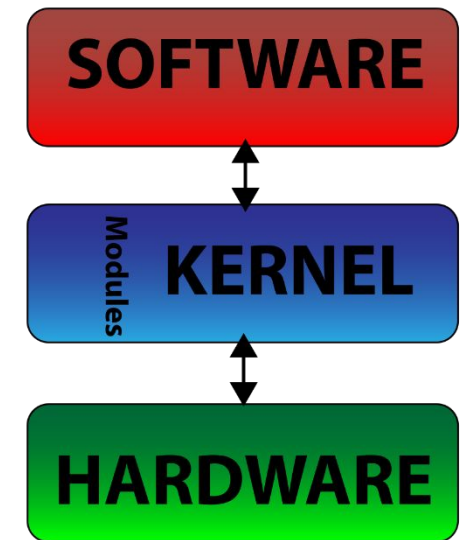
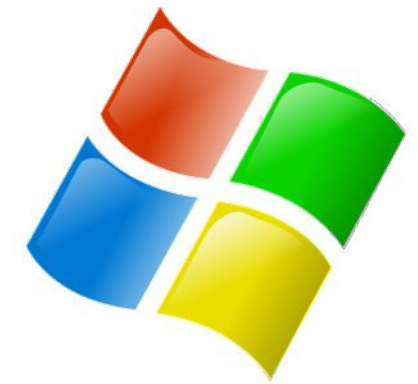




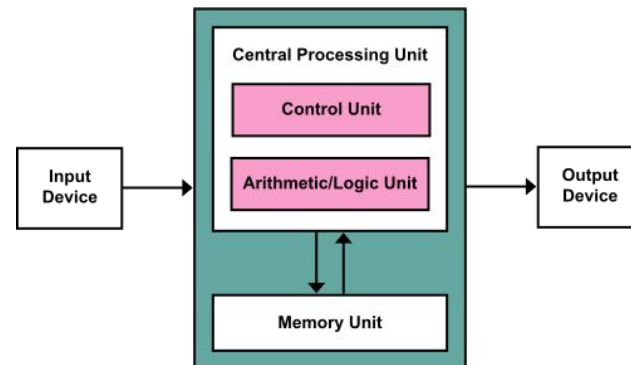
**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Operativi

Memoria virtuale

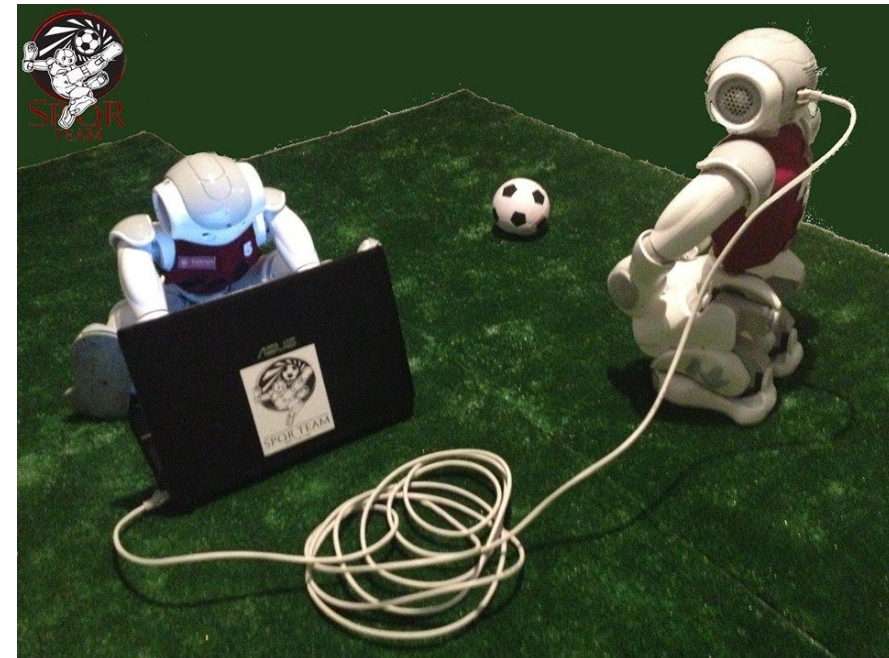
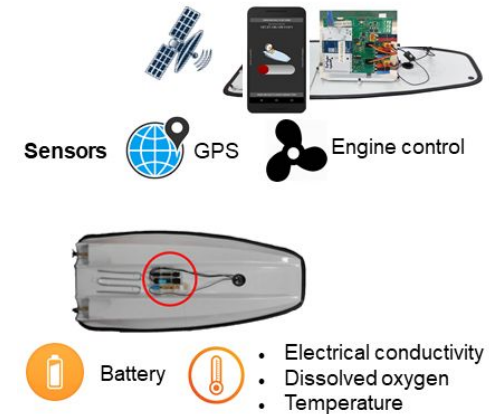


Docente:
**Domenico Daniele
Bloisi**



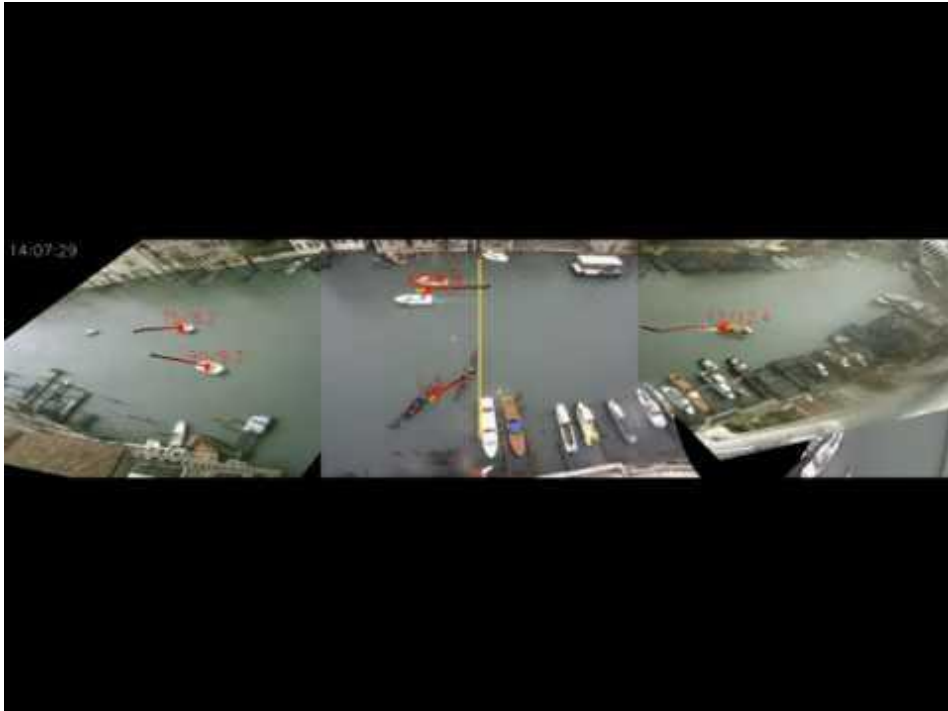
Domenico Daniele Bloisi

- Professore Associato
Dipartimento di Matematica, Informatica
ed Economia
Università degli studi della Basilicata
<http://web.unibas.it/bloisi>
- SPQR Robot Soccer Team
Dipartimento di Informatica, Automatica
e Gestionale Università degli studi di
Roma “La Sapienza”
<http://spqr.diag.uniroma1.it>



Interessi di ricerca

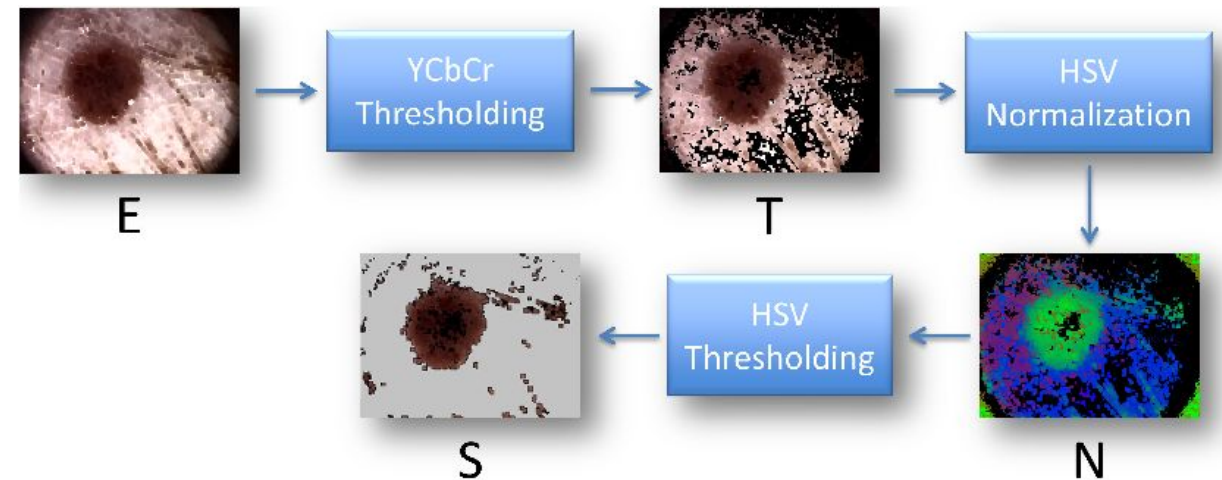
- Intelligent surveillance
- Robot vision
- Medical image analysis



https://youtu.be/9a70Ucgbi_U



<https://youtu.be/2KHNZX7UIWQ>



UNIBAS Wolves <https://sites.google.com/unibas.it/wolves>



- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.

- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome



<https://youtu.be/ji0OmkaWh20>

Informazioni sul corso

- Home page del corso:
<http://web.unibas.it/bloisi/corsi/sistemi-operativi.html>
- Docente: Domenico Daniele Bloisi
- Periodo: I semestre ottobre 2022 – gennaio 2023
 - Lunedì dalle 15:00 alle 17:00 (Aula Leonardo)
 - Martedì dalle 08:30 alle 10:30 (Aula 1)

Ricevimento

- In presenza, durante il periodo delle lezioni:
Lunedì dalle 17:00 alle 18:00
presso Edificio 3D, Il piano, stanza 15
Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni
- Tramite google Meet e al di fuori del periodo delle lezioni:
da concordare con il docente tramite email

Per prenotare un appuntamento inviare
una email a
domenico.bloisi@unibas.it



Programma – Sistemi Operativi

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- **Gestione della memoria centrale**
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

Memoria virtuale

- Le istruzioni di un processo, per essere eseguite, devono risiedere in memoria fisica
- In sistemi multiprogrammati più processi risiedono contemporaneamente in memoria
- La memoria fisica potrebbe non essere sufficientemente capiente
- La **memoria virtuale** è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti in memoria fisica

Memoria virtuale

La **memoria virtuale** facilita la programmazione, poiché il programmatore non deve preoccuparsi della quantità di memoria fisica disponibile, ma può concentrarsi sul problema da risolvere con il programma

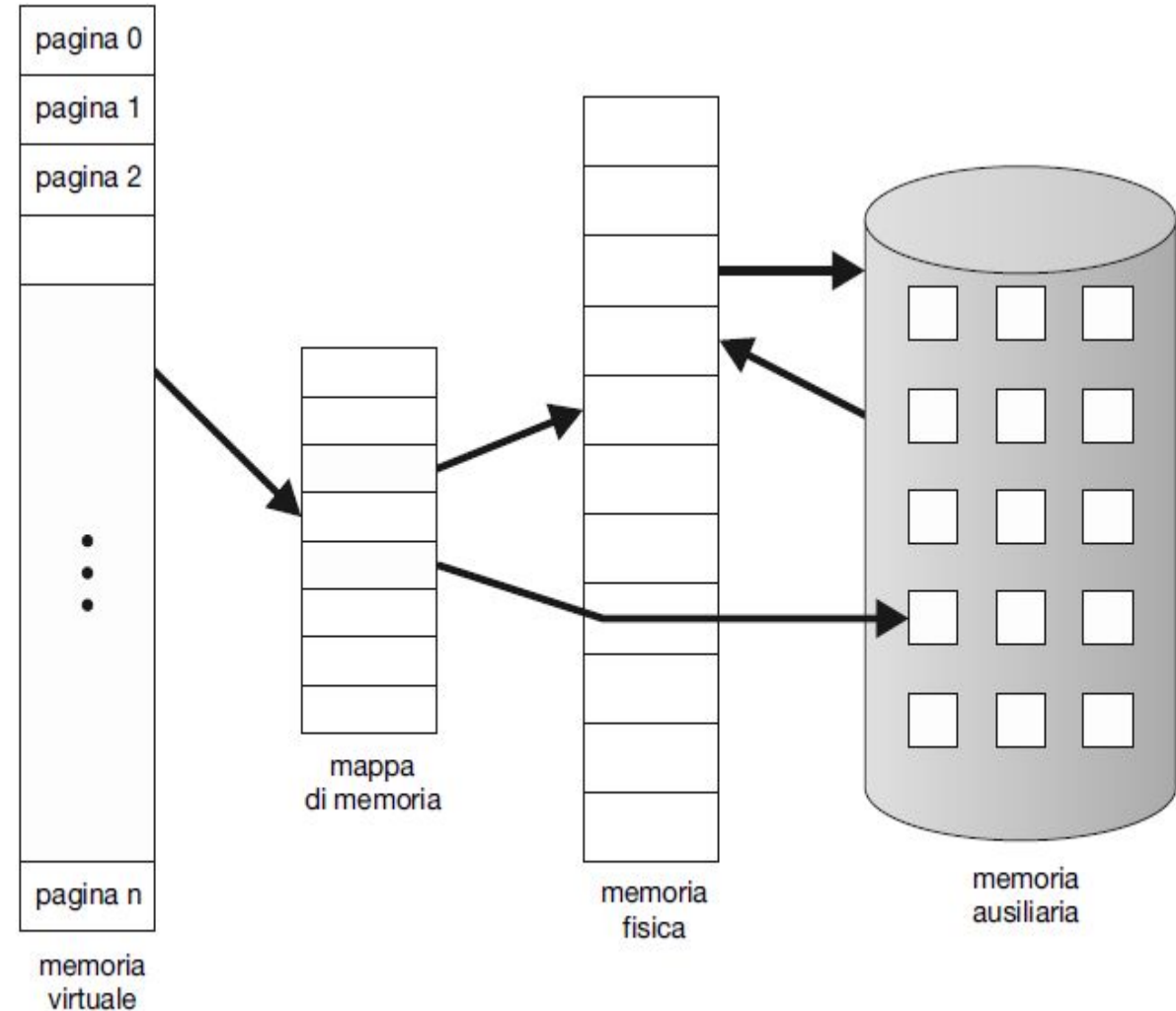


Figura 10.1 Schema che mostra una memoria virtuale più grande di quella fisica.

Vantaggi della memoria virtuale

1. I programmi possono essere più grandi della memoria fisica
2. La memoria centrale viene astratta in un vettore di memorizzazione molto grande e uniforme
3. I processi possono condividere facilmente file
4. Possono essere realizzate memorie condivise

Svantaggi della memoria virtuale

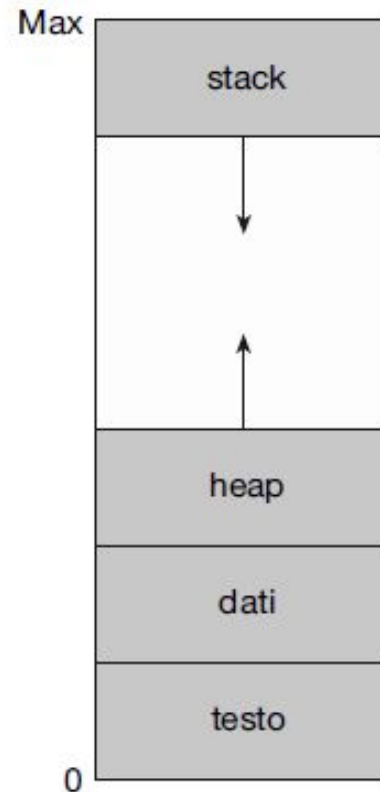
1. Realizzazione complessa
2. Se usata scorrettamente, riduce di molto le prestazioni del sistema

Spazio degli indirizzi virtuali

Spazio degli indirizzi virtuali



collocazione dei processi
in memoria dal punto di
vista **logico** (o **virtuale**)



Uno spazio degli
indirizzi virtuali che
contiene buchi si
definisce sparso

Figura 10.2 Spazio degli indirizzi virtuali di un processo in memoria.

Condivisione delle pagine

Oltre a separare la memoria logica da quella fisica, la **memoria virtuale** offre il vantaggio di condividere i file e la memoria fra due o più processi, mediante la condivisione delle pagine.

