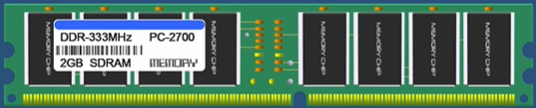
**MEMORIA RAM (RANDOM ACCESS MEMORY)**

La memoria RAM è la **memoria principale** **(MM)** del PC, sul quale la CPU va a cercare la maggior parte dei dati o programmi che gli servono in ogni istante.

È **volatile**, cioè perde il contenuto informativo se non c’è alimentazione.

La RAM si divide in:

* **Statica SRAM**
* **Dinamica DRAM**

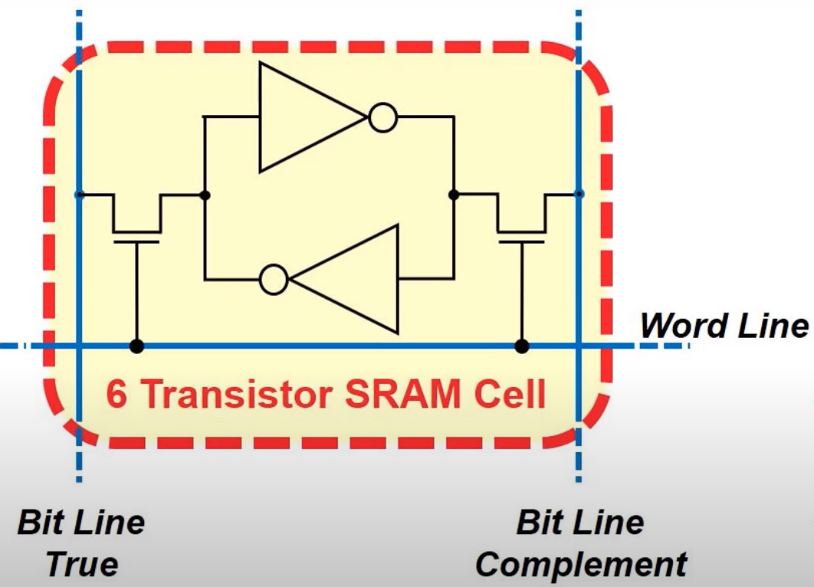


Il numero dopo DDR indica la frequenza di operatività, quello dopo PC indica il larghezza di Banda = bitrate

**SRAM**

Una cella di memoria è composta da 2 transistor e 1 LATCH/FLIP-FLOP (4 transistor).

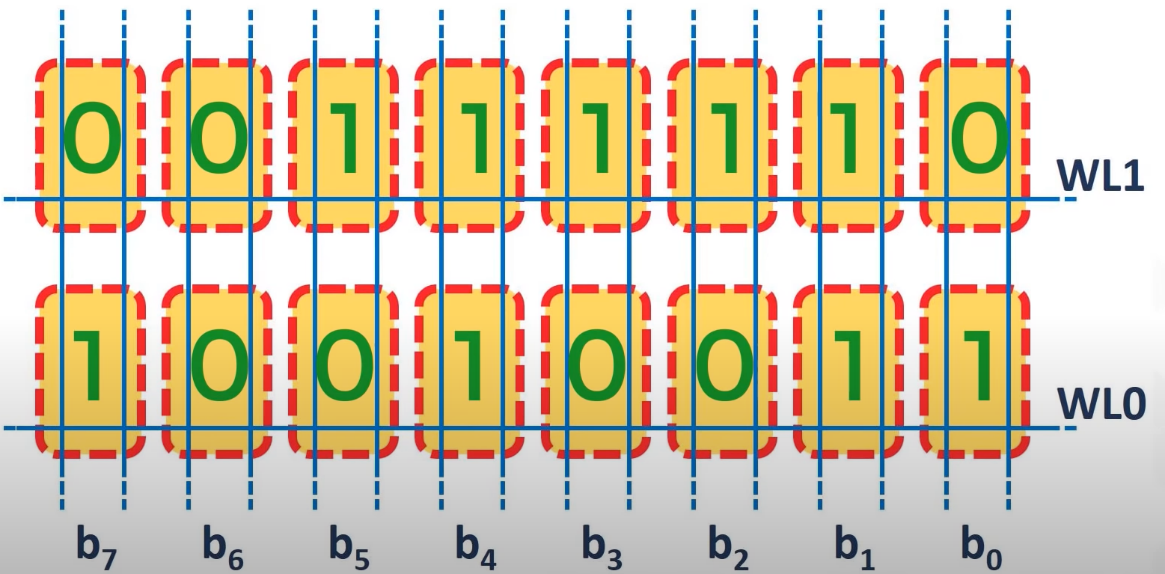
Ogni cella memorizza 1 BIT.

È molto dispendioso dal punto di vista economico.

**WORD LINE:** si attiva quando viene decodificato l’indirizzo di memoria, corrispondente a una locazione.

**BIT LINE**: contiene il contenuto informativo da LEGGERE o SCRIVERE in memoria.

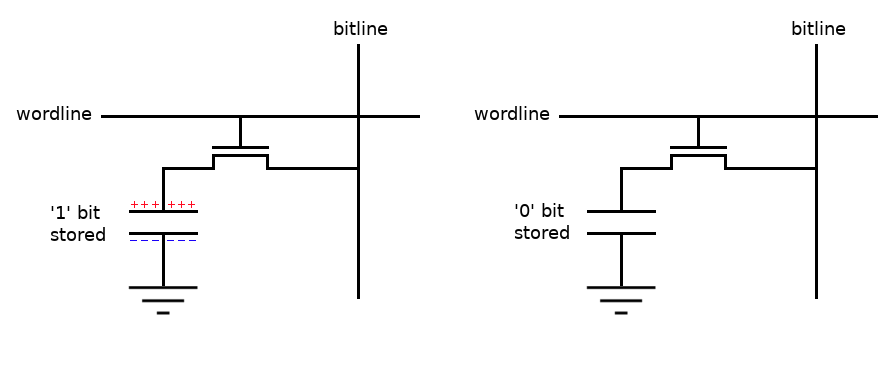
Quando la WL = 1 i transistor si chiudono.

