Gestione della memoria nei SO Linux e Windows

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Gestione della memoria nei SO Linux e Windows

- Sommario
  - Gestione della memoria in Linux
    - Gestione user-space e kernel-space
    - Algoritmi di allocazione (buddy system, slab allocator)
    - Page cache
    - Page Frame Reclaiming
  - Gestione della memoria in Windows
    - Virtual address map
    - Paginazione e working set
- Riferimenti
  - W. Stallings, "Operating Systems", par. 7.2 (Buddy System), 8.4 e 8.5
  - dispensa didattica su memory management in Linux e Windows
  - www.ostep.org, Cap. 23

- Gestione della memoria in Linux

top

Memoria RAM e swap del sistema

so@so-vbox: ~/linux_mem

```
top - 23:20:47 up 41 min,  1 user,  load average: 0,10, 0,14, 0,16
Tasks: 282 total,   1 running, 281 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4,5 us,  1,7 sy,  0,0 ni, 93,7 id,  0,0 wa,  0,0 hi,  0,2 si,  0,0 st
MiB Mem :   1987,5 total,   1051,8 free,    500,6 used,    435,2 buff/cache
MiB Swap:   1873,4 total,   1246,8 free,    626,6 used.   1332,8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1317 so        20   0 4162900 140696  71772 S   6,0   6,9   0:47.74 gnome-+
   1024 so        20   0  530324  28176  11764 S   2,6   1,4   0:18.92 Xorg
   2259 so        20   0  820744  31900  21460 S   2,0   1,6   0:20.97 gnome-+
    670 root      20   0  168264   4000   2472 S   0,3   0,2   0:12.94 vmtool+
   1520 so        20   0  293608   5632   4464 S   0,3   0,3   0:05.77 vmtool+
   3988 root      20   0       0      0      0 I   0,3   0,0   0:00.44 kworke+
   4089 so        20   0   11984   3928   3172 R   0,3   0,2   0:00.05 top
      1 root      20   0  167852   2840   2308 S   0,0   0,1   0:04.01 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:00.01 kthrea+
      3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_pa+
      6 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworke+
      9 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 mm_per+
     10 root      20   0       0      0      0 S   0,0   0,0   0:00.46 ksofti+
     11 root      20   0       0      0      0 I   0,0   0,0   0:01.02 rcu_sc+
     12 root      rt   0       0      0      0 S   0,0   0,0   0:00.02 migrat+
```

Spazio di indirizzamento utilizzato dal processo

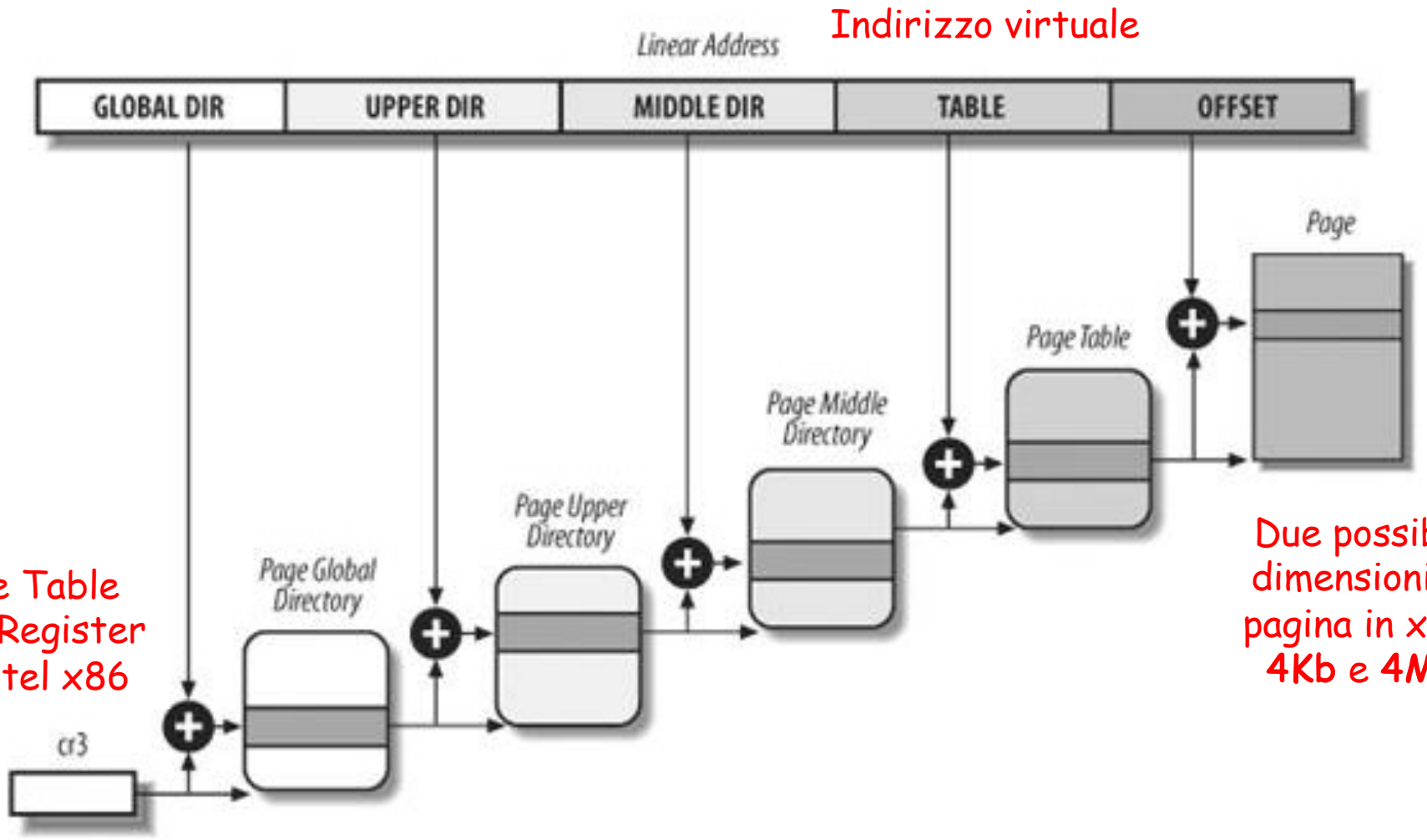
Memoria "residente" (spazio occupato in RAM)

Pagine condivise (es. codice librerie)

Caratteristiche generali

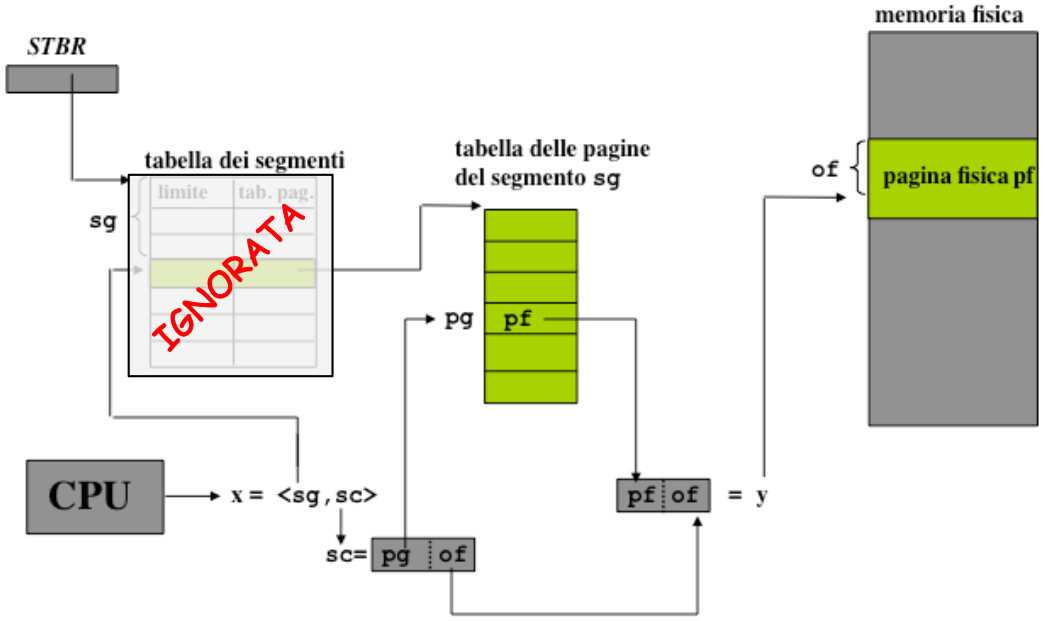
- Gestione **paginata** e **segmentata** della memoria
- **Portabilità** tra diverse architetture
- Supporto per sistemi con grandi quantità di memoria (NUMA) e multi-processore (SMP)

Livelli di paginazione



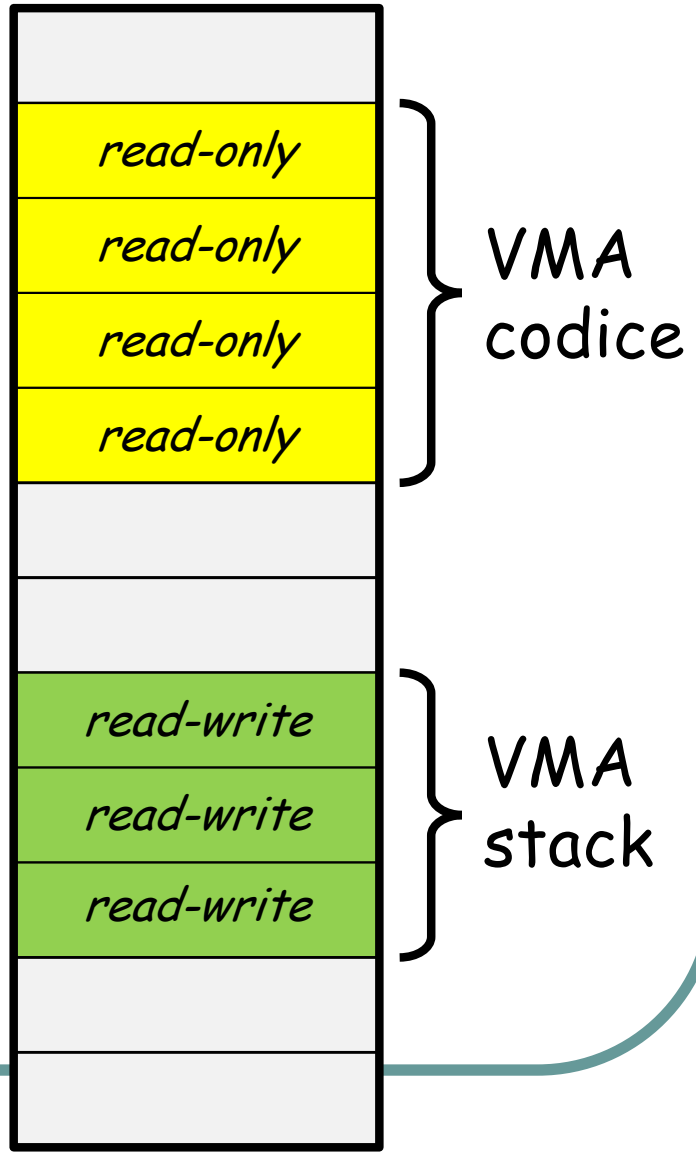
Spazio di indirizzamento virtuale

- Le versioni moderne di Linux **non si avvalgono del processore per la segmentazione**, per motivi di
  - portabilità
  - efficienza del context switch e traduzione degli indirizzi



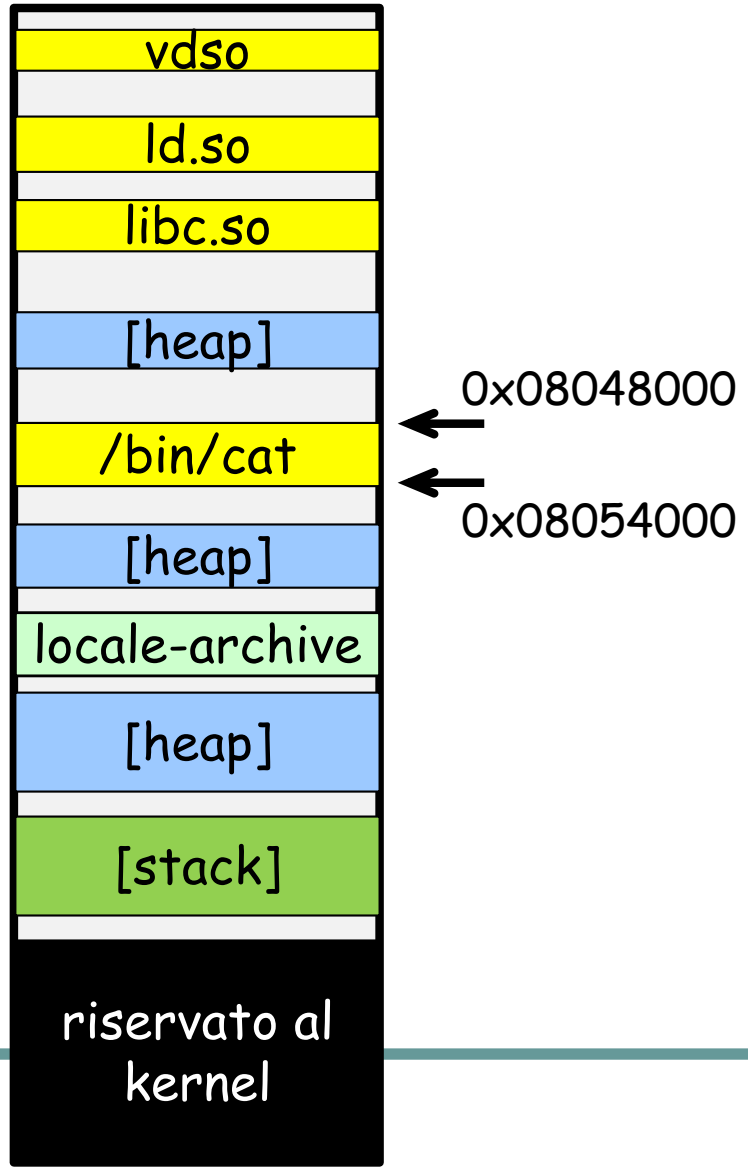
Spazio di indirizzamento virtuale

- La memoria è organizzata in **Virtual Memory Areas** (***VMA***)
- **Pagine virtuali contigue**, contenenti codice, stack, librerie, ...
- **Protezione** e **condivisione** sono ottenute tramite le VMA



