Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

Lezione 14

Martedì 29-11-2016

Algoritmi di rimpiazzamento delle pagine

- Un algoritmo di rimpiazzamento ottimo dovrebbe prevedere quali pagine saranno riferite nel futuro. Abbiamo già incontrato un problema analogo descrivendo l'algoritmo SJF nella schedulazione dei processi.
- Gli algoritmi reali, per predire il futuro, si basano sulle informazioni relative agli accessi effettuati nell'immediato passato.
- Infatti, a partire da un generico istante e per un certo intervallo di tempo, i processi generano indirizzi che sono spesso contenuti all'interno di un ristretto numero di pagine detto **working set** (insieme di lavoro).
- Il **working set** varia nel tempo, ma gradualmente.
- Le chiamate di funzioni provocano una variazione di **working set**.
- L'algoritmo di rimpiazzamento più semplice è il **FIFO** che sceglie la pagina che risiede da più tempo in memoria. Tuttavia è poco efficiente.

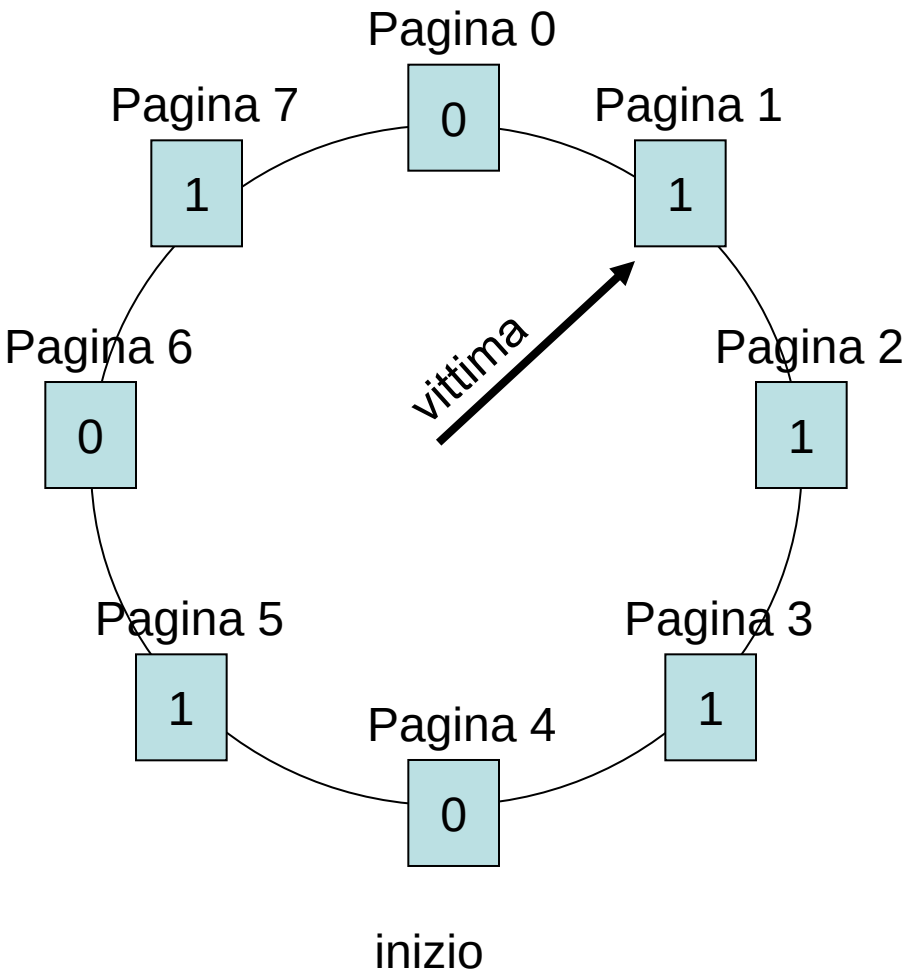
- Un algoritmo molto più complesso, ma molto più efficiente in termini di page-fault, è l'**LRU** (**Least recently Used**) che sceglie, per la sostituzione, la pagina **meno recentemente utilizzata** indipendentemente da quando è stata caricata.
- E' stato sperimentato che LRU è l'algoritmo che fornisce le migliori prestazioni. Tuttavia la sua realizzazione non è conveniente in quanto richiede un complesso supporto hardware e l'implementazione software produce un overhead troppo elevato.
- Molti algoritmi sono approssimazioni dell'LRU.
- Uno di questi è l'algoritmo **second-chance** (seconda scelta) chiamato anche **clock algorithm** (algoritmo dell'orologio).