* **Lettura:** la BL riceve il valore contenuto nella cella
* **Scrittura:** la BL invia il valore nella cella

**Le operazioni di lettura e scrittura possono avvenire all’infinito senza causare danni ai circuiti interni. Tempo Lettura = Tempo Scrittura**

* **Synchronous RAM:** architettura basata sulle pipeline, cioè in parallelo.
  + **BUS** grandi con **trasferimenti** fino a **Gb/s**
  + Molto **costose** per essere sviluppate a causa del **grande** **numero** di **componenti** richiesto (6 transistor = 1 bit)
* **Asynchronus RAM:** 
  + **velocità** di trasferimento **modesta**
  + **Risparmio energetico**

**DRAM**



Come nelle SRAM sono presenti la WORD LINE e la BIT LINE, e hanno la stessa funzione.

In questo caso **1 bit** viene **salvato** all’interno di un **condensatore**, che se è **carico = 1**, **scarico = 0**.

Un condensatore però non mantiene la sua carica all’infinito, ma si scarica in un tempo di 5 = 5 \* (R\*C).

Esso può scaricarsi sulla BL in caso di LETTURA, oppure sul transistor quando esso è aperto, quindi la WL = 0. Ciò vuol dire che non è stata selezionata la locazione.

Se si scarica si perde il contenuto informatico, perciò è necessaria una operazione di REFRESH circa 16 volte al secondo = 16Hz -> 1 / 16 = 64ms

**VANTAGGI E SVANTAGGI DRAM**

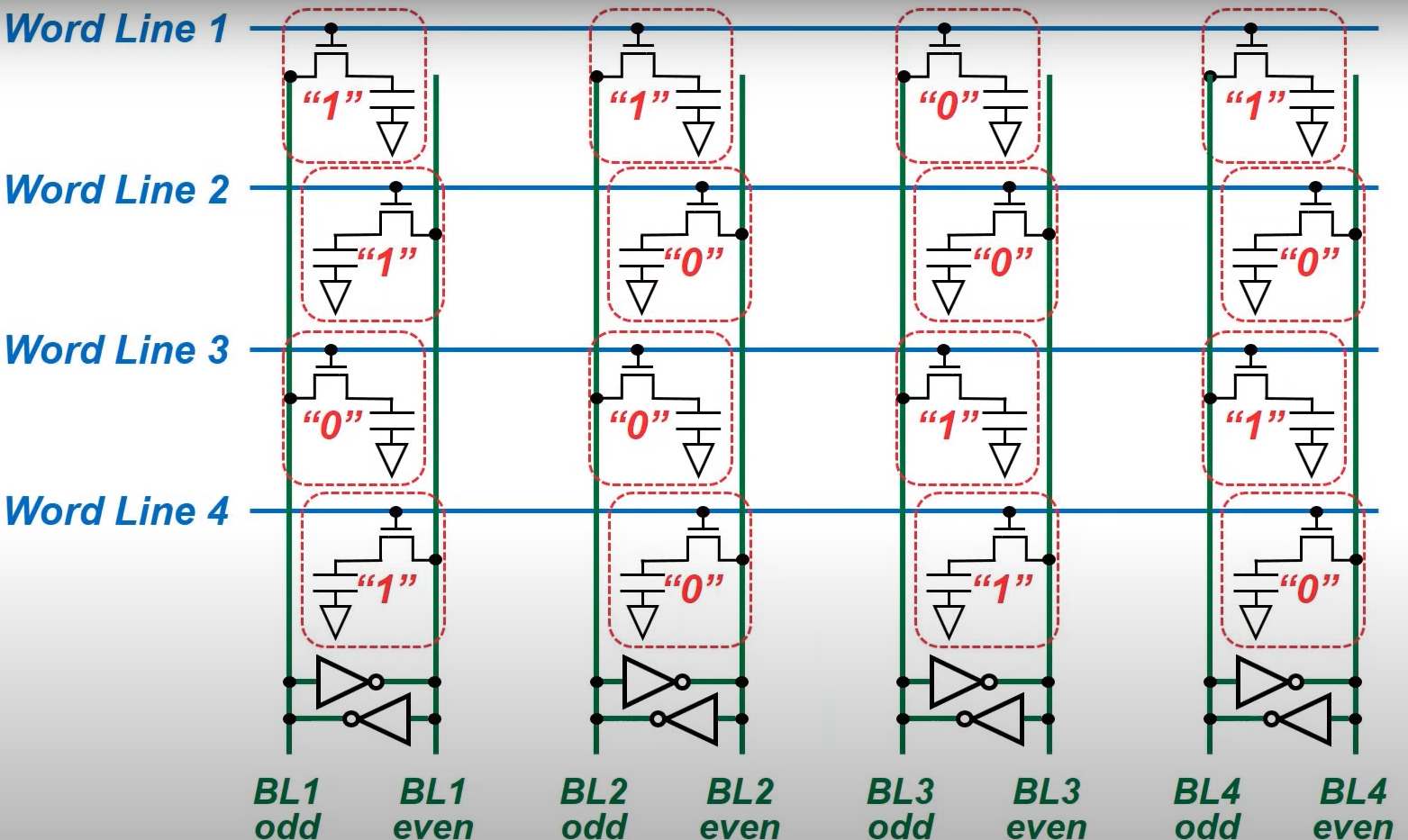
* **Più** **lente** delle **SRAM**
* **Più** **economiche**, richiedono **meno** **componenti**: 1 transistor e 1 Condensatore
* **Complessità** della **conservazione** dello stato: **operazione** di **REFRESH** **onerosa**

