# Il rimpiazzamento delle pagine

Per gestire una *trap* di tipo *page fault* il sistema operativo deve:

- scegliere un frame in cui caricare la pagina; uno libero, se c'è, oppure uno che contiene una pagina che si spera vada bene rimpiazzare (salvandola su disco se modificata)
- leggere da disco la pagina mancante e caricarla nel frame scelto

L'algoritmo di rimpiazzamento che sceglie la pagina ad ogni *page fault* deve cercare di minimizzare il numero di page fault; ne sono stati considerati tanti, fra cui:

- Ottimale (ha bisogno di conoscere il futuro, ma lo si considera per misurare il massimo margine di miglioramento di quelli praticabili)
- Least Recently Used (LRU)
- Not Recently Used (NRU)
- FIFO
- della seconda possibilità o dell'orologio

# Il rimpiazzamento delle pagine

Per analizzare le prestazioni dei diversi algoritmi di rimpiazzamento si confrontano i numeri di page fault che si ottengono applicandoli a un insieme di *stringhe di riferimenti alla memoria*

Una tale stringa può essere data come:

- lista di indirizzi (di istruzioni e dati) generata dall'esecuzione di un processo o lista di pagine logiche
- eventualmente corredati dell'indicazione del tipo di accesso (lettura o scrittura)

Per un solo processo o per processi diversi, perché in un sistema multiprogrammato si possono usare due approcci al rimpiazzamento:

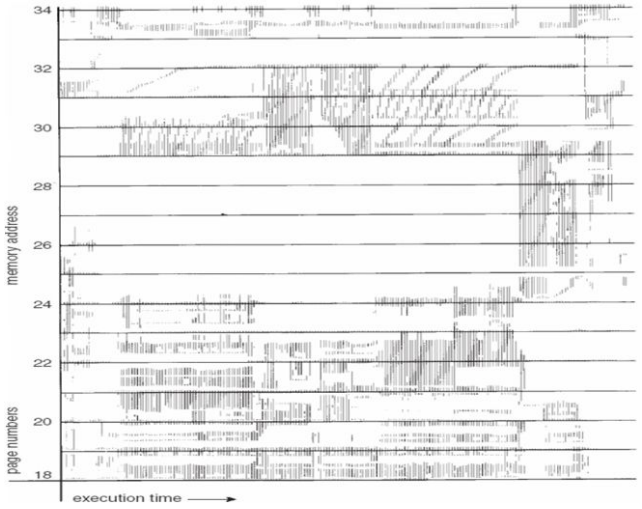
1) rimpiazzamento **locale**: ad ogni processo viene assegnato un numero di frame definito, quando P1 causa un page fault, la vittima per il rimpiazzamento può essere scelta solo tra le pagine di P1;
2) rimpiazzamento **globale**: la vittima per il rimpiazzamento può essere scelta fra tutte le pagine in memoria.

# Il rimpiazzamento delle pagine

Idealmente bisognerebbe che in ogni momento in memoria vi fosse l'intero ***working set*** (WS) di ogni processo, informalmente: l'insieme di pagine necessarie al processo in quella fase della sua esecuzione

Ma come si fa a sapere qual è?

Ricordiamo la **località**, nel tempo e nello spazio (in tempi vicini si accede agli stessi indirizzi o a indirizzi vicini, quindi probabilmente nelle stesse pagine)




Quindi ci si basa sul passato, supponendo di non essere nei (pochi) momenti di transizione; se davvero sono pochi, il costo è accettabile

Fissato un numero *k* di riferimenti da considerare, il *working set* del processo in un dato istante della sua esecuzione può essere definito come l'insieme di pagine utilizzate negli ultimi k riferimenti alla memoria.

Oppure si fissa una "finestra" temporale di ampiezza $\tau$ unità di tempo, definendo il *working set* come l'insieme di pagine riferite nelle ultime $\tau$ unità di tempo di esecuzione.

# Algoritmo di rimpiazzamento ottimale

Esiste, ma richiede di conoscere i riferimenti futuri.

Data una stringa $r_1, r_2, r_3, ..., r_n$ di riferimenti a pagine logiche, le pagine vengono caricate finché non si riempie la RAM o il numero di frame assegnati al processo.

Al primo riferimento $r_k$ per cui é necessario un rimpiazzamento viene scelta come "vittima" la pagina che verrà nuovamente usata più lontano nel futuro. Si dimostra che è ottimale, cioè non si può trovare un criterio che causi meno page fault. Non é realizzabile, ma può essere usato come riferimento per misurare se ci possiamo accontentare di un altro algoritmo (es. negli esperimenti fa solo il …% di page fault in più dell'ottimale)

Esempio: 0, 2, 1, 3, 5, 4, 6, 3, 7, 4, 7, 3, 3, 5, 5, 3, 1, 1, 1, 7, 1, 3, 4, 1

| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
| | | 0 | 2 | 1 | 3 | 3 | 3 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 1 | 4 | |
| | | | 0 | 2 | 1 | 5 | 4 | 4 | 4 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 7 | 7 | 7 | 3 | 3 | 7 | 3 | 3 | |
| | | | | 0 | 2 | 1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 7 | 7 | |
| | * | * | * | * | * | * | * | | * | | | | | | | | * | | | | | | | | Tot.P.F.9 |
| Vittima | | | | | **0** | **2** | **1** | | 6 | | | | | | | | 5 | | | | | | | | |

