Gestione della memoria principale e virtuale

Obiettivi

- Spiegare analogie e differenze con la gestione della CPU
- Illustrare la differenza tra un indirizzo logico e un indirizzo fisico e le tecniche di rilocazione degli indirizzi
- Definire la memoria virtuale e descriverne i benefici
- Illustrare gli aspetti caratterizzanti la gestione della memoria
- Spiegare le strategie first-fit, best-fit e worst-fit per allocare memoria in modo contiguo
- Spiegare la differenza tra frammentazione interna ed esterna
- Spiegare come tradurre indirizzi logici in indirizzi fisici in un sistema segmentato e in uno paginato con TLB
- Illustrare come le pagine vengono caricate in memoria utilizzando la paginazione su richiesta
- Descrivere gli algoritmi per la sostituzione delle pagine e l'allocazione dei frame
- Descrivere i concetti di località e working set di un processo
- Spiegare cos'è il thrashing e perché si verifica
- Descrivere come possono essere organizzate le tabelle delle pagine

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Analogie con la gestione della CPU

- In genere i programmi in attesa di essere eseguiti risiedono su disco nella forma di file eseguibili e formano la cosidetta coda d'ingresso
  - *File eseguibile:* contiene, in formato binario e pronte per essere caricate in memoria, tutte le informazioni relative al programma da eseguire (immagine del processo)
- Per eseguire un programma bisogna
  - caricarlo in memoria principale e
  - attivare un processo che si occupi della sua esecuzione
- Analogamente con quanto accade con la CPU, *non è il processo a dover chiedere l'assegnazione* della memoria principale, perché ciò implicherebbe che esso è in esecuzione e, quindi, che il programma che sta eseguendo è già in memoria principale

Differenze con la gestione della CPU

- Diversamente dalla CPU, regioni diverse della memoria principale possono essere allocate contemporaneamente a processi diversi
  - Condivisione nello spazio: programmi o utenti diversi usano ciascuno una 'parte' della risorsa (es. memoria principale)
  - Condivisione nel tempo: programmi o utenti diversi usano la risorsa 'a turno' (es. CPU)
- Opportunità di condivisione del codice e dei dati
- Necessità di utilizzare meccanismi di protezione per impedire l'accesso non voluto a regioni di memoria del SO o di un processo da parte di altri processi

Differenze con la gestione della CPU

- *Una risorsa virtuale è solitamente una struttura dati* che rappresenta lo stato della risorsa fisica quando questa non è assegnata al processo cui è associata la risorsa virtuale
- Es. una CPU virtuale è una struttura dati che contiene i valori dei registri della CPU ed è allocata in memoria principale (fa parte del PCB)
- Così non è per la memoria virtuale
  - Può anche essere allocata in una memoria diversa dalla memoria principale (es. area di swap del disco)

Gestione della memoria principale e virtuale

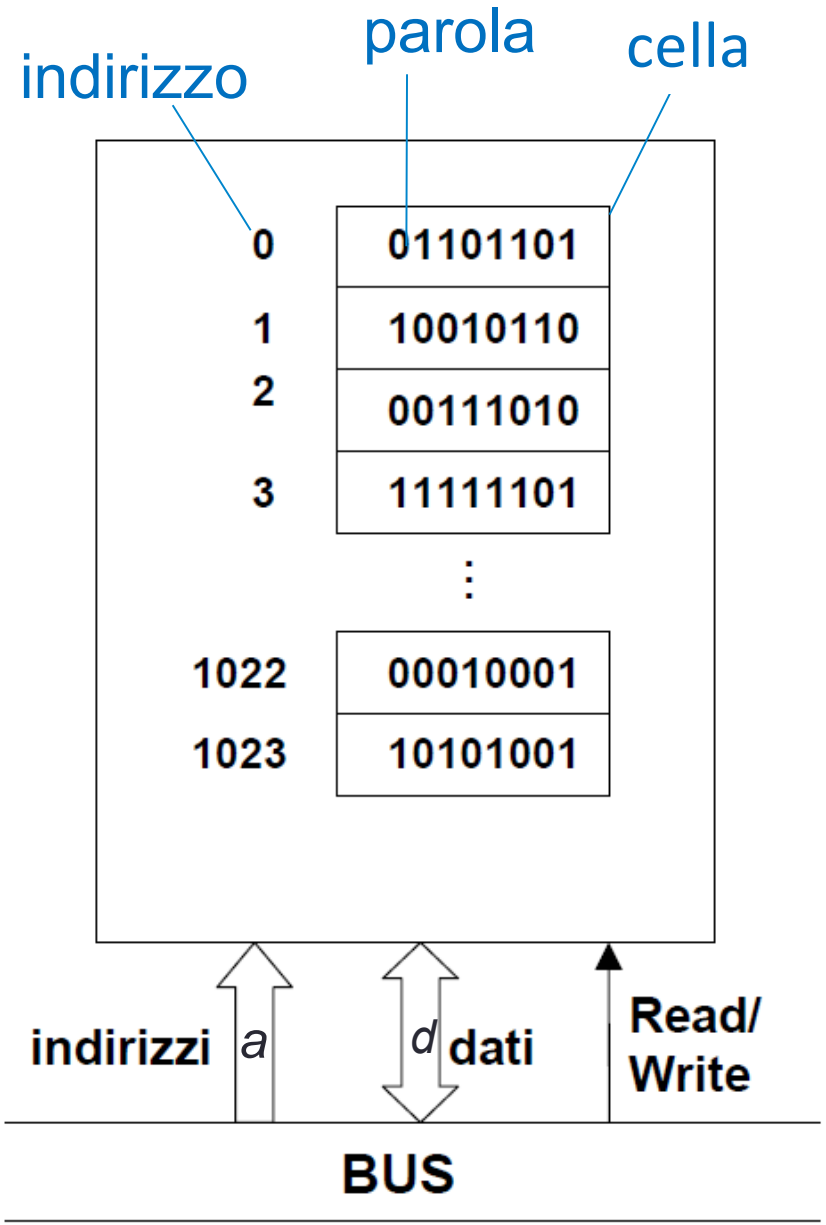
- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Hardware di base

- Memoria principale
- Registri della CPU per istruzioni e dati
- Cache
- Registri della CPU per protezione della memoria

Memoria principale

- Consiste in un grande vettore di celle, che contengono parole, ciascuna con il proprio indirizzo
- Una istruzione o un dato può occupare più celle consecutive
- Il contenuto delle celle non è riconoscibile
  - La memoria vede solo parole e indirizzi ma non sa come essi siano generati, e nemmeno se siano dati o istruzioni
- *d* è l'ampiezza della cella e quindi della parola di memoria
  - Tipicamente, d è multiplo del byte: 8 bit, 16 bit, 32 bit, 64 bit, 128 bit ...
- Spazio di indirizzamento = Max quantità di celle indirizzabili = 2^a
  - a è la lunghezza in bit degli indirizzi



Registri della CPU per istruzioni e dati

- La memoria principale e i registri incorporati nella CPU sono le sole aree di memorizzazione a cui la CPU può accedere direttamente
- Pertanto, qualsiasi istruzione in esecuzione, e tutti i dati utilizzati dalle istruzioni, devono risiedere in uno di questi dispositivi ad accesso diretto
- Dati ed istruzioni che non sono in memoria principale devono essere caricati prima che la CPU possa operare su di essi

Velocità relative di accesso alla memoria fisica

- I registri incorporati nella CPU sono accessibili, in genere, nell'arco di un ciclo del clock della CPU
  - I core di alcune CPU sono in grado di decodificare istruzioni ed eseguire semplici operazioni sui contenuti dei registri alla velocità di una o più operazioni per ciclo
- L'accesso alla memoria principale avviene tramite una transazione sul bus della memoria che può richiedere molti cicli del clock della CPU
- In tali casi, il processore entra necessariamente in stallo, poiché non dispone dei dati necessari per completare l'istruzione che sta eseguendo
- Questa situazione è intollerabile poiché gli accessi alla memoria sono frequenti

Memoria Cache

- Il rimedio consiste nell'interposizione di una memoria veloce, denominata cache, tra CPU e memoria principale
- La cache, in genere
  - è incorporata nel chip della CPU per permettere un accesso più rapido
  - è gestita direttamente dall'HW, senza alcun intervento del SO
- Se i dati necessari per l'esecuzione non sono nella cache lo stallo è inevitabile
  - HW recente implementa anche core multithread, in cui due (o più) thread HW sono assegnati ad ogni core cosicché durante uno stallo della memoria un core multithread può scambiare il thread HW in stallo con un altro thread HW

Registri della CPU per protezione della memoria

Memoria condivisa tra più processi ⇨ necessità di protezione

- Per garantire la corretta esecuzione delle operazioni dei processi bisogna isolare la memoria di processi diversi così da
  - proteggere l'area di memoria del SO dall'accesso da parte dei processi utente
  - nei sistemi multiutente, proteggere l'area di memoria di ogni processo utente dall'accesso non voluto da parte di altri processi
- La protezione deve essere messa in atto a livello HW perché il SO, per una questione di prestazioni, solitamente non interviene negli accessi della CPU alla memoria
- L'HW supporta questa protezione in diversi modi
  - Es. tramite il registro base e il registro limite

Gestione della memoria principale e virtuale

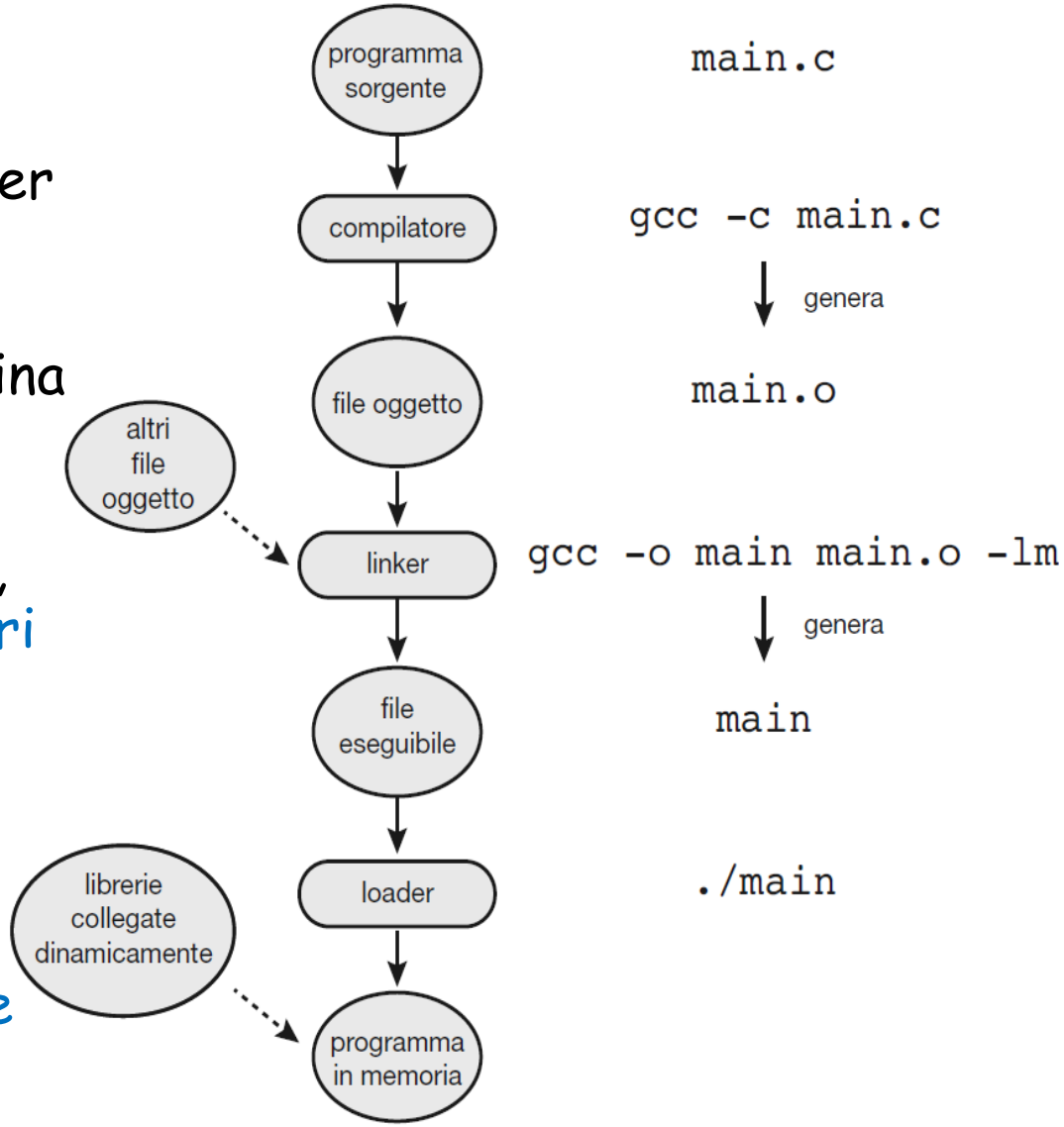
- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Preparazione di un programma utente per l'esecuzione

- Di solito, un programma risiede in memoria secondaria (es. disco) in forma di file binario eseguibile (es. a.out o prog.exe)
- Per diventare idoneo per l'esecuzione su (un core di) una CPU, un programma utente passa attraverso fasi diverse, alcune delle quali possono essere opzionali
- Alla fine, il programma viene caricato in memoria principale e inserito nel contesto di un processo per essere eseguito

Ruolo di Compilatore, Linker e Loader

- Il compilatore trasforma i file sorgenti in file oggetto, i quali sono rilocabili, cioè progettati per essere caricati in qualsiasi posizione della memoria fisica
- Successivamente, il linker combina i file oggetto rilocabili in un singolo file binario eseguibile
- Durante la fase di collegamento, possono essere inclusi anche altri file oggetto o alcune librerie, come la libreria standard del linguaggio C o la libreria matematica standard (flag `-lm`)
- Infine, il loader carica in memoria il file binario eseguibile



Esecuzione del loader

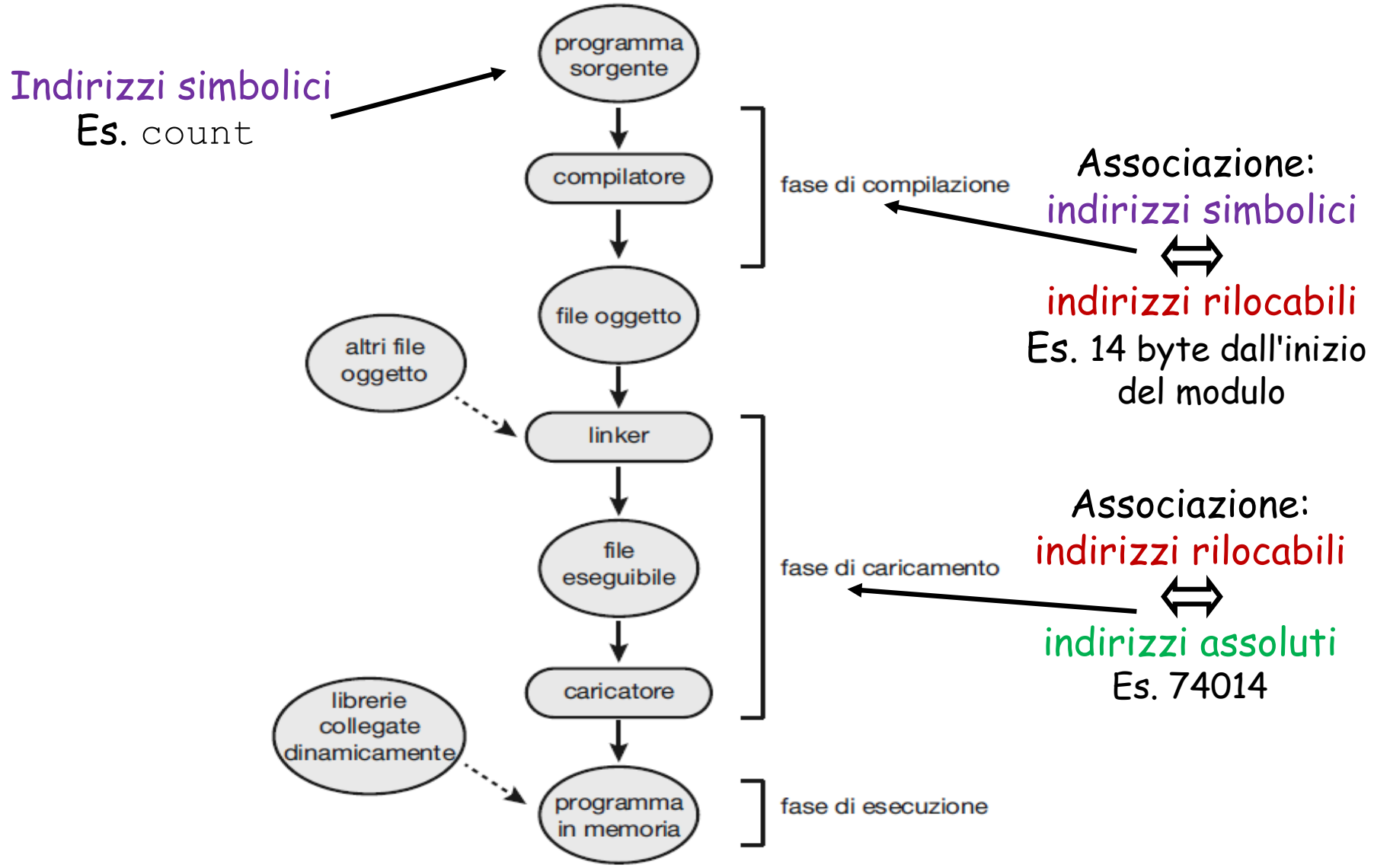
Per eseguire il loader, basta digitare il nome del file eseguibile sulla riga di comando

- Quando si immette il nome di un programma sulla riga di comando nei sistemi UNIX, ad esempio digitando `./main`
  - la shell
    - crea un nuovo processo utilizzando la system call `fork()`
    - quindi richiama il loader con la system call `exec()`, a cui passa come parametro il nome del file eseguibile corrispondente al programma
  - il loader carica in memoria il file eseguibile specificato utilizzando lo spazio di indirizzi del processo appena creato
- Quando si utilizza una GUI, cliccando due volte sull'icona associata al file eseguibile si richiama il loader utilizzando un meccanismo simile

Associazione degli indirizzi

- È una procedura che
  - assegna indirizzi di memoria definitivi alle componenti di un programma
  - risistema il codice e i dati nel programma secondo questi indirizzi in modo che durante l'esecuzione, ad esempio, il codice possa accedere alle sue variabili e richiamare le funzioni di una libreria
    - Infatti, la maggior parte dei sistemi consente ai processi utente di risiedere in qualsiasi area della memoria fisica
    - Pertanto, sebbene lo spazio degli indirizzi del calcolatore inizia all'indirizzo 0, il primo indirizzo di un processo utente non deve necessariamente essere 0
- L'associazione di istruzioni e dati agli indirizzi di memoria fisica può essere eseguita in qualsiasi fase del processo di preparazione di un programma per l'esecuzione

Fasi di preparazione di un programma utente per l'esecuzione



Associazione degli indirizzi

Può avvenire in una qualsiasi delle seguenti fasi:

SW

- Compilazione: se la locazione di memoria in cui va caricato un programma è nota a priori, il compilatore può generare codice statico (o assoluto)
  - Inconveniente: se la locazione cambia, il codice deve essere ricompilato
- Caricamento: se la locazione di memoria non è nota al momento della compilazione, il compilatore genera codice rilocabile
  - Il *caricatore* potrà tradurre gli indirizzi rilocabili in indirizzi assoluti ogni volta che il programma sarà (ri)caricato in memoria (momento in cui la locazione di memoria sarà nota)

HW

- Esecuzione: se un processo durante le varie fasi della sua esecuzione può essere spostato da un'area di memoria ad un'altra, si deve ritardare l'associazione degli indirizzi fino al momento dell'esecuzione
  - In questo caso è necessario un supporto HW dedicato (MMU) poiché la traduzione via SW sarebbe inefficiente

È la più usata

Associazione degli indirizzi

- Nel codice statico gli indirizzi nascono assoluti perché generati in fase di compilazione
  - Si parla di associazione statica degli indirizzi
- Nel codice staticamente rilocabile gli indirizzi nascono relativi e vengono trasformati in assoluti in fase di caricamento del programma in memoria
  - Si parla ancora di associazione statica degli indirizzi
- Nel codice dinamicamente rilocabile gli indirizzi nascono relativi e rimangono tali anche quando il programma viene caricato in memoria ed eseguito
  - La trasformazione di un indirizzo relativo in uno assoluto viene fatta durante l'esecuzione dell'istruzione che usa l'indirizzo
  - Si parla di associazione dinamica degli indirizzi

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

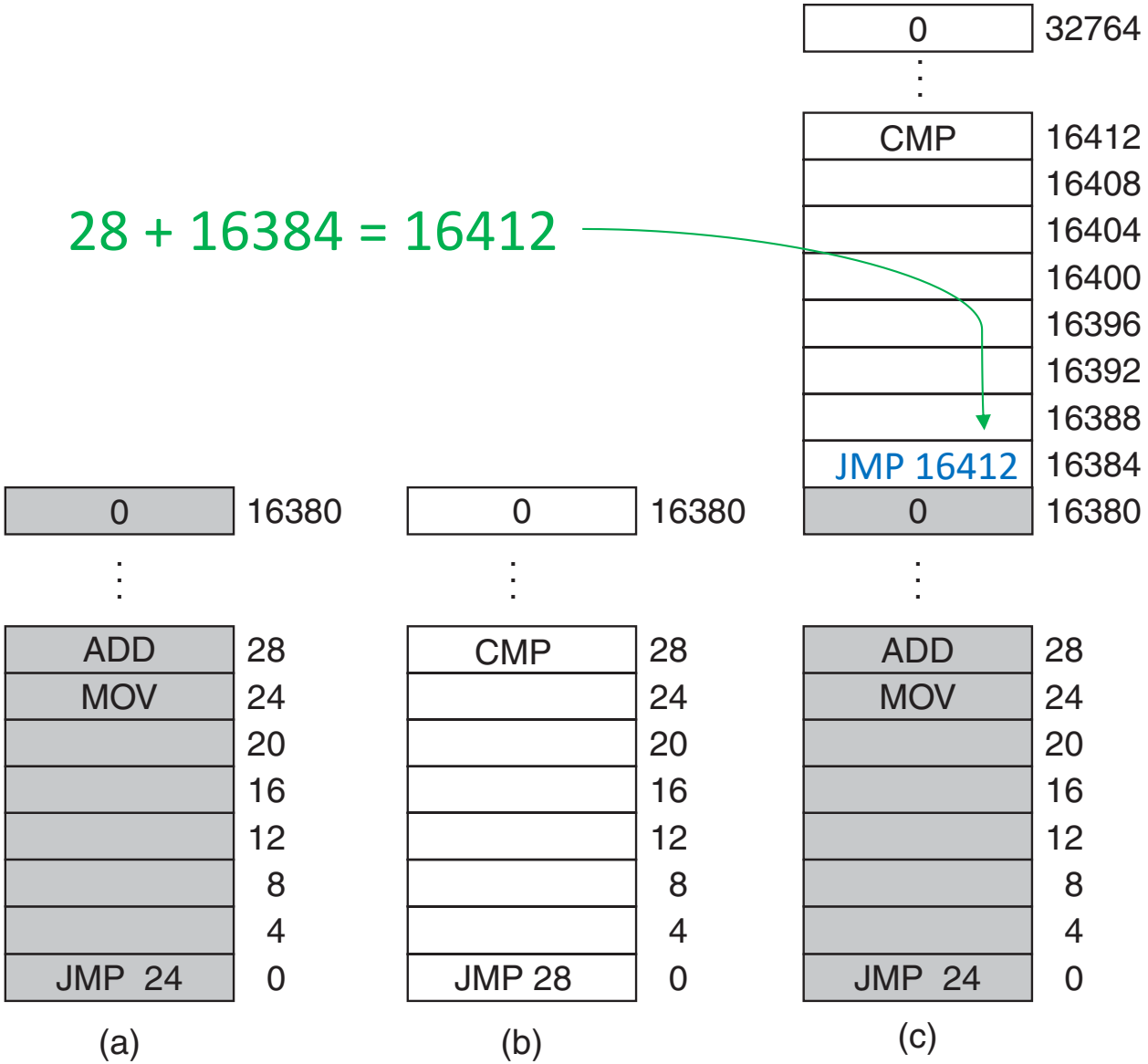
Spazi degli indirizzi logici e fisici

- In caso di associazione dinamica, abbiamo due tipologie di indirizzi
  - Indirizzo logico: indirizzo generato dalla CPU
  - Indirizzo fisico: indirizzo visto dall'unità di memoria, cioè caricato nel registro dell'indirizzo di memoria (Memory Address Register, MAR)
- Spazio degli indirizzi logici (o spazio di indirizzi logico): insieme di tutti gli indirizzi logici generati da un programma
- Spazio degli indirizzi fisici (o spazio di indirizzi fisico): insieme di tutti gli indirizzi fisici messi a disposizione dall'architettura HW

Rilocazione statica

- L'associazione statica degli indirizzi (effettuata nella fase di compilazione o in quella di caricamento)
  - genera indirizzi logici e fisici identici
  - pertanto, gli spazi degli indirizzi logici e fisici coincidono
- I processi utente hanno codice statico o rilocato staticamente cioè al momento del caricamento in memoria
  - Il SO risiede nell'area di memoria alta (indirizzi maggiori)
  - Il primo processo utente è allocato in memoria a partire dall'indirizzo 0; gli altri lo seguono sequenzialmente fino ad arrivare all'area dedicata al SO
  - Quando un processo è caricato, poiché in genere non sarà caricato a partire dall'indirizzo 0, lo si riloca in modo tale che possa essere eseguito nell'area di memoria dove è allocato (es. delimitata tramite registro base e limite)
- In pratica, al momento del caricamento, se necessario il SO modifica gli indirizzi riferiti dal processo sommando loro il valore contenuto nel registro base

Rilocazione statica



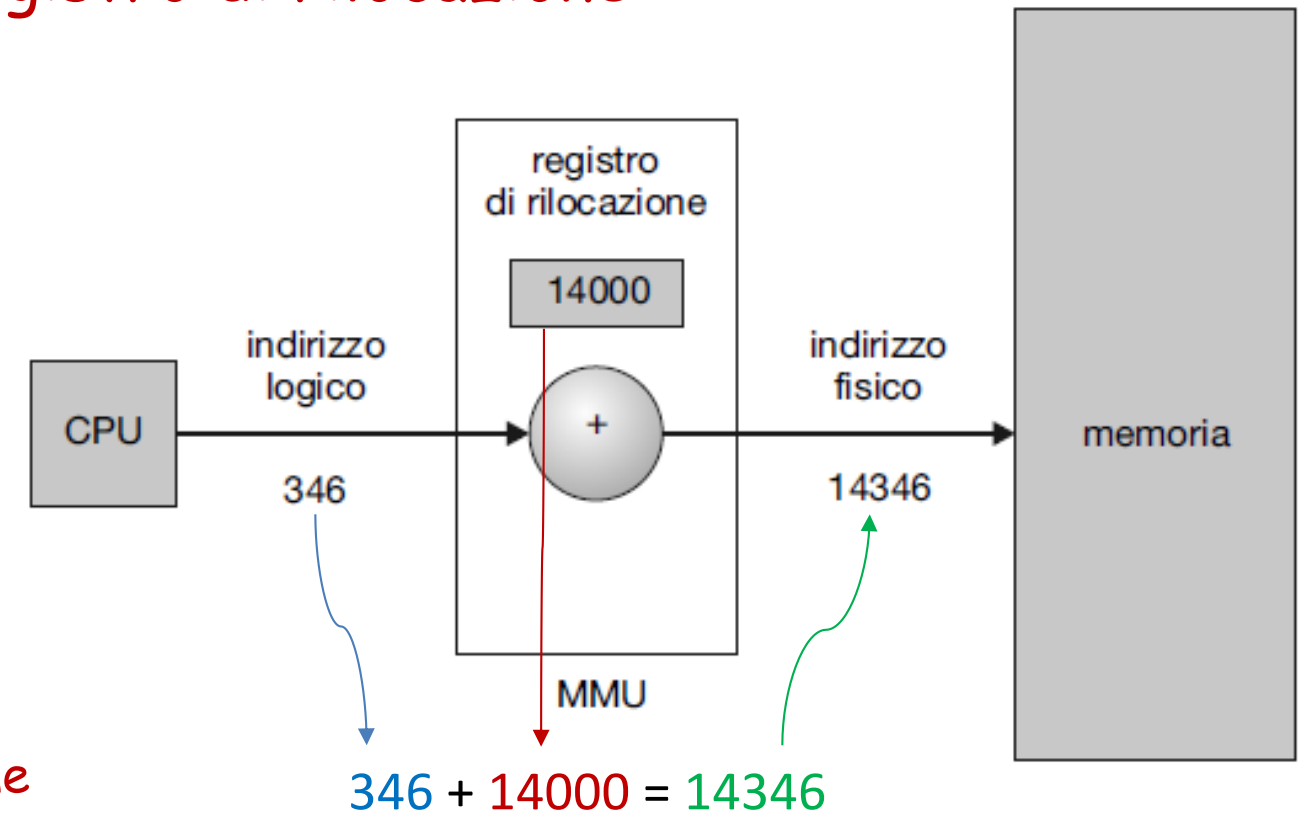
Rilocazione dinamica

- L'associazione dinamica degli indirizzi (effettuata nella fase di esecuzione)
  - comporta indirizzi logici e fisici diversi
  - pertanto, gli spazi degli indirizzi logici e fisici differiscono
- Anziché rilocare i processi staticamente al momento del caricamento in memoria, si fa uso di una MMU (Memory Management Unit), cioè di HW specifico che modifica gli indirizzi dinamicamente, ad ogni riferimento in memoria effettuato
- La MMU implementa una funzione *f*, detta funzione di rilocazione, che permette di calcolare l'indirizzo fisico *y* corrispondente ad un dato indirizzo logico x generato dal programma utente

$$y = f(x)$$

Rilocazione dinamica tramite registro di rilocazione

Es. quando un processo genera un indirizzo, prima dell'invio all'unità di memoria, si somma a tale indirizzo il valore contenuto nel registro di rilocazione



Il registro base
è ora denominato
registro di rilocazione

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Memoria virtuale

- La memoria virtuale si fonda sulla separazione dei concetti di spazio di indirizzi logico e spazio di indirizzi fisico
  - Lo spazio di indirizzi logico è ciò che vede il programma
  - Lo spazio di indirizzi fisico è l'effettiva allocazione della memoria reale
- Questa separazione permette al SO di offrire ai processi una memoria virtuale svincolata dalla memoria fisica
  - È una risorsa virtuale, da supportare tramite quella reale che è la memoria fisica
  - Il SO ha il compito di allocare la memoria fisica per fornire il supporto, di volta in volta, alle memorie virtuali dei diversi processi
- In presenza di memoria virtuale, gli indirizzi logici sono anche detti indirizzi virtuali e il loro insieme è detto spazio degli indirizzi virtuali (o spazio di indirizzi virtuale)

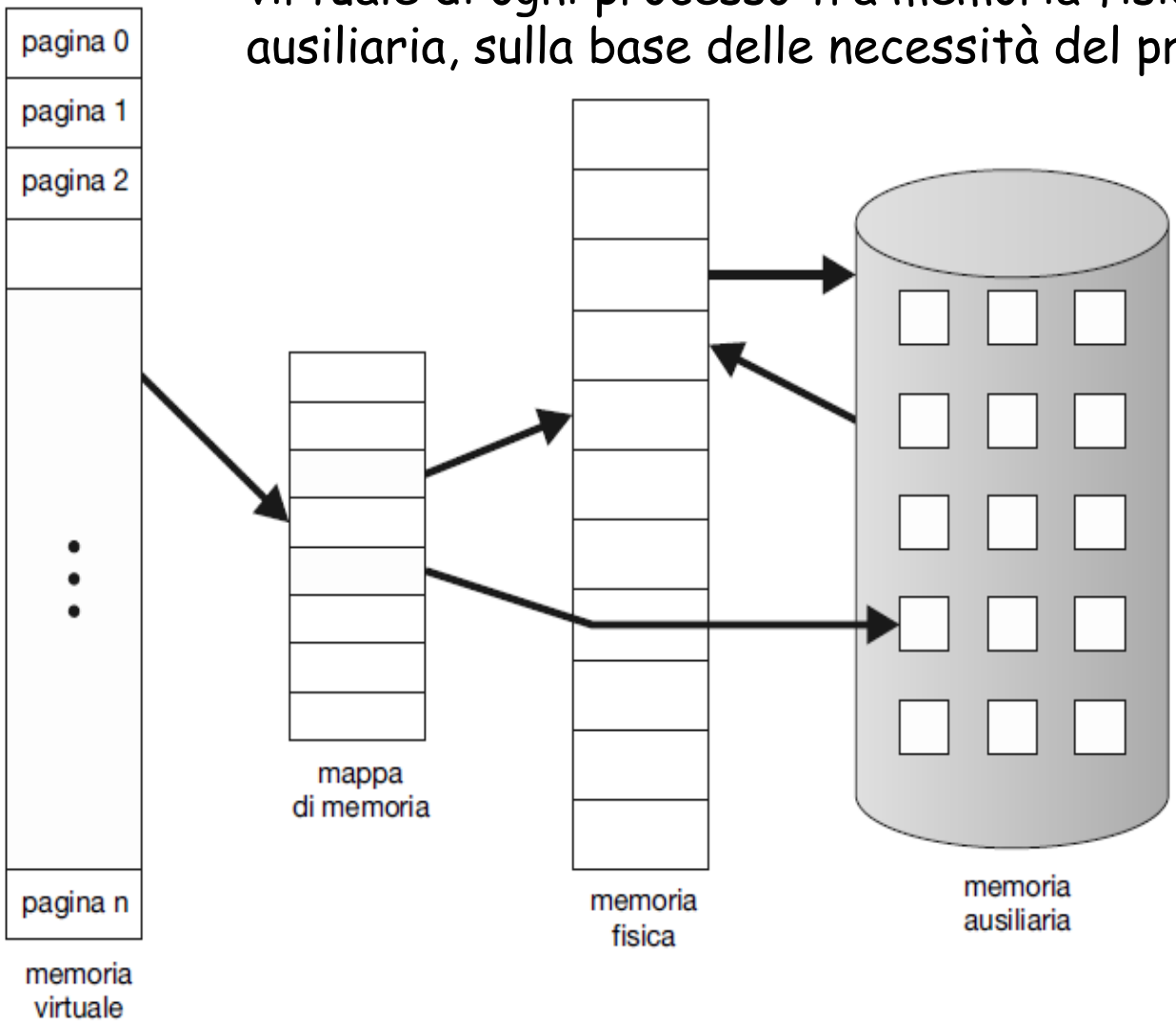
Memoria virtuale: vantaggi

1. Il più immediato vantaggio dato dalla memoria virtuale è quello di fornire ai processi uno spazio di memoria per istruzioni e dati che può essere maggiore di quello disponibile o fisicamente presente sul calcolatore
   - Sia v la dimensione dello spazio di indirizzi virtuale di un processo e f la dimensione dello spazio di indirizzi fisico del sistema
     - v dipende da quanti registri di indirizzamento ha la CPU
     - f dipende dal numero di linee di indirizzo del bus della memoria
   - Se v < f un programma non può indirizzare tutta la memoria fisica
   - Se f < v la memoria fisica è una finestra su quella virtuale

   Generalmente, ad un dato istante, solo la porzione attualmente in uso dello spazio virtuale di un processo è allocata in memoria principale
   - Il resto del suo spazio virtuale (o una copia dell'intero spazio) è allocato in memoria secondaria in un'area gestita dal SO (es. area di swap in ambiente POSIX, file di paging in Windows)

Memoria virtuale e memoria fisica

Il SO si occupa di trasferire porzioni della memoria virtuale di ogni processo tra memoria fisica e ausiliaria, sulla base delle necessità del processo



memoria virtuale = memoria fisica + memoria ausiliaria

Memoria virtuale: vantaggi

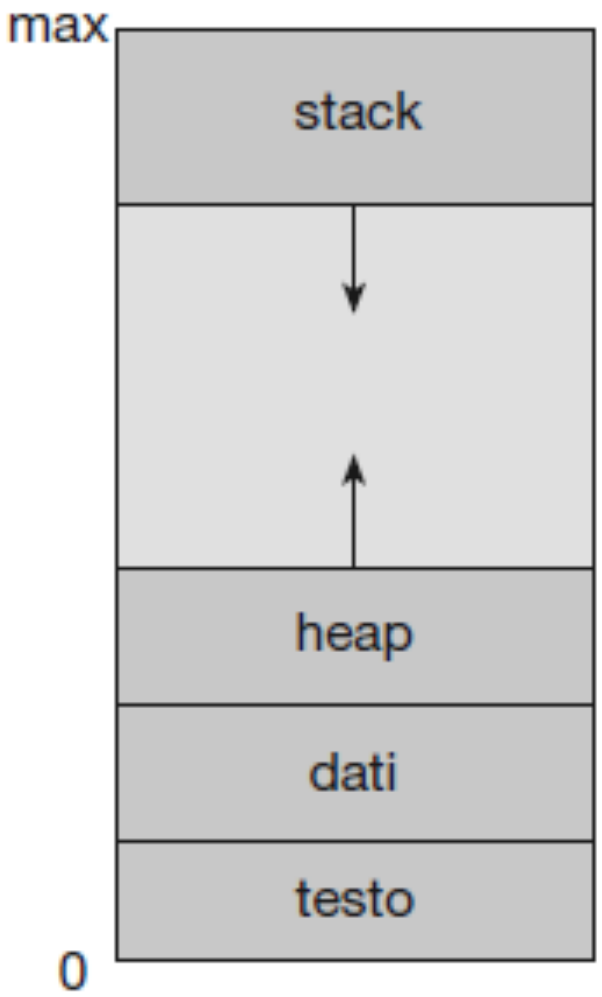
2. Facilita la programmazione
   - Il programmatore non si deve preoccupare della quantità di memoria fisica disponibile e può concentrarsi sul problema da risolvere con il programma
   - L'esecuzione del processo può procedere finché le locazioni virtuali accedute sono nella porzione presente in memoria fisica
   - Il SO gestisce il trasferimento dinamico delle porzioni necessarie in maniera trasparente al processo
   - La corrispondenza tra indirizzi logici e fisici è gestita da meccanismi HW/SW in maniera trasparente al processo
3. Consente di aumentare il grado di multiprogrammazione
   - Il SO può attivare un insieme di processi concorrenti le cui esigenze di memoria totali superano la capacità della memoria fisica a disposizione nel sistema (sovrallocazione della memoria)
4. Permette la condivisione di file e memoria fisica fra due o più processi

Memoria virtuale di un processo

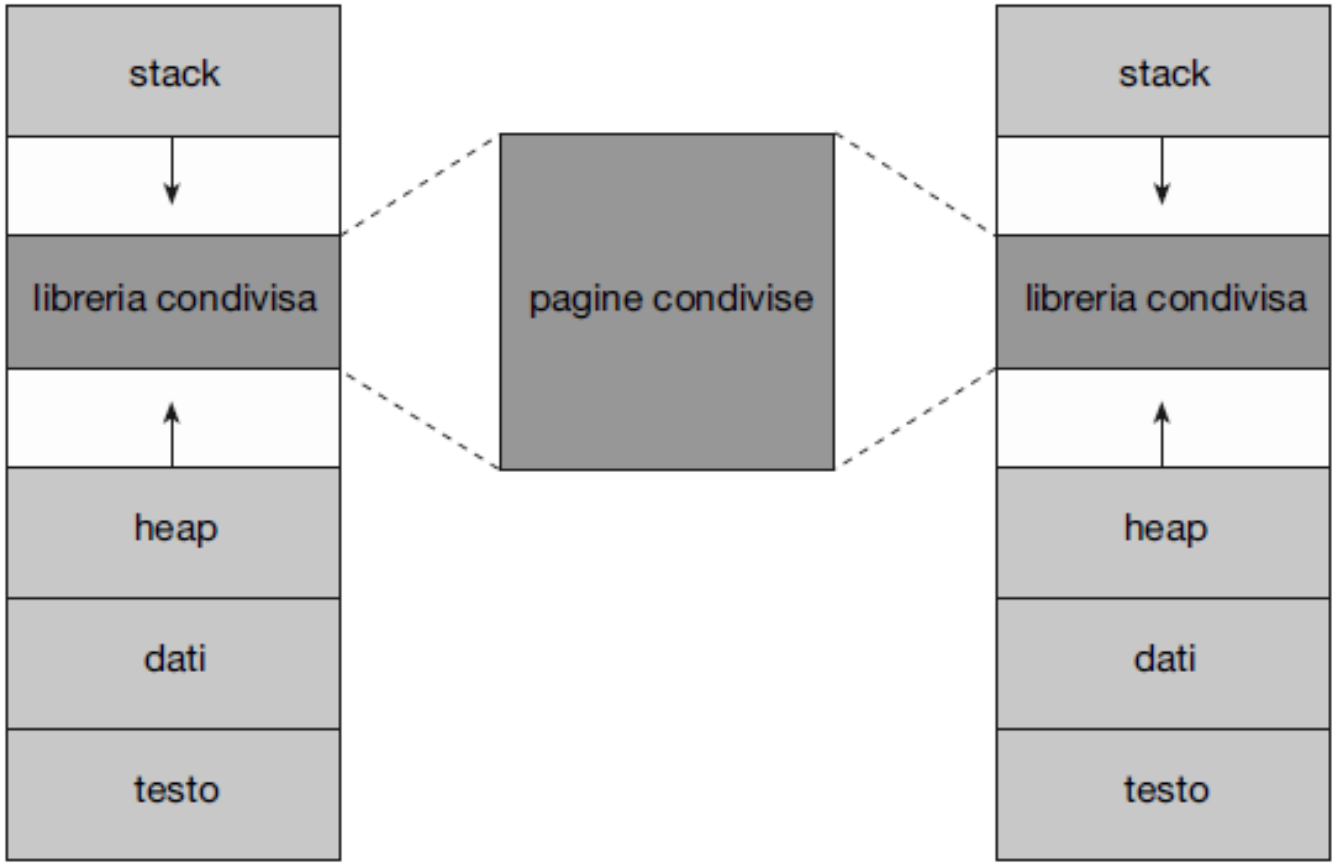
La memoria virtuale di un processo è organizzata in base alle sue esigenze, quindi è strutturata in maniera simile all'eseguibile

È generalmente suddivisa in sezioni

- **Testo**: contiene il codice eseguibile
- **Dati**: contiene le variabili globali
- **Heap**: memoria allocata dinamicamente durante l'esecuzione del processo
- **Stack**: memoria utilizzata temporaneamente durante le chiamate di funzioni
- Lo spazio vuoto che separa heap e stack serve a poter far crescere lo heap verso l'alto e lo stack verso il basso, ma anche per la condivisione di dati, codice e librerie
- È parte dello spazio degli indirizzi virtuali ma richiede memoria fisica solo se viene utilizzato
- Uno spazio virtuale che contiene 'buchi' si dice sparso



Condivisione di librerie tramite memoria virtuale



Benché ogni processo consideri la libreria come parte del proprio spazio di indirizzi virtuale, le pagine fisiche che ospitano effettivamente la libreria in memoria principale sono condivise da entrambi i processi

Memoria virtuale: tecniche associate

- La memoria virtuale abilita diverse tecniche, quali:
  - la condivisione delle pagine fisiche
  - la segmentazione/paginazione a domanda
  - la copiatura su scrittura (COW)
- La memoria virtuale è legata anche alla gestione della memoria secondaria
  - Swapping

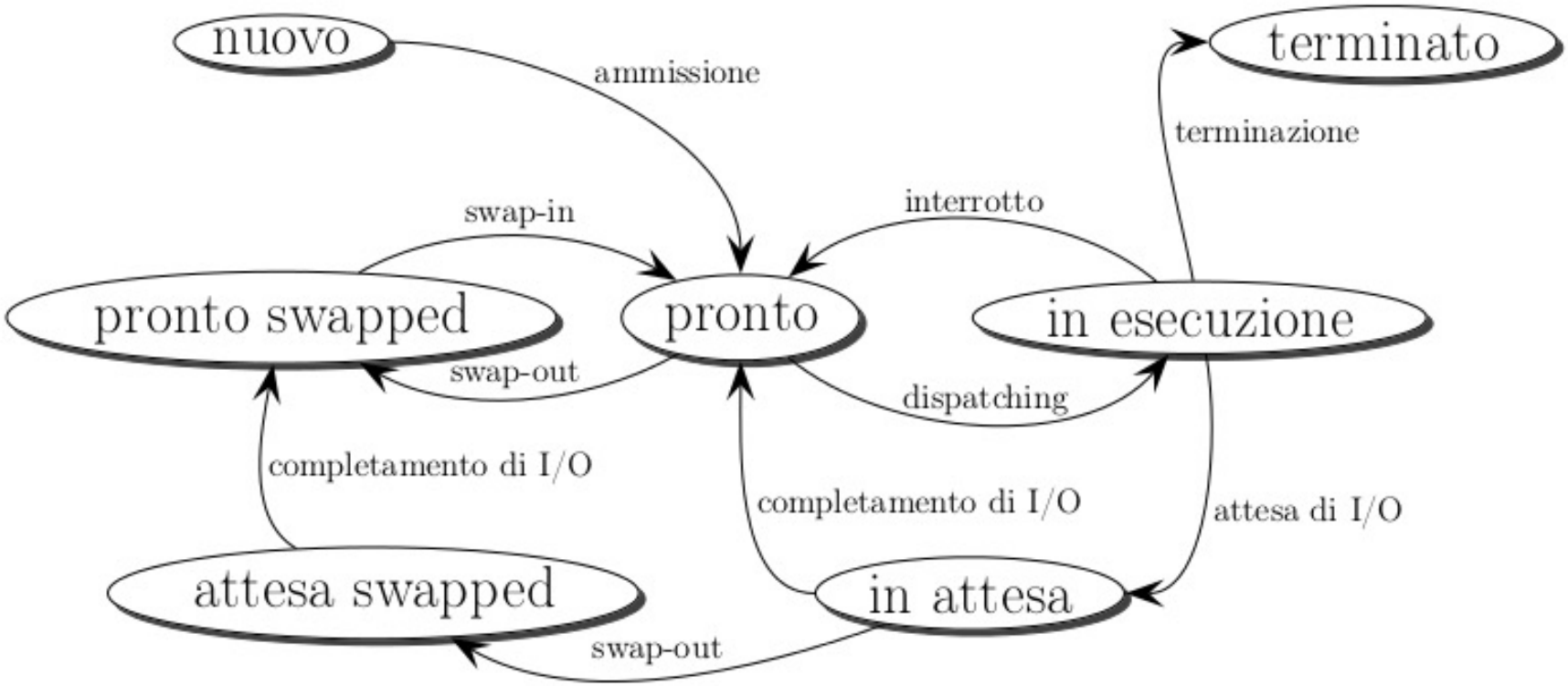
Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Avvicendamento dei processi (Swapping)

