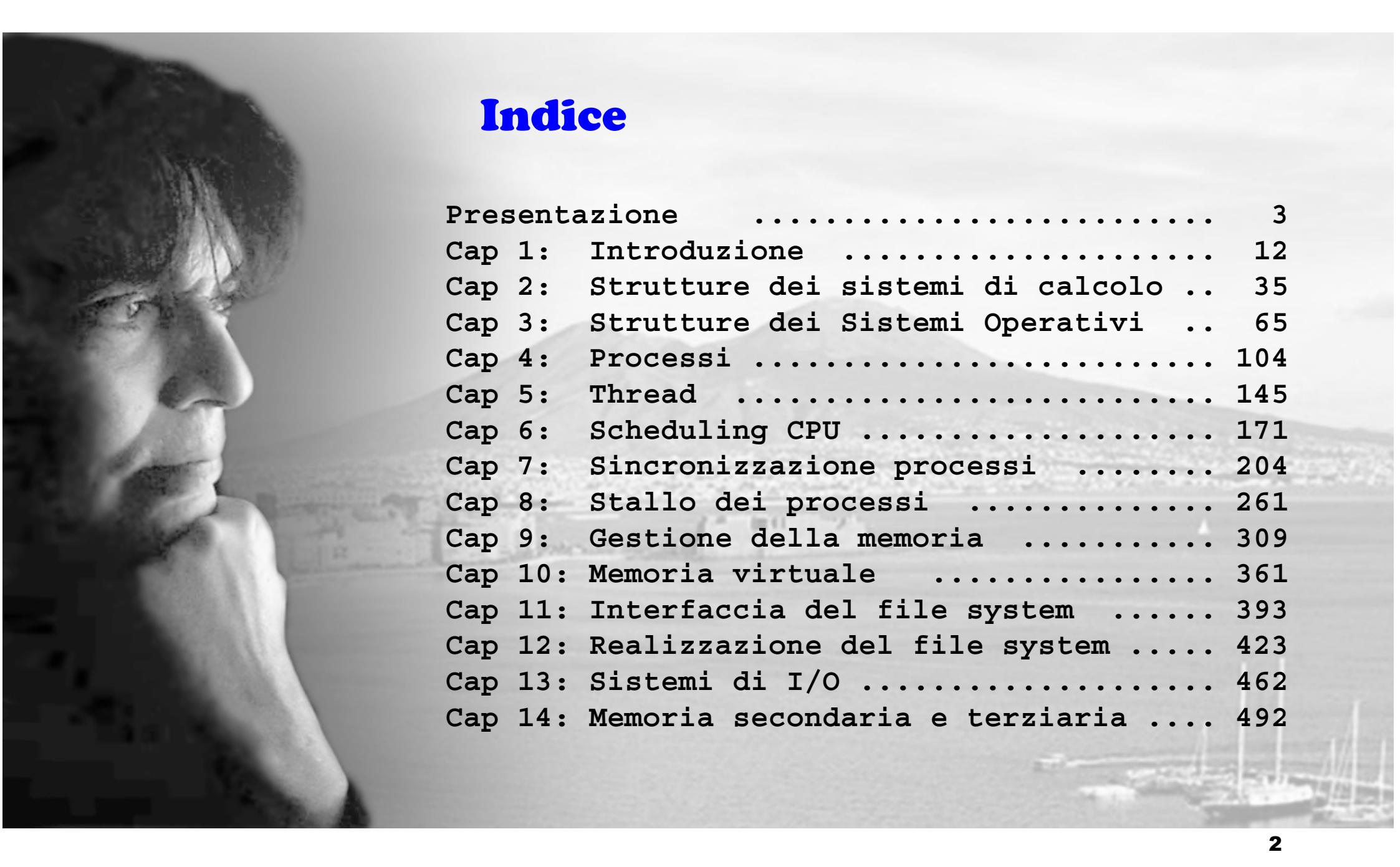


Walter Balzano
Università Federico II
Napoli



Sistemi Operativi



Indice

Presentazione	3
Cap 1: Introduzione	12
Cap 2: Strutture dei sistemi di calcolo ..	35
Cap 3: Strutture dei Sistemi Operativi ..	65
Cap 4: Processi	104
Cap 5: Thread	145
Cap 6: Scheduling CPU	171
Cap 7: Sincronizzazione processi	204
Cap 8: Stallo dei processi	261
Cap 9: Gestione della memoria	309
Cap 10: Memoria virtuale	361
Cap 11: Interfaccia del file system	393
Cap 12: Realizzazione del file system	423
Cap 13: Sistemi di I/O	462
Cap 14: Memoria secondaria e terziaria	492

Presentazione al corso di

Sistemi Operativi

DOCENTE: prof. Walter Balzano

MATRICOLE: Gruppo unico



walter.balzano@gmail.com

SITO WEB:

- 1 <https://sites.google.com/site/walterbalzano/>
- 2 Didattica
- 3 Sistemi Operativi

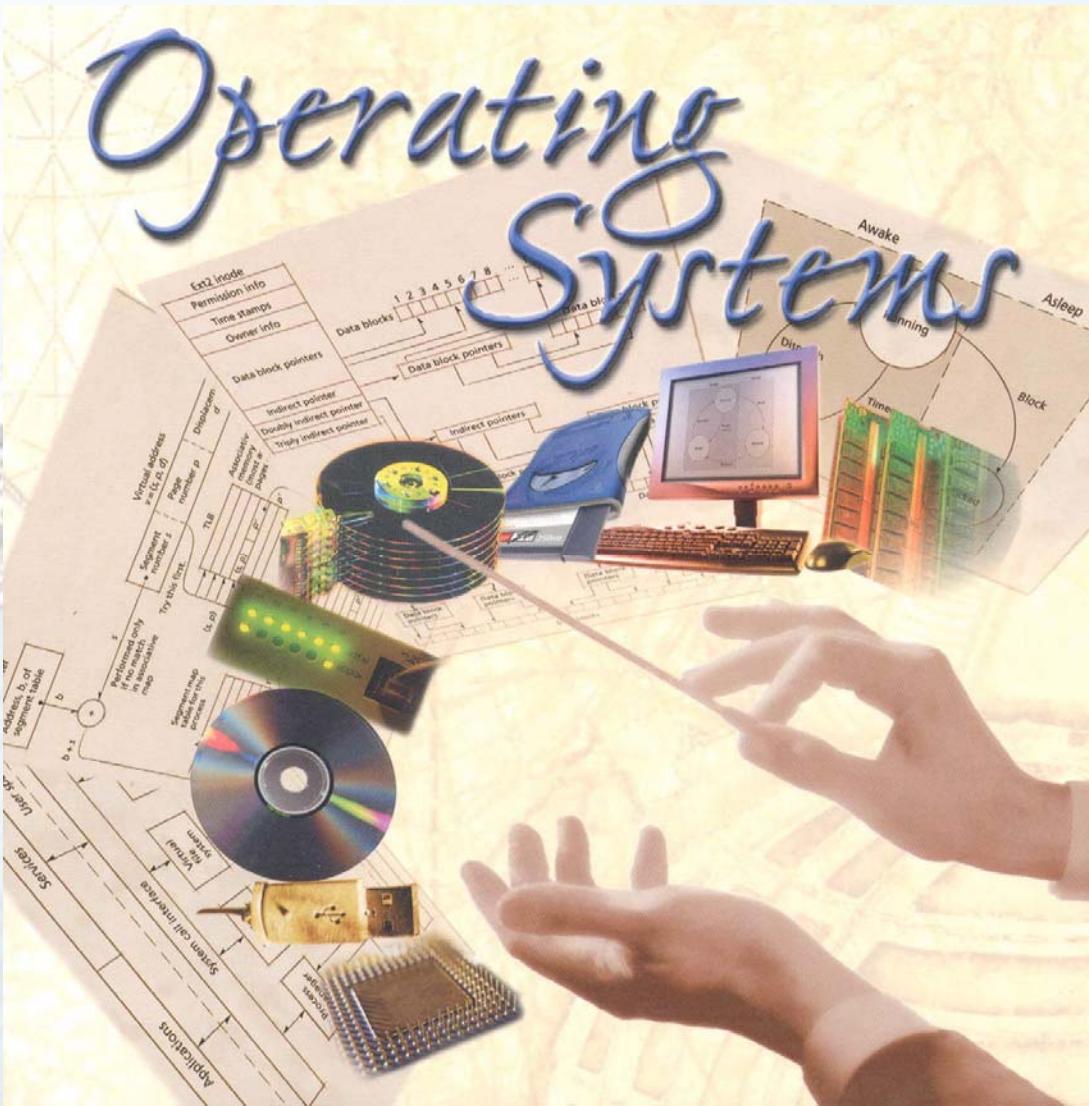
NOTA BENE:

le slides e le video lezioni non sostituiscono i libri di testo!



walter.balzano@gmail.com

Operating Systems



walter.balzano@gmail.com

Programma

Parte prima: Generalità

Capitolo 1: Introduzione

Cos'è un Sistema Operativo; Sistemi mainframe; Sistemi da scrivania; Sistemi con più unità di elaborazione; Sistemi distribuiti; Batterie di sistemi; Sistemi di elaborazione in tempo reale; Sistemi palmari; Migrazione delle funzioni; Ambienti di elaborazione

Capitolo 2: Strutture dei sistemi di calcolo

Funzionamento di un sistema di calcolo; Struttura di I/O; Struttura della memoria; Gerarchia delle memorie; Architetture di protezione; Struttura delle reti di calcolatori

Capitolo 3: Strutture dei sistemi operativi

Componenti del sistema; Servizi di un sistema operativo; Chiamate del sistema; Programmi del sistema; Struttura del sistema; Macchine virtuali; Progettazione e realizzazione di un sistema; Generazione di sistemi



walter.balzano@gmail.com

Programma

Parte seconda: Gestione dei processi

Capitolo 4: Processi

Concetto di processo; Scheduling dei processi; Operazioni sui processi; Processi cooperanti; Comunicazione tra processi; Comunicazione nei sistemi client/server

Capitolo 5: Thread

Introduzione; Modelli di programmazione multithread

Capitolo 6: Scheduling della CPU

Concetti fondamentali; Criteri di scheduling; Algoritmi di scheduling; Scheduling per sistemi con più unità di elaborazione; Scheduling per sistemi di elaborazione in tempo reale; Valutazione degli algoritmi

Capitolo 7: Sincronizzazione dei processi

Introduzione; Problema della sezione critica; Architetture di sincronizzazione; Semafori; Problemi tipici di sincronizzazione; Regioni critiche; Monitor

Capitolo 8: Stallo dei processi

Modello del sistema; Caratterizzazione delle situazioni di stallo; Metodi per la gestione delle situazioni di stallo; Prevenire le situazioni di stallo; Evitare le situazioni di stallo; Rilevamento delle situazioni di stallo; Ripristino da situazioni di stallo



walter.balzano@gmail.com

Programma

Parte terza: Gestione della memoria

Capitolo 9: Gestione della memoria

Introduzione; Avvicendamento dei processi; Assegnazione contigua della memoria; Paginazione; Segmentazione

Capitolo 10: Memoria Virtuale

Introduzione; Paginazione su richiesta; Creazione dei processi; Sostituzione delle pagine

Capitolo 11: Interfaccia del File-System

Concetto di file; Metodi di accesso; Struttura di directory; Montaggio di un file system; Condivisione di file; Protezione

Capitolo 12: Realizzazione del File System

Struttura del file system; Realizzazione del file system; Realizzazione delle directory; Metodi di assegnazione; Gestione dello spazio libero; Efficienza e prestazioni; Ripristino; File system con annotazione delle modifiche



walter.balzano@gmail.com

Programma

Parte quarta: Sistemi di I/O

Capitolo 13: Sistemi di I/O

Introduzione; Architetture e dispositivi di I/O; Interfaccia di I/O per le applicazioni; Sottosistema per l'I/O del nucleo; Trasformazione delle richieste di I/O in operazioni dei dispositivi; Prestazioni

Capitolo 14: Memoria secondaria e terziaria

Struttura dei dischi; Scheduling del disco; Gestione dell'unità a disco; Gestione dell'area di avvicendamento; Strutture RAID; Connessione dei dischi; Strutture per la memorizzazione terziaria



walter.balzano@gmail.com

Libri di testo consigliati:



**Abraham Silberschatz,
Peter Baer Galvin,
Greg Gagne**

**"*Sistemi Operativi.
Concetti ed esempi*"**

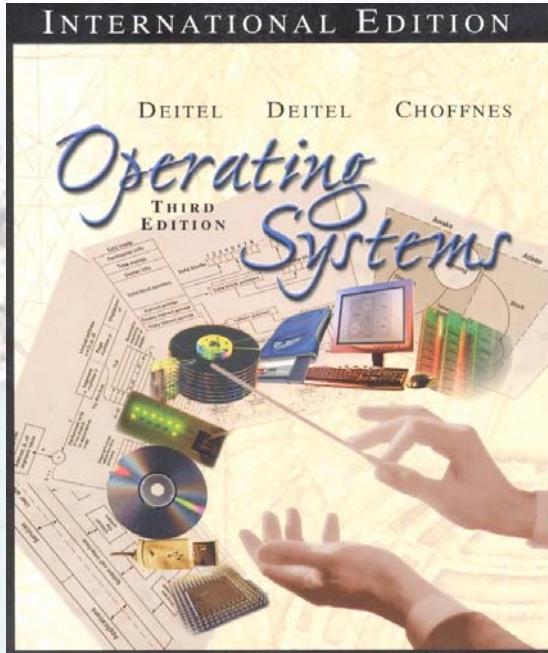
**Sesta ediz., Addison-Wesley, 2003.
ISBN: 88-7192-140-2**

(Costo ~ 44 euro)



walter.balzano@gmail.com

Libri di testo consigliati:



Deitel, Deitel, Choffnes

"*Operating Systems*",

Prentice-Hall



walter.balzano@gmail.com

Capitolo 1: Introduzione

- **Cos'è un sistema operativo**
- **Sistemi mainframe**
- **Sistemi da scrivania**
- **Sistemi con più unità d'elaborazione**
- **Sistemi distribuiti**
- **Batterie di sistemi (cluster)**
- **Sistemi d'elaborazione in tempo reale**
- **Sistemi palmari**
- **Ambienti d'elaborazione**



walter.balzano@gmail.com

Cos'è un sistema operativo

- Un insieme di programmi che funge da intermediario tra un utente e gli elementi fisici di un calcolatore (*hardware*).
- Obiettivi di un sistema operativo:
 - Eseguire programmi d'applicazione e rendere più facile la risoluzione dei problemi che gli utenti devono affrontare.
 - Rendere il sistema conveniente da utilizzare.
- Garantisce un utilizzo efficiente dei dispositivi fisici.



walter.balzano@gmail.com

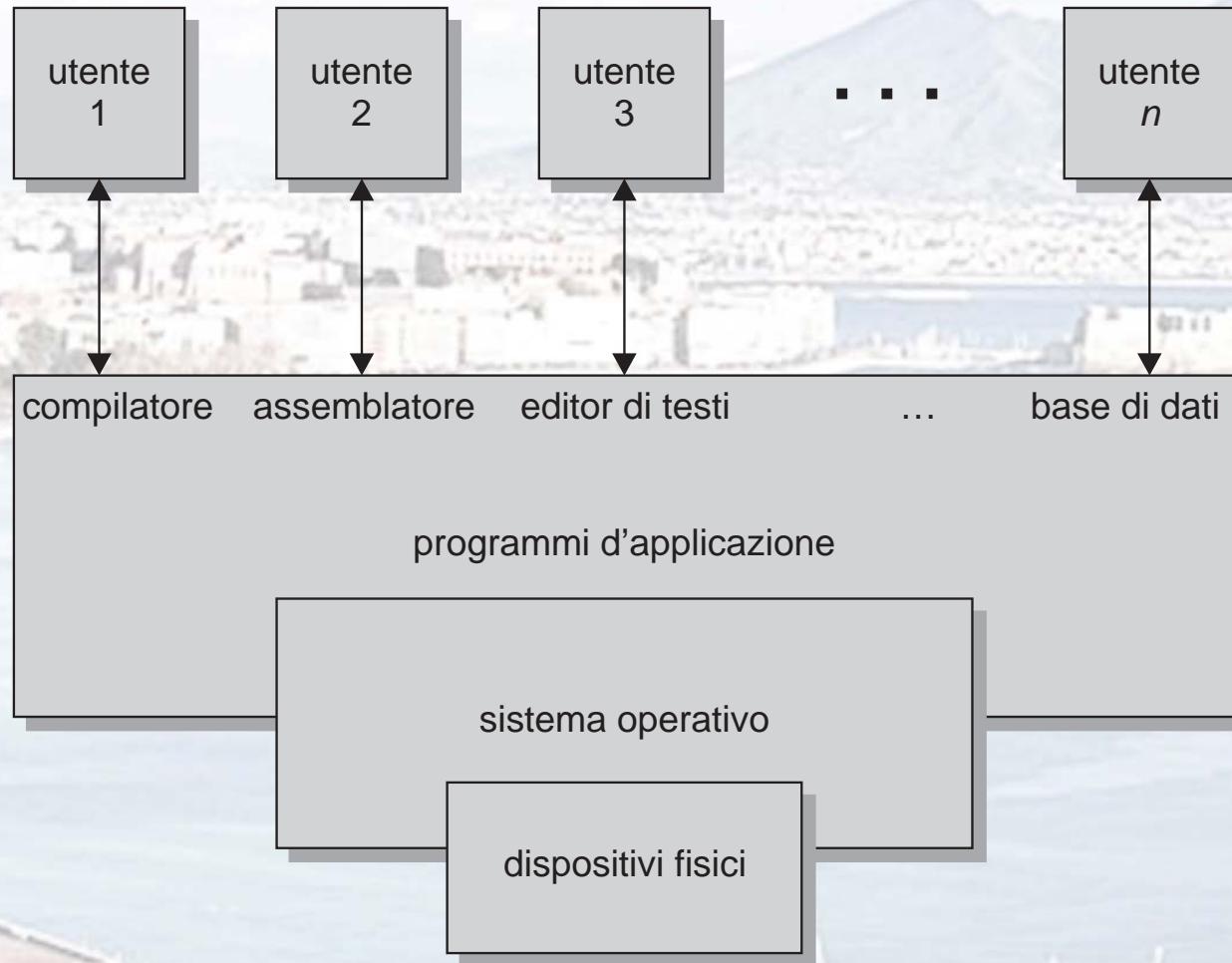
Componenti di un sistema di calcolo

- 1. Dispositivi fisici (hardware):** forniscono le risorse di calcolo fondamentali (CPU, memoria, I/O).
- 2. Sistema operativo:** controlla e coordina l'uso dei dispositivi da parte dei programmi d'applicazione per gli utenti.
- 3. Programmi d'applicazione:** definiscono il modo in cui sono utilizzate le risorse del sistema per risolvere i problemi degli utenti (compilatori, sistemi di basi di dati, videogiochi, applicativi per il business).
- 4. Utenti (persone, macchine, altri calcolatori).**



walter.balzano@gmail.com

Componenti di un sistema di calcolo



walter.balzano@gmail.com

Definizioni di sistema operativo

- **Assegnatore di risorse:** gestisce e assegna le risorse.
- **Programma di controllo:** controlla l'esecuzione dei programmi utenti e le operazioni dei dispositivi di I/O
- **Nucleo (*kernel*):** il solo programma che funziona sempre nel calcolatore (tutti gli altri sono programmi d'applicazione).



walter.balzano@gmail.com

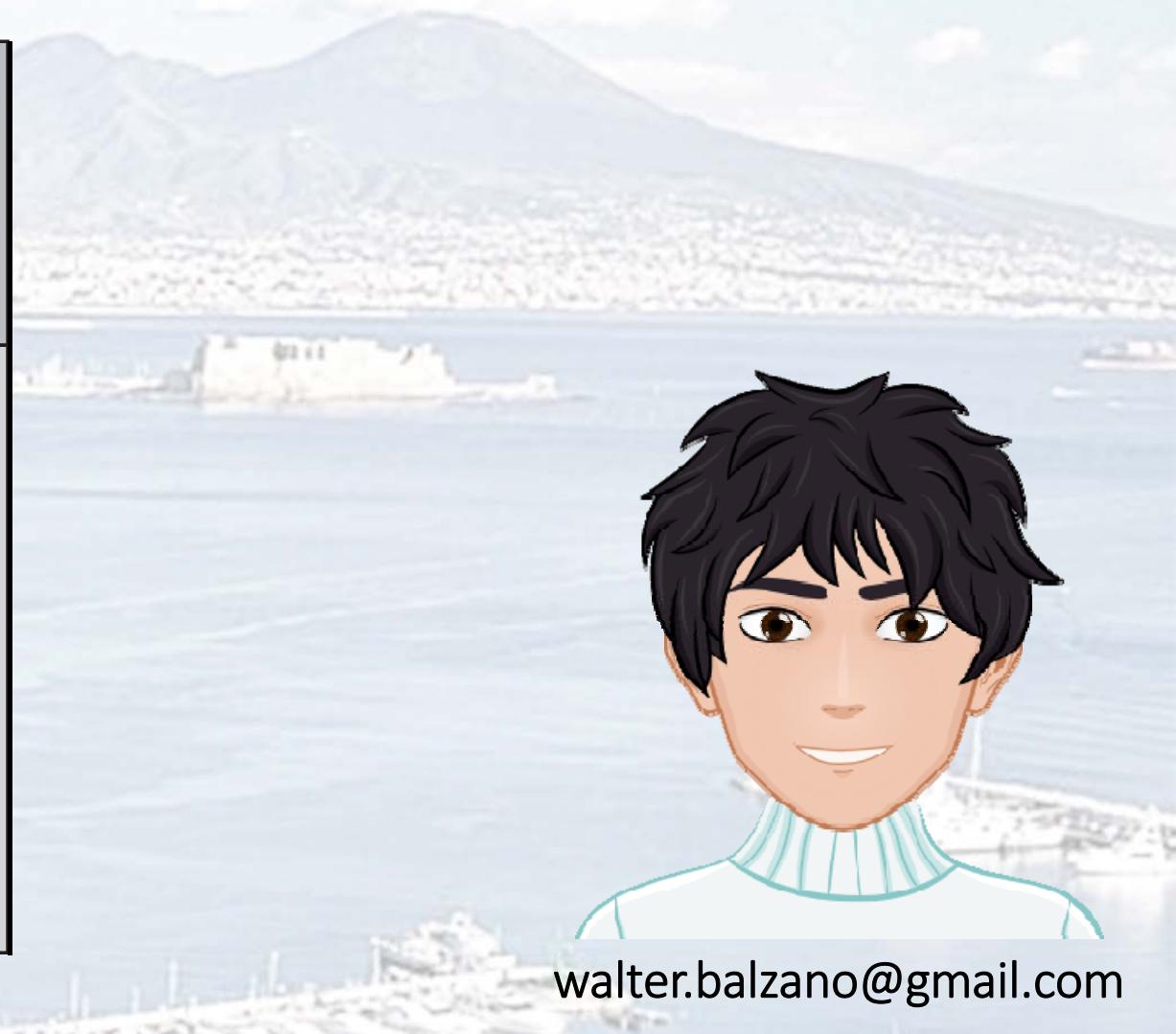
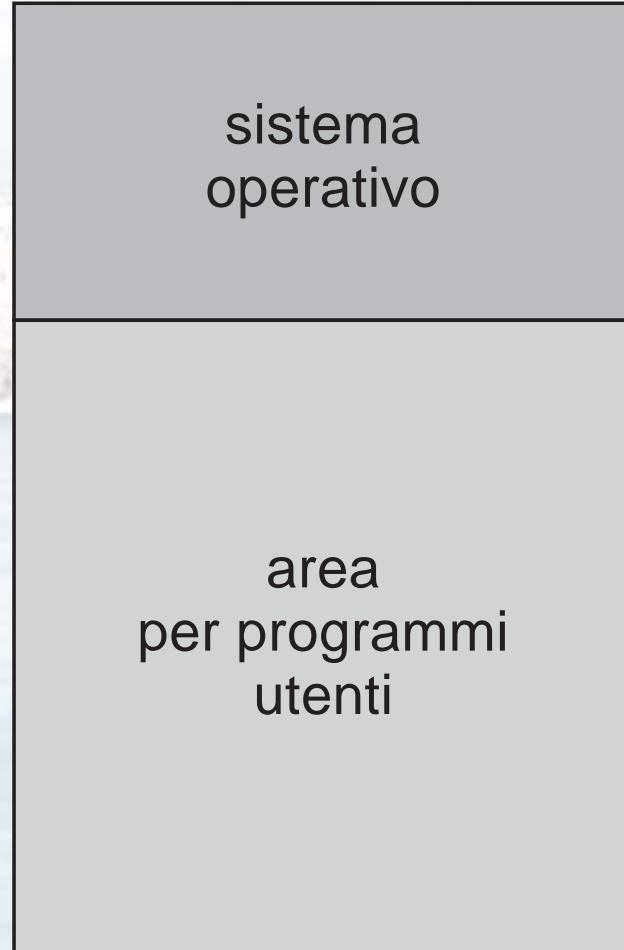
Sistemi mainframe

- Riduzione del tempo di elaborazione raggruppando insieme lavori con requisiti simili
- Trasferimento automatico del controllo da un lavoro a quello successivo. Primo rudimentale sistema operativo.
- Monitor residente



walter.balzano@gmail.com

Configurazione della memoria per un sistema a lotti



walter.balzano@gmail.com

Sistemi multiprogrammati

Il sistema operativo tiene contemporaneamente nella memoria centrale diversi lavori, e la CPU non rimane mai inattiva.



walter.balzano@gmail.com

Caratteristiche di un sistema operativo multiprogrammato

- **Tutti i lavori che entrano nel sistema sono mantenuti in un gruppo che consiste di tutti i processi d'elaborazione presenti nel disco che attendono il caricamento nella memoria centrale.**
- **Gestione della memoria:** il sistema deve ripartire la memoria tra i vari lavori in entrata.
- **Scheduling della CPU:** se in un dato momento più lavori sono pronti per essere eseguiti, il sistema deve scegliere quello da eseguire.
- **Allocazione delle risorse.**



walter.balzano@gmail.com

Sistemi a partizione del tempo d'elaborazione

- La CPU viene ripartita tra i vari lavori tenuti in memoria sul disco (la CPU viene assegnata a un lavoro solo se questo è presente in memoria).
- La CPU esegue più lavori commutando le loro esecuzioni.
- Comunicazione diretta tra utente e sistema; il sistema passa rapidamente da un utente all'altro, quindi ogni utente ha l'impressione di disporre dell'intero calcolatore.
- Ciascun utente dispone di almeno un proprio programma nella memoria.



walter.balzano@gmail.com

Sistemi da scrivania

- ***Personal computer*** : calcolatori utilizzati da un singolo utente.
- Dispositivi di I/O: tastiera, mouse, schermo, piccole stampanti.
- Comodità e prontezza d'uso per l'utente.
- Possibilità di adottare le tecnologie sviluppate per i grandi sistemi.
- Possono funzionare con sistemi operativi diversi (Windows, MacOS, UNIX, Linux)



walter.balzano@gmail.com

Sistemi paralleli

- Sistemi con più unità d'elaborazione, con più CPU in stretta collaborazione.
- Sistemi strettamente connessi: i processori condividono la memoria e i temporizzatori dei cicli di macchina; *and a clock*; la comunicazione di solito avviene attraverso la memoria condivisa.
- Vantaggi dei sistemi paralleli:
 - Maggiore produttività (*throughput*)
 - Economia di scala
 - Incremento dell'affidabilità
 - degradazione controllata
 - tolleranza ai guasti



walter.balzano@gmail.com

Sistemi paralleli (cont.)

- ***Multielaborazione simmetrica (SMP)***

- Ciascuna unità d'elaborazione esegue un'identica copia del sistema operativo.
- Si possono eseguire molti processi contemporaneamente senza causare un rilevante calo delle prestazioni.
- La maggior parte dei moderni sistemi operativi supporta la SMP

- ***Multielaborazione asimmetrica (AMP)***

- A ogni unità d'elaborazione si assegna un compito specifico; un'unità d'elaborazione principale controlla il sistema, le altre attendono istruzioni dall'unità principale.
- Più comune nei sistemi di grandi dimensioni



walter.balzano@gmail.com

Architettura per la multielaborazione simmetrica



walter.balzano@gmail.com

Sistemi distribuiti

- I sistemi distribuiti si basano sulle reti per realizzare le proprie funzioni, sfruttano le capacità di comunicazione per cooperare nella soluzione dei problemi di calcolo e per fornire agli utenti un ricco insieme di funzioni.
- **Sistema distribuito (*loosely coupled system*):** ciascuna unità d'elaborazione ha la propria memoria locale e comunica con le altre per mezzo di linee di comunicazione di vario genere, ad esempio bus ad alta velocità o linee telefoniche.
- **Vantaggi.**
 - Condivisione delle risorse
 - Maggiore velocità
 - Affidabilità
 - Comunicazione



walter.balzano@gmail.com

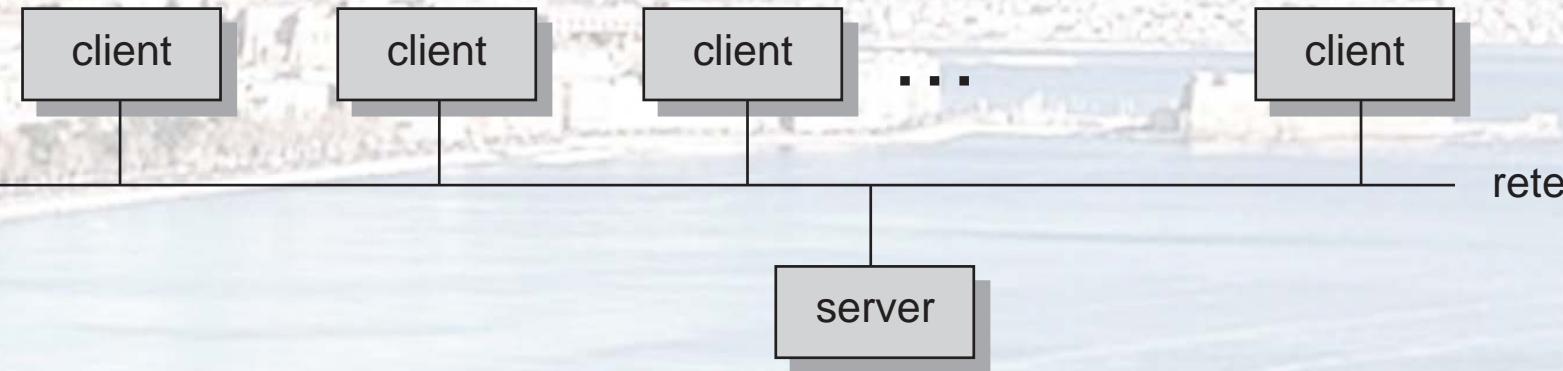
Sistemi distribuiti (cont.)

- Richiedono un'infrastruttura di rete.
- Local Area Networks (LAN) o Wide Area Networks (WAN)
- Possono essere sia sistemi client-server sia sistemi peer-to-peer.



walter.balzano@gmail.com

Struttura generale di un sistema client-server



walter.balzano@gmail.com

Batterie di sistemi (sistemi cluster)

- Le **batterie di sistemi** (*cluster system*) sono basate sull'uso congiunto di più unità d'elaborazione riunite per svolgere attività d'elaborazione comuni.
- Forniscono un'elevata disponibilità.
- **Batterie asimmetriche** (*asymmetric clustering*): un calcolatore rimane nello stato di **attesa attiva** (*hot standby*) mentre l'altro esegue le applicazioni.
- **Batterie simmetriche** (*symmetric clustering*): due o più calcolatori eseguono le applicazioni e allo stesso tempo si controllano reciprocamente.



walter.balzano@gmail.com

Sistemi d'elaborazione in tempo reale

- Spesso impiegati nella gestione dei dispositivi di controllo per applicazioni specifiche (controllo di esperimenti scientifici, rappresentazione d'immagini in medicina, sistemi di controllo industriale, ...).
- Presenta vincoli di tempo fissati e ben definiti entro i quali *si deve* effettuare l'elaborazione.
- Esistono due tipi di sistemi d'elaborazione in tempo reale:
 - sistemi d' elaborazione in tempo reale stretto (*hard real-time*)
 - sistemi d'elaborazione in tempo reale debole (*soft real-time*).



walter.balzano@gmail.com

Sistemi d'elaborazione in tempo reale *(Cont)*

- **Sistemi d'elaborazione in tempo reale stretto:**
 - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
 - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- **Sistemi d'elaborazione in tempo reale debole**
 - Limited utility in industrial control of robotics
 - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.



walter.balzano@gmail.com

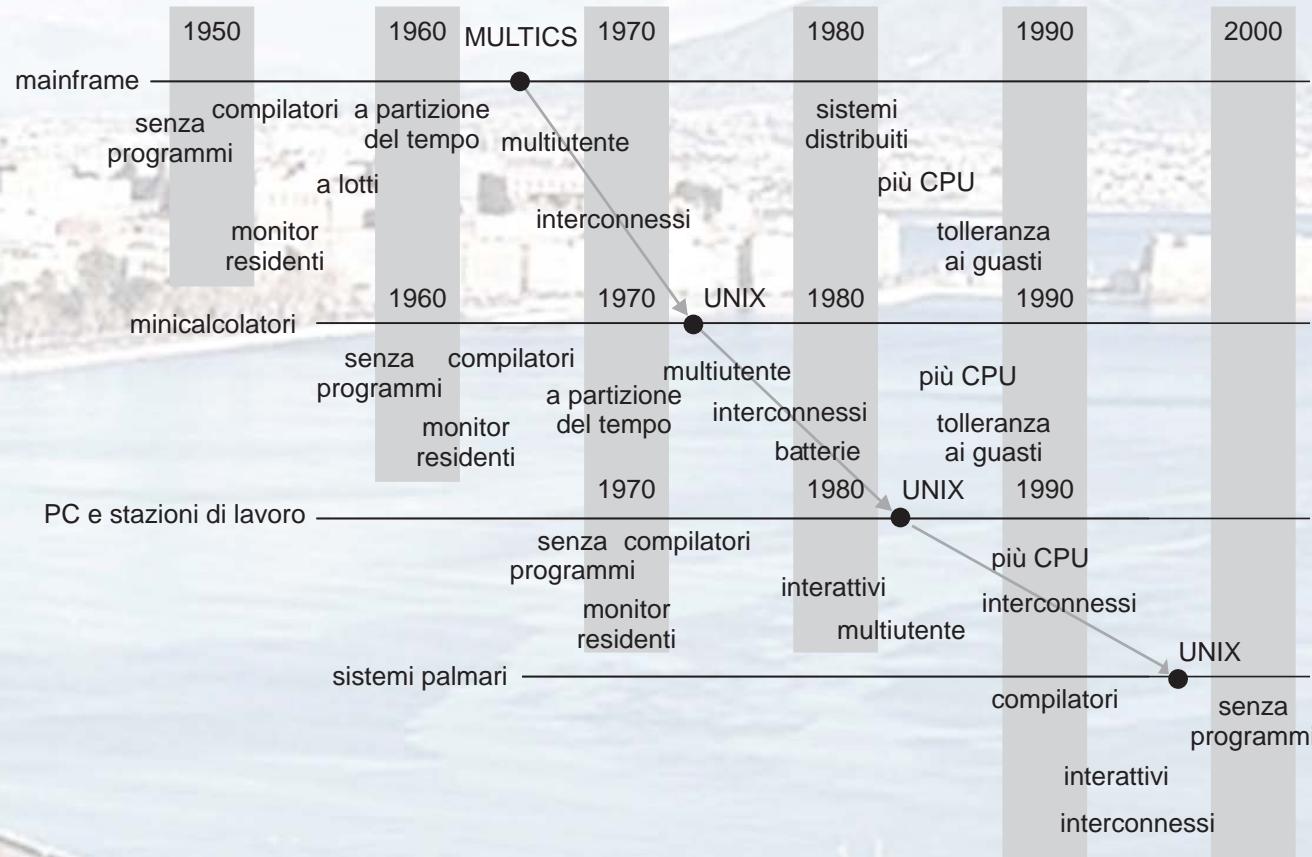
Sistemi palmari

- Personal Digital Assistant (PDA)
- Telefoni cellulari
- A causa delle piccole dimensioni, la maggior parte dei dispositivi palmari dispone di:
 - memoria limitata
 - unità d'elaborazione lente
 - schermi piccoli.



walter.balzano@gmail.com

Migrazione dei concetti e delle caratteristiche dei sistemi operativi



walter.balzano@gmail.com

Ambienti d'elaborazione

- Elaborazione tradizionale
- Elaborazione basata sul Web
- Dispositivi d'elaborazione integrati



walter.balzano@gmail.com

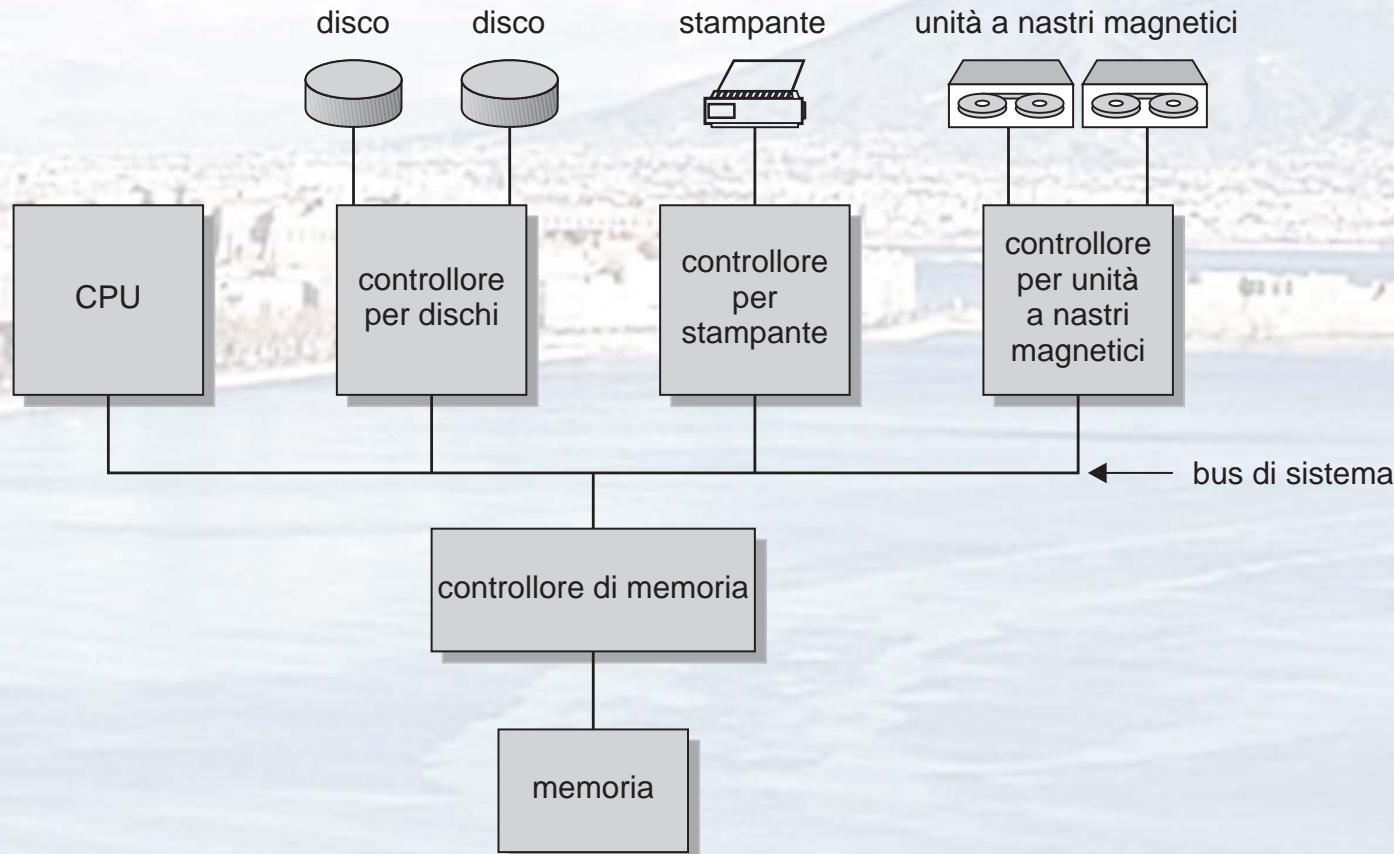
Capitolo 2: Strutture dei sistemi di calcolo

- **Funzionamento** di un sistema di calcolo
- Struttura di **I/O**
- **Struttura della memoria**
- **Gerarchia** delle memorie
- Architetture di **protezione**
- Struttura delle **reti** di calcolatori



walter.balzano@gmail.com

Architettura di un sistema di calcolo



walter.balzano@gmail.com

Funzionamento di un sistema di calcolo

- I dispositivi di I/O e la CPU possono operare in modo concorrente.
- Ciascun controllore si occupa di un particolare tipo di dispositivo fisico (es. unità a disco, dispositivi audio, ...).
- Ciascun controllore ha un buffer locale.
- La CPU sposta i dati da/verso la memoria principale da/verso i buffer locali.
- L'I/O avviene dal dispositivo al buffer locale del controllore.
- Il controllore informa la CPU di aver terminato un'operazione causando un segnale d'interruzione (interrupt).



walter.balzano@gmail.com

Funzioni comuni dei segnali d'interruzione

- Un segnale d'interruzione deve causare il trasferimento del controllo all'appropriata procedura di servizio dell'evento a esso associato.
- L'architettura di gestione delle interruzioni deve anche salvare l'indirizzo dell'istruzione interrotta.
- Un **segnale d'eccezione** (*trap*) può essere causato da un programma in esecuzione a seguito di un evento eccezionale oppure a seguito di una richiesta specifica effettuata da un programma utente.
- Un moderno sistema operativo è detto **guidato dalle interruzioni** (*interrupt driven*).



walter.balzano@gmail.com

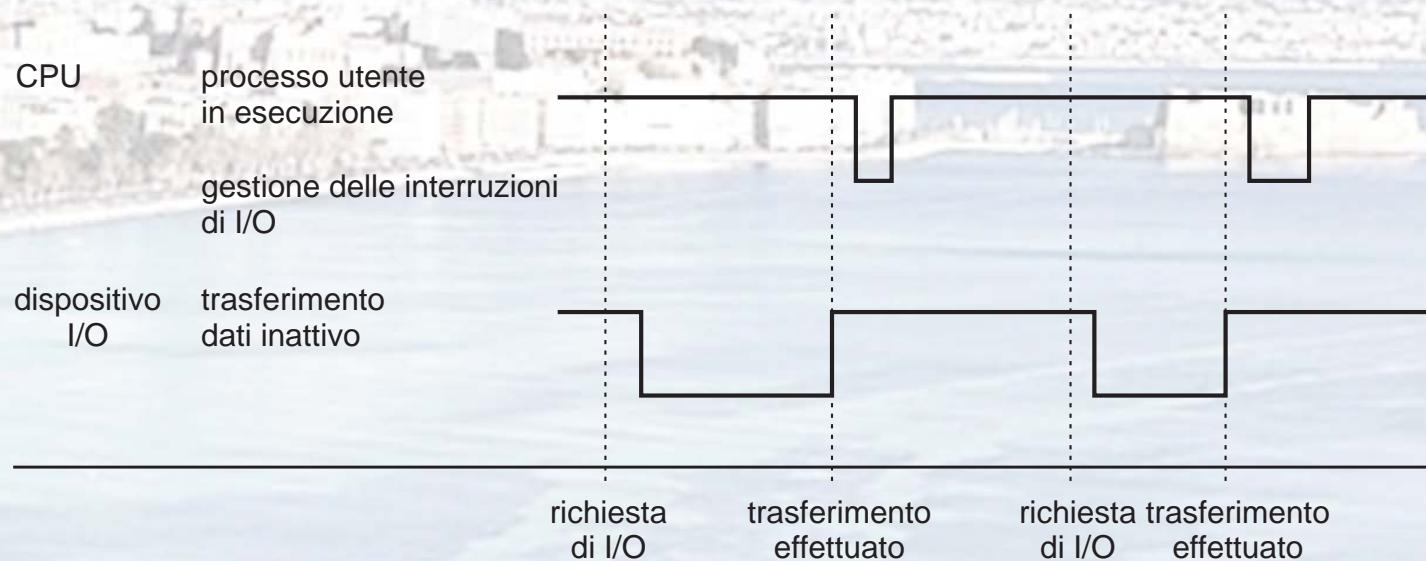
Gestione dell'interruzione

- Il sistema operativo memorizza l'indirizzo di ritorno nella pila (*stack*) di sistema.
- Determina quale tipo di interruzione si è verificato:
 - *polling*
 - *vectored interrupt system*
- Segmenti separati di codice determinano quale azione debba essere intrapresa per ciascun tipo di interruzione.



walter.balzano@gmail.com

Diagramma temporale delle interruzioni per un singolo processo che emette dati



walter.balzano@gmail.com

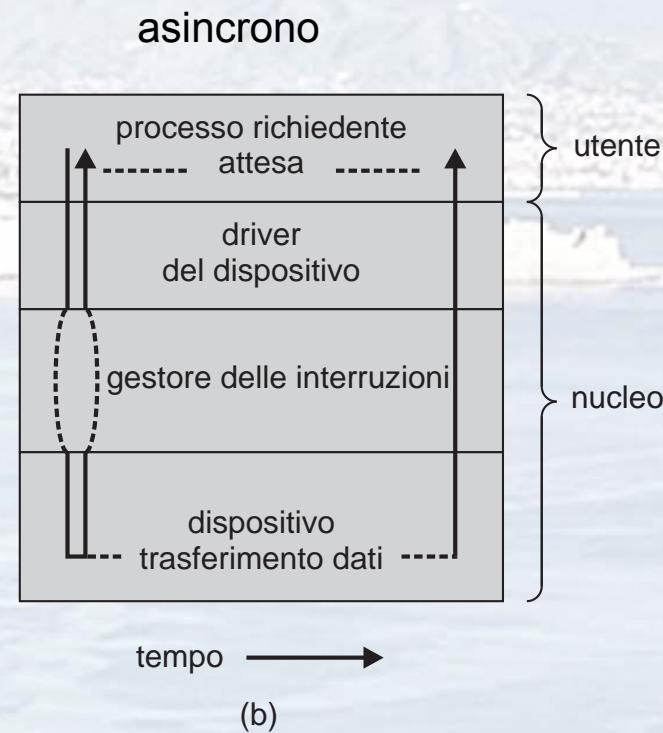
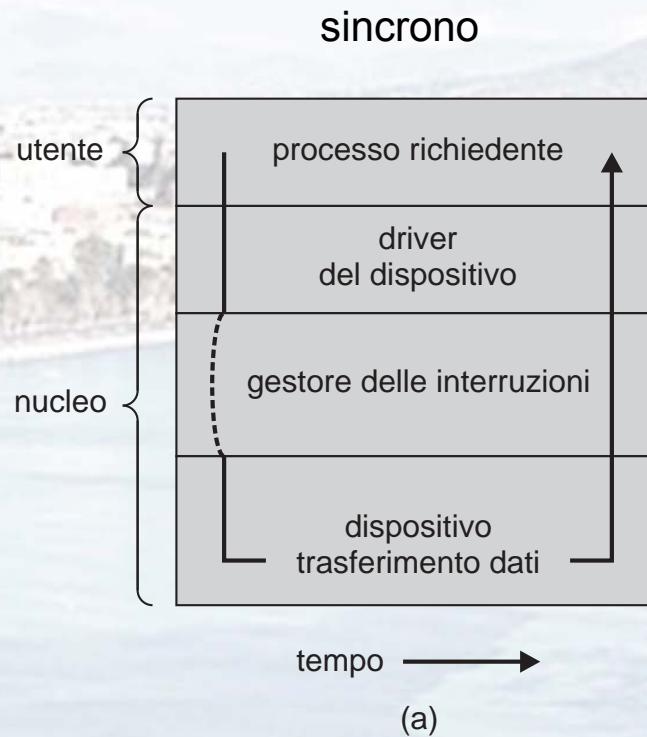
Struttura di I/O

- Una volta iniziata l'operazione di I/O, si restituisce il controllo al processo utente solo dopo il completamento dell'operazione di I/O.
 - L'istruzione `wait` sospende la CPU fino al successivo segnale d'interruzione.
 - Nei calcolatori che non prevedono un'istruzione del genere, si può generare in ciclo d'attesa
 - Si ha al più una richiesta pendente alla volta.
- Una volta iniziata l'operazione di I/O, si restituisce immediatamente il controllo al processo utente, senza attendere il completamento dell'operazione di I/O.
 - *System call* – richiesta al sistema operativo di consentire al programma utente di attendere il completamento dell'operazione.
 - *La tabella di stato dei dispositivi* contiene elementi per ciascun dispositivo di I/O che ne specificano il tipo, l'indirizzo e lo stato.
 - Il sistema operativo individua il controllore del dispositivo che ha emesso il segnale d'interruzione, quindi accede alla tabella dei dispositivi, risale allo stato in cui il dispositivo si trova e modifica l'elemento della tabella indicando l'occorrenza dell'interruzione.



walter.balzano@gmail.com

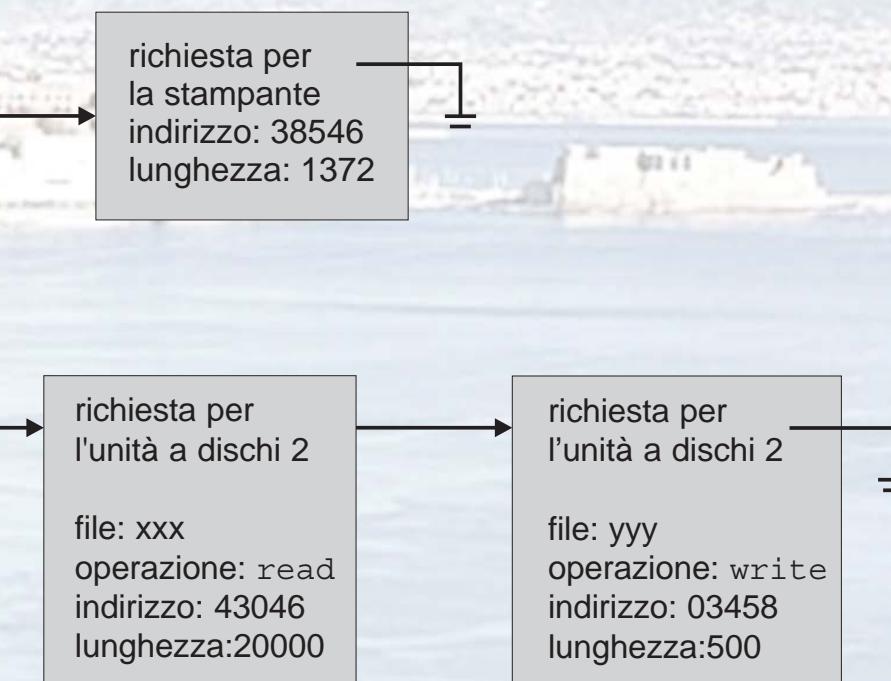
Due metodi di I/O



walter.balzano@gmail.com

Tabella di stato dei dispositivi

dispositivo: tastiera stato: inattivo
dispositivo: stampante stato: attivo
dispositivo: mouse stato: inattivo
dispositivo: unità a dischi 1 stato: inattivo
dispositivo: unità a dischi 2 stato: inattivo
.....



walter.balzano@gmail.com

Accesso diretto alla memoria

- Tecnica usata con dispositivi di I/O veloci.
- **Il controllore trasferisce un intero blocco di dati dalla propria memoria di transito direttamente nella memoria centrale, o viceversa, senza alcun intervento da parte della CPU.**
- Il trasferimento richiede una sola interruzione per ogni blocco di dati trasferito, piuttosto che per ogni byte (o parola).



walter.balzano@gmail.com

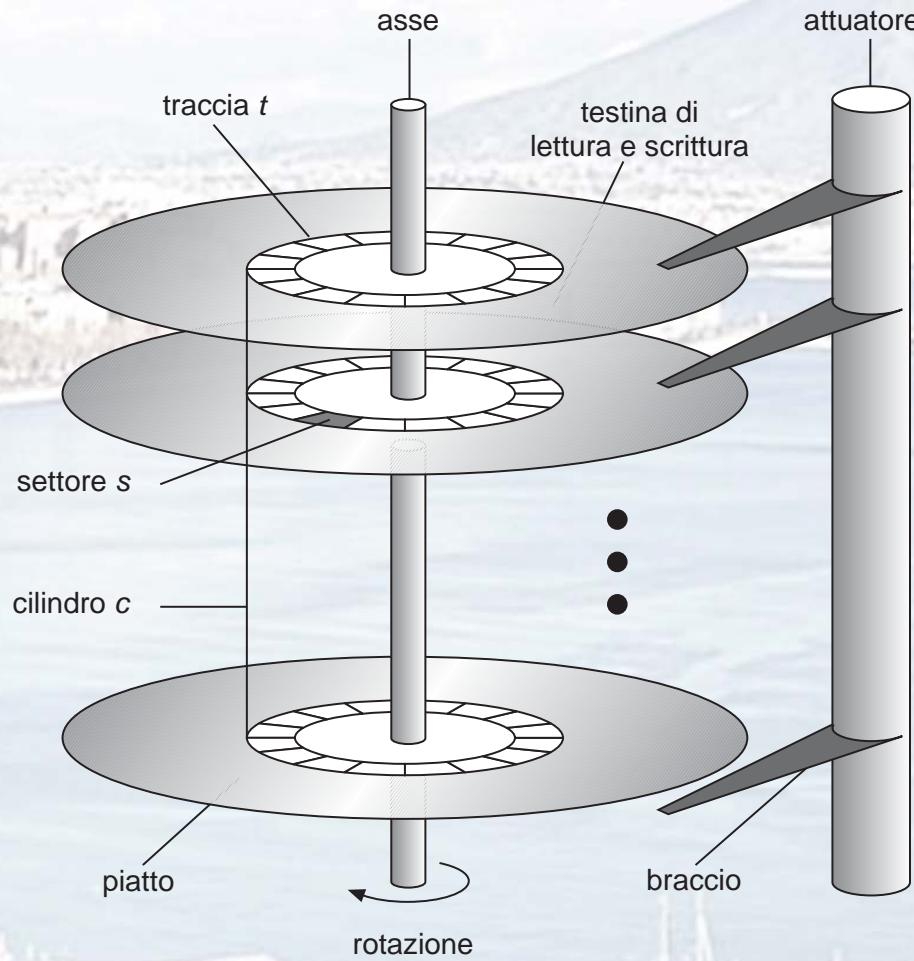
Struttura della memoria

- **Memoria centrale:** dispositivo di memoria direttamente accessibile dalla CPU.
- **Memoria secondaria:** estensione della memoria centrale capace di conservare in modo permanente grandi quantità di informazioni.
- **Disco magnetico:** piatto rigido di metallo o vetro ricoperto di materiale magnetico
 - La superficie del disco è divisa logicamente in *tracce circolari* a loro volta suddivise in *settori*.
 - I *controllori dei dischi* determinano l'interazione logica tra il dispositivo e il calcolatore.



walter.balzano@gmail.com

Schema funzionale di un disco



walter.balzano@gmail.com

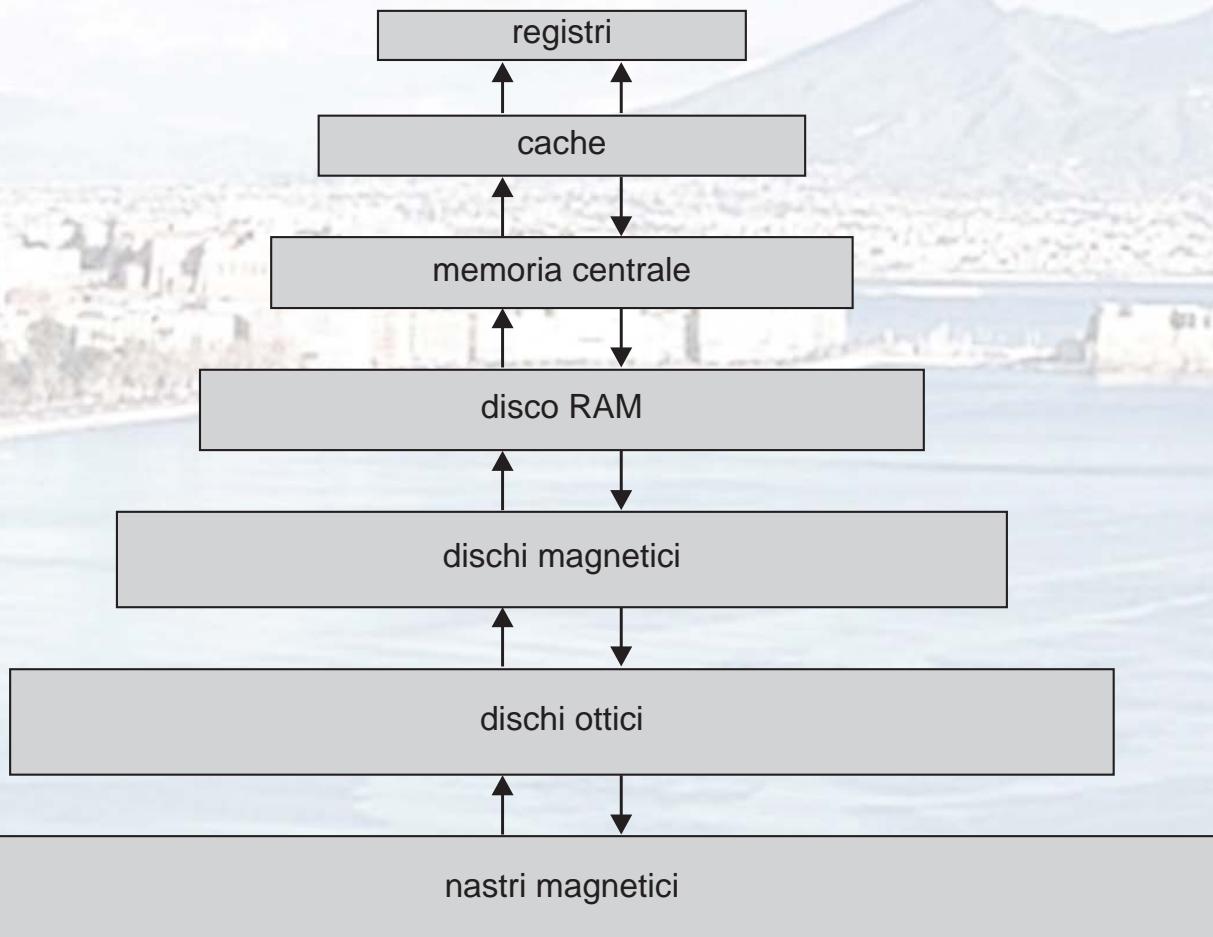
Gerarchia delle memorie

- I componenti di memoria di un sistema di calcolo possono essere organizzati in una struttura gerarchica in base a:
 - velocità
 - costo
 - volatilità
- Cache: copia temporanea di informazioni in un'unità più veloce; la memoria centrale si può considerare una cache per la memoria secondaria.



walter.balzano@gmail.com

Gerarchia dei dispositivi di memoria



walter.balzano@gmail.com

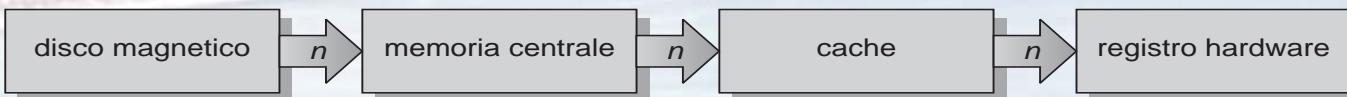
Cache

- Utilizzo di una memoria ad alta velocità per registrare dati ai quali si è avuto accesso recentemente (e che si prevede che presto serviranno ancora).
- Richiede una politica di **gestione della cache**.
- La cache introduce un altro livello nella gerarchia delle memorie. Ciò richiede che i dati che sono memorizzati contemporaneamente su più livelli siano *coerenti*.



walter.balzano@gmail.com

Migrazione di un intero n da un disco a un registro



walter.balzano@gmail.com

Architetture di protezione

- Duplice modo di funzionamento (*dual-mode*)
- Protezione dell'I/O
- Protezione della memoria
- Protezione della CPU



walter.balzano@gmail.com

Duplice modo di funzionamento

- La condivisione delle risorse di sistema rende necessario che il sistema operativo garantisca che un programma malfunzionante non causi una scorretta esecuzione di altri programmi.
- Specifiche caratteristiche dell'architettura di sistema consentono di gestire almeno due modi di funzionamento.
 1. **Modo d'utente (user mode)**: l'istruzione corrente si esegue per conto di un utente.
 2. **Modo di sistema (monitor mode, detto anche modo del supervisore, modo monitor o modo privilegiato)**: l'istruzione corrente si esegue per conto del sistema operativo.



walter.balzano@gmail.com

Duplice modo di funzionamento

1. **Modo d'utente (user mode):** l'istruzione corrente si esegue per conto di un utente.
2. **Modo di sistema (monitor mode, detto anche modo del supervisore, modo monitor o modo privilegiato):** l'istruzione corrente si esegue per conto del sistema operativo.



walter.balzano@gmail.com

Duplice modo di funzionamento (Cont.)

- Il **bit di modo** (*mode bit*) indica quale modo è attivo: di sistema (0) o d'utente (1).
- Ogni volta che si verifica un'interruzione o un'eccezione si passa dal modo d'utente al modo di sistema, cioè si pone a 0 il bit di modo.



La CPU consente l'esecuzione di **istruzioni privilegiate** (*privileged instruction*) soltanto nel modo di sistema.



walter.balzano@gmail.com

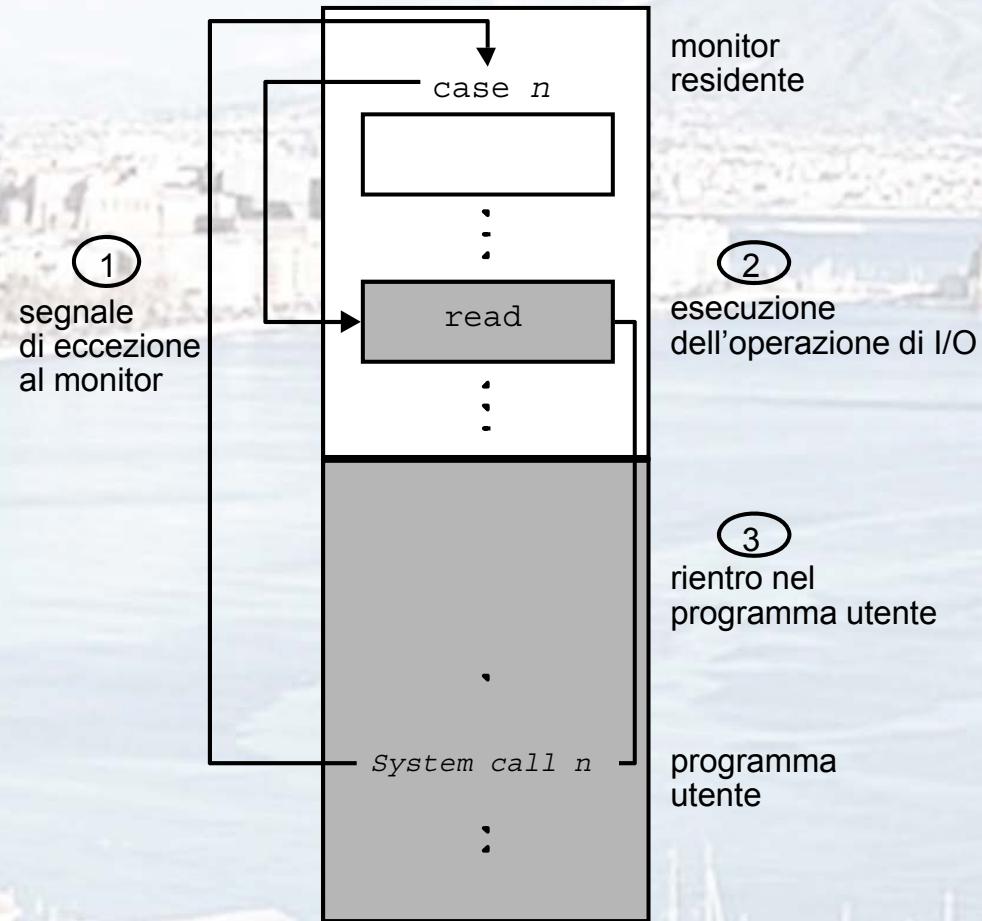
Protezione dell'I/O

- Tutte le istruzioni di I/O sono istruzioni privilegiate.
- Affinché la protezione dell'I/O sia totale, è necessario evitare che l'utente possa in qualsiasi modo ottenere il controllo del calcolatore quando questo si trova nel modo di sistema (es. un programma utente che, come parte della sua esecuzione, memorizza un nuovo indirizzo nel vettore delle interruzioni).



walter.balzano@gmail.com

Uso di una chiamata del sistema per l'esecuzione di una chiamata di I/O



walter.balzano@gmail.com

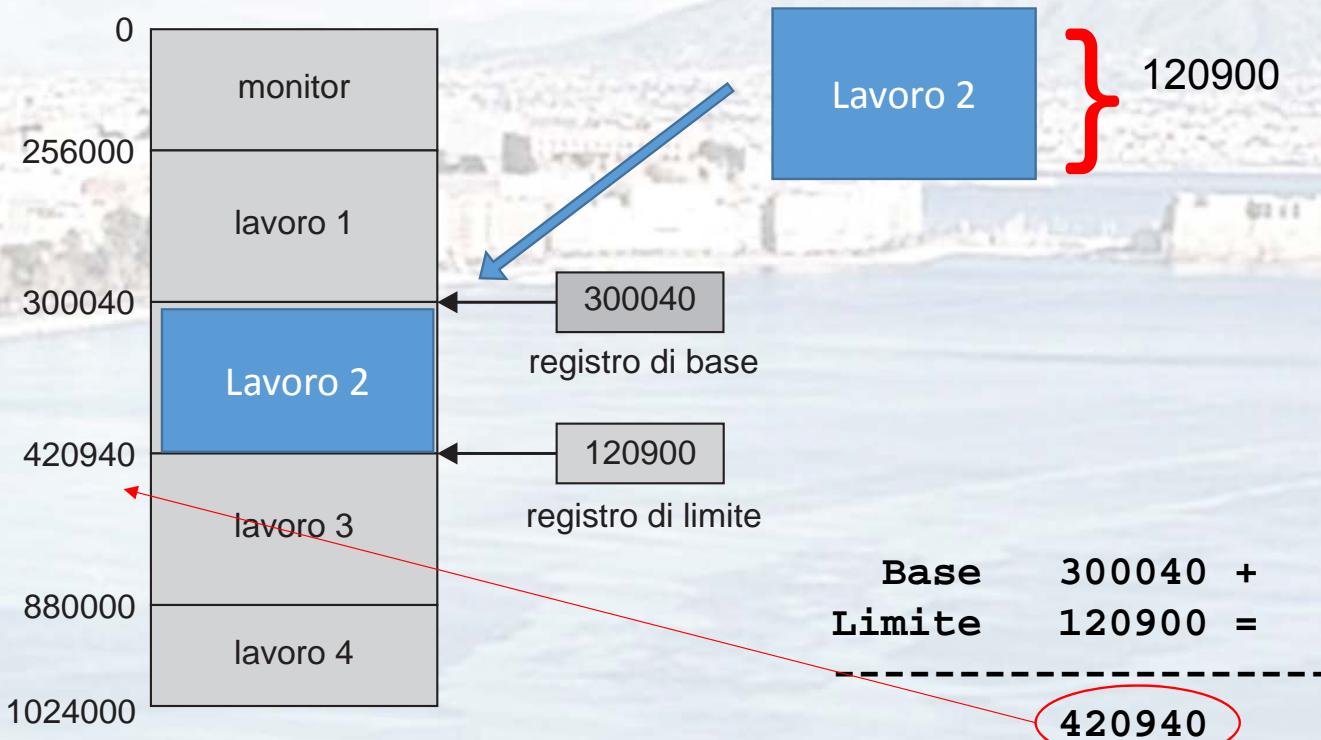
Protezione della memoria

- È necessario fornire protezione della memoria almeno per il vettore delle interruzioni e le relative procedure di servizio contenute nel codice del sistema operativo.
- Questo tipo di protezione si realizza impiegando due registri che contengono l'intervallo degli indirizzi validi cui un programma può accedere:
 - **Registro di base:** contiene il più basso indirizzo della memoria fisica al quale il programma dovrebbe accedere.
 - **Registro di limite:** contiene la dimensione dell'intervallo.
- Le aree di memoria al di fuori dell'intervallo stabilito sono protette.



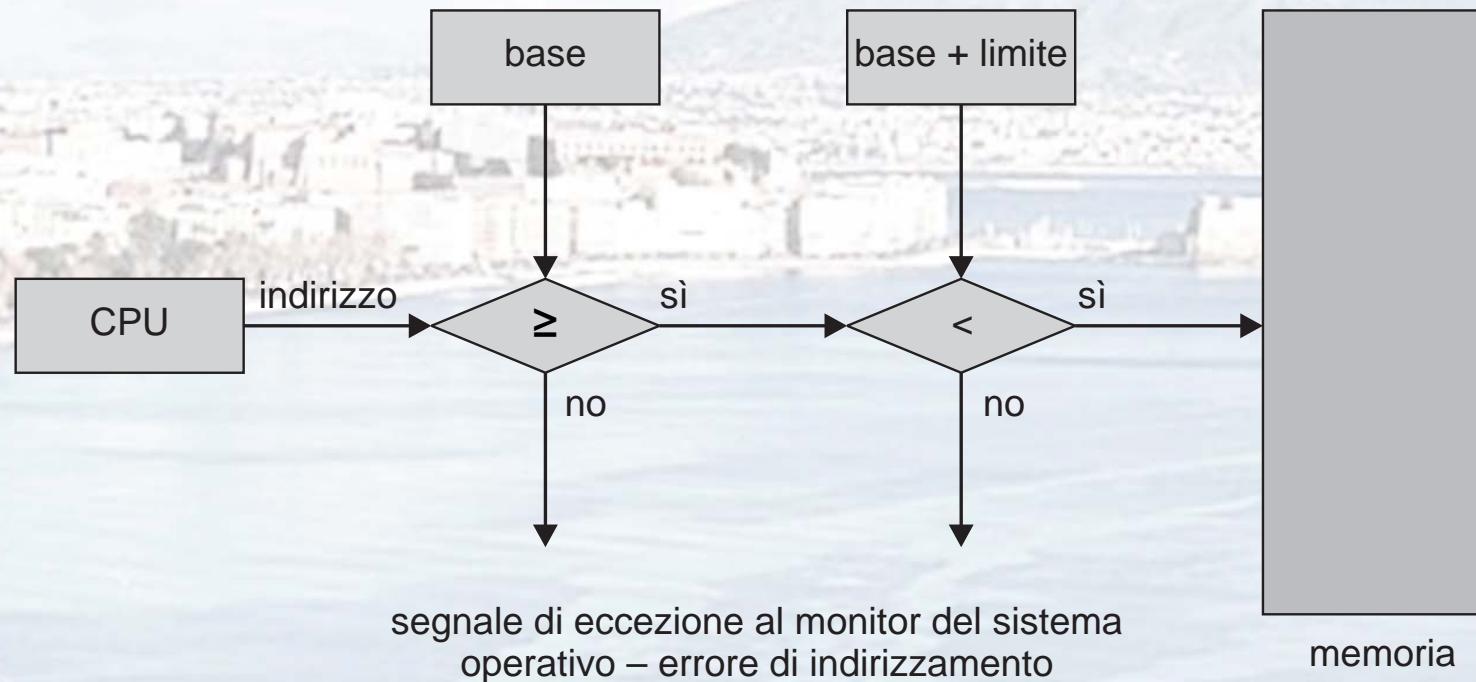
walter.balzano@gmail.com

Uso di un registro di base e un registro di limite



walter.balzano@gmail.com

Architettura di protezione degli indirizzi



walter.balzano@gmail.com

Protezione hardware

- **Funzionando nel modo di sistema, il sistema operativo ha la possibilità di accedere indiscriminatamente sia alla memoria a esso riservata sia a quella riservata agli utenti.**
- **Questo privilegio consente al sistema di caricare i programmi utenti nelle relative aree di memoria.**



walter.balzano@gmail.com

Protezione della CPU

- **Temporizzatore (timer)**: invia un segnale d'interruzione alla CPU a intervalli di tempo specificati per assicurare che il sistema operativo mantenga il controllo dell'elaborazione.
 - Il timer si decrementa a ogni impulso.
 - Quando raggiunge il valore 0, si genera un segnale d'interruzione.
- I temporizzatori si usano comunemente per realizzare la partizione del tempo d'elaborazione (*time sharing*).
- Un altro impiego dei temporizzatori è il calcolo dell'ora corrente.
- Load-timer è un'istruzione privilegiata.



walter.balzano@gmail.com

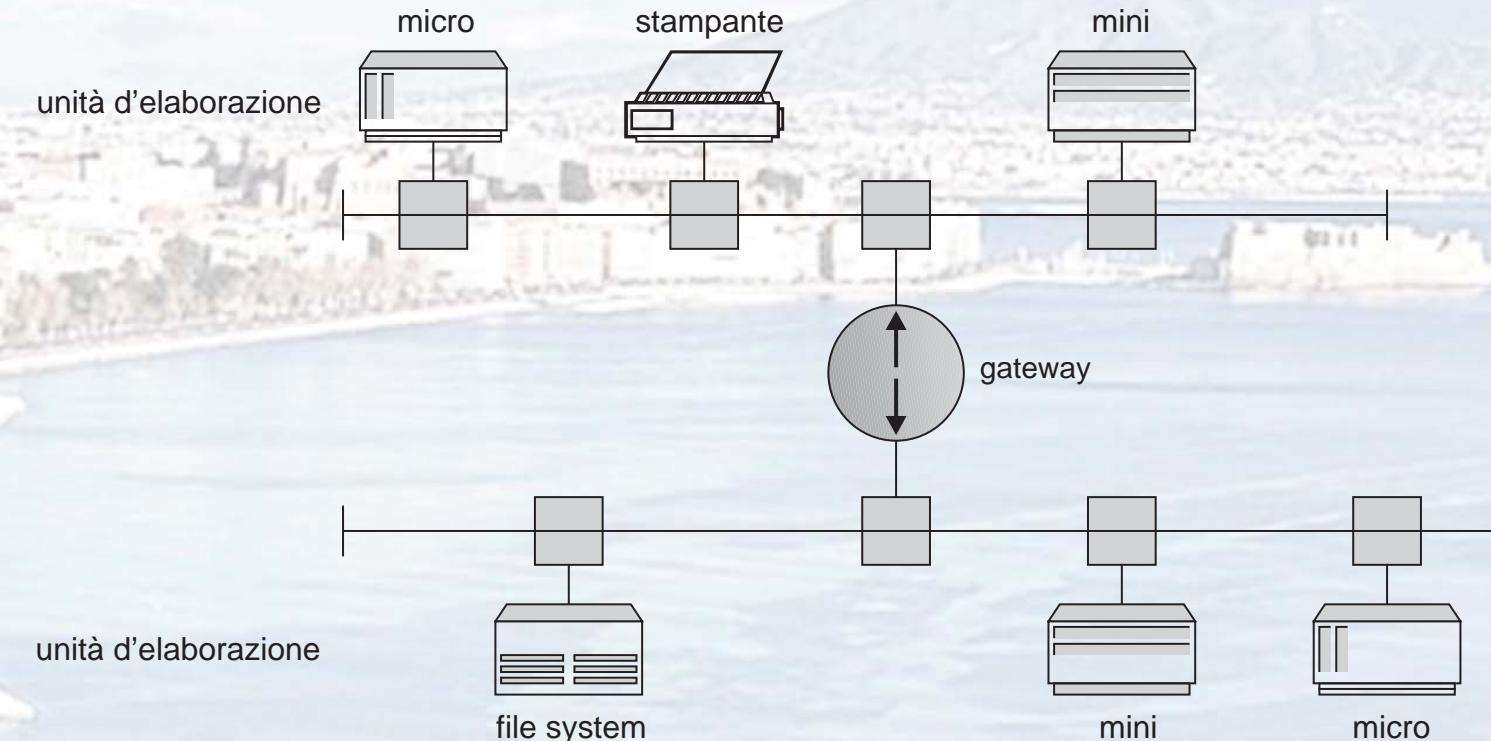
Struttura delle reti di calcolatori

- **Reti locali**
(*LAN, Local Area Networks*)
- **Reti geografiche**
(*WAN, Wide Area Networks*)



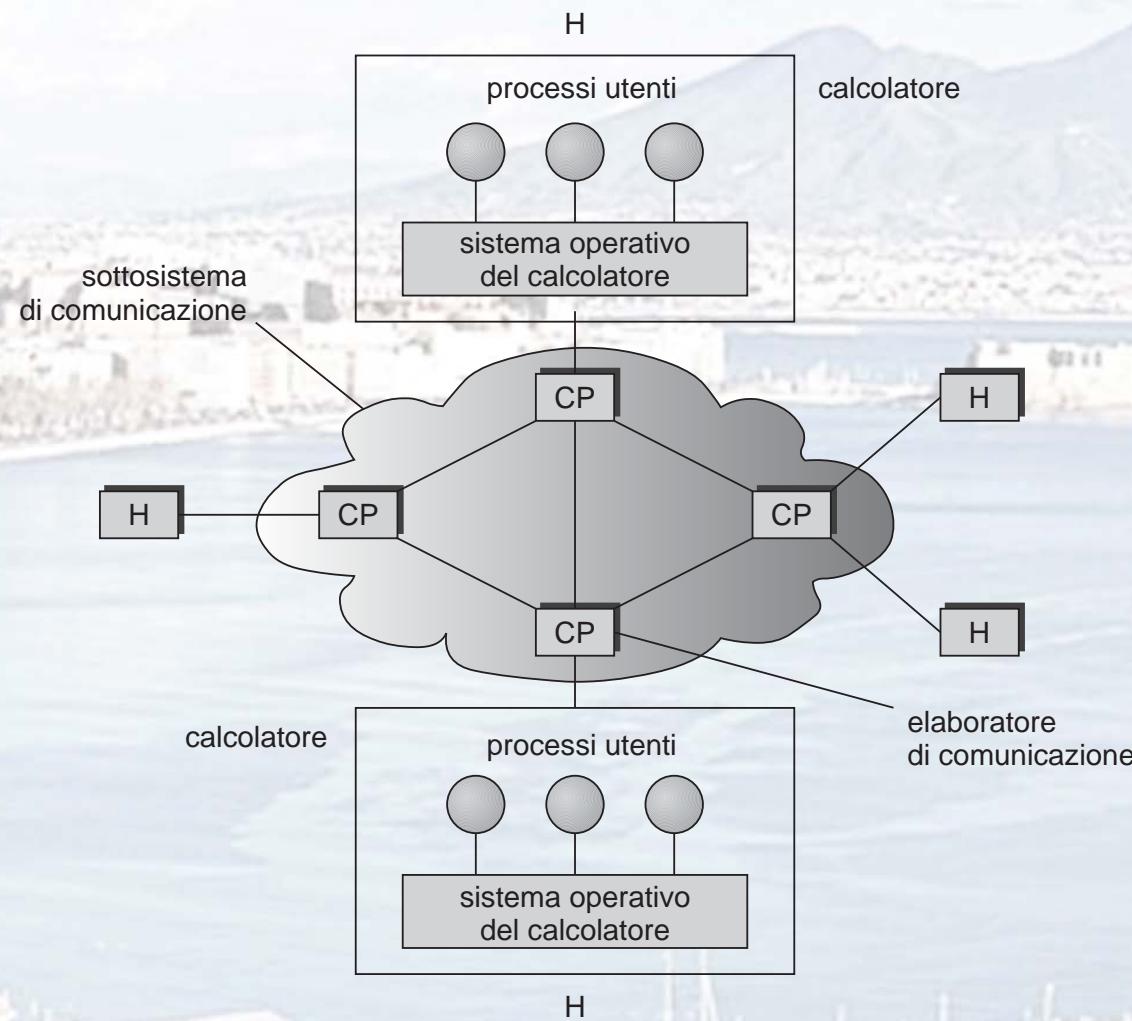
walter.balzano@gmail.com

Struttura di una rete locale



walter.balzano@gmail.com

Struttura di una rete geografica



walter.balzano@gmail.com

Capitolo 3: Strutture dei Sistemi Operativi

- **Componenti del sistema**
- **Servizi di un sistema operativo**
- **Chiamate del sistema**
- **Programmi di sistema**
- **Struttura del sistema**
- **Macchine virtuali**
- **Progettazione e realizzazione di un sistema**
- **Generazione di sistemi**



walter.balzano@gmail.com

Componenti del sistema

- Gestione dei processi
- Gestione della memoria centrale
- Gestione dei file
- Gestione del sistema di I/O
- Gestione della memoria secondaria
- Reti
- Sistema di protezione
- Interprete dei comandi



walter.balzano@gmail.com

Gestione dei processi

- **Un processo è un programma in esecuzione.** Per svolgere i propri compiti, un processo necessita di alcune risorse, tra cui tempo di CPU, memoria, file e dispositivi di I/O.
- Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei processi:
 - **creazione e cancellazione** dei processi utenti e di sistema
 - **sospensione e ripristino** dei processi
 - fornitura di meccanismi per:
 - **sincronizzazione dei processi**
 - **comunicazione tra processi**



walter.balzano@gmail.com

Gestione della memoria centrale

- **La memoria centrale è un vasto vettore** di dimensioni che variano tra le centinaia di migliaia e i miliardi di parole, ciascuna delle quali è dotata del proprio **indirizzo**. È un magazzino di dati velocemente accessibile ed è condivisa dalla CPU e da alcuni dispositivi di I/O.
- **La memoria centrale è volatile**, e perde le informazioni in caso di guasto del sistema.
- Il sistema operativo è responsabile delle seguenti attività connesse alla gestione della memoria centrale:
 - **tenere traccia di quali parti della memoria sono attualmente usate e da che cosa;**
 - **decidere quali processi si debbano caricare nella memoria quando vi sia spazio disponibile;**
 - **assegnare e revocare lo spazio di memoria secondo le necessità.**



walter.balzano@gmail.com

Gestione dei file

- Un file è una raccolta di informazioni correlate definite dal loro creatore. Comunemente, i file rappresentano programmi (sia sorgente sia oggetto) e dati.
- Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei file:
 - creazione e cancellazione di file;
 - creazione e cancellazione di directory;
 - fornitura delle funzioni fondamentali per la gestione di file e directory;
 - associazione dei file ai dispositivi di memoria secondaria;
 - creazione di copie di riserva (*backup*) dei file su dispositivi di memorizzazione non volatili.



walter.balzano@gmail.com

Gestione del sistema di I/O

- Il sistema di I/O è composto dalle seguenti parti:

- un **sistema buffer-caching**
- un'interfaccia generale per i driver dei dispositivi
- i driver per gli specifici dispositivi



walter.balzano@gmail.com

Gestione della memoria secondaria

- Poiché la memoria centrale è troppo piccola per contenere tutti i dati e tutti i programmi, e il suo contenuto va perduto se il sistema si spegne, il calcolatore deve disporre di una **memoria secondaria** a sostegno della memoria centrale.
- La maggior parte dei moderni sistemi di calcolo impiega i dischi come principale mezzo di memorizzazione secondaria, sia per i programmi sia per i dati.
- Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei dischi:
 - gestione dello spazio libero
 - assegnazione dello spazio
 - scheduling del disco



walter.balzano@gmail.com

Reti (sistemi distribuiti)

- Un sistema distribuito è un insieme di unità di elaborazione che **NON** condividono la memoria, i dispositivi periferici o un clock; ciascun processore dispone di una propria memoria locale e di un suo clock.
- Le unità d'elaborazione sono collegate da una rete di comunicazione.
- La comunicazione avviene utilizzando un **protocollo**.
- Un sistema distribuito offre all'utente l'accesso alle varie risorse di sistema.
- L'accesso a una risorsa condivisa permette di:
 - accelerare il calcolo
 - aumentare la disponibilità dei dati
 - incrementare l'affidabilità



walter.balzano@gmail.com

Sistema di protezione

- La **protezione** è definita da ogni meccanismo che controlla l'accesso da parte di programmi, processi o utenti alle risorse di un sistema di calcolo.
- Il meccanismo di protezione deve:
 - distinguere tra uso autorizzato e non autorizzato.
 - specificare i controlli che devono essere attivati.
 - fornire strumenti di miglioramento dell'affidabilità.



walter.balzano@gmail.com

Interprete dei comandi

- Molti comandi si impartiscono al sistema operativo attraverso **istruzioni di controllo** che riguardano:

- creazione e gestione di processi
- I/O
- gestione della memoria secondaria
- gestione della memoria centrale
- accesso al file-system
- protezione
- reti



walter.balzano@gmail.com

Interprete dei comandi (Cont.)

