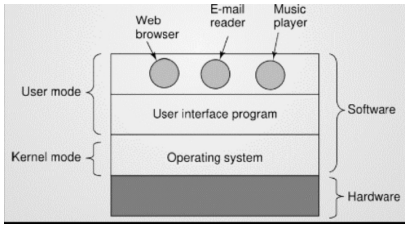
09-03-2023

Riprendendo il discorso...

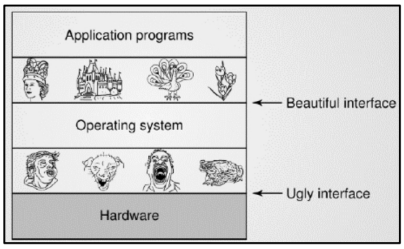
Il SO è un tipo di **software speciale** che fa da **cuscinetto** fra programmi e hardware e ha il compito di attivare l'hardware in base alle richies### Riprendendo il discorso...

Il SO è un tipo di **software speciale** che fa da **cuscinetto** fra programmi e hardware e ha il compito di attivare l'hardware in base alle richieste dell'utente.
Ci sono 2 modalità d'uso:

- **Kernel mode**, ha accesso a tutto ciò che esiste, anche a risorse delicate ed è eseguito dal SO.
- **User mode**, non si ha accesso alle parti più delicate, profonde, della memoria perchè si potrebbe compromettere l'intero funzionamento del sistema.



- Un SO è fortemente legato all'architettura sulla quale opera. Esso usa il concetto di **astrazione** attraverso il quale non ci si deve preoccupare di quello che c'è "sotto" (*uso di un file*)
- Il compito principale del SO è rendere bello ciò che bello non è attraverso l'astrazione, quindi **nascondere la complessità dell'hardware**.



- Si deve avere una visione tale da poter allocare in **maniera ordinata** e **controllata** per sapere in maniera **veloce** ed efficiente come mettere in comunicazione queste componenti. Devono anche gestire l'esecuzione contemporanea di molteplici programmi (*multitask*).

Sistema operativo come gestore di risorse

Per quanto riguarda la condivisione delle risorse si parla di **MULTIPLEX**. Può essere in due modalità:

1. rispetto al **TEMPO**: i programmi si alternano fra loro in modo da concedersi la risorsa
2. rispetto allo **SPAZIO**: ogni processo prende una porzione di risorsa in termini di spazio (utenti che si suddividono la stessa memoria).

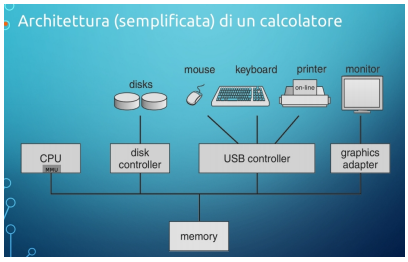
Evoluzione dei SO

Vi furono diverse *generazioni*:

1. `1945-55` : Schede a perforazione. linguaggio macchina. nessun SO e si facevano semplici calcoli matematici

2. `1955-65` : transistor, **mainframe**. I pc diventano più affidabili. Erano grandi ed erano chiusi in stanze dedicate e gestite da operatori professionisti. Acquisiti solo da industrie o università (*erano costosi*). Ogni programma era detto **JOB** in *assembly*. Un processo veniva svolto uno dopo l'altro seguendo la logica di una coda e quest'operazione prendeva il nome di **Batch**. Si facevano equazioni differenziali. Si raccoglievano tutti i job.
    - In questa era ci sono due diversi calcolatori che lavorano insieme in *stanze diverse*.
3. `1965-80` : si cerca di **incorporare** le due attività all'interno di un solo calcolatore piccolo, accessibile. Si vogliono inglobare le attività. Si usano i circuiti integrati e nasce l'**unica** macchina `IBM SYSTEM/360` di piccole dimensioni rispetto a prima che **ingloba le due attività**.
   In questa *terza generazione* ci sono diverse tecniche:
   - gestione della **MULTIPROGRAMMAZIONE**: esecuzione di processi contemporaneamente. Quindi si cerca di occupare tutta la memoria per migliorare le **prestazioni**.
   - **Timesharing**: necessità di far sì che più utenti lavorino allo stesso tempo e la prima macchina di questo tipo fu **MULTICS** ma ebbe poco successo. Da essa si iniziò lo sviluppo di **UNIX**.
   I sistemi erano **COMPLESSI** e scritti in **linguaggio macchina** (**assembly**) quindi commettere errori era molto semplice.
4. `1980-oggi` : Nasce il primo SO **MS-DOS** puramente da usato da **Shell**. *Windows 3.1* fu la prima versione **PARZIALMENTE GRAFICA**. Windows 95 era il primo sistema operativo **INTERAMENTE GRAFICO**.
5. `1990-oggi` : *Mobile computers*, *iOS*, *Android*, *Windows Mobile*

