

## LA MEMORIA VIRTUALE

La motivazione iniziale per cui è stata inventata la memoria virtuale è proprio la necessità di eseguire programmi che richiedono più spazio della RAM disponibile. In sistemi "primitivi" questo si realizzava permettendo ai programmatore di spezzare il codice e i dati di un programma in tante "sezioni" chiamate **overlay**. Il sistema operativo forniva la possibilità di rimpiazzare un overlay con un altro durante l'esecuzione. Lo svantaggio di questo approccio è legato alla complessità della gestione degli overlay, lasciata a carico del programmatore.

La tecnica della **paginazione** svolge in automatico, in modo completamente trasparente per il programmatore, la stessa funzione con overlay tutti della stessa dimensione.

In un sistema multiprogrammato (multitasking) inoltre tale tecnica permette di mantenere in memoria un insieme di processi che complessivamente occuperebbero più spazio della memoria RAM disponibile.

1

Motivazioni: Spazio di memoria più grande, possibilità di avere numero di processi alto a piacere, riducendo numero di page. Costi: indirizzo relativo e fisico può crescere al costo.

Avremo più difficoltà a gestire un memoria efficiente (anche cache x es.)

## LA MEMORIA VIRTUALE PAGINATA

- L'immagine in memoria di ciascun processo viene suddivisa in blocchi di uguale dimensione chiamati **pagine (logiche)**
- La memoria fisica viene suddivisa in **frame** (o pagine fisiche) della stessa dimensione delle pagine logiche
- Per ogni processo viene mantenuta una **tavola delle pagine**, che indica per ciascuna pagina dello spazio di indirizzi del processo se si trova o meno in memoria RAM, e in quale frame. Occorre inoltre tenere traccia della posizione delle pagine logiche su disco (l'immagine completa del processo deve comunque essere memorizzata su disco).
- La MMU, dato un indirizzo logico, dovrà scoprire in quale pagina logica si trova, controllare se la pagina si trova in RAM: in caso affermativo operare la traduzione da indirizzo logico a indirizzo fisico, se no causare una trap di tipo **page fault** (mancanza di pagina) che verrà opportunamente gestita dal sistema operativo (tramite il **page fault handler**)

2

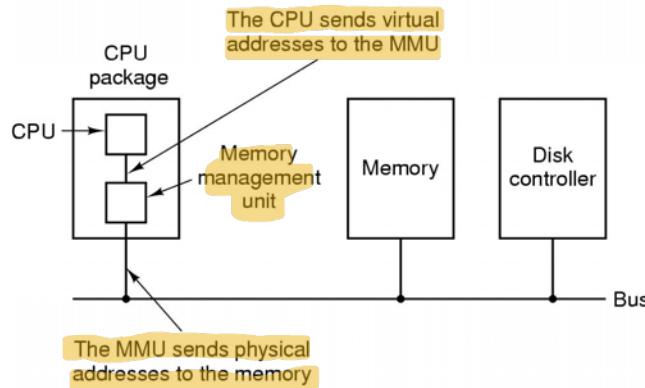
Allocazione è fatta a pezzi: nella memoria fisica ci sono alcune pagine logiche dei nostri processi. Qua i block sono tutti dello stesso dimensione dei pezzi.

Problema: serve struttura del processo che dica in ogni momento se pagina per un determinato processo sta esecuzione sta in memoria cache, e se si dà. Se una pagina non è presente in memoria principale è su disco. È drammatico

# Memoria Virtuale

## MMU

L'unità MMU esegue la conversione degli indirizzi virtuali in indirizzi fisici.

