

GESTIONE DELLA MEMORIA

- La memoria principale (RAM) è una risorsa importante e delicata che deve essere gestita con molta attenzione.
- Il concetto di **gerarchia di memoria**:
 - I computer attuali possono avere pochi MB di memoria **cache** volatile, costosa e molto veloce, qualche GB di **memoria principale**, di media velocità e prezzo, e qualche TB di **storage** su disco non volatile, economico e lento.
- La parte del sistema operativo che gestisce (in parte) la gerarchia di memoria viene chiamato **gestore della memoria**.

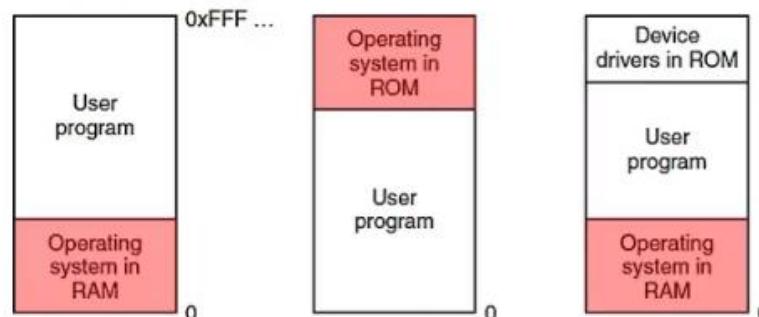
NESSUNA ASTRAZIONE DI MEMORIA

- Il modo più semplice per gestire la memoria è senza forme di astrazione: i processi la «vedono» direttamente così com'è costituita (numero di celle e dimensione del contenuto).
- I primi mainframe (prima del 1960), minicomputer (prima del 1970) e PC (prima del 1980) non avevano alcuna astrazione di memoria.
- Per eseguire un programma è necessario caricarlo nella memoria principale.
- Più programmi possono stare in memoria ma è necessario che non si sovrappongano!



NESSUNA ASTRAZIONE DI MEMORIA

- Anche senza astrazione di memoria il modello fisico può essere organizzato in vari modi:
 - SO in fondo alla memoria nella RAM (mainframe e minicomputer).
 - SO in testa alla memoria nella ROM (palmari e sistemi embedded).
 - SO in fondo alla memoria nella RAM, driver delle periferiche in testa alla memoria nella ROM (PC con MS-DOS).



- Il primo e il terzo modello hanno lo svantaggio che un errore nel programma utente può eliminare il sistema operativo!



Esecuzione di più programmi

- Anche senza l'astrazione di memoria, è possibile fornire all'utente l'idea di avere in esecuzione più programmi «contemporaneamente».
- Il sistema operativo salva l'intero contenuto della memoria in un file sul disco, carica ed esegue il programma successivo (**swaping**).
- Con l'aggiunta di alcuni hardware speciali, è possibile eseguire più programmi contemporaneamente, anche senza swapping.
- La mancanza di una astrazione memoria è ancora comune nei dispositivi quali radio, lavatrici e forni a microonde:
 - il software indirizza la memoria in modo assoluto e funziona solo perché i programmi sono noti in anticipo e gli utenti non sono (ancora) liberi di eseguire il proprio software sul proprio tostapane.



UN'ASTRAZIONE DI MEMORIA: LO SPAZIO DI INDIRIZZAMENTO

- Esporre la memoria fisica ai processi può presentare delle criticità:

1

se i programmi utente possono indirizzare ogni byte della memoria, possono facilmente sovrascrivere il sistema operativo.

2

è complicato avere un'organizzazione rigida della memoria che consenta a più programmi di essere in esecuzione contemporaneamente.



La nozione di spazio degli indirizzi

- Per consentire a più applicazioni di essere in memoria allo stesso tempo senza interferire devono essere risolti due problemi: **protezione** e **riposizionamento**.
- Una soluzione iniziale fu adottata dall'IBM 360 che etichettava pezzi di memoria con una chiave di protezione e, poi, confrontava la chiave del processo di esecuzione con quella di ogni parola di memoria:
 - Questo approccio non risolveva il secondo problema del posizionamento dei programmi, anche se può essere risolto spostandoli al momento del caricamento con aggravio dei tempi e della complessità.



La nozione di spazio degli indirizzi

- Una soluzione migliore è quella di inventare una nuova astrazione per la memoria: lo **spazio degli indirizzi**.
- Poiché il concetto di processo crea una sorta di CPU astratta per eseguire i programmi, lo spazio di indirizzamento crea una sorta di **memoria astratta** per farci vivere i programmi.
- Uno **spazio di indirizzi** è l'insieme di indirizzi che un processo può utilizzare per accedere alla memoria.
- Ogni processo ha un proprio spazio di indirizzamento, indipendente da quello di altri processi.
- Il concetto di uno spazio di indirizzi è molto generale e si verifica in molti contesti (es. numerazioni telefoniche, porte I/O, indirizzi internet IPv4,...).



Registro base e limite

- Questa soluzione utilizza una versione particolarmente semplice di rilocazione dinamica: mappa ogni spazio indirizzi del processo su una diversa parte della memoria fisica.
- La soluzione classica è quella di dotare ogni CPU con due registri hardware speciali: **registro base** e **registro limite**.
- I programmi sono caricati in locazioni di memoria consecutive.
- Quando viene eseguito un processo, il registro base viene caricato con l'indirizzo fisico di memoria dove inizia il programma e il registro limite con la lunghezza del programma.
- Ogni volta che un processo accede alla memoria l'hardware della CPU aggiunge automaticamente il valore base all'indirizzo verificando se l'indirizzo calcolato non eccede il limite di memoria riservato al processo (registro limite).
- In molte implementazioni, i registri base e limite sono protetti (solo il sistema operativo li può modificare).