- A seconda del tipo di gestione della memoria, un processo, o una sua parte, durante la sua esecuzione può essere temporaneamente trasferito in una memoria ausiliaria (*swap out*) e poi riportato nella memoria principale (*swap in*) al momento in cui se ne riprende l'esecuzione
- Memoria ausiliaria (*backing store* o *swap area*): disco ad accesso veloce con spazio sufficiente a memorizzare le copie delle immagini di memoria di tutti i processi
  - Deve anche permettere l'accesso diretto a queste immagini
- Le immagini dei processi pronti possono risiedere in memoria principale o in memoria ausiliaria
  - Quando lo scheduler della CPU decide di eseguire un processo, chiama il dispatcher che eventualmente (ri)carica il processo in memoria principale
- Anche le immagini dei processi in attesa possono possono risiedere in memoria principale o in memoria ausiliaria

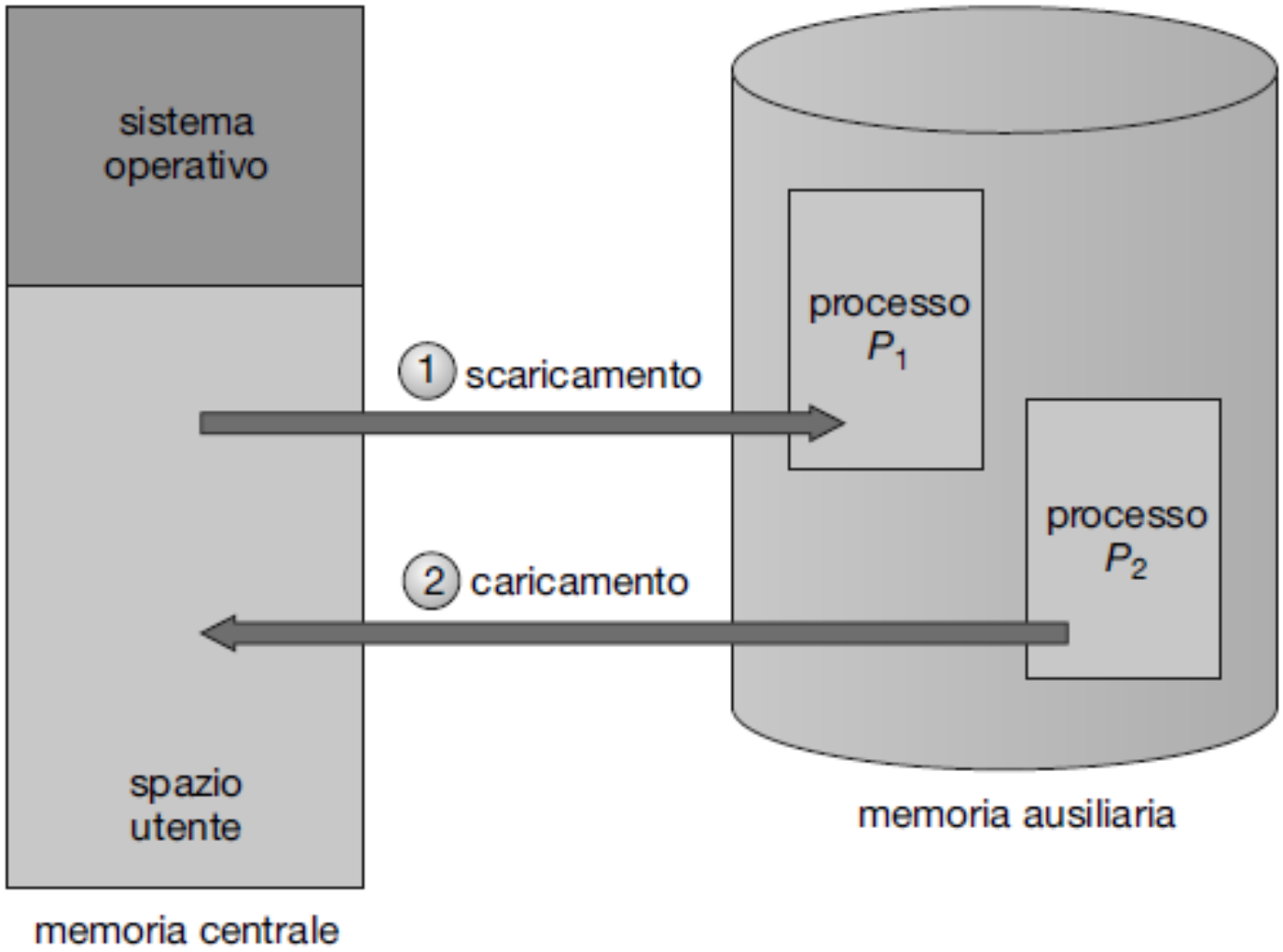
Diagramma degli stati di un processo con swapping



SO con swapping: 2 stati corrispondono a processi pronti o in attesa il cui spazio virtuale si trova nella swap area del disco

Avvicendamento standard

Riguarda lo spostamento di processi interi



Avvicendamento standard

- Quando un processo viene spostato in memoria ausiliaria, anche le strutture dati associate al processo devono essere scritte in memoria ausiliaria
  - In un processo multithread, devono essere scritte anche tutte le strutture dati relative ad ogni thread
- Il SO deve conservare i metadati relativi ai processi che sono stati spostati, in modo da poterli ripristinare quando i processi saranno riportati in memoria principale
- Vantaggio: consente di sovrallocare la memoria fisica, in modo che il sistema possa ospitare più processi rispetto alla quantità di memoria fisica effettivamente disponibile aumentando così il grado di multiprogrammazione

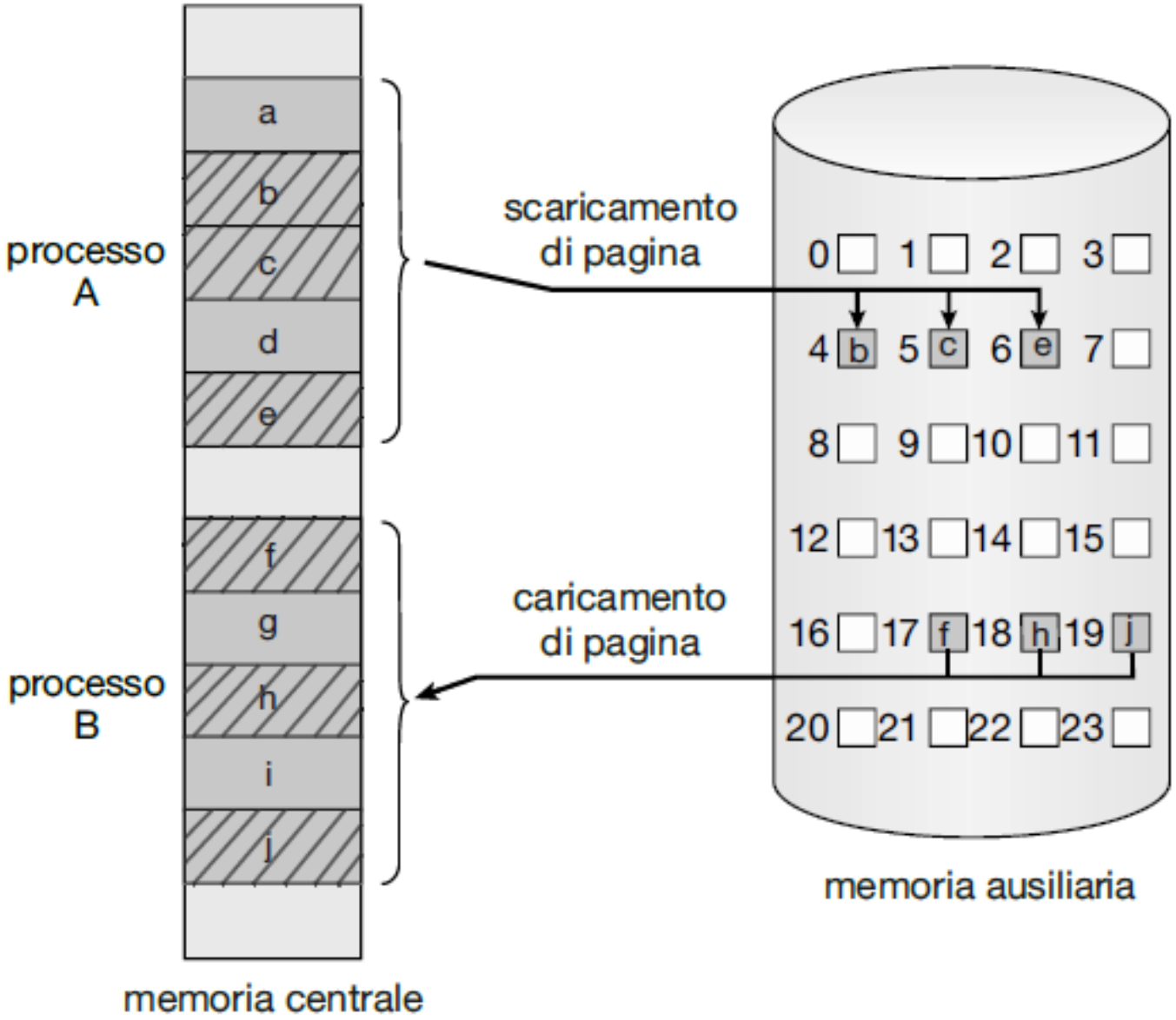
Avvicendamento standard: svantaggi

- Se la rilocazione non è dinamica, si deve ricaricare il processo nello stesso spazio di memoria che occupava in precedenza
  - Perché gli indirizzi a cui la sue istruzioni possono far riferimento sono assoluti (l'associazione è stata fatta al momento della compilazione o del primo caricamento)
- La quantità di tempo necessaria per spostare interi processi tra memoria principale e ausiliaria è proibitiva
  - Per alleggerire il carico dello swapping, se ci si aspetta che la necessità di memoria di un processo possa crescere dinamicamente, conviene allocargli un po' di memoria extra al momento del caricamento (altrimenti, se non potesse crescere dovrebbe essere sospeso o subire uno swap-out)
- Per questo molti SO moderni usano sue varianti

Avvicendamento con paginazione

- È una variante dell'avvicendamento standard, utilizzata in connessione con la gestione della memoria con paginazione, in cui è possibile spostare alcune pagine di un processo anziché l'intero processo
  - Un'operazione page out sposta una pagina dalla memoria principale alla memoria ausiliaria
  - L'operazione opposta è nota come page in
- Vantaggi
  - Consente di sovrallocare la memoria fisica, ma evita il costo dello spostamento di processi interi, poiché presumibilmente solo un piccolo numero di pagine sarà coinvolto nello spostamento
  - Si integra bene con la gestione della memoria virtuale
- Utilizzato dalla maggior parte dei SO moderni, inclusi Linux e Windows

Swapping con paginazione



Avvicendamento nei sistemi mobili

- I sistemi mobili in genere non supportano nessuna forma di avvicendamento dei processi
- Ciò è principalmente dovuto al fatto che, per la memorizzazione non volatile, tali sistemi utilizzano solitamente la memoria flash la quale
  - è meno capiente rispetto ai dischi rigidi
  - può tollerare un numero limitato di scritture prima che diventi inaffidabile
  - ha bassa velocità di trasferimento con la memoria principale

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Aspetti caratterizzanti la gestione della memoria

- Tutte le tecniche per la gestione della memoria sono caratterizzate da quattro parametri principali

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
• STATICA • DINAMICA	• CONTIGUA • NON CONTIGUA	• UNICO • SEGMENTATO	• UNICO • A DOMANDA

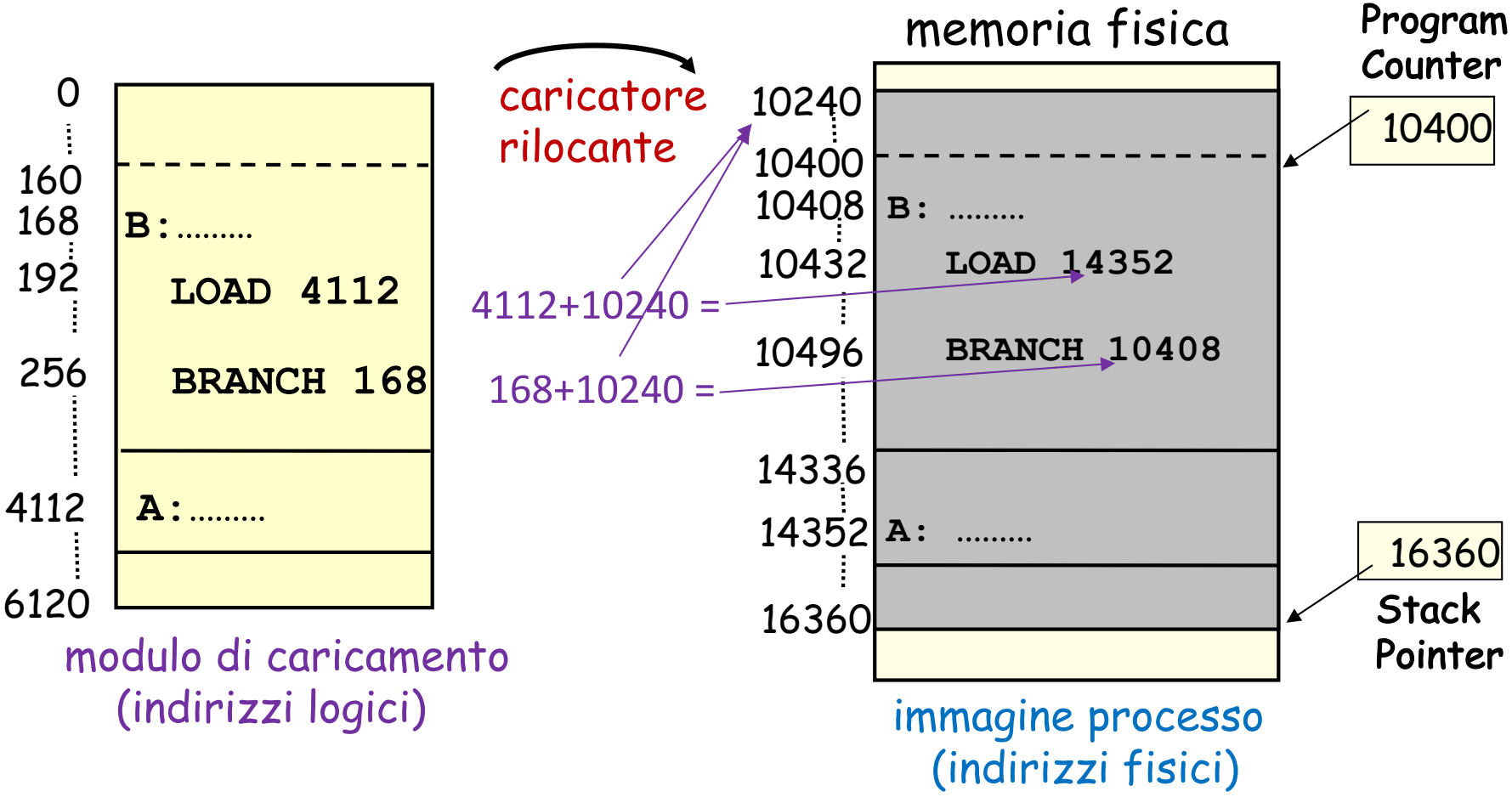
- Tali parametri dipendono:
  - dall'architettura HW del processore
  - dalle scelte di progetto del SO

Rilocazione statica e dinamica

Come si assegnano le aree di memoria ai processi?

- Rilocazione *statica*
  - effettuata nelle fasi di compilazione o caricamento
  - a run-time gli spazi di indirizzi logico e fisico coincidono
  - la CPU genera indirizzi fisici (assoluti)
- Rilocazione *dinamica*
  - effettuata in fase di esecuzione
  - gli spazi di indirizzi logico e fisico differiscono
    - la CPU genera indirizzo logici
    - la MMU associa indirizzi logici a indirizzi fisici

Rilocazione statica degli indirizzi

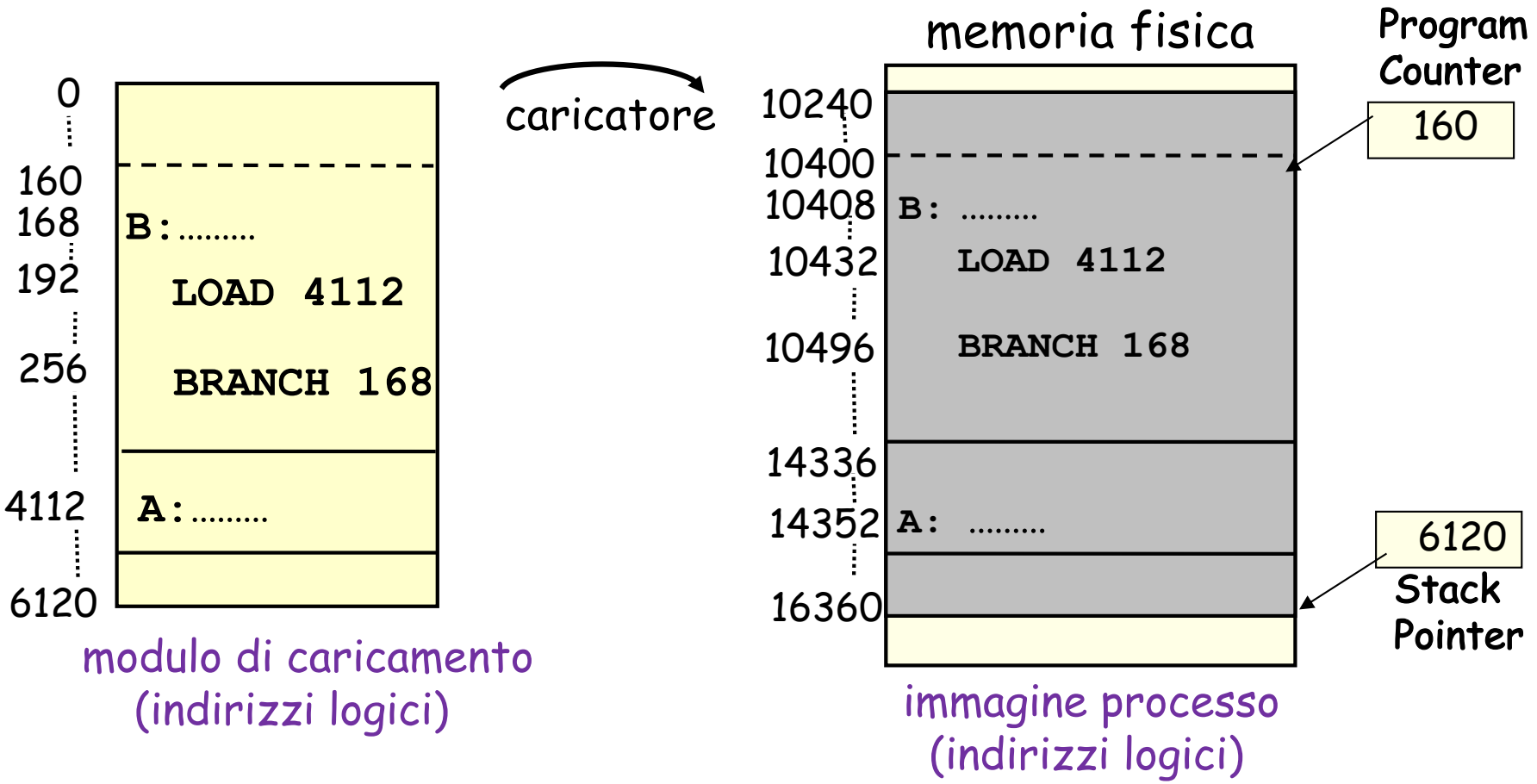


- PC e SP vengono inizializzati con indirizzi fisici
- I processi generano indirizzi fisici

Rilocazione statica degli indirizzi

- Gestione piuttosto semplice
- Inconvenienti
  - non permette di (ri)caricare il processo in un'area di memoria differente (per esempio, a seguito di operazioni di swap out – swap in)
  - infatti, alcune delle informazioni che il processo può aver prodotto prima di subire lo swap out, per esempio informazioni di ritorno da chiamate di procedure, fanno riferimento ad indirizzi fisici originali (dato che la CPU lavora con indirizzi fisici)
  - soluzione dell'inconveniente: ritardare la fase di rilocazione degli indirizzi

Rilocazione dinamica degli indirizzi



- PC e SP vengono inizializzati con indirizzi logici
- I processi generano indirizzi logici, l'architettura HW del sistema, tramite la MMU, converte gli indirizzi logici in indirizzi fisici a run-time

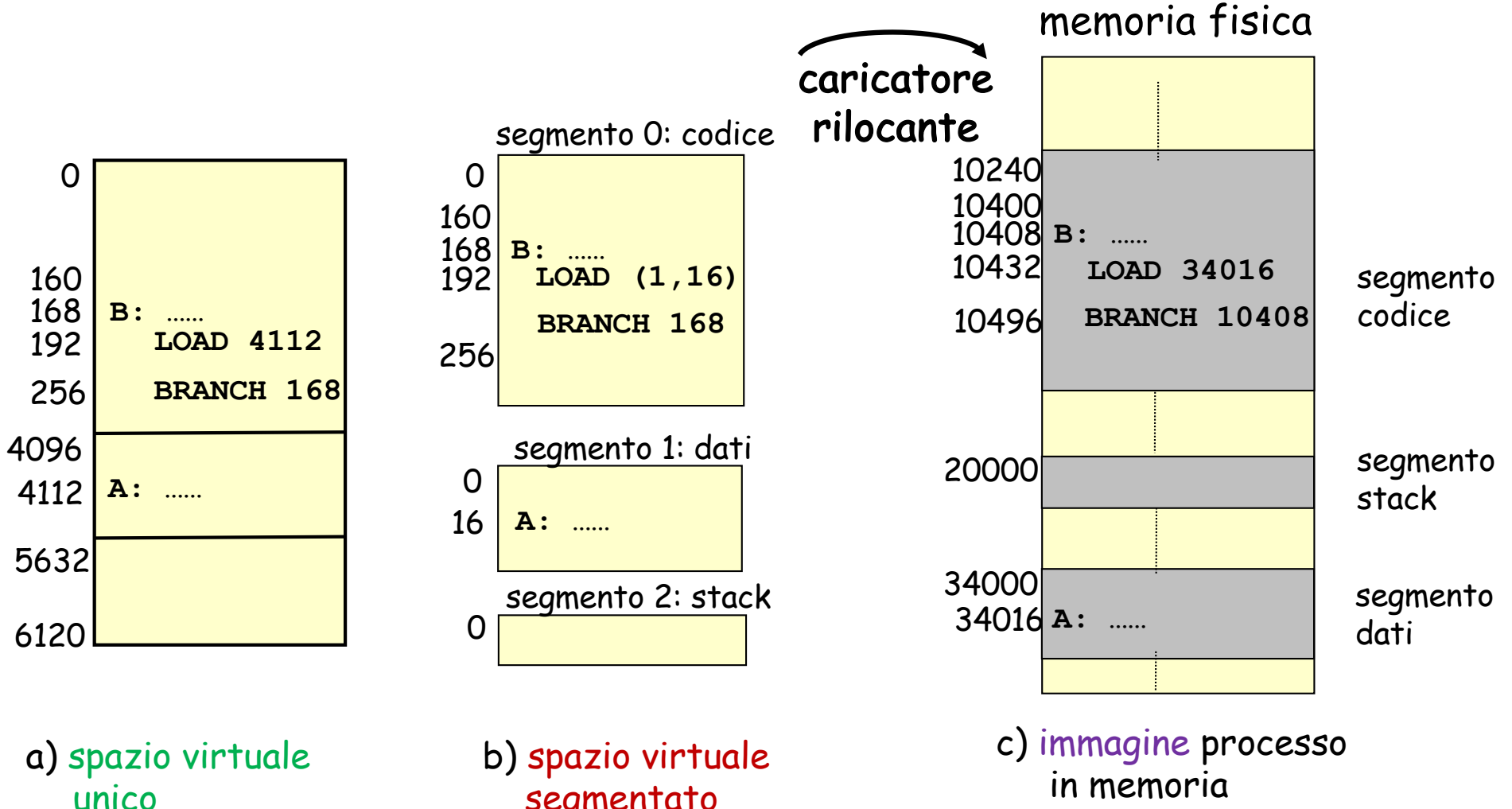
Allocazione della memoria fisica

- Se la rilocazione è statica, dev'essere contigua: *ad indirizzi logici contigui devono corrispondere indirizzi fisici contigui*
  - Se 2 istruzioni di un programma, una successiva all'altra, sono allocate in 2 indirizzi logici contigui, allora il loader deve allocare le 2 istruzioni in 2 indirizzi fisici contigui
  - Infatti, il PC contiene indirizzi fisici, quindi terminata l'esecuzione della prima istruzione, la CPU preleva la seconda istruzione dalla locazione con indirizzo fisico successivo (e non c'è modo di fare diversamente)
- Se la rilocazione è dinamica, può anche essere non contigua
  - Infatti, il PC contiene indirizzi logici, quindi la CPU, terminata l'esecuzione della prima istruzione, preleva la seconda istruzione dalla locazione di indirizzo logico successivo
  - Ciò non implica necessariamente che l'istruzione sarà prelevata dalla locazione di memoria di indirizzo fisico successivo (poiché l'indirizzo fisico sarà ottenuto dall'applicazione della funzione di rilocazione all'indirizzo logico)

Organizzazione dello spazio di memoria virtuale

- Unico: il linker alloca tutti i moduli componenti un programma (es. codice, dati, stack, heap), compresi eventuali funzioni di libreria necessarie, in indirizzi virtuali contigui
- Segmentato: non è necessariamente vero che il primo indirizzo virtule di un modulo è quello successivo all'ultimo indirizzo del modulo che lo precede: i moduli sono ospitati in un certo numero di segmenti, ciascuno avente 0 come indirizzo iniziale
  - Il numero dei segmenti può essere arbitrario
    es. uno per il codice, uno per i dati, uno per lo stack, uno per lo heap
  - Ciascun segmento può essere rilocato in memoria fisica indipendentemente dagli altri
  - Un indirizzo virtuale è una coppia:
    numero di segmento, locazione nell'ambito del segmento
  - Nel caso di rilocazione statica, il compilatore o il caricatore rilocante usa una tabella contenente gli indirizzi fisici iniziali di ciascun segmento
  - Nel caso di rilocazione dinamica, la MMU fa uso dei valori base e limite contenuti in una tabella con tanti elementi quanti sono i segmenti

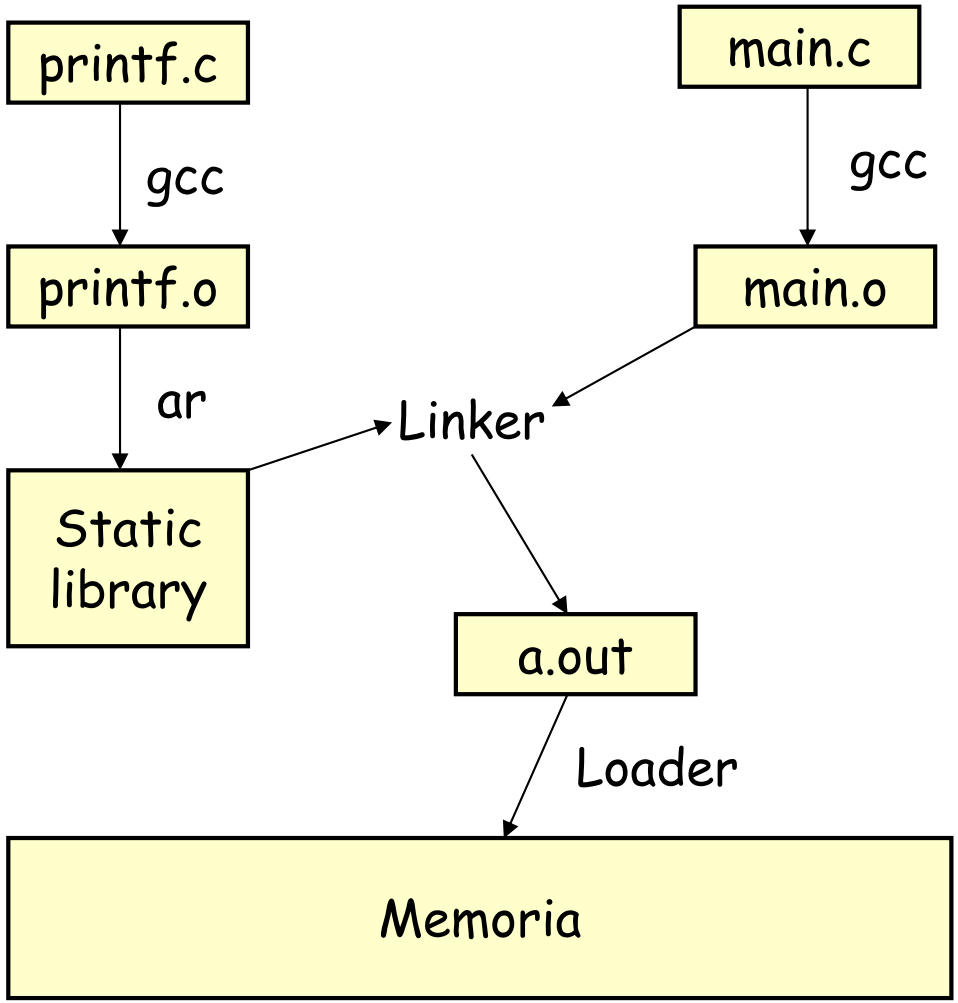
Organizzazione dello spazio di memoria virtuale



Caricamento dello spazio virtuale in memoria fisica

- Unico (o statico): programma e dati di un processo sono interamente e costantemente caricati in memoria
  - Vincolo: le dimensioni della memoria virtuale devono essere minori o uguali a quelle della memoria fisica disponibile
  - Assegnazioni e rilasci della memoria avvengono esclusivamente alla creazione e alla terminazione dei processi
  - Permette rilocazione statica degli indirizzi
- A domanda (o su richiesta, o dinamico): programma e dati non sono necessariamente caricati per intero
  - Vincolo: richiede rilocazione dinamica degli indirizzi
  - La memoria fisica assegnata ad un processo varia nel tempo, sia in quantità che in posizione
  - Le assegnazioni di memoria fisica possono essere effettuate e revocate un numero arbitrario di volte durante l'esistenza dei processi
  - Le dimensioni della memoria virtuale possono essere maggiori di quelle della memoria fisica

Caricamento unico

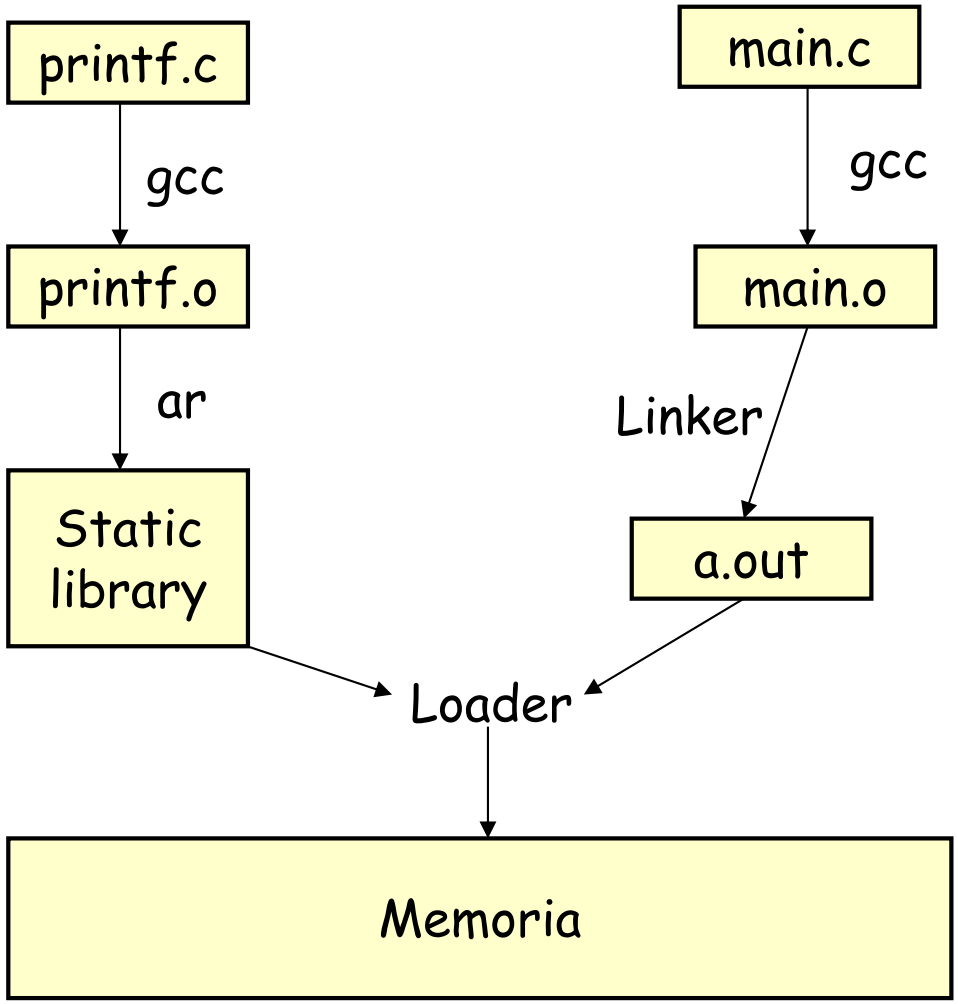


Tutto il programma (codice più tutte le funzioni di libreria) viene caricato in memoria prima di cominciare l'esecuzione

Caricamento a domanda

- Il programma principale viene caricato in memoria e viene eseguito
- Tutte le procedure ausiliarie sono mantenute su disco in un formato di caricamento rilocabile
- Quando durante l'esecuzione una procedura deve chiamarne un'altra, controlla innanzitutto se che sia stata caricata
- In caso contrario, richiama il linking loader per
  - caricare in memoria la procedura richiesta
  - aggiornare le tabelle degli indirizzi del programma per riflettere questa modifica
- Quindi il controllo viene passato alla procedura appena caricata

Caricamento a domanda



Il loader carica solo ciò che serve e quando serve

Caricamento a domanda: considerazioni

- Si carica una procedura solo quando viene invocata
  - Alcune opzioni di un programma sono utilizzate di rado
  - Parti di codice deputate alla gestione di situazioni di errore servono solo se, e quando, l'errore si presenta
- Migliore utilizzo della memoria: una procedura che non viene usata non viene mai caricata
- Non richiede un supporto particolare del SO
  - Spetta agli utenti di progettare i programmi in modo da trarre vantaggio da tale schema
  - Il SO può tuttavia fornire librerie di procedure che lo realizzano
- Permette l'esecuzione dei processi anche quando la somma delle dimensioni degli spazi virtuali dei processi supera la dimensione della memoria fisica
- Presuppone l'esistenza di una memoria ausiliaria (es. swap area) in memoria secondaria:

  memoria virtuale = memoria fisica + memoria ausiliaria

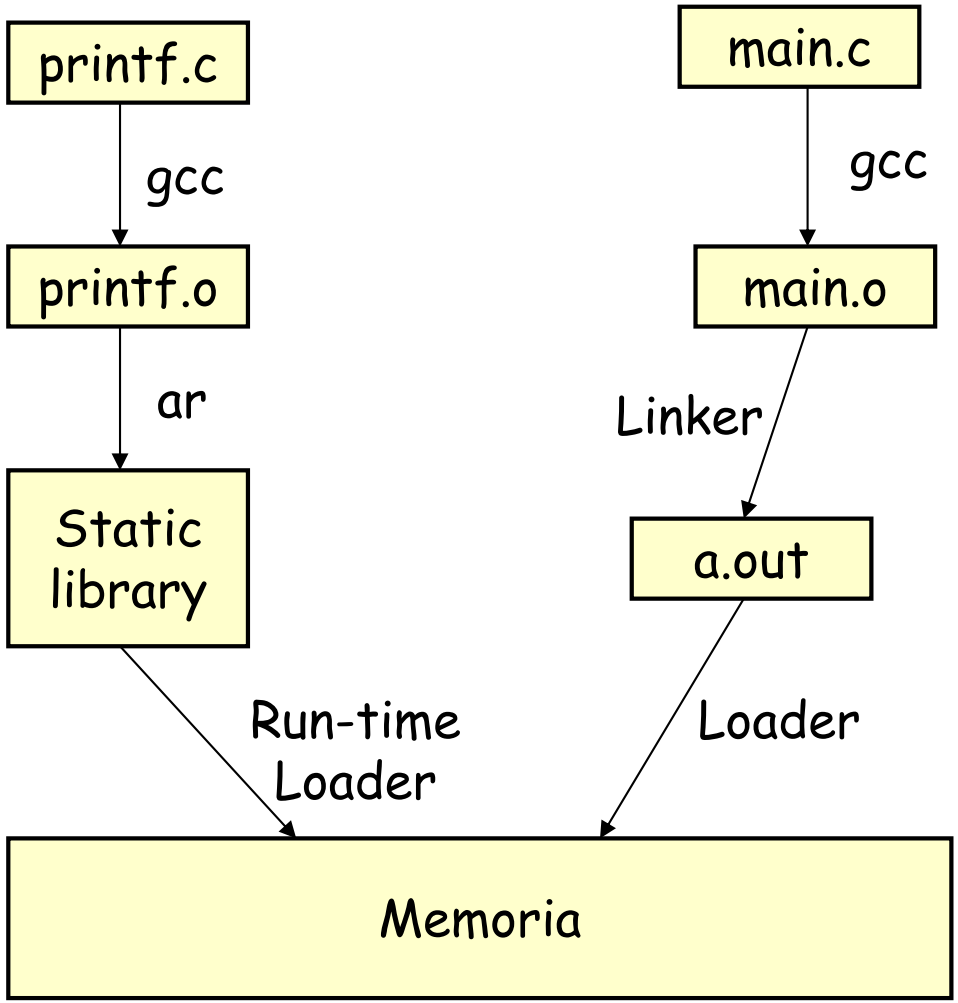
Collegamento (linking) statico

- Con il collegamento statico, le librerie di sistema sono trattate come qualsiasi altro modulo oggetto e sono combinate dal loader nell'immagine binaria del programma
- Tutti i programmi devono contenere all'interno della loro immagine eseguibile una copia della libreria
- Ciò causa un aumento della dimensione del file eseguibile, con possibile spreco di spazio in memoria principale
- Alcuni SO supportano solo il collegamento statico

Collegamento (linking) dinamico

- Non solo il caricamento, ma anche il collegamento di una libreria è differito fino al momento dell'esecuzione
- Il linker inserisce nel programma informazioni di rilocazione che consentono alla libreria di essere collegata dinamicamente e caricata in memoria (dopo che il programma è stato caricato) solo se è richiesta durante l'esecuzione del programma
- Si usa soprattutto con le librerie di sistema (es. librerie di procedure di un linguaggio di programmazione, quale il C) e comunque con librerie che possono essere condivise (es. *Dynamic-Link Library* in Windows)
- La maggior parte dei sistemi consente a un programma di collegare le librerie dinamicamente quando viene caricato

Collegamento dinamico



La libreria sarà collegata dinamicamente e caricata in memoria quando il programma è già caricato

Il run-time loader è comune a tutto il sistema

Collegamento dinamico: considerazioni

- Significativo risparmio della memoria
  - Evita di collegare e caricare librerie che potrebbero non essere utilizzate durante l'esecuzione
  - Consente di caricare un'unica copia condivisa delle funzioni di libreria
- Le librerie collegate dinamicamente possono essere aggiornate e tutti i programmi che fanno riferimento alla libreria useranno automaticamente la nuova versione
- A differenza del caricamento dinamico, il collegamento dinamico richiede assistenza al SO il quale
  - deve controllare se la procedura richiesta da un processo è presente in memoria, altrimenti deve caricarla
  - per via dei meccanismi di protezione della memoria, solo il SO può controllare se la procedura si trova nello spazio di memoria di un altro processo e può consentire a più processi di accedere gli stessi indirizzi di memoria

Collegamento statico vs. dinamico

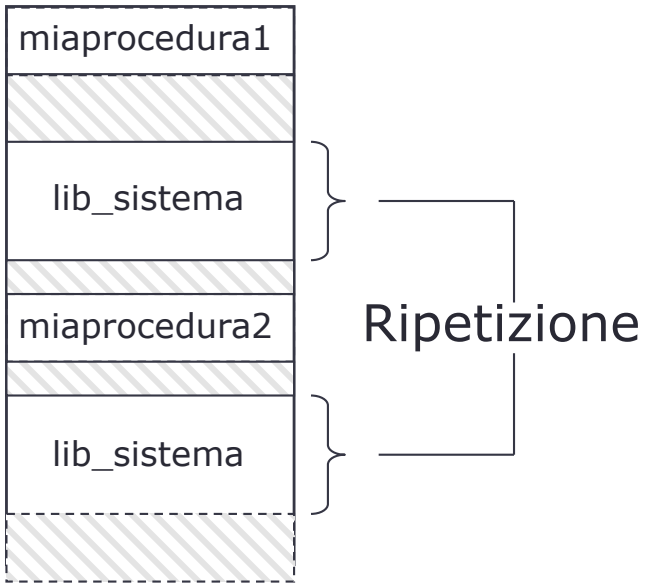
SENZA
COLLEGAMENTO DINAMICO
(COLLEGAMENTO STATICO)

CALL miaprocedura1

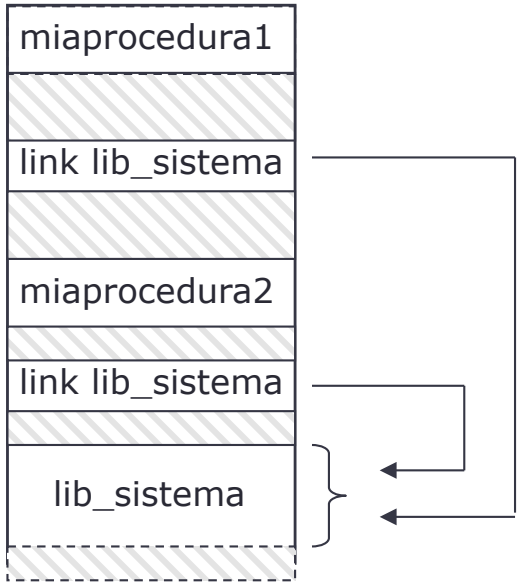
CALL lib_sistema

CALL miaprocedura2

CALL lib_sistema



CON
COLLEGAMENTO DINAMICO



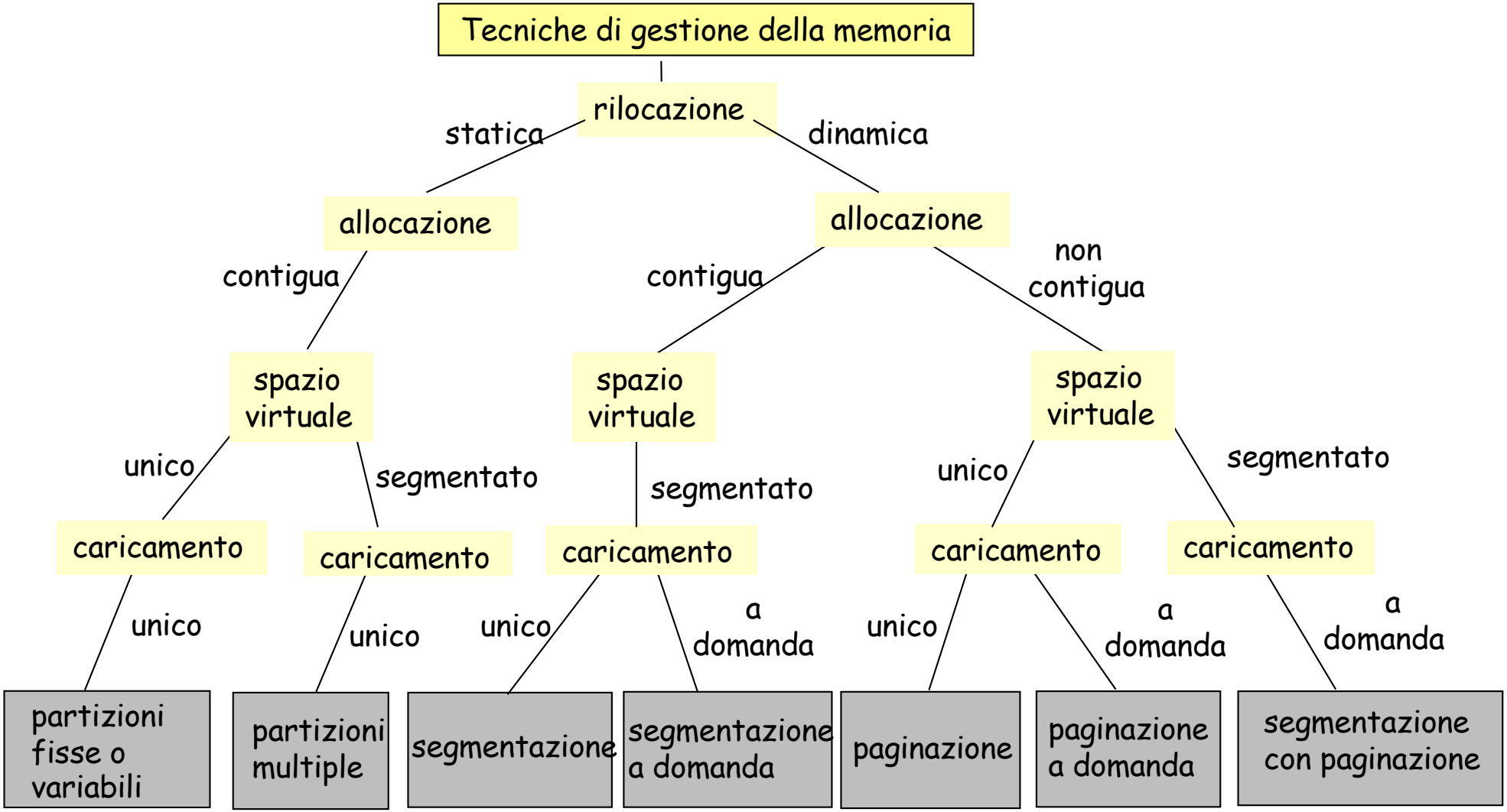
Aspetti caratterizzanti la gestione della memoria

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
• STATICA • DINAMICA	• CONTIGUA • NON CONTIGUA	• UNICO • SEGMENTATO	• UNICO • A DOMANDA

