Sistemi a più processori

Ivan Scagnetto

Università di Udine — Facoltà di Scienze MM.FF.NN.

A.A. 2007-2008

Fatto: Aumento della necessità di potenza di calcolo.
Velocità di propagazione del segnale (20 cm/ns) impone limiti strutturali all'incremento della velocità dei processori (es: 10GHz $\Rightarrow$ max 2 cm)
Tendenza attuale: distribuire il calcolo tra più processori.

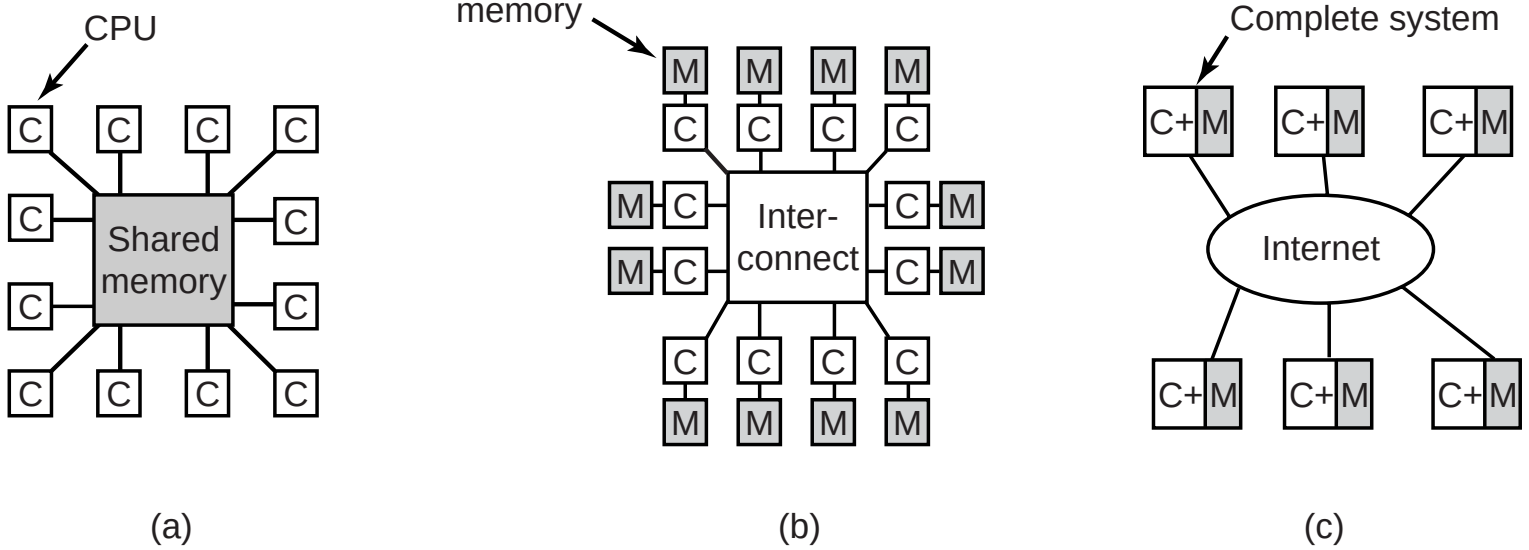
processori strettamente accoppiati:

- sistemi multiprocessore: condividono clock e/o memoria; comunicazione attraverso memoria condivisa: UMA (SMP, crossbar, ...), NUMA (CC-NUMA, NC-NUMA), COMA;
- multicomputer: comunicazione con message passing: cluster, COWS.

processori debolmente accoppiati: non condividono clock e/o memoria; comunicano attraverso canali asincroni molto più lenti

- Sistemi distribuiti: reti.

Multiprocessore, multicomputer, sistema distribuito



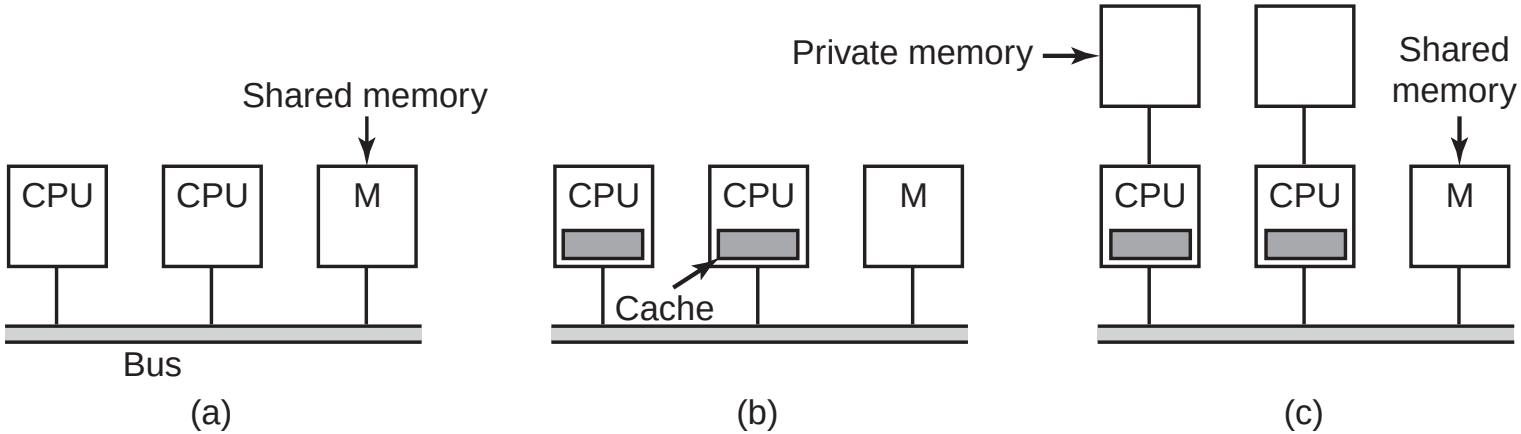
(a) (b) (c)

Differenze:

- costo
- scalabilità
- complessità di programmazione/utilizzo

Hardware multiprocessore: SMP UMA

I processori condividono il bus di accesso alla memoria.

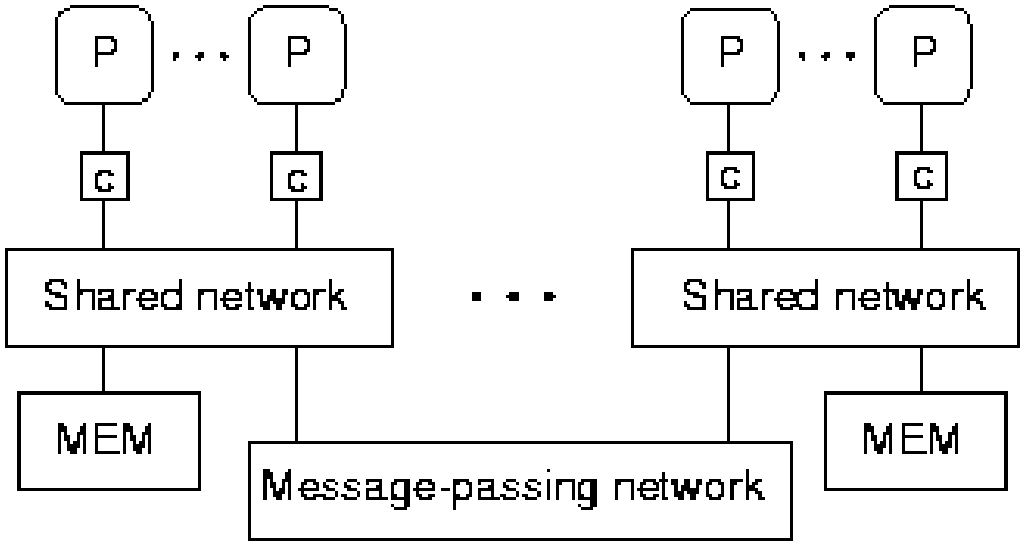


Uso di cache per diminuire la contesa per la memoria.
Problemi di coerenza $\Rightarrow$ uso di cache (bus snooping) e memorie private (necessitano di adeguata gestione da parte del compilatore)
Limitata scalabilità (max 16 CPU, solitamente meno di 8). Per andare oltre, si deve usare reti crossbar o omega, o si passa a NUMA.

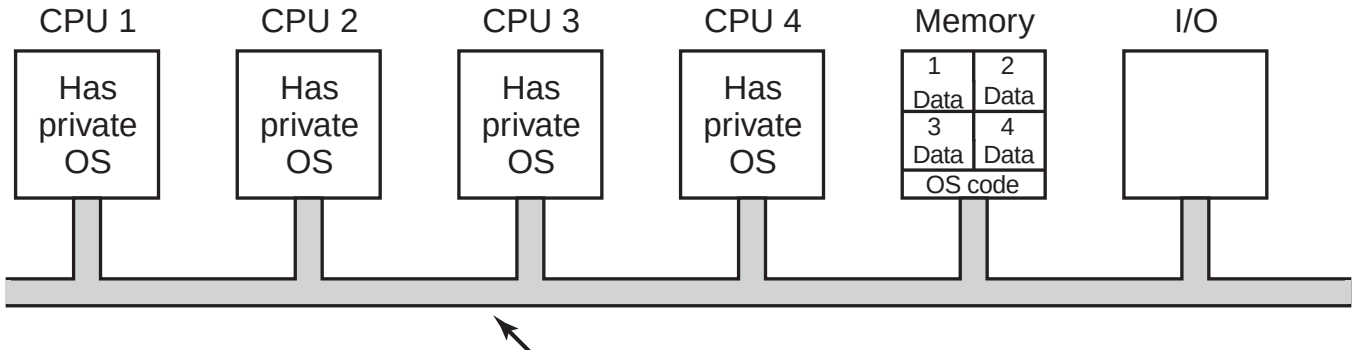
Hardware multiprocessore: NUMA

Applicabili a 100+ processori. Due o più tipi di memorie:

- locale: privata ad ogni CPU/gruppo SMP di CPU
- remota: condivisa tra le CPU; tempo di accesso $2 - 15$ volte quello locale.
- Reindirizzamento via rete/bus, risolto in hardware (MMU)
- Eventualmente, una cache locale per la memoria remota (CC-NUMA)

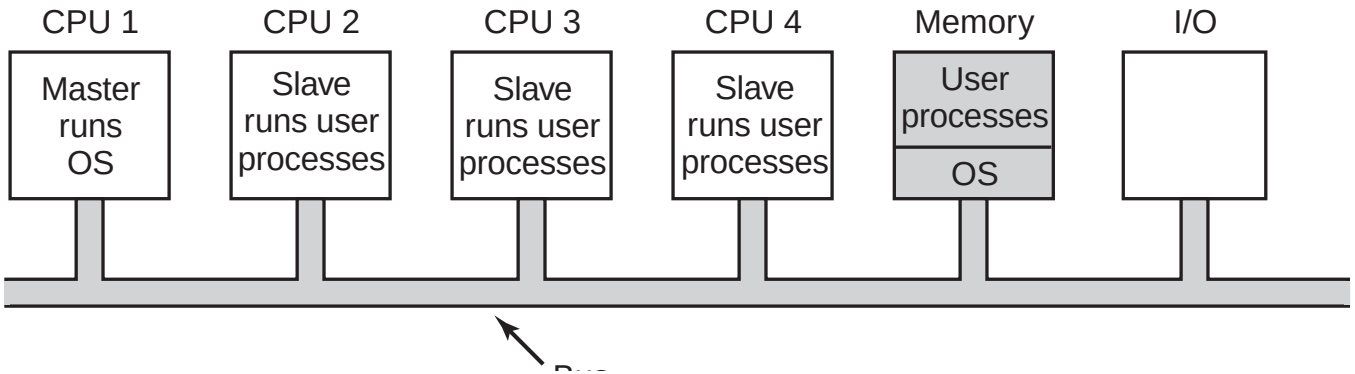


Ciascuna CPU ha il proprio sistema operativo



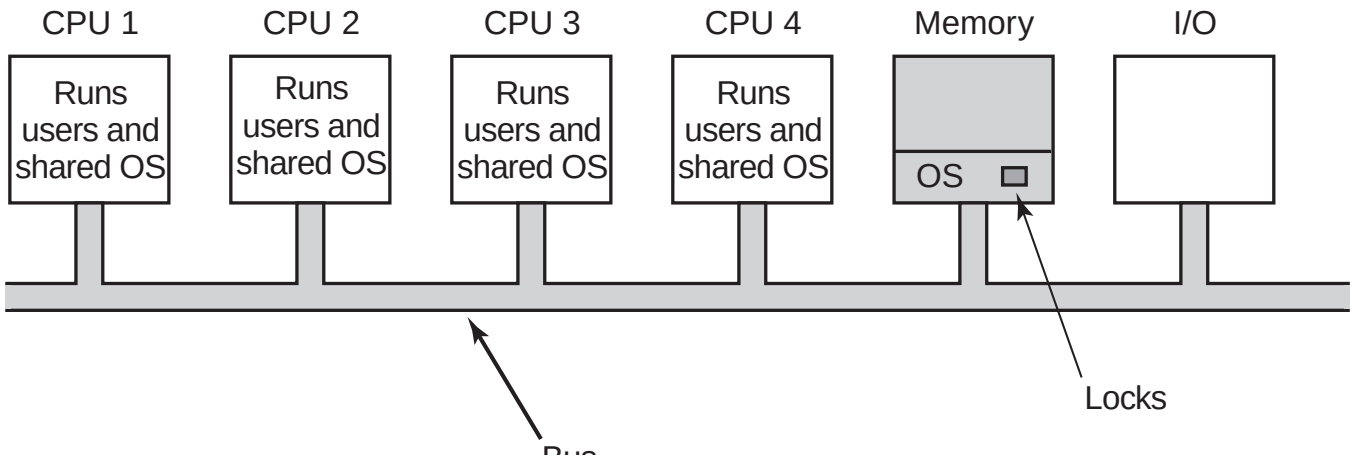
- Ogni CPU esegue privatamente il proprio SO, con i propri dati
- Non esiste distribuzione del carico (solo schedulazione locale), né condivisione della memoria (solo allocazione locale)
- problemi di coerenza tra i vari SO (es: cache di disco). Obsoleto

