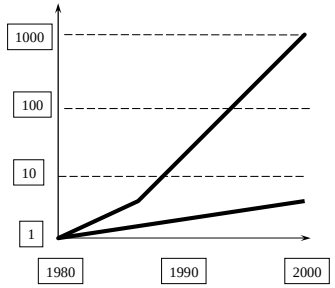
3.2 Gerarchie di memoria
Tecnologie di memoria

L'ideale sarebbe una memoria molto **ampia**, molto **veloce** e molto **economica**

Tecnologia	Tempo di accesso	Dimensioni
registro	1 - 2 ns.	16 - 256 B
cache	3 - 10 ns.	1 - 32 KB
SRAM	20 - 50 ns.	32 - 512 KB
DRAM	60 - 250 ns.	1 - 1024 MB
disco	5 - 20 ms.	1 - 512 GB
CD-ROM	100 - 500 ms.	600 - 1200 MB
nastro	>> 1 s	>> 1 GB

3.2 Gerarchie di memoria
Prestazioni CPU/memoria



- Le CPU hanno avuto un aumento di prestazioni notevole, dovuto ad innovazioni tecnologiche ed **architetturali**
- Le memorie sono migliorate **solo** grazie agli avanzamenti tecnologici

3.2 Gerarchie di memoria
Proprietà dei programmi

- Proprietà **statiche** *(dal file sorgente)*
- Proprietà **dinamiche** *(dall'esecuzione)*
  - **Linearità** dei riferimenti
    - Gli indirizzi acceduti sono spesso consecutivi
  - **Località** dei riferimenti
    - Località **spaziale**
      - *Gli accessi ad indirizzi **contigui** sono **più probabili***
    - Località **temporale**
      - *La zona di accesso **più recente** è quella di permanenza **più probabile***

3.2 Gerarchie di memoria
La congettura 90/10

Un programma impiega mediamente il **90%** del suo tempo di esecuzione alle prese con un numero di istruzioni pari a circa il **10%** di tutte quelle che lo compongono.

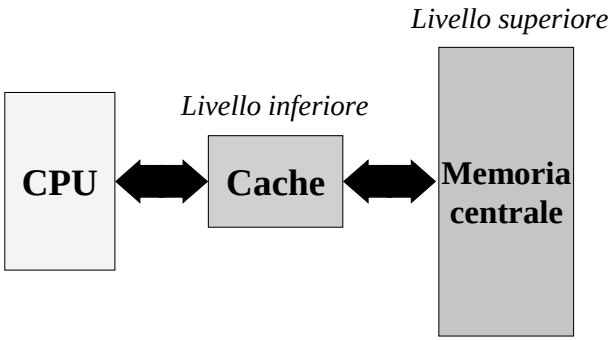
3.2 Gerarchie di memoria
Divide et impera

- Conviene organizzare la memoria su più livelli gerarchici:
  - **Livello 1 (cache)**: molto veloce e molto costosa
    ⇒ dimensioni ridotte, per i dati ad accesso più probabile
  - **Livello 2 (memoria centrale)**: molto ampia e lenta
    ⇐ costo contenuto, per tutti i dati del programma

3.2 Gerarchie di memoria
Organizzazione gerarchica

- Memoria a livelli
  - Al livello più alto (**superiore**) stanno i ‘supporti di memoria’ più capaci, più lenti e meno costosi
  - Ai livelli più bassi (**inferiori**) si pongono supporti più veloci, più costosi e meno capaci
  - La CPU usa direttamente il **livello più basso**
  - Ogni livello superiore deve contenere tutti i dati presenti ai livelli inferiori (ed altri)

3.2 Gerarchie di memoria
Schema concettuale



3.2 Gerarchie di memoria
Suddivisione in blocchi

- Per realizzare un’organizzazione gerarchica conviene suddividere la memoria in **blocchi**
- La **dimensione** di un blocco è la **quantità minima indivisibile** di dati che occorre prelevare (copiare) dal livello superiore
- L’**indirizzo** di un dato diviene l’indirizzo del blocco che lo contiene *sommato* alla posizione del dato all’interno del blocco

3.2 Gerarchie di memoria
Hit e miss

Un dato richiesto dalla CPU può essere presente in cache (**hit**) oppure mancante (**miss**)

- Un **hit**, *successo*, deve essere molto probabile (>90%) se si vuole guadagnare efficienza prestazionale
- Un **miss**, *fallimento*, richiede l'avvio di una procedura di scambio dati (**swap**) con il livello superiore

