

 POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Memorie, processi, MATLAB (strutture)

Matteo Ferroni
matteo.ferroni@polimi.it

13/12/2016

Prima parte - Memorie e processi

(20') La gerarchia di memoria (richiamo teoria)

(10') Cache e tempi di accesso (Es.3 - TE 01/09/2015)

(20') Memoria fisica e memoria virtuale (Es.3 - TE ?)

(15') Stato dei processi (Es.4 - TE 19/02/2015)

La memoria cache

Il problema della memoria: costo vs. prestazioni

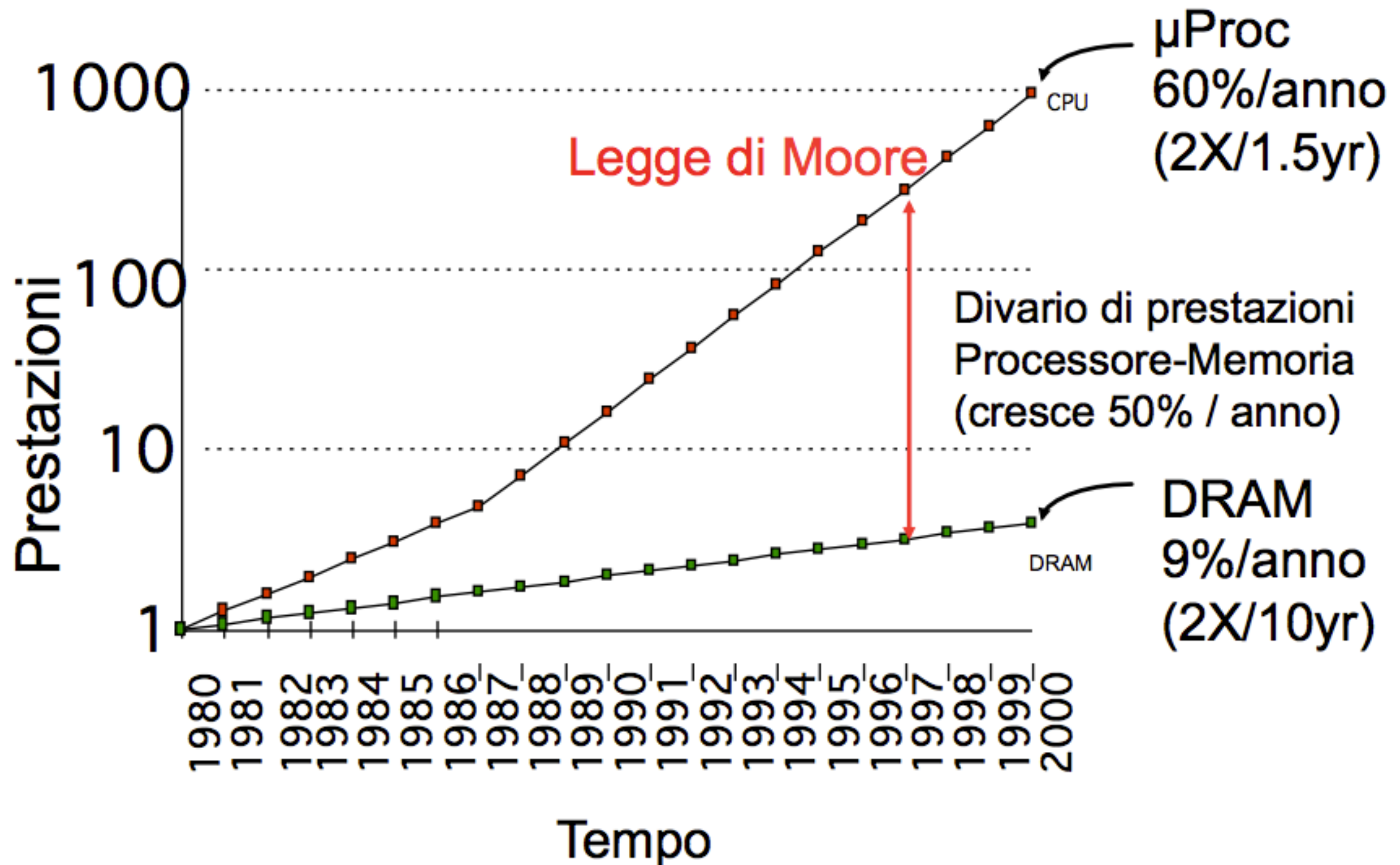
Obiettivo:

- fornire agli **utenti** una memoria grande e veloce
- fornire al **processore** i dati alla velocità con cui è in grado di elaborarli

Problema: Il tasso di crescita nella velocità dei processori non è stato seguito da quello delle memorie

- Tempo di accesso alle **SRAM** (Static Random Access Memory):
 - 2 - 25ns al costo di \$100 - \$250 per Mbyte.
- Tempo di accesso alle **DRAM** (Dynamic Random Access Memory):
 - 60 - 120ns al costo di \$5 - \$10 per Mbyte.
- Tempo di accesso al **disco**:
 - 10 - 20 million ns al costo di \$0.10 - \$0.20 per Mbyte.

Prestazioni di processori e cache



Gerarchia di memoria

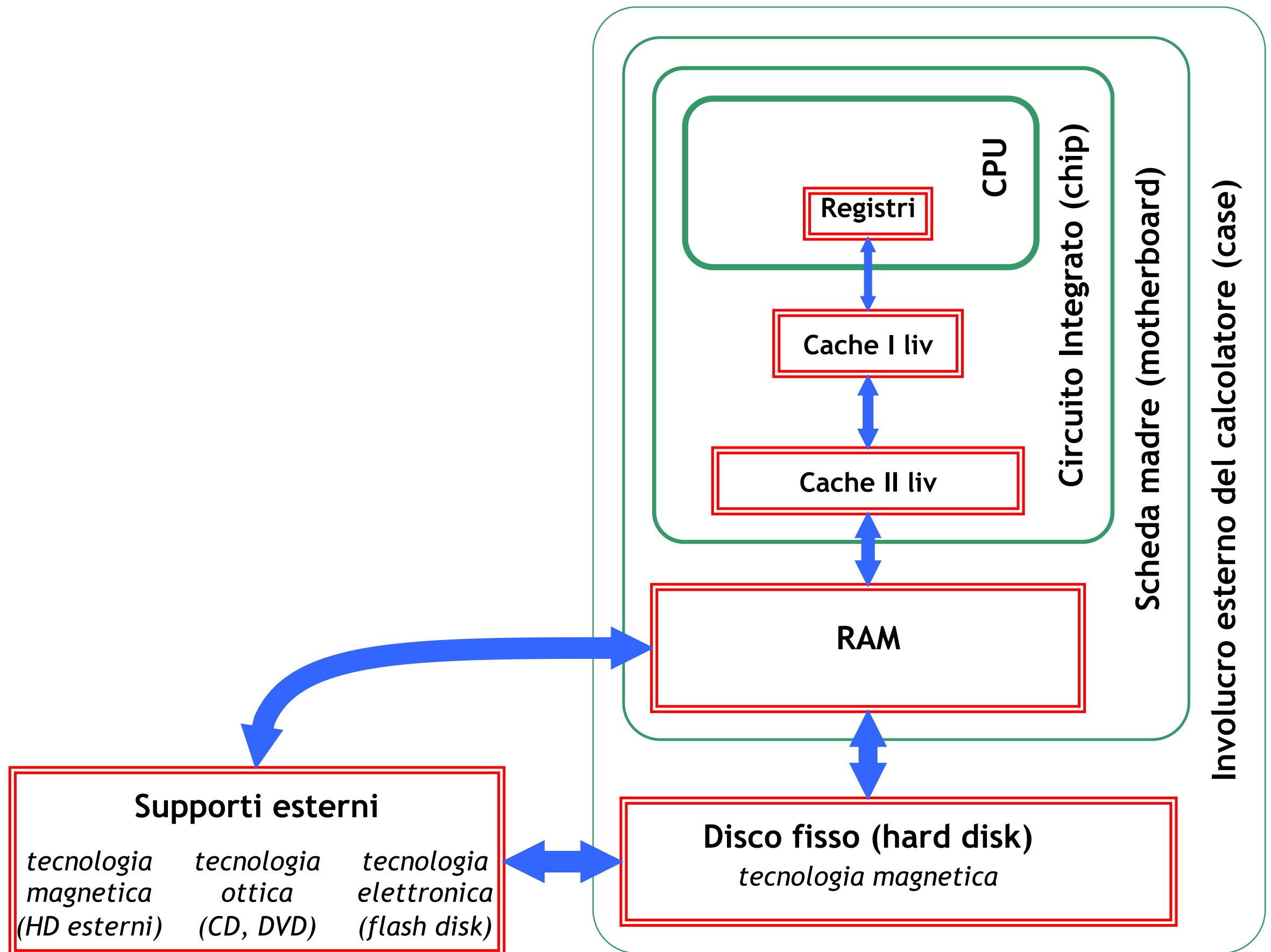
Tipicamente costituita da:

1. **registri** contenuti nella CPU (qualche KB)
2. **cache** (da circa 32KB a circa 4096KB)
3. **memoria principale** (da circa 512MB a qualche GB)
4. **dischi fissi** (da qualche centinaio di GB a qualche TB)
5. **nastri magnetici** e **dischi ottici** (da qualche centinaio di GB a qualche TB per ogni supporto)

Man mano che ci si sposta verso il basso nella gerarchia aumenta il valore dei parametri fondamentali:

- aumenta il **tempo** di accesso;
- aumenta la **capacità** di memorizzazione;
- ma diminuisce il **costo** per bit.

Una gerarchia di memoria



Caratteristiche dei diversi livelli

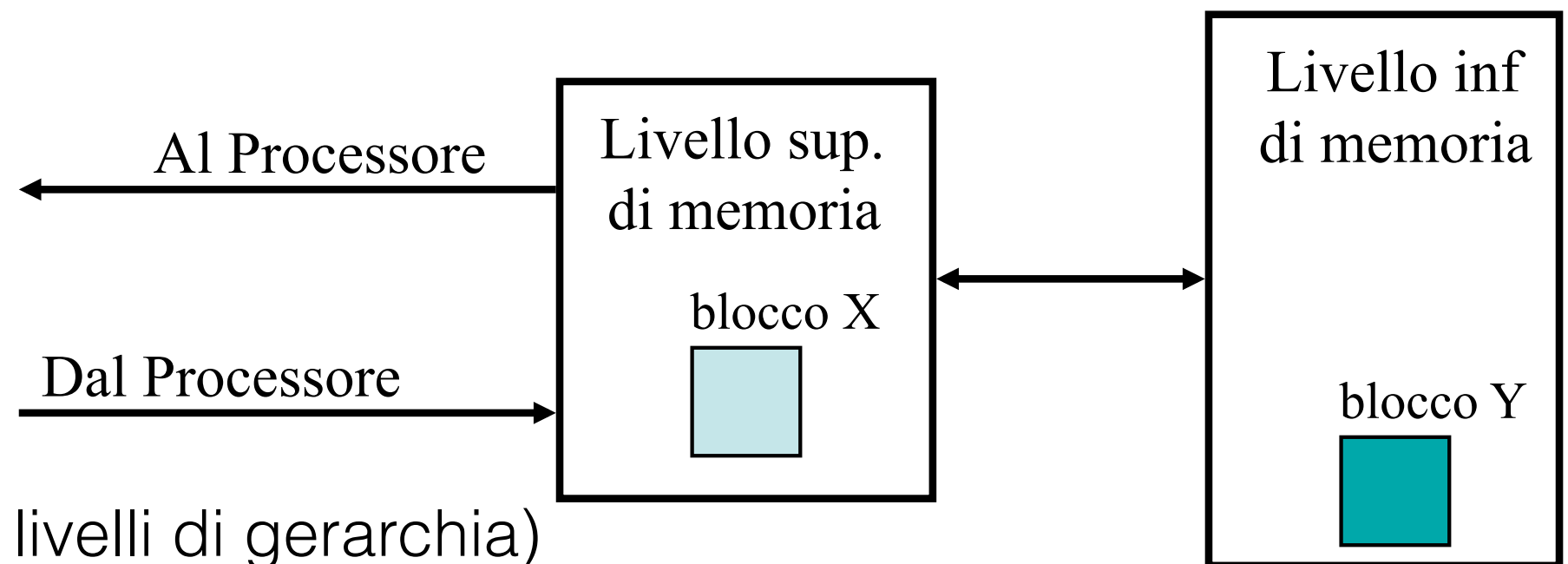
Livello	Capacità	Tempo di accesso	Transfer rate (GB/s)
Registri	~ 1 KB	~ 0.2 ns (1 ciclo di clock)	-
Cache I livello	~ 32 KB	~ 0.4 ns (2/4 cicli di clock)	-
Cache II livello	~ 1/2 MB	~ 1/2 ns (5/10 cicli di clock)	~ 100
Cache III livello	~ 2/8 MB	~ 5 ns	~ 50
Memoria centrale	~ 2/8 GB	~ 50 ns (1 ^a parola richiesta) ~ 10 ns (parole successive)	~ 5/10
Dischi interni	> 300 GB	~ 10 ms	0.15/0.6
Dischi esterni	> 300 GB	~ 10 ms	~ 0.05

Località

- E' la proprietà fondamentale dei programmi che rende possibile sfruttare l'organizzazione gerarchica della memoria per incrementarne le prestazioni
- **Località**: in ogni istante di tempo un programma accede a una parte relativamente piccola del suo spazio di indirizzamento
 - Località **temporale**: se un dato viene referenziato in un dato istante, è probabile che lo stesso dato venga nuovamente richiesto entro breve
 - Località **spaziale**: Se un dato viene utilizzato in un dato istante, è probabile che dati posizionati in celle di memoria adiacenti vengano richiesti entro breve

Cache

- Memoria al livello superiore della gerarchia, veloce ma piccola
- **Obiettivo**: fornire dati al processore in uno o due cicli di clock
- **Idea**: Sfruttare il **principio di località** dei programmi e tenere in memoria cache i dati utilizzati più di recente
- Il processore richiede un dato al sistema di memoria:
 - La richiesta viene *prima* inviata al livello di memoria **superiore** (più *vicino* al processore)
 - Se il dato non è presente nel livello **superiore** (*fallimento* della richiesta) la ricerca viene effettuata nel livello **inferiore**



