Richiami di Architettura degli Elaboratori dell'Informazione

Scopo di questa parte del corso e' richiamare i concetti principali delle architetture degli elaboratori e, in particolare quelle caratteristiche hardware che vengono utilizzate dai sistemi operativi a scopo di protezione e di gestione della memoria virtuale.

A questa parte del corso non corrisponde strettamente alcun capitolo del libro di testo consigliato.

Ove necessario, verra' utilizzata come esempio l'architettura IA-32, cioe' quella del processore Intel 386. Tale architettura ha delle fortissime analogie con i piu' moderni processori Intel e AMD attualmente utilizzati, di cui costituisce la base comune.
I concetti descritti, pero', sono validi anche per i moderni processori di altre famiglie, ad esempio i processori ARM.

Infine, richiameremo alcuni concetti della runtime machine C, ovvero ricorderemo come un programma scritto in linguaggio ANSI C viene caricato in memoria ed eseguito, in particolare accennando all'uso dello stack e alla convenzione di chiamata a funzione in linguaggio C.

Sistema di Elaborazione dell'Informazione (Computer)

- Un **Sistema di Elaborazione dell'Informazione** (o **Elaboratore Elettronico** o **Computer**) è una macchina *digitale*, *elettronica*, *automatica* e *programmabile* capace di effettuare trasformazioni o elaborazioni sui dati.

- **Digitale**: l'informazione è rappresentata in forma numerica discreta.

- **Elettronica**: la logica di memorizzazione e manipolazione dell'informazione sono implementate con tecnologie di tipo elettronico.

- **Automatica**: è in grado di eseguire una successione di operazioni in modo autonomo (senza intervento di un operatore umano).

- **Programmabile**: un operatore umano (o elettronico ma con decisione iniziale di un umano) può modificare "facilmente" la successione di operazioni da eseguire.

Le operazioni che l'elaboratore deve eseguire sono "comandate" all'elaboratore stesso mediante delle istruzioni espresse in una forma (un linguaggio) comprensibile all'elaboratore. Le istruzioni sono delle piccole sequenze di bit.

Le istruzioni (**il programma**) che l'elaboratore esegue costituiscono il **software**.

Software vs. hardware

Il programma (le istruzioni) che l'elaboratore deve eseguire potrebbero anche essere codificate nell'hardware, ma si perderebbe la possibilità di modificare il programma.

Il vantaggio di poter modificare il programma consiste nella flessibilità d'uso del computer. La flessibilità del computer ne fa uno strumento general-purpose, cioè uno strumento di uso generico, che viene specificato e specializzato dal software.

Lo svantaggio del computer rispetto ad un dispositivo esclusivamente hardware risiede nella maggior lentezza di esecuzione.

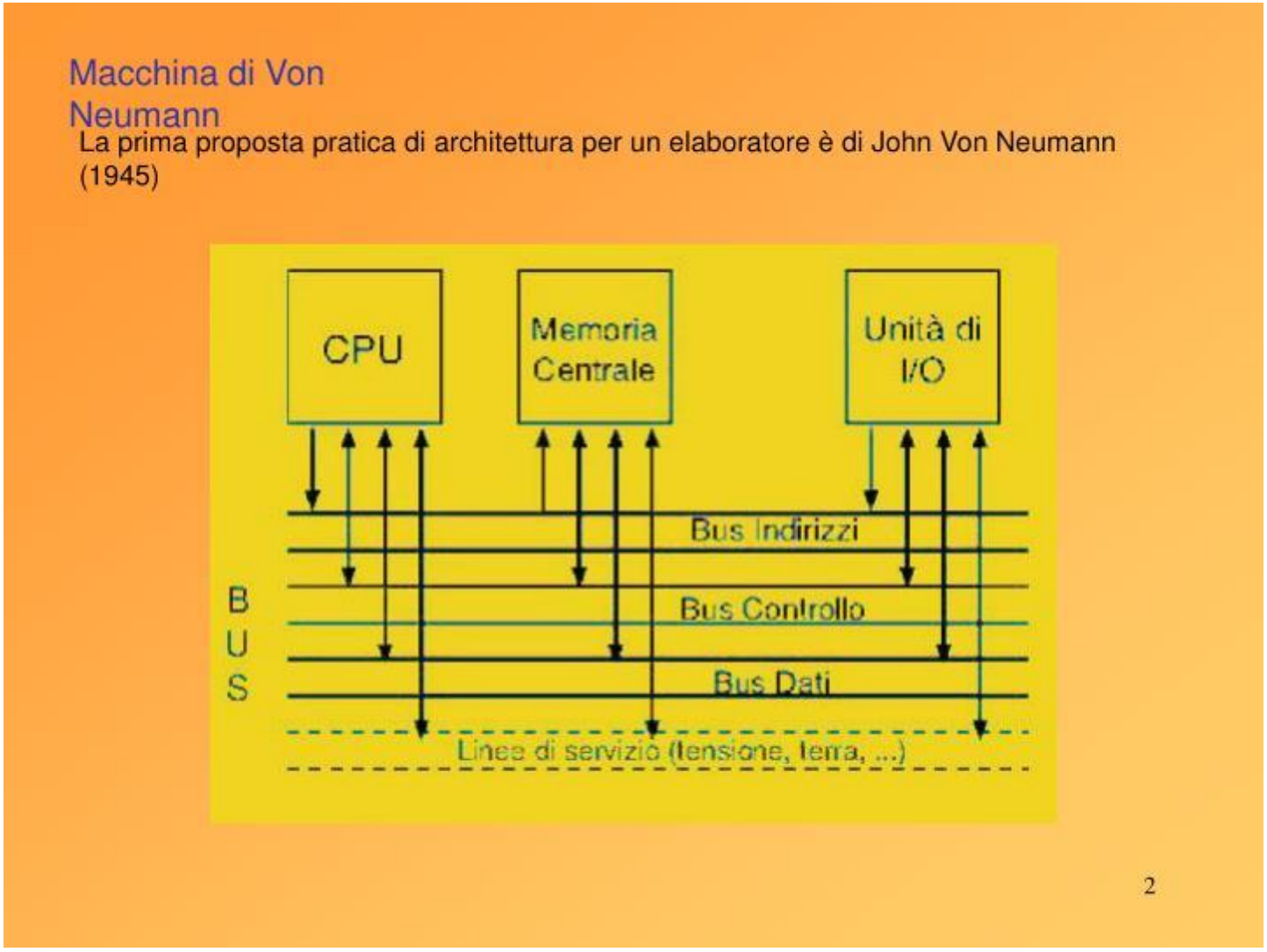
Requisiti di un Elaboratore

Dove mantenere tutto il programma da eseguire?	memoria
Dove mantenere tutti i dati su cui operare?	memoria
Dove eseguire le singole istruzioni per operare sui singoli dati?	CPU(ALU,reg.)
Come trasferire le istruzioni ed i dati per l'esecuzione?	BUS
Dove mantenere i dati quando l'elaboratore è spento?	Dischi
Come far interagire l'utente con l'elaboratore?	Input/Output

Un elaboratore *programmabile* è caratterizzato dalla presenza di CPU e memoria separate. La CPU preleva le istruzioni dalla memoria e le esegue !!!!

Macchina di Von Neumann

La struttura base di un computer rimane quella della Macchina di Von Neumann, con CPU e memoria separate, interconnesse da alcuni BUS, quello degli indirizzi, quello di controllo e quello dei dati. La CPU preleva le istruzioni dalla memoria e le esegue.



Organizzazione di un Computer

Un Computer in generale è organizzato con una struttura “gerarchica”, ossia a livelli:

1. **Hardware**: composto dai componenti *fisici* del sistema.

2. **Firmware**: insieme di microprogrammi che svolgono operazioni più o meno complesse nei singoli componenti. Può essere modificato solo con tecniche speciali.
   es: BIOS, risiede nella ROM, esegue nella CPU al boot.
   es: Firmware delle schede di rete wireless, eseguito nelle schede di rete, coordina la trasmissione dei bit sulla portante radio

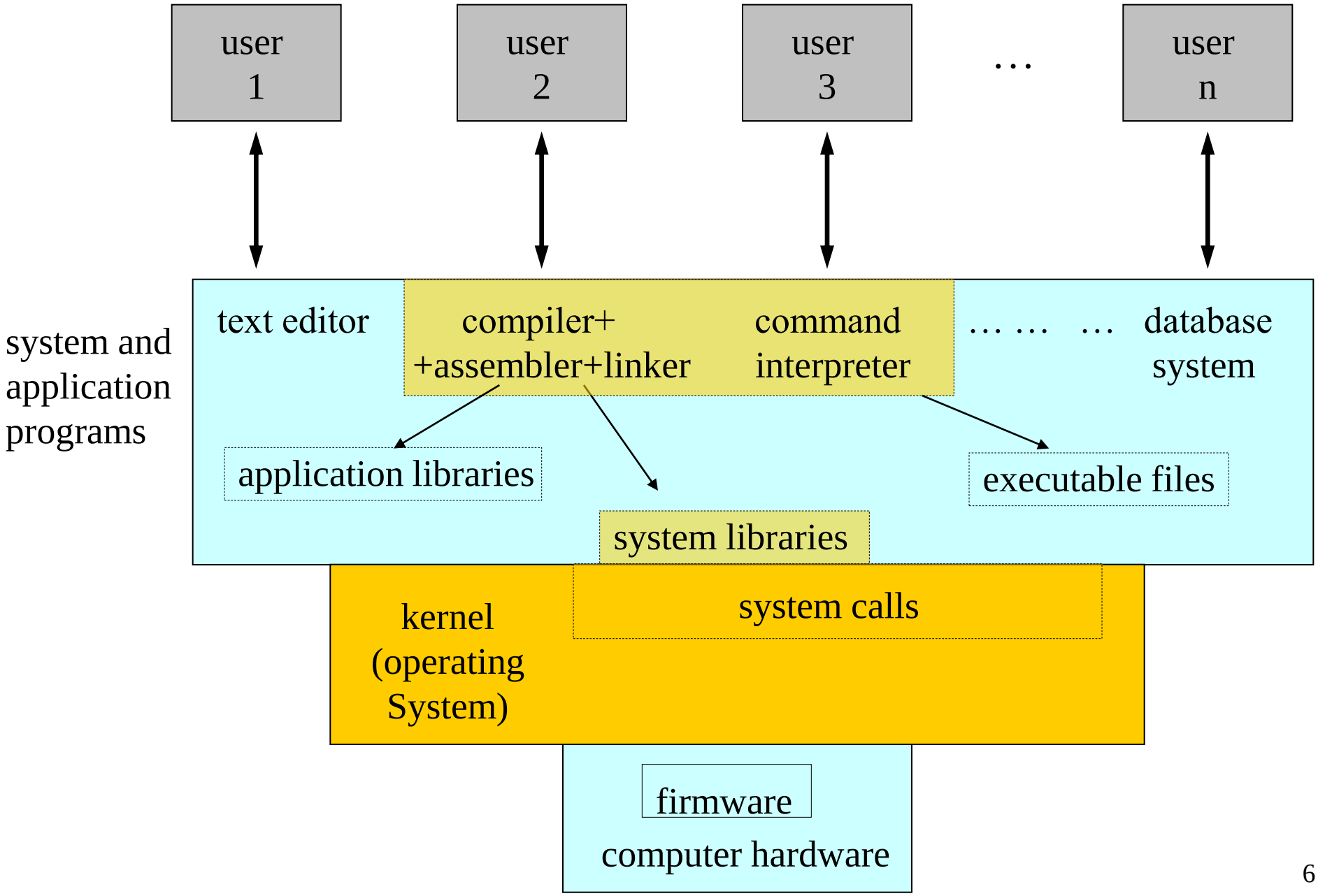
3. **Software**: i programmi eseguiti dal sistema, che a loro volta si suddividono in:

   3.1 Software di base: più conosciuto con il termine di *Sistema Operativo* , è dedicato alla gestione delle risorse hardware, alla gestione di più utenti, alle gestione di più processi (i programmi quando sono in esecuzione).

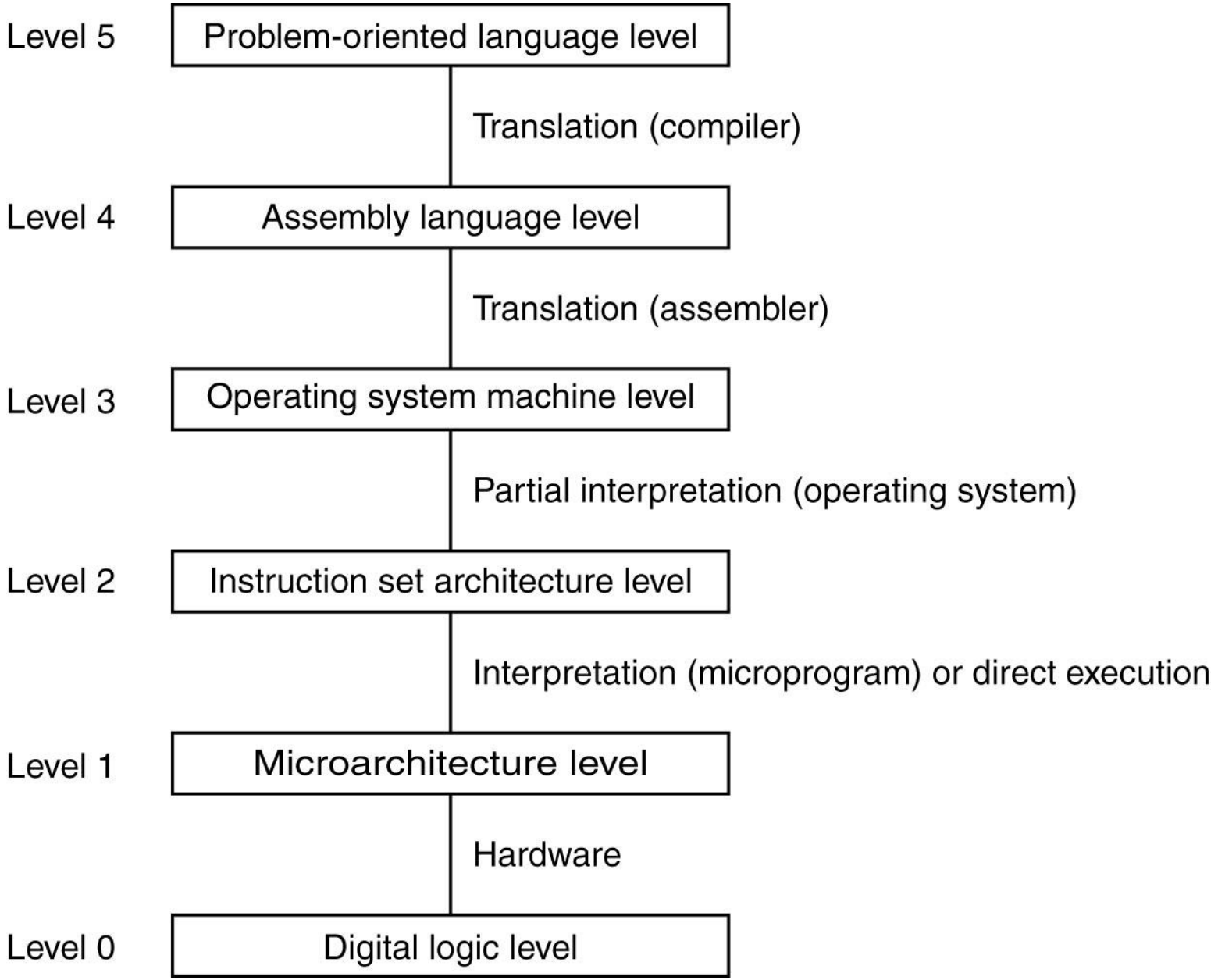
   3.2 Software applicativo: insieme dei programmi dedicati a specifiche esigenze applicative da parte dell’*utente*.