Algoritmo seconda scelta

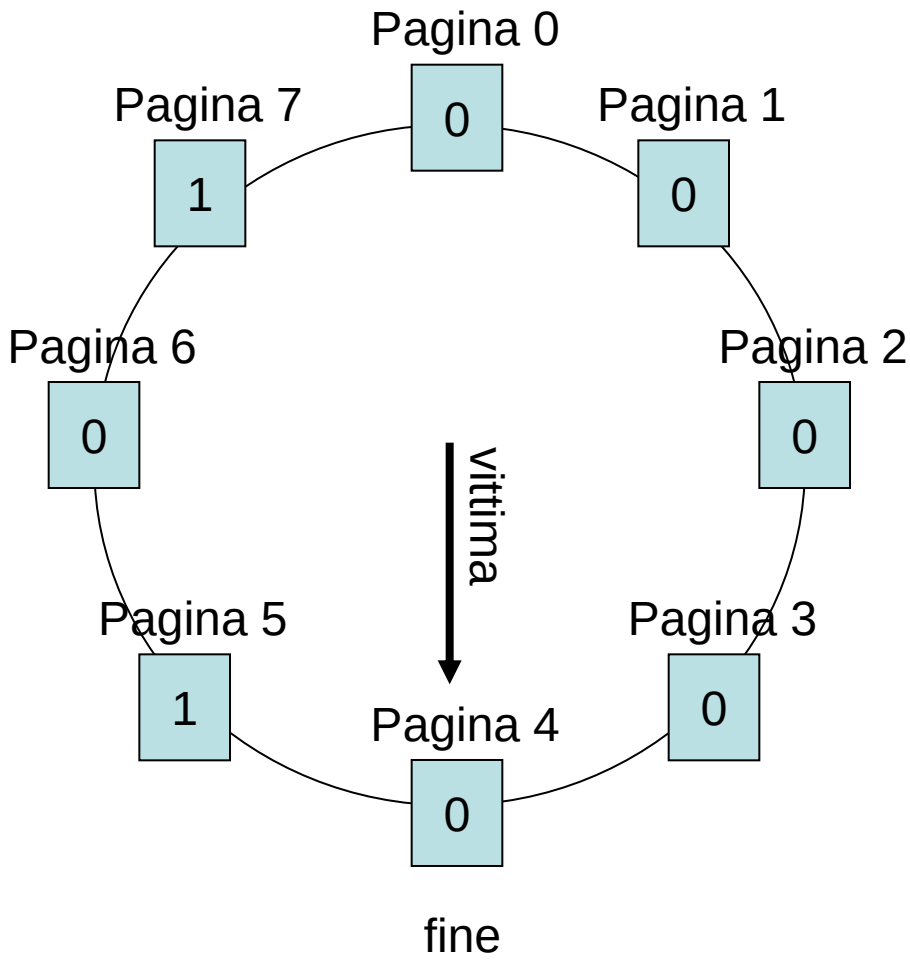
- Per consentire al gestore della memoria di fare statistiche sull'uso delle pagine si utilizzano i bit **U** (uso) e **M** (modifica) associati a ciascuna pagina. Questi due bit devono essere aggiornati ogni volta che una pagina è indirizzata; è quindi fondamentale che siano modificati via hardware.
- All'avvio di un processo, i due bit U e M di ogni pagina sono resettati (posti a 0).
- Periodicamente, ad esempio ogni 20 ms, all'interruzione di un timer, tutti i bit U sono resettati.
- Le pagine sono distinte in due classi: quelle con il bit **U=1** (le più recentemente usate) e quelle con il bit **U=0** (**le meno recentemente usate**). Si sceglie una pagina con politica FIFO, dapprima tra quelle con il bit U=0, se ci sono.

- La **tabella delle pagine fisiche** è gestita come un array circolare (round robin). Viene tenuta aggiornata una variabile di sistema ***vittima,*** contenente l'indice della pagina fisica successiva a quella che è stata rimpiazzata per ultima.
- Quando si verifica un ***page-fault*** la verifica inizia con la pagina il cui indice è contenuto nella variabile *vittima*. Se tale pagina ha il bit **U=0** è scelta per il rimpiazzamento, altrimenti il suo bit U è resettato e si prosegue fino a trovare una pagina che ha il bit U=0.
- Nella figura è mostrato il funzionamento dell'algoritmo disegnando la tabella delle pagine fisiche in forma circolare e mettendo in evidenza per ogni pagina solo il bit d'uso U.

tabella delle pagine fisiche.
Sono visualizzati **solo i bit d'uso U**



inizio



fine

Algoritmo second-chance

- Per ridurre i trasferimenti tra memoria e disco, e quindi per migliorare le prestazioni dell'algoritmo, per la classificazione delle pagine si considera anche il bit di modifica **M**. In tal modo le classi diventano 4, relativamente alle combinazione dei bit **U-M**:

  classe0: **0-0** non usata – non modificata
  classe1: **0-1** non usata- modificata
  classe2: **1-0** usata – non modificata
  classe3: **1-1** usata - modificata