(Si considerino solo due livelli di gerarchia)

Gerarchia di memoria: definizioni

- **Hit (successo)**: dati presenti in un blocco del livello superiore (esempio: Blocco X)
 - Hit **Rate** (“%” di successo): numero di accessi a memoria che trovano il dato nel livello superiore sul numero totale di accessi
 - Hit **Time (tempo di successo)**: tempo per accedere al dato nel livello superiore della gerarchia
- **Miss (fallimento)**: i dati devono essere recuperati dal livello inferiore della memoria (Blocco Y)
 - Miss **Rate** (“%” di fallimento) = $1 - (\text{Hit Rate})$
 - Miss **Penalty (tempo di fallimento)**: tempo per determinare il MISS + tempo necessario a sostituire un blocco nel livello superiore + tempo per trasferire il blocco al processore

Tipicamente si ha: **Hit Time** \ll **Miss Penalty**

Cache e principio di località

- **Tempo medio** di accesso in presenza di memoria cache:
 - semplicemente la **media pesata** con le probabilità

$$\text{HitTime} * \text{HitRate} + \text{MissRate} * \text{MissPenalty}$$

- Le memorie cache sfruttano il principio di **località spaziale** trasferendo dal livello inferiore della gerarchia più dati di quanti non ne siano stati strettamente richiesti (blocco o linea di cache)
- La **località temporale** viene sfruttata nella scelta del blocco da sostituire nella gestione di un fallimento (es: sostituire il blocco a cui si è fatto accesso meno di recente)

Prima parte - Memorie e processi

~~(20') La gerarchia di memoria (richiamo teoria)~~

(10') Cache e tempi di accesso (Es.3 - TE 01/09/2015)

(20') Memoria fisica e memoria virtuale (Es.3 - TE ?)

(15') Stato dei processi (Es.4 - TE 19/02/2015)

Cache e tempi di accesso (Es.3 - TE 01/09/2015)

- Un calcolatore è dotato di un sistema di memoria centrale e memoria cache con le seguenti caratteristiche:
 - **MissRate** = 25%
 - **MissPenalty** = 800 ns
 - **HitTime** = 40ns
 - **HitRate** = 1 - **MissRate** = 75%
- Rispondere alle seguenti domande (giustificando i risultati ottenuti con gli opportuni calcoli):
 - A. Calcolare il tempo medio di accesso ai dati del sistema con memoria cache.

$$\text{Tempo medio} = \text{HitRate} * \text{HitTime} + \text{MissRate} * \text{MissPenalty}$$

- B. Mantenendo invariati Miss Rate e Hit Time, qual è il valore massimo che può avere Miss Penalty affinché il tempo medio sia inferiore a 200 ns?

Cache e tempi di accesso (Es.3 - TE 01/09/2015)

- Un calcolatore è dotato di un sistema di memoria centrale e memoria cache con le seguenti caratteristiche:
 - **MissRate** = 25%
 - **MissPenalty** = 800 ns
 - **HitTime** = 40ns
 - **HitRate** = 1 - **MissRate** = 75%
- Rispondere alle seguenti domande (giustificando i risultati ottenuti con gli opportuni calcoli):
 - A. Calcolare il tempo medio di accesso ai dati del sistema con memoria cache.

$$\text{Tempo medio} = 0.75 * 40\text{ns} + 0.25 * 800\text{ns} = 30 + 200 = \mathbf{230 \text{ ns}}$$

- B. Mantenendo invariati Miss Rate e Hit Time, qual è il valore massimo che può avere Miss Penalty affinché il tempo medio sia inferiore a 200 ns?

$$\begin{aligned} 0.75 * 40 \text{ ns} + 0.25 * x &< 200 \text{ ns} \\ 30 \text{ ns} + 0.25 * x &< 200 \text{ ns} \\ x &< 170 / 0.25 \text{ ns} \\ x &< \mathbf{680 \text{ ns}} \end{aligned}$$

Prima parte - Memorie e processi

~~(20') La gerarchia di memoria (richiamo teoria)~~

~~(10') Cache e tempi di accesso (Es.3 - TE 01/09/2015)~~

(20') Memoria fisica e memoria virtuale (Es.3 - TE ?)

(15') Stato dei processi (Es.4 - TE 19/02/2015)

Memoria fisica e memoria virtuale (Es.3 - TE ?)

Parte A

- Un sistema dispone di 64 Kbyte di memoria fisica indirizzabile e di 128 Kbyte di memoria virtuale indirizzabile; inoltre la memoria virtuale è organizzata in pagine di 512 byte.
- Rispondere alle seguenti domande (giustificando i risultati ottenuti con gli opportuni calcoli):
 1. Quante sono le pagine di memoria virtuale?
 2. Definire la struttura dell'indirizzo virtuale e di quello fisico indicando la lunghezza dei campi che li costituiscono.

Memoria fisica e memoria virtuale (Es.3 - TE ?)

Parte A

- Un sistema dispone di 64 Kbyte di memoria fisica indirizzabile e di 128 Kbyte di memoria virtuale indirizzabile; inoltre la memoria virtuale è organizzata in pagine di 512 byte.

- Rispondere alle seguenti domande (giustificando i risultati ottenuti con gli opportuni calcoli):

1. Quante sono le pagine di memoria virtuale?

64 Kbyte memoria **fisica** = $64 \cdot 1024$ byte

128 Kbyte memoria **virtuale**

512 byte per **pagina** = 0,5 Kbyte

$64 / 0,5 = 128$ pagine di memoria **fisica**

$128 / 0,5 = 256$ pagine di memoria **virtuale**

Memoria fisica e memoria virtuale (Es.3 - TE ?)

2. Definire la struttura dell'indirizzo virtuale e di quello fisico indicando la lunghezza dei campi che li costituiscono.

512 byte per **pagina** = 0,5 Kbyte

- ogni byte deve essere singolarmente indirizzabile
- 512 “valori” diversi con una sola parola binaria
 - ho bisogno di $\text{ceil}(\log(2, 512)) = \mathbf{9 \text{ bit}}$

64/0,5 = 128 pagine di memoria **fisica**

- devo identificare ognuna delle 128 pagine di memoria virtuale
 - ho bisogno di $\text{ceil}(\log(2, 128)) = \mathbf{7 \text{ bit}}$

Indirizzo fisico = 9+7 = **16 bit**

128/0,5 = 256 pagine di memoria **virtuale**

- devo identificare ognuna delle 256 pagine di memoria virtuale
 - ho bisogno di $\text{ceil}(\log(2, 256)) = \mathbf{8 \text{ bit}}$

Indirizzo virtuale = 9+8 = **17 bit**

Memoria fisica e memoria virtuale (Es.3 - TE ?)