4. **Utente**: entita’ capace di qualunque atto dannoso, più o meno involontario (**utonto**?)

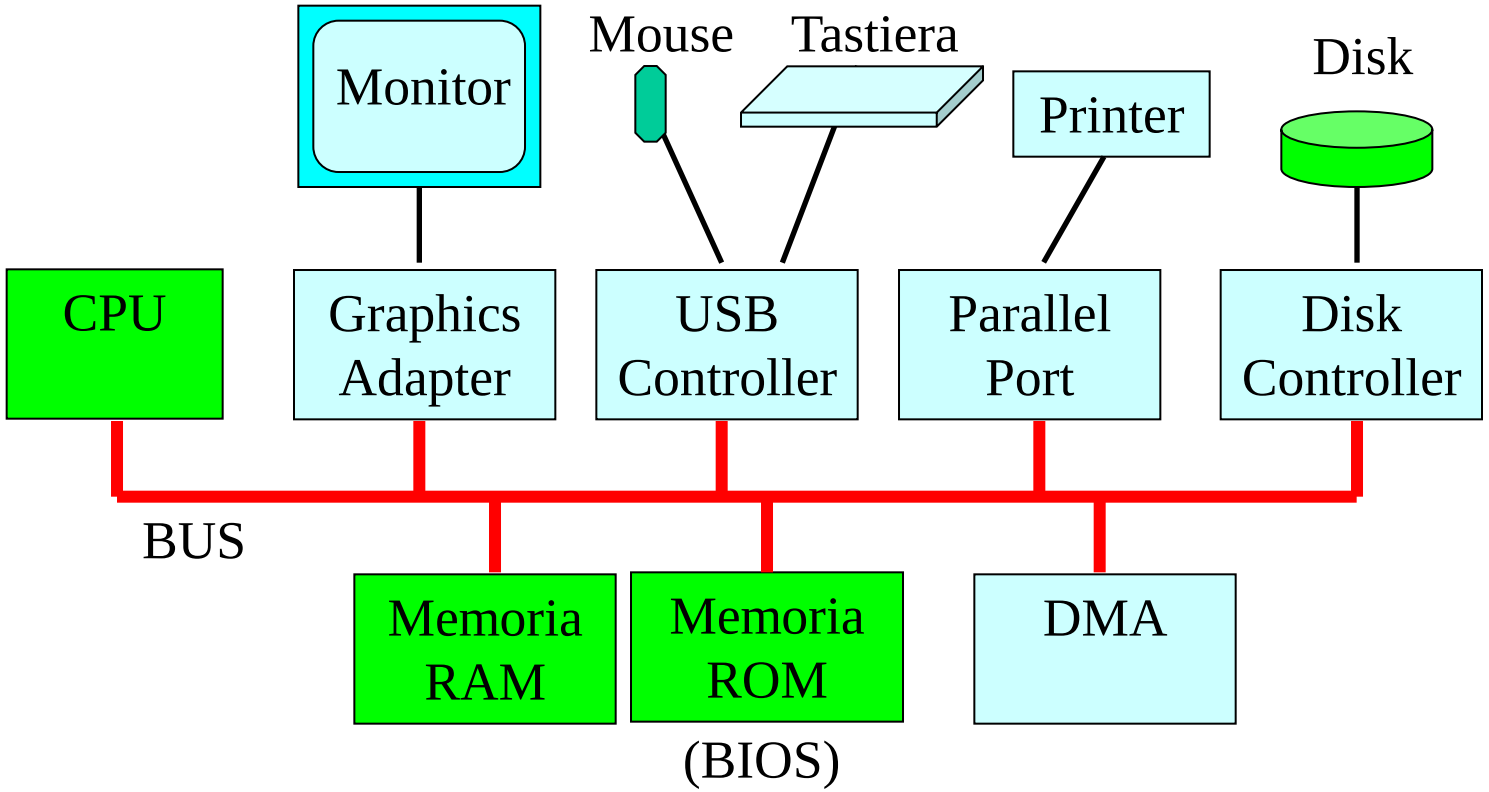
Organizzazione di un Computer



Organizzazione di un Computer – Linguaggi e Strumenti



Hardware di un Computer



CPU esegue istruzioni, effettua calcoli, controlla input/output, coordina spostam. dati.

BUS trasferisce i dati tra la memoria e gli altri dispositivi.

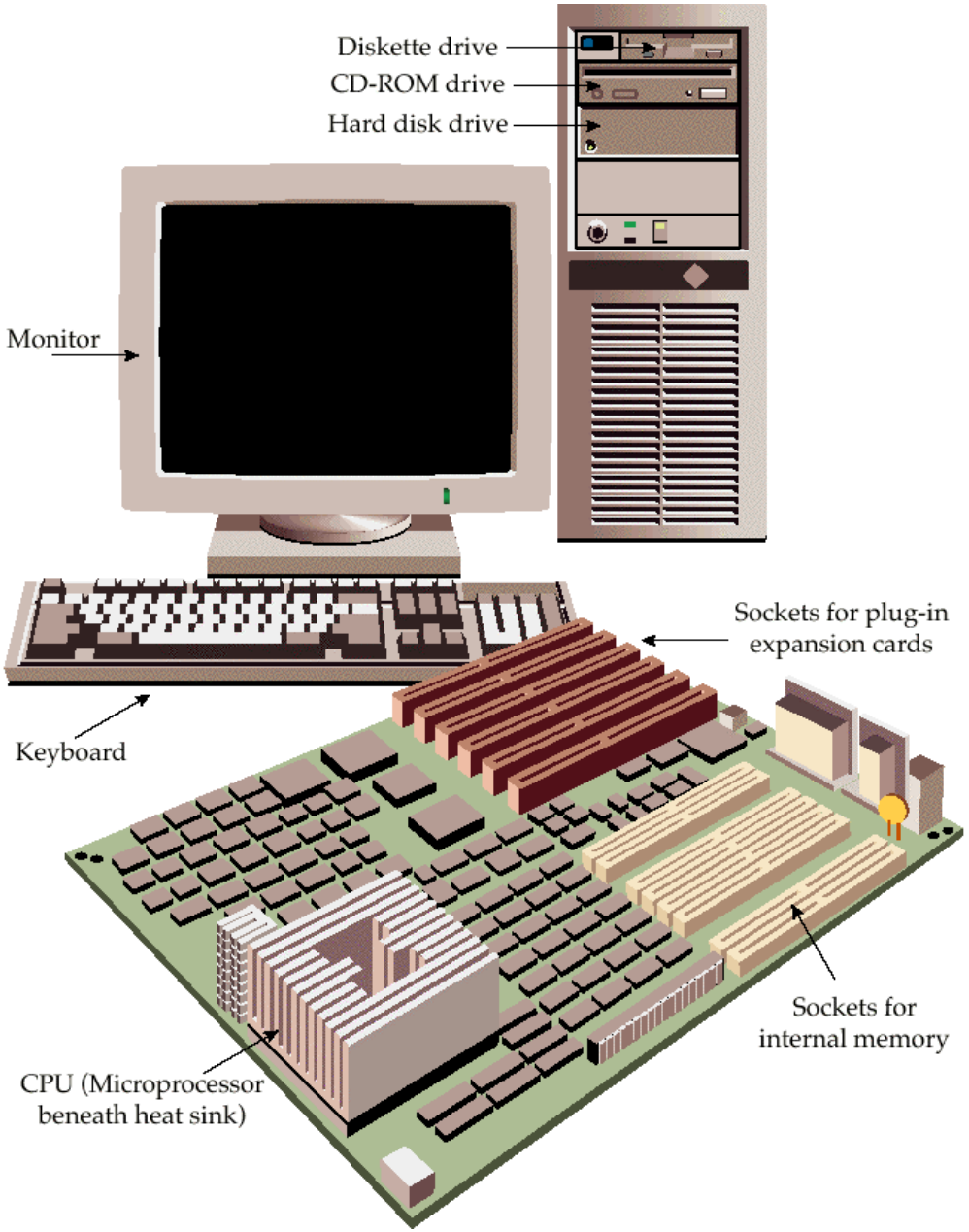
Memoria RAM (volatile) mantiene programmi (seq. di istruzioni) quando devono essere eseguiti, e i dati che i programmi usano, ma solo finché il computer è acceso.

Memoria ROM (permanente) contiene il BIOS, cioè le istruzioni per fornire i servizi base, usate dalla CPU anche nel momento dell'accensione del computer (il BOOT).

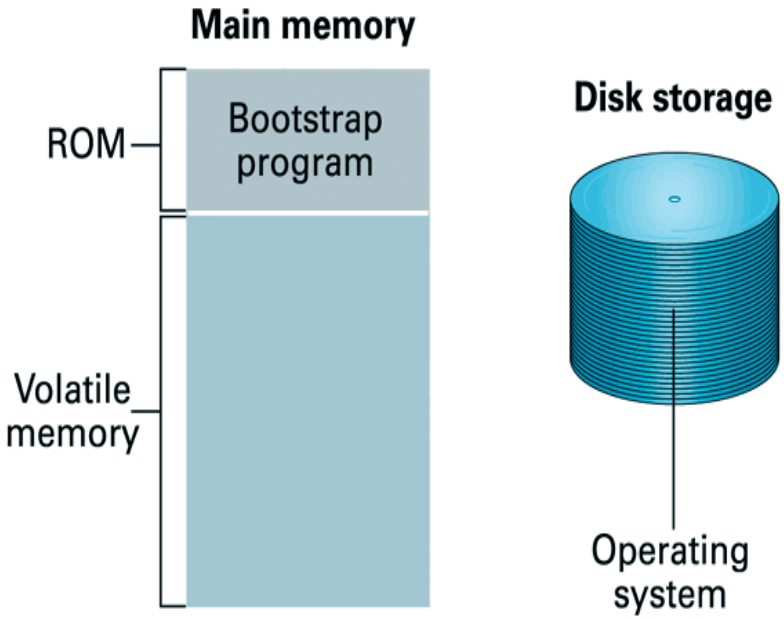
Dischi mantengono enormi quantità di programmi e dati, anche dopo lo spegnimento del computer.

Sono inoltre presenti alcune periferiche di **I/O** (Input/Output).

