

*CdL Informatica*

*a.a. 2016-2017*

# **Architettura degli Elaboratori e Sistemi Operativi**

*Prof.Ing. Donato Impedovo*

---

*Università degli Studi di Bari*

[donato.impedovo@uniba.it](mailto:donato.impedovo@uniba.it)



# Programma

## Architettura

Introduzione agli Elaboratori

Rappresentazione dell'Informazione

Livello Logico–Digitale

Organizzazione dei Sistemi di Calcolo

## OS

Introduzione agli OS

Descrizione e Controllo dei Processi

Thread, SMP e Microkernel

Concorrenza: mutua esclusione e Sincronizzazione

Gestione della memoria

## PRE-REQUISITI

- Nozioni di analisi: funzioni, funzioni ricorsive, serie numeriche e geometriche.
- Elementi di algebra e geometria: insiemi, vettori, matrici, grafi ed alberi.
- Elementi di fisica: elettromagnetismo.

# Testi di riferimento

## TESTI DI RIFERIMENTO:

- A.S. Tanenbaum. Architettura dei calcolatori. Un approccio strutturale, Pearson 2013 (€49)
- A. Silberschatz. Sistemi Operativi: Concetti ed esempi. Pearson 2014 (€42)
- Sistemi Operativi. W. Stallings, Jackson Libri, 2000
- Dispense del Professore (€0)
- Vostri APPUNTI (Priceless)

# Ricevimento studenti / esame

## RICEVIMENTO STUDENTI:

1. Partecipazione ATTIVA ed INTERATTIVA alle lezioni
2. Lunedì 10–13, stanza 609

## ESAME:

Prenotazione all'esame: portale [ESSE3](#)

### Prova scritta inerente:

- Aspetti teorici
- Aspetti progettuali
- Esercizi pratici numerici  
Tipicamente 3 domande

### Prova orale inerente:

- Aspetti teorici
- Aspetti progettuali
- Esercizi pratici numerici  
Tutto il programma

.... mha

Mr. D. Barker, 1408 Chap-  
man Blg.

# MEN WANTED

for hazardous journey, small wages,  
bitter cold, long months of complete  
darkness, constant danger, safe re-  
turn doubtful, honor and recognition  
in case of success.

Ernest Shackleton 4 Burlington st.

MEN—Neat-appearing YOUNG men of  
pleasing posse

# Informatica

## Informazione automatica

Tecnologia che consente il trattamento automatico delle **informazioni** e dei **dati** in maniera automatica per mezzo di **elaboratori** (computer).

- ➔ **SISTEMA:** Realtà di tipo complesso che presenta le seguenti caratteristiche:
  - ➔ Essere costituita da un insieme di elementi identificabili separatamente;
  - ➔ Manifestarsi di interazioni significative tra gli elementi costituenti il sistema stesso;
  - ➔ La presenza di obiettivi o finalità che orientano il comportamento del sistema.
- ➔ **DATO:** rappresentazione originaria di un fenomeno, descrizione di una qualsiasi caratteristica della realtà che necessita di un'interpretazione per produrre conoscenza.
- ➔ **INFORMAZIONE:** l'insieme di uno o più dati, memorizzati, classificati, organizzati, messi in relazione o interpretati nell'ambito di un contesto in modo da assumere un significato.
- ➔ **ELABORATORE:** macchina in grado di elaborare grandi quantità di dati in modo automatico ad altissima velocità, di memorizzare e ritrovare dati organizzati in modo strutturato.

# Information and Communication Technology (ICT)

Insieme delle tecnologie che consentono di elaborare e trasmettere (comunicare) l'informazione attraverso mezzi digitali.

MEZZI per lo sviluppo delle ICT: studio, progettazione, sviluppo, realizzazione, supporto e gestione dei sistemi informativi e di telecomunicazione computerizzati.

FINE: manipolazione dei dati tramite conversione, immagazzinamento, protezione, trasmissione e recupero sicuro delle informazioni.

UTILIZZO: mettere a disposizione dati e informazioni all'interno dell'organizzazione, ridefinire i rapporti con clienti, fornitori e altre organizzazioni. Integrazione dei servizi.

La tecnologia dell'informazione comprende:

- reti di telecomunicazioni,
- l'architettura aperta (client/server),
- la multimedialità

# ...storia

- ABACO: primo calcolatore meccanico, Babilonesi, 500 a.c.
- Pascal (matematico e filosofo francese): elaboratore meccanico ad ingranaggi in grado di fare le 4 operazioni (1642)
- Charles Babbage: calcolatore meccanico in grado di generare carte di navigazione, alimentato da motore a vapore, programmabile tramite schede perforate, 50000 pezzi meccanici (1820)
- Konrad Zuse (ingegnere): primo prototipo degli odierni computer (5Hz), componenti elettromeccaniche (1936)

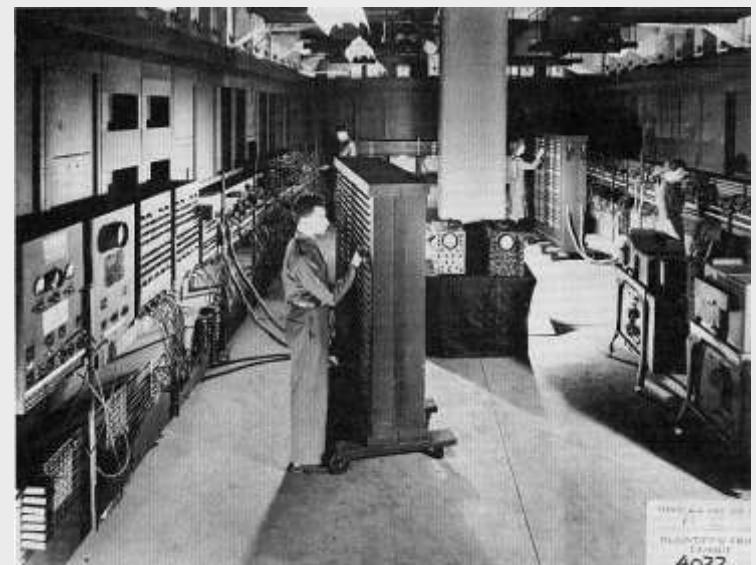
*ENIAC (1946 ca.)*

~~Electronical Numerical Integrator and Calculator~~  
esigenze belliche (calcolo di tavole balistiche)

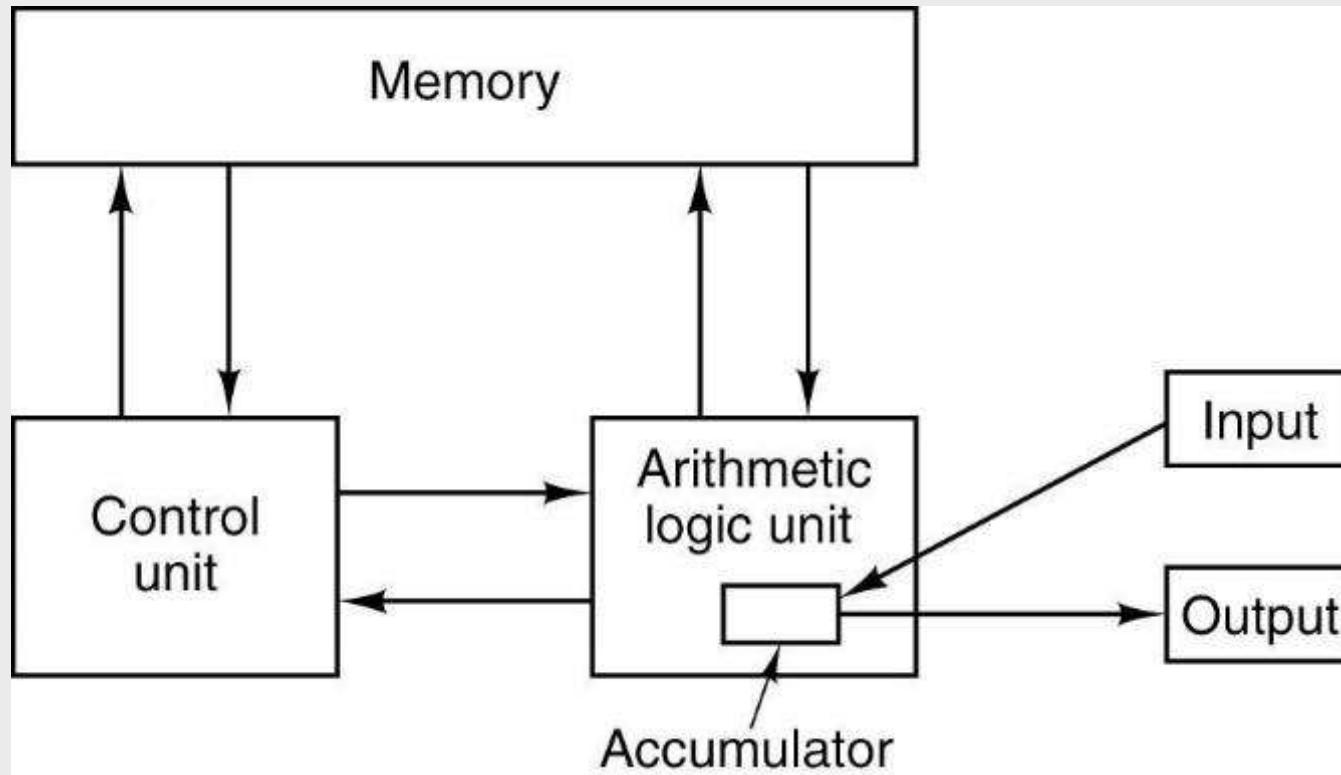
Commissionato dal Dipartimento di Guerra degli Stati Uniti all'Università della Pennsylvania, prototipo realizzato nel 1946.

20000 valvole, occupava una stanza lunga più di 30 metri, pesava 30 tonnellate, dissipava una quantità enorme di energia elettrica.

100000 operazioni al secondo

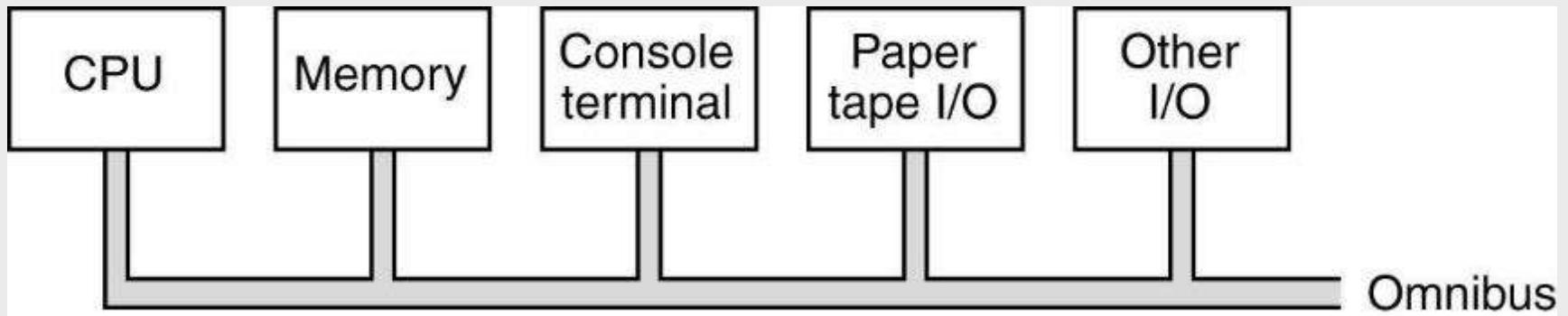


# Von Neumann Machine (1952)



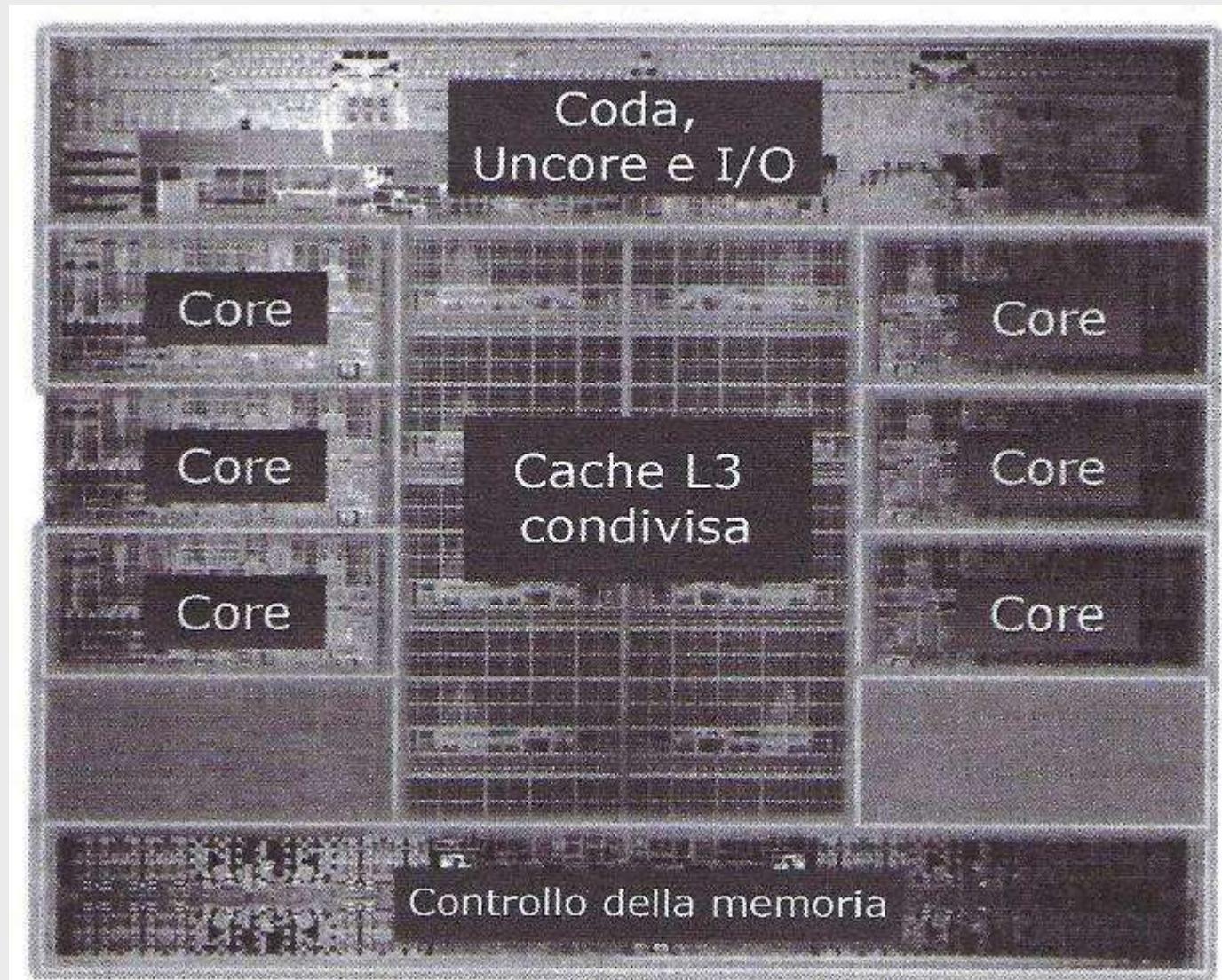
The original Von Neumann machine.

# PDP-8 Innovation – Single Bus (1965)



The PDP-8 omnibus

# Intel i7-3960X (2011)



# Linguaggi, livelli e macchine virtuali

L0: linguaggio di macchina

L1: linguaggio di più alto livello

Se gli utenti si esprimono nel linguaggio L1 e il computer è in grado di eseguire soltanto le istruzioni del linguaggio L0, allora il problema diventa quello di tradurre le istruzioni di L1 in quelle di L0 e lo si può fare usando un **traduttore**.

Un altro modo è quello di scrivere un programma in L0 che accetta come dati in ingresso comandi in L1, tale programma esamina uno dopo l'altro i comandi e li sostituisce con l'equivalente sequenza di istruzioni di L0, questa è la tecnica della interpretazione ed il programma che lo fa è detto **interprete**.

# Linguaggi, livelli e macchine virtuali

**Traduzione:** il programma iniziale viene convertito in linguaggio macchina e la macchina eseguirà le istruzioni tradotte, si parla di **compilazione**

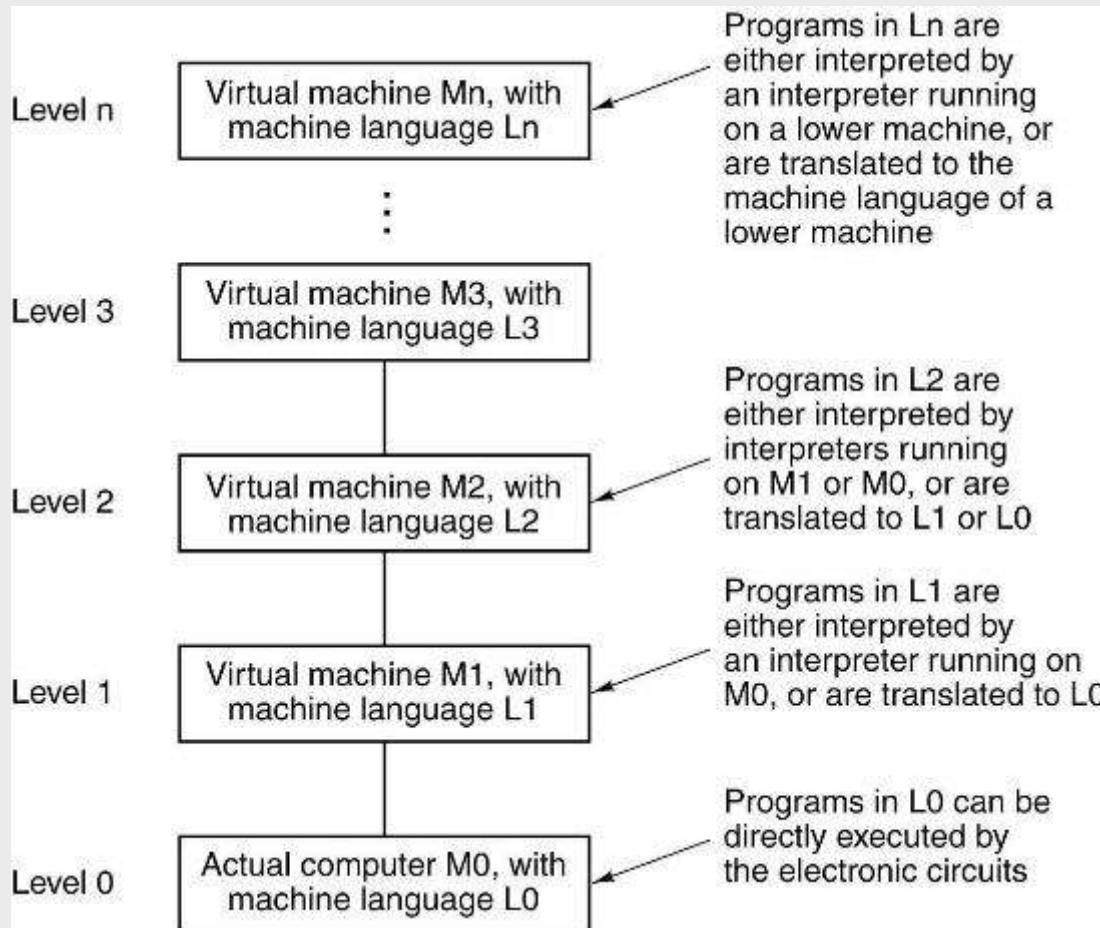
**Interpretazione:** le istruzioni di macchina vengono generate in corrispondenza delle istruzioni del programma utente una dopo l'altra.

Nel primo caso durante l'esecuzione il computer ha il controllo del programma L0 mentre nell'uso dei linguaggi interpretati il computer ha il controllo del linguaggio L1.

Anzicchè parlare di linguaggi si preferisce di parlare di macchine: se M0 è la macchina del linguaggio L0, M1 sarà la macchina del linguaggio L1.

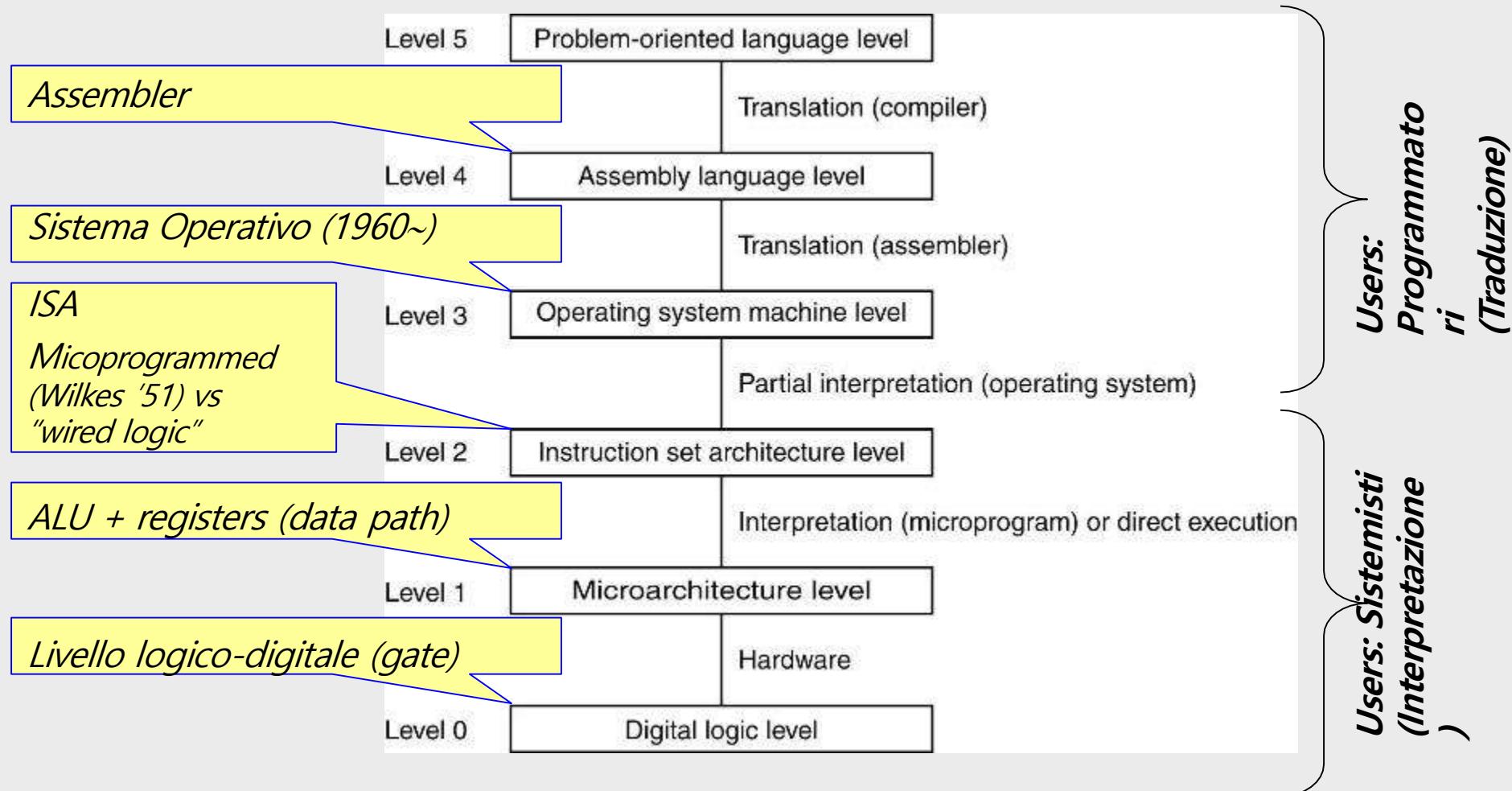
# Linguaggi, livelli e macchine virtuali

## A multilevel machine



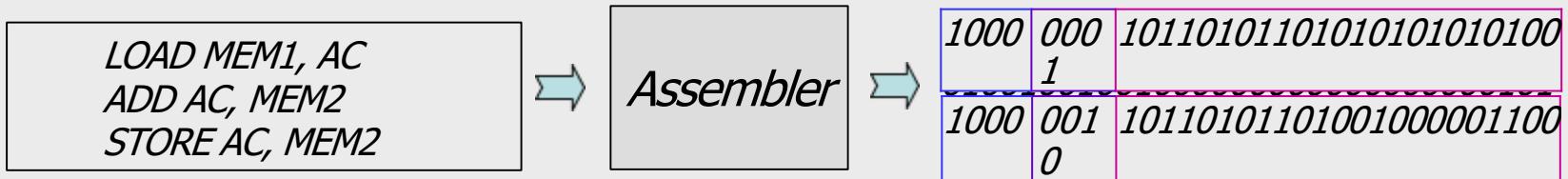
# Macchine Multilivello Attuali

A six-level computer.



# Assembler

- In principio era il linguaggio macchina....
- Per facilitare la programmazione fu introdotto il linguaggio assembly
- L'assembly definisce una notazione simbolica che è in stretta relazione (1:1) con i codici in linguaggio macchina. Il programma scritto in assembly è convertito automaticamente in linguaggio macchina per mezzo del programma traduttore assembler



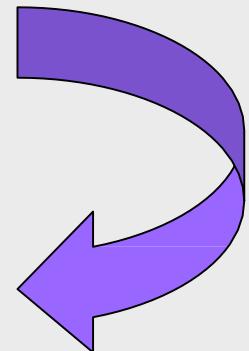
# Assembler e linguaggi di alto livello

- Il programma in assembler...

```
LOAD a,REG1  
LOAD b,REG2  
ADD REG1,REG2  
LOAD c,REG3  
LOAD d,REG4  
ADD REG3,REG4  
MULT REG1,REG3  
STORE REG1,e
```

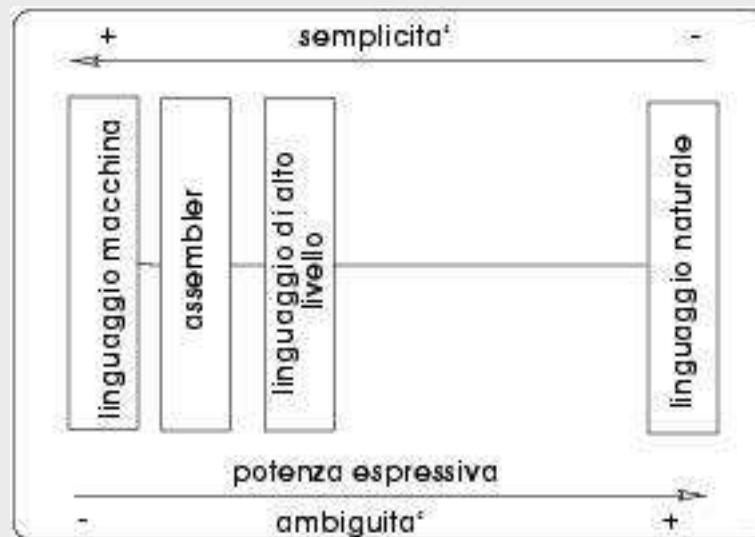
*Corrisponde all'unica  
istruzione C...*

$$e = (a+b)*(c+d);$$



# Linguaggi di programmazione

- Oggi si utilizza l'assembly solo se esistono vincoli stringenti sui tempi di esecuzione e sulla architettura; viceversa si usano linguaggi più vicini al linguaggio naturale, i **linguaggi di alto livello**
- I linguaggi di alto livello sono elementi intermedi di una varietà di linguaggi ai cui estremi si trovano il linguaggio macchina ed i linguaggi naturali



# Linguaggi di programmazione

I linguaggi che non dipendono dall'architettura della macchina offrono due vantaggi fondamentali:

- ◆ i programmatori possono astrarre dai dettagli architettonici di ogni calcolatore
- ◆ i programmi risultano più semplici da leggere e da modificare

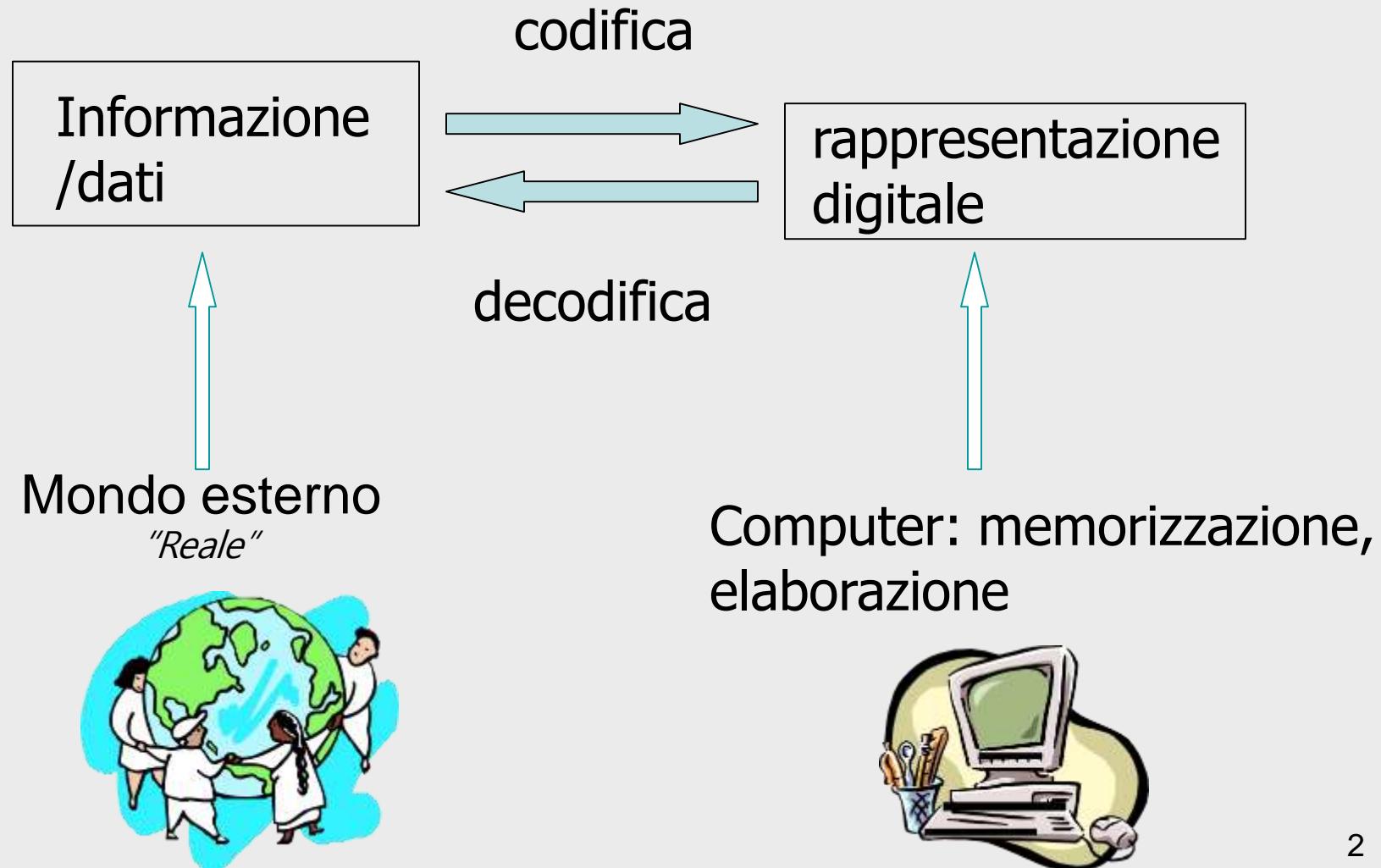
⇒ **portabilità, leggibilità, mantenibilità**

# RAPPRESENTAZIONE DELL'INFORMAZIONE

*Prof. Ing. Donato Impedovo*

---

# Rappresentazione digitale Dell'informazione



# Il BIT

Alfabeto del calcolatore costituito da due simboli: 0,1.

**BIT (binary digit):**

- | Unità elementare di informazione.
- | La cifra binaria può assumere solo due valori alternativi: 0 oppure 1.

- | Archiviato da un dispositivo digitale o un sistema fisico che esiste in uno di due possibili stati distinti.

Es.:

- i due stati stabili di un flip-flop,
- due posizioni di un interruttore elettrico,
- due distinte tensione o gli attuali livelli consentiti da un circuito,
- due distinti livelli di intensità della luce,
- due direzioni di magnetizzazione o di polarizzazione, ecc

Presenza	Assenza
Vero	Falso
1	0
Acceso	Spento
+	-
Sì	No
Favorevole	Contrario
Yang	Yin
Lisa	Bart

# Sequenze di BIT

Per poter rappresentare un numero maggiore di informazione si usano sequenze di bit

Il processo che fa corrispondere ad un dato reale una sequenze di bit prende il nome  
**codifica dell'informazione**

Es.1: un esame può avere quattro possibili esiti: ottimo, discreto, sufficiente, insufficiente. Quanti bit sono necessari per codificare tale informazione?

**due bit:**

ottimo	00
discreto	01
sufficiente	10
insufficiente	11

Es.2: otto colori: nero, **rosso**, **blu**, **giallo**, **verde**, viola, grigio, **arancione**

**tre bit:**

nero	000
<b>rosso</b>	001
<b>blu</b>	010
<b>giallo</b>	011
<b>verde</b>	100
viola	101
grigio	110
<b>arancione</b>	111

# bit, Byte e word

- Con N bit si possono codificare  $2^N$  stati differenti
- Per rappresentare N stati, devo usare almeno  $\log_2(N)$  bit

Vogliamo codificare 19 colori... quanti bit servono??  
 $\text{ceil}(\log_2(N))$  – arrotonda all'intero più grande

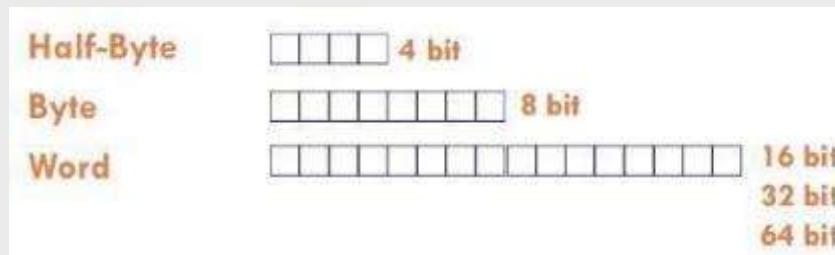
I sistemi moderni memorizzano e manipolano miliardi di bit: necessità di multipli.  
Le informazioni sono rappresentate mediante stringhe di bit

8 bit (b) = 1 Byte (B)



OSSERVAZIONE: con la lettera b, minuscolo, si indicano i bit, con la lettera B, maiuscolo, si indicano i byte

Strutture logiche:



# bit, Byte e multipli

I moderni sistemi memorizzano e manipolano molti bit e byte: necessità di multipli

K – chilo (mille)  
 M – Mega (milioni)  
 G – Giga (miliardi)  
 T – Tera (migliaia di miliardi)

Nome	Sigla	In bit	In byte	In potenze di 2
Bit	Bit	1 bit	1/8	$2^1 = 2$ stati
Byte	Byte	8	1	$2^8 = 256$ stati
KiloByte	KB	8.192	1.024	$2^{10}$ byte
MegaByte	MB	8.388.608	1.048.576	$2^{20}$ byte
GigaByte	GB	8.589.934.592	1.073.741.824	$2^{30}$ byte
TeraByte	TB	8.796.093.022.208	1.099.511.627.776	$2^{40}$ byte

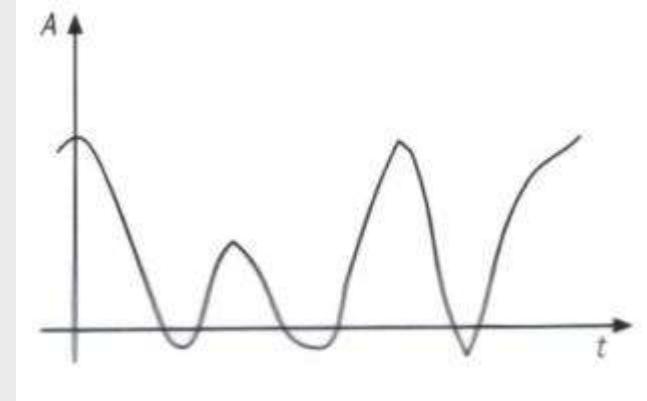
Esempio: un hard disk (HD) da 500GB contiene esattamente:

$$500 \times 2^{30} = 500 \times 2^{10} \times 2^{10} \times 2^{10} = 500 \times 1024 \times 1024 \times 1024 = 536.870.912.000 \text{ byte}$$

E in termini di bit?

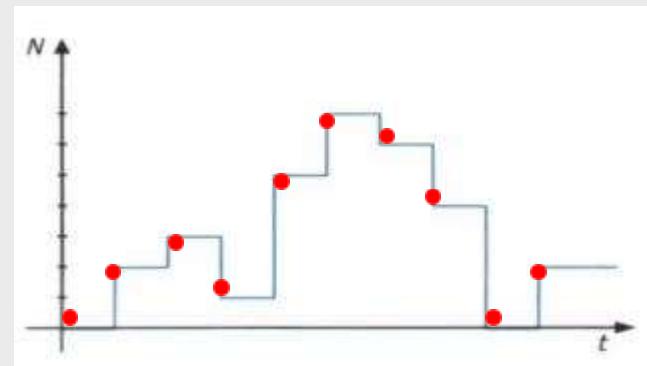
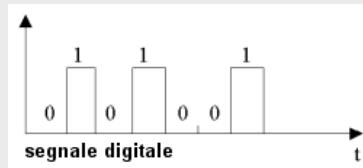
# Rappresentazione digitale dei segnali

**SEGNALE ANALOGICO:** segnale tempo-continuo che può assumere tutti gli infiniti valori della grandezza fisica osservabile che rappresenta contenuti all'interno di un determinato range. I valori utili che lo rappresentano sono continui (infiniti) in un intervallo e non numerabili.



**SEGNALE DIGITALE:** segnale tempo-discreto che all'interno di un determinato range può assumere solo un numero discreto (numerabile finito) di valori.

Es.: onda quadra che assume i valori logici ALTO (1) e BASSO (0).



Vantaggi:

- utilizzo di un unico linguaggio (binario)
- costanza di qualità nel tempo
- possibilità di compressione

# Rappresentazione digitale di segnali

## CAMPIONAMENTO:

processo di conversione di un segnale tempo-continuo in un segnale tempo-discreto,

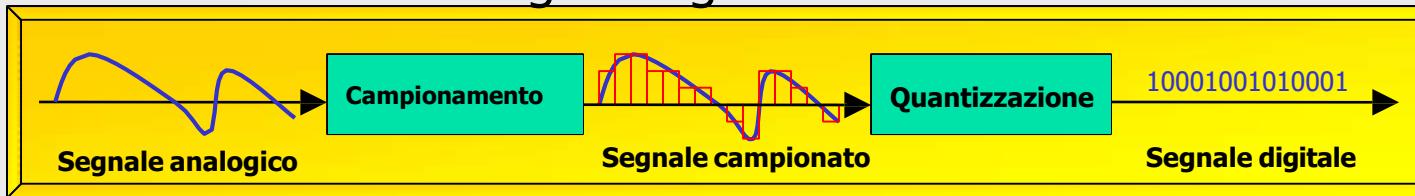
L'ampiezza del segnale continuo viene considerata a intervalli di tempo regolari ( $T$  - periodo di campionamento).

## QUANTIZZAZIONE:

processo di conversione di un segnale a valori continui in uno a valori discreti.

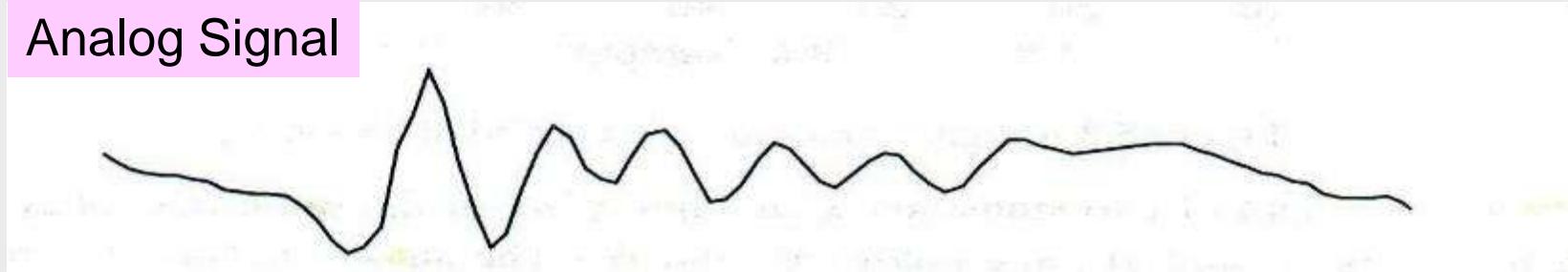
Più è alto il numero di bit utilizzati nella quantizzazione e minore è l'errore che si commette (errore di quantizzazione), cioè si riduce la distanza media tra il valore campionato (continuo) e il corrispondente valore quantizzato

## Analog to Digital Converter



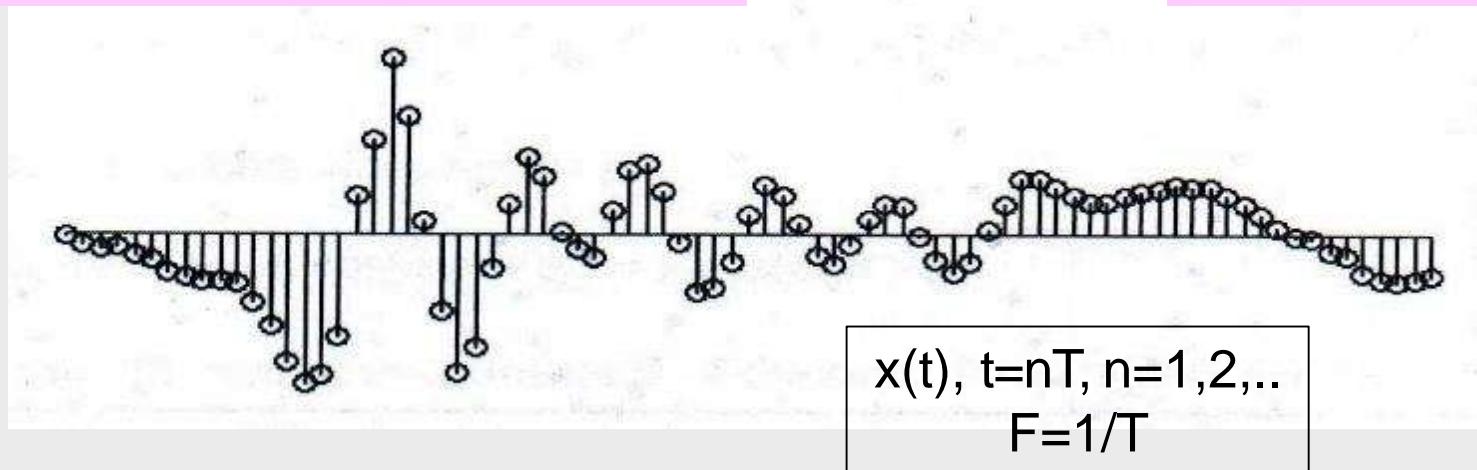
# Rappresentazione digitale di segnali

Analog Signal



Sampling: Discrete-time Signal

Quantization

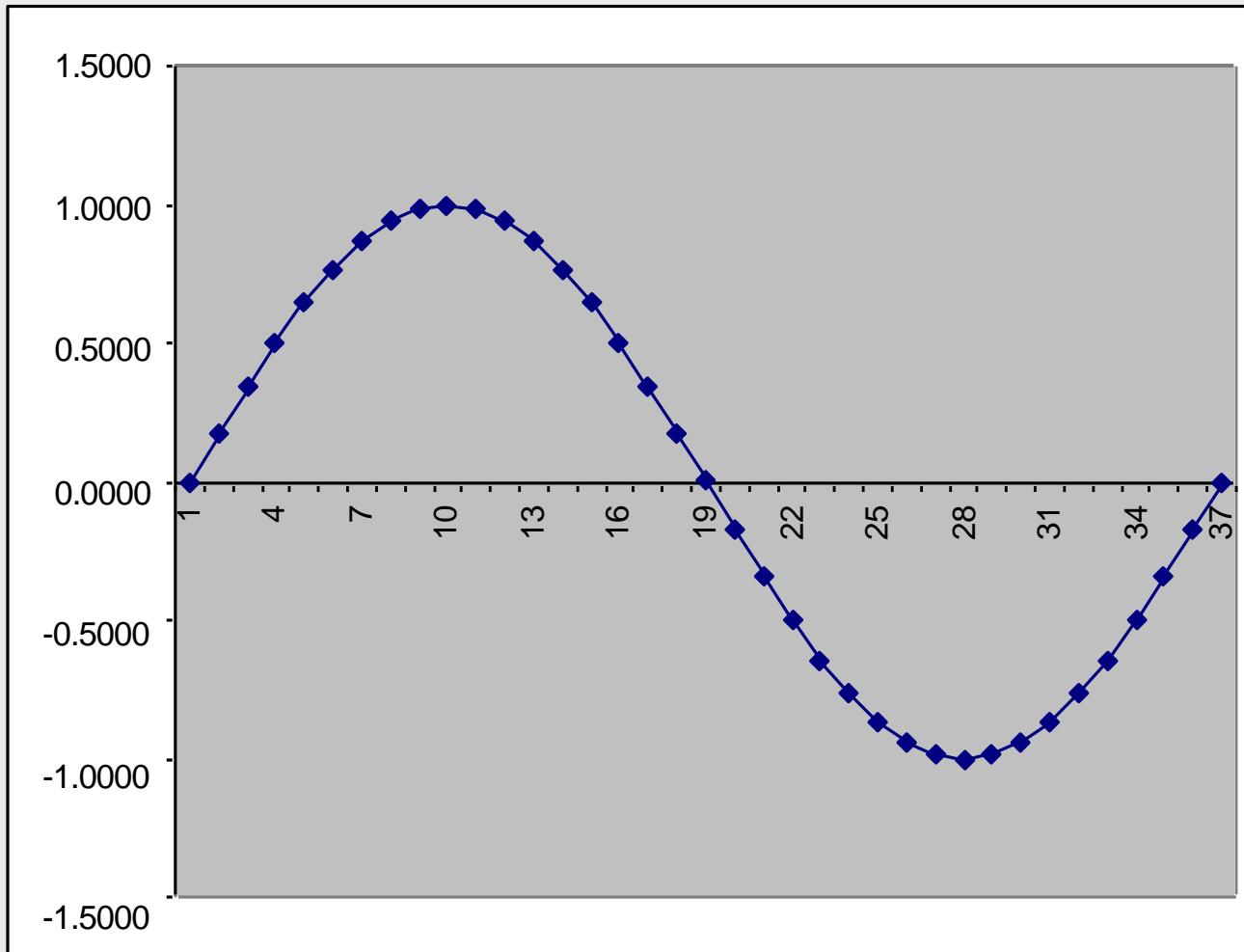


# Problemi legati al campionamento

Segnale sinusoidale a frequenza  $f_s=27$  [Hz]

$f_c=1\text{ KHz}$

$f_c=37\text{ fs}$



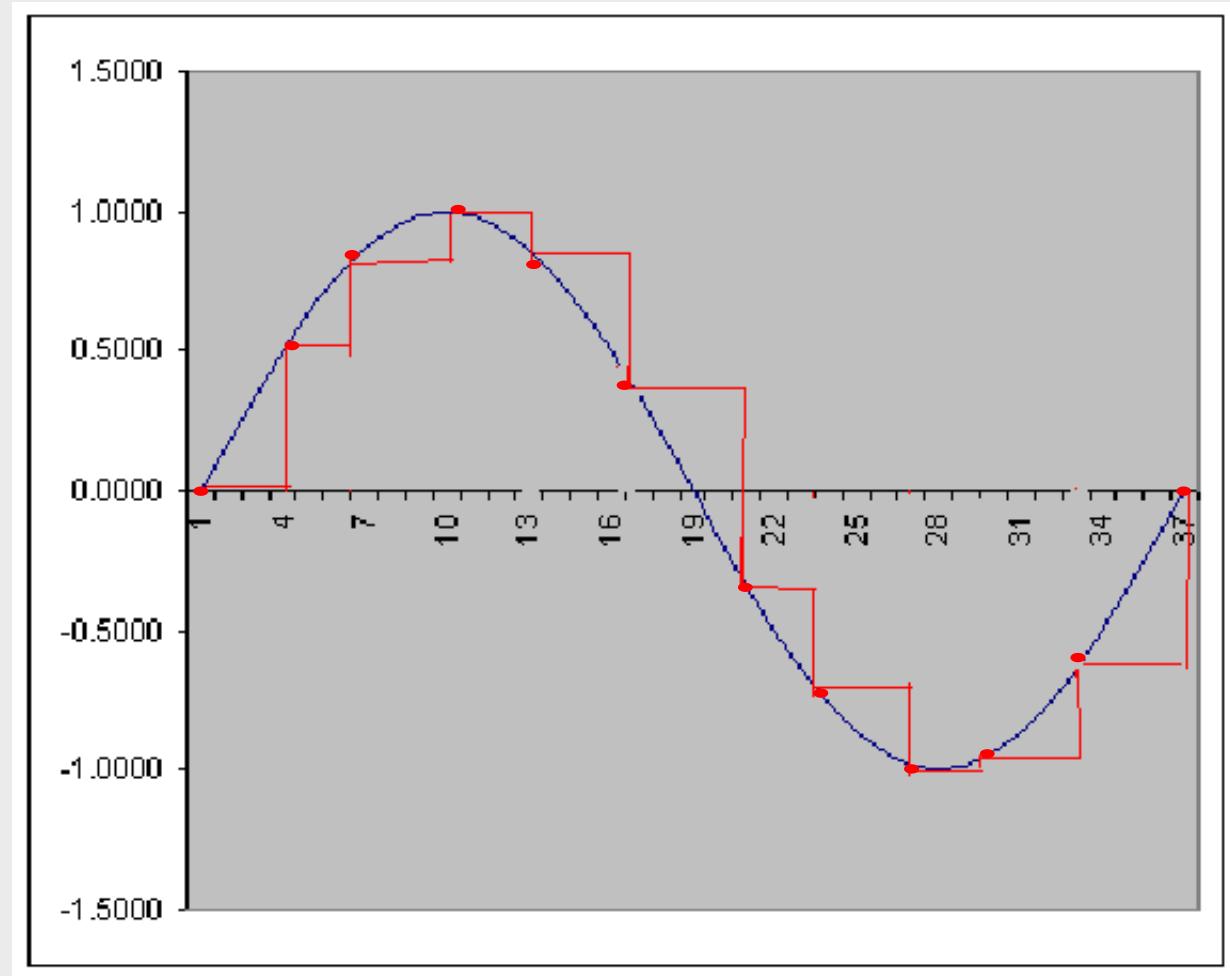
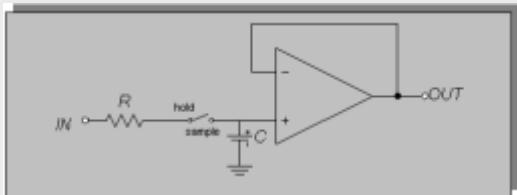
# Problemi legati al campionamento

Segnale sinusoidale a frequenza  $f_s=27$  [Hz]

$fc=333$  Hz

$fc=12,33$  fs

— *Sample and hold*

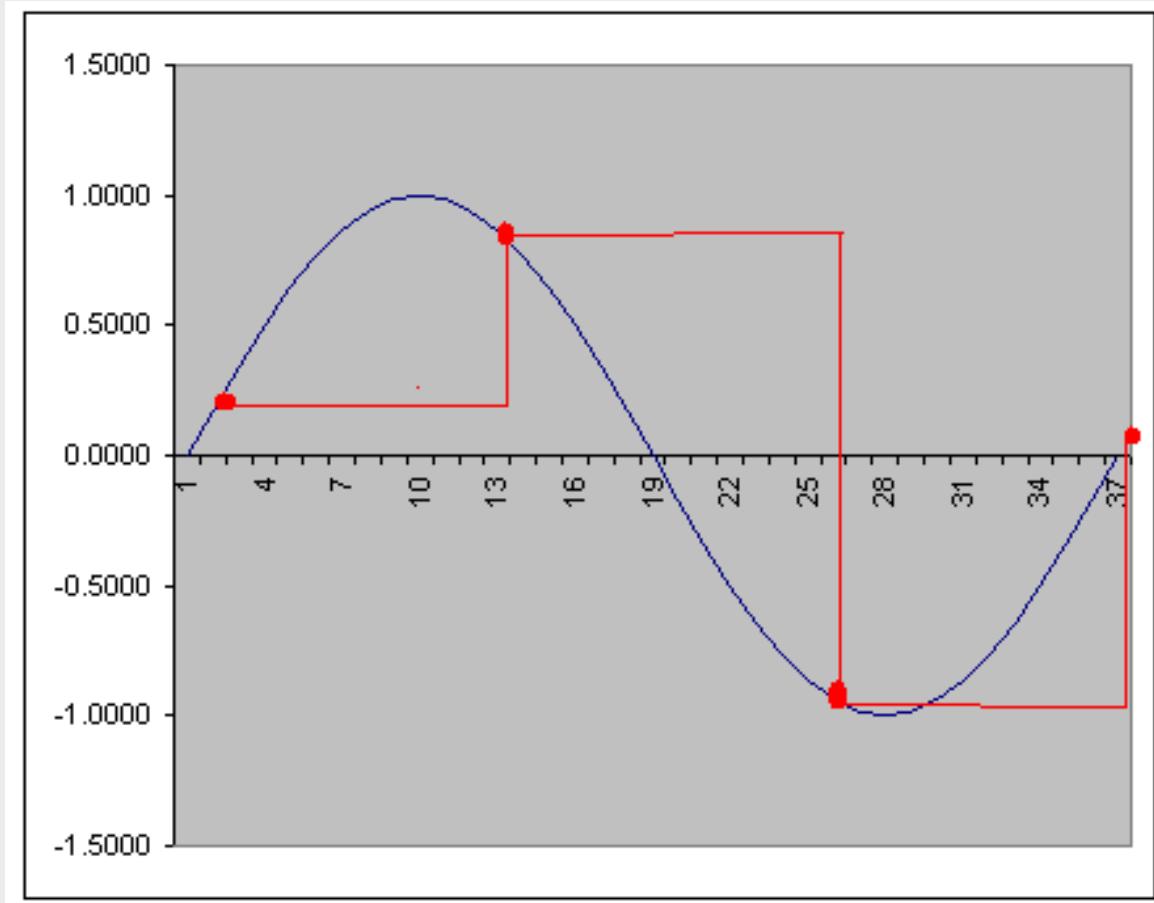


# Problemi legati al campionamento

Segnale sinusoidale a frequenza  $f_s=27$  [Hz]

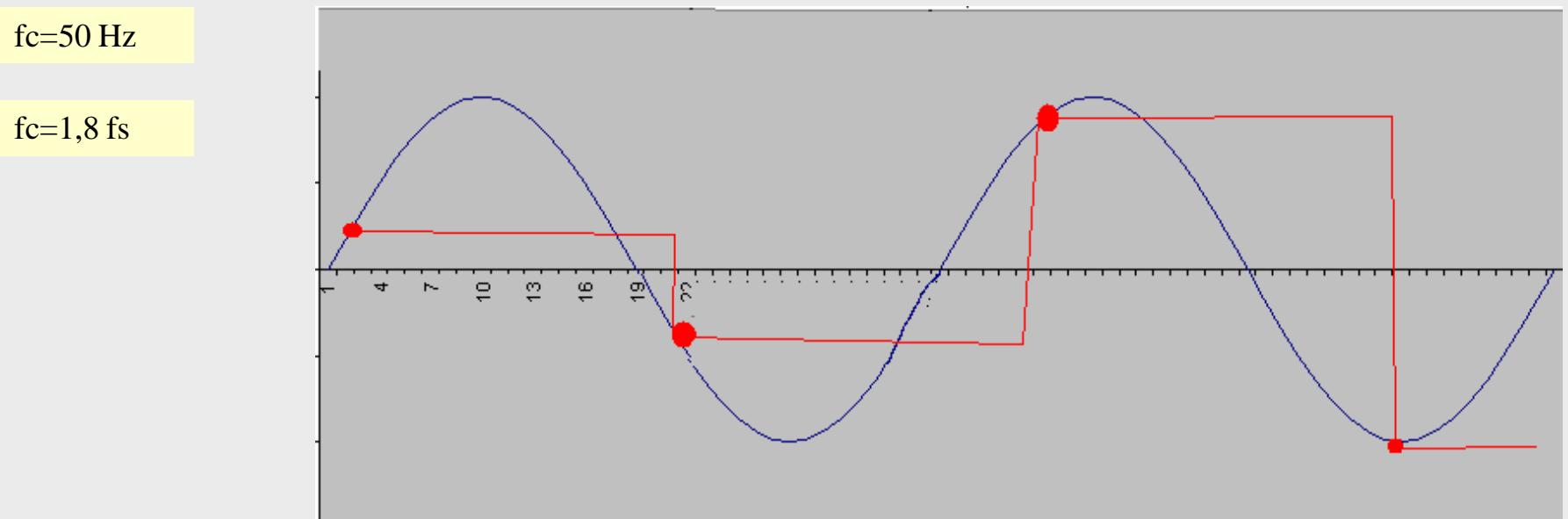
$f_c=110$  Hz

$f_c=4$  fs



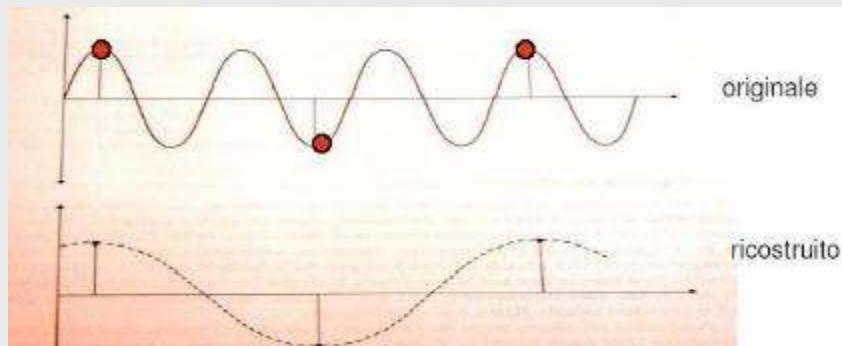
# Problemi legati al campionamento

## Segnale sinusoidale a frequenza $f_s=27$ [Hz]

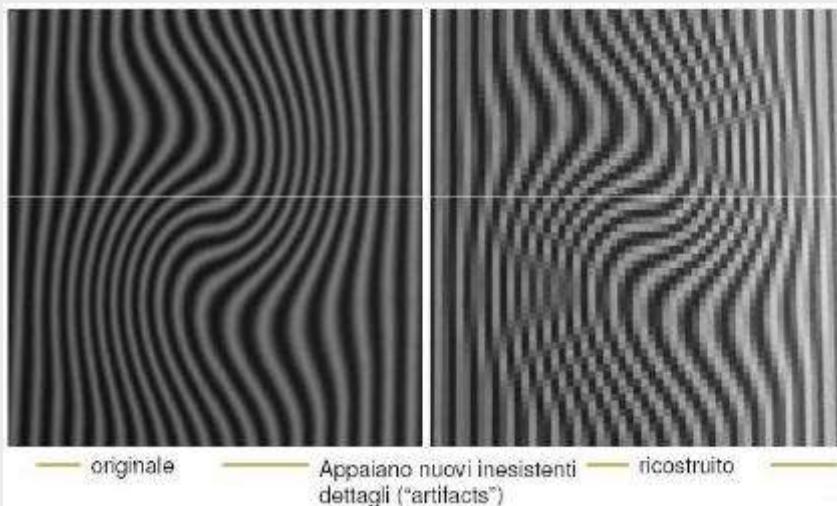


# Problemi del campionamento

## 1. Perdita di informazioni

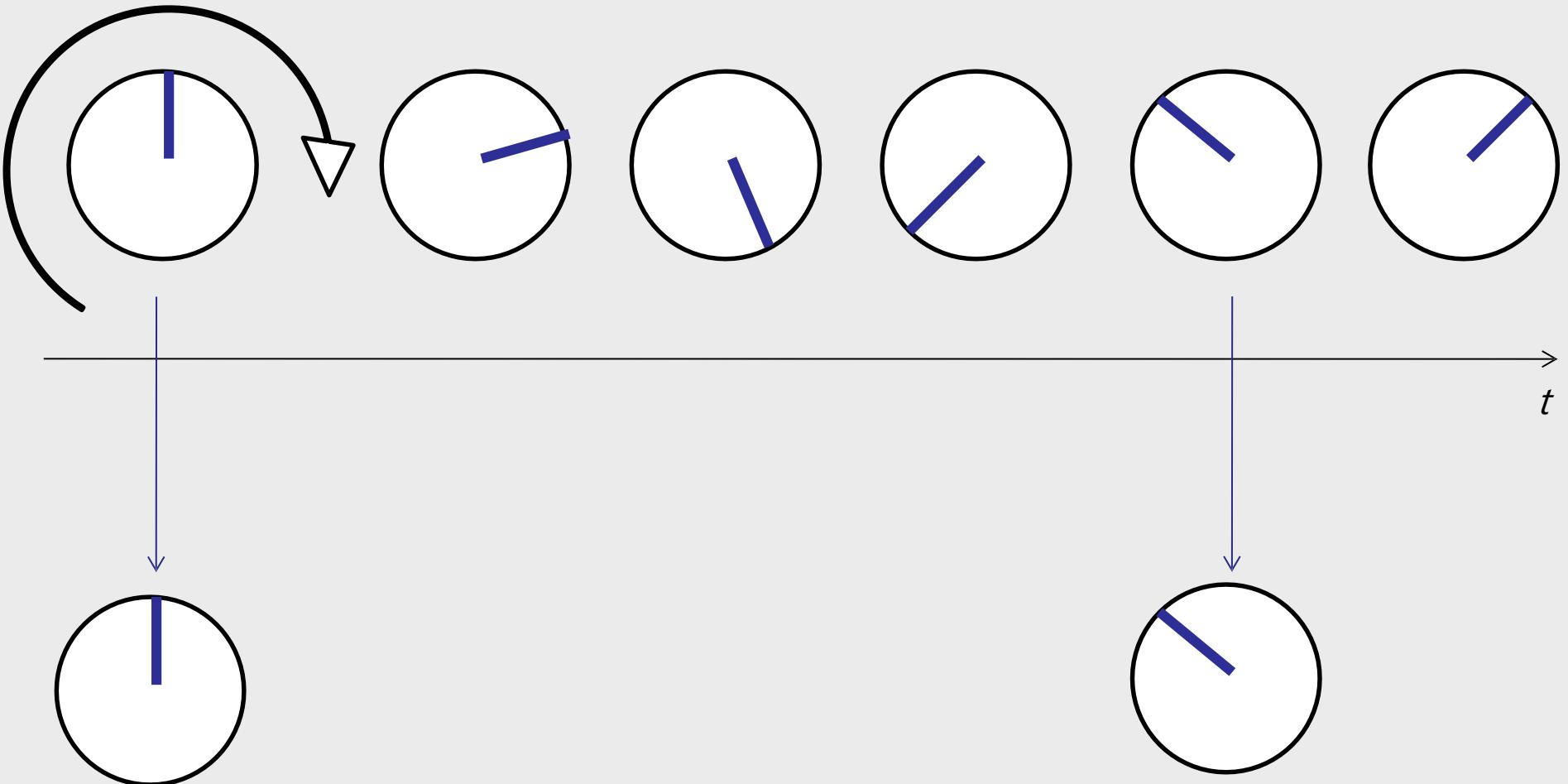


## 2. Comparsa di elementi NON PRESENTI nell'originale. ALIASING



ES.: ruota dell'auto  
Periodo di campionamento  
dell'occhio = 100 ms (10Hz)

# Problemi legati al campionamento



# Teorema del campionamento di Nyquist-Shannon

Dato un segnale, con larghezza di banda  $B$  finita e nota, la frequenza minima di campionamento, per poter ricostruire correttamente tale segnale, deve essere pari ad almeno  $2B$ .

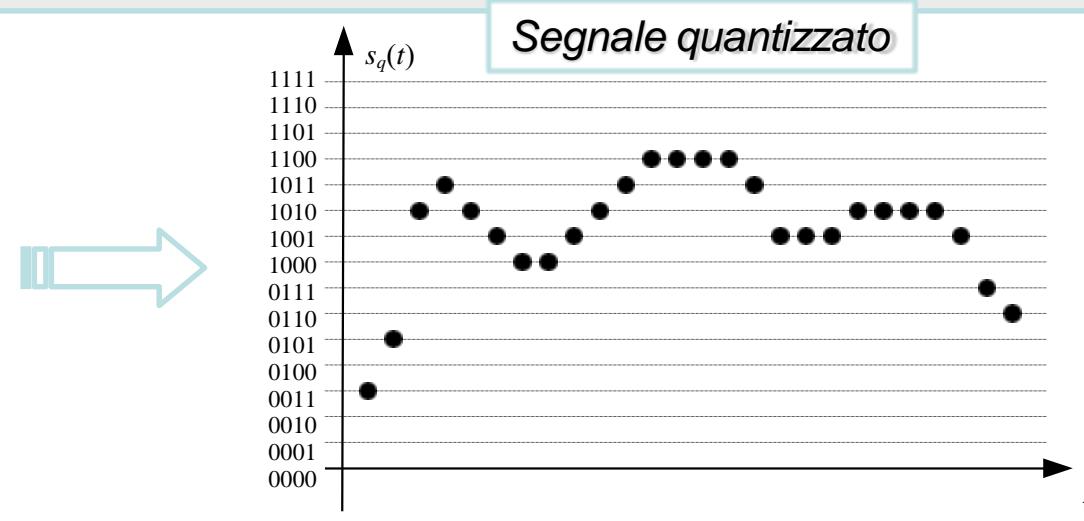
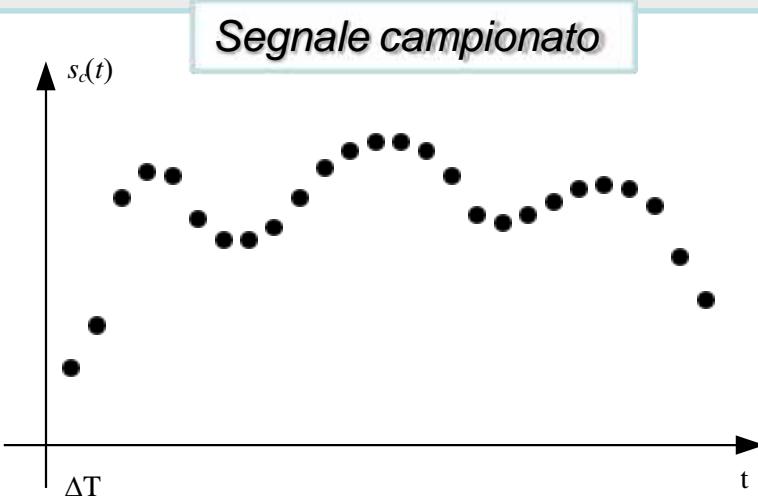
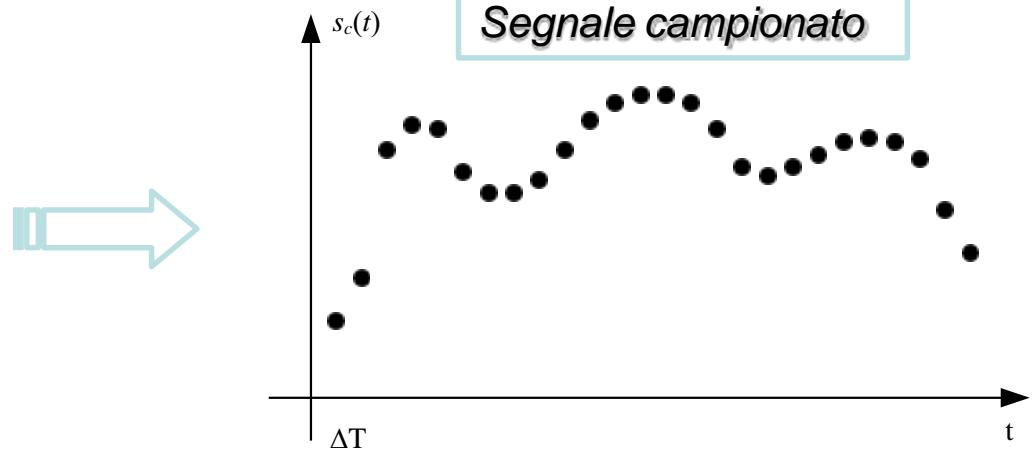
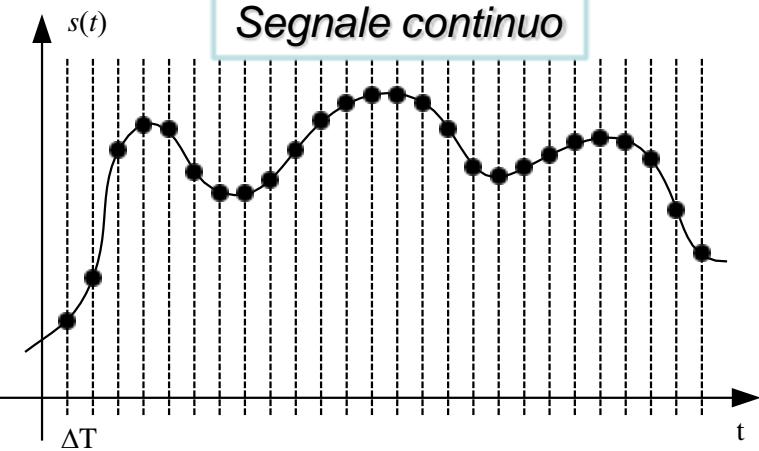
Un segnale campionato con frequenza pari a  $f_c$  avrà massima frequenza risolvibile pari a  $f_c/2$

Es.: i CD audio impiegano una frequenza di campionamento di 44.100 Hz (formato wave), la massima frequenza risolvibile è 22.050 Hz, appena sopra il limite percepibile dall'udito umano di 20.000 Hz.

Speech Production: up to 6–8KHz

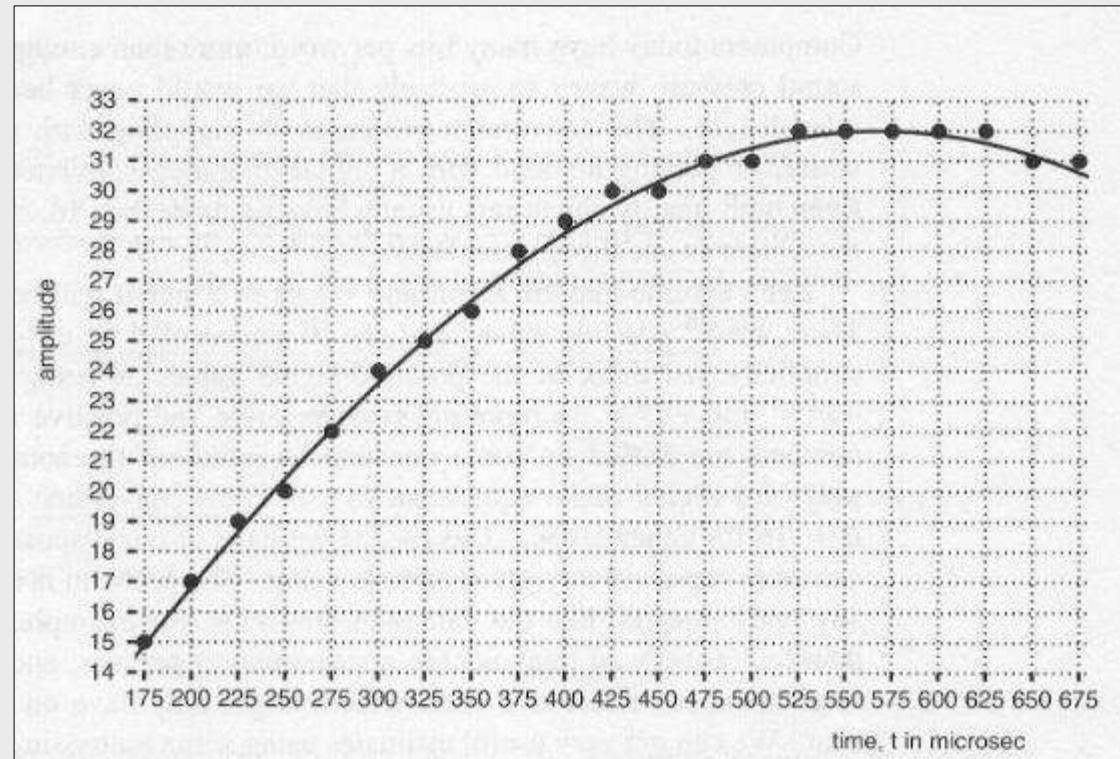
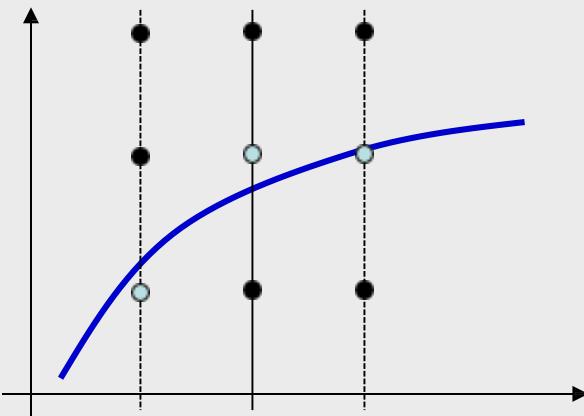
Speech Perception: up to 20KHz

# Rappresentazione digitale di segnali



# Quantizzazione

- ▶ La curva continua rappresenta la forma d'onda originaria, i punti rappresentano i campioni quantizzati al numero intero più vicino
- ▶ Il processo introduce delle imprecisioni dovute all'approssimazione del segnale reale (variabile in maniera continua) tramite un insieme finito di numeri.



# Codifica dei dati: SISTEMA NUMERICO

Un sistema numerico viene determinato per mezzo di:

Un insieme limitato di simboli (le cifre), che rappresentano quantità prestabilite

Es.: sistema decimale: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

sistema binario: 0, 1

sistema romano: I, V, X, L, C, D, M, ...

Regole per costruire i numeri:

1. Sistemi non posizionali, es. Romano, sistema additivo: le cifre sono sommate o sottratte sulla base dell'ordine in cui compaiono

Es: XII = $10+1+1=12$ , IX= $10-1=9$

2. Sistemi posizionali (es. decimale, binario, ecc.): il valore delle cifre dipende dalla loro posizione all'interno del numero

$$\text{BASE } 10: 2562 = 2 \cdot 10^3 + 5 \cdot 10^2 + 6 \cdot 10^1 + 2 \cdot 10^0$$

$$\text{BASE } 2: 101 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$\text{BASE } 8: 72 = 7 \cdot 8^1 + 2 \cdot 8^0$$

Regole che consentano di eseguire operazioni tra i numeri

# Sistemi in base B

La base definisce il numero di cifre diverse del sistema di numerazione

La cifra di minor valore (in termini di valore assoluto) è sempre lo 0, le altre sono, nell'ordine, 1, 2,...,B-1

Se  $B > 10$  occorre introdurre  $B-10$  simboli in aggiunta alle cifre decimali

Es: sistema esadecimale : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Un numero **intero** N si rappresenta con la scrittura  $(c_n c_{n-1} \dots c_2 c_1 c_0)_B$

$$N = c_n B^n + c_{n-1} B^{n-1} + \dots + c_2 B^2 + c_1 B^1 + c_0 B^0$$

$c_n$  è la **cifra più significativa**,  $c_0$  quella **meno significativa**

Un numero **frazionario** N' si rappresenta con la scrittura  $(0, c_1 c_2 \dots c_n)_B$

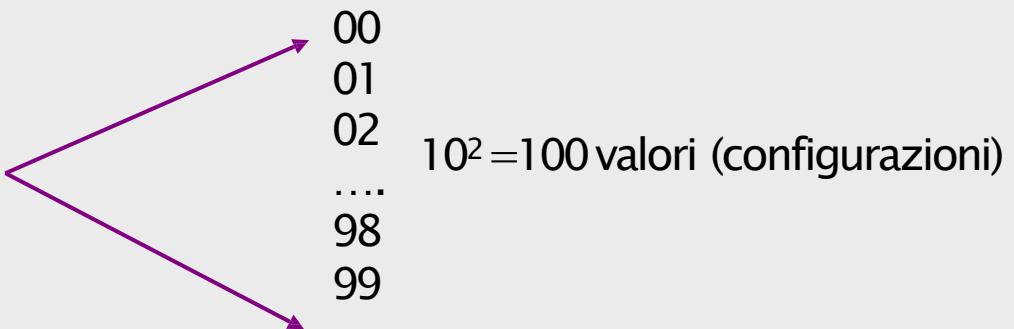
$$N' = c_1 B^{-1} + c_2 B^{-2} + \dots + c_n B^{-n}$$

# Numeri interi senza segno

Con  $n$  cifre, in base  $B$ , si rappresentano tutti i numeri interi positivi da 0 a  $B^n - 1$   
( $B^n$  numeri distinti)

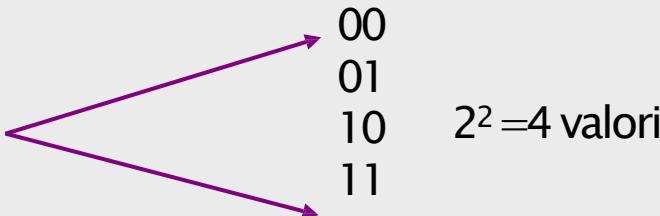
Esempio: base 10

2 cifre: da 0 a  $10^2 - 1 = 99$



Esempio: base 2

2 cifre: da 0 a  $2^2 - 1 = 3$



# Sistema in base 2 (sistema binario)

La base 2 è la più piccola per un sistema di numerazione

Cifre: 0 1 – bit (binary digit)

Esempi:

$$(101101)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ 32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$$

Forma  
polinomia

$$(0,0101)_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = \\ 0 + 0,25 + 0 + 0,0625 = (0,3125)_{10}$$

$$(11,101)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ 2 + 1 + 0,5 + 0 + 0,125 = (3,625)_{10}$$

# Conversioni di base: dec2bin

Si divide ripetutamente il numero intero decimale per 2 fino ad ottenere un quoziente nullo. Le cifre del numero binario sono i resti delle divisioni; la cifra più significativa è l'ultimo resto ottenuto

Esempi:

-Di quanti bit ho bisogno??...

$$\begin{array}{r}
 43 : 2 = 21 + 1 \\
 21 : 2 = 10 + 1 \\
 10 : 2 = 5 + 0 \\
 5 : 2 = 2 + 1 \\
 2 : 2 = 1 + 0 \\
 1 : 2 = 0 + 1
 \end{array}$$

resti

*Quoziente nullo*

$(43)_{10} = (101011)_2$

bit più significativo

$  \begin{array}{r}  34 : 2 = 17 + 0 \\  17 : 2 = 8 + 1 \\  8 : 2 = 4 + 0 \\  4 : 2 = 2 + 0 \\  2 : 2 = 1 + 1  \end{array}  $		$  \begin{array}{r}  12 : 2 = 6 + 0 \\  6 : 2 = 3 + 0 \\  3 : 2 = 1 + 1  \end{array}  $
		1100 100010

Ottenuto il quoziente 1 sulla colonna di sx, lo si replica sulla colonna di dx

## Esercizio

Si verifichi se le seguenti corrispondenze sono corrette:

$$(110010)_2 = (50)_{10}$$

$$(1110101)_2 = (102)_{10}$$

$$(1111)_2 = (17)_{10}$$

$$(10000000)_2 = (128)_{10}$$

$$(11011)_2 = (27)_{10}$$

$$(100001)_2 = (39)_{10}$$

$$(1111111)_2 = (255)_{10}$$

# Conversioni di base: dec2bin

Si moltiplica ripetutamente il numero frazionario decimale per 2, fino ad ottenere una parte decimale nulla o, dato che la condizione potrebbe non verificarsi mai, per un numero prefissato di volte. Le cifre del numero binario sono le parti intere dei prodotti successivi; la cifra più significativa è il risultato della prima moltiplicazione

Esempio: convertire in binario  $(0.25)_{10}$ ,  $(0.21875)_{10}$  e  $(0.45)_{10}$

$$.25 \times 2 = 0.50$$

$$.5 \times 2 = 1.0$$

$$(0.25)_{10} = (0.01)_2$$

$$.21875 \times 2 = 0.4375$$

$$.4375 \times 2 = 0.875$$

$$.875 \times 2 = 1.75$$

$$.75 \times 2 = 1.5$$

$$.5 \times 2 = 1.0$$

$$(0.21875)_{10} = (0.00111)_2$$

$$.45 \times 2 = 0.9$$

$$.90 \times 2 = 1.8$$

$$.80 \times 2 = 1.6$$

$$.60 \times 2 = 1.2$$

$$.20 \times 2 = 0.4$$

$$.40 \times 2 = 0.8$$

$$.80 \times 2 = 1.6 \text{ etc.}$$

$$(0.45)_{10} \approx (0.0111001)_2$$

# Conversioni di base: bin2dec

- Oltre all'espansione esplicita in potenze del 2 – ~~forma polinomiale~~...

$$(101011)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (43)_{10}$$

- si può operare nel modo seguente:
  - si raddoppia il bit più significativo e si aggiunge al secondo bit;
  - si raddoppia la somma e si aggiunge al terzo bit...
  - si continua fino al bit meno significativo

Esempio: convertire in decimale  $(101011)_2$

*bit più significativo*

$\boxed{1} \times 2 = 2$	$+ 0$	↓
$2 \times 2 = 4$	$+ 1$	
$5 \times 2 = 10$	$+ 0$	
$10 \times 2 = 20$	$+ 1$	
$21 \times 2 = 42$	$+ 1$	

$= 43$

$$(101011)_2 = (43)_{10}$$

# Sistema esadecimale

Cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F

La corrispondenza in decimale delle cifre oltre il 9 è

$$\begin{array}{ll} A = (10)_{10} & D = (13)_{10} \\ B = (11)_{10} & E = (14)_{10} \\ C = (12)_{10} & F = (15)_{10} \end{array}$$

Esempio:

$$(3A2F)_{16} = 3 \cdot 16^3 + 10 \cdot 16^2 + 2 \cdot 16^1 + 15 \cdot 16^0 = \\ 34096 + 10256 + 216 + 15 = (14895)_{10}$$

- Una cifra esadecimale corrisponde a 4 bit

0 corrisponde a 4 bit a 0	0000 0	1000 8
	0001 1	1001 9
	0010 2	1010 A
	0011 3	1011 B
	0100 4	1100 C
	0101 5	1101 D
	0110 6	1110 E
	0111 7	1111 F F corrisponde a 4 bit a 1

- Si possono rappresentare numeri binari lunghi con poche cifre(1/4).
- BIN2HEX: raggruppando le cifre binarie in gruppi di 4 e sostituendole con le cifre esadecimali secondo la tabella seguente

# Codifica dei dati: SISTEMA NUMERICO

Decimale	Binario	Ottale	Esadecimale
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

# Sistema esadecimale

## HEX2BIN

- Un numero binario di  $4n$  bit corrisponde a un numero esadecimale di  $n$  cifre

*Esempio: 32 bit corrispondono a 8 cifre esadecimali*

1101	1001	0001	1011	0100	0011	0111	1111
D	9	1	B	4	3	7	F

$(D91B437F)_{16}$

*Esempio: 16 bit corrispondono a 4 cifre esadecimali*

0000	0000	1111	1111
0	0	F	F

$(00FF)_{16}$

# Codifica dei dati: Numeri interi relativi

Per rappresentare numeri con segno in binario, occorre utilizzare 1 bit per definire il segno del numero

Si possono usare 3 tecniche di codifica:

*Modulo e segno*

*Complemento a 1 (in realtà non si utilizza)*

*Complemento a 2*

Rappresentazione modulo e segno

n bit a disposizione per rappresentare un numero intero relativo X

il primo bit (bit più significativo) viene usato per indicare il segno:

Primo bit =0 numero positivo

Primo bit =1 numero negativo

Con n cifre si possono rappresentare i numeri:  $-(2^{n-1} - 1) \leq X \leq +(2^{n-1} - 1)$   
“0” positivo e “0” negativo...

# Codifica dei dati: Numeri interi relativi

- Il bit più significativo rappresenta il segno: 0 per i numeri positivi, 1 per quelli negativi
- Esiste uno zero positivo (00...0) e uno zero negativo (10...0)
- Se si utilizzano n bit si rappresentano tutti i numeri compresi fra  $-(2^{n-1}-1)$  e  $+2^{n-1}-1$

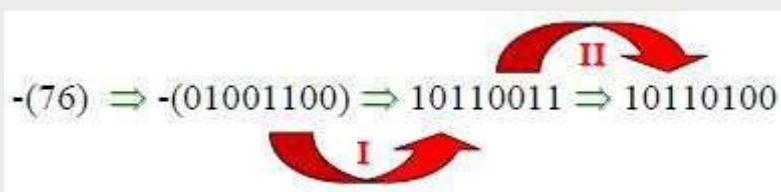
Esempio: con 4 bit si rappresentano i numeri fra  $-7$  ( $-(2^3-1)$ ) e  $+7$  ( $2^3-1$ )

0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4
0101	+5	1101	-5
0110	+6	1110	-6
0111	+7	1111	-7
positivi		negativi	

# Codifica dei dati: Numeri interi relativi

## Rappresentazione in complemento a 2

- Il complemento a 2 di un numero binario  $(N)_2$  a  $n$  cifre è il numero  $2^n - (N)_2$
- il bit più a sinistra indica il segno,
- un numero positivo è individuato dal primo bit uguale a 0 (rappr. modulo e segno)
- per ottenere un numero negativo partendo dalla sua versione positiva, si procede in due passi:
  - Si sostituiscono tutti gli uno con degli zero e viceversa (complemento a 1)
  - Si aggiunge 1 al risultato (somma binaria simile a quella decimale, si riporta 1 se la somma è maggiore di 1) (...vedi slide successiva)



$$\begin{array}{r} \text{ES: } 4 = 0100 \\ 1) \quad 1011 \\ 2) \quad 1 \\ \hline 1100 = 4 \end{array}$$

$$\begin{array}{r} \text{ES: } 6 = 0110 \\ 1) \quad 1001 \\ 2) \quad 1 \\ \hline 1010 = 6 \end{array}$$

- Il campo dei numeri rappresentabili è da  $-2^{n-1}$  a  $+2^{n-1}-1$

Nota: 0111 +7  
1000 -8

# Addizione Binaria

- Consideriamo le 4 possibilità di addizione tra due bit:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$1 + 1 = 0$  con riporto (carry) di 1

- Osservazione....  $(1)_2 + (1)_2 = (10)_2 \dots (1+1=2)_{10}!!$

Esempio

The diagram illustrates binary addition. On the right, a result is shown in a purple box:  
91 +  
90  
---  
181

On the left, a binary addition is performed:  
1 1 1 1      riporti  
0 1 0 1 1 0 1 +  
0 1 0 1 1 0 1 0  
---  
1 0 1 1 0 1 0 1

A purple arrow points from the result box to the left-side addition diagram.

# Codifica dei dati: Numeri interi relativi

- I **numeri positivi** sono rappresentati in modulo e segno
- I **numeri negativi** hanno un 1 nella posizione più significativa e sono rappresentati in complemento a 2
- Lo zero è rappresentato come numero positivo (con una sequenza di n zeri)
- Il campo dei numeri rappresentabili è da  $-2^{n-1}$  a  $+2^{n-1}-1$

*Ad esempio: numeri a 4 cifre*

0000	+0	1000	-8	<i>Nota: 0111 +7 1000 -8</i>
0001	+1	1001	-7	
0010	+2	1010	-6	
0011	+3	1011	-5	
0100	+4	1100	-4	
0101	+5	1101	-3	
0110	+6	1110	-2	
0111	+7	1111	-1	

# Sottrazione Binaria

- Consideriamo le 4 possibilità di sottrazione tra due bit:

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$0 - 1 = 1$  con prestito (borrow) di 1 dalla cifra precedente a sinistra

Esempio

$$\begin{array}{r} 11001 - \\ 101 \hline 10100 \end{array}$$



$$\begin{array}{r} 25 - \\ 5 \hline 20 \end{array}$$

- Il calcolo della sottrazione può divenire complicato: quando si ha una richiesta sulla cifra precedente a sinistra, che è uno 0, l'operazione si propaga a sinistra fino alla prima cifra ad 1 del sottraendo

*Utilizzando la rappresentazione in complemento a 2, addizione e sottrazione sono trattate come una unica operazione*

# Sottrazione Binaria

- Utilizzando la rappresentazione in complemento a 2, addizione e sottrazione sono trattate come una unica operazione

$$N_1 - N_2 = N_1 + (2^n - N_2) - 2^n \quad \text{si trascura il bit } n+1$$

complemento a 2 di  $N_2$  ( $-N_2$ )

- ① Si calcola il complemento a 2 di  $N_2$
- ② Si somma  $N_1$  con il complemento a 2 di  $N_2$
- ③ Si trascura il bit più significativo del risultato

Esempio:  $(010001)_2 - (000101)_2 = (17)_{10} - (5)_{10}$

$$\begin{array}{r}
 010001 \\
 + 111011 \\
 \hline
 1001100 \rightarrow (12)_{10}
 \end{array}$$

# OVERFLOW

Si considerino due numeri binari ad n bit

Si ha overflow quando il risultato di un'operazione tra i due numeri non è rappresentabile correttamente con n bit (eccede il limite superiore o inferiore di rappresentabilità con gli n bit)

Esempio: 5 bit — [-16,+15], rappresentazione compl a 2

$$\begin{array}{r}
 14 + \quad 01110 + \\
 \underline{10} \qquad \underline{01010} \\
 24 \qquad \boxed{11000} \rightarrow -8
 \end{array}$$

$$\begin{array}{r}
 -8 + \quad 11000 + \\
 \underline{-10} \qquad \underline{10110} \\
 -18 \qquad \boxed{101110} \rightarrow +14
 \end{array}$$

- Per evitare l'overflow occorre aumentare il numero di bit utilizzati per rappresentare gli operandi (operazione non possibile se il calcolatore è già stato progettato)
- In caso di overflow la circuteria hardware lo rileva e segnala un errore (bit di overflow nella PSW del processore)

# Numeri razionali

## Notazione in virgola mobile

- Ogni numero può essere rappresentato come prodotto delle sue cifre significative per una potenza della base il cui valore dipende dalla posizione della virgola nel numero di partenza

Es.  $0.0012 = 12 \times 10^{-4}$

$$35.5 = 355 \times 10^{-1}$$

- La IEEE ha previsto uno standard (IEEE 754) per la rappresentazione in virgola mobile
  - singola precisione* ( $32 \text{ bit} = 4 \text{ byte}$ )
  - doppia precisione* ( $64 \text{ bit} = 8 \text{ byte}$ )

Un qualsiasi numero può essere espresso come:  $N = \pm M \times r^E$

Dove  $M$  =Mantissa – CIFRE RAPPRESENTATIVE DEL NUMERO

$r$  =Base del sistema di numerazione

$E$  =Esponente della base

Oss.: rappresentazione non univoca:  $0.0012 = 0.12 \times 10^{-2} = 12 \times 10^{-4}$  NB:  
la mantissa contiene sempre solo le cifre significative del numero

# Numeri razionali

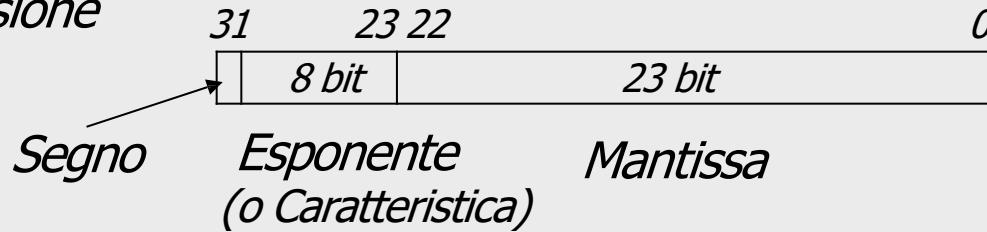
## Notazione in virgola mobile

Un qualsiasi numero può essere espresso come:

$$N = \pm M \times r^E$$

La base può essere omessa dato che nel calcolatore è sempre 2

*Singola precisione*



Considerazioni sulla mantissa:

- M intero
- $1/r \leq M \leq 1$  MANTISSA NORMALIZZATA

Considerazioni sull'esponente:

- Numero intero relativo in complemento a 2

# Numeri razionali

## Notazione in virgola mobile

Un qualsiasi numero può essere espresso come:

$$N = \pm M \times r^E$$

*Esempio*



10111010

- Segno=1  $\rightarrow$  numero negativo
- Esponente=011 in complemento a 2 =+3
- Mantissa normalizzata=0.1010 = $2^{-1} + 2^{-3} = 1/2 + 1/8 = 5/8$

$$N = -5/8 \times 2^3 = -5$$

Esercizio:

- Determinare il più grande numero rappresentabile
- Determinare il più piccolo numero (vicino allo zero) rappresentabile

# Numeri razionali

## Notazione in virgola mobile

(+7) in base 10

- numero positivo  $\rightarrow$  Segno=0
- 7 mantissa su 4 bit  $=0111 \times 2^0 = 0.111 \times 2^3$ , mantissa=1110
- Esponente=011 in complemento a 2 =+3

$$N=0\ 011\ 1110$$

Numero più piccolo (più vicino allo 0)

- numero positivo  $\rightarrow$  Segno=0
- Esponente =-4  $\rightarrow$  100
- Mantissa =0.0001

$$N=0\ 100\ 0001$$

$$01000001$$

- Segno=0  $\rightarrow$  numero positivo
- Esponente =-4
- Mantissa normalizzata=0.0001  $=2^{-4}=1/16$

$$N=+1/16 \times 2^{-4} = +1/256$$

Secondo numero più piccolo

- numero positivo  $\rightarrow$  Segno=0
- Esponente =-4  $\rightarrow$  100
- Mantissa =0.0010

$$N=0\ 100\ 0010$$

$$01000010$$

- Segno=0  $\rightarrow$  numero positivo
- Esponente =-4
- Mantissa normalizzata=0.0010  $=2^{-3}=1/8$

$$N=+1/8 \times 2^{-4} = +1/128$$

$$Distanza=1/128=0.0078125$$

# Numeri razionali

## Notazione in virgola mobile

Numero più grande positivo

- numero positivo  $\rightarrow$  Segno=0
- Esponente =+3  $\rightarrow$  011
- Mantissa =1111

$$N=0\ 011\ 1111$$

00111111

- Segno=0  $\rightarrow$  numero positivo
- Esponente=011 in complemento a 2 =+3
- Mantissa normalizzata=1111 = $2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$   
 $=15/16$

$$N=+15/16 \times 2^3 = 7,5 \text{ (si provi a rappresentare } 8!!)$$

Secondo numero più grande positivo

- numero positivo  $\rightarrow$  Segno=0
- Esponente =+3  $\rightarrow$  011
- Mantissa =1110

$$N=0\ 011\ 1110$$

00111110

- Segno=0  $\rightarrow$  numero positivo
- Esponente=011 in complemento a 2 =+3
- Mantissa normalizzata=1110 = $2^{-1} + 2^{-2} + 2^{-3}$   
 $=7/8$

$$N=+7/8 \times 2^3 = 7 \text{ (si provi a rappresentare } 7.25!!)$$

Distanza=0.5

# Numeri razionali

## Notazione in virgola mobile

(-5.5) in base 10

- numero negativo -> Segno=1
- 5.5 mantissa su 4 bit =  $101.1 \times 2^0 = 0.1011 \times 2^3$ , mantissa=1011  
 $.5 \times 2 = 1.0$
- Esponente=011 in complemento a 2 = +3

$N=1\ 011\ 1011$

(-5.75) in base 10

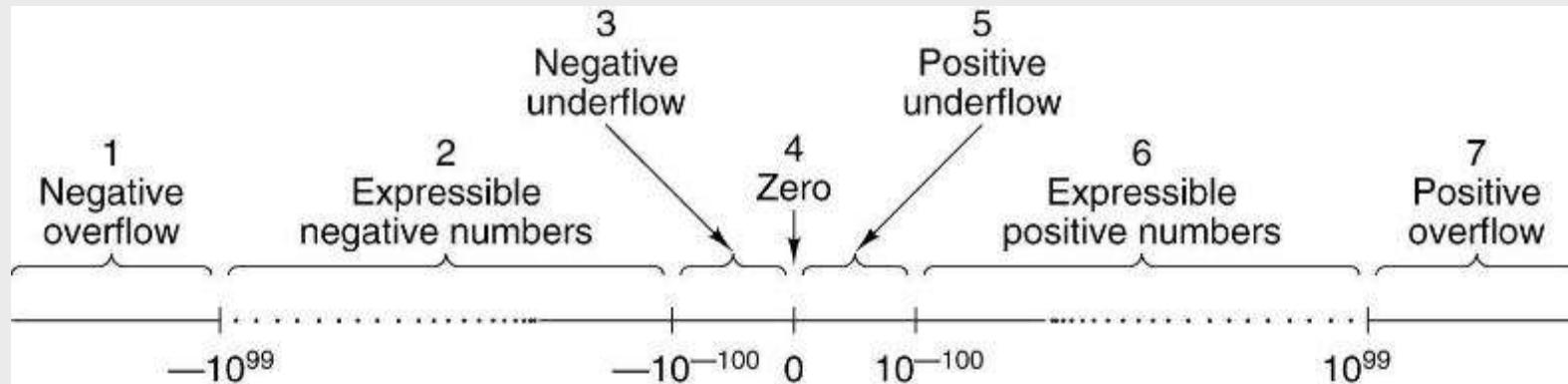
- numero negativo -> Segno=1
- $5.75 = 101.11 \times 2^0 = 0.10111 \times 2^3$ , mantissa=1011 (mantissa su 4 bit!)       $.75 \times 2 = 1.50$   
 $.5 \times 2 = 1.0$
- Esponente=011 in complemento a 2 = +3

$N=1\ 011\ 1011 = -5.5!!!$  ERRORE DI APPROXIMAZIONE

# Numeri razionali

## Notazione in virgola mobile

Characteristics of IEEE floating-point numbers.



Il valore è

$$\begin{aligned} & (-1)^S \cdot 1.M \cdot 2^{E-127} \quad \text{se } E \neq 0 \\ & (-1)^S \cdot 0.M \cdot 2^{-127} \quad \text{se } E = 0 \end{aligned}$$

Normalized	$\pm$	$0 < \text{Exp} < \text{Max}$	Any bit pattern
Denormalized	$\pm$	0	Any nonzero bit pattern
Zero	$\pm$	0	0
Infinity	$\pm$	1 1 1...1	0
Not a number	$\pm$	1 1 1...1	Any nonzero bit pattern

Sign bit

# Numeri razionali

## Notazione in virgola mobile

Characteristics of IEEE floating-point numbers.

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	–126 to +127	–1022 to +1023
Smallest normalized number	$2^{-126}$	$2^{-1022}$
Largest normalized number	approx. $2^{128}$	approx. $2^{1024}$
Decimal range	approx. $10^{-38}$ to $10^{38}$	approx. $10^{-308}$ to $10^{308}$
Smallest denormalized number	approx. $10^{-45}$	approx. $10^{-324}$

# Aritmetica degli elaboratori

- Rango finito dei numeri rappresentabili
  - ❖ Qualunque sia la codifica, esistono sempre il più grande ed il più piccolo numero
  - ❖ I limiti inferiore e superiore dipendono sia dal tipo di codifica, sia dal numero di bit utilizzati
  - ❖ Se il risultato di un'operazione non appartiene al rango dei numeri rappresentabili, si dice che si è verificato un overflow (un underflow, più precisamente, se il risultato è più piccolo del più piccolo numero rappresentabile)
- Precisione finita dei numeri
  - ❖ La precisione della rappresentazione di un numero frazionario è una misura di quanto essa corrisponda al numero che deve essere rappresentato
  - ❖ Negli elaboratori, i numeri frazionari sono rappresentati in virgola mobile (floating point), utilizzando un numero finito di bit
  - ❖ È plausibile che un numero reale non ammetta una rappresentazione finita,
  - ❖ Negli elaboratori si rappresentano soltanto numeri razionali (fino ad una data precisione)

# Codifica dei dati: Rappresentazione dei caratteri

Ciascun carattere alfanumerico, di punteggiatura o di controllo che compone il testo deve essere rappresentato nei termini di codice binario

CODICE ASCII (American Standard Code for Information Interchange) o ISO-646  
utilizza 8bit di cui uno di controllo (di parità): 7bit=128 configurazioni

i caratteri sono distinti in:

- **caratteri di comando (codici di trasmissione o di controllo della stampa)**  
Line Feed (LF) 00001010 Escape (Esc) 00011011
- **caratteri alfanumerici: lettere dell'alfabeto maiuscole e minuscole e cifre numeriche**  
A 01000001 B 01000010 C 01000011  
a 01100001 b 01100010 c 01100011
- **Segni: simboli di punteggiatura e operatori aritmetico-logici**  
; 00111011 " 00100010 [ 01011011  
\* 00101010 /00101111 >00111110

# Codifica dei dati:

## Rappresentazione dei caratteri

CODICE ASCII

NUL	00000000	blank	00100000	@	01000000	'	01100000
SOH	00000001	i	00100001	A	01000001	a	01100001
STX	00000010	"	00100010	B	01000010	b	01100010
ETX	00000011	#	00100011	C	01000011	c	01100011
EOT	00000100	\$	00100100	D	01000100	d	01100100
ENQ	00000101	%	00100101	E	01000101	e	01100101
ACK	00000110	&	00100110	F	01000110	f	01100110
BEL	00000111	'	00100111	G	01000111	g	01100111
BS	00001000	(	00101000	H	01001000	h	01101000
HT	00001001	)	00101001	I	01001001	i	01101001
LF	00001010	*	00101010	J	01001010	l	01101010
VT	00001011	+	00101011	K	01001011	j	01101011
FF	00001100	,	00101100	L	01001100	k	01101100
CR	00001101	-	00101101	M	01001101	m	01101101
SO	00001110	.	00101110	N	01001110	n	01101110
SI	00001111	/	00101111	O	01001111	o	01101111
DLE	00010000	0	00110000	P	01010000	p	01110000
DC1	00010001	1	00110001	Q	01010001	q	01110001
DC2	00010010	2	00110010	R	01010010	r	01110010
DC3	00010011	3	00110011	S	01010011	s	01110011
DC4	00010100	4	00110100	T	01010100	t	01110100
NAK	00010101	5	00110101	U	01010101	u	01110101
SYN	00010110	6	00110110	V	01010110	v	01110110
ETB	00010111	7	00110111	W	01010111	w	01110111
CAN	00011000	8	00111000	X	01011000	x	01111000
EM	00011001	9	00111001	Y	01011001	y	01111001
SUB	00011010	:	00111010	Z	01011010	z	01111010
ESC	00011011	:	00111011	[	01011011	{	01111011
FS	00011100	<	00111100	\	01011100	:	01111100
GS	00011101	=	00111101	]	01011101	}	01111101
RS	00011110	>	00111110	←	01011110	≈	01111110
US	00011111	?	00111111	↑	01011111	DEL	01111111

01000011 01101001 01100001 01101111  
 C i a o

# Codifica dei dati: Rappresentazione dei caratteri

ASCII sviluppato per lingue anglosassoni, quindi non contiene codici per molti caratteri di altre lingue

UNICODE:sistema di codifica che assegna un numero univoco ad ogni carattere usato per la scrittura di testi, in maniera indipendente dalla lingua, dalla piattaforma informatica e dal programma utilizzato.