**TIPOLOGIE DI DRAM**

* **DDR = Double Data Rate:** i dati vengono trasmessi anche quando CLK è basso

Immagine che contiene testo, orologio

Descrizione generata automaticamente

REFRESH

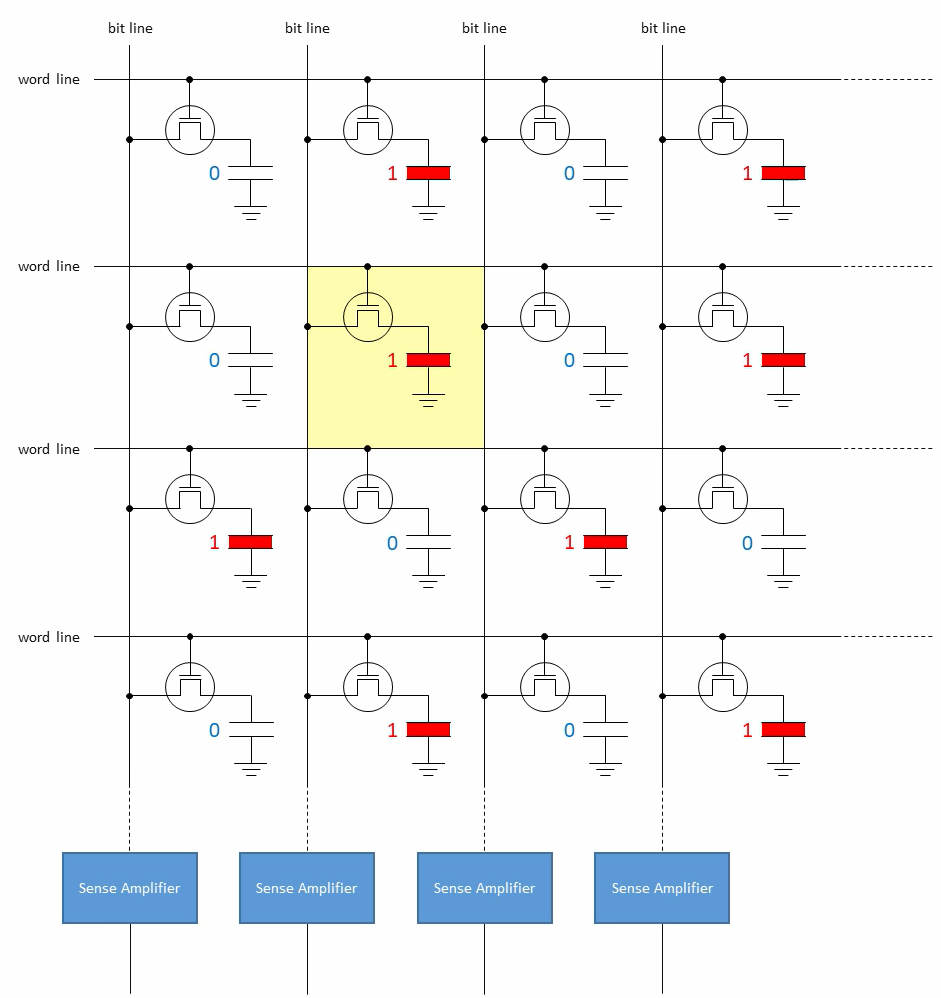
Ci sono **2 WORD LINE**, una per le **locazioni** **DISPARI** e una per quelle **PARI**.

Il circuito sotto è un **Inverter** **LOOP**.

Durante l’operazione di **REFRESH** i due **inverter** sono a una **tensione** **intermedia** **tra** i valori **0 e 1**, circa 0.5V. Se **BLodd = BLeven** si è in una **condizione** di **equilibrio**.

* Se **C = 1**, la sua tensione sarà maggiore di 0.5V, perciò **comincerà** ad **aumentare** quella sulla **BITLINE**. Il **LOOP** **incrementa** sempre di più **la tensione** fino ad arrivare a una condizione in cui una BL vale 1V e l’altra 0V.  
  Quando la **tensione** sulla **BL = 1V** **ricaricherà** il **C** che nel mentre si stava scaricando, **ripristinando il valore logico 1**, presente in precedenza.
* Se **C = 0**, la tensione sulla **BL = 0V**, perciò la tensione viene **subito ripristinata**.