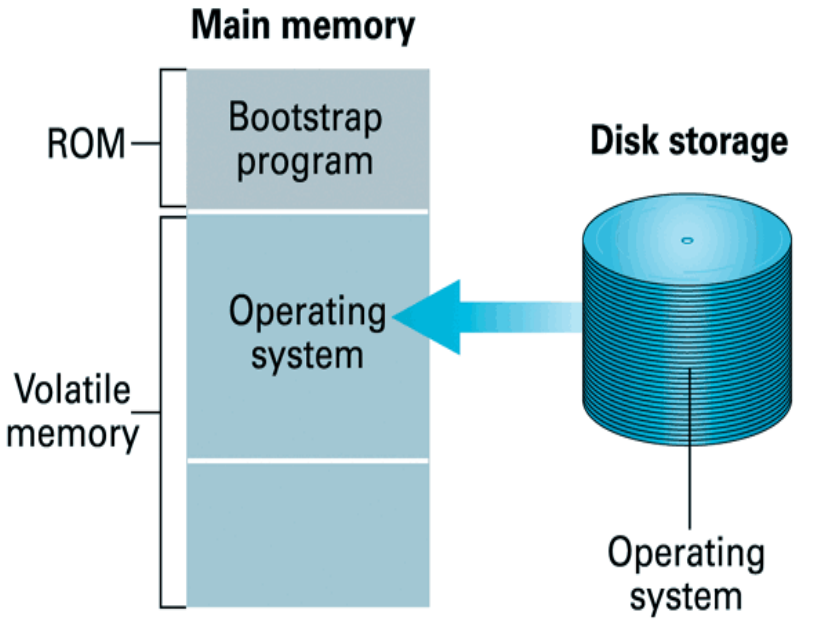
Un tipico Computer



L'avvio - Bootstrap

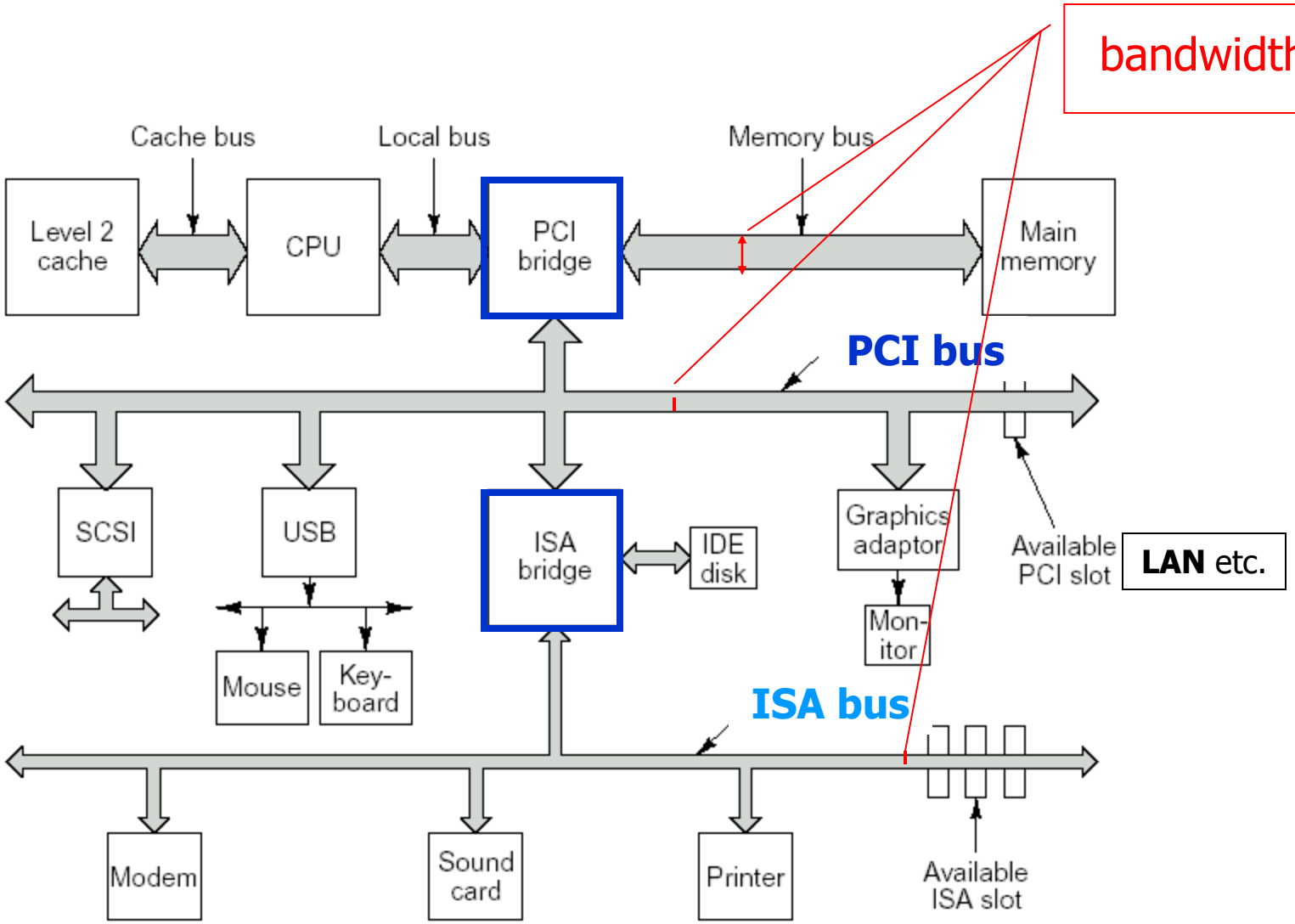


Step 1: Machine starts by executing the bootstrap program already in memory. Operating system is stored in mass storage.



Step 2: Bootstrap program directs the transfer of the operating system into main memory and then transfers control to it.

Diversi tipi di BUS



Codifica Binaria dell'Informazione

Il computer utilizza un **alfabeto binario**:

usiamo dispositivi elettronici digitali in grado di assumere due solo stati: acceso/spento, ON/OFF, 1/0, VERO/FALSO.

Informazione Minima è il **bit** (da **B**inary dig**IT**): quantità d'informazione che si ottiene selezionando una configurazione da un insieme che ne contiene due.

Con un bit rappresento due valori 0, 1.
Con due bit rappresento 4 valori 00, 01, 10, 11.
Con tre bit rappresento 8 valori 000, 001, 010, 011, 100, 101, 110, 111.

In generale con N bit rappresento 2^N valori.

La memoria di un computer è organizzata a blocchi di 8 bit

Byte blocco di 8 bit $\Rightarrow 2^8 = 256$ stati

Notazione Posizionale

Consideriamo i numeri interi assoluti (numeri naturali più lo zero).

Notazione posizionale in base b (generica)**:**

Alfabeto: $\mathbf{c_i \in \{0 \ldots b\text{-}1\}}$ comprende b simboli (cifre)

Un numero è rappresentato come una sequenza di simboli dell'alfabeto.

La sequenza $\mathbf{(c_k\ c_{k-1}\ \ldots\ c_1\ c_0)_b}$

rappresenta, secondo la base b, il numero che vale:

$$\mathbf{N = c_k * b^k + c_{k-1} * b^{k-1} + \ldots + c_0 * b^0 = \Sigma_{(i=0\ldots k)}\ c_i * b^i}$$

Notare che le posizioni (gli indici) iniziano da destra con il valore 0.

es. in base 10: $(937)_{10} = 9*10^2 + 3*10^1 + 7*10^0 = 900 + 30 + 7$

Definizione: Poiché i simboli più a destra nella sequenza sono i coefficenti delle potenze con esponente minore, si dice che **quello più a destra è il simbolo meno significativo**, e **quello più a sinistra è il simbolo più significativo**.

$\mathbf{c_0}$ simbolo meno significativo (Least Significant)

$\mathbf{c_k}$ simbolo più significativo (Most Significant)

Notazione Posizionale – base binaria

Basi più frequentemente utilizzate:

base 2 : $c_i \in \{0, 1\}$

base 8: $c_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$

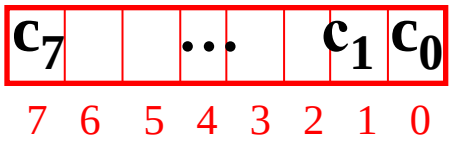
base 10: $c_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

base 16: $c_i \in \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Codifica Binaria

Numeri binari: base **b = 2**

Alfabeto binario: cifre $c_i \in \{0, 1\}$



$(c_k \ldots c_1\ c_0)_2$ rappresenta: $N = \Sigma_{(i=0 \ldots k)}\ c_i * 2^i$

c_k (MSb, Most Significant bit) c_0 (LSb, Least Significant bit)

Conversione Binario->Decimale

$N = (1101)_2$

$N = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$

NB: se non indico la base es: $(..)_?$ intendo riferirmi alla base 10.

Notazione Posizionale – basi importanti

Vediamo un esempio, nel caso di base decimale, che usa i simboli {0, 1, 2,…,8,9}.
Il valore del numero 201 (qui espresso in base 10), è calcolato come:

$2*10^2 + 0*10^1 + 1*10^0 = 2*100+0*10 + 1*1 = 200+0+1=201$

Nel caso di base esadecimale, si usano i 16 simboli {0,1,2,3, …,8,9, A,B,C,D,E,F}, con A che vale 10, B vale 11, C vale 12, D vale 13, E vale 14 e F vale 15.

Il valore del numero $\mathbf{(C9)_{16}}$, espresso in base 16, è calcolato come:

$C*16^1 + 9*16^0 = 12*16+9*1 = 192+9=201$

- **E' importante** sottolineare che con **2 cifre esadecimali** (cioè con una coppia di simboli che possono assumere ciascuno 16 diversi valori) io **posso rappresentare 256 valori, esattamente quanti sono i valori esprimibili con un singolo Byte.**

- Per tale motivo, **spesso si utilizza la base esadecimale per indicare il valore di un Byte, poiché servono esattamente due cifre esadecimali per rappresentare tutti e soli i possibili valori di un byte,** dal valore $0=(00)_{16}$ al valore $255=(FF)_{16}$

Notazione Posizionale – casi importanti – base esadecimale

Codifica Esadecimale

Numeri esadecimali: base **b = 16**

Alfabeto: cifre $\mathbf{c_i \in \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}}$

A vale 10, B vale 11, C vale 12, D vale 13, E vale 14, F vale 15

$\mathbf{(c_k \ldots c_1\ c_0)_{16}}$ rappresenta: $\mathbf{N = \Sigma_{(i=0\ldots\ k)}\ c_i * 16^i}$

Conversione Esadecimale->Decimale

$N = (F2)_{16}$

$N = F \times 16^1 + 2 \times 16^0 = 15*16 + 2*1 = 242_{10}$

NB: se non indico la base es: $(..)_2$ intendo riferirmi alla base 10.

Anticipazione – Notazione Posizionale nel Linguaggio C

Il linguaggio ANSI C, nella sua smisurata saggezza, mette a disposizione una notazione semplice, senza tanti pedici o parentesi, per rappresentare dei numeri interi.

In particolare, nella notazione posizionale, il linguaggio C indica la base mediante la quale è rappresentato un numero, premettendo al numero un prefisso.

Ai numeri espressi in base 2 viene messo il prefisso 0b
Ai numeri espressi in base 8 viene messo il prefisso 0
Ai numeri espressi in base 16 viene messo il prefisso 0x
Ai numeri espressi in base 10 non viene messo alcun prefisso.

In base 2	**0b**101	=	5
In base 2	**0b**10000	=	16
In base 8	**0**20	=	16
In base 8	**0**77	=	63
In base 16	**0x**FF	=	255
In base 16	**0x**10	=	16
In base 16	**0x**32	=	50
In base 10	44		

Rappresentazione di numeri interi positivi in base esadecimale

Rappresentazione di Interi Positivi (>=0)

Se usiamo **un Byte di memoria**, possiamo memorizzare $\mathbf{2^8=256}$ valori da 0 a 255 e li possiamo rappresentare usando 2 cifre esadecimali.

0x00=0 0x20=32 0x33=51
0x80=128 0xC0=192 0xFF=255

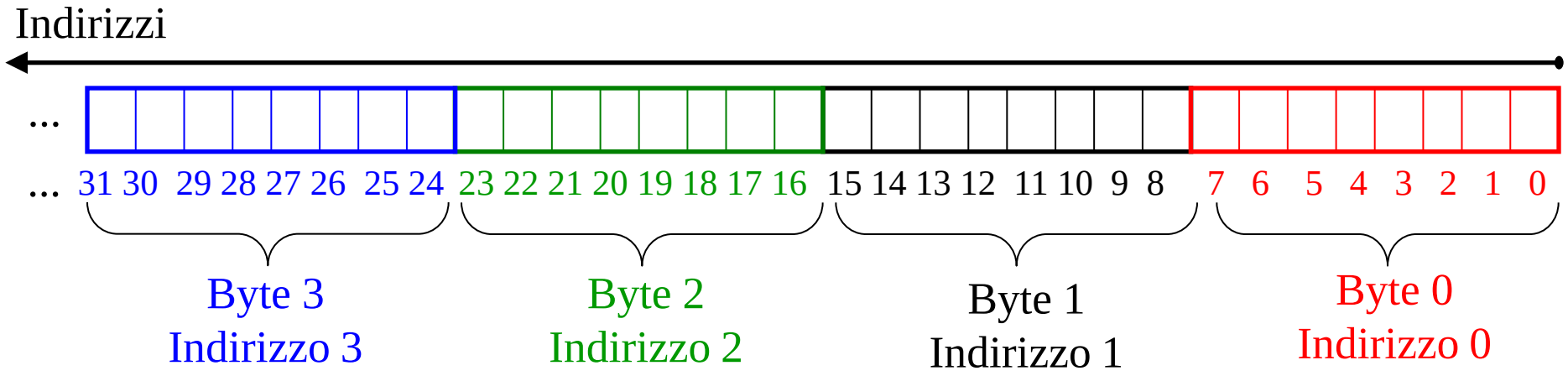
Se usiamo **due Byte di memoria**, possiamo memorizzare $\mathbf{2^{16}=65536}$ valori da 0 a 65535e li possiamo rappresentare usando 4 cifre esadecimali.

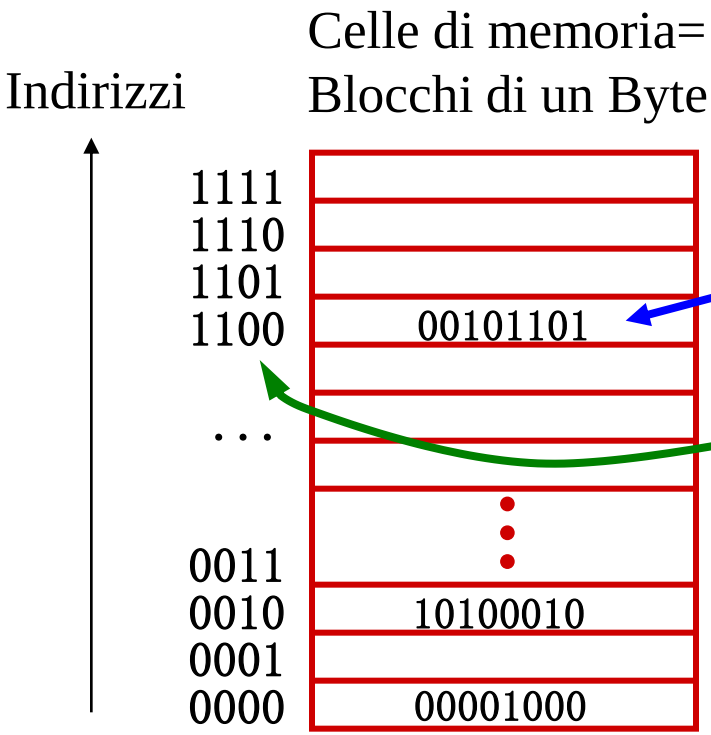
0x0000=0 0x0100=256 0x0400=4*256=1024
0x8000=128*256=32768 0xFFFF=65535

Se usiamo **quattro Byte di memoria**, possiamo memorizzare $\mathbf{2^{32}=4\ 294\ 967\ 296}$ valori da 0 a 4miliardi e un po', e li possiamo rappresentare usando 8 cifre esadecimali.

0xFFFFFFFF=4294967295

Organizzazione della Memoria Principale





La memoria è organizzata come una sequenza di celle ciascuna costituita da un singolo Byte (8 bit).
Una cella di memoria è caratterizzata da:
- Un valore
  Il contenuto della cella stessa, visto come numero binario.
- Un indirizzo
  L'indirizzo di inizio della cella.

