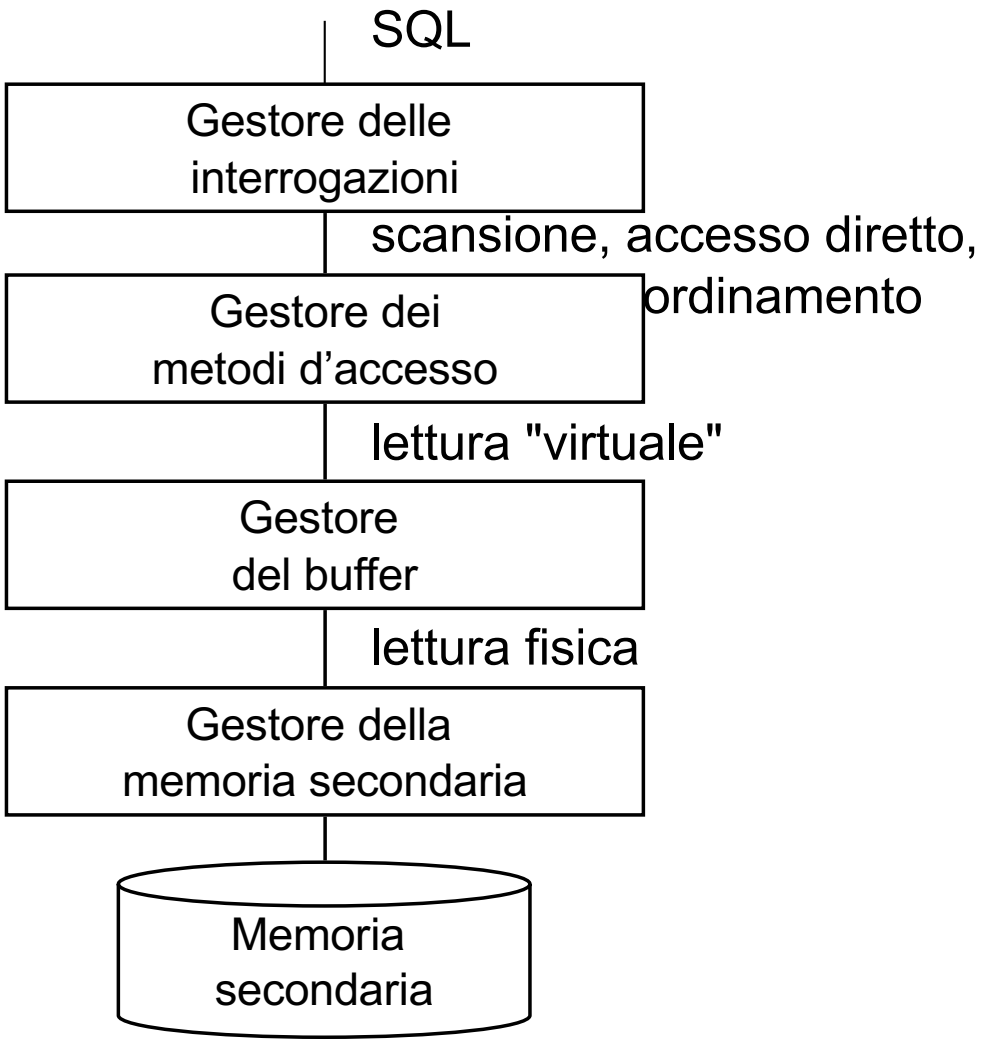
Basi dati II

Organizzazione fisica dei dati

Cap. 11 Basi di dati 5ed. Atzeni et al.
Cap. 7 Sistemi di Gestione dati- Catania et al.

