Sistemi Operativi: Memoria

Amos Brocco, Ricercatore, DTI / ISIN

Basato su:

[STA09] “Operating Systems: Internals and Design Principles”, 6/E, William Stallings, Prentice Hall, 2009

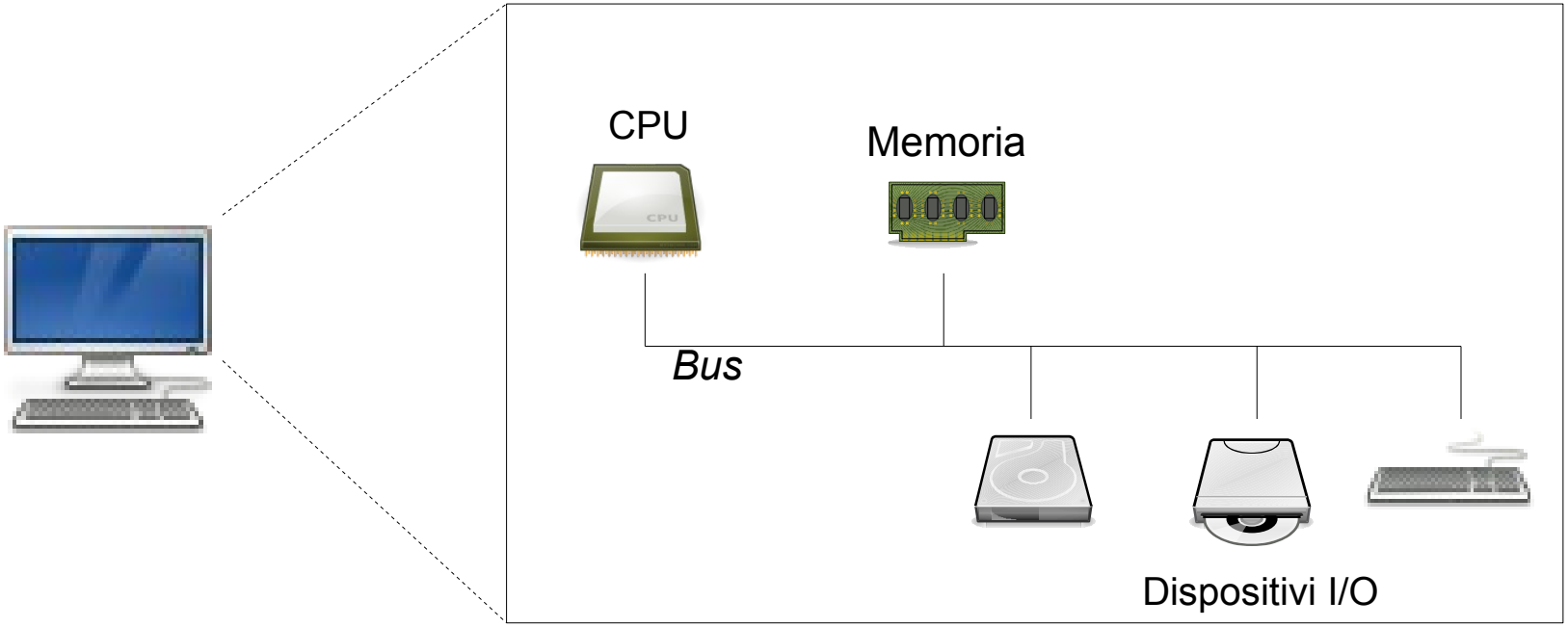
[TAN01] “Modern Operating Systems”, 2/E, Andrew S. Tanenbaum, Prentice Hall, 2001

[TAN09] “Modern Operating Systems”, 3/E, Andrew S. Tanenbaum, Prentice Hall, 2009

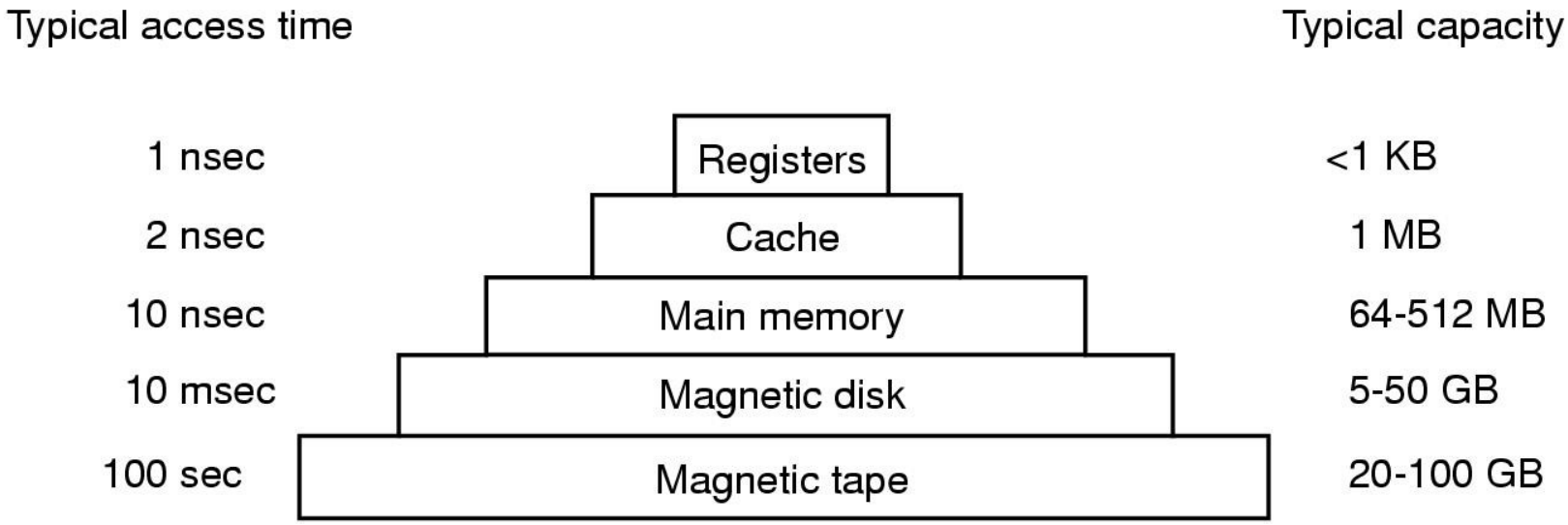
Motivazione

- Perché studiare come la memoria viene gestita in un sistema operativo moderno?
  - per capire come utilizzarla meglio
    - quali sono i limiti dell'hardware e del software
  - per capire cosa significano termini come swapping, memoria virtuale, segmentazione, segmentation fault, paginazione,...

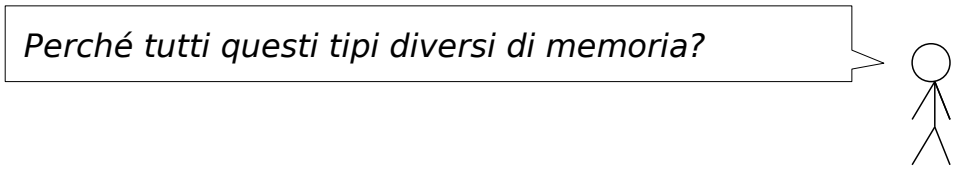
Architettura di un computer



Gerarchia della memoria

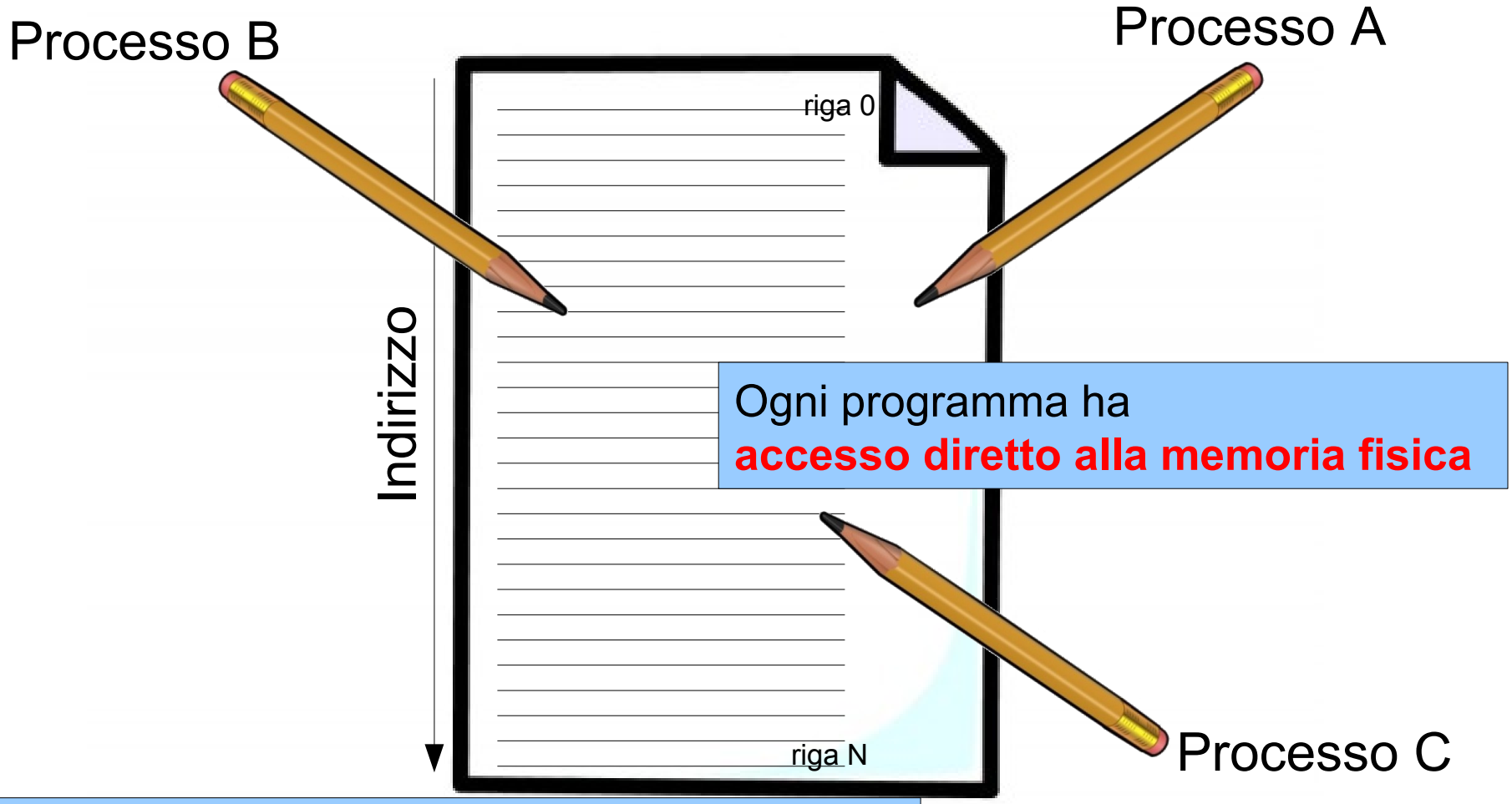


Gerarchia di memoria



- Idealmente i programmatori e gli utenti vorrebbero disporre di memoria che è allo stesso tempo
  - Infinita
  - Veloce
  - A basso costo
  - Non volatile
- *Purtroppo questo obiettivo non è ancora stato raggiunto*
- È compito del sistema operativo gestire efficientemente queste memorie e offrire un'**astrazione** che "nasconda" i limiti dell'hardware

All'inizio... nessuna astrazione: “un grande foglio di carta”



I primi computer (pre-1960)
non offrono nessun tipo di astrazione della memoria

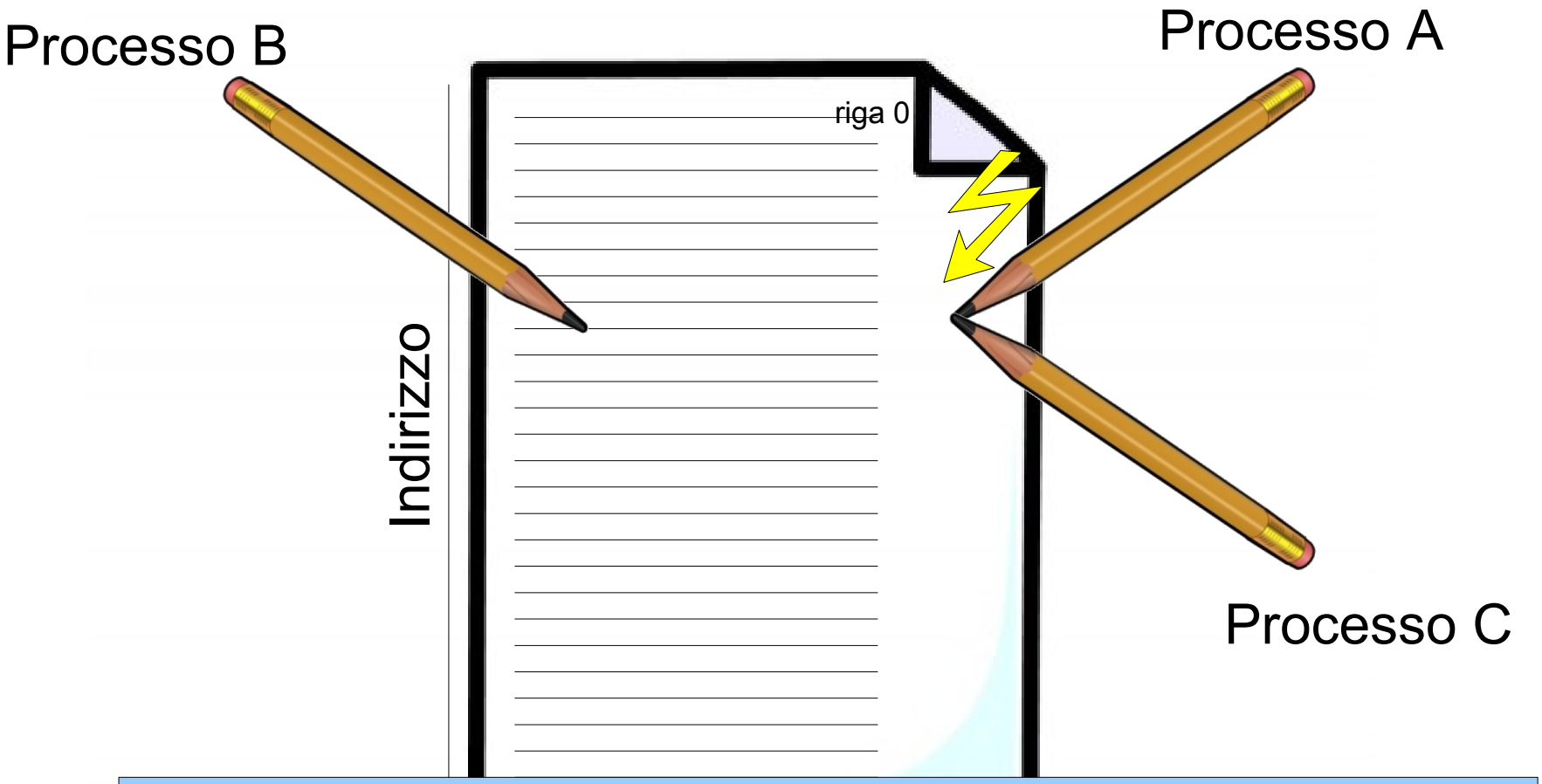
All'inizio... nessuna astrazione

- Gli indirizzi da utilizzare sono codificati nel programma, per es.

```
MOV registro1, 1981
```

sposta il contenuto della posizione di memoria 1981 nel registro numero 1

All'inizio... nessuna astrazione



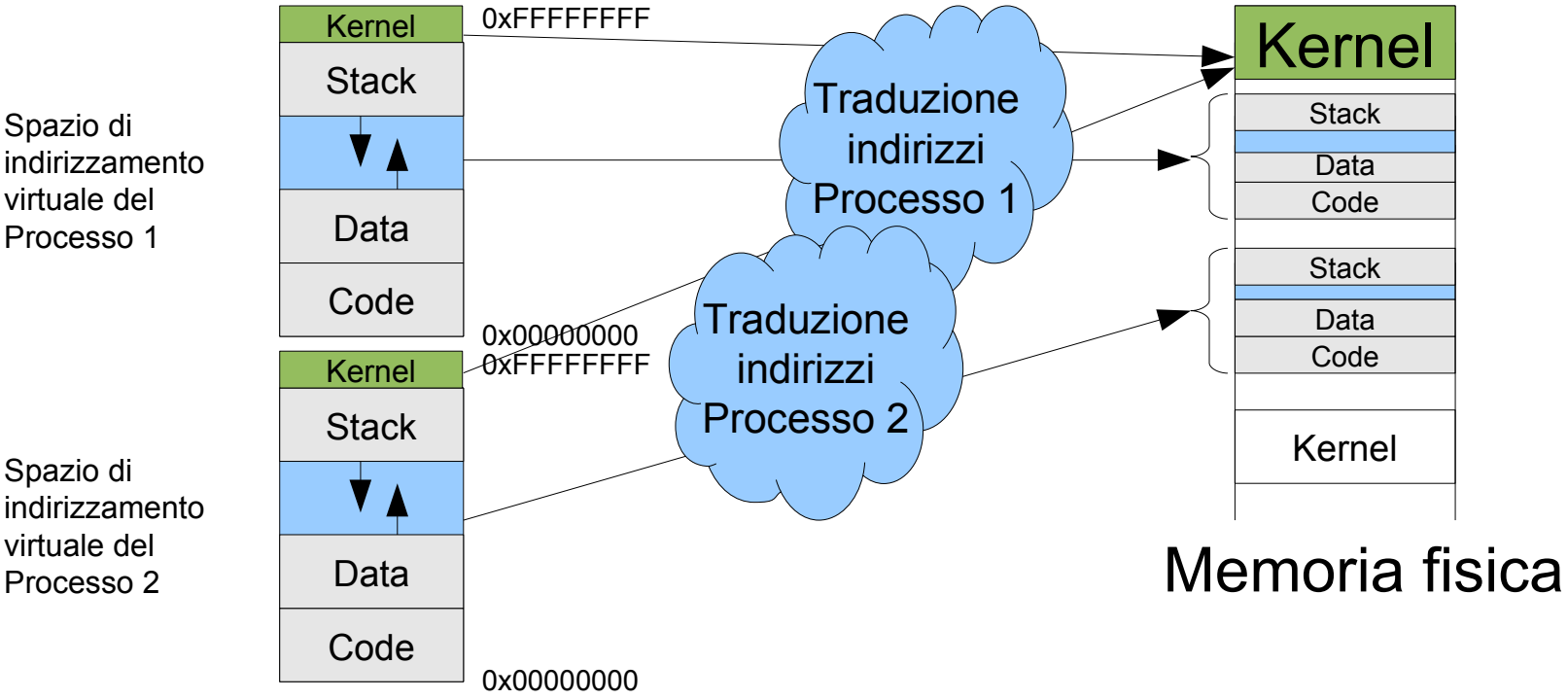
Difficile poter eseguire più di un processo contemporaneamente: processi diversi devono essere scritti in modo da utilizzare indirizzi differenti

Necessità di implementare un'astrazione

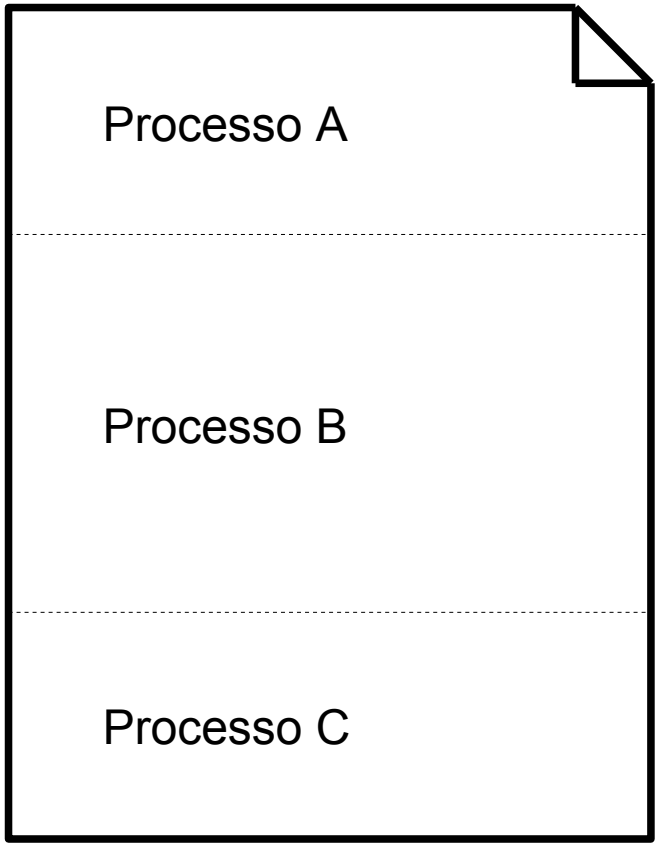
- Per **ovviare alle limitazioni** fisiche (es. dimensione finita della memoria)
- Per **semplificare la multi-programmazione**
  - è praticamente impossibile garantire che ogni programma utilizzi effettivamente indirizzi diversi
- Per **proteggere** i dati
  - se ogni processo ha accesso alla memoria fisica può sovrascrivere i dati di altri processi (o del sistema operativo!)

