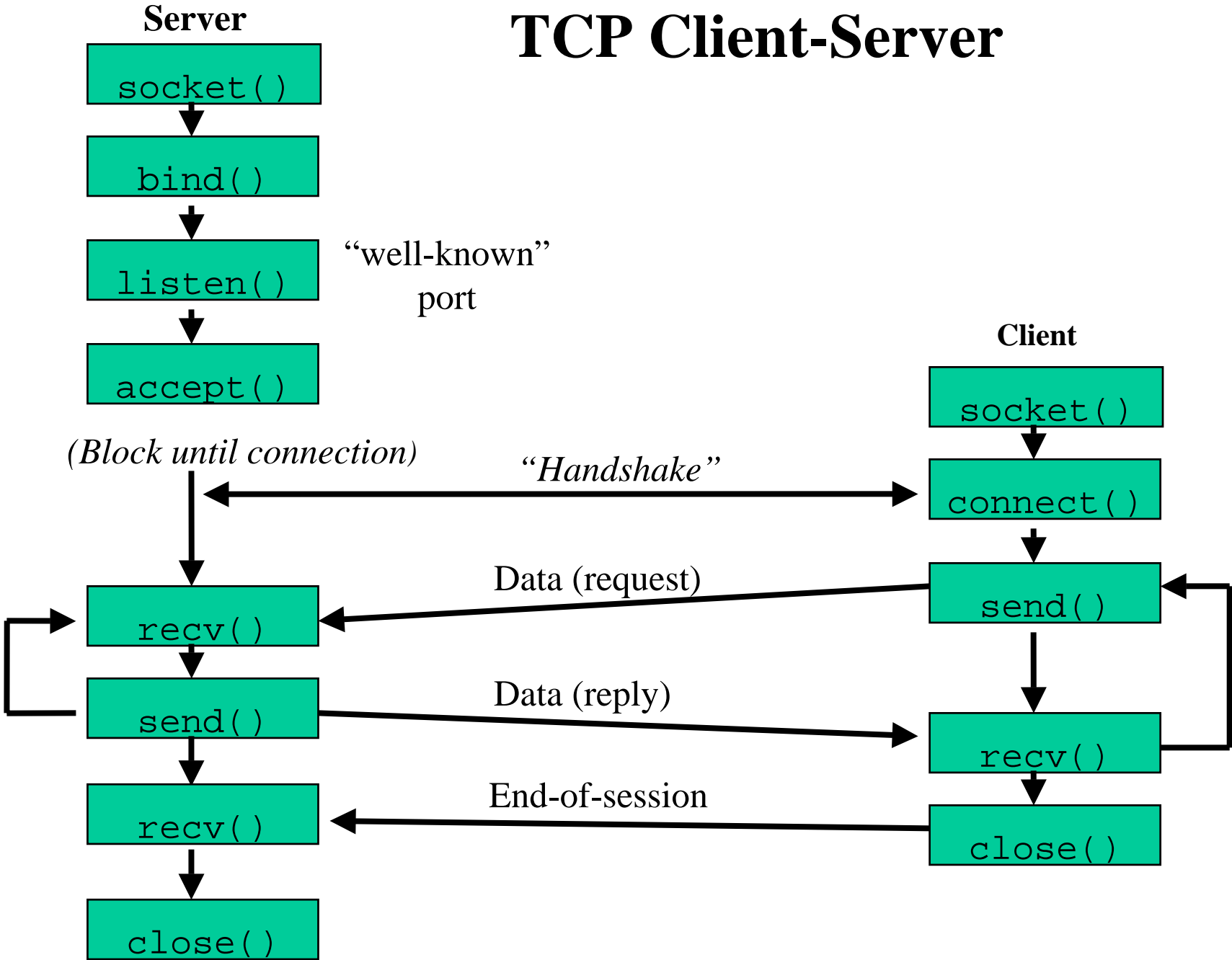
Descrizione invocano spedizione/ricezione di dati tramite socket