3.2 Gerarchie di memoria
Tempo medio di accesso

$\mathbf{T_a}$: Tempo medio di accesso ad un dato in memoria

$$\mathbf{T_a = T_h \times P_h + T_m \times (1 - P_h)}$$

$\mathbf{T_h}$ = tempo di accesso ad un dato **presente** in cache

$\mathbf{T_m}$ = tempo medio di accesso ad un dato **non** in cache
(funzione della dimensione del blocco)

$\mathbf{P_h}$ = probabilità di **hit**
(funzione della dimensione del blocco **e** della politica di gestione)

3.2 Gerarchie di memoria
Tecnica generale

- Suddivisione della memoria centrale in blocchi ***logici***
- Dimensionamento della cache in **multiplo** di blocchi
- Per ogni indirizzo emesso dalla CPU
  - **Hit** $\Rightarrow$ Il dato richiesto viene fornito **immediatamente** alla CPU
  - **Miss** $\Rightarrow$ La cache richiede il dato al livello superiore
    Il blocco contenente il dato viene posto in cache
    Il dato richiesto viene fornito alla CPU

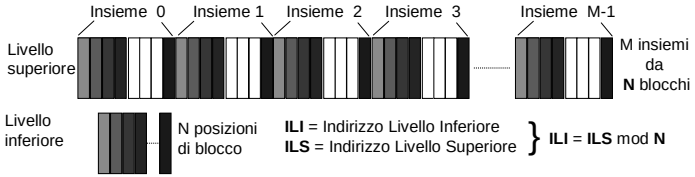
3.2 Gerarchie di memoria
Problematiche

- Organizzazione della cache e tecniche di allocazione
- Individuazione di hit o miss
- Politica di rimpiazzo dei blocchi
- Congruenza dei blocchi

3.2 Gerarchie di memoria
Associazione diretta 1

Tecnica nota come ***direct mapping***

- Ogni blocco del livello superiore può essere allocato *solo* in uno specifica posizione (detta ***line*** o ***slot***) del livello inferiore



3.2 Gerarchie di memoria
Associazione diretta 2

Vantaggi

- Semplicità di traduzione da indirizzo ILS (memoria) ad indirizzo ILI (cache)
- Determinazione veloce di hit o miss

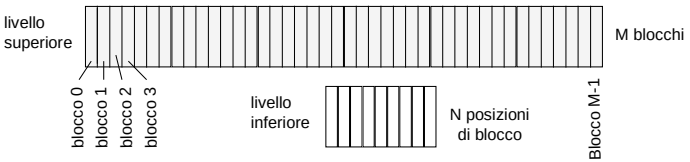
Svantaggi

- Necessità di contraddistinguere il blocco presente in ILI (introduzione di un'etichetta, '**tag**')
- Swap frequenti per accesso a dati di blocchi adiacenti

3.2 Gerarchie di memoria
Associazione completa 1

Tecnica nota come *fully associative*

- Ogni blocco del livello superiore può essere posto in *qualunque* posizione del livello inferiore



3.2 Gerarchie di memoria
Associazione completa 2

Alla cache capace di N blocchi viene associata una tabella di N posizioni, contenenti il numero di blocco effettivo (**tag**) in essa contenuto.

Vantaggi

- Massima efficienza di allocazione

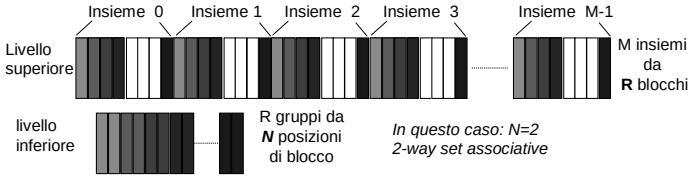
Svantaggi

- Determinazione onerosa della corrispondenza ILI-ILS e della verifica di hit/miss

3.2 Gerarchie di memoria
Associazione a gruppi 1

Tecnica nota come ***N**-way set associative*

- Ogni blocco di un certo insieme di blocchi del livello superiore può essere allocato *liberamente* in uno *specifico* gruppo di blocchi del livello inferiore



3.2 Gerarchie di memoria
Associazione a gruppi 2

- Alla cache, composta da R gruppi di N posizioni di blocco ciascuno, si affiancano R tabelle di N elementi, contenenti le etichette (**tag**) che designano i blocchi effettivi posti nelle posizione corrispondente.
  - **Valutazione**: buona efficienza di allocazione a fronte di una sopportabile complessità di ricerca