- Codice a 16 bit che può essere usato per codificare  $2^{16}$  simboli diversi (codificando i caratteri usati in quasi tutte le lingue vive e in alcune lingue morte, nonché simboli matematici e chimici, cartografici, l'alfabeto Braille, ecc.)
- PRO:
  - Universalità
- Problemi:
  - Non tutti gli editori lo trattano
  - Dimensioni dei file raddoppiano rispetto all'ASCII

*L'ultima versione dello standard definisce 3 tipi diversi di codifica che permettono agli stessi dati di essere trasmessi in byte (8 bit – **UTF-8**), word (16 bit – **UTF-16**) o double word (32 bit – **UTF-32**).*

# Rappresentazione digitale dei suoni

## CAMPIONAMENTO:

processo di conversione di un segnale tempo-continuo in un segnale tempo-discreto,

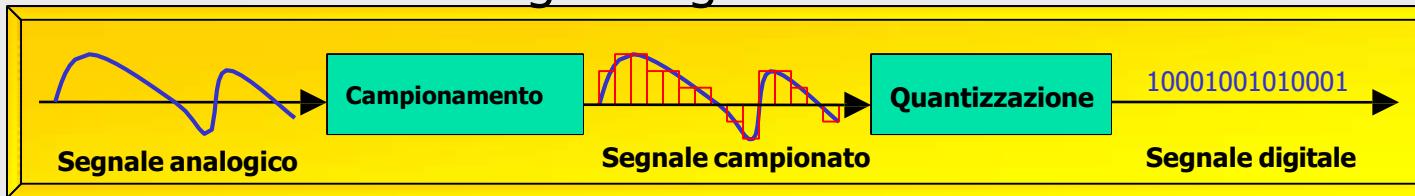
L'ampiezza del segnale continuo viene considerata a intervalli di tempo regolari ( $T$  - periodo di campionamento).

## QUANTIZZAZIONE:

processo di conversione di un segnale a valori continui in uno a valori discreti.

Più è alto il numero di bit utilizzati nella quantizzazione e minore è l'errore che si commette (errore di quantizzazione), cioè si riduce la distanza media tra il valore campionato (continuo) e il corrispondente valore quantizzato

## Analog to Digital Converter



# Algoritmi di compressione per file audio

## Codifica della voce:

Pulse Code Modulation (PCM)

- Trasmissioni telefoniche
- Frequenza campionamento: 8KHz
- Quantizzazione su 8 bit
- Velocità sul canale trasmissivo  $\geq 64\text{Kbps}$

## Codifica della musica

CD:

- Musica campionata a 44100 Hz ( $>2 \times 20000\text{Hz}$  massima freq orecchio umano)
- Campioni per sinistra e destra (stereo)
- 16 bit per campione per canale
- Totale:  $(44100 \times 16 \times 2) = 1\ 411\ 200$  bits x secondo = (0.17MB)
- Quanto spazio occupa una canzone di 4 minuti??

MP3 (compressione lossy): Sfrutta conoscenza dei limiti dell'udito umano per ridurre la quantita' di informazione da memorizzare:

- Esclusi suoni che l'orecchio percepisce poco (un suono ad una frequenza viene percepito meglio di suoni a frequenze adiacenti a più basso volume)
- Quando c'e' un suono particolarmente rumoroso, non registrare gli altri suoni
- Fattore di riduzione: anche 10 volte (= 3-4M per canzone)

# Codifica delle immagini

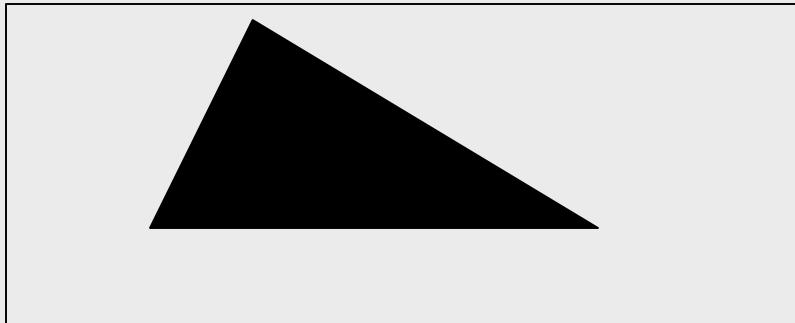
- Le operazioni di campionamento e quantizzazione si applicano nello spazio (2D) invece che nel tempo.
- Il campionamento consiste nel dividere l'immagine in sottoinsiemi regolari (pixel = picture element), per ognuno dei quali si dovrà prelevare un campione che si considera rappresentativo di tutto il sottoinsieme.
- La quantizzazione è la codifica del colore associato a ogni pixel: i più recenti formati utilizzano 32 bit (4 byte) per pixel di cui:
  - 8 bit per ognuna delle tre componenti fondamentali (RGB: red, green, blue)
  - altri 8 per gestire le trasparenze.
  - L'immagine è una mappa di bit (bitmap .bmp)

Esempio di dimensioni:

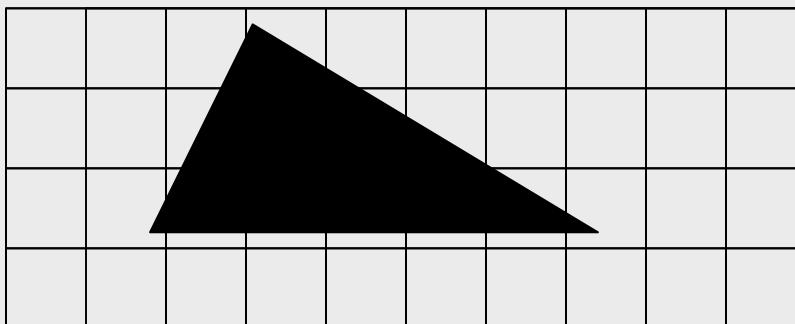
Macchina fotografica a 10Mpixel: 3872 x 2592 pixel

3872 x 2592 x 4B=38,29MB

# Codifica delle immagini

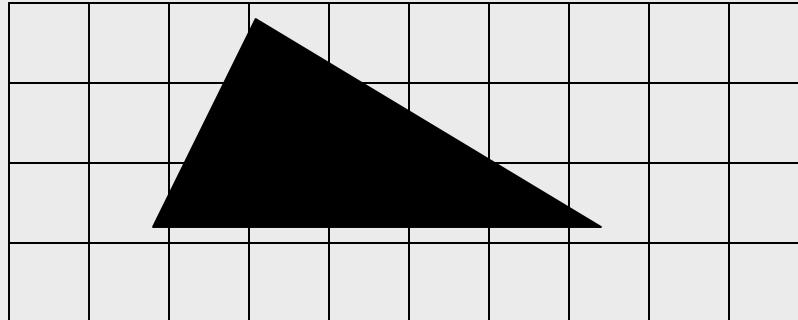


Consideriamo una griglia formata da righe orizzontali e verticali a distanza costante



- Ogni quadratino prende il nome di **pixel** (picture element) e può essere codificato in binario secondo la seguente convenzione:
  - “0” viene utilizzato per la codifica di un pixel corrispondente ad un quadratino in cui il bianco è predominante
  - “1” viene utilizzato per la codifica di un pixel corrispondente ad un quadratino in cui il nero è predominante

# Codifica delle immagini



0	0	0	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0

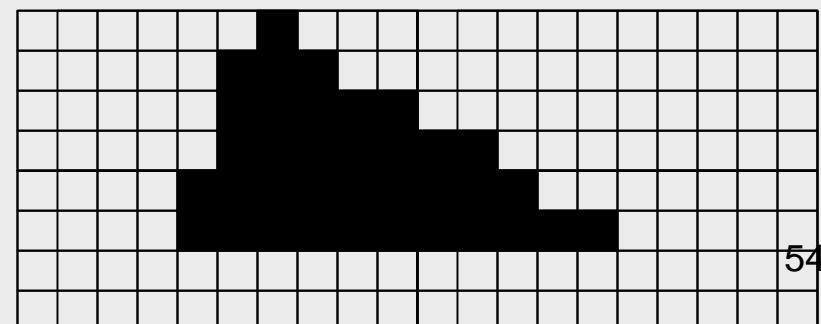
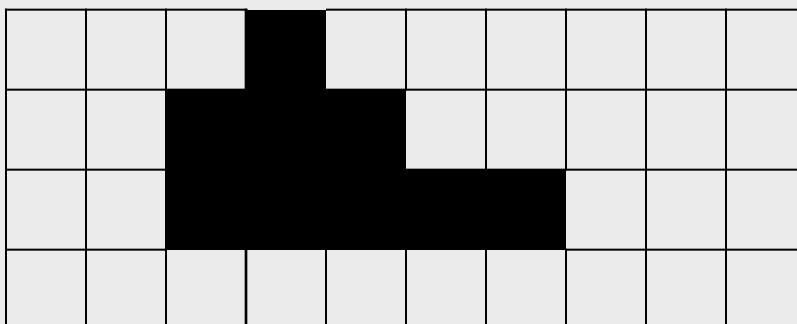
# Codifica delle immagini

0	0	0	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0

Poiché una sequenza di bit è lineare, è necessario definire convenzioni per ordinare la griglia dei pixel in una sequenza.

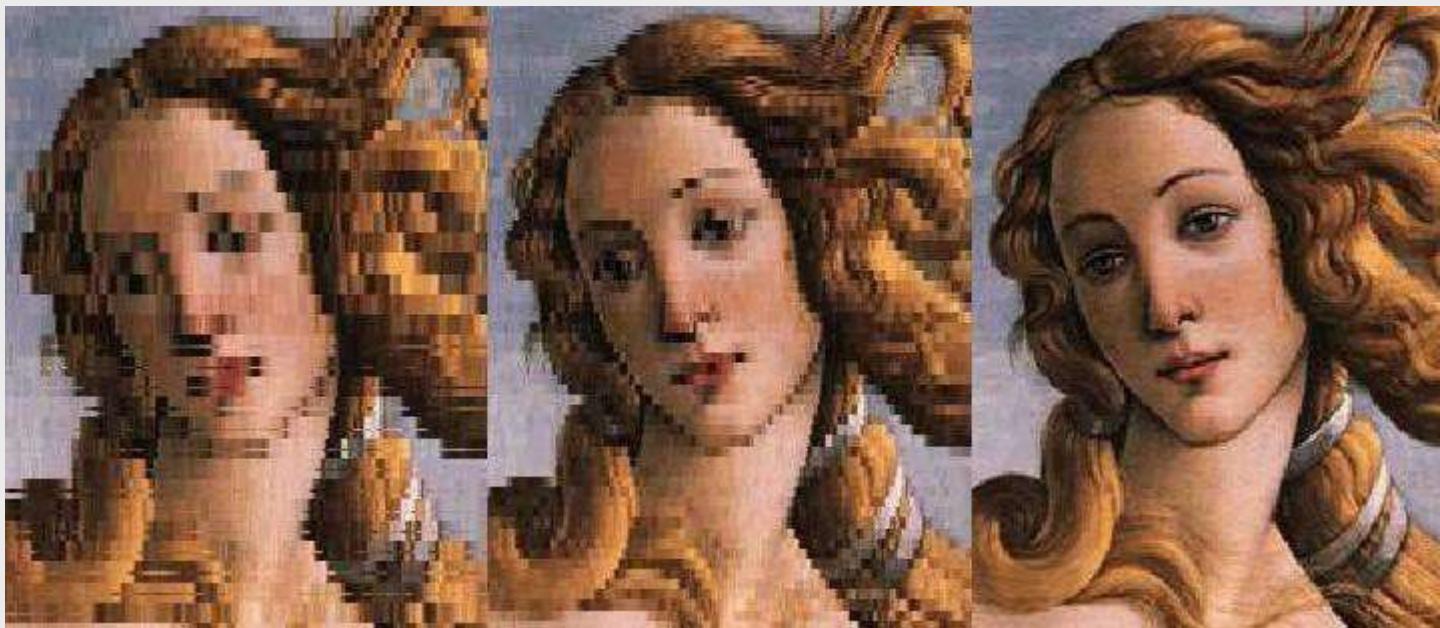
Non sempre il cortorno della figura coincide con le linee della griglia. Quella che si ottiene nella codifica è un'approssimazione della figura originaria

La rappresentazione sarà più fedele all'aumentare del numero di pixel, ossia al diminuire delle dimensioni dei quadratini della griglia in cui è suddivisa l'immagine



# La risoluzione

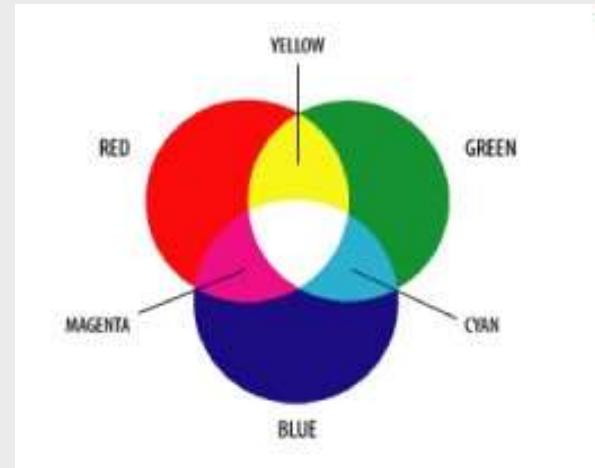
- dpi= dot per inch
  - Numero di pixel presenti su una linea lunga 2,54 cm



# Spazio dei colori

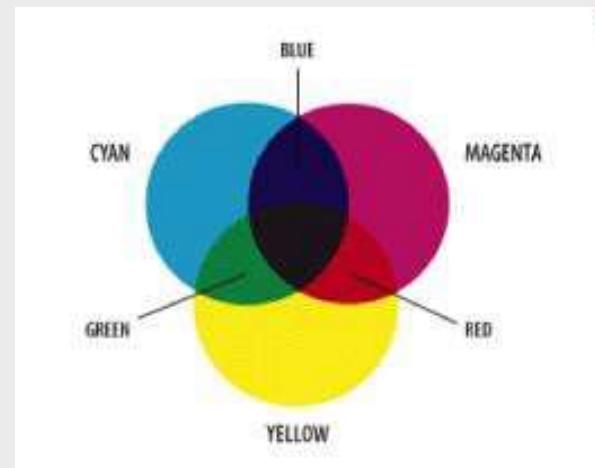
## RGB

Sistema additivo: i colori ROSSO, VERDE e BLU (RGB) si mescolano tra loro creando colori di luminosità maggiore. Viene utilizzato quando si vogliono mischiare fasci di luce colorata, ad esempio sugli schermi e sui monitor.



## CMY

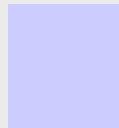
Sistema sottrattivo: CIANO, MAGENTA e GIALLO (CMY). I colori che vengono generati hanno meno luminosità. Questo sistema si usa nel caso in cui si mischiano sostanze colorate (che riflettono la luce): inchiostri, tempere etc.



# Rappresentazione dei colori

- Se si fa corrispondere a ogni pixel un byte, potremo differenziare 256 colori
- Necessità di una tabella di codifica dei colori...

Ad es: 00101101 →



- L'immagine viene fatta corrispondere a una matrice
- Ogni pixel dell'immagine viene codificato dalla sequenza di '0' e '1' associato al suo colore dalla tabella di codifica dei colori utilizzata

La dimensione dell'immagine è data dal numero di pixel che costituiscono la base per il numero di pixel che costituiscono l'altezza.

La profondità dell'immagine è il numero di bit che utilizziamo per rappresentare il colore di un singolo pixel dell'immagine (numero di colori disponibili)

Numero di bit per immagine = dimensione x profondità

# Algoritmi di compressione delle immagini

Formato BITMAP (.bmp)

Esempio di dimensioni:

Macchina fotografica a 10Mpixel: 3872 x 2592 pixel

3872 x 2592 x 4B=38,29MB

Per l'occhio umano lievi cambiamenti di colore in punti contigui sono poco visibili e quindi possono essere eliminati (su questo principio si basano, per esempio, i formati GIF e JPEG);

Formato GIF: riduce l'occupazione di un'immagine limitando il numero di colori che compaiono in essa -> vengono scelti quelli più frequenti, alcune sfumature vengono perse e sostituite dalle sfumature più vicine fra quelle mantenute

Più si limita il numero di colori più l'immagine occuperà meno spazio; il numero può andare da un minimo di 2 ad un massimo di 256.

Adatto ad immagini geometriche.

Formato JPEG riduce l'occupazione di un'immagine diminuendo la qualità di visualizzazione -> consente di usare tutta la gamma RGB.

1. Si sfrutta la diversa sensibilità dell'occhio a diversi livelli di luminosità

2. In aree 8x8 pixel si individuano solo punti significativi per la percezione umana

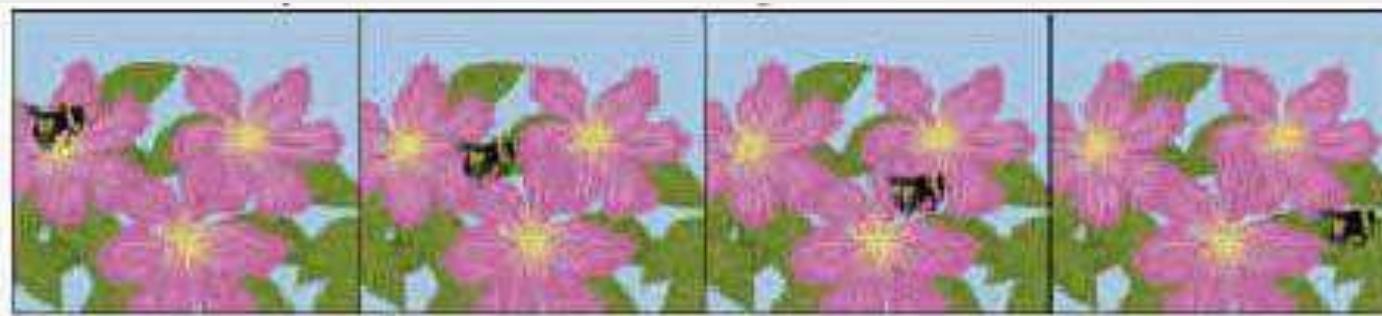
Adatto ad immagini di tipo fotografico o in generale ad immagini dove un limite sul numero di colori produrrebbe differenze troppo significative

# Informazione Multimediale

- **Filmato:** successione di fotogrammi (frame) accompagnata da una flusso audio  
Il movimento è rappresentato in modo discreto: con un numero sufficientemente alto di fotogrammi fissi (24-30 al secondo) l'occhio umano percepisce il movimento come un continuo (l'occhio umano “scatta” 10 foto al secondo)  
*frame per second fps*
- Come codificare un video?  
  
IDEA..
  - Ogni fotogramma è un’immagine: codifica delle immagini
  - Audio: codifica dell’audio.
- PROBLEMA: per codificare un breve filmato servono moltissimi bit!
  - Es: codificando separatamente ogni fotogramma come immagine fissa, lo spazio di memoria richiesto sarebbe enorme (650 MB per un minuto di flusso video)
- SOLUZIONE: codificare solo delle ‘differenze’ fra un fotogramma e l’altro (MPEG)

# Algoritmi di compressione video

MPEG: immagini successive hanno parti uguali e quindi di ogni immagine possono essere memorizzate solo le differenze con l'immagine precedente (come fanno, per esempio, vari standard MPEG)



Video non compresso



Video compresso

# Informazione Multimediale

La dimensione del filmato dipende da almeno cinque fattori:

- **Lunghezza del filmato** (in termini di [s])
- **Risoluzione grafica** (quanto più fitta è la griglia che usiamo per digitalizzare i singoli fotogrammi)
  - Una bassa risoluzione rende il filmato “quadrettato e indistinto”
- **L'ampiezza della ‘palette’** di colori utilizzata (ossia il numero dei colori)
  - Una palette troppo ristretta rende i colori poco realistici
- **Il numero di fotogrammi** (o frame) per secondo
  - Un numero limitato di fotogrammi fa apparire la sequenza di immagini poco fluida
- **La qualità dell'audio**

# Algoritmi di compressione video

Quicktime (Apple)

Avi (Microsoft)

Real Video: usato in applicazioni di streaming video per la fruizione in tempo reale di un video

- i pacchetti audio-video che rappresentano il filmato vengono trasmessi continuamente l'uno dopo l'altro e vengono visualizzati man mano che arrivano nell'ordine;
- poi vengono buttati via

streaming video vs. classico download

# Livello Logico-Digitale

*Prof. Ing. Donato Impedovo*

Applicazioni basate su regole booleane:

- Regole di funzionamento di centraline di controllo  
(ad esempio dei semafori..)
- Regole di movimentazione apparati meccanici

# Algebra di Boole

Considerato un insieme  $S$  di elementi:

- Ognuno degli elementi può assumere solo uno dei due valori: 0, 1
- Nel quale esiste almeno una coppia di elementi  $X, Y \in S \ni X \neq Y$
- Per i quali sia definita una legge di composizione, detta **somma logica** (+), tale che  $Z = X + Y$ , con  $X, Y, Z \in S$
- Per i quali sia definita una legge di composizione, detta **prodotto logico** ( $\cdot$ ), tale che  $Z = X \cdot Y$ , con  $X, Y, Z \in S$
- Che contenga un elemento neutro rispetto alla somma (o zero), indicato dal simbolo 0, tale che  $X + 0 = X$
- Che contenga un elemento neutro rispetto al prodotto (o unità), indicato dal simbolo 1, tale che  $X \cdot 1 = X$

Continua...

# Algebra di Boole

Considerato un insieme  $S$  di elementi:

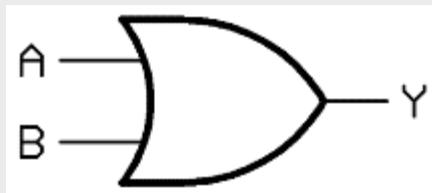
...continua

- Per il quale valgono le proprietà commutative delle operazioni (+ e  $\cdot$ ), tali che  $X + Y = Y + X$  e  $X \cdot Y = Y \cdot X$
- Per il quale valgono le proprietà distributive della operazione  $+$  rispetto alla operazione  $\cdot$  e della operazione  $\cdot$  rispetto alla operazione  $+$ , cioè  $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$ ,  $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
- Preso  $X \in S \Rightarrow \exists (\underline{X} \in S) \ni (X + \underline{X} = 1) \text{ e } (X \cdot \underline{X} = 0)$   
 $\underline{X}$  è detto complemento di  $X$

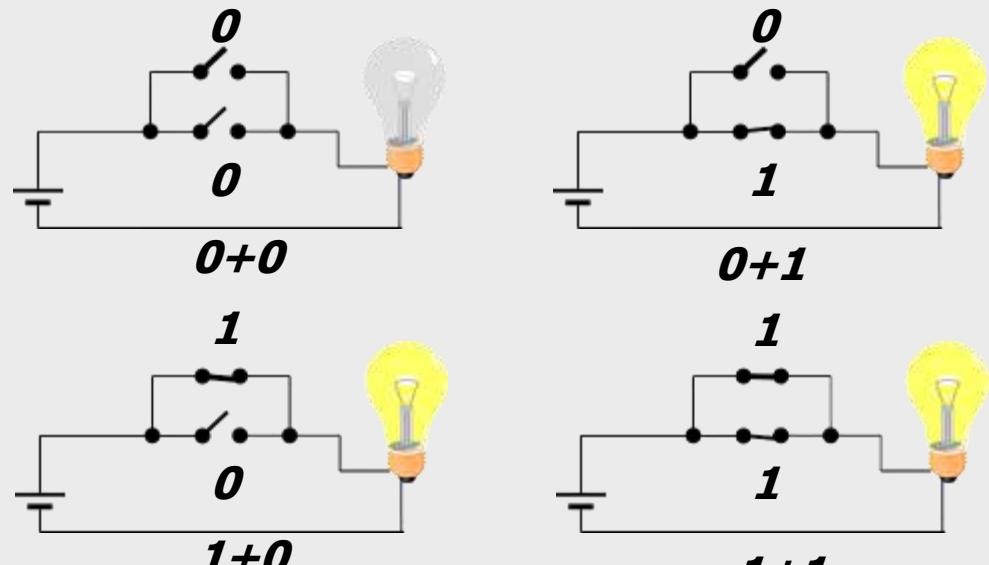
Le proprietà suddette definiscono su  $S$  una struttura algebrica comunemente detta Algebra di Boole.

# Operatore OR

- **Somma logica** (OR): il valore della somma logica è 1 se il valore di almeno uno degli operandi è 1.



A	B	$A+B$ $A \cup B$ $A \text{ OR } B$
0	0	0
0	1	1
1	0	1
1	1	1

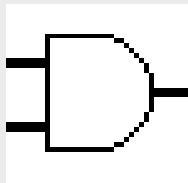


Date n variabili booleane indipendenti la loro somma logica (OR) è

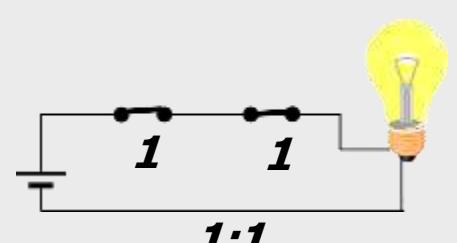
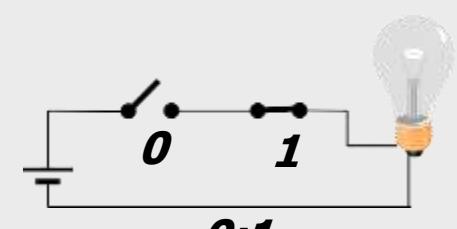
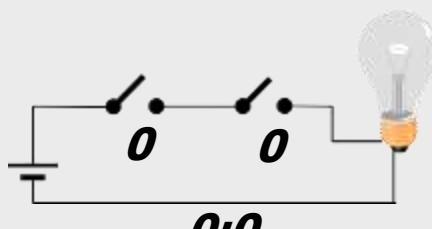
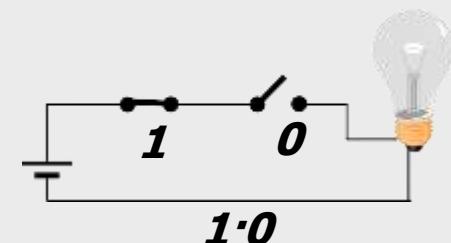
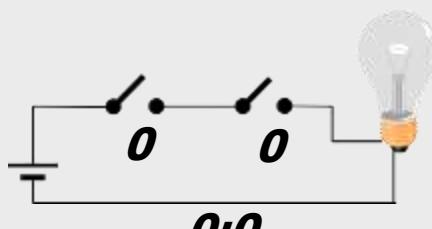
$$x_1 + x_2 + \dots + x_n = \begin{cases} 1 & \text{se almeno una } x_i \text{ vale 1} \\ 0 & \text{se } x_1 = x_2 = \dots = x_n = 0 \end{cases}$$

# Operatore AND

- **Prodotto logico** (AND): il valore del prodotto logico è 1 se il valore di tutti gli operandi è 1.



X	Y	$X \cdot Y$ $X \cap Y$ $X \text{.AND.} Y$
0	0	0
0	1	0
1	0	0
1	1	1



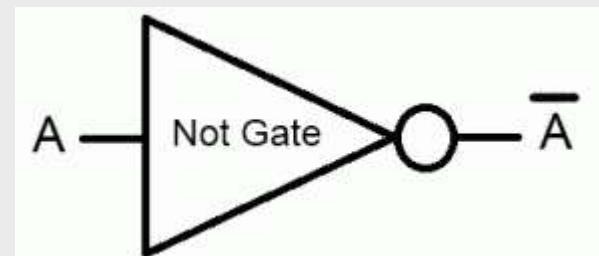
Date n variabili booleane indipendenti il loro prodotto logico (AND) è

$$x_1 \cdot x_2 \cdot \dots \cdot x_n = \begin{cases} 0 & \text{se almeno una } x_i \text{ vale 0} \\ 1 & \text{se } x_1 = x_2 = \dots = x_n = 1 \end{cases}$$

# Operatore NOT

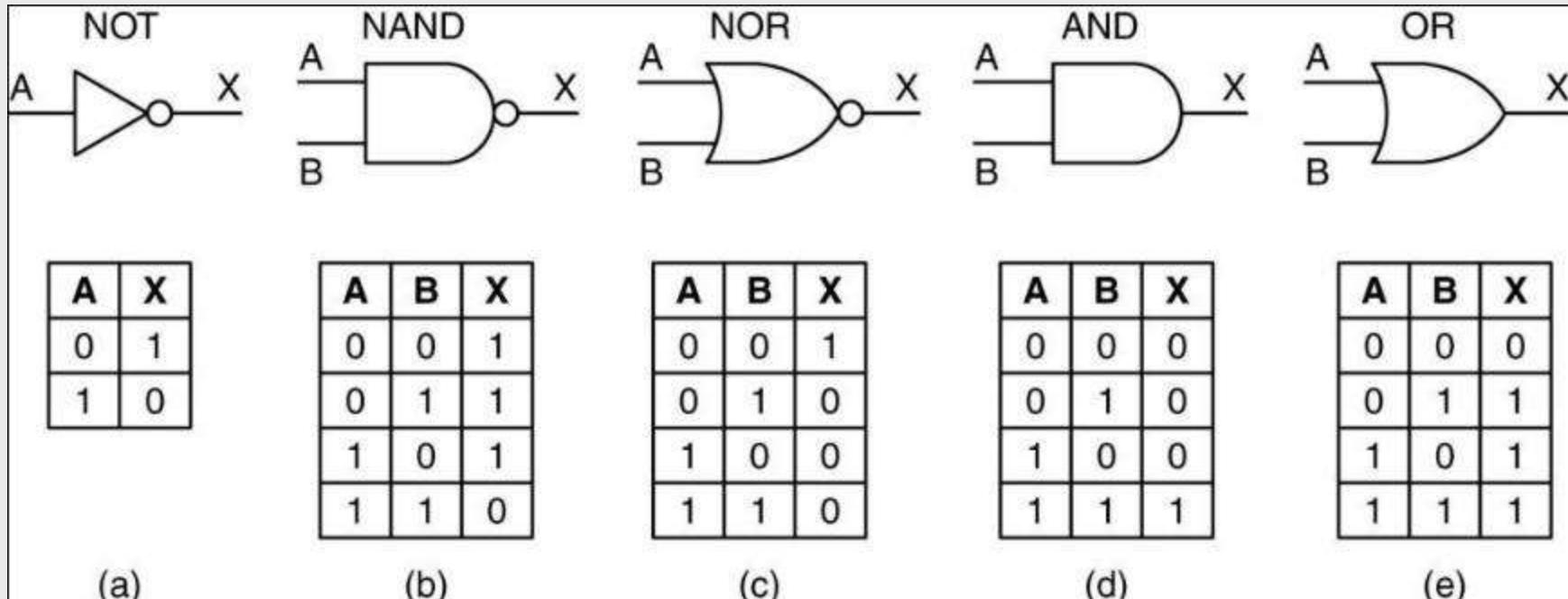
- Operatore di **negazione** (NOT): l'operatore inverte il valore della costante su cui opera. OPERAZIONE UNARIA.

A	$\underline{A}$ !A NOT(A)
0	1
1	0



L'elemento  $\underline{A} = \text{NOT}(A)$  viene detto complemento di A. Il complemento è unico.

# PORTE



# Funzioni logiche

- Una variabile  $y$  è una funzione delle  $n$  variabili indipendenti  $x_1, x_2, \dots, x_n$ , se esiste un criterio che fa corrispondere in modo univoco ad ognuna delle  $2^n$  configurazioni delle  $x$  un valore di  $y$

$$y = F(x_1, x_2, \dots, x_n)$$

- Una rappresentazione esplicita di una funzione è la tabella di verità, in cui si elencano tutte le possibili combinazioni di  $x_1, \dots, x_n$ , con associato il valore di  $y$
- Per costruire la tabella della verità di un'espressione booleana occorre:
  - semplificare, se possibile, l'espressione mediante i teoremi dell'algebra booleana
  - calcolare i termini parziali della funzione riducendoli alle operazioni fondamentali

$$F(a, b, c) = a \cdot b + c$$

a	b	c	$a \cdot b$	$a \cdot b + c$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

# Proprietà

Idempotenza

$$A \text{.or.} A = A$$

$$A \text{.and.} A = A$$

**Proprietà dell'idempotenza:**

$$A + A = A$$

$$A \cdot A = A$$

Infatti:

A	A	A + A
0	0	0
1	1	1

A	A	A · A
0	0	0
1	1	1

# Proprietà

Proprietà dell'elemento neutro per OR e AND:

$$A + 0 = A$$

$$A \cdot 1 = A$$

Infatti:

A	0	$A + 0$
0	0	0
1	0	1

A	1	$A \cdot 1$
0	1	0
1	1	1

# Proprietà

Proprietà dell'elemento nullo per OR e AND:

$$A + 1 = 1$$

$$A \cdot 0 = 0$$

Infatti:

A	1	A + 1
0	1	1
1	1	1

A	0	A · 0
0	0	0
1	0	0

# Proprietà

Proprietà dell'elemento complementare per OR e AND:

$$A + \bar{A} = 1$$

$$A \cdot \bar{A} = 0$$

Infatti:

A	$\bar{A}$	$A + \bar{A}$
0	1	1
1	0	1

A	$\bar{A}$	$A \cdot \bar{A}$
0	1	0
1	0	0

# Proprietà

OR e AND godono delle seguenti proprietà:

**Commutativa**       $A+B=B+A$        $A \cdot B = B \cdot A$

**Associativa**       $A + (B + C) = (A + B) + C = A + B + C$   
 $A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$

**Assorbimento**       $A \cdot (A + B) = A \quad (1)$        $A + (A \cdot B) = A \quad (2)$

Dimostrazione (1):  $A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A$

Dimostrazione (2):  $A + (A \cdot B) = (A + A) \cdot (A + B) = A \cdot (A + B) \dots$  vedi sopra

**Distributiva**       $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$   
 $A + (B \cdot C) = (A + B) \cdot (A + C) \quad (*)$

Dimostrazione di (\*):  $(A + B) \cdot (A + C) = A \cdot A + A \cdot C + B \cdot A + B \cdot C =$   
 $= A + A \cdot C + B \cdot A + B \cdot C = A(1 + C) + B \cdot A + B \cdot C =$   
 $= A + B \cdot A + B \cdot C = A(1 + B) + B \cdot C = A + B \cdot C$

# Proprietà

DE MORGAN

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

A	B	$\overline{A}$	$\overline{B}$	A+B	$\overline{A+B}$	AB	$\overline{AB}$	$\overline{A}\overline{B}$	$\overline{A}\overline{B}$	$\overline{A+B}$
0	0	1	1	0	1	0	1	1	1	1
0	1	1	0	1	1	0	1	0	0	0
1	0	0	1	1	1	0	1	0	0	0
1	1	0	0	1	0	1	0	0	0	0

# Proprietà

Nome	Forma AND	Forma OR
Elemento neutro	$1A = A$	$0 + A = A$
Assorbimento	$0A = 0$	$1 + A = 1$
Idempotenza	$AA = A$	$A + A = A$
Complementazione	$\overline{AA} = 0$	$A + \overline{A} = 1$
Proprietà commutativa	$AB = BA$	$A + B = B + A$
Proprietà associativa	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Proprietà distributiva	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Legge di assorbimento	$A(A + B) = A$	$A + AB = A$
Legge di De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{AB}$

**Figura 3.6** Identità dell'algebra booleana.

# XOR

- La funzione XOR verifica la disegualanza di due variabili

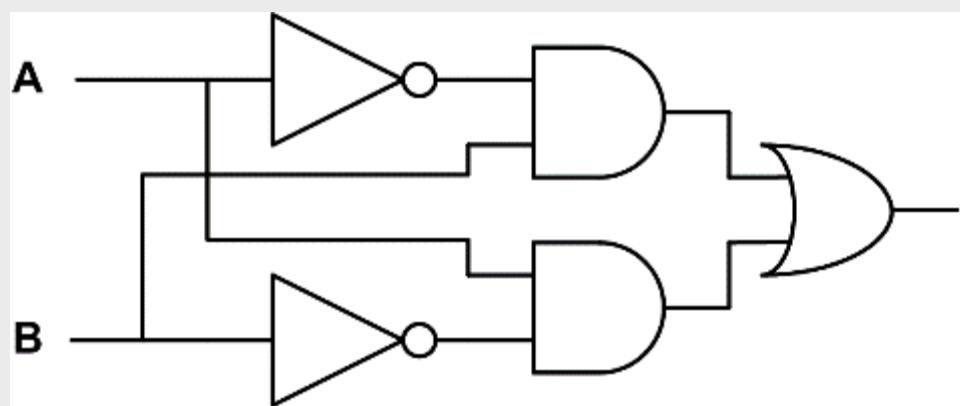
*Exclusive-OR gate*



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Esercizio:

Si verifichi che  $A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$



$$A \oplus B = A'B + AB'$$

# ESERCIZI

Verificare l'equivalenza delle seguenti funzioni logiche:

- $\overline{A} \overline{B} \overline{C} + B \overline{C} + A(B + \overline{B}\overline{C}) \equiv A + \overline{C}$

- $R = \overline{ab} + \overline{a}\overline{b}$   
 $S = \overline{\overline{a}\overline{b}} + ab$   
 $T = (a + b)(\overline{a} + \overline{b})$

- $F = \overline{ab} + a\overline{b} + \overline{a + bc}$   
 $H = \overline{b} + \overline{a}$

# ESERCIZI

Esprimere in forma simbolica la seguente preposizione logica: Il passaggio di un astronauta da una nave di servizio ad un satellite artificiale, è permesso se:

- La nave e il satellite sono uniti e alla stessa pressione interna, oppure se
- Sono separati e l'astronauta indossa la tuta pressurizzata.

In entrambi i casi occorre che le pile solari funzionino e giunga il consenso del controllo da terra.

*variabili*

*P, passaggio dell'astronauta;*

*U, nave e satellite uniti;*

*I, stessa pressione interna;*

*T, l'astronauta indossa la tuta pressurizzata;*

*S, pile solari funzionanti;*

*C, consenso da terra*

NB: veicoli separati = non uniti  
 $(U = 0)$ .

$$\begin{aligned}P &= UISC + \bar{U}TSC = \\&= SC(UI + \bar{U}T)\end{aligned}$$

# ESERCIZI

Esprimere in forma simbolica la seguente preposizione logica:  
l'avanzamento di un nastro trasportatore è permesso secondo due modi di funzionamento:

1. È inserito l'interruttore di alimentazione e vi sono pezzi da trasportare
2. È inserito l'interruttore, vi sono pezzi da trasportare e il numero di pezzi già trasportato è inferiore ad un limite N (prefissato)

Inoltre l'avanzamento si deve arrestare automaticamente in caso di incidente (es. caduta di un pacco, ecc.).

*Variabili*

*M, modo di funzionamento (M= 1, primo modo; M= 0, secondo modo);*

*I, posizione interruttore;*

*Pci sono pezzi da portare;*

*N, i pezzi trasportati sono meno di N;*

*C, c'è stato un incidente;*

*A, avanzamento del nastro*

$$A = (IPM + IP\bar{M}N)\bar{C}$$

# ESERCIZI

Progettare una rete combinatoria a tre ingressi che restituisca 1 solo se almeno due degli ingressi valgono 1

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

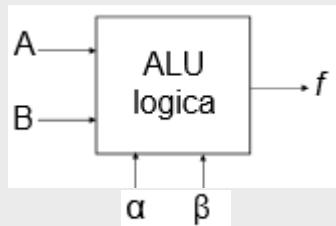
Espressione booleana

$$f = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

1. Si identificano le variabili logiche
2. Si crea la parte sinistra della tavola di verità che ha un numero di righe pari a  $2^N$  ( $N$  numero di variabili)
3. Si scrivono tutte le combinazioni
4. Si interpreta ogni riga in base al problema e si riporta il valore che deve assumere l'output
5. Si considerano solo le righe in cui l'output vale 1 e si scrive la funzione come somma di prodotti logici

# ESERCIZI

- Si scriva, utilizzando gli operatori booleani AND, OR, NOT, la funzione booleana che ritorna in uscita il valore 1 se sono veri due dei quattro input
- Scrivere, utilizzando gli operatori booleani AND, OR e NOT, la funzione logica che riceve in ingresso un numero binario su quattro bit e restituisce VERO se e solo se il numero in ingresso è compreso tra 4 e 7.
- Progettare una rete combinatoria che realizzi un'ALU ad 1 bit capace di eseguire le operazioni logiche bit a bit di AND, OR, NOT, XOR



$$\begin{aligned} f &= AB && \text{se } \alpha=0 \beta=0 \\ f &= A+B && \text{se } \alpha=0 \beta=1 \\ f &= \bar{A} && \text{se } \alpha=1 \beta=0 \\ f &= A \text{ xor } B && \text{se } \alpha=1 \beta=1 \end{aligned}$$

$$\begin{aligned} f = & \bar{\alpha}\bar{\beta}AB + \bar{\alpha}\beta\bar{A}B + \bar{\alpha}\beta A\bar{B} + \bar{\alpha}\beta AB + \\ & + \alpha\bar{\beta}\bar{A}B + \alpha\bar{\beta}\bar{A}B + \alpha\beta\bar{A}B + \alpha\beta A\bar{B} \end{aligned}$$

# ESERCIZI

Trovare le espressioni per le funzioni booleane

$$f(x_1, x_2) \text{ e } g(x_1, x_2)$$

Definite come segue

$$\begin{aligned} f(x_1, x_2) = 0 & \text{ se e solo se} \\ x_1 = 1 \text{ e } x_2 = 0 & \end{aligned}$$

$$\begin{aligned} x_1 & \text{ se } x_2 = 0 \\ g(x_1, x_2) = \{ & \\ \overline{x}_1 & \text{ se } x_2 = 1 \end{aligned}$$

Verificare se la funzione  $g$  è equivalente alla seguente:

$$h(x_1, x_2) = \overline{f(x_1, x_2)} \bullet f(\overline{x_2}, \overline{x_1})$$

# Error Correcting Codes

*Per vari motivi come il cambiamento di tensione di alimentazione del chip, accoppiamento tra le piste, fenomeni atmosferici o altre cause, i valori sulle linee di trasmissione possono essere modificati e i dati trasmessi errati.*

I codici di rilevazione e/o correzione degli errori sono codici che consentono la rilevazione e/o correzione degli errori in una parola.

## Esempio

Parola originaria (trasmessa)      100100

Parola finale (ricevuta)      100110

# Error Correcting Codes

Date due parole di codice :

$$A : a_{n-1} a_{n-2} \dots a_i \dots a_1 a_0$$

e

$$B : b_{n-1} b_{n-2} \dots b_i \dots b_1 b_0$$

La distanza di Hamming tra A e B è definita come:

$$H(A, B) = \sum_{i=0}^{n-1} d_h(a_i, b_i)$$

dove

$$\begin{aligned} d_h(a_i, b_i) &= 0 && \text{se } a_i = b_i \\ d_h(a_i, b_i) &= 1 && \text{se } a_i \neq b_i \end{aligned}$$

# Error Correcting Codes

Si definisce distanza di Hamming di un codice la minima distanza tra due parole di un codice.

Esempio. Il codice di quattro parole valide:

0000000000

1111100000

0000011111

1111111111

ha distanza di Hamming pari a 5.

# Error Correcting Codes

Significato della **distanza di Hamming**: se tra due parole di codice vi è una distanza di Hamming pari a  $d$ , allora saranno necessari  $d$  errori singoli per trasformare una parola nell'altra.

A	B	$H(A,B)$
101	111	1
1100	0011	4
100011	100101	2

# Error Correcting Codes

La distanza di Hamming gioca un ruolo chiave nella rilevazione e correzione di errori in un codice:

- Per rilevare  $d$  errori singoli è necessario un codice con distanza di Hamming  $d+1$  (infatti in questo modo non esiste alcun modo in cui  $d$  errori singoli possono cambiare una parola valida in un'altra parola valida);
- Per correggere  $d$  errori singoli è necessario un codice con distanza di Hamming  $2d+1$  (infatti in questo modo anche con  $d$  cambiamenti la parola di codice originaria continua ad essere “più vicina” rispetto a tutte le altre non esiste alcun modo in cui  $d$  errori singoli possono cambiare una parola valida in un'altra parola valida);

# Error Correcting Codes

Esempio. Dato il codice con distanza di Hamming pari a 5:

```
0000000000  
1111100000  
0000011111  
1111111111
```

In questo caso è possibile:

- rilevare fino a 4 errori ( $d+1=5$ ): 0000000001; 1111000011; 1010101000  
(Non è possibile rilevare i 5 errori che modificano 0000000000 in 0000011111 !!!)
- correggere fino a 2 errori ( $2d+1=5$ ): 001111111 → 111111111;  
0000010101 → 0000011111 (Non è possibile correggere i 3 errori che modificano 0000000000 in 0000000111 !!!)

# Error Correcting Codes

- Nei codici a rilevazione e/o correzione di errori si utilizzano alcuni bit extra (ridondanti) che vengono aggiunti alla parola stessa. Questi bit ridondanti si chiamano anche “bit di controllo”
- L’idea è quella di creare codici con distanza di Hamming maggiore al fine di poter rilevare e/o correggere errori.

Data una parola di  $m$  bit di dati, si aggiungono  $r$  bit extra di controllo.  
Si ottiene così una unità di  $n=m+r$  bit(codeword)

Con una parola di  $m$  bit tutte le  $2^m$  combinazioni sono legali ma, per via di come sono calcolati i bit di controllo, solo  $2^m$  delle  $2^n$  parole di codice sono valide.

# Error Correcting Codes

## Bit di parità

Esempio: Codice con controllo di parità

Dato +1 bit “di parità”

Il bit di parità viene scelto in modo tale che il numero di bit 1 nella parola di codice sia pari (oppure dispari)

Dato	bit di parità
1001011	0
1011	1
1111011	0
1011001111	1

*Un codice con bit di parità ha distanza di Hamming pari a 2: ogni singolo errore genera una parola di codice la cui parità è errata.*

*Esempio:*    A: 10011    1 (parity bit)  
                    B: 10001    0 (parity bit)

*Servono due errori singoli per modificare una parola di codice valida in un'altra parola valida.*

*NB: so che si è verificato un errore singolo....ma non so dove!!*

# Error Correcting Codes

## Codice di Hamming

### Codice di Hamming

Problema: Si vuole realizzare un codice con  $m$  bit dati ed  $r$  bit di controllo, che sia capace di correggere tutti gli errori singoli.

# Error Correcting Codes

## Codice di Hamming

Esempio: Codice di Hamming

Ciascuna delle  $2^m$  parole legali A ha :

- $n$  ( $n=m+r$ ) parole illegali a distanza 1 da essa.
- richiede quindi  $(n+1)$  stringhe di bit ad essa dedicate (1 per la parola corretta ed  $n$  per i possibili errori).

Quindi per poter rappresentare  $2^m$  parole abbiamo bisogno di  $(n+1) \cdot 2^m$  stringhe differenti.

Allora dovrà essere:

$$(n+1) \cdot 2^m \leq 2^n$$

Ovvero

$$(m+r+1) \leq (2^n / 2^m) = 2^r$$

# Error Correcting Codes

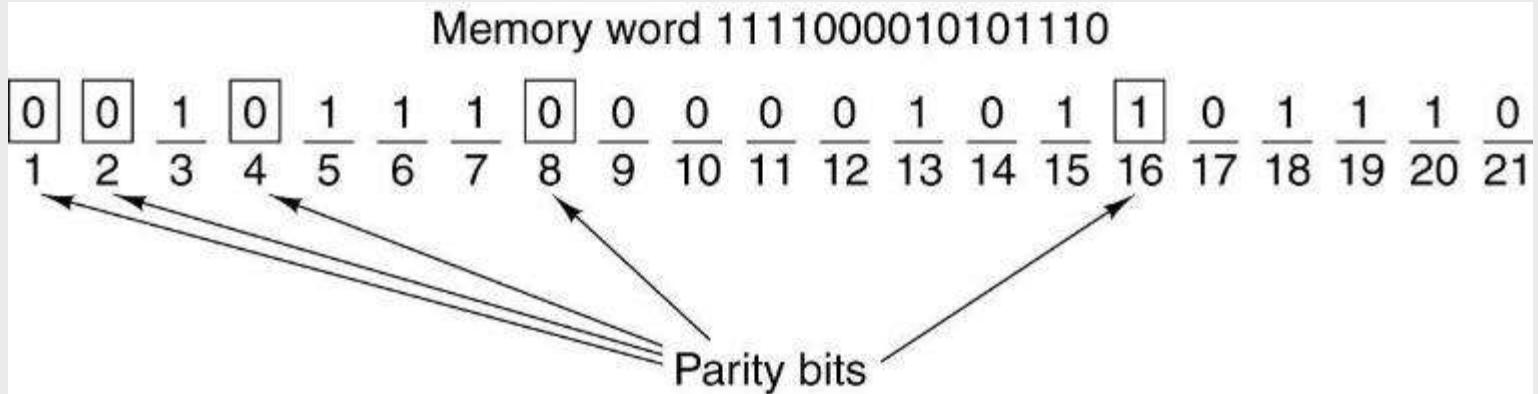
## Codice di Hamming

Number of check bits for a code that can correct a single error  $( (m+r+1) \leq 2^r )$

Word size	Check bits	Total size	Percent overhead
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

# Error Correcting Codes

## Codice di Hamming



- Nel codice di Hamming gli  $r$  bit di parità sono aggiunti a una parola di  $m$  bit, formando una nuova parola di  $n$  bit ( $n=m+r$ ).
- I bit sono numerati da 1 (MSD – Most Significant Digit).
- Tutti i bit la cui posizione è potenza di 2 sono bit di parità (1,2, 4,8, 16, ecc.)
- Quelli restanti sono usati per i dati.

# Error Correcting Codes

## Codice di Hamming

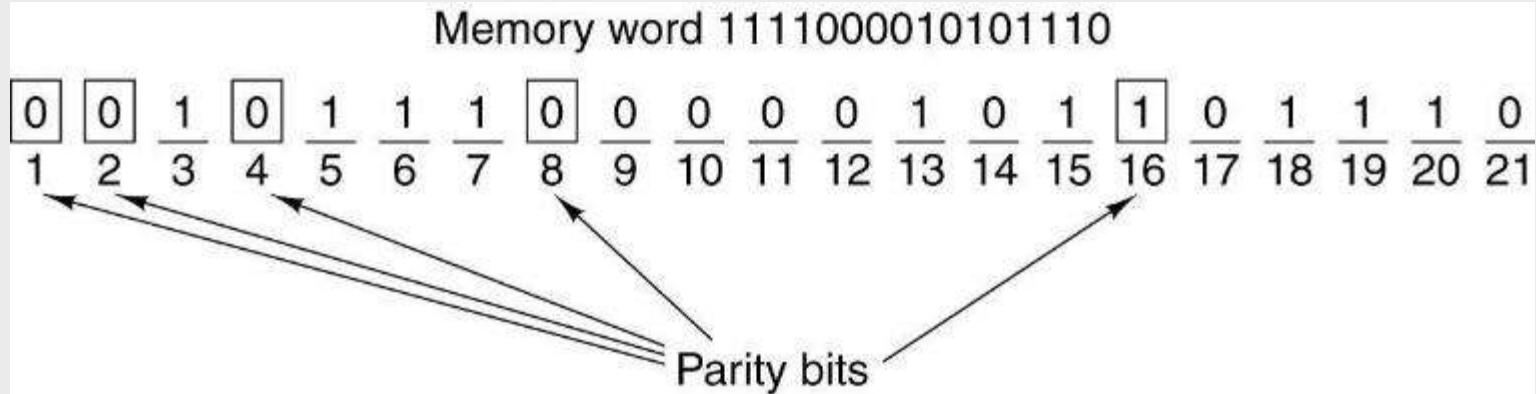
Ciascun bit di parità controlla posizioni specifiche dei bit ed è impostato in modo che sia pari il numero totale dei bit che hanno valore 1 nelle posizioni controllate.

- Il bit 1 controlla i bit dispari: 1,3,5,7,9,11,13,15,17,19,21
- il bit 2 controlla i bit : 2,3,6,7,10,11,14,15,18,19
- Il bit 4 controlla i bit: 4,5,6,7,12 ,13,14,15,20,21
- Il bit 8 controlla i bit: 8 ,9,10,11,12,13,14,15
- il bit 16 controlla i bit: 16,17,18,19,20,21.

In particolare il bit di posto  $b$  è controllato dai bit di controllo  $b_1, b_2, \dots, b_j$  tale che  $b = b_1 + b_2 + \dots + b_j$ .

# Error Correcting Codes

## Codice di Hamming

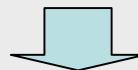


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	*		*		*		*		*		*		*		*		*		*		*
2		*	*		*	*		*	*		*	*		*	*		*	*		*	*
4			*	*	*	*				*	*	*	*						*	*	
8							*	*	*	*	*	*	*	*	*						
16																*	*	*	*	*	*

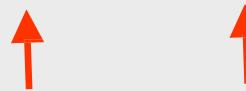
# Error Correcting Codes

## Codice di Hamming

0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	



0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	*		*		*		*		*		*		*		*		*		*		*
2		*	*		*	*		*	*		*	*		*	*		*	*		*	*
4			*	*	*	*				*	*	*	*	*					*	*	
8							*	*	*	*	*	*	*	*	*						
16																*	*	*	*	*	*

# Organizzazione dei sistemi di calcolo

*Prof. Ing. Donato Impedovo*

---

# Calcolatori Elettronici

Distinzioni:

Potenza di calcolo e capacità di memorizzazione

Ambiente e scopo per cui sono utilizzati.

In generale si possono distinguere:

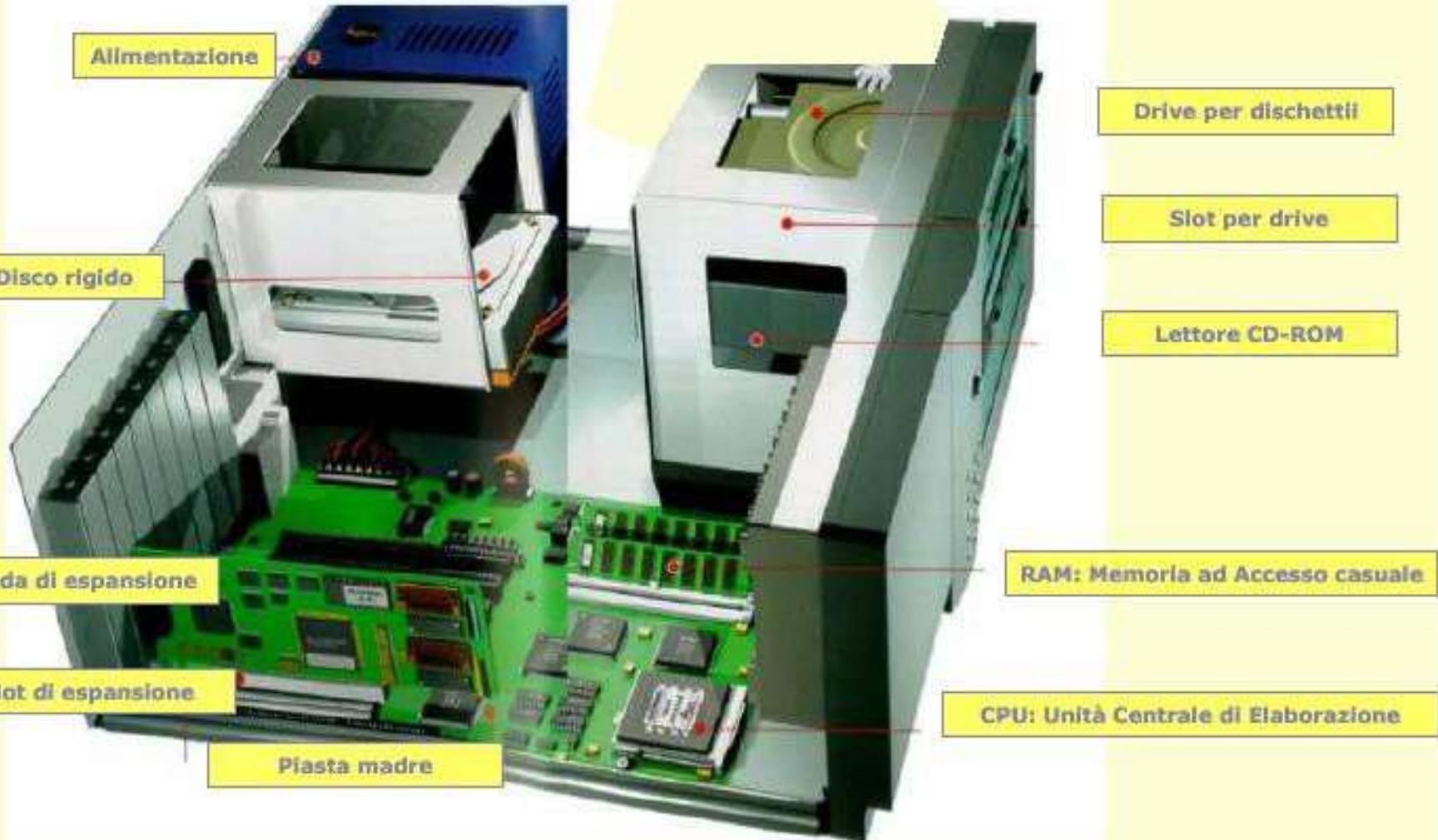
Personal Computer (PC): usati come elaboratori di testo, Internet, banche dati, strumenti da ufficio, etc.

Workstation: usati per il calcolo e la programmazione, per la grafica avanzata e la ricerca.

MainFrame: grandi aziende, banche, gestione di complesse reti di computer e di apparecchiature, applicazioni gestionali

Network computer: computer collegati in rete condividendo dati (dischi) e altre risorse. I “terminali” sono postazioni prive di capacità di elaborazione, dotate solo di monitor e tastiera e collegate ad un computer centrale di cui sfruttano la CPU e la memoria.

# Componenti principali di un Computer



# Architettura dei Calcolatori e Dispositivi I/O

## □ Architettura di Von-Neuman (estesa)

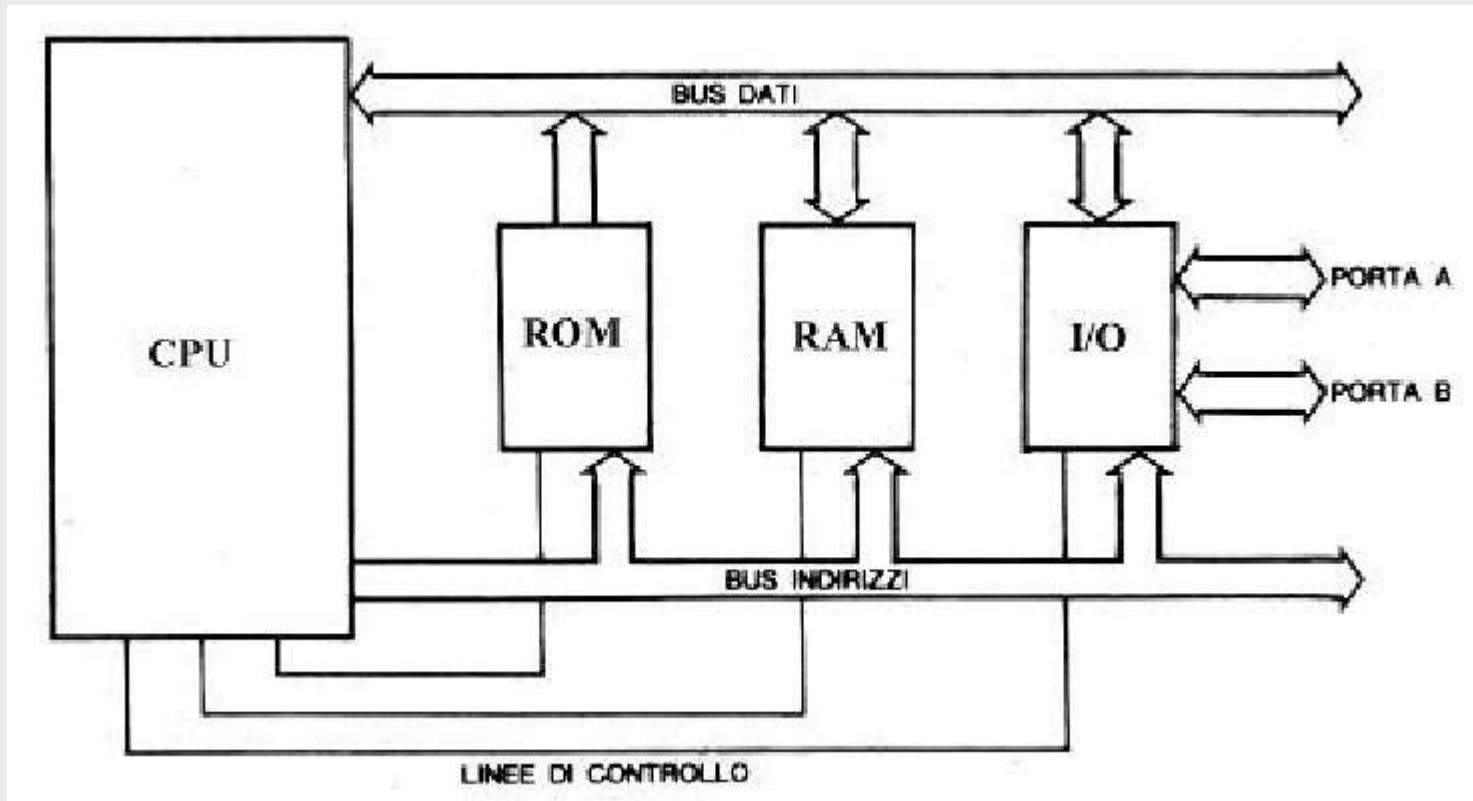
- | CPU
- | ROM
- | RAM
- | Dispositivi di I/O:
  - Monitor
  - Hard Disk
  - Drive CD/DVD
  - Stampanti
- | BUS

- Architettura o schema di progettazione di calcolatori elettronici che prende nome dal matematico John Von Neuman.
- Schematizzazione molto sintetica, ma molto potente: i moderni computer (Personal Computer – PC) sono progettati secondo tale architettura.

Hardware: parte fisica di un personal computer: circuiti elettrici ed elettronici, cavi, supporti, schede, monitor, tastiera, dischi, stampanti, etc...

-hard (duro), ware (manufatto, oggetto)

# Macchina di Von Neuman (estesa)



# Scheda Madre

Raccoglie la circuiteria elettronica di interfaccia fra le componenti principali e fra queste e i bus di espansione e le interfacce verso l'esterno. È responsabile della trasmissione e temporizzazione corretta dei segnali.



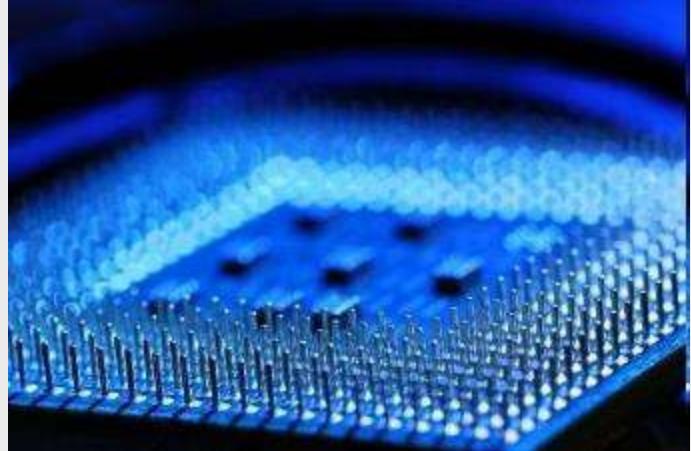
Principali componenti montati sulla scheda madre (tramite slot):  
Processore  
RAM  
ROM (BIOS)  
scheda video, scheda audio

Collegati tramite cavi alla scheda madre:  
l'hard disk  
lettore di DVD  
Alimentatore  
slot lettura penne USB, etc.

# Central Processing Units (CPU): Processore

La CPU esegue le istruzioni di un programma (che deve essere presente in memoria RAM, ROM o CACHE).

Durante l'esecuzione del programma, la CPU legge o scrive dati in memoria; il risultato dell'esecuzione dipende dal dato su cui opera e dallo stato interno della CPU stessa, che tiene traccia delle passate operazioni.



Elementi fondamentali:

Registri: locazioni di memoria interne alla CPU, molto veloci, a cui è possibile accedere molto più rapidamente che alla memoria centrale. Il valore complessivo di tutti i registri della CPU costituisce lo stato in cui essa si trova in un determinato istante. (Intel: circa 20 registri)

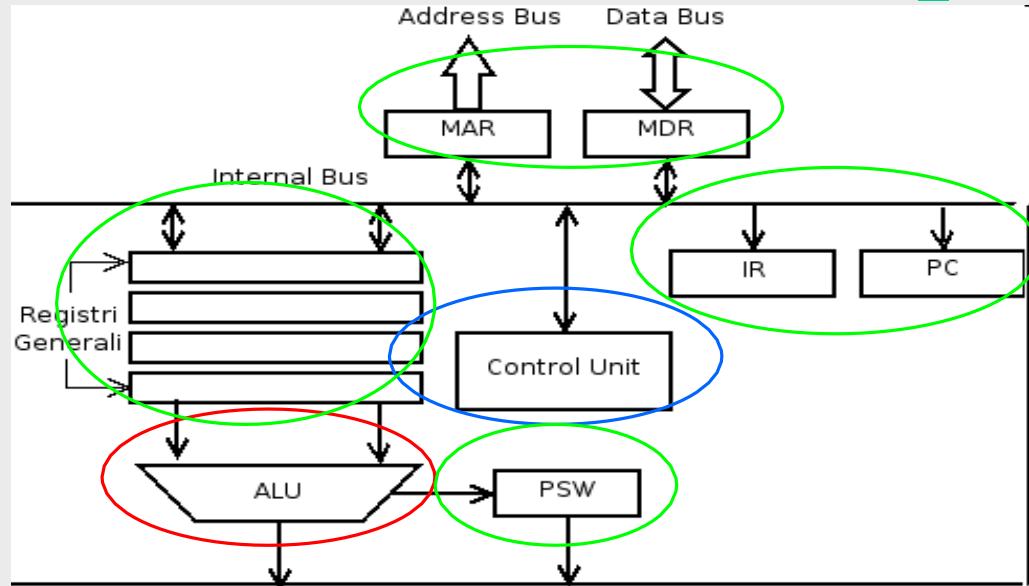
ALU (Arithmetic Logic Unit): esegue le operazioni logiche e aritmetiche.

Unità di controllo CU (Control Unit): “legge” dalla memoria le istruzioni, le decodifica, se occorre legge i dati per l'istruzione, “esegue” l'istruzione e “memorizza” il risultato se c'è, scrivendolo in memoria o in un registro della CPU.

# Central Processing Units (CPU): Processore

Componenti principali:

- registri,
- l'unità logica/aritmetica (ALU)**
- l'unità di controllo**



Svolge semplici azioni in maniera sequenziale.

Lunghezza parola: 32 vs. 64 bit

Prestazioni di un processore:

- └ Velocità del clock [GHz].
- └ Istruzioni al secondo MIPS  
es. Pentium 3.8GHz fa circa 1000MIPS (1GIPS)

# I registri del processore

## □ Visibili all'utente

Disponibili per tutti i programmi

Permettono di minimizzare i riferimenti alla memoria principale

Tipicamente disponibili sono quelli di:

- dati
  - general purpose: *utilizzati per memorizzare temporaneamente dati e/o informazioni di controllo di vario genere*
  - Dedicati: *dedicati sempre ed esclusivamente ad uno scopo*
- indirizzi (es: index register, segment pointer, stack pointer)
- codici di condizione (flag -parzialmente visibili)

## □ Di stato e controllo

Memorizzano l'esito delle operazioni del processore

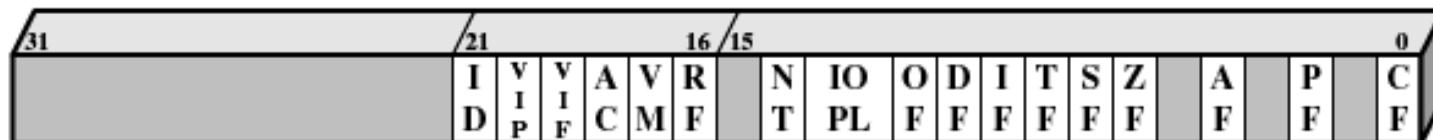
Solo ad alcuni programmi è possibile accedere tramite istruzioni eseguite in modalità di controllo o di sistema operativo

# I registri del processore

- Scambio dati con la memoria:
  - MAR (Memory Address Register): indirizzo di riferimento alla memoria
  - MDR (Memory Data Register): dati provenienti/da inviare alla memoria
- Scambio dati con i moduli di I/O:
  - I/O AR (Input Output Address Register): specifica il dispositivo di I/O
  - I/O BR (Input Output Buffer Register): contiene i dati da scambiare con il dispositivo
- Esecuzione delle istruzioni:
  - PC (Program Counter): indirizzo della successiva istruzione da eseguire
  - IR (Instruction Register): istruzione corrente
- Controllo dell'esecuzione:
  - PSW (Program Status Word)
    - informazioni di stato (abilitazione/disabilitazione di interrupt, bit selezione SU o utente)
    - NB: alcune info di controllo sono specifiche per un SO (designed for...)

# I registri del processore

*Registro i-flag del Pentium:*



ID = Identification flag

VIP = Virtual interrupt pending

VIF = Virtual interrupt flag

AC = Alignment check

VM = Virtual 8086 mode

RF = Resume flag

NT = Nested task flag

IOPL = I/O privilege level

OF = Overflow flag

DF = Direction flag

IF = Interrupt enable flag

TF = Trap flag

SF = Sign flag

ZF = Zero flag

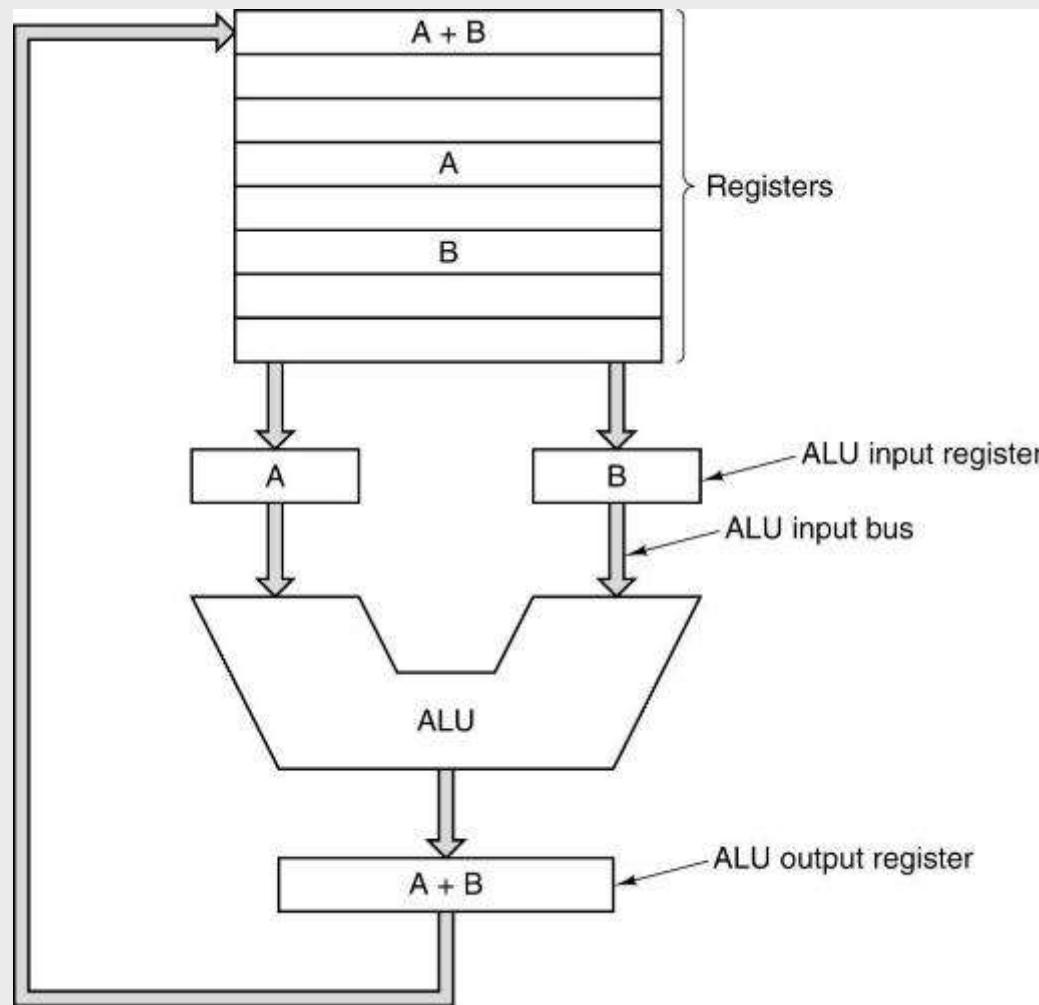
AF = Auxiliary carry flag

PF = Parity flag

CF = Carry flag

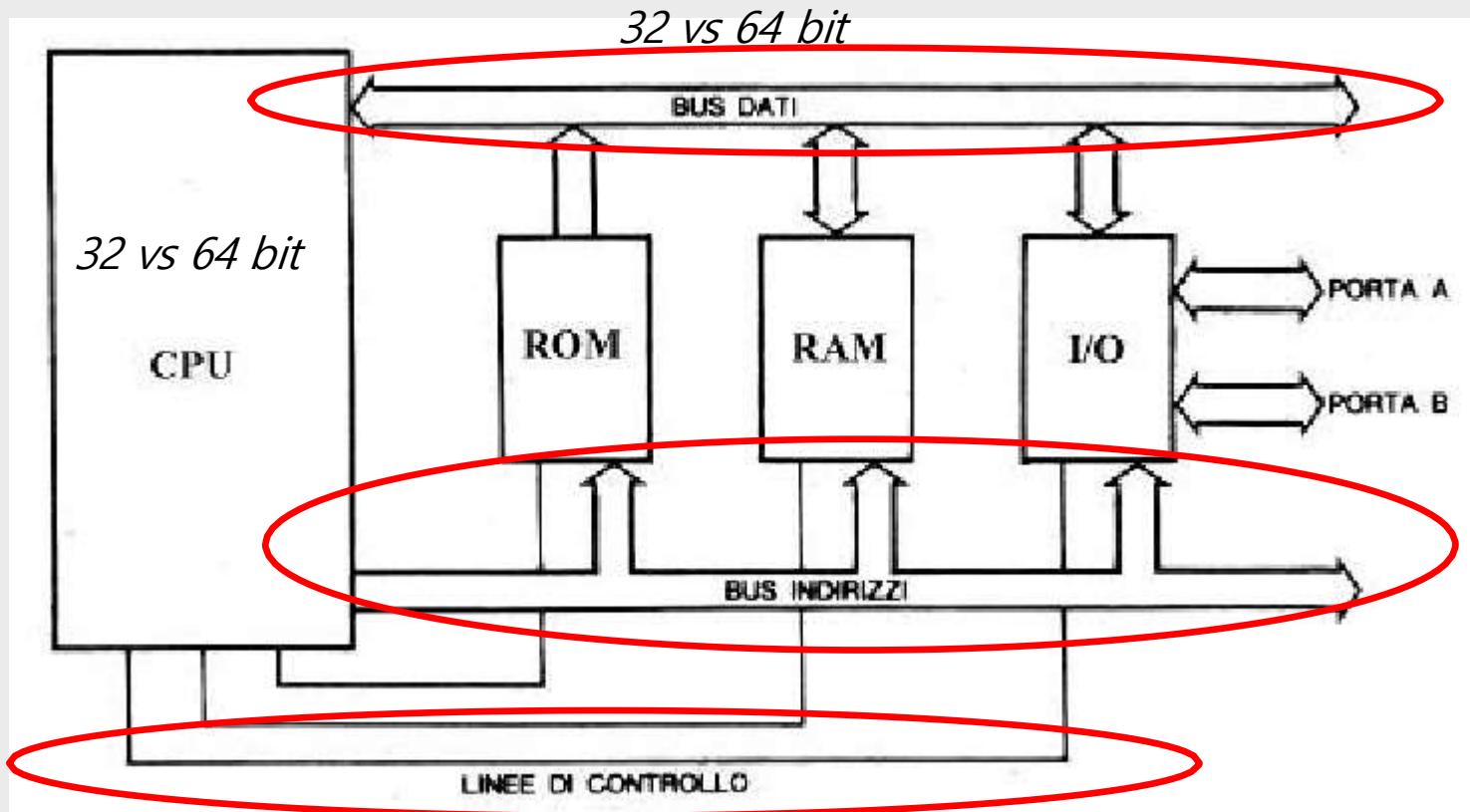
Figure 3.12 Pentium II EFLAGS Register

# Data path in una macchina di Von Neumann (dati nei registri)



# Il BUS

BUS: piste su circuito stampato utilizzate per trasportare i segnali elettrici che rappresentano i bit



# Il BUS

**BUS DATI:** canale di comunicazione attraverso il quale “viaggiano” i dati . usufruibile da tutti i componenti del sistema (scrittura/lettura) bidirezionale.

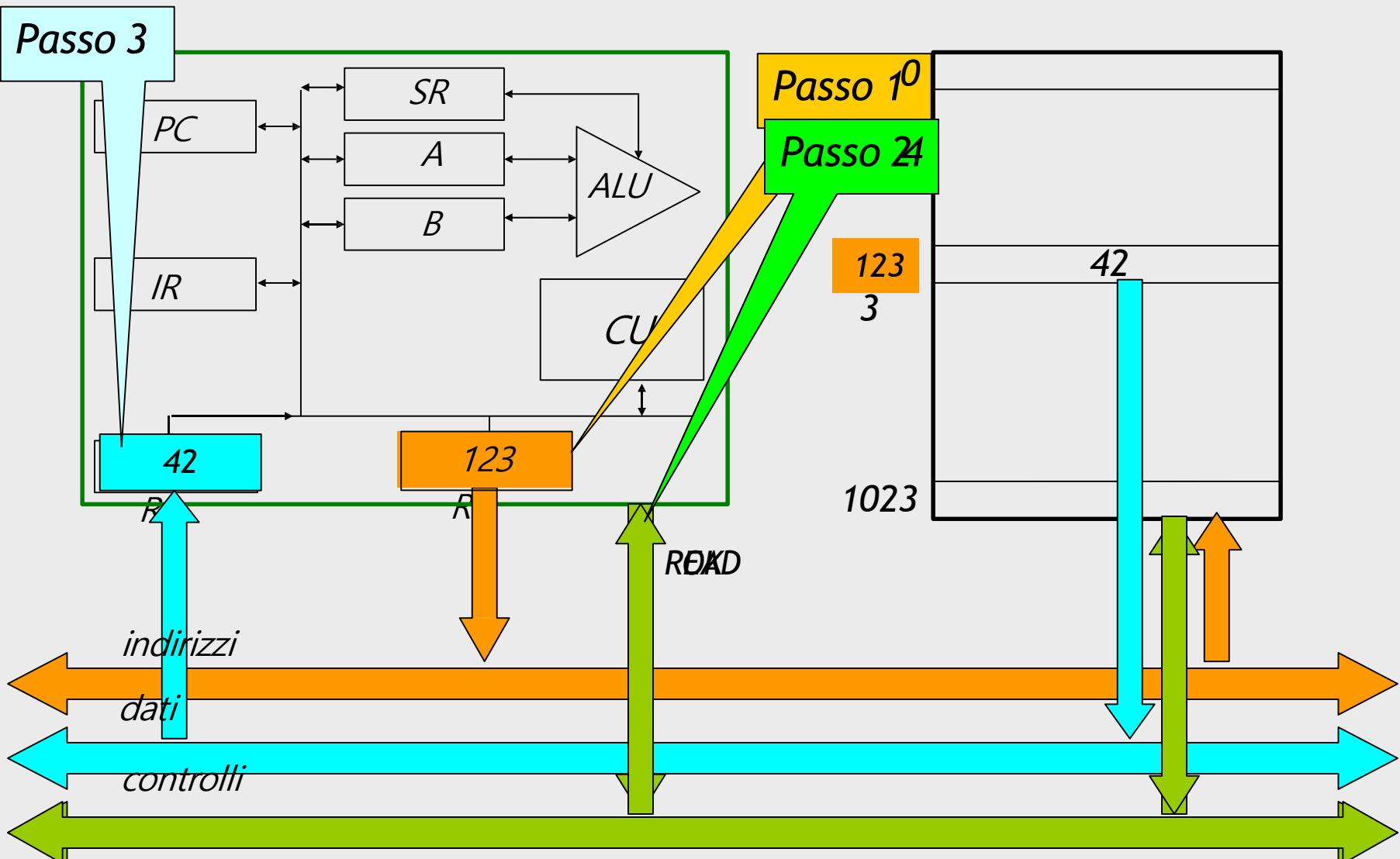
Caratterizzato dall’ampiezza in termini di bit.

Un bus a 32(64) bit può “trasportare” 32(64) bit alla volta.

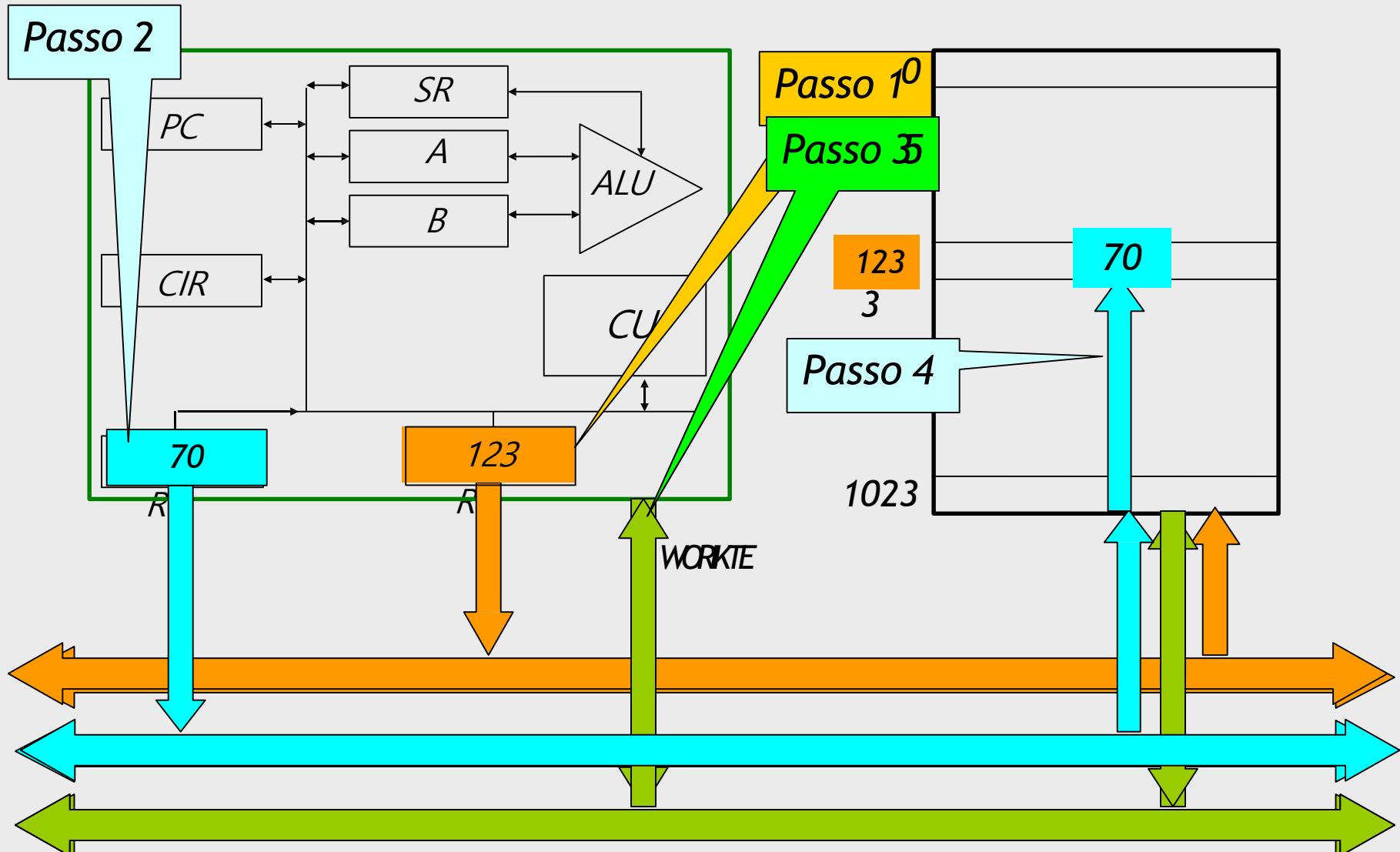
**BUS INDIRIZZI:** canale attraverso il quale la CPU specifica in quale indirizzo (celle di memoria RAM o periferiche di I/O) andare a scrivere/leggere dati  
fruibile in scrittura solo dalla CPU ed in lettura dagli altri componenti  
Monodirezionale

**BUS CONTROLLI:** insieme di collegamenti tramite i quali è possibile coordinare le attività del sistema; tramite esso, la CPU può decidere quale componente deve scrivere sul bus dati in un determinato momento, quale deve leggere l’indirizzo sul bus indirizzi, quali celle di memoria devono scrivere e quali invece leggere, etc.

# *La sequenza di lettura*



# *La sequenza di scrittura*



# Central Processing Units (CPU): Processore

I processori possono implementare al loro interno più unità di esecuzione per eseguire più operazioni contemporaneamente. Questo approccio incrementa le prestazioni delle CPU ma ne complica l'esecuzione: per poter eseguire in modo efficiente più operazioni in parallelo la CPU deve poter organizzare le istruzioni.

La possibilità di disporre di più CPU permette al sistema operativo di far eseguire in parallelo più programmi aumentando notevolmente le prestazioni (multi threading)

- └ Es. processori multicore: CPU dual core: due processori “indipendenti”. Aumento della la potenza di calcolo senza aumentare la frequenza di clock. Minore calore, minore energia assorbita.
- └ Intel core i5 4.8GT/s (trasferimenti al secondo), i7(segmento superiore, ma più calore), AMD Athlon (NB: confrontabilità tra marche diverse sulla base dei GHz....)
- └

# Istruzioni

- Un programma è una sequenza di istruzioni
- Un'istruzione è un comando, eventualmente opera su dati

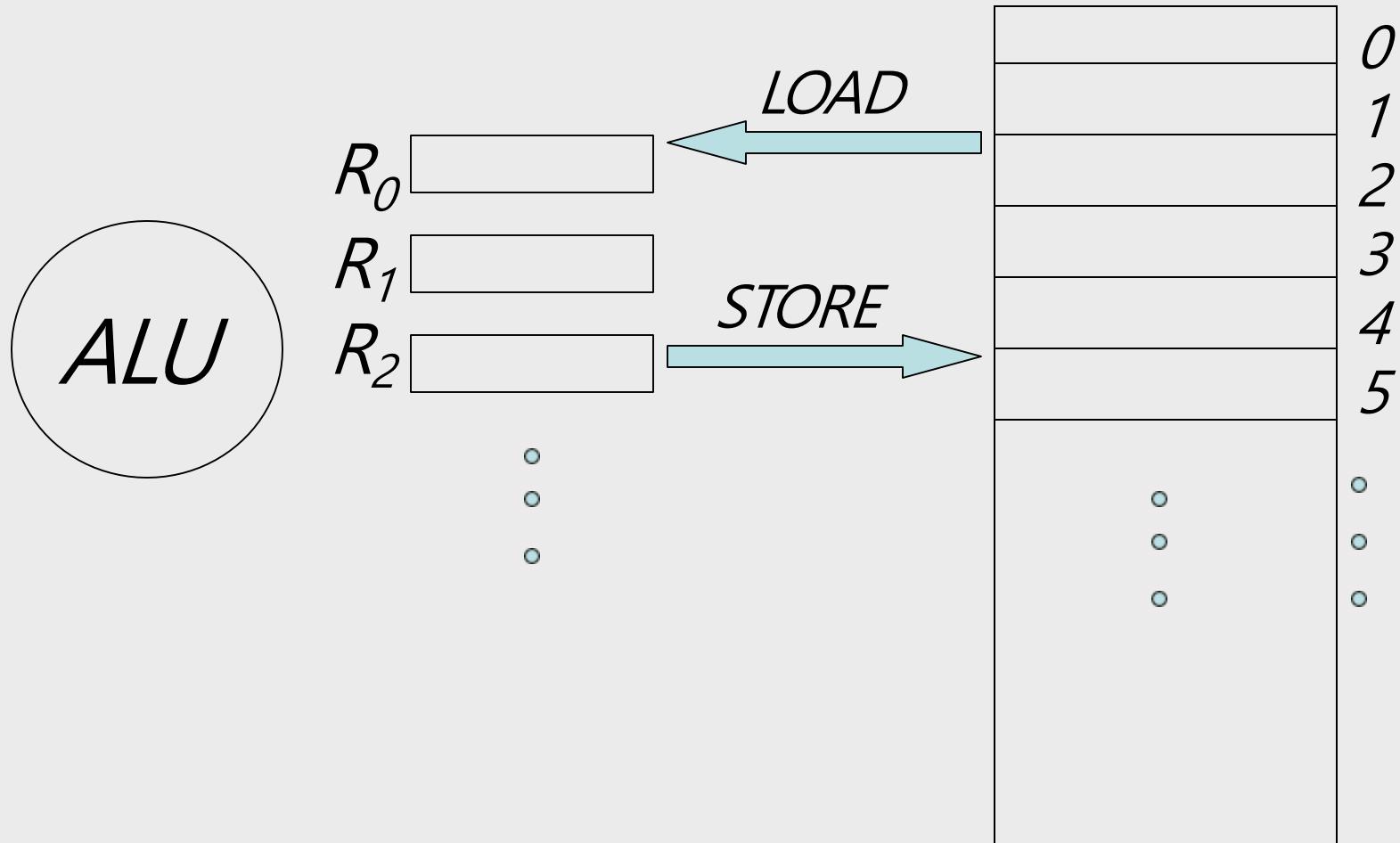
Tipi di istruzioni:

- Trasferimento
  - processore – memoria (e viceversa)
  - processore – modulo di I/O (e viceversa)
- Elaborazione di dati (operazioni logico-aritmetiche)
- Controllo (modifica della sequenza di esecuzione)
- Una istruzione può contenere una combinazione delle precedenti possibilità
- Il set di istruzioni dipende dal processore

# Formato delle Istruzioni

## Alcuni esempi

- Trasferimento REGISTRI  $\leftrightarrow$  RAM

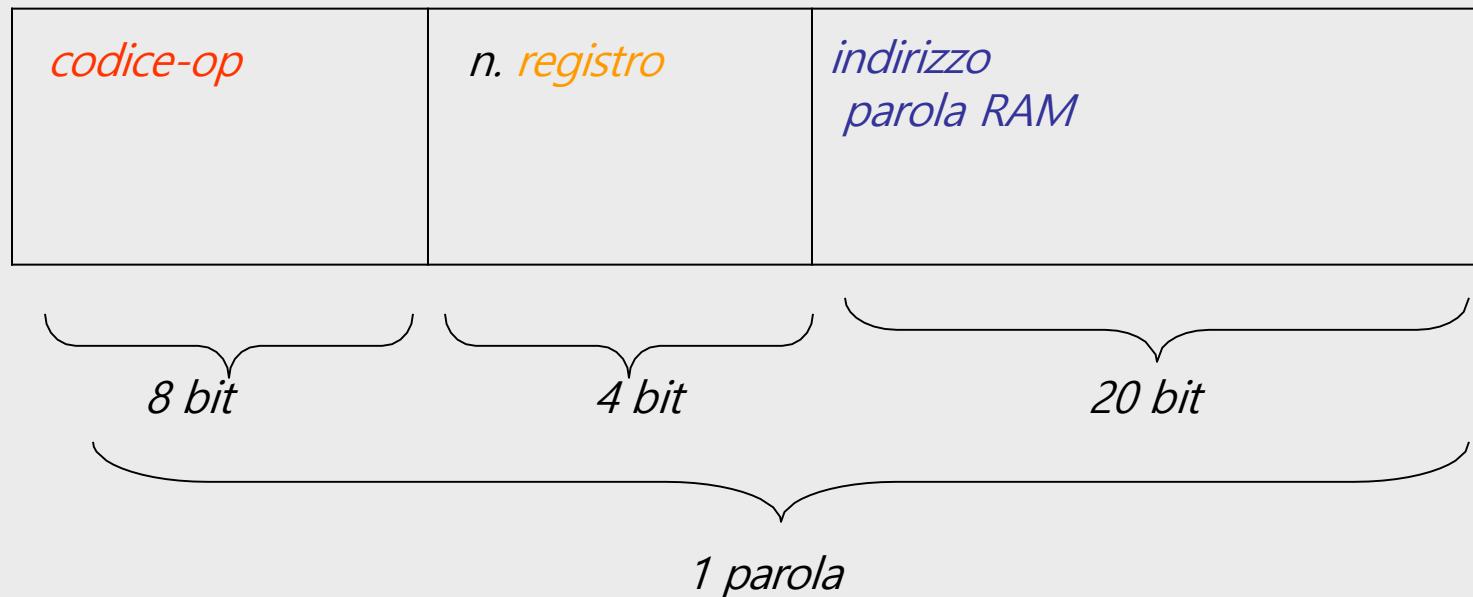


# Formato delle Istruzioni

## Alcuni esempi

- Trasferimento REGISTRI <=>RAM

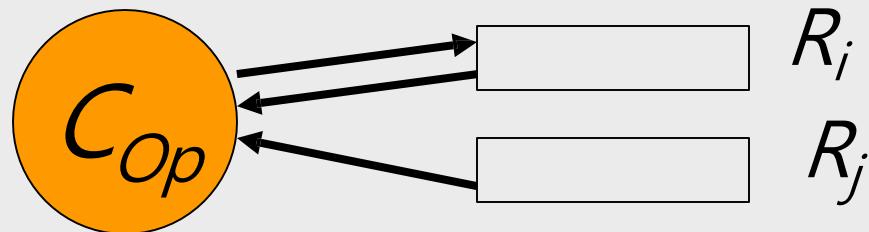
*in binario!*



# Formato delle Istruzioni

## Alcuni esempi

- Operazioni aritmetiche: eseguono somma, differenza, moltiplicazione e divisione usando i registri del processore come operandi



**ADD**    00000010

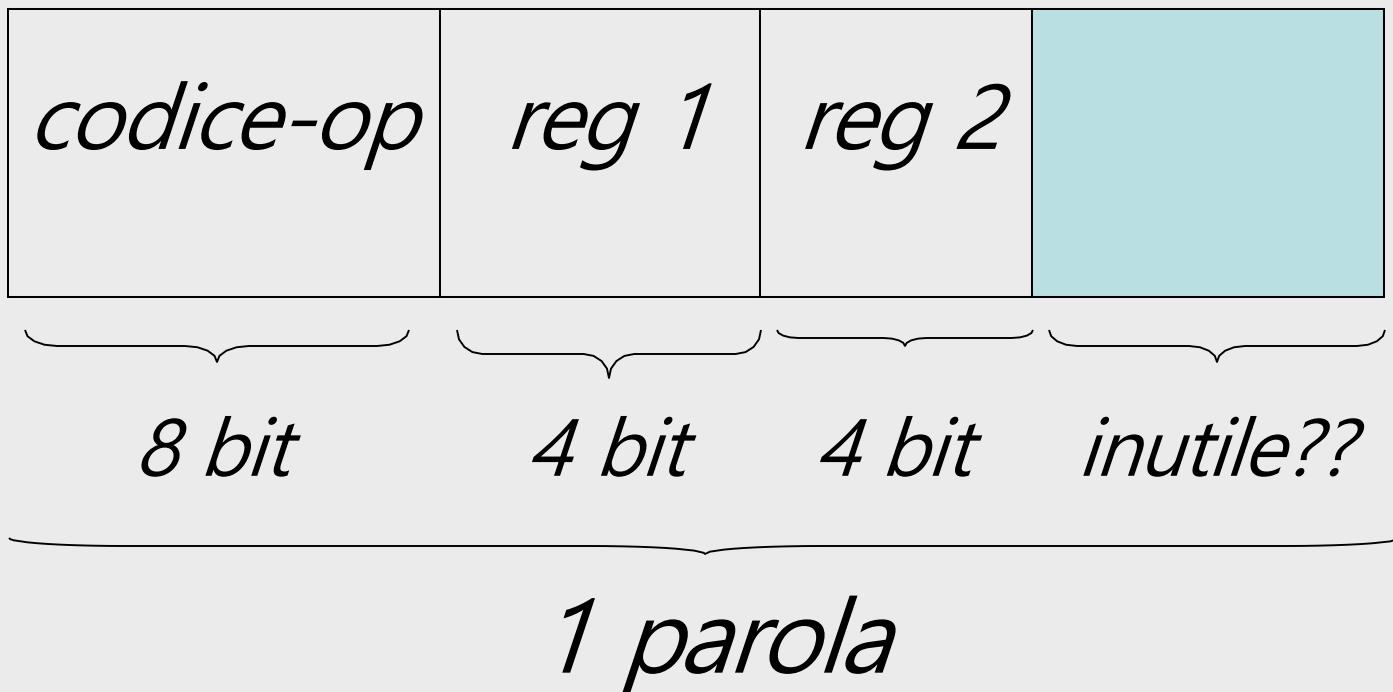
**MULT**    00000110

**MOD**    00001010

**SUB**    00000100

**DIV**    00001000

- Operazioni aritmetiche: eseguono somma, differenza, moltiplicazione e divisione usando i registri del processore come operandi

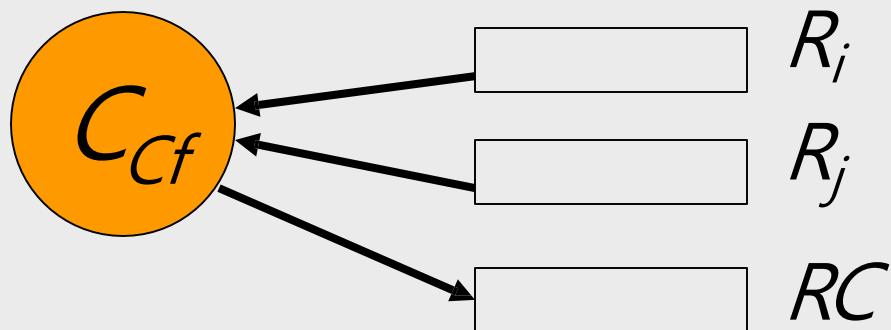


# Formato delle Istruzioni

## Alcuni esempi

Confronto: paragona il contenuto di 2 registri  $R_i$  ed  $R_j$  e:

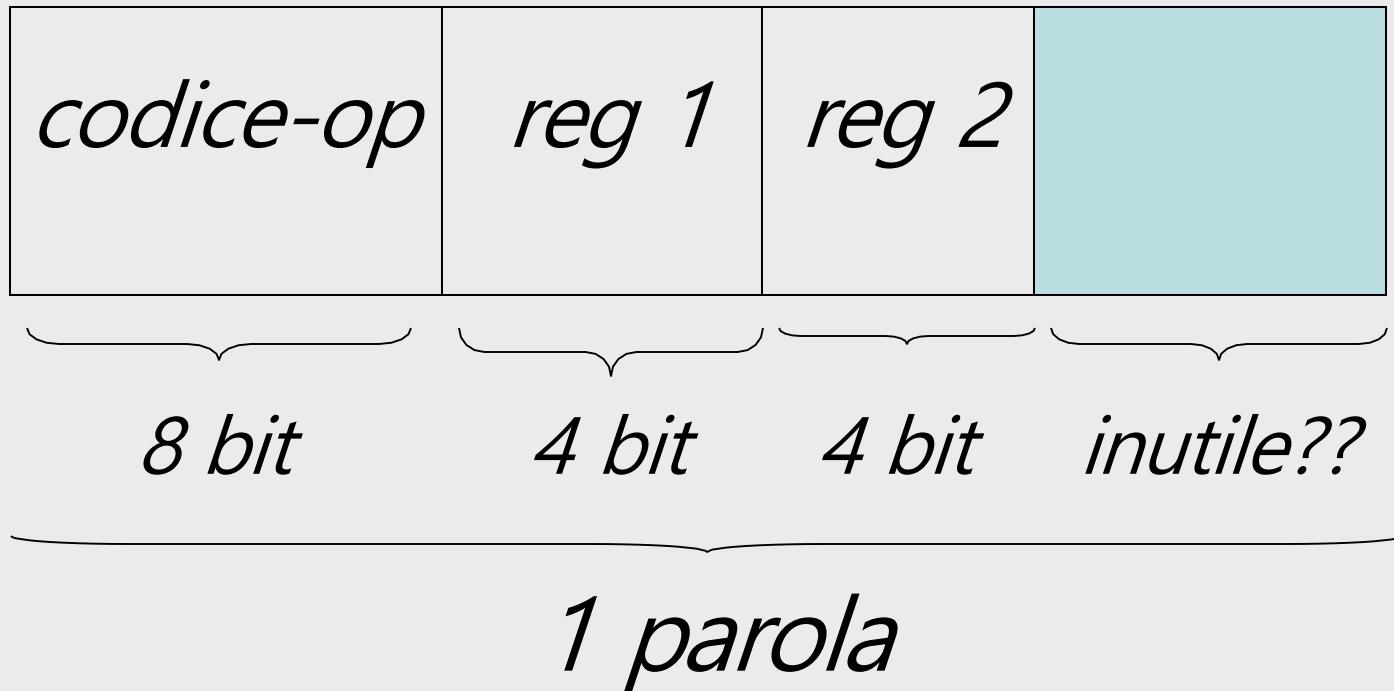
- se  $R_i < R_j$  mette -1 nel registro RC
- se  $R_i = R_j$  mette 0 in RC
- se  $R_i > R_j$  mette 1 in RC



*Codici:*       $COMP$        $00100000$

# Formato delle Istruzioni

## Alcuni esempi



# Formato delle Istruzioni

## Alcuni esempi

RISC «didattico»

### FORMATS:

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRR-type:	3 bits			3 bits			3 bits			4 bits				3 bits		
opcode	reg A				reg B				0				reg C			

### INSTRUCTIONS:

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD:	3 bits			3 bits			3 bits			4 bits				3 bits		
000	reg A				reg B				0				reg C			

ADD:	3 bits			3 bits			3 bits			7 bits						
001	reg A				reg B				signed immediate (-64 to 63)							

NAND:	3 bits			3 bits			3 bits			4 bits				3 bits		
010	reg A				reg B				0				reg C			

LUI:	3 bits			3 bits			3 bits			10 bits						
011	reg A				immediate (0 to 0x3FF)											

SW:	3 bits			3 bits			3 bits			7 bits						
100	reg A				reg B				signed immediate (-64 to 63)							

LW:	3 bits			3 bits			3 bits			7 bits						
101	reg A				reg B				signed immediate (-64 to 63)							

BEQ:	3 bits			3 bits			3 bits			7 bits						
110	reg A				reg B				signed immediate (-64 to 63)							

JALR:	3 bits			3 bits			3 bits			7 bits						
111	reg A				reg B				0							

Bit: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

# Formato delle Istruzioni

## Alcuni esempi

RISC CMPE 414/CMSC 691V

### GENERAL:

5 bit opcode

3 bit operand fields (Rd, Rs1, Rs2) to specify one of the 8 general purpose registers.

