MEMORY ABSTRACTION

Danilo Croce

Novembre 2024

Parafrasi della legge di Parkinson

Programs expand to fill the memory available to hold them.

My laptop with 32GB of RAM has 20,000 times more memory than the IB M 7094 (32768 words of 36 bits)—the largest computer in the world in the early 1960s. And my laptop is always full.

GESTIONE DELLA MEMORIA NEI SISTEMI OPERATIVI

- **Memoria Principale (RAM)**:
  - Fondamentale, cresce rapidamente, ma **i programmi crescono più velocemente**.
  - **Desiderio**: Memoria privata, grande, veloce, persistente e a basso costo.
  - La realtà tecnologica è diversa.
- **Gerarchia della Memoria**:
  - Concetto sviluppato nel tempo.
  - Da memoria veloce e costosa a memoria lenta, economica e di grandi dimensioni.
    - Es. Da pochi MB di memoria veloce a TB di memoria lenta e dispositivi di archiviazione come USB.
- **Compito del Sistema Operativo**:
  - Astrazione di questa gerarchia in un modello utile e gestione dell'astrazione.
- **Gestore della Memoria**:
  - Gestisce la memoria e la sua gerarchia.
  - Traccia l'uso della memoria, alloca e libera memoria per i processi.
- **Focus**: **memoria principale** (storage di massa/dischi in lezioni successive)

GESTIONE DELLA MEMORIA: OUTLINE

- Memory Abstraction
- Virtual Memory
- Algoritmi di sostituzione delle pagine
- Problemi di Progettazione per Sistemi di Paging

GESTIONE DELLA MEMORIA: OUTLINE

- **Memory Abstraction**
- Virtual Memory
- Algoritmi di sostituzione delle pagine
- Problemi di Progettazione per Sistemi di Paging