Gestore degli accessi e delle interrogazioni



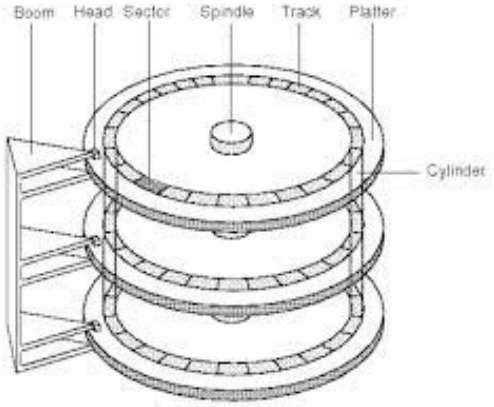
Memoria principale e secondaria

- Le basi di dati debbono essere (sostanzialmente) in memoria secondaria per due motivi:
  - dimensioni
  - persistenza
- I programmi possono fare riferimento solo a dati in memoria principale
- I dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale (questo spiega i termini "principale" e "secondaria")

Memoria principale e secondaria, 2

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di lunghezza (di solito) **fissa** (ordine di grandezza: alcuni KB)
- Le uniche operazioni sui dispositivi sono la lettura e la scrittura di un **intero blocco**
- Accesso a memoria secondaria:
  - tempo di posizionamento della testina
  - tempo di latenza
  - tempo di trasferimento

  in media 10 ms (hard disk 7200 rpm)

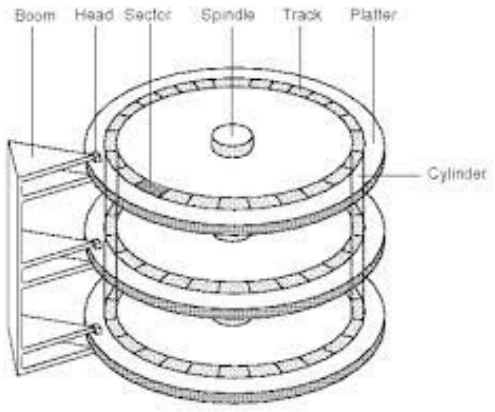


Il blocco rappresenta l'unità di trasferimento dati tra memoria secondaria e memoria principale

