UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Sistemi Operativi

Gestione della Memoria

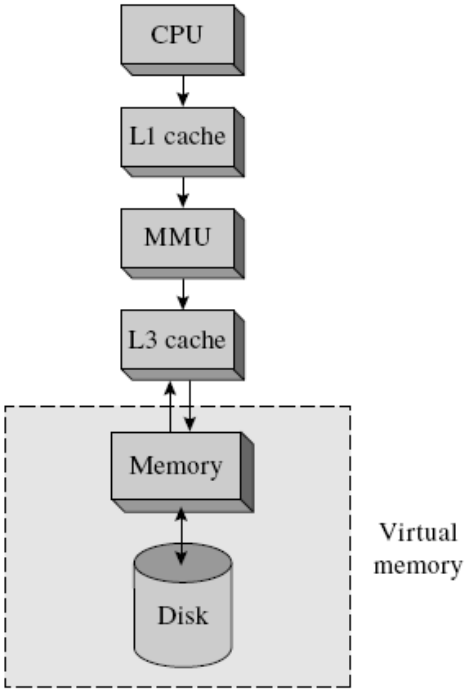
LEZIONE 19

prof. Antonino Staiano

Corso di Laurea in Informatica – Università di Napoli Parthenope

antonino.staiano@uniparthenope.it

Gestione della Gerarchia di Memoria



$$t_{ema} = \boldsymbol{h} \times t_{cache} + (1 - \boldsymbol{h}) \times (t_{tra} + t_{cache})$$
$$= t_{cache} + (1 - \boldsymbol{h}) \times t_{tra}$$

t_{ema} = tempo di accesso alla memoria effettivo
t_{cache} = tempo di accesso alla cache
t_{tra} = tempo impiegato per trasferire un blocco dalla memoria alla cache

Levels	How managed	Performance issues
Caches	Allocation and use is managed by hardware	Ensuring high hit ratios
Memory	Allocation is managed by the kernel and use of allocated memory is managed by run-time libraries	(1) Accommodating more process in memory, (2) Ensuring high hit ratios
Disk	Allocation and use is managed by the kernel	Quick loading and storing of parts of process address spaces

Virtualizzazione della Memoria

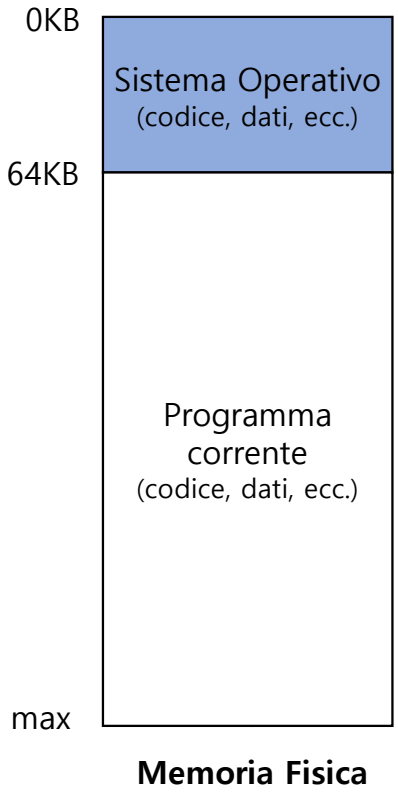
- Cosa è la virtualizzazione della memoria?
  - Il SO virtualizza la sua memoria fisica
  - In questo modo fornisce uno spazio di memoria illusorio per ogni processo
  - E' come se ogni processo usasse tutta la memoria

Benefici della Virtualizzazione

- Semplicità di utilizzo durante la programmazione
- Efficienza della memoria in termini di tempo e spazio
- Garanzia di isolamento per i processi e per il SO
  - Protezione da accessi impropri da parte di altri processi

Il SO nei primi sistemi

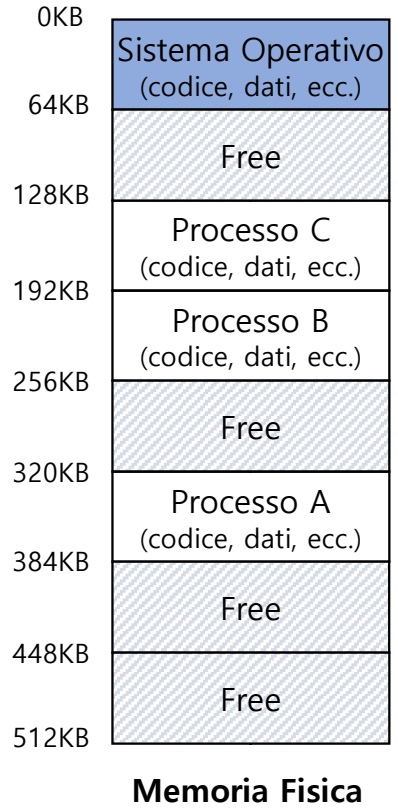
- Un solo processo caricato in memoria
  - Scarso utilizzo ed efficienza



Multiprogrammazione e Time Sharing

- Più processi caricati in memoria
  - Esecuzione di uno per volta per un tempo breve
  - Commutazione tra loro in memoria
  - Incremento di utilizzazione ed efficienza

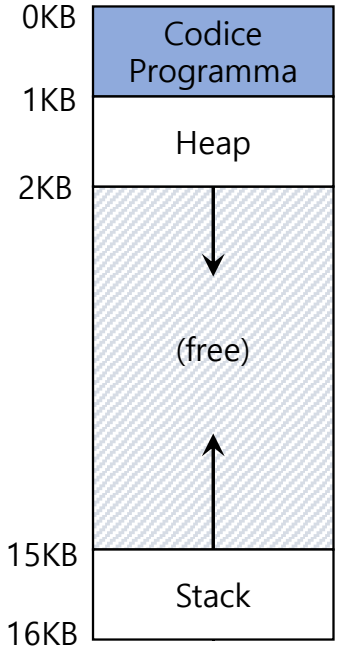
- Causa di un problema rilevante di protezione
  - Accessi impropri da altri processi



Memoria Fisica

Spazio di Indirizzamento

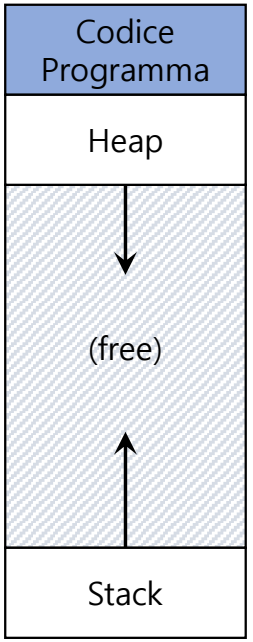
- Il SO crea un'astrazione della memoria fisica
  - Lo spazio di indirizzamento contiene tutto quanto è necessario per un processo
    - Codice programma, heap, stack ecc.



Spazio Indirizzamento

Spazio di Indirizzamento (cont.)