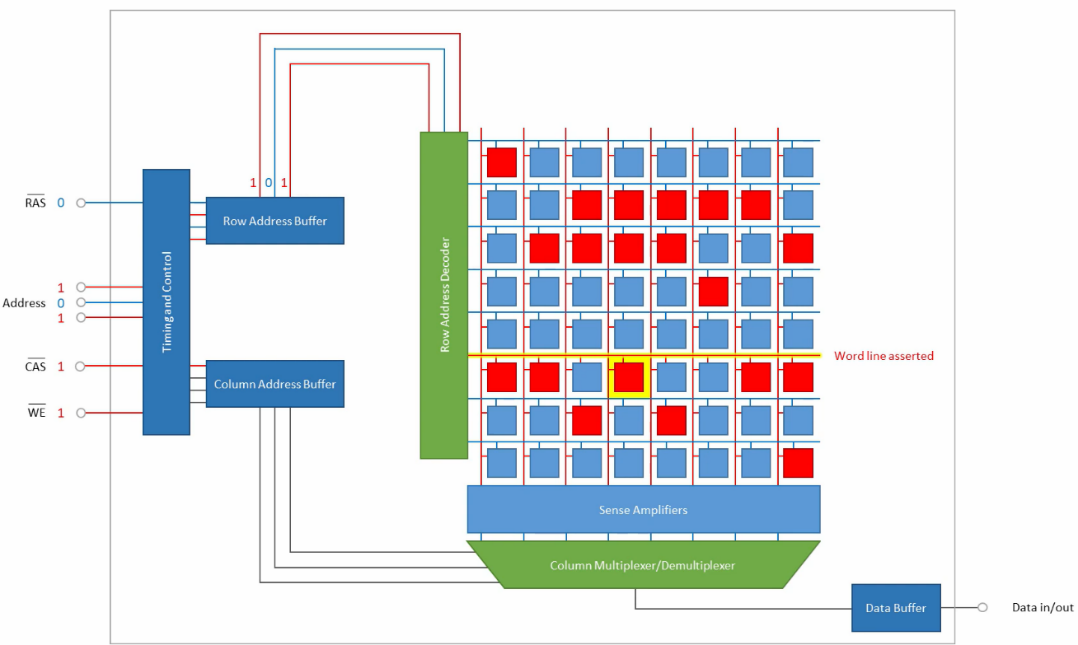
**ACCESSO IN MEMORIA**



I **SENSE AMPLIFIER** sono dei **circuiti logici** **contenenti** dei **LATCH** che **memorizzano** **temporaneamente** il **dato** presente nella cella.

**Dopo ogni** operazione di **LETTURA** il **dato** viene **perso**, perciò è **necessaria** una operazione di **REFRESH** immediata.

**LETTURA E SCRITTURA IN DRAM**



not(RAS) = flag che indica se è il momento di effettuare la fase RAS. Attivo BASSO

not(CAS) = flag che indica se è il momento di effettuare la fase CAS. Attivo BASSO

not(WE) = flag che permette o meno la scrittura in memoria. Attivo BASSO

ADDRESS = numero di bit che servono a codificare un indirizzo di memoria

**N bit -> 2^N allocazioni possibili**

La fase di accesso in memoria si divide in 2 fasi principali:

* **RAS (Raw Address Strobe)** : individua la RIGA (WORDLINE) da attivare
* **CAS (Coloumn Address Strobe)** : individua la cella nella Locazione

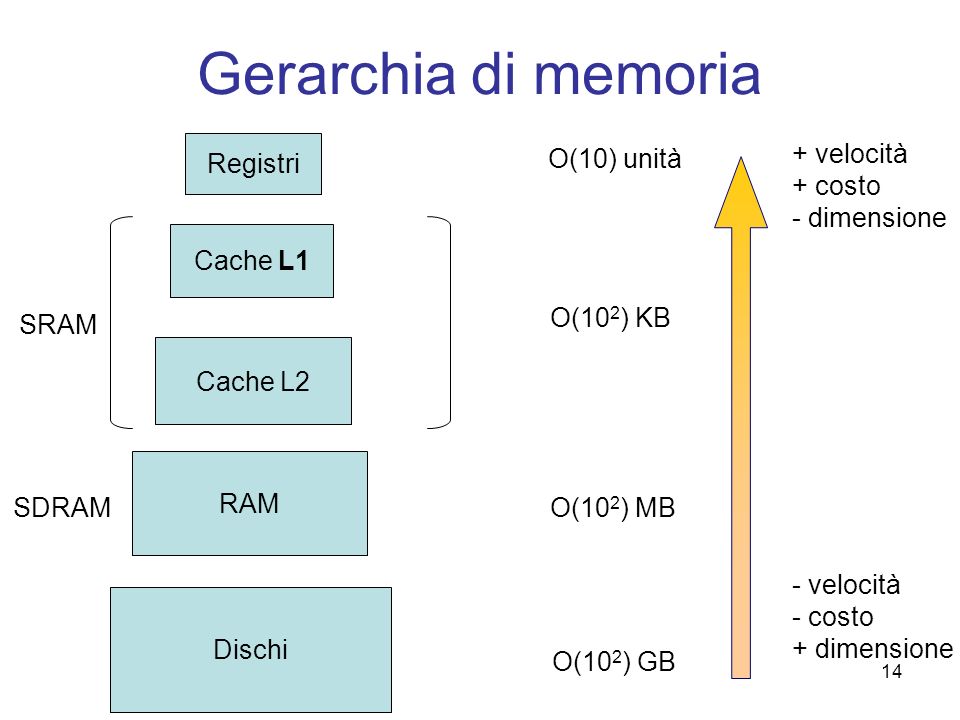
Il modulo di controllo determina le fasi dal punto di vista tempistico e logico, in base ai flag attivi.