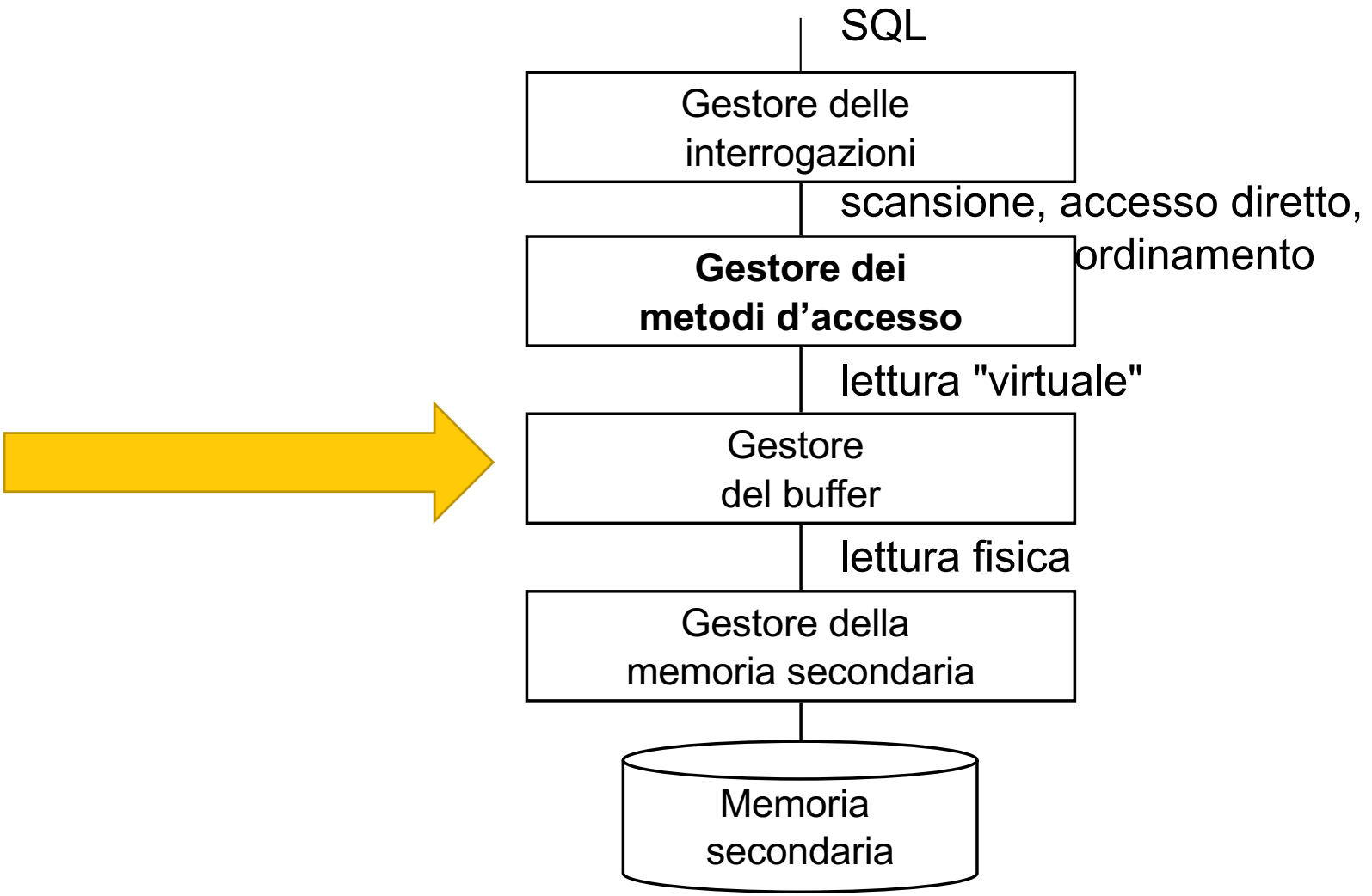
Memoria principale e secondaria, 3

- Il costo di un accesso a memoria secondaria è quattro o più ordini di grandezza maggiore di quello per operazioni in memoria centrale
- Perciò, nelle applicazioni "I/O bound" (cioè con molti accessi a memoria secondaria e relativamente poche operazioni) il costo dipende esclusivamente dal numero di accessi a memoria secondaria
- Inoltre, accessi a blocchi “vicini” costano meno (contiguità)

Come migliorare? **BUFFER**



Gestore degli accessi e delle interrogazioni



Buffer management

- **Buffer**:
  - area di memoria centrale, gestita dal DBMS (preallocata) e condivisa fra le **transazioni/programmi**
  - organizzato in **pagine** di dimensioni pari o multiple di quelle dei blocchi di memoria secondaria
  - è importantissimo per via della grande differenza di tempo di accesso fra memoria centrale e memoria secondaria

Scopo della gestione del buffer

- Ridurre il numero di accessi alla memoria secondaria
  - In caso di lettura, se la pagina è già presente nel buffer, non è necessario accedere alla memoria secondaria
  - In caso di scrittura, il gestore del buffer può decidere di differire la scrittura fisica
- Decidere quali pagine tenere nel buffer:
  - Massimizzare le probabilità che la pagina richiesta sia già nel buffer
  - Se necessario liberare pagine dal buffer, eliminare quelle che creano minor danno (aka che non saranno da ricaricare a breve)
- Le politiche di gestione del buffer sono simili a quelle relative alla gestione della memoria da parte dei sistemi operativi;
  - "località dei dati": è alta la probabilità di dover riutilizzare i dati attualmente in uso
  - "legge 80-20": l'80% delle operazioni utilizza sempre lo stesso 20% dei dati

Gestore del buffer

- Per gestire il buffer, il gestore mantiene un direttorio dove, per ogni pagina, mantiene
  - informazioni sul file fisico e il numero del blocco
  - due variabili di stato:
    - **Pin:** un contatore che indica quanti programmi utilizzano la pagina. Se il contatore è 0, la pagine è libera (unpinned).
    - **Dirty:** un bit che indica se la pagina è stata modificata

Funzioni del buffer manager

- Intuitivamente:
  - riceve richieste di lettura e scrittura (di pagine)
  - le esegue accedendo alla memoria secondaria solo quando indispensabile e utilizzando invece il buffer quando possibile
- Più formalmente, riceve richieste da transazioni attraverso:
  - ***fix***, ***unfix***, ***setDirty***, ***force***.

