1. Cenni di architettura dei calcolatori

Andrea Marongiu

(andrea.marongiu@unimore.it)

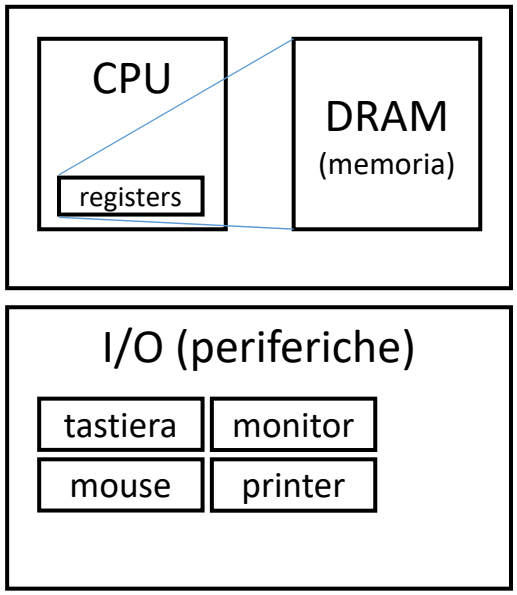
Paolo Valente

UNIMORE

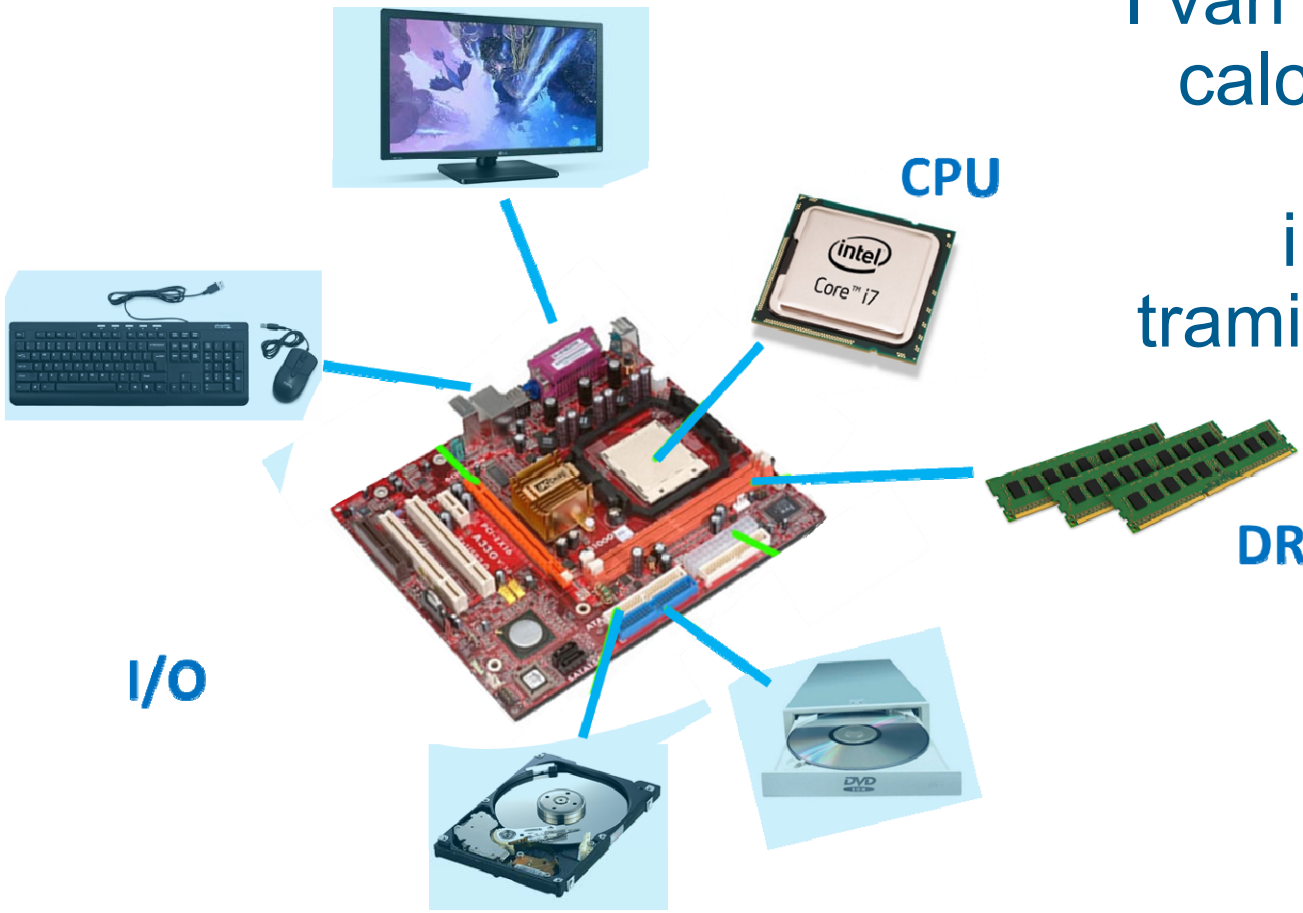
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Schema di un calcolatore