Argomenti

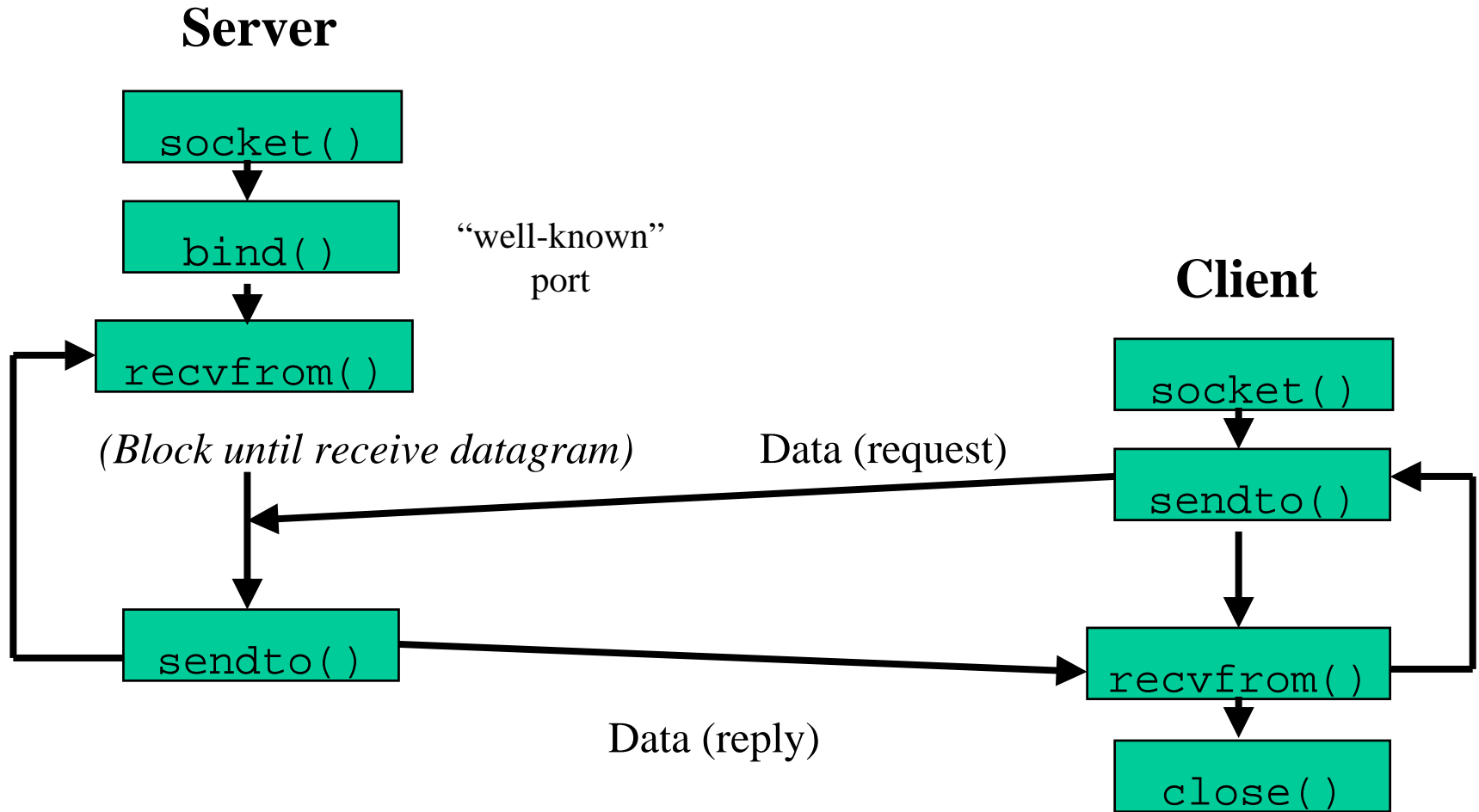
- 1) sock_ds: descrittore di socket locale
- 2) *buff: puntatore al buffer destinato ai dati
- 3) size: taglia dei dati
- 4) flag: specifica delle opzioni di spedizione
- 5) *addr: buffer per l'indirizzo di destinazione/sorgente
- 6) addrlen (*addrlen): lunghezza indirizzo destinazione/sorgente

Restituzione -1 in caso di errore

TCP Client-Server



UDP Client-Server



Byte Ordering

- diverse architetture hardware manipolano i dati di dimensione maggiore di un byte in maniera diversa.