- Il programma che legge e interpreta le istruzioni di controllo ha diversi nomi:
 - interprete di schede di controllo
(control-card interpreter)
 - interprete di riga di comando
(command-line interpreter)
 - **shell** (nello UNIX)

La sua funzione consiste nel prelevare ed eseguire la successiva istruzione di comando.



walter.balzano@gmail.com

Servizi di un sistema operativo

- **Esecuzione di un programma:** capacità del sistema di caricare un programma nella memoria ed eseguirlo.
- **Operazioni di I/O:** poiché i programmi utenti non possono eseguire direttamente operazioni di I/O, il sistema operativo deve offrire mezzi adeguati.
- **Gestione del file system:** capacità del programma di leggere, scrivere, creare e cancellare file.
- **Comunicazioni:** scambio di informazioni tra processi in esecuzione nello stesso calcolatore e tra processi in esecuzione in calcolatori diversi collegati per mezzo di una rete. La comunicazione si può realizzare tramite una *memoria condivisa* o attraverso lo *scambio di messaggi*.
- **Rilevamento d'errori:** capacità di rilevare eventuali errori che possono verificarsi nella CPU e nei dispositivi di memoria, nei dispositivi di I/O e nei programmi utenti.



walter.balzano@gmail.com

Funzioni addizionali di un sistema operativo

Esiste anche un'altra serie di funzioni del sistema operativo che non riguarda direttamente gli utenti, ma assicura il funzionamento efficiente del sistema stesso.

- **Assegnazione delle risorse:** se sono in corso più sessioni di lavoro di utenti o sono contemporaneamente in esecuzione più processi, il sistema operativo provvede all'assegnazione delle risorse necessarie a ciascuno di essi.
- **Contabilizzazione dell'uso delle risorse:** registrazione degli utenti che usano il calcolatore, con segnalazione di quali e quante risorse vengono impiegate, a fini di addebito dei costi o di preparazione di statistiche.
- **Protezione:** assicura che l'accesso alle risorse del sistema sia controllato.



walter.balzano@gmail.com

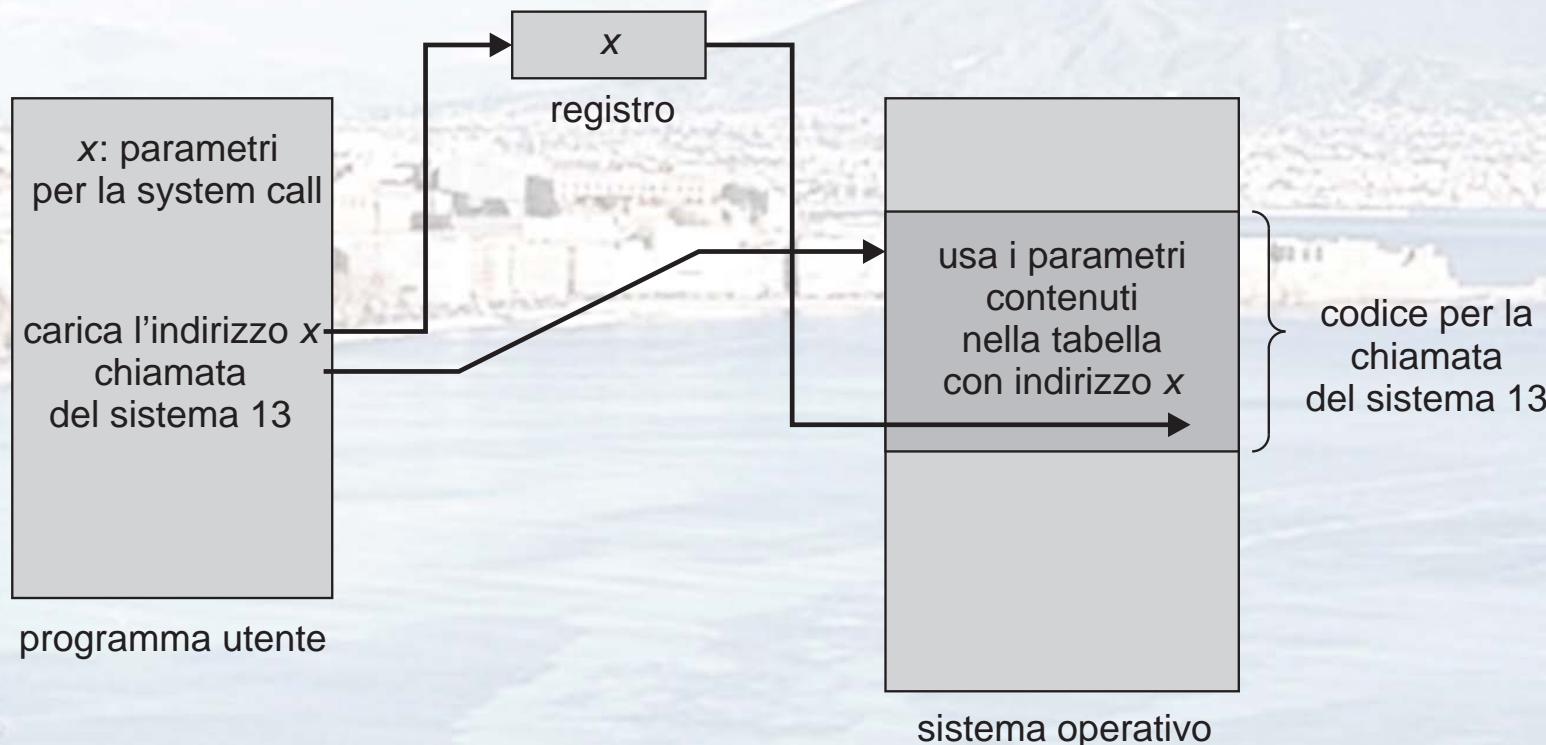
Chiamate del sistema

- **Le chiamate del sistema costituiscono l'interfaccia tra un processo e il sistema operativo.**
 - Generalmente disponibili in forma di istruzioni in linguaggio assemblativo.
 - Certi sistemi consentono che le chiamate del sistema siano invocate direttamente da un programma scritto in un linguaggio di alto livello (es.: C, C++, Perl)
- **Per passare parametri al sistema operativo si usano tre metodi generali.**
 - Passare i parametri in *registri*.
 - Memorizzare i parametri in un *blocco* o tabella di memoria e **passare l'indirizzo del blocco**, in forma di parametro, in un registro.
 - *Collocare (push)* i parametri in una pila da cui sono prelevati (*pop*) dal sistema operativo.



walter.balzano@gmail.com

Passaggio di parametri in forma di tabella



walter.balzano@gmail.com

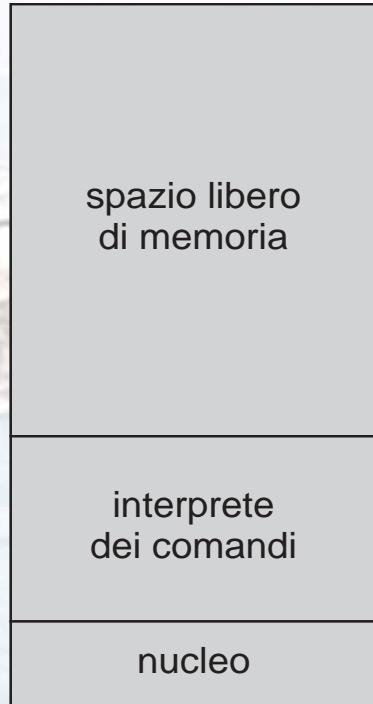
Tipi di chiamate del sistema

- Controllo dei **processi**
- Gestione dei **file**
- Gestione dei **dispositivi**
- Gestione delle **informazioni**
- **Comunicazione**



walter.balzano@gmail.com

Esecuzione nell'MS-DOS



(a)

All'avviamento del sistema



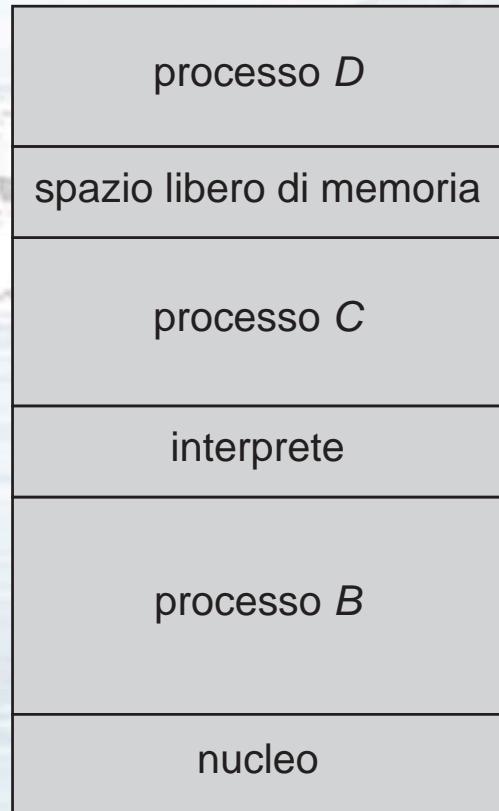
(b)

Durante l'esecuzione di un programma



walter.balzano@gmail.com

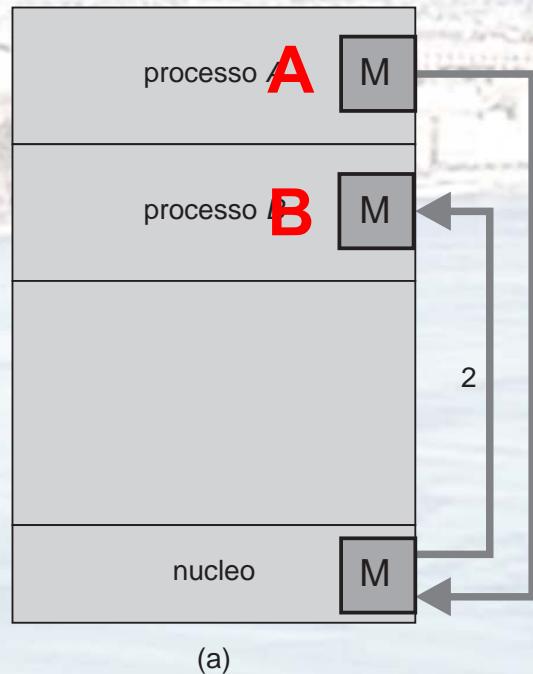
Esecuzione di più programmi nel sistema operativo UNIX



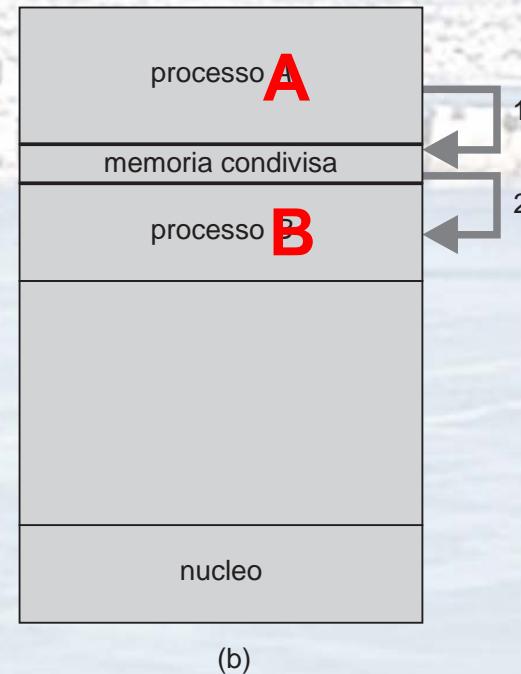
walter.balzano@gmail.com

Modelli di comunicazione

- La comunicazione può avvenire o attraverso il modello a **scambio di messaggi** o mediante il modello a **memoria condivisa**.



Scambio di messaggi



Memoria condivisa



walter.balzano@gmail.com

Programmi di sistema

- I programmi di sistema offrono un ambiente conveniente per lo sviluppo e l'esecuzione dei programmi; in generale si possono classificare nelle seguenti categorie:
 - Gestione dei file
 - Informazioni di stato
 - Modifica dei file
 - Ambienti d'ausilio alla programmazione
 - Caricamento ed esecuzione dei programmi
 - Comunicazioni
 - Programmi d'applicazione
- Per la maggior parte degli utenti, l'**interfaccia col sistema operativo è definita dai programmi di sistema** piuttosto che dalle effettive chiamate di sistema.



walter.balzano@gmail.com

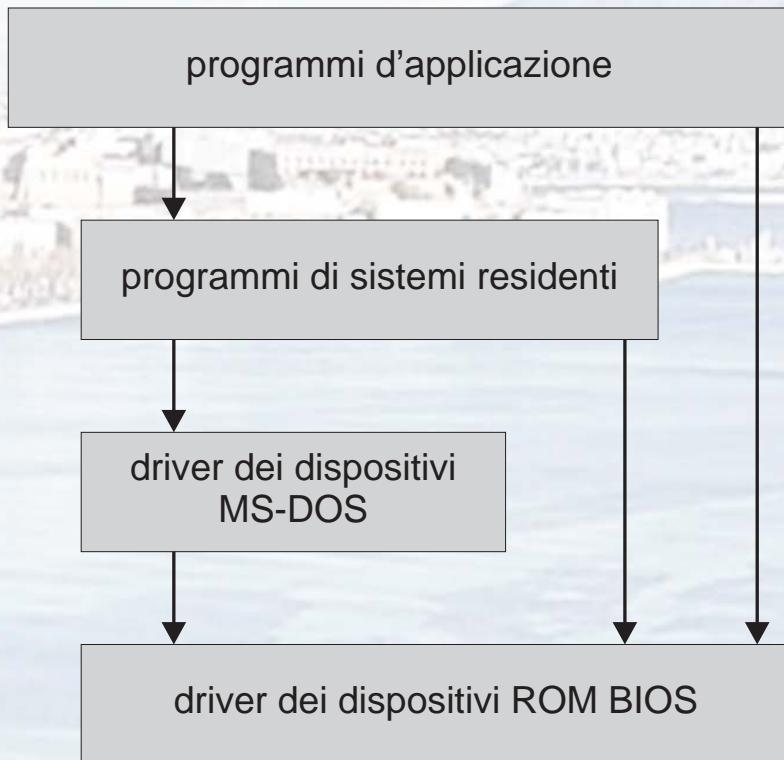
Struttura del sistema MS-DOS

- **MS-DOS: progettato per fornire la massima funzionalità nel minimo spazio**
 - non suddiviso in moduli
 - anche se dotato di una semplice struttura, le sue interfacce e i livelli di funzionalità non sono ben separati.



walter.balzano@gmail.com

Struttura degli strati dell'MS-DOS



walter.balzano@gmail.com

Struttura del sistema UNIX

- Lo UNIX è un altro esempio di strutturazione che inizialmente era limitata dalle funzioni dell'architettura sottostante. E' formato da due parti:

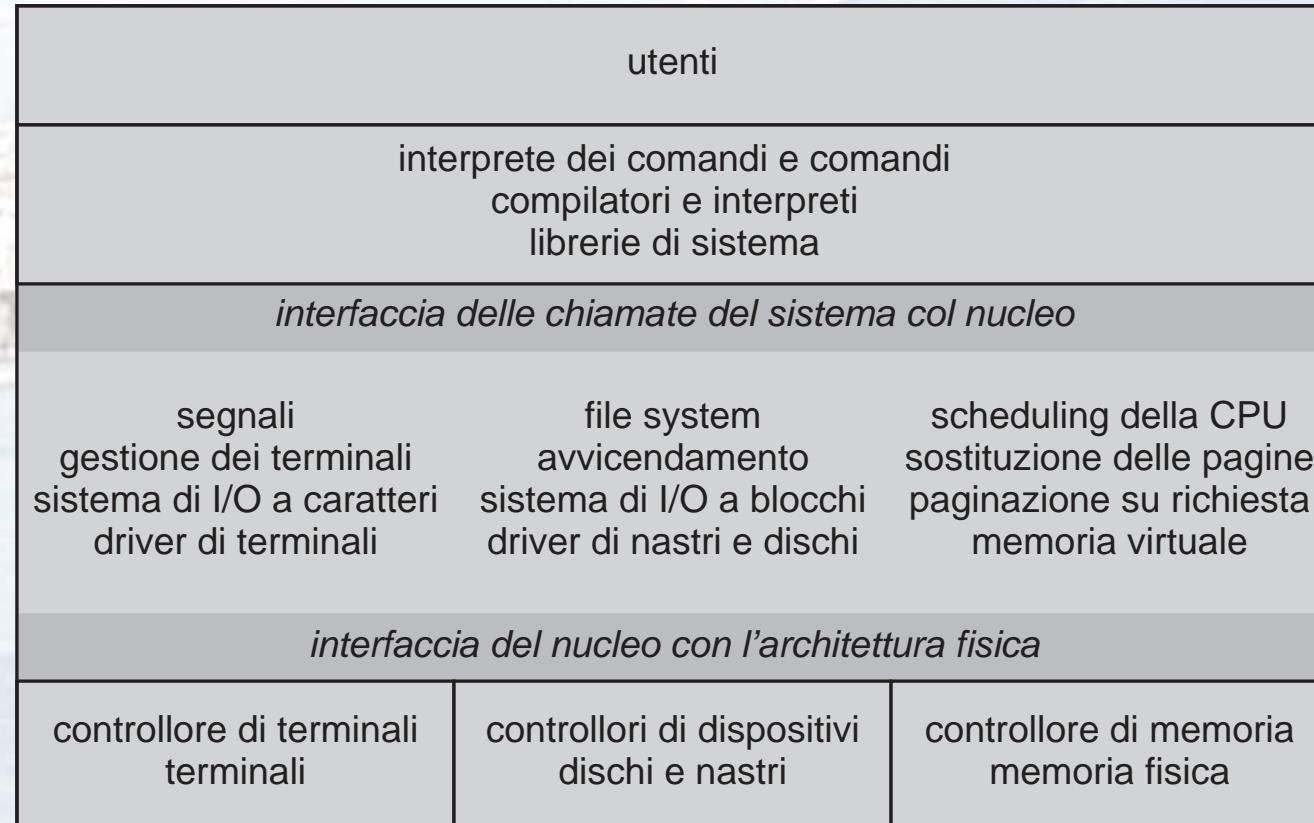
- Programmi di sistema
- Nucleo:

- tutto quello che si trova sotto l'interfaccia delle chiamate di sistema e sopra i dispositivi fisici è il nucleo
- fornisce il file system, lo scheduling della CPU, la gestione della memoria e altre funzioni riguardanti il sistema operativo: in un solo livello sono combinate un'enorme quantità di funzioni.



walter.balzano@gmail.com

Struttura del sistema UNIX



walter.balzano@gmail.com

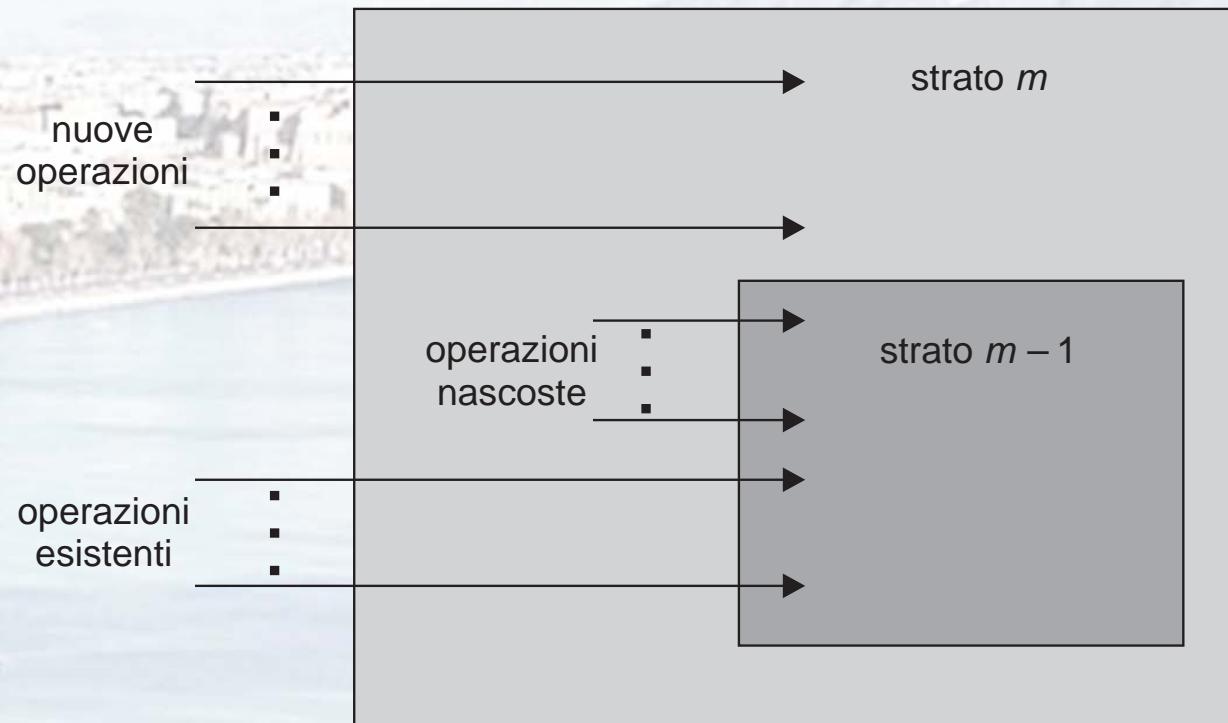
Metodo stratificato

- Con il metodo stratificato si suddivide il sistema operativo in un certo numero di strati (o livelli), ciascuno costruito sopra gli strati inferiori. Lo strato più basso (0) è lo strato fisico; quello più alto (strato n) è l'interfaccia d'utente.
- Ogni strato si realizza impiegando unicamente le operazioni messe a disposizione dagli strati inferiori.



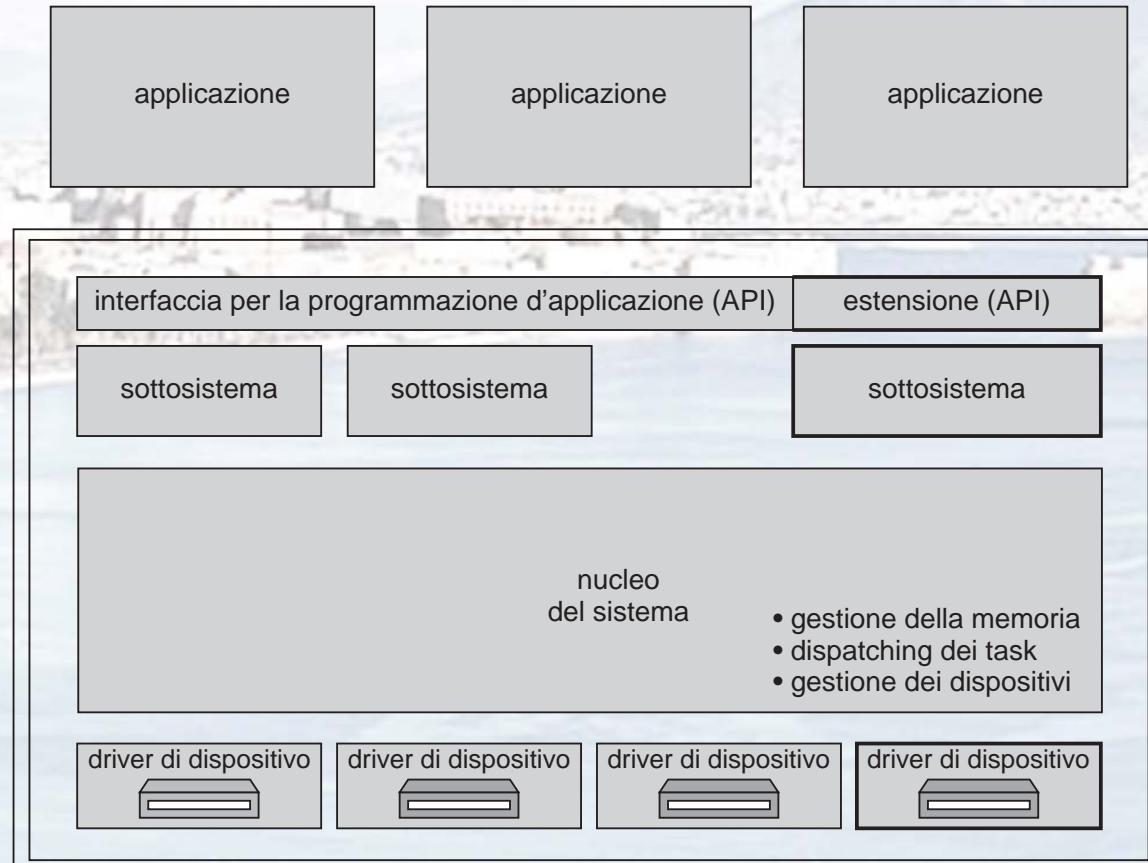
walter.balzano@gmail.com

Uno strato di sistema operativo



walter.balzano@gmail.com

Struttura stratificata dell'OS/2



walter.balzano@gmail.com

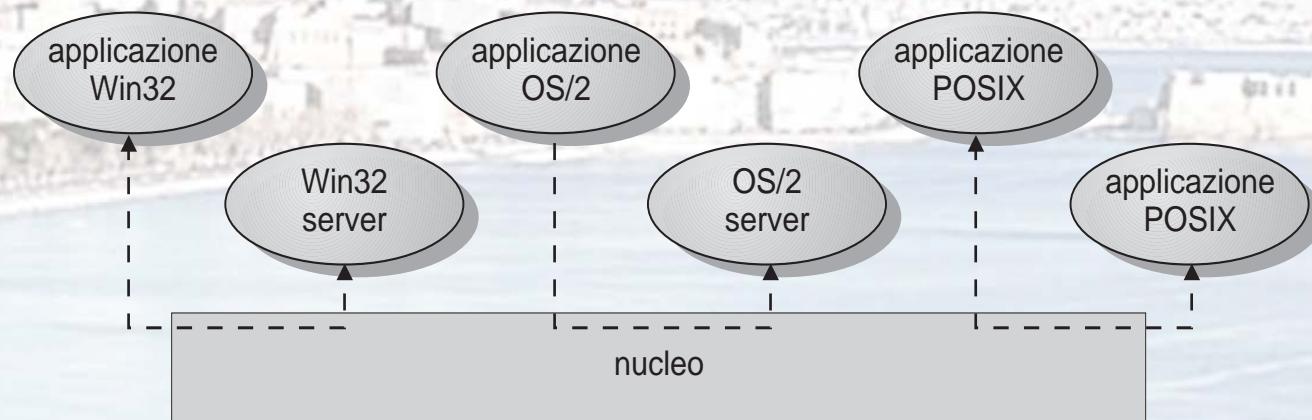
Microkernel - Orientamento a micronucleo

- **Secondo questo orientamento, si progetta il sistema operativo rimuovendo dal nucleo tutti i componenti non essenziali, realizzandoli come programmi del livello d'utente e di sistema.**
- La comunicazione si realizza secondo il modello a scambio di messaggi.
- Vantaggi:
 - facilità di estensione del sistema operativo
 - più semplice da adattare alle diverse architetture
 - più affidabile (i servizi si eseguono in gran parte come processi utenti, non come processi del nucleo)
 - più sicuro



walter.balzano@gmail.com

Struttura client-server del sistema operativo Windows NT



walter.balzano@gmail.com

Macchine virtuali

- Il concetto di **macchina virtuale** si sviluppa logicamente dal metodo stratificato. I programmi d'applicazione possono considerare quello che si trova a un livello gerarchico inferiore come se fosse parte della macchina stessa, anche se i programmi di sistema si trovano a un livello superiore.
- Una macchina virtuale è un'interfaccia *identica* all'architettura sottostante.
- Il sistema operativo crea l'illusione che un processo disponga della propria CPU con la propria memoria (**virtuale**).



walter.balzano@gmail.com

Macchine virtuali (Cont.)

Il calcolatore fisico condivide le risorse in modo da creare macchine virtuali.

- La partizione del tempo d'uso della CPU si può usare sia per condividere la CPU sia per dare l'illusione che gli utenti dispongano di una propria CPU.
- La gestione asincrona delle operazioni di I/O e dell'esecuzione di più processi, unita a un file system, consente di creare lettori di schede e stampanti virtuali.
- Un normale terminale di un sistema a partizione del tempo funziona da console d'operatore della macchina virtuale.



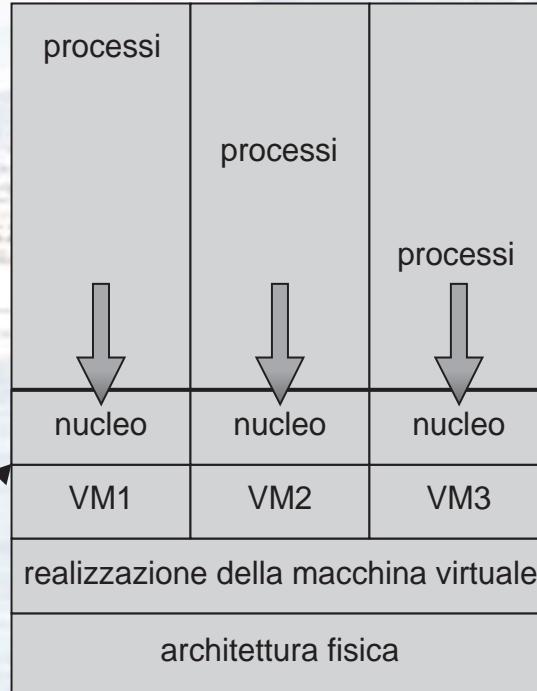
walter.balzano@gmail.com

Modelli di sistema



(a)

Semplice



(b)

Macchina virtuale



walter.balzano@gmail.com

Vantaggi/svantaggi delle macchine virtuali

- L'uso delle macchine virtuali protegge completamente le risorse di sistema poiché ciascuna macchina virtuale è isolata dalle altre. Uno svantaggio di questo tipo d'ambiente è che non c'è una condivisione *diretta* delle risorse.
- Un sistema di macchine virtuali è un perfetto mezzo di ricerca e sviluppo dei sistemi operativi. Lo sviluppo avviene sulla macchina virtuale non sulla macchina fisica, evitando così modifiche che potrebbero causare oscuri errori di programmazione in altri punti.
- A una maggiore complessità della macchina da emulare corrisponde una maggiore difficoltà di realizzazione di un'accurata macchina virtuale, e una maggiore lentezza nell'esecuzione.



walter.balzano@gmail.com

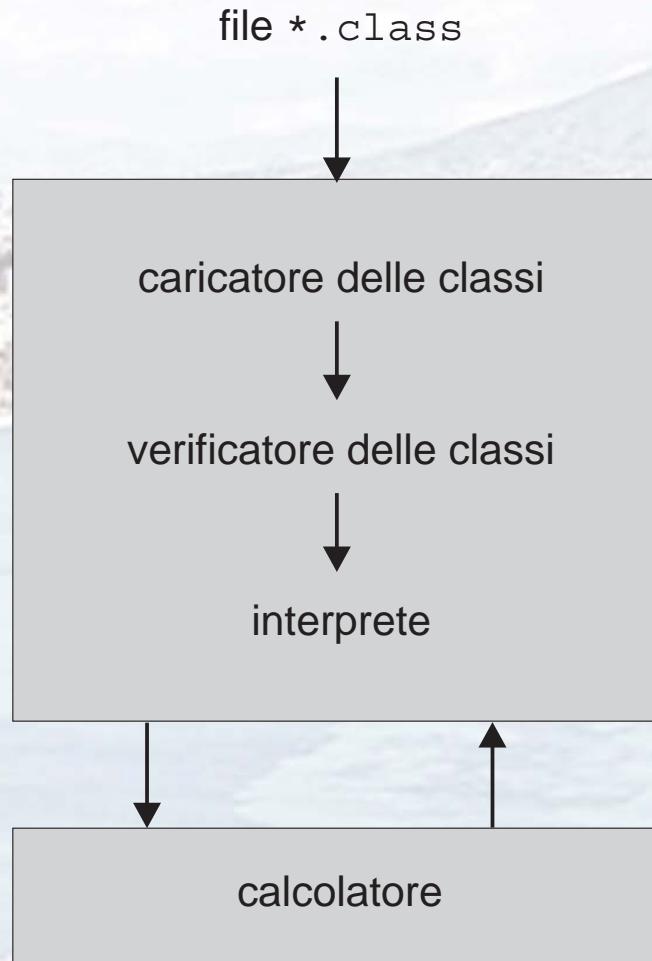
Macchina virtuale Java

- I programmi Java sono **bytecode** indipendente dall'architettura sottostante eseguiti dalla macchina virtuale Java (**JVM, Java Virtual Machine**).
- La JVM consiste di:
 - un caricatore delle classi
 - un verificatore delle classi
 - un interprete del linguaggio che esegue il **bytecode**
- Il compilatore istantaneo Just-In-Time (JIT) ne migliora le prestazioni



walter.balzano@gmail.com

Macchina virtuale dell'ambiente Java



walter.balzano@gmail.com

Scopi della progettazione

- Scopi degli **utenti**: gli utenti desiderano che un sistema sia utile, facile da imparare e usare, affidabile, sicuro ed efficiente.
- Scopi del **sistema**: il sistema operativo deve essere di facile progettazione, realizzazione e manutenzione; deve essere flessibile, affidabile, senza errori ed efficiente.



walter.balzano@gmail.com

Meccanismi e criteri

- I **meccanismi** determinano *come eseguire* qualcosa; i **criteri** invece stabiliscono *cosa si debba fare*.
- La distinzione tra meccanismi e criteri è molto importante, e consente la massima flessibilità, poiché i criteri sono soggetti a cambiamenti rispetto alle situazioni o ai momenti.



walter.balzano@gmail.com

Realizzazione

- Tradizionalmente, i sistemi operativi si scrivevano in un **linguaggio assemblativo**; attualmente si scrivono spesso in linguaggi di alto livello come il **C** o il **C++**.
- Il codice in un linguaggio di alto livello:
 - può essere scritto più rapidamente
 - è più compatto
 - è più facile da capire e mettere a punto.
- Un sistema operativo scritto in un linguaggio di alto livello è più facile da adattare a un'altra architettura (*porting*).



walter.balzano@gmail.com

Generazione di sistemi (SYSGEN)

- I sistemi operativi sono progettati per un impiego su macchine di una stessa classe con configurazioni diverse.
- Il processo di generazione di sistemi (**SYSGEN**) configura o genera il sistema per ciascuna situazione specifica.
- Booting: avviamento di un calcolatore attraverso il caricamento del nucleo.
- Bootstrap program: piccolo segmento di codice memorizzato in una ROM che individua il nucleo, lo carica nella memoria e ne avvia l'esecuzione.



walter.balzano@gmail.com

Capitolo 4: Processi

- Concetto di processo
- Scheduling dei processi
- Operazioni sui processi
- Processi cooperanti
- Comunicazione tra processi
- Comunicazione nei sistemi client-server



walter.balzano@gmail.com

Concetto di processo

- Un sistema operativo esegue differenti programmi:
 - Un **sistema a lotti** (*batch system*) esegue **lavori** (*job*)
 - Un **sistema a partizione del tempo** (*time-shared system*) esegue **programmi utenti** o **task**
- Il vostro libro di testo utilizza i termini *lavoro* e *processo* in modo quasi intercambiabile.
- **Processo:** un programma in esecuzione;
l'esecuzione del processo deve avvenire in modo sequenziale.
- **Un processo comprende:**
 - contatore di programma (*program counter*)
 - pila (*stack*)
 - sezione di dati (*data section*)



walter.balzano@gmail.com

Stato del processo

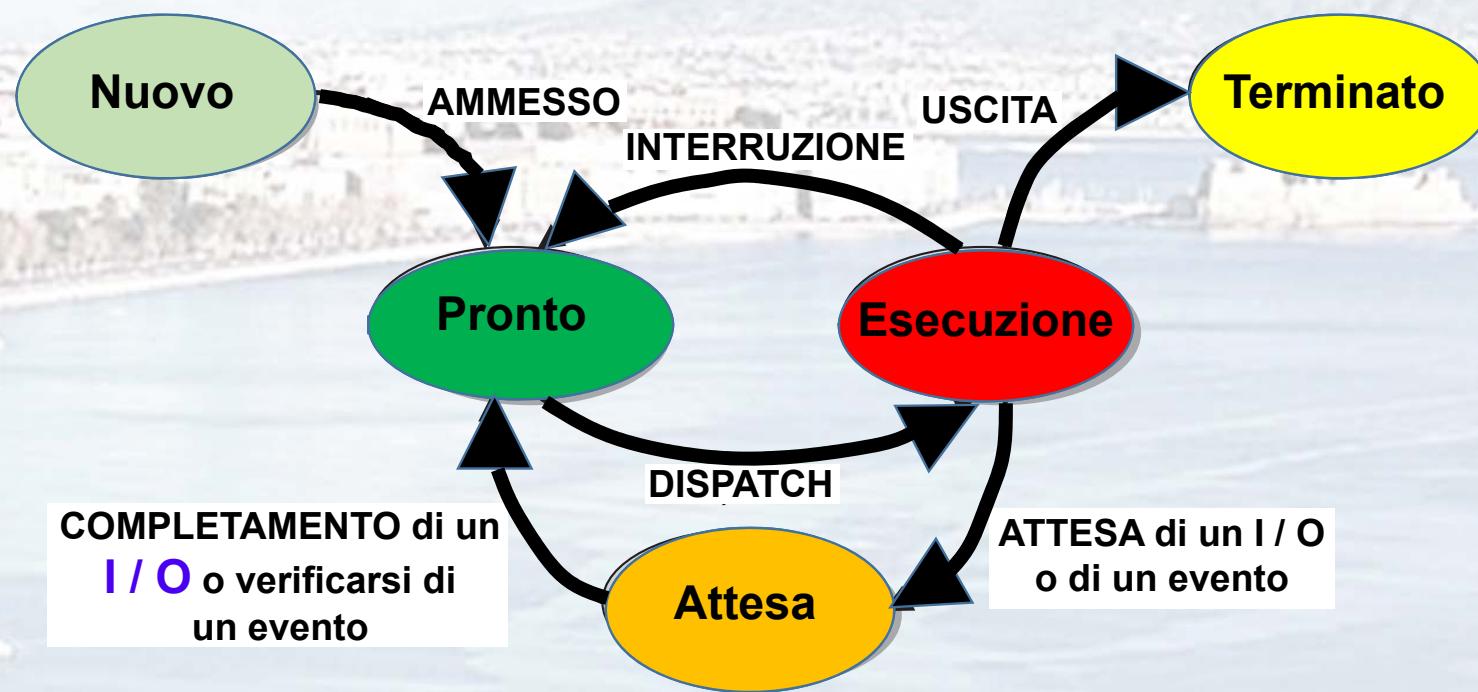
Mentre un processo è in esecuzione,
è soggetto a cambiamenti di **stato**:

- **NUOVO**: si crea il processo.
- **ESECUZIONE**: le istruzioni vengono eseguite.
- **ATTESA**: il processo attende che si verifichi qualche evento.
- **PRONTO**: il processo attende di essere assegnato a un'unità d'elaborazione.
- **TERMINATO**: il processo ha terminato l'esecuzione.



walter.balzano@gmail.com

Diagramma di transizione degli stati di un processo



walter.balzano@gmail.com

Blocco di controllo di un processo (PCB)

Informazioni connesse a un processo specifico.

- **Stato del processo**
- **Contatore di programma**
- **Registri di CPU**
- **Informazioni sullo scheduling di CPU**
- **Informazioni sulla gestione della memoria**
- **Informazioni di contabilizzazione delle risorse**
- **Informazioni sullo stato dell'I/O**



walter.balzano@gmail.com

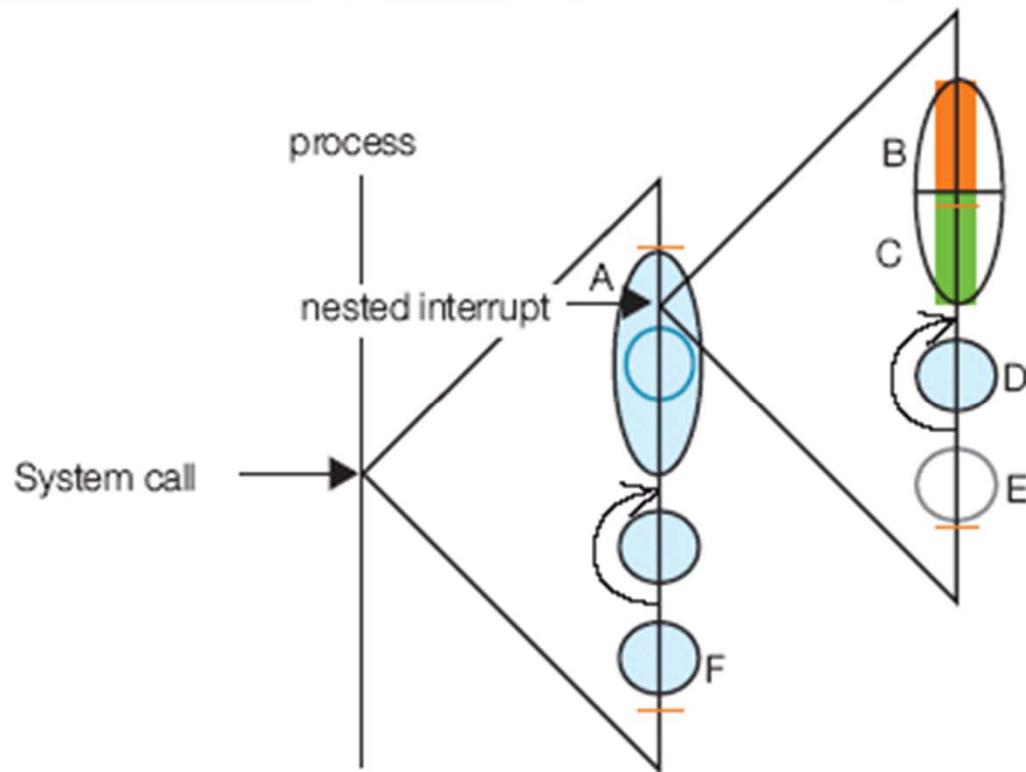
Process Control Block (PCB)

puntatore	stato del processo
numero del processo	
contatore di programma	
registri	
limiti di memoria	
elenco dei file aperti	
:	



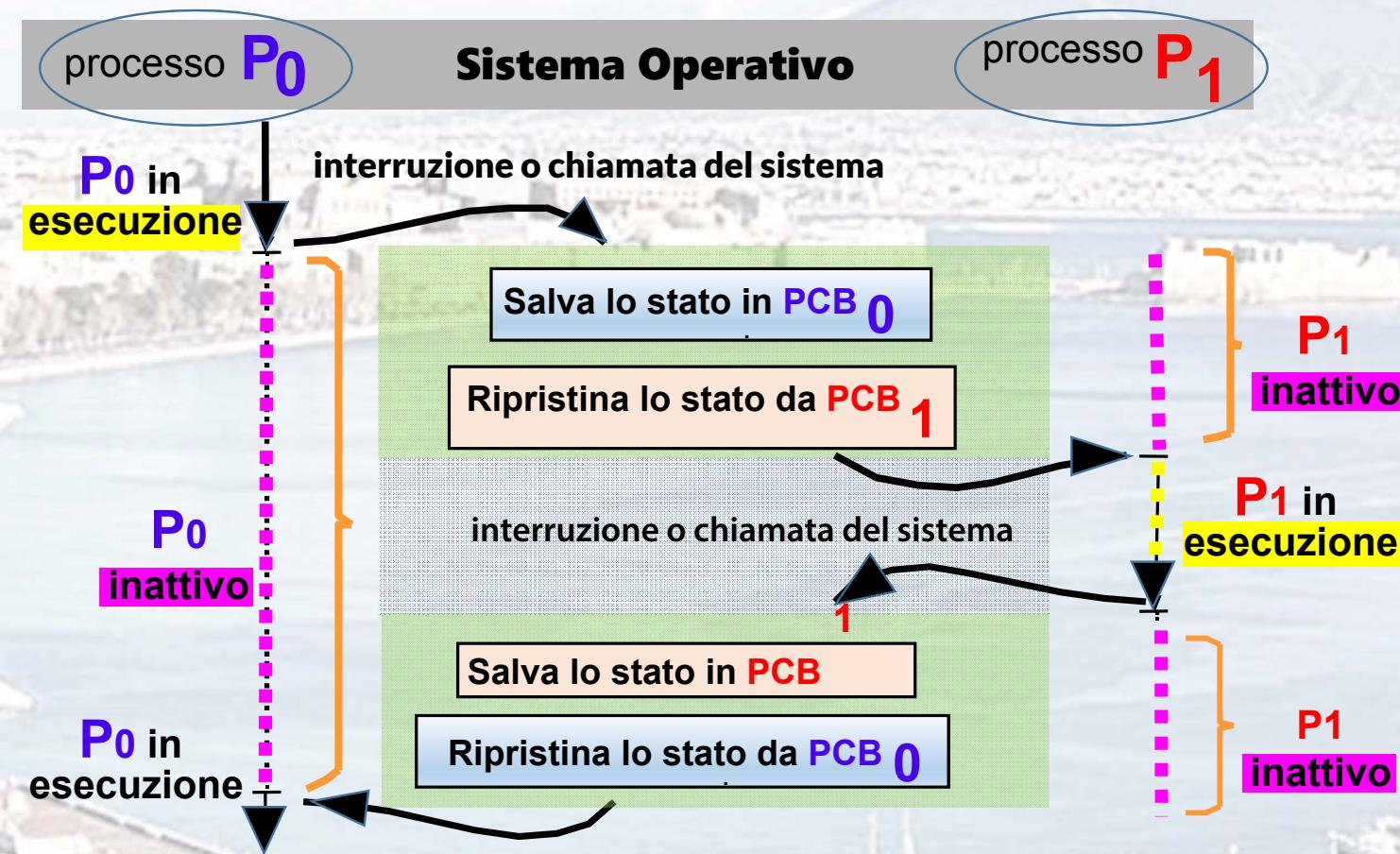
walter.balzano@gmail.com

La CPU può essere commutate tra processi



walter.balzano@gmail.com

La CPU può essere commutate tra processi



walter.balzano@gmail.com

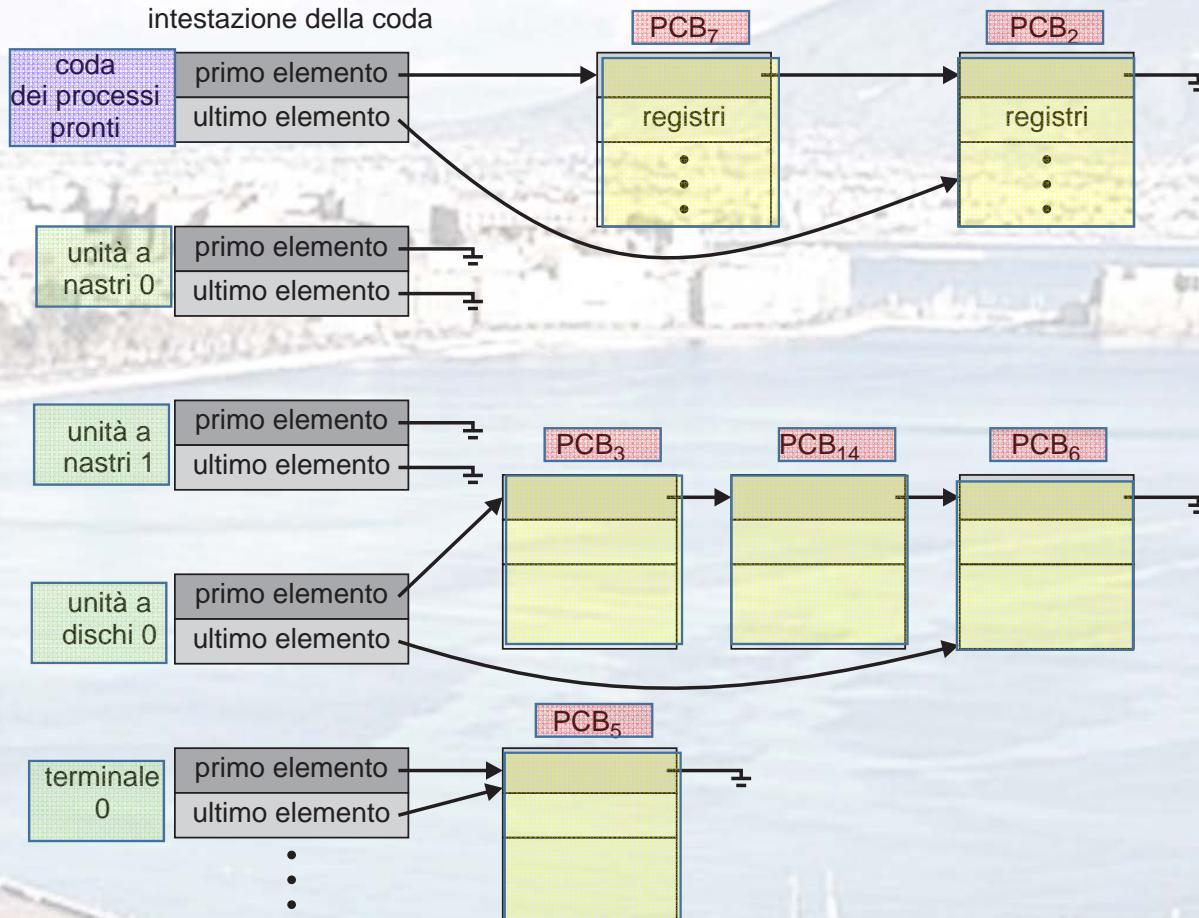
Code di scheduling dei processi

- **Coda di processi (*job queue*)**: insieme di tutti i processi del sistema.
- **Coda dei processi pronti (*ready queue*)**: processi presenti nella memoria centrale, pronti e in attesa di essere eseguiti.
- **Coda del dispositivo (*device queue*)**: elenco dei processi che attendono la disponibilità di un particolare dispositivo di I/O.



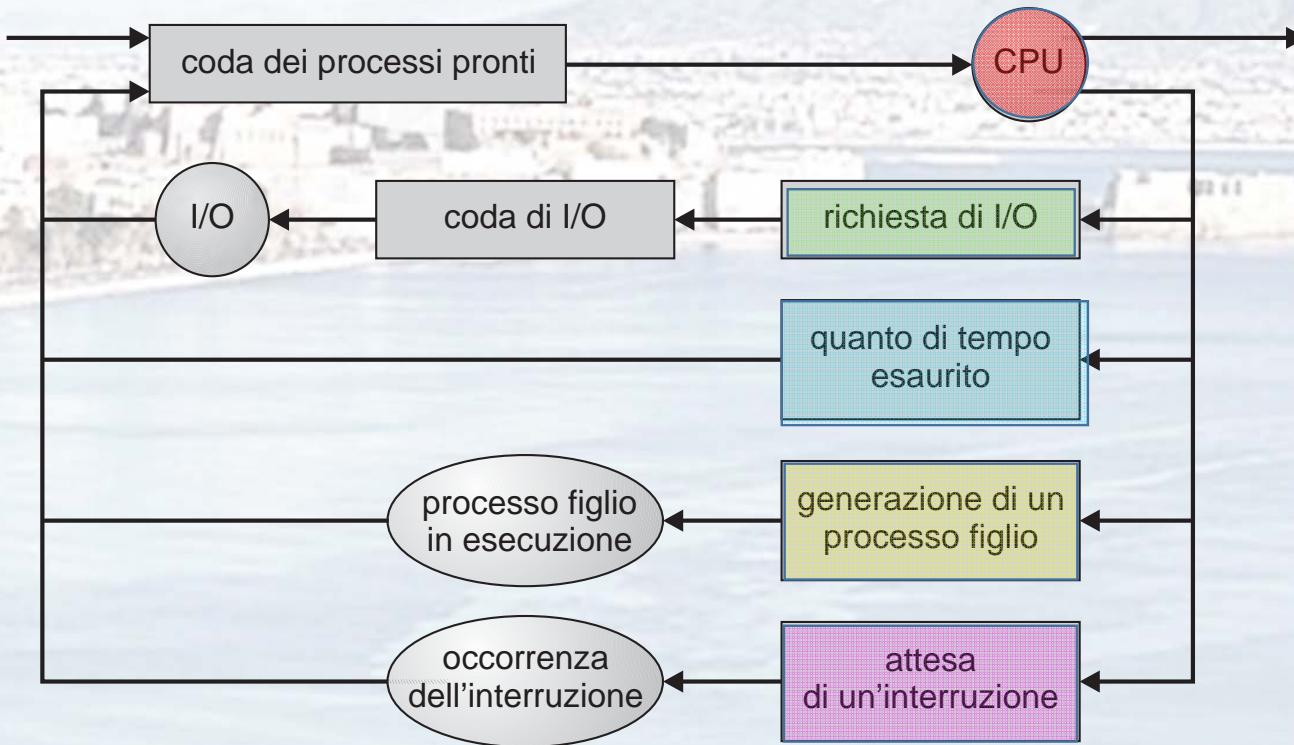
walter.balzano@gmail.com

Coda dei processi pronti e diverse code di dispositivi I/O



walter.balzano@gmail.com

Diagramma di accodamento per lo scheduling dei processi

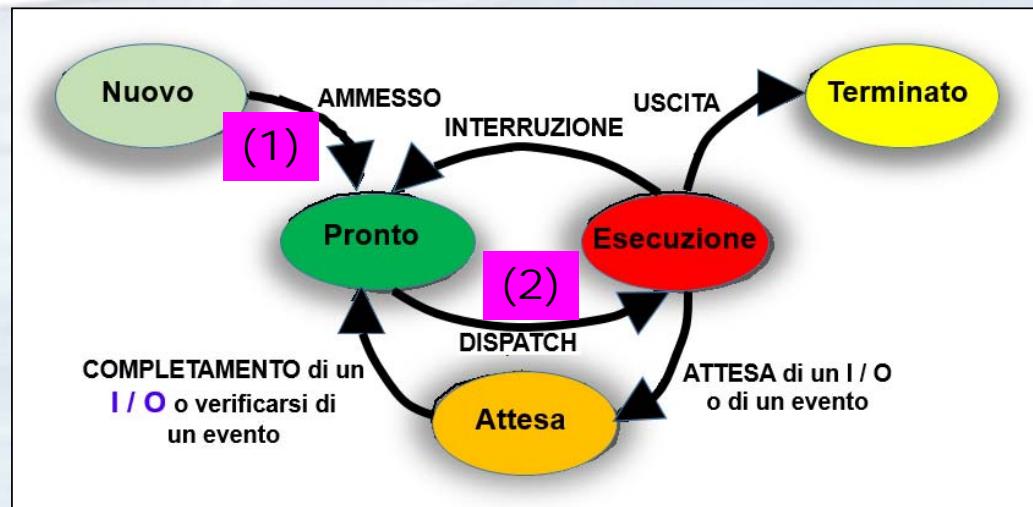


walter.balzano@gmail.com

Scheduler

(1) Lo **Scheduler a lungo termine** (o *job scheduler*) seleziona i processi che devono essere caricati nella coda dei processi pronti.

(2) Lo **Scheduler di CPU** (*short-term scheduler* o *CPU scheduler*) fa la selezione tra i lavori pronti per essere eseguiti e assegna la CPU a uno di essi.



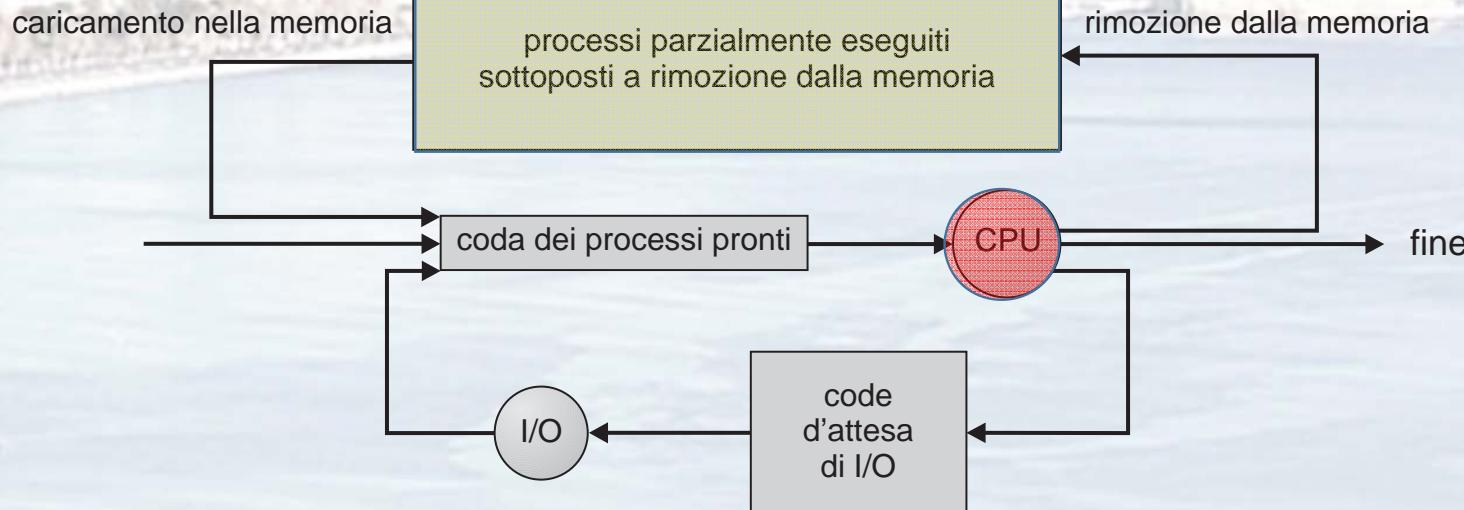
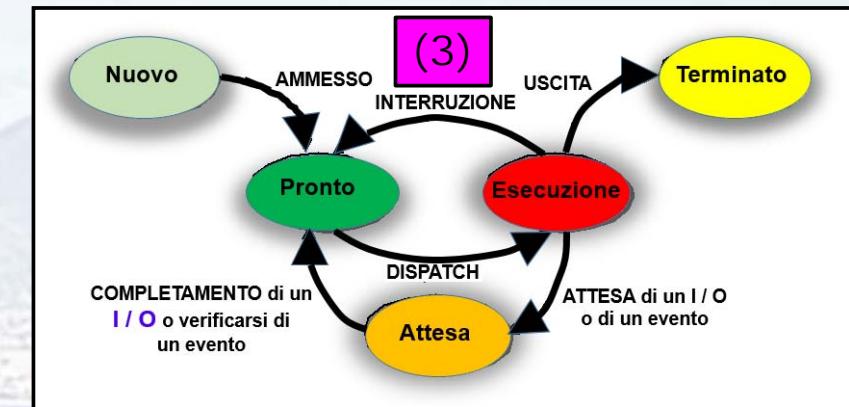
Latenza di dispatch — è il tempo impiegato dal dispatcher per sospendere un processo e avviare l'esecuzione di un nuovo processo



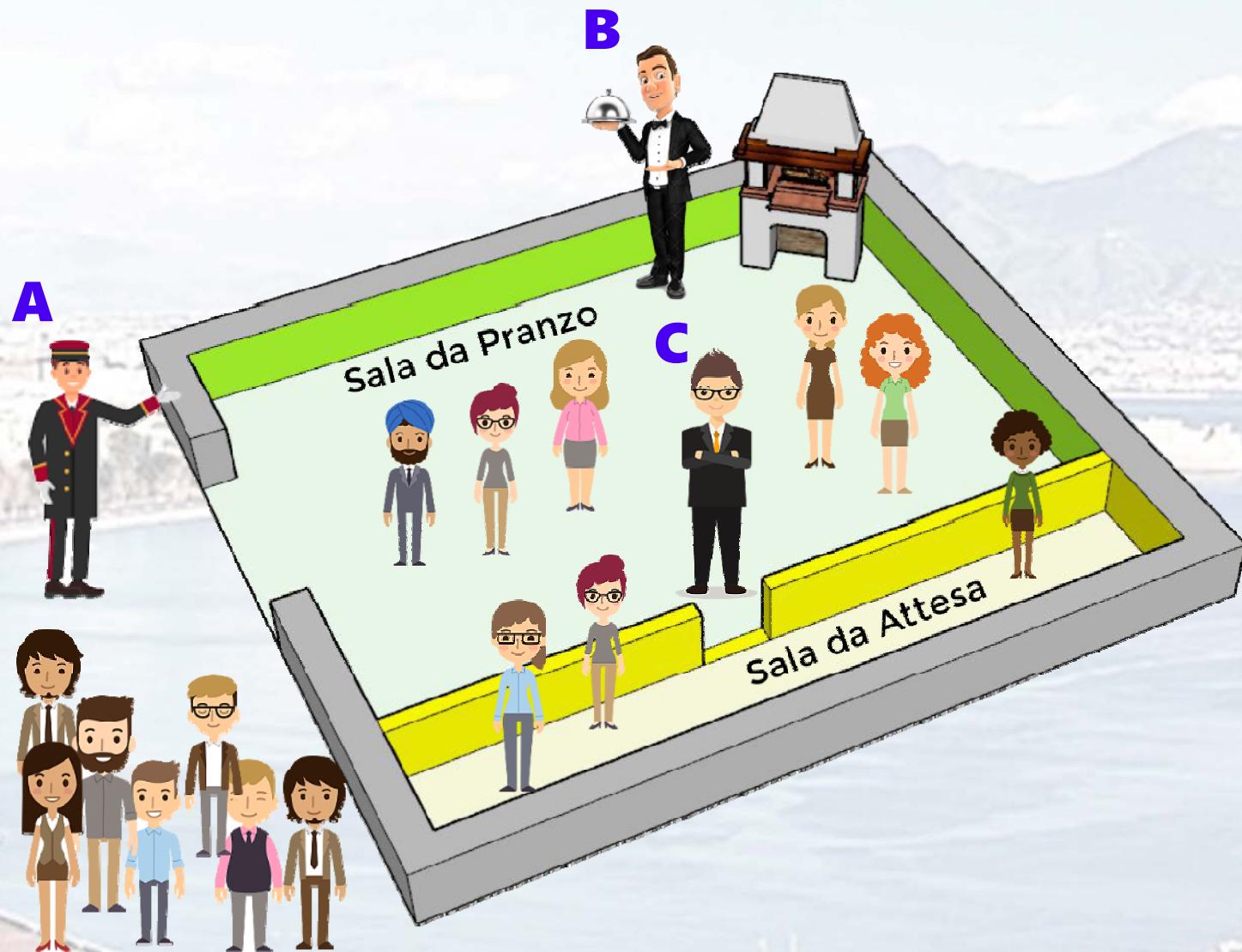
walter.balzano@gmail.com

Aggiunta di scheduling a medio termine al diagramma di accodamento

rimuove processi dalla memoria (e dalla contesa per la CPU)



walter.balzano@gmail.com



walter.balzano@gmail.com

Scheduler (Cont.)

- Lo scheduler di CPU viene invocato frequentemente (millisecondi) ⇒ (deve essere veloce).
- Lo scheduler a lungo termine viene invocato meno frequentemente (secondi, minuti) ⇒ (può essere lento).
- Lo scheduler lungo termine controlla il grado di multiprogrammazione.
- I processi possono essere caratterizzati come:
 - Processo con prevalenza di I/O (*I/O-bound process*): impiega la maggior parte del proprio tempo nell'esecuzione di operazioni di I/O.
 - Processo con prevalenza d'elaborazione (*CPU-bound process*): richiede poche operazioni di I/O e impiega la maggior parte del proprio tempo nelle elaborazioni.



walter.balzano@gmail.com

Cambio di contesto

- Passaggio della CPU a un nuovo processo, con conseguente registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo.
- **Il tempo necessario al cambio di contesto è puro sovraccarico:** il sistema non compie alcun lavoro utile durante la commutazione.
- La durata del cambio di contesto dipende molto dall'architettura.



walter.balzano@gmail.com

Creazione di un processo

- Un processo genitore può creare numerosi processi figli che, a loro volta, possono creare altri processi creando un **albero di processi**.
- **Condivisione delle risorse**
 - Genitore e figlio condividono tutte le risorse.
 - Il processo figlio condivide un sottoinsieme delle risorse del genitore.
 - Genitore e figlio non condividono alcuna risorsa.
- **Esecuzione**
 - Genitore e figlio possono essere eseguiti in modo concorrente.
 - Il processo genitore attende che alcuni o tutti i suoi processi figli terminino.



walter.balzano@gmail.com

Creazione di un processo (Cont.)

Spazio d'indirizzi

- Il processo figlio è un duplicato del processo genitore.
- Nel processo figlio si carica un programma.

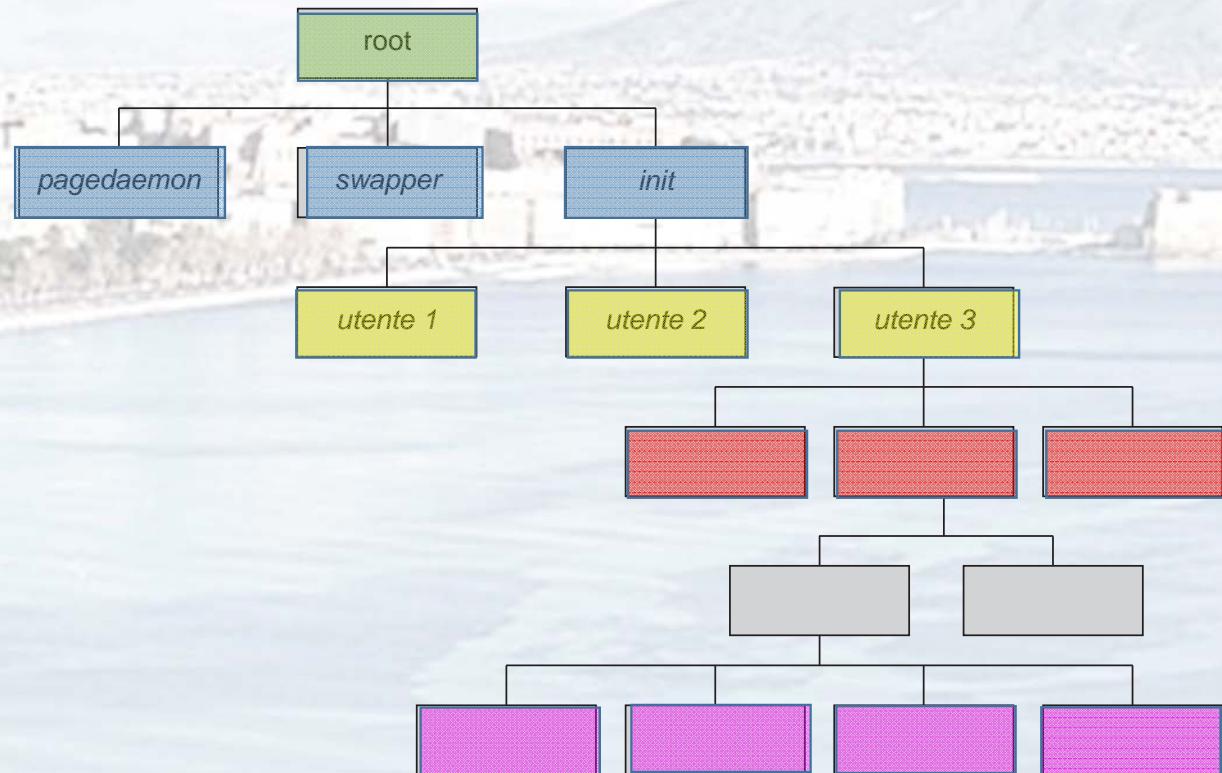
Esempi UNIX

- la chiamata di sistema **fork** crea un nuovo processo
- la chiamata di sistema **exec** dopo una **fork** sostituisce lo spazio di memoria del processo con un nuovo programma.



walter.balzano@gmail.com

Albero di processi in un tipico sistema UNIX



walter.balzano@gmail.com

Terminazione di un processo

- Un processo termina quando finisce l'esecuzione della sua ultima istruzione e chiede al sistema operativo di essere cancellato usando la chiamata del sistema **exit**.
 - Il processo figlio può riportare alcuni dati al processo genitore attraverso la chiamata del sistema **wait**.
 - Tutte le risorse del processo sono liberate dal sistema operativo.
- Un processo genitore può porre termine all'esecuzione di uno dei suoi processi figli (**abort**) perché:
 - Il processo figlio ha ecceduto nell'uso di alcune risorse.
 - Il compito assegnato al processo figlio non è più richiesto.
 - Il processo genitore termina.
 - **Il sistema operativo non consente a un processo figlio di continuare l'esecuzione in tale circostanza.**
 - **Terminazione a cascata** (*cascading termination*).



walter.balzano@gmail.com

Processi cooperanti

- Un processo è **indipendente** se non può influire su altri processi nel sistema o subirne l'influsso.
 - Un processo è **cooperante** se influenza o può essere influenzato da altri processi in esecuzione nel sistema.
- Vantaggi dei processi cooperanti
 - Condivisione d'informazioni
 - Accelerazione del calcolo
 - Modularità
 - Convenienza



walter.balzano@gmail.com

Il problema del produttore e del consumatore

- Usuale paradigma per i processi cooperanti: un processo **produttore** produce informazioni che sono consumate da un processo **consumatore**.
 - **memoria illimitata** (*unbounded-buffer*): non pone limiti alla dimensione del vettore.
 - **memoria limitata** (*bounded-buffer*): presuppone l'esistenza di una dimensione fissata del vettore.



walter.balzano@gmail.com

Memoria limitata – Soluzione con memoria condivisa

Supponiamo di usare un zona di memoria condivisa ed usiamo un vettore circolare di grandezza **DIM_VETTORE** e che fa uso di 2 indici per accedere al vettore.

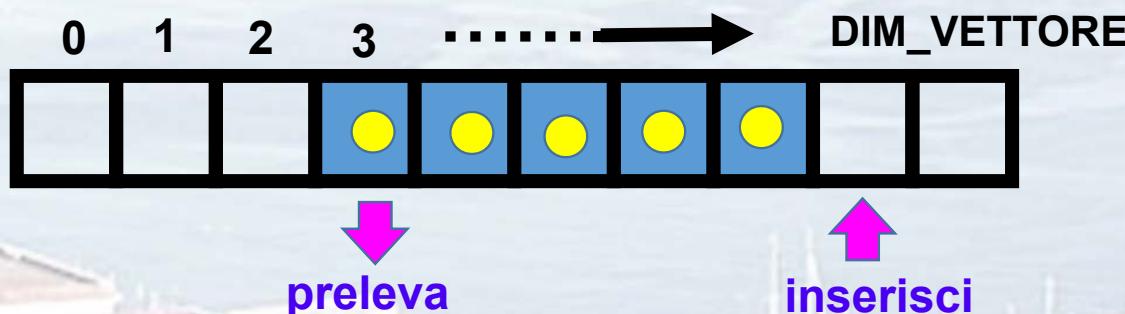
Con l'indice **inserisci** puntiamo alla prima casella disponibile del vettore.

Con l'indice **preleva** puntiamo alla prima posizione occupata del vettore.

Al momento iniziale poniamo a zero sia **inserisci** che **preleva**.

Da tali definizione si ricava che:

- **Se** **inserisci == preleva** **Allora** il vettore è vuoto
- **Se** $((\text{inserisci} + 1) \% \text{DIM_VETTORE}) == \text{preleva}$ **Allora** vettore pieno
- Max elementi nel vettore = **DIM_VETTORE - 1**



walter.balzano@gmail.com

Memoria limitata – Soluzione con memoria condivisa

Dati condivisi

```
#define DIM_VETTORE 10
Typedef struct {
    ...
} elemento;
elemento
    vettore[DIM_VETTORE];
    int inserisci = 0;
    int preleva = 0;
```



walter.balzano@gmail.com

Memoria limitata – Processo produttore

```
item appena_Prodotto;  
  
while (1) {  
    while (((inserisci + 1) % DIM_VETTORE) == preleva)  
        ; /* non fa niente */  
    vettore[inserisci] = appena_Prodotto;  
    inserisci = (inserisci + 1) % DIM_VETTORE;  
}
```



walter.balzano@gmail.com

Memoria limitata – Processo consumatore

```
item da_Consumare;

while (1) {
    while (inserisci == preleva)
        ; /* non fa niente */
    da_Consumare = vettore[preleva];
    preleva = (preleva + 1) %
DIM_VETTORE;
}
```



walter.balzano@gmail.com

Comunicazione tra processi (IPC)

- Attraverso le funzioni di IPC i processi possono comunicare tra loro e sincronizzare le proprie azioni.
- Un sistema di scambio di messaggi permette ai processi di comunicare tra loro senza ricorrere a dati condivisi.
- Un sistema di IPC fornisce le due operazioni:
 - **send (*messaggio*)** : dimensione fissa o variabile
 - **receive (*messaggio*)**
- Se i processi *P* e *Q* vogliono comunicare, devono:
 - stabilire un **canale di comunicazione** tra loro
 - Scambiare messaggi mediante **send/receive**
- Realizzazione di un canale di comunicazione
 - fisica (es. memoria condivisa, bus, reti)
 - logica (es. proprietà logiche)



walter.balzano@gmail.com

Questioni implementative

- Come sono stati creati i canali di comunicazione?
- Può un canale essere associato a più di due processi?
- Quanti canali possono esserci tra ogni coppia di processi comunicanti?
- Qual è la capacità di un canale di comunicazione?
- La dimensione di un messaggio deve essere fissa o variabile?
- Il canale di comunicazione è unidirezionale o bidirezionale?



walter.balzano@gmail.com

Comunicazione diretta

- I processi devono nominarsi reciprocamente in modo esplicito:
 - **send**(P , *messaggio*) : invia messaggio al processo P
 - **receive**(Q , *messaggio*) : riceve, in *messaggio*, un messaggio dal processo Q
- Caratteristiche di un canale di comunicazione:
 - I canali vengono stabiliti automaticamente.
 - Un canale è associato esattamente a una coppia di processi.
 - Esiste esattamente un canale tra ciascuna coppia di processi.
 - Il canale può essere unidirezionale, ma è in genere bidirezionale.



walter.balzano@gmail.com

Comunicazione indiretta

- I messaggi vengono inviati a delle **porte** (dette anche *mailbox*) che li ricevono
 - Ciascuna porta è identificata in modo univoco.
 - Due processi possono comunicare solo se condividono una porta.
- Caratteristiche di un canale di comunicazione:
 - Tra una coppia di processi si stabilisce un canale solo se entrambi i processi condividono una stessa porta
 - Un canale può essere associato a più di due processi.
 - Ciascuna coppia di processi può condividere più canali di comunicazione.
 - Il canale può essere unidirezionale o bidirezionale.



walter.balzano@gmail.com

Comunicazione indiretta

- Operazioni:

- creare una nuova porta
- inviare e ricevere messaggi tramite la porta
- rimuovere una porta

- Primitive:

send(A, messaggio) :

invia **messaggio** alla porta A

receive(A, messaggio) :

riceve, in **messaggio**,
un messaggio dalla porta A



walter.balzano@gmail.com

Comunicazione indiretta

- Condivisione della porta:

- P_1 , P_2 , e P_3 condividono la porta A.
- P_1 invia un messaggio ad A; P_2 e P_3 eseguono una receive da A.
- Quale processo riceverà il messaggio?

- Soluzioni:

- consentire che un canale sia associato al massimo a due processi
- consentire a un solo processo alla volta di eseguire un'operazione receive
- Consentire al sistema di decidere arbitrariamente quale processo riceverà il messaggio. Il sistema può comunicare l'identità del ricevente al trasmittente.



walter.balzano@gmail.com

Sincronizzazione

- Lo scambio di messaggi può essere **sincrono (bloccante)** oppure **asincrono (non bloccante)**.
- Anche le primitive **send** and **receive** possono essere sincrone oppure asincrone.



walter.balzano@gmail.com

Code di messaggi (buffering)

- Se la comunicazione è diretta o indiretta, i messaggi scambiati tra processi comunicanti risiedono in code temporanee.

Fondamentalmente esistono tre modi per realizzare queste code:

1. **Capacità zero**: 0 messaggi

Il trasmittente deve fermarsi finché il ricevente prende in consegna il messaggio (*rendezvous*).

2. **Capacità limitata**: la coda ha lunghezza finita n

Il trasmittente deve attendere se il canale è pieno.