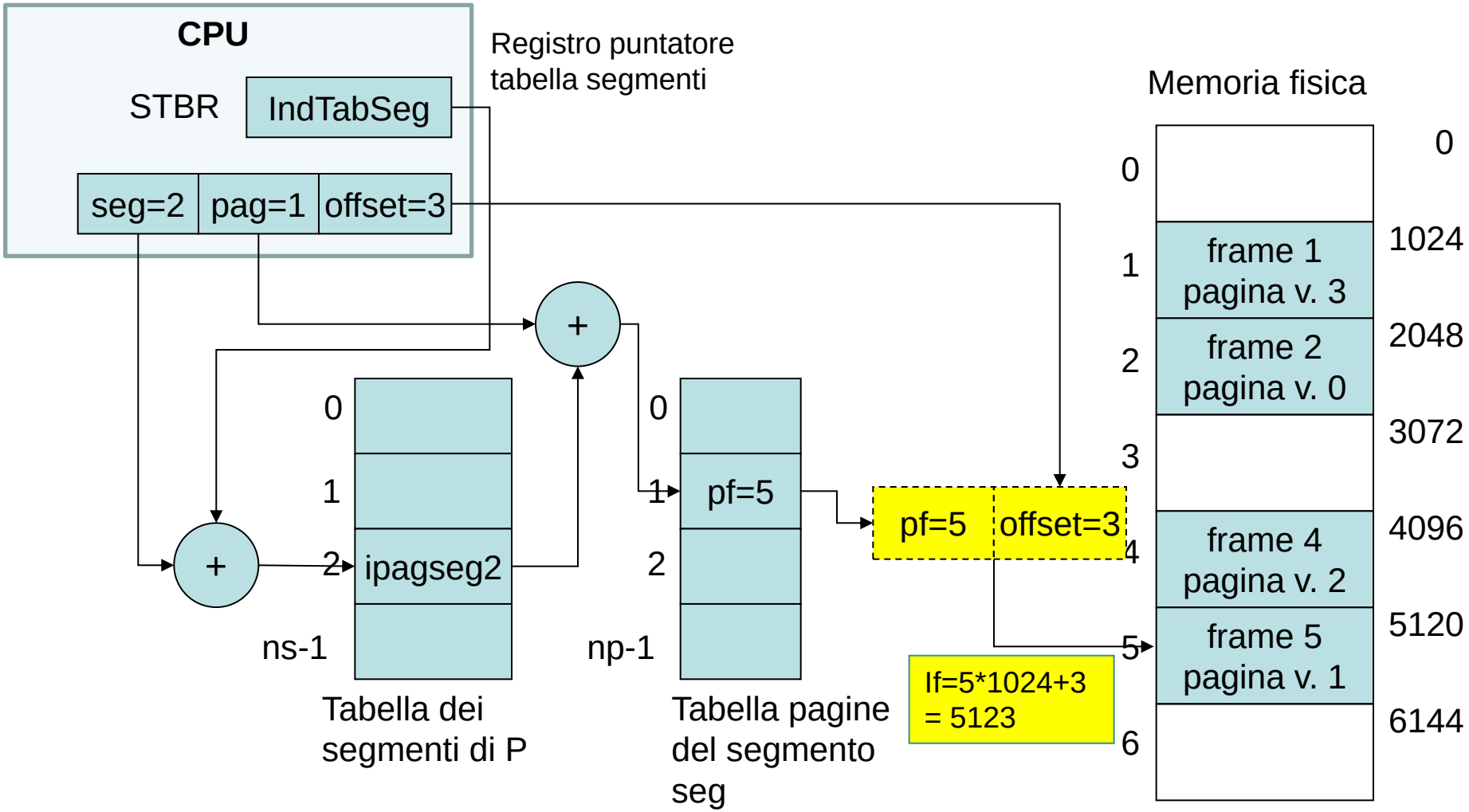
- Rispetto alla scelta basata sul solo bit **U** si privilegiano le pagine con il bit di modifica **M=0** in quanto non modificate e quindi non è necessario riscriverle sul disco.
- La classe1, contiene pagine che sono state indirizzate, M=1 ma che hanno il bit U=0 per via del reset eseguito all'interruzione del clock.

- Sono molti gli algoritmi realizzati per il rimpiazzamento.
- Ogni tipo di algoritmo ha molte varianti in base ai criteri di scelta delle pagine da rimpiazzare. Ad esempio, al momento del page-fault, si potrebbe scegliere di revocare una pagina fisica dal processo che ha generato il page-fault (tecnica del **rimpiazzamento locale**) oppure scegliere la pagina revocandola a qualsiasi processo (tecnica del **rimpiazzamento globale**). Il rimpiazzamento globale consente una scelta più conveniente, in quanto viene fatta su un numero maggiore di pagine.
- Molti sistemi, tra cui UNIX, non consentono di saturare completamente la memoria ma conservano un certo numero di pagine fisiche di riserva. In tal modo la gestione del page-fault avviene più velocemente. D'altra parte però il paginatore, quando verifica che la memoria sta per esaurirsi, deve salvare un numero di pagine in modo da renderle di nuovo libere.

- Per minimizzare il numero di page-fault, molti paginatori tengono traccia del working set di ciascun processo, in modo tale da caricarlo in memoria prima che il processo riprenda l’esecuzione. Questa tecnica è detta **working set model.**

Memoria segmentata e paginata

- Questa tecnica offre i vantaggi della segmentazione e della paginazione.
- Lo spazio virtuale è suddiviso in segmenti che sono divisi in pagine virtuali che saranno allocate in memoria con la tecnica della paginazione.
- Nella figura seguente è mostrato uno schema semplificato della soluzione adottata nel sistema operativo Multics che per primo ha adottato questa tecnica.