Hardware



L'hardware è collegato da vari **bus** e ce ne sono diversi tipi in base al dato che viaggia su una determinata linea: bus dati ecc...
Come vengono messi in comunicazione i vari componenti dal SO? *Qual è la complessità di tali operazioni*? Viene imposto un **vincolo**: le operazioni devono **ridurre il più possibile i costi** in termini di **TEMPO**.

- Aprire un file in lettura è più "leggera" come operazione rispetto alla scrittura.

Processore

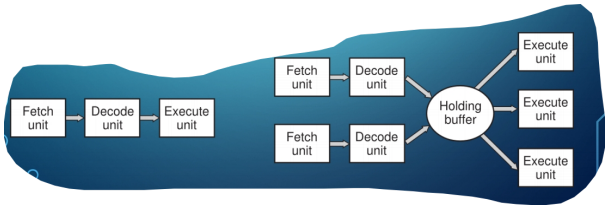
Ciclicamente preleva le istruzioni (**fetch**), le **decodifica** ed esegue le **operazioni**. E' formato da tanti registri utili per memorizzare le variabili più importanti, risultati temporanei (tipo risultati da operazioni aritmetiche):

- Program Counter (PC): contiene memorizzata l'istruzione successiva da eseguire. La fetch preleva l'istruzione da questo registro
- Stack Pointer (SP): punta alla cima di uno stack usato per memorizzare i frame per ogni procedura che è stata eseguita ma non ancora completata (variabili locali, file aperti letti e altro ancora...).
    - *Quando si blocca un programma, il SO deve tenere in mente **tutto quello che ha fatto** fino a quel **momento** e dove si è fermato in modo tale da **riprendere** dal punto preciso.*

- Program Status Word (PSW): registro a bit che contiene la priorità della CPU, la modalità di esecuzione di un programma e altro..

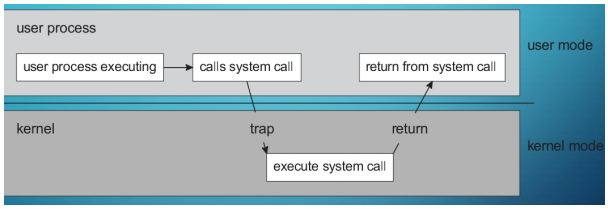
Esistono diverse progettazioni avanzate:

- **PIPELINE**, vuol dire che si eseguono **più istruzioni nello stesso tempo**:
  - *Fetch->Decode->Execute*
- **CPU SUPERSCALARE**: si ha un'unità specializzata a vari tipi di operazioni: booleane, float, interi ecc... Ciò dipende dall'operazione che si deve fare, appunto, ed è più efficiente. Dal buffer (che fa da coda) verranno prelevate le varie operazioni da eseguire.



Chiamata di sistema

Quando si ha una **chiamata di sistema**, un programma utente va in esecuzione e, quando si deve accede a qualcosa di delicato (non concessa all'utente), si attiva una chiamata di sistema attraverso un'istruzione **TRAP** si avviene il passaggio del comando da modalità utente a modalità kernel e chiaramente ogni TRAP ha un proprio costo.
Successivamente vengono eseguite le operazioni (tipo disabilitazione degli interrupt) e dopodiché si ritorna alla modalità utente.



- Il sistema operativo, in modalità kernel, sa esattamente come gestire le situazioni più delicate a differenza dell'utente.