Registro base e limite

- Uno svantaggio della rilocazione per mezzo dei registri base e limite è la necessità di eseguire un'addizione e un confronto ad ogni riferimento di memoria.
- Il confronto può essere fatto in fretta, ma le addizioni sono lente (a causa del tempo di propagazione dei riporti) se non si utilizzano circuiti sommatori speciali.



Swapping

- Se la memoria fisica del computer è abbastanza grande da contenere tutti i processi, gli schemi descritti sono validi.
- Normalmente, la RAM non riesce a contenere tutti i processi e per gestire la memoria sono possibili due approcci generali:

1

Swapping: si prende un processo e lo si esegue per un certo tempo, quando ha terminato si salva sul disco. I processi inattivi sono memorizzati su disco, in modo da non occupare la memoria quando non sono in esecuzione.

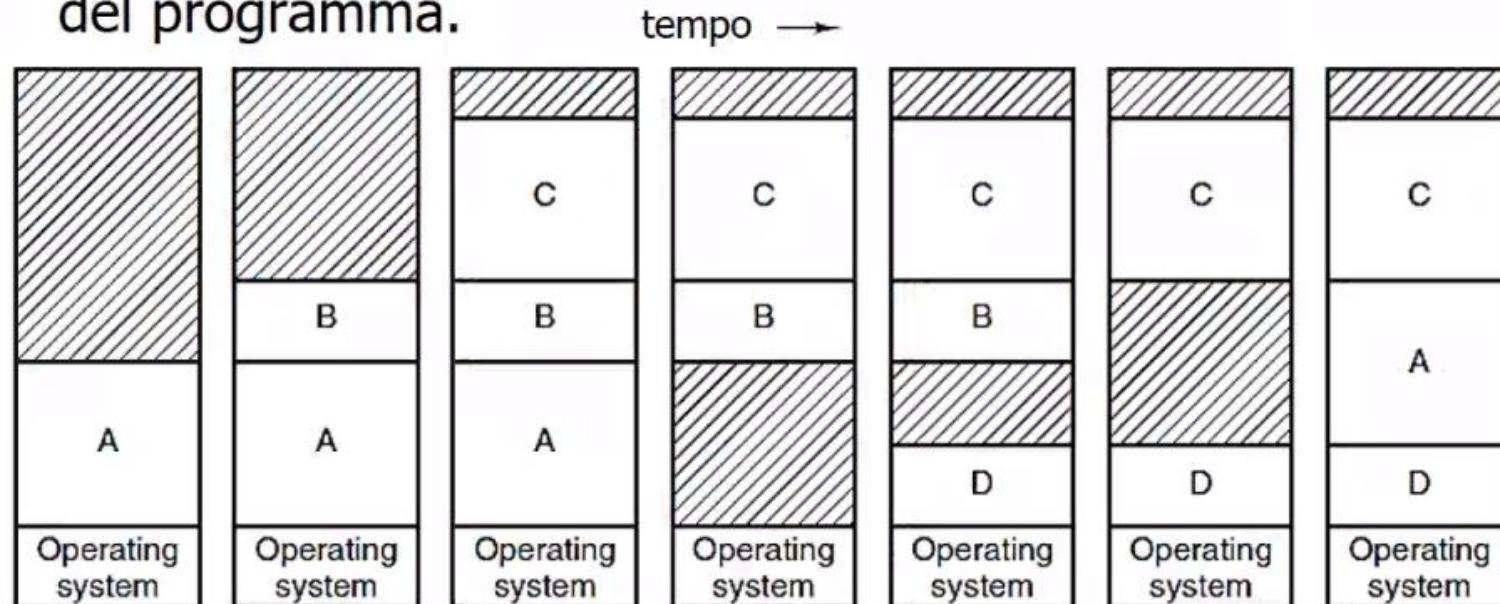
2

Memoria virtuale: consente ai programmi di poter essere eseguiti anche quando sono parzialmente nella memoria principale.



Swapping

- Inizialmente il processo A è in memoria, sono poi creati (o caricati) i processi B e C. Dopo A è salvato sul disco.
- Viene creato D e B termina. Finalmente viene ricaricato A ma in una posizione diversa: i suoi indirizzi devono essere rilocati dal software o dall'hardware durante l'esecuzione del programma.