# Algoritmo di rimpiazzamento LRU (Least recently used)

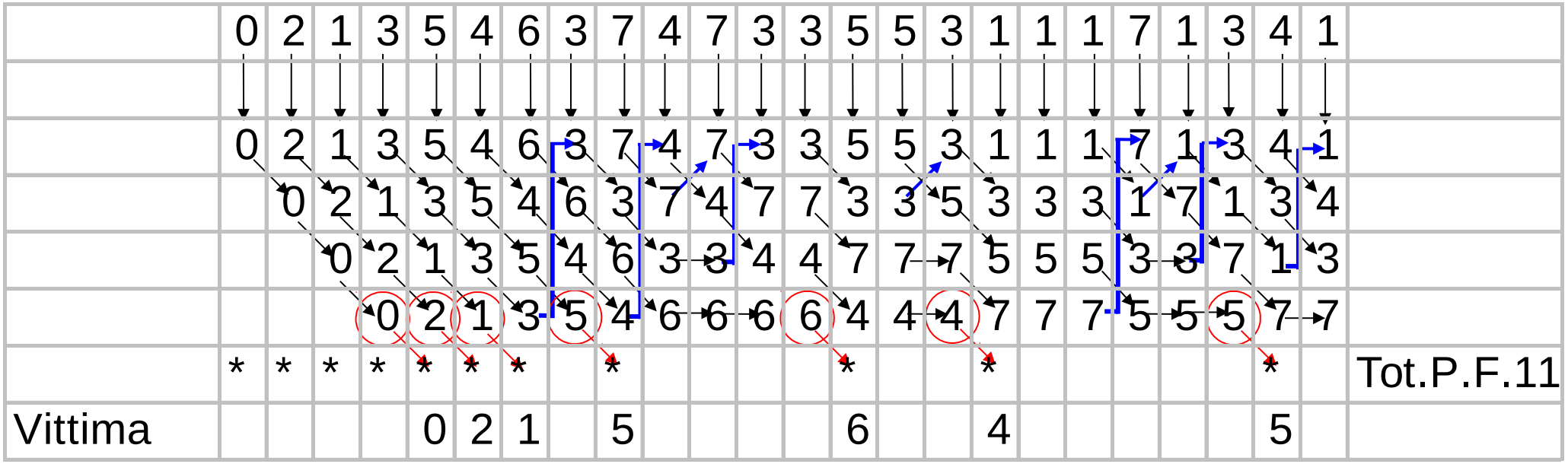
Viene scelta come “vittima” la pagina che é stata usata (per l’ultima volta) più lontano nel passato:

Esempio: 0, 2, 1, 3, 5, 4, 6, 3, 7, 4, 7, 3, 3, 5, 5, 3, 1, 1, 1, 7, 1, 3, 4, 1

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
| | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 | |
| | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 | |
| | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 | |
| | * | * | * | * | * | * | * | | * | | | | | * | | | * | | | | | | * | | Tot.P.F.11 |
| Vittima | | | | | 0 | 2 | 1 | | 5 | | | | | 6 | | | 4 | | | | | | 5 | | |

Si usa il passato come indicazione di ciò che accadrà nel futuro. Il principio di località temporale del resto porta a supporre che una pagina riferita nel passato recente ha più probabilità di essere nuovamente usata nel futuro vicino di una pagina riferita nel passato remoto.

# Algoritmo di rimpiazzamento LRU (Least recently used)



# Algoritmo Not Recently Used

LRU è troppo pesante da realizzare in modo esatto; si utilizzano delle approssimazioni con quello che "passa" l'hardware; in particolare, due bit per ogni frame:

R = bit di riferimento (diventa 1 quando la pagina viene letta/scritta)

M = bit di modifica (*dirty bit,* diventa 1 quando la pagina viene scritta)

| R | M | |
|---|---|---|
| 0 | 0 | Classe 0 |
| 0 | 1 | Classe 1 |
| 1 | 0 | Classe 2 |
| 1 | 1 | Classe 3 |

R e M vengono impostati dall'hardware e azzerati dal S.O.

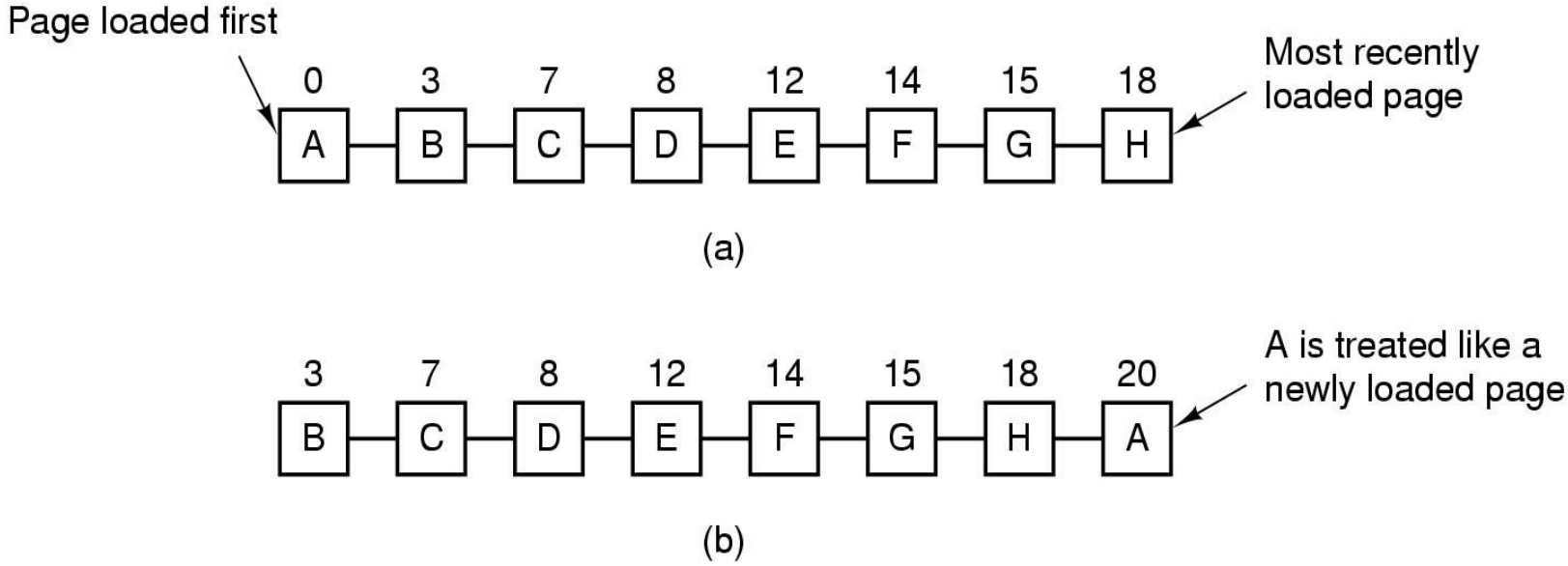
Per l'algoritmo NRU il bit R viene azzerato ad es. ogni k timer interrupt; così significa "c'è stato un accesso recente"

Quando si verifica page fault, se c'è bisogno di un rimpiazzamento, NRU sceglie in ordine crescente di classe (prima i frame in classe 0, poi in classe 1, ecc.).