- Partiamo dal disegnare uno schema semplificato dell'architettura di un calcolatore
- Considereremo solo
  - **CPU**
  - **Memoria** (gerarchia)
  - Dispositivi di input/output (**I/O**)



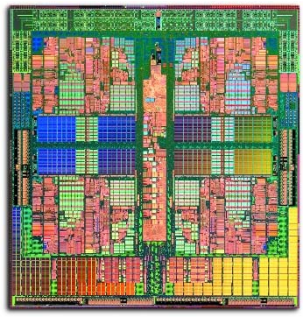
Motherboard



I vari blocchi di un calcolatore sono fisicamente interconnessi tramite la **scheda madre**

Processore

- Central Processing Unit (CPU)



- Tutte le operazioni di elaborazione delle informazioni effettuate da un calcolatore sono
  - svolte direttamente dal processore, oppure
  - svolte da altri componenti dietro comando del processore

Operazioni (1/2)

- Un processore è in grado di compiere solo operazioni molto semplici:
  - lettura/scrittura/copia di una o più *celle di memoria*
  - somma/sottrazione/moltiplicazione/divisione del contenuto di una o più celle di memoria
  - lettura/scrittura in zone di memoria 'speciali' per pilotare dispositivi di ingresso/uscita (ad esempio schede video)
  - altre semplici operazioni sulle celle di memoria

Operazioni (2/2)

- Tipicamente un processore riesce a lavorare su un certo numero di celle contigue alla volta. Tale sequenza di celle è detta **parola di macchina (machine word)**
  - Si dice che un processore ha una architettura a 16, 32 oppure 64 bit se lavora su parole da 2, 4 oppure 8 byte

Linguaggio macchina

- Ogni processore è caratterizzato da un proprio insieme di **istruzioni**, tramite le quali è possibile fargli svolgere le precedenti operazioni
- L'insieme delle istruzioni di un processore viene chiamato **linguaggio macchina** di quel processore
- Ogni istruzione è identificata da una certa configurazione di bit

Instruction Set Architecture (ISA)

Da Wikipedia

- In informatica ed elettronica un instruction set, o Instruction Set Architecture (ISA), (in lingua italiana insieme d'istruzioni) descrive que**gli aspetti dell'architettura di un calcolatore che sono visibili al programmatore**.

- Si tratta di fatto dell'insieme di istruzioni base che il processore può compiere e che costituiscono dunque il suo linguaggio macchina, a partire dal quale vengono scritti i relativi programmi nei vari linguaggi di programmazione a più alto livello di astrazione.

Esempio programma

PC	Instruction	Meaning	ET
81d0	mov r12, #0	i = 0	1
81d4	mul r1, r12, r12	r1 = i×i	3
81d8	mov r2, r12, lsl #3	r2 = i×8	1
81dc	mov r3, #15	r3 = 15	1
81e0	mla r3, r1, r3, r2	r3 = r1×r3 + r2	3
81e4	add r3, r3, #79	r3 = r3 + 79	1
81e8	str r3, [r0, r12, lsl #2]	dest[i] = r3	2
81ec	add r12, r12, #1	i++	1
81f0	cmp r12, #64	is i<64?	1
81f4	bne 81d4	yes? branch 81d4	1
81f8	bx lr	return	1

ISA: Mnemonici comprensibili per l'uomo

Codice binario: la rappresentazione del programma compresa dalla CPU

0011001000110110

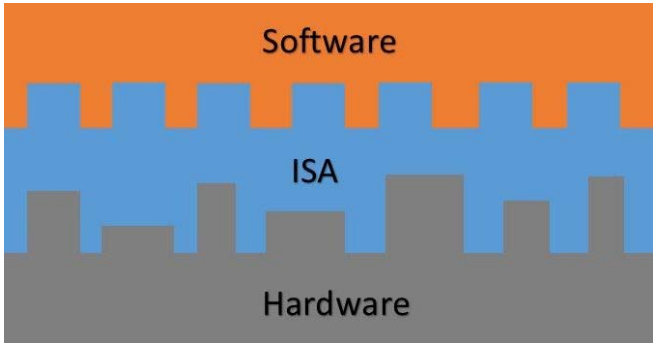
0101010100100011

0011011001010101

1110001111100011

...