Traduzione degli indirizzi nella memoria segmentata e paginata

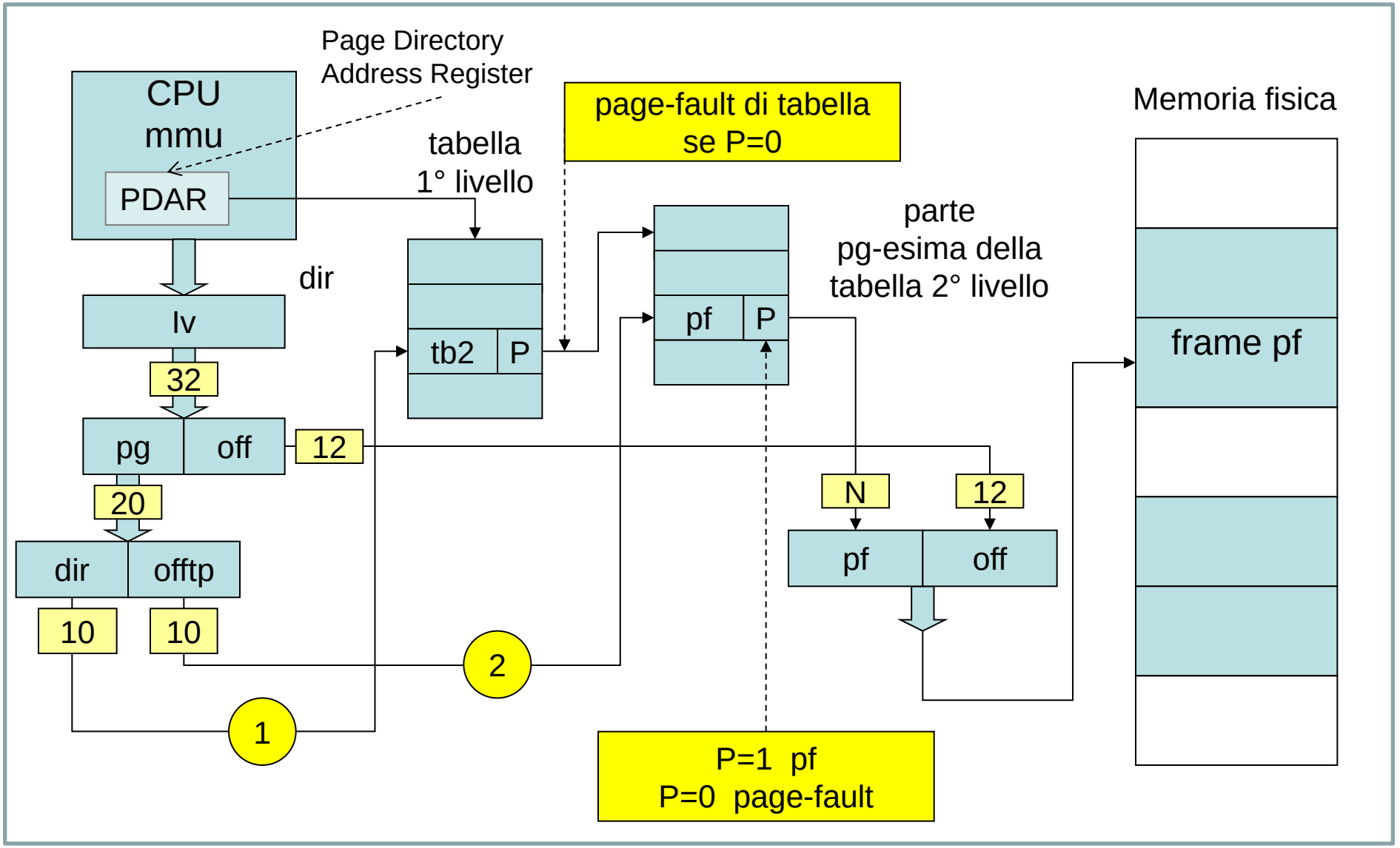
Gestione degli spazi virtuali

- I SO moderni hanno spazi virtuali di grandi dimensioni pertanto anche le tabelle di traduzione degli indirizzi sono paginate, in modo di allocarle in memoria, su richiesta invece che caricarle permanentemente in memoria.

Paginazione a più livelli

- La figura mostra lo schema usato nei microprocessori Intel con indirizzamento a 32 bit. L'indirizzo virtuale a 32 bit (4.294.967.296 (4G) indirizzi) è diviso in due parti:
  - Pagina virtuale **pg** a 20 bit (1.048.576 pagine (1 MB)
  - Scostamento **off** nella pagina virtuale a 12 bit (meno significativi) (4096 (4K) byte dimensione pagina)

- L'elemento della tabella delle pagine è di 4 byte (32 bit), pertanto la dimensione massima della tabella è di 4 MB, essendo composta da 2^{20} elementi di 4 byte (32 bit) ciascuno.
- La tabella è suddivisa in 2^{10} (1024) partizioni consecutive.
- Ciascuna partizione contiene 2^{10} (1024) elementi della tabella ed occupa quindi 4KB (4096) che è la dimensione della pagina fisica.