ES: un intero di 4 byte contenente il valore 258 può essere rappresentato in due modi differenti:

Big Endian

0	3	
2	1	0	0

Little Endian

0	3	
0	0	1	2

- i dati che vanno sulla rete sono sempre in **network order** (big endian)
- tuttavia i dati usati sulla macchina sono in **host order** (little o big endian, dipendente dall'architettura hardware)

Funzioni di conversione

- alcune system call richiedono che certi dati vengano forniti in **network order** (ES: il contenuto di struct sockaddr_in in bind())
- un programma che usi i socket puo' funzionare su una architettura HW ma non su altre, **anche se si usa lo stesso sistema operativo!**

Soluzione: funzioni di conversione

(mascherano differenze architetturali)

```
uint16_t    htons (uint16_t  host16bitvalue);
```

```
uint32_t    htonl (uint32_t  host32bitvalue);
```

Prendono come parametro un intero in host order a 16 o 32 bit
rispettivamente e restituiscono lo stesso intero in network order

```
uint16_t    ntohs (uint16_t  network16bitvalue);
```

```
uint32_t    ntohl (uint32_t  network32bitvalue);
```

Prendono come parametro un intero in network order a 16 o 32 bit
rispettivamente e restituiscono lo stesso intero in host order

Un esempio di applicazione di TCP

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```
#define MAX_DIM 1024
#define CODA 3
```

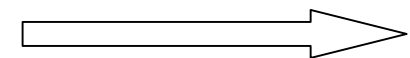
```
void main() {
    int ds_sock, ds_sock_a, rval;
    struct sockaddr_in server;
    struct sockaddr client;
    char buff[MAX_DIM];

    sock = socket(AF_INET, SOCK_STREAM, 0);

    bzero((char*)&server, sizeof(server));
    server.sin_family      = AF_INET;
    server.sin_port       = htons(25000);
    server.sin_addr.s_addr = INADDR_ANY;
```

Processo server

continua



```
bind(ds_sock, &server, sizeof(server));
listen(ds_sock, CODA);

length = sizeof(client);
signal(SIGCHLD, SIG_IGN);

while(1) {

    while( (ds_sock_a = accept(ds_sock, &client, &length))
           == -1);

    if (fork()==0) {
        close(ds_sock);
        do {
            read(ds_sock_a, buff, MAX_DIM);
            printf("messaggio del client = %s\n", buff);
        } while(strcmp(buff,"quit") != 0);
        write(ds_sock_a, "letto", 10);
        close(ds_sock_a);
        exit(0);
    }
    else close(ds_sock_a);
}
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

Processo client

```
#define      MAX_DIM      1024
```

```
void main()  {
    int      ds_sock, length, res;
    struct sockaddr_in client;
    struct hostent *hp;
    char buff[MAX_DIM];

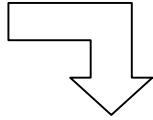
    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    client.sin_family = AF_INET;
    client.sin_port   = htons(25000);

    hp = gethostbyname("hermes.dis.uniroma1.it");
    bcopy(hp->h_addr, &client.sin_addr, hp->h_length);

    res      = connect(ds_sock, &client, sizeof(client));
```

continua



```
    if ( res ==  -1 ) {
        printf("Errore nella connect \n");
        exit(1);
    }

    printf("Digitare le stringhe da  trasferire (quit
           per terminare): ");

    do {
        scanf("%s", buff);
        write(ds_sock, buff, MAX_DIM);
    } while(strcmp(buff,"quit")      != 0);

    read(ds_sock, buff, MAX_DIM);
    printf("Risposta del server: %s\n", buff);

    close(ds_sock);
}
```