1001101011100011



Astrazione (1/4)

In definitiva, data la semplicità delle istruzioni e dei dati su cui lavora un processore si ha che:

- scrivere (interamente) in linguaggio macchina un programma che faccia cose complesse
- Es. un sistema operativo o anche più semplicemente un programma che deve disegnare/aggiornare un'interfaccia grafica ed usarla per interagire con gli utenti

diviene un lavoro estremamente impegnativo e costoso

Astrazione (2/4)

Questo è fondamentalmente il motivo per cui sono stati inventati moltissimi altri **linguaggi** cosiddetti **ad alto livello**, che sono molto più 'vicini' al linguaggio umano rispetto al linguaggio macchina

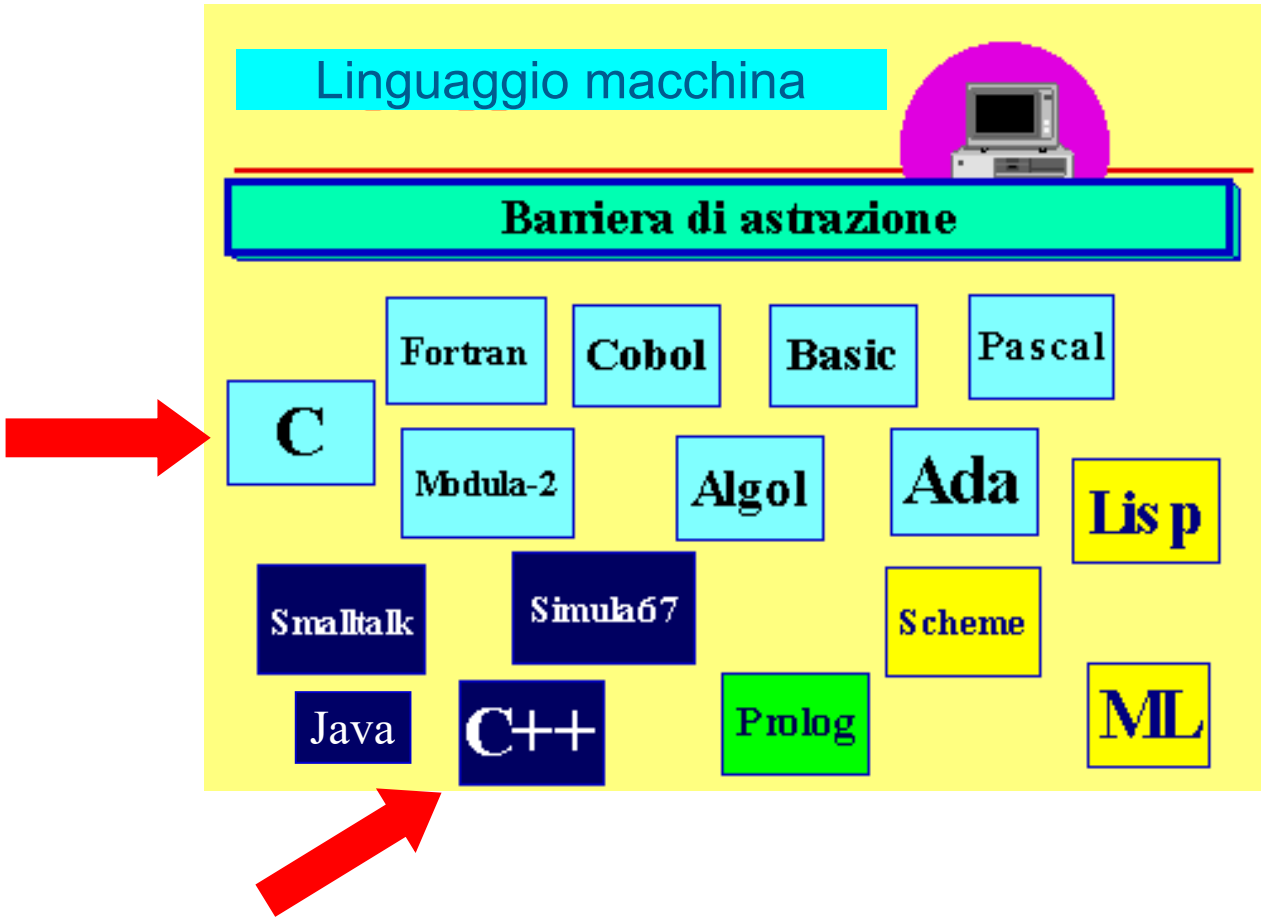
Tali linguaggi si basano sul concetto di ***astrazione*** dalla macchina sottostante: **astraggono dai dettagli**, cosiddetti di *basso livello*

- Ad esempio, non serve conoscere l'ISA del processore usato per programmarlo, bastano delle istruzioni più vicine al modo di ragionare umano.

