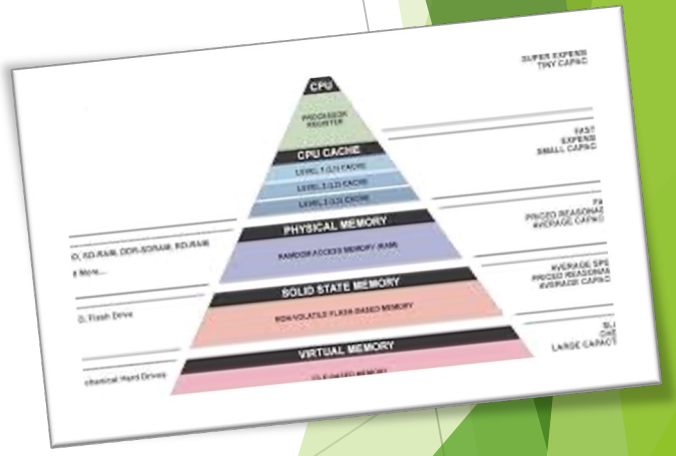
C.d.L. in Informatica
(laurea triennale)
A.A. 2023-2024

Sistemi Operativi (M-Z)

Gestione della Memoria

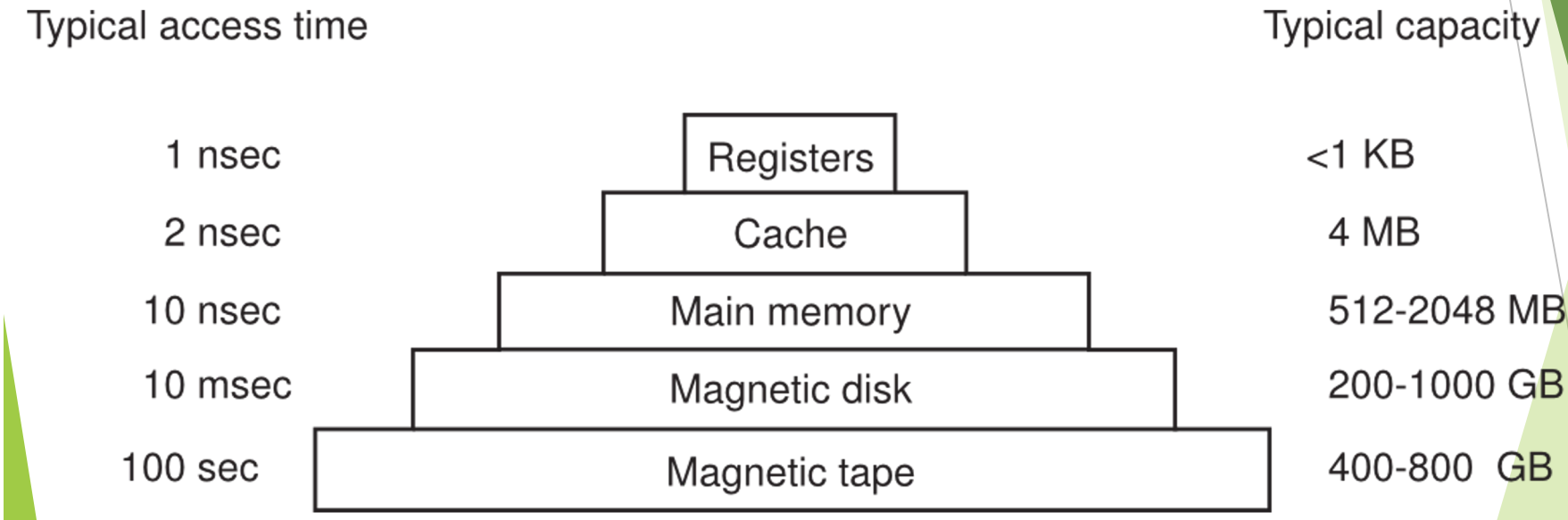


Prof. Mario F. Pavone

Dipartimento di Matematica e Informatica
Università degli Studi di Catania
mario.pavone@unict.it
mpavone@dmi.unict.it
http://www.dmi.unict.it/mpavone/so.html

Memoria centrale e processi

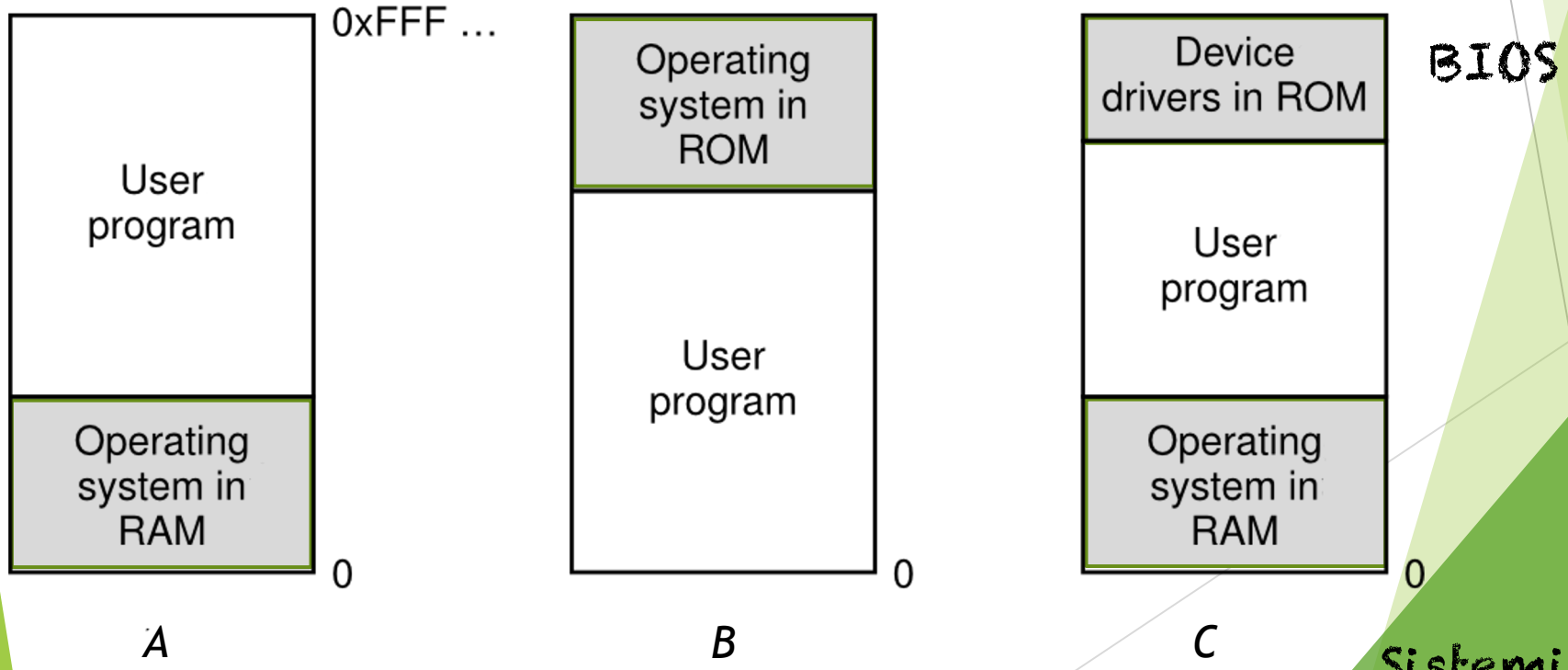
- **Gerarchia di memoria**
  - in particolare la memoria centrale (formata da **RAM**) rappresenta il livello più basso **direttamente utilizzabile dalla CPU**



- **astrazione della memoria a favore dei processi**
  - vari livelli di astrazione (via via più complessi)

Senza alcuna astrazione

- primi mainframe (anni '60) e primi PC (primi anni '80)
- i programmi utilizzano direttamente gli **indirizzi fisici**
- difficile eseguire due programmi contemporaneamente
- vari **modelli**:



Multiprogrammazione senza astrazione

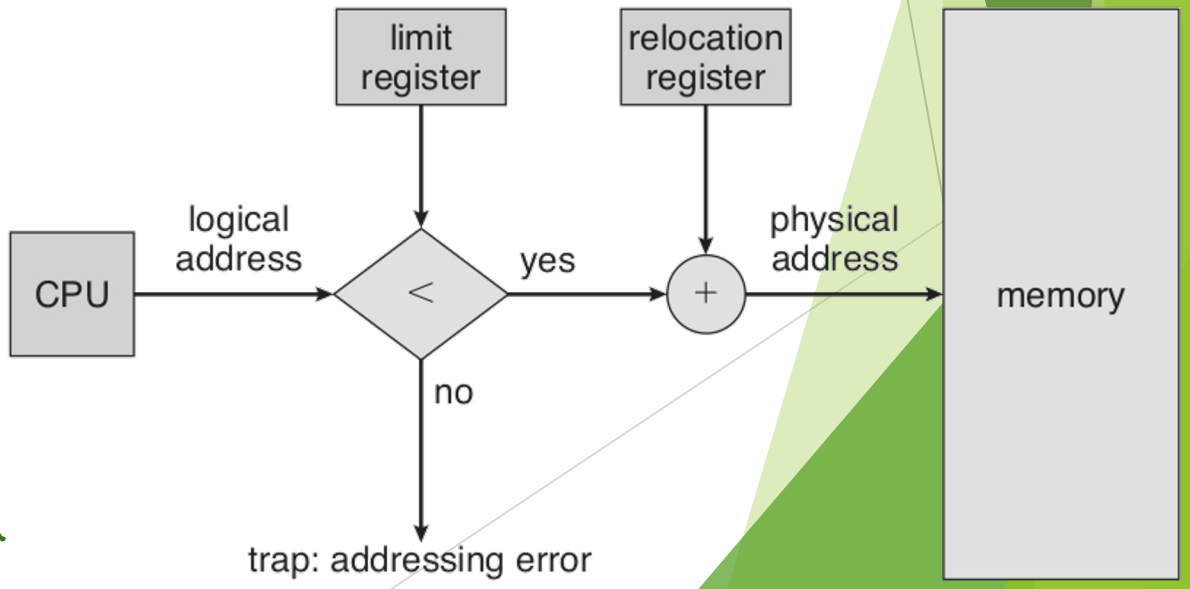
- Manteniamo **più processi in memoria**

- **protezione della memoria**:
  - **lock & key**: blocchi di memoria con delle chiavi di protezione e PSW con la chiave del processo in esecuzione.

- **rilocazione**:
  - **rilocazione statica** in fase di caricamento;
    - rallentamento del loader
  - **rilocazione a compile-time** in fase di compilazione

Spazio degli indirizzi

- **Spazio degli indirizzi**: insieme di indirizzi per indirizzare la memoria
  - astrazione per la memoria utilizzabile da un processo;
- Ogni processo ha un proprio spazio degli indirizzi
- **rilocazione dinamica** con **registro base** e **registro limite**
  - la **CPU controlla** gli accessi alla memoria in base ai registri;
    - nei sistemi più evoluti questo è fatto dalla Memory Management Unit (MMU);
  - i registri vanno aggiornati ad ogni context-switch;
  - usata su:
    - CDC 6600;
    - Intel 8088.
  - **Svantaggio**:
    - **Confronto & Somma**

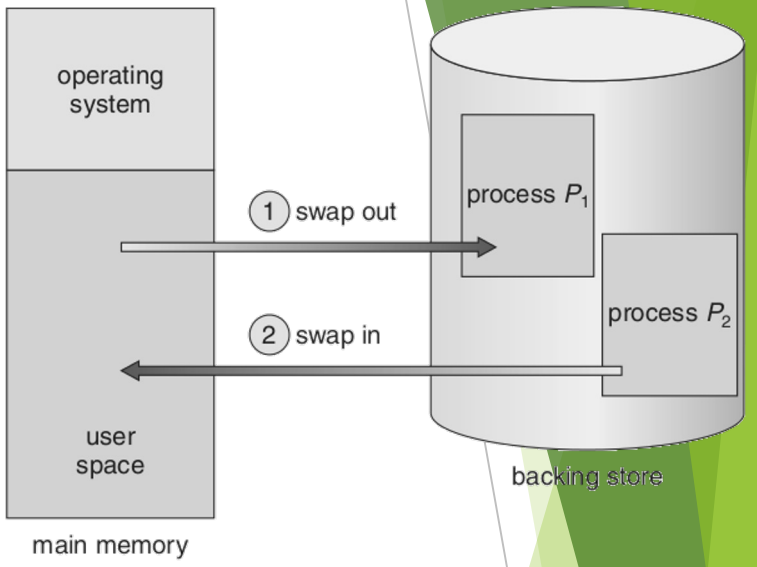


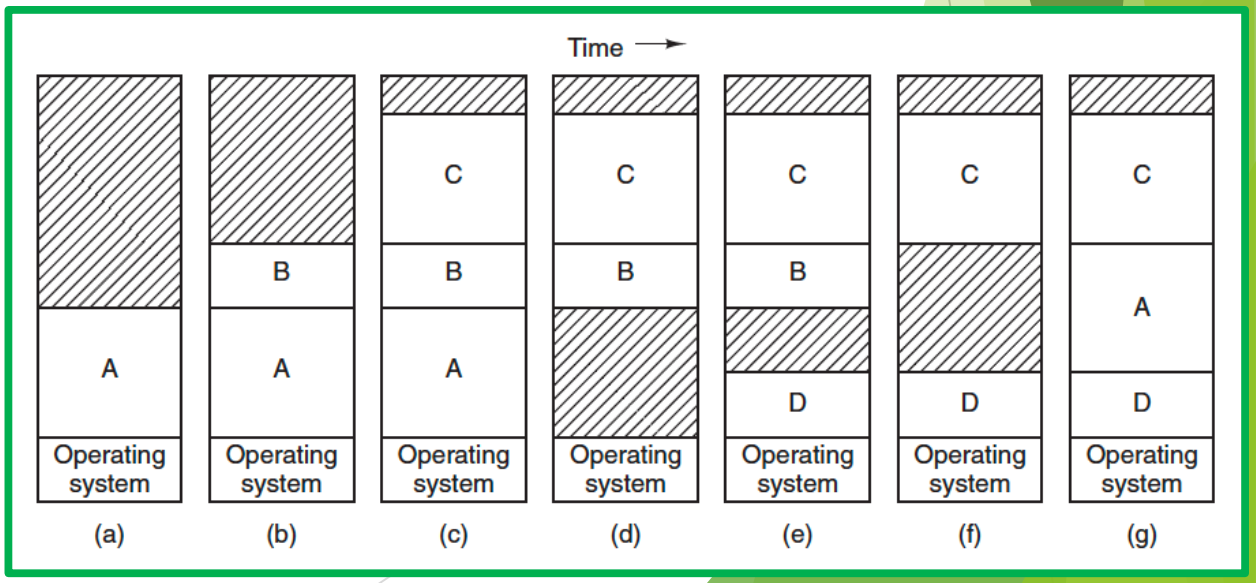
Sovraccarico Memoria

- Il **livello di multiprogrammazione** è seriamente **limitato dalla dimensione della memoria centrale**;
- Diversi programmi applicativi possono occupare molta memoria solo in fase di avvio + elaborazione
- **Gestione sovraccarico** => 2 approcci:
- Swapping
  - considerare il processo nella totalità => eseguirlo => riportarlo sul disco
    - processi inattivi => archiviati sul disco
- Memoria Virtuale
  - eseguire i programmi anche se parzialmente in memoria principale