Traduzione degli indirizzi

- I meccanismi di gestione della memoria che analizzeremo sono basati sul concetto di “traduzione degli indirizzi”
  - Gli indirizzi di memoria contenuti nel codice non vengono utilizzati “così come sono”, cioè come riferimento nella memoria fisica (RAM)

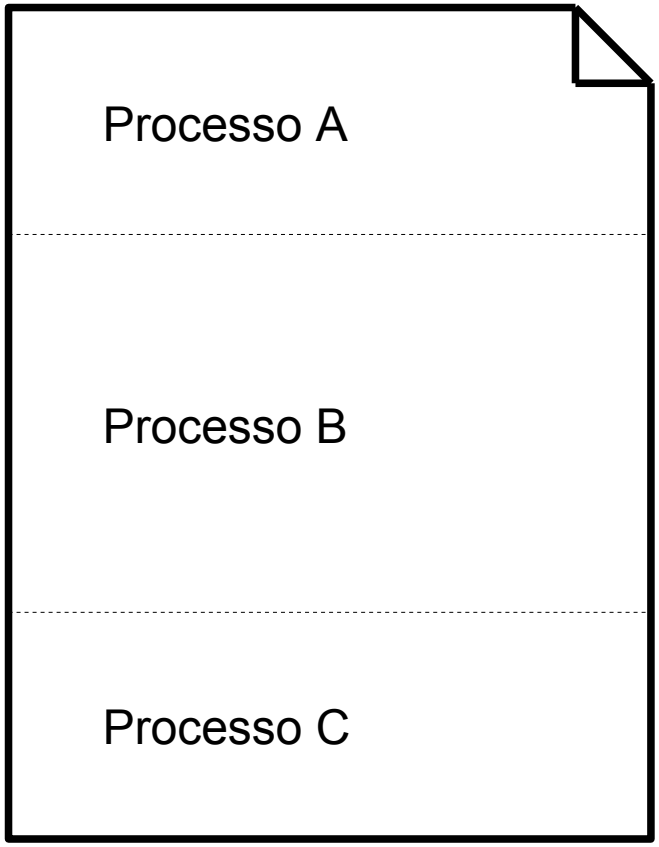


Spazi di indirizzamento



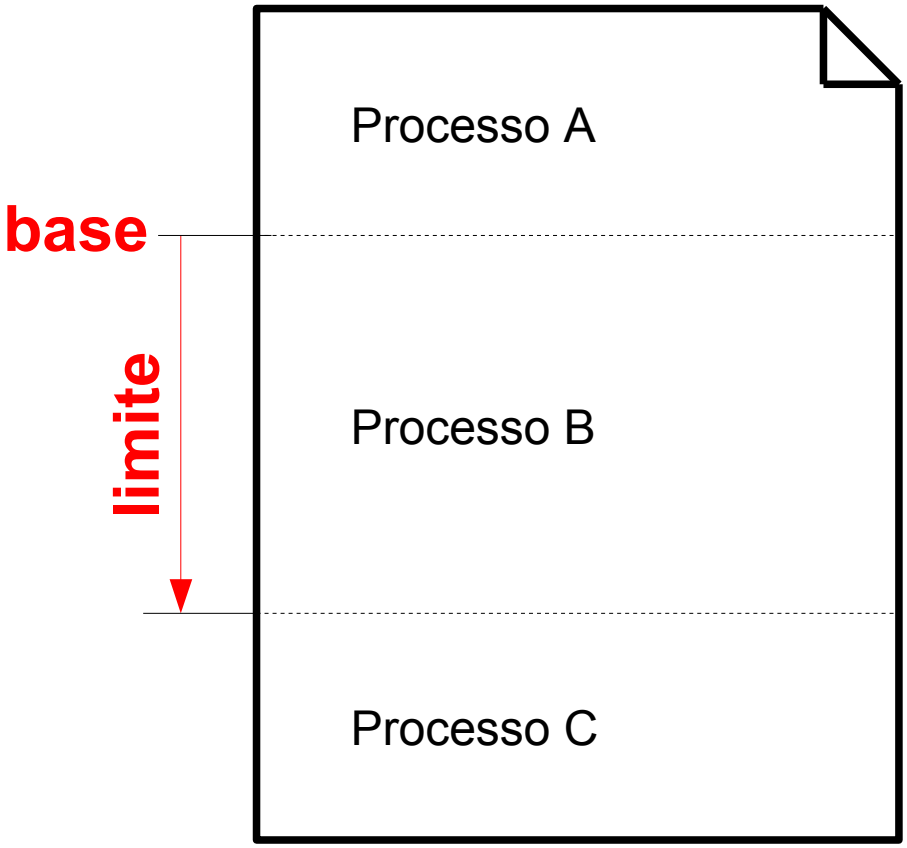
Invece di dare libero accesso a tutta memoria,
ogni processo riceve uno **spazio di indirizzamento**

Spazi di indirizzamento



lo spazio di indirizzamento definisce l'insieme di indirizzi che il processo può utilizzare per accedere alla memoria

Spazi di indirizzamento



Uno spazio di indirizzamento è definito da

- un indirizzo **base** nella memoria fisica
- una dimensione **limite** (quanto è grande lo spazio)

Spazi di indirizzamento

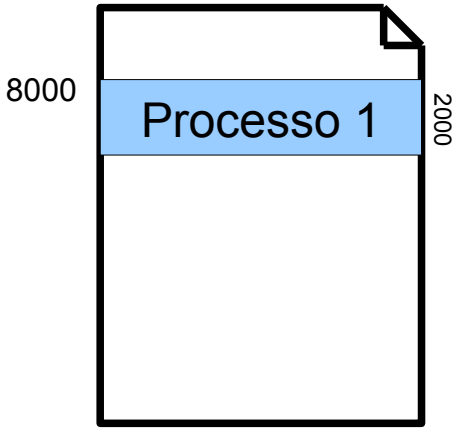
- Quando un processo viene eseguito, base e limite vengono caricati in due registri del processore

- Gli indirizzi codificati in un programma non vengono più considerati come indirizzi fisici ma come **offset** relativi alla base dello spazio di indirizzamento assegnato al processo
  - per ottenere un indirizzo nella memoria fisica **l'offset viene sommato alla base**

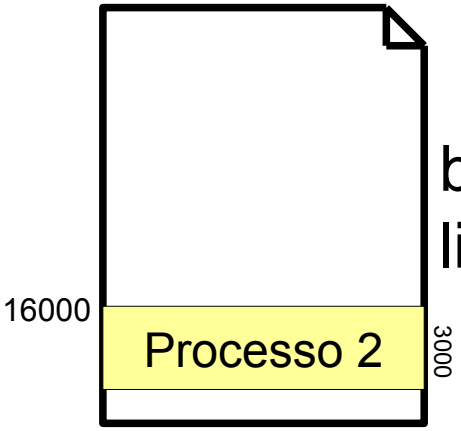
Protezione degli spazi di indirizzamento

- Il sistema controlla anche che l'offset utilizzato non sia maggiore del limite, evitando che un processo possa curiosare o modificare altri spazi di indirizzamento → **protezione della memoria**

Semplificare la multiprogrammazione: spazi di indirizzamento



base: 8000
limite: 2000



base: 16000
limite: 3000

```
MOV registro1, 1981
```

8000+1981

```
MOV registro1, 9981
```

```
MOV registro1, 1981
```

16000+1981

```
MOV registro1, 17981
```

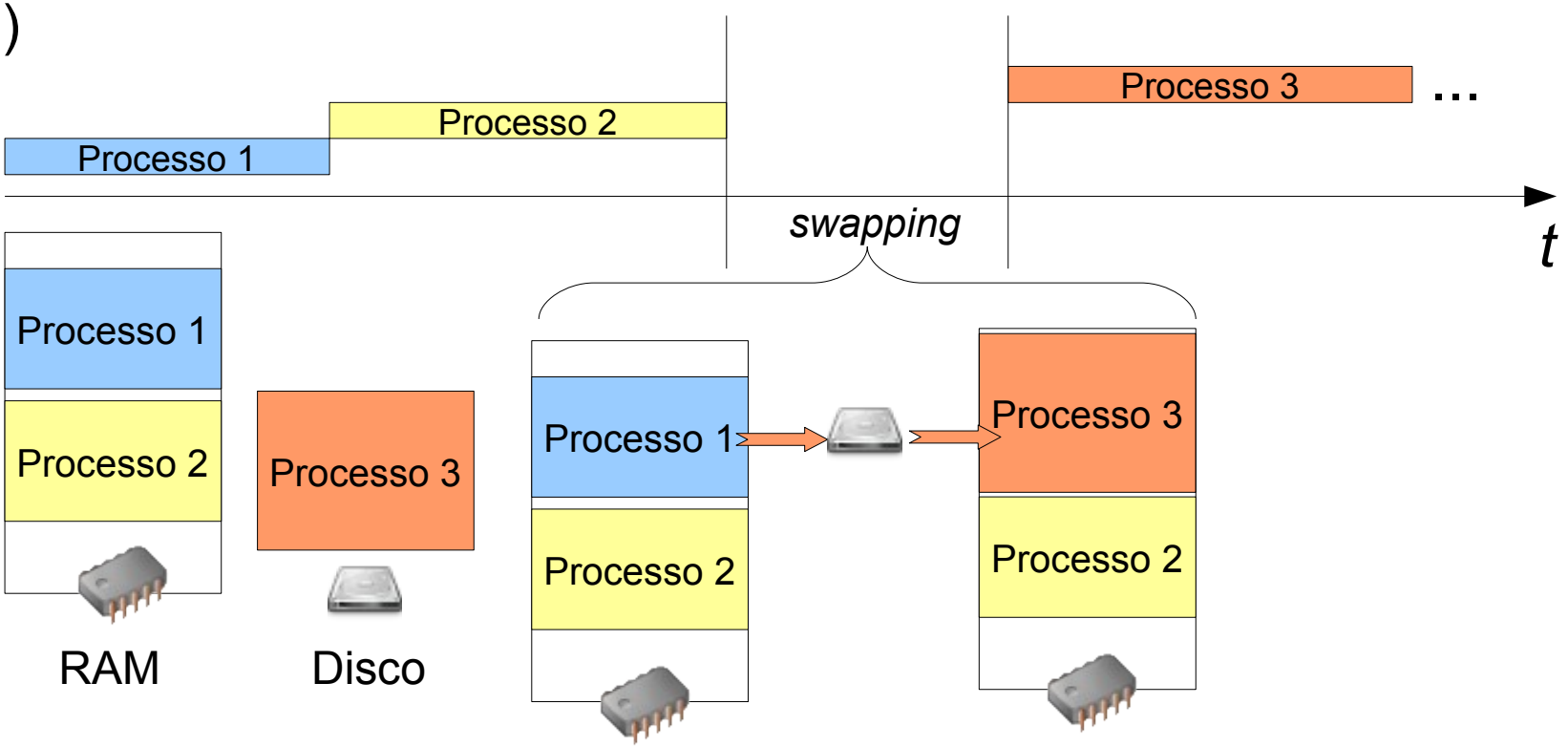
Ovviare alle limitazioni: Swapping e memoria virtuale

- Spesso la memoria RAM richiesta dai processi è maggiore di quella libera disponibile

- Per risolvere questo problema ci sono due soluzioni
  - **swapping**
  - **memoria virtuale**

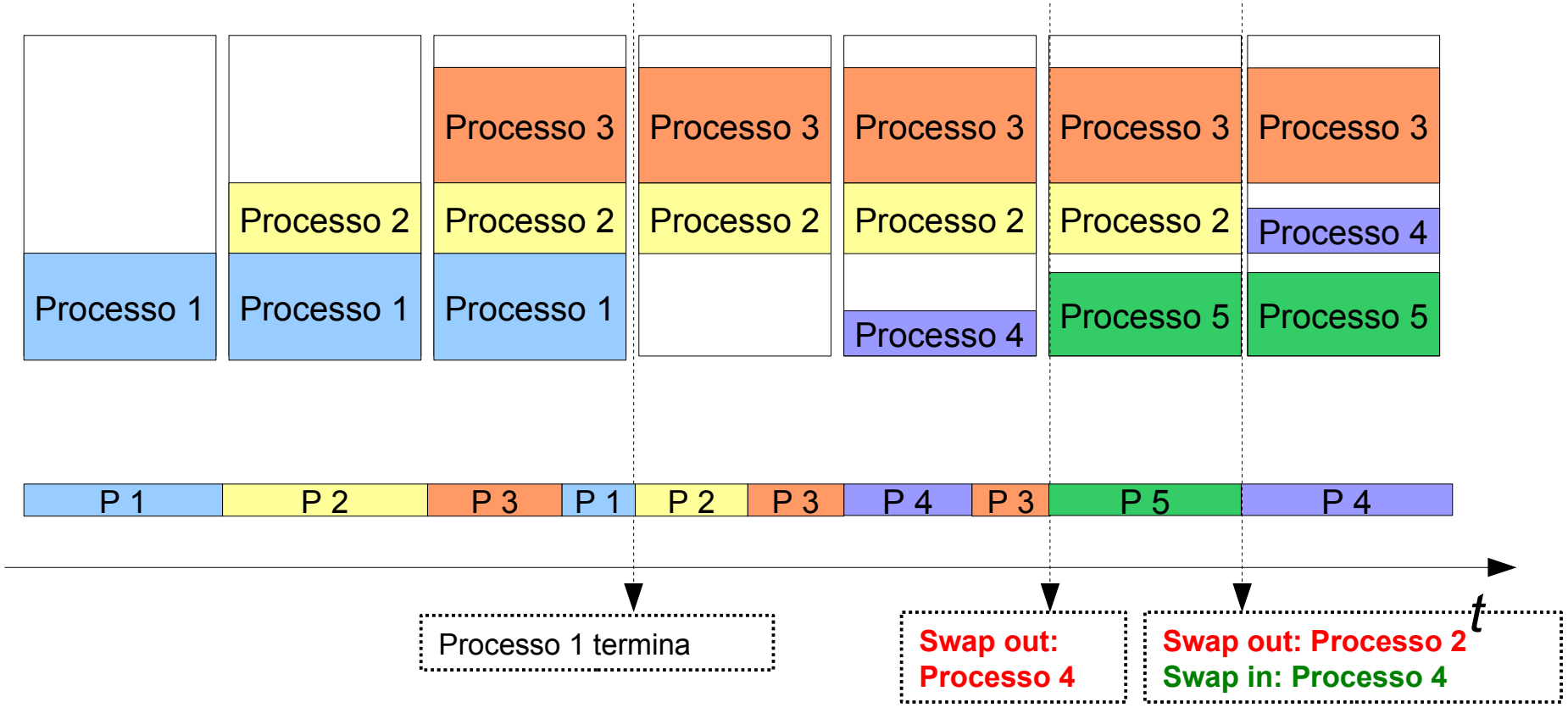
Swapping

- Lo **swapping** (*scambio*) consiste nello spostare la memoria occupata dai processi che non sono correntemente in esecuzione su disco (*swap out*) per poi ricaricarla quando necessario (*swap in*)



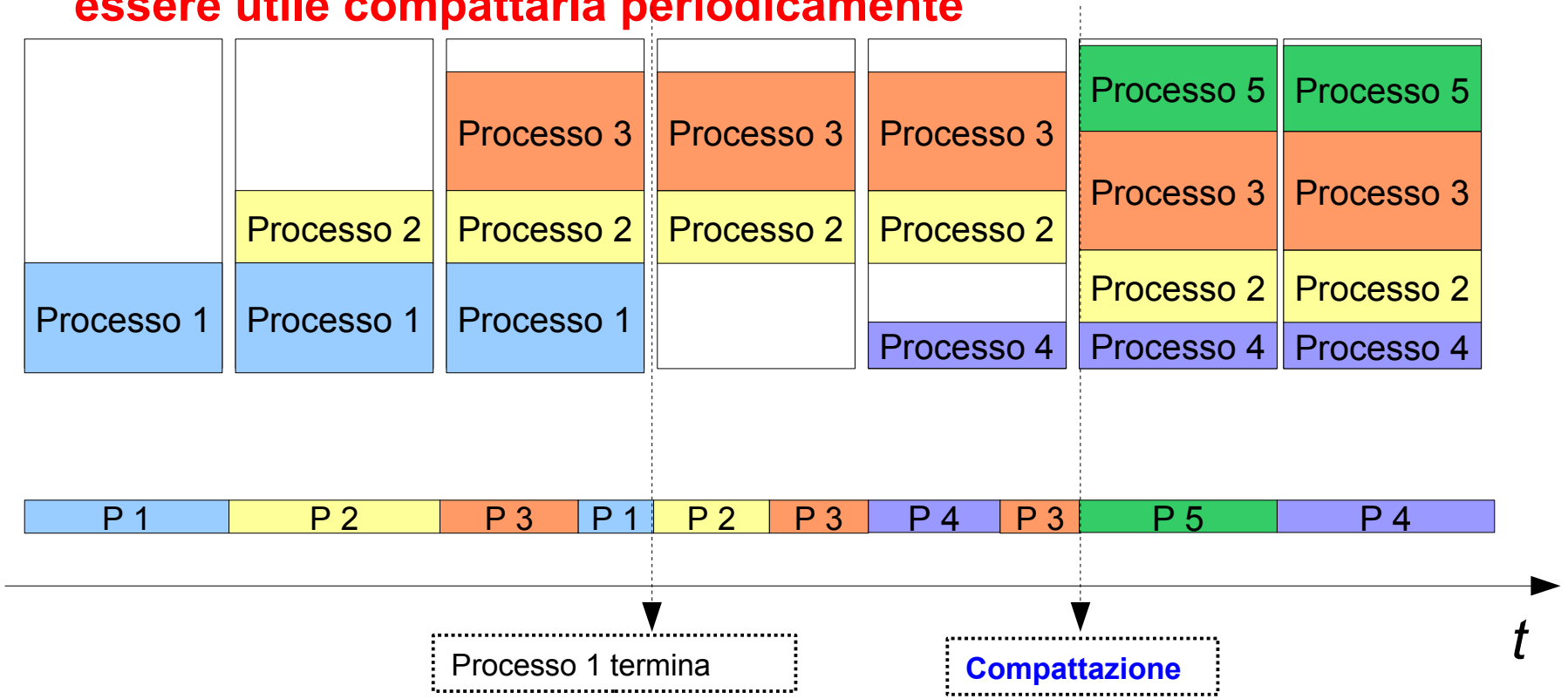
Problemi con lo swapping: frammentazione esterna