Immediates are restricted in length to the number of bit positions remaining in a 16 bit instruction word.

ALL instructions are 16 bits in length.

Shift immediates can be restricted to 2 bits.

### INSTRUCTIONS:

Rd - destination

Rs1 - source 1

Rs2 - source 2

# - 5/8/11-bit immediate

# Formato delle Istruzioni

## Alcuni esempi

### RISC CMPE 414/CMSC 691V

Table 1: OPCODES

Name	Opcode	Format	Notes
SEQ (SetIfEq)	00000	SEQ Rd Rs1 Rs2	Set register Rd to 1 if the values in Rs1 and Rs2 are equal, else set to 0.
ADD	00001	ADD Rd Rs1 Rs2	Add registers Rs1 and Rs2 and store the result in Rd.
SGT (SetIfGtr)	00010	SGT Rd Rs1 Rs2	Set register Rd to 1 if Rs1 > Rs2, else set to 0.
ADDI	00011	ADDI Rd Rs #	Store in register Rd the value Rs + the 5-bit <b>sign-extended</b> immediate.
SUB	00100	SUB Rd Rs1 Rs2	Compute Rs1 - Rs2 and store the result in Rd.
SLT (SetIfLes)	00101	SLT Rd Rs1 Rs2	Set register Rd to 1 if Rs1 < Rs2, else set to 0.
Unused	00110		
OR	00111	OR Rd Rs1 Rs2	Store in register Rd the bitwise OR of Rs1 and Rs2.
ORI	01000	ORI Rd Rs #	Store in register Rd the bitwise OR of Rs and the 5-bit <b>zero-extended</b> immediate.
AND	01001	AND Rd Rs1 Rs2	Store in register Rd the bitwise AND of Rs1 and Rs2.
ANDI	01010	ANDI Rd Rs #	Store in register Rd the bitwise AND of Rs and the 5-bit <b>zero-extended</b> immediate.
XOR	01011	XOR Rd Rs1 Rs2	Store in register Rd the bitwise XOR of Rs1 and Rs2.
XNOR	01100	XNOR Rd Rs1 Rs2	Store in register Rd the bitwise XNOR of Rs1 and Rs2.
NOT	01101	NOT Rd Rs	Store in register Rd the bitwise complement of Rs.
SRA (ShftRgtArith)	01110	SRA Rd Rs #	Store in register Rd the <b>sign-extended</b> value of Rs shifted to the right by the 2-bit immediate. (An immediate of 0 does not shift the operand).

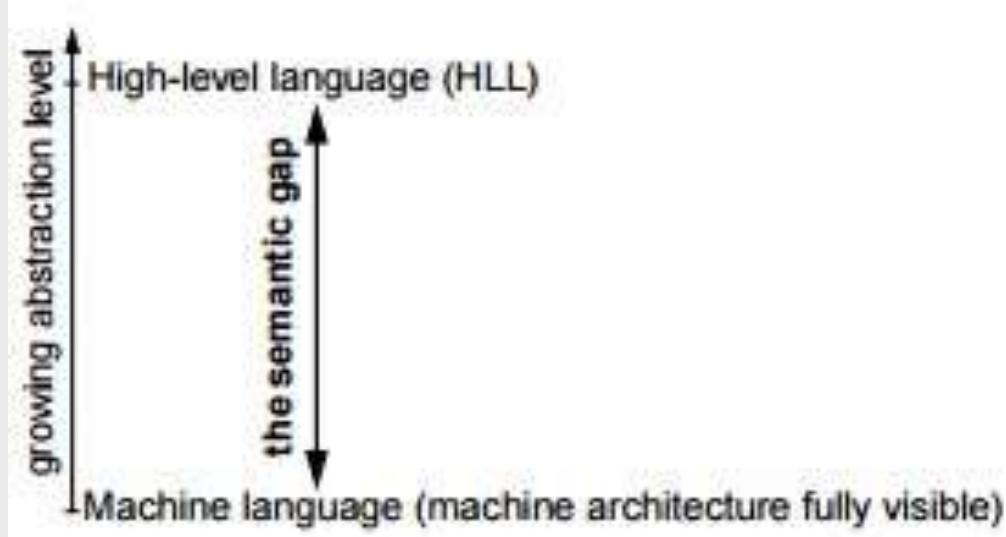
# Formato delle Istruzioni

## Alcuni esempi

### RISC CMPE 414/CMSC 691V

Name	Opcode	Format	Notes
SRL (ShiftRghtLogic)	01111	SRL Rd Rs #	Store in register Rd the <b>zero-extended</b> value of Rs shifted to the right by the 2-bit immediate.
SLL (ShiftLeftLogic)	10000	RLL Rd Rs #	Store in register Rd the <b>zero-extended</b> value of Rs shifted to the left by the 2-bit immediate.
SW (StoreWord)	10001	SW 0 Rs Raddr	Store to memory at address Raddr the value in Rs. (You can assume that address is an even number and the compiler inserts 3 zero bits for Rd).
MUL (Multiply)	10010	MUL Rd Rs1 Rs2	Multiply the lower 8 bits of Rs1 and Rs2 and store the result in Rd.
Unused	10011		
LW (LoadWord)	10100	LR Rd 0 Raddr	Load the word value at memory address Raddr into Rd. (You can assume that address is an even number and the compiler inserts 3 zero bits for Rs).
LBI (LoadByImmed)	10101	LBI Rd #	Store the 8-bit <b>sign-extended</b> immediate into register Rd after sign extending it.
LBIU (LoadByIUnsign)	10110	LBIU Rd #	Write the 8-bit <b>zero-extended</b> immediate into register Rd.
LHI (LoadHighImmed)	10111	LHI Rd #	Write the 8-bit immediate into the upper 8 bits of register Rd and optionally clear the low order 8 bits.
BEQZ (BrIfEqZero)	11000	BEQZ Rs #	Set PC to PC + 8-bit <b>sign-extended</b> immediate if the value in register Rs is zero.
BNEZ (BrIfNotZero)	11001	BNEZ Rs #	Set PC to PC + 8-bit <b>sign-extended</b> immediate if the value in register Rs is non-zero.
BC (BrIfCarrySet)	11010	BC #	Set PC to PC + 11-bit <b>sign-extended</b> immediate if the carry out is set.
BO (BrIfOvrfowSet)	11011	BO #	Set PC to PC + 11-bit <b>sign-extended</b> immediate if the overflow is set.
NOP (NoOperation)	11100	NOP	Do nothing.
J (Jump)	11101	J #	Set the PC to PC + 11-bit <b>sign-extended</b> immediate.
JR (JumpToRegister)	11110	JR 0 Rs	Set the PC to the value in register Rs.
JALR (JumpAndLink)	11111	JALR Rd #	Rd=PC, PC=PC+. Save the current value of PC (which points to the NEXT instruction) to register Rd and set PC to PC + 8-bit <b>sign-extended</b> immediate.

# GAP semantico...



**Problema:** Come facciamo a compilare un linguaggio di alto livello affinchè possa essere eseguito in maniera efficiente sul calcolatore?

**Due possibili risposte:**

1. **CISC**: avere una architettura molto complessa che includa un numero elevato di istruzioni e di modi di indirizzamento e che includa anche istruzioni molto vicine a quelle presenti nel linguaggio di alto livello (...ricordate la storia dell'interprete della prima lezione??)
2. **RISC** : semplificare il set delle istruzioni e adeguarlo alle necessità dell'utente

# Formato delle Istruzioni

## Alcuni esempi CISC e RISC

### CISC Architectures:

#### VAX 11/780

Nr. of instructions: 303

Instruction format: not fixed

Addressing modes: 22

Number of general purpose registers: 16

#### Pentium

Nr. of instructions: 235

Instruction format: not fixed

Addressing modes: 11

Number of general purpose registers: 8

### RISC Architectures:

#### Sun SPARC

Nr. of instructions: 52

Instruction format: fixed

Addressing modes: 2

Number of general purpose registers: up to 520

#### PowerPC

Nr. of instructions: 206

Instruction format: not fixed (but small differences)

Addressing modes: 2

Number of general purpose registers: 32

# Esecuzione di una istruzione

1. Preleva la successiva istruzione dalla memoria e la porta nell'IR
2. Modifica il valore del PC per farlo puntare alla successiva istruzione
3. Determina il tipo di istruzione appena prelevata
4. Se l'istruzione usa una parola in memoria, determina dove si trova
5. Se necessario, preleva la parola dalla memoria per portarla in un registro della CPU
6. Esegue l'istruzione
7. Torna al punto 1 per iniziare l'esecuzione dell'istruzione successiva.

# Esecuzione di una istruzione

L'esecuzione di una istruzione avviene attraverso le seguenti fasi:

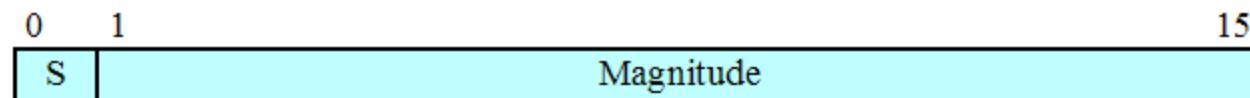
1. **INSTRUCTION FETCH** (Acquisizione dell'istruzione): il processore preleva l'istruzione dalla memoria, specificato dal registro PC (Programm counter), e la carica nell'IR. Viene incrementato il valore del PC.
2. **INSTUCTION DECODE** (Decodifica dell'istruzione): dall'istruzione prelevata viene determinata quale operazione debba essere eseguita e come ottenere gli operandi mediante la conoscenza dei codici operativi (operazione svolta dalla Control Unit)
3. **INSTRUCTION EXECUTE** (Esecuzione dell'istruzione): viene eseguita la computazione indicata
4. **MEMORY ACCESS**: nel caso in cui l'istruzione richieda un accesso alla memoria, questa fase sostituisce o segue la precedente.
5. **WRITE BACK**: scrittura del risultato in memoria

# Istruzioni

Consideriamo una macchina ipotetica



(a) Instruction format



(b) Integer format

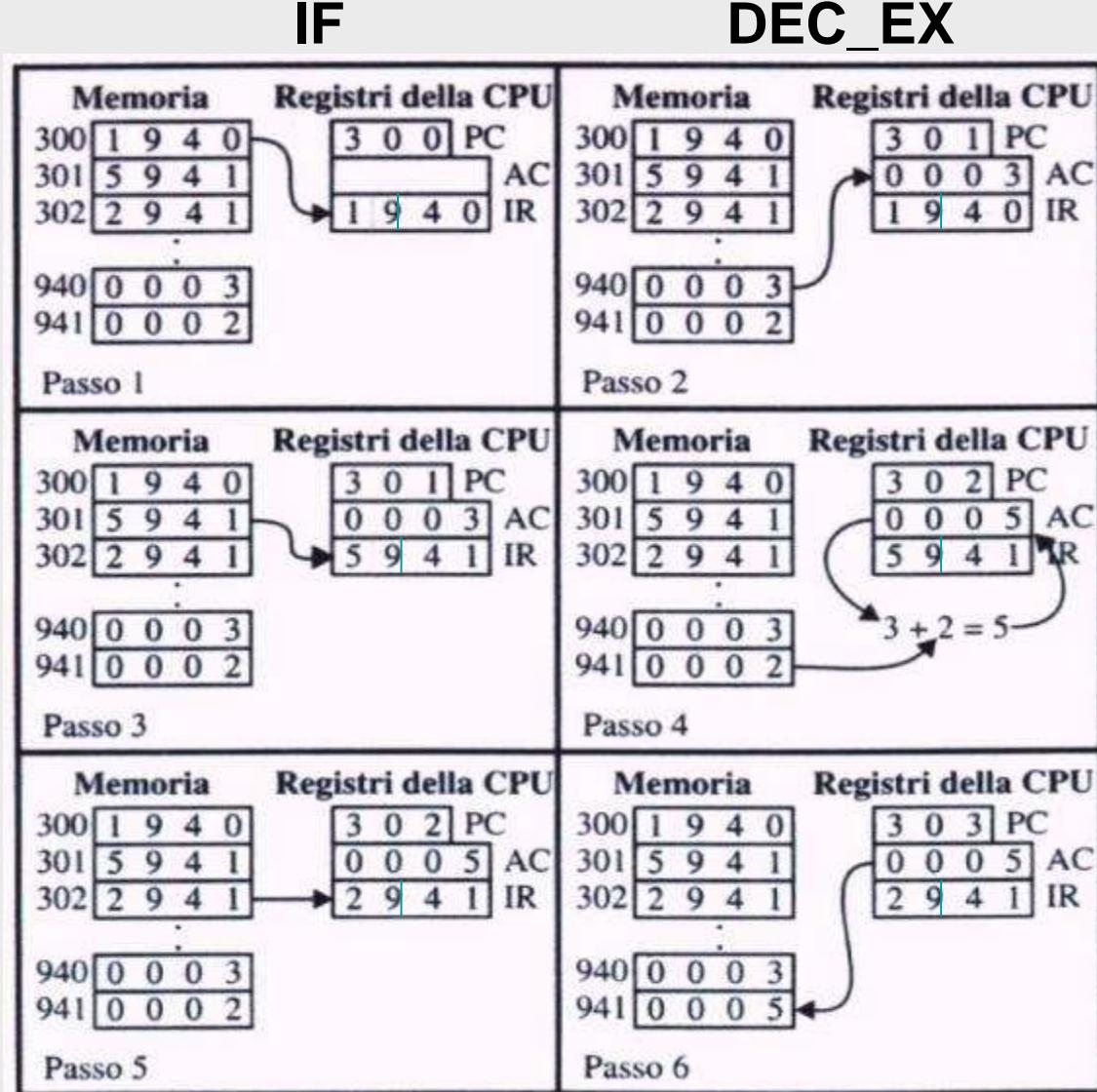
Program Counter (PC) = Address of instruction  
Instruction Register (IR) = Instruction being executed  
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory  
0010 = Store AC to Memory  
0101 = Add to AC from Memory

(d) Partial list of opcodes

# Esempio di Esecuzione IF-DEC\_EX



## Ciclo IF-DEC\_EX

### Legenda:

PC Program Counter

AC Accumulator

IR Instruction Register

### Codici operazioni:

1 Carica in AC

2 Salva in ...

5 Somma ad AC

# Esempio di Esecuzione IF-DEC\_EX

## Registri CPU

PC Program Counter

AC Accumulator

IR Instruction Register

## Formato istruzione:

YYYY

X: codice operativo

YYY: riferimento a memoria

## Codici operazioni:

X=1: Carica da MEM in AC

X=2: Salva AC in MEM

...

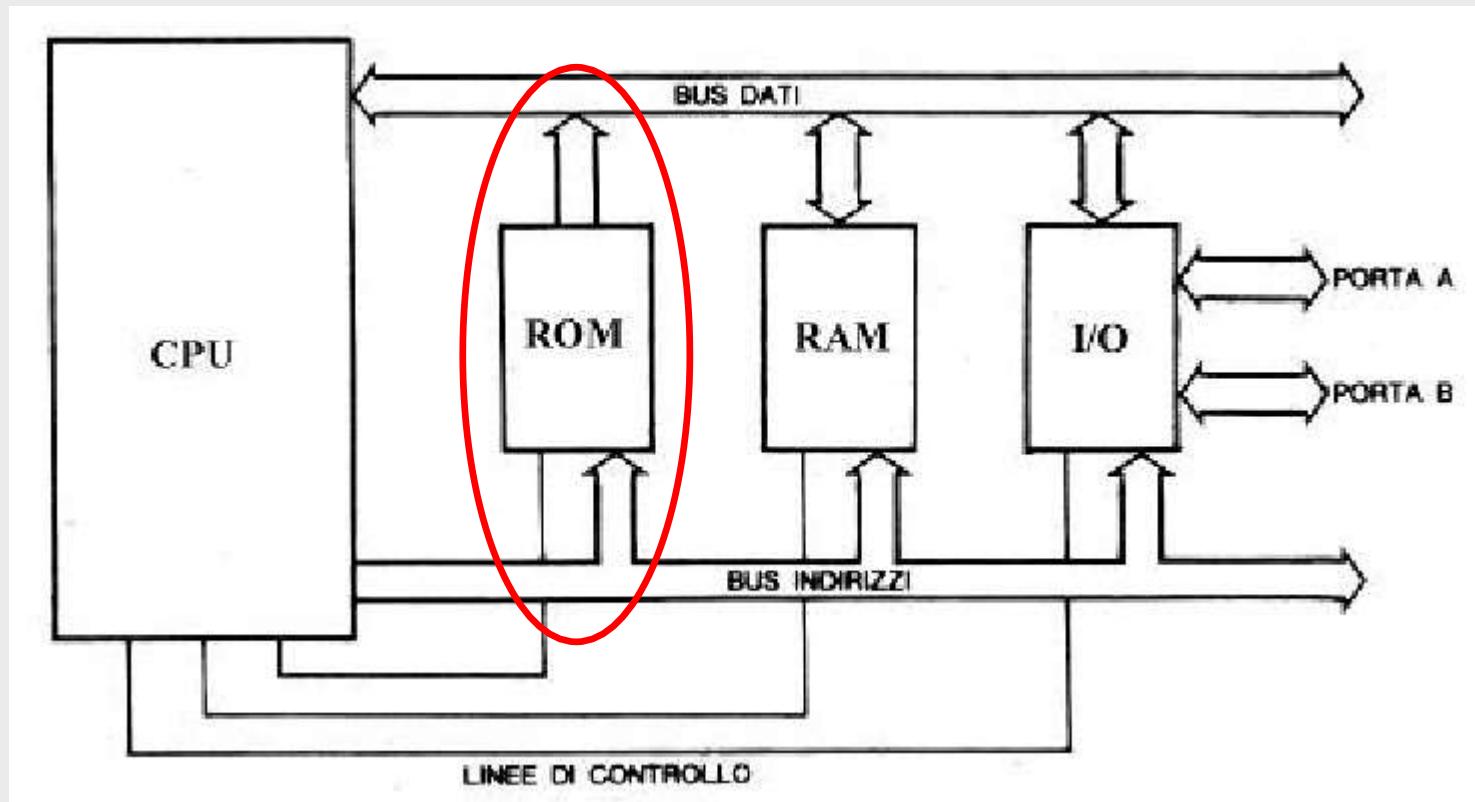
X=6 Sottrai ad AC contenuto MEM

## Ciclo IF-DEC\_EX

MEM		CPU	
212	1940	PC	212
213	6941	IR	
214	2942	AC	0021
	...		
940	0037		
941	0002		
942	0701		

*Si mostri il contenuto dei registri della CPU e della memoria nelle fasi di FETCH e DECODE-EXECUTE con riferimento all'esecuzione sequenziale delle istruzioni contenute in memoria nelle celle 212, 213, 214*

# ROM: Read Only Memory



# ROM (Read Only Memory)

Memoria permanente di sola lettura, scritta in fase di fabbricazione dal costruttore. Non più modificabile.

Contiene:

- └ le informazioni “di base” (la cui modifica comprometterebbe l’uso della macchina)
- └ le istruzioni del programma di avviamento (fase di bootstrap) che si attiva all'accensione della macchina.

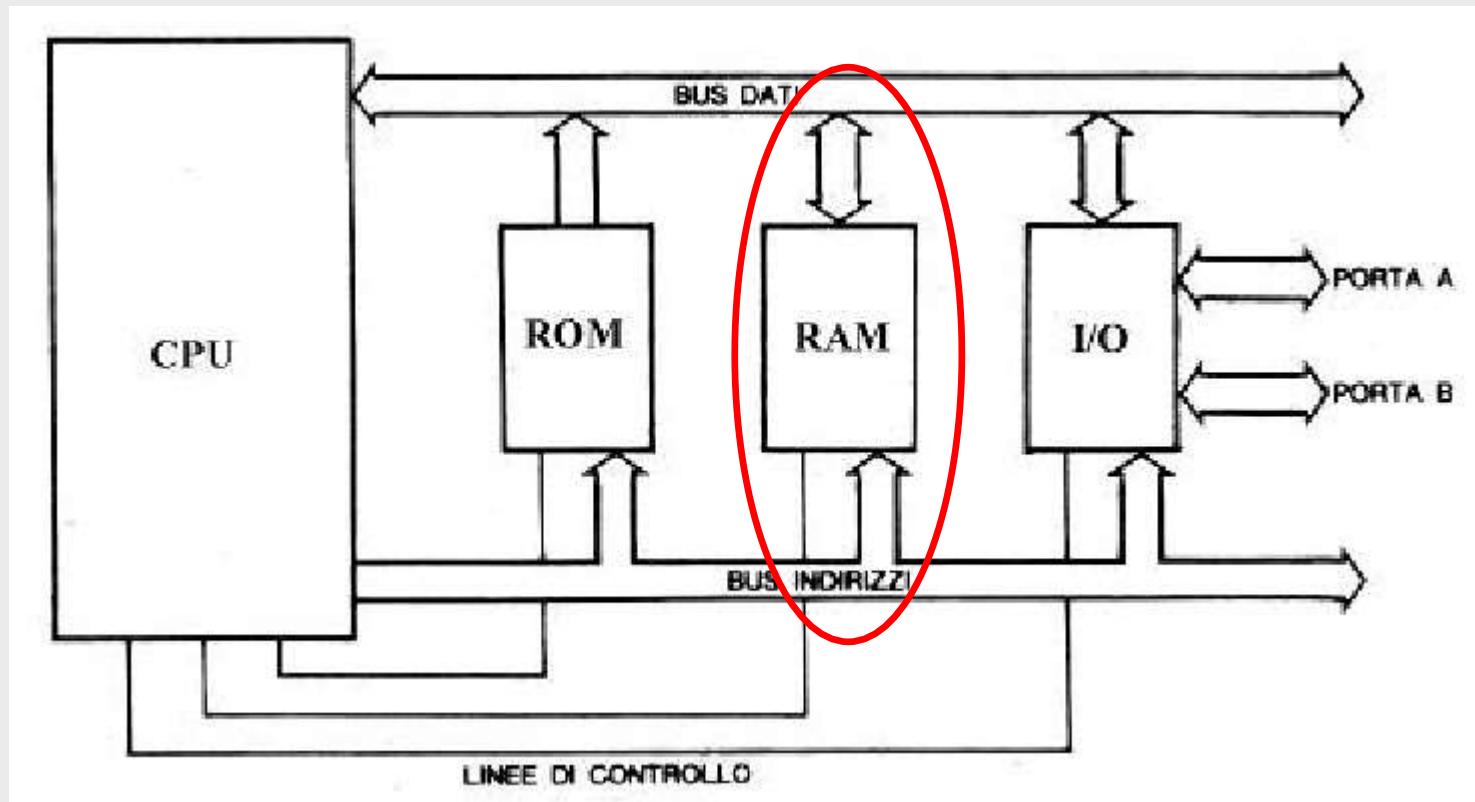


**BIOS (Basic Input/Output System):** sequenza di istruzioni di avvio eseguita automaticamente ad ogni accensione del computer. Si compone di 3 fasi:

- Attivazione dell'hardware installato e test di funzionamento del sistema (verifica dell'hardware)
- Verifica della presenza del sistema operativo e suo caricamento
- Avvio del primo processo

EPROM, (Electric–Programmable ROM) realizzate in modo da consentire sia la cancellazione che la riscrittura del contenuto. Cancellazione per mezzo di raggi ultravioletti..

# RAM: Random Access Memory



# RAM: Random Access Memory

Memoria nella quale vengono conservate le istruzioni del programma attualmente in esecuzione e i suoi dati. Durante l'esecuzione i programmi e i dati devono trovarsi almeno parzialmente nella memoria centrale.

## Memoria volatile

I documenti che si costituiscono utilizzando un qualsiasi programma vengono posti all'interno della RAM e sono trasferiti sul disco (Hard Disk) solo quando l'utente ne richiede il salvataggio. Se manca la corrente, con lo svuotamento della RAM tutto il lavoro svolto dopo l'ultimo salvataggio viene perduto.

La capacità della RAM si misura in GigaByte (GB), ovvero miliardi di byte.

Dimensioni tipiche: 2 – 16 Gbyte

Frequenza di lavoro: più alta è la frequenza, più velocemente la memoria sarà accessibile. MT/s = milioni di trasferimenti al secondo

# RAM: Random Access Memory

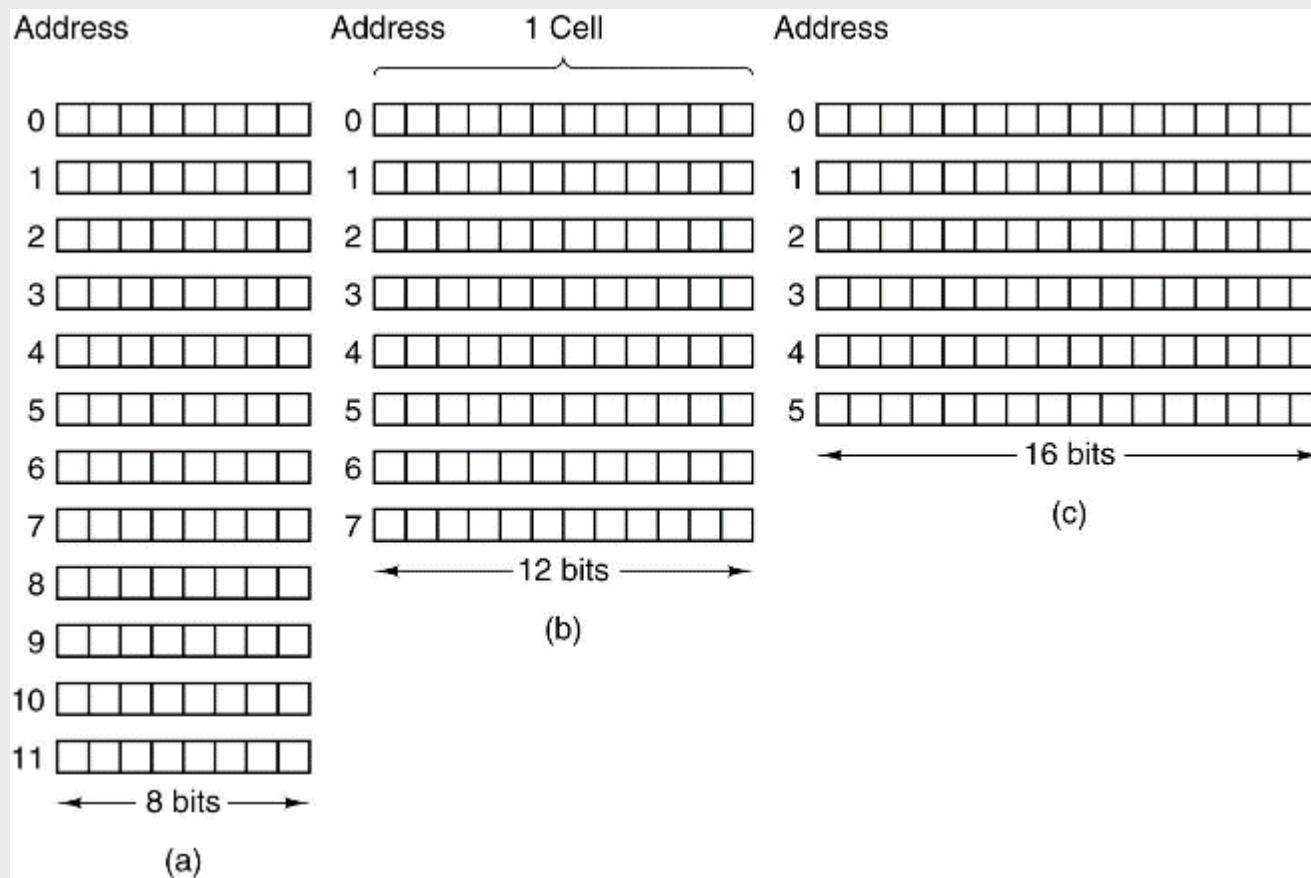
*Le memorie sono costituite da un certo numero di **celle** (o **locazioni**) ciascuna della quali può memorizzare informazioni.*

*Ciascuna cella ha un numero, chiamato **indirizzo**, attraverso il quale un qualsiasi programma può riferirsi ad essa. Se una memoria ha  $n$  celle, i suoi indirizzi variano da 0 a  $n-1$ .*

*Tutte le celle contengono lo stesso numero di bit. Se una cella contiene  $k$  bit allora potrà assumere al massimo  $2^k$  valori.*

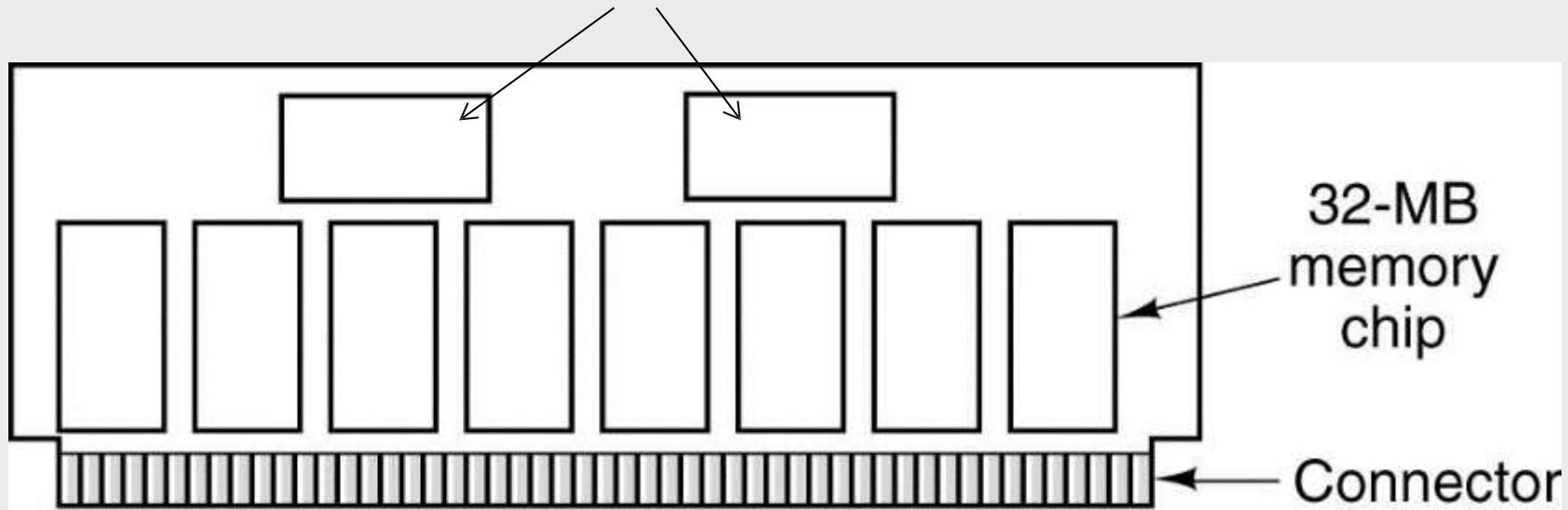
# RAM: Random Access Memory

Indirizzi di memoria e dimensione della “word”: tre modi di organizzare una memoria a 96 bit.



# Memory packaging and types

Two of the chips control the SIMM.



SIMM: Single Inline Memory Module - (tipical: 72 pin -- 32 bit/cycle)

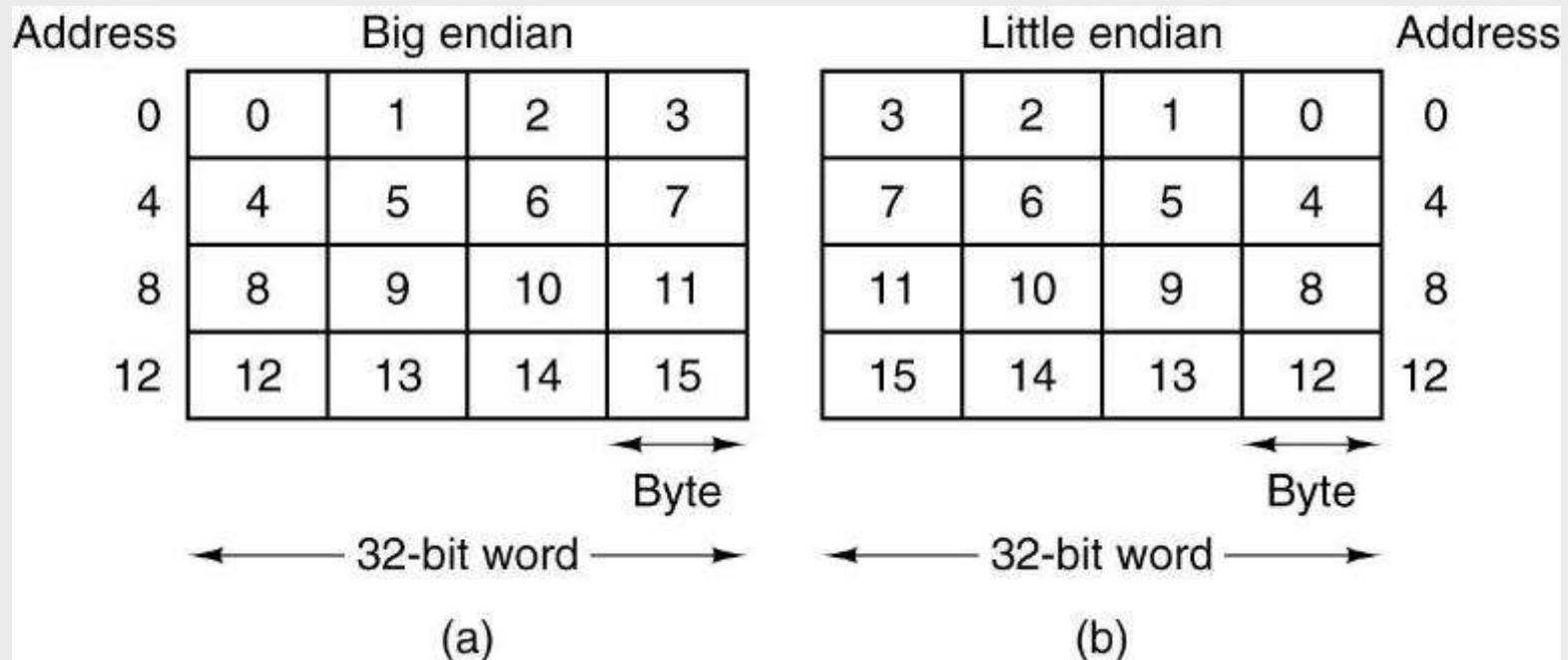
DIMM: Dual Inline Memory Module - (tipical: 84 pin -- 64 bit/cycle)

In un PC se ogni SIMM /DIMM è composta da 8 moduli da 32MB ciascuno si ha:  
32MB x 8 Chip = 256 MB (4 SIMM → 1GB)

# Ordinamento dei Byte

All'interno di una parola di memoria i byte possono essere numerati :

- da sinistra a destra (Big endian – dal byte più significativo al meno significativo)
- da destra a sinistra (Little endian – dal byte meno significativo al più significativo)



# Trasferimento dei Byte

Big endian				Little endian				Transfer from big endian to little endian				Transfer and swap					
0	J	I	M			M	I	J		M	I	J		J	I	M	
4	S	M	I	T	T	I	M	S		T	I	M	S	S	M	I	
8	H	0	0	0	0	0	0	H		0	0	0	H	H	0	0	
12	0	0	0	21	0	0	0	21		21	0	0	0	0	0	0	21
16	0	0	1	4	0	0	1	4		4	1	0	0	0	0	1	4
	(a)				(b)					(c)					(d)		

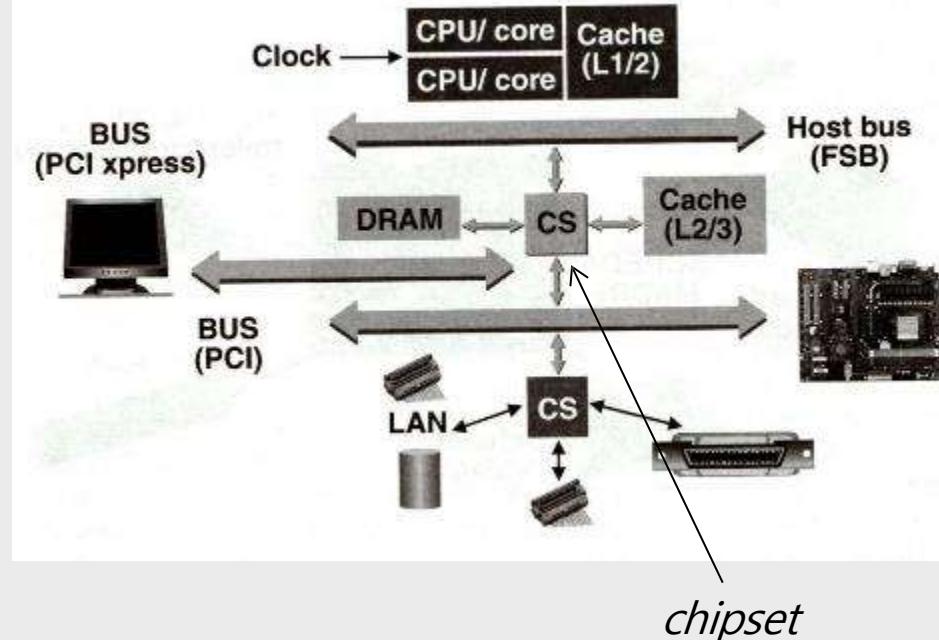
- (a) Macchina “big endian”.
  - (b) Macchina “little endian”.
  - (c) Risultati del trasferimento dati da “big endian” a “little endian”.
  - (d) Byte-swapping di (c).

# RAM :Random Access Memory

Front Side Bus (FSB): trasporta i dati tra la CPU e i dispositivi veloci, tra cui la RAM, L3\$, la memoria della scheda video.

La banda (o throughput) del FSB viene determinata moltiplicando:

- i byte delle word (parole) del processore
- la frequenza di clock (cicli al secondo) del BUS
- il numero di data transfer del BUS ad ogni ciclo.



Es: un sistema con:

- processore a 32 bit (4 byte);
- FSB a 100 MHz;
- 4 trasferimenti a ciclo;

ha una banda di  $4(\text{byte}) \times 100(\text{FSB}) \times 4(\text{tc}) = 1600$  megabyte al secondo (MB/s).

Memoria	Clock	Frequenza I/O (FSB)	Velocità trasferimento dati
DDR3 800	100 MHz	400 MHz	800 MT/s
DDR3 1600	200 MHz	800 MHz	1600 MT/s (=1.6GT/s)

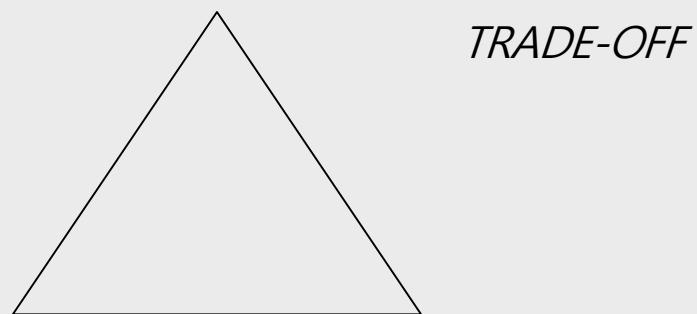
Intel core i5 4.8GT/s  
(trasferimenti al secondo)!!!!!!!

**Necessità di una memoria più veloce!**

# La Memoria

- Parametri:
  - Dimensione (grande)
  - Velocità (elevata)
  - Costo (piccolo)

**Parametri sono in contrasto tra loro!**



- Vincoli tecnologici:
  - Tempo di accesso più breve, maggior costo per bit
  - Maggiore capacità, maggiore tempo di accesso
  - Maggiore capacità, minore costo per bit

# Gerarchie delle Memorie

Soluzione alla inconciliabilità dei tre parametri

- Uso organizzato e integrato di diversi dispositivi di memoria con diverse caratteristiche
- Consente di:
  - diminuire il costo totale per bit
  - aumentare la capacità del sistema
  - contenere i tempi medi di accesso
  - diminuire la frequenza di accesso ai dispositivi più lenti

Scendendo nella gerarchia:

- └ Diminuisce il costo per bit
- └ Aumenta la capacità
- └ Aumenta il tempo di accesso
- └ Diminuisce la frequenza di accesso del processore

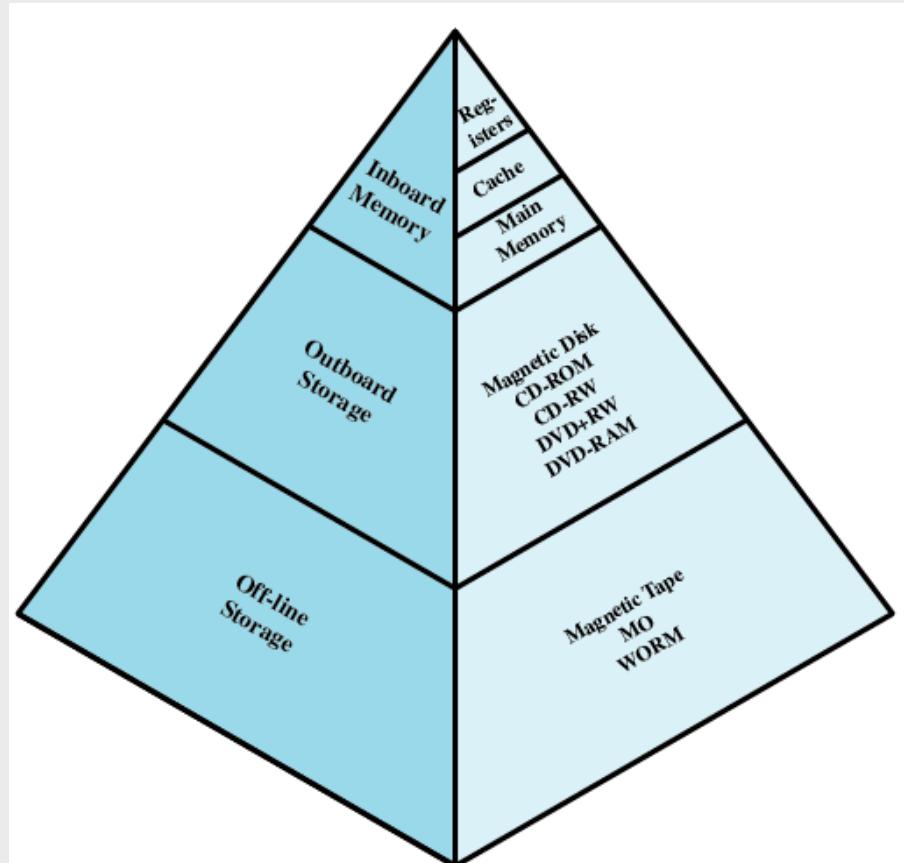


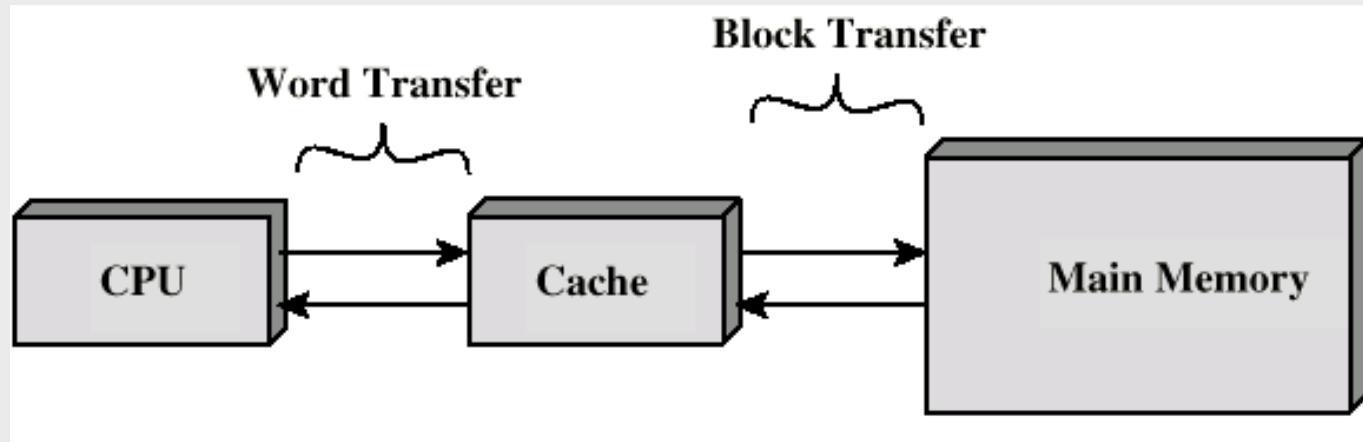
Figure 1.14 The Memory Hierarchy

# Gerarchie delle Memorie

Memoria	Standard	Clock	Frequenza I/O	Velocità trasferimento dati	Banda per canale	Banda dual channel
DDR3 800	PC3-6400	100 MHz	400 MHz	800 MT/s	6,4 GB/s	12,8 GB/s
DDR3 1066	PC3-8500	133 MHz	533 MHz	1066 MT/s	8,5 GB/s	17,0 GB/s
DDR3 1333	PC3-10600	166 MHz	667 MHz	1333 MT/s	10,6 GB/s	21,2 GB/s
DDR3 1600	PC3-12800	200 MHz	800 MHz	1600 MT/s	12,8 GB/s	25,6 GB/s

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

# Memoria CACHE - \$



- Contiene i dati di utilizzo più ricorrente in modo che il processore non li debba cercare nelle aree della memoria centrale.
- Memoria ad accesso molto veloce
- Serve a compensare la differenza di velocità di accesso e di trasferimento dei dati tra la CPU e la memoria RAM

Osservazione:

*Il buon funzionamento della cache deriva dalla capacità del sistema nel mantenere in essa le informazioni che saranno necessarie.*

# Memoria CACHE - \$

**Principio di Località (spaziale):** se in un certo istante viene referenziato un indirizzo di memoria è altamente probabile che in istanti immediatamente successivi possano venire referenziati indirizzi vicini.

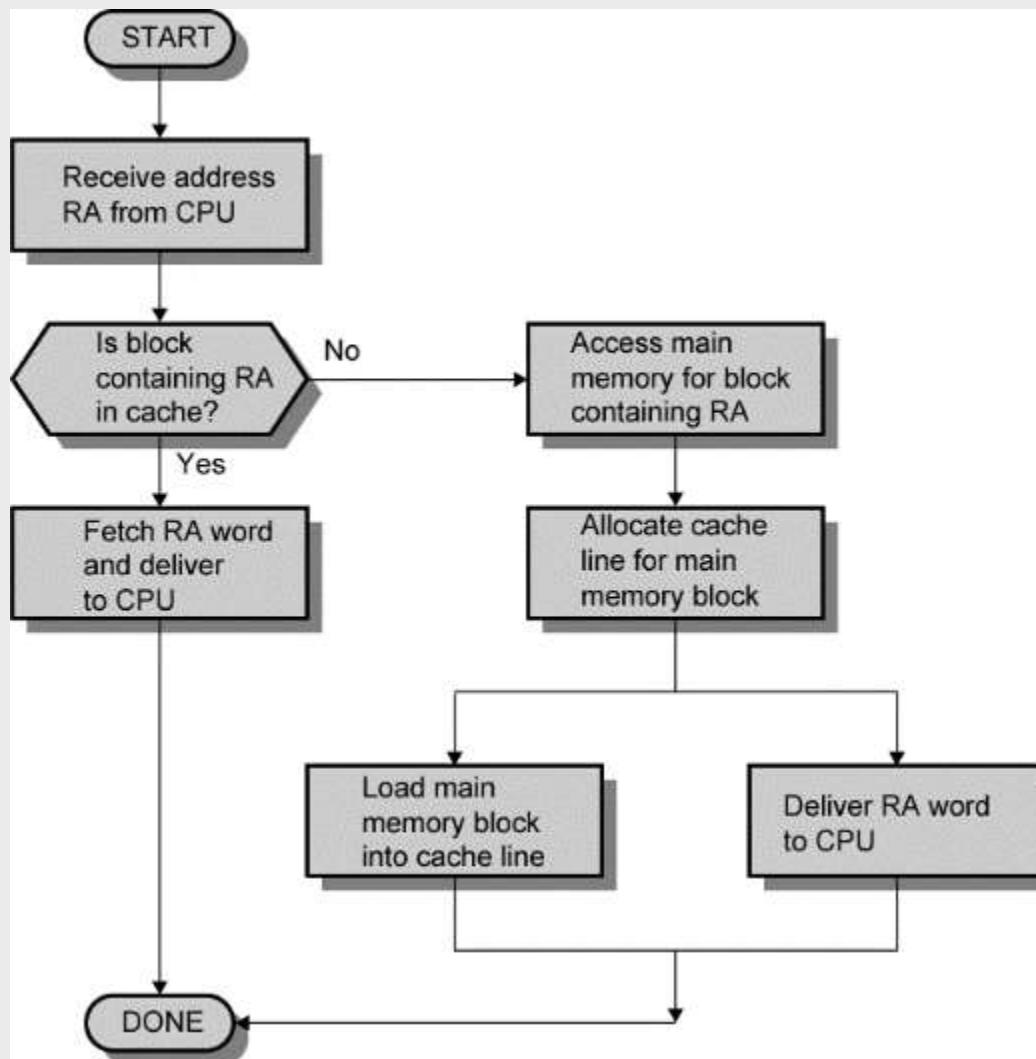
**Principio di Località (temporale):** se in un certo istante viene referenziato un indirizzo di memoria è altamente probabile che in istanti immediatamente successivi lo stesso indirizzo possa essere nuovamente referenziato.

I riferimenti alla memoria fatti in un breve intervallo di tempo tendono ad utilizzare solo una piccola parte della memoria totale

# Motivazioni per la CACHE - \$

- In tutti i cicli di istruzione il processore accede alla memoria principale almeno una volta (prelievo dell'istruzione IF)
- La velocità del processore è limitata dalla velocità della memoria
- La velocità della memoria centrale (per motivi tecnici e/o economici) è minore di quella del processore
- Sfruttando il principio di località si inserisce una memoria piccola e veloce (la cache) fra processore e memoria centrale
- Obiettivo
  - fornire una memoria con velocità prossima a quella del processore
  - disporre di una quantità di spazio sufficiente per non rallentare il processore
  - contenere i costi delle memorie del sistema
- Contenuto:
  - una copia di una porzione della memoria centrale

# Lettura da CACHE - \$



# Memoria CACHE - \$

Se durante un piccolo intervallo di tempo una parola è letta o scritta  $k$  volte, il calcolatore dovrà effettuare:

- 1 riferimento alla memoria (più lenta) (solo la prima volta) (+ 1 accesso, non riuscito, alla cache)
- $(k-1)$  riferimenti alla cache (più veloce)

Posto

$c$ =tempo di accesso alla cache

$m$ =tempo di accesso alla memoria

Si ottiene  $t_{memory} = km$  ;  $t_{cache} = (c+m)+(k-1)c$ .

Il vantaggio nell'uso della cache è quantificabile in:

$$\frac{t_{cache}}{t_{memory}} = \frac{(c+m)+(k-1)c}{km} = \frac{c+m}{km} + \frac{(k-1)}{k} \frac{c}{m}$$

# Memoria CACHE - \$

Nel caso più generale, posto:

$h = \text{hit ratio}$  (frequenza di successi nell'accesso alla cache)  
 $((1-h) = \text{miss ratio})$

Si ottiene che il tempo medio di accesso è pari a:

$$= h \cdot c + (1-h) \cdot (c+m) =$$

$$= hc + c - hc + (1-h)m =$$

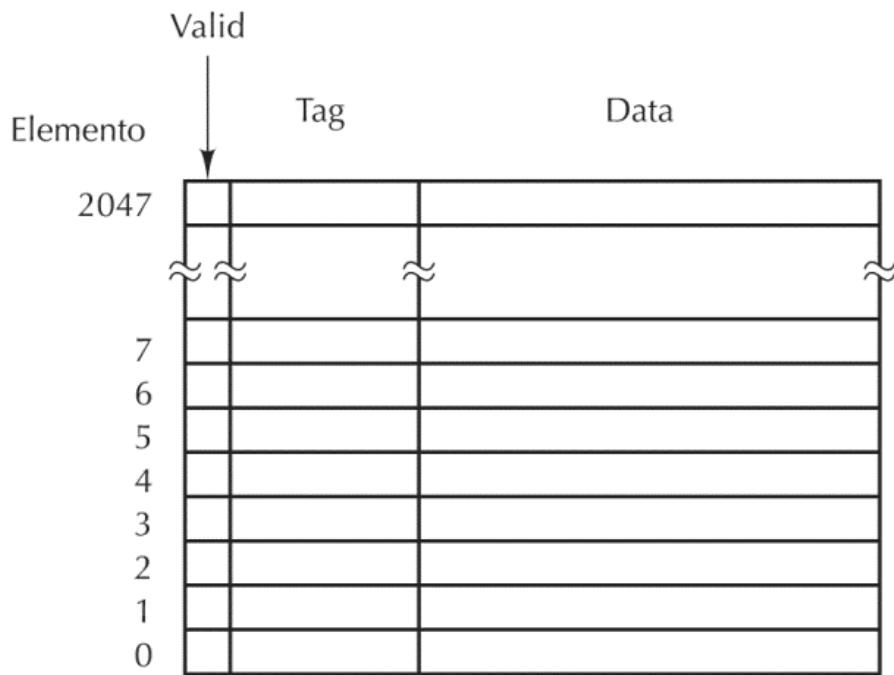
$$= c + (1-h)m$$

# Gestione della CACHE - \$

- Posizionamento di un blocco in cache:
  - Indirizzamento diretto
  - Fully associative
  - Set associative
- Algoritmo di rimpiazzo:
  - sceglie quale blocco rimpiazzare
  - sarebbe preferibile rimpiazzare il blocco meno necessario nel prossimo futuro (Politica di Belady o ottima)
  - Politica LRU (Last Recently Used)
- Politica di scrittura:
  - qualora i contenuti del blocco sono stati modificati bisogna riscriverlo in memoria centrale prima di rimpiazzarlo
  - sono possibili diverse politiche:
    - scrivere ogni volta che il blocco è aggiornato (write through)
      - » politica molto onerosa, ma mantiene la memoria costantemente aggiornata
    - scrivere quando il blocco è rimpiazzato (write back)
      - » ottimizza le scritture in memoria, ma può lasciare la memoria centrale non aggiornata per lunghi periodi

# Posizionamento di un blocco in cache

- **Indirizzamento diretto:** un blocco può essere messo in un solo punto della cache. L'indirizzo del blocco è ottenuto mediante l'operazione (indirizzo di blocco) % (numero di blocchi nella cache).



(a)

*Valid: indica se il dato è valido*

*Tag: fa riferimento alla linea di memoria da cui arrivano i dati*

*Data: contiene il dato di memoria*

# Posizionamento di un blocco in cache

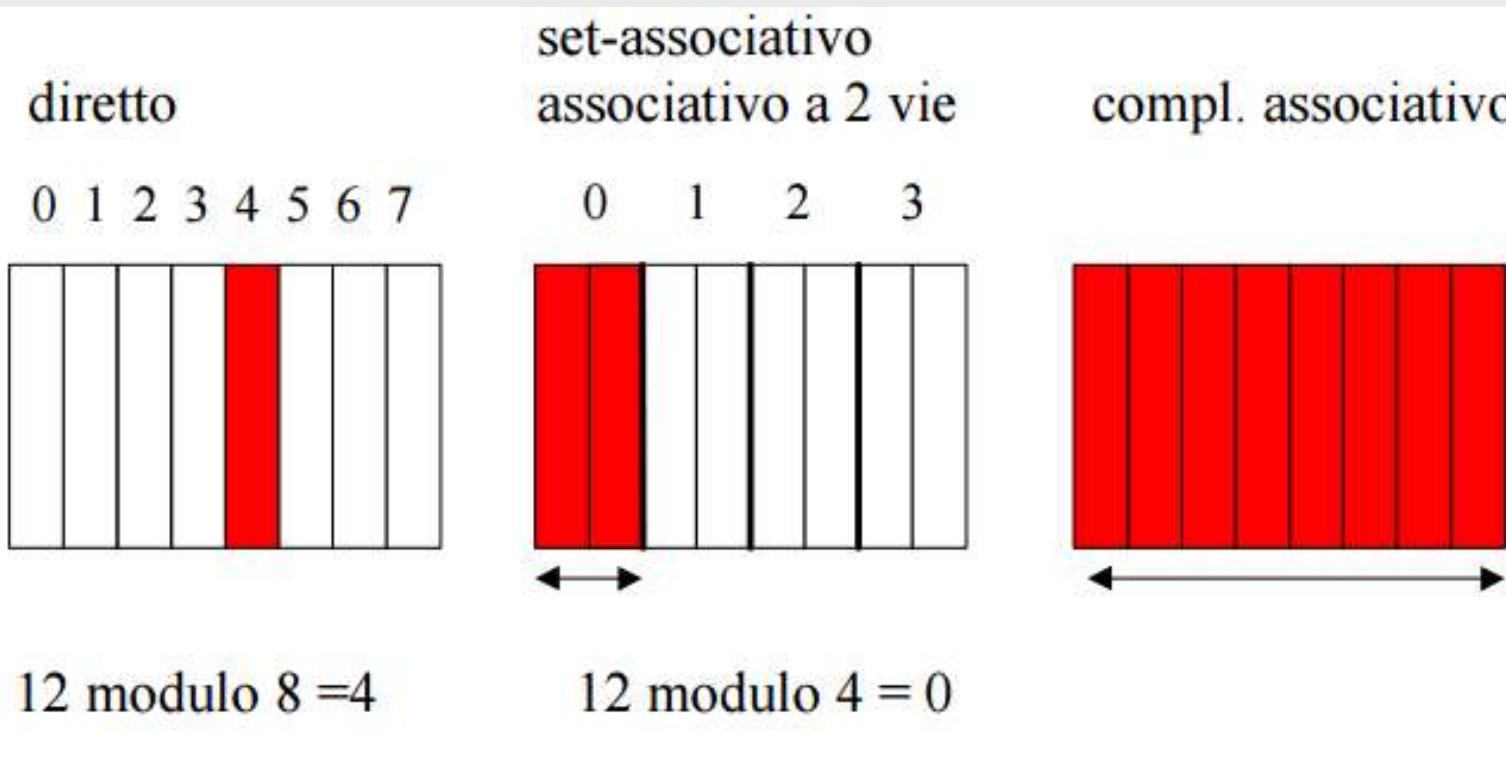
- Fully Associative (completamente associativa): un blocco può essere posto ovunque nella cache.
- PRO: molto facile da realizzare
- CONTRO: se i blocchi possono essere messi ovunque, avrò difficoltà nell'andare a ritrovarli o dovrò introdurre ulteriori meccanismi che mi ridurrebbero la facilità di implementazione

# Posizionamento di un blocco in cache

- **Set Associative:** un blocco può essere posto in un insieme ridotto di posizioni nella cache
  - Un insieme (set) è un gruppo di due o più blocchi della cache
  - Un blocco viene prima messo in corrispondenza di un insieme e poi può essere messo in qualsiasi posizione dell'insieme
  - L'insieme viene scelto con la regola del modulo: (indirizzo blocco) modulo (numero di insiemi della cache)
  - Se ci sono  $n$  blocchi in un insieme, il posizionamento viene definito come set-associativo a  $n$  vie

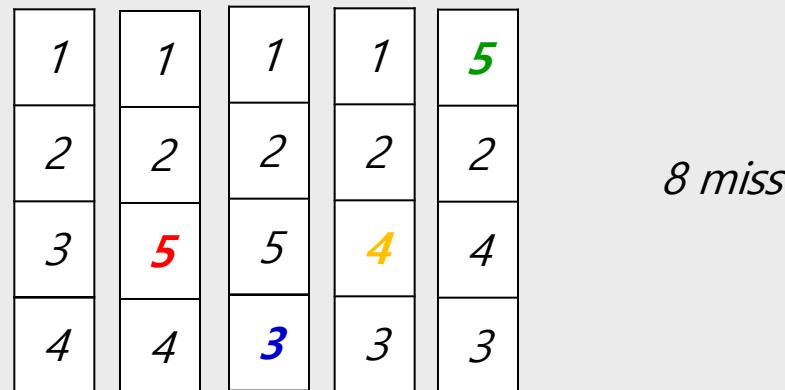
# Posizionamento di un blocco in Cache: un esempio

Esempio. Consideriamo il blocco di indirizzo 12 ed i tre tipi di indirizzamento in una memoria che può contenere 8 blocchi.



# Algoritmi di replacement: Least Recently Used (LRU)

- Si sostituiscono i blocchi meno recentemente utilizzati
  - Ipotesi: il blocco meno utilizzato recentemente è quello che ha la minore probabilità di essere referenziato in futuro. Si approssima il futuro prossimo al passato recente
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



# Algoritmi di replacement: Random

- I blocchi candidati per la sostituzione sono scelti in modo casuale
  - Facilità di realizzazione rispetto a LRU
  - È possibile verificare empiricamente che al crescere della dimensione della cache lo scarto di performance in termini di miss rate è minimo o nullo

Associatività	A 2 vie		A 4 vie		A 8 vie	
Dimensione	LRU	Casuale	LRU	Casuale	LRU	Casuale
16KB	5,18%	5,69%	4,67%	5,29%	4,39%	4,96%
64KB	1,88%	2,01%	1,54%	1,66%	1,39%	1,53%
256KB	1,15%	1,17%	1,13%	1,13%	1,12%	1,12%

Fonte: Hennessy-Patterson, i risultati si riferiscono ad una macchina VAX con cache a blocchi di 16bit. I dati osservati si riferiscono sia a programmi utente che OS

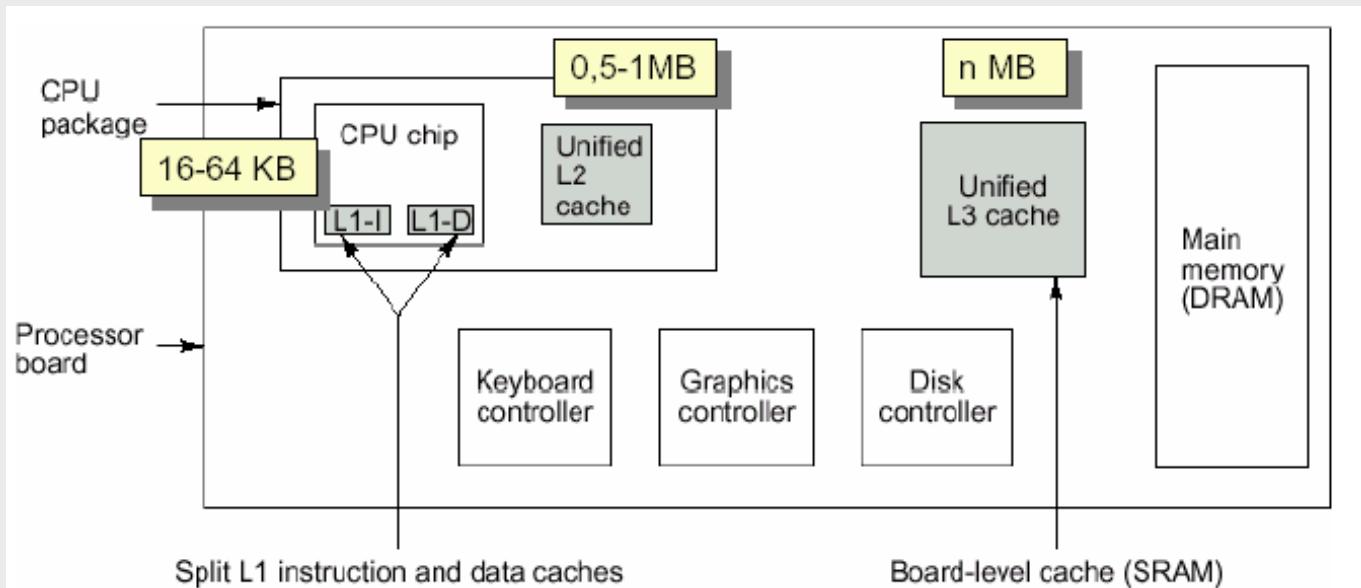
# WRITE POLICY

PROBLEMA: scrittura da CPU in \$, quando aggiornare la RAM?

POLITICHE:

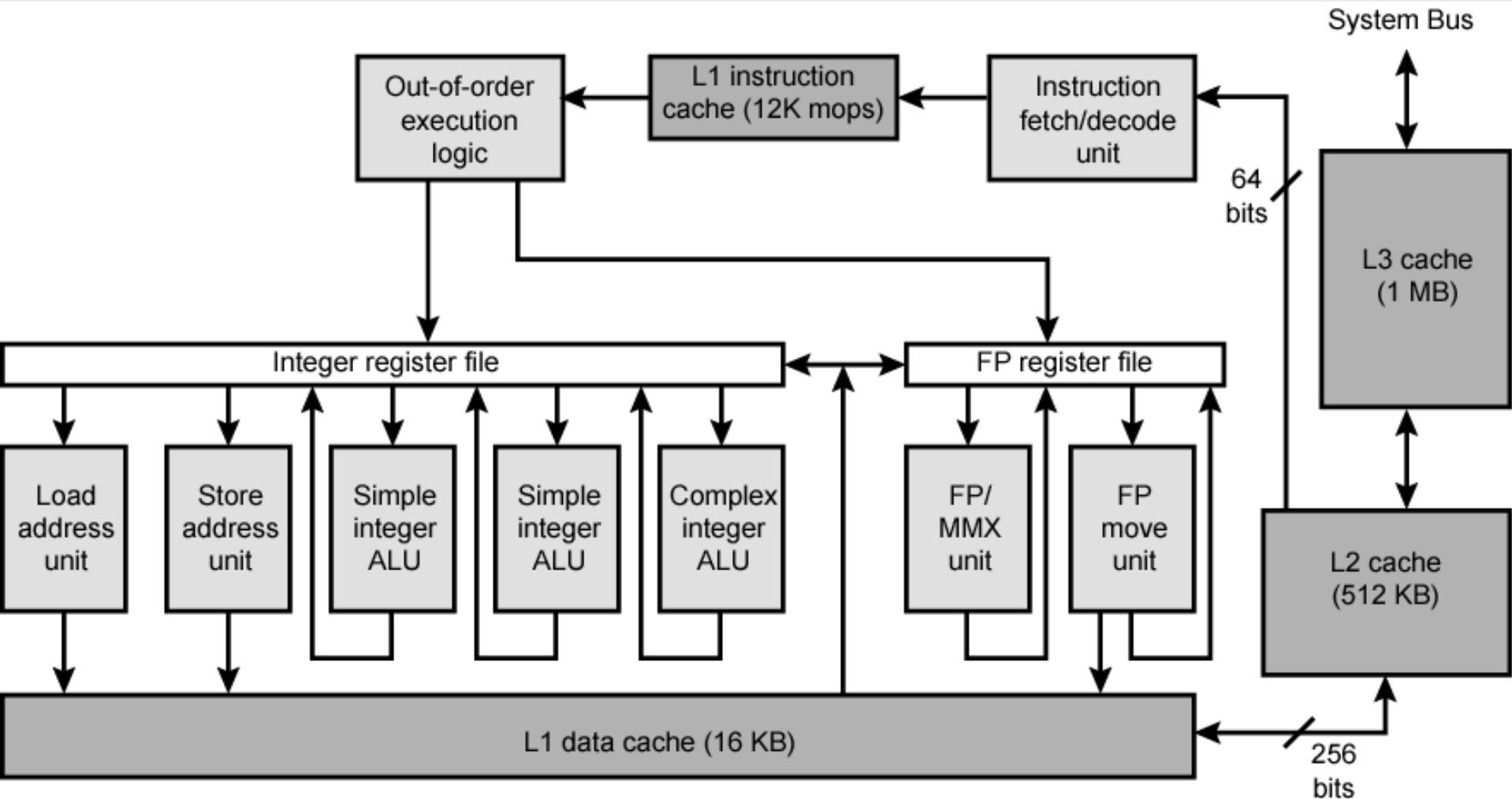
- WRITE THROUGH
  - All writes go to main memory as well as cache
  - Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
  - Lots of traffic
  - Slows down writes
- WRITE BACK
  - Updates initially made in cache only
  - Update bit for cache slot is set when update occurs
  - If block is to be replaced, write to main memory only if update bit is set
  - Other caches get out of sync
  - I/O must access main memory through cache
  - N.B. 15% of memory references are writes

# Memoria CACHE - \$



Dapprima la CPU cerca il dato nella cache di livello 1; se avviene un hit, il processore procede ad alta velocità. Se si verifica un miss, allora viene controllata la \$ di livello 2 e così via, fino ad accedere alla memoria principale.

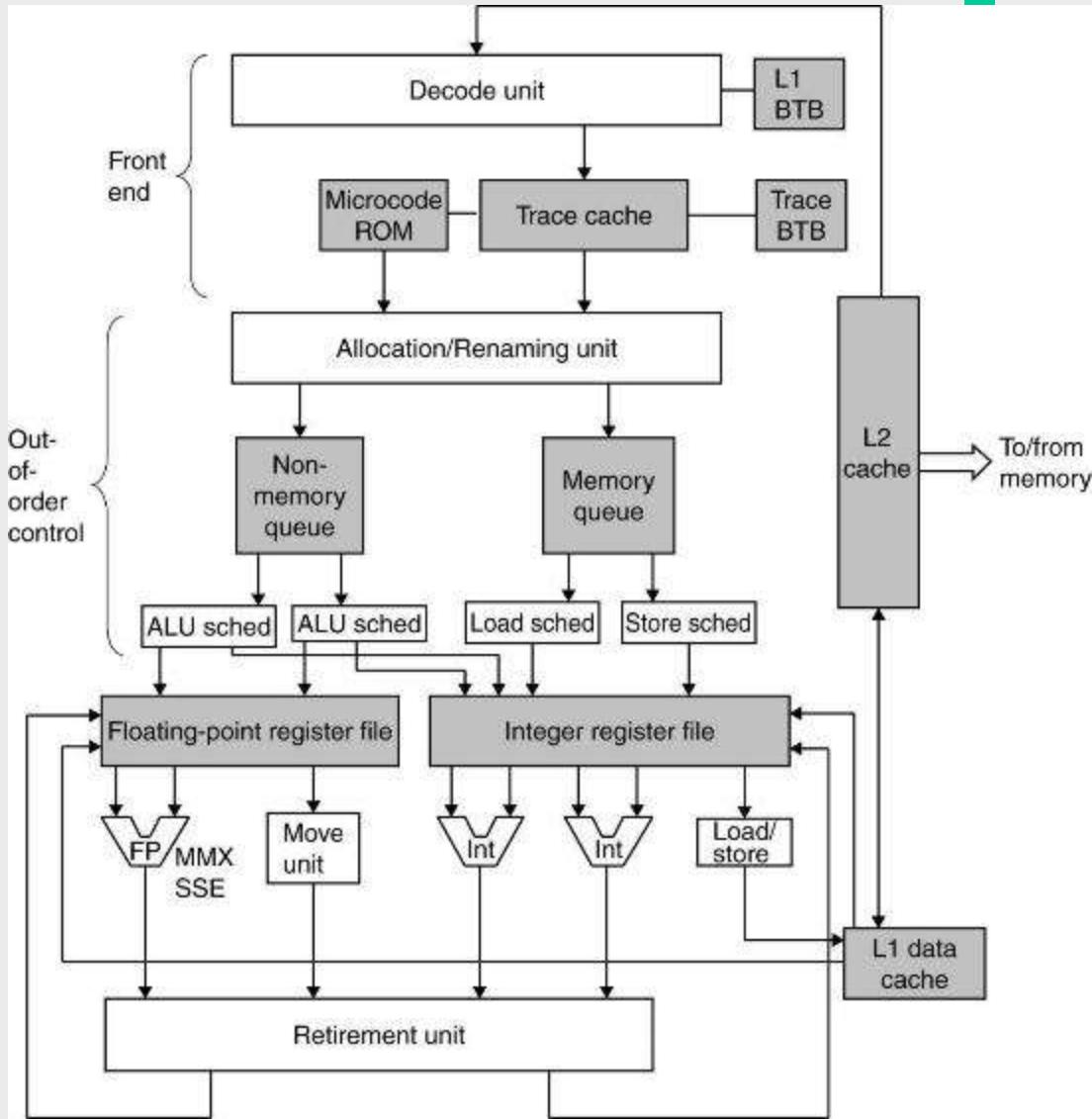
# Memoria CACHE - \$ Pentium 4



# Memoria CACHE - \$ Intel Core i7

Ogni core contiene:

- L2, L1 e la logica per accedere a L3 condivisa
- \$ di pre-fetch che cercano di caricare istruzioni prima che siano referenziate
- Front-end:
  - Prelievo, decodifica in formato RISC e salvataggio in \$ delle istruzioni
  - Le istruzioni prelevate da \$ L1 vengono passate ai decodificatori che determinano le operazioni da svolgere nella pipeline
- Unità di controllo fuori sequenza:
  - Tiene traccia delle micro-operazioni, se le risorse sono disponibili viene inserita in una delle due code altrimenti ritardata
  - Il fuori sequenza è dovuto alla pipeline



# Interruzioni

- Consentono di interrompere l'elaborazione normale del processore
- La funzione principale è migliorare l'efficienza della elaborazione (dispositivi periferici molto più lenti del processore)
  - Es. stampa su stampante
- Permettono al modulo di I/O, al termine del comando di I/O, di segnalare l'evento al processore

## Tipologie di interruzioni:

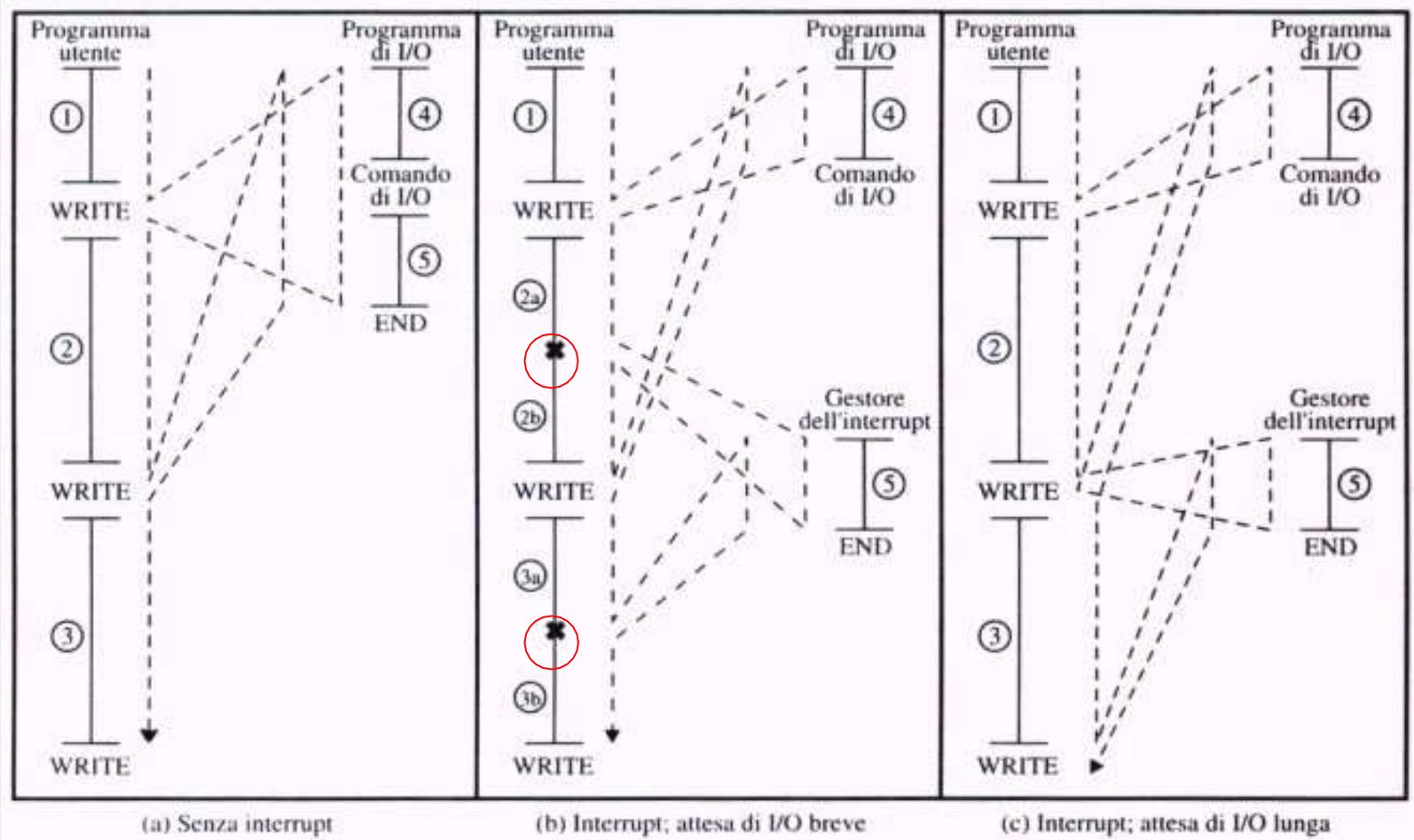
- Interrupt: generate da un dispositivo hardware
- Trap: generate da un programma in esecuzione

## *Linee controllo*

## Classi di interruzione:

- Programma
  - errore di esecuzione (overflow, divisione per zero, violazione spazio di memoria)
- Timer
  - operazioni pianificate (time slot)
- I/O
  - operazioni di I/O
- Errore hardware
  - problemi fisici (caduta di tensione)

# Flusso di controllo con e senza interrupt



WRITE – chiamata di sistema

4 – Preparazione alle op. di I/O, es. copiatura dati da inviare alla stampante

5 – Completamento op. I/O, es. settaggio flag che indica l'esito della operazione

# Gestione dell'interrupt

*Il programma utente non contiene alcun comando speciale per la gestione della operazione di I/O*

*Il programma di gestione dell'interrupt è parte del SO:*

- Determina il tipo di interruzione
- Chiama il programma che gestisce tale evento

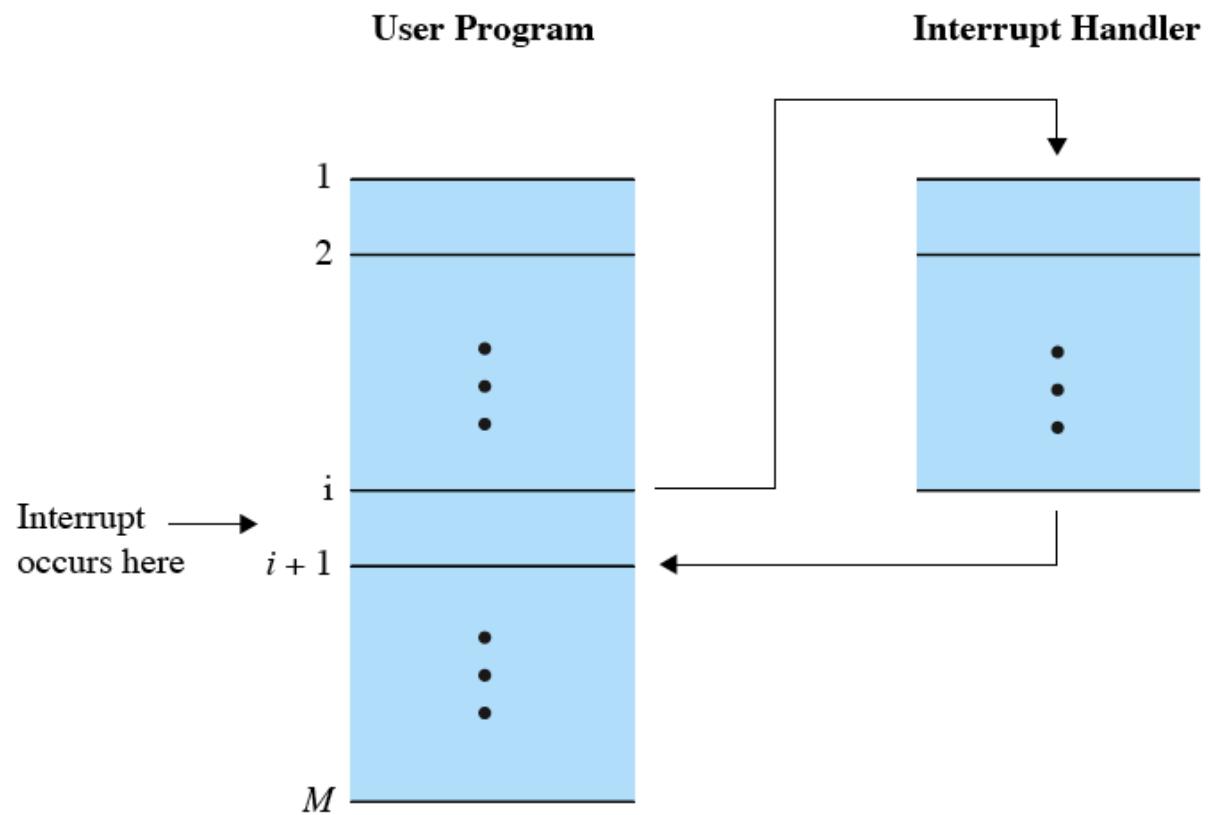
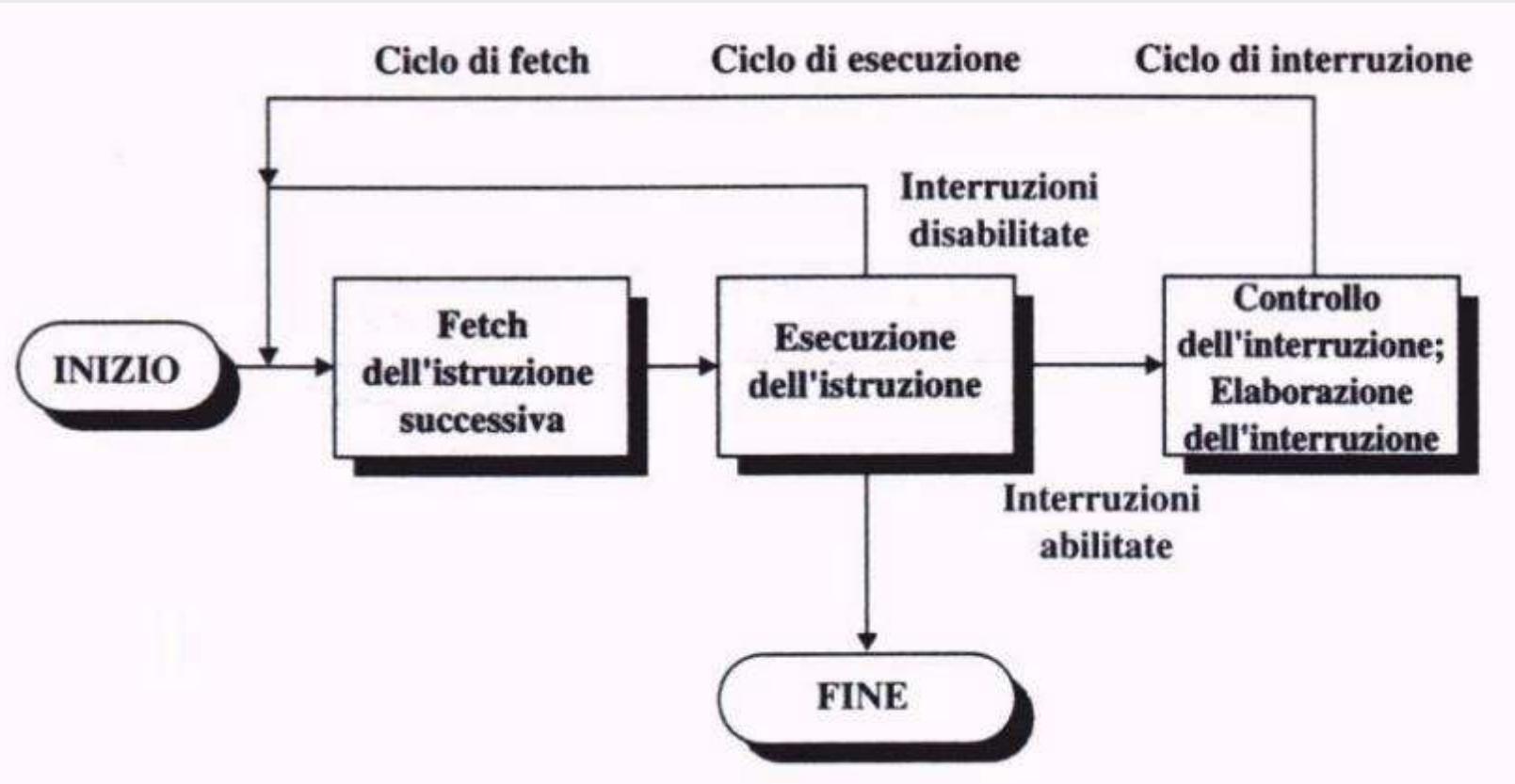


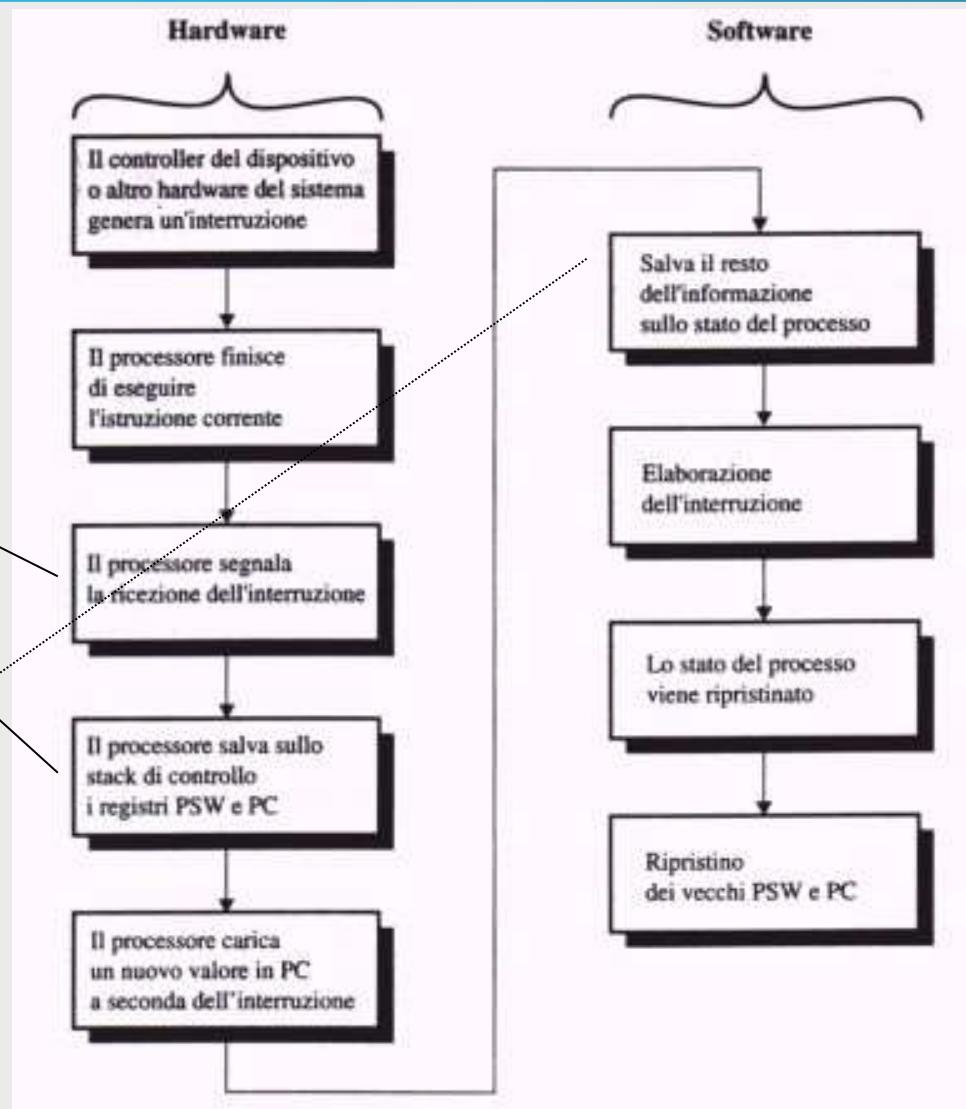
Figure 1.6 Transfer of Control via Interrupts

# Ciclo di esecuzione con interruzione



NB: in questo schema il sistema controlla se ci sono interruzioni solo al termine della fase di exe

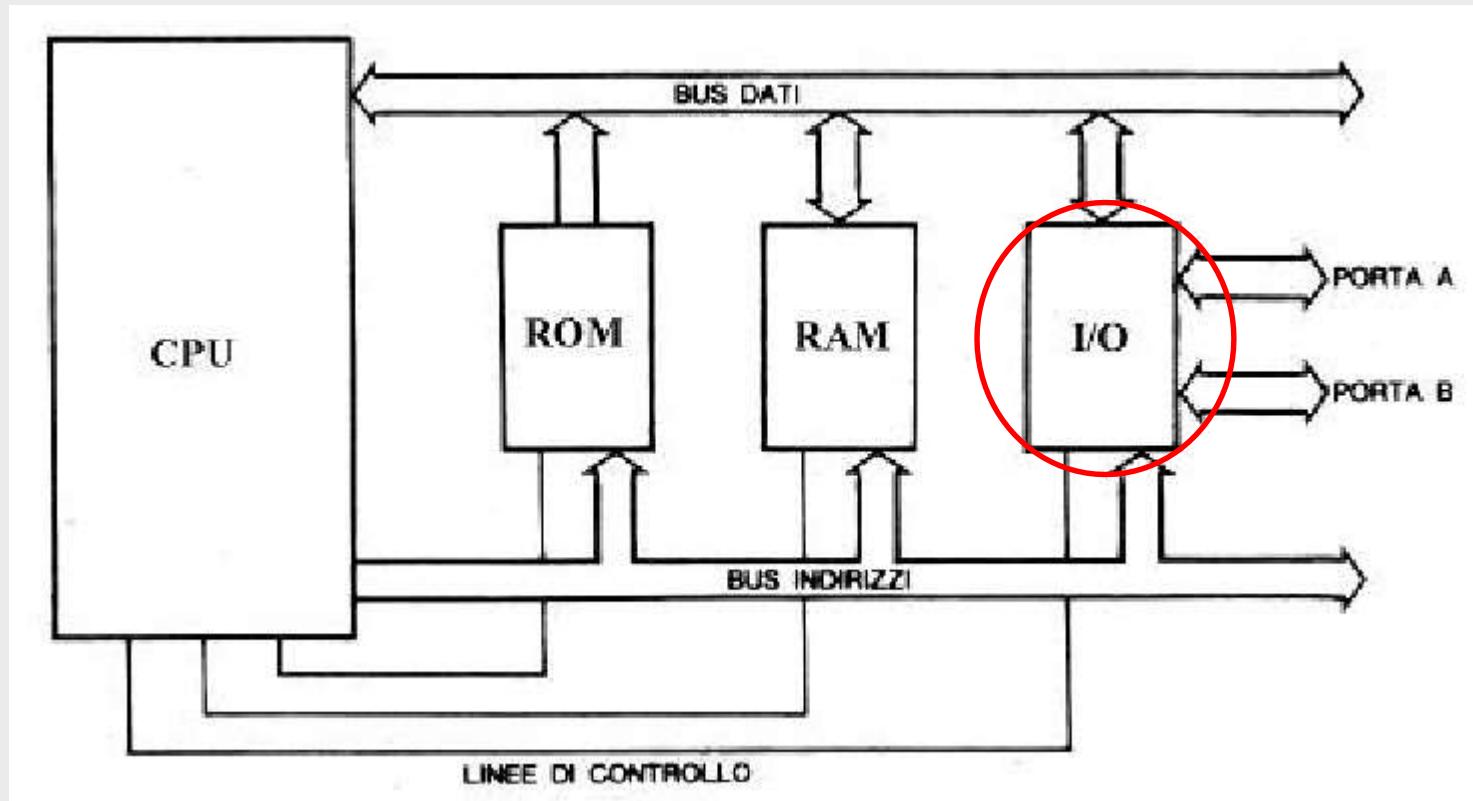
# Elaborazione delle interruzioni



Il programma che gestisce l'interruzione è parte del SO:

- Determina la natura dell'interruzione
- Chiama il modulo che la gestisce

# Macchina di Von Neuman: Dispositivi di I/O



# Dispositivi di I/O

Principali dispositivi di input:

tastiera  
mouse

Principali dispositivi di output:

monitor  
stampante  
casse acustiche

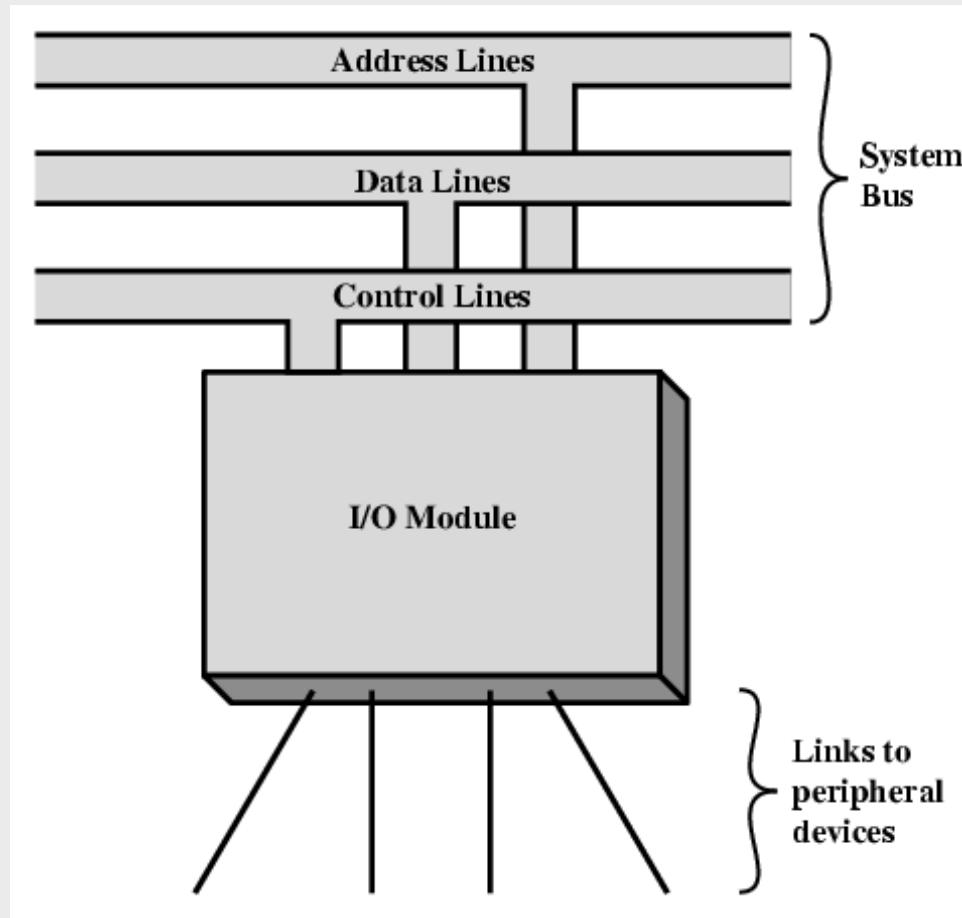
Collegamenti tramite:

1. porte di comunicazione (tastiera e mouse),
2. schede montate sulla motherboard.

Monitor – scheda video, Casse –  
scheda audio.

NB: queste due schede possono anche essere integrate nella scheda madre.

