Sistemi operativi

4. Gestione della memoria

Mauro Fiorentini

4. Gestione della memoria

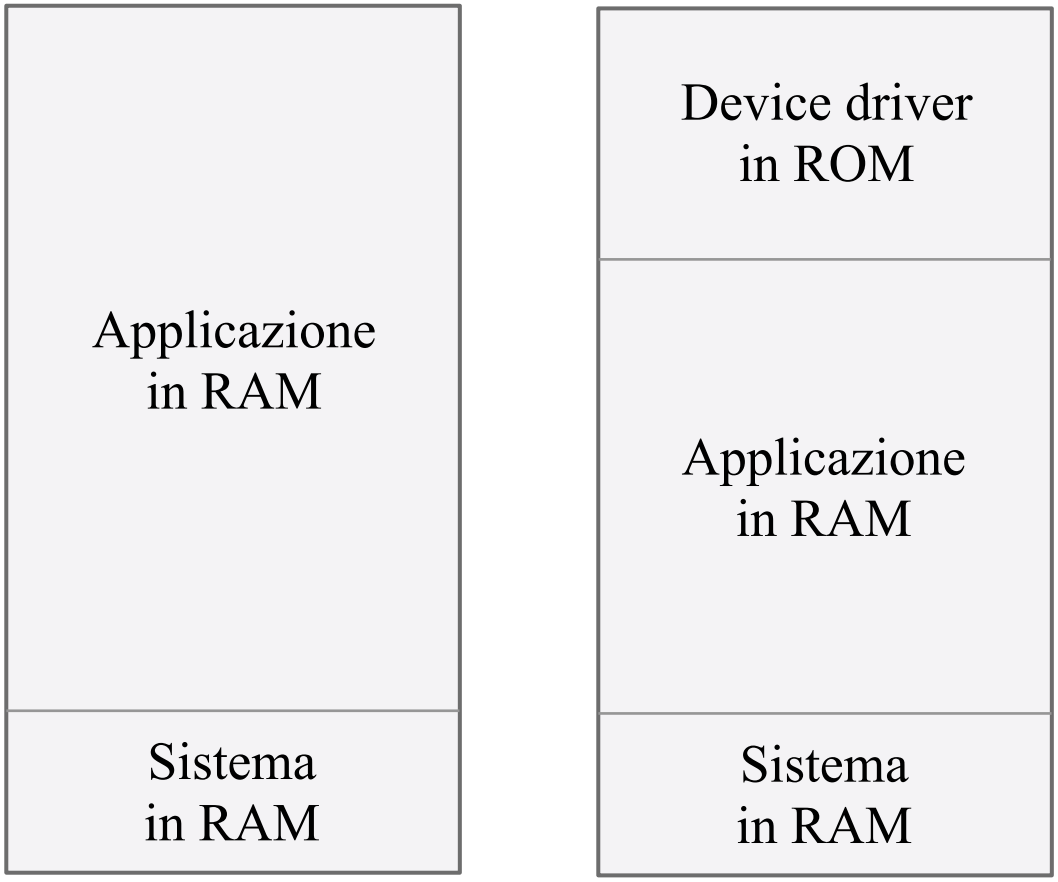
Il problema della memoria (1)

- La multiprogrammazione genera la necessità di un partizionamento della memoria e di protezioni, per evitare che un processo ne possa danneggiare altri.
- Il problema è presente anche in caso di monoprogrammazione.
  - Un programma può richiedere più memoria di quella disponibile.

Il problema della memoria (2)

- Le principali soluzioni:
  - swapping;
  - overlay;
  - memoria virtuale:
    - paginata;
    - segmentata.

Memoria in macchina monoprogrammata



Memoria in macchina monoprogrammata

- Il sistema carica i programmi, uno alla volta, mettendo a loro disposizione tutta la memoria disponibile.
  - Se il programma ne richiede di più, non può essere eseguito.
- L'interprete dei comandi viene ricaricato quando dev'essere eseguito, come un programma qualunque.
  - In alternativa, l'interprete e altri piccoli programmi (detti TSR in MS-DOS) possono rimanere sempre in RAM dopo il caricamento.

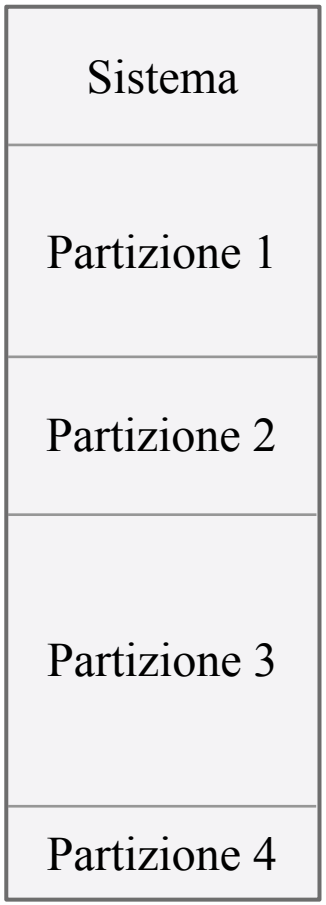
Indirizzamento virtuale (1)

- Gli indirizzi di ogni processo si riferiscono a uno spazio “virtuale” di indirizzamento, identico per tutti i processi e molto grande (da qualche Mbyte a parecchi Gbyte).
- Rende lo spazio di indirizzamento del processo (virtuale) indipendente da quello reale.
- Indispensabile in caso di multiprogrammazione, perché più processi potrebbero richiedere l’uso degli stessi indirizzi.

Indirizzamento virtuale (2)

- Permette a ogni processo di comportarsi come se fosse padrone della macchina, ignorando gli altri.
- Gli indirizzi virtuali vengono tradotti in indirizzi reali da hardware dedicato.
- Di solito in ogni istante solo una parte dello spazio di indirizzamento di un processo riferisce memoria reale; il resto si trova in un'immagine su memoria di massa (disco).

Partizioni fisse



- Utilizzate nei primi sistemi multiprogrammati (Es.: OS/360).
- La memoria viene suddivisa in partizioni fisse.
- A ogni processo viene assegnata una partizione.
- Ogni processo viene compilato come se avesse a disposizione una partizione che inizia a indirizzo zero.
- Per tradurre un indirizzo virtuale in indirizzo reale basta sommargli l'indirizzo di inizio della partizione assegnata al processo.

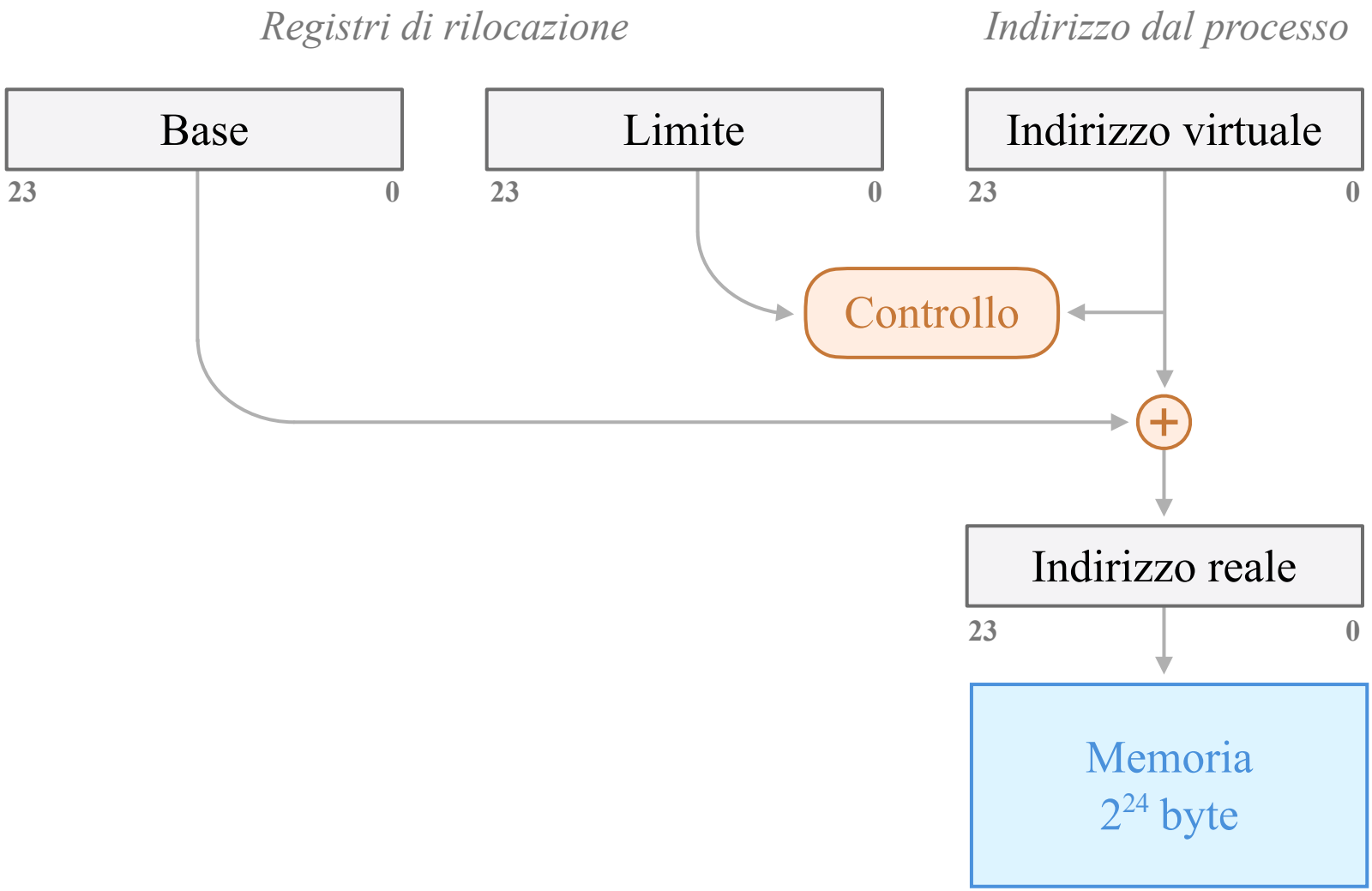
Partizioni e code

- Si può utilizzare una coda di processi in attesa per ogni partizione.
  - Si rischia di lasciare una partizione inutilizzata, mentre più processi attendono su altre code.
- In alternativa si può utilizzare una coda unica e assegnare a ogni processo la più piccola partizione in grado di contenerlo.
  - Il primo processo della coda può essere scavalcato, se la massima partizione libera non può accoglierlo.

Partizioni e rilocazione

- Rilocazione durante il caricamento: agli indirizzi viene sommato l'effettivo indirizzo di inizio della partizione.
  - Il loader deve distinguere gli indirizzi dal resto (istruzioni e dati).
  - Il formato dei file eseguibili si complica.
- Come alternativa la rilocazione può essere eseguita dall'hardware.
  - Serve un registro apposito.
  - Si può verificare il rispetto dei limiti.

Rilocazione via hardware (1)



Rilocazione via hardware (2)

- I registri di rilocazione, base e limite, vengono aggiornati dal sistema a ogni context switch.
- Non possono essere alterati in user mode.
- A volte contengono solo una parte (i bit più significativi) dell'indirizzo, mentre il resto è zero.
  - Si rende più veloce la somma.
  - Le partizioni iniziano a indirizzi multipli di una potenza di 2.

Vantaggi della rilocazione via hardware

- Caricamento più semplice
- Se il sistema decide di spostare un processo, deve copiare la memoria occupata e modificare solo il registro base.
  - Se la rilocazione fosse effettuata al caricamento, non basterebbe ricaricare il processo, perché gli indirizzi calcolati nelle variabili (puntatori) non sarebbero più validi.

Multiprogrammazione a partizioni variabili

- Consiste nell'assegnare a ogni processo, al caricamento, la memoria richiesta.
- La suddivisione non è prefissata, ma dipende dalle esigenze.

Problemi creati dal partizionamento

- Frammentazione: suddivisione della memoria in blocchi liberi e occupati alternati.
  - Riproduce, su scala maggiore, i problemi di frammentazione dello heap dei linguaggi che gestiscono memoria dinamica.
- E' difficile soddisfare una richiesta di ulteriore memoria da parte di un processo in esecuzione.
  - Per esempio, per ampliare stack o heap.