SO Multiprocessori Master-Slave (o AMP)



- Processi e thread di sistema e in spazio kernel vengono eseguiti solo da un processore; gli altri eseguono solo processi utente
- Il master può allocare i processi tra i vari slave, bilanciando il carico
- La memoria viene allocata centralmente
- Poco scalabile: il master è collo di bottiglia

SO Multiprocessori Simmetrici (SMP)



- Completa simmetria fra tutti i processori
- Il S.O. è sempre un collo di bottiglia
  - sia p_{sys} il tempo consumato in modo kernel da un processore
  - la probabilità che almeno 1 su n processori stia eseguendo codice kernel è $p = 1 - (1 - p_{sys})^n$.
  - Per $p_{sys} = 10\%$, $n = 10$: $p = 65\%$. Per $n = 16$: $p = 81.5\%$.

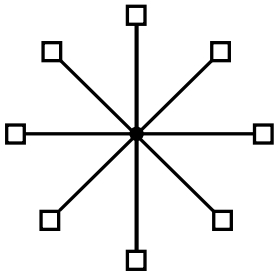
SO Multiprocessori Simmetrici (SMP) (cont.)

- Un mutex generale su tutto il kernel riduce di molto il parallelismo
- bisogna parallelizzare l'accesso al kernel
- Si può suddividere il kernel in sezioni indipendenti, ognuna ad accesso esclusivo
- Ogni struttura dati del kernel deve essere protetta da mutex
- Richiede una accurata rilettura e "decorazione" del codice del sistema operativo con mutex
- Molto complesso e delicato: errate sequenze di mutex possono portare a deadlock di interi processori in modalità kernel, ossia di congelamenti del sistema

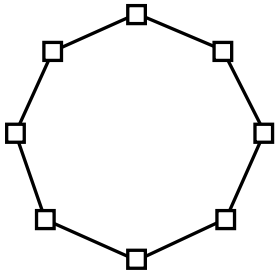
Multicomputer

- I multiprocessori sono comodi perché offrono un modello di comunicazione simile a quello tradizionale
- Problemi di costo e scalabilità
- Alternativa: multicomputer: calcolatori strettamente accoppiati ma senza memoria condivisa, solo passaggio di messaggi
  - esempio: rete di PC con buone schede di rete (Clusters of Workstations)

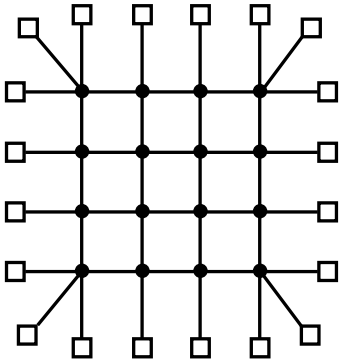
Topologie di connessione per multicomputer



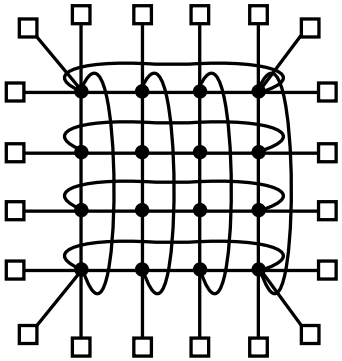
(a)



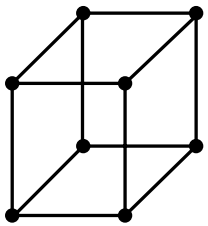
(b)



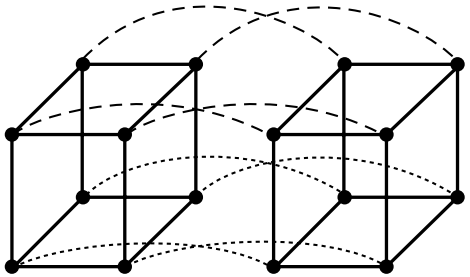
(c)



(d)



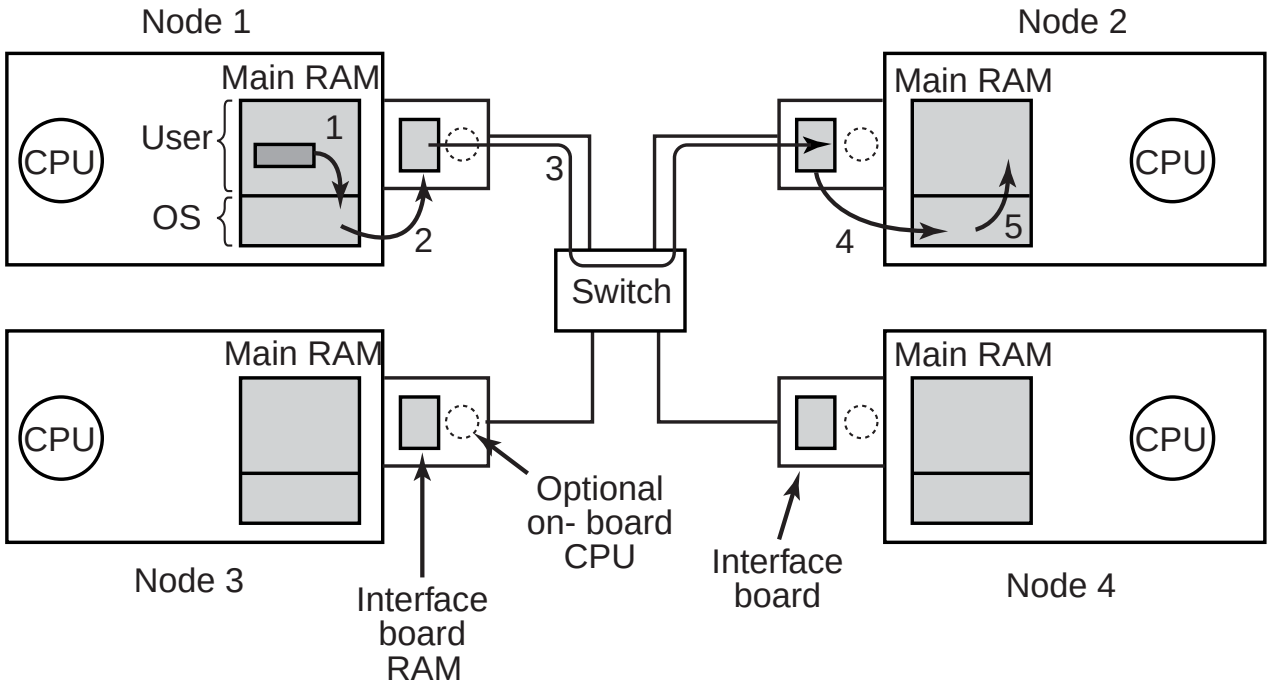
(e)



(f)

Gli ipercubi sono molto usati perché diametro = $\log_2$ nodi

Interfacce di rete



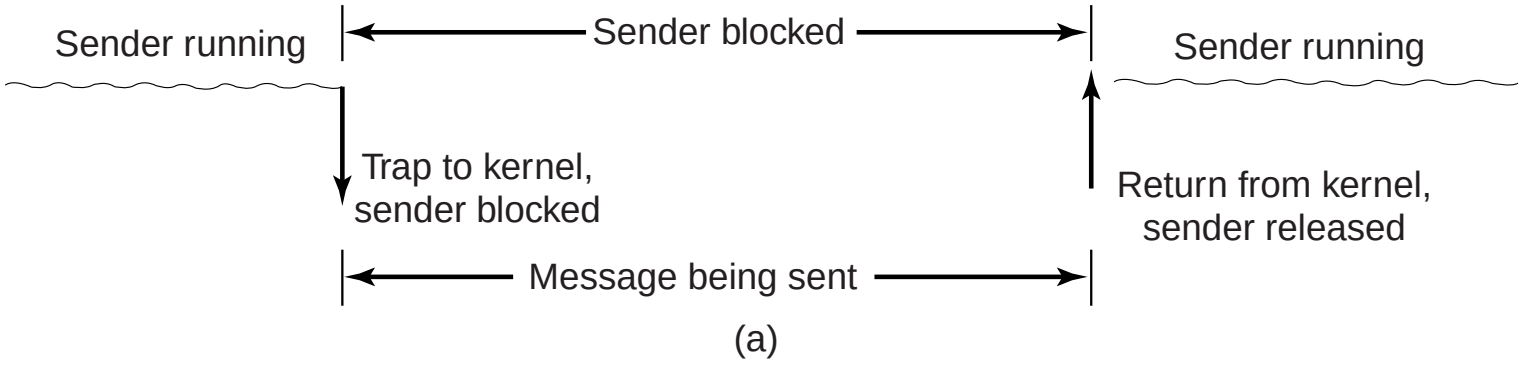
- Le interfacce di rete hanno proprie memorie, e spesso CPU dedicate
- Il trasferimento dei dati da nodo a nodo richiede spesso molte copie di dati e quindi rallentamento
- Può essere ridotto usando DMA, memory mapped I/O e lockando in memoria le pagine dei processi utente contenenti i buffer di I/O

Programmazione in multicomputer: passaggio di messaggi

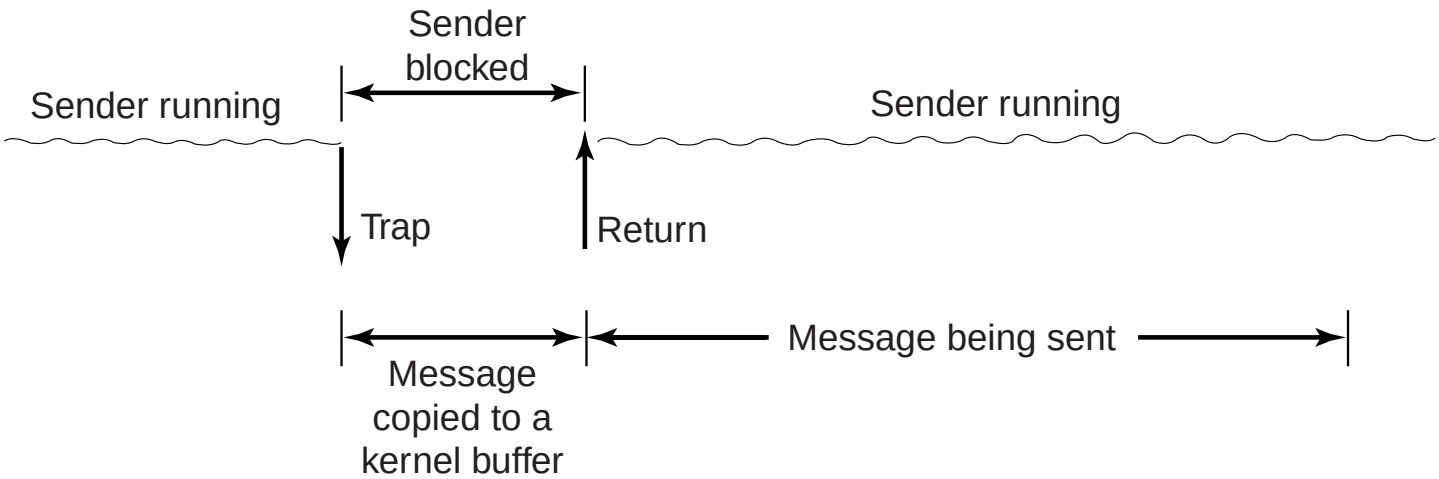
- Il modello base usa comunicazione con passaggio di messaggi
- Il S.O. offre primitive per la comunicazione tra i nodi

  `send(dest, &mptr); receive(addr, &mptr);`
- Il programmatore deve approntare appositi processi client/server tra i nodi, con protocolli di comunicazione
- Assomiglia più ad un sistema di rete che ad un unico sistema di calcolo: il programmatore vede la delocalizzazione del calcolo;
- Adottato in sistemi per calcolo scientifico (librerie PVM e MPI per Beowulf), dove il programmatore o il compilatore parallelizza e distribuisce il codice.