Byte 2, valore 162, indirizzo 0010

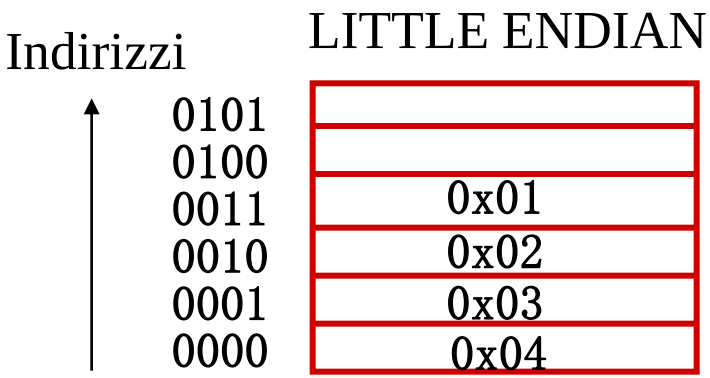
Byte 0, valore 8, indirizzo 0000

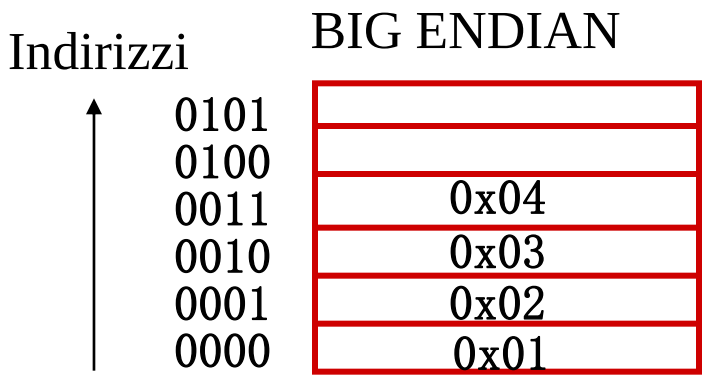
Convenzione: Endianess – Ordine dei Byte degli Interi

- Un numero intero formato da 2, 4 o 8 Byte può essere collocato in quei byte in diversi modi. Essenzialmente i processori si suddividono in due classi a seconda di questa caratteristica, detta Endianess. Si parla di:

- **Little Endian**: i Byte più significativi del numero stanno nei Byte di indirizzo maggiore. (LSB, Least Significant Byte first).
- **Big Endian**: i Byte più significativi del numero stanno nei Byte di indirizzo Minore. (MSB, Most Significant Byte first).

- Ad es. consideriamo il numero formato da 4 Byte e rappresentato in base esadecimale da **0x01020304**.
- Il Byte più significativo è quello che contiene il valore 0x01.
- Il Byte meno significativo è quello che contiene il valore 0x04.
- Ecco come i Byte sono collocati nelle due classi.

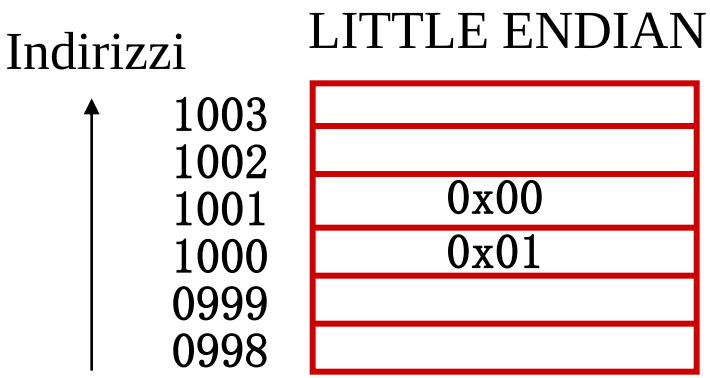




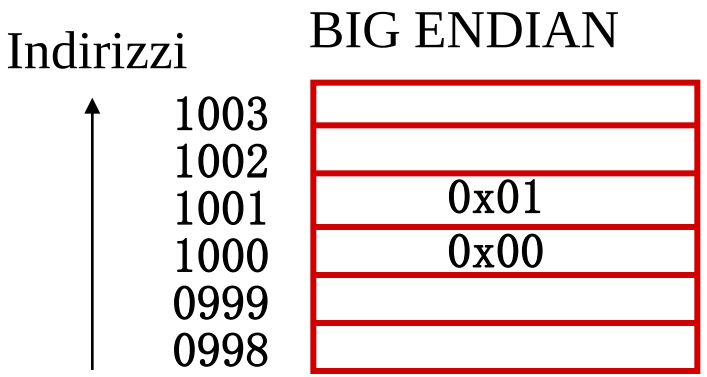
Endianess – Ordine dei Byte degli Interi – un esempio semplice

- Consideriamo un intero senza segno a 16 bit uint16_t
- supponiamo che sia collocato in memoria all'indirizzo 1000
- supponiamo che abbia valore 1

Vediamo come sara' collocato in memoria a seconda del diverso tipo di CPU da cui viene gestito:



Quale sarebbe il valore dell'intero cosi' memorizzato se invece il processore fosse di tipo Big Endian?



Quale sarebbe il valore dell'intero cosi' memorizzato se invece il processore fosse di tipo Little Endian?

Definizioni importanti dipendenti dalle caratteristiche dei BUS (1)

- **Word** o **Parola**. E' un blocco di byte in memoria aventi la stessa dimensione dell'**ampiezza** in byte del **Bus dei Dati**. Indica quanti byte possono essere trasferiti, al massimo, con una unica operazione del bus tra CPU e memoria.

Implicazioni 1: Molto spesso i registri general purpose dei processori hanno la stessa dimensione della word.

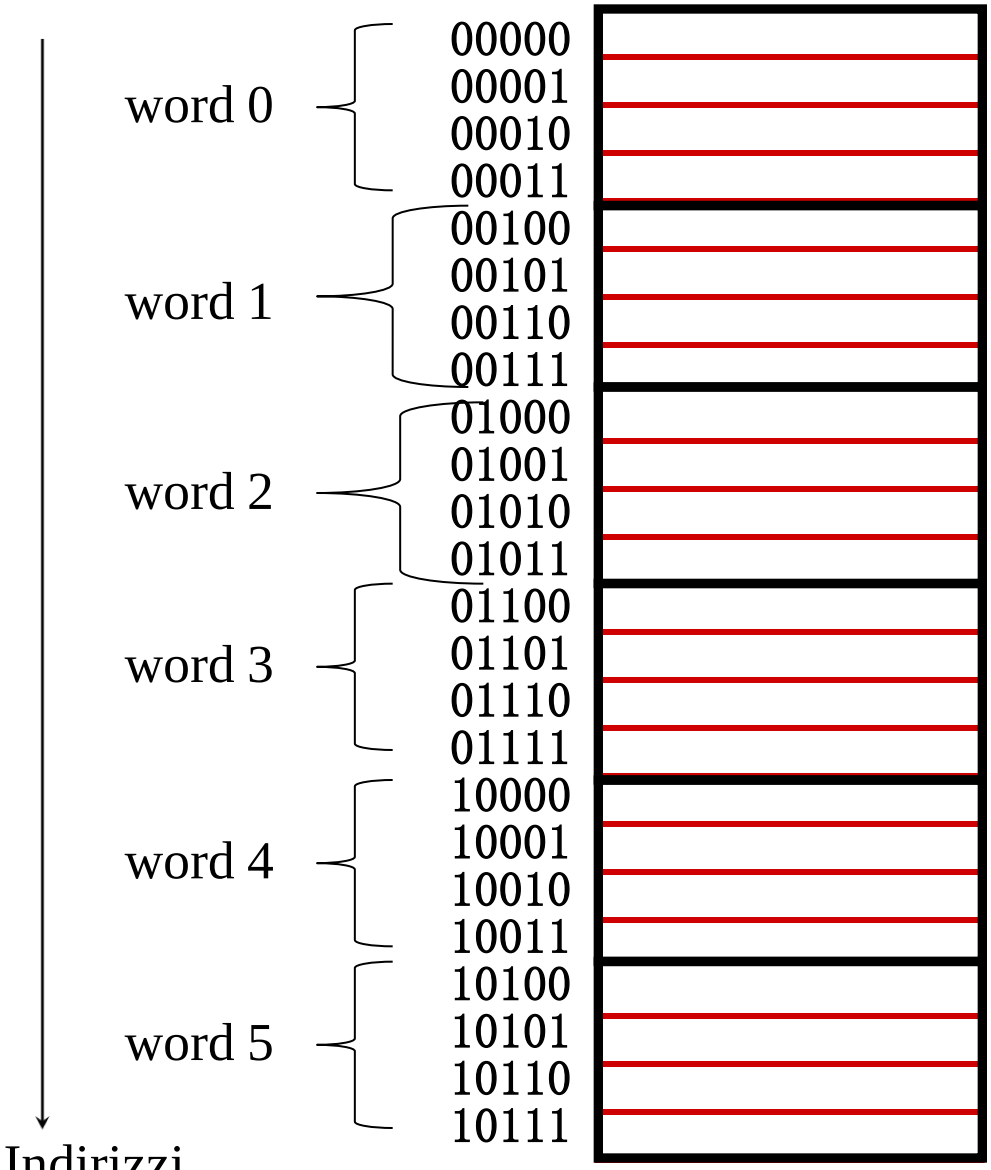
Implicazioni 2: Per semplificare la costruzione di bus e memoria, una word in memoria puo' cominciare solo ad un indirizzo multiplo della word. Possiamo cioe' vedere la memoria come se fosse organizzata a word e come se potessimo accedere solo ad una word per volta.

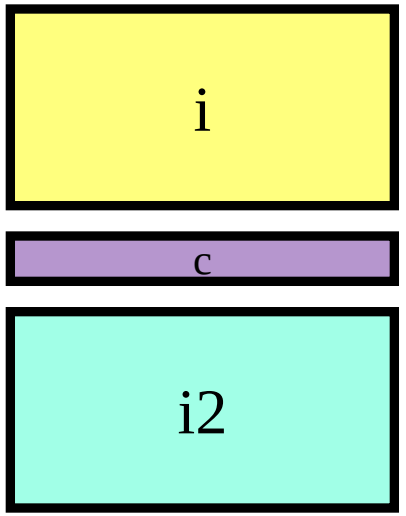
Ad esempio, se ho una word di 4 bytes ed ho un blocco dati di 3 bytes che cominciano all'indirizzo 42 e termina all'indirizzo 44 (compresi gli estremi), allora per trasferire quel blocco dati dalla memoria alla CPU il microprogramma che guida la CPU dovra' effettuare due operazioni di lettura col bus, la prima leggendo i bytes di indirizzo 42 e 43, la seconda leggendo il solo byte di indirizzo 44.

Conseguenze della dimensione della word: Dimensione strutture

```
        /*   discussione del seguente esempio in linguaggio ANSI C  */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h> /* per la definizione dei tipi interi */

int main(void) {      int intero;
        typedef   struct struttura {
                                uint32_t        i    ;
                                uint8_t         c   ;
                                uint32_t        i2  ;
                                } STRUTTURA;

        STRUTTURA s;

        printf( "dimensione int %ld \n\n" ,  sizeof(int) );

        printf( "dimensione uint32_t %ld \n" ,  sizeof(uint32_t) );
        printf( "dimensione uint8_t %ld \n" ,   sizeof(uint8_t) );
        printf( "dimensione struttura %ld \n" , sizeof(STRUTTURA) );

        return(0);
}
```

Strutturazione in word della memoria.
Il bus dati può accedere al massimo ad una word per volta

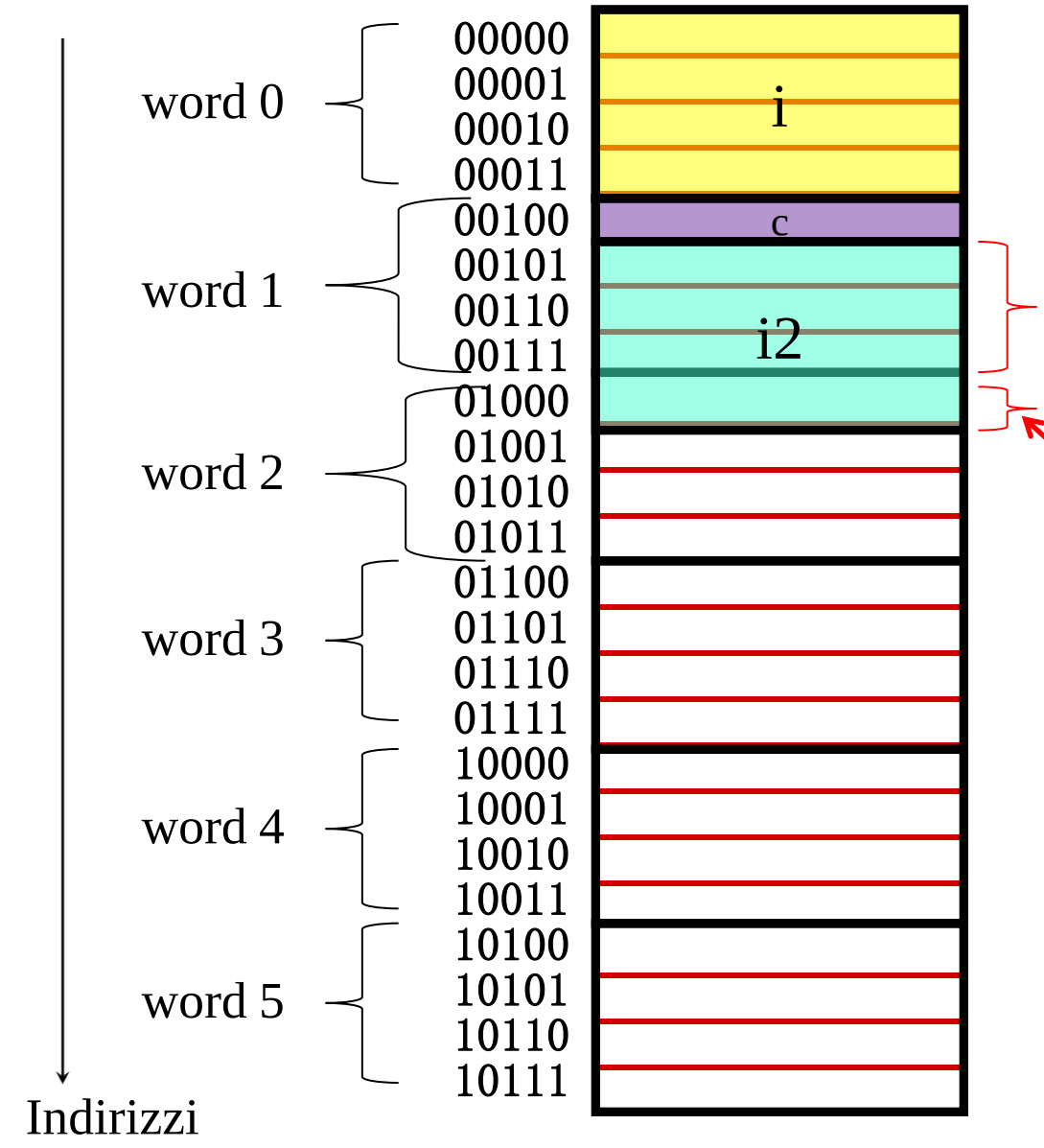




I campi della STRUTTURA

DOVE LI COLLOCA IL COMPILATORE?

