

Sistemi Operativi

Modulo 5: Gestione della memoria *A.A. 2021-2022*

Renzo Davoli
Alberto Montresor

Copyright © 2002-2023 Renzo Davoli, Alberto Montresor

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at: <http://www.gnu.org/licenses/fdl.html#TOC1>

Sommario

- ◆ Binding, loading, linking
- ◆ Allocazione contigua
- ◆ Paginazione
- ◆ Segmentazione
- ◆ Memoria virtuale

Introduzione

- ♦ **Memory manager**
 - ♦ la parte del sistema operativo che gestisce la memoria principale si chiama *memory manager*
 - ♦ in alcuni casi, il memory manager può gestire anche parte della memoria secondaria, al fine di emulare memoria principale
- ♦ **Compiti di un memory manager**
 - ♦ tenere traccia della memoria libera e occupata
 - ♦ allocare memoria ai processi e deallocarla quando non più necessaria
- ♦ **NB: memory manager è software (componente del S.O)**
- ♦ **NB: la Memory Management Unit (MMU) è hardware**

Introduzione

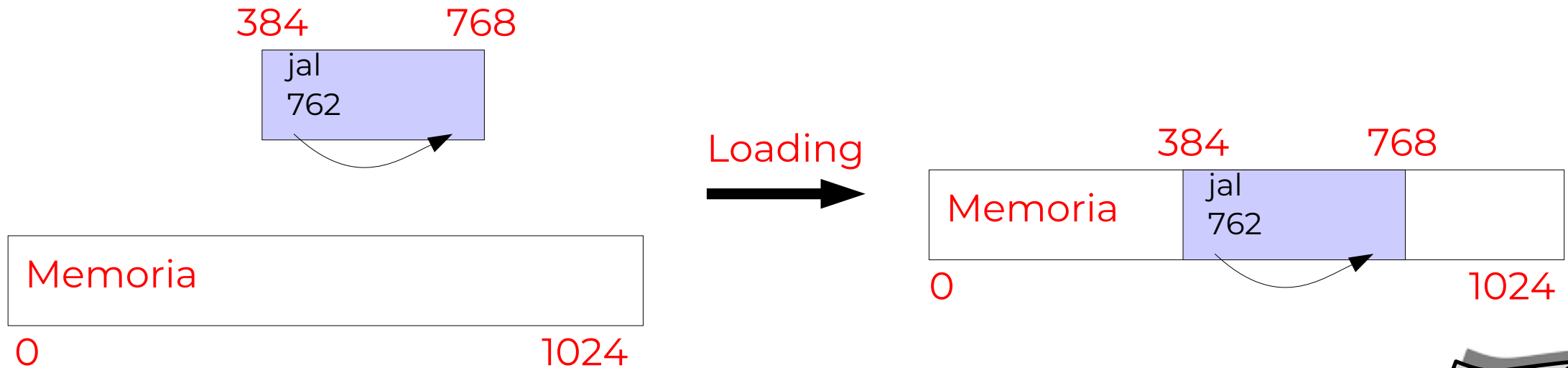
- ◆ **Prospettiva storica**
 - ◆ partiremo vedendo i meccanismi di gestione della memoria più semplici;
 - ◆ a volte possono sempre banali, ma...
- ◆ **... ma nell'informatica, la storia ripete se stessa:**
 - ◆ alcuni di questi meccanismi vengono ancora utilizzati in sistemi operativi speciali per palmari, sistemi embedded (microcontrollori), smart-card

Binding

- ◆ **Definizione**
 - ◆ con il termine *binding* si indica l'associazione di indirizzi logici di memoria (e.g. nomi di variabili, label) ai corrispondenti indirizzi fisici
- ◆ **Il binding può avvenire**
 - ◆ durante la compilazione
 - ◆ durante il caricamento
 - ◆ durante l'esecuzione

Binding

- ♦ Binding durante la compilazione
 - ♦ gli indirizzi vengono calcolati al momento della compilazione e resteranno gli stessi ad ogni esecuzione del programma
 - ♦ il codice generato viene detto codice *assoluto*
 - ♦ Esempi:
 - ♦ codice per microcontrollori, per il kernel, file COM in MS-DOS

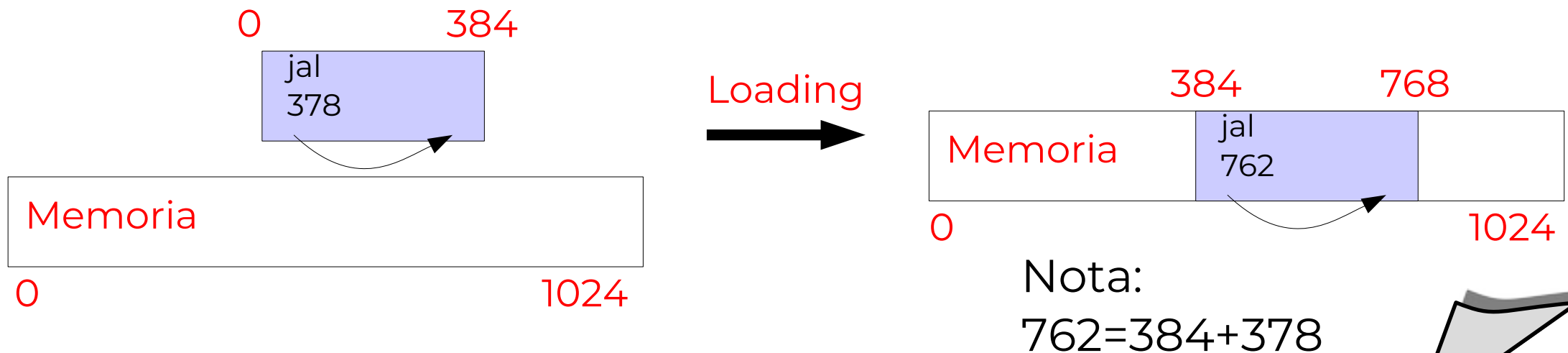


Binding

- ♦ Binding durante la compilazione
 - ♦ vantaggi
 - ♦ non richiede hardware speciale
 - ♦ semplice
 - ♦ molto veloce
 - ♦ svantaggi
 - ♦ non funziona con la multiprogrammazione

Binding

- ◆ Binding durante il caricamento
 - ◆ il codice generato dal compilatore non contiene indirizzi assoluti ma relativi (al PC oppure ad un indirizzo base)
 - ◆ questo tipo di codice viene detto *rilocabile*
- ◆ Durante il caricamento
 - ◆ il loader si preoccupa di aggiornare tutti i riferimenti agli indirizzi di memoria coerentemente al punto iniziale di caricamento



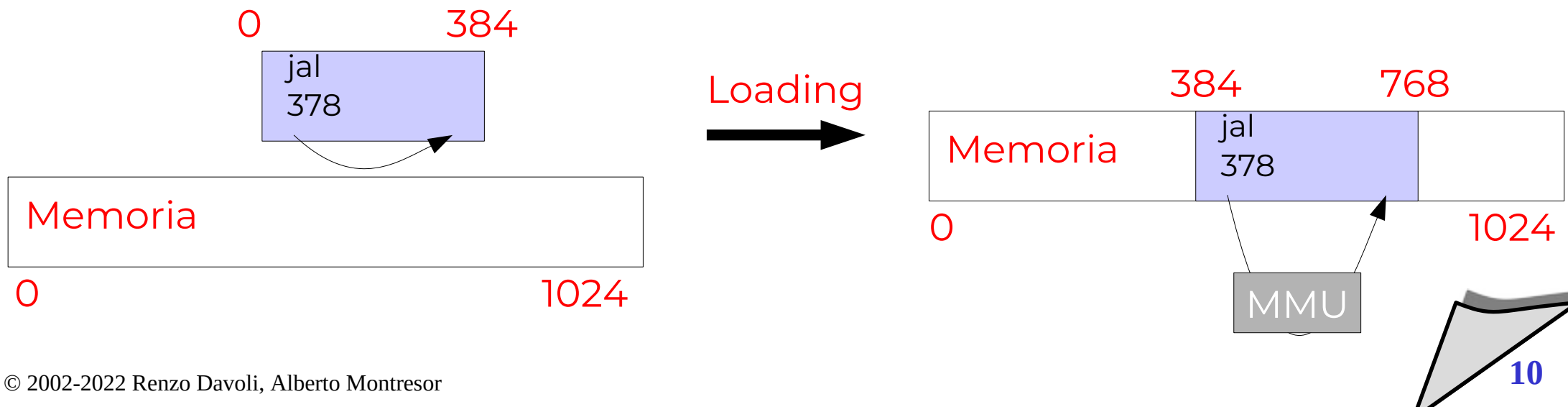
Binding

- ♦ **Binding durante il caricamento**
 - ♦ vantaggi
 - ♦ permette di gestire multiprogrammazione
 - ♦ non richiede uso di hardware particolare
 - ♦ svantaggi
 - ♦ richiede una traduzione degli indirizzi da parte del loader, e quindi formati particolare dei file eseguibili

Binding

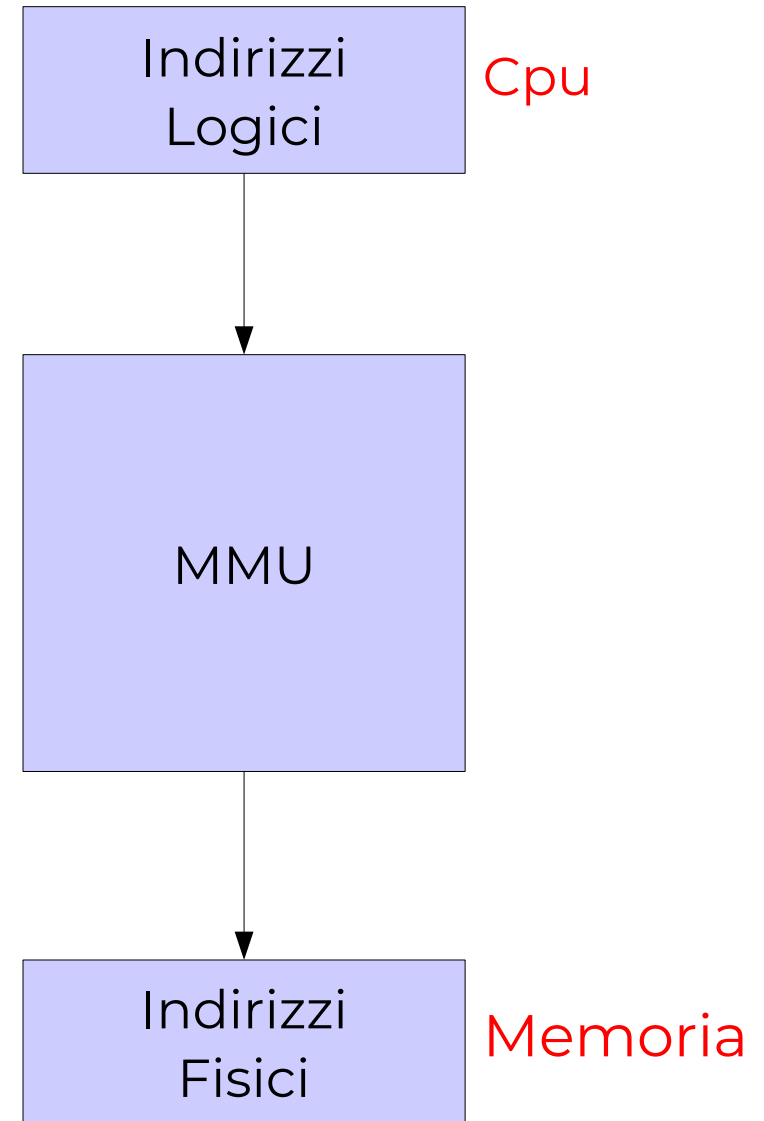
- Binding durante l'esecuzione

- l'individuazione dell'indirizzo di memoria effettivo viene effettuata durante l'esecuzione da un componente hardware apposito: la *memory management unit* (MMU)
- da non confondere con il memory manager (MM)



Indirizzi logici e indirizzi fisici

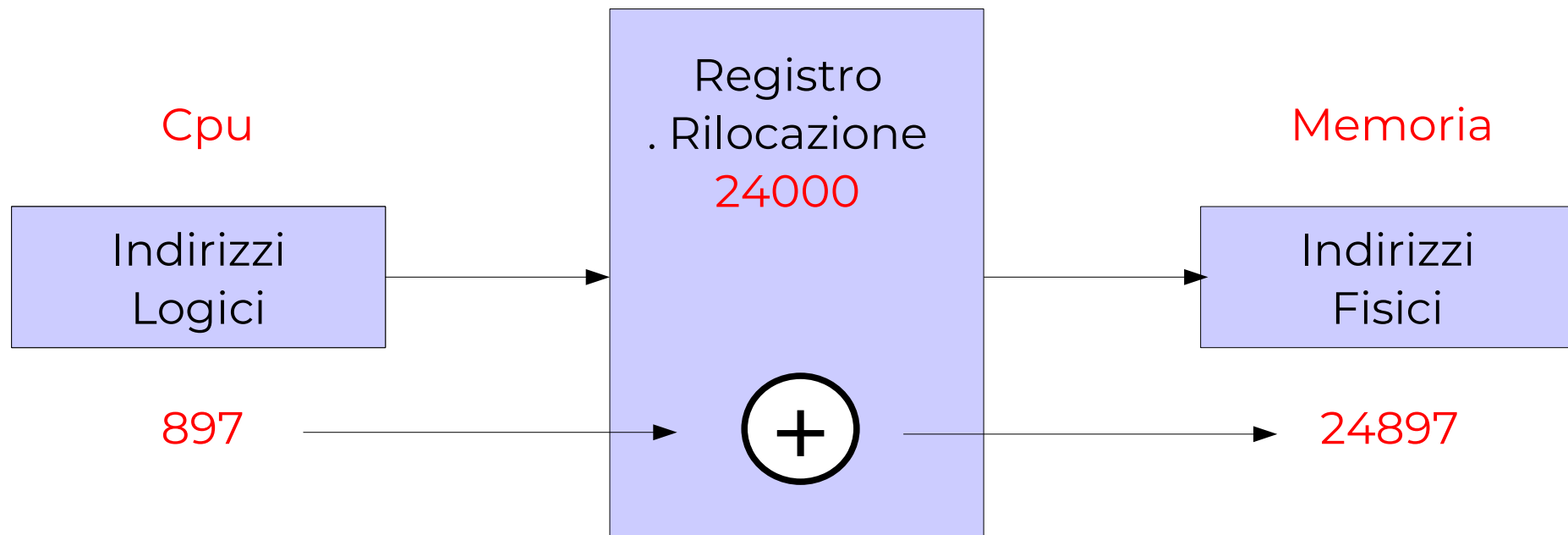
- ♦ Spazio di indirizzamento logico
 - ♦ ogni processo è associato ad uno spazio di indirizzamento logico
 - ♦ gli indirizzi usati in un processo sono indirizzi logici, ovvero riferimenti a questo spazio di indirizzamento
- ♦ Spazio di indirizzamento fisico
 - ♦ ad ogni indirizzo logico corrisponde un indirizzo fisico
 - ♦ la MMU opera come una funzione di traduzione da indirizzi logici a indirizzi fisici



Esempi di MMU - Registro di rilocalazione

• Descrizione

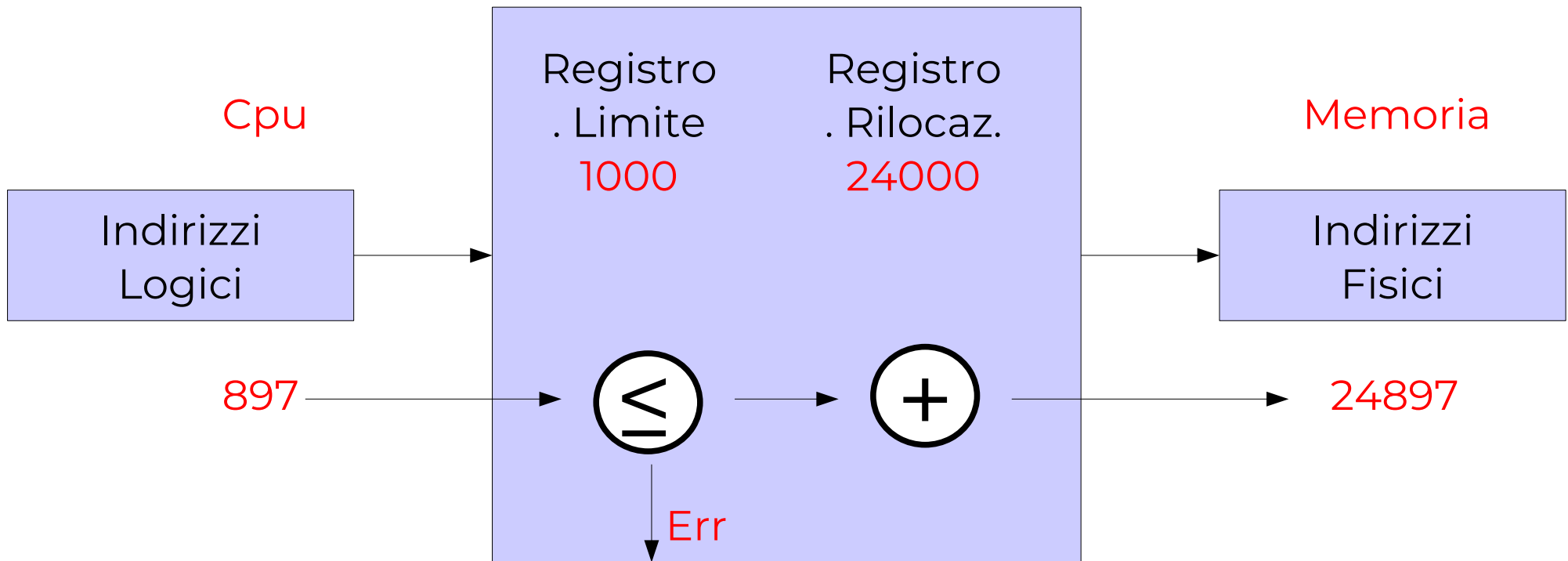
- se il valore del registro di rilocalazione è R , uno spazio logico $0..Max$ viene tradotto in uno spazio fisico $R..R+MAX$
- esempio: nei processori Intel 80x86, esistono 4 registri base per il calcolo degli indirizzi (CS, DS, SS, ES)