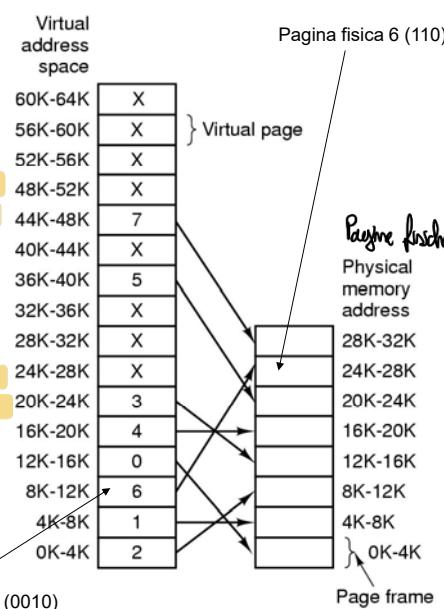


## Funzione della MMU

3

# Paginazione

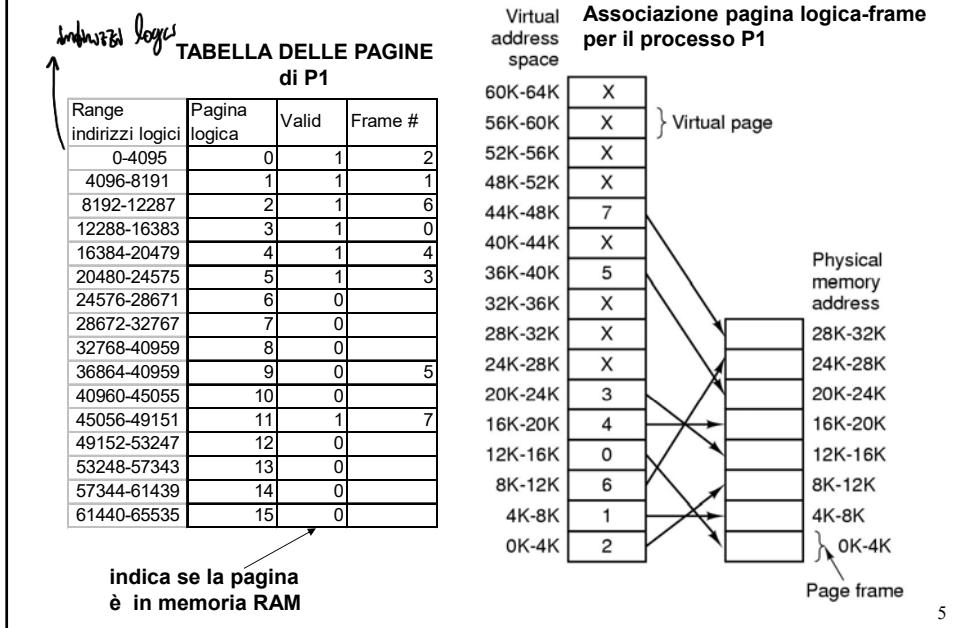
In questo esempio lo spazio di indirizzi del processo P1 comprende 16 pagine logiche da 4kbyte ciascuna (per un totale di  $4K \cdot 16 = 2^{16}$  byte). La memoria RAM ha solo 8 frame (da 4kbyte ciascuno): la figura mostra una situazione in cui sono caricate in RAM le pagine logiche 0, 1, 2, 3, 4, 5, 9, e 11 del processo, rispettivamente nei frame 2, 1, 6, 0, 4, 3, 5 e 7.



4

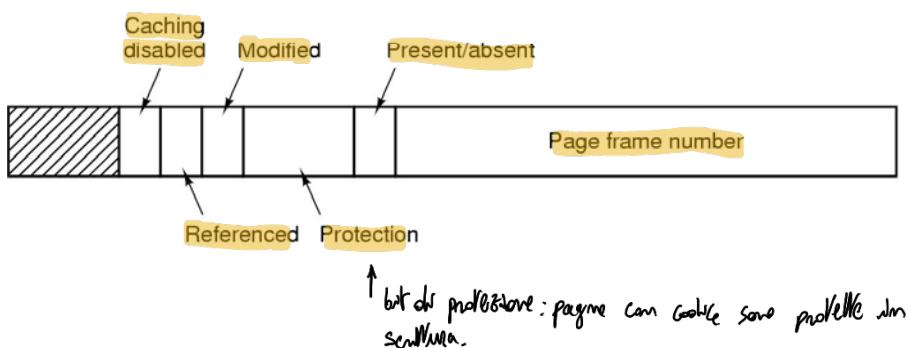
Indirizzo virtuale da 0 a 64K e indirizzo fisico da 0 a 32K

## LA TABELLA DELLE PAGINE



Alla Riga 0 Rigo 2 info: bit che dice se pagina sta o no in memoria. 0 non ci sta, 1 ci sta.  
Nel campo successivo il numero del frame in cui sono allocati i primi 4096 byte.