3. **Capacità illimitata**: la coda ha lunghezza

potenzialmente infinita

Il trasmittente non si ferma mai.



walter.balzano@gmail.com

Comunicazione nei sistemi client-server

- Socket
- Chiamate di procedure remote, RPC
- Invocazione di metodi remoti, RMI (Java)



walter.balzano@gmail.com

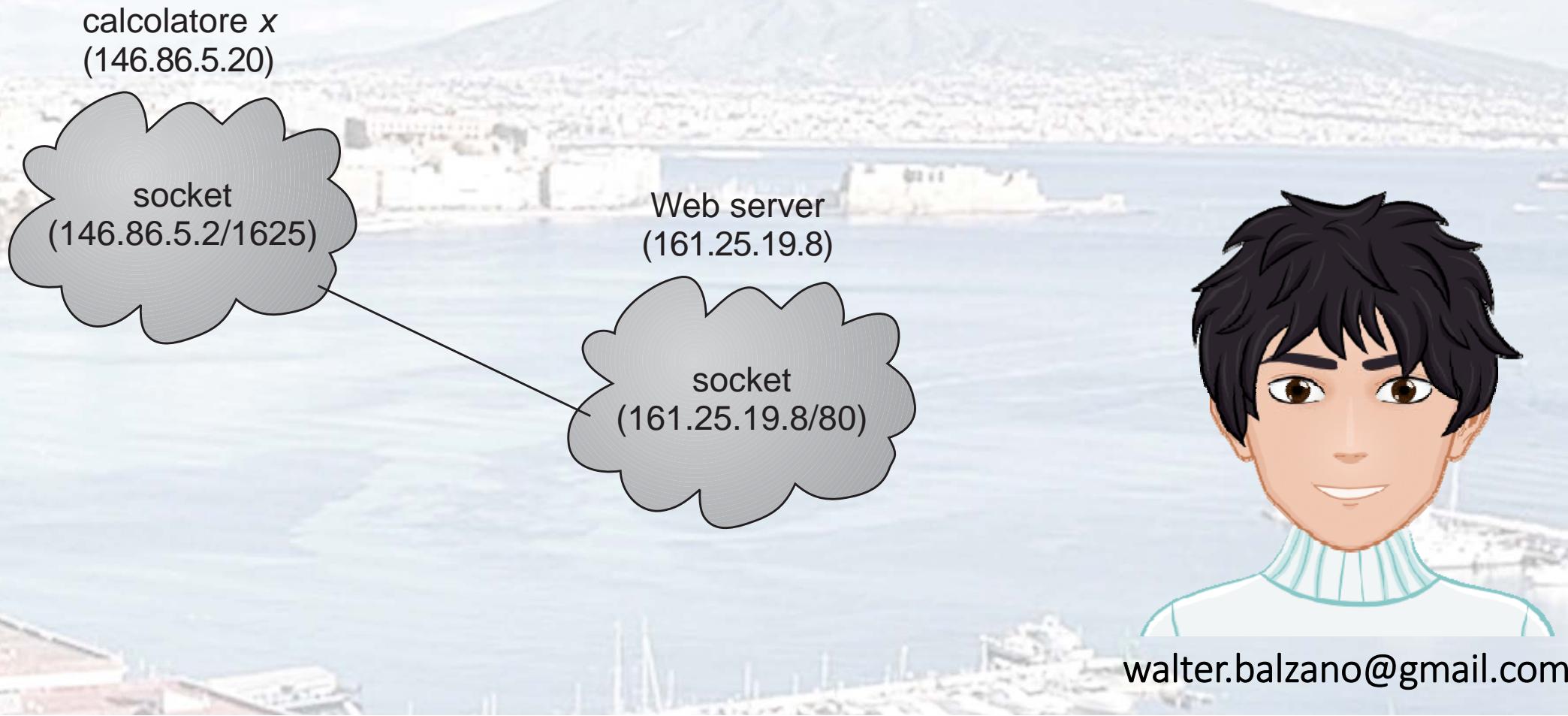
Sockets

- Una socket è definita come *l'estremità di un canale di comunicazione*.
- Ogni socket è identificata da un indirizzo IP concatenato a un numero di porta.
- La socket **161.25.19.8:1625** si riferisce alla porta **1625** sul calcolatore **161.25.19.8**
- La comunicazione avviene attraverso una coppia di socket.



walter.balzano@gmail.com

Comunicazione tramite socket



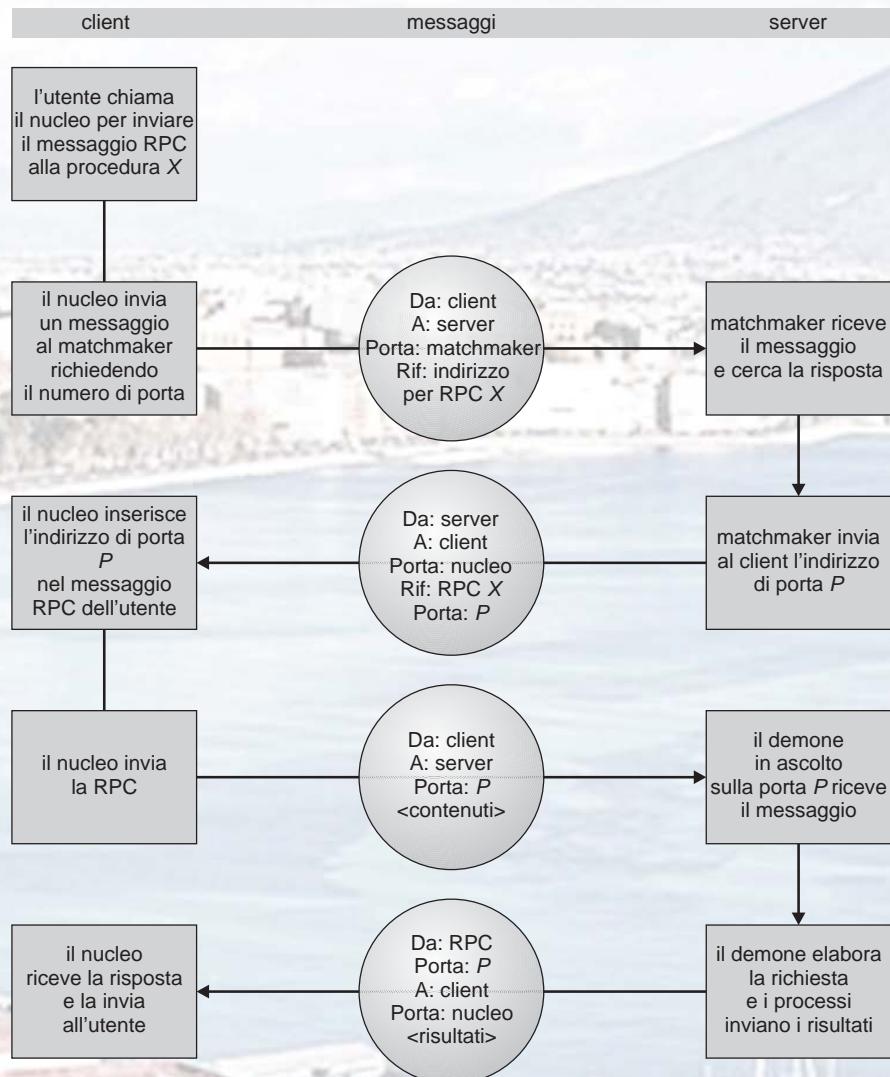
Chiamate di procedure remote (RPC)

- La **RPC** è stata progettata per astrarre il meccanismo della chiamata di procedura affinché si possa usare tra sistemi collegati tramite una rete.
- **Stub:** segmento di codice: proxy lato client per la procedura in corso sul server.
- Il segmento di codice di riferimento individua la porta del server e struttura i parametri (*marshalling*).
- Un analogo segmento di codice i riferimento nel server riceve questo messaggio e invoca la procedura nel server.



walter.balzano@gmail.com

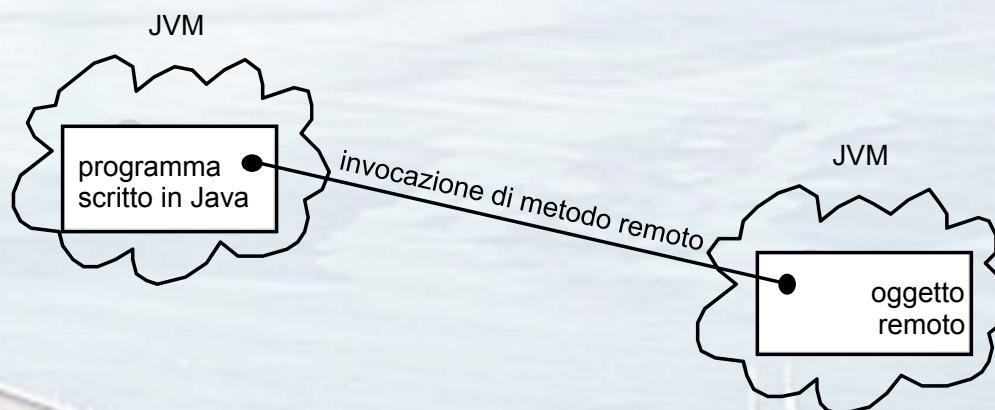
Esecuzione di una RPC



walter.balzano@gmail.com

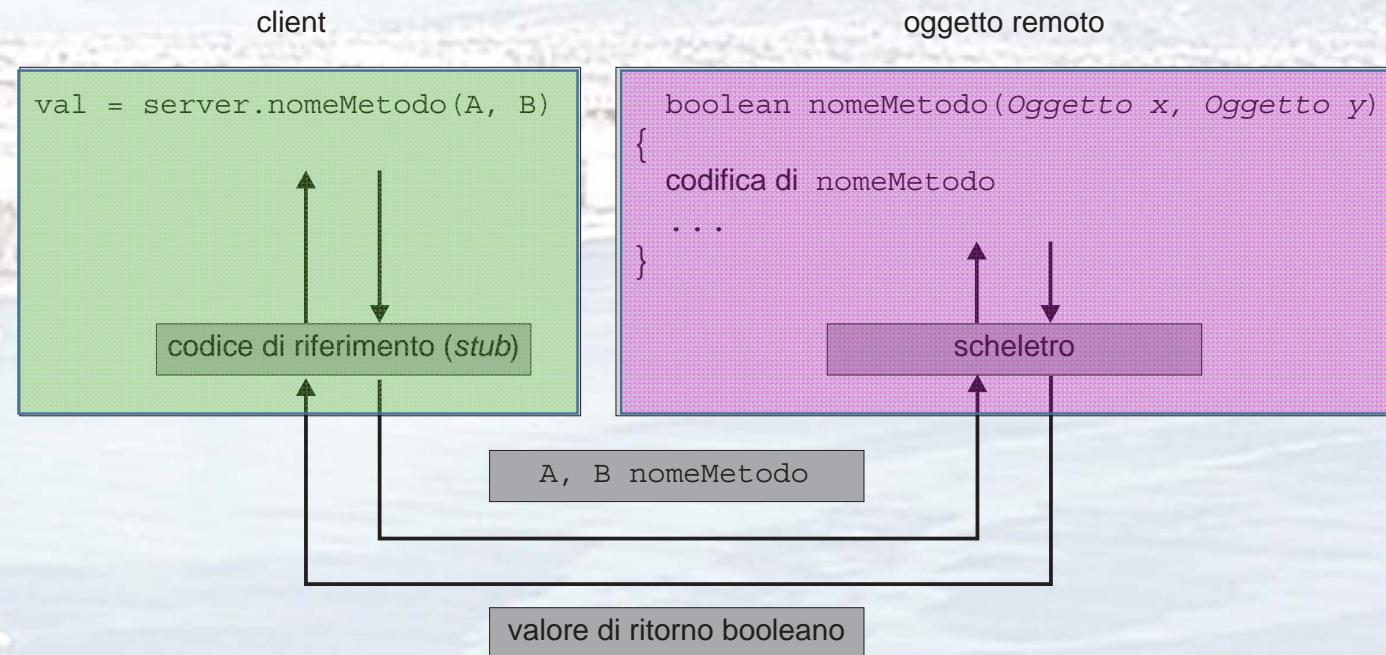
Invocazione di metodi remoti

- L'invocazione di metodi remoti (RMI, remote method invocation) **è una funzione del linguaggio Java simile alla RPC.**
- Una RMI consente a un thread di invocare un metodo di un oggetto remoto.



walter.balzano@gmail.com

Strutturazione dei parametri



walter.balzano@gmail.com

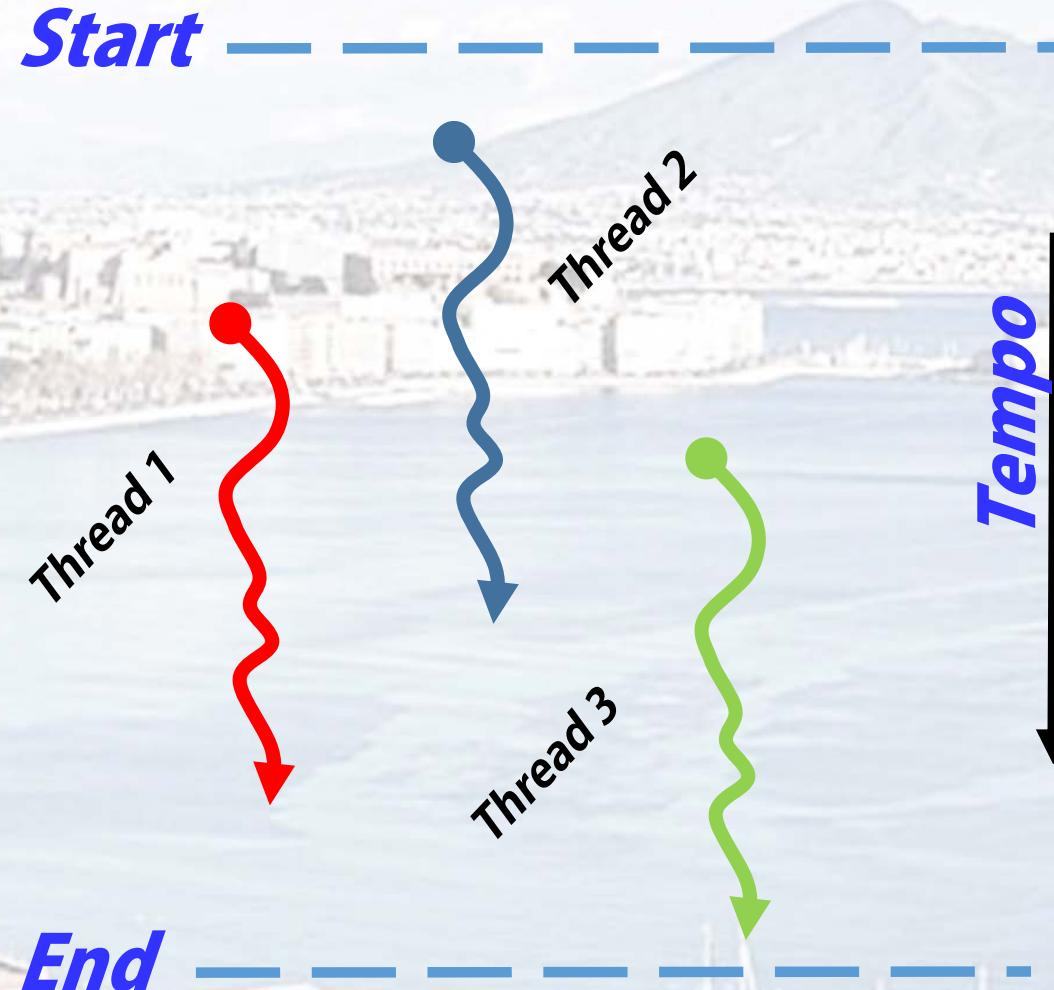
Thread

- Introduzione
- Modelli di programmazione multithread
- Questioni di programmazione multithread
- Pthreads
- Thread nei principali sistemi



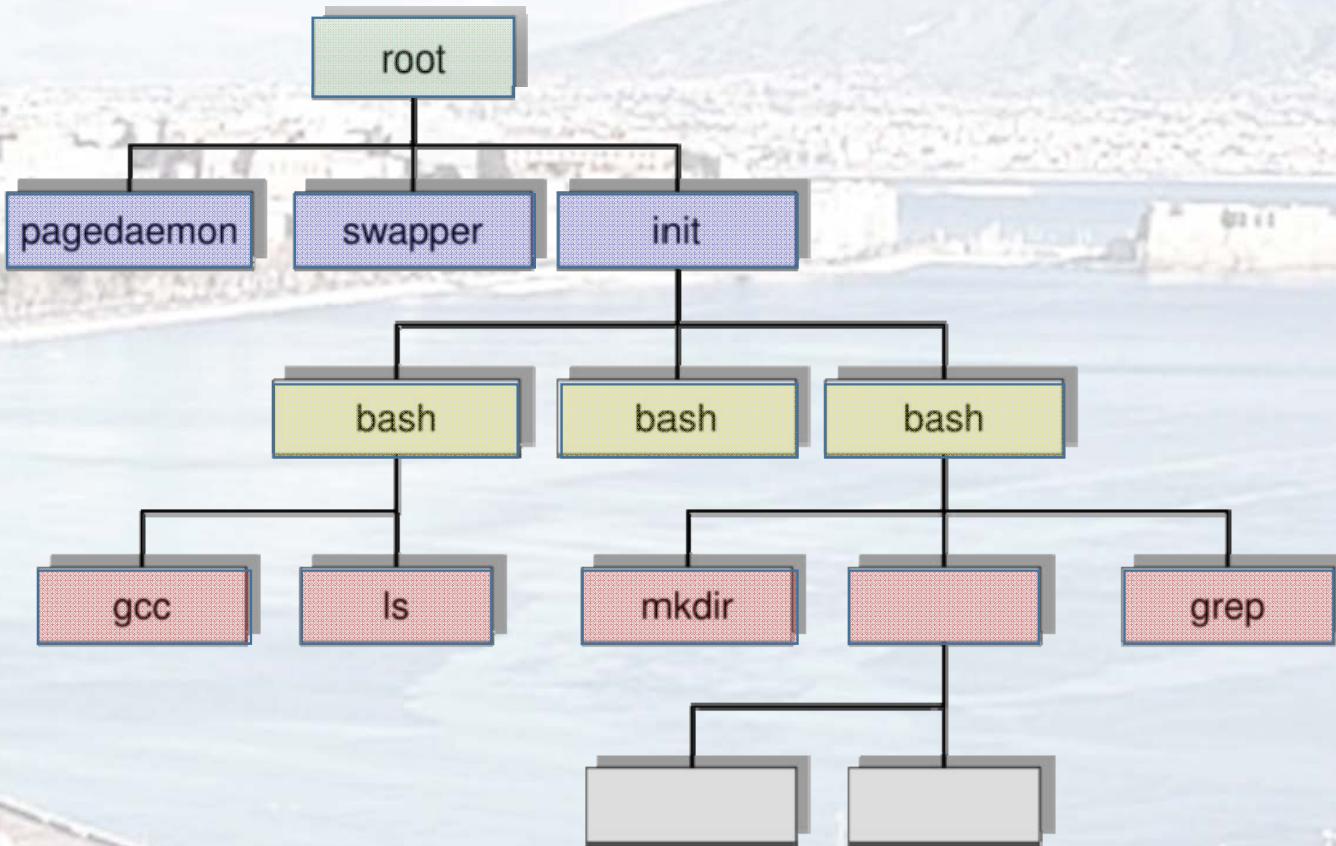
walter.balzano@gmail.com

Thread



walter.balzano@gmail.com

Thread Linux & Unix



walter.balzano@gmail.com

Processi e Thread

Processo (o task)

Programma in esecuzione, con il suo spazio di indirizzamento ed il suo stato. Un processo è ad esempio un'istanza di Word o Chrome. Esso è detto anche “**processo pesante**”, in riferimento al contesto (spazio di indirizzamento, stato) che si porta dietro.

Thread

Singolo flusso sequenziale di controllo all'interno di un processo. Un processo può contenere più thread.
Tutti i thread di un processo condividono lo stesso spazio di indirizzamento. Detto anche “**processo leggero**”, perché ha un contesto semplice.



walter.balzano@gmail.com

Parallelismo

Sistema parallelo

Architettura in cui sono presenti più unità di elaborazione (CPU) sulle quali sono in esecuzione processi e thread. In ogni istante di tempo, posso avere più di un processo o più di un thread fisicamente in esecuzione.

Sistema monoprocesso time-sliced

Vi è una sola unità di elaborazione. Il **parallelismo di processi e thread viene simulato (concorrenza)** allocando a ciascuno una frazione del tempo di CPU (time slice). Allo scadere di ogni unità di tempo, il S.O. opera un cambio di contesto (**context switch**). I thread sono processi leggeri perché il cambio di contesto è veloce.



walter.balzano@gmail.com

Preemption

Sistema non preemptive

il cambio di contesto avviene quando il processo o il thread interrompe la propria esecuzione o **volontariamente**, o perché in attesa di un evento (input, output).

Sistema preemptive

allo scadere del time slice il processo o il thread viene forzatamente interrotto e viene operato il cambio di contesto.
Es.: Unix, Windows sono sistemi multitasking, multithreading, preemptive. Il vecchio Windows 3.1 è un sistema non preemptive.

La schedulazione dei processi e dei thread può avvenire secondo diversi algoritmi (**Round-Robin**, con priorità, ...).



walter.balzano@gmail.com

Motivazioni dei Thread

È spesso necessario dividere il programma anche in “sotto-compiti” indipendenti.

- Un programma può avere diverse funzioni concorrenti:
 - **operazioni ripetute nel tempo** ad intervalli regolari (es. animazioni)
 - esecuzione di compiti laboriosi **senza bloccare la GUI del programma**
 - attesa di messaggi da un altro programma
 - attesa di input da tastiera o dalla rete.
 - Es.: Si pensi ai compiti svolti da un web browser.
- L'alternativa al multithreading è il polling. Il polling, oltre che scomodo da implementare, consuma molte risorse di CPU. È quindi estremamente inefficiente



walter.balzano@gmail.com

Definizioni thread e multithread

- Un thread è un singolo flusso di controllo sequenziale all'interno di un programma.
- ogni thread ha un inizio, una fine, una sequenza ed ad ogni istante un solo punto di esecuzione
- Thread = Lightweight process: utilizza il contesto del processo.
- Execution context: il contesto di un singolo thread prevede inoltre, ad esempio, un proprio stack ed un proprio Program Counter.



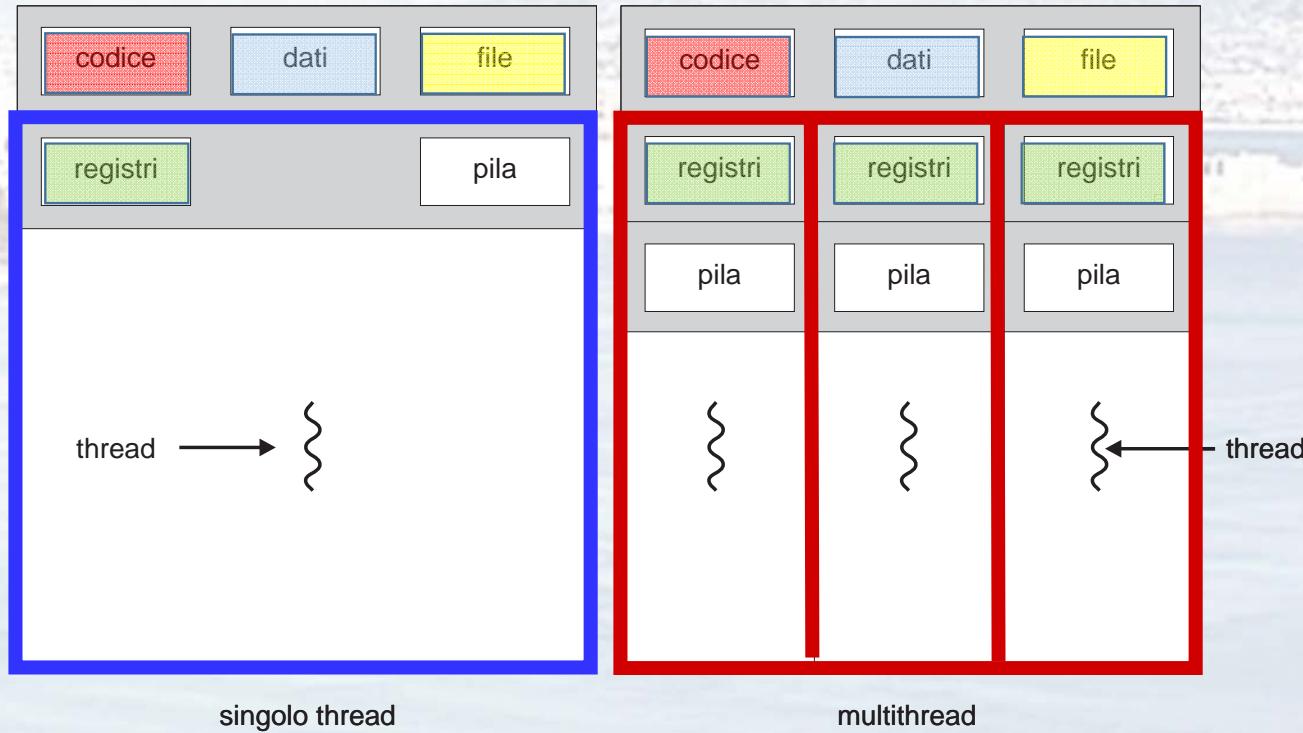
walter.balzano@gmail.com

Esempio



walter.balzano@gmail.com

Processi a singolo thread e multithread



walter.balzano@gmail.com

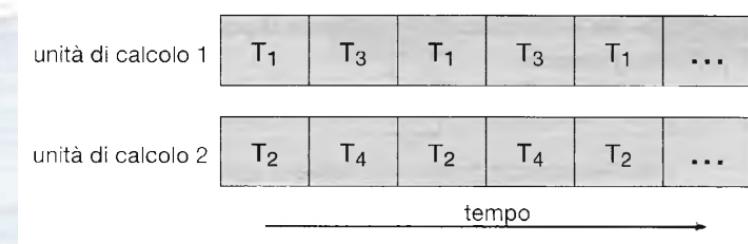
Vantaggi

- **Tempo di risposta**

Esempio: un Browser Web multithread potrebbe permettere l'interazione con un utente mediante un thread mentre una immagine viene caricata da un altro thread.

- **Condivisione delle risorse**

Esempio: stesso spazio di indirizzi



- **Economia**

Assegnare memoria e risorse per la creazione di nuovi processi è costoso... Un thread è 'leggero' e richiede meno assegnazioni di risorse.

- **Uso di più unità d'elaborazione**

I thread possono essere eseguiti in parallelo



walter.balzano@gmail.com

Thread al livello d'utente

- Essi sono Gestiti come uno strato separato sopra il nucleo del sistema operativo, e **realizzati tramite una libreria di funzioni** per la creazione, lo scheduling e la gestione, **senza alcun intervento diretto del nucleo.**
- Esempi:
 - Libreria POSIX *Pthreads*
 - Libreria *C-threads* del sistema Mach
 - *UI-threads* del sistema Solaris 2



walter.balzano@gmail.com

Thread al livello del nucleo

- Gestiti direttamente dal sistema operativo.
- Esempi:
 - Windows 95/98/NT/2000
 - Solaris
 - Tru64 UNIX
 - BeOS
 - Linux



walter.balzano@gmail.com

Modelli di programmazione multithread

- Modello da molti a uno
- Modello da uno a uno
- Modello da molti a molti



walter.balzano@gmail.com

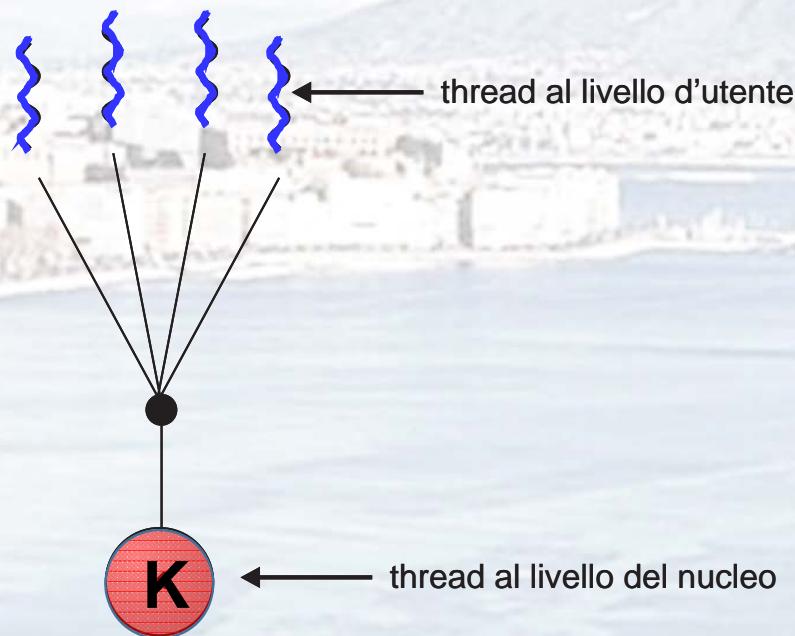
Modello da molti a uno

- Fa corrispondere molti thread al livello d'utente a un singolo thread al livello del nucleo.
- Usato su sistemi che non gestiscono i thread al livello del nucleo.



walter.balzano@gmail.com

Modello da molti a uno



walter.balzano@gmail.com

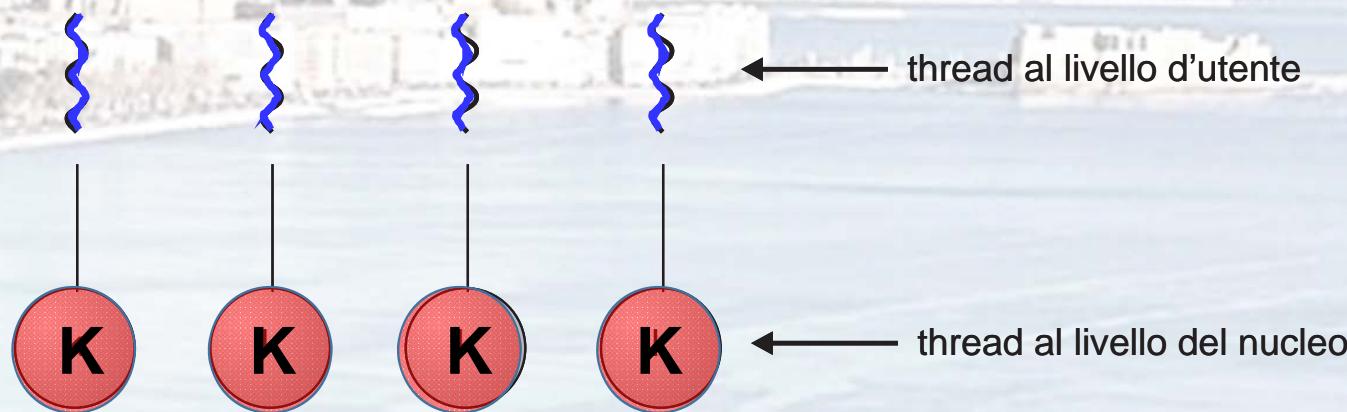
Modello da uno a uno

- Mette in corrispondenza ciascun thread del livello d'utente con un thread del livello del nucleo.
- Esempi:
 - Windows 95/98/NT/2000
 - OS/2



walter.balzano@gmail.com

Modello da uno a uno



walter.balzano@gmail.com

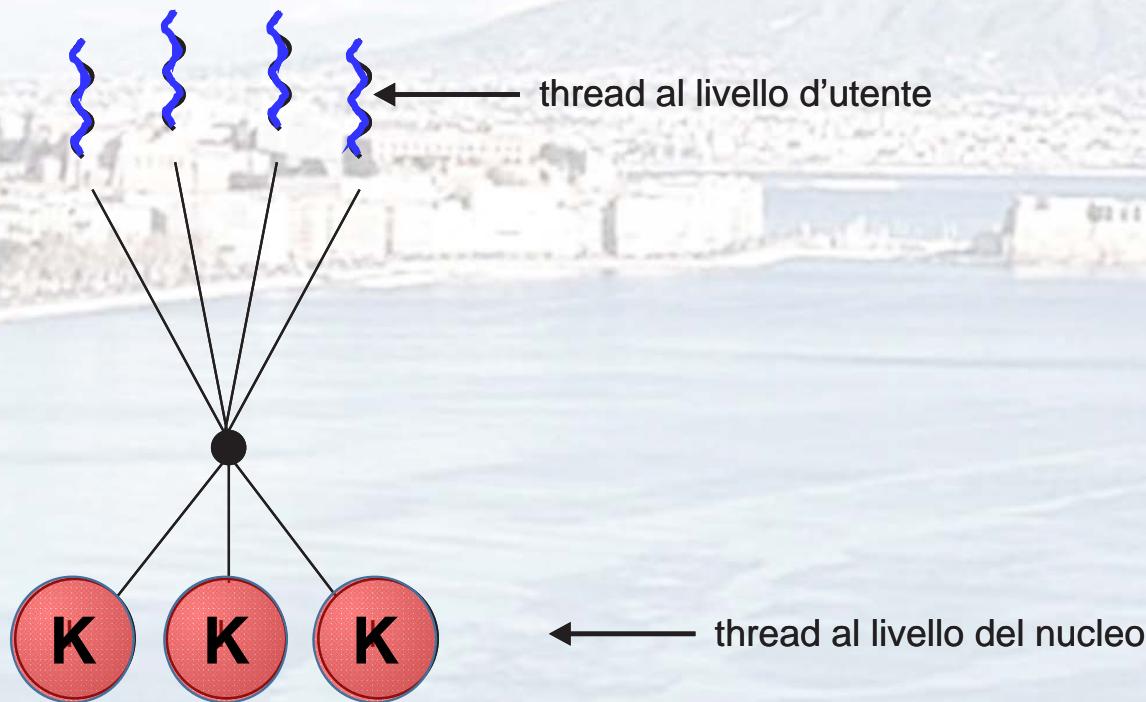
Modello da molti a molti

- Mette in corrispondenza più thread del livello d'utente con un numero minore o uguale di thread del livello del nucleo.
- consente ai programmati di creare liberamente i thread che ritengono necessari.
- Solaris 2
- Windows NT/2000 con il pacchetto *ThreadFiber*



walter.balzano@gmail.com

Modello da molti a molti



walter.balzano@gmail.com

Pthreads

- Lo standard POSIX (IEEE 1003.1c) che definisce l'API per la creazione e la sincronizzazione dei thread.

**POSIX: Portable Operating System Interface
for Unix**

- Non si tratta di una *realizzazione* ma di una *definizione del comportamento dei thread*; i progettisti di sistemi operativi possono realizzare le API così definite come meglio credono.
- Comuni nei sistemi UNIX.



walter.balzano@gmail.com

```

#include <pthread.h>
#include <stdio.h>

int sum; /* questo dato è condiviso dai thread */
void *runner(void *param); /* il thread */

int main(int argc, char *argv[])
{
    pthread_t tid; /* identificatore del thread */
    pthread_attr_t attr; /* insieme di attributi del thread */

    if (argc != 2) {
        fprintf(stderr,"usage: a.out <integer value>\n");
        return -1;
    }
    if (atoi(argv[1]) < 0) {
        fprintf(stderr,"%d must be >= 0\n", atoi(argv[1]));
        return -1;
    }

    /* reperisce gli attributi predefiniti */
    pthread_attr_init(&attr);
    /* crea il thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* attende la terminazione del thread */
    pthread_join(tid,NULL);

    printf("sum = %d\n",sum);
}

/* Il thread assume il controllo da questa funzione */
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}

```

Pthreads

Programma Multithread in **Linguaggio C**
che impiega le API Pthread

Il programma esegue il primo thread
«main» (programma «padre») e poi
suo il thread **«runner»** («figlio»)



walter.balzano@gmail.com

```

#include <windows.h>
#include <stdio.h>
DWORD Sum; /* dato condiviso dai thread */
/* il nuovo thread è eseguito in questa funzione apposita */

DWORD WINAPI Summation(LPVOID Param)
{
    DWORD Upper = *(DWORD*)Param;
    for (DWORD i = 0; i <= Upper; i++)
        Sum += i;
    return 0;
}

int main(int argc, char *argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;
    /* semplice controllo degli errori sui valori di ingresso
     */
    if (argc != 2) {
        fprintf(stderr,"An integer parameter is required\n");
        return -1;
    }
    Param = atoi(argv[1]);
    if (Param < 0) {
        fprintf(stderr,"An integer >= 0 is required\n");
        return -1;
    }

    // crea il nuovo thread
    ThreadHandle = CreateThread(
        NULL, // attributi di default per la sicurezza
        0, // dimensione di default della pila
        Summation, // funzione del thread
        &Param, // parametro della funzione del thread
        0, // flag di creazione di default
        &ThreadId); // restituisce l'identificatore del thread

    if (ThreadHandle != NULL) {
        // attende che il thread figlio termini
        WaitForSingleObject(ThreadHandle,INFINITE);

        // chiude il riferimento al thread figlio
        CloseHandle(ThreadHandle);

        printf("sum = %d\n",Sum);
    }
}

```

Pthreads

Programma Multithread
in linguaggio C che
impiega le API **Win32**



walter.balzano@gmail.com

Thread nel linguaggio Java

- I thread nel linguaggio Java possono essere creati
 - creando una nuova classe derivata dalla classe **Thread**
 - sovrascrivendo il metodo **run** di quella classe
- I thread nel linguaggio Java sono gestiti dalla macchina virtuale (**JVM**).



walter.balzano@gmail.com

Questioni di programmazione multithread (1)

- chiamate del sistema **fork()** ed **exec()**

in un programma multithread la semantica delle chiamate di sistema fork() ed exec() cambia se un thread in un programma invoca la chiamata di sistema fork(), il nuovo processo potrebbe, in generale, contenere un duplicato di tutti i thread oppure del solo thread invocante

- Cancellazione

Permette di terminare un thread prima che completi il suo compito.

- per es. in una ricerca concorrente in un database il primo thread che trova il risultato dovrebbe comportare la cancellazione degli altri).
- Per es. la terminazione di un web browser comporta le terminazioni dei caricamenti di pagina...



walter.balzano@gmail.com

Questioni di programmazione multithread (2)

- **Gestione dei segnali**

- Riguarda la gestione e coordinazione delle occorrenze di eventi che avvengono nel sistema

- **Gruppi di thread**

- Un numero illimitato di thread potrebbe esaurire le risorse del sistema come la CPU o la memoria. Si cerca di creare dei gruppi di thread il cui scopo è quello di snellirne la gestione.

- **Dati specifici dei thread**

- In particolari circostanze, ogni thread può necessitare di una copia privata di certi dati



walter.balzano@gmail.com

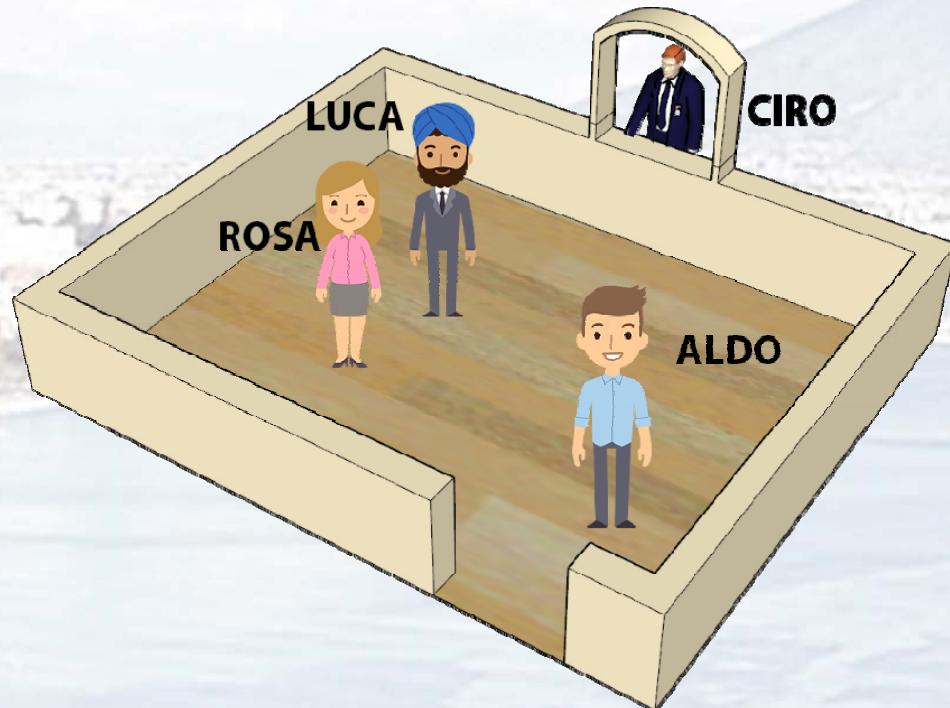
Capitolo 6: Scheduling della CPU

- Concetti fondamentali
- Criteri di scheduling
- Algoritmi di scheduling
- Scheduling per sistemi con più unità d'elaborazione
- Scheduling per sistemi d'elaborazione in tempo reale
- Valutazione degli algoritmi



walter.balzano@gmail.com

«Punti di Vista»



walter.balzano@gmail.com

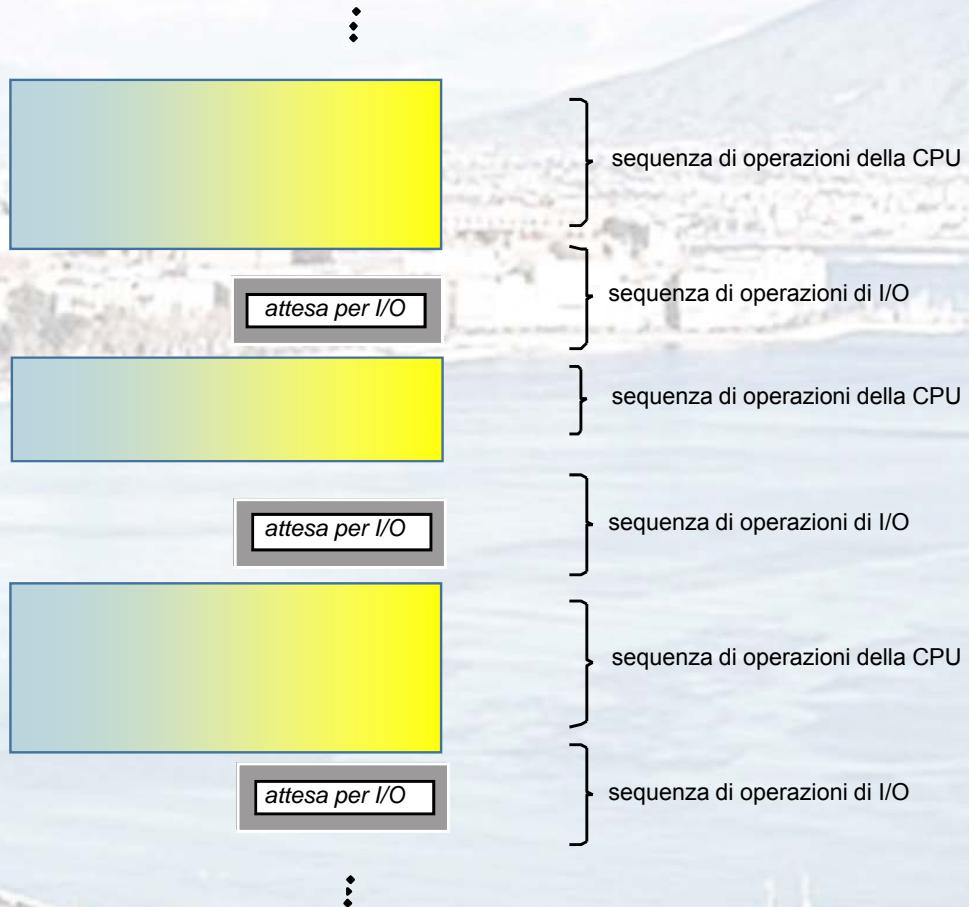
Concetti fondamentali

- L'obiettivo della mutiprogrammazione è avere sempre processi in esecuzione al fine di massimizzare l'utilizzo della CPU.
- L'esecuzione del processo consiste in un ciclo d'elaborazione (svolta dalla CPU) e d'attesa del completamento delle operazioni di I/O. I processi si alternano tra questi due stati.
- L'esecuzione del processo comincia con una sequenza (una "raffica") di operazioni d'elaborazione svolte dalla CPU (**CPU burst**) seguita da una sequenza di operazioni di I/O (**I/O burst**), quindi un'altra sequenza di operazioni della CPU, di nuovo una sequenza di operazioni di I/O, e così via.



walter.balzano@gmail.com

Serie alternata di sequenze di operazioni della CPU e di sequenze di operazioni di I/O



walter.balzano@gmail.com

Diagramma delle sequenze di operazioni della CPU



walter.balzano@gmail.com

Scheduler della CPU

- Ogniqualvolta la CPU passa nello stato d'inattività, il sistema operativo sceglie per l'esecuzione uno dei processi presenti nella coda dei processi pronti.
- Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:
 1. Un processo passa dallo stato di esecuzione a quello di attesa.
 2. Un processo passa dallo stato di esecuzione allo stato pronto.
 3. Un processo passa dallo stato di attesa allo stato pronto.
 4. Un processo termina.
- Nei casi 1 e 4 si dice che lo schema di scheduling è senza diritto di prelazione (*nonpreemptive*).
- In tutti gli altri casi lo schema di scheduling è con diritto di prelazione (*preemptive*).



walter.balzano@gmail.com

Dispatcher

- Il **dispatcher** è il modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine. Questa funzione riguarda:
 - il cambio di contesto
 - il passaggio al modo d'utente
 - il salto alla giusta posizione del programma d'utente per riavviarne l'esecuzione
- **Latenza di dispatch**: il tempo richiesto dal dispatcher per fermare un processo e avviare l'esecuzione di un altro.



walter.balzano@gmail.com

Criteri di scheduling

- **MAX Utilizzo della CPU**: la CPU deve essere più attiva possibile
- **MAX Produttività** (*throughput*): numero di processi completati nell'unità di tempo
- **Min Tempo di completamento**: tempo necessario per eseguire il processo stesso
- **Min Tempo d'attesa**: somma degli intervalli d'attesa passati nella coda dei processi pronti
- **Min Tempo di risposta**: tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta (è dato dal tempo necessario per iniziare una risposta, *non* dal suo tempo d'emissione)



walter.balzano@gmail.com

Scheduling first-come, first-served (FCFS)

<u>Processo</u>	<u>Durata seq.</u>
P_1	24
P_2	3
P_3	3

- Si supponga che i processi arrivino al tempo «0» ma nell'ordine: P_1, P_2, P_3
Lo schema di Gantt risultante è:



- Tempo d'attesa per $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo d'attesa medio: $(0 + 24 + 27)/3 = 17$



walter.balzano@gmail.com

Scheduling FCFS (Cont.)

Si supponga che i processi arrivino nell'ordine

$$P_2, P_3, P_1.$$

- Lo schema di Gantt è:



- Tempo d'attesa per $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Tempo d'attesa medio: $(6 + 0 + 3)/3 = 3$
- Miglioramento rispetto al caso precedente.
- Effetto convoglio: tutti i processi attendono che un lungo processo liberi la CPU.



walter.balzano@gmail.com

Scheduling shortest-job-first (SJF)

- **Associa a ogni processo la lunghezza della successiva sequenza di operazioni della CPU. Quando è disponibile, si assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU.**
- Due schemi:
 - **senza prelazione**: permette al processo correntemente in esecuzione di portare a termine la propria sequenza di operazioni della CPU.
 - **con prelazione**: se arriva un nuovo processo con sequenza di CPU inferiore a quella del tempo restante per eseguire il processo in corso, lo sostituisce al processo attualmente in esecuzione (talvolta chiamato *shortest-remaining-time first*, SRTF).
- L'algoritmo di scheduling SJF è *ottimale* nel senso che rende minimo il tempo d'attesa medio per un dato insieme di processi.

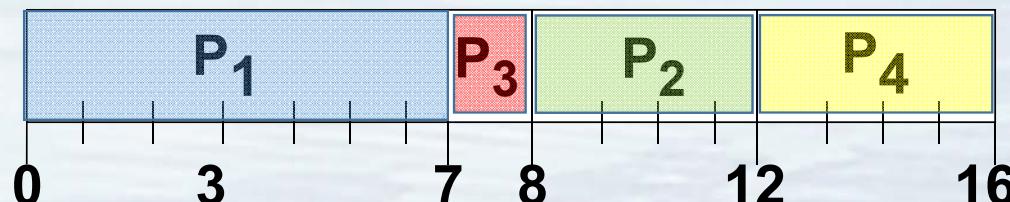


walter.balzano@gmail.com

Esempio di SJF senza diritto di prelazione

<u>Processo</u>	<u>Istante d'arrivo</u>	<u>Durata sequenza</u>
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (senza diritto di prelazione, *non-preemptive*)



- Tempo d'attesa medio = $(0 + 6 + 3 + 7)/4 = 4$

$$T_{\text{Attesa}} = T_{\text{Fine}} - T_{\text{Arrivo}} - T_{\text{Durata}}$$

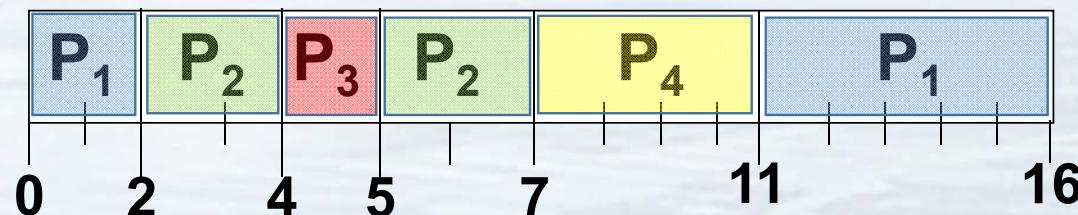


walter.balzano@gmail.com

Esempio di SJF con diritto di prelazione

<u>Processo</u>	<u>Istante d'arrivo</u>	<u>Durata sequenza</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (con diritto di prelazione, *preemptive*)



- Tempo d'attesa medio = $(9 + 1 + 0 + 2)/4 = 3$



walter.balzano@gmail.com

Determinare la lunghezza della successiva sequenza di operazioni della CPU

- Cercare di “predire” il suo valore: è probabile, infatti che sia simile ai precedenti.
- Calcolando un valore approssimato della lunghezza, si può scegliere il processo con la più breve fra tali lunghezze previste.

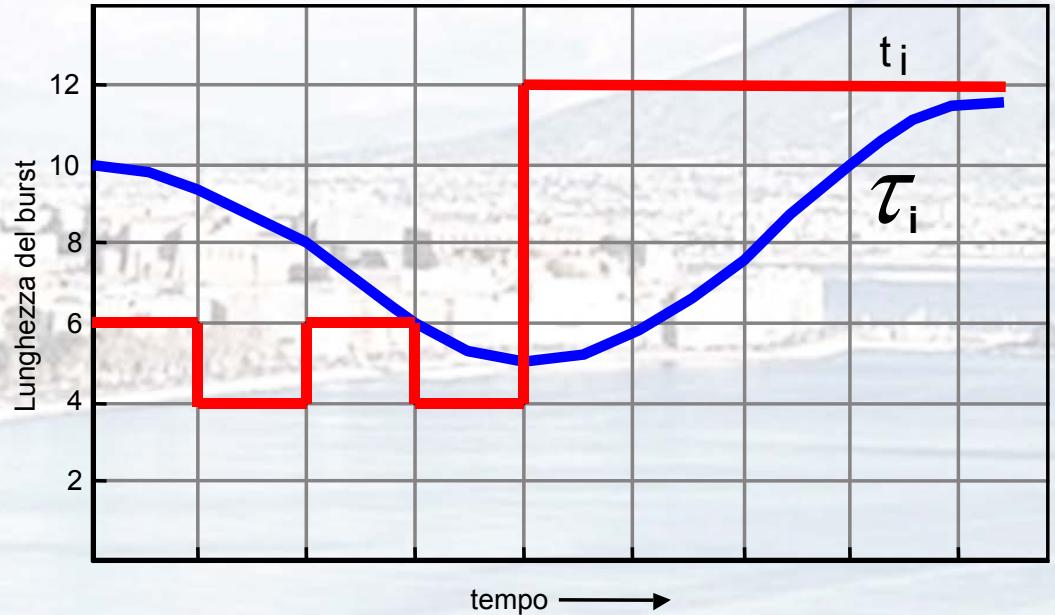
1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$



walter.balzano@gmail.com

Predizione della lunghezza della successiva sequenza di operazioni della CPU



CPU burst (t_i)	6	4	6	4	13	13	13	...	
predizione (τ_i)	10	8	6	6	5	9	11	12	...



walter.balzano@gmail.com

Esempi di media esponenziale

- $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - la storia recente non ha effetto.
- $\alpha = 1$
 - $\tau_{n+1} = t_n$
 - ha significato solo la più recente sequenza di operazioni della CPU.
- Se sviluppiamo la formula, otteniamo:
$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha) \alpha t_n - 1 + \dots \\ &\quad + (1 - \alpha)^j \alpha t_n - 1 + \dots \\ &\quad + (1 - \alpha)^{n-1} t_n \tau_0\end{aligned}$$
- Siccome sia α sia $(1 - \alpha)$ sono minori o uguali a 1, ogni termine ha peso inferiore a quello del suo predecessore.



walter.balzano@gmail.com

Scheduling per priorità

- Si assegna a ciascun processo un numero (intero) di priorità.
- Si assegna la CPU al processo con priorità più elevata (intero più piccolo \equiv priorità più elevata).
 - con diritto di prelazione
 - senza diritto di prelazione
- **L'algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità**, dove la priorità è l'inverso della lunghezza (prevista) della successiva sequenza di operazioni della CPU (a una maggiore lunghezza corrisponde una minore priorità, e viceversa).
- Problema \equiv **Attesa indefinita** (starvation): processi con bassa priorità possono rimanere nell'attesa indefinita della CPU.
- Soluzione \equiv **Invecchiamento** (aging): aumento graduale della priorità dei processi che attendono nel sistema da parecchio tempo.



walter.balzano@gmail.com

Scheduling circolare (round robin, RR)

- Ciascun processo riceve una piccola quantità fissata del tempo della CPU, chiamata quanto di tempo (*time quantum*), che varia generalmente da 10 a 100 millisecondi, e la coda dei processi pronti è trattata come una coda circolare.
- Se nella coda dei processi pronti esistono n processi e il quanto di tempo è pari a q , ciascun processo ottiene $1/n$ -esimo del tempo di elaborazione della CPU in frazioni di, al più, q unità di tempo, allora **ogni processo non deve attendere per più di $(n - 1)q$ unità di tempo.**
- Prestazioni:
 - q molto lungo (indefinito) \Rightarrow FCFS
 - q breve (es. 1 microsecondo) \Rightarrow condivisione della CPU (processor sharing).

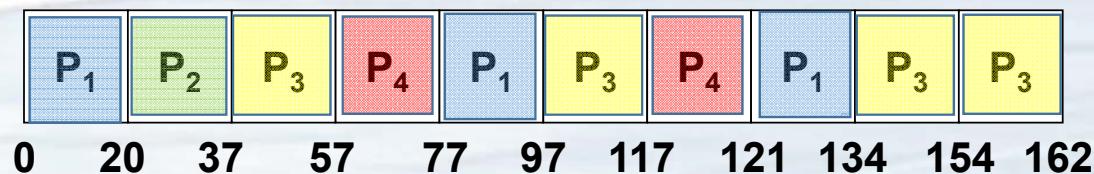


walter.balzano@gmail.com

Esempio di RR con $q = 20$

<u>Processo</u>	<u>Durata sequenza</u>
P_1	53
P_2	17
P_3	68
P_4	24

Lo schema di Gantt è:



- In generale, il tempo di completamento medio può migliorare se la maggior parte dei processi termina la successiva sequenza di operazioni della CPU in un solo quanto di tempo.



walter.balzano@gmail.com

Quanto di tempo e cambi di contesto

Esempio:

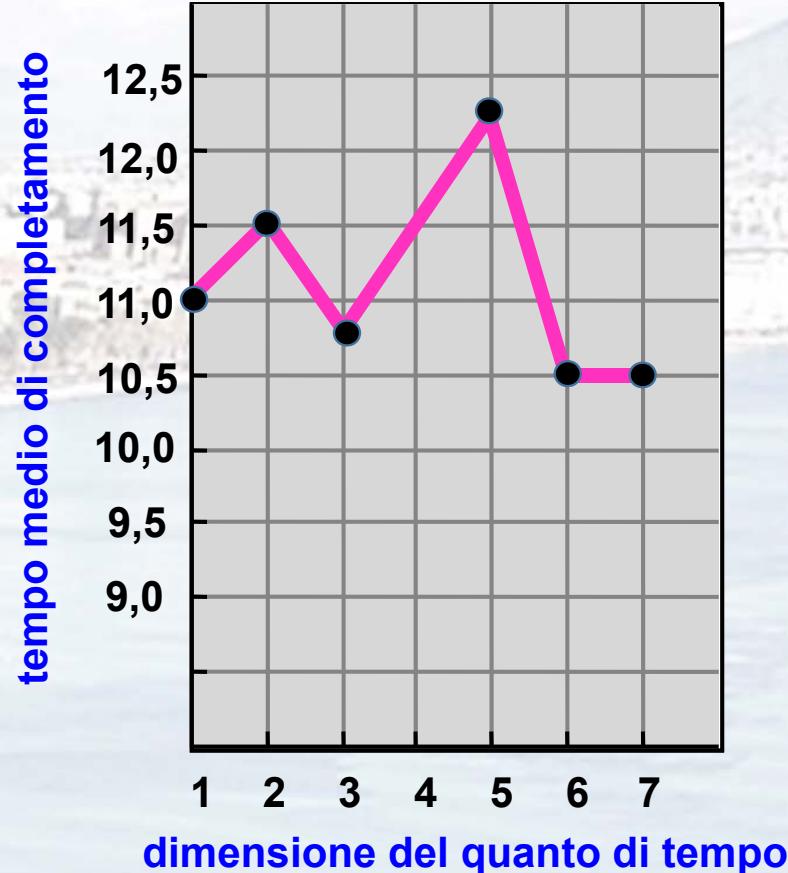
Supponiamo ci sia un solo processo la cui esecuzione richiede 10 unità di tempo

lunghezza del
quanto di tempo cambi
di contesto



walter.balzano@gmail.com

Variazione del tempo di completamento rispetto alla dimensione del quanto di tempo



Processo	Tempo
P_1	6
P_2	3
P_3	1
P_4	7



walter.balzano@gmail.com

Scheduling a code multiple

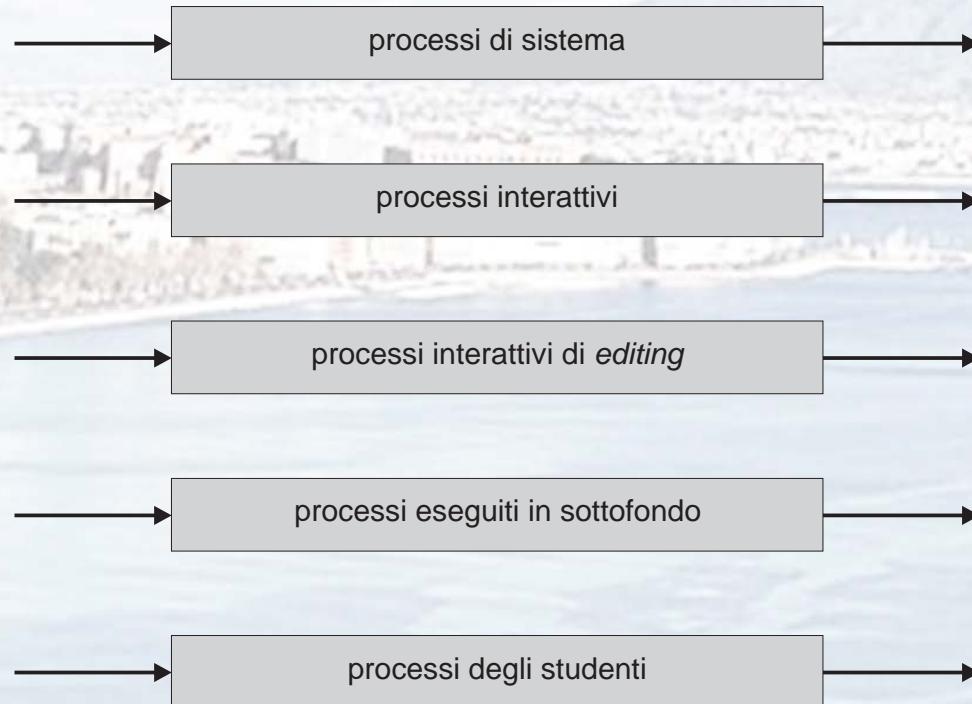
- Una distinzione diffusa è quella che si fa tra i processi che si eseguono:
in **primo piano** (*foreground*), o **interattivi**
in **sottofondo** (*background*), o **a lotti** (*batch*)
- Questi due tipi di processi possono avere diverse necessità di scheduling:
in primo piano: RR
in sottofondo: FCFS
- E' necessario avere uno scheduling tra le code.
 - Scheduling con priorità fissa e prelazione (es: la coda dei processi in primo piano può avere la priorità assoluta sulla coda dei processi in sottofondo). Possibilità di starvation.
 - Possibilità di impostare i quanti di tempo (per ogni coda si stabilisce una parte del tempo d'elaborazione della CPU, che si può a sua volta suddividere tra i processi che la costituiscono: es. 80% alla coda dei processi in primo piano, in RR, 20% a quelli in sottofondo, in FCFS).



walter.balzano@gmail.com

Scheduling a code multiple

priorità più elevata



priorità più bassa



walter.balzano@gmail.com

Scheduling a code multiple con retroazione

- Lo scheduling a code multiple con retroazione (*multilevel feedback queue scheduling*) permette ai processi di spostarsi fra le code. In questo modo si attua una forma d'invecchiamento che impedisce il verificarsi di un'attesa indefinita.
- È caratterizzato dai seguenti parametri:
 - numero di code
 - algoritmo di scheduling per ciascuna coda
 - metodo usato per determinare quando spostare un processo in una coda con priorità maggiore
 - metodo usato per determinare quando spostare un processo in una coda con priorità minore
 - metodo usato per determinare in quale coda si deve mettere un processo quando richiede un servizio.



walter.balzano@gmail.com

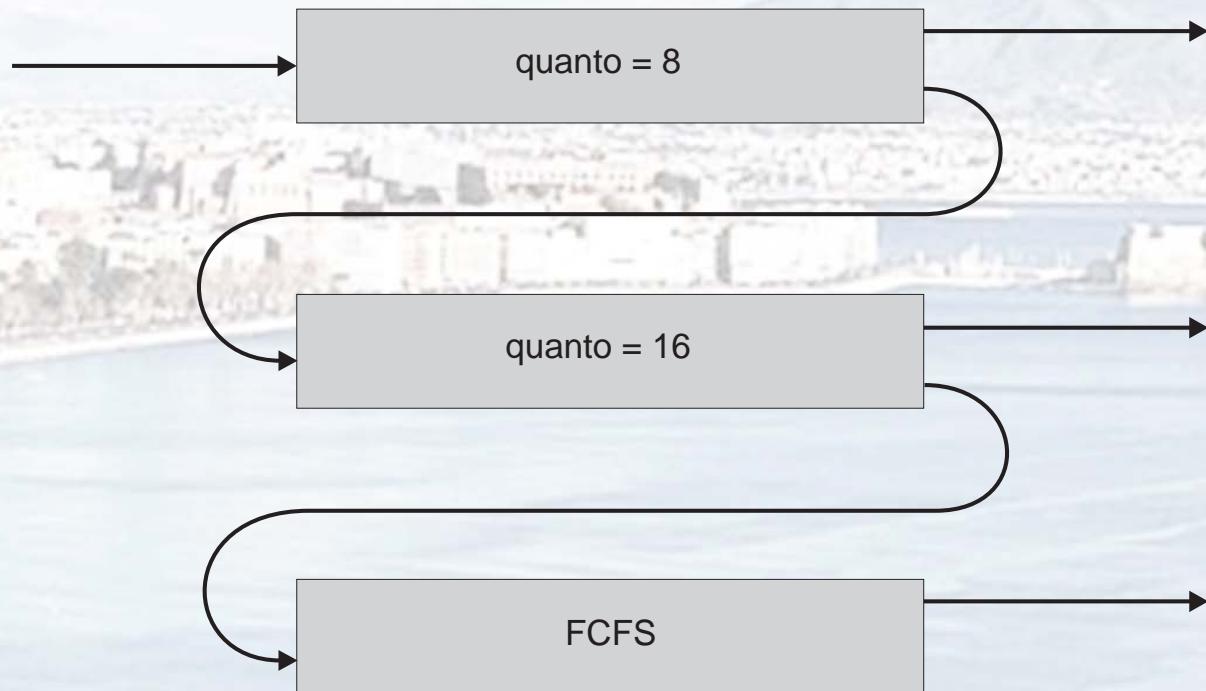
Esempio di code multiple con retroazione

- Tre code:
 - Q_0 – quanto di tempo di 8 millisecondi
 - Q_1 – quanto di tempo di 16 millisecondi
 - Q_2 – FCFS
- Scheduling
 - Un nuovo processo viene assegnato alla coda Q_0 secondo il criterio FCFS e ottiene un quanto di tempo di 8 millisecondi. Se non termina in quel quanto di tempo, viene spostato alla fine della coda Q_1 .
 - Si assegna un quanto di tempo di 16 millisecondi al processo alla testa della coda Q_1 ma se questo non riesce a completare la propria esecuzione, viene sottoposto a prelazione e messo nella coda Q_2 .



walter.balzano@gmail.com

Code multiple con retroazione



walter.balzano@gmail.com

Scheduling per sistemi con più unità d'elaborazione

- Lo scheduling della CPU diventa più complesso nei sistemi con più unità d'elaborazione.
- **Sistemi omogenei**: sistemi nei quali le unità d'elaborazione sono, in relazione alle loro funzioni, identiche.
- **Condivisione del carico** (*load sharing*)
- **Multielaborazione asimmetrica** (*asymmetric multiprocessing*): tutte le attività del sistema sono gestite da un'unica unità d'elaborazione, che quindi assume il ruolo di **unità centrale** (**master server**); le altre unità d'elaborazione eseguono soltanto il codice d'utente.



walter.balzano@gmail.com

- **Sistemi in tempo reale stretto**

(*sistemi real-time hard*): sistemi capaci di garantire il completamento delle funzioni critiche entro un tempo definito (oltre sarebbe inutile)

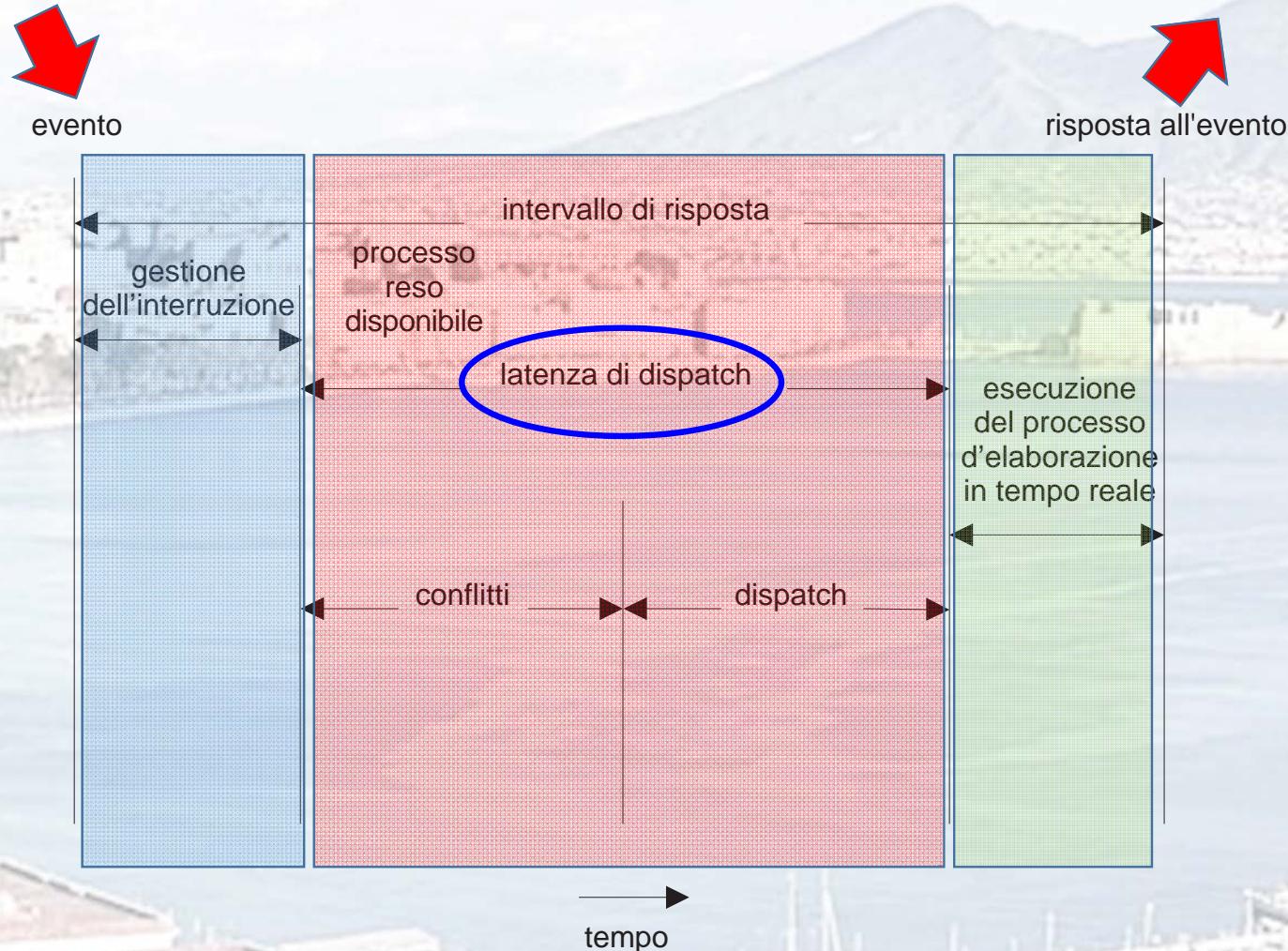
- **Sistemi in tempo reale debole**

(*sistemi real-time soft*): sistemi nei quali i processi critici hanno una priorità maggiore dei processi ordinari.



walter.balzano@gmail.com

Latenza di dispatch



walter.balzano@gmail.com

Valutazione degli algoritmi

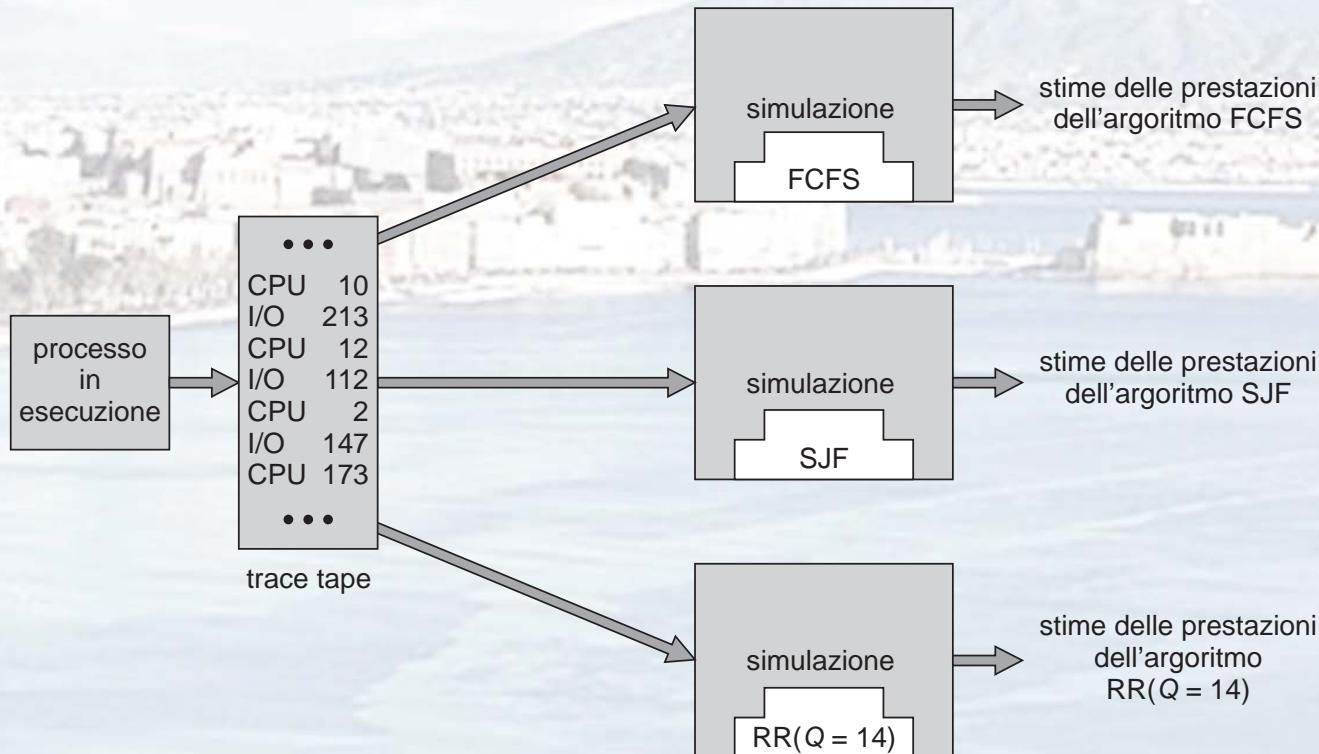
- **Modello deterministico:** tipo di valutazione analitica che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro.
- **Reti di code:** se sono noti le distribuzione degli arrivi e dei servizi si possono calcolare l'utilizzo, la lunghezza media delle code, il tempo medio di attesa, etc....
- **Simulazioni e realizzazione**



walter.balzano@gmail.com

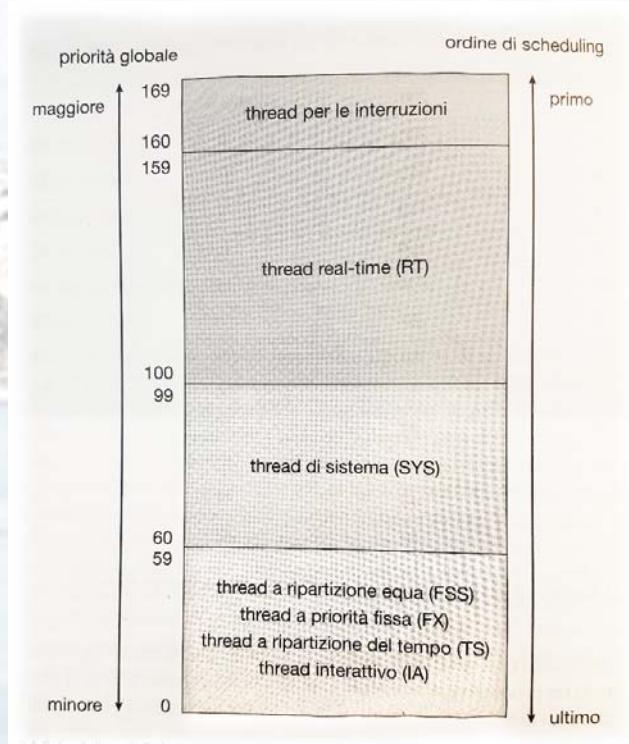
Valutazione di algoritmi di scheduling della CPU tramite una simulazione

Tape trace



walter.balzano@gmail.com

Scheduling nel sistema operativo Solaris



walter.balzano@gmail.com

Priorità nel Sistema Operativo Windows 2000

	REALTIME	HIGH	ABOVE_NORMAL	NORMAL	BELOW_NORMAL	IDLE_PRIORITY
TIME_CRITICAL	31	15	15	15	15	15
HIGHEST	26	15	12	10	8	6
ABOVE_NORMAL	25	14	11	9	7	5
NORMAL	24	13	10	8	6	4
BELOW_NORMAL	23	12	9	7	5	3
LOWEST	22	11	8	6	4	2
IDLE	16	1	1	1	1	1



walter.balzano@gmail.com

Capitolo 7: Sincronizzazione dei processi

- Introduzione
- Problema della sezione critica
- Architetture di sincronizzazione
- Semafori
- Problemi tipici di sincronizzazione
- Regioni critiche
- Monitor
- Sincronizzazione in Windows
- Sincronizzazione in Linux



walter.balzano@gmail.com

Introduzione

- Nei Sistemi Operativi **multi-programmati** diversi processi o thread sono in **esecuzione asincrona** e possono condividere dati.
- I processi cooperanti possono:
 - Condividere uno spazio logico di indirizzi, cioè codice e dati (Thread)
 - Oppure solo dati
- L'accesso concorrente a dati condivisi, se non si adottano politiche di sincronizzazione, può causare incoerenza degli stessi dati.
- Mantenere la coerenza dei dati richiede meccanismi atti ad assicurare un'ordinata esecuzione dei processi cooperanti.



walter.balzano@gmail.com

Esempio: Memoria limitata – Soluzione con memoria condivisa

Supponiamo di usare un zona di memoria condivisa ed usiamo un vettore circolare di grandezza **DIM_VETTORE** e che fa uso di 2 indici per accedere al vettore.

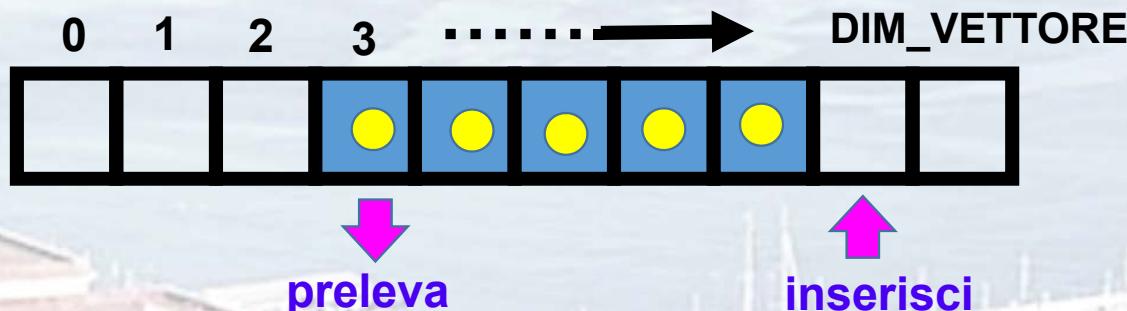
Con l'indice **inserisci** puntiamo alla prima casella disponibile del vettore.

Con l'indice **preleva** puntiamo alla prima posizione occupata del vettore.

Al momento iniziale poniamo a zero sia **inserisci** che **preleva**.

Da tali definizione si ricava che:

- **se** **inserisci == preleva** **Allora** il vettore è vuoto
- **se** $((\text{inserisci} + 1) \% \text{DIM_VETTORE}) == \text{preleva}$ **Allora** vettore pieno
- Max elementi nel vettore = **DIM_VETTORE - 1**



walter.balzano@gmail.com

Esempio: Memoria limitata – Soluzione con memoria condivisa (2)

La soluzione del problema dei produttori e dei consumatori con memoria limitata consente la presenza contemporanea nel vettore di $n - 1$ elementi.