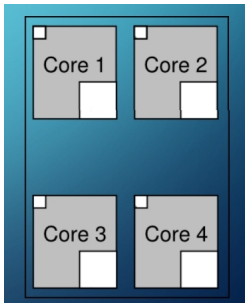
> *E' un grosso rischio assegnare all'utente di gestre la memoria.*

- Più si limitano le operazioni che può fare l'utente, più trap devo usare e quindi più spazio di memoria devo occupare quindi non conviene esagerare.
- Spesso le TRAP vengono causate dall'HardDisk (tipo la divisione per 0)

Più processori

Si possono avere **multiprocessori multicore** oppure si possono avere (*in entrambi i casi*) si possono avere sistemi **multithread**, cioè è come se un programma che viene eseguito, viene diviso in varie parti diverse fra loro. Parti diverse (thread) vengono eseguite come se venissero eseguite insieme(l'esecuzione avviene sempre nel tempo assegnato al programma in sè) in uno **pseudo-parallelismo**.
Ogni parte prende il nome di **THREAD** e **ognuno è differente dall'altro** in modo tale che ogni parte

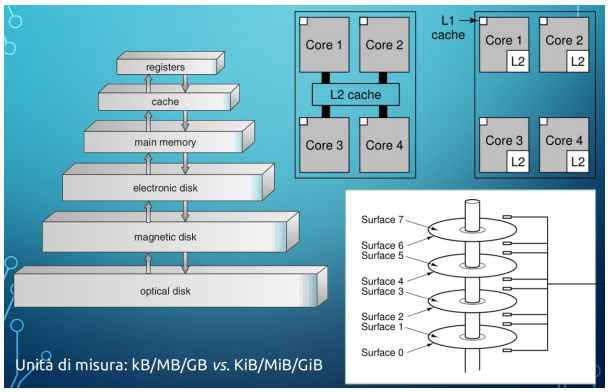
abbia il **proprio compito**.



- Se il SO vede un programma **senza thread** e un pezzo di programma richiede qualcosa che la manda in attesa, esso manda l'intero programma in attesa.
- Se invece ci sono i thread (che non interagiscono fra di loro) e solo un thread blocca l'esecuzione, il SO decide di far andare avanti altri thread e bloccando solo un pezzo fino alla sua liberazione. Nello stesso tempo si può eseguire parti di programma e far andare avanti le varie operazioni.
- Il **SO deve tenere conto dei thread** in modo da poterli sfruttare al massimo. Se non li conosce e si blocca un terzo flusso(thread) allora per il SO si blocca tutto il programma perchè non sa come interpretare certi segnali.
- Deve inoltre sapere **a chi appartengono i thread** perchè se si blocca un thread, il SO, se conosce senza i thread senza i proprietari, può decidere di sostituire con un qualunque altro thread e questo vuol dire sostituire programmi fra loro. Quindi è costretto ad attivare i thread di un altro programma
- **Se invece il SO conosce i thread** e i relativi proprietari, in caso di blocco, può decidere di riassegnare i tempi ai thread dello stesso programma per sfruttare al massimo i vantaggi dei thread.

> *Avere **più processori** non vuol dire migliorare le prestazioni ma **gestire in maniera corretta** vuol dire ottenere prestazioni migliori. Quindi se ho più processori devo avere anche una buona **organizzazione** e **bilanciamento** di essi per ottenere risultati migliori*

Memoria



Avere diversi livelli di memoria consente di avere una memoria in grado di rispondere tempestivamente alle richieste. Ognuna di esse ha un proprio compito.

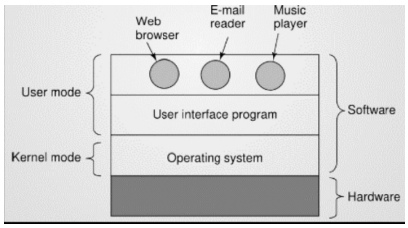
Chi non sa della gerarchia è convinto di avere una sola memoria che risponde in maniera super veloce. Si fanno meno operazioni sull'harddisk e di più sulla RAM.

- Salendo di gerarchia si hanno memorie meno capienti e più veloci.
- Nella cache si memorizza tutto ciò che mi serve **di recente** perchè tutto quello che è usato di recente probabilmente verrà usato di nuovo. In questo modo si perde meno tempo e si velocizzano le operazioni
- Le **cache L1** sono dentro la CPU e sono velocissime.

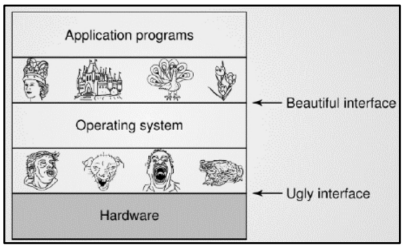
- La **cache L2** può essere di 2 tipi: unica per tutti i componenti core, oppure ognuna ha la sua componente core.