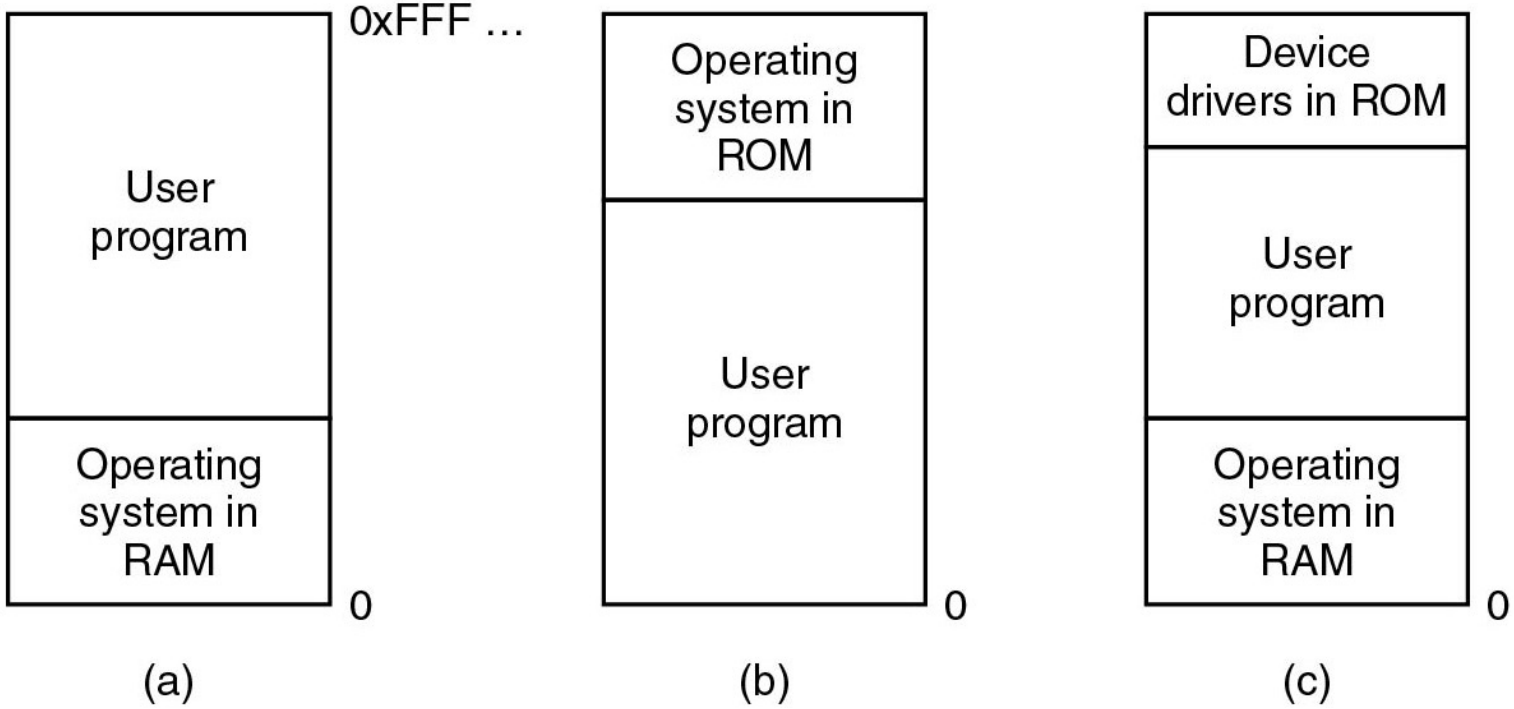
MEMORIA SENZA ASTRAZIONE

- Modello più semplice: uso diretto dalla memora fisica!
- Nei primi computer mainframe, minicomputer e PC, non esisteva astrazione della memoria
  - ogni programma vedeva solo la memoria fisica.
- Ad esempio, con l'istruzione
  - `MOV REGISTER1,1000`

  la locazione fisica di memoria `1000` veniva trasferita in `REGISTER1`.
- Un programma poteva interferire con un altro, causando crash.
  - Esempio/Rischio: un applicativo utente può cancellare il sistema operativo!!!

NESSUNA ASTRAZIONE: MONOPROGRAMMAZIONE

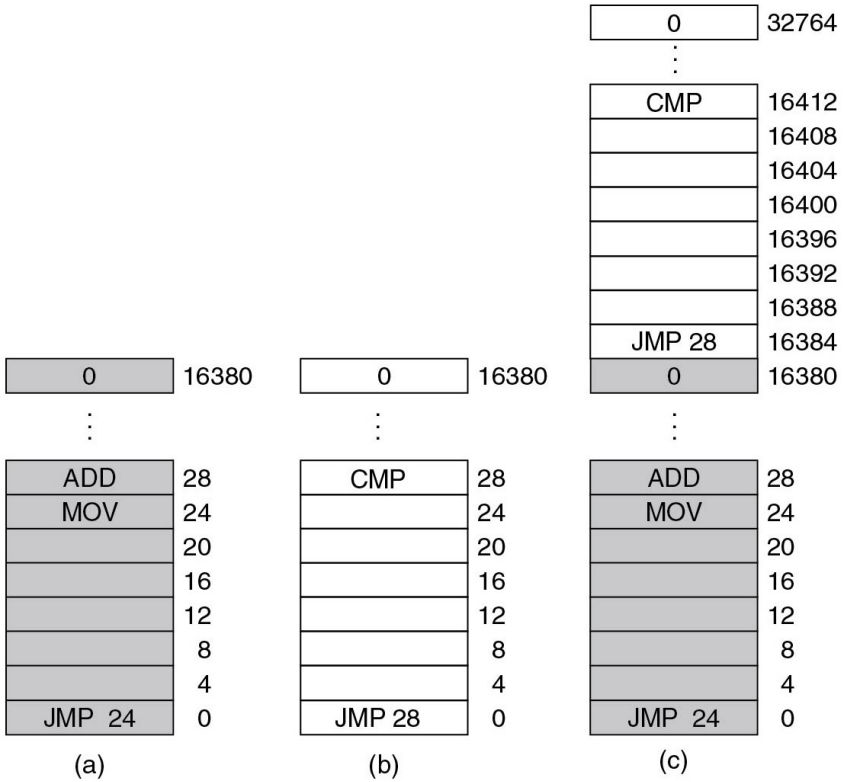
- Tre modelli principali di organizzazione della memoria:
  a) OS in RAM
     - utilizzato sui mainframe e sui minicomputer (desueto)
  b) OS in ROM
     - Es: sistemi embedded
  c) OS+drivers in ROM+RAM
     - primi personal computer (es. MS-DOS: la ROM era chiamata Basic Input Output System - BIOS)



NESSUNA ASTRAZIONE: MULTIPROGRAMMAZIONE

- Possibilità di eseguire più programmi contemporaneamente senza astrazione della memoria usando lo "**swapping**".
  - **Swapping**: salvataggio del contenuto **della memoria in un file su memoria non volatile** e prelievo del programma successivo.

NESSUNA ASTRAZIONE: MULTIPROGRAMMAZIONE



- **Naive Approach**: Caricamento di più programmi in memoria fisica consecutivamente, senza astrazione dell'indirizzo.
  a) Un programma di 16 KB che inizia con l'istruzione `JMP 24`.
  b) Un altro programma di 16 KB che inizia con l'istruzione `JMP 28`.
  c) Entrambi i programmi caricati consecutivamente.
    - Quando il secondo programma viene eseguito, `JMP 28` **indirizza erroneamente all'istruzione del primo programma**, causando errori.
- **Problema Principale**:
  - I programmi utilizzano indirizzi assoluti di memoria fisica, portando a conflitti durante l'esecuzione.
  - La mancanza di astrazione dell'indirizzo può causare il crash dei programmi.