Swapping

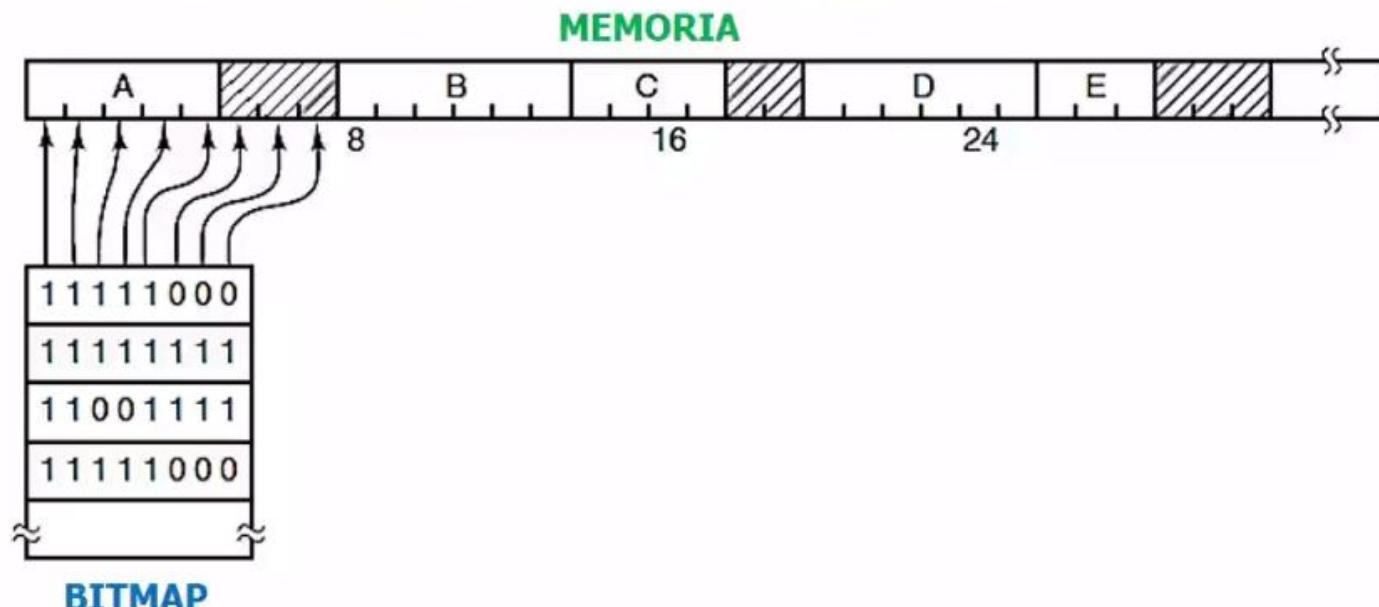
- Lo swapping crea dei buchi nella memoria che è possibile compattare solo spostando i processi, raramente è eseguita poiché si tratta di una operazione onerosa per la CPU.
 - Se i processi hanno una dimensione fissa, l'allocazione è semplice: il sistema operativo alloca esattamente ciò che è necessario.
 - Normalmente i processi crescono di dimensione, attraverso l'allocazione dinamica della memoria heap o lo stack, quindi è buona norma riservare uno spazio in più riservato alla crescita.
-

Gestione della memoria libera

- Quando la memoria viene assegnata dinamicamente, il sistema operativo deve gestirla.
- Ci sono due modi per tenere traccia dell'utilizzo della memoria:
 - Bitmap.
 - Liste collegate di allocazione.

Gestione della memoria con bitmap

- Con una bitmap, la memoria è divisa in unità di allocazione.
- La dimensione di una unità di allocazione può essere di qualche parola fino a diversi KB.
- La bitmap ha bit quante le unità di allocazione della memoria:
 - se il bit è 0 l'unità è libera, se 1 è occupata.



Gestione della memoria con bitmap

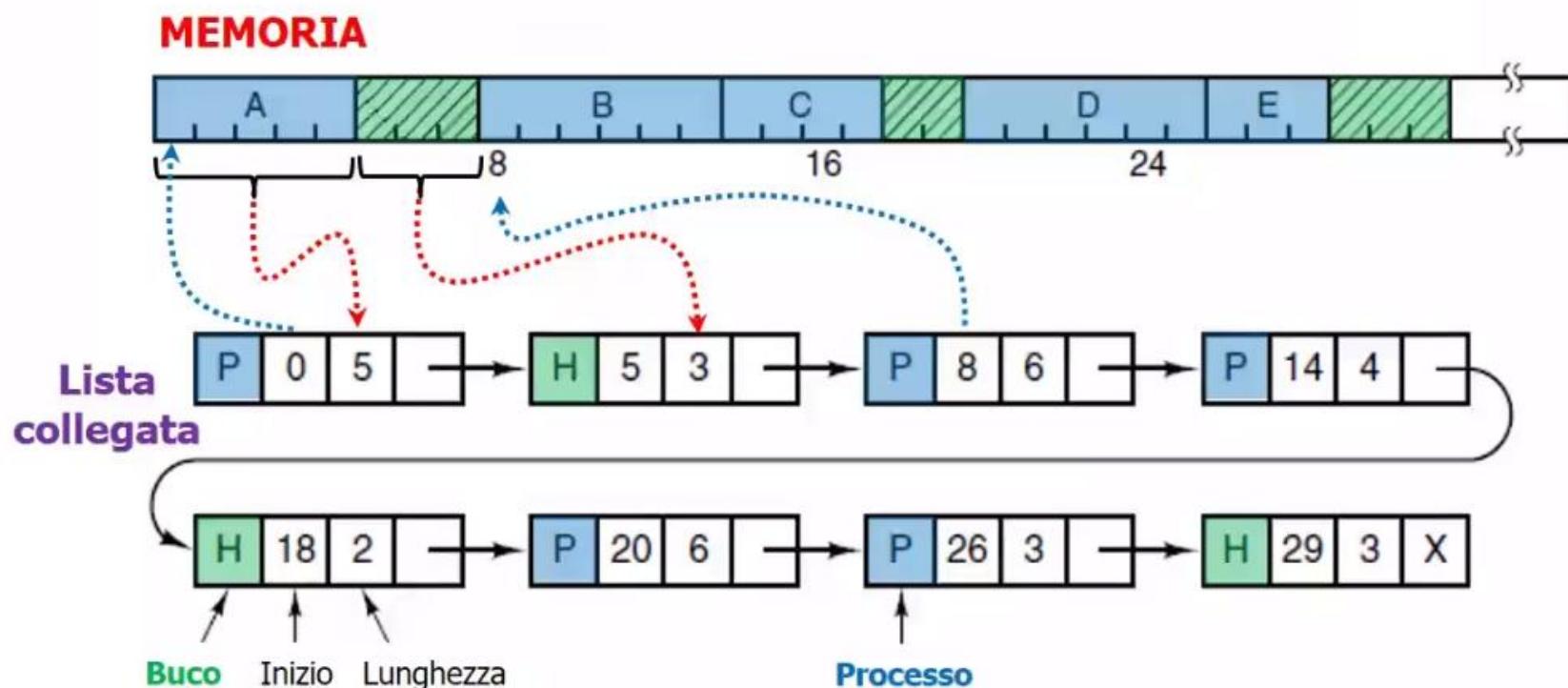
- La dimensione dell'unità di allocazione gioca un ruolo chiave nella progettazione dei sistemi di gestione della memoria: più piccola è l'unità di allocazione maggiore è la dimensione della bitmap.
- Un'unità di allocazione di 4 byte (= 32 bit) richiede solo 1 bit della mappa.
- Una memoria di 32^n bit che utilizza 32 bit per unità di allocazione richiederà 32^{n-1} bit per la mappa, cioè 1/32 (~3%) dello spazio della memoria.