Spazio di indirizzamento virtuale

Spazio virtuale di un processo in Linux



Esempio di spazio virtuale in Linux

```
$ cat /proc/<PID>/maps
```

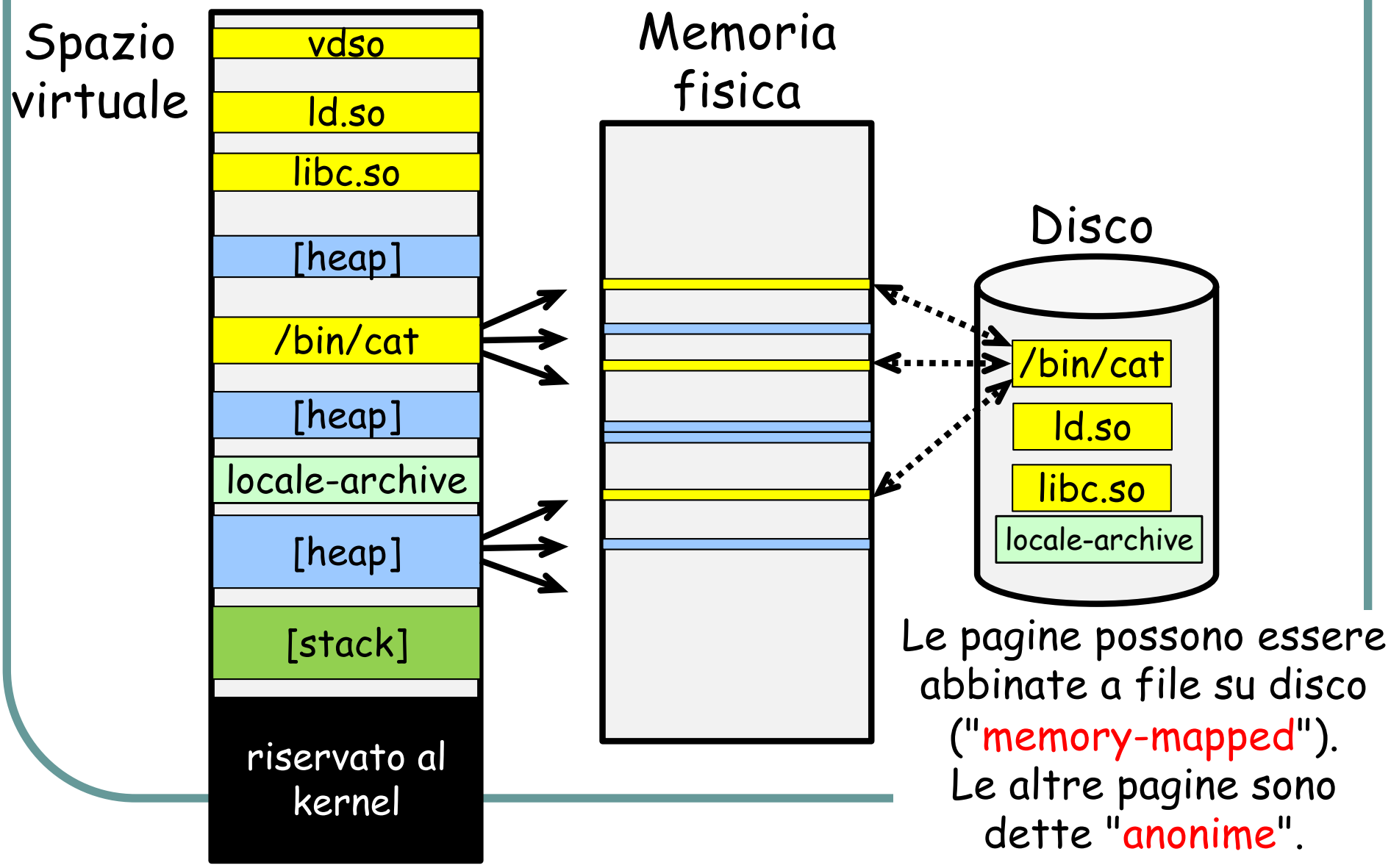
```
0027e000-0027f000 r-xp 00000000 00:00 0          [vdso]
0061a000-00638000 r-xp 00000000 08:02 4872       /lib/ld-2.12.1.so
00638000-00639000 r--p 0001d000 08:02 4872       /lib/ld-2.12.1.so
00639000-0063a000 rw-p 0001e000 08:02 4872       /lib/ld-2.12.1.so
0063c000-007c1000 r-xp 00000000 08:02 5433       /lib/libc-2.12.1.so
007c1000-007c2000 ---p 00185000 08:02 5433       /lib/libc-2.12.1.so
007c2000-007c4000 r--p 00185000 08:02 5433       /lib/libc-2.12.1.so
007c4000-007c5000 rw-p 00187000 08:02 5433       /lib/libc-2.12.1.so
007c5000-007c8000 rw-p 00000000 00:00 0
08048000-08053000 r-xp 00000000 08:02 12016      /bin/cat
08053000-08054000 rw-p 0000a000 08:02 12016      /bin/cat
09607000-09628000 rw-p 00000000 00:00 0          [heap]
b7672000-b7872000 r--p 00000000 08:02 8857       /usr/lib/locale/locale-archive
b7872000-b7873000 rw-p 00000000 00:00 0
b7884000-b7885000 rw-p 00000000 00:00 0
bfaaa000-bfabf000 rw-p 00000000 00:00 0          [stack]
```

Codice

Heap (memoria "anonima")

File memory-mapped

Spazio di indirizzamento virtuale

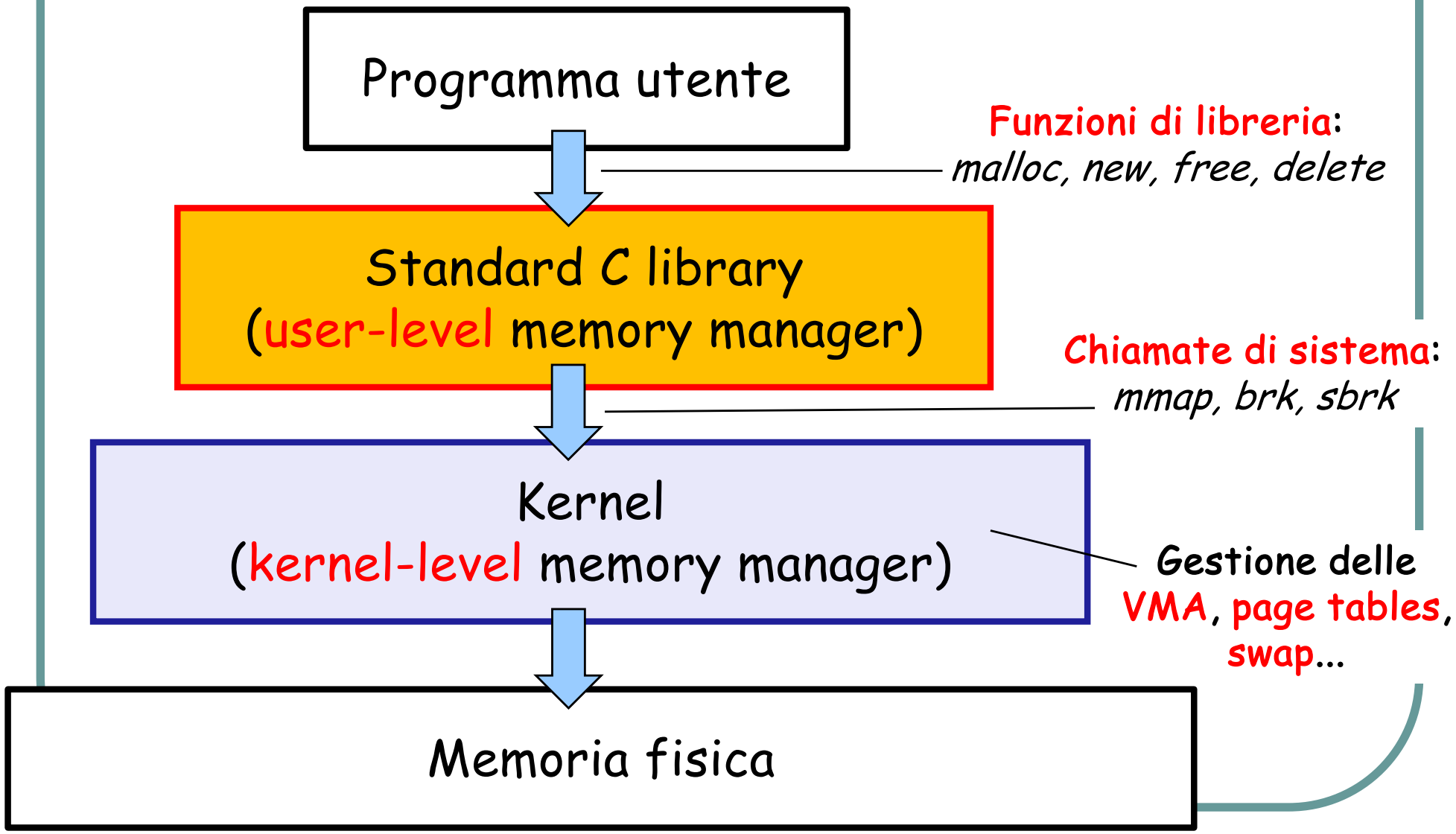


Le pagine possono essere abbinate a file su disco ("memory-mapped"). Le altre pagine sono dette "anonime".

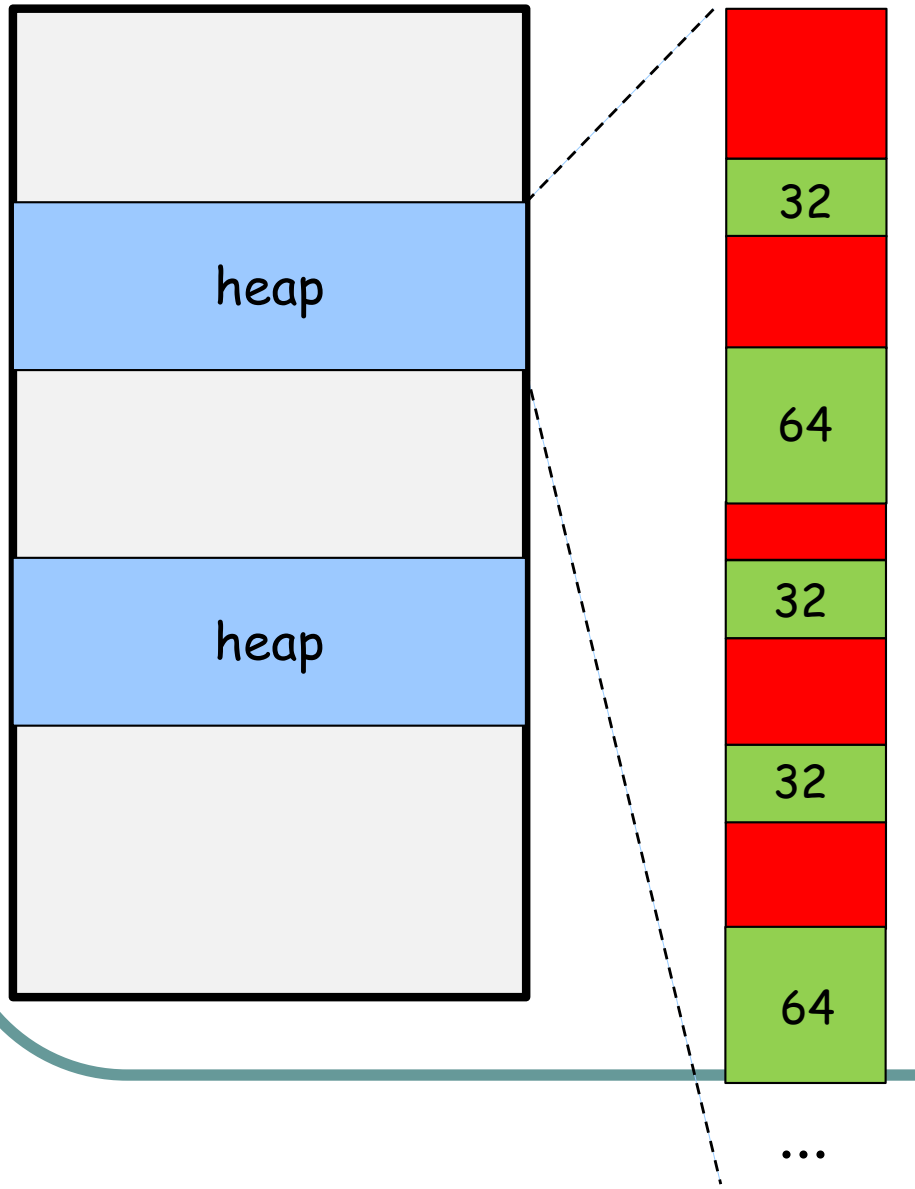
Allocazione di memoria in Linux

- **User-level allocator** (nella **libreria C**):
  - malloc()/free(), new/delete
  - seleziona un blocco libero nella area heap del processo
- **Kernel-level allocator**:
  - se l'area heap è esaurita, malloc() richiede al kernel nuove pagine
  - syscall *brk*, *sbrk* (espande una VMA), *mmap* (aggiunge una nuova VMA)