- Con lo swapping la memoria può diventare frammentata, causando swap non necessari che rallentano l'esecuzione dei processi



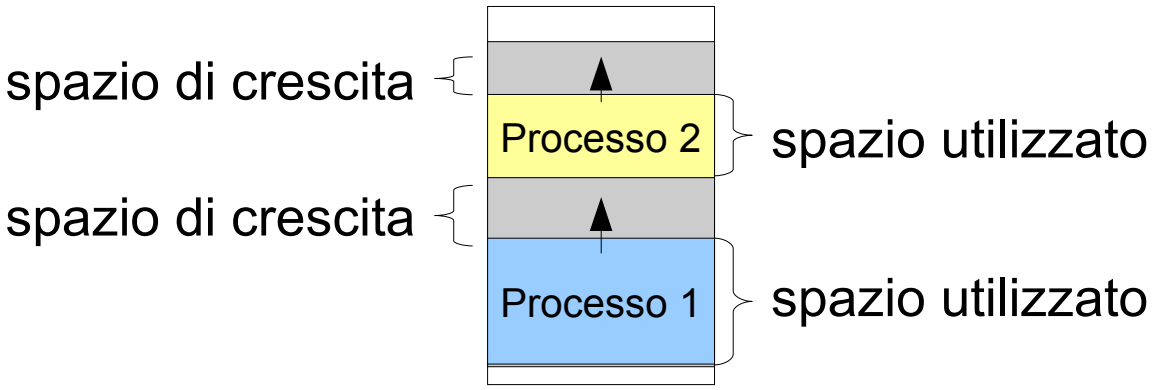
Frammentazione esterna: compattazione

- Con lo swapping la memoria può diventare frammentata, causando swap non necessari che rallentano l'esecuzione dei processi **→ può essere utile compattarla periodicamente**



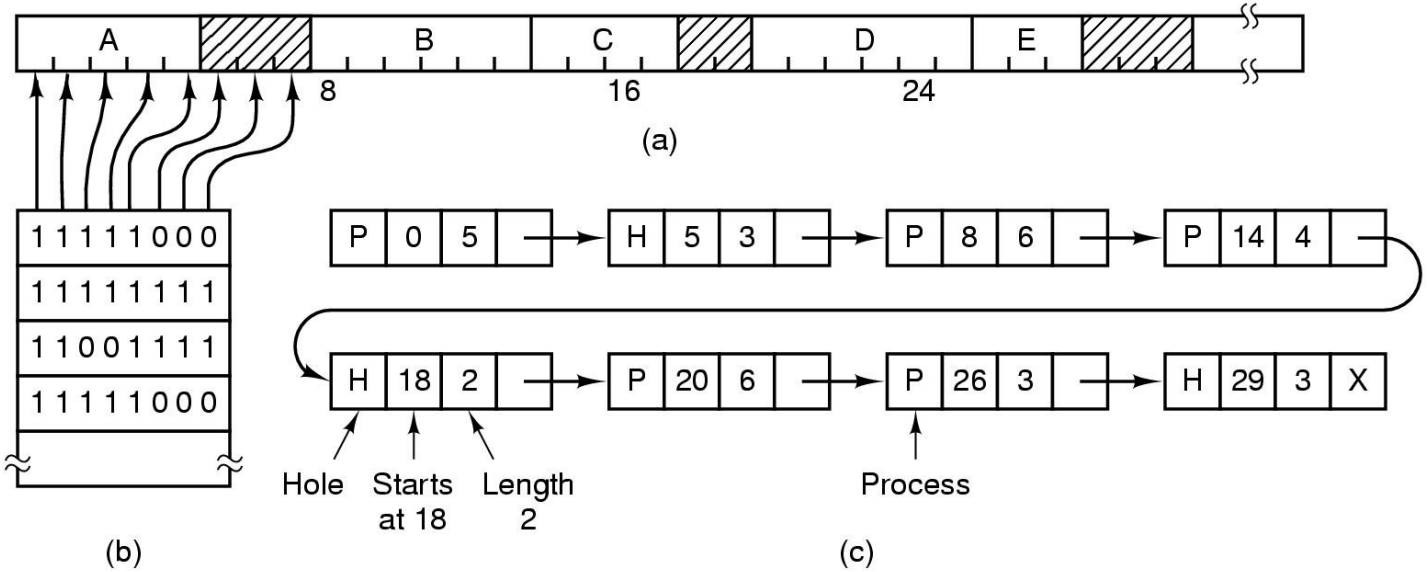
Processi che richiedono più memoria

- Un programma in esecuzione può richiedere memoria dinamicamente...
  - il sistema operativo deve "prevedere" questa situazione e lasciare della memoria disponibile tra quella allocata per evitare swap inutili



Gestione dello spazio libero

- Per sapere dove trovare spazio libero in memoria il sistema operativo deve tener traccia della memoria allocata:
  - **bitmap** (0 = spazio non allocato, 1 = spazio allocato) *(b)*
  - **lista concatenata** *(c)*



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Scelta dello spazio libero

- Quando un processo richiede della memoria il sistema operativo deve trovare un “buco” libero; esistono diversi approcci:
  - **First fit**: il primo slot disponibile di dimensione sufficiente viene scelto
  - **Next fit**: come first fit, ma non ricomincia da capo ogni volta ma continua
  - **Best fit**: cerca dall'inizio alla fine, e utilizza il più piccolo slot disponibile di dimensioni sufficienti
  - **Worst fit**: cerca dall'inizio alla fine e utilizza il più grande slot disponibile
  - **Quick fit**: mantiene delle liste separate per le dimensioni più richieste (es. 4KB, 8KB,...)

Sempre più memoria...

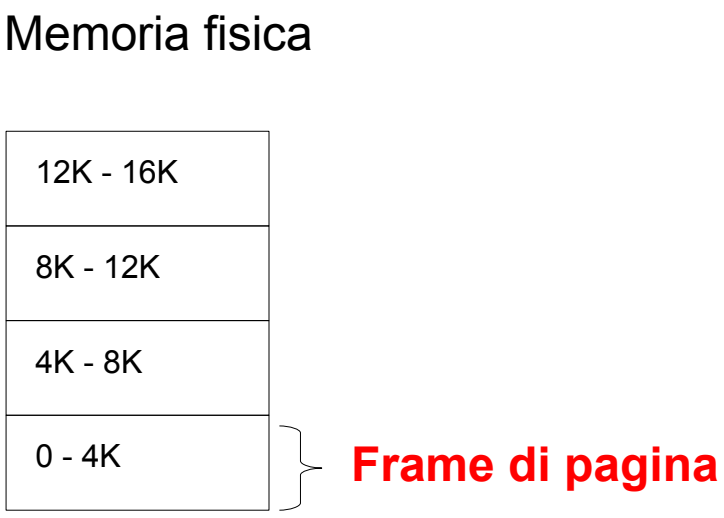
- Sui moderni computer i programmi spesso richiedono più della memoria effettivamente disponibile...
  - astrazioni
  - sistemi multi-utente, sempre più applicazioni
  - audio / video processing (grandi quantità di dati)

Sempre più memoria...

- **Come fare?**
  - tipicamente un programma non utilizza tutta la memoria che ha richiesto nello stesso momento, quindi è possibile dividerla a pezzi che il sistema può gestire indipendentemente con lo swapping
  - visto che dividere grandi programmi in pezzi più piccoli è difficile il compito è lasciato al **sistema operativo, che gestisce un'astrazione detta memoria virtuale**

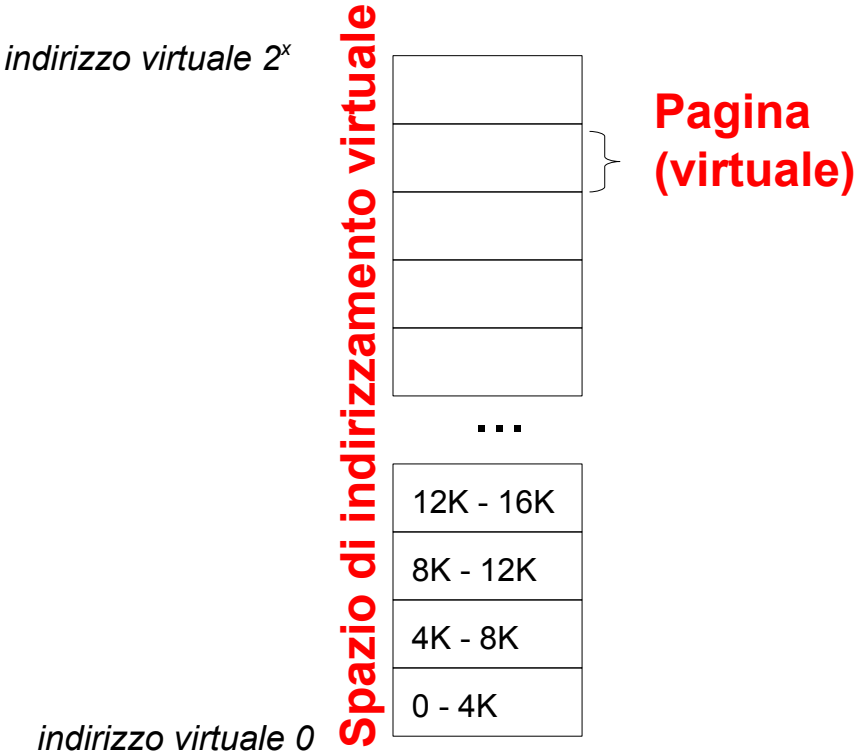
Paginazione

- La memoria fisica (RAM) viene divisa in aree di uguali dimensioni dette **frame di pagina**



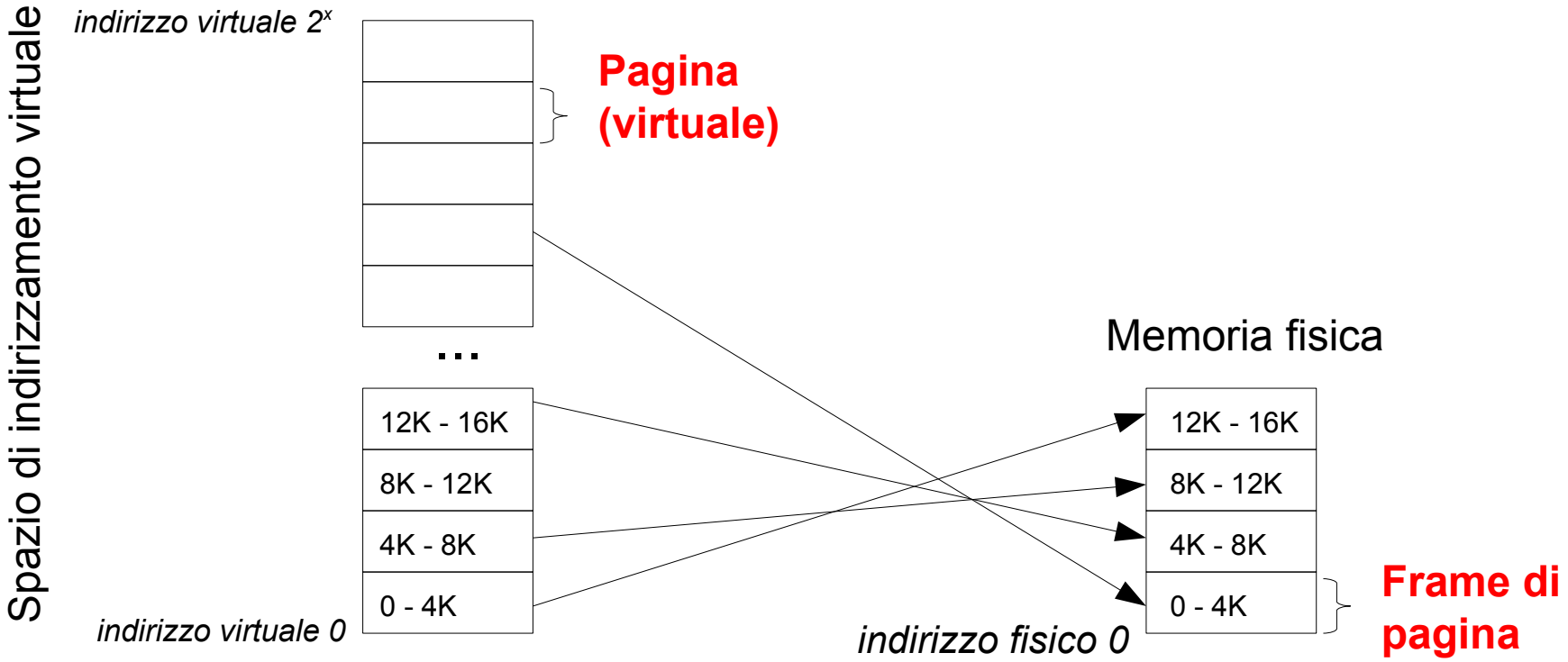
Paginazione

- Lo spazio di indirizzamento di ogni processo viene anch'esso diviso in parti uguali dette **pagine (virtuali)** che hanno la stessa dimensione dei frame di pagina



Paginazione

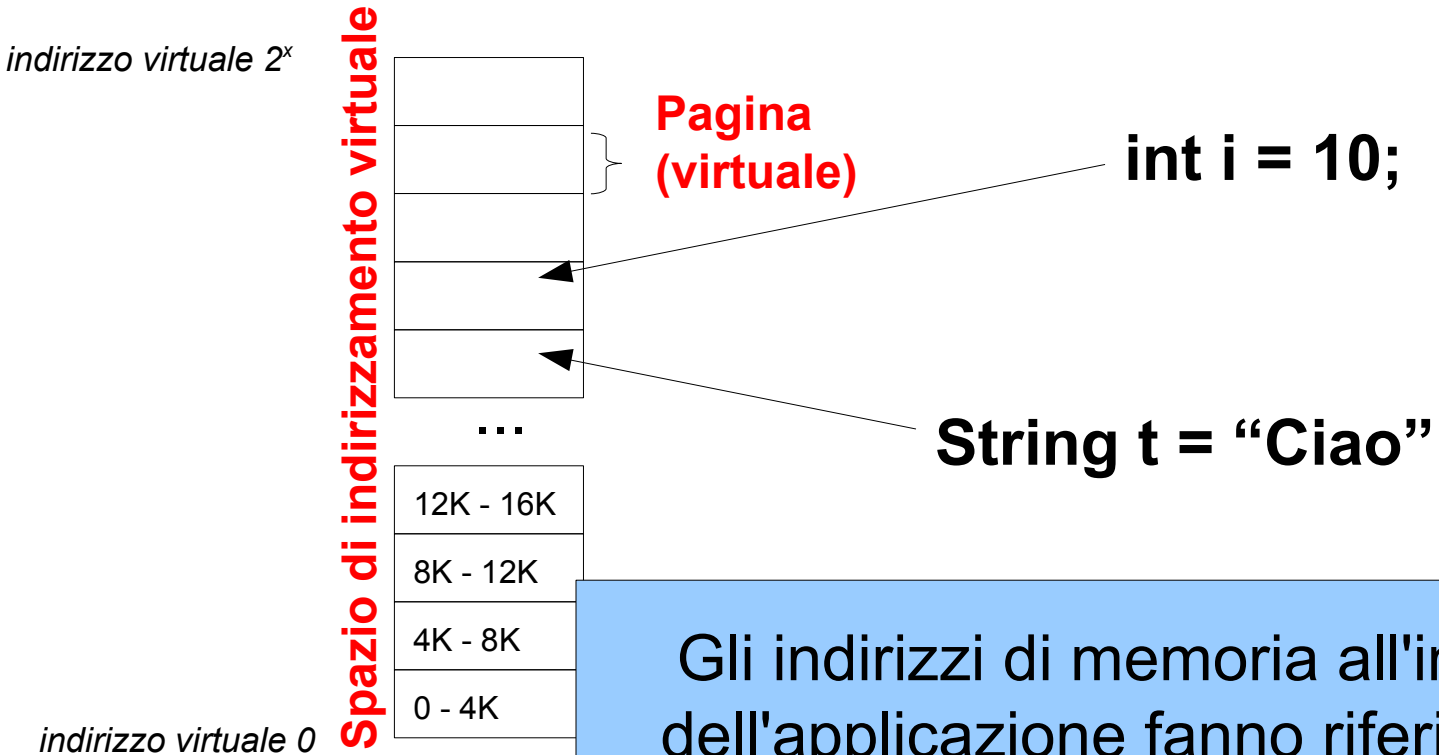
- Le pagine vengono caricate “on demand” (cioè quando servono) nei frame di pagina



Memoria virtuale: Paginazione

- Le pagine hanno una dimensione tipica di 4 KB
  - Ogni pagina contiene un intervallo contiguo di indirizzi
- Il numero di pagine utilizzate da un processo dipende dalla memoria allocata e dalla dimensione di ogni pagina
  - Il processo ha virtualmente accesso a tutto lo spazio di indirizzamento, ma solo le pagine con indirizzi referenziati sono realmente usate
- Il numero di frame di pagina dipende dalla quantità di memoria fisica disponibile e dalla dimensione di ogni pagina