3.2 Gerarchie di memoria
Esempio

Associazione diretta (cache ed insiemi di capacita' **n**=8 blocchi)

Blocco 12 Insieme 0 [*insieme unico*]→ Posizione 12 mod **n** = 4

Blocco 7 Insieme 2 → Posizione (2×8+7) mod n = 7

Associazione completa

Blocco 12 Insieme 0 → Posizione *qualunque* tra 0 e 7

Associazione a gruppi di capacita' N (N=2 , **n**/N=**4** gruppi)

Blocco 12 Insieme 0 → Posizione 0 *o* 1 in gruppo 12 mod **4** = 0

Blocco 7 Insieme 2 → Posizione 0 *o* 1 in gruppo (2×8+7) mod 4 = 3

3.2 Gerarchie di memoria
Politiche di rimpiazzo dei blocchi

Quale blocco conviene sostituire in cache per effettuare uno swap? (*Penalità di miss*)

- **Casuale**, per occupazione omogenea dello spazio
- **Least Recently Used** (**LRU**), per preservare ***località temporale***

P(miss)		**rimpiazzo casuale**			**rimpiazzo LRU**		
	N-way	2	4	8	2	4	8
Ampiezza cache	16 KB	5,69	5,29	4,96	5,18	4,67	4,39
	64 KB	2,01	1,66	1,53	1,88	1,54	1,39
	256 KB	1,17	1,13	1,12	1,15	1,13	1,12

3.2 Gerarchie di memoria
Il problema della scrittura 1

La scrittura dei dati determina *incoerenza* tra il blocco in cache e quello nei livelli superiori

- '**Write through**'
  - Scrittura *contemporanea* in cache e nel livello di memoria superiore
  - Aumento di traffico per frequenti scritture nel medesimo blocco, ma i dati sono sempre coerenti tra i livelli
  - Si ricorre a buffer di scrittura *asincroni* (differiti) verso la memoria.

*N.B.: La memoria contiene istruzioni **e** dati, e solo il 50% delle operazioni sui dati sono scritture (circa 12 % del totale)*

3.2 Gerarchie di memoria
Il problema della scrittura 2

- '**Write back**'
  - Scrittura in memoria superiore *differita* al rimpiazzo del blocco di cache corrispondente
  - Occorre ricordare se sono avvenute operazioni di scrittura nel blocco
  - Consente ottimizzazione del traffico tra livelli
  - Causa periodi di incoerenza

3.2 Gerarchie di memoria
Il problema dei 'miss' 1

- Miss di **primo accesso**, *inevitabile* e non riducibile
- Miss per **capacità insufficiente**, quando la cache *non può* contenere tutti i blocchi necessari all'esecuzione del programma
- Miss per **conflitto**, quando *più* blocchi possono essere mappati (con associazione diretta o a gruppi) su *uno* stesso gruppo

3.2 Gerarchie di memoria
Il problema dei 'miss' 2

- Tecniche 'classiche' di soluzione
  - Maggior dimensione di blocco
    - Buona per fruire di *località spaziale*
    - Causa incremento di miss per conflitto (meno blocchi disponibili)
  - Maggiore associatività
    - Causa incremento del tempo di localizzazione in gruppo (hit)
    - Soggetta alla 'regola del 2:1'
      - Una cache ad N blocchi con associazione diretta ha una probabilità di miss pressoché uguale ad un cache di dimensione N/2 con associazione a 2 vie

3.2 Gerarchie di memoria
Il problema dei 'miss' 3

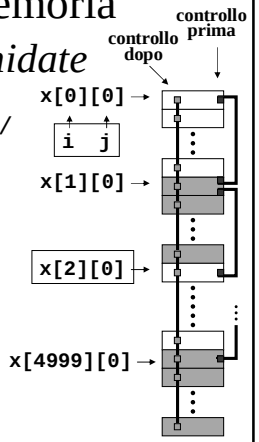
- Altre tecniche
  - Uso di tabelle realizzate con memorie associative
    - Riduce i tempi di ricerca
  - Separazione tra cache *dati* e cache *istruzioni*
  - Ottimizzazione degli accessi mediante compilatori
    - Posizionamento accurato delle procedure ripetitive
    - Fusione di vettori in strutture (località spaziale)
    - Trasformazioni di iterazioni annidate (località spaziale)
    - …

3.2 Gerarchie di memoria
Es.: *Fusione di vettori in strutture*

3.2 Gerarchie di memoria
Es.: *Iterazioni annidate*

```
/* prima della ottimizzazione */
for (j=0;j<100;j=j+1)
  for (i=0;i<5000;i=i+1)
     x[i][j] = 2*x[i][j];

/* dopo l'ottimizzazione */
for (i=0;i<5000;i=i+1)
  for (j=0;j<100;j=j+1)
        x[i][j] = 2*x[i][j];
```