Allocazione di memoria in Linux



Allocatore user-space

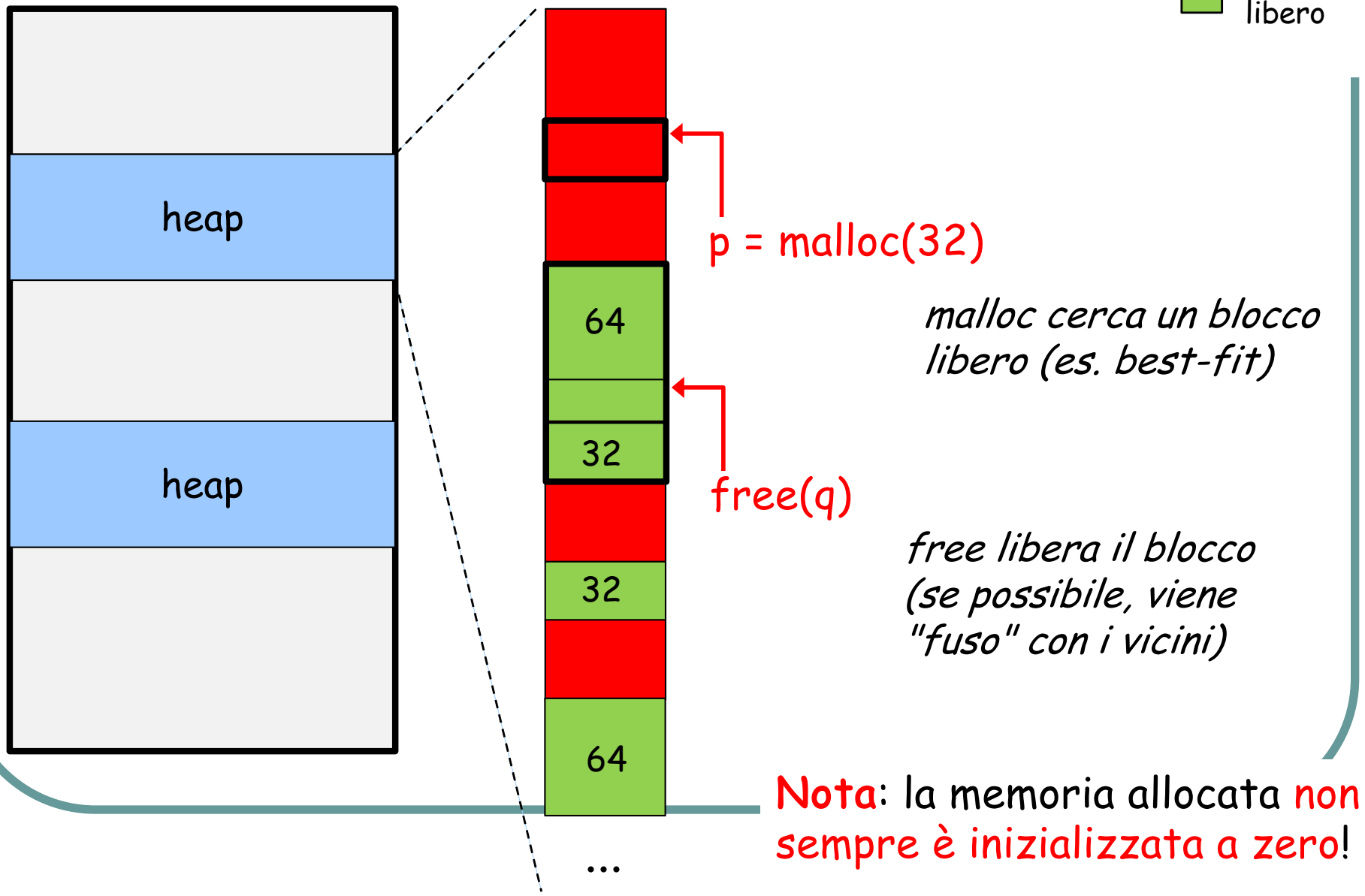


chunk allocato

chunk libero

- L'area heap alterna blocchi liberi e allocati
- I blocchi hanno dimensione variabile

Allocatore user-space

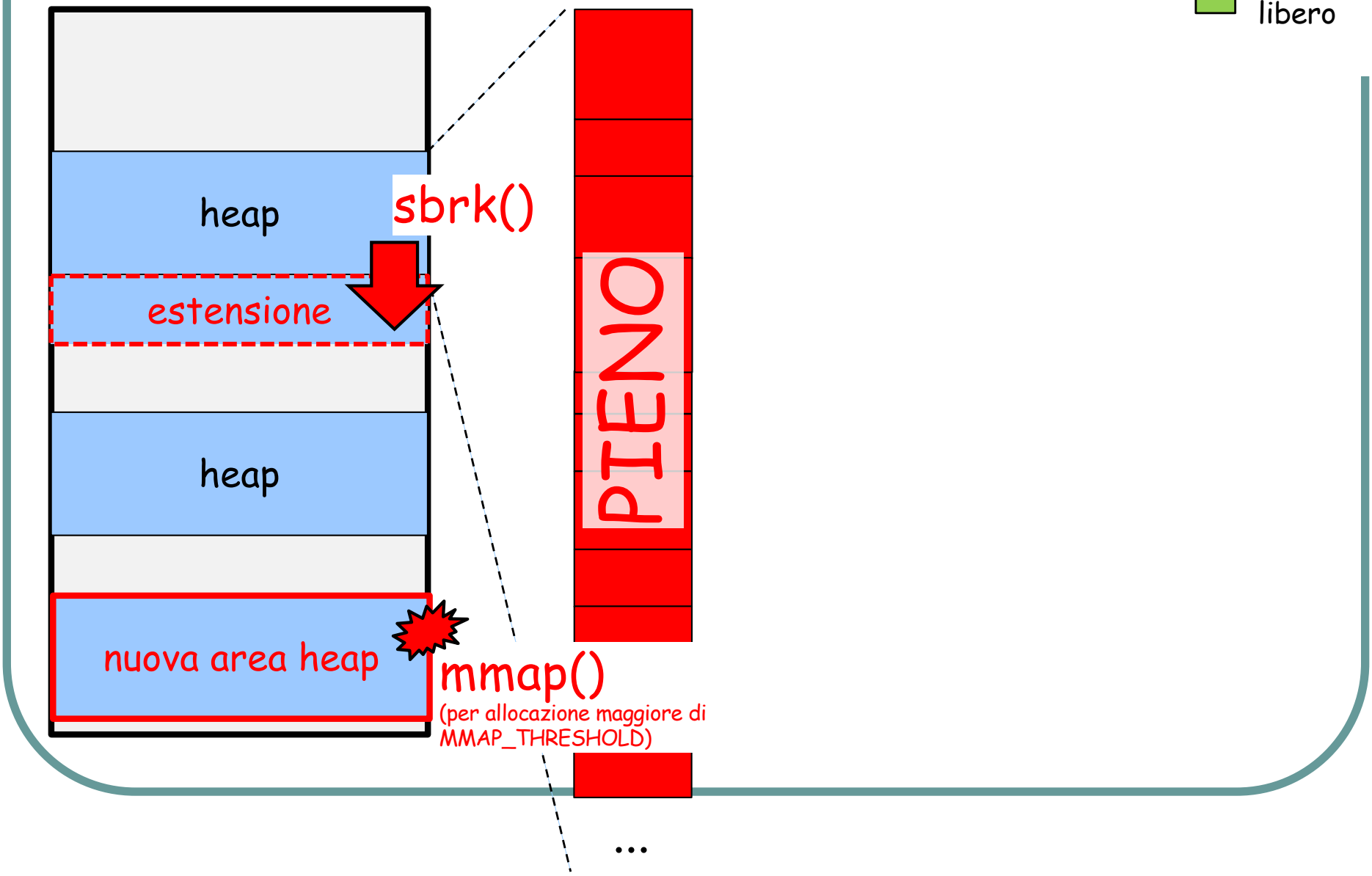


malloc cerca un blocco libero (es. best-fit)

free libera il blocco (se possibile, viene "fuso" con i vicini)

Nota: la memoria allocata non sempre è inizializzata a zero!

