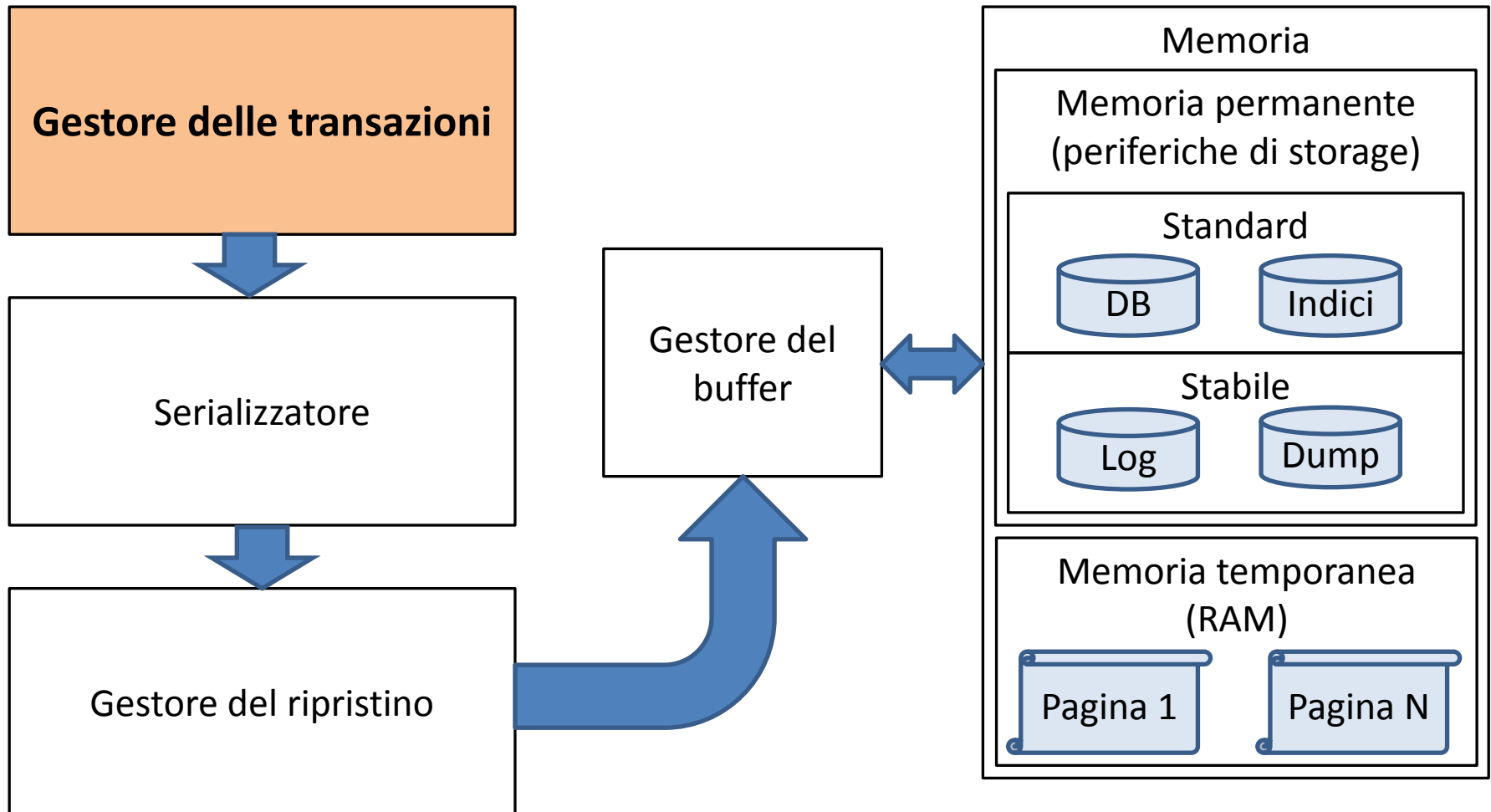


Basi di Dati
Architettura dei DBMS:
i metodi di accesso

Corso B

Architettura dei DBMS



Il concetto di transazione

Una transazione è una **unità di programma**

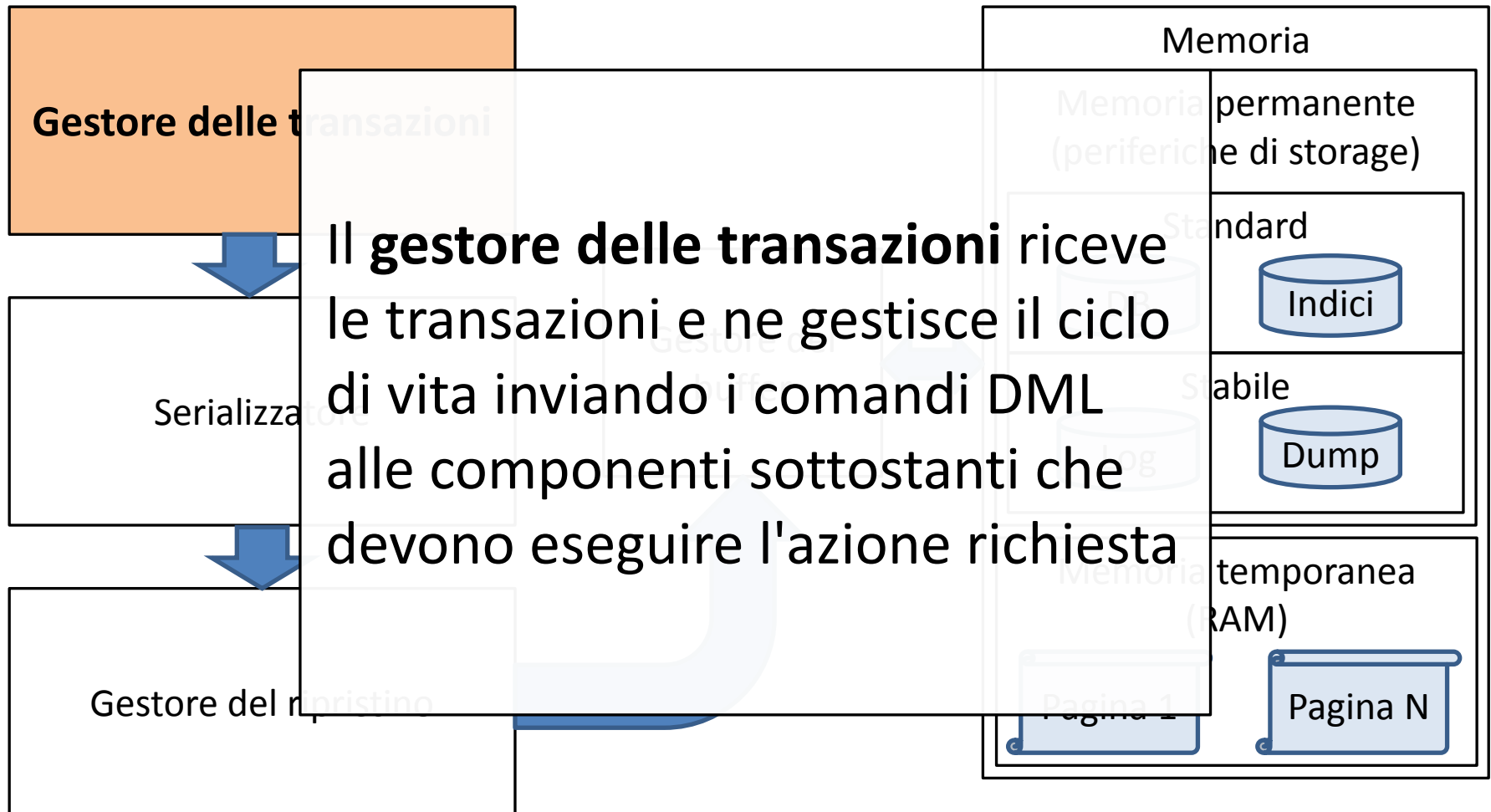
- Una transazione ha un **begin transaction** che comunica al DBMS la richiesta di interazione con esso da parte dell'applicazione
- Il DBMS identifica l'inizio della transazione T_i e la abbina in modo univoco con l'utente/applicazione che ne ha fatto richiesta
- Il DBMS, nell'ambito di T_i , riceve dei comandi DML in sequenza e le abbina alla transazione

Proprietà delle transazioni

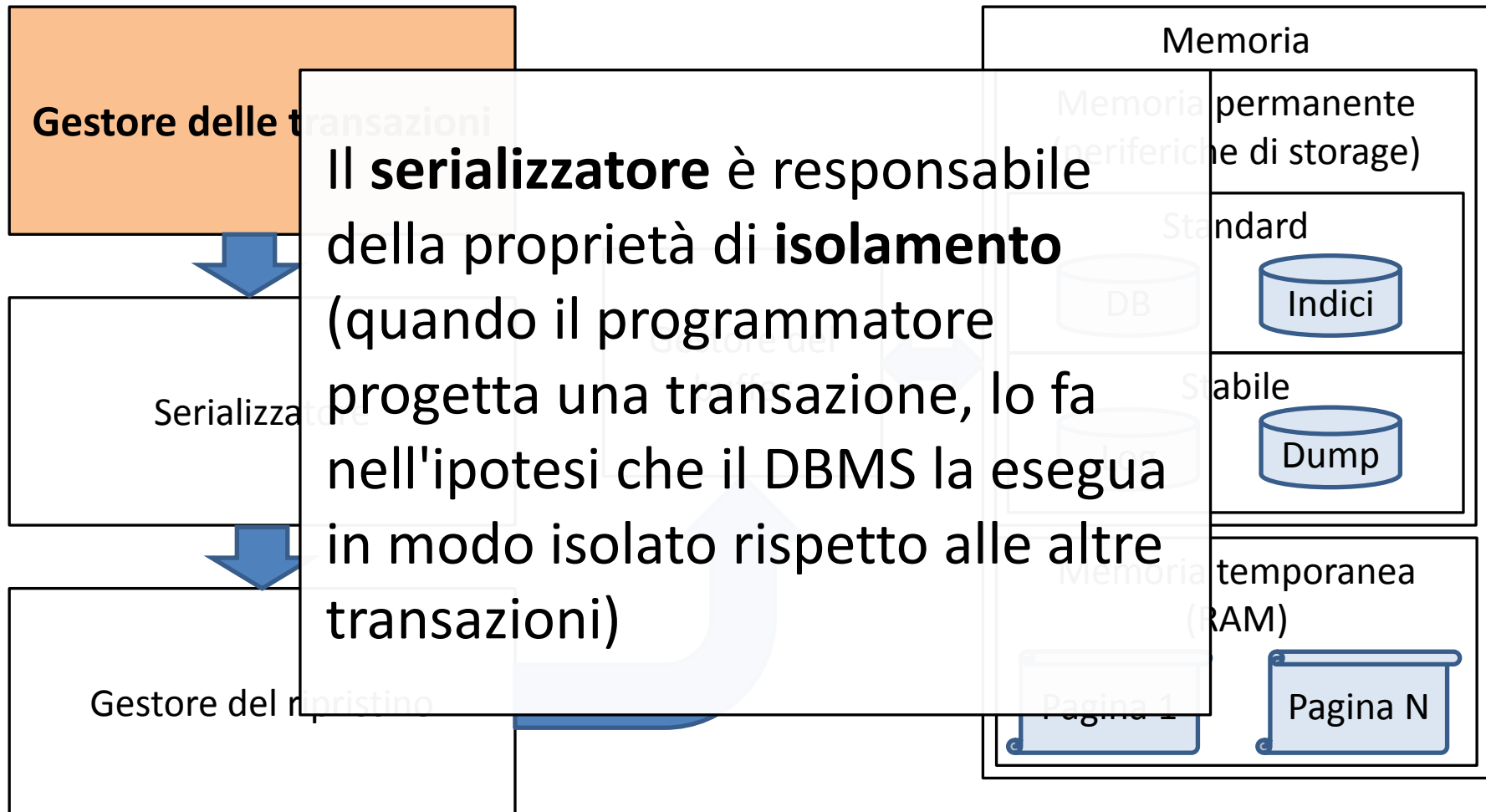
I DBMS gestiscono le transazioni garantendo, per ogni transazione T_i , il soddisfacimento delle proprietà ACID