Swapping

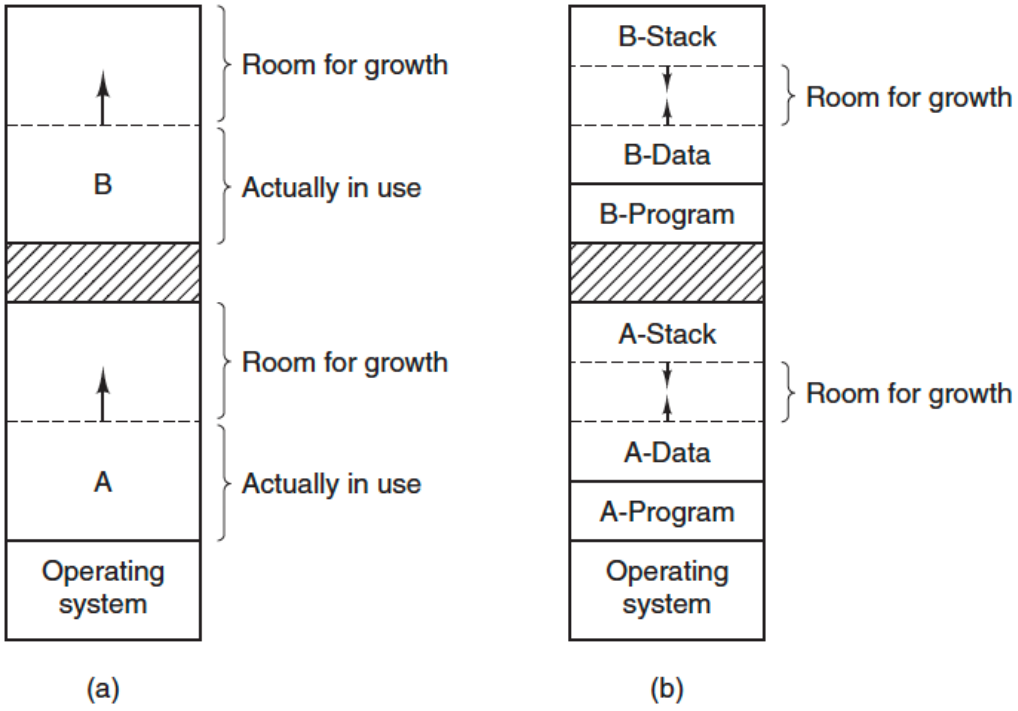
- prima soluzione: swapping;
  - **swapper** (scheduler di medio termine);
  - problemi con **I/O pendenti**;
- **frammentazione:**
  - **interna**;
  - **esterna**;
- **memory compaction**





Swapping

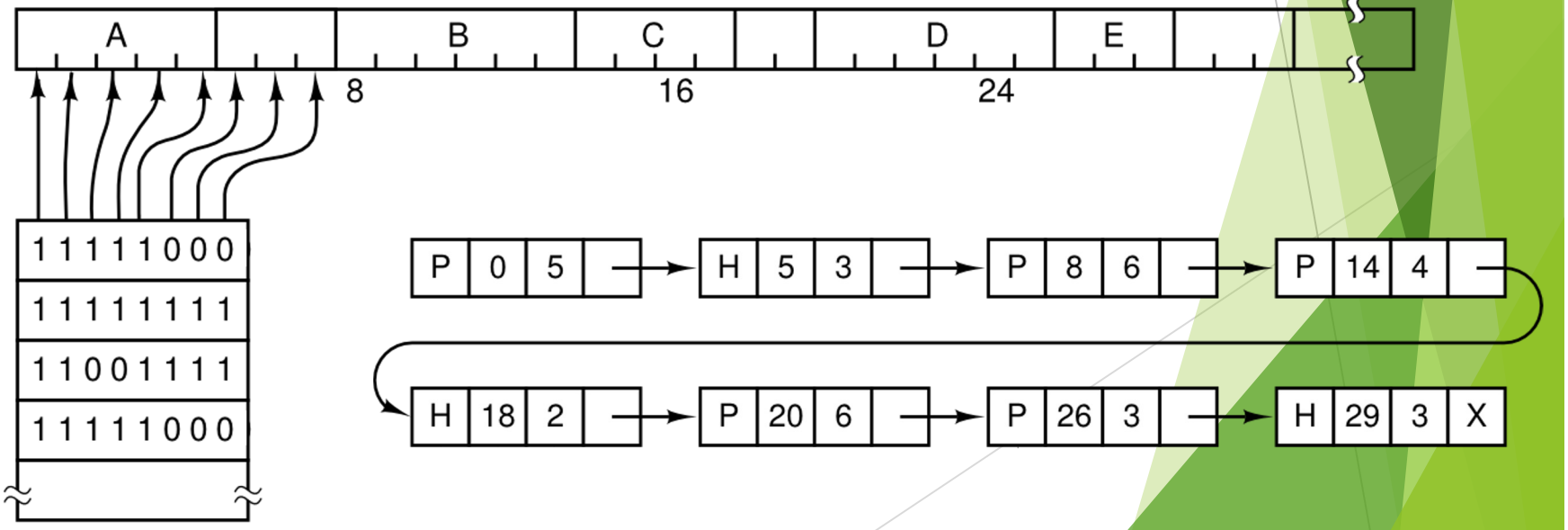
- Quanta memoria allocare?
- strategie di allocazione:
  - dimensione fissa;
  - dimensione dinamica;



La maggior parte dei processi cresce durante la loro esecuzione

Gestione dell'allocazione

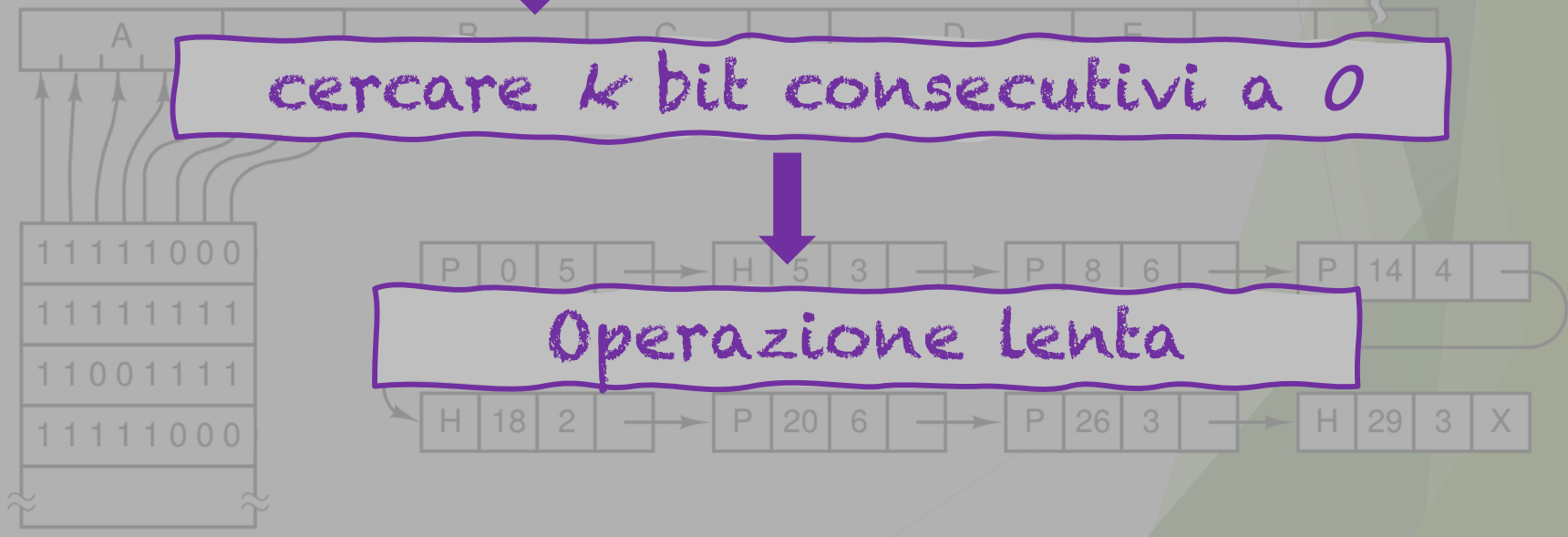
- Assegnazione dinamica => tener traccia dell'utilizzo della memoria
- **Bitmap**
  - Ad ogni unità di memoria corrisponde un bit
  - Dimensione unità di allocazione
    - dimensione del blocco => grandezza bitmap



Gestione dell'allocazione

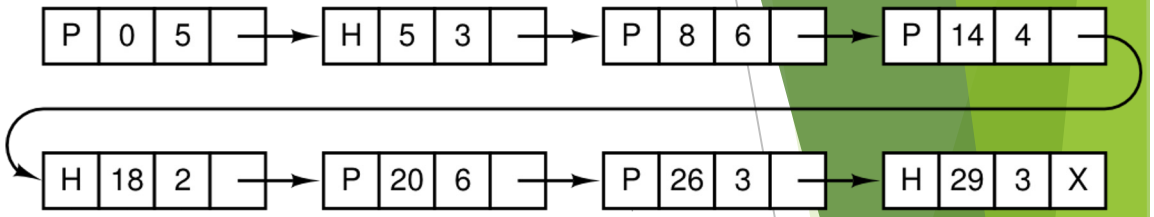
- Assegnazione dinamica => tener traccia dell'utilizzo
  - SVANTAGGIO: RICERCA
- Bitmap
  - Ad
  - Di
    - dimensione del blocco => grandezza bitmap

Allocare una quantità fissa di k unità in memoria per un processo



Gestione dell'Allocazione

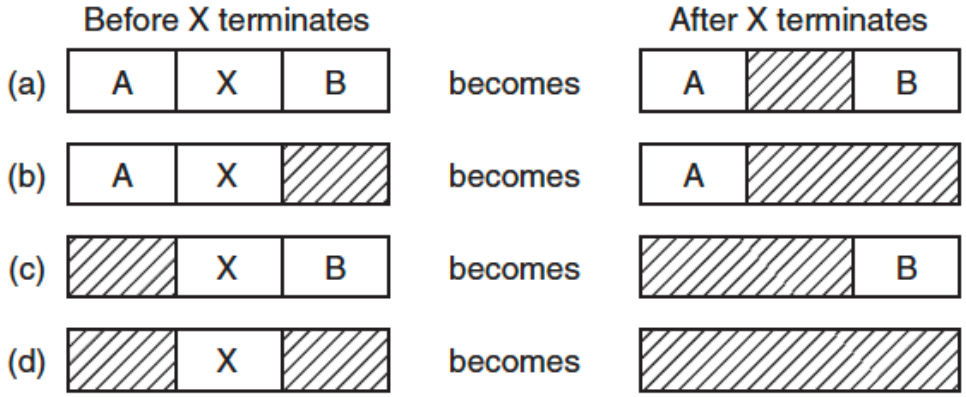
- **Liste dei blocchi liberi e occupati**;
- ordinata per indirizzo;
  - **coalescenza**;
- doppiamente collegata;



- **algoritmi di ricerca in allocazione**:
  - **first fit**, **next fit**, **best fit**, **worst fit**;
- **liste separate**;
  - velocizzare i 4 algs.
  - **lista blocchi liberi ordinata per dimensione**

first fit vs. best fit

best fit più veloce



Memoria Virtuale

- Cosa fare se un programma richiede più memoria di quella fisicamente disponibile?
- **overlay**: gestore di overlay, suddivisione da parte del programmatore;
- Si passa all'**allocazione non contigua** della memoria fisica;

Memoria Virtuale

- Si passa all'**allocazione non contigua** della memoria fisica;
  - spazio di indirizzamento virtuale diviso in **pagine**;
  - spazio della memoria fisica diviso in **frame**;
  - **dimensione** delle pagine/frame:
    - frammentazione interna;
    - in alcuni casi è variabile (Solaris);
  - alcune pagine possono anche non risiedere in memoria;
    - all'occorrenza vengono caricate dalla **memoria secondaria**;
    - si adatta perfettamente ad un **sistema multiprogrammato**;
  - **protezione implicita** tra processi.

Paginazione

- **Indirizzi virtuali**: generati dal programma:
- **Spazio Virtuale degli Indirizzi**
- Gestita dalla **MMU**;
  - **Mappa gli indirizzi virtuali sugli indirizzi della memoria fisica**
  - **bit di presenza**;
  - errore di pagina (**page fault**).

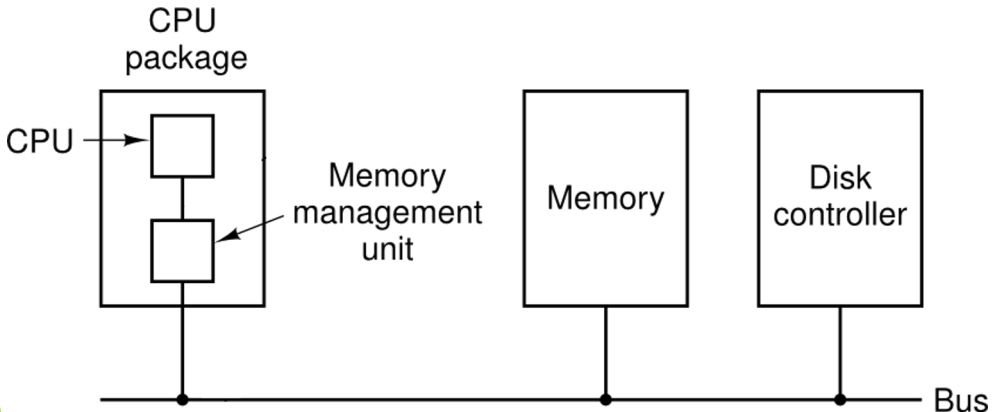


figura 1

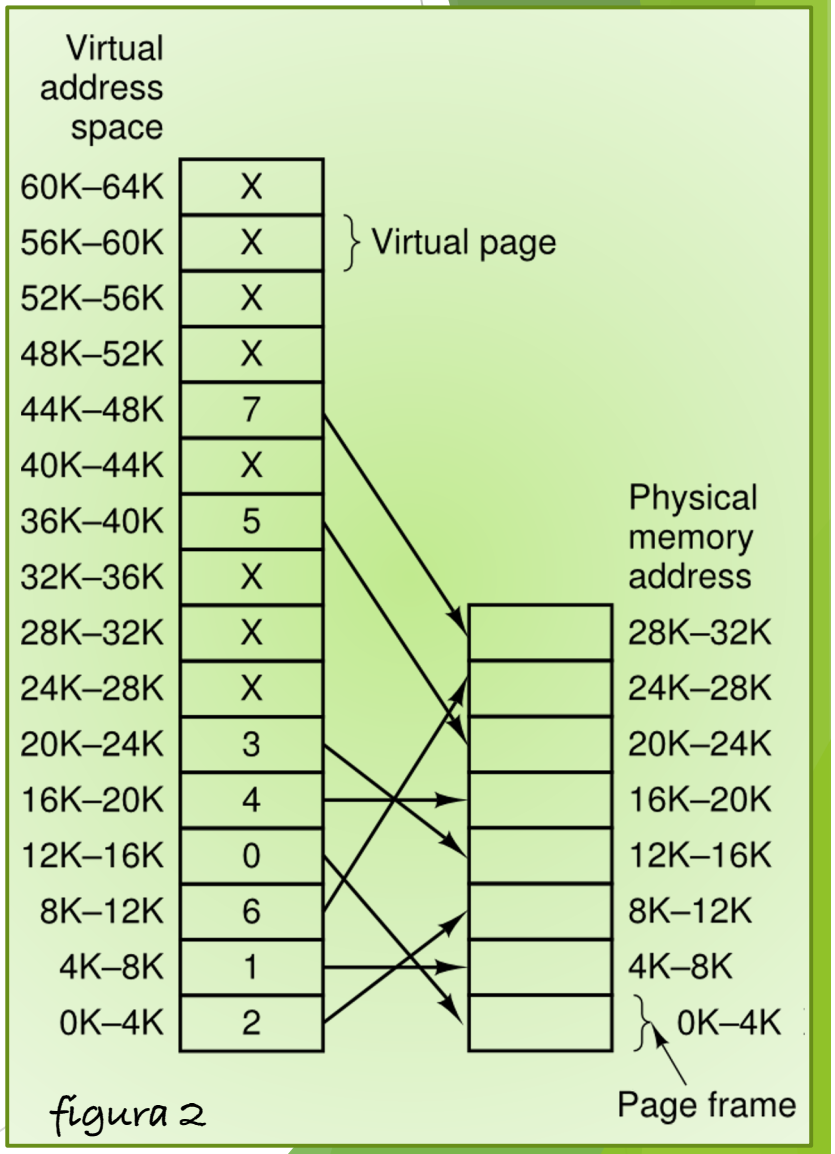


figura 2

Paginazione

- **Indirizzi virtuali**: generati dal programma:
- **Spazio Virtuale degli Indirizzi**
- Gestita dalla **MMU**;
  - Mappa gli indirizzi virtuali sugli indirizzi della memoria fisica
  - bit di presenza;
- errore di pagina (**page fault**).