In caso di multiprocessore ogni processore ha la sua cache. All'interno di essa vengono memorizzate le informazioni su un determinato programma che viene eseguito.
Visto che i processori devono avere un **bilanciamento**, essi devono **sfruttare la cache** del processore perchè quando termina l'esecuzione di un programma, viene occupata da altri processi.

- Si deve cercare di **rieseguire i programmi in quei processi dove sono stati eseguiti precedentemente** cosi si **sfruttano le cache** che gia **contengono i dati** usati da quel programma precedentemente se non sono stati sovvrascritti.
- Le memorie a **disco fisso** sono **più lente** perchè per raggiungere una posizione si deve partire dalla testa. Qui i dati vengono memorizzati in maniera permanente e servono per l'avvio della macchina
- Le memorie elettroniche (**RAM**) sono **più veloci** perchè sono ad accesso casuale quindi si può accedere a una **qualsiasi posizione** di memoria in ogni istante di tempo. Queste memorie sono volatili (quando si spegne il pc si perdono tutti i dati).te dell'utente.
  Ci sono 2 modalità d'uso:
- **Kernel mode**, ha accesso a tutto ciò che esiste, anche a risorse delicate ed è eseguito dal SO.
- **User mode**, non si ha accesso alle parti più delicate, profonde, della memoria perchè si potrebbe compromettere l'intero funzionamento del sistema.



- Un SO è fortemente legato all'architettura sulla quale opera. Esso usa il concetto di **astrazione** attraverso il quale non ci si deve preoccupare di quello che c'è "sotto" (*uso di un file*)
- Il compito principale del SO è rendere bello ciò che bello non è attraverso l'astrazione, quindi **nascondere la complessità dell'hardware**.



- Si deve avere una visione tale da poter allocare in **maniera ordinata** e **controllata** per sapere in maniera **veloce** ed efficiente come mettere in comunicazione queste componenti. Devono anche gestire l'esecuzione contemporanea di molteplici programmi (*multitask*).

Sistema operativo come gestore di risorse

Per quanto riguarda la condivisione delle risorse si parla di **MULTIPLEX**. Può essere in due modalità:

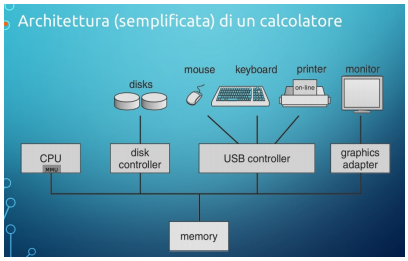
1. rispetto al **TEMPO**: i programmi si alternano fra loro in modo da concedersi la risorsa
2. rispetto allo **SPAZIO**: ogni processo prende una porzione di risorsa in termini di spazio (utenti che si suddividono la stessa memoria).

Evoluzione dei SO

Vi furono diverse *generazioni*:

1. `1945-55` : Schede a perforazione. linguaggio macchina. nessun SO e si facevano semplici calcoli matematici
2. `1955-65` : transistor, **mainframe**. I pc diventano più affidabili. Erano grandi ed erano chiusi in stanze dedicate e gestite da operatori professionisti. Acquisiti solo da industrie o università (*erano costosi*). Ogni programma era detto **JOB** in *assembly*. Un processo veniva svolto uno dopo l'altro seguendo la logica di una coda e quest'operazione prendeva il nome di **Batch**. Si facevano equazioni differenziali. Si raccoglievano tutti i job.
    - In questa era ci sono due diversi calcolatori che lavorano insieme in *stanze diverse*.
3. `1965-80` : si cerca di **incorporare** le due attività all'interno di un solo calcolatore piccolo, accessibile. Si vogliono inglobare le attività. Si usano i circuiti integrati e nasce l'**unica** macchina `IBM SYSTEM/360` di piccole dimensioni rispetto a prima che **ingloba le due attività**.
   In questa *terza generazione* ci sono diverse tecniche:
   - gestione della **MULTIPROGRAMMAZIONE**: esecuzione di processi contemporaneamente. Quindi si cerca di occupare tutta la memoria per migliorare le **prestazioni**.
   - **Timesharing**: necessità di far sì che più utenti lavorino allo stesso tempo e la prima macchina di questo tipo fu **MULTICS** ma ebbe poco successo. Da essa si iniziò lo sviluppo di **UNIX**.
   I sistemi erano **COMPLESSI** e scritti in **linguaggio macchina** (**assembly**) quindi commettere errori era molto semplice.
4. `1980-oggi` : Nasce il primo SO **MS-DOS** puramente da usato da **Shell**. *Windows 3.1* fu la prima versione **PARZIALMENTE GRAFICA**. Windows 95 era il primo sistema operativo **INTERAMENTE GRAFICO**.
5. `1990-oggi` : *Mobile computers*, *iOS*, *Android*, *Windows Mobile*

Hardware



L'hardware è collegato da vari **bus** e ce ne sono diversi tipi in base al dato che viaggia su una determinata linea: bus dati ecc...
Come vengono messi in comunicazione i vari componenti dal SO? *Qual è la complessità di tali operazioni*? Viene imposto un **vincolo**: le operazioni devono **ridurre il più possibile i costi** in termini di **TEMPO**.