- Atomicità
- Consistenza
- Isolamento
- Durabilità

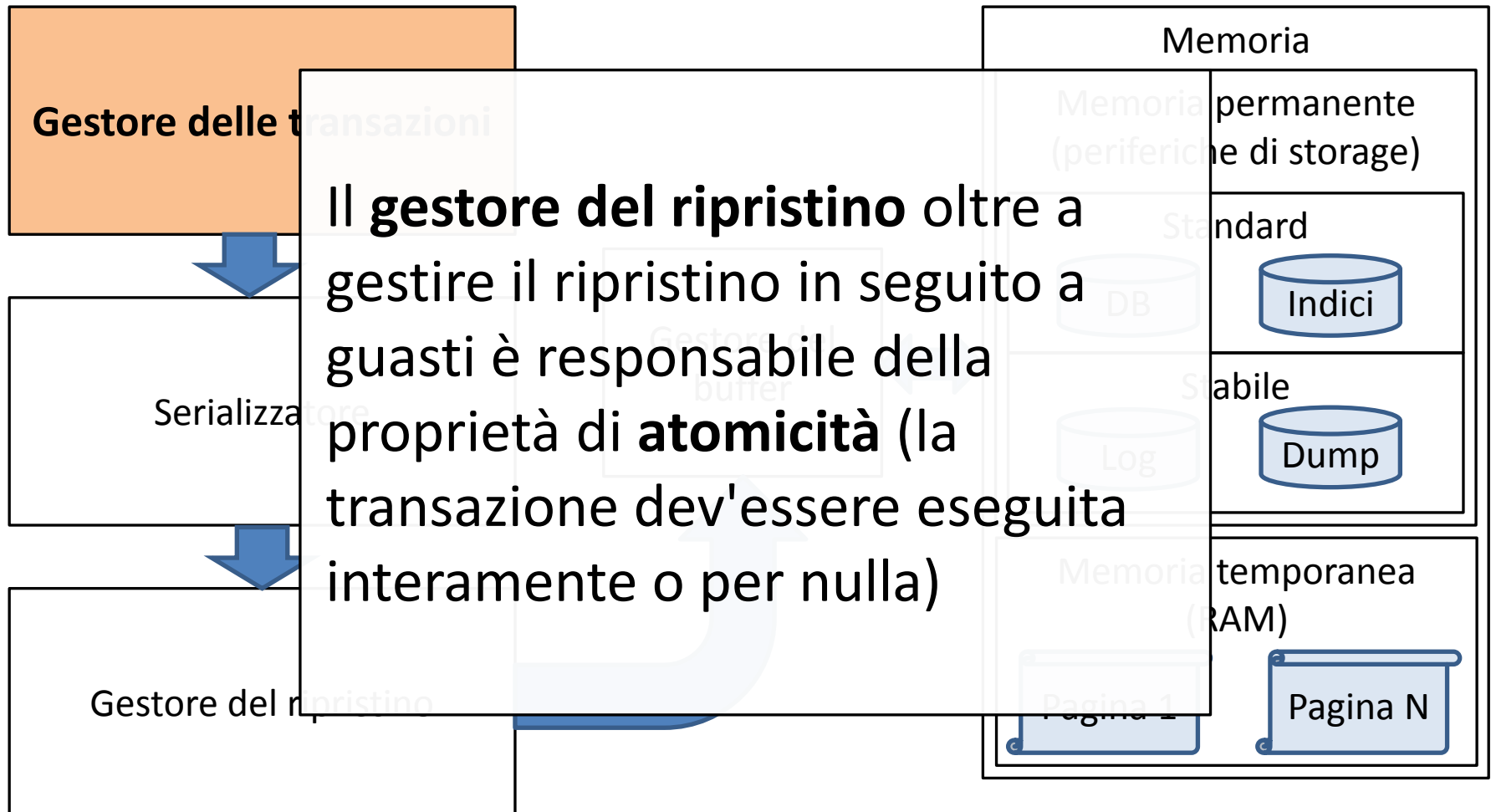
Architettura dei DBMS



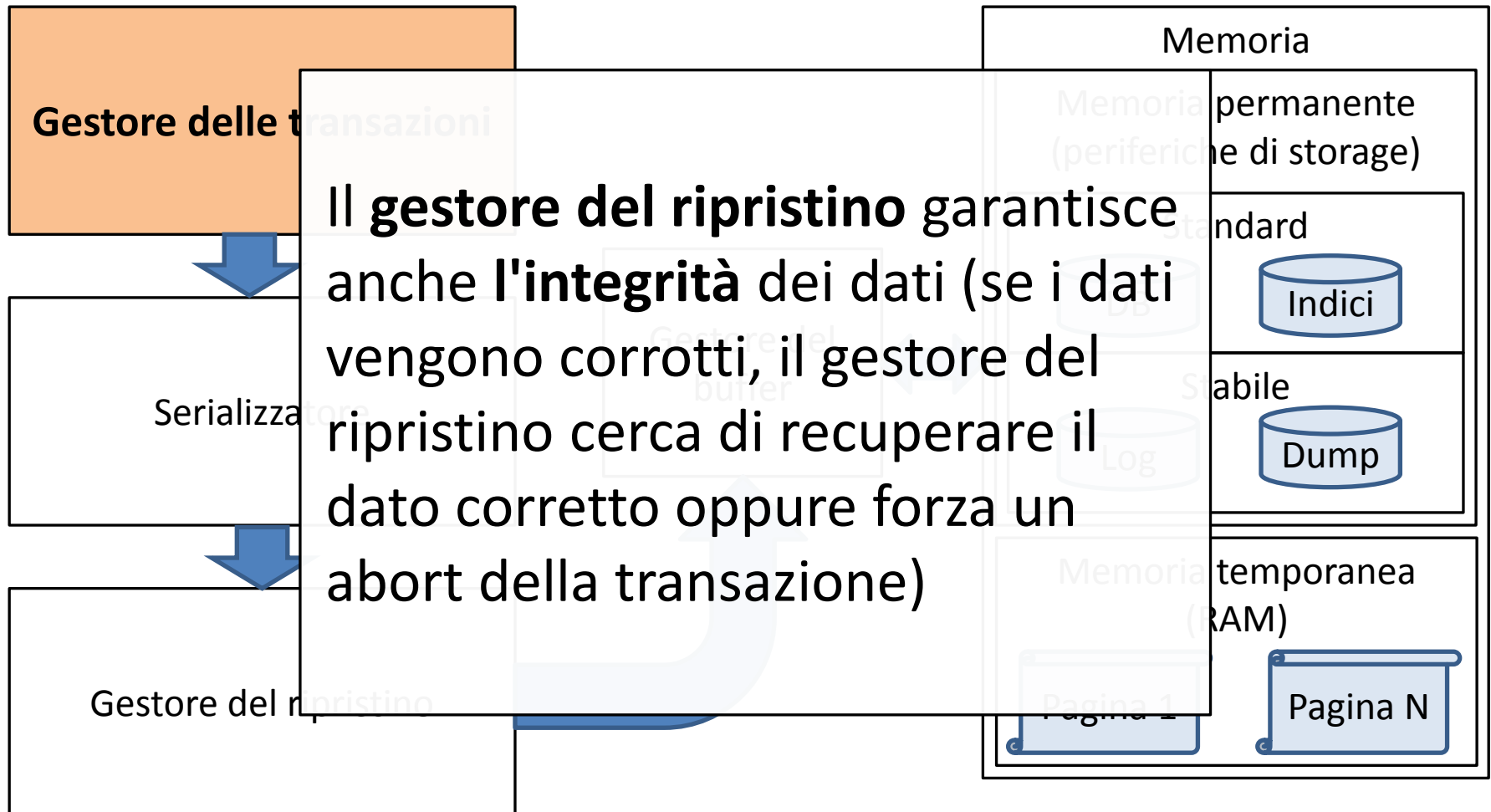
Architettura dei DBMS



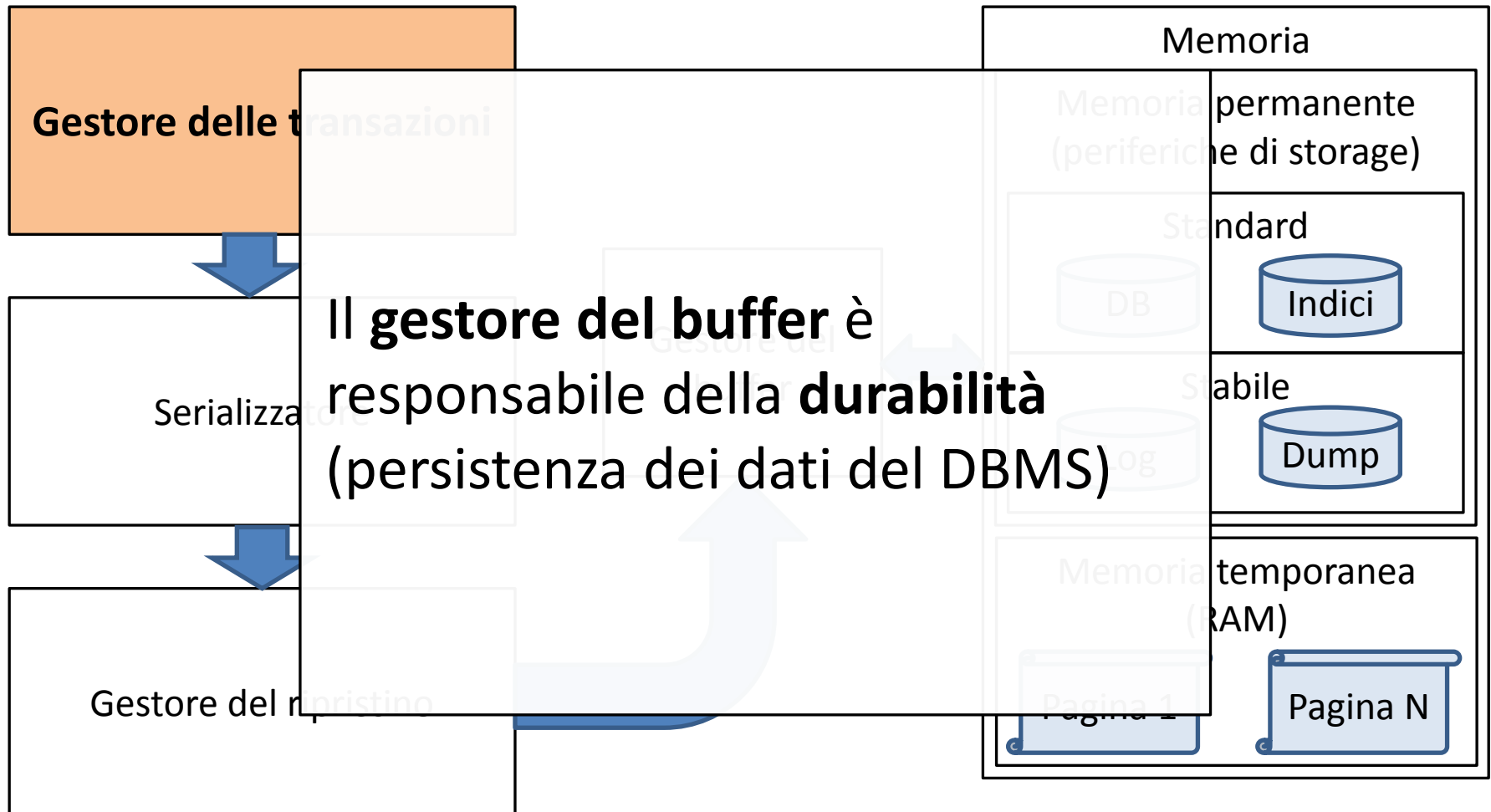
Architettura dei DBMS



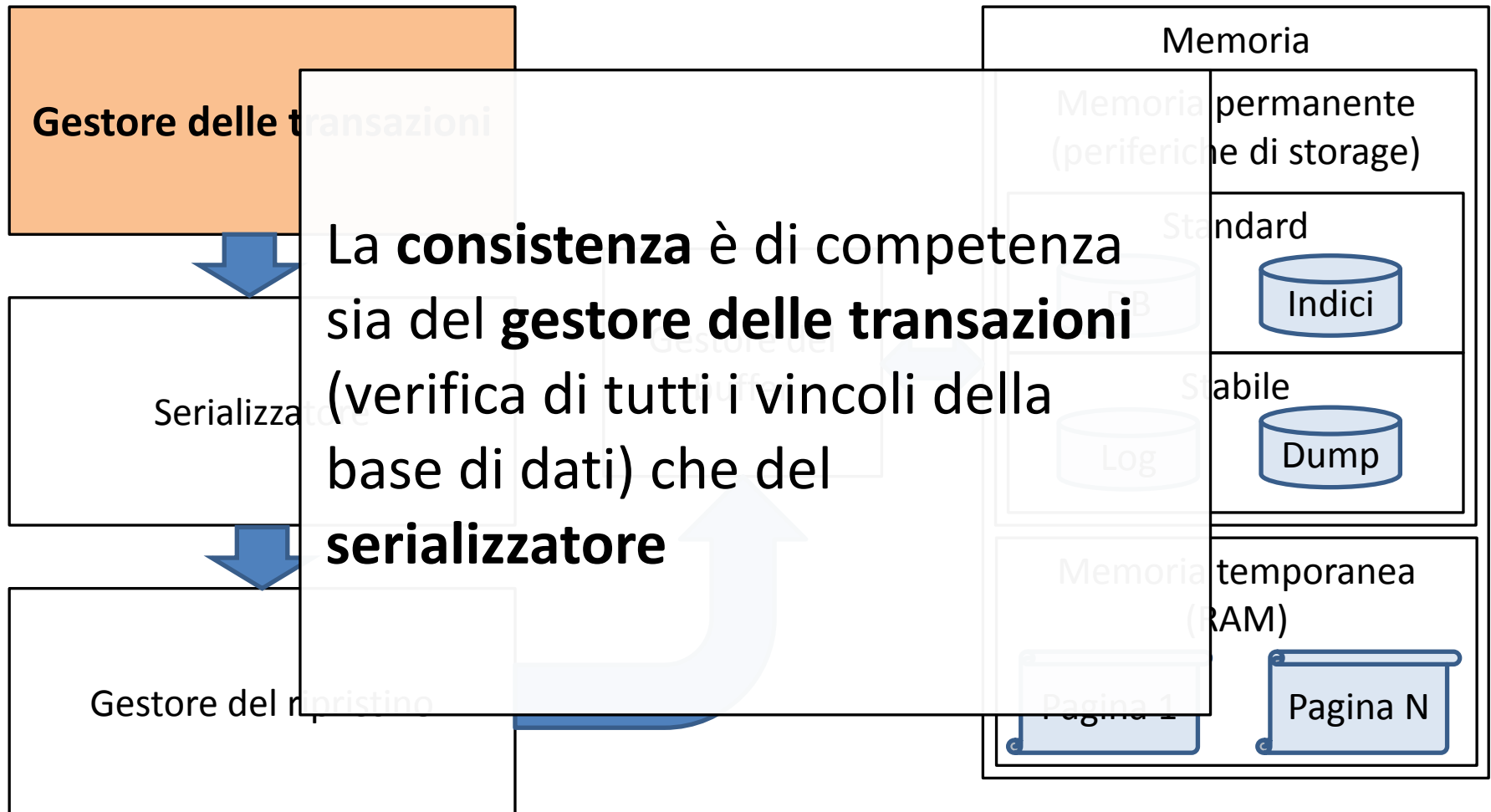
Architettura dei DBMS



Architettura dei DBMS



Architettura dei DBMS



Architettura dei DBMS: argomenti

- Gestione del buffer
- Metodi di accesso
- Gestore della concorrenza (serializzatore)
- Gestore del ripristino

Area di lavoro

Consideriamo l'area di lavoro, ovvero la memoria centrale

Nell'area di lavoro abbiamo la transazione T_i che richiede la lettura/modifica di un record

La transazione quando richiede l'attività su un record invia delle richieste al gestore del buffer

La transazione T_i invia una richiesta chiamata **fix pid** che significa: la transazione ha bisogno di avere a disposizione una pagina dati di indirizzo *pid* (page identification)

Le pagine

- Tutti i dati del database (tabelle, indici, log, dump) sono organizzati in pagine
- Le pagine hanno dimensioni che dipendono dal sistema (anche variabili)
- Le pagine contengono i record
- Se la transazione ha bisogno di lavorare su un ben determinato record (tupla), bisogna cercare una pagina con un certo indirizzo *pid* che si presume contenga il record desiderato

Caricamento delle pagine

- La transazione per lavorare su un dato ha sempre bisogno di chiedere al buffer la disponibilità di una ben precisa pagina
- Il **gestore del buffer** caricherà la pagina nel buffer stesso, prendendola dalla periferica di storage se non già presente nel buffer
- Il gestore del buffer prende i dati memorizzati nel disco (settori di tracce sul disco) e li mette nella cache presente nella periferica stessa
- Dalla cache vengono poste nel buffer della memoria centrale
- Il gestore del buffer risponde alla transazione inviandogli l'indirizzo della memoria centrale in cui trovare la pagina

Caricamento delle pagine

- La transazione acquisisce, tramite il comando fix pid, il puntatore alla zona di memoria (buffer) che contiene la pagina
- In caso di lettura, si accontenta di leggere i dati
- In caso di scrittura o modifica, avviene il processo inverso, ovvero la transazione modifica i dati **nel buffer**
- E' il gestore del buffer che si occupa di copiare la pagina con le modifiche nella periferica

Semplificazione

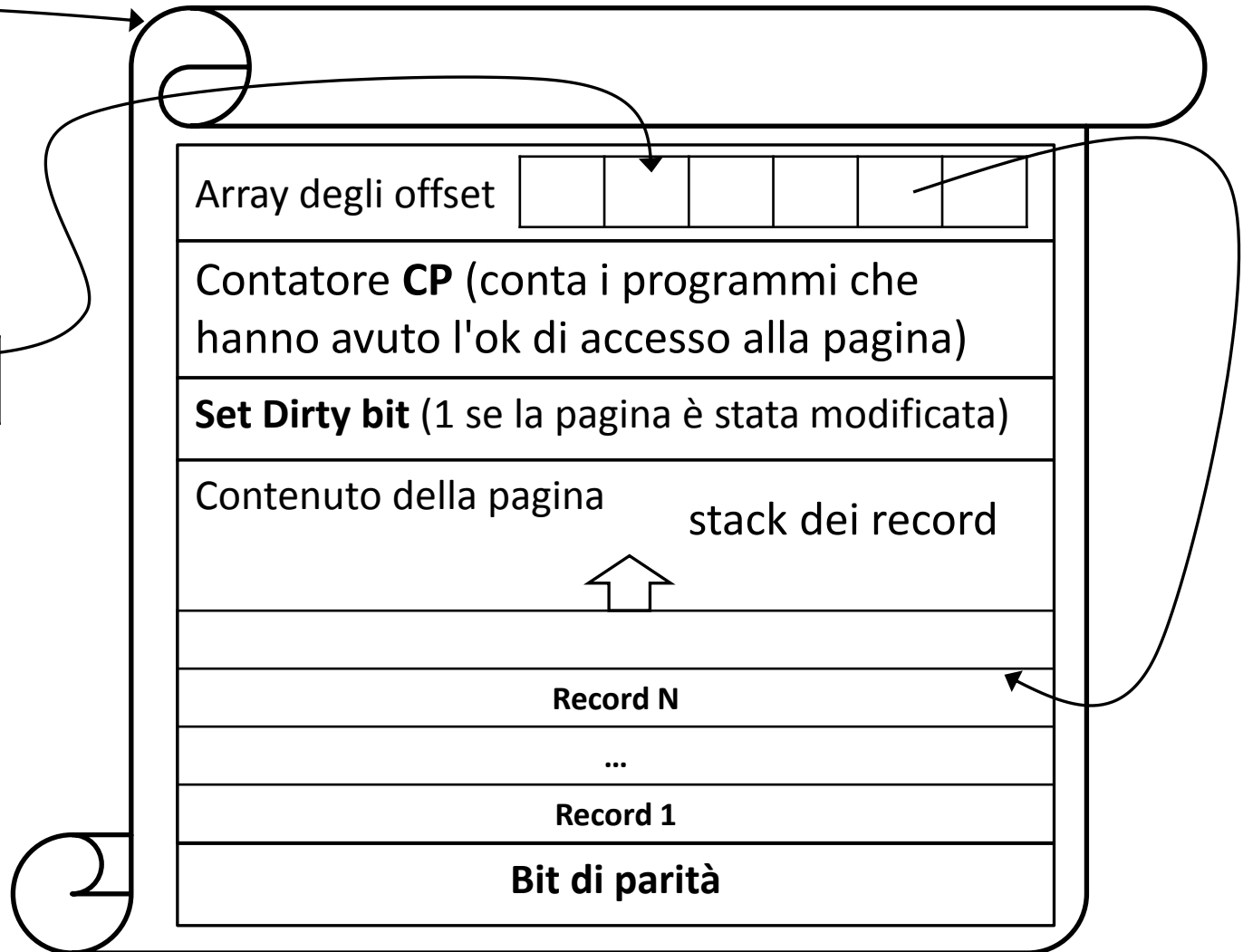
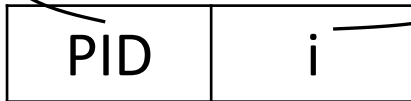
- La pagina è composta da un certo numero di blocchi fisici
- I blocchi fisici, a loro volta, sono composti da un certo numero di settori
- **In questo corso, però, confonderemo il blocco fisico con la pagina (dimensione blocco = dimensione pagina)**
- Le dimensioni caratteristiche dei blocchi fisici vanno dai 512Byte ai 4KB
- Le dimensioni delle pagine vanno dai 4KB e 8KB
- Il dimensionamento della pagina è importante perché influisce sulle prestazioni dell'intero DBMS

Tempi di trasferimento

- Ricordiamo che i tempi di trasferimento di una pagina tra periferiche di storage e RAM (e viceversa) sono dell'ordine dei 5-20 millisecondi (10^{-3} secondi)
- I tempi di trasferimento di pagine in RAM sono nell'ordine dei 50-70 nanosecondi (10^{-9} secondi)
- Abbiamo un fattore di 10^6 di differenza
- Il costo temporale è tutto sulla movimentazione delle pagine da periferica a memoria centrale (e viceversa)
- Nonostante i progressi tecnologici, il divario ha ancora lo stesso ordine di grandezza

Struttura di una pagina

Identificazione di
record: $RID[PID, i]$



RID (Record Identification)

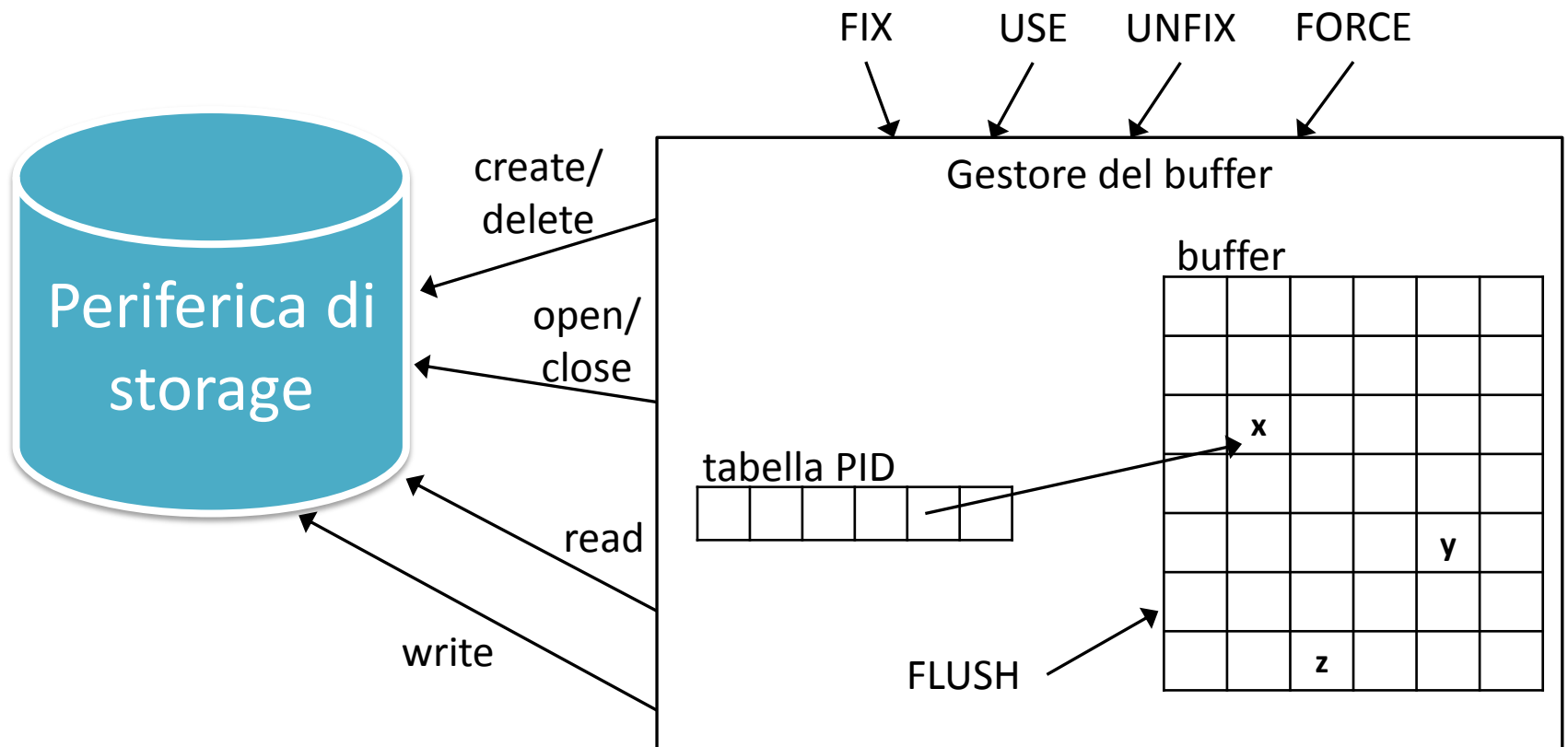
- L'architettura di RID con indice ed offset consente al gestore delle pagine (il file system) la possibilità di modificare la posizione dei record a piacimento all'interno dell'area dati senza influenzare gli indirizzi esterni
- Il gestore deve semplicemente modificare l'offset dell'array rendendo le applicazioni indipendenti rispetto alla movimentazione dei record all'interno dell'area dati della pagina

Considerazione sul RID

- Immaginiamo di avere una pagina piuttosto piena e che un'applicazione faccia un aggiornamento del record variandone la dimensione (esempio: stringa di caratteri da 100 a 1000 caratteri)
- Può succedere che il record non entri più nell'area dati e dev'essere posizionato da un'altra parte
- La tecnica dell'offset permette l'indirizzamento indiretto (la cella dell'array dell'offset può contenere il RID aggiornato)