Gestione della memoria con bitmap

- Se si sceglie l'unità di allocazione grande, la bitmap sarà più piccola, ma si correrà il rischio di sprecare memoria nell'unità se la dimensione del processo non è esattamente multiplo di quella dell'unità di allocazione.
- Il problema principale di questa soluzione è che quando dobbiamo portare in memoria un processo di **k** unità, il gestore deve ricercare nella bitmap **k** bit liberi consecutivi (la ricerca sequenziale è un'operazione lenta).

Gestione della memoria con liste collegate

- Un altro modo per gestire la memoria è attraverso una lista collegata che memorizzi le posizioni dei segmenti occupati dai processi (P) e quelle dei segmenti liberi (H).



Gestione della memoria con liste collegate

- Nell'esempio precedente, la lista collegata è ordinata per indirizzo.
- Il vantaggio è che, quando un processo termina o viene fatto lo swap su disco, aggiornando della lista è facile.
- Quando viene creato un nuovo processo o viene caricato dal disco possono essere utilizzati vari algoritmi per trovargli una posizione in memoria:
 - **First fit.**
 - **Next fit.**
 - **Best fit.**
 - **Worst fit.**
 - **Quick fit.**
- Si assume che il gestore della memoria conosca quanta memoria occorre allocare.

Gestione della memoria con liste collegate: first fit

- **first fit** è il più semplice algoritmo.
- Il gestore della memoria scorre la lista finché non trova uno spazio libero abbastanza grande da contenere il processo:
 - Se lo spazio libero è maggiore dello spazio richiesto viene suddiviso in due parti: una per il processo e uno per la memoria non utilizzata.
 - Altrimenti si usa tutto lo spazio.
- **first fit** è un algoritmo veloce perché cerca di non scorrere la lista: il primo spazio libero è quello buono.

Gestione della memoria con liste collegate: next fit

- Una piccola variante del **first fit** è il **next fit**.
- Funziona allo stesso modo del **first fit**, ma tiene traccia della posizione dove ha trovato l'ultimo spazio libero.
- La prossima volta che è chiamato a cercare uno spazio libero, avvia la ricerca dalla posizione dove si è interrotta l'ultima volta, invece di tornare sempre all'inizio come fa il **first fit**.
- Le simulazioni mostrano che il **next fit** dà performance leggermente peggiori di **first fit**.

Gestione della memoria con liste collegate: best fit

- **Best fit** scorre tutta la lista alla ricerca del più piccolo spazio che riesce a contenere il processo richiesto.
- L'idea è che in questo modo si cerca di minimizzare lo spreco di spazio con quello che si adatta meglio alla dimensione cercata.
- **Best fit** è più lento degli altri due perché deve scorrere tutta la lista ogni volta che viene chiamato.
- Un pò sorprendentemente tende a riempire la memoria con piccoli spazi vuoti inutilizzabili e quindi spreca più memoria di **first** e **next fit**.
- **first fit** genera mediamente buchi più grandi.

Gestione della memoria con liste collegate: **worst fit**

- **Worst fit** funziona con logica inversa a **best fit**, cerca di prendere il più grande buco disponibile, in modo che il nuovo spazio vuoto sarà il più grande possibile.
- Le simulazioni dimostrano che anche **worst fit** non è un buon algoritmo di allocazione della memoria.

Ottimizzazione della gestione della memoria

- Tutti e quattro algoritmi possono essere accelerati mantenendo elenchi separati per i processi e spazi vuoti.
- In questo modo, tutti dedicano la loro energia nella ricerca di spazi vuoti, ma il prezzo inevitabile da pagare per questo incremento di velocità è in termini di complessità:
 - quando si deve deallocare la memoria il segmento del processo deve essere rimosso dall'elenco dei processi e deve essere inserito nella lista degli spazi vuoti.
- Con liste separate, si può ordinare per dimensione la lista degli spazi vuoti, questo facilita la ricerca nel **best** e **worst fit**.

Gestione della memoria con liste collegate: quick fit

- **Quick fit** mantiene elenchi separati per alcuni dei formati più comunemente richiesti: una lista per la lista degli spazi da 4KB, una per quelli da 8KB, e così via.
- Trovare uno spazio libero della dimensione desiderata è estremamente veloce.
- Ha lo stesso svantaggio di tutti gli schemi ordinati per dimensione, quando un processo termina o viene inviato sul disco, trovare i suoi vicini per unire gli spazi vuoti è un processo dispendioso.
- Se non sono fusi insieme gli spazi liberi la memoria si frammenta rapidamente in un gran numero di piccoli buchi in cui non entra nessun processo.