Astrazione (3/4)

In generale, dato un problema da risolvere, disporre di dati ed operazioni più astratti e complessi permette di descrivere in modo molto più semplice e chiaro gli elementi del problema ed i passi che si debbono effettuare

Astrazione (4/4)



Livello di astrazione crescente

Linguaggi di programmazione

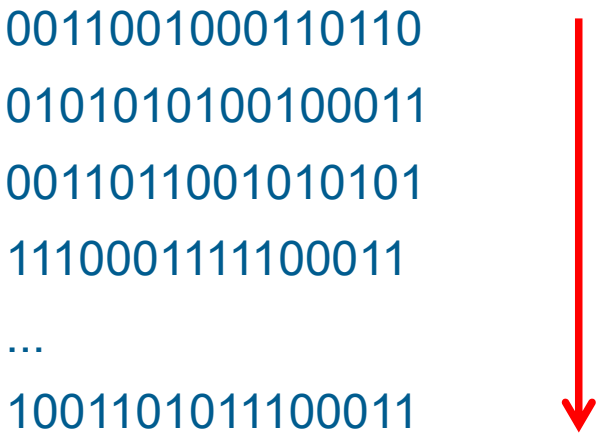
- Il C/C++ è quindi un linguaggio di alto livello
- Il fatto di non coincidere con il linguaggio macchina di nessun processore ha però un prezzo
- Per poter essere eseguito da un calcolatore, un programma scritto in C/C++ va prima tradotto nel linguaggio macchina del processore del calcolatore su cui lo vogliamo eseguire
- Questa operazione viene comunemente chiamata **compilazione**, ed i programmi che la eseguono vengono chiamati **compilatori**

Programma

- Per far eseguire un programma ad un processore, basta
  - memorizzare da qualche parte nella memoria la sequenza di configurazioni di bit relativa alle istruzioni da eseguire
  - dire al processore a che indirizzo si trova la prima di tali istruzioni
- Il processore eseguirà, una dopo l'altra, le istruzioni che trova a partire da tale indirizzo

Ordine di esecuzione (1/2)

- Ordine di esecuzione *predefinito* delle istruzioni: l'una dopo l'altra



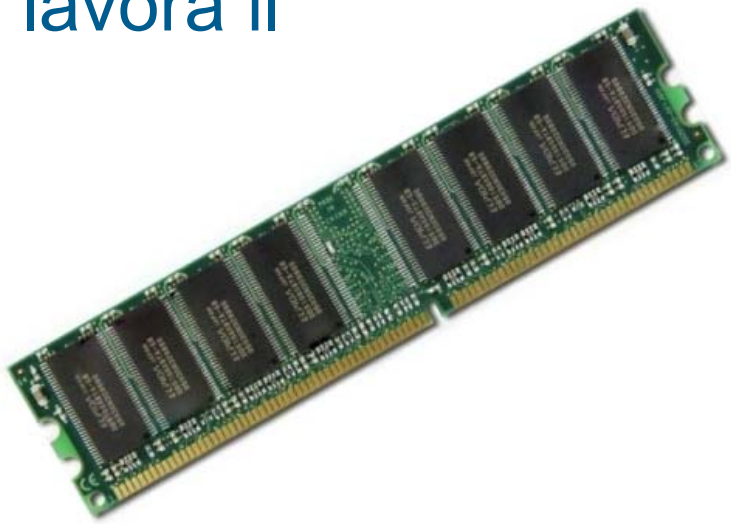
Ordine di esecuzione (2/2)

L'ordine con cui sono eseguite le istruzioni cambia solo se vengono incontrate speciali istruzioni di salto verso un diverso indirizzo

Un salto può anche avvenire all'*indietro*, ossia verso un indirizzo inferiore rispetto a quello in cui si trova l'istruzione di salto stessa

Memoria principale

Definiamo **memoria** (principale) di un elaboratore il contenitore in cui sono memorizzati tutti i dati su cui lavora il processore



Memoria principale e celle

- Possiamo schematizzare la memoria come una sequenza contigua di **celle** (chiamate anche **locazioni di memoria**)

- Ciascuna cella fornisce l'**unità minima di memorizzazione**, ossia l'elemento più piccolo in cui si può memorizzare un'informazione

Contenuto cella

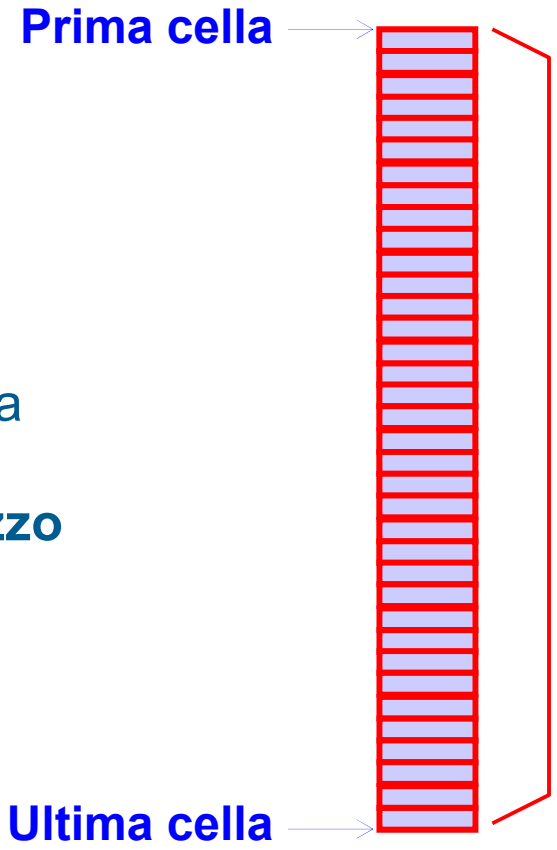
- Ogni cella contiene un *byte*, ossia una sequenza di *bit* (cifre binarie)
  - *Tipicamente un byte è costituito da 8 bit*
    Esempio: 01100101