Gestore del buffer

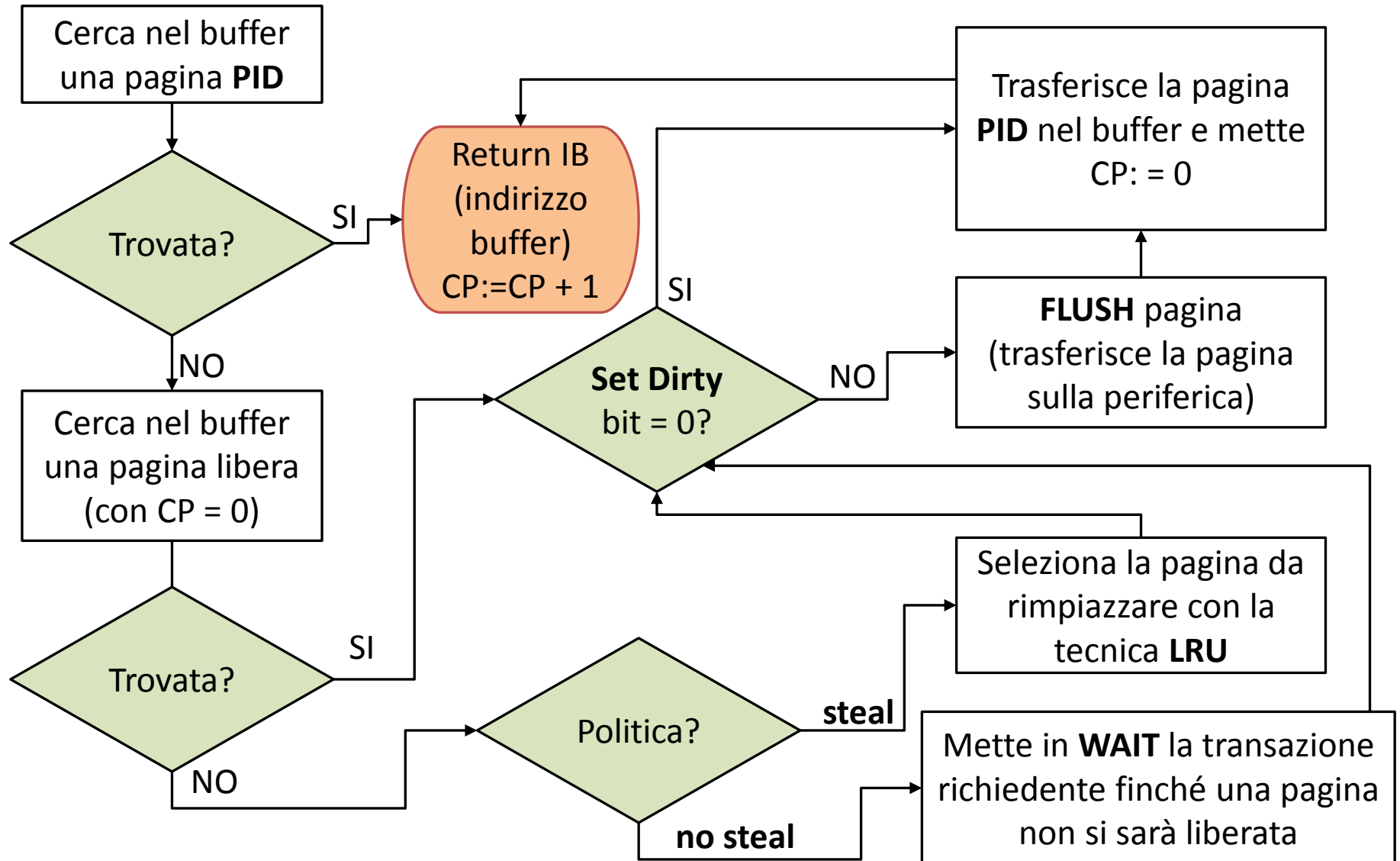
APPLICAZIONI/MODULI DBMS



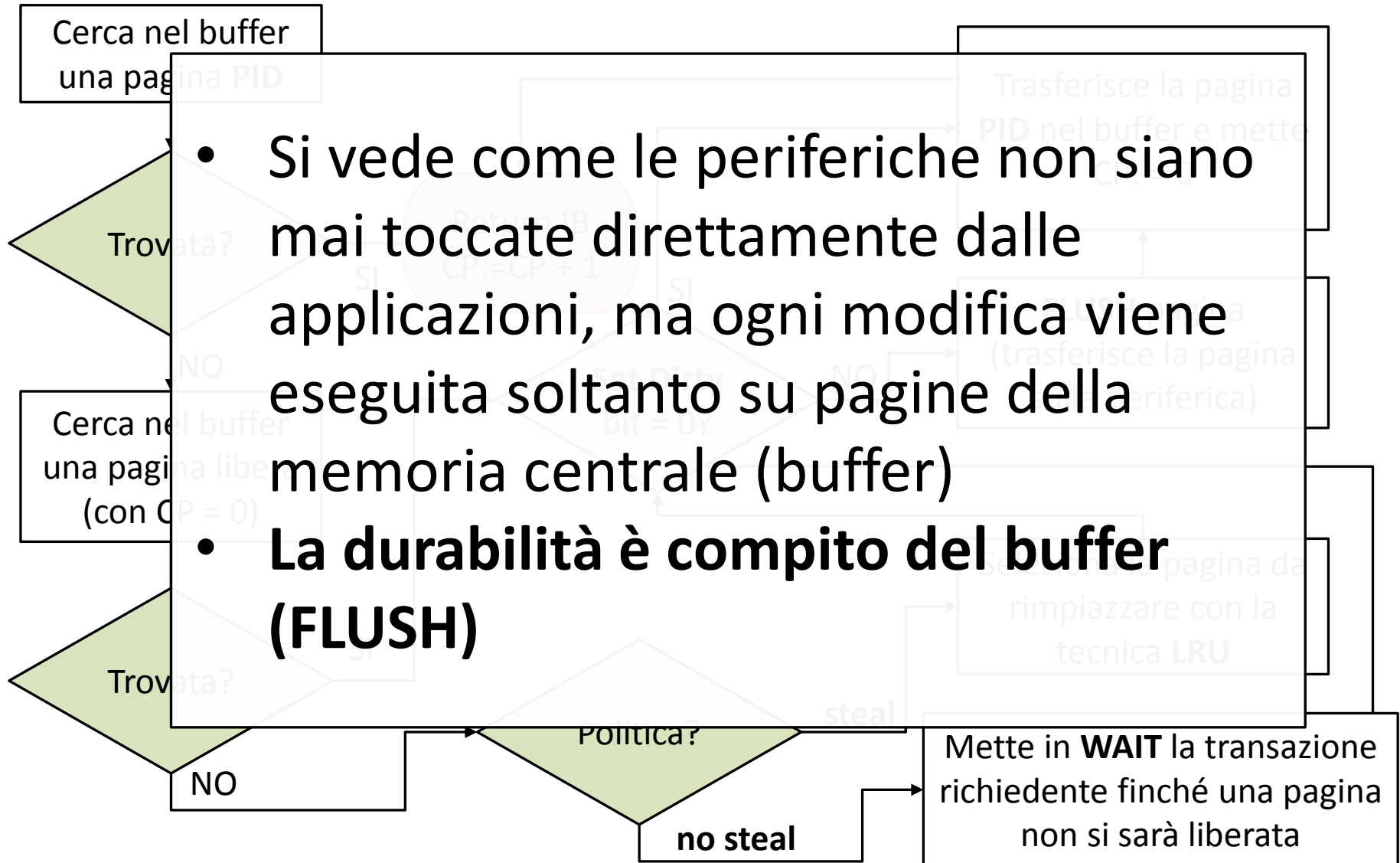
Descrizione dei comandi

- FIX: richiesta di accesso ad una pagina
- USE: richiesta di utilizzo di una pagina per cui l'applicazione ha già ottenuto l'accesso
- UNFIX: quando un'applicazione non ha più bisogno di una pagina, lo comunica al gestore (**$CP := CP - 1$**)
- FORCE: l'applicazione impone al gestore del buffer di trasferire la pagina dal buffer alla periferica
- FLUSH: comando usato dal gestore del buffer per trasferire una pagina sulla periferica

Algoritmo FIX(PID)



Algoritmo FIX(PID)



Organizzazione dei dati

- Per semplicità immagineremo una pagina come un elenco di record (eventualmente parzialmente libera)
- Una tabella (es. STUDENTI) è organizzata come un elenco di pagine a loro volta contenenti record non ordinati
- Esiste tuttavia la possibilità di introdurre un ordinamento dei record (e conseguentemente delle pagine)

Inserimento di dati ordinati

- Possiamo ad esempio immaginare la tabella studenti ordinata per matricola crescente

Pagina 1	
100	
200	
350	
410	

Pagina 2	
480	
500	
600	
750	

- Se devo inserire ad esempio un record con matricola 700 devo rispettare l'ordinamento
- Se c'è spazio nella pagina posso inserirlo ed eventualmente riorganizzo gli offset

Inserimento di dati ordinati

- Cosa succede se devo inserire un record in una pagina già piena?

Pagina 1	
100	
200	
350	
410	

Pagina 2	
480	
500	
600	
750	

- Posso utilizzare le cosiddette pagine trabocco (overflow)

Pagina 2 (area di overflow)	
700	...

Aree trabocco (overflow)

- L'utilizzo di aree trabocco determinano un degrado delle prestazioni perché l'accesso da una pagina all'altra ha un costo in termini di tempo
- Inoltre l'accesso sequenziale, utilizzando il pipelining, viene interrotto per poter accedere a pagine non consecutive della memoria
- I DBA possono riorganizzare le pagine per eliminare gli overflow eccessivi

Organizzazione delle pagine dati

In generale, una pagina fa riferimento ad una ben determinata tabella, i DBMS hanno però la possibilità di **clusterizzare**

Consideriamo una tabella

STUDENTI(MATR,NomeS,Indirizzo)

ed un'altra tabella

ESAMI(MATR,Corso,Voto,DataE)

I DBMS danno la possibilità di avere pagine/file che mettono insieme tabelle diverse

Clusterizzazione

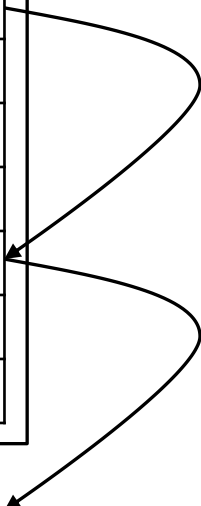
Vengono memorizzati fisicamente adiacenti i record di STUDENTI e ESAMI che si riferiscono alla stessa chiave MATR, in questo modo vengono velocizzati i join

Pagina 1		
333	Piero	TO
333	DB	18
333	Algoritmi	30
...
444	Lucia	CN
444	DB	30
444	Algoritmi	30

Clusterizzazione

Limite: i record di una singola tabella devono essere linkati tra di loro per le applicazioni che hanno bisogno di scandire l'intera tabella (gestione più complessa)

Pagina 1		
333	Piero	TO
333	DB	18
333	Algoritmi	30
...
444	Lucia	CN
444	DB	30
444	Algoritmi	30



Ottimizzazione fisica

- Ricordiamo che l'ottimizzatore fisico prende in input un albero di parsificazione modificato dall'ottimizzatore logico e corrispondente ad un'interrogazione DML
- L'ottimizzatore si basa innanzitutto sui **metodi di accesso**, cioè deve scegliere come accedere ai dati richiesti
- Tutti i DBMS possono presentare più modalità di accesso ai dati

Metodi di accesso

Esistono tre grandi categorie di metodi di accesso

- Metodo di accesso sequenziale (anche detto seriale)
- Metodo d'accesso tabellare
- Metodo d'accesso diretto

Metodo d'accesso sequenziale

E' detto anche metodo d'accesso seriale

- In letteratura, l'accesso **sequenziale** implica l'ordinamento su un attributo, ad esempio, nella tavola STUDENTE, posso decidere di ordinare sulla matricola o sul nome e le tuple vengono caricate nelle pagine seguendo l'ordinamento
- In letteratura, l'accesso **seriale** implica l'assenza di ordinamenti e le tuple sono caricate nelle pagine così come arrivano dal file system

Metodo d'accesso seriale

Come fa l'ottimizzatore fisico a valutare il vantaggio dell'uso di queste tecniche al posto di altre?

Immaginiamo una selezione:

$$\sigma_{MATR=333}(studenti)$$

Consideriamo un'organizzazione dei dati secondo un accesso seriale

Avremo un'organizzazione a pagine da Pagina 1 a pagina N

Il DBMS, non può far altro che leggere le pagine una dopo l'altra

Metodo d'accesso seriale

$$\sigma_{MATR=333}(studenti)$$

Il DBMS, non può far altro che leggere le pagine una dopo l'altra

Dal momento che MATR è chiave, avremo un'unica tupla (record) che sarà contenuta in una determinata pagina

Il metodo è essenzialmente una scansione delle pagine una dopo l'altra

Costo dell'accesso seriale

Il DBMS deve calcolare il costo **a priori** dell'accesso

Esistono dei modelli statistici per il calcolo del costo

Il modello statistico più semplice è di considerare una distribuzione di probabilità che dica la probabilità p_i di trovare il record nella pagina i

Il costo medio è quindi:

$$(1 \cdot p_1 + 2 \cdot p_2 + \dots + N \cdot p_N) = \sum_{i=1}^N i \cdot p_i$$

i = numero di pagine lette, p_i = probabilità che la tupla sia nella pagina i

Costo dell'accesso seriale

Occorre però disporre della distribuzione di probabilità, che nella maggior parte dei casi non è disponibile

Si utilizza l'ipotesi della **distribuzione uniforme**

La tupla ha la stessa probabilità di trovarsi in una qualsiasi pagina

In questo caso la formula si semplifica ($\forall i \ p_i = 1/N$) e diventa

$$\frac{1}{N} \sum_{i=1}^N i = \frac{1}{N} \cdot \frac{N(N+1)}{2} = \frac{N+1}{2}$$

Costo dell'accesso seriale

- Il costo della ricerca con successo è quindi $(N+1)/2$ (circa la metà delle pagine)
- Il costo della ricerca con insuccesso è N

Se utilizzo un'**organizzazione ordinata** (ordinamento su un attributo), cambia solo il costo della ricerca con insuccesso, infatti non ho bisogno di scorrere tutte le pagine, ma mi arresto quando ho superato il valore richiesto

- Il costo della ricerca (sequenziale) con insuccesso è uguale a quello della ricerca con successo: $(N+1)/2$

Costo dell'accesso seriale

Cosa succede se effettuo una selezione su un attributo non chiave?

$$\sigma_{INDIRIZZO='TO'}(studenti)$$

- In questo caso, devo scorrere tutte le pagine: costo N

Stesso discorso per le ricerche di range ($<$, $>$, between)

Questi aspetti non verranno trattati

Osservazione

Gli accessi sequenziali e seriali hanno costi che sono dell'ordine del numero delle pagine (nel miglior caso: metà delle pagine)

Questa organizzazione è troppo costosa se pensiamo a sistemi informativi con migliaia di pagine per le tabelle

Metodo di accesso tabellare

- E' il metodo di accesso più importante, in quanto si avvale degli indici
- Gli indici sono uno degli aspetti più importanti nella progettazione fisica di un DB
- Bisogna conoscerne vantaggi e svantaggi

Gli indici

- L'indice è una struttura separata dall'area primaria (l'area che contiene le pagine dati)
- Prima di introdurre gli indici, studieremo (ripasseremo) brevemente la struttura dati **B-albero**
- E' la struttura utilizzata nella maggior parte dei sistemi che implementano indici accanto alle tavole primarie

B-albero

Il **B-albero** è un albero semi-bilanciato di ordinamento in cui, ogni sottoalbero a sinistra di una chiave k ha chiavi di ricerca strettamente inferiori alla chiave k , e ogni sottoalbero di destra ha chiavi strettamente superiori a k

- Per chiave intendiamo "chiave di ricerca", non chiave relazionale
- E' un attributo qualsiasi su cui si effettua una ricerca

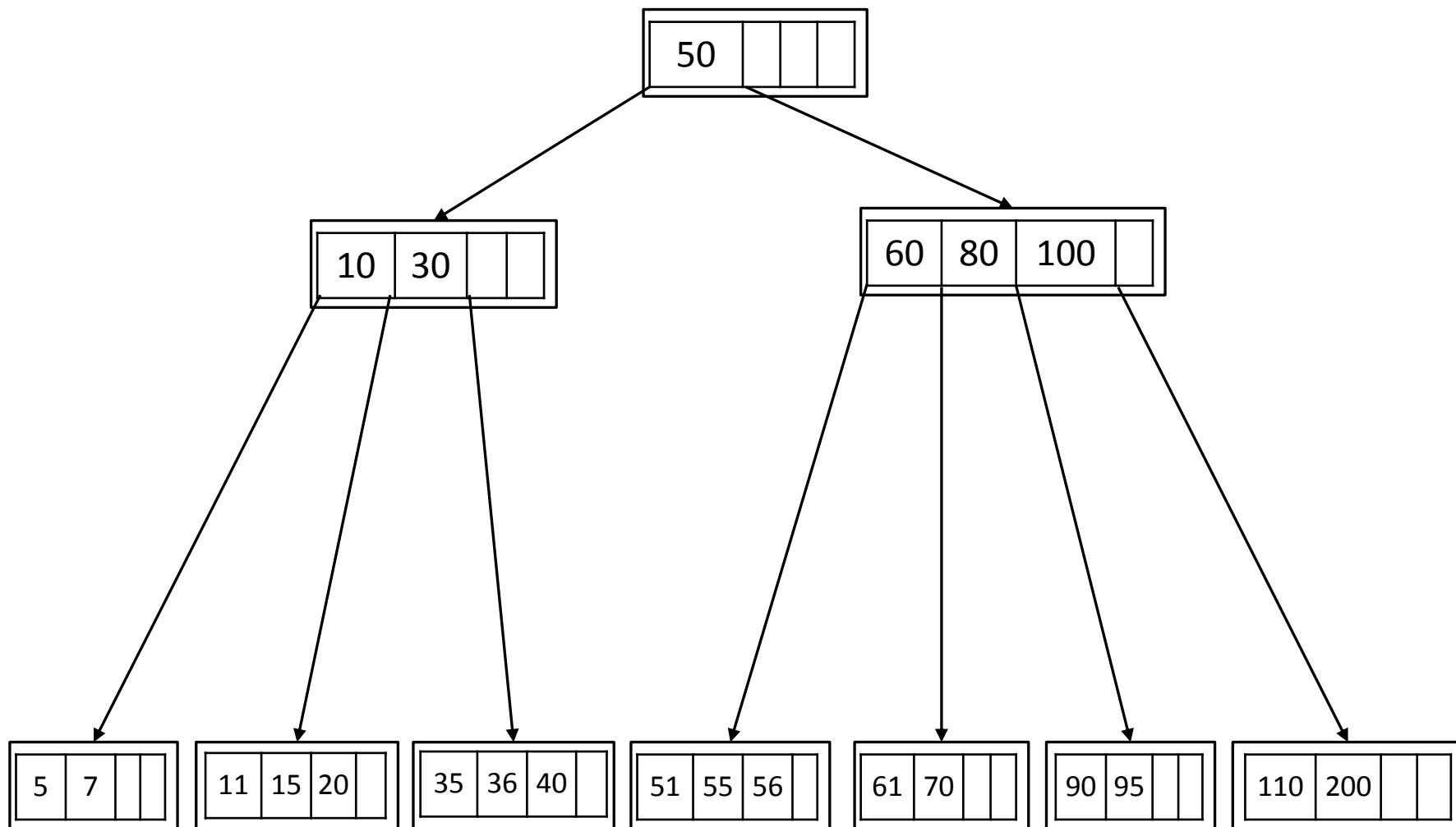
B-albero

Caratterizzeremo il B-albero con un parametro m che è il massimo numero di figli di un nodo (branching factor)