Send bloccanti/non bloccanti



(a)



(b)

Send bloccanti/non bloccanti

Quattro possibilità:

1. `send` bloccante: CPU inattiva durante la trasmissione del messaggio
2. `send` non bloccante, con copia su un buffer di sistema: spreco di tempo di CPU per la copia
3. `send` non bloccante, con interruzione di conferma: molto difficile da programmare e debuggare
4. Copia su scrittura: il buffer viene copiato quando viene modificato

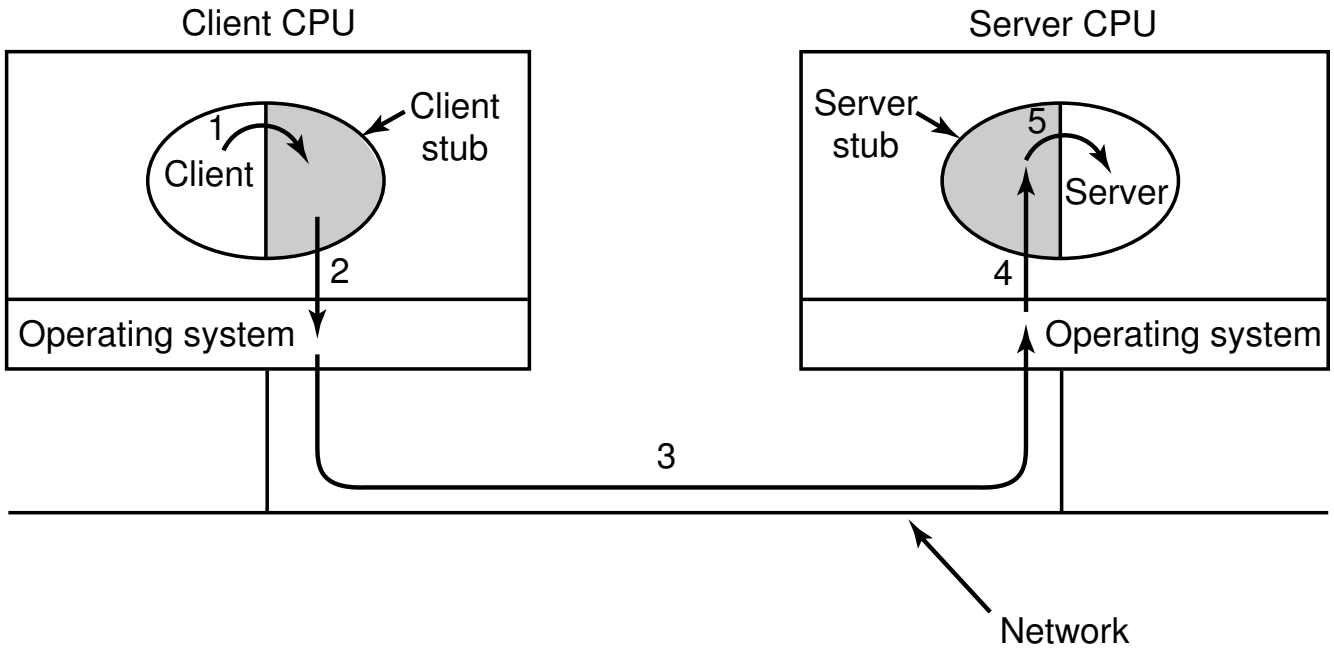
Tutto sommato, la (1) è la migliore (soprattutto se abbiamo a disposizione i thread).

Programmazione in multicomputer: RPC

Chiamate di procedure remote: un modello di computazione distribuita più astratto

- Idea di base: un processo su una macchina può eseguire codice su una CPU remota.
- L'esecuzione di procedure remote deve apparire simile a quelle locali.
- Nasconde (in parte) all'utente la delocalizzazione del calcolo: l'utente non esegue mai send/receive, deve solo scrivere ed invocare procedure come al solito.
- Versione a oggetti: RMI (Remote Method Invocation)

Esempio di RPC



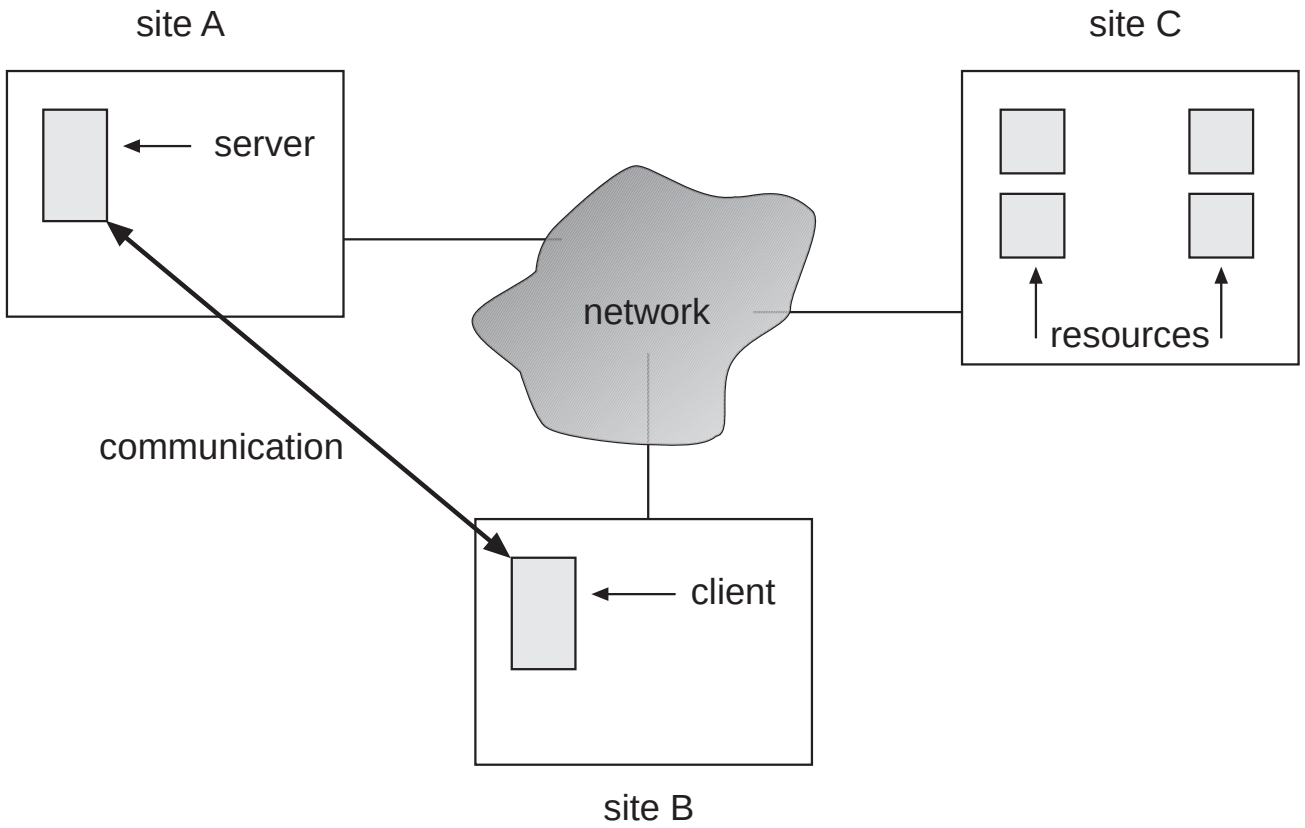
1. Chiamata del client alla procedura locale (allo stub)
2. Impacchettamento (marshaling) dei dati e parametri
3. Invio dei dati al server RPC
4. Spacchettamento dei dati e parametri
5. Esecuzione della procedura remota sul server.

Problemi con RPC

- Come passare i puntatori? A volte si può fare il marshaling dei dati puntati e ricostruirli dall'altra parte (call-by-reference viene sostituito da call-by-copy-and-restore), ma non sempre...
- Procedure polimorfe, con tipo e numero degli argomenti deciso al runtime: difficile da fare uno stub polimorfo
- Accesso alle variabili globali: globali a cosa?

Ciò nonostante, le RPC sono state implementate ed usate molto diffusamente, con alcune restrizioni.

Sono sistemi lascamente accoppiati, normalmente più orientati verso la comunicazione (=accesso a risorse remote), che al calcolo intensivo.



Client, server, protocollo

server: processo fornitore un servizio; rende disponibile una risorsa ad altri processi (locali o remoti), di cui ha accesso esclusivo (di solito). Attende passivamente le richieste dai client.

client: processo fruitore di un servizio: richiede un servizio al server (accesso alla risorsa). Inizia la comunicazione (attivo).

protocollo: insieme di regole che descrive le interazioni tra client e server.

Un processo può essere server e client, contemporaneamente o in tempi successivi.

Motivazioni per i sistemi distribuiti

- Condivisione delle risorse
  - condividere e stampare file su siti remoti
  - elaborazione di informazioni in un database distribuito
  - utilizzo di dispositivi hardware specializzati remoti
- Accelerazione dei calcoli: bilanciamento del carico
- Affidabilità: individuare e recuperare i fallimenti di singoli nodi, sostituire nodi difettosi
- Comunicazione: passaggio di messaggi tra processi su macchine diverse, similarmente a quanto succede localmente.

Confronto tra i vari modelli paralleli

Item	Multiprocessor	Multicomputer	Distributed System
Node configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly all different
File systems	One, shared	One, shared	Each node has own
Administration	One organization	One organization	Many organizations