- Tutte le celle hanno quindi la stessa dimensione in termini di numero di bit

Schema memoria

Memoria calcolatore

Ciascuna cella è **univocamente** individuata mediante un numero naturale, chiamato **indirizzo** della cella



Celle di memoria e numeri

- I bit contenuti in una cella possono essere utilizzati per memorizzare un numero
  - Il numero è rappresentato nella cosiddetta **notazione binaria**
  - *https://it.wikipedia.org/wiki/Sistema_numerico_binario*

- Senza entrare nei dettagli della notazione binaria, facciamo alcuni esempi di come si ottiene questo risultato

Notazione binaria (1/4)

- Sistema numerico posizionale in base 2
- Solo due simboli, 0 e 1, invece delle dieci cifre utilizzate dal sistema numerico decimale

- Quanti numeri si possono rappresentare con 1byte?

- $(numero\ di\ simboli)^{(numero\ di\ bit)} = 2^8 = 256$

Notazione binaria (2/4)

- Quali numeri?
- **Es. interi positivi**
- Facciamo corrispondere un numero ad ogni combinazione (configurazione) di bit

00000000	0
00000001	1
00000010	2
00000011	3
...	
11111111	255

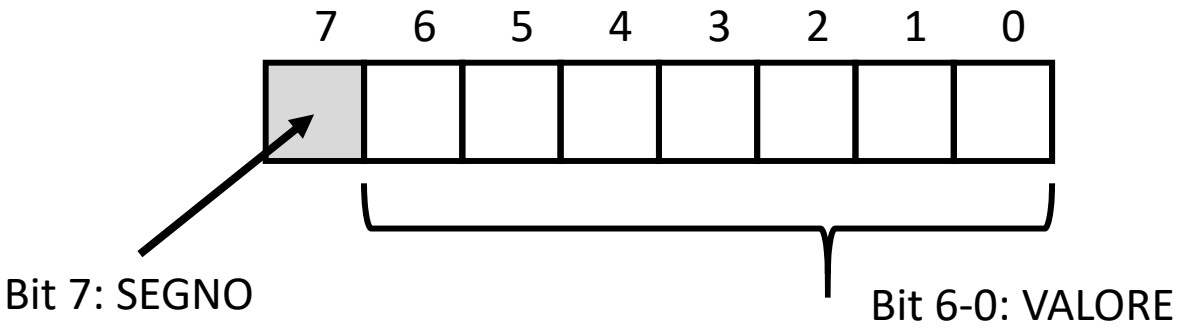
- Con 8 bit si possono rappresentare i numeri nell'intervallo

$$[0 \,..\, 255]$$

Notazione binaria (3/4)

- Con una tecnica simile si possono rappresentare anche **numeri negativi**
- facendo corrispondere un certo sottoinsieme delle possibili configurazioni di bit ai numeri positivi, e l'altro sottoinsieme ai numeri negativi
- Idea più semplice
  - Utilizzare un bit per il segno

Notazione binaria (4/4)



- $(numero\ di\ simboli)^{(numero\ di\ bit)} = 2^7 = 128$
- Con un bit dedicato al segno si possono rappresentare i numeri nell'intervallo

$$[-127\,..+127]$$

Astrazione (cont.)

- Un altro tipico esempio di **astrazione** realizzato dai linguaggi di programmazione sono i **tipi di dato**
  - si astrae dalle singole celle di memoria: non si vedono più le singole celle di memoria in cui sono memorizzati i numeri
  - si può quindi ragionare e scrivere il programma direttamente in termini di numeri di un dato tipo
  - si lavora ad alto livello, senza preoccuparsi di come e dove saranno realmente memorizzati e manipolati tali numeri a basso livello

Tipi di dato (1/5)

- Ogni linguaggio di programmazione definisce diversi **tipi di dato**, i quali usano un diverso numero di bit, adatto per operazioni specifiche

- Es., C/C++
- `char`: caratteri ASCII, **8 bit**
- `int`: numeri interi, **32 bit**
- `float`: numeri frazionari a virgola mobile, **32bit**
- `double`: numeri frazionari a virgola mobile, doppia precisione, **64 bit**

Tipi di dato (3/5)

```
int variabile = 5;
```

- Ci preoccupiamo di come memorizzare i valori in una sequenza di celle?
- Ci preoccupiamo di dove memorizzare esattamente in memoria tale sequenza di celle?