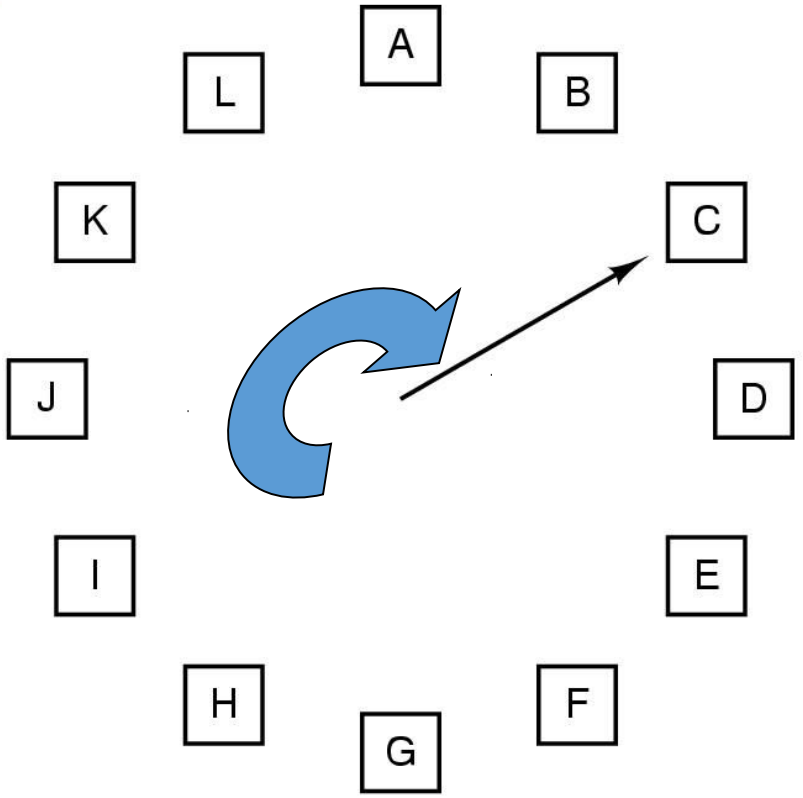
# Algoritmo FIFO e della seconda possibilità

FIFO: le pagine che si trovano in memoria sono mantenute in una lista, *ordinata in base all'istante di caricamento in RAM*. Al momento del rimpiazzamento viene scelta come "vittima" la pagina in testa alla lista. Viene quindi buttata via la pagina presente da più tempo *indipendentemente* (!) da quando sia stata riferita l'ultima volta.

L'algoritmo della seconda possibilità è una variante che utilizza il bit R: la pagina in testa alla lista viene "graziata" se il suo R è 1; viene portata in fondo alla lista (come se fosse stata appena caricata in memoria) e il suo R azzerato e si considera la successiva.



# Algoritmo della seconda possibilità o dell'orologio




When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

È una lieve semplificazione implementativa dell'algoritmo della seconda possibilità: si usa una lista circolare e anziché portare in fondo alla lista la pagina non rimpiazzata, si fa avanzare il puntatore (come se fosse la lancetta di un orologio, da cui il nome). Una nuova pagina va inserita subito prima di quella puntata dal puntatore.

# Algoritmo della seconda possibilità o dell'orologio

La scelta coincide con quella di FIFO se le pagine hanno tutte R=1 (si fa tutto il giro in un solo page fault scegliendo la pagina da cui si era partiti): ma se questo capita spesso, vuol dire che sono tutte molto usate e quindi non c'è da sperare di riuscire a tenere in RAM tutte le pagine "che servono"

Altrimenti (e quindi, si spera, tipicamente), una pagina "graziata" verrà ripresa in considerazione dopo un certo numero di page fault:

- se continua ad essere usata abbastanza di frequente, probabilmente nel frattempo sarà stata usata, avrà di nuovo R=1 e verrà graziata di nuovo
- se no, verrà scelta

Se c'è un numero di frame sufficiente per le pagine usate di frequente, queste tenderanno a non essere scelte

# Il problema del trashing

L'overhead (costo di gestione) in termini di tempo dovuto alla paginazione può diventare enorme in determinate situazioni: questo fenomeno è conosciuto con il nome di "*thrashing*" (produzione di spazzatura).

La causa del thrashing è la presenza contemporanea in memoria di *n* processi tali che la somma delle dimensioni dei loro *working set* eccede la dimensione della RAM

I *sintomi* di questo fenomeno sono:

- un altissimo tasso di *page fault* (page fault molto frequenti)
- bassa utilizzazione della CPU
- alta utilizzazione del disco che svolge il ruolo di *backing store*.