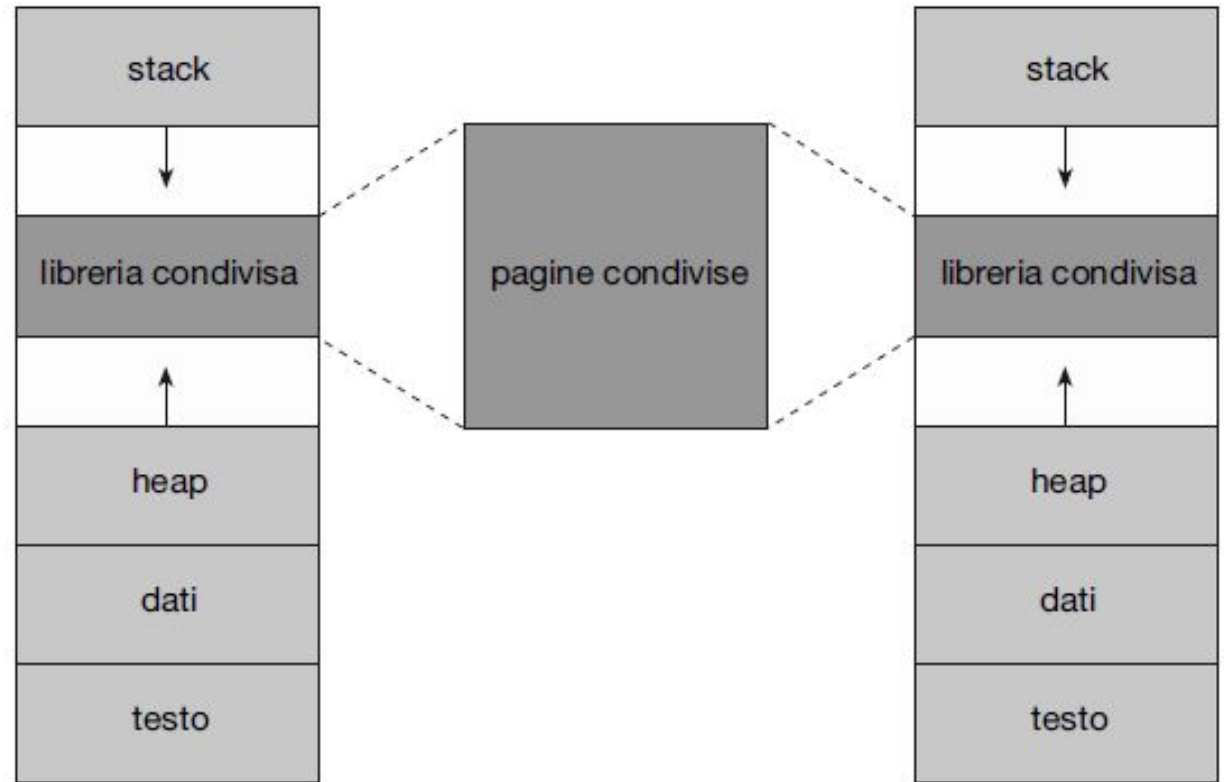


Figura 10.3 Condivisione delle librerie tramite la memoria virtuale.

Paginazione su richiesta

Paginazione su richiesta → strategia di allocazione di memoria che consiste nel caricare le pagine nel momento in cui servono realmente



le pagine sono caricate in memoria solo quando richieste durante l'esecuzione del programma

La **paginazione su richiesta** mostra uno dei principali vantaggi della **memoria virtuale**: caricando solo le parti necessarie dei programmi la memoria viene utilizzata in modo *più efficiente*.

Paginazione su richiesta

1. Mentre un processo è in esecuzione, alcune pagine saranno in memoria, altre saranno in memoria secondaria (es. disco)
2. È necessario un supporto hardware per tenere traccia di questa divisione
3. Possiamo utilizzare lo schema con bit di validità a tale scopo

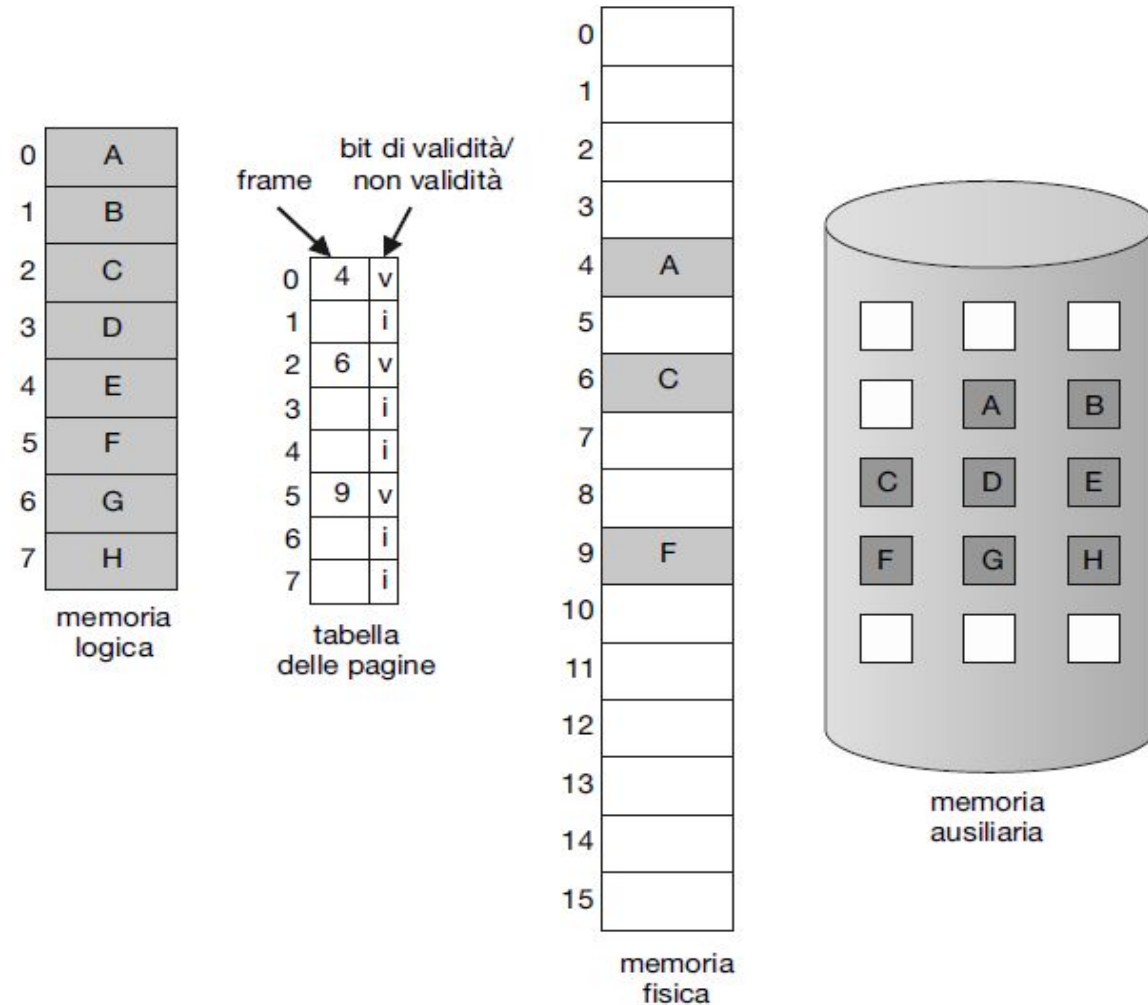


Figura 10.4 Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale.

Page fault

L'accesso a una pagina contrassegnata come non valida causa un evento o eccezione di **page fault** (*pagina mancante*)