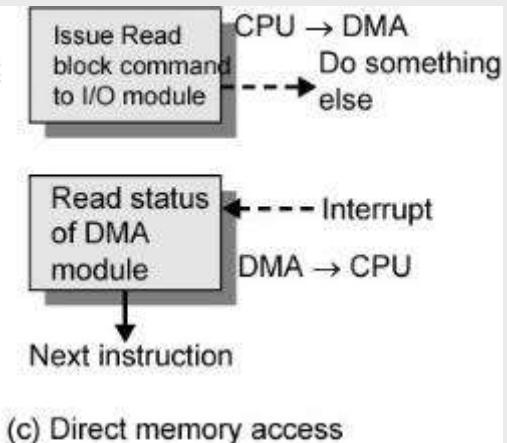
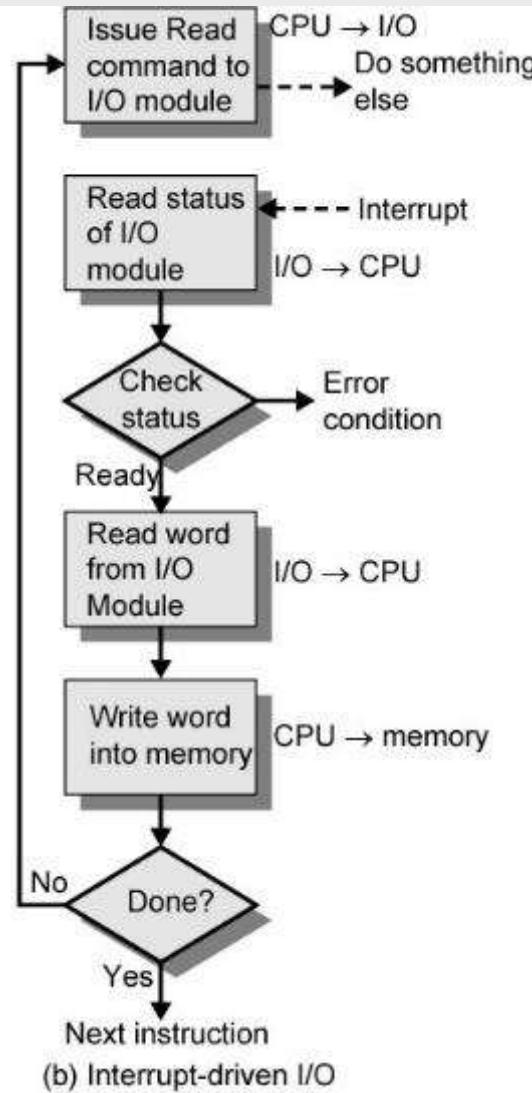
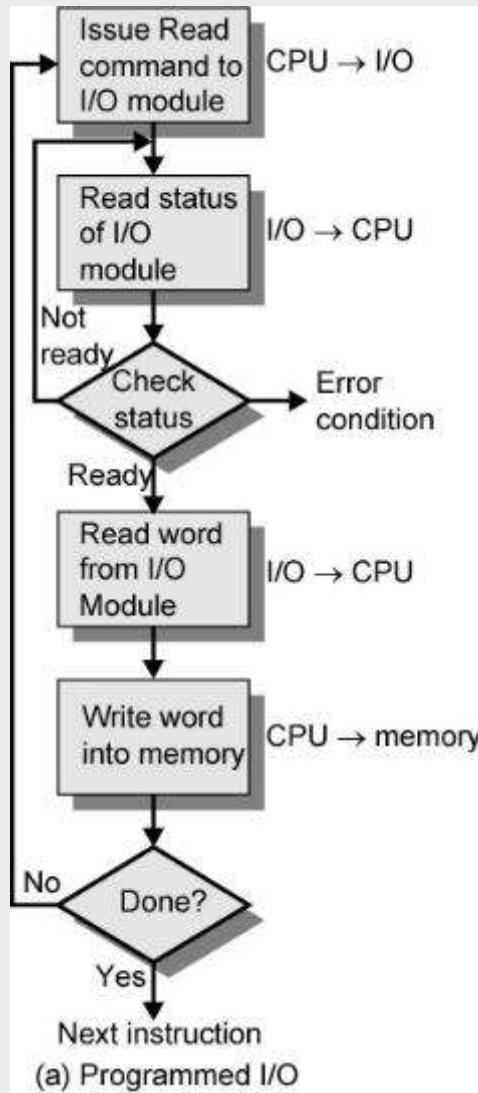
# Modello di un generico modulo di I/O



# Tecniche di I/O

- Programmato
- Interrupt driven
- Direct Memory Access (DMA)

# Tecniche di I/O



# I/O Programmato

- CPU ha controllo diretto sul dispositivo di I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU attende che il modulo di I/O completi l'operazione
- Wastes CPU time

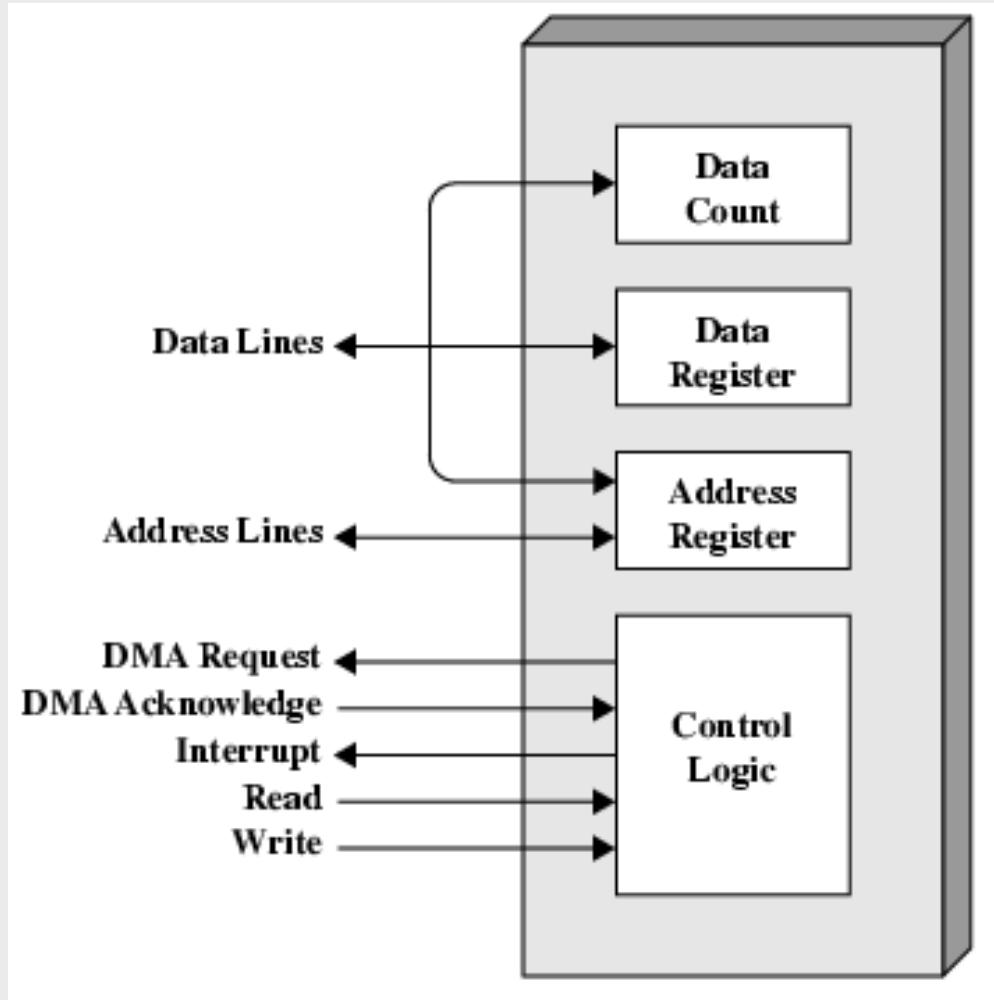
# I/O Interrupt driven

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

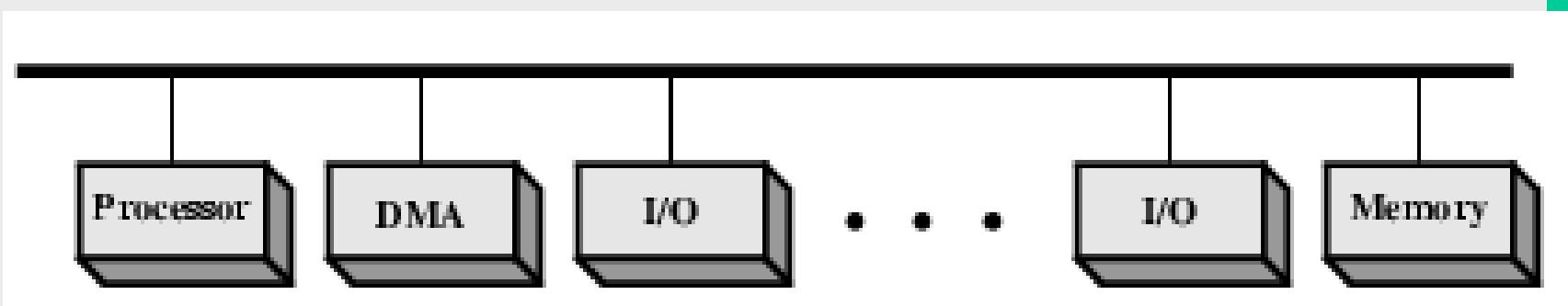
# DMA

- I/O Interrupt driven e programmato richiedono l'intervento attivo della CPU
- DMA is the answer
- Il DMA è un modulo aggiuntivo (hardware) montato sul bus
- OPERAZIONI SVOLTE DAL DMA
  - CPU comunica al controller di DMA l'operazione da svolgere
    - Read/Write
    - Device address
    - Starting address of memory block for data
    - Amount of data to be transferred
  - CPU svolge altri lavori
  - Il controller DMA si occupa del trasferimento
  - Il controller DMA invia un interrupt quando ha concluso il trasferimento

# DMA Module Diagram

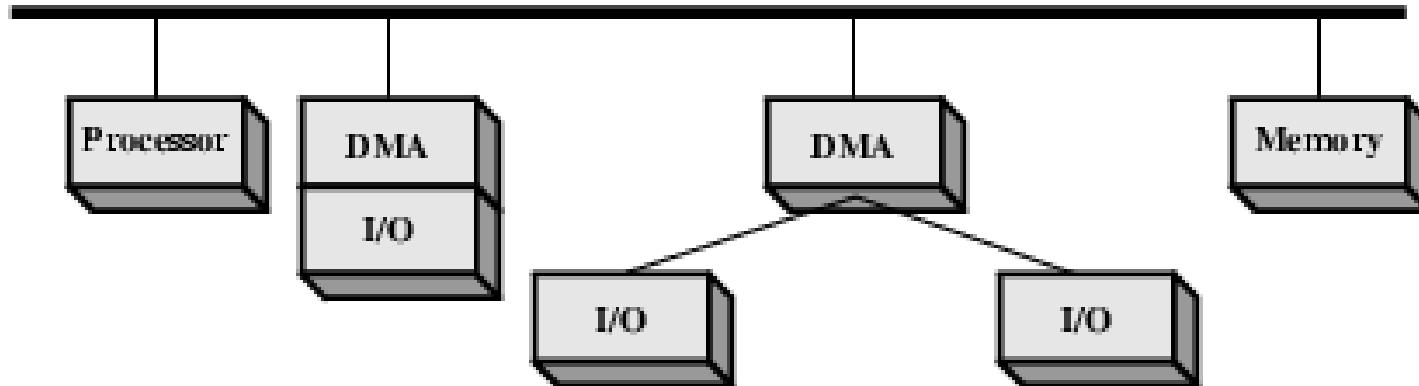


# Configurazioni DMA



- Single Bus,
- Ogni trasferimento utilizza il bus due volte:
  - I/O to DMA then DMA to memory

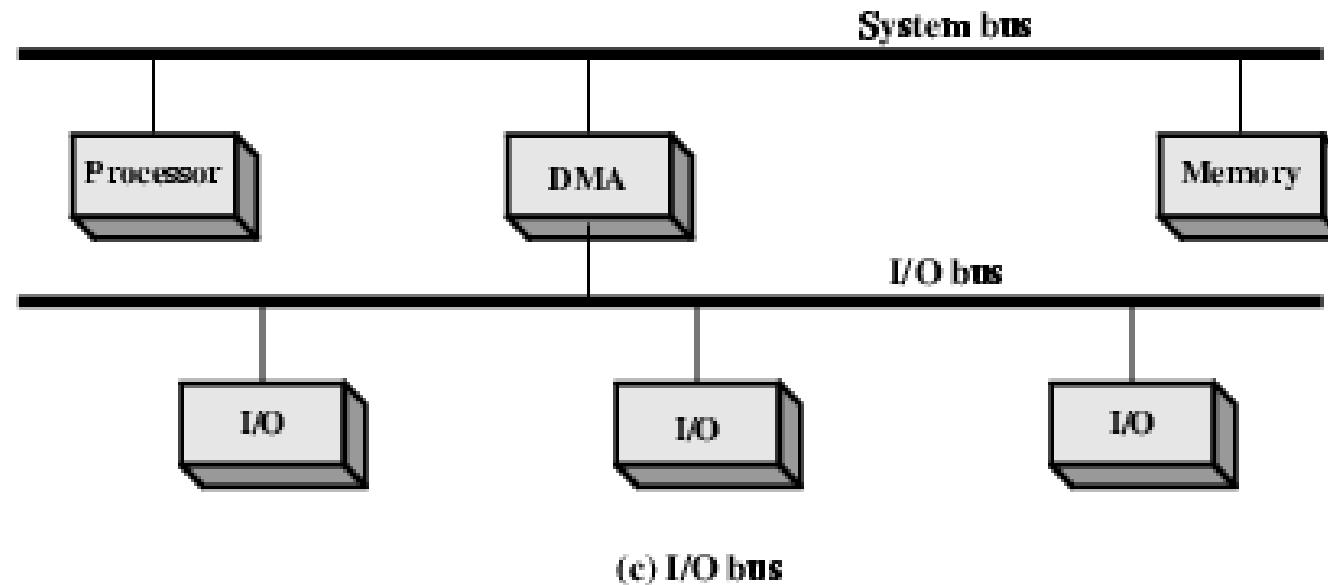
# Configurazioni DMA



(b) Single-bus, Integrated DMA-I/O

- Single Bus,
- Controller may support >1 device
- Ogni trasferimento utilizza il bus una volta
  - DMA to memory

# Configurazioni DMA



- Bus dedicato all'I/O
- Ogni trasferimento impegna il bus una sola volta
  - DMA to memory

# HARD DISK

Memoria permanente.

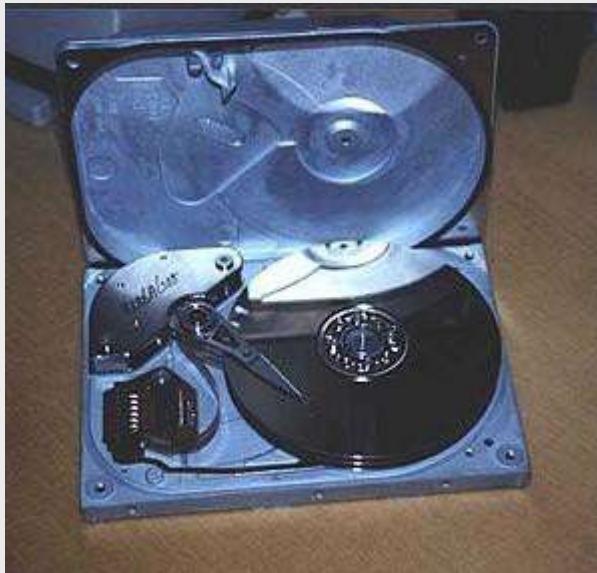
Costituito da uno o più dischi in rapida rotazione, realizzati in alluminio o vetro, rivestiti di materiale ferromagnetico e da due testine per ogni disco (una per lato).

Dimensioni: 2.5 (Laptop), 3.5 Pollici (Desktop)

Prestazioni:

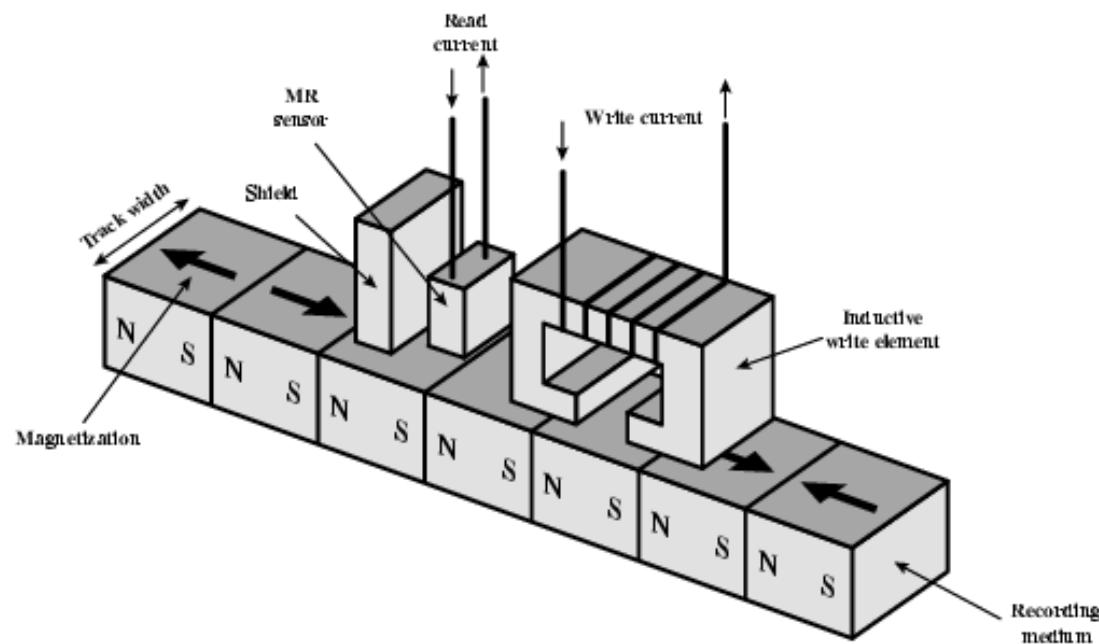
Capacità (GB)

Velocità di rotazione (rpm): 5.200, 5.400, 7.200, 10.000 e 15.000



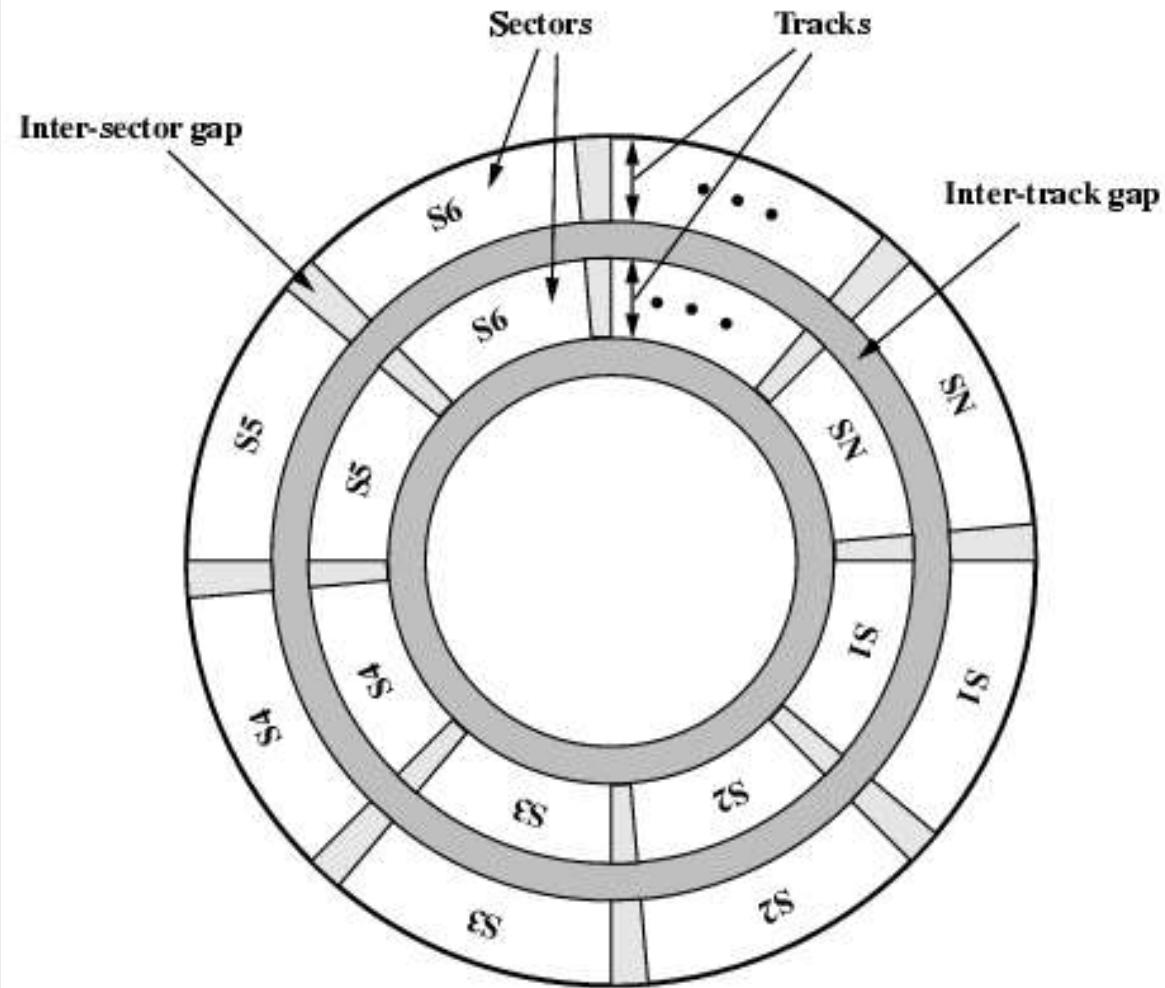
# Meccanismi di scrittura e lettura

- Durante le operazioni di lettura e scrittura la testina è fissa e i piatti ruotano
- Scrittura
  - La corrente attraverso i fili produce un campo magnetico
  - L'informazione magnetica viene memorizzata sulla superficie sottostante
- Lettura
  - Rilevazione del campo magnetico: il campo magnetico in movimento produce nel filo una corrente elettrica



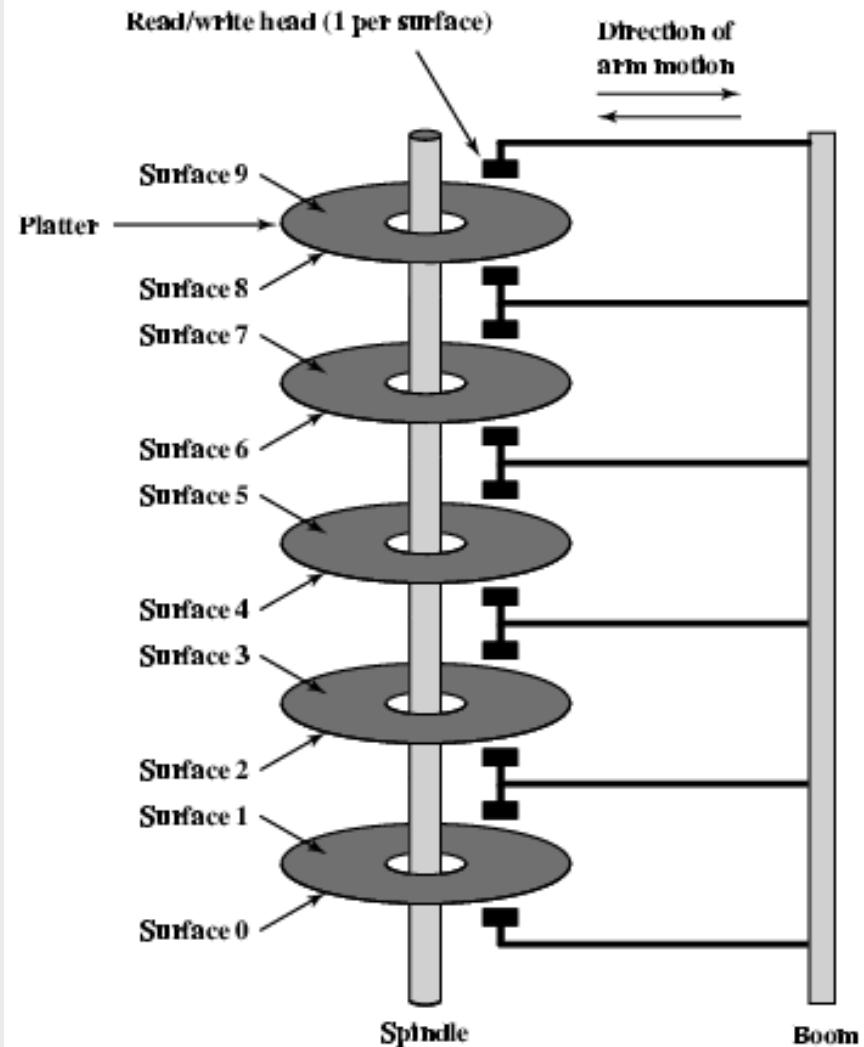
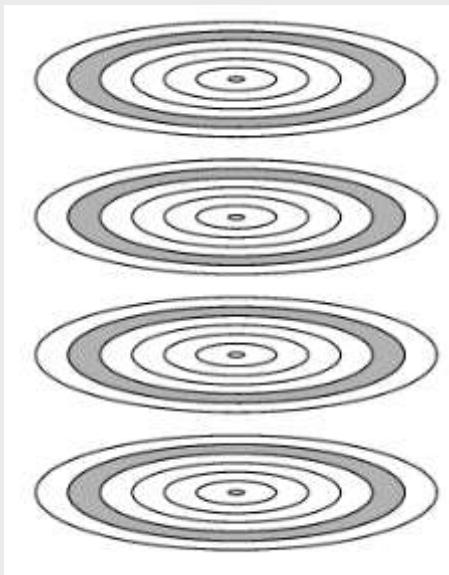
# Organizzazione dei dati

- Anelli concentrici o tracce
  - Gaps tra le tracce
- Le tracce sono divise in settori
- L'unità minima scrivibile/leggibile è il settore



# HARD DISK – piatti multipli

- Una testina per lato (=due testine per piatto)
- Le testine sono tutte allineate
- Le tracce allineate su più piatti individuano un cilindro
- I dati sono distribuiti tra i cilindri:
  - Riduzione del movimento delle testine
  - Aumento della velocità (transfer rate)



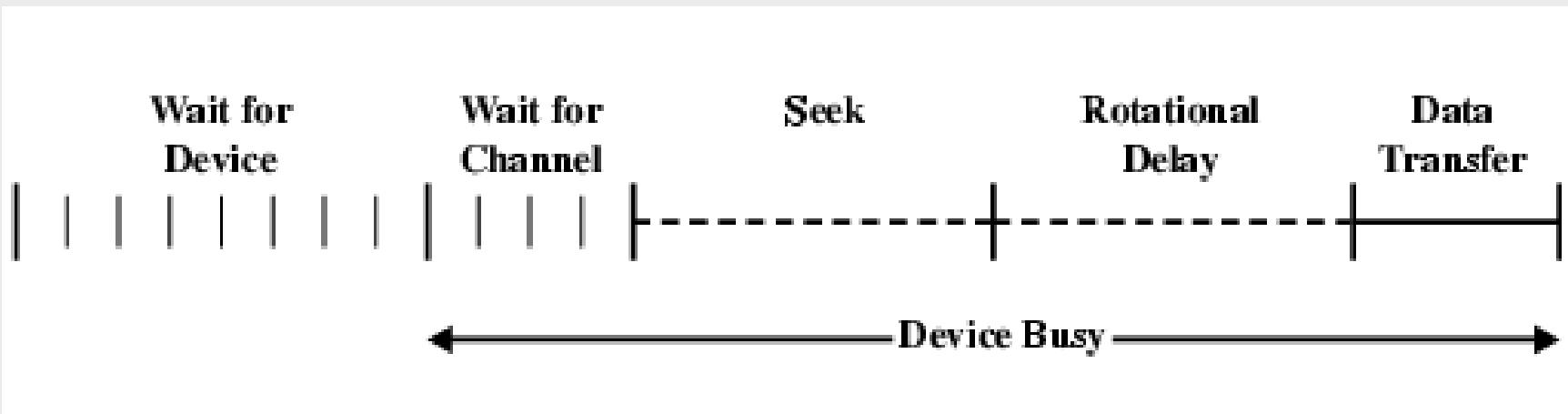
# Velocità di accesso all' HARD DISK

Seek time: tempo necessario alla testina per posizionarsi sulla traccia di interesse.

Rotational delay: tempo necessario affinchè la testina si posizioni sul settore di interesse.

Transfer time: tempo necessario al trasferimento dei dati

Access time: tempo che intercorre tra la richiesta di accesso ai dati e l'istante in cui i dati sono disponibili.



Domande:

1. Su quale parametro influisce la disposizione dei dati secondo cilindri?
2. Su quale parametro influisce la velocità di rotazione del disco?

# HARD DISK

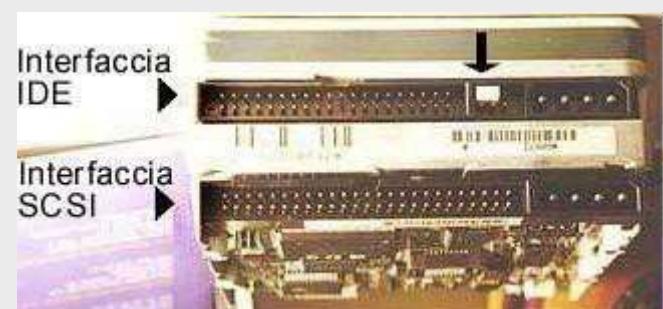
Si differenziano sulla base dell'interfaccia di trasferimento dei dati (vale anche per tutti i dispositivi periferici quali CD, DVD, etc.):

└ EIDE (Enhanced Integrated Drive Electronics) o PATA (Parallel Advanced Technology Attachment):

- └ drive (apparato hardware) per la connessione di dispositivi integrati nella scheda madre
- └ generalmente collocati sui personal computer di medie/basse prestazioni
- └ Velocità di trasferimento: 16.6 MB/s – 33MB/s (ATA-3 o ULTRA ATA)
- └ le operazioni di trasferimento dati impegnando la CPU
- └ non può operare sui due canali contemporaneamente, esegue una operazione di I/O alla volta.
- └ Sostituito da SATA (Serial ATA) velocità fino a 250MB/s (potenzialità: 500MB/s)

└ SCSI (Small Computer Systems Interface):

- └ Sistema definito da interfaccia e controller
- └ Velocità di trasferimento elevata (fino a 640 MB/s)
- └ Il trasferimento dei dati viene gestito dal controller (integrato nel dispositivo)



# RAID – Redundant Array of Independent Disks

Scopo: migliorare prestazioni ed affidabilità di un sistema di memorizzazione di massa utilizzando una batteria di dischi piuttosto che un unico disco (Petterson et. Al. 1988).

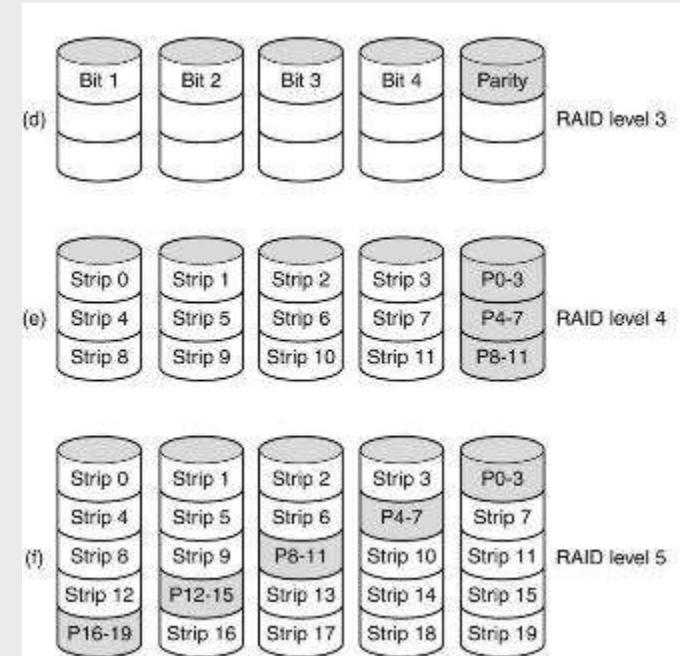
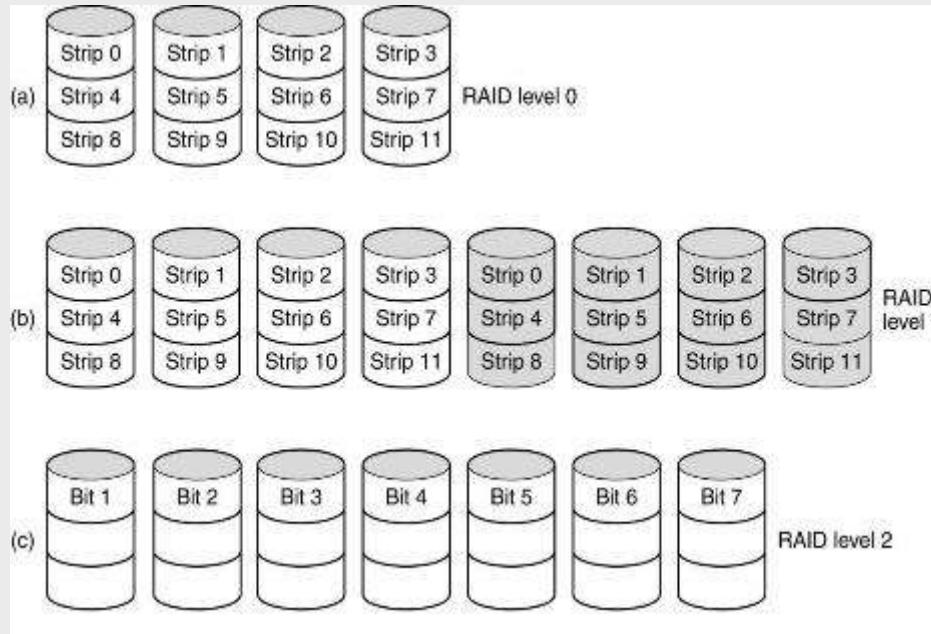
Al sistema operativo un disco RAID appare come un unico disco, potenzialmente con prestazioni e affidabilità maggiori rispetto ai dischi precedenti.

Implementazione:

- tecnologia SCSI che consente l'utilizzo simultaneo di più dischi con buone prestazioni
- un disco RAID consiste in un controllore RAID SCSI più un insieme di dischi SCSI.

I RAID hanno la proprietà di distribuire i dati sulle diverse unità consentendo elaborazioni parallele.

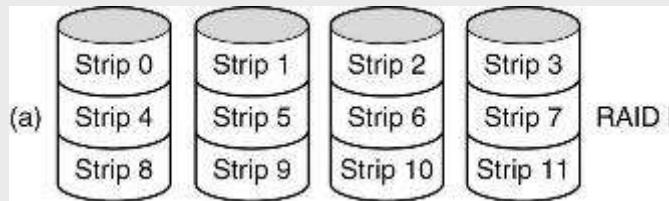
# RAID – Redundant Array of Independent Disks



RAID levels 0 through 5.

Backup and parity disks are shown shaded.

# RAID 0



In questo caso si usa una tecnica di striping : il disco simulato RAID è visto come se ognuno dei suoi  $k$  settori fosse diviso in strip (“strisce”), con (se  $k$  sono i dischi del RAID) :

Strip 0: settori da 0 a  $k-1$  ;

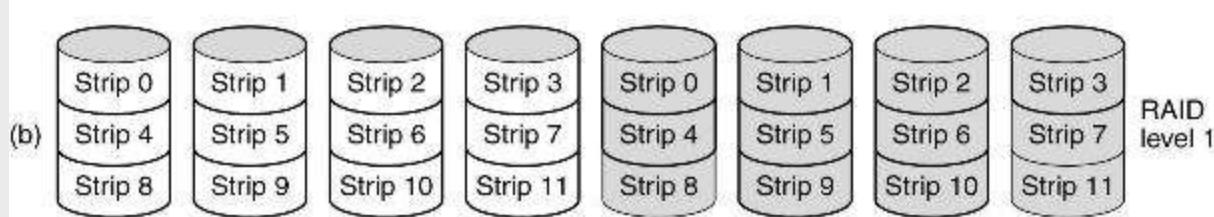
Strip 1: settori da  $k$  a  $2k-1$  ;

...

Per la lettura/scrittura dati di 4 settori successivi il controllore RAID spezzerà questo comando in 4 comandi separati (uno per ciascun disco) e li farà eseguire in parallelo).

RAID 0 lavora meglio quando le richieste sono di grandi dimensioni. RAID 0 non è ridondante (per questo non è considerato un vero e proprio “RAID”).

# RAID 1



In questo caso si usa una tecnica di striping+mirroring. Come nel RAID 0 ogni strip viene mappata su due diversi dischi.

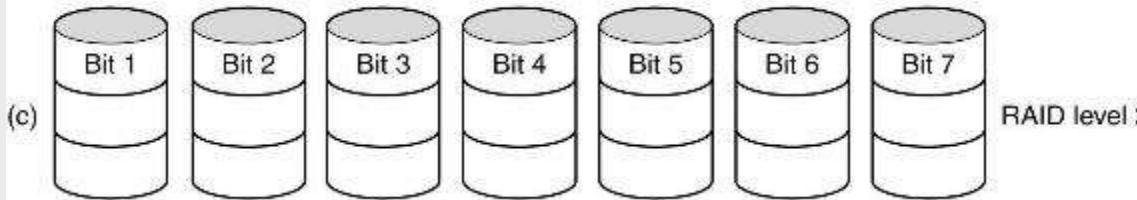
## Vantaggi:

- *Una richiesta di lettura può essere eseguita dal disco più "scarico"*
- *Una richiesta di scrittura deve essere eseguita su due dischi (ma potenzialmente in parallelo)*
- *Il malfunzionamento di un disco è facilmente recuperabile.*

## Svantaggi:

- *Costi elevati*

# RAID 2



Usa una tecnica di accesso parallelo. Tutti i dischi del RAID partecipano all'esecuzione delle richieste di I/O in maniera sincrona (tutte le testine di tutti i dischi assumono posizioni analoghe in ogni momento).

RAID 2 adotta lo striping dei dati, ma questa volta le strisce sono di dimensione di mezzo byte, un byte o una parola.

Lavorando con mezzo byte (+ tre bit di controllo dati) si ottengono i sette bit memorizzati tutti su dischi diversi (come mostrato in figura).

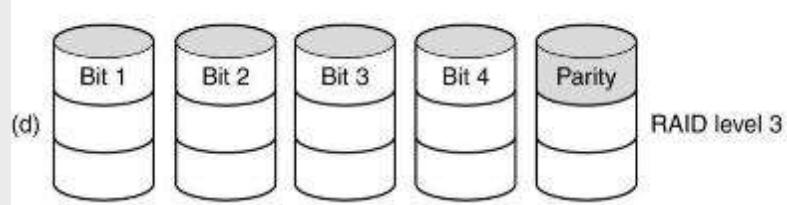
## Vantaggi:

- *Lettura/scrittura in parallelo (di ciascun mezzo byte + tre bit di controllo dati )*

## Svantaggi:

- *Dispendioso (meno che RAID 1)*
- *Rotazione sincronizzata dei dischi*
- *Molto lavoro al controllore che deve controllare rapidamente i bit di controllo*

# RAID 3



Usa una tecnica di accesso parallelo. E' una versione semplificata del RAID 2. Il bit di parità viene calcolato per ogni parola di dati e poi scritto su un apposito disco.

Dato che le parole di dati sono distribuite su più unità, anche in questo caso i dischi devono essere sincronizzati.

Nel caso di una rottura di un disco, attraverso il bit di parità si riesce a recuperare i dati.

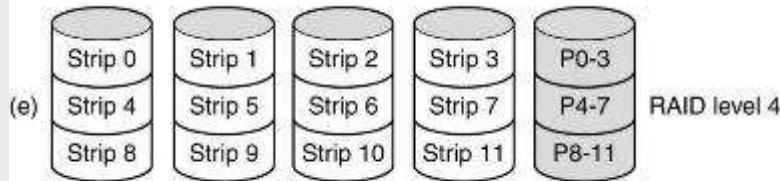
## ***Vantaggi:***

- *Lettura/scrittura in parallelo (di ciascun mezzo byte + 1 bit di parità)*

## ***Svantaggi:***

- *Rotazione sincronizzata dei dischi*

# RAID 4



Usa una tecnica basata sullo striping.

Il RAID 4 è come il RAID 0 con una parità calcolata strip-per-strip, che viene scritta su un disco aggiuntivo.

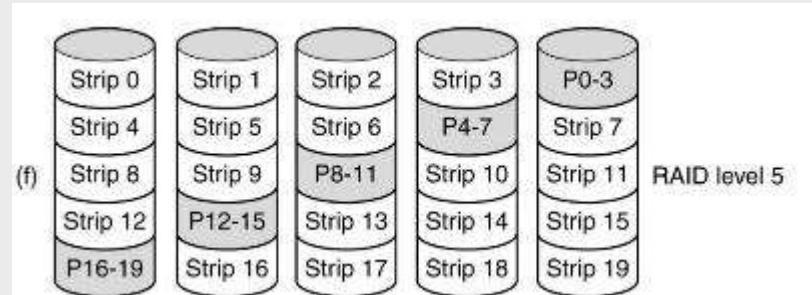
## ***Vantaggi:***

- *Protegge dalla rottura di un disco (l'informazione è recuperabile)*

## ***Svantaggi:***

- *Ha prestazioni scarse se si cambiano piccole moli di dati. Infatti per ogni, pur piccolo, cambiamento è necessario ricalcolare tutto lo strip di parità.*
- *Il disco per la parità diventa un "collo di bottiglia" per il sistema*

# RAID 5



Usa una tecnica basata sullo striping.

Il RAID 5 è come il RAID 4 con gli strip di parità distribuiti su tutti i dischi, in modalità "round-robin" una parità calcolata strip-per-strip, che viene scritta su un disco aggiuntivo.

## ***Vantaggi:***

- *Protegge dalla rottura di un disco (l'informazione è recuperabile)*

## ***Svantaggi:***

- *Ha prestazioni scarse se si cambiano piccole moli di dati. Infatti per ogni, pur piccolo, cambiamento è necessario ricalcolare tutto lo strip di parità.*

# Gestione dell' HARD DISK

**Deframmentazione** (XP: pannello di controllo ->strumenti di amministrazione ->gestione computer  
7: start ->tutti i programmi ->accessori ->utilità sistema)

**Formattazione:** dividere la capacità del disco in una serie di blocchi di uguali dimensioni e fornire una struttura in cui verranno scritte le informazioni.

**FISICA** o di **BASSO LIVELLO**: il disco inizialmente non è suddiviso in tracce e settori. L'operazione che tramite induzione magnetica suddivide il disco in tracce e settori prende il nome di formattazione fisica

**LOGICA** o di **ALTO LIVELLO**: Lo spazio di memoria dell'HD viene organizzato logicamente per poter essere accessibile da uno specifico sistema operativo (FAT32, iNODE..)

**Partizionamento:** suddivisione di un'unità fisica in più unità logiche. Le singole unità logiche vengono viste dal sistema operativo come unità separate e possono essere formattate e gestite in modo indipendente.

Cosa significa cancellare un file??? :D (drive rescue e eraser)

# SSD – Memorie a stato solido

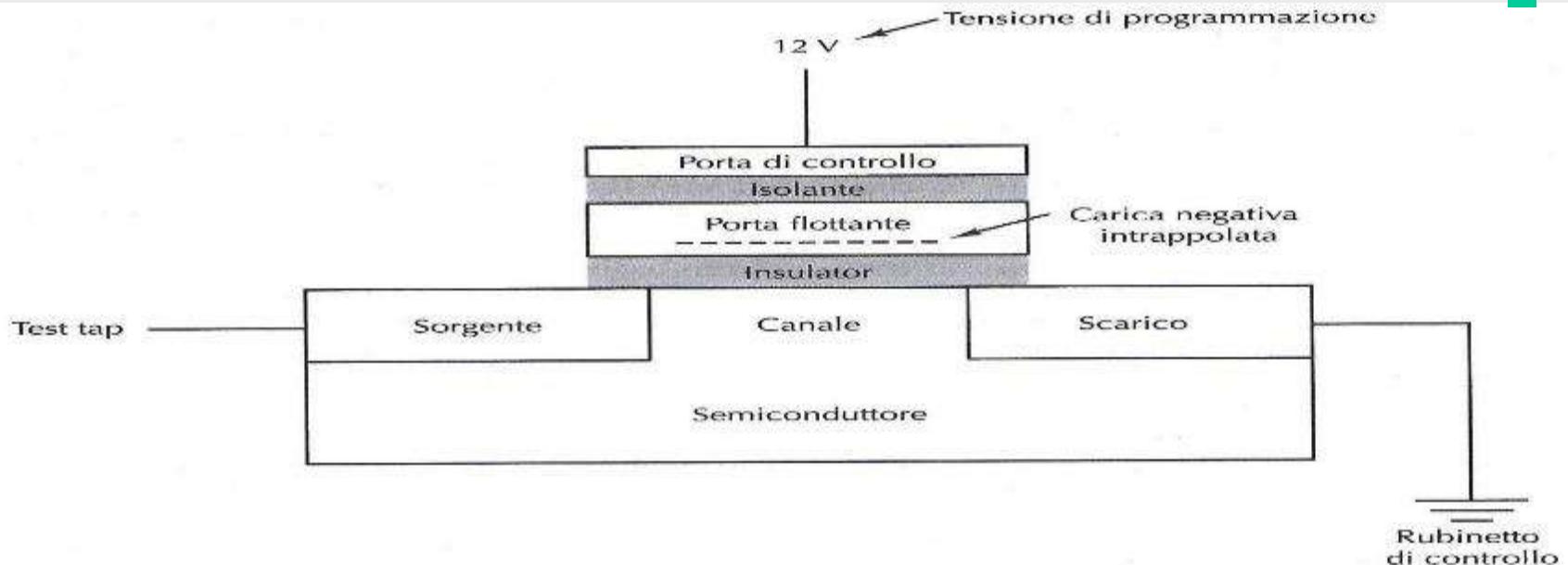


Figura 2.24 Una cella di memoria flash.

Basate su memorie flash.

PRO: Non hanno bisogno di rotazione: lettura/scrittura immediata.

CONTRO: Costo, possono essere scritte soltanto 100.000 volte

Osservati problemi di affidabilità.

# Scheda Video & Scheda Audio

Scheda video e scheda audio sono componenti essenziali di un PC multimediale.

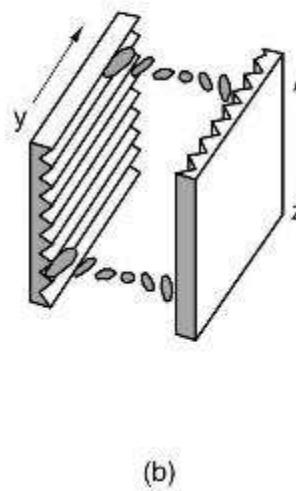
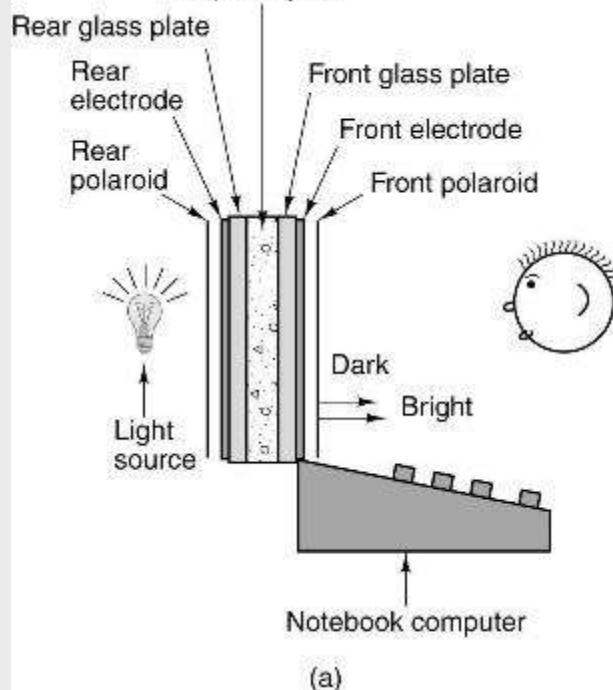
- **Scheda video (Video Graphic Adapter o VGA):** consente di visualizzare sullo schermo del monitor le informazioni elaborate dalla CPU.
  - Dispongono di un processore e di una memoria RAM (dimensioni) perché, oltre a raccogliere le informazioni ricevute dalla CPU, le elaborano prima di inviarle al monitor.
  - Utilizzano la CPU e condividono la RAM del Computer.
- **Scheda audio:** ha il compito di sintetizzare i suoni da inviare alle casse acustiche (riproduzione) o di registrare i suoni (campionamento e quantizzazione) acquisiti da una fonte esterna (microfono, lettore CD, ecc...).
  - Dispone di memoria dedicata e di un proprio processore
  - Integrata nella scheda madre.

# Monitor LCD a matrice attiva (TFT)

## Principali Parametri:

- **Contrasto:** rapporto fra la luminosità del bianco e la luminosità del nero  
Monitor a LED, miglioramento del contrasto sulla singola immagine, agendo dinamicamente sulle varie porzioni di retroilluminazione.  
Forti contrasti sono tuttavia necessari solo per l'uso in piena luce dello schermo.
- **Luminosità**
- **Linearità dei grigi:** imperfezione delle tonalità
- **Angolo di visuale:** angolo massimo sotto cui si può guardare lo schermo mantenendo una luminosità ed un contrasto "accettabili": il grado di "accettabilità" può essere liberamente stabilito dai produttori..diversi produttori, diversi angoli
- **Tempo di risposta:** tempo necessario ai "cristalli liquidi" per passare da uno stato "tutto chiuso" (nero) ad uno "tutto aperto" (bianco), per poi tornare al "tutto chiuso". Da 60 fino a 120 Hz un monitor per PC.

# Monitor LCD a matrice attiva (TFT)



*La tecnologia più comune alla base degli schermi piatti è quella dello schermo a cristalli liquidi, LCD (Liquid Crystal Display).*

*Uno schermo LCD consiste in due lastre di vetro tra le quali è posto un cristallo liquido.*

*Alle lastre sono attaccati degli elettrodi trasparenti utilizzati per creare campi elettrici all'interno del cristallo liquido.*

*Inoltre sono presenti sulle due lastre dei filtri polarizzati.*

*I cristalli liquidi sono molecole organiche viscose che si muovono come un liquido, ma che hanno una struttura spaziale simile a quella di un cristallo.*

*Quando tutte le molecole sono orientate nella stessa direzione, le proprietà ottiche del cristallo dipendono dalla polarizzazione della luce incidente.*

*E' possibile modificare l'allineamento dei cristalli e quindi le proprietà ottiche. In particolare si può controllare l'intensità della luce uscente. Questo principio è utilizzato per la costruzione degli schermi.*

# Categorie di Calcolatori con Tassonomia di Flynn

Singolo Processore

- **Single Instruction Single Data (SISD)**
  - Nessun parallelismo (Von Neumann)

Processori Paralleli

- **Single Instruction Multiple Data (SIMD)**
  - ALU vettoriali
- **Multiple Instruction Single Data (MISD)**
  - Una stessa sequenza di dati è trasmessa a un set di processori
  - Ogni processore esegue una differente sequenza di istruzioni sugli stessi dati
  - Mai implementata
- **Multiple Instruction Multiple Data (MIMD)**
  - Un set di processori esegue simultaneamente diverse sequenze di istruzioni su set diversi di dati

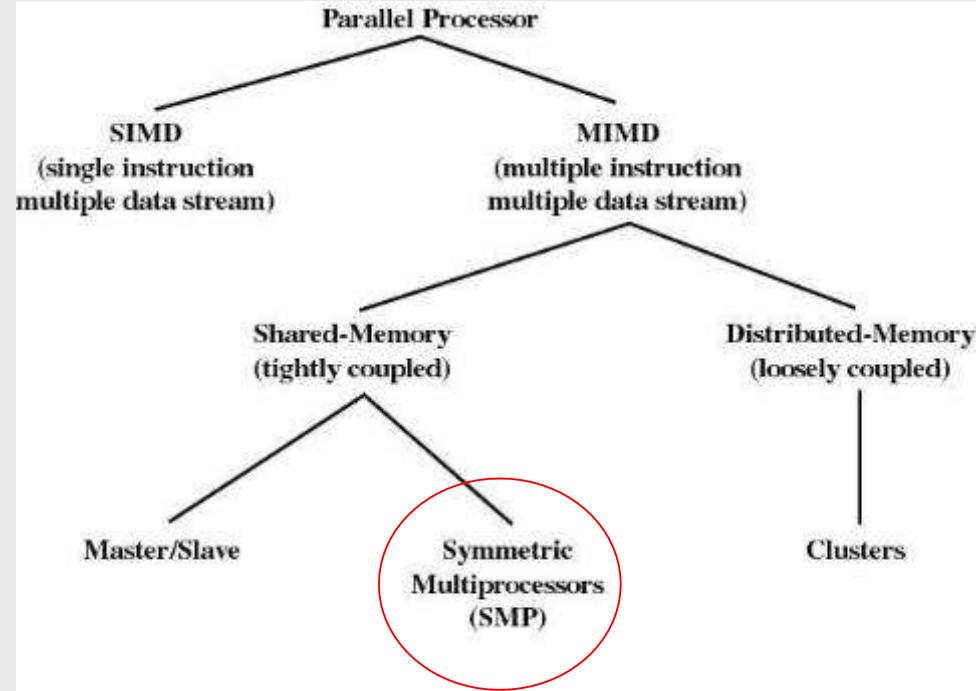


Figure 4.7 Parallel Processor Architectures

# Parallelismo

Parallelismo: capacità di eseguire più azioni nello stesso istante.

Esistono due tipologie di parallelismo:

- a livello di processore: più processori lavorano congiuntamente sullo stesso problema
- a livello di istruzione: il parallelismo viene sfruttato all'interno delle singole istruzioni per fare in modo che l'elaboratore esegue più istruzioni contemporaneamente.

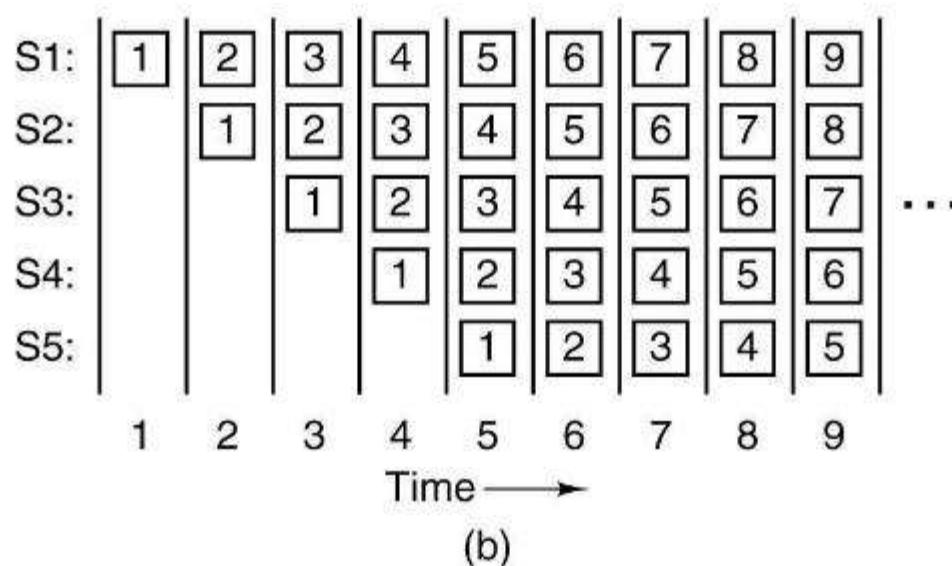
*Uno dei concetti applicati è quello di **pipeline**.*

*Il meccanismo di pipeline divide il ciclo di esecuzione in più parti e ciascuna parte è affidata ad una componente hardware.*

# Parallelismo - Pipeline



(a)



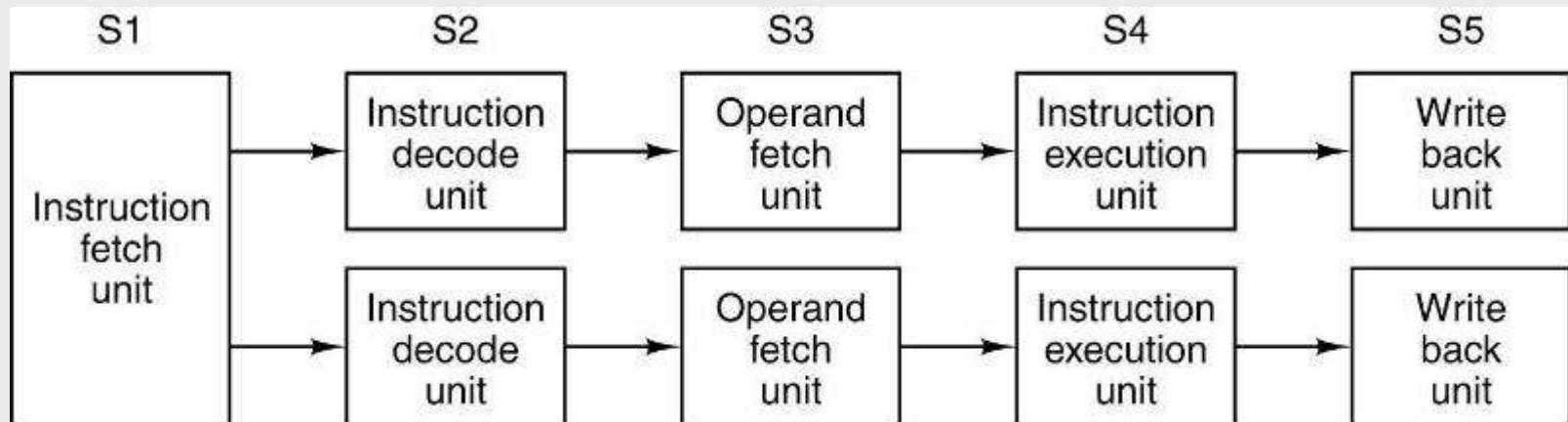
(b)

- Pipeline a cinque fasi
- Avanzamento degli stadi in funzione del tempo. Sono rappresentanti nove cicli

# Parallelismo - Pipeline

L'Intel ha dotato i propri processori, a partire dal 486, di una pipeline, e a partire dal Pentium, di due pipeline a cinque stadi.

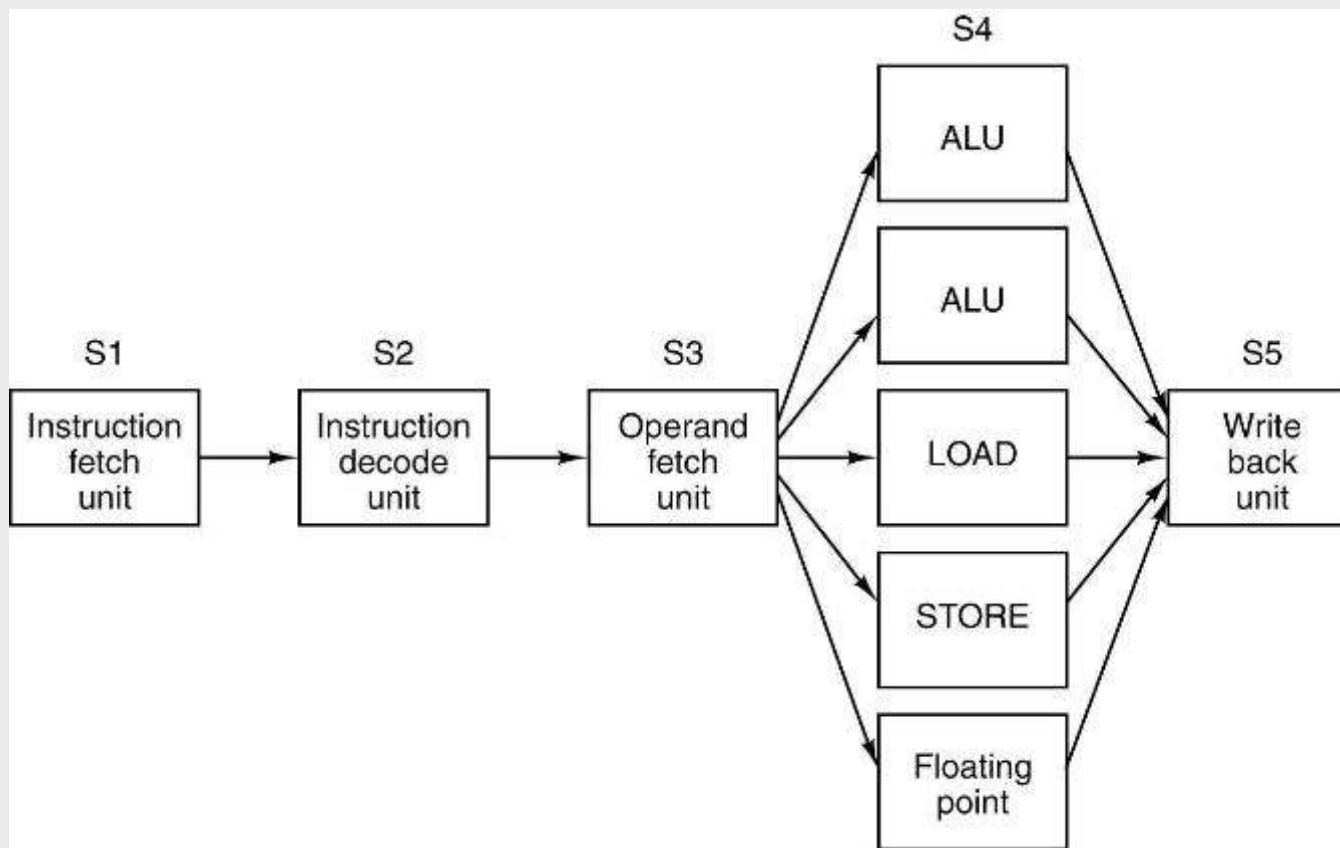
- La pipeline principale detta pipeline u esegue una qualsiasi istruzione Pentium.
- La seconda pipeline, detta pipeline v, esegue solo semplici istruzioni su interi e una su numeri a virgola mobile.



Doppia pipeline a cinque stadi con unità di fetch dell'istruzione in comune.

# Parallelismo - Pipeline

Processore superscalare con 5 unità funzionali: architettura con una sola pipeline alla quale vengono associate più unità funzionali.



# Parallelismo - Pipeline

Sia

- k -numero di stadi della pipeline
- n -numero di istruzioni da eseguire
- t -tempo di uno stadio (tempo massimo).

(si considerino inoltre trascurabili i tempi di commutazione da uno stadio al successivo)

In una macchina sequenziale senza parallelismo il tempo per eseguire le istruzioni è

$$T_{\text{convenzionale}} = nk t$$

In una macchina dotata di pipe (senza salti):

$$T_{\text{pipe}} = [k + (n-1)] \cdot t$$

Il fattore di velocizzazione (speed-up) della pipeline rispetto ad una architettura tradizionale è:

$$\frac{T_{\text{convenzionale}}}{T_{\text{pipe}}} = \frac{nkt}{[k + (n-1)]t} = \frac{nk}{[k + (n-1)]}$$

# Parallelismo – Pipeline

## Trattamento dei salti

Le istruzioni di salto (condizionato) sono uno dei principali problemi che limitano le prestazioni di una pipeline.

Le possibili soluzioni sono:

- Flussi multipli (multiple stream)
- Prelievo anticipato della destinazione (prefetch branch target)
- Buffer circolare (loop buffer)
- Predizione di salto (branch prediction)
- Salto ritardato (delayed branch)

# Parallelismo – Pipeline

## Trattamento dei salti

### Flussi multipli (multiple stream)

- Si replicano le parti iniziali della pipeline per consentirle di prelevare entrambe le istruzioni (coinvolte nel salto condizionato), facendo uso dei due flussi.
- Problemi:
  - Generazione di ritardi per la contesa tra i due flussi
  - In ciascuno dei due flussi si possono presentare altre istruzioni di salto condizionato...

### Prelievo anticipato della destinazione (prefetch branch target)

- Quando viene riconosciuto un salto condizionato viene anche prelevata anticipatamente (prefetching) la sua destinazione.
- L'indirizzo è mantenuto fino a quando si dovrà eseguire l'istruzione di salto di modo che, se occorre compiere il salto, l'indirizzo è già stato prelevato.

# Parallelismo – Pipeline

## Trattamento dei salti

### Buffer circolare (loop buffer)

Piccola e veloce memoria che contiene le ultime n istruzioni prelevate con il fetch. Se occorre effettuare un salto l'hardware prima controlla se la destinazione si trova nel buffer. In caso affermativo la successiva istruzione viene prelevata dal buffer.

#### Vantaggi:

- Anticipando il fetch è possibile avere già nella memoria circolare alcune istruzioni evitando di dover riferimento in memoria alcune volte;
- Nei cicli, se le istruzioni dell'intero ciclo riescono a stare nel buffer circolare, si ha un massimo vantaggio.

# Parallelismo – Pipeline

## Trattamento dei salti

### Predizione di salto (branch prediction)

- Previsione di saltare sempre (approccio statico)
- Previsione di non saltare mai (approccio statico)
- Previsione in base al codice operativo (approccio statico)
- Bit taken/not taken (approccio dinamico)
- Tabella della storia dei salti (approccio dinamico)

# Parallelismo – Pipeline

## Trattamento dei salti

Predizione di salto (branch prediction)  
Bit taken/not taken

Ad ogni istruzione di salto condizionato in cache viene associato un bit (taken/not taken) che memorizza il comportamento recente di quella istruzione.

Questa informazione viene utilizzata per anticipare il comportamento dell'istruzione di salto.

# Parallelismo – Pipeline

## Trattamento dei salti

Predizione di salto (branch prediction)

Tabella con la storia dei salti.

Viene usata una tabella con la storia dei salti.

Ciascuna riga contiene:

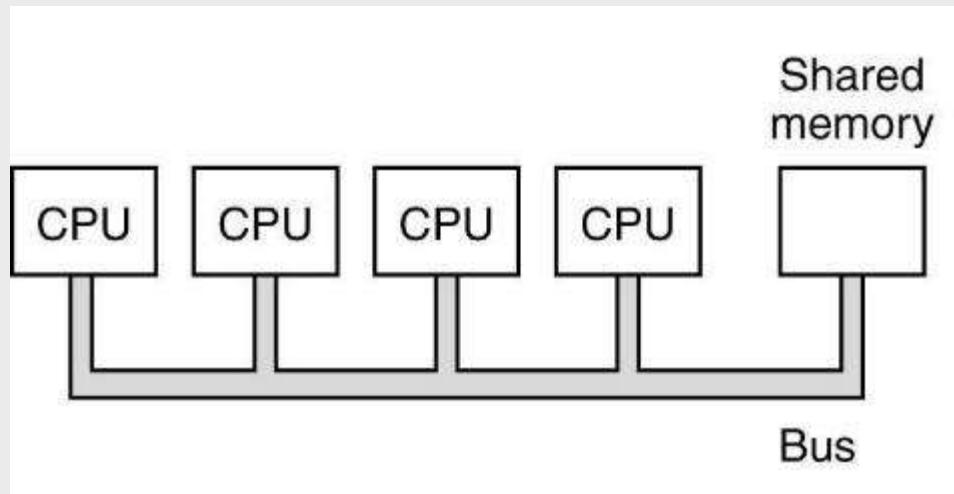
- 1) l'indirizzo dell'istruzione di salto
- 2) un certo numero di bit di storia
- 3) informazioni circa l'istruzione destinazione (di solito il suo indirizzo)

# Parallelismo – Pipeline Trattamento dei salti

Salto ritardato (delayed branch)

Ridisporre automaticamente le istruzioni all'interno di un programma in maniera tale che le istruzioni di salto si verifichino in ritardo rispetto al momento in cui effettivamente desiderate.

# Multiprocessori

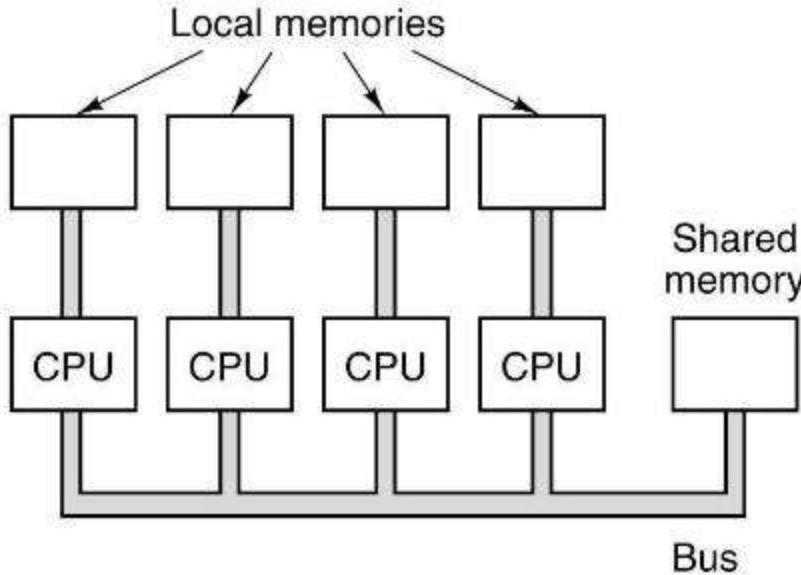


**Multiprocessore:** Più processori, una RAM  
E' necessario che le CPU siano sincronizzate per evitare conflitti nelle operazioni sulla memoria condivisa.  
**CPU fortemente accoppiate.**

*Ad esempio:*

*Se nella memoria è contenuta una immagine da elaborare è possibile affidare una sezione della immagine ad ogni CPU che eseguirebbe il medesimo programma delle altre sulla sezione di immagine ad essa affidata, riducendo il tempo di elaborazione dell'immagine*

# MultiComputer



Più CPU dotate di una memoria privata nel quale vengono contenuti dati che non sono condivisi ma utili all'elaborazione.

Il minor scambio sul bus rende più veloce l'esecuzione.

L'area condivisa è utilizzata per contenere, ad esempio, il codice del programma da eseguire.

Le **CPU sono debolmente connesse**, la comunicazione fra di loro avviene attraverso l'uso di messaggi che viaggiano sul bus che le collega.

# Symmetric Multi Processing (SMP)

- Un calcolatore con molti processori
- I processori condividono le stesse risorse
- Tutti i processori possono effettuare le stesse funzioni
- Ogni processore esegue una stessa copia del SO
- Vantaggi:
  - parallelismo, i thread possono essere schedulati su tutti i processori
  - disponibilità: simmetria dei processori, se uno fallisce, gli altri possono continuare a lavorare
  - crescita incrementale: aggiungere nuovi processori
  - Scalabilità
  - NB: vantaggi potenziali e non garantiti
- Presenza di più processori trasparente all'utente

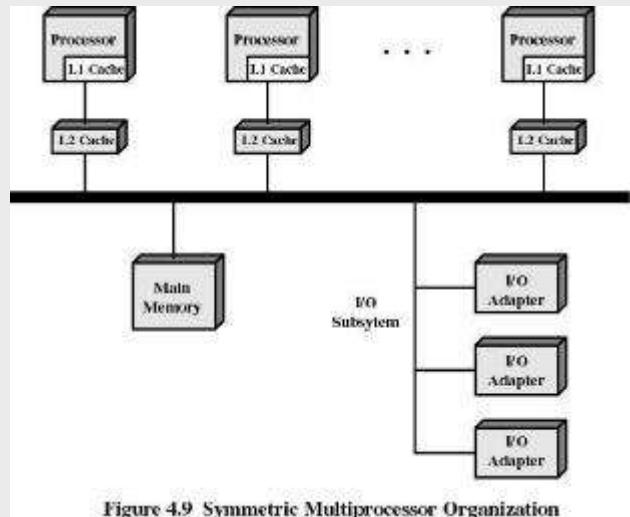
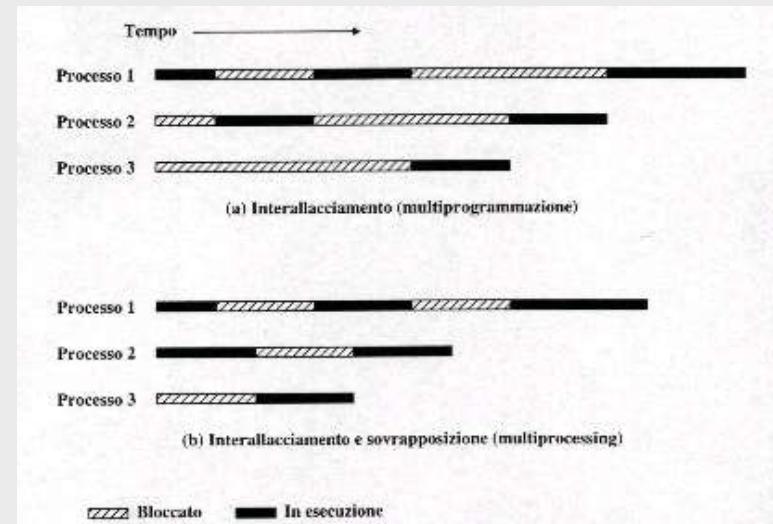


Figure 4.9 Symmetric Multiprocessor Organization



# Vantaggio numerico di un Multiprocessore

Sia  $P$  un processo e supponiamo che sia costituito da una frazione seriale  $f_s$  (realizzabile in un tempo  $T_s$ ) e da una frazione  $f_p$  realizzabile in parallelo (realizzabile in un tempo  $T_p$ ) dove ovviamente  $f_s + f_p = 1$ .

Sia  $T_1$  e  $T_n$  rispettivamente il tempo di esecuzione del processo su 1 ed  $n$  processori:

$$T_1 = T_s + T_p \quad (\text{può essere considerato normalizzato a 1})$$

$$T_n = T_s + T_p/n$$

Lo speed-up  $S(n)$  con  $n$  processori è pari a  $T_1/T_n$  (Legge di Andahl)

$$S(n) = \frac{T_1}{T_n} = \frac{1}{f_s + \frac{f_p}{n}} = \frac{1}{f_s + \frac{1-f_s}{n}} = \frac{n}{nf_s + (1-f_s)}$$

*NB: fatevi qualche esempio numerico per capire quando «serve» un multiprocessore!!!!*

# DRIVE e Dispositivi Mobili

L Lettori/Masterizzatori CD, DVD

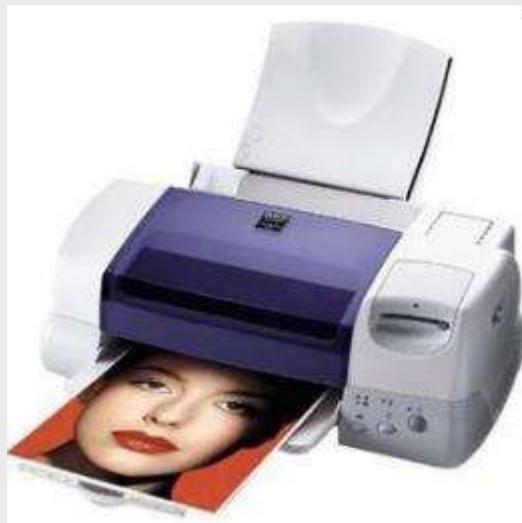
L Supporti in grado di immagazzinare dati fino a 700 MB (CD) e fino a 4.7 GB (DVD).  
Blu Ray (54 GB). DVD+R/+RW

	CD impresso (da matrice)	CD-R	CD-RW	DVD impresso (da matrice)	DVD-R	DVD+R	DVD-RW	DVD+RW	DVD+R DL	DVD-RAM	BD-R
Lettore CD audio	Lettura	Lettura <sup>[1]</sup>	Lettura <sup>[2]</sup>	No	No	No	No	No	No	No	No
Lettore CD-ROM	Lettura	Lettura <sup>[1]</sup>	Lettura <sup>[2]</sup>	No	No	No	No	No	No	No	No
Masterizzatore CD-R	Lettura	Scrittura	Lettura	No	No	No	No	No	No	No	No
Masterizzatore CD-RW	Lettura	Scrittura	Scrittura	No	No	No	No	No	No	No	No
Lettore DVD-ROM	Lettura	Lettura <sup>[3]</sup>	Lettura <sup>[3]</sup>	Lettura	Lettura <sup>[4]</sup>	Lettura <sup>[4]</sup>	Lettura <sup>[4]</sup>	Lettura <sup>[4]</sup>	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD-R	Lettura	Scrittura	Scrittura	Lettura	Scrittura	Lettura	Lettura <sup>[6]</sup>	Lettura	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD-RW	Lettura	Scrittura	Scrittura	Lettura	Scrittura	Lettura <sup>[6]</sup>	Scrittura <sup>[7]</sup>	Lettura	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD+R	Lettura	Scrittura	Scrittura	Lettura	Lettura	Scrittura	Lettura	Lettura	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD+RW	Lettura	Scrittura	Scrittura	Lettura	Lettura	Scrittura	Lettura	Scrittura	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD±RW	Lettura	Scrittura	Scrittura	Lettura	Scrittura	Scrittura	Scrittura	Scrittura	Lettura <sup>[5]</sup>	No	No
Masterizzatore DVD±RW/DVD+R DL	Lettura	Scrittura	Scrittura	Lettura	Scrittura <sup>[8]</sup>	Scrittura	Scrittura <sup>[8]</sup>	Scrittura	Scrittura	No	No
Masterizzatore DVD Super Multi	Lettura	Scrittura	Scrittura	Lettura	Scrittura	Scrittura	Scrittura	Scrittura	Scrittura	Scrittura	No
Masterizzatore Blu-Ray Super Multi	Lettura	Scrittura	Scrittura	Lettura	Scrittura	Scrittura	Scrittura	Scrittura	Scrittura	Scrittura	Scrittura

# Stampanti – Ink Jet

Carrello si muove per tutta la larghezza del foglio, su di esso sono fissate le testine di stampa, che proiettano sul foglio micro-gocce di colore:

- └ Meccanismo termico: resistore in corrispondenza di ogni ugello attraverso il quale vengono fatti passare impulsi di corrente; ad ogni impulso si ha l'espulsione della goccia di inchiostro; (Hewlett-Packard, Canon, Lexmark)
- └ Meccanismo piezoelettrico: sotto ogni ugello è posizionato un canalino circondato da un cristallo piezoelettrico; un impulso elettrico provoca la deformazione del cristallo e conseguentemente la repentina strozzatura del canalino e l'espulsione dell'inchiostro; (Epson)



# Stampanti – Ink Jet

Testine di stampa integrate nelle cartucce

esaurimento inchiostro ->sostituzione testina:

costo cartuccia vs efficienza testine

Essiccamento dell'inchiostro nelle testine

Risoluzione: dpi (dot per inches)

Velocità: 15 ppm b/n, 10 ppm colori

Quadricromia (ciano/magenta/giallo/nero)

Esacromia (ciano/magenta/giallo/nero/arancione/verde)

Costo medio stampa: 10 cent di Euro/pagina

Vita della stampante in pagine stampate

# Stampanti – Laser

Processo di stampa:

1. Il raggio laser infrarosso viene
  - a) modulato secondo la sequenza di pixel che deve essere impressa sul foglio,
  - b) deflesso da uno specchio rotante su un tamburo fotosensibile elettrizzato che si scarica dove colpito dalla luce.
2. L'elettricità statica attira una fine polvere (toner) di materiali sintetici e pigmenti che viene trasferito sulla carta.
3. Il foglio passa sotto un rullo riscaldato ad elevata temperatura, che fonde il toner facendolo aderire alla carta.

Stampa a colori: quattro toner (nero, ciano, magenta e giallo) trasferiti da un unico tamburo oppure da quattro distinti. (I toner possono essere integrati in uno solo)

Risoluzione: 4800x1200 dpi(dot per inches)

Velocità: 70 ppm b/n, 40 ppm colori

Costo medio stampa: 4 cent di Euro/pagina

# Sistemi Embedded

**Dispositivo encapsulato** all'interno del sistema da controllare **progettato per una determinata applicazione** supportato da una piattaforma hardware su misura.

E' essenzialmente un sistema a microprocessore come un PC ma:

- dedicato a svolgere un particolare compito
- ha risorse e dispositivi strettamente necessari
- hardware dedicato
- frequenze di CPU inferiori
- basso consumo
- poca memoria (anche di massa)

Caratteristiche:

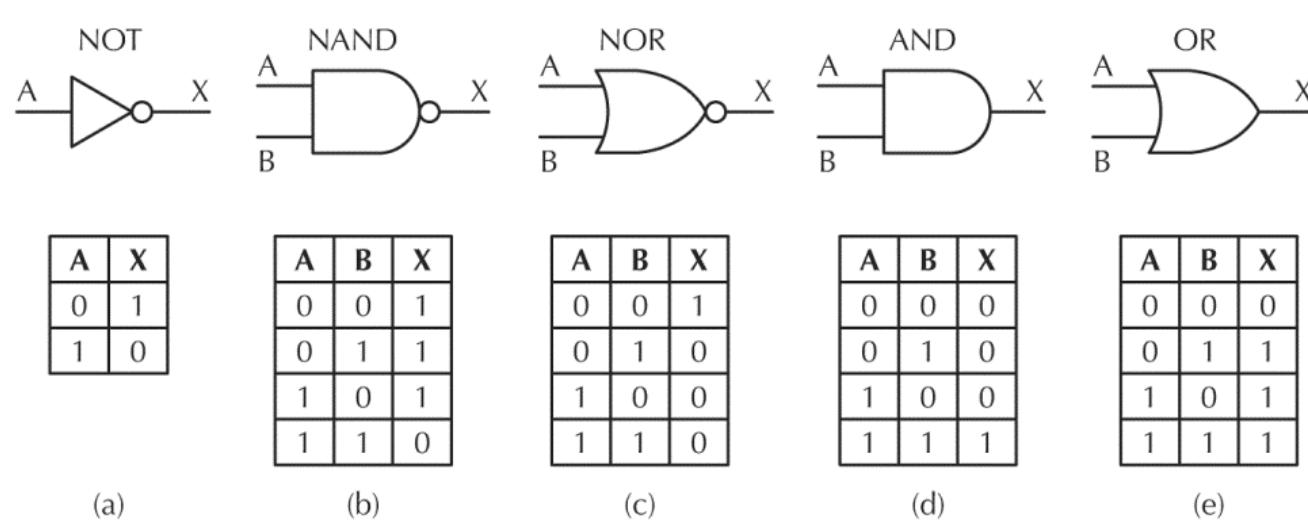
- sistemi privi di interazione umana
- capaci di resistere ad eventi dannosi, a vibrazioni e shock
- ripartire in modo autonomo
- possibilmente di dimensioni ridotte
- raggiungibile tramite un qualche tipo di connessione
- deve avere risorse necessarie per l'esecuzione dei processi indispensabili al suo funzionamento
- spesso con vincoli temporali (sistema real-time)

# Livello Logico-Digitale

*Prof. Ing. Donato Impedovo*

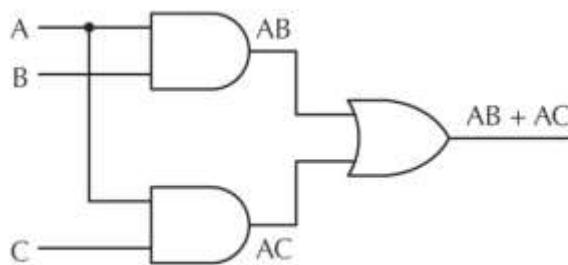
---

# Porte Logiche



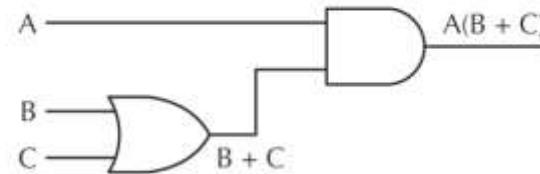
**Figura 3.2** Simboli e comportamenti funzionali di cinque porte logiche elementari.

# Funzioni



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)



A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

**Figura 3.5** Due funzioni equivalenti. (a)  $AB + AC$ . (b)  $A(B + C)$ .

# Proprietà

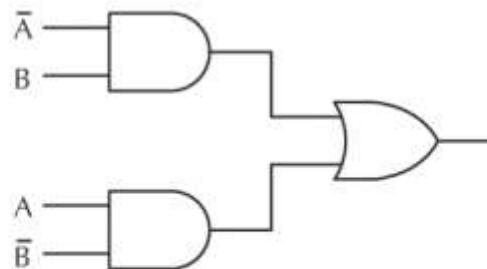
Nome	Forma AND	Forma OR
Elemento neutro	$1A = A$	$0 + A = A$
Assorbimento	$0A = 0$	$1 + A = 1$
Idempotenza	$AA = A$	$A + A = A$
Complementazione	$A\bar{A} = 0$	$A + \bar{A} = 1$
Proprietà commutativa	$AB = BA$	$A + B = B + A$
Proprietà associativa	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Proprietà distributiva	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Legge di assorbimento	$A(A + B) = A$	$A + AB = A$
Legge di De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$A + B = \overline{\overline{A}B}$

**Figura 3.6** Identità dell'algebra booleana.

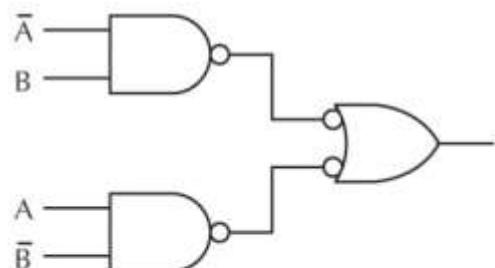
# XOR

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

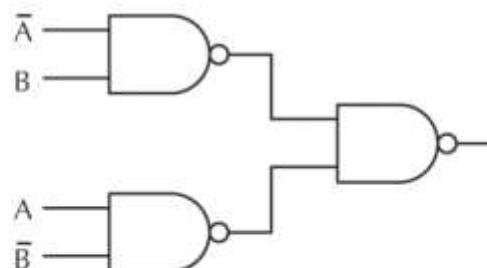
(a)



(b)



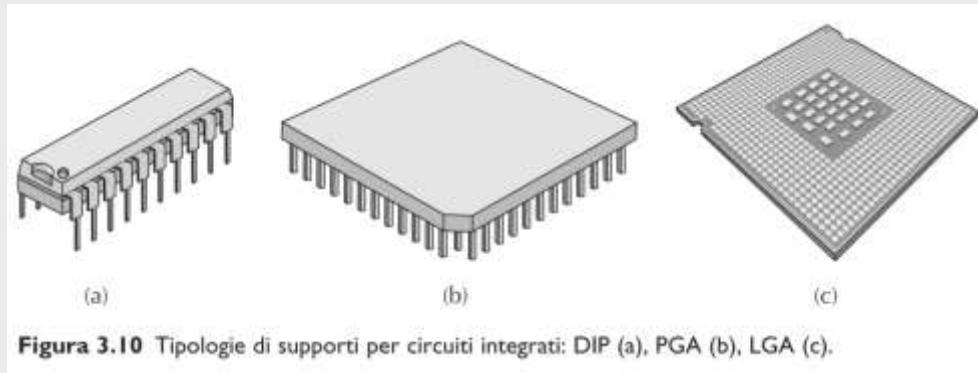
(c)



(d)

**Figura 3.8** (a) Tabella di verità della funzione XOR. (b)-(d) Tre circuiti per calcolarla.

# Usage



**Figura 3.10** Tipologie di supporti per circuiti integrati: DIP (a), PGA (b), LGA (c).

*DIP: Dual Inline Package*

*PGA: Pin Grid Arrays*

*LGA: Land Grid Arrays*

# Multiplexer

In funzione del valore sulle linee A,B e C solo una delle otto linee di entrata viene trasmessa in uscita.

$2^n$  valore input

1 valore output

n valore controllo

A	B	C	D <sub>i</sub>	F
0	0	0	D <sub>0</sub>	D <sub>0</sub>
0	0	1	D <sub>1</sub>	D <sub>1</sub>
0	1	0	D <sub>2</sub>	D <sub>2</sub>
0	1	1	D <sub>3</sub>	D <sub>3</sub>
1	0	0	D <sub>4</sub>	D <sub>4</sub>
1	0	1	D <sub>5</sub>	D <sub>5</sub>
1	1	0	D <sub>6</sub>	D <sub>6</sub>
1	1	1	D <sub>7</sub>	D <sub>7</sub>

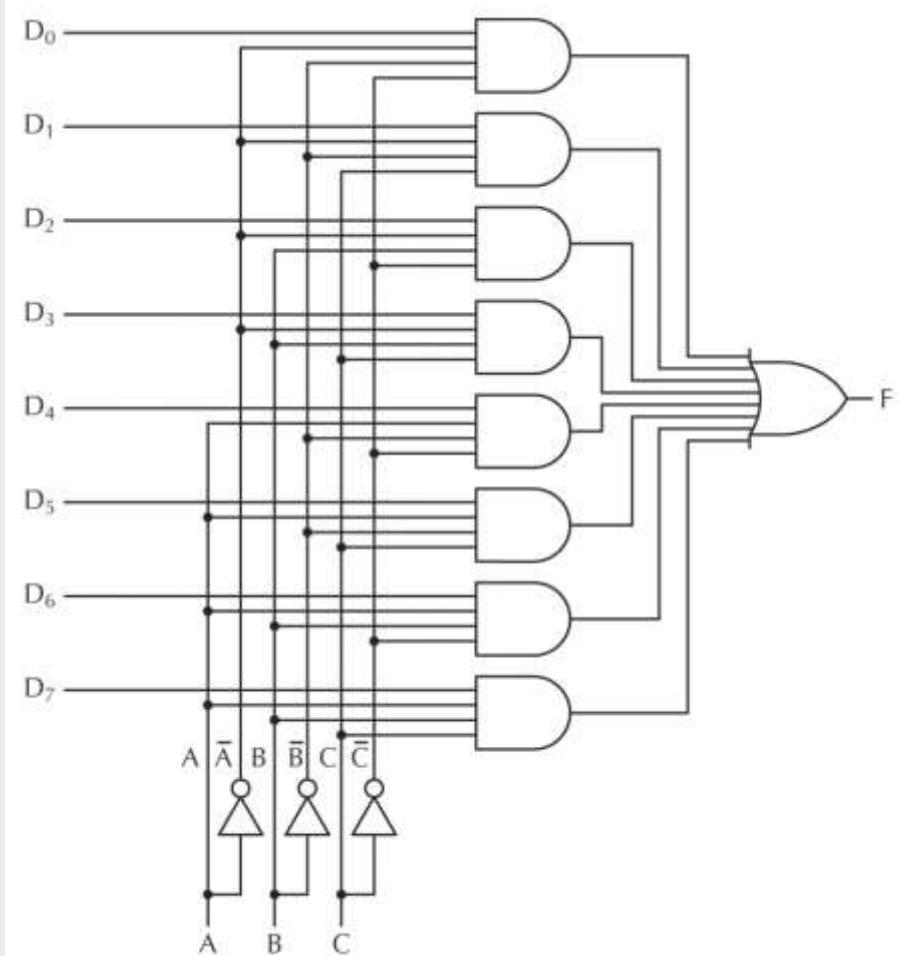


Figura 3.11 Multiplexer a otto vie.

# Multiplexer

Usando un contatore e abilitando in uscita le diverse entrate in sequenza, un byte da parallelo può essere trasmesso in serie: conversione parallelo/serie.

Il multiplexer può essere utilizzato per trasformare il byte corrispondente al carattere di un tasto pigiato in una sequenza di bit, che attraverso echo può essere inviato ad uno schermo.

Il contrario del multiplexer è il demultiplexer che indirizza un segnale in ingresso verso una delle otto uscite selezionata in base al segnale di controllo sulle linee A,B,C., Viene utilizzato per trasformare un segnale da seriale a parallelo.

# Decodificatore

$n$  valore input (controllo)

$2^n$  valore output

- Il circuito decodificatore abilita una sola delle  $2^n$  uscite quando eccitato dalle  $n$  entrate, esattamente quella corrispondente all'indirizzo dell'entrata.
- E' il circuito utilizzato per il processing delle istruzioni dell'ISA durante il ciclo di fetch.

# Decodificatore

3 valore input (controllo)

$8=2^3$  valore output

A	B	C	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

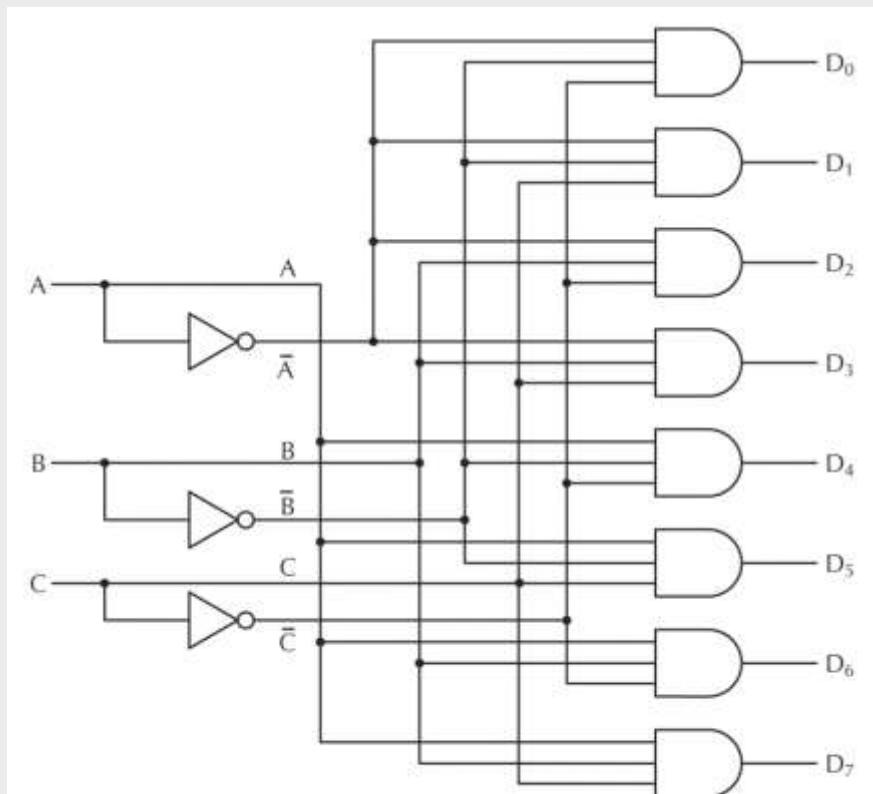
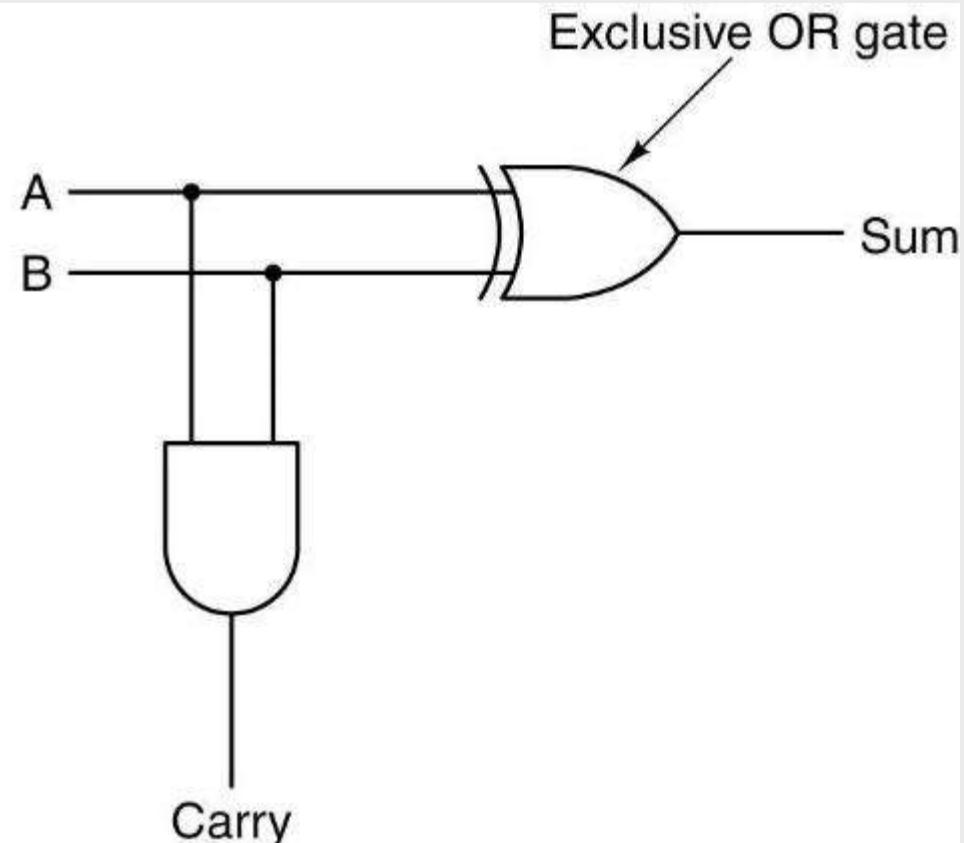


Figura 3.13 Decodificatore da 3 a 8.

# Addizionatore ad un bit

## HALF ADDER

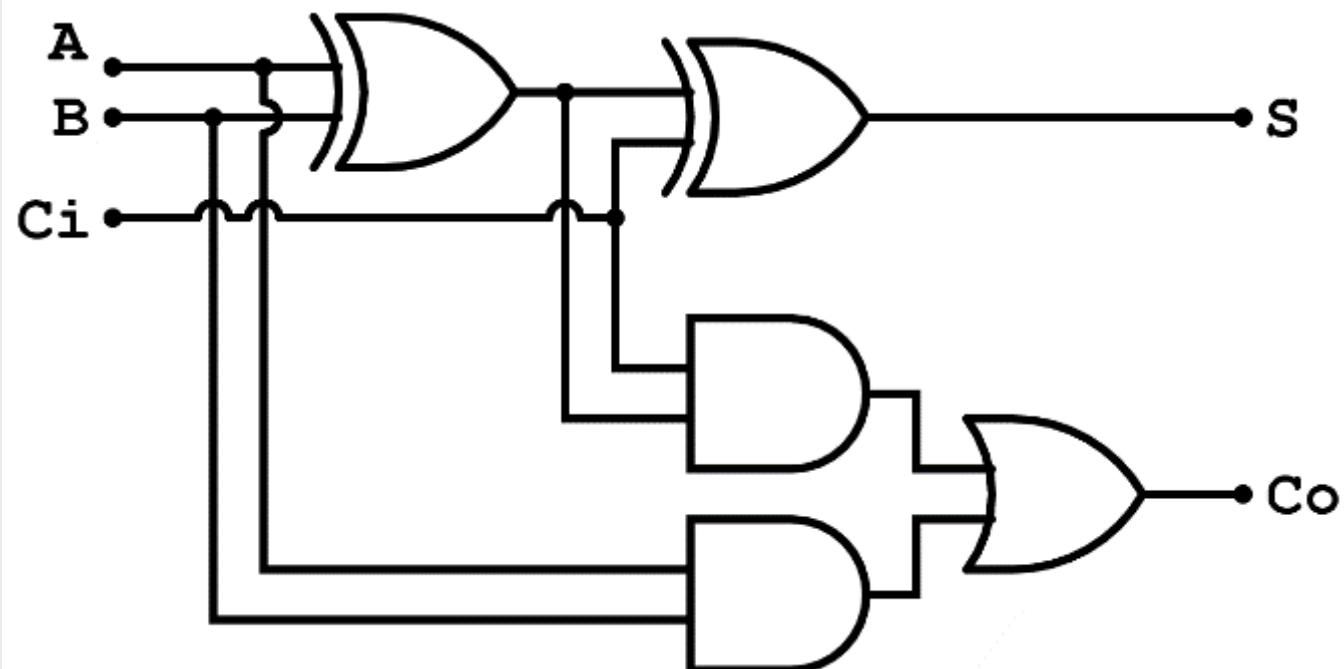
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



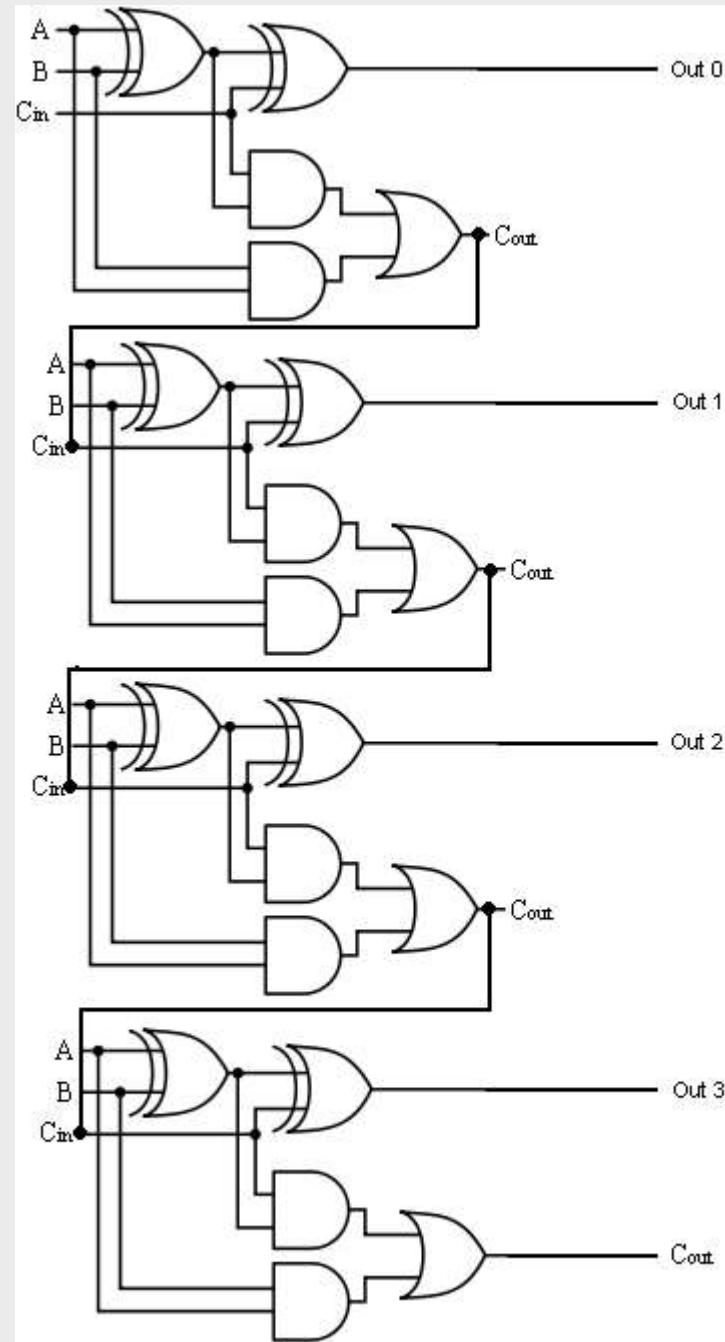
Esercizio: si disegni il circuito di un addizionatore a 2 bit (full adder) per sequenze più lunghe di un bit

# FULL ADDER

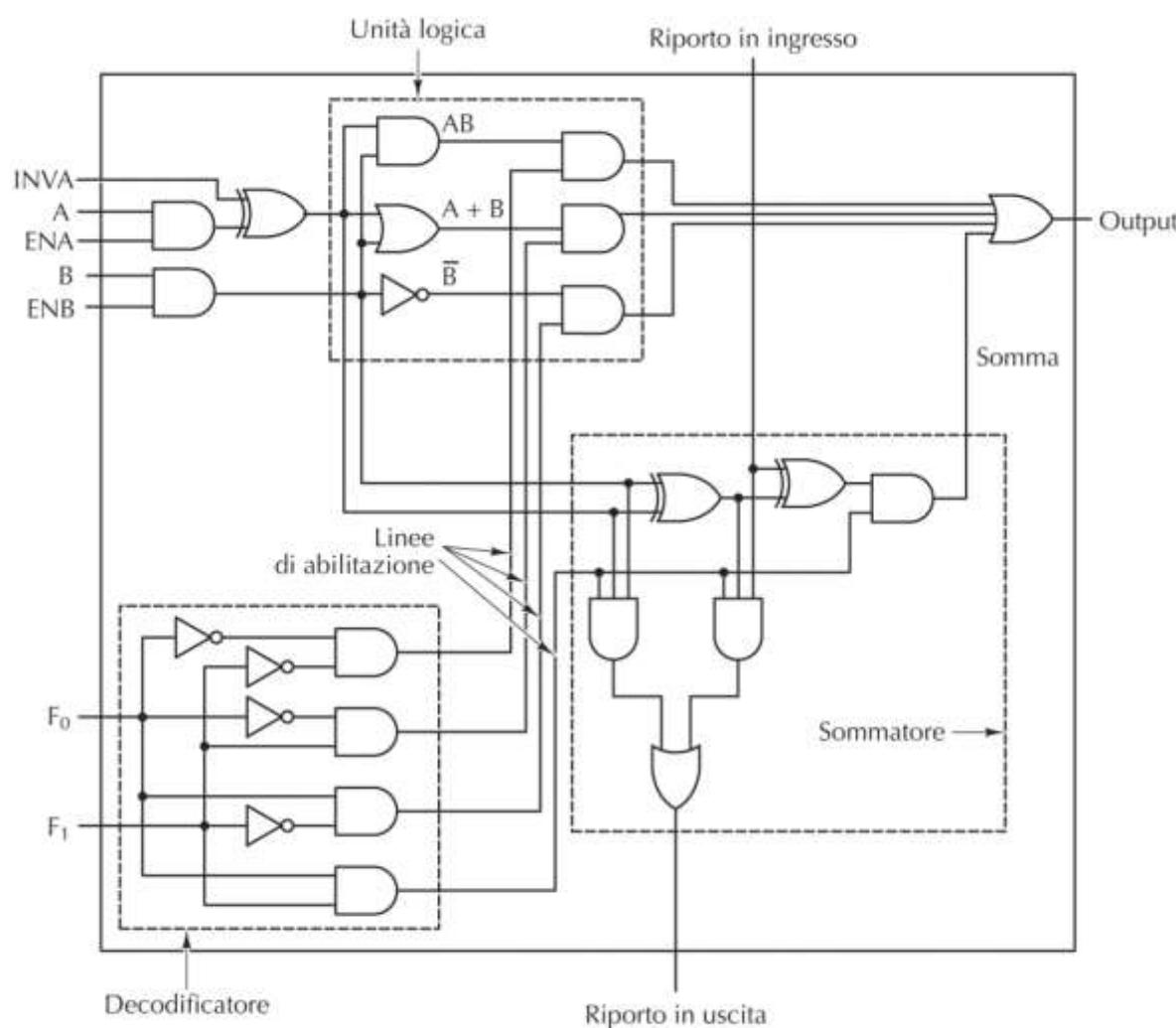
Inputs			Outputs	
A	B	C <sub>i</sub>	C <sub>o</sub>	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



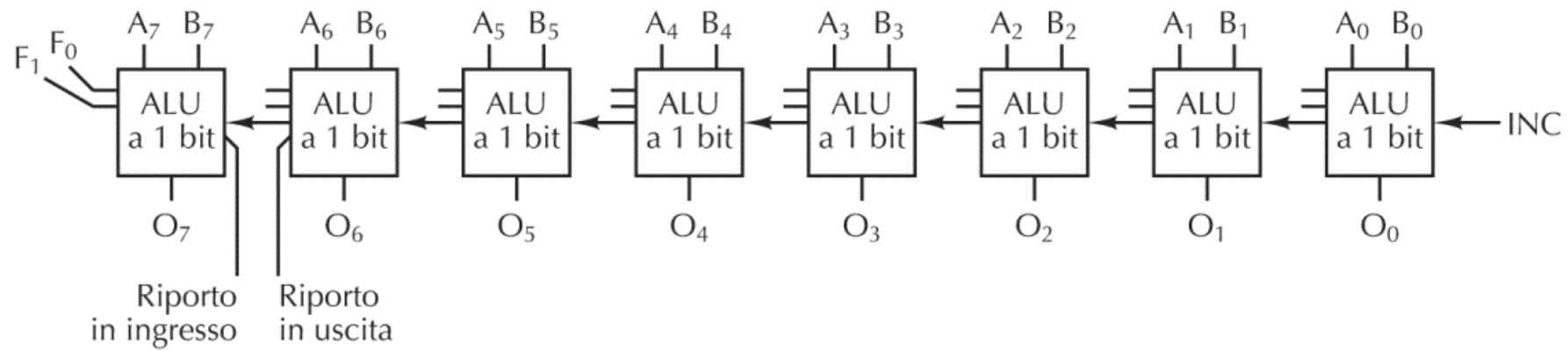
## FULL ADDER A 4 bit



# ALU a 1 bit

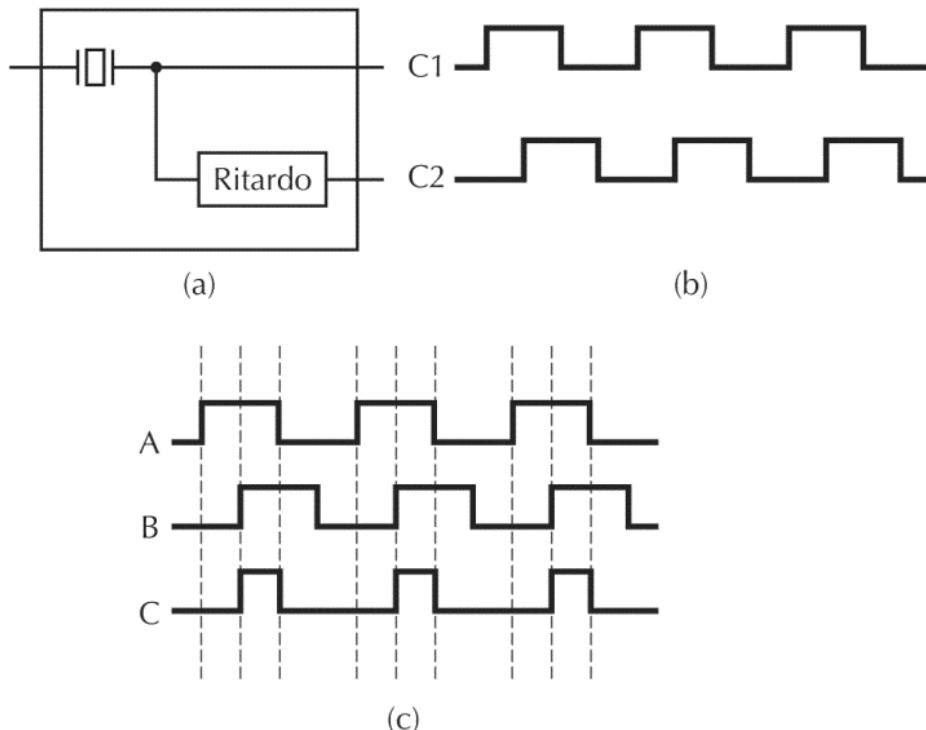


# ALU a 8 bit



**Figura 3.19** Otto ALU a 1 bit connesse per comporre una ALU a 8 bit. Per semplificare non sono mostrati segnali di abilitazione e segnali di inversione.