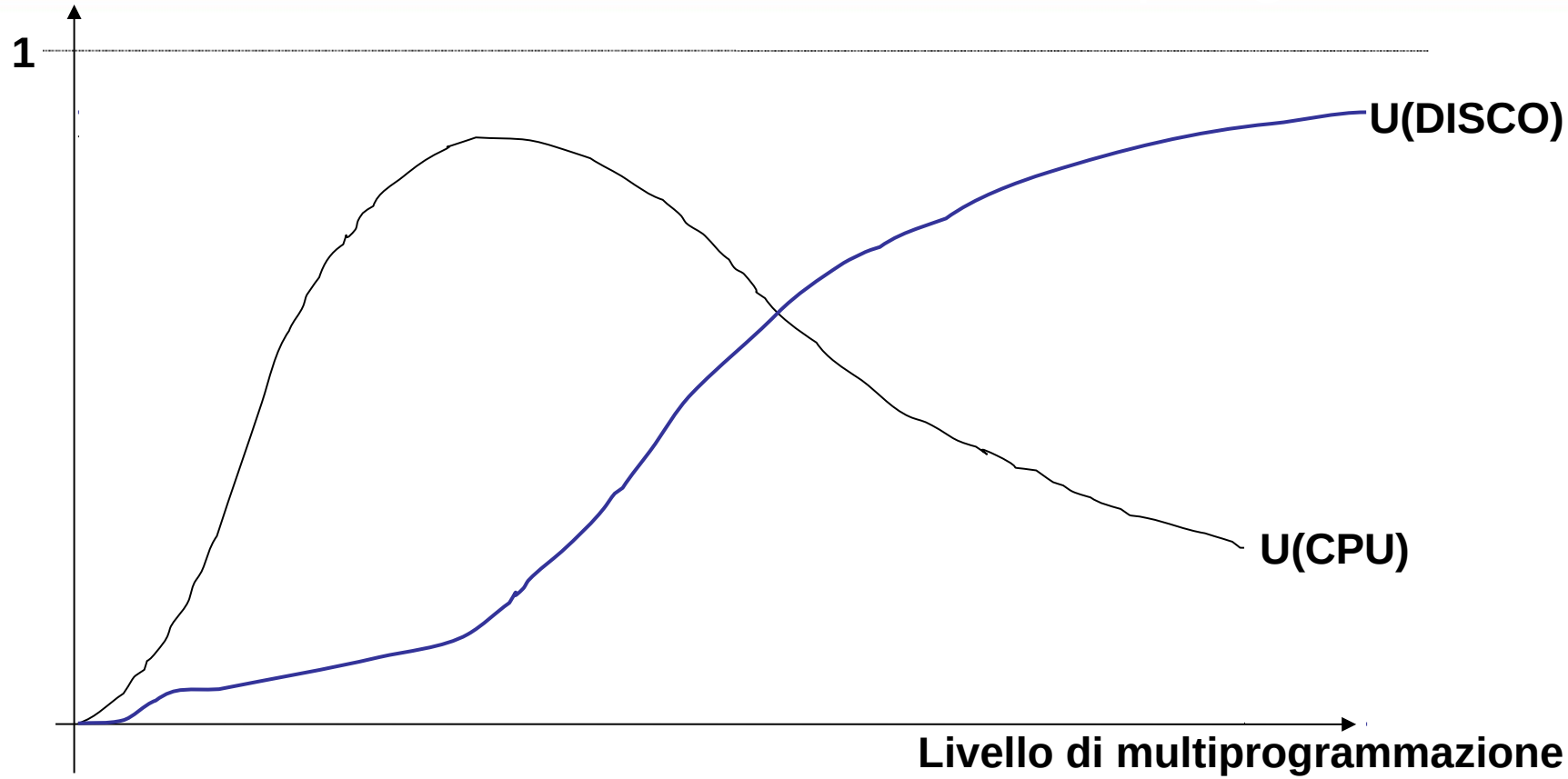
L'*effetto* è rallentare l'esecuzione di tutti i processi, dato che il sistema spende gran parte del tempo a gestire i page fault; la backing store diventa un collo di bottiglia, da cui bisogna passare per mandare avanti l'esecuzione di poche istruzioni di CPU

# Il problema del trashing

Per evitare il thrashing, oltre a cercare di mantenere in RAM solo il *working set* dei processi, occorre:

- monitorare il tasso di *page fault*
- monitorare la dimensione dei *working set* dei processi in memoria
- se necessario, fare swap-out di uno o più processi (liberare tutti i frame occupati da tali processi), in modo da lasciare in memoria un insieme di processi tali che tutti i loro *working set* possano coesistere in memoria (riducendo il livello di multiprogrammazione).

# Trashing e livello di multiprogrammazione




cioè non bisogna semplicemente applicare il criterio:

"se l'utilizzazione della CPU è bassa, allora aumenta il livello di multiprogrammazione"

perché se si arriva al thrashing, aumentare il livello di multiprogrammazione *peggiora* l'utilizzazione della CPU

# Algoritmi di rimpiazzamento basati sul working set

Usando l'informazione sulle pagine che costituiscono il working set dei processi in memoria si può:

- fare il possibile per evitare il thrashing e non dover fare swap out;
- o accorgersi di doverlo fare

Per definire quali pagine fanno parte del working set di un processo:

- si fissa un'ampiezza $\tau$ della *finestra temporale*
- si mantiene traccia per ogni frame di un tempo di ultimo riferimento: l'hardware imposta il bit R ad ogni riferimento, ad ogni timer interrupt:
  - i bit R vengono azzerati
  - ma se il bit R di un frame era 1, viene copiato nel suo campo "*ultimo riferimento*" il tempo di CPU complessivamente utilizzato (chiamato tempo *virtuale*) dal processo a cui appartiene.

Una pagina è nel WS se:

tempo virtuale attuale – tempo virtuale di ultimo riferimento < $\tau$

Se si vuole mantenere traccia dell'intero WS (ad es. per il prepaging) conviene mantenere l'elenco esplicito delle pagine nel PCB

# Algoritmi di rimpiazzamento basati sul working set

Quando avviene un page fault si può applicare l'algoritmo dell'orologio modificato in modo da *cercare di non rimpiazzare pagine contenute nel working set*

Se tutte le pagine sono nel working set, allora se ne sceglie una a caso, con preferenza per quelle che hanno il dirty bit a 0.

