Introduzione ai Sistemi Operativi

Programma – Sistemi Operativi

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

Sistema Operativo

- Un sistema operativo è un software che gestisce l'hardware di un calcolatore.

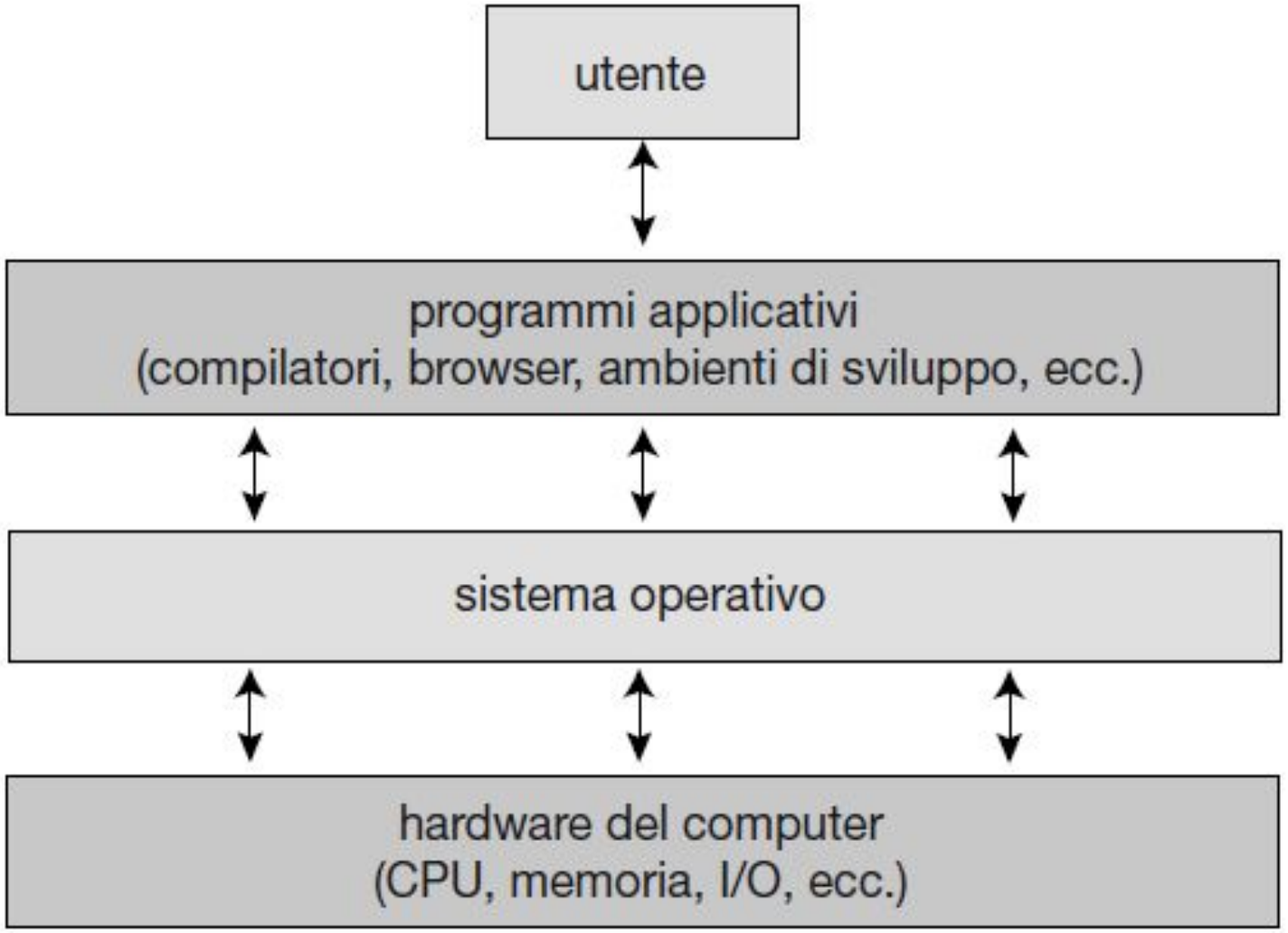
- Lo scopo di un sistema operativo è quello di fornire all'utente un ambiente nel quale l'esecuzione dei programmi possa avvenire in modo conveniente ed efficace.

Il ruolo del sistema operativo

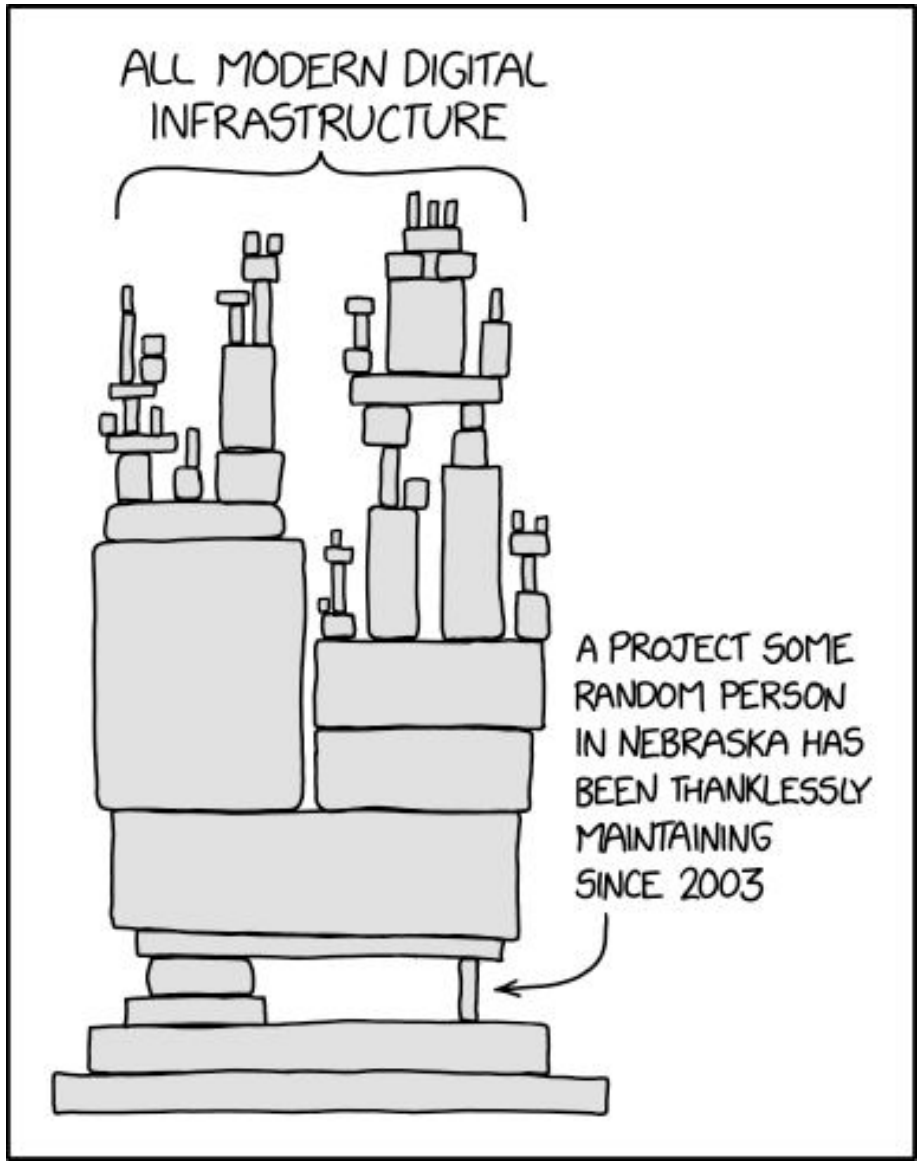
- Un sistema di elaborazione si può suddividere in quattro componenti:

- Un sistema elaborativo si può anche considerare come l'insieme di

Componenti di un sistema elaborativo



Componenti software di un sistema elaborativo



Definizione di Sistema Operativo

1. I **sistemi operativi** esistono poiché rappresentano una soluzione ragionevole al problema di realizzare un sistema elaborativo che si possa impiegare facilmente, per eseguire i programmi e agevolare la soluzione dei problemi degli utenti.
2. Il sistema operativo è il solo programma che funziona sempre nel calcolatore, generalmente chiamato **kernel** (*nucleo*).
3. Oltre al kernel vi sono due tipi di programmi: i **programmi di sistema**, associati al sistema operativo, ma che non fanno necessariamente parte del kernel, e i **programmi applicativi**, che includono tutti i programmi non correlati al funzionamento del sistema.
4. I sistemi operativi mobili non sono costituiti esclusivamente da un kernel, ma anche da un **middleware**, ovvero da una collezione di ambienti software che fornisce servizi aggiuntivi per chi sviluppa applicazioni.

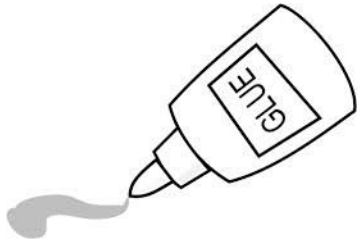
Definizione di Sistema Operativo

Referee

- Manage sharing of resources, protection, isolation
- Resource allocation, isolation, communication

Illusionist

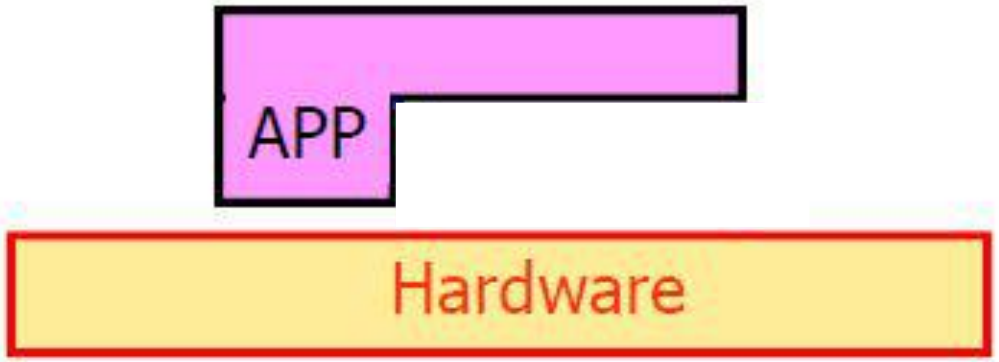
- Provide clean, easy to use abstractions of physical resources
- Infinite memory, dedicated machine
- Higher level objects: files, users, messages
- Masking limitations, virtualization



Glue

- Common services
- Storage, Window system, Networking
- Sharing, Authorization
- Look and feel

History OS: Evolution Step 0

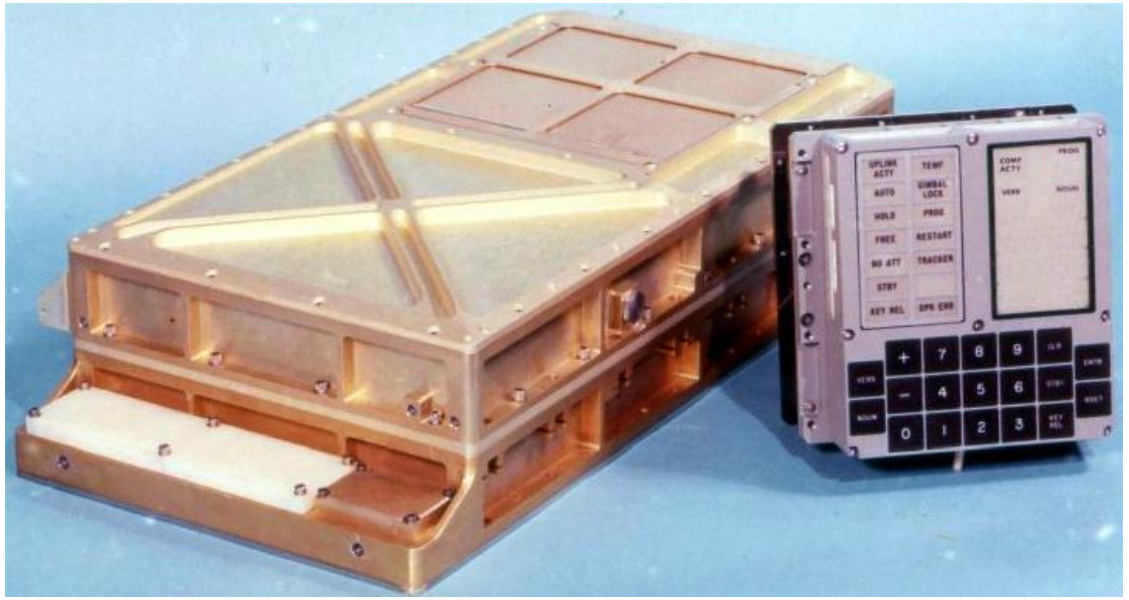


- Simple OS: One program, one user, one machine:
  - examples: early computers, early PCs,
  - embedded controllers such as Nintendo, cars, elevators
  - OS just a library of standard services, e.g. standard device drivers, interrupt handlers, I/O
- Non-problems: No malicious people. No bad programs

  ⇒ A minimum of complex interactions
- Problem: poor utilization, expensive

Apollo Guidance Computer

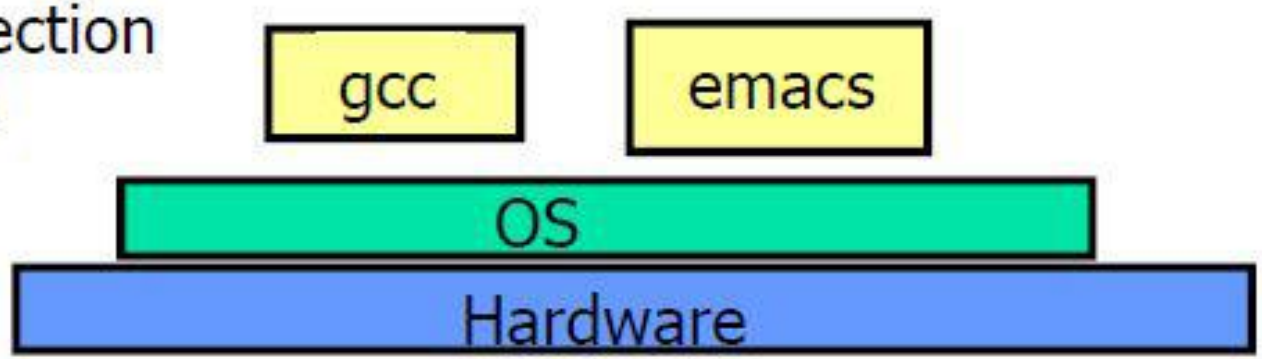
- AGC software was written in AGC assembly language and stored on rope memory. The bulk of the software was on read-only rope memory and thus could not be changed in operation.



- There was a simple real-time operating system designed by J. Halcombe Laning, consisting of the Exec, a batch job-scheduling using cooperative multi-tasking and an interrupt-driven pre-emptive scheduler called the Waitlist, which could schedule multiple timer-driven 'tasks'. The tasks were short threads of execution which could reschedule themselves for re-execution on the Waitlist, or could kick off a longer operation by starting a "job" with the Exec.

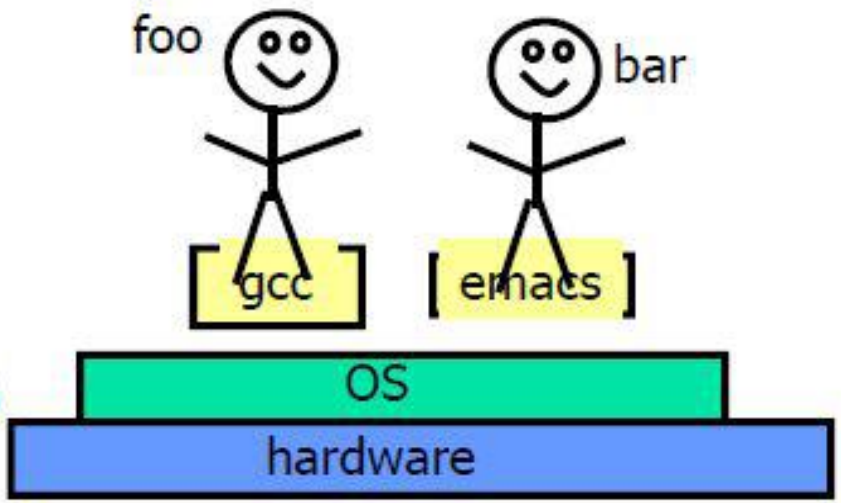
History OS: Evolution Step 1

- Simple OS is inefficient:
  - a waiting process blocks everything else on the machine
- (Seemingly) Simple hack:
  - run more than one process at once
  - when one process blocks, switch to another
- A couple of problems: *what if a program*
  - *does infinite loops or*
  - *starts randomly scribbling on memory?*
- OS adds protection
  - Interposition
  - Preemption
  - Privilege

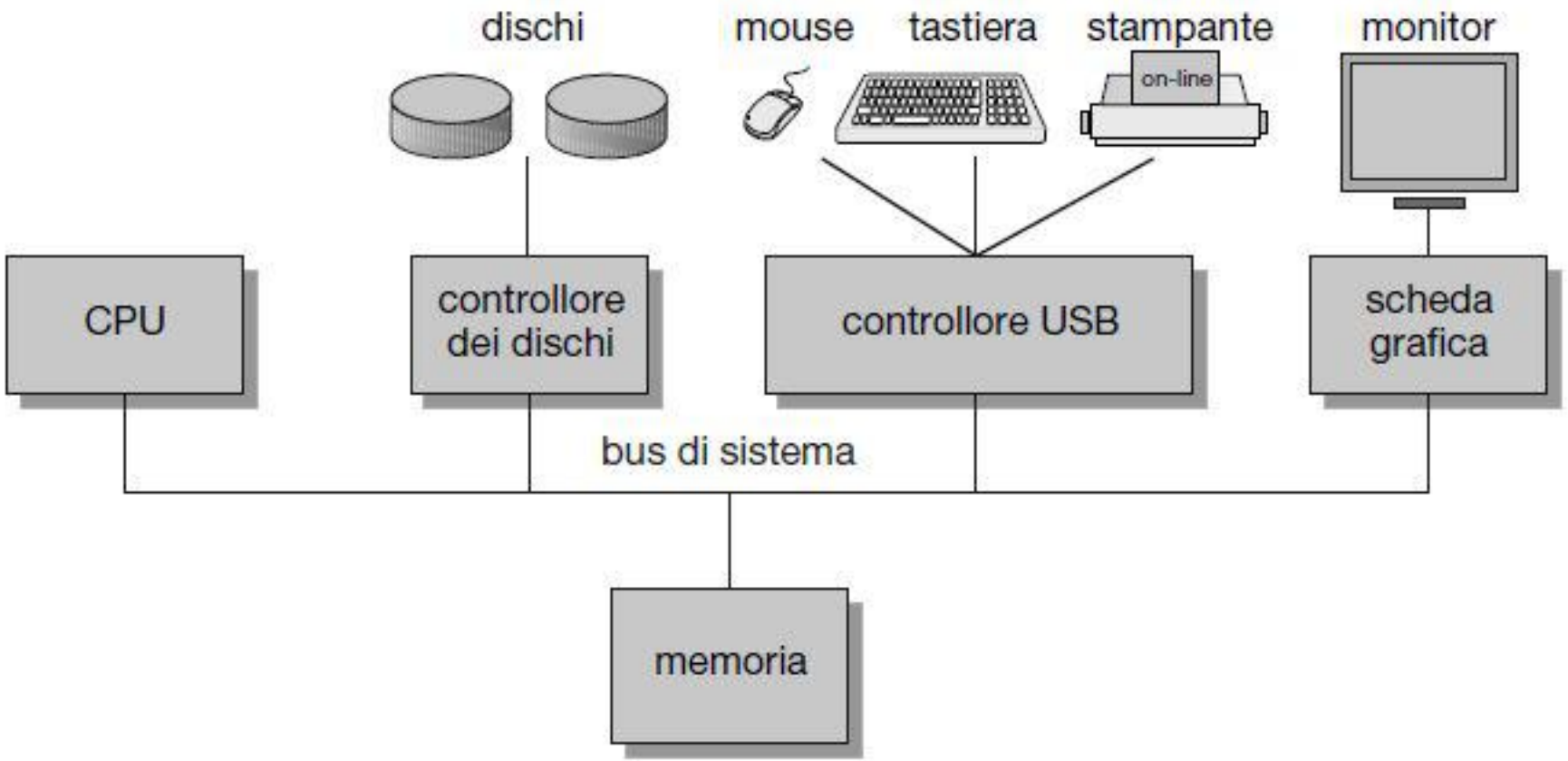


History OS: Evolution Step 2

- Simple OS is too expensive:
  - one user = one computer ⇒
- (Seemingly) simple hack:
  - Allow more than one user at once
  - *Does machine now run n times slower?* Usually not
  - Key observation: users are active in bursts
  - If idle, give resources to others
- Problems: *what if*
  - *users are greedy*
  - *evil*
  - *or just too numerous?*
- OS adds protection
  - (notice: as we try to utilize resources, complexity grows)



Componenti hardware di un sistema elaborativo



Un tipico sistema elaborativo.