Cosa succede se eseguiamo l'istruzione:

MOV REG 32780

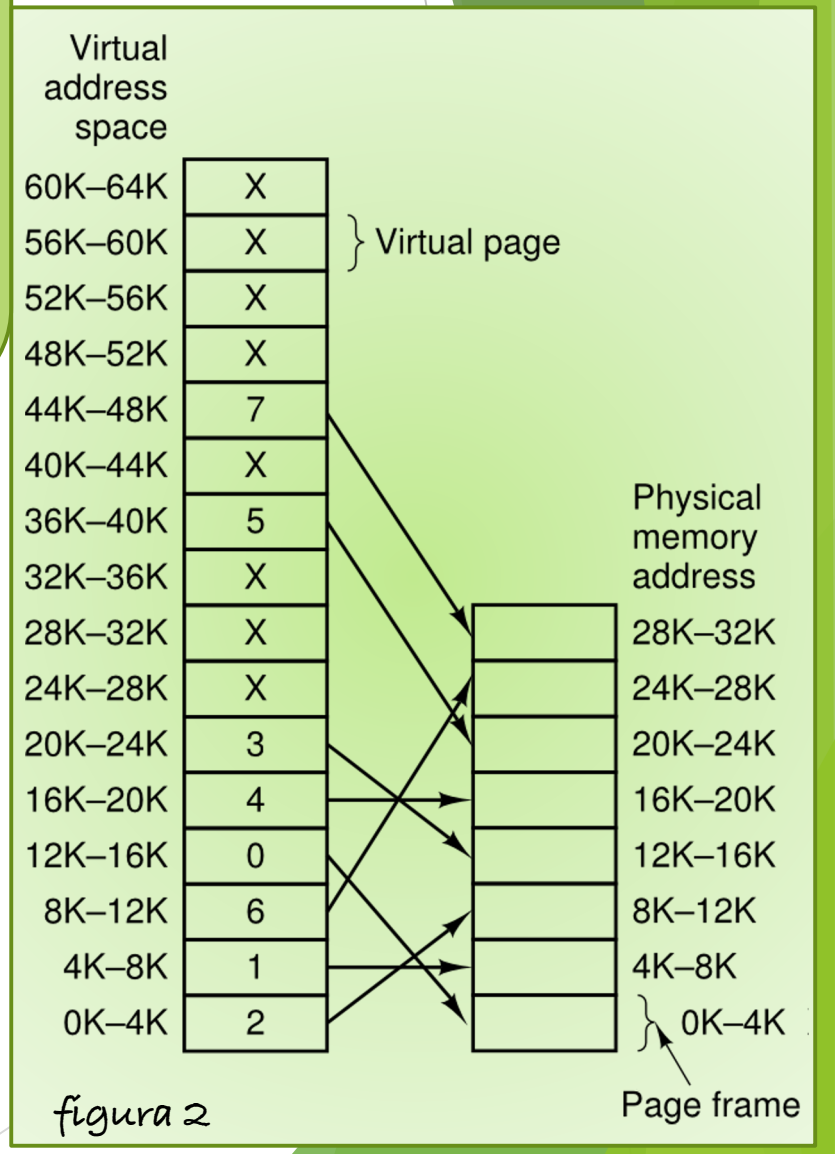
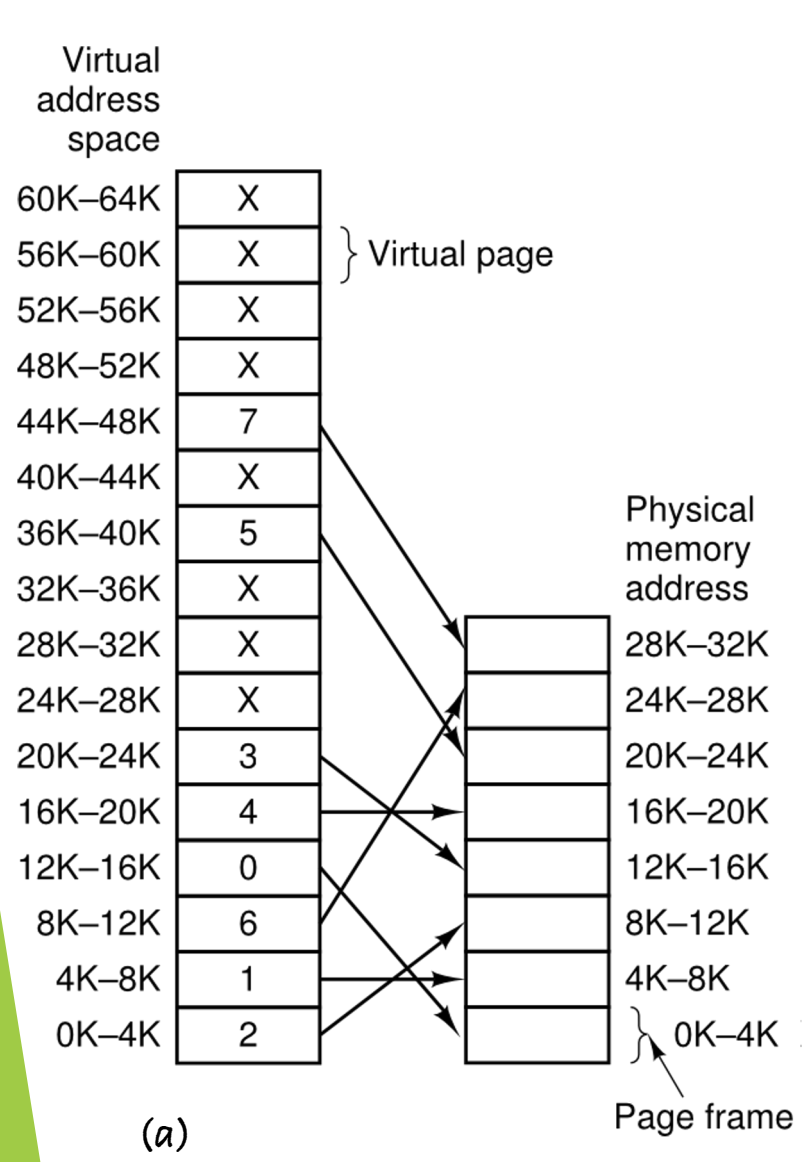


figura 2

Tabella delle Pagine



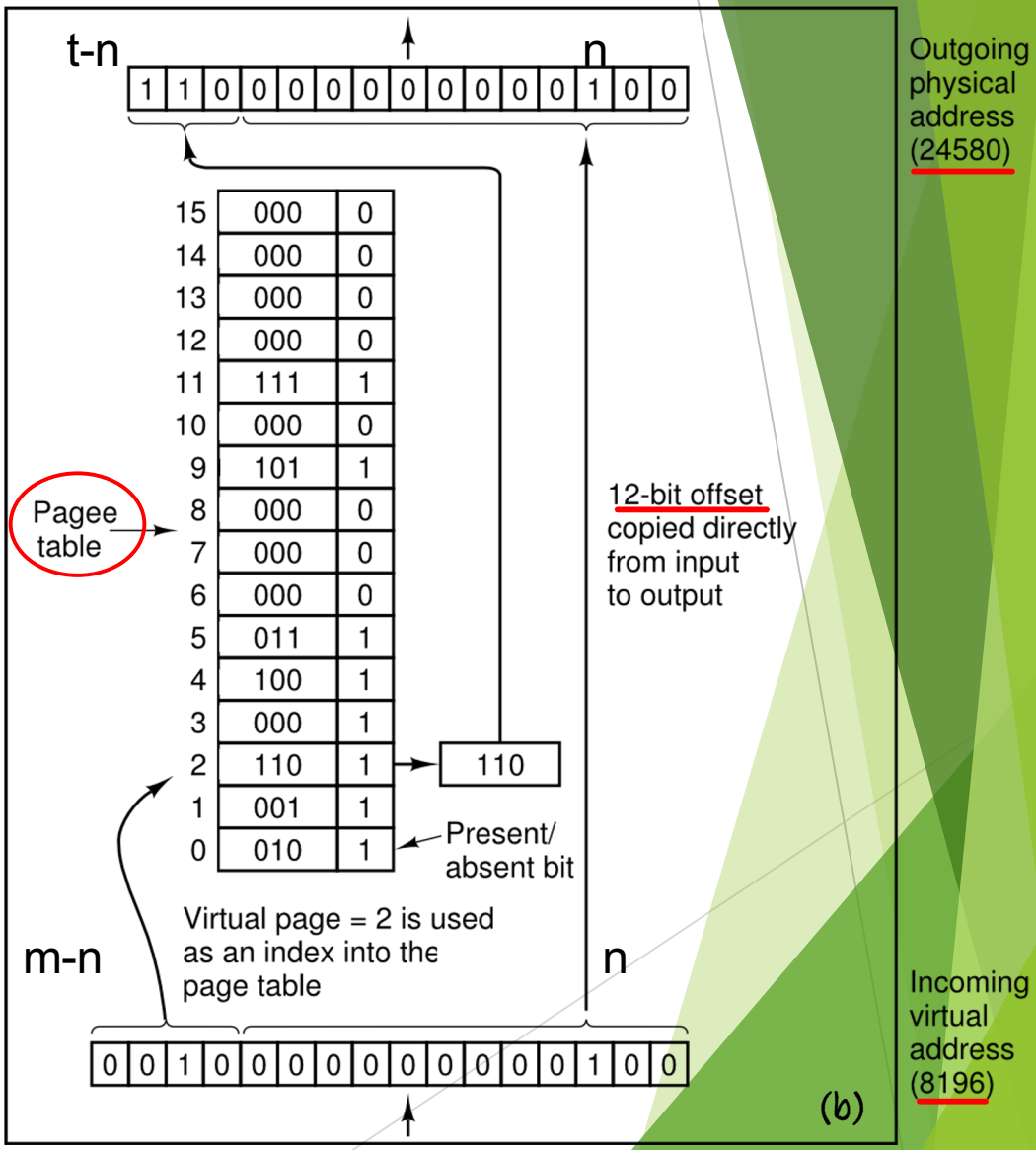


Tabella delle Pagine

- spazio indirizzi virtuali: 2^m
- dimensione pagina: 2^n

- numero di pagina: $(m-n)$
  - bit più significativi dell'indirizzo virtuale
- 2^{m-n} pagine
- offset: n bit meno significativi

- spazio indirizzi fisici: 2^t

→ 2^{t-n} frame
→ $m > t$

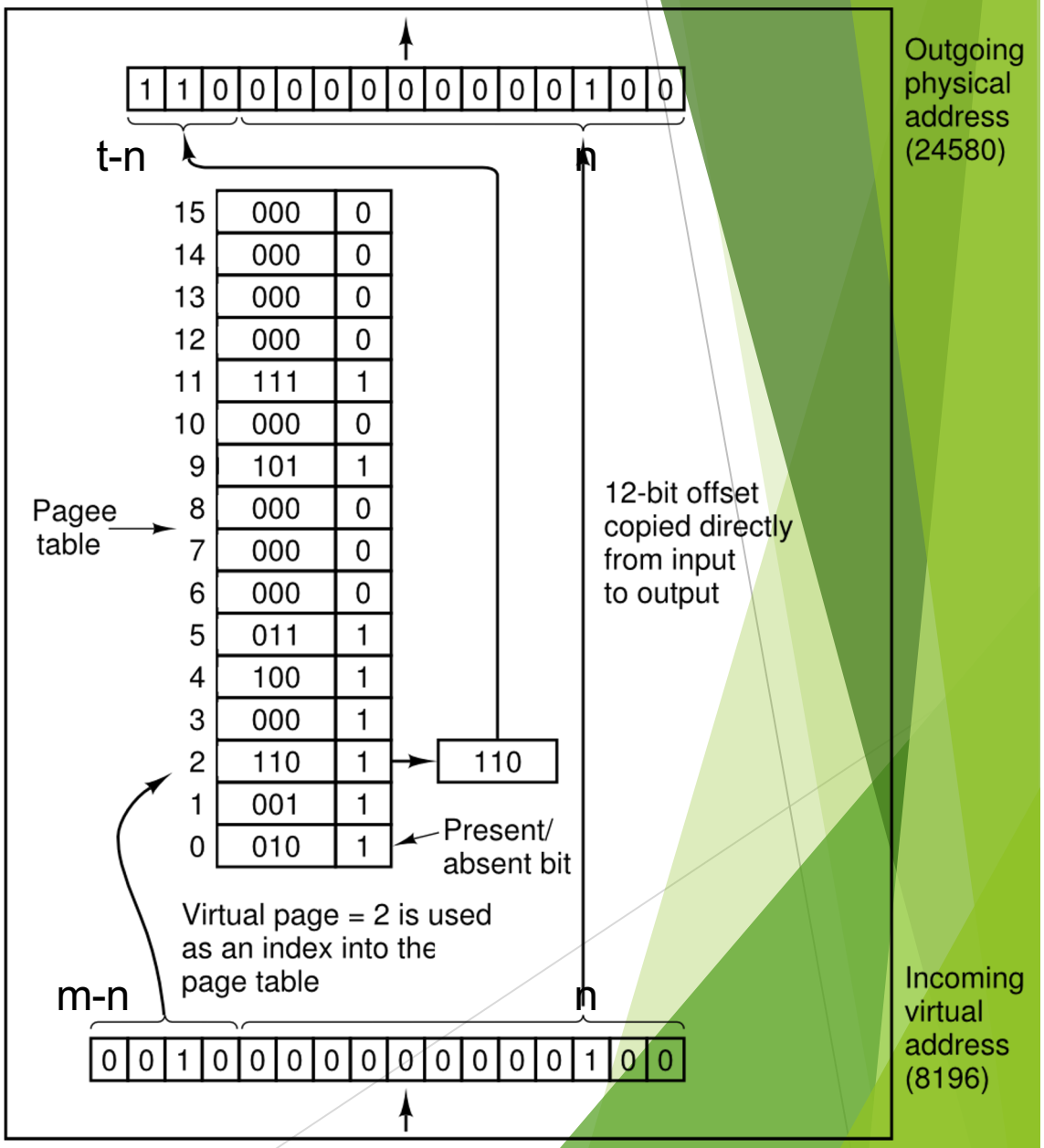
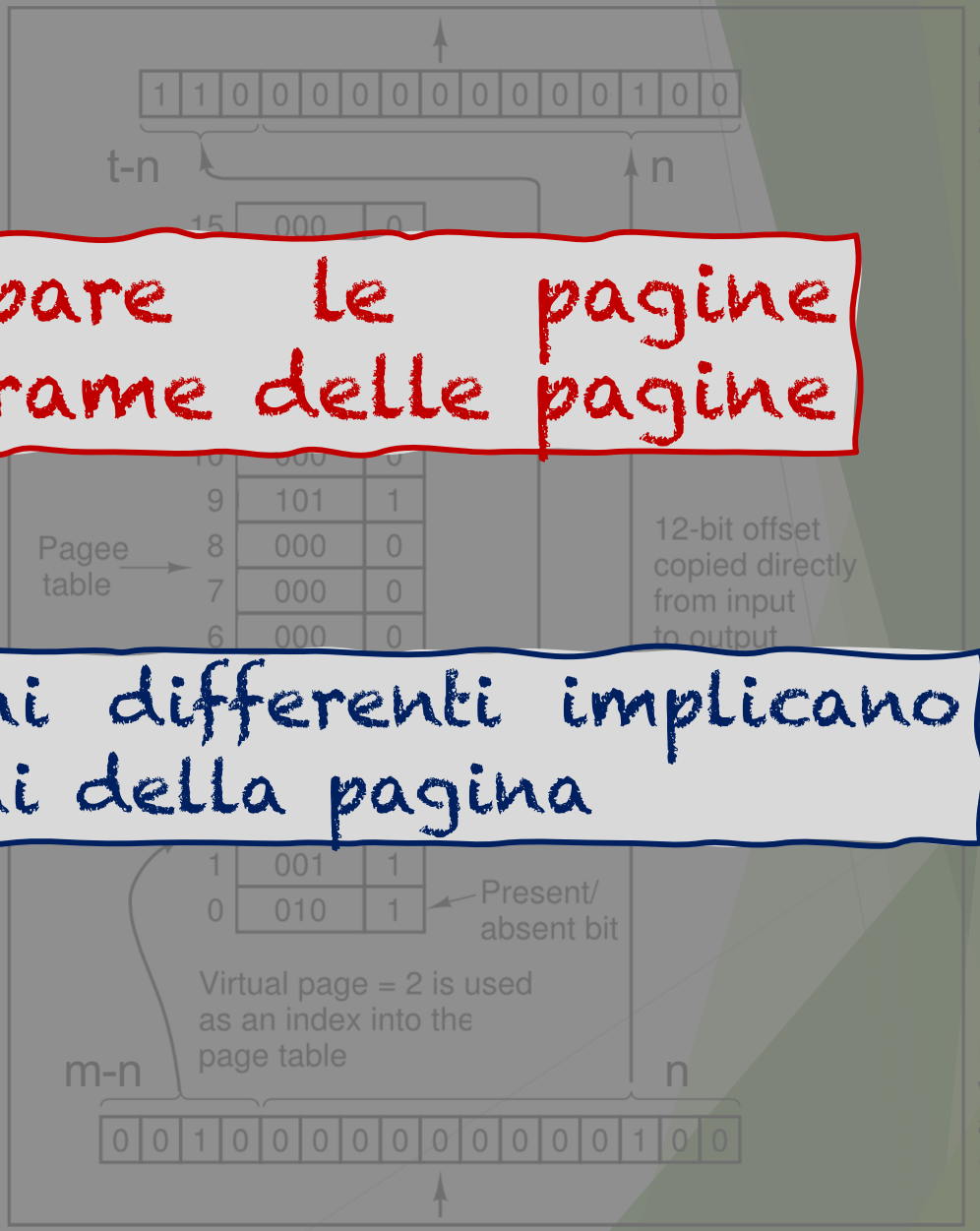