MEMORIA VIRTUALE

- I registri base e limite possono essere utilizzati per creare l'astrazione dello spazio di indirizzamento.
- C'è un altro problema che deve essere risolto: la gestione del **software bloat** (o **bloatware**).
- La dimensione del software aumenta molto più velocemente delle dimensioni della memoria.
- Il problema che i programmi sono sempre più grandi della dimensione della memoria è presente fin dalle origini dell'informatica.
- Lo swapping non è un'opzione interessante poiché i dispositivi di I/O sono troppo lenti rispetto alla velocità della memoria.

Overlay

- Una soluzione adottata nel 1960 è stato quello di dividere i programmi in piccoli pezzi, chiamati **overlay**.
- All'avvio del programma veniva caricato in memoria il **gestore degli overlay** che caricava ed avviava l'overlay 0. Gli **overlay** erano mantenuti sul disco e caricati in memoria all'occorrenza dal gestore degli overlay.
- Il lavoro di suddividere il programma in pezzi veniva fatto manualmente dal programmatore.
- Dividere programmi di grandi dimensioni in piccoli pezzi modulari richiedeva molto tempo ed era noioso e soggetto a errori.
- Pochi i programmatori erano in grado di farlo.
- Il computer invece è in grado di fare un lavoro ripetitivo in modo più rapido, senza errori indipendentemente dalla dimensione del programma....

Paging

- L'idea alla base della memoria virtuale è che ogni programma ha uno spazio di indirizzi che consiste di **pagine**.
- Ogni pagina è un intervallo contiguo di indirizzi ed è mappato sulla memoria fisica.
- Per eseguire un programma non è necessario mantenere tutte le pagine di memoria fisica.
- Quando il programma fa riferimento a una parte del suo spazio di indirizzi che è:
 - nella memoria fisica, l'hardware esegue la mappatura necessaria **on-the-fly**;
 - non nella memoria fisica, il sistema operativo avvisa che manca quel pezzo e poi ri-eseguire l'istruzione che ha fallito.

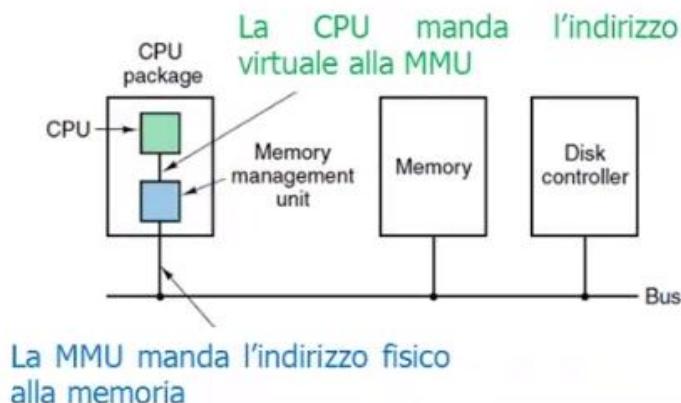
Paging come estensione del concetto di registro base e registro limite

- La memoria virtuale è una generalizzazione dell'idea del registro base e limite.
- Già nel 8088 ci sono registri base separati (ma non aveva i registri limite) per il testo e per i dati.
- Con la memoria virtuale, invece di avere rilocazione separata solo per i segmenti di testo e dati, l'intero spazio di indirizzamento può essere rimappato sulla memoria fisica in pagine di stessa dimensione.
- La memoria virtuale funziona bene in un sistema multiprogrammato, con bit e pezzi di molti programmi contemporaneamente in memoria.
- Mentre un programma è in attesa che venga caricata la pagina richiesta, la CPU può essere assegnata ad un altro processo.

Paging

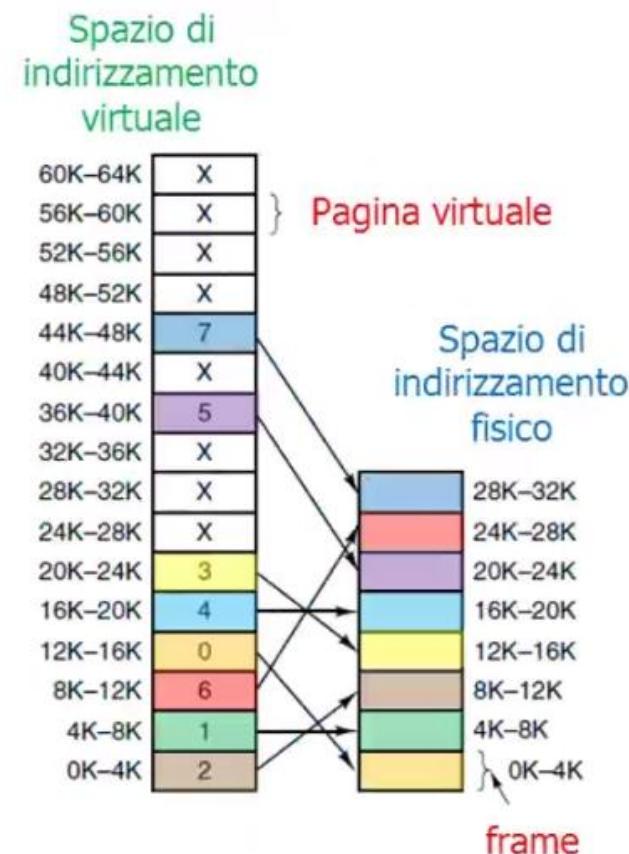
- La maggior parte dei sistemi di memoria virtuale utilizza una tecnica chiamata **paginazione** o **paging**.
- Quando un programma esegue l'istruzione:

copia il contenuto della locazione di memoria 1000 nel registro REG.
- In un computer con memoria virtuale l'indirizzo non finisce sul bus indirizzi ma in una unità chiamata **MMU (Memory Management Unit)**.
- Un bit di **presenza / assenza** tiene traccia di quali pagine sono presenti fisicamente in memoria.