Allocatore user-space

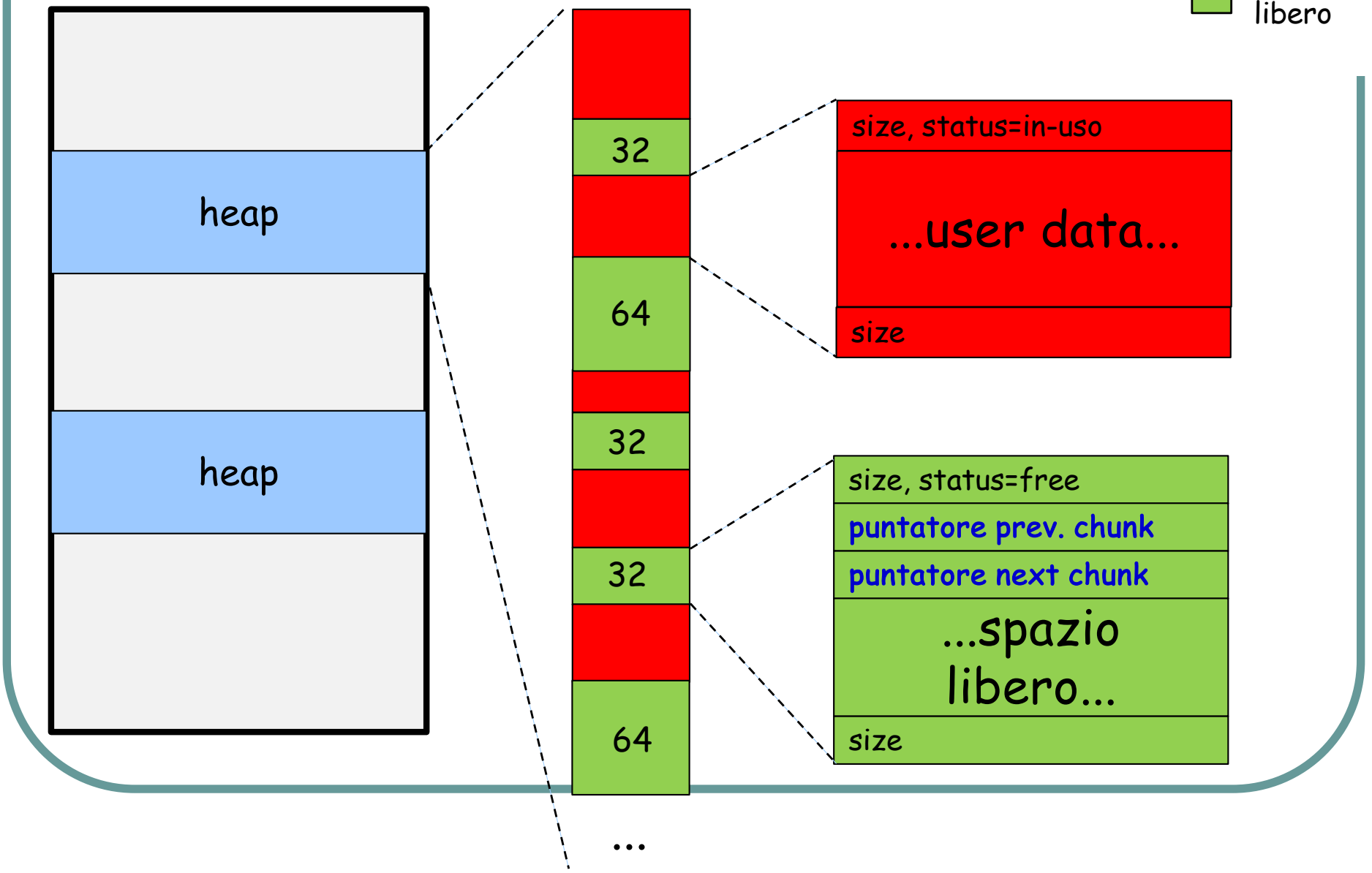


GNU C library

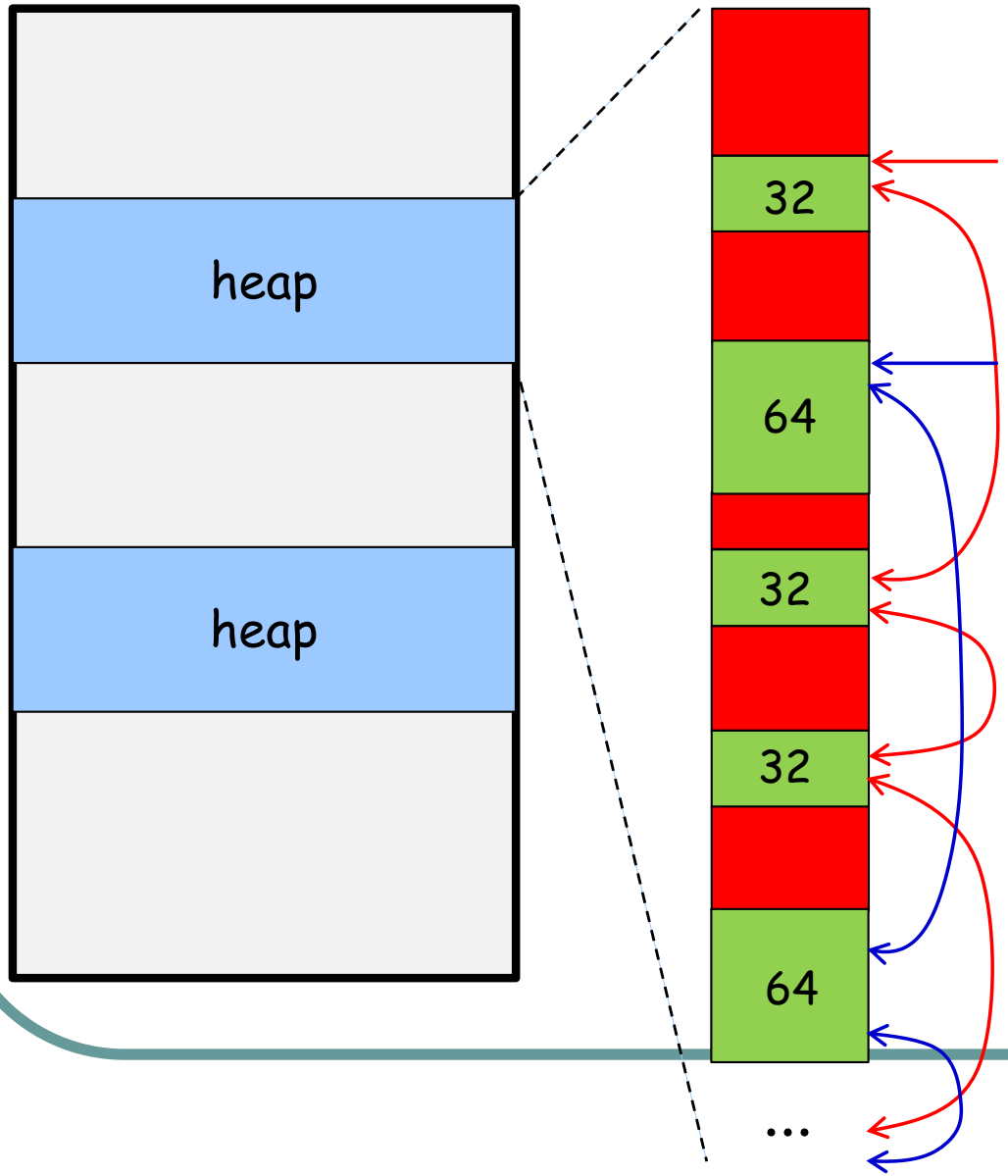
- La libreria C in Linux utilizza l'allocatore ***ptmalloc***, un derivato dell'allocatore tradizionale di Doug Lea (***dlmalloc***)

http://gee.cs.oswego.edu/dl/html/malloc.html

Allocatore user-space

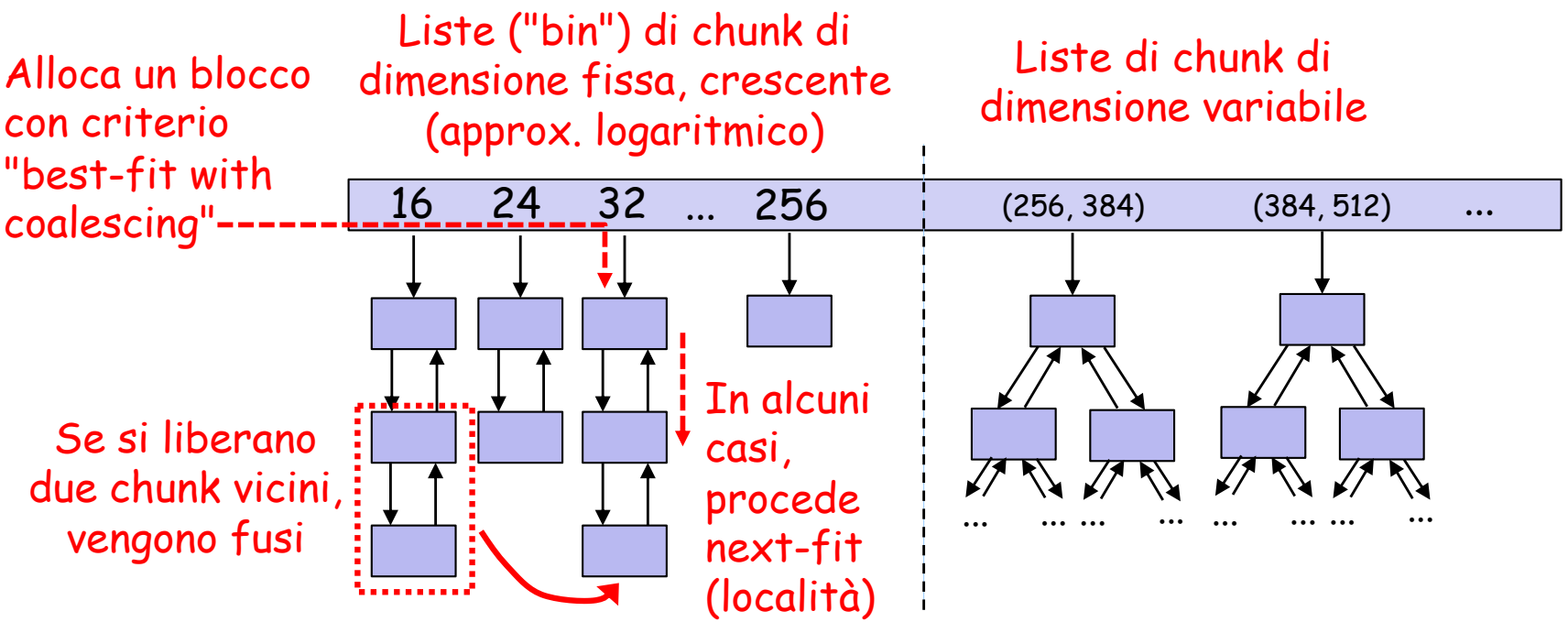


Allocatore user-space



- Tiene traccia dei blocchi di memoria libera ("chunk") tramite **linked list**
- I puntatori sono memorizzati all'interno dei blocchi liberi

GNU C library



http://gee.cs.oswego.edu/dl/html/malloc.html

Puntatori

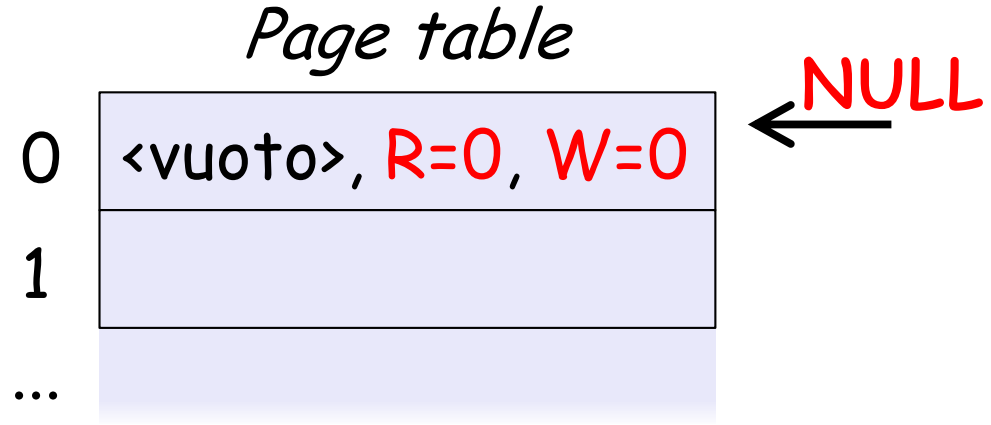
- La MMU verifica la validità dell'indirizzo tramite la tabella delle pagine
- Nel caso di puntatori non validi nel programma, il processo può essere "ucciso" dal SO

- la MMU genera un page fault
- la ISR del SO verifica se l'indirizzo sia incluso nell'intervallo degli indirizzi (VMA) di una delle aree heap/stack/...
- in caso negativo, il SO uccide il processo ("segmentation fault")

Puntatori NULL

```
$ cat pointer-null.c

int main() {

    char * address = NULL;

    printf("Il contenuto della memoria all'indirizzo %p è: %x\n",
                                                    address, *address);
}

$ ./pointer-null

Errore di segmentazione (core dump creato)
```

Puntatori NULL



- La prima pagina virtuale del processo (indirizzi 0-4095) ha sempre i permessi di accesso disattivati
- Qualunque accesso viene considerato illecito
- Utile ai fini del debugging

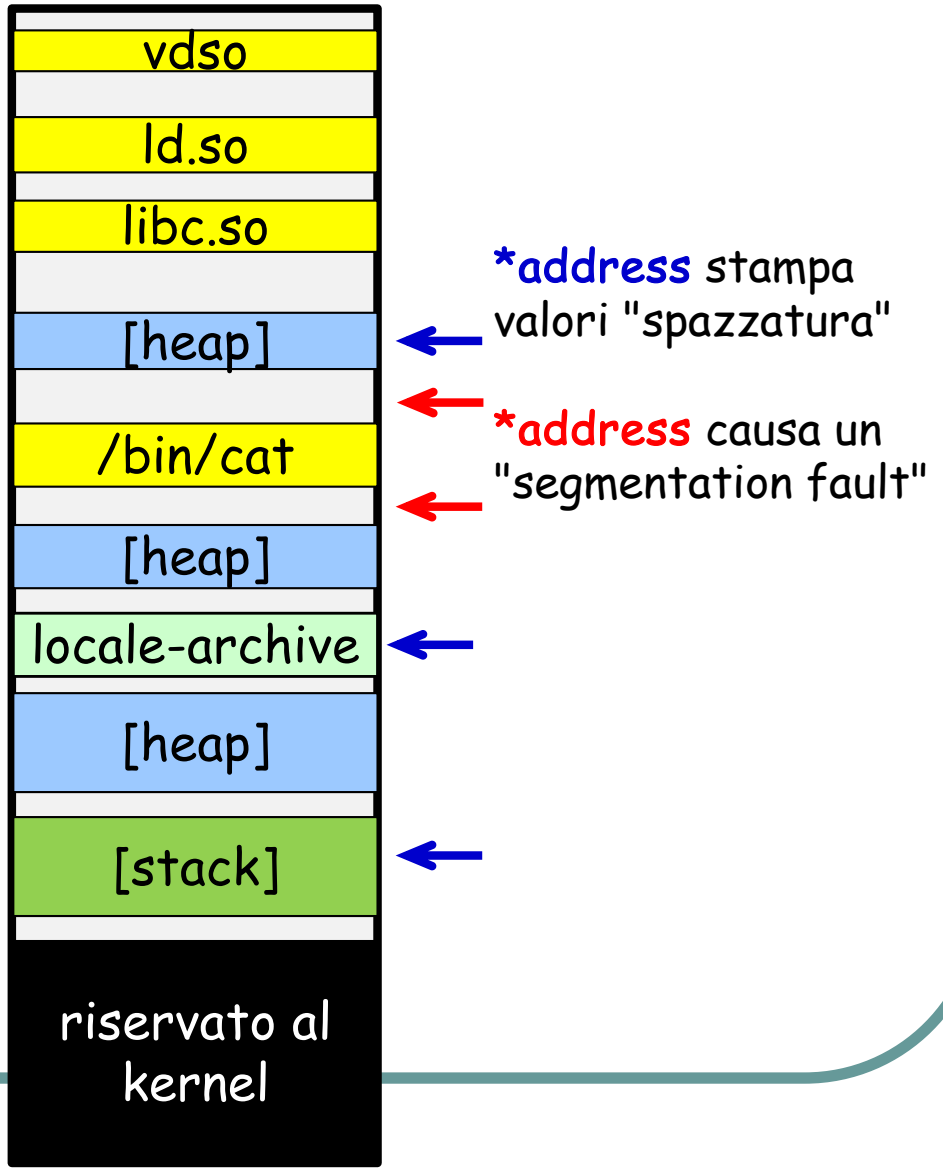
Puntatori non inizializzati

```
$ cat pointer-uninitialized.c

int main() {

    char * address;       // equivale a "address = rand()"

    printf("Il contenuto della memoria all'indirizzo %p è: %x\n",
                                                    address, *address);
}

$ ./pointer-uninitialized

???
```

Questo **comportamento è "undefined"** nello standard del linguaggio C.
Il risultato dipende dal SO, dal compilatore e dalla CPU.

Puntatori non inizializzati

- Due possibilità:
  - il puntatore (per puro caso) ha un indirizzo valido di una delle aree del processo
  - il puntatore contiene un indirizzo non valido (fuori dalle aree)



Spazio virtuale di un processo in Linux

Puntatori

NOTA: malloc() può ritornare il valore NULL se la allocazione fallisce, es.