# Clock



**Figura 3.20** (a) Clock. (b) Diagramma di temporizzazione del clock. (c) Generazione di un clock asimmetrico.

# Latch SR

Un latch è un circuito con capacità di memoria

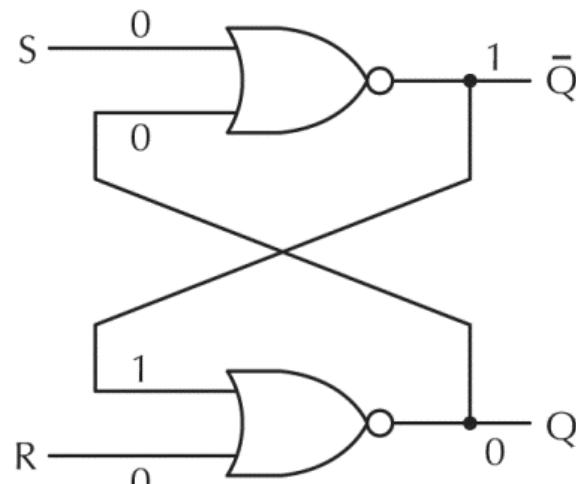
S =setting

R= resetting

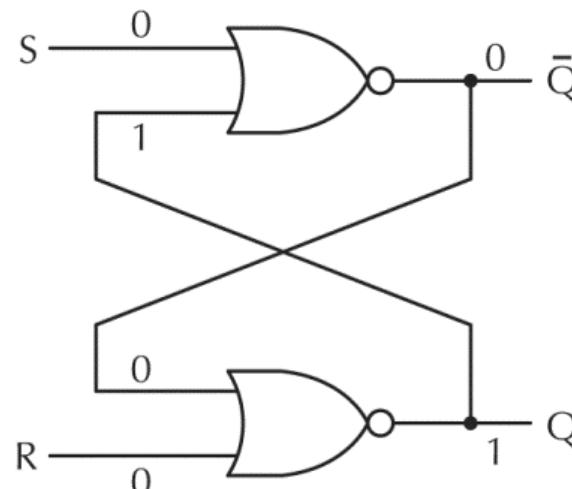
- a) Nello stato 0
- b) Nello stato 1

**BISTABILE!**

S	R	Funzione
0	0	Latch (Memorizzazione)
0	1	Reset: Q = 0 , !Q = 1
1	0	Set: Q = 1 , !Q = 0
1	1	Non è ammesso



(a)



(b)

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

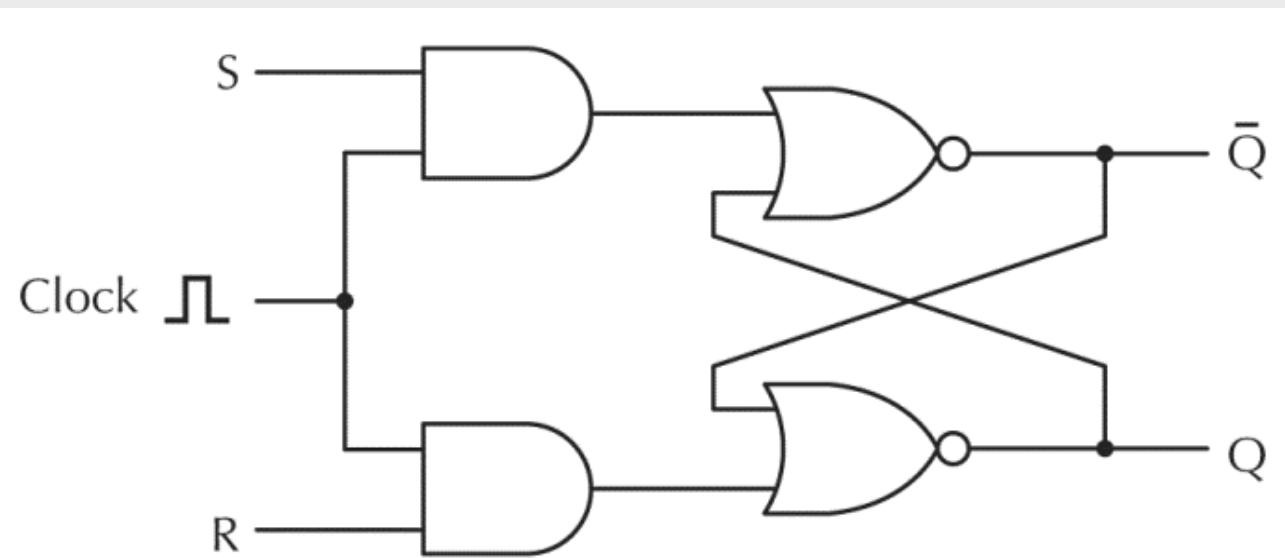
# Latch SR temporizzato

Vogliamo che il cambio di valore avvenga solo in determinati istanti

S =setting

R= resetting

Il clock vale 0, viene impostato ad 1 quando si vuole modificare il contenuto della cella di memoria



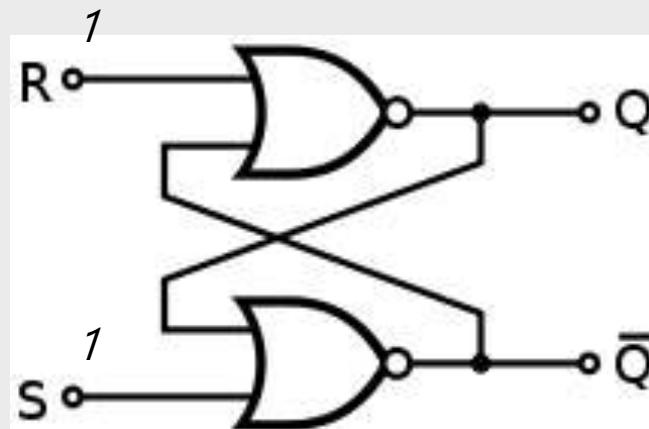
**Figura 3.22** Latch SR temporizzato.

# Ambiguità del Latch SR

Cosa accade se  $S=R=1$ ???

NB: il latch è bistabile

<i>S</i>	<i>R</i>	Funzione
0	0	Latch (Memorizzazione)
0	1	Reset: $Q = 0$ , $\bar{Q} = 1$
1	0	Set: $Q = 1$ , $\bar{Q} = 0$
1	1	Non è ammesso



# Latch D

Utilizzo di un solo input

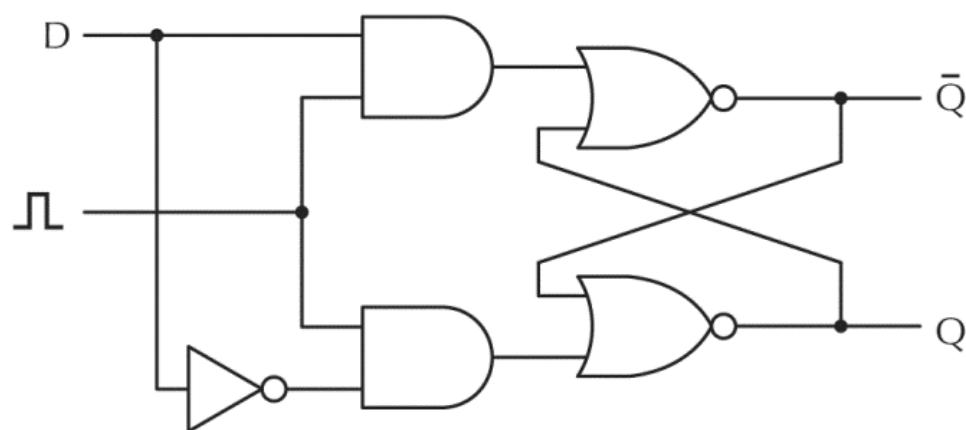


Figura 3.23 Latch D temporizzato.

E	D	Q	$\bar{Q}$
-	-	-	-
1	0	0	1
1	1	1	0
0	-	Q	$\bar{Q}$

NB: per caricare in memoria il nuovo valore di D, occorre un valore 1 sulla linea del clock

# Latch D.... Flip Flop

PRO:

- mantenimento dell'uscita
- adatto all'impiego come interfaccia-memoria nel comando di tastiere o visualizzatori.

CONTRO

- lo stato dell'ingresso si porta all'uscita nell'istante (e per tutto il tempo in cui) l'ingresso E vale 1. Questo può essere causa di comportamenti indesiderati nel caso il componente fosse montato in contesti in cui l'uscita è riportata negata in ingresso. In questo caso essa comincerebbe a oscillare e, non appena E torna a 0, si avrebbe un valore del tutto casuale.

**Flip-flop** D-Edge-Triggered (o semplicemente flip-flop) basate sull'idea di eseguire il campionamento e la marcatura in intervalli temporali ben distinti.

# Flip Flop

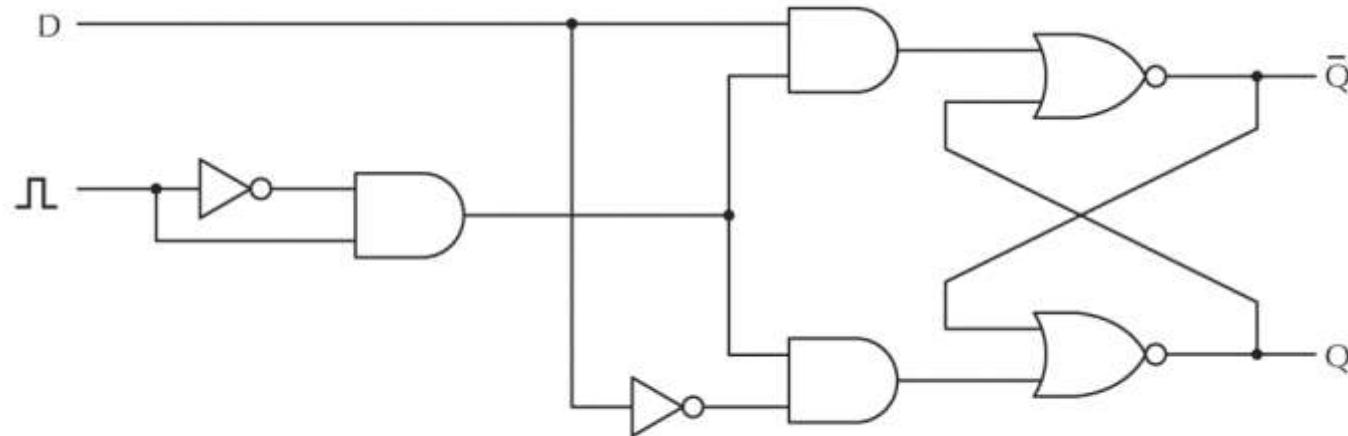


Figura 3.25 Flip-flop D.

La transizione di stato si verifica nella transizione del clock:

- Da 0 a 1 (fronte di salita)
- Da 1 a 0 (fronte di discesa)

La durata dell'impulso non ha importanza.

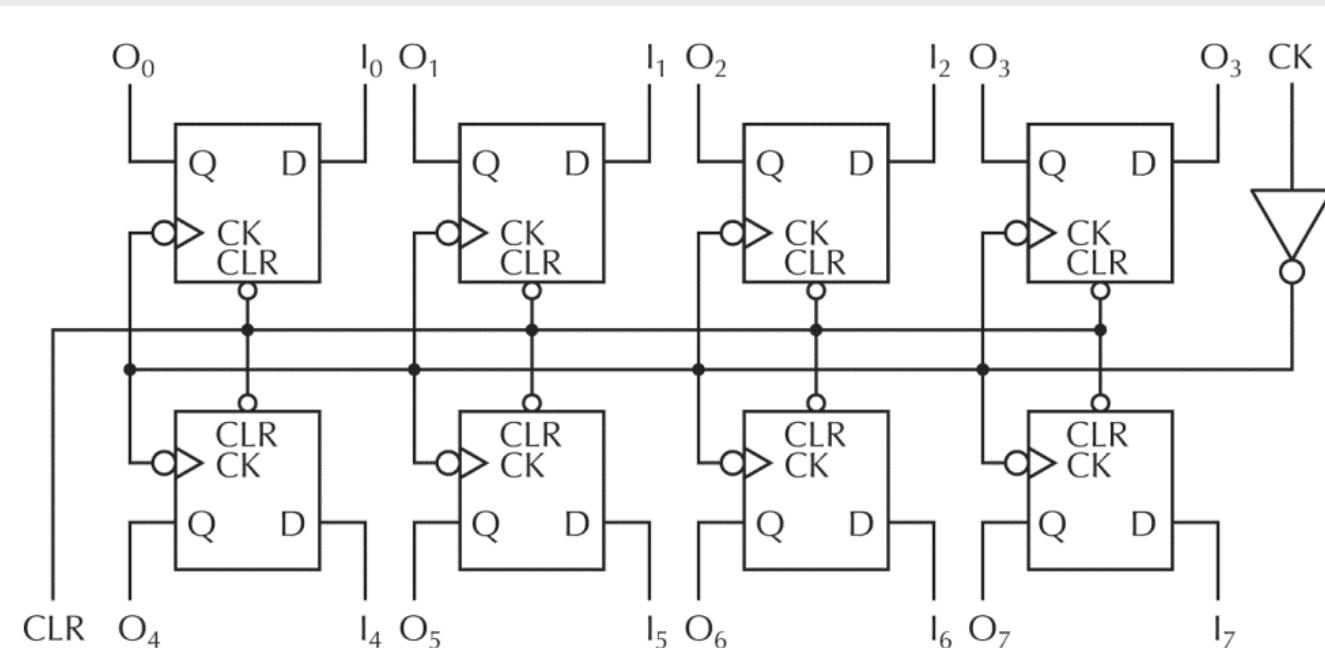
FLIP-FLOP: commutazione su fronte

LATCH: commutazione su livello

# Registro

*Registro a 8 bit*

- 8 flip-flop
- Tutte le linee di clock sono collegate allo stesso clock
- Il valore di ogni bit del registro si modifica solo nell'istante del clk



**Figura 3.27** T Un registro a 8 bit costruito a partire da flip-flop a 1 bit.

# Memoria

Memoria a 4 parole di 3 bit l'una

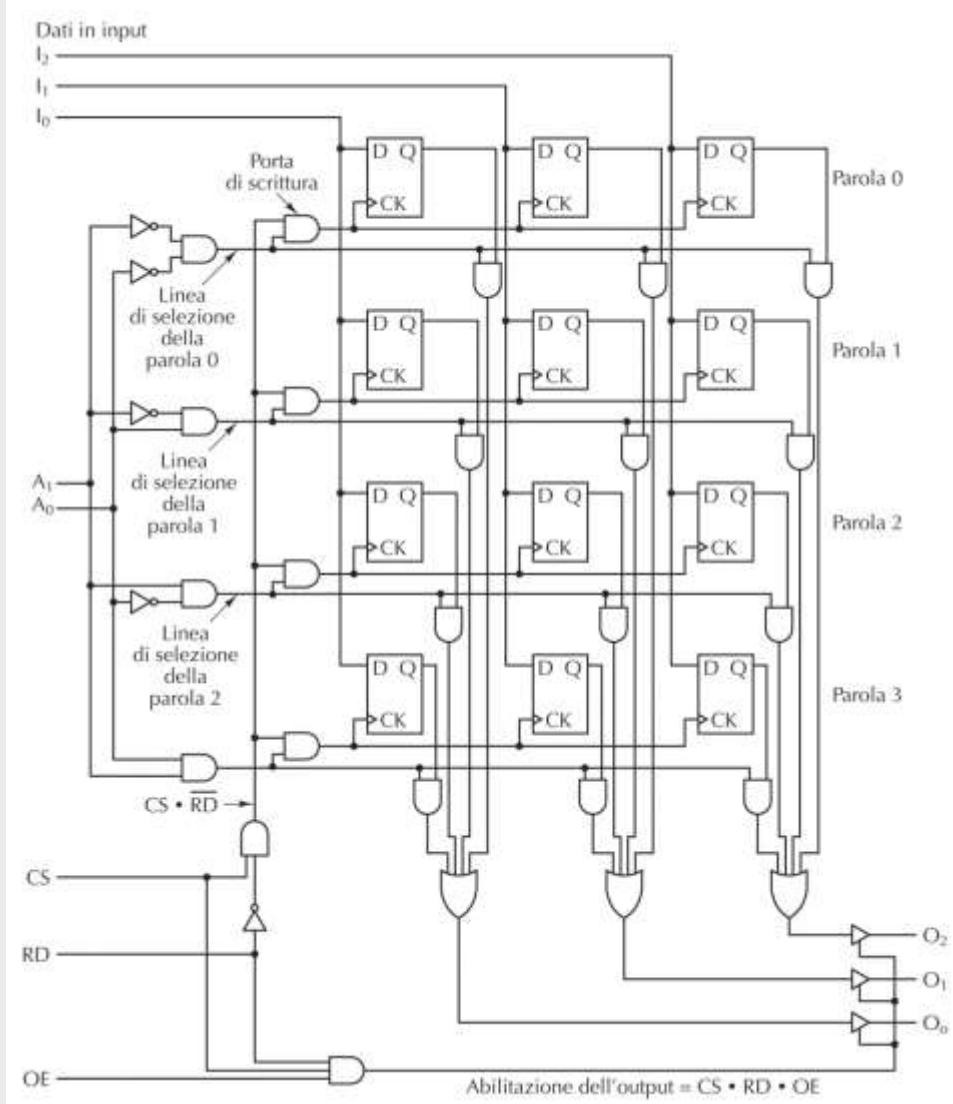
$I_0, I_1, I_2$  : bit in input per ogni parola

$A_0, A_1$ : indirizzano la parola

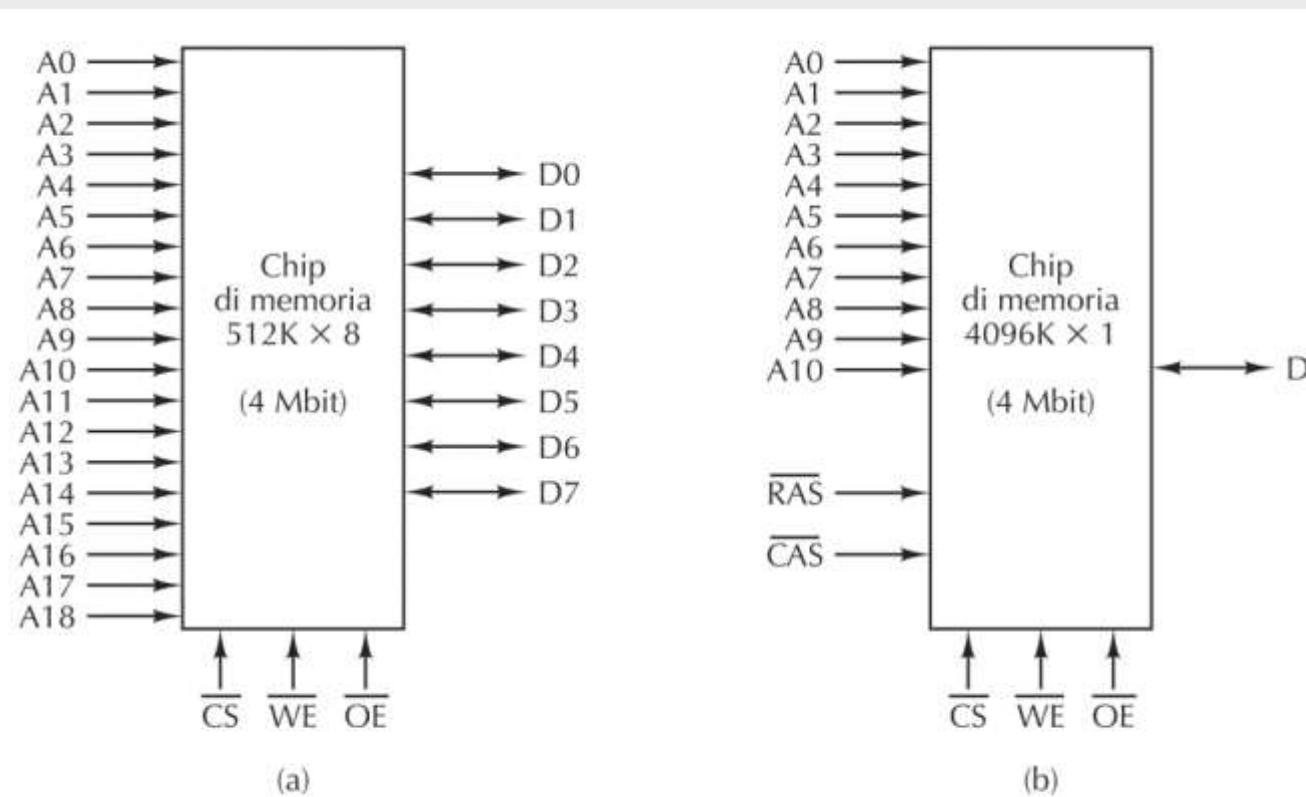
$CS$ : selezione del chip

$RD$ : seleziona la scrittura o la lettura

$OE$ : abilita l'output (Output Enable)



# Chip di Memoria



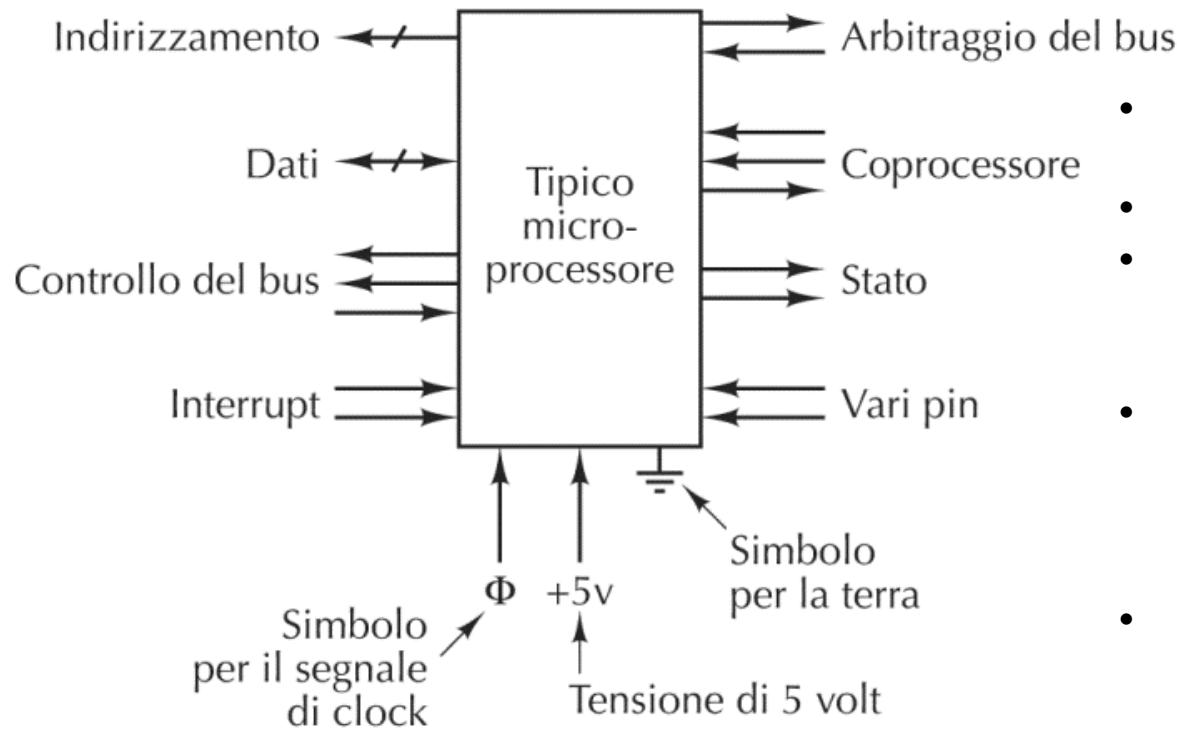
**Figura 3.30** Due modi di organizzare un chip di memoria a 4 Mbit.

# Tipologie di Memoria

Tipo	Categoria	Cancellazione	Byte modificabili	Volatile	Tipico utilizzo
SRAM	Read/write	Elettrica	Sì	Sì	Cache di secondo livello
DRAM	Read/write	Elettrica	Sì	Sì	Memoria centrale (vecchia)
SDRAM	Read/write	Elettrica	Sì	Sì	Memoria centrale (recente)
ROM	Read-only	Impossibile	No	No	Elettrodomestici (prodotti in grandi volumi)
PROM	Read-only	Impossibile	No	No	Dispositivi (prodotti in piccoli volumi)
EPROM	Read-mostly	Raggi UV	No	No	Prototipazione di dispositivi
EEPROM	Read-mostly	Elettrica	Sì	No	Prototipazione di dispositivi
Flash	Read/write	Elettrica	No	No	“Pellicola” per macchine fotografiche digitali

**Figura 3.32** Confronto tra vari tipi di memoria.

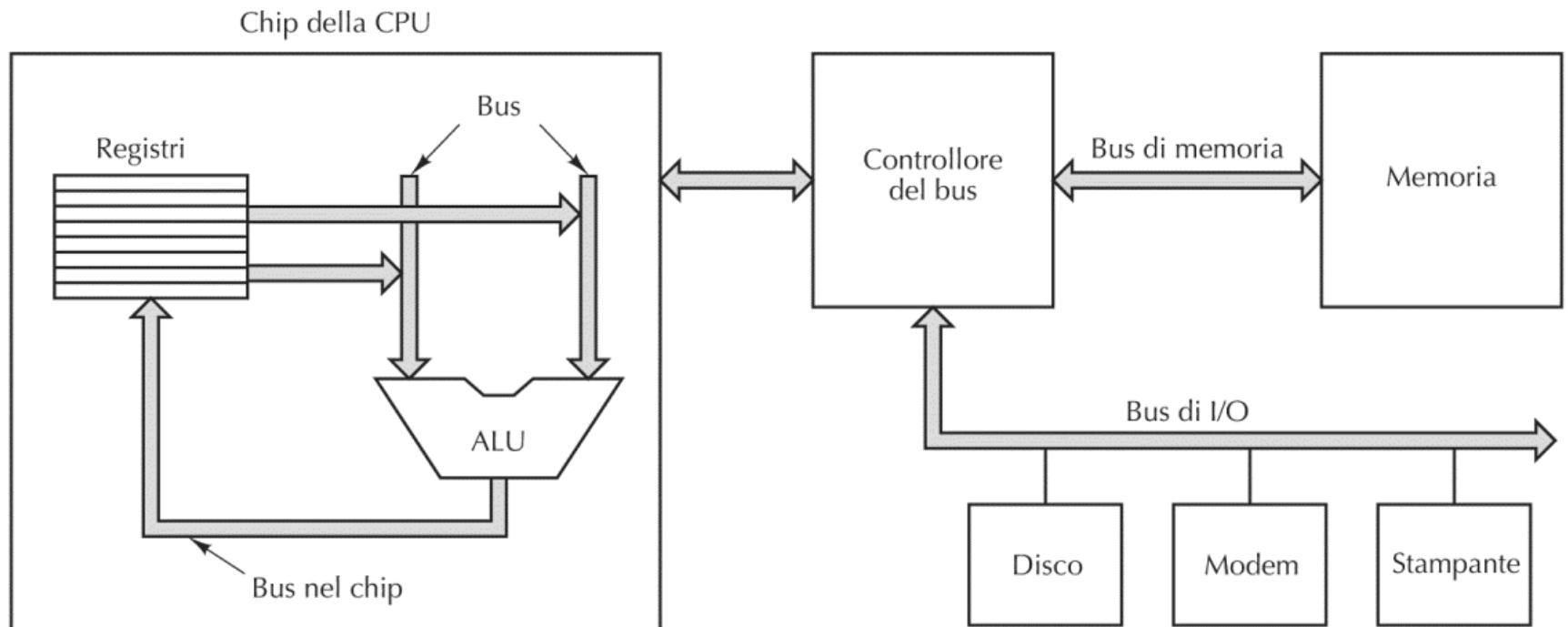
# CPU



- *linee di indirizzamento: : 4,8,16, 20, 22, 32.*
- *linee dati: 4,8,16, 32, 64.*
- *linee di controllo del bus: servono per sincronizzare le trasmissioni*
- *linee di arbitraggio sul bus: gestire i conflitti di richiesta del bus da parte dei diversi dispositivi*
- *Quelle di stato per garantire la compatibilità con precedenti processori*

**Figura 3.34** Una generica CPU. Le frecce indicano i segnali di input e di output. I trattini diagonali indicano pin multipli; per una specifica CPU un valore ne indica la quantità.

# CPU



**Figura 3.35** Sistema di un calcolatore con più bus.

# BUS: master-slave

- Le relazioni tra i dispositivi sul bus sono regolati da una relazione di tipo Master/Slave.

Master	Slave	Esempio
CPU	Memoria	Prelievo delle istruzioni e dei dati
CPU	Dispositivo di I/O	Inizio del trasferimento dei dati
CPU	Coprocessore	Passaggio dell'istruzione al coprocessore da parte della CPU
I/O	Memoria	DMA (Direct Memory Access)
Coprocessore	CPU	Prelievo degli operandi dalla CPU da parte del coprocessore

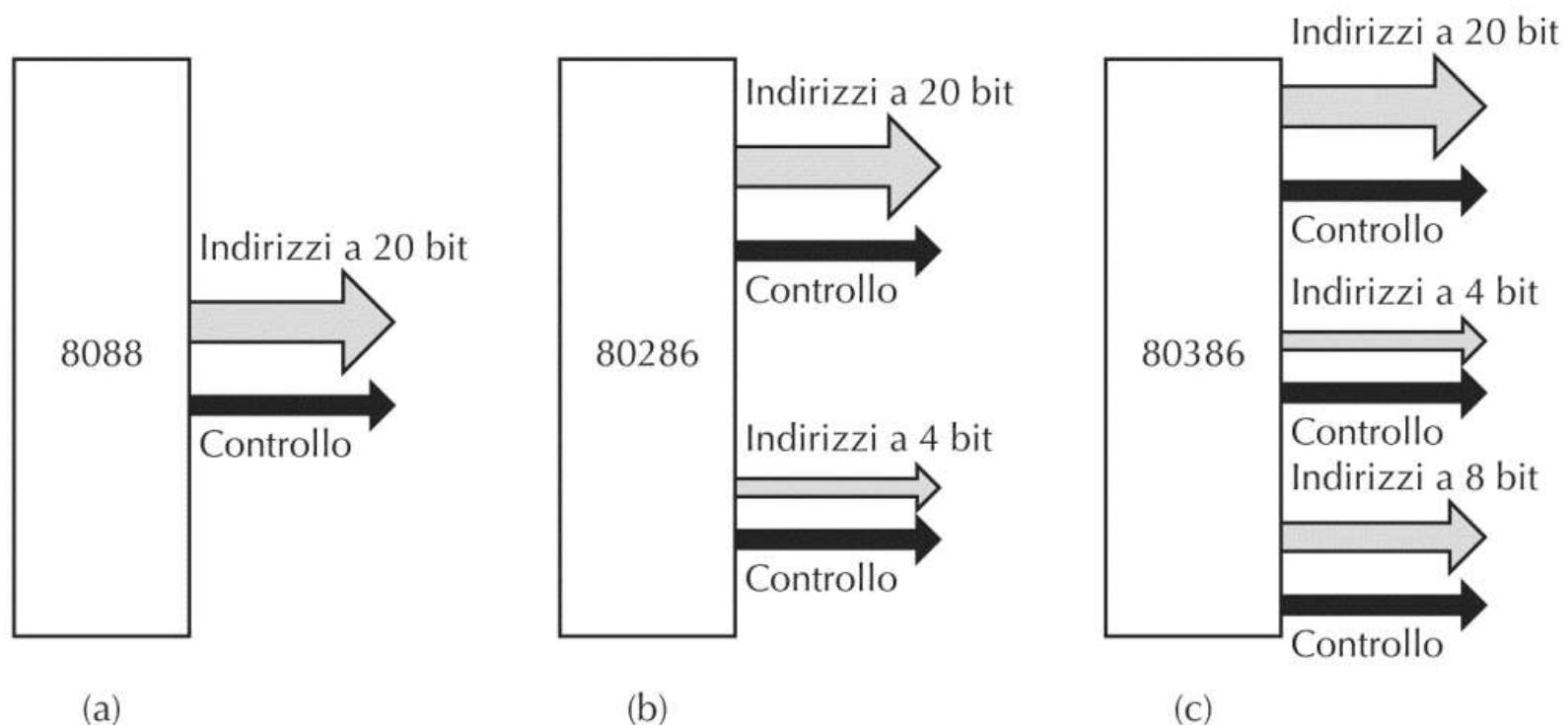
**Figura 3.36** Esempi di master e slave del bus.

*Quando si progetta un bus si tiene presente:*

- L'ampiezza;*
- La temporizzazione;*
- L'arbitraggio;*
- Le operazioni che consente.*

*Da queste scelte dipende la velocità sul bus.*

# BUS: evoluzione nel tempo

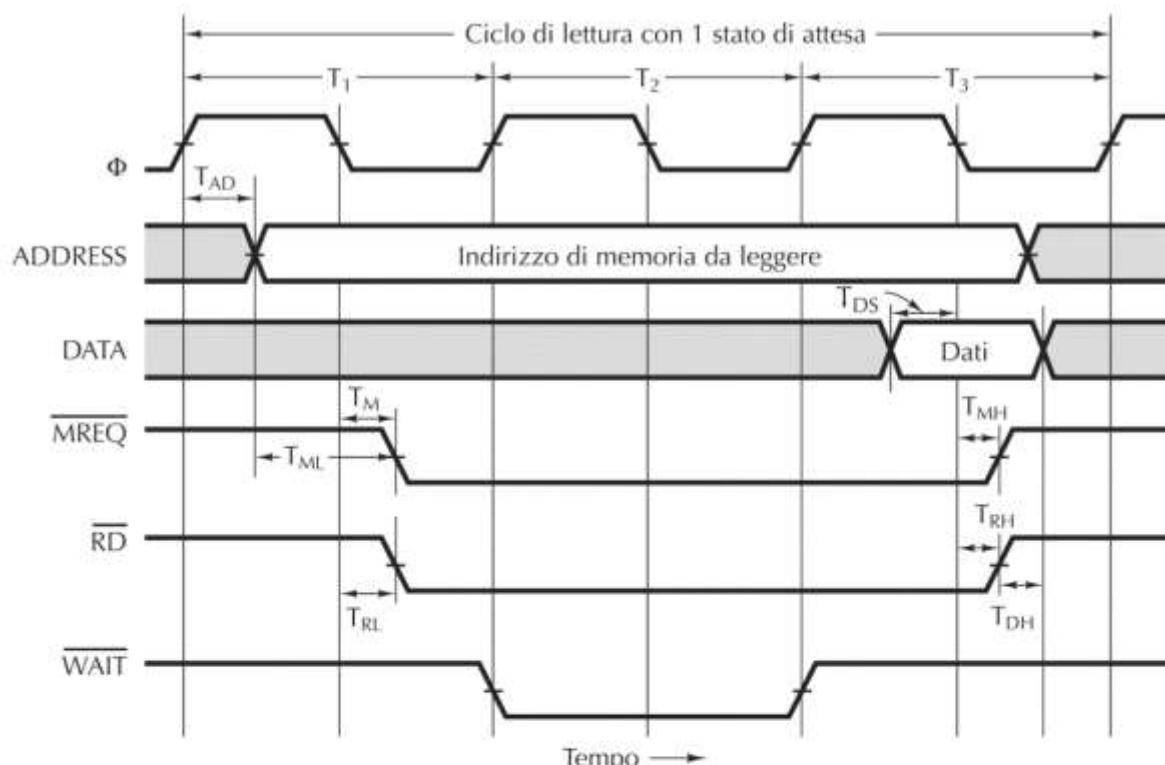


**Figura 3.37** Crescita nel tempo degli indirizzi del bus.

# BUS: temporizzazione

- I bus **sincroni** sono pilotati da un quarzo che genera un'onda quadra tra 5 e 133 MHz, tutte le operazioni vengono scandite da questi impulsi.
- I bus **asincroni** funzionano con il meccanismo dell'invio della domanda e dell'attesa della risposta - non hanno bisogno di clock.

# BUS SINCRONO



Simbolo	Parametro	Min	Max	Unità
$T_{AD}$	Ritardo dell'output dell'indirizzo		4	nsec
$T_{ML}$	Indirizzo stabile prima di MREQ		2	nsec
$T_M$	Ritardo di MREQ rispetto al fronte di discesa di $\Phi$ in $T_1$		3	nsec
$T_{RL}$	Ritardo di RD rispetto al fronte di discesa di $\Phi$ in $T_1$		3	nsec
$T_{DS}$	Tempo di impostaz. dei dati prima del fronte di discesa di $\Phi$	2		nsec
$T_{MH}$	Ritardo di MREQ rispetto al fronte di discesa di $\Phi$ in $T_3$		3	nsec
$T_{RH}$	Ritardo di RD rispetto al fronte di discesa di $\Phi$ in $T_3$		3	nsec
$T_{DH}$	Tempo di mantenimento dei dati dopo la negazione di RD	0		nsec

(b)

- $T_1, T_2, T_3$  : 3 clk
- Il primo ciclo inizia sul fronte di salita di  $T_1$  e corrisponde all'invio dell'indirizzo della locazione
- MREQ indica la richiesta di accesso alla memoria
- RD indica se si vuole leggere o scrivere
- la memoria risponde asserendo il WAIT indicando alla CPU di non aspettarla
- All'inizio di  $T_3$  la memoria nega il wait dichiarando di avere pronta
- al terzo clock i dati possono essere trasferiti

# BUS: asincrono

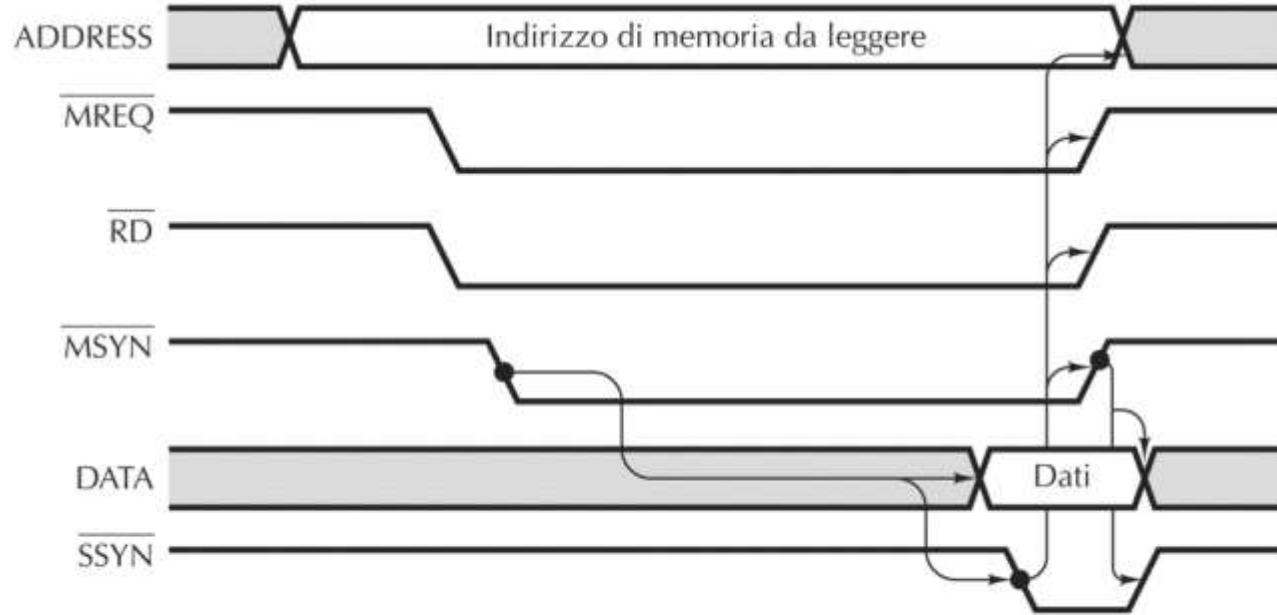


Figura 3.39 Funzionamento di un bus asincrono.

- No clock
- Quando ADDRESS viene lanciato l'indirizzo si propaga sul bus,
- subito dopo viene inviato MREQ e RD ed è con MSYN ( master syncrinism) che si avvia l'operazione di trasferimento dati
- Concluso il trasferimento con la risposta SSYN (slave sincronism) dello slave, viene chiusa la comunicazione
- Successivamente può iniziare una nuova comunicazione

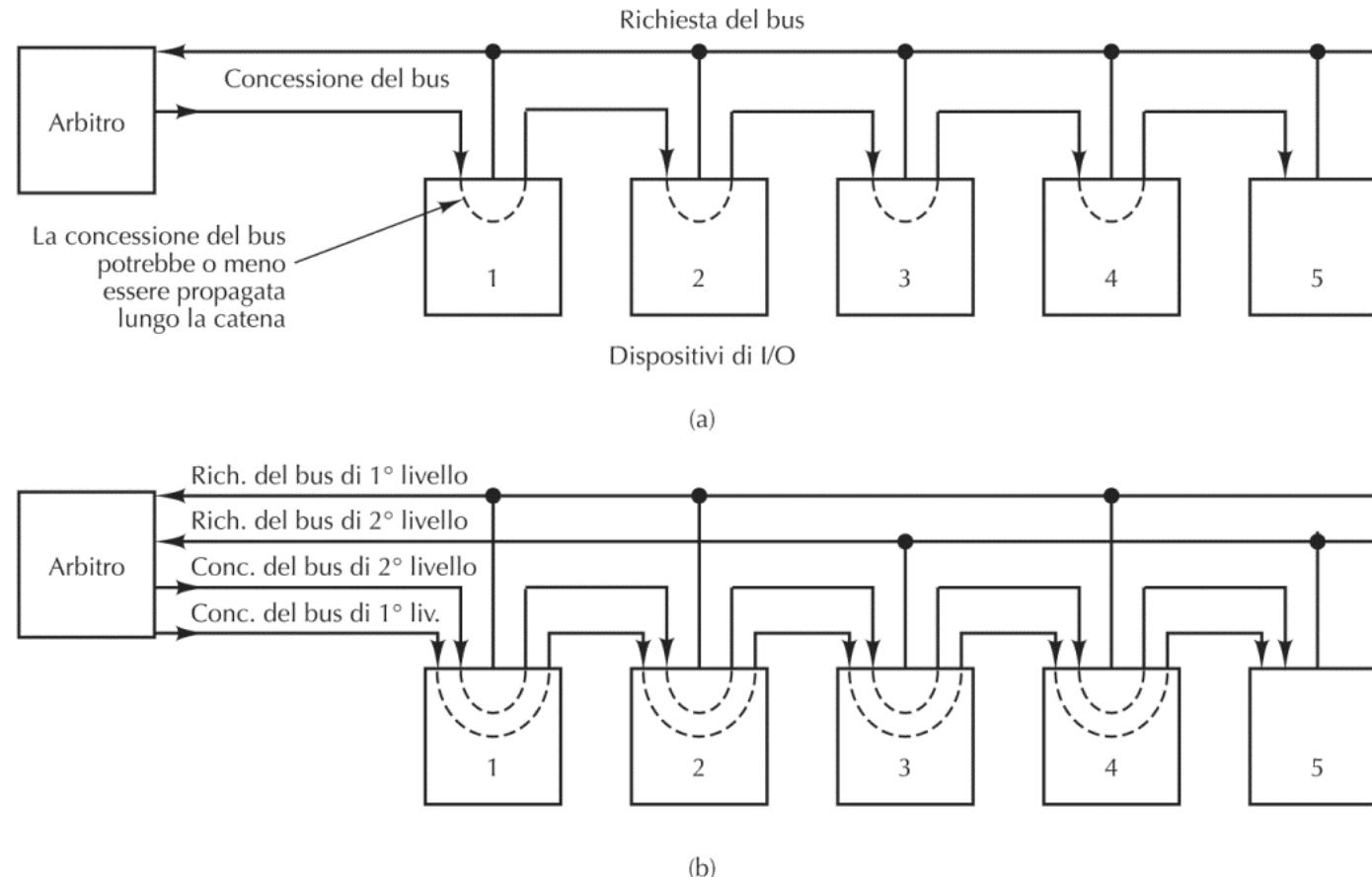
# BUS: arbitraggio

**PROBLEMA:** Come decidere quale dispositivo può accedere al BUS

- **Daisy chaining:**
  - l'arbitro è integrato nel chip della cpu (quasi sempre)
  - tutti i dispositivi sono collegati ad una unica linea nell'ordine che si desidera dare loro (l'ordine corrisponde alla distanza del dispositivo dall'arbitro =livello di priorità)
  - L'arbitro vede solo se c'è una richiesta, se c'è concede il bus e se ne serve il dispositivo più vicino all'arbitro lungo il «festone», che è quello che ha fatto la richiesta, se non l'ha fatta si passa oltre al successivo, la concessione passa lungo tutto il festone di dispositivo in dispositivo fino al dispositivo che ha fatto richiesta.

In alcuni sistemi invece di un festone si usano anche 2, 4 o 8 livelli e così si può assegnare una priorità in base alla distanza e al livello di priorità del festone.

# BUS: arbitraggio



**Figura 3.40** (a) Arbitro del bus centralizzato e a un livello che utilizza un collegamento a festone.  
(b) Stesso arbitro, ma a due livelli.

# BUS: operazioni

SMP – macchine odierne multiprocessore

Cosa accade se più dispositivi richiedono l'uso del bus contemporaneamente?

Soluzione:

- dotare il bus di una ulteriore linea:
  - 0 il bus può essere concesso
  - 1 bus occupato, chi fa richiesta deve aspettare.
  - Ma se quando è 0 sono in due a fare richiesta contemporanea????
    - Soluzione: leggi-modifica-scrivi operazione atomica sul bus

# Tipologie di BUS

- Universal Serial Bus, anche noto come bus “Industrial Serial Architecture” (ISA)
  - utilizzato per connettere tastiere, mouse, stampanti ecc... cioè dispositivi lenti
- PCI (Peripheral Component Interconnected).
  - bus sincrono con relazione master slave tra trasmittente e ricevente,
  - le stesse 64 linee del bus vengono utilizzate sia per indirizzi che per dati con il meccanismo del multiplexing:
    - nel primo ciclo viene immesso l'indirizzo sul bus,
    - nel secondo viene rimosso l'indirizzo ed il bus viene invertito in maniera che lo slave possa utilizzarlo,
    - nel terzo si generano in output i dati richiesti.

# Tipologie di BUS: PCI

Il **Bridge** coordina i dispositivi e le loro diverse velocità.

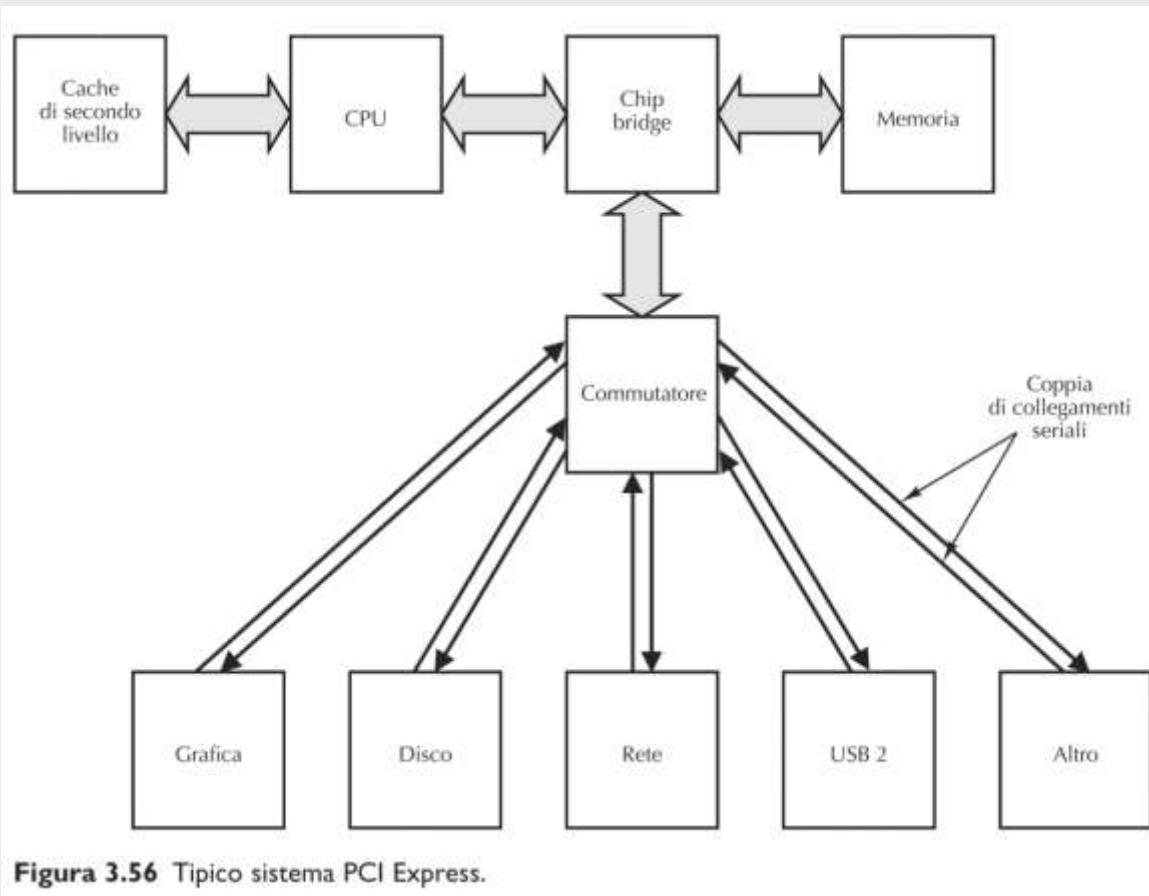


Figura 3.56 Tipico sistema PCI Express.

- Il commutatore che può stare nel Bridge oppure nello stesso chip della CPU.
- Le connessioni consistono di due fili, uno per la massa e l'altro su cui viaggiano i segnali.
- La connessione è punto punto e seriale.
- Possibilità di inserimento e soppressione a caldo dei dispositivi.
- I commutatori sono piccoli dispositivi viene favorita la migliore integrazione e l'uso in computer di più piccole dimensioni ( smart phone, tablet, embedded).

# Tipologie di BUS: USB

## UNIVERSAL SERIAL BUS

Il sistema USB è composto da un **Hub principale** che si collega:

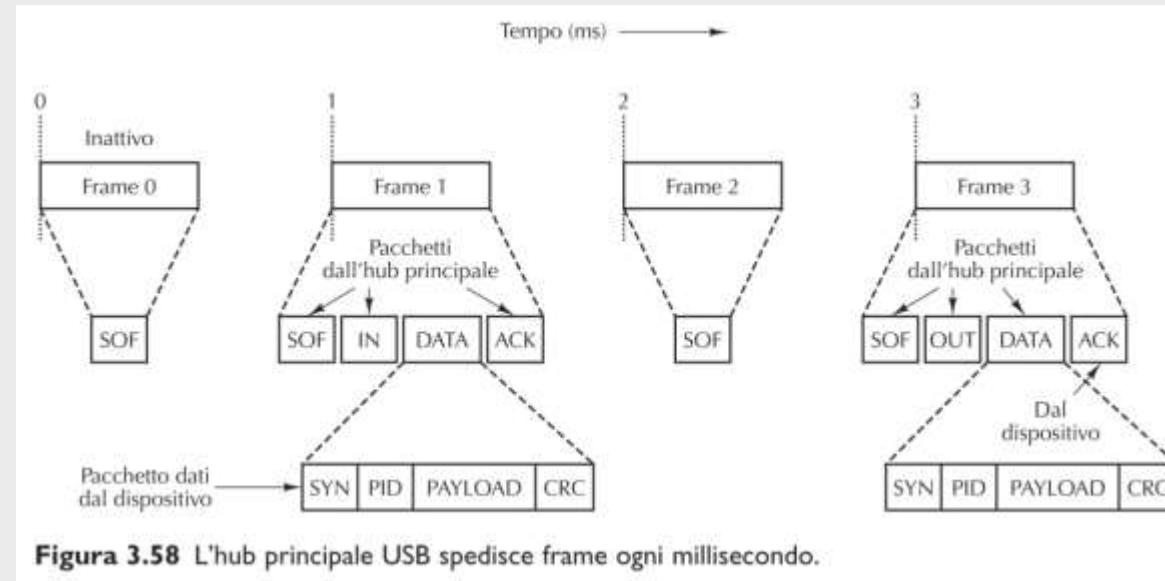
- al bus del sistema
- ai cavi dei dispositivi organizzati ad albero per dare la possibilità di connessioni multiple. La radice dell'albero è nell'hub principale.

Il cavo di connessione consiste di 4 fili:

- 2 per i dati,
- 1 per GND,
- 1 per la tensione a 5V.

# Tipologie di BUS: USB

## Invio di Frame



## 4 tipi di frame

1. controllo utilizzati per configurare i dispositivi, assegnare loro i comandi e interrogarli;
2. Isocroni (microfoni, altoparlanti, ecc.): spediscono a precisi intervalli di tempo ma non possono richiedere la trasmissione in caso di errore;
3. Bulk usati per la trasmissione di grandi volumi di dati;
4. Interrupt

# BUS: operazioni e interrupt

## CHIP 8259A Intel

Come fa un dispositivo ad indicare di voler essere servito?

Interrupt..problemi analoghi ai precedenti

1. *IRx (interrupt Request): richiesta di interrupt da un dispositivo*
2. *Il chip 8259A asserisce la linea INT (INTerrupt)*
3. *Quando la CPU può gestire l'interupt , asserisce INT A (INTerrupt Acknowledgement),*
4. *Il chip 8259A specifica l'input che ha generato l'interrupt,*
5. *La CPU usa quel numero come vettore di interrupt in una tabella che seleziona la procedura.*
6. *La CPU può leggere o scrivere nei registri del chip 8259A utilizzando i cicli di bus ed i pin: RD (ReaD), WR (Write), CS (ChipSelect) e AO (Anable Output).*

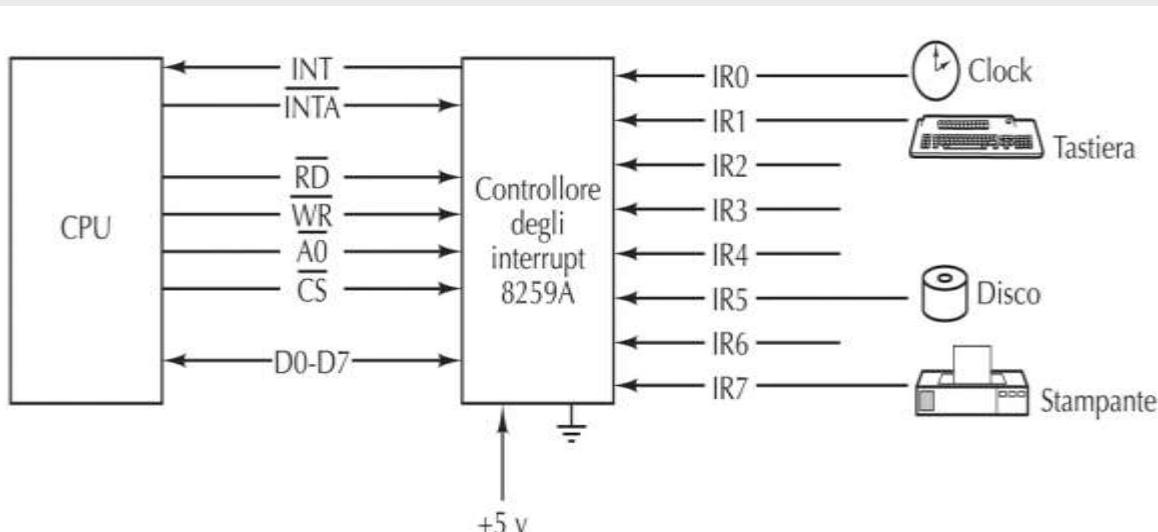


Figura 3.43 Utilizzo del controllore degli interrupt 8259A.

# Un esempio di architettura: Intel i7

Intel Core i7... un po' di numeri

- Introdotto nel novembre 2008
- 4 processori, frequenza di 3,2 GHz
- 447 linee per segnali + 286 per alimentazione + 360 di terra + 62 riservate per utilizzazioni futuri
- La prima versione che usava l'architettura “Nahalem” è poi stata sostituita da una nuova versione “Sandy Bridge”: contiene fino a 6 processori con frequenza di 3.5 GHz

# Un esempio di architettura: Intel i7

Intel Core i7... un po' di sostanza

- 64bit.
- standard IEEE 754 per implementazione in virgola mobile
- numero di processori che va da 2 a 6
- l'hyperthreading (cioè il multithreading simultaneo)
- può eseguire fino a 4 istruzioni per volta: macchina superscalare a 4 livelli
- Ogni processore ha 3 livelli di cache:
  - L1-Dati da 32 KB,
  - L1-Istruzioni da 32 KB,
  - L2 Unificata da 256 KB,
  - L3 Unificata da 4 a 15 MB

# Un esempio di architettura: Intel i7

- Due bus sincroni:
  - uno interno di tipo DDR3 (Dinamic Direct Random Access memory) per l'accesso alla memoria centrale
  - uno PCIe (PCI Express) per i dispositivi di I/O
- Può consumare da 17 a 150W ... problemi di dissipazione del calore.
- Al di sopra può essere montata una ventola di raffreddamento.
- 5 stati diversi con diversi livelli di raffreddamento che vanno dal sonno profondo alla piena utilizzazione.
- Nei vari livelli vengono disattivati alcune funzionalità

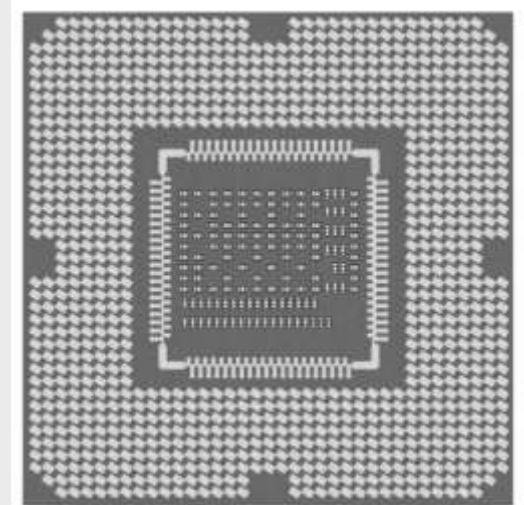


Figura 3.44 Disposizione fisica dei contatti del Core i7.

# Un esempio di architettura: Intel i7

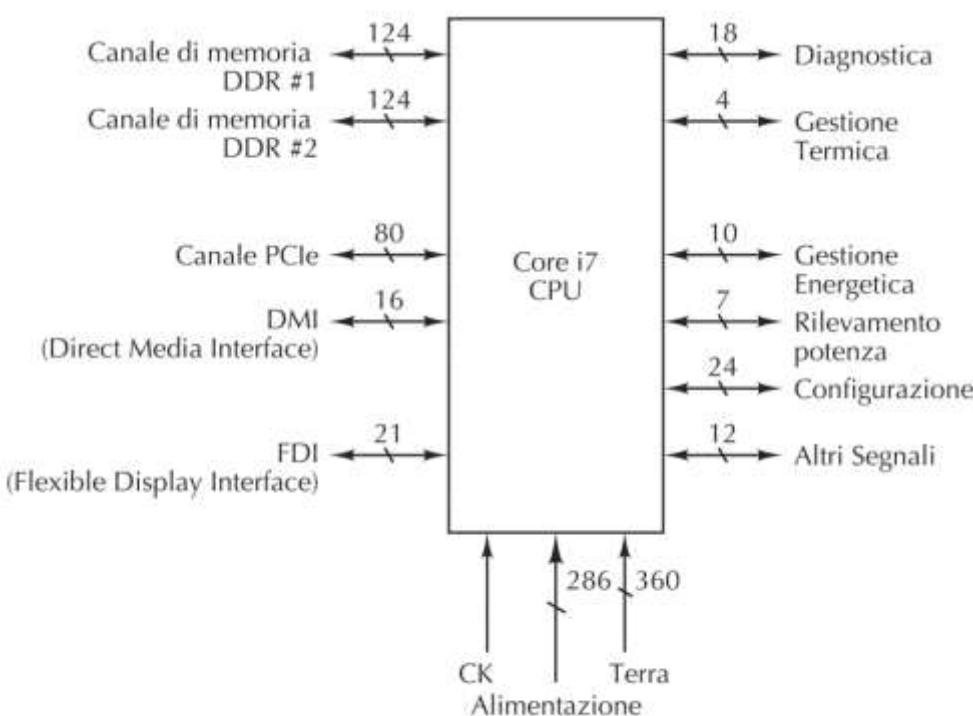


Figura 3.45 Disposizione logica dei contatti del Core i7.

## Lato sinistro:

- Accesso alle memorie,
- gestione delle periferiche
- accesso diretto multimediale
- interfaccia flessibile al display.

## Lato destro:

- gestione termica,
- gestione energetica,
- rilevamento della potenza,
- diagnostica,

# Un esempio di architettura: Intel i7

- Due banchi memoria: DDR #1 e DDR #2, sono compatibili con la DDR3 e lavorano a 666 MHz ciascuno e quindi permettono 1333 milioni di transizioni al secondo.  
La DDR3 ha un'ampiezza a 64 bit e i due banchi lavorano in tandem potendo sopportare programmi fino a 20 GB di dati al secondo.
- Connessione tra CPU e periferiche via bus PCI Express: interfaccia seriale veloce in cui ogni singolo collegamento forma una “lane”. Il Core i7 consente fino a 16 lane offrendo 16 GB/s di trasferimento potendo trasferire comandi read, write, interrupt, comandi di configurazione oltre ovviamente ad indirizzi e dati.
- DMI (Direct Media Interface): connessione tra CPU e chipset, ossia insieme di chip che servono alla gestione delle porte, USB, audio, PCIe, Flash ed anche il DMA e lo fa attraverso il chip ICH10 che al suo interno ha anche il controllo di clock real time.

# Un esempio di architettura: Intel i7

Il bus di memoria DDR3 è strutturato a pipeline a 3 fasi:

**ACTIVATE** della memoria: si apre una riga della DRAM per accessi successivi:

**READ o WRITE**: possono effettuarsi anche accessi multipli;

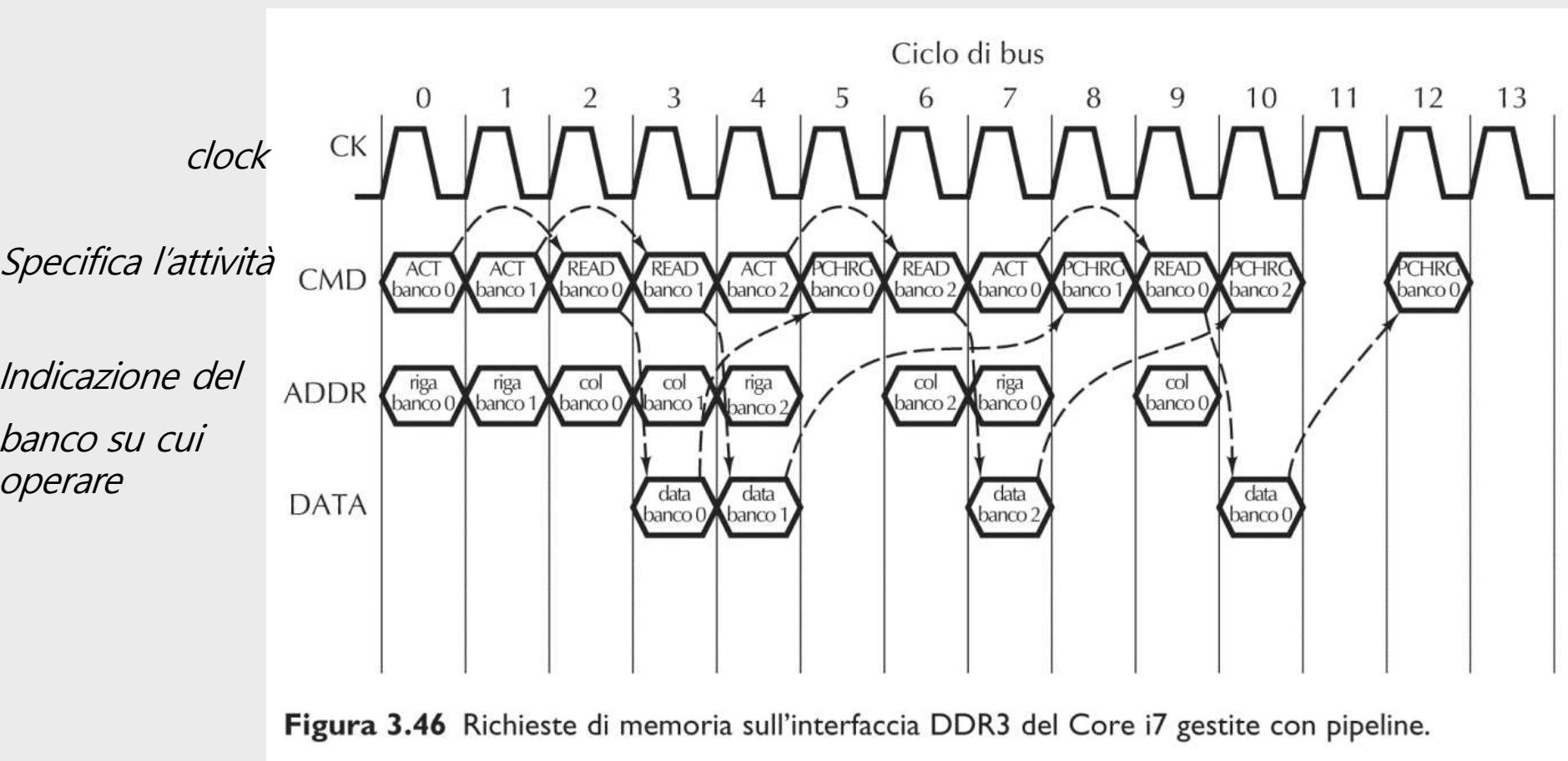
**PRECHARGE**: chiude la riga corrente della DRAM e prepara la memoria per il prossimo ACTIVATE.

L'organizzazione della memoria su più banchi consente l'accesso contemporaneo sui diversi banchi concorrenti.

# Un esempio di architettura: Intel i7

In questo esempio si effettuano 4 accessi di memoria a 3 distinti banchi DRAM.

NB: Le letture avvengono in parallelo sullo stesso chip.



**Figura 3.46** Richieste di memoria sull'interfaccia DDR3 del Core i7 gestite con pipeline.

# Un esempio Intel i7

## STRUTTURA DEL BUS

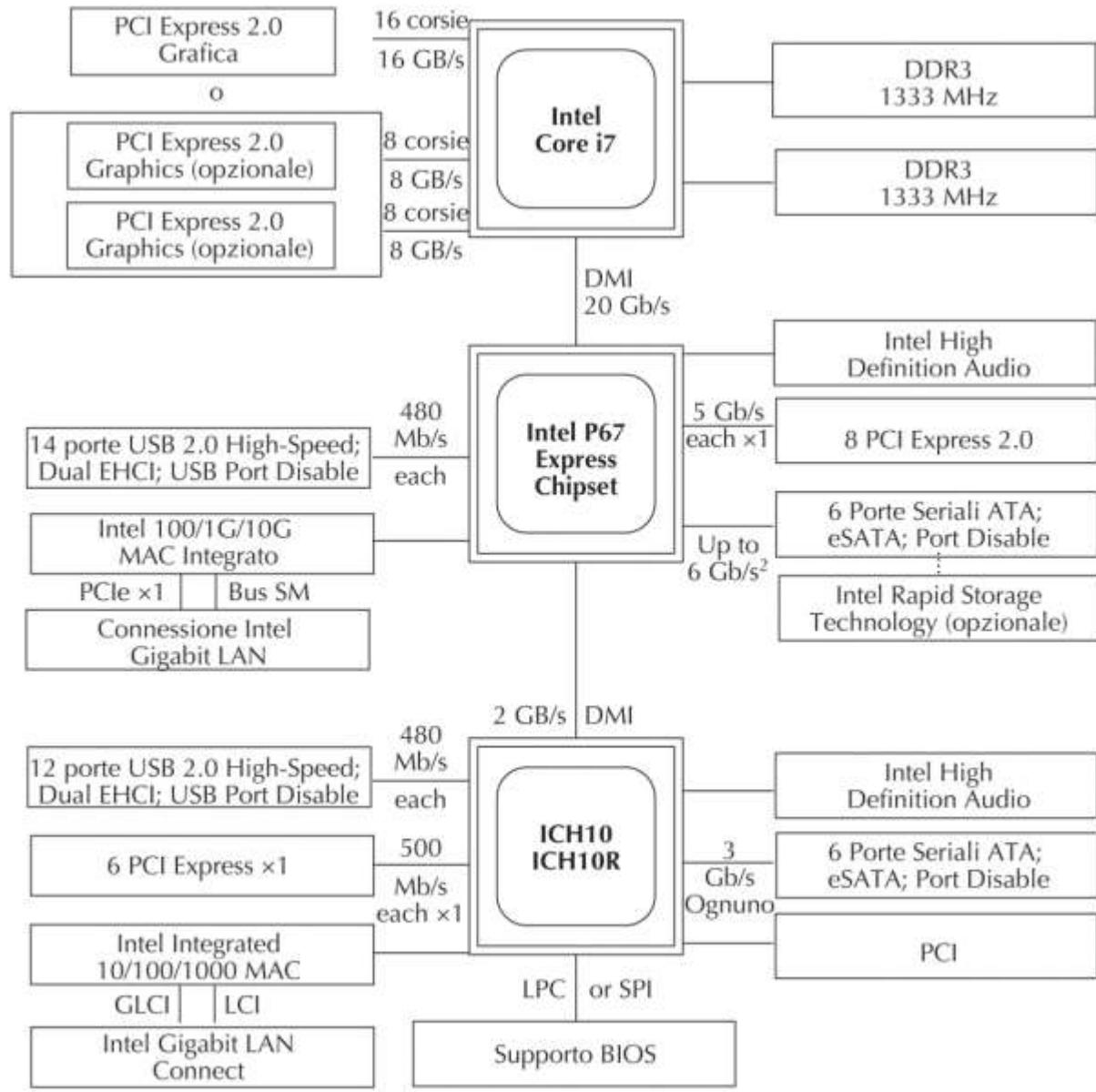


Figura 3.52 La struttura del bus di un moderno sistema Core i7.

# Obiettivi di un OS

**Convenienza nell'uso del calcolatore rispetto ai potenziali utenti**

**Efficienza nell'utilizzo del calcolatore e delle sue parti costitutive**

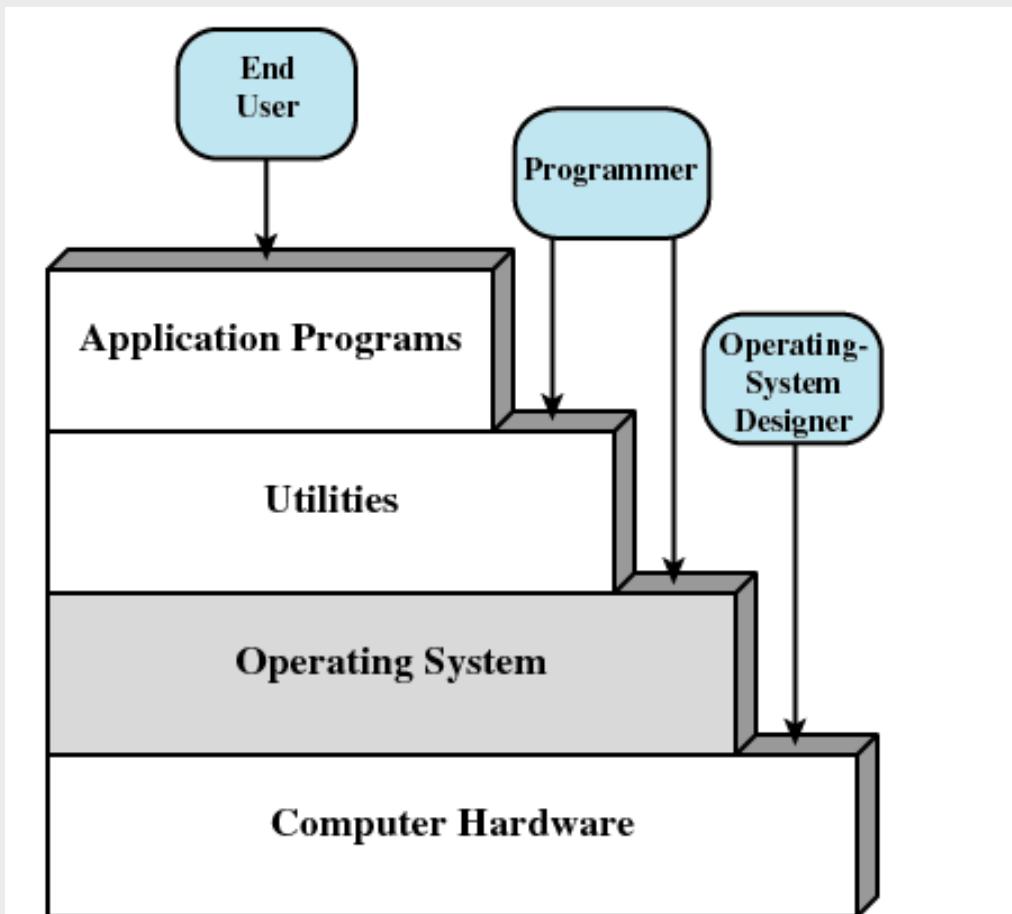
**Capacità di evolversi rispetto a evoluzioni hardware, esigenze degli utenti  
bachi**

e

# Sistema Operativo come interfaccia (convenienza del SO)

Key Word: TRASPARENTE

Struttura Gerarchica di Hardware e Software



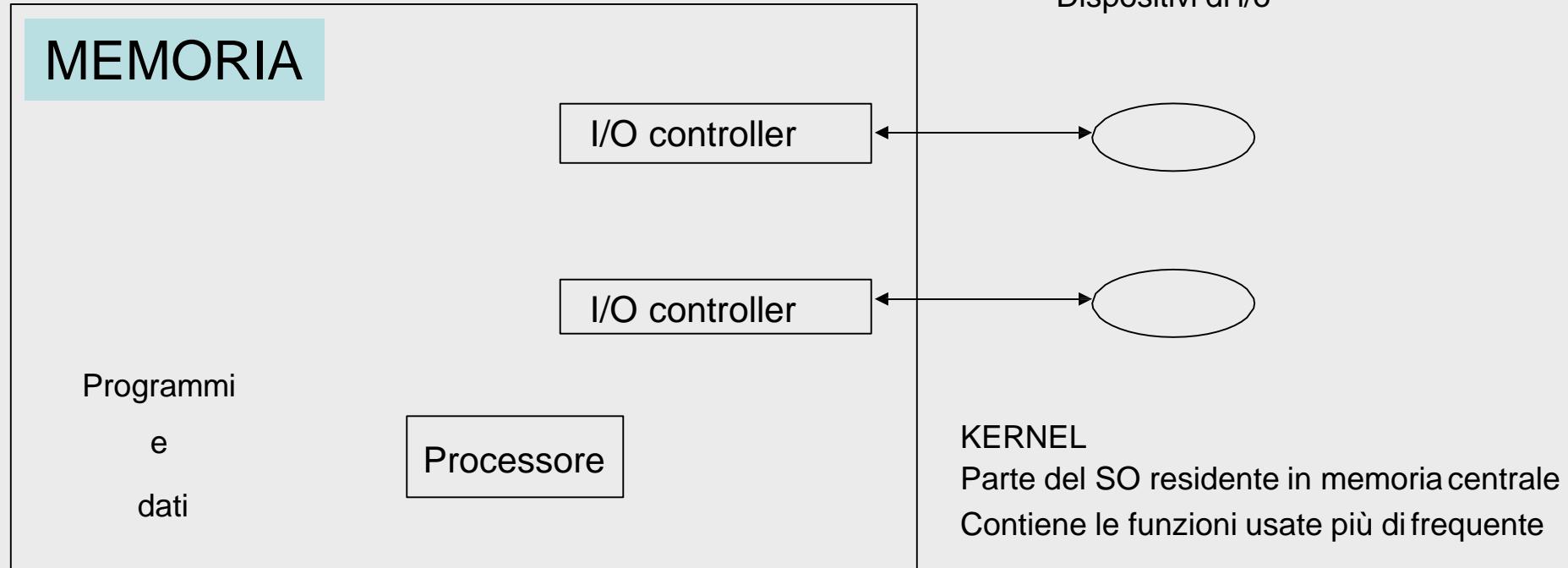
Il sistema operativo:

- nasconde i dettagli hardware al programmatore
- Fornisce una interfaccia per utilizzare il sistema

# Servizi Offerti dal SO

- **Creazione dei programmi:** compilatore, debugger come utilità offerte al programmatore. Non sono parte del SO ma sono accessibili tramite esso
- **Esecuzione dei programmi:** caricamento in memoria dei programmi, inizializzazione dei dispositivi I/O, ecc.
- **Accesso ai dispositivi di I/O:** l'utente/programmatore ignora il set di istruzioni e i segnali dei dispositivi
- **Accesso controllato ai file:** comprensione del formato, meccanismi di protezione, associazione file indirizzi di memoria
- **Accesso al sistema** (inteso in senso lato)
- **Rilevazione e correzione degli errori** hardware o generati da programmi in esecuzione
- **Contabilità e statistiche d'uso delle risorse**, dei tempi di risposta (fine: migliorare le prestazioni)

# SO come gestore delle risorse (efficienza del SO)



Il SO:

- dirige la CPU nell'utilizzo delle altre risorse del sistema e nella temporizzazione dell'esecuzione dei programmi
- decide quando un programma in esecuzione può utilizzare una risorsa
- il processore stesso è una risorsa!!

# Batch Multi-Programmati

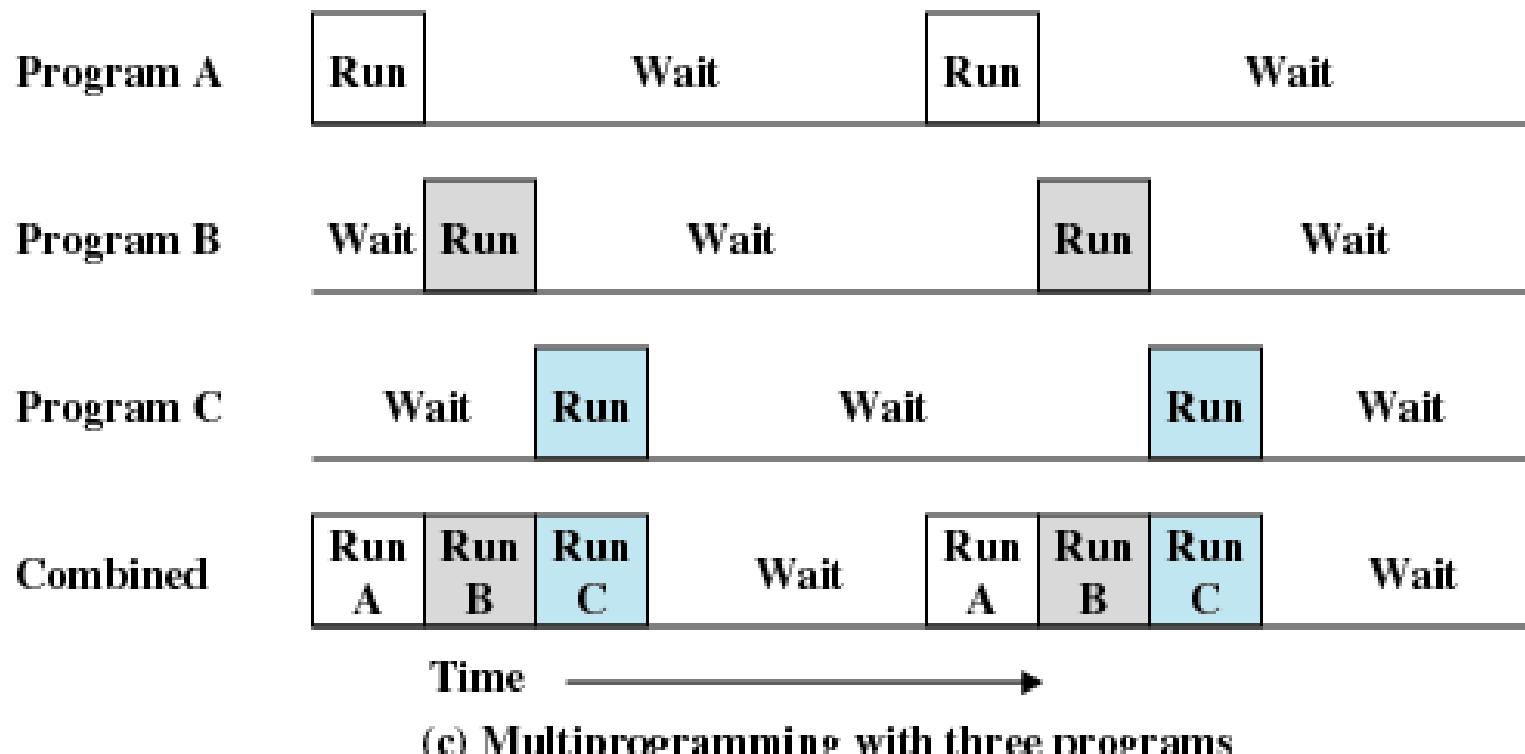
## Mono–Programmazione

Lettura di un record	0.0015 sec.
Esecuzione di 100 istruzioni	0.0001 sec.
Scrittura di un record	0.0015 sec.
TOTALE	0.0031 sec.

$$\text{Percentuale di utilizzo CPU} = \frac{0.0001}{0.0031} = 0.032 = 3.2\%$$

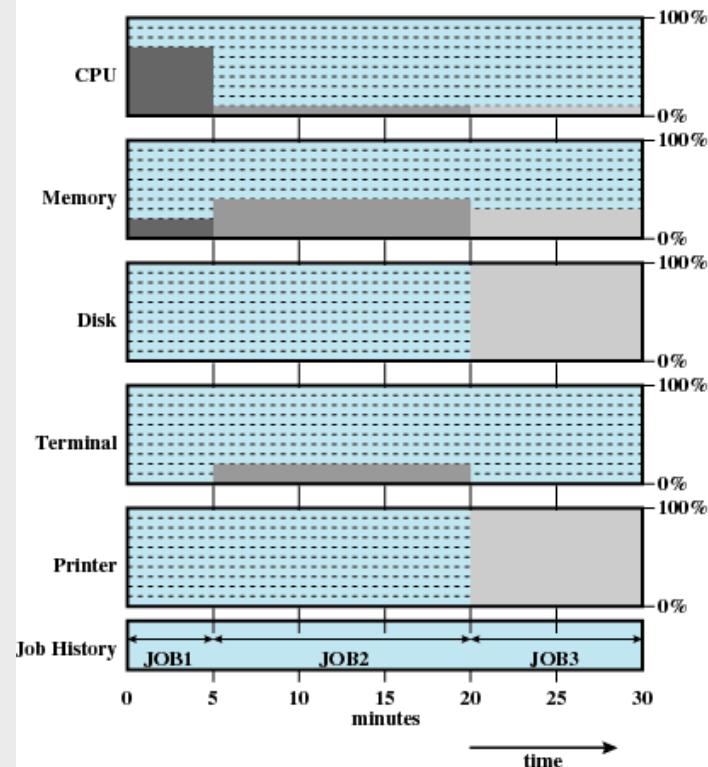
- Multi–programmazione:
  - Presenza di più programmi in memoria
  - Obiettivo: limitare l'inattività del processore, quando un job effettua una operazione di I/O la CPU può essere impegnata da un altro processo
  - Elaborazione seriale dei task

# Multi-Programmazione MULTI-TASKING

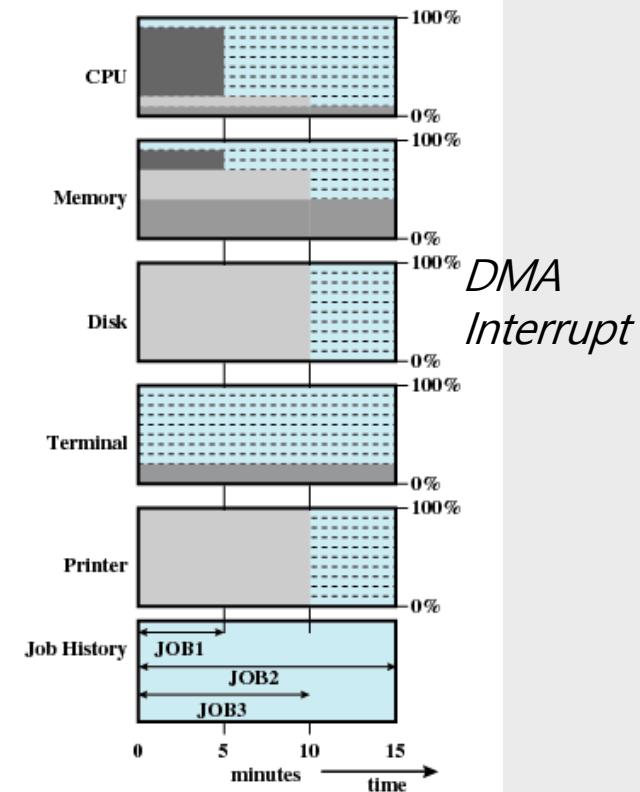


# Mono-Programmazione vs. Multi-Programmazione

	Job 1	Job 2	Job 3
Tipo	calcolo	I/O	I/O
Durata	5 min	15 min	10 min
Mem.	50 KB	100 KB	80 KB
Disco	No	No	Si
Termin.	No	Si	No
Stamp.	No	No	Si



(a) Uniprogramming



(b) Multiprogramming

Figure 2.6 Utilization Histograms

Difficoltà della multiprogrammazione:

- Gestione della memoria
- Decidere quale job mandare in esecuzione (schedulazione)

# Processo = Job = Task

- Un programma in esecuzione
- L'anima di un programma (!?)
- Una entità assegnata ad un processore e da essa eseguita

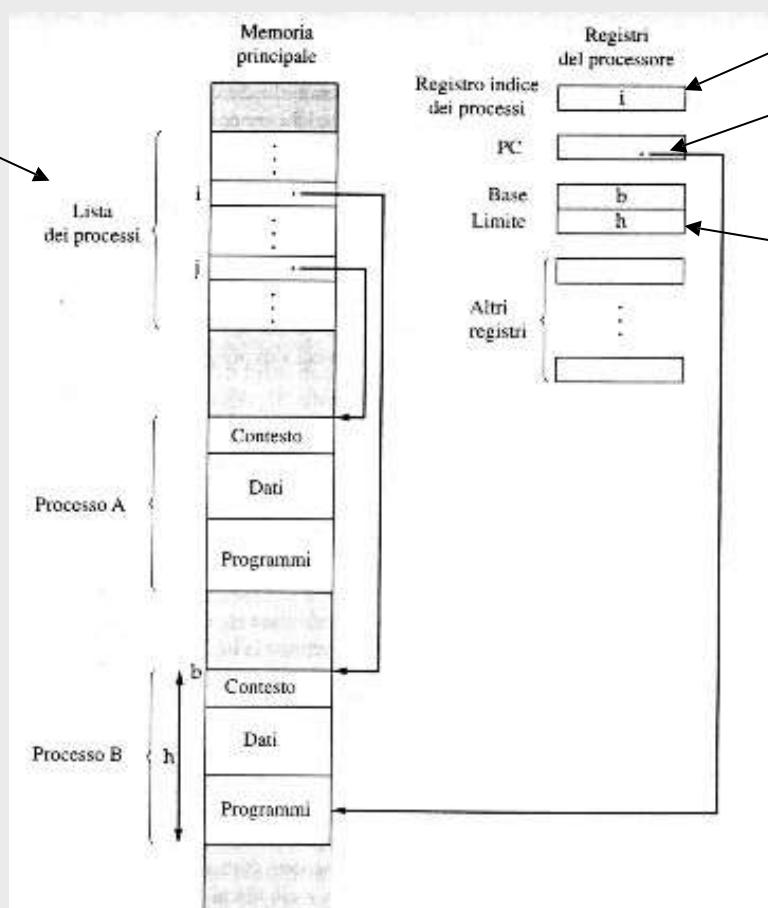
## Componenti

- Programma
  - codice eseguibile
- Dati
  - variabili
  - spazio di lavoro
  - Buffer
- Contesto di esecuzione (info necessarie al SO per gestire il processo)
  - contenuto dei registri della CPU
  - Priorità,
  - Stato di esecuzione
  - Stato di attesa su un dispositivo di I/O

# Implementazione di un processo

Gestita dal SO

```
struct  
{programma,  
dati,  
contesto}  
processo;
```



B è in esecuzione

Punta alla istruzione  
successiva del  
processo da eseguire

Protezione delle aree  
di memoria degli altri  
processi

**Context Switching**  
Passaggio da un processo  
ad un altro:  
1. Salvataggio del primo  
contesto  
2. Caricamento del  
secondo

# Gestione della Memoria

Il SO deve assolvere 5 compiti:

1. Isolamento dei processi
2. Allocazione e gestione automatica della memoria: la gerarchia delle memorie deve essere trasparente all'utente
3. Supporto alla programmazione modulare: variazione di dimensione dei programmi
4. Protezione e controllo dell'accesso: gestione di aree di memoria condivise tra i processi
5. Memorizzazione a lungo termine

Necessità soddisfatte da:

**memoria virtuale:** i programmi indirizzano la memoria con riferimenti logici ignorando gli aspetti fisici, quando un programma è in esecuzione solo una sua parte risiede effettivamente in memoria centrale

**file system:** implementa la memorizzazione a lungo termine

# Schedulazione

La politica di allocazione delle risorse deve considerare i seguenti fattori:

- **Equità:** tutti i processi

- appartenenti ad una stessa classe,
  - o con richieste simili,
  - o stesso costo,

devono avere la stessa possibilità di accesso alla risorsa

- **Tempo di risposta differenziale:** il SO discrimina tra classi che hanno bisogno di risorse diverse e di tempi diversi

- Es.: i processi con forte uso di I/O vengono schedulati per primi

- **Efficienza:** massimizzare il throughput, minimizzare il tempo di risposta

**PROCESSO**

**DESCRIZIONE,  
CONTROLLO e  
SCHEDULAZIONE dei  
PROCESSI**

**In ambienti multi-programmati**

# PROCESSO Process

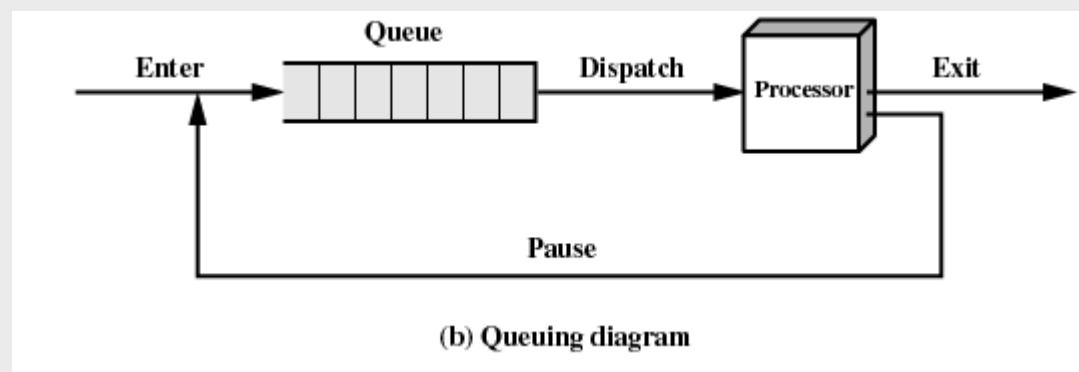
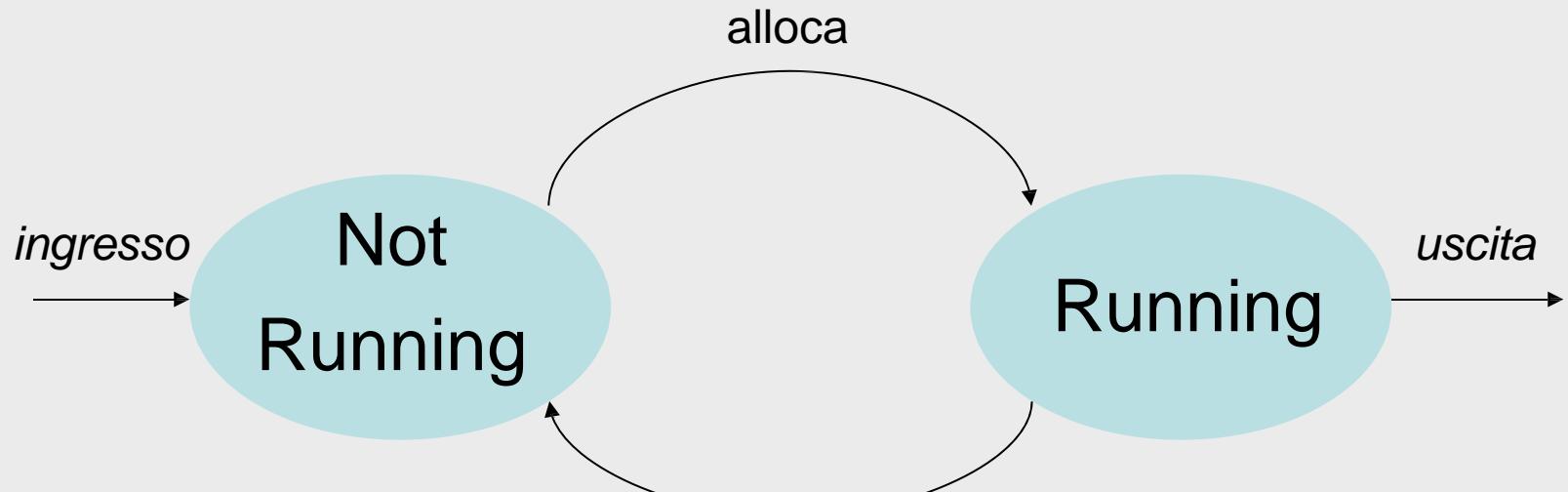
- Una entità che deve essere eseguita su una CPU
- Una attività caratterizzata dall'esecuzione di una sequenza di istruzioni, uno stato corrente e un set di istruzioni di sistema

## Componenti

- Programma
  - codice eseguibile
- Dati
  - variabili
  - spazio di lavoro
  - Buffer
- Contesto di esecuzione (info necessarie al SO per gestire il processo)
  - contenuto dei registri della CPU
  - Priorità, stato di attesa su un dispositivo di I/O

# Modello a due processi

Compito principale di un SO è il controllo dell'esecuzione dei processi



# Descrizione dei Processi

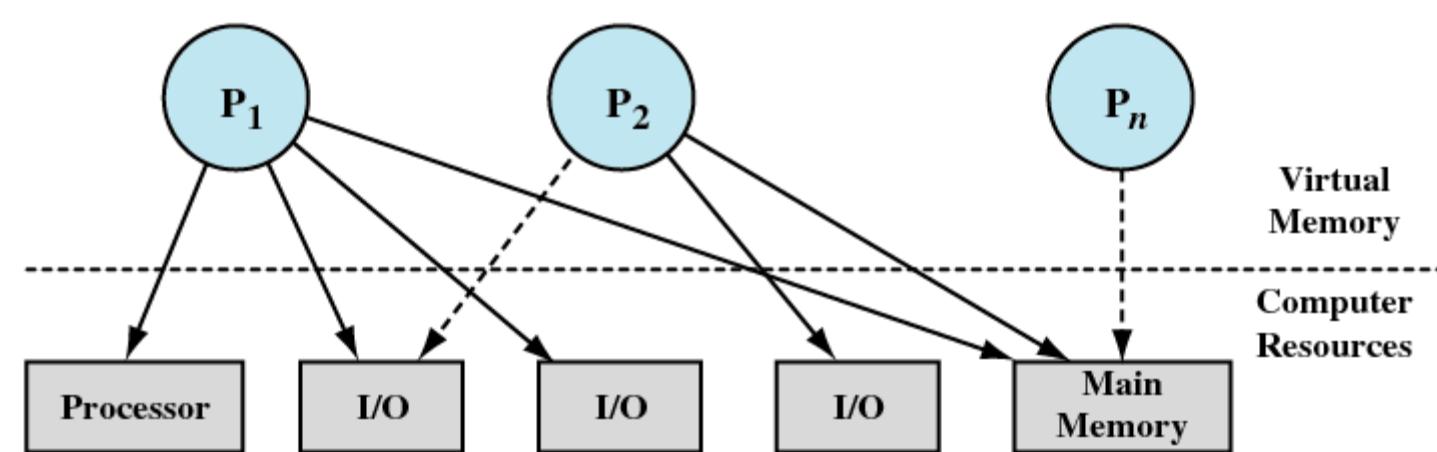


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

Il SO ha necessità di uno strumento per gestire i processi, che tenga traccia di tutte le info disponibili

# Process Control Block PCB

Descrittore di processo attraverso il quale il SO può gestirlo

## IDENTIFICATORE DEL PROCESSO

Process IDentification (PID) – Valore numerico  
(PID del processo genitore)

## INFO SULLO STATO DEL PROCESSORE

Registri Dati visibili all'utente (dipendono dall'arch. del calcolatore)

Registri di controllo e di stato

PC – indirizzo della prossima istruzione da eseguire  
Registri di stato – includono i flag per l'abilitazione degli interrupt  
Registri che contengono codici di condizione (segno, overflow, ecc.)