* **Indirizzi** entrano nel Modulo. Quando **RAS è attivo** **parte** la **fase di RAS**
* Nel **Raw Address Decoder** vengono **inseriti n bit** per la **codifica dell’indirizzo**.
  + **DECODER** = **n ingressi, 2^n uscite** -> **attiva l’uscita** corrispondente alla **codifica in binario degli ingressi**
  + Viene **attivata la RIGA** (WORDLINE) corrispondente alla codifica
  + **Tutta la LOCAZIONE** viene **salvata** all’interno dei **SENSE AMPLIFIER**
  + La **LOCAZIONE** viene **inserita** nel **Coloumn Address DE/Multiplexer**
* **LETTURA**:
  + **CAM**: viene **mandato** in **uscita** l’**ingresso** **corrispondente** alla **codifica** in binario **degli** ingressi **selettori** (n bit degli indirizzi)
* **SCRITTURA**
  + **CAD**: viene **mandato** in **una** delle 2^n **uscite**, **selezionata** dai **selettori**, il **segnale** di **ingresso**.

**MEMORIE PERMANENTI**

Sono memorie non volatili, che mantengono i dati anche in caso di mancanza di alimentazione.

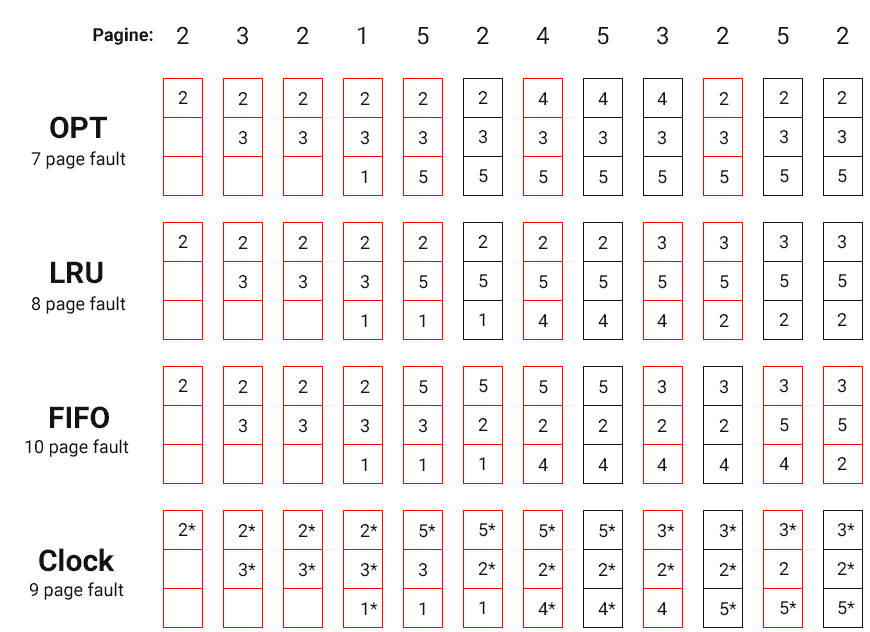
* **ROM** **(Read Only Memory)**: sono scritte nel momento in cui sono fabbricate e non possono essere modificate. Solitamente le ROM contengono le informazioni utili durante il BOOT della macchina.
  + **PROM (Programmable ROM)**: programmabili 1 volta
  + **EPROM (Eresable PROM)**: possono essere cancellate e riscritte
  + **EEPROM (Electrically EPROM)**: cancellabili elettricamente
  + **FLASH**
* **DISCHI RIGIDI**: usati per le memorie di massa ad alta capacità
  + **HDD Hard Disk Drive**: Alta capacità ma bassa velocità
  + **SSD Solid State Drive**: minore capacità ma alta velocità



Nei livelli più alti ci si avvicina alla CPU.

La CPU contatta prima i registri interni per cercare il dato occorrente.

In caso di mancanza del dato comincia a scendere nella piramide.

**POLITICA DI RIMPIAZZO**

[**info**](https://informaticabrutta.it/memoria-virtuale/)

[**info2**](http://wpage.unina.it/fasolino/so/materiale/9-MemoriaVirtuale.pdf)

* **LRU (Least Recently Used) :** comunemente usato per la SET ASSOCIATIVE
  + A ogni SET è associato un contatore di età, che ha un valore massimo di N (numero totale di SET)
  + A ogni accesso a un SET il contatore viene resettato a 0
  + Verrà aggiornata quella che ha il contatore maggiore
* **FIFO (First In First Out)**
  + Usato nella cache set associative
  + Puntatore circolare che trova la pagina che è stata più a lungo in MM e la rimuove, aggiornandola con la nuova
  + Viene resettato il contatore della linea appena aggiornata
  + Gli altri contatori vengono incrementati di 1

**POLITICA DI SCRITTURA IN MEMORIA**

i dati presenti in cache devono essere aggiornati in memoria principale.

* **Write through :** il dato viene scritto prima in cache e poi in memoria, oppure contemporaneamente
* **Write back:** il dato viene riscritto solamente in cache. Vengono modificati i flag di stato D (Dirty) e M (Modified)

è possibile che si verifichi un MISS in scrittura.

La scrittura in cache ha senso solo se l’indirizzo di memoria associato era già presente in essa, altrimenti si genera solo traffico inutile.

* **Write allocate:** viene trovata la locazione di MM interessata, poi viene trasportata in cache dove viene scritto il dato
* **Write no allocate:** il dato viene salvato direttamente in MM, saltando la cache

**MEMORIA PRINCIPALE**

è suddivisa in 2 regioni

* SISTEMA OPERATIVO: ha uno spazio in RAM riservato a se per le varie procedure di sistema, come gli interrupt
* PROCESSI: virtualmente illimitata

Teoricamente un **processo** dovrebbe avere una **porzione** di **MM** **riservata** ad esso per **tutta** la sua **durata** dall’esecuzione.