Hardware dei sistemi distribuiti: LAN e WAN

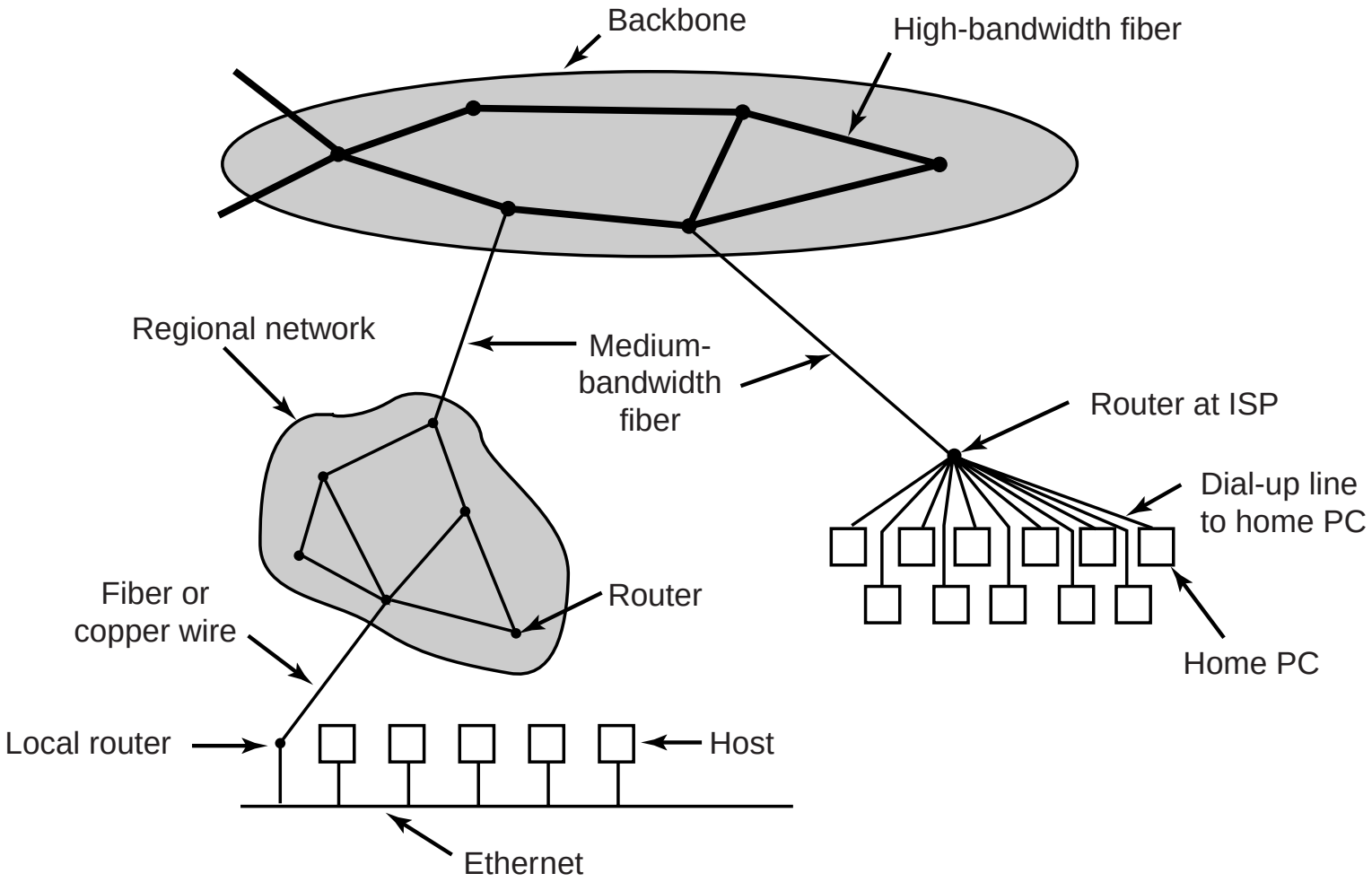
LAN: copre piccole aree geografiche

- struttura regolare: ad anello, a stella o a bus
- Velocità: $\approx$ 100 Mb/sec, o maggiore.
- Broadcast è fattibile ed economico
- Nodi: workstation e/o personal computer; qualche server e/o mainframe

WAN: collega siti geograficamente distanti

- connessioni punto-punto su linee a lunga distanza (p.e. cavi telefonici)
- Velocità $\approx$ 100kbit/sec – 34Mb/sec
- Il broadcast richiede solitamente più messaggi

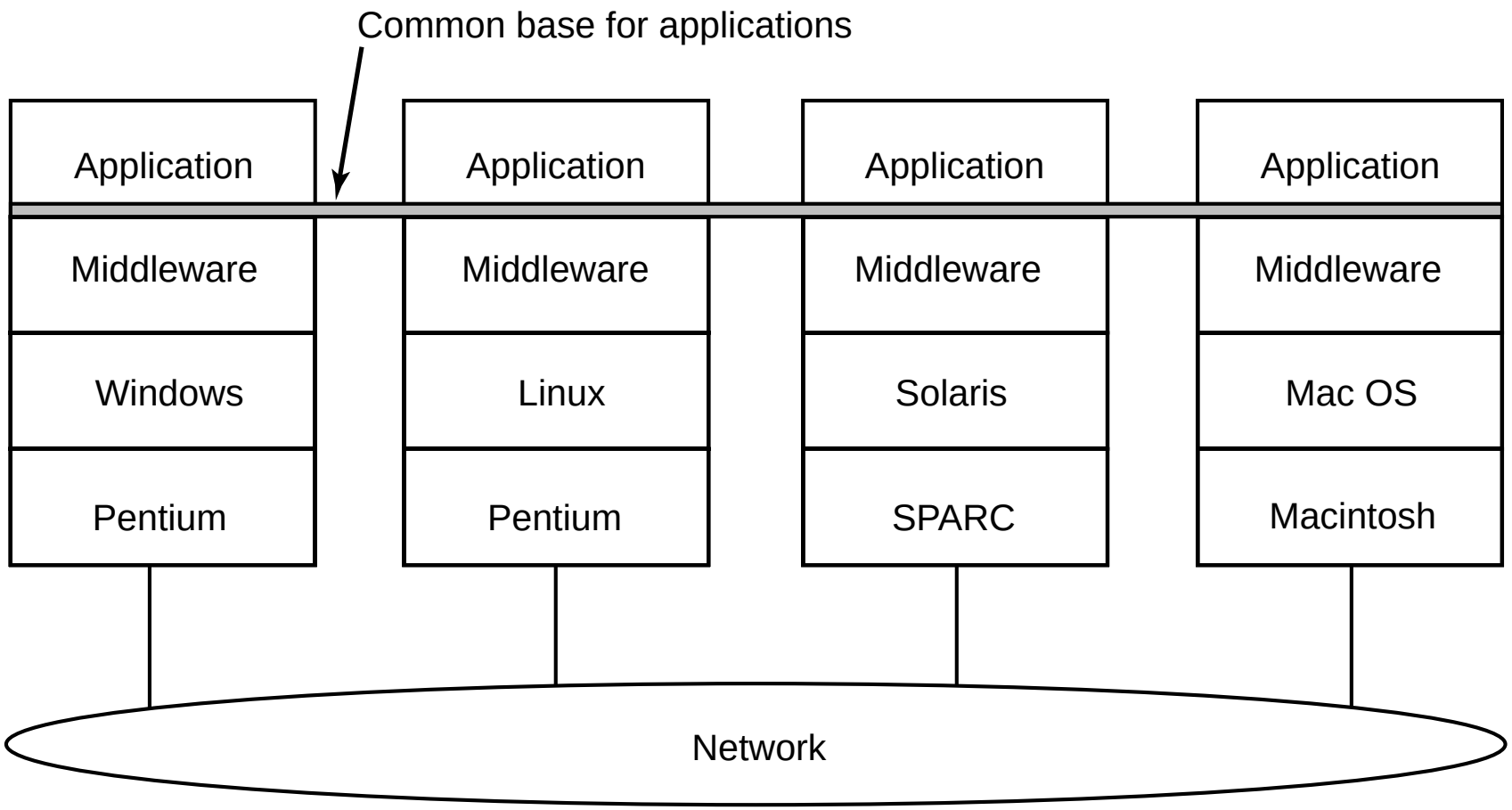
Hardware dei sistemi distribuiti: LAN e WAN



Programmazione nei sistemi distribuiti

- L'apertura e l'accoppiamento lasco dei sistemi distribuiti aggiunge flessibilità
- Mancando un modello uniforme, la programmazione è complessa
- Spesso le applicazioni di rete devono reimplementare le stesse funzionalità (es: confrontate FTP, IMAP, HTTP, ...)
- Un sistema (operativo) distribuito aggiunge un paradigma (modello comune) alla rete sottostante, per uniformare la visione dell'intero sistema
- Spesso può essere realizzato da uno strato al di sopra del sistema operativo, ma al di sotto delle singole applicazioni: il middleware

Programmazione nei sistemi distribuiti



Servizi di rete e servizi distribuiti

Un sistema operativo con supporto per la rete può implementare due livelli di servizi:

- servizi di rete: offre ai processi le funzionalità necessarie per stabilire e gestire le comunicazioni tra i nodi del sistema distribuito (es.: socket)
- servizi distribuiti: sono modelli comuni (paradigmi di comunicazione) trasparenti che offrono ai processi una visione uniforme, unitaria del sistema distribuito. (es: file system remoto).

Tutti i S.O. moderni offrono servizi di rete; pochi offrono servizi distribuiti, e per modelli limitati.

Servizi e Sistemi operativi di rete

gli utenti sono coscienti delle diverse macchine in rete, e devono passare esplicitamente da una macchina all'altra

- Login su macchine remote (telnet)
- Trasferimento dati tra macchine (FTP)
- Posta elettronica, HTTP,...

Esempio di servizio di rete

```
ten$ ftp maxi
ftp> cd pippo/pluto
ftp> get paperino
ftp> quit
ten$
```

- Locazione non trasparente all'utente
- altro ambiente da imparare
- Manca vera condivisione della risorsa (duplicazione, possibili incoerenze)

Servizi di rete

I servizi sono funzionalità offerte a host e processi.

servizio orientato alla connessione "come un tubo", o una telefonata: si stabilisce una connessione, che viene usata per un certo periodo di tempo trasferendo una sequenza di bit, quindi si chiude la connessione

servizio senza connessione "come cartoline": si trasferiscono singoli messaggi, senza stabilire e mantenere una vera connessione

Ogni servizio ha anche un'affidabilità:

- affidabile: i dati arrivano sempre, non vengono persi né duplicati.
- non affidabile: i dati possono non arrivare, o arrivare duplicati, o in ordine diverso da quello originale

Servizi di rete

I servizi affidabili = servizi non affidabili + opportune regole di controllo (protocolli), come pacchetti di riscontro.
Introducono overhead (di traffico, di calcolo, di tempo) spesso inutile o dannoso. Es: voce/musica/video digitale

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages of a book
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connectionless	Unreliable datagram	Network test packets
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Protocolli di rete

- L'implementazione dei servizi di rete avviene stabilendo delle regole di comunicazione tra i vari nodi: i protocolli
- Un protocollo stabilisce le regole per attivare, mantenere, utilizzare e terminare un servizio di rete.
- L'implementazione dei protocolli per servizi evoluti è complesso; viene semplificato se i protocolli vengono stratificati
  - ogni strato dello stack (o suite) implementa nuove funzionalità in base a quelli sottostanti
  - in questo modo, si ottiene maggiore modularità e semplicità di progettazione e realizzazione.
- Soprattutto nei sistemi distribuiti, è fondamentale seguire protocolli standardizzati.

Modello ISO/OSI

Fisico: gestisce i dettagli fisici della trasmissione dello stream di bit.

Strato data-link: gestisce i frame, parti di messaggi di dimensione fissa, nonché gli errori e recuperi dello strato fisico

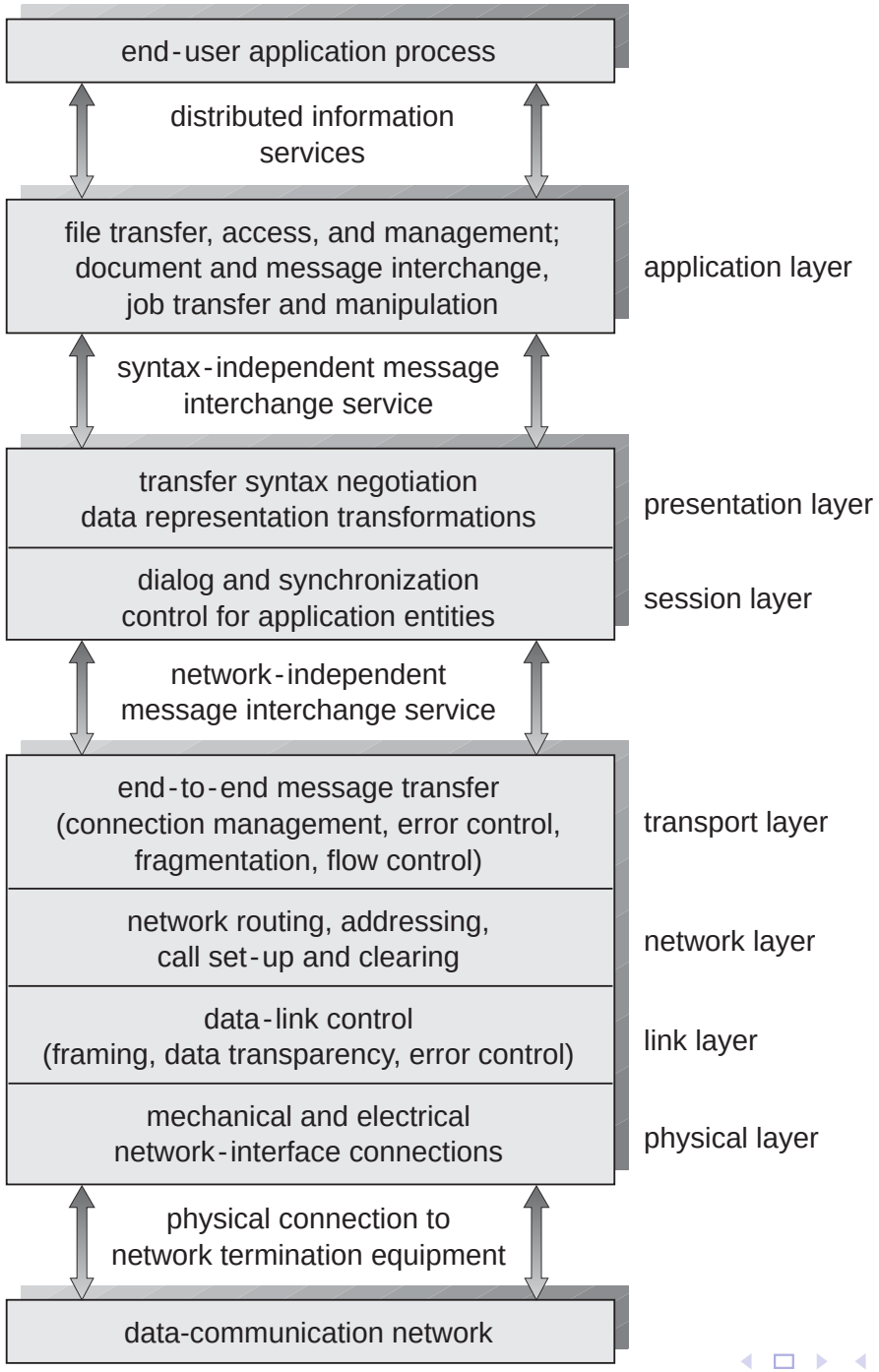
Strato di rete: fornisce le connessioni e instrada i pacchetti nella rete. Include la gestione degli indirizzi degli host e dei percorsi di instradamento.