Organizzazione di un sistema elaborativo

1. Un moderno calcolatore general-purpose è composto da una o più **CPU** e da un certo numero di **controllori di dispositivi** connessi attraverso un canale di comunicazione comune (*bus)* che permette l'accesso alla memoria condivisa dal sistema (Figura 1.2).
2. I sistemi operativi possiedono in genere per ogni controllore di dispositivo un **driver del dispositivo** che gestisce le specificità del controllore e funge da interfaccia uniforme con il resto del sistema.
3. La **CPU** e i **controllori** possono eseguire operazioni in parallelo, competendo per i cicli di memoria.

interruzioni

- L'hardware della CPU dispone di un filo chiamato **linea di richiesta di interruzione** (*interrupt-request line*) che la CPU controlla dopo l'esecuzione di ogni istruzione.

- La maggior parte delle CPU ha **due linee di richiesta di interruzione**:

1. non mascherabile (*nonmaskable interrupt*) ⟶ riservata a eventi come errori irreversibili di memoria

2. mascherabile (*maskable interrupt)* ⟶ utilizzata dai controllori dei dispositivi per richiedere un servizio

Concatenamento delle interruzioni (*interrupt chaining*) ⟶ ogni elemento nel vettore delle interruzioni punta alla testa di un elenco di gestori

Interruzioni

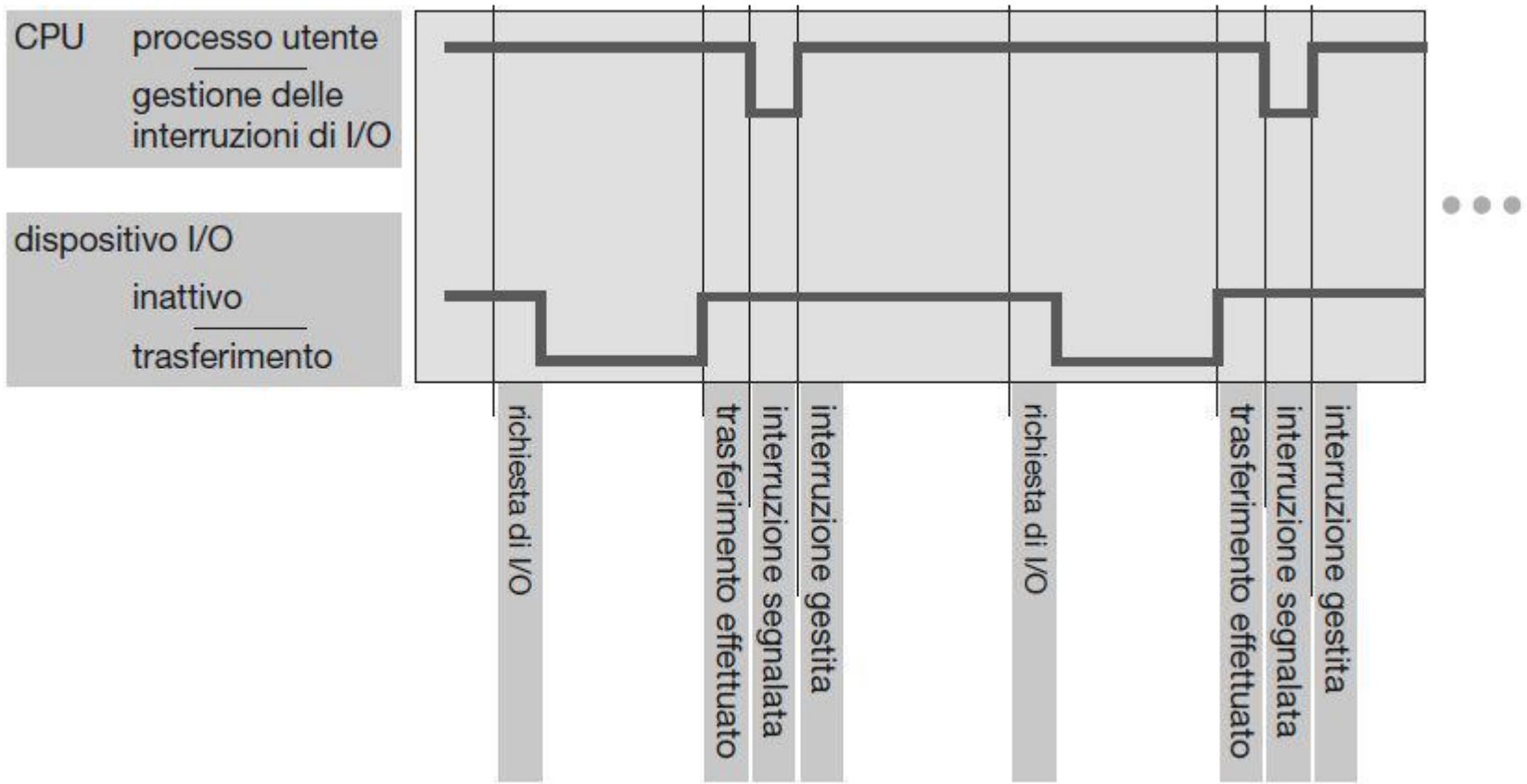
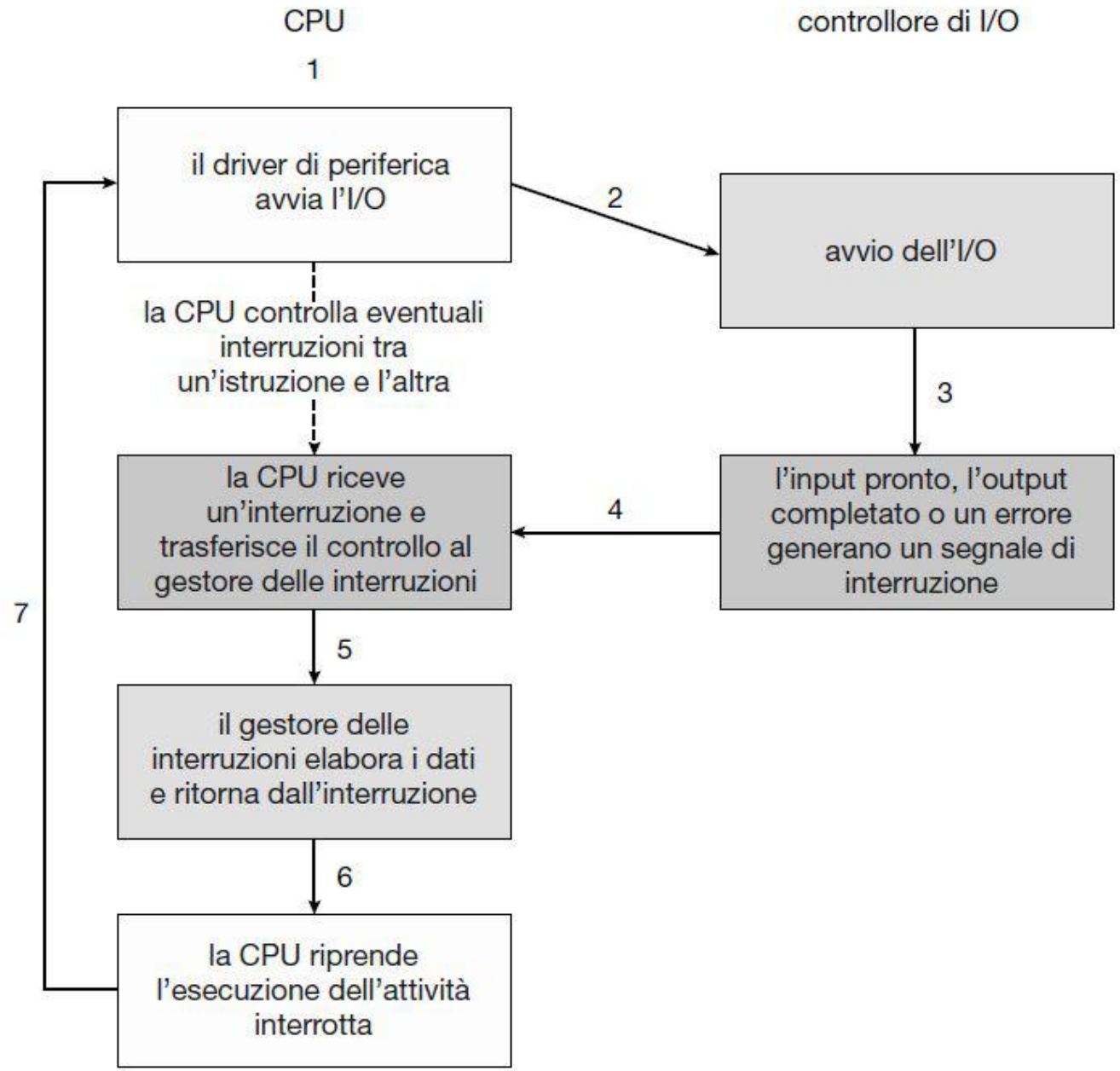


Diagramma temporale delle interruzioni per un singolo programma che invia dati in output.

Ciclo di I/O interrupt driven



Ciclo di I/O guidato dalle interruzioni.

Eventi

numero di vettore	descrizione
0	errore di divisione
1	eccezione di debug
2	interruzione null
3	breakpoint
4	eccezione di overflow
5	eccezione di range exceeded
6	codice operativo non valido
7	dispositivo non disponibile
8	doppio errore
9	overrun del segmento coprocessore (riservato)
10	task state segment (tss) non valido
11	segmento non presente
12	errore di stack
13	protezione generale
14	errore di pagina
15	(riservato Intel, non utilizzare)
16	errore in virgola mobile
17	controllo dell'allineamento
18	controllo della macchina
19–31	(riservato Intel, non utilizzare)
32–255	interruzioni mascherabili

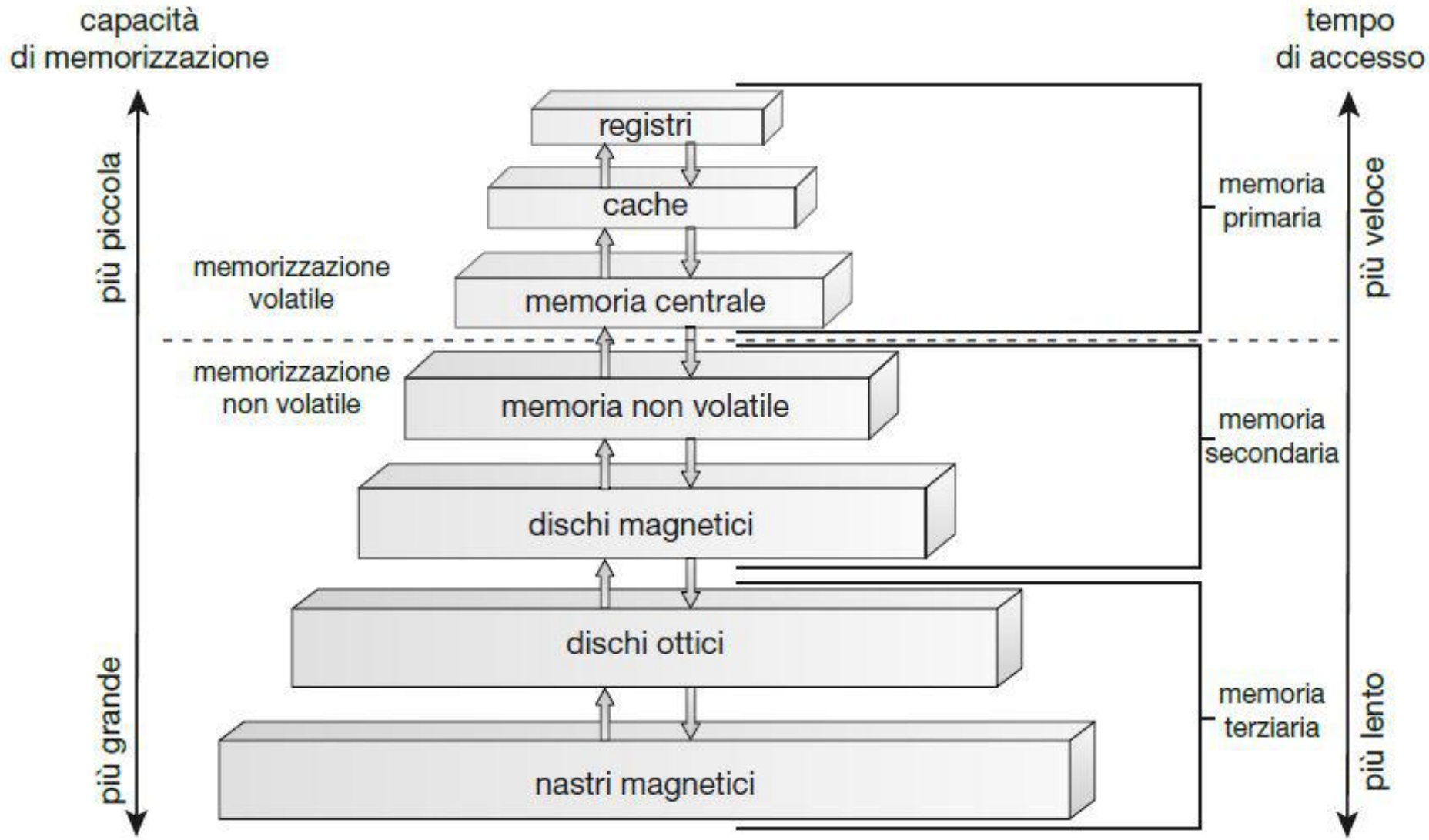
Tabella degli eventi di un processore Intel.

Struttura della memoria

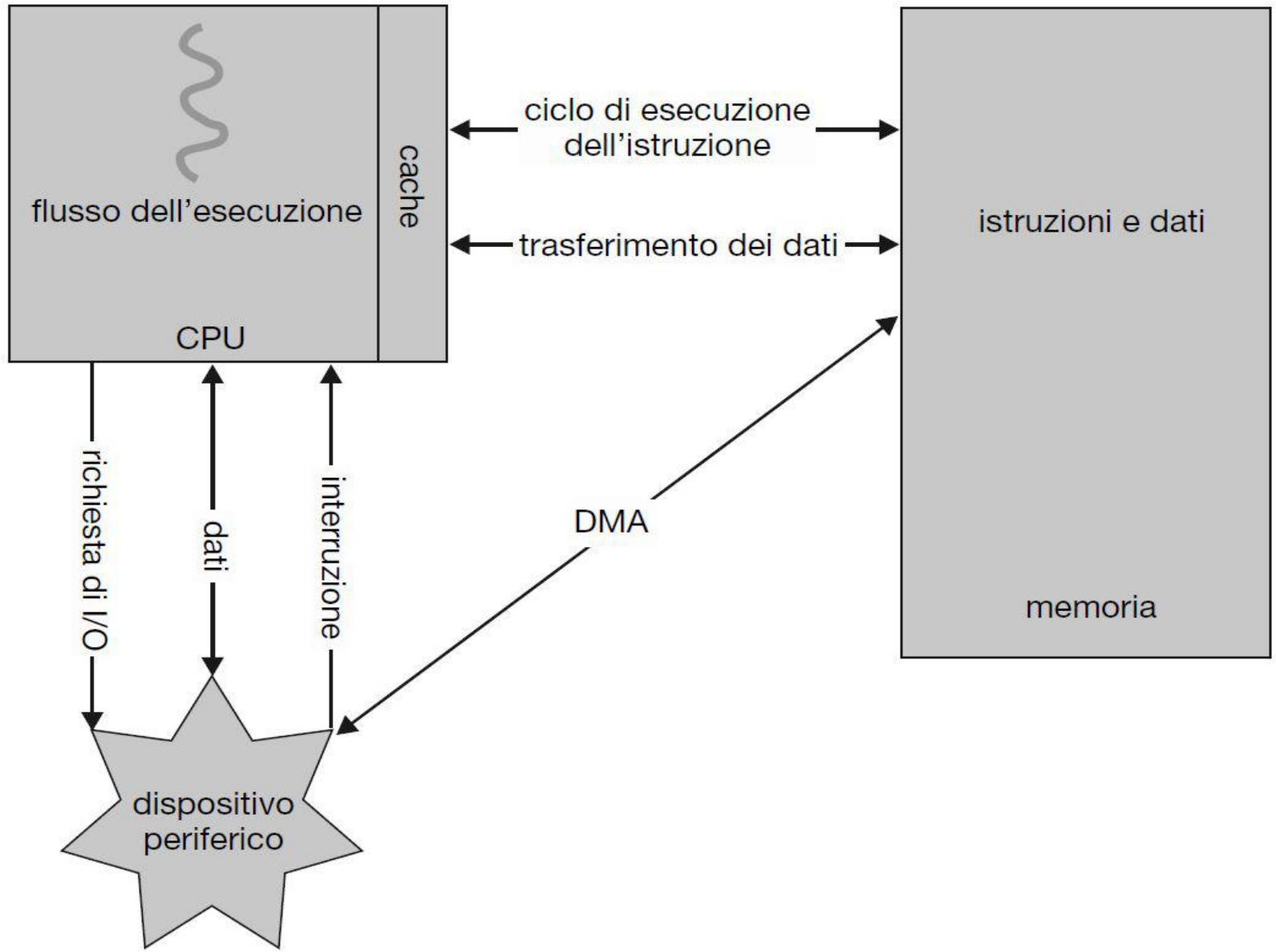
MEMORIA PRINCIPALE O CENTRALE: **memoria ad accesso casuale** (*random access memory, RAM)* → *memoria volatile*

I computer utilizzano una **memoria di sola lettura elettricamente cancellabile e programmabile (EEPROM)** e altre forme di archiviazione del firmware che vengono riscritte raramente e che non sono volatili.

MEMORIA SECONDARIA: estensione della memoria centrale → capacità di conservare in modo permanente grandi quantità di informazioni



Scala gerarchica dei sistemi di memorizzazione.



Funzionamento di un moderno sistema operativo.

Architettura degli elaboratori

Sistemi monoprocessore

↓

Diversi anni fa la maggior parte dei sistemi utilizzava un solo processore contenente un'unica CPU con un unico nucleo di elaborazione (o unità di calcolo, o *core).*

Sistemi multiprocessore

↓

Multielaborazione simmetrica

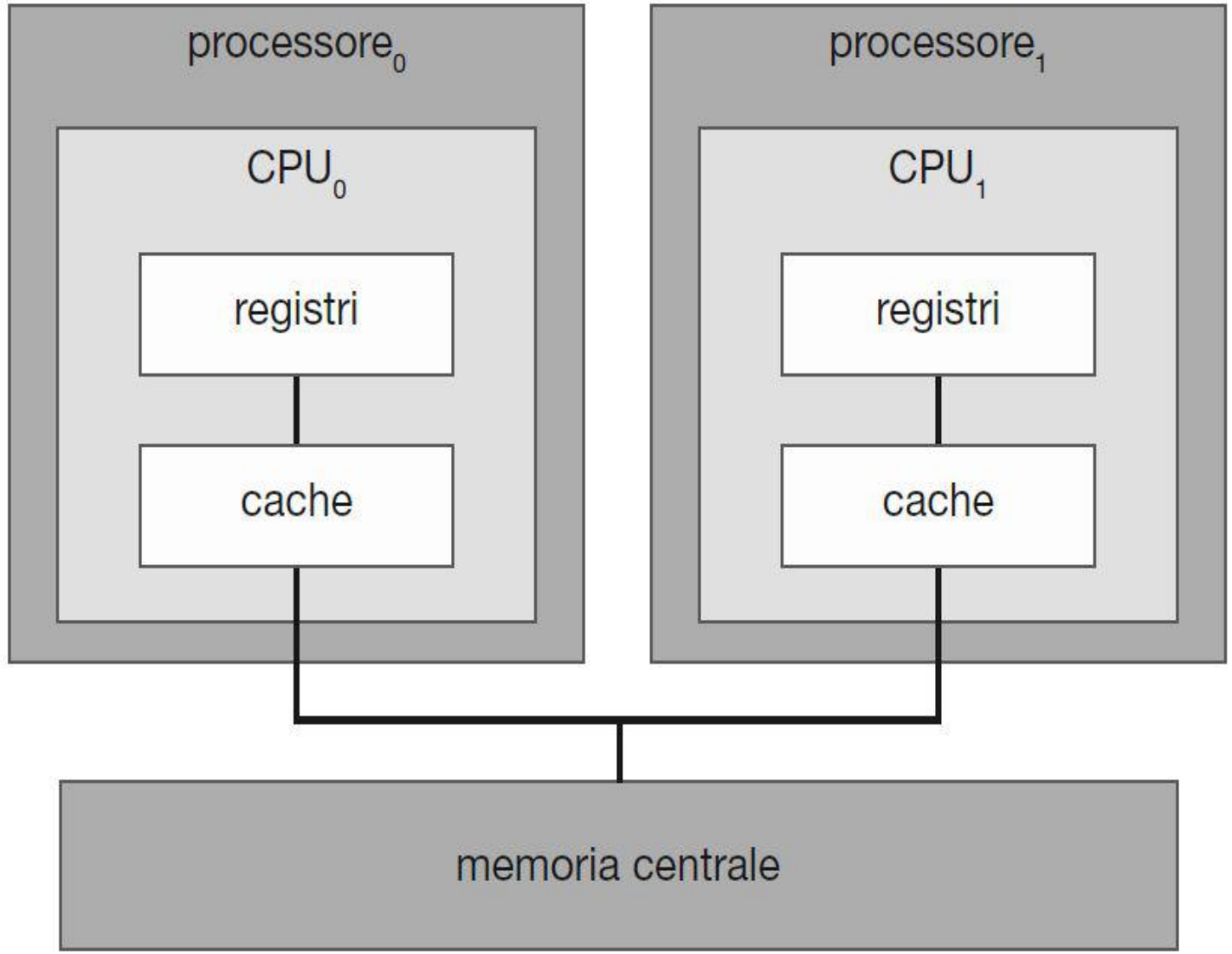
La definizione di multiprocessore si è evoluta nel tempo e include ora i sistemi multicore, in cui più unità di calcolo (core) risiedono su un singolo chip.