No, usiamo il **tipo `int`** nel suo significato astratto di *contenitore di numeri interi*

Tipi di dato (4/5)

DOMANDA

- E' possibile memorizzare il contenuto di una variabile di tipo `int` all'interno di una singola cella di memoria?

RISPOSTA

- Se il valore è più grande di 255 allora certamente no!

Tipi di dato (4/5)

- L'astrazione dei tipi di dato gestisce per noi in maniera trasparente l'interpretazione del contenuto di più celle di memoria contigue come un singolo dato
- Cioè, per rappresentare numeri più grandi di quelli rappresentabili con una sola cella, si accorpano più celle consecutive
  - Es., tutte le configurazioni possibili di bit di una sequenza di due o quattro celle contigue

Tipi di dato (5/5)

- Una variabile di tipo **`int`** è tipicamente rappresentata su 4 celle (byte) consecutive

00000001	16777216-33554431
…	
…	
…	
11111111	4278190080-4294967295

Cella (byte) 3

00000001	65536-131071
…	
…	
…	
…	
11111111	16711680-16777215

Cella (byte) 2

00000001	256-511
00000010	512-767
00000011	768-1023
00000100	1024-1279
00000101	1280-1535
…	
11111111	65280-65535

Cella (byte) 1

00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
…	
11111111	255

Cella (byte) 0

- Col tipo **`int`** si possono esprimere $2^{32} = 4294967296$ numeri

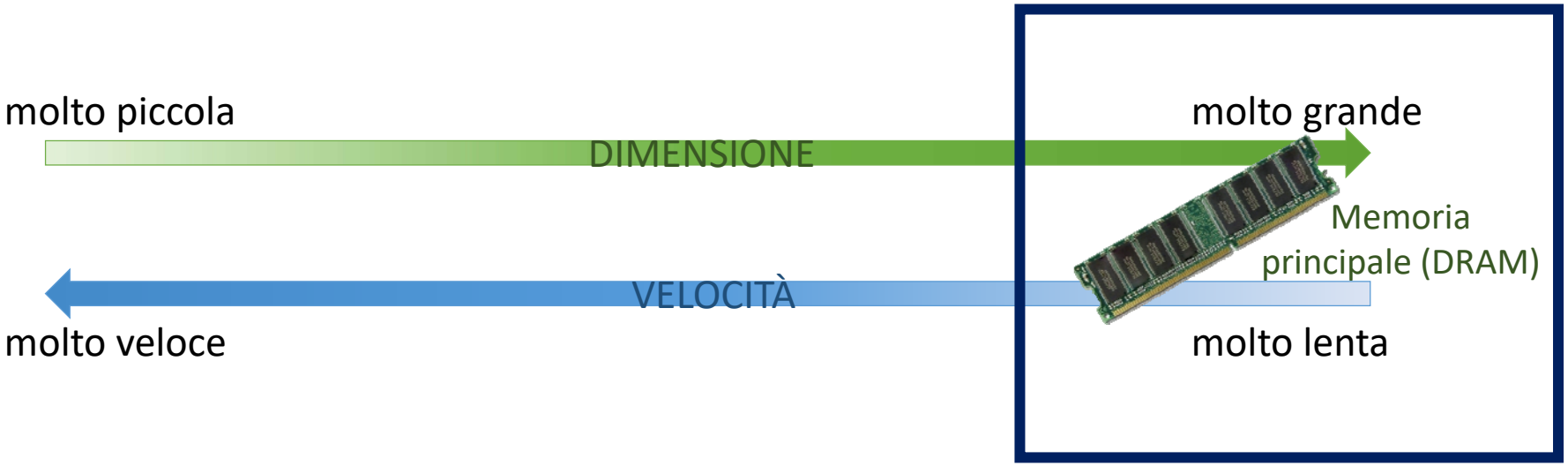
Gerarchia di memoria (1/4)

- La memoria principale è grande abbastanza da contenere svariati programmi, ma ha due limitazioni:

1. È **lenta**
   - Considerando le attuali tecnologie per lo sviluppo di memorie esiste un compromesso tra la loro dimensione e la loro velocità
2. È **volatile**
   - È capace di conservare l'informazione solo finché il computer rimane acceso