Trasporto: responsabile dell'accesso di basso livello alla rete e per il trasferimento dei messaggi tra gli host. Include il partizionamento di messaggi in pacchetti, riordino di pacchetti, generazione di indirizzi fisici.

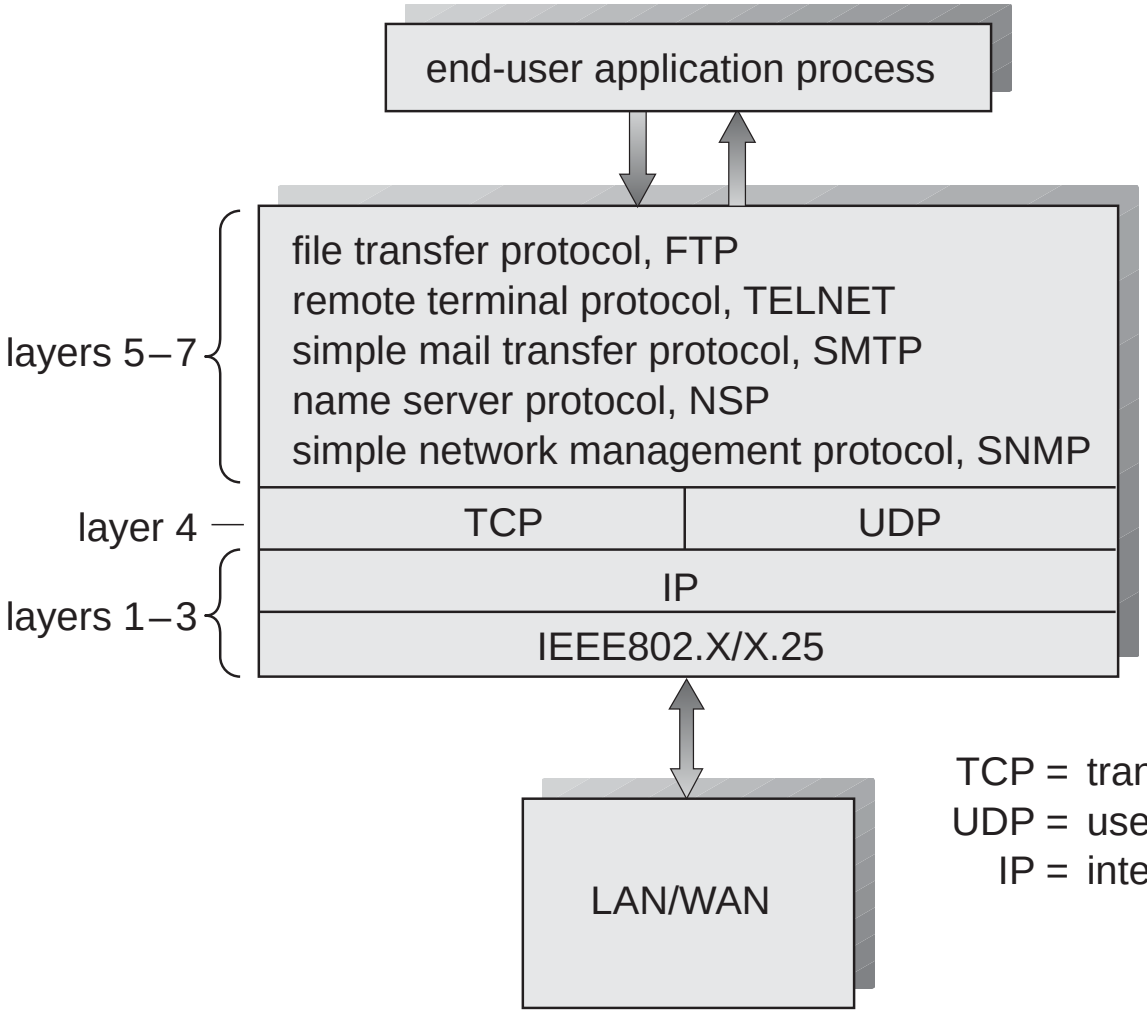
Sessione: implementa sessioni, ossia comunicazioni tra processi

Presentazione: risolve le differenze tra i varî formati dei diversi siti (p.e., conversione di caratteri)

Applicazione: interagisce con l'utente: trasferimento di file, protocolli di login remoto, trasferimento di pagine web, e-mail, database distribuiti, ...


end-user application process
distributed information services
file transfer, access, and management; document and message interchange, job transfer and manipulation
application layer
syntax-independent message interchange service
transfer syntax negotiation data representation transformations
presentation layer
dialog and synchronization control for application entities
session layer
network-independent message interchange service
end-to-end message transfer (connection management, error control, fragmentation, flow control)
transport layer
network routing, addressing, call set-up and clearing
network layer
data-link control (framing, data transparency, error control)
link layer
mechanical and electrical network-interface connections
physical layer
physical connection to network termination equipment
data-communication network

Stack TCP/IP



TCP = transmission control protocol
UDP = user datagram protocol
IP = internet protocol

Modelli di riferimento di rete e loro stratificazione

<table>
<tr><th>ISO
reference
model</th><th>ARPANET
reference
model</th><th>4.2BSD
layers</th><th>example
layering</th></tr>
<tr><td>application</td><td rowspan="3">process
applications</td><td rowspan="2">user programs
and libraries</td><td rowspan="2">telnet</td></tr>
<tr><td>presentation</td></tr>
<tr><td>session</td><td>sockets</td><td>sock_stream</td></tr>
<tr><td>transport</td><td rowspan="2">host–host</td><td rowspan="2">protocol</td><td>TCP</td></tr>
<tr><td rowspan="2">network
data link</td><td>IP</td></tr>
<tr><td>hardware</td><td>network
interface</td><td>network
interfaces</td><td>Ethernet
driver</td></tr>
<tr><td></td><td>network
hardware</td><td>network
hardware</td><td>interlan
controller</td></tr>
</table>

Socket

- Una socket (“presa, spinotto”) è un’estremità di comunicazione tra processi
- Una socket in uso è solitamente legata (bound) ad un indirizzo. La natura dell’indirizzo dipende dal dominio di comunicazione del socket.
- Processi comunicanti nello stesso dominio usano lo stesso formato di indirizzi
- Una socket comunica in un solo dominio. I dominî implementati sono descritti in `<sys/socket.h>`.
  I principali sono
  - il dominio UNIX (AF_UNIX)
  - il dominio Internet (AF_INET, AF_INET6)
  - il dominio Novell (AF_IPX)
  - il dominio AppleTalk (AF_APPLETALK)

Tipi di Socket

- **Stream socket** forniscono stream di dati affidabili, duplex, ordinati. Nel dominio Internet sono supportati dal protocollo TCP.
- **socket per pacchetti in sequenza** forniscono stream di dati, ma i confini dei pacchetti sono preservati. Supportato nel dominio AF_NS.
- **socket a datagrammi** trasferiscono messaggi di dimensione variabile, preservando i confini ma senza garantire ordine o arrivo dei pacchetti. Supportate nel dominio Internet dal protocollo UDP.
- **socket per datagrammi affidabili** come quelle a datagrammi, ma l'arrivo è garantito. Attualmente non supportate.
- **socket raw** permettono di accedere direttamente ai protocolli che supportano gli altri tipi di socket; p.e., accedere TCP, IP o direttamente Ethernet. Utili per sviluppare nuovi protocolli.

Strutture dati per le socket

Struttura “generica” (fa da jolly)

```
struct sockaddr {
   short sa_family;   /* Address family */
   char sa_data[14];  /* Address data.  */
 };
```

Per indicare una socket nel dominio AF_UNIX

```
struct sockaddr_un {
   short sa_family;       /* Flag AF_UNIX */
   char  sun_path[108];   /* Path name  */
 };
```

Per indicare una socket nel dominio AF_INET

```
struct sockaddr_in {
   short sa_family;       /* Flag AF_INET */
   short sin_port;        /* Numero di porta  */
   struct in_addr sin_addr;  /* indir. IP */
   char  sin_zero[8];     /* riempimento */
 };
```

dove `in_addr` rappresenta un indirizzo IP

```
struct in_addr {
 u_long s_addr;           /* 4 byte */
};
```

Chiamate di sistema per le socket

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

crea una socket. Se il protocollo è 0, il kernel sceglie il protocollo più adatto per supportare il tipo specificato nel dominio indicato.

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

Lega un nome ad una socket. Il dominio deve corrispondere e il nome non deve essere utilizzato.

```
int connect(int sockfd,
            struct sockaddr *serv_addr,
            int addrlen)
```

Stabilisce la connessione tra `sockfd` e la socket indicata da `serv_addr`

Chiamate di sistema per le socket (cont.)

- Un processo server usa `listen` per fissare la coda di clienti e `accept` per mettersi in attesa delle connessioni. Solitamente, per ogni client viene creato un nuovo processo (fork) o thread.
- Una socket viene distrutta chiudendo il file descriptor associato alla connessione o con la `shutdown`.
- Con la `select` si possono controllare trasferimenti di dati da più file descriptors/socket descriptor
- Le socket a messaggi si usano con le system call `sendto` e `recvfrom`

```
int sendto(int s, const void *msg, int len, unsigned int
           flags, const struct sockaddr *to, int tolen);
int recvfrom(int s, void *buf, int len, unsigned int flags
             struct sockaddr *from, int *fromlen);
```

Esempio di server: maiuscolatore

```
/* upperserver.c : un server per "maiuscolare" linee di testo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/times.h>

#define SERVER_PORT 1313
#define LINESIZE 80

void upperlines(int in, int out) {
  char inputline[LINESIZE];
  int  len, i;

  while ((len = read(in, inputline, LINESIZE)) > 0) {
    for (i=0; i < len; i++)
      inputline[i] = toupper(inputline[i]);
    write(out, inputline, len);
  }
}
```

Esempio di server: maiuscolatore

```
int main (unsigned argc, char **argv) {
  int sock, client_len, fd;
  struct sockaddr_in server,client;

  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* prepariamo la struttura per il server */
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(SERVER_PORT);

  /* leghiamo il socket alla porta */
  if (bind(sock, (struct sockaddr *)&server, sizeof server) < 0) {
    perror("binding socket");
    exit(2);
  }

  listen(sock, 1);
```

Esempio di server: maiuscolatore

```
/* ed ora, aspettiamo i "clienti" */
 while (1) {
  client_len = sizeof(client);
  if ((fd=accept(sock, (struct sockaddr *)&client, &client_len))<0) {
    perror("accepting connection");
    exit(3);
  }

  fprintf(stderr, "Aperta connessione.\n");
  write(fd, "Benvenuto all'UpperServer!\n", 27);
  upperlines(fd, fd);
  close(fd);
  fprintf(stderr, "Chiusa connessione.\n");
 }
}
```