- Aprire un file in lettura è più "leggera" come operazione rispetto alla scrittura.

Processore

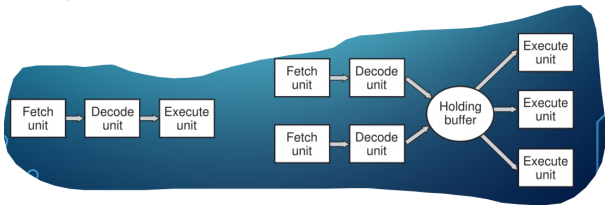
Ciclicamente preleva le istruzioni (**fetch**), le **decodifica** ed esegue le **operazioni**. E' formato da tanti registri utili per memorizzare le variabili più importanti, risultati temporanei (tipo risultati da operazioni aritmetiche):

- Program Counter (PC): contiene memorizzata l'istruzione successiva da eseguire. La fetch preleva l'istruzione da questo registro

- Stack Pointer (SP): punta alla cima di uno stack usato per memorizzare i frame per ogni procedura che è stata eseguita ma non ancora completata (variabili locali, file aperti letti e altro ancora...).
  - *Quando si blocca un programma, il SO deve tenere in mente **tutto quello che ha fatto** fino a quel **momento** e dove si è fermato in modo tale da **riprendere** dal punto preciso.*
- Program Status Word (PSW): registro a bit che contiene la priorità della CPU, la modalità di esecuzione di un programma e altro..

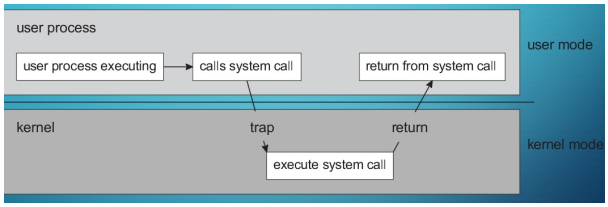
Esistono diverse progettazioni avanzate:

- **PIPELINE**, vuol dire che si eseguono **più istruzioni nello stesso tempo**:
  - *Fetch->Decode->Execute*
- **CPU SUPERSCALARE**: si ha un'unità specializzata a vari tipi di operazioni: booleane, float, interi ecc... Ciò dipende dall'operazione che si deve fare, appunto, ed è più efficiente. Dal buffer (che fa da coda) verranno prelevate le varie operazioni da eseguire.



Chiamata di sistema

Quando si ha una **chiamata di sistema**, un programma utente va in esecuzione e, quando si deve accede a qualcosa di delicato (non concessa all'utente), si attiva una chiamata di sistema attraverso un'istruzione **TRAP** si avviene il passaggio del comando da modalità utente a modalità kernel e chiaramente ogni TRAP ha un proprio costo.
Successivamente vengono eseguite le operazioni (tipo disabilitazione degli interrupt) e dopodiché si ritorna alla modalità utente.



- Il sistema operativo, in modalità kernel, sa esattamente come gestire le situazioni più delicate a differenza dell'utente.

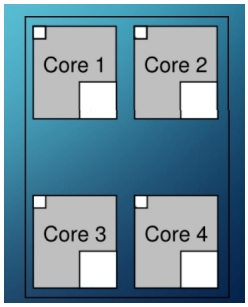
  > *E' un grosso rischio assegnare all'utente di gestre la memoria.*

- Più si limitano le operazioni che può fare l'utente, più trap devo usare e quindi più spazio di memoria devo occupare quindi non conviene esagerare.
- Spesso le TRAP vengono causate dall'HardDisk (tipo la divisione per 0)