Tabella delle Pagine

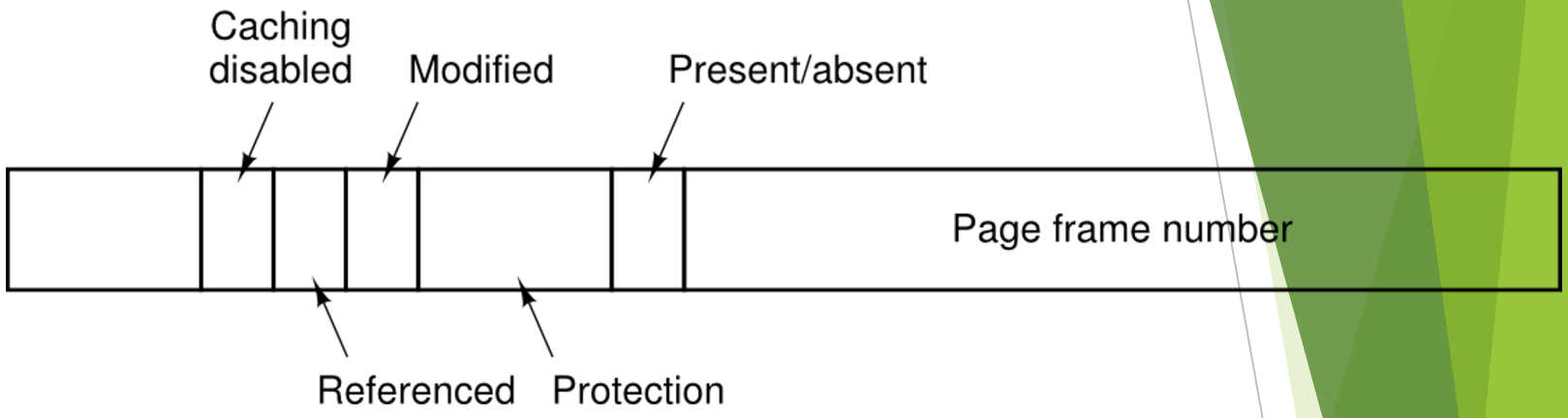
- spazio indirizzi virtuali: 2^m
- dimensione pagina: 2^n
- numero pagine
  - bit più significativi dell'indirizzo virtuale
- 2^{m-n} pagine
- spazio indirizzi fisici: 2^t
- $\rightarrow 2^{t-n}$ frame
- $\rightarrow m > t$

GOAL: Mappare le pagine virtuali sui frame delle pagine

NOTA: Suddivisioni differenti implicano diverse dimensioni della pagina



Dettaglio su una voce della tabella delle pagine



- **Numero del frame**;
- **bit presente/assente**;
- **protezione**: lettura/scrittura ed eventualmente esecuzione;
- **bit modificato** (**dirty bit**): indica se la pagina è stata modificata;
- **bit referenziato**: impostato quando si fa un riferimento qualunque alla pagina;
- bit per **disabilitare la cache**;
- nella pratica esiste anche un **bit di validità** (o di allocazione).

Tabella dei Frame

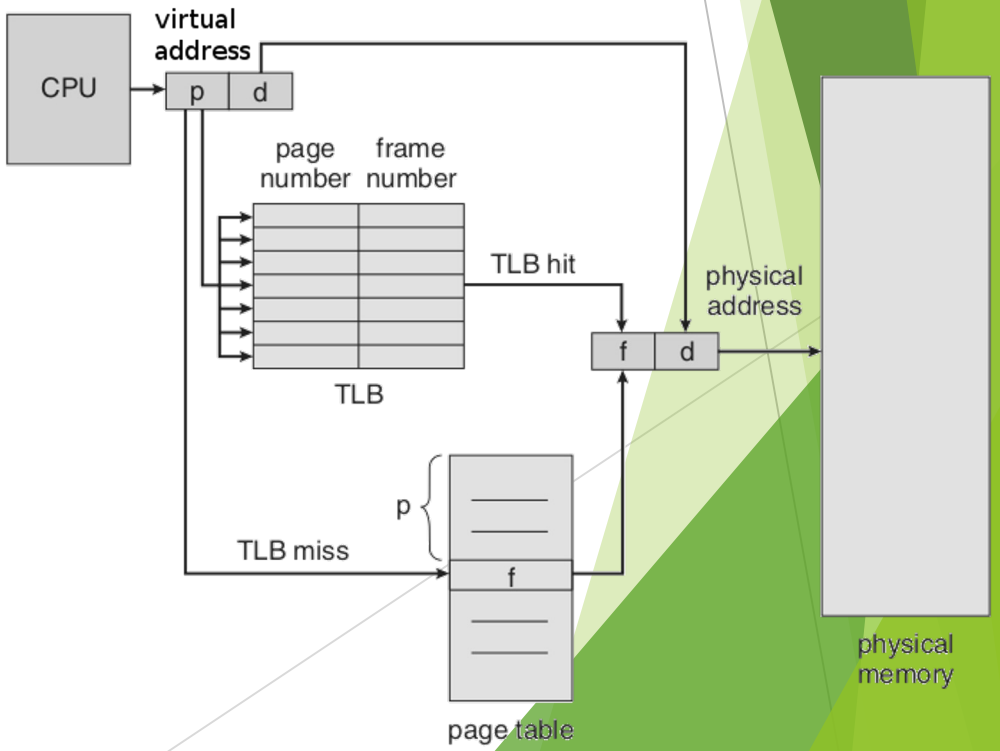
- Il S.O. tiene **traccia dello stato di occupazione di ogni frame** fisico attraverso la **tabella dei frame**;
  - stato: **occupato** / **libero**;
  - se occupato: **da quale processo?**
- viene consultata:
  - ogni volta che viene creato un **nuovo processo** per creare la relativa tabella delle pagine di quel processo;
  - ogni volta che un processo chiede di allocare **nuove pagine**.

Progettazione di una Tabella delle Pagine

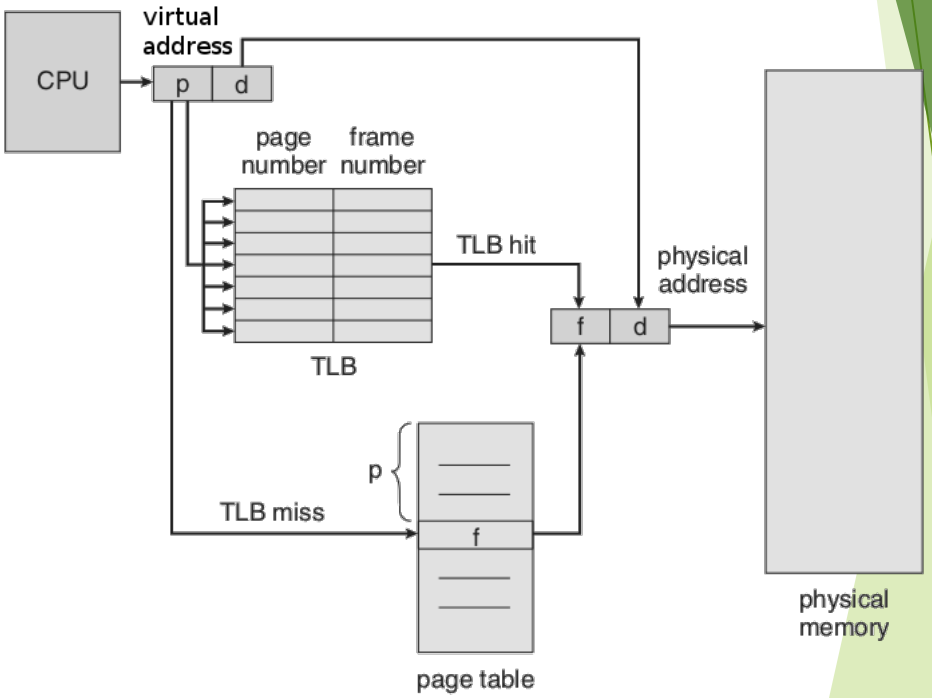
- Sono due gli **aspetti principali** da curare:
  - **velocità** nella consultazione;
  - **dimensione**.
- Affrontiamo per ora il fattore velocità:
  - due possibili idee di implementazione:
    - avere un **numero sufficiente di registri** su cui caricare l'intera tabella;
    - tabella interamente residente in memoria con registro PTBR (Page-Table Base Register);
      - servono **due accessi** alla memoria per prelevare un dato dalla memoria;
      - **context switch molto veloce**.

Uso di Memoria Associativa

- **Translation Lookaside Buffer** (**TLB**) dentro la MMU– **Memoria associativa**;
- osservazione: ***un programma usa un gran numero di riferimenti ad un piccolo numero di pagine***;
- un numero ridotto di registri (tra 64 e 1024), con le seguenti **voci**:
  - numero di pagina virtuale;
  - codice di protezione;
  - dirty bit;
  - bit per validità della voce della TLB;
  - numero di frame.



Uso di Memoria Associativa



- **ricerca parallelizzata** in hardware;
- **TLB miss** *vs.* **TLB hit**;
- possibilità di **voci vincolate**;
- uso di **address-space identifiers** (**ASID**) *vs.* **flush** della TLB.

(invalidare la TLB)

Effective Access Time (EAT)

- Facciamo un **esempio**:
  - tempo di accesso alla memoria = 100 nsec;
  - tempo di accesso alla TLB = 20 nsec;
- **tempo effettivo di accesso** sarà in questo caso:
  - 120 nsec per TLB hit;
  - 220 nsec per TLB miss;
- ipotizziamo un **TLB ratio** (percentuale di successi) dell'80%;
  - *tempo (medio) effettivo di accesso:*

$$0.8 \times 120 + 0.2 \times 220 = 140 \text{ nsec}$$

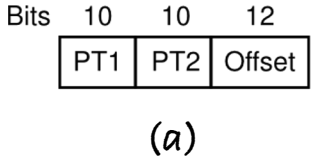
Effective Access Time (EAT)

- *Hit ratio* dell'80%;
  - *tempo (medio) effettivo di accesso:*

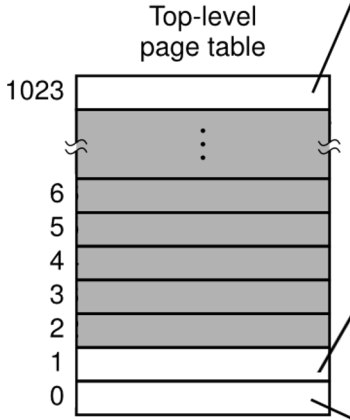
    0.8 x 120 + 0.2 x 220 = 140 nsec
- cosa succede se **hit ratio** è **60%** o **98%**?
- in generale:
  - **tempo di accesso alla memoria**: α
  - **tempo di accesso alla TLB**: β
  - **TLB/hit ratio**: ε
    - $EAT = \varepsilon (\alpha + \beta) + (1 - \varepsilon)(2\alpha + \beta)$

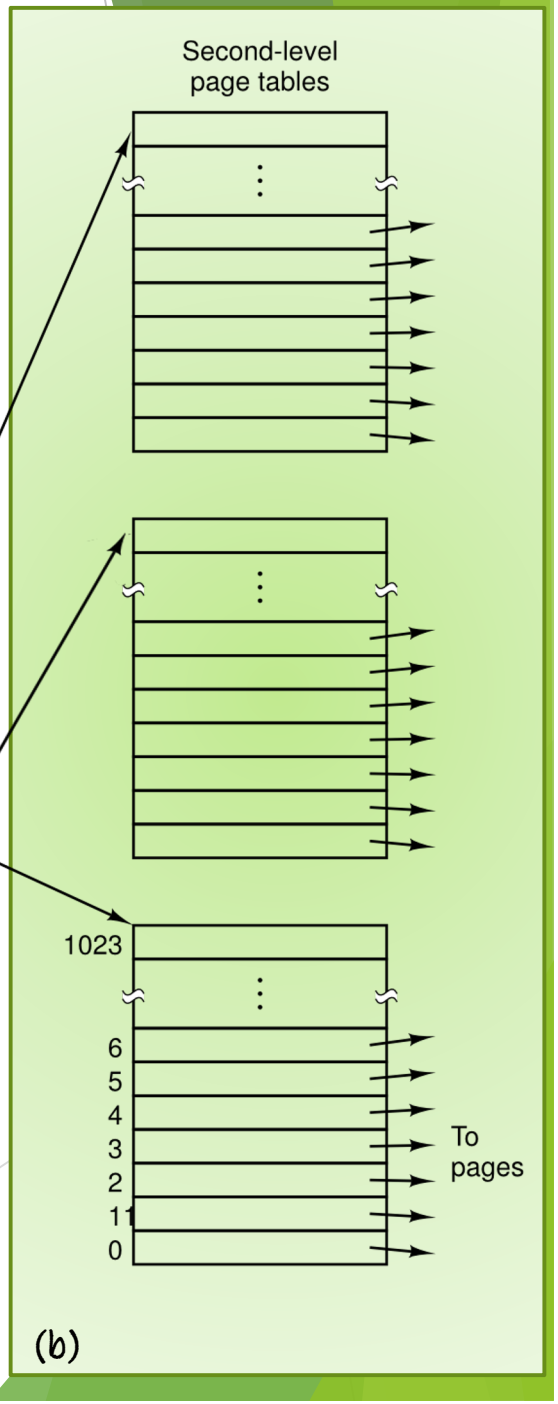
Tabella delle Pagine Multilivello

- Resta il **problema delle dimensioni**:
  - indirizzi virt. a 32 bit (Pentium): 1 milione di pagine da 4KB;
    - tabella grande circa 4MB (supponendo voci da 32 bit);
    - necessita di memoria contigua;
  - indirizzi virt. a 64 bit: 2^{52} pagine da 4KB;



(a)

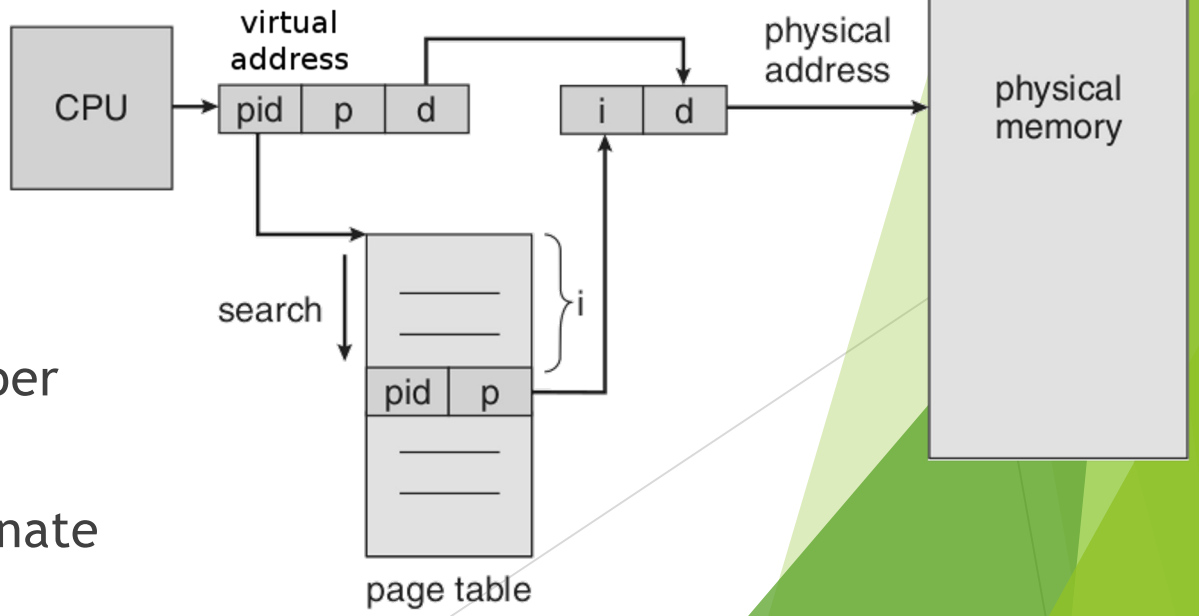




- soluzione: **non mantenere l'intera tabella in memoria**
  - **tabelle multilivello** (paginazione gerarchica);
    - nel caso di 2 livelli, possono essere necessari 3 accessi alla memoria;
  - anche su **più livelli** (ma senza esagerare...).