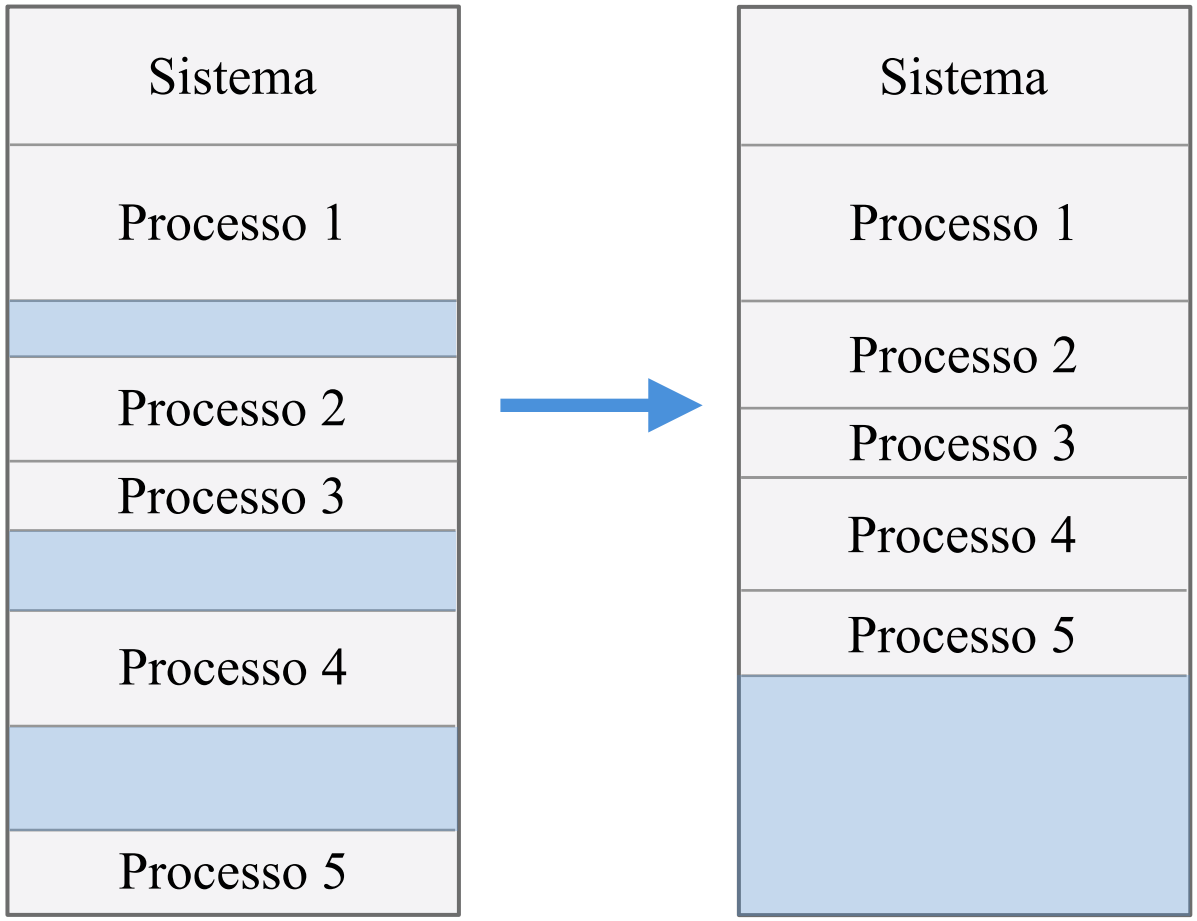
Frammentazione

- I blocchi liberi possono essere numerosi, ma piccoli: si ha la **frammentazione esterna**.
- Può non esserci abbastanza memoria libera contigua per un nuovo processo, anche se il totale di memoria disponibile è molto maggiore della memoria richiesta.

Soluzioni al problema della frammentazione

- Aspettare a caricare il nuovo processo.
  - La memoria può essere molto sottoutilizzata.
  - Si può permettere il caricamento di un processo più piccolo
    - Bisogna evitare che un processo sia scavalcato nella coda troppe volte.
- "Compattare" la memoria, spostando i blocchi allocati e creando un unico grande blocco libero a un'estremità.
  - Raramente attuata, perché costa moto tempo.

Frammentazione e compattazione



Gestione della memoria

- Serve una struttura dati per gestire l'allocazione:
  - bitmap;
  - liste;
  - liste multiple;
  - buddy system ("blocchi gemelli").
- La struttura è analoga a quelle utilizzate per lo heap nei linguaggi con memoria dinamica.

Gestione a bitmap

- La memoria disponibile viene suddivisa in elementi di uguale lunghezza.
  - Da pochi byte a qualche centinaio.
- Una tabella contiene un bit per ogni elemento, che indica se l'elemento è libero o occupato.

Caratteristiche delle bitmap

- La tabella ha dimensioni contenute.
  - Basso overhead, proporzionale alla memoria.
- Ridotto spreco per ogni blocco allocato:
  - minore della dimensione di ogni elemento;
  - in media mezzo elemento per blocco allocato.
- Per allocare un'area, bisogna cercare nella tabella una stringa di zeri consecutivi di lunghezza adeguata.
  - Operazione piuttosto lenta.
- Raramente utilizzate.

Gestione a liste

- Si utilizza un descrittore per ogni blocco.
  - Indica se libero od occupato, indirizzo e dimensione.
- I descrittori sono collegati in una lista.
  - Solitamente ordinata per indirizzo di memoria.
  - Quando si suddivide un blocco, si aggiunge un descrittore.
  - Quando si libera un blocco contiguo a uno già libero, si collassano i blocchi, eliminando un descrittore.

Caratteristiche delle liste

- Ridotta occupazione di memoria.
  - Proporzionale al numero di blocchi.
- Lo spreco per ogni blocco allocato è minimo.
- Utilizzate spesso.
  - Varianti di questa tecnica sono la norma nella gestione di heap.
  - Nella gestione di heap, i descrittori sono posti di solito in testa ai blocchi.

Ricerca di blocchi liberi

- Per allocare un blocco, si cerca un descrittore di un blocco libero di lunghezza adeguata.
- Per velocizzare le ricerche, si possono mantenere due liste separate di blocchi liberi e occupati.
  - Con alcuni algoritmi, conviene ordinare la lista dei blocchi liberi per dimensione crescente dei blocchi, l'altra per indirizzo.

Collasso di blocchi liberi

- Quando un blocco viene deallocato, è necessario esaminare i blocchi adiacenti; se sono liberi, si collassano in un unico blocco, riducendo la frammentazione.
- Può essere necessario scandire tutta la lista dei blocchi, per trovare quello precedente.
  - Se si usa una sola lista, conviene che il collasso dei blocchi liberi sia fatto durante l'allocazione, quando si scandisce comunque una parte della lista.

Algoritmi di ricerca (1)

- Alcuni esaminano solo una parte della lista, quindi sono più veloci.
  - **First fit**: si utilizza il primo blocco valido.
  - **Next fit**: come first fit, ma la lista è circolare e la ricerca inizia dall'ultimo blocco allocato.
    - Generalmente non migliore.
- Utilizzati quando la frammentazione è un problema minore (p. es., quando sono allocati blocchi di poche dimensioni diverse).

Algoritmi di ricerca (2)

- Altri esaminano tutta la lista, per ridurre la frammentazione.
  - **Best fit**: si utilizza il minimo blocco valido.
    - Tende a produrre molti blocchi liberi piccoli.
  - **Worst fit**: si utilizza il blocco massimo.
    - Tende a produrre pochi blocchi liberi grandi.
    - In pratica non dà risultati molto migliori.
- Utilizzabili nella forma base solo se i blocchi sono relativamente pochi.

Liste multiple

- Alcuni algoritmi utilizzano più liste di blocchi liberi, per blocchi di dimensione diversa, di solito in progressione geometrica (es.: aree da 2K, 4K, 8K, 16K ecc.).
  - **Quick fit**: si utilizza la prima area valida.
    - In pratica un first fit, tra blocchi di dimensione omogenea.
    - Molto veloce, buon compromesso.
  - Avendo più liste, mediamente corte, si può ricorrere al best fit.

Buddy system (1)

- L'area richiesta viene arrotondata alla minima potenza di due.
- Si usa una lista per ogni possibile dimensione dei blocchi.
- Il blocco da allocare si sceglie col quick fit.
  - Se non esiste un blocco libero della dimensione richiesta, si divide in due un blocco della minima dimensione disponibile.
  - La suddivisione viene ripetuta finché una delle parti ha la dimensione richiesta.

Buddy system (2)

- Molto veloce nella ricerca di blocchi liberi adiacenti da collassare.
  - Solo la lista di blocchi della stessa dimensione dev'essere esaminata.
- Provoca **frammentazione interna**: spreco di spazio **all'interno** dei blocchi allocati.
  - In media, 1/4 della memoria utilizzata.

Liste separate

- Ogni descrittore di blocco libero appartiene contemporaneamente a due liste:
  - la lista dei blocchi nello stesso intervallo di dimensioni, utilizzata per l'allocazione;
  - la lista dei blocchi in ordine di indirizzo, utilizzata per collassare aree libere adiacenti.
- La ricerca di aree libere adiacenti comporta l'esame di due soli possibili candidati.
- Le liste sono doppiamente concatenate, per rendere più veloci le operazioni.

Risultati importanti

- In un sistema con n processi vi sono in media $n / 2$ blocchi liberi (Knuth 1973).
- Se la dimensione media dei blocchi liberi è k volte quella dei blocchi allocati, la frazione di memoria inutilizzata è in media $k / (k + 2)$.
  - Conviene ridurre k per migliorare l'utilizzo della memoria.

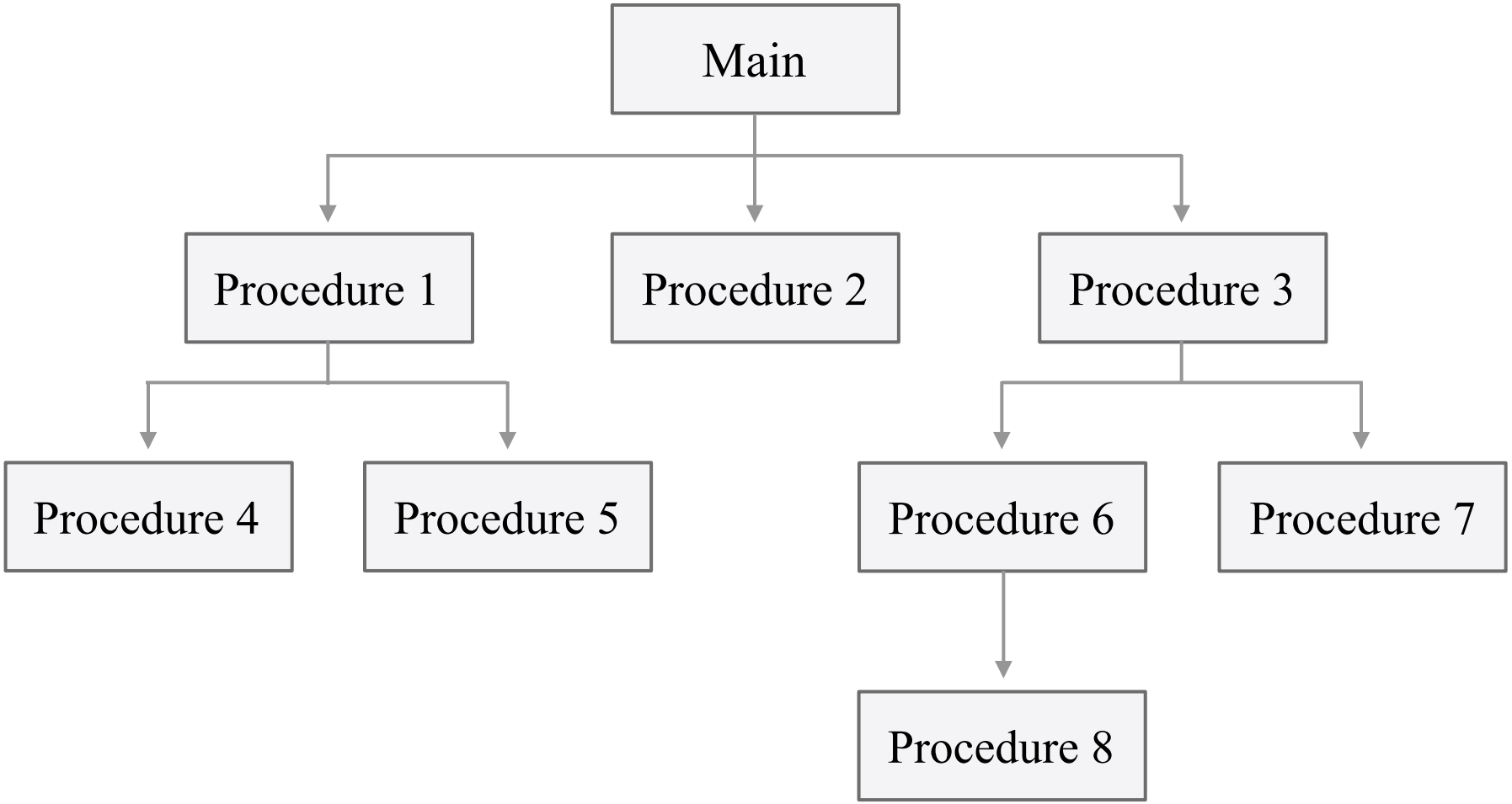
Swapping

- Consiste nello scaricare temporaneamente su disco la memoria utilizzata da uno o più processi, per far posto ad altri.
  - Se frequente, impone un carico di I/O elevato.
  - La memoria può essere sottoutilizzata.
  - Nessun processo può usare più memoria di quella fisica.