↓

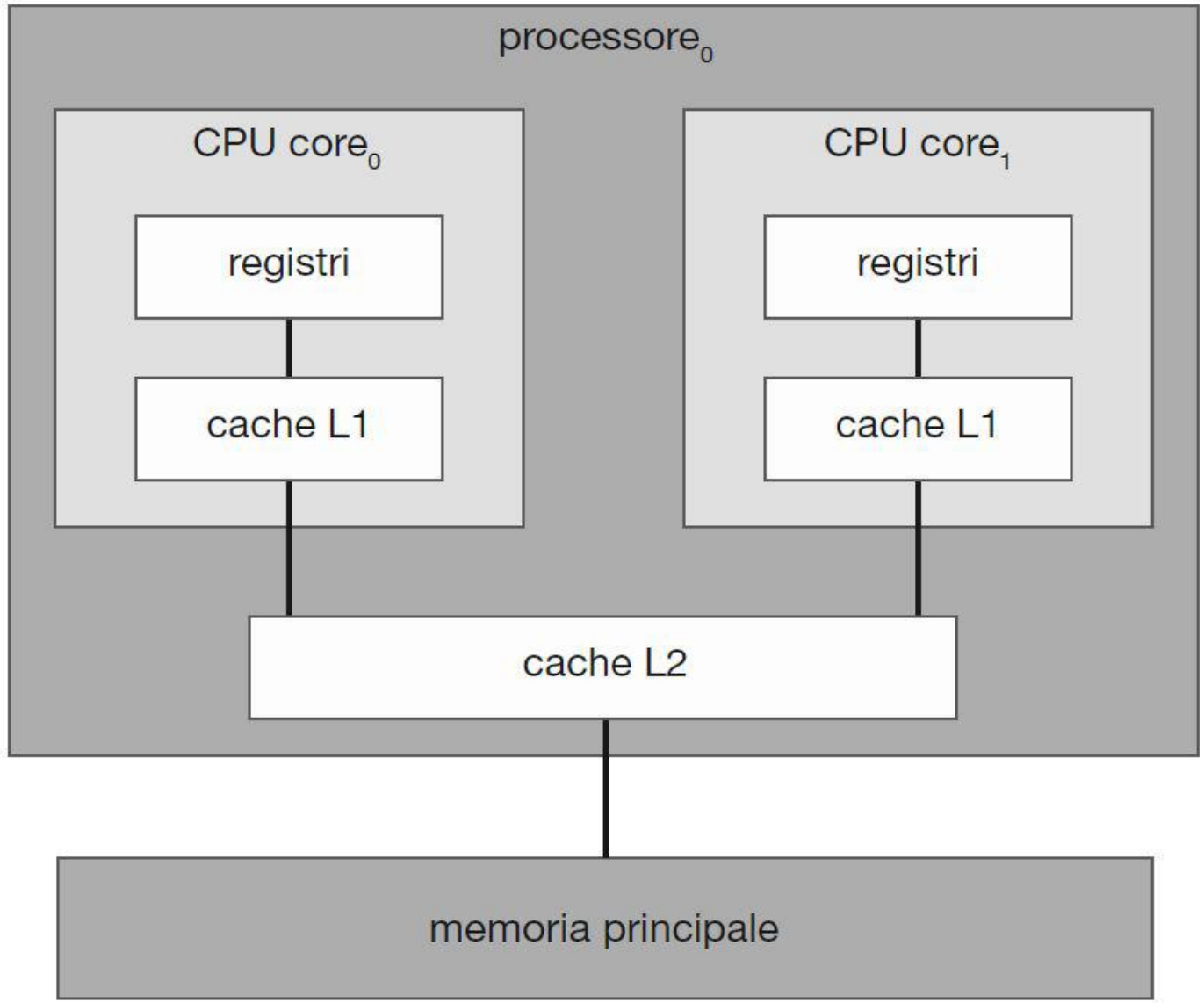
Architettura **dual-core** = due unità sullo stesso chip

Architettura degli elaboratori

- **CPU**: componente hardware che esegue le istruzioni.
- **Processore**: chip che contiene una o più CPU.
- **Unità di calcolo** (*core*): unità di elaborazione di base della CPU.
- **Multicore**: che include più unità di calcolo sulla stessa CPU.
- **Multiprocessore**: che include più processori.



Architettura di multielaborazione simmetrica.



Architettura dual-core, con due unità sullo stesso chip.

Aggiungere **nuove CPU** a un multiprocessore ne aumenta la potenza di calcolo, ma ne

peggiora le prestazioni → fornire a ciascuna CPU (o a ciascun gruppo di CPU) la propria memoria locale accessibile per mezzo di un bus locale piccolo e veloce

↓

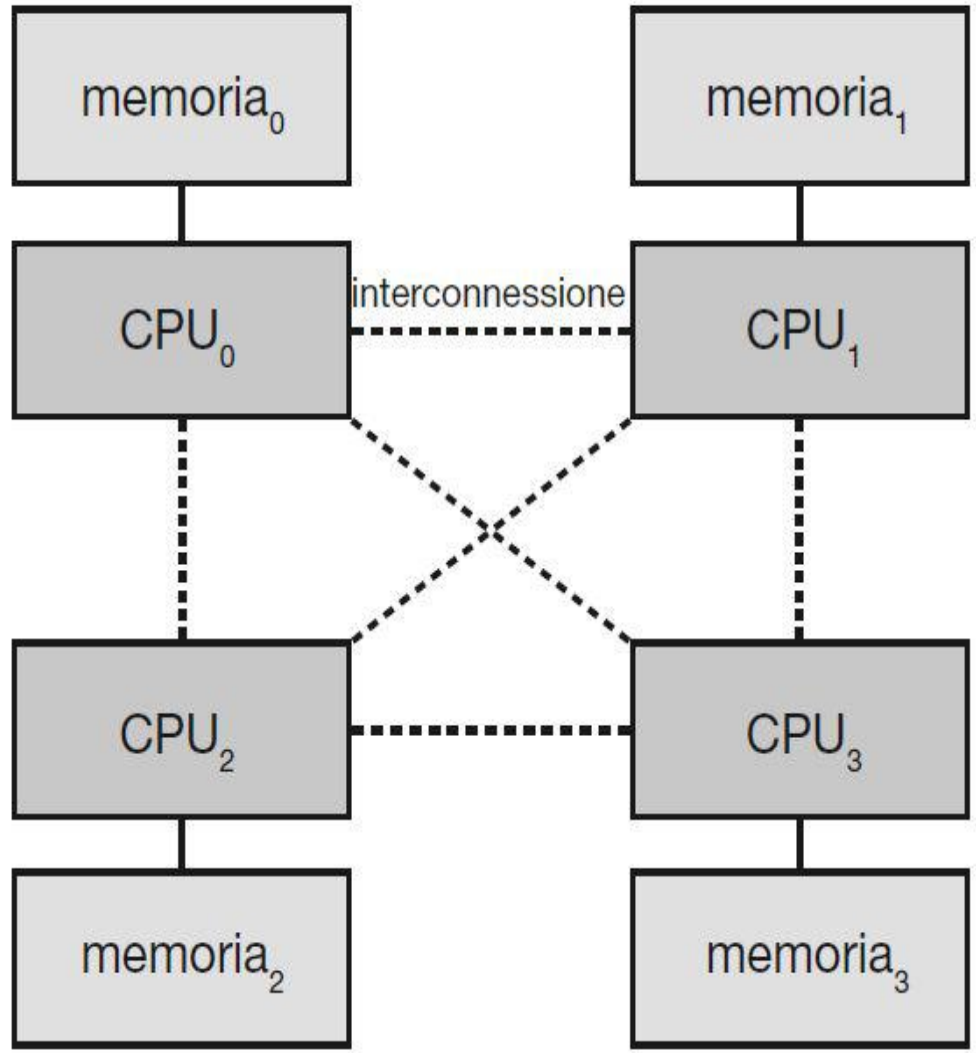
accesso non uniforme alla memoria o **NUMA**

↓

i **sistemi NUMA** possono scalare in modo più efficace con l'aggiunta di più processori

↓

sempre più diffusi nei server e nei sistemi di elaborazione ad alte prestazioni



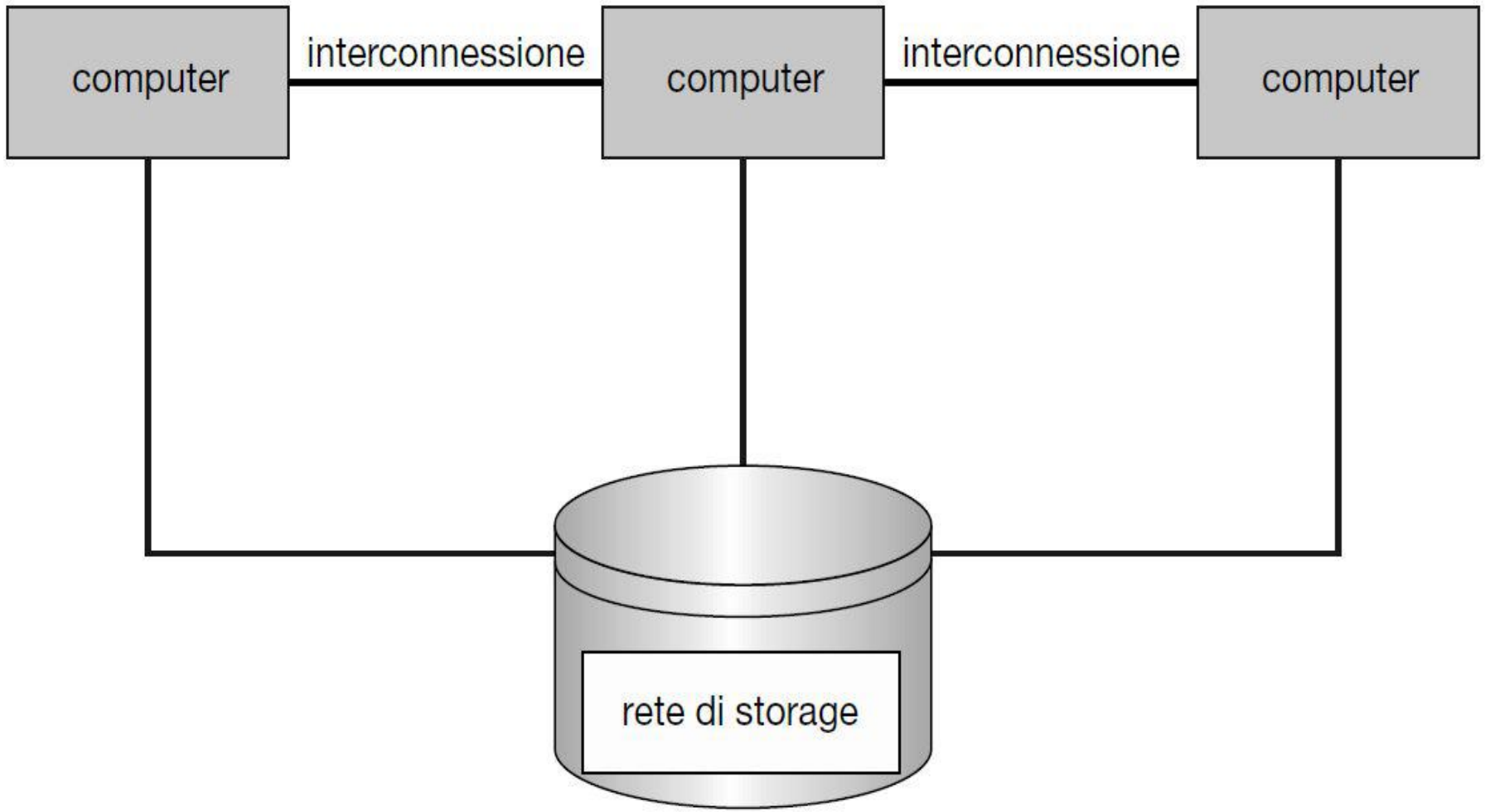
Architettura multiprocessore NUMA.

Cluster di elaboratori (*clustered systems)* o **cluster:** un altro tipo di sistemi multiprocessore, basati sull'uso congiunto di più CPU, ma differiscono dai sistemi multiprocessore perché composti di
due o più calcolatori completi – detti **nodi** – collegati tra loro.

↓

debolmente accoppiati

- Cluster asimmetrici
- Cluster simmetrici
- Cluster paralleli



Struttura generale di un cluster.

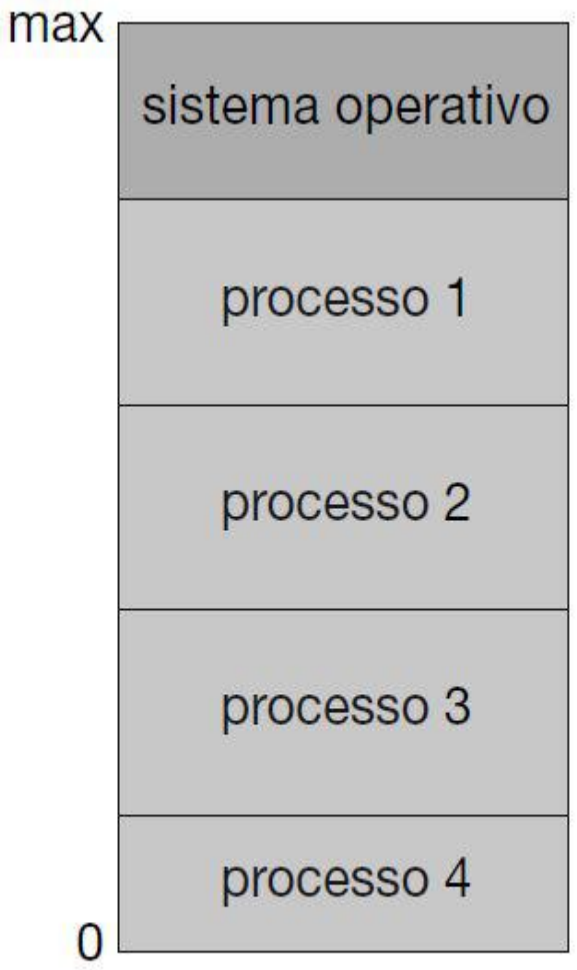
Attività del sistema operativo

1. **Programma di avviamento (***bootstrap program***)**
2. Il **kernel** inizia a offrire servizi al sistema e agli utenti
3. **Interruzioni** ed **eccezioni** (*traps* o *exceptions*)
4. **Chiamata di sistema** (**system call**)
5. **Multiprogrammazione** → *multitasking.*

↓ ↓

file system memoria virtuale

Memoria e multiprogrammazione



Configurazione della memoria per un sistema con multiprogrammazione.

Dual-mode operation

- **modalità utente** e **modalità di sistema** (detta anche *modalità kernel, modalità supervisore, modalità monitor o modalità privilegiata).*

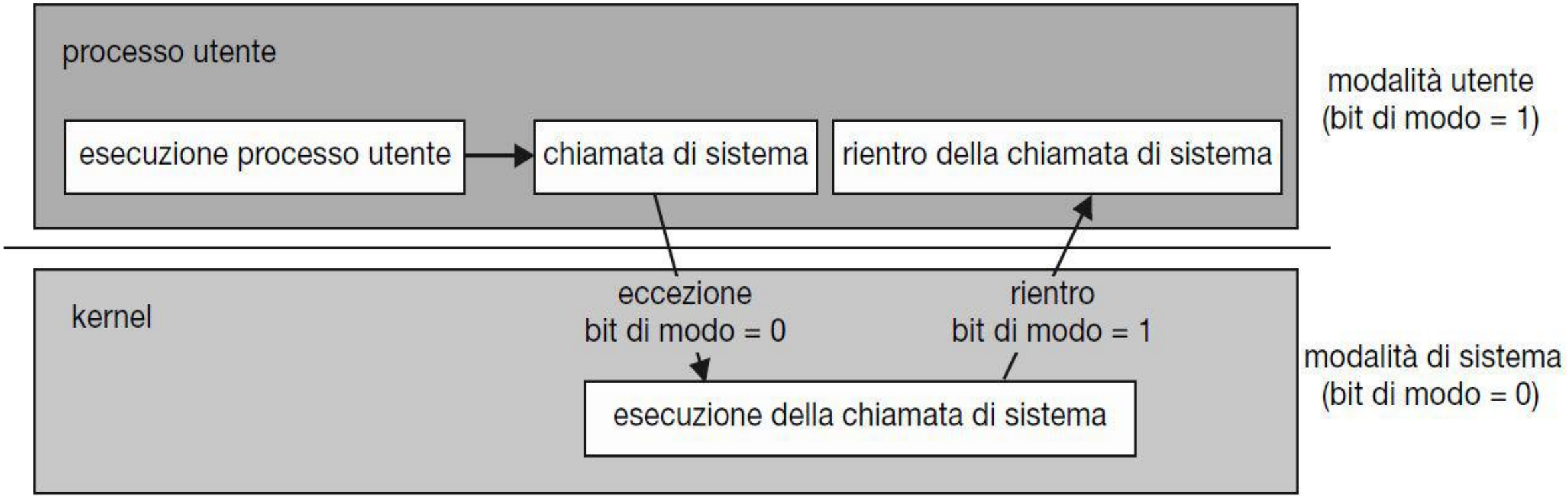
- Per indicare quale sia la modalità attiva, l'architettura della CPU deve essere dotata di un bit, chiamato appunto bit di modalità: **kernel** (0) o **user** (1).

- La duplice modalità di funzionamento (*dual-mode)* consente la protezione del sistema operativo e degli altri utenti dagli errori di un utente.

- Le **chiamate di sistema** (*system call*) sono gli strumenti con cui un programma utente richiede al sistema operativo di compiere operazioni a esso riservate, per conto del programma utente.

- **Timer** → invia un segnale d'interruzione alla CPU a intervalli di tempo specificati e assicura che il sistema operativo mantenga il controllo della CPU.

Dual-mode operation



Transizione da modalità utente a modalità di sistema.

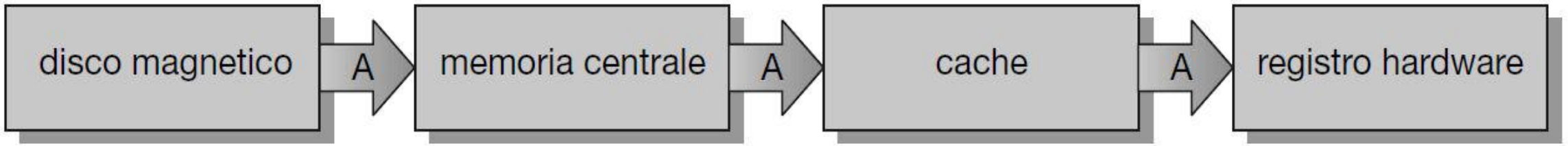
Gestione delle risorse

Forme di memoria

Livello	1	2	3	4	5
Nome	registri	cache	memoria centrale	disco a stato solido	disco magnetico
Dimensione tipica	< 1 KB	< 16 MB	< 64 GB	< 1 TB	< 10 TB
Tecnologia	memoria dedicata con porte multiple (CMOS)	CMOS SRAM (on-chip o off-chip)	CMOS DRAM	memoria flash	disco magnetico
Tempo d'accesso (ns)	0,25 – 0,5	0,5 – 25	80 – 250	25.000-50.000	5.000,000
Ampiezza di banda (MB/s)	20.000 – 100.000	5000 – 10.000	1000 – 5000	500	20 – 150
Gestito da	compilatore	hardware	sistema operativo	sistema operativo	sistema operativo
Supportato da	cache	memoria centrale	disco	disco	disco o nastro

Caratteristiche di varie forme di archiviazione dei dati.

Caricamento delle istruzioni



Migrazione di un intero A da un disco a un registro.

Sicurezza e protezione

Protezione → ciascun meccanismo di controllo dell'accesso alle risorse possedute da un elaboratore, da parte di processi o utenti.

La **protezione** migliora l'affidabilità rilevando errori nascosti alle interfacce tra i componenti dei sottosistemi.