Upperserver: esempio di funzionamento

```
miculan@maxi:Socket$ ./upperserver
Aperta connessione.
Chiusa connessione.
Aperta connessione.
Chiusa connessione.
^C
miculan@maxi:Socket$
miculan@coltrane:miculan$ telnet maxi 1313
Trying 158.110.144.170...
Connected to maxi.
Escape character is '^]'.
Benvenuto all'UpperServer!
ahd aksjdh kajsd akshd
AHD AKSJDH KAJSD AKSHD
aSdAs
ASDAS
^]
telnet> close
Connection closed.
miculan@coltrane:miculan$
```

Servire più client contemporaneamente

- Assegnare un thread/processo ad ogni client. Esempio:

```
sock = socket(...); bind(sock, ...);
listen(sock, 2);
while(1) {
  fd = accept(sock,...);
  if (fork() == 0) {
     /* gestione del client */
     ...
     exit(0);
  } else  /* padre */
    close(fd);
}
```

- Si possono usare i processi se i varî server sono scorrelati e per sicurezza; meglio i thread se i server devono accedere a strutture comuni e per maggiore efficienza.
- Se il sistema non supporta i thread ed è necessario avere uno stato comune, meglio usare la `select(2)`

Client connection-oriented: oraesatta

```
/* oraesatta.c : un client TCP per avere l'ora esatta */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/times.h>

/* la porta standard per la data, sia TCP che UDP */
#define DAYTIME_PORT 13
#define LINESIZE 80

int main (unsigned argc, char **argv) {
  int sock, count, ticks;
  struct sockaddr_in server;
  struct hostent *host;
  char inputline[LINESIZE];
  struct tms buffer;

  if (argc != 2) {
    fprintf(stderr, "usage: %s <server address>\n", argv[0]);
    exit(2);
  }

  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }
```

Client connection-oriented: oraesatta

```
  /* recuperiamo l'indirizzo del server */
  host = gethostbyname(argv[1]);
  if (host == NULL) {
    perror("unknown host");
    exit(1);
  }

  /* prepariamo la struttura per il server */
  server.sin_family = AF_INET;
  memcpy(&server.sin_addr.s_addr, host->h_addr, host->h_length);
  server.sin_port = htons(DAYTIME_PORT);

  /* parte il cronometro! */
  ticks = times(&buffer);

  /* colleghiamoci al server */
  if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) {
    perror("connecting to server");
    exit(1);
  }
  /* e leggiamo la data */
  count = read(sock, inputline, LINESIZE);
  ticks = times(&buffer) - ticks;
  printf("Su %s sono le %s", argv[1], inputline);
  printf("Ticks impiegati: %d\n", ticks);

  exit(0);
}
```

Client connection-oriented: oraesatta

Esempio di funzionamento:

```
miculan@maxi:Socket$ ./oraesatta www.sissa.it
Su www.sissa.it sono le Fri May 12 12:38:44 METDST 2000
Ticks impiegati: 8
miculan@maxi:Socket$ ./oraesatta www.sissa.it
Su www.sissa.it sono le Fri May 12 12:38:45 METDST 2000
Ticks impiegati: 9
miculan@maxi:Socket$ ./oraesatta www.sissa.it
Su www.sissa.it sono le Fri May 12 12:38:46 METDST 2000
Ticks impiegati: 8
```

Client connectionless: bigben

```
/* bigben.c : un client UDP per avere l'ora esatta  */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <sys/times.h>
/* la porta standard per la data, sia TCP che UDP */
#define DAYTIME_PORT 13
#define LINESIZE 80

int main (unsigned argc, char **argv)
{
  int sock, count, server_len, ticks;
  struct sockaddr_in client, server;
  struct hostent *host;
  char inputline[LINESIZE];
  struct tms buffer;

  if (argc != 2) {
    fprintf(stderr, "usage: %s <server address>\n", argv[0]);
    exit(2);
  }

  sock = socket(AF_INET, SOCK_DGRAM, 0);
  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }
```

Client connectionless: bigben

```
/* prepariamo la struttura per il client */
client.sin_family = AF_INET;
client.sin_addr.s_addr = htonl(INADDR_ANY);
client.sin_port = 0;

/* leghiamo l'indirizzo */
if (bind(sock,(struct sockaddr *)&client,sizeof client) < 0) {
  perror("bind failed");
  exit(1);
}

/* recuperiamo l'indirizzo del server */
host = gethostbyname(argv[1]);
if (host == NULL) {
  perror("unknown host");
  exit(1);
}

/* prepariamo la struttura per il server */
server.sin_family = AF_INET;
memcpy(&server.sin_addr.s_addr, host->h_addr, host->h_length);
server.sin_port = htons(DAYTIME_PORT);

/* ed ora, spediamo un pacchetto dummy, solo per svegliare il server */
ticks = times(&buffer);
count = sendto(sock, "\n", 1, 0, (struct sockaddr *)&server, sizeof server);
```

Client connectionless: bigben

```
  /* riceviamo la risposta, contenente la data locale */
  server_len = sizeof server;
  count = recvfrom(sock, inputline, LINESIZE, 0,
                   (struct sockaddr *)&server, &server_len);
  ticks = times(&buffer) - ticks;
  printf("Su %s sono le %s", argv[1], inputline);
  printf("Ticks impiegati: %d\n", ticks);

  exit(0);
}
```

```
miculan@maxi:Socket$ ./bigben www.sissa.it
Su www.sissa.it sono le Fri May 12 12:38:37 METDST 2000
Ticks impiegati: 3
miculan@maxi:Socket$ ./bigben www.sissa.it
Su www.sissa.it sono le Fri May 12 12:38:37 METDST 2000
Ticks impiegati: 4
```

Monitoraggio socket: netstat

```
miculan@coltrane:$ netstat --program
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address   Foreign Address         State       PID/Program name
tcp        0      0 coltrane:34179  popmail.inwind.it:pop   ESTABLISHED 1741/fetchmail
tcp        0      0 coltrane:34178  farfarello:http         TIME_WAIT   -
tcp        0      0 coltrane:34174  maxi:ssh                ESTABLISHED -
tcp        0      0 coltrane:34184  lacerta.cc.uni:webcache ESTABLISHED 1267/opera
tcp        0      0 coltrane:34185  lacerta.cc.uni:webcache ESTABLISHED 1267/opera
tcp        0      0 coltrane:34180  lacerta.cc.uni:webcache ESTABLISHED 1267/opera
tcp        0      0 coltrane:34181  lacerta.cc.uni:webcache TIME_WAIT   -
tcp        0      0 coltrane:34182  lacerta.cc.uni:webcache TIME_WAIT   -
tcp        0      0 coltrane:34183  lacerta.cc.uni:webcache TIME_WAIT   -
tcp        0      0 coltrane:34177  lacerta.cc.uni:webcache ESTABLISHED 1267/opera
tcp        0      0 coltrane:ipp    ten.dimi.uniud.it:791   TIME_WAIT   -
tcp        0      0 coltrane:34176  ten.dimi.uniud.it:791   TIME_WAIT   -
tcp        0      0 coltrane:34186  ten.dimi.uniud.it:791   TIME_WAIT   -
tcp        0      0 coltrane:34175  ten.dimi.uniud.it:791   TIME_WAIT   -
tcp        0      0 coltrane:32772  ten.dimi.uniud.it:imap  ESTABLISHED 1233/pine
tcp        0      0 coltrane:32772  ten.dimi.uniud.it:imap  ESTABLISHED 1233/pine
```

Monitoraggio socket: netstat

```
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags Type       State         I-Node PID/Program name    Path
unix  11     [ ]   DGRAM                    815    -                   /dev/log
unix  3      [ ]   STREAM     CONNECTED     5852   -                   /tmp/.X11-unix/X0
unix  3      [ ]   STREAM     CONNECTED     5851   1267/opera
unix  3      [ ]   STREAM     CONNECTED     5462   -                   /tmp/.X11-unix/X0
unix  3      [ ]   STREAM     CONNECTED     5461   1259/xdvi.bin
unix  3      [ ]   STREAM     CONNECTED     2314   -                   /tmp/.X11-unix/X0
unix  3      [ ]   STREAM     CONNECTED     2313   1145/xemacs
unix  3      [ ]   STREAM     CONNECTED     2194   -
unix  3      [ ]   STREAM     CONNECTED     2193   1144/gnome-terminal
unix  3      [ ]   STREAM     CONNECTED     2192   -
unix  3      [ ]   STREAM     CONNECTED     2191   1144/gnome-terminal
unix  3      [ ]   STREAM     CONNECTED     1948   -                   /tmp/.X11-unix/X0
unix  2      [ ]   DGRAM                    1683   -
[...]
unix  2      [ ]   DGRAM                    1619   -
unix  2      [ ]   DGRAM                    1540   -
unix  2      [ ]   DGRAM                    1489   -
unix  2      [ ]   DGRAM                    1273   -
unix  2      [ ]   DGRAM                    1165   -
unix  2      [ ]   DGRAM                    1107   -
unix  2      [ ]   DGRAM                    881    -
unix  2      [ ]   DGRAM                    824    -
miculan@coltrane:$
```

I daemon e i runlevel di Unix

- daemon = un processo di Unix che è sempre in esecuzione: viene lanciato al boot time e terminato allo shutdown da appositi script. Esempio, il sendmail (che gestisce l'email):
  lanciato da `/etc/rc.d/rc3.d/S16sendmail`
  terminato da `/etc/rc.d/rc0.d/K10sendmail`.
- Questi processi offrono servizi di sistema (login remoto, email, server di stampa, oraesatta...) o di "applicazione" (http, database server...).
- Tradizionalmente, il nome termina per "d" (kpiod, kerneld, crond, inetd...).
- Un insieme di daemon in esecuzione forma un runlevel.

I daemon e i runlevel di Unix (cont.)

Convenzionalmente, ci sono 7 runlevel:

- 0 = shutdown: terminazione di tutti i processi
- 1 = single user: no rete, no login multiutente
- 2 = multiuser, senza supporto di rete
- 3 = multiuser, supporto di rete
- 4 = non usato, sperimentale
- 5 = multiuser, supporto di rete, login grafico
- 6 = terminazione di tutti i processi e reboot

I daemon e i runlevel di Unix (cont.)

Ogni runlevel viene definito

- nei BSD-like: da uno script (es. `/etc/rc3`) che viene eseguito dal processo 1 (`init`) al boot
- nei SVR4-like: da una directory di script (es. `/etc/rc.d/rc3.d`), uno per servizio. All'ingresso nel runlevel, quelli che iniziano per `S` vengono eseguiti da `init` con l'argomento "start" (sequenza di startup)

I daemon e i runlevel di Unix (cont.)