Se:

- la frequenza di page fault è elevata e
- molto spesso non si trovano pagine non appartenenti al WS

allora per prevenire il thrashing è bene fare swap-out si un processo, diminuendo il livello di multiprogrammazione

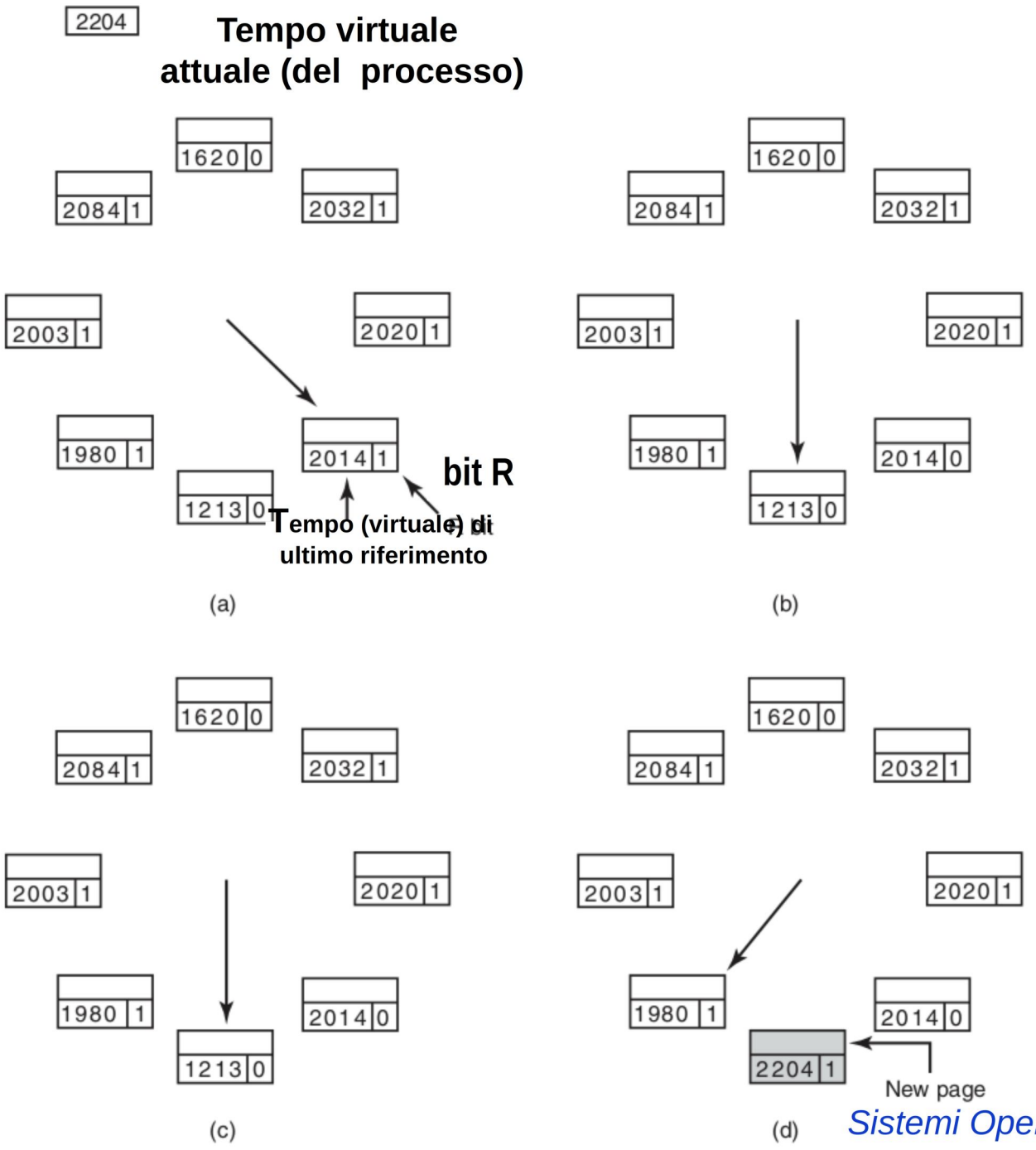
Si osservi che la scelta di $\tau$ influenza la dimensione del WS: un valore troppo grande potrebbe classificare come parte del WS molte pagine che invece non sono più utili, e viceversa

# Algoritmo dell'orologio modificato

Esempio con $\tau$ = 800

(a)-(b) La pagina con bit R = 1 viene risparmiata, e il campo "ultimo riferimento" aggiornato, mentre R viene azzerato.

(c)-(d) Quella successiva, con R=0 viene rimpiazzata perchè non appartiene al W.S. (2204-1213 = 991 > $\tau$)