Si supponga di voler modificare il codice del produttore-consumatore aggiungendo una variabile intera **counter** (contatore) inizializzata a 0 e che si incrementa ogniqualvolta si preleva un elemento dal vettore.



walter.balzano@gmail.com

Memoria limitata

Dati condivisi:

```
#define BUFFER_SIZE 10
typedef struct {
    ...
} item;
item buffer[BUFFER_SIZE ];
int in = 0;
int out = 0;
int counter = 0;
```



walter.balzano@gmail.com

Memoria limitata

- Processo produttore

```
item nextProduced;  
  
while (1)  
{  
    while (counter == BUFFER_SIZE)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```



walter.balzano@gmail.com

Memoria limitata

- Processo consumatore

```
item nextConsumed;  
  
while (1)  
{  
    while (counter == 0)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```



walter.balzano@gmail.com

Memoria limitata - Processo consumatore

```
item nextConsumed;  
  
while (1)  
{  
    while (counter  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```

P1

```
item nextConsumed;
```

P2

```
while (1)  
{  
    while (counter == 0)
```

```
    item nextConsumed;
```

P3

```
    while (1)  
{  
    while (counter == 0)
```

```
        ; /* do nothing */
```

```
        nextConsumed = buffer[out];  
        out = (out + 1) % BUFFER_SIZE;  
        counter--;
```

```
}
```



walter.balzano@gmail.com

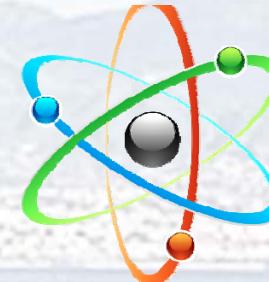
Memoria limitata

- Le istruzioni

counter++;
counter--;

devono essere eseguite in modo atomico.

- Un'operazione è eseguita in modo atomico se si esegue completamente senza interruzione.



walter.balzano@gmail.com

Memoria limitata

L'istruzione “**count++**” si può codificare in linguaggio macchina come:

```
register1 = counter  
register1 = register1 + 1  
counter = register1
```

L'istruzione “**count --**” può essere realizzata come:

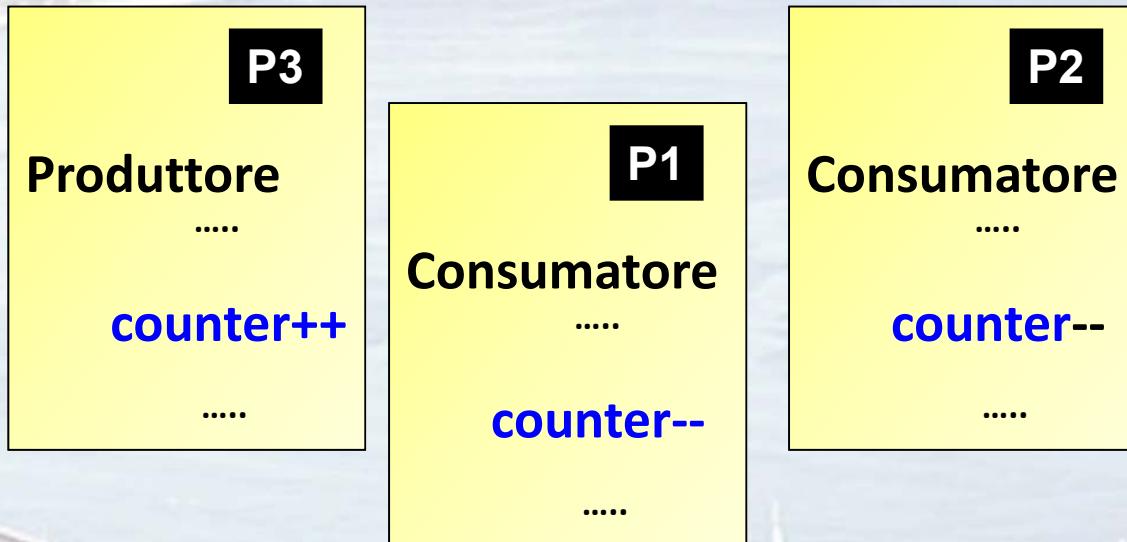
```
register2 = counter  
register2 = register2 - 1  
counter = register2
```



walter.balzano@gmail.com

Memoria limitata

Se sia il **produttore** sia il **consumatore** tentano di aggiornare **concorrentemente** il registro, le istruzioni del linguaggio macchina possono risultare intercalate (*interleaved*).



walter.balzano@gmail.com

Memoria limitata

- Si supponga che il valore della variabile counter sia inizialmente 5. Una sequenza è:

producer: **register1 = counter** (*register1 = 5*)

producer: **register1 = register1 + 1** (*register1 = 6*)

consumer: **register2 = counter** (*register2 = 5*)

consumer: **register2 = register2 - 1** (*register2 = 4*)

producer: **counter = register1** (*counter = 6*)

consumer: **counter = register2** (*counter = 4*)

- Il valore della variabile potrebbe essere 4 o 6, ma il risultato corretto dovrebbe essere 5.



walter.balzano@gmail.com

Race condition

- **Race condition:** situazione in cui più processi accedono e modificano gli stessi dati in modo concorrente, e i risultati dipendono dall'ordine degli accessi.
- Per prevenire questa situazione, i processi concorrenti devono essere sincronizzati.



walter.balzano@gmail.com

Problema della sezione critica

- Si considerino ***n*** processi concorrenti.
- Ciascun processo ha un segmento di codice chiamato **sezione critica (*critical section*)** nel quale il processo può modificare variabili comuni.
- Problema: quando un processo è in esecuzione nella propria sezione critica, non si deve consentire a nessun altro processo di essere in esecuzione nella propria sezione critica.



walter.balzano@gmail.com

Soluzione al problema della sezione critica

Nella gestione della sezione critica, le garanzie principali da fornire sono tre:

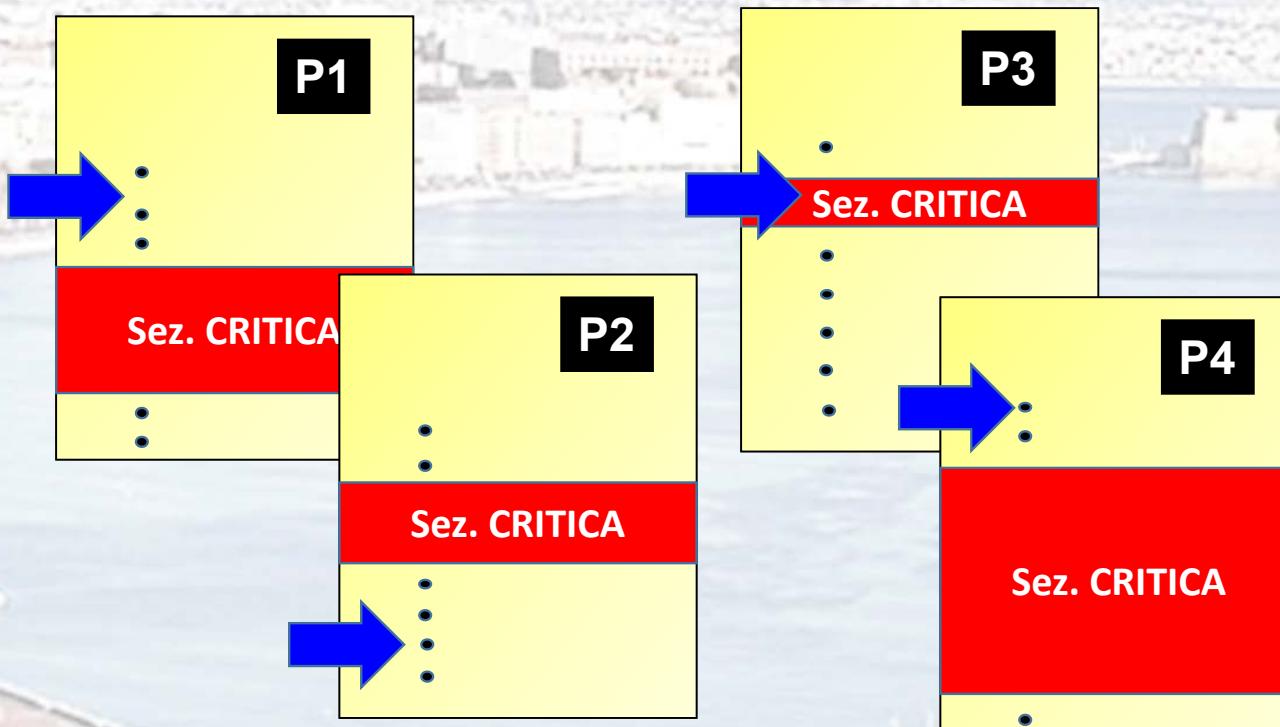
- 1. Mutua Esclusione.**
- 2. Progresso.**
- 3. Attesa limitata.**



walter.balzano@gmail.com

Soluzione al problema della sezione critica

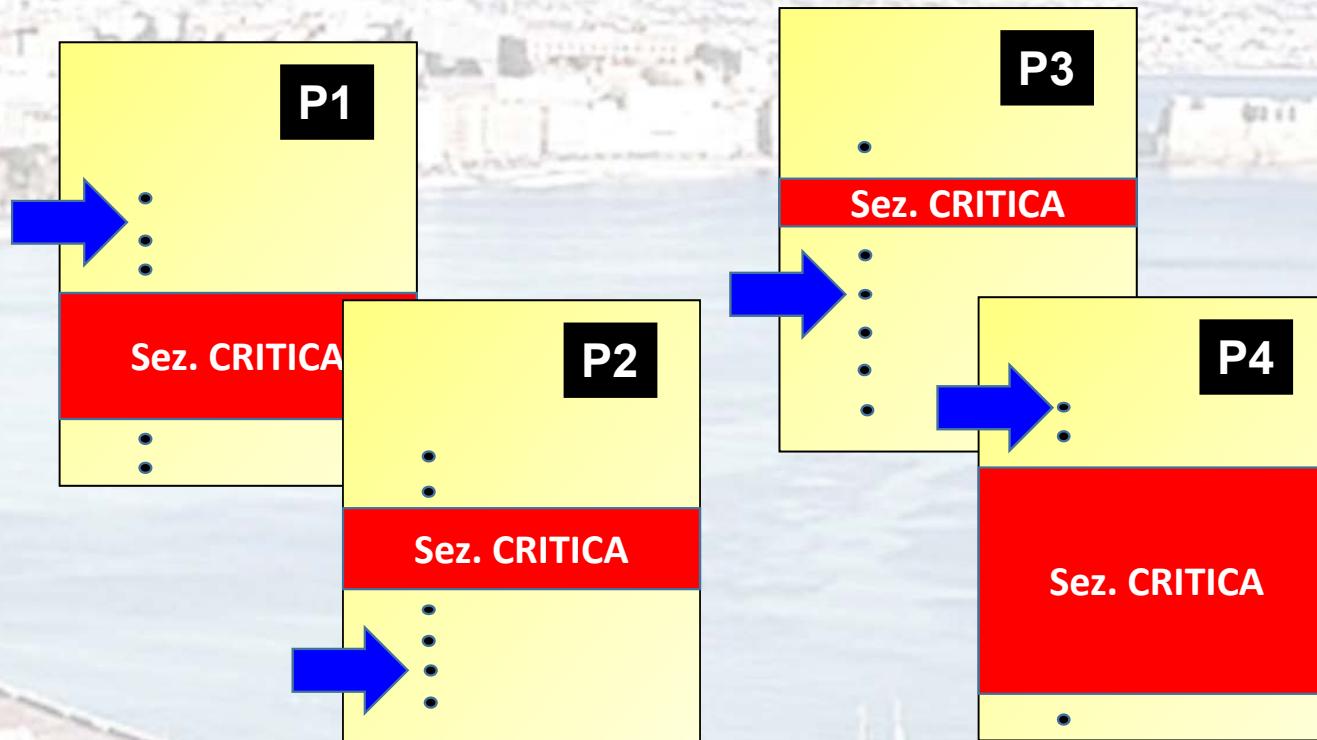
1. Mutua Esclusione. Se il processo P_i è in esecuzione nella sua sezione critica, nessun altro processo può essere in esecuzione nella propria sezione critica.



walter.balzano@gmail.com

Soluzione al problema della sezione critica

2. Progresso. se nessun processo è in esecuzione nella sua sezione critica, solo i processi che desiderano entrare nella propria sezione critica possono partecipare alla decisione su chi sarà il prossimo ad entrare in sezione critica e tale decisione non può essere ritardata indefinitamente..

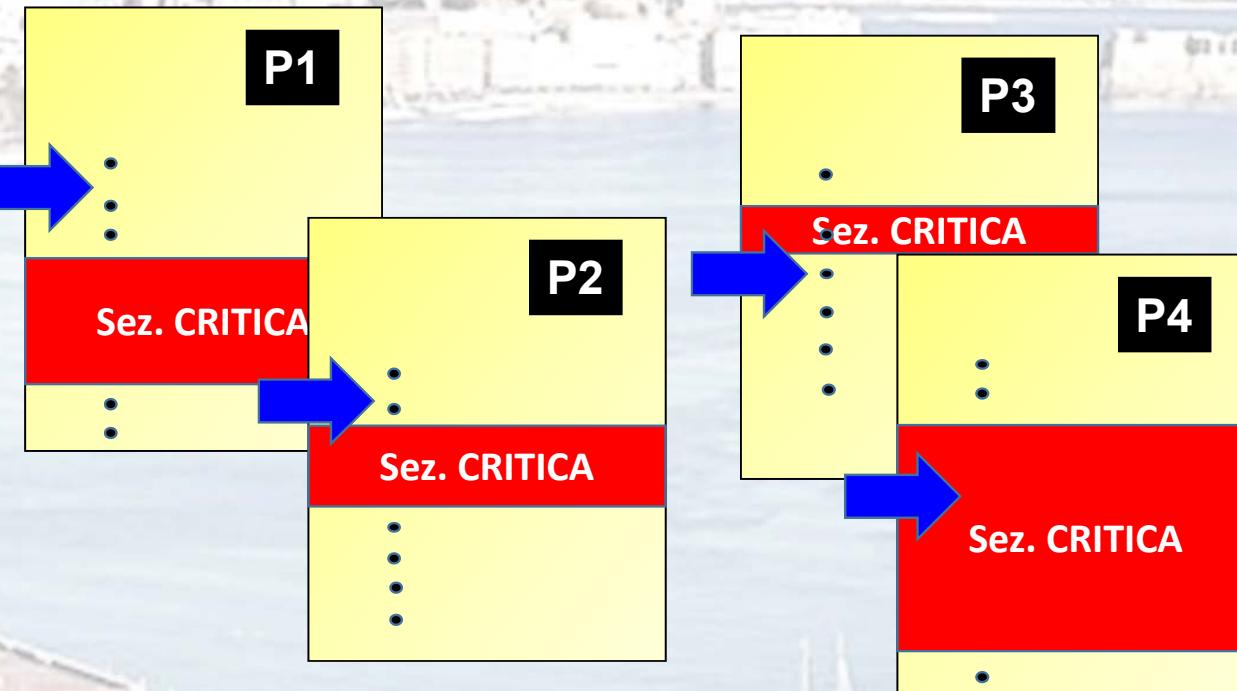


walter.balzano@gmail.com

Soluzione al problema della sezione critica

3. Attesa limitata. Se un processo ha già richiesto l'ingresso nella sua sezione critica, esiste un limite al numero di volte che si consente ad altri processi di entrare nelle rispettive sezioni critiche prima che si accordi la richiesta del primo processo.

- Si suppone che ogni processo sia eseguito a una velocità diversa da zero
- Non si può fare alcuna ipotesi sulla velocità relativa degli n processi.



walter.balzano@gmail.com

Primi tentativi di risolvere il problema

- Solo 2 processi, P_0 e P_1
- Struttura generale del processo P_i
(l'altro processo è P_j)

do {

sezione d'ingresso

sezione critica

sezione d'uscita

sezione non critica

} while (1);

- I processi possono condividere alcune variabili comuni per sincronizzare le loro azioni.



walter.balzano@gmail.com

Algoritmo 1

- **Variabili condivise:**

- int turno;
inizialmente turno = 0
- turno = i $\Rightarrow P_i$ entra nella propria sezione critica

- **Processo P_i ,**
do {

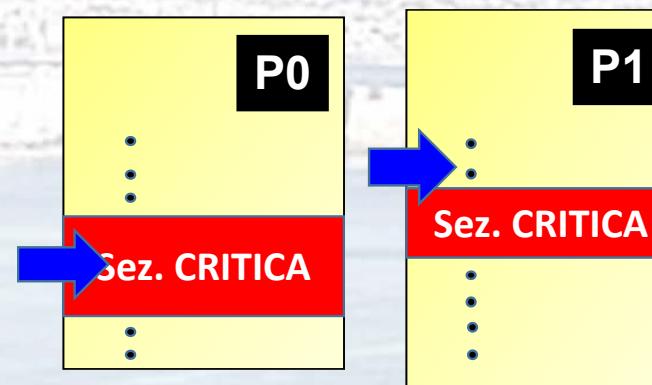
- while (turno != i);**

- sezione critica**

- turno = j;**

- sezione non critica**

- } while (1);



- Soddisfa la mutua esclusione
ma non il requisito di progresso



walter.balzano@gmail.com

Algoritmo 2

- **Variabili condivise:**

- boolean pronto[2]; inizialmente pronto [0] = pronto [1] = false.
- pronto [i] = true $\Rightarrow P_i$ pronto a entrare nella sua sezione critica

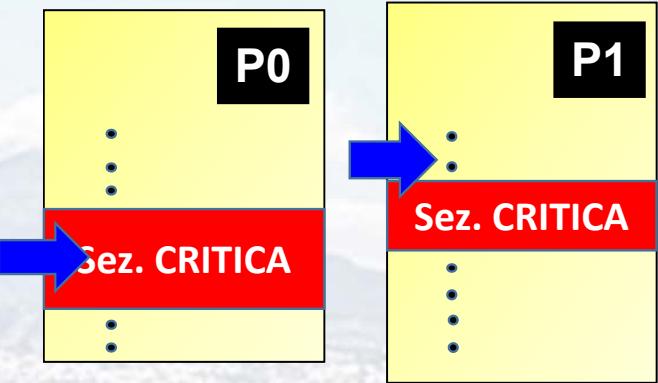
- **Processo P_i**
do {

```
pronto[i] := true;  
while (pronto[j]) ;
```

sezione critica

```
pronto[i] = false;  
sezione non critica  
} while (1);
```

- Soddisfa la mutua esclusione ma non il requisito di progresso.



walter.balzano@gmail.com

Algoritmo 3

- Combina le variabili condivise degli algoritmi 1 e 2.

- Processo P_i

```
do {
```

```
    pronto[i] = true;
```

```
    turno = j;
```

```
    while (pronto[j] && turno = j) ;
```

```
        sezione critica
```

```
        pronto[i] = false;
```

```
        sezione non critica
```

```
    } while (1);
```

- Soddisfa tutti i tre requisiti; risolve il problema della sezione critica per i due processi.



walter.balzano@gmail.com

Algoritmo del fornaio

Sezione critica per n processi

- Prima di entrare nella sua sezione critica, il processo riceve un numero.
Chi detiene il numero più basso entra nella sezione critica.
- Se i processi P_i e P_j ricevono lo stesso numero, se $i < j$, allora P_i è servito per primo; altrimenti viene servito P_j .
- Lo schema di numerazione genera sempre numeri in ordine crescente;
es.: 1,2,3,3,3,4,5...



walter.balzano@gmail.com

Algoritmo del fornaio

```
do {
```

```
    scelta[i] = true;  
    numero[i] = max(numero[0], ..., numero [n – 1])+1;  
    scelta[i] = false;
```

```
    for (j = 0; j < n; j++) {
```

```
        while (scelta[j]) ;
```

```
        while ((numero [j] != 0) &&
```

```
        {
```

```
            (numero [j] < numero [i]) ||
```

```
            ((numero[j] == numero[i]) && (j<i))
```

```
        } );
```

```
}
```

sezione critica

```
    numero [i] = 0;
```

sezione non critica

```
} while (1);
```



walter.balzano@gmail.com

Architetture di sincronizzazione

Controlla e modifica il contenuto
di una parola di memoria in modo atomico.

```
boolean TestAndSet(boolean &obiettivo)
```

```
{
```

```
    boolean valore = obiettivo;  
    obiettivo = true;  
    return valore;
```

```
}
```



walter.balzano@gmail.com

Mutua esclusione con TestAndSet

Dati condivisi:

```
boolean blocco = false;
```

- Processo Pi

```
do {  
    while (TestAndSet(blocco)) ;  
    sezione critica  
    blocco = false;  
    sezione non critica  
} while (1)
```



walter.balzano@gmail.com

Architetture di sincronizzazione

L'istruzione **swap** agisce sul contenuto di due parole di memoria; come per **TestAndSet** è anch'essa eseguita in modo atomico.

```
void Swap(boolean &a, boolean &b)
{
    boolean temp = a;
    a = b;
    b = temp;
}
```



walter.balzano@gmail.com

Mutua esclusione con Swap

- Dati condivisi (inizializzati a false):
boolean blocco;
boolean waiting[n];
- Processo P_i

```
do {  
    chiave = true;  
    while (chiave == true)  
        Swap(blocco,chiave);  
    sezione critica  
    blocco = false;  
    sezione non critica  
} while (1)
```



walter.balzano@gmail.com

Semafori

- Strumento di sincronizzazione che non richiede attesa attiva.
- Un semaforo S è una variabile intera.
- A questa variabile si può accedere solo tramite due operazioni atomiche predefinite

wait (S):

```
while S≤ 0 do no-op;  
S--;
```

signal (S):

```
S++;
```



walter.balzano@gmail.com

Sezione critica di n processi

- Dati condivisi:
semaphore mutex; //inizialmente mutex = 1

- Processo P_i :

```
do {  
    wait(mutex);  
    sezione critica  
    signal(mutex);  
    sezione non critica  
} while (1);
```



walter.balzano@gmail.com

Realizzazione di semafori

- Definire il semaforo come una struttura del linguaggio C

```
typedef struct {  
    int valore;  
    struct processo *L;  
} semaforo;
```

- Due semplici operazioni:
 - block sospende il processo che la invoca.
 - wakeup(P) riprende l'esecuzione di un processo P bloccato.



walter.balzano@gmail.com

Realizzazione

Le operazioni dei semafori si possono definire ora come segue:

wait(S):

```
S.valore--;
if (S.valore < 0) {
    aggiungi questo processo a S.L;
    block;
}
```

signal(S):

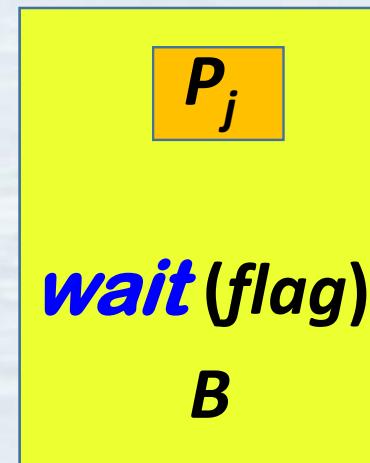
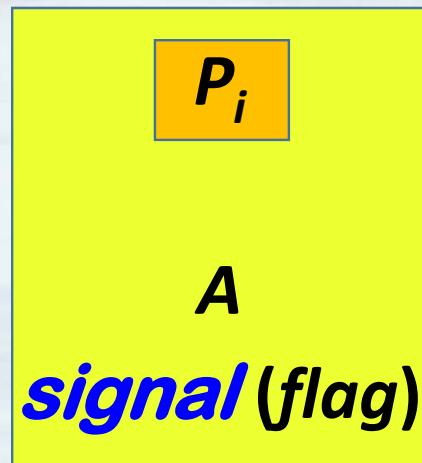
```
S.valore++;
if (S.valore <= 0) {
    togli un processo P da S.L;
    wakeup(P);
}
```



walter.balzano@gmail.com

Semaforo come strumento generale di sincronizzazione

- Esegue B in P_j solo dopo che A è stato eseguito in P_i
- Usa un **flag** inizializzato a 0
- Codice:



walter.balzano@gmail.com

Stallo e attesa indefinita

- Stallo (*deadlock*): situazione in cui due o più processi attendono indefinitamente un evento che può essere causato solo da uno dei processi in attesa.
- Siano S e Q due semafori inizializzati a 1

P_0
wait(S);
wait(Q);
:
signal(S);
signal(Q)

P_1
wait(Q);
wait(S);
:
signal(Q);
signal(S);

- Attesa indefinita (*starvation*): situazione d'attesa indefinita nella coda di un semaforo.



walter.balzano@gmail.com

Tipi di semafori

- **Semaforo contatore** (*counting semaphore*): il suo valore intero può variare in un dominio logicamente non limitato.
- **Semaforo binario** (*binary semaphore*): il suo valore intero può essere soltanto 0 o 1; può essere più semplice da realizzare.
- È possibile implementare un semaforo contatore *S* in termini di un semaforo binario.



walter.balzano@gmail.com

Realizzazione di S come semaforo binario

- Strutture dati:

semaforo_binario S1, S2;

int C;

- Inizializzazione:

S1 = 1

S2 = 0

C = valore iniziale

del semaforo contatore S



walter.balzano@gmail.com

Realizzazione di S

- Operazione *wait*

```
wait(S1);
C--;
if (C < 0) {
    signal(S1);
    wait(S2);
}
signal(S1);
```

- Operazione *signal*

```
wait(S1);
C++;
if (C <= 0)
    signal(S2);
else
    signal(S1);
```



walter.balzano@gmail.com

Problemi tipici di sincronizzazione

- Problema dei produttori e dei consumatori con memoria limitata
- Problema dei lettori e degli scrittori
- Problema dei cinque filosofi



walter.balzano@gmail.com

Produttori e consumatori con memoria limitata



- Dati condivisi
semaforo piene, vuote, mutex;

Inizialmente:
piene = 0, vuote = n, mutex = 1



walter.balzano@gmail.com

Produttori e consumatori con memoria limitata

- Processo produttore

```
do {
```

```
...
```

```
produce un elemento in appena_prodotto
```

```
...
```

```
wait(vuote);  
wait(mutex);
```

```
...
```

```
inserisci appena_prodotto in vettore
```

```
...
```

```
signal(mutex);  
signal(piene);
```

```
} while (1);
```



walter.balzano@gmail.com

Produttori e consumatori con memoria limitata - Processo consumatore

```
do {  
    wait(piene)  
    wait(mutex);  
    ...  
    rimuovi un elemento da vettore e  
    inseriscilo in da_consumare  
    ...  
    signal(mutex);  
    signal(vuote);  
    ...  
    consuma l'elemento contenuto in  
    da_consumare  
    ...  
} while (1);
```



walter.balzano@gmail.com

Problema dei lettori e degli scrittori

- Dati condivisi

semaforo mutex, scrittura;

Inizialmente

mutex = 1,

scrittura = 1,

numlettori = 0



walter.balzano@gmail.com

Problema dei lettori e degli scrittori

Processo scrittore

wait(scrittura);

...

esegui l'operazione di scrittura

...

signal(scrittura);



walter.balzano@gmail.com

Problema dei lettori e degli scrittori

Processo lettore

```
wait(mutex);
numlettori++;
if (numlettori == 1)
    wait(scrittura);
signal(mutex);
```

...

esegui l'operazione di lettura

...

```
wait(mutex);
numlettori --;
if (numlettori == 0)
    signal(scrittura);
signal(mutex);
```



walter.balzano@gmail.com

Problema dei cinque filosofi

Possibili Soluzioni:

1. Solo 4 filosofi possono stare contemporaneamente a tavola
2. Un filosofo può prendere le sue bacchette solo se sono entrambe disponibili (questa operazione va eseguita in una sezione critica).
3. Si adotta una soluzione asimmetrica: un filosofo dispari prende prima la bacchetta di sinistra e poi quella di destra; invece un filosofo pari prende prima la bacchetta di destra e poi quella di sinistra



walter.balzano@gmail.com

Problema dei cinque filosofi

Sono in 6 ☺
Non mangiano
riso ma il
problema è lo
stesso !!!



- Dati condivisi
semaforo bacchetta[5];
Inizialmente tutti i valori sono 1



walter.balzano@gmail.com

Problema dei cinque filosofi

- **Filosofo i:**

```
do {  
    wait(bacchetta[i])  
    wait(bacchetta[(i + 1) % 5])  
    ...  
    mangia  
    ...  
    signal(bacchetta[i]);  
    signal(bacchetta[(i + 1) % 5]);  
    ...  
    pensa  
    ...  
} while (1);
```



walter.balzano@gmail.com

Regioni critiche

- Costrutto di sincronizzazione ad alto livello
- Una variabile condivisa **v** di tipo **T** è dichiarata come:
v: shared T
- Alla variabile **v** si può accedere solo dall'interno di un'istruzione
region v when B do S
dove **B** è un'espressione booleana.
- Mentre si esegue l'istruzione **S** nessun altro processo può accedere alla variabile.



walter.balzano@gmail.com

Regioni critiche

- Regioni che si riferiscono alla stessa variabile condivisa si escludono vicendevolmente .
- Quando un processo vuole accedere alla variabile condivisa **v** nella regione critica, si valuta l'espressione booleana **B**, se risulta vera si esegue l'istruzione **S**; altrimenti il processo rilascia la mutua esclusione ed è ritardato fino a che **B** diventa vera e nessun altro processo si trova nella regione associata a **v**.



walter.balzano@gmail.com

Esempio

Problema dei produttori e consumatori con memoria limitata

- Dati condivisi:

```
struct vettore {  
    elemento gruppo[n];  
    int contatore, inserisci, preleva;  
}
```



walter.balzano@gmail.com

Esempio - processo produttore

Problema dei produttori e consumatori con memoria limitata

- Il processo produttore inserisce

appena_prodotto nel vettore condiviso

```
region vettore when( contatore < n) {  
    gruppo[inserisci] = appena_prodotto;  
    inserisci := (inserisci + 1) % n;  
    contatore++;  
}
```



walter.balzano@gmail.com

Esempio – processo consumatore

Problema dei produttori e consumatori con memoria limitata

Il processo consumatore rimuove un elemento dal vettore condiviso e lo inserisce in **da_consumare**

```
region vettore when (contatore > 0) {  
    da_consumare = gruppo[preleva];  
    preleva = (preleva + 1) % n;  
    contatore --;  
}
```



walter.balzano@gmail.com

Monitor

- Costrutto di sincronizzazione di alto livello, che consente la condivisione sicura di un tipo di dato astratto fra processi concorrenti.