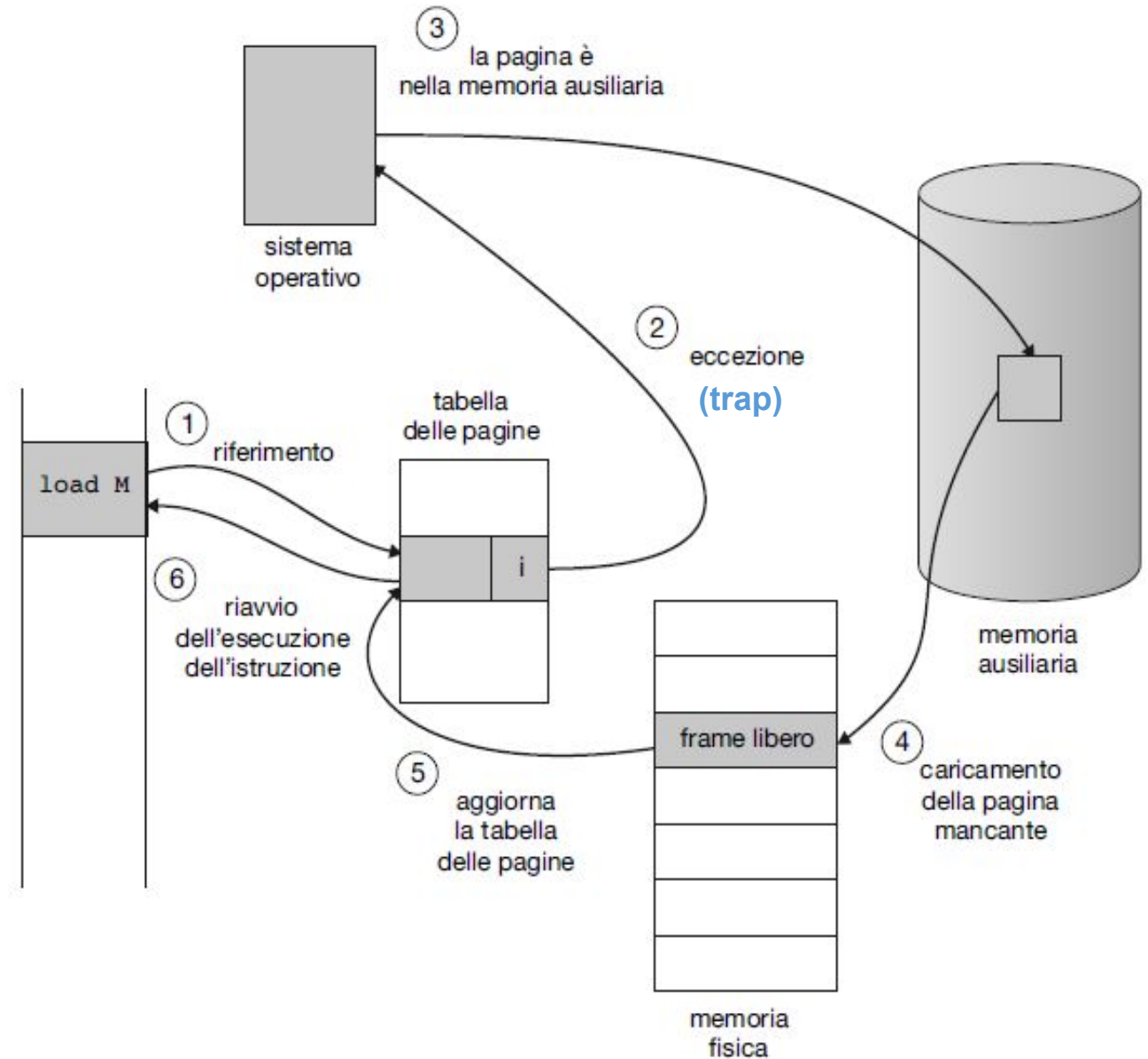


Figura 10.5 Fasi di gestione di un page fault.

Hardware per la paginazione su richiesta

1. Tabella delle pagine con **bit di validità**
2. Memoria secondaria costituita da un disco ad alta velocità (**dispositivo di swap**). La porzione di memoria dedicata per lo swapping si chiama swap space

Lista dei frame liberi

Lista dei frame liberi → insieme di frame disponibili e utilizzabili per soddisfare le richieste

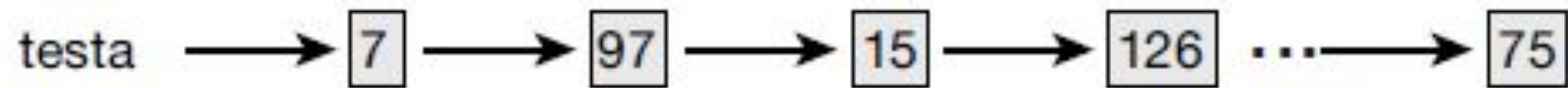


Figura 10.6 Lista dei frame liberi.

- All'avvio di un sistema tutta la memoria disponibile viene inserita nella lista dei **frame liberi**.
- A un certo punto la **lista** diventa **vuota** → *deve essere ripopolata*

Allocazione dei frame liberi

I sistemi operativi allocano generalmente i frame liberi usando una tecnica nota come **zero-fill-on-demand**:

- i frame liberi vengono azzerati su richiesta prima di essere allocati, cancellando così il loro precedente contenuto

Prestazioni della paginazione su richiesta

La **paginazione su richiesta** può avere un effetto rilevante sulle prestazioni di un calcolatore.

Il motivo si può comprendere calcolando il **tempo di accesso effettivo** per una memoria con paginazione su richiesta.

Tempo di accesso effettivo EAT

$$T_{EAT} = p \times t_{pf} + (1 - p) \times t_{ma}$$

dove

p è la probabilità di page fault, con $0 \leq p \leq 1$

t_{ma} è il tempo di accesso in memoria (tipicamente **nanosecondi**)

t_{pf} è il tempo di gestione del page fault, che comprende:

- Gestione dell'eccezione (trap) di page fault (tipicamente **microsecondi**)
- Lettura della pagina mancante da disco (tipicamente **millisecondi**)
- Riavvio del processo (tipicamente **microsecondi**)

Esempio calcolo T_{EAT}

Si consideri una situazione in cui ci siano stati 15 riferimenti in memoria con 8 page fault

Denotando con t_{ma} il tempo di accesso in memoria e con t_{pf} il tempo necessario alla gestione del page fault, si ottiene:

tempo effettivo di accesso (Effective Access Time)

$$T_{EAT} = 7 \times t_{ma} + 8 \times t_{pf}$$

La probabilità di page fault sarà $p_{pf} = 8 / 15 = 0.53$
pari al 53%

Copiatura su scrittura

In caso di **fork**, le pagine del processo genitore sono inizialmente condivise con il processo figlio

Le pagine condivise si contrassegnano come pagine da copiare su scrittura

- se un processo (genitore o figlio) scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina

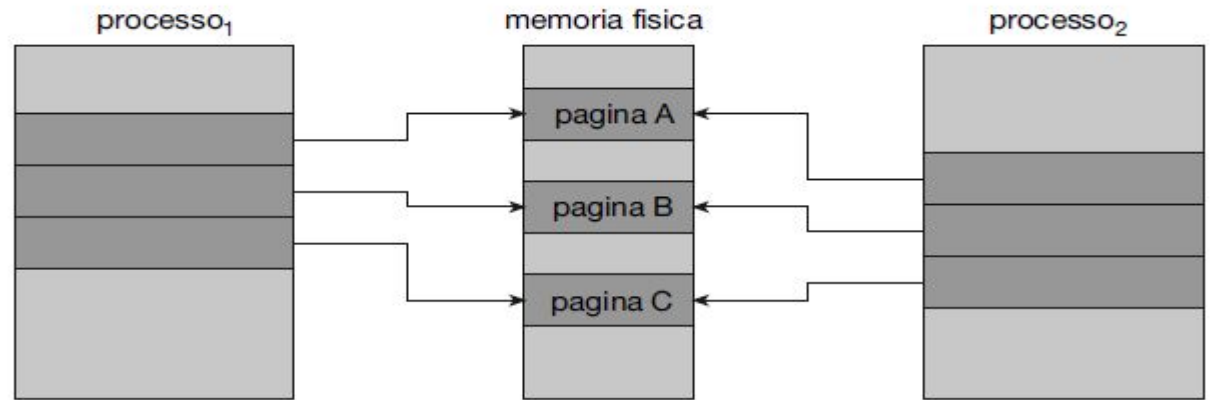


Figura 10.7 Prima della modifica alla pagina C da parte del processo 1.

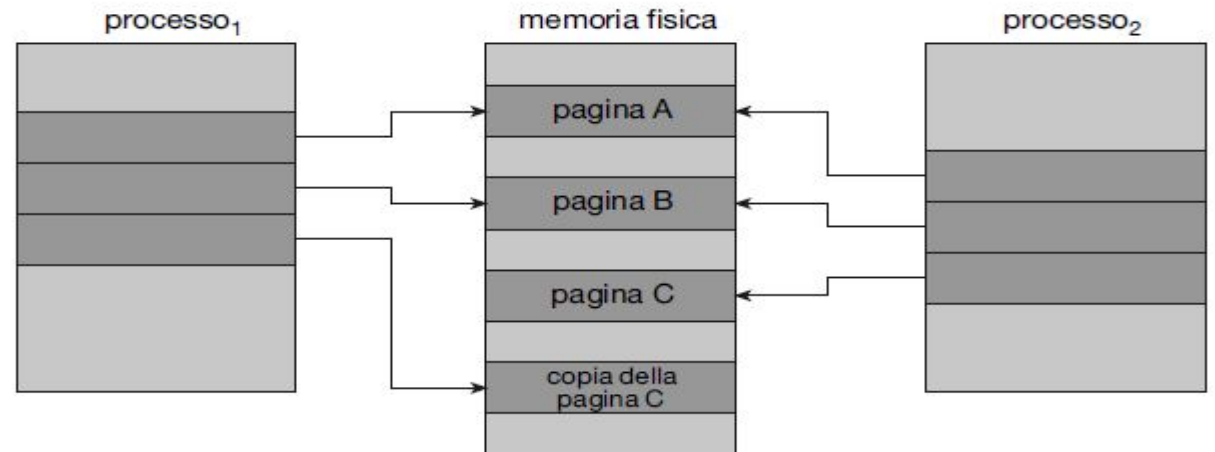


Figura 10.8 Dopo la modifica alla pagina C da parte del processo 1.