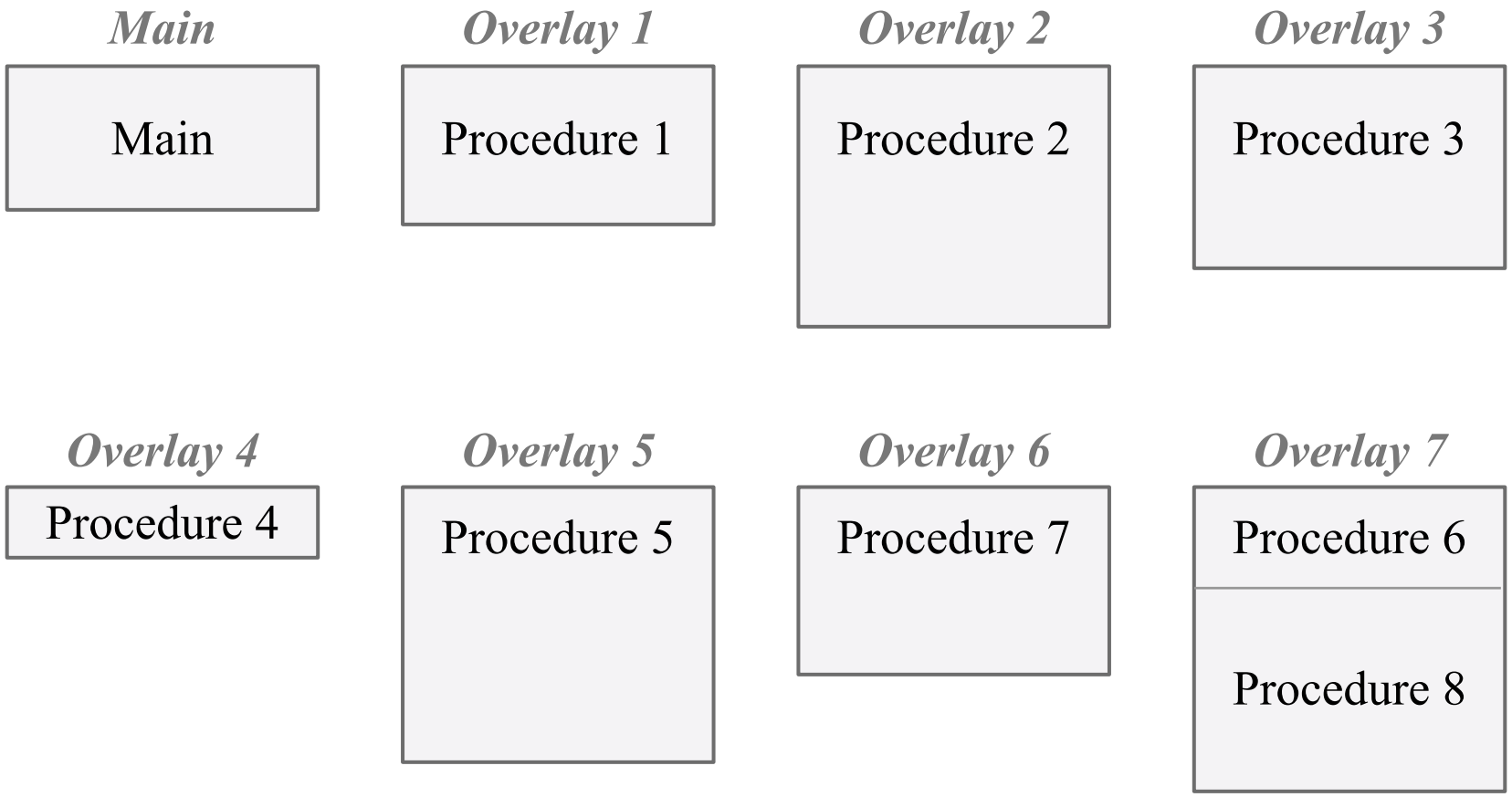
Overlay

- Tecnica sviluppata negli anni ‘60.
- Un programma viene suddiviso in parti che **possono non essere simultaneamente presenti in memoria**.
- Si utilizzano due suddivisioni differenti e parallele per codice e dati.

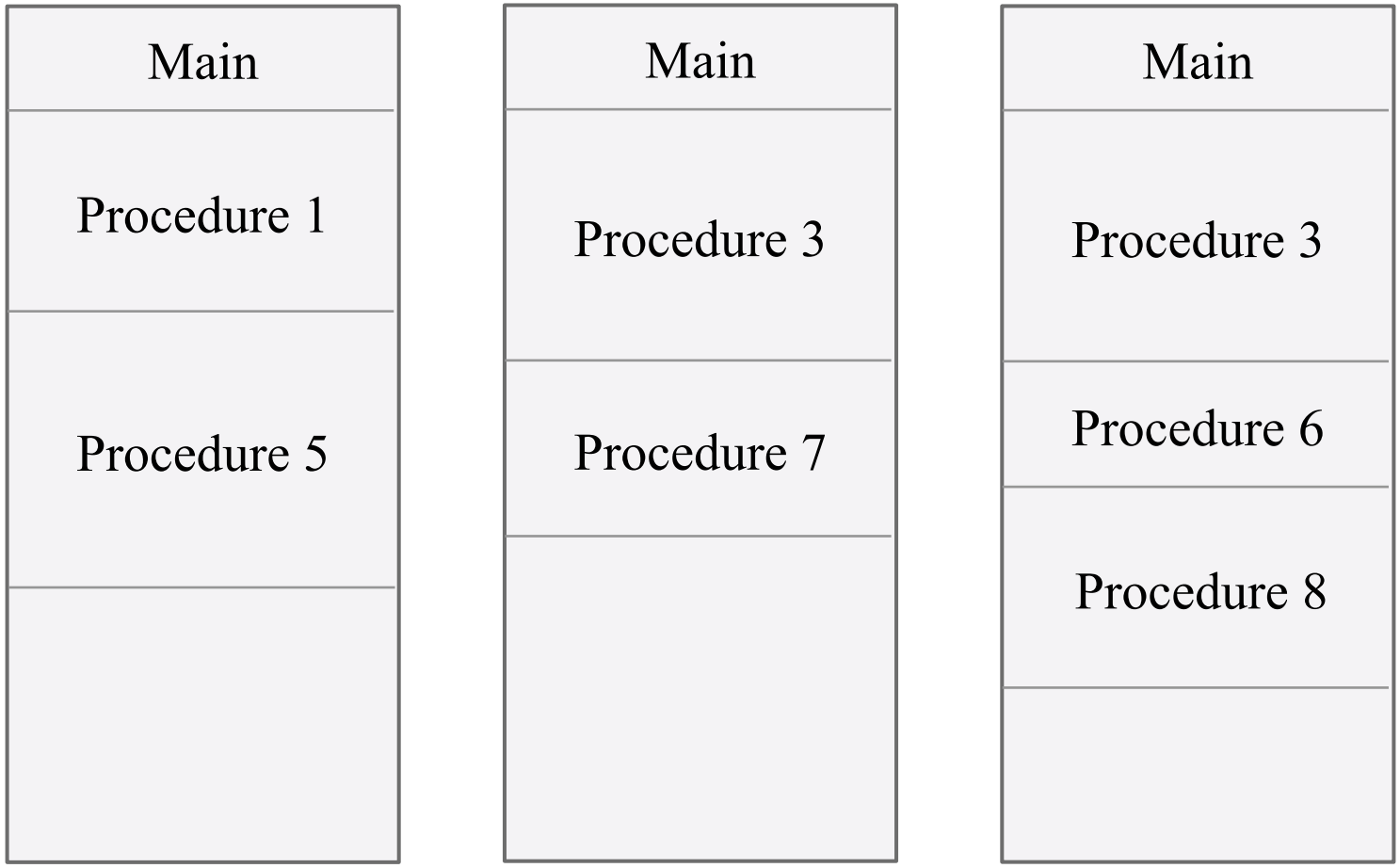
Overlay: il codice



Overlay: la suddivisione



Overlay: la memoria



Gestione degli overlay

- Ogni funzione viene caricata sempre nella stessa posizione.
- L'immagine del programma consiste di più parti separate, dette overlay.
- I vari overlay vengono letti da disco quando necessari.
- Gli overlay dei dati devono essere salvati su disco prima di essere sovrascritti.

Definizione degli overlay

- Vengono creati mediante direttive al linker.
- Interamente **a carico del programmatore**.
  - La suddivisione di programmi complessi diviene un incubo.
  - In presenza di ricorsione tra funzioni diverse diviene complicata e spesso inefficiente.

Caricamento degli overlay

- Alle origini a carico del programmatore.
  - Effettuato mediante chiamate di sistema.
- In seguito parzialmente automatizzato.
  - Le chiamate al sistema vengono inserite dal linker, modificando le chiamate alle funzioni appartenenti a un diverso overlay.
- Il tempo di caricamento può essere sfruttato, cedendo il controllo a un altro processo.

Limiti degli overlay

- Soluzione adeguata per il codice se:
  - utilizza poco la ricorsione;
  - può essere suddiviso in blocchi indipendenti;
  - è organizzato gerarchicamente.
- Soluzione valida per i dati se:
  - allocati staticamente;
  - stack e heap non esistono o hanno dimensioni minime.

Vantaggi degli overlay

- Interamente gestibili via software.
  - Non aumentano i costi dell'hardware.
- Permettono di far girare programmi **molto più grandi della memoria fisica**.
  - Addirittura più grandi dell'address space.

Svantaggi degli overlay (1)

- Irrigidiscono la struttura dei programmi.
  - Non è sempre possibile chiamare qualsiasi funzione.
  - Per passare da una funzione di un overlay a una di un altro, bisogna passare da una funzione che possa coesistere in memoria con i due overlay.
- Le modifiche ai programmi possono costringere a **riorganizzare gli overlay**.

Svantaggi degli overlay (2)

- Non applicabili in tutti i casi.
- Soluzione valida per linguaggi non ricorsivi ad allocazione prevalentemente **statica** (COBOL, FORTRAN).
- Difficili da gestire per programmi di grandi dimensioni.
- Utilizzati solo nei sistemi più semplici.

Memoria virtuale paginata

- Generalizzazione degli overlay, a gestione automatica.
- La memoria reale viene suddivisa in frammenti (pagine), tutti della stessa dimensione (da 1 a 16 Kbyte).
- Copie delle pagine si trovano su disco.
- Le pagine vengono caricate dal sistema quando richieste.

Vantaggi della memoria virtuale

- Gestione interamente automatica.
- Permette di far girare programmi **più grandi della memoria fisica**.
- Uso ottimale della memoria e della CPU.
  - Se vi sono più processi, la CPU non resta quasi mai inoperosa.
- I/O ridotto rispetto allo swapping.

Accesso alla memoria virtuale

- Per ogni accesso la macchina:
  - suddivide l'indirizzo in numero di pagina e offset;
  - verifica la presenza della pagina di memoria richiesta e la sua accessibilità;
  - traduce l'indirizzo virtuale in indirizzo reale;
  - accede alla memoria reale o genera un errore (page fault).

Traduzione degli indirizzi

- Richiede un complesso supporto hardware.
- Per ogni processo il sistema operativo alloca una page table.
  - Il numero di pagina funge da **indice** nella page table e permette di accedere all'indirizzo reale della pagina.
- L'offset viene **concatenato** all'indirizzo di base della pagina

Gestione delle pagine

- Quando un processo tenta di accedere a una pagina non presente o non accessibile, si verifica un **page fault**, scoperto dall'hardware tramite i bit di stato/ protezione.
- Può trattarsi di:
  - accesso non autorizzato; il sistema prende i provvedimenti (es.: uccide il processo);
  - accesso a memoria non fisicamente presente.

Gestione dei page fault

- Il sistema operativo, caricata la pagina mancante, deve annullare gli effetti collaterali dell'istruzione interrotta, per poterla far rieseguire.
- In particolare deve:
  - ripristinare nel program counter l'indirizzo dell'istruzione;
    - può puntare oltre l'istruzione o in mezzo;
  - annullare gli effetti di eventuali autoincrementi/ autodecrementi di registri.

Supporto hardware alla gestione dei page fault

- Varia molto da macchina a macchina:
  - alcuni processori annullano tutti gli effetti, prima di generare un trap (es.: VAX);
  - altri salvano lo stato interno, in modo da poter ripartire dall'interruzione da metà strada (es.: Motorola 68010);
  - altri salvano solo il program counter prima del fault, costringendo il sistema a decodificare l'istruzione per annullarne gli effetti.

Page fault e RISC

- Su moderni processori RISC superscalari, istruzioni successive possono essere state eseguite dopo il page fault.
- I loro effetti **vanno annullati**.
- Il ripristino dello stato può essere molto complicato.