È compito della **sicurezza** difendere il sistema da attacchi provenienti dall'interno o dall'esterno.

identificatori utente (*user ID*) → identificano univocamente l'utente

Virtualizzazione

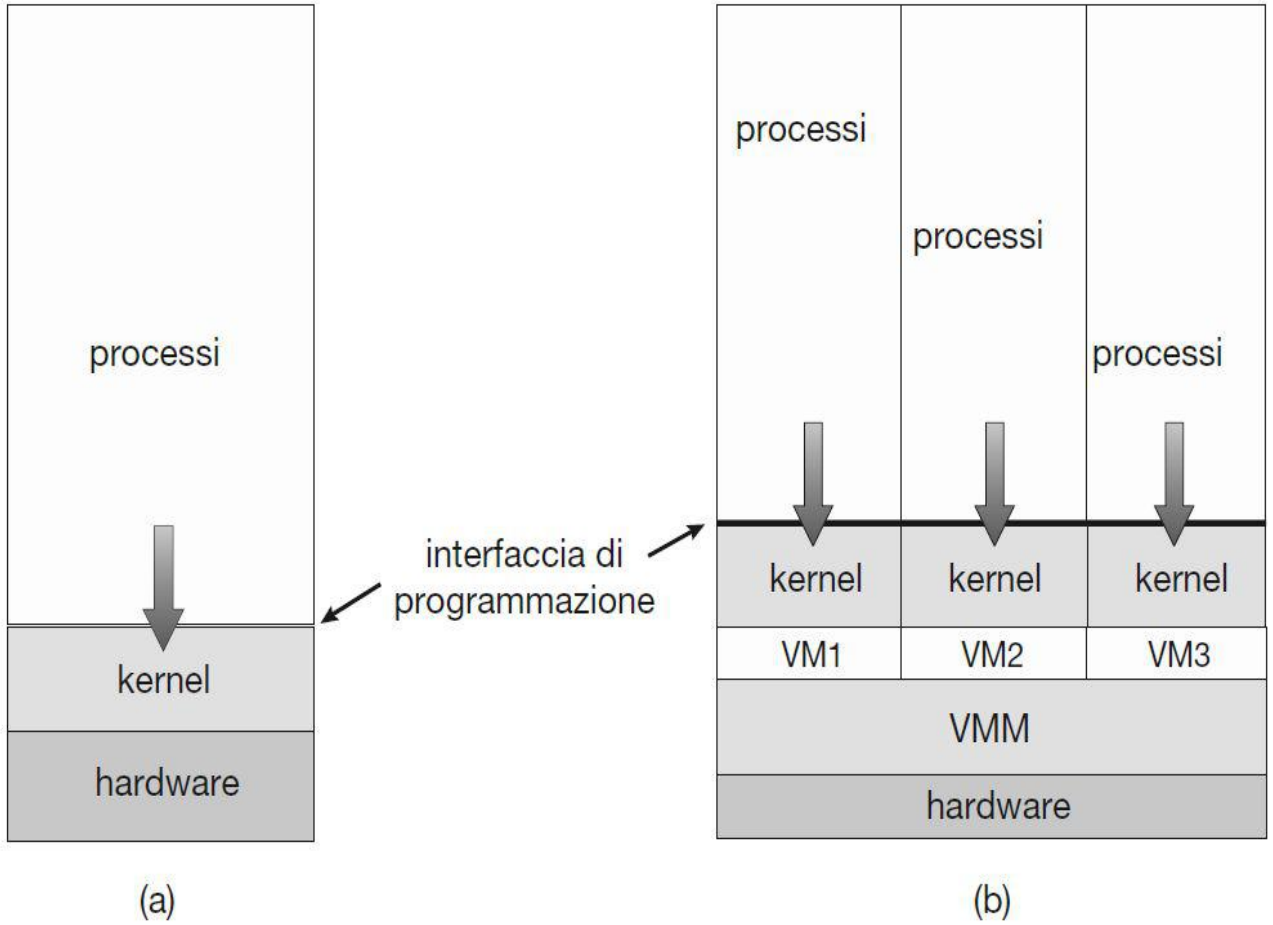
Virtualizzazione → tecnica che permette di astrarre l'hardware di un singolo computer in diversi ambienti di esecuzione, creando così l'illusione che ogni distinto ambiente sia in esecuzione sul suo proprio computer.

Permette ai sistemi operativi di funzionare come applicazioni all'interno di altri sistemi operativi

- Con la **virtualizzazione** un sistema operativo compilato per una particolare architettura viene eseguito all'interno di un altro sistema operativo progettato per la stessa CPU.

- **Macchina virtuale (VM)** e **gestore della macchia virtuale**

Virtualizzazione



Un computer che ha in esecuzione (a) un singolo sistema operativo e (b) tre macchine virtuali.

Sistemi distribuiti

Sistema distribuito → un insieme di elaboratori fisicamente separati e con caratteristiche spesso eterogenee, interconnessi da una rete per consentire agli utenti l'accesso alle varie risorse dei singoli sistemi.

Rete → un canale di comunicazione tra due o più sistemi.

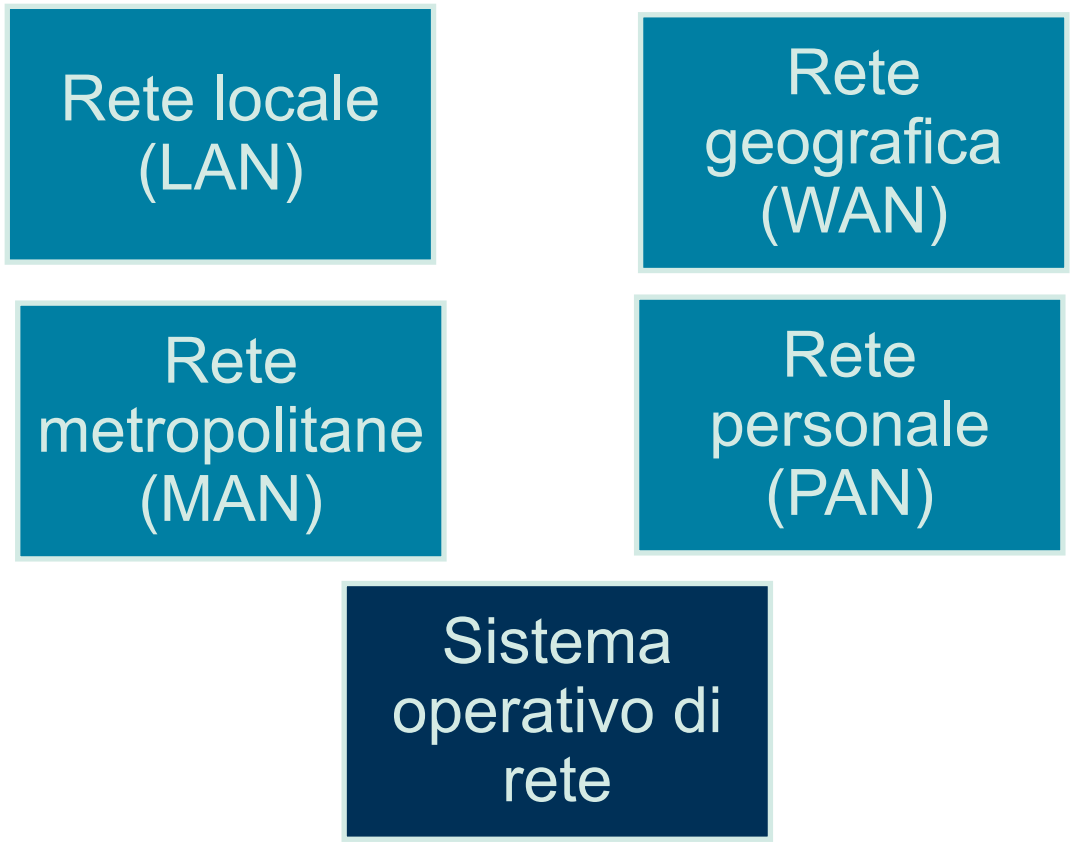
↓

protocollo

↓

TCP/IP

Sistemi distribuiti - Reti



Strutture dati del kernel - Liste

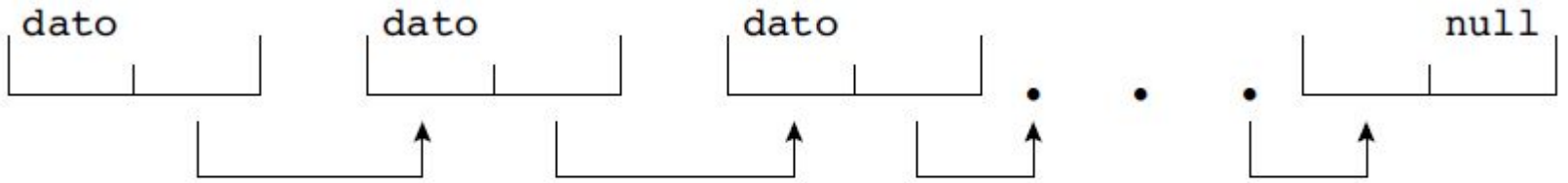


Figura 1.17 Lista semplicemente concatenata.

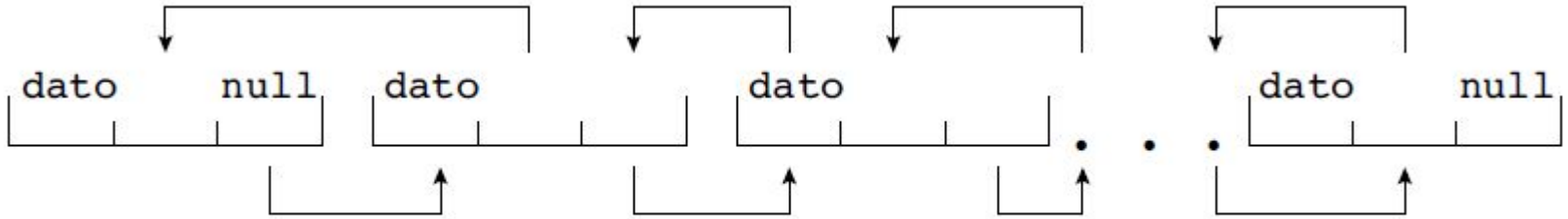
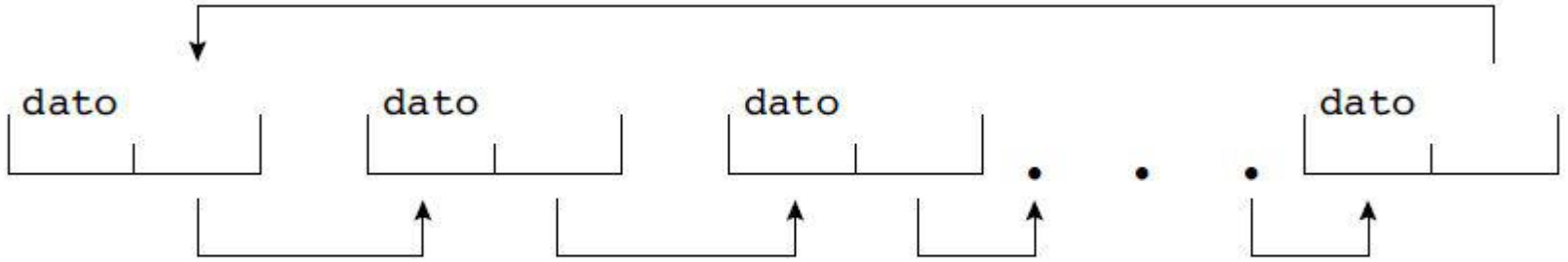


Figura 1.18 Lista doppiamente concatenata.



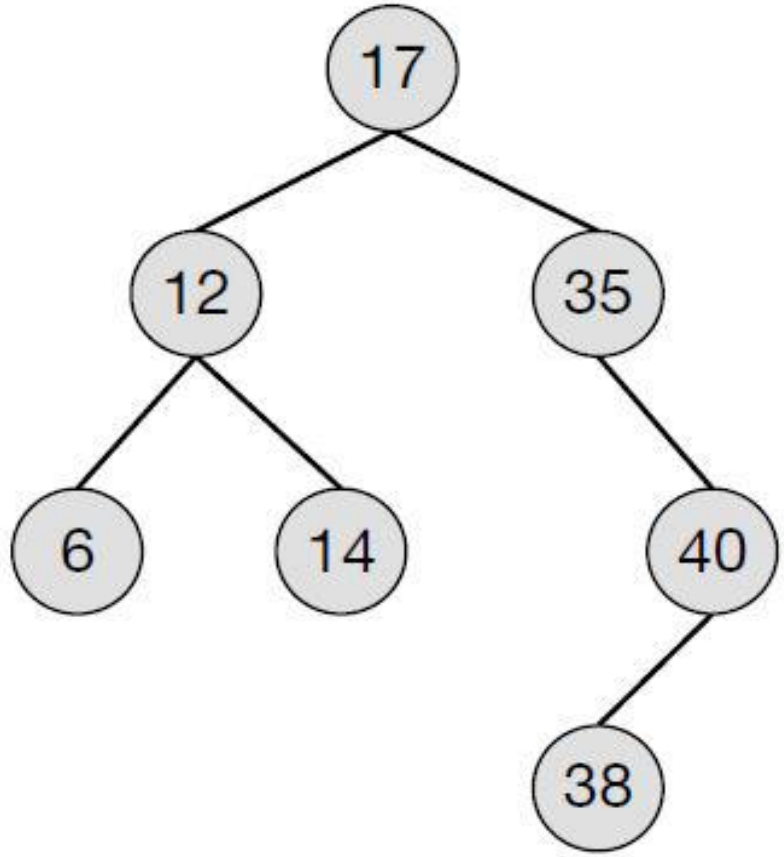
Lista circolare.

Alberi

Un **albero** è una struttura dati utilizzabile per rappresentare i dati in maniera gerarchica.

- Generico albero
- Albero binario
- Albero binario di ricerca
- Albero di ricerca binario bilanciato

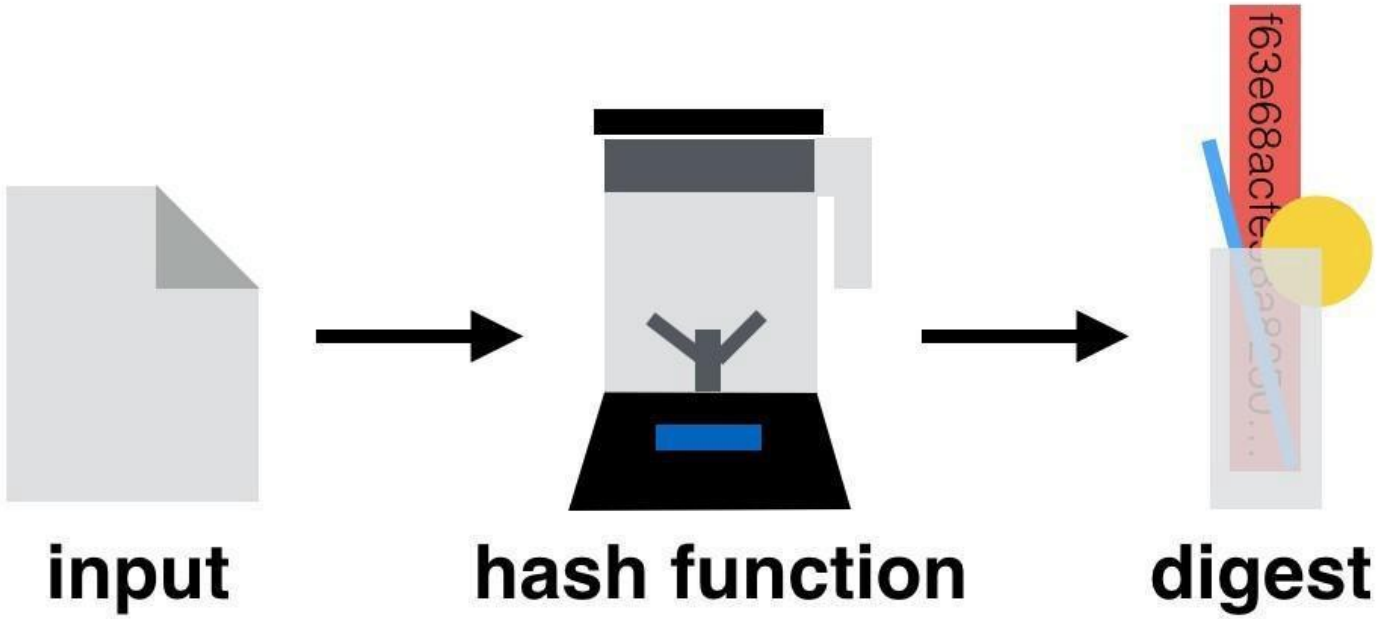
Alberi - esempio



Albero binario di ricerca.

Funzioni hash

Una **funzione hash** riceve dati in input, realizza operazioni numeriche sui dati e restituisce un valore numerico.



Mappe hash

Una funzione hash può essere utilizzata per creare una **tabella hash** (o mappa hash) che associ (o mappi) coppie [chiave:valore].

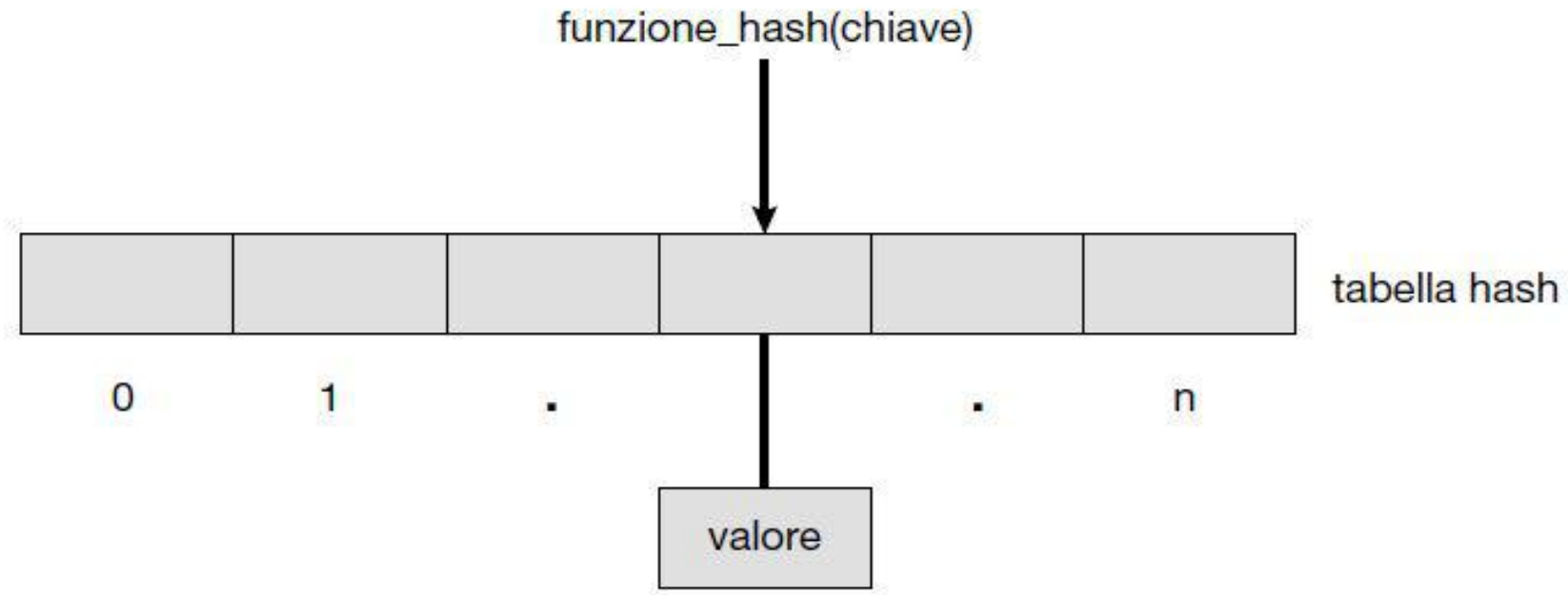


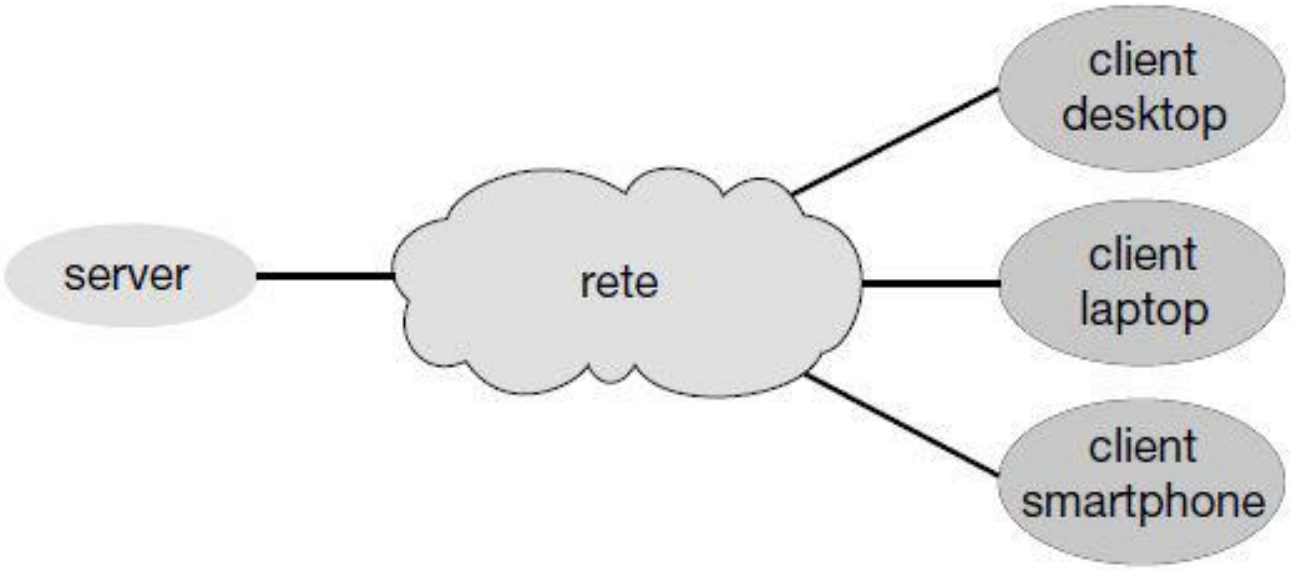
Tabella hash.