Puntatori allo stack

Usato per procedure e funzioni

# Process Control Block

## INFO DI CONTROLLO DEL PROCESSO

### Schedulazione e info di stato

stato del processo (running, ready, blocked, ecc.)

priorità nelle code di scheduling

info correlate alla schedulazione (tempo di attesa, tempo di esecuzione, ecc.)

evento (del quale è in attesa)

### Strutturazione dati

puntatori ad altri processi (figli/padre o per l'implementazione di code)

### Comunicazione tra Processi

flag, segnali e messaggi per la comunicazione

### Privilegi

in relazione all'uso di memoria, dispositivi, ecc.

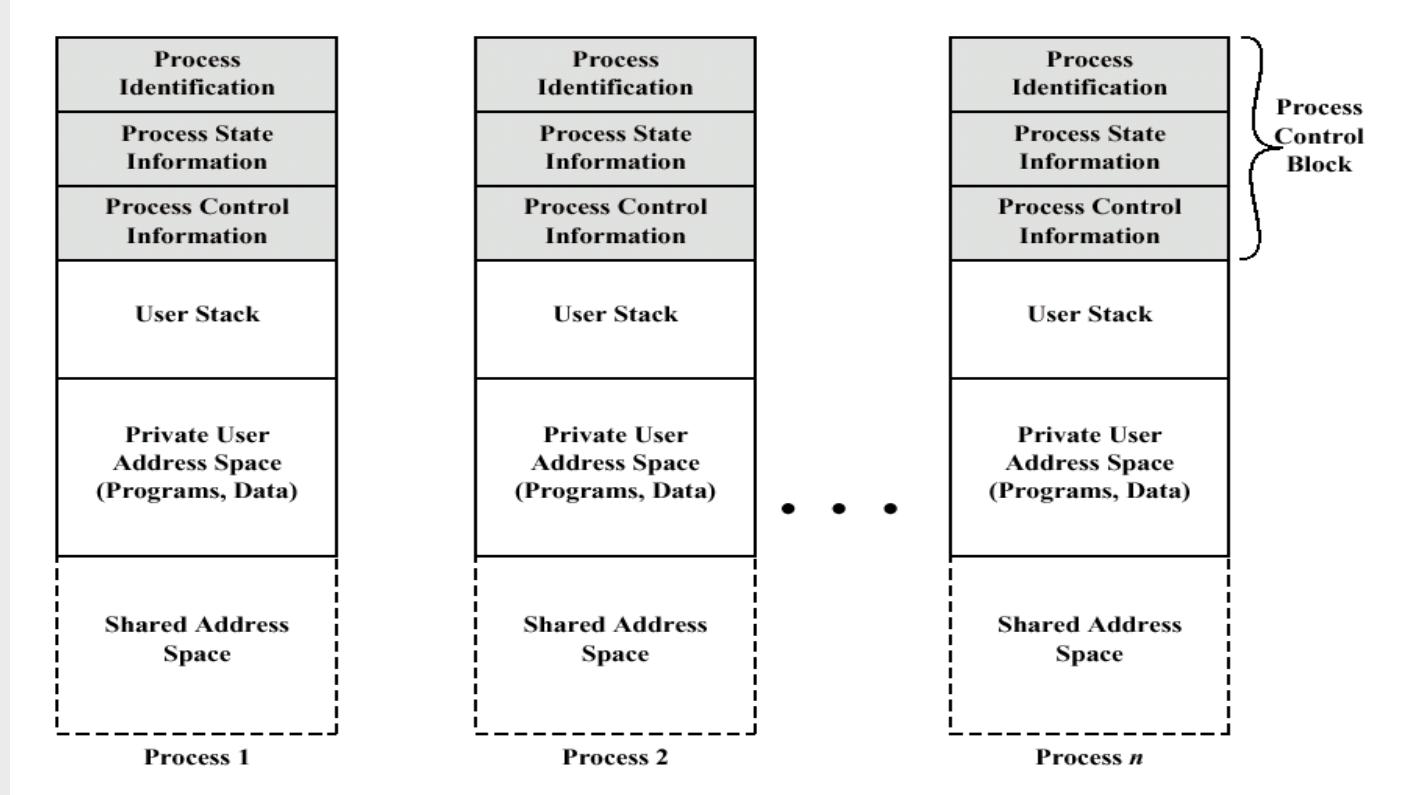
### Gestione memoria

limiti di memoria: insieme degli indirizzi accessibili (base, indice)

### Contabilizzazione delle risorse

risorse controllate dal processo (lista file aperti, lista dispositivi I/O) e loro storia

# Immagine dei processi in memoria



Nell'esempio le immagini occupano locazioni contigue di memoria, in una implementazione reale ciò può non essere vero. Dipende dalla politica di gestione della memoria

# Creazione e Terminate dei Processi

QUESTION

Eventi che portano alla **creazione** del processi

1. Richiesta da terminale (un utente accede al sistema)
2. Il SO genera un processo sulla base della richiesta di un processo utente  
Es: stampa – il processo generatore continua la sua esecuzione
3. Un processo utente genera un nuovo processo  
Processo padre - Processo figlio (generazioni)  
Es. sfruttare il parallelismo  
processo server che genera diverse istanze per gestire diverse richieste

ANSWER

Eventi che portano alla **terminazione** del processi

1. Terminazione normale (end)
2. Uscita dell'utente dalla applicazione
3. Superamento del tempo massimo
4. Memoria non disponibile
5. Violazione dei limiti di memoria
6. Fallimento di una operazione (aritmetica - I/O)
7. Terminazione del genitore
8. Richiesta del genitore

# Modello a 5 stati

Nel modello a 2 stati, lo stato “not-running” include 2 possibilità:

- Not-running
  - ready to execute
- Blocked
  - In attesa di I/O

Il Dispatcher non può semplicemente scegliere il processo da più tempo in attesa, poichè esso potrebbe essere in attesa di trasferimento I/O

Lo stato “not-running” viene suddiviso in due stati: ready e blocked

- New
- Ready
- Running
- Blocked (Waiting)
- Exit (Terminated)

# Modello a 5 stati

## Stati e Transizioni

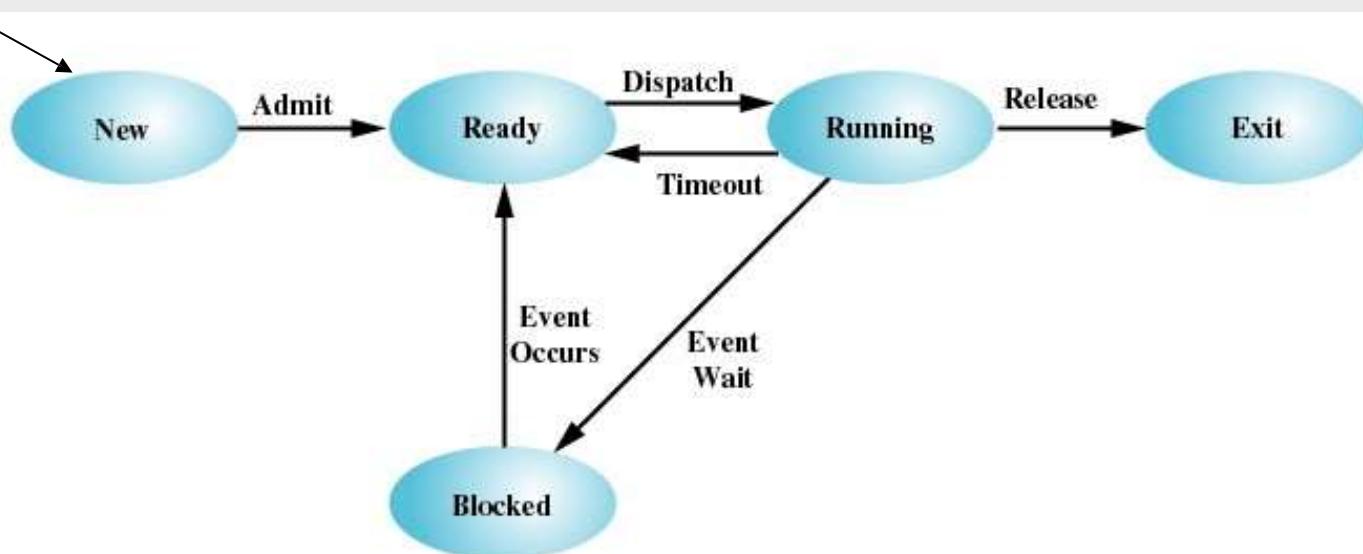


Figure 3.6 Five-State Process Model

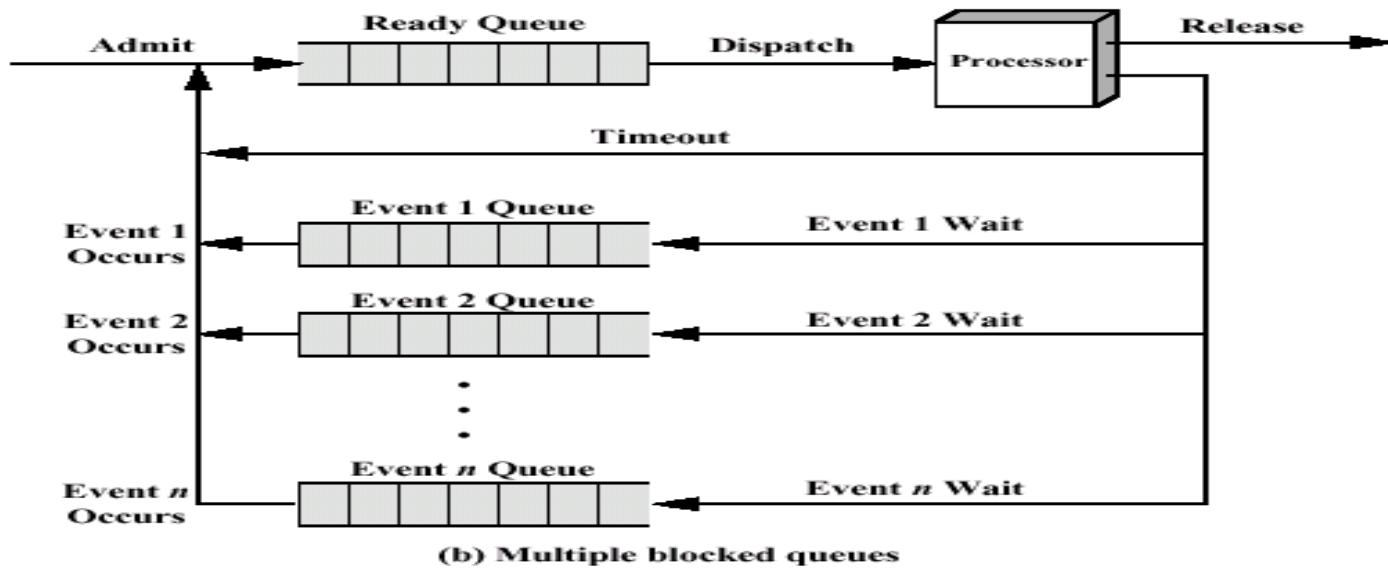
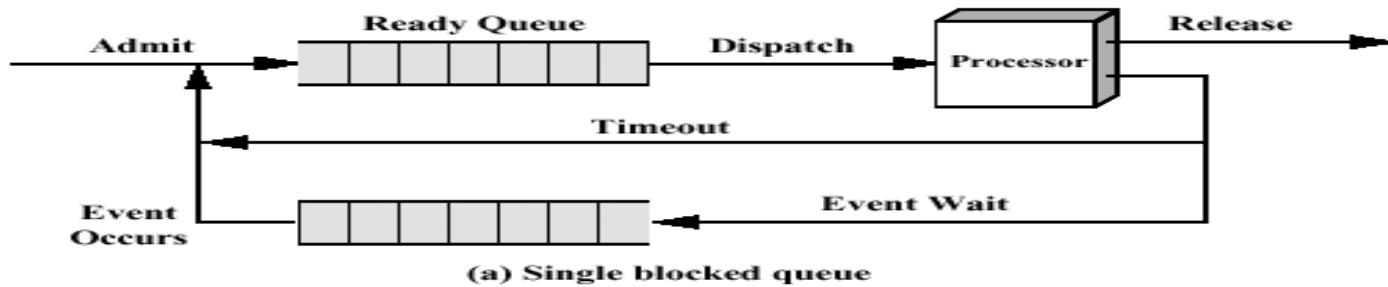
Ulteriori Transizioni:

ready → exit (genitore che termina figlio)  
blocked → exit (come sopra)

**New:** SO associa al processo il PID, alloca e costruisce le tabelle per la gestione del processo.  
**NB:** il processo non è caricato in memoria

**Exit:** rilascio delle risorse. Il SO può mantenere alcune info (es. contabilità)

# Strategie di accodamento



# Context Switch

Riguarda il passaggio della CPU ad un nuovo processo

Cause:

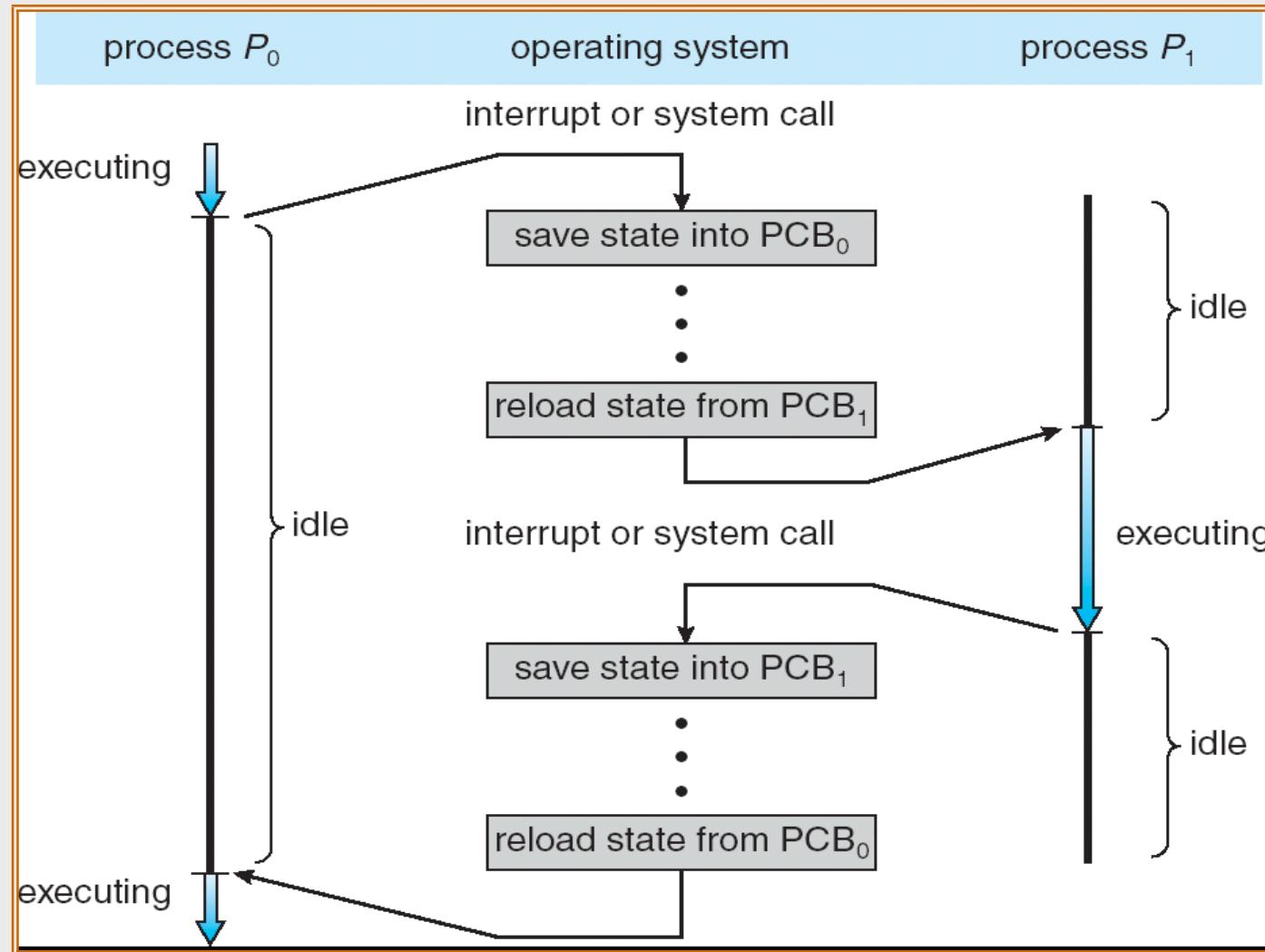
- Clock interrupt
  - Il processo ha terminato il tempo a sua disposizione (torna in ready)
- I/O interrupt
  - Una operazione di I/O è terminata, il SO sposta il processo in attesa di tale evento da blocked a ready e decide se riprendere l'esecuzione del processo precedente
- Memory fault
  - L'indirizzo di memoria generato è sul disco (memoria virtuale): deve essere portato in RAM. Il SO carica il blocco, nel frattempo il processo che ha generato la richiesta è in blocked, al termine del trasferimento andrà in ready
- Trap
  - Errore di esecuzione (il processo potrebbe andare in exit)
- Supervisor call
  - Es.: file open, il processo utente va in blocked

# Context Switch

Operazioni svolte dal SO in modalità supervisor al momento del cambio di processo in stato di running:

- Salvataggio del contesto del processo che abbandona la CPU (valori dei registri della CPU: pc, psw, reg, ecc.)
- Cambio del valore di stato nel PCB (running -> [ready, blocked, exit])
- Spostamento del PCB in nuova coda (ready o blocked) o deallocare le sue risorse (exit)
- Aggiornamento delle strutture dati gestione memoria (area dello stack)
  
- selezione di nuovo processo per lo stato running (dispatcher)
- aggiornamento del suo stato nel PCB
- ripristino del contesto
  
- Context-switch time è overhead; il sistema non svolge nessun compito utile (all'utente)
- Il tempo dipende dalla complessità del SO e dall'hardware

# Context Switch



# Gestione dei Processi

Modi di esecuzione dei Processi:

- Modo Utente: esecuzione di processi utenti
- Modo Sistema o Kernel o Controllo: esecuzione di istruzioni che hanno come scopo:
  - Gestione dei Processi
    - Creazione e terminazione
    - Schedulazione
    - Cambio di contesto
    - Sincronizzazione
    - PCB
  - Gestione della Memoria
    - Allocazione
    - Trasferimento disco/RAM e vmersa
    - Gestione paginazione, segmentazione
  - Gestione I/O
    - Gestione buffer
    - Allocazione canali I/O
  - Supporto
    - Gestione Interruzioni
    - Contabilità

# Creazione dei Processi

- 1) Assegnare al processo un PID unico;  
aggiungere una entry level alla tabella dei processi
- 2) Allocare lo spazio per il processo e per tutti gli elementi della sua immagine  
(PCB, User Stack, Area di memoria dati e istruzioni, aree condivise). Info valutate per difetto, indicazione dell'user, dal processo padre.

Es. Address space

- Child duplicate of parent
- Child has a program loaded into it

Possibilità di esecuzione:

- Processo padre e figlio sono concorrenti
- Il padre attende la terminazione del figlio

- 3) Inizializzazione del PCB

stato del processore =0

PC =prossima istruzione

puntatori allo stack...

stato = ready (ready-suspended)

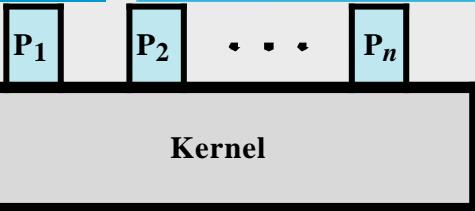
- 4) Inserimento nella coda ready

- 5) Estende le strutture al fine della fatturazione o delle statistiche.

# Terminazione dei Processi

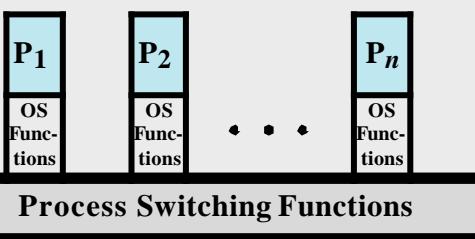
- Il processo termina poichè ha eseguito l'ultima istruzione, chiede al SO di cancellarlo (exit)
  - Un processo figlio può riportare dati al padre (in wait)
  - Le risorse del processo sono deallocate
- Un processo padre può uccidere unprocesso figlio (abort)
  - Il figlio ha ecceduto l'uso di risorse
  - Il task del figlio non è più richiesto
  - Se un genitore termina, in base all'applicazione alcuni SO non consentono che i figli rimangano in esecuzione: cascading termination
    - Es: terminazione di firefox causa la terminazione di tutte le schede ad esso associate

# Modalità di esecuzione del SO



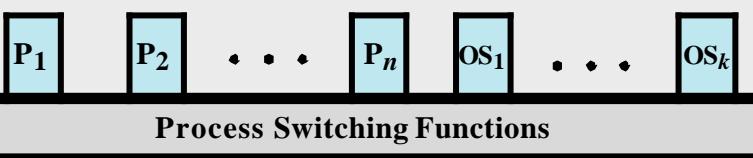
(a) Separate kernel

- Kernel separato dai processi (obsoleto)
  - Occupa una regione di memoria specifica
  - Ha il proprio stack
  - Eseguito come entità separata



(b) OS functions execute within user processes

- All'interno dei processi utente sono presenti anche programmi, dati e stack del kernel
  - Quando vi è una chiamata a SO, il Kernel cambia il modo di esecuzione (salvataggio contesto utente, modo utente → kernel ) NB: non c'è cambio di processo (non interviene lo schedulatore) poichè la funzione del kernel è nel processo utente



(c) OS functions execute as separate processes

- SO basato su processi
  - Processi di sistema
  - Pro: interfacce pulite e semplici tra i moduli
  - Utile in sistemi multi processore

# Evoluzione del modello a 5 stati

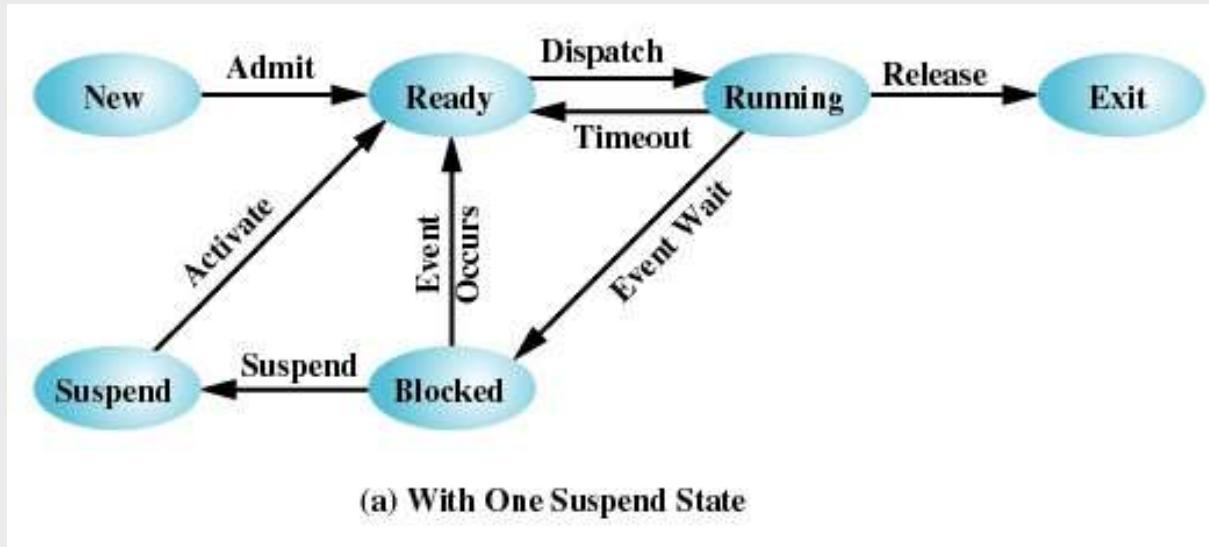
- Nonostante la memoria Virtuale, un programma per essere eseguito deve essere in RAM
- Con elevata probabilità tutti i processi in memoria restano in attesa di operazioni di I/O



- Processore inattivo (molto più veloce di I/O)
- Soluzioni
  - espandere la memoria
    - Costo
    - Poco efficiente (programmi sempre più grandi)
  - effettuare lo swapping: spostare un processo dalla RAM al disco
    - introduzione nuovo stato –stato suspend
    - NB: lo swapping è una ulteriore op. di I/O, ma in generale è la più rapida tra le op. di I/O

# Evoluzione del modello a 5 stati

## Modello a 6 stati



Swap out: scaricamento del processo sul disco blocked ->suspended  
Swap in..

Tra processi in new e in suspended quale scelgo da portare in Ready??  
Ricordiamoci che il Suspended è in attesa di un evento..  
Se l'evento si verifica quel processo potrebbe andare in ready

*suspended*  $\begin{cases} \rightarrow \text{Ready-suspended} \\ \rightarrow \text{Blocked-suspended} \end{cases}$

# Evoluzione del modello a 5 stati Modello a 7 stati

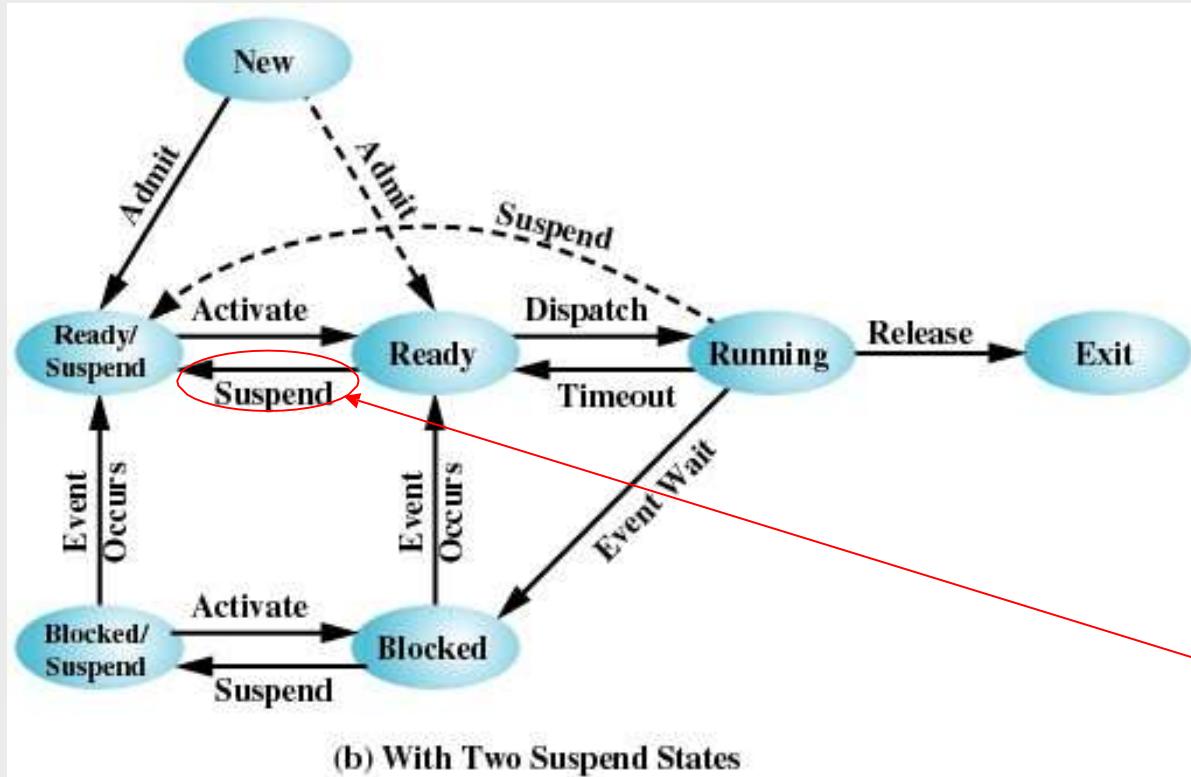


Figure 3.9 Process State Transition Diagram with Suspend States

## Osservazione

Memoria virtuale: un processo può trovarsi solo parzialmente Ram, quando si fa riferimento a un indirizzo su disco, questo viene caricato  
⇒Inutilità del suspended

MA: immaginiamo il caso di molti processi tutti parzialmente presenti in RAM...

*Necessità di maggiore memoria per allocare un processo più grande o a maggiore priorità*

*tutti -> exit*

# Scheduling

Con scheduling si intende un insieme di tecniche e di meccanismi interni del sistema operativo che amministrano l'ordine in cui il lavoro viene svolto

Obiettivo primario dello scheduling è l'ottimizzazione delle prestazioni del sistema.

Il sistema operativo può prevedere fino a 3 tipi di scheduler:

- scheduler di lungo termine (SLT)
- scheduler di medio termine (SMT)
- scheduler di breve termine (SBT)

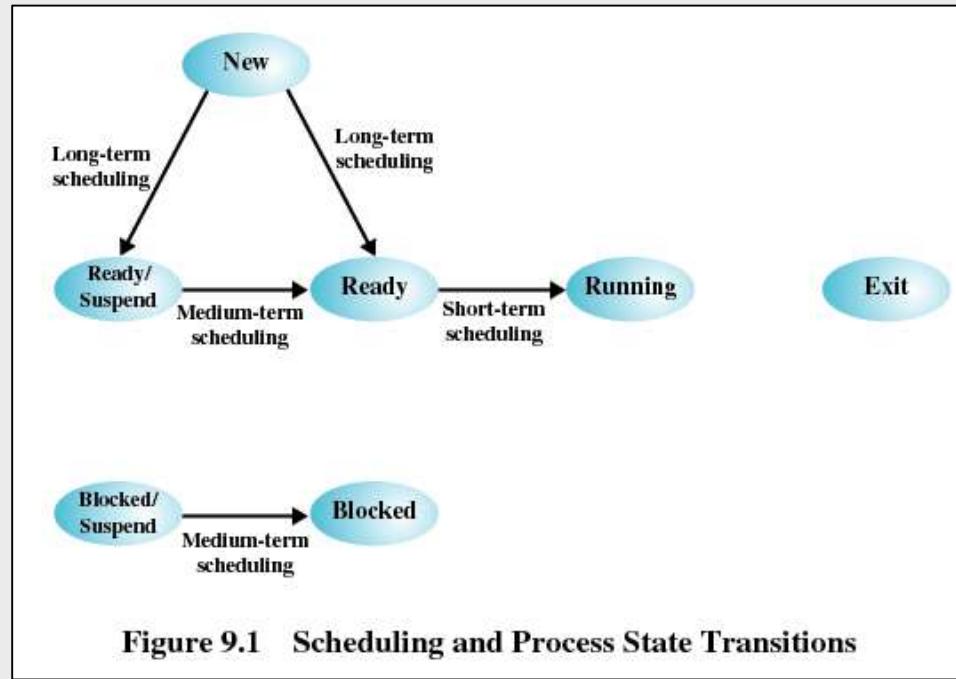


Figure 9.1 Scheduling and Process State Transitions

# Scheduling

## Scheduling a lungo termine:

- Determina quali programmi sono ammessi nel sistema per essere processati (new -> ready o ready suspended)
- Controlla il grado di multiprogrammazione (new -> ready)
  - Più processi = minor tempo percentuale di esecuzione per ognuno di essi
- Stime effettuate dal programmatore (o dal sistema) forniscono informazioni sulle risorse necessarie alla esecuzione, come le dimensioni della memoria, il tempo di esecuzione totale, ecc.
- Il lavoro dello scheduler di lungo termine si basa quindi sulla stima del comportamento globale dei job.

## Le strategie principali:

1. fornire alla coda dei processi pronti (e quindi allo scheduler di breve termine) gruppi di processi che siano bilanciati tra loro nello sfruttamento della CPU e dell'I/O;
2. aumentare il numero di processi provenienti dalla coda batch, quando il carico della CPU diminuisce;
3. diminuire (fino anche a bloccare) i lavori provenienti dalla coda batch, quando il carico aumenta e/o i tempi di risposta del sistema diminuiscono.

La frequenza di chiamata dell'SLT è bassa e consente di implementare strategie anche complesse di selezione dei lavori e di dimensionamento del carico dei processi da inviare alla coda pronti (ready)

# Scheduling

Scheduling a medio termine:

- ready suspended ->ready
- Blocked suspended ->blocked
- Parte della funzione di swapping
- Basato sulla necessità di gestire il livello di multi-programmazione

*La presenza di molti processi sospesi in memoria, riduce la disponibilità per nuovi processi pronti. In questo caso lo **scheduler di breve termine** è obbligato a scegliere tra i pochi processi pronti...*

- *utilizza le informazioni del **Descrittore di Processo** (PCB) per stabilire la richiesta di memoria del processo;*
- *tenta di allocare spazio in memoria centrale;*
- *riposiziona il processo in memoria nella coda dei pronti.*

*Viene attivato:*

- quando si rende disponibile lo spazio in memoria;
- quando l'arrivo di processi pronti scende al di sotto di una soglia specificata.

# Scheduling

Scheduling a breve termine (dispatcher):

- Ready -> Run
- Eseguito molto frequentemente
- Invocato quando si verifica un evento:
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

*La sua principale strategia è orientata alla massimizzazione delle prestazioni del sistema secondo un specifico insieme di obiettivi.*

# Scheduling della CPU (scheduling a breve termine)

Esecuzione di un processo:

1. ciclo di elaborazione (CPU)
2. attesa di completamento di I/O

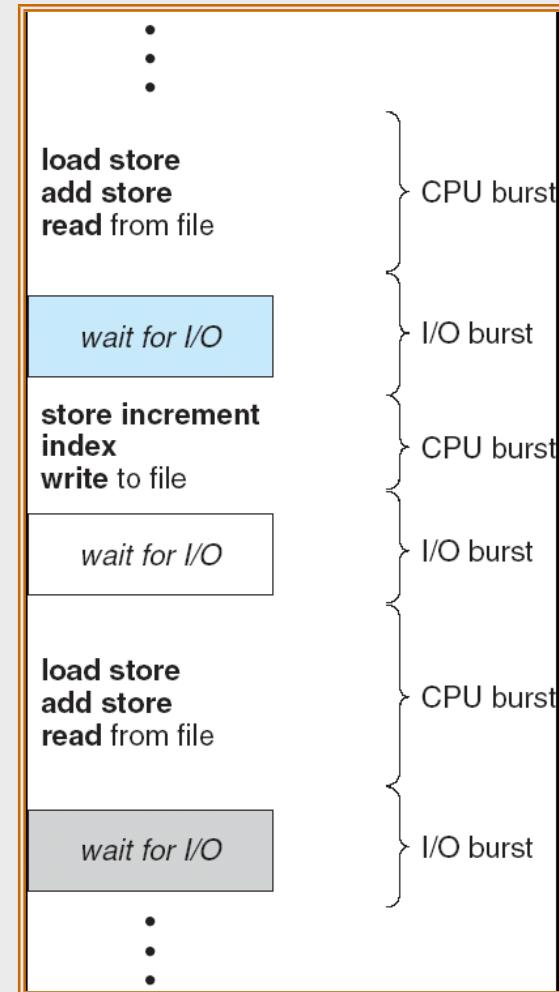
Lo scheduling della CPU riguarda la distribuzione delle sequenze di elaborazione della CPU

Processo I/O bound:

molte sequenze di operazioni di CPU di breve durata

Processo CPU bound:

poche sequenze di operazioni di CPU di lunga durata



# Scheduling della CPU: valutazione Delle prestazioni

Approcci:

User-oriented

Turnaround time: tempo intercorso tra la sottomissione di un processo e la sua terminazione.

Tempo di risposta: per un processo interattivo è il tempo trascorso tra la sottomissione di richiesta e il ritorno del primo output. Il processo può continuare nel suo ciclo di esecuzione mentre produce output.

Deadlines: quando la deadline di un processo è specificata, lo scheduling deve fare in modo di completare (o avviare) il processo entro la deadline

System-oriented

Throughput: numero di processi terminati per unità di tempo

Utilizzo del processore: % del tempo in cui la CPU è impegnata

- per eseguire programmi degli utenti
- per eseguire moduli del sistema operativo

Evitare la starvation

Utilizzare tutte le risorse (dispositivi di I/O, CPU, ecc.)

**Criteri di ottimizzazione:**

Max utilizzazione della CPU

Max throughput

Min turnaround time

Min tempo di risposta

# Scheduling della CPU: valutazione Delle prestazioni

## Tempo di ricircolo (Turnaround time)

Tempo trascorso l'avvio (immissione nel sistema) e la terminazione di un processo

$$T_{loading} + T_{ready} + T_{CPU} + T_{I/O}$$

## Tempo di attesa

- misura il tempo che un processo trascorre in attesa delle risorse a causa di conflitti con altri processi.
- E' la penalità che si paga per condividere risorse ed è espressa come: tempo di ricircolo - tempo di esecuzione.
- Valuta in sostanza la sorgente di inefficienza

## Tempo di risposta

- nei sistemi time-sharing:

misura il tempo che trascorre dal momento in cui viene introdotto l'ultimo carattere al terminale all'istante in cui viene restituita la prima risposta

- nei sistemi real-time:

misura il tempo che trascorre dal momento in cui viene segnalato un evento esterno all'istante in cui viene eseguita la prima istruzione della relativa routine di gestione

# Priorità – Event Driven

A ogni processo viene assegnato un livello di **priorità**. Lo scheduler sceglie sempre il processo pronto con priorità maggiore

La priorità può essere assegnata dall'utente o dal sistema e può essere di tipo statico o dinamico

la priorità dinamica varia in funzione:

- del valore iniziale
- delle caratteristiche del processo
- della richiesta di risorse
- del comportamento durante l'esecuzione

*Modello **Event Driven** applicato nei sistemi dove il tempo di risposta, soprattutto ad eventi esterni, è critico.*

Caratteristiche:

- *il sistemista può influire sull'ordine in cui uno scheduler serve gli eventi esterni modificando le priorità assegnate ai processi*
- *le prestazioni sono dipendenti da una accurata pianificazione nell'assegnazione delle priorità*
- *non è in grado di garantire il completamento di un processo in un intervallo di tempo finito dalla sua creazione.*

# Priorità – Event Driven

- Un numero di priorità (intero) è associato a ciascun processo
- La CPU è allocata al processo con la più alta priorità (di solito la più alta priorità corrisponde all'intero più piccolo)
  - Preemptive: requisizione della CPU se la priorità del nuovo processo in ready è maggiore
  - Non-preemptive
- Le priorità possono essere definite:
  - internamente al SO (utilizzando grandezze misurabili): uso di memoria, file aperti, rapporto tra picchi medi di I/O e di CPU
  - Esternamente al SO rilevanza del processo, criticità
- Problemi: Starvation (processi a bassa priorità non vengono mai eseguiti)
- Soluzione: Aging, al passare del tempo in stato di ready la priorità del processo aumenta

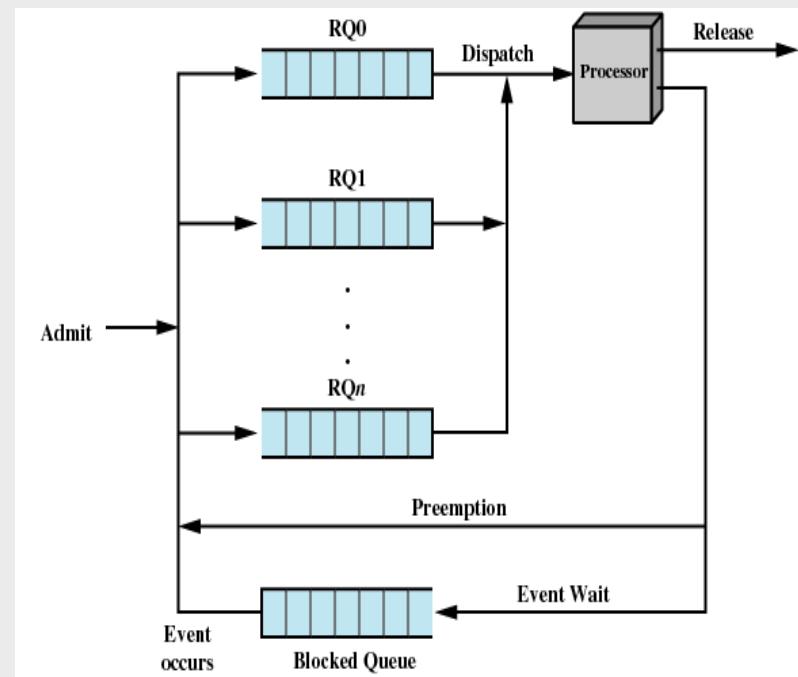


Figure 9.4 Priority Queuing

# Scheduling della CPU

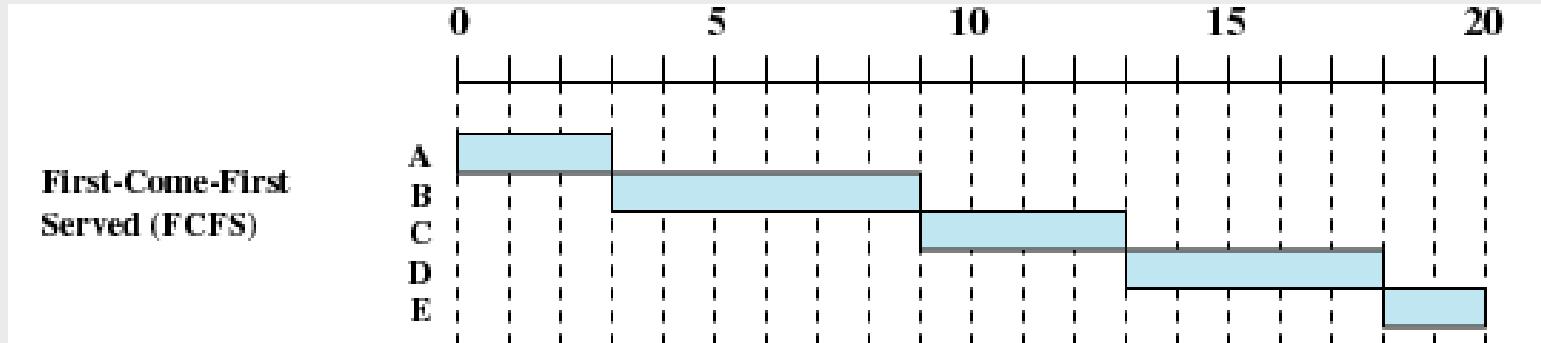
## DECISION MODE:

- Non-Preemptive
  - Un processo nello stato di running abbandonerà tale stato solo se termina l'esecuzione o se si blocca per una operazione di I/O
- Preemptive
  - Il processo attualmente nello stato di running può essere interrotto e spostato nello stato di ready dal SO
  - PRO: Nessun processo può monopolizzare il processo
  - PROBLEMI: processi che condividono dati -> meccanismi di sincronizzazione

## DISPATCHER:

- Modulo del SO che passa il controllo della CPU al processo selezionato dallo scheduler a breve termine attraverso:
  - switch del contesto
  - switch del modo utente/supervisore
  - Salto alla locazione opportuna del programma utente (contenuta nel PCB) per farlo ripartire
- *Dispatch latency* – tempo che il dispatcher usa per fermare un processo e far partire un altro.

# First Come First Served - FCFS



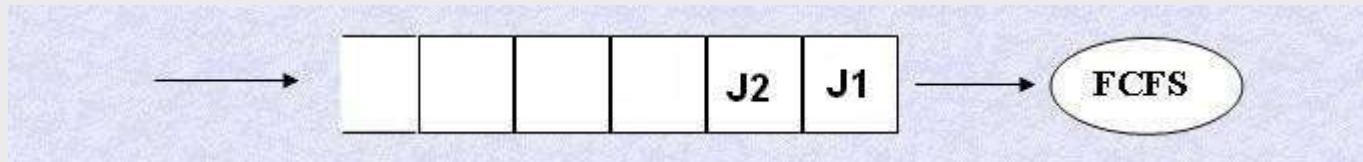
- Ogni processo entra nella coda di ready
- Quando un processo abbandona lo stato di running si seleziona il processo che da più tempo è nello stato di ready
- Un processo I/O bound o che richiede poco tempo di esecuzione potrebbe attendere molto tempo prima che gli venga assegnata la CPU
- Favorisce i processi CPU-bound
- Tempo medio di attesa in coda elevato
- Effetto convoglio: tutti i processi in coda attendono che un processo CPU-bound termini
- Senza prelazione:
  - *basso sfruttamento dei componenti*
  - *basso lavoro utile del sistema*

# First Come First Served - FCFS

Le prestazioni dipendono dall'ordine di arrivo dei jobs

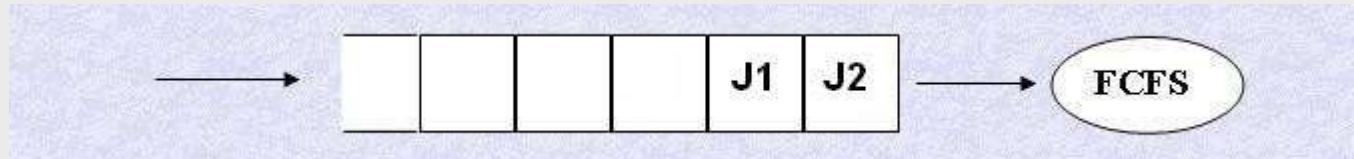
**esempio :** Siano  $J_1$  e  $J_2$  due job con tempi di esecuzione totali rispettivamente pari a  $t_1=20$  e  $t_2=2$  unità di tempo.

**I CASO**



Tempo di riclico di  $J_1=20$ ; Tempo di riclico di  $J_2=22$ ; Tempo medio di riclico=21;  
Tempo di attesa di  $J_1=0$ ; Tempo di attesa  $J_2=20$ ; Tempo medio di attesa=10;

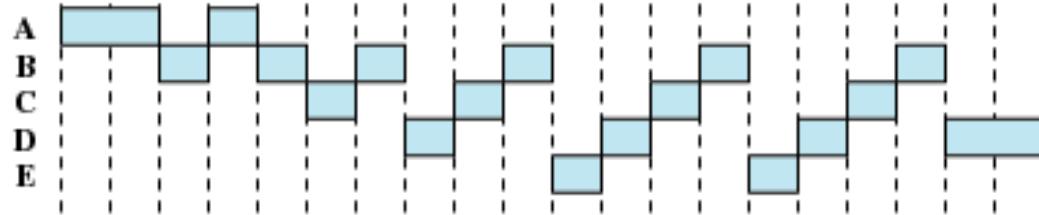
**II CASO**



Tempo di riclico di  $J_2=2$ ; Tempo di riclico di  $J_1=22$ ; Tempo medio di riclico=12;  
Tempo di attesa di  $J_2=0$ ; Tempo di attesa  $J_1=2$ ; Tempo medio di attesa=1;

# Round Robin – time slice

Round-Robin  
(RR),  $q = 1$



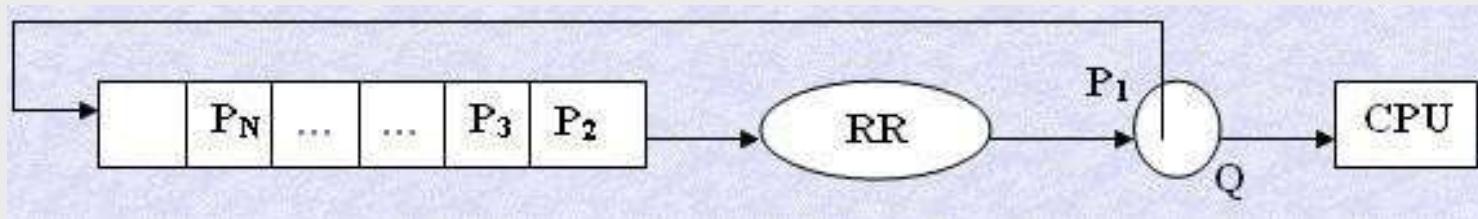
- Preemption basata sul clock (clock interrupt)
- Ogni processo utilizza il processore per un dato intervallo di tempo (time slice). Valori tipici: 10-100msec
- Al verificarsi dell'interrupt il processo in esecuzione viene portato nella coda di ready (gestita FIFO)
- Con  $n$  processi in ready e un time quantum  $q$ , ogni processo ottiene  $1/n$  del tempo di CPU in frazioni di tempo al più pari a  $q$ . Tempo massimo di attesa in ready:  $(n-1)q$ .
- Prestazioni dipendenti dal quanto di tempo:
  - $q$  grande  $\Rightarrow$  FCFS
  - $q$  piccolo  $\Rightarrow$  incrementa il numero di context switch

# Round Robin – time slice

La schedulazione round-robin fornisce una buona condivisione delle risorse del sistema:

- i processi più brevi possono completare l'operazione in un q (buon tempo di risposta)
- i processi più lunghi sono forzati a passare più volte per la coda dei processi pronti (tempo proporzionale alle loro richieste di risorse)
- per i processi interattivi lunghi, se l' esecuzione tra due fasi interattive riesce a completarsi in un q, il tempo di risposta è buono

La realizzazione di uno scheduler RR richiede il supporto di un Timer che invia un'interruzione alla scadenza di ogni q, forzando lo scheduler a sostituire il processo in esecuzione. Il timer viene riazzerato se un processo cede il controllo al Sistema Operativo prima della scadenza del suo q.



# Highest Response Ratio Next

Siano:

w – tempo speso in coda di ready (attesa della disponibilità del processore)  
s – tempo di servizio previsto

si definisce il Response Ratio come:

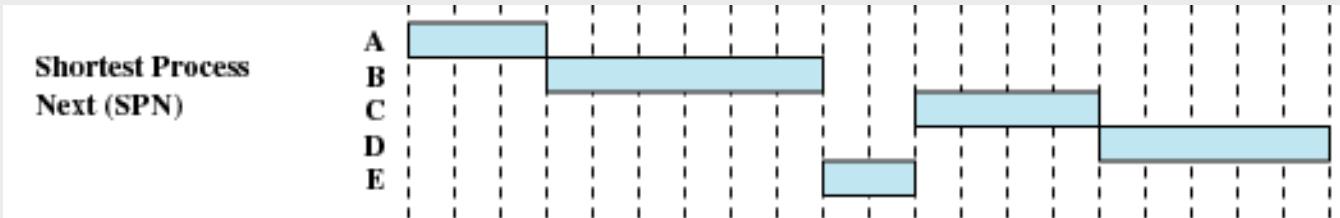
$$RR = \frac{w + s}{s}$$

Si sceglie il processo con il più alto valore di RR

Osservazioni:

- quando un processo entra in coda per la prima volta,  $RR=1$
- tiene considerazione l'età del processo

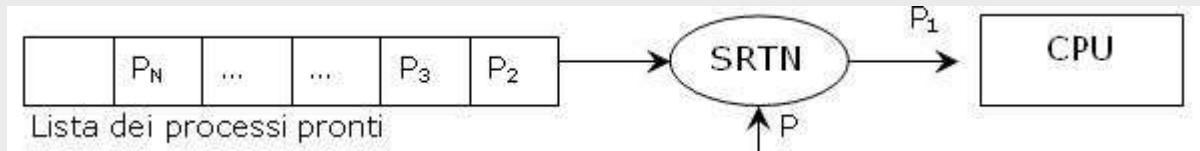
# Shortest Process Next - SPN



- Il processo scelto dalla coda di ready è quello con il più breve tempo di esecuzione stimato: più breve sequenza di operazioni svolte dal processore
- Possibilità di starvation per processi fortemente CPU bound
- SPN è ottimale nel senso che fornisce il tempo medio di attesa minimo, per un dato set di processi.
- Utilizzato nello scheduling a lungo termine
- Difficile stimare la durata della prossima sequenza di CPU (oltre che oneroso)
- Due possibili schemi:
  - nonpreemptive;
  - preemptive – se arriva nuovo processo con una sequenza di CPU minore del tempo necessario per la conclusione della sequenza di CPU del processo attualmente in esecuzione, si ha il prerilascio della CPU a favore del processo appena arrivato. Questo schema è anche noto come Shortest-Remaining-Time-First (SRTF)

# Shortest Remaining Time Next

## SPN Con Prerilascio



P<sub>i</sub> i-esimo processo con i=1,...,N;

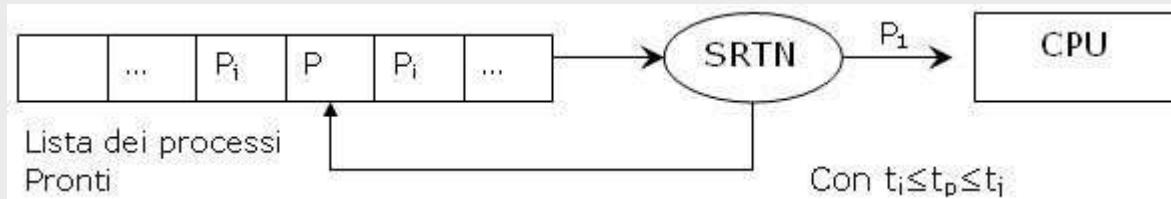
t<sub>i</sub> tempo di esecuzione dell'i-esimo processo;

Per ogni i,j =1,...,N i≤j: t<sub>i</sub> ≤ t<sub>j</sub> ;

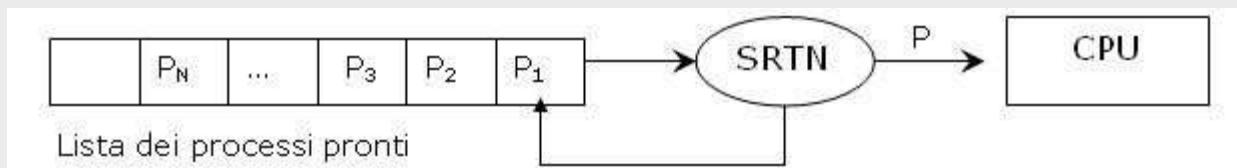
t<sub>p</sub> tempo di esecuzione del processo P;

t<sub>1r</sub> tempo di esecuzione residuo del processo P<sub>1</sub>;

I CASO:  $t_{1r} \leq t_p$

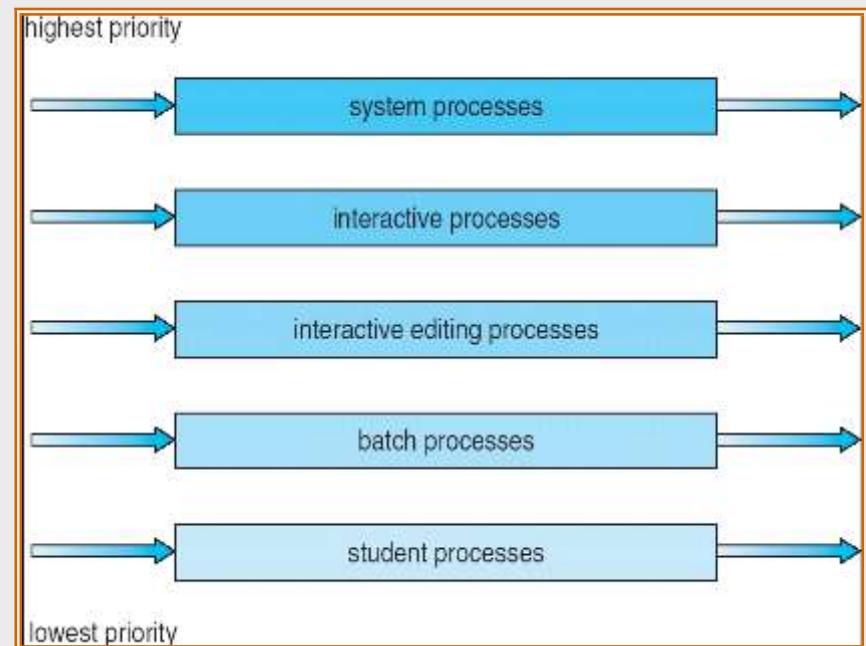


II CASO:  $t_{1r} > t_p$



# Schedulazione a code multiple

- La coda di ready è suddivisa in sotto-code:
  - foreground (interactive)
  - background (batch)
- Ogni coda ha un proprio algoritmo di schedulazione
  - Es.: foreground – RR, background – FCFS
- Lo Scheduling deve essere effettuato tra le code:
  - Scheduling per priorità fissa e con prelazione (i.e., serve all from foreground then from background). Possibilità di starvation.
  - Time slice – ad ogni coda è associato un certo ammontare di tempo di CPU; i.e., 80% to foreground in RR, 20% to background in FCFS

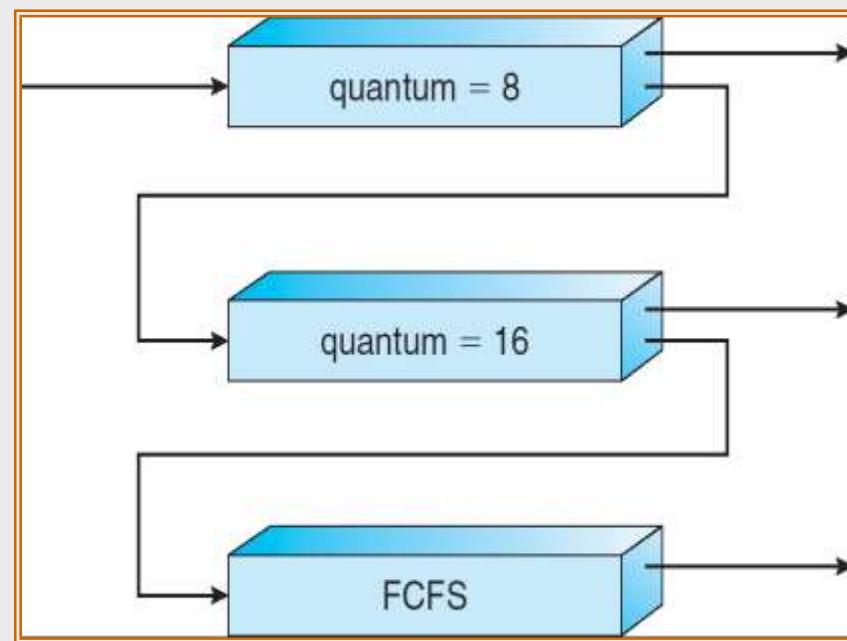


# Schedulazione a code multiple con feedback

- Un processo può essere spostato da una coda all'altra (implementazione dell'aging)
- Code Multilevel-Feedback definite dai seguenti parametri:
  - Numero di code
  - Algoritmo di scheduling per ogni coda
  - Metodi usati per l'up-grading e il down-grading di ogni processo

Esempio:

- Tre code:
  - $Q_0$  – RR con time quantum 8 milliseconds
  - $Q_1$  – RR con time quantum 16 milliseconds
  - $Q_2$  – FCFS
- Scheduling
  - Un nuovo processo entra nella coda  $Q_0$  (FCFS).
  - Quando ottiene la CPU, la impegna per 8 ms. Se non termina entro gli 8 ms è spostato in  $Q_1$ .
  - Il processo in  $Q_1$  viene nuovamente servito con politica FCFS e riceve la CPU per ulteriori 16 ms.
  - Se ancora non termina viene spostato in  $Q_2$ .



# Ricapitolando

- First come first served: Seleziona il processo che ha il tempo di attesa più lungo.
- Round robin: Usa il time-slicing per limitare tutti i processi in fase di esecuzione ad un piccolo periodo d'uso del processore e ruota su tutti i processi Ready.
- Shortest process next: Seleziona il processo con il minore tempo di uso del processore previsto e non prerilascia il processo.
- Shortest remaining time: Seleziona il processo con il minore tempo di uso del processore previsto; un processo può essere prerilasciato quando un altro processo diventa Ready.
- Highest response ratio next: Basa la decisione di scheduling su una stima del tempo di turnaround normalizzato.
- Feedback: Crea un insieme di code per lo scheduling, ed alloca i processi in tali code a seconda della loro storia di esecuzione, o su altri criteri.

# Windows

Dispatcher: porzione del kernel che si occupa dello scheduling

Un thread viene eseguito fino a:

1. Terminazione
2. Prelazione da parte di un thread a priorità più alta
3. Esaurimento del quanto di tempo
4. Chiamata bloccante

32 livelli di priorità divisi in due macro-classi:

variable: thread con priorità da 1 a 15

real time: ...da 16 a 31

priorità 0: thread per la gestione della memoria

Una coda per ogni livello di priorità.

Se non esiste nessun thread in ready  
viene eseguito un thread di idle

La priorità di un thread dipende dalla  
priorità della classe a cui appartiene e  
dalla priorità relativa del thread nella classe

*Valori di ciascuna classe*

	real-time	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

*Priorità relativa del  
thread nella classe*

# Windows

- Quando il time slice termina il thread riceve una priorità più bassa tornando in ready
- L'abbassamento della priorità non porta mai la priorità sotto il livello minimo per una specifica classe
- Se un thread va in blocked la sua priorità sarà aumentata in dipendenza del tipo di evento di attesa (thread che attende dati da tastiera riceverà priorità maggiore rispetto a quello che attende dati da HD)
- *La strategia mira a fornire tempi di risposta rapidi per thread interattivi (interfacce basate su mouse,...)*
- Tipi di processi interattivi:
  - in primo piano (foreground): correntemente selezionati sullo schermo
  - in secondo piano (background): non attualmente selezionati
  - Quando un processo passa da background a foreground la sua priorità viene aumentata di un fattore 3

# Linux Scheduling

- Due algoritmi:
  - time-sharing con diritto di prelazione ad equità
  - real-time: le priorità assolute sono più importanti dell'equità
- Time-sharing
  - Partizionamento del tempo basato sui crediti: il processo con più crediti viene schedulato
  - Ad ogni interruzione dovuta allo scadere del tempo i crediti sono decrementati di una unità
  - Quando credit=0, il processo viene sospeso e si sceglie un altro processo
  - Periodicamente vengono riassegnati i crediti per tutti i processi (utente e di sistema) con la regola:
    - Crediti=crediti/2 + priorità
    - In questo modo i processi blocked acquisiscono grande priorità
- Real-time
  - Posix.1b compliant – due classi
    - FCFS (senza prelazione) and RR (con prelazione)
    - Viene sempre eseguito il processo con più alta priorità
  - Soft real time: nessuna garanzia sul completamento dei processi entro le loro deadlines (in linux non si può sospendere un processo kernel per dare spazio ad uno utente)

# Schedulazione MULTIPROCESSORE

## Classificazione dei sistemi multi-processore

- Processori debolmente accoppiati o distribuiti, o cluster
  - Ogni processore ha la propria memoria e i propri canali di I/O
- Processori specializzati per funzioni
  - Esempio: I/O processor
  - Controllati da un processore master a cui forniscono servizi
- Processori strettamente accoppiati
  - I processori condividono la memoria centrale
  - Sono sotto il controllo del SO

## Granularità di sincronizzazione tra i processori

- Parallelismo indipendente,:
  - Non c'è sincronizzazione tra i processi (sistemi time-sharing)
  - Da un punto di vista di scheduling è indifferente avere uno o più processori
- Parallelismo a grana grossa e molto grossa:
  - Bassa sincronizzazione tra i processi
  - Può essere immaginato pari al caso di un processore multiprogrammato
- Parallelismo a grana media:
  - Una applicazione è un insieme di threads che interagiscono frequentemente
  - Lo scheduling può influenzare le prestazioni
- Parallelismo a grana fine:
  - Per applicazioni ad elevato parallelismo / Sperimentale...

# Assegnazione dei processi ai processori

- Se i processori sono tutti uguali (SMP) possono essere trattati come un pool di risorse e i processi possono essere eseguiti su ogni processore
  - Assegnazione permanente (statica):
    - Un processo viene eseguito sempre su un specifico processore
    - Coda a breve termine dedicata per ogni processore
    - Poco overhead nella gestione dello scheduling
    - Un processore può rimanere inutilizzato a seguito di una coda vuota mentre un altro ha un carico arretrato
  - Assegnazione dinamica:
    - Coda globale
    - Non c'è svantaggio rispetto al caso precedente se i PCB sono in RAM condivisa

# Assegnazione dei processi ai processori

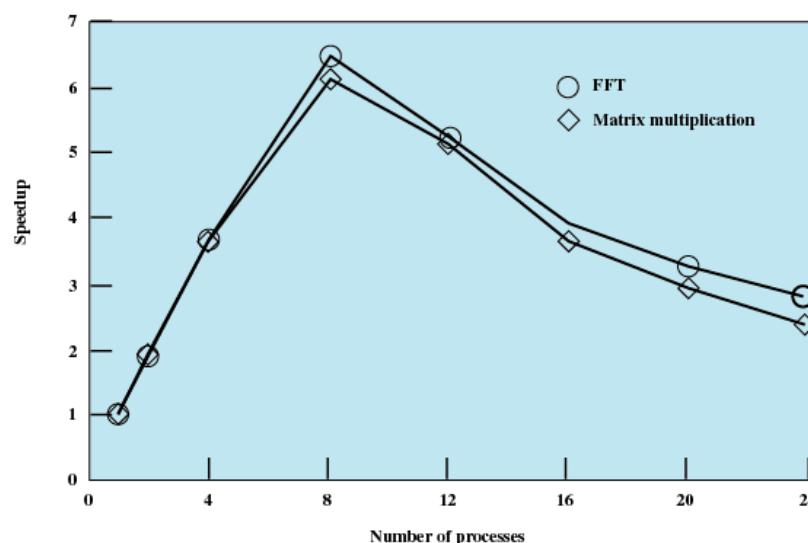
- Architettura Master/slave
  - Kernel functions sempre eseguite su un particolare processore
  - Master responsabile per lo scheduling
  - Slave inviano richieste di servizio al master
  - Vantaggi:
    - Solo il master ha il controllo della memoria e dei dispositivi di I/O (gestione conflitti semplificata)
  - Svantaggi
    - Un fallimento del master determina il fallimento dell'intero sistema
    - Master può diventare il collo di bottiglia delle prestazioni
- Architettura Peer
  - OS eseguito su ogni processore
  - Ogni processore fa fronte al proprio scheduling
  - Difficoltà per l'OS:
    - Due processori non devono scegliere lo stesso processo

# Scheduling dei threads

- Applicazione: set di threads che cooperano e sono eseguiti concorrentemente nello stesso spazio di indirizzamento
- Su monoprocesso i threads vengono usati per sovrapporre sequenze di I/O a sequenze di esecuzione
- Elevati guadagni di performance se threads di una stessa applicazione vengono eseguiti simultaneamente su più processori
- Approcci per l'assegnazione:
  - Load sharing – condivisione del carico
    - I processi non sono assegnati a specifici processori ed esiste una coda globale dei threads pronti
  - Gang scheduling – schedulazione a gruppi
    - Un set di threads correlati è schedulato per essere eseguito nello stesso istante su un set di processori, sulla base di un processo per processore
  - Assegnazione dedicata
    - I Threads sono assegnati ad uno specifico processore
    - Teoricamente ad un processo sono assegnati tanti processori quanti sono i suoi threads
  - Scheduling Dinamico
    - Il numero dei threads può essere modificato durante l'esecuzione

# Assegnazione di processore dedicato – Scheduling Statico

- Forma estrema di gang scheduling: un gruppo di processori viene dedicata ad una applicazione fino al suo termine
- Quando una applicazione è schedulata ognuno dei suoi threads viene assegnato ad un processore fino al termine dell'esecuzione dell'applicazione
- SVANTAGGI:
  - Alcuni processori potrebbero rimanere “fermi”
  - No multiprogramming of processors
- VANTAGGI:
  - Caso di sistemi con molti processori (centinaia)...
  - Annullamento totale dei cambi di contesto per tutta l'esecuzione di un programma



Es.

- Due applicazioni in esecuzione contemporanea
- 16 processori
- Ogni applicazione può essere suddivisa in più threads
- L'incremento di velocità diminuisce se il numero di threads diventa maggiore del numero di processori:
  - Maggior numero di switching e schedulazione

# Scheduling dinamico

- Il numero dei threads di un processo può essere modificato dinamicamente
- Il SO può modificare il carico per migliorare l'utilizzazione dei processori
- SO e applicazione collaborano nell'attività di scheduling:
  - SO responsabile dell'assegnazione dei processori ai processi
  - L'applicazione decide quali threads eseguire, quali sospendere, ecc.
- Politiche applicate dal SO:
  - Assegnare il processore ad un nuovo processo se tale processore è in idle
  - Assegnare un processore ad un nuovo processo requisendolo ad uno che ne detiene già più di uno
  - Se una richiesta non può essere soddisfatta, essa viene sospesa fino a quando un processore non diventa disponibile
  - Assegna il processore ad un processo che non ha correntemente alcun processore allocato

*NB: il vantaggio derivante dallo scheduling dinamico può essere annullato dall'overhead della sua gestione*

# Thread, SMP e MicroKernel

**THREAD  
SMP  
MICROKERNEL**

I moderni sistemi operativi

# THREAD

## Processo

- **possiede delle risorse:**

spazio di indirizzamento virtuale che contiene l'immagine del processo;

può chiedere ulteriore memoria;

può chiedere il controllo di canali di i/o;

può chiedere il controllo di dispositivi;

può chiedere files;

- **unità di esecuzione**

ha uno stato (ready, running, blocked, swapped, ....)

ha una priorità

deve essere schedulato

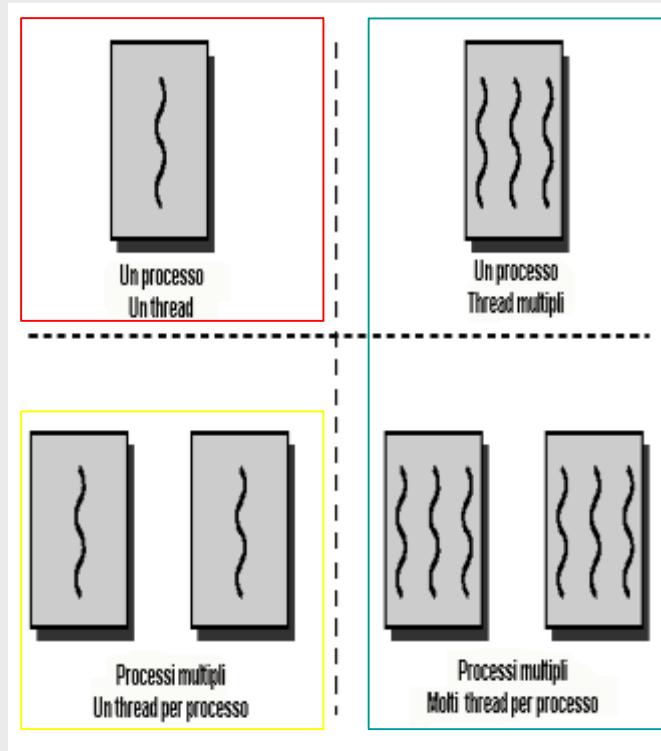
Caratteristiche trattate in maniera indipendente dal SO

## Thread

- **light weight process (LWP):** non possiede risorse  
elemento che viene allocato (main memory)  
è una traccia in esecuzione in uno o più programmi

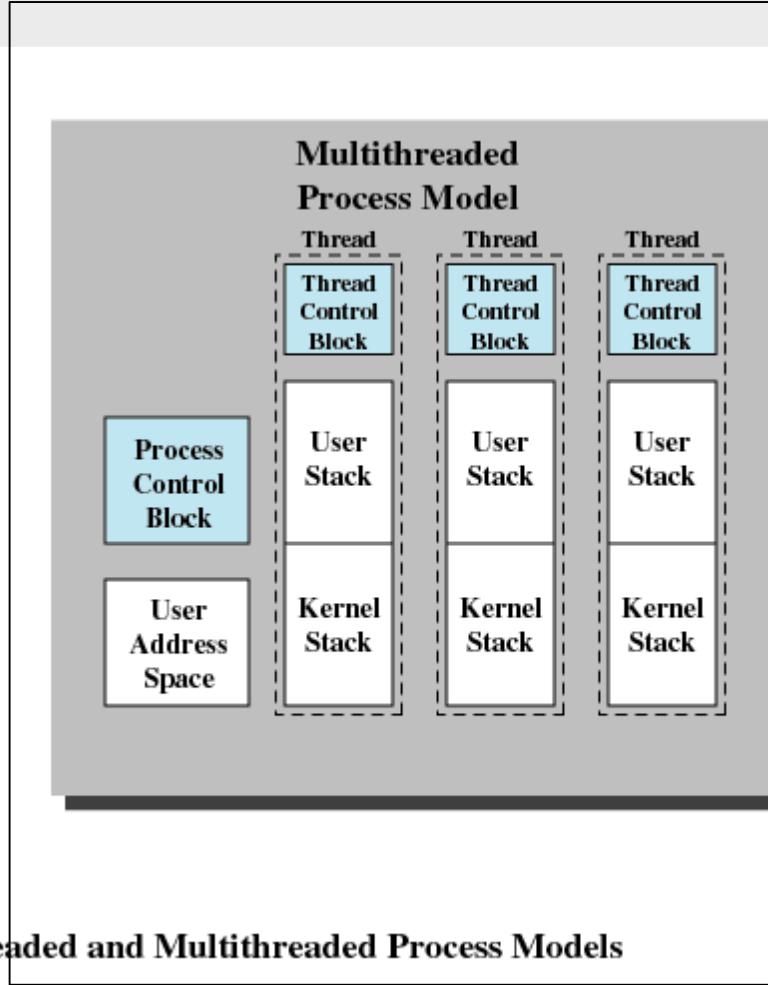
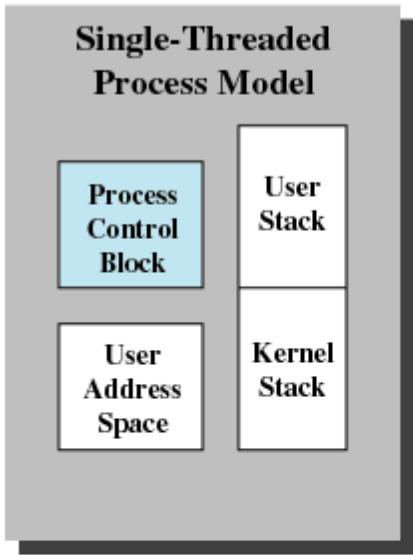
# MULTI-THREADING

Capacità di un SO di supportare più thread per ogni processo



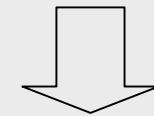
- **SINGOLO PROCESSO SINGOLO THREAD**  
MS-DOS (*il thread esiste nel processo*)
- **PROCESSI MULTIPLI A SINGOLO THREAD**  
UNIX
- **MULTITHREAD**  
WINDOWS  
SOLARIS  
MACH  
OS/2

# Processi a singolo e multi thread



## I thread

- Condividono lo stato e le risorse del processo a cui appartengono
- Risiedono nello stesso spazio di indirizzamento
- Hanno accesso agli stessi dati



Facilità di condivisione di informazioni...

Figure 4.2 Single Threaded and Multithreaded Process Models

# Vantaggi dei thread

- Tempo di creazione di un nuovo thread < tempo di creazione di un nuovo processo
- Tempo di terminazione di un thread < tempo di terminazione di un processo
- Tempo necessario allo switch tra threads all'interno dello stesso processo < tempo di switch tra processi
- I threads all'interno di uno stesso processo condividono memoria e files: scambio dati senza richiedere l'intervento del kernel. Necessità di sincronizzare le attività dei threads.

Alcuni esempi:

- Esecuzione in foreground e in background  
foglio di calcolo: *un thread gestisce il menu e legge i comandi, un altro thread esegue i comandi e aggiorna il foglio.*
- ELABORAZIONE ASINCRONA  
elaboratore di testo: *un thread di scarico su disco ad ogni minuto evita perdite per cadute di tensione.*
- VELOCITA' DI ESECUZIONE: *Lettura e calcolo fatti da threads diversi aumentano la velocità.*

# Qualche difficoltà...

- La sospensione di un processo richiede che tutti i thread siano sospesi contemporaneamente perché si deve liberare spazio in memoria e tutti i thread utilizzano lo stesso spazio di memoria condivisa.
- La terminazione di un processo richiede che tutti i thread siano terminati.

# Stato dei thread

- **READY**
- **RUNNING**
- **BLOCKED**
- **SUSPENDED**: non ha senso per un thread, è presente a livello di PROCESSO
  - Se un processo viene scaricato dalla memoria centrale, lo stesso avviene per tutti i suoi threads

## OPERAZIONI BASE per il cambio di stato dei thread:

- Creazione  
creaz. di un processo -> creaz. di un thread. Un thread può creare altri thread
- Blocco (attesa di un evento)  
salvataggio del contesto per il thread: PC, Stack pointer, registri CPU
- Sblocco  
lo stato nel TCB viene modificato (Blocked -> Ready)  
il thread viene accodato a quelli in attesa di processore.
- Terminazione  
deallocazione del contesto registri, deallocazione stack

il blocco di un thread blocca l'intero processo???

# Esempi di multithreading

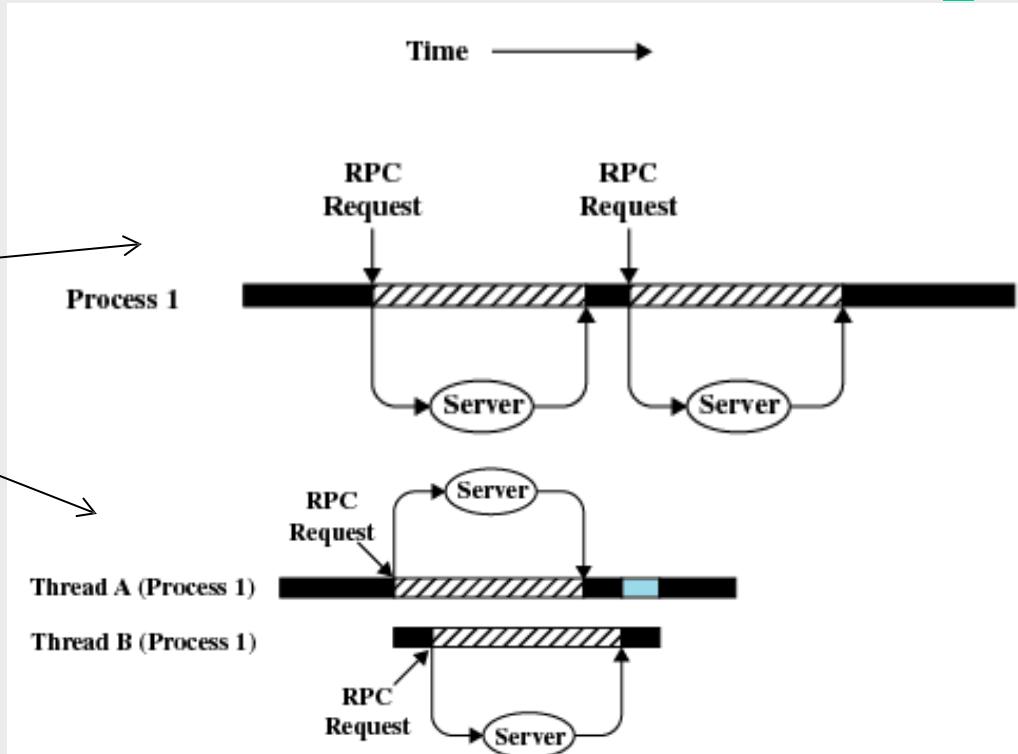
# Remote Procedure Call (RPC)

## Singolo processore

Due chiamate a RPC a due diversi host

a) programma a un solo thread

b) un thread per ogni RPC



**(b) RPC Using One Thread per Server (on a uniprocessor)**

 Blocked, waiting for response to RPC

 Blocked, waiting for processor, which is in use by Thread B

Running

# Esempi di multithreading

ALDUS Page maker: programma di scrittura, gestione e pubblicazione di pagine su desktop.(per OS/2)

Tre thread sempre attivi:

1 - Gestione degli eventi

2- Gestione dei servizi (stampa – viene chiamato un thread e non un altro processo, lettura dati, disposizione testo, attivazione altri T)

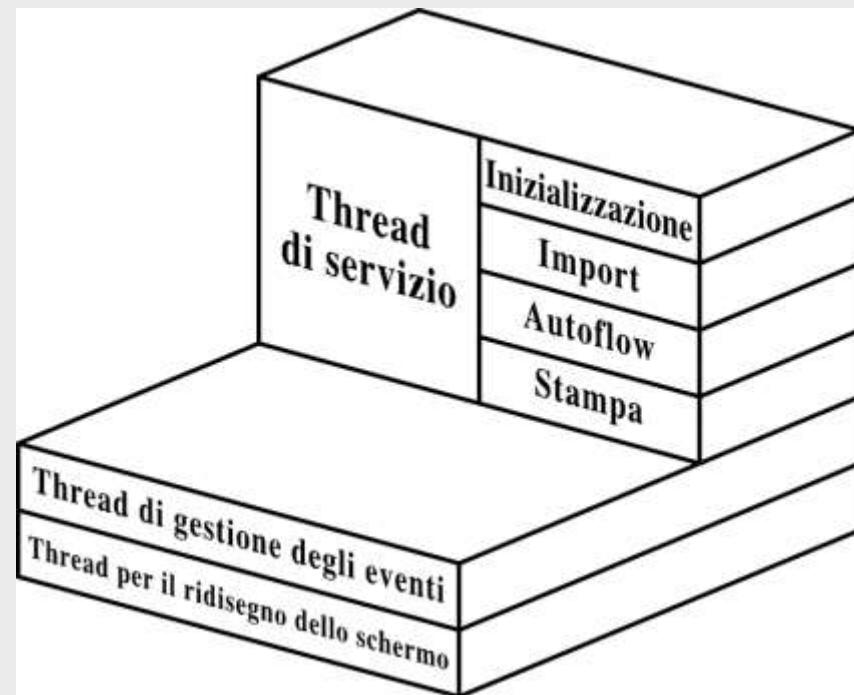
3 - Disegno dello schermo

*Esempio sorrisimento pagina con barra laterale:*

- Il thread eventi controlla la barra di scorrimento
- Il thread di ridisegno dello schermo ridisegna la pagina in base allo spostamento

*Necessità di sincronizzazione tra i due threads*

*NB: esistono attività bloccanti per tutti i threads: compare il cursore "busy"*

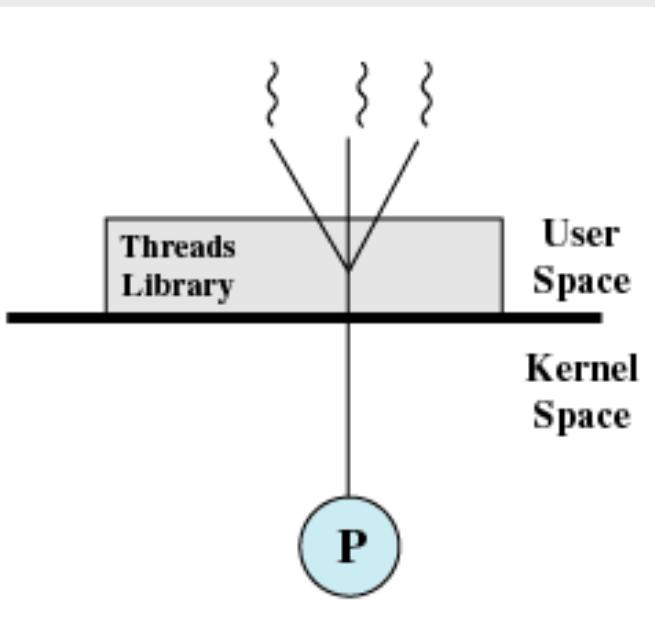


# Categorie di thread

- **ULT (User Level Thread)**
  - Realizzati tramite librerie senza l'intervento del kernel.  
Es. di librerie: Posix Pthread, Mach C-threads, UI-threads Solaris2
  - Trasparenti al kernel
  - Svantaggi: se il kernel è a singolo thread il blocco del thread di livello utente blocca l'intero processo (NB: il SO continua a schedulare processi)
- **KLT (Kernel Level Thread)**
  - Il kernel si occupa della creazione, scheduling e gestione
  - Possono essere eseguiti su diversi processori
  - Gestione più lenta degli ULT

# User-Level Thread

- Il lavoro di gestione dei threads è svolto dalla libreria utente
- Il kernel ignora l'esistenza dei threads
- Modello Molti a Uno



*La libreria permette:*

1. *Creazione e distruzione dei threads*
2. *Scambio messaggi tra threads*
3. *Schedulazione*
4. *Salvataggio e caricamento dei contesti dei threads*



*Attività svolte all'interno del singolo processo utente*

*Il kernel continua a schedulare i processi come unità a se stanti*

# ULT

## VANTAGGI:

- Risparmio di sovraccarico:
  - il cambio di Thread avviene all'interno dello spazio di indirizzamento utente
  - Non viene richiesto l'intervento del Kernel
- Schedulazione diversa per ogni applicazione:
  - Ottimizzazione in base al tipo di applicazione
- Ult eseguito da qualsiasi sistema operativo:
  - Libreria a livello utente condivisa dalle applicazioni

## SVANTAGGI:

- La chiamata a sistema da parte di un thread blocca tutti i thread del processo
- Il kernel assegna un processo ad un singolo processore quindi non si può avere multiprocessing a livello di thread (thread dello stesso processo su più processori)

### *Soluzioni Parziali:*

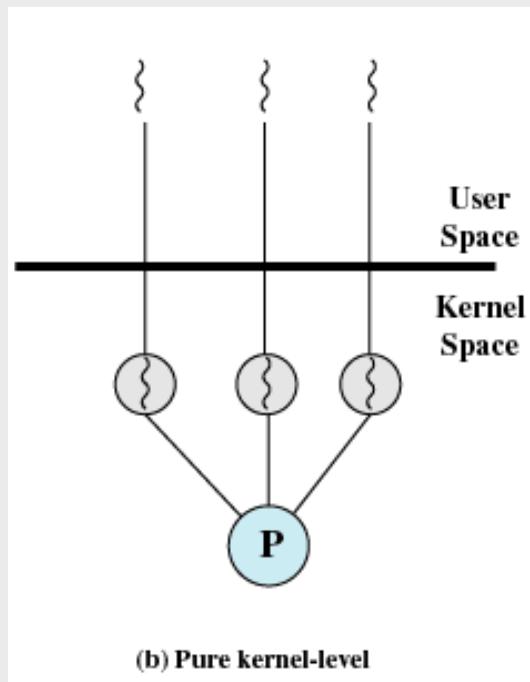
1. Sviluppo dell'applicazione a livello di processi (addio vantaggi dei thread)
2. *jacketing*: conversione di una chiamata bloccante in una non bloccante

Es.: Nel caso di I/O si invoca una procedura di jacketing che verifica se il dispositivo è occupato, in caso affermativo il thread passa in ready e un altro thread va in run.

qui

# Kernel-Level Thread PURO

- Il lavoro di gestione dei threads è svolto dal Kernel: modello uno a uno
- A livello utente una API consente l'accesso alla parte del Kernel che gestisce I thread



*Il kernel mantiene info su:*

1. Contesto del processo
2. Contesto dei threads
3. Scambio messaggi tra threads

*Schedulazione effettuata a livello di thread*

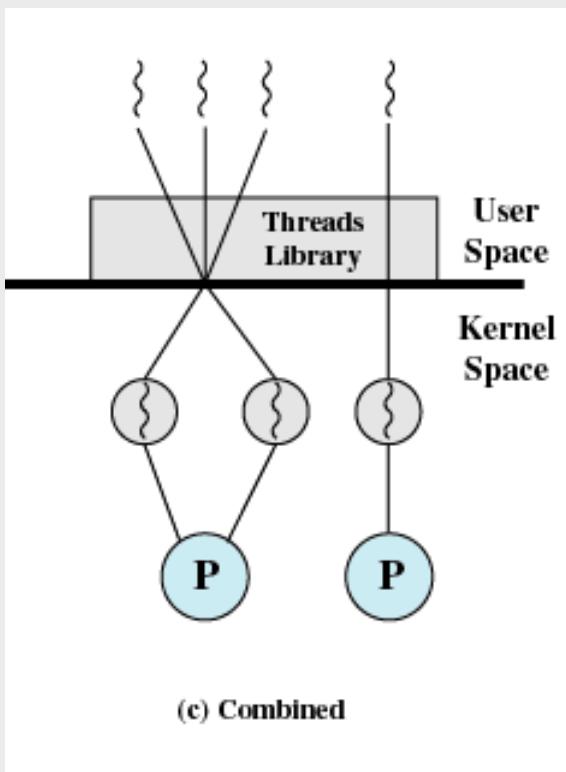
- se un thread di un P è bloccato, un altro thread dello stesso processo può essere eseguito
- Thread di uno stesso P possono essere schedulati su diversi processori

*SVANTAGGI:*

*Overhead: trasferimento del controllo da un thread ad un altro richiede l'intervento del kernel*

# Approcci MISTI

Modello molti a molti: più thread di livello utente sono in corrispondenza con più thread di livello kernel



I Thread sono creati nello spazio utente

Vari thread di uno stesso processo possono essere eseguiti contemporaneamente su più processori (Sun ULTRA 2, Solaris)

Una chiamata bloccante non blocca necessariamente l'intero processo

Necessità di comunicazione fra kernel e libreria di thread per mantenere un appropriato numero di kernel thread allocati all'applicazione.

**Light Weight Process (LWP)** – struttura intermedia appare alla libreria dei thread utente come un **processore virtuale** sul quale schedulare l'esecuzione.

Es. Una applicazione CPU-bound su un sistema monoprocessoresso implica che un solo thread per volta possa essere eseguito, quindi per essa sarà sufficiente un unico LWP per thread.

Una applicazione I/O-bound tipicamente richiede un LWP per ciascuna chiamata di sistema bloccante.

# Relazioni tra THREAD e PROCESSI

Thread: Processi	Descrizione	Sistemi
1:1	Ogni thread di esecuzione è un processo unico con il proprio spazio di indirizzamento e le proprie risorse	Molte implementazioni di UNIX
M:1	Ogni processo ha associato un proprio spazio di indirizzamento e delle risorse. In ogni processo si possono creare ed eseguire molti thread.	Windows NT, Solaris, OS/2, OS/390, MACH
1:M	<p><i>Ambienti distribuiti: threads possono spostarsi tra più calcolatori</i></p> <p>Un thread può spostarsi da un processo all'altro; ciò permette di spostare facilmente i thread fra sistemi diversi.</p>	Ra(Clouds), Emerald
M:M	Combina le proprietà degli approcci M:1 e 1:M.	TRIX

# Calcolatori attuali: Symmetric Multi Processing (SMP)

- Un calcolatore con molti processori
- I processori condividono le stesse risorse
- Tutti i processori possono effettuare le stesse funzioni
- Ogni processore esegue una stessa copia del SO
- Ogni processore gestisce la schedulazione dei processi o thread disponibili
- Difficoltà:
  - I processori non devono schedulare lo stesso processo
  - I processi in coda non devono essere persi
  - Comunicazione tra processori: memoria condivisa (possibilità di effettuare accessi simultanei alla memoria – memoria multiporta)
  - Coerenza della \$: RAW, WAR, RAR, WAW (risolti a livello hardware)

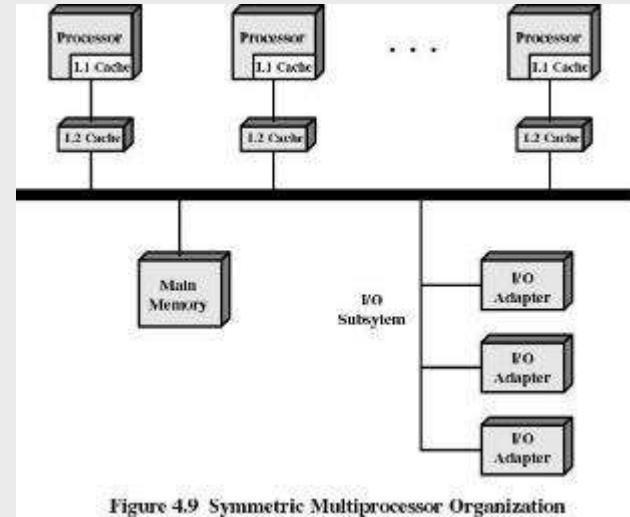


Figure 4.9 Symmetric Multiprocessor Organization

# SO per SMP

Multi-Processore trasparente all'utente:  
multiprogrammazione su monoprocessoress

## PUNTI CRITICI DELLA PROGETTAZIONE:

- Processi e Thread del Kernel concorrenti: l'esecuzione contemporanea su diversi processori non deve compromettere le strutture di gestione del SO (tabelle, ecc.)
- Schedulazione: necessità di evitare conflitti - *capitolo schedulazione*
- Sincronizzazione: mutua esclusione e ordinamento degli eventi
- Gestione della memoria condivisa
- Tolleranza ai guasti: in caso di “perdita di un processore” devono essere aggiornate le strutture di controllo del SO

# MicroKernel

Piccolo Nucleo del SO

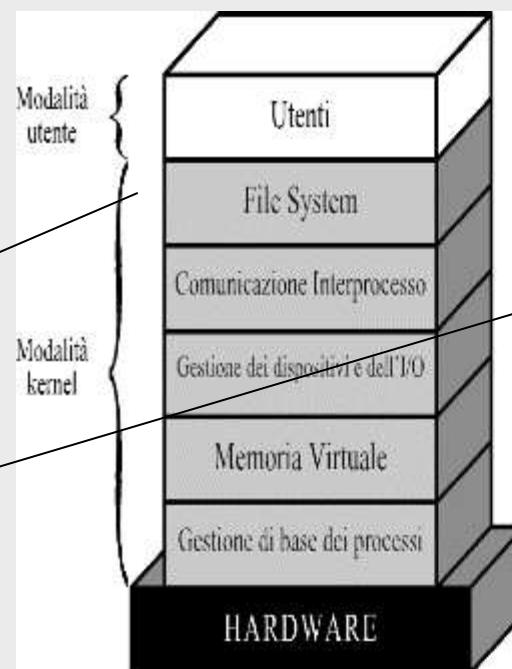
Contiene le funzioni essenziali del SO

Servizi tradizionalmente inclusi nel SO sono sottosistemi esterni al microkernel ed eseguiti in modalità utente:

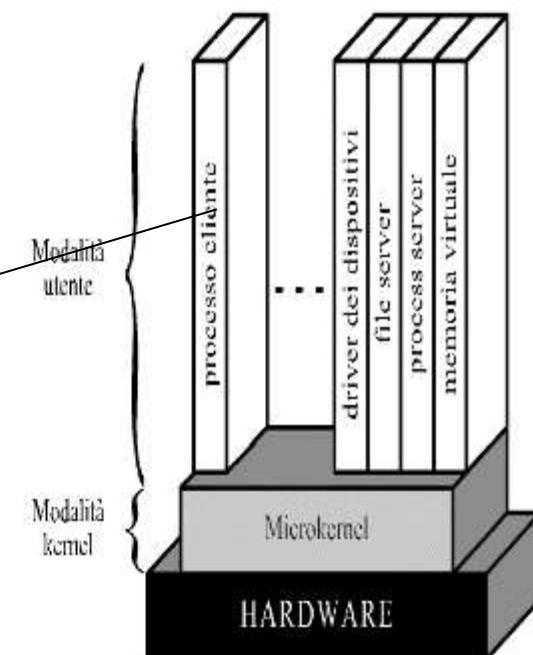
- Device drivers
- File systems
- Virtual memory manager
- Windowing system
- Security services

*Interazione solo tra strati adiacenti*

*La comunicazione avviene attraverso il MicroKernel che ridireziona i messaggi*



(a) Kernel a livelli



(b) Microkernel

# Vantaggi derivanti dal MicroKernel

- Interfaccia uniforme: I moduli usano le stesse interfacce per le richieste al microKernel
- Estensibilità: introduzione di nuovi servizi o modifiche non richiedono modifiche del microKernel
- Flessibilità: a seconda delle applicazioni certe caratteristiche possono essere ridotte o potenziate per soddisfare al meglio le richieste dei clienti. Es. Windows 7 home – professional – ultimate
- Portabilità: Il cambio dell'hardware comporterà unicamente la modifica del microkernel.
- Affidabilità: lo sviluppo di piccole porzioni di codice ne permette una migliore ottimizzazione e test.
- Supporto ai sistemi distribuiti: ogni servizio è identificato da un numero nel microkernel e una richiesta da client non è necessario che sappia dove si trova il server in grado di soddisfare la stessa. Messaggistica gestita dal microkernel

# MicroKernel Design: What's in, What's out?

Il microkernel deve contenere:

- Le funzioni che dipendono direttamente dall'hardware (gestione degli interrupt e I/O)
- Le funzioni per la comunicazione tra processi (IPC)
- Gestione primitiva della memoria

Problema delle prestazioni per sistemi microkernel:

Costruire, inviare, accettare, decodificare un messaggio costa più che una chiamata a SO

Possibili soluzioni:

Aggiungendo funzionalità al microkernel (MACH OS) si riduce il numero di cambiamenti di stato (utente/kernel)

Riduzione di flessibilità, interfacce minime..

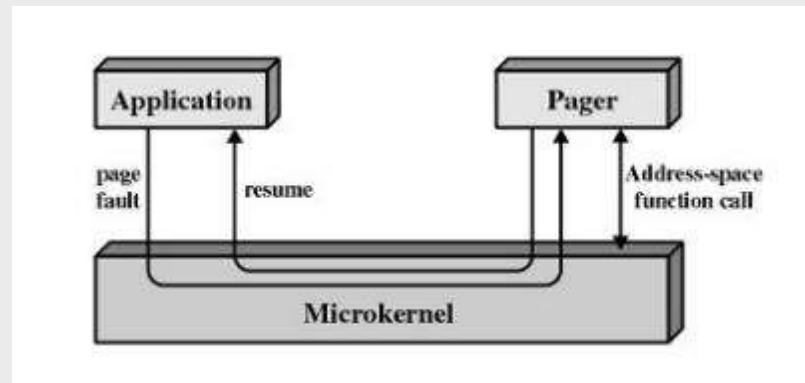
Ulteriore riduzione del microkernel

# Funzioni minime del microkernel: Gestione Primitiva della memoria

Un modulo esterno al microkernel mappa pagine virtuali in pagine fisiche.

Il mapping è conservato in memoria principale.

- Una applicazione che accede ad una pagina che non si trova in memoria genera un page fault
- l'esecuzione passa al microkernel che invia un msg al paginatore comunicando la pagina richiesta
- La pagina viene caricata (paginatore e kernel collaborano per il mapping memoria reale-virtuale)
- Quando viene caricata la pagina il pager invia un msg all'applicazione



# Comunicazione tra processi

Messaggio = (intestazione) + (corpo) + (puntatore+inform. di contr.)

- Intestazione (*Mittente, Ricevente*)
- Corpo (*dati del messaggio*)
- Puntatore (*informazioni di controllo del processo, blocco dati*)
- Associata ad ogni processo c'è una PORTA: capability list indica chi può inviare messaggi. Porta amministrata dal Kernel

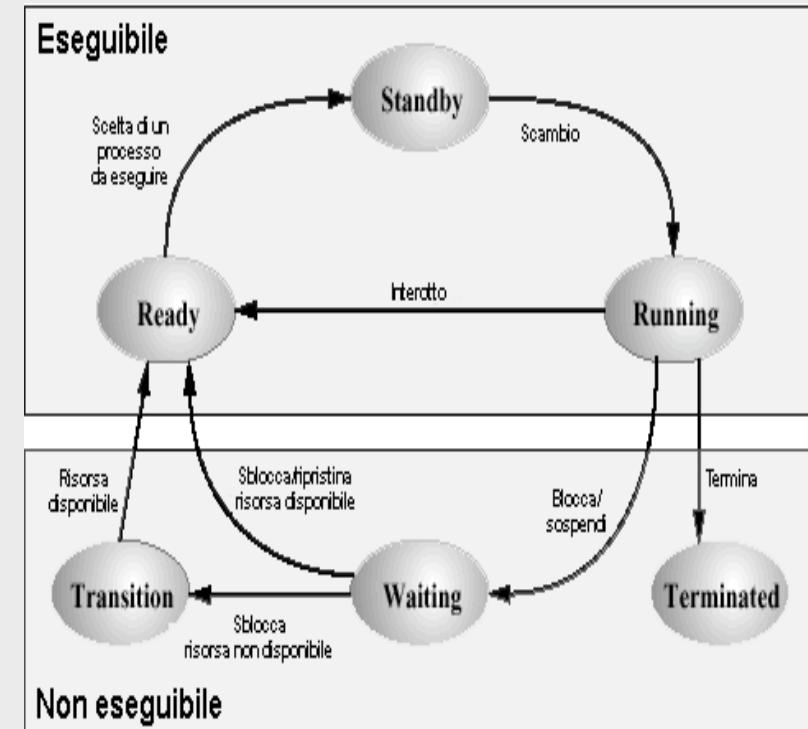
# Gestione Interrupt e I/O

- Il microkernel riconosce gli interrupt ma non li gestisce direttamente.
- Il microkernel **trasforma l'interrupt in messaggio a livello utente**, che invia al processo che gestisce l'interrupt

```
driver thread:  
do  
    wait(msg, mittente);  
    if mittente=mio_interrupt.hardware  
        then leggi/scrivi le porte di I/O;  
              azzerà l'interrupt hardware  
    else....  
Endif  
enddo
```

# Stati dei Thread in Windows

- **Ready**
- **Standby**: legato alla disponibilità del particolare **processore** (SMP) richiesto per il thread.  
Se la priorità è sufficientemente alta il processo in running può essere interrotto.
- **Running**
- **Waiting**: I/O, attesa per sincronizzazione
- **Transition**: thread pronto per l'esecuzione ma le risorse non sono disponibili (es. lo stack può essere stato spostato su disco mentre era in waiting).
- **Terminated**



*Supporto di SMP:*

- I thread (inclusi quelli del kernel) possono essere eseguiti su ogni processore
- Il primo thread in ready viene assegnato al primo processore disponibile
- Thread appartenenti allo stesso processo possono essere eseguiti (contemporaneamente) su diversi processori
- Esecuzione di un thread sempre sullo stesso processore: dati ancora in cache.