Paging

- Nei computer senza memoria virtuale, l'indirizzo virtuale viene messo direttamente sull'address bus (la parola di memoria fisica è coincide con quella virtuale).
- Lo spazio di indirizzo virtuale è suddiviso in unità di dimensione fissa chiamati **pagine**.
- Le unità fisiche corrispondenti alle pagine (virtuali) nella memoria sono chiamate **frame** o **page frame**.
- Le **pagine** e i **frame** hanno in genere le stesse dimensioni.
- Nella figura pagina il frame 0K-4K indica l'intervallo di indirizzi da 0 a 4095. Il successivo frame (4K-8K) fa riferimento agli indirizzi da 4096 al 8191 e così via.

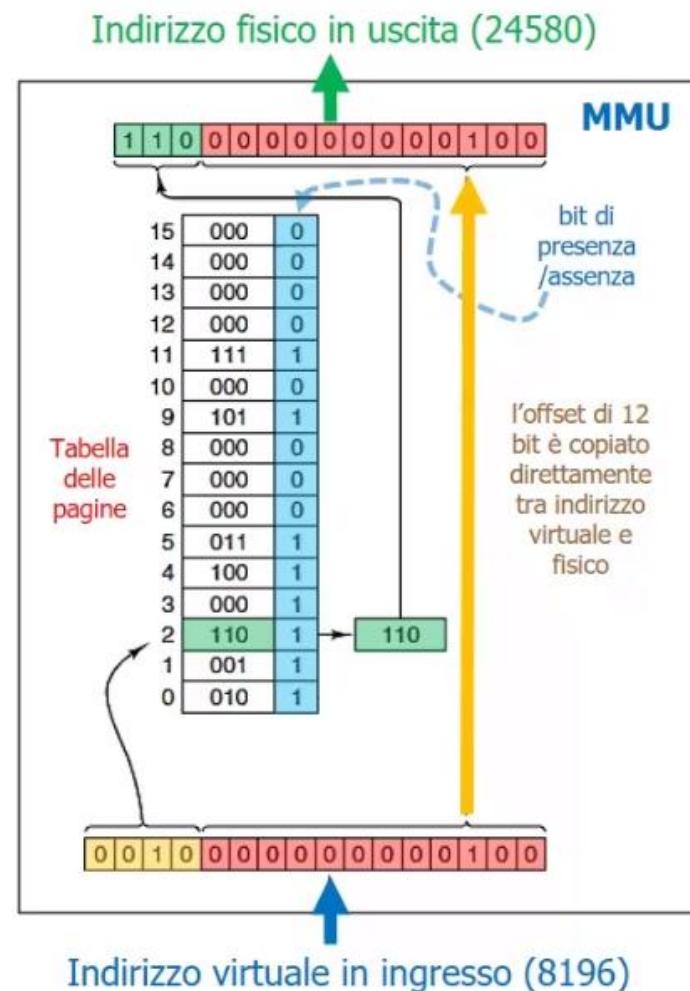


Paging

- Quando una pagina virtuale non è nella memoria, la MMU si accorge che il riferimento nell'istruzione non è in memoria e causa un errore di pagina (**page fault**) al sistema operativo.
- 1 Il sistema operativo seleziona il frame meno utilizzato e lo scrive sul disco (se è stato modificato).
 - 2 Poi sovrascrive il suo contenuto con la pagina appena caricata.
 - 3 Cambia la mappa all'interno della MMU.
 - 4 Riavvia l'istruzione che aveva causato la **trap**.
- All'interno della MMU c'è una **tabella delle pagine** che permette di traslare un indirizzo virtuale in uno fisico.

MMU: indirizzo virtuale → indirizzo fisico

- Una parte dell'indirizzo virtuale è utilizzato come indice nella tabella delle pagine, il resto come offset all'indirizzo estratto dalla tabella.



Struttura della riga nella tabella delle pagine

- Anche se la struttura fisica del record dipende dalla macchina, il tipo di informazioni presenti è generalmente lo stesso:

N° Frame	Presenza / Assenza	Protezione	Pagina Modificata	Pagina Referenziata	Disabilità caching	
----------	--------------------	------------	-------------------	---------------------	--------------------	--

- Il **numero del frame** (o **page frame**).
- Il bit di **presenza(1)/assenza(0)**, quando è 0 significa che la pagina virtuale non è in memoria (page fault) e va caricata.
- I bit di **protezione** specificano che tipo di accesso è consentito (come i bit rwx di UNIX).

Struttura della riga nella tabella delle pagine

- Il bit **pagina modificata** (o sporca) è asserito quando la pagina è stata modificata, prima di sovrascriverla in memoria e va scritta su disco.
- Il bit **pagina referenziata** è impostato se è stato fatto un accesso in lettura o in scrittura alla pagina. È utilizzato come criterio per lo swap delle pagine quando si verifica un page fault.
- Il bit **disabilità cache** permette di disabilitare il caching della pagina.

N° Frame	Presenza / Assenza	Protezione	Pagina Modificata	Pagina Referenziata	Disabilità caching	
----------	--------------------	------------	-------------------	---------------------	--------------------	--

Velocizzare la paginazione

- In qualsiasi sistema di paging, devono essere analizzati due aspetti importanti:
 - 1 La conversione dell'indirizzo virtuale in quello fisica deve essere veloce.
 - 2 Più è ampio lo spazio di indirizzamento virtuale maggiore è la dimensione della tabella delle pagine.
- Il più semplice approccio è disporre di una tabella delle pagine che contiene una serie di registri hardware indicizzati per pagina:
 - è un approccio semplice che non richiede riferimenti di memoria durante la mappatura;
 - è costoso se la tabella delle pagine è grande e può avere basse prestazioni poiché richiede il caricamento della intera tabella delle pagine ad ogni cambio di contesto.