Abbiamo quindi

- $m - 1$: massimo numero di chiavi di un nodo
- $\lceil (m/2) - 1 \rceil$: minimo numero di chiavi presenti in un nodo interno
- Se j è il numero di chiavi presenti in un nodo: $1 \leq j \leq m - 1$ (nella radice), mentre $\lceil (m/2) - 1 \rceil \leq j \leq m - 1$ (nei nodi interni)
- Se un nodo non foglia ha j chiavi deve avere $j + 1$ figli
- Le foglie devono essere tutte sullo stesso livello

Esempio di B-Albero (m=5)



B-albero

Il **B-albero** è un albero di ricerca, cioè data una qualsiasi chiave di ricerca all'interno di un nodo, tutte le chiavi del sottoalbero di sinistra sono minori della chiave di separazione, tutte le chiavi del sottoalbero di destra sono maggiori della chiave separatrice

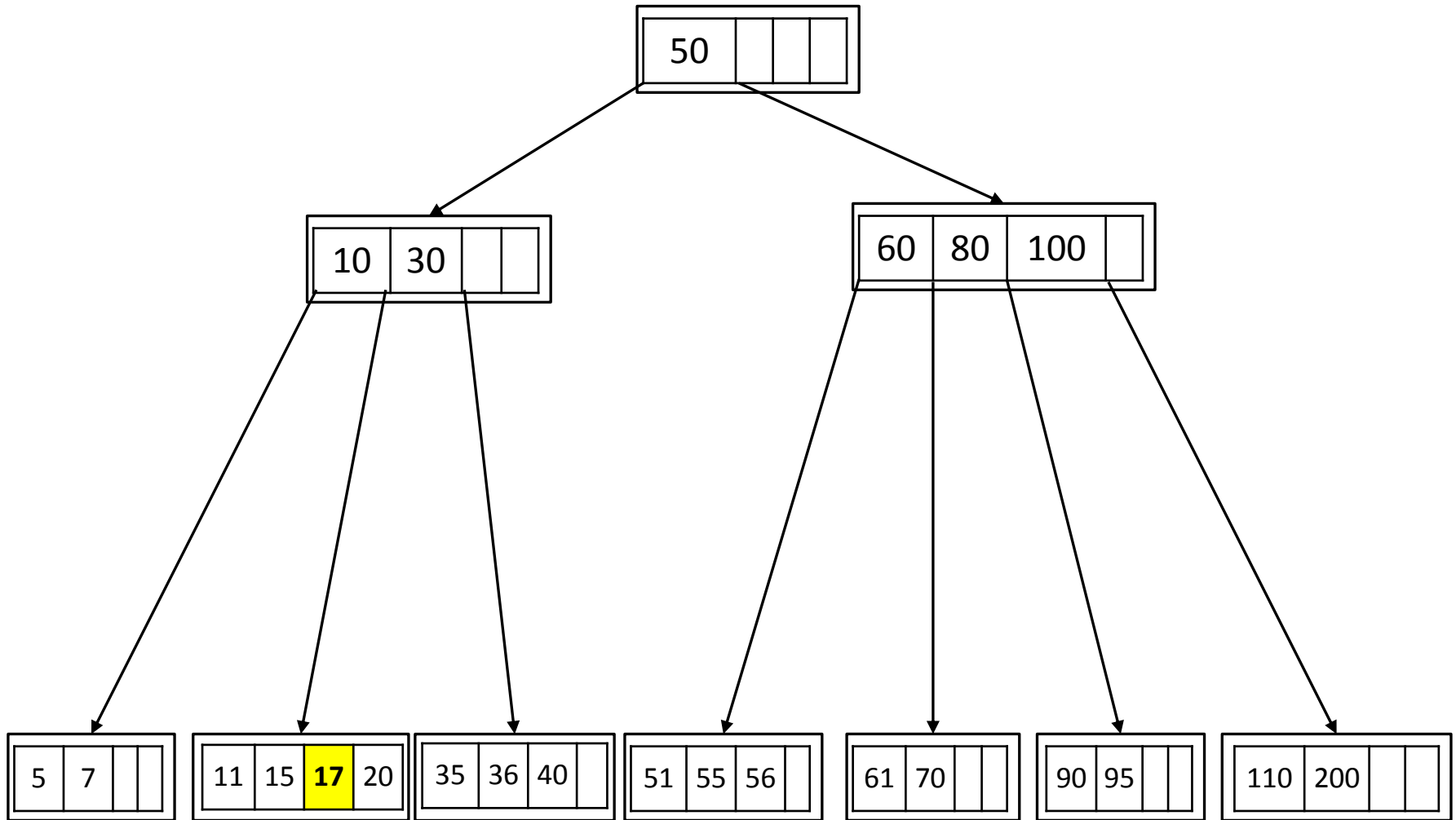
Algoritmo di inserzione

- Supponiamo di avere un B-albero che rispetta tutte le proprietà
- Supponiamo di volere inserire 17 all'interno dell'albero
- Data la chiave di ricerca che si vuole inserire, la prima fase dell'algoritmo di inserzione è uguale all'algoritmo di ricerca

Algoritmo di ricerca

- Per cercare 17 devo entrare nella radice, confronto la chiave di ricerca con le chiavi della radice ($17 < 50$) e scendo nel sottoalbero di sinistra
- Confronto la chiave di ricerca con le chiavi del nodo e scopro che ($10 < 17 < 30$) quindi entro nel secondo sottoalbero
- Dato che 17 non è contenuto nell'albero, prima o poi arriverò ad una foglia
- Ci sono due possibilità:
 1. La foglia contiene meno chiavi del massimo ($m - 1$)
 2. La foglia contiene esattamente $m - 1$ chiavi
- Nel primo caso inserisco direttamente la chiave nella foglia

Esempio di B-Albero (m=5)



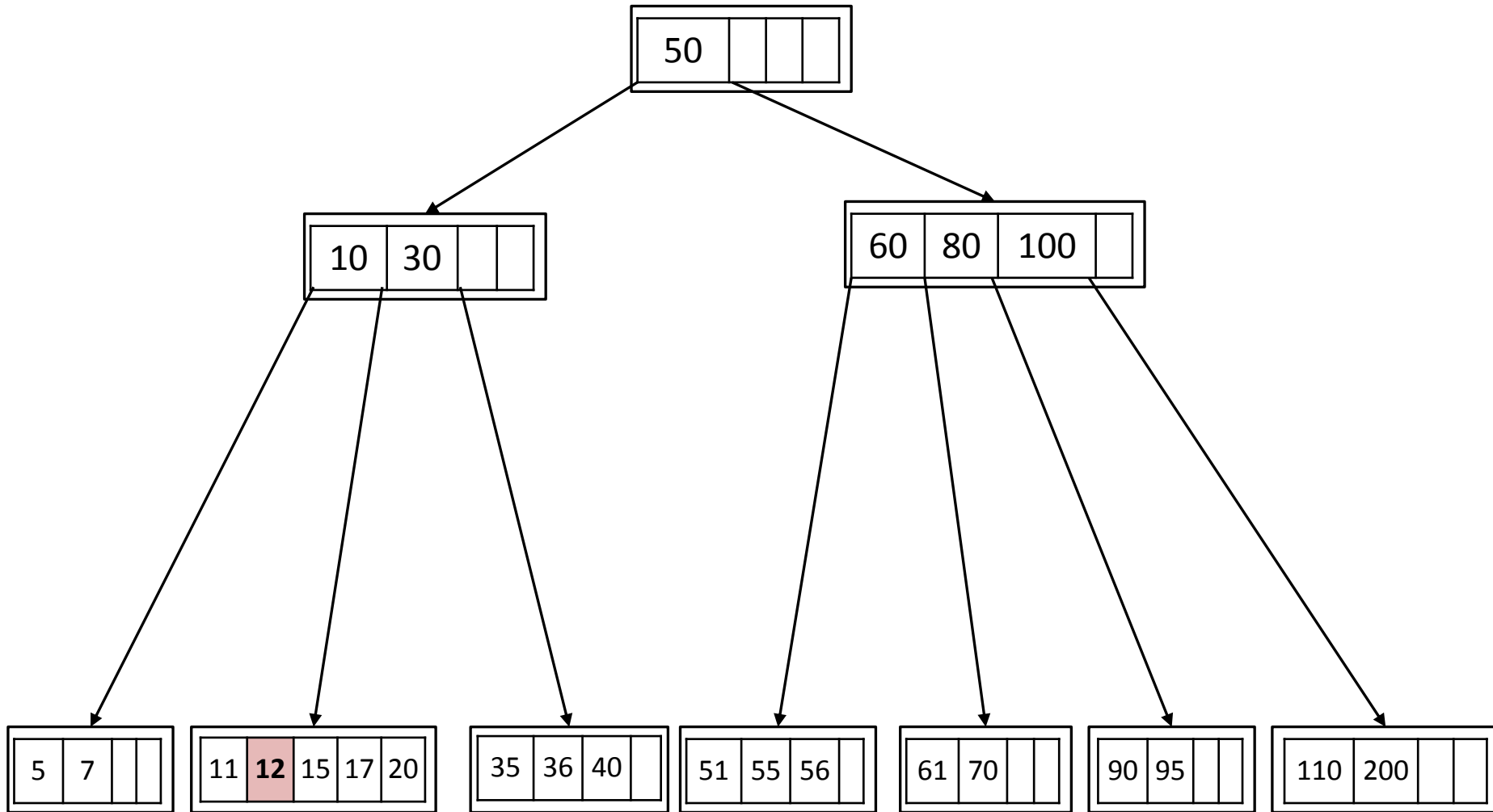
Algoritmo di inserzione

L'algoritmo di inserzione diventa interessante quando ci troviamo nel secondo caso (inserimento di una chiave in una foglia satura)

Proviamo ad inserire la chiave 12

Se si volesse forzare l'inserimento della chiave, ci troveremmo in una situazione di nodo "supersaturo"

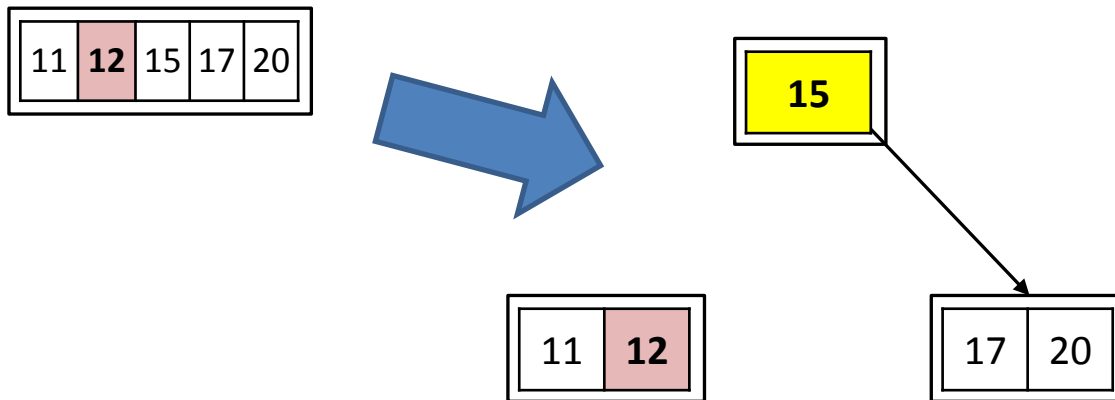
Esempio di B-Albero (m=5)



Algoritmo di inserzione

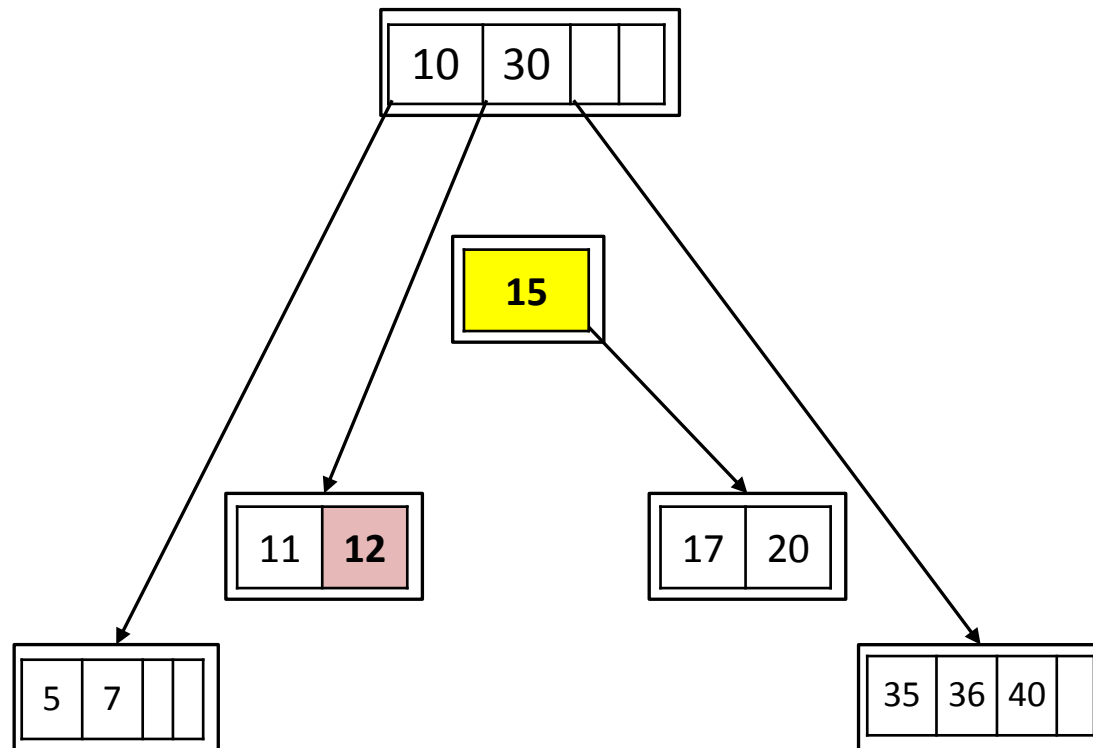
A questo punto avviene il passo di split:

1. Si divide la foglia "supersatura" in due foglie (generando un nuovo nodo) e si prende la chiave intermedia (nel nostro caso è 15)
2. Si estrae la chiave e la si collega al nuovo nodo