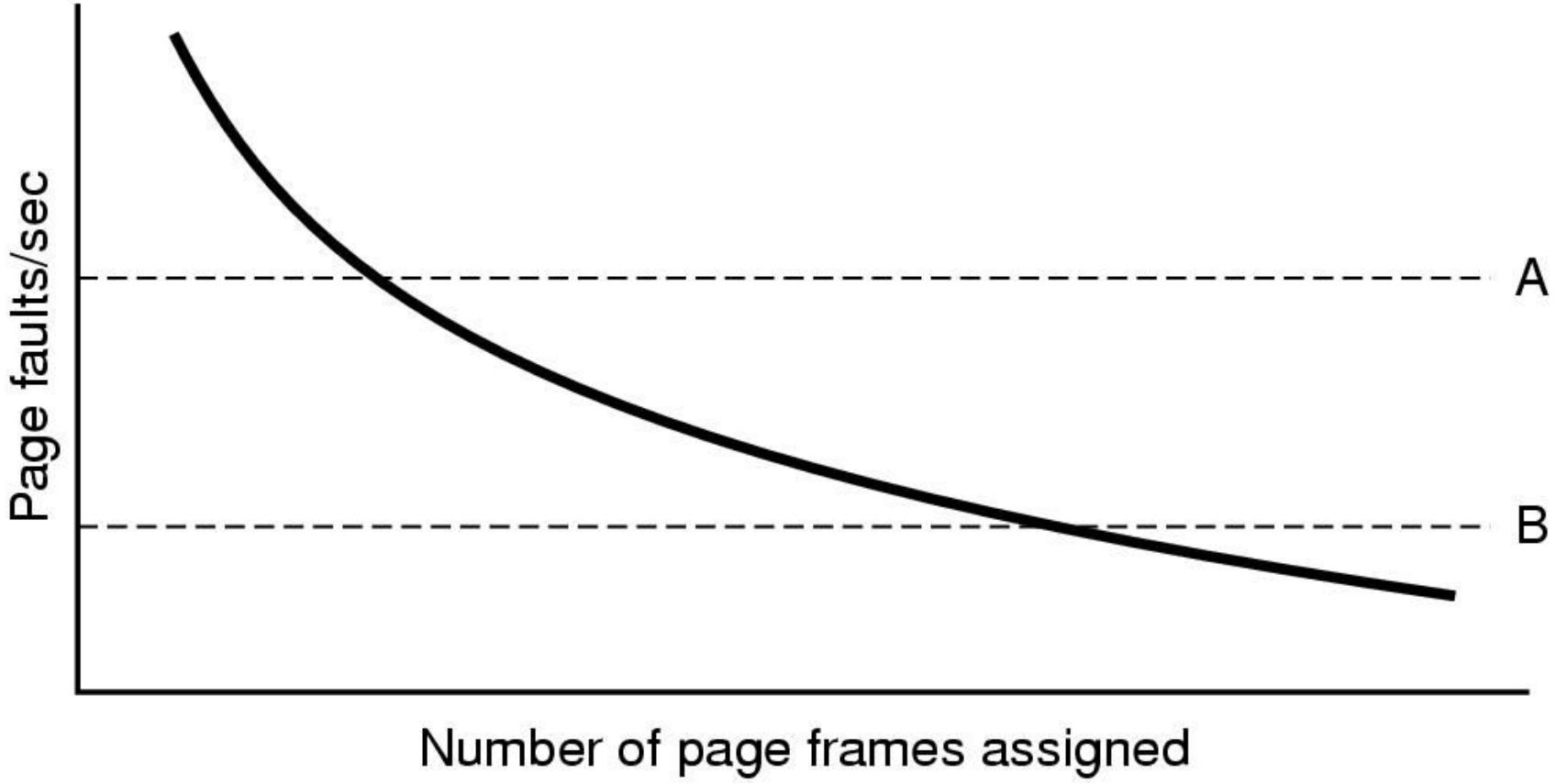


# Politiche locali e globali

Per una politica globale si può assegnare ad ogni processo un numero di frame variabile nel tempo, tenendo conto della frequenza di page fault:

se sale sopra A, è segno che il suo working set è più grande del numero di frame assegnati, per continuare a farlo girare bene bisogna assegnargliene altri; se non si può, si fa *swap out* in attesa di tempi migliori

se scende sotto B, gli si possono portare via dei frame



# Il "Page Daemon"

- Per assicurare che nel momento in cui serve avere un frame da rimpiazzare esista qualche frame "pulito" (dirty bit = 0) è utile mantenere un processo in background, chiamato "paging daemon" che periodicamente venga attivato per controllare la situazione dei frame in memoria

  (NB nella mitologia greca un "*daemon*" (δαίμων) è uno spirito benevolo; nei S.O. è un processo che svolge un servizio)

- Quando non ci sono frame disponibili (o ve ne sono troppo pochi) il daemon seleziona una pagina utilizzando lo stesso algoritmo di rimpiazzamento usato dal page fault handler, e se la pagina non è pulita richiede la sua scrittura su disco.

- Una pagina selezionata dal paging daemon è un candidato ideale per un rimpiazzamento, ma non viene "buttata via": in questo modo potrà essere recuperata nel caso sia riferita prima di essere effettivamente rimpiazzata.

# Pagine condivise

La condivisione di pagine dati è usata ad esempio per il codice (read-only, le pagine vengono etichettate come tali)

In Unix è usata anche quando un processo ne genera un altro con *fork:* tipicamente il nuovo processo fa *exec* poco dopo, allora conviene non fare la copia subito di tutta l'immagine del processo, ma solo per le pagine che uno dei due processi, dopo la *fork,* cerca di modificare ***(****copy-on-write****):***

- le pagine vengono etichettate read-only
- il tentativo di scrivervi causa una trap
- nella gestione di questa viene fatta la copia della pagina, in due pagine read-write, una per processo
- viene fatta ripartire l'istruzione che ha causato la trap