Aspetti caratterizzanti la gestione della memoria

- Alcune combinazioni di valori non sono significative
- Nel caso di rilocazione statica degli indirizzi
  - l'allocazione della memoria fisica è sempre contigua
  - il caricamento dello spazio virtuale è unico (cioè avviene "tutto insieme")
- Nel caso di rilocazione dinamica degli indirizzi
  - se l'allocazione della memoria fisica è contigua, allora lo spazio virtuale è segmentato (per agevolare la condivisione, ove possibile)
  - se l'allocazione della memoria fisica è non contigua e lo spazio virtuale è segmentato, allora il caricamento è a domanda

Tecniche di gestione della memoria



Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Memoria partizionata

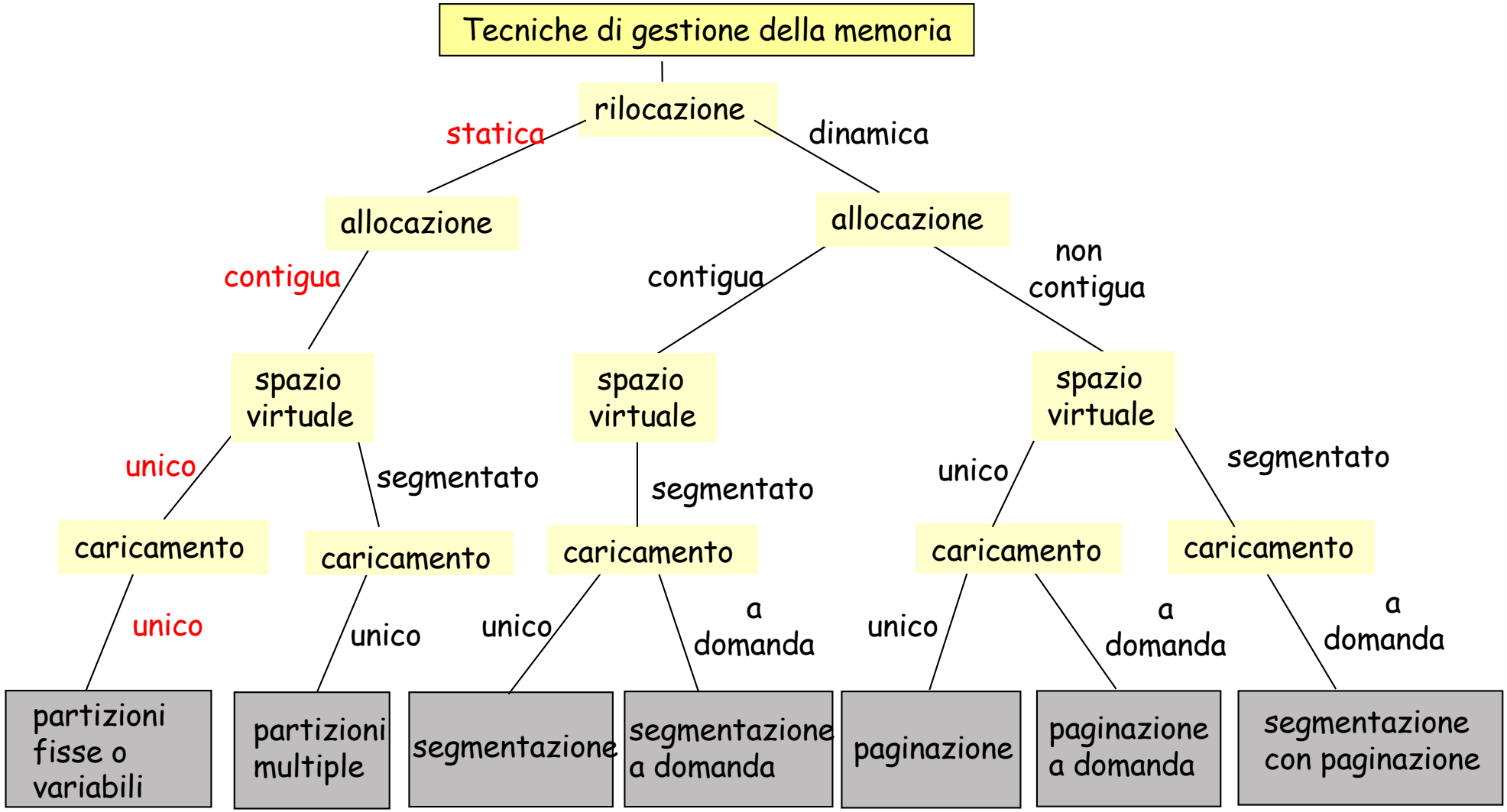
- La memoria fisica è divisa in due parti:
  - una parte per il SO residente (solitamente nella memoria bassa, dove si trova il vettore delle interruzioni)
  - l'altra parte è dedicata ad ospitare i processi utente
- La rilocazione degli indirizzi è statica tramite compilatore o caricatore rilocante (non ci sono supporti HW tipo MMU)
- Due tecniche:
  - partizioni fisse o variabili (spazio virtuale unico)
  - partizioni multiple (spazio virtuale segmentato)

Partizioni fisse o variabili

La memoria virtuale è costituita da un unico spazio virtuale contiguo

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
STATICA	CONTIGUA	UNICO	UNICO

Partizioni fisse o variabili



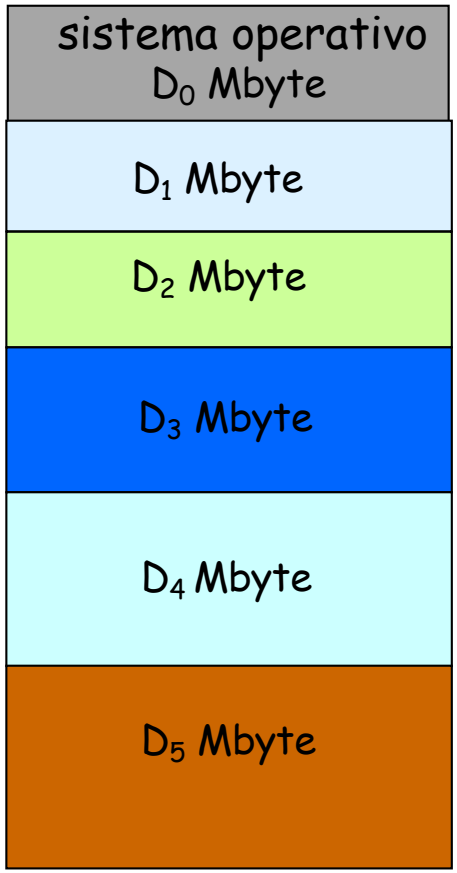
Partizioni fisse o variabili

- Il SO prende nota dei requisiti di memoria dei processi man mano che questi entrano nel sistema
- Quando poi decide di assegnare spazio in memoria a un processo, il SO cerca un'area contigua di memoria di dimensioni sufficienti a contenere la sua immagine
  - Il processo viene quindi caricato e può competere per l'assegnazione della CPU
- L'area di memoria resterà assegnata al processo fino a quando questo non termina
- Se il processo subisce uno swap out, all'atto del successivo swap in sarà necessario ricaricarlo nella stessa area di memoria (per via della rilocazione statica degli indirizzi)
- A tale scopo, la parte di memoria destinata ai processi utente è suddivisa in partizioni, ciascuna può essere assegnata ad un solo processo per volta
- Due diversi schemi di gestione:
  - partizioni fisse
  - partizioni variabili

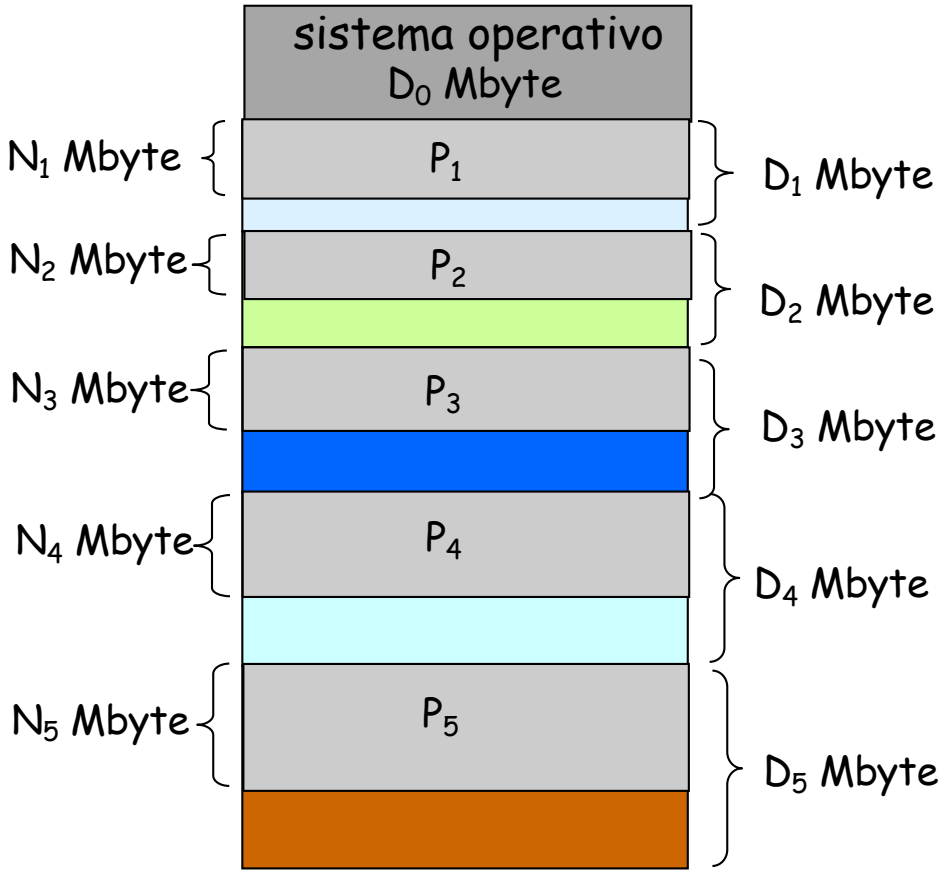
Partizioni fisse

- La memoria destinata ai processi utente è suddivisa in un numero fisso di partizioni di indirizzo iniziale e dimensioni fissate, definite in fase di installazione del SO
- Il SO conserva una tabella delle partizioni in cui sono indicate le partizioni di memoria disponibili e quelle occupate
  - Ogni partizione può ospitare un singolo processo
  - Quando si carica un processo gli si assegna una partizione disponibile (diversi algoritmi possibili, es. best-fit/first-fit, ne parleremo poi)
  - Quando un processo termina, o subisce uno swap out, la partizione ad esso assegnata ridiventa disponibile
- Pur essendo statica la rilocazione degli indirizzi, la memoria fisica può essere allocata ai processi dinamicamente
  - Per ogni partizione può essere mantenuta una coda di processi che si alternano nel suo uso tramite operazioni di swap out e swap in

Partizioni fisse: esempio



a) partizioni libere



b) frammentazione interna =
$(D_1-N_1)+ (D_2-N_2)+(D_3-N_3)+(D_4-N_4)+(D_5-N_5)$

Partizioni fisse: considerazioni

- Originariamente usato da IBM OS/360, attualmente non più in uso
- Malgrado la tecnica richieda un basso overhead, l'uso della memoria è inefficiente
- Frammentazione interna (alle partizioni): la memoria allocata è in generale maggiore di quella utilizzata
  - Raramente le dimensioni delle immagini dei processi coincidono con le dimensioni delle partizioni in cui sono ospitate
  - La differenza tra le due quantità costituisce un'area di memoria interna alla partizione che resta inutilizzata
  - Si presenta ogniqualvolta la memoria si alloca in blocchi di dimensioni prestabilite (quindi anche con la paginazione)
- Mancanza di flessibilità: numero e dimensioni delle partizioni sono fissate una volta per tutte in fase di installazione
  - Grado di multiprogrammazione limitato dal numero delle partizioni
  - Impossibilità di caricare (e, quindi, eseguire) processi la cui immagine in memoria è più grande della dimensione delle singole partizioni

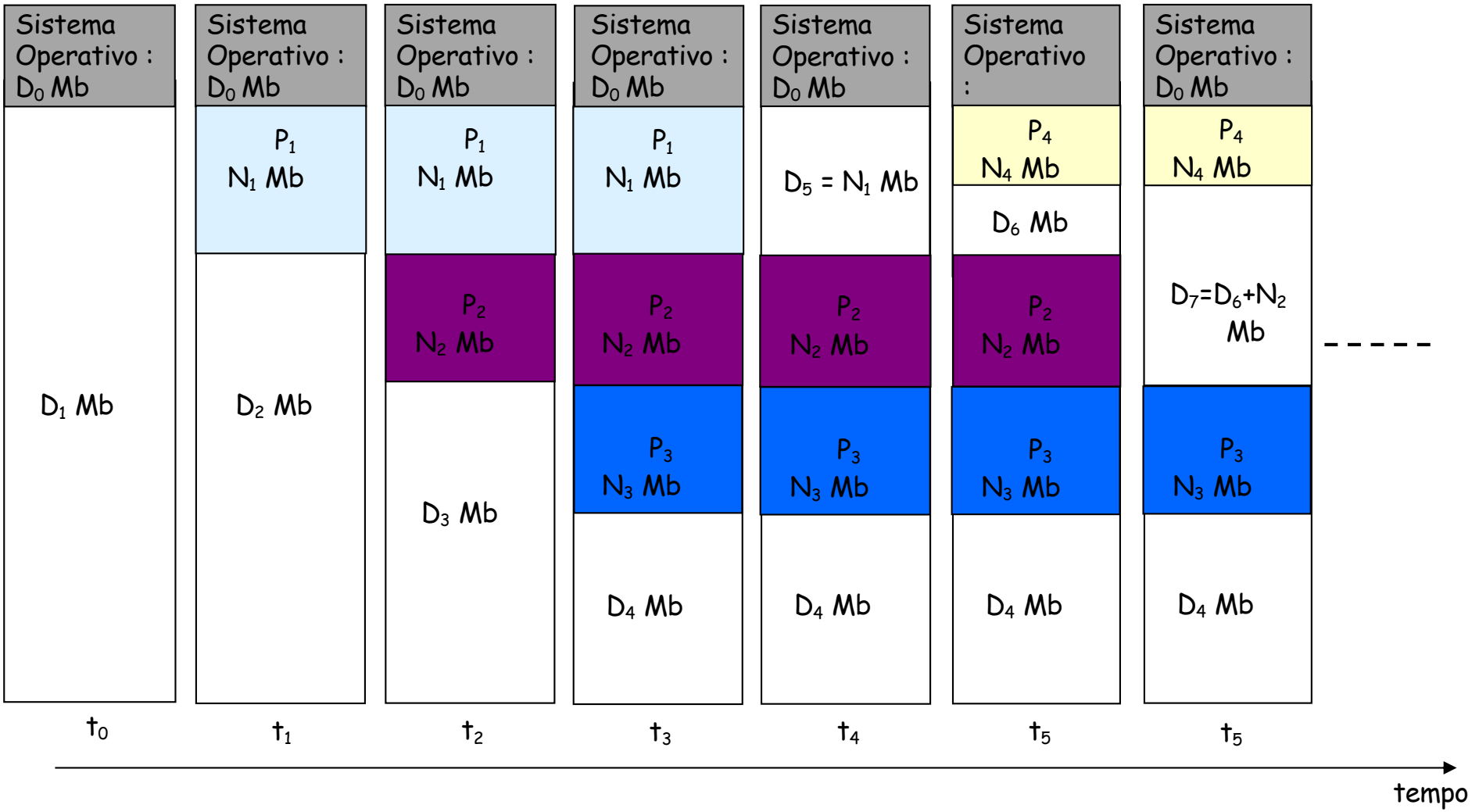
Partizioni variabili

- Generalizza lo schema precedente
- Le caratteristiche (numero e dimensioni) delle partizioni sono definite dinamicamente, in modo che corrispondano alle effettive esigenze di memoria dei processi in esse caricati
  - Inizialmente tutta la memoria per i processi utente costituisce un'unica partizione
  - Tale partizione viene dinamicamente suddivisa in più parti in base delle esigenze dei processi

Compiti del SO

- Quando decide di caricare un processo in memoria, il SO cerca una partizione disponibile abbastanza grande per ospitarlo
  - Se ne esiste una, assegna al processo la quantità di memoria necessaria e trasforma la parte restante in una partizione disponibile (eventualmente accorpandola ad una partizione contigua)
  - Altrimenti, il SO può
    - attendere che si crei una partizione disponibile sufficientemente grande, oppure
    - scorrere la coda d'ingresso alla ricerca di un processo le cui necessità di memoria siano soddisfacibili e, quindi, caricarlo
- Quando un processo termina, o subisce uno swap out, la partizione ad esso assegnata ridiventa disponibile
  - Se è contigua ad altre partizioni disponibili, il SO le accorpa e ne crea una sola
  - Successivamente, il SO controlla se la partizione disponibile appena creata soddisfa le richieste di memoria di qualche processo in coda d'ingresso (nel qual caso, lo carica in memoria)

Partizioni variabili: esempio



Algoritmi di assegnazione della memoria

- Problema: data una lista di partizioni disponibili, come soddisfare una richiesta di allocazione di un'area di memoria di una certa dimensione
- *Soluzione*: varie strategie per *l'assegnazione dinamica della memoria* quando più partizioni disponibili possono soddisfare la richiesta
  - **First-fit**: si alloca la *prima* partizione disponibile sufficientemente grande
  - **Best-fit**: si alloca la *più piccola* tra le partizioni disponibili sufficientemente grandi
    - Si devono esaminare tutte le partizioni disponibili, a meno che non siano ordinate per dimensione crescente
    - Produce le partizioni disponibili inutilizzate più piccole
  - **Worst-fit**: si alloca la *più grande* tra le partizioni disponibili sufficientemente grandi
    - Si devono esaminare tutte le partizioni disponibili, a meno che non siano ordinate per dimensione decrescente
    - Produce le partizioni disponibili inutilizzate più grandi

Algoritmi di assegnazione della memoria: considerazioni

- Simulazioni mostrano che
  - first-fit è più veloce di best-fit, mentre il loro utilizzo della memoria è simile
  - entrambi sono migliori in termini di velocità e utilizzo della memoria rispetto a worst-fit

Strutture dati usate dal SO

- Il gestore della memoria principale mantiene aggiornata una lista delle partizioni disponibili
- Si tratta di una lista di elementi che contengono indirizzo iniziale e dimensione delle partizioni disponibili
  - l'indirizzo iniziale della prima partizione disponibile è mantenuto in una locazione di memoria prestabilita
  - le prime due locazioni di ogni partizione disponibile contengono
    - la dimensione della partizione in questione
    - l'indirizzo iniziale della partizione disponibile successiva
- La lista viene mantenuta ordinata:
  - Per dimensioni crescenti delle partizioni (alg. best-fit)
    - L'accorpamento delle partizioni richiede la scansione dell'intera lista
  - Per indirizzi crescenti delle partizioni
    - Facilita l'accorpamento di partizioni disponibili adiacenti

Protezione e condivisione delle informazioni

- La protezione tra processi allocati contemporaneamente in memoria si può realizzare con supporto HW
  - Es. i 2 registri della CPU detti base e limite, gestiti dal SO, che contengono l'indirizzo iniziale e la dimensione della partizione assegnata al processo
    - Ogni processo ha un'area di memoria completamente privata determinata dai valori dei due registri (salvati nel PCB)
    - I processi sono così isolati l'uno dall'altro e dal SO
    - Il SO ha il bit di rilocazione a 0 (nel registro PS), cosicché può accedere tutta la memoria
- Nessuna forma di condivisione è possibile poiché lo spazio virtuale è unico (e quindi richiede allocazione di memoria fisica contigua)

Partizioni variabili: considerazioni

- Vantaggi rispetto allo schema a partizioni fisse
  - flessibilità
  - eliminazione del problema della frammentazione interna
- Problema: frammentazione esterna (alle singole partizioni)
  - Si presenta allorquando, a fronte di una richiesta, le partizioni disponibili sono ciascuna di dimensione inferiore alla quantità di memoria richiesta sebbene la somma delle loro dimensioni sia sufficiente a soddisfarla
  - Regola del 50%: l'analisi statistica dell'algoritmo first-fit rivela che, pur con alcune ottimizzazioni, per *n* blocchi assegnati, si perdono altri 0,5 *n* blocchi a causa della frammentazione esterna, ciò significa che potrebbe essere inutilizzabile un terzo della memoria
- Soluzioni per ridurre la frammentazione esterna:
  - Compattazione: riordinare il contenuto della memoria per riunire la memoria libera in un unico blocco; soluzione ideale ma non applicabile poiché richiede rilocazione dinamica
  - Rinunciare all'assegnazione di memoria contigua

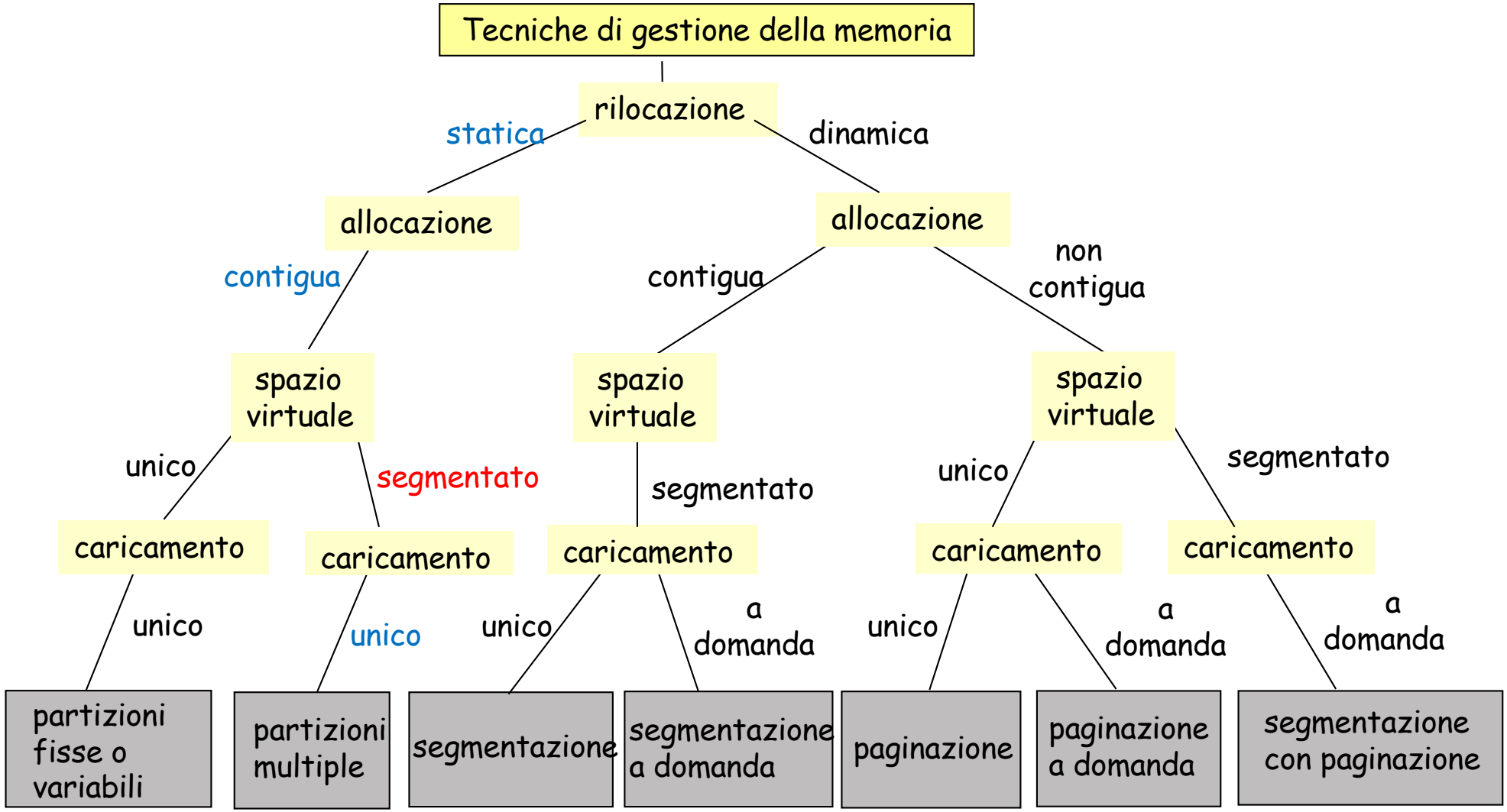
    ⇨ Partizioni multiple (ma anche Paginazione, se riloc. dinamica)

Partizioni multiple

Per ridurre la frammentazione esterna, e consentire la condivisione di codice/dati, si può segmentare lo spazio virtuale (es. 4 segmenti: codice, dati, stack e heap)

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
STATICA	CONTIGUA	SEGMENTATO	UNICO

Partizioni multiple



Partizioni multiple

- Ogni segmento virtuale, pur essendo ancora singolarmente allocato in locazioni fisiche contigue (di una singola area di memoria principale), può essere allocato in maniera indipendente dagli altri segmenti del programma
- Vantaggi
  - Condivisione di codice e dati
  - Riduzione degli effetti negativi della frammentazione esterna e agevolazione dell'allocazione dei processi
    - Anzicché un'unica partizione di grandi dimensioni, si usa un certo numero di partizioni di dimensioni più piccole
- Svantaggi
  - Maggiore complessità del linker che deve gestire più segmenti nella memoria virtuale di un programma

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

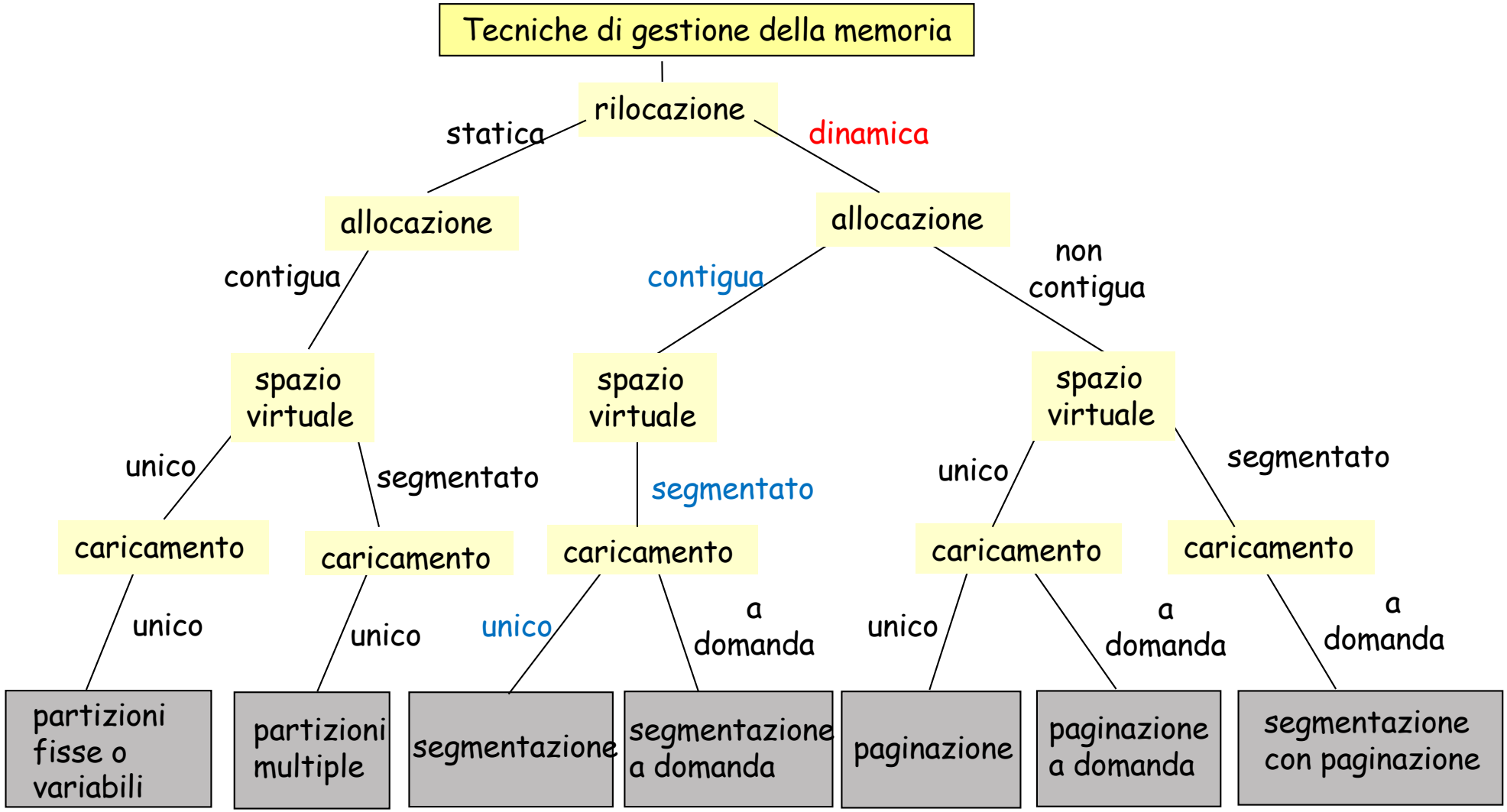
Segmentazione

La tecnica della segmentazione si ottiene a partire dalla tecnica a partizioni multiple consentendo la rilocazione dinamica delle partizioni (è infatti anche detta tecnica delle partizioni rilocabili)

- Consente la compattazione (riordinare il contenuto della memoria per riunire la memoria libera in un unico blocco)

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
DINAMICA	CONTIGUA	SEGMENTATO	UNICO

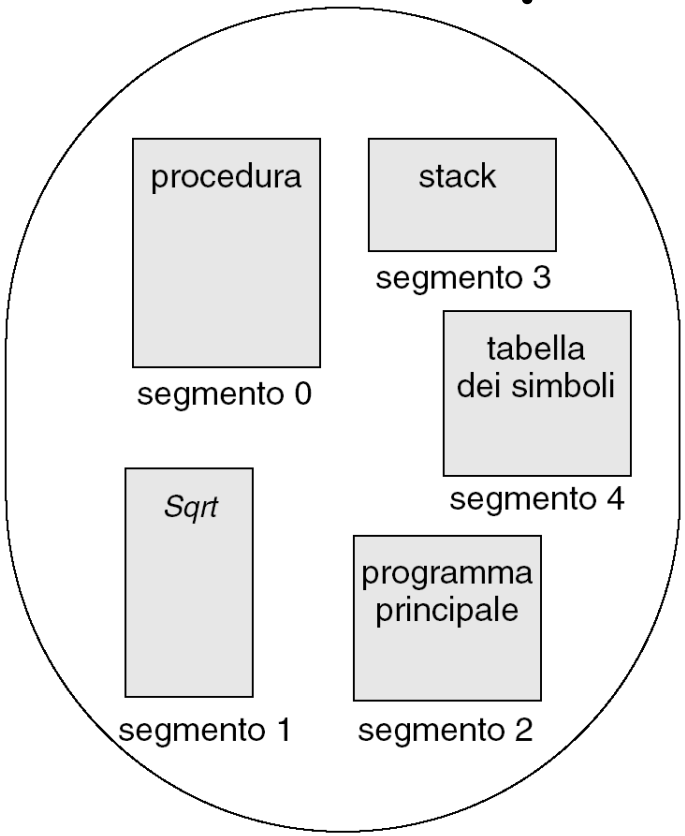
Segmentazione



Spazio virtuale e programma

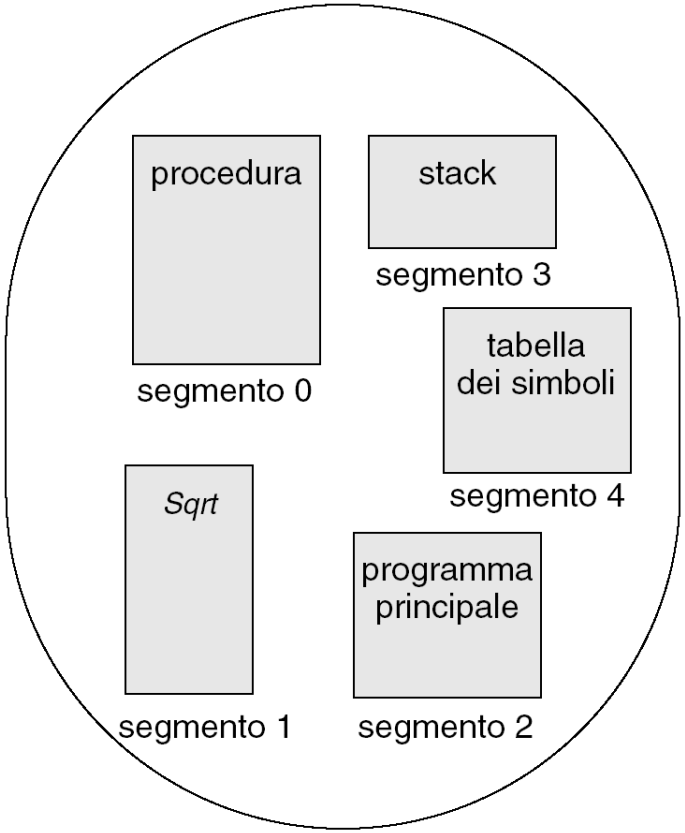
- Uno spazio virtuale segmentato non è vincolato ad essere suddiviso nei soli elementi codice, dati, stack e heap
- Lo spazio virtuale può assumere una struttura che riflette quella del programma in esso allocato, così com'è percepita dal programmatore
- Un programma è un insieme di unità logiche, es.
  - programma principale
  - procedure
  - funzioni
  - metodi
  - oggetti
  - variabili locali e globali
  - stack
  - tabelle di simboli
  - ...

Programma dal punto di vista del programmatore



Spazio di indirizzi logico

Vista logica della segmentazione



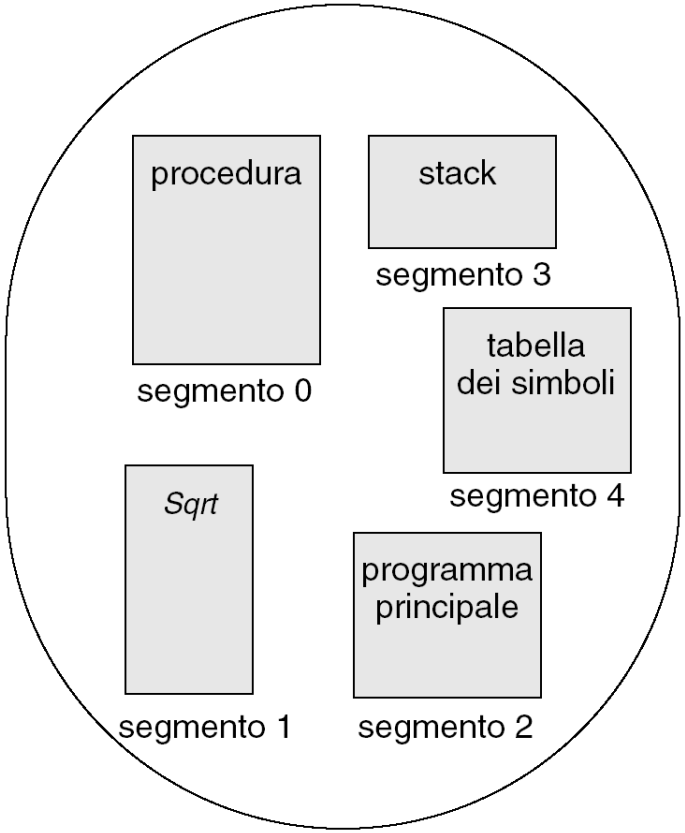
Spazio di indirizzi logico

memoria fisica

Traduzione degli indirizzi

- Per tradurre a run-time gli indirizzi logici generati da un processo nei corrispondenti indirizzi fisici, bisogna poter determinare il segmento a cui un indirizzo appartiene
- Un indirizzo logico *x* ha due componenti *<sg, of>*
  - *sg* è il numero di segmento
  - *of* è lo scostamento (*offset*) dall'inizio del segmento
- Per via dell'elevato numero di segmenti che, in generale, fanno parte dello spazio virtuale di un processo, (es. già Intel 386 poteva gestire uno spazio virtuale di 2^{14} segmenti!), non è possibile mantenere una coppia di registri base – limite per ogni segmento
- Tabella dei segmenti (allocata nella memoria fisica del processo): l'elemento di indice *n* contiene l'indirizzo fisico iniziale (base) e la dimensione del segmento *n*
  - Ogni elemento della tabella è detto descrittore di segmento

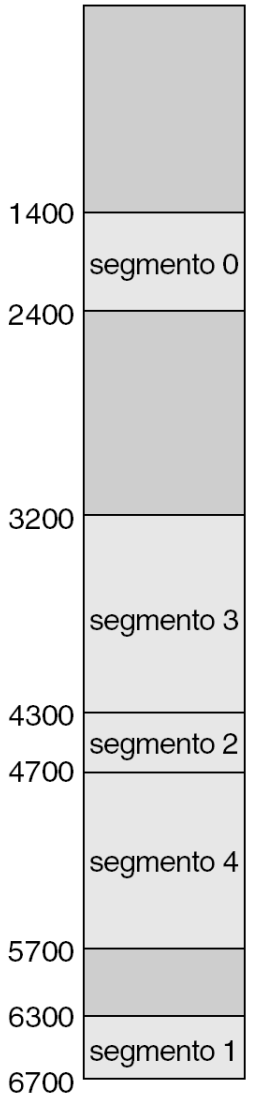
Tabella dei segmenti



Spazio di indirizzi logico

	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

tabella
dei segmenti

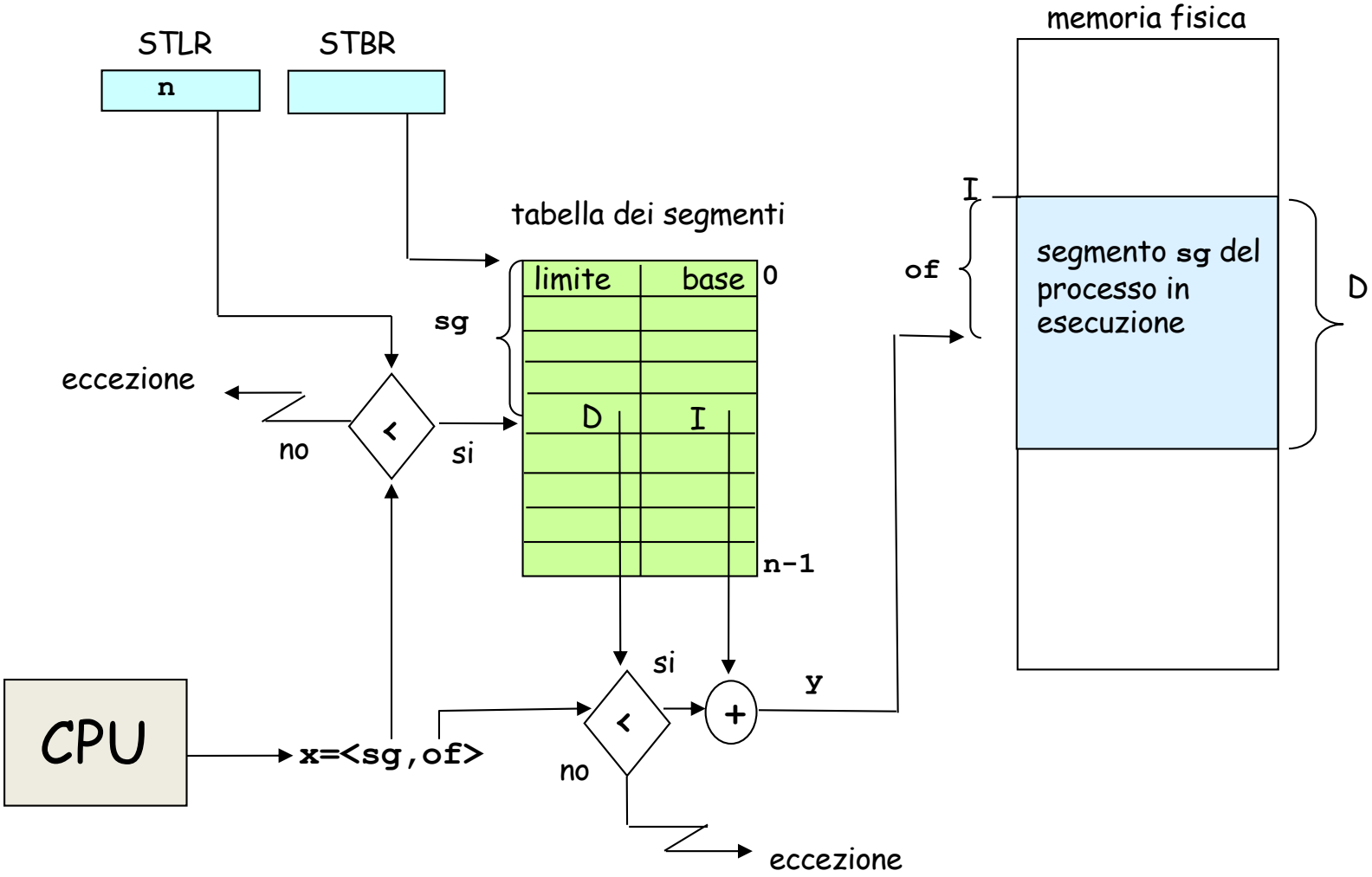


memoria fisica

Traduzione degli indirizzi

- Per individuare la locazione in memoria della tabella dei segmenti di un processo, si fa uso di due registri
  - STBR (*segment table base register*) contiene l'indirizzo della locazione di memoria fisica della tabella dei segmenti
  - STLR (*segment table limit register*) contiene il numero di segmenti del processo (cioè il numero degli elementi della sua tabella dei segmenti)
- Traduzione:
  - indirizzo logico *x* = <*sg*, *of*>
    - *sg* = numero di segmento
    - *of* = scostamento (*offset*) dall'inizio del segmento
  - *sg* è usato come indice della tabella dei segmenti per selezionare il descrittore di segmento che contiene l'indirizzo fisico iniziale e la dimensione del segmento
  - Se lo scostamento *of* è **minore** della dimensione del segmento, l'indirizzo fisico corrispondente a *x* è ottenuto **sommando** *of* all'indirizzo fisico iniziale del segmento

Traduzione degli indirizzi (a carico della MMU)



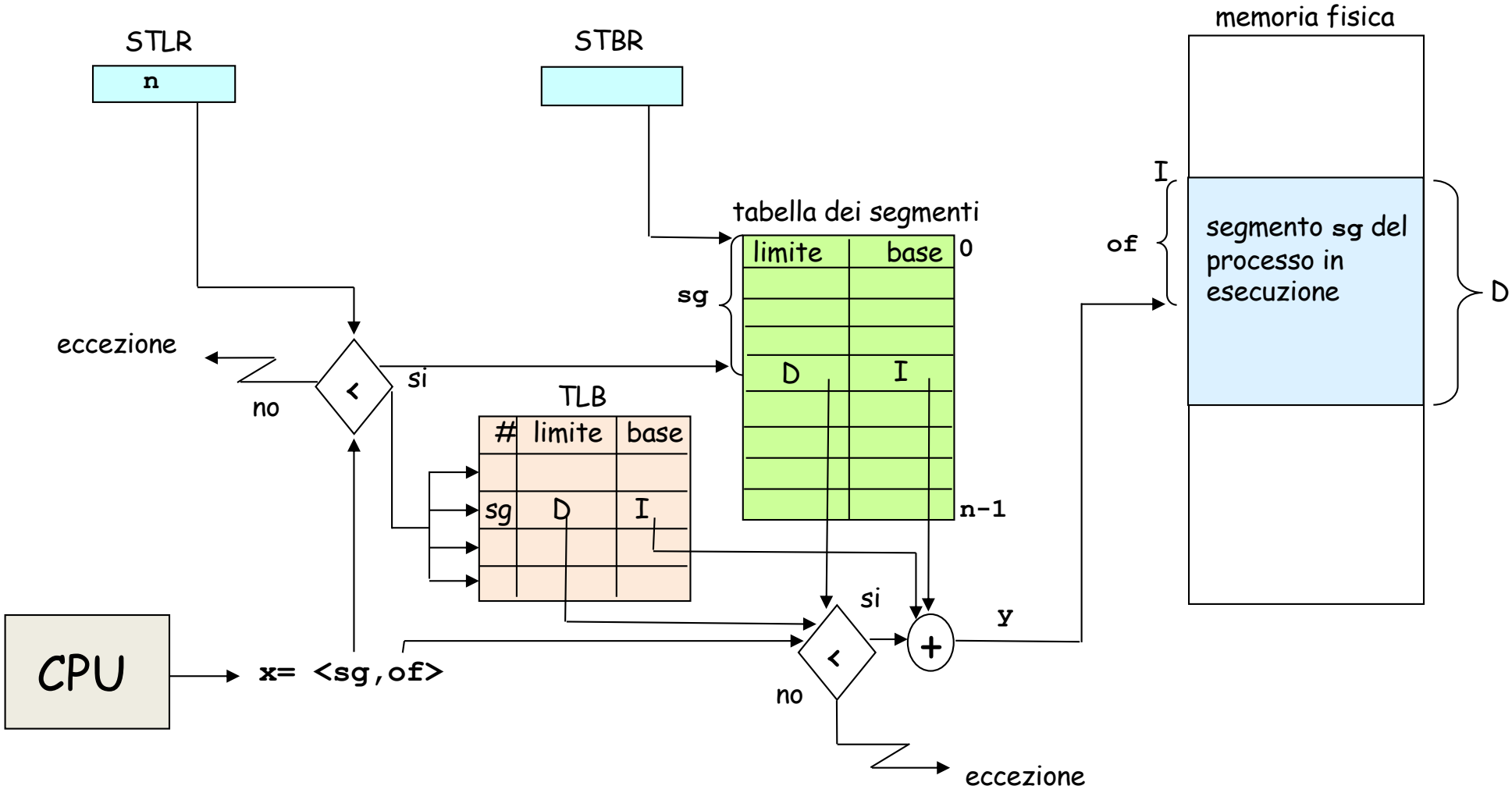
Inconveniente della traduzione

- Perdita di efficienza: per ogni indirizzo generato dalla CPU è necessario fare 2 accessi in memoria
  - uno per accedere alla tabella dei segmenti e trasformare l'indirizzo da virtuale a fisico
  - uno per accedere effettivamente l'informazione voluta
- Soluzione:
  - la MMU utilizza una piccola cache (HW) contenente le informazioni sulle traduzioni più recenti
  - si tratta di alcuni registri associativi (tipicamente tra 8 e 4096) detti TLB (*Translation Lookaside Buffer*)

Traduzione con TLB

- Ciascun registro memorizza un numero di segmento ed i corrispondenti valori base e limite (cioè un descrittore di segmento)
  - Es. se i registri sono 64, essi memorizzano le informazioni relative agli ultimi 64 segmenti acceduti
- Quando la CPU genera un riferimento in memoria, per tradurre l'indirizzo virtuale, l'HW avvia la ricerca nel TLB e, se fallisce, nella tabella dei segmenti in memoria
  - La ricerca in TLB, essendo questa una memoria associativa, avviene confrontando in parallelo il numero di segmento riferito con il numero di segmento memorizzato in ogni registro
  - Se la ricerca in TLB ha successo, la traduzione procede utilizzando i corrispondenti valori base e limite
  - Altrimenti, la traduzione utilizza i valori base e limite restituiti dalla ricerca nella tabella dei segmenti
    - A seconda di come è organizzata la tabella dei segmenti, tale ricerca può comportare più di un accesso in memoria
    - Le informazioni restituite dalla ricerca saranno inserite nel TLB eventualmente al posto di quelle di un altro segmento scelto ad esempio in maniera casuale, o con politica round-robin, o least recently used