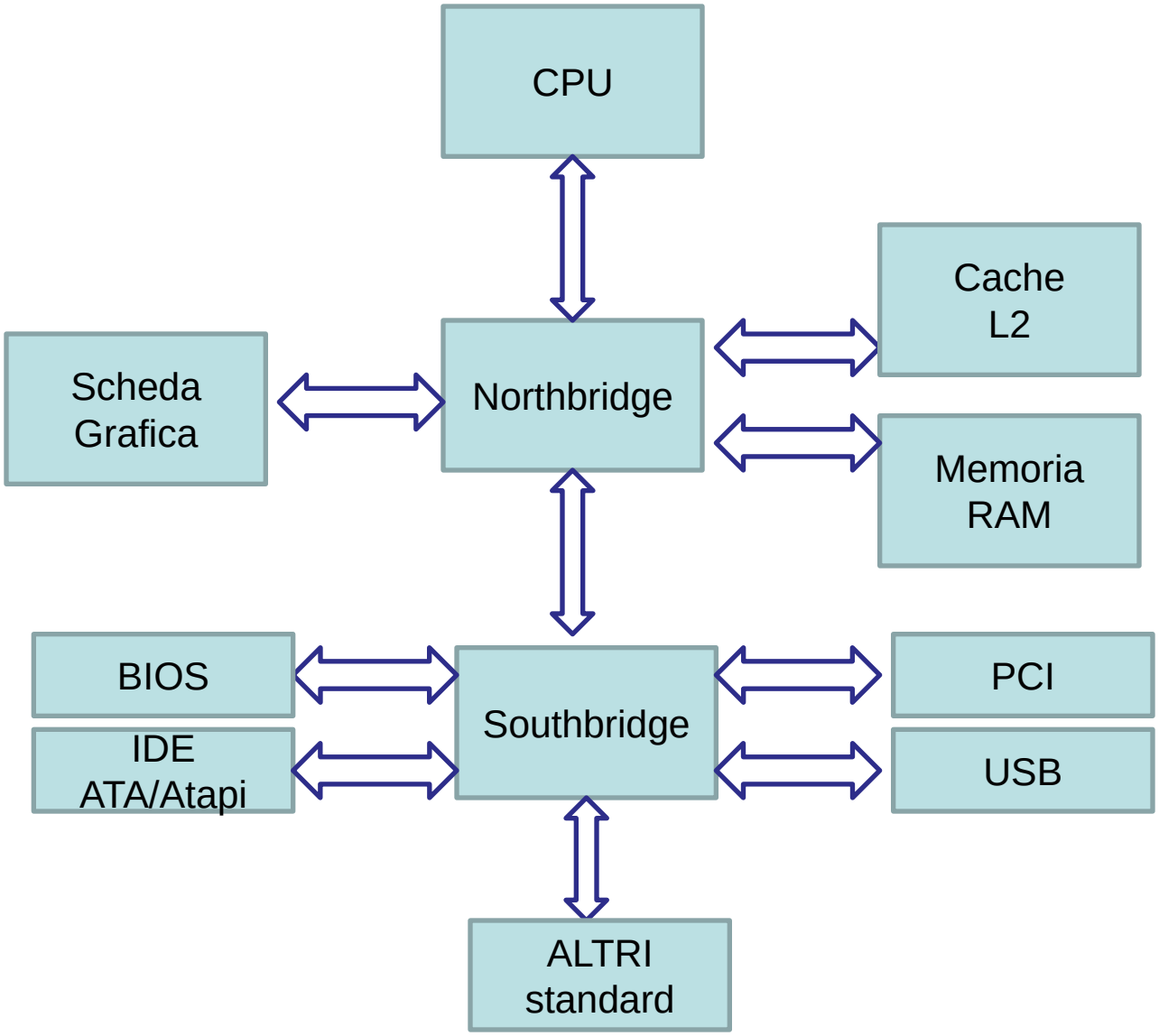
Paginazione a più livelli

- La tabella di 1° livello deve essere garantita in memoria quando il processo è in esecuzione.