```
monitor nome_monitor
{
    dichiarazioni di variabili condivise
    procedure body P1 (...) {
        ...
    }
    procedure body P2 (...) {
        ...
    }
    procedure body Pn (...) {
        ...
    }
    {
        codice d'inizializzazione
    }
}
```



walter.balzano@gmail.com

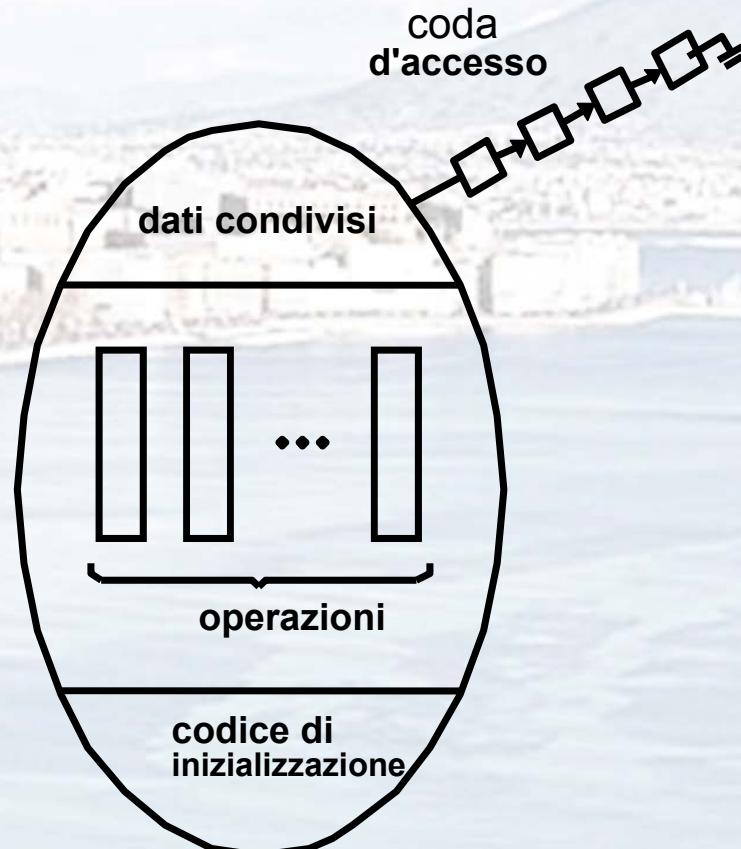
Monitor

- Per consentire a un processo di attendere all'interno di un monitor, occorre definire una variabile condizionale come
condition x, y;
- Le uniche operazioni eseguibili su una variabile **condition** sono **wait** e **signal**.
 - L'operazione
x.wait();
implica che il processo che la invoca rimanga sospeso fino a che un altro processo non invoca l'operazione
 - L'operazione **x.signal()** risveglia esattamente un processo sospeso. Se non esistono processi sospesi, l'operazione **signal** non ha alcun effetto.



walter.balzano@gmail.com

Schema di un monitor



walter.balzano@gmail.com

Sincronizzazione in Linux

Nelle prime versioni di Linux non era gestita la prelazione
La tecnica di sincronizzazione più semplice nel kernel di
Linux è l'itero atomico rappresentato mediante il tipo di dato
opaco

atomic_t.

Le operazioni matematiche che usano interi atomici
vengono eseguite senza interruzione.

Altre tecniche più sofisticate disponibili nel kernel:

1. **mutex_lock()** (per entrare nella sez.critica)
2. **mutex_unlock()** (in uscita dalla sezione critica)



walter.balzano@gmail.com

Sincronizzazione in Windows

- Quando il nucleo di tale sistema operativo accede a una risorsa globale in un sistema **a singola CPU**, disabilita temporaneamente le interruzioni aventi procedure di gestione che potrebbero accedere alla stessa risorsa globale.
- In un sistema **con più unità d'elaborazione**, protegge l'accesso alle risorse globali con i semafori ad attesa attiva (**spinlocks**).
- Per la sincronizzazione fuori dal nucleo, il sistema operativo offre gli **oggetti dispatcher** (*dispatcher objects*) che permettono ai thread di sincronizzarsi servendosi di diversi meccanismi, inclusi mutex, semafori ed eventi.
- Gli eventi sono un meccanismo di sincronizzazione che si può usare in modo simile alle variabili condizionali.



walter.balzano@gmail.com

Capitolo 8: Stallo dei processi

- Modello del sistema
- Caratterizzazione
- Metodi di gestione
- Prevenzione
- Rilevamento
- Ripristino
- Approccio combinato



walter.balzano@gmail.com

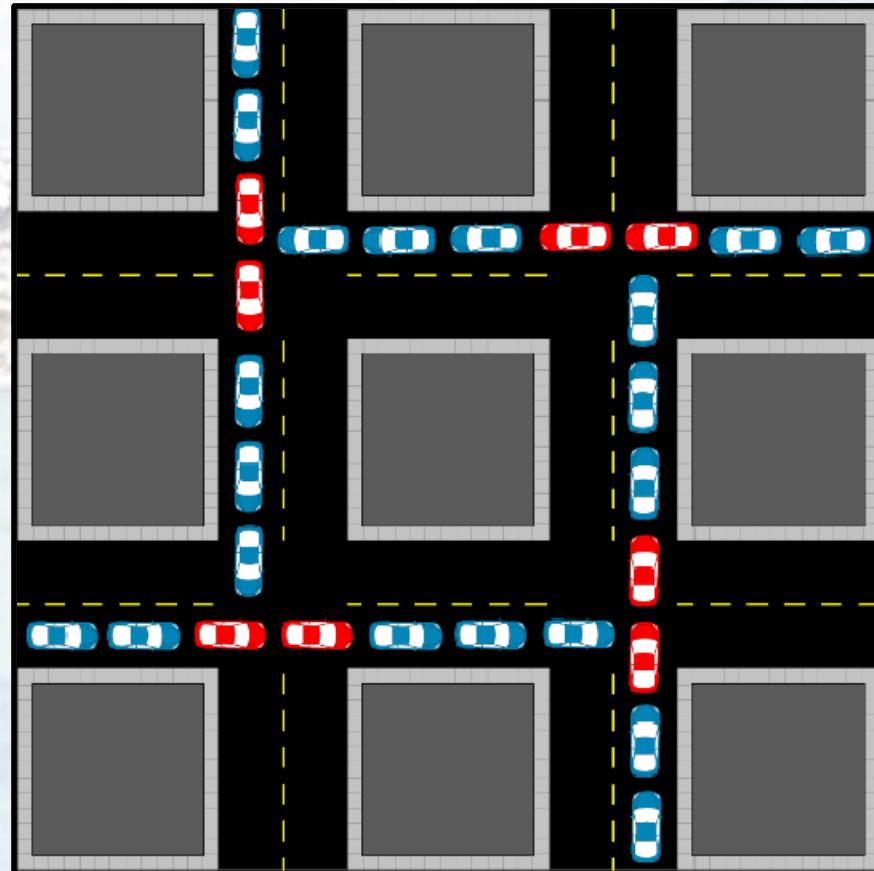
Capitolo 8: Stallo dei processi

Legge del Kansas del XX secolo: quando due treni convergono ad un incrocio, ambedue devono arrestarsi, e nessuno dei due può ripartire prima che l'altro si sia allontanato



walter.balzano@gmail.com

Capitolo 8: Stallo dei processi



walter.balzano@gmail.com

Il problema dello stallo

- Un insieme di processi bloccati, in cui ciascun processo detiene una risorsa e attende di accedere a una risorsa in possesso di un altro processo.

- **Esempio 1**

- Il sistema dispone di 2 unità a nastri.
- P_1 e P_2 possiedono ciascuno una di queste unità e ciascuno richiede un'altra unità a nastri.

- **Esempio 2**

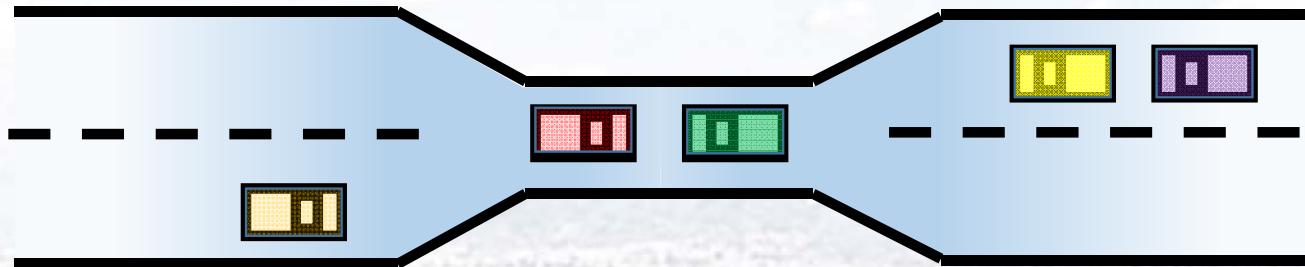
- semafori A e B , inizializzati a 1

P_0	P_1
<i>wait (A);</i>	<i>wait(B)</i>
<i>wait (B);</i>	<i>wait(A)</i>



walter.balzano@gmail.com

Esempio dell'attraversamento del ponte



- Traffico in una sola direzione.
- Ciascuna sezione del ponte può essere considerata una risorsa.
- Se si verifica uno stallo, può essere risolto se una macchina fa retromarcia: prelazione di risorse e ristabilimento di uno stato sicuro (**rollback**).
- Più macchine potrebbero dover fare retromarcia, in caso di stallo.
- È possibile si verifichi un'attesa indefinita (**starvation**).



walter.balzano@gmail.com

Modello del sistema

- Tipi di risorse R_1, R_2, \dots, R_m
*cicli di CPU, spazio di memoria,
dispositivi di I/O*
- Ciascun tipo di risorsa R_i ha W_i istanze.
- Ciascun processo utilizza una risorsa nella seguente sequenza:

- richiesta
- uso
- rilascio



walter.balzano@gmail.com

Caratterizzazione delle situazioni di stallo

Si può avere una situazione di stallo solo se si verificano contemporaneamente le seguenti quattro condizioni.

- **Mutua esclusione:** solo un processo alla volta può utilizzare una risorsa.
- **Possesso e attesa:** un processo in possesso di almeno una risorsa attende di acquisire risorse già in possesso di altri processi.
- **Impossibilità di prelazione:** una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, dopo aver terminato il proprio compito.
- **Attesa circolare:** deve esistere un insieme $\{P_0, P_1, \dots, P_n\}$ di processi tale che P_0 attende una risorsa posseduta da P_1 , P_1 attende una risorsa posseduta da P_2 , ..., P_{n-1} attende una risorsa posseduta da P_n , e P_n attende una risorsa posseduta da P_0 .



walter.balzano@gmail.com

Grafo di assegnazione delle risorse

Un insieme di vertici **V** e di archi **E**.

Tipi di Vertici:

- $P = \{P_1, P_2, \dots, P_n\}$ = tutti i processi.
- $R = \{R_1, R_2, \dots, R_m\}$ = tutti i tipi di risorsa.

Tipi di Archi:

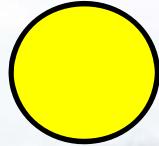
- Arco di richiesta:
arco orientato $P_1 \rightarrow R_j$
- Arco di assegnazione:
arco orientato $R_j \rightarrow P_i$



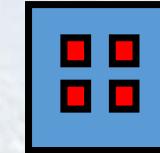
walter.balzano@gmail.com

Grafo di assegnazione delle risorse - 2

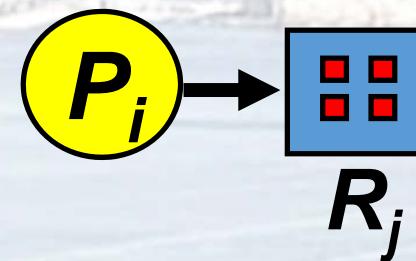
Processo



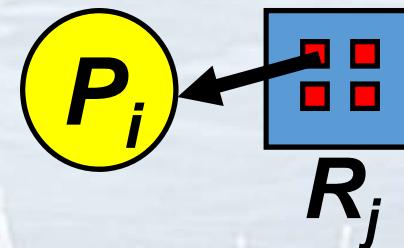
Tipo di risorsa
con 4 istanze



P_i richiede un'istanza
del tipo di risorsa R_j

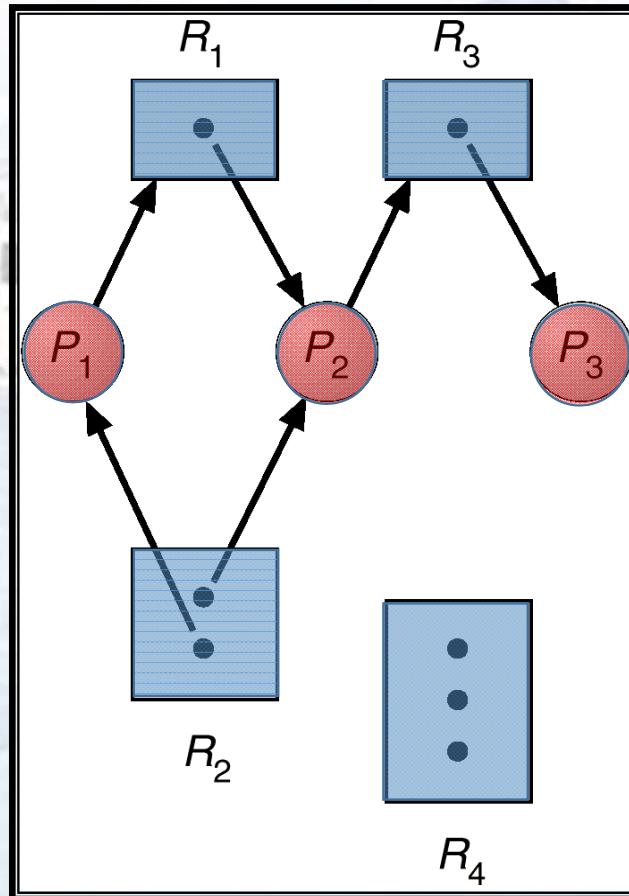


P_i possiede un'istanza
del tipo di risorsa R_j



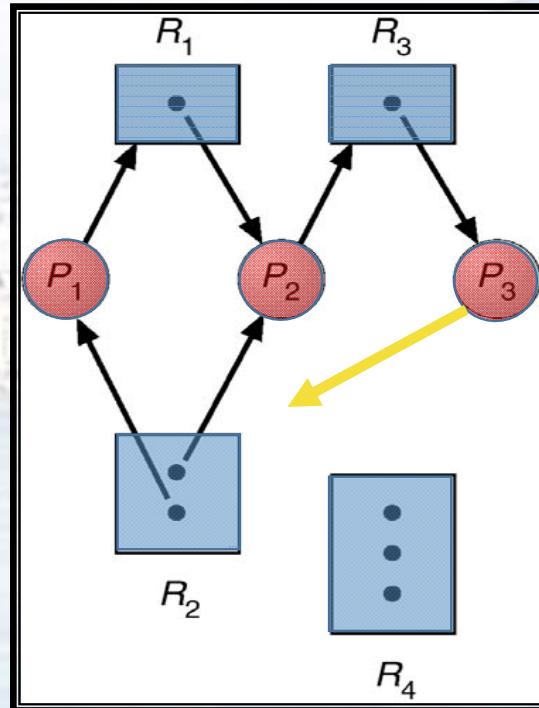
walter.balzano@gmail.com

Grafo di assegnazione delle risorse



walter.balzano@gmail.com

Grafo di assegnazione delle risorse con stallo



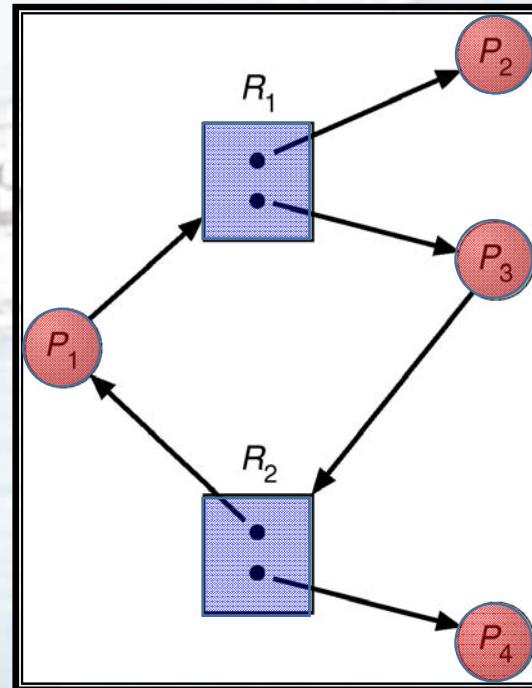
Ci sono due Cicli:

1. $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
2. $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



walter.balzano@gmail.com

Grafo di assegnazione delle risorse con un ciclo ma senza stallo



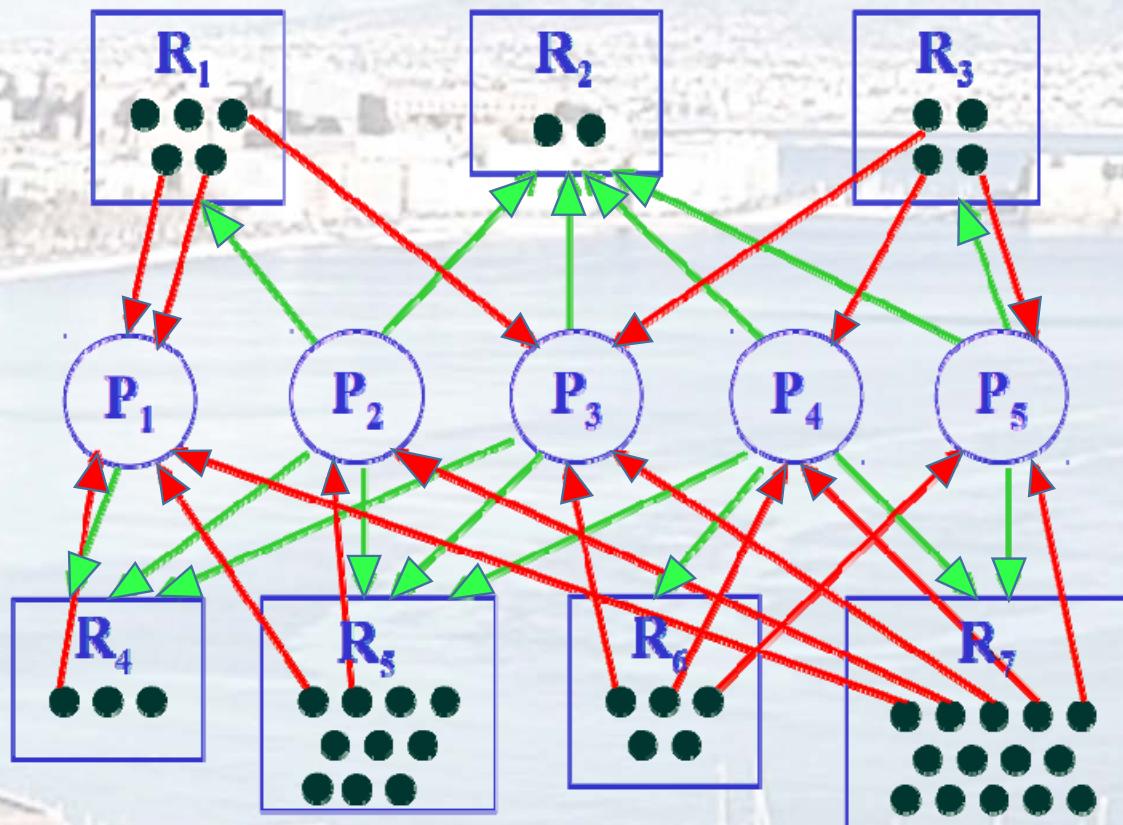
Nota: il processo P_4 **usa** ma **non attende** e quindi non è vera la condizione di «possesso e attesa»

walter.balzano@gmail.com

Grafo di assegnazione delle risorse con risorse multiple

in rosso le assegnazioni

in verde le richieste



walter.balzano@gmail.com

Osservazioni

- **Se il grafo NON contiene cicli:**

- non si verificano situazioni di stallo.

- **Se il grafo contiene un ciclo:**

- se c'è solo un'istanza per tipo di risorsa, allora si verifica una situazione di stallo.
 - se vi sono più istanze per tipo di risorsa, allora c'è la **possibilità** che si verifichi una situazione di stallo.



walter.balzano@gmail.com

Metodi per la gestione delle situazioni di stallo

- Assicurare che il sistema non entri *mai* in stallo
- Consentire al sistema di entrare in stallo, individuarlo e quindi eseguire il ripristino.
- Ignorare del tutto il problema “fingendo” che le situazioni di stallo non possano mai verificarsi nel sistema; è la soluzione usata nella maggior parte dei sistemi operativi, compreso UNIX.

Se il deadlock si verifica ogni 50 anni allora applichiamo l’Algoritmo dello Struzzo
(far finta di niente) ☺



walter.balzano@gmail.com

Prevenire le situazioni di stallo

Si può prevenire il verificarsi di uno stallo assicurando che almeno una delle quattro condizioni necessarie non si verifichi.

- **Mutua esclusione:**

- non richiesta per le risorse condivisibili; deve invece **valere per le risorse non condivisibili**.

- **Possesso e attesa:**

- occorre garantire che ogni volta che un processo richiede una risorsa, non ne possegga altre.
- Ogni processo, prima di iniziare la propria esecuzione, deve richiedere tutte le risorse che gli servono, e queste gli devono essere assegnate; oppure occorre permettere a un processo di richiedere risorse solo se non ne possiede.
- Basso utilizzo delle risorse; possibile attesa indefinita.



walter.balzano@gmail.com

Prevenire le situazioni di stallo - 2

• Impossibilità di prelazione:

- Se un processo che possiede una o più risorse ne richiede un'altra che non gli si può assegnare immediatamente, allora si esercita la prelazione su tutte le risorse attualmente in suo possesso.
- Le risorse si aggiungono alla lista delle risorse che il processo sta attendendo.
- Il processo viene nuovamente avviato solo quando può ottenere sia le vecchie risorse sia quelle che sta richiedendo.

• Attesa circolare:

- impone un ordinamento totale all'insieme di tutti i tipi di risorse e un ordine crescente di numerazione per le risorse richieste da ciascun processo.



walter.balzano@gmail.com

Evitare le situazioni di stallo

Un metodo alternativo per evitare le situazioni di stallo consiste nel richiedere ulteriori informazioni sui modi di richiesta delle risorse.

- Il modello più utile e più semplice richiede che ciascun processo dichiari il *numero massimo* delle risorse di ciascun tipo di cui necessita.
- L'algoritmo per evitare lo stallo esamina dinamicamente lo stato di assegnazione delle risorse per garantire che non possa esistere una condizione di attesa circolare.
- Lo *stato di assegnazione delle risorse* è definito dal numero di risorse disponibili e assegnate, e dalle richieste massime dei processi.



walter.balzano@gmail.com

Stato sicuro

- Uno stato si dice sicuro se il sistema è in grado di assegnare risorse a ciascun processo (fino al suo massimo) in un certo ordine e impedire il verificarsi di uno stallo.
- Un sistema si trova in uno stato sicuro solo se esiste una sequenza sicura.
- La sequenza $\langle P_1, P_2, \dots, P_n \rangle$ è sicura se per ogni P_i , le richieste che P_i può ancora fare si possono soddisfare impiegando le risorse attualmente disponibili ed in + le risorse possedute da tutti i P_j , con $j < i$.
 - Se le risorse necessarie al processo P_i non sono immediatamente disponibili, allora P_i può attendere che tutti i P_j abbiano finito.
 - A quel punto, P_i può ottenere tutte le risorse di cui ha bisogno, completare il compito assegnato, restituire le risorse assegnate, e terminare.
 - Quando P_i termina, P_{i+1} può ottenere le risorse richieste, e così via.



walter.balzano@gmail.com

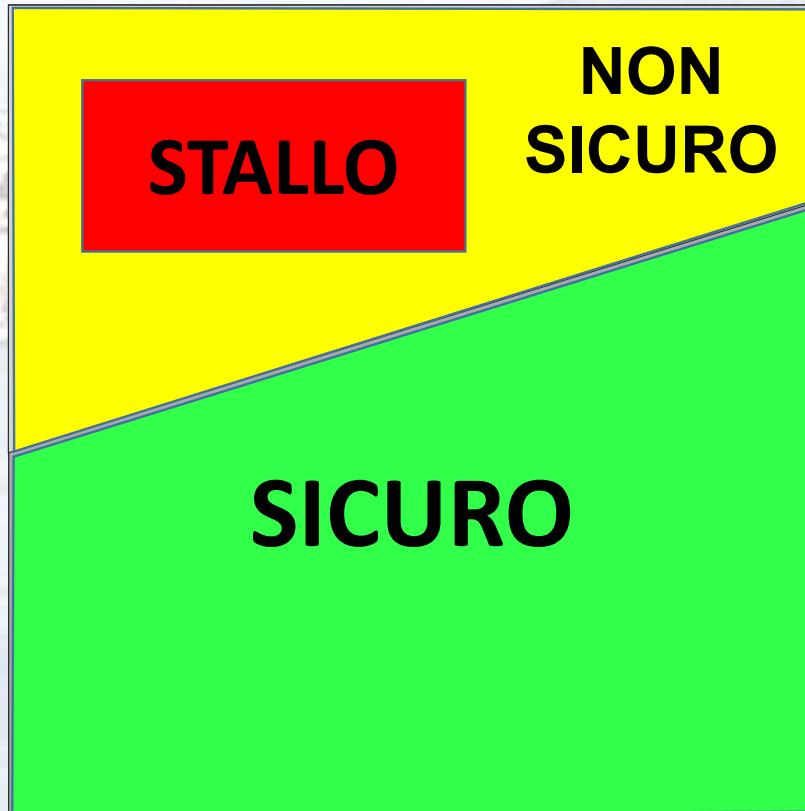
Osservazioni

- Se un sistema è in uno **stato sicuro**
⇒ non si verificano situazioni di stallo.
- Se un sistema è in uno **stato non sicuro**
⇒ possibilità di stallo.
- Evitare lo stallo
⇒ assicurare che il sistema non si trovi mai in uno stato non sicuro.



walter.balzano@gmail.com

Spazi degli stati sicuri, non sicuri e di stallo



walter.balzano@gmail.com

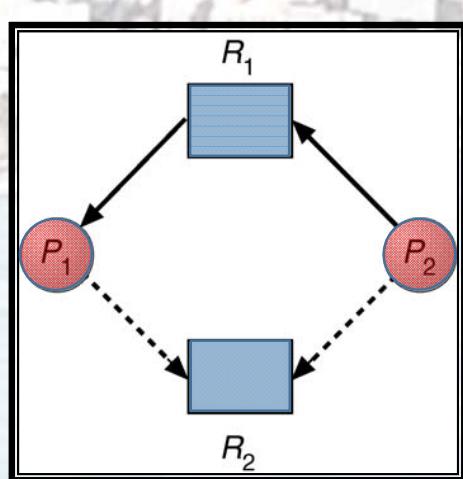
Algoritmo con grafo di assegnazione delle risorse

- **Arco di reclamo** $P_i \rightarrow R_j$ indica che il processo P_i può richiedere la risorsa R_j ; rappresentato con una linea tratteggiata. 
- Quando un processo richiede una risorsa, l'arco di reclamo diventa un **arco di richiesta**. 
- Quando un processo rilascia una risorsa, l'arco di assegnazione ridiventa un arco di reclamo.
- Le risorse devono essere reclamate a priori nel sistema.

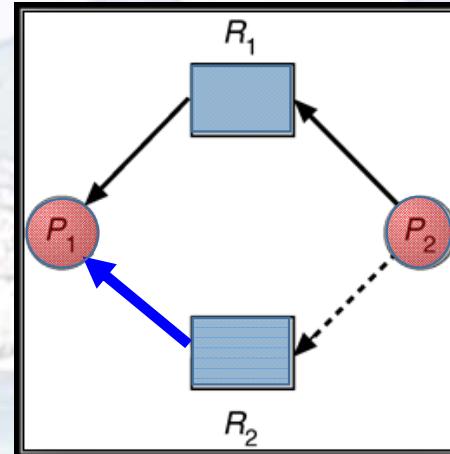


walter.balzano@gmail.com

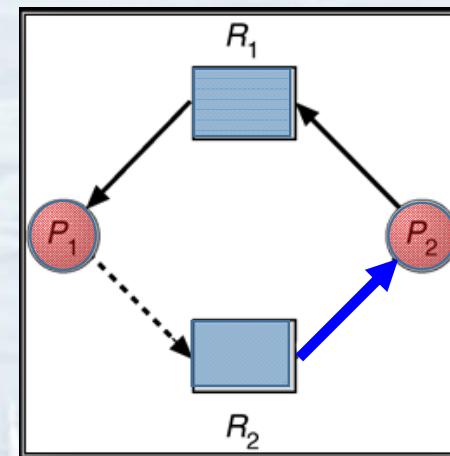
Grafo di assegnazione delle risorse per evitare le situazioni di stallo



Prima P1



Prima P2



walter.balzano@gmail.com

Grafo di assegnazione delle risorse per evitare le situazioni di stallo



**DeadLock
Possibile**



DeadLock



walter.balzano@gmail.com

Grafo di assegnazione delle risorse per evitare le situazioni di stallo



walter.balzano@gmail.com

Grafo di assegnazione delle risorse per evitare le situazioni di stallo



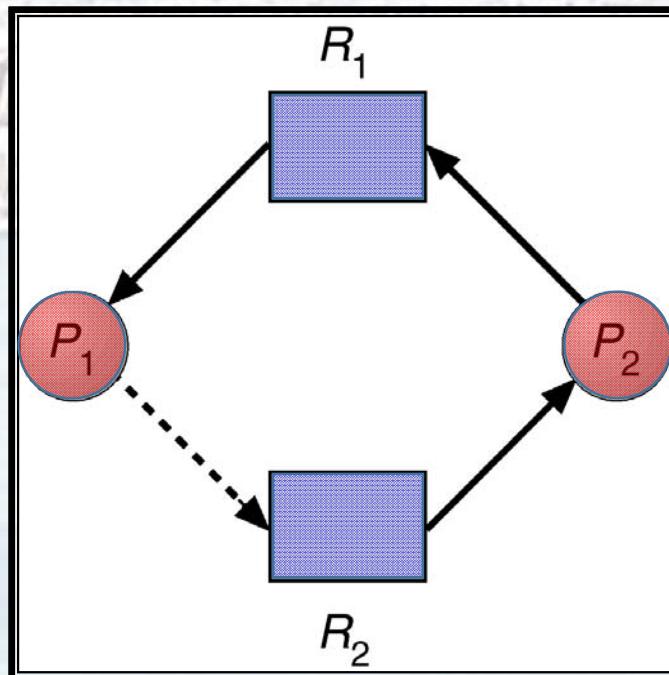
walter.balzano@gmail.com

Grafo di assegnazione delle risorse per evitare le situazioni di stallo



walter.balzano@gmail.com

Uno stato non sicuro in un grafo di assegnazione delle risorse



walter.balzano@gmail.com

Algoritmo del banchiere

- Istanze multiple.
- Ciascun processo deve dichiarare a priori il numero massimo delle istanze di ciascun tipo di risorsa di cui necessita.
- Quando un processo richiede una risorsa, può dover attendere.
- Quando un processo ottiene tutte le sue risorse, deve restituirle entro un intervallo di tempo definito.



walter.balzano@gmail.com

Strutture dati per l'algoritmo del banchiere

Sia n il numero di processi del sistema e m il numero dei tipi di risorsa.

- **Disponibili:**

un vettore di lunghezza m indica il numero delle istanze disponibili per ciascun tipo di risorsa. **Se $\text{disponibili}[j] = k$, significa che sono disponibili k istanze del tipo di risorsa R_j .**

- **Massimo:**

una matrice $n \times m$ definisce la richiesta massima di ciascun processo. **Massimo $[i,j] = k$ significa che il processo P_i può richiedere al più k istanze del tipo di risorsa R_j .**

- **Assegnate:**

una matrice $n \times m$ definisce il numero delle istanze di ciascun tipo di risorsa attualmente assegnate a ciascun tipo di processo. **Assegnate $[i,j] = k$ significa che al processo P_i sono assegnate k istanze del tipo di risorsa R_j .**

- **Necessità:**

una matrice $n \times m$ indica la necessità residua di risorse relativa a ogni processo. **Necessità $[i,j] = k$ significa che il processo P_i può aver bisogno di altre k istanze di R_j per completare il suo compito.**

$$\text{Necessità } [i,j] = \text{Massimo}[i,j] - \text{Assegnate } [i,j].$$



walter.balzano@gmail.com

Algoritmo di verifica della sicurezza

1. Siano *Lavoro* e *Fine* vettori di lunghezza rispettivamente m e n .

Inizializza:

Lavoro = Disponibili

Fine[i] = falso per $i - 1, 3, \dots, n$.

2. Cerca un indice i tale che:

(a) *Fine*[i] = falso

(b) *Necessità*, \leq *Lavoro*

se tale i non esiste, esegue il passo 4.

3. *Lavoro* := *Lavoro* + Assegnate,

Fine[i] := vero

torna al passo 2.

4. Se *Fine*[i] == vero per ogni i , allora il sistema è in uno stato sicuro.



walter.balzano@gmail.com

Algoritmo di richiesta delle risorse

Richieste = vettore delle richieste per il processo P_i .

Se $\text{Richieste}_i[j] = k$ allora il processo P_i richiede k istanze del tipo di risorsa R_j .

1. Se $\text{Richieste}_i \leq \text{Necessità}_i$, esegue il passo 2. Altrimenti, riporta una condizione d'errore, poiché il processo ha superato il numero massimo di richieste.
2. Se $\text{Richieste}_i \leq \text{Disponibili}$, esegue il passo 3. Altrimenti P_i deve attendere, poiché le risorse non sono disponibili.
3. Il sistema simula l'assegnazione al processo P_i delle risorse richieste modificando lo stato di assegnazione delle risorse:

$\text{Disponibili} := \text{Disponibili} - \text{Richieste}_i;$

$\text{Assegnate} := \text{Assegnate}_i + \text{Richieste}_i;$

$\text{Necessità}_i := \text{Necessità}_i - \text{Richieste}_i.$

- se lo stato è sicuro \Rightarrow le risorse sono assegnate a P_i
- se lo stato è non sicuro $\Rightarrow P_i$ deve attendere, e si ripristina il vecchio stato di assegnazione delle risorse.



walter.balzano@gmail.com

Un esempio di algoritmo del banchiere

- 5 processi da P_0 a P_4 ; 3 tipi di risorse: il tipo di risorse A ha 10 istanze, B ne ha 5 e C 7.
- Si supponga che all'istante T_0 si sia verificata la seguente istantanea:

	<u>Assegnate</u>	<u>Massimo</u>	<u>Disponibili</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	



walter.balzano@gmail.com

Un esempio (Cont.)

- Il contenuto della matrice *Necessità* è definito come *Massimo – Assegnate*.

Necessità

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

- Il sistema si trova attualmente in uno stato sicuro poiché la sequenza $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ soddisfa i criteri di sicurezza.



walter.balzano@gmail.com

Un esempio (Cont.)

- Si verifica la condizione **Richieste \leq Disponibili** (vale a dire, $(1,0,2) \leq (3,3,2)$ \Rightarrow vera).

	<u>Allocation</u>			<u>Necessità</u>			<u>Disponibili</u>		
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	1	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

- L'esecuzione dell'algoritmo di verifica della sicurezza mostra che la sequenza $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ soddisfa il requisito di sicurezza.
- Può essere soddisfatta la richiesta da parte di P_4 di $(3,3,0)$?
- Può essere soddisfatta la richiesta da parte di P_0 di $(0,2,0)$?



walter.balzano@gmail.com

Svantaggi per algoritmo del banchiere

**L'applicazione dell'algoritmo del
banchiere presuppone una
conoscenza completa del sistema...**



walter.balzano@gmail.com

Rilevamento delle situazioni di stallo

Se un sistema non si avvale di un algoritmo per prevenire o per evitare le situazioni di stallo, è possibile che una situazione di stallo si verifichi effettivamente. In tal caso il sistema deve fornire i seguenti algoritmi:

- **un algoritmo che esamini lo stato del sistema per stabilire se si è verificato uno stallo**
- **un algoritmo che ripristini il sistema dalla condizione di stallo.**



walter.balzano@gmail.com

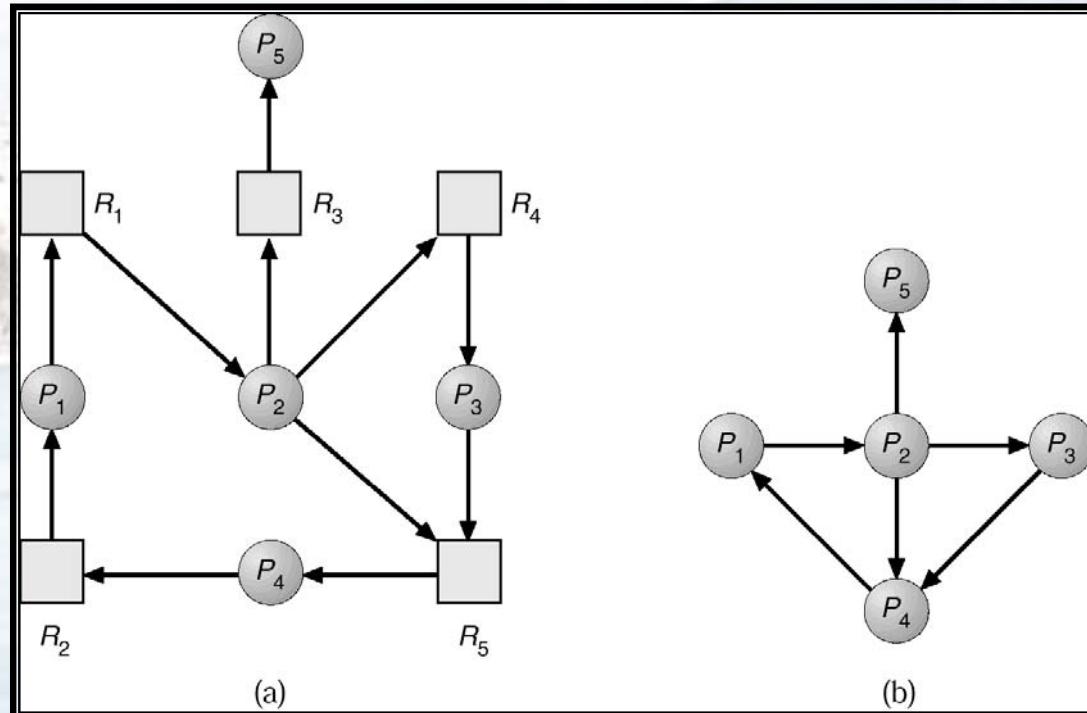
Istanza singola di ciascun tipo di risorsa

- Grafo d'attesa
 - I nodi sono processi.
 - $P_i \rightarrow P_j$ se P_i è in attesa di P_j .
- Per individuare le situazioni di stallo il sistema deve conservare il grafo d'attesa e invocare periodicamente un algoritmo che cerchi un ciclo all'interno del grafo.
- L'algoritmo per il rilevamento di un ciclo all'interno di un grafo richiede un numero di operazioni dell'ordine di n^2 , dove con n si indica il numero dei vertici del grafo.



walter.balzano@gmail.com

Grafo di assegnazione delle risorse e grafo d'attesa



Grafo di assegnazione
delle risorse

Grafo d'attesa
corrispondente



walter.balzano@gmail.com

Più istanze di ciascun tipo di risorsa

- **Disponibili.** Vettore di lunghezza m che indica il numero delle istanze disponibili per ciascun tipo di risorsa.
- **Assegnate.** Matrice $n \times m$ che definisce il numero delle istanze di ciascun tipo di risorse correntemente assegnate a ciascun processo.
- **Richieste.** Matrice $n \times m$ che indica la richiesta attuale di ciascun processo. Se $\text{Richieste}[i_j] = k$, significa che il processo P_i sta richiedendo altre k istanze del tipo di risorsa R_j .



walter.balzano@gmail.com

Algoritmo di rilevamento

1. Siano *Lavoro* e *Fine* due vettori di lunghezza rispettivamente m e n . Inizializza:

- (a) *Lavoro* = *Disponibili*
- (b) per $i = 1, 2, \dots, n$, se *Assegnate* $\neq 0$, allora *Fine*[i] = falso; altrimenti, *Fine*[i] = vero.

2. Cerca un indice i tale che:

- (a) *Fine*[i] == falso
- (b) *Richieste* $_i \leq Lavoro$

Se tale i non esiste, esegue il passo 4.



walter.balzano@gmail.com

Algoritmo di rilevamento (Cont.)

3. *Lavoro* := *Lavoro* + *Assegnate_i*,

Fine[i] = *vero*

torna al passo 2.

4. Se *Fine[i]* == *falso*, per qualche i , $1 \leq i \leq n$, allora il sistema è in stallo. Inoltre, se *Fine[i]* == *falso*, allora il processo P_i è in stallo.

Tale algoritmo richiede un numero di operazioni dell'ordine di $m \times n^2$ per controllare se il sistema è in stallo.



walter.balzano@gmail.com

Esempio di algoritmo di rilevamento

- Cinque processi da P_0 a P_4 ; tre tipi di risorse: A (7 istanze), B (2 istanze) e C (6 istanze).
- Si supponga di avere, all'istante T_0 :

Assegnate Richieste Disponibili

	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

La sequenza $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ da come risultato $Fine[i] = \text{vero}$ per ogni i .



walter.balzano@gmail.com

Esempio (Cont.)

- Si supponga ora che il processo P_2 richieda un'altra istanza di tipo C.

Richieste

	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Stato del sistema

- Ora il sistema è in stallo Anche se si possono reclamare le risorse possedute dal processo P_0 , il numero delle risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi.
- Si verifica uno stallo composto dei processi P_1 , P_2 , P_3 e P_4 .



walter.balzano@gmail.com

Uso dell'algoritmo di rilevamento

- Per sapere quando è necessario ricorrere all'algoritmo di rilevamento, occorre considerare i seguenti fattori:
 - **frequenza** (presunta) con la quale si verifica uno stallo
 - **numero dei processi** che sarebbero influenzati da tale stallo
- Non è conveniente richiedere l'algoritmo di rilevamento in momenti arbitrari, poiché nel grafo delle risorse possono coesistere molti cicli e, normalmente, non si può dire quale fra i tanti processi in stallo abbia “causato” lo stallo.



walter.balzano@gmail.com

Ripristino da situazioni di stallo: terminazione di processi

- Terminazione di tutti i processi in stallo.
- Terminazione di un processo alla volta fino all'eliminazione del ciclo di stallo.
- In quale ordine effettuare la terminazione?
 - ✓ Priorità dei processi.
 - ✓ Tempo trascorso dalla computazione e tempo ancora necessario per completare i compiti assegnati ai processi.
 - ✓ Quantità e tipo di risorse impiegate dai processi.
 - ✓ Quantità di ulteriori risorse di cui i processi hanno ancora bisogno per completare i propri compiti.
 - ✓ Numero di processi che si devono terminare.
 - ✓ Tipo di processi: interattivi o a lotti.



walter.balzano@gmail.com

Ripristino da situazioni di stallo: prelazione di risorse

- **Selezione di una vittima:**
minimizzare i costi.
- **Rollback:**
ristabilimento di un precedente
stato sicuro, dal quale il processo
può essere riavviato.
- **Attesa indefinita (*starvation*):**
può accadere che si scelga sempre
lo stesso processo come vittima.



walter.balzano@gmail.com

Approccio combinato per la gestione dello stallo

- La combinazione dei tre approcci di base

- prevenire
- evitare
- rilevare

**consente l'uso dell'approccio ottimale
per ciascuna delle risorse del sistema.**

- Partizione delle risorse in classi ordinate gerarchicamente.
- Uso della tecnica più appropriata per la gestione dello stallo all'interno di ciascuna classe.



walter.balzano@gmail.com

Capitolo 9: Gestione della memoria

- **Introduzione**
- **Avvicendamento dei processi
(swapping)**
- **Assegnazione contigua della memoria**
- **Paginazione**
- **Segmentazione**
- **Segmentazione con paginazione**



walter.balzano@gmail.com

Introduzione

- Per essere eseguito, un programma deve essere caricato nella memoria e inserito all'interno di un processo.
- Coda d'ingresso (*input queue*): l'insieme dei processi presenti nei dischi che attendono d'essere trasferiti nella memoria per essere eseguiti.
- Prima di essere eseguito, un programma passa attraverso numerose fasi.



walter.balzano@gmail.com

Associazione di istruzioni e dati a indirizzi di memoria

Generalmente, l'associazione di istruzioni e dati a indirizzi di memoria si può compiere in qualsiasi fase del seguente percorso.

- **Compilazione:** se nella fase di compilazione si sa dove il processo risiederà nella memoria, si può generare codice assoluto (*absolute code*); se, in un momento successivo, la locazione iniziale cambiasse, sarebbe necessario ricompilare il codice.
- **Caricamento:** Se nella fase di compilazione non è possibile sapere in che punto della memoria risiederà il processo, il compilatore deve generare codice rilocabile (*relocatable code*).
- **Esecuzione:** Se durante l'esecuzione il processo può essere spostato da un segmento di memoria a un altro, si deve ritardare l'associazione degli indirizzi fino alla fase d'esecuzione. Per realizzare questo schema sono necessarie specifiche caratteristiche dell'architettura.



walter.balzano@gmail.com

Fasi di elaborazione per un programma utente



walter.balzano@gmail.com

Spazio di indirizzi logici e fisici

- Il concetto di **spazio degli indirizzi logici** associato a uno **spazio degli indirizzi fisici** separato è fondamentale per una corretta gestione della memoria.
 - **Indirizzo LOGICO (*logical address*)**: indirizzo generato dalla CPU; detto anche **indirizzo virtuale (*virtual address*)**.
 - **Indirizzo FISICO (*physical address*)**: indirizzo visto dall'unità di memoria.
- I metodi di associazione degli indirizzi nelle fasi di compilazione e di caricamento producono indirizzi logici e fisici identici. Con i metodi di associazione nella fase d'esecuzione, invece, gli indirizzi logici non coincidono con gli indirizzi fisici.



walter.balzano@gmail.com

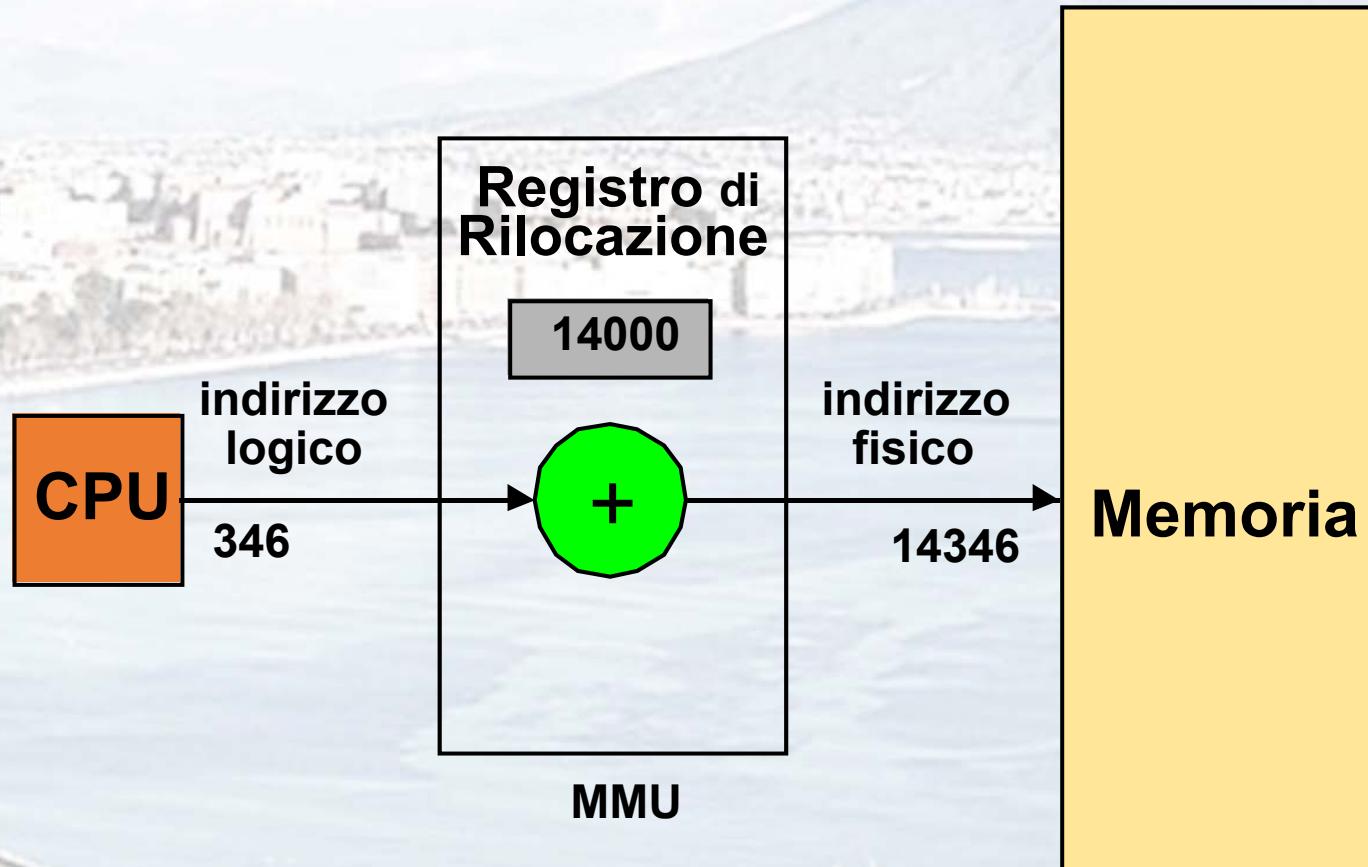
Unità di gestione della memoria (MMU)

- La **MMU** è un dispositivo per l'associazione, nella fase di esecuzione, degli indirizzi virtuali agli indirizzi fisici.
- Nello schema **MMU**, quando un processo utente genera un indirizzo, prima dell'invio all'unità di memoria, **si somma a tale indirizzo il valore contenuto nel registro di rilocazione**.
- Il programma utente tratta con gli indirizzi logici; non considera mai gli indirizzi fisici *reali*.



walter.balzano@gmail.com

Rilocazione dinamica tramite un registro di rilocazione



walter.balzano@gmail.com

Caricamento dinamico

- **Una procedura non viene caricata fino a quando non è richiamata.**
- **Migliore utilizzo dello spazio di memoria:** una procedura che non si adopera non viene caricata.
- **Utile quando servono grandi quantità di codice per gestire casi non frequenti (es. procedure di gestione degli errori).**
- **Non richiede un intervento particolare da parte del sistema operativo.**



walter.balzano@gmail.com

Collegamento dinamico

- Il collegamento viene differito fino al momento dell'esecuzione.
- Una piccola porzione di codice di riferimento (*stub*) indica come localizzare la giusta procedura di libreria residente nella memoria.
- Il codice (*stub*) sostituisce se stesso con l'indirizzo della procedura, che viene poi eseguita.
- Richiede generalmente l'assistenza del sistema operativo per controllare se la procedura richiesta da un processo è nello spazio di memoria di un altro processo.
- Il collegamento dinamico si rivela molto utile per le librerie.



walter.balzano@gmail.com

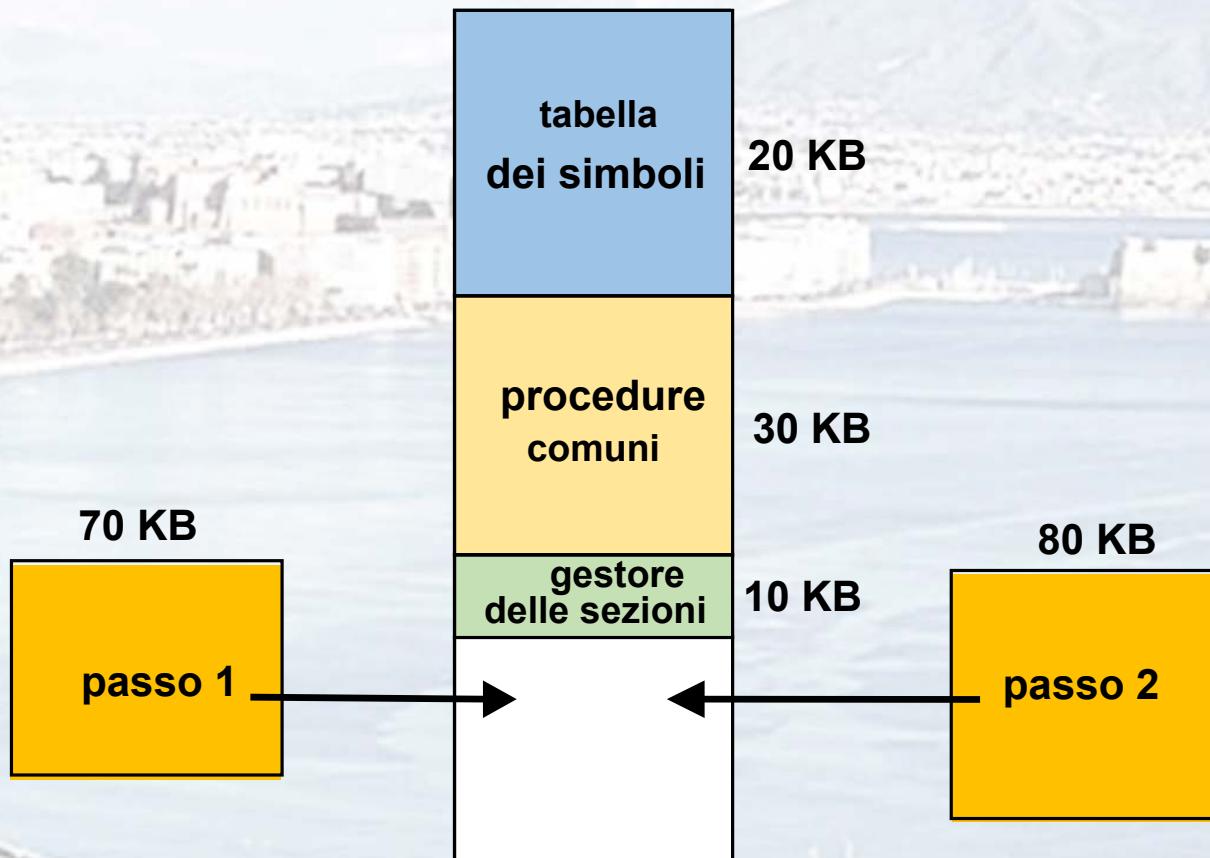
Sovrapposizione di sezioni (overlay)

- Mantiene nella memoria soltanto **le istruzioni e i dati che si usano con maggior frequenza**.
- Necessario per consentire a un processo di essere **più grande della memoria che gli si assegna**.
- Questa tecnica non richiede alcun intervento speciale del sistema operativo, e può essere realizzata direttamente dall'utente; molto complessa è invece la sua progettazione e programmazione.



walter.balzano@gmail.com

Sovrapposizione di sezioni per un assemblatore a due passi



walter.balzano@gmail.com

Avvicendamento dei processi (swapping)

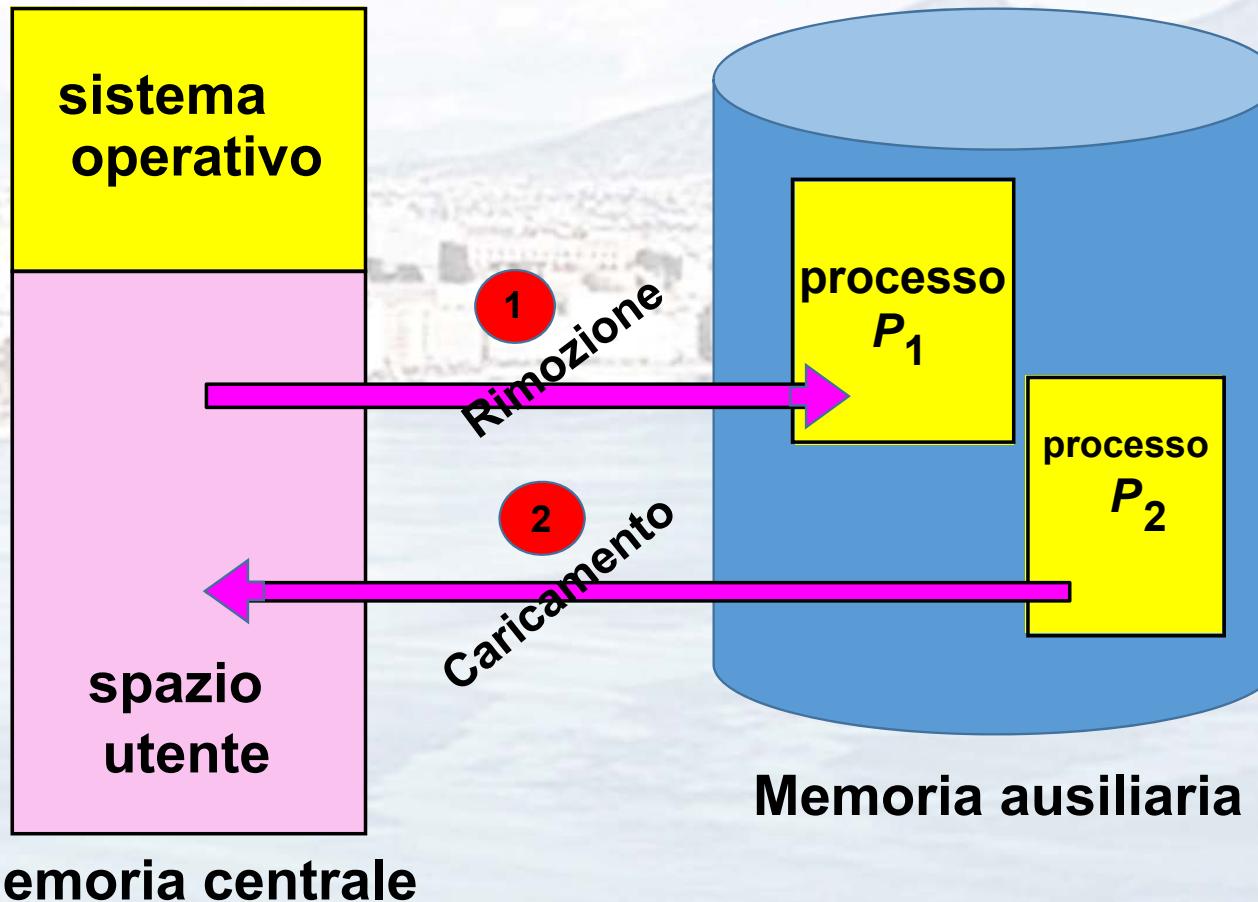
Un processo può essere trasferito temporaneamente nella **memoria ausiliaria** (*backing store*) e poi riportato nella memoria centrale al momento di riprenderne l'esecuzione.

- **Memoria ausiliaria:** disco veloce sufficientemente capiente da contenere le copie di tutte le immagini di memoria di tutti i processi; deve permettere un accesso diretto a queste immagini di memoria.
- **Roll OUT, Roll IN:** variante impiegata per gli algoritmi di scheduling basati sulle priorità: il processo con priorità inferiore viene scaricato dalla memoria centrale per fare spazio all'esecuzione del processo con priorità maggiore.