Parte B

- Si consideri un sistema con le seguenti caratteristiche:
 - Indirizzo virtuale di 8 bit
 - Indirizzo fisico di 7 bit
 - Dimensione pagine 16 byte
 - ogni byte deve essere singolarmente indirizzabile
 - 16 “valori” diversi con una sola parola binaria
 - ho bisogno di $\text{ceil}(\log(2, 16)) = \mathbf{4 \text{ bit}}$
- L'indirizzo virtuale 00101000 può corrispondere all'indirizzo fisico 0010100? Giustificare la risposta.

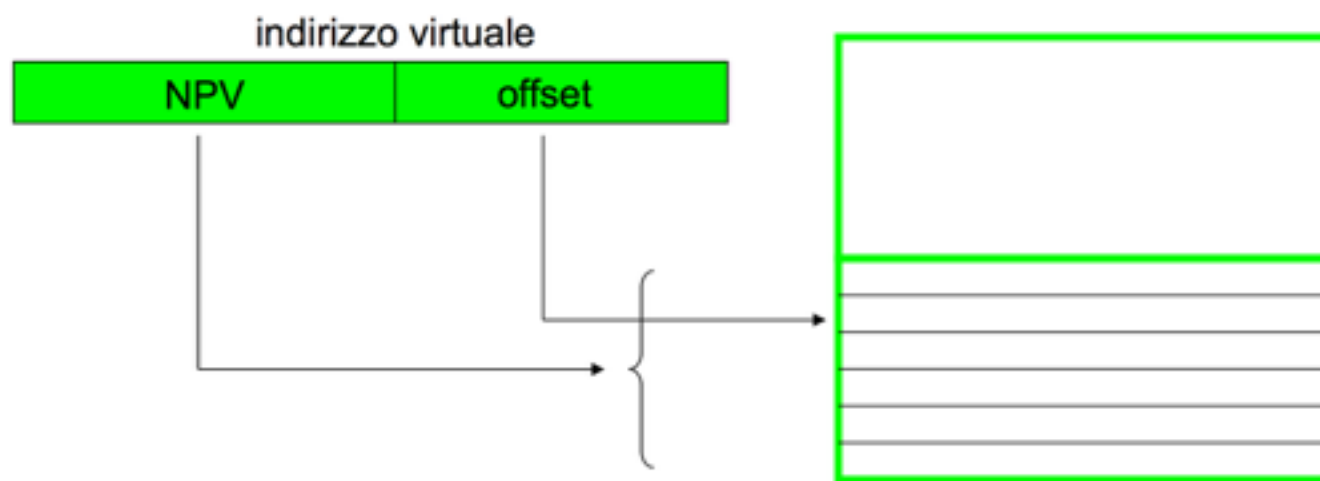
Memoria fisica e memoria virtuale (Es.3 - TE ?)

Parte B

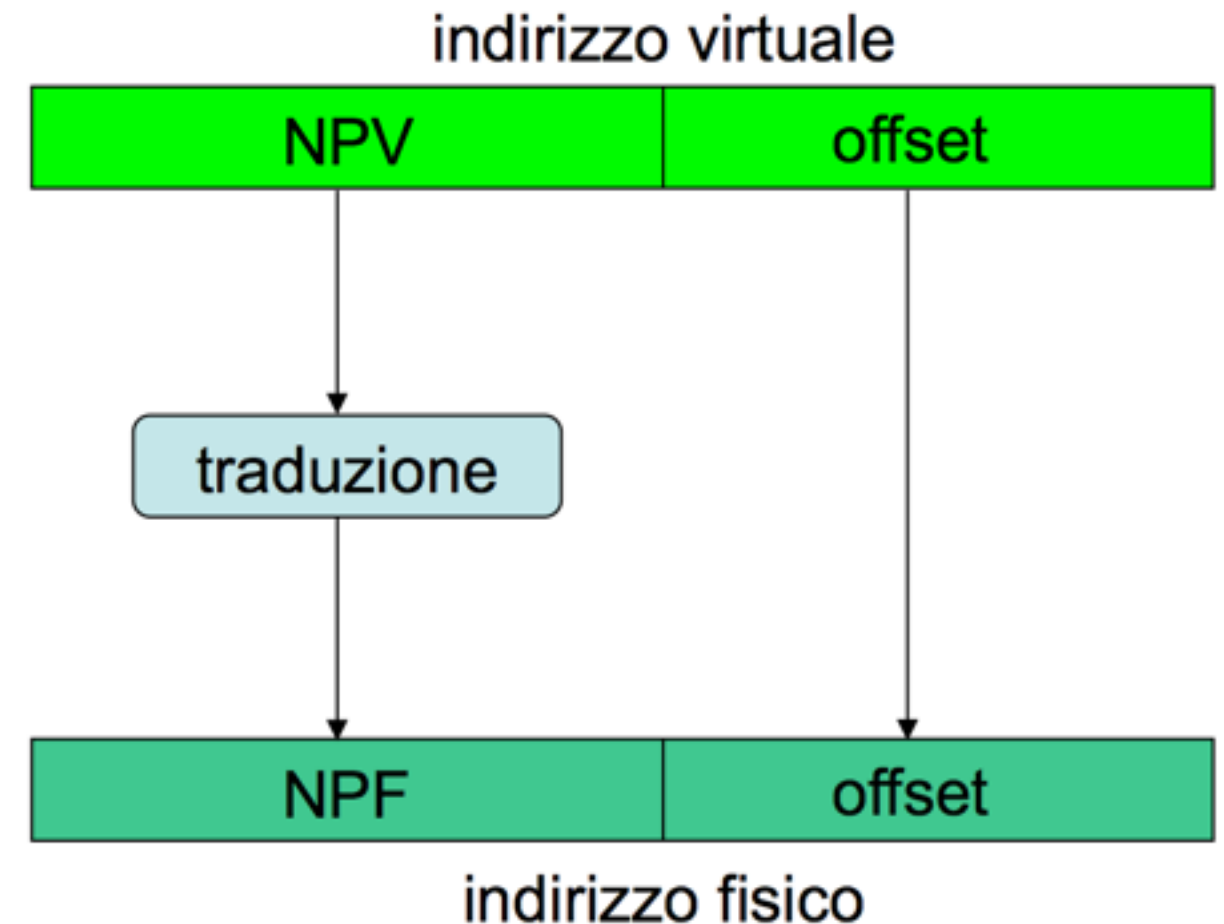
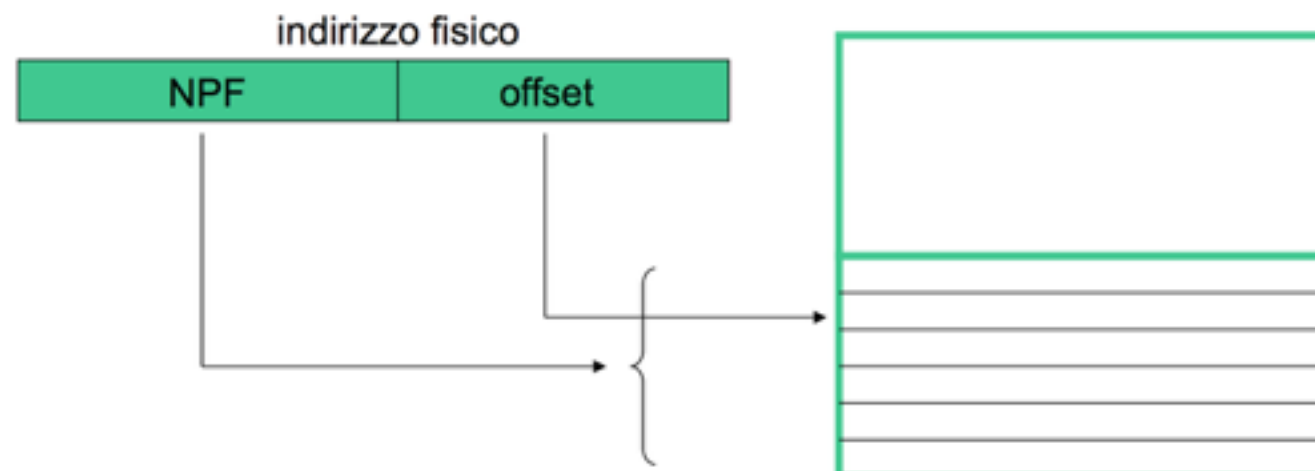
- Si consideri un sistema con le seguenti caratteristiche:
 - Indirizzo virtuale di 8 bit
 - Indirizzo fisico di 7 bit
 - Dimensione pagine 16 byte
 - ogni byte deve essere singolarmente indirizzabile
 - 16 “valori” diversi con una sola parola binaria
 - ho bisogno di $\text{ceil}(\log(2, 16)) = \mathbf{4 \text{ bit}}$
- L'indirizzo virtuale **00101000** può corrispondere all'indirizzo fisico **0010100**? Giustificare la risposta.