- Codice
  - Dove risiedono le istruzioni
- Heap
  - Per l'allocazione dinamica della memoria
    - `malloc` nel linguaggio C
    - `new` nei linguaggi OO
- Stack
  - Memorizza gli indirizzi di ritorno o valori
  - Contiene variabili locali e gli argomenti delle routine



Spazio Indirizzamento

Indirizzi Virtuali

- In un programma in esecuzione ogni indirizzo è virtuale
  - Il SO traduce l'indirizzo virtuale in indirizzo fisico

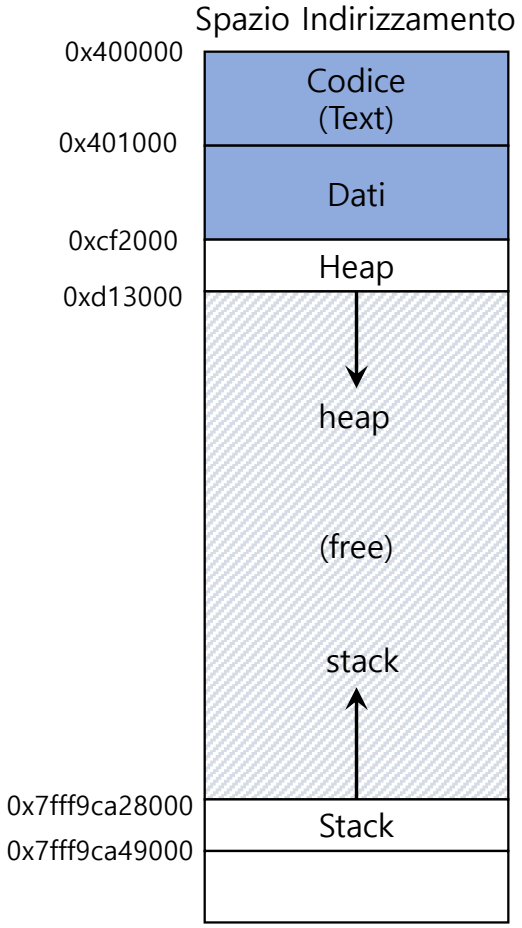
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

    printf("posizione del codice  : %p\n", (void *) main);
    printf("posizione dello heap  : %p\n", (void *) malloc(1));
    int x = 3;
    printf("posizione dello stack : %p\n", (void *) &x);

    return x;
}
```

Programma per stampare indirizzi

Indirizzi Virtuali (cont.)

- Output in una macchina Linux a 64-bit

```
posizione del codice   : 0x40057d
posizione dello heap   : 0xcf2010
posizione delllo stack : 0x7fff9ca45fcc
```



Esempio: Traduzione Indirizzo

- Codice C

```
void func()
        int x=3000;
        ...
        x = x + 3; // linea di codice a cui siamo interessati
```

  - **Carica** un valore dalla memoria
  - **Incrementa** di tre
  - **Memorizza** il valore in memoria

Esempio: Traduzione Indirizzo (cont.)

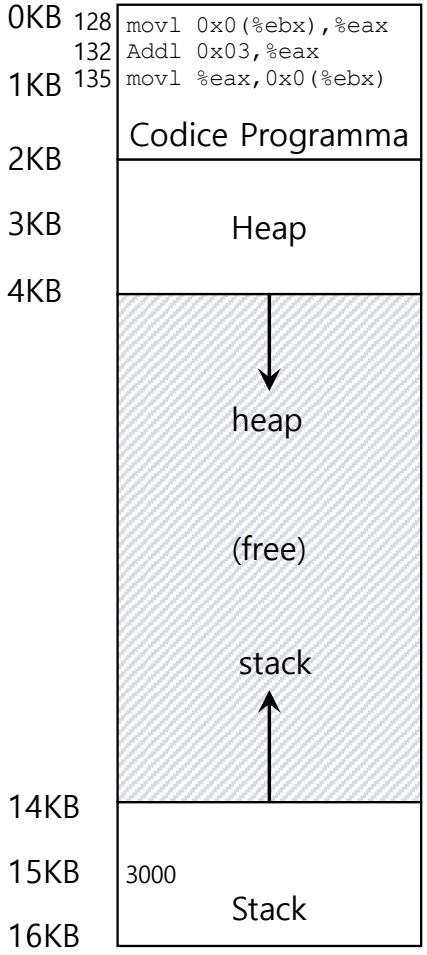
- Assembly