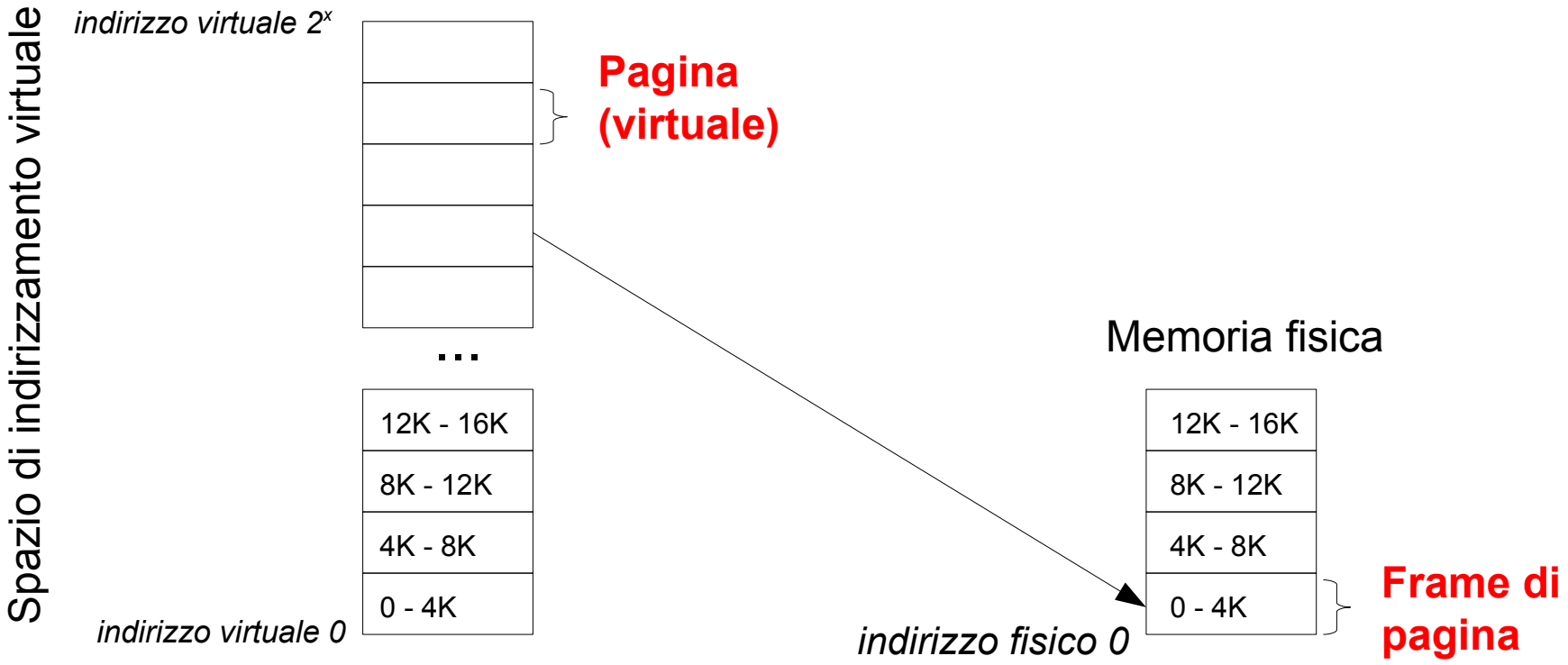
Indirizzi virtuali



Gli indirizzi di memoria all'interno dell'applicazione fanno riferimento allo **spazio di indirizzamento virtuale**

Paginazione

- Le pagine con i dati che un processo sta utilizzando vengono copiate in un frame di pagina



Paginazione

- È possibile che **non tutte le pagine di un processo siano nella RAM**: quelle che non sono correntemente utilizzate possono essere copiate su disco (*swap out* su partizione o file di swap)

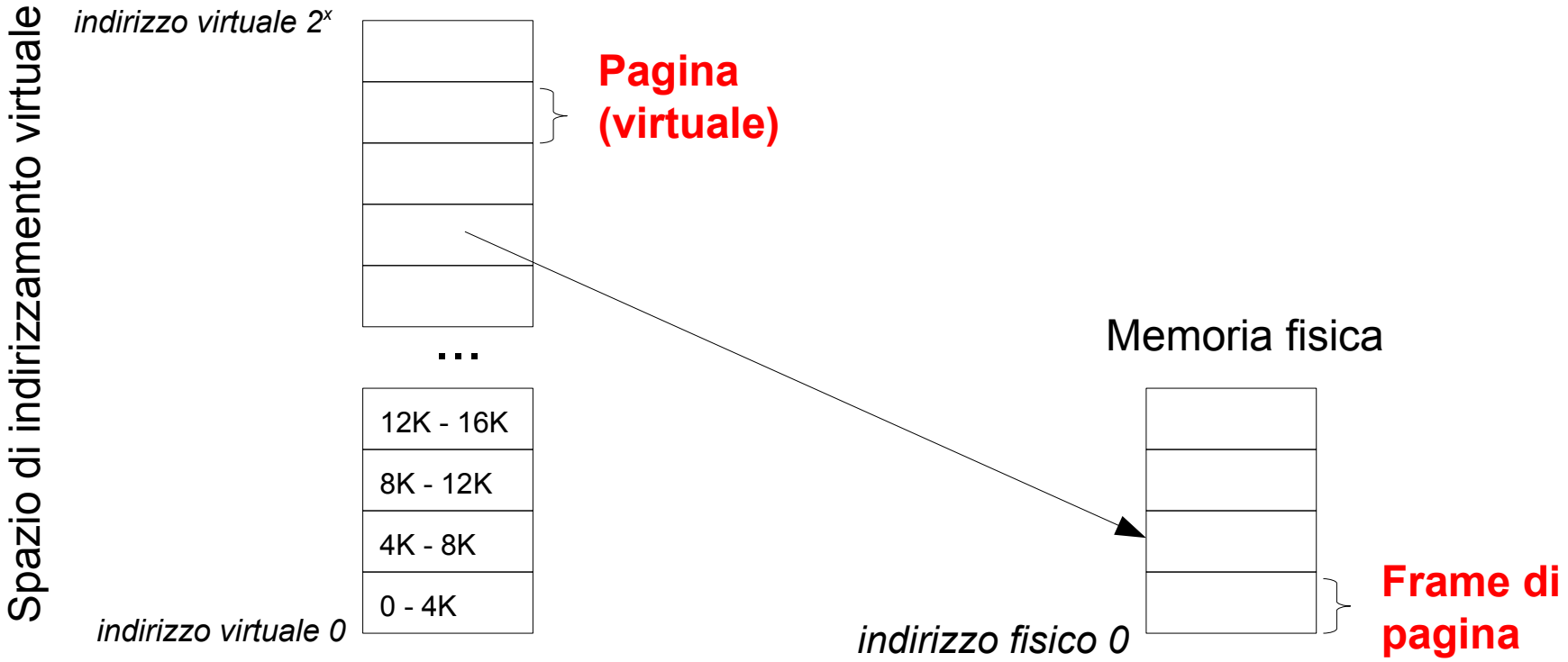
- I frame di pagina possono essere occupati anche da pagine appartenenti a processi diversi

Memoria fisica

Pagina 5, processo B
Pagina 5, processo D
Pagina 7, processo A
Pagina 12 processo A

Paginazione: swap in

- Quando un'applicazione accede a un dato che si trova in una pagina che non è attualmente in memoria il sistema operativo deve caricarla (*swap in*)



Come funziona la paginazione: traduzione degli indirizzi

- Ogni volta che un'applicazione fa riferimento a un indirizzo (nella memoria virtuale!) dobbiamo effettuare una "traduzione" per ottenere un indirizzo fisico

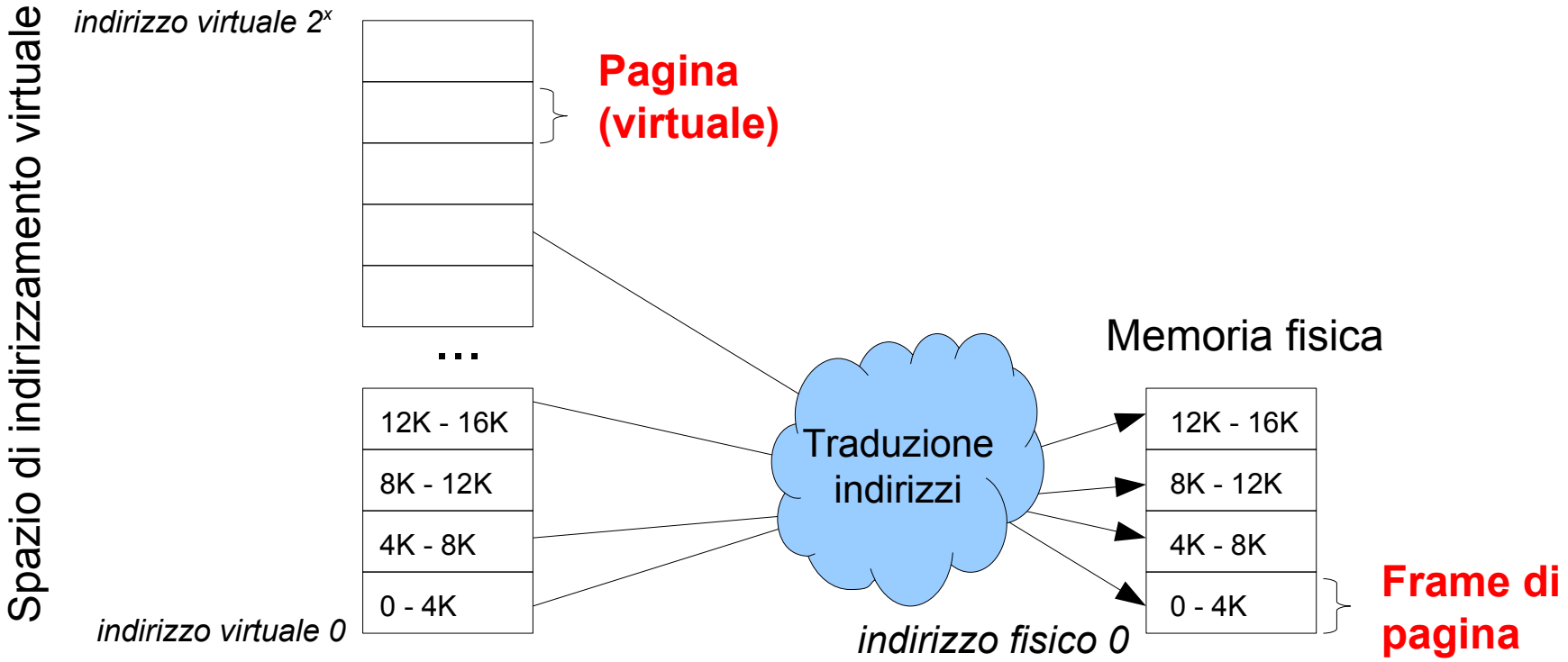


Tabella delle pagine

- Per ogni processo il sistema operativo mantiene una **tabella delle pagine** che serve a tradurre gli indirizzi virtuali in indirizzi nella memoria fisica, associando un **numero di pagina** con **l'indirizzo base del frame in memoria**

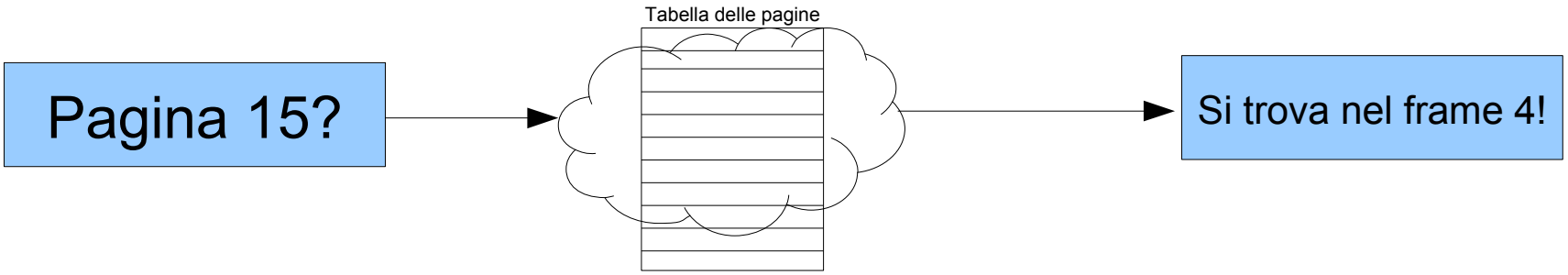


Tabella delle pagine

- Per ogni pagina, **un bit indica se la pagina è in memoria oppure no**

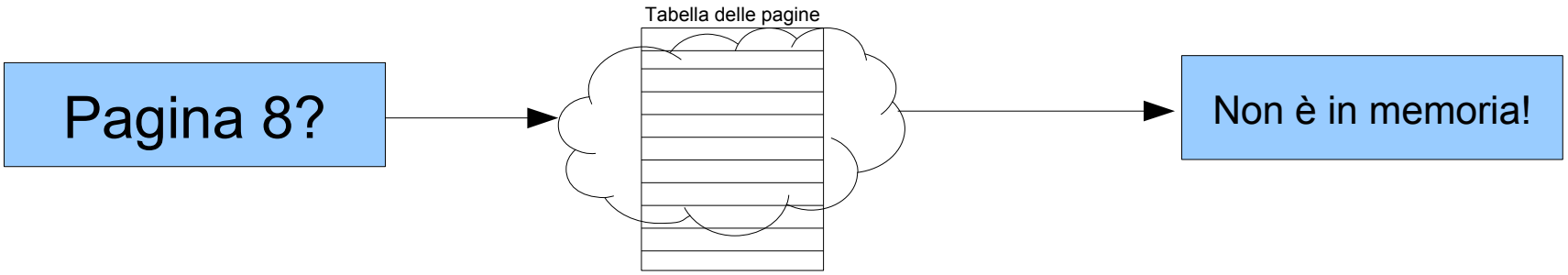


Tabella delle pagine

- Per ogni pagina, **un bit indica se la pagina è in memoria oppure no**
  - se non è in memoria il sistema operativo riceve un segnale (*page fault*) e carica la pagina dal disco
  - se la pagina non esiste neppure su disco il processo riceve un segnale di terminazione

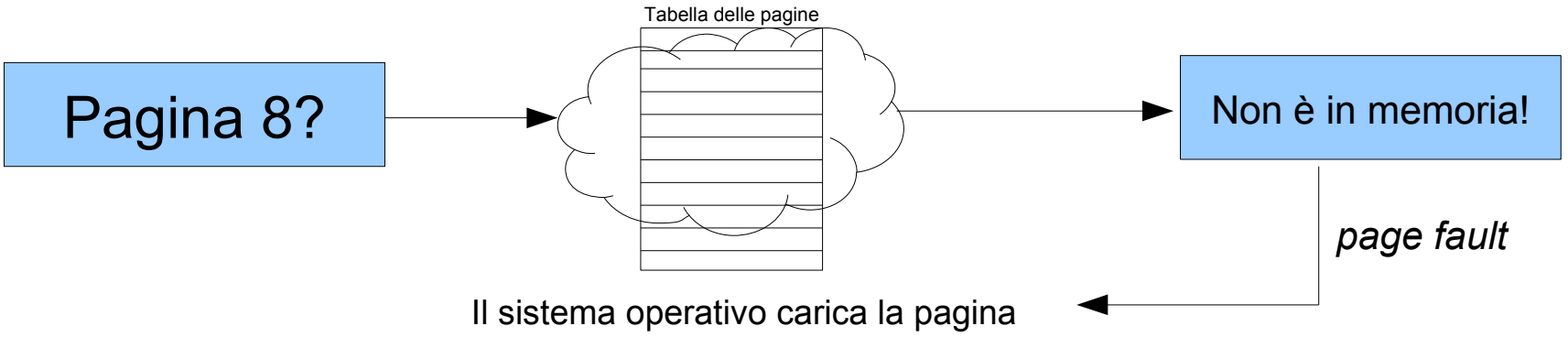
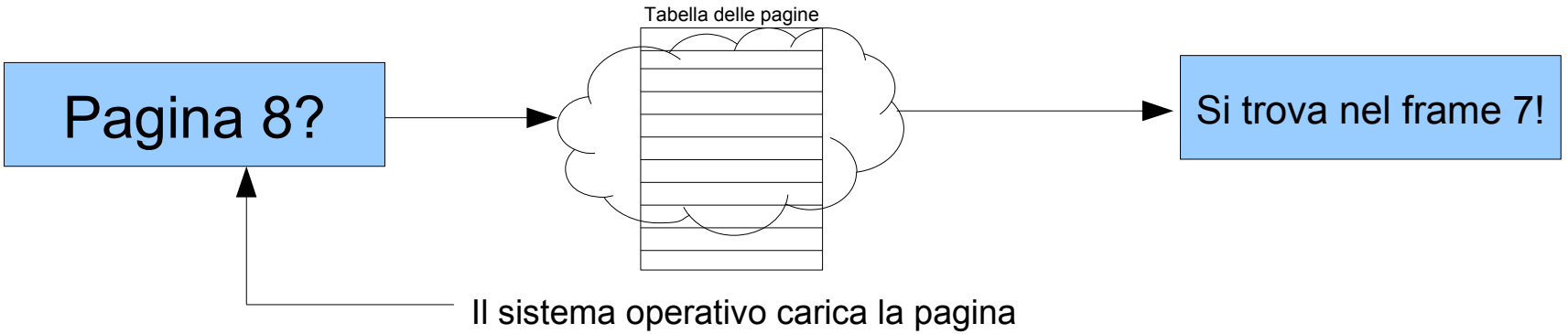
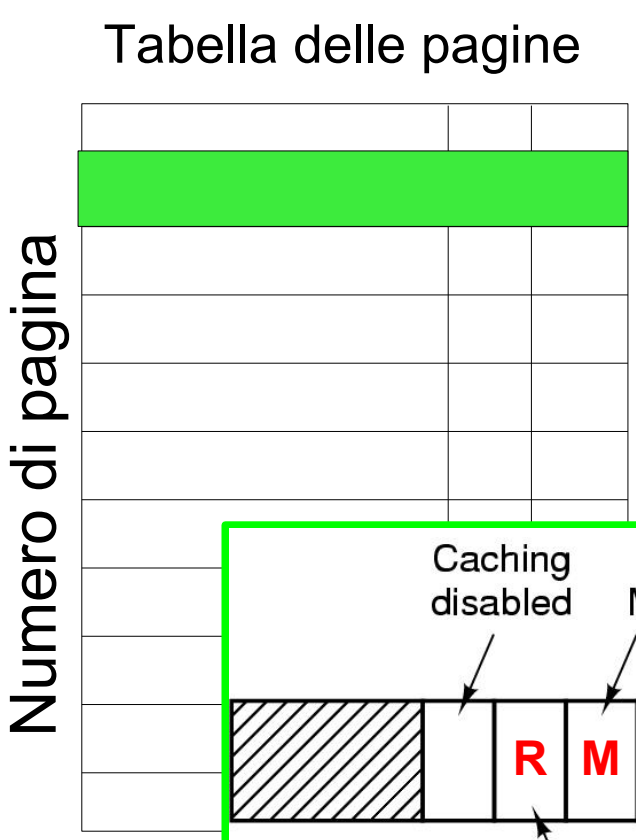


Tabella delle pagine

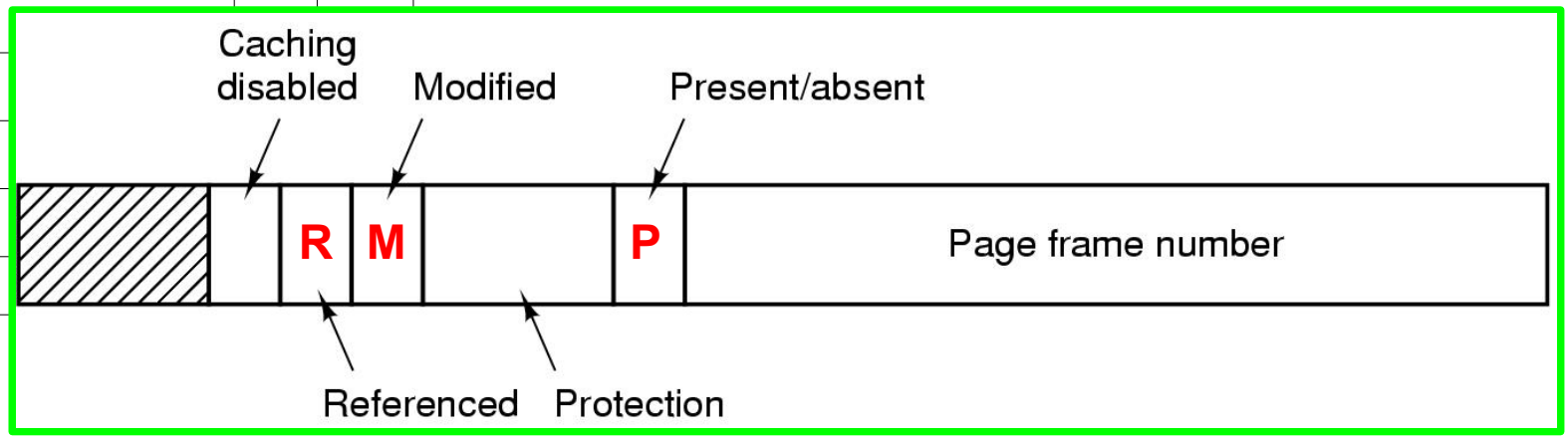
- Per ogni pagina, **un bit indica se la pagina è in memoria oppure no**
  - se non è in memoria il sistema operativo riceve un segnale (*page fault*) e carica la pagina dal disco
  - se la pagina non esiste neppure su disco il processo riceve un segnale di terminazione



Cosa c'è nella tabella delle pagine

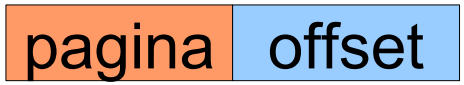


Ogni elemento della tabella associa un numero di pagina a un frame.
Un bit (**P**) indica se la pagina è in memoria oppure no. Il bit (**R**) indica se la pagina è stata recentemente referenziata, il bit (**M**) indica se è stata modificata (**dirty bit**). I bit di protezione indicano i permessi (RW) e l'accesso (utente/supervisor).