Memoria fisica e memoria virtuale (Es.3 - TE ?)

- Un indirizzo virtuale è costituito da un numero di pagina virtuale (NPV) e da uno spiazzamento (offset) all'interno della pagina



- E' del tutto analoga: si hanno un numero di pagina fisica (NPF) e da uno spiazzamento (offset) all'interno della pagina



le pagine virtuali e quelle fisiche hanno la stessa dimensione, quindi l'offset è lo stesso

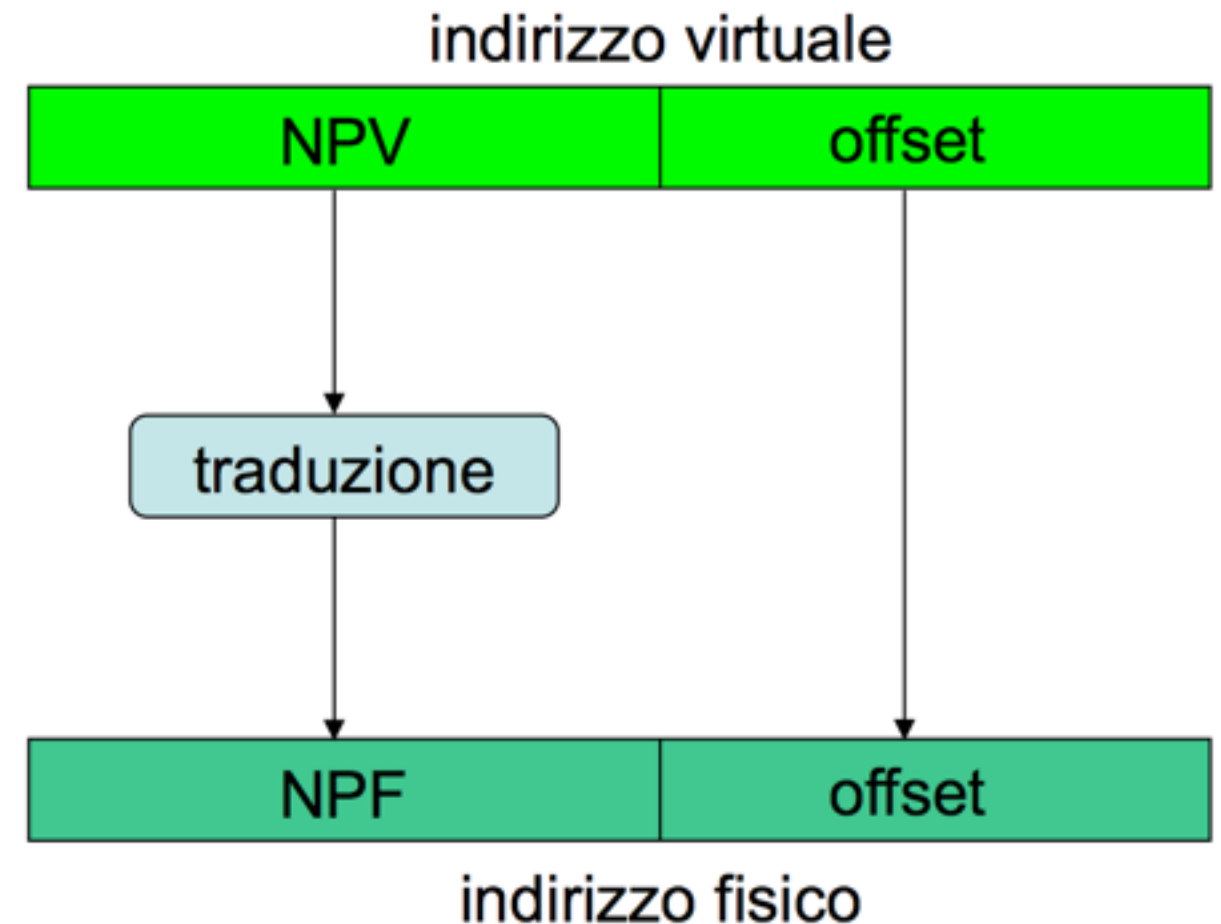
Memoria fisica e memoria virtuale (Es.3 - TE ?)

- Dimensione pagine 16 byte
-> **4 bit**
- L'indirizzo virtuale **00101000**
può corrispondere all'indirizzo
fisico **0010100**?

No!

Infatti, il primo indirizzo identifica il bit **1000** della pagina **0010** mentre il secondo identifica il bit **0100** della pagina **001**.

Siccome gli **spiazzamenti** dei bit all'interno delle pagine corrispondenti sono **sempre uguali** — cambiano infatti solo gli indici delle pagine — non abbiamo un match tra gli indirizzi.



le pagine virtuali e quelle fisiche hanno la stessa dimensione, quindi l'offset è lo stesso