3.2 Gerarchie di memoria
Dimensionamento tipo

Dimensione cache	1 - 256 KByte
Dimensione del blocco	4 - 128 Byte
Durata accesso con hit	1 - 4 cicli clock
Durata accesso con miss	8 - 32 cicli (ed anche più)
Tempo di localizzazione	6-10 cicli
Tempo di trasferimento	2 - 22 cicli
Percentuale miss	1% - 20%

3.2 Gerarchie di memoria
Memorie a componenti dinamici 1

- Denominate **Dynamic RAM**
- Memorizzazione su componenti capacitivi
  - 1 transistor per bit (4-6 per Static RAM), dunque maggiore densità
- Richiede un ciclo di *refresh* ogni 8-10 ms.
  - Non richiesto per Static RAM

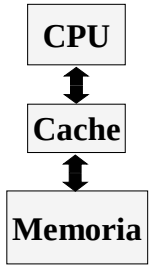
3.2 Gerarchie di memoria
Memorie a componenti dinamici 2

- Organizzazione interna a matrice bidimensionale con indirizzamento in due fasi sullo *stesso* canale di accesso
  - Per riga (*Row Access Strobe*)
  - Per colonna (*Column Access Strobe*)
- Maggiore capacita' ma tempo di accesso maggiore delle Static RAM
  - Dovuto alla diversa modalità di indirizzamento

3.2 Gerarchie di memoria
Memorie a componenti dinamici 2

Anno	Capacità	Ciclo di lettura
1980	64 KB	250 ns.
1983	256 KB	220 ns.
1986	1 MB	190 ns.
1989	4 MB	165 ns.
1992	16 MB	145 ns.
1995	64 MB	120 ns.

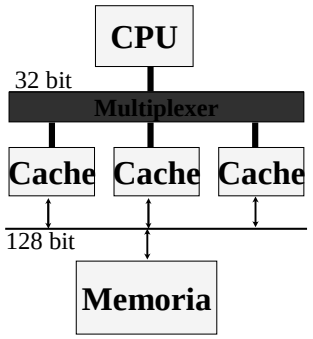
3.2 Gerarchie di memoria
Memoria principale 1



Organizzazione diretta

Il bus tra CPU e cache ha la stessa dimensione di quello tra cache e memoria principale

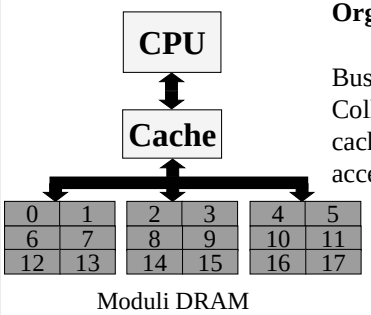
3.2 Gerarchie di memoria
Memoria principale 2



Organizzazione estesa

Un instradatore tra CPU ed N cache.
Bus più ampio tra le cache e la memoria principale

3.2 Gerarchie di memoria
Memoria principale 3

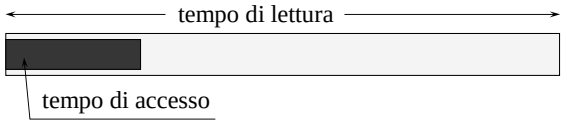


Organizzazione interallacciata

Bus semplice tra CPU e cache.
Collegamento interallacciato tra cache e moduli DRAM con accessi simultanei

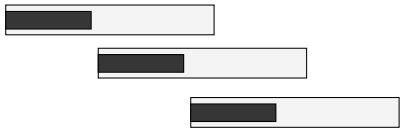
3.2 Gerarchie di memoria
Tecniche d'uso di DRAM 1

- Tempo di accesso: tempo necessario alla memoria per accedere alla cella indirizzata
- Tempo di lettura: tempo necessario alla CPU per acquisire un dato dal DB, dall'emissione dell'indirizzo in AB



3.2 Gerarchie di memoria
Tecniche d'uso di DRAM 2

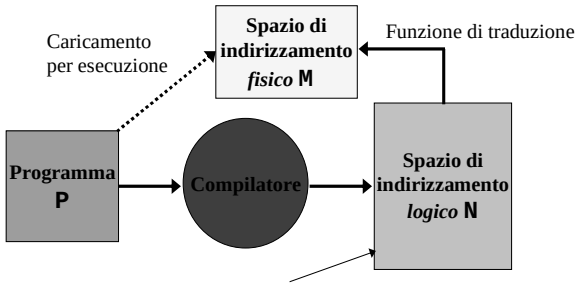
- Memorie interallacciate (*interleaving*): Accesso parallelo in memoria *senza* attendere la conclusione della lettura precedente