```
128 : movl 0x0(%ebx), %eax          ; carica 0+ebx in eax
132 : addl $0x03, %eax              ; addiziona 3 al registro
135 : movl %eax, 0x0(%ebx)          ; memorizza eax in memoria
```

  - Supponiamo che l'indirizzo di `'x'` sia messo nel registro `ebx`
  - **Carica** il valore di quell'indirizzo nel registro `eax`
  - **Addiziona** 3 al registro `eax`
  - **Memorizza** il valore in `eax` in memoria

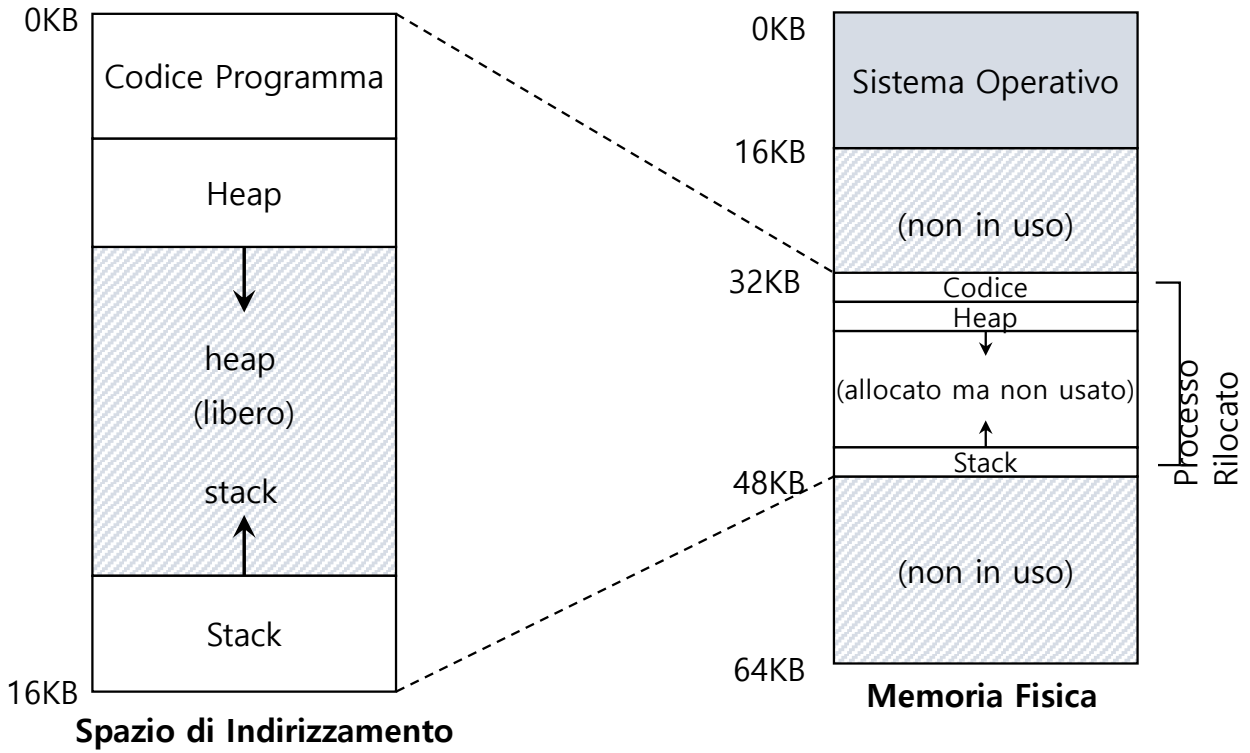
Esempio: Traduzione Indirizzo (cont.)

- Preleva (fetch) istruzione all'indirizzo 128
- Esegue (execute) questa istruzione (carica da indirizzo 15KB)
- Preleva istruzione all'indirizzo 132
- Esegue questa istruzione (nessun riferimento alla memoria)
- Preleva istruzione all'indirizzo 135
- Esegue questa istruzione (memorizza all'indirizzo 15 KB)

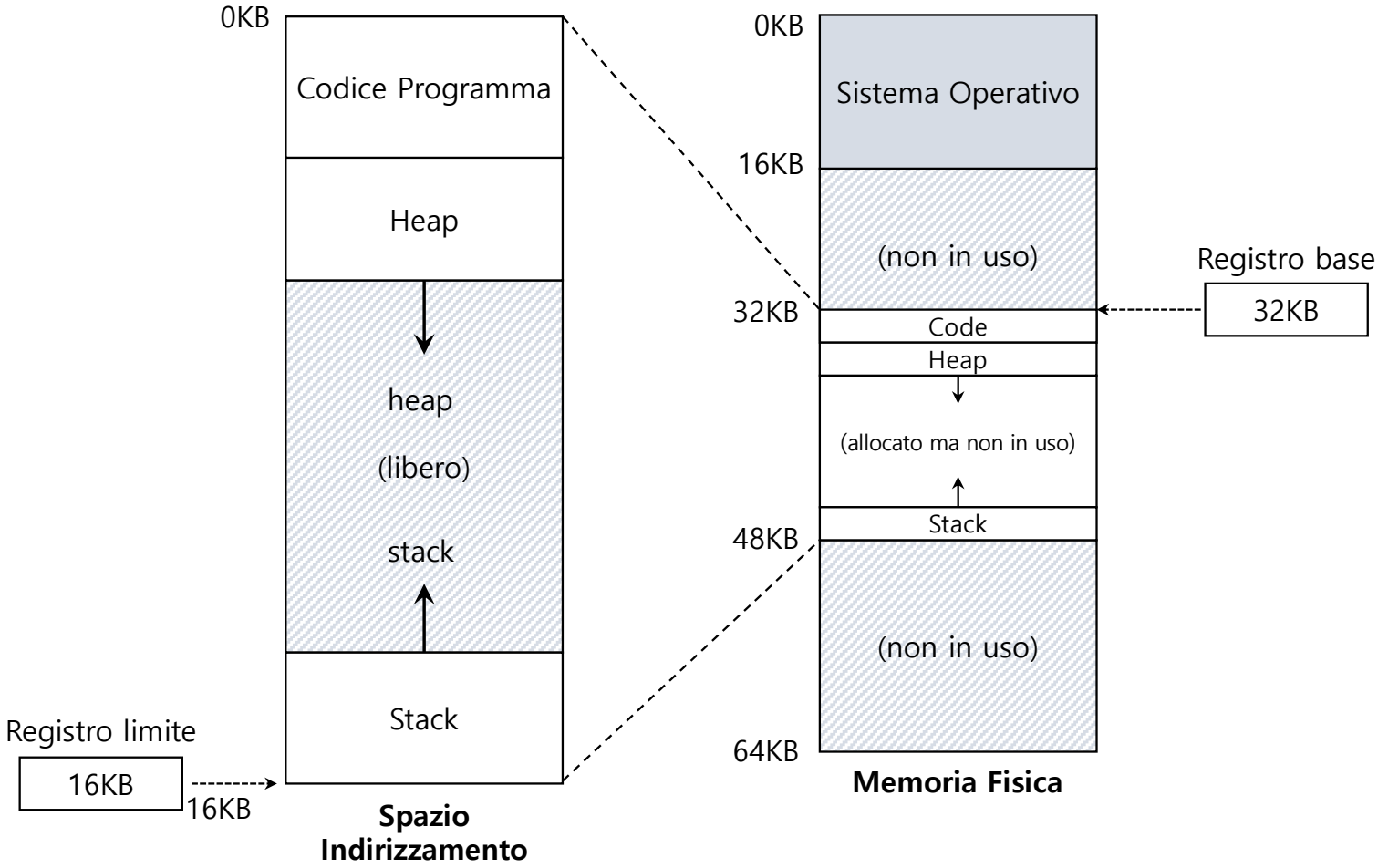


Rilocazione Dinamica: Registri Base e Limite

- Il SO pone il processo **da qualche altra parte** nella memoria fisica, non all'indirizzo 0
  - Lo spazio di indirizzamento inizia a 0



Registri Base e Limite



Rilocazione dinamica (basata su HW)

- Quando inizia l'esecuzione di un programma, il SO decide **dove deve essere caricato il processo nella memoria fisica**
  - Imposta un valore nel registro **base**

$$indirizzo\ fisico = indirizzo\ virtuale + base$$

  - Ogni indirizzo virtuale non deve essere maggiore del limite e negativo

$$0 \leq indirizzo\ virtuale < limite$$

Rilocazione e Traduzione Indirizzo

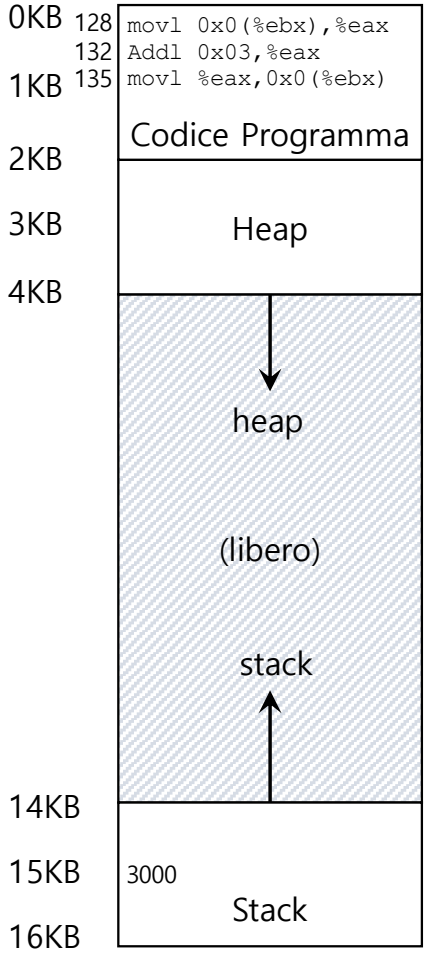
```
128 : movl 0x0(%ebx), %eax
```

- **Prelievo** istruzione all'indirizzo 128