Traduzione degli indirizzi con TLB



Traduzione con TLB

- L'*hit rate* (tasso di successo) tipico di un TLB è di oltre il 99% ed il suo tempo di accesso è solitamente inferiore al 10% del tempo di accesso alla memoria
- Ottimizzazione: in alcune architetture, la ricerca nel TLB e nella memoria procede in parallelo
- Complicazioni dell'uso del TLB
  - Quando avviene il context switch, bisogna invalidare il contenuto del TLB (TLB flush) perché le informazioni memorizzate non sono valide per il nuovo processo; di solito ciò viene fatto dall'HW quando cambia il valore del STBR
    - Ciò però comporta un iniziale alto numero di TLB *miss*
  - Perciò, in alternativa al TLB flush, alcuni sistemi mantengono nel TLB informazioni relative a processi differenti e, per proteggere lo spazio di indirizzi dei processi, ogni elemento del TLB contiene anche un valore per identificare lo spazio virtuale (*Address-Space Identifier*, ASID) del processo autorizzato ad accedervi (a cui il segmento appartiene)
  - Se cambia l'allocazione della memoria per il processo corrente (es. un segmento viene spostato), bisogna invalidare alcuni registri del TLB; anche tale operazione è supportata dall'HW

Tempo effettivo di accesso alla memoria

- Vediamo come l'uso del TLB rende più efficiente la traduzione degli indirizzi
- Sia p, con $0 \leq p \leq 1$, la probabilità che la ricerca in TLB abbia successo (TLB hit)
  - (1 – p) è la probabilità che ci sia un TLB miss
  - se p = 1 ogni accesso in TLB ha successo
- Tempo effettivo di accesso alla memoria (Effective Memory Access Time, EMAT)

  EMAT = p x (tempo accesso TLB + tempo accesso memoria)

  + (1 - p) x (2 x tempo accesso memoria

  [+ tempo accesso TLB])

  Il secondo tempo di accesso al TLB manca se l'architettura consente la ricerca in parallelo nel TLB e nella memoria

Calcolo di EMAT

- Supponiamo che
  - tempo di accesso alla memoria = 100 nanosecondi (tipicamente è compreso tra 10 e 200 nanosecondi)
  - tempo di accesso al TLB = 10 nanosecondi (10% del tempo di accesso alla memoria)
  - p = 99% (= probabilità che la ricerca in TLB abbia successo)
- Allora, istanziando con i dati la formula precedente

EMAT = p x (tempo accesso TLB + tempo accesso memoria)
+ (1 - p) x (2 x tempo accesso memoria
[+ tempo accesso TLB])

abbiamo

EMAT = 0,99 x (100 + 10) + 0,01 x (2 x 100 + 10)
= 108,9 + 2,1 = 111 nanosecondi

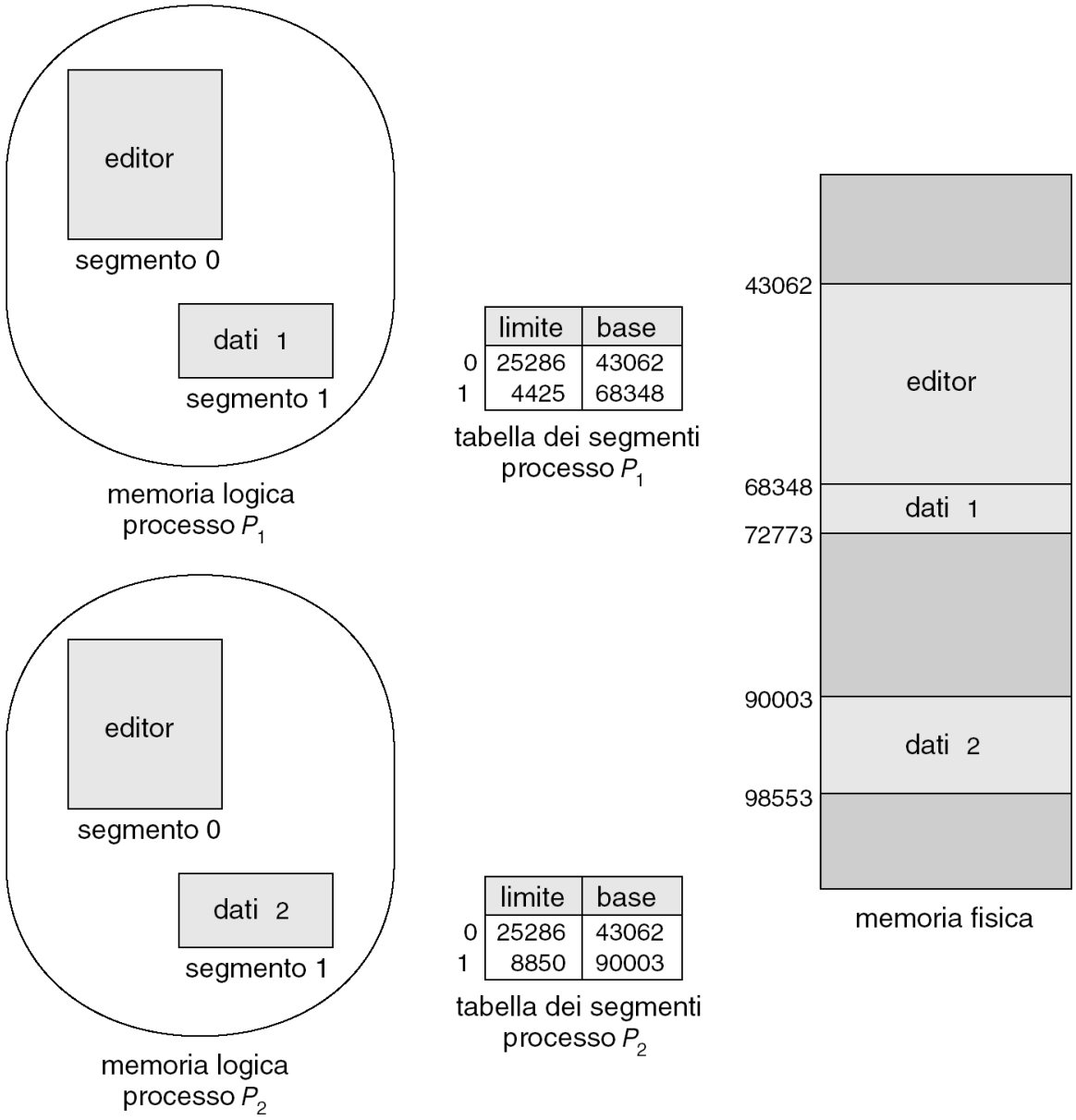
Informazioni nel PCB

- Relativamente alla gestione della memoria, nel PCB di ogni processo c'è un campo che contiene due informazioni:
  - l'indirizzo in memoria della tabella dei segmenti del processo
  - il numero di segmenti dello spazio logico del processo
- Tali valori sono usati al momento in cui la CPU è assegnata al processo per inizializzare i registri STBR e STLR

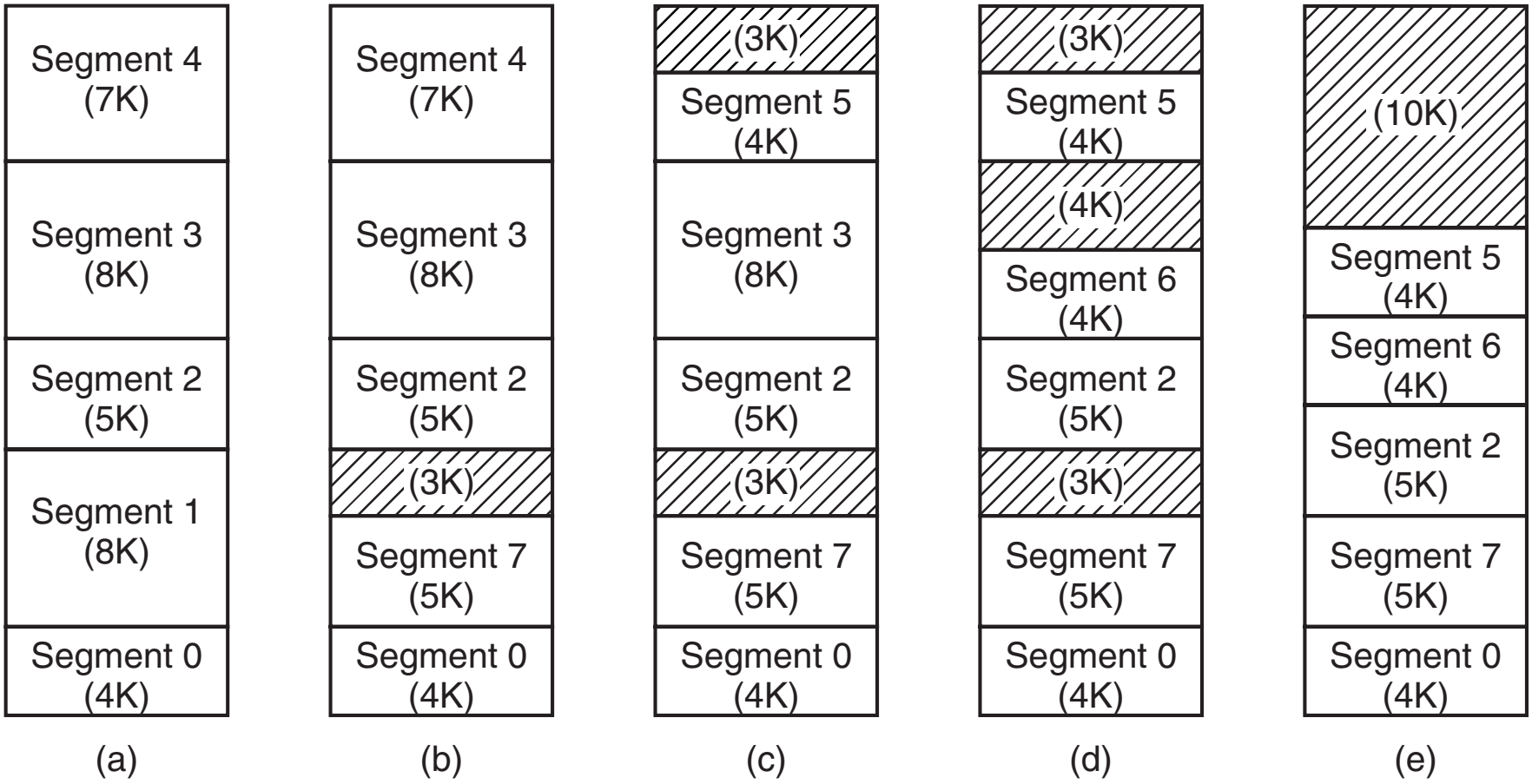
Vantaggi della segmentazione

- Protezione: si effettuano 3 diversi controlli quando si traduce *<sg, of>*
  - *sg* < contenuto di STLR
  - *of* < dimensione del segmento
  - controllo dei diritti di accesso al segmento
    - ogni elemento della tabella dei segmenti contiene un terzo campo contenente dei bit che rappresentano diritti di accesso: R, W, X, …
- Condivisione: processi diversi possono condividere segmenti, purché i segmenti condivisi abbiano lo stesso indice negli spazi logici di tutti i processi che li condividono
  - Infatti, gli indirizzi di memoria contenuti nei segmenti sono logici
  - Quindi quelli contenuti in un segmento condiviso fanno riferimento ad informazioni che sono allocate in posizioni identiche negli spazi logici dei processi che le condividono
  - Di conseguenza, se fanno riferimento a posizioni di segmenti condivisi questi devono occupare le stesse posizioni
- Si possono spostare segmenti per compattare la memoria ed eliminare la frammentazione esterna

Condivisione dei segmenti



Eliminazione della frammentazione esterna tramite compattazione dei segmenti

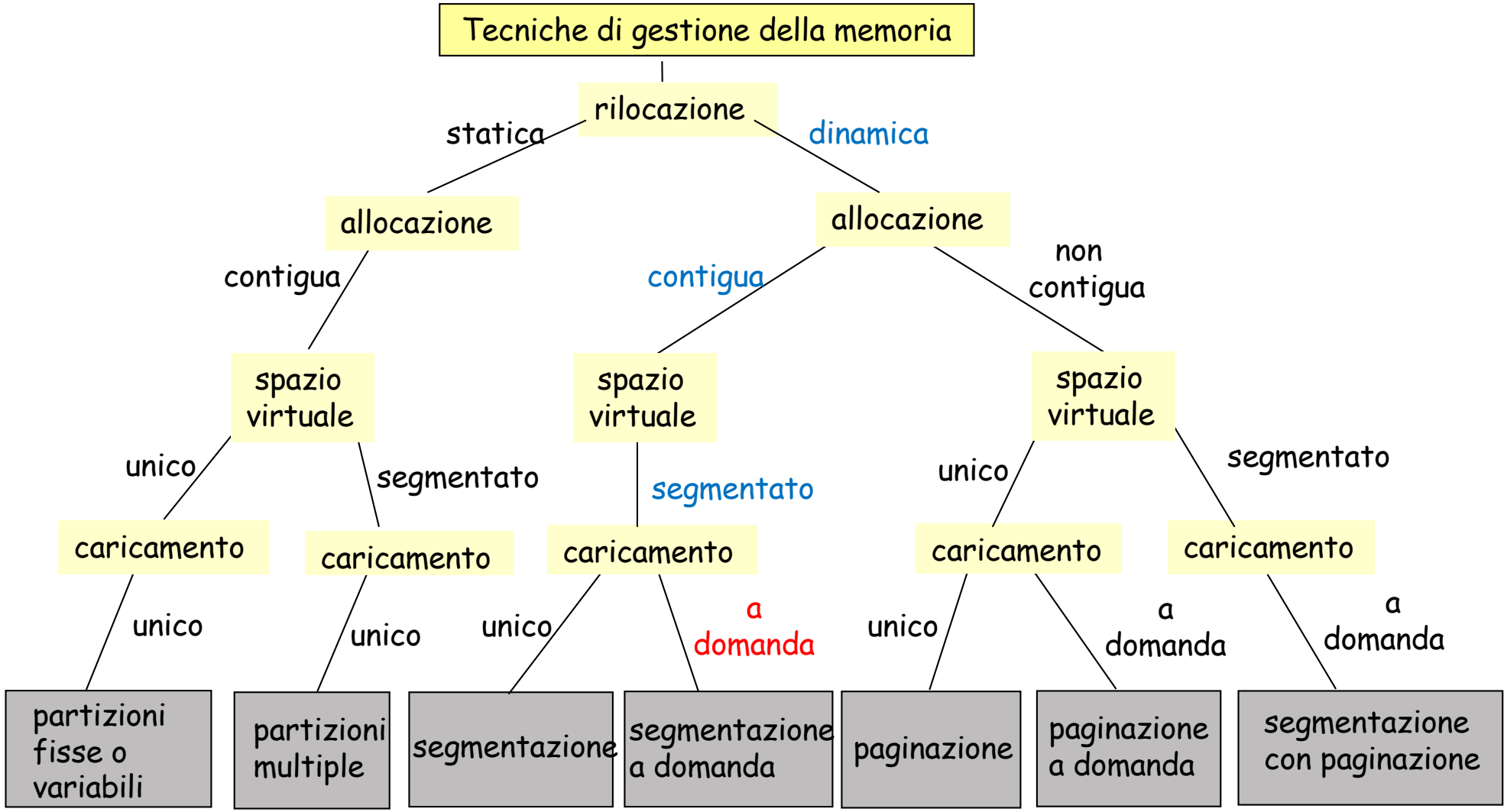


Segmentazione a domanda

Lo spazio virtuale di un processo può essere parzialmente caricato in memoria fisica

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
DINAMICA	CONTIGUA	SEGMENTATO	A DOMANDA

Segmentazione a domanda



Vantaggi della segmentazione a domanda

Lo spazio logico di un processo può essere parzialmente caricato in memoria fisica

- Lo spazio logico di un processo può essere più grande della memoria fisica
- Lo swapping riguarda singoli segmenti di un processo, non il suo intero spazio logico
- È possibile schedulare per l'uso della CPU anche un processo che non ha segmenti in memoria fisica

Rilocazione degli indirizzi

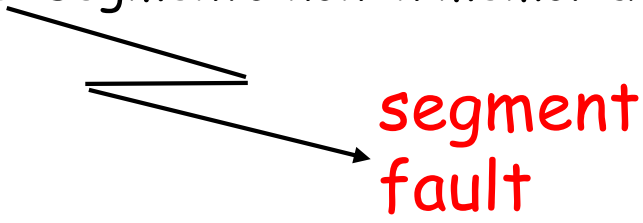
- Più complessa perché il SO deve gestire anche il caso in cui viene generato l'indirizzo logico di un segmento che non è già caricato in memoria
- Per ogni descrittore di segmento, cioè per ogni elemento nella tabella dei segmenti, si usa uno specifico bit di controllo P (bit di presenza)
  - è a 1 se il segmento è presente in memoria fisica: i campi *base* e *limite* contengono valori significativi
  - è a 0 se il segmento non è presente: se si genera un indirizzo logico che contiene l'indice del segmento, viene lanciata un'interruzione *segment fault*
- La routine di gestione dell'interruzione segment fault si occuperà di caricare in memoria fisica il segmento corrispondente

Algoritmi di sostituzione

- Quando il SO deve caricare un segmento, se non c'è spazio in memoria fisica anche dopo una eventuale compattazione della memoria, deve scaricare nella swap area uno o più segmenti dello stesso processo o di altri processi
- La scelta del segmento da rimpiazzare, effettuata da un algoritmo di sostituzione, è un aspetto fondamentale per l'efficienza complessiva del sistema e può dare origine ad un overhead eccessivo
- Due ulteriori bit per ogni elemento nella tabella dei segmenti, i *bit di controllo* M ed U, permettono l'implementazione di tali algoritmi
  - U (bit di uso, o **di referenziazione**): 1 se il segmento è stato riferito di recente (serve a valutare la frequenza d'uso)
  - M (bit di modifica, o **dirty bit**): 1 se il segmento è stato modificato dopo essere stato caricato in memoria (se 0, quando il segmento subisce lo swap out non c'è bisogno di aggiornarne la copia su disco)
- Descriveremo gli algoritmi di sostituzione quando parleremo di **paginazione**

Descrittore di segmento

- R e W: diritti di accesso in lettura e scrittura (per scopi di protezione)
- M e U: bit di modifica e di uso (per gli algoritmi di sostituzione)
- P: bit di presenza (per la traduzione degli indirizzi)
  - P = 1: segmento valido
  - P = 0: segmento non in memoria



Gestione della memoria principale e virtuale

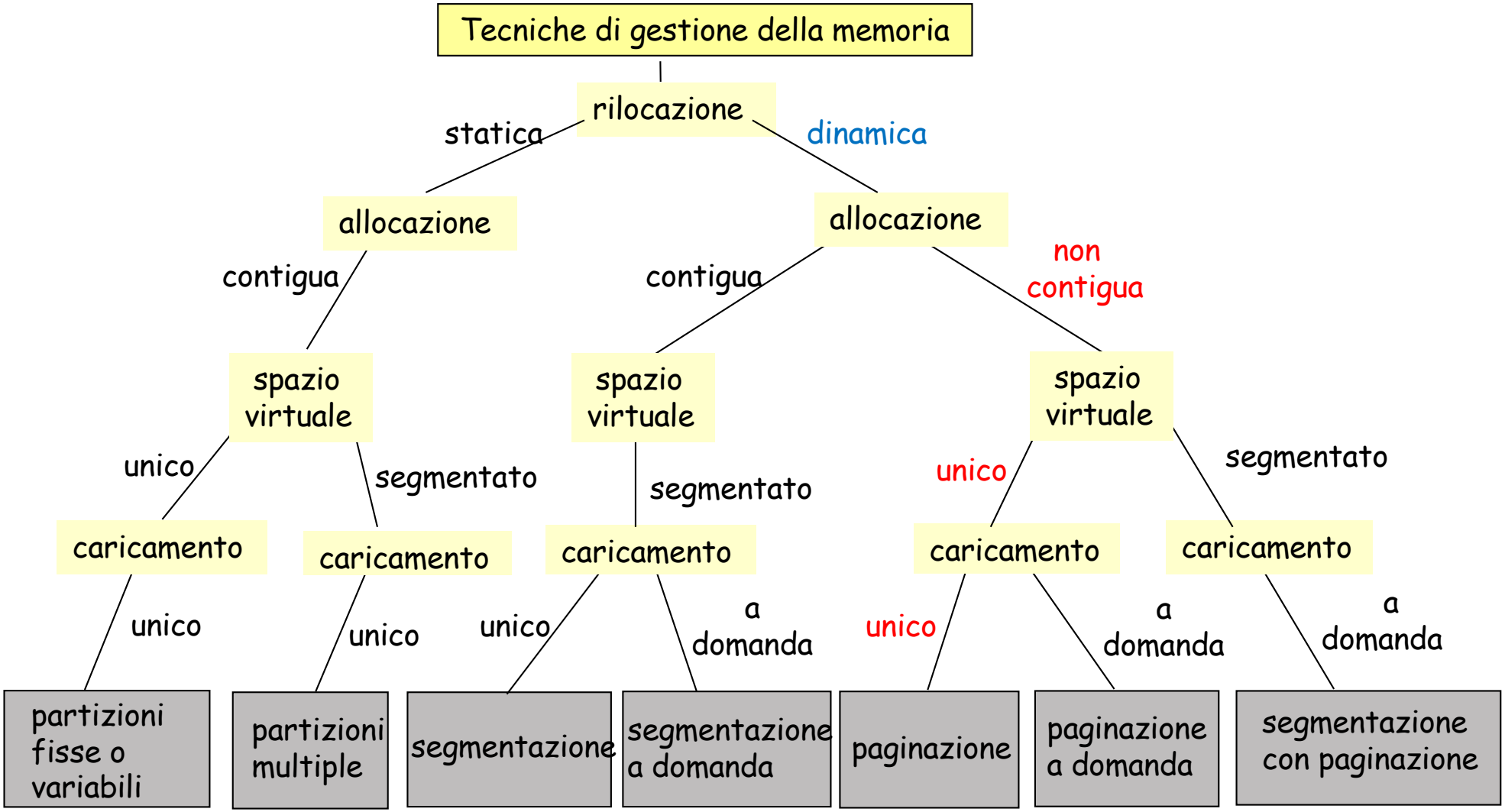
- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Paginazione

- Facciamo un passo indietro e torniamo a vedere lo spazio virtuale di un processo come un unico blocco
- Essendo la rilocazione dinamica, la memoria fisica invece non è necessariamente allocata in maniera contigua

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
DINAMICA	NON CONTIGUA	UNICO	UNICO

Paginazione



Paginazione

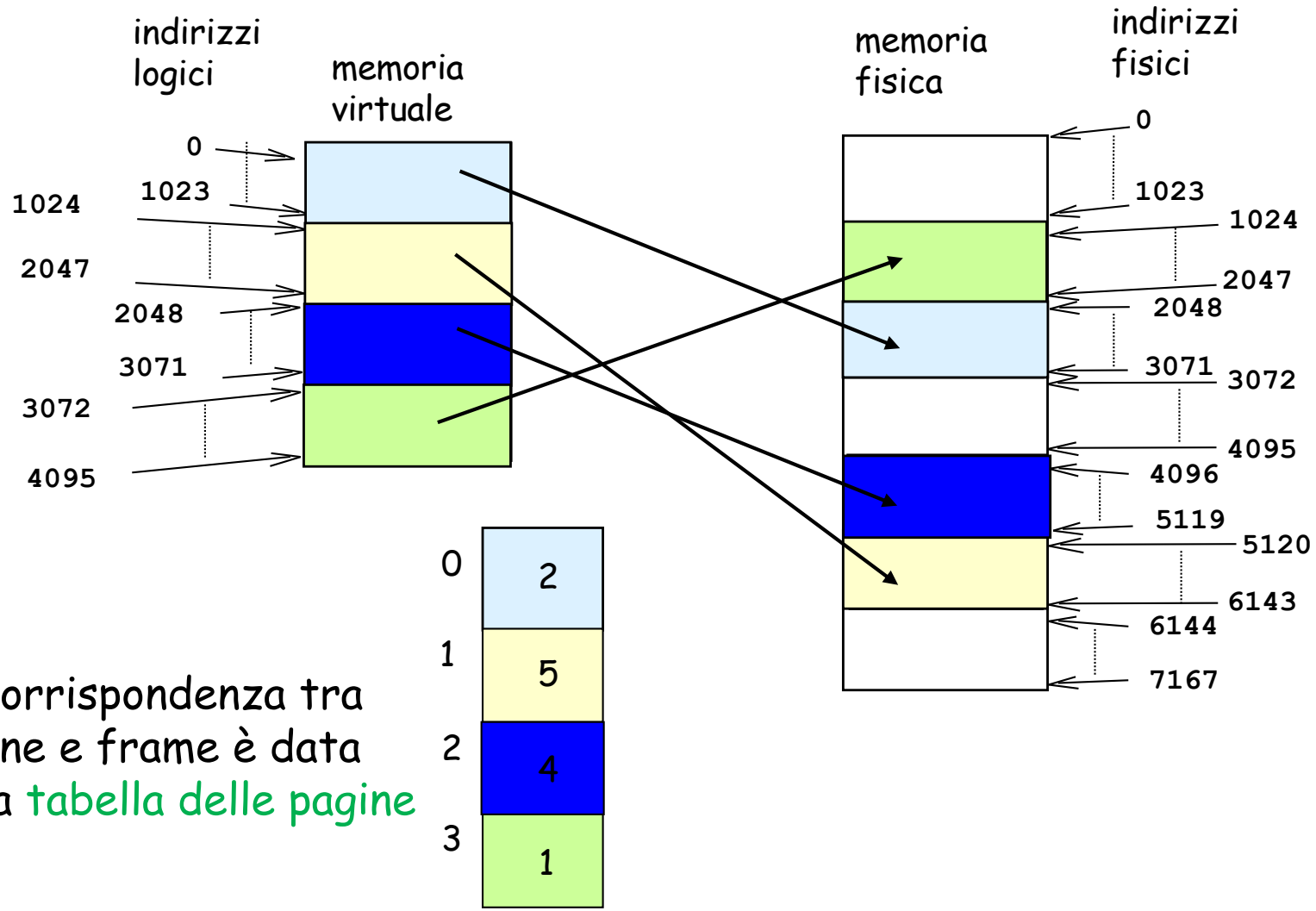
- Per eliminare alla radice il problema della frammentazione, bisognerebbe poter allocare in memoria fisica, in locazioni non necessariamente contigue, informazioni i cui indirizzi virtuali sono contigui
- La rilocazione dinamica rende concettualmente possibile tale soluzione
- Però, se le locazioni dello spazio virtuale fossero allocate singolarmente in locazioni fisiche indipendenti, servirebbe una tabella delle corrispondenze delle stesse dimensioni della memoria virtuale!
- Compromesso:
  - le locazioni dello spazio virtuale sono ripartite in blocchi
  - ogni blocco è allocato indipendentemente dagli altri

Pagine e frame

- Lo spazio virtuale è suddiviso in blocchi di locazioni contigue, detti pagine (virtuali), di dimensioni fisse
  - È conveniente che la dimensione sia una potenza di 2
- Lo spazio fisico è suddiviso in blocchi di indirizzi fisici, detti frame (o pagine fisiche)
  - Per ora assumiamo che pagine e frame abbiano le stesse dimensioni
- Ogni pagina viene allocata in un frame e pagine consecutive possono essere allocate in frame non necessariamente consecutivi
- Per tradurre un indirizzo virtuale nel corrispondente indirizzo fisico è necessario registrare in una tabella delle pagine la corrispondenza tra pagine e frame
  - Ogni elemento della tabella è detto descrittore di pagina
  - Ogni processo allocato in memoria principale possiede una propria tabella delle pagine

Esempio: tabella delle pagine

Esempio con dimensione delle pagine/frame di 1024 locazioni

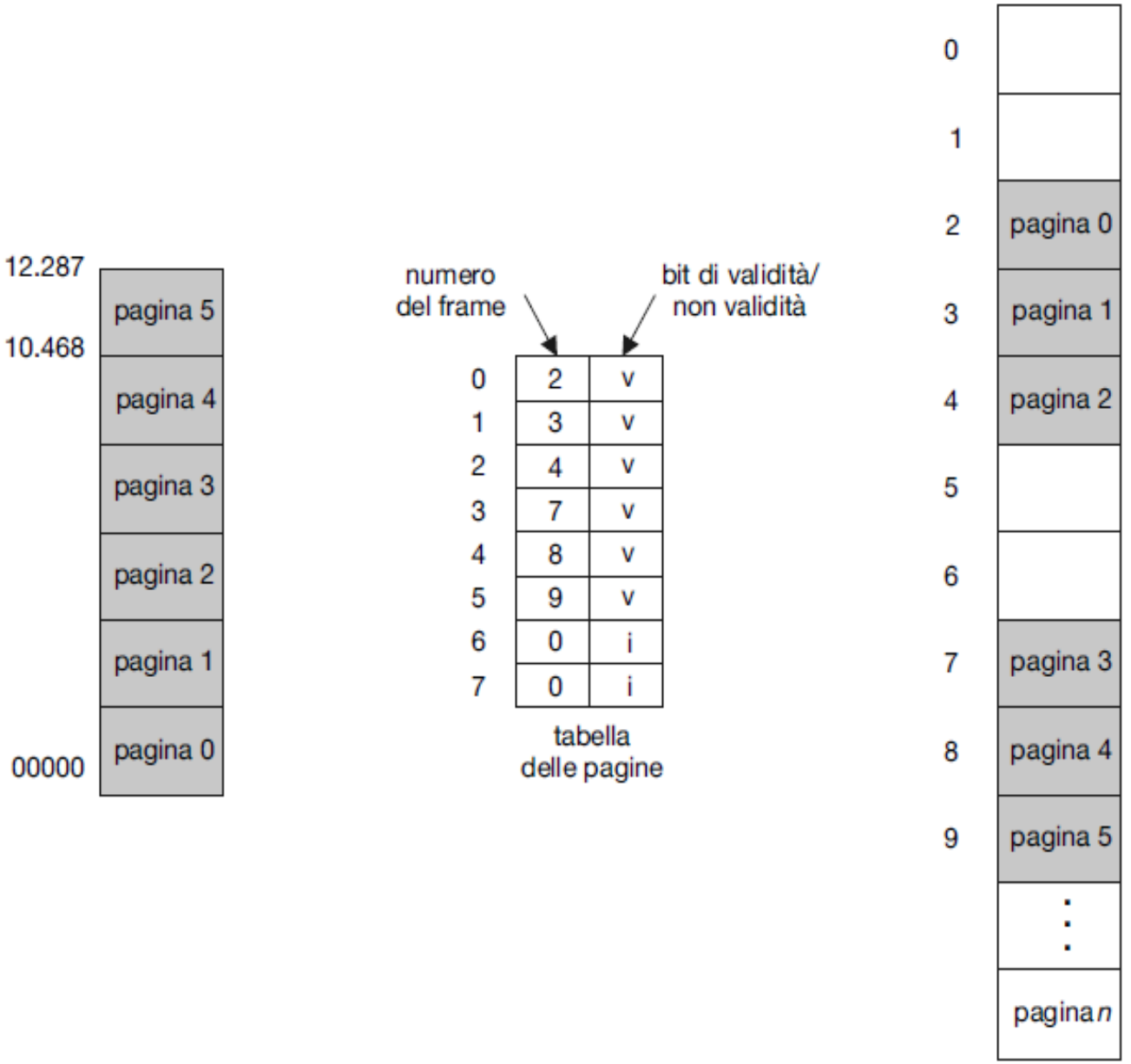


La corrispondenza tra pagine e frame è data dalla tabella delle pagine

Implementazione della tabella delle pagine

- La tabella delle pagine di un processo in memoria risiede anch'essa nella sua memoria principale
- Per individuare la locazione in memoria della tabella delle pagine di un processo, si fa uso di due registri
  - PTPR (*page table pointer register*) contiene l'indirizzo iniziale della locazione di memoria fisica della tabella delle pagine
  - PTLR (*page table length register*) contiene il numero delle pagine del processo (cioè il numero degli elementi della sua tabella delle pagine)
- PTLR può non esserci; in tal caso, le tabelle delle pagine di tutti i processi hanno la stessa lunghezza e, per protezione della memoria, si associa un bit di validità a ciascun elemento della tabella:
  - 1 (valido) indica che la pagina è nello spazio degli indirizzi logici del processo ed è quindi una pagina legale
  - 0 (non valido) indica che la pagina non è nello spazio degli indirizzi logici del processo

Protezione della memoria con bit di validità



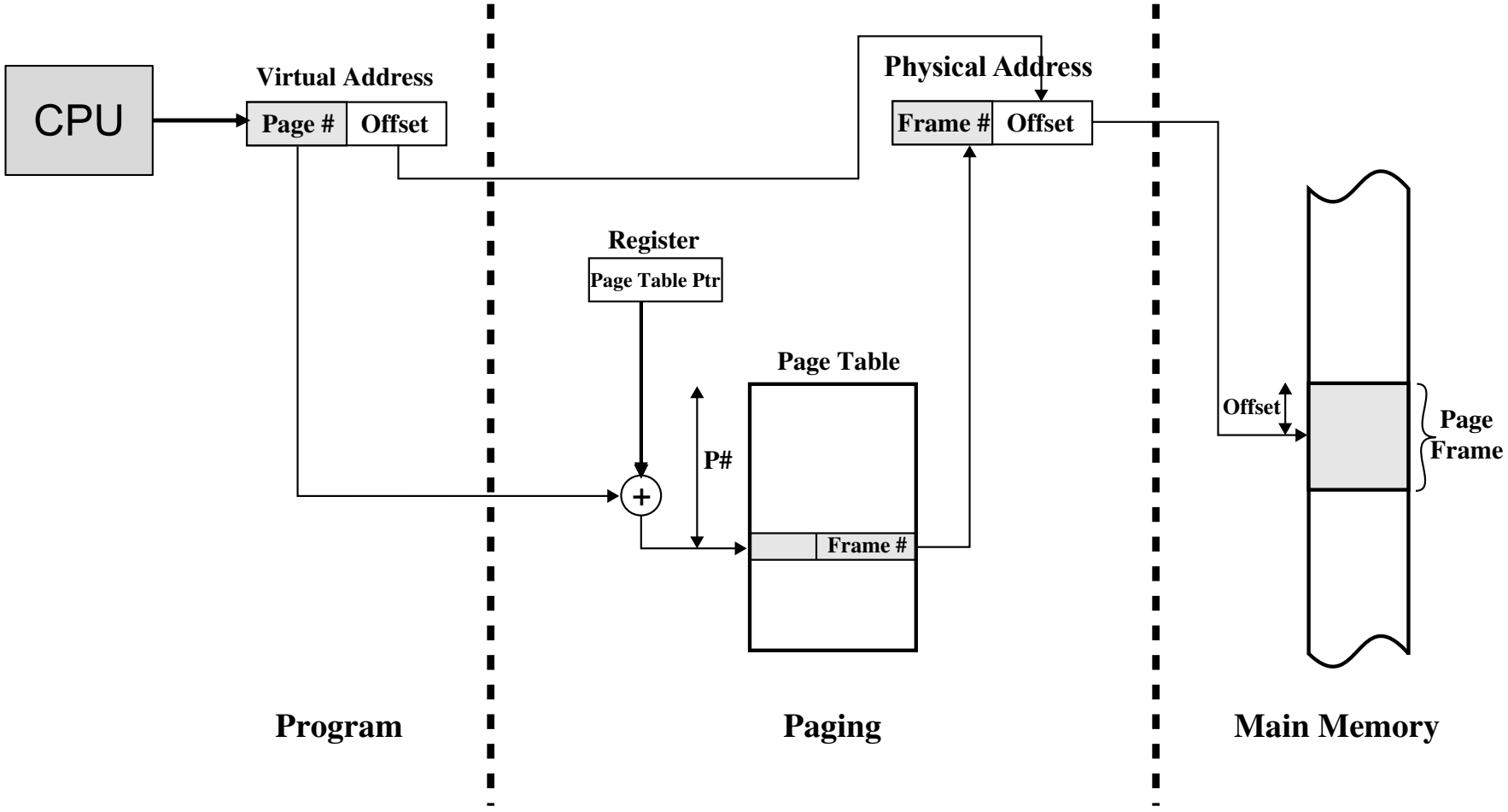
Traduzione degli indirizzi

- Per poter tradurre a run-time indirizzi virtuali generati dalla CPU in indirizzi fisici, bisogna determinare la pagina a cui un indirizzo appartiene
- Un indirizzo logico *x* deve essere scomposto in due componenti <*pg*, *of*>
  - *pg* è il numero di pagina
  - *of* è lo scostamento (offset) dall'inizio della pagina
- In pratica, se *d* è la dimensione delle singole pagine
  - *pg* è quoziente della divisione di *x* per *d*
  - *of* è il resto della divisione di *x* per *d*

Traduzione degli indirizzi

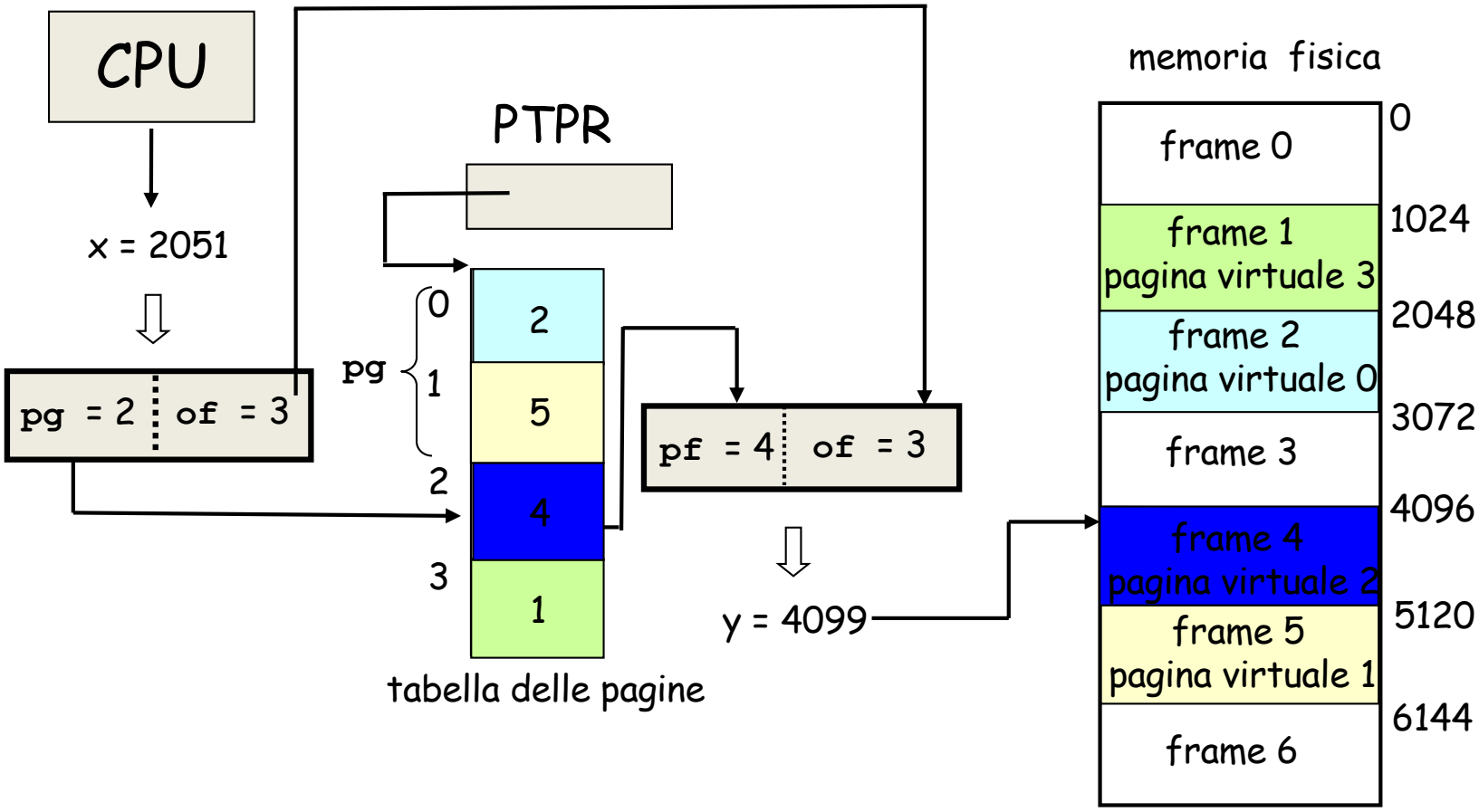
- Se la dimensione *d* delle pagine (e dei frame) è una potenza di 2, diciamo $d = 2^y$, dato che un indirizzo x è già espresso in forma binaria, resto e quoziente della divisione di x per d sono rispettivamente
  - *of* (resto): gli *y* bit meno significativi di *x*
  - *pg* (quoziente): i restanti bit di *x*
- Traduzione dell'indirizzo logico $x = pg \cdot of$ ($\cdot$ = concatenazione)
  - *pg* è usato come indice nella tabella della pagine per selezionare il descrittore che contiene l'indice del frame *fg* che ospita la pagina
  - L'indirizzo fisico corrispondente a *x* è $fg \cdot of$ (concatenazione dell'indice del frame *fg* che ospita la pagina con lo scostamento *of*)
- A differenza della segmentazione, **non** è necessario alcun confronto (pagine e frame hanno le stesse dimensioni) o somma
  - Basta fare una ricerca nella tabella delle pagine e (se la pagina è valida) una sostituzione di bit!