1) Lo spazio di memoria virtuale è esaurito
2) Lo spazio in memoria fisica è esaurito

```
$ cat pointer.c

int main() {

    char * address = malloc(100);

    printf("Il contenuto della memoria all'indirizzo %p è: %x\n",
                                                address, *address);
}
```

Out-of-memory killer

```
$ cat loop-calloc.c

int main() {
    void* block=NULL;
    int count=0;
    while(1) {
          block = (void *) malloc(1024*1024);     // alloca 1 MB
          if (!(block)) break;
          memset(block,1,1024*1024);    // scrive degli "1" sulle pagine allocate
          printf("Currently allocating %d MB\n", ++count);
    }
    printf("Done\n");
}

$ ./loop-calloc
...
Currently allocating 2881 MB
Currently allocating 2882 MB
Currently allocating 2883 MB
Ucciso
```

Il processo satura lo spazio su RAM + swap

```
$ dmesg | tail
...
[  653.021745] Out of memory: Killed process 2888 (loop-calloc) ...
[  653.112625] oom_reaper: reaped process 2888 (loop-calloc) ...
```

Out-of-memory killer

In caso di scarsità di memoria (sia su RAM sia su swap), il kernel **uccide un processo** (*Out-Of-Memory killer*)

```
hog invoked oom-killer: gfp_mask=0x201da, order=0, oom_adj=0
hog cpuset=/ mems_allowed=0
Pid: 27104, comm: hog Not tainted 2.6.33.3-85.fc13.i686 #1
.....................
DMA: 3*4kB 4*8kB 4*16kB 4*32kB 3*64kB 2*128kB 5*256kB 4*512kB
2*1024kB 1*2048kB 0*4096kB = 8108kB
Normal: 56*4kB 156*8kB 91*16kB 58*32kB 42*64kB 33*128kB 25*256kB
10*512kB 9*1024kB 4*2048kB 0*4096kB = 40624kB
HighMem: 0*4kB 0*8kB 1*16kB 1*32kB 1*64kB 1*128kB 1*256kB 0*512kB
0*1024kB 0*2048kB 0*4096kB = 496kB
143 total pagecache pages
0 pages in swap cache
Swap cache stats: add 966799, delete 966799, find 1057/1466
Free swap  = 0kB
Total swap = 0kB
524230 pages RAM
297928 pages HighMem
8711 pages reserved
514 pages shared
502119 pages non-shared
Out of memory: kill process 27104 (hog) score 486699 or a child
Killed process 27104 (hog) vsz:1946796kB, anon-rss:1944356kB, file-rss:28kB
```

Saturazione delle memory zones

Punteggio legato alla qtà di memoria consumata, e alla priorità

Deferred ("lazy") memory allocation

```
$ cat loop.c

int main() {
    void* block=NULL;
    int count=0;
    while(1) {
          block = (void *) malloc(1024*1024);     // alloca 1 MB
          if (!(block)) break;
          // il blocco rimane inutilizzato
          printf("Currently allocating %d MB\n", ++count);
    }
    printf("Done\n");
}

$ ./loop
...
Currently allocating 4072 MB
Currently allocating 4073 MB
Currently allocating 4074 MB
Currently allocating 4075 MB
Done
```

Il processo riempie tutto il suo spazio di memoria virtuale (4GB)

Prerequisiti:

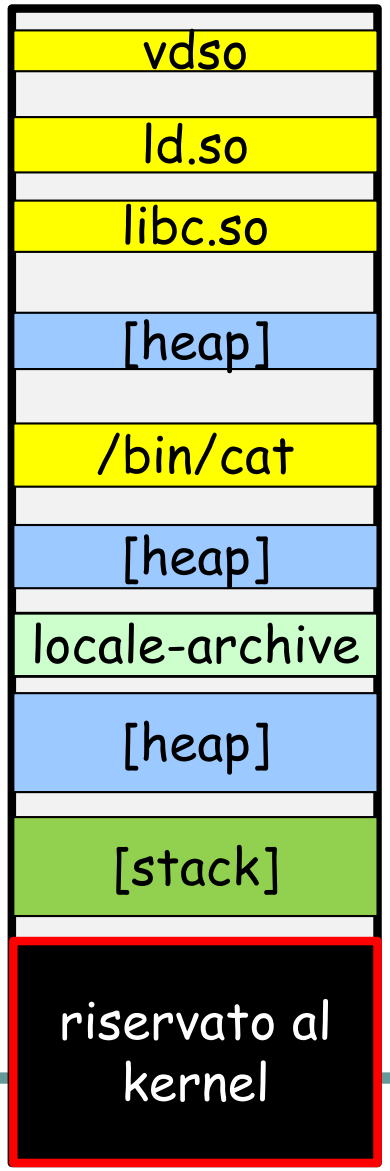
```
$ sudo apt-get -y install gcc-multilib g++-multilib
```

Compilare con:

```
$ gcc -m32 loop.c -o loop
```

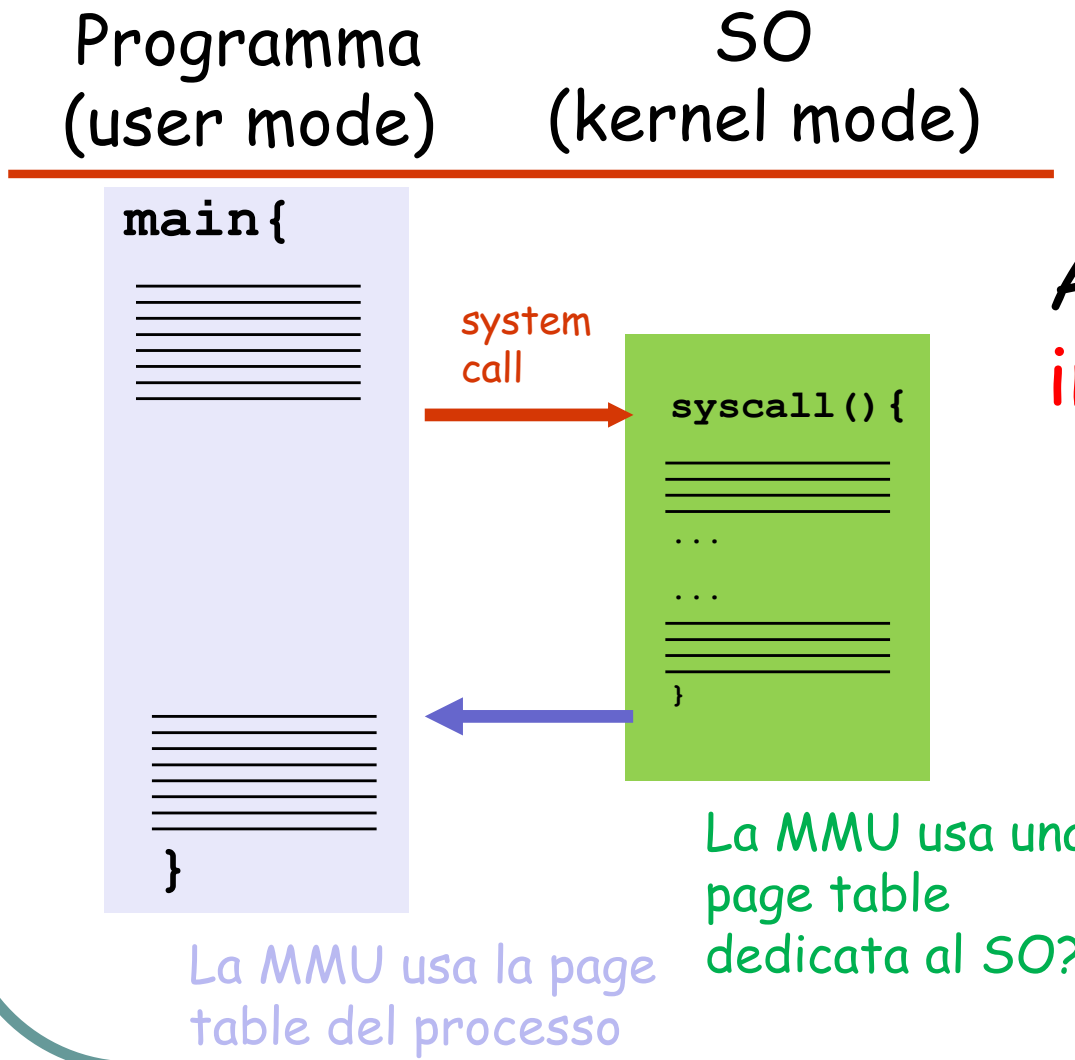
Spazio riservato al kernel (user/kernel memory split)

Spazio virtuale di un processo in Linux



Insieme di indirizzi virtuali **usati dal kernel** (es. system call)

User/kernel memory split

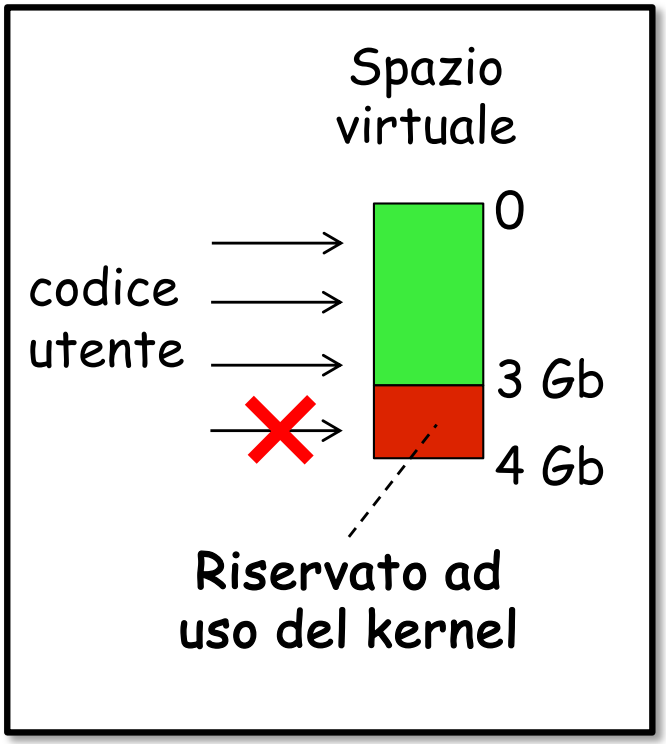


Anche il SO utilizza
indirizzi virtuali

No, a causa di minori
prestazioni
(es. TLB flush al
cambio di contesto)

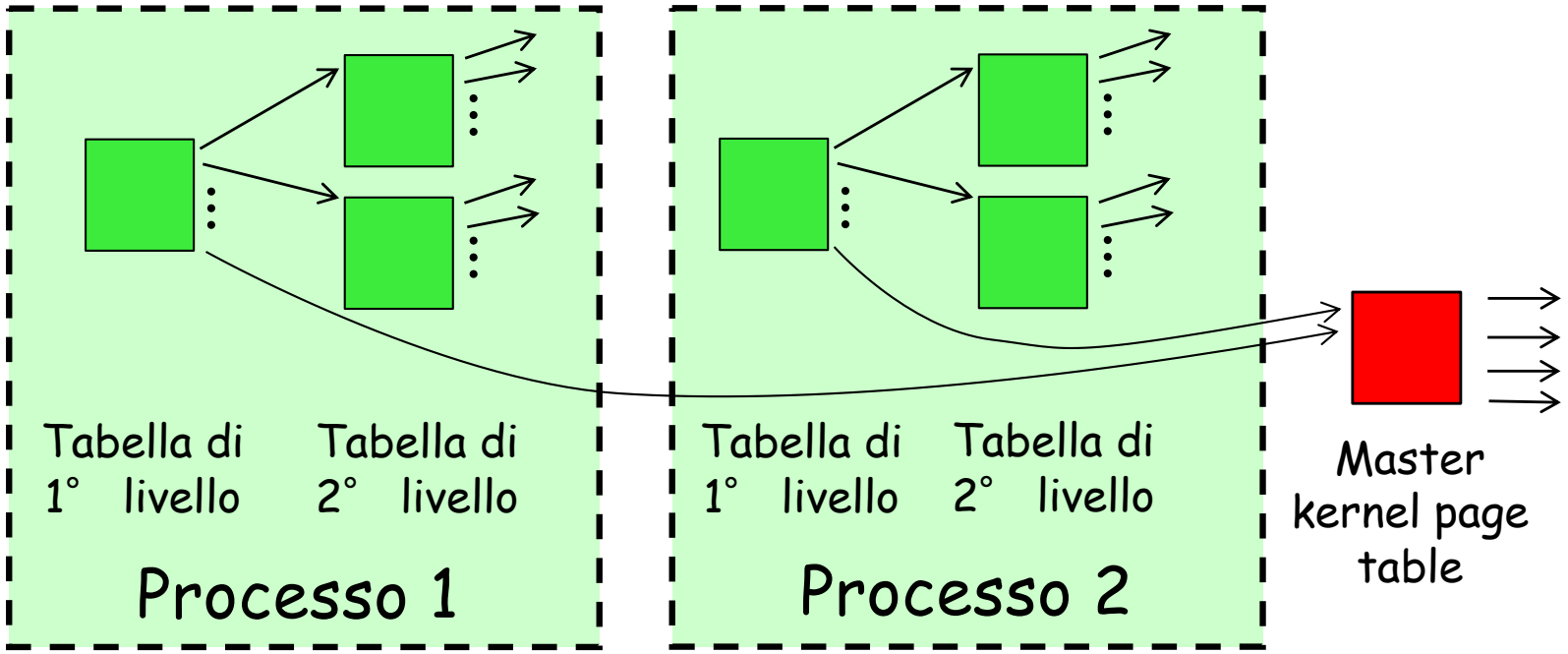
User/kernel memory split

- Il kernel riusa parte dello spazio virtuale del processo interrotto
- La tabella delle pagine rimane inalterata (no flush)
- Migliori prestazioni



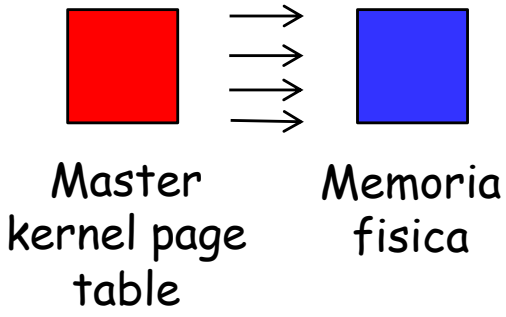
Master kernel page table

- Lo spazio virtuale riservato al kernel è gestito dalla **master kernel page table**
- Uguale e condivisa fra tutti i processi



Master kernel page table

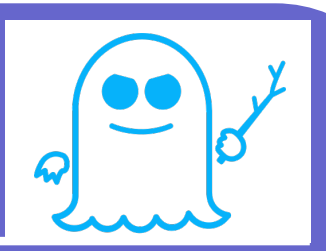
- Associazione "lineare" tra indirizzi virtuali e fisici