Esercizio fork

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    fork();

    fork();

    fork();

    printf("ciao dal processo: %d\n", getpid());

    return 0;
}
```

Quanti processi
vengono creati
con questo
codice?

Soluzione Esercizio fork

```
bloisi@bloisi-U36SG: ~/Desktop
bloisi@bloisi-U36SG:~/Desktop$ gcc fork-multiple.c -o fork-multiple
bloisi@bloisi-U36SG:~/Desktop$ ./fork-multiple
ciao dal processo: 5598
ciao dal processo: 5600
ciao dal processo: 5602
ciao dal processo: 5599
ciao dal processo: 5604
bloisi@bloisi-U36SG:~/Desktop$ ciao dal processo: 5601
ciao dal processo: 5605
ciao dal processo: 5603
```

Sovrallocazione

Se un processo di 10 pagine ne impiega effettivamente solo la metà, la paginazione su richiesta fa risparmiare il tempo di I/O necessario a caricare le 5 pagine mai usate

Se la memoria dispone di 40 frame, possiamo allocare fino a 8 processi in memoria invece dei 4 allocabili se ciascun processo richiedesse 10 pagine impiegandone effettivamente solo la metà

Cosa succede se un processo ha improvvisamente necessità di caricare tutte e dieci le sue pagine?

Necessità di sostituzione di pagine

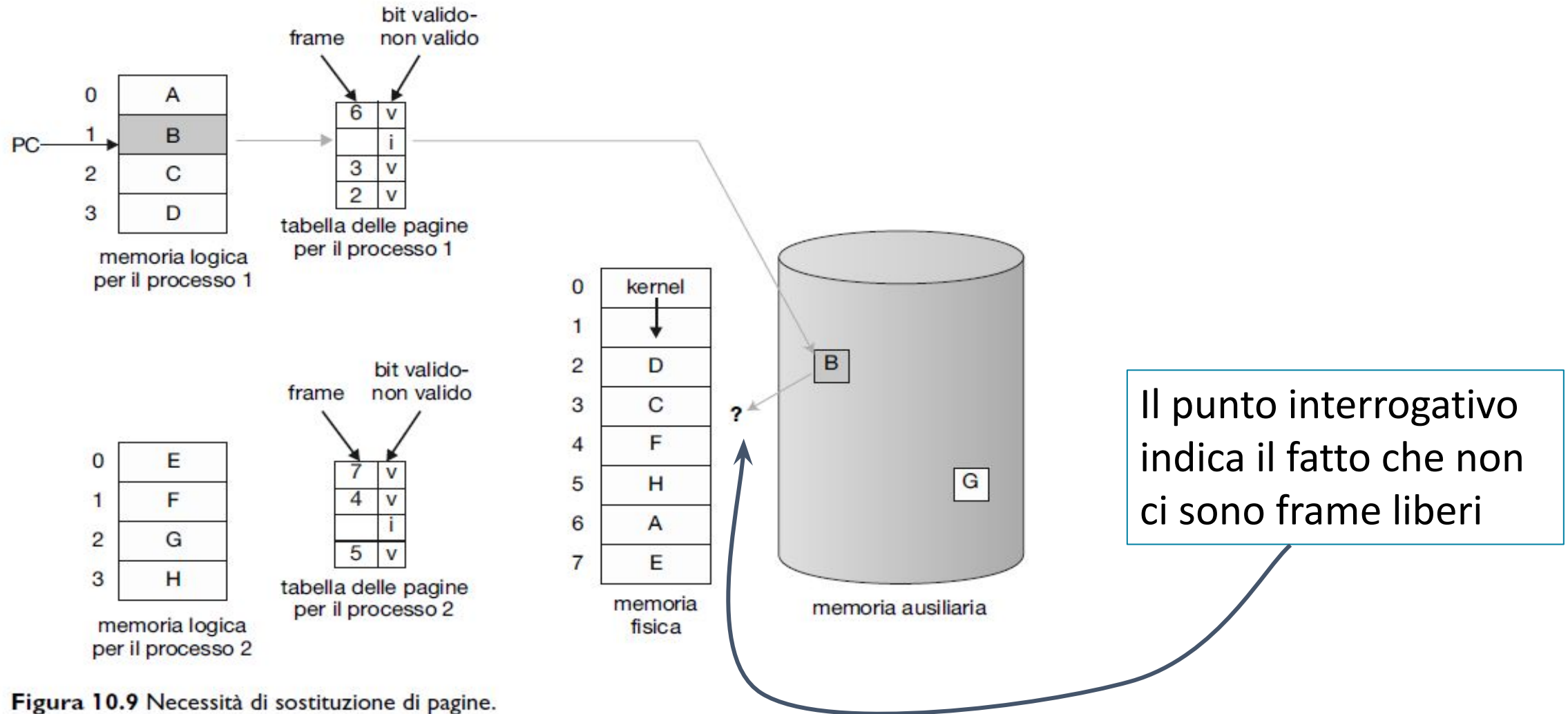


Figura 10.9 Necessità di sostituzione di pagine.

Sostituzione di pagina

Se nessun frame è libero ne viene liberato uno attualmente inutilizzato (vittima)

La vittima viene copiata nello spazio di swap e la tabella delle pagine viene modificata

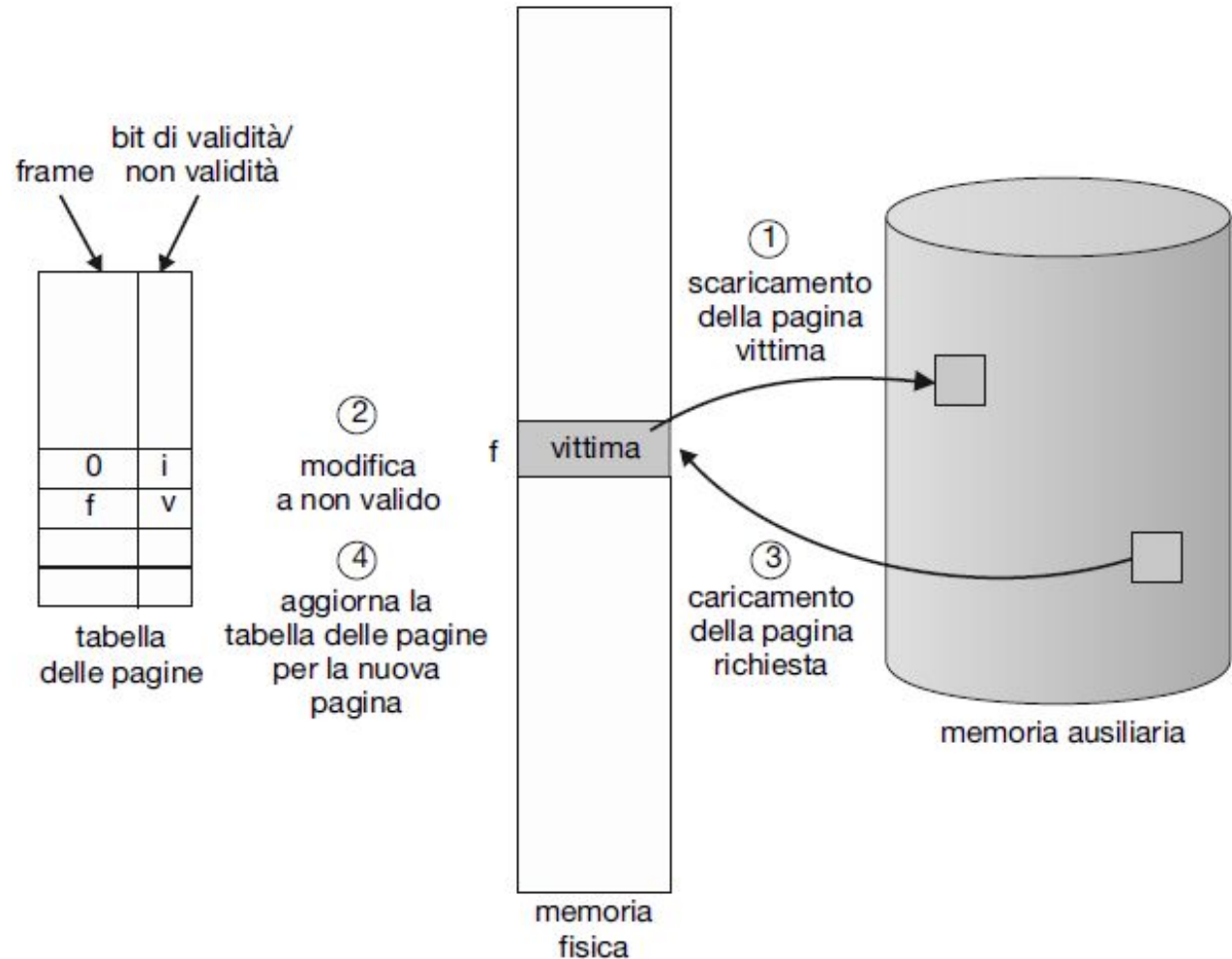


Figura 10.10 Sostituzione di una pagina.

Dirty bit

Se non esiste alcun frame libero, sono necessari due trasferimenti di pagina

1. da memoria principale a memoria secondaria
2. da memoria secondaria a memoria principale

Per limitare il tempo del page fault, l'hardware del calcolatore dispone di un bit di modifica (dirty bit), che viene posto a 1 quando si modifica una pagina

Quando si sceglie una pagina da sostituire, si legge il suo dirty bit:

- Se vale 1, la pagina deve essere scritta su disco
- Se vale 0, non è necessario scrivere su disco, la pagina già c'è.

Paginazione su richiesta

Per realizzare la **paginazione su richiesta** è necessario sviluppare



algoritmo di sostituzione
delle pagine

Quali frame sostituire in
caso di page fault