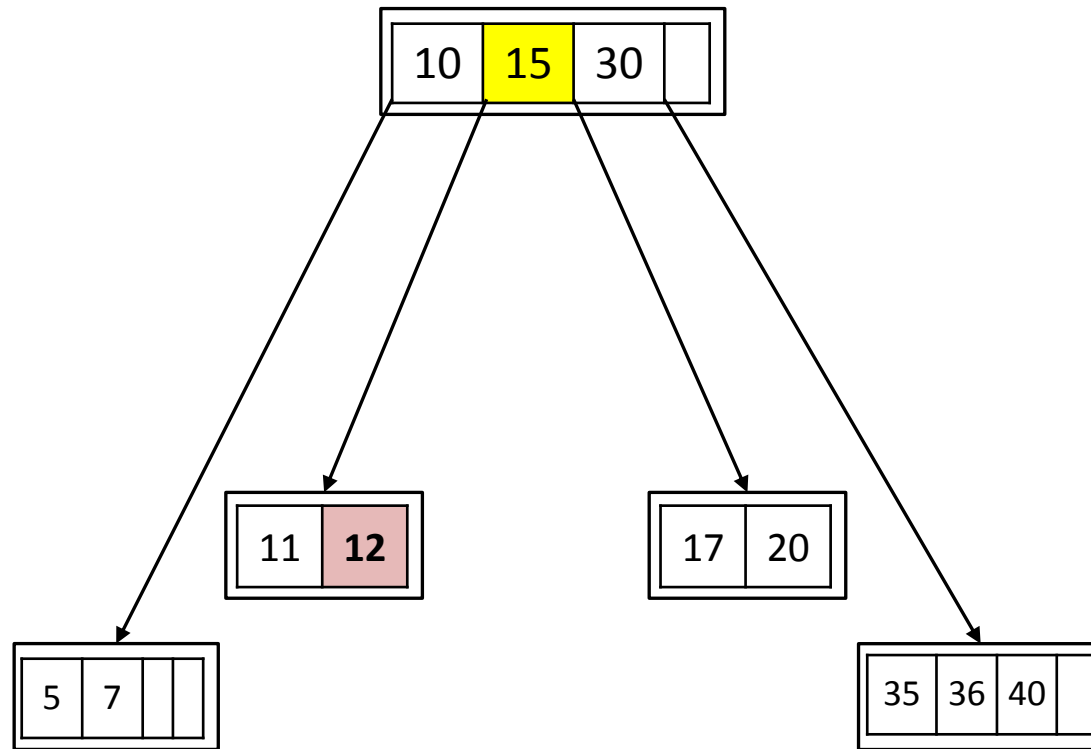
Algoritmo di inserzione

Consideriamo ora il nodo superiore



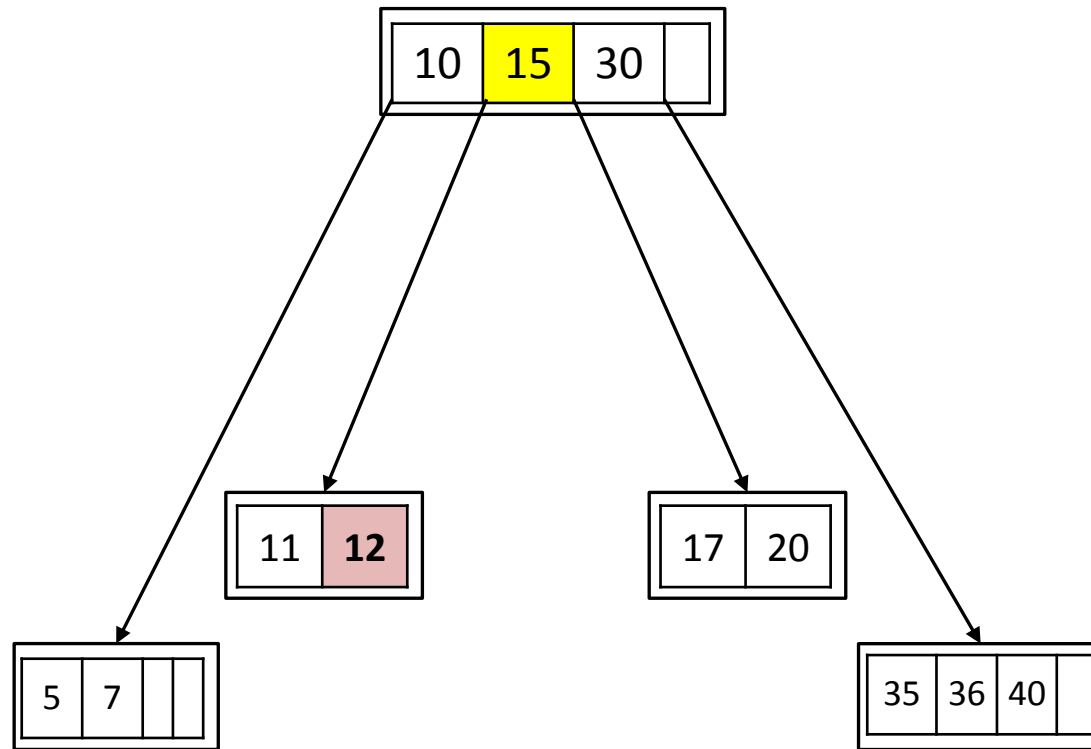
Algoritmo di inserzione

La chiave isolata viene inserita all'interno del nodo padre



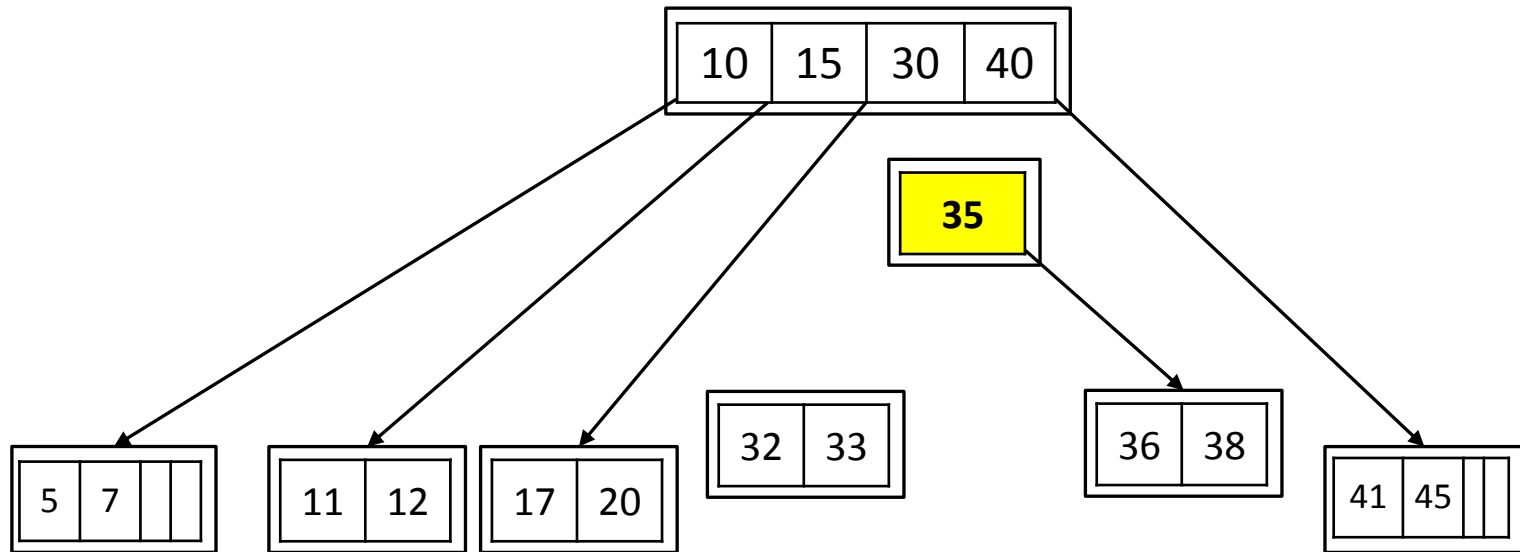
Algoritmo di inserzione

La trasformazione continua a mantenere vere le proprietà, in particolare il numero minimo di chiavi rimane $\lceil (m/2) - 1 \rceil$



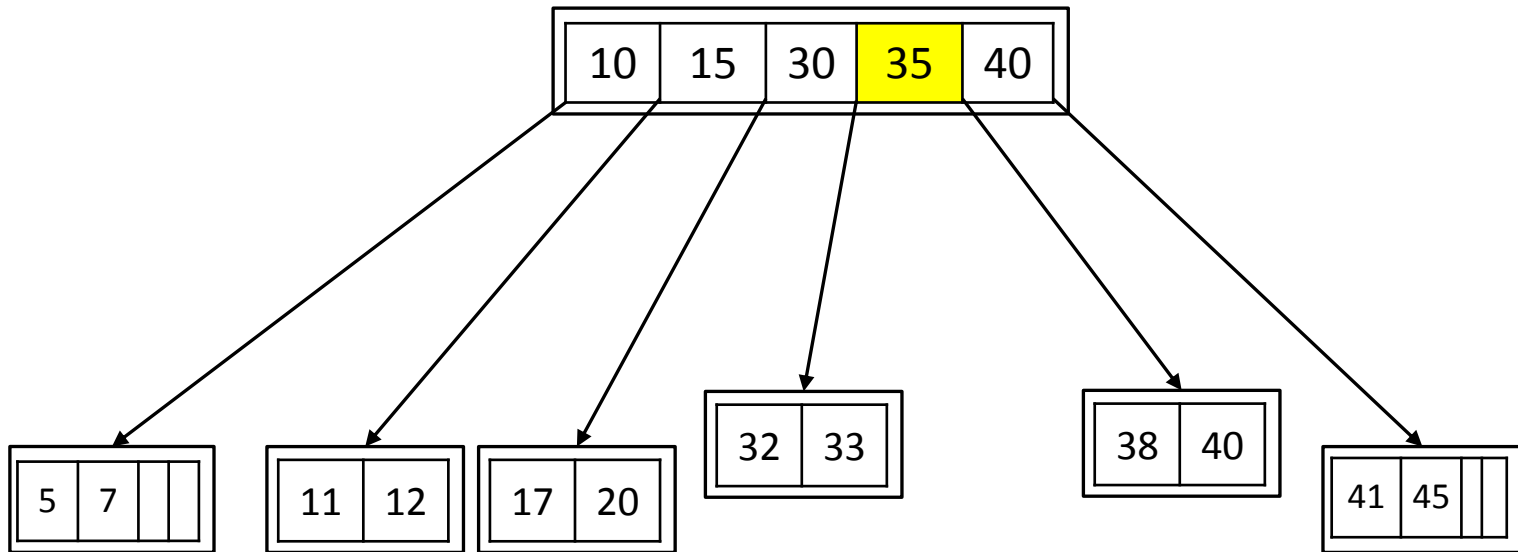
Algoritmo di inserzione

Se ad un certo punto il nodo padre si ritrova con esattamente $m - 1$ chiavi e c'è bisogno di inserire una chiave risultato di uno split dal livello inferiore, l'algoritmo è sempre lo stesso



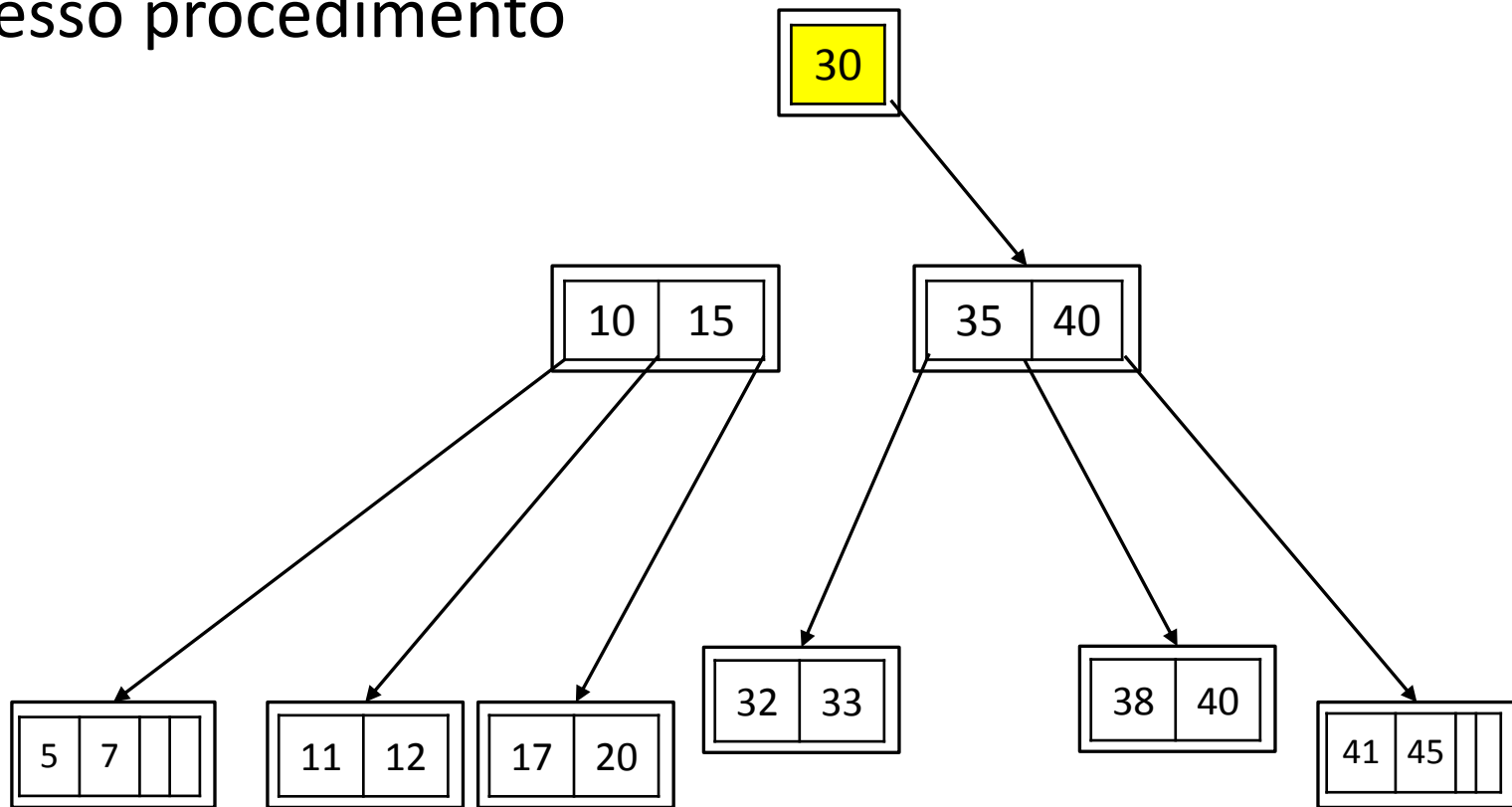
Algoritmo di inserzione

Abbiamo quindi un nodo supersaturo e applichiamo lo stesso procedimento



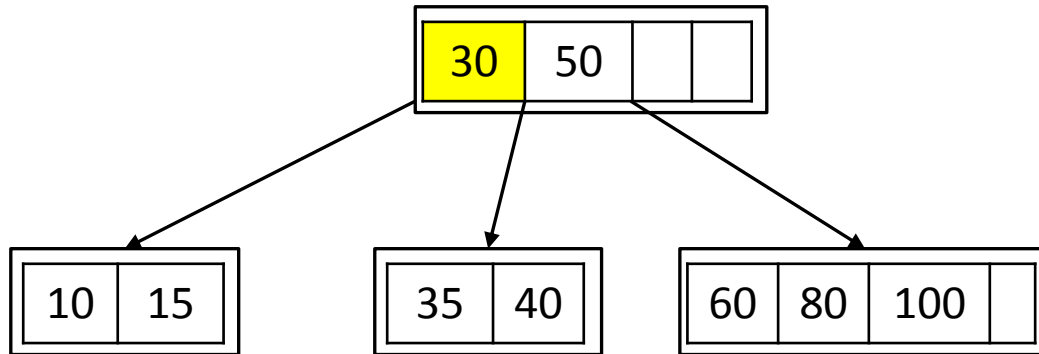
Algoritmo di inserzione

Abbiamo quindi un nodo supersaturo e applichiamo lo stesso procedimento



Algoritmo di inserzione

In questo caso inseriamo la chiave nella radice



Osservazioni

- Quando si fanno split di foglie, non si aumenta il numero di livelli di un albero
- Quando si fanno split di nodi interni, non si aumenta il numero di livelli dell'albero
- Quando la radice diventa satura, l'algoritmo aumenta il numero di livelli dell'albero

Algoritmo di inserzione

Dopo varie inserzioni, la radice diventa supersatura

30	50	100	150	200
----	----	-----	-----	-----

Algoritmo di inserzione

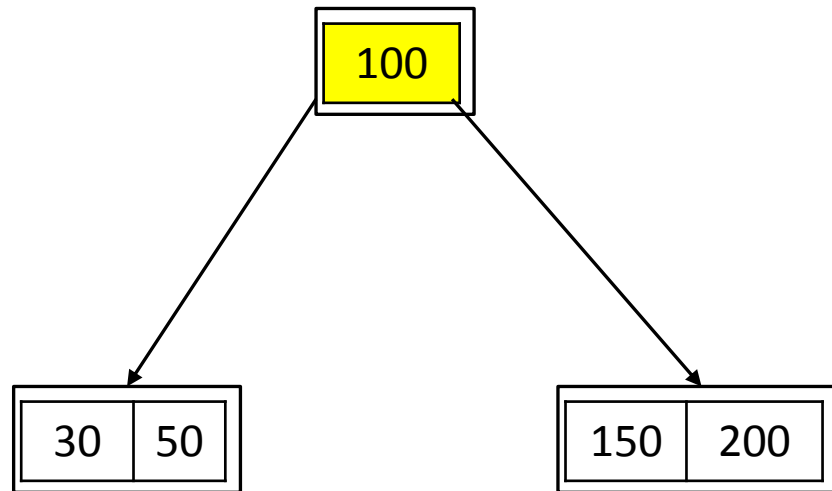
Splittiamo il nodo, non essendoci più padri **si crea un nuovo nodo**

30	50	100	150	200
----	----	-----	-----	-----

Algoritmo di inserzione

Splittiamo il nodo, non essendoci più padri **si crea un nuovo nodo**

Questo spiega anche perché la radice può contenere anche una sola chiave



Caso particolare

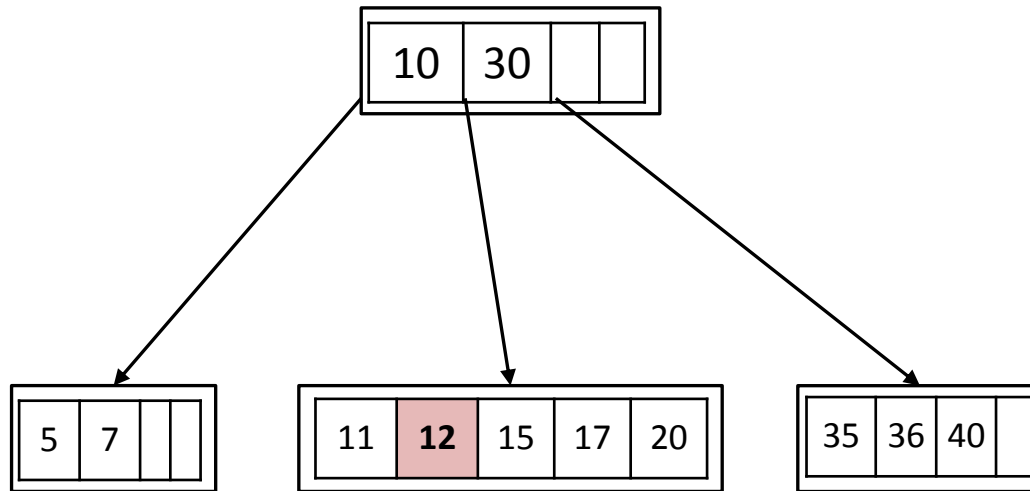
Se m è un numero pari, il progettista può scegliere di splittare su 30 o 35, ma le proprietà del B-albero rimangono inalterate

10	15	30	35	40	45
----	----	----	----	----	----

Bilanciamento

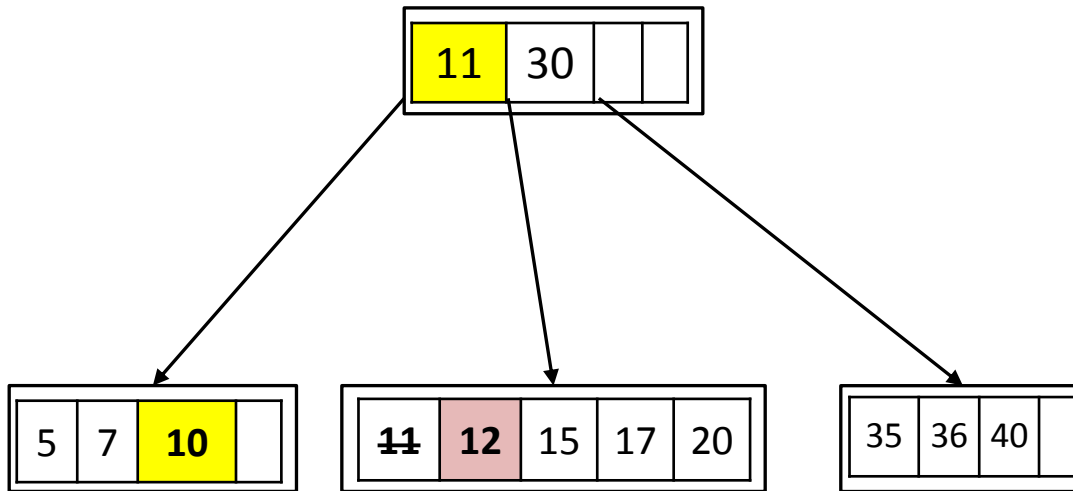
Lo split non è sempre richiesto (è costoso)

Prima di effettuare lo split è possibile adottare una tecnica detta di **bilanciamento**



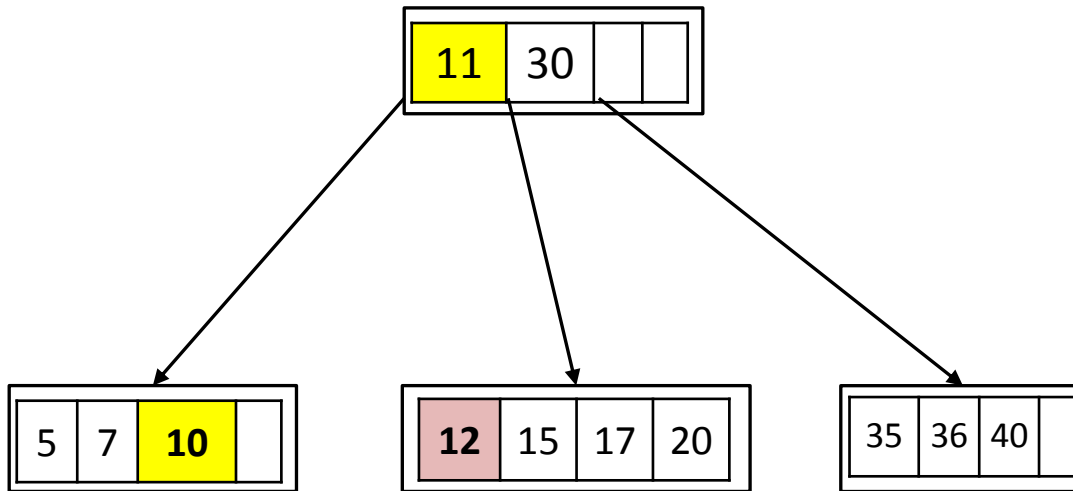
Bilanciamento

Il bilanciamento consiste nel trasferire la chiave 10 dal nodo padre al nodo immediatamente accanto (in questo caso quello di sinistra) spostando 11 nel nodo padre



Bilanciamento

Il bilanciamento consiste nel trasferire la chiave 10 dal nodo padre al nodo immediatamente accanto (in questo caso quello di sinistra) spostando 11 nel nodo padre



Analisi quantitativa

Il dato quantitativo importante per i DBMS è il numero di livelli (nell'esempio 3)

Come si fa a stimare il numero di livelli del B-albero?