Esempi di MMU- Registro di rilocazione e limite

- **Descrizione**

- il registro limite viene utilizzato per implementare meccanismi di protezione della memoria



Prestazioni speciali 1 - Loading dinamico

- ◆ Cosa significa?
 - ◆ consente di poter caricare alcune routine di libreria solo quando vengono richiamate
- ◆ Come viene implementato?
 - ◆ tutte le routine a caricamento dinamico risiedono su un disco (codice rilocabile), quando servono vengono caricate
 - ◆ le routine poco utili (e.g., casi di errore rari...) non vengono caricate in memoria al caricamento dell'applicazione
- ◆ Nota
 - ◆ spetta al programmatore sfruttare questa possibilità
 - ◆ il sistema operativo fornisce semplicemente una libreria che implementa le funzioni di caricamento dinamico

Prestazioni speciali 2 - Linking dinamico

- ♦ **Linking statico**

- ♦ se il linker collega e risolve tutti i riferimenti dei programmi...
- ♦ le routine di libreria vengono copiate in ogni programma che le usa
(e.g. printf in tutti i programmi C)

- ♦ **Linking dinamico**

- ♦ è possibile posticipare il linking delle routine di libreria al momento del primo riferimento durante l'esecuzione
- ♦ consente di avere eseguibili più compatti
- ♦ le librerie vengono implementate come codice reentrant:
 - ♦ esiste una sola istanza della libreria in memoria e tutti i processi eseguono il codice di questa istanza

Prestazioni speciali 2 - Linking dinamico

- ♦ **Vantaggi**

- ♦ risparmio di memoria
- ♦ consente l'aggiornamento automatico delle versioni delle librerie
 - ♦ le librerie aggiornate sono caricate alla successiva attivazione dei programmi

- ♦ **Svantaggi**

- ♦ può causare problemi di "versioning"
 - ♦ occorre aggiornare le versioni solo se non sono incompatibili
 - ♦ cambiamento numero di revisione e non di release

Prestazioni speciali 3 – Loading dinamico tramite Linking dinamico

- ◆ Dlopen: consente di caricare librerie dinamiche a run time
- ◆ E' il metodo per implementare plug-in

Allocazione



- ♦ E' una delle funzioni principali del gestore di memoria
- ♦ Consiste nel reperire ed assegnare uno spazio di memoria fisica
 - ♦ a un programma che viene attivato
 - ♦ oppure per soddisfare ulteriori richieste effettuate dai programmi durante la loro esecuzione

Allocazione: definizioni

- ◆ **Allocazione contigua**

- ◆ tutto lo spazio assegnato ad un processa deve essere formato da celle consecutive

- ◆ **Allocazione non contigua**

- ◆ è possibile assegnare a un processo aree di memorie separate

- ◆ **Nota**

- ◆ la MMU deve essere in grado di gestire la conversione degli indirizzi in modo coerente
- ◆ esempio: la MMU basata su rilocalazione gestisce solo allocazione contigua

Allocazione: statica o dinamica

- ♦ **Statica**

- ♦ un processo deve mantenere la propria area di memoria dal caricamento alla terminazione
- ♦ non è possibile rilocare il processo durante l'esecuzione

- ♦ **Dinamica**

- ♦ durante l'esecuzione, un processo può essere spostato all'interno della memoria

Allocazione a partizioni fisse

- **Descrizione**

- la memoria disponibile (quella non occupata dal s.o.) viene suddivisa in partizioni
- ogni processo viene caricato in una delle partizioni libere che ha dimensione sufficiente a contenerlo

- **Caratteristiche**

- statica e contigua
- vantaggi: molto semplice
- svantaggi: spreco di memoria, grado di parallelismo limitato dal numero di partizioni



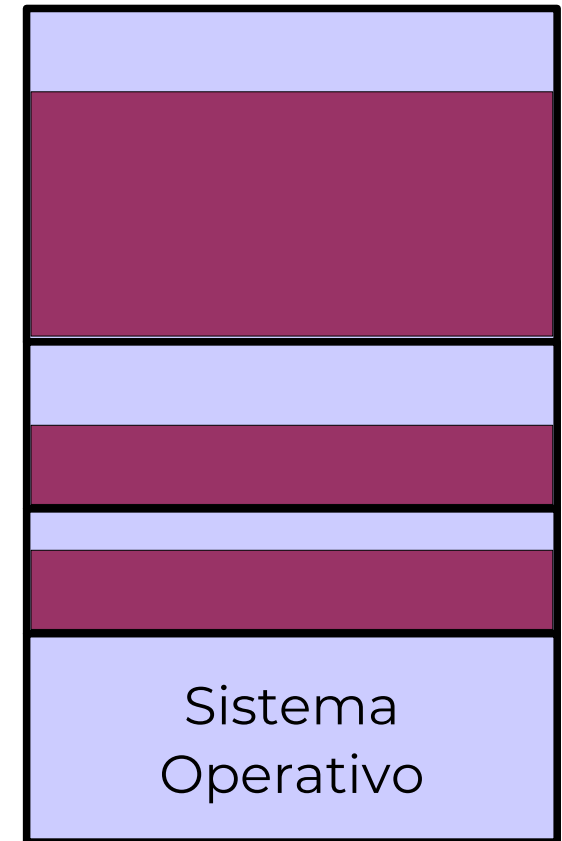
Allocazione a partizioni fisse

- ◆ **Gestione memoria**
 - ◆ è possibile utilizzare una coda per partizione, oppure una coda comune per tutte le partizioni
- ◆ **Sistemi monoprogrammati**
 - ◆ esiste una sola partizione, dove viene caricato un unico programma utente
 - ◆ esempio:
 - ◆ MS-DOS free-DOS
 - ◆ sistemi embedded



Frammentazione interna

- ◆ Nell'allocazione a partizione fisse
 - ◆ se un processo occupa una dimensione inferiore a quella della partizione che lo contiene, lo spazio non utilizzato è sprecato
 - ◆ la presenza di spazio inutilizzato all'interno di un'unità di allocazione si chiama *frammentazione interna*
- ◆ Nota:
 - ◆ il fenomeno della frammentazione interna non è limitata all'allocazione a partizioni fisse, ma è generale a tutti gli approcci in cui è possibile allocare più memoria di quanto richiesto (per motivi di organizzazione)



Allocazione a partizioni dinamiche

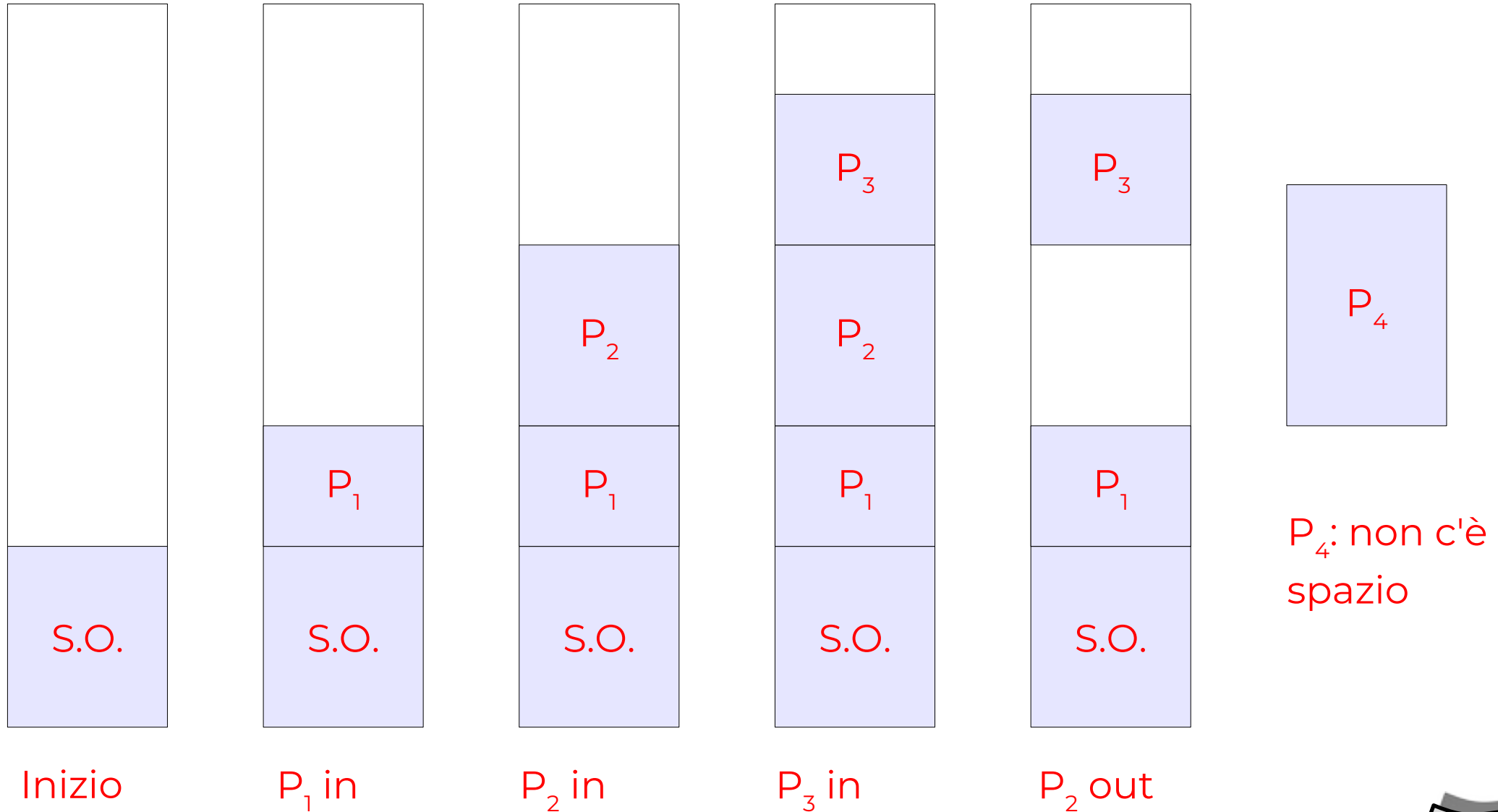
- ◆ **Descrizione**

- ◆ la memoria disponibile (nella quantità richiesta) viene assegnata ai processi che ne fanno richiesta
- ◆ nella memoria possono essere presenti diverse zone inutilizzate
 - ◆ per effetto della terminazione di processi
 - ◆ oppure per non completo utilizzo dell'area disponibile da parte dei processi attivi

- ◆ **Caratteristiche**

- ◆ statica e contigua
- ◆ esistono diverse politiche per la scelta dell'area da utilizzare

Allocazione a partizioni dinamiche



Frammentazione esterna

- ◆ **Problema**

- ◆ dopo un certo numero di allocazioni e deallocazioni di memoria dovute all'attivazione e alla terminazione dei processi lo spazio libero appare suddiviso in piccole aree
- ◆ è il fenomeno della *frammentazione esterna*

- ◆ **Nota**

- ◆ la frammentazione *interna* dipende dall'uso di unità di allocazione di dimensione diversa da quella richiesta
- ◆ la frammentazione *esterna* deriva dal susseguirsi di allocazioni e deallocazioni

Compattazione

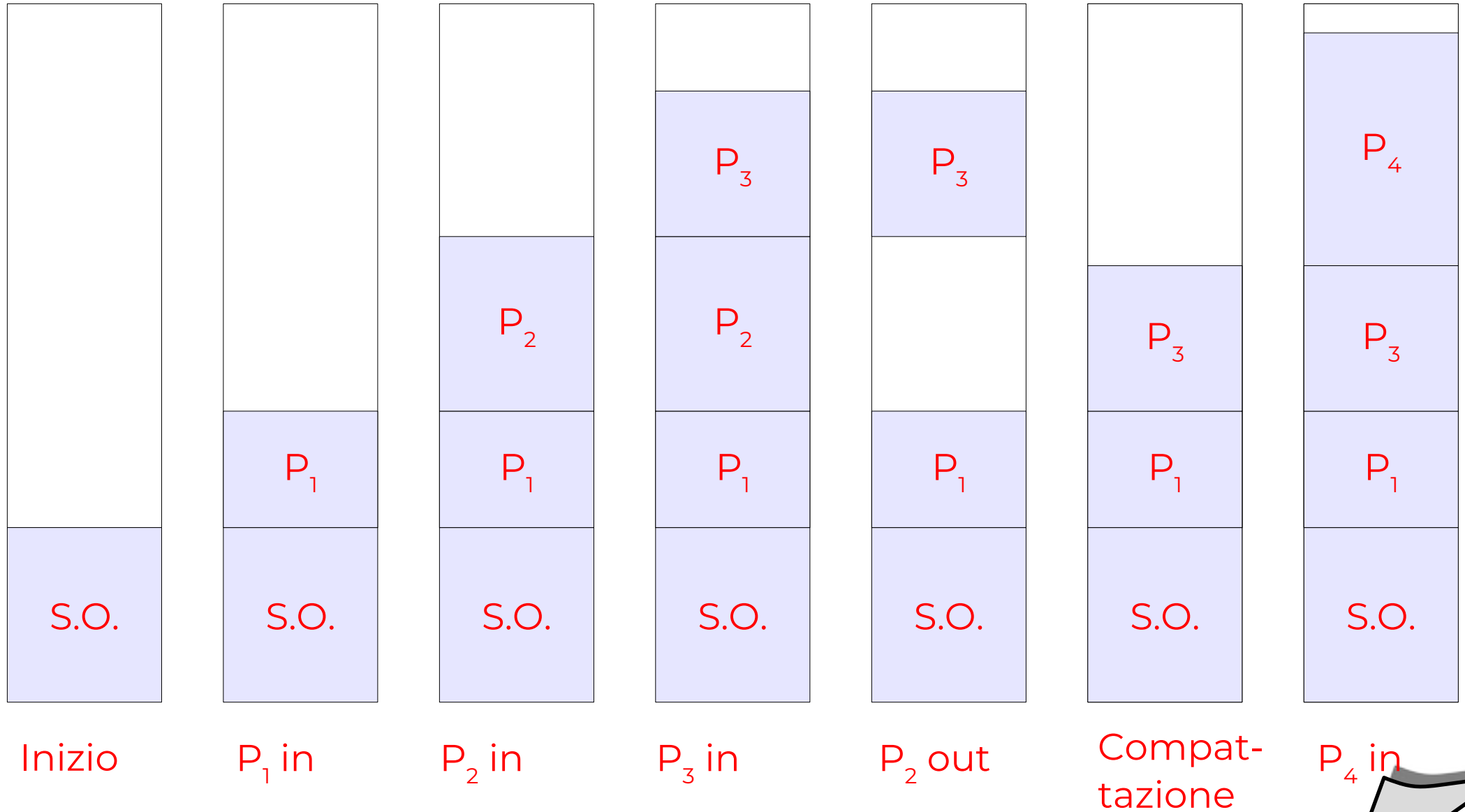
- ◆ **Compattazione**

- ◆ se è possibile rilocare i processi durante la loro esecuzione, è allora possibile procedere alla *compattazione* della memoria
- ◆ compattare la memoria significa spostare in memoria tutti i processi in modo da riunire tutte le aree inutilizzate
- ◆ è un'operazione volta a risolvere il problema della frammentazione esterna

- ◆ **Problemi**

- ◆ è un'operazione molto onerosa
 - ◆ occorre copiare (fisicamente) in memoria grandi quantità di dati
- ◆ non può essere utilizzata in sistemi interattivi
 - ◆ i processi devono essere fermi durante la compattazione

Compattazione



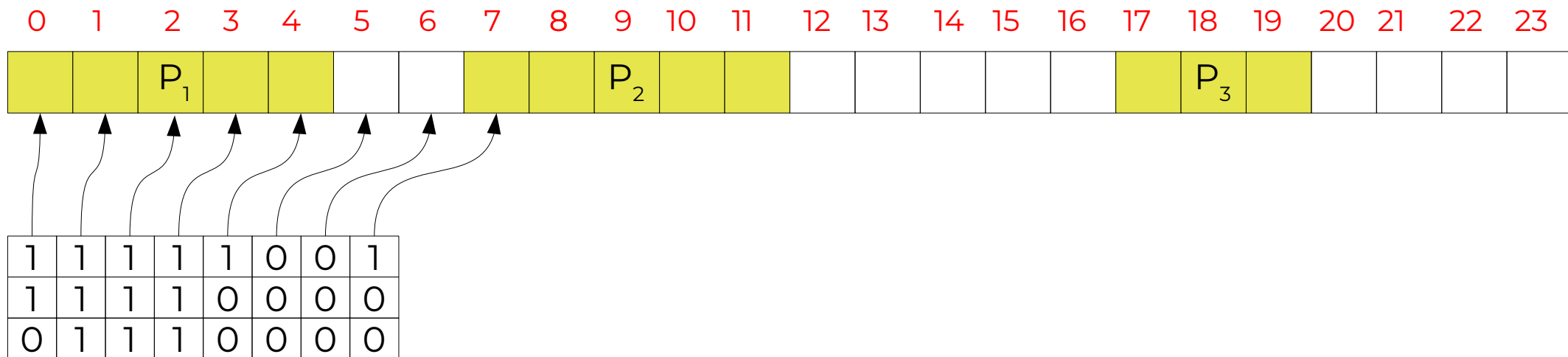
Allocazione dinamica - Strutture dati

- ◆ Quando la memoria è assegnata dinamicamente
 - ◆ abbiamo bisogno di una struttura dati per mantenere informazioni sulle zone libere e sulle zone occupate
- ◆ Strutture dati possibili
 - ◆ mappe di bit
 - ◆ liste con puntatori
 - ◆ ...