Il **numero** delle **partizioni** generate **indica** il **grado** di **multiprogrammazione**.

Quando un **processo** **termina** il suo **spazio** viene **liberato**, e può essere destinato a un **altro** **processo**.

È possibile che lo **spazio** **non** sia **sufficiente** perciò rimane un “**buco**” inutilizzato.

Oppure può arrivare un processo che ha una **dimensione maggiore** del **massimo** buco **disponibile**. In questo caso si deve **riordinare**.

Il **compattamento** della MM **richiede rilocazione di processi**, **dati** e **istruzioni** == **sistema** **bloccato** fino al completamento

Per risolvere questo problema si attua una **memoria** **suddivisa** in **pagine**, tutte di **dimensione** **costante** e determinata.

Vien da sé che un **processo** non **avrà** una **memoria** contigua, ma potrà essere **frazionata** **nelle** **pagine**.

Alla creazione di un processo:

* viene creato un sottoinsieme di pagine dedicate ad esso
* non viene creata nessuna pagina, ma si attua la politica **demand paging**

**PROBLEMA**

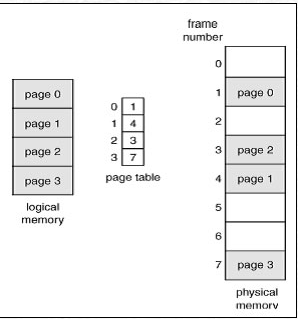
Il sistema di paginazione non presenta una suddivisione logica tra i dati richiesti dai processi

* area riservata per le chiamate di sistema e per il OS
* area destinata ai programmi utente
* area per le procedure utente
* suddivisioni ulteriori della memoria richieste da programmi

Di conseguenza non fornisce alcun sistema di protezione per l’accesso in MM da parte di processi non autorizzati

**MEMORIA VIRTUALE PAGINATA**

* + le pagine possiedono una dimensione fissa
  + la dimensione è una potenza del 2
  + può essere presente sia in MM e di massa
  + crea una suddivisione della memoria dei processi, rendendoli non più contigui ma suddivisi in pagine



**PAGE TABLE** associa a ogni numero di pagina:

* bit per segnalare la presenza in MM. 1 = MM ; 0 = MASSA
* indirizzo base: indirizzo fisico di inizio della pagina di MM
* bit di protezione per evitare l’accesso di processi a zone di memoria non destinate ad esso

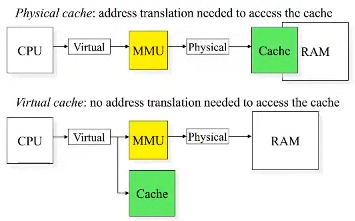
**ACCESSO**

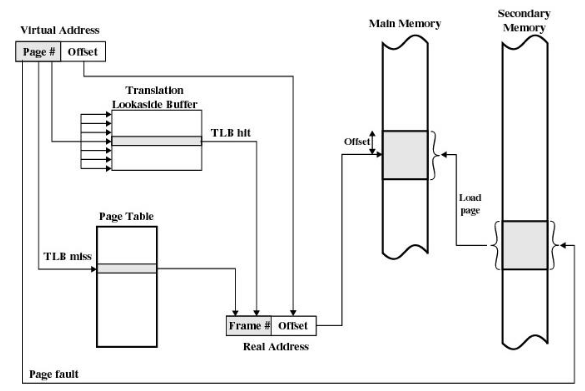
**MMU (Memory Management Unit):** dispositivo posto all’interno della CPU responsabile dell’interfacciamento tra CPU e MM

* Deve controllare nella tabella delle pagine, che risiede anch’essa in MM.
* **Traduzione indirizzi LOGICI → FISICI**
* **Controllo permessi di accesso da parte dei processi**
  + **superuser (ADMIN / ROOT)**
  + **user (nomeUtente)**

Per evitare che un singolo accesso in MM si trasformi in 3, viene implementato un **TLB = Cache della PAGE TABLE**

**TLB (Traslation Lookaside Buffer)**

****

****

Sfrutta al massimo il principio di località spaziale, perché associando avendo delle pagine grandi è sufficiente accedere a una sola per trovare i dati.

La cache può essere

* **fisica**, quindi una volta effettuata la conversione dal MMU viene cercata la pagina in RAM
* **logica:** prima di effettuare la traduzione viene interrogata la Cache per capire quale PT cercare per ottenere il RPN
  + questo caso è più efficiente perché la ricerca del RPN viene svolta parallelamente sia dalla Cache che dal MMU, e il primo che ottiene il RPN lo fornisce

Per capire in quale pagine è presente un indirizzo si segue la seguente formula, che è in comune con l’accesso in cache.

**N° pagina = indirizzo logico / dimensione pagina**

Una volta trovata la pagina si effettua un controllo se la pagina è presente in MM. Se non è presente sia genera un **PAGE FAULT.**

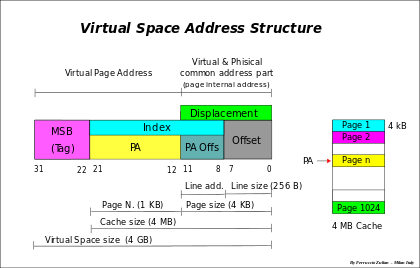
Viene cercata nella memoria di massa, e poi caricata in MM.

Viene calcolato l’indirizzo fisico della MM su cui cercare il dato.