Trattamento di un page fault (1)

- La sequenza di eventi innescata da un page fault è la seguente.
  - L'hardware rileva il fault e genera un **trap**, salvando informazioni sul fault stesso sullo stack o in registri appositi.
  - Una routine di gestione trap (scritta in assembler) salva tutti i registri e chiama una funzione del sistema.
  - Il sistema identifica la pagina richiesta, utilizzando le informazioni salvate nei registri o decodificando l'istruzione.

Trattamento di un page fault (2)

- Il sistema verifica che il processo abbia il permesso di accedere alla pagina richiesta.
- Il sistema seleziona una pagina fisica dove caricare la pagina richiesta.
  - Se non vi sono pagine libere utilizza l'algoritmo di scelta della pagina da scaricare su disco.
- Se la pagina è stata modificata, viene iniziata la scrittura su disco.
  - La pagina viene intanto marcata "bloccata e non scrivibile" per evitare ulteriori modifiche.

Trattamento di un page fault (3)

- La pagina richiesta viene caricata da disco.
  - Durante gli accessi alla memoria esterna il programma viene posto in attesa e un altro processo diventa running.
- Il sistema aggiorna la page table.
  - Se la pagina è condivisa, vengono aggiornate anche le page table dei processi che la condividono.
- L'immagine dei registri del processo interrotto viene riportata a uno stato che consenta la riesecuzione dell'istruzione, se necessario.

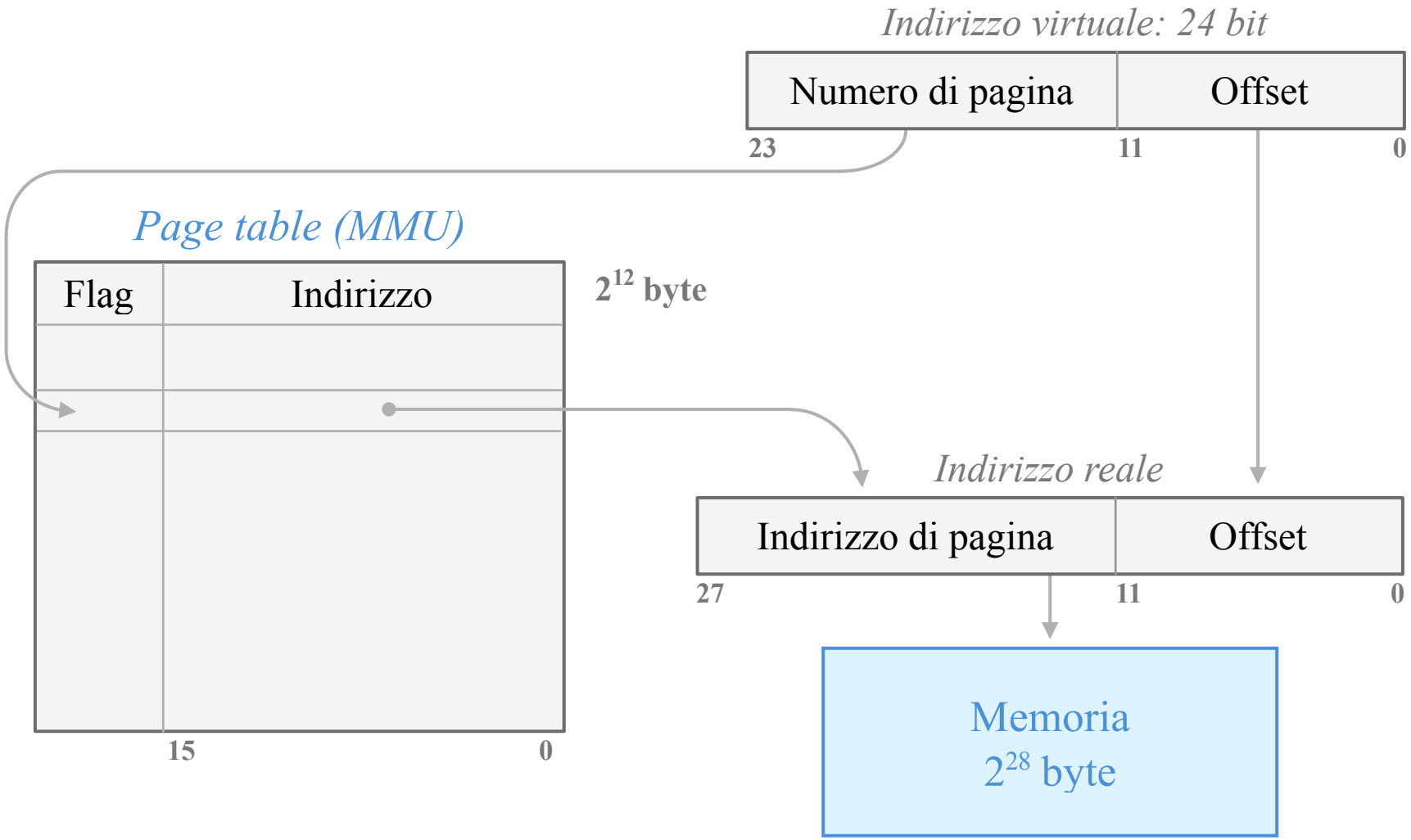
Trattamento di un page fault (4)

- L'istruzione che ha causato il fault viene rieseguita.
  - In alcune macchine viene solo completata, perché l'hardware salva le informazioni necessarie per riprenderla da metà.
- Il sistema ritorna alla routine di gestione trap.
- La routine ricarica i registri e ritorna allo stato utente.
  - Per il processo il fault è completamente invisibile, tranne per il tempo trascorso.

MMU

- Nei primi sistemi la page table risiedeva in appositi componenti fisici: MMU (Memory Management Unit).
  - Il numero di pagine è forzatamente ridotto.
  - A ogni context switch bisogna ricaricare **tutta la page table** dalla copia in memoria.
  - La traduzione è molto veloce.
  - Soluzione adeguata solo per page table piuttosto piccole.

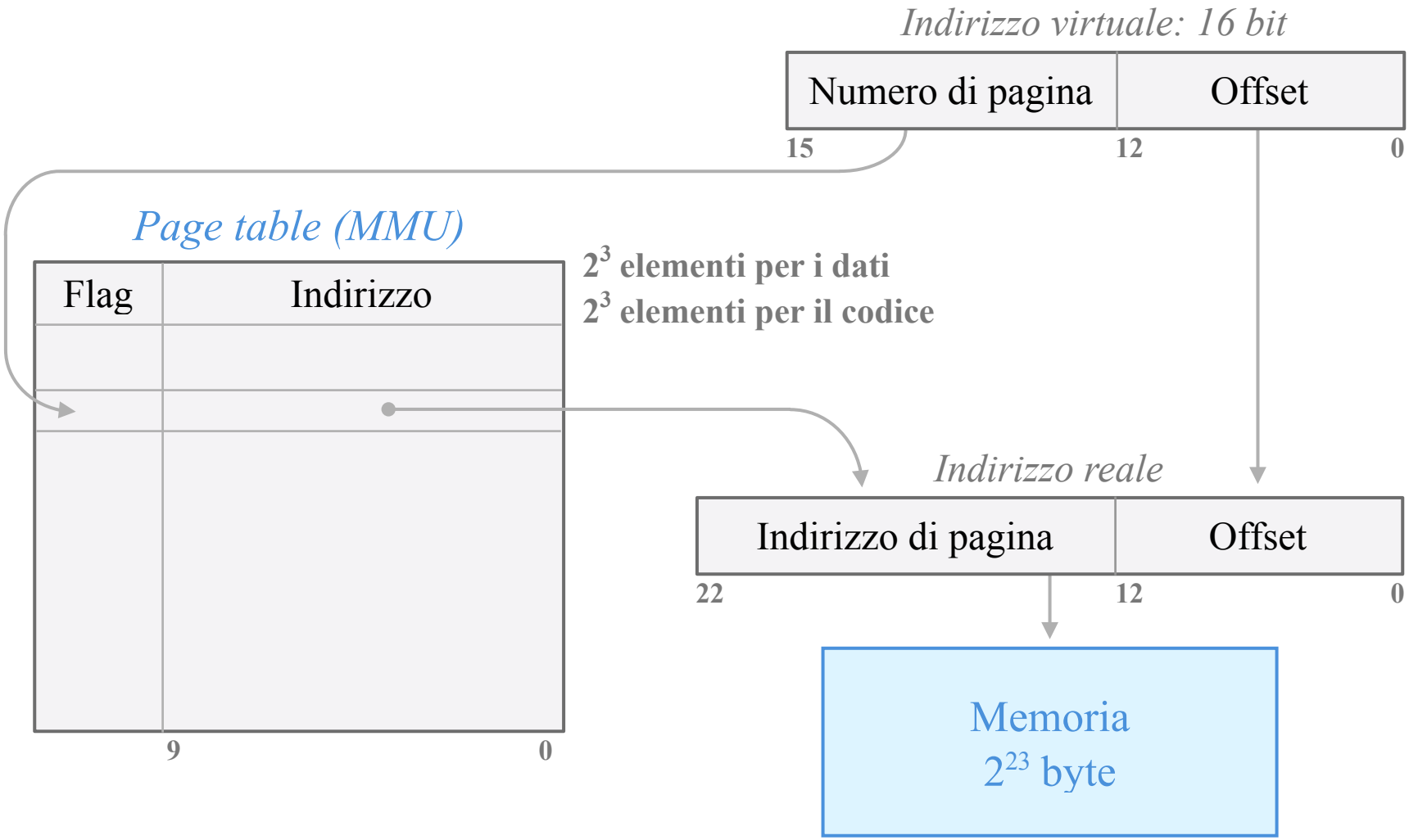
Traduzione con MMU



Esempio di paginazione con MMU PDP-11

- Sul PDP-11 ogni processo aveva a disposizione fino a 64 Kbyte per i dati e altrettanti per le istruzioni.
  - Potevano esserci fino a 4 Mbyte di memoria reale.
- Lo spazio di indirizzamento era diviso in pagine da 8 Kbyte.
  - Vi erano 16 registri in tutto (8 per i dati, 8 per le istruzioni) per la paginazione.
  - Il context switch implicava l'aggiornamento di questi 16 registri.

Paginazione sul PDP-11

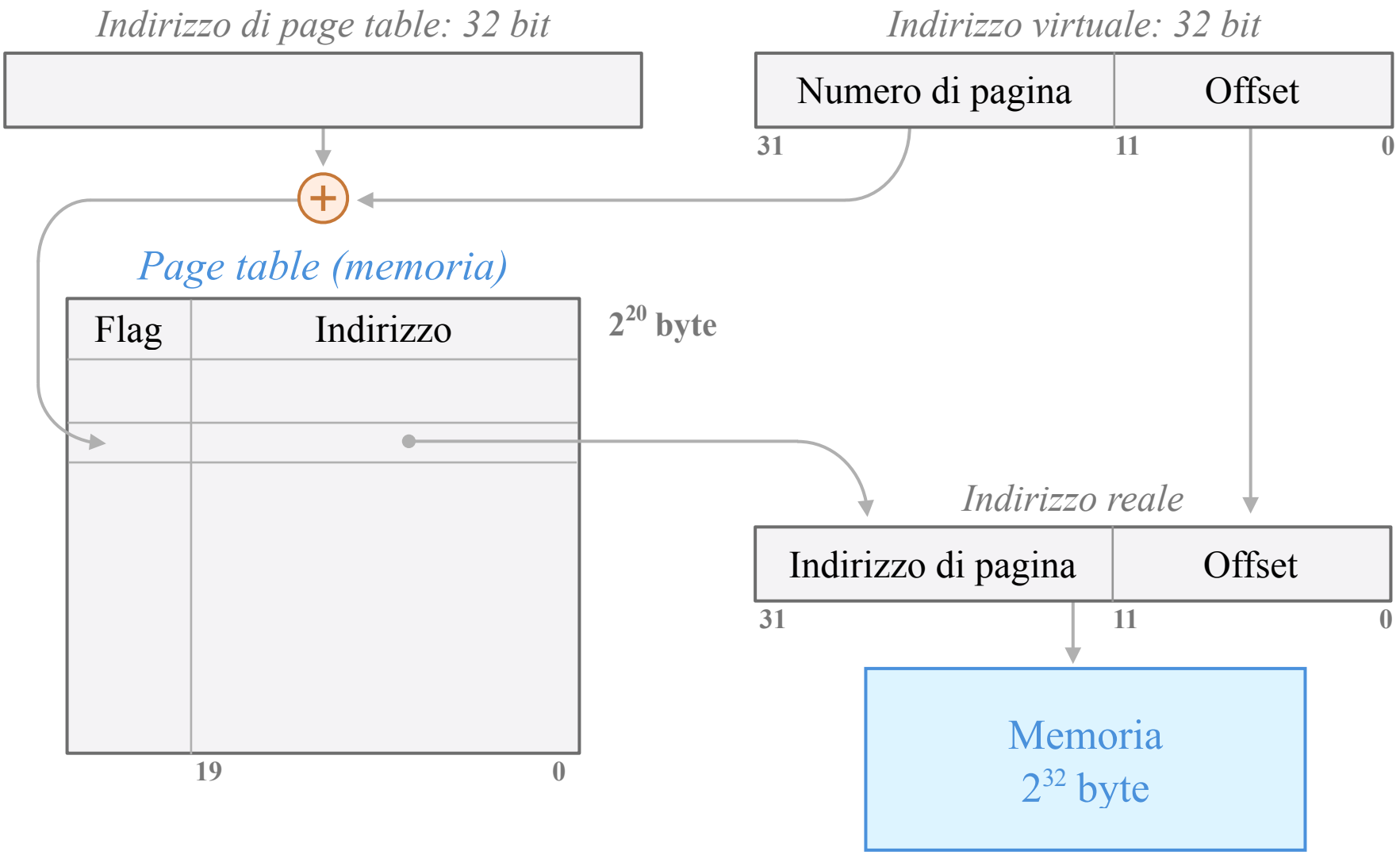


Page table in memoria

- Per disporre di spazi di indirizzamento maggiori è necessario tenere la page table in memoria.
  - Un apposito registro contiene l'indirizzo della page table.
  - Ogni accesso virtuale alla memoria richiede **due accessi fisici**:
    - uno per leggere l'entry di page table;
    - l'altro per l'accesso vero e proprio;

    quindi il tempo di accesso aumenta.
  - A ogni context switch basta modificare il registro di indirizzamento della page table.