- Es, su Linux (RedHat, SysV-like):

```
miculan@coltrane:~$ ls /etc/rc.d/rc3.d
K20nfs        K73ypbind   S10network  S25netfs       S80sendmail
K20rwhod      K74nscd     S12syslog   S28autofs      S85gpm
K46radvd      K74ntpd     S13portmap  S55sshd        S90crond
K50snmpd      K92ipchains S14nfslock  S56rawdevices  S90xfs
K50snmptrapd  K92iptables S17keytable S56xinetd      S95anacron
K65identd     S05kudzu    S20random   S60lpd         S95atd
miculan@coltrane:~$
```

- Si sceglie il runlevel
  - di default: è scritto nella tabella `/etc/inittab`
  - al boot: argomento del kernel (e.g., `linux 1`)
  - al runtime: con il comando `telinit <n>`

I servizi e le porte standard di Unix

- Ad ogni servizio viene assegnata una porta
- I servizi più comuni utilizzano porte standard, sia TCP che UDP
- Le porte < 1024 possono essere utilizzate solo da processi con UID=0, per “garantire” la controparte della liceità del demone
- Chiunque può impiegare porte ≥ 1024, se non sono usati da altri processi
- Si può far girare i servizi su porte non standard (es.: alcuni httpd sono su porte diverse da 80)

Il “super server” `inetd`

- I daemon che vengono lanciati dagli script di runlevel sono “standalone”
- Viceversa, molti demoni standard vengono eseguiti sotto il “super server” inetd (o xinetd): il processo viene creato solo al momento della richiesta
- meno memoria consumata, ma più lento nello startup
- inetd si configura da `/etc/inetd.conf`

```
...
# These are standard services.
#
ftp     stream tcp   nowait  root   /usr/sbin/tcpd in.ftpd -l -a
telnet  stream tcp   nowait  root   /usr/sbin/tcpd in.telnetd
#
# Shell, login, exec, comsat and talk are BSD protocols.
#
shell   stream tcp   nowait  root   /usr/sbin/tcpd in.rshd
login   stream tcp   nowait  root   /usr/sbin/tcpd in.rlogind
...
```

```
miculan@coltrane:miculan$ telnet ten 25
Trying 158.110.144.132...
Connected to ten.
Escape character is '^]'.
220 ten.dimi.uniud.it 5.67a/IDA-1.5
Sendmail is ready at Tue, 20 Apr 1999 12:55:12 +0200
vrfy miculan
250 Marino Miculan <miculan>
mail from: me
250 me... Sender ok
rcpt to: miculan
250 miculan... Recipient ok
data
354 Enter mail, end with "." on a line by itself
questo e` un messaggio di esempio
.
250 Ok
quit
221 ten.dimi.uniud.it closing connection
Connection closed by foreign host.
miculan@coltrane:miculan$
```

Servizi e Sistemi operativi distribuiti

Problemi di progetto:

Trasparenza e località: i sistemi distribuiti dovrebbero apparire come sistemi convenzionali e non distinguere tra risorse locali e remote

Mobilità dell'utente: presentare all'utente lo stesso ambiente (i.e., home dir, applicativi, preferenze,...) ovunque esso si colleghi

Tolleranza ai guasti: i sistemi dovrebbero continuare a funzionare, eventualmente con qualche degrado, anche in seguito a guasti

Scalabilità: aggiungendo nuovi nodi, il sistema dovrebbe essere in grado di sopportare carichi proporzionalmente maggiori

Sistemi su larga scala: il carico per ogni componente del sistema deve essere limitato da una costante indipendente dal numero di nodi, altrimenti non si può scalare oltre un certo limite

Tolleranza ai guasti

Per assicurare che il sistema sia robusto, si deve

- Individuare i fallimenti di link e di siti
- Riconfigurare il sistema in modo che la computazione possa proseguire
- Recuperare lo stato precedente quando un sito/collegamento viene riparato

Rilevamento dei guasti: Handshaking

- Ad intervalli fissati, i siti *A* e *B* si spediscono dei messaggi I-am-up. Se il messaggio non perviene ad *A* entro un certo tempo, si assume che *B* è guasto, o il link è guasto, o il messaggio da *B* è andato perduto
- Quando *A* spedisce la richiesta Are-you-up?, specifica anche un tempo massimo per la risposta. Passato tale periodo, *A* conclude che almeno una delle seguenti situazioni si è verificata:
  - B è spento
  - Il link diretto (se esiste) da *A* a *B* è guasto
  - Il percorso alternativo da *A* a *B* è guasto
  - Il messaggio è andato perduto

  Non si può sapere con certezza quale evento si è verificato

Riconfigurazione

- È una procedura che permette al sistema di riconfigurarsi e continuare il funzionamento
- Se il link tra *A* e *B* si è guastato, questa informazione deve essere diramata agli altri siti del sistema, in modo che le tabelle di routing vengano aggiornate
- Se si ritiene che un sito si è guastato, allora ogni sito ne viene notificato affinché non cerchi di usare i servizi del sito guasto

Ripristino dopo un guasto

- Quando un collegamento o sito guasto viene recuperato, deve essere reintegrato nel sistema in modo semplice e lineare
- Ad esempio, se il collegamento tra *A* e *B* era guasto, quando viene riparato sia *A* che *B* devono essere notificati. Si può implementare ripetendo la procedura di handshaking
- Se il sito *B* era guasto, quando viene ripristinato deve notificare gli altri siti del sistema che è di nuovo in piedi. Il sito *B* può quindi ricevere dati dagli altri sistemi per aggiornare le tabelle locali

Modelli dei servizi distribuiti

Modi essenziali per implementare un modello omogeneo per un servizio distribuito: il middleware può implementare

Migrazione di dati: offre un modello di dati omogeneo tra i nodi (non distinguo “dove stanno i dati”)

Migrazione delle computazioni: offre un modello uniforme di calcolo distribuito (non distinguo “dove viene eseguita la prossima istruzione”)

Migrazione dei processi: offre un modello uniforme di schedulazione (non distinguo “dove viene eseguito un processo”)

Coordinazione distribuita: offre un modello uniforme di memoria associativa distribuita (non distinguo “dove stanno i dati consumabili”)

Migrazione di dati

Quando un processo deve accedere ad un dato, si procede al trasferimento dei dati dal server al client.

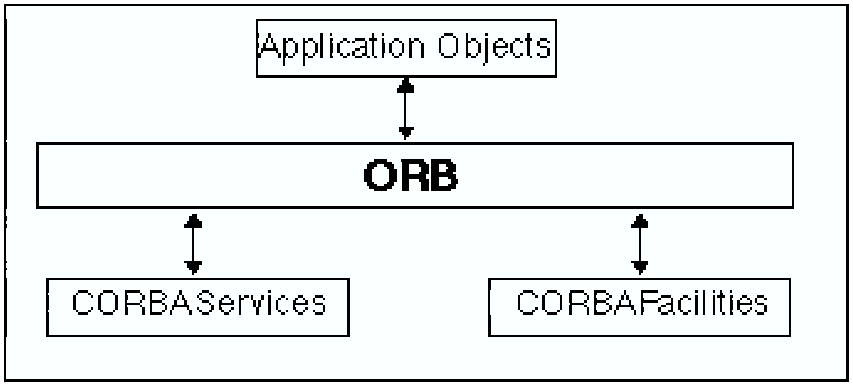
- Middleware basato su documenti: un modo uniforme per raccogliere ed organizzare documenti distribuiti eterogenei. Es: WWW, Lotus Notes.
- Middleware basato su file system distribuiti: decentralizzazione dei dati.
  - upload/download: copie di interi file (AFS, Coda) $\Rightarrow$ caching su disco locale, adeguato per accessi a tutto il file (i.e., connectionless)
  - accesso remoto: copie di parti di file (NFS, SMB) $\Rightarrow$ efficiente per pochi accessi, adeguato su reti locali (affidabili)

Migrazione di computazioni

- Quando un processo deve accedere ad un dato, si procede al trasferimento della computazione dal client al server. Il calcolo avviene sul server e solo il risultato viene restituito al client.
- Efficiente se trasferire la computazione costa meno dei dati
- client e server rimangono processi separati
- Implementazioni tipiche:
  - RPC (Remote Procedure Calls), RMI (Remote Method Invocation): il server implementa un tipo di dato astratto (o un oggetto) le cui funzioni (metodi) sono accessibili dai programmi client.
  - CORBA, Globe: il middleware (es. i server Object Request Brokers) instrada chiamate a metodi di oggetti remoti. L'utente vede un insieme di oggetti condivisi, senza sapere dove sono realmente localizzati.

Migrazione trasparente di computazioni: CORBA in OMA

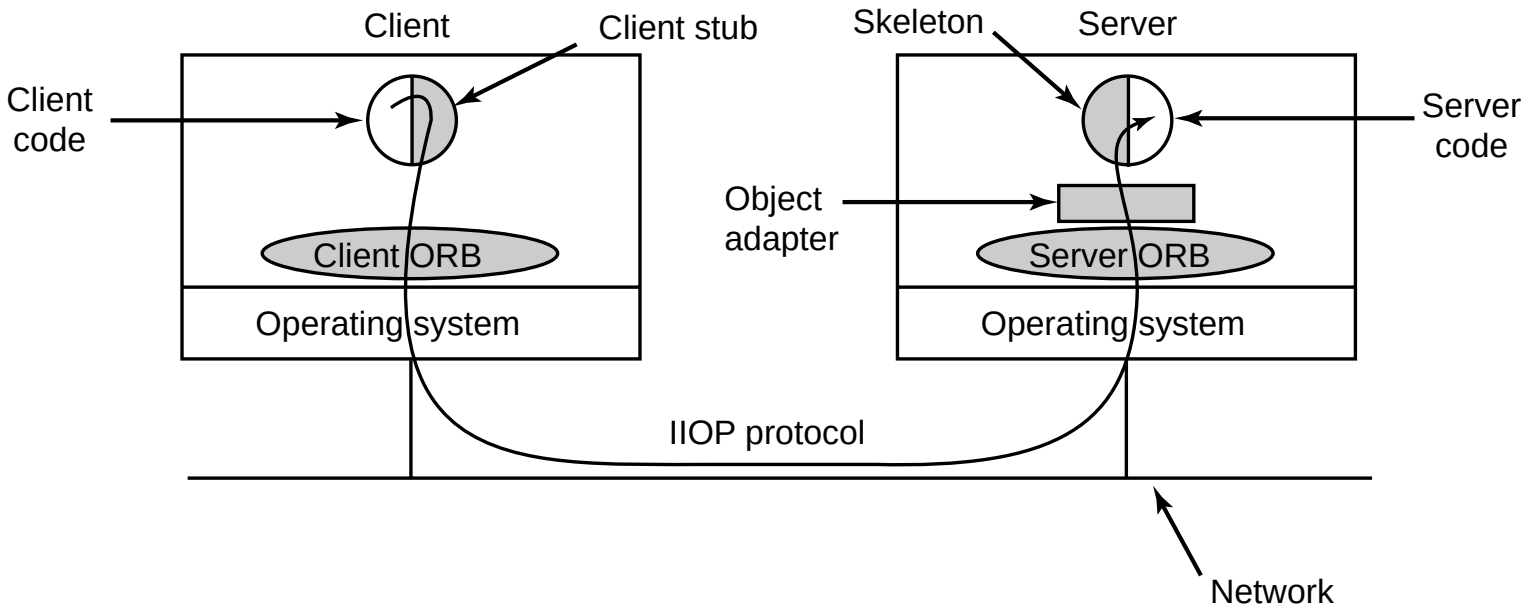
- Parte della Object Management Architecture, pensata per esser cross-platform (sistemi operativi, architetture, linguaggi differenti)



- Un oggetto CORBA esporta un insieme di metodi specificati da una interfaccia descritta in un formato standard (Interface Definition Language)
- La creazione di un oggetto CORBA restituisce un riferimento che può essere passato ad altri oggetti, o inserito in directories.
- Un processo client può chiamare i metodi di qualsiasi oggetto di cui ha il riferimento (che può ottenere da una directory)

CORBA (Cont.)

- La chiamata è mediata dal middleware, i server ORB. Simile a RMI.



- Una chiamata può attraversare più ORB. Protocollo standard (IIOP) per la comunicazione client-server e server-server.
- Limitazione: ogni oggetto è localizzato su un solo server $\Rightarrow$ scalabilità limitata. Risolto in Globe.

Migrazione di processi/thread

- Interi processi (o thread) vengono spostati da una macchina all'altra. L'utente non sa dove effettivamente viene eseguito un suo processo/thread.
- Gli scheduler dei singoli sistemi operativi comunicano per mantenere un carico omogeneo tra le macchine.
- Vantaggi:
  - Bilanciamento di carico
  - Aumento delle prestazioni (parallelismo)
  - Utilizzo di software/hardware specializzati
  - Accesso ai dati

  Svantaggio: spostare un processo è costoso e complesso
- Esempio: MOSIX, OSF DCE, Solaris Full Moon, Microsoft Wolfpack.