Schema di traduzione degli indirizzi



L'uso della tabella delle pagine è simile all'uso di una tabella di registri base (o di rilocazione), uno per ciascun frame

Schema di traduzione degli indirizzi

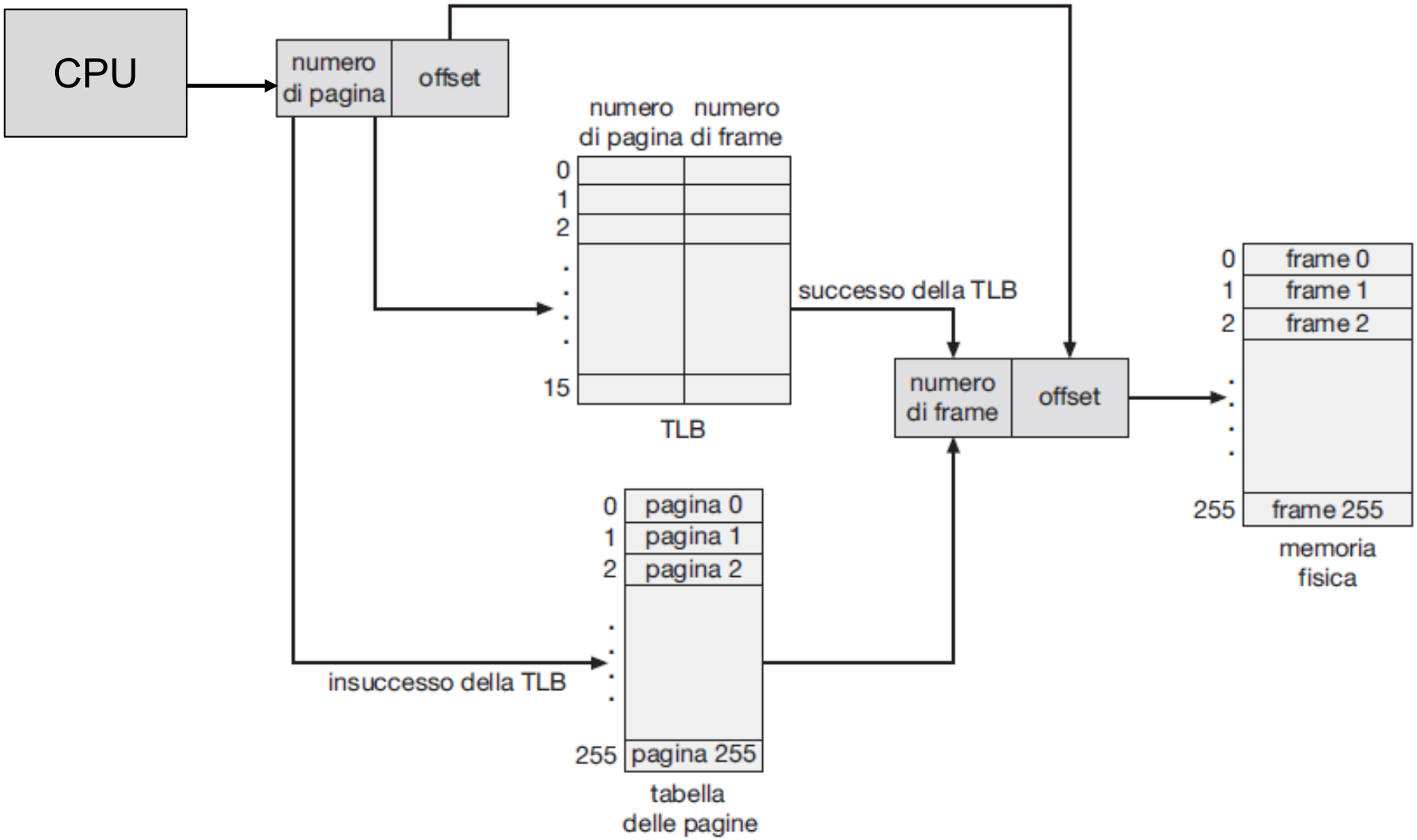


Esempio con dimensione delle pagine/frame di 1024 locazioni

Inconveniente della traduzione

- Perdita di efficienza: per ogni indirizzo generato dalla CPU è necessario fare 2 accessi in memoria
  - uno per accedere alla tabella delle pagine e trasformare l'indirizzo da virtuale a fisico
  - uno per accedere effettivamente all'informazione voluta
- Soluzione:
  - la MMU utilizza un TLB (come nella segmentazione)
- *Sul funzionamento e sui vantaggi e inconvenienti dell'uso del TLB valgono le considerazioni fatte nel caso della segmentazione*

Schema di traduzione degli indirizzi con TLB



Considerazioni

- Le CPU moderne possono fornire più livelli di TLB
- Il calcolo del tempo effettivo di accesso alla memoria

  EMAT = p x (tempo accesso TLB + tempo accesso memoria)
  + (1 - p) x (2 x tempo accesso memoria
  [+ tempo accesso TLB])

  diventa quindi più complicato
  - Es. Intel Core i7 ha un TLB L1 da 128 elementi per le istruzioni e un TLB L1 da 64 elementi per i dati
  - In caso di TLB miss in L1 sono necessari 6 cicli di CPU per cercare nel TLB L2 da 512 elementi
  - Un TLB miss in L2 richiede centinaia di cicli di CPU per cercare nella tabella delle pagine in memoria
- Per un funzionamento ottimale il progetto di un SO per una data piattaforma deve implementare la paginazione basandosi sull'architettura dei TLB della piattaforma

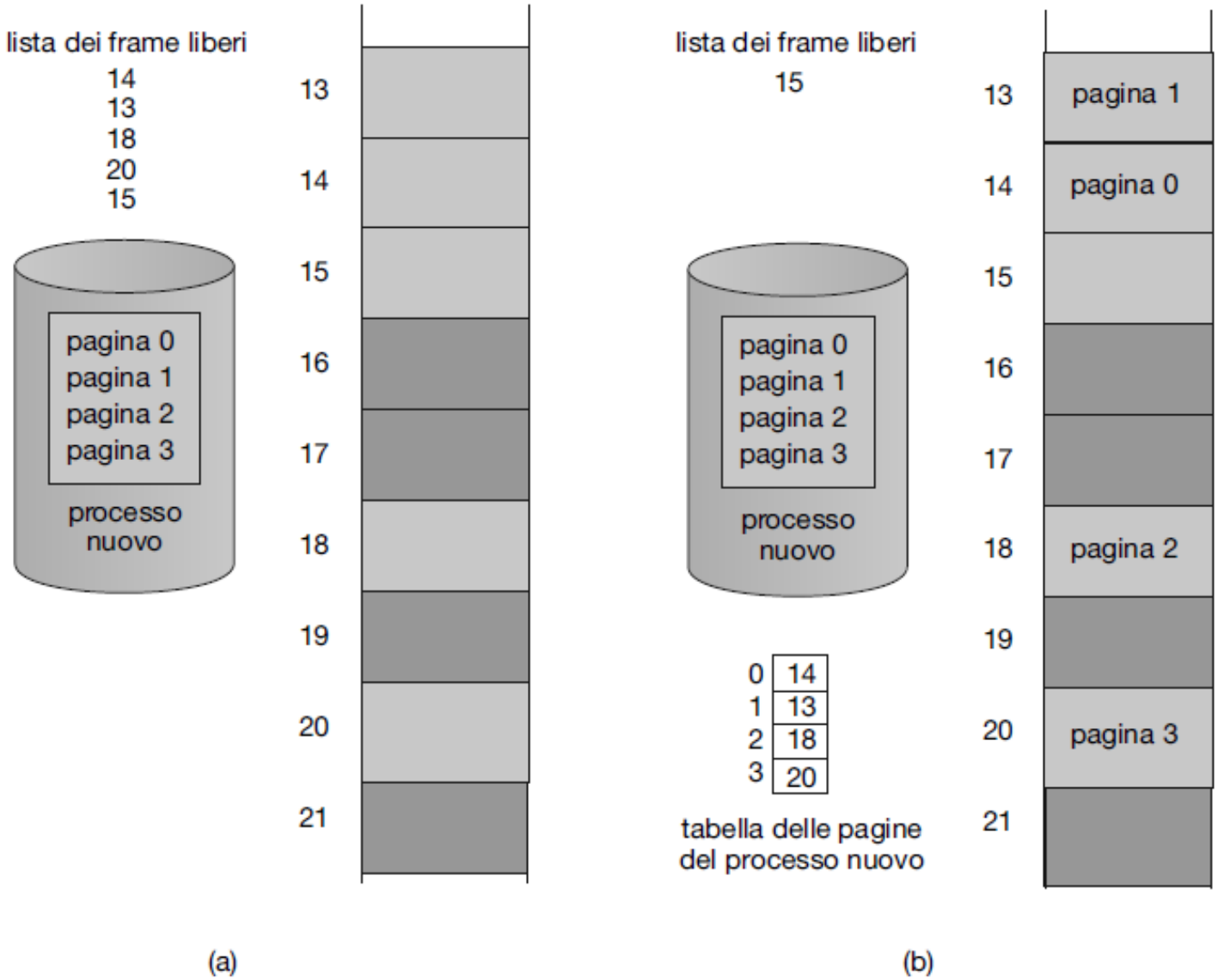
Informazioni nel PCB

- Relativamente alla gestione della memoria, nel PCB di ogni processo c'è un campo che contiene due informazioni:
  - l'indirizzo di memoria della tabella delle pagine del processo
  - il numero di pagine del suo spazio virtuale (che è solitamente anche la lunghezza della tabella delle pagine)
- Tali valori sono usati al momento in cui la CPU è assegnata al processo (context switch) per inizializzare i registri PTPR e PTLR

Tabella dei frame

- Il gestore della memoria fisica mantiene aggiornato l'elenco dei frame disponibili in una struttura dati, normalmente una tabella con tanti elementi quanti sono i frame
  - Tabella dei frame: ogni elemento della tabella indica se il frame corrispondente è libero; altrimenti contiene l'identitificatore del processo (o dei processi) a cui è allocato e l'indice della sua pagina virtuale ospitata
- Quando un processo dev'essere caricato in memoria sono richiesti al gestore tanti frame (non necessariamente consecutivi) quante sono le pagine del processo
  - N.B. il processo va caricato per intero
- Se non ci sono abbastanza frame, vengono scaricati (swap out) un certo numero di altri processi fino a liberare i frame necessari

Frame liberi



Prima dell'allocazione

Dopo l'allocazione

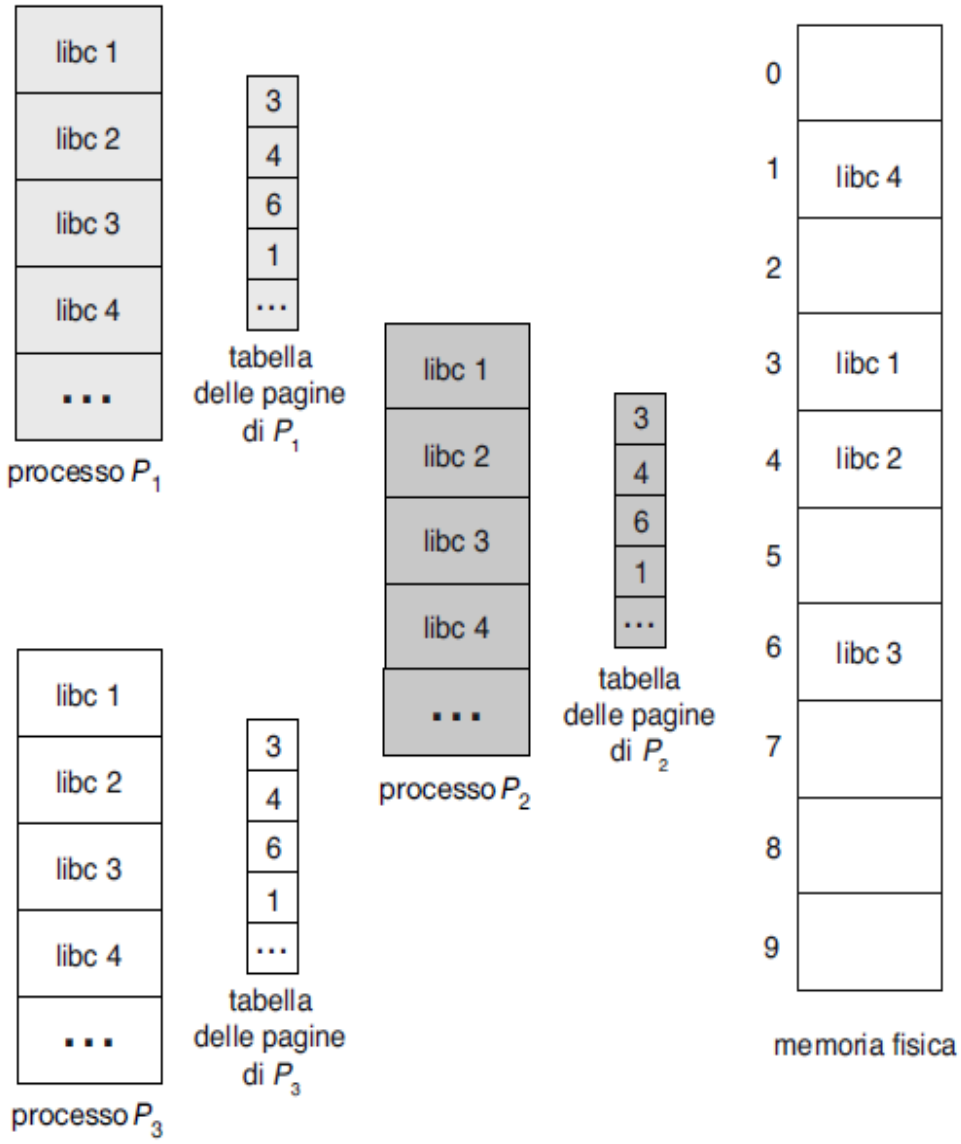
Considerazioni sulle tabelle della pagine

- La dimensione delle pagine, tipicamente compresa tra 4KB e 8KB (anche se in alcune architetture arriva a 2MB), è un parametro importante
  - Al suo diminuire, aumenta il numero delle pagine e quindi la dimensione della tabella delle pagine
  - Al suo aumentare, aumenta la frammentazione interna (l'ultima pagina di ogni processo mediamente è usata solo per metà)
- La maggior parte dei processi sono piccoli, quindi la maggior parte degli elementi della loro tabella delle pagine non sono utilizzati
- Persino i processi di grandi dimensioni utilizzano in maniera sparsa il loro spazio di indirizzi virtuali (ad es. codice in basso, stack in alto, il resto vuoto)
- Idealmente, ogni tabella delle pagine dovrebbe essere ospitata in una singola pagina
  - Problema: nelle architetture moderne, le tabelle delle pagine possono essere molto grandi
  - Soluzione: strutturare le tabelle delle pagine (**ne parleremo poi**)

Vantaggi della paginazione

- L'allocazione della memoria è semplificata: al momento del caricamento di un processo è sufficiente individuare tanti frame liberi quante sono le pagine di memoria virtuale del processo, ovunque tali frame risiedano in memoria
- Lo swapping dei processi è semplificato poiché tutte le sue pagine ed i frame di memoria hanno la stessa dimensione
- Protezione: ogni elemento della tabella delle pagine può contenere anche alcuni bit di protezione (es. R, W, X per i diritti di accesso)
- Condivisione possibile ma problematica: a differenza di un segmento, una pagina non individua un elemento logico del programma
  - Come per i segmenti condivisi, vale il vincolo che le pagine condivise devono occupare le stesse posizioni nei rispettivi spazi virtuali (cioè devono avere gli stessi indici)

Condivisione della libreria standard del C in ambiente paginato

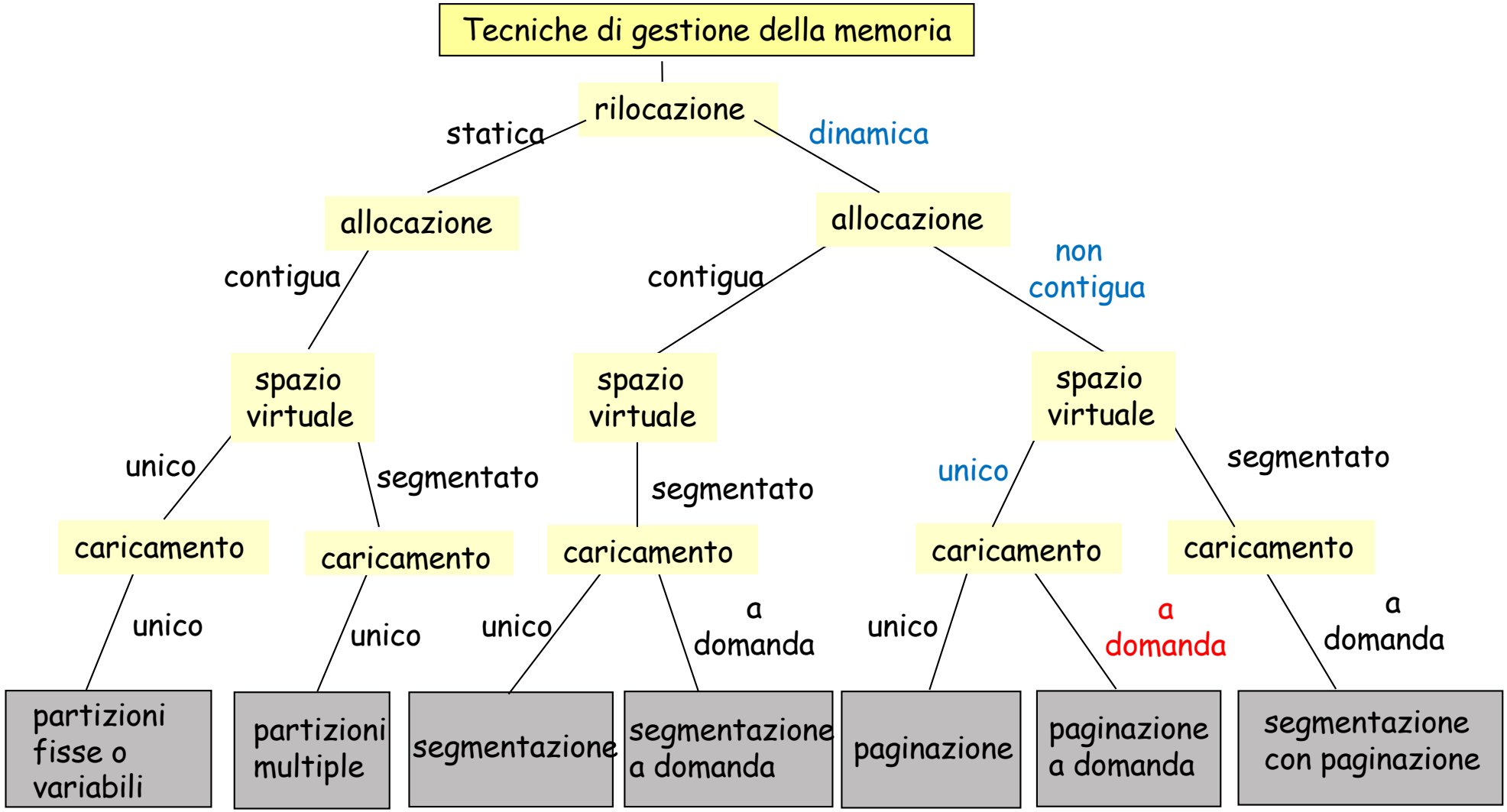


Paginazione a domanda

Lo spazio virtuale di un processo può essere parzialmente caricato in memoria fisica

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
DINAMICA	NON CONTIGUA	UNICO	A DOMANDA

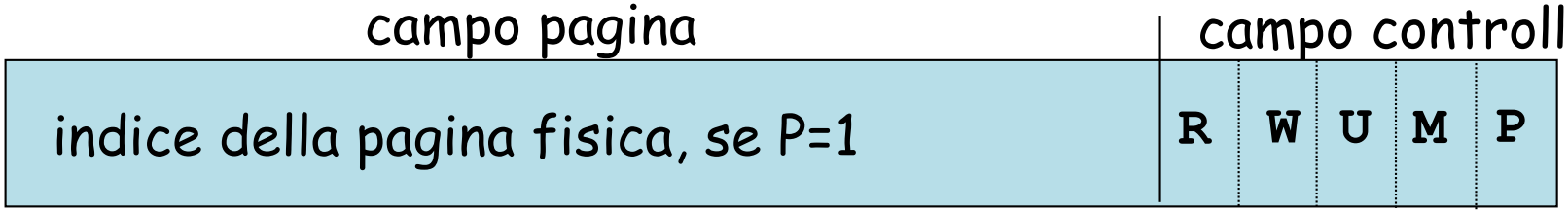
Paginazione a domanda



Paginazione a domanda

- Si usano gli stessi 3 bit di controllo usati nel caso della segmentazione: P (presenza), M (modifica) ed U (uso)
- Alla creazione di un processo, il suo spazio virtuale risiede completamente nella swap area su disco
- La tabella delle pagine è creata con tutti i bit P a 0
- Il processo può essere schedulato per l'uso della CPU anche se ancora nessuna delle sue pagine è stata caricata in memoria principale
- Il meccanismo HW di traduzione degli indirizzi genera un'interruzione *page fault* quando la CPU produce l'indirizzo virtuale di una pagina che non è stata ancora caricata (P = 0)
- La routine di gestione dell'interruzione carica la pagina prendendola dalla swap area eventualmente dopo aver richiamato un algoritmo di sostituzione
- L'istruzione che ha causato il page fault viene rieseguita

Descrittore di pagina



elemento della tabella delle pagine

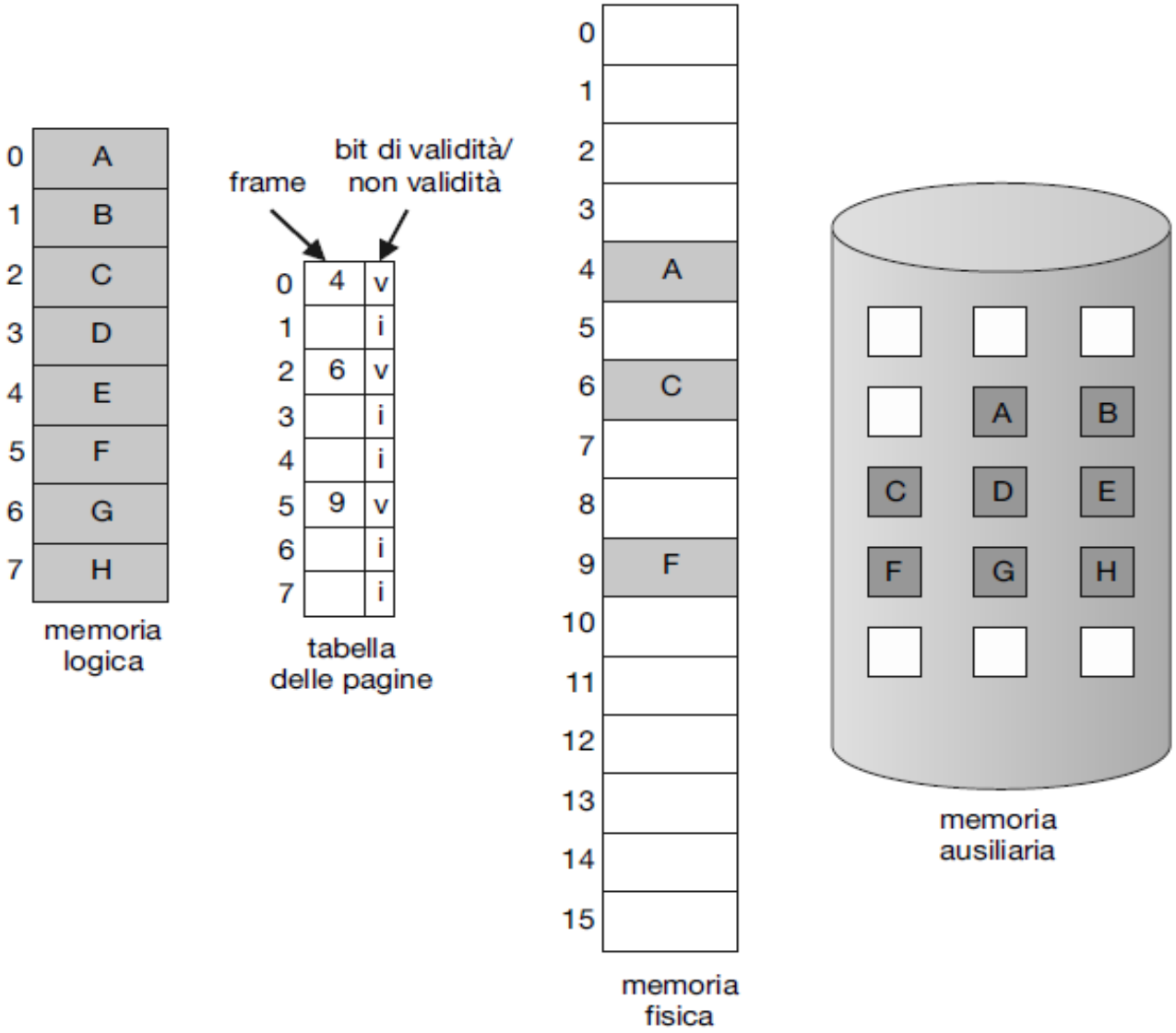
- R e W: diritti di accesso in lettura e scrittura
- M e U: bit di modifica e di uso (per gli algoritmi di sostituzione)
- P: bit di presenza
  - P = 1: pagina in memoria
  - P = 0: pagina non in memoria

page fault

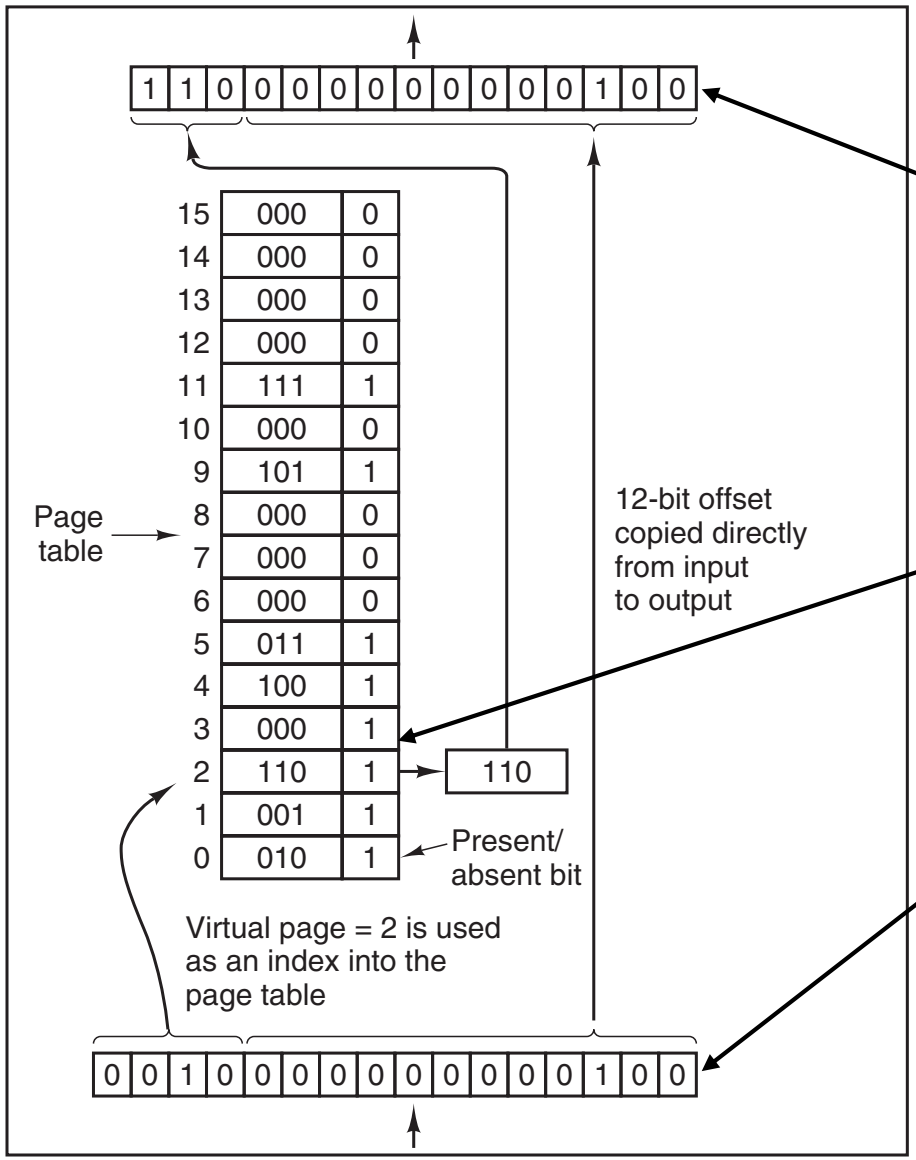
- Il descrittore mantiene solo informazioni utili all'HW per la traduzione degli indirizzi virtuali in fisici: quindi, l'indirizzo della swap area in cui è ospitata la pagina quando questa non è in memoria (cioè P=0), non viene inserito nel descrittore
- Tali informazioni, necessarie al SO per gestire i page fault, sono memorizzate in alcune tabelle interne del SO

Tabella delle pagine in cui alcune pagine non sono in memoria principale

bit di presenza
=
bit di validità



Traduzione di un indirizzo

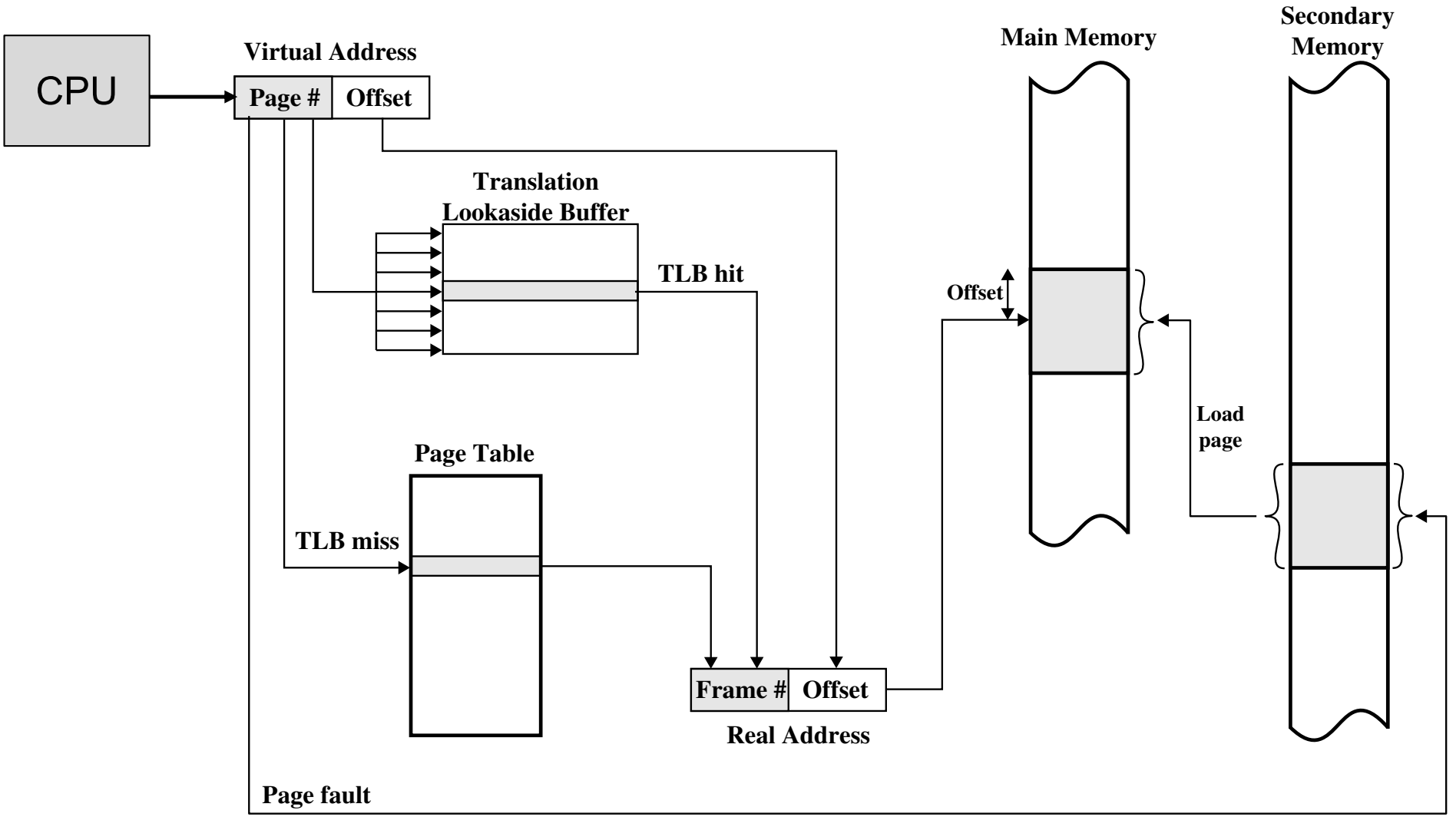


Indirizzo fisico a 15 bit (3+12), 8 frame di 4KB

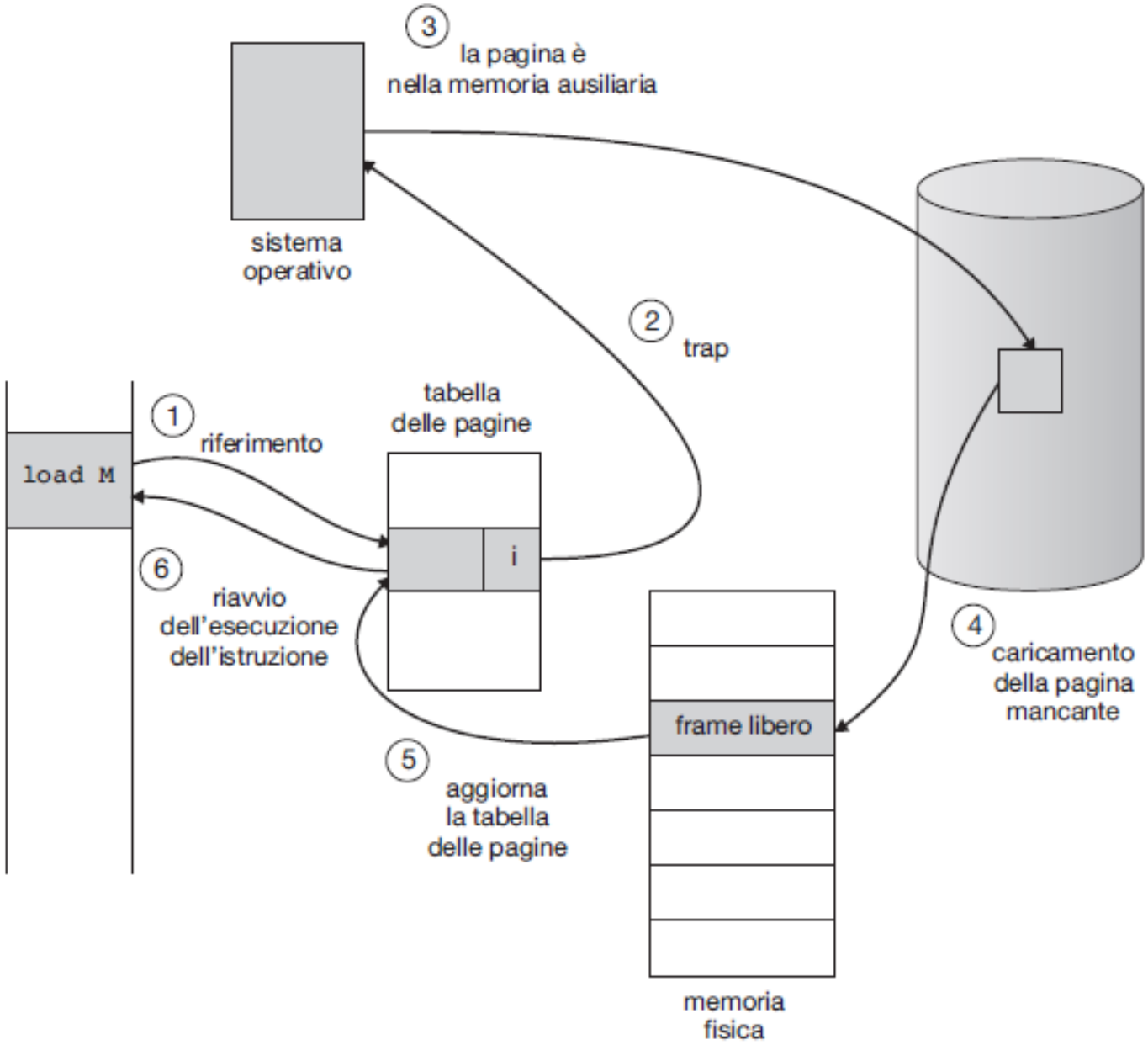
Bit di presenza della pagina in memoria

Indirizzo virtuale a 16 bit (4+12), 16 pagine virtuali di 4KB

Schema di traduzione degli indirizzi con TLB e page fault



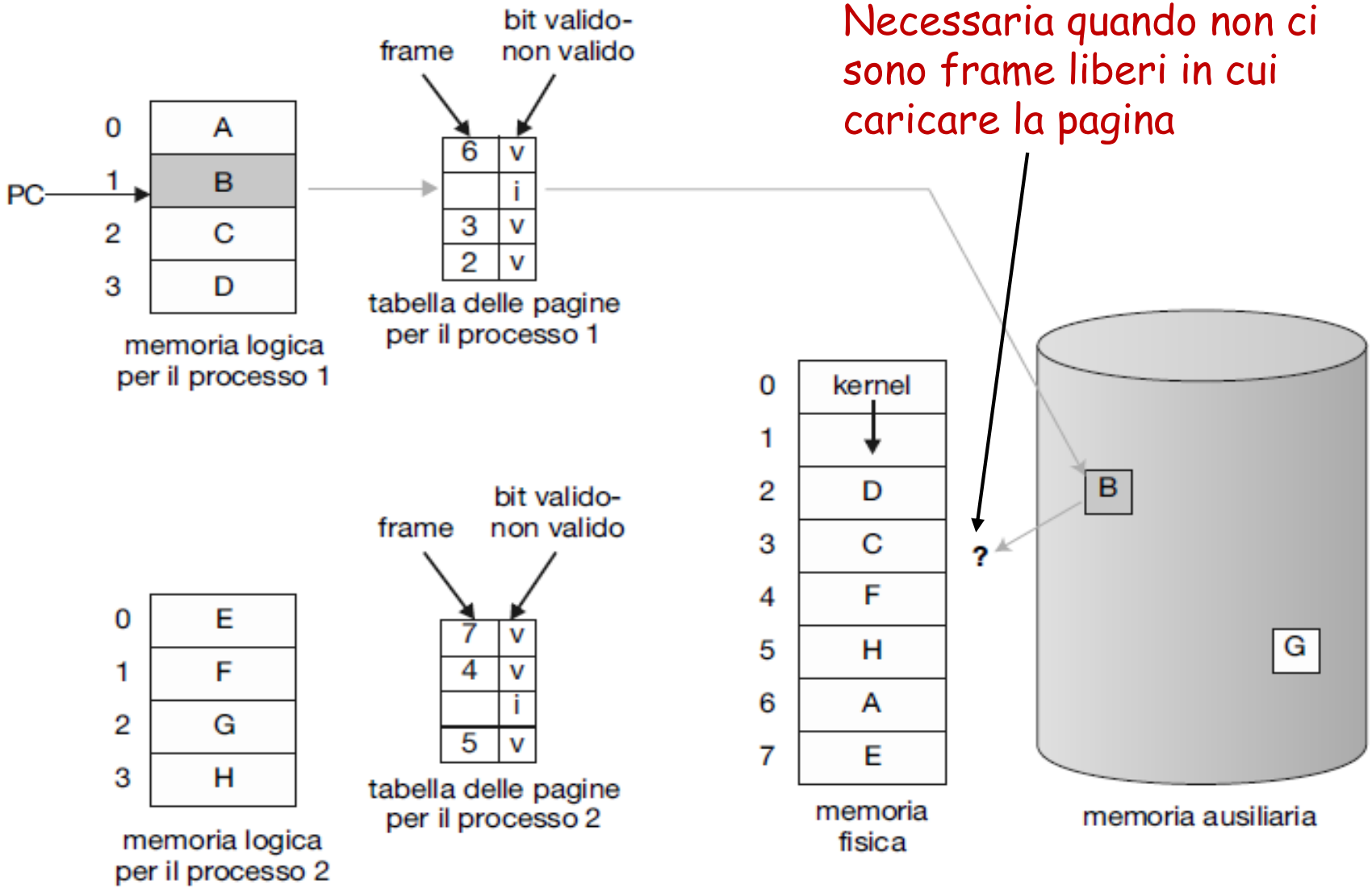
Traduzione di un indirizzo con page fault



Procedura di gestione di un page fault

1. Il SO controlla in una sua tabella interna relativa al processo (memorizzata insieme al suo PCB) per stabilire se il riferimento è un accesso alla memoria virtuale valido
2. Se il riferimento non è valido, fa terminare il processo; altrimenti, determina la locazione su disco della pagina desiderata
3. Individua un frame libero, usando la lista dei frame liberi
4. Carica la pagina desiderata nel frame libero
5. Aggiorna la tabella interna e la tabella delle pagine del processo
6. Riporta il processo che ha generato il page fault nella coda dei pronti (riprenderà l'esecuzione dall'istruzione che ha causato il page fault)

Sostituzione di pagine



Procedura di gestione di un page fault con invocazione di un algoritmo di sostituzione

1. Il SO controlla in una sua tabella interna relativa al processo (memorizzata insieme al suo PCB) per stabilire se il riferimento è un accesso alla memoria virtuale valido o no
2. Se il riferimento non è valido, fa terminare il processo; altrimenti, determina la locazione su disco della pagina desiderata
3. Individua un frame libero, usando la lista dei frame liberi
   - *se c'è, lo usa e aggiorna la lista*
   - *altrimenti, invoca un algoritmo di sostituzione per selezionare un pagina 'vittima' e scriverla su disco, e aggiorna le tabelle di conseguenza*
4. Carica la pagina desiderata nel frame libero
5. Aggiorna la tabella interna e la tabella delle pagine del processo
6. Riporta il processo che ha generato il page fault nella coda dei pronti (riprenderà dall'istruzione che ha causato il page fault)

Considerazioni sulla sostituzione di pagine

- Più semplice che nel caso della segmentazione perché tutte le pagine e i frame hanno le stesse dimensioni
- La sostituzione non ha bisogno di trasferire su disco la pagina vittima se questa non è mai stata modificata dopo il suo caricamento (M = 0)
- Certe pagine fisiche non possono essere selezionate
  - es. pagine che ospitano un buffer su cui i dispositivi di I/O possono leggere/scrivere tramite un canale DMA

  Nella tabella dei frame ogni elemento ha un bit, detto bit di lock, che se posto ad 1 impedisce lo scaricamento del frame
- La scelta della pagina da rimpiazzare è un fattore critico e va fatta in modo da limitare il più possibile i page fault
  - La generazione dei page fault e la necessità di gestirli diminuisce l'efficienza della gestione della memoria
  - *Thrashing*: *stato in cui l'attività della CPU è principalmente dedicata a trasferire pagine avanti e indietro dalla swap-area e alla gestione di page-fault*

Algoritmi necessari

Per realizzare la paginazione a domanda è necessario utilizzare

- algoritmi di sostituzione delle pagine
- algoritmi di allocazione dei frame

Posizionamento sul disco delle pagine scaricate

- Il metodo più semplice è di avere una partizione dedicata per lo swapping, come fa UNIX (area di swap), con un file system particolare
- Due alternative per lo spazio su disco allocato per ospitare le pagine di un processo
  - Spazio contiguo assegnato staticamente (per la gestione, basta un solo indirizzo del disco nel PCB; ma che succede se il processo 'cresce' ed ha bisogno di più spazio?)
  - Spazio assegnato dinamicamente al verificarsi dello swap out di ciascuna pagina (per ogni processo, bisogna mantenere una tabella di indirizzi di pagine su disco non necessariamente contigue)
- Non sempre è possibile avere una partizione di swap fissa: si possono allora usare uno o più file speciali allocati nel file system normale, come fa Windows

Tempo effettivo di accesso in memoria con page fault

- Sia p, con $0 \leq p \leq 1$, la probabilità che la pagina manchi
  - 1-p è la probabilità che la pagina sia presente
  - se p = 0 non ci sono pagine mancanti
  - se p = 1 ogni accesso genera un'eccezione pagina mancante
- Tempo effettivo di accesso in memoria con page fault ($EMAT_{pf}$)

  $EMAT_{pf}$ = (1-p) x tempo di accesso in memoria +
  p x tempo di gestione del page fault
- Gestione del page fault =

  servizio del segnale di eccezione +
  [scaricamento di una pagina +]
  caricamento della pagina richiesta +
  riavvio del processo

Tempo effettivo di accesso in memoria con page fault: osservazioni

- Il tempo di accesso alla memoria in genere è compreso tra 10 e 200 nanosecondi
- La gestione dell'interruzione pagina mancante richiede parecchie (centinaia) istruzioni, ciascuna delle quali impiega da 1 a 100 microsecondi
  - Se poi è necessario effettuare anche lo scaricamento di una pagina, il tempo di gestione dell'interruzione pagina mancante aumenta
- Il tempo medio di gestione del page fault può essere stimato intorno agli 8 millisecondi, che è il tempo necessario per il caricamento della pagina (supponendo che la coda del disco sia vuota)

Calcolo del Tempo effettivo di accesso in memoria con page fault

- Tempo di accesso alla memoria = 200 nanosecondi
- Tempo di gestione dell'interruzione pagina mancante = 8ms

$$EMAT_{pf} = (1 - p) \times 200 + p \times 8.000.000$$
$$= 200 + 7.999.800 \times p \quad \text{(nanosecondi)}$$

- $EMAT_{pf}$ è direttamente proporzionale a p (probabilità di pagina mancante) ed il costo di gestione dell'interruzione è preponderante rispetto a quello di accesso alla memoria
- Infatti, se un accesso ogni 1000 genera un'eccezione pagina mancante si ha

$$EMAT_{pf} = 200 + 7.999.800 \times 0.001 \sim 8.200 \text{ nanosecondi}$$

- Impiegando la paginazione su richiesta, il calcolatore è rallentato di un fattore pari a 40!

Calcolo del Tempo effettivo di accesso in memoria con page fault

- Tempo di accesso alla memoria = 200 nanosecondi
- Tempo di gestione dell'interruzione pagina mancante = 8ms

$$EMAT_{pf} = (1 - p) \times 200 + p \times 8.000.000$$
$$= 200 + 7.999.800 \times p \quad \text{(nanosecondi)}$$

- Quale dovrebbe essere la probabilità che la pagina manchi se siamo disposti a tollerare un rallentamento del tempo di accesso alla memoria di al più il 10%?

Bisogna che il seguente vincolo sia soddisfatto:

$$200 + 7.999.800 \times p < 220 \; (= 200 + 0.1 \times 200)$$
$$7.999.800 \times p < 20$$
$$p < 1/400.000 = 0{,}0000025$$

cioè possiamo permettere al più una pagina mancante ogni 400.000 accessi alla memoria!