Tabella delle Pagine Invertite

- **Una voce per ogni frame fisico**;
  - ogni voce riporta: **(id processo, pagina virtuale)**;
- tabella al quanto piccola a paragone delle t.p.;
  - questa implicherebbe una **ricerca molto lenta**:
    - **tabella hash** indicizzata sull'indirizzo virtuale;
    - accoppiata con una **TLB**.



- servono ancora le **tabelle per processo**:
  - informazioni necessarie per gestire i page fault
  - possono essere però paginate a loro volta.

Tabella delle Pagine Invertite

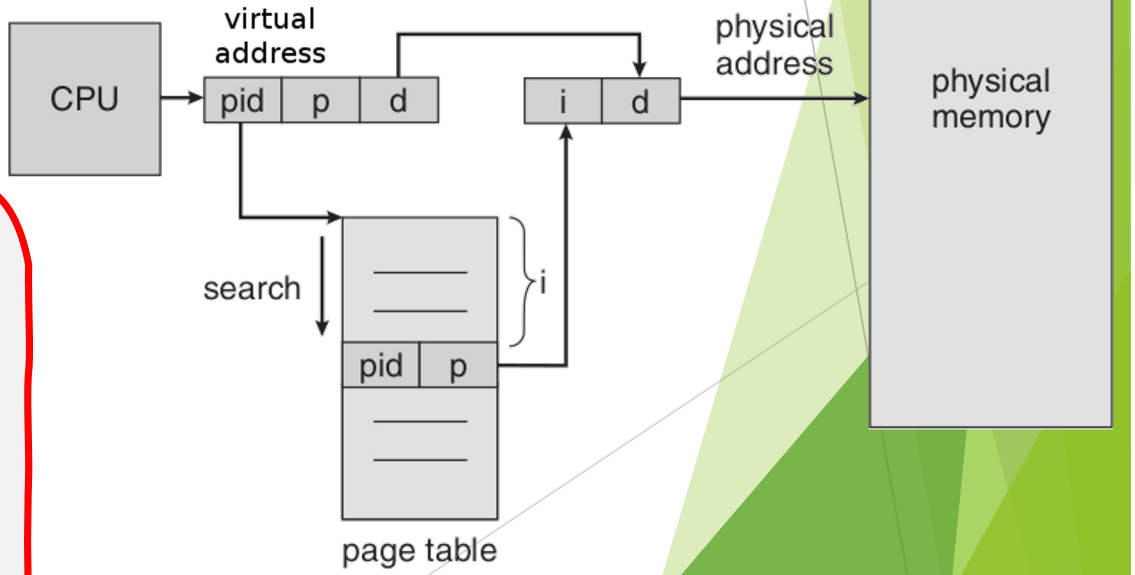
- Una voce per ogni frame fisico:
  - informazioni necessarie per gestire i page fault
  - possono essere però paginate a loro volta.

Vantaggio:

non deve mai essere più grande della dimensione della memoria fisica

Tabella delle Pagine Invertite

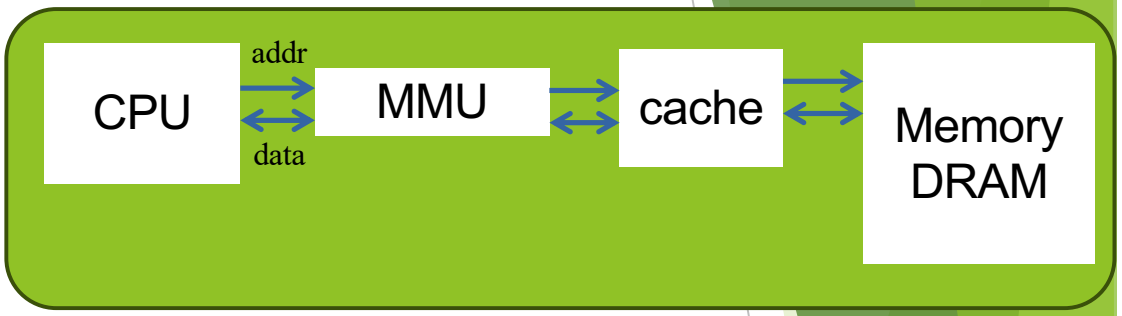
- **Una voce per ogni frame fisico**;
  - ogni voce riporta: **(id processo, pagina virtuale)**;
- tabella alquanto piccola a paragone delle t.p.;
  - questa implicherebbe una **ricerca molto lenta**:
    - **tabella hash** indicizzata sull'indirizzo virtuale;
    - accoppiata con una **TLB**.

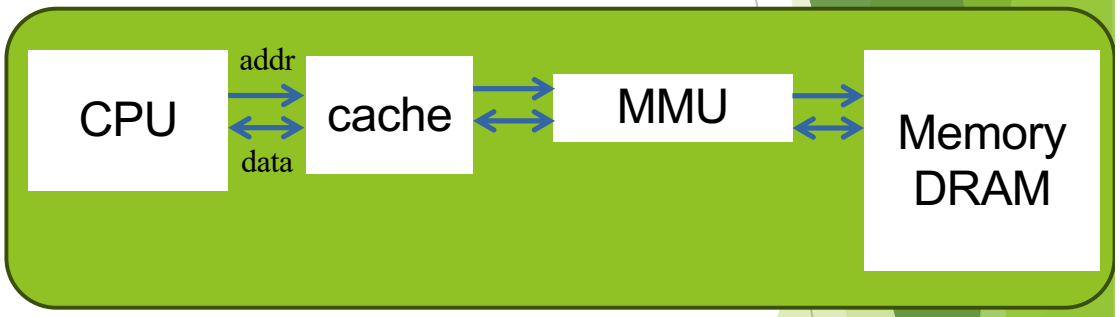


- servono ancora le **tabelle per processo**:
  - informazioni necessarie per gestire i page fault
  - possono essere però paginate a loro volta.

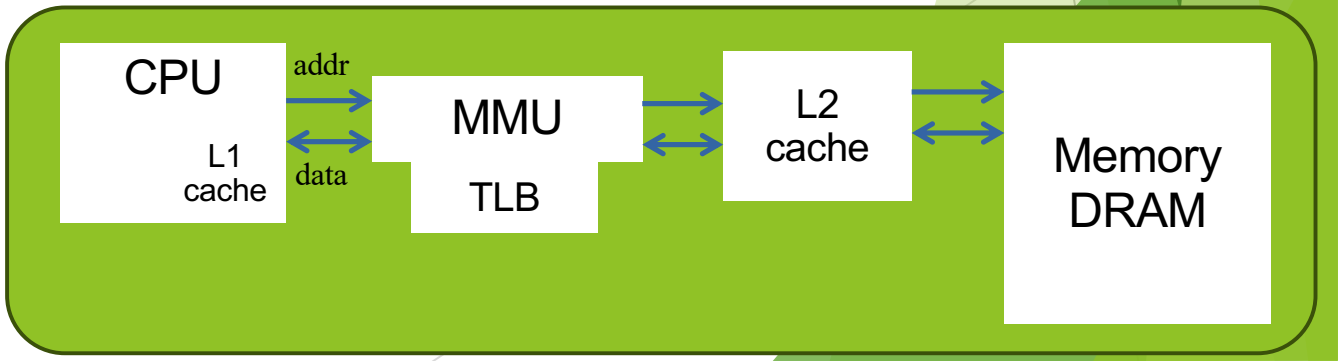
Cache della Memoria vs. Memoria Virtuale

- La **cache della memoria** può essere:
  - **basata sugli indirizzi fisici**:
    - non serve invalidarla sul context-switch;
    - caching poco efficace;
  - **basata sugli indirizzi virtuali**:
    - servono gli ASID per non invalidarla ogni volta;
    - maggiore efficacia.





- Cosa si usa **in pratica**?
  - cache L1 basata su indirizzi virtuali;
  - cache L2 e successive basate su indirizzi fisici.



Algoritmi di Sostituzione delle Pagine

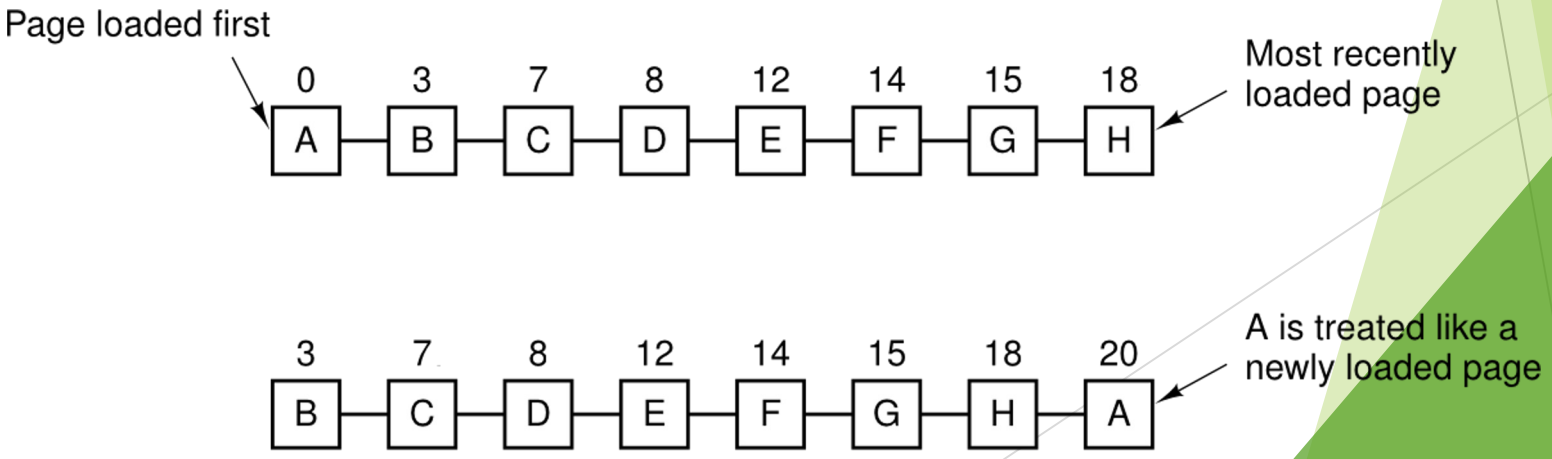
- In caso di **page fault** ed in assenza di frame liberi è necessario scegliere una **pagina vittima**;
  - **come sceglierla?**
    - problema simile alla **gestione delle cache**;
  - **obiettivo**: ***minimizzare il numero di page fault in futuro***;
- **Soluzione Ottimale** (algoritmo **OPT**):
  - scegliamo la pagina che verrà referenziata in un futuro più lontano;
  - ottimale ma **difficilmente realizzabile**;
  - rappresenta comunque un **termine di paragone.**

Algoritmo Not Recently Used (NRU)

- Raccogliamo un po' di **statistiche** sull'uso della pagine caricate:
  - bit di **referenziamento** (R) e di **modifica** (M);
    - aggiornati tipicamente in **hardware**;
    - **azzerati** dal S.O.;
      - bit di referenziamento azzerato **periodicamente**;
- algoritmo **Not Recently Used** (NRU):
  - distinguiamo **4 classi di pagine**:
    - **classe 0**: non referenziato, non modificato;
    - **classe 1**: non referenziato, modificato;
    - **classe 2**: referenziato, non modificato;
    - **classe 3**: referenziato, modificato;
  - **viene scelta una pagina dalla classe non vuota di numero più basso**.

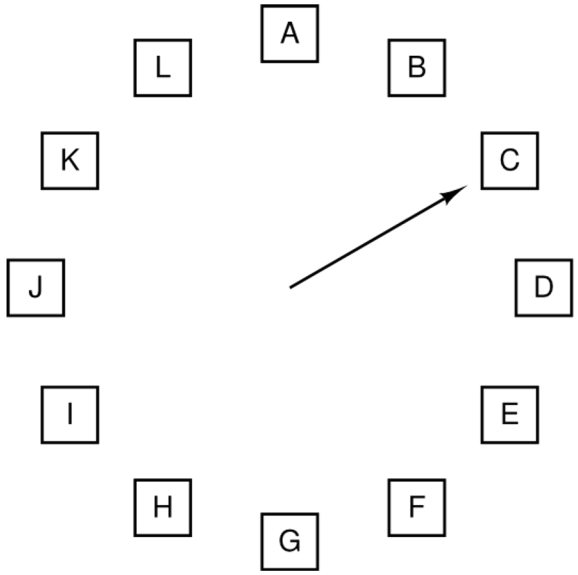
Algoritmo FIFO & della Seconda Chance

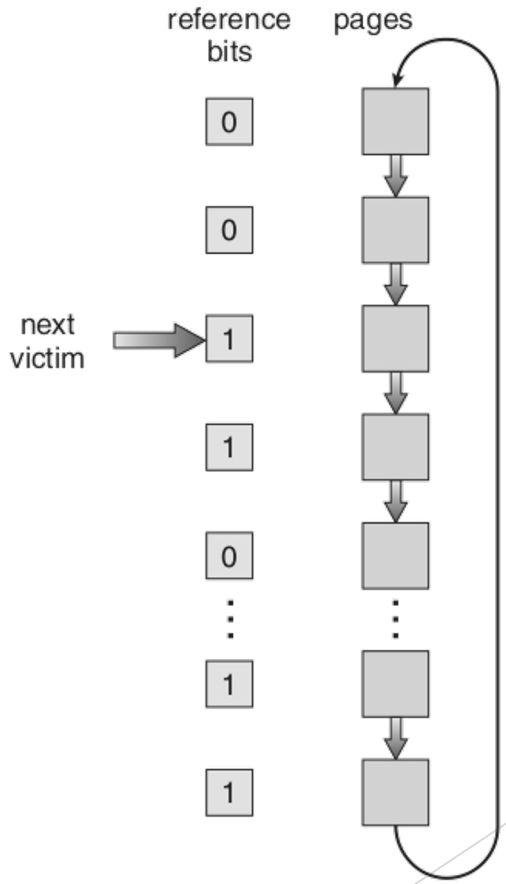
- Algoritmo **First-In First-Out** (FIFO):
  - viene rimossa la **pagina più vecchia**;
  - scelta non sempre felice: può rimuovere pagine, sì vecchie, ma magari molto usate;
- algoritmo della **Seconda Chance**:
  - si tiene conto dell'attuale stato del **bit R**;
  - viene rimossa la pagina più vecchia se non usata di recente.