Page table in memoria



Esempio di page table in memoria VAX

- Sul VAX ogni processo poteva indirizzare 1 Gbyte per dati e istruzioni e altrettanto per lo stack.
- Lo spazio di indirizzamento era diviso in pagine da 512 byte.
  - Troppo piccole per le macchine di oggi.
  - La page table di ogni processo poteva arrivare a 8 Mbyte.
  - Le page table stesse erano tenute nel segmento di sistema e potevano essere scaricate.

I segmenti sul VAX

- Lo spazio di indirizzamento era diviso in 4 segmenti, in base ai primi 2 bit:
  - **00** codice e dati;
  - **01** stack;
  - **10** sistema;
  - **11** riservato.
- Il segmento del sistema era parte dello spazio indirizzi di ogni processo, ma inaccessibile in modo utente.
  - La relativa page table non veniva mai scaricata.

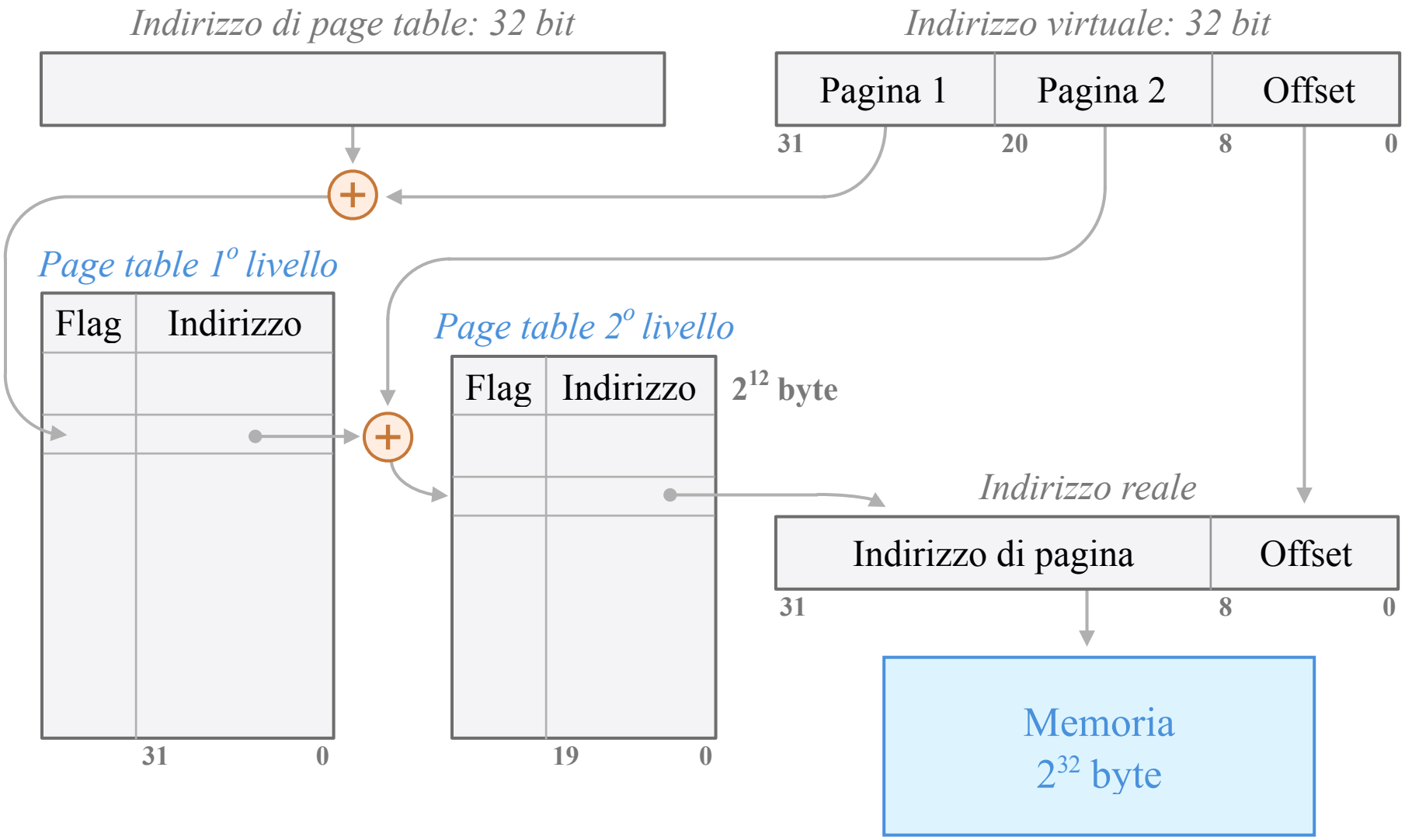
Paginazione a più livelli (1)

- Per molti processi servono poche pagine, quindi allocare grandi page table è uno spreco.
- Talvolta la page table sarebbe tanto grande da costituire un costo inaccettabile.
- Si ricorre allora a una page table a più livelli.
  - Possono servire 3 o più accessi alla memoria reale per ogni accesso virtuale.

Paginazione a più livelli (2)

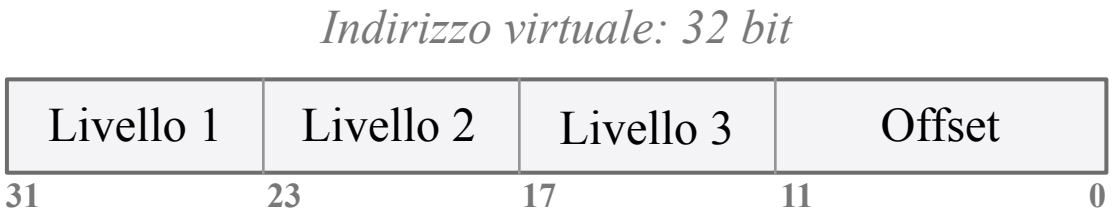
- L'indirizzo viene suddiviso in 3 o più parti:
  - le prime sono indici di pagina a due livelli;
  - l'ultima costituisce l'offset;
  - ogni page table contiene gli indirizzi di inizio di quelle del livello successivo
- Viene allocata comunque una page table di primo livello per ogni processo.
  - Ha dimensioni contenute.
- Vengono allocate solo le pagine dei successivi livelli realmente necessarie.

Page table a 2 livelli



Esempio di paginazione a 3 livelli SPARC

- Spazio di indirizzamento virtuale di 32 bit, diviso in pagine da 4 Kbyte.
  - Primo livello: 8 bit, tabelle da 512 elementi.
  - Secondo livello: 6 bit, tabelle da 64 elementi.
  - Terzo livello: 6 bit, tabelle da 64 elementi.



Esempio di paginazione a più livelli Motorola 68030

- Spazio di indirizzamento virtuale di 32 bit.
- Un registro permette di programmare:
  - dimensione delle pagine, da 256 byte a 32 Kbyte;
  - numero di livelli, da 0 a 4;
  - numero di bit per livello.

Page table invertita

- Page table normale:
  - tabella contenente **indirizzi di pagina reali, indicizzata dal numero di pagina virtuale**.
- Page table invertita:
  - tabella contenente **indirizzi di pagina virtuali, indicizzata dal numero di pagina reale**.

Page table invertita

Flag	Indirizzo virtuale
•	•
•	•
•	•

Page frame 0

Page frame 1

Page frame 2

•
•
•

Uso di page table invertita

- A ogni accesso, l'indirizzo di pagina virtuale viene cercato nella tabella:
  - se trovato, **l'indice** fornisce **l'indirizzo di pagina reale**;
  - altrimenti si ha un **page fault**.
- La ricerca nella tabella deve essere molto veloce.
  - Viene effettuata da hardware dedicato.
  - Tecniche di hashing permettono di ridurre il numero di accessi in media a meno di 2.

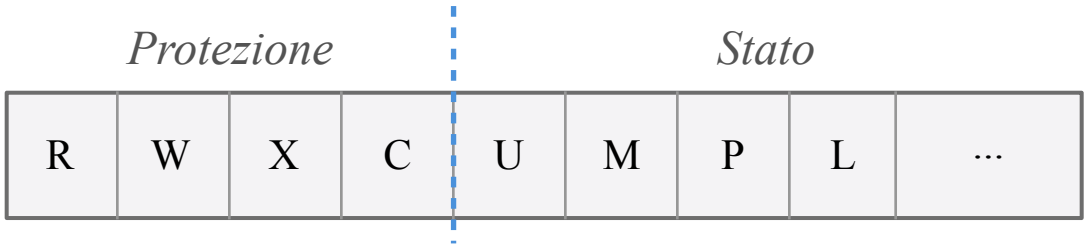
Vantaggi della page table invertita

- La tabella è **unica** nel sistema.
  - Per ogni processo vanno solo conservate le informazioni necessarie a reperire le pagine sul disco.
- La tabella ha dimensioni **minori**.
  - Dipende dallo **spazio di indirizzamento reale**.

Flag nella page table

- Si dividono in due categorie.
  - Flag di protezione:
    - scritti dal sistema;
    - utilizzati dall'hardware per verificare gli accessi.
  - Flag di stato:
    - azzerati dal sistema al caricamento della pagina;
    - scritti dall'hardware;
    - utilizzati dall'algoritmo di paginazione del sistema.
- I dettagli variano da macchina a macchina.

Flag nella page table



Bit	*Vale* **1** *quando*
R	ammessa lettura
W	ammessa scrittura
X	ammessa esecuzione
C	abilitato l'uso della cache
U	la pagina è stata usata
M	a pagina è stata modificata
P	presente in memoria
L	la pagina non può essere scaricata su disco

Flag nella page table

- Il sistema può gestire i flag di stato al posto dell'hardware.
  - P. es., se il flag U (M) non è gestito automaticamente, il sistema può:
    - marcare le pagine inaccessibili (read only),
    - mettere il flag a 1 al primo page fault provocato da un tentativo di accesso (scrittura), poi marcare la pagina come accessibile (scrivibile) e far ripetere l'accesso.

Memoria condivisa (1)

- La paginazione permette di far condividere a più processi una pagina in modo molto semplice.
  - La stessa pagina fisica viene descritta nella page table dei vari processi, a indirizzi virtuali **indipendenti tra loro**.

Memoria condivisa (2)

- In fase di caricamento/scaricamento e deallocazione il sistema deve tener conto degli utilizzi “multipli” di una pagina.
  - Una pagina condivisa dev’essere eliminata solo quando l’ultimo processo che la usa l’ha liberata o è morto.
  - Scaricare una pagina poco usata da ogni singolo processo, ma complessivamente molto usata, può aumentare il numero di page fault.

Vantaggi della condivisione di pagine

- Permette di **evitare di caricare copie multiple del codice** di un'applicazione che più utenti eseguono simultaneamente.
  - Spesso si riesce a evitare di creare una copia di una pagina, anche contenente dati, che serve a due processi.
- Può essere anche utilizzato per **scambiare dati tra processi**.

Copy on write

- Serve a evitare la duplicazione effettiva di pagine che servono a due processi
- La stessa pagina fisica viene riferita nelle page table dei due processi.
  - La pagina viene marcata come read-only.
- Alla prima scrittura, il meccanismo di gestione del page fault:
  - copia effettivamente la pagina;
  - modifica le protezioni, abilitando la scrittura;
  - aggiorna le page table.
- In assenza di scritture ne esiste una copia unica.