ASTRAZIONE DELLA MEMORIA

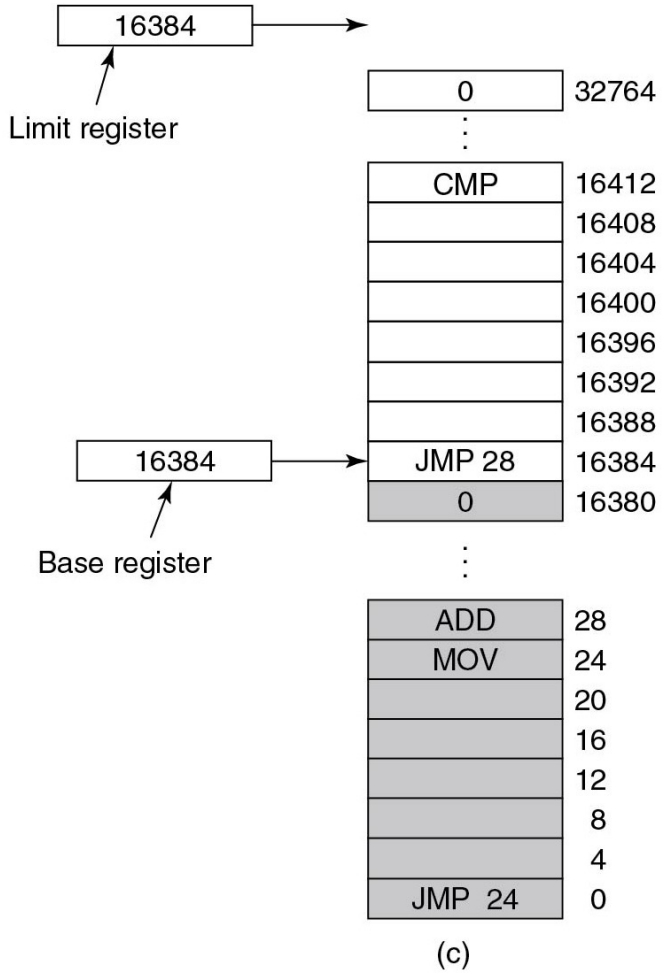
ASTRAZIONE DELLA MEMORIA E SPAZI DEGLI INDIRIZZI

- **Problema**: L'accesso diretto alla memoria fisica da parte dei programmi può causare problemi come la distruzione del sistema operativo e la difficoltà di esecuzione simultanea di più programmi.
- **Soluzione**: Astrazione della memoria per separare e proteggere i programmi in esecuzione.
- **Concetto di Spazio degli Indirizzi**:
  - Ogni programma ha **un insieme unico di indirizzi** (spazio degli indirizzi) che può usare per indirizzare la memoria.
  - Questo spazio è indipendente da altri processi e rappresenta una forma di memoria astratta.

IMPLEMENTAZIONE CON REGISTRI BASE E LIMITE

- **Vecchia soluzione**: mappare lo spazio degli indirizzi di ogni processo in parti diverse della memoria fisica.
- **Registri Base e Limite:** Due registri hardware speciali presenti in molte CPU.
  - **Registro Base**: contiene l'indirizzo fisico di inizio di un programma in memoria.
  - **Registro Limite**: contiene la lunghezza del programma.
- Gli indirizzi generati dai programmi vengono aggiustati automaticamente aggiungendo il valore del registro base.

ADDRESS SPACES



- Il registro di base mette in atto la rilocazione dinamica
- Il registro limite applica la protezione

Checks for `MOV Reg1, Addr`**:**

IF(Addr > LIMIT) → **NOT OKAY**
IF(BASE + Addr < BASE) → **NOT OKAY**

Base and limit registers

FUNZIONAMENTO E LIMITI DEI REGISTRI

- Ogni riferimento alla memoria da parte di un programma:
  - **Aggiunge il valore del registro base** all'indirizzo generato.
  - **Confronta con il registro limite** per assicurare che l'accesso sia entro i limiti consentiti.
- **Vantaggi**: Offre a ogni processo uno **spazio degli indirizzi separato e protetto**.
- **Svantaggi**: Necessità di eseguire somme e confronti ad ogni accesso alla memoria, il che può essere lento.