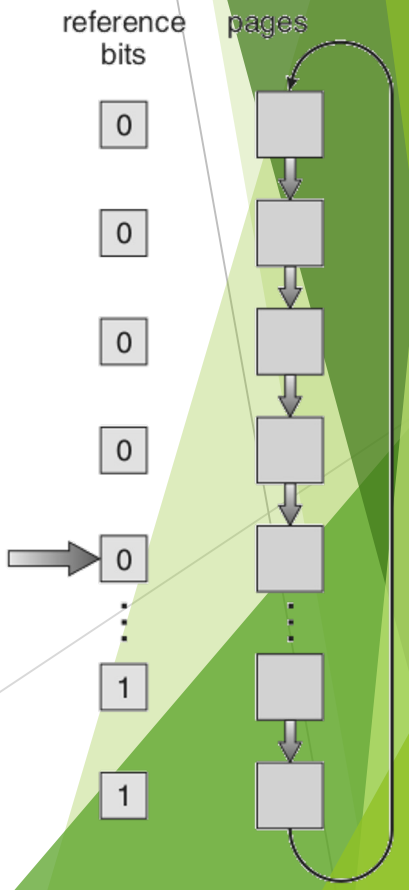


Algoritmo Clock

- L'idea dell'algoritmo della seconda chance è buona ma si può implementare in modo più efficiente:
  - algoritmo dell'orologio (**clock**).







Algoritmo least Recently Used (LRU)

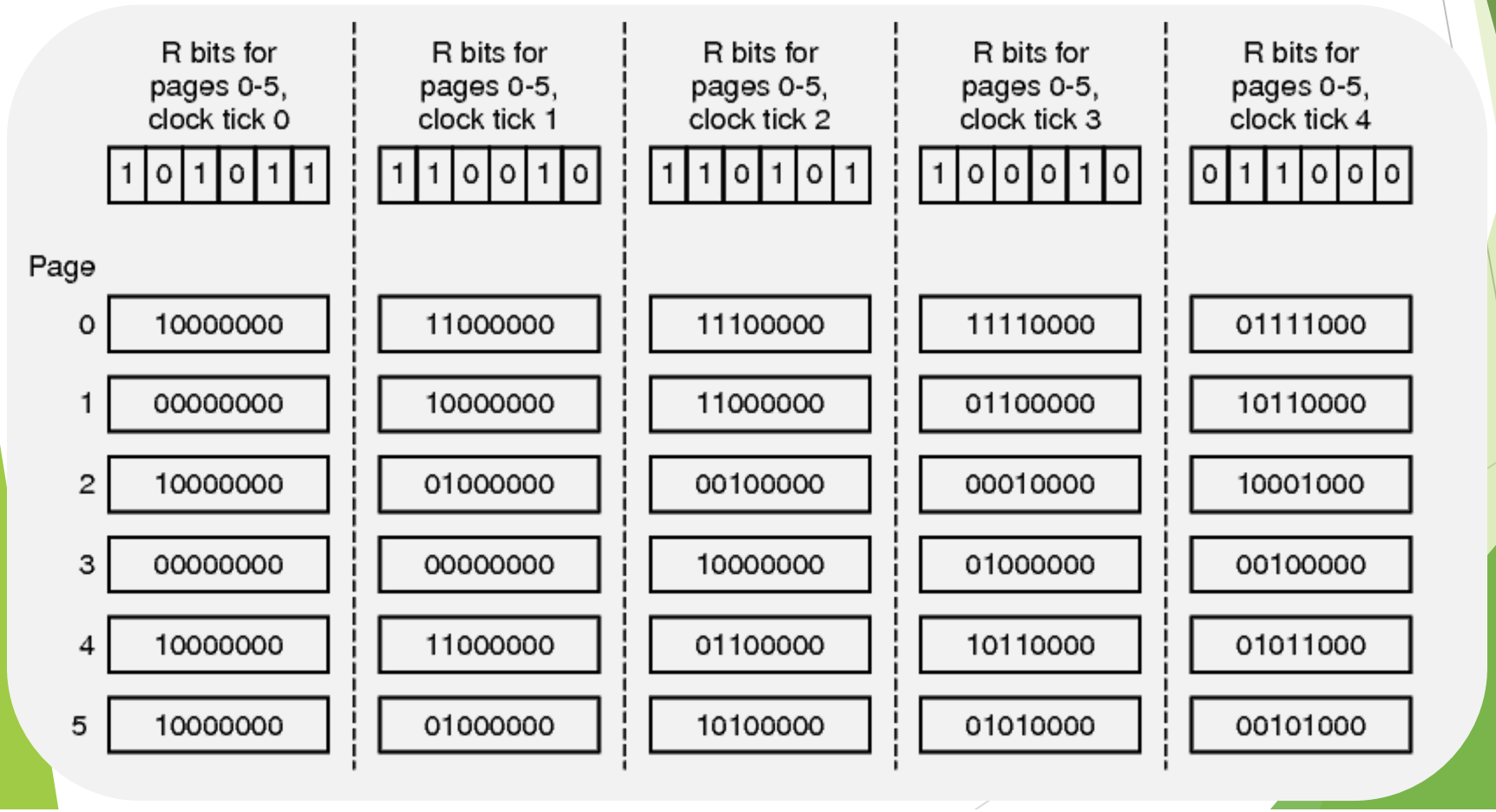
- Probabilmente le pagine più usate di recente lo saranno anche in futuro;
  - **idea**: rimuovere le pagine meno usate di recente;
- algoritmo **Least Recently Used** (LRU):
  - buona idea ma non semplice e dispendiosa da implementare:
    - con **supporto hardware**:
      - **contatore** nella CPU e campi relativi nella tabella delle pagine;

Algoritmo Not Frequently Used (NFU)

- LRU, anche se implementato in hardware, è dispendioso:
  - pensiamo ad una sua approssimazione;
- algoritmo **Not Frequently Used** (NFU):
  - si tratta di una dell'algoritmo LRU; approssimazione (via software)
  - un **contatore** in ogni voce della tabella delle pagine;
  - periodicamente il valore del **bit R**, prima di essere azzerato, viene sommato a tale contatore;
  - viene rimossa la pagina con il **contatore più basso**;
- **problema**: può erroneamente **privilegiare pagine che sono state molto utilizzate in passato** ma che invece sono scarsamente usate di recente: queste lo saranno, probabilmente, anche nel prossimo futuro.

Algoritmo di Aging

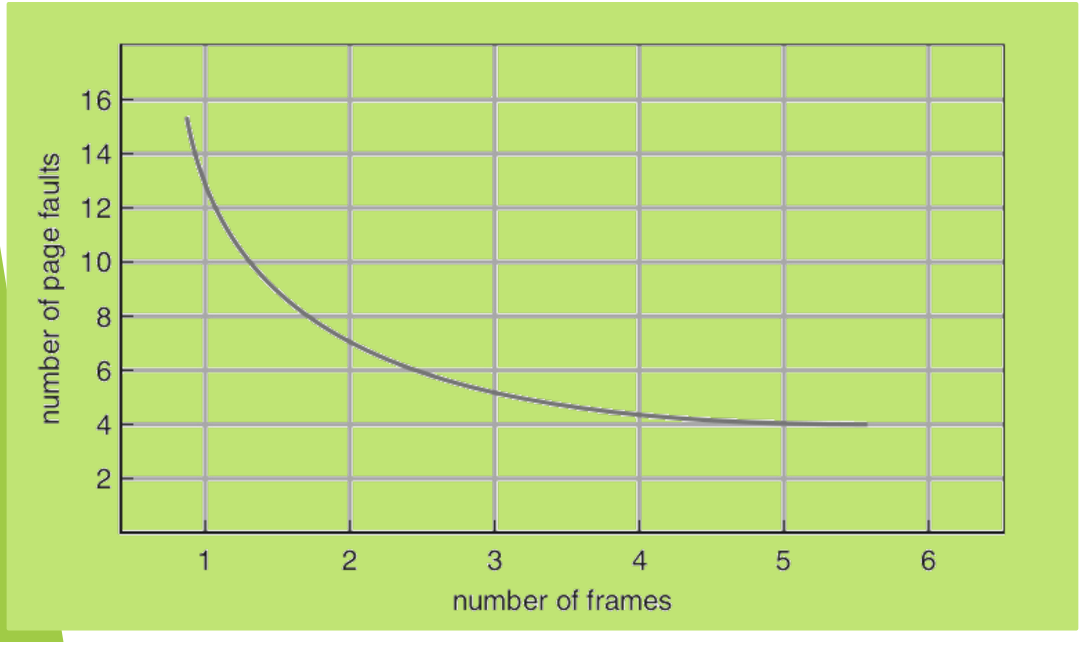
- Algoritmo di **Aging**:
  - ad ogni scadenza del clock:
    - **shift a destra** del contatore associato ad ogni pagina;
    - accostamento a sinistra (**come bit più significativo**) del bit R.



Page	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

Confronto delle Prestazioni

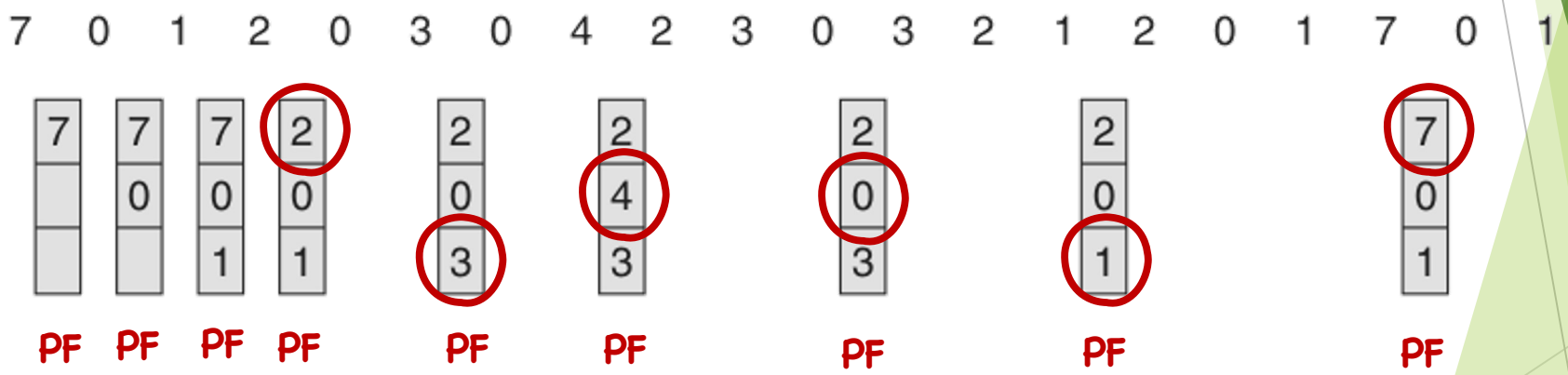
- Scegliamo una **metrica**:
  - **numero di fault di pagina**;
- a parità di condizioni, fissiamo:
  - **numero di frame:** 3 frame
  - sequenza degli indirizzi virtuali a cui accedere;
    - in modo equivalente: **sequenza compatta delle pagine**;
    - ***7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1***



Aumento del numero dei Frame => diminuzione del numero di PageFault

Confronto delle Prestazioni

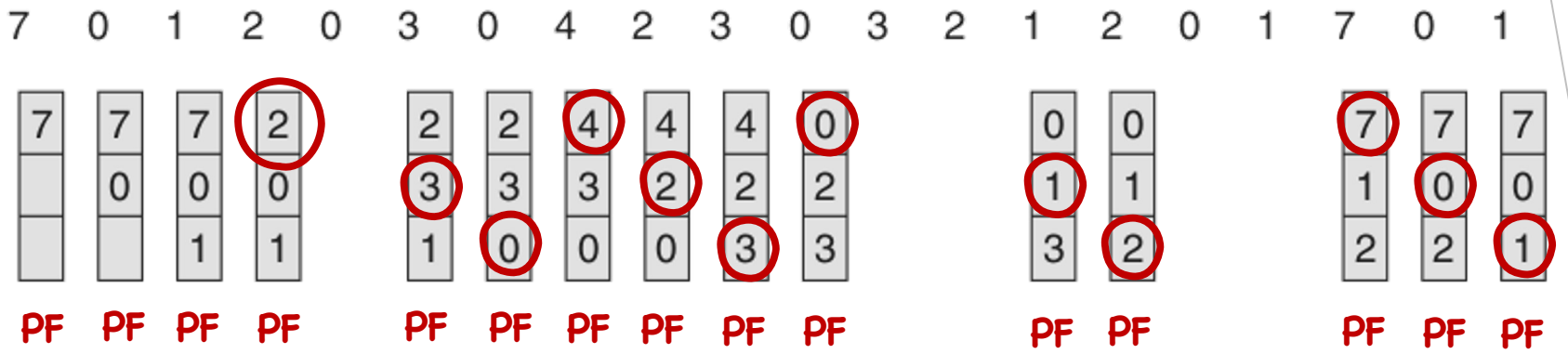
- *7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1*
- algoritmo **OPT**:



- 9 fault di pagina

Confronto delle Prestazioni

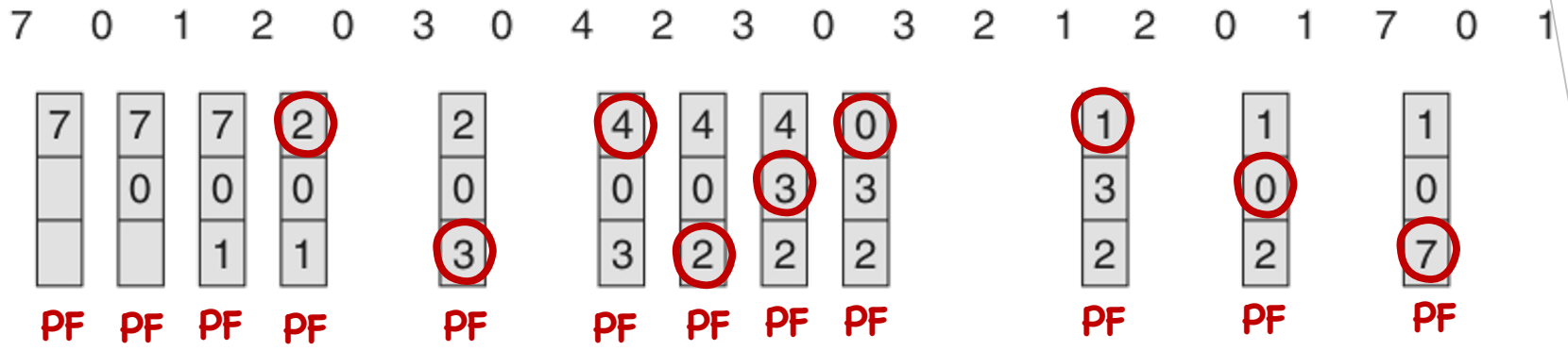
- algoritmo **FIFO**:



- 15 fault di pagina

Confronto delle Prestazioni

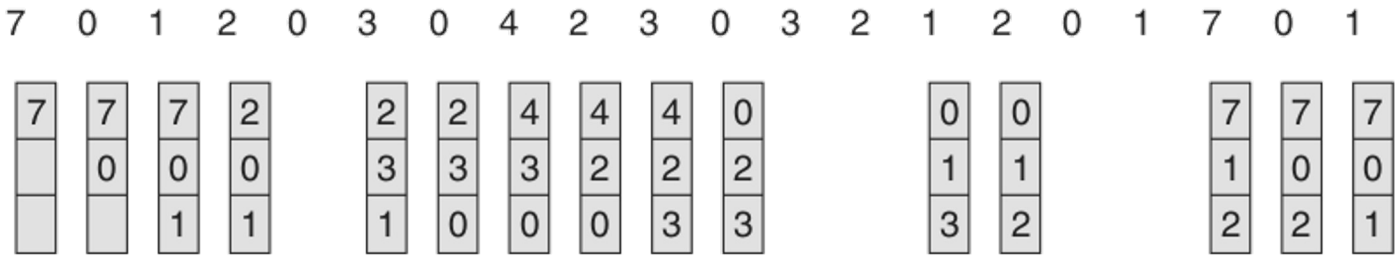
- algoritmo **LRU**:



  - 12 fault di pagina

Confronto delle Prestazioni

- algoritmo **FIFO**:



15 Page Fault

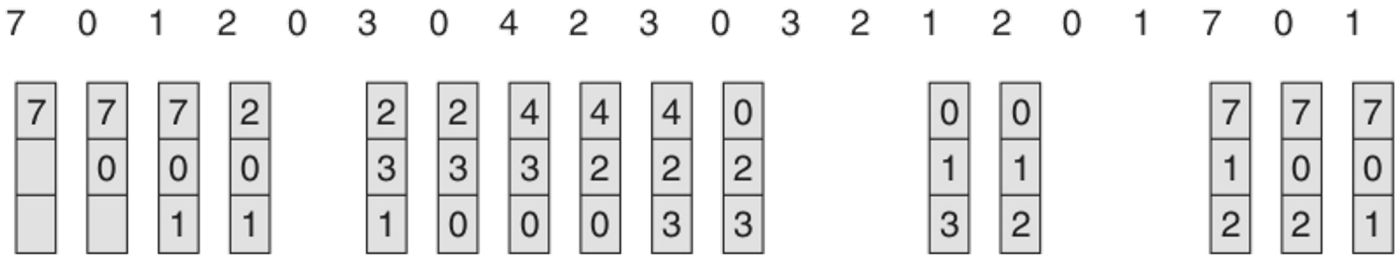
- ma c'è qualcosa di strano...
  - sequenza: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
    - frame #3 vs frame #4

9 Page Fault

10 Page Fault

Confronto delle Prestazioni

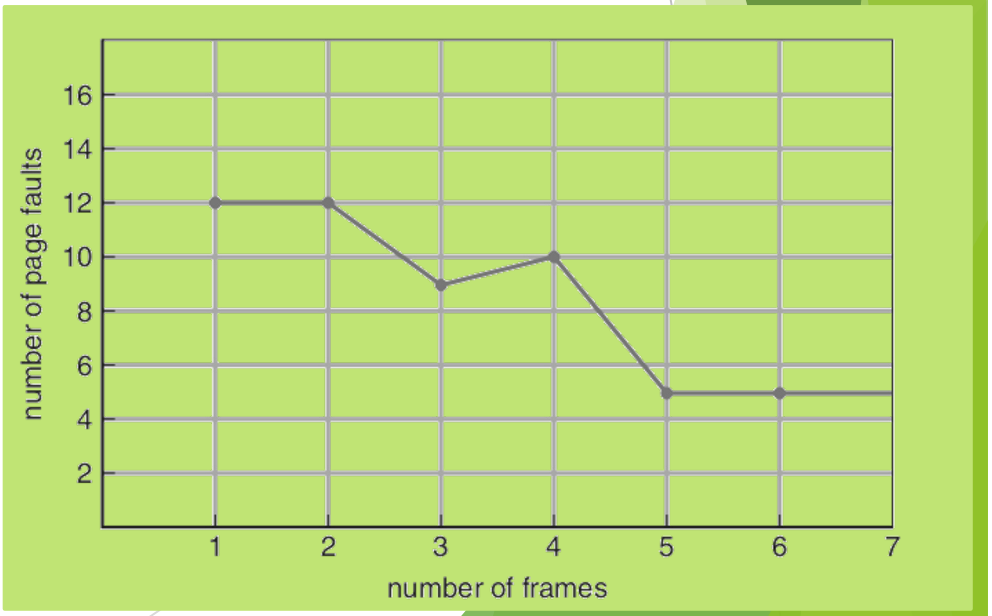
- algoritmo **FIFO**:



15 Page Fault

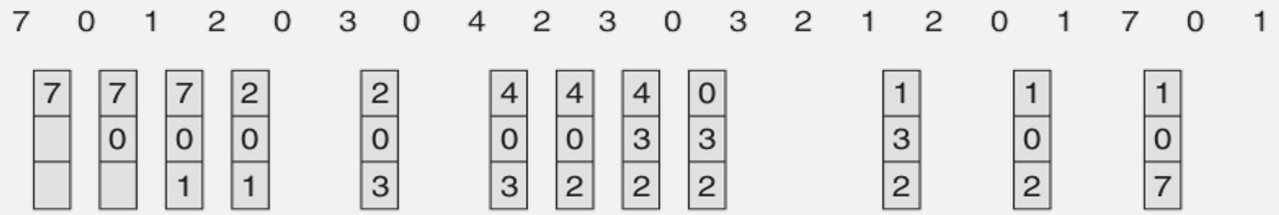
- ma c'è qualcosa di strano...
  - sequenza: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
    - frame #3 vs frame #4

- **anomalia di Belady**: aumentando i frame disponibili, aumentano i fault di pagina (!?!)



Confronto delle Prestazioni

- algoritmo **LRU**:



  - 12 fault di pagina

- non soffre dell'anomalia precedente, infatti vale la
  - **proprietà di inclusione**: l'insieme di pagine caricati avendo n frame è incluso in quello che si avrebbe avendo n+1 frame.
    - $B_t(n) \subseteq B_t(n+1) \quad \forall\ t,n$
- altri algoritmi visti:
  - **NFU, aging**: godono della proprietà di inclusione (appross. LRU);
  - **seconda chance, clock**: soffrono dell'anomalia (riducono a FIFO);
  - **NRU**: soffre dell'anomalia (riduce a FIFO).

Algoritmi di Sostituzione delle Pagine: Riepilogo

OPT: non implementabile, ma utile come termine di paragone;

NRU*: approssimazione rozza dell'LRU;

FIFO*: può portare all'eliminazione di pagine importanti;

Seconda chance*: un netto miglioramento rispetto a FIFO;

Clock*: come S.C. ma più efficiente;

LRU: eccellente idea (vicina a quella ottima) ma difficilmente realizzabile se non in hardware;

NFU: approssimazione software abbastanza rozza dell'LRU;

Aging: buona approssimazione di LRU con implementazione software efficiente.

* soffre dell'anomalia di Belady

Allocazione dei Frame

Paginazione su richiesta (pure **demand paging**);

quant

- **mini**
  - str
    inc
- **mas**

strate

- alloc
- alloc
  - al
    - a
    - d
  - *adeguamenti al livello di multiprogrammazione;*
- allocazione per **priorità**.

Ex: sistema monoutente con 512Kb di memoria e pagine dimensione 1Kb

512 frame

SO -> 128Kb

Programmi utente -> ***384 frame***

Allocazione dei Frame

Paginazione su richiesta (pure **demand paging**);

quant

- **mini**
  - str
    inc
- **mas**

strate

- alloc
- alloc
  - al
    - a
    - dove, $S = \sum s_i$;
  - *adeguamenti al livello di multiprogrammazione;*
- allocazione per **priorità**.

- *L'istruzione che causa il PF deve essere riavviata*
- *Cosa succede se viene consentito l'indirizzamento indiretto?*
- *E se abbiamo più livelli di indirizzamento indiretto?*

Allocazione dei Frame

Paginazione su richiesta (pure **demand paging**);

quanti frame **assegnare** ad ogni processo?

- **minimo**:
  - strutturale (set istruzioni, livelli di indirizzamento indiretto);
- **massimo**: memoria libera;

strategie:

- allocazione **equa**;
- allocazione **proporzionale**:
  - al processo *i* di dimensione s_i:
    - assegniamo ($a_i = (s_i / S) \times m$) frame;
    - dove, $S = \sum s_i$;
  - *adeguamenti al livello di multiprogrammazione;*
- allocazione per **priorità**.

NOTA: a_i deve essere:
- maggiore del minimo numero di frame richiesto
- inferiore a m !!

Allocazione dei Frame

- Quali pagine considerare per la rimozione?
  - solo quelle dello stesso processo: **allocazione locale**;
  - tutte le pagine (anche di altri processi): **allocazione globale**;
- Cosa succede se ci sono **pochi frame** assegnati ad un processo?
  - sotto il **minimo strutturale**: viene sospeso e si fa swapping su disco;
  - un po' sopra:
    - può andare in **trashing**;
  - quando il trashing riguarda tutti i processi si parla di sistema in **sovraccarico** (eccessivo livello di multiprogrammazione);

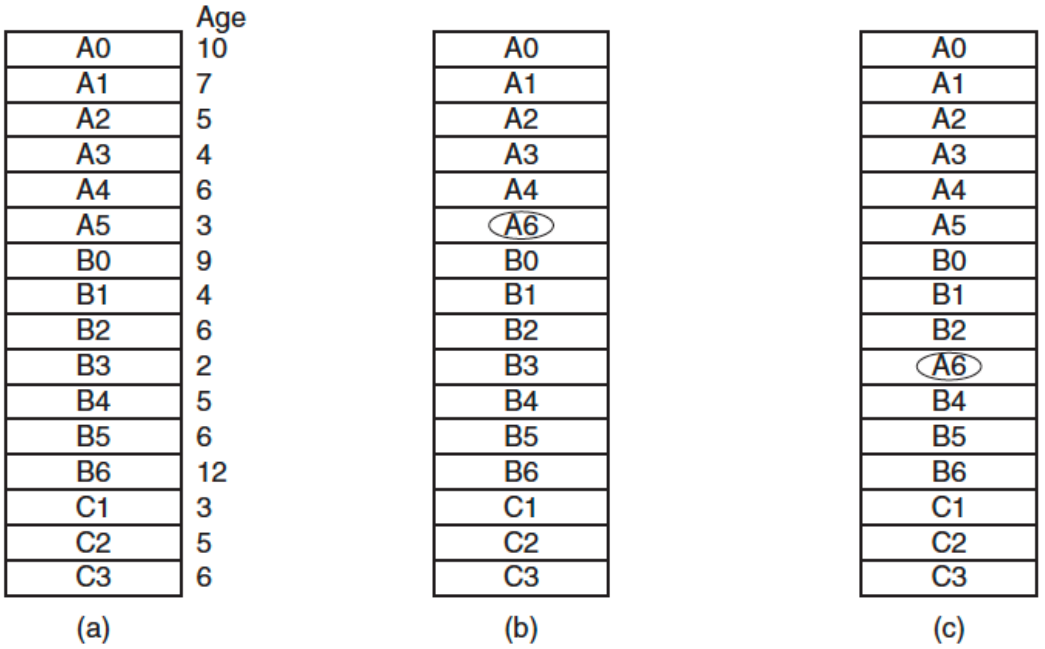


Figure 3-22. Local versus global page replacement. (a) Original configuration. (b) Local page replacement. (c) Global page replacement.

Allocazione dei Frame

- Bisognebbe assegnare un numero di frame commisurato alle "**necessità** del processo":
  - **modello di località**;
  - concetto di **località**;
  - strategia di base.

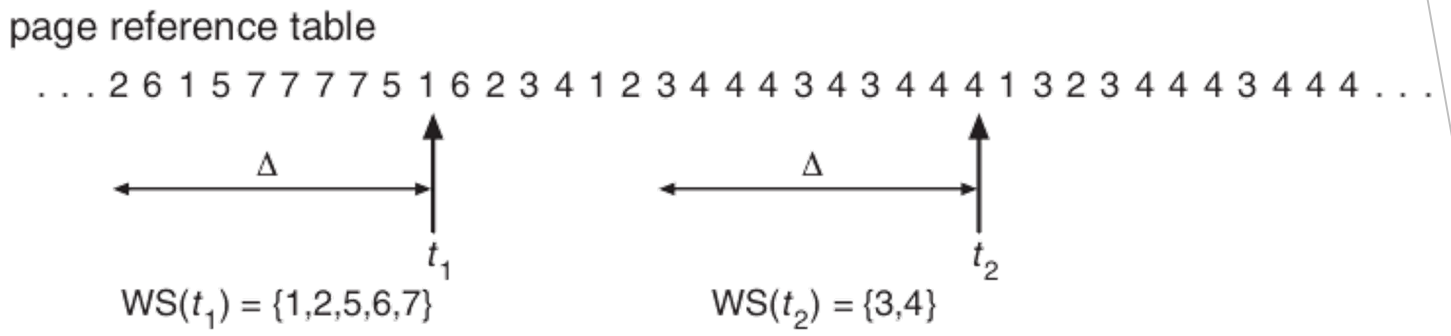
Località: insieme di pagine che vengono utilizzate attivamente insieme

definite dalla struttura del programma e dalle sue strutture dati

Il modello di località afferma che tutti i programmi mostreranno la struttura di riferimento della memoria di base

Working Set

- Per ogni processo manteniamo un **working set**:
  - pagine usate negli ultimi Δ accessi alla memoria;
  - scelta del **parametro** Δ per adattarlo alla **località corrente**;



- conoscendo il working set attuale di ogni processo:
  - **richiesta globale di frame** ($D = \sum WSS_i$) *vs.* memoria disponibile;
  - prevenzione del **trashing**
- come si calcola il working set in **pratica**? Si può approssimare usando:
  - **interrupt** periodici;
  - bit di **referenziamento** R;
  - un log che conserva la ”storia di R” in base al parametro Δ.

Working Set

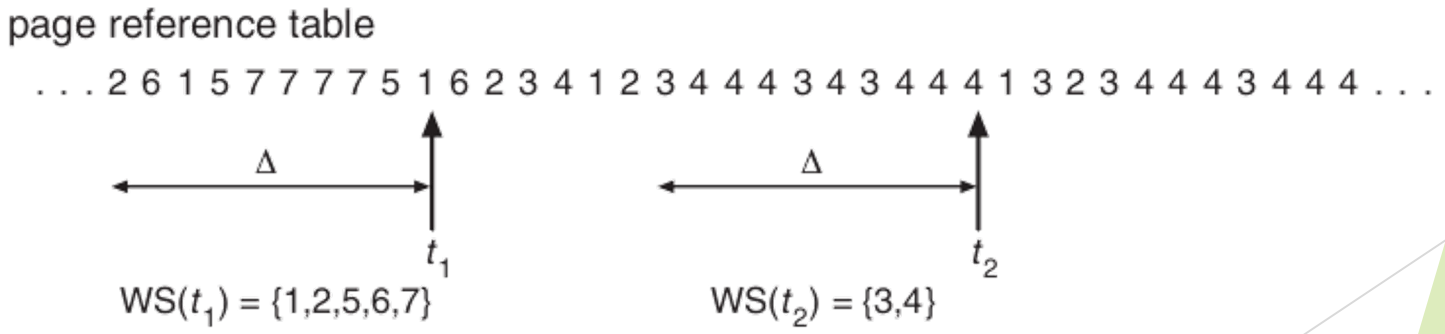
- Per ogni processo manteniamo un **working set**:
  - pagine usate negli ultimi Δ accessi alla memoria;
  - scelta del **parametro** Δ per adattarlo alla **località corrente**;
-
-
  - bit di **referenziamento** R;
  - un log che conserva la ”storia di R” in base al parametro Δ.