Indirizzo virtuale = Indirizzo fisico + offset

0xC0000123 0x00000123 0xC0000000

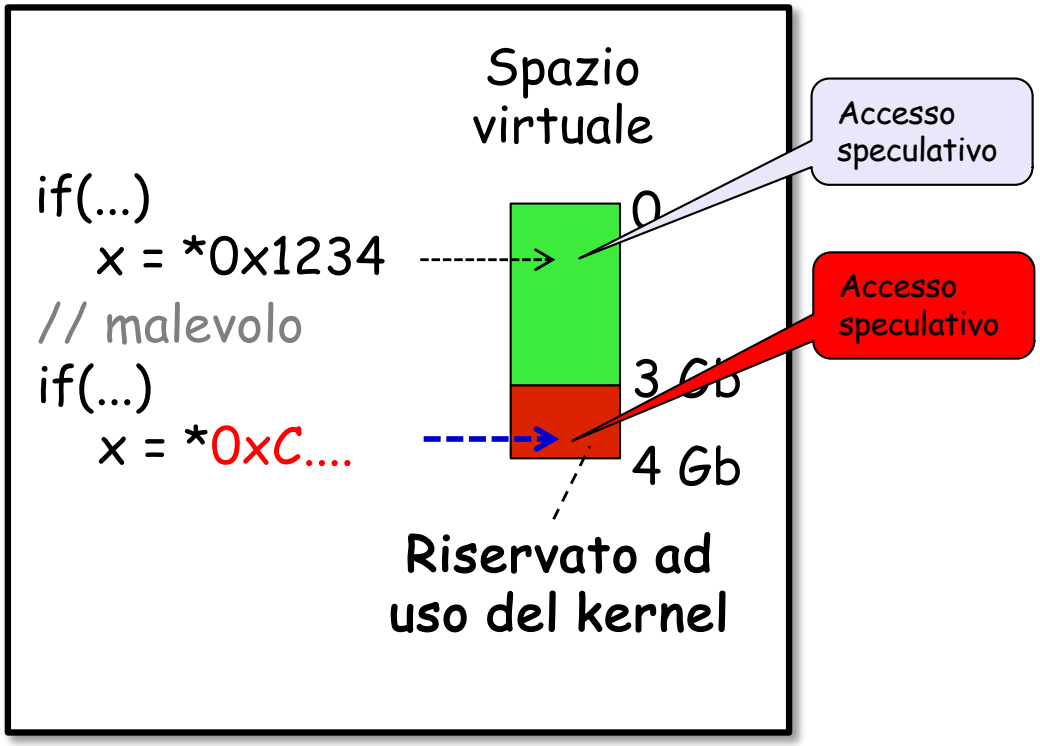
Meltdown/Spectre



- Lo user/kernel split è stato causa di problemi di sicurezza
- Le CPU moderne fanno esecuzione speculativa
  - La CPU tenta di "indovinare" le prossime istruzioni, e le esegue in anticipo
  - La speculazione lascia tracce nella cache, branch predictor, etc.

https://meltdownattack.com/

Meltdown/Spectre

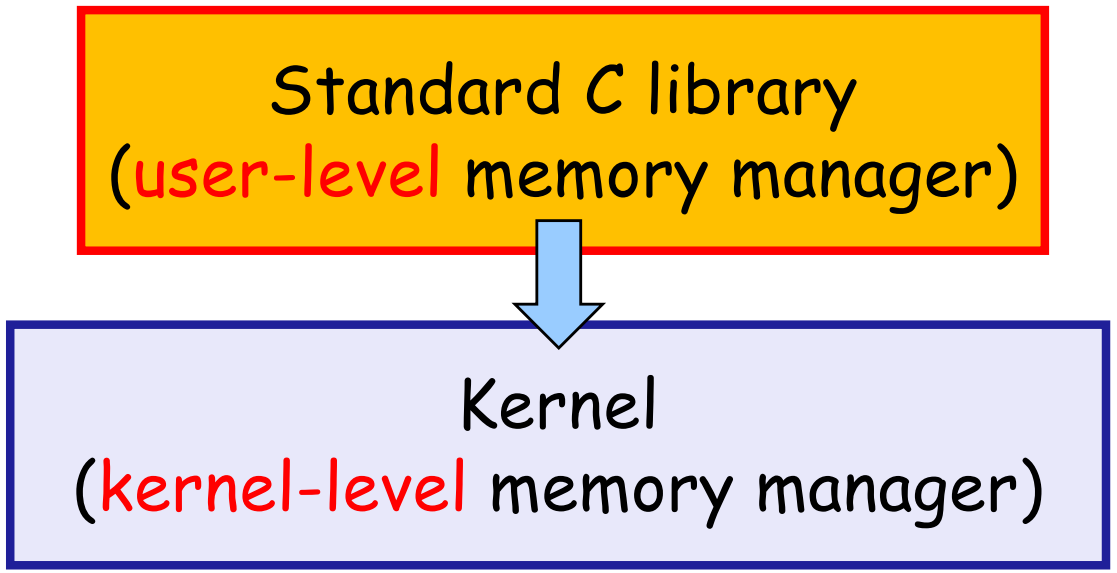


Per prevenire gli attacchi, occorre dedicare al kernel una page table separata, rallentando le performance

https://meltdownattack.com/

Gestione della memoria a livello kernel

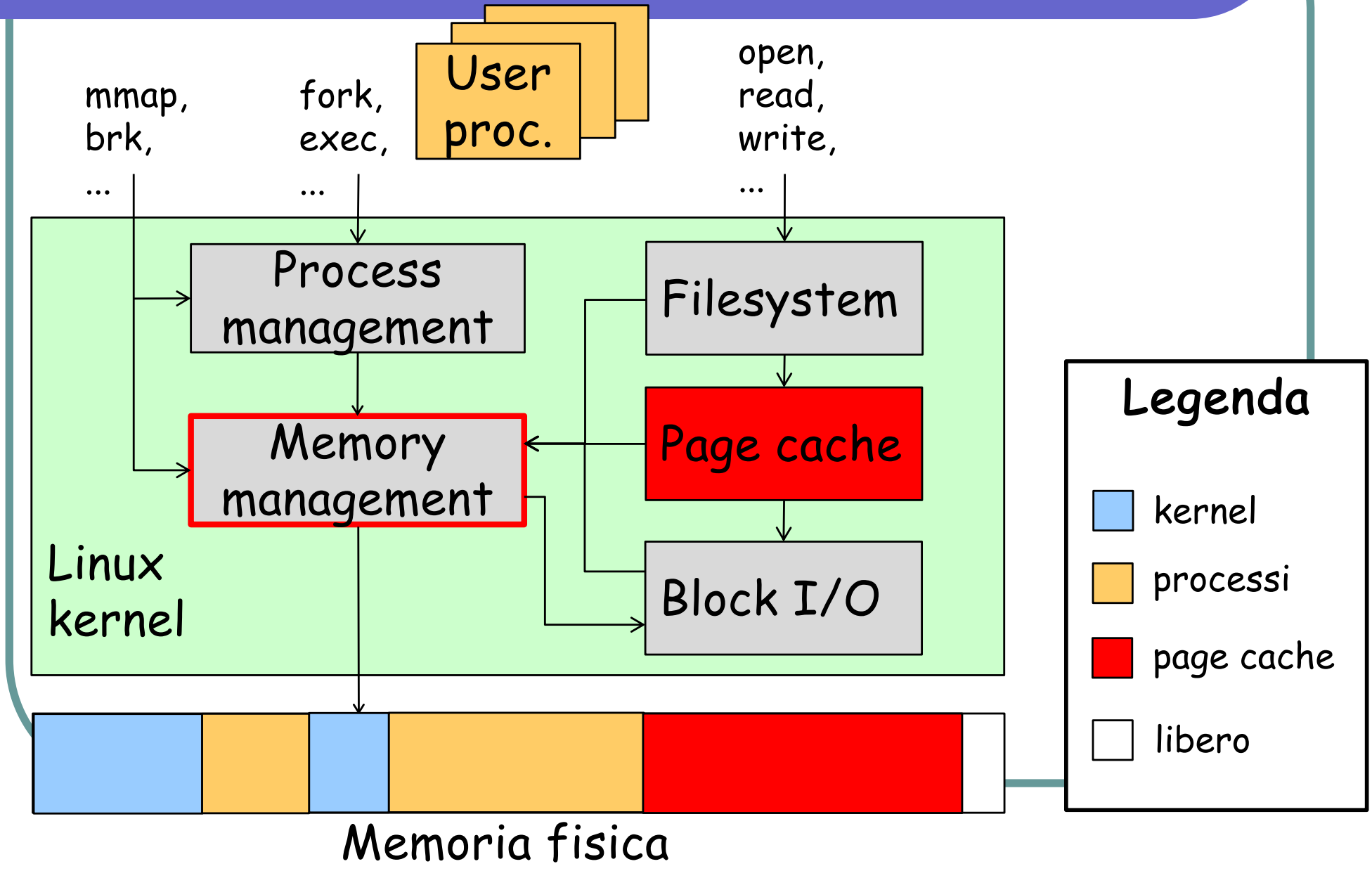
- Il **kernel** contiene a sua volta un gestore della **memoria fisica**



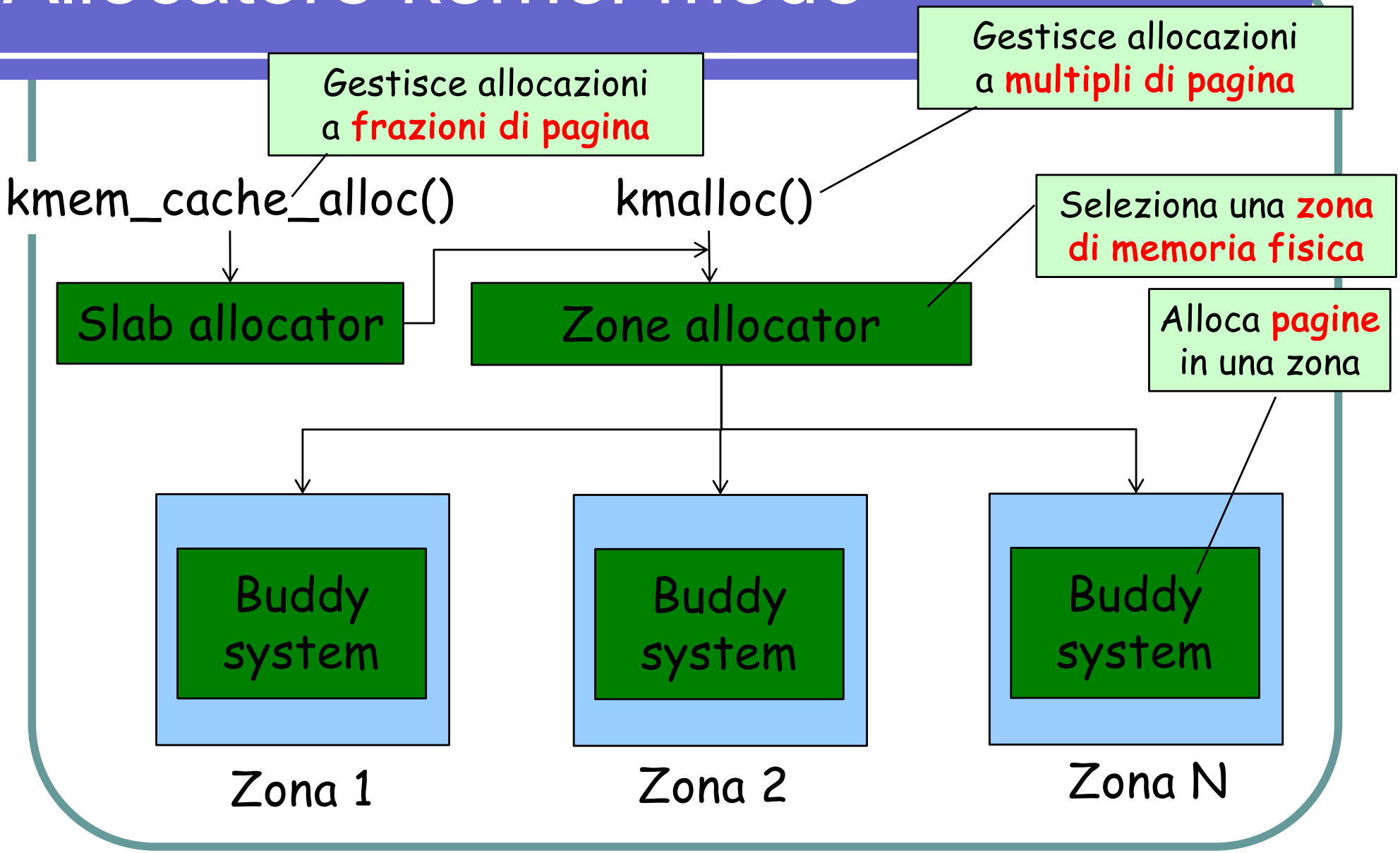
Alloca pagine sia ai processi, sia per uso interno al kernel:

- **Buffer** per i trasferimenti di **I/O**
- **Caching** di dati recenti letti/scritti sui dischi
- Descrittori dei **processi**
- Tabelle delle pagine
- Metadati del **filesystem**
- ...