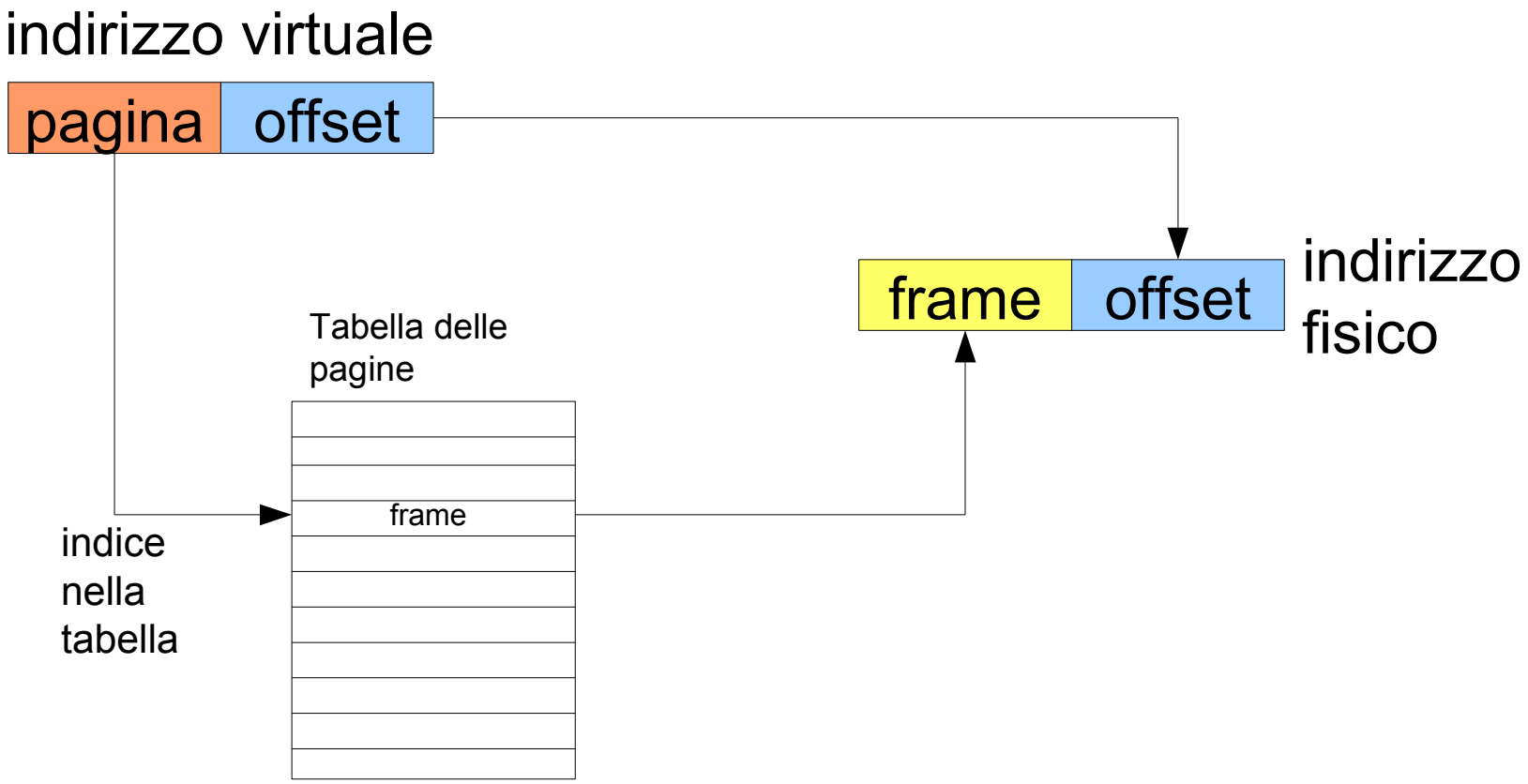
Traduzione degli indirizzi

- Per ritrovare gli elementi nella tabella delle pagine l'indirizzo virtuale viene diviso in due parti:



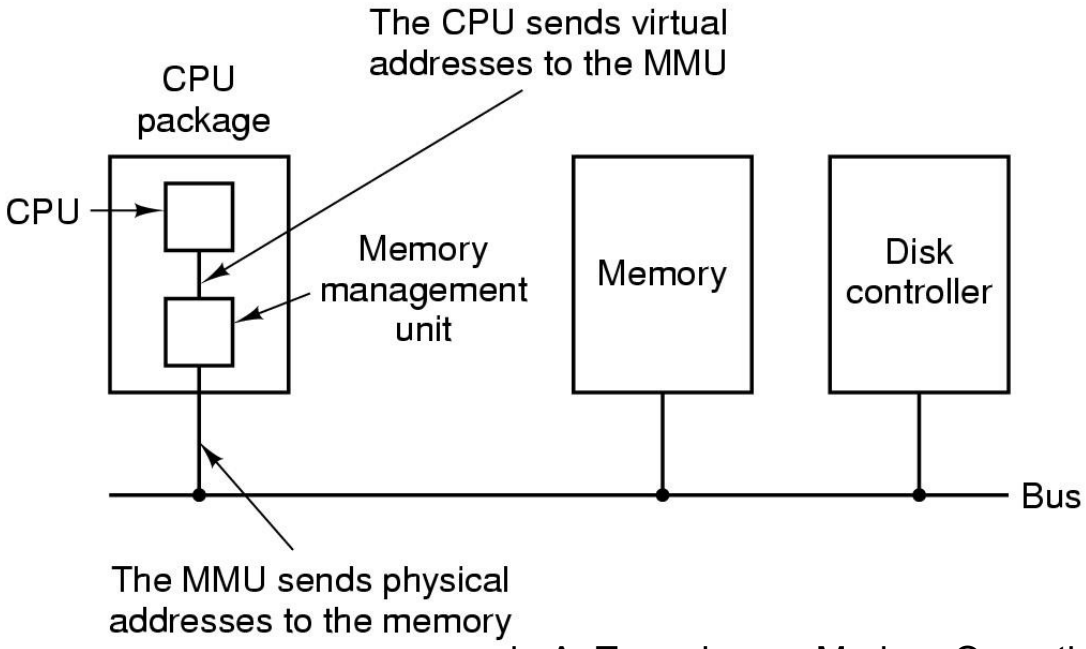
- Pagina è il numero di pagina virtuale, da usare come indice nella tabella
- Offset è un indice all'interno della pagina

Traduzione degli indirizzi



Traduzione degli indirizzi

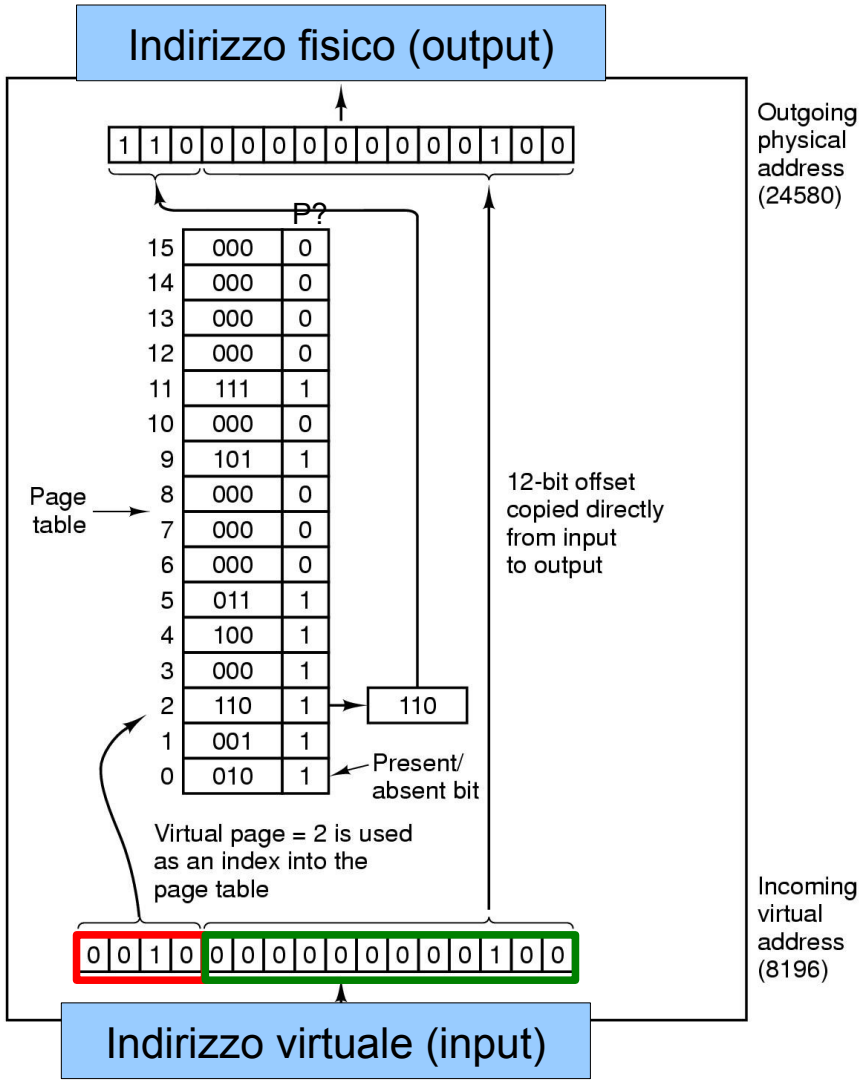
- La traduzione viene affidata a un componente hardware chiamata **MMU** (Memory Management Unit)
  - Un registro nella CPU (CR3) contiene l'indirizzo della tabella delle pagine per il processo corrente



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

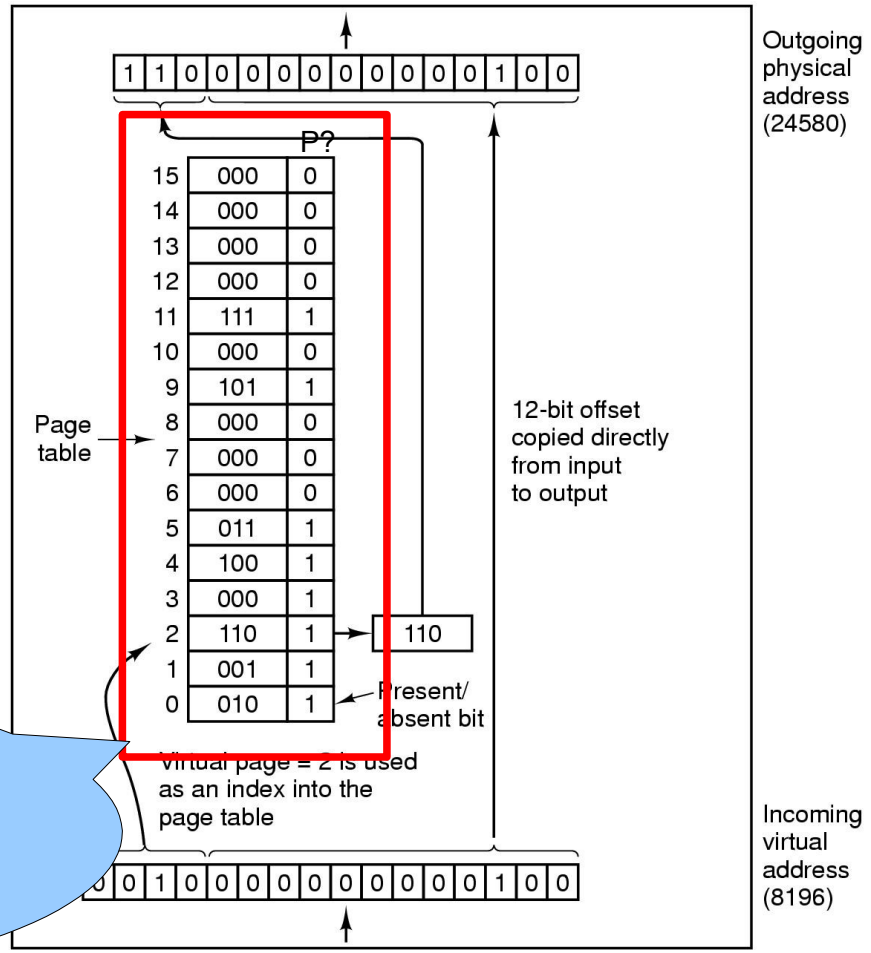
Esempio di traduzione

numero di pagina
***offset* all'interno della pagina**



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Esempio di traduzione



La pagina n° 2 (0010b)
si trova nel frame 6 (110b)

da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Swapping di pagine

- Il sistema operativo può spostare su disco le pagine che non sono correntemente utilizzate, liberando la memoria fisica (**page swapping**)
  - quando un processo fa riferimento a una pagina che non è in memoria (**bit = 0**), il sistema operativo viene notificato con un *page fault* e provvede a caricarla dal disco
    - se non ci sono frame liberi sarà necessario togliere una delle pagine in memoria e salvarla su disco

Pagine dirty

- Non sempre è necessario ri-salvare una pagina su disco quando vogliamo liberare la memoria fisica
  - se la pagina è già sul disco e non è stata modificata dall'ultimo "swap in"
  - altrimenti la pagina è detta "**dirty**" e il sistema deve riscriverla su disco
- Nella tabella delle pagine viene utilizzato un bit (**dirty bit**) per indicare se il contenuto è stato modificato oppure no

Trashing

- Solitamente il page swapping è abbastanza veloce perché coinvolge solo alcune pagine

- Se la memoria fisica a disposizione non è sufficiente il numero di page fault può essere molto alto, e il sistema operativo inizia a fare swapping aggressivo
  - si parla di trashing

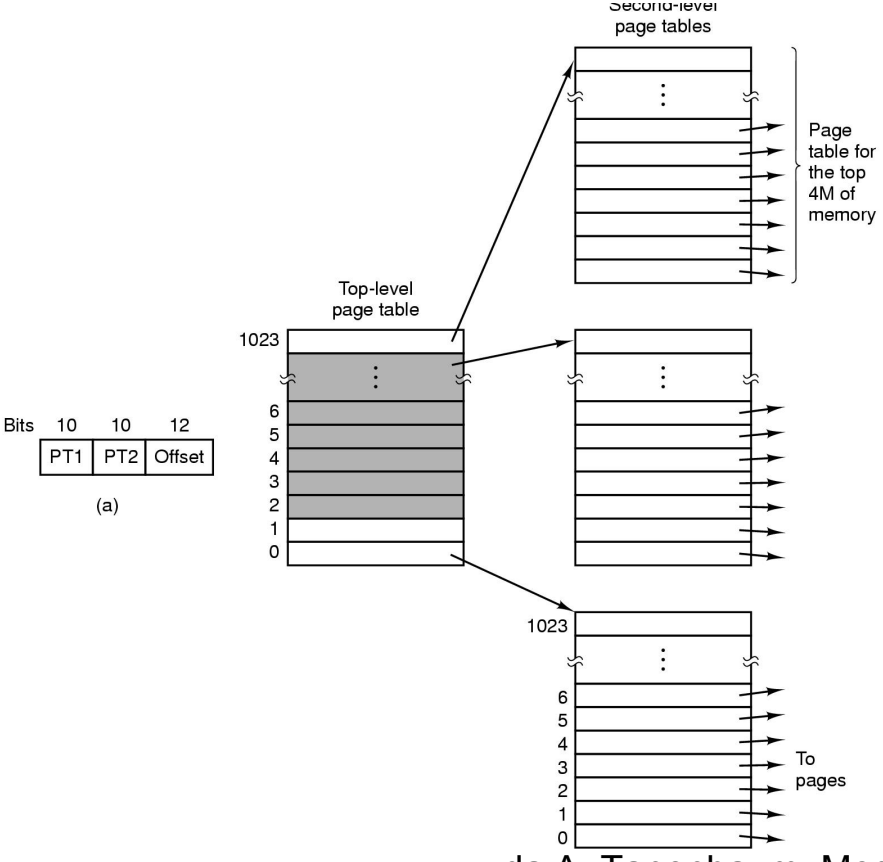
TLB

- Con la paginazione ogni indirizzo virtuale deve “subire” la fase di traduzione
  - Questo necessita l'accesso costante alla tabella delle pagine: **lento!**
- Per velocizzare la traduzione esiste una componente hardware che mantiene una cache di alcune elementi della tabella:
  - **TLB (Translation Lookaside Buffer)**

- Il sistema operativo è incaricato di gestire questo buffer, per esempio per cancellarne il contenuto quando avviene un context-switch (**TLB flush**)