Concorrenza

# CONCORRENZA: Mutua Esclusione e Sincronizzazione

# Concorrenza: genesi

- Multiprogrammazione:
  - gestione di più processi su un singolo processore.
- Multiprocessing:
  - gestione di più processi su più processori.
- Processi distribuiti
  - cluster
- Competizione tra processi (threads) per ottenere (e condividere) le risorse:
  - Cpu
  - Memoria
  - Canali di I/O
  - Files
  - ecc..

# Concorrenza: terminologia

- **Sessione Critica:** porzione di codice all'interno di un processo (thread) che richiede accesso a risorse condivise.
- **Deadlock:** situazione nella quale due o più processi sono impossibilitati dal procedere poiché sono in attesa l'uno dell'altro
- **Livelock:** situazione nella quale due o più processi cambiano continuamente il proprio stato a causa del cambiamento di stato degli altri (senza fare alcun lavoro utile)
- **Mutua esclusione:** requisito per il quale, quando un processo è nella propria sezione critica, nessun altro processo può essere nella propria sezione critica se questa fa riferimento a risorse condivise con il primo processo.
- **Race condition:** situazione nella quale thread o processi leggono e scrivono un dato condiviso e il risultato dipende dalla loro velocità reciproca
- **Starvation:** situazione nella quale un processo non riceve mai l'utilizzo di una risorsa e viene costantemente scavalcato da altri processi

# Vantaggi e Problemi derivanti dalla concorrenza

## VANTAGGI

benefici sulla esecuzione nonostante il **sovraffollamento (context switching)**.  
Migliore utilizzazione delle risorse.

## PROBLEMI

### • Singolo Processore

- Condivisione pericolosa: ordine delle operazioni di lettura e scrittura su aree di memoria condivise
- Difficoltà nell'assegnare le risorse ai processi in maniera ottimale
- Difficoltà nella rilevazione degli errori nel codice e dei conflitti di interlacciamento

```
void echo ()  
{  
    char in,out;  
    scanf ("%c", &in);  
    out = in;  
    printf ("%c", out);  
}
```

*Condivisione della procedura echo():*

- Risparmio dello spazio di memoria
- Due processi concorrenti.
  - P1 viene interrotto dopo la scanf,
  - P2 esegue tutto echo,
  - P1 viene riattivato da scanf in poi e ha perso il dato che aveva letto

**Soluzione: un solo processo alla volta (MUTUA ESCLUSIONE)**

# Vantaggi e Problemi derivanti dalla concorrenza

- **SMP**

- stessi problemi di un calcolatore a singolo processore (una interruzione può fermare l'esecuzione di un processo in un qualsiasi istante)
- interlacciamento esecuzione processi paralleli

**Processo P1 - Processore1**

```
.....  
scanf ("%c", &in);  
.....  
out = in;  
printf ("&c", out);  
.....
```

**Processo P2 - Processore2**

```
.....  
scanf ("%c", &in);  
.....  
out = in;  
.....  
printf ("&c", out);
```

*Il carattere letto da P1 è perso prima di poter essere stampato (perdita di aggiornamento)*

**Soluzione: MUTUA ESCLUSIONE**

Un solo programma per volta può entrare nella propria sezione critica

Es.: Un solo programma per volta può inviare comandi alla stampante

# Requisiti per la mutua esclusione

I meccanismi che provvedono alla mutua esclusione devono garantire i seguenti requisiti:

1. Un solo processo alla volta deve accedere alla sezione (o risorsa) critica;
2. Un processo fuori della sezione critica non deve interferire con il processo nella sezione critica
3. Ogni processo deve poter accedere dopo un tempo finito di attesa in coda alla risorsa critica (no stall o starvation)
4. Se nessun processo è nella sezione critica, un processo deve poter entrare nella sezione critica senza attese
5. Non ci devono essere supposizioni sulla velocità di esecuzione relativa dei processi
6. Il tempo di permanenza nella sezione critica è (de)finito

# Meccanismi di Mutua Esclusione

**Approcci software:** i processi, senza ausilio del Sistema Operativo o del linguaggio di programmazione, devono coordinarsi tra loro (Dekker)

- Aumento del tempo di esecuzione
- Errori frequenti

Utilizzo di particolari **istruzioni di macchina** (test-set, scambio)

- Riduzione del sovraccarico
- Non soddisfacenti

**Supporto del sistema operativo o del linguaggio di programmazione**

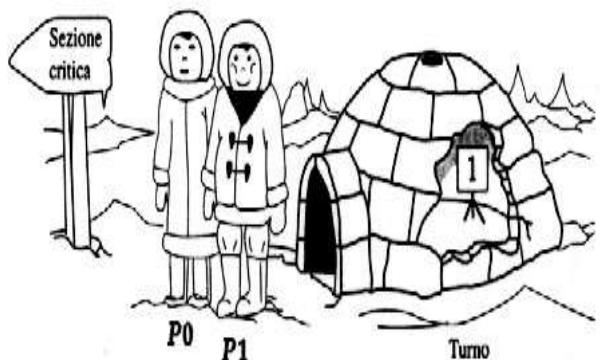
- Scambio Messaggi (Inter Process Communication)
- Semafori
- Monitor

# **Mutua Esclusione: un approccio software**

# Algoritmo di Dekker

## 1° tentativo

MUTUA ESCLUSIONE: Un solo accesso per volta accede alla risorsa condivisa



**Protocollo dell'Iglù:** prima di entrare nella sezione critica, i processi controllano uno alla volta una variabile turno

Busy wait: consuma tempo utile di esecuzione

```
var turno= 0..1; //variabile globale condivisa  
Processo 0
```

```
....  
while (turno!=0)  
{nulla} //busy wait  
<sezione critica>;  
turno=1;
```

```
Processo 1
```

```
....  
while (turno!=1)  
{nulla} //busy wait  
<sezione critica>;  
turno=0;  
....
```

PRO: garantisce la mutua esclusione

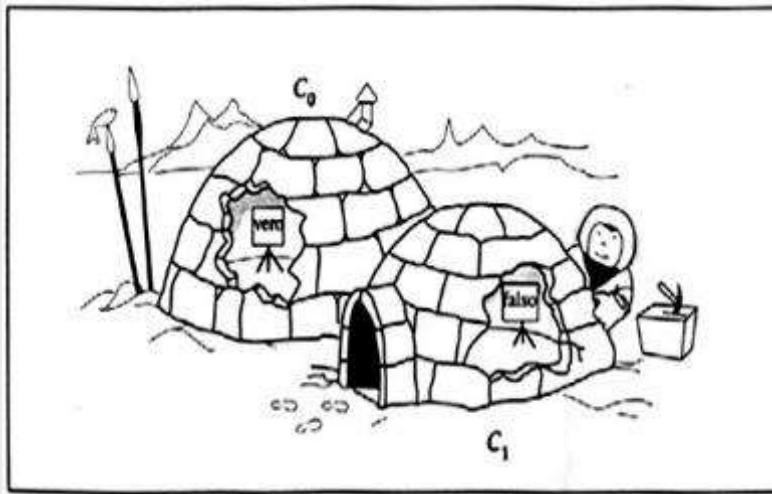
PUNTI DEBOLI:

- 1.I processi devono osservare l'alternanza, il più lento determina la velocità di avanzamento di entrambi i processi
- 2.Se un processo fallisce nella propria sezione critica, l'altro processo rimarrà bloccato per sempre

# Algoritmo di Dekker

## 2° tentativo

*Ogni processo ha un flag relativo all'utilizzo della risorsa e può leggere il flag dell'altro senza modificarlo*



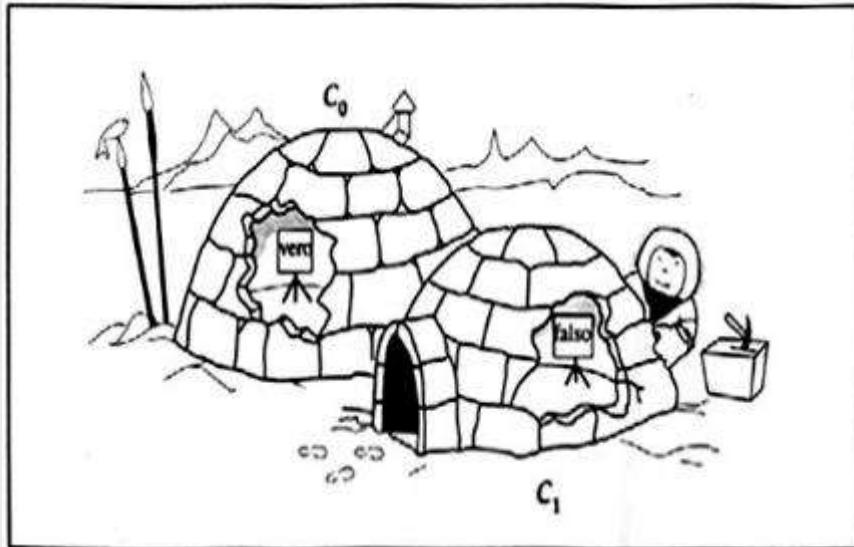
```
boolean flag[2];  
  
//Processo 0  
...  
while (flag[1])  
    {nulla;}  
flag[0]=true;  
<sezione critica>  
flag[0]= false;  
...
```

PRO: se un processo fallisce fuori della sua sezione critica l'altro può continuare a lavorare  
CONTRO:

1. se un processo fallisce entro la sezione critica o prima di mettere false nel suo flag allora l'altro è bloccato per sempre.
2. se entrambi vedendo il flag dell'altro false, mettono il loro flag a true, entrambi vanno nella sezione critica, senza mutua esclusione. Dipendenza della velocità dei processi.

# Algoritmo di Dekker

## 3º tentativo



PRO: la mutua esclusione è garantita

PROBLEMI:

- Se un processo fallisce entro la sua sezione critica l'altro è bloccato
- Se entrambi settano il flag a true prima che uno dei due verifichi la condizione del while si ha stallo

```
boolean flag[2];  
  
// Processo 0  
...  
flag[0] = true; //indico prima del  
//test di voler andare in sezione  
//critica  
while (flag[1])  
{nulla}  
<sezione critica>;  
flag[0] = false;  
...
```

# Algoritmo di Dekker

## 4° tentativo

### Processo 0

```
...
flag[0]= true;
while (flag[1])
{
    flag[0] = false;
    <pausa>
    flag[0] = true;
}
<sezione critica>;
flag[0] = false;
...
```

### Processo 1

```
...
flag[1] = true;
while (flag[0])
{
    flag[1] = false;
    <pausa>;
    flag[1] = true;
}
<sezione critica>;
flag[1] = false;
...
```

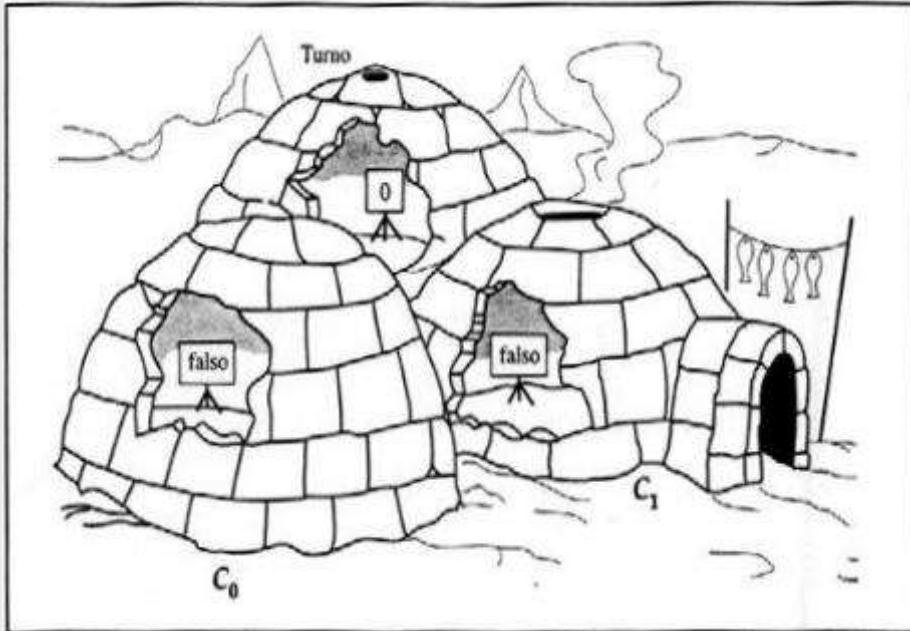
*PRO: mutua esclusione garantita*

*PROBLEMI:*

- P0: “vado io”, P1: “vado io”, P0:“non vado io”, P1: “non vado io”, attesa....  
*in realtà non’è uno stallo poiché chi esce prima dal ciclo di attesa riesce ad entrare nella sezione critica*
- Se un processo fallisce entro la sua sezione critica l’altro è bloccato

# Algoritmo di Dekker

## Una soluzione corretta



Ogni processo ha una propria variabile **flag** che indica se vuole andare nella sezione critica o meno

Variabile **turno** specifica chi ha il diritto di insistere nel tentativo di entrare nella propria sezione critica

Es.:

P0 vuole entrare (pone il suo flag a true)

P0 controlla il flag di P1

se falso entra nella sezione critica

se vero controlla il turno

se turno è 0 (il suo) continua a controllare periodicamente il flag di P1

se turno è 1 pone il suo flag a falso lasciando il passo a P1

# Algoritmo di Dekker

## Una soluzione corretta

```
boolean flag[2];
int turno;
void main()
{   flag[0]=false;           flag[1]=false;           turno=1; //turno=0;
...
processo P0;
processo P1;
...
}

//processo P0
{
...
    flag[0]=true;
    while(flag[1])
    {
        if (turno==1)
        {
            flag[0]=false;
            while(turno==1)
                {<>nulla...ATTESAATTIVA>}
            flag[0]=true;
        }
    }
    <sezione critica>
    flag[0]=false;
...
}
```

# Algoritmo di Dekker

## Una soluzione corretta

- Algoritmo “complesso”
- Attesa attiva: un processo controlla continuamente il flag dell’altro processo se pari a true
- Se un processo fallisce nella propria sezione critica l’altro processo rimane bloccato per sempre

# Mutua Esclusione: Supporto Hardware

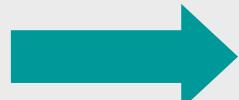
Macchine monoprocessoressi: i processi si alternano in esecuzione (EXE)  
la mutua esclusione si può ottenere evitando l'interruzione di un processo attivando e  
disattivando gli interrupt (supporto hardware)

<disattiva le interruzioni>  
<sezione critica>  
<attiva le interruzioni>

PRO: mutua esclusione garantita

PROBLEMI:

1. Efficienza peggiora poiché il processore non può alternare i processi liberamente
2. Non funziona su macchine SMP



Soluzione su SMP

*Istruzioni macchina speciali per l'accesso a locazioni di memoria in modo atomico (non interrompibile)*  
test-and-set  
scambio (swap)

l'accesso sequenziale ad una locazione di memoria è garantita dall'hardware  
Se si eseguono due test-and-set (swap) contemporaneamente, esse vengono  
serializzate

# Mutua Esclusione: Supporto Hardware - test&set

```
boolean TestAndSet (boolean *target)
{
    boolean val = *target;
    *target = TRUE;
    return val;
}
```

- Utilizzo di test&set per garantire la mutua esclusione  
sia lock una variabile boolean condivisa inizializzata a falso (la risorsa è libera).

```
while (true) {
    while ( TestAndSet (&lock ) )
        ; /* do nothing
    <critical section>
    lock = FALSE; //rilascio
    ...
}
```

# Mutua Esclusione: Supporto Hardware - swap

```
void swap (boolean *a, boolean *b)
{
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```

- Utilizzo di swap per garantire la mutua esclusione  
sia lock una variabile booleana condivisa inizializzata a FALSE //indica risorsa  
accessibile

```
while (true)  {
    key = TRUE;
    while (key == TRUE)
        swap (&lock, &key ); //in key torna false
    <critical section>
    lock = FALSE;
    ...
}
```

# Mutua Esclusione: Supporto Hardware

## Vantaggi

- si può applicare a un qualsiasi numero di processi anche su multiprocessori a memoria condivisa;
- si può usare per gestire più di una sezione critica, ciascuna con una propria variabile;

## Svantaggi

- Attesa attiva (i processi consumano tempo di cpu)
- Starvation (la scelta di quale processo andrà nella sezione critica è arbitraria)
- Stallo
  - Es (singolo processore)
    - P1 in sezione critica
    - P2 ha priorità più alta di P1
    - P2 tenterà di accedere (tramite test&set o swap) alla risorsa bloccata da P1 e nella sua attesa attiva non lascerà mai il posto a P1 che ha priorità più bassa

# Mutua Esclusione: Supporto del SO e dei ling. di prog.

## SEMAFORI

SEMAFORO: variabile (intera) sulla quale sono possibili 3 operazioni:

1. Inizializzazione ad un valore non negativo
2. Operazione atomica **wait()**: decrementa il valore della variabile. Se il valore della variabile diventa negativa, il processo che ha eseguito la wait viene bloccato.
3. Operazione atomica **signal()**: incrementa il valore della variabile. Se il valore della variabile è negativo, uno dei processi bloccati sull'operazione di wait viene sbloccato

- *Si associa un semaforo ad ogni risorsa condivisa*
- *Il processo che vuole utilizzare la risorsa effettua una operazione di wait*
- *Il processo che rilascia la risorsa effettua il signal*
- *La variabile numerica indica il numero di istanze di una specifica risorsa condivisa (semaforo contatore)*
- *Se la variabile è negativa, essa rappresenta (presa in valore assoluto) il numero di processi in attesa*

```
wait (S);  
<Critical  
Section>  
signal (S);
```

# Implementazione dei semafori Contatore

```
typedef struct {
    int istanze;
    struct processo *P; //lista dei processi in coda
} semaforo;
```

```
void wait(semaforo s)
{
    s.istanze--;
    if(s.istanze<0)
    {
        <poni processo in coda>
        <blocca questo processo: running->blocked >
    }
}
```

```
void signal(semaforo s) :
{
    s.istanze++;
    if(s.istanze<=0)
    {
        <rimuovi un processo in coda>
        <sveglia il processo: blocked->ready >
    }
}
```

# Implementazione dei semafori binari

*Semaforo binario: semaforo il cui valore intero può essere solo 0 o 1  
Gestione più complessa che non con semafori contatore*

```
typedef struct {
    boolean val;
    struct processo *P; //lista dei processi in coda
} semaforo_bin;
```

```
void wait(semaforo_bin s)
{
    if (s.val==1)
        s.val=0;
    else
        {
            <poni processo in coda a P>
            <blocca questo processo: running->blocked >
        }
}
```

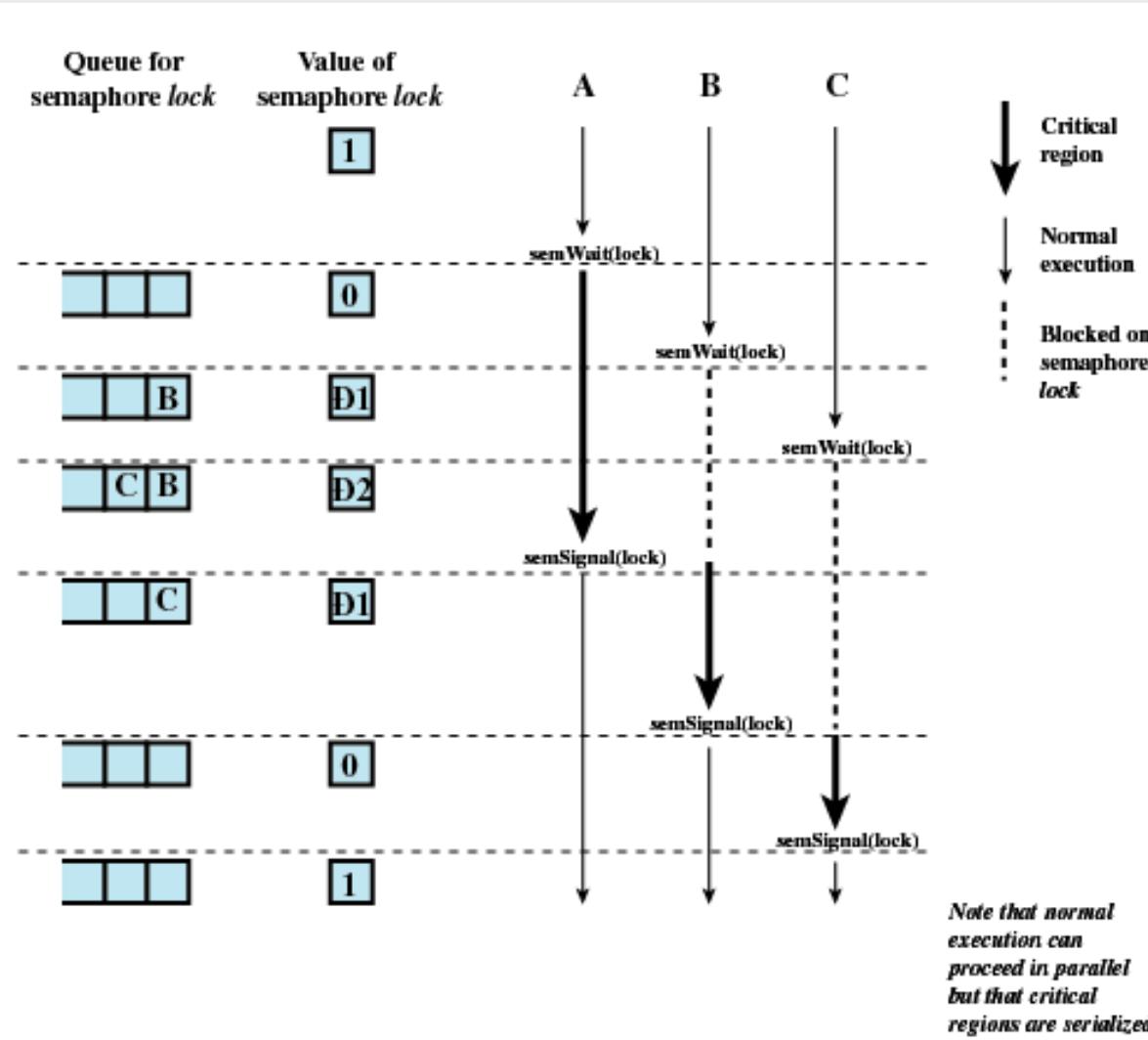
```
void signal (semaforo_bin s)
{
    if(*P==NULL) //coda vuota
        s.val=1;
    else
        {
            <rimuovi un processo in coda>
            <sveglia il processo: blocked->ready >
        }
}
```

# Implementazione dei semafori ...

Come fare affinchè signal e wait siano atomiche??

1. **Implementarle in hardware o firmware**
2. Dekker o Peterson... sovraccarico di elaborazione
3. **Utilizzo dell'istruzione atomica test&set**  
Esercizio: si scriva lo pseudocodice C
4. Se il sistema è monoprocessoresso basta disabilitare gli interrupt durante le operazioni  
Esercizio: si scriva lo pseudocodice C

# Accesso a dato condiviso tramite l'uso dei semafori



# Deadlock e Starvation

- Deadlock – due o più processi sono in attesa di un evento che può essere determinato 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);

- $P_1$  non rilascia  $S$  fino a quando non ottiene  $Q$
- $P_0$  non rilascia  $Q$  fino a quando non ottiene  $S$

*Le signal non saranno mai eseguite: STALLO*

- Starvation – indefinite blocking. Un processo non viene mai rimosso dalla coda al semaforo. Si immagini una gestione LIFO.

# Produttori e Consumatori

## - Situazione tipica di processi concorrenti -

Tipica rappresentazione dei processi concorrenti:

Uno o più produttori generano dati inserendoli in un buffer  
Un consumatore preleva i dati uno alla volta

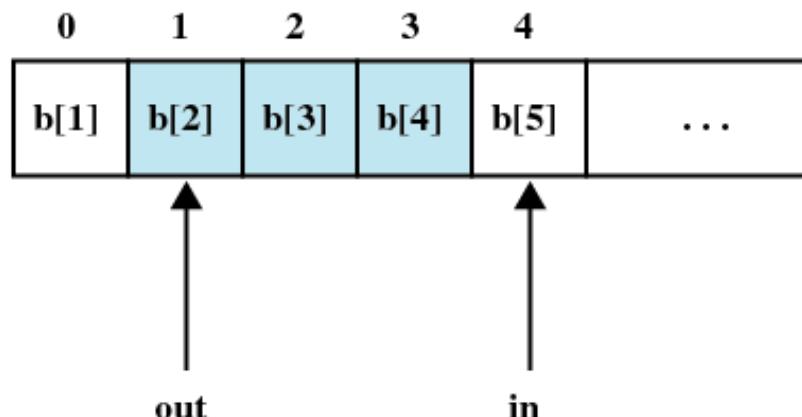
- Ipotesi di buffer infinito: l'accesso al buffer tra produttore e consumatore deve essere mutuamente esclusivo

produttore

```
<produce dato>
buffer[in]=dato;
in++;
```

consumatore

```
while(in >= out)
{
    w = buffer[out];
    out++;
    <consuma w>
}
```



# Produttori e Consumatori

## Buffer infinito – semafori binari

```
/* program producerconsumer */
int n; //Numero di elementi nel buffer
binary_semaphore s = 1; //gestisce l'accesso al buffer
binary_semaphore delay = 0; //gestisce il caso di nessun
void producer()
{
    while (true)
    {
        produce();
        semWaitB(s);
        append();
        n++;
        if (n==1) semSignalB(delay);
        semSignalB(s);
    }
}
void consumer()
{
    int m; /* a local variable */
    semWaitB(delay);
    while (true)
    {
        semWaitB(s);
        take();
        n--;
        m = n;
        semSignalB(s);
        consume();
        if (m==0) semWaitB(delay);
    }
}
void main()
{
    n = 0;
    parbegin (producer, consumer);
}
```

...con semafori binari...

Essendo il buffer infinito, il produttore può inserire tutto quello che produce

Se  $n=1$ , significa che il buffer prima era vuoto ed occorre avvisare il consumatore

Si pone in attesa che il primo elemento venga prodotto

Se il consumatore ha svuotato il buffer, si mette in attesa che venga prodotto un nuovo elemento

# Produttori e Consumatori

## Buffer infinito – semafori contatore

```
/* program producerconsumer */
semaphore n = 0; //Gestisce il caso di nessun elemento nel buffer
semaphore s = 1; //Gestisce l'accesso al buffer
void producer()
{
    while (true)
    {
        produce();
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (true)
    {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

...con semafori contatore...

→ È indifferente il loro  
ordine???

SI

→ È indifferente il loro  
ordine???

NO..

Il consumatore  
entrerebbe nella sezione  
critica quando il buffer è  
vuoto e nessun  
produttore potrebbe  
aggiungere elementi:  
STALLO

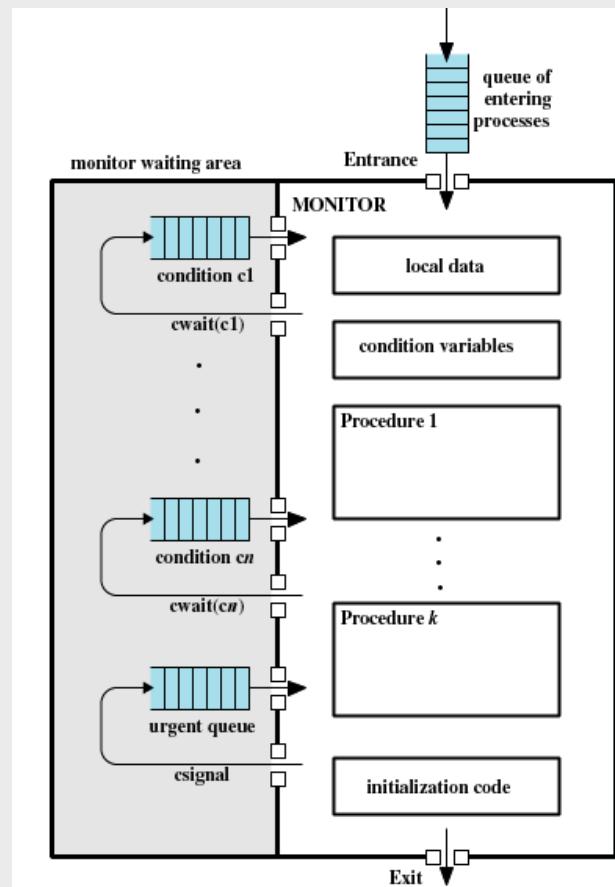
Figure 5.11 A Solution to the Infinite-Buffer Producer/Consumer Problem

# MONITOR

I semafori sono primitive potenti per gestire la mutua esclusione, mala scrittura di un programma con l'utilizzo dei semafori può essere tutt'altro che semplice.

## MONITOR:

- costrutto di sincronizzazione di alto livello
- modulo software le cui caratteristiche principali sono:
  - Variabili locali accessibili solo dalle procedure (metodi) definite al suo interno e non da procedure esterne
  - Un processo entra nel monitor chiamando le sue procedure
  - Un solo processo alla volta può essere in esecuzione all'interno del monitor. Gli altri processi sono sospesi nell'attesa che il monitor diventi disponibile
- UTILIZZO:
  - Mutua esclusione
  - Protezione delle strutture dati condivise
- Sincronizzazione tramite variabili di condizione. Su tali variabili si opera mediante:
  - *cwait ( c )* : sospende l'esecuzione del processo chiamante sulla condizione c
  - *csignal ( c )* : riattiva un processo sospeso sulla condizione c. NB: se non c'è nessun processo in attesa su tale condizione il segnale viene perso



## Concorrenza

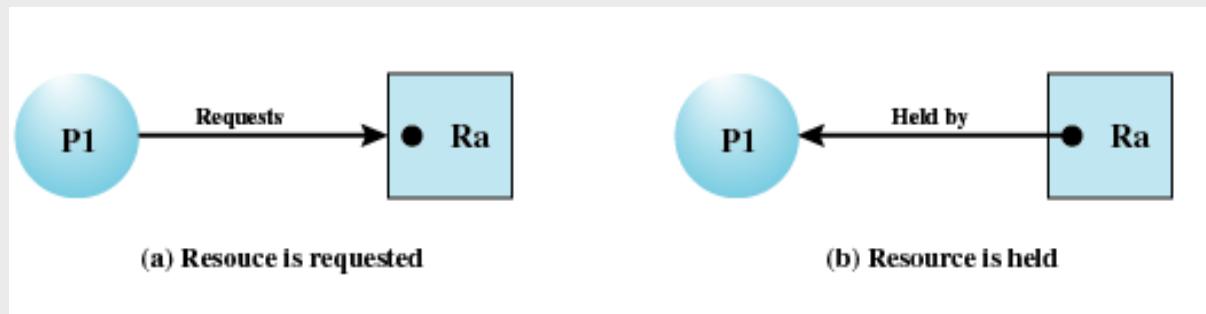
# CONCORRENZA: Stallo e Starvation

# Introduzione allo Stallo

- Definizione :  
*Un insieme di processi è in stallo se ogni processo nell'insieme è in attesa di un evento che solo un altro processo nello stesso insieme può generare.*
- Generalmente l'evento è il rilascio di una risorsa
- Nessuno dei processi in stallo può ...
  - passare in esecuzione
  - rilasciare risorse
  - essere riattivato

Grafici di Allocazione delle risorse:

- Il processo P1 richiede la risorsa Ra (fig. a)
- La risorsa Ra è posseduta dal processo P1 (fig. b)



# Condizioni per lo Stallo

**Mutua esclusione:** un solo processo alla volta può usare una risorsa

**“hold & wait” – possesso e attesa:** un processo può mantenere il possesso delle risorse allocate mentre attende di averne altre

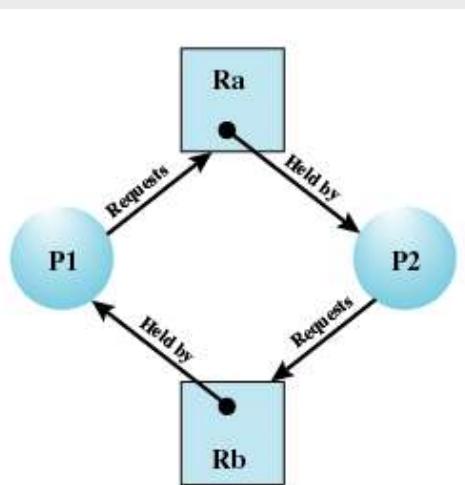
**Assenza di prerilascio:** i processi non possono essere forzati a rilasciare in anticipo le risorse acquisite

## Attesa circolare

- almeno due processi
- ogni processo aspetta una risorsa occupata da un altro processo in attesa circolare

*Condizioni necessarie ma non sufficienti*

*Condizioni necessarie e sufficienti*



*NB:*

- le prime tre condizioni derivano direttamente dalla progettazione (in molti casi sono auspicabili),
- la quarta è un evento che si può verificare e dipende dalla particolare sequenza di richieste e rilasci.

# Stallo

## Strategie per affrontare il problema dello stallo

1. **Ignorare il problema (*algoritmo dello struzzo*)**
  - Soluzione adottata dalla maggior parte dei SO inclusi Unix e Windows
  - Soluzione ragionevole se lo stallo capita raramente o evitare lo stallo risulta molto costoso
  - Degrado delle prestazioni: blocco totale del sistema e riavvio manuale
  - Compromesso tra convenienza e correttezza
2. **Prevenirlo:** progettare un SO in modo che la possibilità di avere uno stallo sia esclusa a priori tramite la negazione di una delle 4 condizioni
3. **Esclusione:** le tre condizioni necessarie sono permesse, un algoritmo verifica dinamicamente che una richiesta non produca una situazione di stallo
4. **Consentire il verificarsi dello stallo, rilevarlo e risolverlo.**

# Prevenzione dello Stallo

Classi di prevenzione: si impongono vincoli sulla richiesta delle risorse

Metodi indiretti: prevengono il verificarsi di una delle tre condizioni necessarie

Metodi diretti: prevengono il verificarsi dell'attesa circolare

- Mutua Esclusione
  - Un processo non deve mai attendere una risorsa condivisibile
  - Devono essere possibili accessi multipli contemporanei alle risorse condivise
  - Non applicabile....
- Hold and Wait
  - Ogni processo deve richiedere all'inizio della sua esecuzione tutte le risorse
  - Il processo entra in blocked fino a quando tutte le richieste non vengono soddisfatte contemporaneamente
  - Approccio inefficiente:
    - Prima di andare in run tutte le risorse devono essere disponibili, in realtà potrebbe procedere utilizzandone solo una parte
    - Le risorse assegnate al processo potrebbero rimanere inutilizzate per molto tempo, gli altri processi sono in attesa indefinita

# Prevenzione dello Stallo

- Assenza di prerilascio  
Due possibilità:
  1. Un processo che non riesce ad ottenere le risorse di cui necessita, rilascia quelle che detiene per richiederle nuovamente in un istante successivo
  2. OS può richiedere il pre-rilascio delle risorse al processo che le detiene per assegnarle al richiedente
    - Approccio applicabile solo se lo stato della risorsa è facilmente salvabile (CPU)
    - Non applicabile nel caso di stampanti....
- Attesa Circolare (metodi diretti)
  - Si definisce un ordine lineare (di numerazione) per tutte le risorse
  - Se un processo richiede una risorsa R, successivamente potrà richiedere solo una risorsa che nell'ordinamento segue R
  - Es.:  $R_i$  precede  $R_j$  se  $i < j$ , un processo potrà chiedere le risorse solo nell'ordine  $R_i, R_j$
  - Questo metodo può essere inefficiente

# Esclusione dello Stallo DeadLock Avoidance

Esclusione:

- le tre condizioni necessarie

- Mutua esclusione
- Possesso e attesa
- Assenza di pre-rilascio

sono permesse,

- un algoritmo verifica dinamicamente che una richiesta non produca una situazione attesa circolare

L'esclusione permette più concorrenza rispetto alla prevenzione

Approcci per escludere (evitare) lo stallo:

1. **Non avviare un processo** le cui richieste possono provocare lo stallo
2. **Non concedere ulteriori risorse** ad un processo se la loro allocazione può portare ad uno stallo

# DeadLock Avoidance: Rifiuto del permesso di esecuzione

$n$  – processi,  $m$  – tipi di risorse

Risorse =  $(R_1, R_2, \dots, R_m)$  Numero di istanze per ogni risorsa

Disponibili =  $(V_1, V_2, \dots, V_m)$  Numero di istanze disponibili per ogni risorsa

$$Richieste = \begin{pmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{pmatrix}_{n \times m}$$

$C_{ij}$  – richieste complessive da parte del processo  $i$  sulla risorsa  $j$

$$Assegnazioni = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{pmatrix}_{n \times m}$$

$A_{ij}$  – assegnazioni correnti

Risulta:

$$R_i = V_i + \sum_{k=1}^n A_{ki}, \quad \forall i$$

$$C_{ki} \leq R_i, \quad \forall k, i$$

$$A_{ki} \leq C_{ki}, \quad \forall k, i$$

Un processo  $P_{n+1}$  può essere avviato se e solo se:  $R_i \geq C_{(n+1)i} + \sum_{k=1}^n C_{ki}, \quad \forall i$

Il processo può essere avviato se e solo se è possibile (per ogni risorsa) soddisfare la sua richiesta e quelle dei processi correnti.

Strategia non ottimale: si suppone che i processi necessitino delle risorse nello stesso istante.

Difficoltà: le richieste devono essere note all'avvio dei processi...

Info dichiarate all'inizio

# DeadLock Avoidance: Rifiuto di allocazione delle risorse

- Stato del sistema: assegnazione corrente di risorse ai processi, si costituisce delle matrici:
  - *Risorse*
  - *Disponibili*
  - *Richieste*
  - *Assegnazioni*

Lo stato in cui si trova il sistema può essere:

- SICURO: esiste almeno una sequenza di esecuzione dei processi che ne consente la terminazione senza incorrere in una situazione di stallo
- NON SICURO

**Quando un processo richiede una risorsa disponibile, il sistema :**

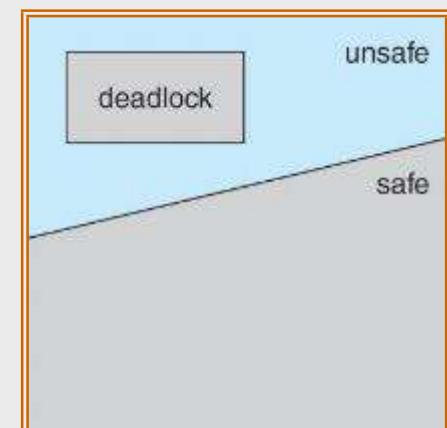
- Simula l'assegnazione delle risorse e l'aggiornamento dello stato**
- Verifica che il nuovo stato sia uno stato sicuro**
- In caso positivo assegna le risorse,**
- In caso negativo rifiuta la richiesta e il processo viene bloccato fino a che la risorsa non si rende disponibile**

# Stato Sicuro

Osservazioni:

- Se le risorse richieste da  $P_i$  non sono immediatamente disponibili, allora  $P_i$  deve attendere la terminazione di un processo che rilascia un numero sufficiente di risorse.
- Quando un processo  $P_j$  termina,  $P_j$  può ottenere le risorse necessarie all'esecuzione e terminare a sua volta rilasciando le risorse detenute.
- Stato sicuro  $\Rightarrow$  no deadlocks.
- Stato non sicuro  $\Rightarrow$  possibilità di deadlock.
- Avoidance  $\Rightarrow$  assicurarsi che il sistema non entri in uno stato non sicuro.

Uno stato non sicuro non rappresenta uno stato di stallo, ma uno stato in cui lo stallo è possibile. La strategia di esclusione dello stallo non si basa sulla certezza che uno stato sia di stallo, ma sulla possibilità che uno stato non sicuro determini uno stallo.



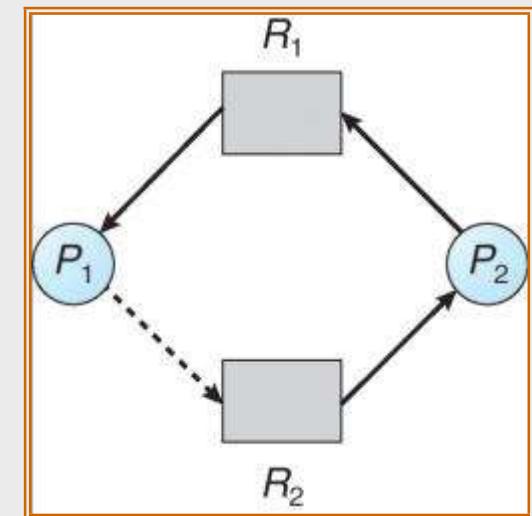
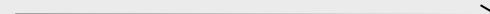
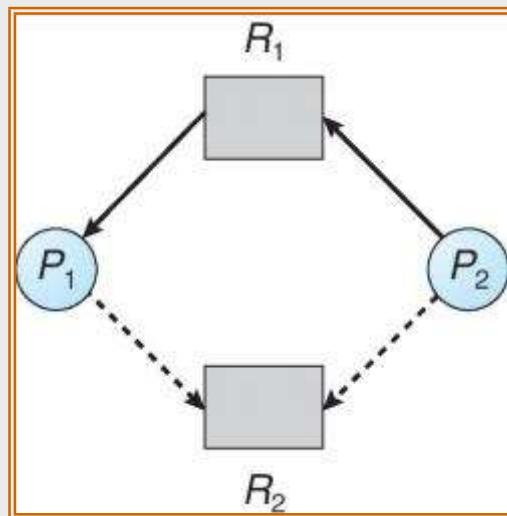
# Esclusione dello stallo

## Grafo di assegnazione delle risorse

Singola istanza per ciascuna risorsa.

- Arco di reclamo:  $P_i \rightarrow R_j$  indica che il processo  $P_j$  può richiedere la risorsa  $R_j$ ; linea tratteggiata.
- L'arco di reclamo diventa un arco di richiesta quando la risorsa viene effettivamente richiesta.
- L'arco di richiesta diventa di assegnazione quando la risorsa viene effettivamente assegnata.
- Le risorse devono essere reclamate a priori nel sistema.
- Si supponga che  $P_1$  richieda  $R_1$

La richiesta può essere garantita solo se la conversione di un arco di richiesta in un arco di assegnazione non comporta la presenza di un ciclo nel grafo di assegnazione delle risorse (stato non sicuro).



# Logica di esclusione dello stallo Algoritmo del Banchiere

Istanze multiple per le risorse.

```
struct state
{
    int resource[m];
    int available[m];
    int claim[n][m];
    int alloc[n][m];
}
```

*stato del sistema*

```
(a) global data structures
if (alloc [i,*] + request [*] > claim [i,*])
    < error >;
else if (request [*] > available [*])
    < suspend process >;
else
{
    < define newstate by:
    alloc [i,*] = alloc [i,*] + request [*];
    available [*] = available [*] - request [*] >;
}
if (safe (newstate))
    < carry out allocation >;
else
{
    < restore original state >;
    < suspend process >;
}
```

*(b) resource alloc algorithm*

*Si verifica che la somma tra le risorse attualmente assegnate e richieste non superi la dichiarazione iniziale*

*Se non ci sono risorse sufficienti, il processo viene sospeso*

*Si definisce il nuovo stato ipotizzando di assegnare le risorse al processo richiedente*