3.3 Memoria virtuale
Generalità

- Il concetto di *memoria virtuale* si propone di incrementare l'efficienza d'uso di un sistema, ampliandone, a costi contenuti, la capacita' di memoria
  - Permettendo la presenza in memoria di **più** programmi simultaneamente
  - Permettendo l'esecuzione di programmi di dimensioni **maggiori** dell'ampiezza della memoria principale
  - Consentendo un utilizzo più avanzato dei vari livelli di memoria disponibili

3.3 Memoria virtuale
Spazi di indirizzamento - 1



L'insieme di tutti gli indirizzi, noti a tempo di compilazione, che P potrà generare a tempo di esecuzione

3.3 Memoria virtuale
Spazi di indirizzamento - 2

- A tempo di esecuzione di **P**, il processore genera indirizzi **logici** appartenenti allo spazio **N** assegnato dal compilatore al programma
- **N** è ampio al più 2^p indirizzi, dove p è l'ampiezza in bit dell'indirizzo **logico**
- La memoria principale offre uno spazio di indirizzamento **fisico M** al programma caricato per l'esecuzione
- L'ampiezza di **M** **non coincide** necessariamente con quella di **N**

3.3 Memoria virtuale
Spazi di indirizzamento - 3

- Nel caso di allocazione **statica**, **N** ed **M** coincidono
  - Una traduzione statica può essere necessaria se **N** ed **M** usano una **base diversa**
  - Tale traduzione avviene nella fase di caricamento (*linking loader*)
- Nel caso di allocazione **dinamica**, **M** varia sia in ampiezza che in posizionamento
  - Ciò richiede l'uso di una **funzione di rilocazione** $f : \mathbf{N} \rightarrow \mathbf{M}$ di cui si occupa il gestore della memoria

3.3 Memoria virtuale
Allocazione dinamica - 1

- La funzione di rilocazione f può utilizzare
  - **Paginazione**
    decomponendo sia **N** che **M** in blocchi di ampiezza **fissa** e contenenti informazioni **contigue** (**pagine**)
  - **Segmentazione**
    decomponendo **N** ed **M** in aree di ampiezza **variabile**, contenenti informazioni **contigue** e corrispondenti a **specifiche** entità del programma (**segmenti**)
- La funzione f viene valutata per ogni indirizzo logico generato dal processore per il programma in esecuzione

3.3 Memoria virtuale
Allocazione dinamica - 2

- L'allocazione statica o dinamica dello spazio *fisico* del programma è **altra cosa** rispetto all'allocazione delle informazioni nello spazio *logico* del programma
  - Il programmatore può operare sulla seconda mediante costrutti appositi del linguaggio
- Entrambe possono richiedere supporto di sistema operativo

3.3 Memoria virtuale
Gestione

- La gestione della memoria virtuale è assai simile a quella della cache
  - Memoria **principale** (livello inferiore) più veloce e costosa; memoria **secondaria** (livello superiore) molto piu estesa, lenta ed economica
  - Suddivisione in blocchi (pagine o segmenti)
  - *Fault di memoria* se il dato richiesto non è presente in memoria principale
  - Prelievo del blocco contenente il dato dal livello superiore ed allocazione (con rimpiazzo) al livello inferiore

3.3 Memoria virtuale
Parallelo cache - memoria virtuale

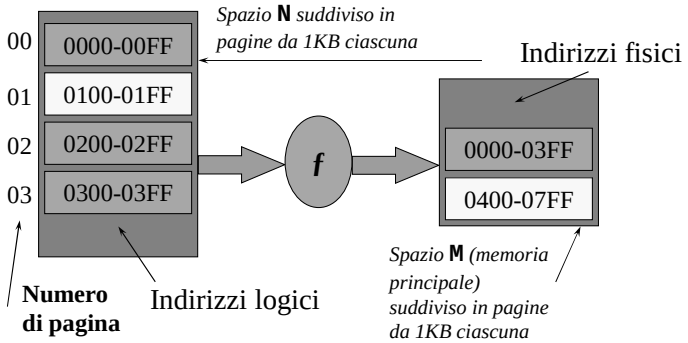
La cache *differisce* dalla memoria virtuale per:

- La *dimensione dei blocchi*, molto inferiore per la cache
- La *gestione*, che, per la cache, viene risolta interamente a **livello hardware**, mentre per la memoria virtuale richiede l'intervento del **livello sistema operativo**

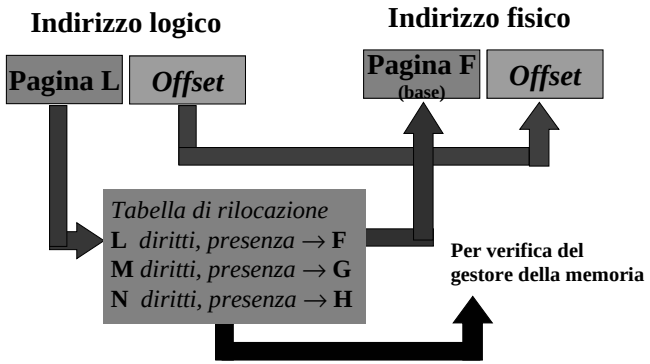
3.3 Memoria virtuale
Paginazione - 1

- **N** ed **M** suddivise in blocchi della medesima dimensione fissa e predefinita
- Lo stesso avviene per memoria principale e memoria secondaria
- La funzione f mappa **N** su **M**, generando un indirizzo (di pagina) in memoria principale
- La pagina richiesta può però trovarsi *solo* in memoria secondaria

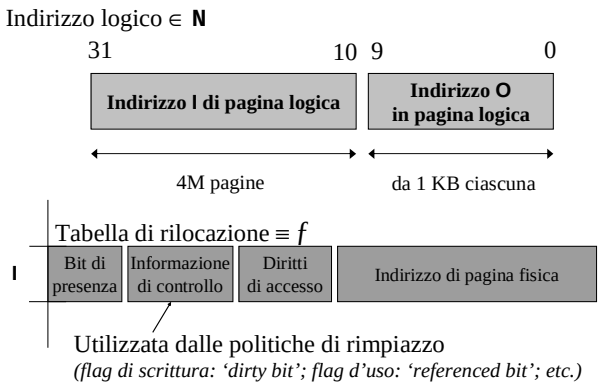
3.3 Memoria virtuale
Paginazione - 2



3.3 Memoria virtuale
Paginazione - 3



3.3 Memoria virtuale
Paginazione - 4



3.3 Memoria virtuale
Paginazione - 5

- *Page Fault*, quando la pagina indirizzata non è presente in memoria principale
- *Protection Fault*, quando il richiedente non ha diritti sufficienti per la richiesta
- Il *controllore degli accessi* è tipicamente un componente di sistema operativo (talvolta un componente fisico dedicato)

3.3 Memoria virtuale
Paginazione - 6

- La tabella di rilocazione può risiedere in memoria principale ed essere paginata
  - Il suo beneficio d'uso però scema con l'aumentare della frequenza d'accesso
- L'alternativa è l'uso di un dispositivo detto *Memory Management Unit* (**MMU**) contenente (parte del)la tabella
  - I programmi tendono a concentrare i riferimenti su una *piccola* quantità di pagine

3.3 Memoria virtuale
Paginazione - 7

- Le prime MMU erano realizzate in h/w, con intervento del S/O solo per la gestione degli eventi *page fault*
- Attualmente, la gestione delle pagine avviene prevalentemente a livello di S/O
- Per tabelle di rilocazione larghe abbastanza (p.es. 64 ingressi **I**), con bassa frequenza di miss, la gestione s/w è generalmente adeguata

3.3 Memoria virtuale
Frammentazione interna - 1

- Non vi è dimensione *ottimale* di pagina, ma un ottimo *locale* per applicazione
- Una dimensione *ridotta* di pagina è preferibile per ridurre la **frammentazione interna**
  - Dati *correlati* del programma vengono posti nella medesima pagina
  - La dimensione dei dati spesso non corrisponde ad un multiplo *esatto* di pagine
  - Il grado di frammentazione interna denota lo spazio di pagina inutilizzato

3.3 Memoria virtuale
Frammentazione interna - 2

- Se compattiamo i dati del programma sulle pagine disponibili, l'ultima pagina resterà parzialmente inutilizzata
  - In media, *metà* dell'ultima pagina
- Per un programma di dimensione s byte, p byte per pagina, ed e byte per ingresso in tabella di rilocazione, il costo medio C è

  $C = (s/p)e + p/2$ il cui minimo per p è $p = \sqrt{2se}$
  - Per s = 1MB ed e = 8B si ha p = 4KB (valore attuale)