Tabella delle pagine a più livelli

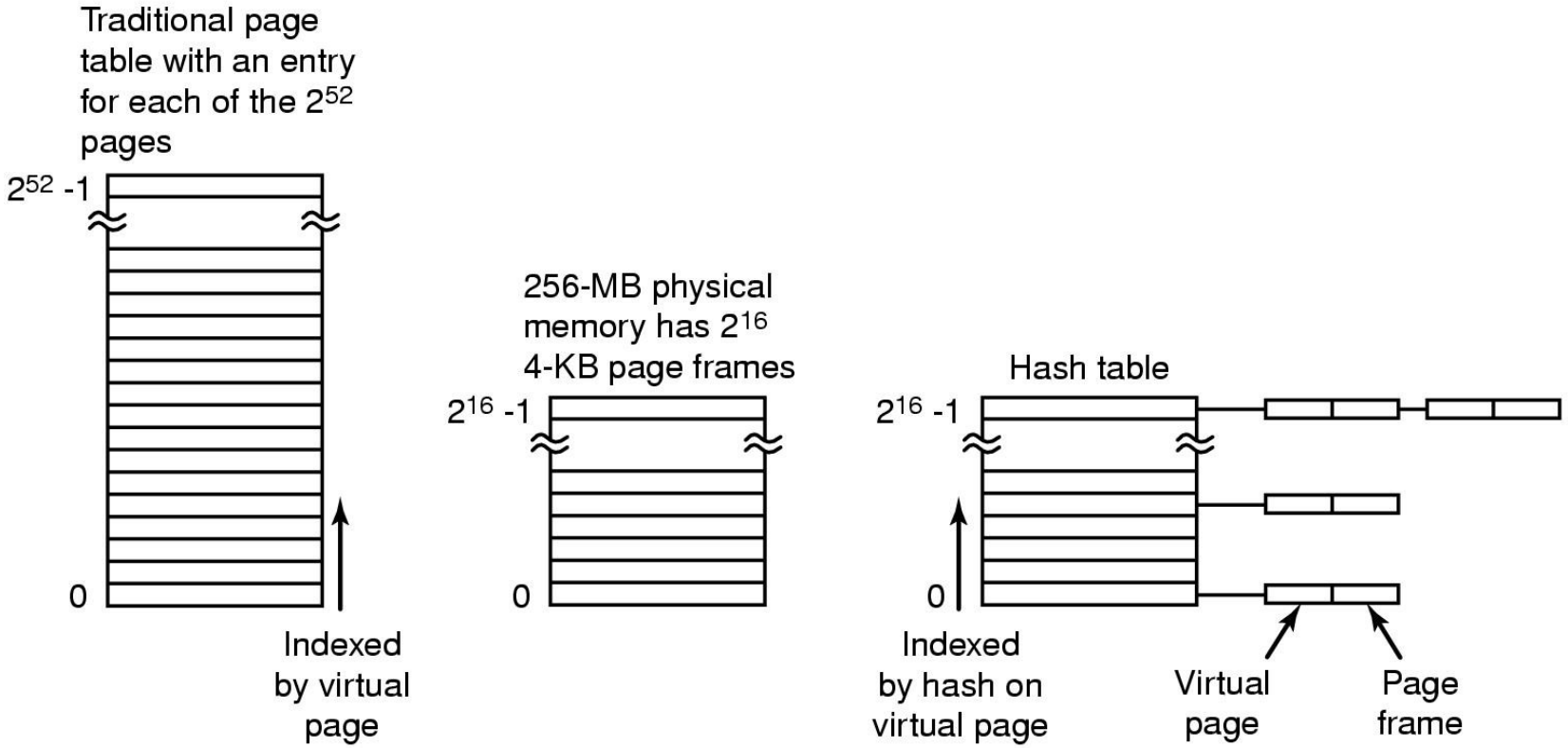
- Se lo spazio di indirizzamento è molto ampio il numero di pagine può diventare elevato; per questo motivo si possono usare delle tabelle di pagine a più livelli.



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Tabelle inverse

- Un'altro modo per evitare di avere tabelle troppo ampie è utilizzare una tabella invertita, con un elemento per ogni frame.



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Algoritmi di rimpiazzamento delle pagine

- Quando è necessario "liberare" la memoria fisica, un algoritmo deve scegliere quale pagina spostare su disco
  - esempi:
    - **FIFO** (First In First Out)
    - Algoritmo della **seconda opportunità**
    - **LRU** (Least Recently Used): sposta la pagina che è stata meno recentemente utilizzata
    - **Working Set**: cerca di mantenere le pagine più frequentemente utilizzate in memoria

L'algoritmo ottimale

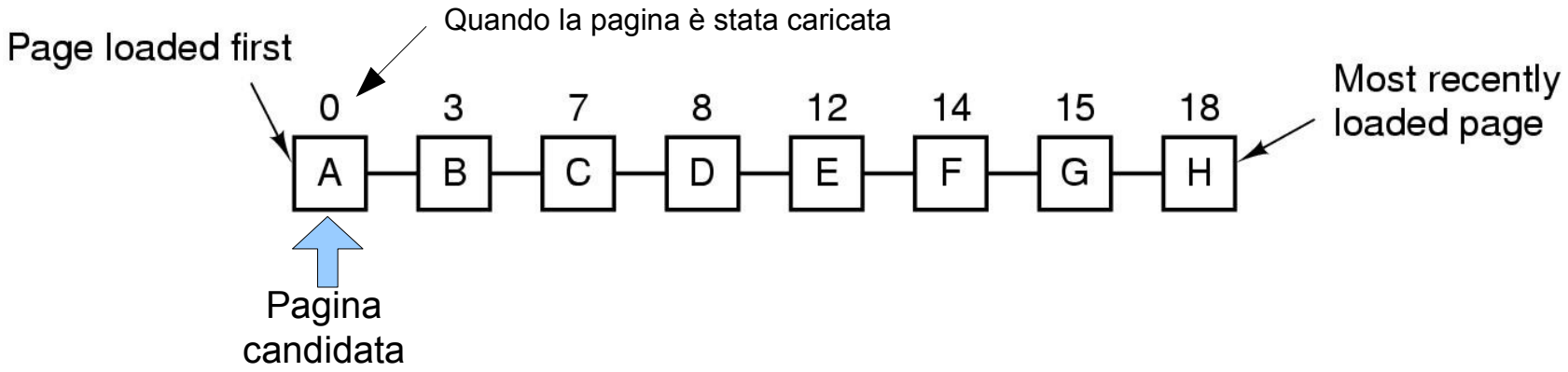
- L'algoritmo ottimale cerca di minimizzare il numero di page fault
  - Questo richiede una conoscenza delle pagine che verranno richieste nel futuro...
  - Se facessimo girare un programma una prima volta potremmo analizzare il suo comportamento, in modo da ottimizzare il caricamento delle pagine nelle successive esecuzioni... sempre che il comportamento sia esattamente lo stesso!
  - In generale questo non è possibile (e non implementabile), quindi ci accontentiamo di approssimazioni ...

NRU (Not recently used, non usata di recente)

- Come abbiamo visto, ad ogni pagina sono associati due bit, R e M
- L'algoritmo NRU utilizza questi bit come segue:
  - Quando un processo viene lanciato, i bit R e M delle sue pagine sono a 0
  - Periodicamente il bit R è rimesso a 0, per tener traccia delle pagine che sono state referenziate recentemente
  - Quando devo rimpiazzare una pagina, tutte le pagine caricate vengono classivicate secondo i valori dei bit R e M corrispondenti:
    - **Classe 0: non referenziata, non modificata**
    - **Classe 1: non referenziata, modificata**
    - **Classe 2: referenziata, non modificata**
    - **Classe 3: referenziata modificata**
  - L'algoritmo sceglie una pagina a caso tra quelle nelle classi più basse

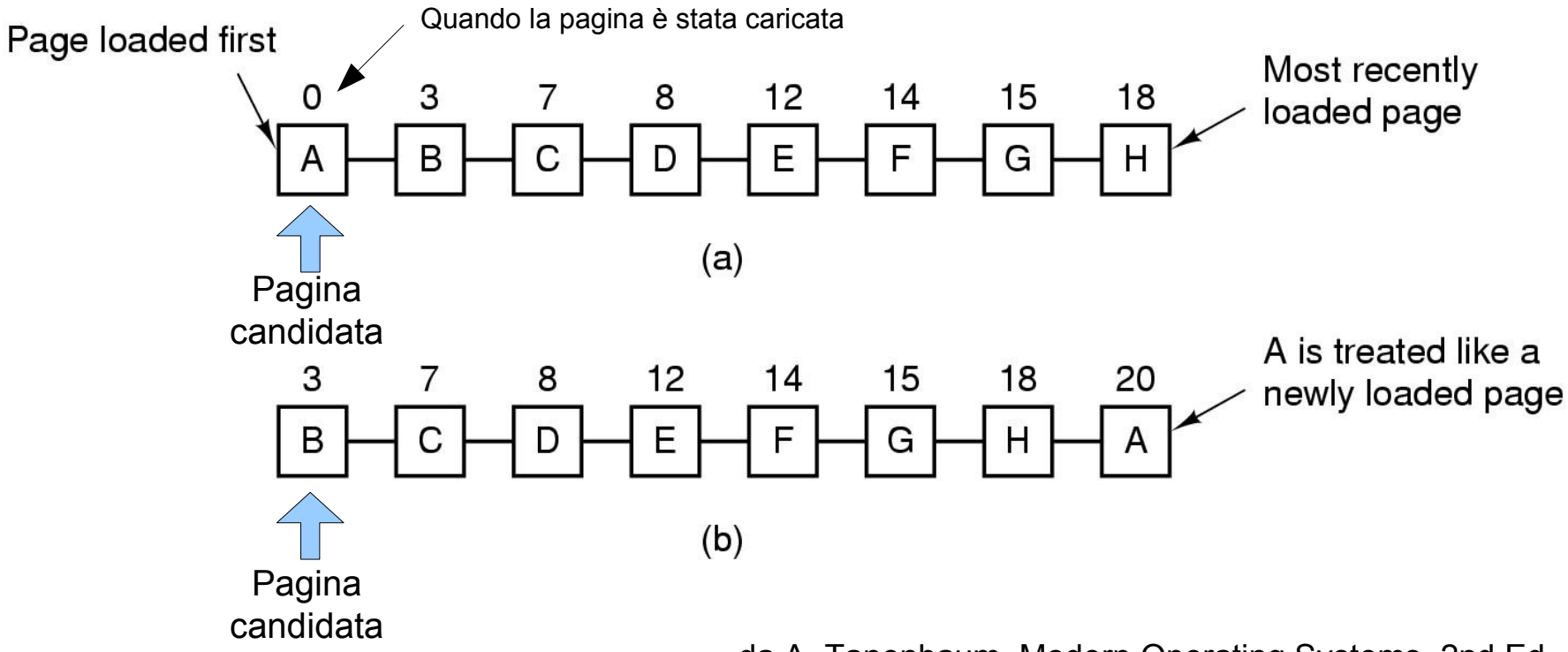
FIFO

- Quando le pagine vengono caricate nelle frame creiamo una lista FIFO
  - La pagina che è stata caricata per prima è anche quella candidata per essere rimossa



Algoritmo della seconda opportunità

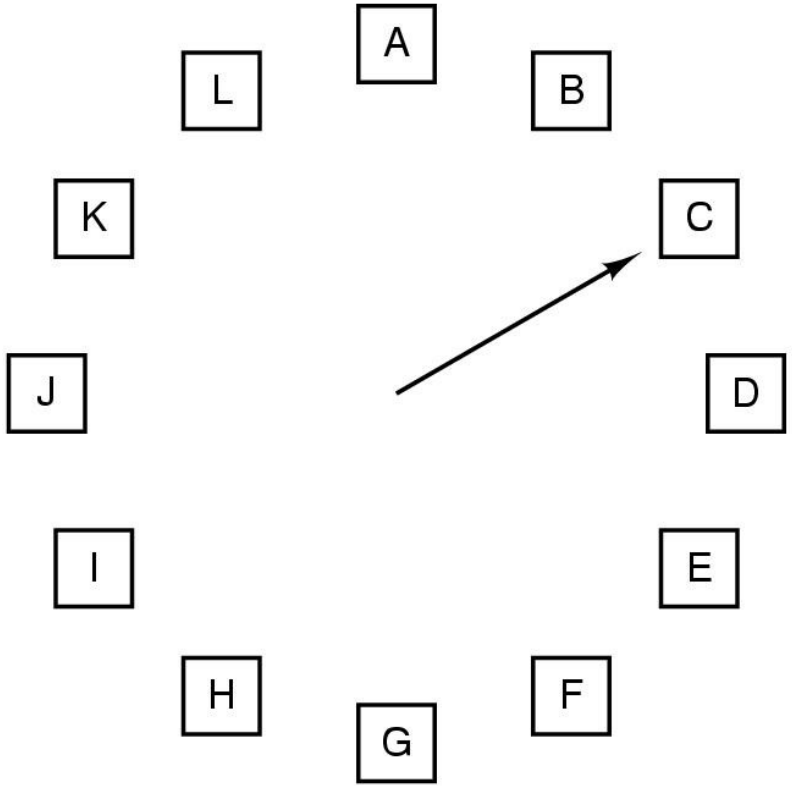
- Per non eliminare una pagina utilizzata frequentemente, guardiamo il bit **R**:
  - Se il bit **R** è 1, lo mettiamo a 0 e rimettiamo la pagina in fondo alla lista
  - Se il bit **R** è 0, la pagina viene rimossa dal frame



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Algoritmo dell'orologio

- Implementazione efficiente dell'algoritmo della seconda opportunità
  - Non richiede l'aggiornamento di puntatori come in FIFO



Quando ho un page fault, guardo la pagina che sta sotto la lancetta. Se il bit **R**=0 tolgo la pagina, altrimenti azzero **R** e avanzo.

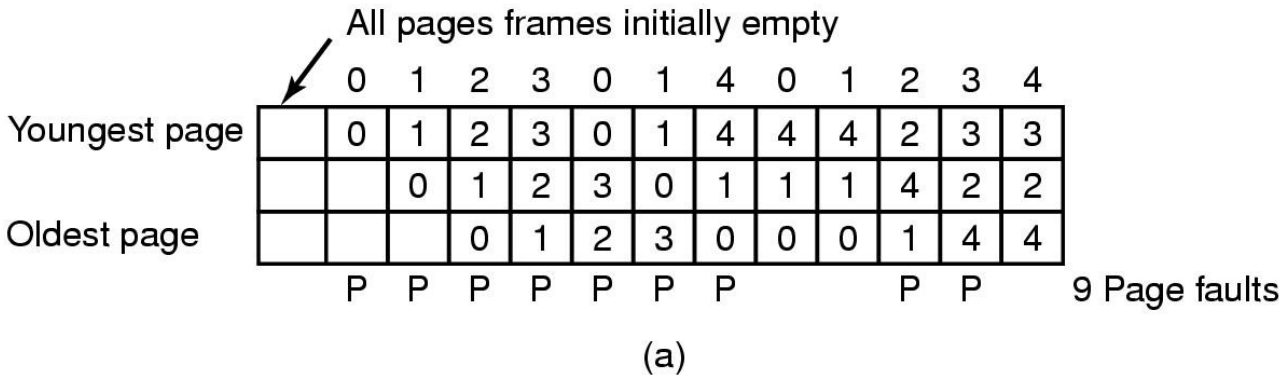
da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

LRU (Least Recently Used)

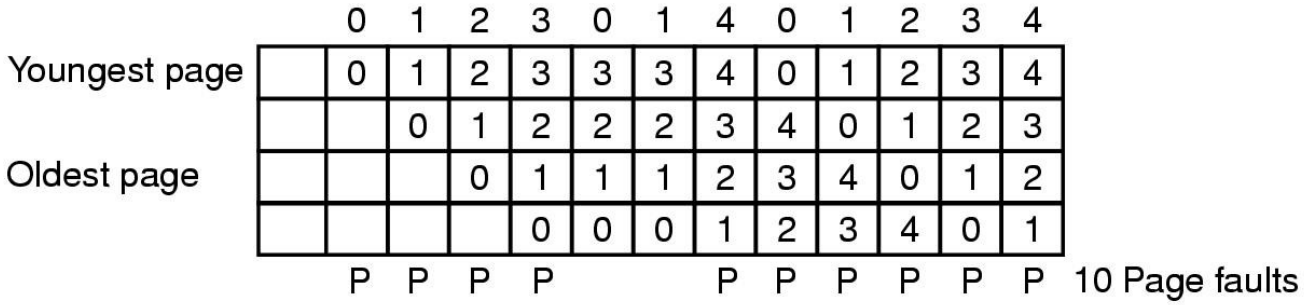
- Spesso una pagina viene utilizzata ripetutamente per un certo numero di istruzioni, mentre le pagine utilizzate poco non saranno necessarie per lunghi periodi di tempo.
  - L'idea dell'algoritmo LRU è di togliere la pagina che è rimasta inutilizzata per più tempo

LRU: Esempi

Riferimenti: 0 1 2 3 0 1 4 0 1 2 3 4



(a)



(b)

da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

LRU: implementazione

- Con una lista concatenata
  - In testa le pagine più recenti, in fondo quelle meno recenti
  - Problema: bisogna aggiornare la lista ad ogni accesso/riferimento alla memoria!