PREPAGING - PREPAGINAZIONE

Tenendo traccia del Working Set è possibile caricare le pagine prima che il processo vada in esecuzione

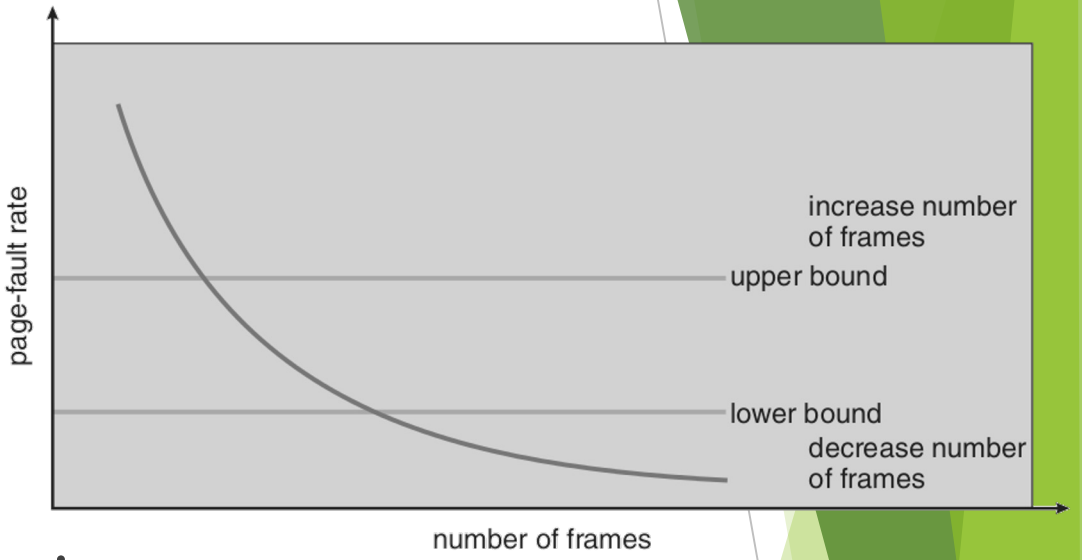
Working Set

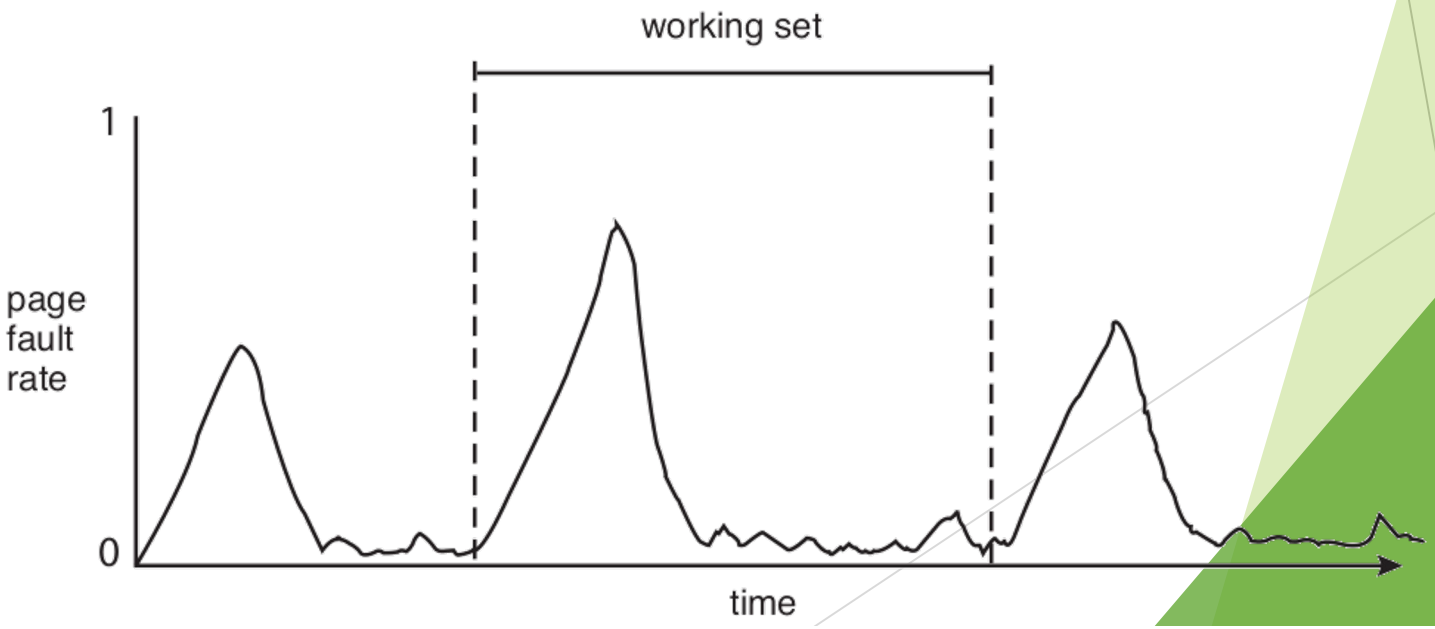
- come si calcola il working set in **pratica**? Si può approssimare usando:
  - **interrupt** periodici;
  - bit di **referenziamento** R;
  - un log che conserva la ”storia di R” in base al parametro Δ.



Page Fault Frequency

- Esiste un modello più diretto per approcciare il **trashing**:
  - monitoraggio della **Page Fault Frequency** (**PFF**) dei processi;
- caratterizzazione dei sistemi in **sovraccarico**;
- in realtà i due modelli sono **in relazione**





Politica di Pulitura

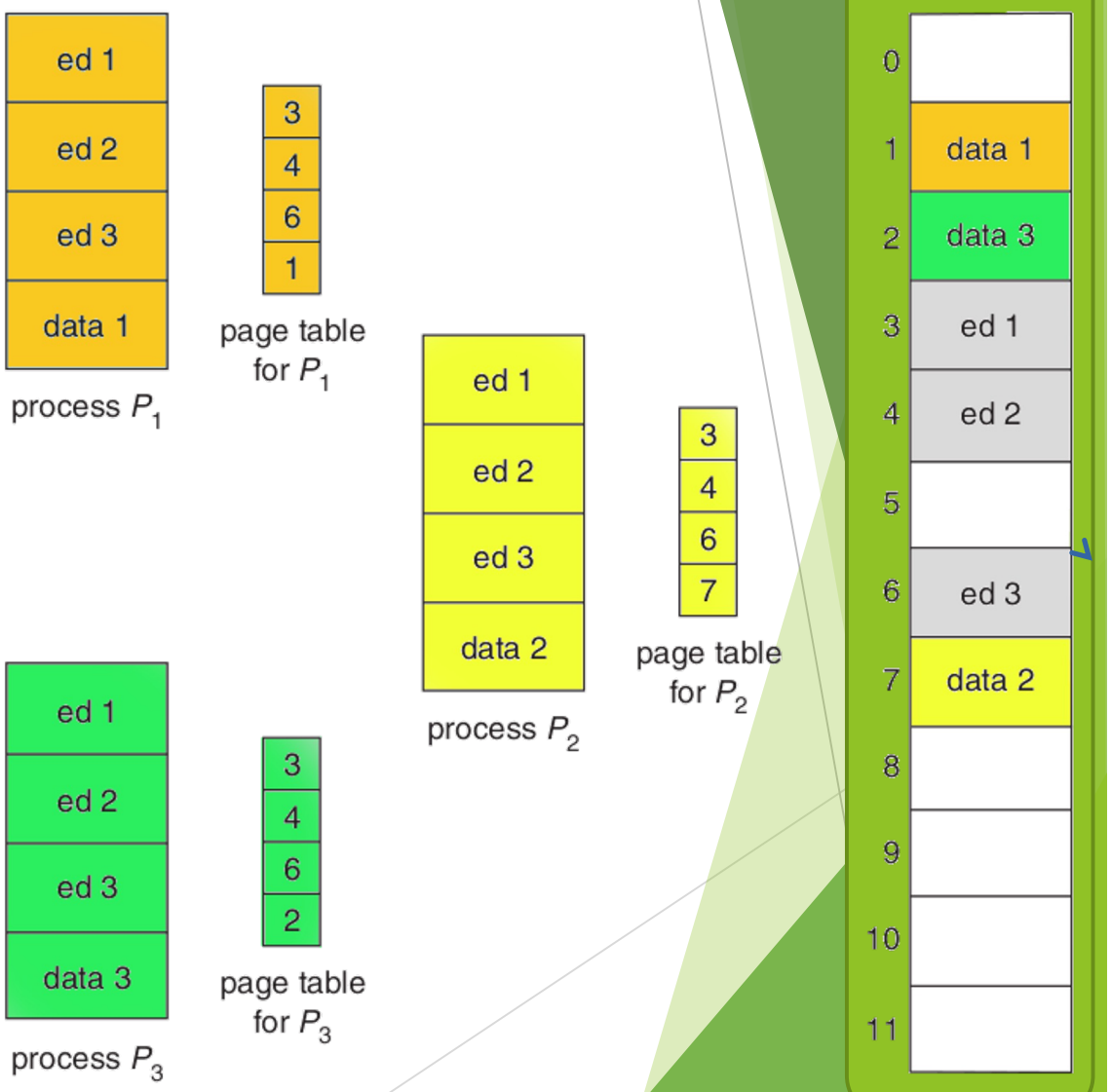
- Il meccanismo di gestione dei page fault è efficiente soprattutto se ci sono **frame liberi** sempre disponibili;
- ciò velocizza la gestione dei page-fault;
- **paging daemon**: processo di servizio che controlla lo stato di occupazione globale dei frame del sistema;
  - seleziona, libera ed, eventualmente, pulisce pagine;
  - mantiene un **pool di frame liberi**;
  - possibilità di ripescaggio dal pool in caso di richiesta;
  - usato ad esempio su Linux e Windows.

Dimensione della Pagina

- La scelta della dimensione della pagina di base è importante:
  - vantaggi di una **pagina grande**:
    - tabella della pagine più piccola;
    - migliore efficienza nel trasferimento I/O;
    - tende a minimizzare il numero di page fault (minore overhead);
  - vantaggi di una **pagina piccola**;
    - minore frammentazione interna;
    - migliore risoluzione nel definire il working set in memoria (meno memoria sprecata);
- relazione con la **dimensione del blocco su disco**.

Pagine Condivise

- I processi posso anche condividere in vari modi alcune pagine:
  - **solo lettura**:
    - codice eseguibile condiviso (**codice rientrante**);
  - **lettura/scrittura**:
    - IPC tramite memoria condivisa;
- implementazione su **tabella delle pagine ordinaria o multilivello:**
  - semplice ed efficiente.



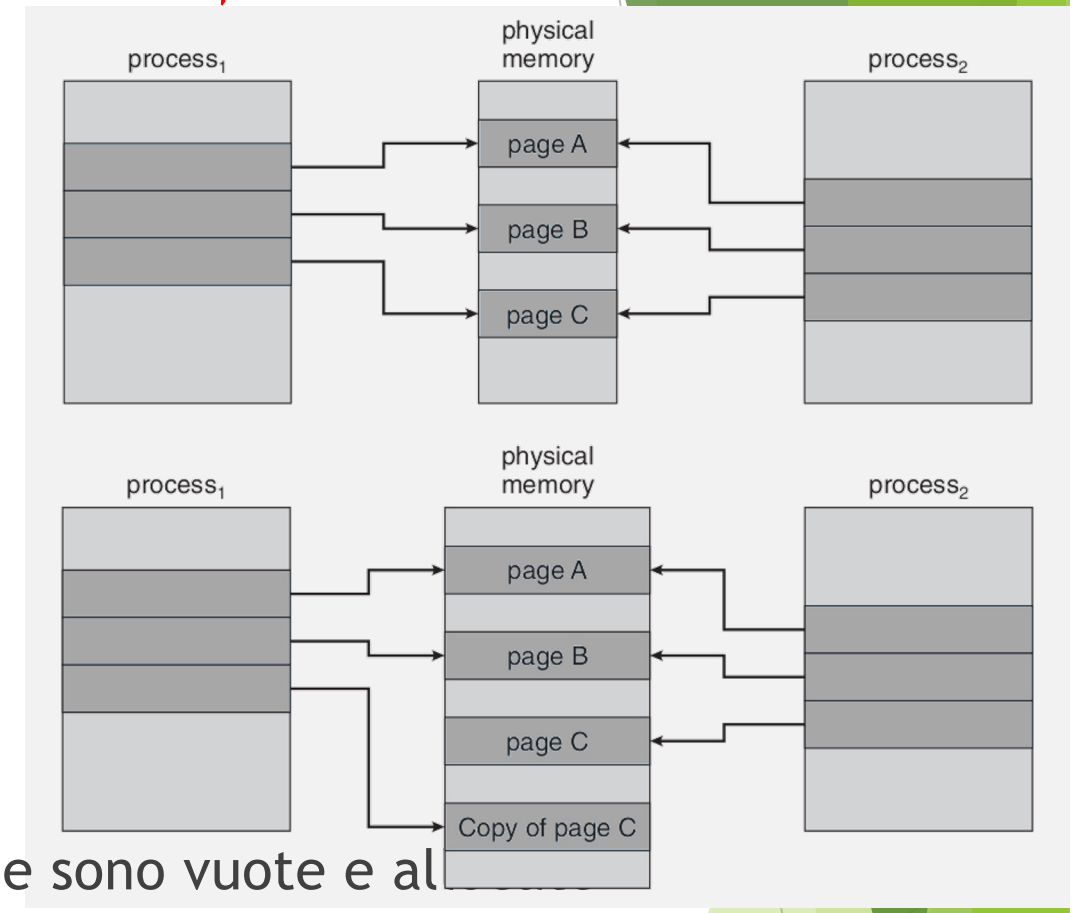
Pagine Condivise

- **Gestione della cache:**
  - **problemi di sincronizzazione** con cache basate su indirizzi virtuali (anche se usano gli ASID);
    - *problema di aliasing*
  - soluzioni:
    - disabilitare la cache sulle pagine condivise;
    - usare **cache con ricerca basata su indirizzi virtuali e tag fisici (VIPT - Virtually Indexed and Physically Tagged):**
      - la cache ricerca in parallelo con la TLB sulla base dell'indirizzo virtuale;
      - per capire se si tratta di un duplicato dobbiamo aspettare che la TLB dia in output l'indirizzo fisico;

Pagine Condivise

- **tabella delle pagine invertita**:
  - *singolo core*
    - alterazione tabella su context switch o su page fault;
  - *multi core*
    - difficilmente gestibile se non con teoriche tabelle delle pagine invertite con corrispondenze molti-a-uno.

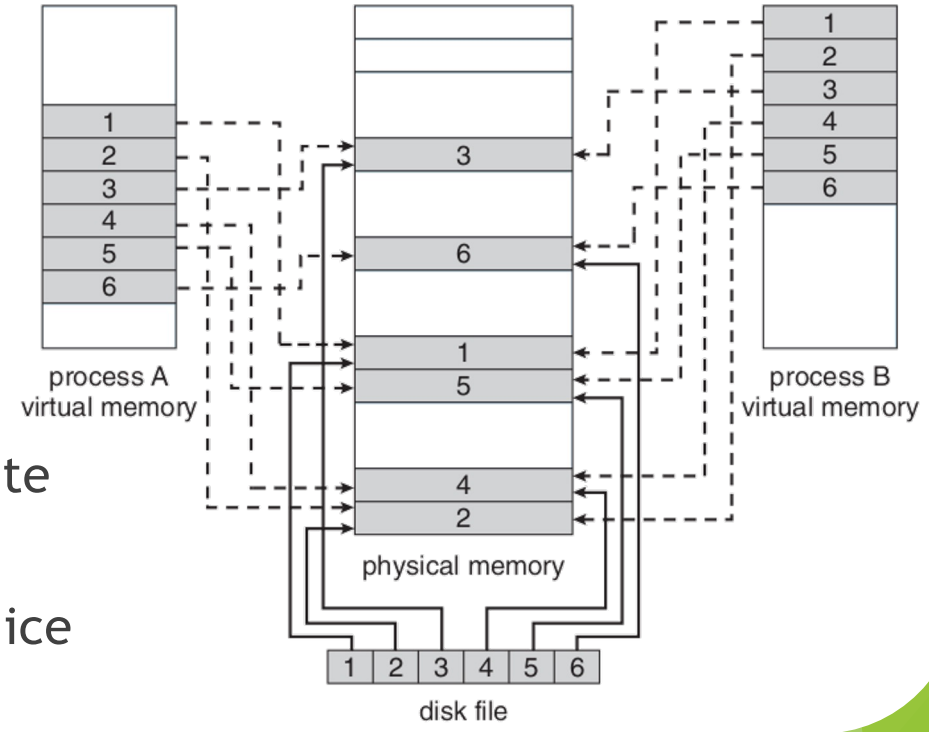
Copy-on-write & Zero-fill-on-demand



- **copy-on-write**:
  - condivide finché possibile tutti i tipi di pagine (codice e dati);
- **zero-fill-on-demand**:
  - principio di base: le nuove pagine sono vuote e al su richiesta;
    - azzeramento efficiente gestito dal kernel;
    - sicurezza;
    - prestazioni.
  - copy-on-write su una *read-only static zero page (Linux)*.

***NOTA**: riempire a zero le pagine può essere fatto quando il sistema è inattivo!!*

Librerie Condivise



- *Grandi librerie* condivise sono comunemente usate;
  - **Linking statico**: *inclusione* del codice *in fase di linking*;
  - **Linking dinamico**: collegamento e *caricamento a run-time* di librerie condivise:
    - risparmio di spazio su disco e in RAM;
    - sviluppo indipendente e facilità di aggiornamento delle librerie;

Stub: piccola porzione di codice che verifica se la routine necessaria è già in memoria o meno

File Mappati

- File mappati in memoria:
  - modello alternativo di I/O su file;
  - possibilità di condivisione;
  - accesso al file più veloce;
  - gestiscono automaticamente:
    - librerie condivise;
    - caricamento codice eseguibile;
    - caricamento dei dati statici.
  - consentire a più processi di mappare lo stesso file contemporaneamente

Allocazione della Memoria per il Kernel

- **Memoria dei processi utente**: paginata ma con frammentazione interna;
- **Memoria interna al kernel**:
  - miriamo a **frammentazione interna** minima o nulla;
  - **impossibilità di paginare** l'allocazione in alcuni casi.