- La maggior parte del tempo d'avvicendamento è data dal tempo di trasferimento. Il tempo di trasferimento totale è direttamente proporzionale alla quantità di *memoria* interessata.
- Si possono trovare versioni modificate di tecniche di avvicendamento su molti sistemi, es. UNIX, Linux e Windows.



walter.balzano@gmail.com

Visione schematica dell'avvicendamento



walter.balzano@gmail.com

Assegnazione contigua della memoria

- La memoria centrale si divide di solito in due partizioni:

- una per il sistema operativo residente.
- una per i processi utenti.

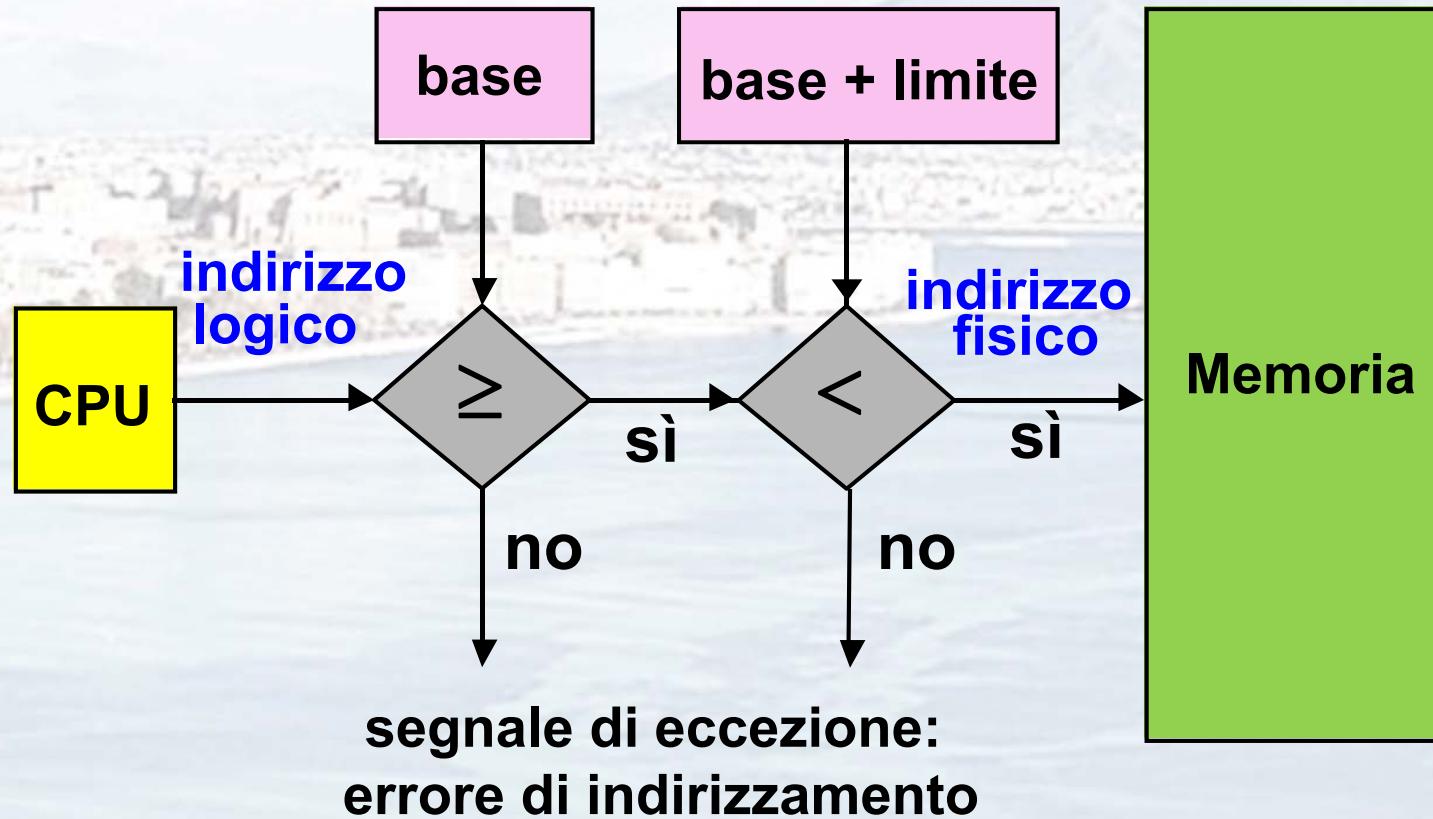
• Protezione della memoria

- Lo schema a registro di rilocazione viene usato per proteggere il sistema operativo dai processi utenti e i processi utenti dagli altri processi utenti.
- Il **registro di RILOCAZIONE** contiene il valore dell'indirizzo fisico minore;
- il **registro di LIMITE** contiene l'intervallo di indirizzi logici (ciascun indirizzo logico deve essere minore del contenuto del registro di limite).



walter.balzano@gmail.com

Registri di rilocazione e di limite

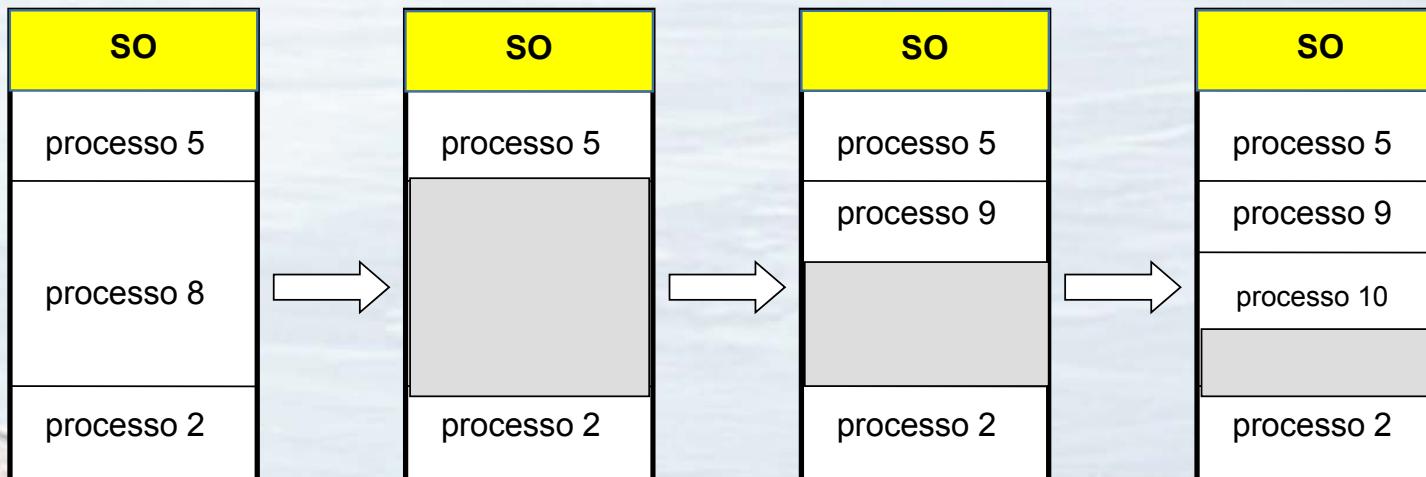


walter.balzano@gmail.com

Assegnazione contigua della memoria (Cont.)

- Assegnazione su più partizioni

- Buco (*hole*): blocco di memoria disponibile; ve ne sono di varie dimensioni, sparsi all'interno della memoria.
- Quando si carica un processo che necessita di memoria, **occorre cercare un buco sufficientemente grande da contenerlo.**
- Il sistema operativo tiene traccia di:
 - a) partizioni assegnate
 - b) partizioni libere (buchi)



walter.balzano@gmail.com

Assegnazione dinamica della memoria

Come soddisfare una richiesta di dimensione n data una lista di buchi liberi

- **First-fit:** si assegna **il primo buco abbastanza grande.**
- **Best-fit:** si assegna **il buco più piccolo** in grado di contenere il processo; occorre compiere la ricerca su tutta la lista, sempre che non sia ordinata per dimensione. **Produce le parti di buco inutilizzate più piccole**, ed è pertanto noto come *best-fit*.
- **Worst-fit:** si assegna **il buco più grande**; anche in questo caso occorre esaminare tutta la lista. **Produce le parti di buco inutilizzate più grandi.**

First-fit e best-fit funzionano meglio di worst-fit poiché riducono il tempo e l'utilizzo della memoria.



walter.balzano@gmail.com

Frammentazione

- **Frammentazione esterna:** lo spazio di memoria totale è sufficiente per soddisfare una richiesta, ma non è contiguo.
- **Frammentazione interna:** la memoria assegnata può essere leggermente maggiore della memoria richiesta; la memoria è interna a una partizione, ma non è in uso.
- Una soluzione al problema della frammentazione esterna è data dalla **compattazione**:
 - riordina il contenuto della memoria per riunire la memoria libera in un unico grosso blocco.
 - è possibile *solo* se la rilocazione è dinamica, e si compie nella fase di esecuzione.



walter.balzano@gmail.com

Paginazione

- **Metodo di gestione della memoria che permette che lo spazio degli indirizzi fisici di un processo non sia contiguo.**
- Suddivide la memoria fisica in **blocchi di memoria** di dimensioni fisse (noti anche come **pagine fisiche** o *frame*). La dimensione di una pagina è tipicamente una potenza di 2, compresa tra 512 byte e 16 MB.
- Suddivide la memoria logica in blocchi di uguale dimensione detti **pagine** (*pages*).
- Tiene traccia di tutti i blocchi liberi.
- Utilizza una tabella delle pagine per tradurre gli indirizzi logici in indirizzi fisici.
- **Può evitare la frammentazione esterna.**



walter.balzano@gmail.com

Schema di traduzione degli indirizzi

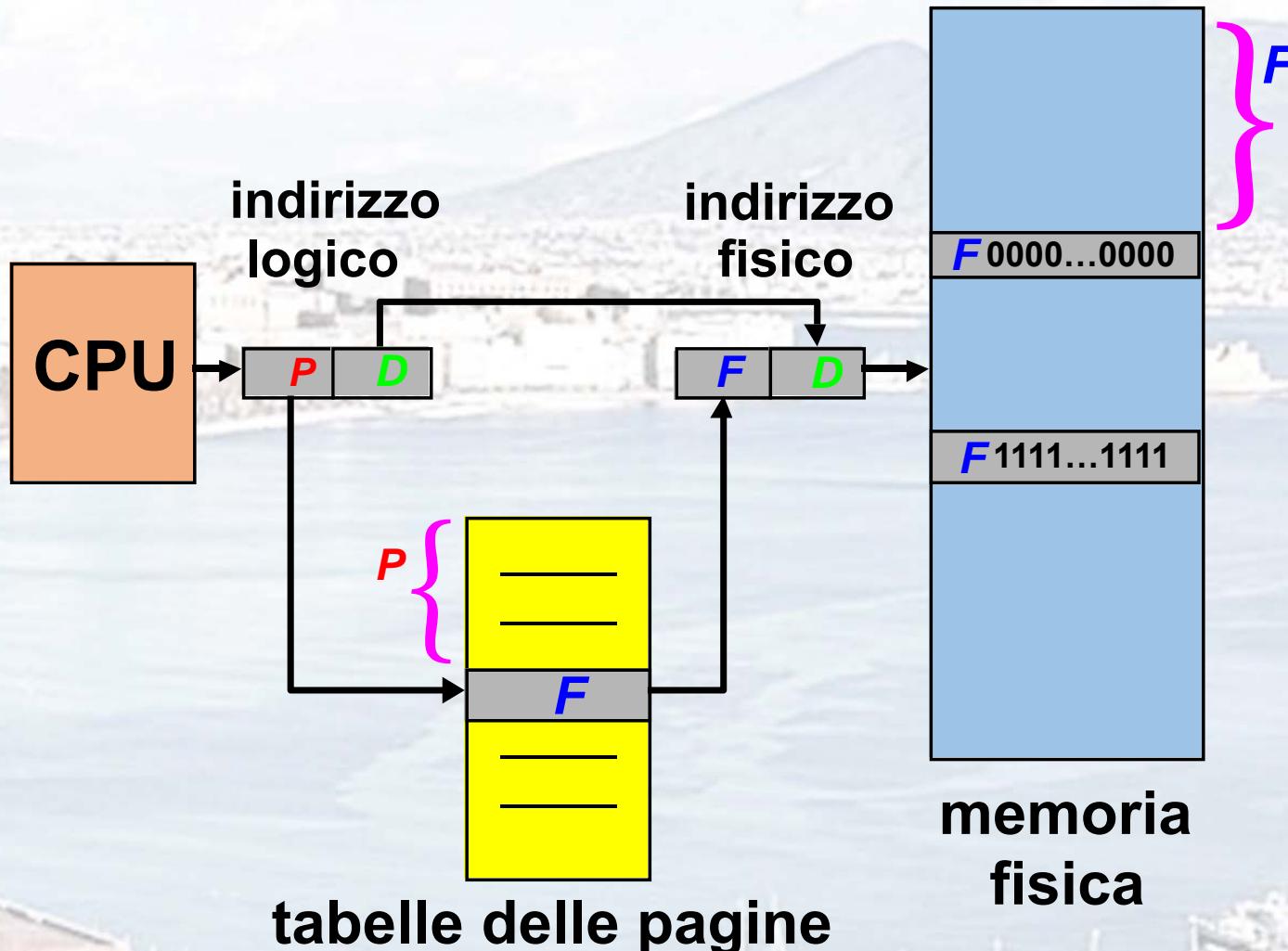
Ogni indirizzo generato dalla CPU è diviso in:

- **Numero di pagina (p):** serve come indice per la *tabella delle pagine*, che contiene l'**indirizzo di base nella memoria fisica di ogni pagina.**
- **Scostamento di pagina (d):** combinato con l'indirizzo di base definisce l'**indirizzo della memoria fisica**, che si invia all'unità di memoria.



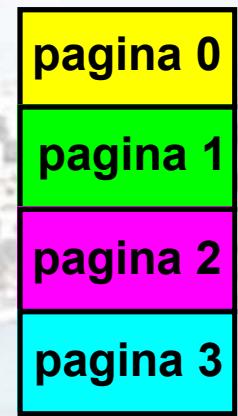
walter.balzano@gmail.com

Architettura di paginazione



walter.balzano@gmail.com

Esempio di paginazione

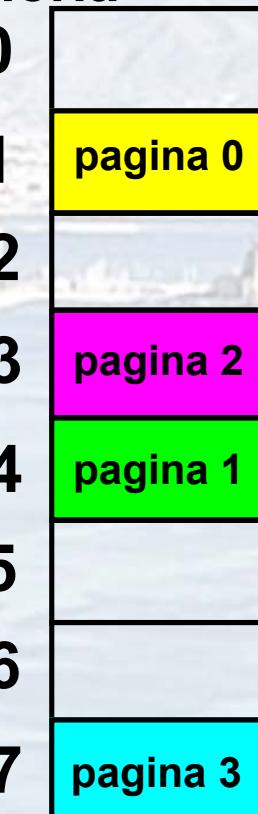


memoria logica

0	1
1	4
2	3
3	7

tabella delle pagine

numero
del blocco
di memoria



memoria fisica



walter.balzano@gmail.com

Esempio di paginazione

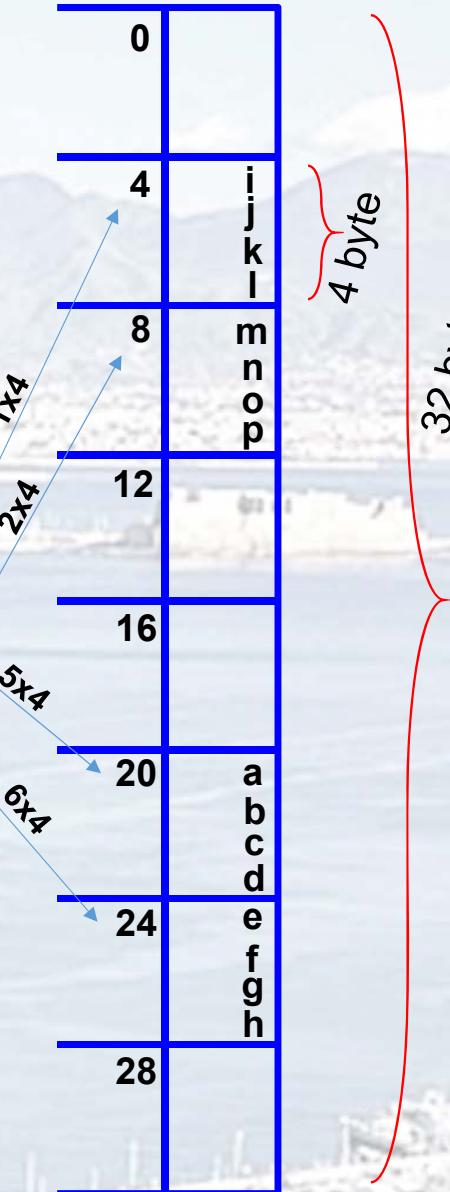
Paginazione per memoria di 32 byte
con pagine di 4 byte

Memoria logica

0	a
1	b
2	c
3	d
4	e f g h
5	
6	
7	
8	i j k l
9	
10	
11	
12	m n o p
13	
14	
15	

Tabella delle pagine

0	5
1	6
2	1
3	2



Memoria fisica

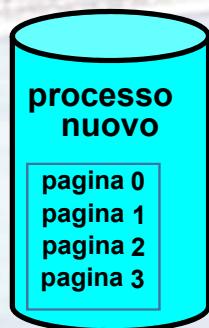


walter.balzano@gmail.com

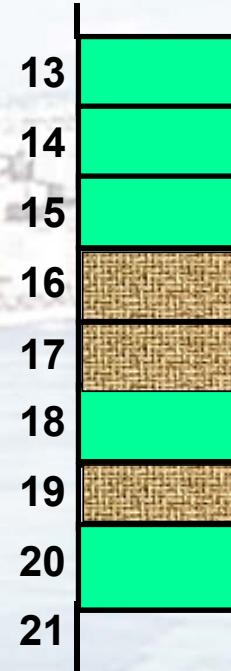
Blocchi di memoria liberi

lista dei blocchi
di memoria liberi

14
13
18
20
15



prima
dell'assegnazione

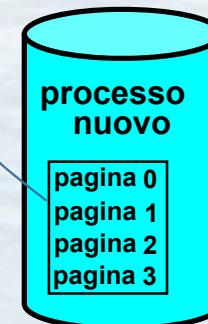


lista dei blocchi
di memoria liberi

15

Tabella delle
pagine del nuovo
processo

0	14
1	13
2	18
3	20



dopo
l'assegnazione



walter.balzano@gmail.com

Architettura di paginazione

- La tabella delle pagine è mantenuta nella memoria principale.
- Un **registro di base della tabella delle pagine** (*page-table base register, PTBR*) punta alla tabella stessa.
- Un **registro di lunghezza della tabella delle pagine** (*page-table length register, PRLR*) indica la dimensione della tabella delle pagine.
- Con questo metodo, per accedere a un byte occorrono **due accessi alla memoria**, uno per l'elemento della tabella delle pagine e uno per il byte stesso.
- La soluzione tipica a questo problema consiste nell'impiego di una speciale, piccola cache di memoria veloce detta **TLB** (*translation look-aside buffer*), una memoria associativa ad alta velocità.



walter.balzano@gmail.com

Memoria associativa (TLB)

- **Memoria associativa – Ricerca parallela**



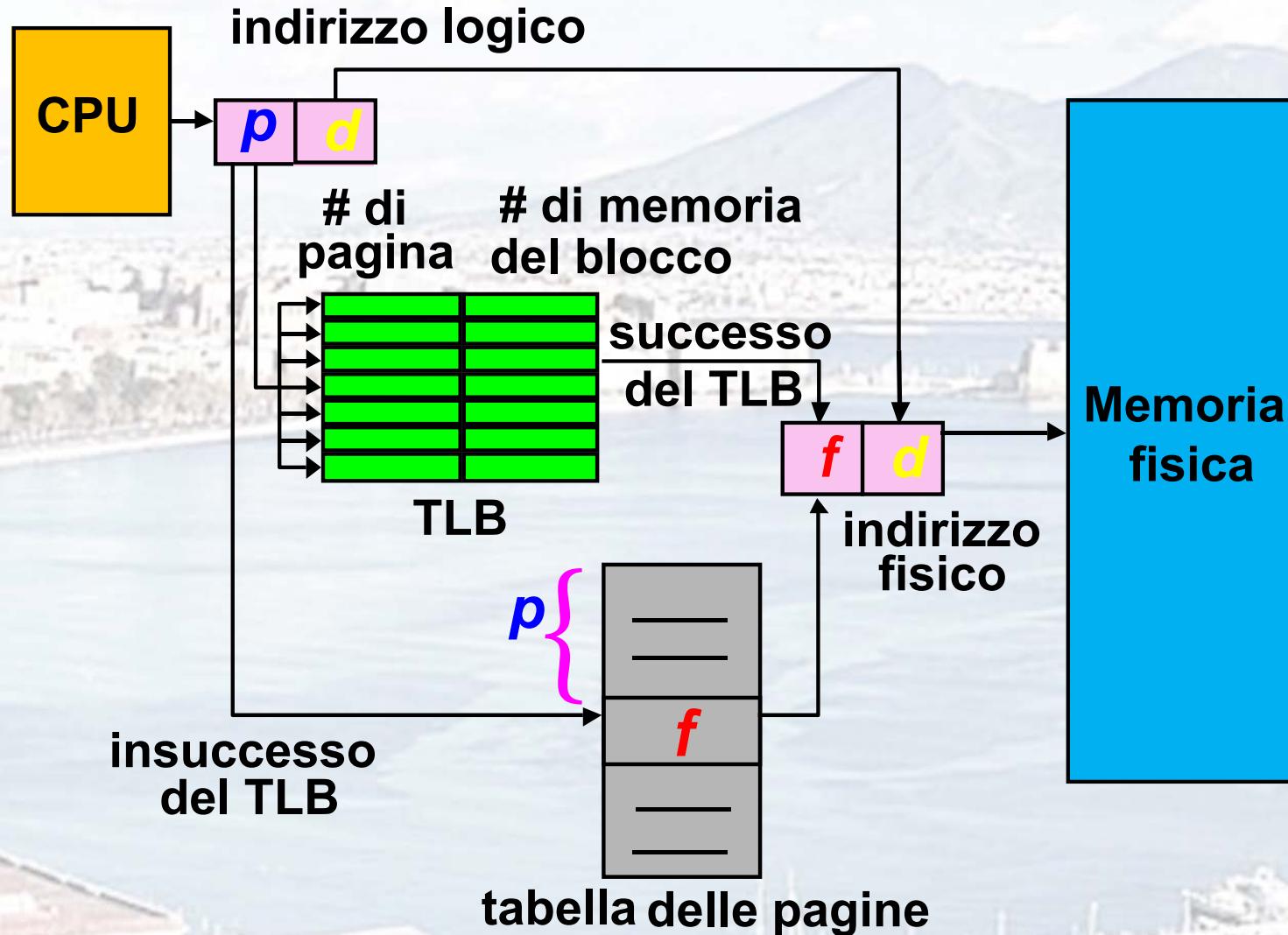
Traduzione dell'indirizzo (A' , A'')

- Se A' è presente nel TLB, il corrispondente numero di blocco di memoria è immediatamente disponibile e si usa per accedere alla memoria.
- Se A' non è presente si deve consultare la tabella delle pagine nella memoria.



walter.balzano@gmail.com

Architettura di paginazione con TLB



walter.balzano@gmail.com

Tempo effettivo d'accesso

- Lookup associativo = ε unità di tempo
- Si assume un tempo d'accesso alla memoria di 1 microsecondo
- **Tasso di successi** (*hit ratio*): percentuale di volte che un numero di pagina si trova nel TLB; relativo al numero di registri associativi.
- Tasso di successi = α
- **Tempo effettivo d'accesso (EAT, effective access time)**

$$\begin{aligned} \text{EAT} &= (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\ &= 2 + \varepsilon - \alpha \end{aligned}$$



walter.balzano@gmail.com

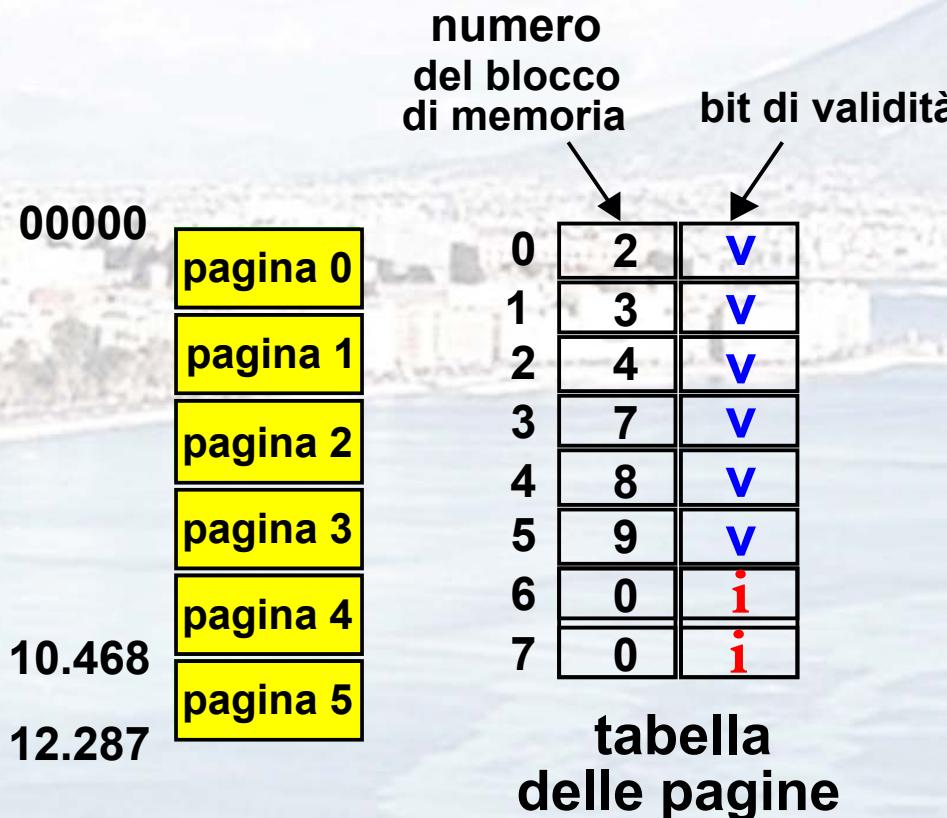
Protezione della memoria

- In un ambiente paginato, la protezione della memoria è assicurata dai bit di protezione associati a ogni blocco di memoria.
- Di solito si associa un **bit di validità** a ciascun elemento della tabella delle pagine:
 - “**bit valido**” indica che la pagina corrispondente è nello spazio d’indirizzi logici del processo, quindi è una pagina valida.
 - “**bit non valido**” indica che la pagina non è nello spazio d’indirizzi logici del processo.



walter.balzano@gmail.com

Bit di validità in una tabella delle pagine



0	
1	
2	pagina 0
3	pagina 1
4	pagina 2
5	
6	
7	pagina 3
8	pagina 4
9	pagina 5
:	
n	



walter.balzano@gmail.com

Struttura della tabella delle pagine

- Paginazione gerarchica
- Tabella delle pagine di tipo hash
- Tabella delle pagine invertita



walter.balzano@gmail.com

Paginazione gerarchica

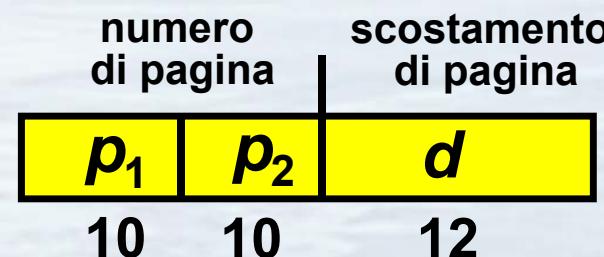
- **Suddivide la tabella delle pagine in parti più piccole.**
- Un metodo consiste nell'adottare un algoritmo di paginazione a due livelli, nel quale **si pagina la stessa tabella delle pagine.**



walter.balzano@gmail.com

Esempio di paginazione a due livelli

- Un indirizzo logico (su una macchina a 32 bit con dimensione delle pagine di 4K) è suddiviso in:
 - un numero di pagina di 20 bit
 - uno scostamento di pagina di 12 bit
- Paginando la tabella delle pagine, anche il numero di pagina è a sua volta diviso in:
 - un numero di pagina di 10 bit
 - uno scostamento di pagina di 10 bit
- Quindi, l'indirizzo logico è:

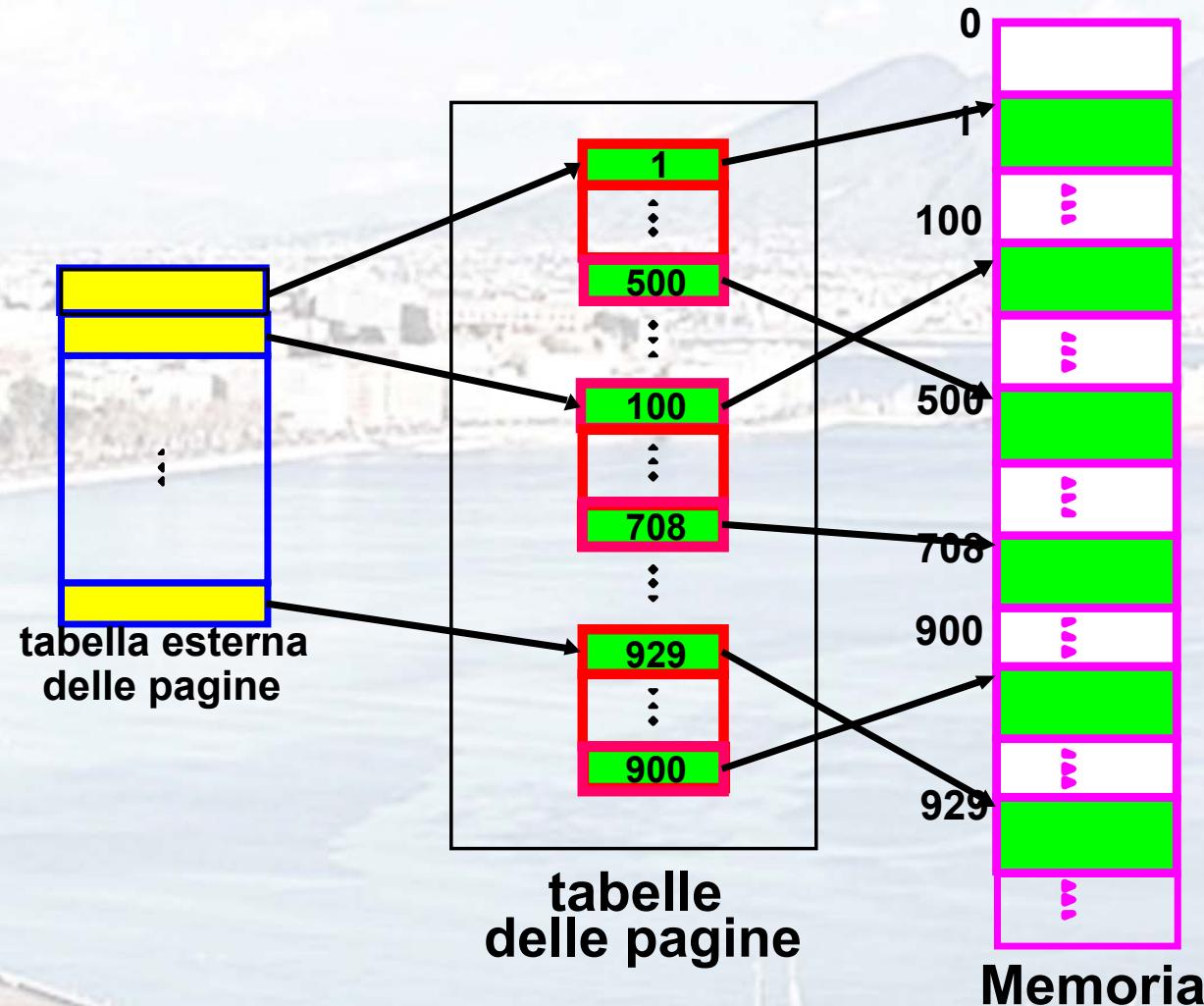


dove p_1 è un indice della tabella esterna delle pagine, e p_2 è lo scostamento all'interno della pagina indicata dalla tabella esterna delle pagine.



walter.balzano@gmail.com

Schema di una tabella delle pagine a due livelli

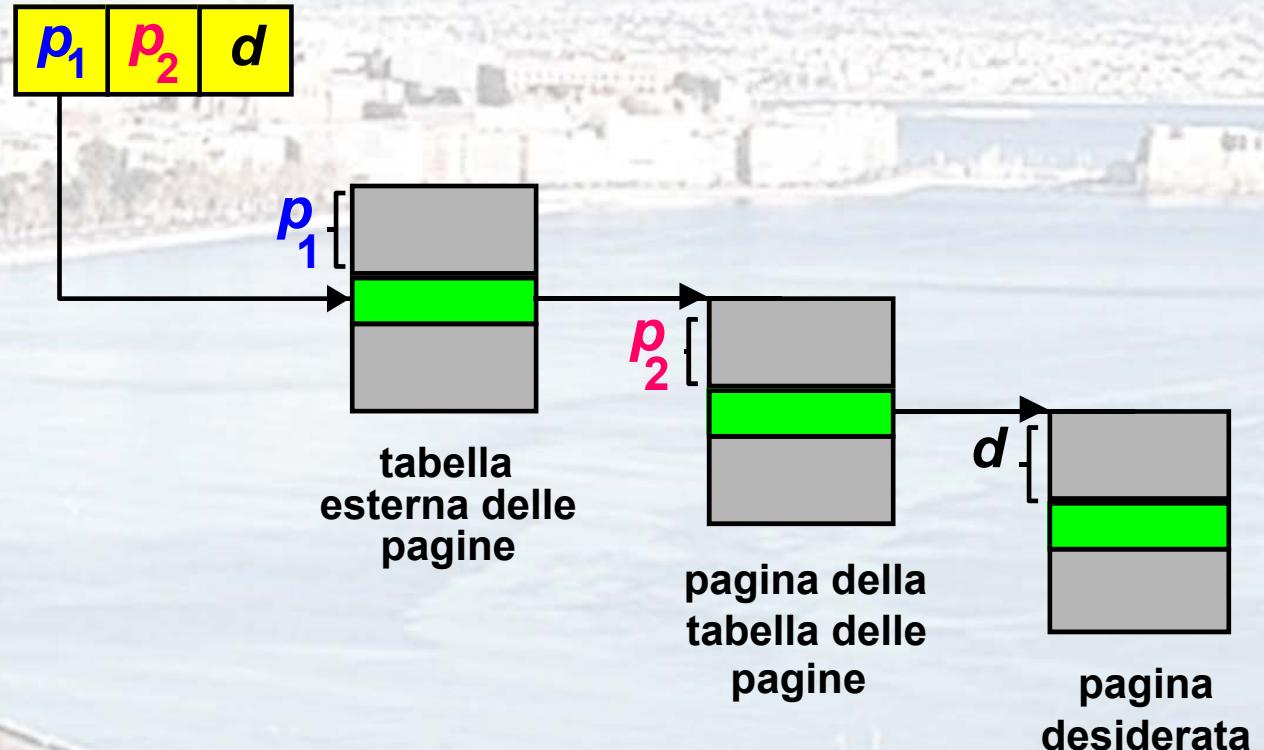


walter.balzano@gmail.com

Schema di traduzione degli indirizzi

- Traduzione degli indirizzi per un'architettura a 32 bit con paginazione a due livelli.

indirizzo logico



walter.balzano@gmail.com

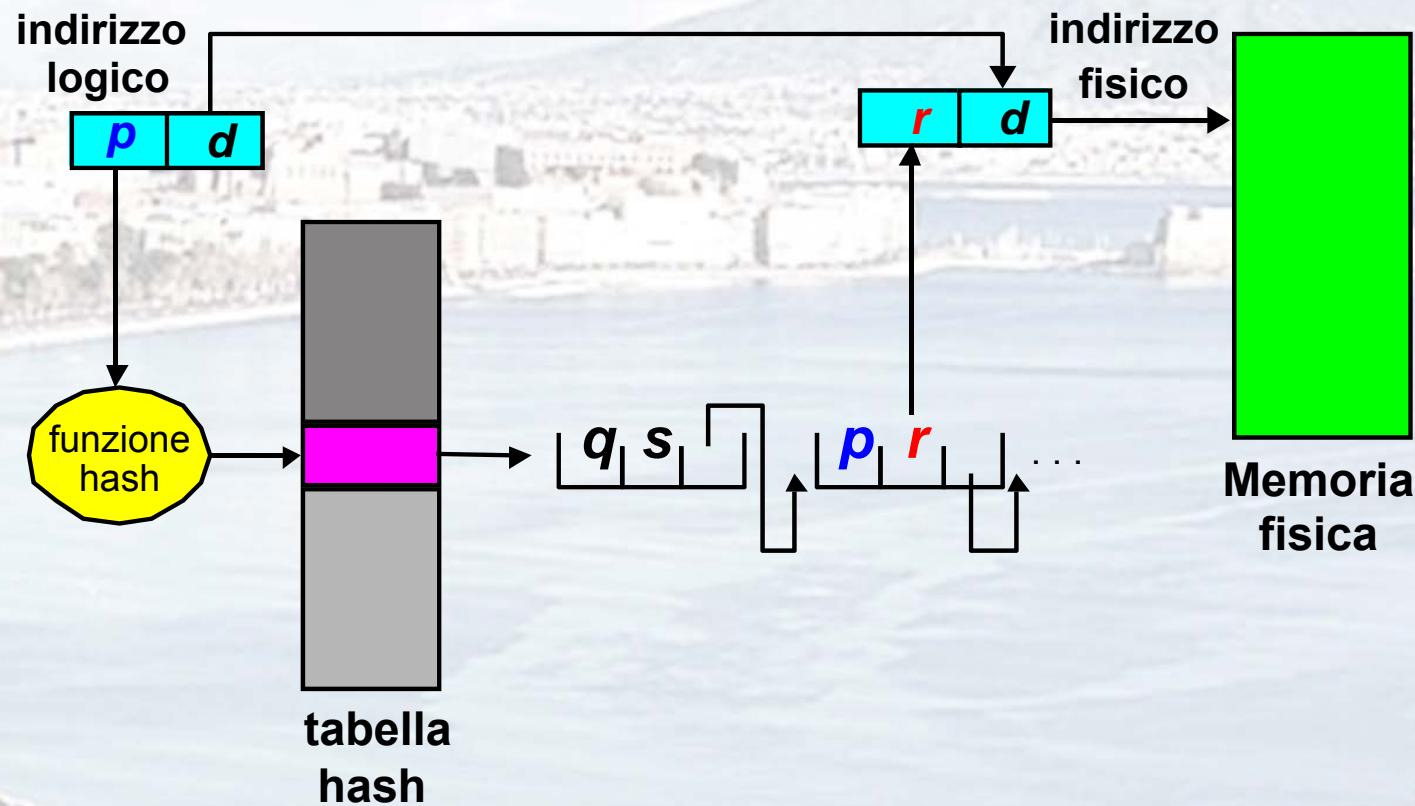
Tabella delle pagine di tipo hash

- Comune negli spazi d'indirizzi relativi ad architetture > 32 bit.
- L'argomento della funzione di hash è il numero della pagina virtuale. Ogni elemento della tabella hash contiene una lista concatenata di elementi che la funzione hash fa corrispondere alla stessa locazione.
- I numeri di pagina virtuali vengono confrontati e, se coincidono, si usa l'indirizzo del relativo blocco di memoria per generare l'indirizzo fisico desiderato.



walter.balzano@gmail.com

Tabella delle pagine di tipo hash



walter.balzano@gmail.com

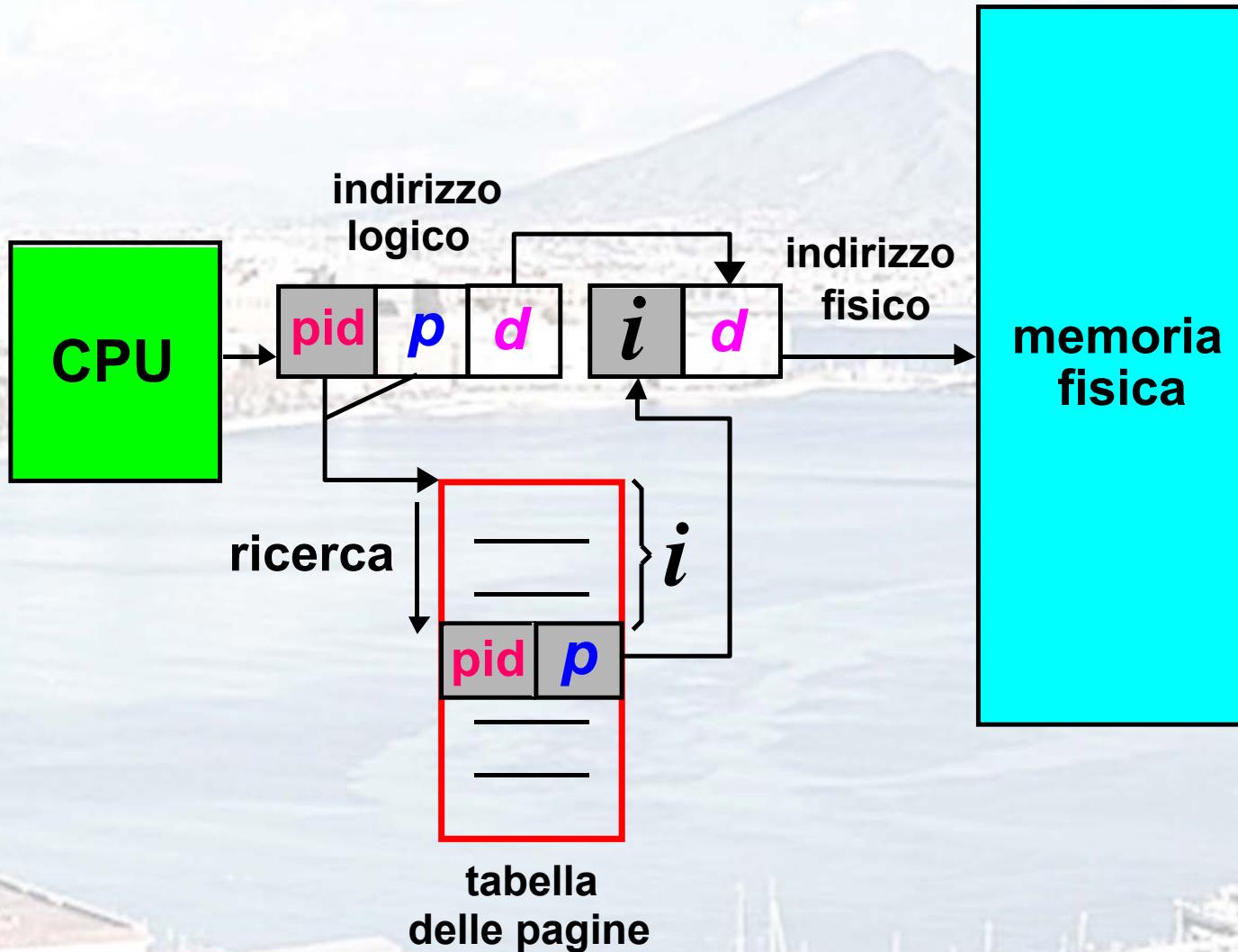
Tabella delle pagine invertita

- **Un elemento per ogni pagina reale (blocco di memoria).**
- **Ciascun elemento è costituito dall'indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria, con informazioni sul processo che possiede tale pagina.**
- **Riduce la quantità di memoria necessaria per memorizzare ogni tabella delle pagine, ma aumenta il tempo di ricerca nella tabella quando si fa riferimento a una pagina.**
- **Utilizza una tabella hash che riduce la ricerca a un solo – o a pochi – elementi della tabella delle pagine.**



walter.balzano@gmail.com

Tabella delle pagine invertita



walter.balzano@gmail.com

Pagine condivise

- **Codice condiviso**

- Una copia di codice rientrante condiviso tra i processi (es. editor di testi, compilatori, interfacce a finestre).
- Il codice condiviso deve essere collocato nella stessa locazione nello spazio degli indirizzi logici di tutti i processi.

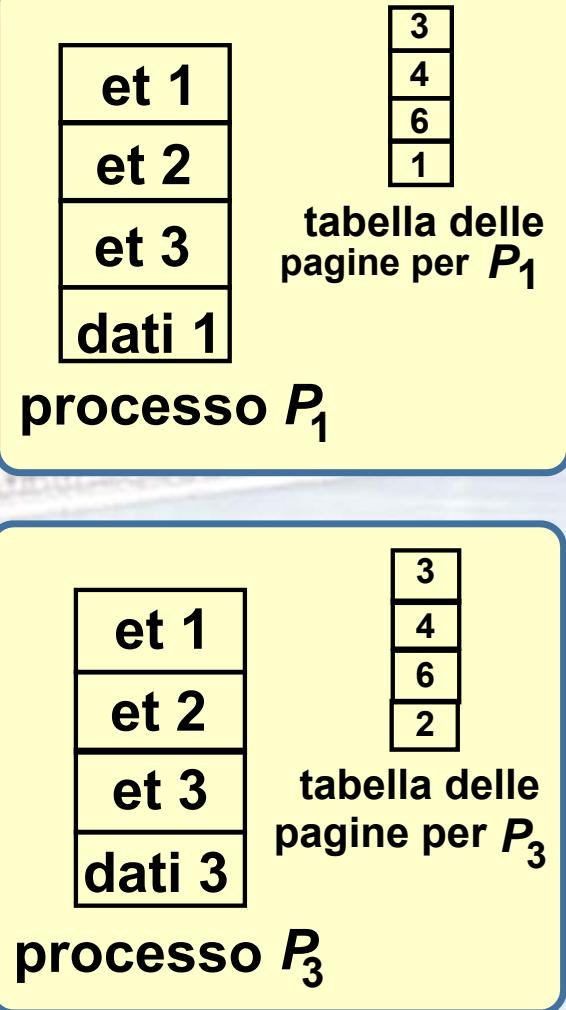
- **Codice privato e dati**

- Ciascun processo dispone di una propria copia di codice e dati.
- Le pagine per il codice e i dati possono essere collocate in un qualsiasi punto nello spazio degli indirizzi logici.



walter.balzano@gmail.com

Esempio di pagine condivise



0	
1	dati 1
2	dati 3
3	et 1
4	et 2
5	
6	et 3
7	dati 2
8	
9	
10	



walter.balzano@gmail.com

Segmentazione

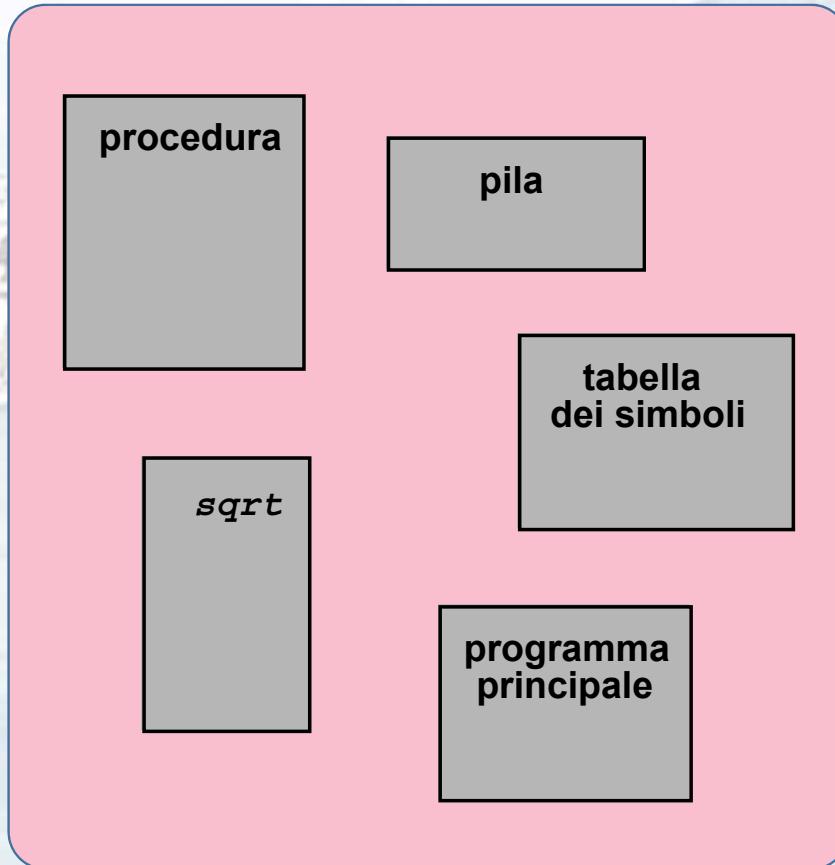
- Un aspetto importante della gestione della memoria è quello della separazione tra la visione della memoria dell'utente e l'effettiva memoria fisica.
- Un programma è un insieme di segmenti. Un segmento è un'unità logica come:

**programma principale,
procedure,
funzioni,
metodi,
oggetti,
variabili locali,
variabili globali,
pile,
tabella dei simboli,
vettori**



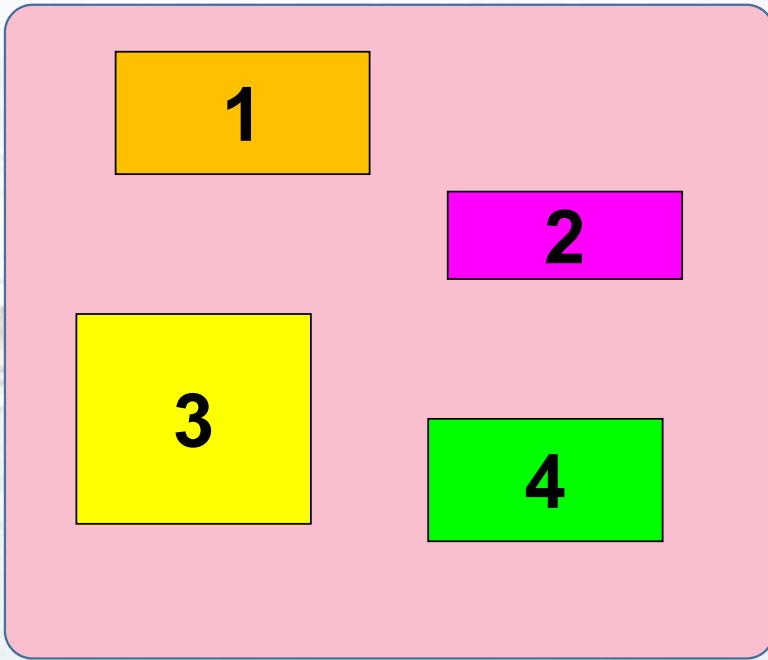
walter.balzano@gmail.com

Un programma dal punto di vista dell'utente



walter.balzano@gmail.com

Visione logica della segmentazione



spazio d'utente



spazio della
memoria fisica



walter.balzano@gmail.com

Architettura di segmentazione

- Un indirizzo logico è una **coppia**:
<numero di segmento, scostamento>,
- Tabella dei segmenti (*segment table*): traduce gli indirizzi bidimensionali definiti dall'utente negli indirizzi fisici unidimensionali; ogni elemento è composto da:
 - *base*: contiene l'indirizzo fisico iniziale della memoria al quale il segmento risiede.
 - *limite*: contiene la lunghezza del segmento.
- **Registro di base della tabella dei segmenti** (*segment-table base register, STBR*) punta alla tabella dei segmenti in memoria.
- **Registro di limite della tabella dei segmenti** (*segment-table length register, STLR*) indica il numero di segmenti usati da un programma;
numero di segmenti s è valido se $s < \text{STLR}$.



walter.balzano@gmail.com

Architettura di segmentazione (Cont.)

- **Rilocazione**

- dinamica
- tramite tabella dei segmenti

- **Condivisione**

- segmenti condivisi
- stesso numero di segmenti

- **Assegnazione della memoria**

- first fit/best fit
- frammentazione esterna



walter.balzano@gmail.com

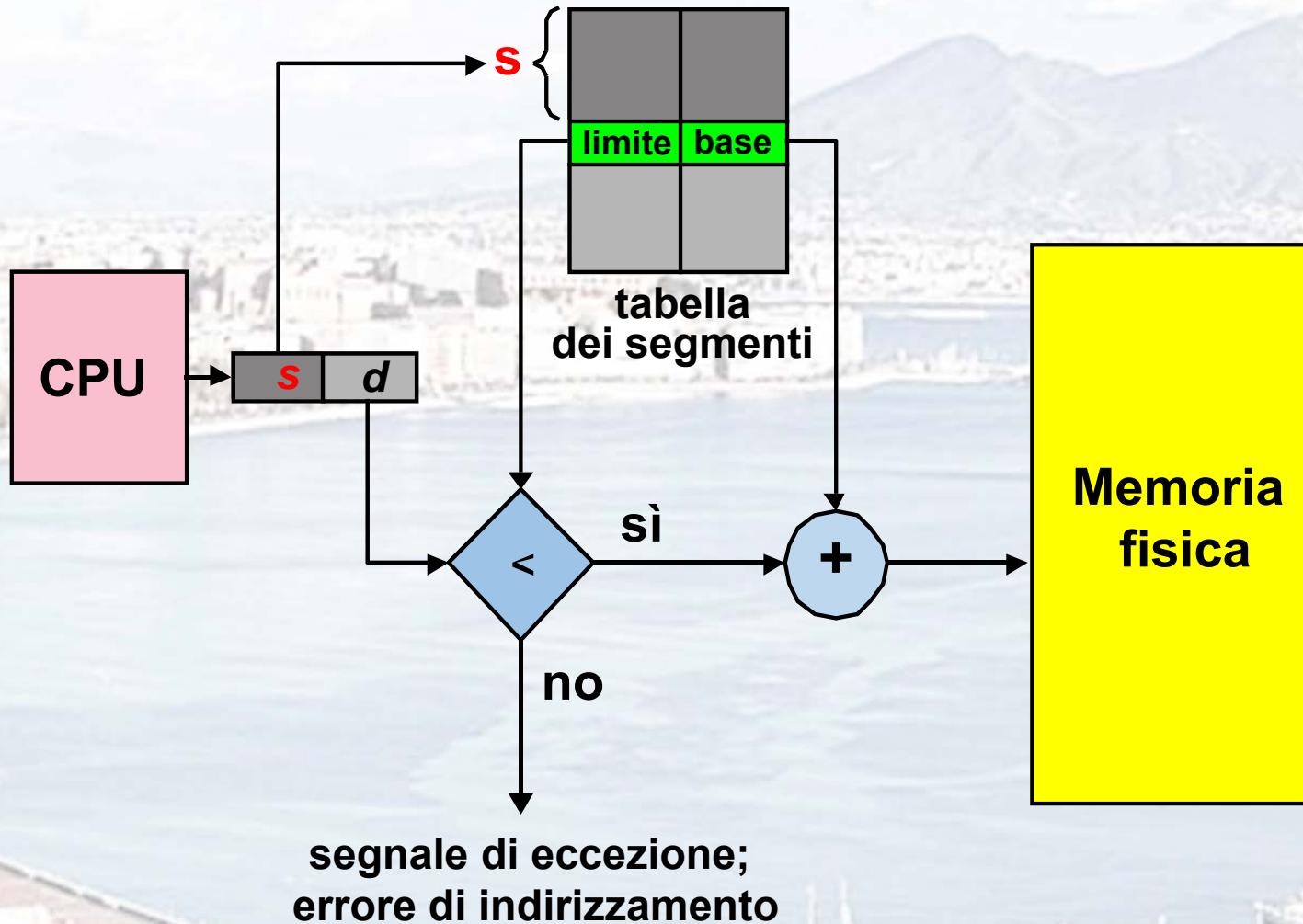
Architettura di segmentazione (Cont.)

- **Protezione.** A ciascun elemento della tabella dei segmenti è associato:
 - Bit di validazione = 0 \Rightarrow segmento illegale
 - Privilegi read/write/execute
- Bit di protezione associati ai segmenti; la condivisione di codice avviene a livello di segmento.
- Poiché i segmenti variano in lunghezza, l'assegnazione della memoria è un problema di allocazione dinamica.
- Nei diagrammi seguenti sono illustrati esempi di segmentazione.



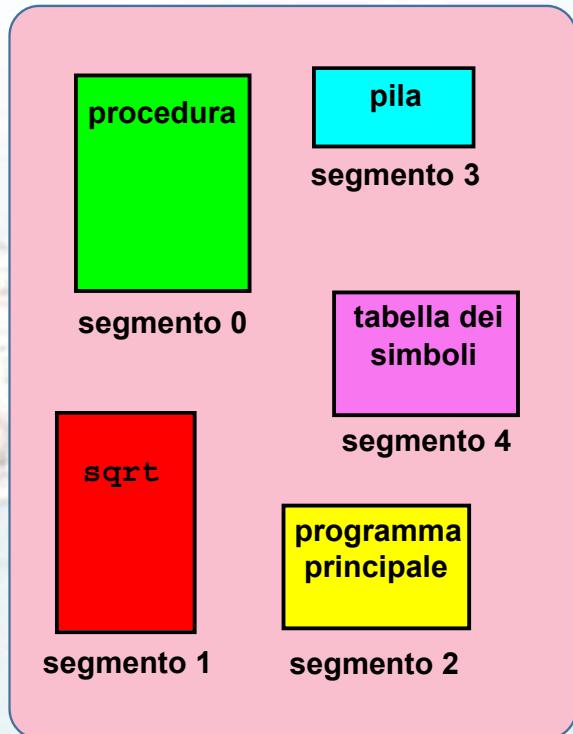
walter.balzano@gmail.com

Architettura di segmentazione



walter.balzano@gmail.com

Esempio di segmentazione



spazio degli
indirizzi logici

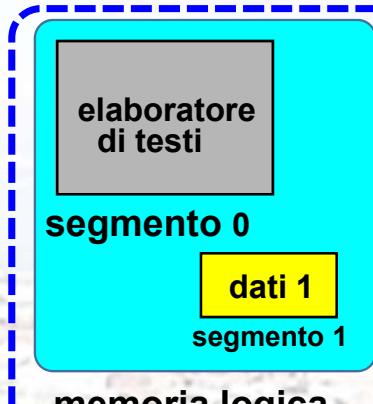
	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

**tabella
dei segmenti**



walter.balzano@gmail.com

Condivisione di segmenti



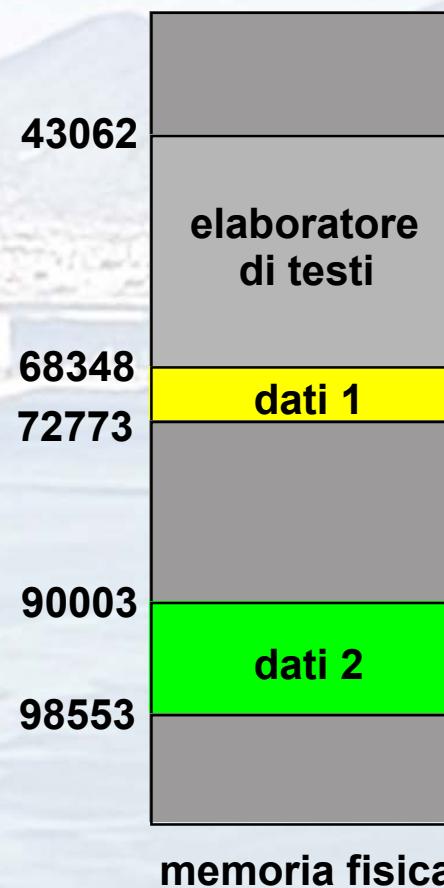
	limite	base
0	25286	43062
1	4425	68348

tabella dei segmenti del processo P_1



	limite	base
0	25286	43062
1	8850	90003

tabella dei segmenti del processo P_2



walter.balzano@gmail.com

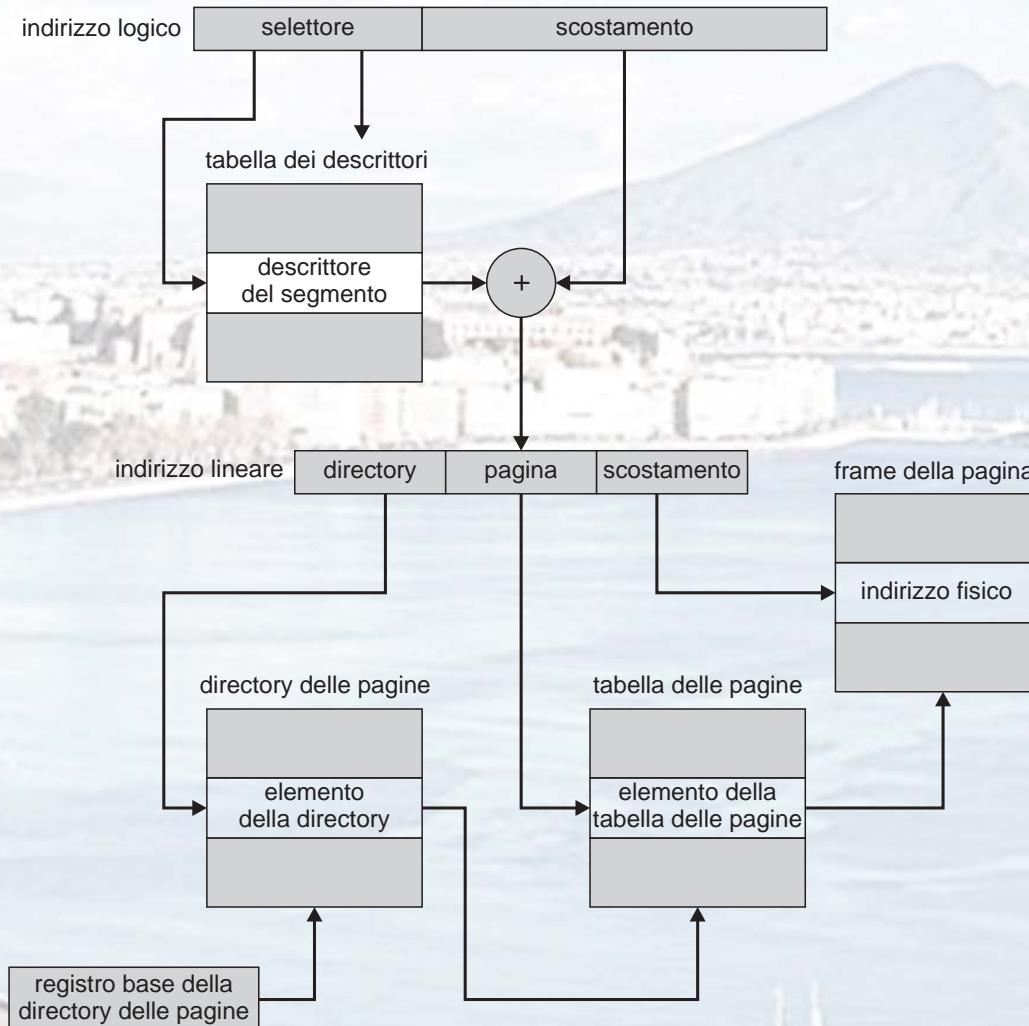
Segmentazione con paginazione – Intel 386

- Come illustrato nel seguente diagramma, la CPU Intel 386 adotta la segmentazione con paginazione per la gestione della memoria, con uno schema di paginazione a due livelli.



walter.balzano@gmail.com

Traduzione degli indirizzi nell'Intel 30386



walter.balzano@gmail.com

Capitolo 10: Memoria virtuale

- Introduzione
- Paginazione su richiesta
- Creazione dei processi
- Sostituzione delle pagine
- Assegnazione dei blocchi di memoria
- Attività di paginazione degenere
(thrashing)
- Esempi tra i sistemi operativi



walter.balzano@gmail.com

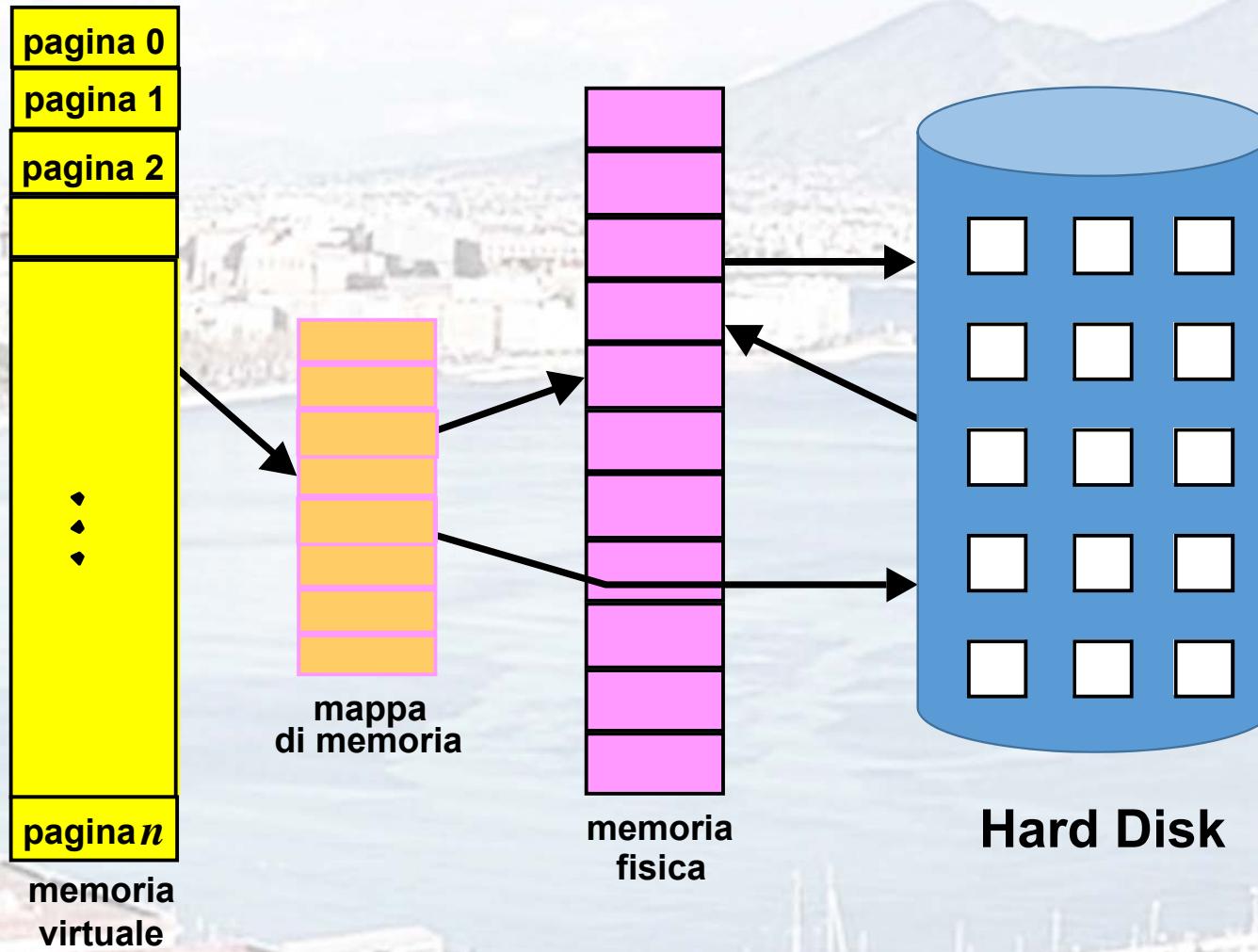
Introduzione

- La memoria virtuale rappresenta la separazione della memoria logica, vista dall'utente, dalla memoria fisica.
 - Solo parte del programma deve essere in memoria per l'esecuzione.
 - Lo spazio degli indirizzi logici può essere maggiore dello spazio degli indirizzi fisici.
 - Permette la condivisione di file e memoria tra diversi processi tramite la condivisione delle pagine.
 - Consente anche un miglioramento delle prestazioni durante la creazione dei processi.
- La memoria virtuale generalmente si realizza nella forma di:
 - Paginazione su richiesta (*demand paging*)
 - Segmentazione su richiesta (*demand segmentation*)



walter.balzano@gmail.com

Schema di memoria virtuale più grande della memoria fisica



walter.balzano@gmail.com

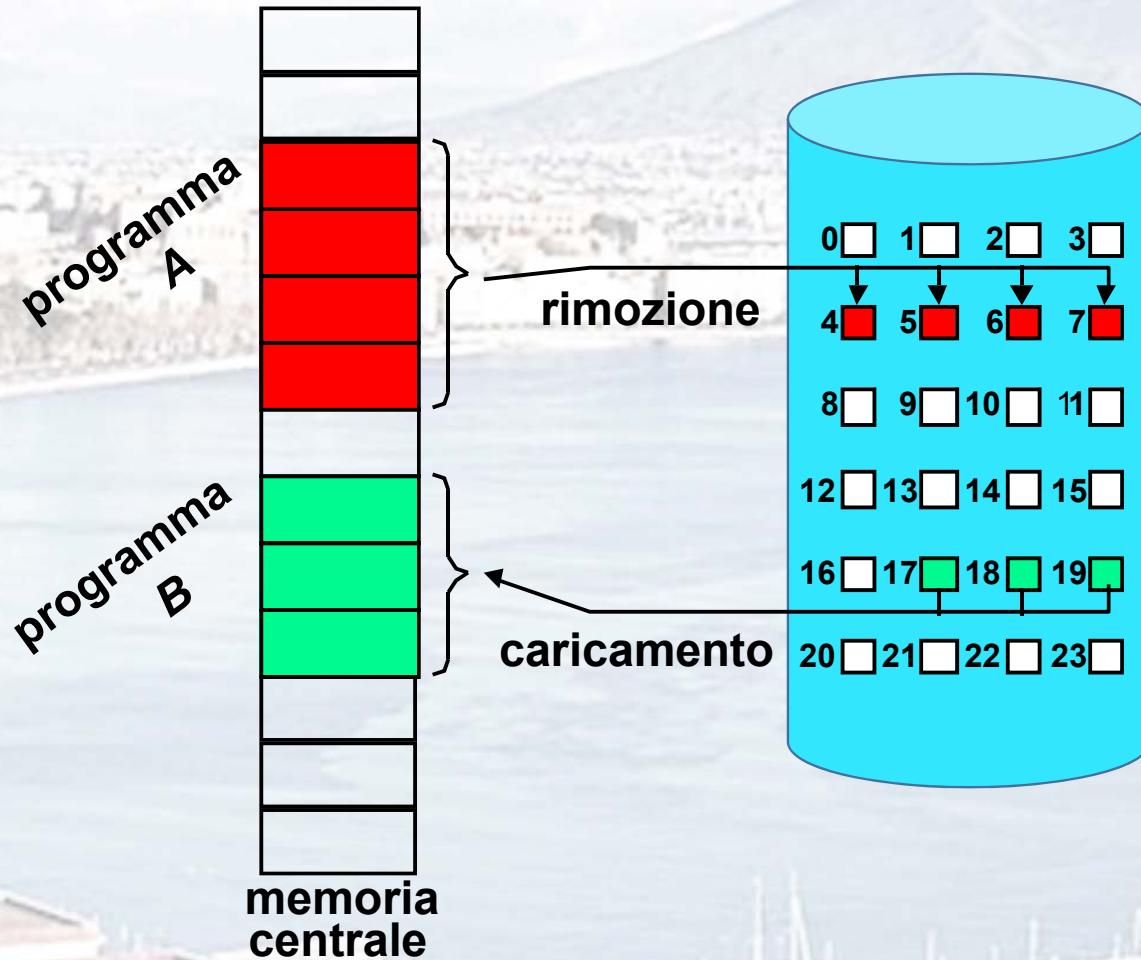
Paginazione su richiesta

- **Non si carica mai nella memoria una pagina che non sia necessaria.**
 - Minore tempo di avvicendamento
 - Minore quantità di memoria fisica
 - Risposta più veloce
 - Più utenti



walter.balzano@gmail.com

Trasferimento di una memoria paginata nello spazio contiguo di un disco



walter.balzano@gmail.com

Bit di validità

- A ciascun elemento della tabella delle pagine è associato un **bit di validità** (**1** \Rightarrow **in memoria**, **0** \Rightarrow **NON in memoria**).
- Inizialmente il bit di validità è impostato a 0.
- Instantanea di una tabella delle pagine.

blocco di memoria	bit di validità
	1
	1
	1
	1
	0
:	
	0
	0

tabella delle pagine

Durante la traduzione degli indirizzi, se il bit di validità nella tabella delle pagine è 0 \Rightarrow pagina non valida.

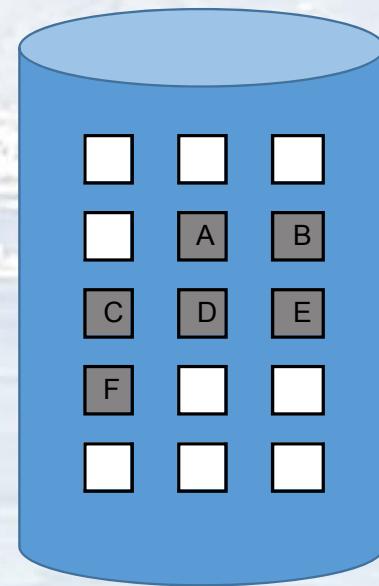
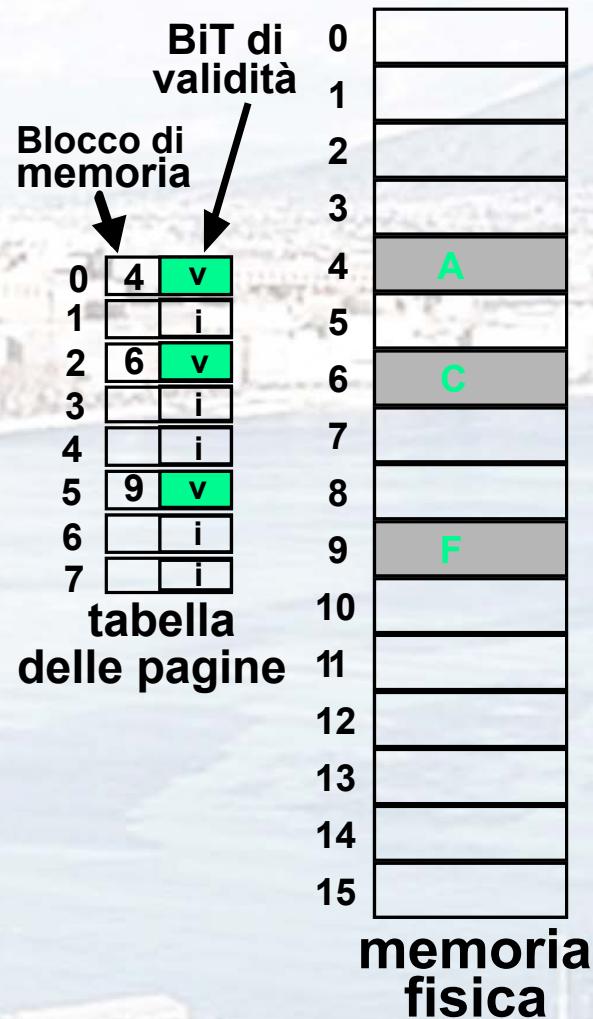


walter.balzano@gmail.com

Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

memoria logica



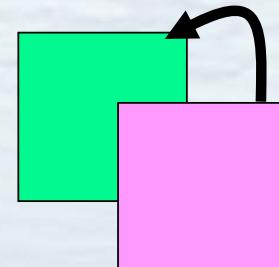
Hard Disk



walter.balzano@gmail.com

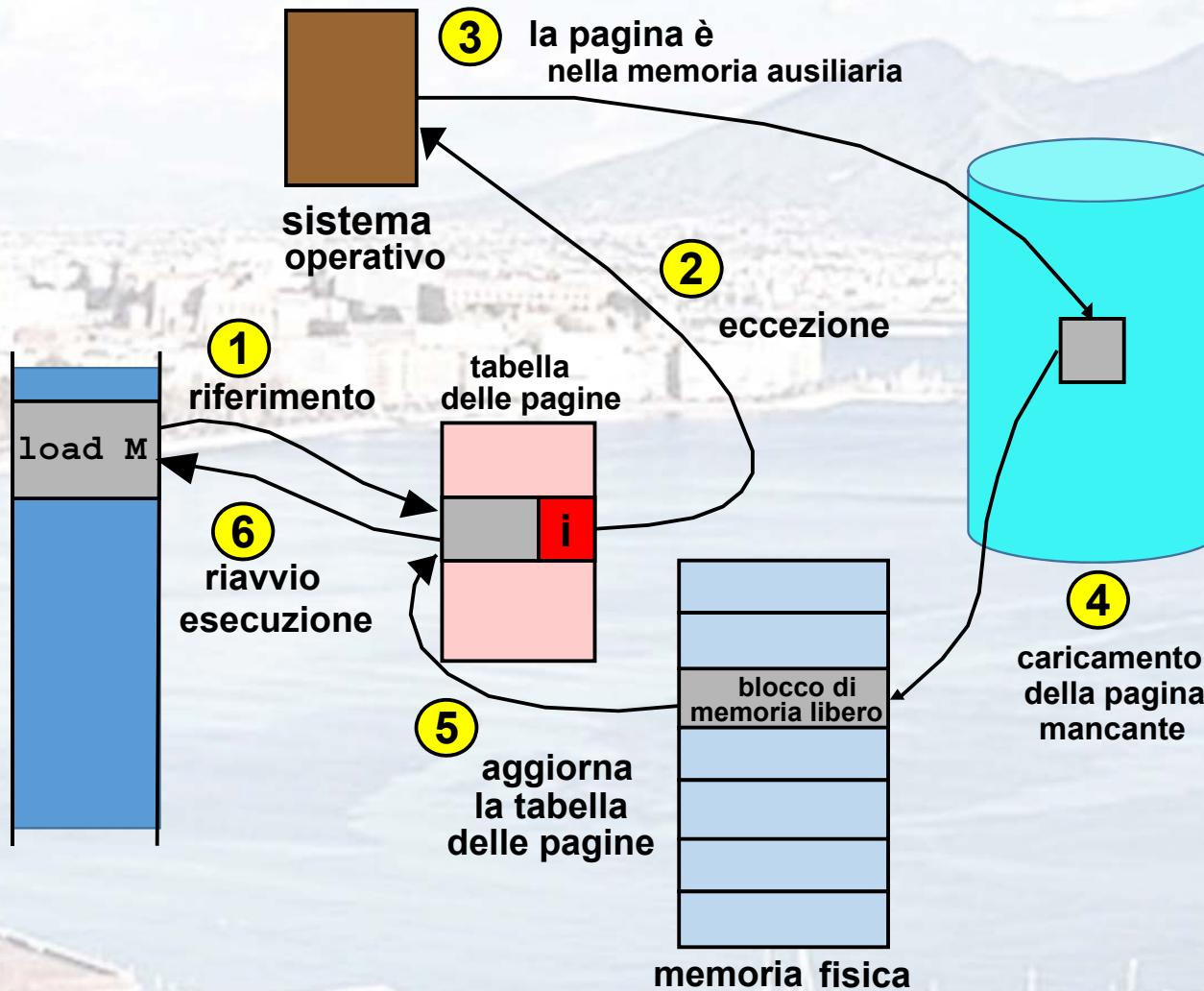
Eccezione di pagina mancante

- Se il processo tenta di accedere a una pagina che non era stata caricata nella memoria, si verifica un'eccezione di pagina mancante (*page fault trap*).
- L'architettura di paginazione, traducendo l'indirizzo, nota che il bit non è valido e invia un segnale d'eccezione al sistema operativo.
- Il sistema operativo deve decidere:
 - errore di indirizzo non valido \Rightarrow abort.
 - insuccesso del sistema operativo nella scelta delle pagine da caricare in memoria.
- Si individua un blocco di memoria libero.
- Si trasferisce la pagina desiderata nel blocco di memoria libero.
- Si aggiornano le tabelle, bit di validità = 1.
- Si riavvia l'istruzione che era stata interrotta dal segnale di eccezione.



walter.balzano@gmail.com

Fasi di gestione di un'eccezione di pagina mancante



walter.balzano@gmail.com

Prestazioni della paginazione su richiesta

- Frequenza di assenza di pagina $0 \leq p \leq 1.0$
 - se $p = 0$ non si verifica assenza di pagina (*page fault*)
 - se $p = 1$, ogni riferimento è un'assenza di pagina
- Tempo d'accesso effettivo (EAT, *effective access time*)
$$\text{EAT} = (1 - p) \times \text{accesso alla memoria}$$
$$+ p \text{ (page fault overhead)}$$
$$+ [\text{swap page out}]$$
$$+ \text{swap page in}$$
$$+ \text{restart overhead})$$



walter.balzano@gmail.com

Creazione dei processi

- La memoria virtuale offre altri vantaggi durante la creazione dei processi:
 - Copiatura su scrittura (*copy-on-write*)
 - Associazione alla memoria (*memory-mapping*) di un file



walter.balzano@gmail.com

Copiatura su scrittura

- La copiatura su scrittura (**COW, copy-on-write**) consente la condivisione iniziale delle pagine da parte dei processi genitori e dei processi figli.
- Se un processo (genitore o figlio) scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina.
- Questa tecnica consente una maggiore efficienza poiché solo le pagine modificate vengono copiate.
- Le pagine libere si trovano in un gruppo (*pool*) di pagine libere che di solito si assegnano quando la pila di un processo deve espandersi oppure proprio per gestire pagine da copiare su scrittura.



walter.balzano@gmail.com

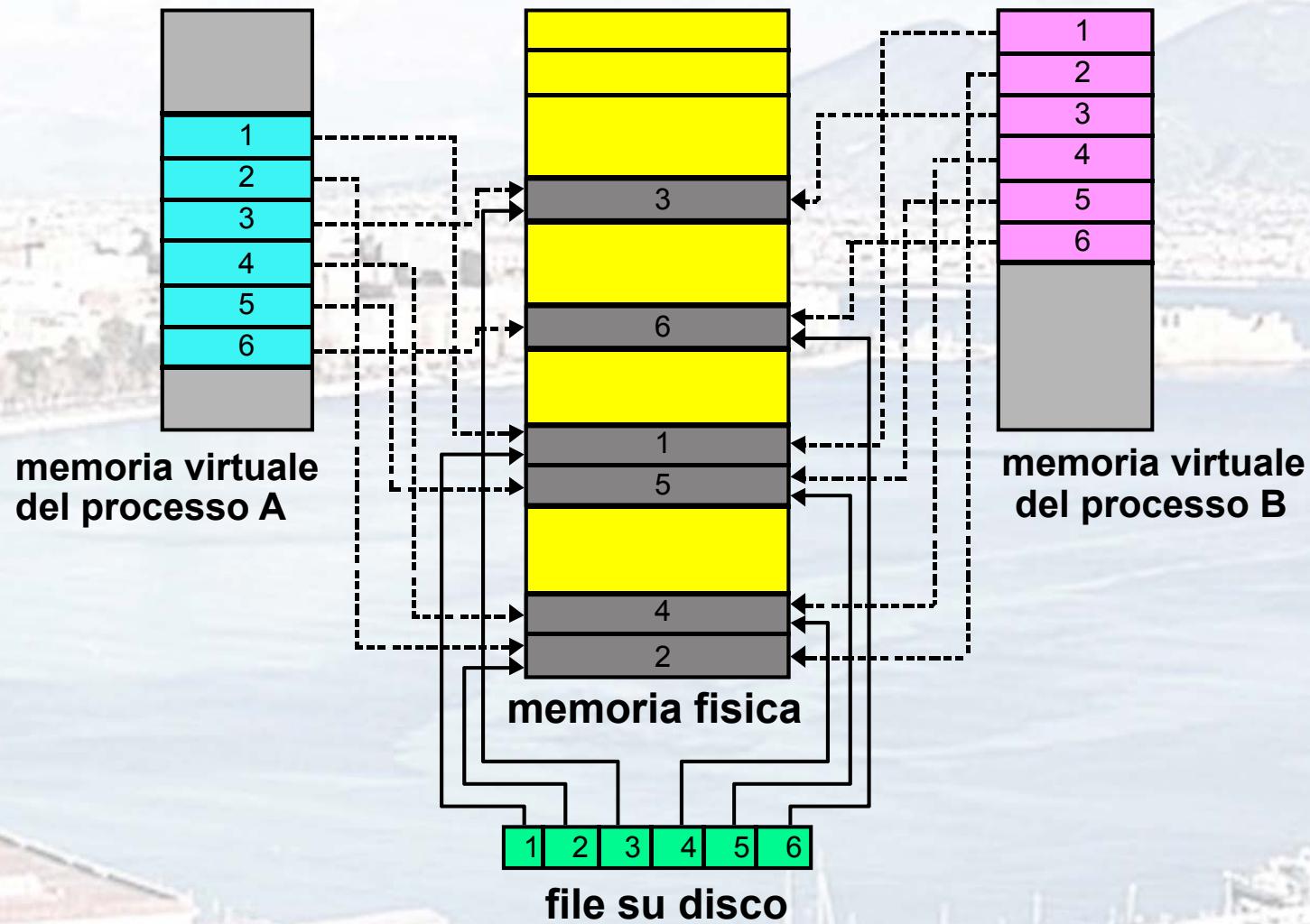
Associazione dei file alla memoria

- L'associazione alla memoria di un file consiste nel permettere che una parte dello spazio degli indirizzi virtuali sia associata logicamente a un file.
- L'accesso iniziale al file avviene per mezzo dell'ordinario meccanismo di paginazione su richiesta, che determina un'assenza di pagina cui segue il caricamento in una pagina fisica di una porzione di file della dimensione di una pagina, leggendola dal file system. Le operazioni successive di lettura/scrittura sul file si gestiscono come normali accessi alla memoria.
- In questo modo si semplifica l'accesso e l'uso dei file, si permette la manipolazione degli stessi attraverso la memoria e si evita il carico dovuto alle chiamate del sistema `read()` e `write()`.
- Si può permettere l'associazione dello stesso file alla memoria virtuale di più processi, allo scopo di condividere dati.



walter.balzano@gmail.com

Associazione dei file alla memoria



walter.balzano@gmail.com

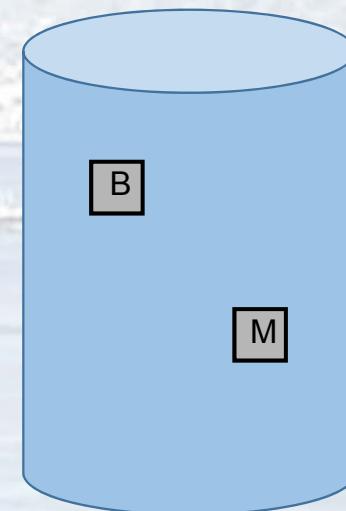
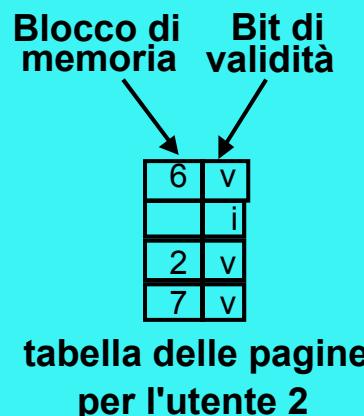
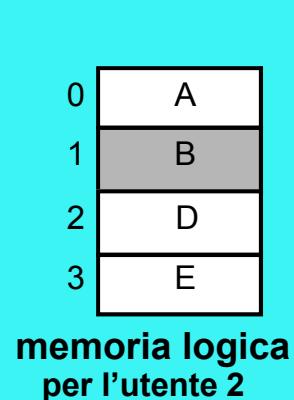
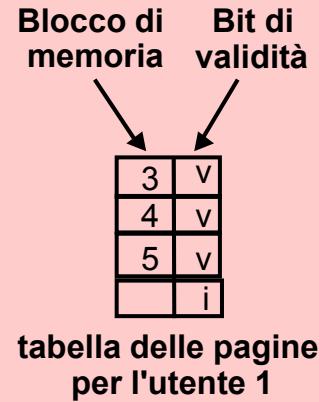
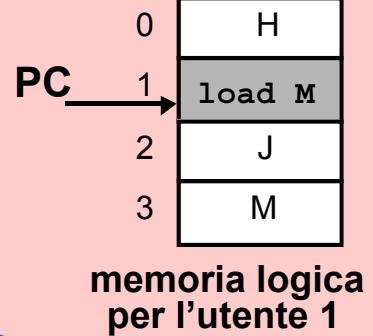
Sostituzione delle pagine

- Previene la sovrassegnazione della memoria
- Utilizza un **bit di modifica** (*dirty bit*) per ridurre il sovraccarico: solo le pagine modificate vengono scritte su disco.
- La sostituzione delle pagine completa la separazione tra memoria logica e memoria fisica.



walter.balzano@gmail.com

Necessità di sostituzione di pagine



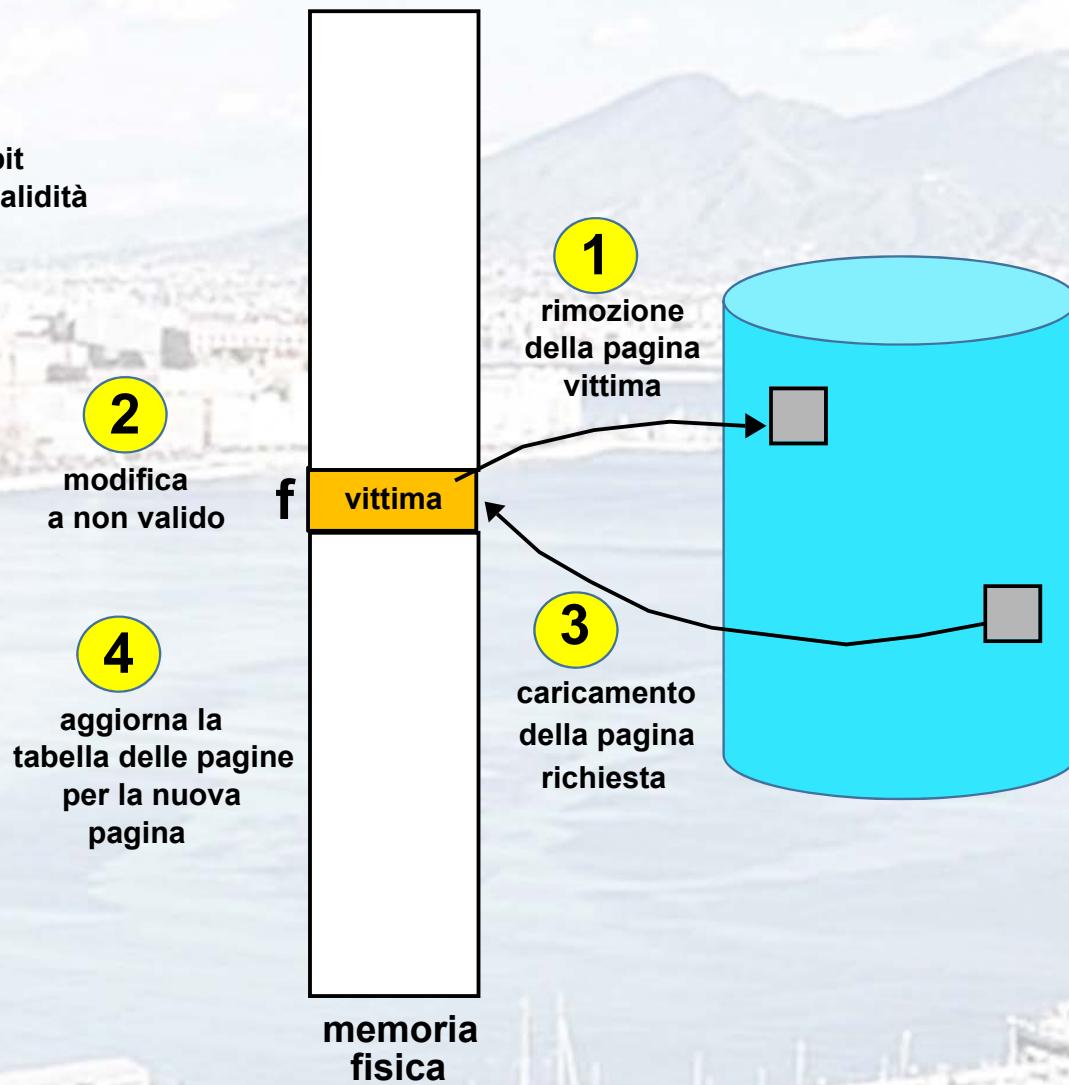
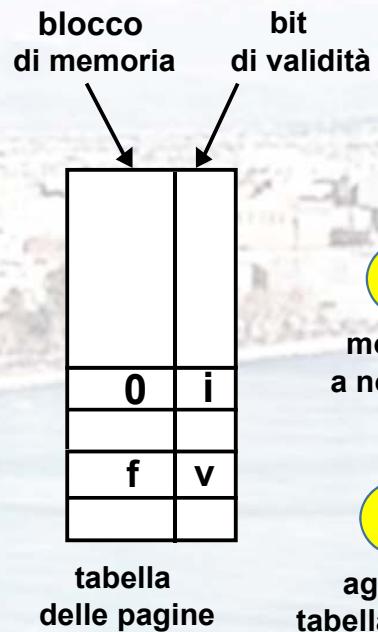
Schema di base

- 1. Si individua la locazione nel disco della pagina richiesta.**
- 2. Si cerca un blocco di memoria libero:**
 - se esiste, lo si usa;
 - altrimenti, si impiega un algoritmo di sostituzione delle pagine per scegliere un blocco di memoria “vittima”
- 3. Si scrive la pagina richiesta nel blocco di memoria appena liberato; si modificano le tabelle delle pagine e dei blocchi di memoria.**
- 4. Si riavvia il processo utente.**



walter.balzano@gmail.com

Sostituzione di una pagina



walter.balzano@gmail.com

Algoritmi di sostituzione delle pagine

- Si desidera la minor frequenza di assenza di pagine.
- Si valuta l'algoritmo su una particolare successione (stringa) di riferimenti (*reference string*) e si calcola il numero di assenze di pagine su quella successione.
- In tutti i nostri esempi, la stringa di riferimento è:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



walter.balzano@gmail.com

Numero delle assenze di pagine rispetto al numero dei blocchi di memoria



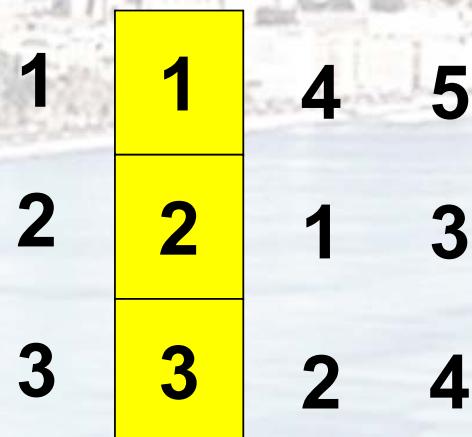
walter.balzano@gmail.com

Algoritmo FIFO (First-In-First-Out)

Successione di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 blocchi di memoria

9 assenze di pagine



4 blocchi di memoria

10 assenze di pagine

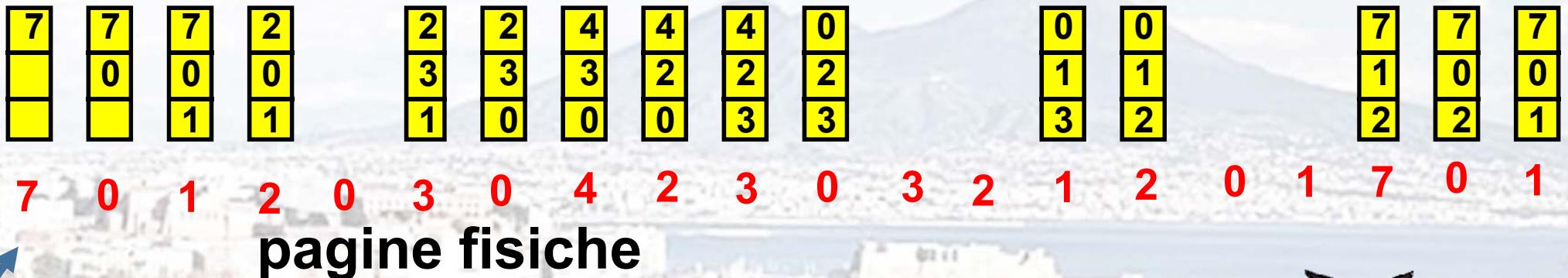


- Sostituzione FIFO – Anomalia di Belady
- più blocchi di memoria \Rightarrow meno assenze di pagine



walter.balzano@gmail.com

Algoritmo di sostituzione delle pagine FIFO - 3 slot



successione dei riferimenti:

70120304230321201701

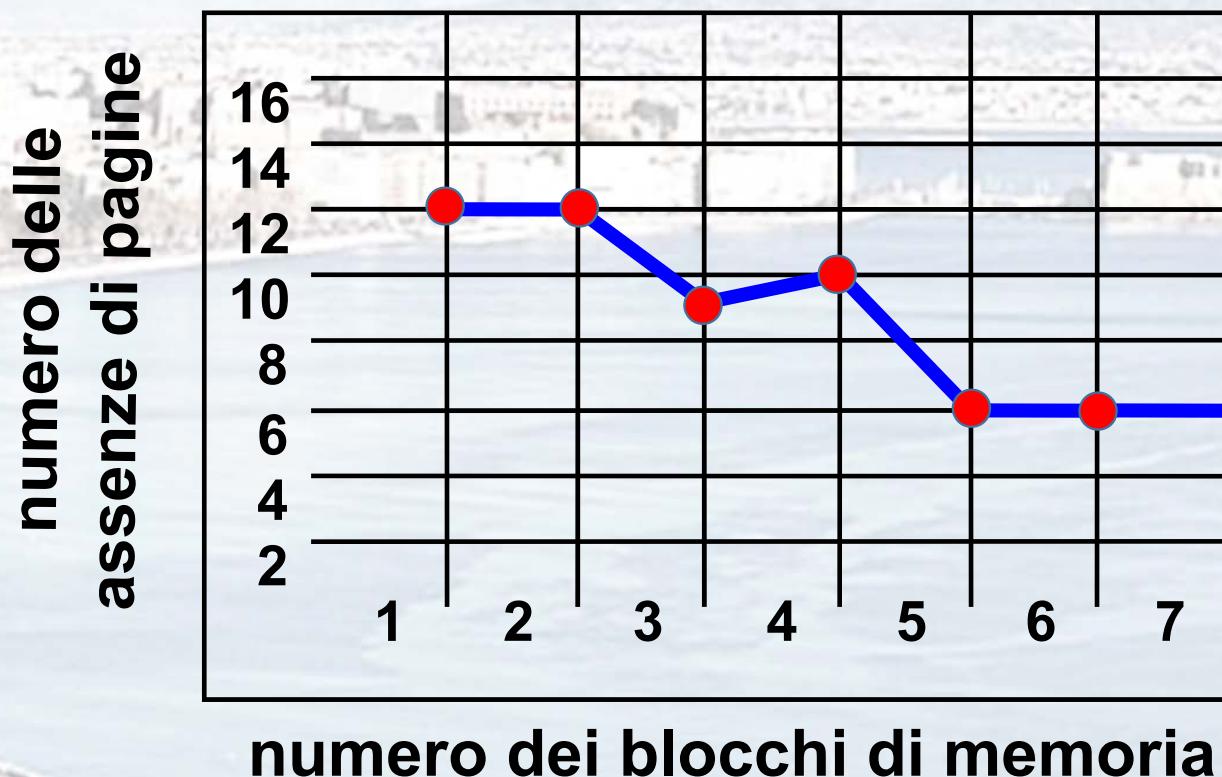
Totale = 15 page fault



walter.balzano@gmail.com

Curva delle assenze di pagine per sostituzione FIFO su una successione di riferimenti

Successione dei riferimenti: 1,2,3,4,1,2,5,1,2,3,4,5



walter.balzano@gmail.com

Sostituzione delle pagine ottimale

Si sostituisce la pagina che non si userà per il periodo di tempo più lungo.

Esempio:

4 blocchi di memoria e **Riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

6 assenze
di pagine



- È possibile conoscere in anticipo la successione dei riferimenti?
- Usato principalmente per studi comparativi, per valutare le prestazioni degli algoritmi.



walter.balzano@gmail.com

Algoritmo ottimale di sostituzione delle pagine

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



blocchi di memoria

successione dei riferimenti:

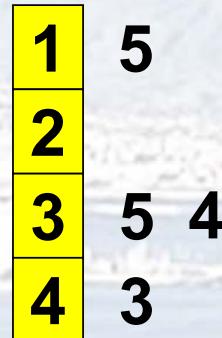
70120304230321201701



walter.balzano@gmail.com

Algoritmo LRU (Least Recently Used)

- Successione dei riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Metodo 1: Contatori

A ogni elemento della tabella delle pagine si associa un campo del momento d'uso, e alla CPU si aggiunge un contatore che si incrementa a ogni riferimento alla memoria.

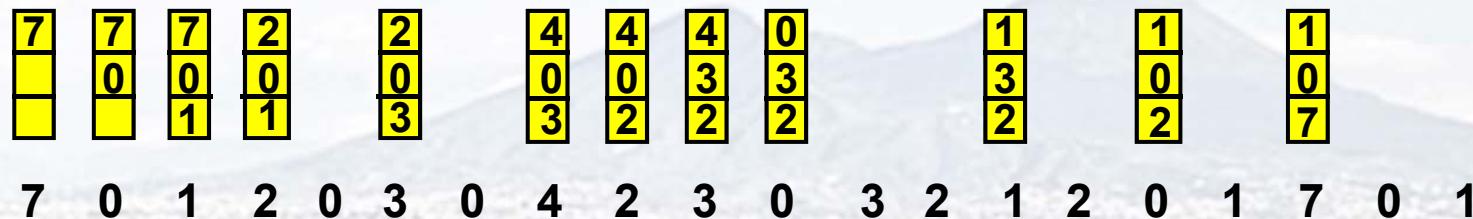
Ogni volta che si fa un riferimento a una pagina, si copia il contenuto del registro contatore nel campo del momento d'uso nella tabella relativa a quella specifica pagina.

- Quando occorre sostituire una pagina, si sostituisce quella con il valore associato più piccolo.



walter.balzano@gmail.com

Algoritmo LRU (Cont.)



blocchi di memoria

successione dei riferimenti
70120304230321201701



walter.balzano@gmail.com

Algoritmo LRU (Cont.)

Metodo 2: Stack:

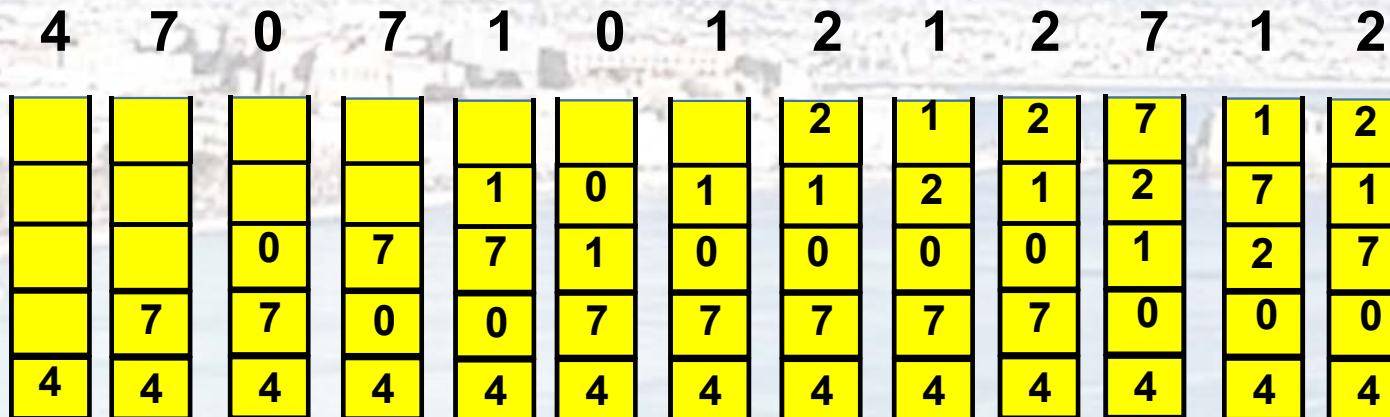
- si ottiene usando una lista doppiamente concatenata, con un puntatore all'elemento iniziale e uno a quello finale.
 - Ogni volta che si fa riferimento a una pagina:
 - La si estrae dallo stack e la si colloca in cima
 - Nel caso peggiore è necessario modificare sei puntatori
 - Per una sostituzione non si deve compiere alcuna ricerca



walter.balzano@gmail.com

Uso di uno Stack per registrare i più recenti riferimenti alle pagine

successione dei riferimenti:



In tal modo:

- in cima allo stack si trova sempre la pagina usata per ultima
- in fondo si trova la pagina usata meno recentemente



walter.balzano@gmail.com

Sostituzione delle pagine per approssimazione a LRU

- **Bit di riferimento**

- A ciascuna pagina è associato un bit, inizialmente = 0
- Ogni volta che si fa riferimento a quella pagina, il bit è impostato a 1.
- Si sostituisce quella che è a 0 (se esiste). Non è però possibile conoscere l'ordine d'uso.

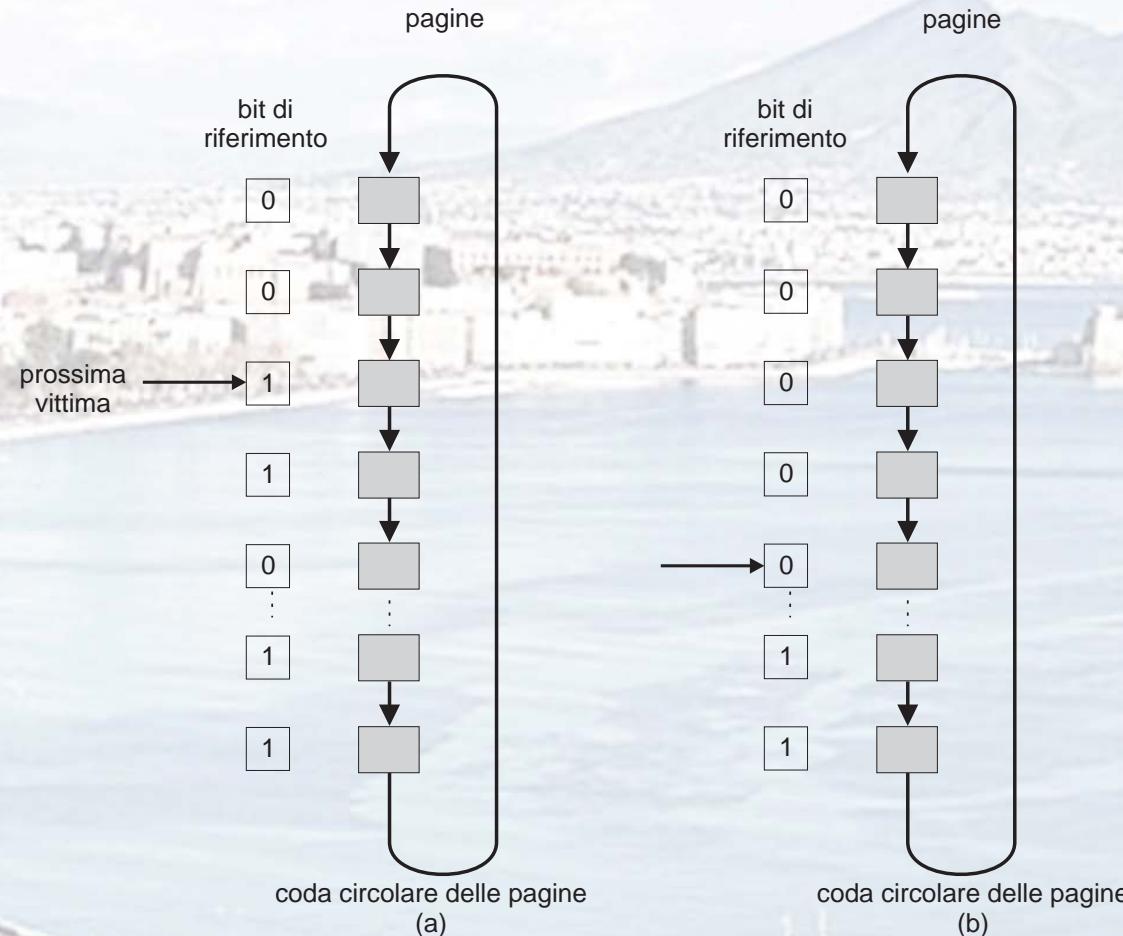
- **Con seconda chance**

- Occorre il bit di riferimento.
- Un puntatore indica qual è la prima pagina da sostituire.
- Se la pagina da sostituire (in ordine di clock) ha il bit di riferimento = 1, allora:
 - imposta il bit di riferimento a 0.
 - lascia la pagina in memoria.
 - sostituisce la pagina successiva (in ordine di clock), soggetta alle stesse regole.



walter.balzano@gmail.com

Algoritmo di sostituzione delle pagine con seconda chance (orologio)



walter.balzano@gmail.com

Sostituzione delle pagine basata su conteggio

- Mantiene un contatore del numero di riferimenti che sono stati fatti a ciascuna pagina.
- Algoritmo LFU (*least frequently used*): sostituisce la pagina con il conteggio più basso.
- Algoritmo MFU (*most frequently used*): è basato sul fatto che, probabilmente, la pagina con il contatore più basso è stata appena inserita e non è stata ancora usata.



walter.balzano@gmail.com

Capitolo 11: Interfaccia del file system

- Concetto di file
- Metodi d'accesso
- Struttura di directory
- Montaggio di un file system
- Condivisione di file
- Protezione



walter.balzano@gmail.com

File Concept

- Un file è un insieme di informazioni, correlate e registrate nella memoria secondaria, cui è stato assegnato un nome
- Tipi:
 - Dati
 - numerici
 - alfabetici
 - alfanumerici
 - binari
 - Programmi



walter.balzano@gmail.com

Struttura dei file

- **Nessuna:** sequenza di parole, byte
- **Strutture semplici**
 - Righe
 - Lunghezza fissa
 - Lunghezza variabile
- **Strutture complesse**
 - Documento formattato
 - Relocatable load file



walter.balzano@gmail.com

Attributi dei file

- **Nome:** è l'unica informazione in forma umanamente leggibile.
- **Tipo:** informazione necessaria ai sistemi che gestiscono tipi di file diversi.
- **Locazione:** puntatore al dispositivo e alla locazione del file in tale dispositivo.
- **Dimensione:** dimensione corrente del file.
- **Protezione:** controlla chi può leggere, scrivere o far eseguire il file.
- **Ora, data e identificazione dell'utente:** dati utili ai fini della protezione e del controllo di utilizzo.

Le informazioni sui file sono conservate nella struttura di directory, che risiede a sua volta nella memoria secondaria.



walter.balzano@gmail.com

Operazioni sui file

- **Creazione** di un file
- **Scrittura** di un file
- **Lettura** di un file
- **Riposizionamento** in un file
- **Cancellazione** di un file
- **Troncamento** di un file
- **Open(F_i)**: ricerca nella directory del disco l'elemento F_i , e ne sposta il contenuto in memoria.
- **Close (F_i)** – rimuove il contenuto dell'elemento F_i dalla memoria alla directory del disco.



walter.balzano@gmail.com

Tipi di file: nomi, estensioni

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information



walter.balzano@gmail.com

Metodi d'accesso

- **Accesso sequenziale**

*read next
write next
reset
no read after last write
(rewrite)*

- **Accesso diretto**

*read n
write n
position to n