Il numero di livelli può essere stimato da un range molto stretto (con una tolleranza di ± 1)

Stima del numero di livelli

La stima del numero di livelli L , si basa sul numero di nodi del livello e del numero di chiavi del livello

- Calcoliamo prima il massimo numero di livelli raggiungibile quando il B-albero è caricato con N chiavi (limite superiore del range)
- Ipotizziamo una strategia di inserzione tale per cui il numero di chiavi per ogni nodo sia il minimo possibile (1 nella radice, $\lceil (m/2) - 1 \rceil$ in tutti gli altri)
- **Intuizione:** con un albero poco caricato, ci aspettiamo un numero di livelli alto (limite superiore)

Stima del confine superiore

- Livello 1: il numero di chiavi del livello 1 (radice) è 1
- Nel Livello 2 avrò due nodi (perché il nodo padre ha solo una chiave)
- I nodi interni devono avere $\lceil (m/2) - 1 \rceil$ chiavi
- Chiamo $k = \lceil (m/2) - 1 \rceil$
- Nel secondo livello abbiamo due nodi, cioè $2k$ chiavi
- Al Livello 3 abbiamo $2(k+1)$ nodi (ogni nodo al livello superiore ha k chiavi e genera $k + 1$ figli)
- Nel terzo livello abbiamo quindi $2(k+1)k$ chiavi

Stima del confine superiore

- Al livello 4 avremo quindi $2(k+1)^2$ nodi (ognuno dei $2(k+1)$ nodi del livello superiore genera $k+1$ figli)
- Nel quarto livello abbiamo $2(k+1)^2k$ chiavi
- Al livello L abbiamo $2(k+1)^{L-2}$ nodi e $2(k+1)^{L-2}k$ chiavi

Stima del confine superiore

Immaginiamo di avere inserito tutte le N chiavi con la strategia descritta, il numero di chiavi totali dell'albero si può calcolare come:

$$1 + 2k \cdot \sum_{i=0}^{L-2} (k + 1)^i$$

che però non è esattamente uguale a N , ma in generale può essere minore di N (può avanzare qualche chiave da inserire nei nodi) abbiamo quindi

$$1 + 2k \cdot \sum_{i=0}^{L-2} (k + 1)^i \leq N$$

Stima del confine superiore

Sviluppiamo la sommatoria:

$$1 + 2k \cdot \sum_{i=0}^{L-2} (k+1)^i = 1 + 2k \cdot \frac{1 - (k+1)^{L-1}}{1 - (k+1)} \leq N$$

E semplificando:

$$(k+1)^{L-1} \leq \frac{N+1}{2}$$

ricordiamo che $k = \lceil (m/2) - 1 \rceil$, quindi:

$$\left(\left\lceil \frac{m}{2} \right\rceil \right)^{L-1} \leq \frac{N+1}{2}$$

Stima del confine superiore

$$\left(\left\lceil \frac{m}{2} \right\rceil\right)^{L-1} \leq \frac{N+1}{2}$$

Passiamo ai logaritmi:

$$L - 1 \leq \log_{\left\lceil \frac{m}{2} \right\rceil} \left(\frac{N+1}{2} \right)$$

Quindi :

$$L \leq \log_{\left\lceil \frac{m}{2} \right\rceil} \left(\frac{N+1}{2} \right) + 1$$

Stima del confine inferiore

Ricordiamo che il numero massimo di chiavi che un nodo può ricevere è $h = m - 1$

Mi aspetto di ottenere il numero minimo di livelli con N chiavi quando sfrutto al massimo le capacità dei nodi

- Al livello 1 (radice) avrò esattamente h chiavi
- Al livello 2 avrò $h + 1$ nodi e $(h + 1)h$ chiavi
- Al livello 3, avremo $(h+1)^2$ e $(h+1)^2h$ chiavi
- ...
- Al livello L , avremo $(h+1)^{L-1}$ nodi e $(h+1)^{L-1}h$ chiavi

Stima del confine inferiore

Possiamo calcolare il numero totale di chiavi inserite al livello L:

$$h \cdot \sum_{i=0}^{L-1} (h+1)^i = h \cdot \frac{1 - (h+1)^L}{1 - (h+1)}$$

che però non è in generale uguale a N (N dovrebbe essere esattamente divisibile per h), ma può essere maggiore

$$h \cdot \frac{1 - (h+1)^L}{1 - (h+1)} \geq N$$

Stima del confine inferiore

Ricordiamo che $h = m - 1$,

$$h \cdot \frac{1 - (h + 1)^L}{1 - (h + 1)} = (h + 1)^L - 1 = m^L - 1 \geq N$$

quindi

$$m^L \geq N + 1$$

e passando ai logaritmi

$$L \geq \log_m(N + 1)$$

Stima del range di L

Mettendo insieme le due disequazioni

$$\log_m(N + 1) \leq L \leq \log_{\lceil \frac{m}{2} \rceil} \left(\frac{N + 1}{2} \right) + 1$$

Considerazione sugli indici

Nell'indice in una basi di dati, i nodi dell'indice sono pagine e i puntatori ai livelli inferiori sono PID

Nella tecnologia moderna le pagine hanno dimensioni consistenti, ma solitamente le pagine degli indici vanno dimensionate in modo generoso

- Immaginiamo una pagina di 4KB
- Immaginiamo che una chiave di ricerca (insieme al PID) sia di 40Byte
- Ogni pagina può contenere quindi 100 chiavi di ricerca
- Le chiavi di ricerca, nei grandi sistemi informativi, possono essere centinaia di milioni

Approssimazione della formula

Dal momento che abbiamo valori alti di N e piuttosto alti di m , la formula

$$\log_m(N + 1) \leq L \leq \log_{\lceil \frac{m}{2} \rceil} \left(\frac{N + 1}{2} \right) + 1$$

si può approssimare in questo modo

$$\log_m(N) \leq L \leq \log_m(N) + 1$$

Quindi si può dire che il numero di livelli è circa $\log_m(N)$

Considerazione su L

Consideriamo $m=100$ e 3 livelli ($L=3$), N diventa 1 milione

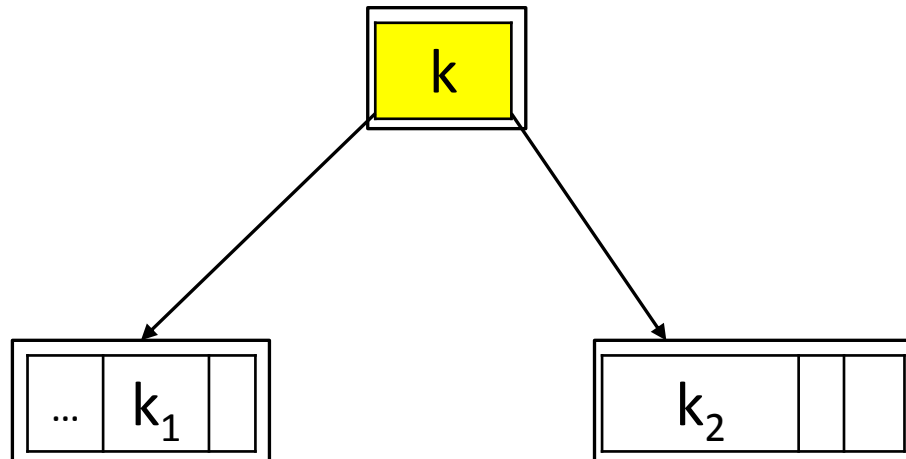
Questo significa che nella tecnologia attuale, con il dimensionamento standard della pagine, con soli tre livelli possiamo indicizzare un milione di chiavi (con nodi totalmente carichi)

Nei sistemi DBMS, con le tecniche di bilanciamento gli indici reggono bene un carico del 70% della capacità massima (con 3 livelli si reggono bene 700mila chiavi)

Per accedere ad un valore, al più movimento **3 pagine!**

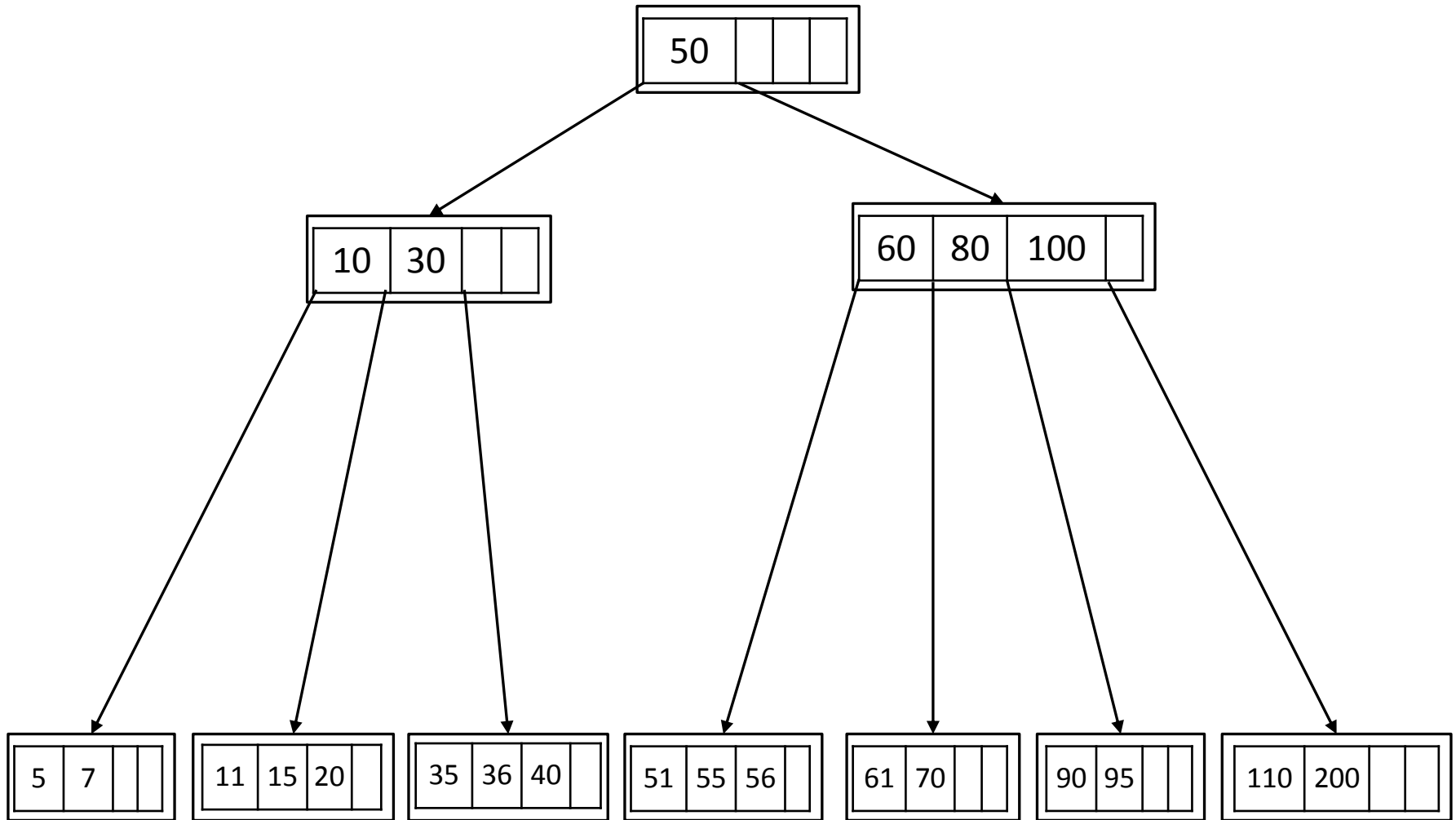
B-Alberi e indici

Una proprietà di tutti gli alberi di ricerca è che le chiavi interne sono chiavi separatrici di due insiemi di chiavi



Questo significa che la sequenza k_1, k, k_2 è una successione di chiavi logicamente contigue

B-Alberi e indici

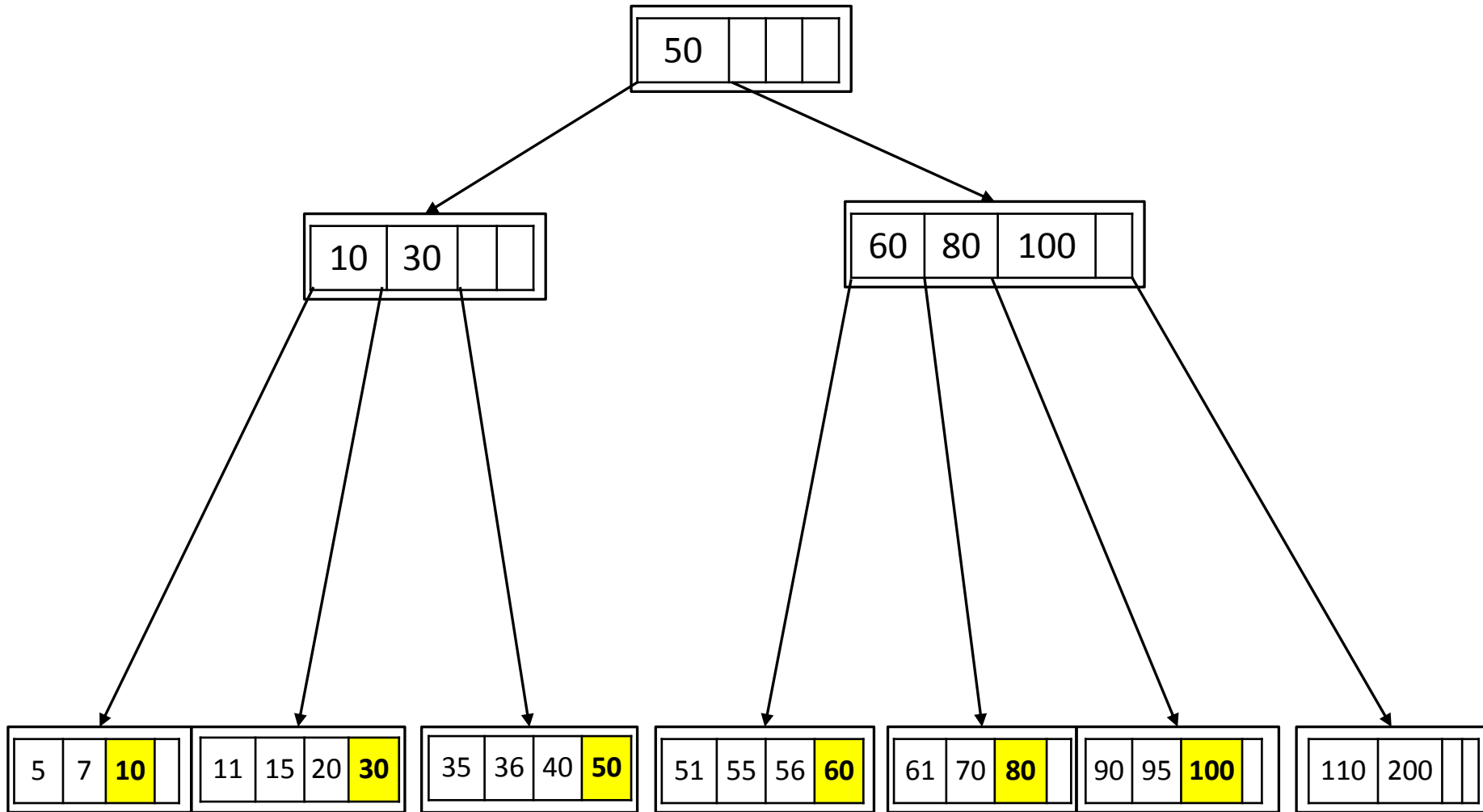


B+Albero

Si sceglie di compiere la seguente trasformazione

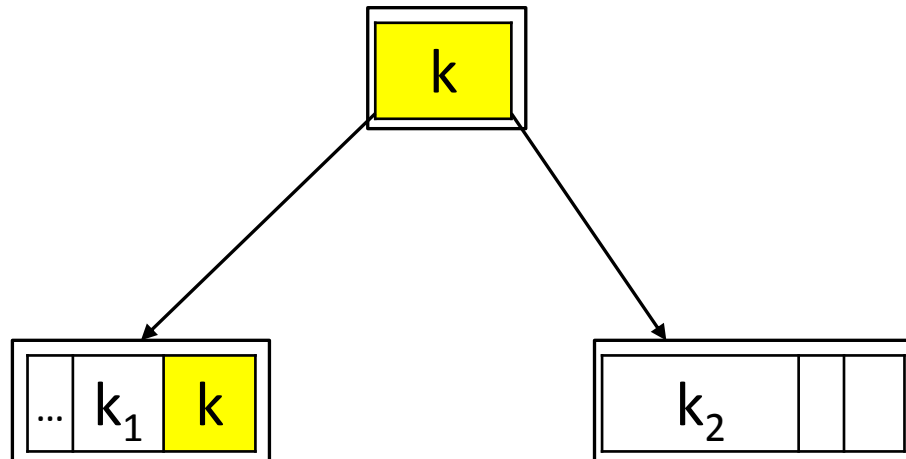
- parto dalla foglia più a sinistra
- copio la chiave separatrice (che trovo risalendo l'albero) dentro la foglia di sinistra

B-Alberi e indici



B-Alberi e indici

Copio la chiave **k** nel sottoalbero di sinistra



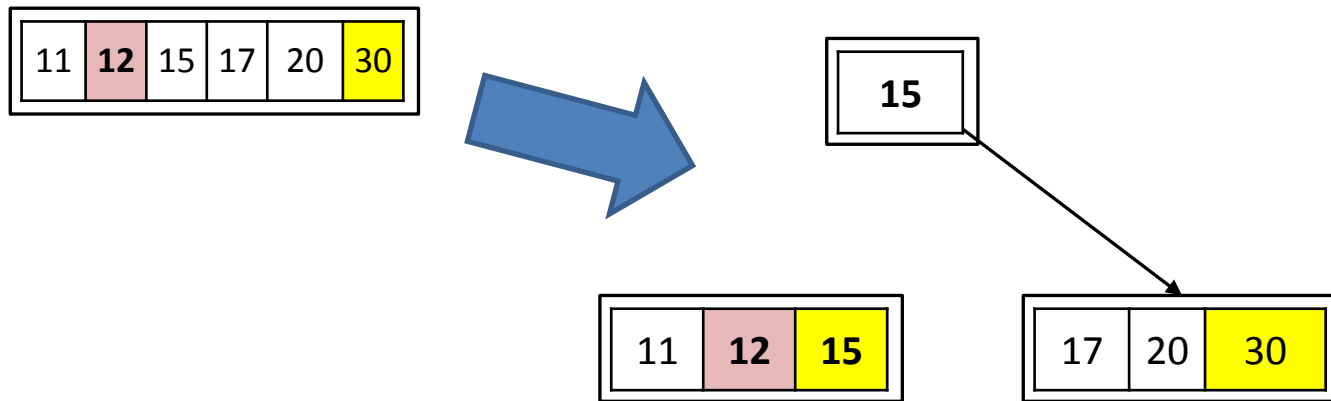
B+Albero

- Questa scelta non altera le proprietà del B-Albero
- L'unico cambiamento riguarda le foglie che nel B+Albero contengono una chiave in più
- Ogni chiave dei nodi interni è ricopiata in una foglia
- L'algoritmo di inserzione cambia leggermente

Inserzione nel B+Albero

Quando l'algoritmo di inserzione isola nella foglia la chiave separatrice, ne fa una copia esterna con il puntatore nella nuova foglia prodotta dallo split, ma viene lasciata nella foglia sotto elaborazione