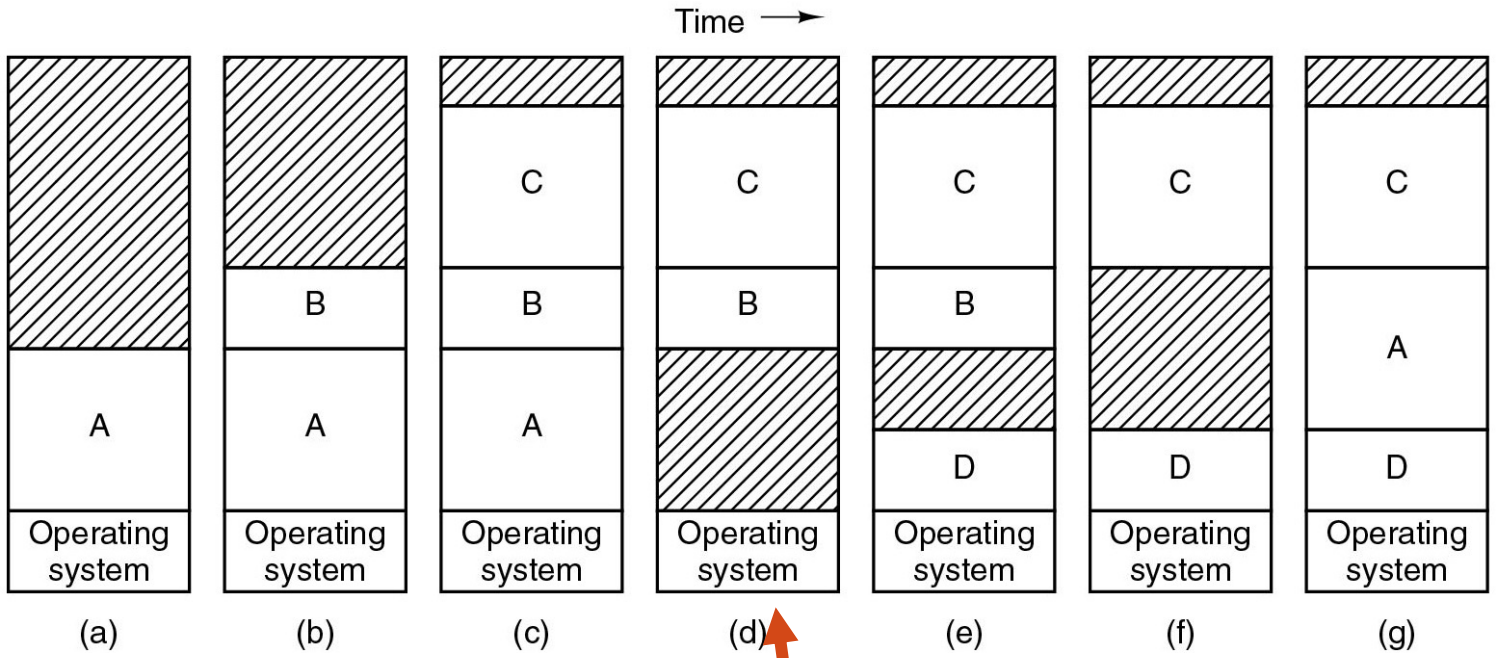
WHAT IF WE RUN OUT OF MEMORY?

- Un computer medio al suo avvio può avere 50-100 processi o più.
  - Applicazioni come Photoshop possono richiedere fino a 1 GB solo per avviarsi.
  - La memoria fisica totale necessaria è spesso superiore a quella disponibile.
- **Strategie per gestire il sovraccarico di memoria**

1. **Swapping (Scambio) dei processi**:
   - Sposta interi processi tra la memoria RAM e la memoria non volatile (disco/SSD).
   - Processi inattivi archiviati su memoria non volatile (vedi slide successiva)
2. **Memoria Virtuale**:
   - Permette l'esecuzione dei programmi anche se **solo parzialmente presenti nella memoria principale** (prossime lezioni).