Gestione software del TLB

- Abbiamo finora assunto che ogni elaboratore con memoria virtuale segmentata/paginata disponga di un TLB
  - La gestione del TLB ed il trattamento degli errori del TLB sono interamente svolti dall'HW della MMU
  - I trap al SO avvengono solo quando un segmento/pagina non è in memoria (cioè si verifica un segment/page-fault)
- Tuttavia, molte macchine RISC moderne, tra cui SPARC e MIPS, non hanno un TLB HW
  - Quasi tutta la gestione delle pagine è effettuata dal SO che simula via SW il funzionamento del TLB
  - Naturalmente il tutto deve avvenire con una manciata di istruzioni, poiché i TLB miss sono molto più frequenti dei page fault
  - Se il TLB è ragionevolmente grande (almeno 64 elementi) la sua gestione SW si rivela accettabilmente efficiente

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

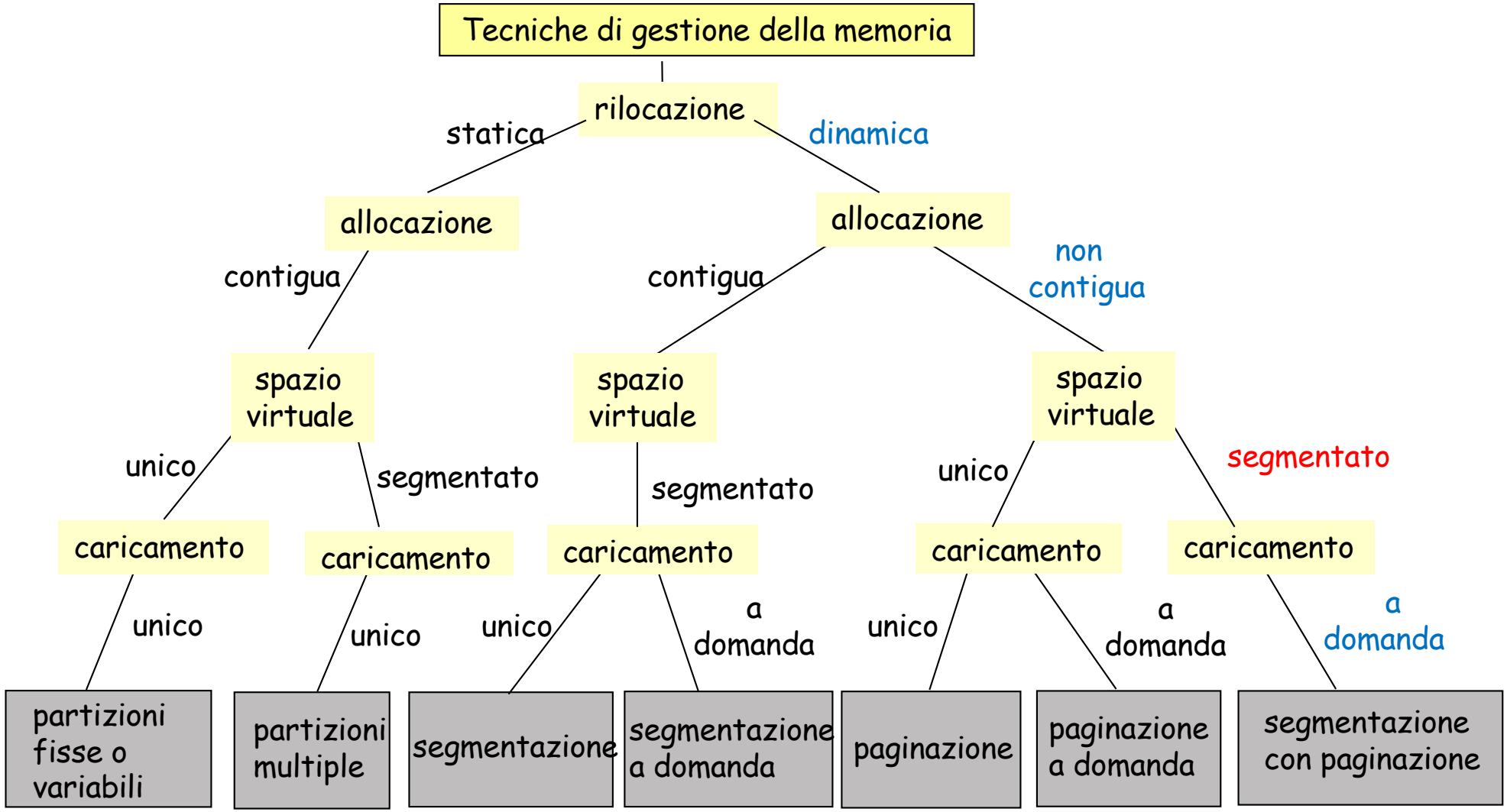
Segmentazione con paginazione

Strutturazione dello spazio virtuale a segmenti ed allocazione della memoria ad ogni segmento mediante paginazione

- Se i segmenti sono grandi, potrebbe essere impossibile o non conveniente, mantenerli in memoria principale per intero
- I segmenti sono allora suddivisi in pagine dimodoché devono essere mantenute in memoria solo quelle pagine che sono necessarie per l'esecuzione

rilocazione degli indirizzi	allocazione della memoria	spazio virtuale	caricamento
DINAMICA	NON CONTIGUA	SEGMENTATO	A DOMANDA

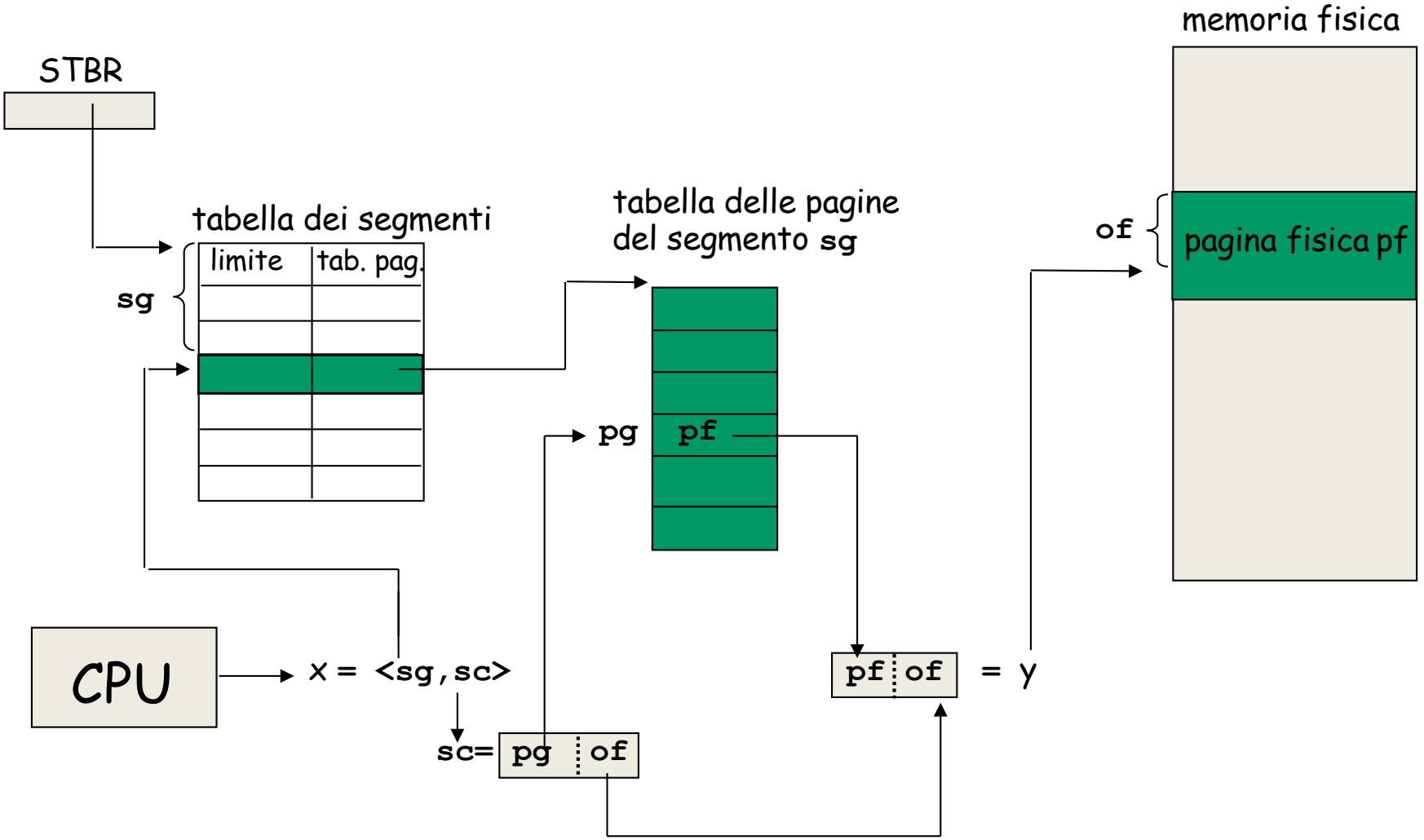
Segmentazione con paginazione



Segmentazione con paginazione

- Usato per primo da MULTICS, che ha introdotto vari concetti che sono alla base dei SO moderni
- Preserva i vantaggi di concepire lo spazio virtuale organizzato in unità logiche (segmenti, come il programma, per modularità, protezione e condivisione) e di allocare la memoria fisica in maniera non contigua (pagine di uguali dimensioni, riduce la frammentazione)
- Gli elementi della tabella dei segmenti non contengono l'indirizzo di base di un segmento ma l'indirizzo base della tabella delle pagine per quel segmento
- L'indirizzo virtuale è della forma <*sg*,*sc*>, dove lo *scostamento sc* a sua volta è strutturato come <*pg*,*of*>

Schema di traduzione degli indirizzi



Considerazioni sulla traduzione degli indirizzi

- Lo schema di traduzione degli indirizzi appena visto è semplificato: mancano i controlli sugli indici sg e pg
- La traduzione degli indirizzi può generare vari tipi di interruzione:
  - segment fault: se il bit di presenza P nel descrittore del segmento è a 0 (cioè, la tabella delle pagine del segmento non è in memoria)
  - page fault: se il bit di presenza P nel descrittore della pagina è a 0 (cioè, la pagina riferita non è in memoria)
- Uno schema simile a quello visto è supportato dai processori Intel
  - La paginazione del singolo segmento è opzionale (dipende da un bit del registro di stato della CPU)

Gestione della memoria principale e virtuale

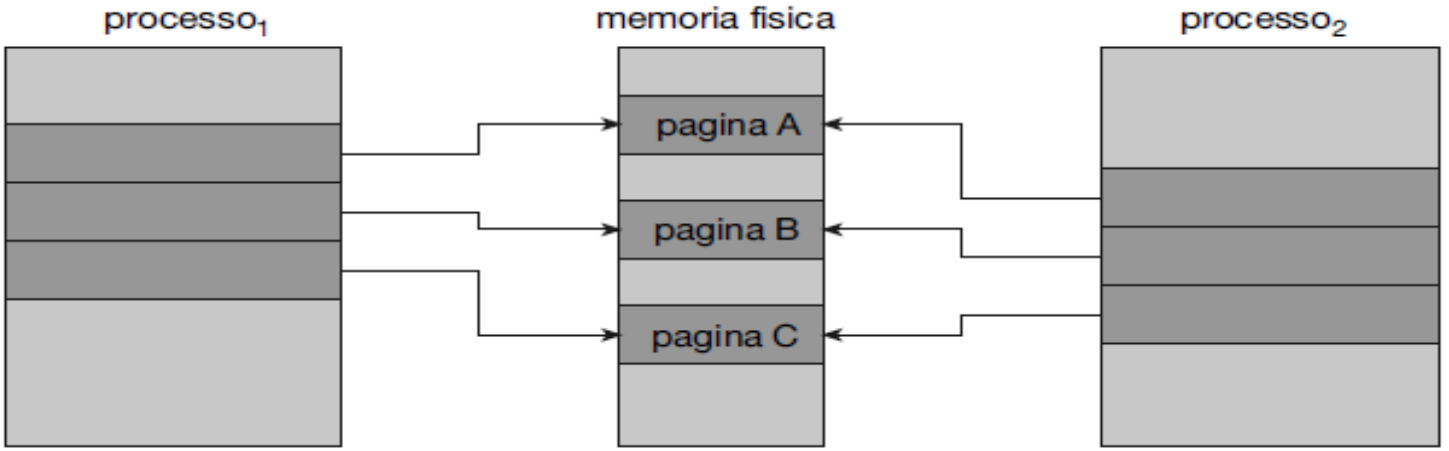
- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Copiatura su scrittura

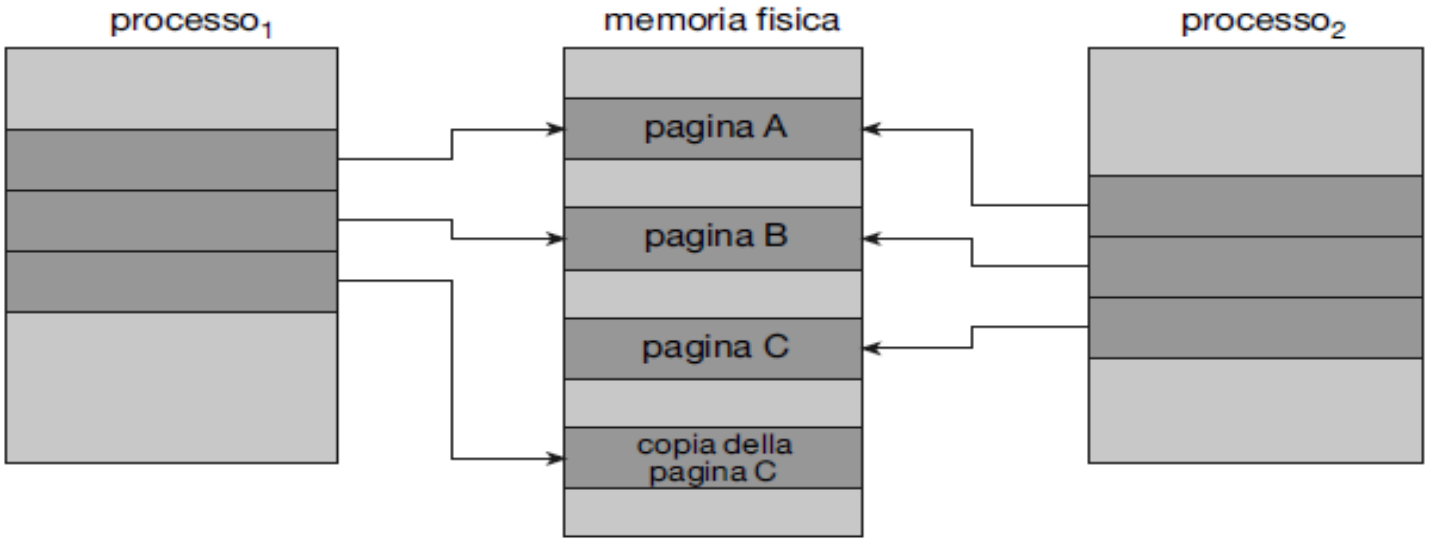
- Nella paginazione a domanda, un processo può essere schedulato per l'uso della CPU anche se ancora nessuna delle sue pagine è stata caricata in memoria principale
  - Il suo spazio virtuale risiede completamente nella swap area su disco e le sue pagine virtuali vengono caricate via via che vengono riferite tramite la gestione dell'interruzione page fault generata dal meccanismo HW di traduzione degli indirizzi
- La generazione di processi tramite la system call `fork()` può migliorare le prestazioni usando la condivisione della memoria per evitare la generazione dei page fault iniziali e sfruttando una tecnica nota come copiatura su scrittura che si fonda su
  - condivisione iniziale delle pagine tra processi genitori e figli
  - successiva creazione di una copia di una pagina nel momento in cui uno dei processi scrive nella pagina condivisa
- Così facendo, si copiano soltanto le pagine modificate da uno dei due processi, le altre restano condivise

Copiatura su scrittura

Prima della modifica della pagina C da parte del processo$_1$



Dopo della modifica della pagina C da parte del processo$_1$



Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Algoritmi di sostituzione delle pagine

- In caso di page fault e memoria principale non disponibile, scelgono una pagina logica *vittima*, cioè da rimpiazzare con la pagina logica il cui riferimento ha generato il page fault
  - Algoritmi analoghi si possono usare nel caso di segment fault e sostituzione di segmenti (più complessa per via del fatto che i segmenti hanno dimensioni differenti)
- Vari algoritmi:
  - *Ottimo*
  - *FIFO*
  - *LRU*
  - ...

  hanno un impatto differente sul numero di page fault risultanti
- Ciascun *page fault* comporta un considerevole ritardo nei tempi d'esecuzione di un processo, perciò le prestazioni globali di un SO dipendono in modo cruciale dalla qualità del suo algoritmo di sostituzione

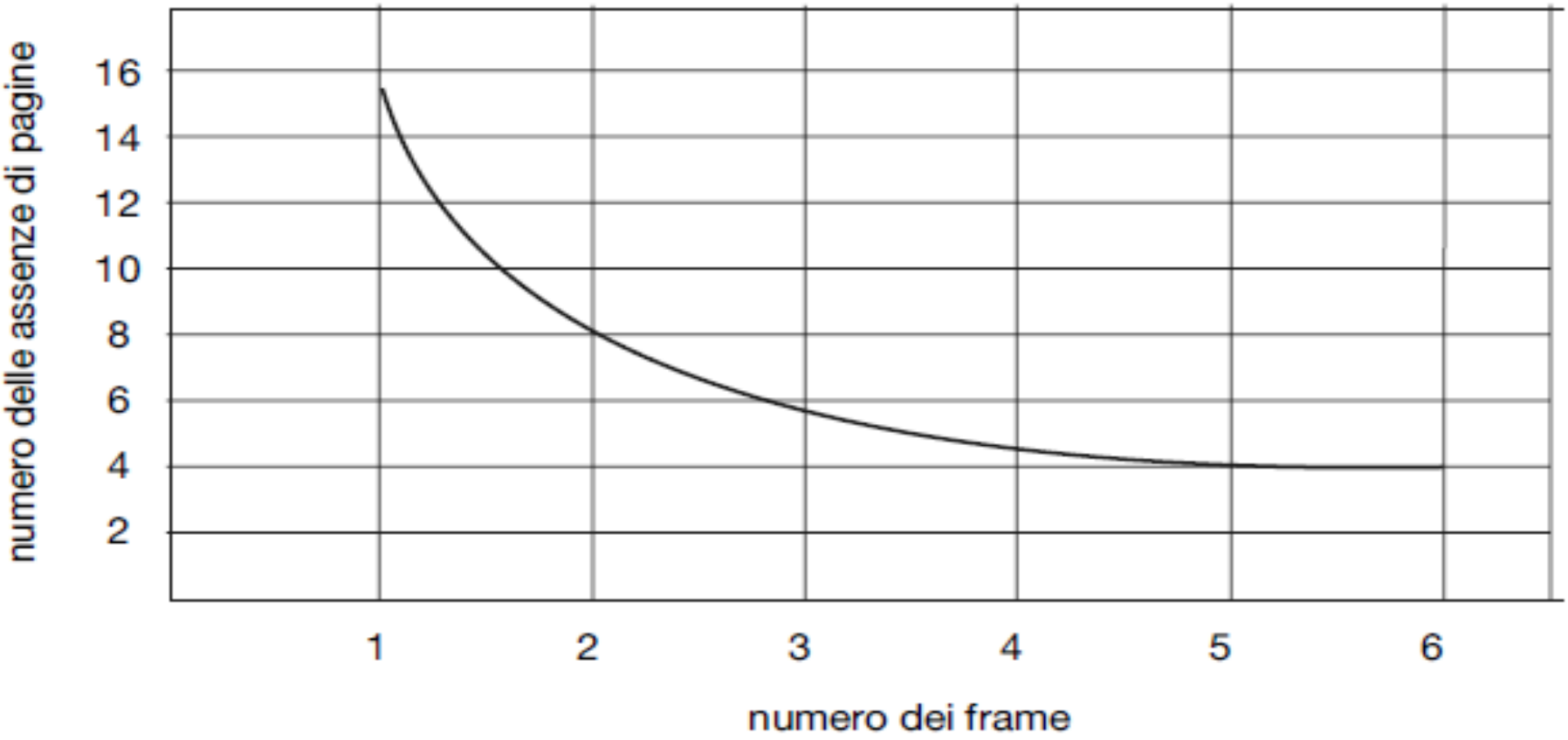
Algoritmi di sostituzione delle pagine

- Pertanto una metrica per valutare la qualità di un algoritmo di sostituzione è il numero di page fault risultanti dalla sua applicazione
  - Si fissa una quantità di frame a disposizione
  - Si fissa una determinata sequenza di accessi alle pagine logiche
  - Si simula il comportamento dell'algoritmo di sostituzione delle pagine e si conta il numero di page fault risultanti
  - Tanto minore è il numero di page fault, tanto migliore è l'algoritmo
- Valuteremo i vari algoritmi effettuandone l'esecuzione sulla seguente successione dei riferimenti alla memoria

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

  e calcolando il numero di page fault risultanti nell'ipotesi che la memoria fisica metta a disposizione 3 frame

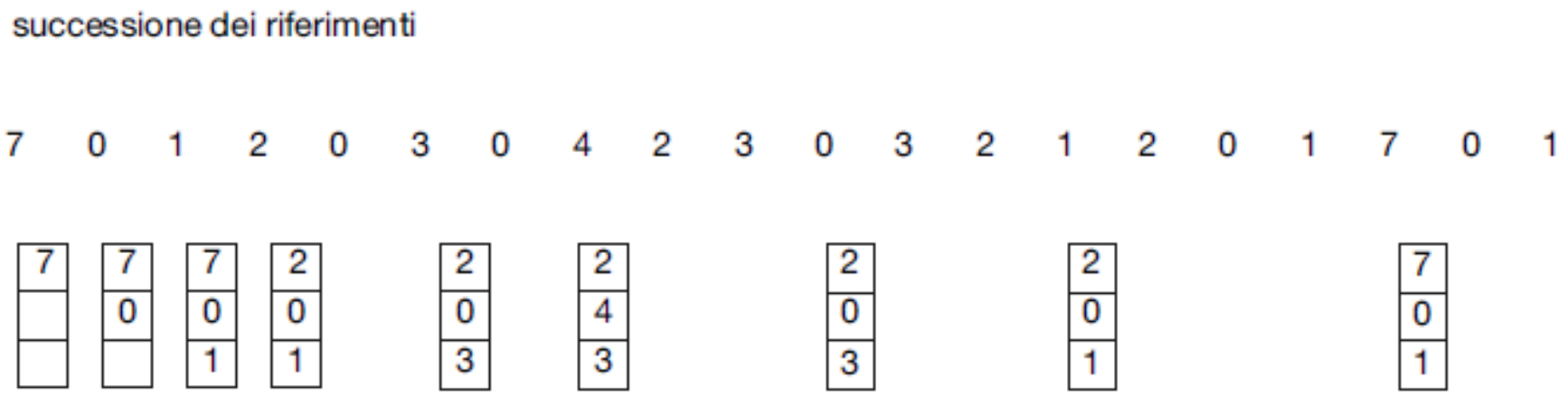
Numero atteso delle mancanze di pagina in funzione del numero di frame



Aumentando il numero dei frame, ci si attende che il numero di page fault diminuisca fino ad un livello minimo

Algoritmo ottimo

Sceglie come pagina da rimpiazzare una di quelle che sicuramente non sarà più riferita in futuro o, quantomeno, quella che sarà riferita più tardi nel tempo (cioè che non si userà per il periodo di tempo più lungo)



9 assenze di pagina

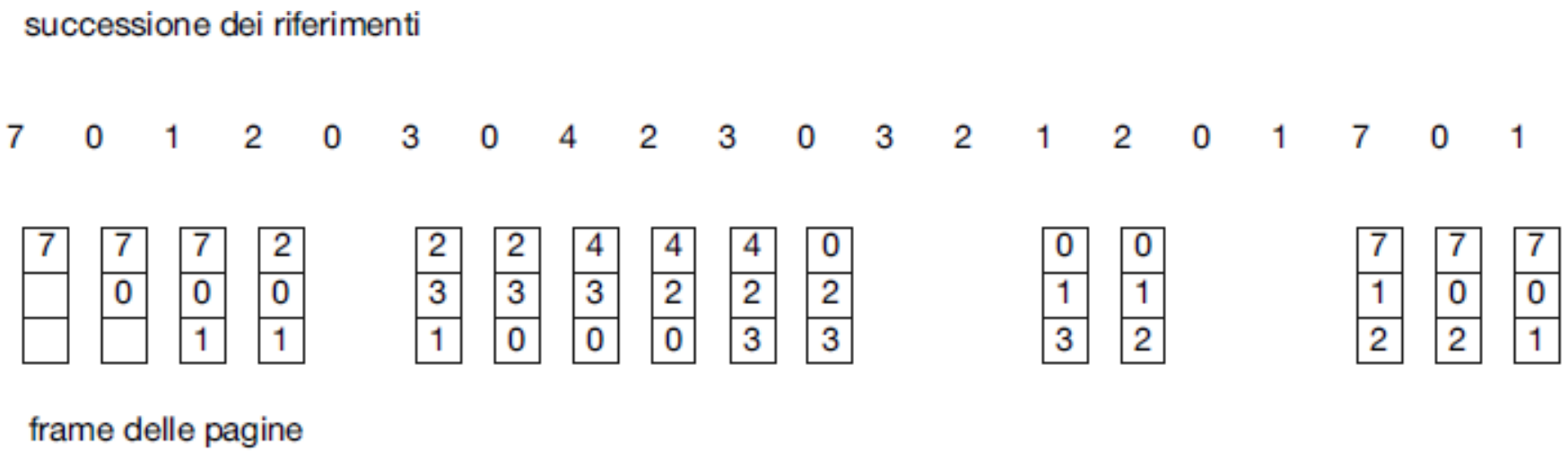
Algoritmo ottimo: considerazioni

Sceglie come pagina da rimpiazzare una di quelle che sicuramente non sarà più riferita in futuro o, quantomeno, quella che sarà riferita più tardi nel tempo (cioè che non si userà per il periodo di tempo più lungo)

- È un algoritmo *ideale*, non realizzabile perché non si può prevedere il futuro
- È comunque utile come punto di riferimento per misurare la qualità degli altri algoritmi
- Nella pratica, si usano informazioni relative ad accessi effettuati nell'immediato passato

Algoritmo FIFO (first in first out)

Associa a ogni pagina l'istante di tempo in cui la pagina è stata caricata in memoria e sceglie come vittima la pagina caricata per prima in ordine di tempo



15 assenze di pagina

FIFO: considerazioni

Associa a ogni pagina l'istante di tempo in cui la pagina è stata caricata in memoria e sceglie come vittima la pagina caricata per prima in ordine di tempo

- Semplice realizzazione: gli elementi della tabella dei frame sono concatenati in modo da comporre una coda FIFO; si scarica di volta in volta l'elemento in testa
  - Non c'è bisogno di registrare l'istante di tempo in cui le pagine sono state caricate in memoria
- Inconveniente: non sempre la pagina caricata da più tempo è non più necessaria o è quella che sarà necessaria il più tardi possibile
  - Ciò porta all'anomalia di Belady

Più memoria: non sempre è meglio!

Anomalia di Belady: più frame ⇒ più mancanze di pagina!

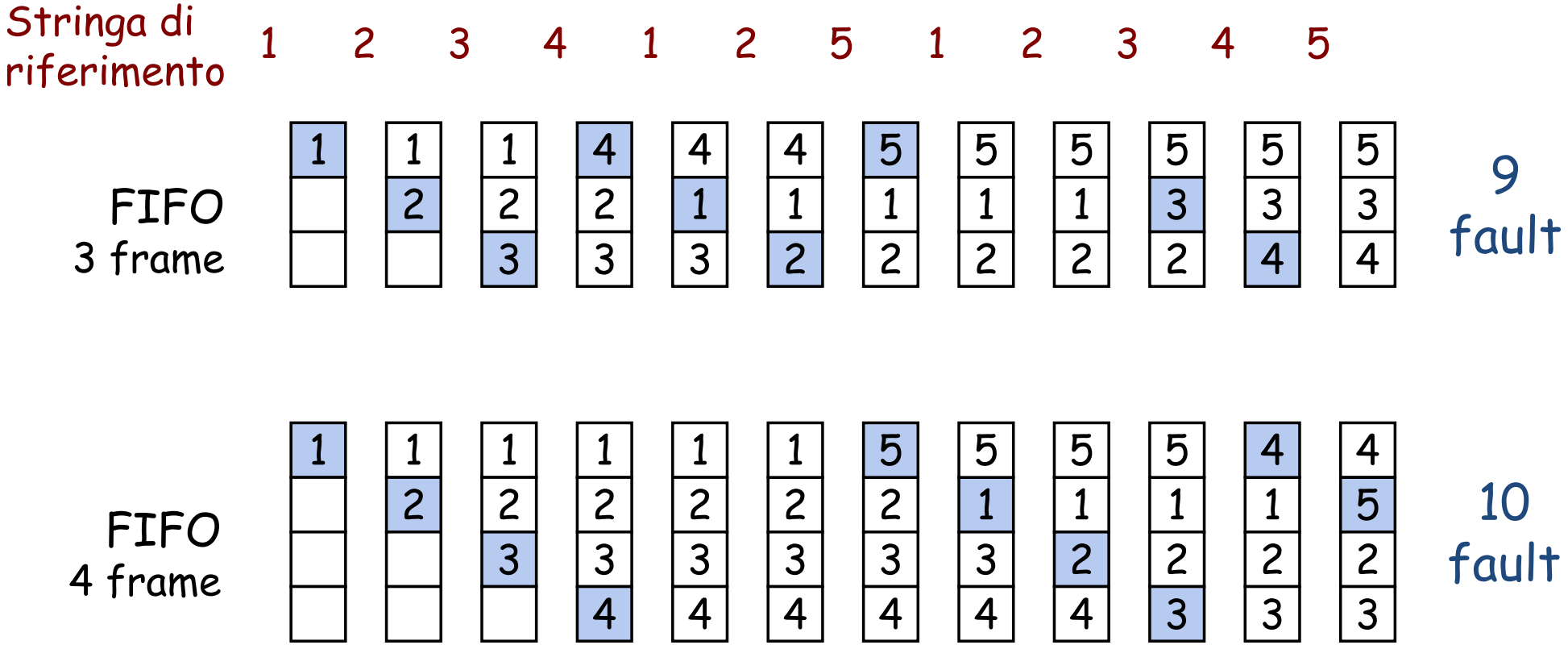
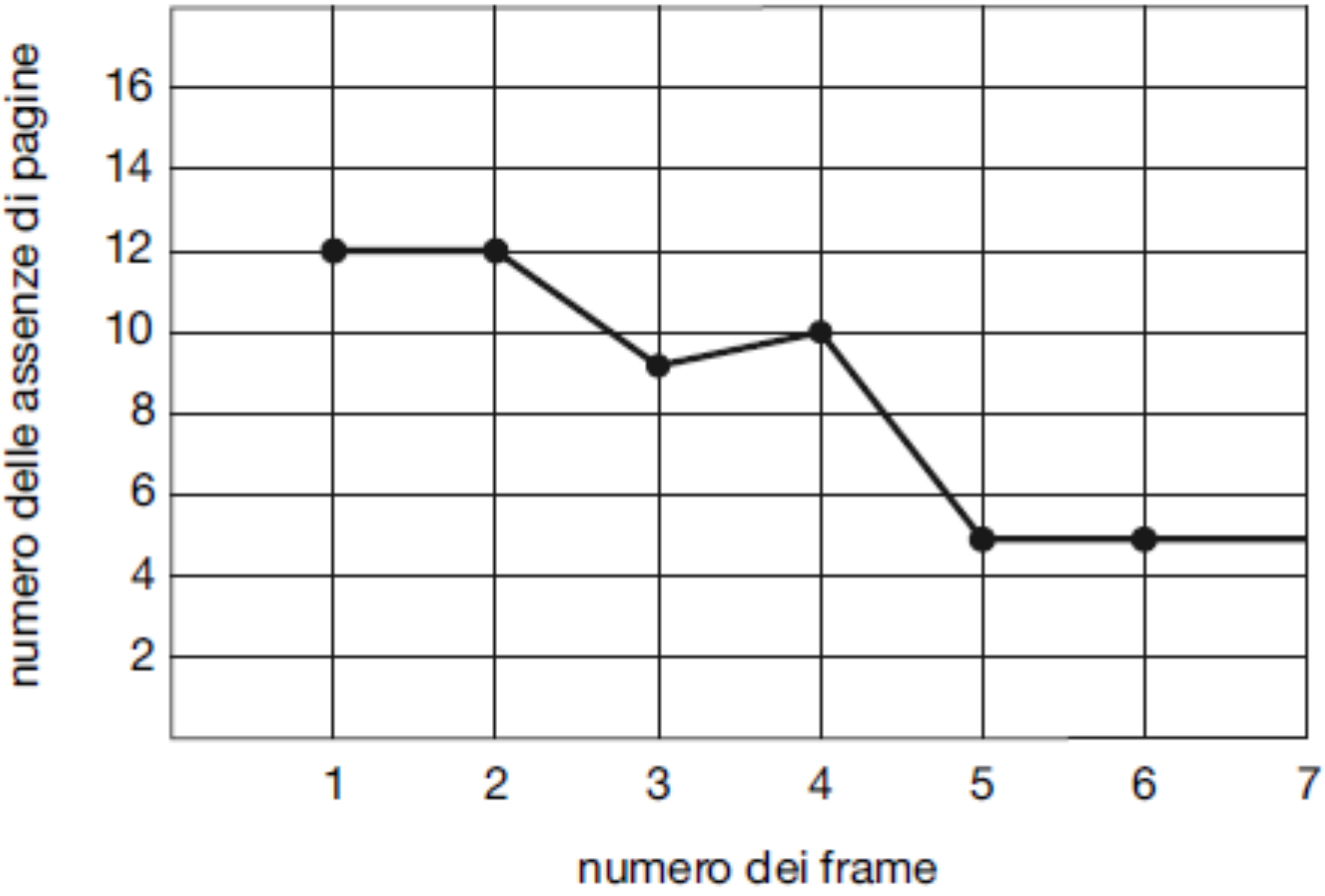


Grafico che illustra l'anomalia di Belady

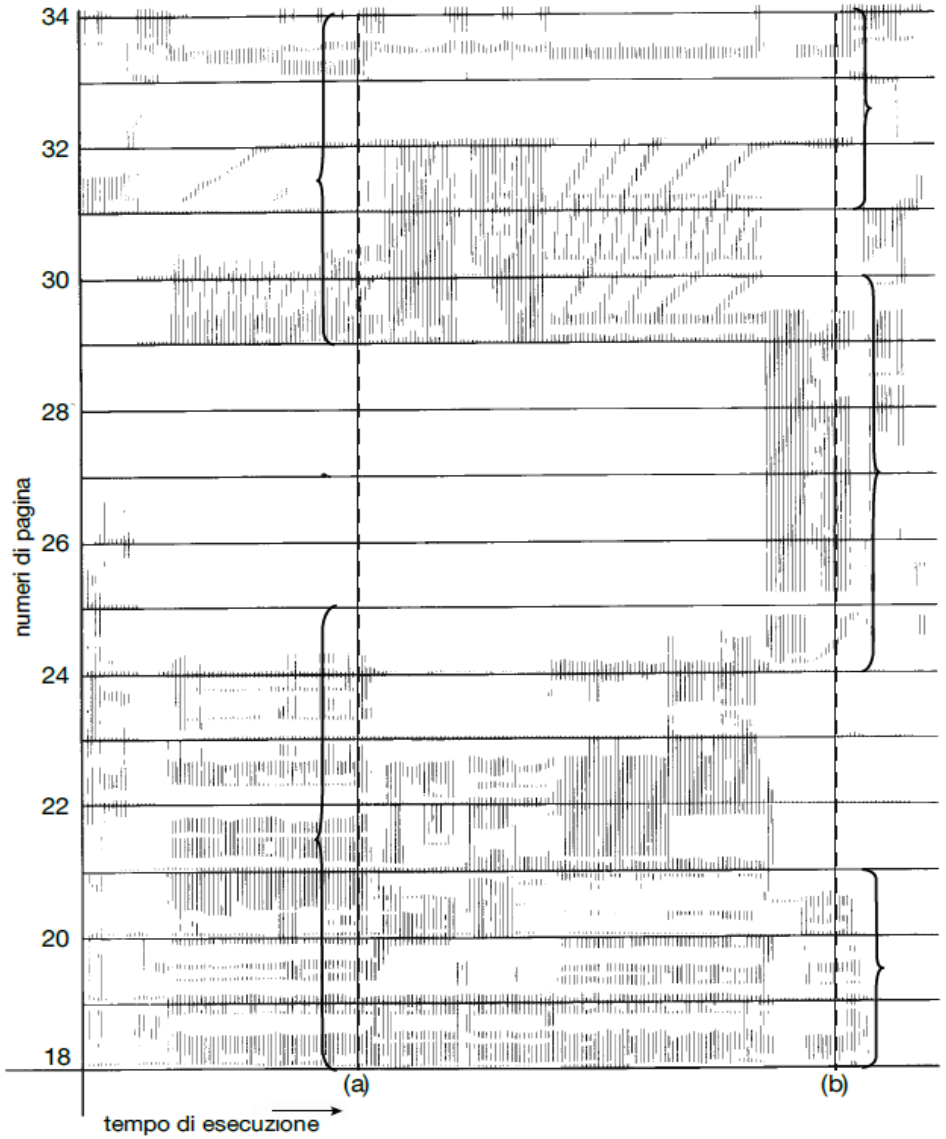


Con alcuni algoritmi di sostituzione delle pagine, quali FIFO, il tasso di page fault può *aumentare* con il numero dei frame assegnati ai processi

Località dei riferimenti alla memoria

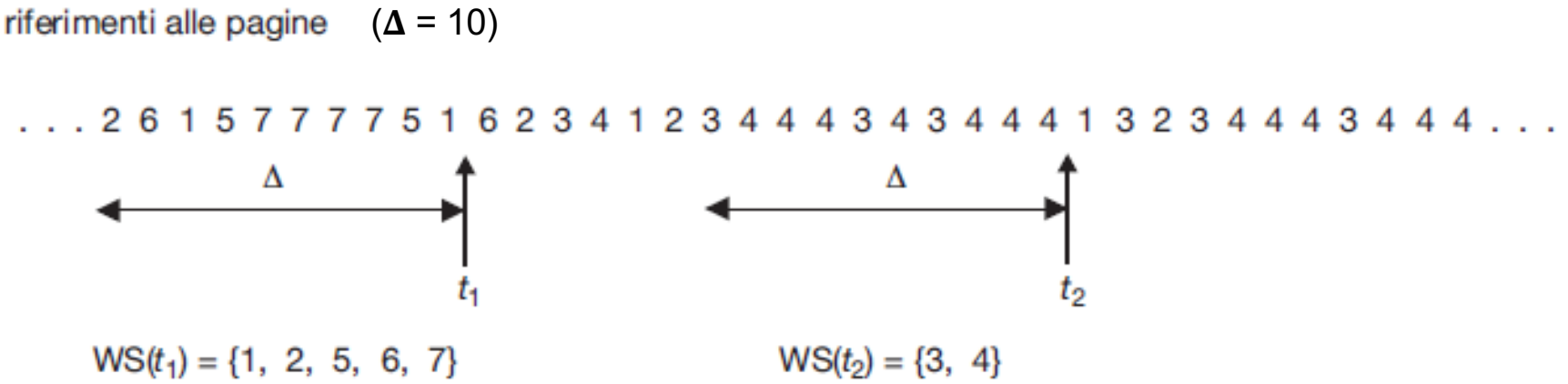
Si usano allora informazioni relative ad accessi nell'immediato passato

- La località di un processo ad un dato istante è l'insieme delle sue pagine virtuali che sta attivamente usando
- Durante l'esecuzione, un processo attraversa località diverse
- Le località sono determinate dalla struttura del programma e dalle sue strutture di dati
- Principio di località spaziale: un processo durante la sua esecuzione si sposta di località in località in maniera graduale
- Ad esempio, un cambio di località tipicamente si verifica quando è chiamata una funzione



Working set

- Idealmente, ad un dato istante, ad ogni processo bisognerebbe assegnare i frame sufficienti ad ospitare tutte le pagine virtuali che fanno parte della sua località
- La località di un processo può essere approssimata tramite il concetto di working set (o insieme di lavoro): l'insieme delle sue pagine virtuali riferite nei più recenti **Δ** riferimenti (o nelle **Δ** più recenti unità di tempo)
  - Se una pagina è in uso attivo si trova nel working set
  - Se non è più usata esce dal working set **Δ** riferimenti dopo il suo ultimo riferimento

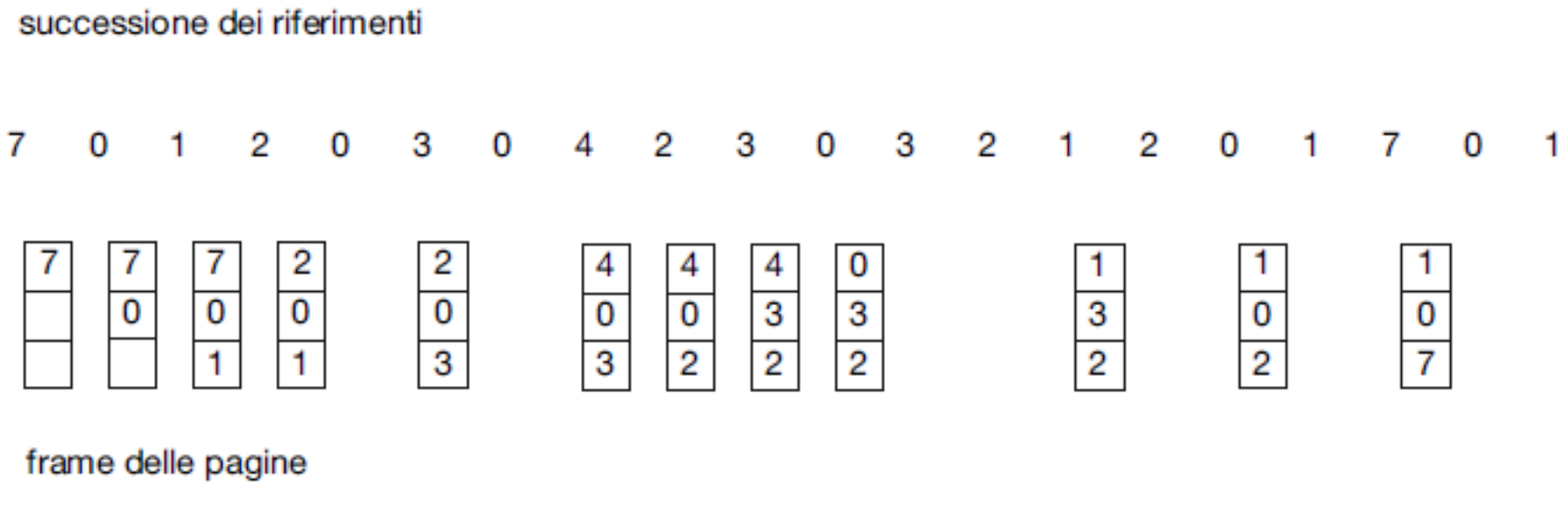


Località & Working set

- Principio di località temporale: la probabilità di accedere ad una pagina utilizzata di recente è maggiore della probabilità di accedere ad una pagina utilizzata nel lontano passato
- La precisione con cui è calcolato il WS dipende da **Δ**
  - Se Δ è troppo piccolo, non include l'intera località
  - Se Δ è troppo grande, può ricomprendere più località
  - Al limite, se Δ è infinito, il working set coincide con l'insieme di tutte le pagine riferite dal processo durante la sua esecuzione
- Per velocizzare la traduzione degli indirizzi, i descrittori delle pagine del working set del processo in esecuzione dovrebbero essere nel TLB

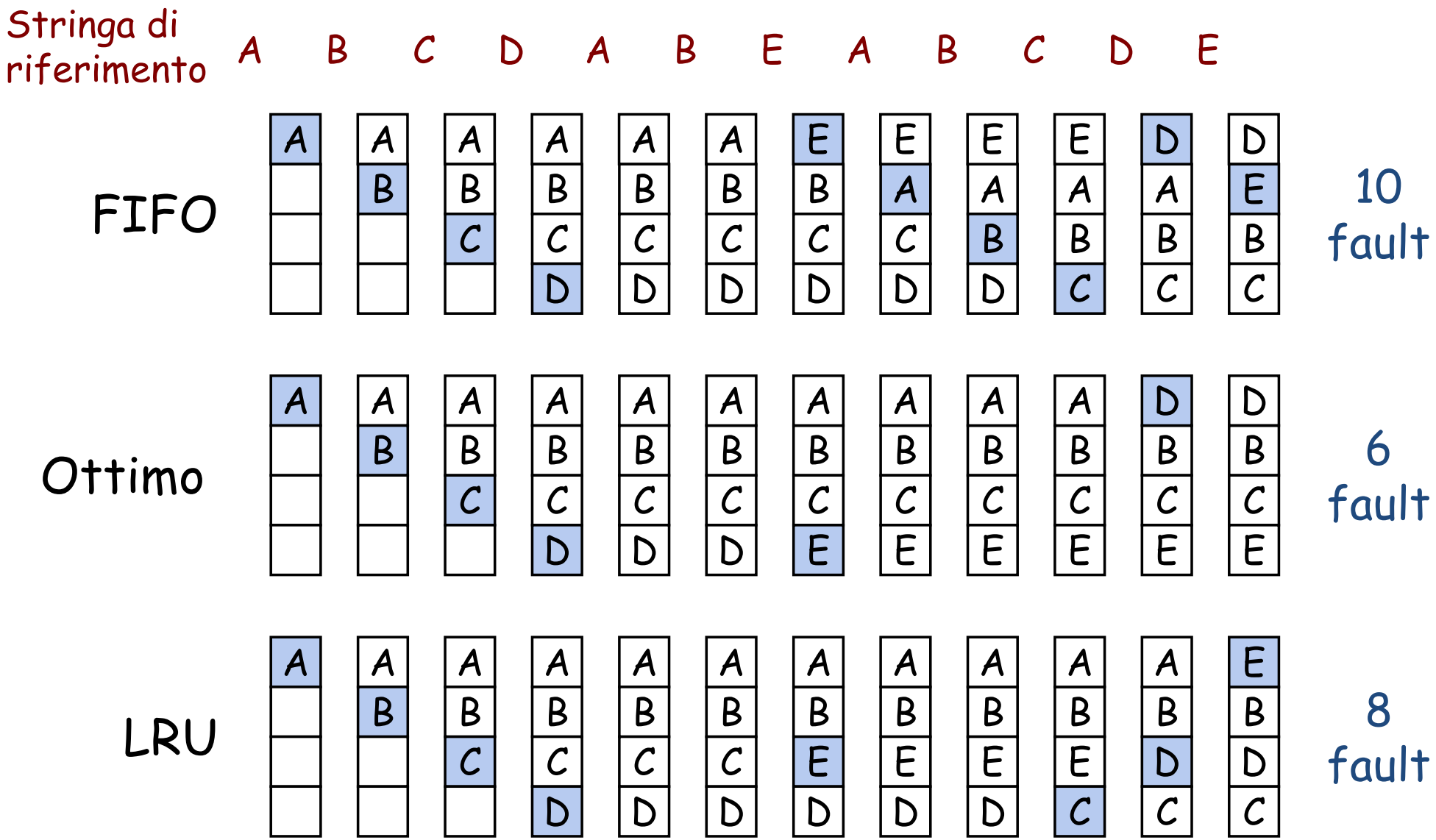
Algoritmo LRU (least recently used)

Associa a ogni pagina l'istante di tempo in cui la pagina è stata acceduta per l'ultima volta e sceglie come vittima la pagina che non è stata acceduta da più tempo



12 assenze di pagina

Esempio di applicazione degli algoritmi di sostituzione



LRU: implementazione

Associa a ogni pagina l'istante di tempo in cui la pagina è stata acceduta per l'ultima volta e sceglie come vittima la pagina che non è stata acceduta da più tempo

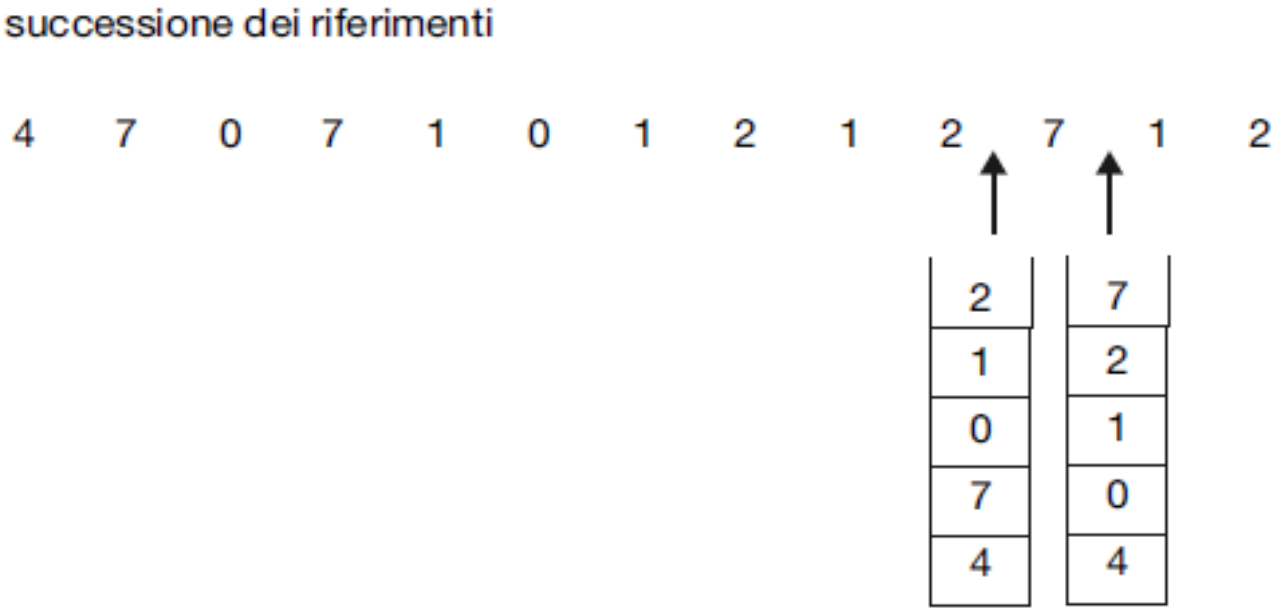
- La difficoltà più evidente nell'implementare l'algoritmo LRU è gestire la memorizzazione dell'istante dell'ultimo accesso a ciascuna pagina
- Due possibili implementazioni:
  - Contatore incrementato ad ogni accesso in memoria e salvato nella voce della tabella delle pagine relativa alla pagina acceduta
  - Stack di numeri di pagina ordinato in base all'ultimo accesso

LRU con contatori

- La CPU possiede un registro contatore che incrementa ad ogni accesso
  - Di solito si usa il clock di sistema come contatore
- Ogni elemento della tabella delle pagine ha un campo "momento di utilizzo"
- *Quando si riferisce una pagina*, si aggiorna il campo momento di utilizzo della pagina riferita copiando al suo interno il valore del registro contatore
- *Quando si esegue l'algoritmo*, si cerca nella tabella della pagine la pagina LRU che è quella il cui campo "momento di utilizzo" ha valore più piccolo

LRU con stack

- Si mantiene uno stack di numeri di pagina tramite una lista a doppio collegamento
- *Quando si riferisce una pagina*, si mette il numero di pagina in cima allo stack (eventualmente togliendolo dalla posizione che già occupava nello stack)
- *Quando si esegue l'algoritmo*, la pagina LRU è quella il cui numero è in fondo allo stack (accessibile tramite un apposito puntatore)



LRU: considerazioni

- Vantaggio principale: non soffre dell'anomalia di Belady, così come l'algoritmo ottimo
- Infatti LRU fa parte di una classe più ampia di algoritmi di sostituzione, detti stack algorithms, che godono della seguente proprietà:

  L'insieme delle pagine caricate in memoria avendo n frame disponibili è un sottinsieme dell'insieme delle pagine che sarebbero caricate in memoria avendo n+1 frame disponibili
- Gli stack algorithms non soffrono dell'anomalia di Belady

LRU: considerazioni

- Sperimentalmente è l'algoritmo le cui prestazioni si avvicinano di più a quelle dell'algoritmo ottimo
- Realizzazione costosa
  - Quando si rende necessario scaricare una pagina:
    - la realizzazione con contatori comporta ricerca e scrittura all'interno della tabella delle pagine
    - la realizzazione con stack comporta l'aggiornamento di un certo numero di puntatori (fino a 6)
  - Indispesabile un supporto HW per evitare che, ad ogni accesso alla memoria, si debba far ricorso al meccanismo delle interruzioni per chiedere al SO di modificare tali strutture dati
    - Rallenterebbe il sistema di un fattore 10!