Velocizzare la paginazione

- Se invece la tabella delle pagine è interamente in memoria principale:
 - L'hardware ha bisogno solo di un registro (che punta alla tabella delle pagine) e quindi è molto flessibile.
 - L'accesso alla memoria rallenta le prestazioni.

Translation Lookaside Buffers (TLB)

- La tabella delle pagine è normalmente in memoria.
- Partendo dall'osservazione che i programmi tentano a fare molti riferimenti ad un numero ridotto di pagine (e non il contrario!):
 - solo una piccola porzione delle righe della tabella delle pagine sono lette pesantemente, mentre il resto sono poco accedute.
- La soluzione è di equipaggiare i computer con un piccolo dispositivo hardware che mappi gli indirizzi virtuali verso i fisici senza andare nella tabella delle pagine.
- Questo dispositivo si chiama **TLB (Translation Lookaside Buffer)**, o talvolta **memoria associativa**, è interna alla MMU e consiste di un ridotto numero di righe (8 o al massimo 64).
- Ogni riga contiene informazioni su una pagina, incluso il numero della pagina virtuale, un bit che è impostato quando la pagina è modificata, i permessi e il frame di dove è memorizzata la pagina in memoria.

Gestione software della TLB

- Le righe della TLB vengono caricate dal sistema operativo e quando si verifica una **TLB miss**, l'MMU passa il problema al sistema operativo con un errore TLB.
 - Il SO deve farsi carico di cercare la pagina nella tabella delle pagine, sostituire la riga nella TLB e rieseguire l'istruzione che ha generato la trap.
 - Tutto deve avvenire in poche istruzioni poiche le **TLB miss** sono molto più frequenti che le **page fault**.
- Se la TLB è ragionevolmente grande (64 righe) e il software di gestione efficiente (pochi **TLB miss**) questo riduce la complessità della MMU e libera spazio sulla CPU (es. per le cache).
- Sono state ideate varie soluzioni per ridurre le **TLB miss** a volte il sistema operativo può caricare in anticipo la TLB con delle pagine che è probabile che nel breve possano essere utilizzate.

Gestione software della TLB

- Quando accade una **TLB miss** (hardware o software) occorre andare nella tabella delle pagine e cercare la pagina referenziata.
- Se le pagine che contengono la tabella delle pagine è sempre nella TLB questa ricerca è più veloce.
- Un modo è di riservare uno spazio di posizioni fisse nella TLB in modo che queste righe non possano essere rimosse quando accade un **TLB miss**.

Gestione software della TLB

- Nella gestione della TLB via software ci sono due tipi di errori:
 - **soft miss**, accadono quando la pagina è referenziata e non è nella TLB, ma è in memoria → bisogna aggiornare la TLB.

Nessun accesso al disco è necessario.

Tipicamente una **soft miss** richiede 10-20 istruzioni macchina per essere gestita e può essere completata in pochi **ns**.
 - **hard miss**, accadono quando la pagina non è in memoria (e neanche nella TLB) quindi è richiesto un accesso al disco per caricare la pagina (ordine dei **ms**).
- Un **hard miss** è alcuni milioni di volte più lento rispetto ad un **soft miss**.

Tabelle delle pagine per memorie grandi

- I dispositivi TLB sono utilizzati per incrementare la velocità nella traduzione degli indirizzi virtuali in fisici.
- L'altro problema affrontato è come gestire spazio virtuali ampi, esistono due differenti approcci:

1

Tabelle delle pagine multilivello.

2

Tabella delle pagine invertite.

Tabella delle pagine multilivello

- Supponiamo di avere un indirizzo virtuale a 32-bit partizionato in tre campi: **PT1** (10 bit), **PT2** (10 bit) e **Offset** (12 bit).
- In questo esempio si hanno **2²⁰ pagine di 4 KB** (**Offset**).
- La tabella delle pagine è organizzata in due livelli: il primo è indirizzato da **PT1** e il secondo da **PT2**.
- Quando la MMU deve tradurre un indirizzo virtuale, utilizza i primi bit più significativi come **PT1** e i seguenti per **PT2** e **Offset**.
- Per esempio l'indirizzo virtuale 0x00403004 può essere diviso come segue:

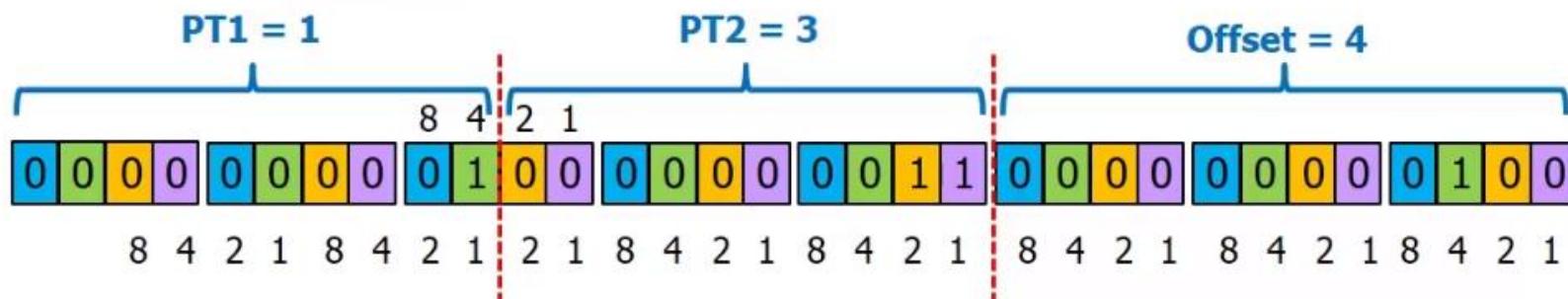


Tabella delle pagine multilivello

- La MMU utilizza prima **PT1** per indirizzare la tabella delle pagine di primo livello e ottenere la riga delle pagine di secondo livello (indirizzi da 4M a 8M).
- La MMU poi utilizza **PT2** come indice nella tabella delle pagine di secondo livello per individuare ed estrarre la riga che contiene il numero di frame.