IMPROVEMENT: DYNAMIC PARTITIONS AND SWAPPING

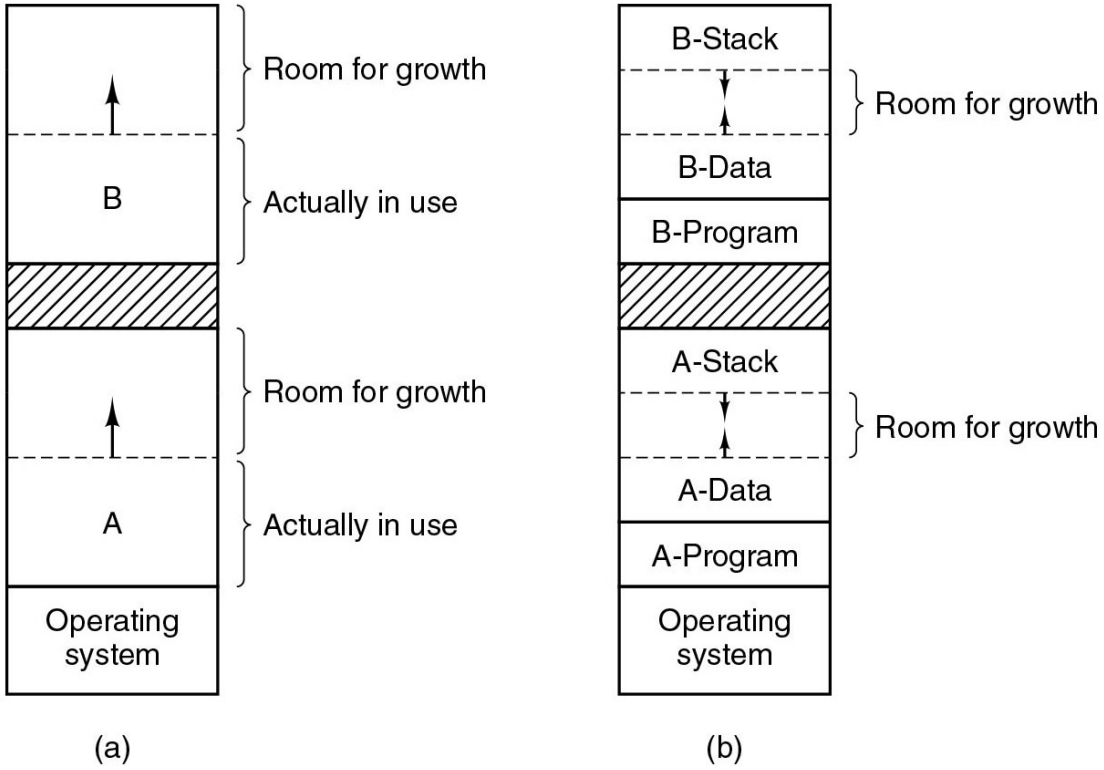
- Lo scambio può portare alla **frammentazione** della memoria
- Necessaria compattazione della memoria
  - Estremamente lenta



Necessario allocare D spostando A su memora non volatile

GESTIONE DELLO SPAZIO E CRESCITA DEI PROCESSI

- **Sfida**: gestire processi con segmenti di dati in crescita.
- **Memory Compaction**: sposta processi per liberare spazio, ma richiede tempo.
- **Soluzione**: allocare memoria extra durante lo swapping o lo spostamento dei processi.
- Cosa fare in caso di Out of Memory?
  - «Uccidere» il processo
  - Trasferire il processo
  - Swapping



(a) Spazio allocato per segmento dati in crescita.
(b) Spazio per stack e segmento dati che crescono.

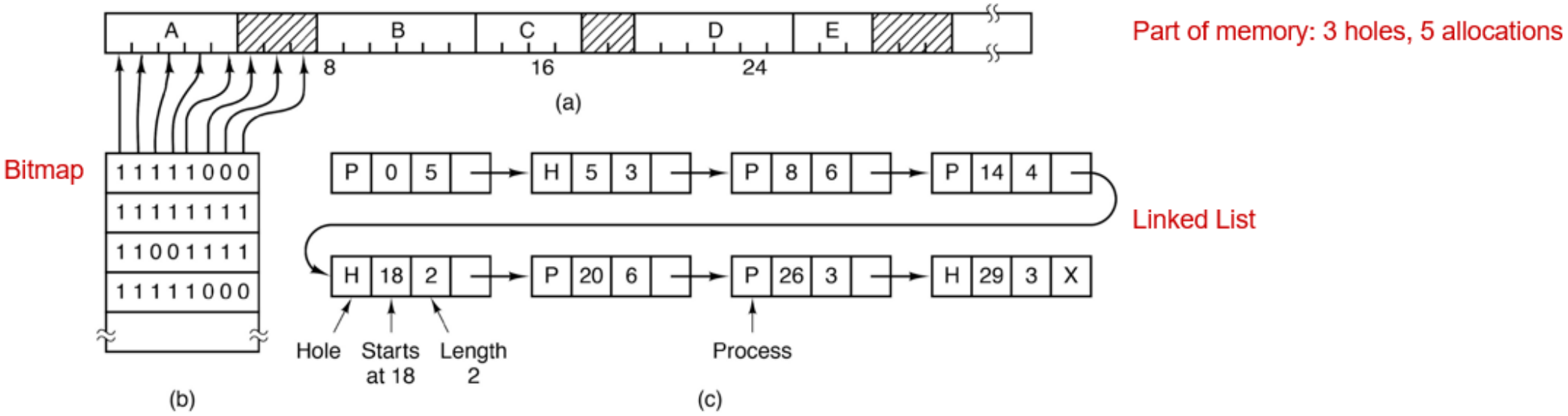
GESTIONE DELLA MEMORIA LIBERA

GESTIONE DINAMICA DELLA MEMORIA

- **Obiettivo**: Tenere traccia dell'utilizzo della memoria.
  - Ad esempio ogni blocco di 4 bytes
- **Metodi principali**:
  - **Soluzione 1: Bitmap** tiene traccia di quali blocchi vengono allocati
  - **Soluzione 2:** Una **lista** collegato tiene traccia della memoria non allocata
- **Importanza**: Questo tracciamento non riguarda solo la memoria, ma anche risorse come i file system.