Se la pagina è presente in MM, viene calcolato il n° della pagina, e il numero viene convertito nell’indirizzo base. Per trovare la giusta locazione nella pagina, viene calcolato anche il resto della divisione precedente (ind virtuale / dim pagina). Il resto corrisponde all’OFFSET, perciò il discostamento dall’indirizzo base.

* Quando la pagina trovata non è presente in MM si genera un FAULT.
* Bisogna quindi caricarla in MM, ma in uno spazio abbastanza grande da poterla contenere.
* Una volta che la pagina viene caricata con successo in suo bit di validità viene impostato a 1, per segnalarne la presenza in MM.
* Viene aggiornata la page table
* Viene ricercato l’indirizzo di base nella MM, per poter accedere al dato

**INDIRIZZAMENTO VIRTUALE**

****

indirizzi a 32 bit, spazio di indirizzamento massimo 4GB RAM

Cache a 4 MB → linea di cache 256Byte

**INDEX = PA + PAOFFSET = indice per la linea di cache**

**Anche la Cache è paginata**

**PA[10bit] = indirizzo base della pagina cache 2^10 = 1024 pagine**

**PAOFF [4bit] = offset all’interno della pagina.**

**2^4 = 16 linee di cache = dimensione pagina**

**MSB + PA = page number = indirizzo base della pagina Memoria virtuale** Dimensione pagina 4 KB

Viene cercata la corrispondenza all’interno della PAGETABLE, la quale sarà contenuta in parte nel TLB.

Il TLB accede usando MSB + BA alla propria riga per cercare l’indirizzo fisico

**In un sistema MULTIPROCESS avviene un CONTEXT-SWITCH**

**=** cambio di contesto nel quale si sta operando = vengono eseguiti processi di un altro programma rispetto a quello in esecuzione fino ad ora

PROBLEMI:

* diversi INDIRIZZI LOGICI == stesso FISICO
* stesso LOGICO == diversi FISICI

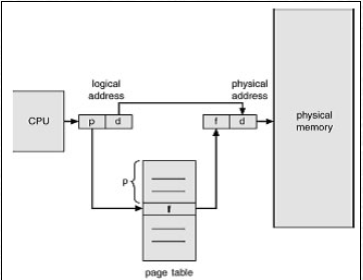
**All’interno di un CONTESTO vi è corrispondenza univoca**

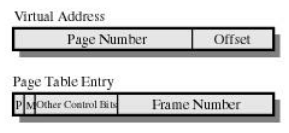
SOLUZIONI

* **FLUSHING** della Cache a ogni switch
* Aggiunta di un ID per distinguere i vari processi
* Aggiungere in cache l’indirizzo fisico ottenuto in precedenza dal TLB durante una ricerca in RAM

**INDIRIZZO FISICO = INDIRIZZO BASE , OFFSET (bit meno significativi dell’indirizzo logico iniziale)**

**INDIRIZZO LOGICO → INDIRIZZO FISICO**





Gli indirizzi vengono classificati in

* **LOGICI**: sono gli indirizzi che un programma possiede e interroga. Vengono opportunamente mappati per essere convertiti in FISICI
* **FISICI**: sono gli indirizzi fisici dell’hardware.

**LOGICO:** generato dal programma in esecuzione (processo)

* **numero di pagina:** viene usato come indice per selezionare la RIGA della PAGE TABLE. Ogni riga della page table associa un **INDIRIZZO BASE**
  + **BASE** punta all’indirizzo fisico di inizio della pagina in cui andare a cercare il dato
  + lunghezza del campo **numero pagina = log2 lunghezza PAGE TABLE → PAGE TABLE = 2^lunghezza**
* **OFFSET:** viene sommato all’indirizzo base per ottenere l’indirizzo FISICO completo. Indica anche la dimensione di ogni pagina.
* Con n bit OFFSET scelgo quale locazione tra le 2^n contenute nella pagina. N = log2 dimensione pagina

**FISICO:** indirizzo della locazione di MM del dato occorrente

**DIMENSIONE DELLA PAGINA**

* GRANDE: per ridurre il numero di accessi in memoria di massa, molto più lenta della MM.

Oppure per sfruttare al meglio il principio di località spaziale.

Page Table compatte

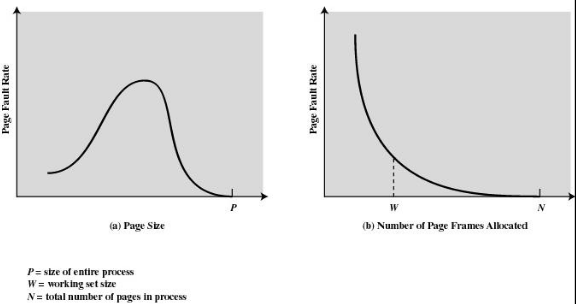
* PICCOLA: sfruttare al meglio l’area di MM dedicata ai page frame

Ridurre il tempo di accesso alla Memoria di massa

PAGE FRAME: area di MM dedicata a contenere una pagina fisica, perciò che contiene l’effettivo indirizzo fisico della MM.

I page frame devono essere condivisi per tutti i processi.

**PAGE FAULT= MISS**

****

Dal punto di vista tempistico un page fault blocca la CPU (TRAP) per 1 milione circa di cicli di CLK.

Per gestire al meglio il rimpiazzo è opportuno attuare delle politiche gestite non più dall’HW come nel caso delle Cache, ma dal SOFTWARE.