3.3 Memoria virtuale
Politiche di rimpiazzo delle pagine - 1

- **Least Recently Used** (LRU)
  - Si rimpiazza la pagina usata *meno recentemente*
  - Di difficile implementazione
    - Occorre una lista ordinata, aggiornata ad ogni accesso, dove ogni pagina acceduta viene posta in cima → LRU = fondo della lista
- **Not Recently Used** (NRU)
  - Si rimpiazza una pagina *non utilizzata di recente*
  - Bit di utilizzo, periodicamente azzerato

3.3 Memoria virtuale
Politiche di rimpiazzo delle pagine - 2

- L'implementazione di LRU richiede supporto h/w spesso non disponibile
- Il livello S/O ne implementa una variante detta **Not Frequently Used** (NFU)
  - NFU usa la nozione approssimata di invecchiamento dall'ultimo accesso ('*aging*') mediante contatore di n bit
  - Con periodo t, il gestore trasla a destra di 1 il contatore di età e pone il bit di riferimento in MSb
  - Il contatore minore indica la pagina riferita meno frequentemente nell'intervallo di osservazione ($t \times n$), non necessariamente la LRU

3.3 Memoria virtuale
Politiche di richiesta di pagina

- **A richiesta** (*on demand*)
  - Si richiede la pagina mancante quando si è verificato un *page fault*
- **Con pre-caricamento** (*prefetching*)
  - Si *anticipa* la richiesta delle pagine adiacenti a quello in uso (giovandosi della proprietà di località spaziale)
  - Può causare ulteriori *page fault*

3.3 Memoria virtuale
Congruenza delle pagine

- Il gestore della memoria virtuale deve assicurare congruenza tra memoria principale e secondaria a seguito di scrittura dei dati
- L'alto costo di accesso alla memoria secondaria comporta l'uso della tecnica **write back** vista per la cache

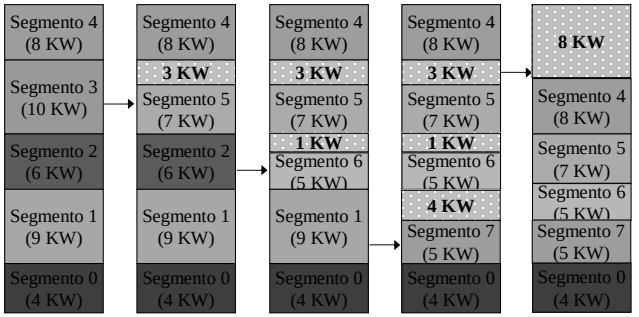
3.3 Memoria virtuale
Segmentazione - 1

- La **paginazione** offre uno spazio di indirizzamento *uni-dimensionale*
  - Le pagine sono entità *dissociate* dalle esigenze e dalle strutture del programma (spazio di indirizzamento continuo)
- La **segmentazione** offre uno spazio di indirizzamento *multi-dimensionale*
  - I segmenti hanno dimensione variabile sia nello spazio che nel tempo
  - Ogni segmento costituisce uno spazio di indirizzamento a se stante
  - I segmenti sono entità *strettamente legate* alle strutture del programma

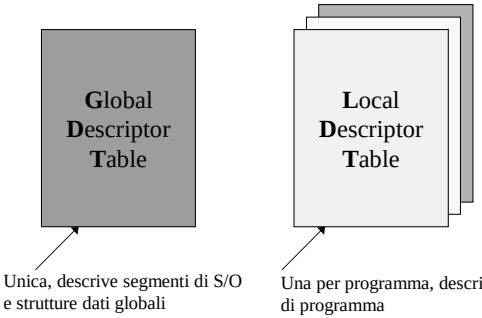
3.3 Memoria virtuale
Segmentazione - 2

- La segmentazione
  - Risponde ad esigenze *diverse da e superiori a* quelle della paginazione
    - Separazione tra dati ed istruzioni
    - Protezione di accesso più puntuale e specifica
  - Ha maggiore costo di implementazione ma anche maggiore flessibilità
    - Le dimensioni di segmento possono variare dinamicamente
  - Causa **frammentazione esterna**
  - Può essere combinata con la paginazione
    - Segmenti grandi possono essere paginati