algoritmo di allocazione dei
frame

Quanti frame allocare ad
ogni singolo processo

Valutazione degli algoritmi di sostituzione

Per valutare un **algoritmo di sostituzione delle pagine** si conta il numero di page fault data successione di riferimenti in memoria, la quale contiene i numeri di pagina da accedere, e il numero di frame disponibili

Sostituzione delle pagine

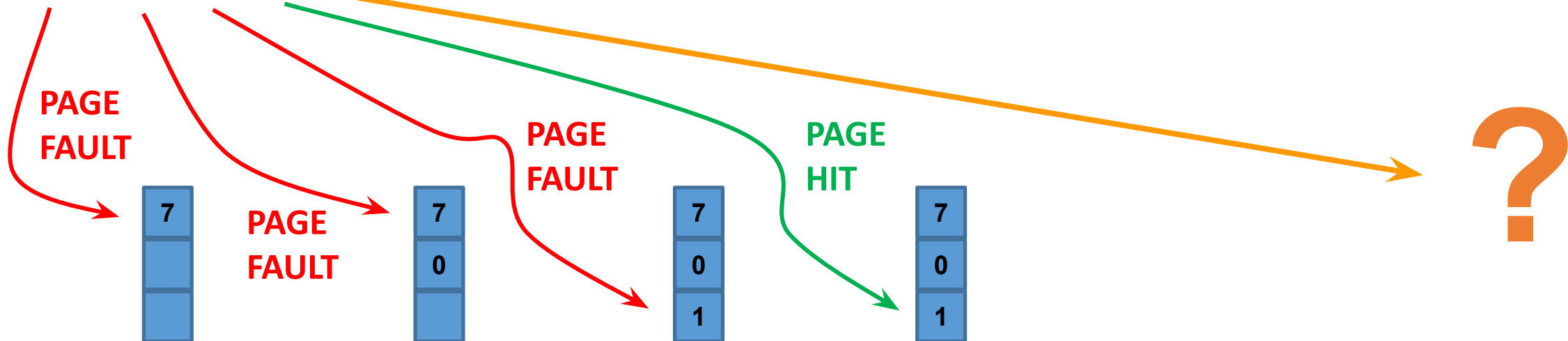
Esempio di successione di riferimenti in memoria:

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

Avendo 3 frame di memoria disponibili:



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



Valutazione degli algoritmi di sostituzione

Aumentando il numero dei **frame**, il numero di **page fault** diminuisce fino al livello minimo

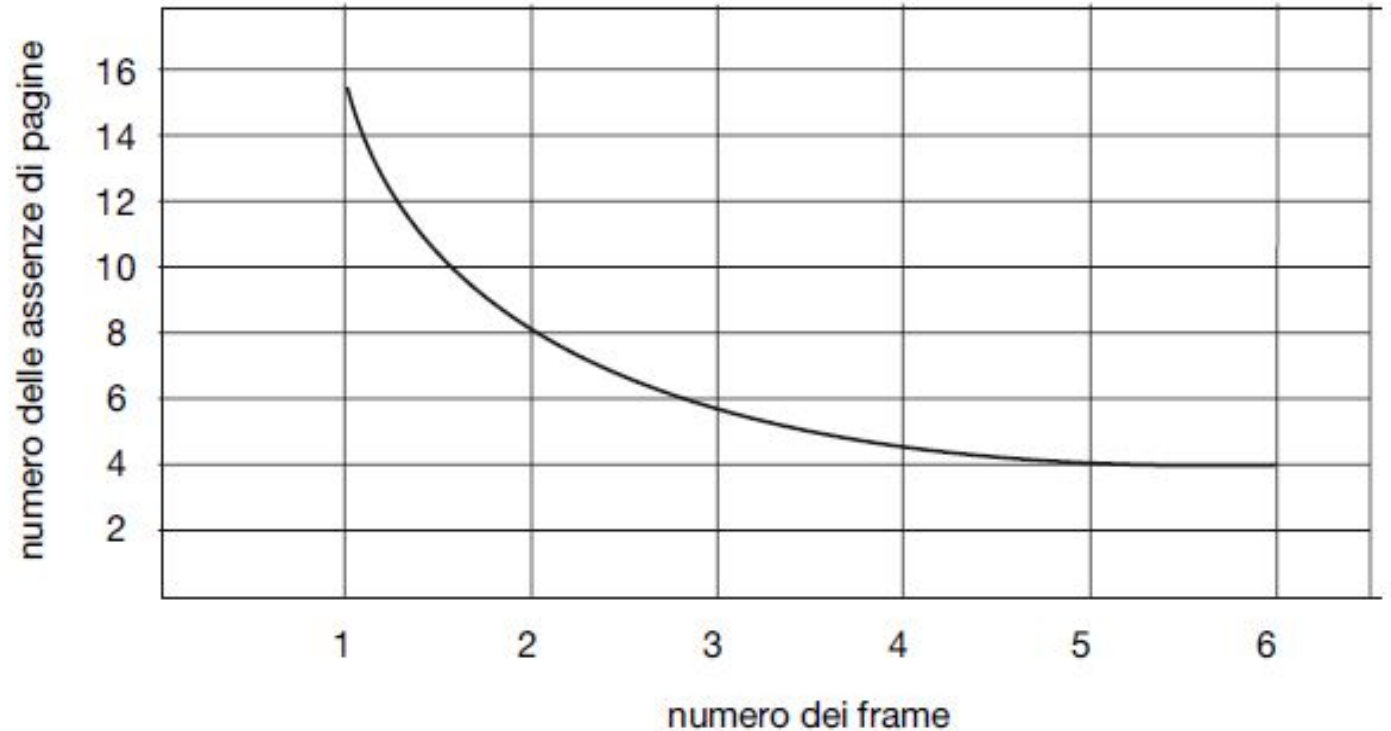


Figura 10.11 Grafico che illustra il numero di page fault rispetto al numero dei frame.

Algoritmi di sostituzione

Esistono molti **algoritmi di sostituzione delle pagine**: in genere si sceglie quello con il minimo tasso di page fault. I principali sono:

Sostituzione delle pagine secondo
l'ordine d'arrivo (FIFO)

Sostituzione ottimale delle pagine
(OPT)

Sostituzione delle pagine usate meno
recentemente (LRU)

Algoritmo di sostituzione delle pagine FIFO

algoritmo FIFO → algoritmo di sostituzione delle pagine più semplice. Associa a ogni pagina l'istante di tempo in cui quella pagina è stata portata in memoria.

Se si deve sostituire una pagina, si seleziona quella presente in memoria da più tempo.

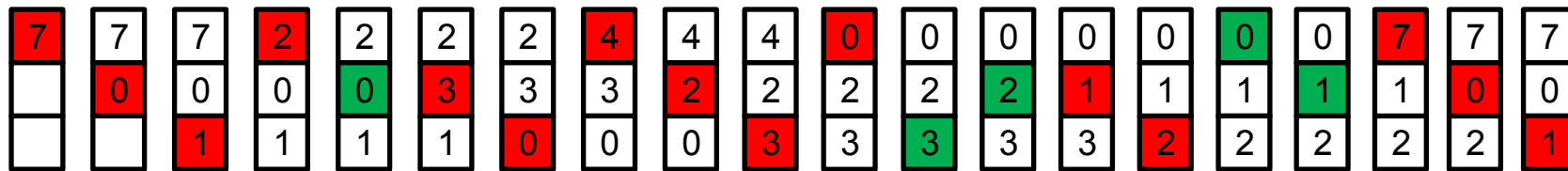
Algoritmo di sostituzione delle pagine FIFO

Esempio

Sia data la seguente successione di riferimenti in memoria con 3 frame disponibili e un algoritmo di sostituzione FIFO.

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

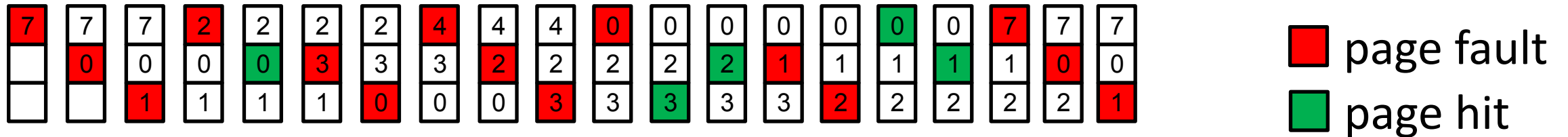
Quanti page fault si verificheranno?



■ page fault
■ page hit

Tasso di page fault

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



Il tasso di page fault per la successione di riferimenti dato e una memoria di tre frame è pari a

$$\text{Tasso di page fault} = 15 / 20 = 75\%$$

Cosa succede aumentando la memoria?

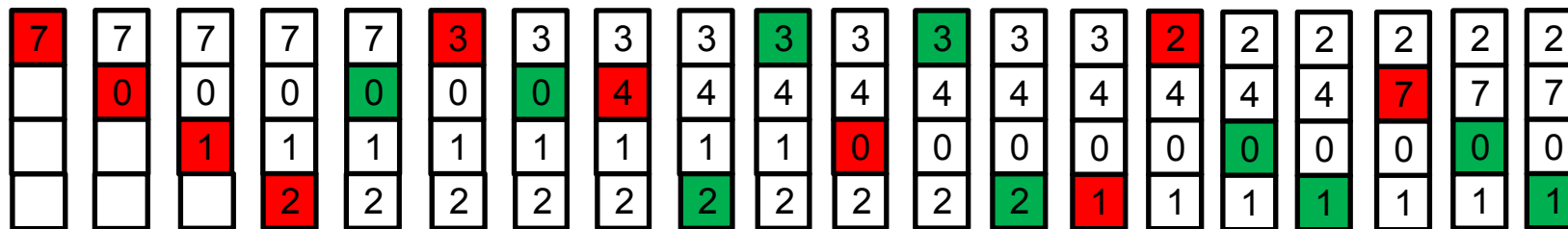
Algoritmo di sostituzione delle pagine FIFO

Successione di riferimenti:

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

Pagine in memoria:

4



■ page fault
■ page hit

Tasso di page fault = $10 / 20 = 50\%$

Anomalia di Belady

Gli algoritmi **FIFO** soffrono dell'**anomalia di Belady**: il tasso di page fault può *aumentare* con il numero dei frame assegnati ai processi.

Esercizio

Verificare che la seguente successione di riferimenti
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
produca il grafico riportato sotto



Figura 10.13 Curva dei page fault per la sostituzione FIFO su una successione di riferimenti.

Algoritmo di sostituzione delle pagine OPT

In seguito alla scoperta dell'anomalia di Belady si è ricercato un **algoritmo ottimale di sostituzione delle pagine**

Algoritmo di sostituzione OPT: *si sostituisce la pagina che non verrà usata per il periodo di tempo più lungo.*

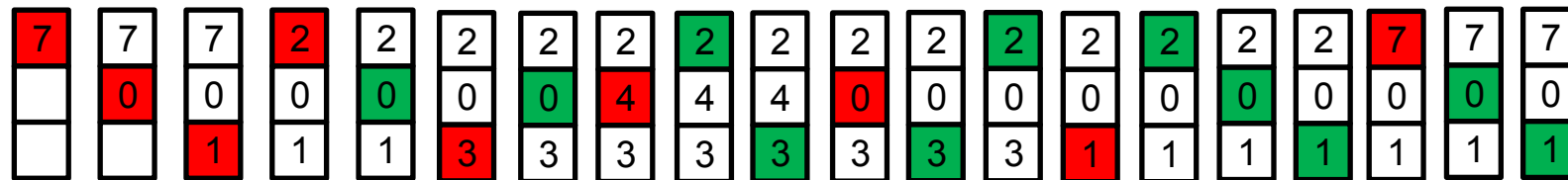
Algoritmo di sostituzione delle pagine OPT

Esempio

Sia data la seguente successione di riferimenti in memoria con 3 frame disponibili e un algoritmo di sostituzione OPT

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

Quanti page fault si verificheranno?



■ page fault
■ page hit

Tasso di page fault = $9 / 20 = 45\%$

Algoritmo di sostituzione delle pagine OPT

L'**algoritmo ottimale di sostituzione OPT** delle pagine è difficile da realizzare, perché richiede la conoscenza futura della successione della successione dei riferimenti

OPT assicura il tasso minimo di page fault per un dato numero di frame

Quindi, OPT si impiega soprattutto per studi comparativi

Algoritmo di sostituzione delle pagine LRU

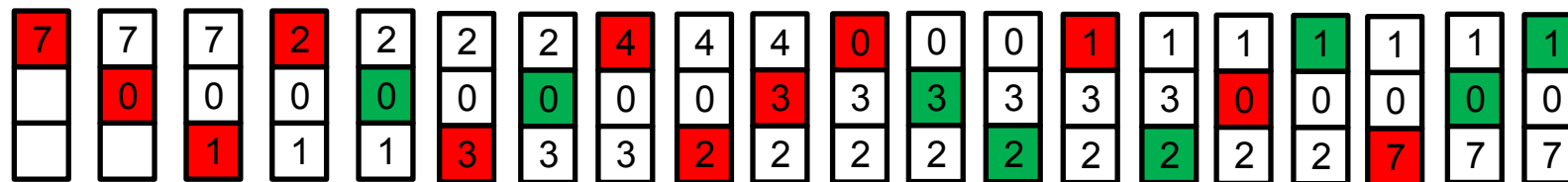
La **sostituzione LRU** associa a ogni pagina l'istante in cui è stata usata per l'ultima volta. Quando occorre sostituire una pagina, l'**algoritmo LRU** sceglie quella che non è stata usata per il periodo più lungo.

Successione di riferimenti:

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

Pagine in memoria:

3



■ page fault
■ page hit

Tasso di page fault = $12 / 20 = 60\%$

Algoritmo di sostituzione delle pagine LRU

- Il criterio **LRU** si usa spesso come **algoritmo di sostituzione** delle pagine ed è considerato valido