Allocazione Dinamica - Mappa di bit

♦ Mappa di bit

- ♦ la memoria viene suddivisa in unità di allocazione
- ♦ ad ogni unità di allocazione corrisponde un bit in una bitmap
- ♦ le unità libere sono associate ad un bit di valore 0, le unità occupate sono associate ad un bit di valore 1



Allocazione Dinamica - Mappa di bit

- ♦ **Note**

- ♦ la dimensione dell'unità di allocazione è un parametro importante dell'algoritmo
- ♦ trade-off fra dimensione della bitmap e frammentazione interna

- ♦ **Vantaggi**

- ♦ la struttura dati ha una dimensione fissa e calcolabile a priori

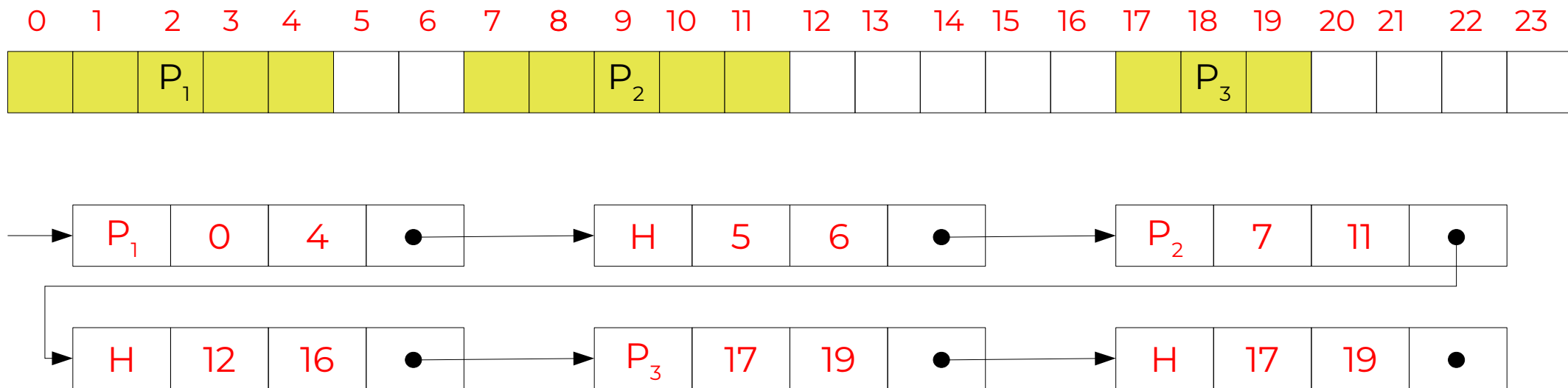
- ♦ **Svantaggi**

- ♦ per individuare uno spazio di memoria di dimensione **k** unità, è necessario cercare una sequenza di **k** bit 0 consecutivi
- ♦ in generale, tale operazione è **$O(m)$** , dove **m** rappresenta il numero di unità di allocazione

Allocazione dinamica - Lista con puntatori

- Liste di puntatori

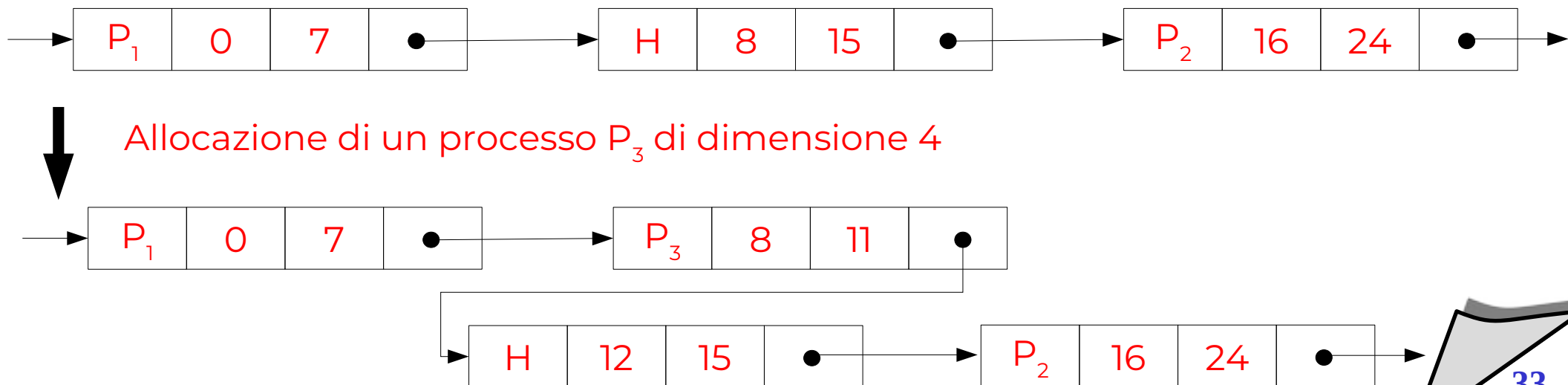
- si mantiene una lista dei blocchi allocati e liberi di memoria
- ogni elemento della lista specifica
 - se si tratta di un processo (P) o di un blocco libero (hole, H)
 - la dimensione (inizio/fine) del segmento



Allocazione dinamica - Lista con puntatori

- Allocazione di memoria

- un blocco libero viene selezionato (vedi slide successive)
- viene suddiviso in due parti:
 - un blocco processo della dimensione desiderata
 - un blocco libero con quanto rimane del blocco iniziale
- se la dimensione del processo è uguale a quella del blocco scelto, si crea solo un nuovo blocco processo

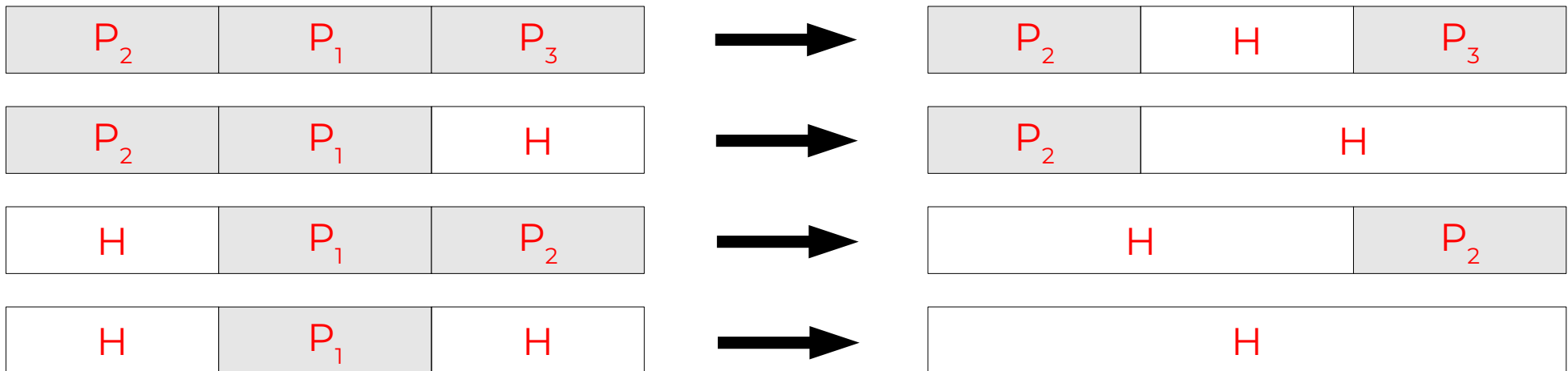


Allocazione dinamica - Lista puntatori

- Deallocazione memoria

- a seconda dei blocchi vicini, lo spazio liberato può creare un nuovo blocco libero, oppure essere accorpato ai blocchi vicini
- l'operazione può essere fatta in tempo $O(1)$

Rimozione P_1 , quattro casi possibili:



Allocazione dinamica - Selezione blocco libero

- ♦ L'operazione di selezione di un blocco libero è concettualmente indipendente dalla struttura dati
- ♦ **First Fit**
 - ♦ scorre la lista dei blocchi liberi fino a quando non trova il primo segmento vuoto grande abbastanza da contenere il processo
- ♦ **Next Fit**
 - ♦ come First Fit, ma invece di ripartire sempre dall'inizio, parte dal punto dove si era fermato all'ultima allocazione
- ♦ **Commenti**
 - ♦ Next Fit è stato progettato per evitare di frammentare continuamente l'inizio della memoria
 - ♦ ma sorprendentemente, ha performance peggiori di First Fit

Allocazione dinamica - Selezione blocco libero

- ◆ **Best Fit**

- ◆ seleziona il più piccolo fra i blocchi liberi presenti in memoria

- ◆ **Commenti**

- ◆ più lento di First Fit, in quanto richiede di esaminare tutti i blocchi liberi presenti in memoria
- ◆ genera più frammentazione di First Fit, in quanto tende a riempire la memoria di blocchi liberi troppo piccoli

- ◆ **Worst fit**

- ◆ seleziona il più grande fra i blocchi liberi presenti in memoria

- ◆ **Commenti**

- ◆ proposto per evitare i problemi di frammentazione di First/Best Fit
- ◆ rende difficile l'allocazione di processi di grosse dimensioni

Allocazione dinamica - Strutture dati (ancora)

- ♦ Trade-off tra costi di allocazione e deallocazione
 - ♦ nella struttura proposta in precedenza, il costo della deallocazione è $O(1)$
 - ♦ è possibile ottimizzare il costo di allocazione
 - ♦ mantenendo una lista di blocchi liberi separata
 - ♦ eventualmente, ordinando tale lista per dimensione
- ♦ Dove mantenere queste informazioni
 - ♦ per i blocchi occupati
 - ♦ ad esempio, nella tabella dei processi
 - ♦ per i blocchi liberi
 - ♦ nei blocchi stessi!
 - ♦ è richiesta un unità minima di allocazione

Paginazione

- ◆ **Problema**

- ◆ i meccanismi visti (partizioni fisse, partizioni dinamiche) non sono efficienti nell'uso della memoria
 - ◆ frammentazione interna
 - ◆ frammentazione esterna

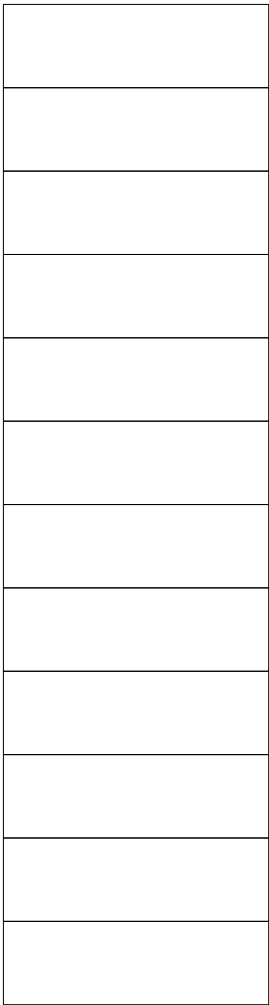
- ◆ **Paginazione**

- ◆ è l'approccio contemporaneo
 - ◆ riduce il fenomeno di frammentazione interna
 - ◆ minimizza (elimina) il fenomeno della frammentazione esterna
- ◆ attenzione però: necessita di hardware adeguato

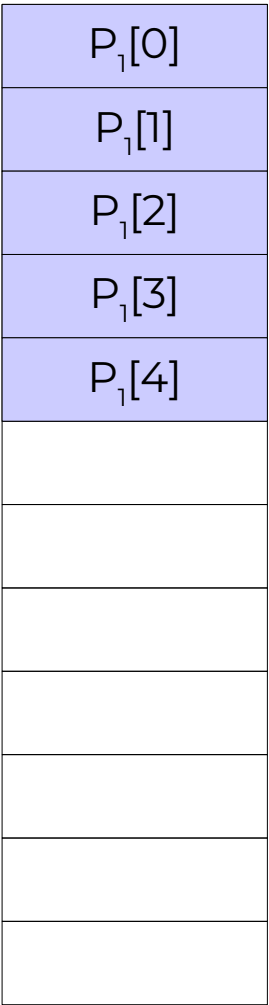
Paginazione

- ♦ Lo spazio di indirizzamento logico di un processo
 - ♦ viene suddiviso in un insieme di blocchi di dimensione fissa chiamati *pagine*
- ♦ La memoria fisica
 - ♦ viene suddivisa in un insieme di blocchi della stessa dimensione delle pagine, chiamati *frame*
- ♦ Quando un processo viene allocato in memoria:
 - ♦ vengono reperiti ovunque in memoria un numero sufficiente di frame per contenere le pagine del processo