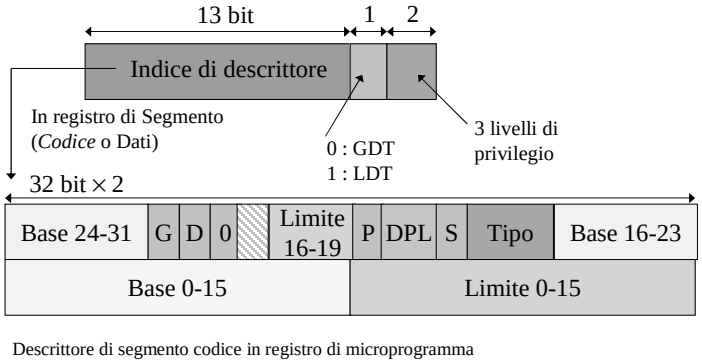
3.3 Memoria virtuale
Frammentazione e ricompattazione



3.3 Memoria virtuale
La soluzione Intel Pentium - 1



3.3 Memoria virtuale
La soluzione Intel Pentium - 2



Descrittore di segmento codice in registro di microprogramma

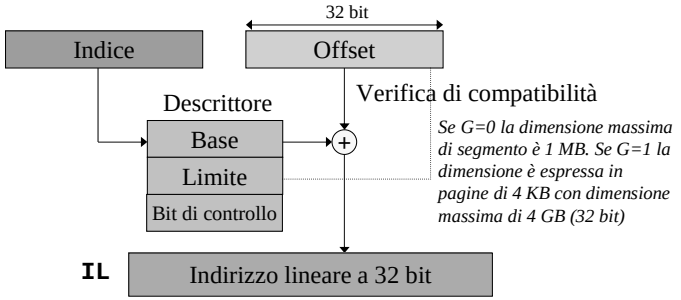
3.3 Memoria virtuale
La soluzione Intel Pentium - 3

Bit di controllo nel descrittore di segmento

G → 0 : Limite (*dimensione*) espresso in byte ; 1 : in pagine di 4kB
D → 0 : segmento a 16 bit ; 1 : a 32 bit
P → 0 : segmento assente da memoria ; 1 : presente
DPL → 3 livelli di privilegio
S → 0 : sistema ; 1 : applicazione
Tipo → tipo di segmento (Codice/Dati) e protezione

L'**indice** designa al più 8k descrittori su 16k disponibili
La **dimensione** di segmento dipende da G (*granularità*)

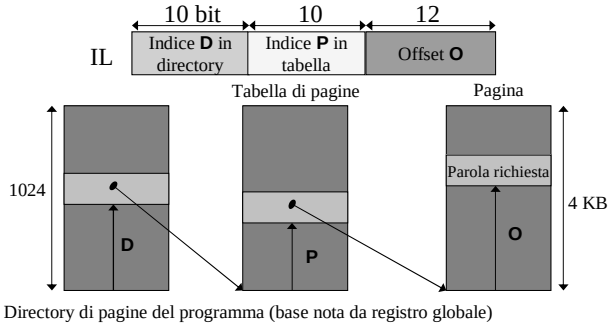
3.3 Memoria virtuale
La soluzione Intel Pentium - 4



3.3 Memoria virtuale
La soluzione Intel Pentium - 5

- Un bit in un registro globale di controllo indica se la paginazione è abilitata o meno
- Se *disabilitata*, abbiamo uno schema a segmentazione *pura*, dove IL rappresenta un indirizzo fisico
  - Segmenti diversi possono sovrapporsi
- Se *abilitata*, IL viene trattato come un indirizzo *virtuale*

3.3 Memoria virtuale
La soluzione Intel Pentium - 6



3.3 Memoria virtuale
La soluzione Intel Pentium - 7

- La gestione coinvolge *sia* il livello di macchina microprogrammata *che* il livello di macchina virtuale (S/O)
- Il S/O interviene ogniqualvolta si verifichino **eccezioni** (violazioni di diritti di accesso, offset erroneo, assenza di segmento, …)
- Tali eccezioni sono **sincrone** all'esecuzione del programma e si chiamano **trap**

3.3 Memoria virtuale
Segmentazione - 3

- Traduzione complessa e costosa
- Delicata la scelta di dove allocare nuovi segmenti in memoria principale
  - **Best-fit**, **Worst-fit**, **First-fit**
- La frammentazione esterna richiede ricompattazione periodica
  - **Garbage collection**

3.3 Memoria virtuale
Considerazioni finali

- La memoria virtuale estende lo spazio di indirizzamento *oltre* il limite fisico della memoria principale
- La paginazione è *trasparente* al programmatore, la segmentazione *no*
- La segmentazione consente maggior protezione, anche tra processi *concorrenti*
- La tabella di rilocazione può essere complessa