Elaborazione client-server

Un’architettura di rete contemporanea realizza un sistema in cui alcuni **server** soddisfano le richieste dei **sistemi client**

↓

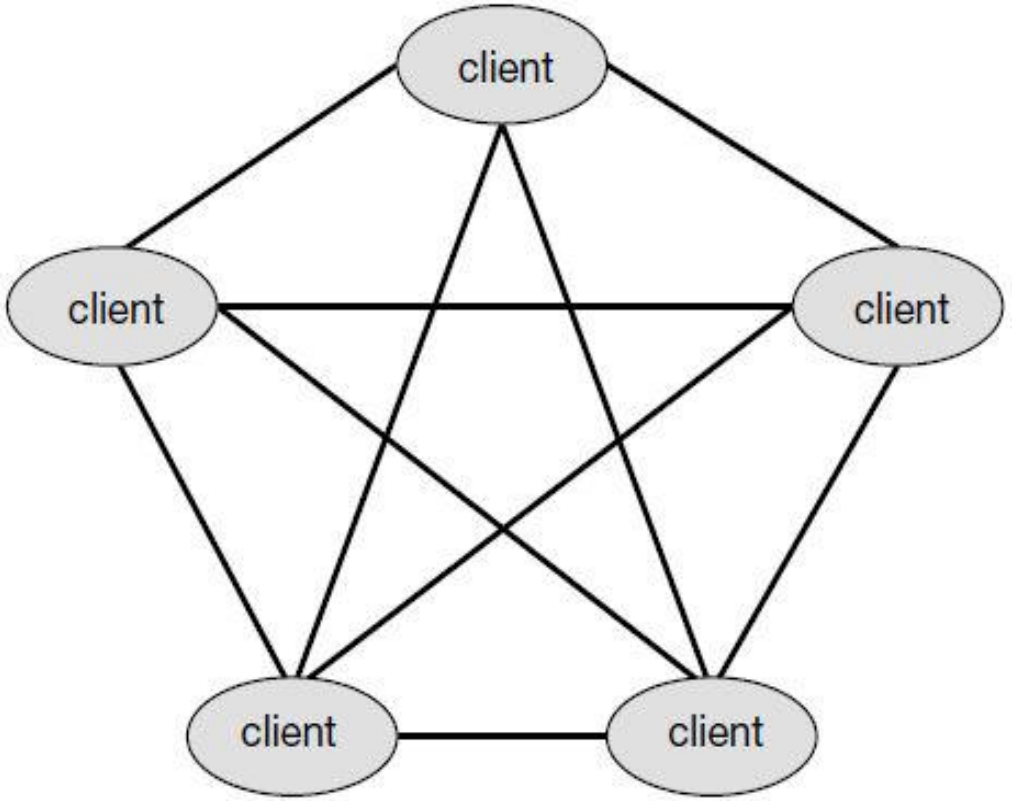
sistemi client-server



Struttura generale di un sistema client-server.

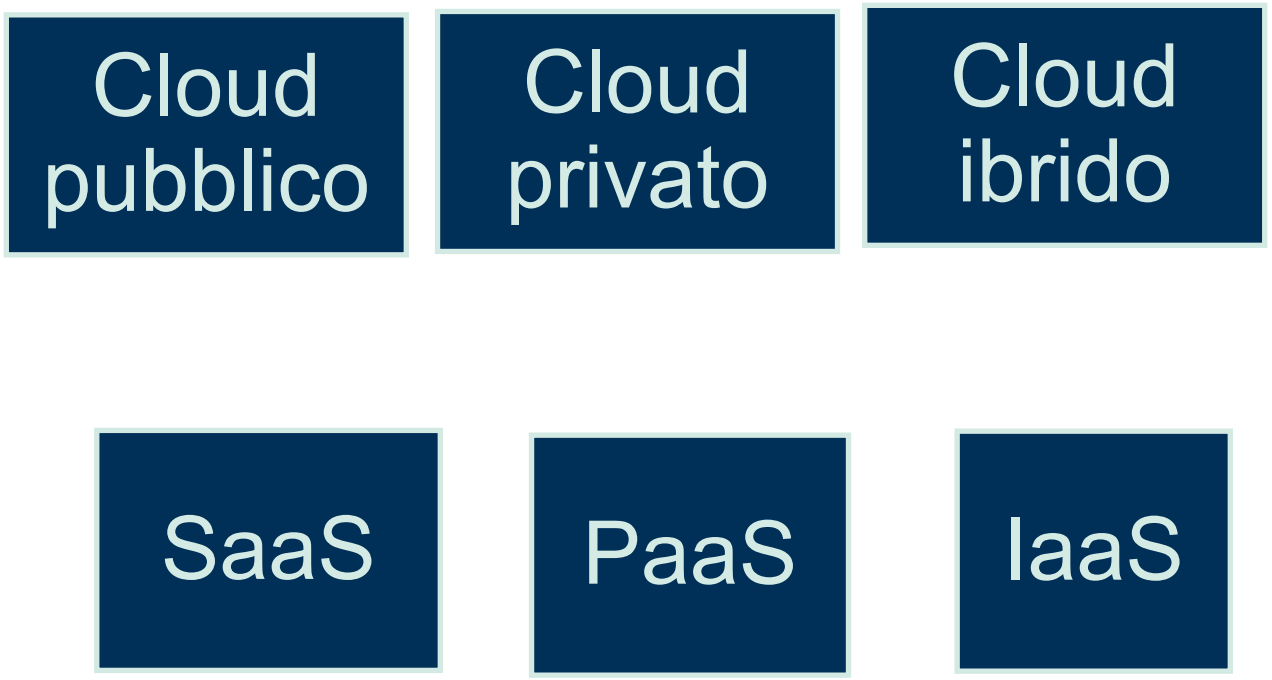
Elaborazione peer-to-peer

peer-to-peer (**P2P**) → cade la distinzione tra client e server; tutti i nodi all'interno del sistema sono su un piano di parità, e ciascuno può fungere ora da client, ora da server, a seconda che stia richiedendo o fornendo un servizio.

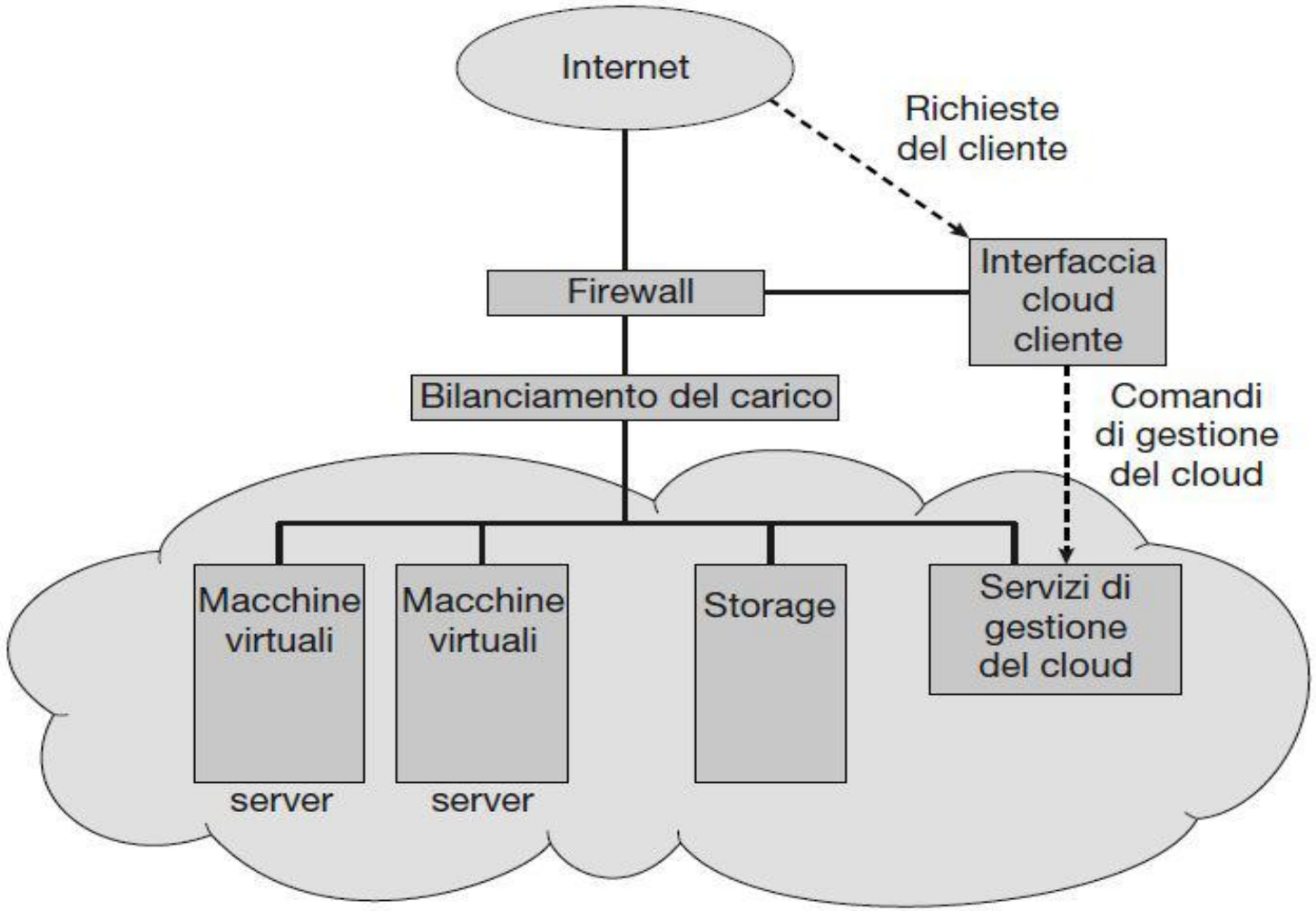


Sistema peer-to-peer senza servizi centralizzati.

Cloud computing



Cloud computing



Cloud computing.

Sistemi operativi liberi e open-source

Sia i **sistemi operativi liberi** sia i **sistemi operativi open-source** sono disponibili in formato sorgente anziché come codice binario compilato.

Software libero → non solo rende il codice sorgente disponibile, *ma è* anche dotato di una licenza che consente l'uso, la ridistribuzione e la modifica senza costi.

Il **software open-source** non offre necessariamente tale licenza

- **GNU/Linux**, **FreeBSD** e **Solaris** sono esempi diffusi di sistemi operativi open-source
- Microsoft Windows è un **software proprietario**

Servizi di un sistema operativo

Un sistema operativo offre un ambiente in cui eseguire i programmi e fornire i seguenti servizi.

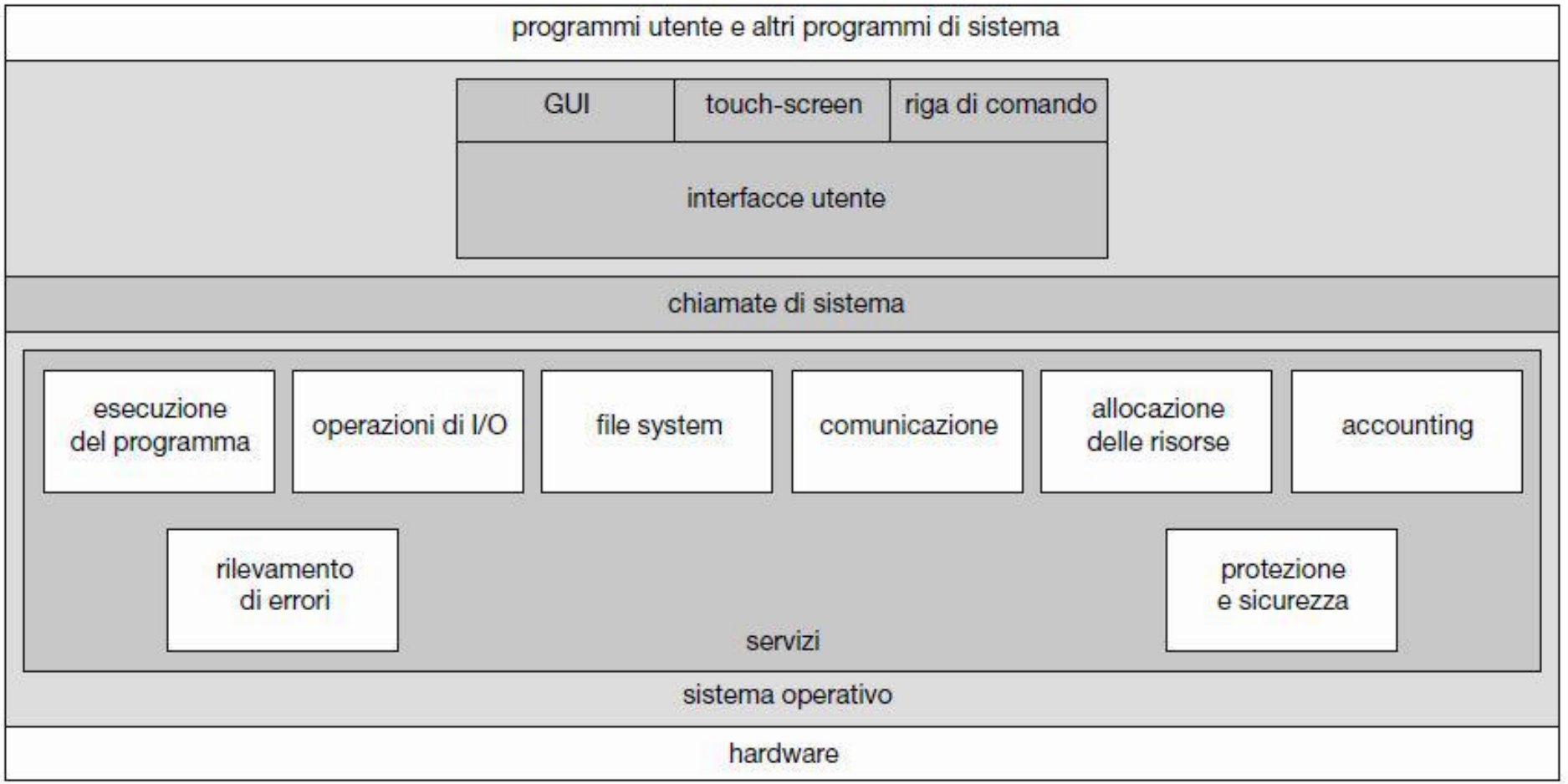
Servizi di un sistema operativo

Allocazione delle risorse

Logging

Protezione e sicurezza

Servizi di un sistema operativo



Panoramica dei servizi del sistema operativo.

Interfaccia con l’utente del sistema operativo

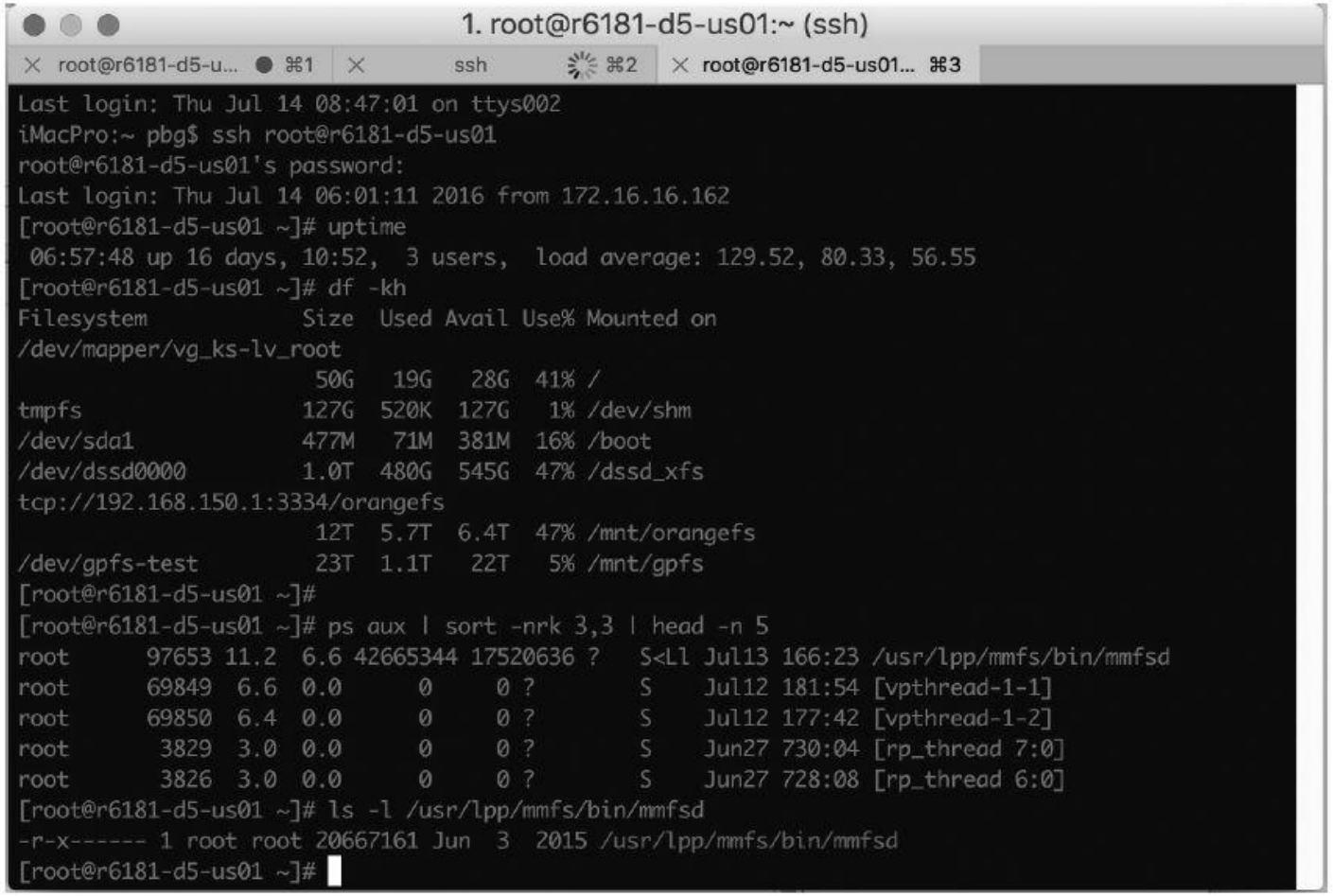
1. interfaccia a riga di comando o

   interprete dei comandi

2. Interfaccia **touch-screen**

3. Interfaccia grafica con l’utente o **GUI**

Interprete dei comandi



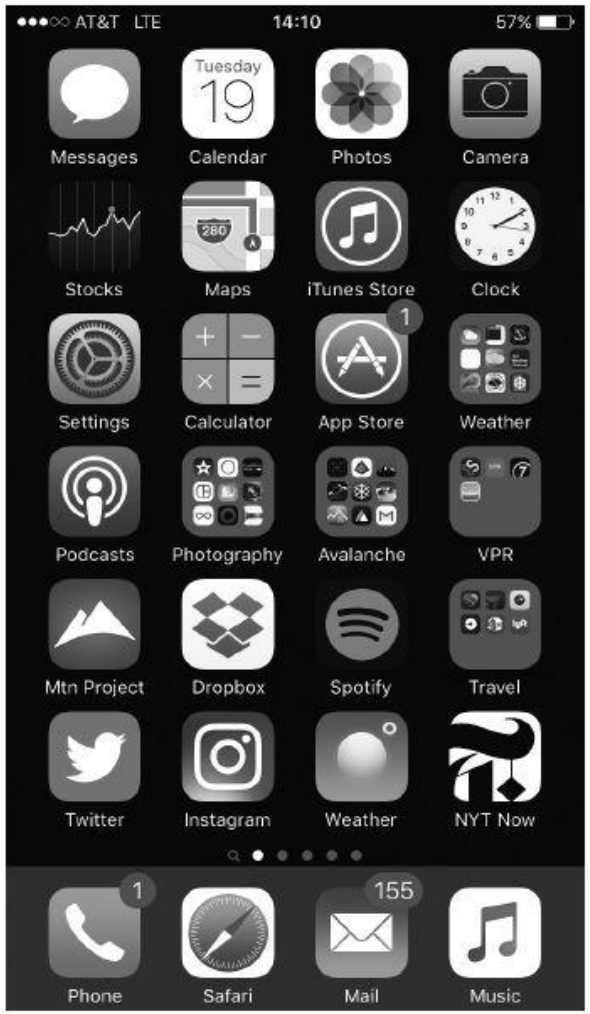
La shell bash, l'interprete dei comandi utilizzato in macOS.