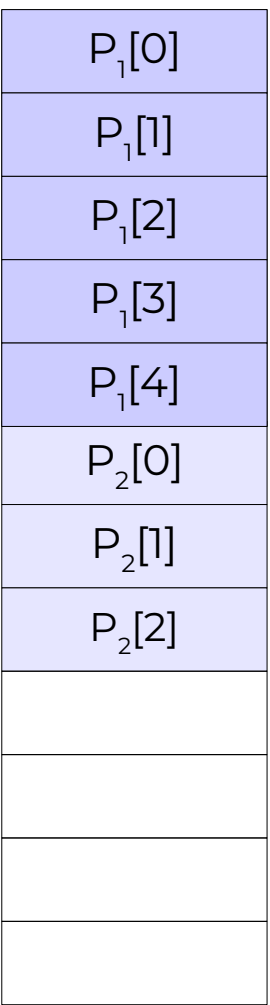
Paginazione - Esempio



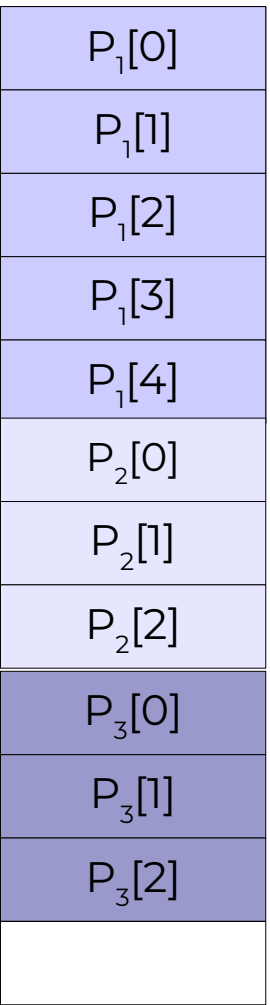
Tutto libero



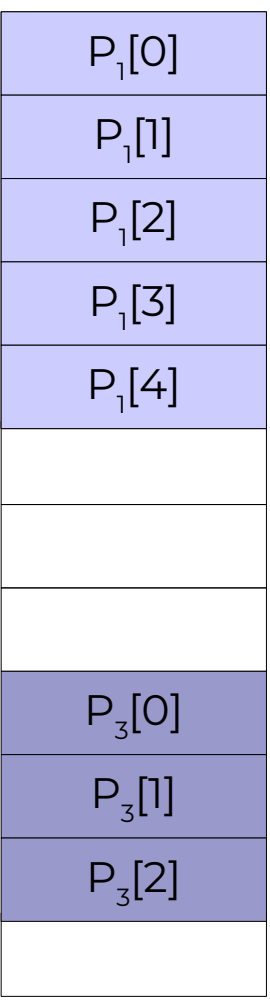
P_1 in



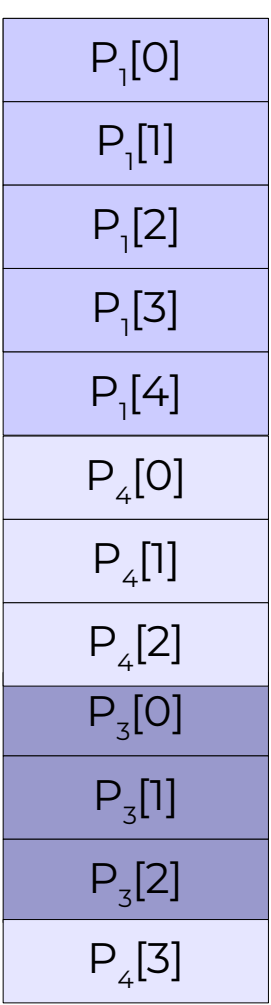
P_2 in



P_3 in

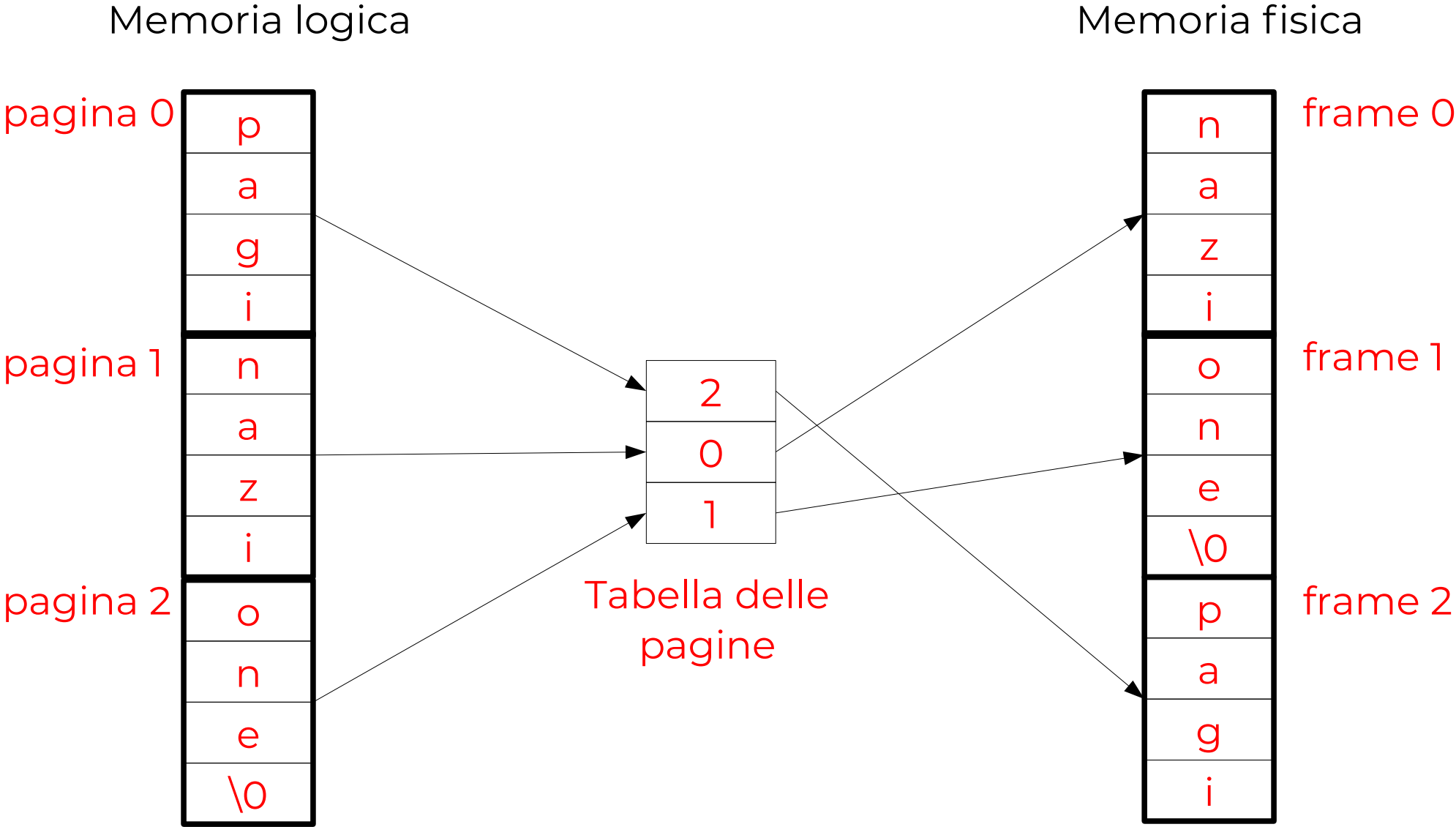


P_2 out



P_4 in

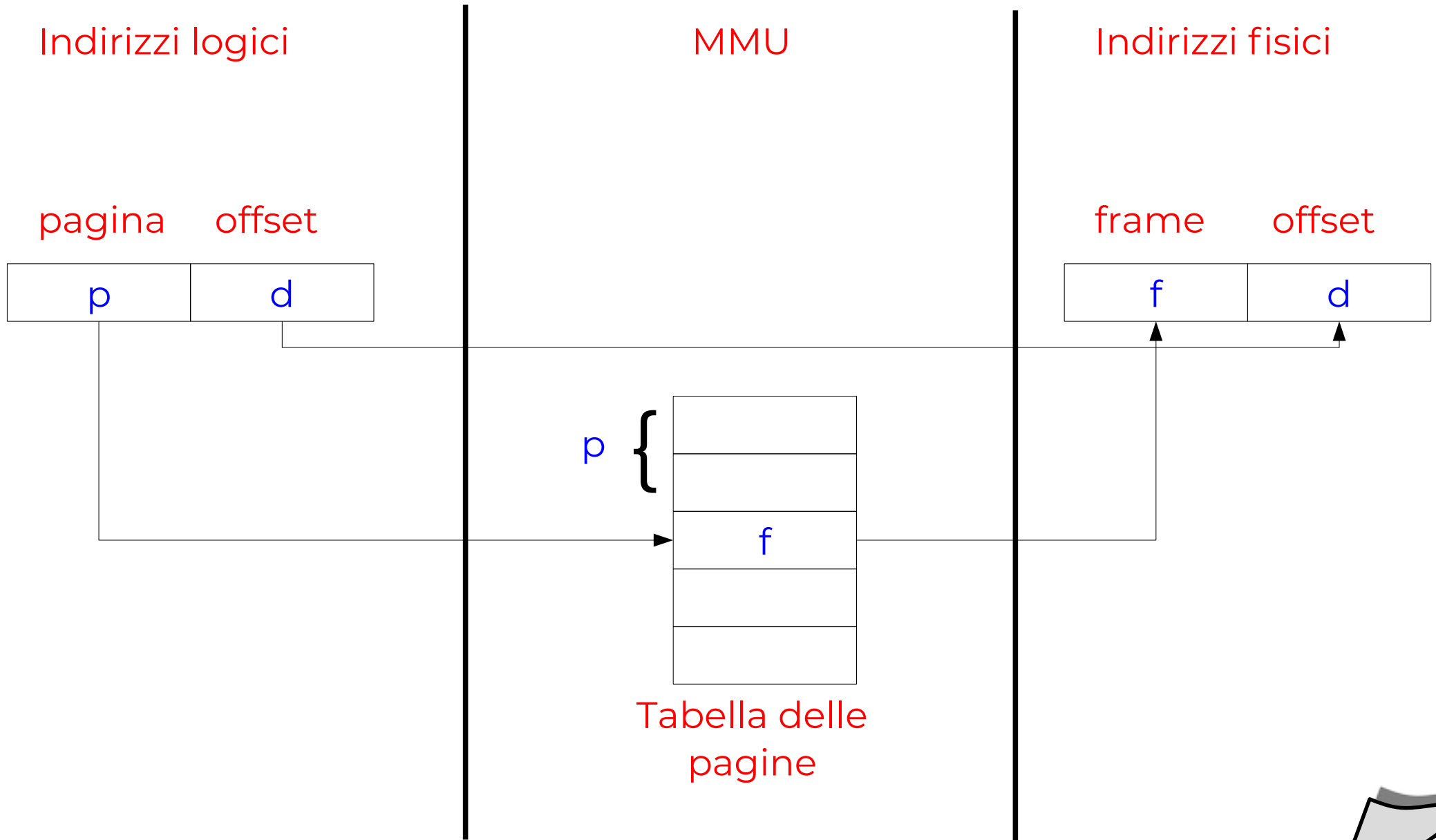
Paginazione - Esempio



Dimensione delle pagine

- ◆ Come scegliere la dimensione delle pagine?
 - ◆ la dimensione delle pagine deve essere una potenza di due, per semplificare la trasformazione da indirizzi logici a indirizzi fisici
 - ◆ la scelta della dimensione deriva da un trade-off
 - ◆ con pagine troppo piccole, la tabella delle pagine cresce di dimensioni
 - ◆ con pagine troppo grandi, lo spazio di memoria perso per frammentazione interna può essere considerevole
 - ◆ valori tipici: 1KB, 2KB, 4KB

Supporto hardware (MMU) per paginazione



Implementazione della page table

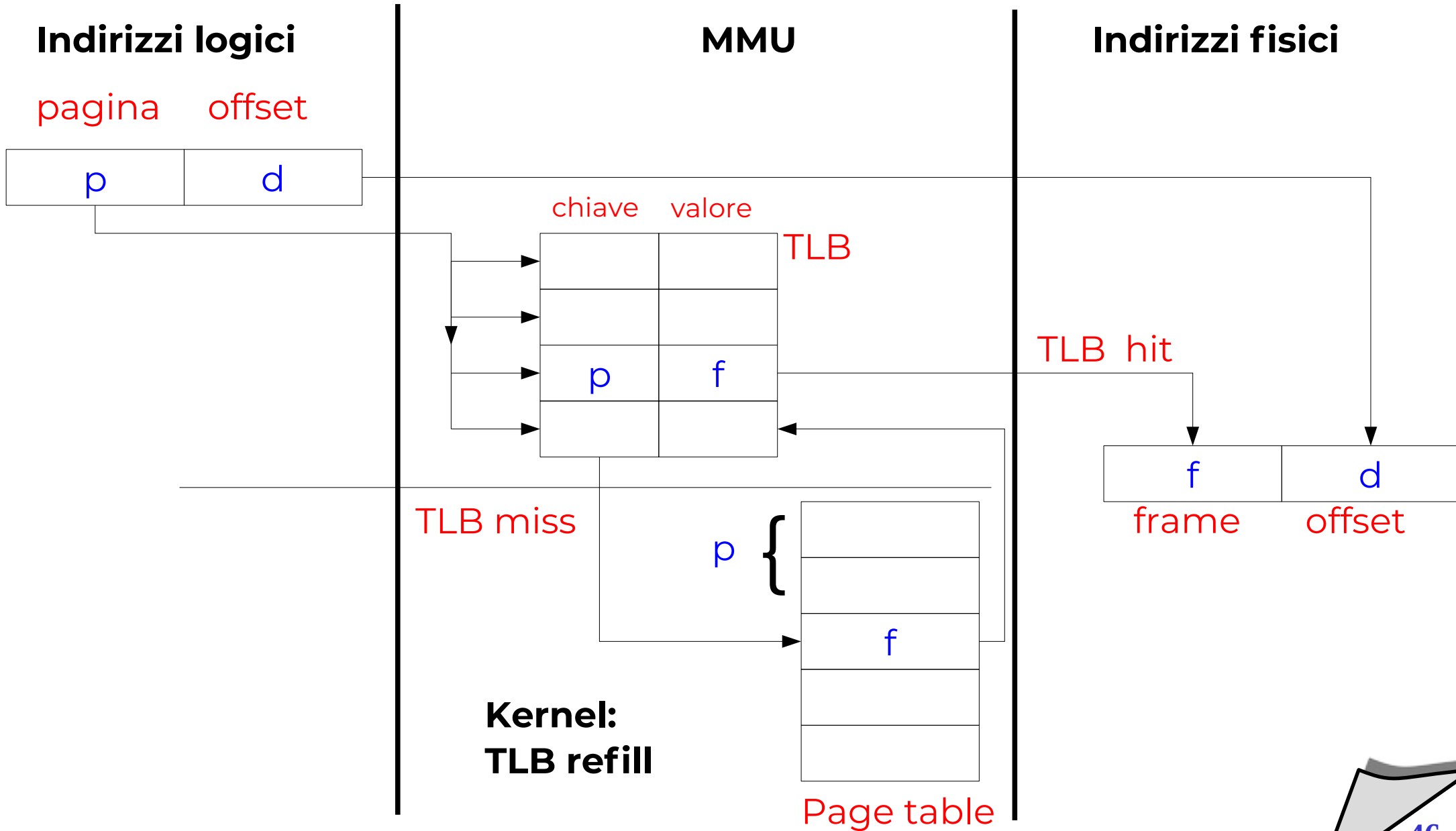
- ◆ Dove mettere la tabella delle pagine?
- ◆ Soluzione 1: registri dedicati
 - ◆ la tabella può essere contenuta in un insieme di registri ad alta velocità all'interno del modulo MMU (o della CPU)
 - ◆ problema: troppo costoso
 - ◆ esempio:
 - ◆ pagine di 4K, processore a 32 bit
 - ◆ numero di pagine nella page table: 1M (1.048.576)
- ◆ Soluzione 2: totalmente in memoria
 - ◆ problema: il numero di accessi in memoria verrebbe raddoppiato; ad ogni riferimento, bisognerebbe prima accedere alla tabella delle pagine, poi al dato

Translation lookaside buffer (TLB)

♦ Descrizione

- ♦ un TLB è costituito da un insieme di registri associativi ad alta velocità
- ♦ ogni registro è suddiviso in due parti, una chiave e un valore
- ♦ operazione di lookup
 - ♦ viene richiesta la ricerca di una chiave
 - ♦ la chiave viene confrontata simultaneamente con tutte le chiavi presenti nel buffer
 - ♦ se la chiave è presente (TLB hit), si ritorna il valore corrispondente
 - ♦ se la chiave non è presente (TLB miss), si utilizza la tabella in memoria

Translation lookaside buffer (TLB)



Translation lookaside buffer (TLB)

- ◆ Note

- ◆ la TLB agisce come memoria cache per le tabelle delle pagine
- ◆ il meccanismo della TLB (come tutti i meccanismi di caching) si basa sul principio di località
- ◆ l'hardware per la TLB è costoso
- ◆ dimensioni dell'ordine 8-2048 registri

Segmentazione



- ◆ In un sistema con segmentazione
 - ◆ la memoria associata ad un processo è suddivisa in aree differenti dal punto di vista funzionale
- ◆ Esempio
 - ◆ aree text:
 - ◆ contengono il codice eseguibile
 - ◆ sono normalmente in sola lettura (solo i virus cambiano il codice)
 - ◆ possono essere condivise tra più processi (codice reentrant)
 - ◆ aree dati
 - ◆ possono essere condivise oppure no
 - ◆ area stack
 - ◆ read/write, non può assolutamente essere condivisa

Segmentazione

- ◆ In un sistema basato su segmentazione
 - ◆ uno spazio di indirizzamento logico è dato da un insieme di segmenti
 - ◆ un segmento è un'area di memoria (logicamente continua) contenente elementi tra loro affini
 - ◆ ogni segmento è caratterizzato da un *nome* (normalmente un indice) e da una *lunghezza*
 - ◆ ogni riferimento di memoria è dato da una coppia *<nome segmento, offset>*
- ◆ Spetta al programmatore o al compilatore la suddivisione di un programma in segmenti

Segmentazione vs Paginazione

♦ Paginazione

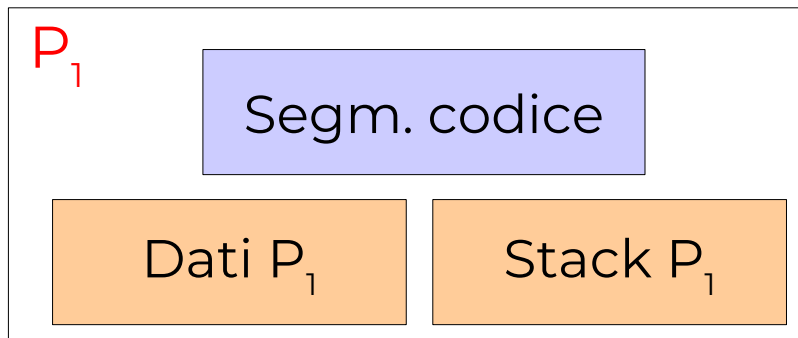
- ♦ la divisione in pagine è automatica.
- ♦ le pagine hanno dimensione fissa
- ♦ le pagine possono contenere informazioni disomogenee (ad es. sia codice sia dati)
- ♦ una pagina ha un indirizzo
- ♦ dimensione tipica della pagina: 1-4 KB