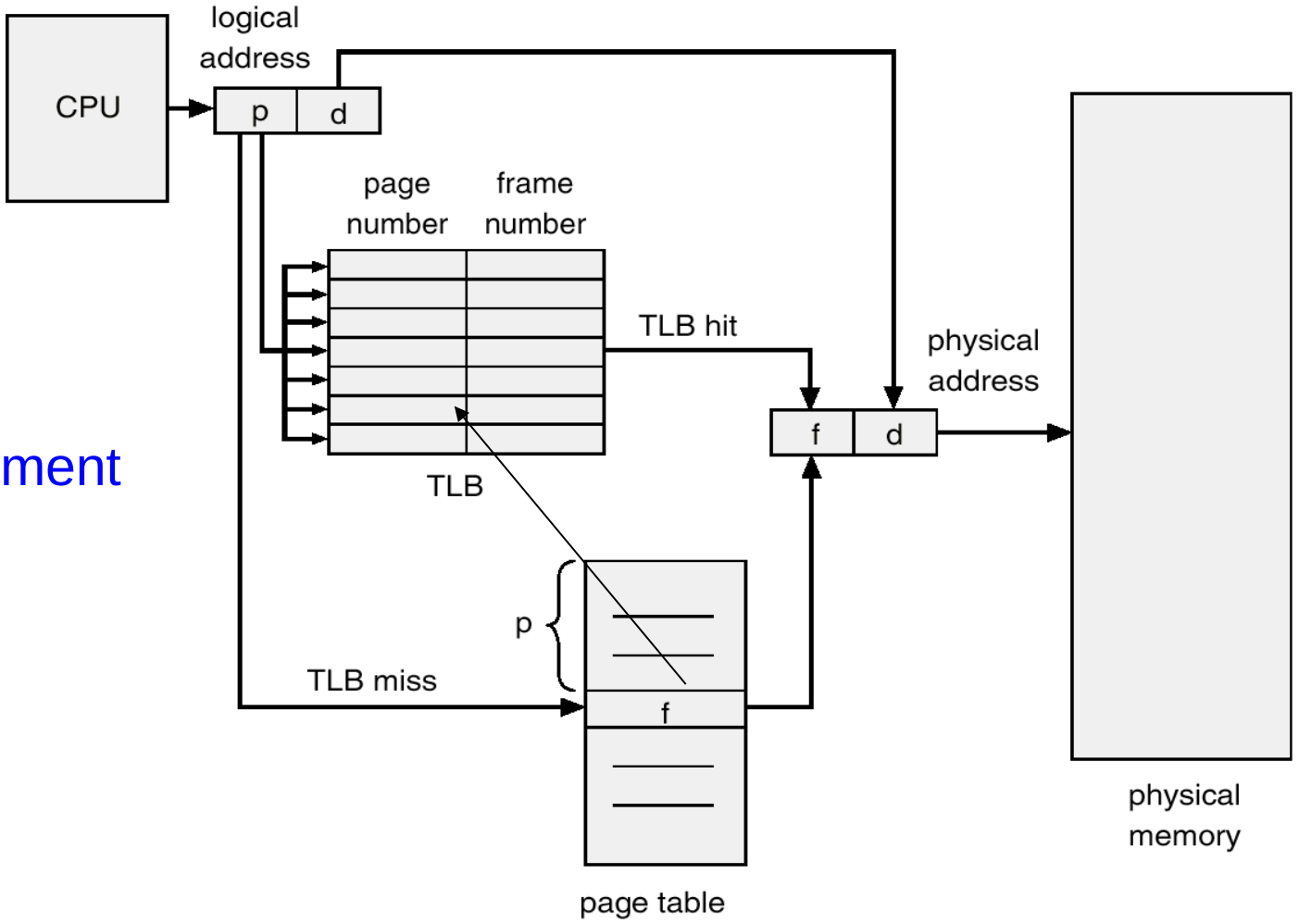
# Overhead

La paginazione introduce diversi tipi di *overhead* (costi di gestione):

- spazio:
  - per ogni processo occorre mantenere una tabella delle pagine che può essere molto grande (dipende dalla dimensione delle pagine).
  - si risolve la frammentazione esterna ma vi è *frammentazione interna* (si perde mediamente ½ pagina per ciascun processo, dato che la dimensione dei processi raramente è un multiplo della dimensione della pagina).
- tempo:
  - ad ogni context switch occorre aggiornare le informazioni utilizzate dalla MMU per la traduzione degli indirizzi
  - il trattamento dei page fault può far crescere di molto i tempi medi di accesso alla memoria

# Dove si trova la tabella delle pagine

Memory Management Unit




TLB = Translation Lookaside Buffer, è una memoria associativa: la ricerca di un numero di pagina viene fatta in parallelo su tutti gli elementi

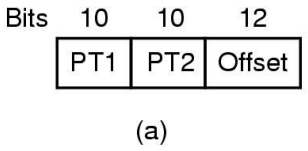
É una *cache* della tabella delle pagine: un elemento non trovato vi viene messo (al posto di un altro). Deve essere invalidata ad ogni context switch