Distribuzione accorpata in memoria dei campi di strutture. I singoli campi superano i confini delle word



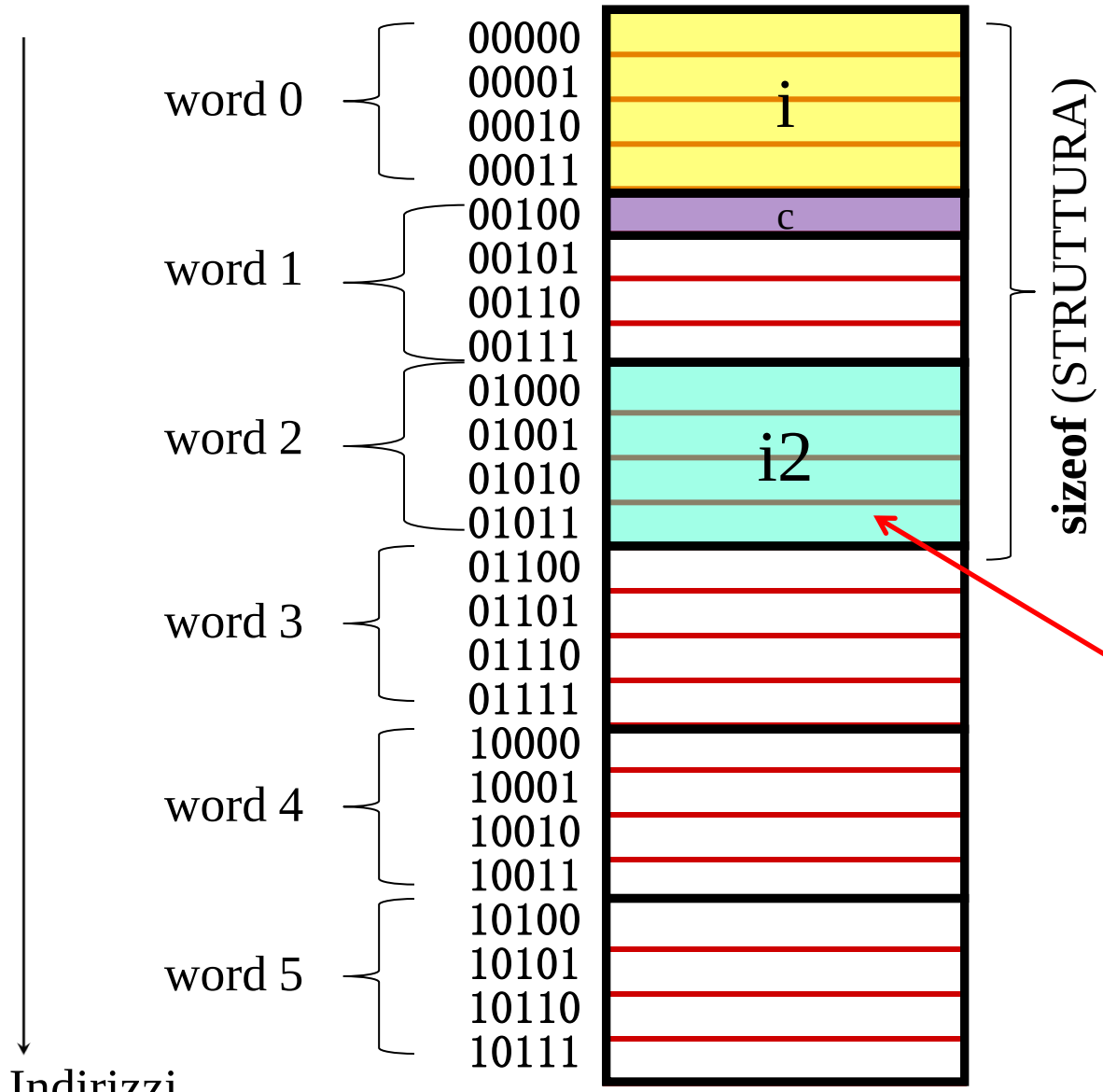
Poiché il bus dati può accedere al massimo ad una word per volta, se un campo supera i confini di una word allora occorrono più letture per leggere il campo

i primi 3 byte di i2 stanno in word 1

il 4° byte di i2 sta in word 2

per leggere tutto il campo i2 occorrono due accessi in lettura mediante il bus dati

Distribuzione efficiente in memoria dei campi di strutture. I singoli campi non devono superare i confini delle word



L'efficienza consiste nel limitare il numero di letture e scritture delle word per accedere ai campi

Il campo i2 sta in una unica word, la word 2

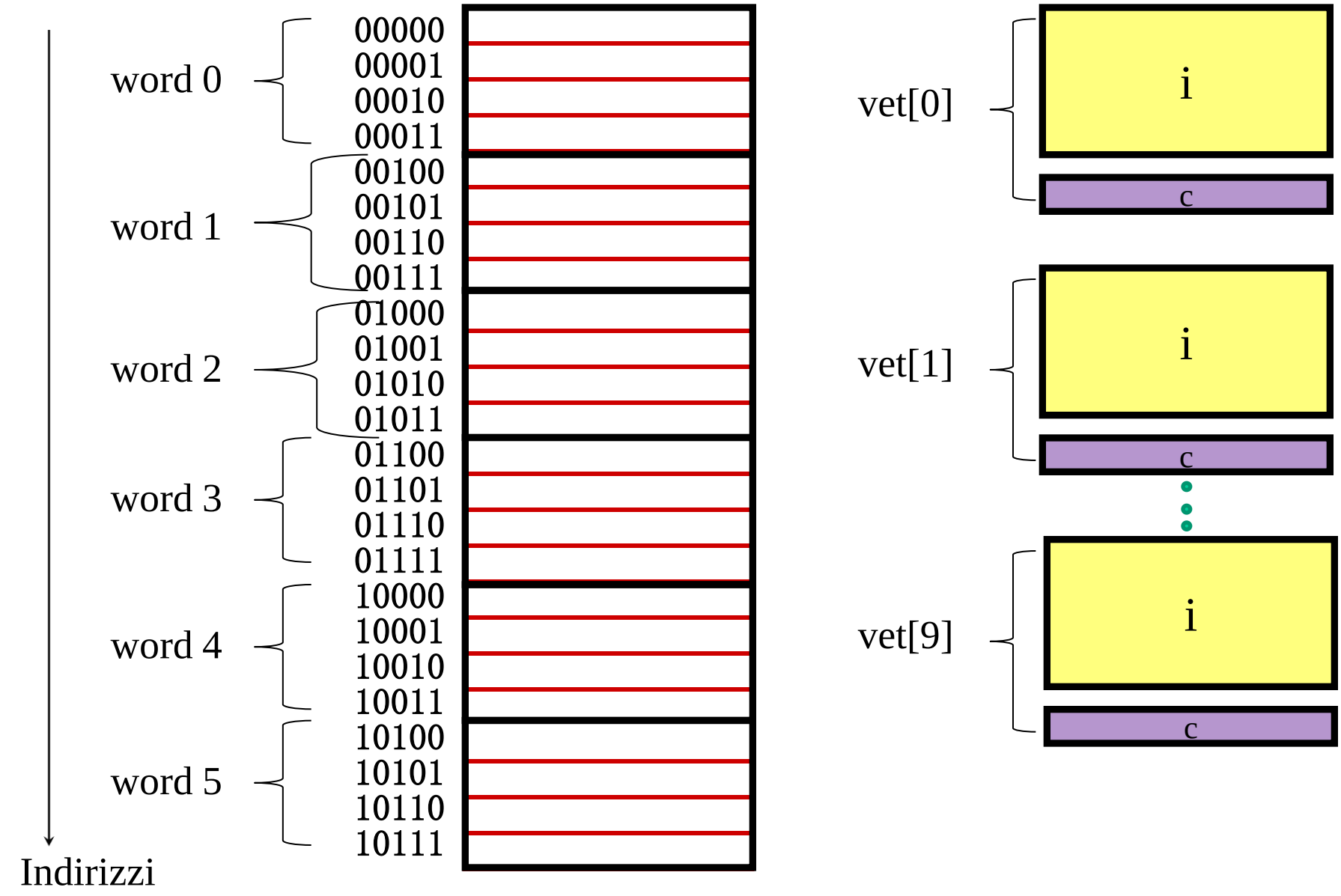
Per leggere il campo i2 occorre la lettura della sola word 2

Conseguenze della dimensione della word: Dimensione vettori

```
        /*   discussione del seguente esempio   in linguaggio ANSI C */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h> /* per la definizione dei tipi interi */

int main(void) {      int intero;
        typedef   struct struttura {
                                uint32_t i;
                                uint8_t  c;
                                } STRUTTURA;

        STRUTTURA          s;
        STRUTTURA          vet [10];

        printf( "dimensione struttura %ld \n" , sizeof( s ) );
        printf( "dimensione struttura %ld \n" , sizeof( vet ) );

        return(0);
}
```

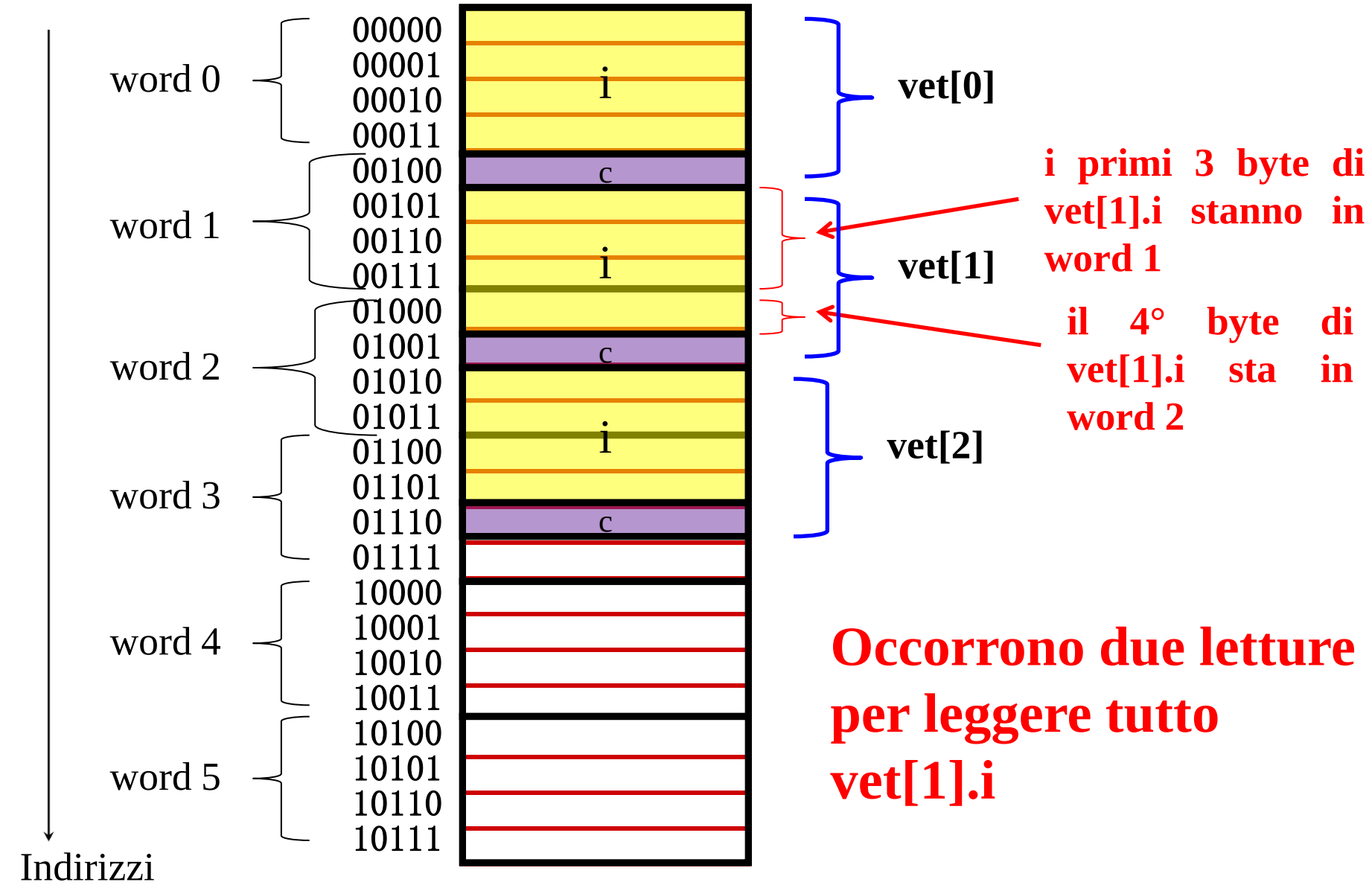
Altro esempio. Struttura più piccola

Gli elementi del vettore di tipo STRUTTURA. Dove li colloca il compilatore?



Distribuzione accorpata degli elementi del vettore.