Ripetitività delle traduzioni (1)

- La traduzione da indirizzo virtuale a fisico è onerosa e può richiedere più accessi fisici alla memoria.
- I programmi tendono a effettuare numerosi accessi a locazioni di memoria vicine:
  - codice della funzione corrente;
  - variabili locali della funzione corrente;
  - variabili di uso frequente.

Ripetitività delle traduzioni (2)

- Le stesse traduzioni di indirizzo tendono a ripresentarsi a breve distanza.
- Per evitare di ripeterle si ricorre ai **Translation Lookaside Buffer** (**TLB**).
  - Sono una piccola memoria indirizzabile per contenuto, interna alla CPU.

TLB

- Contengono gli ultimi indirizzi di pagina virtuali tradotti, con i corrispondenti indirizzi reali e i flag di protezione e validità.
- Rappresentano una sorta di **cache delle traduzioni di indirizzi**.
- Come le cache **possono essere a due livelli**:
  - Un primo livello veloce, con pochi TLB (da 16 a 64);
  - Un secondo livello più lento di capacità maggiore (fino a un migliaio).

TLB

Numero di pagina	V	Indirizzo di pagina	Flag
•	•	•	•
•	•	•	•
•	•	•	•
20 bit	1 bit	20 bit	7 bit

La tabella contiene un piccolo numero di elementi.

Il bit V indica gli elementi validi.

Ricerca nei TLB

- Ogni numero di pagina viene ricercato nei TLB (in parallelo), prima di iniziare la traduzione.
- Se la ricerca ha successo, l'indirizzo è disponibile quasi immediatamente (di solito con 1 ciclo di ritardo), altrimenti la traduzione viene effettuata e l'indirizzo ottenuto memorizzato in un TLB.

Gestione dei TLB

- Di solito gestiti dall'hardware con tecnica LRU.
  - La macchina li carica e pone a 1 il bit V.
- Il sistema operativo deve invalidarli tutti se avviene un context switch o se modifica l'allocazione delle pagine in memoria.
  - Si utilizzano istruzioni apposite.

Uso esasperato dei TLB: MIPS 2000

- Il MIPS 2000 non aveva un meccanismo hardware per la traduzione da indirizzo virtuale a reale, ma ben 64 TLB.
  - Il chip diventa più semplice.
- In caso di fallimento della ricerca nei TLB viene generato un trap.
  - Il sistema deve eseguire la traduzione via software e caricare un TLB.
  - Simulazioni avevano mostrato che è abbastanza poco frequente e non penalizza le prestazioni.

Rimpiazzamento delle pagine

- Ogni volta che si carica una pagina è necessario sceglierne una da sovrascrivere.
  - Se la pagina è stata modificata, è necessario anche aggiornare la copia su disco.
- Un algoritmo ideale dovrebbe **minimizzare il numero di accessi a disco**.
  - Il sistema non può sapere **in anticipo** quali pagine saranno necessarie a breve termine, quindi compie **scelte euristiche**.
  - Il minimo possibile può essere calcolato solo a **posteriori**, per effettuare valutazioni.

Scelta delle pagine

- I vari algoritmi cercano di scartare pagine che non serviranno per un tempo lungo.
  - Sfruttano la località dei riferimenti.
- Gli algoritmi vengono confrontati tramite:
  - simulazione;
  - misure in sistemi reali.

Algoritmi a stack

- Si dicono "**a stack**" gli algoritmi che hanno la proprietà che, a parità di sequenza di accessi, l'insieme di pagine conservate in memoria con n pagine fisiche a disposizione è un sottoinsieme di quelle conservate avendone a disposizione $n + 1$.
- In altri termini, aumentando il numero di pagine fisiche, il numero di page fault **non aumenta mai**.
  - Non è detto però che diminuisca.

Algoritmi di scelta delle pagine

- NRU: Not Recently Used.
- FIFO: First In First Out.
  - FIFO “seconda chance”.
  - FIFO “a orologio”.
- LRU: Least Recently Used.
  - NFU: Not Frequently Used.
  - NFU con “invecchiamento”.

Algoritmo NRU: l'idea

- Ipotesi di partenza: se una pagine non è stata usata di recente, probabilmente non servirà per un certo tempo.
- Si cerca di scartare le pagine inutilizzate da più tempo.

Algoritmo NRU in dettaglio

- Utilizza i flag U e M.
- Il sistema azzera periodicamente il flag U.
- Si suddividono le pagine in 4 categorie:
  - 0: non usata, non modificata;
  - 1: non usata, modificata;
  - 2: usata, non modificata;
  - 3: usata, modificata.
- Si sceglie a caso una pagina dalla più bassa categoria possibile.

Vantaggi dell'algoritmo NRU

- Semplice.
- Non è indispensabile il supporto hardware.
  - Se disponibile, migliora comunque le prestazioni.
- Buon comportamento medio.

Algoritmo FIFO

- Si ordinano le pagine in ordine di caricamento.
  - Di solito si usa una lista di descrittori.
- Si scarta la pagina presente in memoria da più tempo.

Caratteristiche dell'algoritmo FIFO

- Semplice.
- Non serve supporto hardware.
- Una pagina presente da molto in memoria potrebbe però essere frequentemente utilizzata.
  - Potenzialmente **inefficiente**.
  - Raramente utilizzato in forma pura.

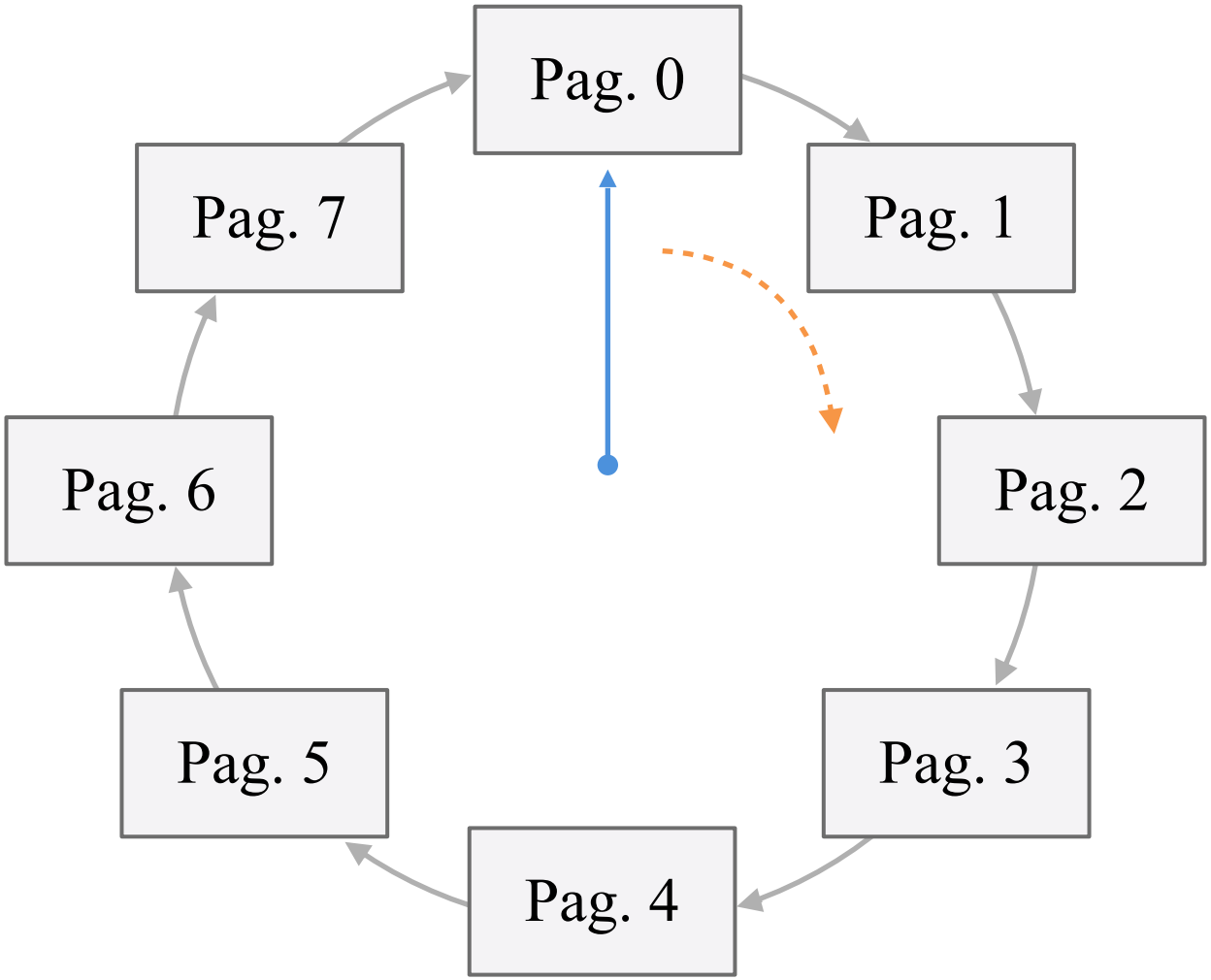
FIFO “seconda chance”

- Durante la ricerca delle pagine da scartare, quelle con il flag U a 1 vengono ricollocate in coda alla lista, come se fossero state appena caricate, azzerando il flag U.
- Le pagine col flag U a 0 vengono scartate.
- Elimina il principale problema dell’algoritmo FIFO puro.
  - Una pagina frequentemente usata non viene scartata.

FIFO “a orologio”

- Equivale al FIFO “seconda chance”, ma evita movimenti di elementi nelle liste.
- La lista delle pagine è circolare.
- Un puntatore indica il prossimo descrittore di pagina da esaminare e viene fatto avanzare ogni volta che una pagina è esaminata:
  - se il flag U è 1, viene azzerato;
  - se è zero, la pagina viene rimpiazzata.

Schema di FIFO “a orologio”



Anomalia di Belady

- Normalmente all'aumentare del numero di pagine di memoria fisica disponibili, il numero di page fault provocati da una sequenza di accessi fissata **diminuisce**.
- Con l'algoritmo FIFO, per sequenze particolari di riferimenti, **può aumentare**.
  - L'algoritmo FIFO non è a stack.
- Questo comportamento anomalo ha interesse teorico, ma in pratica le sequenze che lo producono sono rarissime.

Esempio di anomalia di Belady (1)

- Consideriamo 5 pagine, riferite nell'ordine:

  0 1 2 3 0 1 4 0 1 2 3 4.

- Con 3 pagine fisiche disponibili sono generati 9 page fault.

Richiesta	0	1	2	3	0	1	4	0	1	2	3	4
Page fault	X	X	X	X	X	X	X				X	
Piu recente	0	1	2	3	0	1	4	4	4	2	3	3
…		0	1	2	3	0	1	1	1	4	2	2
Piu vecchia			0	1	2	3	0	0	0	1	4	4

Esempio di anomalia di Belady (2)

- Con 4 pagine disponibili sono generati 10 page fault.

Richiesta	0	1	2	3	0	1	4	0	1	2	3	4
Page fault	X	X	X	X			X	X	X	X	X	X
Piu recente	0	1	2	3	3	3	4	0	1	2	3	4
…		0	1	2	2	2	3	4	0	1	2	3
…			0	1	1	1	2	3	4	0	1	2
Piu vecchia				0	0	0	1	2	3	4	0	1

Algoritmo LRU

- Si utilizza il passato come approssimazione del futuro.
  - Come NRU, ma cercando una maggior precisione.
- Si ordinano le pagine in ordine di accesso.
- Si scarta la pagina alla quale non si accede da più tempo.

Caratteristiche dell'algoritmo LRU

- Tipico algoritmo **a stack**.
- Serve supporto hardware, che aggiorni un numero progressivo (conservato nella page table) a ogni riferimento in memoria.
- La ricerca del valore minimo richiede comunque una lunga ricerca in memoria.
- Minimizza il numero di page fault in media.

Varianti sul tema LRU (1)

- Se l'hardware non aggiorna un contatore per ogni accesso, si può ottenere un'approssimazione software, con granularità inferiore.
  - Si possono determinare le pagine utilizzate tra due aggiornamenti dei flag.

Varianti sul tema LRU (2)

- Si utilizza un contatore per ogni pagina, tenuto nella page table.
  - Periodicamente si modificano i contatori delle pagine utilizzate.
  - Si cercano le pagine col flag U a 1 e si azzera il flag.
- Può essere necessario molto lavoro per esaminare tutta la page table.
  - Accettabile se le pagine sono poche.

Algoritmo NFU

- Periodicamente i contatori delle pagine utilizzate vengono incrementati di 1.
  - L'intervallo di tempo può variare da decimi di secondo a vari secondi.
- Si scarta la pagina col valore di contatore più basso.

Caratteristiche dell'algoritmo NFU

- Non richiede supporto hardware.
- Non dimentica nulla: una pagina con contatore elevato resta per sempre in memoria, anche se non più necessaria.
- Raramente utilizzato in forma pura.