Lo split nei nodi interni si esegue invece normalmente

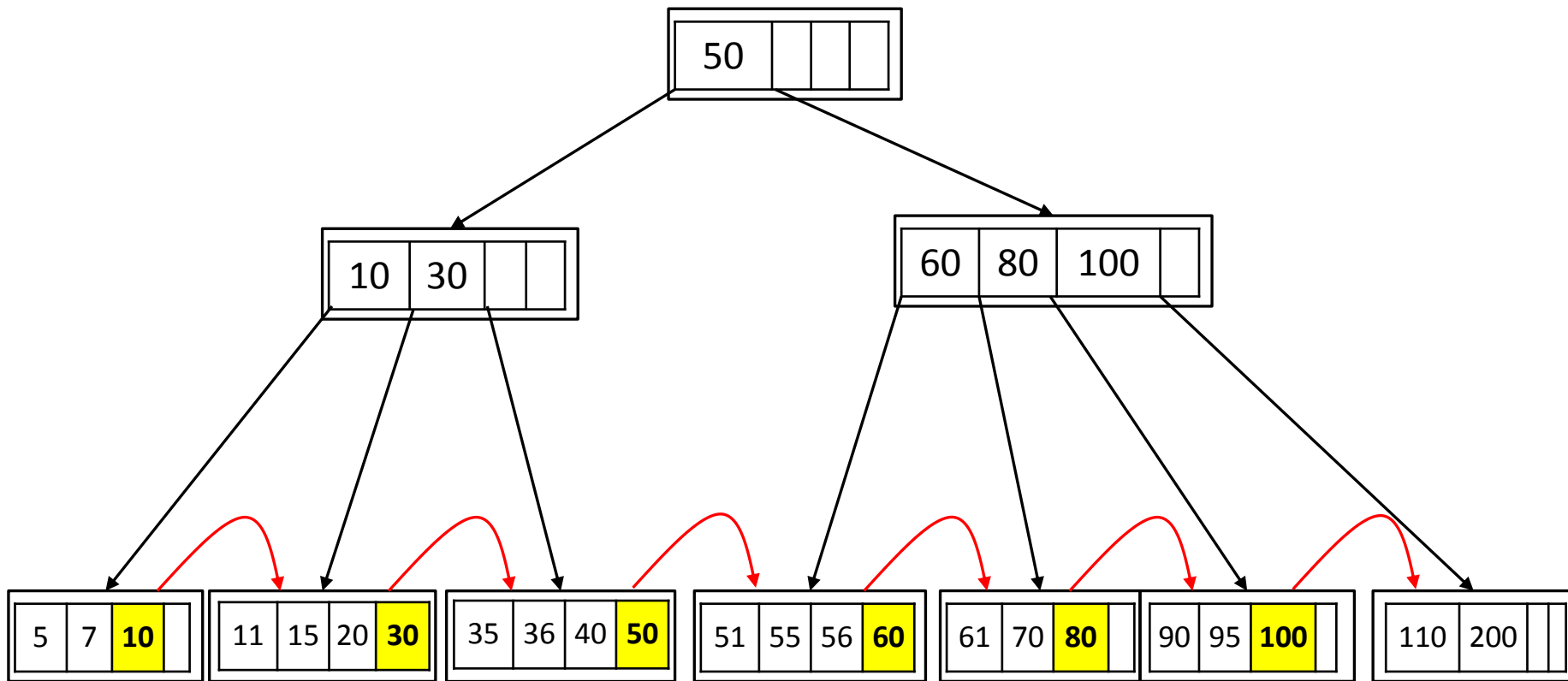


Collegamento con l'area primaria

- L'area primaria è l'elenco delle pagine della tabella
- Immaginiamo che i numeri riportati siano le matricole degli studenti
- Con il B+Albero abbiamo tutte le chiavi di ricerca nelle foglie
- L'indice viene arricchito dai RID abbinati a ciascuna chiave
- Esempio: per la chiave 35 avrò il RID che punta al record nell'area primaria corrispondente allo studente con matricola 35
- Le foglie dell'indice contengono i **data entry k^*** , cioè l'informazione abbinata alla chiave di ricerca della locazione della tupla cercata: sono coppie $\langle k, RID \rangle$

Collegamenti tra foglie

Infine le foglie sono tutte linkate tra di loro



Nodi interni e foglie

- Dal punto di vista strutturale, nodi interni e foglie sono simili
- Un nodo interno ha j chiavi e $j+1$ link (PID) ai figli
- Una foglia ha j chiavi e j puntatori (RID) ai record nell'area primaria più un puntatore (PID) alla foglia contigua (quindi $j+1$ link in tutto)

Ricerca puntuale nell'indice

Immaginiamo questa selezione

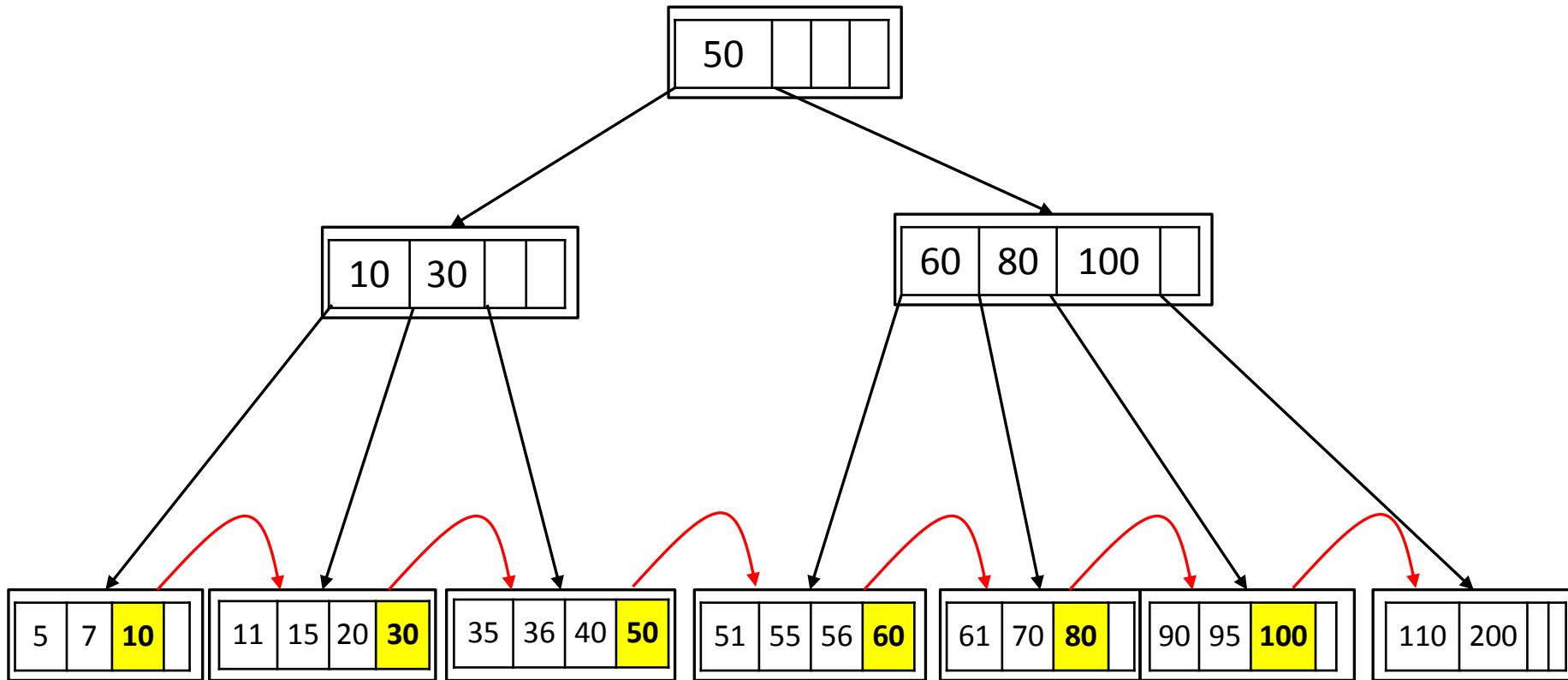
$$\sigma_{\text{MATR}=56}(\text{studenti})$$

- Il DBMS sa che c'è l'indice sul campo MATR, prende il valore (56) e lo confronta con la radice, dato che $56 > 50$ scende nel sottoalbero di destra ($56 < 60$) e successivamente nel sottoalbero di sinistra
- Trova il valore nella foglia e utilizza il RID per accedere al record
- In tutto sono stati necessari $L + 1$ accessi a pagine

Nei sistemi informativi reali, con centinaia di migliaia di chiavi di ricerca, in genere gli indici hanno 2/3 livelli

Collegamenti tra foglie

Ricerca della chiave 56



Ricerca di range di valori

Immaginiamo questa selezione

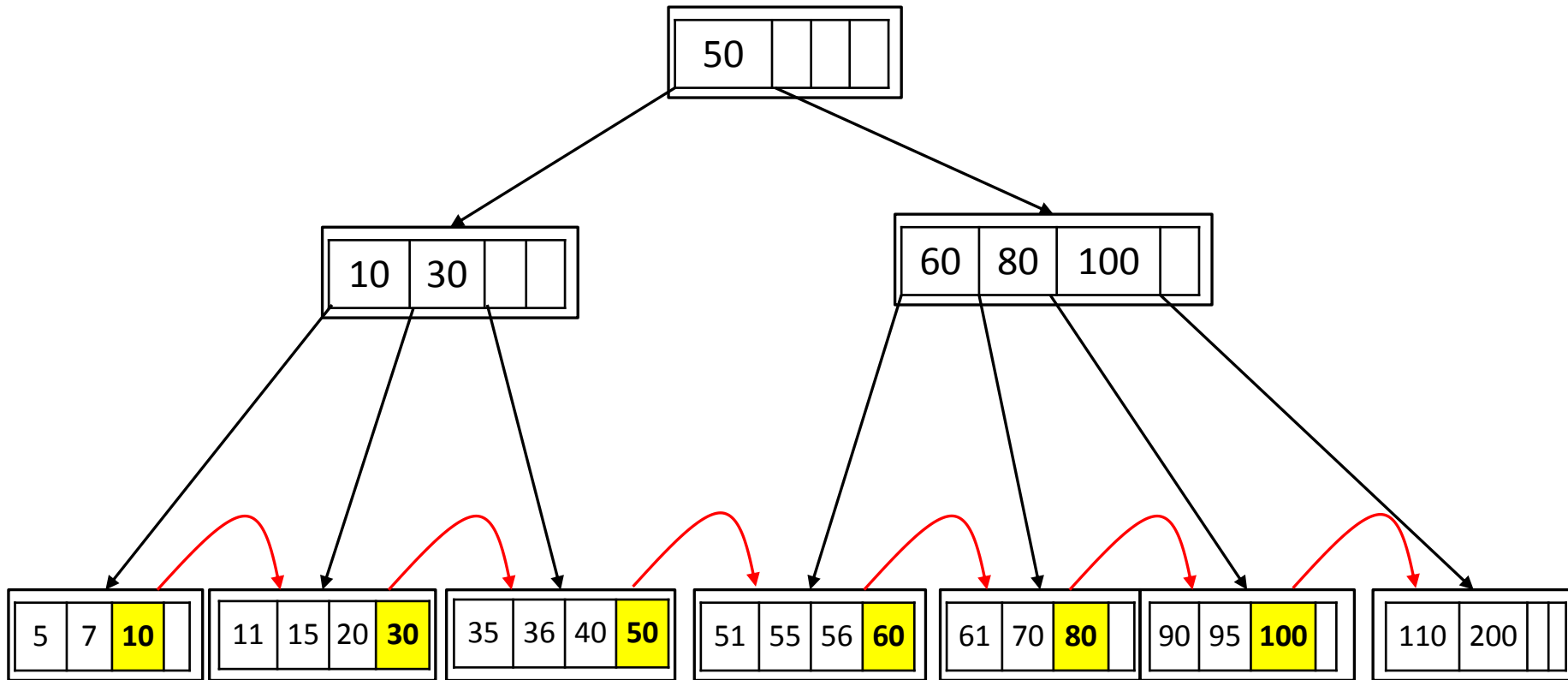
$$\sigma_{53 \leq \text{MATR} \leq 61}(\text{studenti})$$

La strategia è la seguente:

- Si parte dal valore minimo, la ricerca di 53 viene effettuata come nel caso di ricerca puntuale
- Non trovo 53 ma trovo 55 che è il primo valore utile maggiore di 53
- Basta scandire sequenzialmente le foglie sino a che non si raggiunge la foglia che può contenere il valore massimo (61) sfruttando i link alle foglie contigue

Collegamenti tra foglie

Ricerca di chiavi comprese tra 53 e 61



Ricerca di range di valori

- Si parte dalla radice e si raggiunge il valore minimo
- Da quel punto in poi si scandiscono sequenzialmente le foglie fino a quella che potenzialmente contiene il valore massimo
- Tramite i RID si risale alle tuple di interesse nell'area primaria

Variante

Un altro tipo di data entry k^* è $\langle k, lista_di_RID \rangle$

Serve quando un progettista crea un indice su un attributo che non è chiave relazionale

Ad esempio, se il B+Albero dell'esempio carica le chiavi dell'attributo "crediti superati", ogni valore può far riferimento a più record di studenti

Altro esempio: indice su attributo Città

Perché gli indici?

- L'indice è una struttura separata rispetto all'area primaria
- Posso costruire diversi indici sulla stessa area primaria:
 - Indice su matricola
 - Indice su città
 - Indici su più attributi (ad esempio coppie)
- Un DB administrator crea un indice quando il carico di lavoro in termini di interrogazioni è tale per cui il costo di accesso all'indice riduce notevolmente i tempi di risposta rispetto alla scansione sequenziale

Indici su tutto?

- E' vero che l'indice sveltisce molto le interrogazioni, ma appesantisce le modifiche
 - Le inserzioni/cancellazioni modificano gli indici
 - Le modifiche ai valori possono modificare gli indici
- Non ha senso, ad esempio, definire indici su attributi molto movimentati dagli update

Caratterizzazione degli indici

- Indice **principale**: è l'indice definito sulla chiave primaria
- Indici **secondari**: tutti gli altri (indici su attributi non chiave primaria)
- In una relazione c'è al più un indice principale, ma zero o più indici secondari
- Tutti gli indici visti finora sono esempi di **indici densi** (l'indice contiene tutti i valori distinti dell'attributo su cui è definito)

Indici densi

- L'**indice denso** è un indice con tante chiavi di ricerca quanti sono i valori distinti dell'attributo su cui è costruito
- L'indice denso permette di gestire efficacemente anche la ricerca con insuccesso
- Se non trova un valore, non c'è bisogno di andare nell'area primaria e fa solo L accessi a pagine

Indici clusterizzati

Se nell'area primaria c'è un ordinamento dei record in base ad un attributo c'è una corrispondenza tra l'ordine delle chiavi nelle foglie e l'ordine delle tuple nella pagina dell'area primaria

In questo caso abbiamo un **indice clusterizzato**

Ci può addirittura essere contiguità tra le pagine dell'indice e le rispettive pagine dell'area primaria indicizzata

Indici sparsi

Consideriamo ancora l'area primaria con record ordinati in base ad un attributo

E' possibile introdurre l'**indice sparso**:

nelle foglie dell'indice non ci sono più tutte i valori distinti della chiave di ricerca ma solo un valore della chiave (il massimo)

Nella pagina indicata dal puntatore si trovano tutte le chiavi \leq al valore massimo ma strettamente maggiori della chiave del nodo padre

Data entry $\langle k, \text{tupla} \rangle$

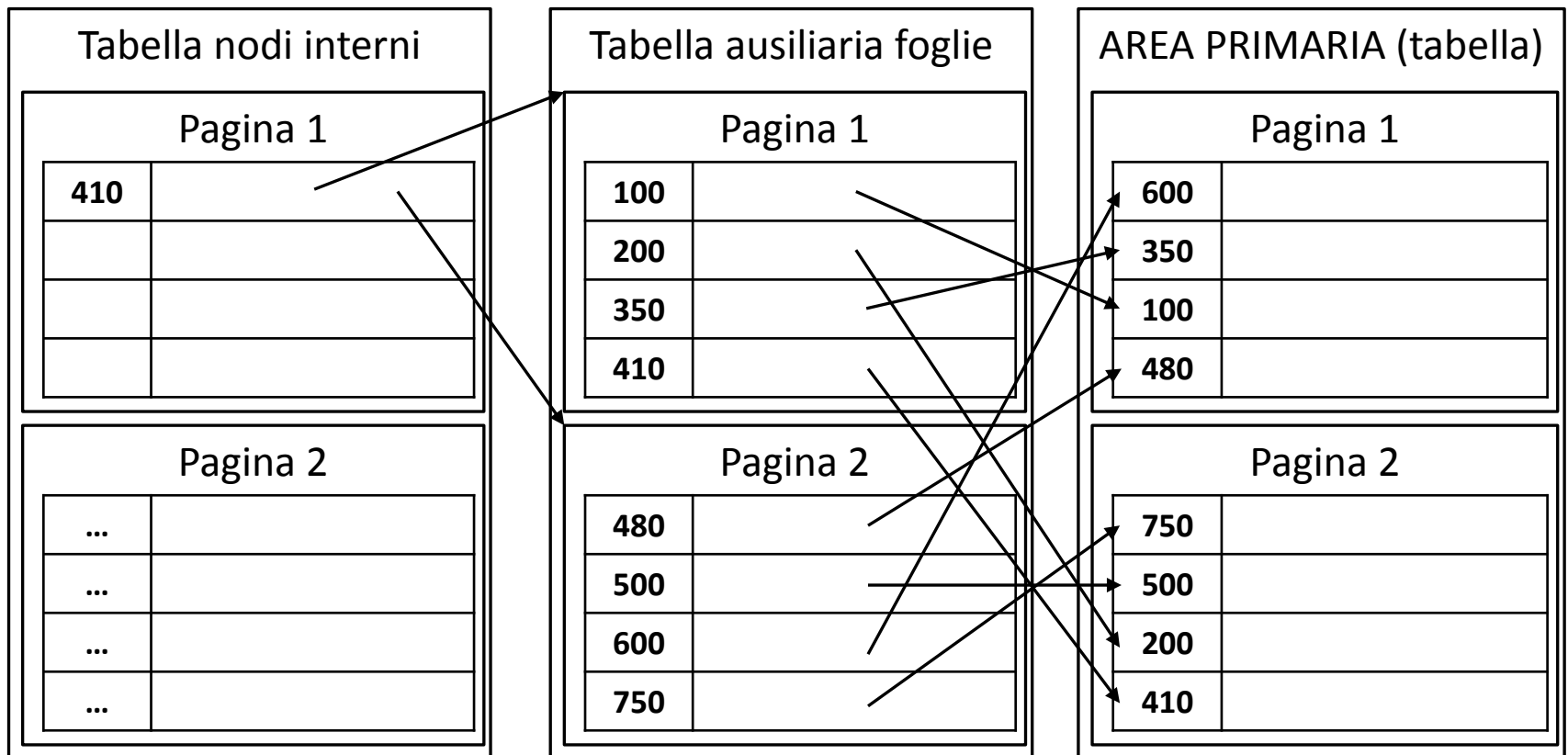
- Un indice clusterizzato/sparso può essere fuso con l'area primaria, ovvero l'organizzazione dell'area primaria è effettuata come un B+Albero
- Alcune pagine conterranno delle indicizzazioni, altre pagine conterranno le tuple
- Le foglie del B+Albero diventano le pagine dati vere e proprie
- L'indice può essere definito, ovviamente, solo su un attributo
- In questo caso il costo di accesso è esattamente L

Tipi diversi di data entry K^*

- **<k,tupla>**: il data entry è la tupla stessa
- **<k,RID>**: il data entry è puntatore al record nell'area primaria
- **<k,lista_di_RID>**: il data entry è una lista di puntatori a record che condividono la stessa chiave k

Metodi di accesso tabellare

Se ci astraiamo dalla fatto che l'indice sia un B+Albero, la struttura generica degli indici può si rappresenta con tabelle



Split ed efficacia nel B+Albero

Mediamente, quanti split bisogna realizzare per introdurre tutte le N chiavi nell'indice?