- Un algoritmo di sostituzione delle pagine LRU può richiedere una notevole assistenza da parte dell'hardware

Algoritmo di sostituzione delle pagine LRU

Per la realizzazione della **sostituzione delle pagine LRU** si possono utilizzare due soluzioni:

1. utilizzo di un **contatore**
2. utilizzo di uno **stack** dei numeri delle pagine

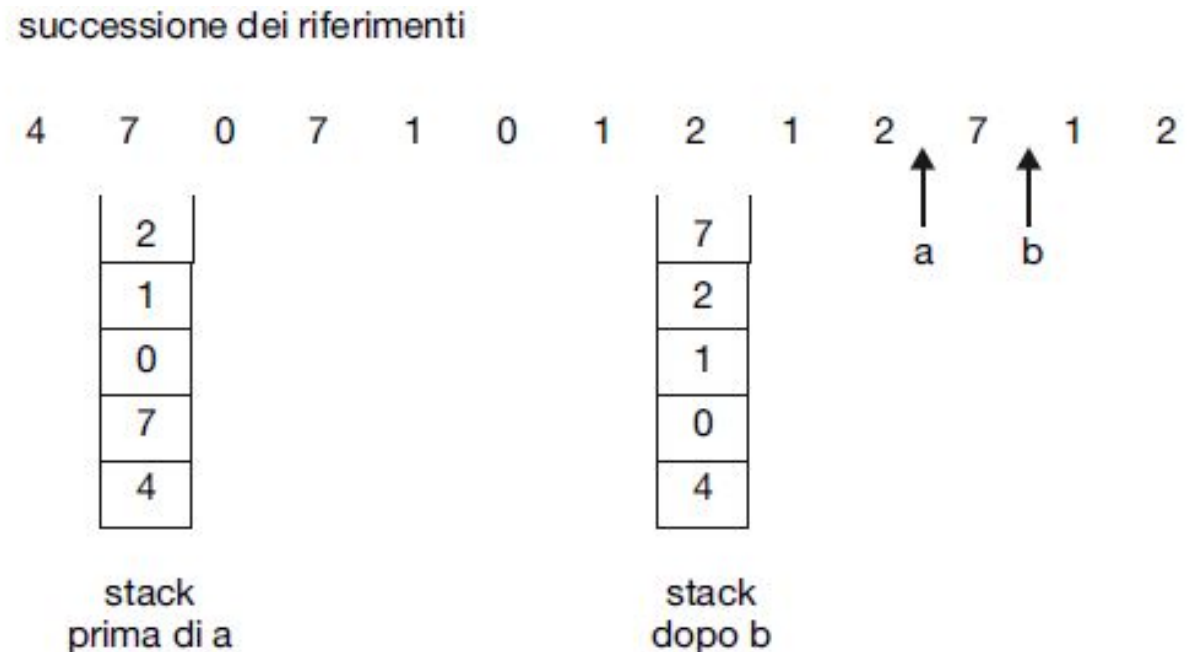


Figura 10.16 Uso di uno stack per registrare i più recenti riferimenti alle pagine.

Anomalia di Belady

Gli algoritmi OPT e LRU appartengono a una classe di algoritmi di sostituzione delle pagine chiamati **algoritmi a stack**

Gli algoritmi OPT e LRU non soffrono dell'**anomalia di Belady**

Altri algoritmi di sostituzione delle pagine

Sostituzione delle pagine per
approssimazione a LRU

Algoritmo con bit
supplementari di riferimento

Algoritmo con seconda
chance

Algoritmo con seconda
chance migliorato

Sostituzione delle pagine
basata su conteggio

Algoritmi con buffering delle
pagine

Bit di riferimento

Il bit di riferimento a una pagina è settato automaticamente dall'hardware del sistema ogni volta che si fa riferimento a quella pagina, che sia una lettura o una scrittura su qualsiasi byte della pagina.

I bit di riferimento sono associati a ciascun elemento della tabella della pagine

Algoritmo di sostituzione delle pagine con seconda chance

L'algoritmo **con seconda chance** è un algoritmo di tipo FIFO, con la seguente modifica:

Dopo aver selezionato la pagina si controlla il bit di riferimento:

- se il valore è 0, si sostituisce la pagina
- se il valore è 1, si dà una seconda chance alla pagina e si passa alla successiva pagina nella lista FIFO

Quando una pagina riceve la seconda chance, si azzera il suo bit di riferimento.

Algoritmo di sostituzione delle pagine con seconda chance

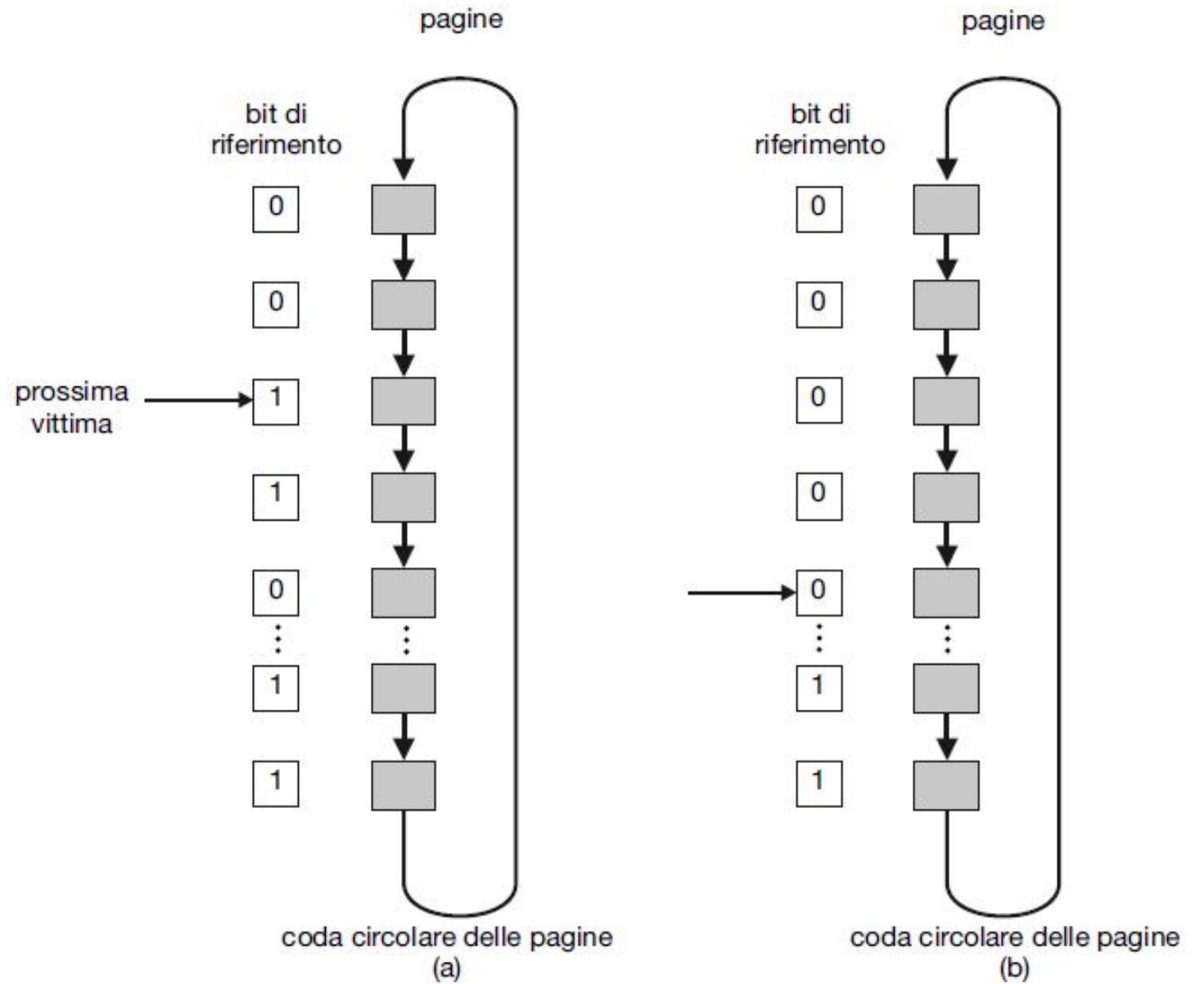


Figura 10.17 Algoritmo di sostituzione delle pagine con seconda chance (orologio).

Algoritmo di sostituzione delle pagine con seconda chance

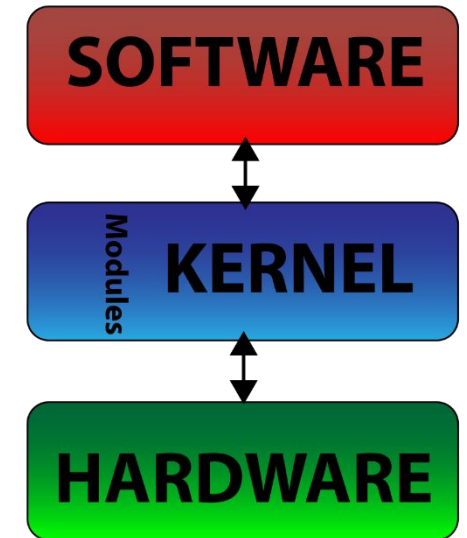
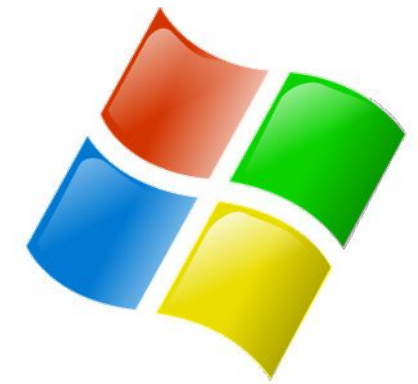
Poiché l'algoritmo **con seconda chance** è un algoritmo di tipo FIFO, anch'esso soffre dell'anomalia di Belady



**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Operativi

Memoria virtuale



Docente:
**Domenico Daniele
Bloisi**