Interprete dei comandi

↓

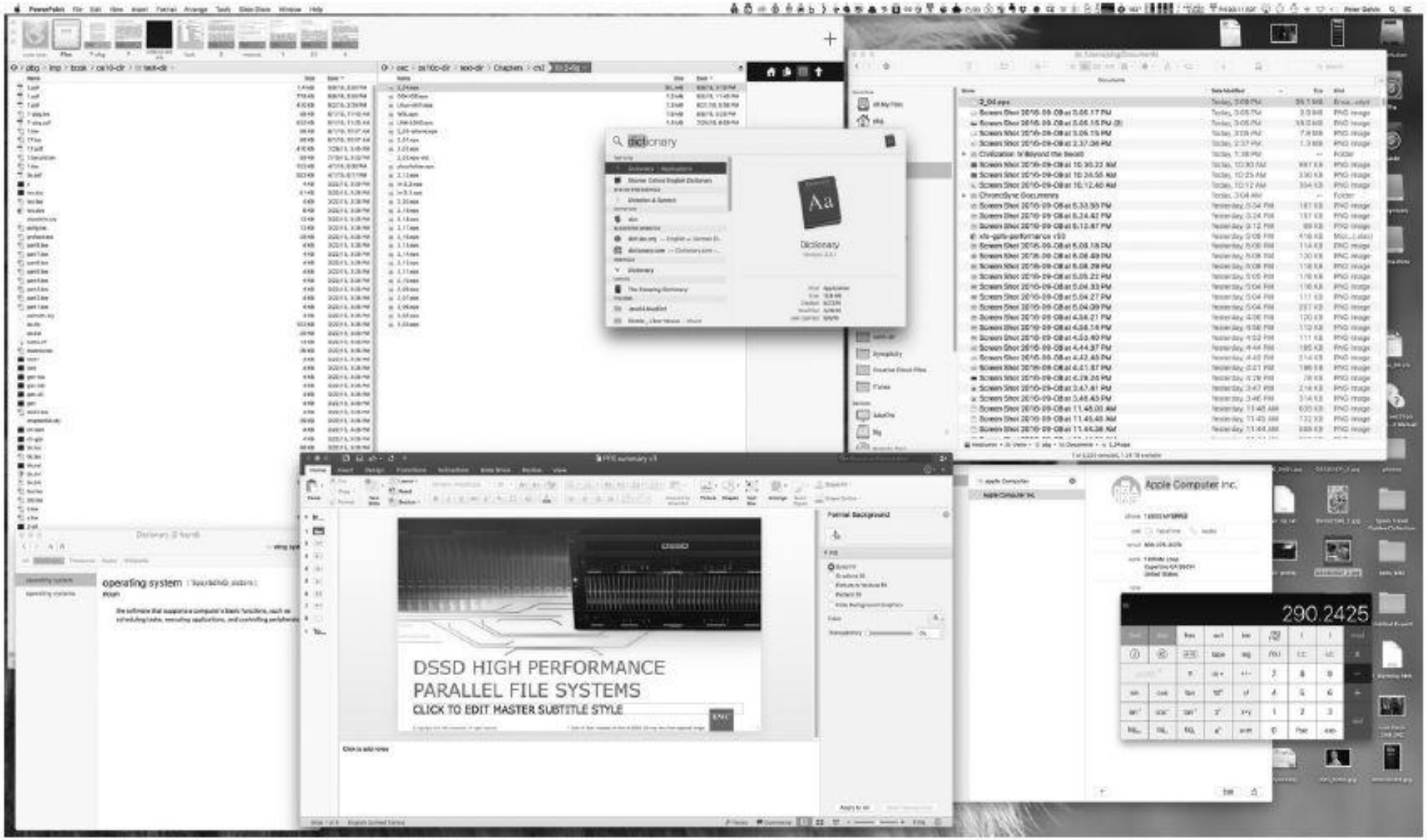
shell

Interfaccia touch-screen



Il touch-screen di un iPhone.

GUI



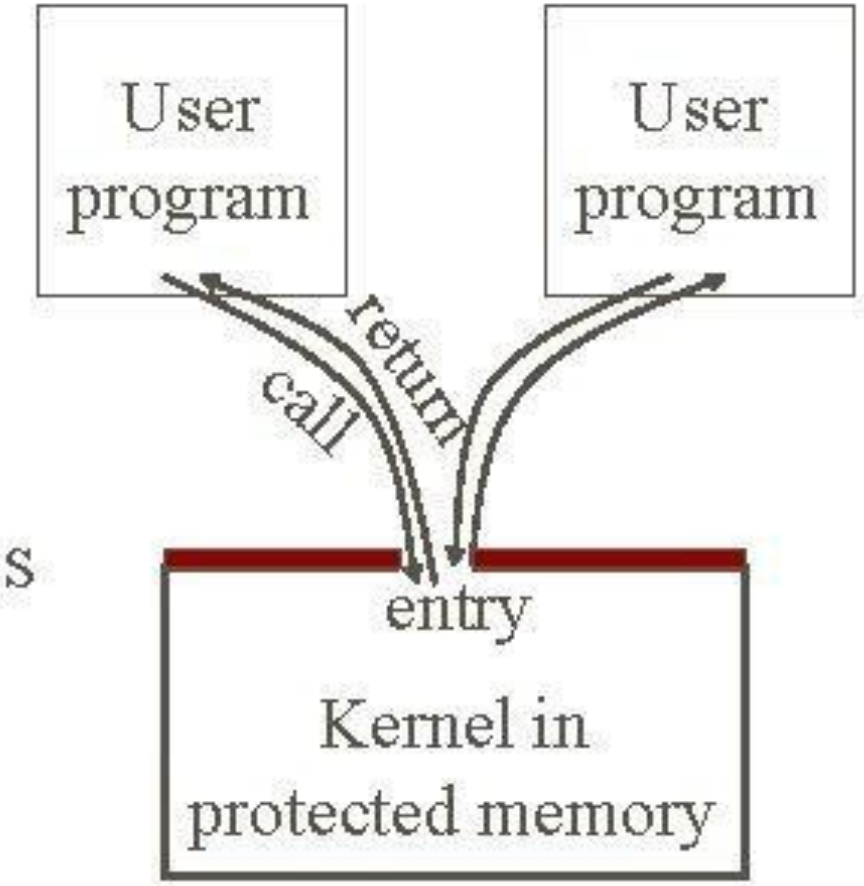
Interfaccia grafica di macOS.

System calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are **Win32 API** for Windows, **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and **Java API** for the Java virtual machine (JVM)

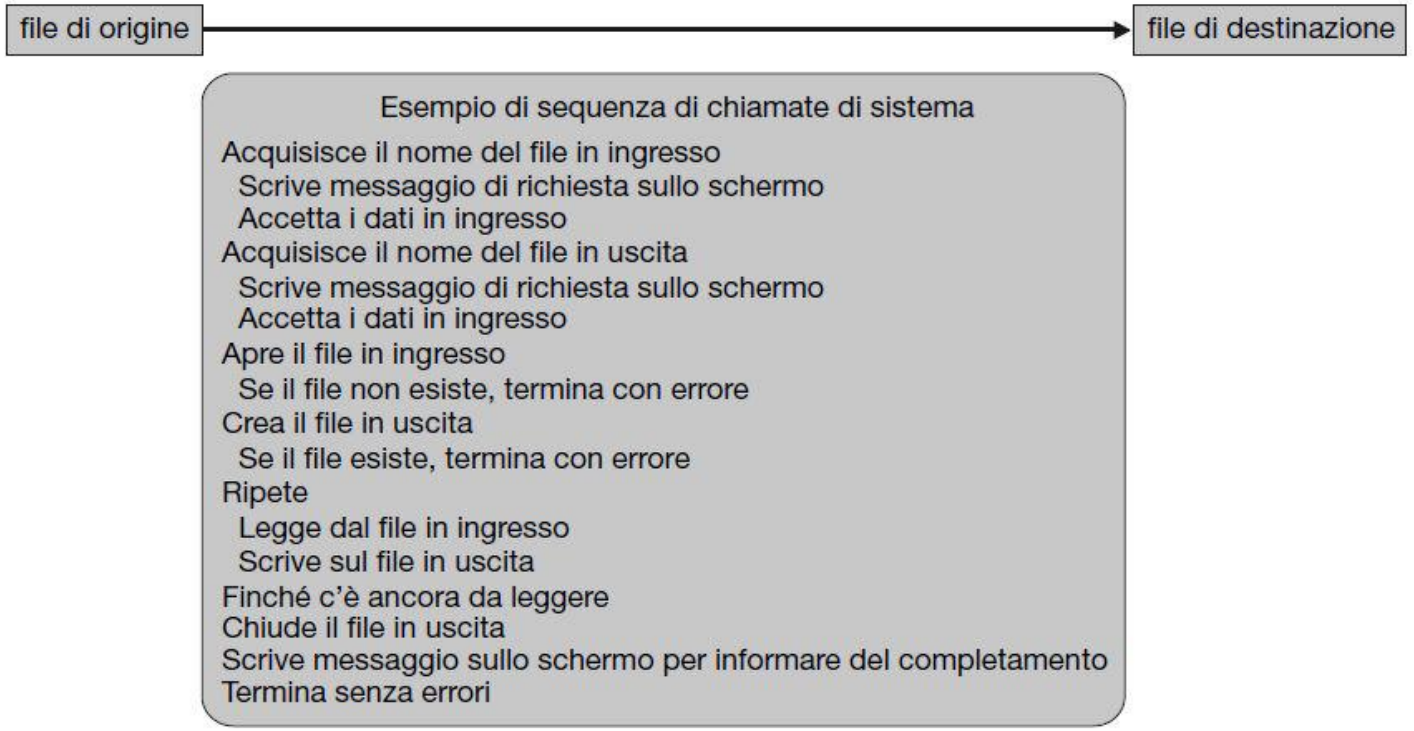
System call mechanism

- User code can be arbitrary
- User code cannot modify kernel memory
- Makes a system call with parameters
- The call mechanism switches code to kernel mode
- Execute system call
- Return with results



Esempio system call

Le **chiamate di sistema** (*system call*) costituiscono un'interfaccia per i servizi resi disponibili dal sistema operativo.



Esempio d'uso delle chiamate di sistema.

System call - implementazione

- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface **hidden from programmer by API**
    - Managed by **run-time support library** (set of functions built into libraries included with compiler)

API - Interfaccia per la programmazione di applicazioni

API: Specifica un insieme di funzioni a disposizione del programmatore e dettaglia i parametri necessari all'invocazione di queste funzioni, insieme ai valori restituiti.

ESEMPIO DI API STANDARD

Come esempio di API standard consideriamo la funzione `read()` disponibile in Unix e Linux. L'API per questa funzione si può ottenere digitando

```
man read
```

da riga di comando. Una descrizione di questa API è la seguente:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

valore restituito — nome della funzione — parametri

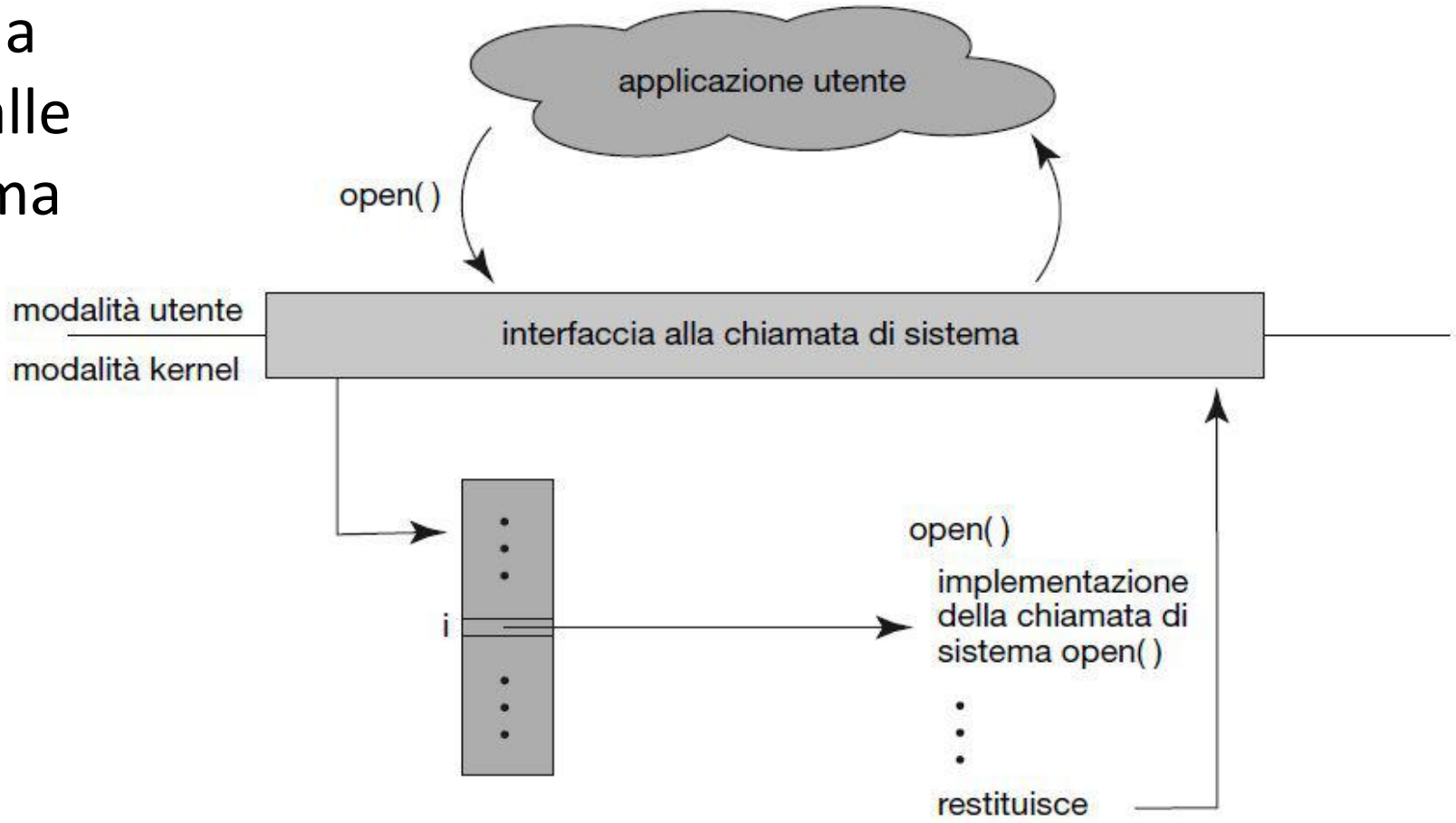
Un programma che utilizza la `read()` deve includere il file `unistd.h` che, tra le altre cose, definisce i tipi di dato `ssize_t` e `size_t`. I parametri passati alla `read()` sono i seguenti:

- `int fd` — il descrittore del file da leggere
- `void *buf` — un buffer nel quale vengono messi i dati letti
- `size_t count` — il massimo numero di byte da leggere e inserire nel buffer

Quando una `read()` è completata con successo viene restituito il numero di byte letti. La `read()` restituisce 0 in caso di fine del file e –1 quando si è verificato un errore.

API, system call e SO

Le relazioni fra una API, l'interfaccia alle chiamate di sistema e il sistema operativo



Gestione della chiamata di sistema `open( )` invocata da un'applicazione utente.

System call – passaggio dei parametri

Per passare parametri al sistema operativo si usano tre metodi generali:

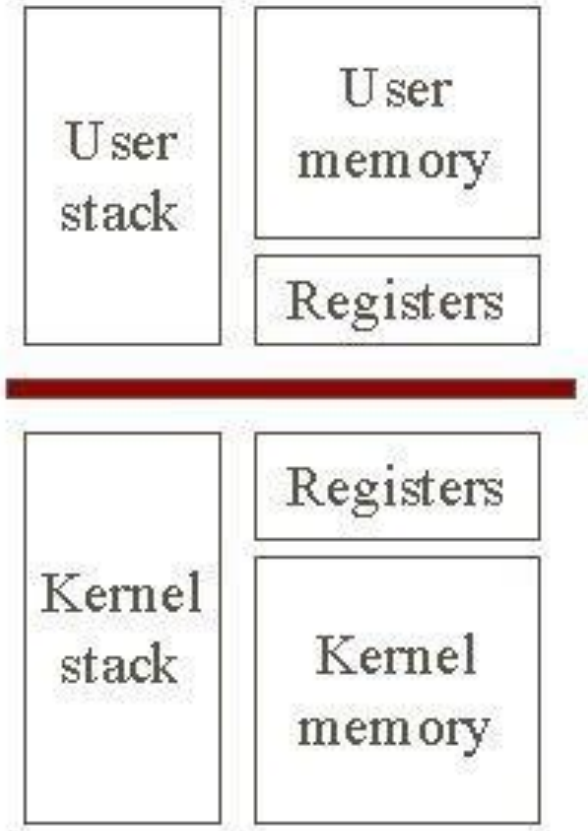
1. in *registri*

2. in un *blocco* o *tabella di memoria*

3. nello *stack* da cui sono prelevati (*pop*) dal sistema operativo

Passaggio dei parametri in registri

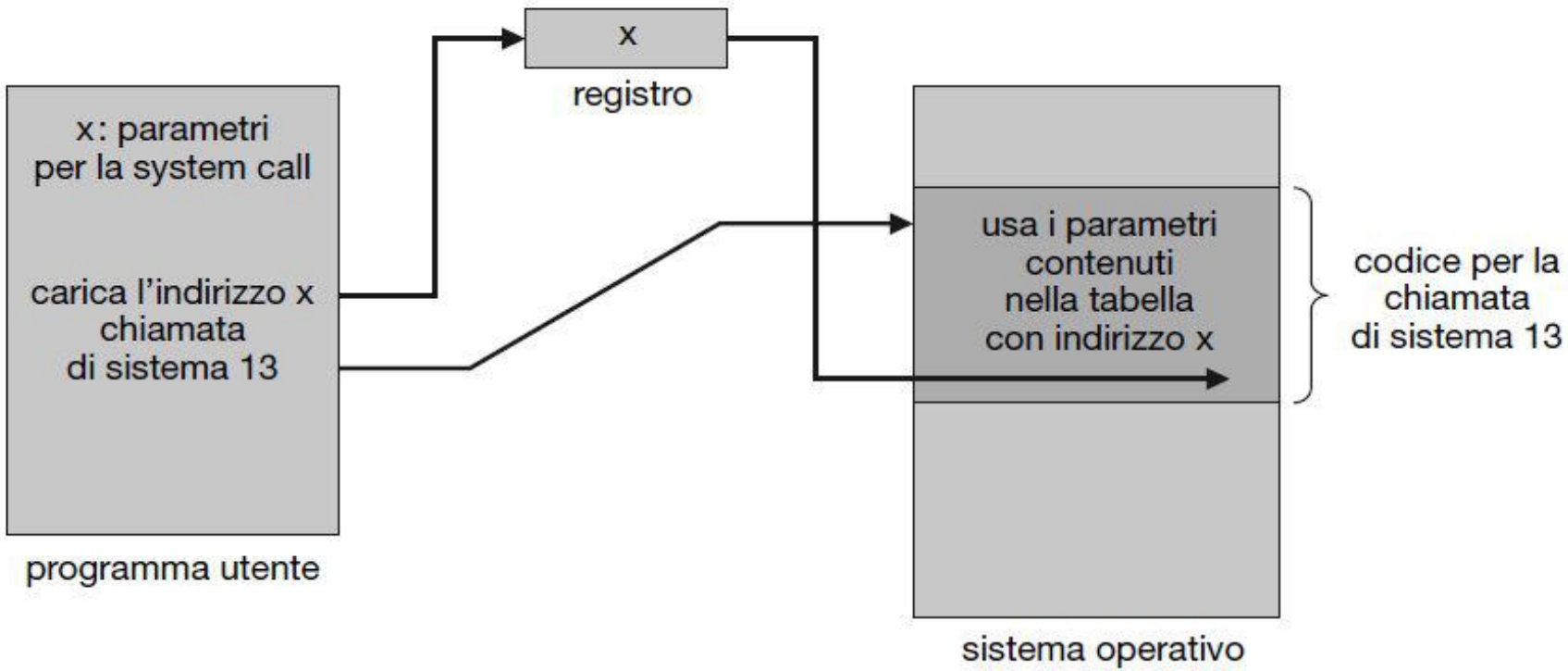
- È il metodo più semplice e veloce
- In alcuni casi potrebbero esserci più parametri che registri



Passaggio dei parametri in forma di tabella

Parameters stored in a **block**, or table, in memory, and **address** of block passed as a parameter in a **register**

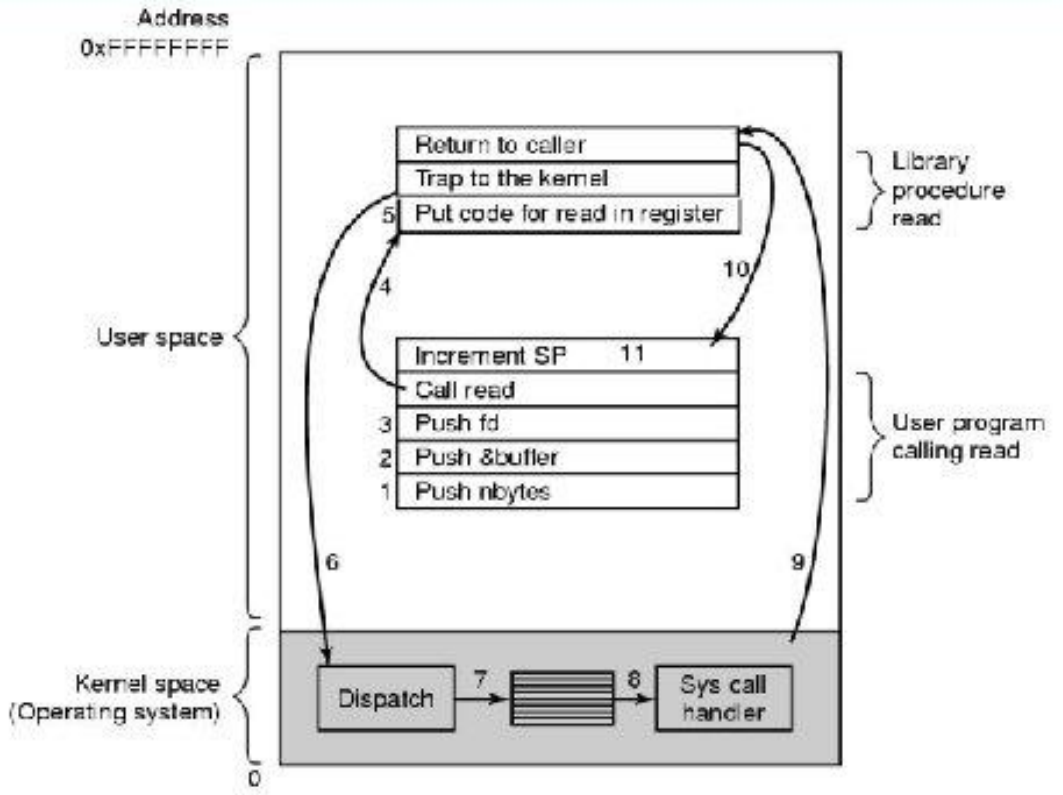
- This approach taken by Linux and Solaris



Passaggio di parametri in forma di tabella.