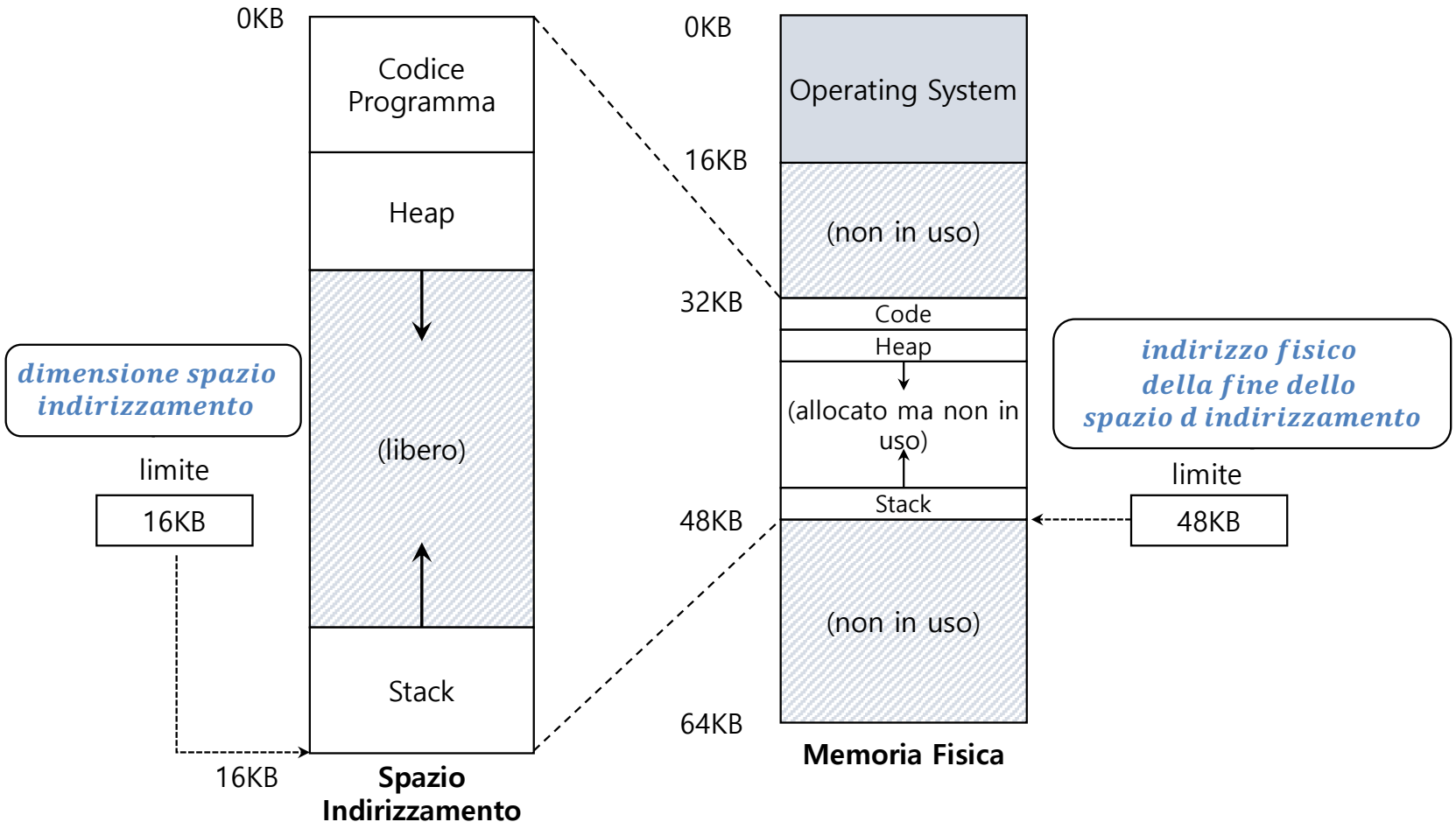
$$32896 = 128 + 32KB(base)$$

- Esegue questa istruzione
  - Carica dall'indirizzo 15KB

$$47KB = 15KB + 32KB(base)$$



Rilocazione e Traduzione Indirizzo



Requisiti Hardware

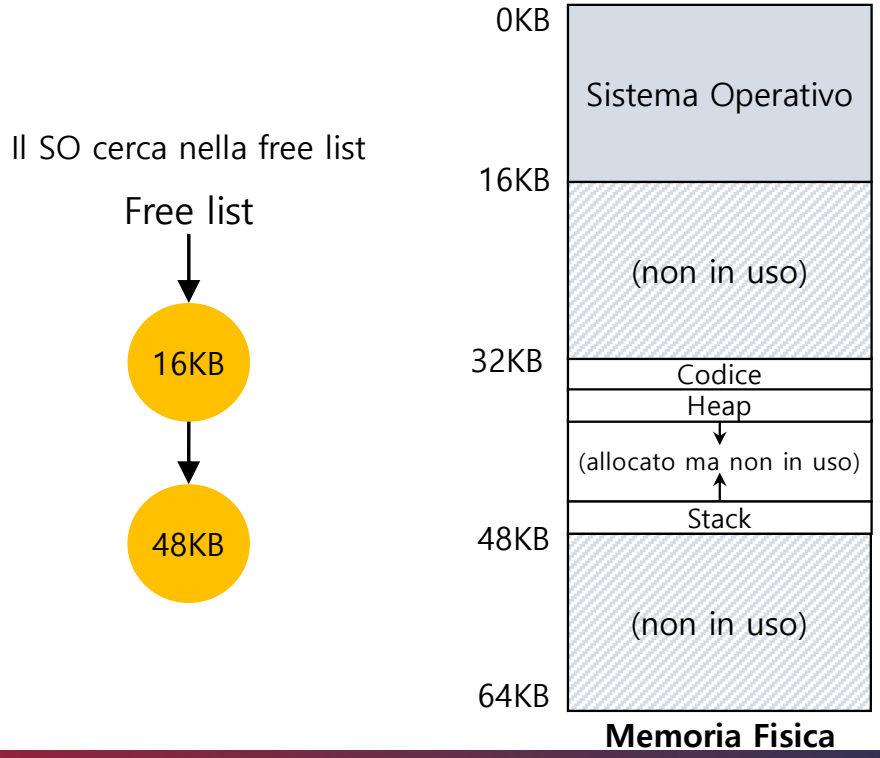
- Modalità privilegiata: previene a processi utente di eseguire operazioni privilegiate
- Registri base/limite: coppia di registri della CPU per supportare la traduzione ed il controllo dei limiti
- Capacità di tradurre indirizzi virtuali e controllare se nei limiti; circuiteria per le traduzioni
- Istruzioni privilegiate per aggiornare registri base e limite: il SO deve impostare I valori prima che un programma utente sia eseguito
- Istruzioni privilegiate da registrare: in caso di eccezioni, il SO deve essere in grado di dire all' HW quale codice di gestore di eccezione eseguire
- Capacità di generare un'eccezione quando i processi cercano di accedere a istruzioni privilegiate o a memoria fuori limite

Criticità del SO per Virtualizzare la Memoria

- Il SO deve eseguire delle azioni per implementare l'approccio base-limite
- Tre punti critici:
  - Quando un processo è avviato
    - Trovare spazio per lo spazio di indirizzamento nella memoria fisica
  - Quando un processo è terminato
    - Reclamare la memoria per riutilizzarla
  - Quando si verifica un context switch
    - Salvataggio e memorizzazione della coppia base-limite

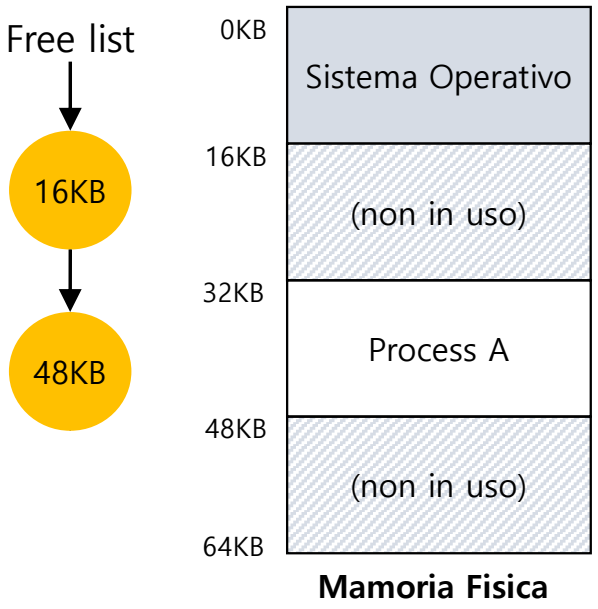
Problematiche del SO: Avvio di un Processo

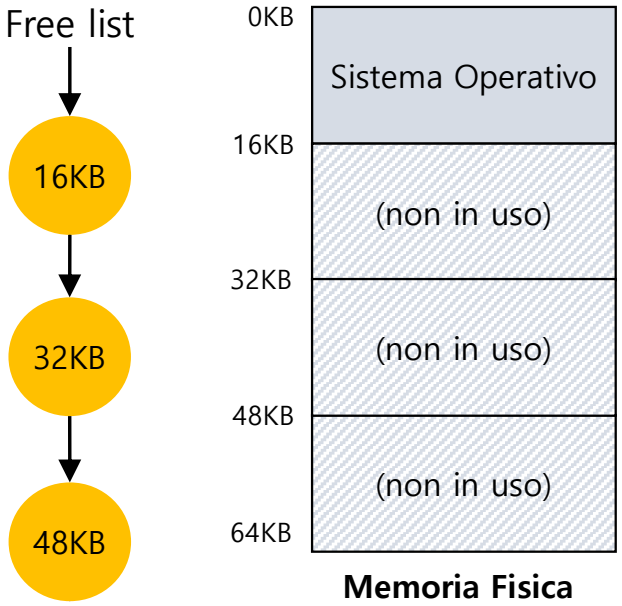
- Il SO deve trovare spazio libero per un nuovo spazio di indirizzamento