Controllo avanzato dei protocolli di rete

- se il secondo argomento di una system call `socket()` è `SOCK_RAW` viene creata una “raw socket”
- le raw socket sono utilizzate per poter scrivere informazioni direttamente sopra il livello di rete (e non sopra il livello di trasporto come avviene con `SOCK_STREAM` e `SOCK_DGRAM`)
- un utilizzo accorto permette anche di modificare il livello di rete
- per poter creare una raw socket bisogna avere privilegi di root

Creazione di un Raw Socket

`socket (AF_INET, SOCK_RAW, protocol)`

IPPROTO_ICMP
IPPROTO_IGMP
IPPROTO_EGP

...

identifica un protocollo
specifico (sopra il livello IP)
con cui si vuole lavorare

0

non identifica un
protocollo specifico
sopra il livello IP.
(si lavora con tutti
i protocolli su IP)

Lettura da un raw socket

- la ricezione di pacchetti IP da un raw socket avviene tramite la system call **recvfrom()**
- un raw socket il cui campo “protocol” è 0 riceve quasi tutti i pacchetti ICMP, **nessun** pacchetto TCP o UDP, e tutti i pacchetti degli altri protocolli
- se il campo “protocol” è diverso da 0, il raw socket riceve solo i pacchetti IP che specificano quel valore nel campo “protocollo”
- se viene effettuata una system call bind() su un raw socket, essa riceve solo i pacchetti destinati all’indirizzo IP specificato nella bind (port viene ignorato) – utile per IP multipli (schede di rete multiple)
- se viene effettuata una system call connect() su un raw socket, essa riceve solo i pacchetti provenienti dall’indirizzo IP specificato nella connect (il campo port viene ignorato).

Un pacchetto ricevuto da un raw socket include anche l’header del protocollo IP

Opzioni su Socket

- un socket può avere una serie di opzioni
- ogni opzione permette di controllare il comportamento di alcuni livelli del protocollo di comunicazione

```
int setsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen)
```

```
int getsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen)
```

Descrizione system call che permettono di cambiare una delle opzioni sul socket o di leggerne lo stato, rispettivamente

Argomenti

- 1) sockfd: descrittore di un socket
- 2) level: identificatore del tipo di opzione (opzione relativa al socket o relativa a qualche protocollo specifico)
- 3) optname: identificatore della specifica opzione da cambiare/leggere
- 4) optval: puntatore a una variabile contenente il valore da cambiare o in cui verrà registrato il valore da leggere.
- 5) optlen: puntatore alla lunghezza della variabile puntata da optval o lunghezza della variabile puntata da optval

Ritorno 0 se tutto OK, -1 in caso di errore

Scrittura su un raw socket

- la spedizione di pacchetti IP da una raw socket avviene tramite la system call **sendto()**
- il modo in cui la socket interpreta il pacchetto da spedire dipende dal valore dell'opzione **IP_HDRINCL**

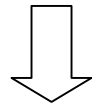
Attivazione e disattivazione di **IP_HDRINCL**

```
const int on = 1; /* Per attivare */  
const int on = 0; /* Per disattivare */  
setsockopt (sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
```

- se **IP_HDRINCL** non è attivo il pacchetto passato al raw socket viene interpretato come il contenuto di un pacchetto IP. Il sistema operativo costruisce un header IP. Il contenuto del campo “protocol” all’interno del socket() specifica il valore che viene scritto nel campo “protocollo” dell’header IP
- se **IP_HDRINCL** è attivo il pacchetto passato al raw socket viene interpretato come un pacchetto IP completo di header

Manipolare pacchetti da/per raw socket

- **in teoria:** conoscendo l'esatta struttura dell'header di un pacchetto da inviare su un raw socket, si può scrivere il pacchetto byte per byte, header di protocollo incluso (tenendo presente che ogni campo di protocolli standard va scritto in network order)
- **in pratica:** la scrittura di un pacchetto byte per byte è onerosa e poco maneggevole



Esistono strutture dati che riflettono accuratamente la struttura degli header di un protocollo

ESEMPI:

- in `<netinet/ip.h>` è definita una **struct ip**
- in `<netinet/ip_icmp.h>` è definita una **struct icmp**

Riferimenti

Rago, S.: UNIX System V Network Programming, Addison-Wesley, 1993.