Più processori

Si possono avere **multiprocessori multicore** oppure si possono avere (*in entrambi i casi*) si possono avere sistemi **multithread**, cioè è come se un programma che viene eseguito, viene diviso in varie parti diverse fra loro. Parti diverse (thread) vengono eseguite come se venissero eseguite insieme(l'esecuzione avviene sempre nel tempo assegnato al programma in sè) in uno **pseudo-parallelismo**.
Ogni parte prende il nome di **THREAD** e **ognuno è differente dall'altro** in modo tale che ogni parte

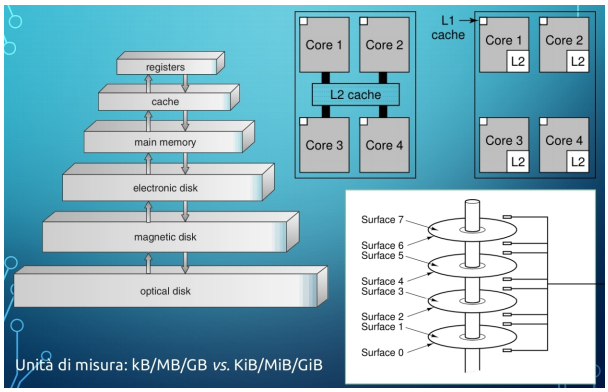
abbia il **proprio compito**.



- Se il SO vede un programma **senza thread** e un pezzo di programma richiede qualcosa che la manda in attesa, esso manda l'intero programma in attesa.
- Se invece ci sono i thread (che non interagiscono fra di loro) e solo un thread blocca l'esecuzione, il SO decide di far andare avanti altri thread e bloccando solo un pezzo fino alla sua liberazione. Nello stesso tempo si può eseguire parti di programma e far andare avanti le varie operazioni.
- Il **SO deve tenere conto dei thread** in modo da poterli sfruttare al massimo. Se non li conosce e si blocca un terzo flusso(thread) allora per il SO si blocca tutto il programma perchè non sa come interpretare certi segnali.
- Deve inoltre sapere **a chi appartengono i thread** perchè se si blocca un thread, il SO, se conosce senza i thread senza i proprietari, può decidere di sostituire con un qualunque altro thread e questo vuol dire sostituire programmi fra loro. Quindi è costretto ad attivare i thread di un altro programma
- **Se invece il SO conosce i thread** e i relativi proprietari, in caso di blocco, può decidere di riassegnare i tempi ai thread dello stesso programma per sfruttare al massimo i vantaggi dei thread.

> *Avere **più processori** non vuol dire migliorare le prestazioni ma **gestire in maniera corretta** vuol dire ottenere prestazioni migliori. Quindi se ho più processori devo avere anche una buona **organizzazione** e **bilanciamento** di essi per ottenere risultati migliori*

Memoria



Avere diversi livelli di memoria consente di avere una memoria in grado di rispondere tempestivamente alle richieste. Ognuna di esse ha un proprio compito.

Chi non sa della gerarchia è convinto di avere una sola memoria che risponde in maniera super veloce. Si fanno meno operazioni sull'harddisk e di più sulla RAM.

- Salendo di gerarchia si hanno memorie meno capienti e più veloci.
- Nella cache si memorizza tutto ciò che mi serve **di recente** perchè tutto quello che è usato di recente probabilmente verrà usato di nuovo. In questo modo si perde meno tempo e si velocizzano le operazioni
- Le **cache L1** sono dentro la CPU e sono velocissime.

- La **cache L2** può essere di 2 tipi: unica per tutti i componenti core, oppure ognuna ha la sua componente core.

In caso di multiprocessore ogni processore ha la sua cache. All'interno di essa vengono memorizzate le informazioni su un determinato programma che viene eseguito.
Visto che i processori devono avere un **bilanciamento**, essi devono **sfruttare la cache** del processore perchè quando termina l'esecuzione di un programma, viene occupata da altri processi.

- Si deve cercare di **rieseguire i programmi in quei processi dove sono stati eseguiti precedentemente** cosi si **sfruttano le cache** che gia **contengono i dati** usati da quel programma precedentemente se non sono stati sovvrascritti.
- Le memorie a **disco fisso** sono **più lente** perchè per raggiungere una posizione si deve partire dalla testa. Qui i dati vengono memorizzati in maniera permanente e servono per l'avvio della macchina
- Le memorie elettroniche (**RAM**) sono **più veloci** perchè sono ad accesso casuale quindi si può accedere a una **qualsiasi posizione** di memoria in ogni istante di tempo. Queste memorie sono volatili (quando si spegne il pc si perdono tutti i dati).