Partiamo dall'ipotesi di conoscere il numero di nodi N_O dell'albero (foglie comprese)

Il numero di split è $n_{split} \leq N_O - 1$

- $N_O - 1$ è il numero di split per ottenere N_O nodi
- Ma $n_{split} \leq N_O - 1$ perché per ogni split genero un nuovo nodo, tranne per la radice, per cui genero due nuovi nodi

Split ed efficacia nel B+Albero

Consideriamo ora l'ipotesi già fatta di albero scarico

Quando l'albero è scarico, ho solo il numero minimo di chiavi per ogni nodo, calcolo quindi il numero di chiavi

- Per la radice ho 1 sola chiave
- Per tutti gli altri nodi ho $\lceil m/2 \rceil - 1$ chiavi

Quindi ho in tutto

$$1 + (\lceil m/2 \rceil - 1) \cdot (N_o - 1) \leq N$$

(minore per via dei residui) da cui ricavo

$$(N_o - 1) \leq \frac{N - 1}{\lceil m/2 \rceil - 1}$$

Split ed efficacia nel B+Albero

Sappiamo che il numero di split è $n_{split} \leq N_O - 1$, quindi

$$n_{split} \leq (N_O - 1) \leq \frac{N - 1}{\lceil m/2 \rceil - 1}$$

Possiamo fare una maggiorazione

$$n_{split} < \frac{N}{\lceil m/2 \rceil - 1}$$

Da cui otteniamo il numero medio di split **per ogni** inserzione

$$\frac{n_{split}}{N} < \frac{1}{\lceil m/2 \rceil - 1}$$

Split ed efficacia nel B+Albero

Con $m=100$ ho in media uno split ogni 50 inserzioni

In realtà il numero di split è molto inferiore perché intervengono i bilanciamenti che cercano di compensare i fenomeni di caricamento minimo dei nodi

Stima dei costi

Adottando la scansione sequenziale della tabella, il DBMS mediamente legge la metà delle pagine di cui è costituita la tabella

Vediamo ora come l'ottimizzatore fisico stima il costo dell'accesso attraverso l'indice

Stima dei costi

Ricordiamo che il DBMS, nel dizionario dati, conserva un insieme di statistiche sulla base di dati

- Per ogni attributo il dizionario conserva il numero di valori distinti dell'attributo A nella tabella T : $VAL(A,T)$
- Se c'è un indice, il dizionario conserva anche il numero di pagine che costituiscono le foglie dell'indice (ci riferiamo all'indice denso): $N_{foglie}(A,T)$
- Abbiamo la cardinalità della tavola T : $CARD(T)$
- Abbiamo il numero di pagine di una tabella: $N_{page}(T)$

Stima dei costi

Ricordiamo anche il fattore di selettività di un predicato

Ricordiamo che quando si usa un indice ci si riferisce sempre a predicati del tipo

- $A = v$
- $v_1 \leq A \leq v_2$

Predicato p	f_p
$A = v$	$1/\text{VAL}(A, T)$
$v_1 \leq A \leq v_2$	$(v_2 - v_1)/(\text{MAX}(A, T) - \text{MIN}(A, T))$

Modello di costo

Il modello di costo utilizzato dagli ottimizzatori fisici segue la seguente formula additiva:

$$C_A = C_I + C_D$$

C_I : costo di accesso all'indice, ovvero costo per arrivare alla foglia dell'indice

C_D : costo di accesso ai dati, ovvero costo di accesso alla pagina dei dati tramite il RID

Costo di accesso all'indice C_I

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

- Dopo aver individuato la foglia che contiene il minimo, l'algoritmo di ricerca esegue una scansione sequenziale delle foglie fino a trovare quella che contiene il massimo
- L'algoritmo non tocca le foglie che precedono né quelle che seguono, ma solo le foglie che contengono le chiavi di interesse

Costo di accesso all'indice C_I

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

- Il numero di chiavi di interesse (numero di chiavi che soddisfano il predicato) è stimato con:

$$f_p \cdot \text{VAL}(A, T)$$

- Bisogna quindi contare quante pagine sono state toccate in questa esplorazione

Costo di accesso all'indice C_I

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

- Per questo bisogna conoscere il numero medio di valori chiave per ogni foglia
- Il numero di pagine toccate è stimato da:

$$\lceil f_p \cdot \text{VAL}(A, T) / \text{numero medio di chiavi per foglia} \rceil$$

Ma il numero medio è facilmente calcolabile

Costo di accesso all'indice C_I

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

Calcolo del numero medio di valori chiave per foglia:

$$VAL(A,T)/N_{foglie}(A,T)$$

Il costo dell'indice diventa quindi:

$$C_I = \lceil f_p \cdot VAL(A,T) / (VAL(A,T) / N_{foglie}(A,T)) \rceil = \lceil f_p \cdot N_{foglie}(A,T) \rceil$$

Costo di accesso all'indice C_I

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

$$C_I = \lceil f_p \cdot N_{\text{foglie}}(A, T) \rceil$$

N.B.: il costo di attraversamento dell'albero ($L - 1$) è trascurabile nei sistemi attuali in cui abbiamo un L uguale a 2 o 3, inoltre si presume che la radice dell'indice sia già **nel buffer**

Il costo è tutto sulla **navigazione delle foglie**

Costo di accesso ai dati C_D

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

Devo stimare il numero di RID che utilizzo per la mia interrogazione (potrei avere anche più RID per ogni chiave, se l'attributo non è chiave primaria)

Posso calcolare facilmente il numero di accessi (numero medio di **registrazioni**) nell'area primaria come:

$$E_{\text{reg}} = f_p \cdot \text{CARD}(T)$$

Costo di accesso ai dati C_D

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

Il costo di accesso ai dati è quindi

$$C_D = E_{\text{reg}} = f_p \cdot \text{CARD}(T)$$

che stima il numero di accessi all'area primaria

E' però una stima per eccesso, infatti più RID possono puntare ad una stessa pagina che, dopo la prima lettura, si trova già nel buffer

Costo di accesso ai dati C_D

Consideriamo l'interrogazione range, che è più generale

$$p = v_1 \leq A \leq v_2$$

In realtà tutti i DBMS, prima di utilizzare il singolo RID, collezionano tutti i RID estratti dall'interrogazione, vanno su una pagina ed estraggono tutte le tuple corrispondenti a RID presenti nella collezione

La stima data dal C_D è quindi in eccesso

Formula di Cardenas

I DBMS correggono la stima attraverso un procedimento statistico che porta alla **formula di Cardenas**, con due parametri:

$$\Phi(E_{\text{reg}}, N_{\text{page}}(T))$$

- E_{reg} : numero di registrazioni/read (per semplicità k)
- $N_{\text{page}}(T)$: numero di pagine dell'area primaria (lo chiamo n)

Riscriviamola quindi come $\Phi(k, n)$

La formula di Cardenas è basata sulla distribuzione uniforme per cercare di stimare mediamente quante pagine bisogna estrarre dall'area primaria per risolvere l'interrogazione

Formula di Cardenas

- Dato un RID qualsiasi non so in quale pagina si trova, quindi, dato un RID immagino che ogni pagina abbia la stessa probabilità $1/n$ di contenere una tupla con quel RID
- $(1 - 1/n)$ è la probabilità che una pagina **non** contenga il RID cercato (dato un RID qualsiasi)
- $(1 - 1/n)^k$ diventa la probabilità che una pagina ha di **non** contenere **nessun** RID risolutivo per l'interrogazione
- $[1 - (1 - 1/n)^k]$ è la probabilità che una pagina ha di contenere **almeno un** RID
- dato che abbiamo n pagine, la formula di Cardenas è:

$$\Phi(k,n) = n \cdot [1 - (1 - 1/n)^k]$$

Formula di Cardenas

La formula di Cardenas

$$\Phi(k,n) = n \cdot [1 - (1 - 1/n)^k]$$

stima il numero di pagine interessate (che possono contenere uno o più RID) dall'interrogazione e ci dà la vera stima di accesso ai dati utilizzata dal DBMS

Si può approssimare la formula di Cardenas grazie al suo andamento asintotico da cui si evince che

$$\Phi(k,n) \leq \min\{k,n\}$$

I DBMS usano quest'approssimazione e scelgono il minimo tra numero di read previsto e il numero di pagine dell'area primaria

Metodo di accesso diretto

Anche detto metodo di accesso procedurale
statico/dinamico

L'obiettivo di questo metodo è di ottenere il dato con
circa un accesso

Si fa in modo che l'applicazione che ha bisogno di accedere frequentemente ad un dato possa accedervi con un numero minimo di accessi (ad esempio, aggiornamento del saldo di un conto corrente)

Organizzazione dati

In questa modalità le pagine vengono chiamate bucket

- bucket sono numerati da 0 a $m - 1$
- In tutto ci sono m bucket

Per accedere si usa una funzione di hashing h , che applicata al valore k della chiave, genera in maniera deterministica l'indirizzo del bucket in cui trovare la tupla

AREA PRIMARIA (tabella)	
Bucket 0	
600	
350	
100	
480	
...	
Bucket $m - 1$	
750	
500	
200	
410	

Uso della funzione di hashing

La funzione di hashing viene usata negli inserimenti

- calcolo la funzione di hashing per la chiave della tupla che voglio inserire
- il risultato mi dice in quale bucket inserire la tupla

Viene ovviamente usata anche nelle interrogazioni

- calcolo la funzione di hashing per la chiave di ricerca
- il risultato mi dice in quale bucket troverò la tupla corrispondente alla chiave di ricerca

Funzione di hashing

La funzione h di hash deve avere caratteristiche precise

- h deve essere **suriettiva** (devo poter sfruttare tutte le pagine disponibili) quindi per ogni valore del risultato della funzione di h (codominio) deve esistere un valore della chiave d'accesso che punta su quel bucket
- h deve sfruttare al meglio tutto lo spazio disponibile, che dipende dal numero m di bucket e dalla capacità c del bucket (in termini di tuple), quindi $m \cdot c$

Consideriamo il metodo di accesso procedurale **statico**

Funzione di hashing

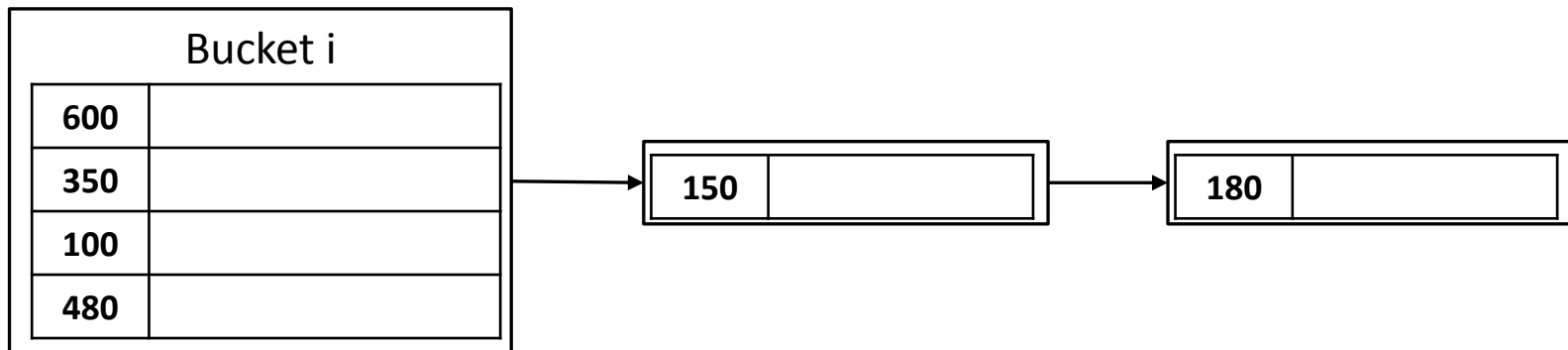
La funzione h deve fare in modo di distribuire in modo uniforme le varie tuple nei vari bucket in modo che questi ultimi si riempiano in modo abbastanza uniforme

Infatti se una funzione h crea squilibrio saturando alcuni bucket e lasciando vuoti gli altri, il sistema degrada in prestazioni

Collisioni di chiave

La funzione h manda chiavi distinte nello stesso bucket, può quindi succedere che $h(k_1)=h(k_2)$ (collisione di chiave)

Quando un bucket i si satura e arriva una nuova chiave k che dà come risultato $h(k)=i$, nei sistemi con metodo procedurale statico si iniziano ad usare le **liste di overflow**



Degrado delle prestazioni

Quando si effettua una ricerca di un record in base ad una chiave k , e la funzione di h restituisce un bucket in cui c'è una lista di overflow, se la chiave non è presente nel bucket, l'algoritmo di ricerca continua a cercare nella lista di overflow

Il costo di accesso diventa maggiore di 1

Il costo medio è in realtà pari a $1+\delta$

Minimizzazione del δ

Considerando N chiavi possibili, bisogna porre la condizione che la capacità totale del sistema sia maggiore di N , cioè, $m \cdot c \geq N$

Chiamiamo $d = N/(m \cdot c)$ il **fattore di caricamento** del sistema, vogliamo quindi $d \leq 1$

La condizione non è però sufficiente a garantire che non esistano bucket pieni

Funzione di hash

Bisogna imporre che la funzione h abbia la caratteristica dell'uniformità (distribuzione uniforme) ovvero, data una chiave qualsiasi k , la probabilità che h generi un valore i qualsiasi tra 0 e $m - 1$ è sempre $p_i = 1/m$

E' noto che la funzione di congruenza lineare

$$h(k) = (a \cdot k + b) \bmod m$$

è la funzione migliore per generare i valori di indirizzi di bucket (k è un intero positivo, ma non è una limitazione, posso sempre ricondurmi ad interi positivi)

Ci sono poi delle condizioni particolari su a e b

Funzione di hash

Nell basi di dati si adotta spesso una forma semplificata

$$h(k) = k \bmod m$$

dove però m è un numero primo piuttosto grande (in genere > 20) o un prodotto di pochi numeri primi

Stima del costo

Esiste un modello probabilistico basato sulla distribuzione di Poisson che permette di identificare alcuni valori di costo, dati il fattore di caricamento d e la capacità media c di ogni bucket

C/d	50%	70%	90%
20	1,00	1,02	1,27
50	1,00	1,00	1,08
100	1,00	1,00	1,04

Il DBMS usa queste tabelle

Un fattore di caricamento basso, pur abbassando i tempi di caricamento, comporta uno spreco di spazio, mentre i valori per un caricamento intorno al 70% sono accettabili

Indici hash

Questa stessa organizzazione può essere utilizzata per gli indici, in questo caso parliamo di **indici hash**

I bucket contengono il valore della chiave più i RID

Gli indici hash non vengono quasi mai utilizzati perché hanno costi confrontabili con quelli basati sui B+Alberi, ma ammettono soltanto la ricerca puntuale, non quella di range

Accesso procedurale dinamico

- Nel caso dei metodi di accesso procedurale dinamico non si fissa il numero di bucket
- Si dà una stima iniziale del numero di bucket e poi si autoadattano
- Ci sono algoritmi di autoadattamento che permettono di mantenere il giusto equilibrio tra fattore di caricamento e prestazioni

Quando definire gli indici?

Consideriamo un esempio

STUDENTI(MATR,Nome,DataNascita,Genere,Indirizzo)

ESAMI(MATR,Corso,Voto,DataEsame)

Ricordiamo che, se è vero che un indice favorisce le interrogazioni, ha comunque un costo di mantenimento per le inserzioni, cancellazioni e update

Esistono delle euristiche generali per capire se conviene o meno definire un indice

Euristiche/consigli

- 1. Evitare gli indici su tabelle di poche pagine**
2. Evitare indici su attributi volatili (ad esempio, Saldo del conto corrente)
- 3. Evitare indici su chiavi poco selettive**
- 4. Evitare indici su chiavi con valori sbilanciati**
- 5. Limitare il numero di indici**
- 6. Definire indici su chiavi relazionali ed esterne**
- 7. Gli indici velocizzano le scansioni ordinate**
- 8. Usare l'indice hash solo per interrogazioni puntuali**
- 9. Conoscere a fondo il DBMS**

1. Indici su tabelle piccole

Una tabella relazionale di poche pagine (~10 pagine) non ha esigenza di indici, perché le memorie sono tali per cui queste tabelle possono essere caricate interamente nei buffer

3. Selettività dell'indice

Ricordiamo che il costo d'accesso legato all'indice è dato da

$$C_A = C_I + C_D$$

con un certo costo per raggiungere la foglia

$$C_I = f_p \cdot N_{\text{foglie}}$$

(nella foglia trovo il valore della chiave più la lista di RID che rinviano all'area primaria)

Il costo dell'accesso ai dati è dato dalla formula di Cardenas

$$C_D = \Phi(N_{\text{page}}(T), E_{\text{reg}}) = \min(N_{\text{page}}(T), E_{\text{reg}})$$

dove $E_{\text{reg}} = f_p \cdot \text{CARD}(T)$

3. Selettività dell'indice

La selettività dell'indice è data dal rapporto:
pagine restituite per $\sigma_{k=\text{costante}}(T)$ / pagine di T

che si calcola come

$$f_s = \Phi(N_{\text{page}}(T), E_{\text{reg}}) / N_{\text{page}}(T)$$

L'euristica suggerisce di definire un indice su k se

$$f_s < 20\%$$

3. Esempi

STUDENTI(MATR,Nome,DataNascita,Genere,Indirizzo)

ESAMI(MATR,Corso,Voto,DataEsame)

- Un indice sulla chiave MATR ha un'altissima selettività quindi va bene
- Può aver senso mettere un indice sulle date di nascita
- Non ha nessun senso mettere un indice sul genere (solo due valori possibili): il DBMS non userà mai l'indice sul genere

4. Attributi sbilanciati

Bisogna fare attenzione alla distribuzione dei valori

- Se c'è un grosso squilibrio nella distribuzione dei valori di una chiave gli indici su quella chiave saranno poco efficienti
- Ricordiamo infatti che l'ipotesi su cui si basano gli indici è quella della distribuzione uniforme

5. Quanti indici definire?

- Non ci sono indicazioni di carattere generale
- Gli indici selettivi favoriscono le interrogazioni ma la loro manutenzione costa
- Un'indicazione di massima è di limitarsi al massimo a 4/5 indici per relazioni corpose

6. Indici sulle chiavi

Un indice su una chiave è sempre consigliato

Analogamente si consiglia di definire indici su chiavi esterne, perché spesso ci sarà bisogno del join

Esempio:

studenti ⋈_{studenti.MATR=esami.MATR} *esami*

In ESAMI, MATR non è chiave relazionale (la chiave è MATR,Corso) ma si consiglia ugualmente di definire un indice

7. Scansione ordinata

- Quando consideriamo le foglie del B+Albero, i valori delle chiavi sono sempre ordinati per definizione
- Di conseguenza, se si vuole esplorare l'area dati secondo l'ordine della chiave indicizzata k è sufficiente eseguire una scansione sequenziale delle foglie
- L'indice ha anche come impiego il reperimento delle tuple dell'area primaria secondo l'ordine della chiave indicizzata

8. Indice hash

L'indice hash è utile solo quando ci si limita ad interrogazioni puntuali o quando l'attributo con l'indice è utilizzato con l'equi-join

9. Conoscenza del DBMS

Si raccomanda di scegliere gli indici conoscendo a fondo le strategie di ottimizzazione del DBMS

E' l'ottimizzatore del DBMS a scegliere se utilizzare o meno l'indice e ogni DBMS ha la sua strategia