Gestione della memoria a livello kernel



Allocatore kernel-mode

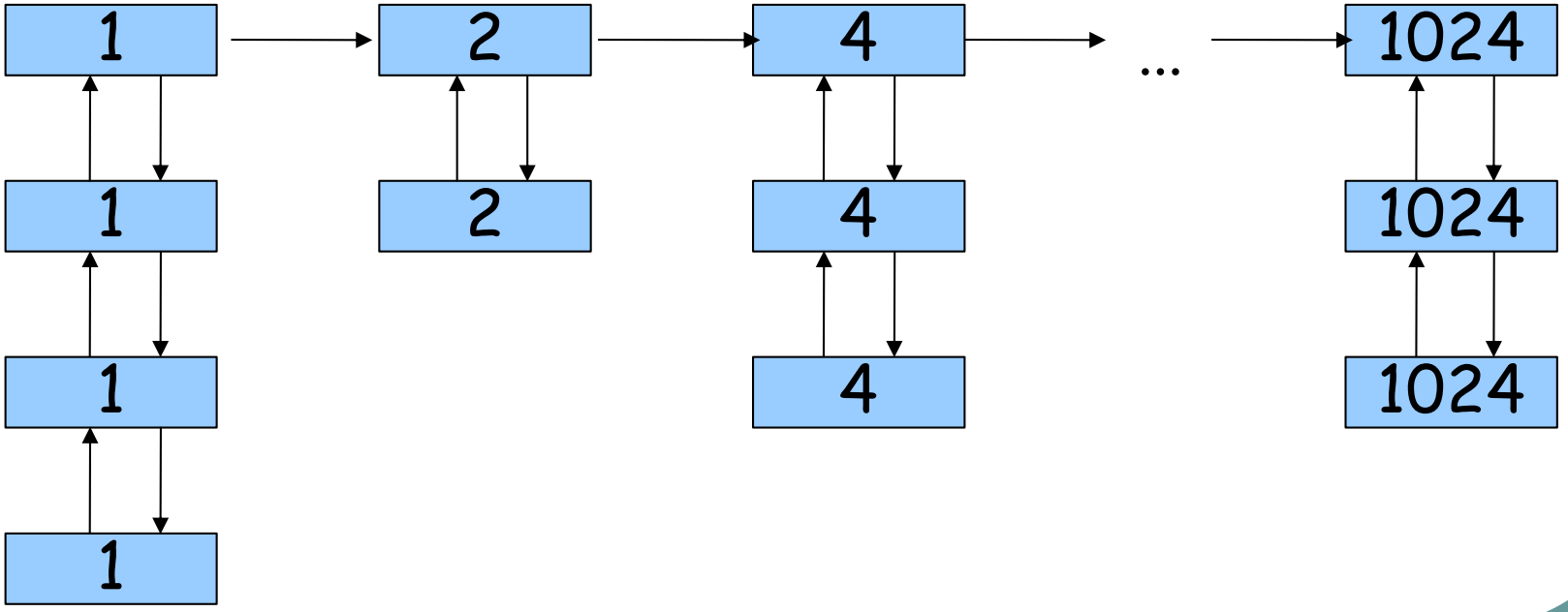


Buddy System

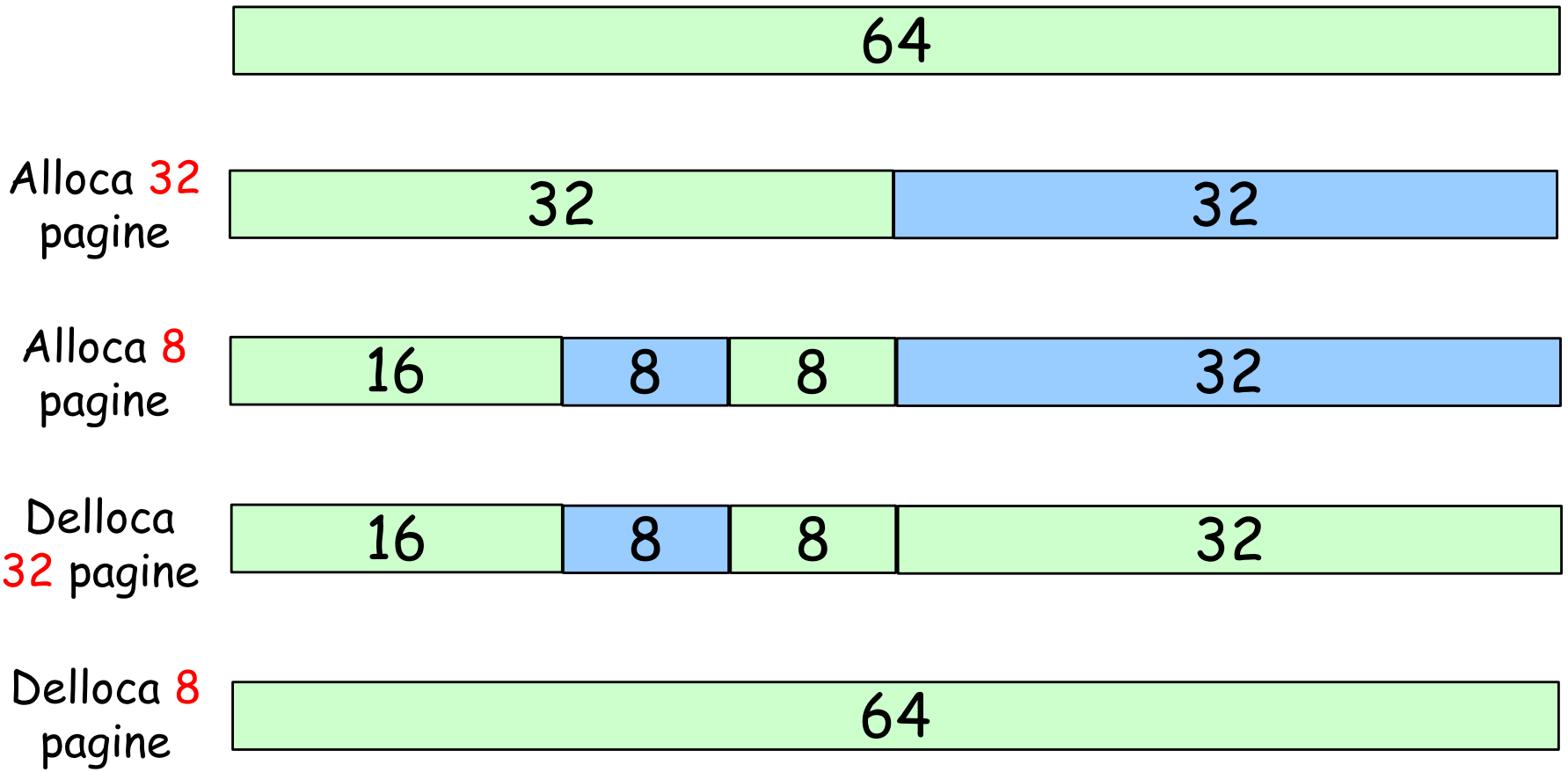
- Algoritmo usato per allocare gruppi di **pagine fisiche contigue**
- In teoria, le pagine fisiche possono essere allocate in modo non-contiguo...
- ...allocare pagine fisiche contigue permette di:
  - ridurre la dimensione della tabella (gerarchica) delle pagine
  - avere minor consumo di cache e di TLB
  - avere compatibilità con DMA e dispositivi di I/O legacy

Buddy System

- Una allocazione avviene per **multipli di 2^k pagine contigue** (1,2,4,8,...,1024 pagine)
- Il kernel traccia i blocchi liberi tramite **linked list**
  - Le liste sono visibili tramite il file /proc/buddyinfo



Buddy System: esempio



Buddy System: algoritmo

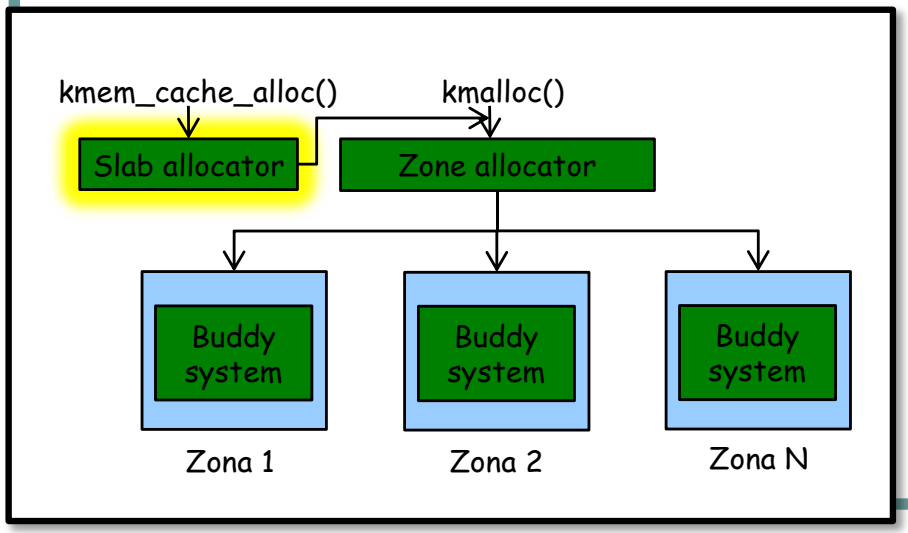
- Se un blocco di dimensione 2^k è già disponibile, la richiesta è subito soddisfatta
- Altrimenti, si verifica se esiste un blocco di dimensione 2^(k+1); se sì, lo si divide in 2 parti
  - La prima metà viene allocata, la seconda viene inserita nella lista dei blocchi con 2^k pagine
- Altrimenti, si cerca un blocco di dimensione 2^(k+2); se sì, lo si divide in 4 parti
  - Una parte viene allocata; gli altri 3 blocchi finiscono nelle liste
- ...

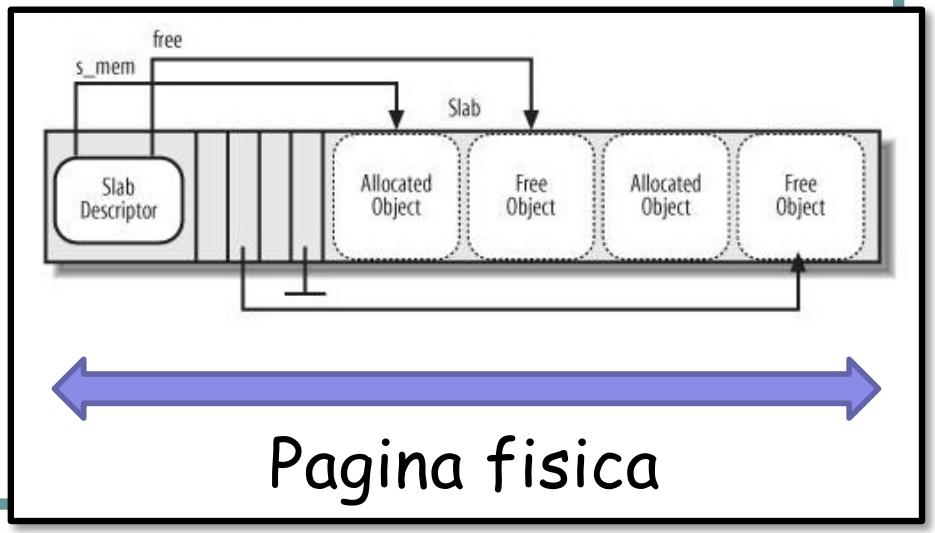
Buddy System: algoritmo

- Quando un blocco di 2^k pagine viene de-allocato, si aggiunge un blocco alla lista dei blocchi di dimensione 2^k
- Se vi è un altro blocco libero adiacente con 2^k pagine, i due blocchi vengono uniti
  - Essi sono rimossi dalla lista coi blocchi di 2^k pagine; un blocco libero è aggiunto alla lista di quelli con 2^(k+1) pagine
- L'unione di più blocchi contigui viene ripetuta ricorsivamente, finchè è possibile

Slab Allocator

- Lo Slab Allocator permette di allocare **porzioni piccole** di memoria, di dimensione variabile (da 32 a 4080 byte)
- Pagine con oggetti **pre-allocati** di stessa dimensione
- Utile per strutture dati piccole come PCB, buffer di I/O, descrittori di file aperti, ...





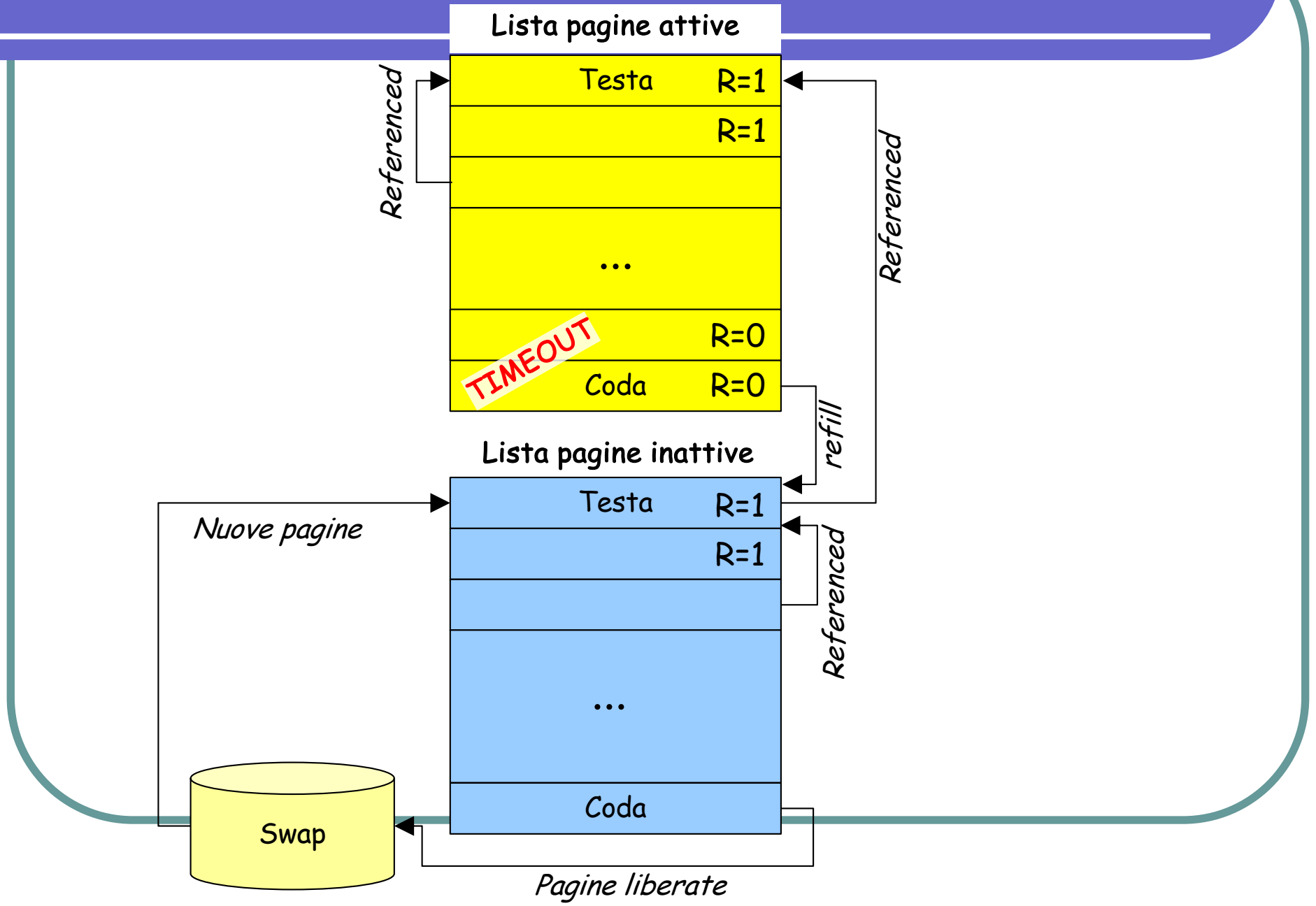
Page Frame Reclaiming Algorithm

- L'algoritmo di rimpiazzo delle pagine (**PFRA**) è una variante dell'algoritmo ***second chance***