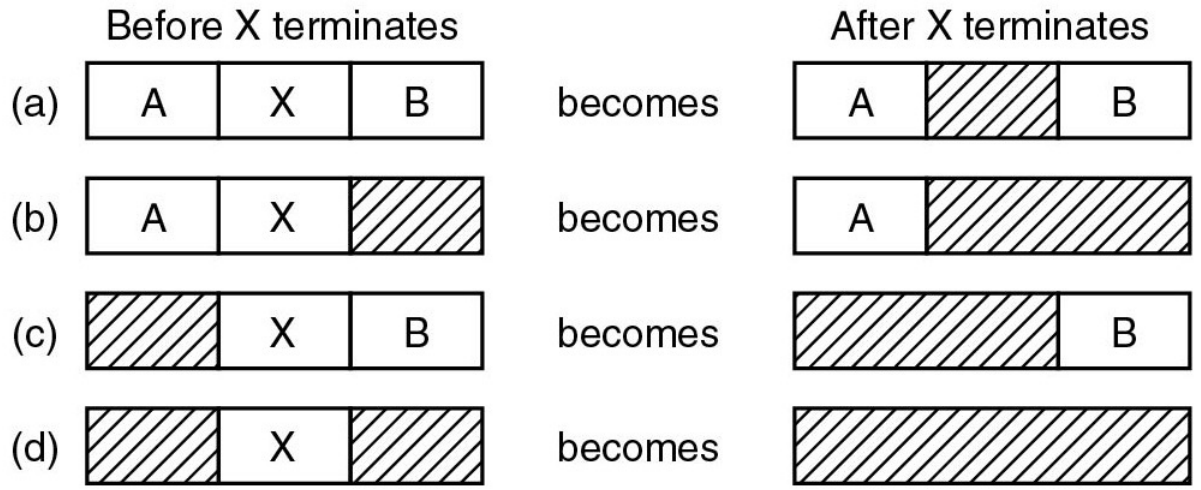
MEMORY MANAGEMENT: BITMAPS VS LINKED LISTS

- **Bitmap**: trovare i fori richiede una scansione (lenta)
- **Lista**: liste di processi/«buchi»
  - Allocazione lenta contro deallocazione lenta
  - Buchi ordinati per indirizzo per una rapida coalescenza (come fusione tra gocce)



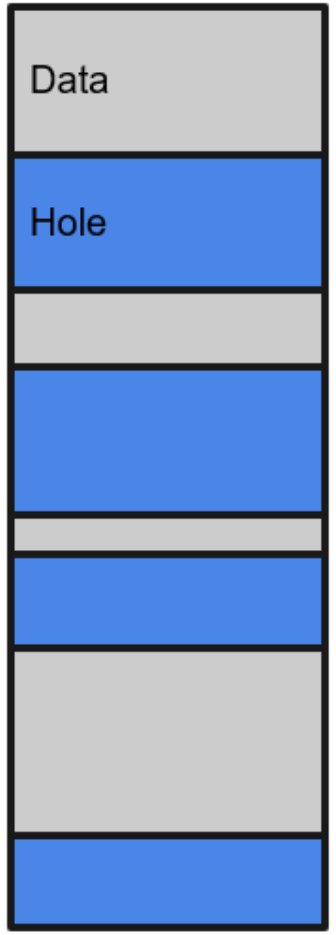
MEMORY MANAGEMENT E LINKED LISTS

- Nella pratica **viene spesso usata una doppia linked list**
  - Rende **più facile gestire lo spazio libero**
    - Può controllare facilmente se il precedente spazio è libero
    - Può regolare facilmente i puntatori



SCHEMI DI ALLOCAZIONE DELLA MEMORIA

- **First Fit**: Seleziona il primo spazio disponibile.
  - Opzione più semplice (MINIX3).
- **Next Fit**: Seleziona il successivo spazio disponibile.
  - Più lento del First Fit in pratica.
- **Best Fit**: Sceglie lo spazio più adeguato.
  - Tende alla frammentazione.
- **Worst Fit**: Sceglie lo spazio meno adeguato.
  - Prestazioni scadenti in pratica.
- **Quick Fit**: Mantiene spazi di dimensioni diverse (le più richieste).
  - Scarsa performance nella coalescenza.
- **Buddy Allocation** (Linux): Migliora la performance di coalescenza del Quick Fit.