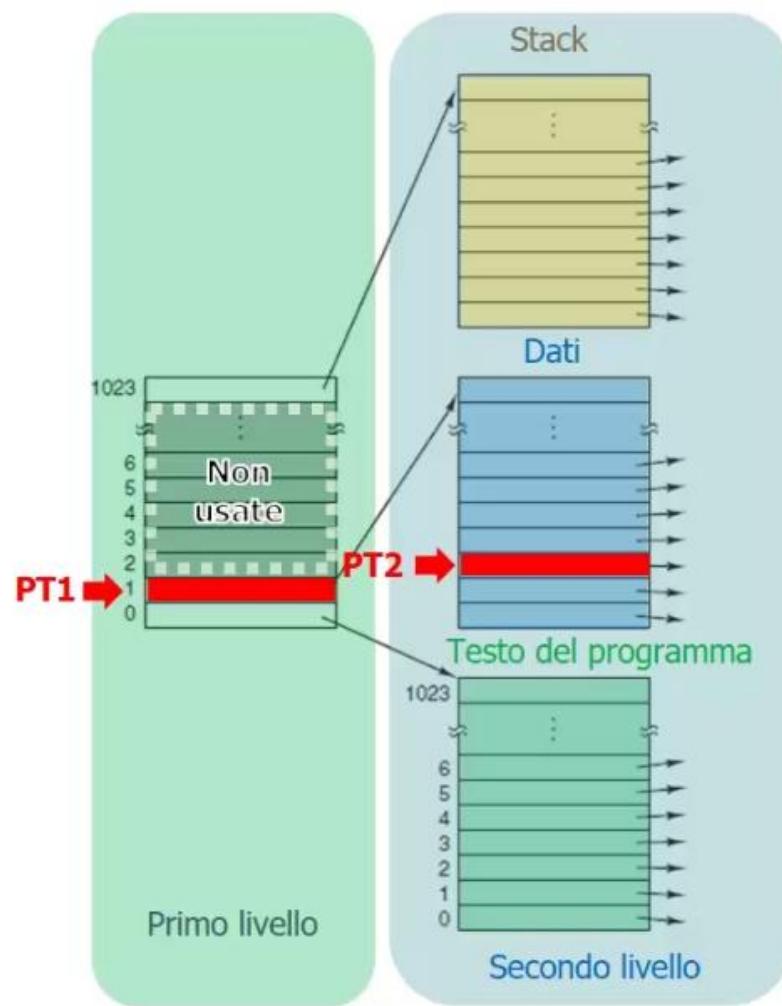


Tabella delle pagine multilivello

- La cosa intressante da osservare è che sebbene lo spazio di indirizzamento contenga più di un **milione di pagine**, sono utilizzate solo **4 tavole delle pagine!** (la tabella di primo livello e quelle di secondo livello: programma, dati e stack).
- I bit di **presenza/assenza** di 1021 righe della tabella di primo livello inutilizzate sono impostati a zero a garanzia che nessun processo abbia mai accesso a tali righe, se ciò accadesse verrebbe generato un **page fault** e il SO se ne accorgerebbe.
- Il sistema può essere esteso anche a più di due livelli favorendo così una maggiore flessibilità.

Tabelle delle pagine invertite

- Le tavole delle pagine multilivello lavora bene fino a 32-bit di spazio di indirizzi virtuali.
- Con computer da 64-bit la tabella delle pagine diventerebbe enorme (più di **30 millioni di GB**).
 - Poiché lo spazio di indirizzamento è 2^{64} byte, se si utilizzano pagine da **4 KB**, la tabella delle pagine ha **252** righe.
 - Se ogni riga occupa **8 Byte**, la tabella è di oltre **30 PB!**

Tabelle delle pagine invertite

- Una soluzione differente è quella della **tabella delle pagine invertite**: si utilizza una riga per frame nella memoria reale piuttosto che una riga per pagina di indirizzo virtuale.
- Con un indirizzamento virtuale di 64-bit, una pagina di 4 KB e 1 GB di RAM, la tabella delle pagine invertita richiede solo 262.144 righe.
- La riga tiene traccia di ciò che è memorizzato nel frame (processo, pagina virtuale).
- La traduzione da indirizzo virtuale a fisico diventa più difficile.

Tabelle delle pagine invertite

- Quando un processo **n** riferenzia una pagina virtuale **p**, l'hardware non può utilizzare **p** come indice nella tabella delle pagine, ma deve cercare la coppia **<n,p>** in tutta la tabella delle pagine invertite.
- Questa ricerca deve essere fatta per ogni riferimento di memoria, non solo sui **page fault!**
- Una possibile soluzione al problema è attraverso l'impiego di una TLB che memorizza le pagine più pesantemente utilizzate, in questo caso la traduzione è veloce come accadeva per la classica tabella delle pagine:
 - Se accade una **TLB miss**, deve essere effettuata la ricerca nella tabella delle pagine invertite. In questo caso si può utilizzare una tabella hash sugli indirizzi virtuali.