- Viene eseguito quando:
  - La memoria libera è al di sotto di una soglia
  - Periodicamente

https://www.kernel.org/doc/gorman/html/understand/understand013.html

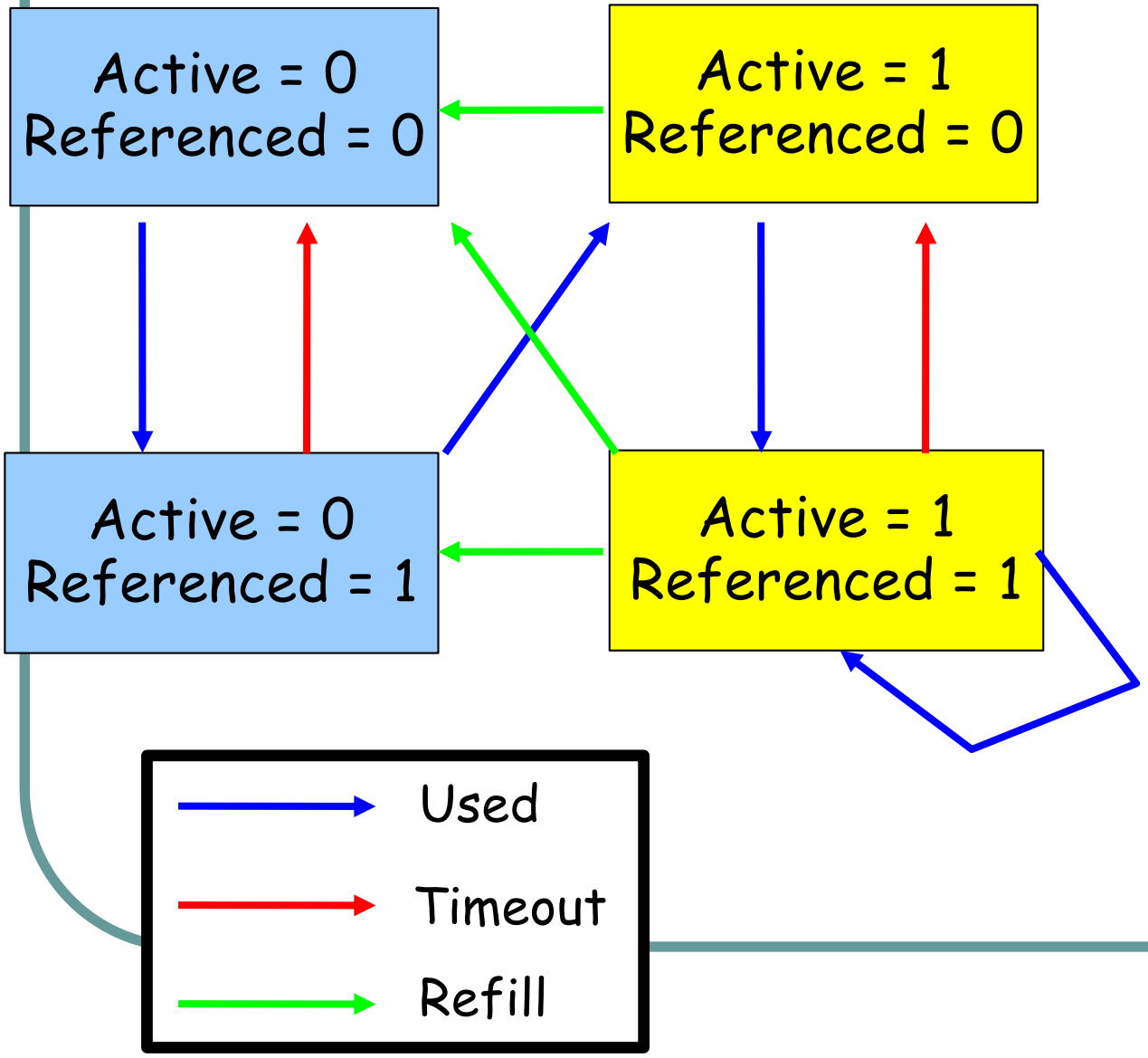
Page Frame Reclaiming Algorithm



Page Frame Reclaiming Algorithm

- Le nuove pagine sono inserite in una lista globale di "pagine inattive"
- PFRA scansiona le liste, spostando le pagine referenced verso la cima
- La lista "active" approssima il working set
- Le pagine non utilizzate sono “degradate”, finendo infine nella swap

Page Frame Reclaiming Algorithm



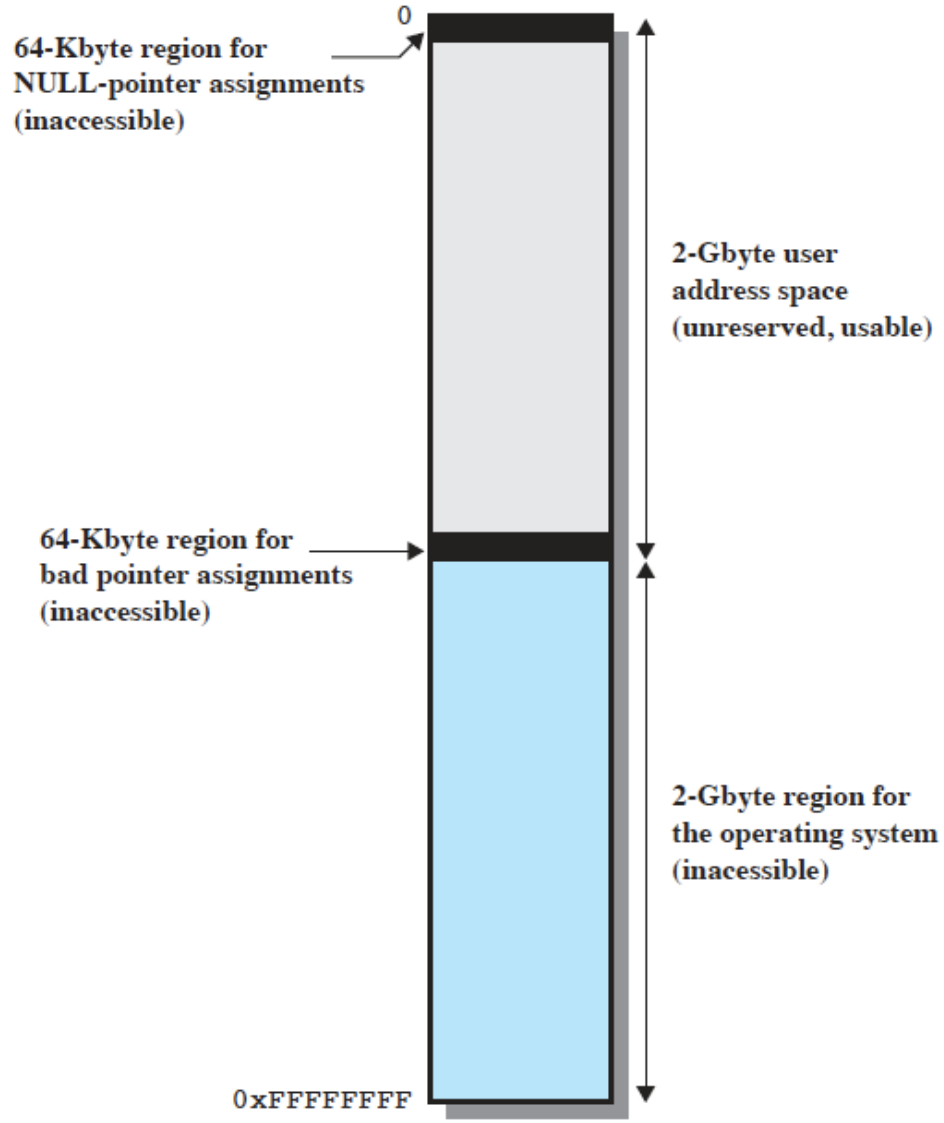
- **Referenced**: la pagina è stata recentemente acceduta (azzerato periodicamente)
- **Active**: la pagina è stata acceduta più volte nel recente passato
- La selezione si focalizza sulle pagine non attive

- Gestione della memoria in Windows

Windows Virtual Address Map

- Sulle piattaforme a 32 bit, ogni processo Windows "vede" 4 GB di memoria
- Una porzione di questo spazio (2 GB) è dedicata al sistema operativo
  - Ogni processo ha a disposizione 2 GB di spazio di indirizzamento virtuale
  - Tutti i processi condividono i 2 GB allocati al sistema
- Un'opzione consente di aumentare a 3 GB la memoria utente, e ridurre a 1 GB la memoria di sistema
- Sulle piattaforme a 64 bit, ogni processo utente può avere accesso fino a 8 TB di memoria virtuale, su Windows Vista e Windows 7

Spazio di indirizzamento a 32 bit



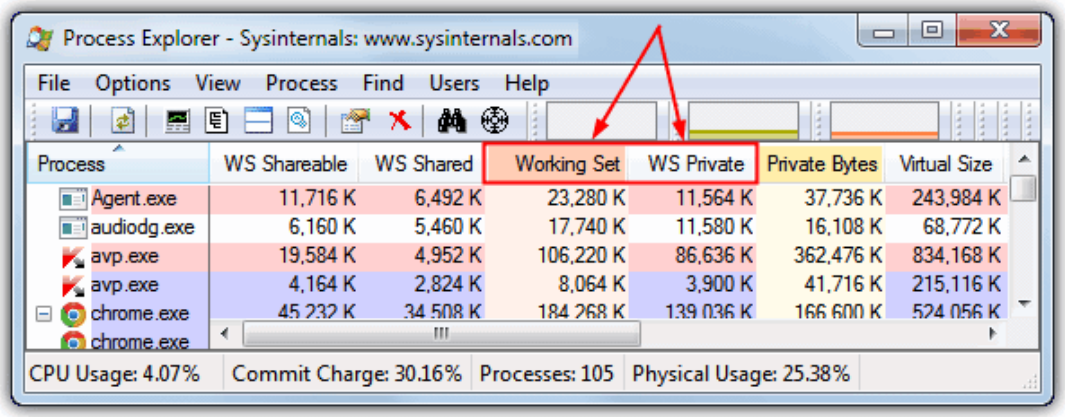
- 0x00000000-0x0000FFFF: allocate per individuare i null pointer
- 0x00010000-0x7FFEFFFF: allocate ai processi utente
- 0x7FFF0000-0x7FFFFFFF: allocate per individuare i bad pointer
- 0x80000000-0xFFFFFFFF: allocate al sistema operativo (Windows Executive, kernel, drivers, …)

Windows Paging

- Windows usa la paginazione su richiesta per gruppi di pagine (*demand paging with **clustering***)
  - Regioni di indirizzi virtuali contigui di 64Kb
- Ogni cluster della memoria può essere in uno dei seguenti stati:
  - **Available**: memoria non utilizzata dal processo
  - **Reserved**: memoria messa da parte per il processo dal SO, preservata per successivi usi (es. spazio preallocato per far crescere lo stack)
  - **Committed**: memoria allocata ed utilizzata dal processo; le relative pagine possono risiedere sia in memoria centrale, sia sul disco (nel file di paging)
- La distinzione tra reserved e committed è utile per usare efficientemente lo spazio di memoria virtuale, e per consentire di riservare pagine prima della loro effettiva allocazione

Gestione del Working Set

- Il SO tiene traccia del numero di pagine (**variabile**) attribuite ad ogni processo
- La sostituzione delle pagine avviene nel contesto del processo che produce assenze di pagina
- Il SO garantisce di assegnare un **minimo insieme di lavoro** (*working set minimum = 50 pagine*)
- Se vi è molta memoria disponibile, il SO incrementa il working set fino a un **massimo** (*working set maximum = 345 pagine*).
  - Quando occorre un page fault, la pagina richiesta è aggiunta al processo senza rimuovere una vecchia pagina → il working set cresce



Gestione del Working Set

- Se la memoria libera scende sotto una soglia critica, si applica la **regolazione automatica dell'insieme di lavoro** (*automatic working set trimming*) per riportare il valore sopra la soglia, riducendo i processi che:
  - hanno ottenuto un numero di pagine superiori a quello minimo
  - sono stati idle per molto tempo e non hanno acceduto alle pagine del working set
- La gestione del working set è fatta da una routine denominata **Working Set Manager**, eseguita nel contesto di un thread di sistema (**Balance Set Manager Thread**)