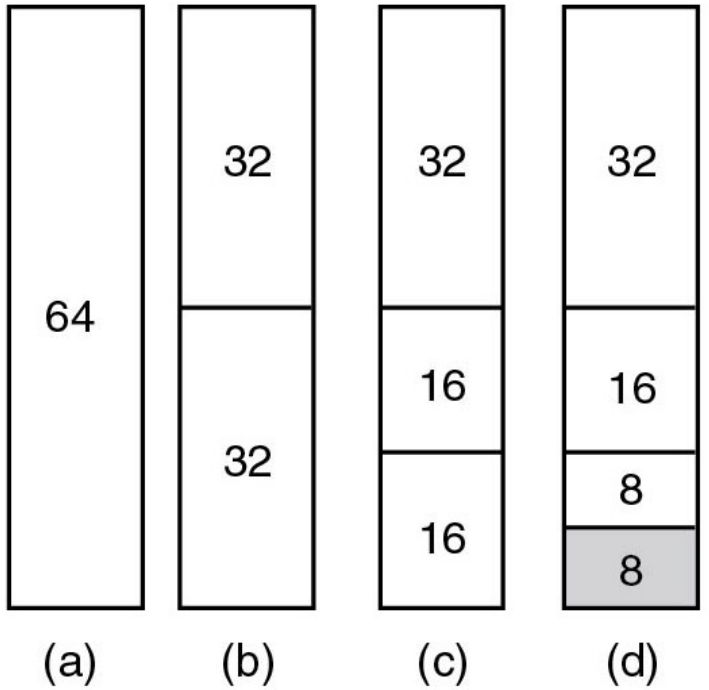


ALLOCAZIONE DELLA MEMORIA IN LINUX (CAP 10.4 DEL LIBRO)

- Linux utilizza vari meccanismi per l'allocazione della memoria.
- Il principale è una allocazione delle pagine basato sull'algoritmo di **Buddy Memory Allocation**.
- **Funzionamento**
  - La memoria inizia come un singolo pezzo contiguo.
  - Ad ogni richiesta, la memoria viene divisa secondo una potenza di 2.
  - Blocchi di memoria contigui vengono uniti quando rilasciati (coalescenza)

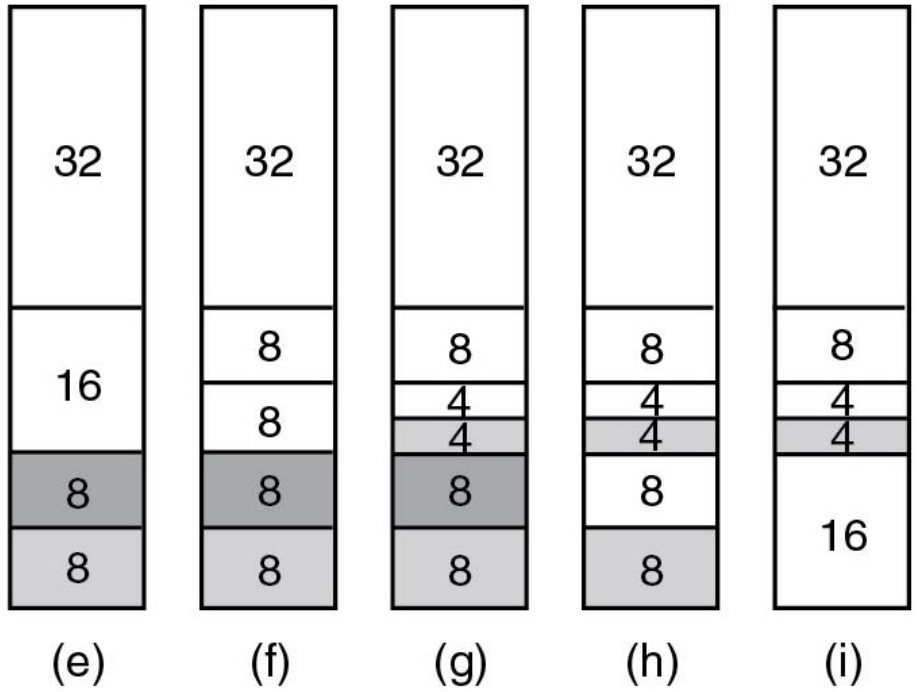
BUDDY MEMORY ALLOCATION UN ESEMPIO

- Inizialmente la memoria consiste di un singolo pezzo contiguo,
  - 64 pagine nel semplice esempio (a).
- Arriva una richiesta da 8 otto pagine
  - Arrotondata a potenza di 2
- L'intero pezzo di memoria viene quindi diviso a metà, come mostrato in (b)
- Poiché ciascuno dei due pezzi è ancora troppo grande, il pezzo più in basso viene diviso ancora a metà (c) e poi ancora (d).
- Ora abbiamo un pezzo della dimensione corretta, che viene così allocato al chiamante, grigio in (d).