Le pagine hanno una struttura completamente associativa, analoga a quella delle cache, e utilizza una politica di rimpiazzo LRU:

le pagine più usate recentemente stanno in testa, e saranno quelle con minore probabilità di essere rimosse

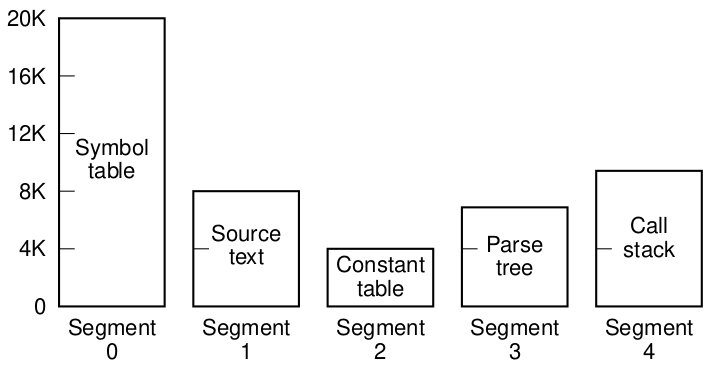
Genera una chiamata a una procedura di sistema

* cerca uno spazio in cui inserire n pagine d
* se non esiste viene sovrascritto usando una politica di rimpiazzo:
  + **LRU**: quello non usato di recente
  + **FIFO**: quello caricato meno recentemente
* Se la pagina è stata modificata, Dirty Bit = 1, prima viene aggiornata quella in memoria di massa
* Una volta aggiornata viene caricata in MM

**WORKING SET** è l’insieme delle pagine in uso corrente dai processi in esecuzione

**MEMORIA PRINCIPALE SEGMENTATA**

**SUDDIVISIONE LOGICA MEMORIA PROCESSI**

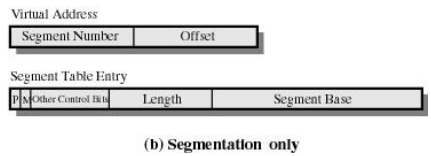


I vari segmenti logici risiedono in zone di memoria distinte

**Segmentata:** organizza la memoria virtuale in segmenti di dimensione diversa

**INDIRIZZO LOGICO = numero segmento , OFFSET**

* **semplice gestione di strutture dati crescenti**
* **semplice condivisione tra processi**
* **protezione accessi con un sistema di privilegi**

****

P = present flag

M = modified flag

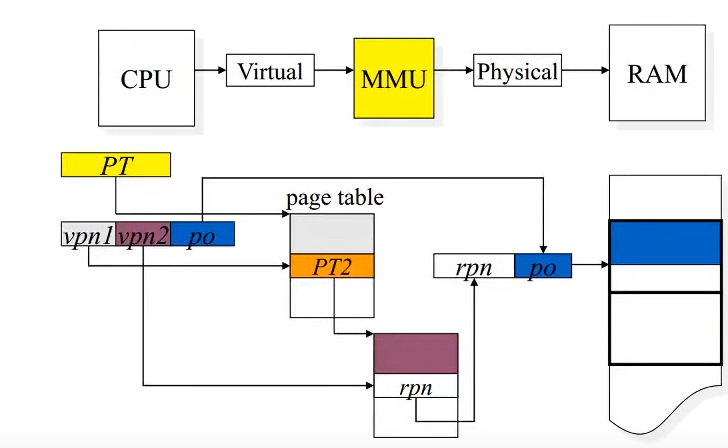
Control bits per privilegi e sicurezza

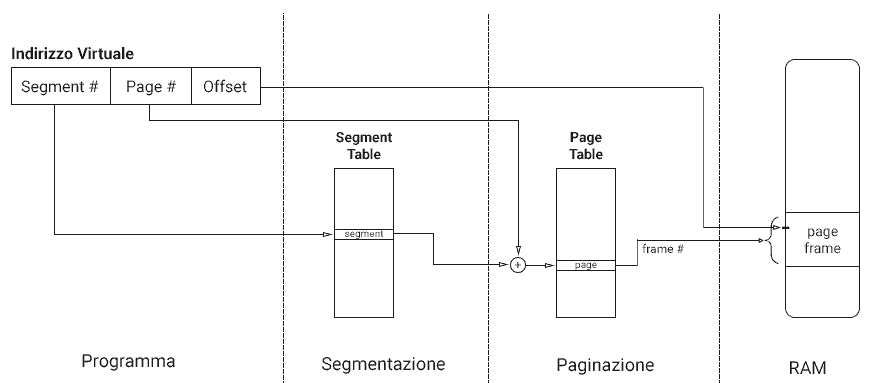
Length = lunghezza del segmento

BASE = indirizzo base del segmento in MM

**Indirizzo FISICO = BASE , OFFSET**

**SEGMENTAZIONE PAGINATA**

****

Lo schema introduce una paginazione a 2 livelli.

La prima SEGMENT TABLE punta a una seconda PAGE TABLE nel quale è contenuto il page number FISICO, a cui aggiungere OFFSET

Questo meccanismo si opera per andare a suddividere la enorme grandezza della PAGE TABLE effettiva necessaria, la quale potrebbe superare anche la dimensione della MM.

La PAGE TABLE LOGICA viene anch’essa suddivisa in PAGINE, indirizzabili dal campo vpn2 nell’indirizzo LOGICO

Quante PT2 ci sono? 2^segment#length = numero di righe della PT1

La ST è necessario che sia in MM.

Le PT invece possono risiedere anche in memoria di massa, perché non tutte saranno interrogate. Solitamente le PT2 hanno la stessa dimensione delle PAGINE FISICHE in MM o massa.

Ogni riga della PT2 rappresenta una pagina diversa in MM