NFU con “invecchiamento”

- Periodicamente i contatori vengono dimezzati; a quelli delle pagine col flag U a 1 viene sommato un valore elevato.
  - In pratica si effettua uno shift a destra ed eventualmente un or con un bit a 1 nella posizione più a sinistra.
  - I contatori delle pagine non più utilizzate scendono rapidamente a zero.
- Si scarta la pagina col valore di contatore più basso.

Caratteristiche dell'algoritmo NFU con invecchiamento

- Non richiede supporto hardware.
- Le pagine non utilizzate da molto tempo "invecchiano", riducendo progressivamente il valore del loro contatore.
- Ottime prestazioni.

Lock di pagine

- Alcune pagine devono essere bloccate in memoria e non scaricate.
  - Es.: una pagina nella quale una periferica sta trasferendo dati.
- Il sistema di solito usa un flag per marcare tali pagine.
- Gli algoritmi di scelta delle pagine da scartare ignorano le pagine marcate.

Algoritmi di paginazione locali

- Si dicono “locali” gli algoritmi che caricano una pagina sempre al posto di una **dello stesso processo**.
  - Ricercano la pagina da scartare tra un numero ridotto di pagine, quindi sono più veloci.
  - Lasciano in memoria pagine di processi inattivi per lunghi periodi.
  - E’ difficile stabilire quante pagine fisiche assegnare a ogni processo.

Algoritmi di paginazione globali

- Si dicono “globali” gli algoritmi che scelgono la pagina da scartare tra **tutte le pagine disponibili**.
  - Evitano di lasciare pagine inutilizzate in memoria troppo a lungo.
  - Un processo che generi un gran numero di page fault ha effetti negativi sulle prestazioni dell’intero sistema.
  - In genere più complessi, ma danno prestazioni migliori.

Dimensione delle pagine

- E' frutto di un compromesso.
  - Se le pagine sono troppo grandi:
    - aumenta lo spreco per le pagine parzialmente utilizzate: in media **mezza pagina per segmento**.
  - Se le pagine sono troppo piccole:
    - aumenta la dimensione della page table (può diventare eccessiva per una MMU);
    - Può migliorare l'efficienza degli accessi al disco.
- Le dimensioni più comuni sono da 1 a 16 Kbyte.
  - Su alcune macchine sono programmabili.

Dimensioni ottimali

- Con s byte in media per processo, pagine di p byte ed e byte per elemento di page table, lo spreco medio è

$$\frac{es}{p} + \frac{p}{2}$$

- Derivando rispetto a p si trova che il minimo si ha per

$$-\frac{es}{p^2} + \frac{1}{2} = 0 \qquad \text{da cui} \qquad p = \sqrt{2es}$$

- Per esempio, con $s = 1$ Mbyte, $e = 4$ byte, lo spreco è minimo per $p = 228$ byte.

Paginazione

- In genere un processo, dopo un gran numero di page fault iniziali, tende a usare relativamente poche pagine: il **working set**.

Working set minimo

- Per **evitare un deadlock**, deve essere garantito che ogni processo attivo abbia un numero minimo di pagine in memoria.
  - Di solito pari al numero massimo di operandi in memoria di un'istruzione più uno.
  - Deve essere garantita la possibilità di eseguire almeno una istruzione.
  - Vi sono complicazioni se un'istruzione o un'operando possono estendersi a cavallo di due pagine o se gli indirizzi possono essere indiretti.

Strategie di caricamento

- Comunemente le pagine sono caricate quando necessario, dopo un page fault.
  - Strategia detta **demand paging.**
- Alcuni sistemi non rendono running un processo prima di aver comunque caricato il working set in memoria.
  - Strategia detta **prepaging**; serve a ridurre i page fault.
  - Bisogna però identificare il working set.

Thrashing (1)

- Si verifica quando la paginazione è eccessiva e il sistema spende troppo tempo in I/O.
  - I processi riescono a eseguire poche istruzioni tra un page fault e l'altro.
  - Gran parte del tempo di CPU viene speso nella gestione dei fault.

Thrashing (2)

- Può essere causato da una memoria sottodimensionata o da eccesso di processi.
- Un processo può causare da solo un page fault ogni poche istruzioni.
  - Piuttosto infrequente, ma può essere provocato da alcuni algoritmi:
    - operazioni su grandi strutture dati (vettori o matrici) con accesso non sequenziale;
      - grandi matrici, con accessi per colonne.
    - alcuni algoritmi di garbage collection della memoria dinamica.

Gestione della memoria

- Per evitare il trashing, il sistema può contare il numero di page fault provocati da ogni processo in un'unità di tempo.
  - Si possono allocare più pagine ai processi che generano più page fault.
  - Si possono bloccare alcuni processi, per ridurre il carico del sistema.

Algoritmo PFF

- Si basa sul calcolo della frequenza di page fault (page fault frequency).
  - Un ibrido tra locali e globali.
  - La pagina da eliminare viene scelta da un algoritmo locale.
  - Il numero di pagine assegnate a ogni processo viene ricalcolato periodicamente, misurando la frequenza di page fault.
  - I processi che ne causano molti ricevono più pagine.

Paging daemon (1)

- E' preferibile avere sempre a disposizione alcune pagine libere, piuttosto che doverne liberare una al momento di un page fault.
- In molti sistemi vi è un processo in background, detto paging daemon, che periodicamente controlla il numero di pagine libere.
  - Se sono poche, ne libera alcune eseguendo l'algoritmo di rimpiazzamento.

Paging daemon (2)

- Oltre ad assicurare un minimo di pagine libere, anticipa le scritture su disco di quelle da scartare, garantendo che in ogni momento alcune pagine possano essere scartate senza doverle copiare su disco.
- Le pagine scartate, ma ancora fisicamente presenti in memoria, possono essere recuperate senza rileggerle, se vengono riferite.

Area di swap

- Parte di disco utilizzata per conservare copia delle pagine.
  - Può essere un file qualsiasi.
  - Più comunemente è una partizione del disco o un file preallocato, in modo da mantenerlo contiguo. L'area allocata per ogni processo non è necessariamente contigua.
- La pagina dev'essere un multiplo dell'unità di accesso a disco (settore), in modo che ogni settore appartenga a un'unica pagina.

Gestione dell'area di swap

- Un processo al lancio viene di solito caricato nell'area di swap, non in memoria.
- L'area di swap può essere:
  - preallocata per ogni processo, creando problemi se l'area dati deve espandersi;
  - allocata secondo necessità, con possibilità di overflow imprevisti.
- In caso di overflow è impossibile allocare memoria e creare processi.
  - E' necessario uccidere un processo.

Segmentazione

- Consiste nel suddividere la memoria assegnata a ogni processo in aree di dimensione variabile (es.: da 1K a 1 Gbyte).
- Uno o più per ogni parte del programma:
  - codice, dati statici, stack, heap.
- Gli indirizzi sono divisi in numero di segmento e offset.
  - Spesso tenuti in registri differenti.

Le origini della segmentazione

- Nasce come alternativa agli overlay.
- Utilizzata per **estendere lo spazio di indirizzamento** oltre i limiti naturali del singolo registro, mantenendo contenuta la dimensione del codice.
  - P. es, su macchine a 16 bit, usando un registro per il numero di segmento e uno per l'offset, si può passare dai 64K ai Gbyte.
    - Es.: macchine a 16 bit con indirizzamento a 20 (Intel 8086) o 23 (Z8000).

Caratteristiche della segmentazione

- Permette di ridurre le dimensioni del codice.
  - Molti indirizzi possono essere specificati solo in termini di offset.
  - Questa tecnica crea però notevoli complicazioni al programmatore.
  - Al diminuire del costo della memoria, gli svantaggi dovuti alla complessità di gestione superano i vantaggi.

Vantaggi della segmentazione (1)

- Ogni segmento può essere rilocato indipendentemente dagli altri.
- Ogni segmento può crescere indipendentemente senza problemi.
- Per il sistema allocare più parti piccole è più semplice che allocarne una grande e riduce gli sprechi.
- L'hardware di supporto necessario è semplice.

Vantaggi della segmentazione (2)

- Si minimizza lo spreco, in assenza di memoria virtuale.
  - La dimensione dei segmenti può essere scelta con granularità molto fine.
  - A volte anche in multipli di soli 16 byte (Intel 8086).

Vantaggi della segmentazione (3)

- Rende semplice la condivisione di memoria tra processi differenti.
  - Se la memoria è paginata, per implementare la condivisione si simula di solito la segmentazione, raggruppando in un segmento le pagine da condividere.

Traduzione degli indirizzi con memoria segmentata

- Richiede un supporto hardware.
- Per ogni processo viene allocata una segment table.
  - Il numero di segmento funge da **indice** e permette di accedere all'indirizzo reale di inizio del segmento e alle altre informazioni.

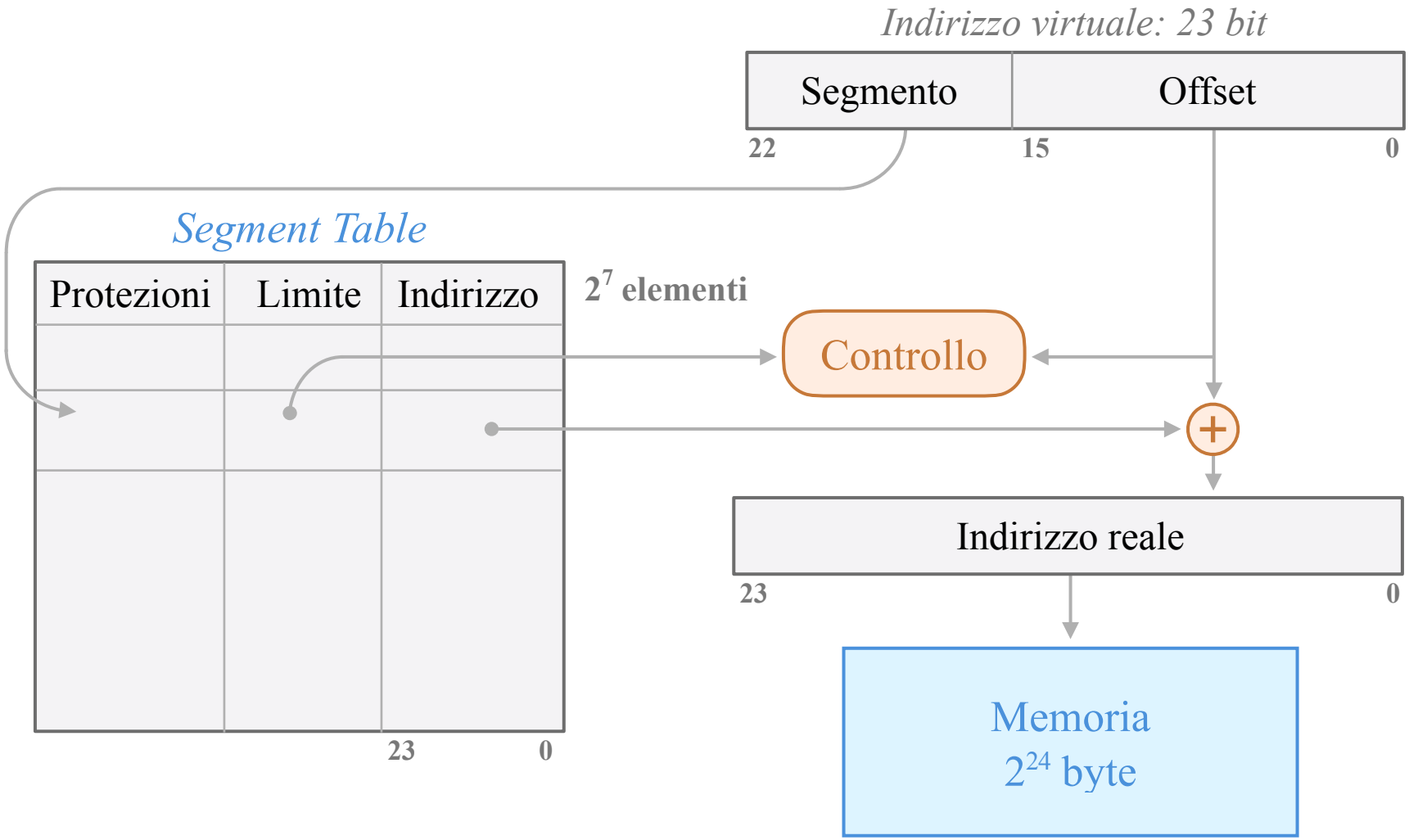
Accesso alla memoria segmentata

- Per ogni accesso la macchina:
  - suddivide l'indirizzo in numero di segmento e offset;
  - usa il numero di segmento come indice nella segment table;
  - legge il descrittore di pagina;
  - **somma** l'offset all'indirizzo reale del segmento;
  - verifica il rispetto del limite di segmento;
  - accede alla memoria reale o genera un trap (segment fault).

Segment table

- Se il numero di segmenti per processo è piccolo, si può tenere la segment table in una MMU o nella stessa CPU.
- Se il numero è elevato si ricorre alla memoria.
  - I segmenti attivi in ogni momento sono di solito pochi e i relativi elementi di segment table possono essere facilmente tenuti nella CPU. (es.: Intel 80386).
- Gestione simile a quella di una page table.