- Con una timestamp per ogni pagina
  - La timestamp viene aggiornata ad ogni riferimento
  - Scegliamo la pagina con la timestamp più bassa

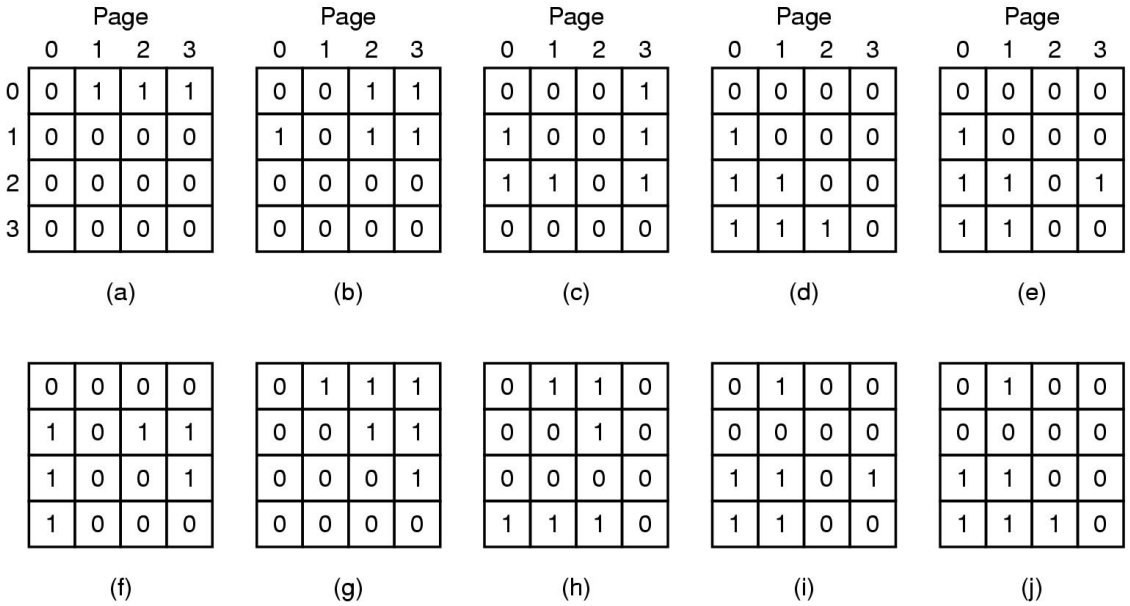
LRU in hardware (contatore)

- Supponiamo di avere un contatore a 64 bit (implementato nell'hardware) che venga incrementato dopo ogni istruzione
- Ogni elemento della tabella delle pagine contiene un campo sufficientemente grande per contenere il valore del contatore
- Dopo ogni riferimento alla memoria, il valore del contatore è salvato nel campo della pagina
- Quando ho un page fault la pagina con il valore del contatore più basso viene rimossa

LRU in hardware (matrice)

- Utilizzamo una matrice quadrata di n x n bit, inizializzati a 0, dove n è il numero di frame disponibili
- Quando faccio un riferimento al frame k, tutti i bit della riga k sono messi a 1, e tutti i bit della colonna k a 0
- Il frame con il valore binario più basso è quello meno usato recentemente

Riferimenti:
0 1 2 3 2 1 0 3 2 3



da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

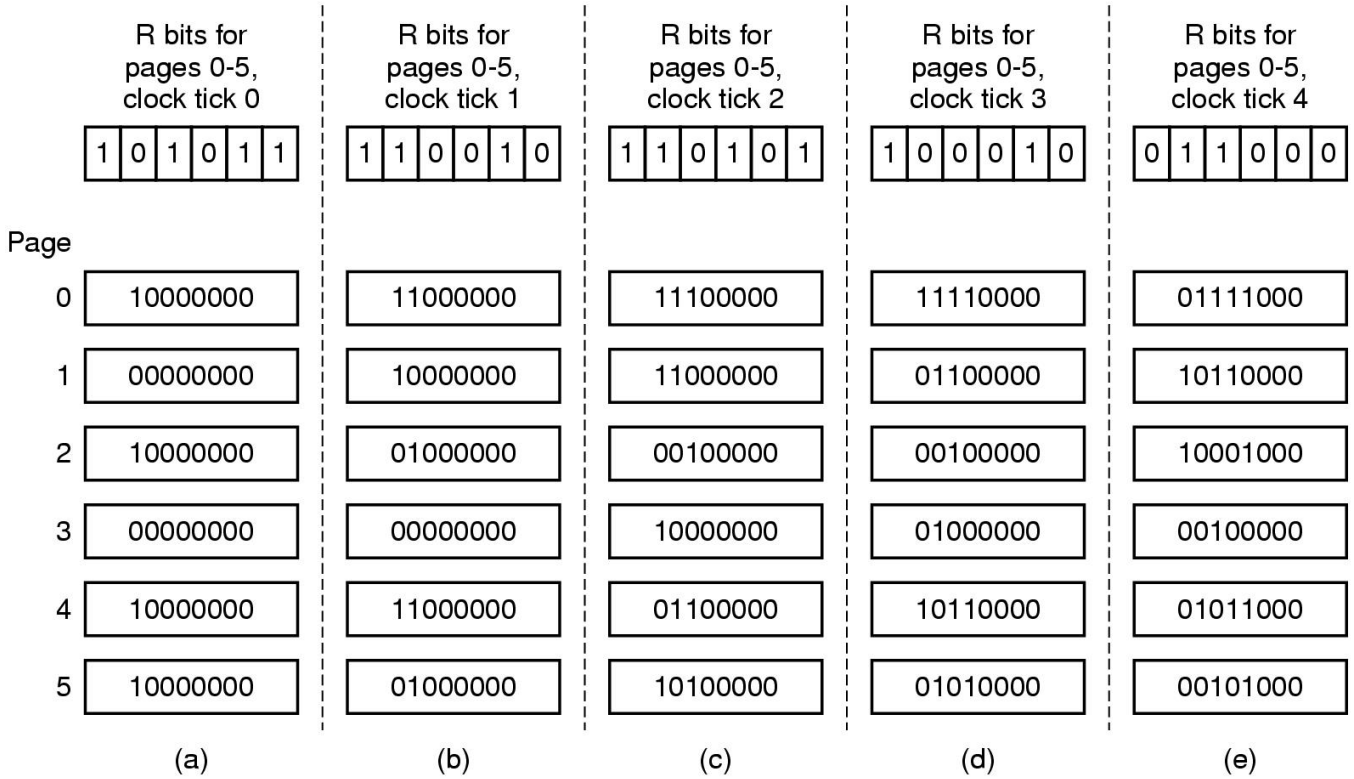
LRU in software: NFU (Not Frequently Used)

- Una prima implementazione di LRU senza il supporto hardware è chiamata NFU (Not Frequently Used)
- Ad ogni pagina in memoria associo un contatore, inizialmente uguale a 0
- Periodicamente sommo il valore del bit R a questo contatore (quindi + 0 o + 1)
- Quando devo sostituire una pagina, scelgo quella con il contatore di valore più basso

Problema: se una pagina riceve tanti riferimenti in un primo momento ma poi non viene più usata, il suo contatore rimarrà comunque superiore ad altri per diverso tempo

NFU (modificato)

- Prima di aggiungere il valore di R, viene eseguito uno shift a destra dei valori dei contatori
- Il bit R viene aggiunto a sinistra anziché a destra

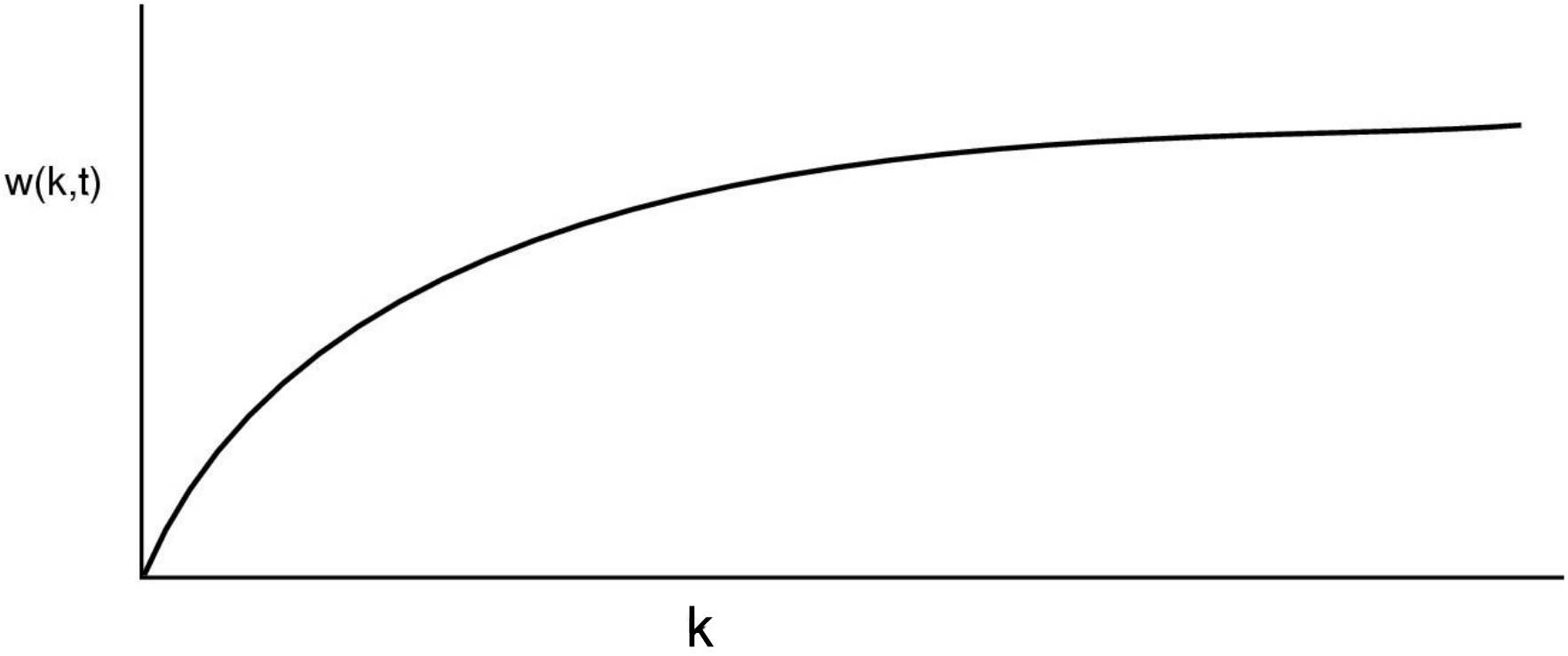


da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Demand paging vs Prepaging

- **Demand paging**
  - Le pagine vengono caricate quando vengono referenziate
- **Prepaging**
  - Le pagine vengono precaricate tenendo conto del working set di ogni processo, ovvero delle pagine che il processo utilizza in quel momento
    - Tiene conto del **principio di località**: i processi tipicamente fanno riferimento solo a una piccola parte delle pagine per volta

Working set



Working set: insieme delle pagine usate dai k più recenti accessi alla memoria; **w(k,t)** è la dimensione del working set al tempo k

da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Algoritmo del Working Set

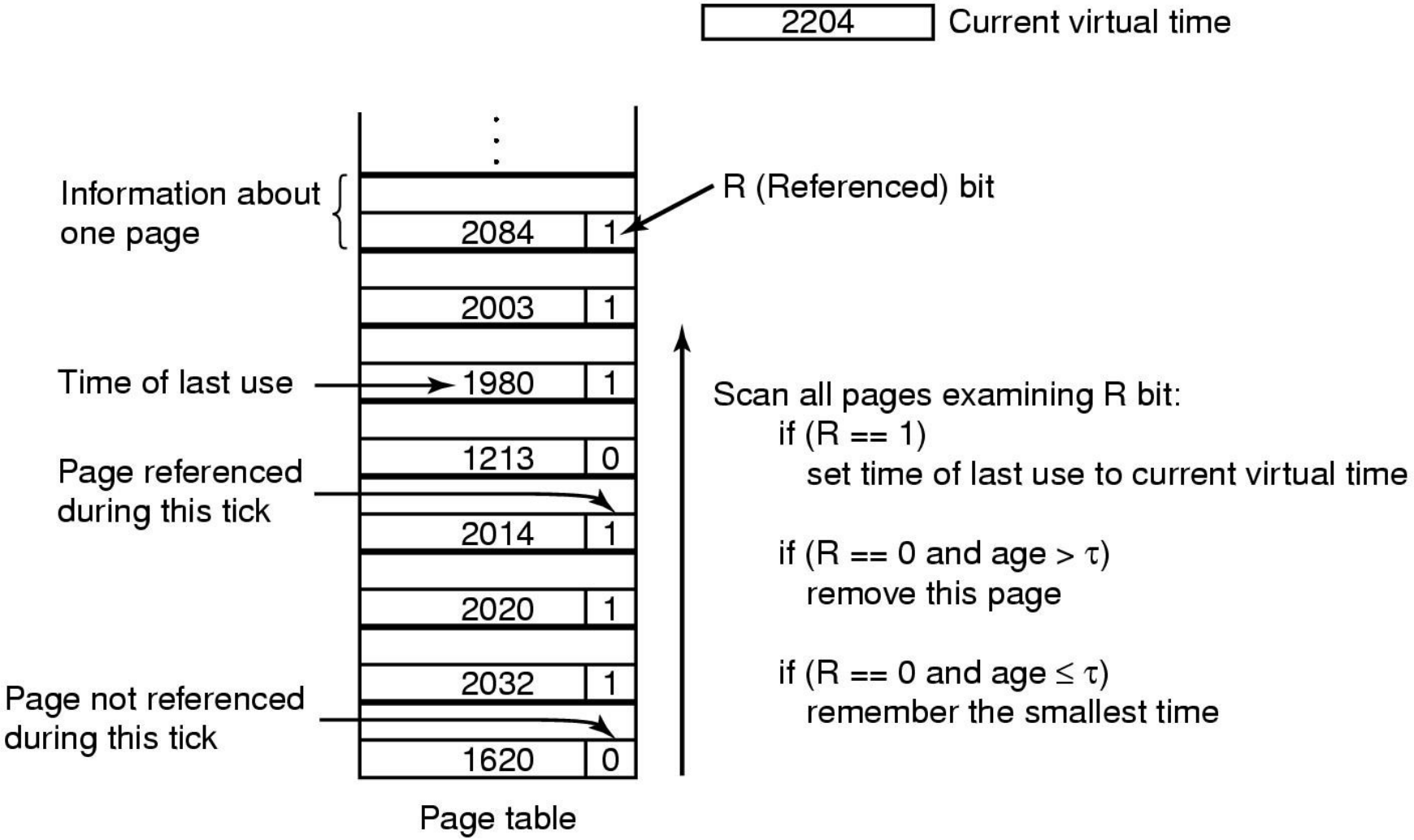
- **Idea:** cercare di identificare il working set del processo (pagine correntemente utilizzate o che saranno utilizzate a breve) in modo da caricarle in memoria prima che il processo ne abbia bisogno

- **Problema:** quanto è grande il working set (i.e. qual'è il valore k) ?
- **Problema:** come identifico le pagine nel working set?

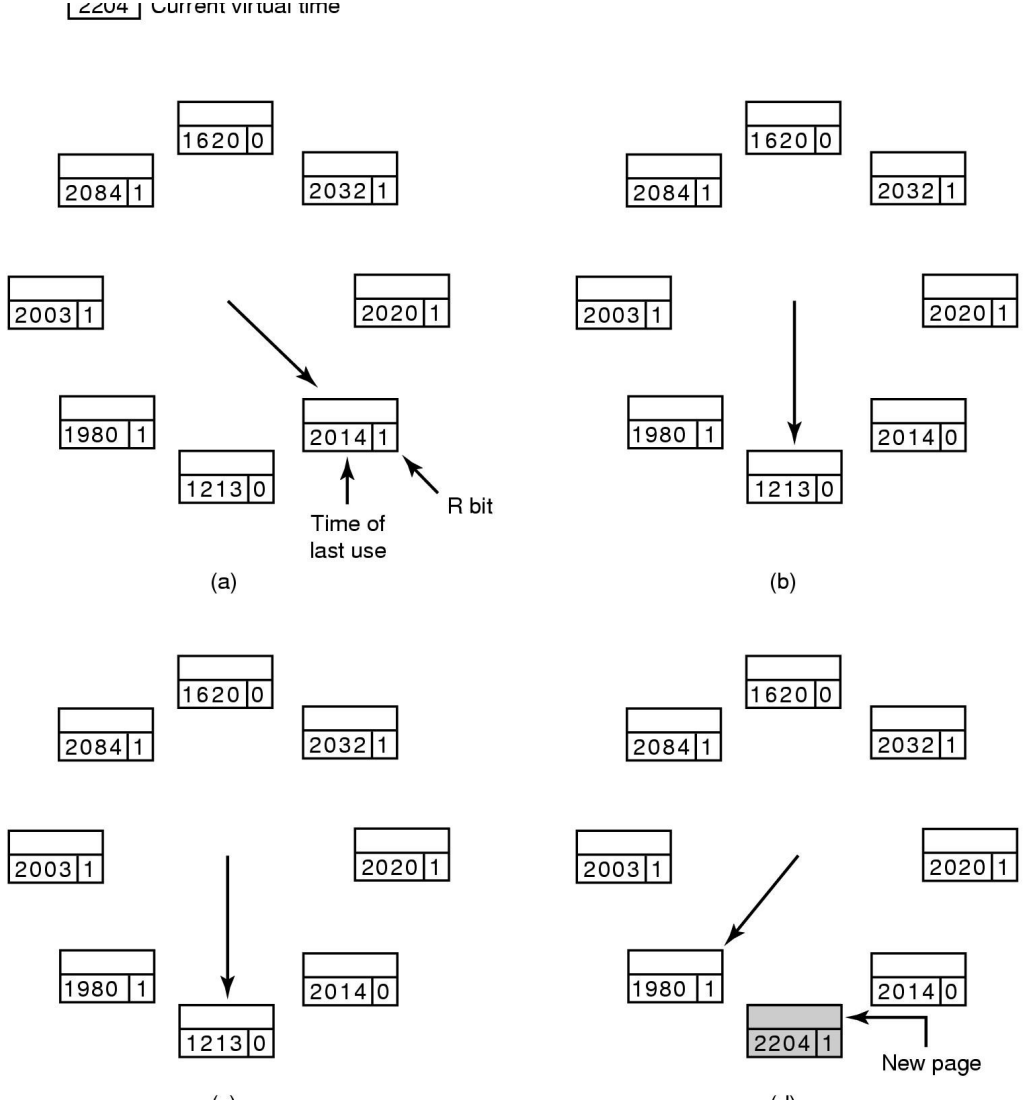
Implementazione del Working Set

- Mantengo un tempo virtuale di esecuzione (timestamp che indica per quanto tempo il processo è stato in esecuzione)
    - Nella tabella delle pagine, ad ogni pagina riservo un campo per una timestamp
- Quando devo togliere una pagina, scorro la tabella:
  - se **R=1**, la timestamp della pagina diventa il tempo virtuale di esecuzione corrente (e metto R=0)
  - se **R=0**, la pagina diventa candidata per essere tolta dal working set: Se le pagine hanno una timestamp più vecchia di una soglia τ possono essere tolte
- Se tutte le pagina avevano R=1, ne tolgo una a caso

Implementazione del Working Set

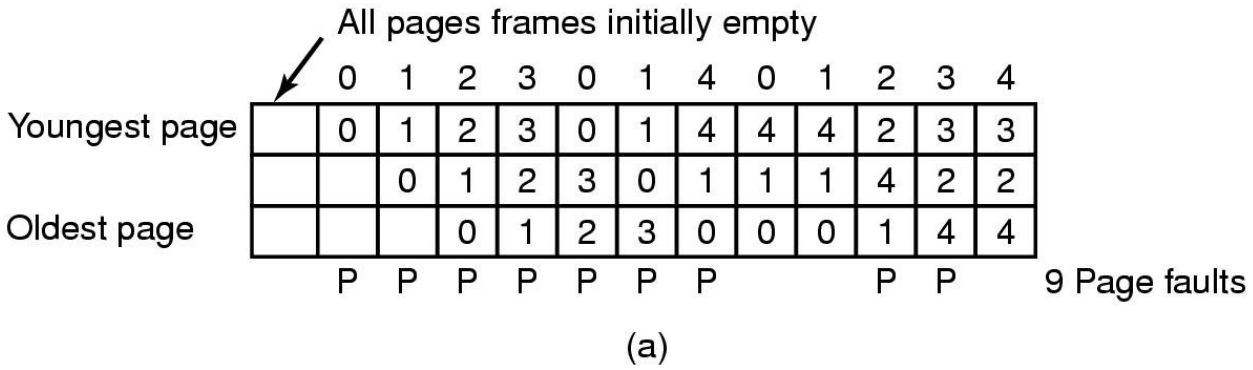


Algoritmo WSClock

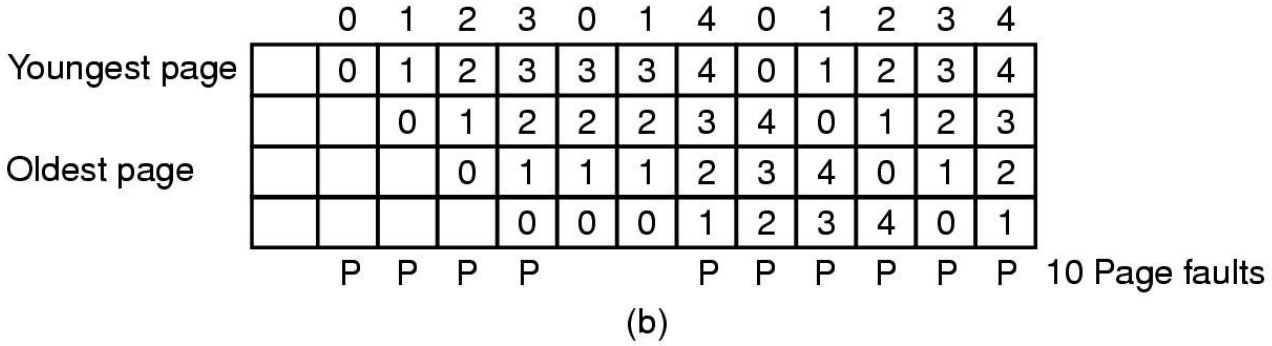


Anomalia di Belady

- **Un numero più grande di frame non sempre porta a un numero inferiore di page fault (questo succede negli algoritmi a stack)**



(a)



(b)

da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

Condivisione delle pagine

- Le pagine possono essere condivise tra più processi, semplicemente "mappandole" nei rispettivi spazi di indirizzamento
- Le pagine condivise contengono:
  - Codice (istruzioni)
    - es. librerie condivise (DLL, so), codice dello stesso programma
  - Dati

- Le pagine condivise sono tipicamente in sola lettura
  - Se la condivisione non è esplicita e uno dei processi modifica le pagine viene utilizzata la tecnica del **copy on write** (duplicazione delle pagine solo se necessario).