read next
write next
rewrite n*

n = numero relativo del blocco



walter.balzano@gmail.com

File ad accesso sequenziale



walter.balzano@gmail.com

Simulazione dell'accesso sequenziale a un file ad accesso diretto

Accesso sequenziale	Realizzazione nel caso di accesso diretto
reset	<code>cp = 0;</code>
read next	<code>read cp; cp = cp+1;</code>
write next	<code>write cp; cp = cp + 1;</code>



walter.balzano@gmail.com

Esempio di indice e relativo file

numero
logico
cognome del record

Adami	
Artusi	
Astolfi	
:	
:	
:	
Rossi	

file indice

Rossi, Mario	previdenza sociale	età

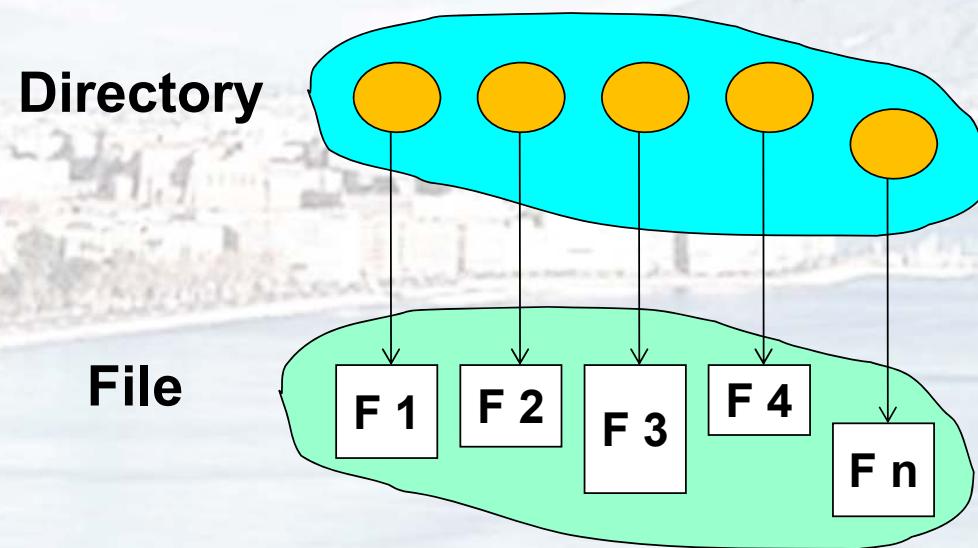
file relativo



walter.balzano@gmail.com

Struttura di directory

Un insieme di nodi contenenti informazioni su tutti i file.

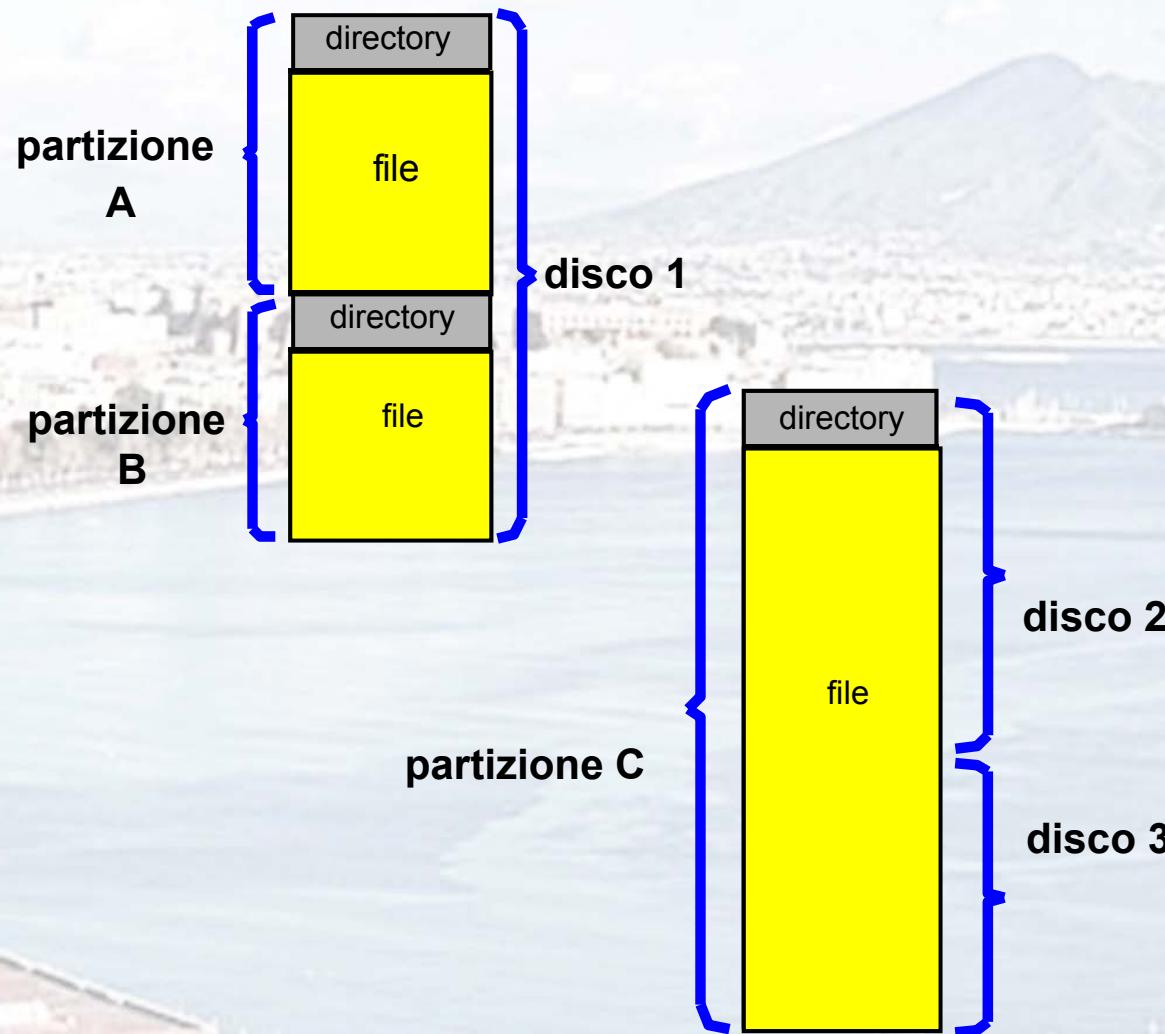


Sia la directory sia i file risiedono sul disco.
Le copie di backup di queste due strutture
vengono in genere archiviate su nastro.



walter.balzano@gmail.com

Tipica organizzazione di un file system



walter.balzano@gmail.com

Informazioni sui file nella directory

- Nome
- Tipo
- Indirizzo
- Dimensione corrente
- Dimensione massima
- Data dell'ultimo accesso
- Data dell'ultimo aggiornamento
- ID del proprietario
- Informazioni di protezione



walter.balzano@gmail.com

Operazioni eseguite su una directory

- **Ricerca di un file**
- **Creazione di un file**
- **Cancellazione di un file**
- **Elencazione di una directory**
- **Ridenominazione di un file**
- **Attraversamento del file system**



walter.balzano@gmail.com

Organizzare (logicamente) una directory per ottenere

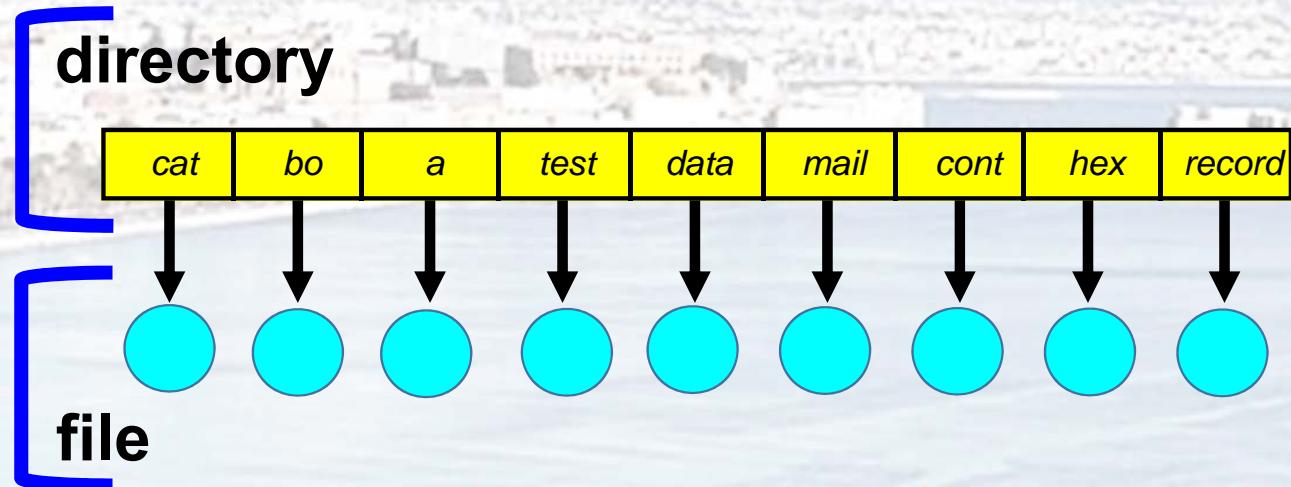
- **Efficienza** – individuare rapidamente un file.
- **Scelta del nome** – conveniente per gli utenti.
 - Due utenti possono usare lo stesso nome per file differenti.
 - Lo stesso file può avere più nomi diversi tra loro.
- **Raggruppamento** – raggruppamento logico del file in base a specifiche caratteristiche (es.: tutti i programmi Java, tutti i giochi, ...)



walter.balzano@gmail.com

Directory a livello singolo

Una singola directory per tutti gli utenti.



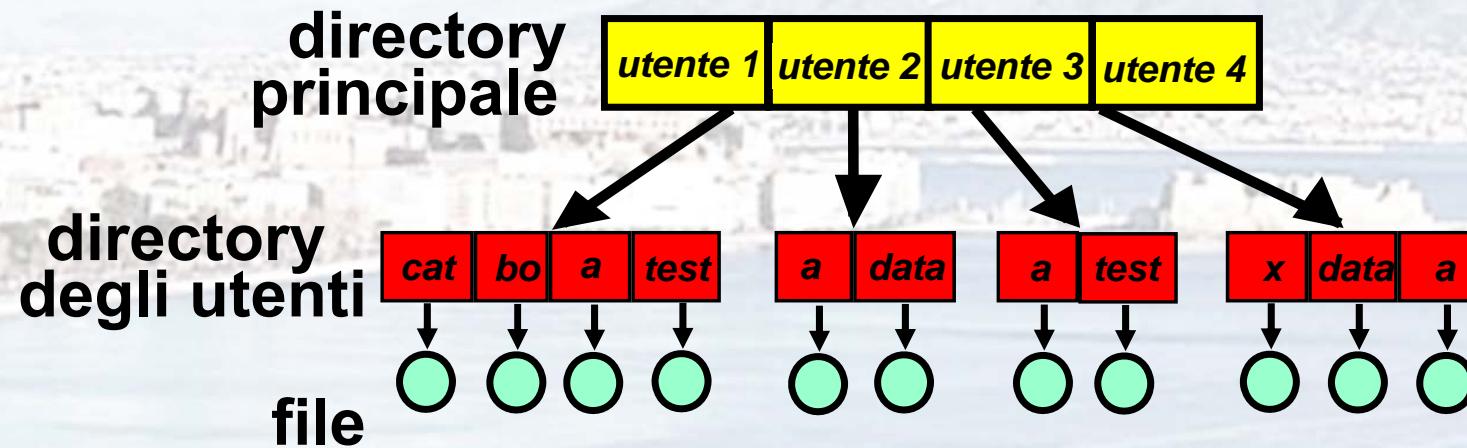
**Confusione nei nomi dei file
Problemi di raggruppamento**



walter.balzano@gmail.com

Struttura di directory a due livelli

Directory separate per ciascun utente.

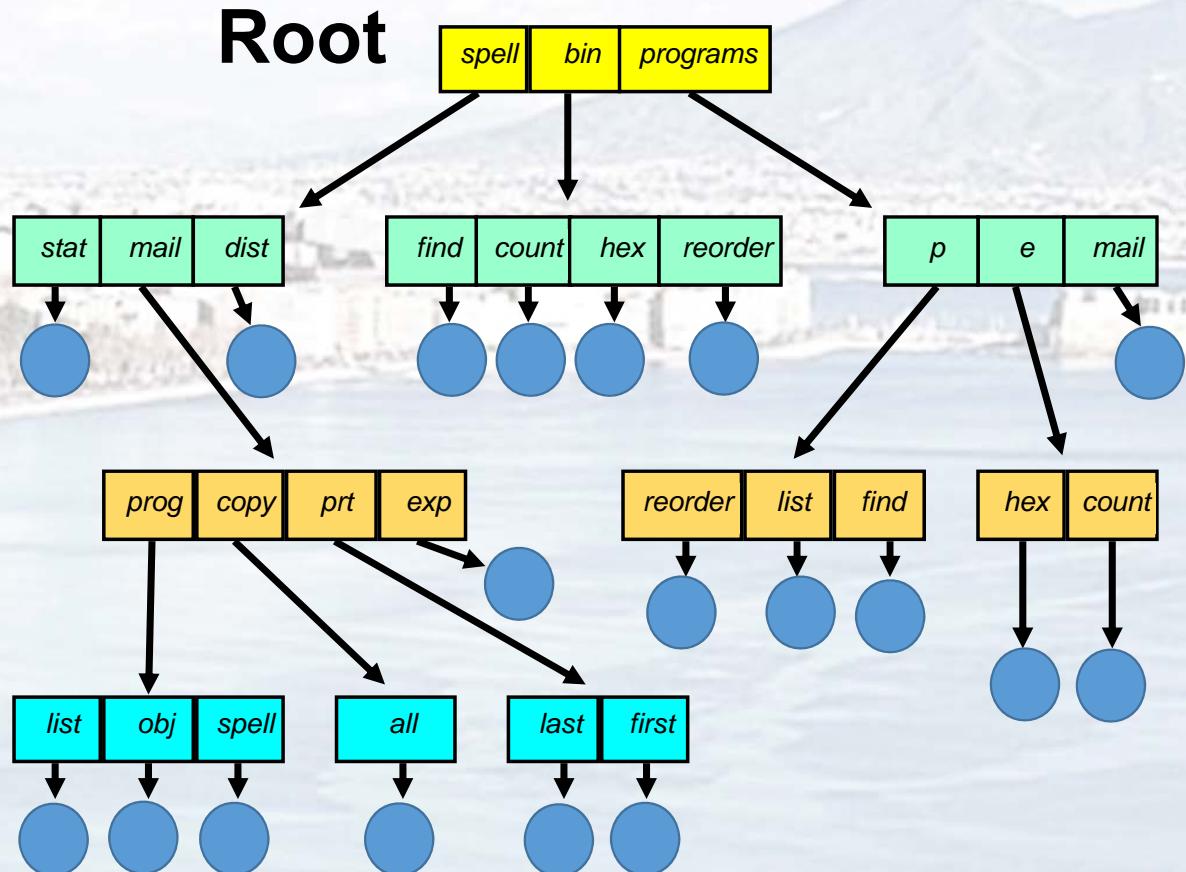


- Nome di percorso (*path name*)
- Un file può avere lo stesso nome per utenti diversi
- Ricerca efficiente
- Non è possibile il raggruppamento



walter.balzano@gmail.com

Struttura di directory ad albero



walter.balzano@gmail.com

Struttura di directory ad albero (cont.)

- Ricerca efficiente
- Possibilità di raggruppamento
- **Directory corrente**
(quella che contiene la maggior parte dei file di interesse corrente)
 - cd /spell/mail/prog
 - type list



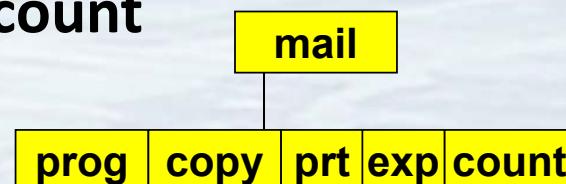
walter.balzano@gmail.com

Struttura di directory ad albero (cont.)

- Nomi di percorso assoluti o relativi
- La creazione di un nuovo file avviene nella directory corrente.
- Cancellazione di un file
`rm <nome_file>`
- La creazione di una nuova sottodirectory avviene nella directory corrente.
`mkdir <nome_dir>`

Esempio: se nella directory corrente `/mail`

`mkdir count`



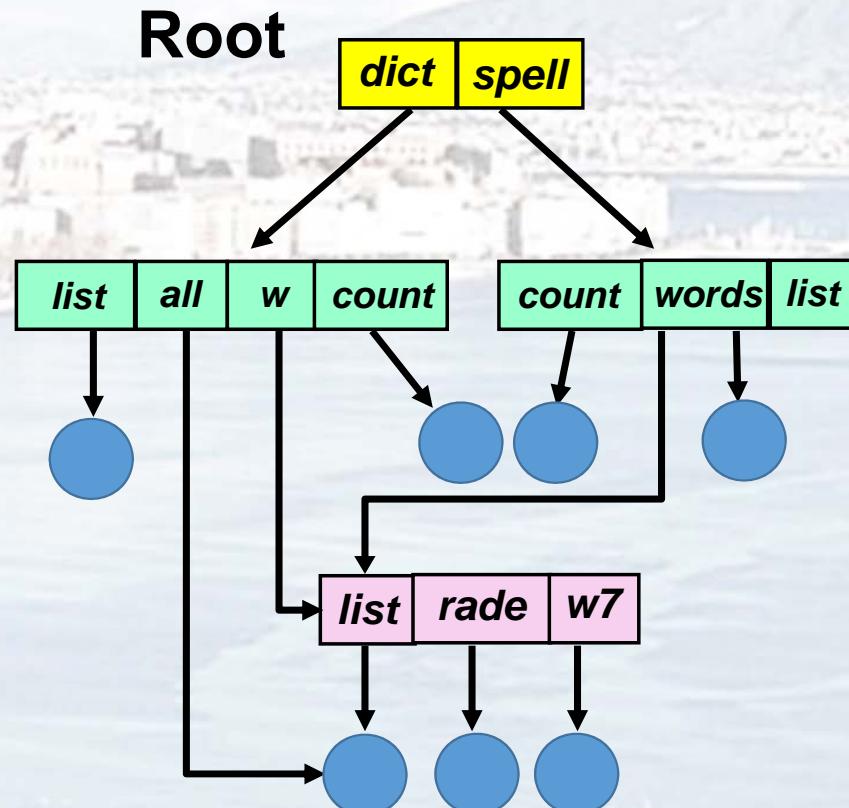
Cancellando “mail” ⇒
si cancella l’intera struttura sottostante.



walter.balzano@gmail.com

Struttura di directory a grafo aciclico

Consente alle directory di avere sottodirectory e file condivisi



walter.balzano@gmail.com

Struttura di directory a grafo aciclico (Cont.)

- Nomi diversi possono riferirsi allo stesso file (*aliasing*)
- se a ogni operazione di cancellazione segue l'immediata rimozione del file, potrebbero rimanere puntatori a un file che ormai non esiste più.

Soluzioni:

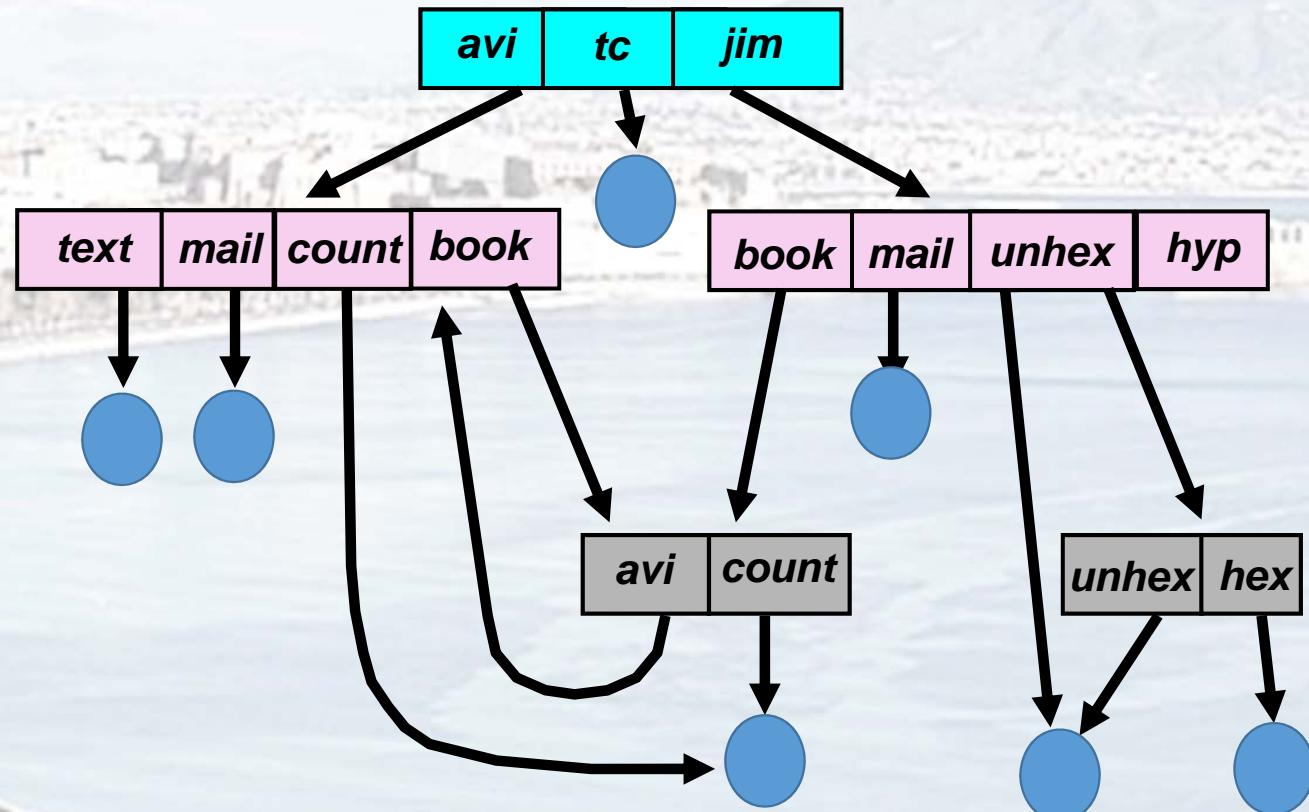
- cercare tutti questi collegamenti e rimuoverli (*backpointers*) .
- conservazione del file fino a che non siano stati cancellati tutti i riferimenti a esso.



walter.balzano@gmail.com

Directory a grafo generale

Root



walter.balzano@gmail.com

Directory a grafo generale (Cont.)

- Come è possibile garantire l'assenza di cicli?
 - Consentire collegamenti solo ai file, non alle sottodirectory.
 - Utilizzare un metodo di “ripulitura” (*garbage collection*).
 - Ogni volta che si aggiunge un nuovo collegamento, utilizzare un semplice algoritmo di individuazione di cicli per verificare che sia tutto OK.



walter.balzano@gmail.com

Montaggio di un file system

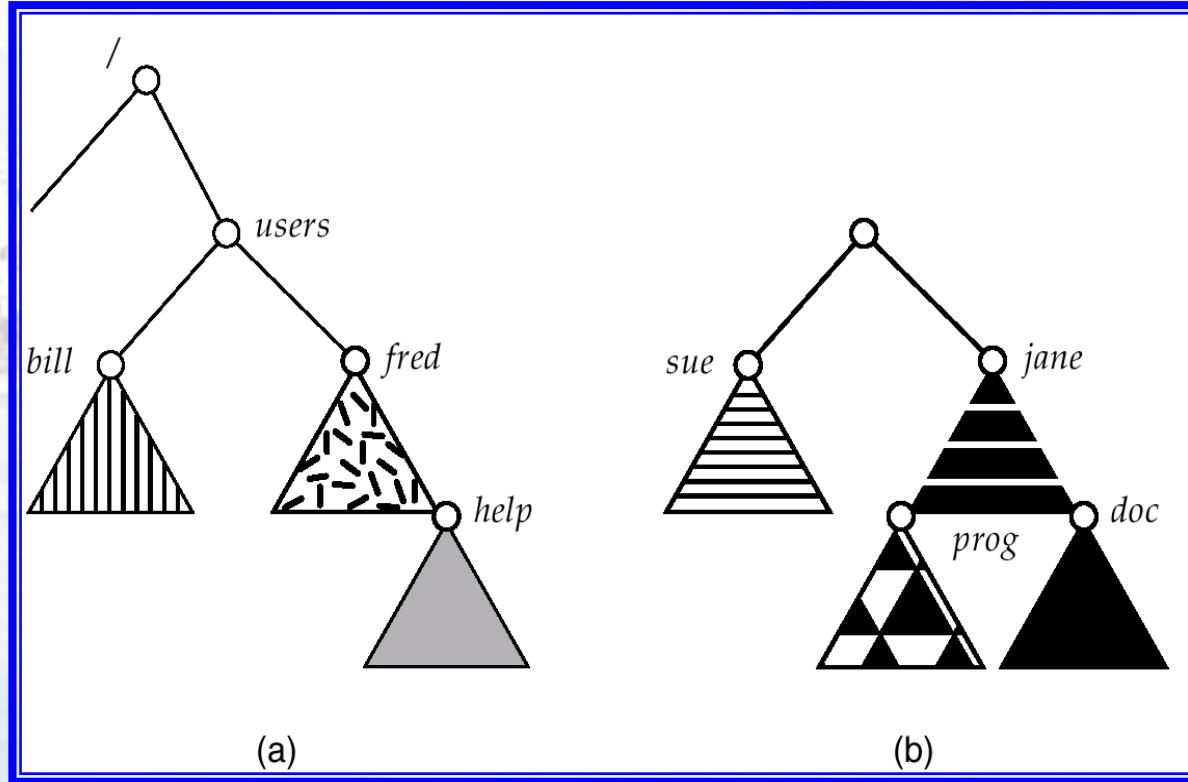
Per essere reso accessibile, un file system deve essere montato.

- La procedura di montaggio è molto semplice: si fornisce al sistema operativo il nome del dispositivo e la sua locazione (detta punto di montaggio, *mount point*) nella struttura di file e directory alla quale agganciare il file system.



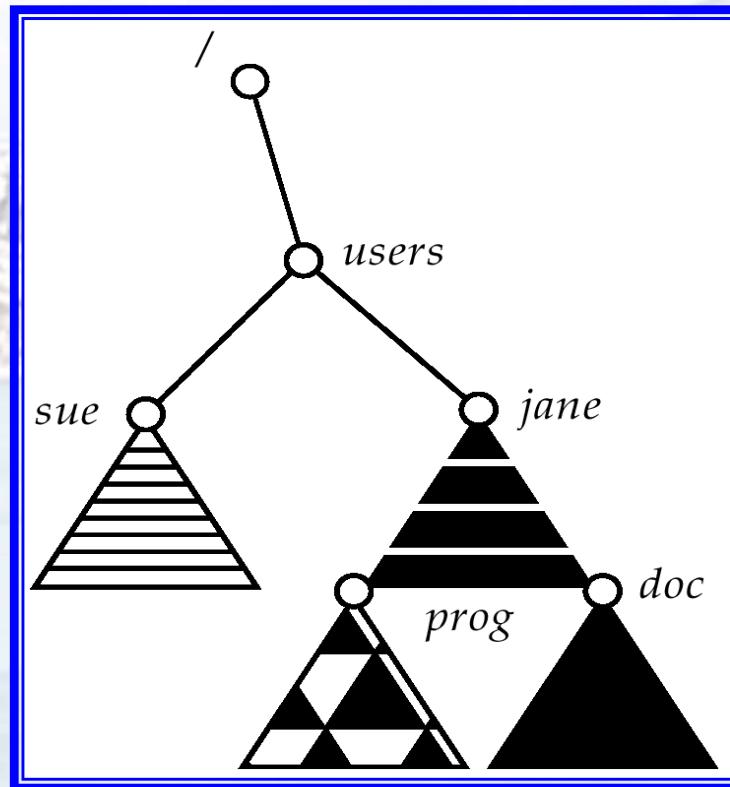
walter.balzano@gmail.com

(a) esistente; (b) partizione non montata



walter.balzano@gmail.com

Punto di montaggio



walter.balzano@gmail.com

Condivisione di file

- La **condivisione** di file nei sistemi multiutente è una caratteristica certamente desiderabile.
- La condivisione può avvenire attraverso uno ***schema di protezione***.
- Nei sistemi distribuiti, i file sono condivisi attraverso una rete.
- Il file system di rete (**NFS, network file system**) è un metodo comune di condivisione di file.



walter.balzano@gmail.com

Protezione

- Il proprietario/creatore di un file deve essere in grado di controllare:
 - che cosa può essere fatto
 - da chi
- Tipi di accesso
 - Lettura
 - Scrittura
 - Esecuzione
 - Aggiunta
 - Cancellazione
 - Elencazione



walter.balzano@gmail.com

Controllo degli accessi e gruppi

Modalità di accesso: lettura (R), scrittura (W), esecuzione (X)

Tre classi di utenti

a) proprietario 7 \Rightarrow **RWX**
 1 1 1

b) gruppo 6 \Rightarrow **RWX**
 1 1 0

c) universo 1 \Rightarrow **RWX**
 0 0 1



walter.balzano@gmail.com

Capitolo 12: Realizzazione del file system

- **Struttura del file system**
- **Realizzazione del file system**
- **Realizzazione delle directory**
- **Metodi di assegnazione**
- **Gestione dello spazio libero**
- **Efficienza e prestazioni**
- **Ripristino**
- **NFS**



walter.balzano@gmail.com

Struttura del file system

- Struttura del file
 - Unità di memorizzazione logica
 - Raccolta di informazioni correlate
- Il file system risiede permanentemente nella memoria secondaria.
- Il file system ha una struttura stratificata.
- Blocco di controllo dei file (**FCB**, *file control block*), struttura di memorizzazione che contiene informazioni sui file, come la proprietà, i permessi e la posizione del contenuto.



walter.balzano@gmail.com

File system stratificato

programmi d'applicazione

file system logico

modulo di organizzazione dei file

file system di base

controllo dell'I/O

dispositivi



walter.balzano@gmail.com

Tipico descrittore di file

permessi per il file

**data e ora di creazione, di ultimo accesso
e di ultima scrittura**

proprietario del file, gruppo, ACL

dimensione del file

blocchi di dati del file



walter.balzano@gmail.com

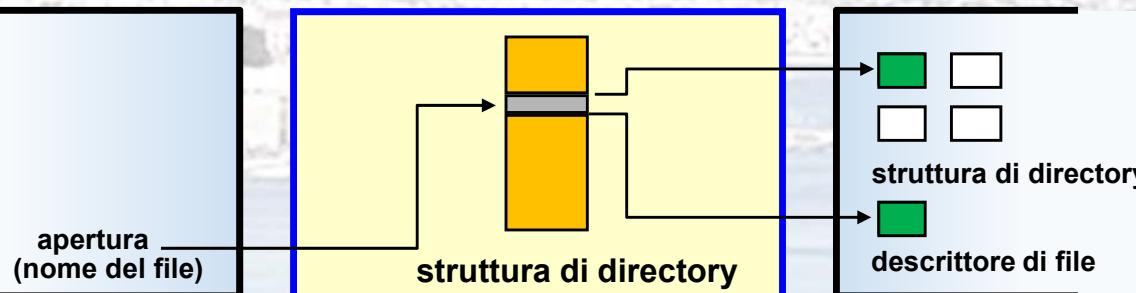
Strutture del file system che si mantengono nella memoria

spazio
d'utente

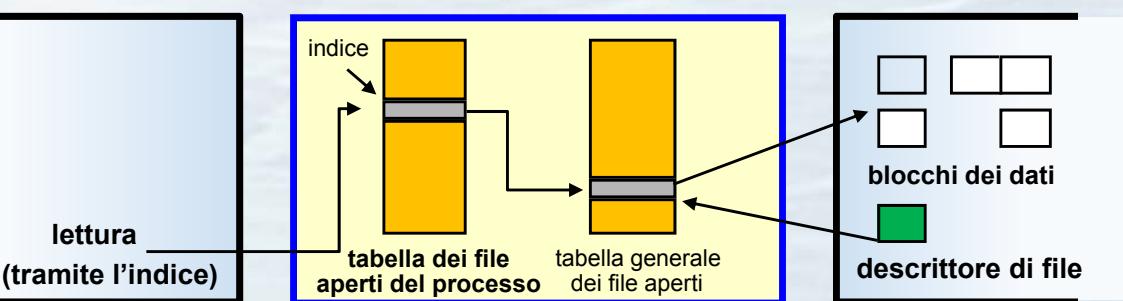
memoria
del nucleo

memoria
secondaria

**APERTURA
file**



**LETTURA
file**



walter.balzano@gmail.com

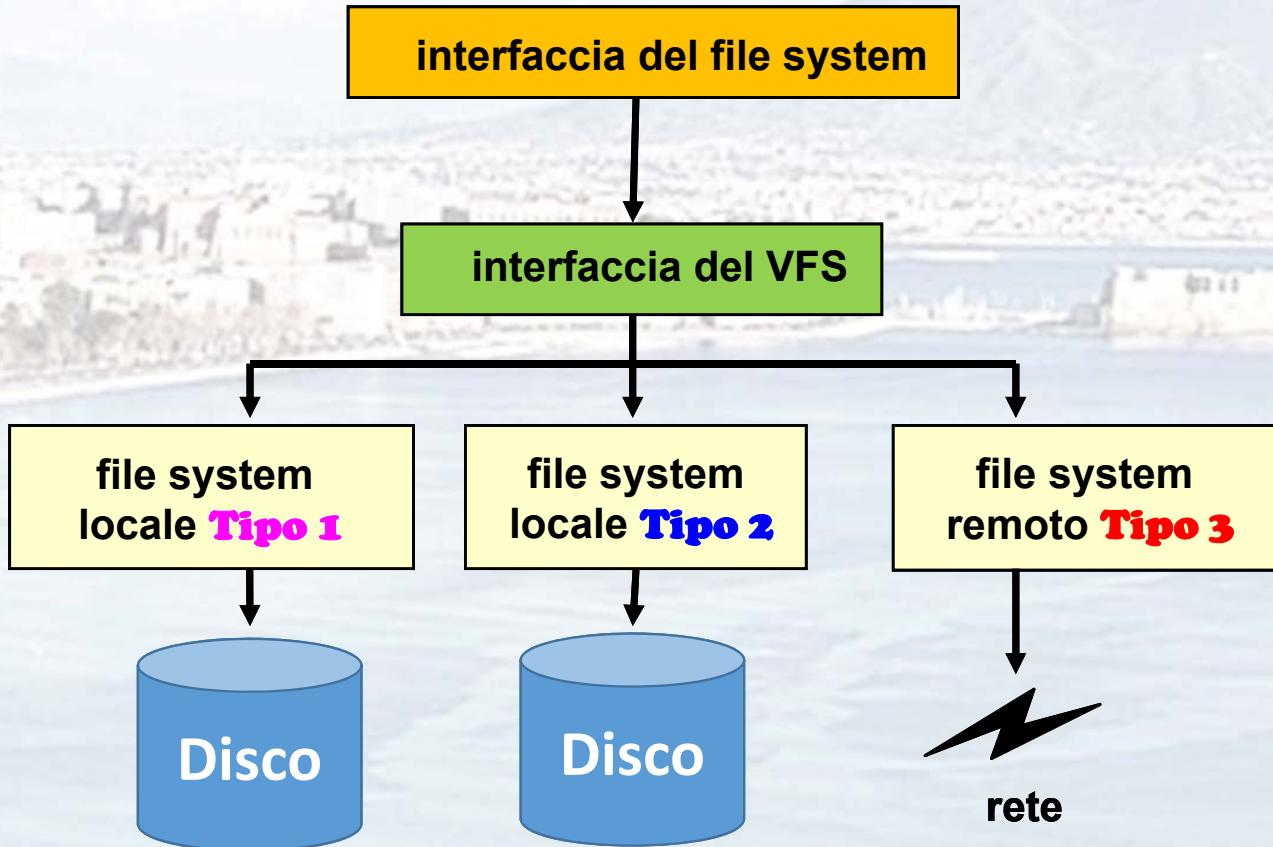
File system virtuali

- Il **file system virtuale** (VFS, *virtual file system*) fornisce una tecnica object-oriented per la realizzazione del file system.
- Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme.
- Il VFS è basato su una struttura di rappresentazione dei file detta **vnode** che contiene un indicatore numerico unico per tutta la rete di ciascun file.



walter.balzano@gmail.com

Schema di un file system virtuale



walter.balzano@gmail.com

Realizzazione delle directory

- **Lista lineare contenente i nomi dei file con puntatori ai blocchi di dati.**
 - metodo di facile programmazione
 - esecuzione onerosa in termini di tempo

-
- **Tabella hash: lista lineare con struttura di dati hash.**
 - diminuisce notevolmente il tempo di ricerca nella directory
 - collisioni: situazioni in cui da due nomi di file si ottiene un riferimento alla stessa locazione
 - dimensione fissa



walter.balzano@gmail.com

Metodi di assegnazione

**Un metodo di assegnazione
indica il modo in cui i blocchi di
disco vengono assegnati ai file:**

- **Assegnazione contigua**
- **Assegnazione concatenata**
- **Assegnazione indicizzata**



walter.balzano@gmail.com

Assegnazione contigua

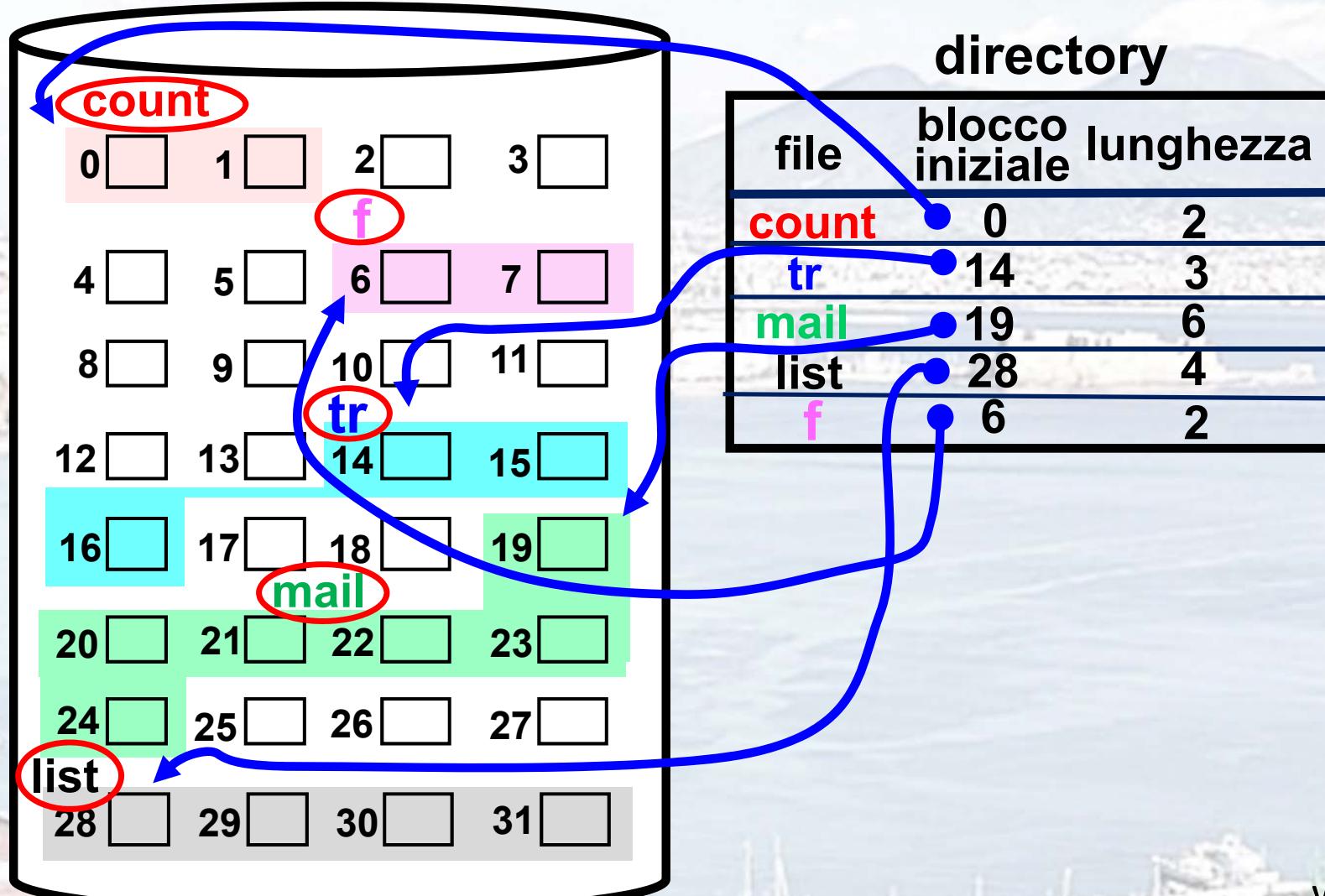
Ciascun file deve occupare un
insieme di blocchi contigui nel disco.

- Semplice: **è richiesto solo l'indirizzo del primo blocco** (inteso come numero di blocco) e **la lunghezza** (espressa in blocchi).
- Impiego di spazio (problema dell'assegnazione dinamica della memoria).



walter.balzano@gmail.com

Assegnazione contigua dello spazio dei dischi



walter.balzano@gmail.com

Assegnazione concatenata

Ciascun file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso.

blocco =  puntatore



walter.balzano@gmail.com

Assegnazione concatenata

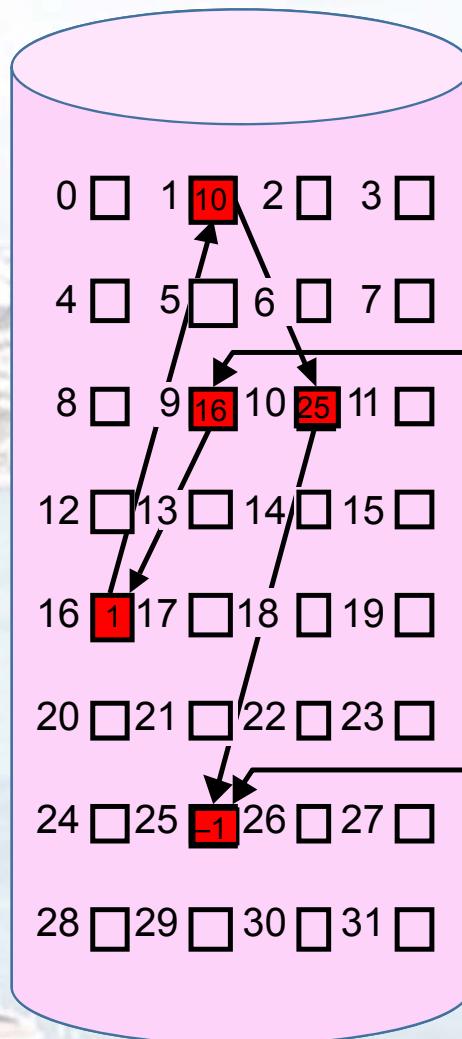
E' semplice perché necessita solo dell'indirizzo di partenza

- Sistema di gestione dello spazio libero:
non c'è spreco di spazio
- **Non vi è accesso casuale**
- Una variante importante del metodo di assegnazione concatenata consiste nell'uso della tabella di assegnazione dei file (**FAT, file allocation table**). Tale metodo di assegnazione, semplice ma efficiente, dello spazio dei dischi è usato nei sistemi operativi MS-DOS e OS/2.



walter.balzano@gmail.com

Assegnazione concatenata



directory		
file	blocco iniziale	blocco finale
jepp	9	25



walter.balzano@gmail.com

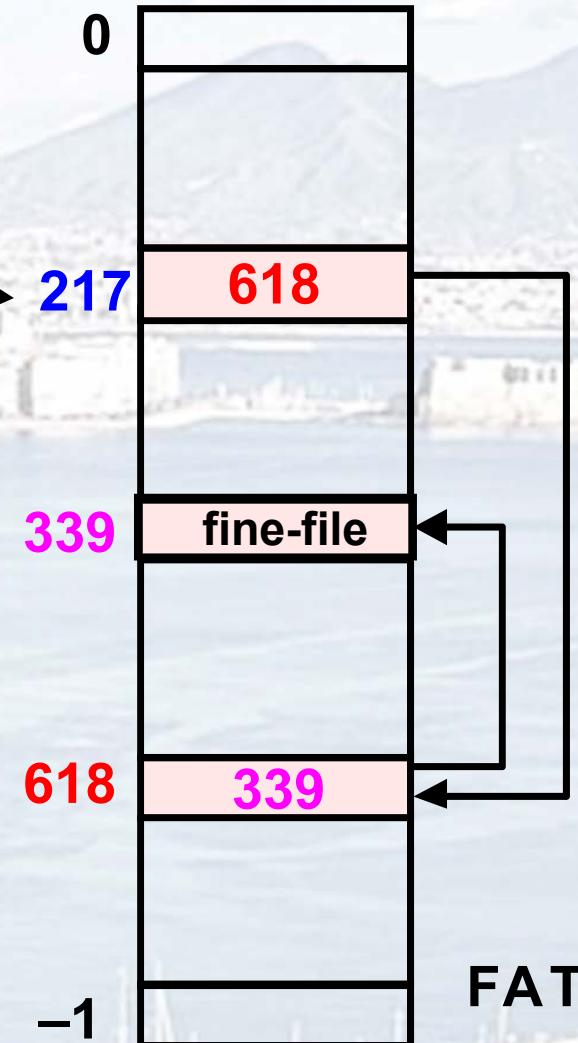
Tabella di assegnazione dei file

elemento della directory



nome

blocco
iniziale



numero dei blocchi
del disco



walter.balzano@gmail.com

Assegnazione indicizzata

Raggruppa tutti i puntatori in una sola locazione: il blocco indice.

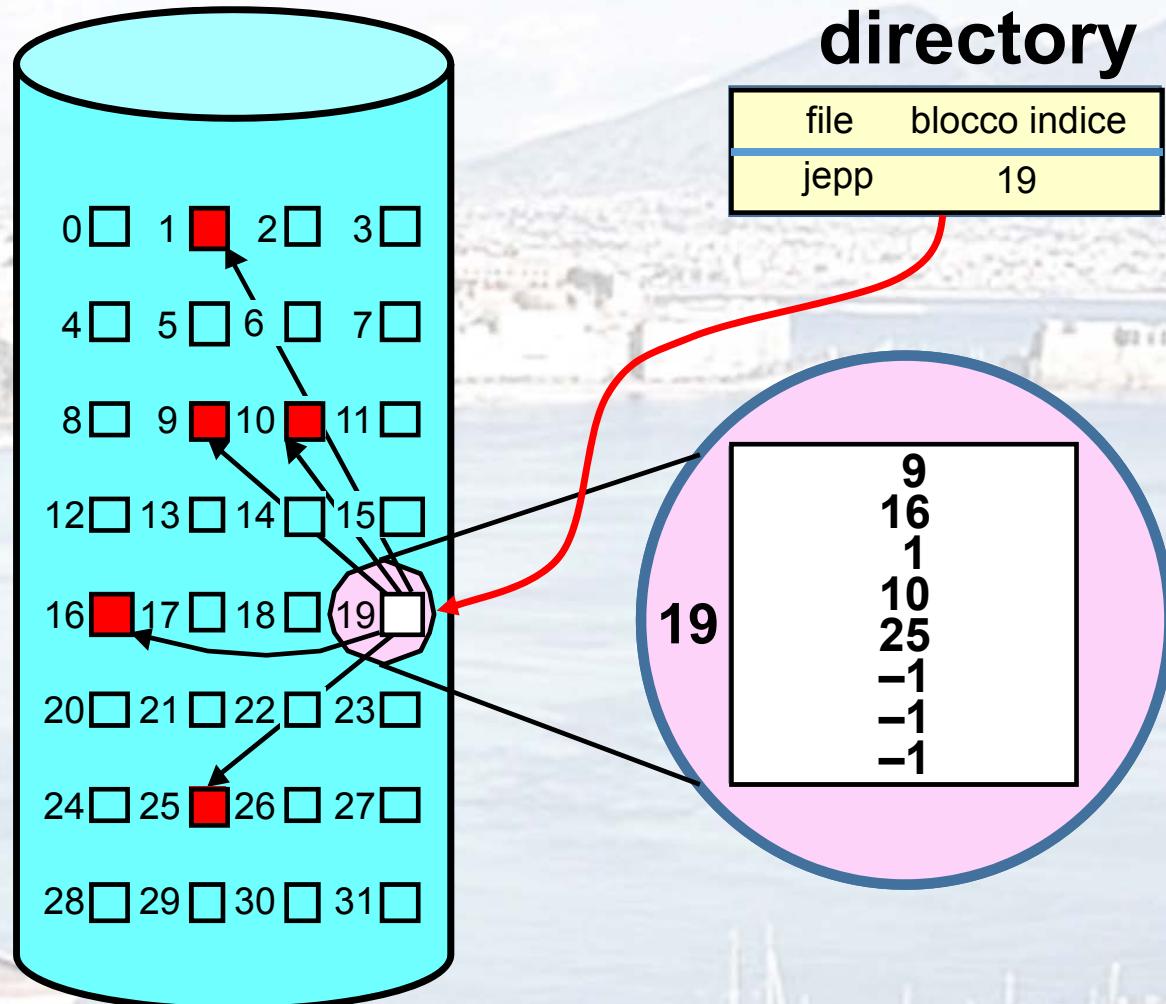


tabella indice



walter.balzano@gmail.com

Esempio di assegnazione indicizzata



walter.balzano@gmail.com

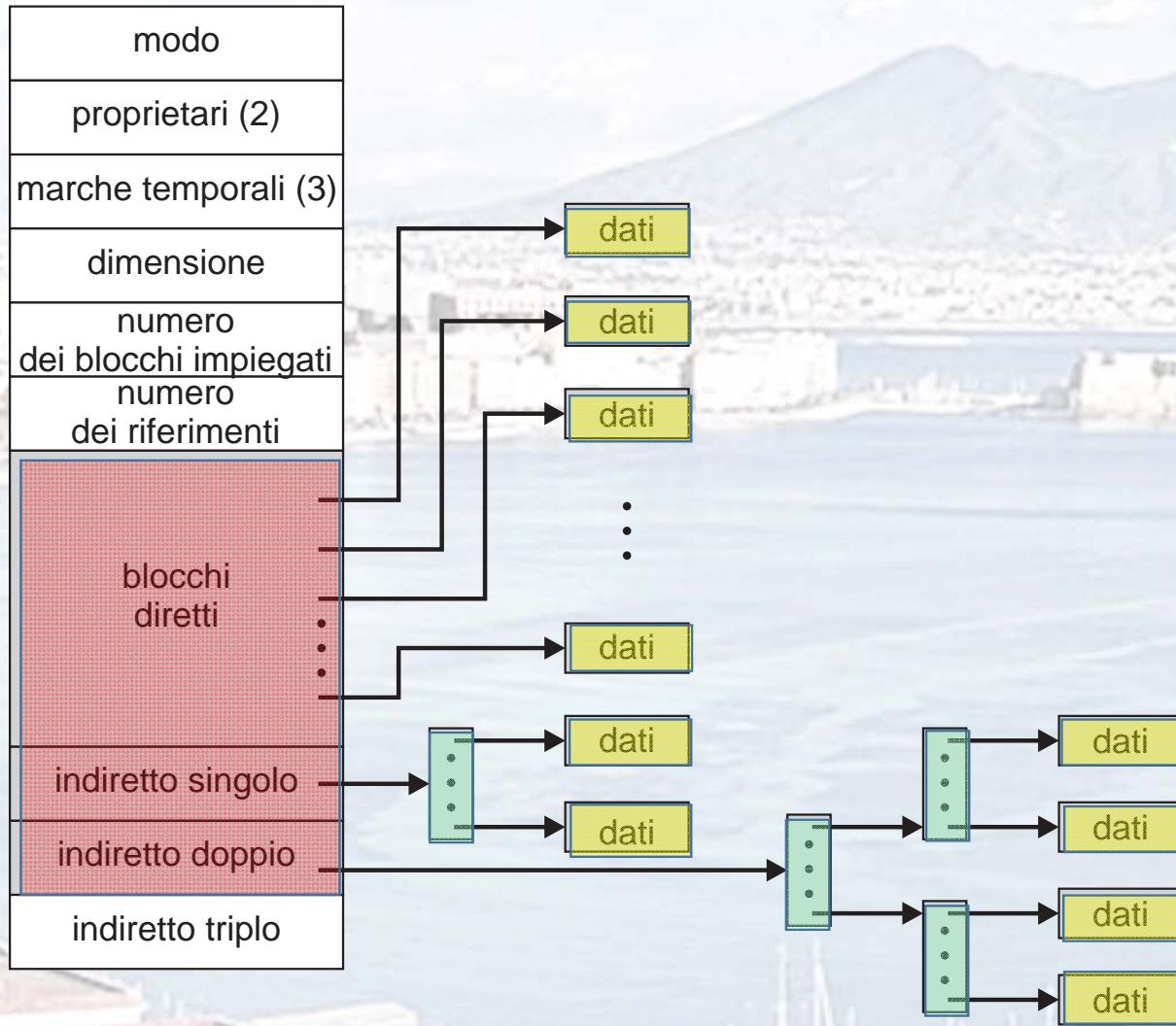
Assegnazione indicizzata

- Necessita di **tabella indice**
- **Accesso casuale**
- **Accesso dinamico senza frammentazione esterna.**
- Ogni file deve avere un blocco indice, quindi è auspicabile che questo sia quanto più piccolo possibile; ma **se il blocco indice è troppo piccolo, non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni**, quindi è necessario disporre di un meccanismo per la gestione di questa situazione:
 - **schema concatenato**
 - **indice a più livelli**
 - **schema combinato.**



walter.balzano@gmail.com

Schema combinato: UNIX



walter.balzano@gmail.com

Gestione dello spazio libero

- Vettore di bit (n blocchi)



$$\text{bit}[i] = \begin{cases} 1 & \Rightarrow \text{blocco}[i] \text{ libero} \\ 0 & \Rightarrow \text{blocco}[i] \text{ assegnato} \end{cases}$$

Esempio:

Un disco che ha liberi i seguenti blocchi:

2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27

E gli altri sono occupati.

La mappa di bit dello spazio libero è:

001111001111110001100000011100000



walter.balzano@gmail.com

Gestione dello spazio libero

- Vettore di bit (n blocchi)


$$\text{bit}[i] = \begin{cases} 1 & \Rightarrow \text{blocco}[i] \text{ libero} \\ 0 & \Rightarrow \text{blocco}[i] \text{ assegnato} \end{cases}$$

Il numero del primo blocco libero è dato dalla seguente espressione:

(numero di bit per parola) **X**

(numero di parole di valore 0) **+**
scostamento del primo bit 1



walter.balzano@gmail.com

Gestione dello spazio libero

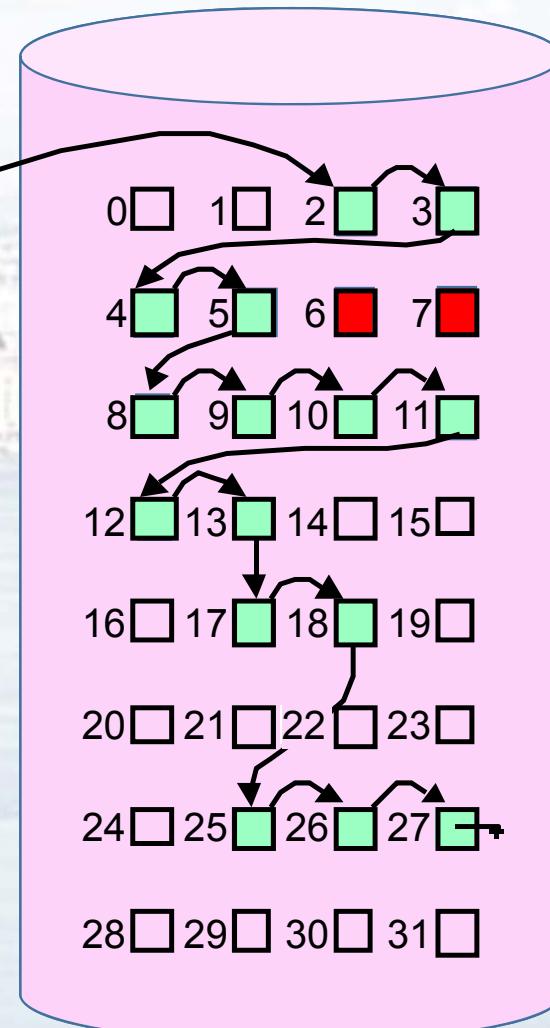
- I vantaggi principali di questo metodo sono la sua relativa semplicità ed efficienza nel trovare il primo blocco libero o n blocchi liberi consecutivi nel disco.
- Le caratteristiche dell'architettura guidano le funzioni del sistema operativo. Sfortunatamente, i vettori di bit sono efficienti solo se tutto il vettore è mantenuto nella memoria centrale, e viene di tanto in tanto scritto nella memoria secondaria allo scopo di consentire eventuali operazioni di ripristino; è possibile tenere il vettore nella memoria centrale se i dischi sono piccoli, come quelli usati nei microcalcolatori; tale soluzione non è applicabile ai dischi più grandi.



walter.balzano@gmail.com

Lista concatenata dei blocchi liberi

**primo elemento
della lista
dei blocchi liberi**



walter.balzano@gmail.com

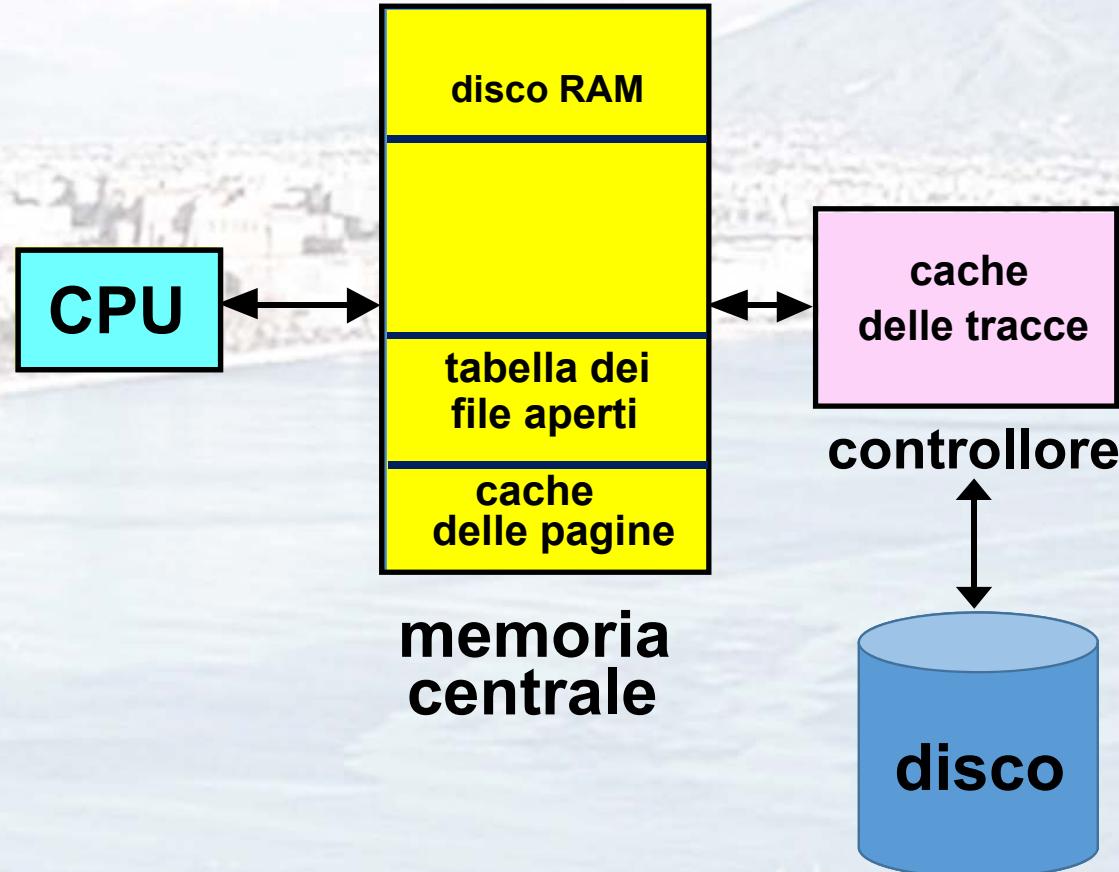
Efficienza e prestazioni

- L'efficienza dipende da:
 - algoritmi usati per l'assegnazione del disco
 - gestione delle directory
- Prestazioni
 - **Cache del disco**: sezione separata della memoria centrale dove tenere i blocchi in previsione di un loro utilizzo entro breve tempo.
 - **Rilascio indietro (free-behind)** e **lettura anticipata (read-ahead)**: tecniche di ottimizzazione degli accessi sequenziali.
 - Per migliorare le prestazioni nei PC si riserva e si gestisce una sezione della memoria come un **disco virtuale** o **disco RAM**.



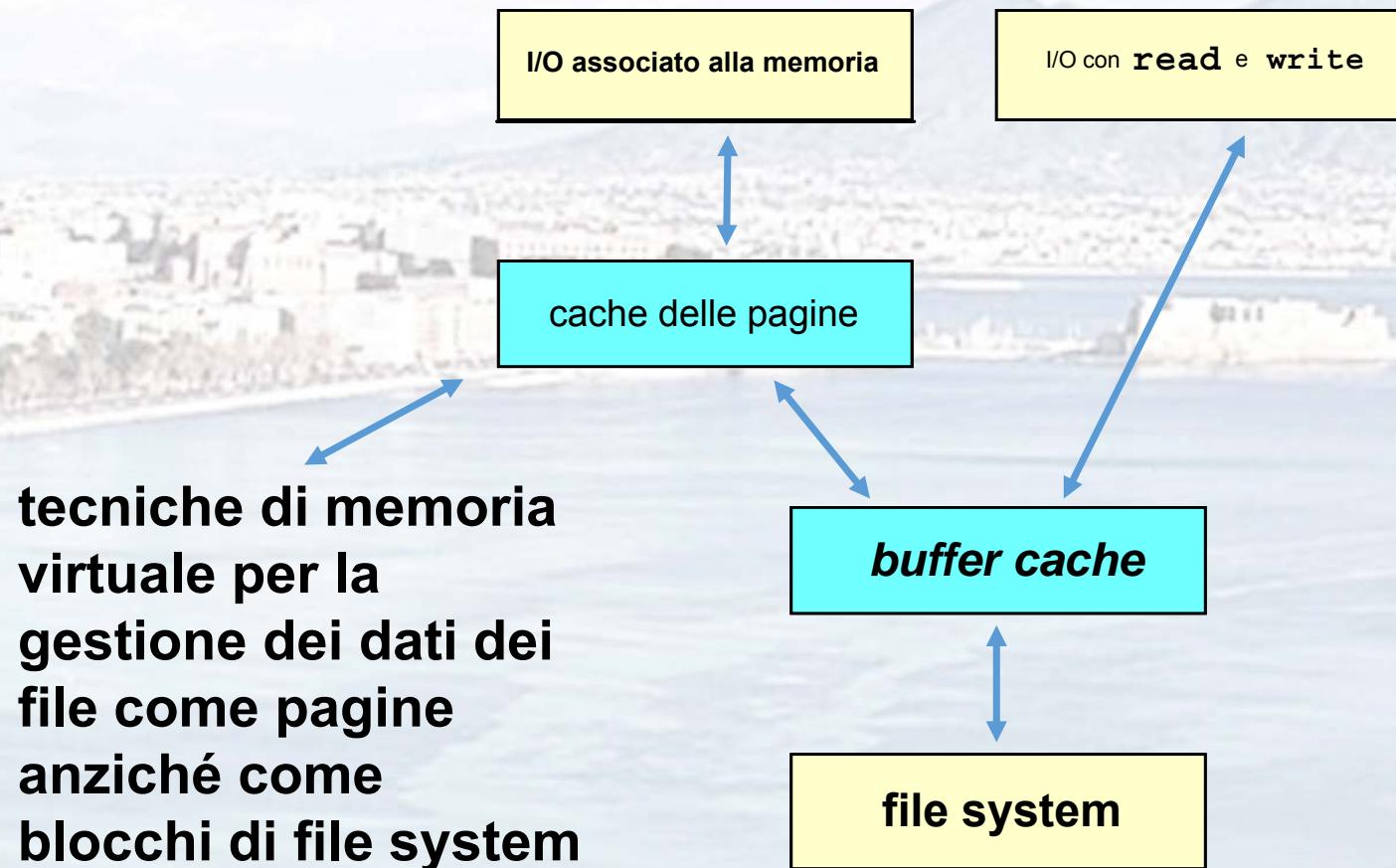
walter.balzano@gmail.com

Diverse locazioni di cache per i dischi



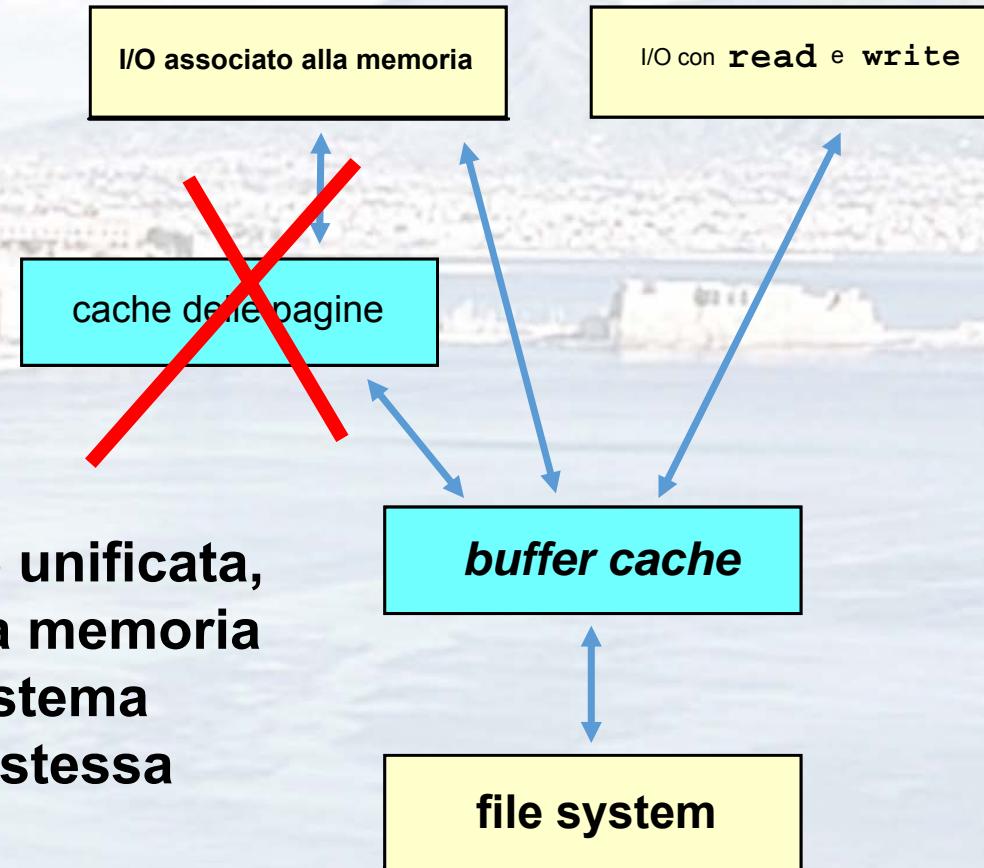
walter.balzano@gmail.com

I/O SENZA una buffer cache unificata



walter.balzano@gmail.com

I/O CON una buffer cache unificata



Con una buffer cache unificata,
sia l'associazione alla memoria
sia le chiamate del sistema
read e **write** usano la stessa
cache delle pagine.



walter.balzano@gmail.com

Ripristino

- **Verifica della coerenza:** si confrontano i dati delle directory con quelli contenuti nei blocchi dei dischi, tentando di correggere ogni incoerenza.
- Si utilizzano i programmi di sistema che consentono di fare delle copie di riserva (**backup**) dei dati residenti nei dischi su altri dispositivi di registrazione dati (dischetti, nastri magnetici o dischi ottici).
- Il ripristino della situazione antecedente la perdita di un singolo file o di un intero disco richiederà il recupero (**restore**) dei dati dalle copie di riserva.



walter.balzano@gmail.com

NFS

- **NFS (Network File System)** è sia una realizzazione sia una definizione di un sistema per l'accesso a file remoti attraverso LAN (o anche WAN).
- Nel contesto dell'NFS si considera un **insieme di stazioni di lavoro interconnesse come un insieme di calcolatori indipendenti con file system indipendenti.** Lo scopo è quello di permettere un certo grado di condivisione tra questi file system, su richiesta esplicita, in modo trasparente.



walter.balzano@gmail.com

NFS

- Una directory remota è montata su una directory di un file system locale. La directory montata assume l'aspetto di un sottoalbero integrante del file system locale, e sostituisce il sottoalbero che discende dalla directory locale.
- La directory remota si specifica come argomento dell'operazione di montaggio in modo esplicito; si deve fornire la locazione (o il nome del calcolatore) della directory remota. Tuttavia, da questo momento in poi, gli utenti possono accedere ai file della directory remota in modo del tutto trasparente.



walter.balzano@gmail.com

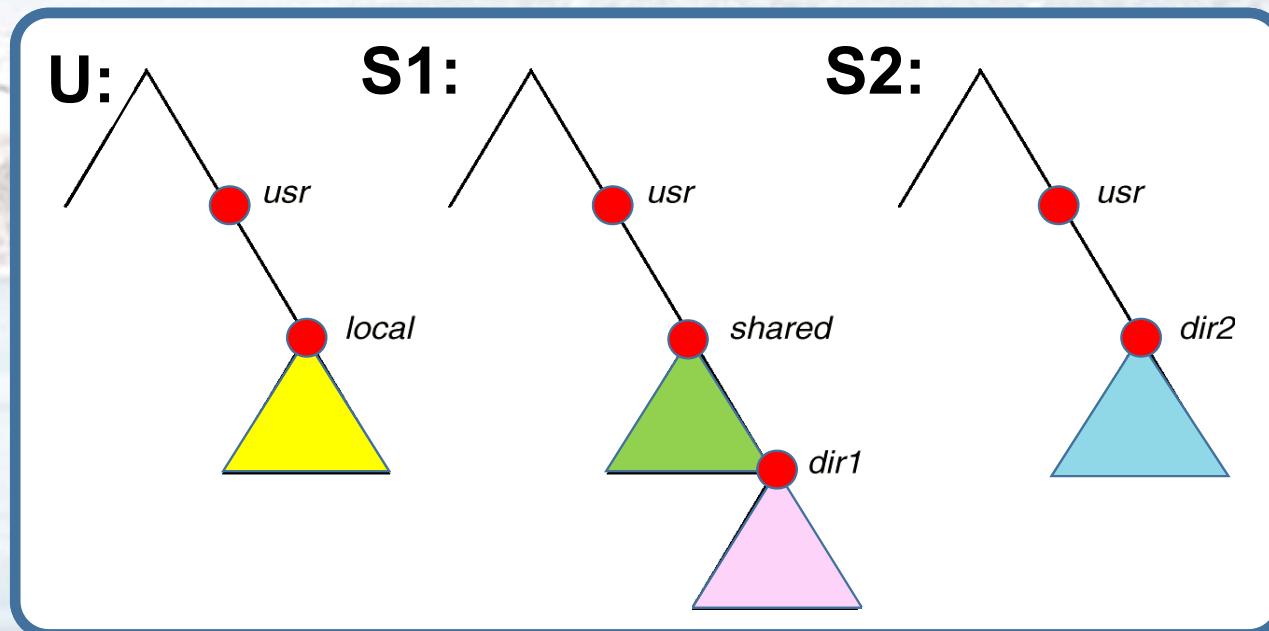
NFS

- Uno degli scopi nella progettazione dell’NFS era quello di **operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete**. La definizione dell’NFS è indipendente da questi mezzi e quindi incoraggia altre realizzazioni.
- Questa indipendenza si ottiene usando primitive RPC costruite su un protocollo i rappresentazione esterna dei dati (XDR, *external data representation*) usato tra due interfacce indipendenti dalla realizzazione.
- La definizione dell’NFS distingue tra i servizi offerti da un meccanismo di montaggio e gli effettivi servizi d’accesso ai file remoti.



walter.balzano@gmail.com

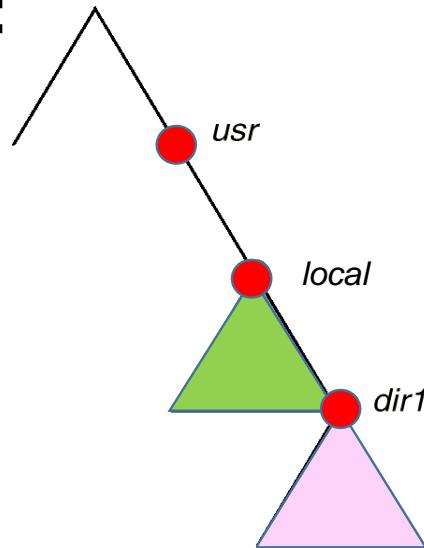
Tre file system indipendenti



walter.balzano@gmail.com

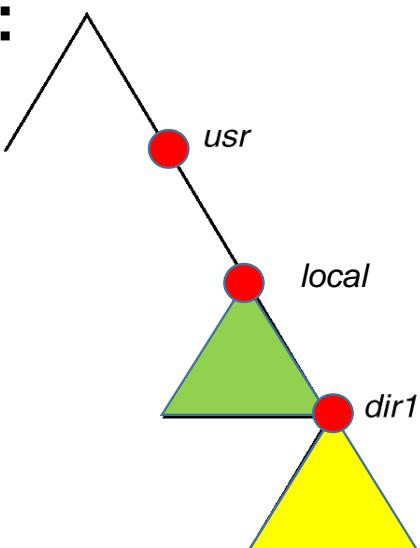
Montaggio nell'NFS

U:



Montaggio

U:



Montaggi a cascata



walter.balzano@gmail.com

Protocollo di montaggio

- Stabilisce la connessione logica iniziale tra un server e un client.
- Comprende il nome della directory remota da montare e il nome del calcolatore server in cui tale directory è memorizzata.
 - La richiesta di montaggi si associa alla RPC corrispondente e s'invia al server di montaggio in esecuzione nello specifico calcolatore server.
 - Lista di esportazione: specifica i file system locali esportati per il montaggio e i nomi dei calcolatori ai quali è permessa tale operazione.
- Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client una maniglia di file (*file handle*) da usare come chiave per ulteriori accessi al file.
- La maniglia di file contiene tutte le informazioni di cui ha bisogno il server per gestire un proprio file.



walter.balzano@gmail.com

Protocollo NFS

- Offre un insieme di RPC per operazioni su file remoti che svolgono le seguenti operazioni:
 - ricerca di un file in una directory
 - lettura di un insieme di elementi di una directory
 - manipolazione di collegamenti e di directory
 - accesso ad attributi di file
 - lettura e scrittura di file
- Una caratteristica importante dei server NFS è l'assenza dell'informazione di stato: ogni richiesta deve fornire un insieme completo di argomenti.
- I dati modificati si devono riscrivere nei dischi del server prima che i risultati siano riportati al client.
- Il protocollo NFS non fornisce meccanismi per il controllo della concorrenza.



walter.balzano@gmail.com

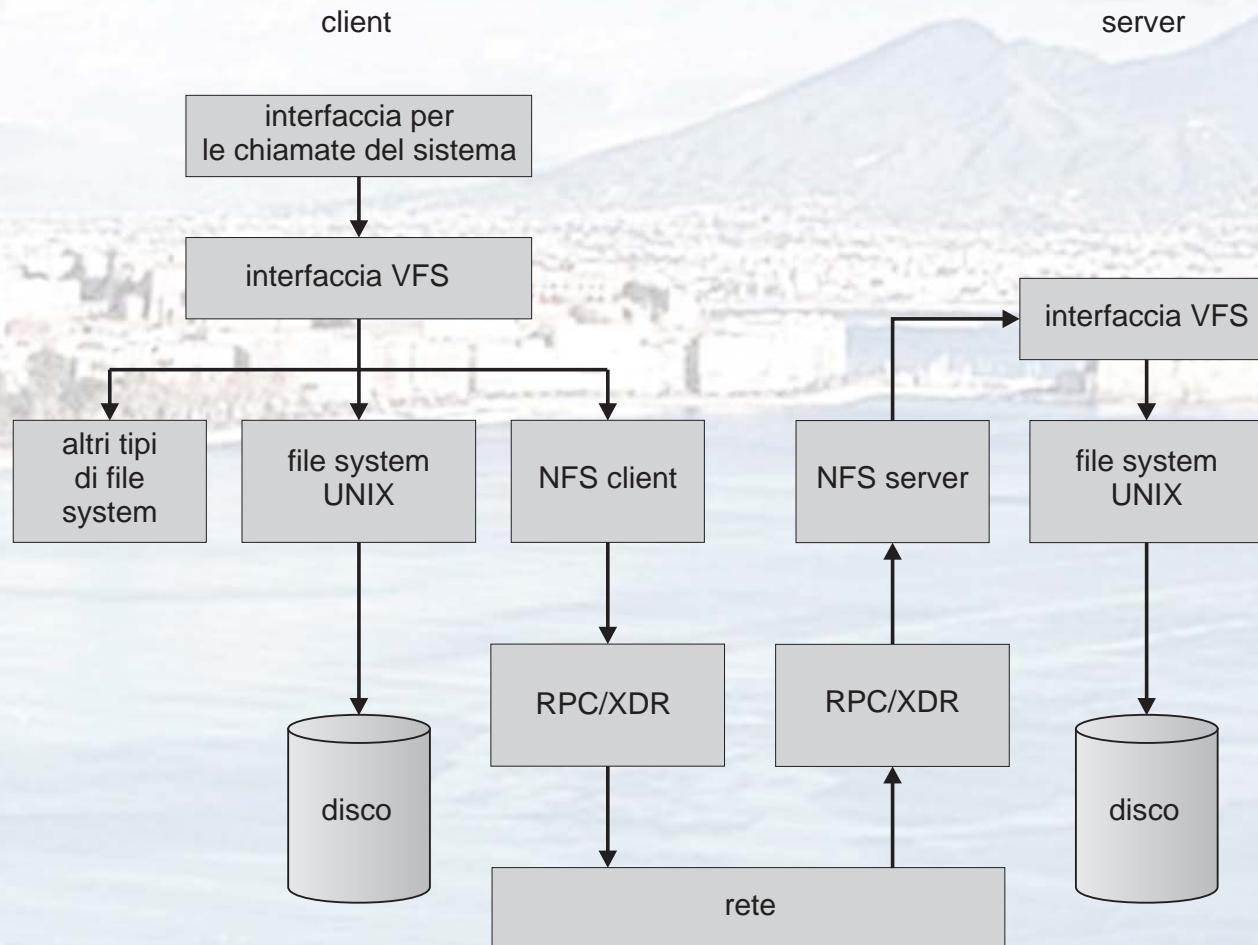
Architettura NFS

- **Interfaccia del file system UNIX** (basata sulle chiamate **open, read, write e close** e sui descrittori di file).
- **File system virtuale** (**VFS, virtual file system**): identifica i file locali da quelli remoti e invoca l'appropriata operazione del file system.
- Un vantaggio di questa architettura è che il client e il server sono identici; così un calcolatore può essere un client, un server o entrambi.



walter.balzano@gmail.com

Schema dell'architettura NFS



walter.balzano@gmail.com

Traduzione dei nomi di percorso

- La traduzione dei nomi di percorso si compie suddividendo il percorso stesso in nomi componenti ed eseguendo una chiamata lookup nell'NFS separata per ogni coppia formata da un nome componente e un *vnode* di directory.
- Una cache per la ricerca dei nomi delle directory, nel sito del client, conserva i *vnode* per i nomi delle directory remote; in questo modo si accelerano i riferimento ai file con lo stesso nome di percorso iniziale.



walter.balzano@gmail.com

Funzionamento remoto

- Tra le normali chiamate del sistema dello UNIX per operazioni su file e le RPC del protocollo NFS esiste una corrispondenza quasi da uno a uno.
- Dal punto di vista concettuale, l'NFS aderisce al paradigma del servizio remoto, ma in pratica si usano tecniche di memorizzazione transitoria e cache per migliorare le prestazioni.
- Non c'è corrispondenza diretta tra un'operazione remota e una RPC, le RPC prelevano blocchi e attributi del file che memorizzano localmente nelle cache. Le successive operazioni remote usano i dati nella cache, soggetti a vincoli di coerenza.
- Esistono due cache: la cache degli attributi dei file (informazioni sugli *inode*) e la cache dei blocchi di file.
- I client non liberano i blocchi di scrittura differita finché il server non ha confermato che i dati sono stati scritti sui dischi.



walter.balzano@gmail.com

Capitolo 13: Sistemi di I/O

- Architetture e dispositivi di I/O
- Interfaccia di I/O per le applicazioni
- Sottosistema per l'I/O del nucleo
- Trasformazione delle richieste di I/O in operazioni dei dispositivi
- STREAMS
- Prestazioni



walter.balzano@gmail.com

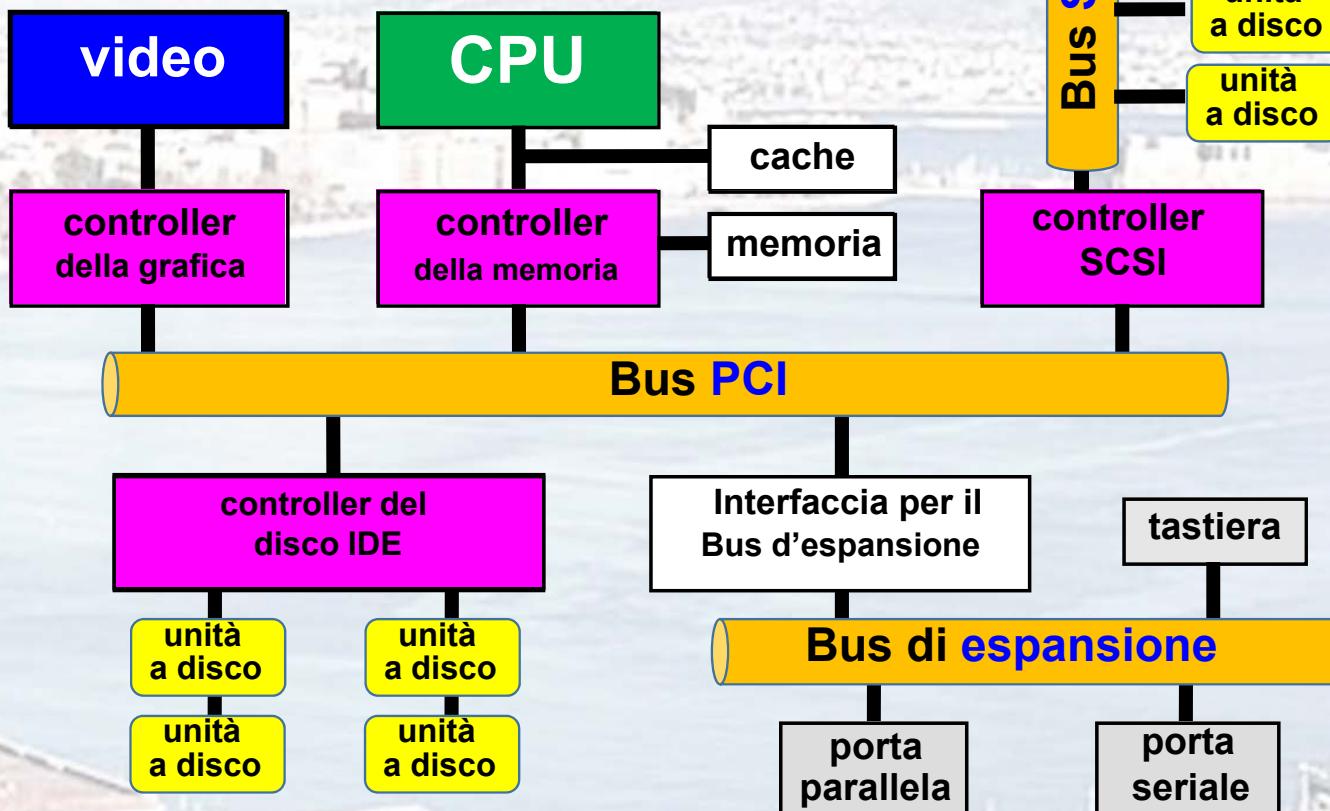
Architetture e dispositivi di I/O

- Grande numero di tipi di dispositivi
- Concetti comuni
 - Porta
 - Bus (collegamento a margherita o accesso diretto condiviso)
 - Controller
- Dispositivi di controllo delle istruzioni di I/O
- I dispositivi hanno indirizzi usati da:
 - Speciali istruzioni di I/O
 - I/O associato alla memoria



walter.balzano@gmail.com

Tipica struttura del bus in un PC



walter.balzano@gmail.com

Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale)

indirizzi per l'I/O (in esadecimale)	dispositivo
000-00F	controller DMA
020-021	controller delle interruzioni
040-043	temporizzatore
200-20F	controller dei giochi
2F8-2FF	porta seriale (secondaria)
320-32F	controller del disco
378-37F	porta parallela
3D0-3DF	controller della grafica
3F0-3F7	controller dell'unità a dischetti
3F8-3FF	porta seriale (principale)



walter.balzano@gmail.com

Interrogazione ciclica (polling)

- Determina lo stato del servizio
 - command-ready
 - busy
 - error
- L'interrogazione ciclica è in sé un'operazione efficiente; tale tecnica diviene però inefficiente se le ripetute interrogazioni trovano raramente un dispositivo pronto per il servizio mentre altre utili elaborazioni attendono la CPU.



walter.balzano@gmail.com

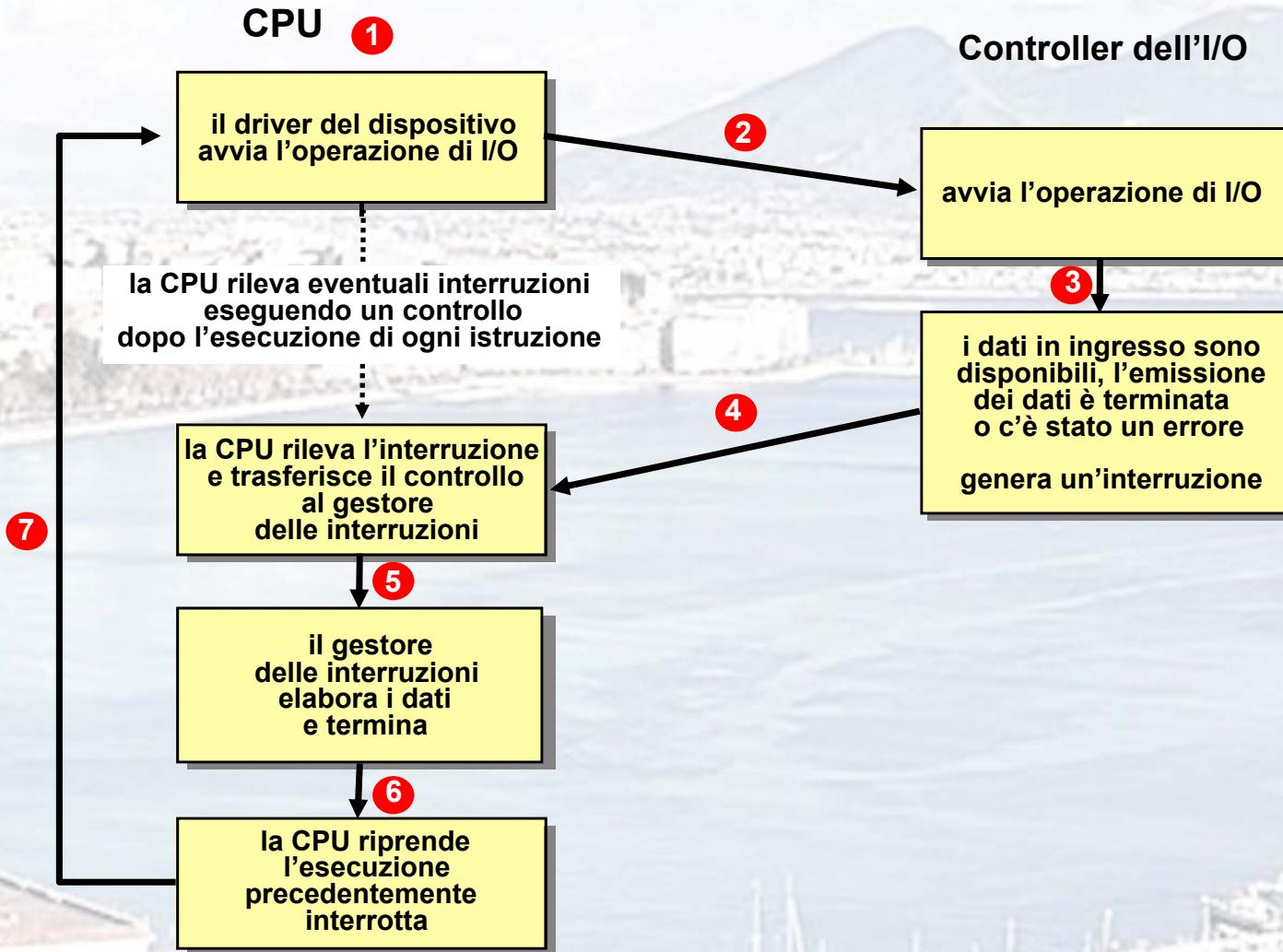
Interrogazioni (interrupt)

- I segnali d'interruzione sono usati diffusamente dai sistemi operativi moderni per gestire eventi asincroni e per eseguire nel modo supervisore le procedure del nucleo.
- I controller dei dispositivi, gli errori e le chiamate del sistema generano segnali d'interruzione al fine d'innescare l'esecuzione di procedure del nucleo.
- Poiché le interruzioni sono usate in modo massiccio per affrontare situazioni in cui il tempo è un fattore critico, è necessario avere un'efficiente gestione delle interruzioni per ottenere buone prestazioni del sistema.



walter.balzano@gmail.com

Ciclo di I/O basato sulle interruzioni



walter.balzano@gmail.com

Vettore delle interruzioni della CPU Intel Pentium

indice del vettore	descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	divide not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19-31	(Intel reserved, do not use)
32-255	maskable interrupts



walter.balzano@gmail.com

Accesso diretto alla memoria (DMA)

- Usato per evitare l'I/O programmato per **trasferimenti di grandi quantità di dati**
- Richiede un controller DMA
- Il controller DMA agisce direttamente sul bus della memoria, presentando al bus gli indirizzi di memoria necessari per eseguire il trasferimento **senza l'aiuto della CPU.**



walter.balzano@gmail.com

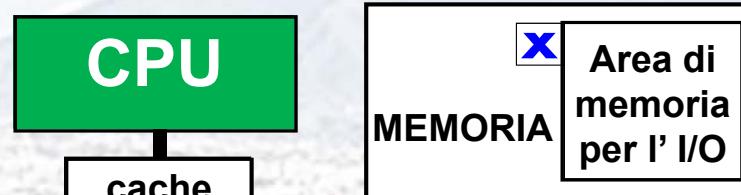
Passi di un trasferimento DMA

5. il controller DMA trasferisce i byte alla locazione di memoria di indirizzo X, incrementando l'indirizzo di memoria e decrementando C finché C = 0

1. al driver del dispositivo si chiede di trasferire dati dal disco al buffer di indirizzo X

2. il driver chiede al controller del disco di trasferire C byte dal disco alla locazione di memoria di indirizzo X

controller DMA



6. quando C = 0, il controller DMA genera un segnale d'interruzione per segnalare alla CPU che il trasferimento è terminato

Bus di memoria della CPU

Bus PCI

controller del disco IDE

unità a disco
unità a disco

unità a disco
unità a disco

3. il controller del disco avvia il trasferimento DMA

4. il controller del disco invia i dati byte per byte al controller DMA



walter.balzano@gmail.com

Interfaccia di I/O per le applicazioni

- Le chiamate del sistema di I/O encapsulano il comportamento dei dispositivi in alcune classi generiche
- Lo scopo dello strato dei driver dei dispositivi è di nascondere al sottosistema di I/O del nucleo le differenze fra i controller dei dispositivi
- I dispositivi possono differire in molti aspetti:
 - trasferimento a flusso di caratteri o a blocchi
 - accesso sequenziale o diretto
 - sincroni o asincroni
 - condivisibili o riservati
 - velocità di funzionamento
 - lettura e scrittura, solo lettura o solo scrittura

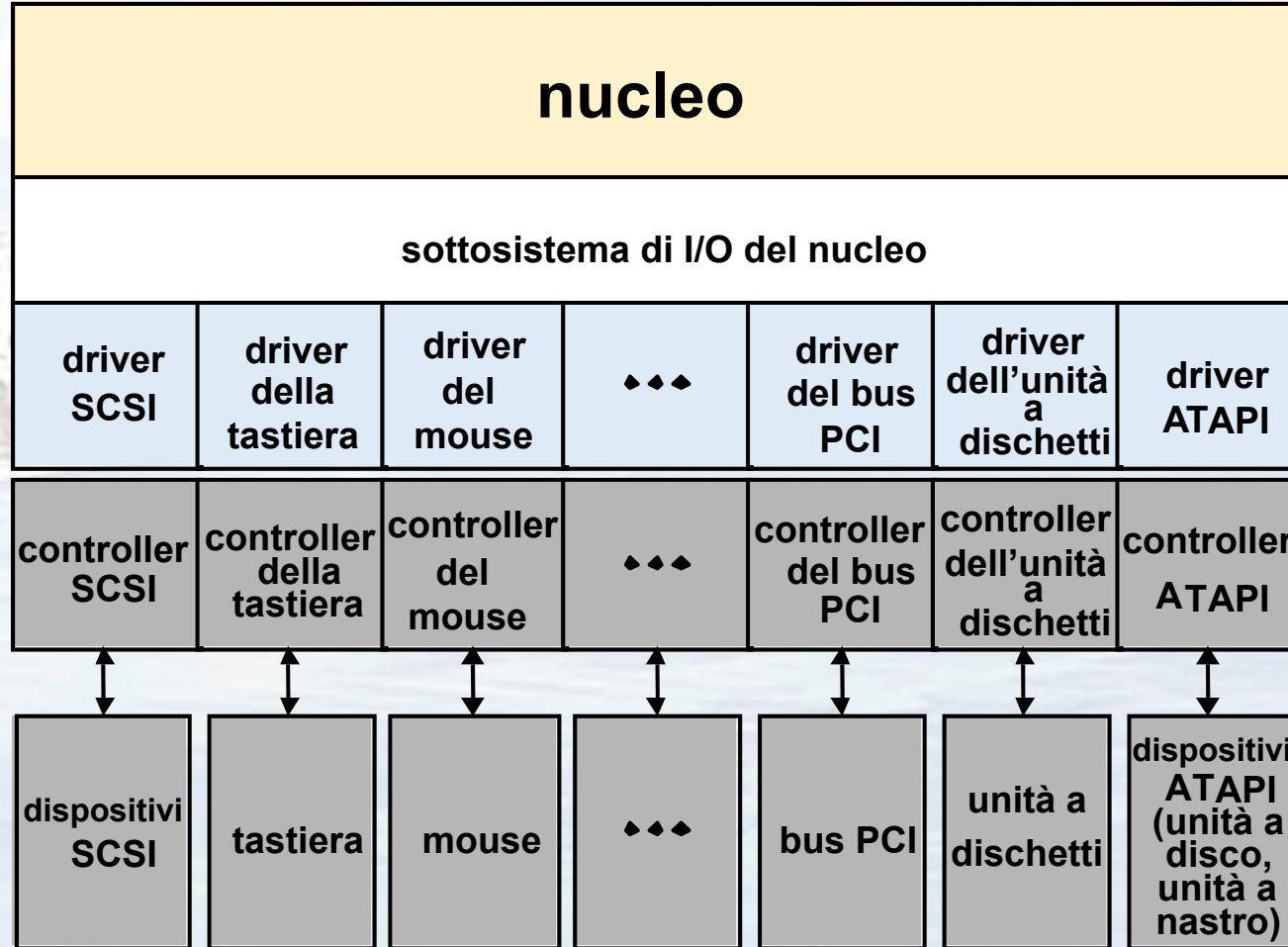


walter.balzano@gmail.com

Struttura relativa all' I/O di un nucleo

programmi

dispositivi fisici



walter.balzano@gmail.com

Caratteristiche dei dispositivi per l'I/O

aspetto	variazione	esempio
modo di trasferimento dei dati	a caratteri a blocchi	terminale unità a disco
modo d'accesso	sequenziale casuale	modem lettore di CD-ROM
prevedibilità dell'I/O	sincrono asincrono	unità a nastro tastiera
condivisione	dedicato condiviso	unità a nastro tastiera
velocità	latenza tempo di ricerca velocità di trasferimento attesa fra le operazioni	
direzione dell'I/O	solο lettura solο scrittura lettura e scrittura	lettore di CD-ROM controller della grafica unità a disco



walter.balzano@gmail.com

Dispositivi con trasferimento a blocchi o a caratteri

- **I dispositivi con trasferimento a blocchi includono le unità a disco**
 - Le istruzioni comprendono read, write e seek
 - I/O a basso livello (*raw I/O*) o accesso tramite file-system
 - Possibile accesso al file associato alla memoria
- **I dispositivi con trasferimento a caratteri includono tastiere, mouse, porte seriali**
 - I comandi comprendono **get**, **put**
 - È possibile costruire servizi aggiuntivi quali l'accesso riga per riga



walter.balzano@gmail.com

Dispositivi di rete

- Poiché i modi di indirizzamento e le prestazioni tipiche dell'I/O di rete sono notevolmente differenti da quelli dell'I/O delle unità a disco, la maggior parte dei sistemi operativi fornisce un'interfaccia per l'I/O di rete diversa da quella caratterizzata dalle operazione `read`, `write` e `seek` usata per i dischi.
- Un'interfaccia disponibile in molti sistemi operativi, tra i quali Unix e Windows NT è l'interfaccia di rete *socket* (letteralmente “presa di corrente”)
 - Separa il protocollo di rete dalle operazioni di rete
 - Include la funzione `select`
- Gli approcci variano ampiamente (pipe half-duplex, code FIFO full-duplex, STREAMS, code di messaggi e socket)



walter.balzano@gmail.com

Orologi e temporizzatori

- **Segnano l'ora corrente, segnalano il tempo trascorso, regolano un temporizzatore**
- Il dispositivo che misura la durata di un lasso di tempo e che può avviare un'operazione si chiama temporizzatore programmabile
- `ioctl` (UNIX) tratta gli aspetti dell'I/O quali orologi e temporizzatori



walter.balzano@gmail.com

I/O bloccante e non bloccante

- **Bloccante:** si sospende l'esecuzione dell'applicazione
 - codice più facilmente comprensibile
 - insufficiente per alcune necessità
- **Non bloccante:** sovrappone elaborazione e I/O
 - interfaccia d'utente, applicazione per il video digitale
 - si realizza attraverso il multithreading
 - restituisce rapidamente il controllo dell'applicazione fornendo un parametro che indica quanti byte di dati sono stati trasferiti
- **Chiamate del sistema asincrone:** restituiscono immediatamente il controllo al chiamante senza attendere che l'I/O sia stato completato
 - di difficile utilizzo
 - Il completamento dell'I/O è successivamente comunicato all'applicazione per mezzo dell'impostazione del valore di una variabile nello spazio d'indirizzi dell'applicazione



walter.balzano@gmail.com

Sottosistema per l'I/O del nucleo

- **Scheduling**

- fare lo scheduling di un insieme di richieste di I/O significa stabilirne un ordine d'esecuzione efficace; l'ordine in cui si verificano le chiamate del sistema delle applicazioni è raramente la scelta migliore.