  - free list : una lista di pezzi di memoria fisica non in uso



Problematiche del SO: Fine di un Processo

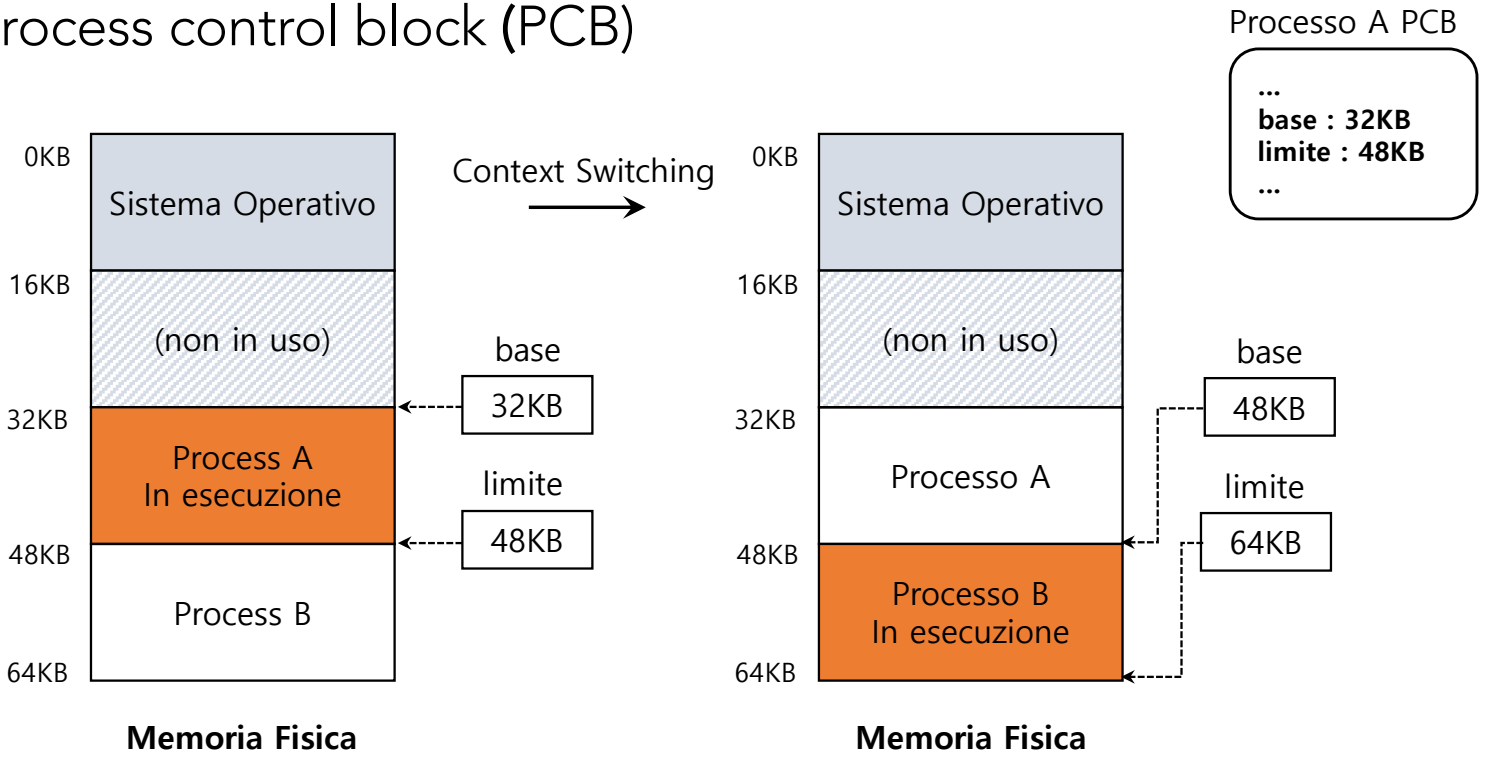
- Il SO deve reinserire la memoria appena usata nella free list





Problematiche del SO: Context Switch

- Il SO deve salvare e ripristinare la coppia base-limite
  - Nel process control block (PCB)

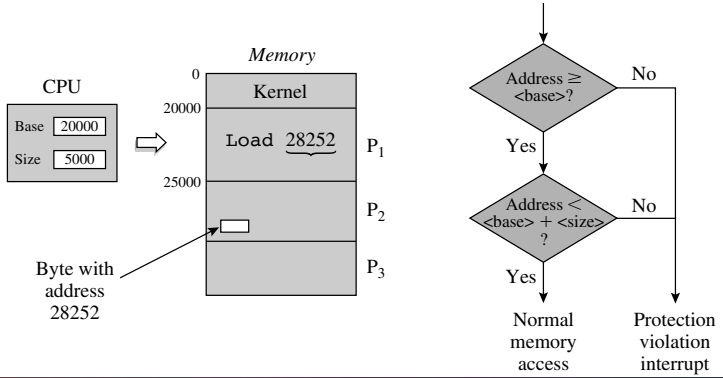


Problematiche del SO: Gestori Eccezioni

- Il SO deve fornire gestori delle eccezioni
  - Installa gli handler durante il boot (via istruzioni privilegiate)
  - Gestori eccezioni per i segmentation fault

Protezione della memoria

- La protezione della memoria usa i registri **base** e limite (**size**)
  - E' generato un interrupt di *violazione della protezione della memoria* se un indirizzo usato in un programma si trova al di fuori del loro range
    - Nel servire l'interrupt, il kernel fa l'abort del processo che ha causato l'eccezione
  - I registri base/size costituiscono il campo informazione di protezione della memoria (MPI) del PSW
    - Il kernel carica i valori appropriati mentre schedula un processo
      - Il caricamento ed il salvataggio sono istruzioni privilegiate
        - Un processo utente non può alterare i registri di protezione della memoria
    - Quando è usato un *registro di rilocazione*, questo registro ed il registro size costituiscono il campo MPI del PSW



Gestione dello Heap

- Riuso della memoria
  - Mantenimento di una Free list
  - Esecuzione nuove allocazioni usando la free list
  - Frammentazione della memoria
- Buddy system e allocatori delle potenze di 2