Prima parte - Memorie e processi

~~(20') La gerarchia di memoria (richiamo teoria)~~

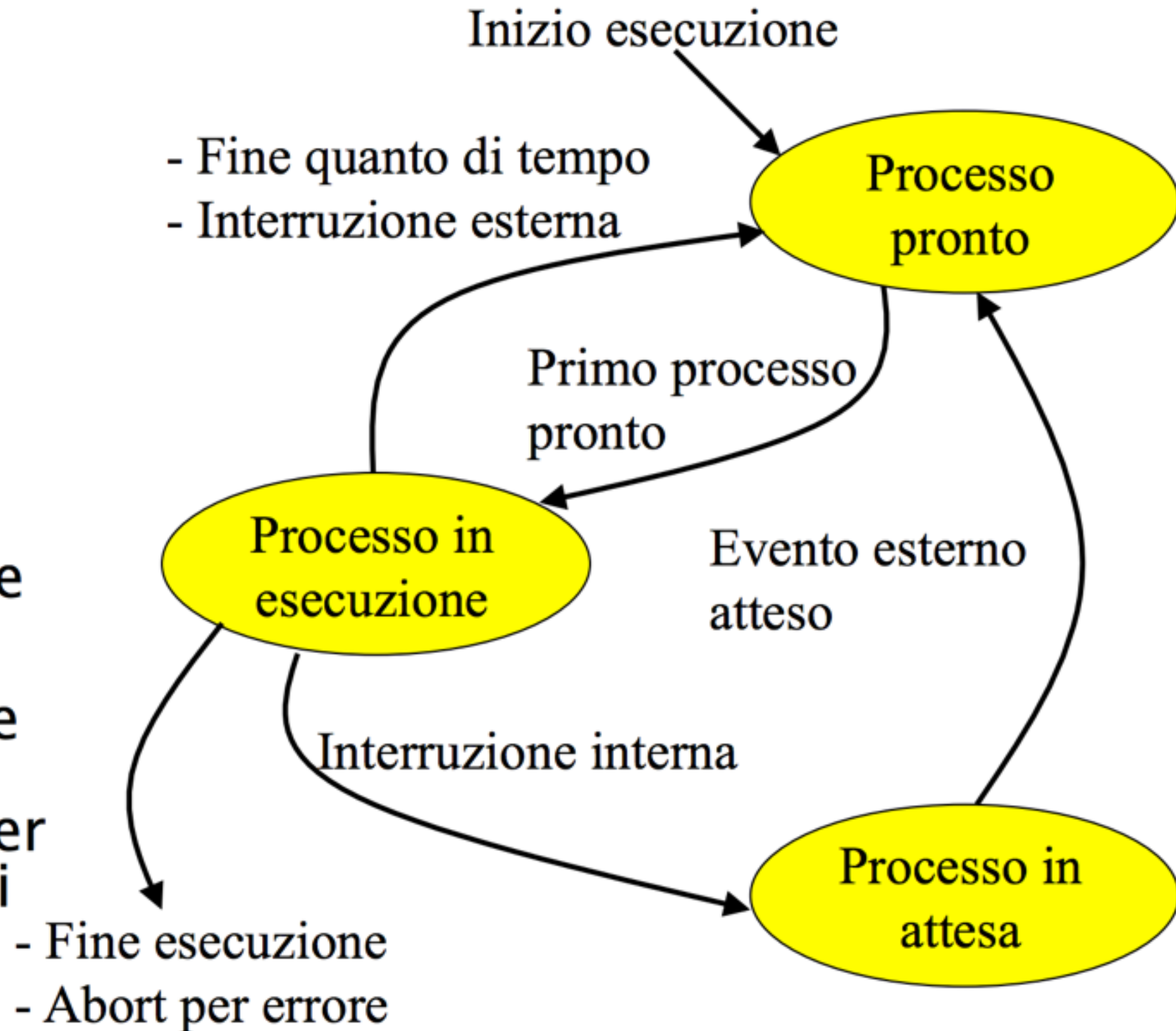
~~(10') Cache e tempi di accesso (Es.3 - TE 01/09/2015)~~

~~(20') Memoria fisica e memoria virtuale (Es.3 - TE ?)~~

(15') Stato dei processi (Es.4 - TE 19/02/2015)

Stato dei processi (teoria)

- **In esecuzione:** assegnato al processore ed eseguito da esso
- **Pronto:** può andare in esecuzione, se il gestore dei processi lo decide
- **In attesa:** attende il verificarsi di un evento esterno per andare in stato di pronto



Stato dei processi (Es.4 - TE 19/02/2015)

Si consideri un **sistema monoprocessore** in cui si avviano **quattro diversi processi** (A, B, C, D) che eseguono il seguente programma; si consideri ciascuna delle situazioni elencate qui sotto e si dica se può verificarsi.

```
#include <stdio.h>
int main() {
    int dato, risultato, i;

    while (1){
        risultato=0;
        printf("\n\nInserire un numero intero positivo da tastiera: ");
        scanf("%d", &dato);
        for (i=2; i<(dato/2)+1; i++)
            if (i*i==dato)
                risultato = i;
        printf("\nDato: %d\nRisultato: %d",dato, risultato);
    }
    return 0;
}
```

Stato dei processi (Es.4 - TE 19/02/2015)

1. A, B, C sono tutti e quattro in stato di pronto.

Si – **tutti** gli eseguibili **caricati in memoria** ed associati al rispettivo processo, **in attesa** che lo **scheduler** li metta in esecuzione

```
#include <stdio.h>
int main() {
    int dato, risultato, i;

    while (1){
        risultato=0;
        printf("\n\nInserire un numero intero positivo da tastiera: ");
        scanf("%d", &dato);
        for (i=2; i<(dato/2)+1; i++)
            if (i*i==dato)
                risultato = i;
        printf("\nDato: %d\nRisultato: %d",dato, risultato);
    }
    return 0;
}
```

Stato dei processi (Es.4 - TE 19/02/2015)

2. A, B, C sono tutti e quattro in stato di attesa.

Si – ad esempio tutti **bloccati sulla scanf**

```
#include <stdio.h>
int main() {
    int dato, risultato, i;

    while (1){
        risultato=0;
        printf("\n\nInserire un numero intero positivo da tastiera: ");
        scanf("%d", &dato);
        for (i=2; i<=(dato/2)+1; i++)
            if (i*i==dato)
                risultato = i;
        printf("\nDato: %d\nRisultato: %d",dato, risultato);
    }
    return 0;
}
```


Stato dei processi (Es.4 - TE 19/02/2015)

3. *A, B, sono in stato di esecuzione e C e D sono in stato di attesa.*

No – sistema **monoprocessore**

```
#include <stdio.h>
int main() {
    int dato, risultato, i;

    while (1){
        risultato=0;
        printf("\n\nInserire un numero intero positivo da tastiera: ");
        scanf("%d", &dato);
        for (i=2; i<(dato/2)+1; i++)
            if (i*i==dato)
                risultato = i;
        printf("\nDato: %d\nRisultato: %d",dato, risultato);
    }
    return 0;
}
```

Stato dei processi (Es.4 - TE 19/02/2015)

4. *A è in stato di attesa, B è in stato di esecuzione, C e D sono in stato di pronto.*

Si – ad esempio: **A è bloccata** sulla scanf, **B è in esecuzione** sul processore, **C e D sono pronti**, in attesa che lo scheduler li metta in esecuzione

```
#include <stdio.h>
int main() {
    int dato, risultato, i;

    while (1){
        risultato=0;
        printf("\n\nInserire un numero intero positivo da tastiera: ");
        scanf("%d", &dato);
        for (i=2; i<(dato/2)+1; i++)
            if (i*i==dato)
                risultato = i;
        printf("\nDato: %d\nRisultato: %d",dato, risultato);
    }
    return 0;
}
```


Prima parte - Memorie e processi

~~(20') La gerarchia di memoria (richiamo teoria)~~

~~(10') Cache e tempi di accesso (Es.3 TE 01/09/2015)~~

~~(20') Memoria fisica e memoria virtuale (Es.3 - TE ?)~~

~~(15') Stato dei processi (Es.4 TE 19/02/2015)~~

Seconda parte - MATLAB: strutture ed esercizi

(20') Strutture e array di strutture (teoria)

(10') Esercizio: rilievi altimetrici

(20') Esercizio: vettori e funzioni base

(20') Esercizio: vettori, vettori everywhere!