♦ Segmentazione

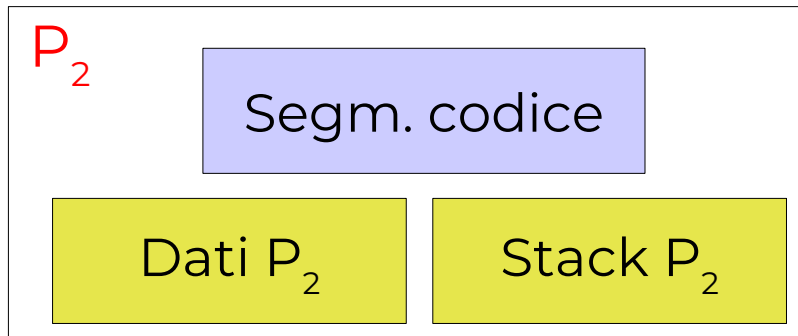
- ♦ la divisione in segmenti spetta al programmatore.
- ♦ i segmenti hanno dimensione variabile
- ♦ un segmento contiene informazioni omogenee per tipo di accesso e permessi di condivisione
- ♦ un segmento ha un nome.
- ♦ dimensione tipica di un segmento: 64KB – molti MB

Segmentazione e condivisione

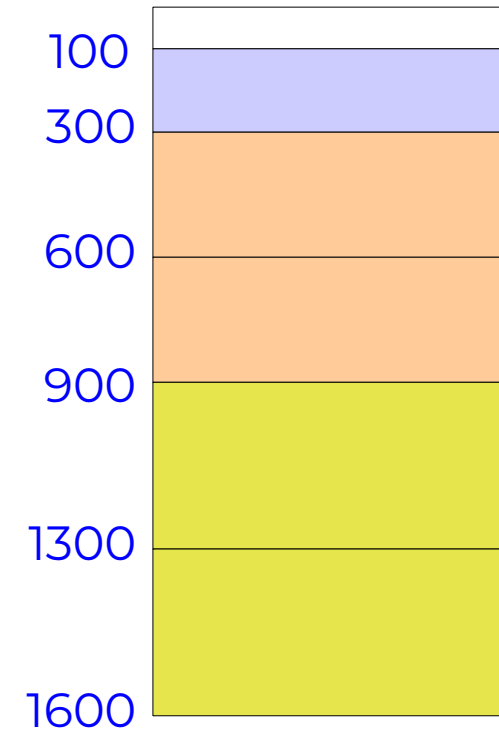
- La segmentazione consente la condivisione di codice e dati
- Esempio: editor condiviso



Code	100	200
Data	300	300
Stack	600	300



Code	100	200
Data	900	400
Stack	1300	400



Segmentazione e frammentazione

- ◆ Problema
 - ◆ allocare segmenti di dimensione variabile è del tutto equivalente al problema di allocare in modo contiguo la memoria dei processi
 - ◆ è possibile utilizzare
 - ◆ tecniche di allocazione dinamica (e.g., First Fit)
 - ◆ compattazione
- ◆ ma così torniamo ai problemi precedenti!

Segmentazione e paginazione

- ◆ **Segmentazione + paginazione**
 - ◆ è possibile utilizzare il metodo della paginazione combinato al metodo della segmentazione
 - ◆ ogni segmento viene suddiviso in pagine che vengono allocate in frame liberi della memoria (non necessariamente contigui)
- ◆ **Requisiti hardware**
 - ◆ la MMU deve avere sia il supporto per la segmentazione sia il supporto per la paginazione
- ◆ **Benefici**
 - ◆ sia quelli della segmentazione (condivisione, protezione)
 - ◆ sia quelli della paginazione (no frammentazione esterna)

Memoria virtuale

- ◆ **Definizione**

- ◆ è la tecnica che permette l'esecuzione di processi che non sono completamente in memoria

- ◆ **Considerazioni**

- ◆ permette di eseguire in concorrenza processi che nel loro complesso (o anche singolarmente) hanno necessità di memoria maggiore di quella disponibile
- ◆ la memoria virtuale può diminuire le prestazioni di un sistema se implementata (e usata) nel modo sbagliato

Memoria virtuale



- ♦ **Requisiti di un'architettura di Von Neumann**
 - ♦ le istruzioni da eseguire e i dati su cui operano devono essere in memoria
- ♦ **ma...**
 - ♦ non è necessario che l'intero spazio di indirizzamento logico di un processo sia in memoria
 - ♦ i processi non utilizzano tutto il loro spazio di indirizzamento *contemporaneamente*
 - ♦ routine di gestione errore
 - ♦ strutture dati allocate con dimensioni massime ma utilizzate solo parzialmente
 - ♦ passi di avanzamento di un programma (e.g. compilatore a due fasi)

Memoria virtuale

- ◆ Implementazione

- ◆ ogni processo ha accesso ad uno *spazio di indirizzamento virtuale* che può essere più grande di quello fisico
- ◆ gli indirizzi virtuali (logici)
 - ◆ possono essere mappati su indirizzi fisici della memoria principale
 - ◆ oppure, possono essere mappati su memoria secondaria
- ◆ in caso di accesso ad indirizzi virtuali mappati in memoria secondaria:
 - ◆ i dati associati vengono trasferiti in memoria principale
 - ◆ se la memoria è piena, si sposta in memoria secondaria i dati contenuti in memoria principale che sono considerati meno utili

Memoria virtuale

- ♦ **Paginazione a richiesta (*demand paging*)**
 - ♦ si utilizza la tecnica della paginazione, ammettendo però che alcune pagine possano essere in memoria secondaria
- ♦ **Nella tabella delle pagine**
 - ♦ si utilizza un bit (*v*, per valid) che indica se la pagina è presente in memoria centrale oppure no
- ♦ **Quando un processo tenta di accedere ad un pagina non in memoria**
 - ♦ il processore genera un trap (*page fault*) (o diventa un caso particolare del TLB refill)
 - ♦ un componente del s.o. (*pager*) si occupa di caricare la pagina mancante in memoria, e di aggiornare di conseguenza la tabella delle pagine

Memoria virtuale - Esempio

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

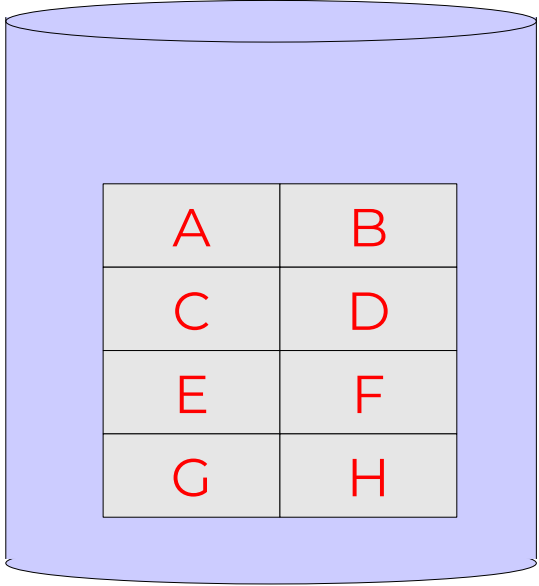
Memoria Logica

frame	bit valid/ invalid	
0	4	v
1		i
2	6	v
3		i
4		i
5	1	v
6		i
7		i

Page Table

0	
1	F
2	
3	
4	A
5	
6	C
7	
8	
9	
10	
11	

Memoria principale



Memoria secondaria

Pager/swapper

- ◆ **Swap**

- ◆ con questo termine si intende l'azione di copiare l'intera area di memoria usata da un processo
 - ◆ dalla memoria secondaria alla memoria principale (*swap-in*)
 - ◆ dalla memoria principale alla memoria secondaria (*swap-out*)
- ◆ era una tecnica utilizzata nel passato quando demand paging non esisteva

- ◆ **Paginazione su richiesta**

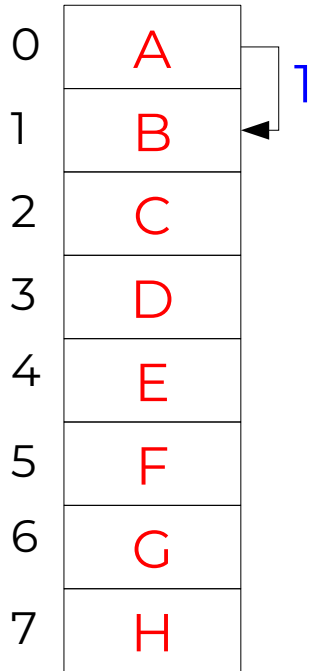
- ◆ può essere vista come una tecnica di swap di tipo lazy (pigro)
- ◆ viene caricato solo ciò che serve

Pager/swapper

- ◆ Per questo motivo
 - ◆ alcuni sistemi operativi indicano il pager con il nome di *swapper*
 - ◆ è da considerarsi una terminologia obsoleta
- ◆ Nota
 - ◆ però utilizziamo il termine *swap area* per indicare l'area del disco utilizzata per ospitare le pagine in memoria secondaria

Gestione dei page fault

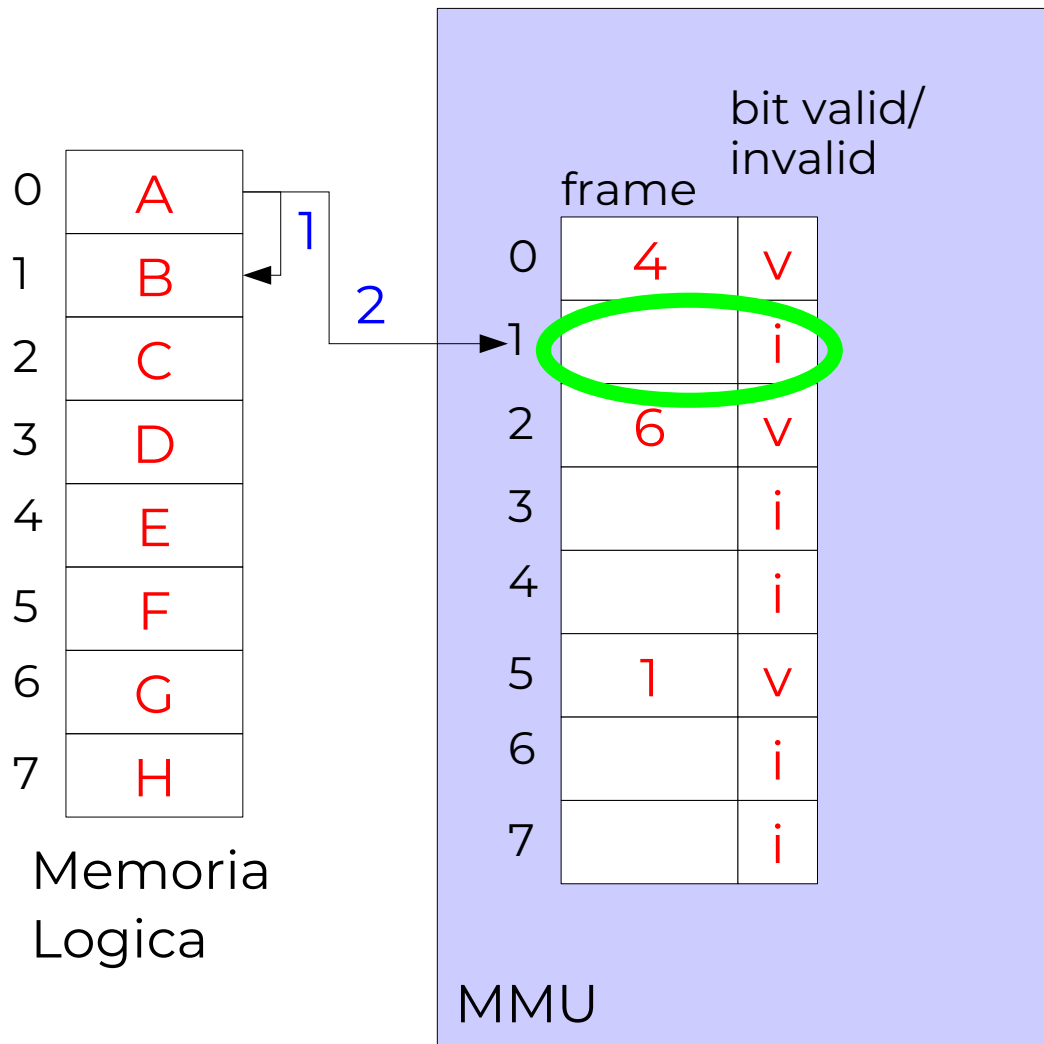
- Supponiamo che il codice in pagina 0 faccia riferimento alla pagina 1