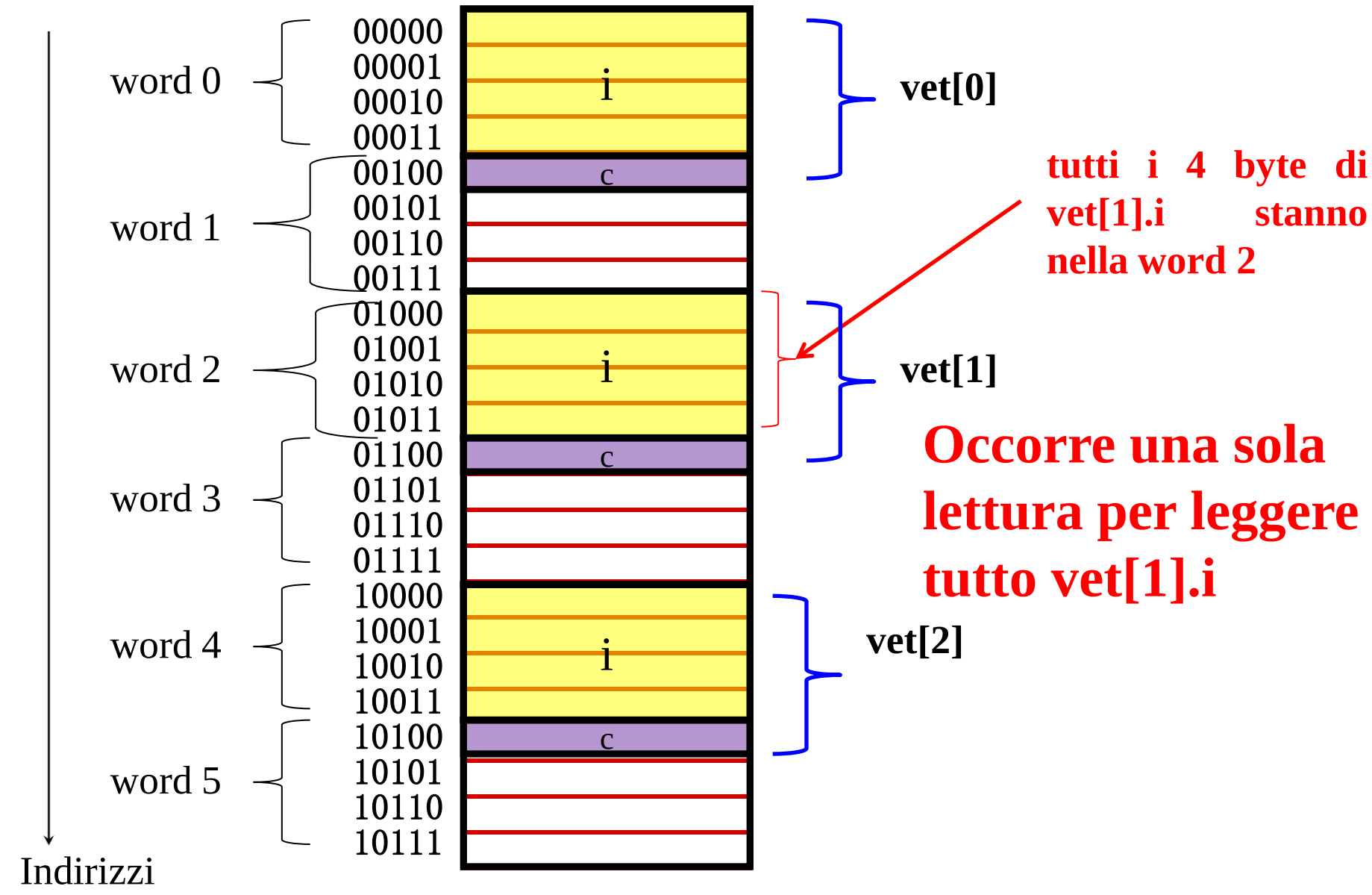
vet[1] segue immediatamente vet[0]



Occorrono due letture per leggere tutto vet[1].i

Distribuzione efficiente degli elementi del vettore.

vet[1] inizia nella prima word vuota che segue vet[0]



Conseguenze della dimensione della word: strutture packed

```
        /*   discussione del seguente esempio   in linguaggio ANSI C */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h> /* per la definizione dei tipi interi */

int main(void) {      int intero;
        typedef   struct struttura {
                                uint32_t i;
                                uint8_t  c;
                                }  __attribute__(( packed ))     STRUTTURA;

        STRUTTURA          s;
        STRUTTURA          vet [10];

        printf( "dimensione struttura %ld \n" , sizeof( s ) );
        printf( "dimensione struttura %ld \n" , sizeof( vet ) );

        return(0);
}
```

Definizioni importanti dipendenti dalle caratteristiche dei BUS (2)

- **Capacita' di indirizzamento in memoria**. "**a 32 bit**" "**a 64 bit**" E' **l'ampiezza** in bit del **Bus degli Indirizzi** quindi e' la dimensione in bit degli indirizzi utilizzati dall'hardware per accedere ai byte in memoria.

Per quanto visto prima, se n e' la dimensione in bit degli indirizzi allora 2^n e' il numero di byte indirizzabili in memoria.

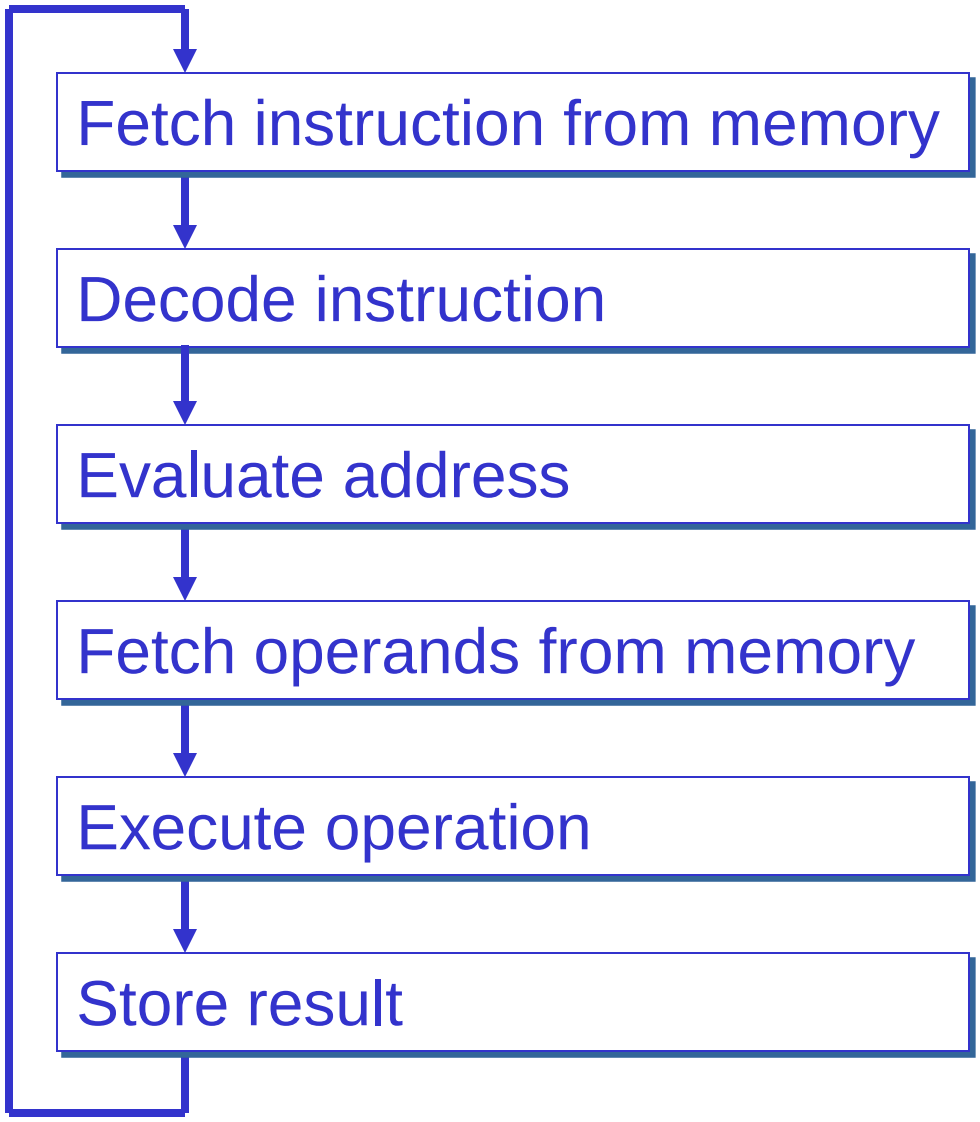
Con 20 bit (come nell'8088) posso indirizzare 1.048.576 bytes di memoria cioe' 1 MB.

Con 32 bit (i386) posso indirizzare 4.294.967.296 bytes di memoria cioe' 4 GB.

Con 64 bit (i7) posso indirizzare 18.446.744.073.709.551.616 bytes di memoria.

Implicazioni: all'aumentare della dimensione degli indirizzi aumenta la quantita' di memoria indirizzabile ma aumenta anche la dimensione delle istruzioni che specificano degli indirizzi. Per questo motivo, uno stesso programma compilato per un sistema a 64 bit risultera' piu' grande dello stesso programma compilato per un sistema a 32 bit.

Instruction Processing - Ciclo di Neumann



Un esempio ipotetico di organizzazione interna della CPU

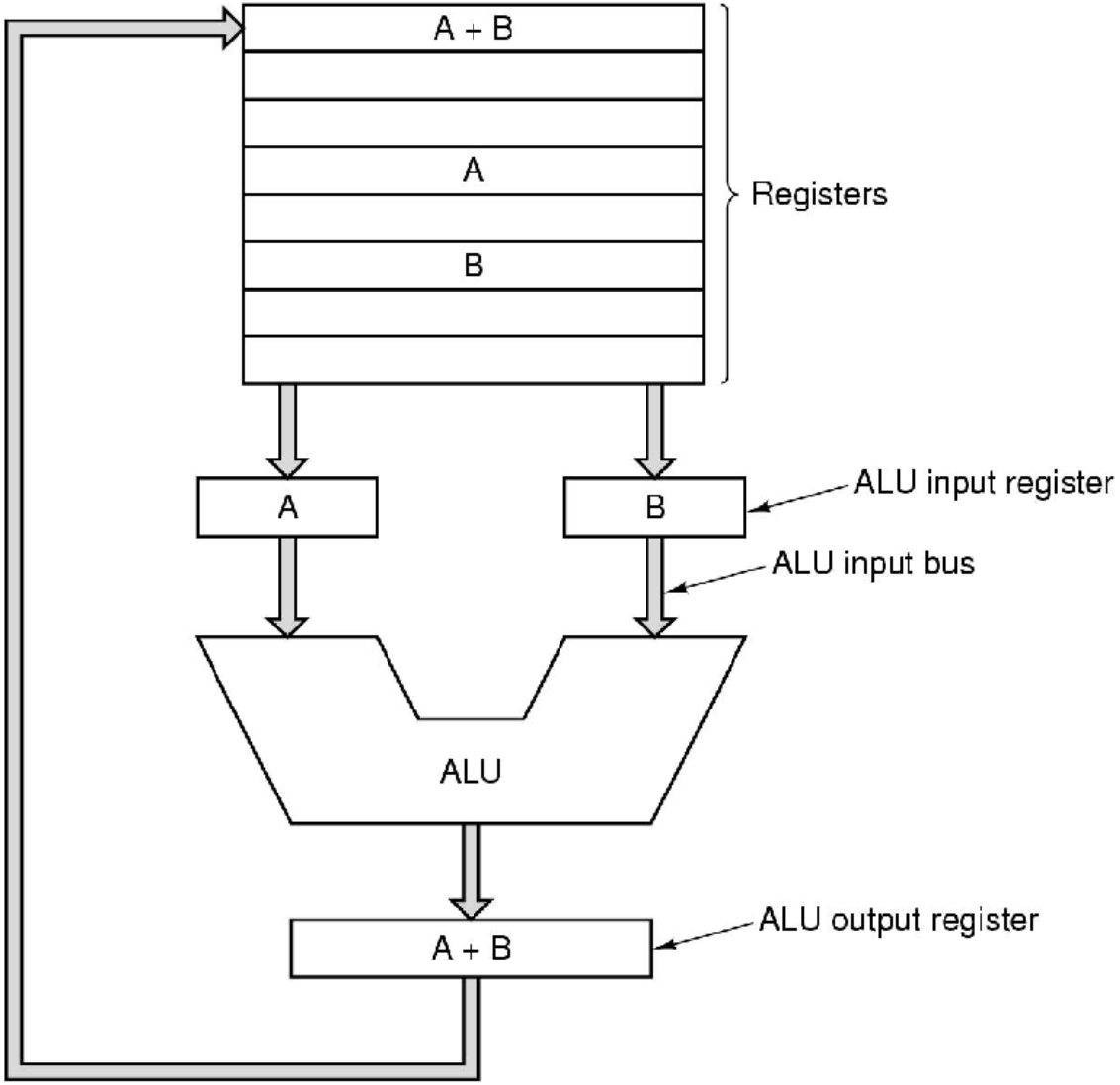
Data Path: organizzazione interna di una CPU (registri, ALU, bus interno)

Registro: memorie veloci per dati temporanei

Registri A e B, input dell'ALU, e A+B output dell'ALU

Istruzioni registro-registro e registro-memoria

Ciclo del data path



Relazione tra “Ciclo di Neumann” e “Ciclo di data path”