- alcuni sistemi operativi tentano di essere equi

- **Memorizzazione transitoria:** un buffer (memoria di transito) è un'area di memoria che contiene dati mentre vengono trasferiti tra due dispositivi o fra un'applicazione e un dispositivo

- **necessità di gestire la differenza di velocità fra il produttore e il consumatore di un flusso di dati**
- **gestione dei dispositivi che trasferiscono dati in blocchi di dimensioni diverse**
- **realizzazione della “semantica delle copie”**



walter.balzano@gmail.com

Sottosistema per l'I/O del nucleo

- Cache: **regione di memoria veloce per copie di dati**
 - sempre solo copia di informazioni già memorizzate
 - migliora l'efficienza
- **Code (*spooling*): memoria di transito contenente dati per un dispositivo che non può accettare flussi di dati intercalati**
 - quando il dispositivo può gestire solo una richiesta alla volta
 - es.: una stampante
- **Uso esclusivo dei dispositivi: fornisce accesso esclusivo a un dispositivo**
 - Un processo può accedere a un dispositivo che non sia già attivo riservandosene l'uso e restituendolo al sistema quando non ne ha più bisogno
 - Le applicazioni hanno la responsabilità di evitare situazioni di stallo



walter.balzano@gmail.com

Gestione degli errori

- Un sistema operativo che usi la protezione della memoria può proteggersi da molti tipi di **errori dovuti ai dispositivi o alle applicazioni**
- Di norma, una chiamata del sistema per l'I/O riporta un bit d'informazione sullo stato d'esecuzione della chiamata
- Molti dispositivi SCSI mantengono alcune pagine di informazioni sugli errori avvenuti; queste pagine possono essere richieste dalla macchina, ma ciò accade raramente



walter.balzano@gmail.com

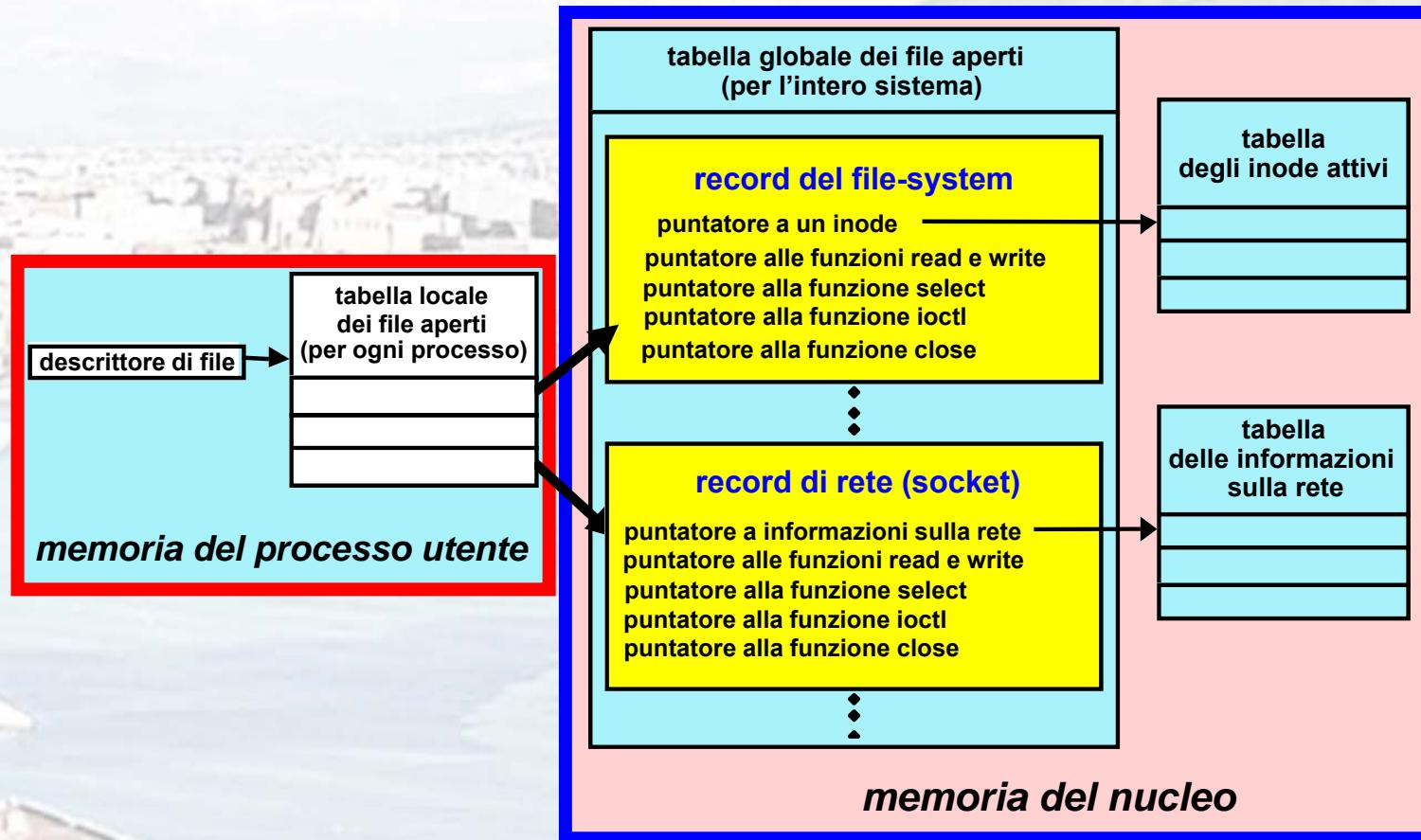
Strutture di dati del nucleo

- Il nucleo ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di I/O
- Il sistema operativo UNIX, per mezzo del file system, permette l'accesso a diversi oggetti: i file degli utenti, i dispositivi, lo spazio d'indirizzi dei processi, e altri ancora
- Alcuni sistemi operativi applicano metodi orientati agli oggetti e un sistema basato sullo scambio di messaggi



walter.balzano@gmail.com

Strutture di dati del nucleo per la gestione dell'I/O nello UNIX



walter.balzano@gmail.com

Trasformazione delle richieste di I/O in operazioni dei dispositivi

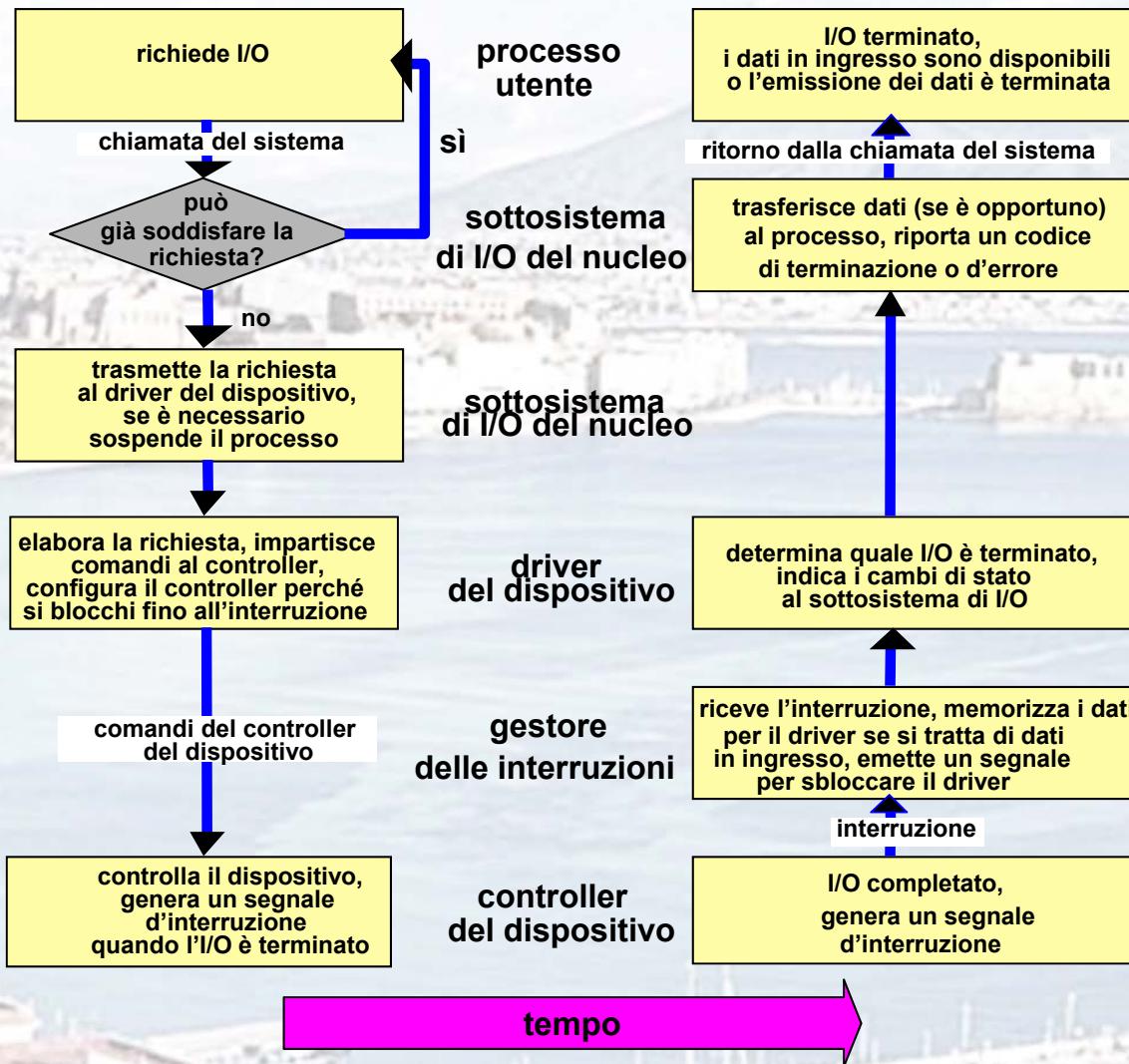
**Si consideri la lettura di un file
da un'unità a disco:**

- Si determina il dispositivo che detiene il file
- Si traduce il nome nella rappresentazione del dispositivo
- Si trasferiscono i dati dal disco al buffer
- Si rendono disponibili i dati al processo
- Si restituisce il controllo al processo



walter.balzano@gmail.com

Schema d'esecuzione di una richiesta di I/O



walter.balzano@gmail.com

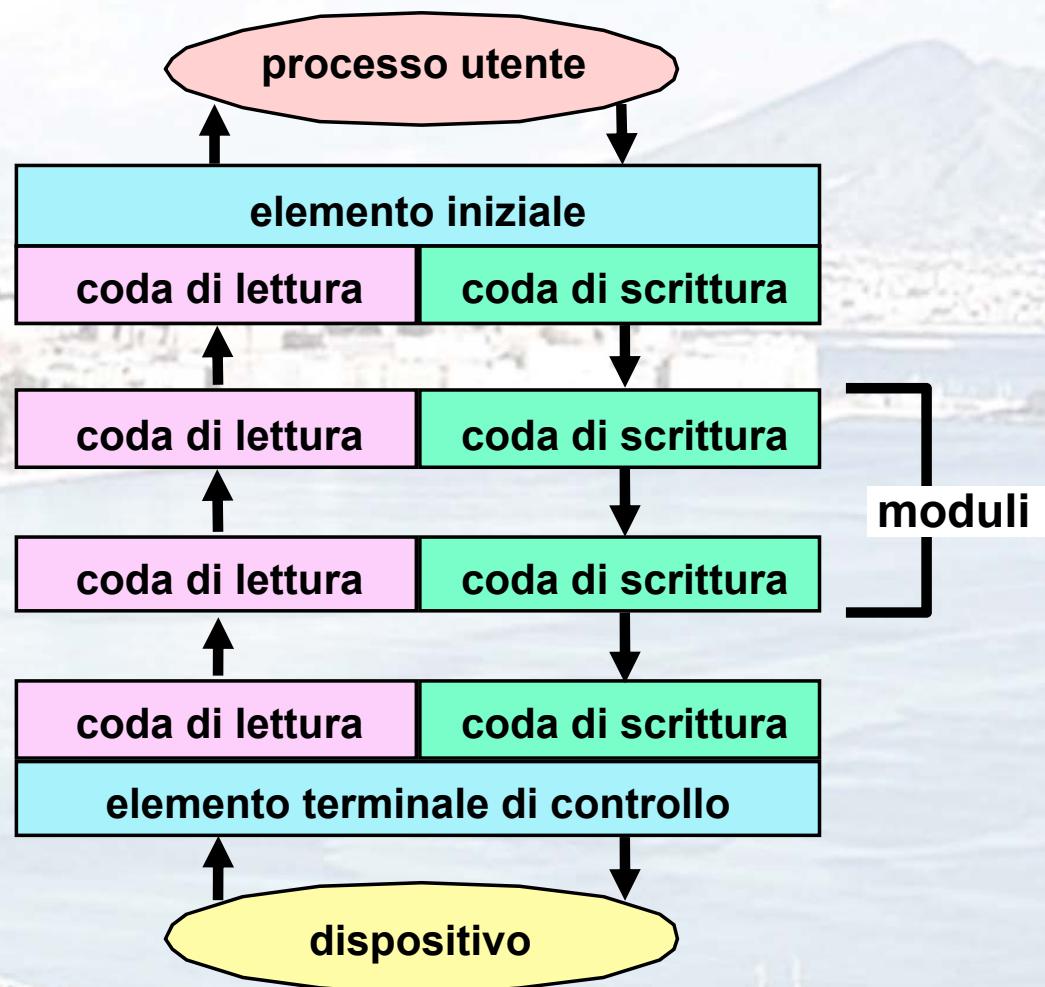
STREAMS

- **STREAMS**: connessione full-duplex tra un driver dispositivo e un processo utente.
- **STREAMS** consiste di:
 - un elemento iniziale d'interfaccia per il processo utente (*STREAM head*)
 - un elemento terminale che controlla il dispositivo (*driver end*)
 - un certo numero di moduli intermedi fra questi due estremi.
- Tutti questi elementi possiedono una coppia di code, una di lettura e una di scrittura.
- Per il trasferimento dei dati tra le due code, si usa uno schema a scambio di messaggi.



walter.balzano@gmail.com

Struttura di STREAMS



walter.balzano@gmail.com

Prestazioni

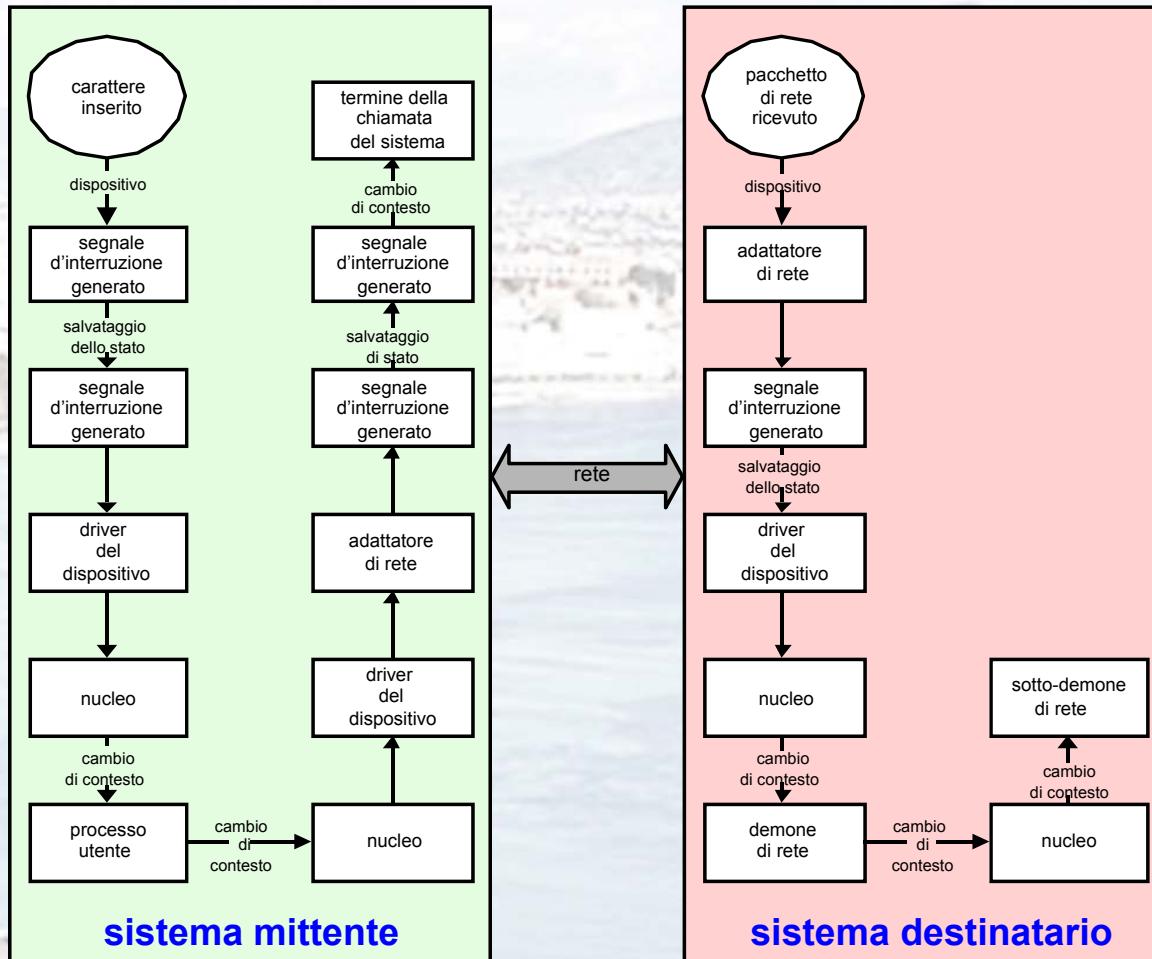
- L'I/O è uno tra i principali fattori che influiscono sulle prestazioni di un sistema:

- richiede un notevole impegno di CPU per l'esecuzione del codice del driver e per uno scheduling equo ed efficiente
- i risultanti cambi di contesto sfruttano fino in fondo la CPU e le sue memorie cache
- copia dei dati
- traffico di rete



walter.balzano@gmail.com

Comunicazione fra calcolatori



walter.balzano@gmail.com

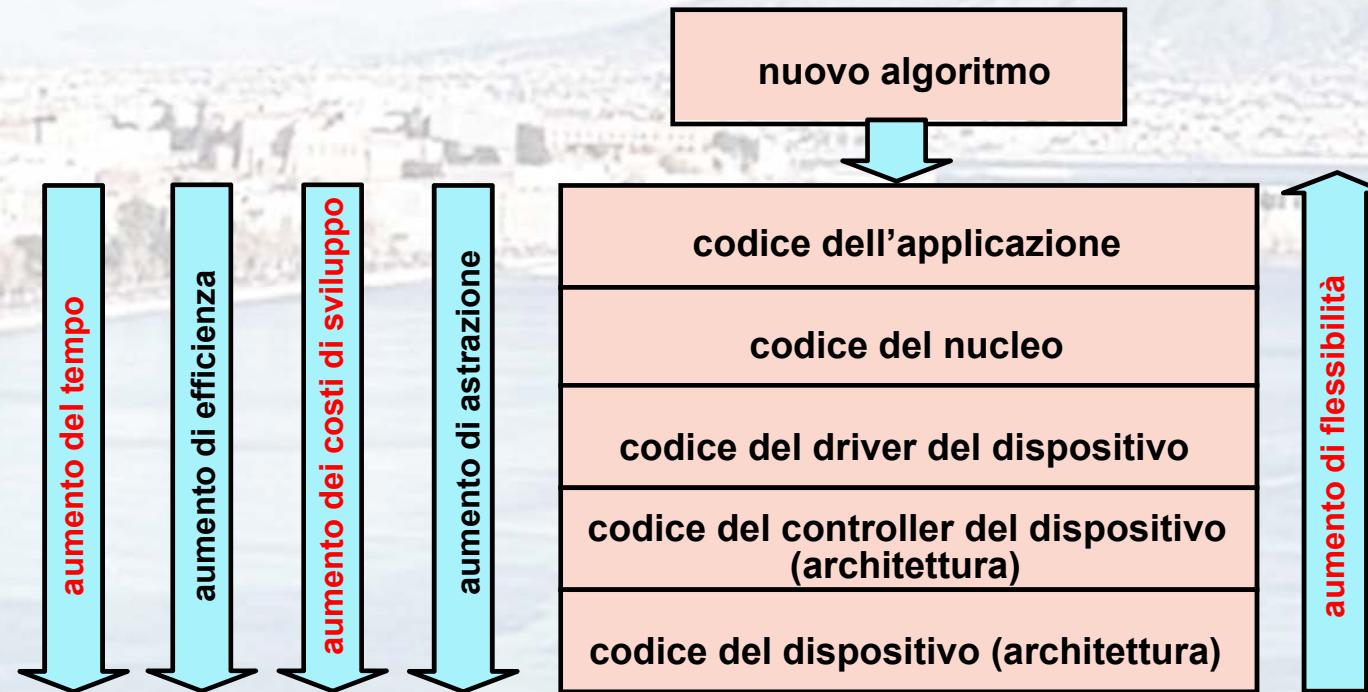
Migliorare le prestazioni

- Per migliorare l'efficienza dell'I/O si possono applicare diversi principi:
 - ridurre il numero dei cambi di contesto
 - ridurre il numero di copiature dei dati
 - ridurre la frequenza delle interruzioni tramite il trasferimento di grandi quantità di dati in un'unica soluzione, l'uso di controller intelligenti e mediante l'interrogazione ciclica
 - uso di controllori DMA intelligenti
 - equilibrare le prestazioni della CPU, del sottosistema per la gestione della memoria, del bus e dell'I/O



walter.balzano@gmail.com

Successione delle funzionalità dei servizi di I/O



walter.balzano@gmail.com

Capitolo 14: Memoria secondaria e terziaria

- Struttura dei dischi
- Scheduling del disco
- Gestione dell'unità a disco
- Strutture RAID
- Connessione dei dischi
- Memoria terziaria
- Prestazioni



walter.balzano@gmail.com

Struttura dei dischi

- I dischi sono considerati un grande vettore monodimensionale di **blocchi logici**, dove un **blocco logico è la minima unità di trasferimento**.
- Il vettore monodimensionale di blocchi logici corrisponde in modo sequenziale ai settori del disco:
 - Il settore 0 è il primo settore della prima traccia sul cilindro più esterno.
 - La corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via di cilindro in cilindro, **dall'esterno verso l'interno**.



walter.balzano@gmail.com

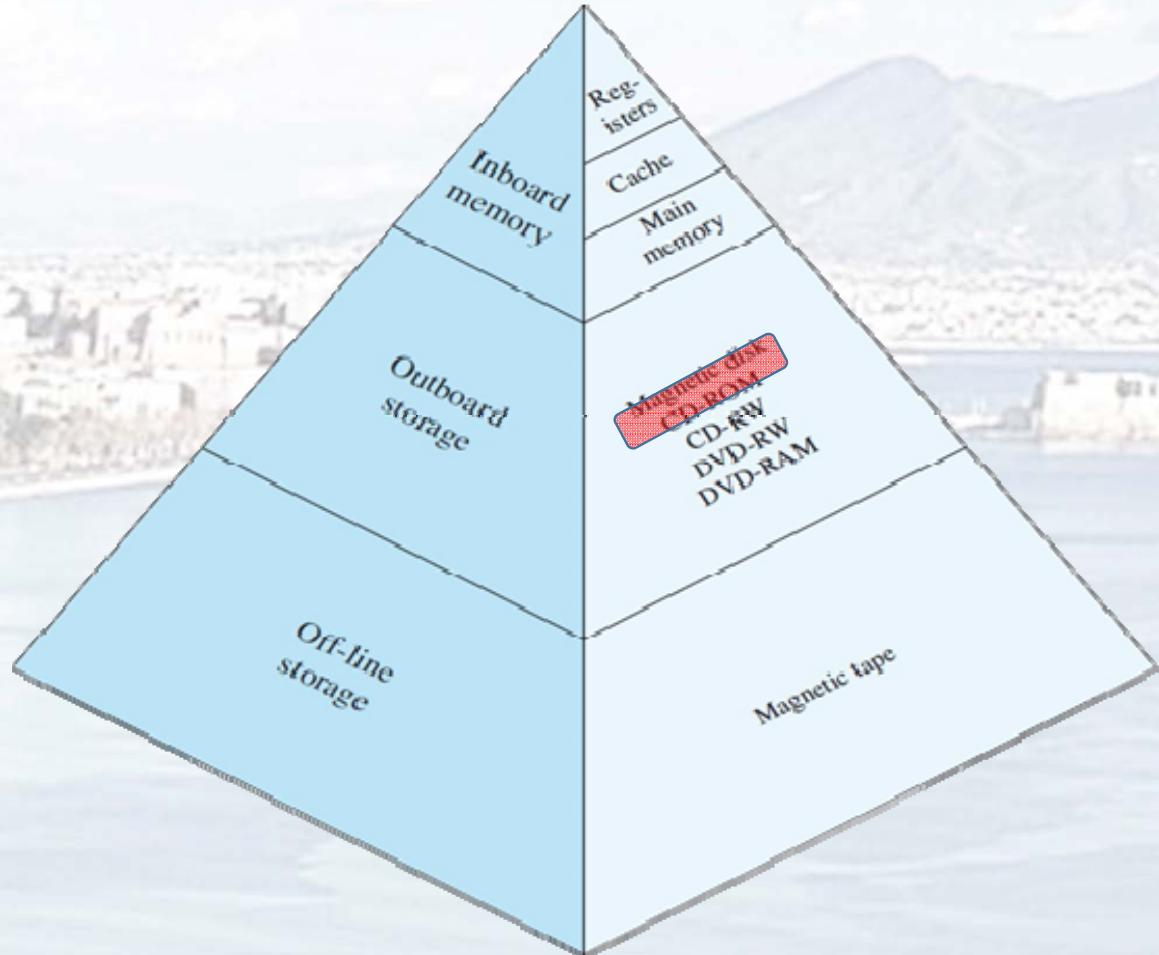
Caratteristiche hard disk

- Dispositivo meccanico
- assai più lento dei dispositivi elettronici.
- Esso è di tipo CAV (fino a 10.000 e 15.000 giri al minuti - RPM)
- I tempi di accesso alle informazioni su un Hard disk sono dipendono fortemente da:
 - Lo spostamento della testina in senso radiale fino a raggiungere la traccia desiderata (**seek time**)
 - L'attesa che il settore desiderato si trovi a passare sotto la testina; tale tempo dipende dalla velocità di rotazione del disco (**latency time**);
 - Il tempo di lettura vero e proprio dell'informazione.



walter.balzano@gmail.com

L'hard disk nella gerarchia



walter.balzano@gmail.com

Hard Disk tradizionale vs SSD (Solid State Disk)

HD



SSD



Basso

Alta

Alta

Bassa

COSTO

CAPACITA'

DELICATEZZA

VELOCITA'

Alto

Bassa

Bassa

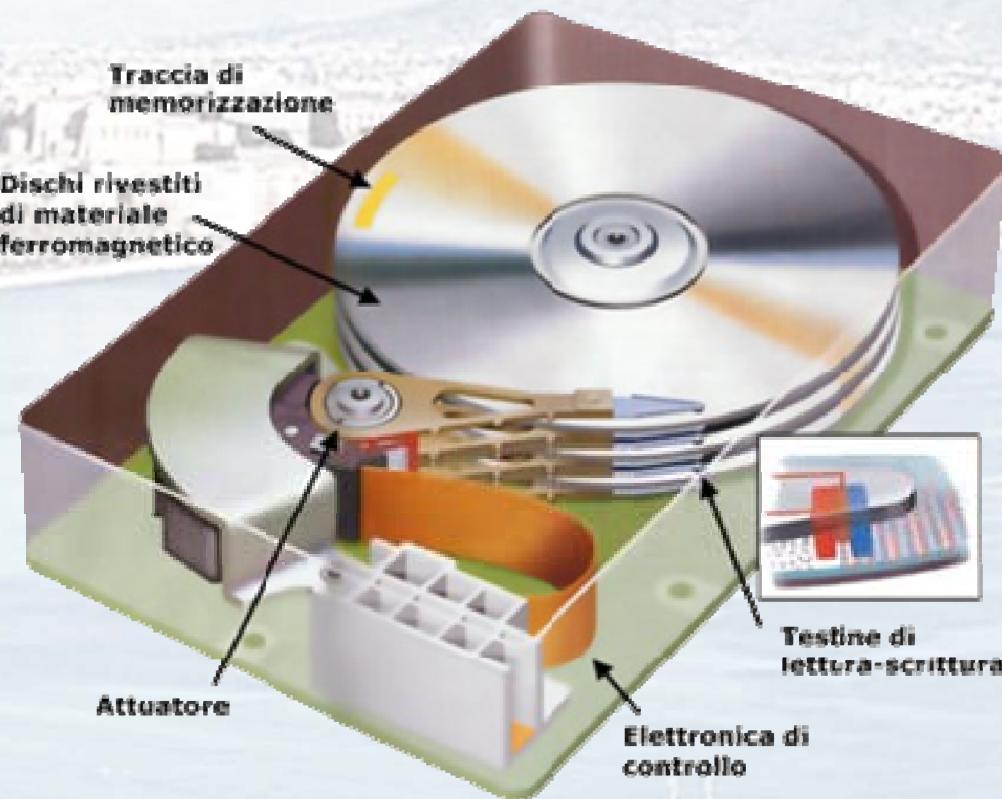
Alta



walter.balzano@gmail.com

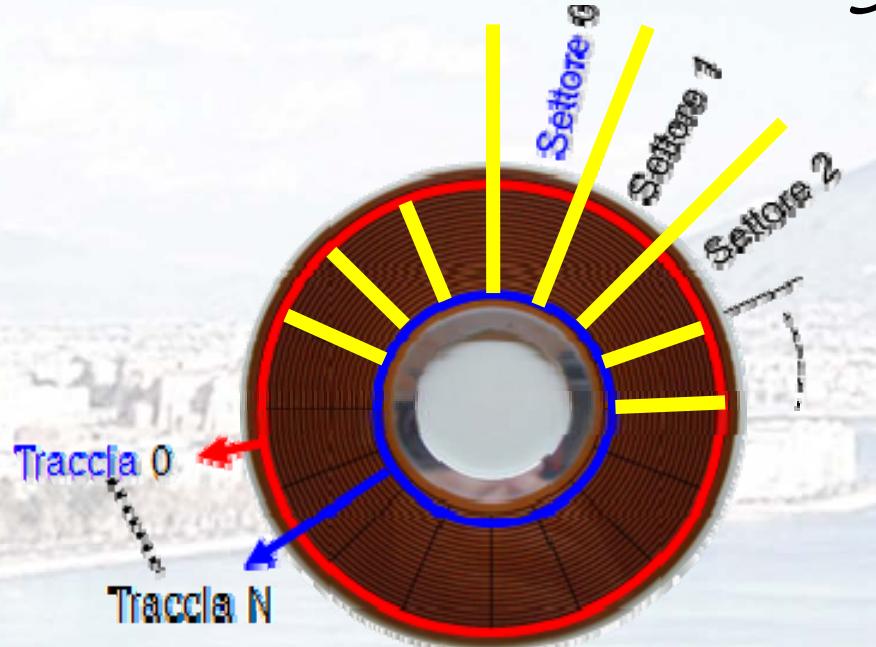
Struttura Hard Disk

Un **Hard Disk** è uno dei tipi di dispositivi di memoria di massa attualmente più utilizzati. Esso è un dispositivo di tipo magnetico che utilizza uno o più dischi magnetizzati. Si suppone che a breve sarà completamente soppiantato da analoga unità a stato solido: l'**SSD (Solid State Disk)**.

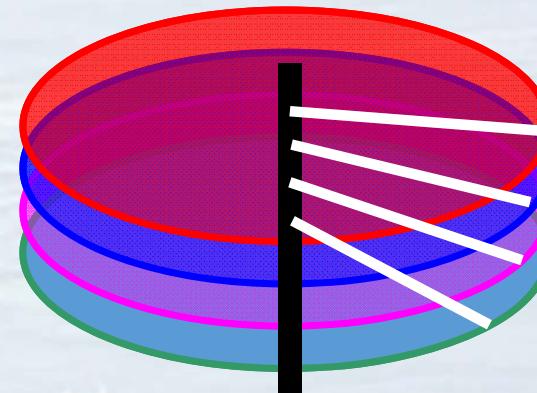


walter.balzano@gmail.com

Struttura Hard Disk

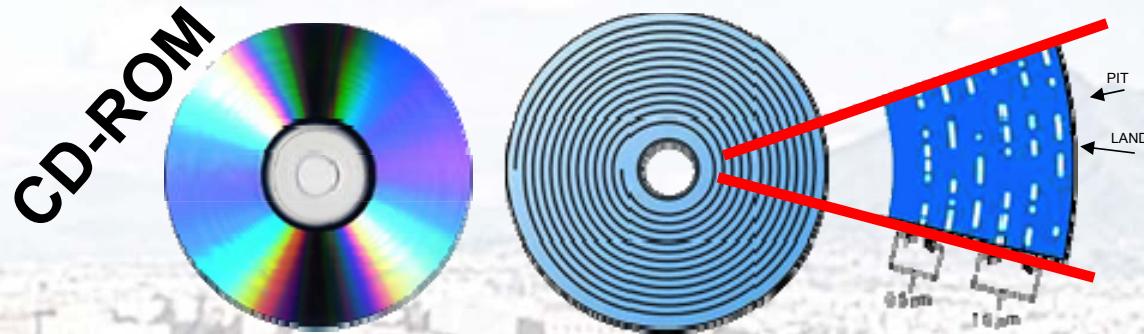


L'insieme delle tracce dei dischi che compongono lo stesso Hard Disk e che hanno lo stesso raggio formano quello che viene detto 'cilindro'



walter.balzano@gmail.com

Esempi di struttura: Floppy-disk, Cd-ROM....



- Il **Cd-ROM** è un disco su cui i dati sono scritti su una lunga spirale (circa 6 km per un CD) che inizia dall'interno e finisce all'esterno del disco. Nel funzionamento il disco ruota ad una velocità lineare costante (**CLV**: Costant Linear Velocity)
- Il **floppy disk** è costituito da una sostanza elettromagnetica distribuita in modo uniforme su tutta la superficie del disco. Nel funzionamento il disco ruota ad una velocità angolare costante (**CAV**: Costant Angular Velocity). Le **TRACCE** sono aree circolari, numerate da 0 (traccia esterna) ad N (traccia interna).



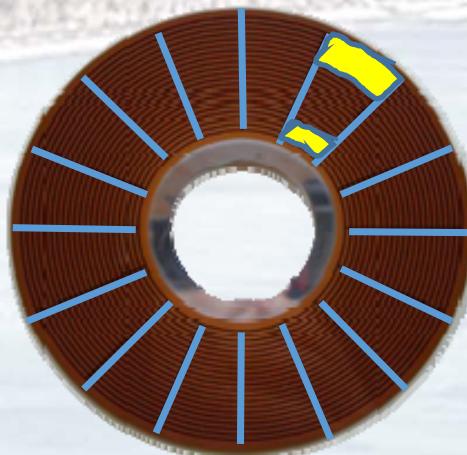
walter.balzano@gmail.com

Hard Disk Z-CAV

Su un disco a tracce concentriche, la lunghezza fisica della traccia aumenta all'aumentare della distanza dal centro. Quindi, a densità di dati costante, la capacità di una traccia aumenta assieme alla distanza dal centro. Per gestire tale particolarità si è progettata una nuova strategia nota col nome di **Zone Bit Recording (ZBR)** il cui obiettivo è quello di memorizzare più settori nelle tracce esterne.

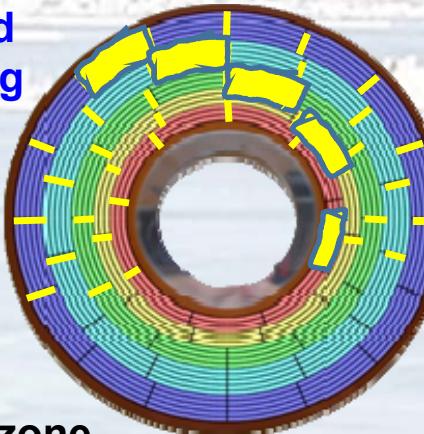
Poiché la tipologia usuale degli Hard Disk è di tipo **CAV (Constant Angular Velocity)** la strategia ZBR dovrà opportunamente gestire la velocità variabile della lettura e della scrittura: la velocità aumenta nelle tracce esterne.

“NON-ZBR” Zoned Bit Recording



Esempio: 20 tracce – 16 settori

“ZBR” Zoned Bit Recording



Esempio:
20 tracce – 5 zone
blu → 5 tracce – 16 settori
cyan → 5 tracce – 14 settori
verde → 4 tracce – 12 settori
giallo → 3 tracce – 11 settori
rosso → 3 tracce – 9 settori

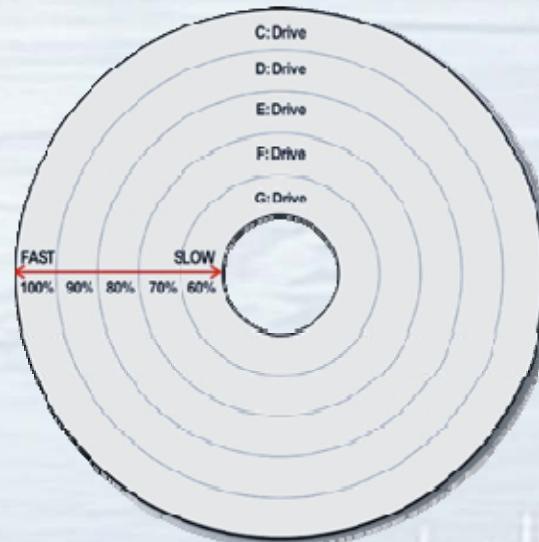
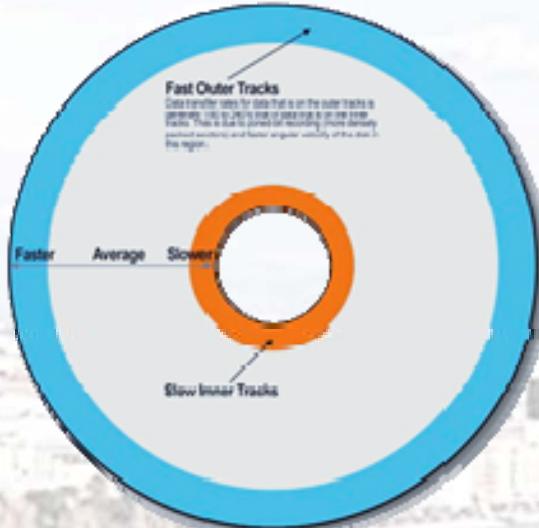


walter.balzano@gmail.com

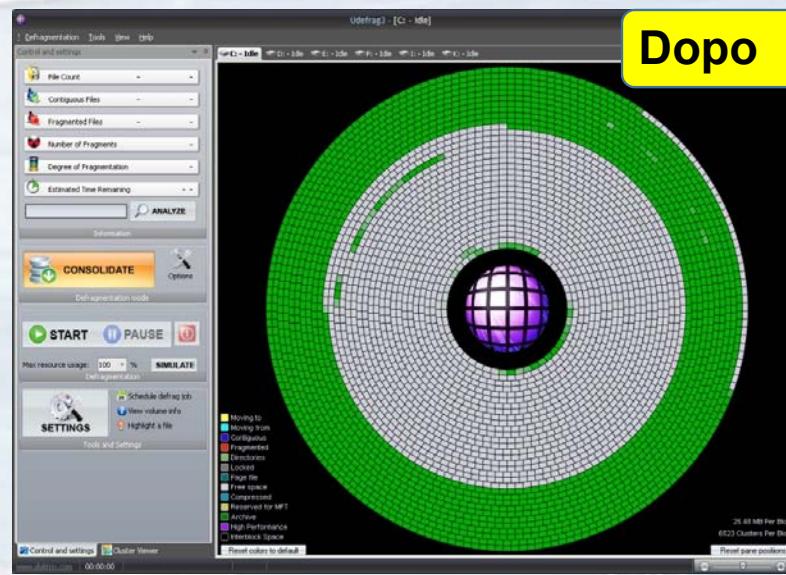
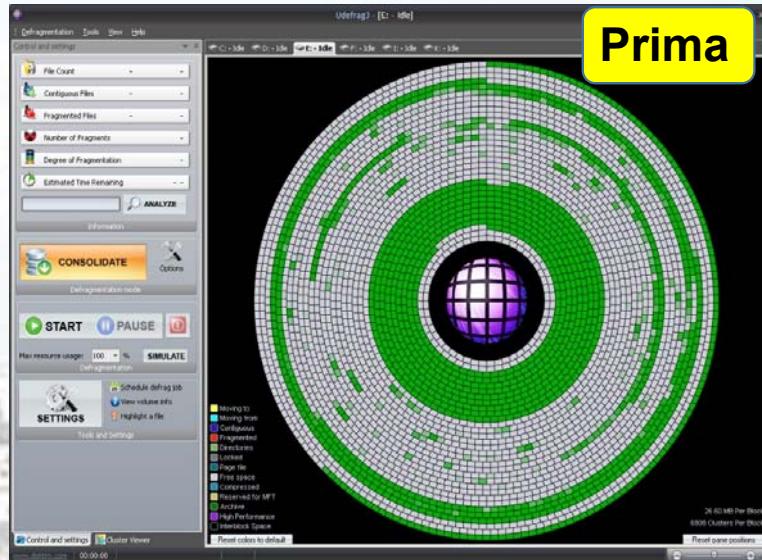
Partizionamento di un Hard Disk

Ci sono molti motivi che inducono a suddividere i dati di un Hard Disk in diverse aree logiche che siano differenziate anche fisicamente in modo da migliorarne la loro gestione:

- Grandi capacità di contenimento di un Hard Disk
- Necessità di separare al massimo aree di dati logicamente molti differenti tra loro
- **Necessità di evitare Seek-Time troppo elevati**



walter.balzano@gmail.com

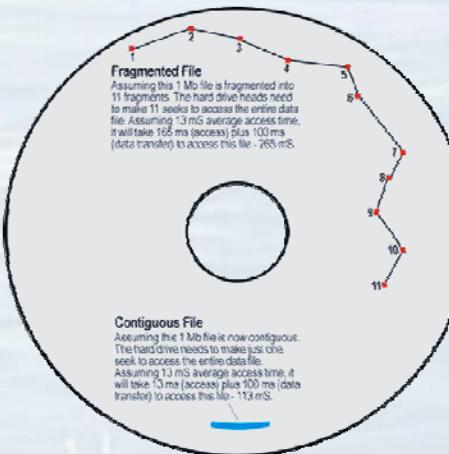
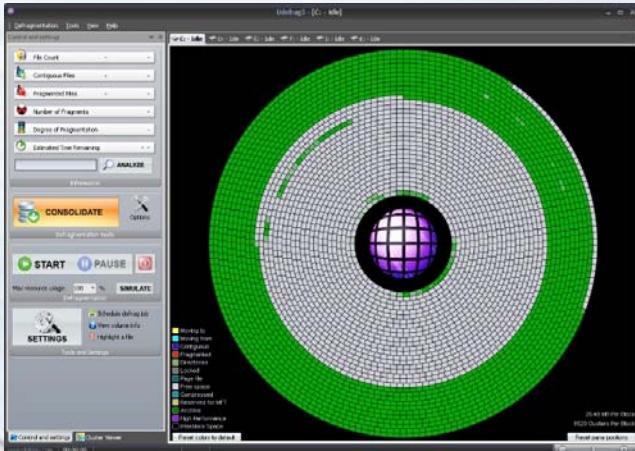
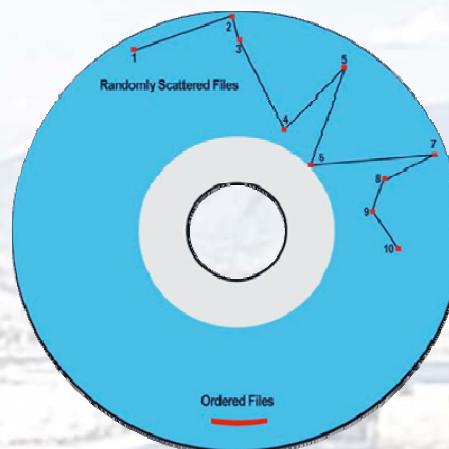
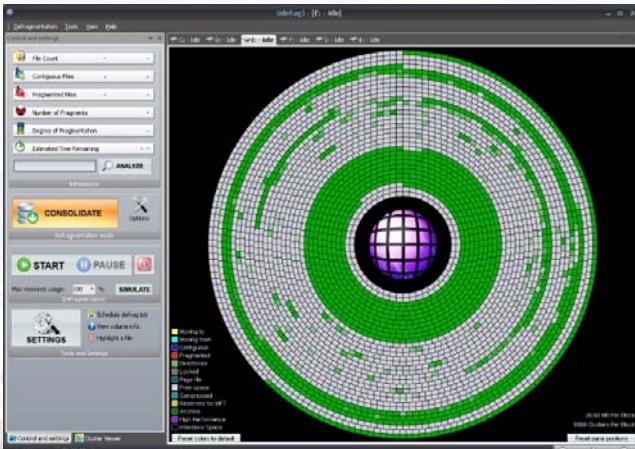


Frammentazione



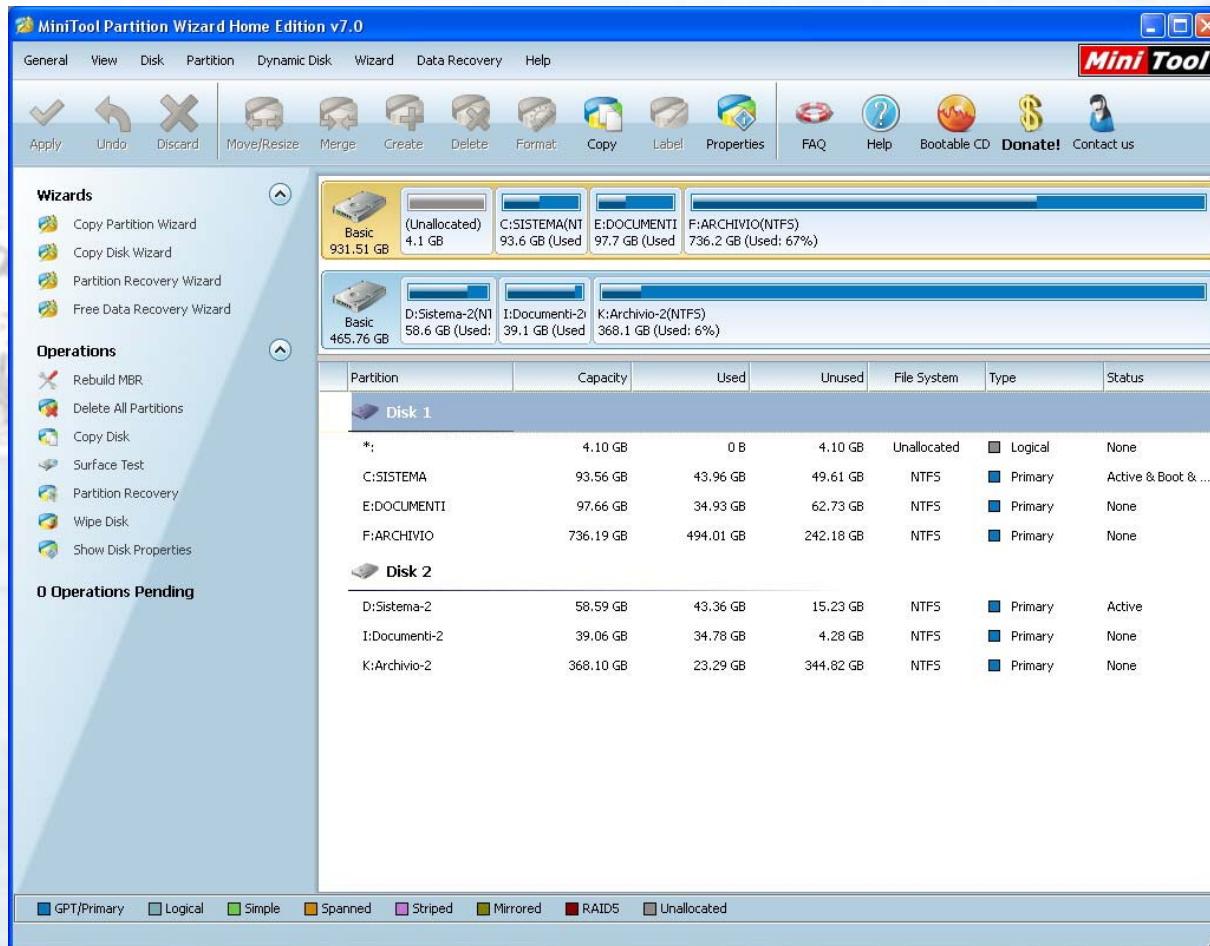
walter.balzano@gmail.com

Frammentazione



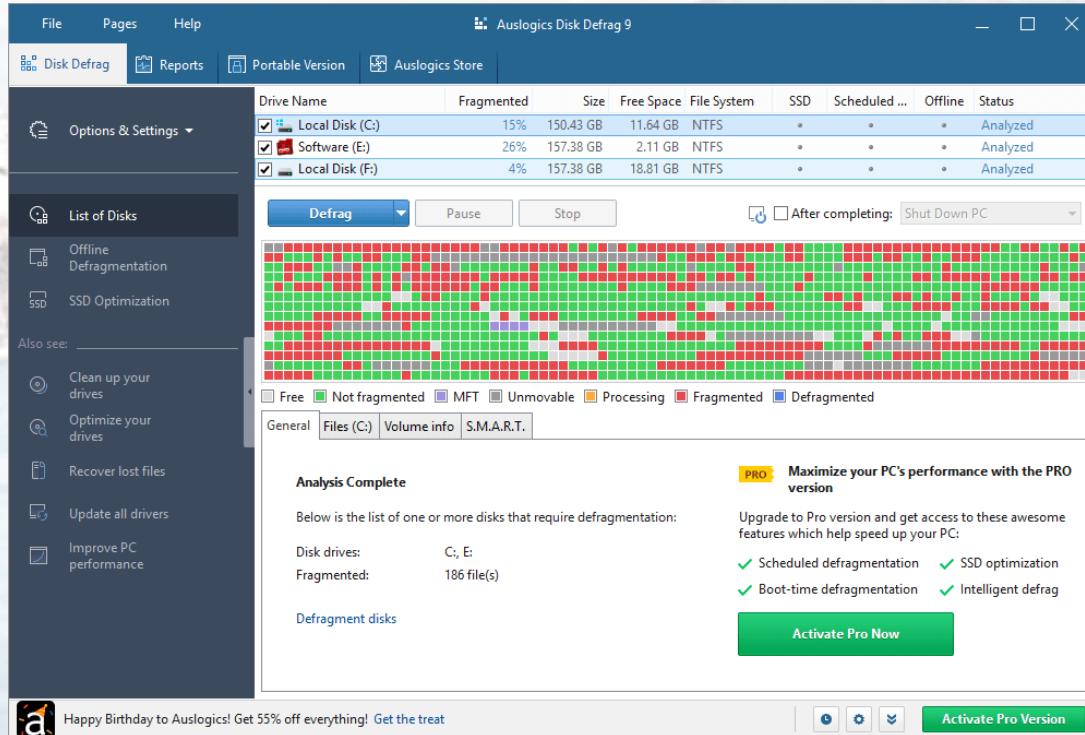
walter.balzano@gmail.com

Frammentazione



walter.balzano@gmail.com

De-Frammentazione



walter.balzano@gmail.com

Scheduling dell'Hard Disk (tradizionale)

Spesso il problema di scheduling dell'Hard Disk viene paragonato all'analogo problema di ottimizzazione dell'energia elettrica consumata dagli spostamenti di un ascensore (problema reale per edifici molto alti).

Esempio:

SE

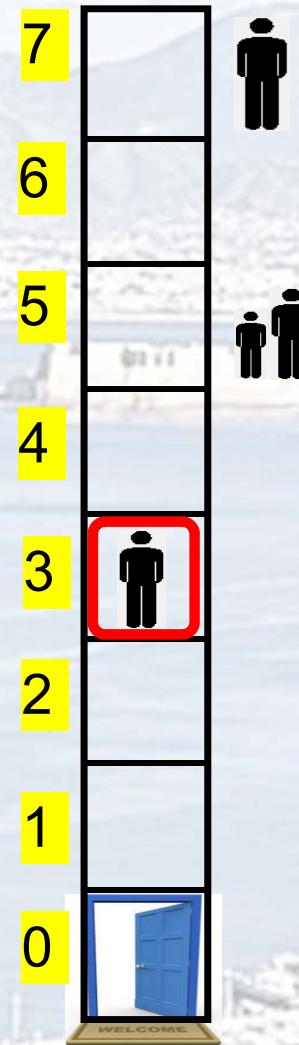
inizialmente l'ascensore si trova al 3° piano e, per esempio, le richieste di accesso agli altri piani sono 2,6,1,5,7

ALLORA

Quale sono i criteri che conviene adottare per consumare meno energia elettrica?

Alcuni possibili casi:

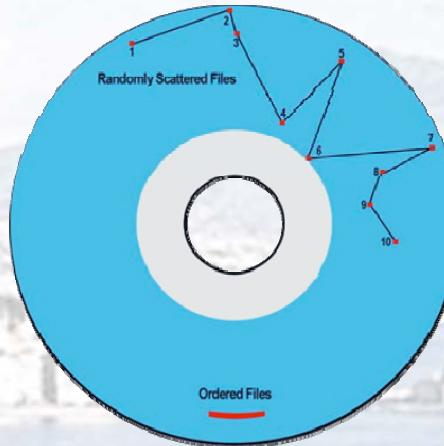
- Se l'ascensore percorre i piani nell'ordine 2,6,1,5,7 allora, partendo da 3, la distanza totale percorsa è pari a 16 piani.
- Se invece l'ascensore percorre i piani nell'ordine 2,1,5,6,7 allora, partendo da 3, la distanza totale percorsa è pari a 8 piani.



walter.balzano@gmail.com

Scheduling dell'Hard Disk (tradizionale)

Per accedere in diversi punti del disco il braccetto impiega un tempo proporzionale allo spostamento compiuto; la spezzata che congiunge tutti i punti determina la lunghezza totale



- Gli obiettivi principali della gestione (o scheduling) dell'Hard Disk consistono nel minimizzare i tempi di accesso al dispositivo stesso. Se occorre quindi accedere a dati che sono memorizzati sull'Hard Disk in diverse posizioni, occorre allora stabilire una metodologia ottimale che determini in quale ordine sia preferibile accedere a tali dati.
- L'ordine di accesso ai dati contenuti sull'Hard Disk ne determina il tempo totale di accesso.



walter.balzano@gmail.com

Scheduling del disco

- Il sistema operativo è responsabile di una gestione efficiente delle risorse fisiche: nel caso delle unità a disco, far fronte a questa responsabilità significa garantire tempi d'accesso contenuti e ampiezze di banda elevate.
- Il tempo d'accesso ha due componenti principali:
 - Il **tempo di ricerca** (*seek time*) è il tempo necessario affinché il braccio dell'unità a disco sposti le testine fino al cilindro contenente il settore desiderato.
 - La **latenza di rotazione** (*rotational latency*) è il tempo aggiuntivo necessario perché il disco ruoti finché il settore desiderato si trovi sotto la testina.
- Minimizzare il tempo d'accesso
- L'**ampiezza di banda del disco** (*disk bandwidth*) è il numero totale di byte trasferiti diviso il tempo totale intercorso fra la prima richiesta e il completamento dell'ultimo trasferimento.



walter.balzano@gmail.com

Scheduling dell'Hard Disk: alcune metodologie

Ecco alcune metodologie di scheduling:

• **FCFS** (First Come First Served) **Il primo arrivato è il primo servito**: le richieste vengono cioè soddisfatte nell'ordine in cui esse sono pervenute. Ha il vantaggio di essere un criterio semplice da realizzare ma non ottimizza la distanza da percorrere.

• **SSTF** (Shortest Seek Time First) **Prima i luoghi più vicini all'attuale luogo**: in altre parole seleziona la richiesta più vicina rispetto all'attuale posizione della testina. Presenta lo svantaggio di poter incorrere in situazioni di attesa indefinita (starvation) di alcune richieste.

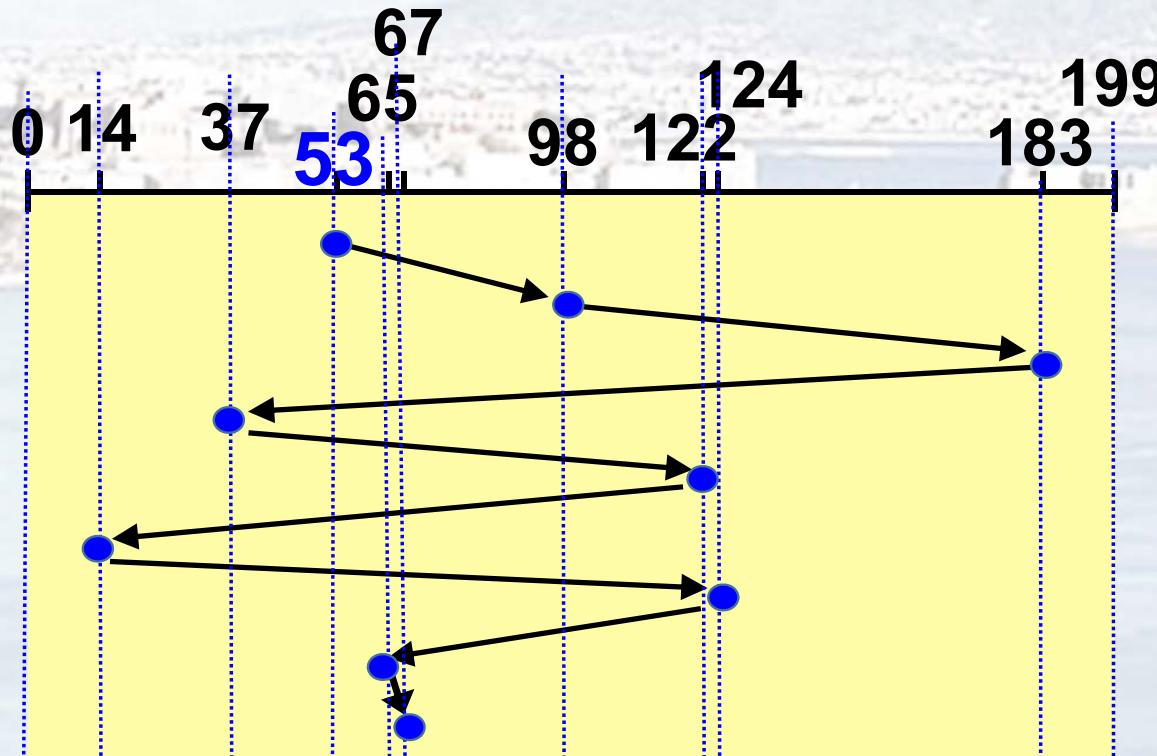
• **SCAN** **Il braccetto dell'unità a disco parte da un estremo del disco e si sposta nella sola direzione possibile**, servendo le richieste mentre attraversa i cilindri, fino a che non giunge all'altro estremo del disco: a questo punto, il braccio inverte la marcia, e la procedura continua.



walter.balzano@gmail.com

Scheduling **FCFS** - esempio

- Disco con 200 cilindri (da 0 a 199)
- coda delle richieste = 98, 183, 37, 122, 14, 124, 65, 67
- la testina è posizionata inizialmente sul cilindro **53**



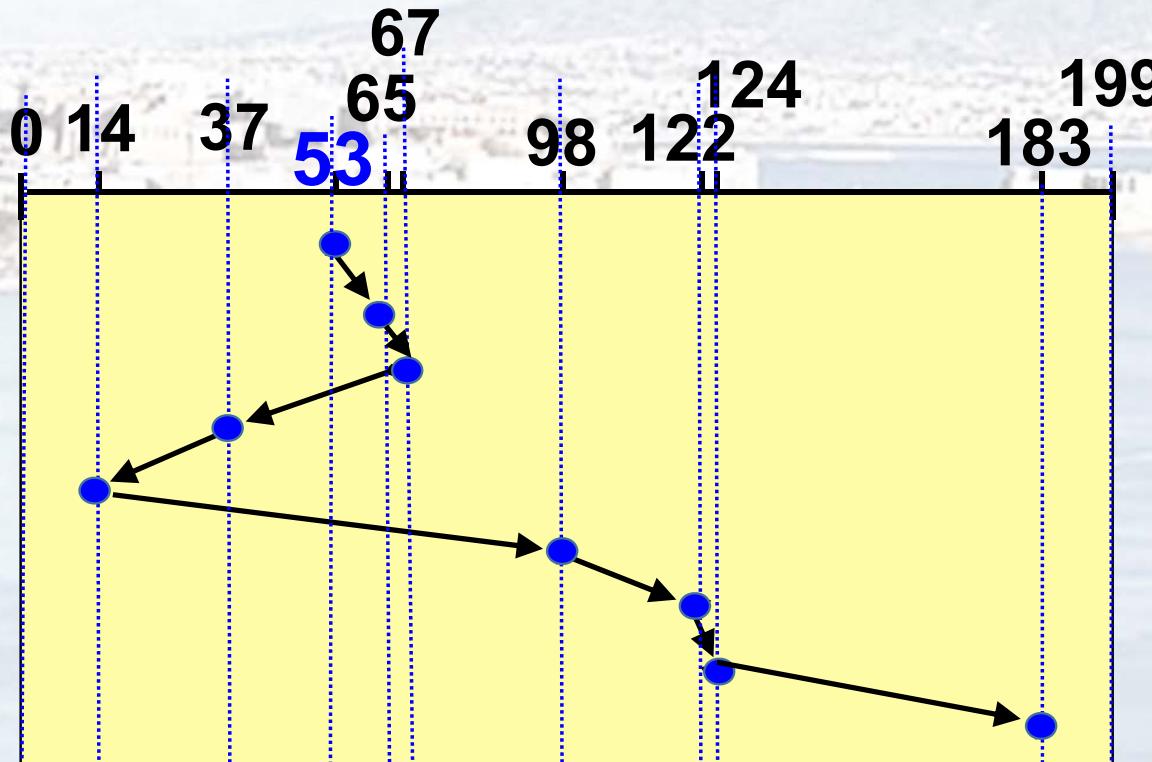
Totale = 640 cilindri



walter.balzano@gmail.com

Scheduling **SSTF** - esempio

- Disco con 200 cilindri (da 0 a 199)
- coda delle richieste = 98, 183, 37, 122, 14, 124, 65, 67
- la testina è posizionata inizialmente sul cilindro **53**



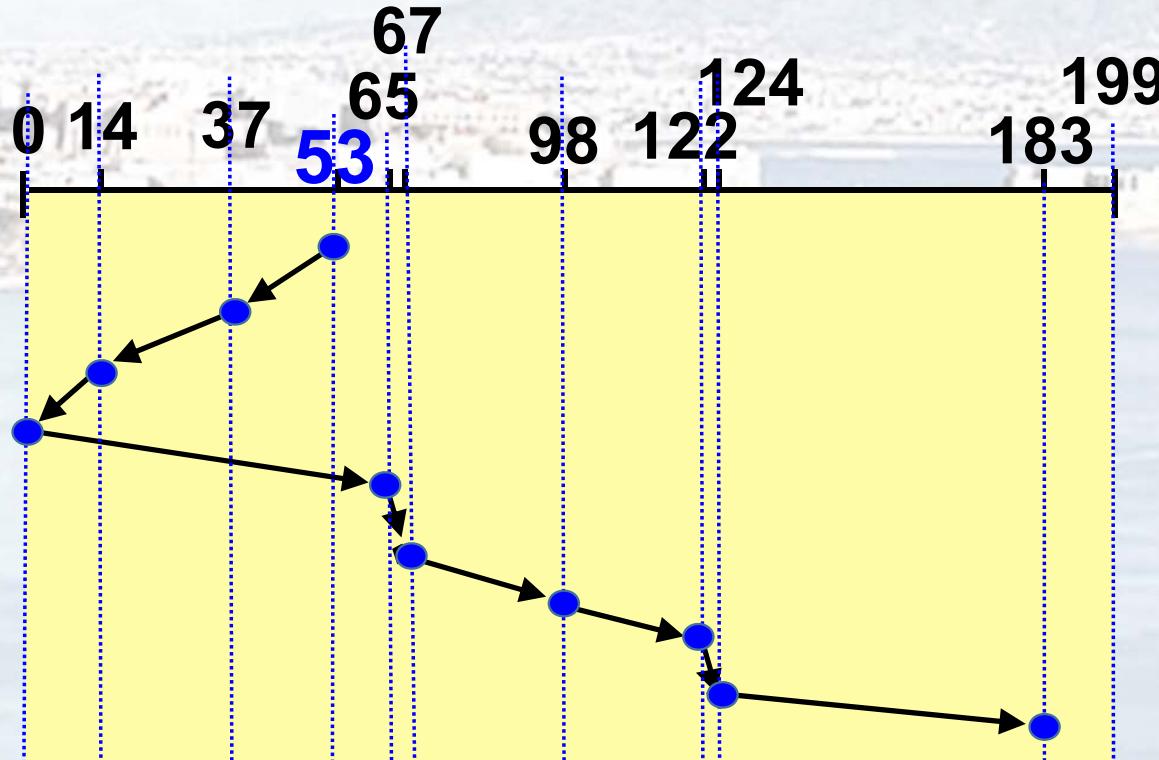
Totale = 236 cilindri



walter.balzano@gmail.com

Scheduling **SCAN** - esempio

- Disco con 200 cilindri (da 0 a 199)
- coda delle richieste = 98, 183, 37, 122, 14, 124, 65, 67
- la testina è posizionata inizialmente sul cilindro **53** (in viaggio verso 0)



Totale = 236 cilindri



walter.balzano@gmail.com

Scheduling per scansione circolare (C-SCAN)

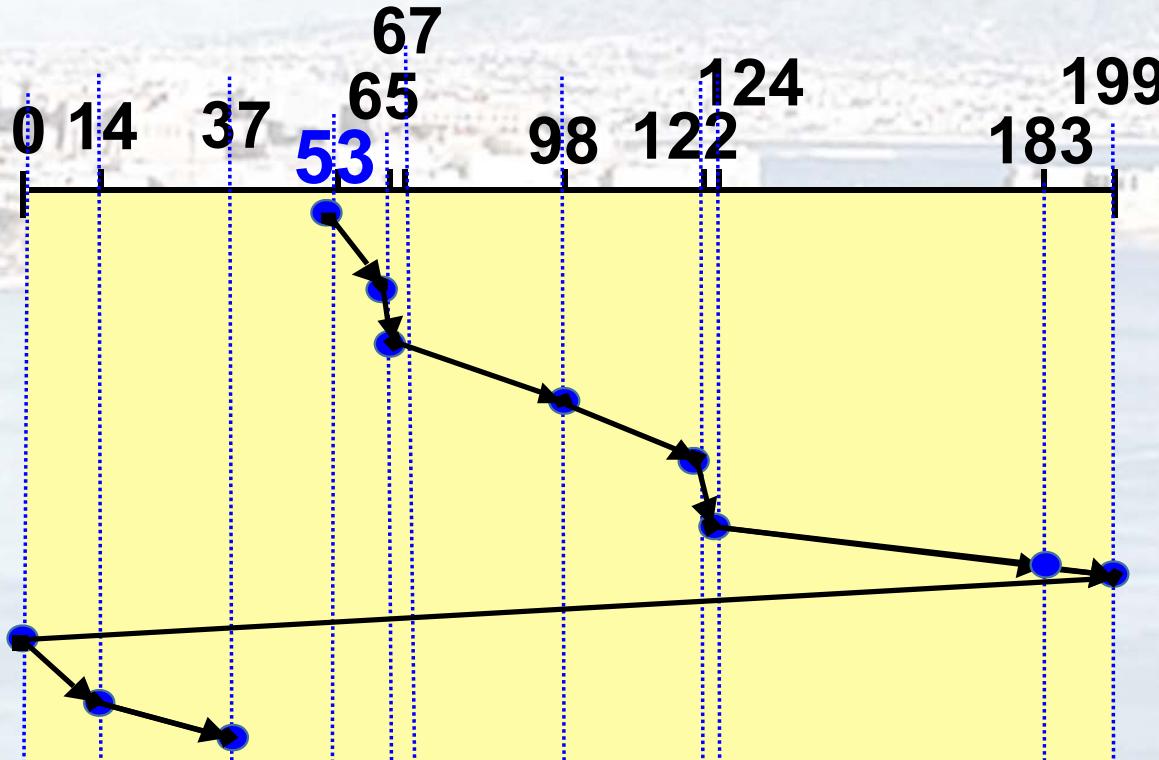
- L'algoritmo SCAN circolare (*circular SCAN*, C-SCAN) è una variante dello scheduling SCAN concepita per garantire un **tempo d'attesa meno variabile**.
- Anche l'algoritmo C-SCAN, come lo SCAN, sposta la testina da un estremo all'altro del disco, servendo le richieste lungo il percorso; tuttavia, quando la testina giunge all'altro estremo del disco, ritorna immediatamente all'inizio del disco stesso, senza servire richieste durante il viaggio di ritorno.
- L'algoritmo di scheduling C-SCAN, essenzialmente, tratta il disco come una lista circolare, cioè come se il primo e l'ultimo cilindro fossero adiacenti.



walter.balzano@gmail.com

Scheduling C-SCAN - esempio

- Disco con 200 cilindri (da 0 a 199)
- coda delle richieste = 98, 183, 37, 122, 14, 124, 65, 67
- la testina è posizionata inizialmente sul cilindro **53** (in viaggio verso 199)



Totale = 382 cilindri



walter.balzano@gmail.com

C-LOOK

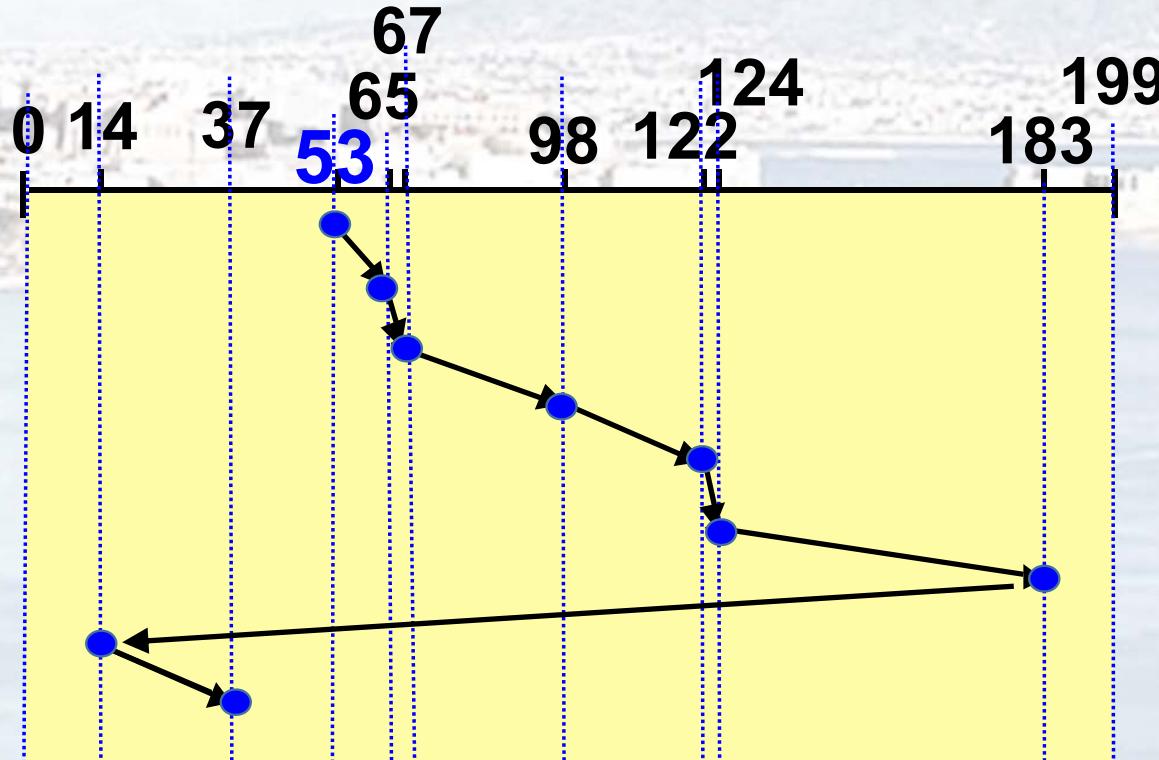
- Versione di C-SCAN
- Il braccio si sposta solo finché ci sono altre richieste da servire in ciascuna direzione, dopo di che cambia immediatamente direzione, senza giungere all'estremo del disco.



walter.balzano@gmail.com

Scheduling *C-LOOK* - esempio

- Disco con 200 cilindri (da 0 a 199)
- coda delle richieste = 98, 183, 37, 122, 14, 124, 65, 67
- la testina è posizionata inizialmente sul cilindro **53** (in viaggio verso 199)



Totale = 322 cilindri



walter.balzano@gmail.com

Scelta di un algoritmo di scheduling

- **SSTF è molto comune e naturalmente attraente.**
- **SCAN e C-SCAN offrono migliori prestazioni in sistemi che sfruttano molto le unità a disco.**
- **Le prestazioni dipendono in larga misura dal numero e dal tipo di richieste.**
- **Le richieste di I/O per l'unità a disco possono essere notevolmente influenzate dal metodo adottato per l'assegnazione dei file.**
- **L'algoritmo di scheduling del disco dovrebbe costituire un modulo a sé tante del sistema operativo così da poter essere sostituito da un altro algoritmo qualora ciò fosse necessario.**
- **Sia SSTF sia LOOK costituiscono un ragionevole algoritmo di partenza.**



walter.balzano@gmail.com

Gestione dell'unità a disco

- Prima che un disco magnetico possa memorizzare dati, deve essere diviso in settori che possano essere letti o scritti dal controllore.
- Per usare un disco come contenitore di informazioni, il sistema operativo deve registrare le proprie strutture dati all'interno del disco. Ciò avviene in due passi.
 - Suddividere il disco in uno o più gruppi (*partizioni*).
 - Creare un file system (*formattazione logica*).
- Il blocco d'avviamento (*boot block*) inizializza il sistema.
 - Un piccolo caricatore d'avviamento (*bootstrap loader*) è memorizzato nella ROM.
- La formattazione fisica mette anche da parte dei settori di riserva non visibili al sistema operativo: si può istruire il controller affinché sostituisca da un punto di vista logico un settore difettoso con uno dei settori di riserva non utilizzati. Questa strategia è nota come accantonamento di settori (*sector sparing*).



walter.balzano@gmail.com

Configurazione del disco nell'MS-DOS

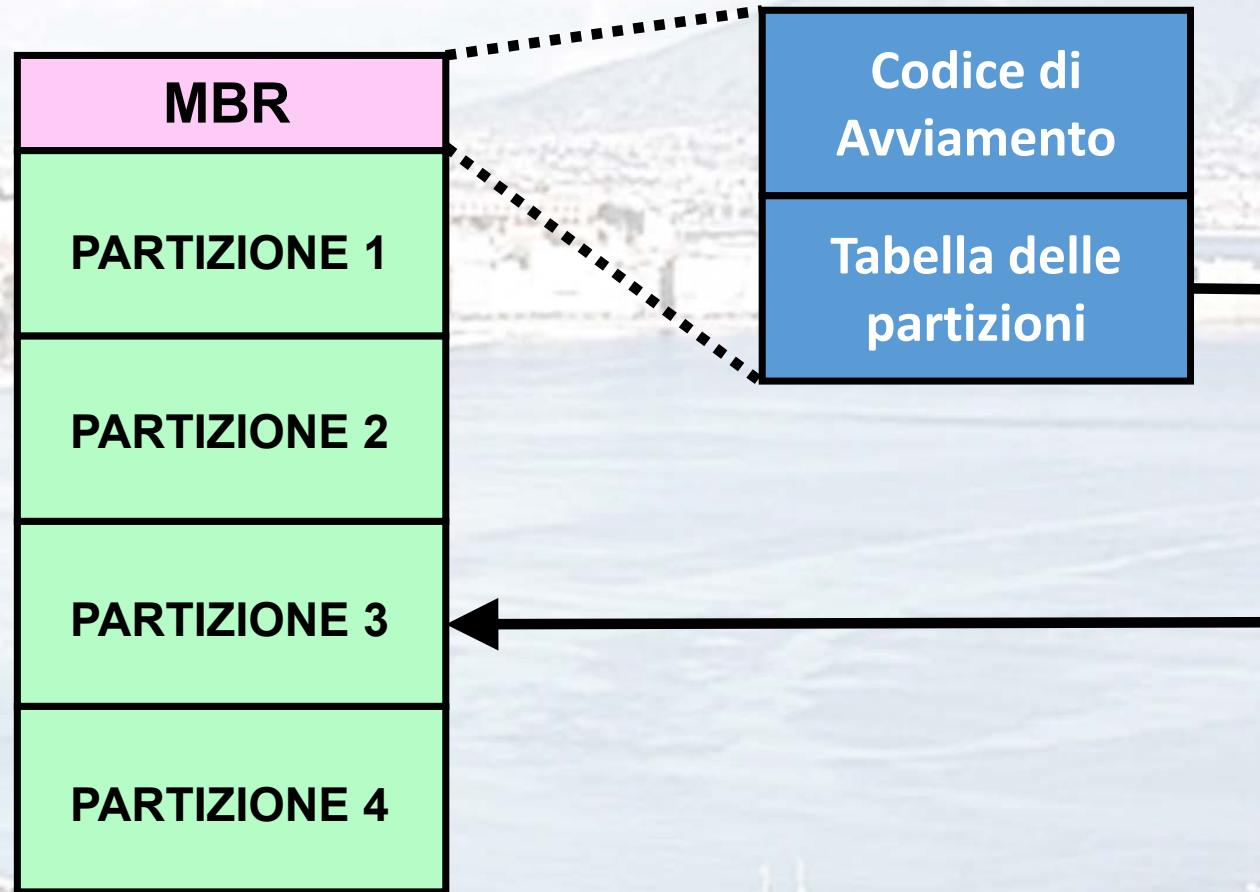
Settore 0

Settore 1



walter.balzano@gmail.com

Configurazione del disco in Windows



walter.balzano@gmail.com

Gestione dell'area di avvicendamento

- **Area d'avvicendamento (swap-space): la memoria virtuale usa lo spazio dei dischi come estensione della memoria centrale.**
- **L'area d'avvicendamento può essere ricavata all'interno del normale file system o, più comunemente, può trovarsi in una partizione separata del disco.**
- **Gestione dell'area di avvicendamento**
 - Nella versione 4.3BSD si assegna l'area di avvicendamento a un processo quando questo è avviato; si riserva spazio sufficiente per il segmento di testo dove è contenuto il programma e per il segmento dei dati.
 - Due mappe d'avvicendamento per ogni processo servono al nucleo per tenere traccia dell'area d'avvicendamento correntemente impiegata.



walter.balzano@gmail.com

Configurazioni RAID

(Redundant Array of Independent/inexpensive Disk)

- In sintesi, un sistema RAID è una combinazione fra più Hard Disk realizzata con lo scopo di migliorare l'affidabilità del sistema, oppure le prestazioni o entrambi gli aspetti. (per certi aspetti è paragonabile alla tecnica di accesso 'combinato' della RAM come quella Dual-Channel) Le tecniche RAID possono essere realizzate sia in Hardware che in Software: la soluzione hardware è quella che risulta essere più performante.
- Gran parte della strategia RAID è basata sulla capacità di poter leggere/scrivere in parallelo un insieme di dischi collegati al nostro sistema.
- È possibile realizzare differenti tipologie di collegamento degli Hard Disk: ogni configurazione di collegamento è definita come "LIVELLO RAID x" (con $x=0,1,2,\dots$) ed ogni specifica configurazione offre vantaggi/svantaggi.



walter.balzano@gmail.com

Configurazioni RAID

(Redundant Array of Independent/inexpensive Disk)

I livelli più usati (singolarmente o combinazioni di essi) sono:

RAID Level 0. È la più semplice: permette la distribuzione automatica dei dati su più dischi ma poiché i dati non sono replicati non garantisce la tolleranza ai guasti in caso di rottura di un disco.

RAID Level 1. È la tecnica più diffusa; i dati vengono replicati (**mirroring**) e nel caso di rottura di un disco si passa automaticamente ad un altro disco senza alcuna perdita di dati.

RAID Level 3. Simile alla tecnica di livello 0 ma dedica un hard disk al **recupero automatico d'errore** mediante il controllo di parità.

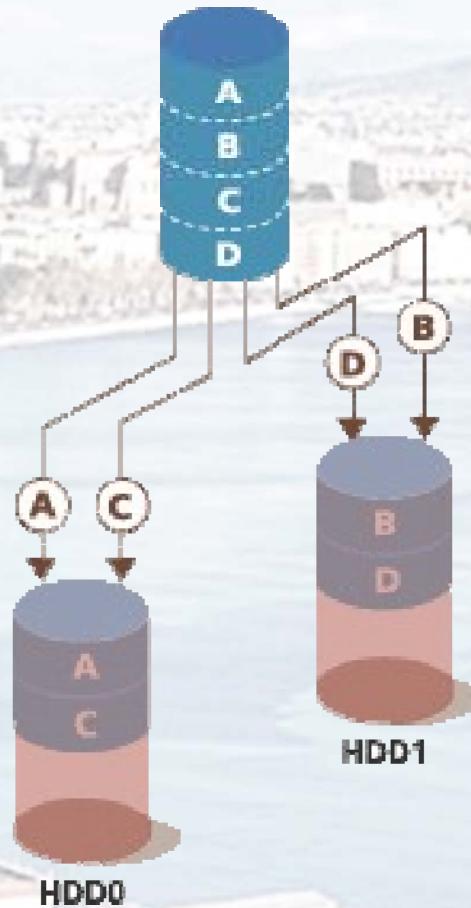
RAID Level 5. Si usa nel caso di sistemi che richiedono una garanzia di elevata protezione dei dati e alte prestazioni. Usa **data striping completo** e un disco per la correzione d'errore.



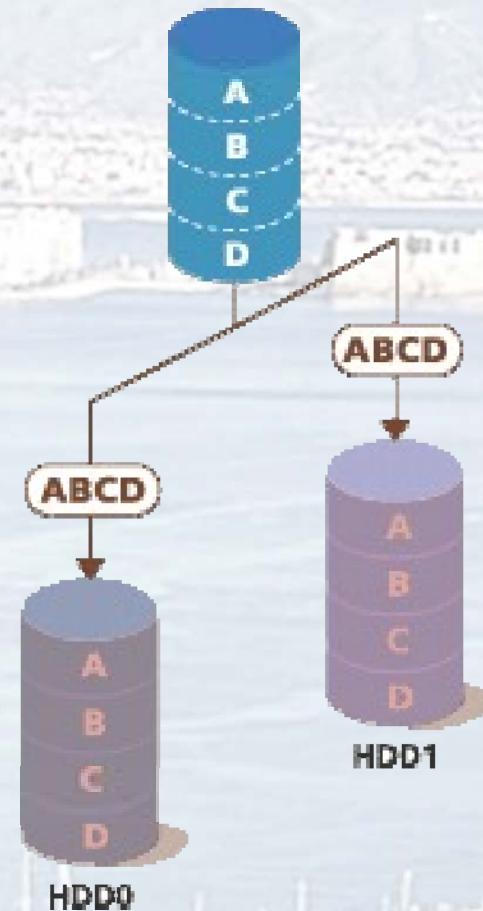
walter.balzano@gmail.com

Configurazioni RAID: esempi

Raid 0



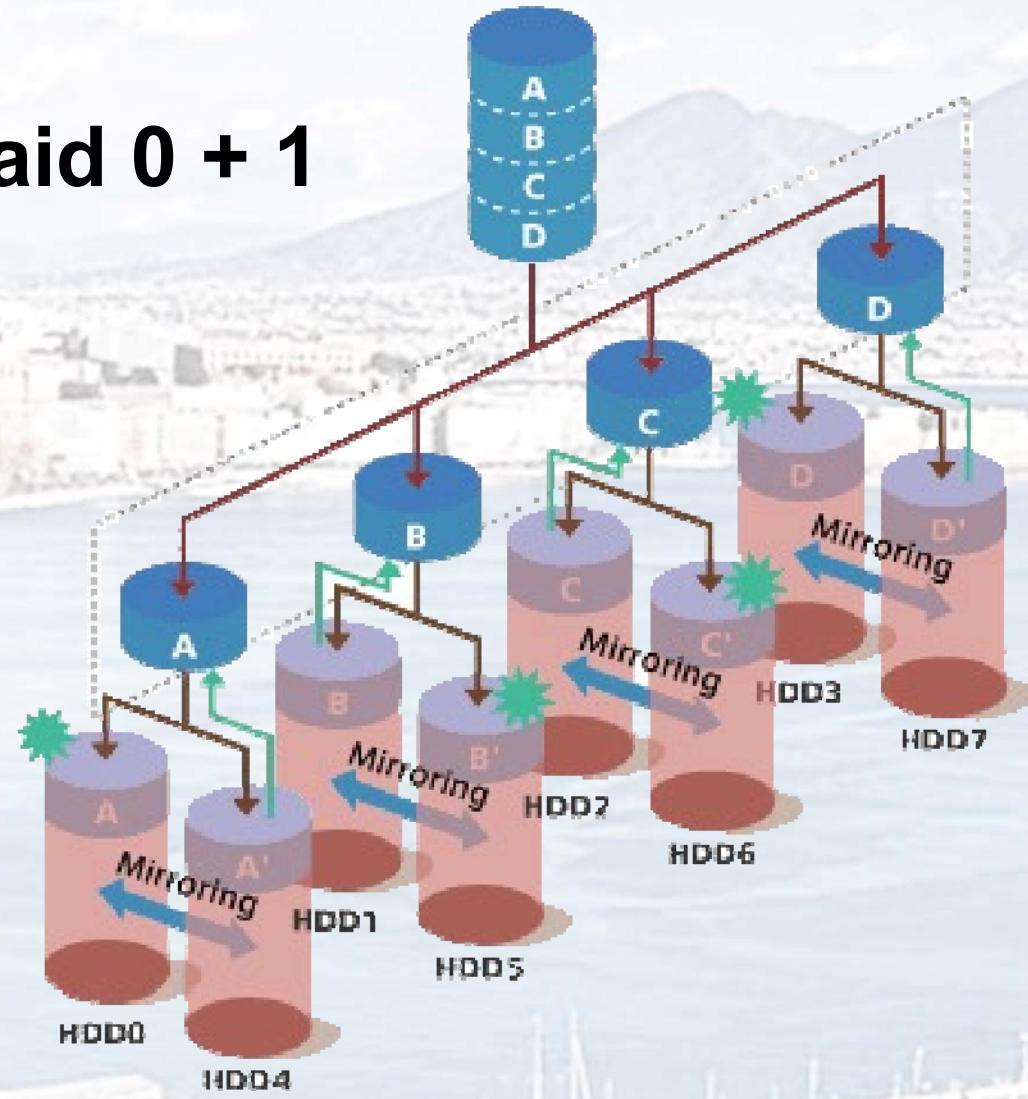
Raid 1



walter.balzano@gmail.com

Configurazioni RAID: esempi

Raid 0 + 1



walter.balzano@gmail.com

Livelli RAID

Livello	
0	Sezionamento senza ridondanza
1	Copiatura speculare
2	Codici per la correzione degli errori
3	Bit di Parità intercalati
4	Blocchi di Parità intercalati
5	Blocchi intercalati a parità distribuita
6	Ridondanza P + Q



walter.balzano@gmail.com

Connessione dei dischi

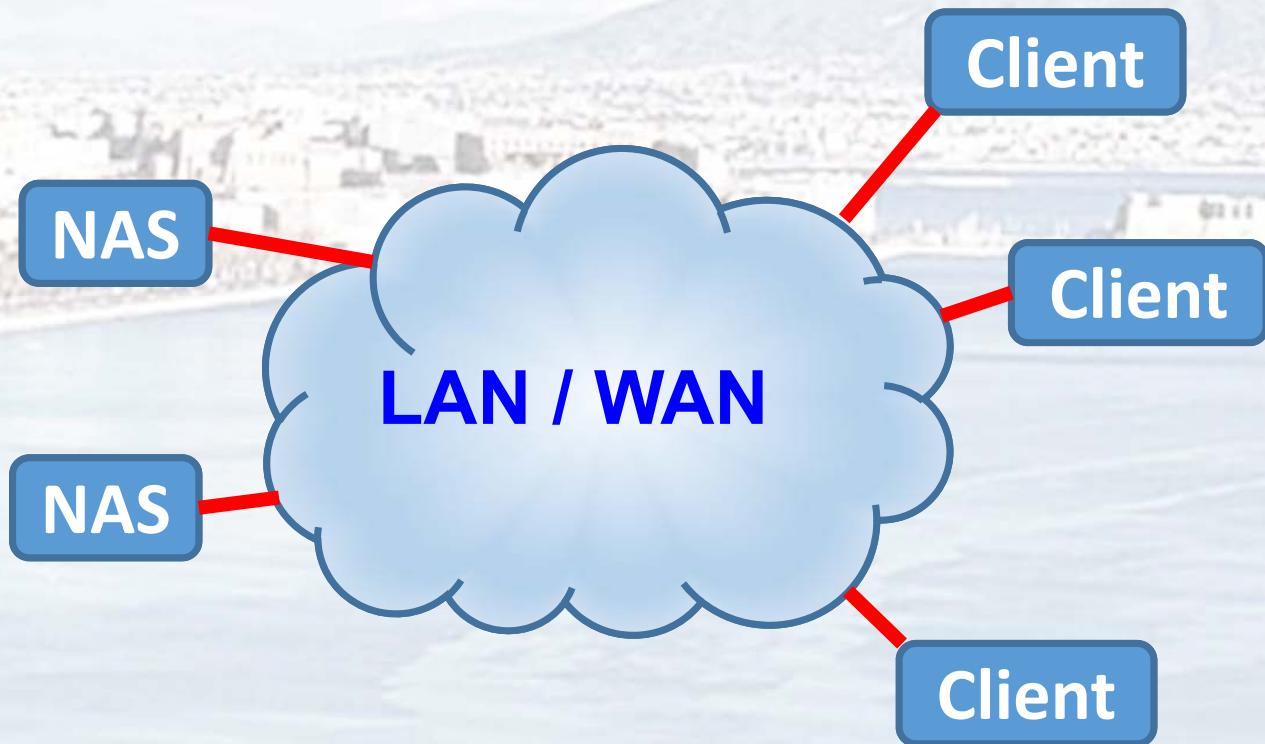
I calcolatori accedono alla memoria secondaria in due modi:

1. tramite le porte di I/O
(memoria secondaria connessa alla macchina, *host-attached storage*)
2. per mezzo di un file system distribuito
(memoria secondaria connessa alla rete, *network attached storage*)



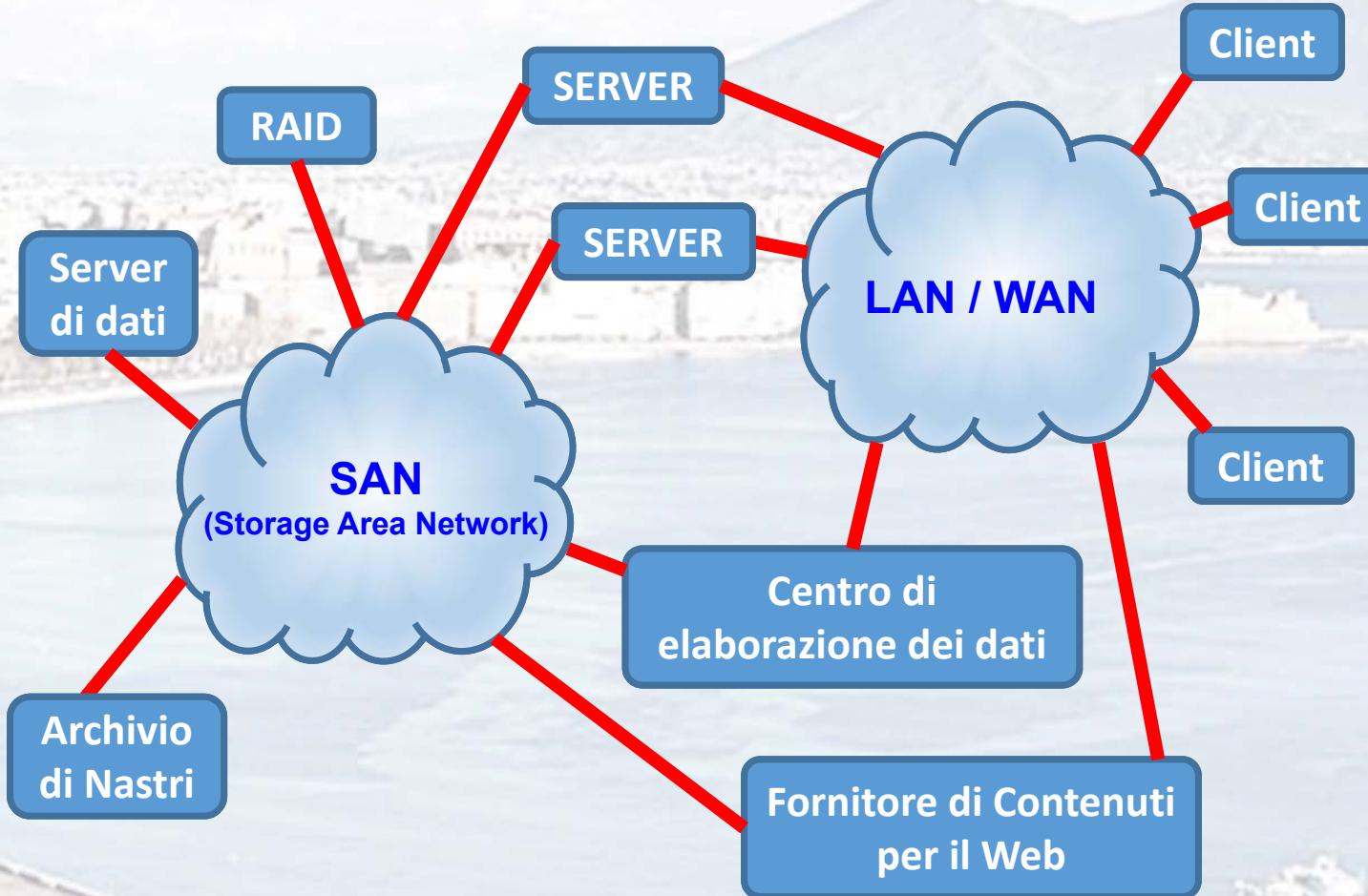
walter.balzano@gmail.com

Memoria secondaria connessa alla rete



walter.balzano@gmail.com

Rete di memoria secondaria



walter.balzano@gmail.com

Dispositivi per la memorizzazione terziaria

- La caratteristica peculiare delle memoria terziaria è il suo basso costo.
- Generalmente, la memoria terziaria è costituita da mezzi rimovibili.
- Alcuni esempi di mezzi rimovibili sono i CD-ROM ma sul mercato sono disponibili anche molti altri prodotti.



walter.balzano@gmail.com

Dischi rimovibili

- I dischi magneto-ottici registrano i dati su un disco rigido ricoperto da materiale magnetico.
 - La testina emette un raggio laser verso la superficie del disco, puntandolo sull'area dove si vuole scrivere un bit.
 - L'unità a disco legge i bit sfruttando una proprietà della luce laser detta effetto Kerr.
 - Il materiale magnetico è protetto da uno spesso strato di plastica o vetro; di conseguenza il disco è più resistente a eventuali collisioni della testina.
- I dischi ottici non sfruttano il magnetismo ma usano materiali speciali che la luce laser può alterare in modo da creare punti relativamente chiari o scuri.



walter.balzano@gmail.com

Dischi WORM

- I dati memorizzati sui dischi a lettura e scrittura possono essere aggiornati e modificati.
- I dischi WORM (*Write Once, Read Many*) possono essere scritti solo una volta.
- Una sottile pellicola di alluminio viene inserita tra due piatti di plastica o vetro.
- Per scrivere un bit, l'unità usa un raggio laser per praticare un piccolo foro nell'alluminio; poiché questo processo non è reversibile, le informazioni possono essere lette ma non alterate.
- I dischi WORM sono considerati durevoli e affidabili.
- I dischi a sola lettura come i CD-ROM e i DVD sono commercializzati con un contenuto pre-registrato.



walter.balzano@gmail.com

Nastri

- Rispetto a un disco, un nastro è meno costoso e contiene più dati, ma l'accesso è molto più lento.
- I nastri magnetici sono un mezzo conveniente qualora non si richiedano rapidi accessi diretti, e quindi per copie di riserva, anche nei grandi centri di calcolo.
- Grandi stazioni di registrazione a nastri usano meccanismi automatici per spostare i nastri dalle unità ad appositi contenitori in un archivio di nastri.
- Un file non immediatamente necessario può essere archiviato su nastro a un costo per gigabyte che può essere inferiore; quando il file si renderà necessario, il calcolatore potrà installarlo nuovamente nel disco.



walter.balzano@gmail.com

Compiti del sistema operativo

- Due tra gli obiettivi primari di un sistema operativo sono la gestione dei dispositivi fisici e la presentazione di una macchina virtuale alle applicazioni.
- Relativamente ai dischi, il sistema operativo realizza due astrazioni:
 - Dispositivo a basso livello: un semplice vettore di blocchi di dati.
 - File system: il sistema operativo accoda e organizza le richieste provenienti da diverse applicazioni.



walter.balzano@gmail.com

Nomi dei file

- L'assegnazione dei nomi dei file sui mezzi rimovibili è complicata nel caso s'intenda scrivere dati su un mezzo rimovibile in un certo calcolatore e poi riutilizzare lo stesso mezzo in un altro calcolatore.
- In genere gli attuali sistemi operativi lasciano irrisolto il problema, confidando nel fatto che le applicazioni o gli utenti forniranno una chiave di lettura e di interpretazione dei dati.
- Alcuni tipi di mezzi rimovibili (ad es. i CD) sono così ben standardizzati da essere usati allo stesso modo da tutti i calcolatori.



walter.balzano@gmail.com

Gestione gerarchica della memoria

- Un sistema di gestione gerarchica della memoria estende la gerarchia di memorizzazione oltre la memoria centrale e secondaria, comprendendo la memoria terziaria; quest'ultima è di solito costituita da un juke-box di nastri o di dischi rimovibili.
- In genere la memoria terziaria viene incorporata estendendo il file system.
 - I file piccoli e frequentemente usati rimangono nei dischi magnetici.
 - I file vecchi, ingombranti e raramente necessari, si archiviano nel juke-box.
- La gestione gerarchica della memoria (HSM, *hierarchical storage management*) si trova di solito in centri di calcolo basati su supercalcolatori e in altri grandi sistemi che possiedono enormi quantità di dati.



walter.balzano@gmail.com

Velocità

- La velocità della memoria terziaria è definita da due fattori: ampiezza di banda e latenza.
- L'ampiezza di banda si misura in byte al secondo.
 - Ampiezza di banda sostenuta: velocità media di trasferimento nel caso di una rilevante quantità di dati; in altre parole, il numero di byte diviso il tempo di trasferimento.
 - Ampiezza di banda sostenuta: numero di byte trasferiti rapportato al tempo di I/O totale, inclusi il tempo richiesto da una **seek** o una **locate**, e l'attesa eventualmente dovuta a cambi di dischi o nastri eseguita dal juke-box.



walter.balzano@gmail.com

Velocità

- **Latenza d'accesso:** quantità di tempo necessaria per accedere ai dati.
 - **Tempo d'accesso per un disco:** si sposta il braccio al cilindro selezionato e si aspetta che il settore interessato ruoti sotto la testina.
 - Un accesso diretto a un nastro richiede lo svolgimento o il riavvolgimento della bobina finché il blocco richiesto raggiunge la testina, in decine o in centinaia di secondi.
 - In linea generale l'accesso diretto a un nastro è oltre mille volte più lento dell'accesso diretto a un disco.
- La convenienza economica della memoria terziaria è dovuta alla possibilità di usare molte cartucce (a disco o a nastro) a basso costo, con poche costose unità di lettura e scrittura.
- Un archivio di dati rimovibili è soprattutto adatto alla registrazione di dati usati raramente, perché il numero delle richieste di I/O soddisfacibili per ogni ora d'uso di un tale archivio è relativamente basso.



walter.balzano@gmail.com

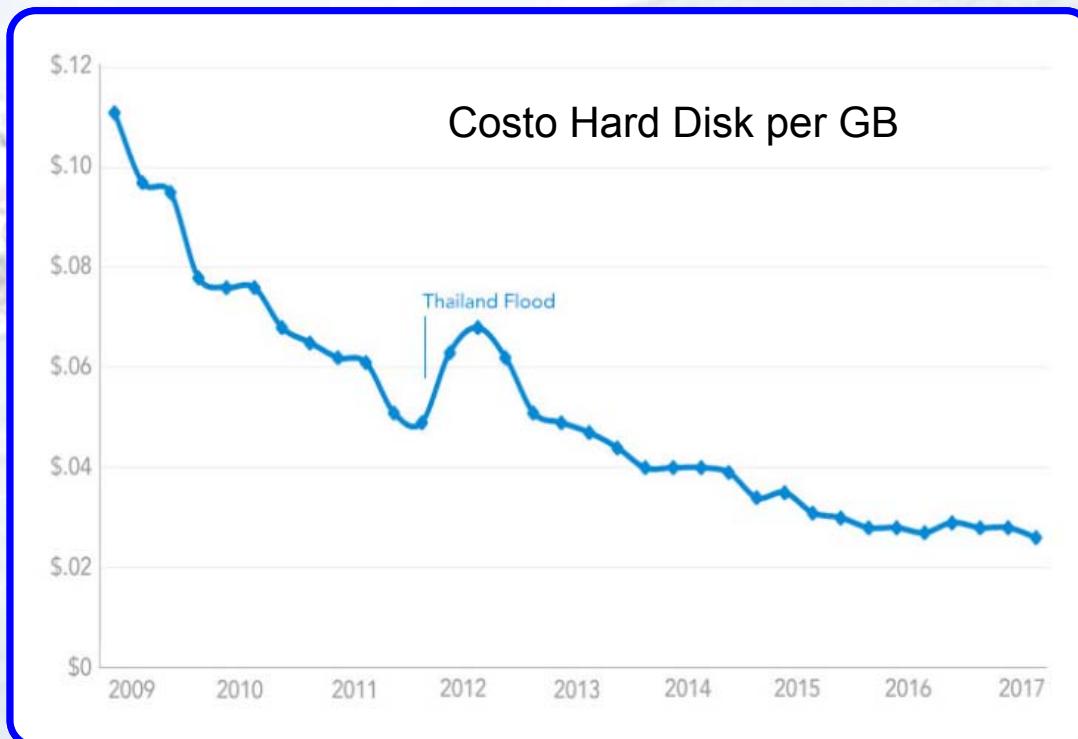
Affidabilità

- I dischi magnetici rimovibili sono meno affidabili dei dischi fissi.
- I dischi ottici sono considerati più affidabili di un disco o un nastro magnetico perché lo strato che memorizza le informazioni è protetto da uno strato trasparente di plastica o vetro.
- Anche le unità a disco fisso hanno punti deboli: la collisione della testina col disco in genere distrugge i dati, mentre il guasto di un'unità a nastro o di un'unità a dischi ottici lascia spesso intatto il mezzo di memorizzazione in uso al momento del guasto.



walter.balzano@gmail.com

Costi



walter.balzano@gmail.com

Fine