Memoria
Logica

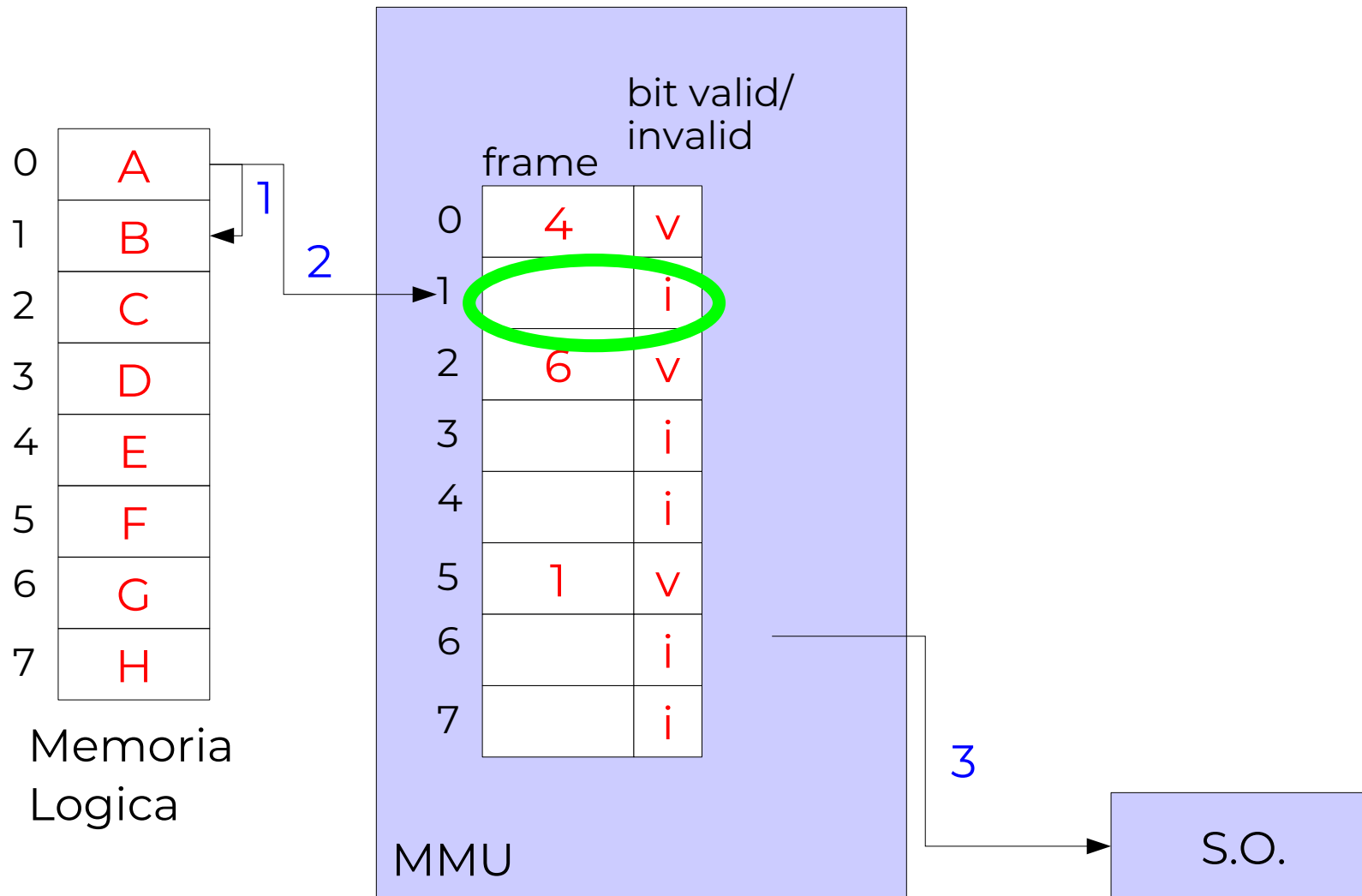
Gestione dei page fault

- La MMU scopre che la pagina 1 non è in memoria principale



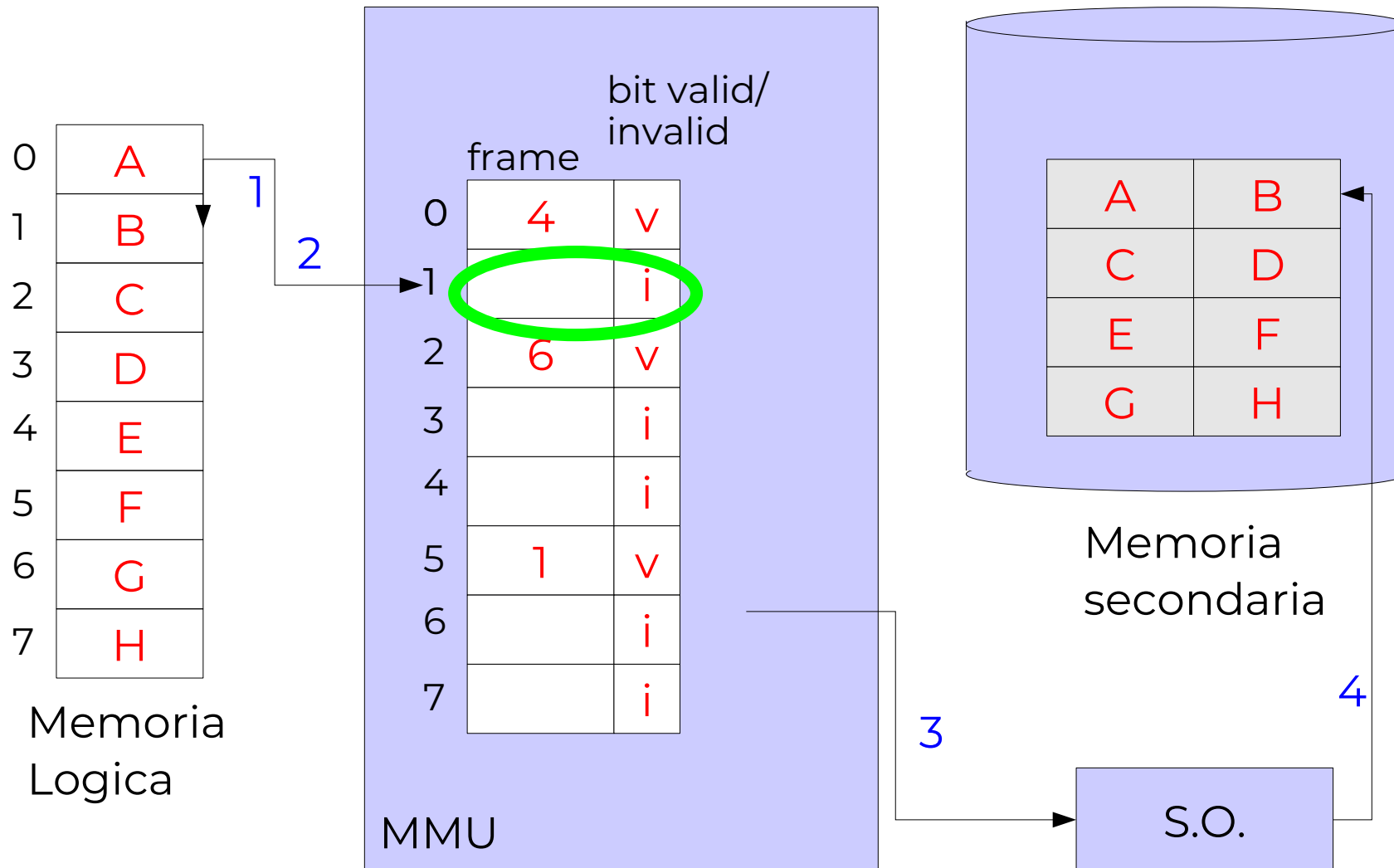
Gestione dei page fault

- Viene generato un trap "page fault", che viene catturato dal s.o.



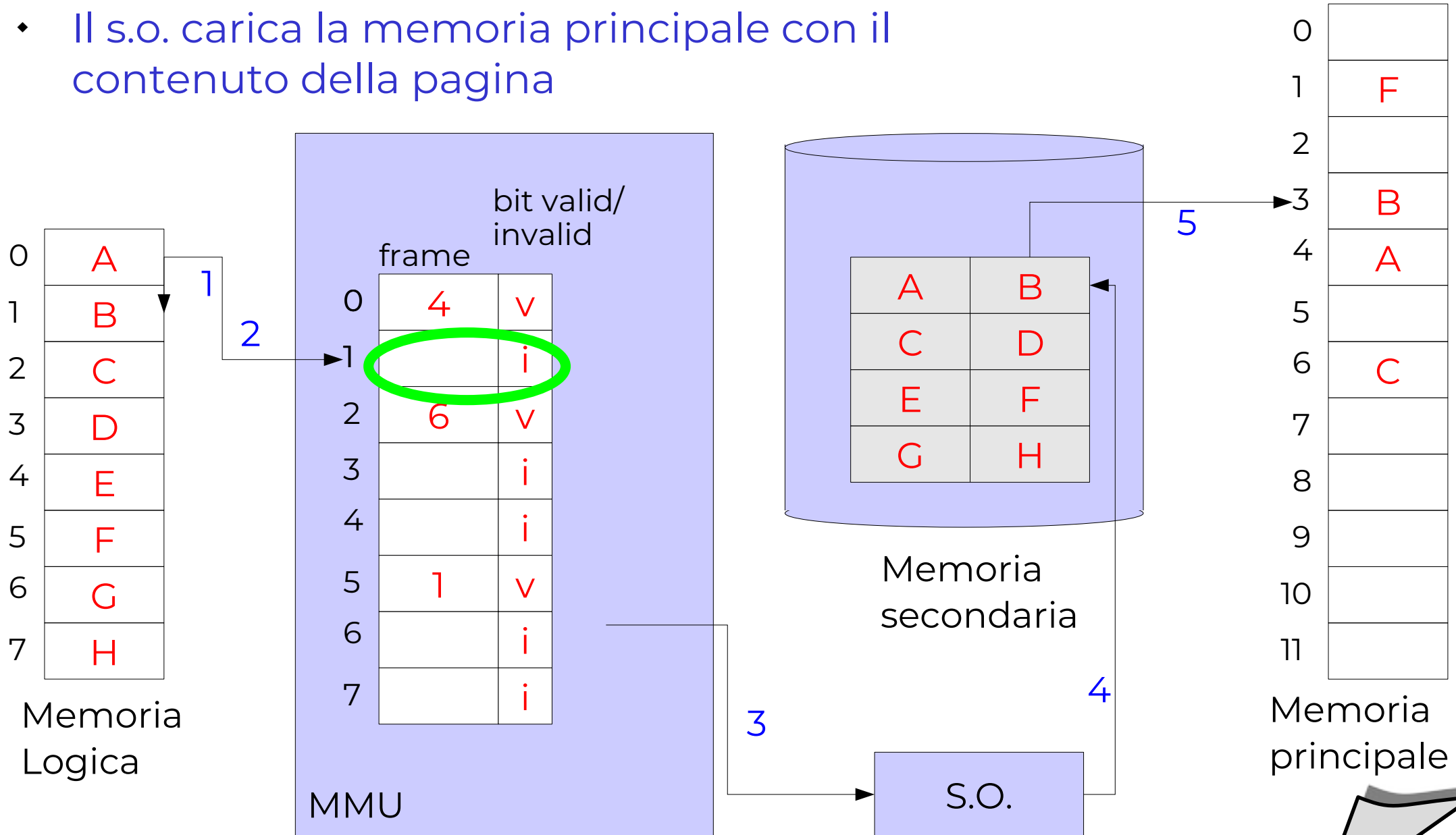
Gestione dei page fault

- Il s.o. cerca in memoria secondaria la pagina da caricare



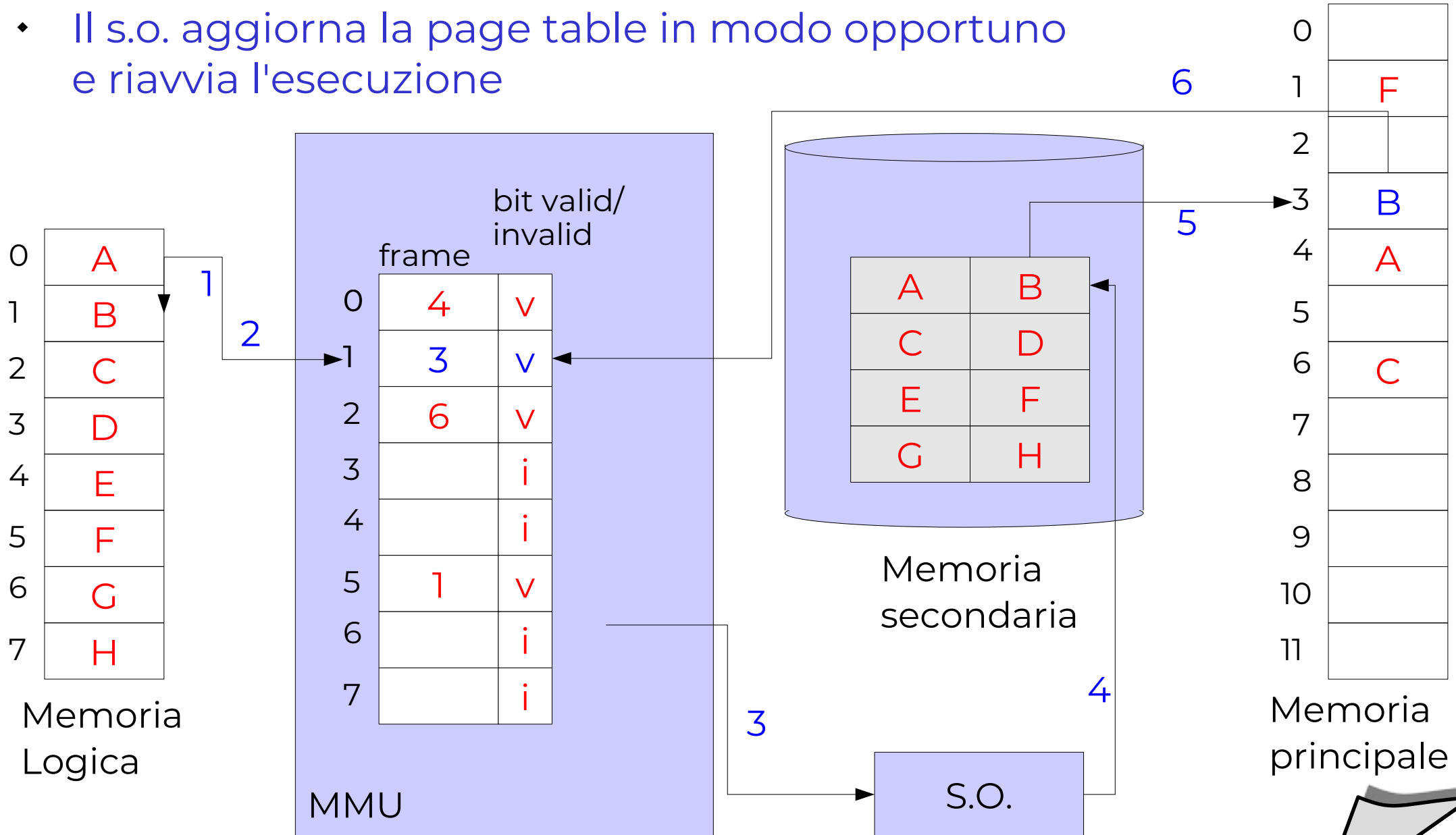
Gestione dei page fault

- Il s.o. carica la memoria principale con il contenuto della pagina



Gestione dei page fault

- Il s.o. aggiorna la page table in modo opportuno e riavvia l'esecuzione



Gestione dei page fault

- ♦ Cosa succede in mancanza di frame liberi?
 - ♦ occorre "liberarne" uno
 - ♦ la pagina vittima deve essere la meno "utile"
- ♦ Algoritmi di sostituzione o rimpiazzamento
 - ♦ la classe di algoritmi utilizzati per selezionare la pagina da sostituire

Algoritmo del meccanismo di demand paging

- Individua la pagina in memoria secondaria
- Individua un frame libero
- Se non esiste un frame libero
 - richiama algoritmo di rimpiazzamento
 - aggiorna la tabella delle pagine (invalida pagina "vittima")
 - se la pagina "vittima" è stata variata, scrive la pagina sul disco
 - aggiorna la tabella dei frame (frame libero)
- Aggiorna la tabella dei frame (frame occupato)
- Leggi la pagina da disco (quella che ha provocato il fault)
- Aggiorna la tabella delle pagine (caricato il TLB)
- Riattiva il processo

Algoritmi di rimpiazzamento

- ♦ Obiettivi
 - ♦ minimizzare il numero di page fault
- ♦ Valutazione
 - ♦ gli algoritmi vengono valutati esaminando come si comportano quando applicati ad una *stringa di riferimenti* in memoria
- ♦ Stringhe di riferimenti
 - ♦ possono essere generate esaminando il funzionamento di programmi reali o con un generatore di numeri random

Algoritmi di rimpiazzamento

- ♦ **Nota:**

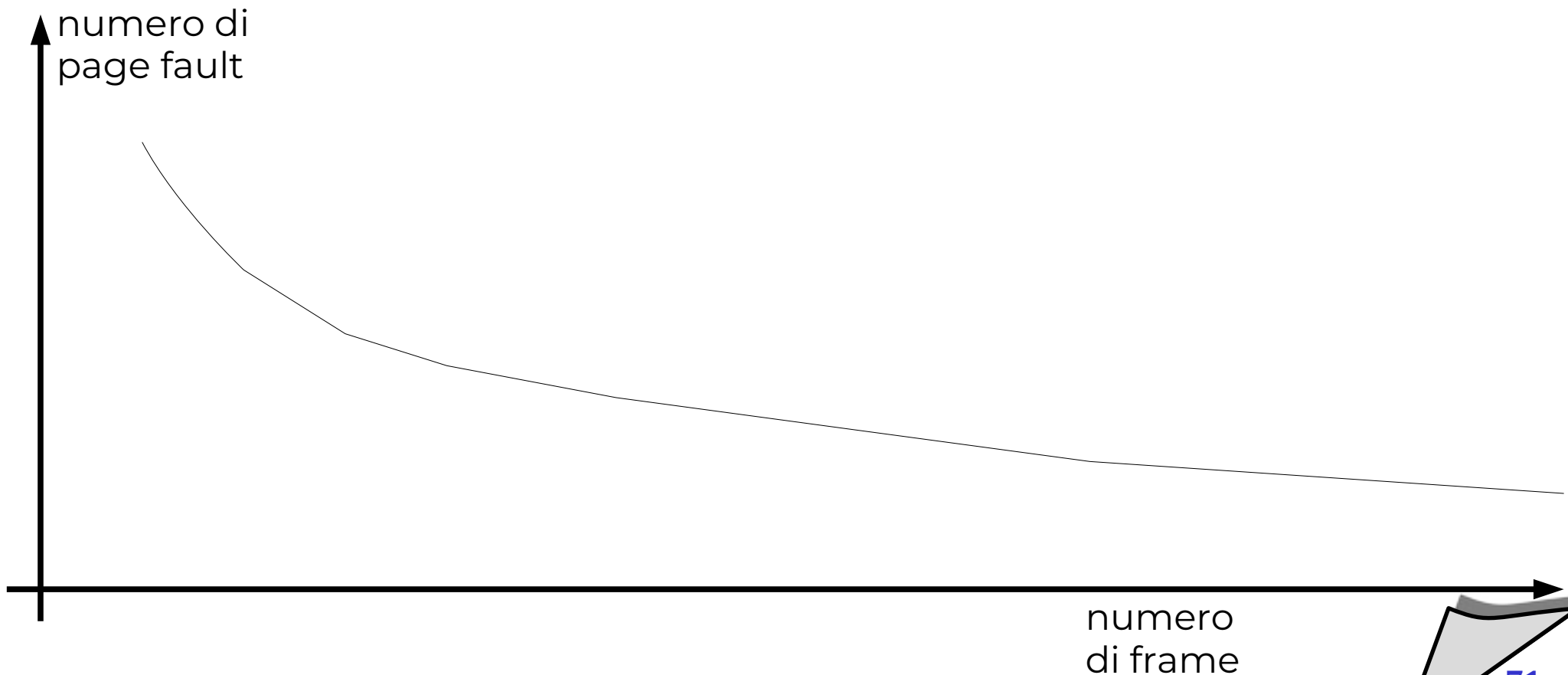
- ♦ la stringa di riferimenti può essere limitata ai numeri di pagina, in quanto non siamo interessati agli offset

- ♦ **Esempio**

- ♦ stringa di riferimento completa (in esadecimale):
 - ♦ 71,0a,13,25,0a,3f,0c,4f,21,30,00,31,21,1a,2b,03,1a,77,11
- ♦ stringa di riferimento delle pagine (in esadecimale, con pagine di 16 byte)
 - ♦ 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,1

Algoritmi di rimpiazzamento

- ♦ Andamento dei page fault in funzione del numero di frame
 - ♦ ci si aspetta un grafo monotono decrescente...
 - ♦ ma non sempre è così



Algoritmo FIFO

- ◆ **Descrizione**

- ◆ quando c'è necessità di liberare un frame viene individuato come “vittima” il frame che per primo fu caricato in memoria