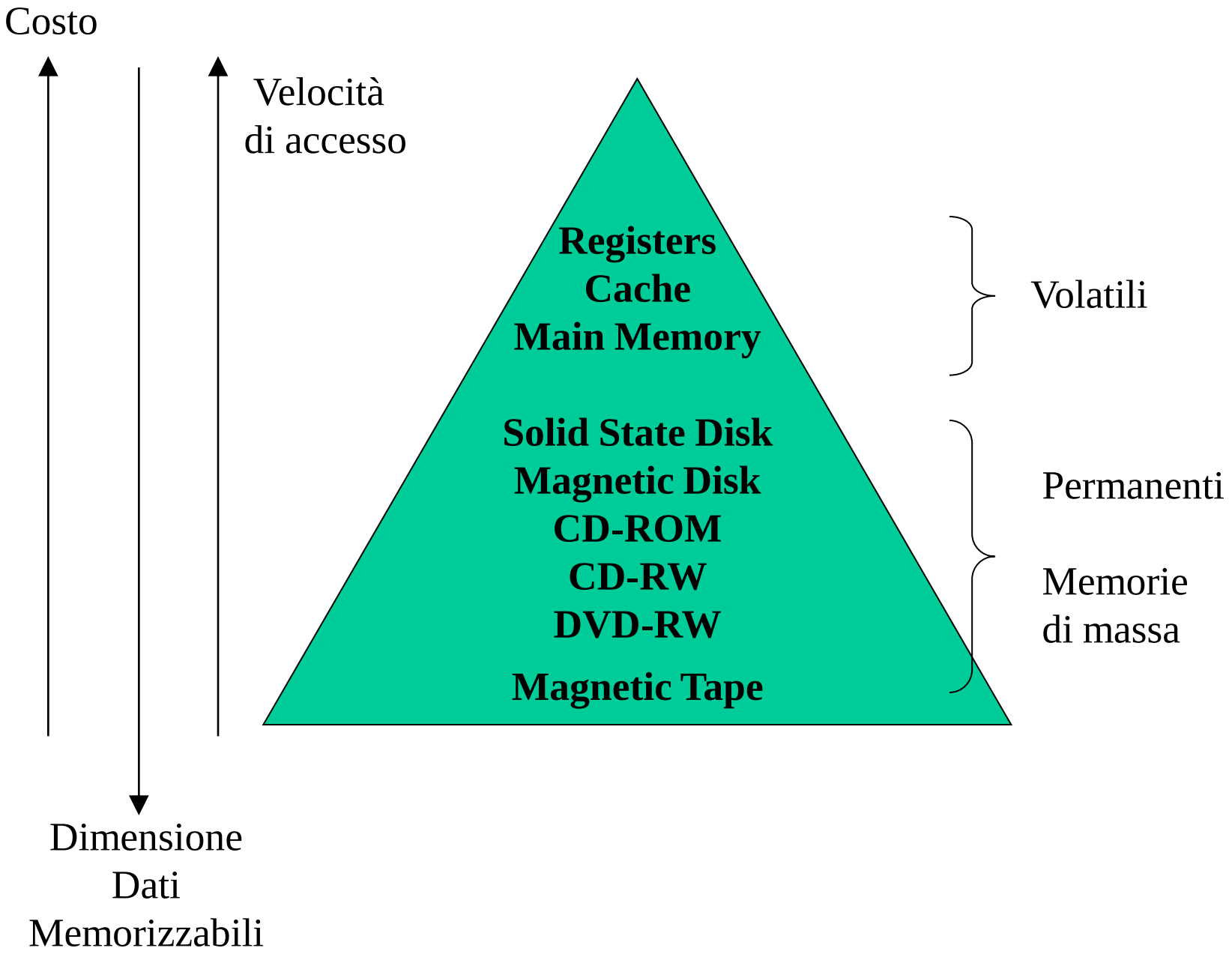
- come visto nella slide precedente, il processo di un’istruzione macchina (detto ciclo di Neumann) è composto da diverse fasi,
- alcune di queste fasi richiedono di accedere alla memoria (in lettura (fetch dell’istruzione, fetch degli operandi) o in scrittura (store del risultato)
- altre fasi possono richiedere di svolgere dei calcoli aritmetici o valutazioni logiche

- Ciascuna di queste fasi viene eseguita utilizzando uno o più cicli di data path

- Quindi c’e’ qualcosa che si occupa di far corrispondere
  - ad una istruzione macchina
  - la sequenza di cicli di data path necessari alla completa esecuzione dell’istruzione.

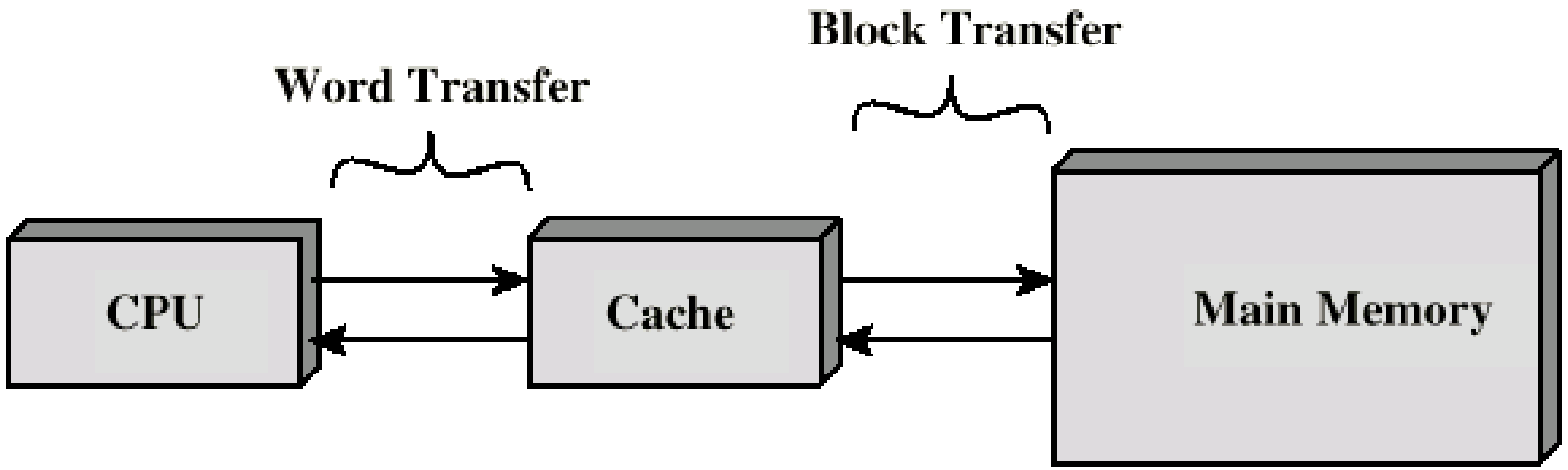
- Nelle CPU meno recenti, questo qualcosa era realizzato esclusivamente mediante circuiteria elettronica (hardware).
- Attualmente, invece, **esiste del software collocato all’interno della CPU stessa (firmware di microprogrammazione) che si occupa di interpretare le istruzioni macchina e di far eseguire la corrispondente sequenza di cicli di data path**

Gerarchie di Memorie

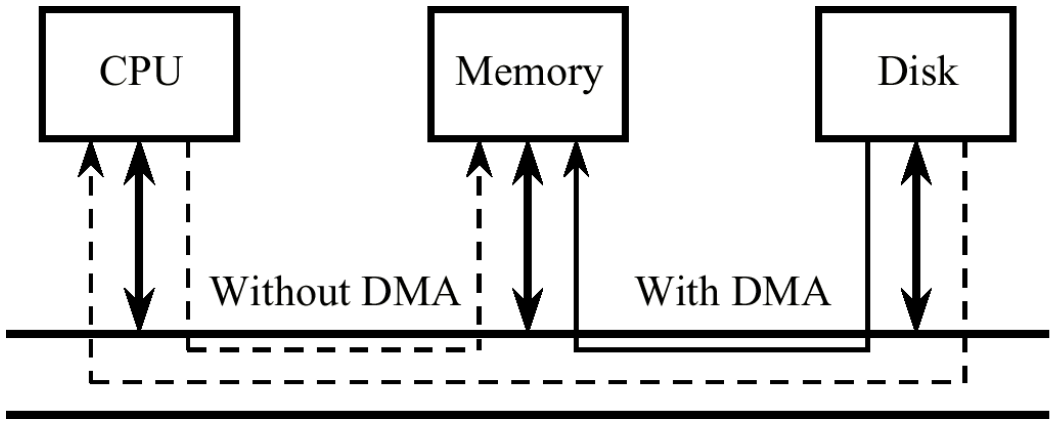


Il concetto di Cache della memoria

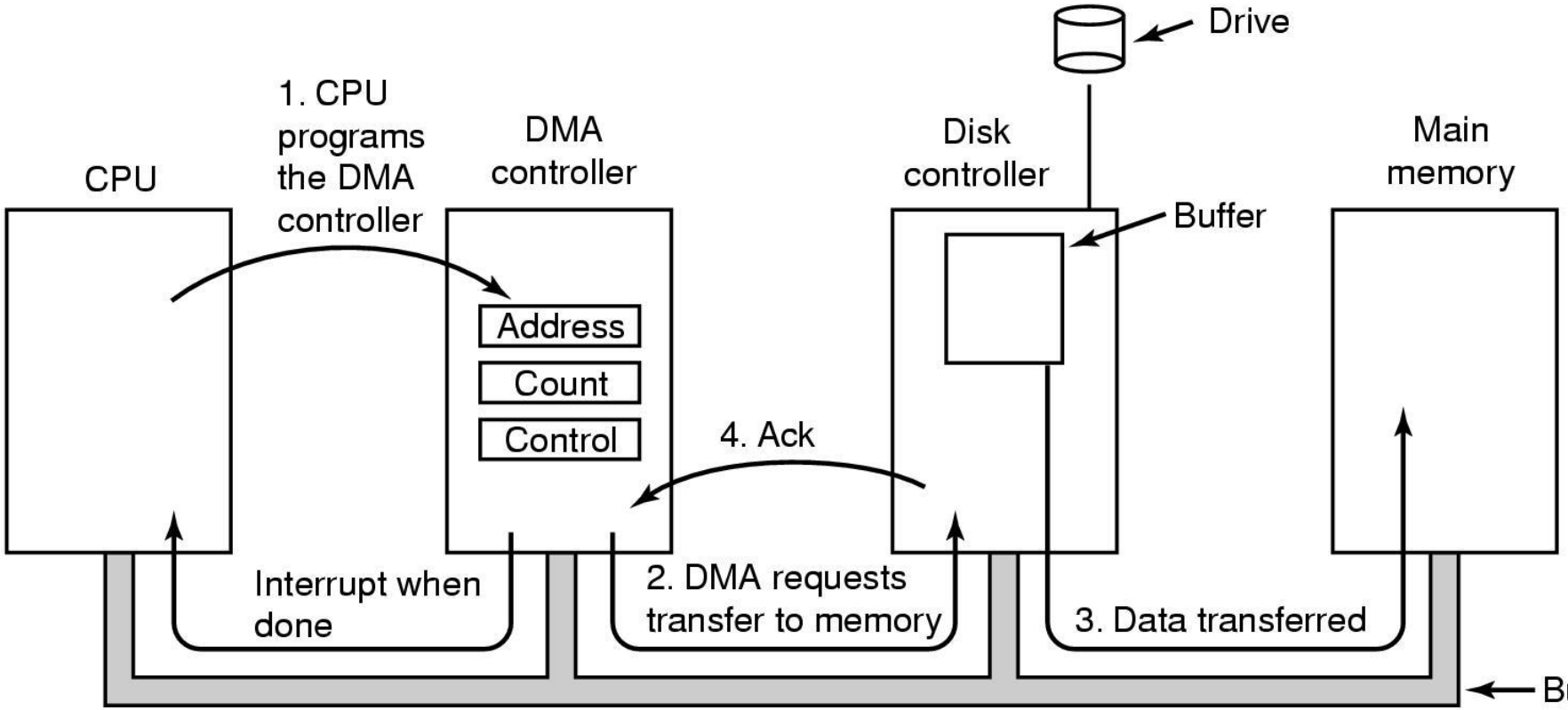
- La memoria veloce è poca, perché costa troppo.
- Viene inserita una piccola quantità di memoria veloce tra la CPU e la memoria principale.
- Può essere collocata anche sul disk controller, o anche tra i diversi componenti della CPU.



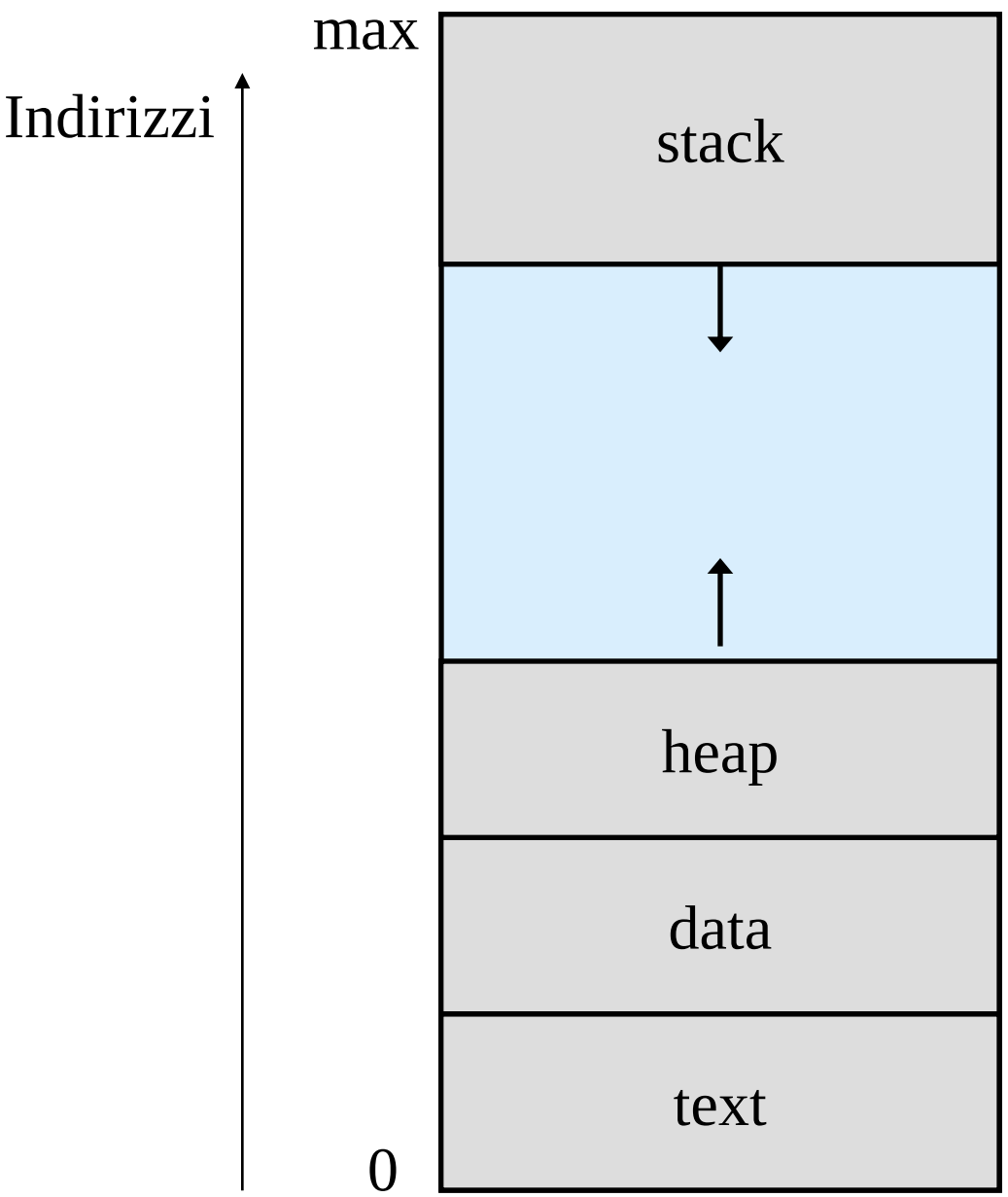
Accesso Diretto alla memoria – DMA - Direct Memory Access



Il trasferimento dei dati, dal disco alla memoria, effettuato dal DMA non coinvolge la CPU tranne che:
- Quando la CPU ordina al DMA lo spostamento del blocco dati.
- Quando il DMA avvisa la CPU dell'avvenuto trasferimento.