Interfaccia offerta dal buffer manager

- ***fix:*** richiesta di un blocco; richiede una lettura su memoria secondaria solo se il blocco non è nel buffer (incrementa il contatore associato alla pagina)
- ***setDirty:*** comunica al buffer manager che la pagina è stata modificata
- ***unfix:*** indica che la transazione ha concluso l'utilizzo della pagina (decrementa il contatore associato alla pagina)
- ***force***: trasferisce in modo sincrono una pagina in memoria secondaria

Esecuzione della fix

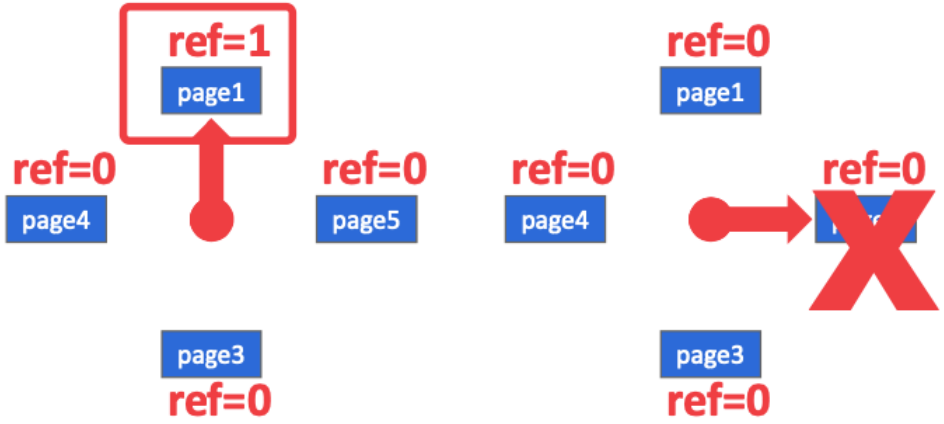
- Cerca il blocco nel buffer
  - se il blocco è già in una pagina del buffer, incrementa il contatore e restituisce l'indirizzo
  - altrimenti:
    - cerca una pagina unpinned nel buffer (contatore a zero);
      - tra queste ne sceglie una, seguendo diverse strategie (replacement policy);
      - se pagina selezionata è stata modificata/dirty
        - viene aggiornata in memoria secondaria (flush)
      - restituisce l'indirizzo
    - altrimenti, due alternative
      - **steal**: seleziona una "vittima ", pagina occupata del buffer (**replacement policy**);
        - I dati della vittima sono scritti in memoria secondaria (flush); viene caricata la pagina di interesse dalla memoria secondaria e si restituisce l'indirizzo
      - **no-steal**: la transazione che ha richiesto la pagina viene posta in attesa

Replacement policy

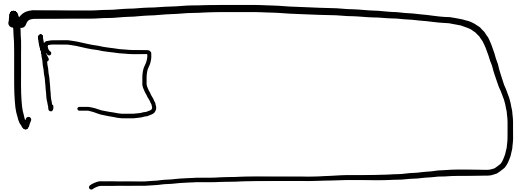
- LRU- Least recently used.
  - Si scarica la pagina che è stata usata meno di recente. Questo richiede di memorizzare per ogni pagina l'orario dell'ultimo accesso.
    - ++ la prob. di accesso ad una pagina non utilizzata da tempo è bassa
- FIFO – first in first out.
  - Si scarica la pagina che è da più tempo nel buffer
    - -- Un blocco che rimane tanto tempo nel buffer è perchè è richiesto. Si rischia di scaricare la pagina per poi doverla ricaricare a breve.

Replacement policy

- Clock replacement (politica dell'orologio)
  - Come LRU, ma si controllano le pagine in modo circolare (come round robin), tenendo in considerazione gli ultimi accessi alle pagine
- MRU - Most recent used.
  - Si scarica la pagina che è stata utilizzata più recentemente