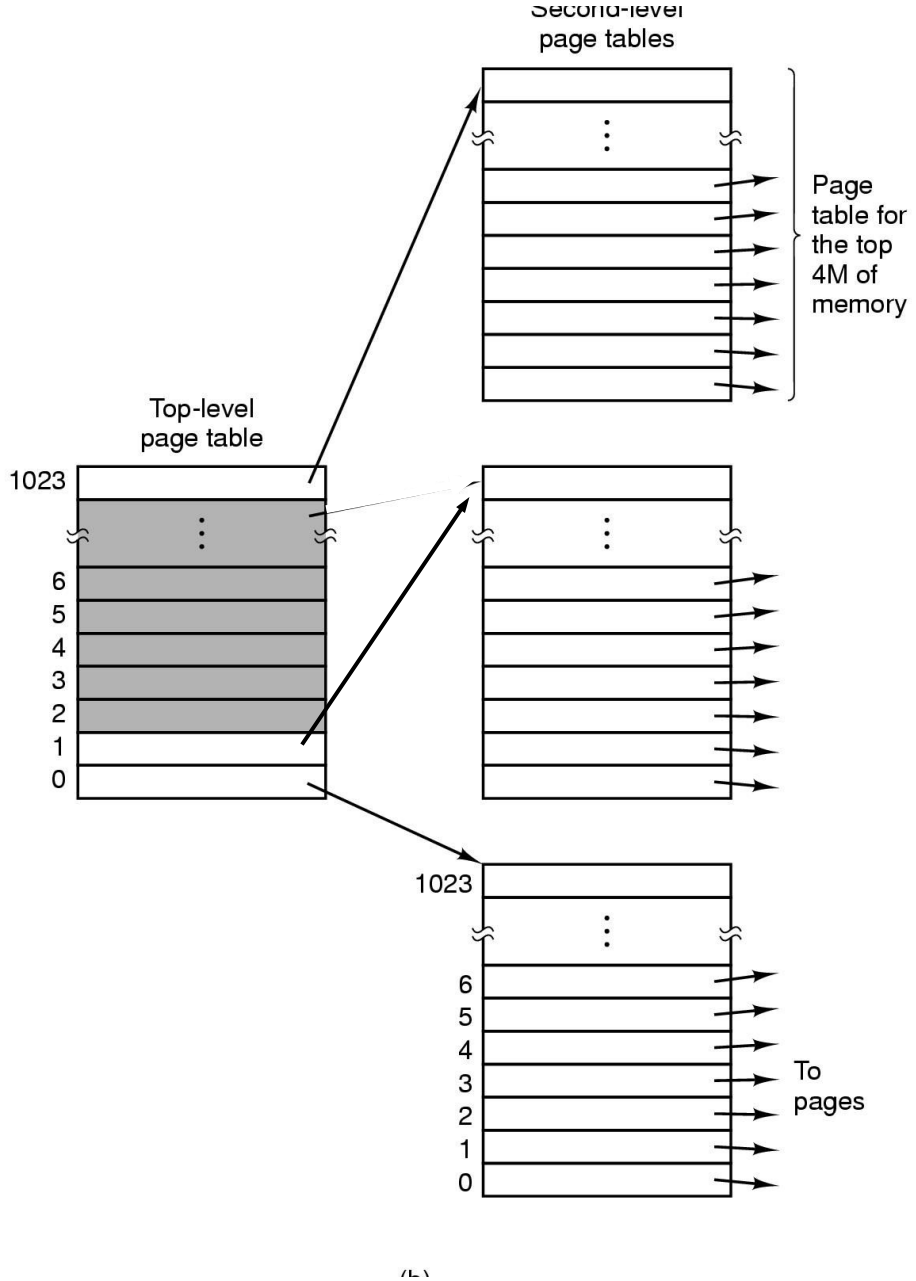
# Tabella delle pagine a due livelli

La tabella è a sua volta suddivisa in pagine

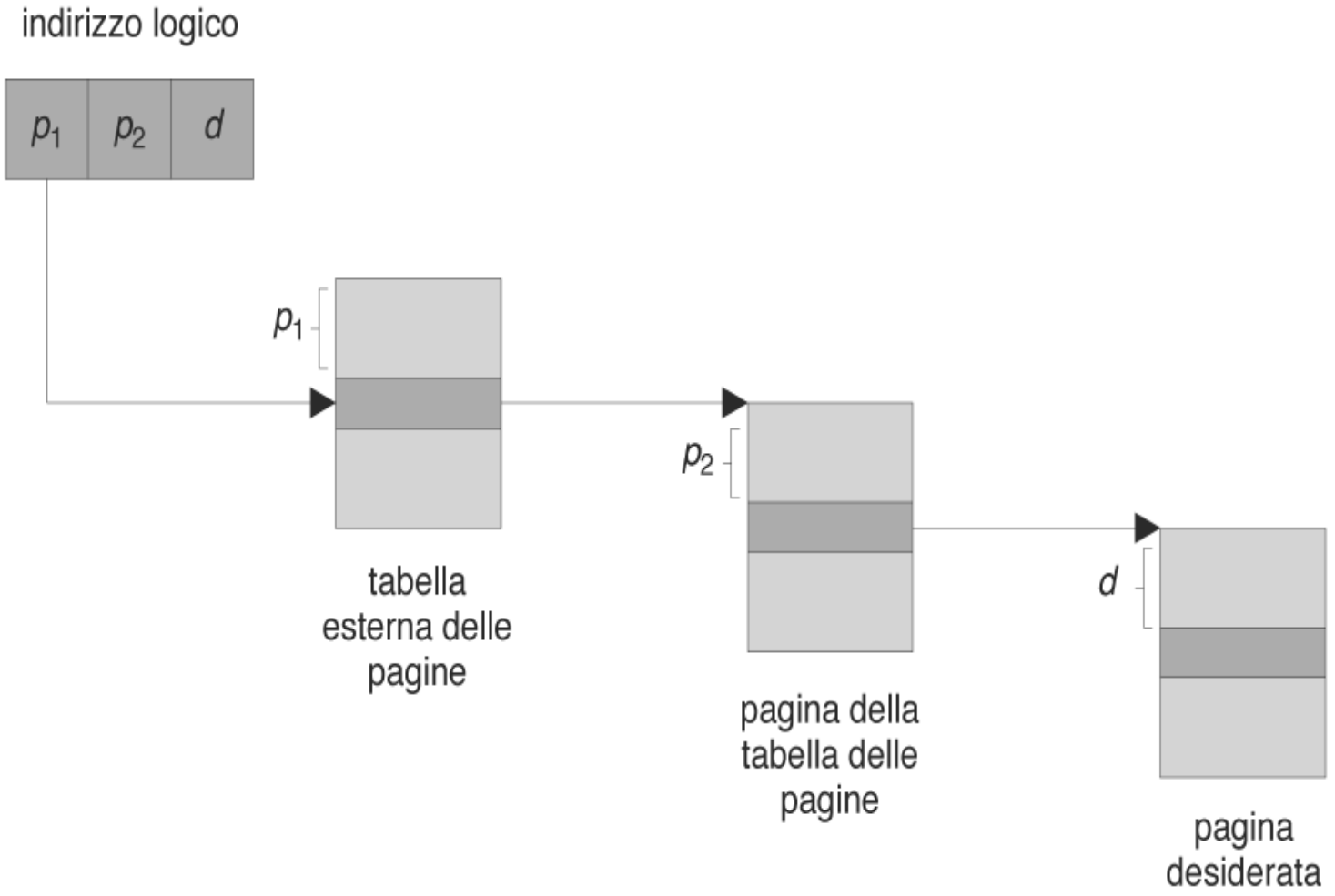
In un ampio spazio di indirizzi virtuali (4Gbyte indirizzi a 32 bit, di più con 64) programma e dati a un estremo, la stack all'altro, e un enorme spazio non usato in mezzo

⇒ non si allocano le "pagine" di indirizzi di pagina per lo spazio centrale



# Schema di traduzione degli indirizzi



# Quando interviene il SO nella gestione della memoria con paginazione

In quattro situazioni:

1. Creazione di un processo
   - deve determinare la dimensione del processo, allocare spazio per tale processo in backing store, creare la page table
2. Cambiamento di stato di un processo Ready ⇒ Running
   - deve reimpostare la MMU per il nuovo processo, il TLB deve essere invalidato
3. Quando si verifica un page fault
   - deve determinare la pagina richiesta, scegliere una vittima da rimpiazzare, se necessario, fare swap-out di tale pagina, fare swap in di quella richiesta.
4. Terminazione di un processo
   - rilasciare page table, e i frame occupati dalle pagine del processo