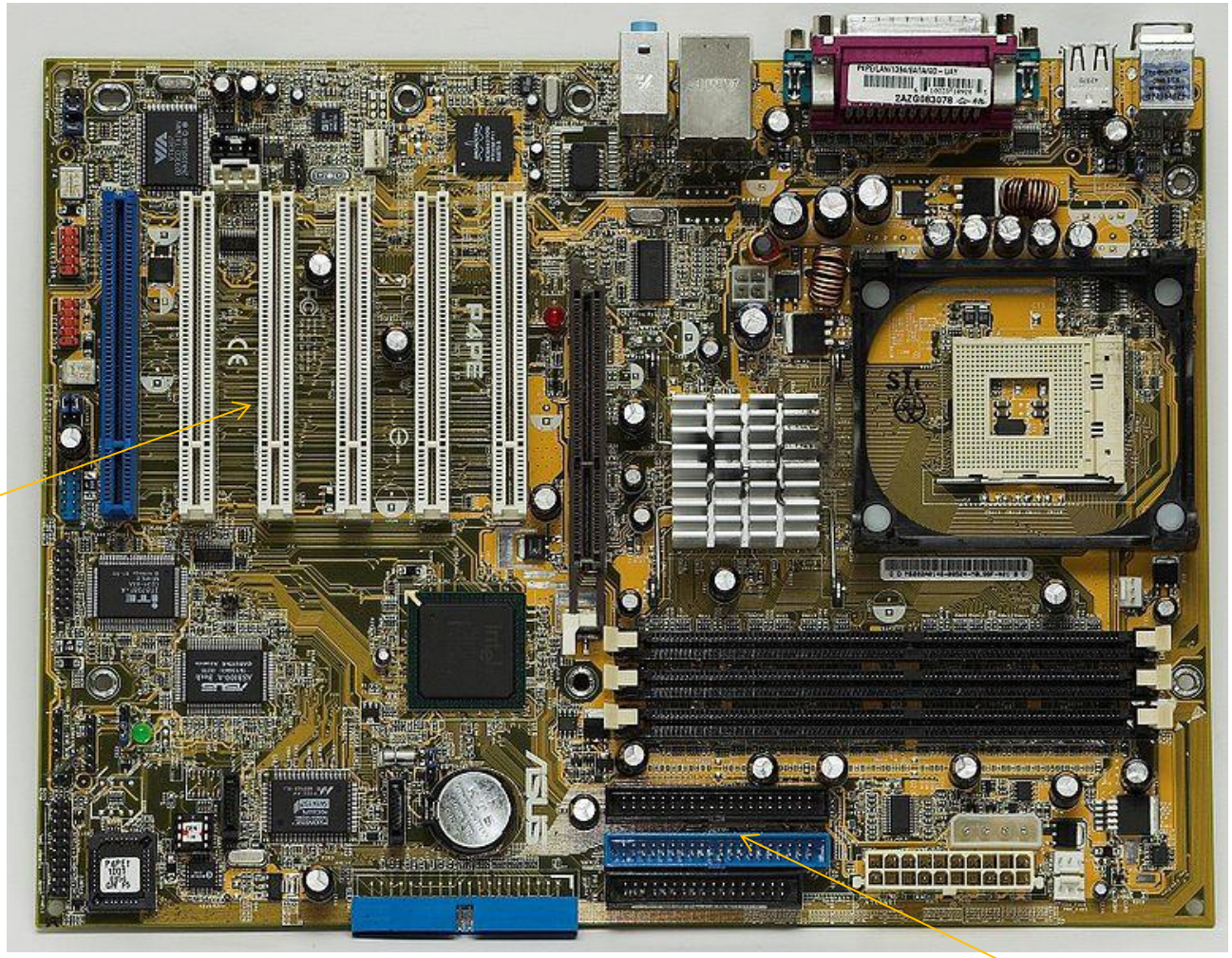
Gestione dell'I/O

- Le operazioni svolte da un processo possono essere classificate in due classi:
  - **Operazioni di I/O**
  - **Operazioni di computazione**
- Spesso, anche i processi applicativi sono classificati in base alla prevalenza di un tipo di operazione rispetto all'altra:
  - **Processi I/O-bound** (se le operazioni di I/O sono prevalenti)
  - **Processi CPU-bound** (se le operazioni di computazione sono prevalenti)

- La figura seguente mostra l'architettura semplificata di un calcolatore, in cui sono evidenziati alcuni moduli principali.



Architettura di un moderno computer



Scheda madre di un computer

Vedi dispositivi

- Ogni dispositivo è collegato ai bus di sistema tramite un interfaccia hardware (***controller***). Il controller ha il compito di gestire e controllare il dispositivo e comunica con esso tramite un proprio protocollo. Il controller è dotato di registri, che la CPU vede come locazioni di memoria (memory-mapped) o indirizza con particolari istruzioni di I/O, tramite i quali avvengono le operazioni di lettura e scrittura dei dati.
- Per comunicare con i registri del controller a basso livello, usando l'assembly sarebbe necessario conoscere l'insieme dei registri (e in dettaglio le loro funzioni) contenuti nel controller per poter gestire il dispositivo.
- Un compito fondamentale del sottosistema di I/O è fornire **un'astrazione dei dispositivi**per evitare che i programmatori debbano conoscere in dettaglio il funzionamento hardware dei dispositivi.
- Ciascuna periferica viene vista dai processi applicativi come una **risorsa astratta** alla quale è possibile accedere tramite un insieme di funzioni le **I/O API – Input Output Application Programming Interface**.

Classificazione dei dispositivi

- Le velocità con cui i dispositivi effettuano il trasferimento di dati sono molto diverse, anche di molti ordini di grandezza.