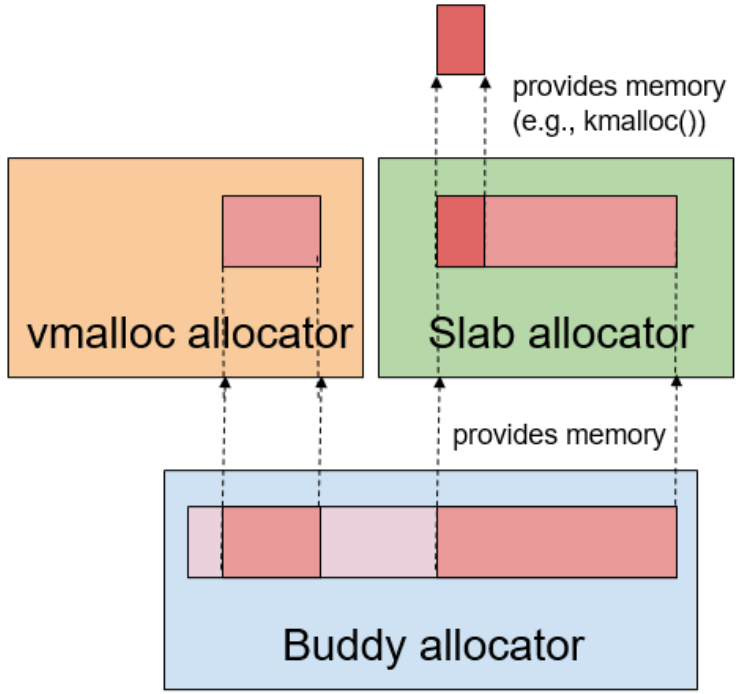
BUDDY MEMORY ALLOCATION UN ESEMPIO (2)

- Arriva una seconda richiesta di otto pagine; questa può essere soddisfatta immediatamente (e).
- A questo punto arriva una terza richiesta di quattro pagine.
- Il pezzo disponibile più piccolo viene diviso (f) e ne viene assegnata la metà (g).
- Successivamente, il secondo pezzo di otto pagine viene rilasciato (h).
- Infine, anche l'altro pezzo di otto pagine viene rilasciato. Poiché i due pezzi di otto pagine adiacenti appena liberati provengono dallo stesso pezzo di 16 pagine, **vengono uniti** per riottenere il pezzo di 16 pagine (i).



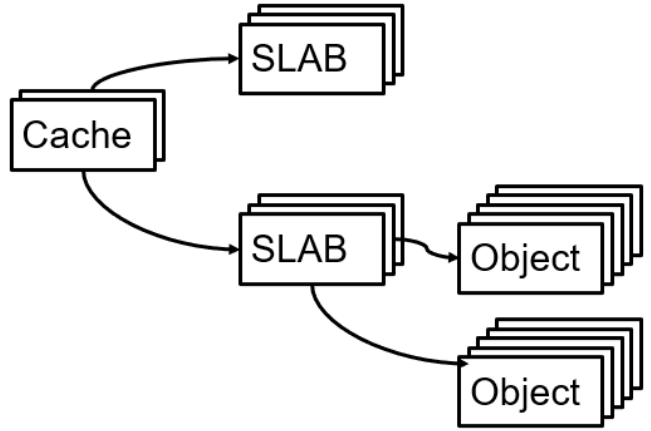
FRAMMENTAZIONE E SLAB ALLOCATOR

- Il **Buddy Algorithm può causare frammentazione** interna:
  - una richiesta di 65 pagine richiede l'allocazione di 128 pagine.
- Lo **SLAB allocator** in Linux risolve questo problema,
  - prendendo blocchi tramite l'algoritmo buddy e
  - ritagliando unità più piccole (**slab**) per gestirle separatamente.



SLAB ALLOCATOR

- **Principio di Base**
  - Il kernel spesso ha bisogno di creare e distruggere piccoli oggetti di dimensioni e tipi specifici.
  - Senza ottimizzazione, questa operazione potrebbe portare a una significativa frammentazione della memoria.
- **Come Funziona:**
  - Nello slab allocation, la **memoria è divisa in blocchi chiamati "slabs"**
    - ulteriormente suddivisi in chunk di dimensioni uniformi,
    - adeguati per ospitare un oggetto di un certo tipo.
  - Un "slab" può essere in uno dei seguenti stati:
    - pieno (tutti i chunk sono utilizzati),
    - parzialmente pieno (alcuni chunk sono liberi)
    - o vuoto (tutti i chunk sono liberi).



SLAB E CACHING

- Quando un oggetto viene deallocato, non viene immediatamente restituito al sistema come memoria libera.
  - viene mantenuto nella cache in modo che, se viene richiesta un'altra istanza dello stesso tipo di oggetto, possa essere rapidamente riallocata senza l'overhead di inizializzazione.

- In questo esempio, uno Slab contiene:
  - puntatore all'inizio della memoria con gli slot degli oggetti
  - indice del prossimo slot libero
  - `bufctl`: array di indici dei prossimi oggetti liberi
  - Slot per gli oggetti