*Si verifica se il nuovo stato è sicuro*

# Logica di esclusione dello stallo

## Algoritmo del Banchiere

Algoritmo per la determinazione di stato sicuro/nonsicuro

```
boolean safe (state S)
{
    int currentavail[m];
    process rest[<number of processes>];
    currentavail = available;
    rest = {all processes};
    possible = true;
    while (possible)
    {
        <find a process Pk in rest such that
        claim [k,*] - alloc [k,*] <= currentavail;>
        if (found)                                /* simulate execution of Pk */
        {
            currentavail = currentavail + alloc [k,*];
            rest = rest - {Pk};
        }
        else
            possible = false;
    }
    return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

# Algoritmo del Banchiere: esempio

- 5 processi:  $P_0, \dots, P_4$ ; 3 tipi di risorse: A (10 istanze), B (5 istanze), C (7 istanze).
- Snapshot a  $T_0$ :

	<u>alloc[5,3]</u>	<u>claim[5,3]</u>	<u>available[1,3]</u>	<u>request=claim-alloc</u>
	A B C	A B C	A B C	A B C
$P_0$	0 1 0	7 5 3	3 3 2	7 4 3
$P_1$	2 0 0	3 2 2		1 2 2
$P_2$	3 0 2	9 0 2		6 0 0
$P_3$	2 1 1	2 2 2		0 1 1
$P_4$	0 0 2	4 3 3		4 3 1

- If  $\text{request} > \text{available}$  suspend process:
  - $P_0, P_2, P_4$  vengono sospesi
- Per  $P_1$  risulta  $\text{Request} \leq \text{Available}$  (that is,  $(1,2,2) \leq (3,3,2) \Rightarrow \text{true}$ ).

	<u>alloc</u>	<u>request</u>	<u>available</u>
	A B C	A B C	A B C
$P_0$	0 1 0	7 4 3	2 1 0
$P_1$	3 2 2	0 0 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	

- Per determinare se questo è uno stato sicuro si verifica se altri processi, dopo la terminazione di  $P_1$  e il rilascio delle risorse potranno essere completati.

# Considerazioni sulla logica di esclusione dello stallo

Vantaggi:

- Non'è necessario interrompere processi e riportarli in uno stato precedente (problema presente nelle strategie di rilevamento dello stallo)

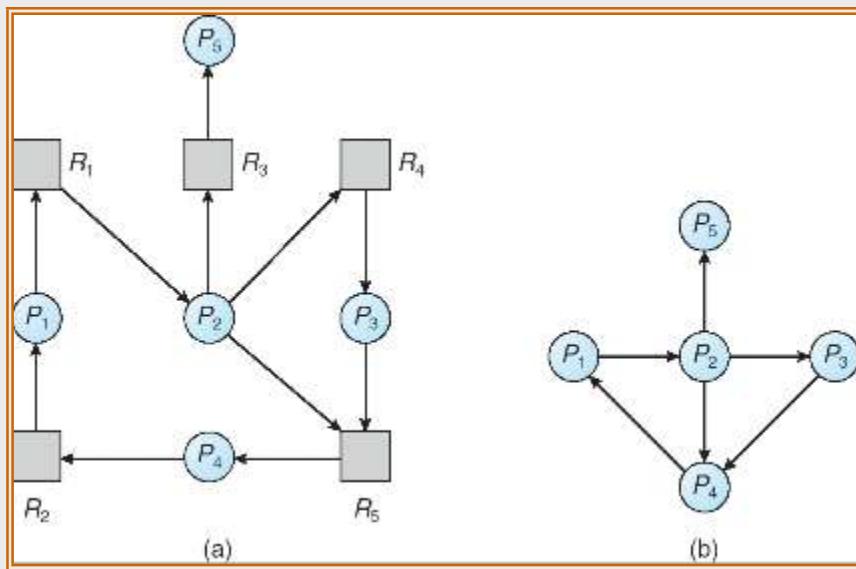
Restrizioni:

- Il numero massimo di risorse necessitate da ogni processo deve essere noto a priori (prima dell'esecuzione)
- I processi devono essere indipendenti: l'ordine di esecuzione non deve essere vincolato a esigenze di sincronizzazione
- Deve esserci un numero fissato di risorse da allocare
- Quando un processo richiede risorse potrebbe essere posto in attesa
- Quando un processo ottiene tutte le risorse di cui necessita, deve rilasciarle in un tempo finito.

# Deadlock Detection: Rilevamento dello stallo

1. Quando è possibile le richieste vengono sempre soddisfatte
2. Il SO verifica periodicamente che non si sia verificato uno stallo
3. Il SO applica uno schema di recovery per risolvere lo stallo

**Individuare lo stallo con una sola risorsa di ogni tipo**



- Un ciclo nel grafo di attesa denota uno stallo
- L'algoritmo può essere invocato periodicamente
- Numero di operazioni pari a  $n^2$  dove  $n=\text{numero di processi}$

# Deadlock Detection: Rilevamento dello stallo

## Individuare lo stallo con più istanze per ogni risorsa

Consideriamo le matrici:

- *Available*: vettore di lunghezza  $m$ : ogni elemento indica il numero di istanze disponibili per ciascuna risorsa in un dato istante
- *Allocation*: matrice  $n \times m$ : ogni elemento indica il numero di istanze di ciascuna risorsa assegnate a ciascun processo
- *Request*: matrice  $n \times m$ : ogni elemento indica la richiesta di risorse da parte dei processi.

L'algoritmo di rilevamento indaga su ogni possibile sequenza di assegnazione per i processi che devono ancora essere completati. Vengono marcati i processi che non sono in stallo.  
Inizialmente nessun processo è marcato.

1. Si marca ogni processo che ha una riga di 0 nella matrice *Allocation* (il processo non ha risorse assegnate);
2. Si inizializza  $Work = Available$ ;
3. Si cerca un indice  $i$  (processo) tale che:
  1.  $i$  non è marcato (può essere in stallo);
  2.  $Request[i,:] \leq Work$ ;

Se tale processo non esiste l'algoritmo termina
4. Se si trova il processo al punto 3. esso viene marcato e  $Work = Work + Allocation[i,:]$

**Un processo è in stallo solo se alla fine dell'algoritmo esso non risulta marcato.**

**L'algoritmo richiede un numero di operazioni dell'ordine di  $O(m \times n^2)$**

# Deadlock Detection: Rilevamento dello stallo

Individuare lo stallo con più istanze per ogni risorsa

Filosofia dell'algoritmo:

- Trovare un processo le cui richieste possono essere soddisfatte con le disponibilità attuali
- Supporre che il processo vada a conclusione e rilasci le risorse
- Cercare un altro processo da soddisfare

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix A

	R1	R2	R3	R4	R5
	2	1	1	2	1

Resource vector

	R1	R2	R3	R4	R5
	0	0	0	0	1

Available vector

Figure 6.10 Example for Deadlock Detection

*P4 viene marcato (non detiene alcuna risorsa)*

*Work=Available= [0, 0, 0, 0, 1]*

*Request[P3,:] ≤ Work, P3 viene marcato e Work=Work+Allocation[P3,:]=[0, 0, 0, 1, 1]*

*Non è possibile determinare alcun altro indice (processo) per il quale soddisfare le richieste  
L'algoritmo termina.*

*P1, P2 non sono marcati: P1, P2 sono in stallo*

# Deadlock Detection: Rilevamento dello stallo

## Utilizzo dell'algoritmo di rilevamento dello stallo

- Quando e quanto spesso invocare l'algoritmo dipende da:
  - Frequenza presunta con la quale si verifica lo stallo
  - Numero di processi coinvolti nello stallo
- In generale uno stallo si verifica quando un processo avanza una richiesta che non può essere soddisfatta immediatamente.

UTILIZZO DELL'ALGORITMO AD OGNI RICHIESTA. Consente la determinazione dello stallo e del processo la cui richiesta ha cagionato lo stallo.  
Aumento notevole del carico... (overhead)
- INVOCARE L'ALGORITMO QUANDO LA PERCENTUALE DI UTILIZZO DELLE RISORSE SCENDE AL DISOTTO DI UNA SOGLIA
- INVOCARE L'ALGORITMO AD ISTANTI ARBITRARI: nel grafo di assegnazione delle risorse potrebbero esistere molti cicli e diventa difficile determinare quale processo ha “causato” lo stallo...

# Deadlock Detection: Ripristino

Terminazione dei processi:

- Abort di tutti i processi coinvolti nello stallo. Soluzione più comunemente adottata.
- Abort di un processo alla volta fino a quando il ciclo non'è eliminato. Dopo ogni abort si deve rieseguire l'algoritmo di determinazione dello stallo. Ordine con cui abortire i processi dato da funzione di minimo costo basata su
  - Priorità dei processi
  - Tempo trascorso in esecuzione e necessario al completamento
  - Risorse già utilizzate e risorse richieste (processo in fase di stampa...)
  - Tipo di processo (interattivo, ecc.)

Prelazione di risorse:

- Selezionare un processo vittima – minimize cost.
- Rollback del processo a cui è stata sottratta la risorsa – return ad uno stato sicuro per poterlo riavviare in seguito.

*Salvataggio periodico dello stato dei processi. In caso di stallo:*

*Si ripristina il processo all'ultimo stato salvato*

*Si fa ripartire il processo...il non-determinismo dei processi concorrenti dovrebbe garantire il non ri-verificarsi dello stallo*

In generale conviene uccidere il processo.

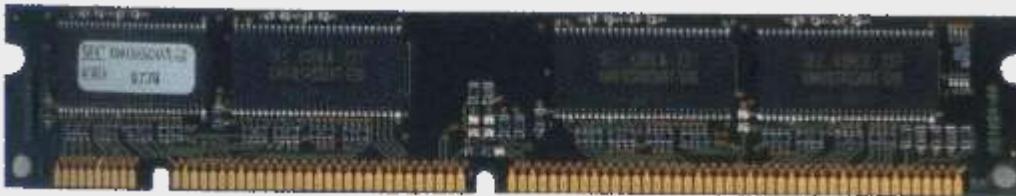
- Starvation – alcuni processi potrebbero essere selezionati costantemente come vittime, include number of rollback in cost factor.

# Strategie a confronto

**Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance  
Approaches for Operating Systems [ISLO80]**

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	<ul style="list-style-type: none"><li>• Works well for processes that perform a single burst of activity</li><li>• No preemption necessary</li></ul>	<ul style="list-style-type: none"><li>• Inefficient</li><li>• Delays process initiation</li><li>• Future resource requirements must be known by processes</li></ul>
		Preemption	<ul style="list-style-type: none"><li>• Convenient when applied to resources whose state can be saved and restored easily</li></ul>	<ul style="list-style-type: none"><li>• Preempts more often than necessary</li></ul>
		Resource ordering	<ul style="list-style-type: none"><li>• Feasible to enforce via compile-time checks</li><li>• Needs no run-time computation since problem is solved in system design</li></ul>	<ul style="list-style-type: none"><li>• Disallows incremental resource requests</li></ul>
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	<ul style="list-style-type: none"><li>• No preemption necessary</li></ul>	<ul style="list-style-type: none"><li>• Future resource requirements must be known by OS</li><li>• Processes can be blocked for long periods</li></ul>
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	<ul style="list-style-type: none"><li>• Never delays process initiation</li><li>• Facilitates on-line handling</li></ul>	<ul style="list-style-type: none"><li>• Inherent preemption losses</li></ul>

# Gestione della Memoria Centrale



Un programma e i suoi dati al fine di essere eseguiti (run) devono essere, almeno parzialmente, presenti in memoria centrale

## **Il gestore della memoria:**

- si occupa di allocare memoria fisica ai processi,
- in un ambiente multiprogrammato deve garantire la protezione dei dati, impedendo ai processi attivi di sconfinare nello spazio di indirizzamento di altri processi,
- In un ambiente multiprogrammato deve permettere meccanismi di condivisione affinché i processi cooperanti possano accedere ad aree comuni di memoria.

L'utilizzo globale delle risorse e le prestazioni di un calcolatore vengono influenzate dalle prestazioni del *modulo di gestione della memoria* sia in funzione della sua efficienza nell'allocare memoria, sia per l'influenza che può avere sullo *scheduler*.

# Introduzione

La cpu preleva le istruzioni dalla memoria centrale sulla base del contenuto del PC.

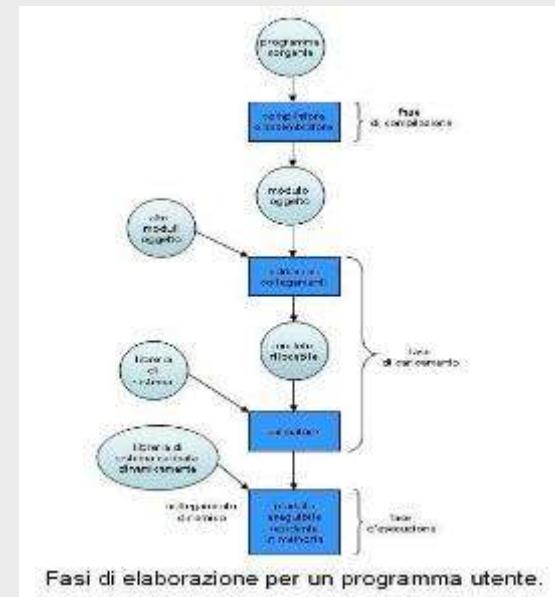
Il programma prima di essere eseguito attraversa diversi stadi (alcuni opzionali):

- programma sorgente con indirizzi **simbolici** (es.: *a, x, somma, contatore*)
- Compilazione: generazione di indirizzi **rilocabili**,
- Caricatore (loader) fa corrispondere a questi indirizzi gli indirizzi **assoluti**, dopo l'intervento dell'editor dei collegamenti “linkage editor”

Modulo sorgente - indirizzi simbolici ( es. : *x*)

Modulo compilato - indirizzo rilocabile (es.: +14 bytes dall'inizio di questo modulo)

Modulo Eseguibile- indirizzo assoluto ( es.:  $74000+14=74014$ )



## Generazione di indirizzi di memoria (assoluti) a tempo di:

**COMPILAZIONE** Si genera codice assoluto che deve risiedere in memoria (es. nell'MS-DOS file .COM).

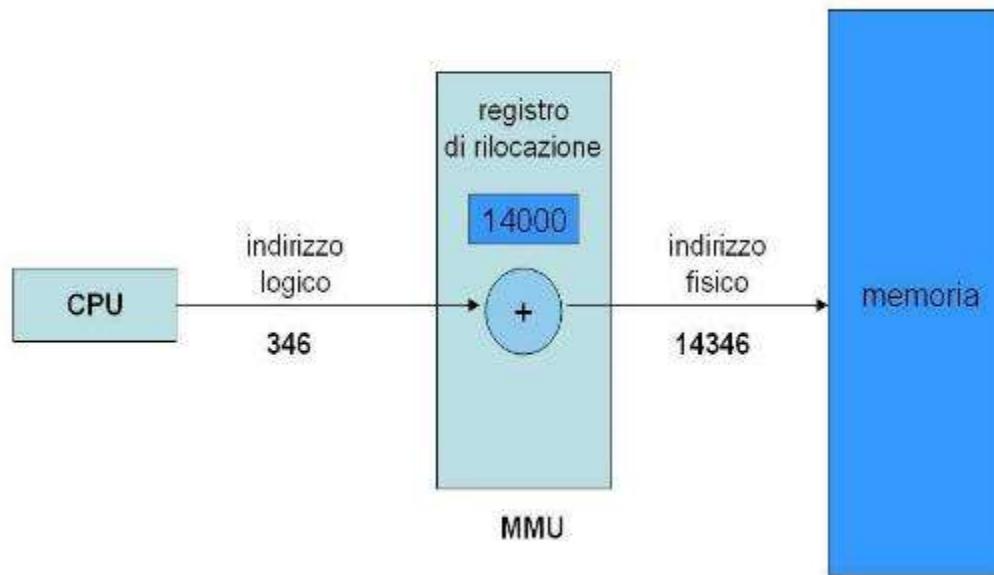
**CARICAMENTO** Se non si sa dove deve risiedere viene generato codice rilocabile, in fase di caricamento si fa l'associazione agli indirizzi assoluti.

**ESECUZIONE** Se durante l'esecuzione il processo può essere spostato da un segmento di memoria ad un altro, si deve ritardare l'associazione degli indirizzi fino alla fase di esecuzione.

# Introduzione

- ↓ Indirizzo Logico (virtuali): indirizzo generato dalla cpu
- ↓ Indirizzo Fisico: indirizzo reale (MAR) visto dalla memoria

**MMU ( Memory Management Unit):** modulo che si occupa della traduzione.



Rilocazione dinamica tramite un registro di rilocazione.

# Allocazione della Memoria

I moduli di gestione della memoria si differenziano in base al tipo di allocazione della stessa, che può essere:

**CONTIGUA**

**NON CONTIGUA**

**Allocazione Contigua:** la memoria viene allocata in modo tale che ciascun oggetto occupa un insieme di locazioni i cui indirizzi sono strettamente consecutivi. Esistono vari tipi di allocazione contigua:

**MONOALLOCAZIONE**

**PARTIZIONAMENTO STATICO**

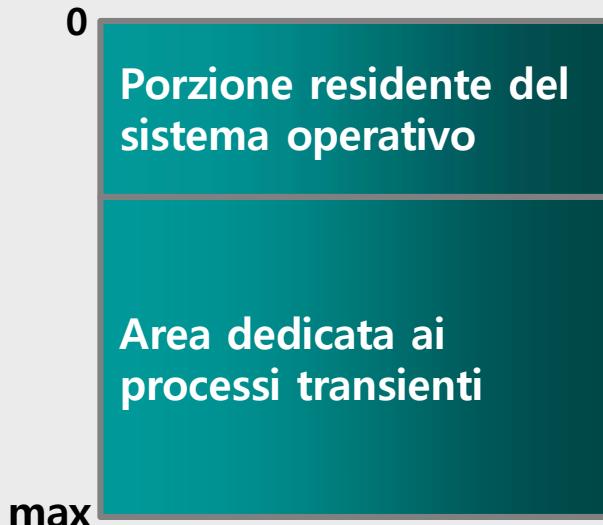
**PARTIZIONAMENTO DINAMICO**

**SEGMENTAZIONE**

# Monoallocazione (Monitor Monoprocesso)

La memoria viene suddivisa in due aree contigue:

- la prima allocata permanentemente ad una porzione residente del sistema operativo (Monitor)
- la seconda assegnata ai processi in esecuzione, processi utente o porzioni non residenti del sistema operativo, per il tempo necessario al completamento.



Il sistema operativo:

- tiene traccia della prima e dell'ultima locazione disponibile per i processi transienti;
- viene posto ad uno dei due estremi della memoria;
- ingloba i vettori di interrupt;

Schema viene generalmente utilizzato da sistemi operativi monoprocesso per microcomputer.

# Monoallocazione (Monitor Monoprocesso)

Alla richiesta di esecuzione di un programma, il sistema operativo:

- si assicura che le dimensioni del processo siano compatibili con la memoria disponibile;
- conferisce il controllo al processo utente fino al suo completamento o ad eventuali condizioni di errore;
- al termine del processo la memoria viene liberata e può essere assegnata ad un altro processo in attesa

Raramente un monitor monoprocesso supporta meccanismi di protezione tra processi utente in quanto in ogni istante vi può essere al massimo un solo processo residente in memoria. Gli eventuali meccanismi si riferiscono alla protezione del codice del sistema operativo da eventuali sconfinamenti del processo transiente in esecuzione.

La protezione del sistema operativo dai processi utente è spesso effettuata mediante supporti hardware:

- ***Registri Barriera;***
- ***Diritti di accesso mediante bit di protezione;***
- ***Sistema operativo in memoria a sola lettura.***

# Monoallocazione

## Meccanismi di protezione

**Registri barriera:** usati per tracciare un confine tra le aree dei processi di sistema e dei processi transienti. Nel registro viene memorizzata la prima locazione disponibile al processo e il sistema operativo effettua i controlli di sconfinamento.

**Diritti di accesso mediante bit di protezione:** Ad ogni parola di memoria viene associato un bit di protezione (1 nelle zone che contengono il sistema operativo, 0 nelle restanti). I processi utente accedono solo a parole con bit 0, mentre il sistema operativo ha accesso illimitato.

**Sistema operativo in memoria a sola lettura:** non è solitamente utilizzato per la sua scarsa flessibilità e impossibilità di correggere o aggiornare il codice del sistema operativo.

**Condivisione del codice e dei dati:** non ha molto senso negli ambienti monoprocesso e raramente viene supportata. Tuttavia programmi utente possono condividere dati mediante accordi interni, ponendoli in locazioni di memoria che non vengono sovrascritte durante l'esecuzione di processi partecipanti, o mediante file.

# Monoallocazione Prestazioni

## Difetti:

- Assenza di multi-programmazione: abbassamento dell'efficienza della memoria e della CPU;
- La memoria può risultare sovradimensionata per la maggioranza dei processi;
- Processi di dimensione più grande della memoria non possono essere eseguiti, oppure richiedono particolari suddivisioni del codice (overlay);
- I programmi tendono ad essere ottimizzati rispetto alla dimensione della memoria, comportando spesso sacrifici di funzionalità e velocità.

## Pregi:

- Basso costo di progettazione del modulo di gestione della memoria;
- Contenuti supporti hardware specifici per la gestione della memoria;

Utilizzato per micro-computer dedicati permanentemente a specifiche applicazioni.

# Partizionamento Statico

Supportare la multiprogrammazione:

dividere la memoria in diverse partizioni, ciascuna delle quali può essere allocata a un processo diverso.

Partizionamento statico:

la suddivisione della memoria viene fatta “fuori linea”

le partizioni hanno dimensioni fisse

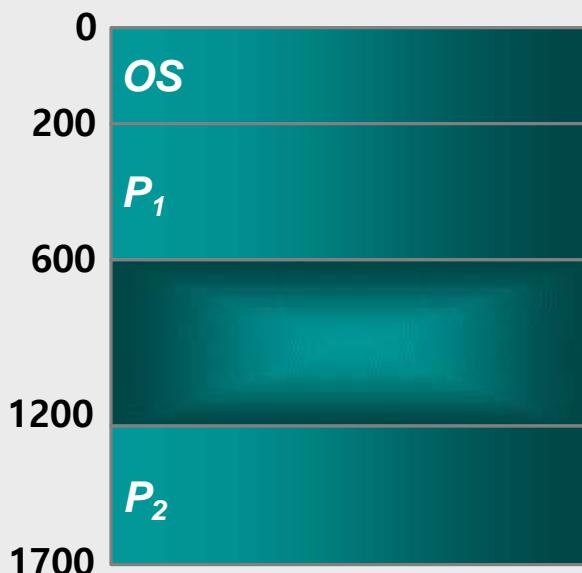
Il numero e la dimensione di ciascuna partizione viene determinato durante la generazione del sistema considerando:

- capacità della memoria fisica disponibile;
- grado desiderato di multiprogrammazione;
- dimensioni dei processi più frequentemente eseguiti.

# Partizionamento Statico

Il sistema operativo deve tener traccia delle partizioni definite. Lo stato corrente delle partizioni ed i loro attributi vengono raccolti in una struttura dati chiamata **tabella di descrizione delle partizioni** (TDP).

L'unico campo variabile nella TDP è lo stato della partizione che indica in ogni istante se la partizione è allocata o no. Gli altri campi contengono valori definiti al momento del partizionamento.



Numero partizione	Base partizione	Dimensione partizione	Stato partizione
0	0 kB	200 kB	Allocata
1	200 kB	400 kB	Allocata
2	600 kB	600 kB	Libera
3	1200 kB	500 kB	Allocata

*Esempio di TDP*

# Partizionamento Statico

## Caricamento di un processo:

SO consulta la TDP. Se il test è positivo, il campo “stato della partizione” viene impostato ad “Allocata” e l’immagine del processo viene caricata nella corrispondente partizione. Viene aggiornato il campo nel PCB.

Metodi più comuni per la ricerca di una partizione libera:

- **First fit**: allocare la prima partizione libera sufficientemente grande per contenere il processo; molto veloce.
- **Best fit**: allocare la più piccola delle partizioni libere che contenga il processo; prima dell’allocazione devono essere controllate tutte le locazioni libere; ottimizzazione dello spazio di memoria, l’algoritmo produce la più piccola frammentazione interna.
- **Worst fit**

## Terminazione di un processo:

Deallocazione della partizione.

Aggiornamento della TDP.

Il funzionamento del gestore della memoria e dello scheduler sono strettamente correlati:

- lo scheduler determina quali processi sono pronti, e quindi residenti in memoria, in attesa della CPU;
- il gestore della memoria può decidere di “sfrattare” un processo in memoria per liberare una partizione.

# Partizionamento Statico

Cause di mancata disponibilità di una partizione:

- 1) non vi sono partizioni di memoria sufficientemente grandi per contenere il processo entrante;

Cause:

- errore di scelta della dimensione delle partizioni;
- Processo eccezionalmente più grande della norma.

Soluzione:

la dimensione delle partizioni deve essere ridefinita, oppure il programma d'origine deve essere progettato utilizzando uno schema di *overlay*.

- 2) tutte le partizioni sono allocate;

Soluzioni:

- attesa di una partizione libera;
- *swapping* di un processo.

# Partizionamento Statico

**Swapping:** rimozione dalla memoria di un processo sospeso e caricamento di un nuovo processo.

1. Lo scheduler decide l'introduzione in memoria di un nuovo processo
2. Il gestore dello swapping ne sceglie uno da rimuovere in base a:
  - dimensione della partizione richiesta;
  - priorità del processo;
  - tipo di evento atteso dal processo;
  - tempo già trascorso in memoria.

Un processo residente in memoria, già parzialmente eseguito, comprende:

- |                       |                             |
|-----------------------|-----------------------------|
| ● codice eseguibile;  | ● registri di stato;        |
| ● dati già elaborati; | ● file aperti;              |
| ● stack;              | ● descrittore del processo. |

Al momento dello swapping questi oggetti vengono registrati in un file su hard disk detto file di swapping.

# Partizionamento Statico

Tutti i processi che hanno subito uno swapping possono essere mantenuti:

- in un file unico per tutto il sistema

- creato al momento dell'inizializzazione del sistema
- collocato su una periferica veloce di memorizzazione secondaria
- L'indirizzo e la dimensione sono solitamente statici in modo da beneficiare diretto su  
di un indirizzamento  
disco
- Parametro critico: dimensione:
  - troppo grande spreca spazio su dischi veloci
  - troppo piccolo può rendere indisponibile l'operazione di swapping.

- in file separati, ciascuno associato ad un singolo processo.

- creato dinamicamente al momento della creazione del processo o staticamente al momento della preparazione del programma.

Vantaggi:

- Eliminazione del problema del dimensionamento del file unico;
- Nessuna restrizione sul numero di processi attivi.

Svantaggi:

- Maggior spreco di spazio;
- Maggiori tempi di accesso ai file distribuiti sul disco.

# Partizionamento Statico

Uno swapping efficiente richiede che i processi siano *rilocabili dinamicamente*:

- il processo può iniziare l'esecuzione in qualunque area di memoria;
- può essere spostato in un'altra area di memoria;
- il calcolo degli indirizzi viene effettuato dinamicamente ad esempio con l'uso di un *registro base*.

Processi non rilocabili dinamicamente sono legati alle partizioni in cui iniziano l'esecuzione e rendono inefficiente lo swapping.

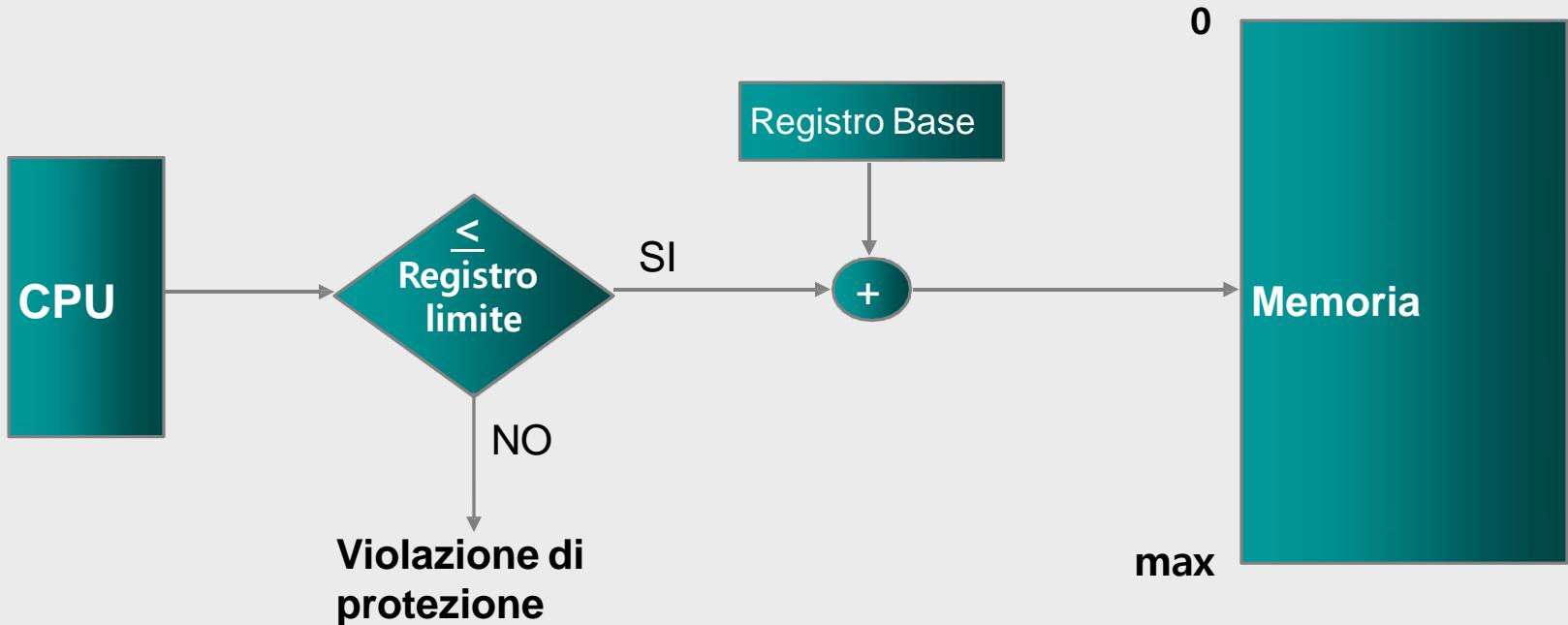
L'integrità di un sistema multiprogrammato dipende anche dalla sua capacità di garantire l'isolamento tra spazi di indirizzi separati.

- Registro base (per la rilocazione)
- Registro limite (per la protezione)

# Partizionamento Statico: protezione

Il registro base contiene l'indirizzo più basso del processo.

Il registro limite contiene il valore dell'indirizzo virtuale più alto contenuto dal programma.



# Partizionamento Statico: Condivisione

*Condivisione controllata di dati e codice tra processi cooperanti.*

I sistemi a partizioni fisse basano i propri meccanismi sull'isolamento degli spazi di indirizzamento => difficoltà di condivisione

Metodi di condivisione:

1. Affidamento degli oggetti condivisi al sistema operativo.

Svantaggi:

- l'area del sistema operativo dovrebbe poter variare dinamicamente;
- le routine dei processi utente devono poter essere "linkate" al sistema operativo dinamicamente.

2. Ogni processo possiede una copia identica dell'oggetto condiviso, che usa e diffondono gli aggiornamenti. Svantaggi:

- Se il sistema supporta lo swapping, uno o più processi potrebbero non essere in memoria e quindi non ricevere gli aggiornamenti

3. Collocare i dati in una partizione comune dedicata. Il sistema operativo considera come violazione i tentativi dei processi partecipanti alla condivisione di accedere a zone di memoria esterne alle rispettive partizioni. Se il sistema usa registri base e limite sono necessari accorgimenti per indirizzare partizioni che potrebbero non essere contigue.

# Partizionamento Statico: Conclusioni

Partizionamento statico:

- Metodo semplice di gestione della memoria
- Supporta la multiprogrammazione;
- Supporto hardware modesto;
- Si adegua ad ambienti statici con carico predicibile di lavoro come ambienti in cui vengono eseguiti solo applicativi, controllo di processi e sistemi di tipo bancario.

Svantaggi:

- *frammentazione interna*;
- può richiedere progettazione dei programmi con schemi ad overlay;
- non si adatta a contenere oggetti come *stack* che possono crescere;
- il numero di partizioni fissato limita il grado di multiprogrammazione.

# Partizionamento Dinamico

- Sistema delle partizioni analogo a quello del partizionamento statico.
- Il gestore della memoria crea ed assegna partizioni di *dimensione variabile dinamicamente* in base alle richieste dei processi.

## Richiesta di partizione:

- Il gestore della memoria ricerca una zona di memoria libera contigua di dimensione sufficiente.
- La partizione viene creata registrando la sua base, la sua dimensione e il suo stato (Allocata) nella TDP o in una tabella equivalente.
- L'eventuale parte rimanente di memoria libera viene restituita all'insieme della memoria libera e posta a disposizione del modulo di allocazione.

## Terminazione di un processo o swapping:

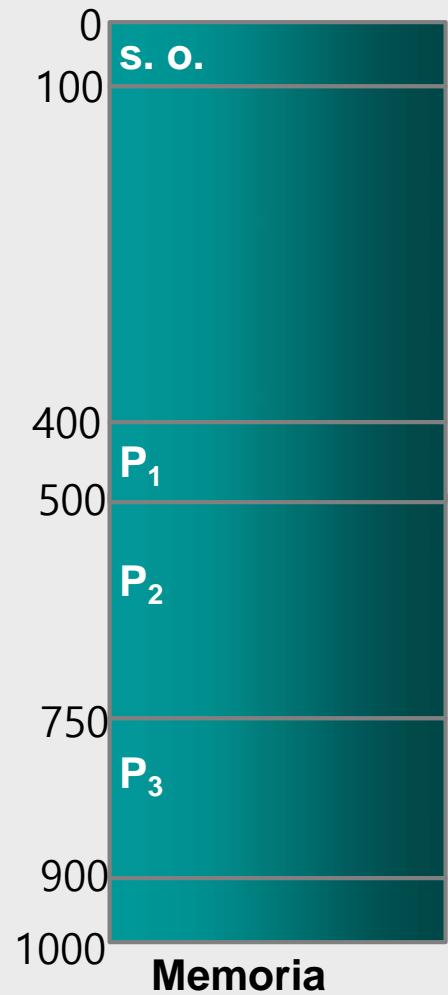
- il gestore della memoria restituisce lo spazio liberato all'insieme di aree di memoria libere e cancella la riga corrispondente nella TDP.

Il gestore della memoria può creare e allocare partizioni finché non viene esaurita la memoria fisica o finché non viene raggiunto il massimo grado di multiprogrammazione.

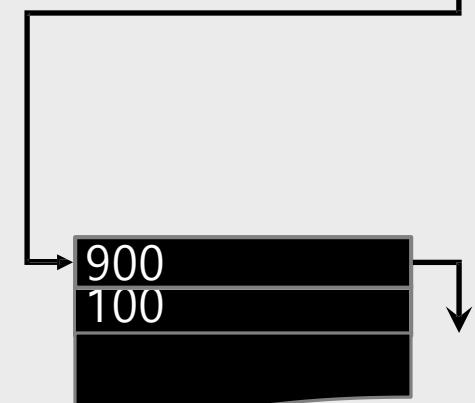
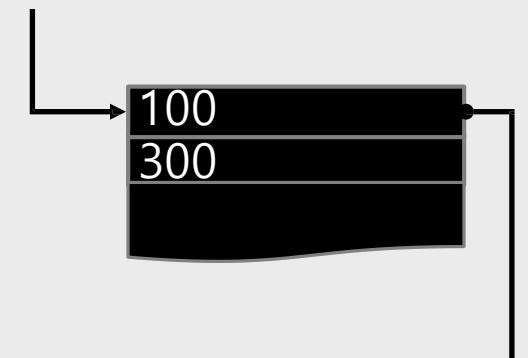
# Partizionamento Dinamico: Esempi di partizioni

0	100	Allocata
1	-	-
2	400	100
3	500	250
4	750	150
5	-	-
6	-	-
7	-	-

TDP



Puntatore alla prima  
area libera



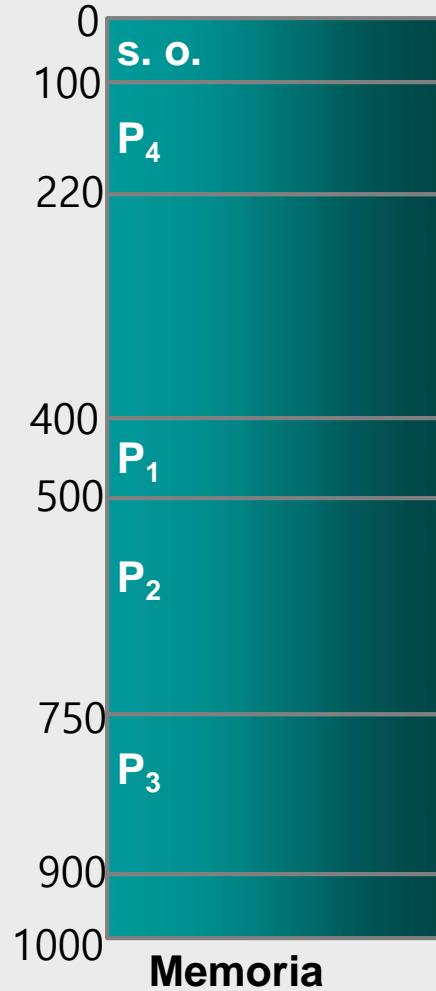
**Lista aree libere**

# Partizionamento Dinamico: Esempi di partizioni

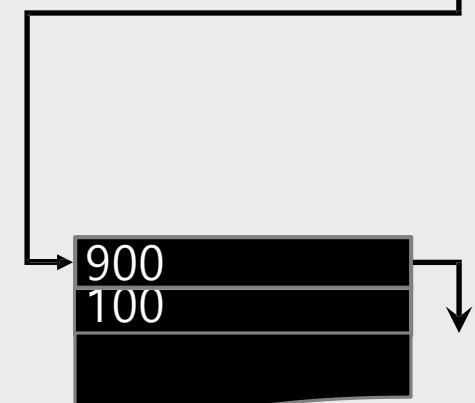
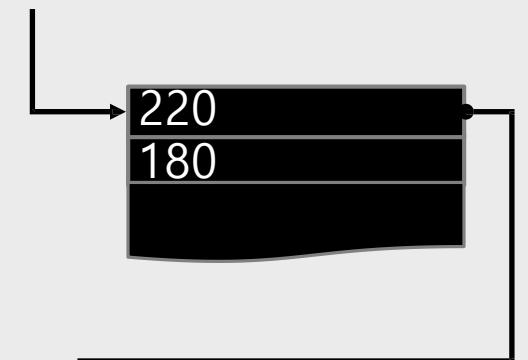
Richiesta di un'area pari a 120KB per il P4

0	100	Allocata
1	100	Allocata
2	400	Allocata
3	500	Allocata
4	750	Allocata
5	-	-
6	-	-
7	-	-

TDP



Puntatore alla prima  
area libera

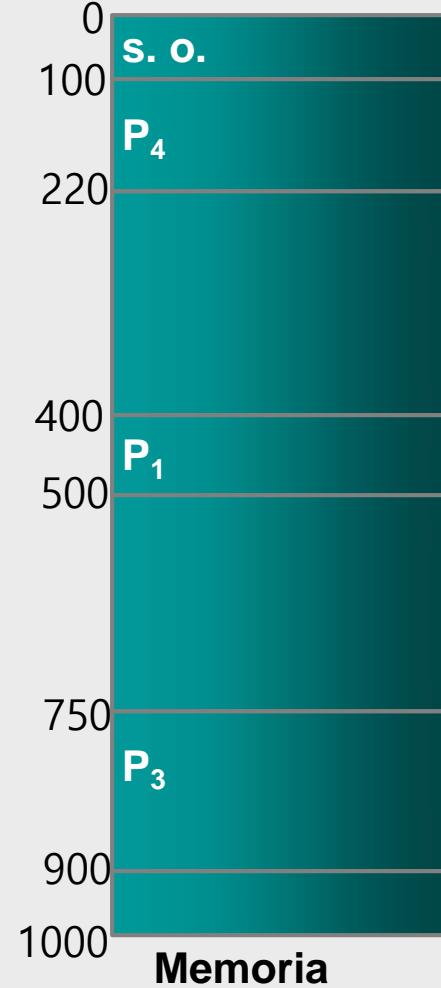


**Lista aree libere**

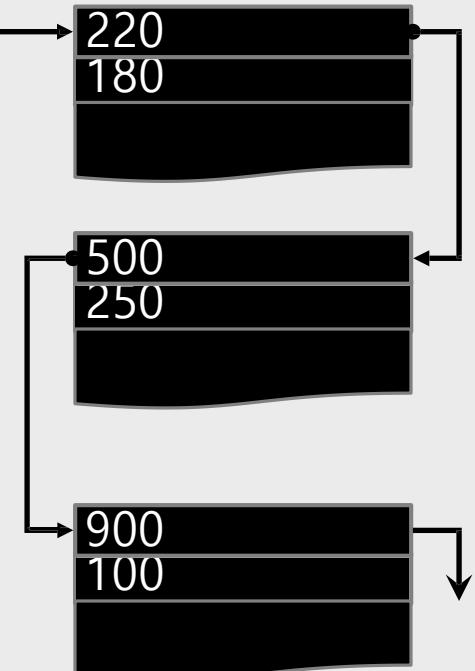
# Partizionamento Dinamico: Esempi di partizioni

La partizione occupata da P2 viene liberata

TDP	Indirizzo	Stato
0	100	Allocata
1	100	Allocata
2	400	Allocata
3	-	-
4	750	Allocata
5	-	-
6	-	-
7	-	-



Puntatore alla prima  
area libera



Lista aree libere

# Partizionamento Dinamico

Algoritmi per la selezionare un'aria di memoria:

- **First-fit.** termina la ricerca quando viene individuato il primo blocco di memoria sufficientemente grande da contenere il processo.
  - tempo di ricerca molto veloce
- **Next-fit.** simile al first fit, ma il puntatore alla lista delle aree libere, dopo avere effettuato un'allocazione, viene salvato cosicché la ricerca successiva continua da dove si era fermata la precedente
  - tempo di ricerca molto veloce
- **Best-fit.** viene utilizzato il blocco di memoria libero più piccolo che può contenere il processo. Vengono prodotte le parti di “buco” inutilizzate più piccole.
- **Worst-fit.** viene utilizzato il blocco di memoria libero più grande. Vengono prodotti “buchi inutilizzati” grandi.

# Partizionamento Dinamico

Qualsiasi algoritmo di selezione delle aree crea “buchi” di memoria inutilizzabile.

**FRAMMENTAZIONE ESTERNA:** creazione di aree di memoria libere di piccole dimensioni non contigue e impossibilità di allocazione di memoria richiesta da un processo, pur contenendo la memoria globalmente un'area di dimensione sufficiente.

**COMPATTAZIONE DELLA MEMORIA:** spostare i processi residenti in memoria in modo da creare partizioni libere più grandi. Questa operazione generalmente richiede un grande costo di CPU. La compattazione può essere effettuata:

1. Continuamente:

- la memoria viene compattata ogni volta che un processo libera un'area;
- la frammentazione è ridotta al minimo;
- grande costo di CPU.

2. Su necessità:

- la compattazione viene effettuata quando non si riesce ad allocare memoria ad un processo richiedente;
- un controllo preliminare verifica se il totale dell'area liberabile è sufficiente a contenere il processo.

# Partizionamento Dinamico

La compattazione può essere effettuata in due modi:

1. Spostamento selettivo incrementale:

- ricerca di una strategia ottima di movimento per compattare al meglio la memoria;
- consente di risparmiare tempo negli spostamenti;
- Poco utilizzata perché richiede un sovraccarico notevole.

2. Spostamento globale:

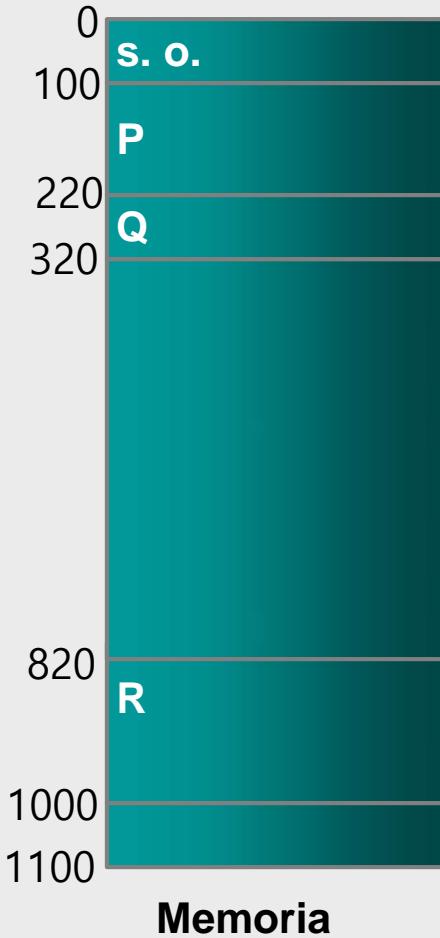
- tutte le partizioni vengono rilocate ad uno dei due estremi della memoria;
- il tempo di trasferimento delle partizioni non è ottimizzabile;
- non richiede strategie particolari.

*NB: compattazione possibile solo se la rilocazione è dinamica (run time) non possibile se la rilocazione è statica o fatta a tempo di compilazione o caricamento*

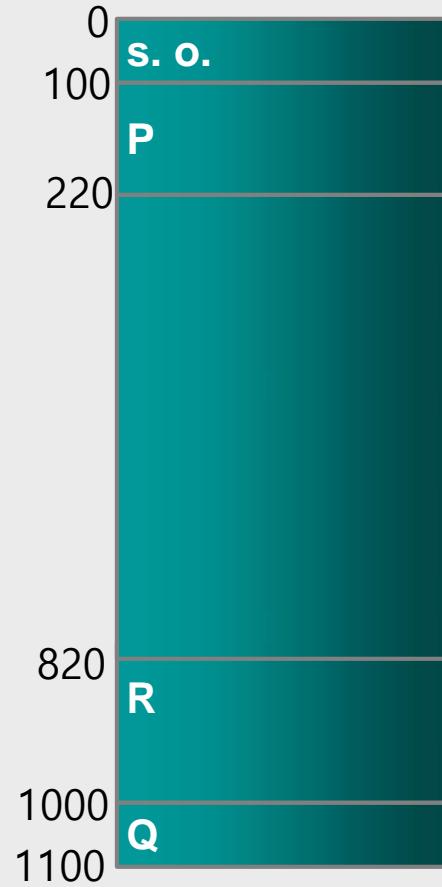
# Partizionamento Dinamico

## Esempi di compattazione

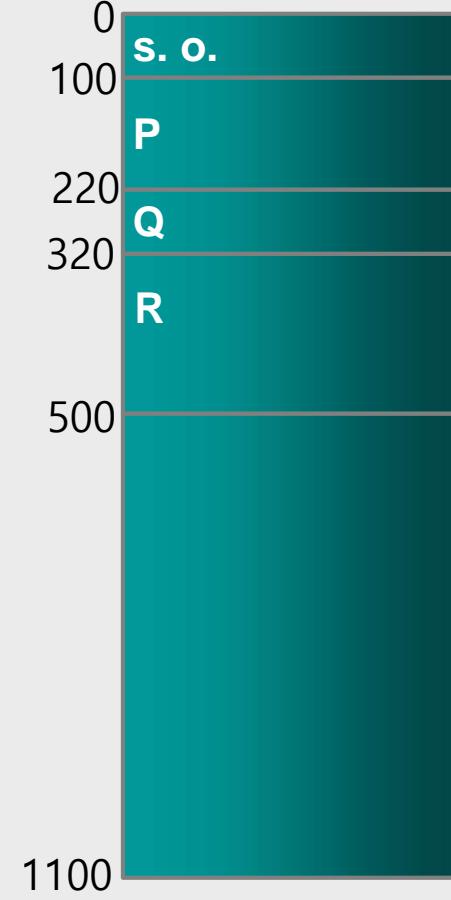
Situazione iniziale:



Spostamento selettivo



Spostamento globale



# Partizionamento Dinamico

Protezione e condivisione analoghe al partizionamento statico.

Nel partizionamento dinamico è possibile una particolare forma di condivisione:

- a due partizioni contigue è consentito sovrapporsi, mettendo un'area in comune;
- forma di condivisione molto restrittiva, può avvenire solo tra due processi.

## CONCLUSIONI

Il partizionamento dinamico:

- richiede un supporto hardware modesto, analogo al partizionamento statico;
- la differenza tra i due schemi risiede sostanzialmente nel software;
- elimina la frammentazione interna ma produce quella esterna che può essere eliminata con la compattazione;
- si adegua ad ambienti con carico non predicibile di lavoro come ad esempio sviluppo di software.

# Segmentazione

Schema di gestione della memoria che supporta la visione che l'utente ha della memoria

Un programma è una collezione di segmenti.

Un segmento è una unità logica come:

main program,  
procedure,  
function,  
method,  
object,  
local variables, global variables,  
common block,  
stack,  
symbol table, arrays

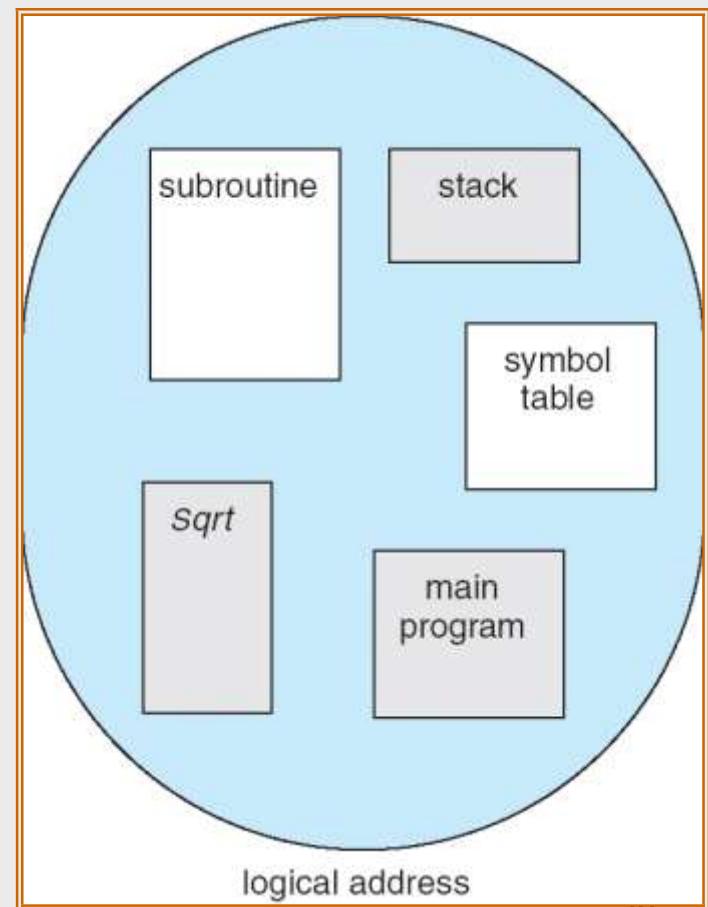
Ciascuno di questi oggetti può essere posto in un segmento diverso di dimensioni differenti.

Ogni entità (segmento) è identificata da un nome.

Le entità possono essere presenti in memoria anche in maniera non contigua.

**L'indirizzo logico si compone di due parti:**

1. il nome dell'entità identificato dal suo numero
2. lo scostamento all'interno del segmento



# Segmentazione

La segmentazione condivide alcune proprietà:

- **degli schemi di allocazione contigua** relativamente ad un singolo segmento: i dati di ogni singola entità logica devono essere posti in un'area contigua di memoria;
  - **degli schemi di allocazione non contigua** relativamente all'intero spazio di indirizzamento del processo: blocchi logici diversi possono essere messi in segmenti non contigui.
- 
- La raccolta degli oggetti in segmenti viene predisposta dal programmatore;
  - Ciascun segmento inizia all'indirizzo virtuale zero;
  - Ogni segmento ha un nome che viene poi tradotto in un indirizzo in fase di caricamento in memoria;
  - Un singolo dato all'interno del segmento viene identificato dallo "spiazzamento" relativo all'inizio del segmento cui appartiene.

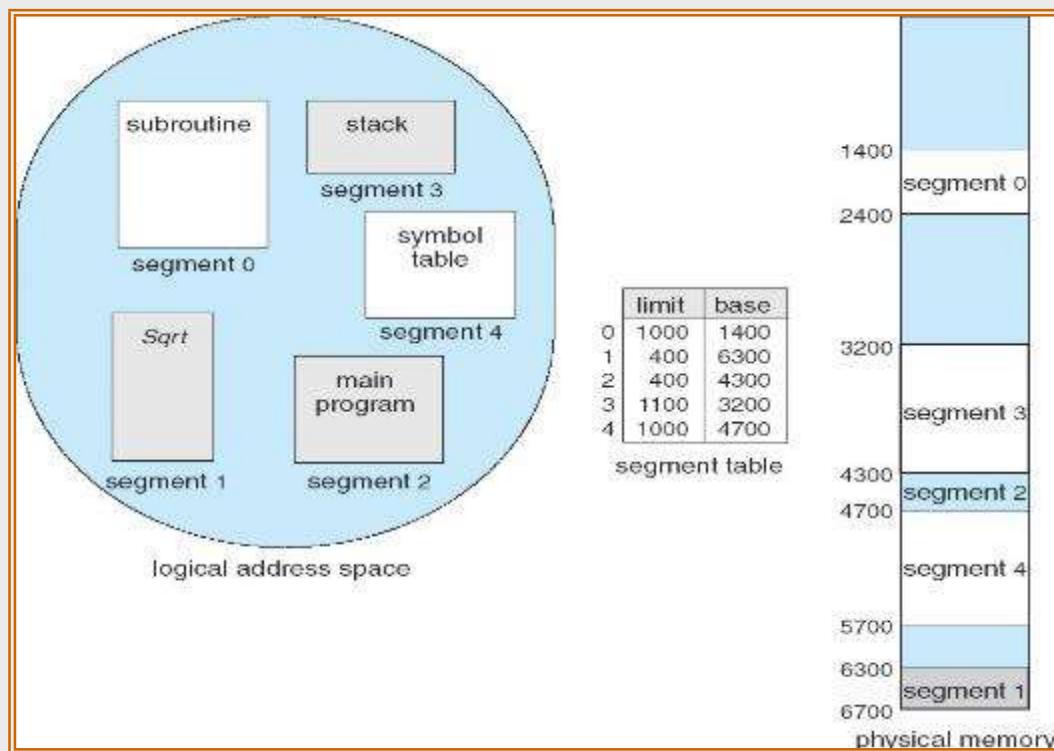
L'**indirizzamento** nella segmentazione è di tipo **bidimensionale**, la designazione univoca di un dato o di una istruzione richiede il nome del segmento e lo spiazzamento all'interno del segmento.

La memoria fisica mantiene invece l'**indirizzamento lineare**: è necessario un meccanismo per la traduzione degli indirizzi bidimensionali di segmenti virtuali in indirizzi fisici.

# Segmentazione

Caricamento di un processo segmentato:

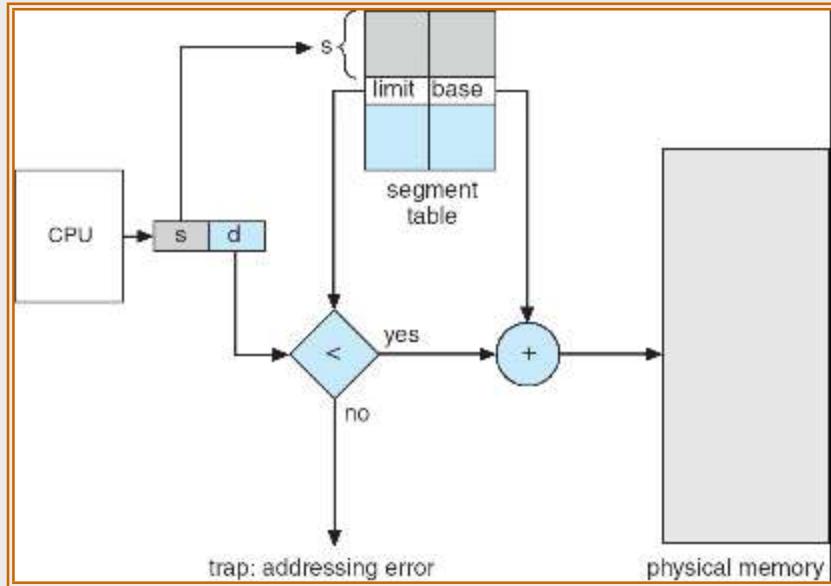
- il S.O. cerca di allocare N partizioni adatte per gli N segmenti in cui è suddiviso il processo;
- il S.O. crea un descrittore di segmento registrandovi l'indirizzo fisico in cui è stato posto il segmento e la sua dimensione;
- l'insieme dei descrittori di segmento di un processo viene raccolto nella tabella dei descrittori di segmento (segment table).



- **base** – indirizzo fisico di inizio del segmento in memoria
- **limit** – specifica la lunghezza del segmento

# Segmentazione

## TRADUZIONE DEGLI INDIRIZZI



- il *segment (s)* viene utilizzato per ritrovare l'indirizzo fisico del segmento nella segment table (base)
- Il *displacement (d)* viene sommato all'indirizzo fisico del segmento per ottenere l'indirizzo fisico richiesto

Due ulteriori registri:

- **Segment-table base register (STBR)** punta alla segment table in memoria
  - **Segment-table length register (STLR)** indica il numero di segmenti utilizzati da un programma;
- Possibilità di controllo di accesso a segmenti non assegnati ad un processo

*NB: Quando un processo subisce uno swapping, al ritorno in memoria va aggiornata la sua segment table.*

# Segmentazione

Considerazioni:

La **traduzione** di ciascun indirizzo bidimensionale virtuale richiede **due accessi in memoria**:

1. alla segment table per tradurre l'indirizzo del segmento da virtuale a fisico;
2. per accedere fisicamente alla locazione richiesta.

⇒raddoppia il tempo per l'accesso ad una locazione di memoria.

Soluzione:

*Se i descrittori di segmento sono pochi, o alcuni sono utilizzati più frequentemente, conviene specificare il loro caricamento in opportuni registri hardware.*

Famiglia INTEL : CS: Code Segment; DS: Data Segment; SS: Stack Segment; ES: Extra Segment.

E' possibile definire **diritti e modalità di accesso per ogni “tipo” di segmento**. Esempio:

- accesso in modalità “execute” (o anche “read-only”) al segmento codice;
- accesso in modalità “read/write” al segmento stack;
- accesso “read-only” o “read/write” al segmento dati.

I diritti di accesso possono essere registrati in un campo della TDS.

# Segmentazione

## CONDIVISIONE:

- Gli oggetti condivisi sono collocati in segmenti dedicati e separati.
- Il segmento condiviso può essere mappato, attraverso la segment table, nello spazio di indirizzamento virtuale di tutti i processi autorizzati ad accedervi.
- Lo spiazzamento interno di un dato risulta identico per tutti i processi che lo condividono.

## COLLEGAMENTO DINAMICO:

- Caricamento di una procedura, ad esempio una libreria, in fase di esecuzione di un processo e solo su sua richiesta.
- Lo spazio in memoria per una procedura viene occupato solo se e quando tale procedura viene richiesta.
- La segmentazione consente di aggiungere nuovi segmenti al processo richiedente, aggiornando la segment table e i registri Segment-table base register (STBR) Segment-table length register (STLR)

# Segmentazione: Conclusioni

- Eliminando la necessità di allocare processi in aree contigue si rende più efficiente la gestione della memoria fisica;
- no frammentazione interna;
- Supporto efficiente di protezione e condivisione;
- Supporto del collegamento dinamico.

## SVANTAGGI:

- Frammentazione esterna (buchi troppo piccoli per ospitare un intero segmento), necessità di effettuare comunque la compattazione (resa anche più complessa);
- il duplice accesso alla memoria deve essere supportato da hardware apposito;
- la gestione della memoria da parte del sistema operativo è più complessa rispetto a quella per il partizionamento statico e dinamico;
- non rende ancora possibile l'esecuzione di processi più grandi delle dimensioni fisiche della memoria.

# Allocazione non contigua

La memoria viene allocata in modo tale che un unico oggetto logico viene posto in aree separate e non adiacenti.

Lo spazio degli indirizzi fisici non'è contiguo.

Durante l'esecuzione di un processo viene eseguita la traduzione degli indirizzi per ristabilire la corrispondenza tra lo spazio virtuale di indirizzamento (contiguo) e gli indirizzi fisici.

**PAGINAZIONE**

**MEMORIA VIRTUALE**

# Paginazione

La memoria fisica viene suddivisa in pagine di dimensione fissa chiamate *pagine fisiche* o *frame*.

La memoria logica viene suddivisa in *pagine logiche* o *virtuali* della stessa dimensione delle pagine fisiche.

Caricamento in memoria di un processo:

- si prelevano le pagine logiche dalla memoria ausiliaria
- si caricano le pagine logiche nelle pagine fisiche (blocchi) della memoria centrale

**Traduzione degli indirizzi:** ogni indirizzo generato dalla CPU è diviso in due parti:

- bit più significativi → P - numero di pagina (virtuale);
- bit meno significativi → D - spiazzamento (reale);

Il sistema operativo tiene traccia della corrispondenza tra pagine virtuali e pagine fisiche mediante la *tabella delle pagine* (TDP) o *page table*.

- la TDP viene costruita al momento del caricamento del processo;
- ogni riga della TDP contiene l'indirizzo di partenza della pagina fisica in cui è allocata la corrispondente pagina virtuale
- la tabella è costituita da un numero di righe pari al numero di pagine occupate.

# Paginazione esempio

ES:

Memoria:

1 Mb

Dimensione Pagina:

256 byte (se la memoria è indirizzabile al byte  
occorrono 8bit per il displacement)

Numero di pagine:

4096 (12 bit per indirizzare la pagina)

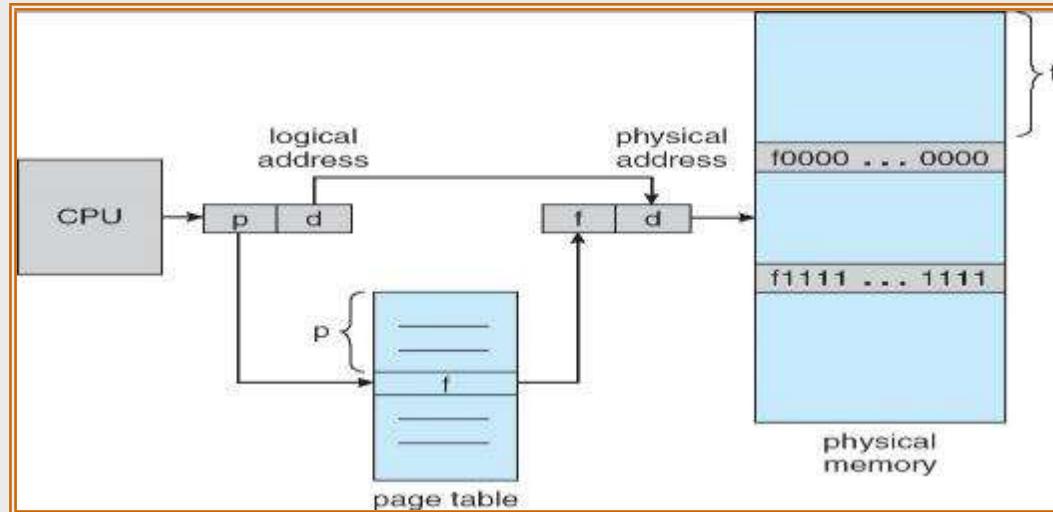
Lunghezza dell'indirizzo:

20 bit (12+8)

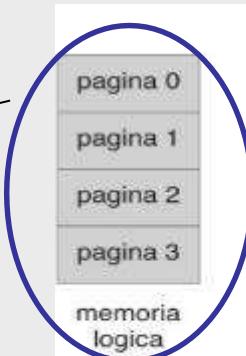
Lo spazio di indirizzamento di un ipotetico processo utente che occupi 1008 byte  
viene diviso in quattro pagine virtuali numerate da 0 a 3.

# Paginazione

## Meccanismo di traduzione



*Memoria vista dall'utente*



numero del frame	pagina
0	1
1	4
2	3
3	7

tabella delle pagine

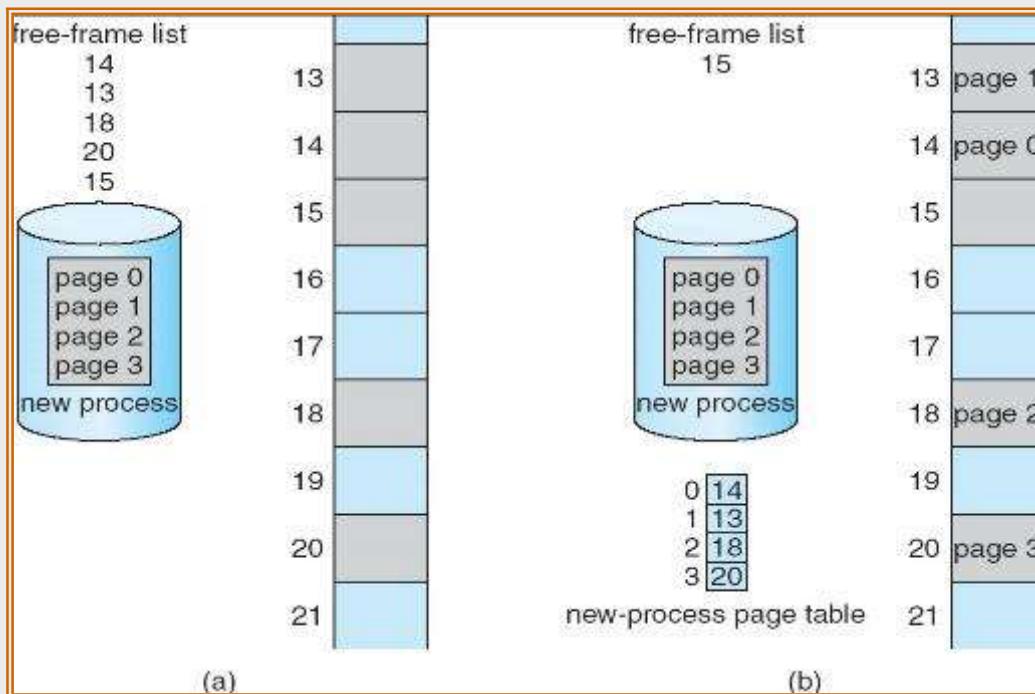
memoria fisica
pagina 0
pagina 2
pagina 1
pagina 3

38

# Paginazione

Il sistema operativo tiene traccia dello stato di tutte le pagine fisiche mediante la *tabella di memoria* (TDM):

- in ogni riga della TDM è indicato lo stato della relativa pagina: “Libera” o “Allocata”;
- il numero di righe della tabella è uguale al numero di pagine fisiche.
- esiste la tabella delle pagine (frame) libere



Quando si deve eseguire un processo si esamina la sua grandezza in pagine e se ci sono sufficienti frame disponibili, ad ogni pagina si assegna un frame. Si carica la prima pagina in un frame e si inserisce il numero del frame ad essa assegnato nella tabella delle pagine, la successiva in un altro frame e così via.

## OSSERVAZIONI:

1. Un processo non può accedere alle pagine di un altro processo poiché queste non sono contenute nella sua tabella delle pagine
2. Ad ogni processo è associata una tabella delle pagine => ulteriori dati da gestire nel context switching

# Paginazione

**Frammentazione interna:** l'ultima pagina assegnata ad un programma non sarà completamente piena

Pagine molto piccole:

- minore frammentazione
- Carico eccessivo nell'accesso al meccanismo di traduzione degli indirizzi
- Numerosi accessi alla memoria di massa,

ATTUALMENTE LA DIMENSIONE DELLE PAGINE E' COMPRESA TRA I 4KB E 8 KB

# Paginazione: Memorizzazione di aree libere

La consultazione della *Tabella di Memoria* statica per trovare  $n$  pagine fisiche libere richiede una ricerca su un numero di righe che in media è  $x = n / q$ .

$q$  è la probabilità che una pagina sia libera e si lega alla percentuale di memoria libera  $u$ :  $q = u / 100$        $0 \leq q \leq 1$

## **x aumenta all'aumentare dell'occupazione di memoria**

Per questo motivo si preferisce sostituire la TDM con una lista a puntatori contenente i numeri delle pagine fisiche:

- $n$  pagine da allocare si individuano nei primi  $n$  nodi della lista
- in fase di deallocazione vengono inseriti in testa alla lista gli  $n$  nodi relativi alle pagine liberate.

### Vantaggi:

- il tempo di allocazione e deallocazione non dipende dalla percentuale di occupazione della memoria.

### Svantaggi:

- per il singolo elemento vi è un lieve spreco di memoria e di tempo rispetto alla gestione di una tabella statica.

# Architettura di paginazione

Se per ogni processo c'è una Page Table, il PCB deve contenere un puntatore a tale tabella.

⇒ Per un meccanismo veloce di traduzione l'intera tabella deve essere posizionata in opportuni registri del processore

⇒ Limitazione alla dimensione di tale tabella... meccanismo non idoneo per i moderni SO

I meccanismi hardware per la paginazione vengono utilizzati per due scopi:

1. **Risparmiare la memoria necessaria per le TDP**
2. **Velocizzare la traduzione da indirizzi virtuali a fisici**

E' ragionevole costruire tabelle con un numero di righe uguale alle pagine effettivamente utilizzate.

- a) La tabella delle pagine viene mantenuta in memoria
- b) registro **base** della tabella delle pagine (PTBR): contiene l'indirizzo di partenza della tabella. Il cambio di tabelle implica il cambio di questo registro. La *velocità si dimezza, doppio accesso alla memoria (TDP e pagina)*...

# Architettura di paginazione

## Velocizzazione del tempo di traduzione

La traduzione di ciascun indirizzo virtuale richiede due accessi in memoria:

- uno alla TDP per prelevare il numero di pagina fisica;
- l'altro per accedere fisicamente alla locazione richiesta.

## SOLUZIONE:

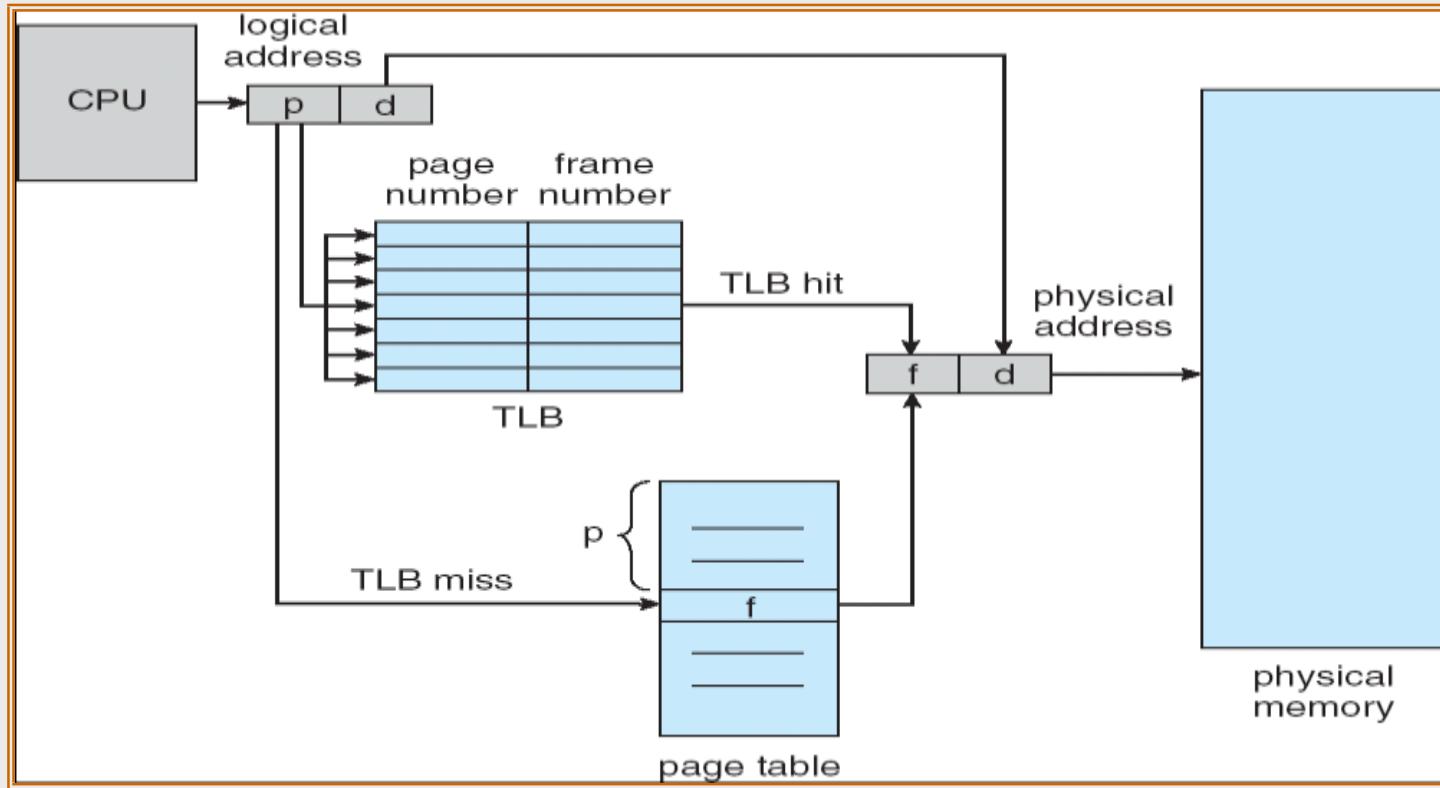
Utilizzo di una cache di traduzione (eventualmente di tipo associativo) “Translation Look-aside buffer” (**TLB**) in cui vengono mantenuti gli indirizzi delle pagine usate più frequentemente (porzione della tabella delle pagine).

Ogni elemento della \$ è costituito da due parti: una chiave ed un valore.

- Si confronta la chiave generata (ad esempio tramite l'operazione mod se la \$ è set associativa...) a partire dall'indirizzo logico della pagina con le chiavi presenti nella TLB:
  - se presente il corrispondente valore (indirizzo della pagina fisica) è usato per accedere alla memoria
  - se assente si è verificato un miss e si deve accedere alla page table in memoria

Aggiornando la \$ otterremo un più rapido accesso al referenziamento successivo.

# Architettura di paginazione



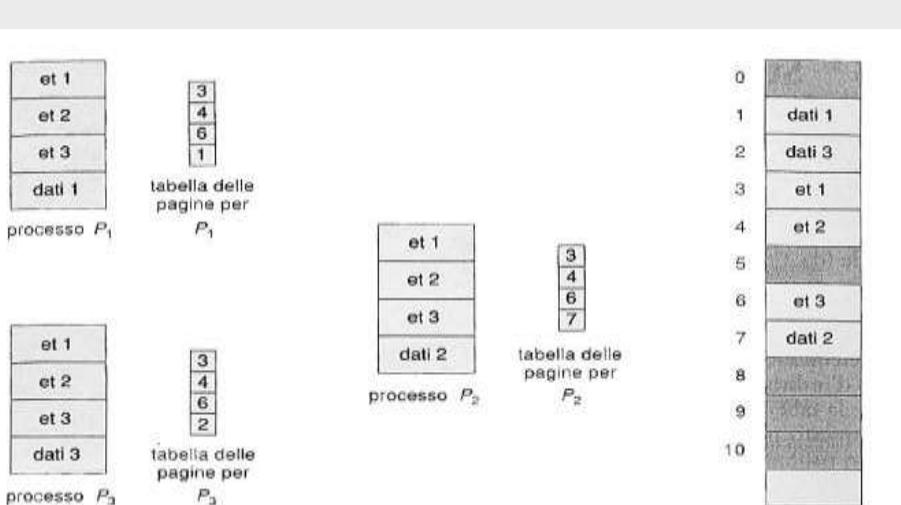
- Se a seguito di un miss si deve inserire nella TLB il nuovo riferimento e non c'è spazio si può usare un algoritmo di sostituzione ad es. LRU...
- Alcune pagine si possono vincolare (wired-down) nella TLB per esempio quelle del Kernel.
- Una TLB può anche contenere un ASID (Address Space Identifier): viene usato per controllare se il processo che tenta di accedere è autorizzato altrimenti si genera un miss TLB. ASID associato ai processi...

# Paginazione: protezione

- ▶ i valori dei registri sono modificati esclusivamente da istruzioni privilegiate del sistema operativo;
- ▶ **l'aggiunta di alcuni bit alle righe delle TDP consente di definire e controllare il tipo di accesso alla pagina (r,rw,exe);**
- ▶ non ha senso la protezione interna allo spazio di indirizzamento del processo, poiché non prevede tipizzazione di dati.
- ▶ bit di validità: valido=pagina corrispondente è realmente utilizzata dal processo. Un processo può utilizzare solo un sottoinsieme delle pagine a sua disposizione
- ▶ Alcuni sistemi utilizzando il Page table length register (PTLR) come indicazione sul numero di pagine appartenenti ad un processo.

# Paginazione: condivisione

- una copia unica di una pagina fisica può essere mappata in molti spazi di indirizzamento;
  - i diversi processi possono avere un tipo di accesso diverso alla pagina;
  - la condivisione viene riconosciuta ed effettuata da programmi di sistema, poiché la paginazione è trasparente all'utente;
  - il codice condiviso deve essere eseguito in mutua esclusione.
- 
- Le pagine condivise vengono utilizzate da alcuni sistemi per inviare messaggi tra processi.



ES.: pagine et1,et2, et3 editor di testo..

# Paginazione Gerarchica

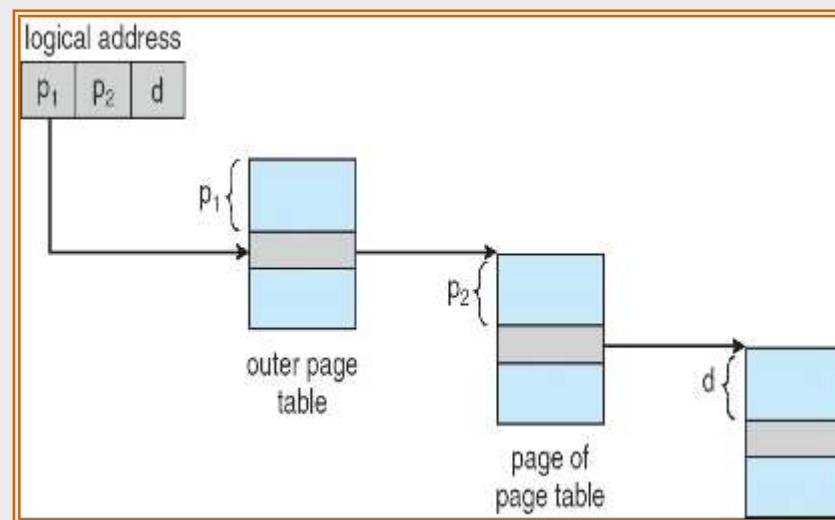
**PROBLEMA:** Nelle moderne architetture a 32 e 64 bit, la dimensione della tabella delle pagine può arrivare ad occupare alcuni MB per ogni processo

**SOLUZIONE:** paginazione a due livelli:

- Viene paginata anche la tabella delle pagine
- Viene generato l'indirizzo logico:

<i>page number</i>	<i>page offset</i>	
$p_1$	$p_2$	$d$

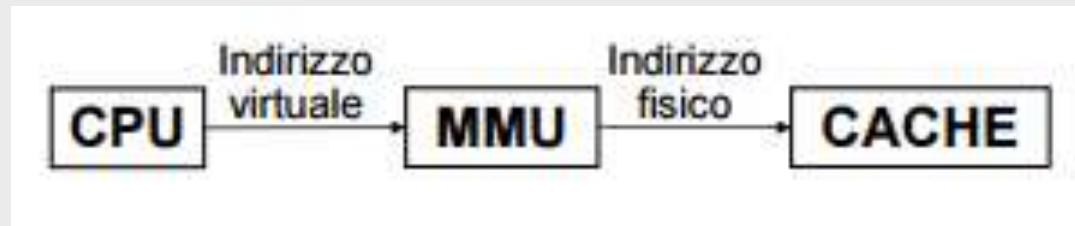
- P1:indirizzo logico della tabella di primo livello
- P2:scostamento all'interno della pagina puntata da P1



# Cpu - \$ - RAM .....

Cosa accade se tra CPU e RAM re-introduciamo il concetto di cache....????

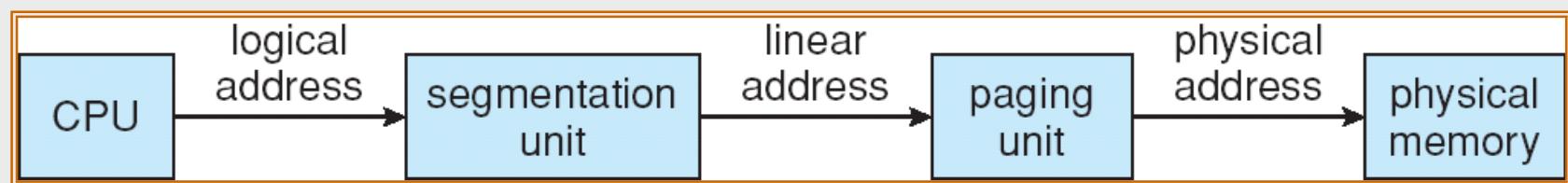
1. l'accesso alla cache può avvenire usando l'indirizzo fisico di memoria:
  - Prima dell'accesso in \$ si traduce l'indirizzo virtuale in indirizzo fisico



2. Alcune architetture consentono l'accesso alla \$ con indirizzi virtuali (raro)

# INTEL PENTIUM

- **Supporta sia la segmentazione che la segmentazione con paginazione**
- La CPU genera indirizzi logici
  - Dati all'unità di segmentazione
  - Produce indirizzi lineari
  - Gli indirizzi lineari vengono dati all'unità di paginazione
  - Genera indirizzi fisici per la memoria centrale



In sostanza I segmenti sono costituiti da pagine.

# Memoria Virtuale

Schema di gestione della memoria in cui soltanto una parte dello spazio di indirizzamento virtuale di un processo “residente” viene effettivamente caricata in memoria.

**Tecnica che consente di eseguire processi senza che essi siano completamente contenuti in memoria**

- la somma di tutti gli spazi di indirizzamento virtuali dei processi attivi può superare la capacità della memoria fisica;
- La dimensione ammissibile per lo spazio di indirizzamento virtuale di un singolo processo può superare la capacità della memoria fisica disponibile in un sistema .

Molte parti di un processo vengono usate raramente.

Es: Il codice per le correzione degli errori;

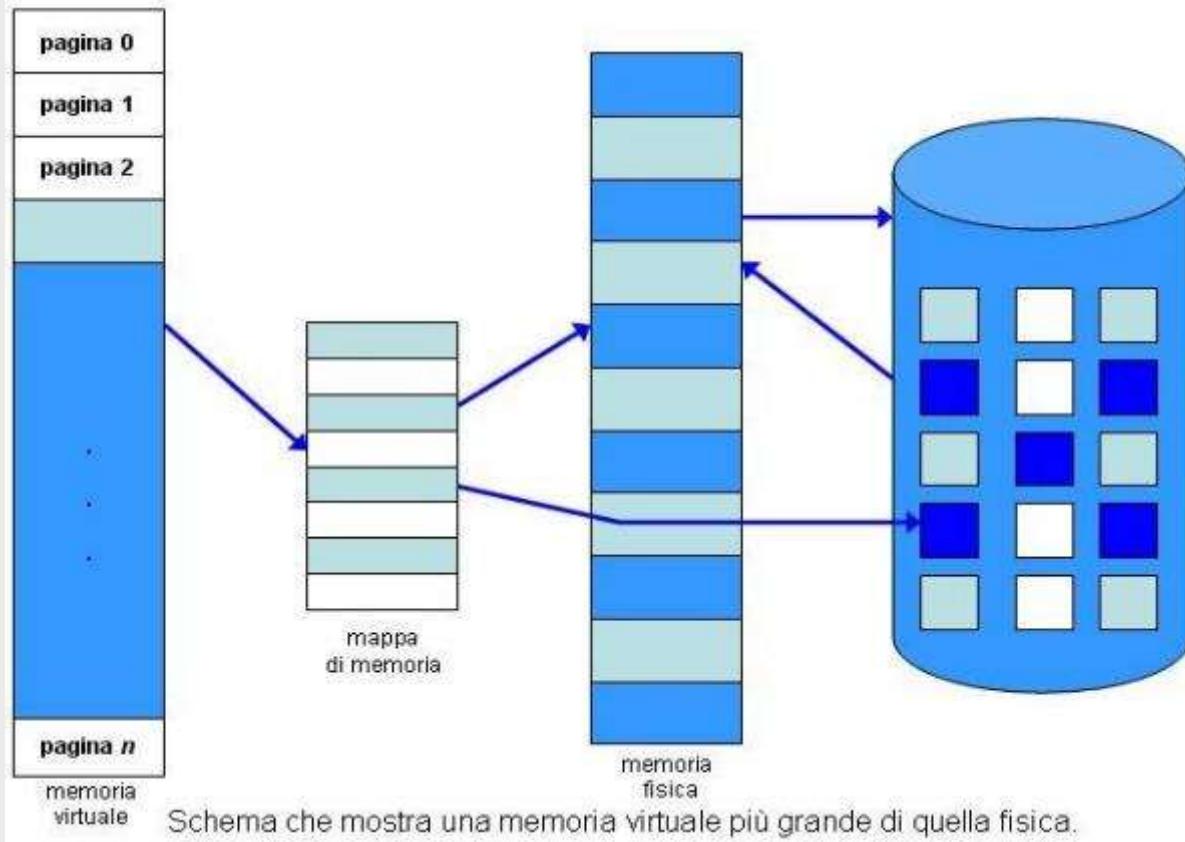
Opzioni usate quasi mai;

Risorse che sono state per prudenza sovradimensionate.

Apparente uso della memoria superiore al 100%

# Memoria Virtuale

*La memoria virtuale si realizza (solitamente) nella forma di paginazione su richiesta*



In memoria secondaria si mantiene l'immagine dell'intero spazio virtuale di indirizzamento del processo;

Si trasferiscono in memoria fisica le diverse parti solo al momento in cui sono necessarie (vengono referenziate);

Il sistema operativo sceglie tempi e modalità di trasferimento in memoria fisica tenendo conto di:

- *richieste dei processi attivi;*
- *priorità dei processi;*
- *disponibilità globali del sistema.*

# Memoria Virtuale

## ***Punto di vista del programmatore:***

- i dettagli della gestione sono trasparenti;
- non è necessario l'adeguamento di un programma ad una memoria limitata;
- un programma può essere eseguito su sistemi con disponibilità di memoria fisica diversa, senza alcun adattamento.

## ***Punto di vista del Sistema Operativo:***

- un processo può essere caricato in uno spazio di dimensione arbitrarie;
- non è necessario modificare l'ordine previsto di esecuzione di un processo;
- il quantitativo di memoria assegnato ad un processo può variare durante la sua esecuzione;
- il sistema operativo può dinamicamente scegliere di velocizzare l'esecuzione di un processo allocando un quantitativo maggiore di memoria o, in alternativa, aumentare il grado di multiprogrammazione.
- si riduce la frammentazione esterna;

# Memoria Virtuale

## Osservazioni:

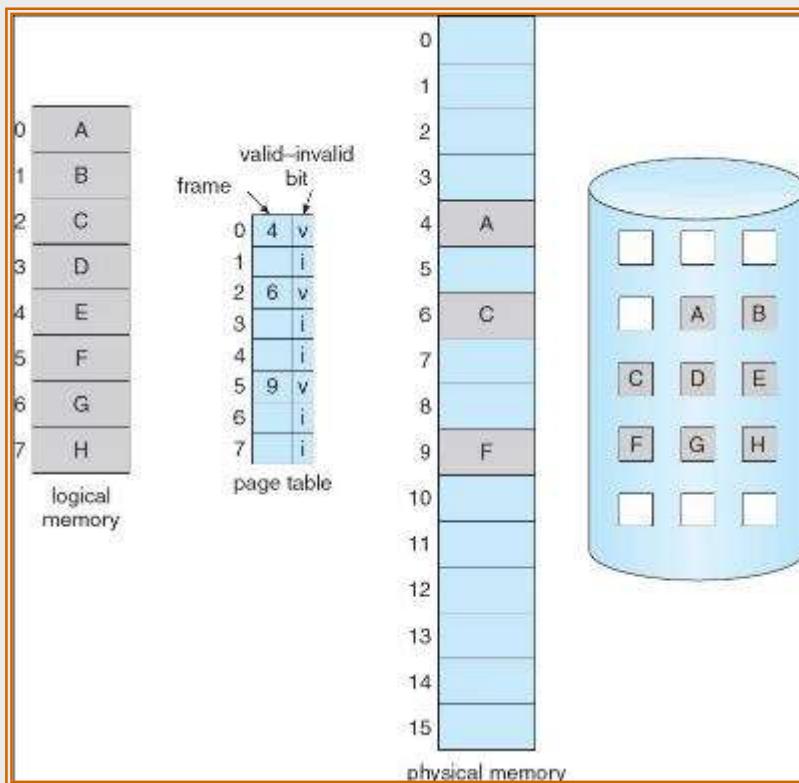
1. Una istruzione può essere completata soltanto se quanto gli è necessario (codice, stack, dati) è presente in memoria;
2. un processo che richiede un “oggetto” fuori dalla memoria viene sospeso (blocked) per un tempo “relativamente” lungo.

## **Importanza dell'uso dello schema di memoria virtuale:**

- durante una specifica esecuzione, alcune parti del programma non vengono indirizzate;
- le diverse condizioni interne o esterne causano diversi tracciati ad ogni esecuzione di un processo (if-else-if, switch-case);
- alcune parti di un programma vengono eseguite molto raramente (es. routine di gestione errori);

# Bit di validità

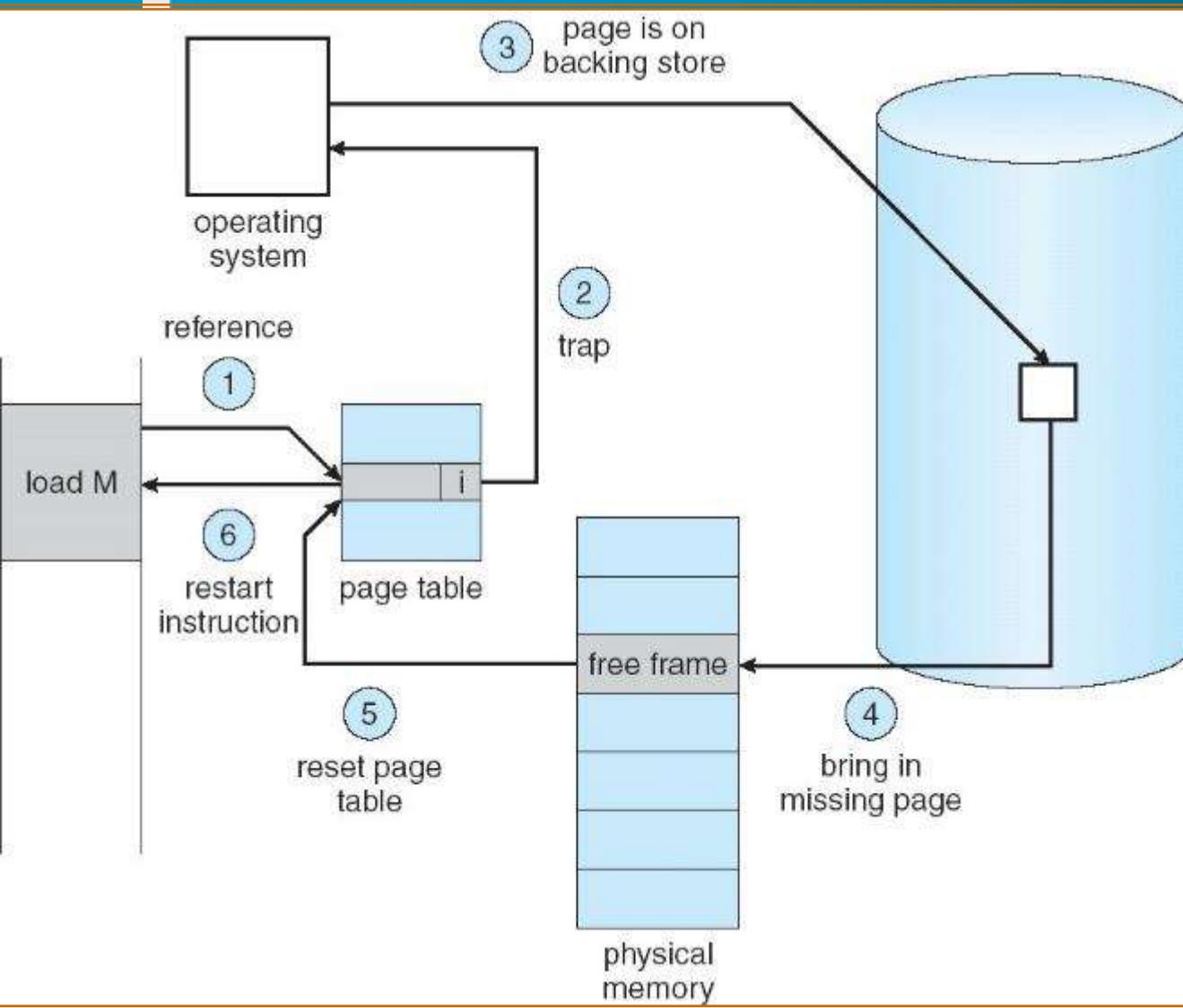
- Al momento del caricamento del processo in memoria, il paginatore opera una predizione sulle pagine che saranno utilizzate
  - Vengono trasferite in memoria solo tali pagine
- **Necessità di un meccanismo che indichi se una pagina è in memoria o su disco**
- **BIT di validità** nella tabella delle pagine: (**v** ⇒ in-memory, **i** ⇒ not-in-memory)



*NB: **i** può anche indicare che la pagina referenziata non è valida per il processo*

**PAGE FAULT TRAP:** eccezione di pagina mancante – riferimento a una pagina non presente in memoria centrale

# Gestione del PAGE FAULT



- OS (attraverso l'uso di una tabella esterna indicata nel PCB) determina se:
  - Invalid reference  $\Rightarrow$  abort
  - Pagina non in memoria
- Individuazione di un blocco libero
- Trasferimento della pagina nel blocco
- Aggiornamento della tabella delle pagine (include, validation bit = **V**)
- Riavviare l'istruzione che aveva cagionato il page fault...questa volta la pagina referenziata sarà in memoria...

# Paginazione su richiesta

## **PAGINAZIONE SU RICHIESTA PURA**

Una pagina non viene trasferita in memoria fino a quando non'è richiesta

- E' possibile avviare l'esecuzione di un processo senza pagine in memoria
- Al primo riferimento si genera un page fault
- Una volta caricata la pagina l'esecuzione del processo prosegue fino al successivo page fault

Meccanismi di ausilio al SO:

1. Tabella delle pagine con bit di validità
2. Memoria ausiliaria

La parte di memoria ausiliaria dedicata all'avvicendamento dei processi prende il nome di area di avvicendamento o di swap (scambio)

NB: sistema di paginazione su richiesta trasparente all'utente e si colloca tra CPU e memoria

# Performance della Paginazione su richiesta

*Fino a quando non si verificano page fault, il tempo di accesso effettivo è uguale al tempo di accesso alla memoria (nella ipotesi di assenza di \$)*

Sia  $p$  la probabilità che si verifichi un Page Fault ( $0 \leq p \leq 1$ )

- $p = 0$  no page faults
  - $p = 1$ , every reference is a fault
- 
- Tempo di accesso effettivo EAT=  
$$= (1 - p) * \text{memory access} + p * (\text{tempo gestione pagina mancante})$$

## Gestione pagina mancante:

- Generazione trap
- Salvataggio contesto processo (run->blocked)
- Determinazione della natura della trap
- Controllo correttezza riferimento
- Lettura da disco e trasferimento
  - Attesa in blocked per lo specifico dispositivo (la CPU può essere assegnata ad altri processi)
  - Seektime+rotational latency+transfert time

continua...

# Performance della Paginazione su richiesta

...continua

- Generazione interrupt (I/O completato)
- Salvataggio contesto processo nello stato di Run
- Gestione interruzione
- Aggiornamento tabella delle pagine per il processo che aveva generato il page fault, blocked->ready
- Attesa in ready
- Ripristino del contesto

Esempio:

• Memory access time = 100 nsec

• Average page-fault service time = 25 msec

$$\begin{aligned} \text{EAT} &= (1 - p) \times 100 + p \text{ (25 milliseconds)} \\ &= (1 - p) \times 100 + p \times 25,000,000 \\ &= 100 + p \times 24,999,900 \end{aligned}$$

• Se un accesso ogni 1000 genera un page fault: EAT = 25,000 nsec

RALLENTAMENTO DEL 250%!!!

Per avere un rallentamento del 10% il sistema deve garantire meno di una assenza ogni 2,500,000 di riferimenti

*Tempo di accesso effettivo proporzionale alla frequenza di assenza delle pagine*

# Paginazione su richiesta

*Come selezionare quali pagine portare in memoria:*

**CONCETTO DI LOCALITÀ:** i programmi hanno una forte tendenza a favorire un sottoinsieme del loro spazio di indirizzamento durante l'esecuzione.

Una parte notevole degli accessi, in un periodo di tempo, vengono effettuati su un set ridotto delle pagine virtuali di un processo. Un processo si “muove” lentamente da una località ad un'altra nel corso della sua esecuzione.

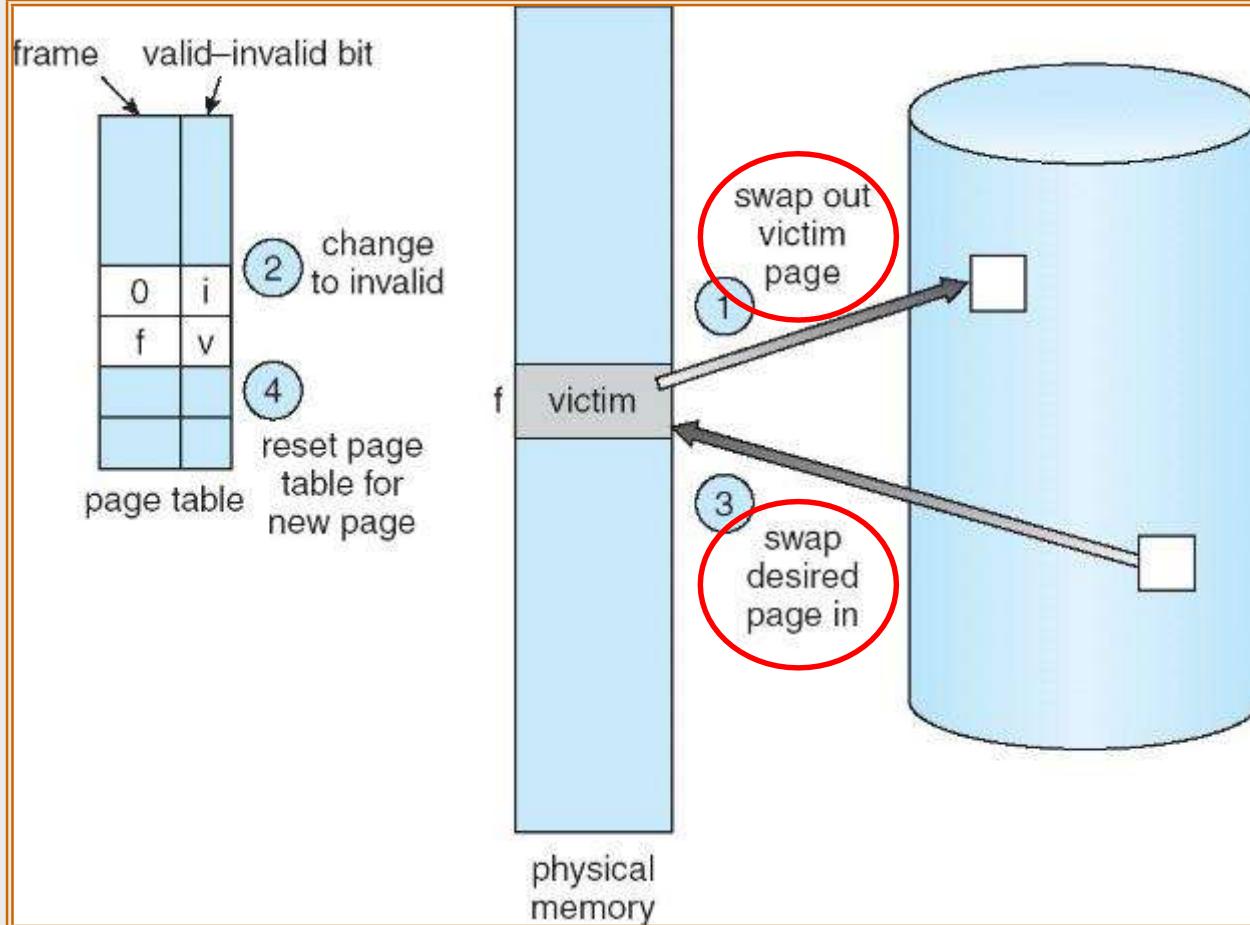
Il set di pagine più referenziate varia lentamente nel tempo. Questa proprietà è utilizzata per realizzare le strategie di allocazione e sostituzione delle pagine nello schema di gestione della memoria virtuale.

*Come gestire il caso di:*

**MANCANZA DI SPAZIO LIBERO IN MEMORIA:**

1. Terminare un processo utente... *scelta peggiore*
2. Scaricare dalla memoria l'intero processo e ridurre il grado di multiprogrammazione
3. **Sostituzione delle pagine:**
  1. Necessità di un algoritmo di selezione della pagina “vittima”
  2. Due trasferimenti da/a memoria (swap-in swap-out)
  3. Sovraccarico al 2. riducibile tramite *dirty bit* (blocco in memoria da portare su disco non modificato dal suo caricamento)

# Page Replacement



Algoritmo di replacement:

- Minor page-fault possibile
- Valutare l'algoritmo su specifiche sequenze di accesso alla memoria
- Negli esempi la sequenza è:

**1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

# Algoritmi di replacement: FIFO

**Vengono sostituite le pagine che da più tempo risiedono in memoria.**

Il gestore della memoria tiene traccia dell'ordine di caricamento delle pagine in memoria, ad esempio con una coda FIFO dei numeri di pagina.

Pregi:

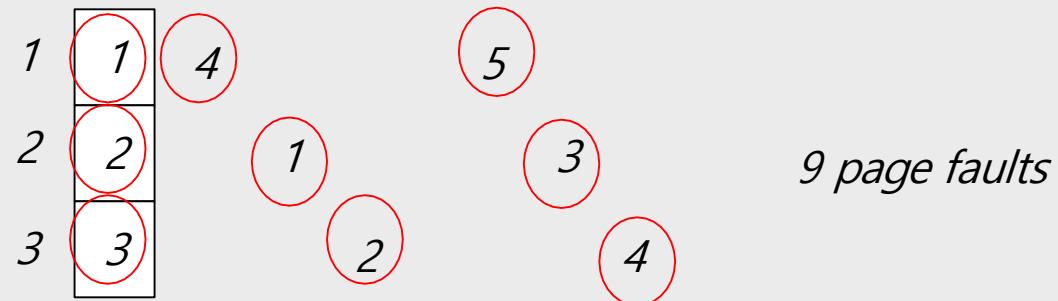
- semplice realizzazione;
- aggiornare la coda solo ad ogni page fault;
- non richiede supporti hardware specifici.

Difetti:

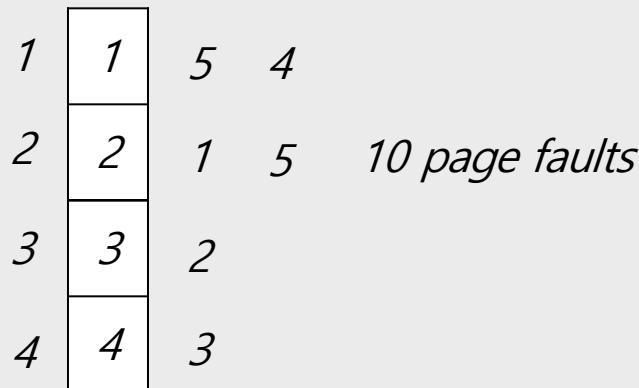
- swap out di pagine più frequentemente indirizzate, poiché per loro natura permangono più tempo in memoria;
- non tiene traccia dell'ordine di riferimento delle pagine.

# Algoritmi di replacement: FIFO

- Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages in memoria)



- 4 frames

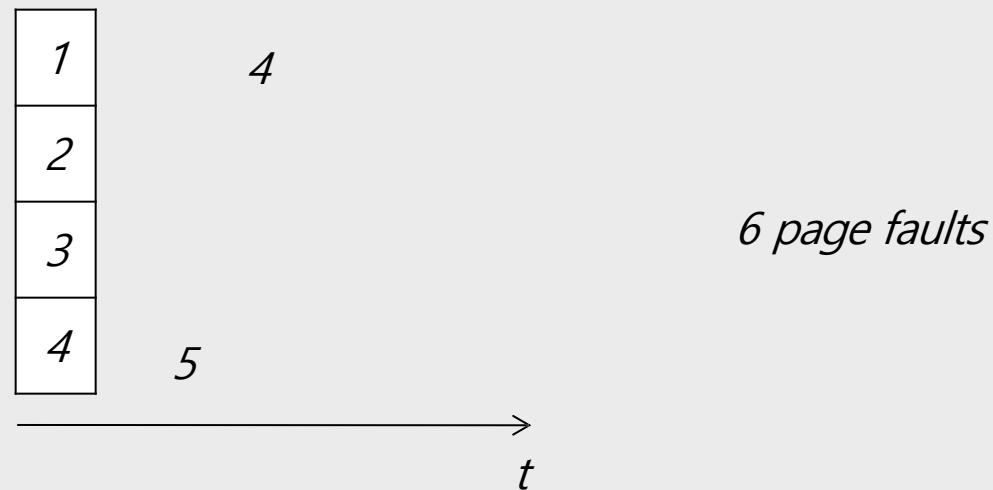


- Anomalia di Belady: con alcuni algoritmi di sostituzione delle pagine, più frames  $\Rightarrow$  più page faults

# Algoritmi di replacement: Algoritmo Ottimale o di Belady

- **Sostituire le pagine che non saranno utilizzate per il periodo di tempo più lungo (pagine che saranno utilizzate solo dopo molto tempo)**
- Algoritmo ottimale: minimizza il numero di page faults
- 4 frames, es:

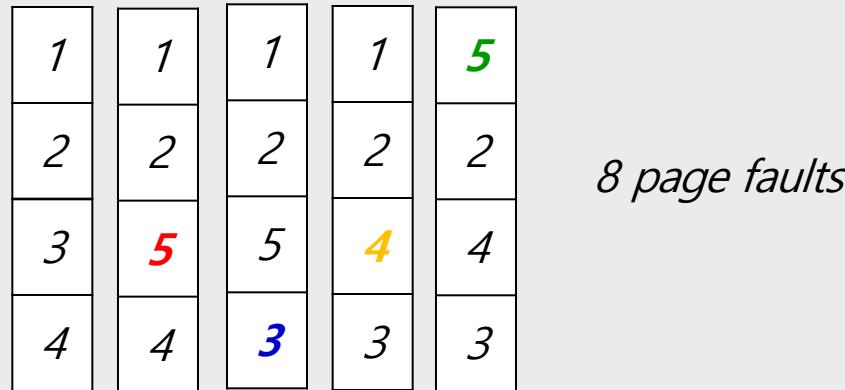
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Non realizzabile: richiede la conoscenza futura della successione dei riferimenti
- Consente di confrontare le prestazioni degli altri algoritmi.
- NB: non soggetto ad anomalia di Belady

# Algoritmi di replacement: Least Recently Used (LRU)

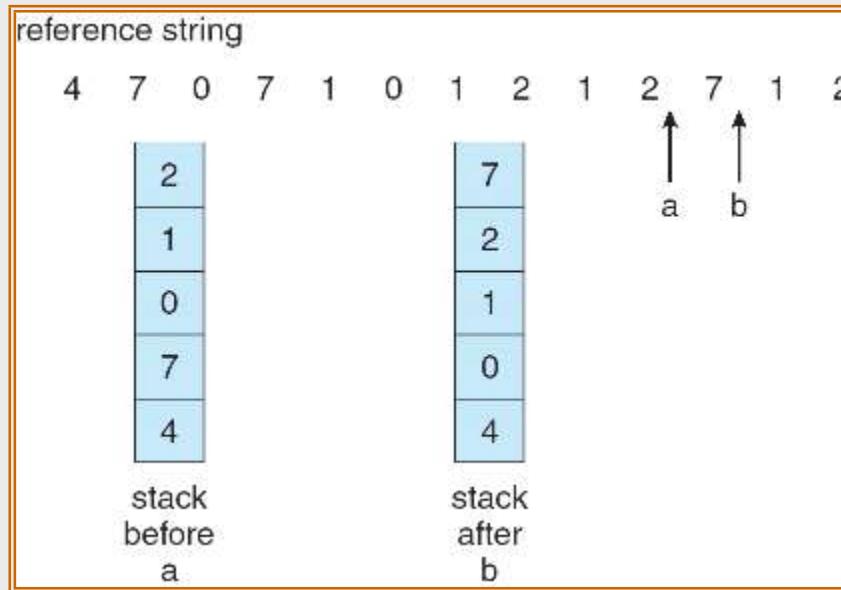
- **Si sostituiscono le pagine meno recentemente utilizzate**
  - Ipotesi: la pagina meno utilizzata recentemente è quella che ha la minore probabilità di essere referenziata in futuro. Si approssima il futuro prossimo al passato recente
- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



- Non soggetto ad anomalia di Belady
- **Strategie implementative (forte assistenza del SO)**
  - Associare ad ogni pagina l'istante in cui è stata usata per l'ultima volta
    - Campo aggiuntivo nella page table
    - Necessità di dover ricercare la pagina con il più vecchio marcitore temporale
  - Memorizzare l'uso delle pagine in una struttura di tipo stack:
    - Ogni volta che una pagina viene referenziata viene spostata in cima allo stack
    - La pagina da swap out viene presa dalla base dello stack
    - Necessità di dover costantemente aggiornare lo stack vs. semplice selezione della pagina vittima

# Algoritmi di replacement: Least Recently Used (LRU)

*Utilizzo dello stack...*



Pregi:

- in media si comporta meglio dell'algoritmo FIFO.

Difetti:

- l'aggiornamento dello stack va effettuato ad ogni riferimento alla pagina (praticamente ad ogni accesso in memoria)
- l'organizzazione dell'LRU è tale da richiedere un supporto hardware sofisticato e dedicato alle operazioni relative all'aggiornamento dello stack.

# Algoritmi di replacement: Approssimazione a LRU

- Utilizzo di un contatore per numerare il numero di accessi alle pagine
- **LFU Algorithm – Least Frequently Used:** sostituzione delle pagine con il valore di contatore più basso
  - Filosofia: le pagine utilizzate frequentemente hanno valori contatori elevati
  - Alcune pagine potrebbero essere referenziate frequentemente all'avvio di un processo e poi non essere più necessarie. Soluzione: azzerare il contatore periodicamente...
- **MFU Algorithm – Most Frequently Used:** sostituzione delle pagine con il più alto valore di contatore
  - Filosofia: la pagina con valore basso di contatore è stata appena portata in memoria e sarà utilizzata in futuro

**Algoritmi poco utilizzati: male approssimano l'algoritmo ottimale.**

# Algoritmi di replacement: Approssimazione a LRU

- **Reference bit (bit di riferimento)**
  - Ad ogni pagina (nella tabella delle pagine) è associato un bit, inizialmente =0
  - Quando una pagina viene referenziata (sia in lettura che in scrittura) si setta il bit a 1
  - Sostituire le pagine che hanno il bit a 0
    - NB: non si conosce l'ordine di utilizzo
    - I bit possono essere azzerati ad intervalli regolari
- **Seconda chance (basato su logica FIFO)**
  - Utilizzo del reference bit
  - Clock replacement (per la generazione della coda FIFO)
  - Dopo aver selezionato una pagina dalla coda FIFO si controlla il bit di riferimento:
    - 0 => sostituzione della pagina
    - 1 => si dà una seconda chance alla pagina (non la si sostituisce)
      - Si pone il bit a 0
      - si passa ad esaminare un'altra pagina nella coda FIFO
- **Seconda chance migliorato**
  - Utilizzo di reference e dirty bit:
    - (0,0) – pagina non referenziata nè modificata ->migliore pagina da sostituire
    - (0,1) – non referenziata ma modificata -> prima di sostituire la pagina la si deve salvare
    - (1,0) – usato recentemente ma non modificato -> potrebbe essere nuovamente referenziata
    - (1,1) – usato recentemente e referenziato ->...
    - MACH OS

# Il File System

Le applicazioni su un calcolatore hanno bisogno di memorizzare e rintracciare informazioni.

Le informazioni memorizzate nello spazio degli indirizzi di un processo vengono perse al termine dell'esecuzione del processo.

Un processo può utilizzare il suo spazio degli indirizzi per memorizzare un quantitativo limitato di informazioni.

Può essere necessario che le informazioni siano memorizzate per lungo tempo.

Più processi possono aver bisogno delle stesse informazioni contemporaneamente per cui è necessario che queste e la loro allocazione siano indipendenti dal processo.

Per memorizzare grandi quantità di informazioni, rendere la memoria permanente e dare la possibilità a più processi di accedere alle stesse informazioni è necessario registrare tali informazioni in dischi o altri supporti in unità dette file.

# I File...

Il sistema operativo consente la gestione dei file mediante il modulo di gestione del **File System**.

Oltre al contenuto proprio del file, il file system gestisce informazioni complementari relative al file come:

- struttura;
- nome;
- tipo di accesso;
- protezione, ecc.

Il nome del file si suddivide generalmente in:

**Nome.Estensione**

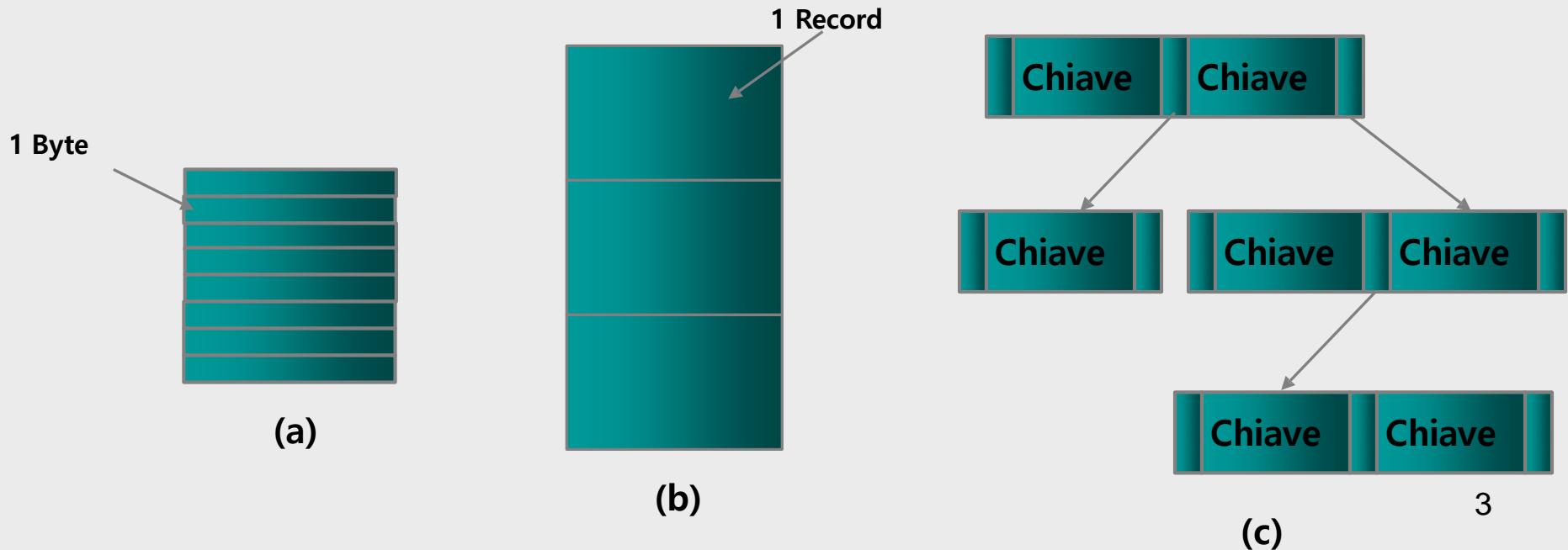
entrambi con un numero variabile di caratteri e varia utilizzazione del minuscolo/maiuscolo.

Spesso l'estensione indica un particolare tipo di file...

# I File (struttura)...

I file, in genere, contengono le informazioni in tre tipi di strutture:

- **sequenza non strutturata di byte:** consente la massima flessibilità al programma utente.  
Dos e Unix utilizzano questa struttura (a); Accesso diretto ai singoli byte
- **sequenza di record a lunghezza fissa:** ha perso popolarità con il tempo (b);
- **albero di record con campo chiave:** ancora in uso sui grossi mainframe usati per l'elaborazione di dati commerciali (c).



# I File (tipi e formati)...

Tipi di file:

- **file regolari**: contengono le informazioni dell'utente, un programma eseguibile, ecc.
- **directory**: conservano la struttura del file system ed informazioni sui file regolari; **file**
- **speciali a caratteri**: usati per modellare unità di input/output seriali come video, stampanti, reti, ecc;
- **file speciali a blocchi**: usati per modellare unità di input/output a blocchi come i dischi.

Le informazioni nei file sono generalmente contenute in due formati:

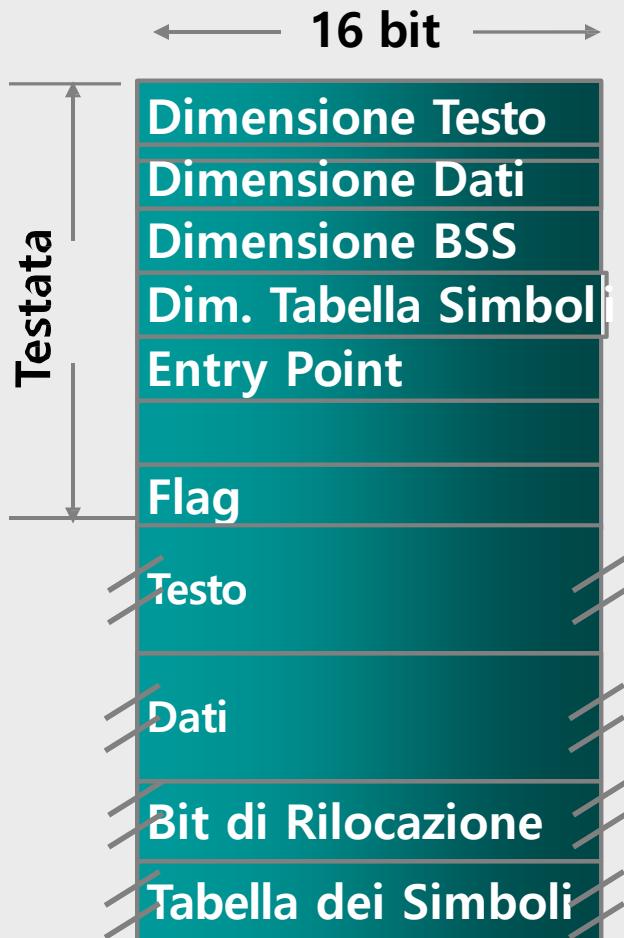
- **codice ASCII (o UNICODE, ecc.)**

- ✗ utilizza 8bit di cui uno di controllo: 7bit=128 configurazioni
  - ✗ UNICODE: Codice a 16 bit (codifica i caratteri usati in quasi tutte le lingue vive e in alcune lingue morte, nonché simboli matematici e chimici, cartografici, l'alfabeto Braille, ideogrammi etc )
  - ✗ i file programma sorgente ed i documenti sono generalmente registrati in questo formato.

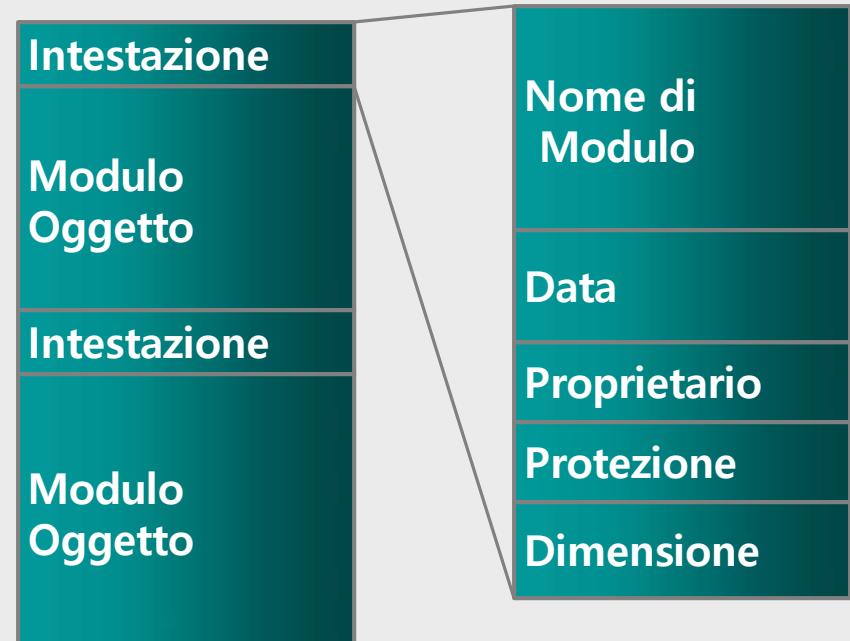
- **codice Binario:**

- ✗ le sequenze di codici sono “illeggibili”;
  - ✗ di solito i file memorizzati in questa modalità hanno una struttura interna.

# I File (formati)...



file eseguibile Unix



archivio contenente procedure

# I File (accesso)...

## ● **Accesso Sequenziale:**

- ✖ i byte o i record che costituiscono il file sono letti uno dopo l'altro, dall'inizio alla fine;

## ● **Accesso Casuale (diretto).**

- ✖ l'accesso al blocco di informazioni avviene in maniera diretta;
- ✖ l'avvento dei dischi ha consentito l'introduzione di questo metodo di accesso;
- ✖ Meccanismo utilizzato dalle basi di dati
- ✖ generalmente l'accesso è diretto al blocco e sequenziale all'interno del blocco.

# I File (attributi)...

I sistemi operativi associano ai file altre informazioni come:

- data e ora della creazione;
- proprietario;
- dimensione, ecc.

<u>Campo</u>	<u>Significato</u>
Protezione	Chi può accedere al file e in che modo
Password	Parola d'ordine necessaria per accedere al file
Flag di sola lettura	0 per lettura/scrittura, 1 per sola lettura
Flag di sistema	0 per file normale, 1 per file di sistema
Flag ASCII/binario	0 per file ASCII, 1 per file binario
Lunghezza record	Numero di byte in un record
Tempo di creazione	Data e ora del momento in cui è stato creato il file
Dimensione attuale	Numero di byte nel file
Dimensione massima	Dimensione massima che il file può raggiungere

# I File (operazioni)...

Il sistema operativo pone a disposizione dell'utente **comandi di alto livello** e **chiamate di sistema** per la gestione dei file. Alcune chiamate di sistema più comuni sono:

- **create**: creazione del file;
- **delete**: cancellazione del file;
- **open**: apertura del file con varie modalità (read-only, append, ecc.);
- **close**: chiusura del file;
- **read**: lettura di blocchi di dati;
- **write**: scrittura di blocchi di dati;
- **append**: aggiunta, in coda al file, di nuovi dati;
- **seek**: posizionamento del puntatore all'interno del file.

# File mappati in memoria...

Un file viene caricato tutto o in parte in memoria e gli vengono assegnati indirizzi virtuali.

- I processi possono accedere in maniera trasparente ai file (possono essere quindi contenuti in memoria centrale per interi o per blocchi)
- Si velocizzano le operazioni compiute sul file, soprattutto quando si riferiscono a parti limitate e non all'intero file.

La virtualizzazione dell'accesso comporta alcuni problemi :

- In caso di apertura contemporanea del file da parte di più processi alcune informazioni possono risultare inconsistenti;
- Quando il processo che sta usando il file termina, quest'ultimo viene riscritto sul disco.
- In caso di caduta del sistema alcune informazioni possono essere perse;

# Le directory

I vecchi sistemi operativi prevedevano un' **unica directory** in cui confluivano tutti i file.

Questo concetto è ormai superato dai **sistemi gerarchici** di directory.

Tipicamente un *elemento della directory* contiene:

- il nome di un file ed eventualmente include anche gli **attributi** del file;
- in alternativa, il nome di un file ed un **puntatore** ad un'altra struttura in cui si trovano gli attributi e gli indirizzi del disco.

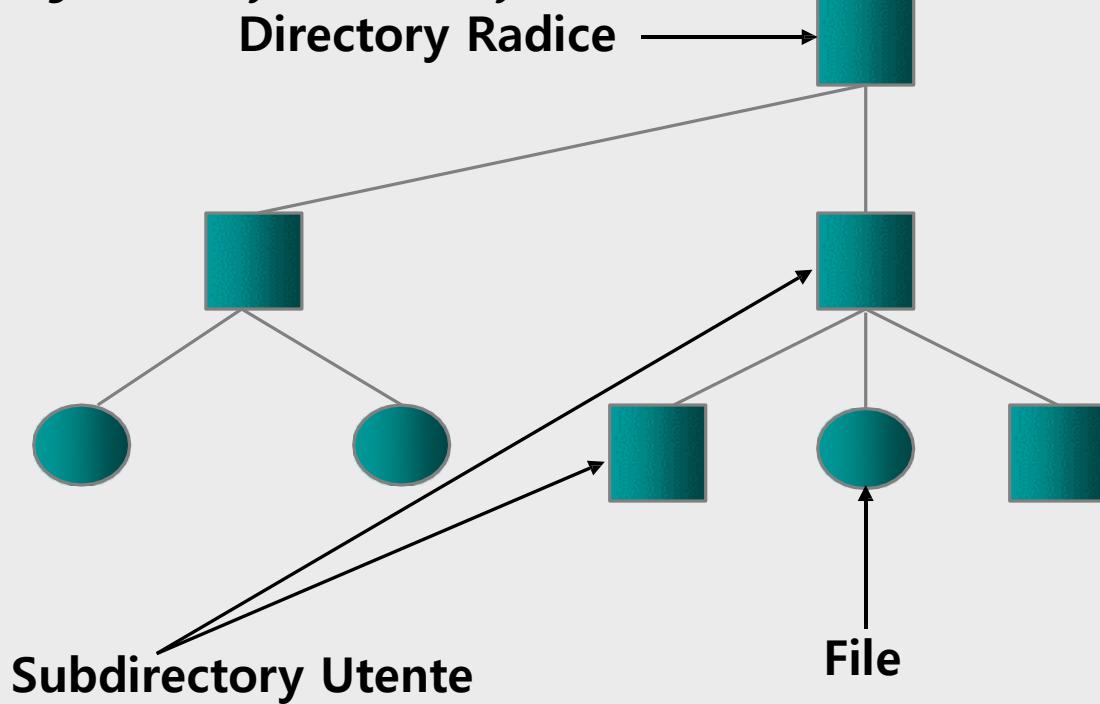
Ogni directory può contenere:

- **file regolari**;
- altre **directory**.

In questo modo, a partire da una directory **radice** (**root** directory) è possibile generare un albero di directory e **subdirectory**.

# Le directory

Progetto di file system / directory



Il tipo di organizzazione dell'*albero delle directory* dipende dalle scelte dell'amministratore di sistema. Un tipico esempio prevede:

- la root directory include alcune directory di sistema come *etc, bin, lib, tmp, ecc.*;
- la directory *usr* contiene una subdirectory per ogni utente ed ogni utente organizza il proprio sottoalbero nel modo che ritiene più efficiente e comodo per il tipo di attività che svolge.

- **path name assoluto:** cammino dalla directory radice al file. I componenti del cammino sono separati da opportuni simboli di separazione come “*/*” in Unix
- **path name relativo:** si usa congiuntamente al concetto di directory di lavoro. Un utente può definire una directory come directory di lavoro corrente. In questo caso tutti i path name che non iniziano con la directory radice sono considerati relativi alla directory di lavoro.

# Le directory (operazioni)

Alcune tipiche chiamate di sistema per la gestione delle directory sono:

- **create:** crea una directory vuota;

*mkdir nome*

*Crea directory nidificate: mkdir -p dir1/dir2*

- **delete:** cancella una directory;

*cancellare directory vuota: rmdir nome*

*cancellare directory non vuota: rm -rf nome*

- **Spostarsi tra le directory:**

*cd newdir*

*cd .. (directory precedente)*

*cd (directory home)*

*cd ~bill (directory home dell'utente bill)*

- **rename:** rinomina una directory;

*mv source dest*

- **Elencazione file:**

*ls*

*ls -l elenco dettagliato*

# Implementazione del file system

Un fattore chiave nell'implementazione della memorizzazione dei file è tener traccia di quali blocchi del disco associare a ciascun file.

Le modalità di allocazione dei blocchi e del relativo reperimento sono:

*Allocazione contigua*

*Allocazione a lista concatenata*

*Allocazione a lista concatenata con indice*

*Allocazione mediante uso di tabelle i-node*

# Allocazione contigua

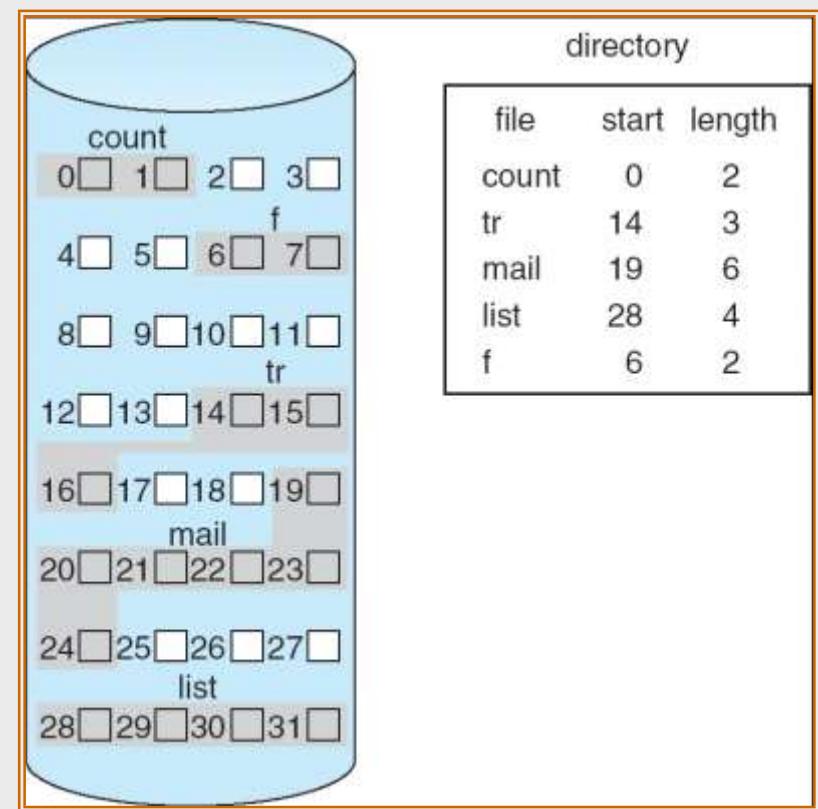
Memorizza il file in blocchi di disco consecutivi.

Pregi:

- semplice da implementare: occorre tenere traccia dell' indirizzo di inizio del file e della sua lunghezza;
- efficiente perché permette di leggere l'intero file con una sola operazione.

Difetti:

- è necessario conoscere la dimensione massima del file;
- il disco risulta frammentato (frammentazione esterna) e molto spazio viene sprecato.
- Difficoltà di reperire spazio libero
  - Best fit
  - First fit
  - Worst fit



# Allocazione a lista concatenata

Ogni blocco associato al file contiene:

- dati veri e propri;
- il numero del blocco successivo assegnato al file.

Pregi:

- non implica spreco di spazio perché i blocchi vengono allocati dinamicamente;
- no frammentazione esterna;
- nell'elemento della directory viene memorizzato l'indirizzo del primo blocco e dell'ultimo blocco.

Difetti:

- La lettura sequenziale del file è semplice, ma l'accesso diretto ai blocchi è estremamente lento;
- Ogni blocco deve contenere il puntatore al blocco successivo. Tale area deve essere non accessibile ai programmi utenti... (protezione)

# Allocazione a lista concatenata con indice

- I blocchi di disco assegnati al file non sono necessariamente contigui;
- L'elenco dei blocchi assegnati al file è mantenuto mediante una tabella (sezione su disco all'inizio di ciascuna partizione).
- L'elemento della tabella indicizzato dal numero di blocco contiene il numero di blocco successivo del file. L'ultimo blocco punta a NULL
- Blocchi vuoti sono indicati da NULL
- L'elemento directory contiene il numero del primo blocco del file

Questo tipo di allocazione è utilizzata nel sistema operativo MS-DOS: FAT (File Allocation Table)

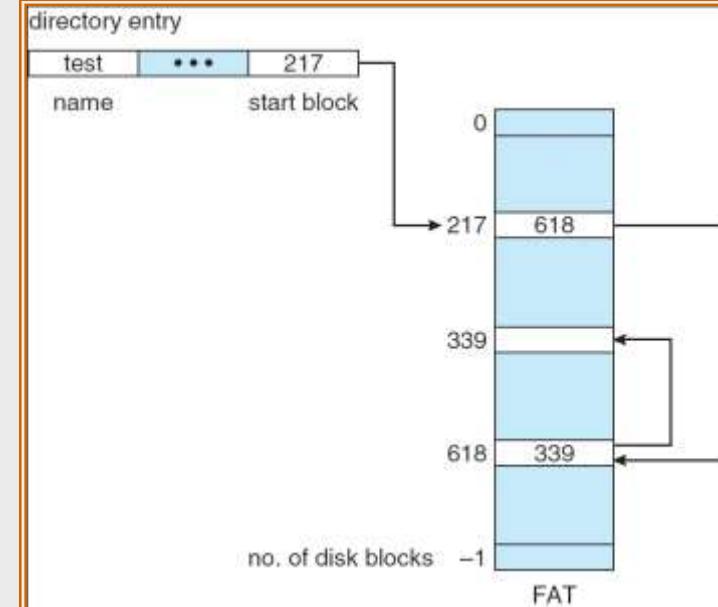
## Pregi:

- l'intero blocco è disponibile per i dati;
- l'accesso diretto ad un blocco è rapido;

## Difetti:

- Ad un gran numero di blocchi di disco corrisponde una tabella grande che occupa molto spazio in memoria centrale.
- La lettura della FAT può causare numerosi spostamenti della testina.

Soluzione: FAT in \$



# Allocazione mediante uso di tabelle i-node

Allocazione indicizzata: tutti i puntatori ai blocchi di un file sono raggruppati in una sola locazione:  
blocco indice.

Ogni file ha il proprio blocco indice

L'elemento directory contiene l'indirizzo del blocco indice

Allocazione utilizzata in Unix.

Gli i-node sono piccole tabelle che contengono:

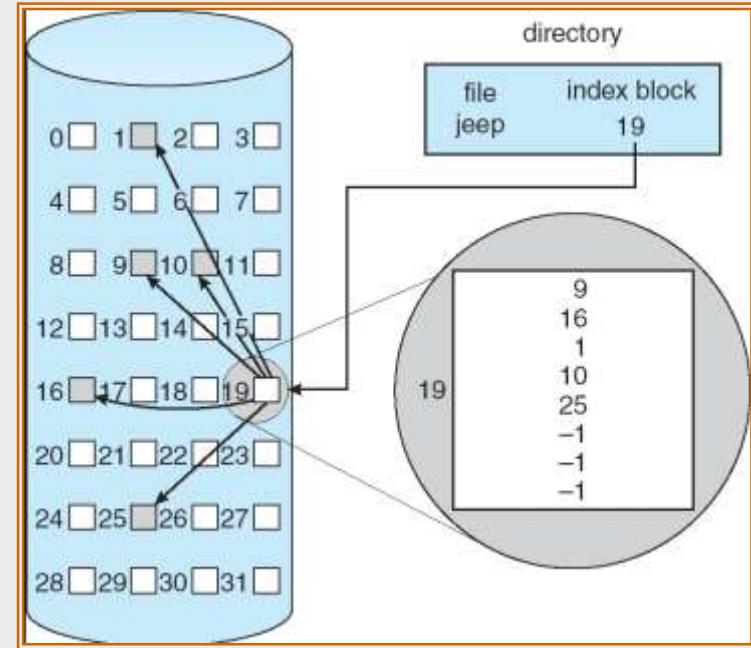
- gli attributi del file;
- gli indirizzi dei primi quindici blocchi assegnati al file:

- I primi 12 puntano a blocchi diretti: contengono direttamente l'indirizzo dei blocchi del file
- Gli altri 3 puntano a blocchi indiretti.

Blocco indiretto singolo: il blocco indirizzato non contiene dati ma indirizzi di altri blocchi i quali contengono dati

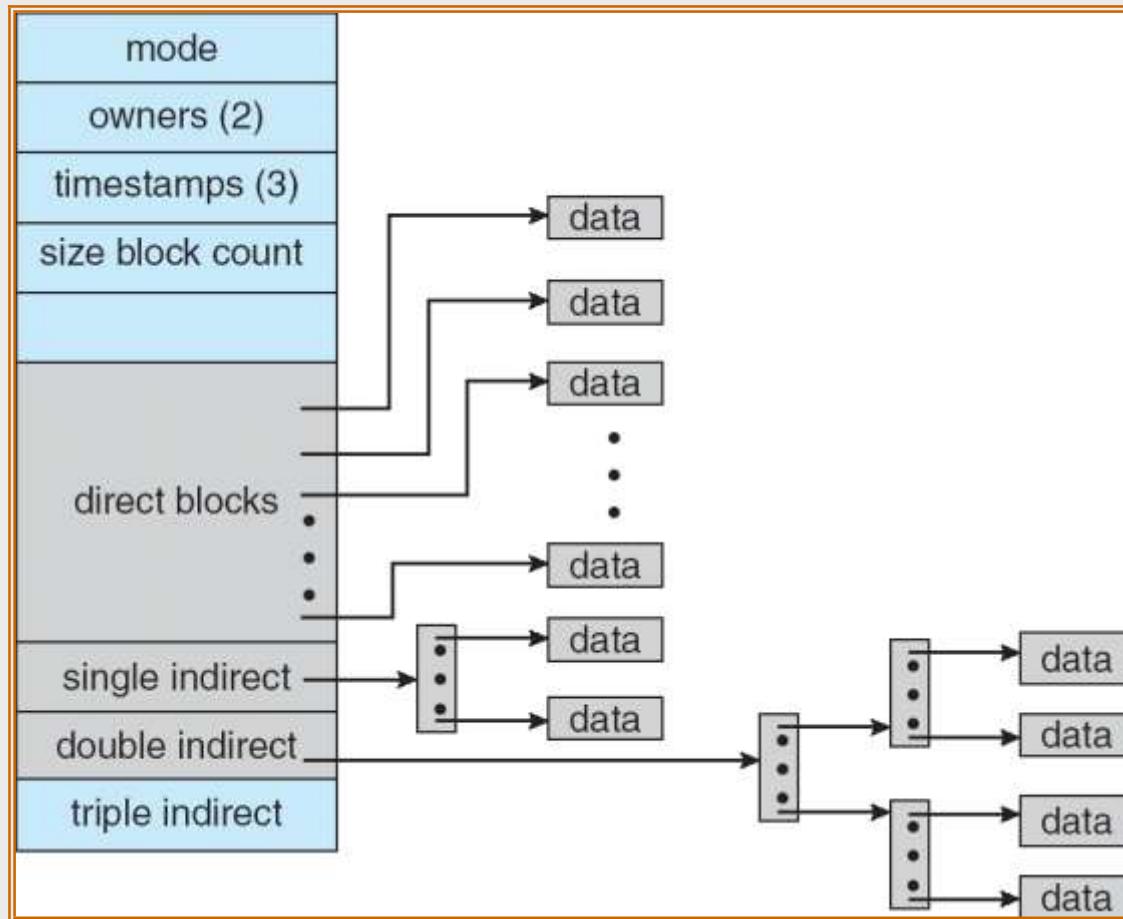
Blocco indiretto doppio

Blocco indiretto triplo



# Allocazione mediante uso di tabelle i-node

Combined Scheme: UNIX (4K bytes per block)



## Pregi

- l'elemento di directory contiene solo il nome del file ed il numero di i-node;
- consente di mantenere in memoria gli i-node dei file aperti sprecando poco spazio;
- consente di indirizzare un numero enorme di blocchi di disco;
- per i file di piccole dimensioni consente di reperire rapidamente l'indirizzo dei primi blocchi di disco.

# Gestione dello spazio su disco: Dimensione dei blocchi

Blocco molto grande:

- rende più veloce il tempo di lettura (seek+rotational+transfert)
- provoca maggior spreco di spazio (frammentazione interna).

Blocco molto piccolo:

- causa numerosi accessi ai blocchi per leggere i dati
- consente una gestione più efficiente dello spazio.

L'efficienza nel tempo e nello spazio sono intrinsecamente in conflitto.

Confrontando la velocità di trasferimento dei dati e l'efficienza dello spazio su disco risulta che a una buona utilizzazione dello spazio corrisponde una bassa percentuale di dati e viceversa.

COMPROMESSO:

Dimensione blocco di allocazione: 512 byte, 1 KB, 2 KB, 4KB, 8KB

NB: il settore è la minima unità fisica leggibile dal disco =>la dimensione del settore sul disco è generalmente di 512 byte e la lettura di un blocco di allocazione di 1 K causerà sempre la lettura di due settori di disco.

# Gestione dello spazio libero su disco

I metodi usati comunemente sono due:

- lista concatenata: tutti i blocchi liberi sono concatenati ed un puntatore indica il primo blocco libero. Necessità di proteggere il puntatore al primo blocco libero.
- mappa di bit: un disco di  $n$  blocchi richiede una mappa di  $n$  bit. I blocchi liberi si rappresentano con 1 sulla mappa, i blocchi allocati con 0 (o viceversa).

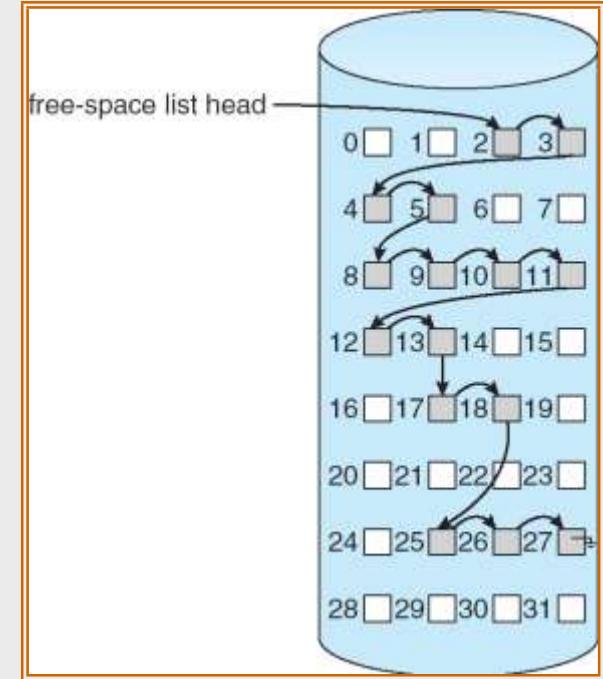
*Bit map richiede extra space*

*Es:*

$$\text{block size} = 2^{12} \text{ bytes (4KB)}$$

$$\text{disk size} = 2^9 2^{30} \text{ bytes (500 gigabyte)}$$

$$n = 2^{39}/2^{12} = 2^{27} \text{ bits (or 16M bytes)}$$



*Easy to get contiguous files*

Il metodo della lista concatenata è più efficiente se il disco è quasi pieno, poiché il numero di blocchi è molto limitato. Inoltre in queste condizioni, la mappa di bit potrebbe risultare spesso piena e richiedere continui accessi ad altre parti di mappa per assegnare blocchi liberi.

Se c'è abbastanza memoria principale per contenere la mappa di bit, è generalmente preferibile quest'ultimo metodo.

# Affidabilità del Fyle System

Il file system può contenere dati che non possono essere persi poiché di difficile o costoso recupero.

L'affidabilità del file system va perseguita spesso anche a scapito dell'efficienza o di economie.

I dispositivi di memorizzazione presentano spesso blocchi inutilizzabili:

- difetti di fabbricazione;
- usura;
- urti violenti.

I blocchi inutilizzabili non devono essere assegnati a nessun file, necessita una gestione appropriata.

# Affidabilità del Fyle System

Gestione dei blocchi danneggiati:

## ➊ soluzione hardware

- una parte di disco è riservata per tenere traccia dei blocchi rovinati e contiene dei blocchi di riserva;
- in fase di inizializzazione il controllore del dispositivo legge questi blocchi e li sostituisce con blocchi di riserva, memorizzandone l'avvenuta sostituzione;
- soluzione trasparente al file system.

## ➋ soluzione software

- il file system tiene traccia dei blocchi rovinati assegnandoli tutti ad un file speciale da non utilizzare

# Affidabilità del Fyle System BackUp

Nella pianificazione delle operazioni di backup vanno valutati principalmente:

- la frequenza con cui effettuare le copie di backup;
- i file che presentano la maggior necessità di essere copiati;
- la tecnica migliore per effettuare il backup.

# Affidabilità del Fyle System BackUp

*Frequenza:* la scelta va pianificata in base alla quantità di lavoro svolta nel tempo ed al costo di ripristino.

Generalmente la frequenza è di tipo:

- giornaliero;
- settimanale;
- mensile;
- annuale (solo per archivi storici).

# Affidabilità del Fyle System BackUp

*Scelta dei file:* la scelta va pianificata in base al contenuto dei file ed al costo di ripristino. Ad esempio:

- i file *critici* potrebbero essere raggruppati in directory dedicate delle quali si effettua copia con maggiore frequenza;
- la duplicazione di alcuni file *critici* può essere opportuna se il quantitativo di memoria sprecato è giustificabile.

# Affidabilità del Fyle System BackUp

*Tecnica di backup:* la scelta va pianificata in base al tempo disponibile ed al costo sopportabile per i supporti. Le tecniche di backup possono essere:

- backup su nastri dell'intero file system;
- backup incrementale (copia solo dei file nuovi o modificati dall'ultimo backup);
- duplicazione incrociata dei dati su due dispositivi: metà di ogni disco contiene i propri dati e l'altra metà contiene i dati dell'altro disco.

# Affidabilità del Fyle System Prestazioni

Accesso a disco operazione lenta.

Molti file system prevedono tecniche per ridurre il numero di accessi al disco:

- caching;
- riduzione dei movimenti del braccio del disco (soluzione soft);
- riduzione dei movimenti del braccio del disco (soluzione hard).

# Affidabilità del Fyle System Prestazioni

***block cache***: un certo numero di blocchi in memoria vengono riservati per contenere blocchi di disco.

- ad ogni richiesta di un blocco viene scandita la tabella dei blocchi in memoria per verificarne la presenza;
- se non è presente il blocco viene caricato dal disco, copiato nella chache e poi reso disponibile alle richieste;
- la lista può essere gestita con normali algoritmi di rimpiazzamento delle pagine come FIFO e LRU;

UNIX ogni 30 secondi riscrive tutti i blocchi modificati su disco (oppure a richiesta dell'utente),

WINDOWS riscrive immediatamente ogni blocco modificato.

# Affidabilità del Fyle System

## Prestazioni

Riduzione dei movimenti del braccio del disco (**soluzione soft**):

- *raggruppamento di blocchi in uso*. Andando a stimare i blocchi a cui si accede con maggiore probabilità in sequenza al fine di ridurre i movimenti della testina.
- *posizione dell'indice dei blocchi*. La lettura di un file richiede comunque almeno 2 accessi (uno all'i-node ed uno al blocco). Il posizionamento degli i-node o della tabella di allocazione dei file al centro del disco per ridurre la distanza tra questi ed i blocchi a cui questi fanno riferimento.
- *partizioni aperte*. Si tende ad avvicinare il più possibile gli i-node ai blocchi cui fanno riferimento nel seguente modo:
  - ✖ divisione del disco in gruppi di cilindri, ognuno con i propri i-node e la propria lista dei blocchi liberi;
  - ✖ quando una partizione esaurisce i blocchi, un blocco può essere allocato in una partizione diversa.

# Affidabilità del Fyle System Prestazioni

## *Interleave.*

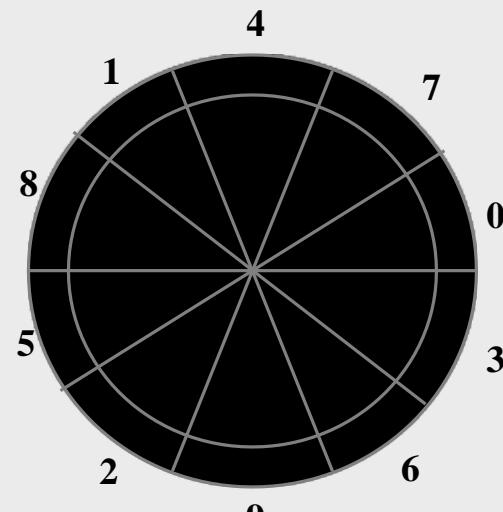
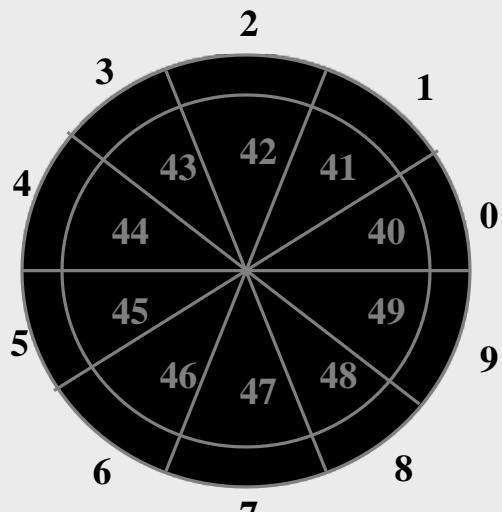
un processo richiede un tempo  $t_b$  per richiedere e recuperare un blocco.  
In situazioni (frequenti) di richieste consecutive di blocchi sequenziali può accadere che:

- ✖ il blocco richiesto può essere già passato sotto la testina;
- ✖ bisogna attendere una rotazione intera.

**La tecnica di interleaving alloca i blocchi fisici del disco in ordine non sequenziale stretto, ma sequenziale con salti (fattore di interleaving).**

# Affidabilità del Fyle System Prestazioni

fattore di interleaving :  $\frac{1}{4}$  di giro



# Affidabilità del Fyle System

## Prestazioni

La riduzione dei movimenti del braccio del disco (**soluzione hard**) avviene mediante la *schedulazione del braccio del disco*. Il tempo necessario per leggere un blocco di disco dipende da:

- seek time: spostamento della testina sul cilindro;
- latency time: tempo di rotazione per raggiungere il settore richiesto;
- transfer time: tempo di trasferimento.

# Affidabilità del Fyle System Prestazioni

Il ritardo predominante è dato dal seek time.

Algoritmi per ottimizzare il movimento del braccio:

- FCFS (First Come First Served):

- ✖ serve le richieste nell'ordine in cui giungono;
- ✖ non richiede nessun specifico supporto hardware o software;
- ✖ è l'algoritmo più corretto nei riguardi delle richieste
- ✖ non ottimale;

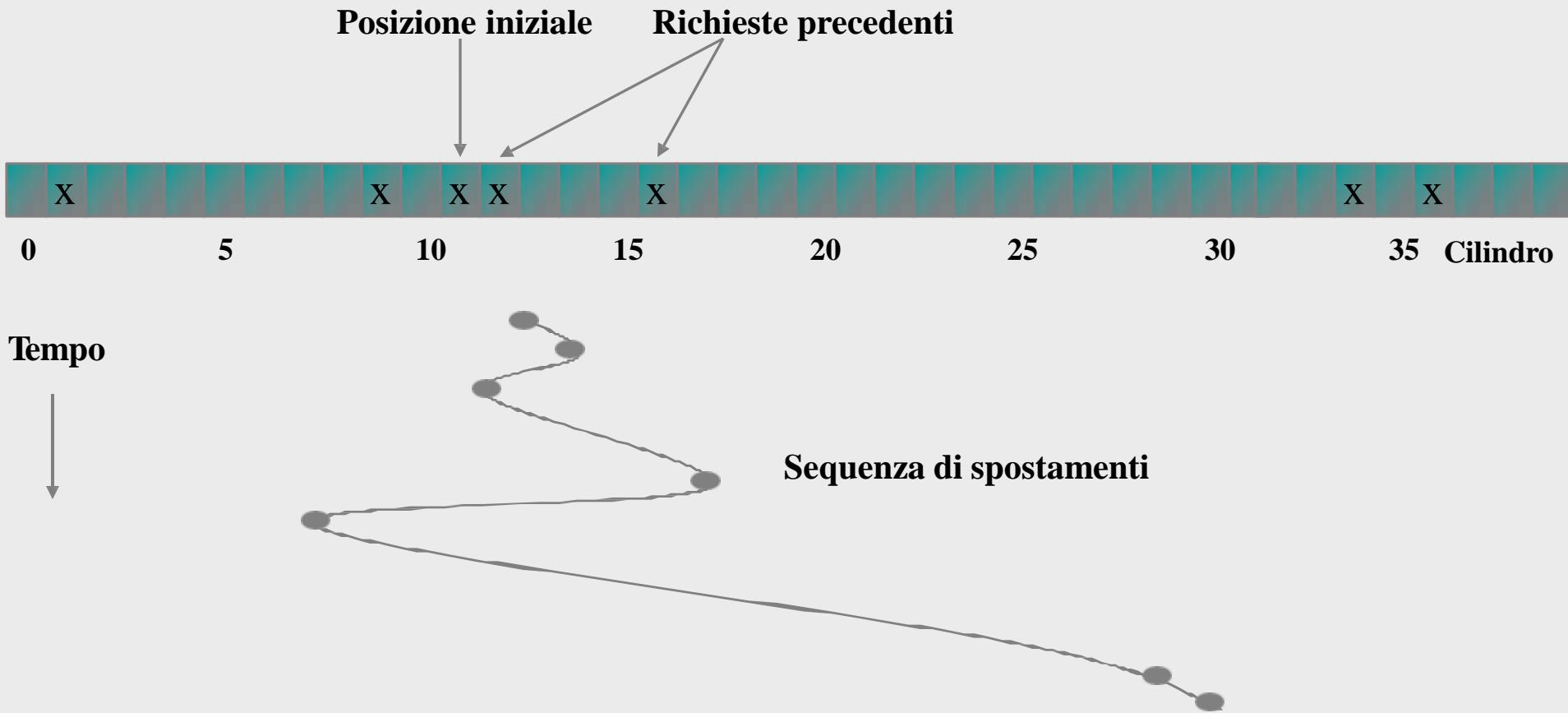
# Affidabilità del Fyle System Prestazioni

## • SSF (Shortest Seek First):

- ✖ mantenendo in tabella tutte le richieste pendenti, l'SSF si sposta sempre sulla richiesta più vicina a quella appena servita;
- ✖ riduce di circa la metà gli spostamenti sul disco rispetto al FCFS;
- ✖ presenta un problema di localizzazione delle richieste.

# Affidabilità del Fyle System Prestazioni

Algoritmo SSF per la schedulazione dei movimenti del braccio



# Affidabilità del Fyle System

## Prestazioni

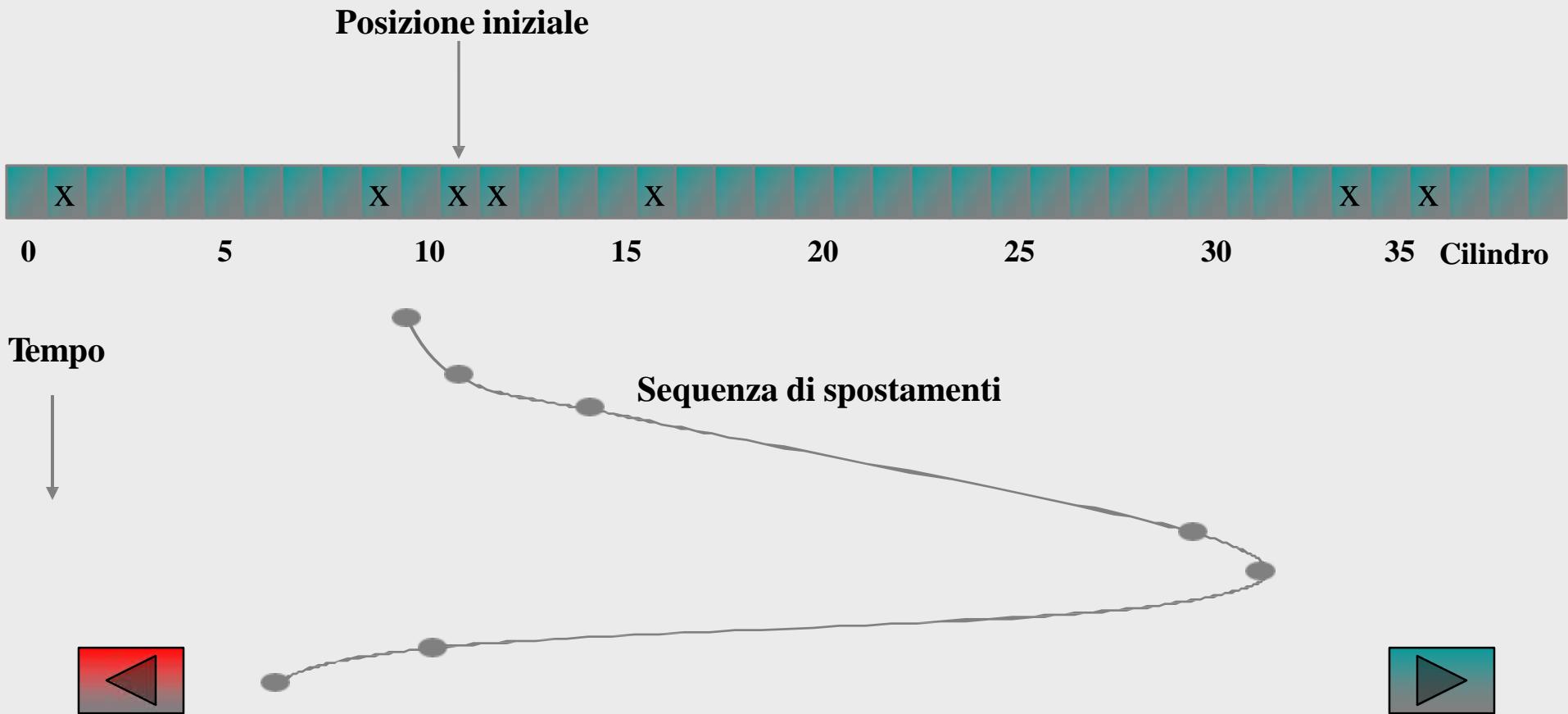
- algoritmo dell'ascensore:

- ✖ mantiene in tabella le richieste pendenti;
- ✖ un bit *up/down* indica la direzione attuale del braccio del disco;
- ✖ serve le richieste più vicine proseguendo sempre nella stessa direzione;
- ✖ cambia direzione quando raggiunge l'estremità o quando non ci sono più richieste nella direzione attuale;

# Affidabilità del Fyle System

## Prestazioni

## **Algoritmo dell'ascensore**



# Identificazione degli utenti

I meccanismi di protezione degli archivi sono basati sul presupposto della sicura identificazione degli utenti.

- Nei sistemi interattivi multi-utente, il progenitore di tutti i processi di un generico utente è quello che viene generato quando esso si connette al sistema attraverso un terminale.
- Per ogni processo diverso dal progenitore, la responsabilità di definire il proprietario spetta esclusivamente al sistema operativo
- la sicurezza degli archivi è fondata sulla corretta definizione del proprietario del processo progenitore, che a sua volta dipende dalla sicura identificazione dell'utente che si connette attraverso il terminale.
- La tecnica comunemente usata è quella della parola d'ordine (password),