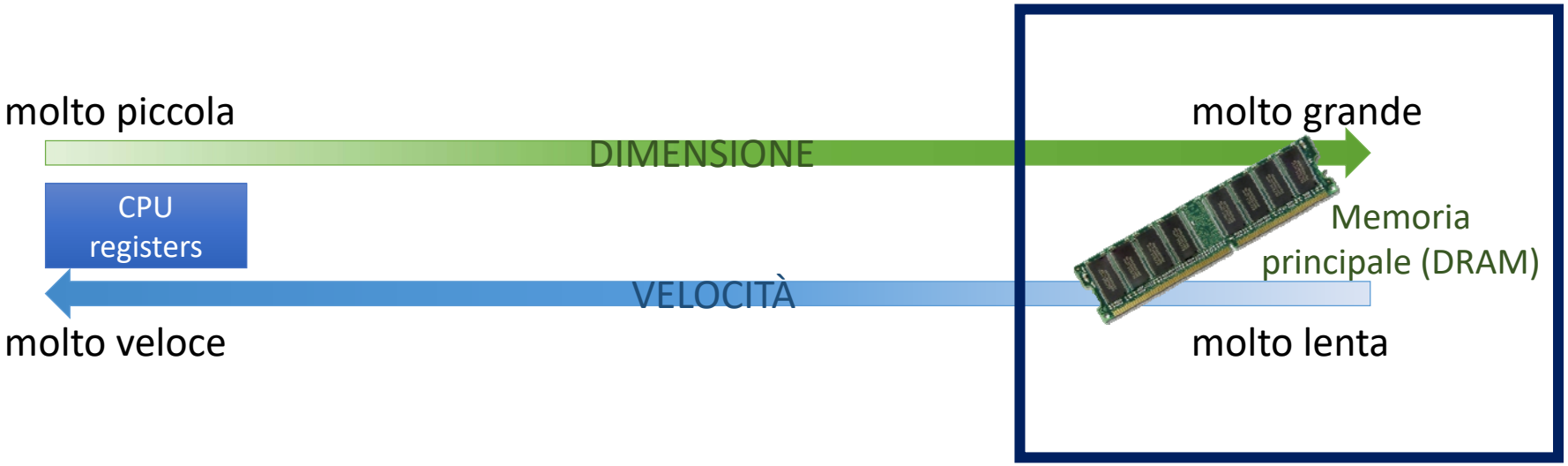
Gerarchia di memoria (2/4)

- Per ovviare alla lentezza della memoria principale si costruiscono delle gerarchie
- Le tecnologie più veloci sono le più costose, quindi ci si può permettere solo piccole memorie veloci



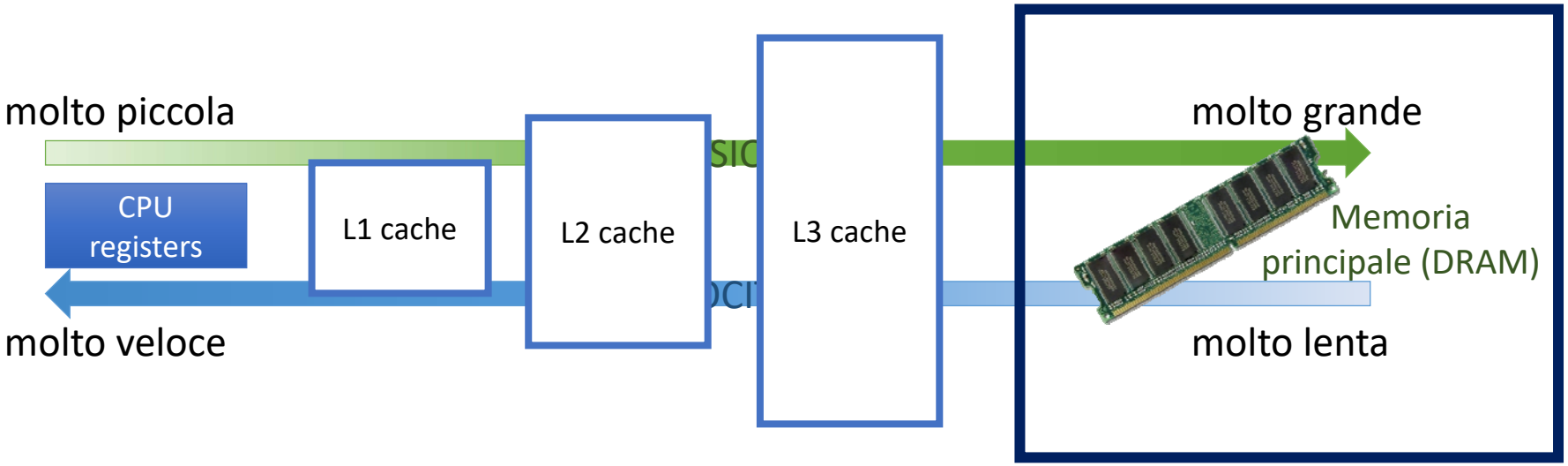
Gerarchia di memoria (3/4)

- La memoria più veloce è usata per realizzare i **registri** dentro la CPU
- ..ma è piccolissima, occorre aggiornarne in continuazione il contenuto



Gerarchia di memoria (4/4)

- Tra i **registri** e la **memoria principale** ci sono diversi livelli di **CACHE**
- una memoria gestita trasparentemente dal sistema
  - Porta vicino ai registri i dati necessari
  - Nasconde l'alta **latenza** della memoria principale