Organizzazione della Memoria usata dalla parte utente dei Processi



Contiene i Record di attivazione delle chiamate a funzione

Contiene le zone di Memoria allocata dinamicamente

Contiene le Variabili globali

Contiene il Codice eseguibile (istruzioni macchina)

Supporto ai Processi da parte del livello ISA

I processori moderni hanno caratteristiche comuni che permettono di fornire ai livelli superiori una interfaccia di programmazione denominata **livello ISA** (Instruction Level Architecture) variabile da processore a processore ma avente alcune analogie:

Modalità di protezione (modo kernel, modo utente) e istruzioni per switch tra i modi.

ALU, unità aritmetico-logica e istruzioni per comandarla.

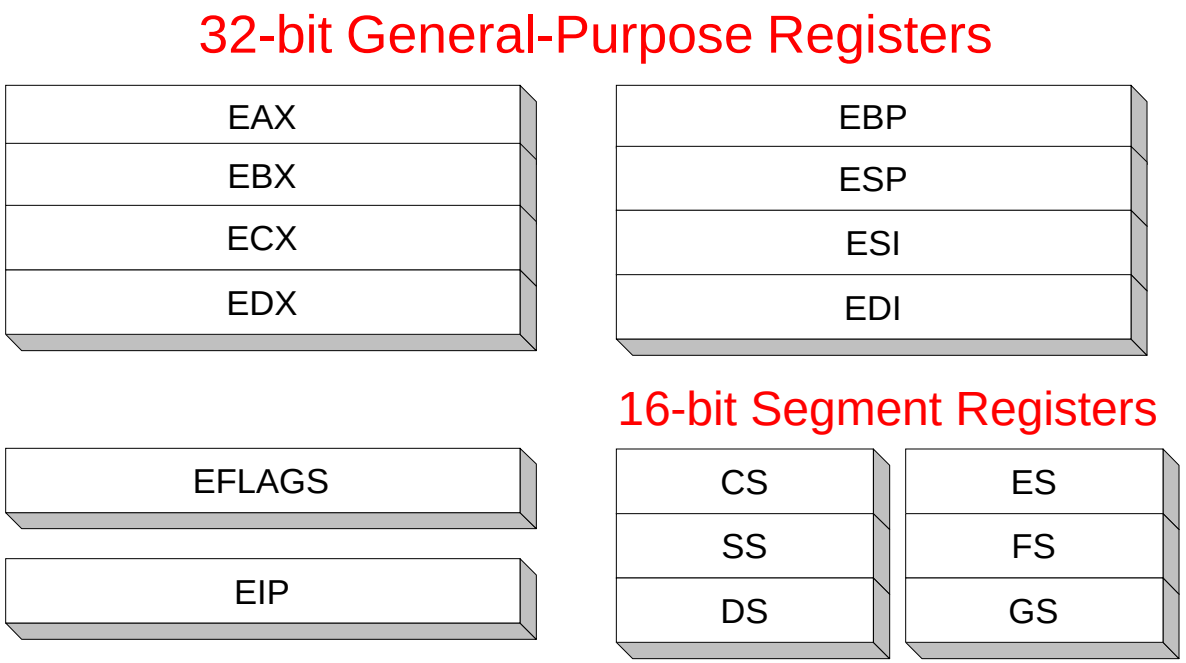
Registri general pourpose (di uso generale) e istruzioni per scambio dati.

Registri Specializzati (nome differente a seconda del processore, qui ad es. per IA-32):

- Extended Instruction Pointer (EIP) serve a formare l'indirizzo in memoria (detto Program Counter (PC)) della prossima istruzione da eseguire.
- Program Status (PS) detto anche Registro di Stato (Status Register).
- Stack Segment Register (SS). Punta all'inizio dello stack.
- Code Segment Register (CS). Punta all'inizio della sezione text, il codice eseguibile.
- Data Segment Register (DS). Punta all'inizio della sezione data, con le var globali.

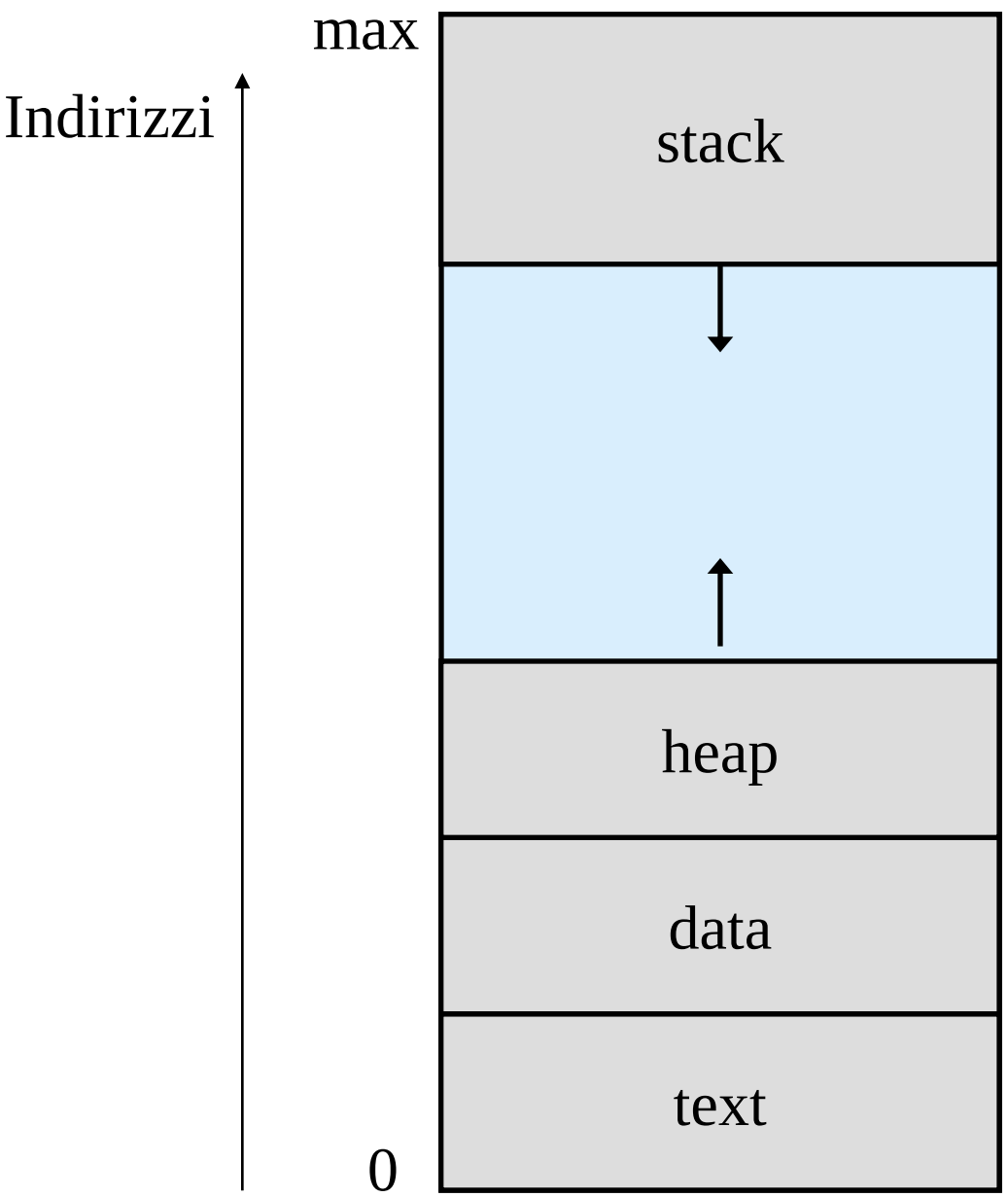
Registri disponibili ai Programmi nei processori di tipo IA-32 (i386, Penthium, ..)

- Registers are high speed memory inside the CPU
  - Eight 32-bit general-purpose registers
  - Six 16-bit segment registers
  - Processor Status Flags (EFLAGS) and Instruction Pointer (EIP)



  - Inoltre, Registro di Stato (Status Register), non disponibile ai programmi, definisce modalità di funzionamento, utente o kernel

Organizzazione della Memoria usata dalla parte utente dei Processi



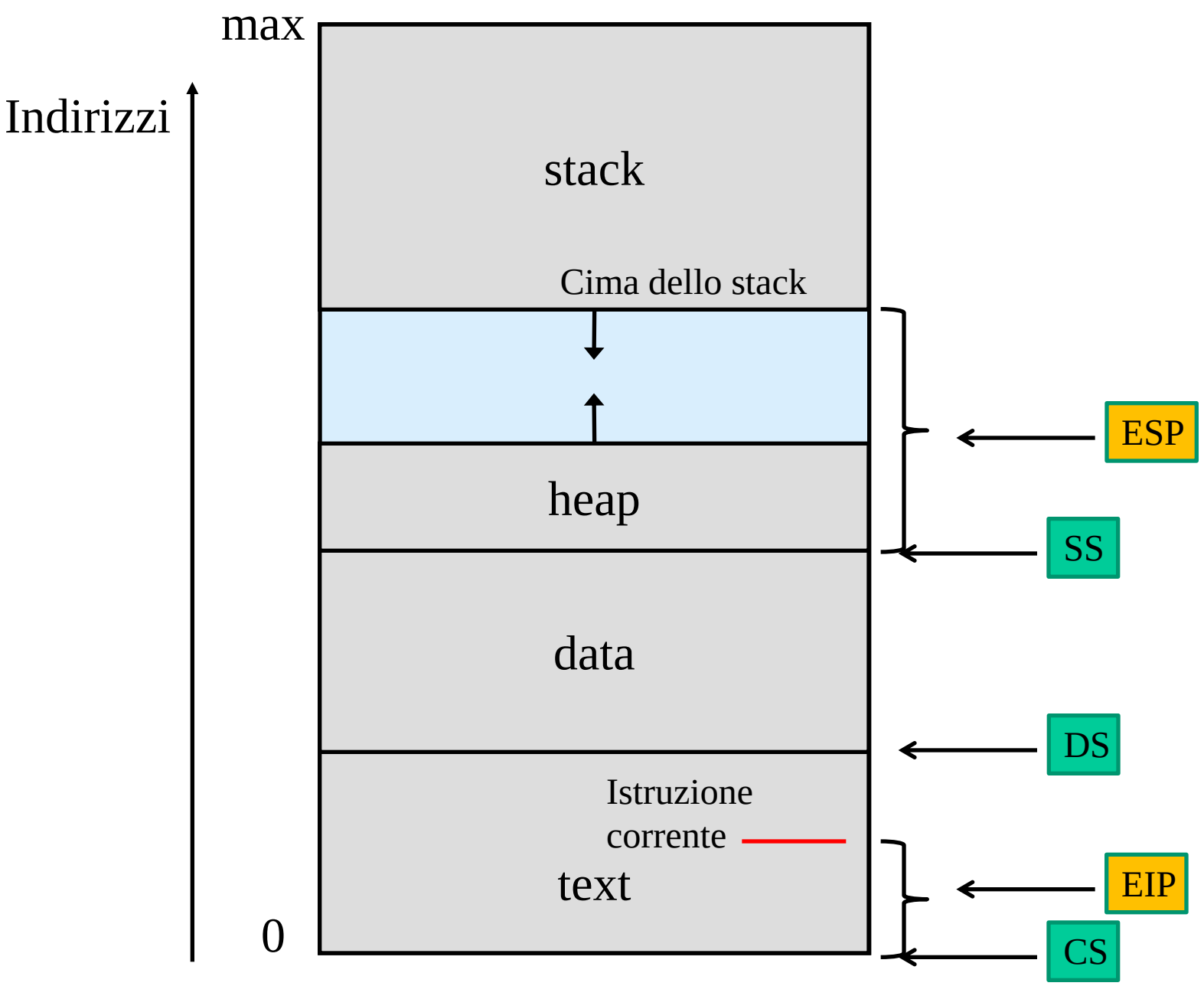
Record di attivazione
delle chiamate a funzione

Memoria allocata dinamicamente

Variabili globali

Codice eseguibile,
istruzioni macchina

Uso dei registri di segmento e di offset nella parte utente dei Processi



Stato di un processo in esecuzione

- Lo stato di un processo in esecuzione ad un determinato istante e' univocamente rappresentato
  - dal contenuto della memoria che il processo utilizza
  - e dal contenuto dei registri del processore.

- Salvare lo stato di un processo significa percio' memorizzare queste informazioni da qualche parte (in memoria o su disco) per poi recuperarle quando occorre far continuare l'esecuzione del processo.

- Riprendere l'esecuzione di un processo significa recuperare lo stato del processo, ripristinando il contenuto della memoria utilizzata dal processo e ripristinando il contenuto di tutti i registri.

- Salvare e ripristinare lo stato di un processo consente di intervallare l'esecuzione di piu' processi che potranno cosi' utilizzanre a turno la CPU (interleaving) dando un'impressione di un'esecuzione contemporanea.

Il concetto di Thread nei Processi