## Record di una tabella delle pagine

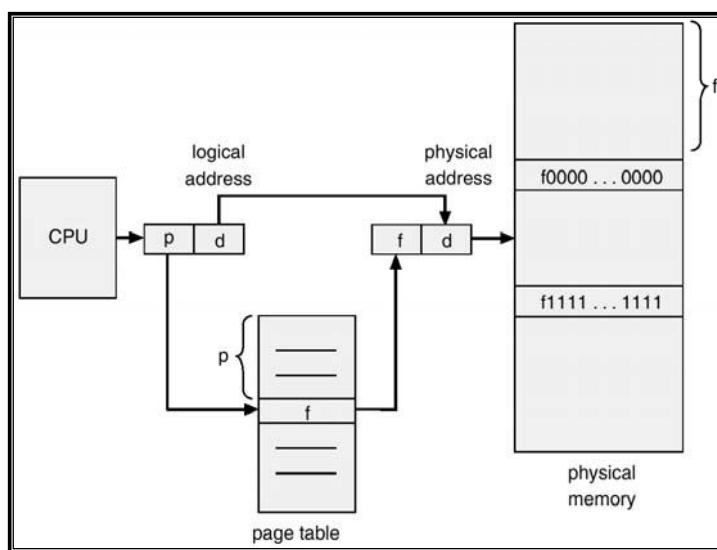


*Note: se pagina viene modificata, la copia della pagina su disco è diversa da quella in memoria centrale, quindi questa va ricopiatà.*

## Record di una tabella delle pagine

7

## Traduzione indirizzo logico indirizzo fisico (schema)



8

Come avviene la traduzione? Tutto è dividibile in 2 pezzi: il primo è il numero di pagina logica, il secondo è l'offset nella pagina.  
Offset, quando ricopio la pagina è sempre uguale se pagina viene mappata, questo non ci dà problemi. Traduziono quindi il numero di pagina logica da p

## TRADUZIONE DA INDIRIZZO LOGICO A INDIRIZZO FISICO

La MMU traduce gli indirizzi logici (IL) in indirizzi fisici (IF)

- calcolo numero di pagina logica (NP) e offset (O):

$$NP = IL / DimPag; O = IL \bmod DimPag$$

(Se la DimPag è pari a  $2^k$ : O sarà pari ai k bit meno significativi e NP i restanti bit)

- if (TabPag[NP].Valid == 0) then trap(page fault)**

else NF = TabPag[NP].Frame (dove mappa dlr pagina fisica)

- indirizzo fisico = indirizzo inizio frame fisico + O = DimPag \* NF + O

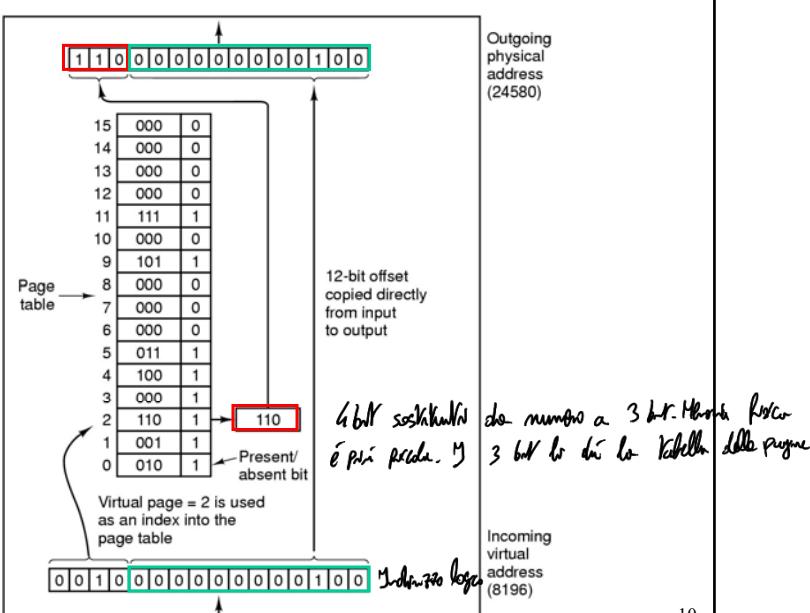
Nel caso si verifichi trap(page fault) il sistema operativo deve:

- cercare un frame libero in RAM: se non c'è scegliere una pagina "vittima" per lo swap-out (tramite l'algoritmo di sostituzione)
- se necessario, salvare la "vittima" su disco
- caricare la pagina che ha causato il page fault da disco a RAM (nel frame libero) e aggiornare di conseguenza la tabella delle pagine (Valid bit = 1, Frame = frame fisico in cui si è caricata la pagina).