Una **struttura** è un tipo di dato composto da elementi eterogenei

- Ogni elemento individuale è chiamato **campo** e ha un nome
- Come con gli scalari, si può passare da un elemento singolo (matrice 1×1) a un vettore (matrice $1 \times n$)
- Ci sono due modi per creare una struttura:
 - Campo per campo mediante assegnamento
 - Tutto in una volta mediante la funzione struct



- Esempio: la struttura studente
 - `studente.nome = 'Giovanni Rossi';`
 - `studente.indirizzo = 'Via Roma 23';`
 - `studente.citta = 'Cosenza';`
 - `studente.media = 25;`
 - `whos studente`

Name	Size	Bytes	Class	Attributes
studente	1x1	568	struct	

- `%aggiungo un nuovo studente... -> array 1x2`
- `studente(2).nome = 'Giulia Gatti';`
- `studente(2).media = 30;`
- Nota: quando un elemento viene definito, tutti i suoi campi sono creati e inizializzati a valore nullo (vettore vuoto `[]`)



- Consente di preallocare una struttura o un array di strutture
 - `str_array = struct('campo1', val1, 'campo2', val2, ...)`
- Esempio

```
>> rilievoAltimetrico=struct('latitudine',20,'longitudine',30, 'altitudine', 1300)
```

```
rilievoAltimetrico =  
  latitudine: 20  
  longitudine: 30  
  altitudine: 1300
```

Creazione di array di strutture

- Se si allunga un array assegnando un valore a una componente di indice $>$ dimensione corrente
 - ▶ i nuovi elementi, in posizione precedente a quello inserito esplicitamente, vengono inizializzati al solito valore 'nullo' []

Creazione di array di strutture

- Se si allunga un array assegnando un valore a una componente di indice $>$ dimensione corrente
 - ▶ i nuovi elementi, in posizione precedente a quello inserito esplicitamente, vengono inizializzati al solito valore 'nullo' []
- Esempio
 - ▶ `rilieviAltimetrici(1000)=struct('latitudine',80,'longitudine',[],`
`'altitudine', 1450)`
 - `rilieviAltimetrici =`
 - 1x1000 struct array with fields
 - `latitudine`
 - `longitudine`
 - `altitudine`



- Se si allunga un array assegnando un valore a una componente di indice $>$ dimensione corrente
 - ▶ i nuovi elementi, in posizione precedente a quello inserito esplicitamente, vengono inizializzati al solito valore 'nullo' []

- Esempio

- ▶ `rilieviAltimetrici(1000)=struct('latitudine',80,'longitudine',[],`
`'altitudine', 1450)`

- `rilieviAltimetrici =`
 - 1x1000 struct array with fields:
 - `latitudine`
 - `longitudine`
 - `altitudine`

Array vuoto. Attenzione: se si inserisce un valore (es. 20), questo viene assunto dal campo `longitudine` dell'elemento 1000, ma non dallo stesso campo degli altri elementi dell'array



- Aggiunta di un campo: facciamo riferimento alla definizione di studente delle slide precedenti
 - `studente(2).esami = [20 25 30];`



- Aggiunta di un campo: facciamo riferimento alla definizione di studente delle slide precedenti
 - `studente(2).esami = [20 25 30];`
- Il campo esami viene aggiunto a tutte le strutture che fanno parte di studente
 - Avrà un valore iniziale per `studente(2)`. Sarà vuoto per tutti gli altri elementi dell'array

Uso dei dati nelle strutture

- Notazione con il “punto”, uguale al C
- Esempi
 - `studente(2).nome`
 - `studente(2).esami(2)`
 - `unNome = studente(1).nome`
 - `studente(2).indirizzo=studente(1).indirizzo`
 - `%mean` calcola la media degli elementi di un array
 - `mean(studente(2).esami)`

Uso dei dati nelle strutture

- Notazione con il “punto”, uguale al C
- Esempi
 - `studente(2).nome`
 - `studente(2).esami(2)`
 - `unNome = studente(1).nome`
 - `studente(2).indirizzo=studente(1).indirizzo`
 - `%mean` calcola la media degli elementi di un array
 - `mean(studente(2).esami)`
- Estrazione dei valori che un campo assume in tutti gli elementi di un array di strutture (NB: ipotizziamo che le strutture dell'array *studente* abbiano un campo 'media' e che l'array abbia due componenti)
 - `a = [studente.media]` \longrightarrow `a = [25 30]`



- Un campo di un array di strutture può essere di qualsiasi tipo (come in C)
- E` quindi possibile avere un campo che è, di nuovo, una struttura. Esempio
 - `studente(1).corso(1).nome= 'InformaticaB' ;`
 - `studente(1).corso(1).docente= 'Von Neumann' ;`
 - `studente(1).corso(2).nome= 'Matematica' ;`
 - `studente(1).corso(2).docente= 'Eulero' ;`

Seconda parte - MATLAB: strutture ed esercizi

~~(20') Strutture e array di strutture (teoria)~~

(10') Esercizio: rilievi altimetrici

(20') Esercizio: vettori e funzioni base

(20') Esercizio: vettori, vettori everywhere!



- Si sviluppi un programma in matlab che acquisisce da tastiera i dati relativi a rilievi altimetrici e stampa a video l'altitudine media di tutti quelli che hanno latitudine compresa tra 10 e 80 e longitudine tra 30 e 60


```
more = input('vuoi inserire valori altimetrici? (s/n)');
```

[illegible]



```
more = input('vuoi inserire valori altimetrici? (s/n)');  
ii=1;  
while more=='s'  
    arch(ii).altitudine = input('altitudine ');  
    arch(ii).longitudine = input('longitudine ');  
    arch(ii).latitudine = input('latitudine ');  
  
end
```



```
more = input('vuoi inserire valori altimetrici? (s/n)');  
ii=1;  
while more=='s'  
    arch(ii).altitudine = input('altitudine ');  
    arch(ii).longitudine = input('longitudine ');  
    arch(ii).latitudine = input('latitudine ');  
  
    more = input('vuoi inserire altri valori altimetrici? (s/n)');  
end
```



```
more = input('vuoi inserire valori altimetrici? (s/n)');  
ii=1;  
while more=='s'  
    arch(ii).altitudine = input('altitudine ');  
    arch(ii).longitudine = input('longitudine ');  
    arch(ii).latitudine = input('latitudine ');  
    ii = ii+1;  
    more = input('vuoi inserire altri valori altimetrici? (s/n)');  
end
```



```
jj=1;  
for ii=1:length(arch)
```

```
end
```



```
jj=1;  
for ii=1:length(arch)  
    %attenzione: la condizione deve essere scritta sulla stessa  
    linea...  
    if arch(ii).latitudine>=10&&arch(ii).latitudine<=80 &&  
        arch(ii).longitudine>=30&&arch(ii).longitudine<=60  
  
    end  
end
```




```
jj=1;
for ii=1:length(arch)
    %attenzione: la condizione deve essere scritta sulla stessa
    linea...
    if arch(ii).latitudine>=10&&arch(ii).latitudine<=80 &&
        arch(ii).longitudine>=30&&arch(ii).longitudine<=60
        elemSelez(jj) = arch(ii).altitudine;
        jj=jj+1;
    end
end
```



```
jj=1;
for ii=1:length(arch)
    %attenzione: la condizione deve essere scritta sulla stessa
    linea...
    if arch(ii).latitudine>=10&&arch(ii).latitudine<=80 &&
        arch(ii).longitudine>=30&&arch(ii).longitudine<=60
        elemSelez(jj) = arch(ii).altitudine;
        jj=jj+1;
    end
end
disp(['la media degli elementi selezionati e ` '
    num2str(mean(elemSelez))]);
```