Allocazione della memoria ad un Processo: modello di allocazione della memoria

- Quando il kernel esegue un comando utente crea un nuovo processo e deve decidere quanto memoria allocargli per le seguenti componenti:
  - Codice e dati statici
  - Stack
  - PCD data (heap)

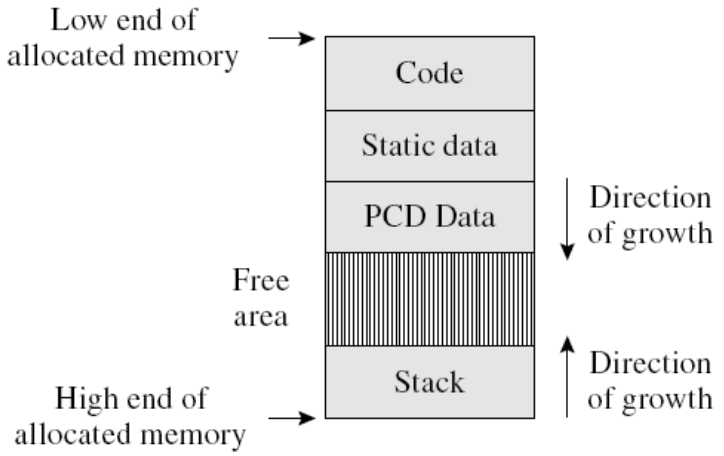


Figure 11.9 Memory allocation model for a process.

- Le routine di libreria del linguaggio di programmazione eseguono le allocazioni e le deallocazioni nello heap
- Il kernel non è coinvolto in questo tipo di gestione

Allocazione di memoria ad un processo: Stack e Heap

- Il compilatore di un linguaggio di programmazione genera il codice di un programma, alloca i suoi dati statici e crea un modulo oggetto del programma
- Il linker collega il programma con le funzioni di libreria e il supporto run-time del linguaggio di programmazione, prepara l'eseguibile e lo memorizza in un file
  - La dimensione è memorizzata nell'entrata della directory per il file
- Il supporto run-time alloca due tipi di dati durante l'esecuzione del programma
- Stack: allocazioni/deallocazioni di tipo LIFO (*push* e *pop*)
  - La memoria è allocata quando si invoca una funzione o procedura o si entra in un blocco ed è deallocata quando si ritorna o si esce dal blocco

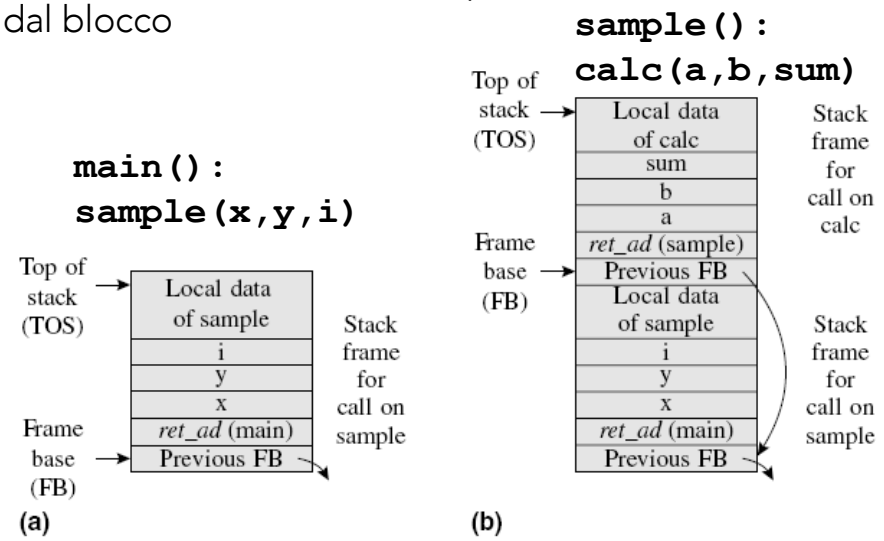


Figure 11.7 Stack after (a) `main` calls `sample`; (b) `sample` calls `calc`.

Allocazione di memoria ad un processo: Stack e Heap

- Un *heap* permette l'allocazione/ deallocazione casuale
  - Usato per i dati dinamici controllati da programma (dati PCD)