9

3) Operazioni non necessarie, basta concatenare bit. I bit meno significativi rappresentano offset, la parte alta al numero di pag. log.

## ESEMPIO DI TRADUZIONE



10

Indirizzi logici sono di 16 bit. Indirizzo fisico da 0 a 32K vuole 15 bit (64K)

E' dentro codice ho indirizzo logico da 8196, chiamato per 4096 (che è il primo 12 bit per offset).

Quindi 2 (che è questa pagina nella tabella), resto 4, così offset. Prendo 2 e resto a quella riga.

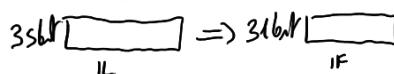
Quattro bit di validità: pari a 1 e ho un altro numero. Pagina logica 2 ha associato valore 110, che viene incrementato. 110 con 12 zeros davanti è 26376+4, non devo fare moltiplicazioni né divisioni

Punte più significative che usano nella tabella delle pagine ed estrarre bit da copiare non volta all'offset.

ES:

MV=32Gb, con memoria fisica MF=2Gb, P=64Kb

Indirizzo Logico ( $G=30\text{bit}$ ) :  $2^{30}\text{bit}$ . Indirizzo fisico fatto IF=31bit



Di 35 bit, 16 meno signif. rappresentano offset.  $35-16=19$  bit sono  $2^{19}$  pagine logiche, circa 512K pagine logiche da 64Kb.

Più IF ha  $31-16=15$  pagine fisiche in memoria (frame in memoria, circa 32K pagine di 64Kb).

Importante valutare il numero di pagine che vuole ad avere. Le tabelle possono essere grandi. Per ogni processo ha qualche tabella che verrà comunque salvata. La tabella ha costo sia in termini di spazio (struttura dati che non avrà), sia in termini di tempo. Si ritiene che sia essa sull'intera memoria centrale.

## Tabella delle Pagine

- La tabella delle pagine può essere estremamente estesa
  - con indirizzi a 32 bit e pagine di 4K, la tabella delle pagine ha un milione di record \*
- La traduzione degli indirizzi deve essere veloce
  - più accessi alla page table per istruzione
  - il tempo di accesso alla page table deve essere comparabile al tempo di esecuzione medio di una istruzione (es. se il tempo di esecuzione medio è di 4ns, l'accesso alla tabella delle pagine deve essere dell'ordine di 1ns)

11

\* Spazio occupato da singola riga? Come minimo, bit Valid/Inv e numero del frame fisico

Almeno 1bit + 10bit per ogni pagina di memoria fisica, più o meno potremmo dire su 4 byte.

Per ogni accesso ne deve fare un altro per accedere alla tabella delle pagine. Non se fare

1° Costo: tempo di traduzione

## OVERHEAD PER PAGINAZIONE

La paginazione introduce diversi tipi di overhead:

### - tempo:

- tempo di traduzione indirizzo logico in indirizzo fisico (il tempo di accesso alla page table deve essere comparabile al tempo di esecuzione medio di una istruzione)
- ad ogni context switch occorre caricare la tabella delle pagine (o parte di essa) nella MMU.
- il trattamento dei page fault possono far crescere di molto i tempi di accesso alla memoria.

### - spazio:

- per ogni processo occorre mantenere una tabella delle pagine che può essere molto grande (dipende dalla dimensione delle pagine).
- vi è frammentazione interna (si perde mediamente  $\frac{1}{2}$  pagina per ciascun processo, dato che la dimensione dei processi raramente è un multiplo della dimensione della pagina).

12

↳ Un processo non occupa quasi mai esattamente un numero di pagine intero.

Soluzione per il tempo: aggiungo HW: ho un pessimo hardware aggiuntivo, un buffer (memoria, sostituzione di register)

A differenza della RAM è una memoria associativa. Register contengono coppie chiave-valore. Memoria è associativa perché acc. avviene per chiave. Se chiave è presente, memoria restituisce valore associato a quella chiave. Qui, chiave = numero di pagina logica.