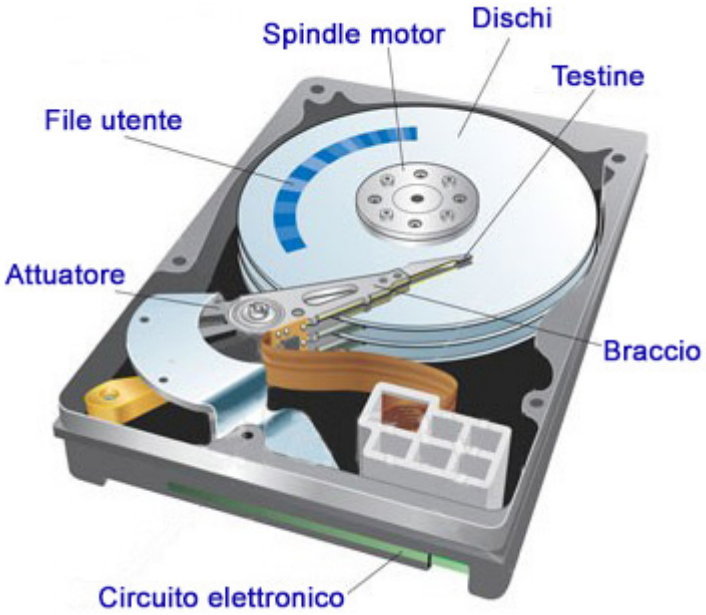
Memorie di massa (1/5)

- Come gestire il problema della volatilità della memoria principale?

- Per memorizzare le informazioni in modo permanente si usano **dispositivi di memorizzazione di massa**

Memorie di massa (2/5)

- L'esempio più rappresentativo è il **disco fisso**, o **hard disk**



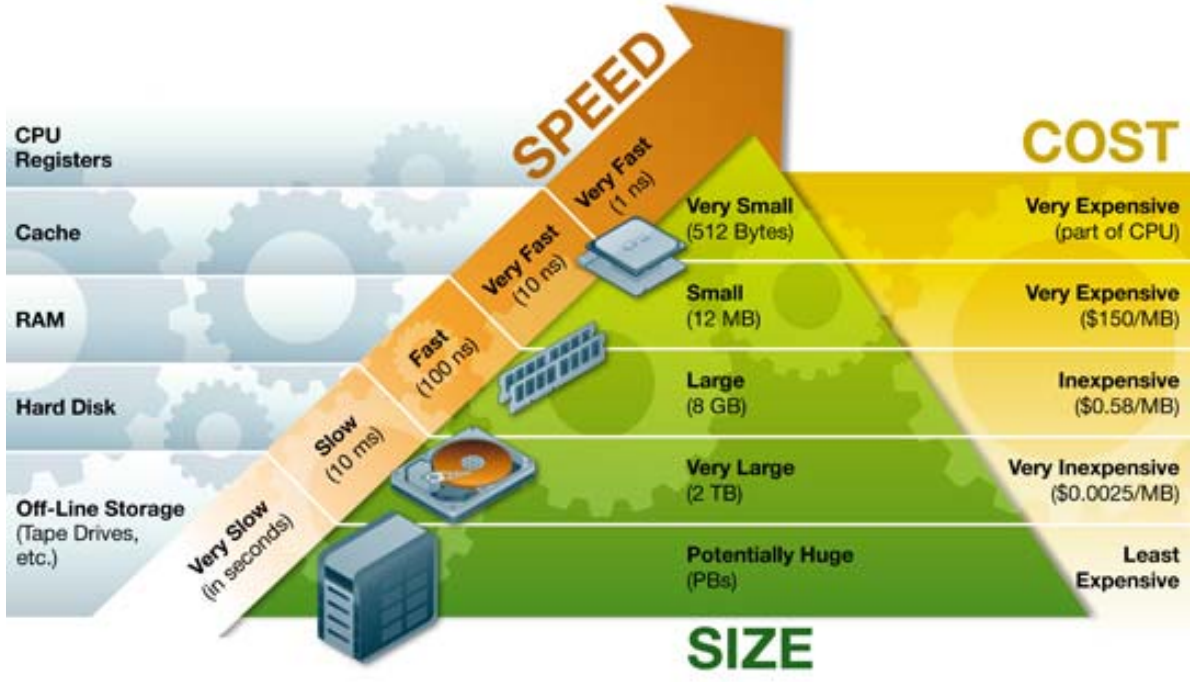
https://www.youtube.com/watch?v=3owqvmMf6No&feature=related

Memorie di massa (3/5)

- Oltre ad essere non-volatili, le memorie di massa hanno una dimensione molto maggiore rispetto alla memoria principale (DRAM)
- Ma quanto sono veloci? (o lente)?
- Di fatto, le memorie di massa costituiscono parte della gerarchia di memoria, conservando il compromesso velocità/dimensione

Memorie di massa (4/5)

Gerarchia di memoria completa di un calcolatore



Memorie di massa (5/5)

CURIOSITÀ

- L’evoluzione dei sistemi di storage: dalle schede forate al cloud