```
float *floatptr1, *floatptr2;
int *intptr;
floatptr1 = (float *) calloc(5, sizeof(float));
floatptr2 = (float *) calloc(4, sizeof(float));
intptr = (int *) calloc(10, sizeof(int));
free(floatptr2);
```

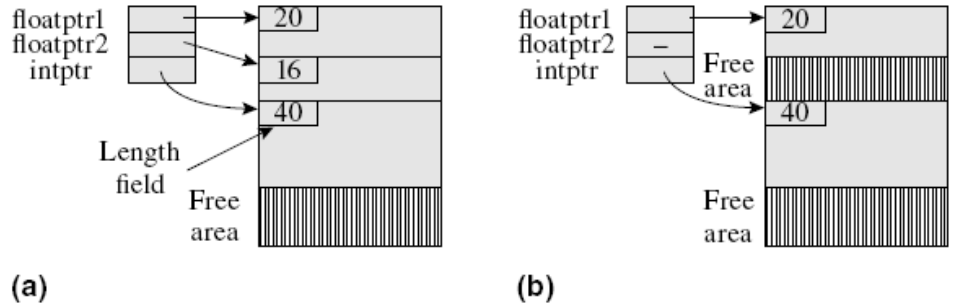


Figure 11.8 (a) A heap; (b) A "hole" in the allocation when memory is deallocated.

Riuso della memoria

Table 11.2 Kernel Functions for Reuse of Memory

Function	Description
Maintain a free list	The *free list* contains information about each free memory area. When a process frees some memory, information about the freed memory is entered in the free list. When a process terminates, each memory area allocated to it is freed, and information about it is entered in the free list.
Select a memory area for allocation	When a new memory request is made, the kernel selects the most suitable memory area from which memory should be allocated to satisfy the request.
Merge free memory areas	Two or more adjoining free areas of memory can be merged to form a single larger free area. The areas being merged are removed from the free list and the newly formed larger free area is entered in it.

Gestione delle Free list

- Per ogni area di memoria nella free list, il kernel gestisce
  - Dimensione dell'area di memoria
  - Puntatori usati per formare la lista
- Il kernel memorizza questa informazione in pochi byte all'inizio della stessa area di memoria libera

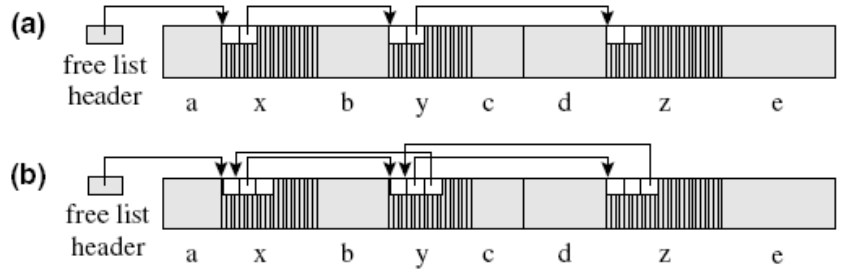


Figure 11.10 Free area management: (a) singly linked free list; (b) doubly linked free list.

Esecuzione di nuove allocazioni mediante free list

- Possono essere usate tre tecniche:
  - First-fit: usa la prima area sufficientemente grande
  - Best-fit: usa l'area sufficientemente grande più piccola
  - Next-fit: usa la successiva area sufficientemente grande

Esempio: 3 aree libere da **200**, **170** e **500** byte. I processi richiedono **100**, **50** e **400** byte.

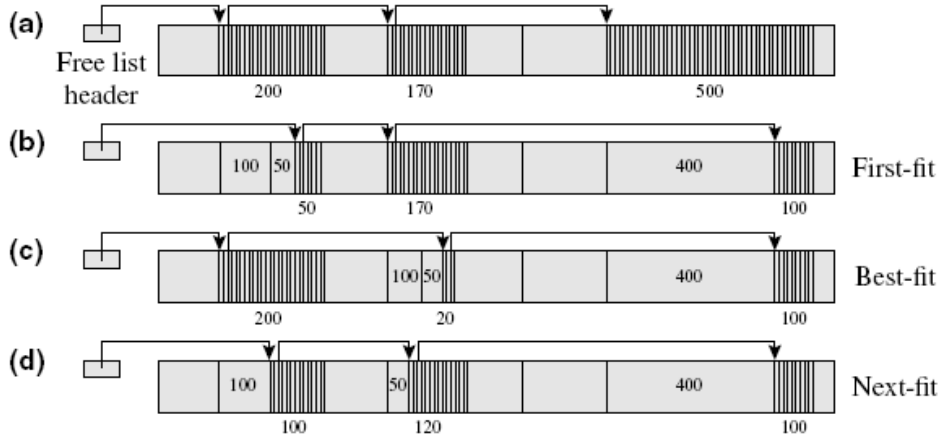


Figure 11.11 (a) Free list; (b)–(d) allocation using first-fit, best-fit and next-fit.

Frammentazione della memoria

- La frammentazione porta ad un uso inefficiente della memoria

Definition 11.3 Memory Fragmentation The existence of unusable areas in the memory of a computer system.

Table 11.3 Forms of Memory Fragmentation

Form of fragmentation	Description
External fragmentation	Some area of memory is too small to be allocated.
Internal fragmentation	More memory is allocated than requested by a process, hence some of the allocated memory remains unused.

Fusione di aree di memoria libere

- La frammentazione esterna può essere gestita unendo aree di memoria libera
- Due tecniche generali
  - Boundary tag
  - Compattazione della memoria

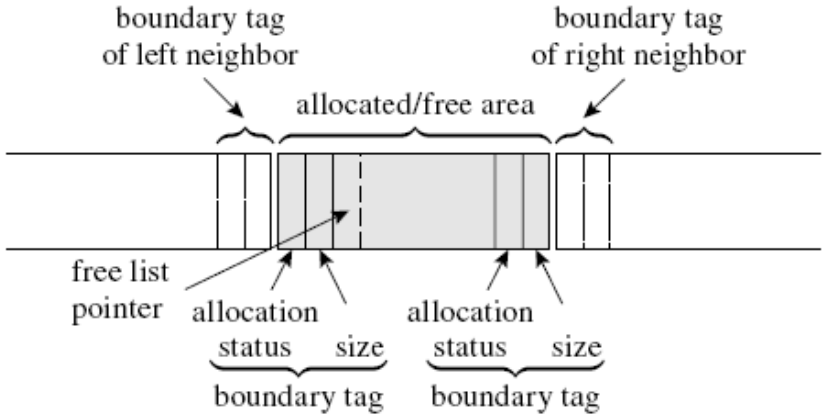


Figure 11.12 Boundary tags and the free list pointer.

Fusione di aree di memoria libera (cont.)

- Un tag è un descrittore di stato di un'area di memoria
  - Quando un'area di memoria diviene libera, il kernel controlla i tag boundary delle sue aree vicine
    - Se un'area vicina è libera, è unita con l'area appena liberatasi

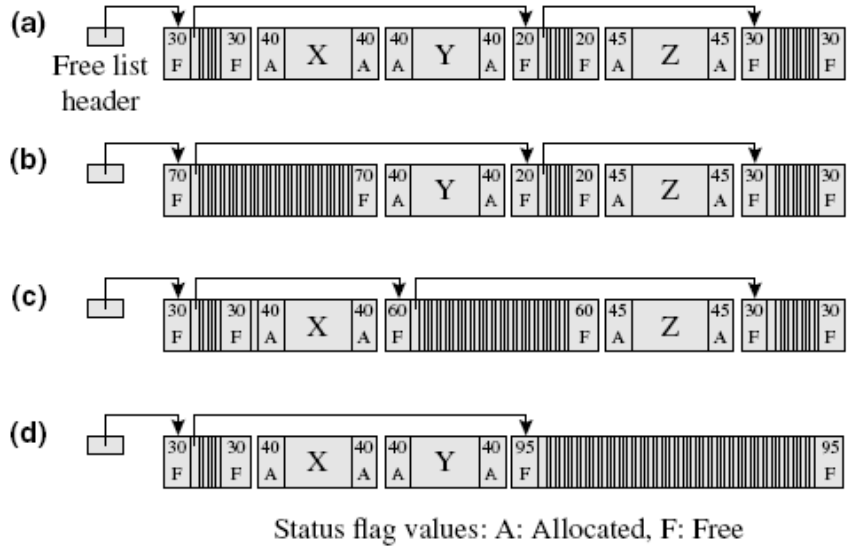


Figure 11.13 Merging using boundary tags: (a) free list; (b)–(d) freeing of areas X, Y, and Z, respectively.

Fusione di aree di memoria libera (cont.)

- Quando è eseguita l'unione, è rispettata la *regola del 50 percento*

Number of allocated areas, $n = \#A + \#B + \#C$
Number of free areas, $m = \frac{1}{2}(2 \times \#A + \#B)$
In the steady state $\#A = \#C$. Hence $m = n/2$

- Utile per stimare la dimensione della free list e l'area di memoria libera totale

Fusione di aree di memoria libera: compattazione

- La compattazione della memoria è ottenuta impacchettando tutte le aree allocate verso un estremità della memoria
  - Possibile solo se è fornito un registro di rilocazione

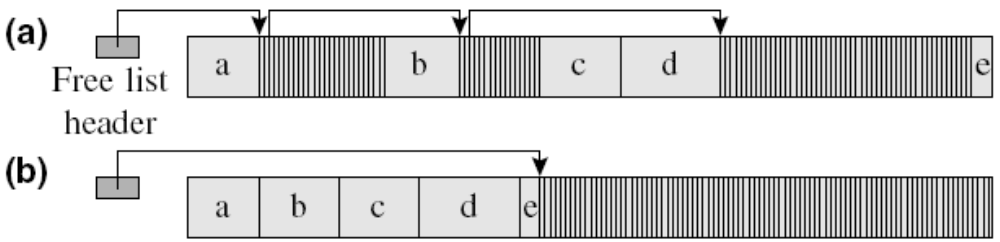