Seconda parte - MATLAB: strutture ed esercizi

~~(20') Strutture e array di strutture (teoria)~~

~~(10') Esercizio: rilievi altimetrici~~

(20') Esercizio: vettori e funzioni base

(20') Esercizio: vettori, vettori everywhere!

Esercizio: Vettori e funzioni base

Scrivere un programma che permetta all'utente di inserire un vettore di numeri interi. Dopo aver verificato che l'array inserito sia numerico, effettuare i seguenti controlli:

- Verificare se tutti i numeri sono positivi
- Verificare se esiste un numero negativo
- Applicare la radice quadrata a tutti i valori: ci sono dei valori complessi?
- Verificare se tutti i numeri sono pari e trovarne le posizioni
- Verificare se esiste un numero dispari
- Contare i numeri dispari se esistono e dire in che posizione sono

Esercizio: Vettori e funzioni base

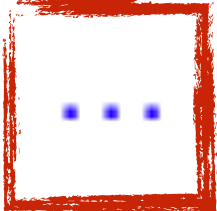
```
% Scrivere un programma che permetta all'utente di  
% inserire un vettore di numeri interi.  
  
% Dopo aver verificato che l'array inserito sia numerico,  
%effettuare i seguenti controlli:
```

Esercizio: Vettori e funzioni base

```
% Scrivere un programma che permetta all'utente di  
% inserire un vettore di numeri interi.  
vett = input('Inserisci un vettore: ');  
  
% Dopo aver verificato che l'array inserito sia numerico,  
%effettuare i seguenti controlli:
```

Esercizio: Vettori e funzioni base

```
% Scrivere un programma che permetta all'utente di
% inserire un vettore di numeri interi.
vett = input('Inserisci un vettore: ');

% Dopo aver verificato che l'array inserito sia numerico,
%effettuare i seguenti controlli:
if isnumeric(vett)
    
else
    disp('Devi inserire un array numerico');
end
```

Esercizio: Vettori e funzioni base

```
% Verificare se tutti i numeri sono positivi
```


Esercizio: Vettori e funzioni base

```
% Verificare se tutti i numeri sono positivi
if all(vett > 0)
    disp('tutti i numeri sono positivi');
else
    disp('non tutti i numeri sono positivi');
end

% Verificare se esiste un numero negativo
```

Esercizio: Vettori e funzioni base

```
% Verificare se tutti i numeri sono positivi
if all(vett > 0)
    disp('tutti i numeri sono positivi');
else
    disp('non tutti i numeri sono positivi');
end

% Verificare se esiste un numero negativo
if any(vett < 0)
    disp('esiste un numero negativo');
else
    disp('non esiste alcun numero negativo');
end
```

Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:  
% ci sono dei valori complessi?
```

Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:  
% ci sono dei valori complessi?  
sqrt_vett = sqrt(vett)
```

Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:  
% ci sono dei valori complessi?  
sqrt_vett = sqrt(vett)  
if isreal(sqrt_vett)  
    disp('non esistono valori complessi');  
else  
    disp('esistono valori complessi');  
end  
  
% Verificare se tutti i numeri sono pari e  
% trovarne le posizioni
```

Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:  
% ci sono dei valori complessi?  
sqrt_vett = sqrt(vett)  
if isreal(sqrt_vett)  
    disp('non esistono valori complessi');  
else  
    disp('esistono valori complessi');  
end  
  
% Verificare se tutti i numeri sono pari e  
% trovarne le posizioni  
vett_mod = mod(vett,2);
```

Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:
% ci sono dei valori complessi?
sqrt_vett = sqrt(vett)
if isreal(sqrt_vett)
    disp('non esistono valori complessi');
else
    disp('esistono valori complessi');
end

% Verificare se tutti i numeri sono pari e
% trovarne le posizioni
vett_mod = mod(vett,2);
if(all(vett_mod==0))
    disp('tutti i numeri sono pari');
else
    disp('non tutti i numeri sono pari');
end
```


Esercizio: Vettori e funzioni base

```
% Applicare la radice quadrata a tutti i valori:
% ci sono dei valori complessi?
sqrt_vett = sqrt(vett)
if isreal(sqrt_vett)
    disp('non esistono valori complessi');
else
    disp('esistono valori complessi');
end

% Verificare se tutti i numeri sono pari e
% trovarne le posizioni
vett_mod = mod(vett,2);
if(all(vett_mod==0))
    disp('tutti i numeri sono pari');
else
    disp('non tutti i numeri sono pari');
end
pos_pari = find(1-vett_mod);
disp('I numeri pari sono in posizione: ');
disp(pos_pari);
```

Esercizio: Vettori e funzioni base

```
% Verificare se esiste un numero dispari
```

Esercizio: Vettori e funzioni base

```
% Verificare se esiste un numero dispari
if any(vett_mod)
    disp('esiste almeno un numero dispari');

    % Contare i numeri dispari se esistono e dire in che posizioni sono

else
    disp('non esistono numeri dispari');
end
```

Esercizio: Vettori e funzioni base

```
% Verificare se esiste un numero dispari
if any(vett_mod)
    disp('esiste almeno un numero dispari');

    % Contare i numeri dispari se esistono e dire in che posizioni sono
    count_dispari = sum(vett_mod);
    disp(['i numeri dispari sono ' num2str(count_dispari)]);

else
    disp('non esistono numeri dispari');
end
```

Esercizio: Vettori e funzioni base

```
% Verificare se esiste un numero dispari
if any(vett_mod)
    disp('esiste almeno un numero dispari');

    % Contare i numeri dispari se esistono e dire in che posizioni sono
    count_dispari = sum(vett_mod);
    disp(['i numeri dispari sono ' num2str(count_dispari)]);
    pos_dispari = find(vett_mod);
    disp('i numeri dispari sono in posizione: ');
    disp(pos_dispari);
else
    disp('non esistono numeri dispari');
end
```

Seconda parte - MATLAB: strutture ed esercizi

~~(20') Strutture e array di strutture (teoria)~~

~~(10') Esercizio: rilievi altimetrici~~

~~(20') Esercizio: vettori e funzioni base~~

(20') Esercizio: vettori, vettori everywhere!