- ◆ **Vantaggi**

- ◆ semplice, non richiede particolari supporti hardware

- ◆ **Svantaggi**

- ◆ vengono talvolta scaricate pagine che sono sempre utilizzate

Algoritmo FIFO - Esempio 1

- Caratteristiche

- numero di frame in memoria: 3
- numero di page fault: 15 (su 20 accessi in memoria)

tempo 1 20

stringa riferim.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

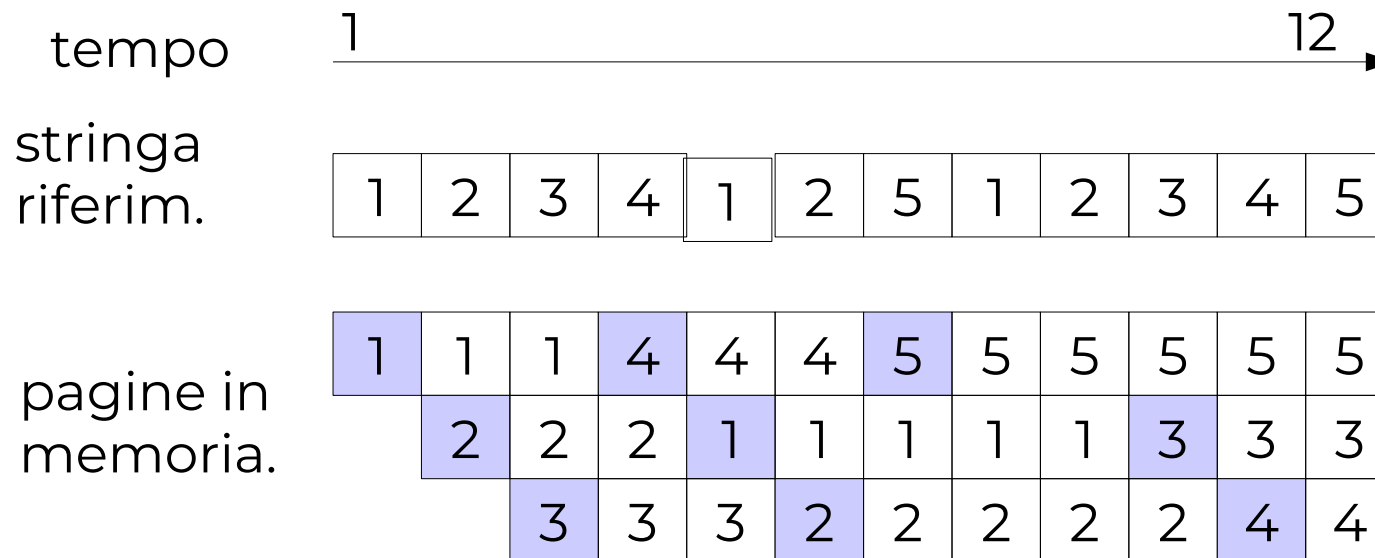
pagine in memoria.

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

Algoritmo FIFO - Esempio 2

- Caratteristiche

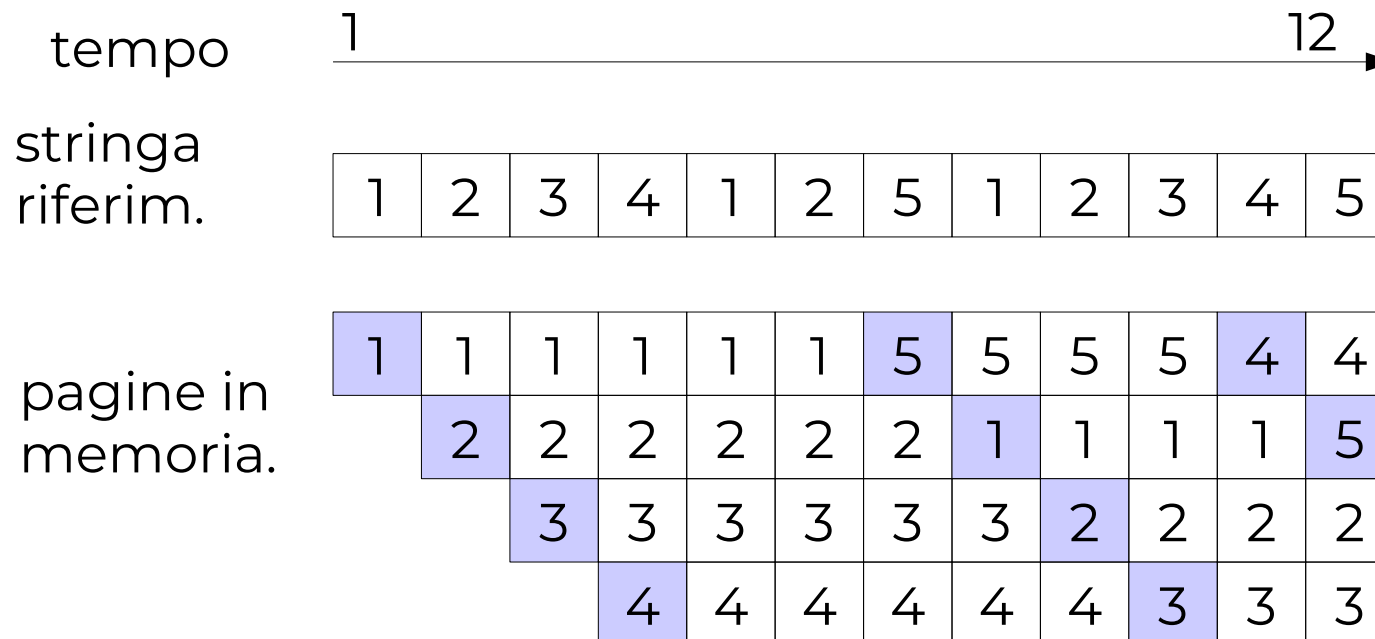
- numero di frame in memoria: 3
- numero di page fault: 9 (su 12 accessi in memoria)



Algoritmo FIFO - Esempio 3

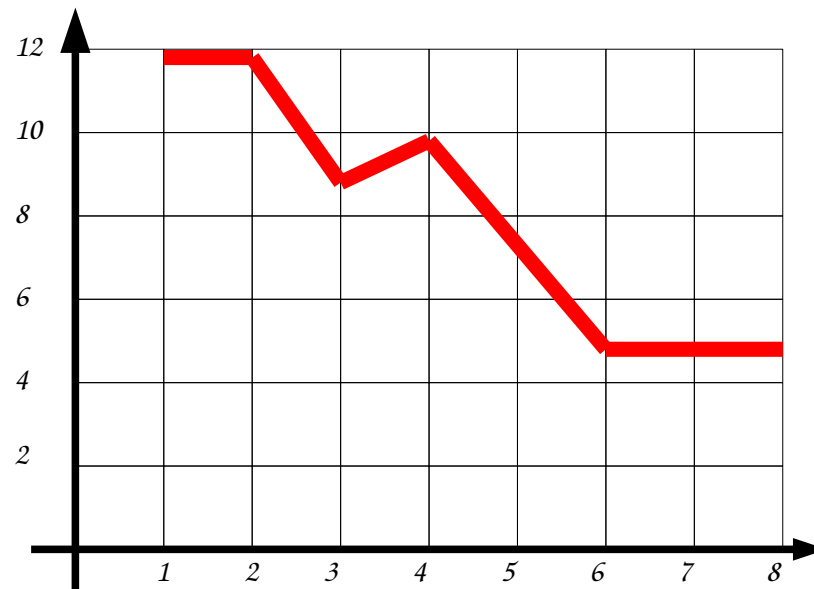
- Caratteristiche

- numero di frame in memoria: 4
- numero di page fault: 10! (su 12 accessi in memoria)
- il numero di page fault è aumentato!



Anomalia di Belady

- ♦ In alcuni algoritmi di rimpiazzamento:
 - ♦ non è detto che aumentando il numero di frame il numero di page fault diminuisca (e.g., FIFO)
- ♦ Questo fenomeno indesiderato si chiama **Anomalia di Belady**



Algoritmo MIN - Ottimale

- ◆ **Descrizione**

- ◆ seleziona come pagina vittima una pagina che non sarà più acceduta o la pagina che verrà acceduta nel futuro più lontano

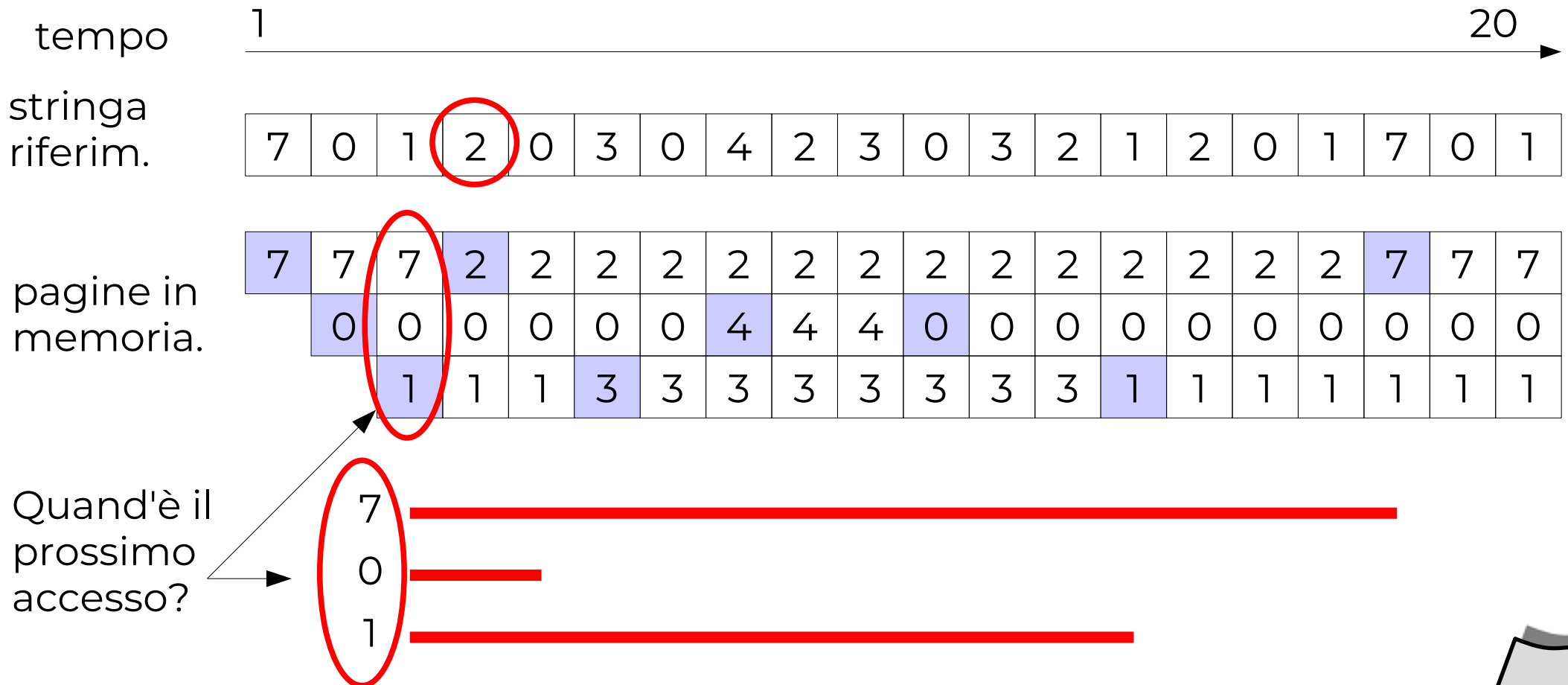
- ◆ **Considerazioni**

- ◆ è ottimale perché fornisce il minimo numero di page fault
- ◆ è un algoritmo teorico perché richiederebbe la conoscenza a priori della stringa dei riferimenti futuri del programma
- ◆ viene utilizzato a posteriori come paragone per verificare le performance degli algoritmi di rimpiazzamento reali

Algoritmo MIN - Esempio

Caratteristiche

- numero di frame in memoria: 3
- numero di page fault: 9 (su 20 accessi in memoria)



Algoritmo LRU (Least Recently Used)

- ◆ **Descrizione**

- ◆ seleziona come pagina vittima la pagina che è stata usata meno recentemente nel passato

- ◆ **Considerazioni**

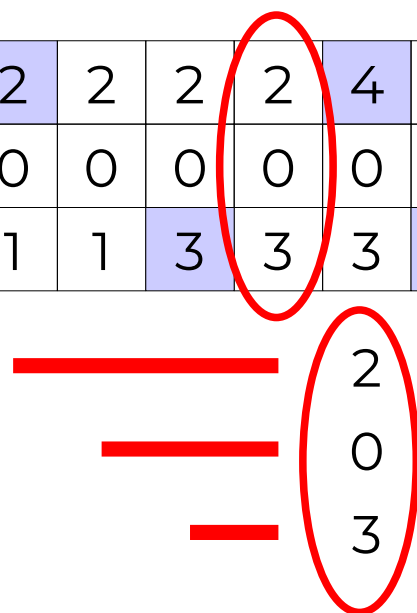
- ◆ è basato sul presupposto che la distanza tra due riferimenti successivi alla stessa pagina non vari eccessivamente
- ◆ stima la distanza nel futuro utilizzando la distanza nel passato

Algoritmo LRU - Esempio

Caratteristiche

- numero di frame in memoria: 3
- numero di page fault: 12 (su 20 accessi in memoria)

tempo	1																			20																			→
stringa riferim.	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1																			
pagine in memoria.	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1																			
		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0																			
			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	2	7	7	7																		



Quand'è stato l'ultimo accesso?

Algoritmo LRU - Implementazione

- ♦ E' necessario uno specifico supporto hardware
- ♦ La MMU
 - ♦ deve registrare nella tabella delle pagine un time-stamp quando accede ad una pagina
 - ♦ il time-stamp può essere implementato come un contatore che viene incrementato ad ogni accesso in memoria
- ♦ Nota
 - ♦ bisogna gestire l'overflow dei contatori (wraparound)
 - ♦ i contatori devono essere memorizzati in memoria e questo richiede accessi aggiuntivi alla memoria
 - ♦ la tabella deve essere scandita totalmente per trovare la pagina LRU

Algoritmo LRU - Implementazione

- ◆ **Implementazione basata su stack**

- ◆ si mantiene uno stack di pagine
- ◆ tutte le volte che una pagina viene acceduta, viene rimossa dallo stack (se presente) e posta in cima allo stack stesso
- ◆ in questo modo:
 - ◆ in cima si trova la pagina utilizzata più di recente
 - ◆ in fondo si trova la pagina utilizzata meno di recente

- ◆ **Nota**

- ◆ l'aggiornamento di uno stack organizzato come double-linked list richiede l'aggiornamento di 6 puntatori
- ◆ la pagina LRU viene individuata con un accesso alla memoria
- ◆ esistono implementazioni hardware di questo meccanismo

Algoritmi a stack

♦ Definizione

- ♦ Data una stringa di riferimenti s
- ♦ si indichi con $S_t(s, A, m)$ l'insieme delle pagine mantenute in memoria centrale al tempo t dell'algoritmo A , data una memoria di m frame relativamente alla stringa s
- ♦ un algoritmo di rimpiazzamento viene detto "*a stack*" (*stack algorithm*) se per ogni istante t e per ogni stringa s si ha:

$$S_t(s, A, m) \subseteq S_t(s, A, m+1)$$

♦ In altre parole

- ♦ se l'insieme delle pagine in memoria con m frame è sempre un sottoinsieme delle pagine in memoria con $m+1$ frame
- ♦ Per ogni stringa per ogni tempo per ogni ampiezza di memoria

Algoritmi a stack

- ♦ Teorema:
 - ♦ un algoritmo a stack non genera casi di Anomalia di Belady
- ♦ Teorema
 - ♦ l'algoritmo di LRU è a stack

Algoritmo LRU - Implementazione

- ♦ In entrambi i casi (contatori, stack), mantenere le informazioni per LRU è troppo costoso
- ♦ In realtà
 - ♦ poche MMU forniscono il supporto hardware per l'algoritmo LRU
 - ♦ alcuni sistemi non forniscono alcun tipo di supporto, e in tal caso l'algoritmo FIFO deve essere utilizzato
- ♦ Reference bit
 - ♦ alcuni sistemi forniscono supporto sotto forma di reference bit
 - ♦ tutte le volte che una pagina è acceduta, il reference bit associato alla pagina viene aggiornato a 1

Approssimare LRU

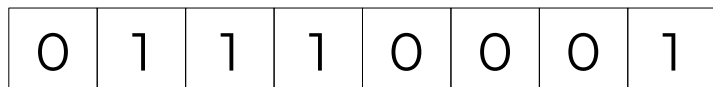
- ♦ **Come utilizzare il reference bit**
 - ♦ inizialmente, tutti i bit sono posti a zero dal s.o.
 - ♦ durante l'esecuzione dei processi, le pagine in memoria vengono accedute e i reference bit vengono posti a 1
 - ♦ periodicamente, è possibile osservare quali pagine sono state accedute e quali non osservando i reference bit
- ♦ **Tramite questi bit**
 - ♦ non conosciamo l'ordine in cui sono state usate
 - ♦ ma possiamo utilizzare queste informazioni per *approssimare l'algoritmo LRU*