Passaggio dei parametri nello stack

- Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
- Block and stack methods **do not limit** the number or length of parameters being passed



The 11 steps in making the system call read(fd, buffer, nbytes)

Categorie di chiamate di sistema

- Controllo dei processi
- Gestione dei file
- Gestione dei dispositivi
- Gestione delle informazioni, comunicazioni e protezione

Categorie di chiamate di sistema

Controllo dei processi

- creazione e arresto di un processo
- caricamento, esecuzione
- terminazione normale e anormale
- esame e impostazione degli attributi di un processo
- attesa per il tempo indicato
- attesa e segnalazione di un evento
- assegnazione e rilascio di memoria

Categorie di chiamate di sistema

Gestione dei file

- creazione e cancellazione di file
- apertura, chiusura
- lettura, scrittura, posizionamento
- esame e impostazione degli attributi di un file

Categorie di chiamate di sistema

Gestione dei dispositivi

- richiesta e rilascio di un dispositivo
- lettura, scrittura, posizionamento
- esame e impostazione degli attributi di un dispositivo
- inserimento logico ed esclusione logica di un dispositivo

Categorie di chiamate di sistema

Gestione delle informazioni, comunicazioni e protezione

- Gestione delle info…
  - esame e impostazione dell'ora e della data
  - esame e impostazione dei dati del sistema
  - esame e impostazione degli attributi dei processi, file e dispositivi
- Comunicazione
  - creazione e chiusura
  - invio e ricezione di messaggi
  - informazioni sullo stato di un trasferimento
  - inserimento ed esclusione di dispositivi remoti
- Protezione
  - visualizzazione dei permessi di un file
  - impostazione dei permessi di un file

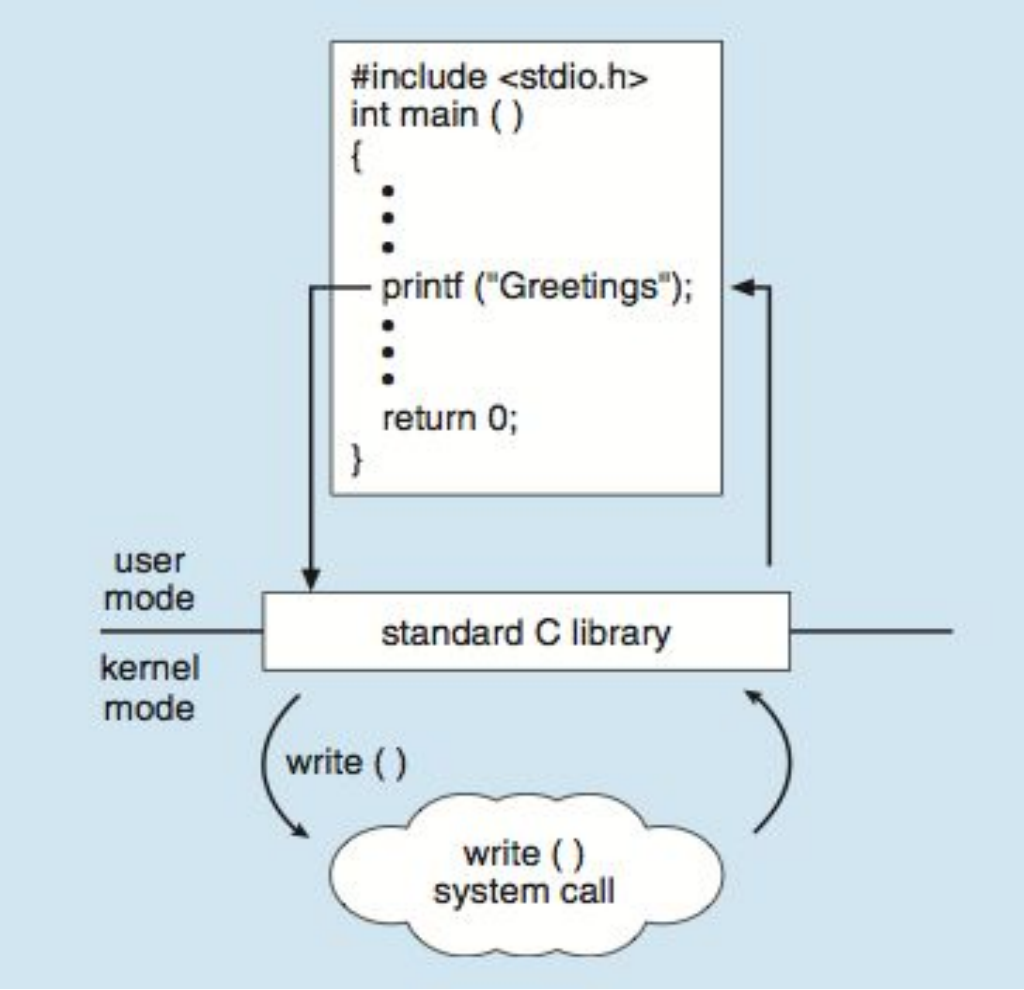
Esempi di chiamate di sistema

ESEMPIO DI CHIAMATE DI SISTEMA DI WINDOWS E UNIX

	Windows	UNIX
Controllo dei processi	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
Gestione dei file	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Gestione dei dispositivi	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Gestione delle informazioni	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Comunicazione	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protezione	`SetFileSecurity()`	`chmod()`
	`InitializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

Standard C Library Example

- C program invoking `printf()` library call, which calls `write()` system call



Sistema monoprogrammato

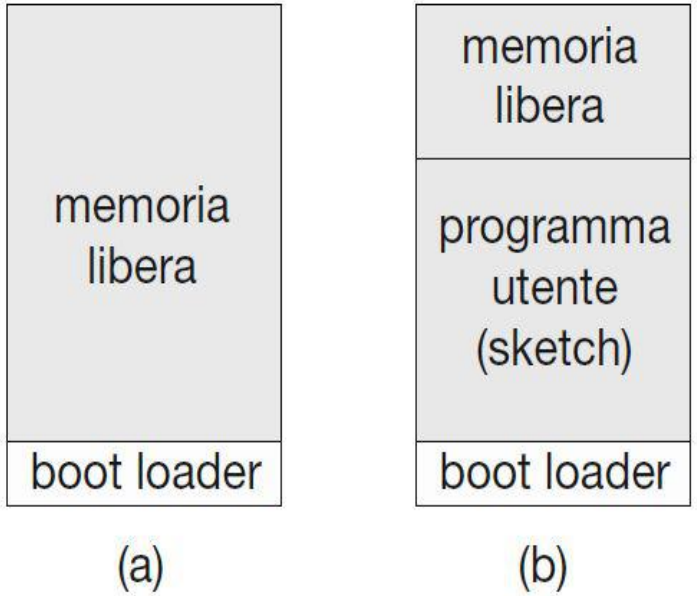
Arduino è una semplice piattaforma hardware composta da un microcontrollore e da sensori di ingresso che rispondono a diversi eventi

Arduino è un esempio di un sistema monoprogrammato

Sistema monoprogrammato

La creazione di un programma per Arduino prevede:

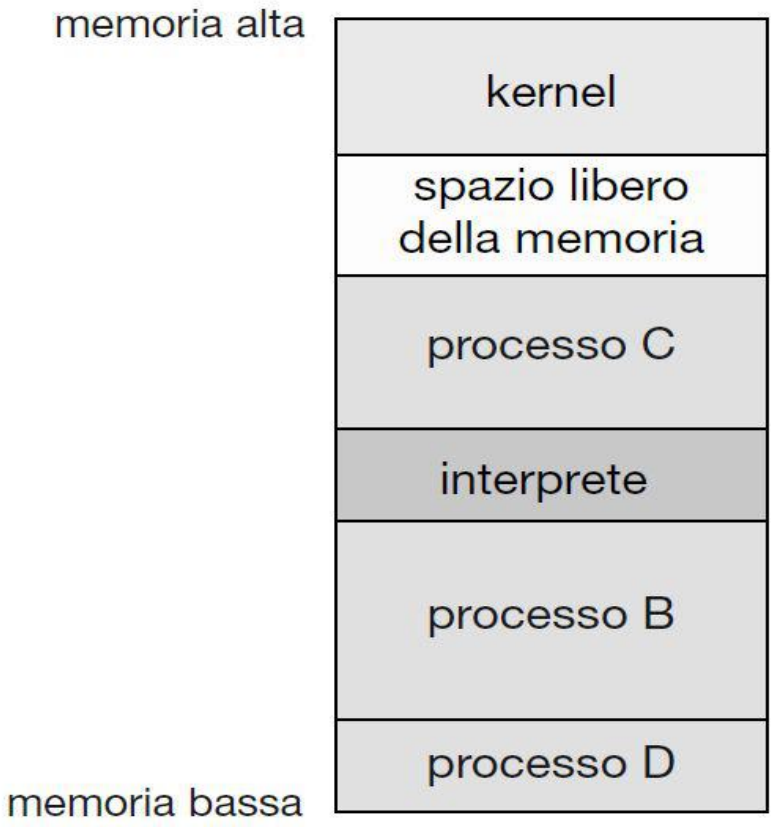
1. la scrittura del programma su un PC
2. Il caricamento del programma compilato (noto come *sketch)* dal PC alla memoria flash di Arduino tramite una connessione USB



Esecuzione in Arduino. (a) All'avviamento del sistema. (b) Durante l'esecuzione di un programma.

Sistema multitasking

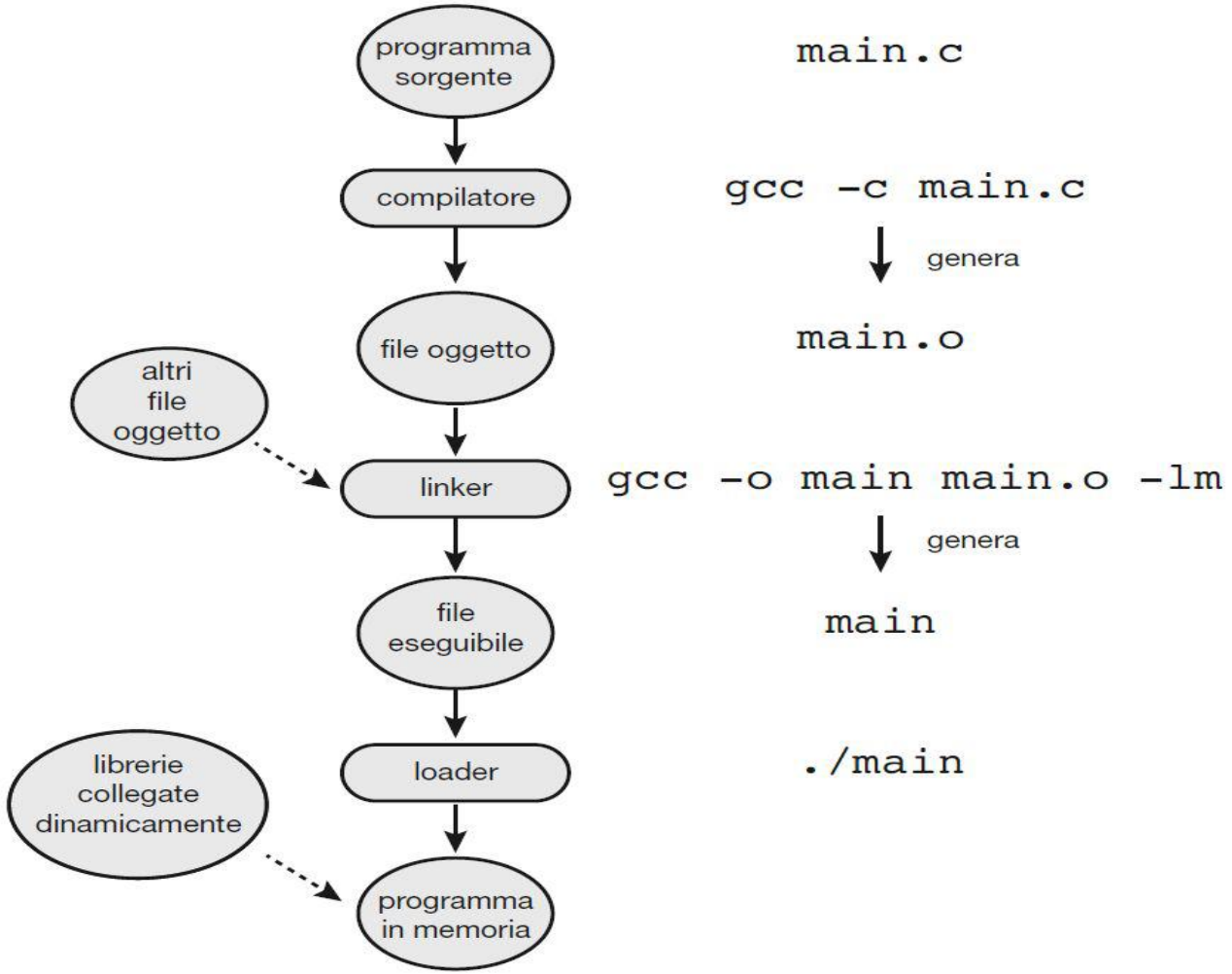
FreeBSD (derivato da UNIX Berkeley)
è un esempio di
sistema
multitasking



Esecuzione di più programmi nel sistema operativo FreeBSD.

Servizi di sistema/utilità di sistema

Linker e loader



Il ruolo di linker e loader.

Perchè le applicazioni dipendono dal sistema operativo

- Fondamentalmente le applicazioni compilate su un sistema operativo *non sono eseguibili* su altri sistemi operativi.

- Ogni sistema operativo fornisce *un insieme univoco di chiamate di sistema*.

Esecuzione su più sistemi operativi

Tre modi per consentire a un'applicazione di essere resa disponibile per l'esecuzione su più sistemi operativi:

1. Può essere scritta in un linguaggio interpretato che ha un interprete disponibile per più sistemi operativi (per esempio Python)

2. Può essere scritta in un linguaggio che utilizza una macchina virtuale contenente l'applicazione in esecuzione (per esempio Java)

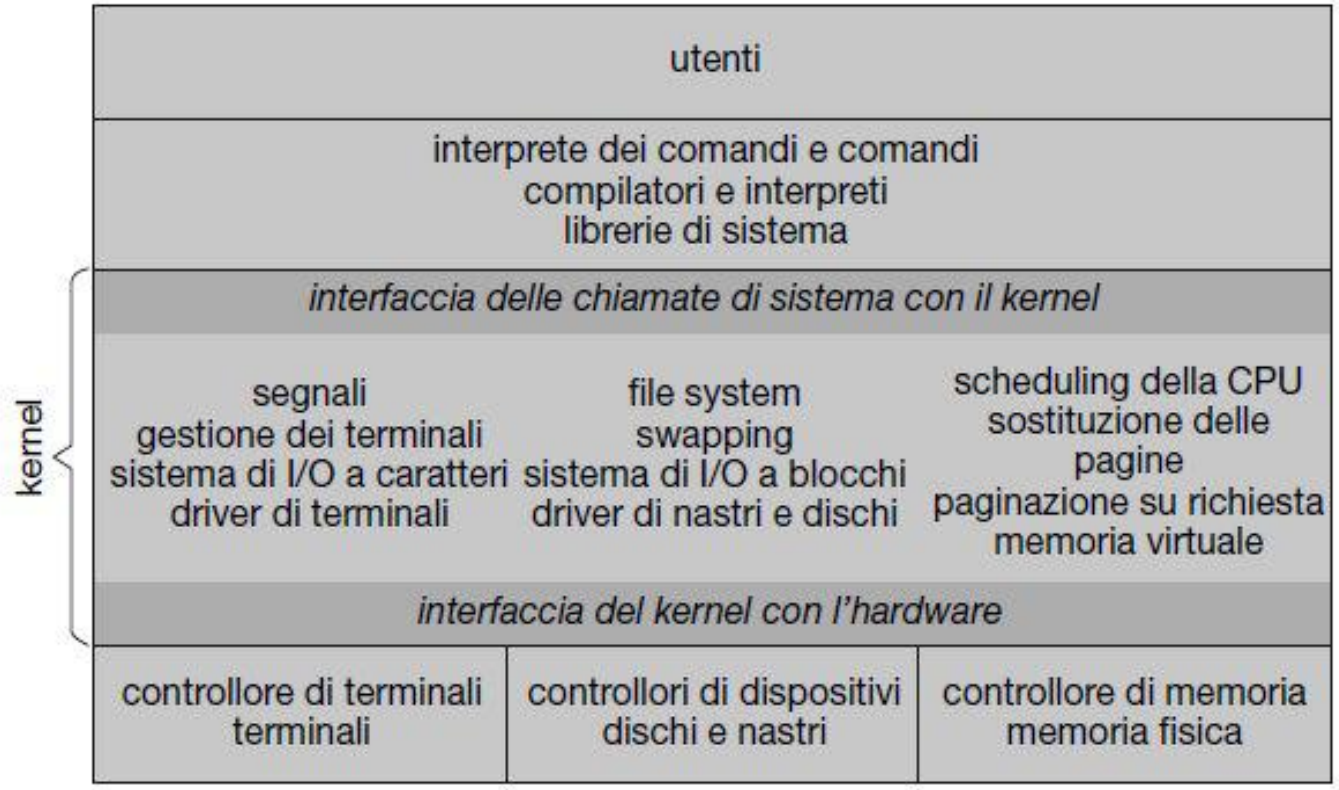
3. Lo sviluppatore di un'applicazione può utilizzare un linguaggio o un'API standard in cui il compilatore genera binari nel linguaggio specifico del sistema operativo e della macchina (per esempio C/C++)

Struttura del sistema operativo

Struttura monolitica

Un sistema monolitico viene anche chiamato **sistema strettamente accoppiato** (*tightly coupled)*

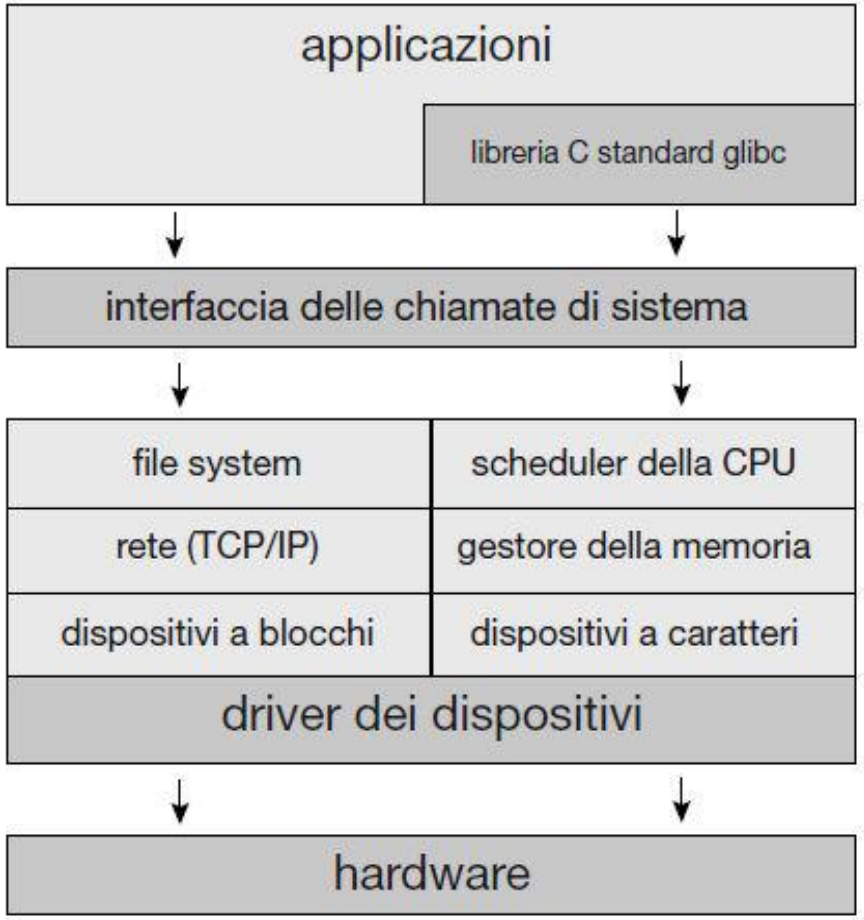
In alternativa, è possibile progettare un **sistema debolmente accoppiato** (*loosely coupled)*



Struttura del sistema UNIX.