Implementazione con bit di riferimento

- Implementare un meccanismo che memorizzi gli istanti di tempo oppure l'ordine degli ultimi accessi alle pagine di memoria è costoso anche a livello HW
- Tuttavia molte architetture mettono a disposizione un bit di riferimento (o di uso) per ogni pagina
  - È associato al corrispondente elemento della tabella delle pagine del processo
  - Inizialmente il SO lo imposta a 0
  - L'HW lo imposta a 1 quando la pagina viene riferita
  - Periodicamente il SO lo reimposta a 0
- Grazie ai bit di riferimento è possibile conoscere quali pagine di memoria sono state accedute in un certo intervallo di tempo (ma non l'ordine esatto)
  - Ad un dato istante, le pagine con bit di riferimento a 1 costituiscono una stima del working set

Algoritmi di sostituzione delle pagine che approssimano LRU

- Gli algoritmi di sostituzione delle pagine nei SO moderni utilizzano i bit di riferimento per approssimare l'algoritmo LRU
- Ne vedremo alcuni
  - Algoritmo con bit supplementari di riferimento (**aging**)
  - Algoritmo con seconda chance
  - Algoritmo con seconda chance migliorato
  - Algoritmi basati su conteggio
- Molti sistemi, oltre ad uno specifico algoritmo, adottano una memorizzazione transitoria (buffering) delle pagine per migliorare ulteriormente le prestazioni

Algoritmo con bit supplementari di riferimento

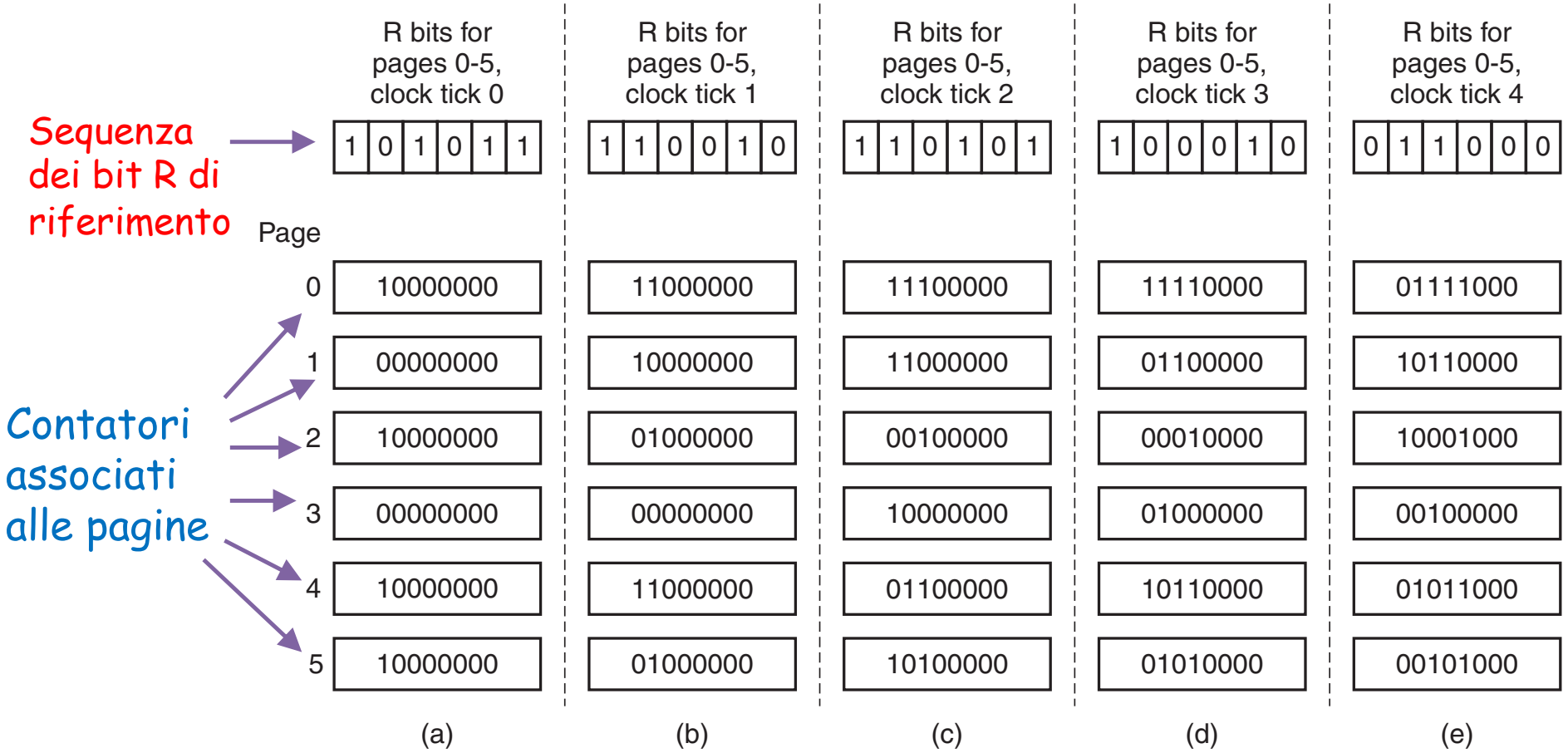
Tale algoritmo, che da alcuni autori è chiamato aging, è una buona approssimazione dell'algoritmo LRU e sfrutta i bit di riferimento per impostare i valori di alcuni contatori software

- Informazioni sull'ordine con cui sono accedute le pagine possono essere ottenute
  - campionando periodicamente i bit di riferimento delle pagine e
  - salvandone lo stato in una tabella in memoria formata da un contatore, ad esempio un byte, per ogni pagina
- Ad intervalli regolari, es. 100 ms, un interrupt del timer trasferisce il controllo al SO il quale per ciascuna pagina
  - legge il valore del bit di riferimento
  - lo salva come bit più significativo del contatore corrispondente alla pagina, dopo averne effettuato uno shift a destra del contenuto (scartando quindi il bit meno significativo)
  - azzera il bit di riferimento

Algoritmo con bit supplementari di riferimento

- Tali contatori contengono quindi la storia dell'utilizzo delle pagine negli ultimi intervalli
  - 8 intervalli, se sono formati da un byte
- Interpretando tali byte come interi senza segno, la pagina cui è associato l'intero minore è la pagina LRU e può essere sostituita
  - Se più pagine hanno lo stesso valore, si può usare una selezione FIFO o casuale o si possono restituire tutte
- Il numero di bit che formano il contatore di ogni pagina può variare in base all'architettura
  - È un compromesso tra durata del periodo di osservazione, quantità di memoria occupata dai contatori, e costo dell'operazione di aggiornamento
  - Prendendo 0 bit, quindi usando il solo bit di riferimento, si ottiene l'algoritmo con seconda chance

Algoritmo con bit supplementari di riferimento



- Per ciascuna pagina si usa un contatore con 8 bit, ciascuno memorizza il valore del bit di riferimento in uno degli ultimi 8 intervalli
- Al page fault, si sceglie la pagina con il valore del contatore più piccolo

Algoritmo con seconda chance

- Le pagine vengono mantenute in una lista FIFO in base a quando sono state caricate in memoria
  - In cima alla lista c'è la pagina presente in memoria da più tempo
- Quando viene invocato, l'algoritmo
  1. Considera la pagina in cima alla lista
  2. Controlla il bit di riferimento di tale pagina
     - Se R = 0: seleziona la pagina come 'vittima'
     - Se R = 1: pone R = 0, sposta la pagina in fondo alla lista (trattandola come se fosse stata appena caricata in memoria) e torna al punto 1
- La pagina selezionata viene sostituita da una nuova pagina che viene inserita in fondo alla lista (con R = 1)

Algoritmo con seconda chance: considerazioni

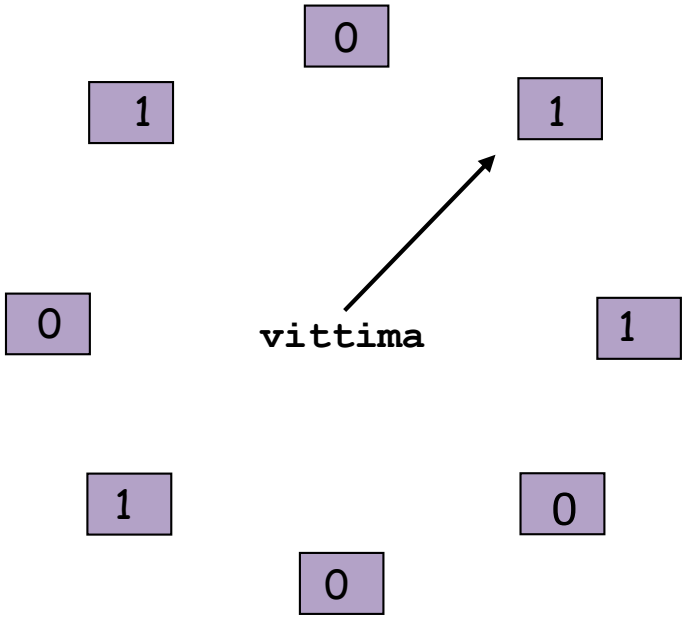
- È una variante dell'algoritmo FIFO che sceglie come vittima la pagina presente in memoria da più tempo e non riferita di recente
- Il bit di riferimento R partiziona le pagine in 2 categorie:
  - R = 1: quelle usate recentemente (stima del *working set*)
  - R = 0: quelle usate meno di recente
- Nel caso più sfavorevole (tutte le pagine hanno R = 1), l'algoritmo seleziona la pagina da cui è iniziata la ricerca e a cui aveva dato una seconda chance
- Usa solo il bit di riferimento R, senza bit supplementari, quindi l'ordine esatto con cui le pagine sono state usate non è noto

Algoritmo dell'orologio

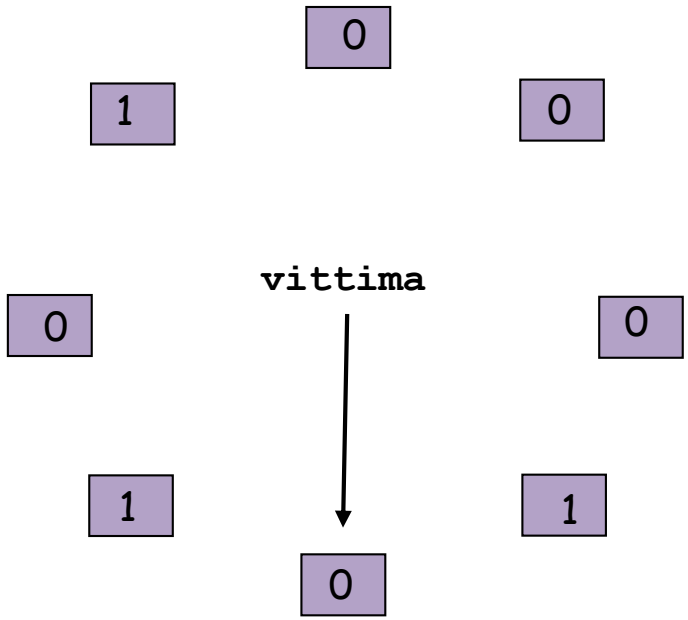
- È un'implementazione efficiente dell'algoritmo con seconda chance in cui le pagine sono gestite come una lista circolare
- Mantiene nella variabile *vittima* l'indice della prima pagina da esaminare (che è la pagina successiva a quella che è stata selezionata per ultima)
- Quando viene invocato, l'algoritmo
  1. Considera la pagina il cui indice è in *vittima*
  2. Controlla il bit di riferimento di tale pagina
     - Se R = 0: seleziona la pagina, incrementa la variabile *vittima* e termina
     - Se R = 1: pone R = 0, incrementa la variabile *vittima* e torna al punto 1
- La pagina selezionata viene sostituita da una nuova pagina che viene inserita nella lista circolare nella posizione corrispondente (con R = 1)

Algoritmo dell'orologio: esempio

L'implementazione dell'algoritmo si basa su una lista circolare in cui il puntatore vittima indica qual è la prima pagina da esaminare



a) all'inizio dell'algoritmo



b) alla fine dell'algoritmo

Questa figura chiarisce perché è noto come algoritmo dell'orologio

Algoritmo con seconda chance vs. Algoritmo dell'orologio

- Alcuni autori distinguono l'algoritmo con seconda chance da quello dell'orologio, ma solo per ciò che concerne l'implementazione:
  - il primo usa una lista (delle pagine) FIFO e le relative operazioni
  - il secondo usa semplicemente la variabile vittima ed operazioni di incremento modulare (non usa operazioni sulle liste)
- Il secondo è una semplificazione del primo
  - Incrementare la variabile vittima equivale infatti a spostare la pagina in fondo alla lista (come se la pagina fosse stata appena caricata in memoria)

Algoritmo con seconda chance migliorato

- Molti sistemi offrono un altro bit oltre a quello di riferimento: il bit di modifica (o dirty bit)
  - È associato al corrispondente elemento della tabella delle pagine
  - È impostato a 0 esclusivamente dal SO
  - È impostato a 1 dall'HW ad ogni accesso in scrittura alla pagina corrispondente
- L'uso del bit di modifica permette di migliorare la scelta fatta dall'algoritmo con seconda chance
  - Se la pagina che dev'essere sostituita non è stata più modificata dal suo ultimo salvataggio in memoria secondaria, il SO non ha necessità di aggiornare il suo contenuto
  - Perciò le pagine non modificate possono essere sostituite molto più velocemente di quelle modificate

Algoritmo con seconda chance migliorato

Le pagine sono classificate in 4 categorie sulla base dei valori dei bit di riferimento e modifica (R,M):

- (0,0): nè recentemente usata nè modificata
  migliore pagina da sostituire
- (0,1): non usata di recente, ma modificata
  non è così buona per la sostituzione perché dovrà essere copiata in memoria secondaria prima di essere sostituita
- (1,0): usata recentemente, ma non modificata
  probabilmente sarà riusata presto, meglio sotituire una pagina modificata che una usata di recente
- (1,1): usata recentemente e modificata
  probabilmente sarà riusata presto, inoltre dovrà essere copiata in memoria secondaria prima di essere sostituita

N.B. Si preferisce sostituire le pagine modificate a quelle usate

Algoritmo con seconda chance migliorato

Quando viene invocato, l'algoritmo

a) Scorre la lista a partire da *vittima* alla ricerca di una pagina etichettata (0,0); se ne trova una, la utilizza e termina dopo aver incrementato *vittima*

b) Altrimenti, scorre (nuovamente) la lista a partire da *vittima*, impostando a 0 il bit di riferimento per tutte le pagine incontrate, alla ricerca di una pagina etichettata (0,1); se ne trova una, la utilizza e termina dopo aver incrementato *vittima*

c) Altrimenti, (tutti i bit di riferimento sono ora impostati a 0) ripete i passi a) e, eventualmente, b) (ciò permetterà sicuramente di trovare una pagina da sostituire)

Per selezionare la nuova vittima la lista delle pagine viene scandita più volte (fino a 4)

Algoritmi basati su conteggio

Vari algoritmi possibili, basati sul conteggio del numero di riferimenti fatti tramite l'uso di un contatore per ogni pagina

- Algoritmo di sostituzione delle pagine meno frequentemente usate (*least frequently used*, LFU)
  - Si sostituisce la pagina con valore minore del contatore
  - Per evitare situazioni anomale (es. pagine molto usate ma solo durante la fase iniziale di un processo), si possono spostare i valori dei contatori a destra di un bit ad intervalli regolari
- Algoritmo di sostituzione delle pagine più frequentemente usate (*most frequently used*, MFU)
  - Si sostituisce la pagina con valore maggiore del contatore
  - Razionale: le pagine con valore minore del contatore sono state usate poco perché inserite più di recente

LFU e MFU non sono molto comuni poiché

- la loro realizzazione è costosa in termini di tempo d'esecuzione
- inoltre, non approssimano bene l'algoritmo ottimo

Tecniche SW di ottimizzazione

Gli algoritmi di sostituzione delle pagine possono beneficiare di tecniche SW che riducono i tempi d'esecuzione delle singole sostituzioni

- Riserva (pool) di frame liberi
- Trasferimento spontaneo delle pagine modificate

Riserva di frame liberi

Il SO gestisce una riserva (pool) di frame liberi da utilizzare per soddisfare nell'immediato le richieste di memoria usando la seguente procedura:

- Ogni richiesta di memoria per un singolo frame viene esaudita attingendo al pool, se possibile
- Ogni frame rilasciato viene inserito nel pool
- L'algoritmo di sostituzione delle pagine è attivato con bassa priorità quando il numero di frame nel pool scende al di sotto di una certa soglia
- Se il pool si svuota completamente, ogni richiesta di frame deve essere esaudita previa sostituzione di una pagina, come al solito

Riserva di frame liberi: vataggi

- Le richieste di frame liberi tendono ad essere esaudite con maggiore velocità
  - Infatti non è necessario attendere la conclusione di un trasferimento in memoria secondaria per esaudire una richiesta
  - Il trasferimento di dati avviene in modo concorrente rispetto all'esecuzione del processo che ha richiesto il frame
- Il SO può tenere traccia delle pagine che erano memorizzate nei frame del pool
  - Un accesso ad una di queste pagine risulta in un page fault che può essere gestito senza trasferimenti dalla memoria secondaria

Trasferimento spontaneo delle pagine modificate

Il SO scandisce periodicamente ma con bassa priorità le tabelle delle pagine e avvia il trasferimento in memoria secondaria delle pagine modificate (resettandone di conseguenza il bit di modifica M)

- In questo modo aumenta la probabilità che alla richiesta di un frame libero possa essere scelta una vittima che non debba essere copiata in memoria secondaria

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Algoritmi di allocazione dei frame

- Un'altra componente del SO legata alla gestione della memoria virtuale è quella che controlla le assegnazioni di memoria fisica ai processi
- Anche a causa della sovrallocazione della memoria, la memoria fisica effettivamente disponibile per un processo può essere inferiore alla dimensione del suo spazio di memoria virtuale
- Varie strategie sono possibili per allocare la memoria libera ai processi

Algoritmi di allocazione dei frame

- Si può usare una lista di frame liberi e utilizzabili per soddisfare le richieste
  - All'avvio del sistema tutta la memoria disponibile (non occupata dal SO) viene inserita nella lista dei frame liberi
  - Man mano che vengono generati page fault, i frame della lista vengono utilizzati per ospitare le pagine riferite
  - Quando la lista si esaurisce, si usa un algoritmo di sostituzione delle pagine per selezionare una pagina da scaricare
- Con paginazione a domanda, quando l'esecuzione di un processo comincia, solo una pagina è caricata in memoria, le altre saranno caricate in conseguenza dei page fault che genererà
  - Per migliorare le prestazioni si può cercare di prevenire l'alto numero di page fault che si verificherebbero al suo avvio, quando il processo tenta di portare in memoria la sua località iniziale, caricando in memoria il suo intero working set (prepaginazione)
  - Similmente, quando il processo è sospeso, si memorizza il suo working set così da riportarlo in memoria prima del suo prossimo riavvio

Numero di frame allocati

- Le strategie di allocazione dei frame ai singoli processi sono soggette a vari vincoli, quali ad esempio
  1. Non si possono assegnare più frame di quelli disponibili
  2. È necessario allocare almeno un numero minimo di frame

- Il numero massimo dipende dalla quantità di memoria fisica disponibile
- Il numero minimo dipende dall'architettura del calcolatore
  - I frame allocati devono essere in numero sufficiente per contenere tutte le pagine cui ogni istruzione può far riferimento
  - Alcune istruzioni possono fare riferimento a più di un frame
- In mezzo, vi è un'ampia gamma di scelte
  - Ovviamente, al decrescere del numero di frame allocati a ciascun processo aumenta il tasso di page fault e decrescono le prestazioni

Algoritmi di allocazione

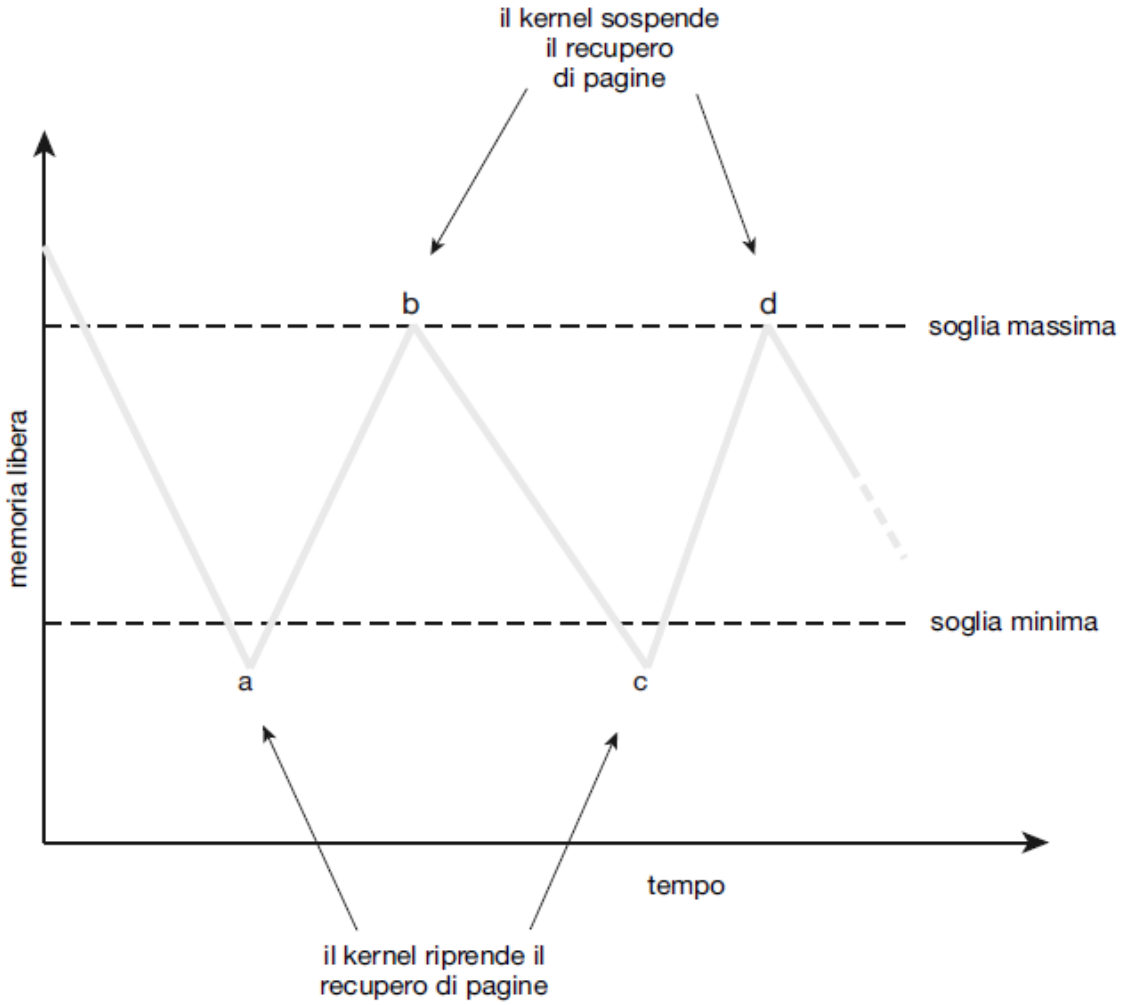
- Due strategie principali di allocazione
  - Uniforme: a tutti i processi viene assegnato lo stesso numero di frame
  - Proporzionale: il numero di frame assegnato ad un processo è proporzionale alla dimensione della sua memoria virtuale o alla sua priorità (però c'è un limite minimo al di sotto del quale non conviene scendere)
- Due politiche di sostituzione
  - Globale: la pagina viene scelta indipendentemente dal processo a cui è allocata
    - L'insieme dei frame allocati ad un processo non dipende solo dal suo comportamento di paginazione ma anche da quello di altri processi
  - Locale: la pagina viene scelta tra quelle allocate al processo che ha generato il page fault
    - L'insieme dei frame allocati ad un processo dipende solo dal suo comportamento di paginazione

La sostituzione globale è la politica più usata perché permette una maggiore produttività del sistema, dato che adegua il numero dei frame assegnati ai processi alle loro esigenze effettive

Implementazione di una politica globale

Strategia: *garantire che ci sia sempre sufficiente memoria libera per soddisfare nuove richieste*

Il kernel del SO attiva e sospende tempestivamente l'attività di recupero di pagine (che usa un qualsiasi algoritmo di sostituzione) facendo in modo che la lunghezza della lista dei frame liberi si mantenga costantemente tra un minimo ed un massimo



Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Thrashing

- Thrashing (o paginazione degenere): *stato in cui l'attività della CPU è principalmente dedicata a trasferire pagine avanti e indietro dalla swap-area e alla gestione di page-fault*
- Causa notevoli problemi di prestazioni
  - Il sistema spende più tempo per la paginazione rispetto al tempo destinato all'esecuzione dei processi applicativi

I scenario

- Consideriamo cosa succede se un processo non ha abbastanza frame per ospitare le pagine del suo working set
- Il processo incorrerà ben presto in un page fault
- A questo punto, se la politica di sostituzione è locale, si dovrà sostituire una pagina del suo working set, che per definizione sarà necessaria a breve
- Di conseguenza, si verificano parecchi page fault perché si sostituiscono pagine che presto dovranno essere riportare in memoria
- Si è in una situazione di thrashing e la *produttività del sistema precipita*

II scenario

- In alcuni sistemi, il SO regola il grado di multiprogrammazione basandosi sull'uso della CPU
  - Se l'utilizzo della CPU è troppo basso, il SO aumenta il grado di multiprogrammazione
- Supponiamo venga utilizzata una politica di sostituzione delle pagine globale e che un processo entri in una fase della sua esecuzione in cui ha bisogno di più frame
  - Il processo genera una serie di page fault che causano la sottrazione di frame ad altri processi
- Questi processi però hanno bisogno di quelle pagine, e quindi a loro volta generano altri page fault, sottraendo frame ad altri processi ancora
- Per effettuare caricamento e scaricamento delle pagine per tutti questi processi si deve utilizzare il dispositivo di paginazione
- Mentre i processi sono in coda sul dispositivo, la coda dei pronti si svuota e così l'utilizzo della CPU diminuisce

II scenario

- Il SO, vedendo diminuire l'utilizzo della CPU, aumenta di conseguenza il grado di multiprogrammazione
- Per cominciare l'esecuzione del nuovo processo vengono sottratti frame ai processi in esecuzione
- Ciò causa ulteriori page fault ed allunga la coda di attesa per il dispositivo di paginazione
- Di conseguenza, l'utilizzo della CPU diminuisce ulteriormente e il SO cerca di aumentare ulteriormente il grado di multiprogrammazione
- Alla fine tutti i processi sono bloccati in attesa di trasferimenti di I/O, e l'unica attività della CPU consiste nell'eseguire l'algoritmo di sostituzione delle pagine
- Si è in una situazione di thrashing e la *produttività del sistema precipita*

Thrashing

- Il thrashing si verifica perché la somma delle dimensioni delle località dentro cui si muovono i processi è superiore alla dimensione della memoria fisica totale del sistema
- Per controllare il thrashing, oltre che usare algoritmi di sostituzione più efficienti, si possono usare
  1. una valutazione approssimata, tramite il working set, delle pagine virtuali che fanno parte delle località dei processi
  2. una frequenza accettabile di mancanze di pagina

Uso del working set

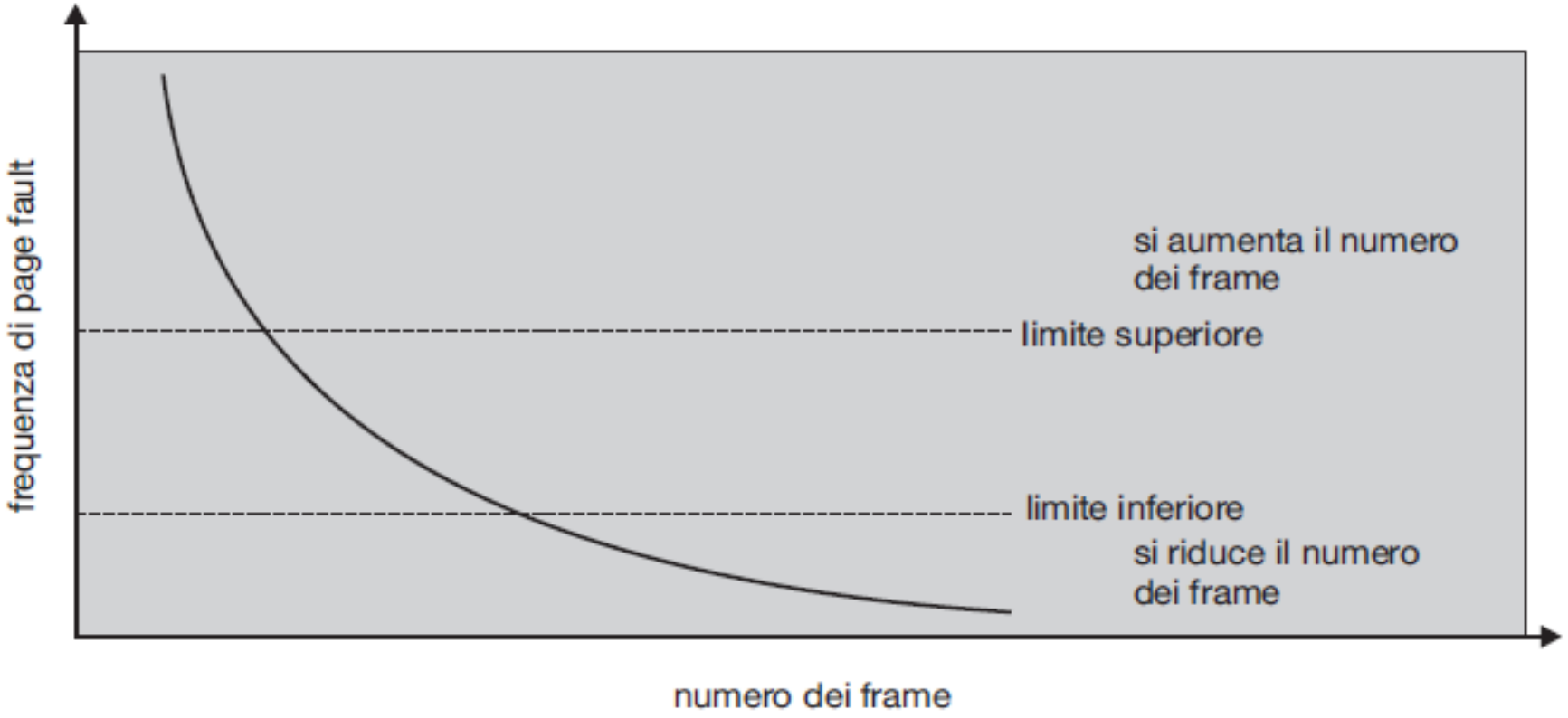
- Idealmente, ad un dato istante, ad ogni processo bisognerebbe assegnare i frame sufficienti ad ospitare tutte le pagine virtuali che fanno parte della sua attuale località
  - Così, finché la località non cambia, non genererà page fault
  - Se si assegnano meno frame rispetto alla dimensione della località corrente, la paginazione del processo degenera perché non si possono tenere in memoria tutte le pagine che il processo sta attivamente utilizzando
- In pratica, si sfrutta il fatto che il working set approssima la località del processo
- Il SO controlla il working set di ogni processo e assegna al processo un numero di frame sufficienti per ospitare il suo working set
- Se i frame ancora liberi sono in numero sufficiente, il SO può anche aumentare il grado di multiprogrammazione

Uso del working set

- Se la somma delle dimensioni dei working set dei processi aumenta, superando il numero totale di frame disponibili, il SO seleziona un processo da sospendere
  - Le pagine del processo vengono scaricate (swap-out) e i suoi frame vengono riallocati ad altri processi
  - Il processo sospeso potrà essere riavviato in seguito
- Questa strategia previene il thrashing mantenendo il più alto grado di multiprogrammazione possibile
- Pertanto, ottimizza l'utilizzo della CPU
- Difficoltà: mantenere aggiornate le info sui working set
  - Si possono usare un interrupt da timer ed un certo numero di bit di memoria per ogni pagina, insieme al bit di riferimento, per approssimare la costituzione del working set (similmente agli algoritmi di sostituzione con bit supplementari di riferimento)

Stabilire una frequenza accettabile di mancanze di pagina

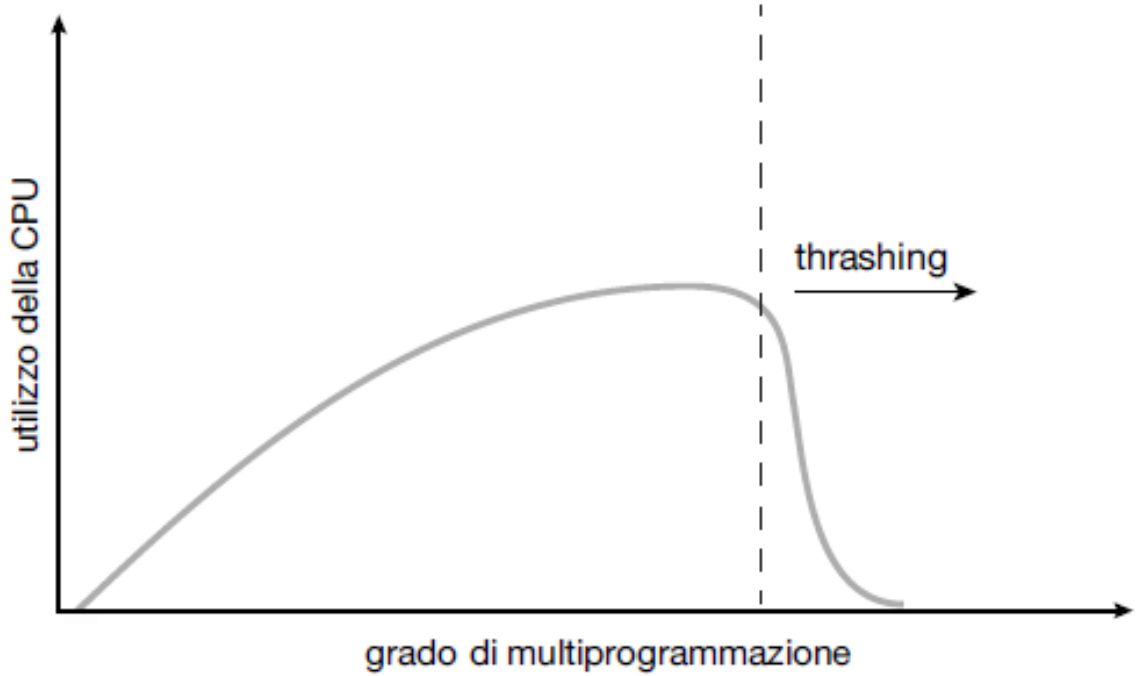
- La frequenza dei page fault generati da un processo, in generale, varia in funzione del numero di frame assegnati al processo
- Se la frequenza attuale è troppo bassa, è possibile che al processo siano stati assegnati troppi frame
- Se la frequenza attuale è troppo alta, il processo ha bisogno di più frame



Utilizzo della CPU in funzione del grado di multiprogrammazione

Queste soluzioni sono a volte solo parziali per cui, per aumentare l'utilizzo della CPU, non resta che ridurre il grado di multiprogrammazione del sistema e fornire ad ogni processo tutti i frame di cui necessita

- Aumentando il grado di multiprogrammazione aumenta l'utilizzo della CPU, anche se via via più lentamente, fino a raggiungere un valore massimo
- Aumentando ulteriormente il grado di multiprogrammazione, l'attività di paginazione degenera e fa crollare l'utilizzo della CPU



Thrashing: considerazioni

- Il thrashing è stato un problema serio per i primi sistemi che hanno utilizzato paginazione a domanda, ad esempio i sistemi time-sharing con decine o centinaia di utenti:
  - Punto di vista del singolo utente: perché dovrei sospendere i miei processi al solo scopo di far progredire i tuoi?

  Il sistema doveva necessariamente gestire il thrashing in maniera automatica!
- I moderni SO non si preoccupano troppo di questo problema: con l'avvento dei PC, gli utenti possono gestirlo direttamente
  - gestendo manualmente l'insieme dei processi attivi, oppure
  - acquistando più memoria

  La memoria è così a buon mercato che non ha senso mettersi nella situazione di dover gestire un sistema in cui la memoria è continuamente sovrautilizzata: meglio comprare più memoria!

Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

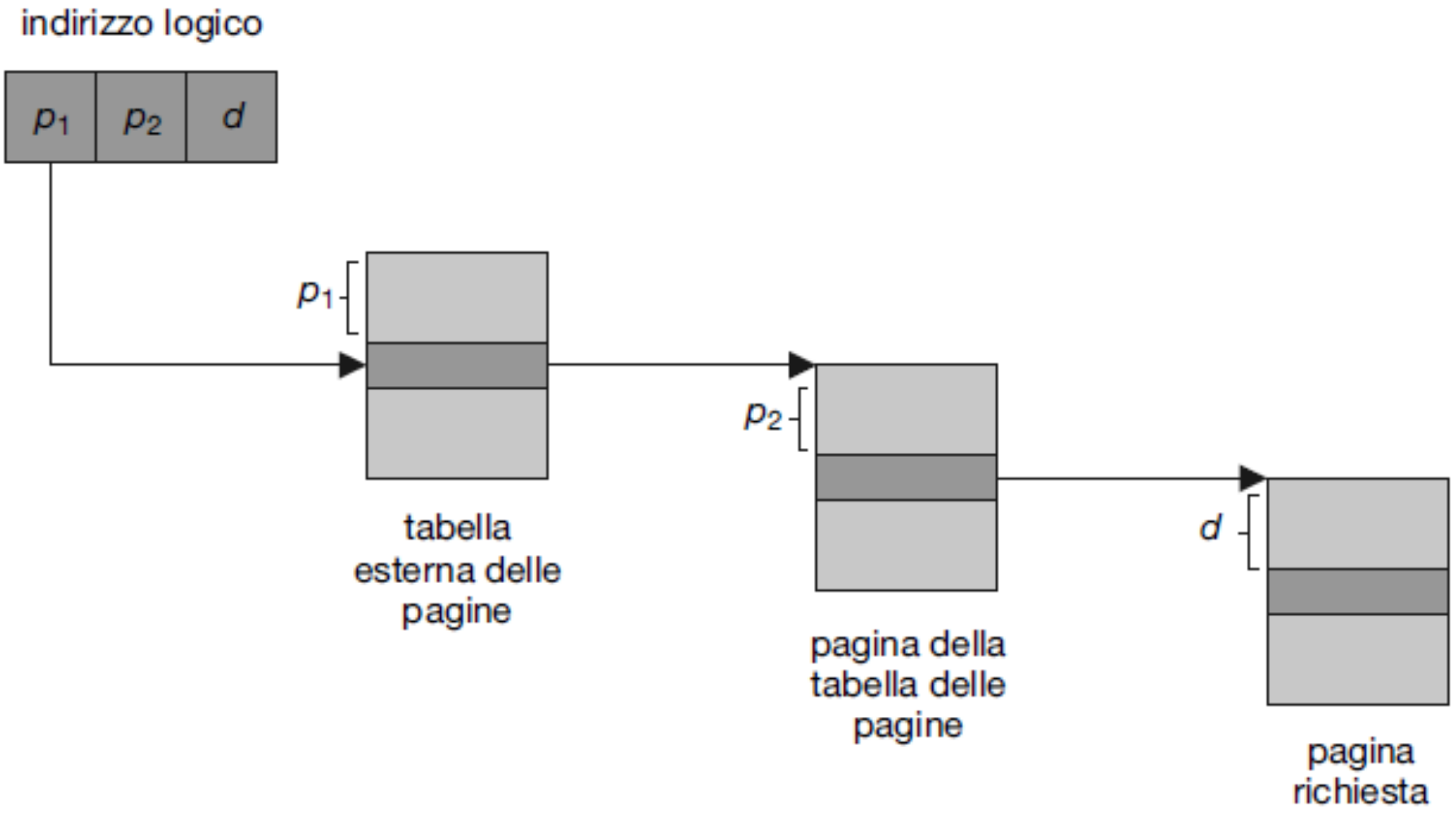
Dimensione delle tabelle delle pagine

- Problema: nei sistemi moderni, la tabella delle pagine di un processo può avere dimensioni enormi
- Esempio: consideriamo un sistema paginato con indirizzi virtuali a 32 bit e dimensione delle pagine 4 KB = 2^{12} byte
  - Un indirizzo logico viene diviso in:
    - uno scostamento di pagina di 12 bit (i 12 bit meno significativi)
    - un indice di pagina di 20 bit (i 20 bit più significativi)
  - Si possono quindi indirizzare fino a 2^{20} pagine
  - La tabella delle pagine dovrebbe avere altrettanti elementi
  - Se ogni elemento fosse costituito da 4 byte (l'equivalente di un indirizzo virtuale), servirebbero 4MB (= 2^{20} x 4 byte) di memoria fisica, cioè 1024 pagine contigue, solo per ospitare la tabella delle pagine!
- Soluzione: strutturare la tabella delle pagine
  - tabella delle pagine organizzata in maniera gerarchica
  - tabella delle pagine di tipo hash
  - tabella delle pagine invertita

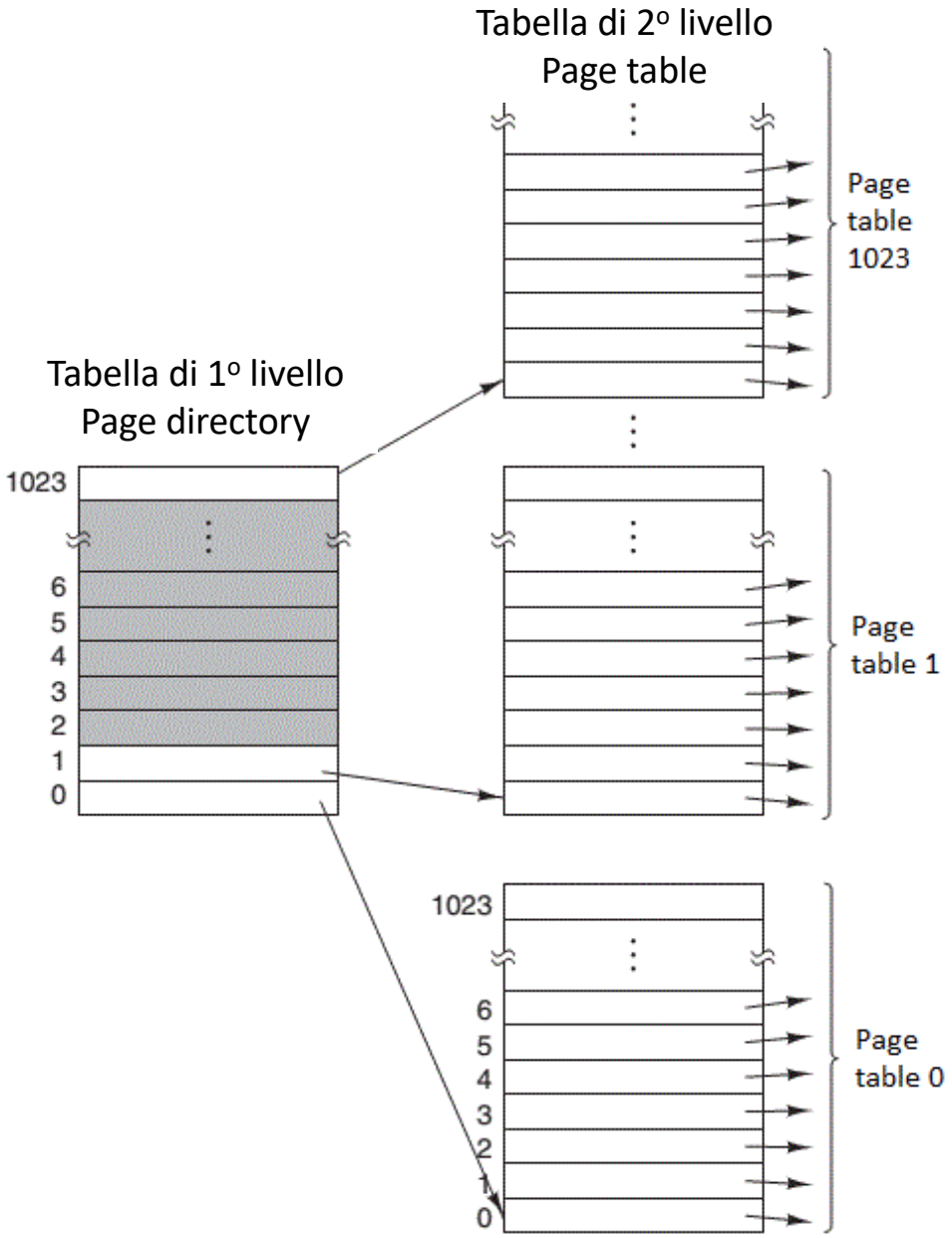
Esempio di paginazione a due livelli

- Una tecnica relativamente semplice è quella della **paginazione gerarchica a 2 livelli** usata nei microprocessori Intel a 32 bit (ma anche in altre CPU a 32 bit, quali Mororola 68000, SPARC, ...)
- La tabella delle pagine, composta da 2^{20} elementi di 4 byte ciascuno, è suddivisa in 2^{10} porzioni consecutive, ciascuna di 2^{10} elementi di 4 byte (ogni porzione occupa 4KB, cioè un frame)
- Le porzioni costituiscono le tabelle delle pagine di 2° livello e possono essere allocate in memoria fisica in modo non contiguo e solo se necessario
- Una tabella di 1° livello (*page directory*) con 2^{10} elementi, uno per ogni porzione, è mantenuta in memoria fisica quando il processo è in esecuzione
- L'indirizzo della page directory è mantenuto dal registro PDAR (Page Directory Address Register)
- Se una tabella di 2° livello è presente in memoria, il numero del frame che la contiene è mantenuto nell'elemento della page directory corrispondente alla tabella
- Perciò, il numero di pagina *pg* di 2^{20} bit è suddiviso in:
  - un indice di pagina *dr* di 10 bit (i più significativi) per accedere alla page directory
  - un scostamento di pagina *stp* di 10 bit (i meno significativi)