Traduzione di indirizzo segmentato Z8000

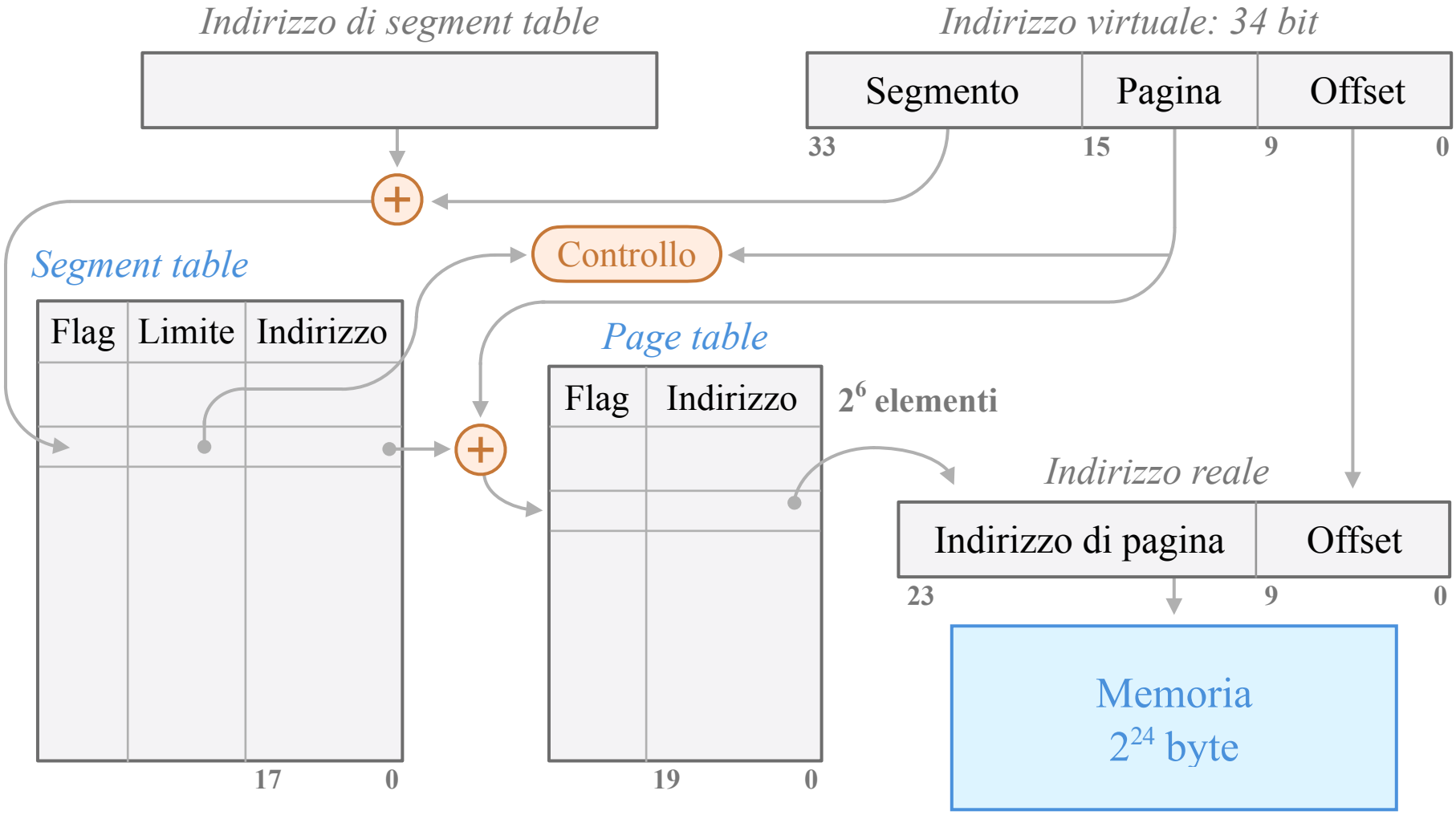


Memoria segmentata paginata

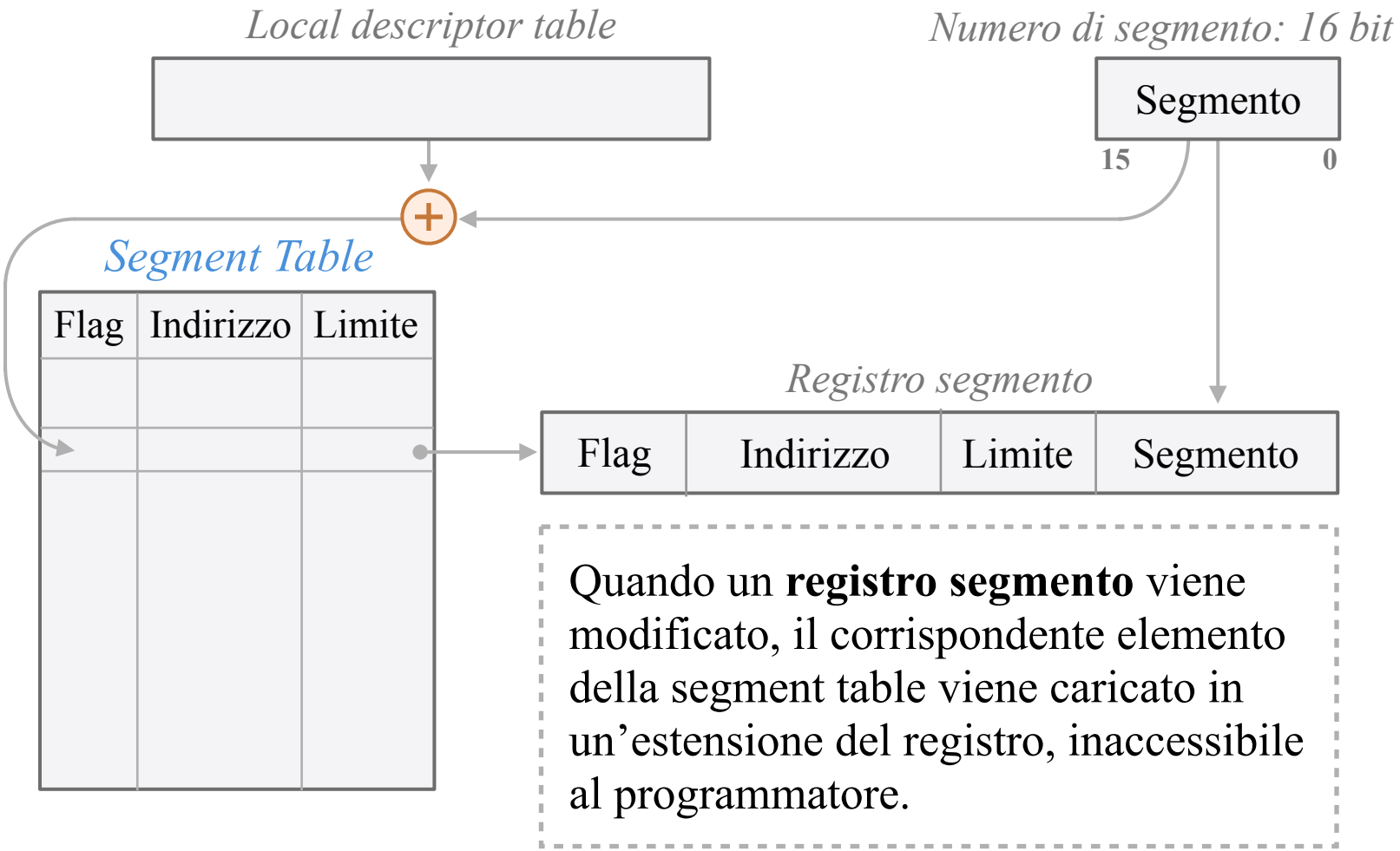
- Segmentazione e paginazione possono coesistere

  (es.: Honeywell 6000, Intel).
  - Inevitabile se si usa la memoria virtuale, perché lo swapping di interi segmenti sarebbe troppo oneroso.
- La paginazione viene applicata sugli indirizzi generati dalla segmentazione.
- Le protezioni sono a livello di segmento.
  - Più facili da gestire.
  - Meno ridondanza, minore occupazione.

Memoria segmentata paginata MULTICS

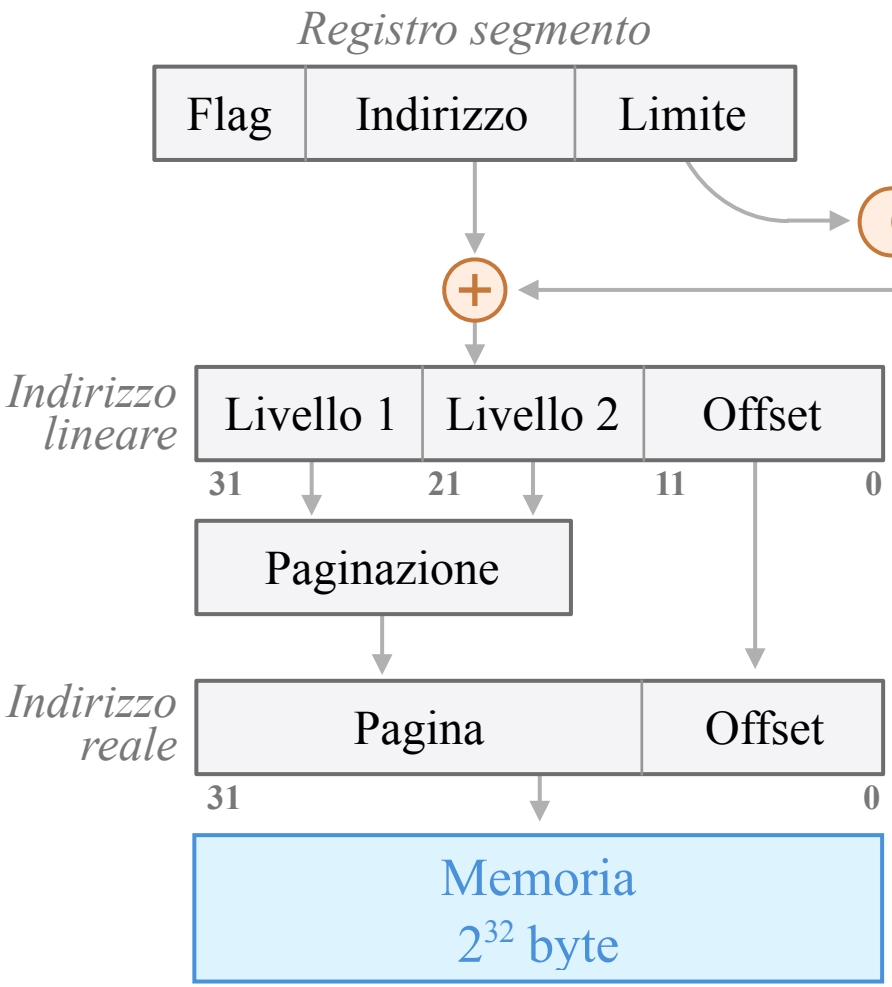


Memoria segmentata paginata Intel 80386 (1)



Quando un **registro segmento** viene modificato, il corrispondente elemento della segment table viene caricato in un'estensione del registro, inaccessibile al programmatore.

Memoria segmentata paginata Intel 80386 (2)



La paginazione può essere disabilitata, rendendo l'indirizzo lineare coincidente con quello reale.

Differenze tra segmentazione e paginazione (1)

- Dimensioni:
  - le pagine hanno **dimensione fissa**;
  - i segmenti hanno **dimensione variabile**.
- Numero; un processo tipicamente usa:
  - migliaia di pagine;
  - pochi segmenti (da 3 a un centinaio).
- Controllo:
  - una pagina, se accessibile, lo è **interamente**;
  - un segmento **ha un limite**, che va verificato.

Differenze tra segmentazione e paginazione (2)

- Traduzione dell'indirizzo; l'offset viene:
  - **concatenato** all'indirizzo reale della pagina;
  - **sommato** all'indirizzo reale del segmento.
- Spazio di indirizzamento virtuale:
  - lineare con la paginazione;
  - non lineare con la segmentazione.

Differenze tra segmentazione e paginazione (3)

- Frammentazione:
  - **solo interna** con la paginazione (spreco modesto);
    - Anche in presenza di segmentazione;
  - **solo esterna** con la segmentazione (spreco elevato).

Segmentazione e paginazione

- Unendo le due tecniche si sommano i vantaggi e si elimina lo svantaggio della frammentazione esterna.
  - La granularità della segmentazione è di solito uguale alla dimensione di una pagina.
  - Si usa una page table per ogni segmento.
    - Occupazione ridotta come nella paginazione a 2 livelli.
- Lo svantaggio è dato dalla traduzione degli indirizzi **più complessa** e quindi più lenta.