Valore: n. di frame fisico  
TRAUDUZIONE AVVIENE ALLO STESSO MODO. Chiave è confrontata in parallelo. Ovviamente è memoria con dimensione ridotta, contiene un numero limitato di pagine. Tuttavia può fallire e dopo accedere alla memoria principale alla tabella delle pagine.

## TLB - Translation Lookaside Buffer (overhead di tempo)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

### Esempio di TLB

13

ES:  $T_{LB} = 20 \text{ MS}$   
 $MF = 100 \text{ MS}$

memoria fisica

Senza TLB:  $T = 200 \text{ MS}$  (doppio accesso per MISS)

Con TLB:  $T = 100 + 20 \text{ MS}$  (hit: c'è nel TLB)

Hp:  $\text{Per cento HIT } 80\% = d \Rightarrow 0.8 \cdot 120 + 0.2 \cdot 200 = 136 \text{ MS}$

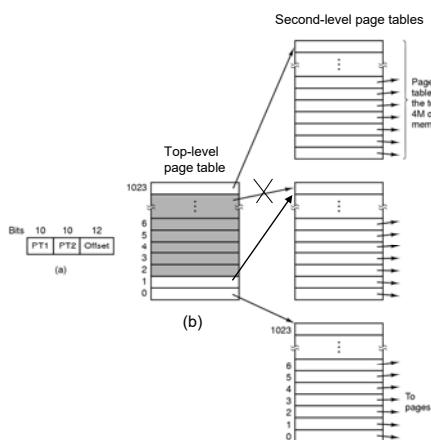
Importante avere un TLB che in ogni momento ha le righe giuste

NOTA: A ogni cambio di contesto devo scrivere nel mio TLB.

1° SOL: avere solo le righe del processo running nel TLB.

2° SOL: fare coesistere righe diverse, ma in quel caso, Key = num. pagina + id del processo