File mappati in memoria (memory mapped files)

- I processi possono (attraverso una chiamata di sistema, mmap) mappare un file in una parte del loro spazio di indirizzamento
  - Il file è mappato su una o più pagine in memoria
  - Questo meccanismo permette di lavorare su file come se fossero un array in memoria
  - Se due o più processi mappano lo stesso file allo stesso momento possono utilizzare questo meccanismo per comunicare tramite memoria condivisa

Partizione swap e file di paging

- Windows utilizza **un file su disco** (pagefile.sys) detto file di paging
- Linux tipicamente utilizza **una partizione speciale** del disco come spazio per lo swap, ma è possibile utilizzare anche un file
  - **`mkswap`** (comando per creare un'area di swap su disco o su una partizione)
  - **`swapon, swapoff`** (comandi per attivare/disattivare un'area di swap)

Un'altra problematica...

- I processi utilizzano più intervalli di memoria, ognuno in modo diverso
  - codice macchina
  - variabili globali
  - stack
  - ...
- Questi intervalli hanno bisogno di privilegi diversi
  - non tutti gli intervalli contengono codice eseguibile
  - non tutti gli intervalli devono essere modificabili durante l'esecuzione

Segmentazione

- Il meccanismo della segmentazione permette di dividere le diverse parti logiche di un processo (dati, codice,...) in spazi di indirizzamento separati

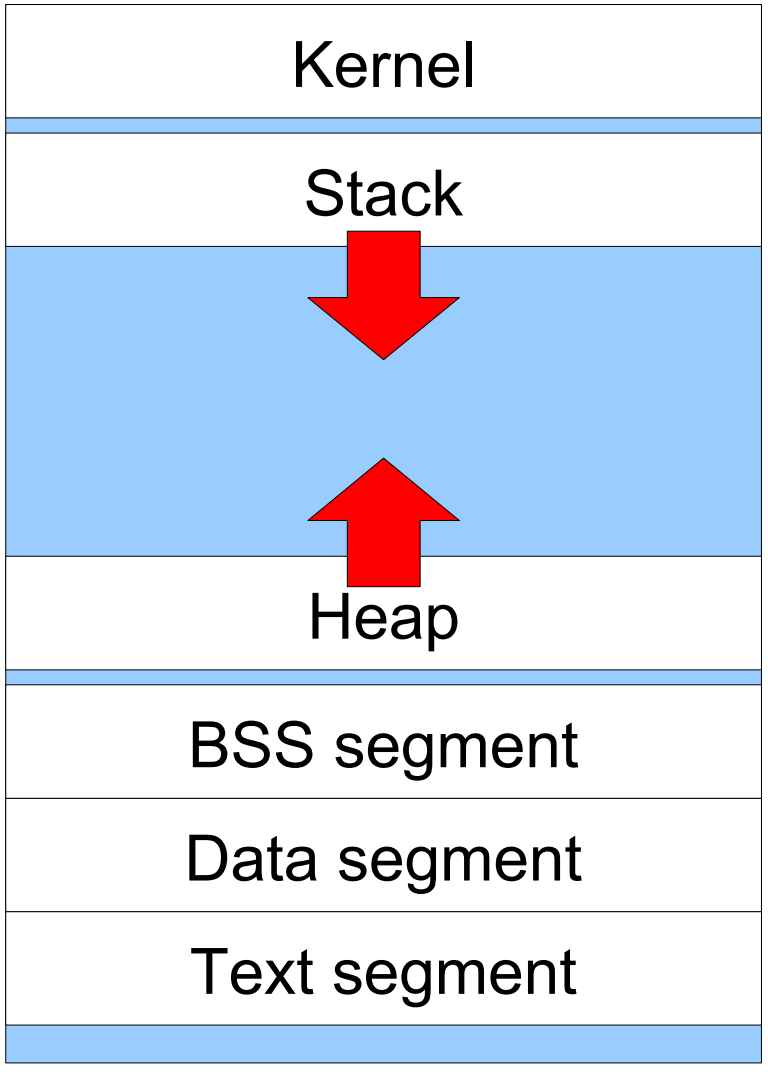
- Ogni "sezione" di memoria è chiamata **segmento**
  - è definito da un **indirizzo base** e una **dimensione** (o lunghezza) che determinano un nuovo spazio di indirizzamento
  - può avere privilegi specifici (r, w, x,...) → **protezione**
  - può essere utilizzato da più processi → **condivisione**

Segmentazione

- la "traduzione" degli indirizzi è affidata alla MMU
- a differenza del paging è utilizzabile direttamente dal programmatore (o dal compilatore) per **dividere la memoria in unità logiche differenti**

Mappa di un processo in memoria

- Codice
- Dati
  - Data segment (variabili inizializzate)
  - BSS * segment (variabili non inizializzate)
  - Heap (variabili allocate dinamicamente)
- Stack



* Block Started by Symbol

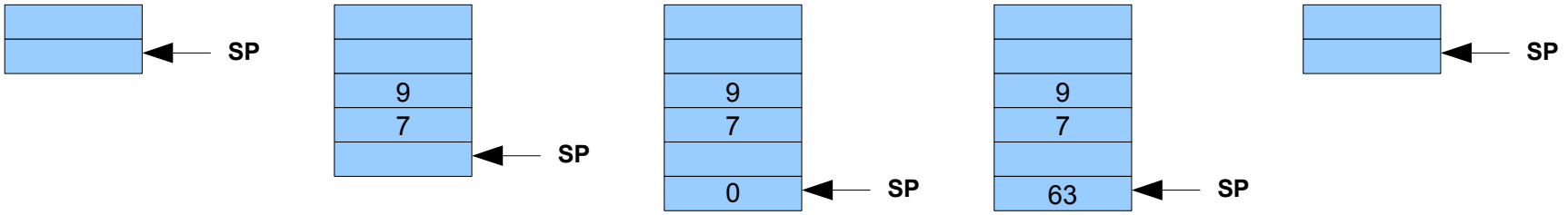
Piccola parentesi: lo stack

- Lo **stack** (pila) è una struttura dati con semantica LIFO (Last-In, First-Out)
  - L'ultimo elemento a venir aggiunto è il primo a venir rimosso
- È utilizzato durante l'esecuzione dei processi per mantenere le variabili locali e per il passaggio di parametri
- Un registro del processore (SP, stack pointer) contiene l'indirizzo della cima dello stack

Piccola parentesi: lo stack

```
int moltiplica(int a, int b) {
        int risultato;
        risultato = a * b;
        return risultato;
}

void main(void) {
        int k;
        k = moltiplica(7,9);
}
```

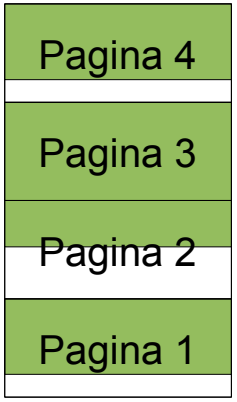


Lo stack “cresce” verso il basso

Frammentazione

- I segmenti hanno dimensione variabile
  - possono portare a **frammentazione esterna**

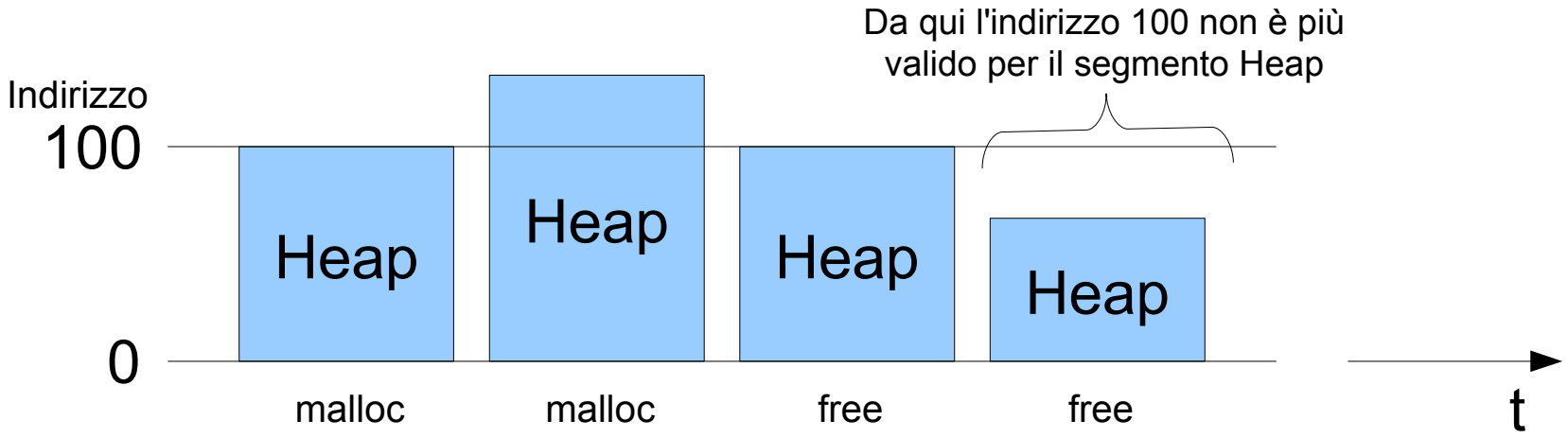
- Le pagine hanno dimensione fissa
  - possono portare a **frammentazione interna** quando non tutto lo spazio di una pagina viene utilizzato



Bisogna tenerne conto quando si implementa un sistema di paginazione!

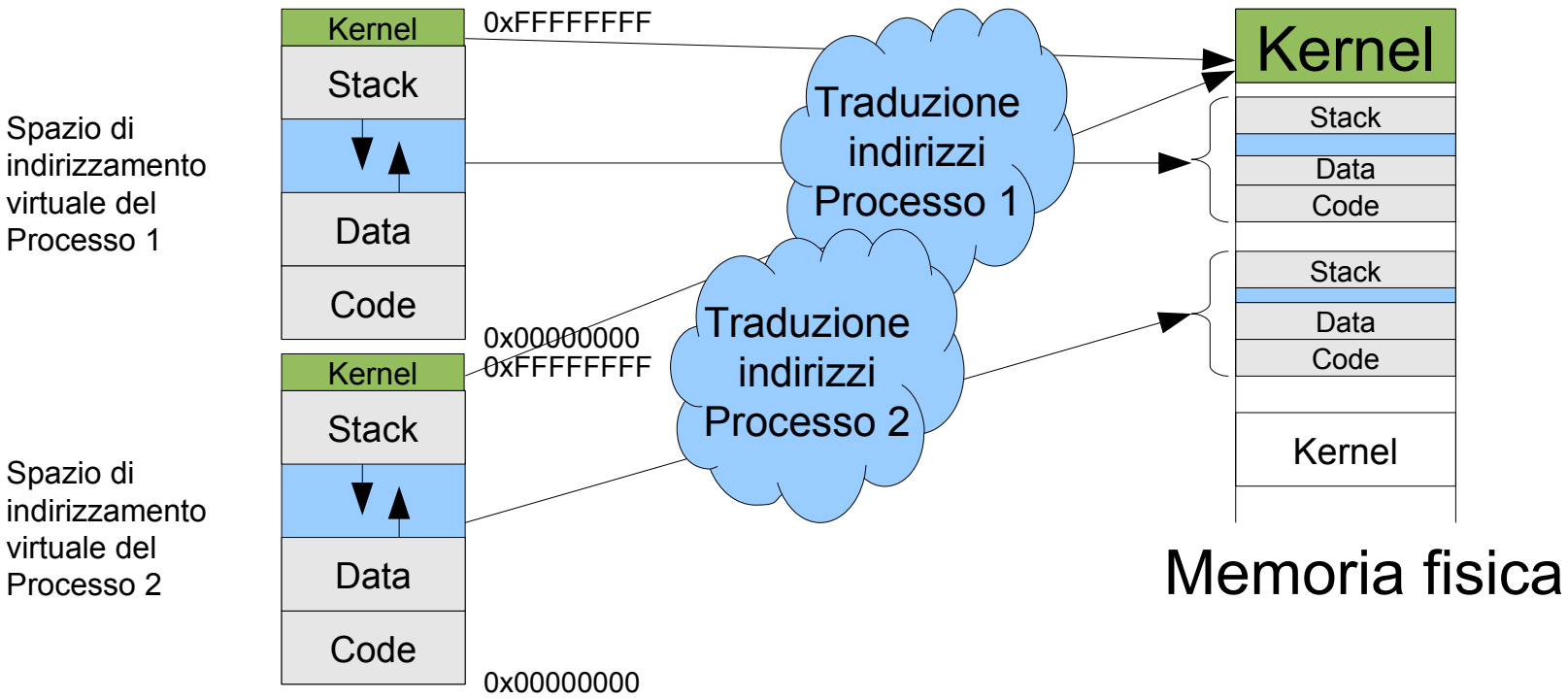
Segmentation fault

- Quando la memoria viene allocata dinamicamente (es. malloc/free in C) il segmento Heap viene ridimensionato
- Quando un processo cerca di accedere a un indirizzo al di fuori di un segmento valido il sistema genera un errore di segmentazione, il **segmentation fault**



Segmentazione e paginazione

- I segmenti possono essere divisi su più pagine



Confronto tra segmentazione e paginazione

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No*	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No*	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

da A. Tanenbaum, Modern Operating Systems, 2nd Ed.

* È comunque possibile allocare pagine con privilegi diversi e condividere pagine tra processi

Confronto tra segmentazione e paginazione

- La segmentazione non è utilizzata nelle architetture moderne
  - Nell'architettura amd64 non è possibile utilizzare la segmentazione in modalità 64 bit (long mode)
- La paginazione permette di gestire protezione e condivisione a livello di pagina, la segmentazione a livello di aree di memoria di dimensione variabile
- La protezione dei segmenti di memoria è più fine rispetto alle pagine:
  - Protection ring vs User/Supervisor Bit
  - Per ovviare a questo problema sono state aggiunte delle estensioni nei sistemi di paginazione (es. NX bit)
- La paginazione presenta un modello di memoria più semplice (flat memory model)

Memoria fisica occupata, memoria virtuale: **ps aux**

[X] bash

```
utente@host:~/Documenti$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
utente    2734  0.0  0.0  50204  3624 ?        Sl   Dec05   0:00 /usr/bin/gnome-keyring-daemon --daemonize
--login
utente    2743  0.0  0.1  29792  6096 ?        Ssl  Dec05   0:00 gnome-session
utente    2752  0.0  0.0   3588   632 ?        S    Dec05   0:00 dbus-launch --sh-syntax --exit-with-session
utente    2753  0.0  0.0  14484  1752 ?        Ssl  Dec05   0:04 /bin/dbus-daemon --fork --print-pid 5
--print-address 7 --session
utente    2842  0.0  0.1   8768  4360 ?        S    Dec05   0:32 /usr/libexec/gconfd-2
utente    2850  0.0  0.3 149372  3884 ?        Ssl  Dec05   0:24 /usr/libexec/gnome-settings-daemon
utente    2852  0.0  0.1  27364  7904 ?        Ss   Dec05   0:00 seahorse-daemon
utente    2859  0.0  0.0   7544  2356 ?        S    Dec05   0:00 /usr/libexec/gvfsd
utente    2866  0.0  0.0  39520  2436 ?        Ssl  Dec05   0:00 /usr/libexec//gvfs-fuse-daemon
/home/utente/.gvfs
utente    2871  0.0  0.4  54632  6716 ?        S    Dec05   0:14 gnome-panel
utente    2875  0.2  0.1 113152  4936 ?        S<sl Dec05   3:22 /usr/bin/pulseaudio --start --log-
target=syslog
utente    2881  0.0  0.0  11156  2732 ?        S    Dec05   0:00 /usr/libexec/pulse/gconf-helper
utente    2891  0.0  4.4 793812 80788 ?        S    Dec05   0:48 nautilus
utente    2893  0.0  0.0  46872  3420 ?        Ssl  Dec05   0:00 /usr/libexec/bonobo-activation-server --ac-
activate --ior-output-fd=18
```

Resident Set Size memoria fisica utilizzata (non swappata su disco)

Memoria virtuale (kilobytes)

% della memoria fisica occupata (RSS / RAM)

Mappa della memoria, segmenti : **pmap -x PID**

[X] bash

```
utente@host:~/Documenti$ pmap -x 13242
13242:   gnome-terminal
Address   Kbytes     RSS   Dirty Mode  Mapping
00101000     332     196       0 r-x-- libORBit-2.so.0.1.0
(...)
00177000       8       8       8 rw--- libICE.so.6.3.0
00179000       4       0       0 rw---   [ anon ]
0017a000      48      16       0 r-x-- libnss_files-2.12.so
(...)
003d3000     160      56       0 r-x-- libm-2.12.so
003fb000       4       4       0 r---- libm-2.12.so
003fc000       4       4       4 rw--- libm-2.12.so
b609b000     608     124       0 r---- DejaVuSans.ttf
b6133000     296      44       0 r---- DejaVuSansMono-Bold.ttf
b617d000     316      64       0 r---- DejaVuSansMono.ttf
b61cf000      24      24       0 r--s- b79f3aaa7d385a141ab53ec885cc22a8-le32d4.cache-3
b61d5000       8       8       0 r--s- 0b1bcc92b4d25cc154d77dafe3bceaa0-le32d4.cache-3
(...)
bfb70000     108      96      96 rw---   [ stack ]
--------- ------- ------- ------- -----
total kB    5804       -       -       -
```

Permessi (read, write, execute, shared,...)

Pagine dirty (kilobytes)

Resident Set Size memoria fisica utilizzata

Dimensione (kilobytes)

Indirizzo base (virtuale)

Memoria utilizzata, libera e swapping : **free** e **vmstat**

[X] bash

```
utente@host:~/Documenti$ free
             total       used       free     shared    buffers     cached
Mem:       4030168    2696576    1333592          0     273684    1351132
-/+ buffers/cache:    1071760    2958408
Swap:      6127608          0    6127608
```

Page Cache per operazioni su file

Memoria fisica e swap (kilobytes)

Buffer per operazioni I/O

[X] bash

```
utente@host:~/Documenti$ vmstat
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 1332684 273552 1349896    0    0     3     8   10   84  3  1 96  0  0
```

swpd: swap usato
free: libera
buff: buffers
cache: page cache

si: swap-in / secondo
so: swap out / secondo