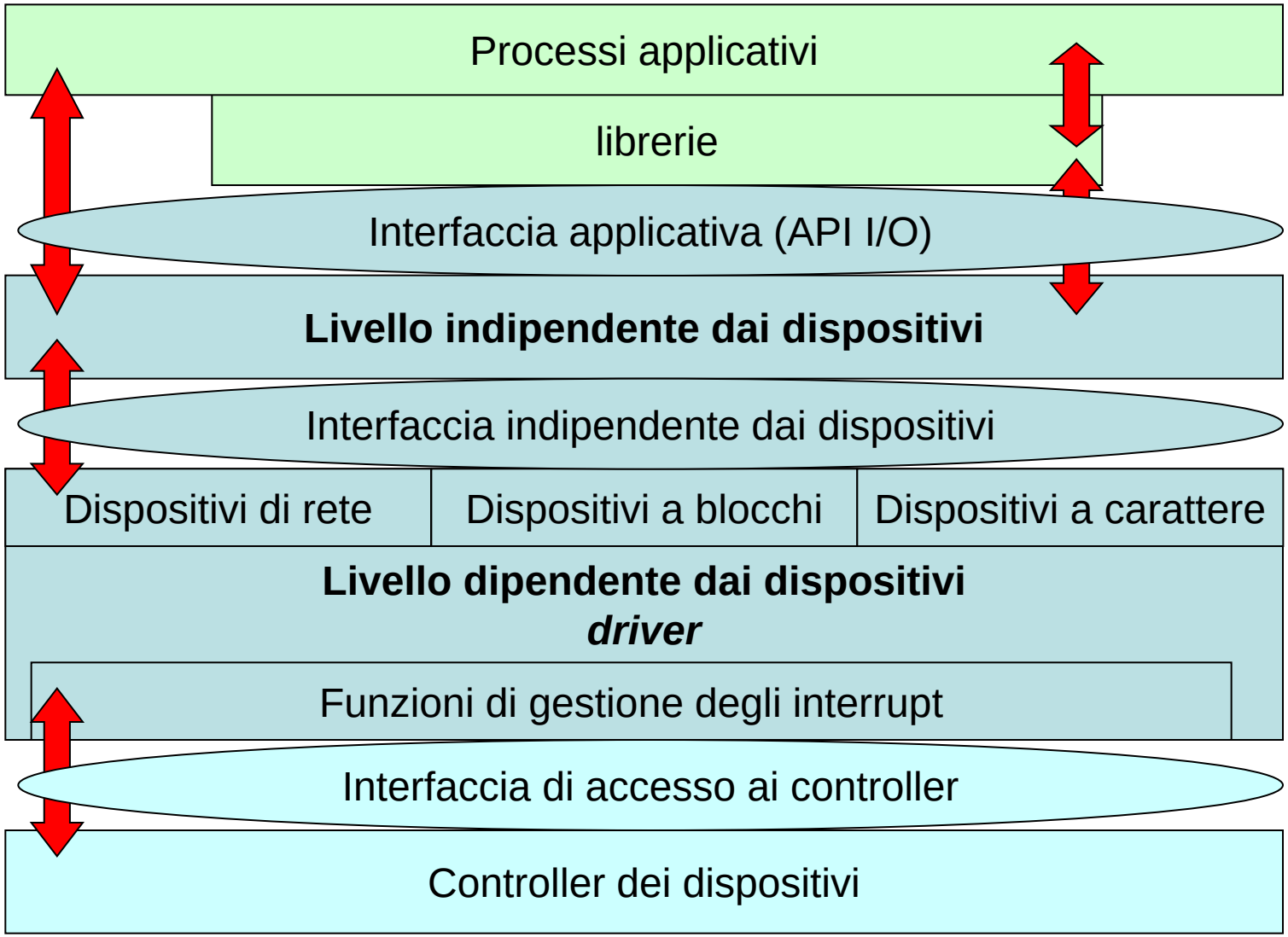
Dispositivo	Velocità di trasferimento
Tastiera	10 B/s
Mouse	100 B/s
Modem 56 Kbit/s	7 KB/s
Scanner	400 KB/s
802.11g wireless	6,75 MB/s
Cd-rom 52x	7,8 MB/s
Fast ethernet	12,5 MB/s
USB 2.0	60 MB/s
Disco SCSI ultra 2	80 MB/s
Gigabit ethernet	125 MB/s
Disco SATA 2.0	300 MB/s
Bus PCI	528 MB/s
Disco SATA 3.0	600 MB/s
USB 3.0	600 MB/s

- I dispositivi possono essere classificati in base a vari criteri, uno dei quali, particolarmente utile ai fini della loro gestione, è relativo al modo in cui sono organizzati i dati da trasferire:

  - **Dispositivi a blocchi**
    dispositivi di memoria secondaria (dischi, cd, pennette usb..) che memorizzano i dati in blocchi di dimensione fissa. Ogni blocco ha un proprio indirizzo e può quindi essere letto o scritto indipendentemente dagli altri.
  - **Dispositivi a carattere**
    i dati consistono in sequenze di byte (mouse, tastiere, stampanti, schede di rete..)
  - **Dispositivi speciali**
    dispositivi che non rientrano nelle precedenti classi, come ad esempio il timer che genera segnali di interruzione.

- A prescindere dalle tipologie di dispositivi, il sottosistema di I/O deve fornire ai processi applicativi un insieme di funzioni ad alto livello come **read** e **write**, **open** e **close** ecc.
- Ad esempio, quando un processo esegue un operazione di lettura/scrittura su un file è opportuno che il codice che il processo esegue non cambi al variare del supporto su cui il file è memorizzato.
- Il sottosistema di I/O deve anche gestire tutti i tipi di **eccezione** che si possono verificare nel trasferimento di dati.
- Un altro importante compito che deve svolgere è il **naming**, che consiste nell'identificare i dispositivi mediante nomi logici in modo tale che i processi possono fare riferimento ad essi semplicemente tramite il nome.

Struttura logica del sottosistema di I/O



Livello indipendente dai dispositivi

- Alcune funzioni del sottosistema di I/O sono indipendenti dai dispositivi.
- I servizi offerti da questo livello rendono più semplice, efficiente ed affidabile l'uso dei dispositivi.
- I progettisti tendono a rendere uniforme l'interfaccia applicativa del file system a quella dei dispositivi. Ad esempio nel sistema UNIX, un dispositivo è visto dai processi applicativi, come un file (speciale). Pertanto i comandi:
  - **cp miofile.txt lettere/miofile2.txt**
    crea una copia di miofile.txt nella directory lettere assegnandogli il nome miofile2.txt
  - **cp miofile.txt /dev/lp**
    stampa il file miofile.txt in quanto **lp** è un **file speciale** associato ad una stampante.

- Appartengono a questo livello anche le funzioni di **naming** dei dispositivi e dei file. La funzione di naming dei dispositivi ha il compito di identificare i dispositivi mediante un nome simbolico. Ad esempio in Windows, **c:** indica la directory radice di un disco indicato con la lettera c:. In unix il simbolo **/** indica la *root* dell'intero file system.