Figure 11.14 Memory compaction.

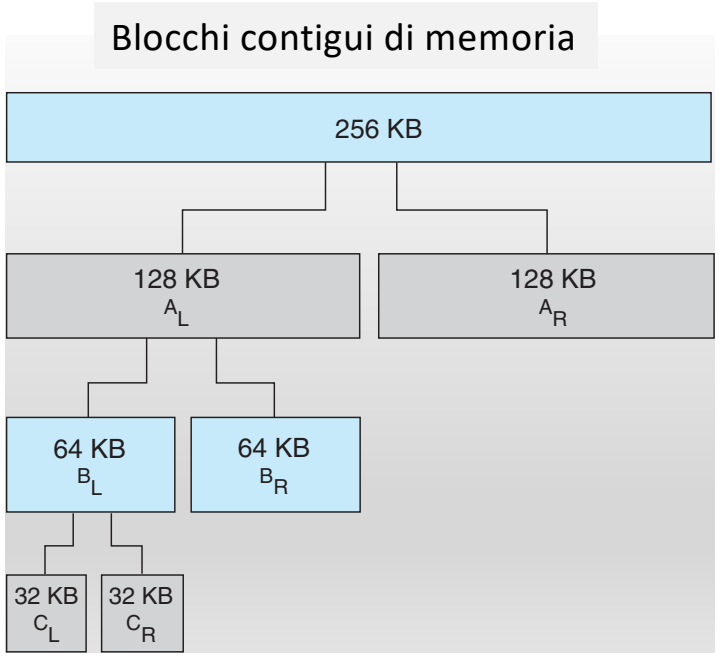
Buddy system e allocatori delle potenze di 2

- Questi allocatori eseguono l'allocazione della memoria in blocchi di poche dimensioni predefinite
  - Porta ad una frammentazione interna
  - Permette all'allocatore di gestire free list separate per blocchi di dimensioni di blocco diverse
    - Evita costose ricerche nella free list
    - Porta ad allocazioni e de allocazioni veloci
- Il buddy system esegue una fusione di blocchi liberati
- L'allocatore delle potenze di 2 non esegue la fusione

Buddy System

- Divide e ricombina blocchi in modo predefinito durante le allocazioni / deallocazioni
- I blocchi generati dalle suddivisioni sono chiamati *blocchi buddy*
- I blocchi buddy liberi sono uniti per formare il blocco da cui si erano originati
  - Operazione definita *coalescenza* (o fusione)
- I blocchi liberi adiacenti che non sono buddy non sono sottoposti a coalescenza
- Ad esempio, un buddy system binario, divide un blocco in due parti (buddy) di pari dimensioni
  - Le dimensioni dei blocchi sono 2^n, per differenti valori di n>=t, con t soglia prefissata
  - Con questo vincolo ci si assicura che i blocchi di memoria non siano *inutilmente* piccoli

Buddy System



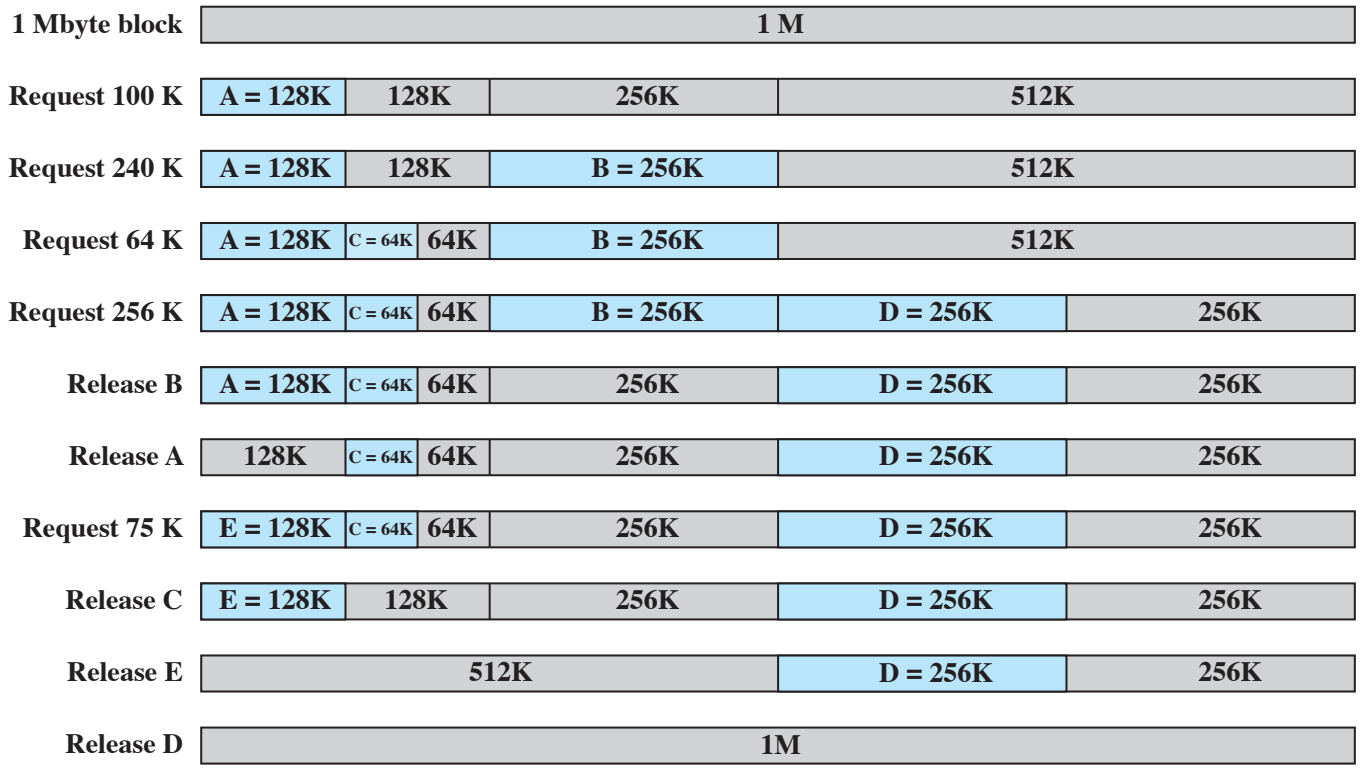
Buddy System (cont.)

- L'allocatore associa un tag di 1 bit ad ogni blocco per indicare se il blocco è libero o allocato
  - Il tag può trovarsi nel blocco o all'esterno
- Sono gestite diverse liste di blocchi liberi
  - Ogni lista è una lista doppiamente linkata
  - Formata da blocchi di stessa dimensione, ad esempio 2^k, per qualche k >= t
- L'allocatore comincia con un singolo blocco libero di dimensioni 2^z, con z > t
  - Messo nella free list di taglia 2^z
- Supponiamo che un processo richieda un'area di memoria di **m** byte
  - Il sistema trova la più piccola potenza di 2 maggiore di m. Sia $m = 2^i$
  - Se la lista di blocchi di dimensione 2^i non è vuota, allora alloca il primo blocco presente nella lista (il tag viene cambiato da libero ad allocato)
  - Se la lista di taglia 2^i è vuota, cerca la lista di blocchi di dimensione 2^{i+1}
    - Preleva un blocco da tale lista e lo divide in due metà di dimensione 2^i
    - Mette uno di tali blocchi nella free list per blocchi di taglia 2^i e usa l'altro per soddisfare la richiesta
    - Se non esiste un blocco di dimensione 2^{i+1} cerca nella lista di blocchi di taglia 2^{i+2} e così via

Buddy System (cont.)

- Quando un processo libera un blocco di memoria di taglia 2^i, il sistema cambia il tag in libero e controlla il tag del suo blocco buddy per vedere se è libero
  - Se si, fonde i due blocchi in un blocco singolo di taglia 2^{i+1}
  - Ripete il controllo di coalescenza iterativamente
    - Controlla se il buddy di questo nuovo blocco di taglia 2^{i+1} è libero, e così via
  - Pone un blocco in una free list solo quando trova che il suo blocco buddy non è libero

Buddy System (cont.)



Allocatore potenze di 2

- La dimensione dei blocchi di memoria sono potenze di 2
- Sono mantenute free list separate per i blocchi di dimensione diversa
- Ogni blocco contiene un header
  - Contiene l'indirizzo di una free list a cui dovrebbe essere aggiunto quando diventa libero
- E' allocato un intero blocco su richiesta
  - Non avviene alcuna suddivisione dei blocchi
- Nessuno sforzo per unire blocchi contigui
  - Quando è rilasciato, un blocco è restituito alla sua free list

Confronto Allocatori di Memoria

- Il confronto avviene in termini di velocità di allocazione ed efficienza d'uso della memoria
- Gli allocatori buddy e potenze di 2 sono più veloci degli allocatori best, next e first fit
  - Non devono cerca nelle free list
- Allocatori best, next e first soffrono di frammentazione esterna
- Allocatori buddy e potenze di 2 solo di frammentazione interna, eventualmente