Struttura del sistema Linux

Il sistema operativo **Linux** è basato su UNIX ed è strutturato in modo simile

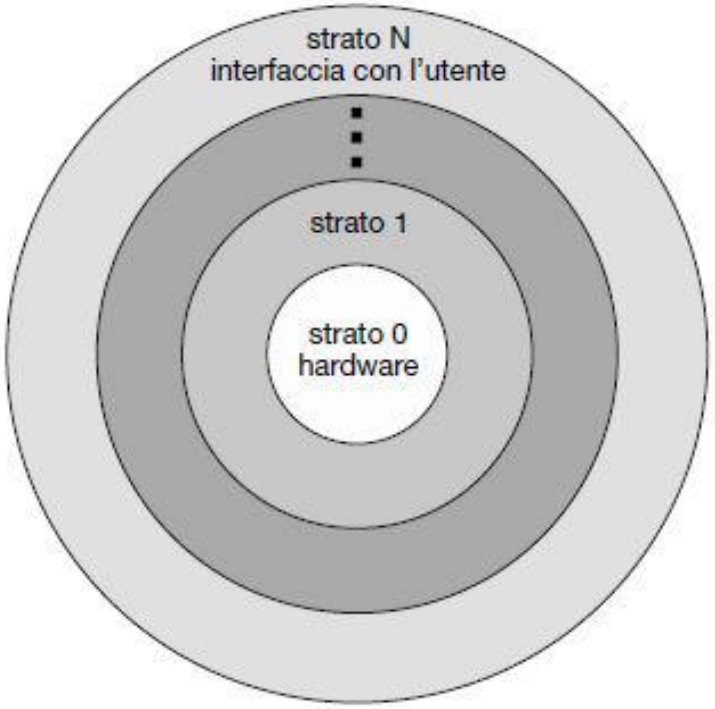


Struttura del sistema Linux.

Approccio stratificato

Per realizzare un **sistema debolmente accoppiato** (*loosely coupled)* è possibile suddividere le funzioni del kernel in moduli

Vi sono molti modi per rendere modulare un sistema operativo. Uno di essi è l'**approccio stratificato.**

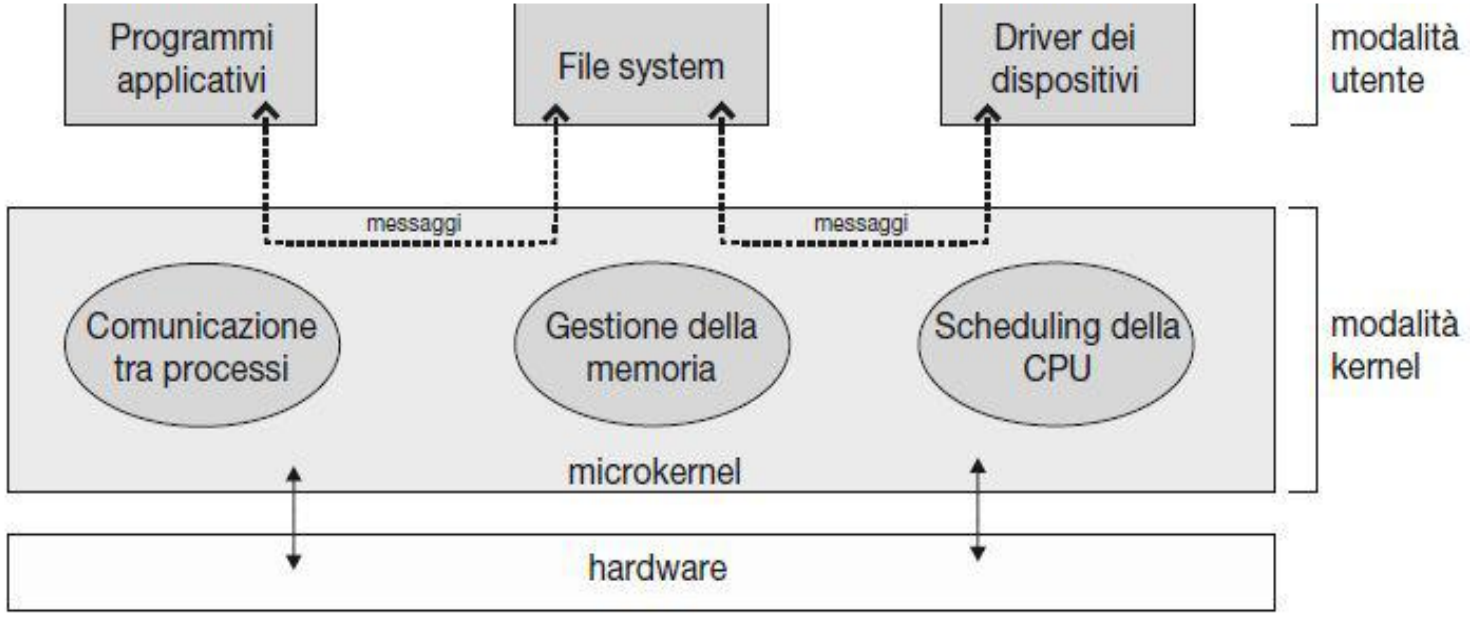


Struttura a strati di un sistema operativo.

Microkernel

Verso la metà degli anni ’80 fu realizzato un sistema operativo, **Mach**, con il kernel strutturato in moduli secondo il cosiddetto **orientamento a microkernel**.

Scopo principale del microkernel → fornire funzioni di comunicazione tra i programmi client e i vari servizi, anch’essi in esecuzione nello spazio utente.

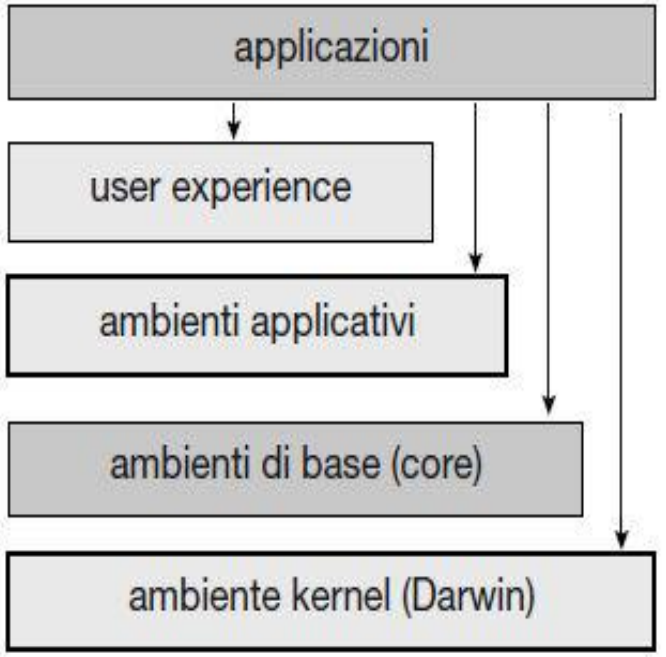


Architettura tipica di un microkernel.

macOS e iOS

Il sistema operativo **macOS** di Apple è progettato per funzionare principalmente su *computer desktop* e *laptop*, mentre **iOS** è un sistema operativo mobile progettato per *iPhone* e *iPad*.

- Strato dell'interfaccia utente (***user experience***)
- Strato degli **ambienti applicativi**
- **Ambienti di base (*core*)**
- **Ambiente kernel** (noto anche come **Darwin**)

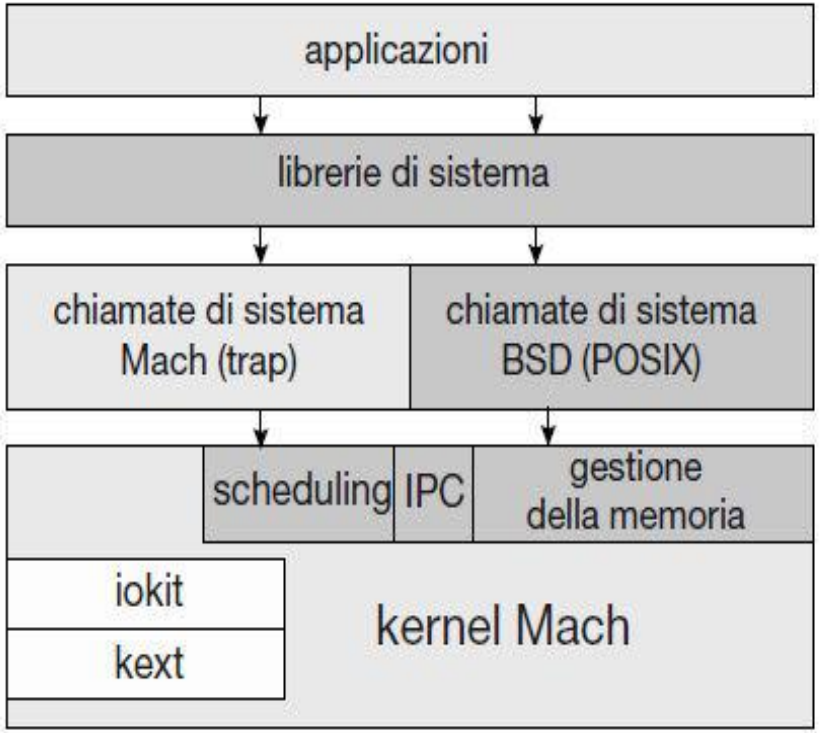


Architettura dei sistemi operativi macOS e iOS di Apple.

Darwin

Darwin è un sistema a strati costituito principalmente dal microkernel Mach e dal kernel BSD UNIX.

Apple ha rilasciato il sistema operativo Darwin come open-source

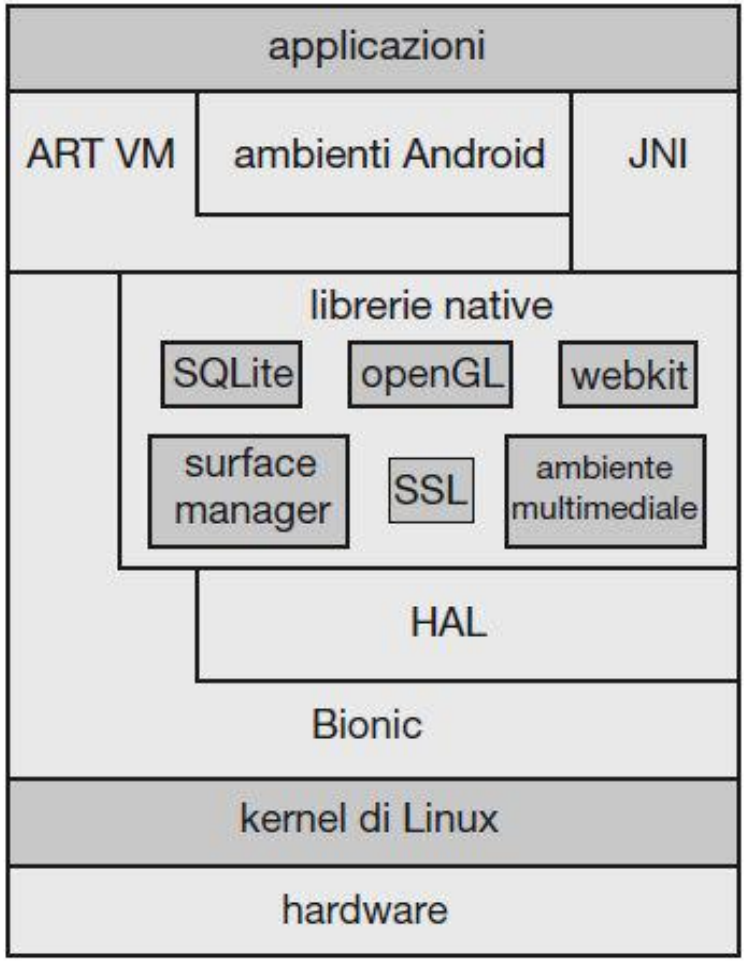


La struttura di Darwin.

Android

Mentre iOS è progettato per funzionare su dispositivi mobili di Apple ed è un software proprietario, (Google) **Android** gira su una varietà di piattaforme mobili ed è open-source.

Poiché Android può essere eseguito su un numero quasi illimitato di dispositivi, Google ha scelto di astrarre l'hardware attraverso uno strato di astrazione hardware detto HAL (hardware abstraction layer).

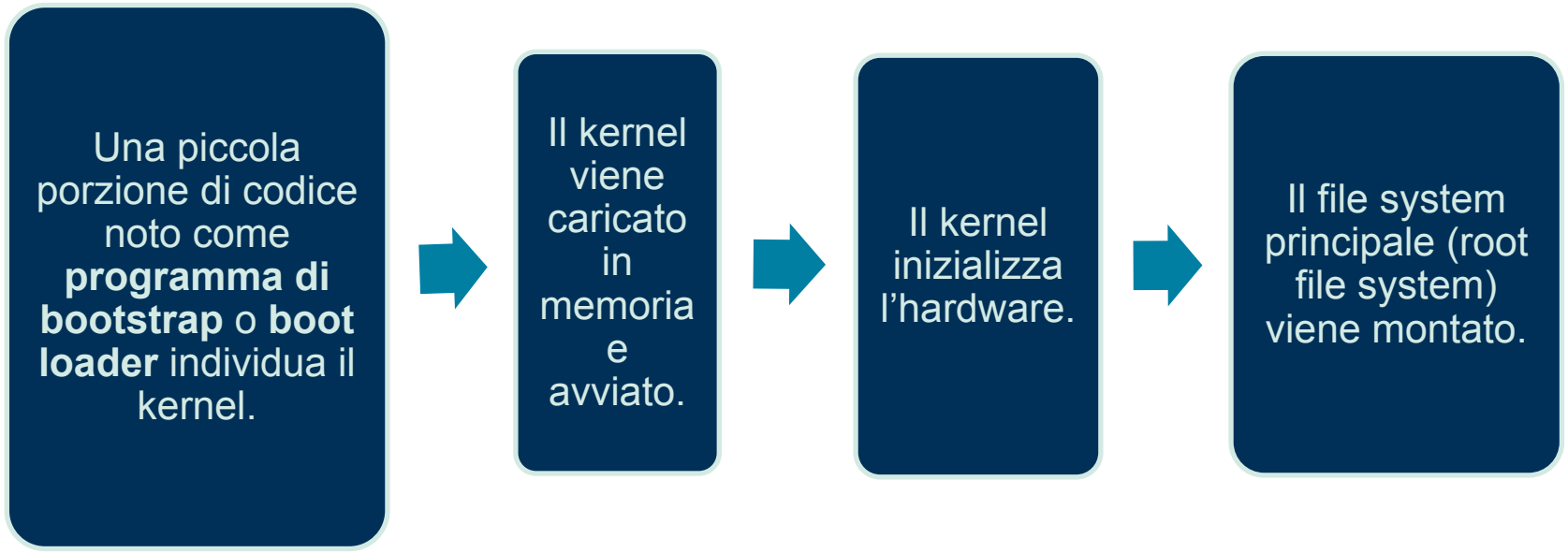


Architettura di Google Android.

Generare e avviare un OS

Avvio del sistema operativo

Il processo di avvio di un computer, caricando il kernel del sistema operativo, è noto come ***boot.***



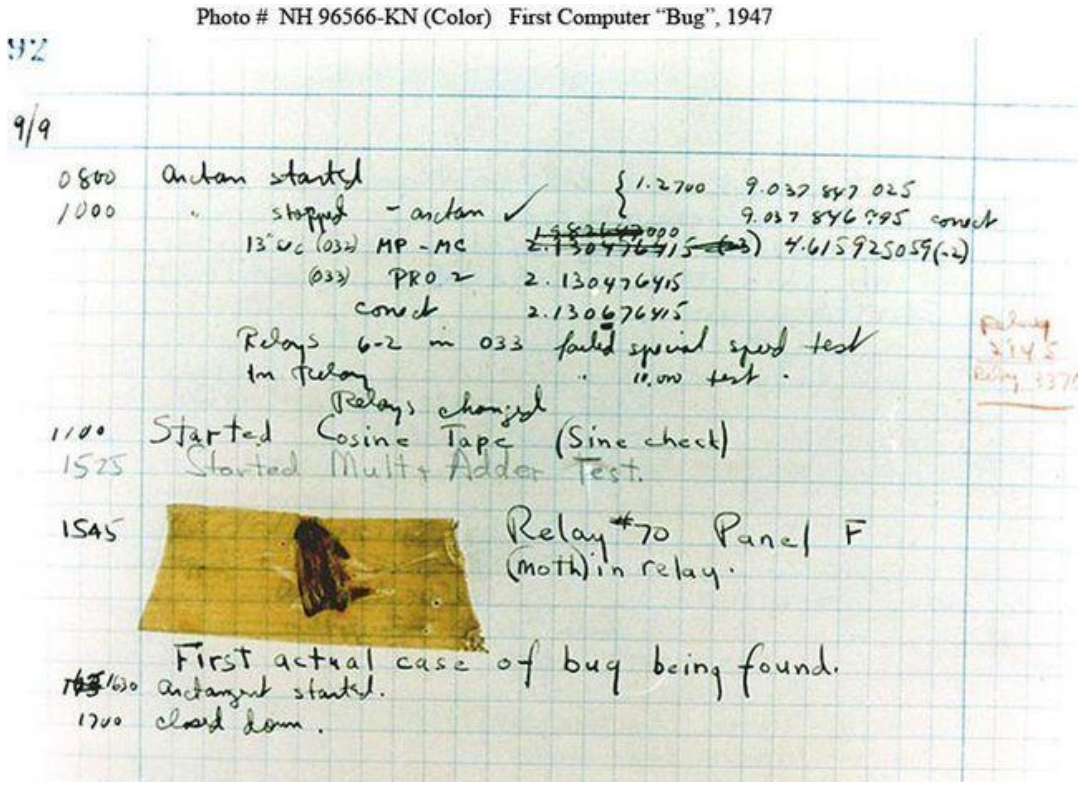
System boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Debugging

Debugging → l'attività di individuare e risolvere errori hardware e software nel sistema, i cosiddetti **bachi (*bug*),** *ma anche*

- **regolazione delle prestazioni (*performance tuning*)**
- **colli di bottiglia (*bottleneck*)** del sistema



Debugging

Se il *debugging di processi* a livello utente è una sfida, a livello del kernel del sistema operativo è un'attività ancora più difficile a causa della dimensione e della complessità del kernel, del suo controllo dell'hardware e della mancanza di strumenti per eseguire il debugging a livello utente.

↓

Un guasto nel kernel viene chiamato **crash**

LA LEGGE DI KERNIGHAN

"Il debugging è due volte più complesso rispetto alla stesura del codice. Di conseguenza, chi scrive il codice nella maniera più intelligente possibile non è, per definizione, abbastanza intelligente per eseguirne il debugging."

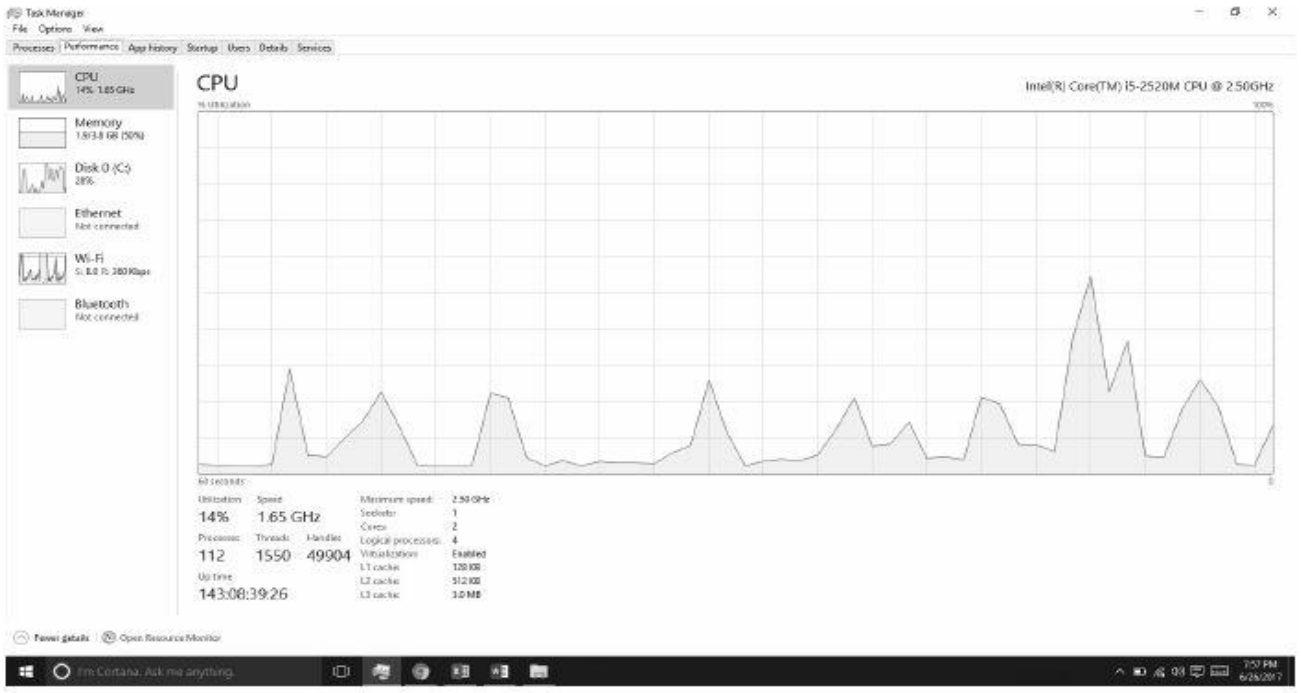
Prestazioni

Tracing o tracciamento

gli strumenti di tracing raccolgono i dati relativi a uno specifico evento

Contatori

contano per esempio il numero di chiamate di sistema effettuate o il numero di operazioni eseguite su un dispositivo o su un disco di rete



Il task manager di Windows 10.