Bufferizzazione

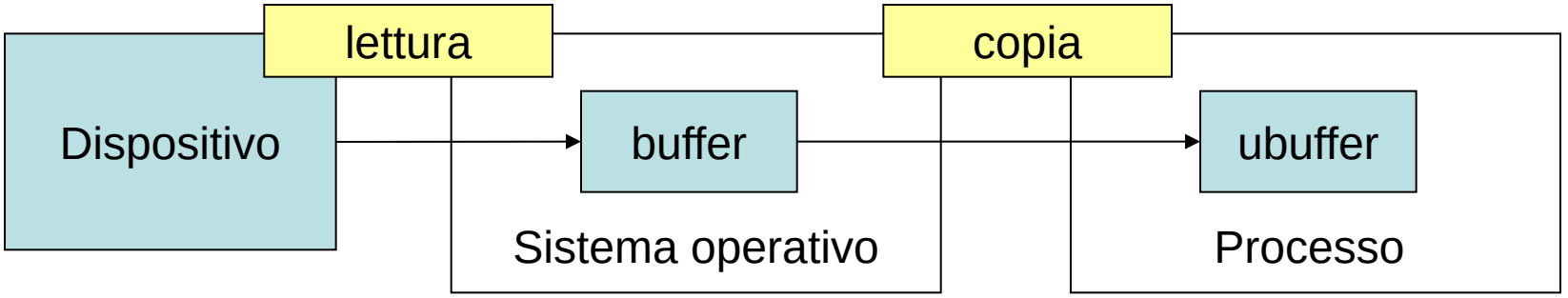
- Un'importante funzione del livello indipendente dai dispositivi è la funzione di bufferizzazione (buffering), che consiste nell'utilizzare un area di memoria gestita dal kernel detta ***buffer di sistema*** per effettuare il trasferimento dei dati, tra il dispositivo e l'area di memoria virtuale del processo applicativo.
- La bufferizzazione serve principalmente per far fronte alle notevoli differenze di velocità di funzionamento con cui operano il processore e i dispositivi.

- Consideriamo il caso di dispositivi a blocchi, come ad esempio i dischi. Supponiamo che un processo **P** esegua una chiamata di sistema di I/O del tipo:

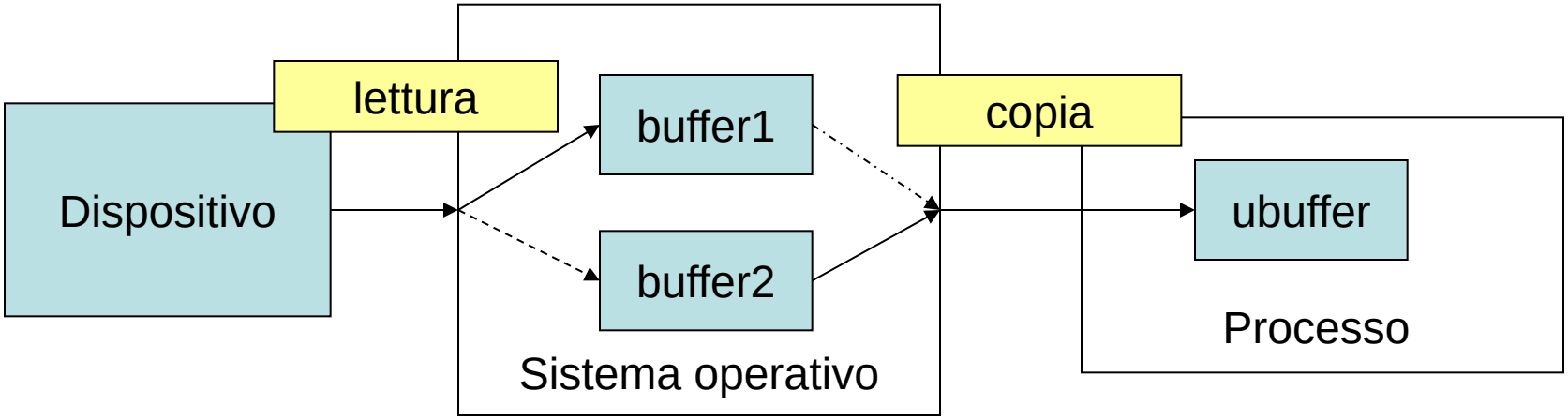
  n = read(disp, userbuffer, numbyte);

  dove, **disp** specifica il dispositivo, **userbuffer** è l'indirizzo del buffer del processo e **numbyte** indica il numero di byte che devono essere trasferiti.

  Il processo **P** eseguendo una chiamata di sistema di I/O passa nello stato di bloccato, fino al termine della chiamata che trasferirà in **userbuffer** i ***numbyte*** richiesti.
- Se la funzione **read** viene realizzata in modo che utilizzi un buffer di sistema come indicato nella figura seguente, l'operazione di lettura si realizza trasferendo i dati da disco al **buffer di sistema** e quindi da questo al buffer ***userbuffer*** dello spazio virtuale del processo.



Operazione di lettura con un solo buffer



Operazione di lettura con doppio buffer

- Al termine del trasferimento il processo può riprendere la sua esecuzione. Il vantaggio di questa soluzione si ha quando, come spesso avviene, il processo deve leggere ed elaborare una sequenza di blocchi.
- In questo caso, il tempo di attesa del processo, e il numero di commutazioni di contesto, si riduce se l'elaborazione di un blocco di dati da parte del processo viene eseguita in parallelo alla lettura nel buffer del successivo blocco di dati (read ahead).
- Un miglioramento del parallelismo si ha con il doppio buffering.