## Tabella delle Pagine a due livelli (overhead di spazio - dim. della tabella)



(a) Indirizzi a 32 bit con due campi per la tabella delle pagine

(b) Tabella delle pagine a due livelli

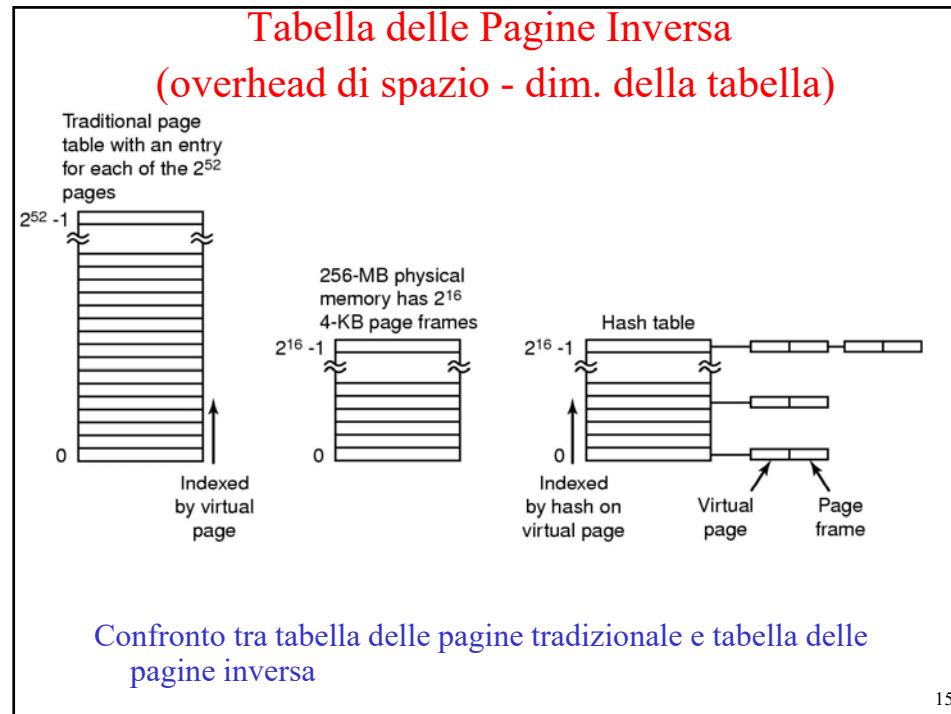
14

Come riduco spazio occupato dalle tabellen delle pagine? Pagine a più livelli: es. della tabella con 1 milione di righe con 20 bit più significativi. Se li divido in blocchi da 10, ho 1024 tabelline da 1024 righe. Ho un po' bisogno di una tabellina madre per le 1024 tabelline. Quindi ora prendo i 20 bit, parto con i primi 10 per accedere alla tabellina madre (1024 righe con indirizzo delle tabellen delle pagine). Quel è il vantaggio? Se uso tutto il milione di pagine, ho anche tabella degli indirizi. Per la traduzione richiede 2 passi! Vantaggio è se quel processo usa solo una parte di quelle pagine. Quindi tabella madre contiene 3 puntatori.

**Soluzione alternativa:** traduzione in fatto da indirizzo logico. Qui la traduzione avviene con algoritmo di hashing.  
 Invece di usare una tabella che ha tutte le possibili pagine logiche del processo, stabilisce che queste ne ha solo 250K da 1M, per esempio.  
 Questo lo faccio con algoritmo di hashing. Problema della collisione di cui dobbiamo tenere conto però.  
 Come risolvilo? Faccio in modo che tabella non sia lineare, ma si liste concatenatale.  
 NOTA: nella tabella delle pagine, non c'è numero di pagina logica di sotto, ma qui sì, per la collisione.

Problema delle vecchie tabelle se non ci sono tutte le pagine, debbo fare ricerca.

Entrambe le situazioni funzionano peggio nel caso peggiore.



## Dimensione delle Pagine (overhead di spazio)

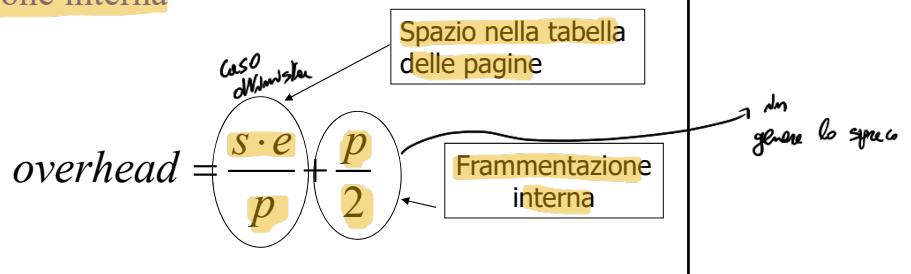
### Pagine piccole

- **Vantaggi**
  - Riducono la frammentazione interna
  - Si adattano meglio a varie strutture dati e sezioni di codice
  - Limitano l'ampiezza dello spazio di indirizzamento inutilizzato caricato in memoria
- **Svantaggi**
  - I programmi necessitano di parecchie pagine, tabelle delle pagine più grandi

# Dimensione delle Pagine

## (overhead di spazio)

- Overhead dovuto alla tabella delle pagine e alla frammentazione interna



- Dove

- s = dimensione media di un processo (in bytes)
- p = dimensione della pagina (in bytes)
- e = descrittore di pagina (in bytes)

L' spazio occupato da singola riga della tabella

Ottimizzata quando

$$p = \sqrt{2se}$$

7

overhead: quanto spazio di memoria speso con paginazione

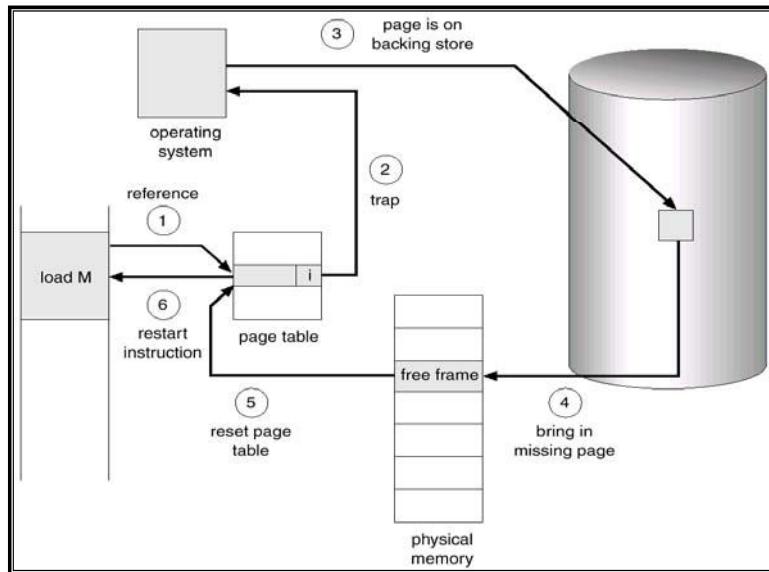
# Implementazione della Paginazione

Il Sistema operativo invoca i meccanismi di paginazione in quattro circostanze:

- Creazione di un Processo
  - Determina la dimensione del programma
  - Crea la tabella delle pagine
- Esecuzione di un Processo (*context switch*)
  - Deve reimpostare la MMU per il nuovo processo
  - Aggiornamento del TLB (*flush*)
- Page fault
  - Determina l'indirizzo logico che ha causato il page fault
  - Sposta una pagina su disco (se necessario) e carica la pagina richiesta
  - Ripristina l'istruzione che ha causato il page fault
- Terminazione di un Processo
  - Dealloca la tabella delle pagine e le pagine del processo

18

## Passi della gestione di un Page Fault



19

In caso di page fault deve sapere questo componente Kernel dove andare a prendere file, mentre trovando frame fisico libero. Non è ancora il caso pagine phys: es. non trovo buchi liberi nella memoria centrale. Non c'è frame fisico libero in questo esempio. Problema è trovare lo pagina.

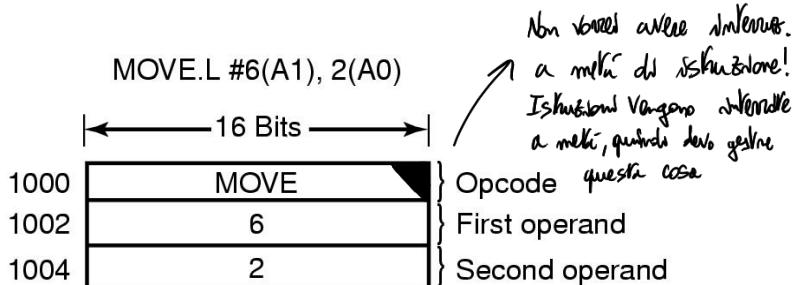
## Cosa accade quando non c'è un frame libero ?

- **Sostituzione delle pagine**  
va scelta una pagina in memoria che va spostata sul disco (*swap out*).
  - Algoritmo di sostituzione: si vuole un algoritmo che dia il numero minimo di page fault.
- Alcune pagine possono essere spostate dal disco alla memoria centrale e viceversa varie volte.

1. Voglio eseguire programmi con loro settato a 1 per scrollare, ma anche funziona da keygen page non uso.

20

## Ripristino dell'Istruzione



- Un'istruzione che ha causato un page fault
- Alcune macchine hanno dei registri interni che mantengono informazioni di ripristino (backup)

21

DMA cosa fa?  
Accede alla memoria dopo essere stato programmato

## PROBLEMI DI INTERAZIONE GESTIONE DELLA MEMORIA E I/O

- Memoria virtuale e I/O interagiscono occasionalmente
- Un processo richiede una lettura da un dispositivo su un buffer
  - Mentre attende per l'I/O passa in esecuzione un altro processo
  - Il processo in esecuzione causa un page fault
  - La pagina contenente il buffer del primo processo può essere selezionata per essere scaricata dalla memoria
  - Se il dispositivo di I/O stesse utilizzando il DMA per il trasferimento in memoria, parte dei dati verrebbero scritti nel buffer e parte nella nuova pagina appena caricata
- Necessità di vincolare alcune pagine
  - Le pagine vincolate non possono essere scaricate dalla memoria (pinning)

22

DMA: direct memory access: dispositivo HW che consente di eseguire le operazioni di scrittura e lettura sul disco. Lavora direttamente col processore quando lui non accede alla memoria centrale e si sovrappongono rendendo accessi al disco più veloci ed efficienti. Se SO quando ha page fault decide di togliere da memoria fisica pagina degli oggetti operazioni DMA, lui continua a scrivere la pagina non gestita. Poi non prendere in considerazione pagina non uso del DMA, con un word per ogni flusso a 1 del DMA.