Approssimare LRU

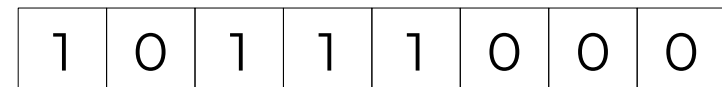
- ◆ **Additional-Reference-Bit-Algorithm**

- ◆ possiamo aumentare le informazioni di ordine "salvando" i reference bit ad intervalli regolari (ad es., ogni 100 ms)
- ◆ esempio: manteniamo 8 bit di "storia" per ogni pagina
- ◆ il nuovo valore del reference bit viene salvato tramite shift a destra della storia ed inserimento del bit come most signif. bit
- ◆ la pagina vittima è quella con valore minore; in caso di parità, si utilizza una disciplina FIFO

Storia pagina x



Storia pagina x



Nuovo valore ref. bit



Approssimare LRU

- ◆ **Second-chance algorithm**

- ◆ conosciuto anche come algoritmo dell'orologio
- ◆ corrisponde ad un caso particolare dell'algoritmo precedente, dove la dimensione della storia è uguale a 1

- ◆ **Descrizione**

- ◆ le pagine in memoria vengono gestite come una lista circolare
- ◆ a partire dalla posizione successiva all'ultima pagina caricata, si scandisce la lista con la seguente regola
 - ◆ se la pagina è stata acceduta (reference bit a 1)
 - il reference bit viene messo a 0
 - ◆ se la pagina non è stata acceduta (reference bit a 0)
 - la pagina selezionata è la vittima

Approssimare LRU

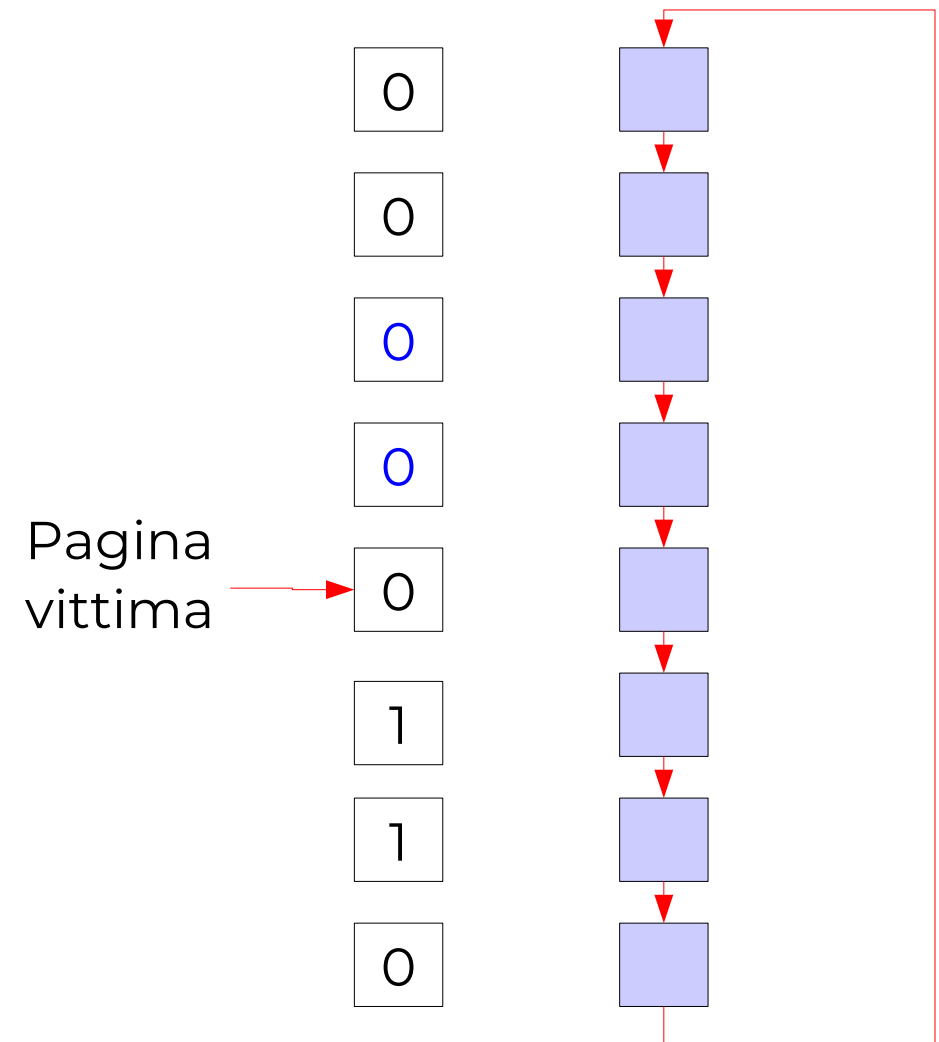
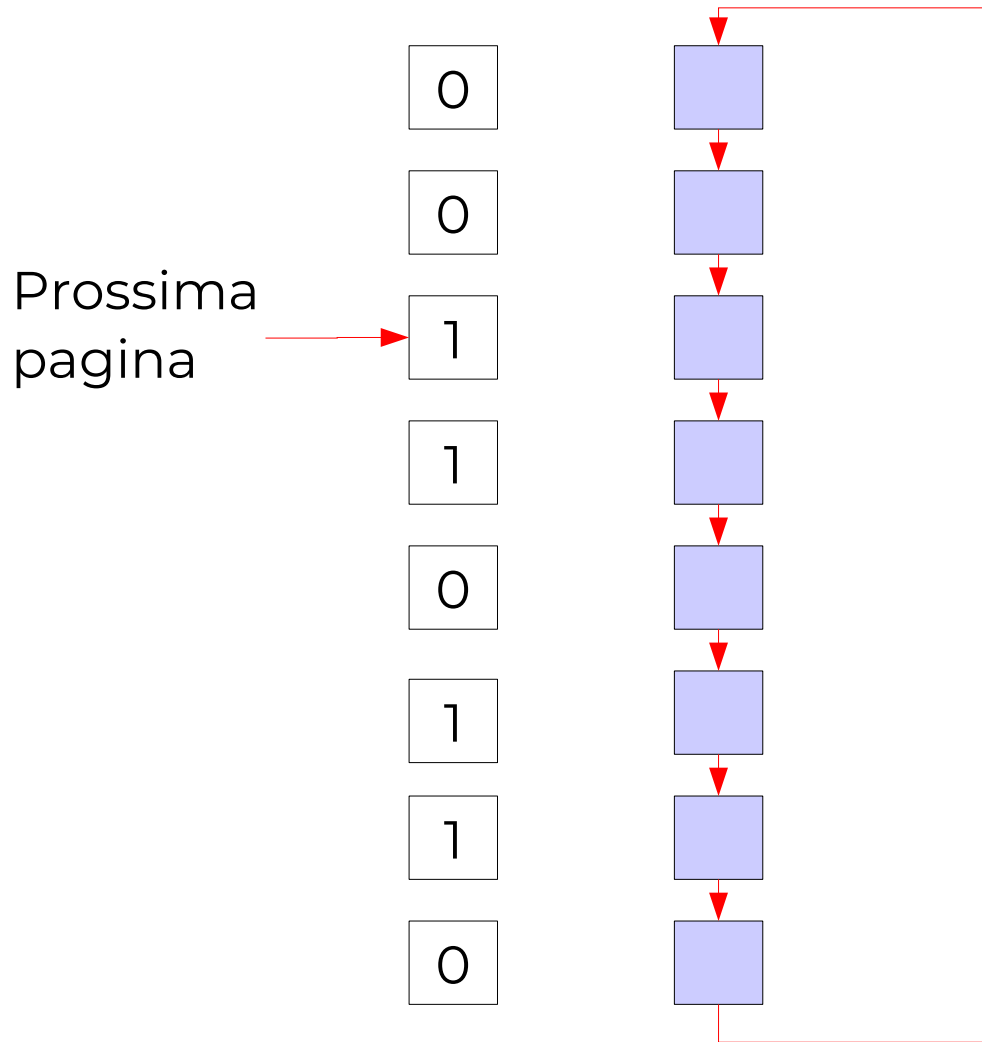
- ◆ **Considerazioni**

- ◆ l'idea è semplice:
 - ◆ l'algoritmo seleziona le pagine in modo FIFO
 - ◆ se però la pagina è stata acceduta, gli si dà una "seconda possibilità" (second chance);
 - ◆ si cercano pagine successive che non sono state accedute
- ◆ se tutte le pagine sono state accedute, degenera nel meccanismo FIFO

- ◆ **Implementazione**

- ◆ è semplice da implementare
- ◆ non richiede capacità complesse da parte della MMU

Second chance - Esempio



Altri algoritmi di rimpiazzamento

- ◆ **Least frequently used (LFU)**

- ◆ si mantiene un contatore del numero di accessi ad una pagina
- ◆ La frequenza e' il valore del contatore diviso per il "tempo" di permanenza in memoria (i.e. Il numero di accessi con quella pagina presente)
- ◆ la pagina con il valore minore viene scelta come vittima

- ◆ **Motivazione**

- ◆ una pagina utilizzata spesso dovrebbe avere un contatore molto alto

- ◆ **Implementazione**

- ◆ può essere approssimato tramite reference bit

- ◆ **Problemi**

Allocazione

- ♦ **Algoritmo di allocazione (per memoria virtuale)**
 - ♦ si intende l'algoritmo utilizzato per scegliere quanti frame assegnare ad ogni singolo processo
- ♦ **Allocazione locale**
 - ♦ ogni processo ha un insieme proprio di frame
 - ♦ poco flessibile
- ♦ **Allocazione globale**
 - ♦ tutti i processi possono allocare tutti i frame presenti nel sistema (sono in competizione)
 - ♦ può portare a *trashing*

Trashing



- ◆ **Definizione**

- ◆ un processo (o un sistema) si dice che è in trashing quando spende più tempo per la paginazione che per l'esecuzione

- ◆ **Possibili cause**

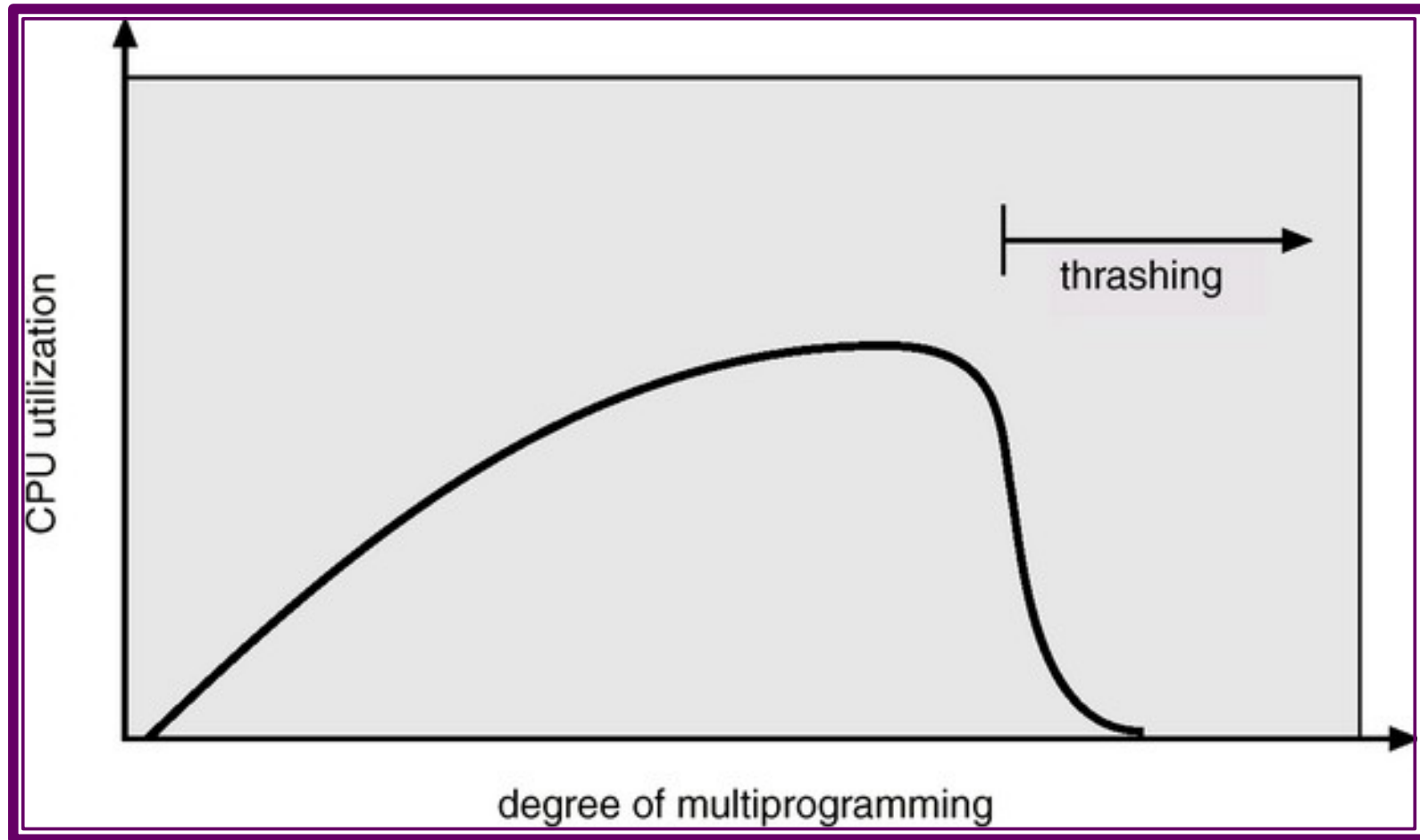
- ◆ in un sistema con allocazione globale, si ha trashing se i processi tendono a "rubarsi i frame a vicenda",
- ◆ i.e. non riescono a tenere in memoria i frame utili a breve termine (perchè altri processi chiedono frame liberi) e quindi generano page fault ogni pochi passi di avanzamento

Trashing

- ◆ **Esempio**

- ◆ esaminiamo un sistema che accetti nuovi processi quando il grado di utilizzazione della CPU è basso
- ◆ se per qualche motivo gran parte dei processi entrano in page fault:
 - ◆ la ready queue si riduce
 - ◆ il sistema sarebbe indotto ad accettare nuovi processi....
 - ◆ E' UN ERRORE!
- ◆ statisticamente, il sistema:
 - ◆ genererà un maggior numero di page fault
 - ◆ di conseguenza diminuirà il livello della multiprogrammazione

Trashing



Working Set

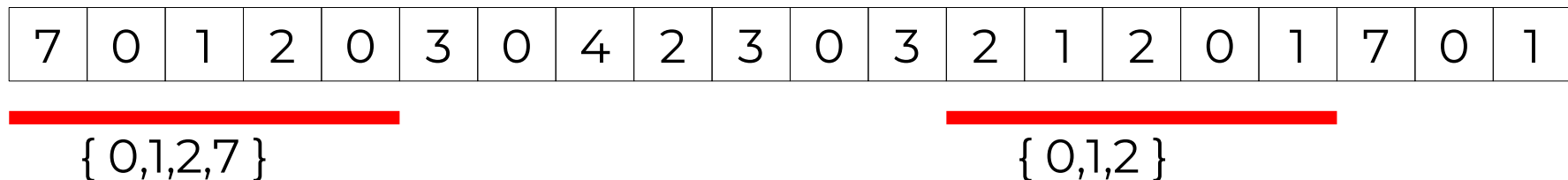
- ◆ **Definizione**

- ◆ si definisce *working set di finestra Δ* l'insieme delle pagine accedute nei più recenti Δ riferimenti

- ◆ **Considerazione**

- ◆ è una rappresentazione approssimata del concetto di località
- ◆ se una pagina non compare in Δ riferimenti successivi in memoria, allora esce dal working set; non è più una pagina su cui si lavora attivamente

- ◆ **Esempio: $\Delta = 5$**



Working Set

- ♦ Cosa serve il Working Set?
 - ♦ Se l'ampiezza della finestra è ben calcolata, il working set è una buona approssimazione dell'insieme delle pagine "utili"...
 - ♦ sommando quindi l'ampiezza di tutti i working set dei processi attivi, questo valore deve essere sempre minore del numero di frame disponibili
 - ♦ altrimenti il sistema è in trashing

Working Set

- ♦ Se si sceglie Δ troppo piccolo
 - ♦ si considera non più utile ciò che in realtà serve
 - ♦ Si sottovaluta il numero di pagine necessarie per il processo
 - ♦ Falsi negativi di trashing
- ♦ Se si sceglie Δ troppo grande
 - ♦ si considera utile anche ciò che non serve più
 - ♦ Si sopravvaluta il numero di pagine necessarie
 - ♦ Falsi positivi di trashing

Working Set

- ◆ Come si usa il Working Set
 - ◆ serve per controllare l'allocazione dei frame ai singoli processi
 - ◆ quando ci sono sufficienti frame disponibili non occupati dai working set dei processi attivi, allora si può attivare un nuovo processo
 - ◆ se al contrario la somma totale dei working set supera il numero totale dei frame, si può decidere di sospendere l'esecuzione di un processo