Stevens, W.R.: UNIX Network Programming, Prentice Hall, 1998.

Peterson - Davie: "Computer Networks: A system approach" Morgan Kaufmann 2000.

Windows sockets (much like UNIX sockets)

```
SOCKET socket(int address_family, int type,  
              int protocol)
```

Descrizione

- invoca la creazione di un socket

Argomenti

- `address_family`: specifica la famiglia di indirizzi con cui il socket può operare (un elenco completo può essere trovato nel file `winsock2.h` di Visual C++), il dominio di indirizzi di interesse per la nostra trattazione è `AF_INET`
- `type`: specifica la semantica della comunicazione associata al socket `SOCK_STREAM` e `SOCK_DGRAM`
- `protocol`: specifica il particolare protocollo di comunicazione per il socket (usare 0 per il default)

Restituzione

- un descrittore di socket in caso di successo; `INVALID_SOCKET` in caso di fallimento

Associazione di indirizzi

```
int bind(SOCKET ds_sock, struct sockaddr *my_addr,  
         int addrlen)
```

Descrizione

invoca l'assegnazione di un indirizzo al socket

Argomenti

- `ds_sock`: descrittore di socket
- `*my_addr`: puntatore al buffer che specifica l'indirizzo
- `addrlen`: lunghezza (in byte) dell'indirizzo

Restituzione

- 0 in caso di successo; `SOCKET_ERROR` in caso di fallimento

Attesa di connessioni

```
SOCKET accept(SOCKET ds_sock, struct sockaddr  
              *addr, int *addrlen)
```

Descrizione

- invoca l'accettazione di una connessione su un socket

Argomenti

- `ds_sock`: descrittore di socket
- `*addr`: puntatore al buffer su cui si copierà l'indirizzo del chiamante
- `*addrlen`: puntatore al buffer su cui si scriverà la taglia dell'indirizzo del chiamante

Restituzione

- il descrittore di un nuovo socket in caso di successo;
`INVALID_SOCKET` in caso di errore

Connessioni

```
int connect(SOCKET ds_socks, struct sockaddr  
            *addr, int addrlen)
```

Descrizione

- invoca la connessione su un socket

•Argomenti

- ds_sock: descrittore del socket locale
- *addr: puntatore al buffer contenente l'indirizzo del socket al quale ci si vuole connettere
- addrlen: la taglia dell'indirizzo del socket al quale ci si vuole connettere

Restituzione

- 0 per una connessione corretta, SOCKET_ERROR in caso di errore

Backlog e chiusura di socket

```
listen(SOCKET ds_sock, int backlog)
```

```
closesocket(SOCKET ds_socket)
```

Comunicazione

```
int recv(SOCKET sock_ds, const char *buff,  
         int size, int flag)
```

```
int send(SOCKET sock_ds, const char *buff, int  
         size, int flag)
```

```
int recvfrom(SOCKET sock_ds, char *buff, int  
            size, int flag, struct sockaddr *addr, int  
            *addrlen)
```

```
int sendto(SOCKET sock_ds, const void *buff, int  
          size, int flag, struct sockaddr *addr, int  
          addrlen)
```

Inizializzazione interfaccia Winsocket

```
int WSStartup( WORD wVersionRequested,  
              LPWSADATA lpWSADATA )
```

Parametri

- `wVersionRequested`: la piu' alta versione delle Window Sockets che il processo chiamante puo' supportare. Il byte piu' significativo specifica il numero di minor version; il byte piu' significativo specifica la major version. Vedi funzione `MAKEWORD(x,y)`
- `lpWSADATA`: puntatore ad una struttura `WSADATA` che riceve i dettagli sull'implementazione

Restituzione

0 se ha successo, altrimenti un codice di errore