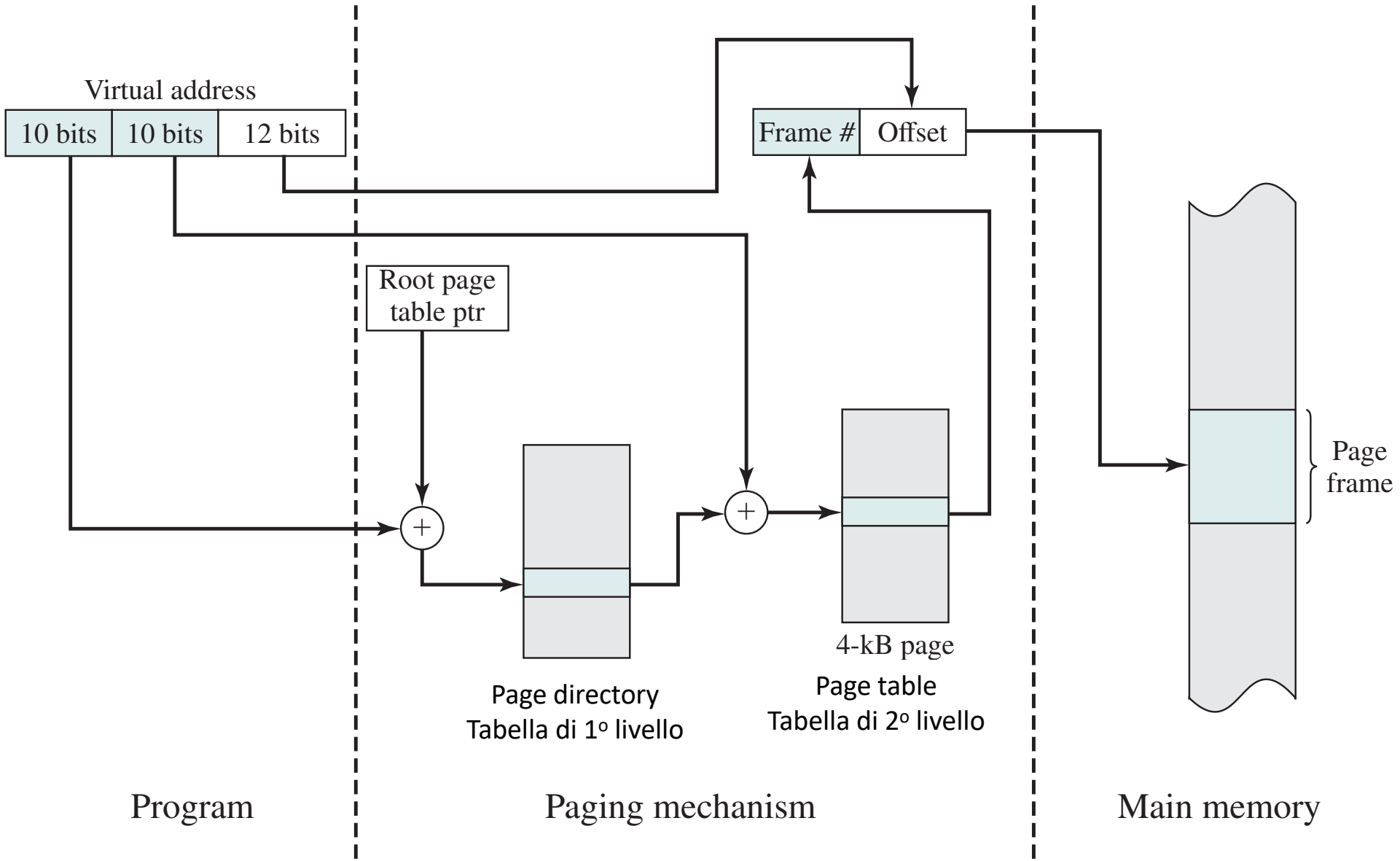
Organizzazione di un indirizzo a 32 bit



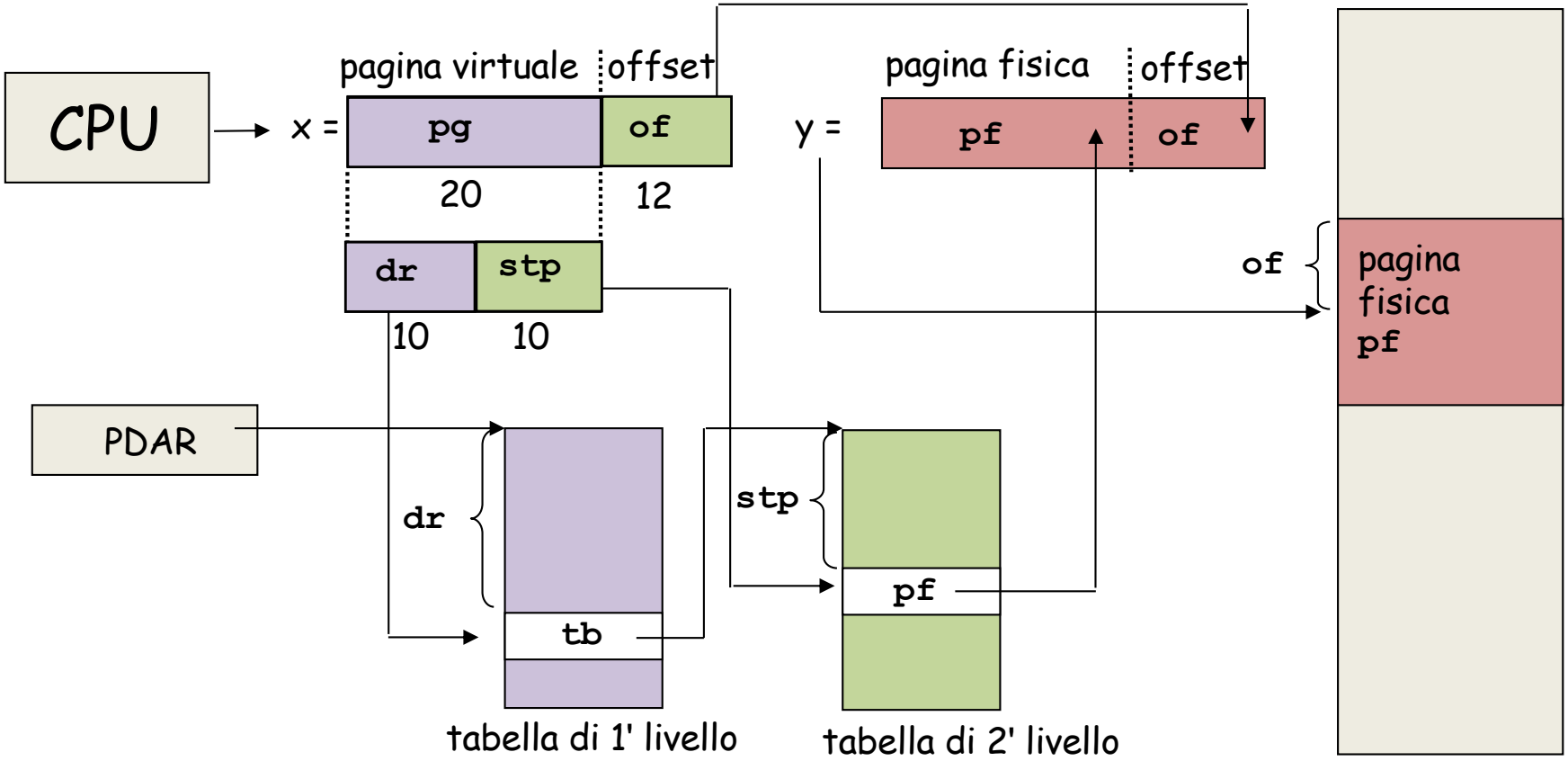
Schema di una tabella delle pagine a 2 livelli



Schema di traduzione con tabella delle pagine a 2 livelli



Schema di traduzione degli indirizzi



Poiché la traduzione degli indirizzi si svolge dalla tabella esterna delle pagine verso l'interno, questo metodo è anche noto come tabella delle pagine ad associazione diretta (*forward-mapped page table*)

Schema di una tabella delle pagine a 2 livelli

Vantaggio: tenere in memoria solo le tabelle delle pagine necessarie

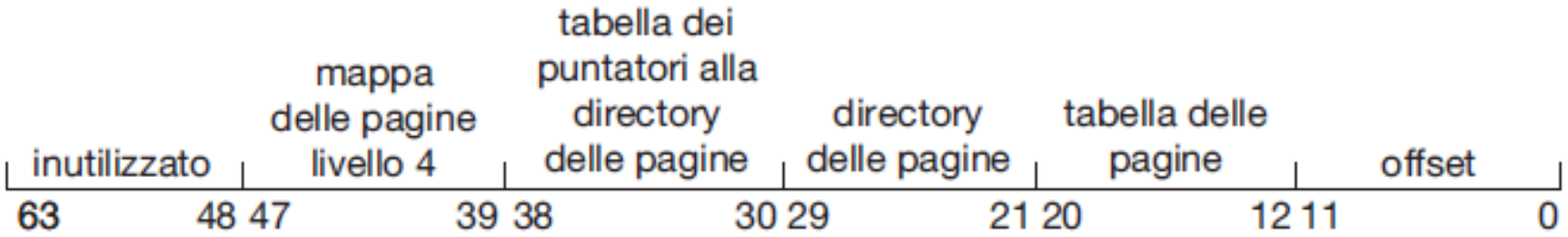
- Solo la tabella delle pagine di primo livello deve essere necessariemente in memoria principale
- Le tabelle dei livelli inferiori vengono allocate solo se quelle porzioni dello spazio di indirizzi virtuali del processo sono utilizzate

Paginazione a livelli e indirizzi virtuali di 64 bit

- Nel caso di sistemi con spazi di indirizzi virtuali di 64 bit, se la dimensione delle pagine è di 4KB (= 2^{12} byte), la tabella delle pagine potrebbe contenere fino a 2^{52} elementi
  - Con descrittori di pagina a 8 byte avremmo che la tabella delle pagine di ogni processo occuperebbe 8×2^{52} byte = 2^{55} byte = 32PB!
- Bisogna adottare uno schema di paginazione a più di due livelli

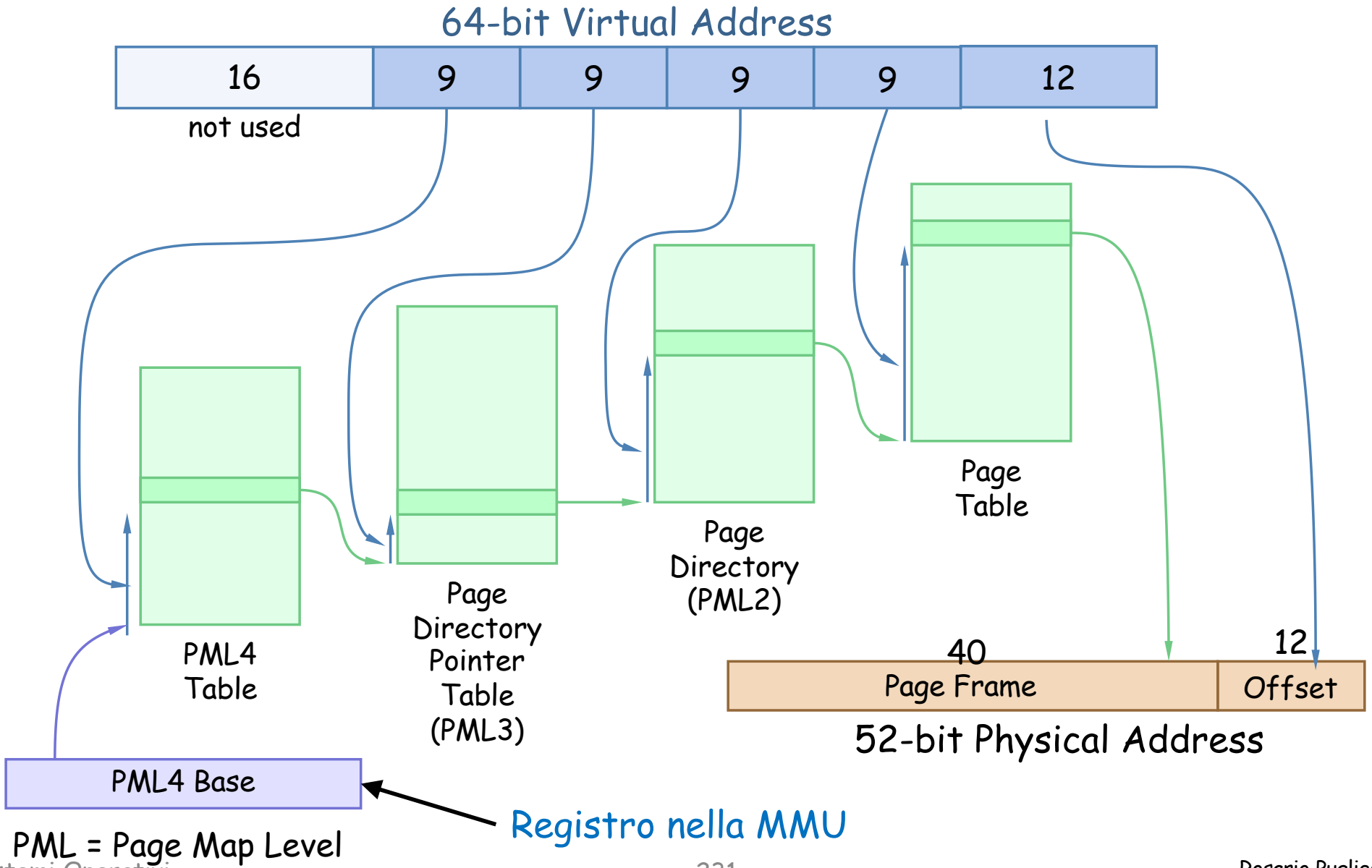
Tabelle delle pagine multilivello: Intel x86-64

- Consideriamo l'architettura di indirizzamento di Intel x86-64
- Indirizzi virtuali a 64 bit consentono di indirizzare potenzialmente 16 exabyte (=2^{64} byte) di memoria, ma vengono effettivamente utilizzati solo i 48 bit inferiori
- Pagine di 4 Kbyte (= 4096 byte): i 12 bit meno significativi sono l'offset all'interno della pagina
- 4 livelli di tabelle delle pagine, ciascuno indicizzato con 9 bit di indirizzo virtuale
- Ogni tabella delle pagine è ospitata in una pagina (gli elementi della tabella delle pagine sono di 8 byte)



Traduzione degli indirizzi in x86-64

È necessario effettuare 4 accessi in memoria per tradurre un indirizzo!



Tempo effettivo di accesso in memoria

- Dato che ogni livello è memorizzato in RAM, la conversione dell'indirizzo logico in indirizzo fisico può necessitare di diversi accessi alla memoria
- L'uso del TLB (per fare il caching degli indirizzi delle pagine accedute più frequentemente) permette di ridurre drasticamente l'impatto degli accessi multipli
- Per esempio, con paginazione a 4 livelli, indicando con p, con $0 \leq p \leq 1$, la probabilità che la ricerca in TLB abbia successo (TLB hit), la formula per il calcolo del Tempo effettivo di accesso in memoria (Effective Memory Access Time, EMAT) diventa

EMAT = p x (tempo accesso TLB + tempo accesso memoria)
+ (1 - p) x (5 x tempo accesso memoria
[+ tempo accesso TLB])

Paginazione a livelli e indirizzi virtuali di 64 bit

- Con lo schema di paginazione a livelli, per tradurre ciascun indirizzo logico, l'UltraSPARC a 64 bit avrebbe 7 livelli di paginazione e richiederebbe un numero proibitivo di accessi alla memoria
- Perciò le tabelle delle pagine multilivello sono in genere considerate inappropriate per le architetture a 64 bit "vere"

Tabella delle pagine di tipo hash

- Organizzazione spesso usata per trattare spazi di indirizzi più grandi di 32 bit
- Ogni voce della tabella hash contiene una lista concatenata di elementi che hanno lo stesso valore della funzione hash utilizzata
- Ciascun elemento della lista è composto da tre campi:
  - Il numero della pagina virtuale (più in generale, il suo descrittore)
  - L'indirizzo del frame che ospita la pagina virtuale
  - Un puntatore all'elemento successivo nella lista
- Quando viene generato un indirizzo virtuale, il suo numero di pagina è fornito in input alla funzione hash il cui output è usato come indice nella tabella hash
- La lista di descrittori di pagina così individuata viene scandita, elemento per elemento, alla ricerca dell'elemento corrispondente alla pagina virtuale che si vuole accedere
- Dopodiché, si estrae l'indice del frame corrispondente

Tabella delle pagine di tipo hash
Schema di traduzione degli indirizzi

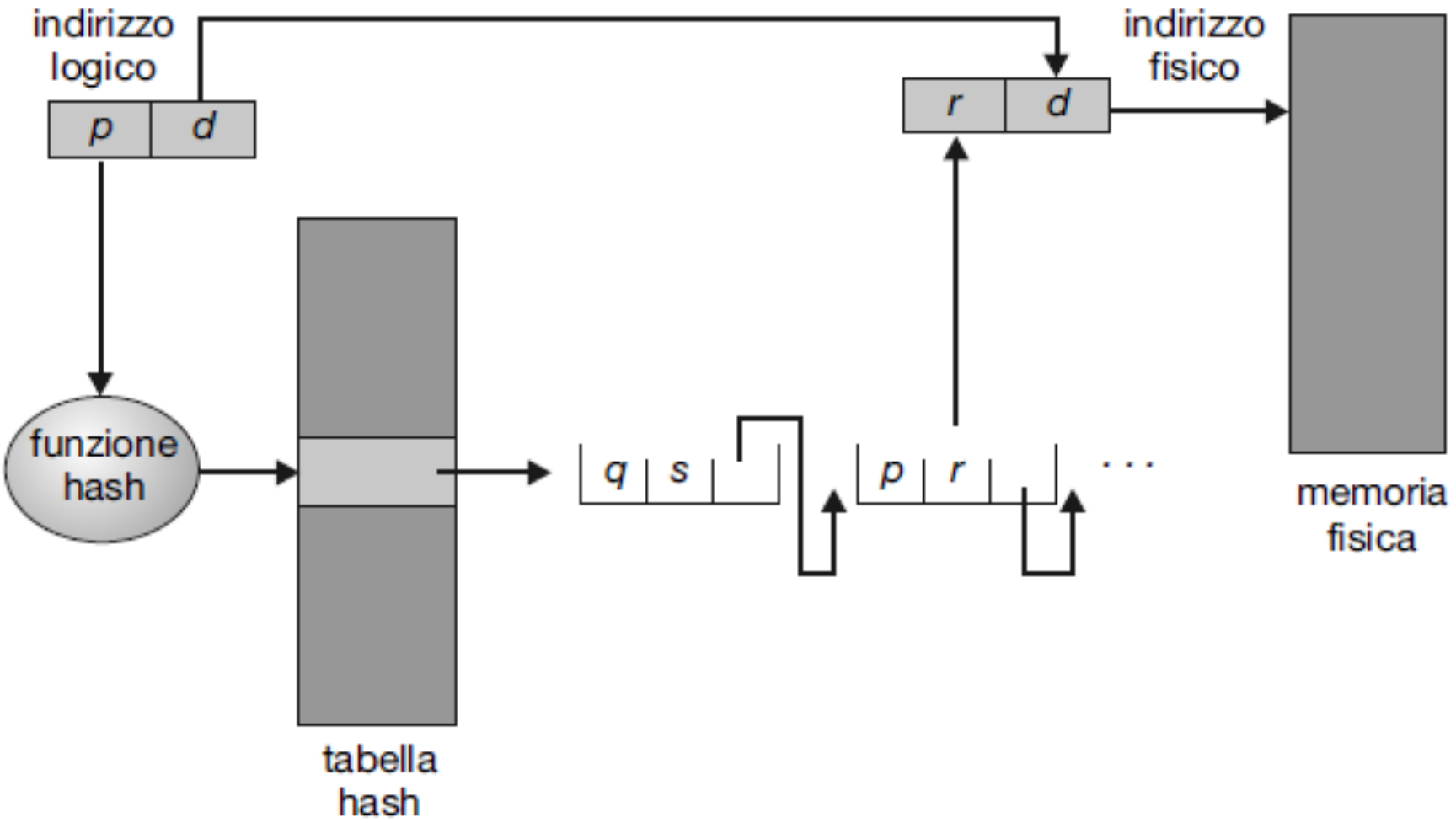


Tabella delle pagine a gruppi (clustered page table)

- Variante adatta a spazi di indirizzi a 64 bit (es. usata da Oracle Solaris su CPU SPARC)
- È similile ad una tabella delle pagine di tipo hash da cui però differisce per il fatto che ciascun elemento di una lista non è un singolo descrittore di pagina ma un gruppo (cluster) di descrittori di pagine virtuali contigue (es. 16)
- È particolarmente utile per gli spazi di indirizzi sparsi in cui i riferimenti alla memoria non sono contigui ma distribuiti per tutto lo spazio di indirizzi

Tabella delle pagine a gruppi
Schema di traduzione degli indirizzi

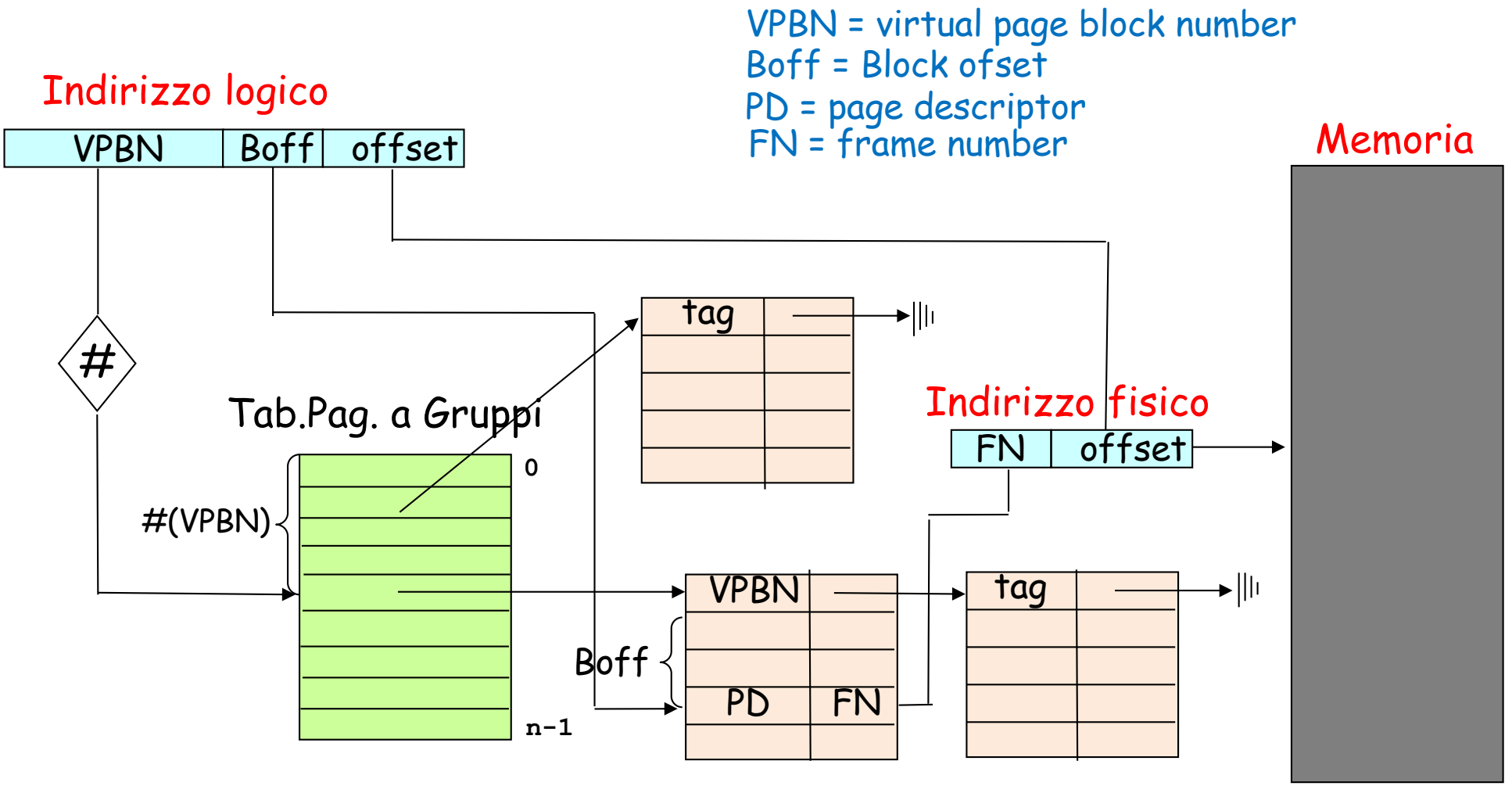
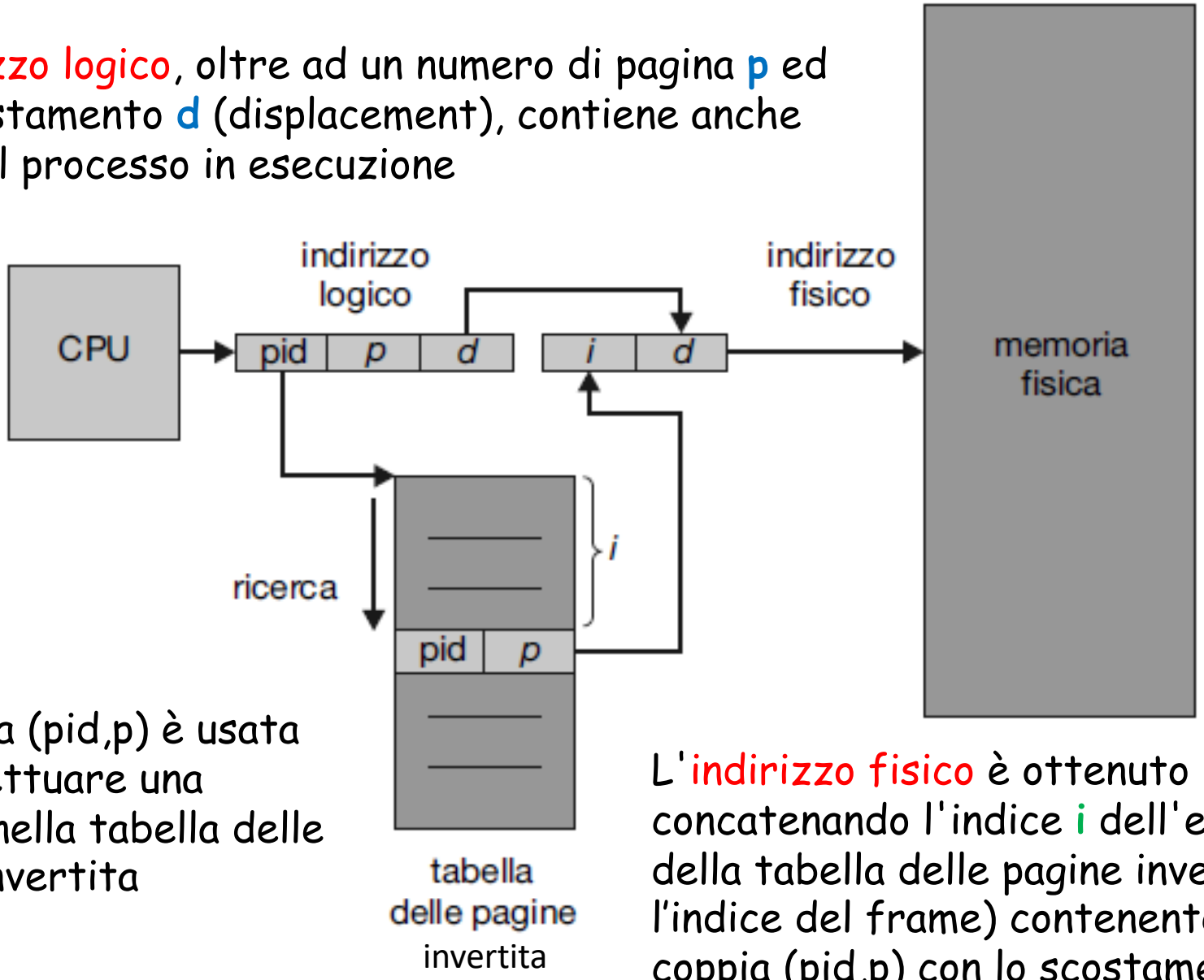


Tabella delle pagine invertita

- Si usa una sola tabella delle pagine per tutto il sistema
  - La tabella ha un elemento per ogni pagina fisica della memoria
  - Ciascun elemento contiene, tra l'altro,
    - l'identificativo dello spazio virtuale (*address-space identifier, ASID*) del processo a cui la pagina virtuale ospitata appartiene
    - l'indirizzo virtuale della pagina logica memorizzata in quella posizione di memoria fisica
- Schema usato su diversi RISC a 64 bit (es. UltraSPARC, PowerPC), dove la tabella delle pagine di ogni singolo processo potrebbe occupare petabytes di memoria
  - Es. con pagine da 4KB, e quindi 2^{52} pagine, e descrittori di pagina a 8 byte avremmo che ogni tabella delle pagine occuperebbe 8×2^{52} byte = 2^{55} byte = 32PB!
- Anche con una dimensione della pagina molto grande, il numero di voci nella tabella delle pagine è enorme
  - Es. con pagine di 4MB e indirizzi virtuali a 64 bit, sono necessarie 2^{42} voci nella tabella delle pagine

Tabella delle pagine invertita
Schema di traduzione degli indirizzi

L'indirizzo logico, oltre ad un numero di pagina p ed uno scostamento d (displacement), contiene anche il pid del processo in esecuzione



La coppia (pid,p) è usata per effettuare una ricerca nella tabella delle pagine invertita

L'indirizzo fisico è ottenuto concatenando l'indice i dell'elemento della tabella delle pagine invertita (è l'indice del frame) contenente la coppia (pid,p) con lo scostamento d

Tabella delle pagine invertita: considerazioni

- Vataggio: permette di diminuire la quantità di memoria principale necessaria
  - Si apprezza maggiormente quando lo spazio virtuale è molto superiore alla memoria fisica
- Non contiene le informazioni che sono necessarie in caso di page fault per portare in memoria la pagina riferita
  - Serve una tabella delle pagine per ogni processo che contiene le informazioni relative alla locazione in memoria secondaria di ciascuna pagina virtuale
  - Tali tabelle servono solo in caso di page fault quindi possono non risiedere in memoria, nel qual caso il sistema di paginazione genera un ulteriore page fault!

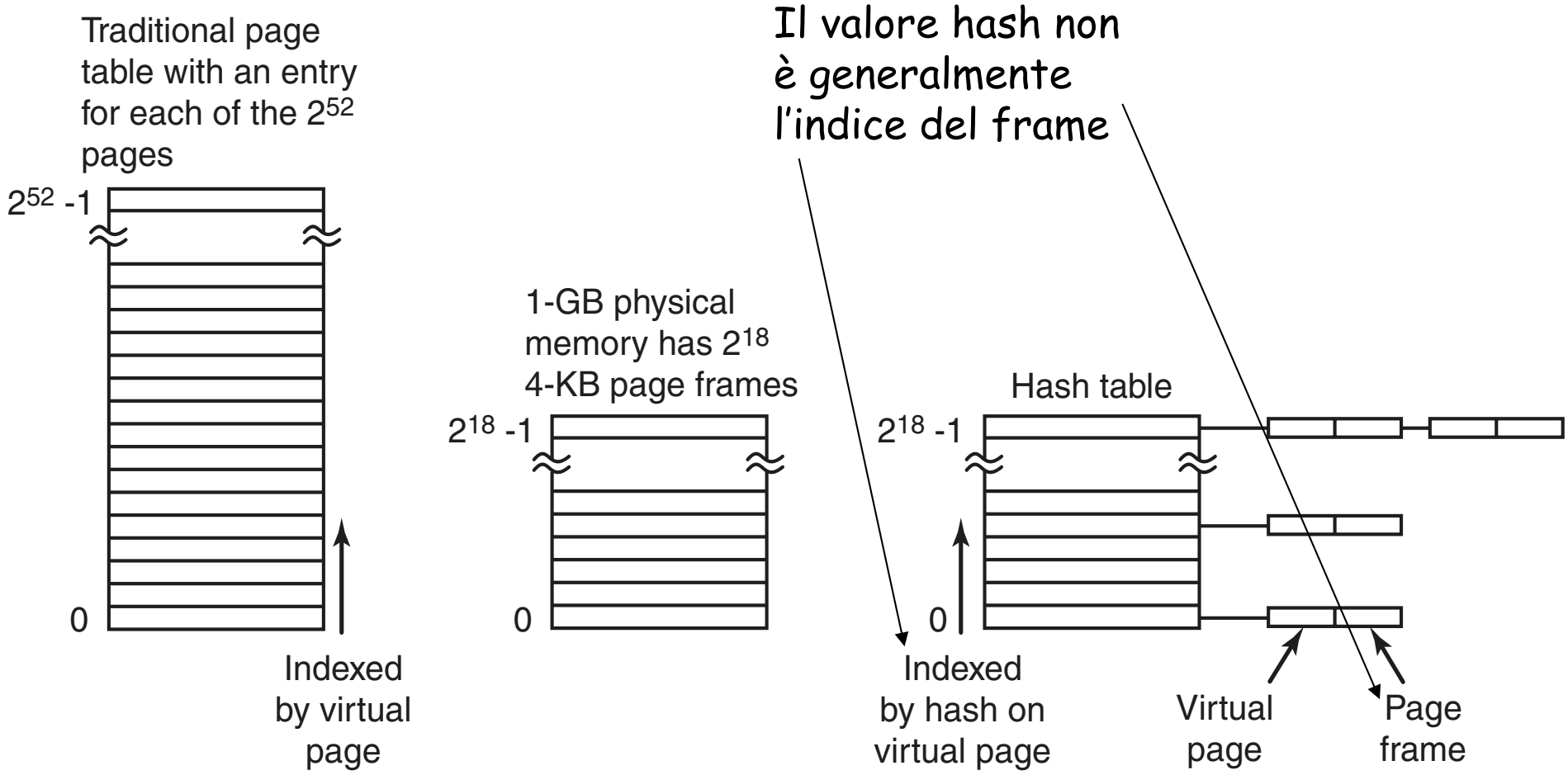
Tabella delle pagine invertita: considerazioni

- Svantaggio: condivisione della memoria più costosa
  - Con l'organizzazione standard della tabella delle pagine, ogni processo ha la propria tabella, il che consente di mappare pagine virtuali di processi differenti sullo stesso frame
  - Con la tabella delle pagine invertita, ogni pagina fisica è mappata in una sola pagina virtuale, quindi in un dato istante un solo indirizzo virtuale può essere mappato su un dato indirizzo fisico
  - Un riferimento da parte di un altro processo che condivide la memoria provocherà un page fault che sostituirà la pagina virtuale su cui è mappata la pagina fisica

Tabella delle pagine invertita: considerazioni

- Svantaggio: la traduzione diventa più difficile
  - Aumenta il tempo di ricerca nella tabella poiché la tabella della pagine invertita è ordinata per indirizzi fisici, mentre le ricerche si fanno per indirizzi virtuali
  - *Ad ogni riferimento in memoria*, l'HW deve cercare nell'intera tabella delle pagine invertita la voce (*pid,p*)
  - L'impiego di un TLB fornisce una soluzione parziale
- Si può usare una tabella hash (unica per tutto il sistema) per limitare la ricerca a uno o al più a pochi elementi
  - Tutte le coppie (*pid,p*) che hanno lo stesso hash sono collegate insieme, ognuna con il proprio indice di frame

Tabella delle pagine tradizionale vs tabella delle pagina invertita con hash



Gestione della memoria principale e virtuale

- Analogie e differenze con la gestione della CPU
- Hardware di base
- Associazione degli indirizzi
- Spazi di indirizzi logici e fisici
- Memoria virtuale
- Swapping
- Aspetti caratterizzanti
- Memoria partizionata
- Segmentazione
- Paginazione
- Segmentazione con paginazione
- Copiatura su scrittura
- Algoritmi di sostituzione delle pagine
- Algoritmi di allocazione dei frame
- Thrashing
- Organizzazione delle tabelle delle pagine
- Considerazioni

Alcune considerazioni

- Rilocazione dinamica degli indirizzi
- Portata del TLB
- Dimensione delle pagine
- Interazione tra memoria virtuale e I/O

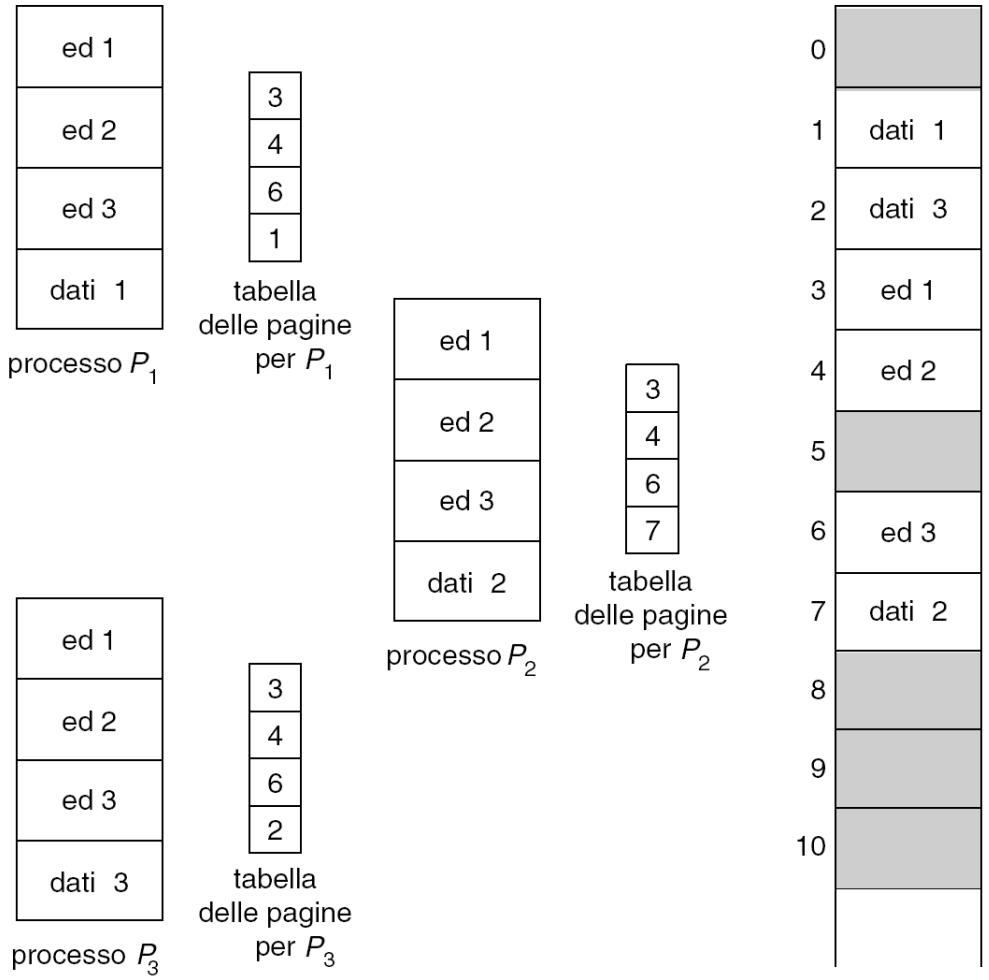
Rilocazione dinamica degli indirizzi

- Il context switch è più costoso perché, per cambiare la funzione di rilocazione:
  - bisogna commutare le informazioni presenti nella MMU
  - eventualmente invalidare i registri associativi TLB

  che contengono dati relativi alla funzione di rilocazione del processo che era precedentemente in esecuzione
- Impone dei vincoli alla condivisione delle informazioni (codice, dati, …)

Condivisione delle informazioni

Se una porzione di uno spazio virtuale è condivisa allora è necessario che essa sia allocata nelle stesse posizioni negli spazi virtuali dei processi interessati alla condivisione



Portata del TLB

- Tasso di successi di un TLB: percentuale di traduzioni di indirizzi virtuali risolte dal TLB senza dover usare la tabella delle pagine in memoria
  - Proporzionale al numero di elementi del TLB
- Idealmente il TLB dovrebbe contenere i metadati relativi al working set del processo in esecuzione
- La portata del TLB esprime la quantità di memoria accessibile tramite il TLB
  - Numero di elementi del TLB moltiplicato per la dimensione delle pagine
- Per aumentare la portata del TLB si può
  - aumentare il numero dei suoi elementi: però la memoria associativa usata per costruire il TLB è costosa e consuma molta energia
  - aumentare la dimensione delle pagine
  - affidarne la gestione al SO per permettere che i suoi elementi possano far riferimento a pagine di dimensioni differenti e maggiori di quelle standard (fino anche a 2 MB)

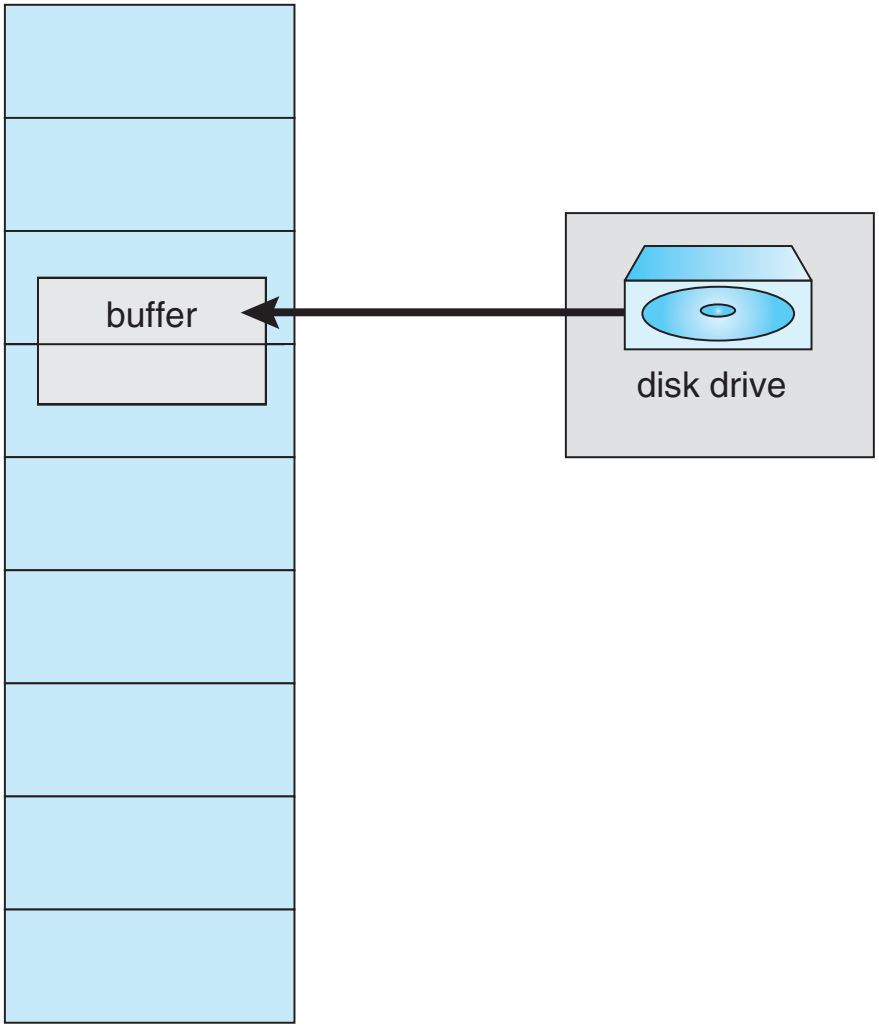
Dimensione delle pagine

- La dimensione delle pagine è tipicamente compresa tra 4KB e 8KB
  - Ma in alcuni sistemi arriva fino a 2MB (es. *huge page* in Linux)
- Alcuni fattori sono a favore delle piccole dimensioni
  - Frammentazione interna: mediamente metà dell'ultima pagina di un processo è sprecata
  - Memoria totale allocata e attività complessiva di I/O: pagine di piccole dimensioni si adattano con più precisione alla località di un programma permettendo di portare in memoria solo ciò che è necessario
- Altri fattori sono a favore delle grandi dimensioni
  - Dimensione della tabella delle pagine
  - Numero di page fault
  - Portata del TLB
  - Tempo complessivo di I/O: a parità di dati letti o scritti, è preferibile trasferire meno pagine ma più grandi
- La tendenza nei sistemi moderni è verso l'incremento della dimensione delle pagine

Interazione tra memoria virtuale e I/O

- Un processo ha appena invocato una system call per leggere da un certo file o dispositivo di I/O ed inserire i dati letti in un buffer all'interno del suo spazio di indirizzi
- Il processo viene sospeso in attesa che si completi l'I/O e un altro processo viene eseguito
- Se il processo in esecuzione origina un page fault e l'algoritmo di sostituzione delle pagine è globale esiste una possibilità non nulla che una pagina che ospita il buffer per l'I/O venga selezionata come vittima per essere rimossa dalla memoria
- Se un dispositivo di I/O sta facendo un trasferimento in DMA a quella pagina, la rimozione della pagina farebbe sì che parte dei dati venga scritta nel buffer e parte nella pagina appena caricata!
- Una situazione simile si potrebbe presentare anche in caso di swapping di un intero processo

Interazione tra memoria virtuale e I/O



Vincolo di I/O e vincolo delle pagine

Possibili soluzioni:

- Eseguire tutto l'I/O in appositi buffer del kernel del SO e copiare in seguito i dati nelle pagine dei processi utente
- Bloccare in memoria le pagine che ospitano un buffer di I/O, in modo che non possano essere rimosse
  - Tale blocco è detto pinning ed è effettuato tramite l'impostazione di un bit nel descrittore di pagina, detto bit di lock