Coordinazione su memoria associativa distribuita

- Modo moderno per unire coordinazione e condivisione di dati
- Viene mantenuto uno spazio di memoria associativa distribuita (“tuplespace”) ove possono essere caricate delle tuple (record, struct). Non sono oggetti, solo liste di scalari:

```
("abc", 2, 5) ("matrix-1", 1, 6, 3.14)
```

- Operazioni sullo spazio condiviso:
  - `out("abc",2,5)`: inserimento (non bloccante)
  - `in("abc", 2, ?i)`: rimozione bloccante con pattern matching
  - `read("abc", 2, ?i)`: lettura non distruttiva con pattern matching
  - `eval("worker", worker())`: creazione di un processo nel TupleSpace

Coordinazione distribuita

- Esempio: implementazione di un semaforo condiviso:

```
up(S) = Out("semaforo", S);
down(S) = In("semaforo", S);
```

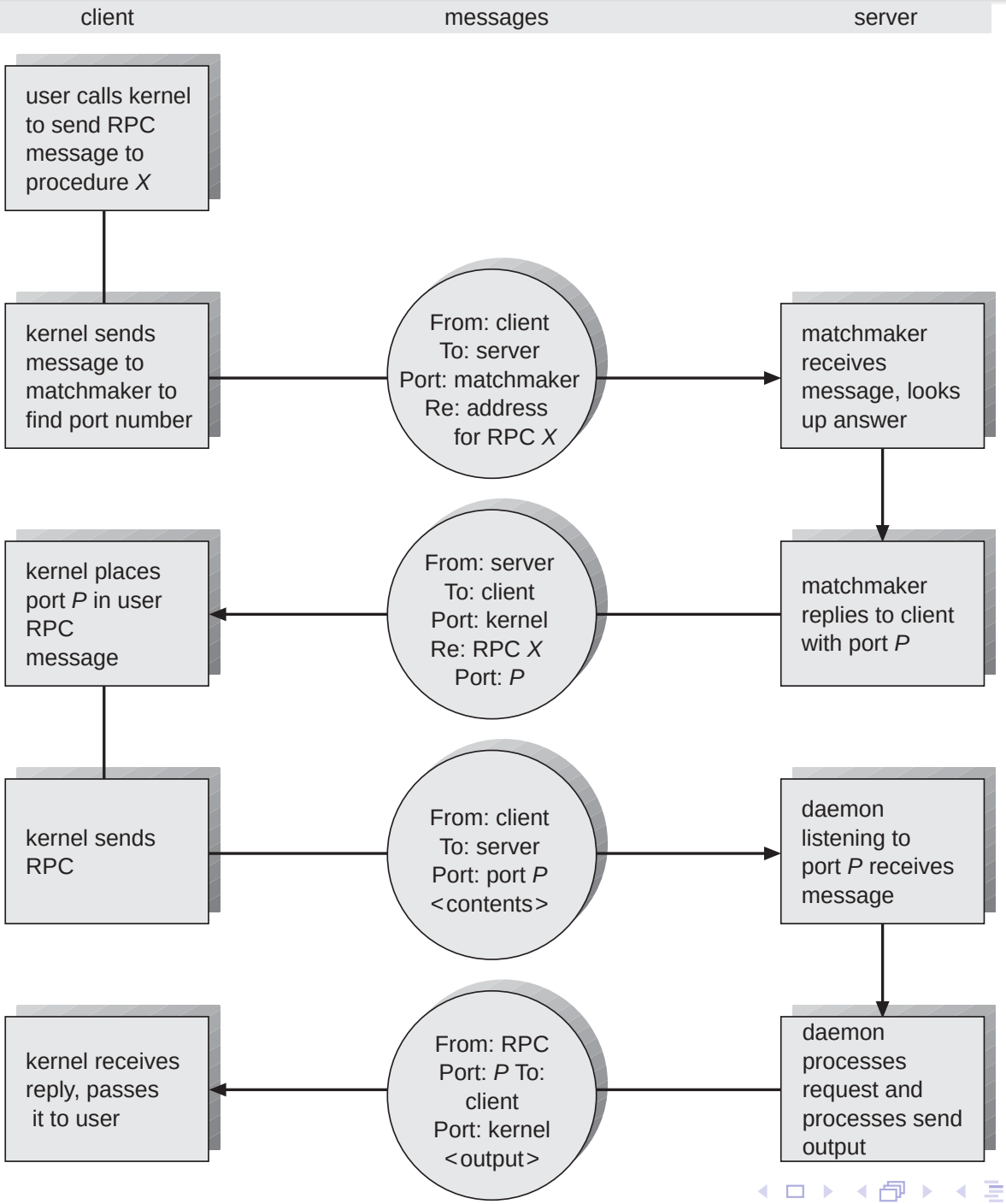
- Pro: flessibili, generali
- Cons: difficili da implementare in modo realmente distribuito (spesso si usa un server centrale)
- Implementato in Linda, `publish/subscribe`, Jini, Klaim

Servizi distribuiti su RPC

- Richieste di accesso ad un file remoto: la richiesta è tradotta in messaggi al server, che esegue l'accesso al file, impacchetta i dati in un messaggio e lo rispedisce indietro al client.
- Un modo comune per implementare questo meccanismo è con le Remote Procedure Call (RPC), concettualmente simili a chiamate locali
  - I messaggi inviati ad un demone RPC in ascolto su una porta indicano quale procedura eseguire e con quali parametri. La procedura viene eseguita, e i risultati sono rimandati indietro con un altro messaggio
  - Una porta è un numero incluso all'inizio del messaggio. Un sistema può avere molte porte su un solo indirizzo di rete, per distinguere i servizi supportati.
- Maggiore differenza rispetto a chiamate locali: errori di comunicazione ne cambiano la semantica. Il S.O. deve cercare di ottenere questa astrazione.

Associazione porta/servizio per RPC

- Le informazioni di collegamento possono essere fissate (porta fissata a priori)
  - Al compile time, ogni chiamata RPC ha un numero di porta fissato
  - Il server non può cambiare la porta senza ricompilare il client
  - Molte porte/servizi sono standardizzati (eg: telnet=23, SMTP=25, HTTP=80 (ma questi non sono servizi RPC))
- L'associazione può essere stabilita dinamicamente con un meccanismo di rendezvous
  - Il S.O. implementa un demone di rendezvous su una porta fissata (111: portmapper o matchmaker)
  - I client, prima di eseguire la vera RPC, chiedono qual è la porta da utilizzare al demone di rendezvous


client
messages
server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client To: server Port: matchmaker Re: address for RPC X
matchmaker receives message, looks up answer
kernel places port P in user RPC message
From: server To: client Port: kernel Re: RPC X Port: P
matchmaker replies to client with port P
kernel sends RPC
From: client To: server Port: port P <contents>
daemon listening to port P receives message
kernel receives reply, passes it to user
From: RPC Port: P To: client Port: kernel <output>
daemon processes request and processes send output

Esempio di schema RPC

Un file system distribuito può essere implementato come un insieme di chiamate RPC

- I messaggi sono inviati alla porta associata al demone file server, sulla macchina su cui risiedono fisicamente i dati.
- I messaggi contengono l'indicazione dell'operazione da eseguirsi (i.e., **read**, **write**, **rename**, **delete**, o **status**).
- Il messaggio di risposta contiene i dati risultati dall'esecuzione della procedura, che è eseguita dal demone per conto del client

Il Sun Network File System (NFS)

- Una implementazione e specifica di un sistema software per accedere file remoti attraverso LAN (o WAN, se proprio uno deve...)
- Le workstation in rete sono viste come macchine indipendenti con file system indipendenti, che permettono di condividere in maniera trasparente
  - Una dir remota viene montata su una dir locale, come un file system locale.
  - La specifica della dir remota e della macchina non è trasparente: deve essere fornita. I file nella dir remota sono accessibili in modo trasparente
  - A patto di averne i diritti, qualsiasi file system o dir entro un file system può essere montato in remoto

NFS (Cont.)

- NFS è progettato per funzionare in un ambiente eterogeneo, di macchine e S.O. differenti; la specifica è indipendente dagli strati sottostanti
- Questa indipendenza si ottiene attraverso chiamate RPC costruite sopra il protocollo eXternal Data Representation (XDR)
- La specifica NFS distingue tra i servizi del protocollo di mount e quello di vero accesso ai file remoti

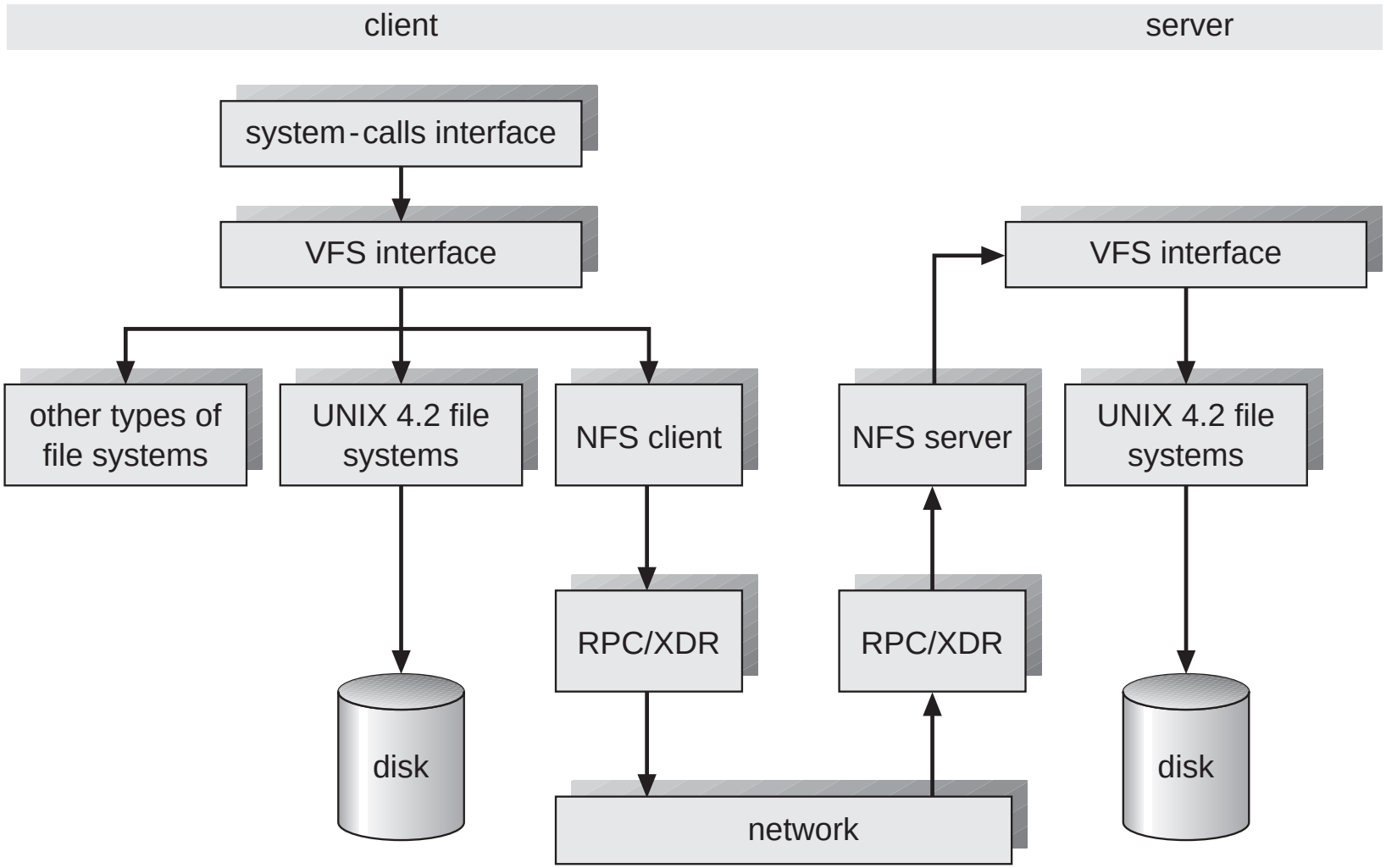
Protocollo NFS Mount

- Stabilisce la connessione logica iniziale tra server e client
- L'operazione di mount include il nome della dir remota e della macchina server che lo contiene
  - La richiesta di mount è tradotta in una chiamata RPC e inoltrata al mount daemon sulla macchina server
  - Export list: specifica quali file system locali sono esportati per il mounting, con il nome delle macchine che possono montarli
- Il risultato dell'operazione di mount è un file handle, una chiave per le richieste successive.
- File handle: file-system identifier, e un numero di inode che identifica la directory montata nel file system esportato.
- L'operazione di mount cambia solo lo stato del client ma non modifica lo stato del server

Protocollo NFS

- Fornisce un insieme di RPC per le operazioni remote su file.
  - ricerca di un file in una directory
  - lettura di un insieme di entries in una dir
  - manipolazione di link e dir
  - accesso agli attributi dei file
  - lettura e scrittura dei file
- I server NFS sono stateless: ogni richiesta è a se stante, e deve fornire tutti gli argomenti che servono
- Per aumentare l'efficienza, NFS usa cache di inode e di blocchi sia sul client che sul server, con read-ahead (lettura dei blocchi anticipata) e write-behind (scrittura asincrona posticipata) $\Rightarrow$ problemi di consistenza
- Il protocollo NFS non fornisce controlli di concorrenza (accesso esclusivo, locking, ...). Vengono implementati da un altro server (lockd)

Architettura NFS



Architettura NFS