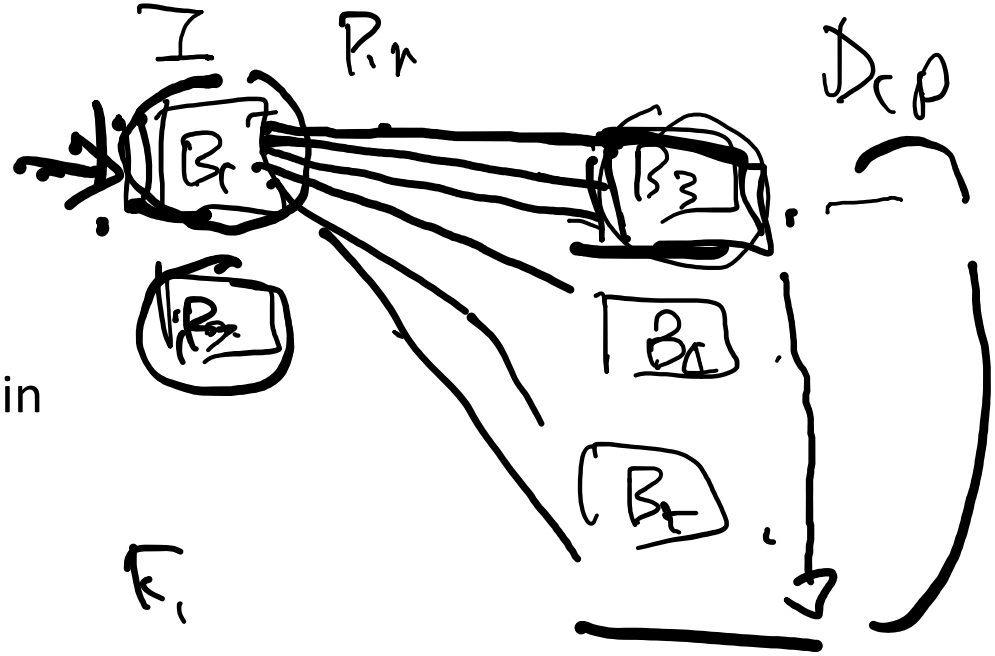
Replacement policy: Esempio



- Operazione di join Impiegati |x| Dipartimenti (assumendo che le relazioni siano in due file diversi)

Nested Loop Join

for each tuple I in IMPIEGATI **do**
  for each tuple D in DIPARTIMENTO$_s$ **do**
    if I.dip=D.ID
      add them in the result of the join
  end
end



- Relazione Impiegati:
  - una volta che una tupla della relazione è stata usata non è più necessaria
  - non appena tutte le tuple di un blocco sono state esaminate il blocco non serve più (strategia toss immediate)

15

Replacement policy: Esempio

- Operazione di join Impiegati |x| Dipartimenti (assumendo che le relazioni siano in due file diversi)

Nested Loop Join

for each tuple I in IMPIEGATI **do**
  for each tuple D in $DIPARTIMENTO_s$ **do**
    if I.dip#=D.dip#
      add them in the result of the join
  end
end

- Relazione Dipartimenti
  - il blocco più recentemente acceduto sarà riferito di nuovo solo dopo che tutti gli altri blocchi saranno stati esaminati
  - la strategia migliore per il file Dipartimenti è di rimuovere l'ultimo blocco esaminato (strategia most recently used MRU)

16

strategie: Force/No-force

- Il buffer manager richiede scritture in due contesti diversi:
  - in modo **sincrono** quando è richiesto esplicitamente con una force
  - in modo **asincrono** quando lo ritiene opportuno (o necessario, flush);

Strategia Force/no force:

- Force: tutte le pagine coinvolte da una transazione attiva vengono scritte in memoria di massa appena essa fa commit
- No-force: ci si affida al flush per scrivere la pagine di transazioni che hanno fatto commit

Altre ottimizzazioni

- Si può decidere di anticipare o posticipare scritture delle pagine (no-force policy) per coordinarle e/o sfruttare la disponibilità dei dispositivi:
  - Pre-flushing. Scaricamento anticipato delle pagine libere che sono state modificate nel corso del loro utilizzo (bit di stato con valore dirty).
  - pre-fetching. Si può anticipare anche i tempi di caricamento rispetto alle richieste delle transazioni, in quei casi in cui sono note a priori le modalità di accesso alle pagine della base di dati da parte di una transazione
- Osservazione: una pagina utilizzata da molte applicazioni può restare a lungo nel buffer, subendo varie modifiche, e venire trascritta in memoria secondaria con una sola operazione di scrittura.