

# Indici

**Leggere Cap 5 Riguzzi et al. Sistemi Informativi**

**Lucidi derivati da quelli di Hector Garcia-Molina**

# Tipi di indice

- indice primario:
  - Indice su un campo in base al quale e' ordinato il file
  - Sono anche detti indici clustered o clustering
- indice secondario
  - Indice su un campo sul quale il file non e' ordinato
- indice denso:
  - contiene un record per ciascun record del file
- indice sparso:
  - contiene un record per ogni pagina del file

# Tipi di strutture ad indice

- ISAM: Indexed Sequential Access Method
  - Definito dall'IBM
  - Il piu' antico
- B+-Tree
  - Piu' recente
  - Piu' efficiente
- Vediamo prima ISAM

# ISAM: Indici primari

File sequenziale ordinato

10	
20	

30	
40	

50	
60	

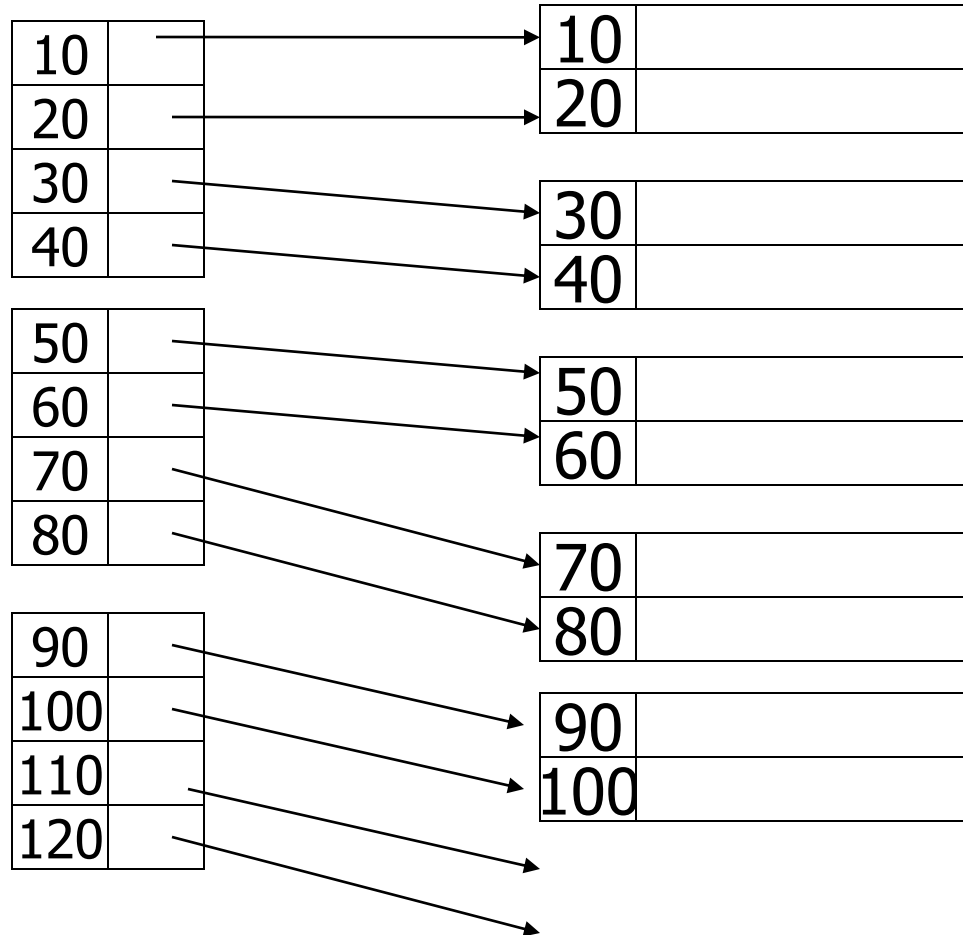
70	
80	

90	
100	

# Indici primari

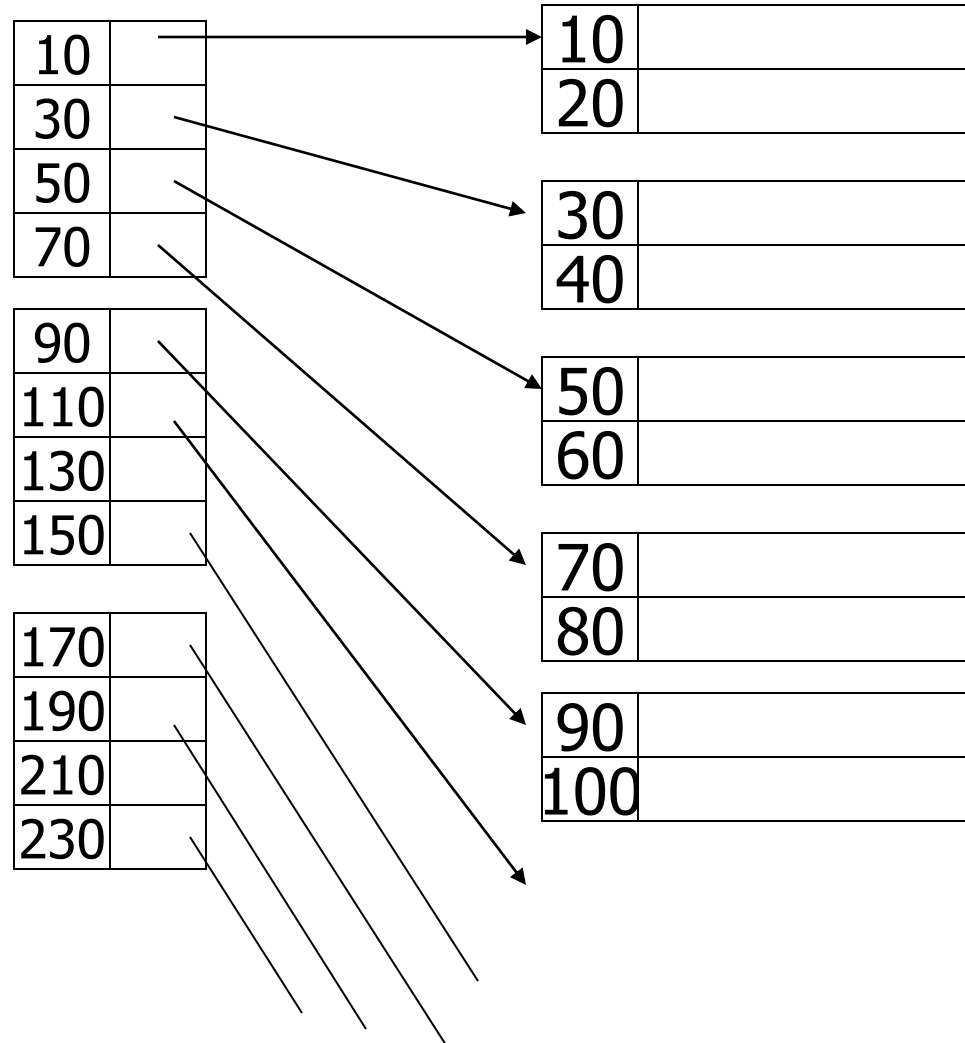
Indice denso

File sequenziale ordinato



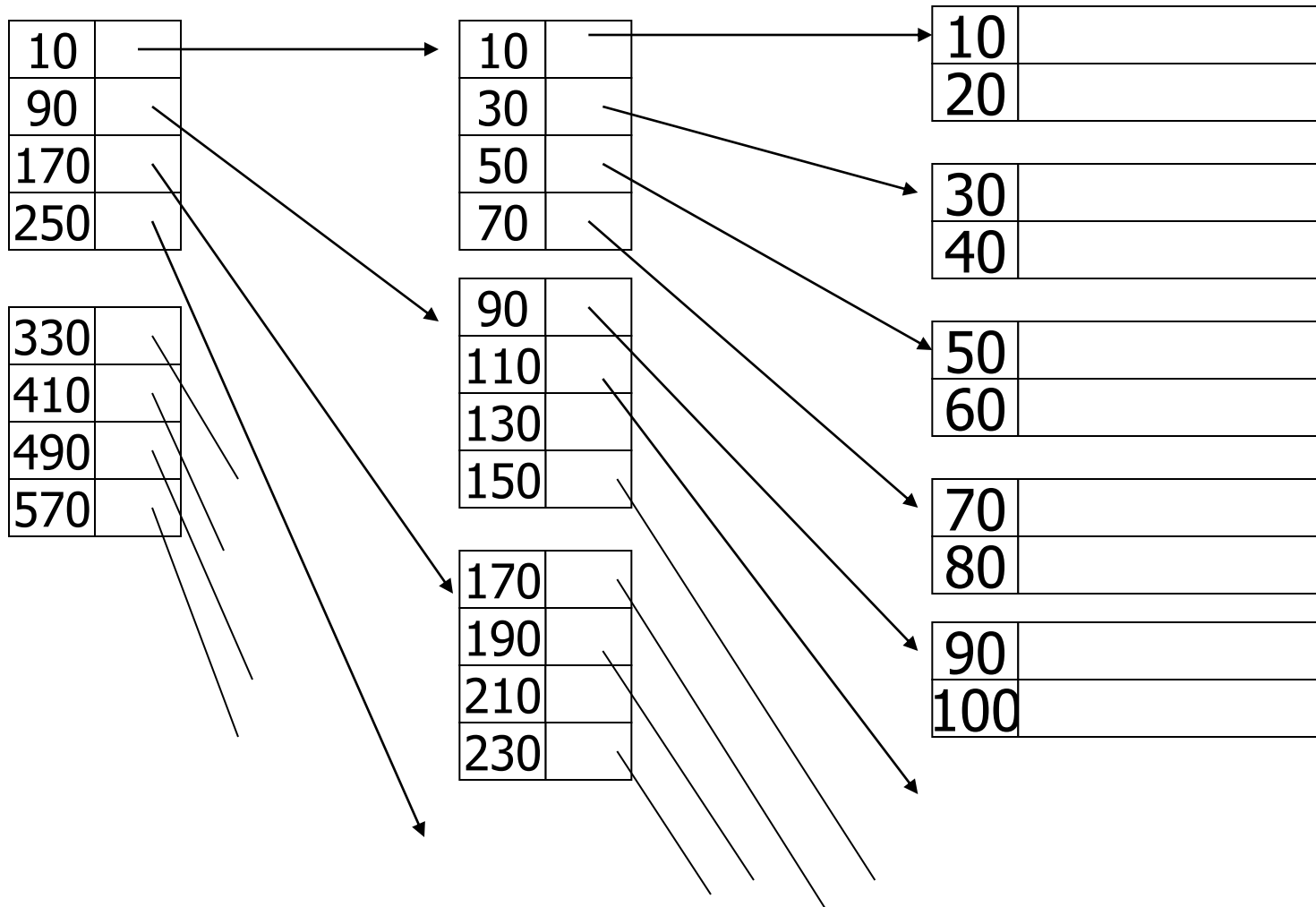
## Indice sparso

## File sequenziale ordinato



## 2o livello sparso

## File sequenziale ordinato



# Tipi di indice, commenti

- Sparso: indice più piccolo, se ne può tenere una maggior parte in memoria
- Denso: si può determinare se un record è presente senza accedere al file
- Un indice primario può essere denso o sparso
- Uno secondario deve essere denso



## Tipi di indice, commenti

- Ogni file può avere al più un indice primario e un numero qualunque di indici secondari (su campi diversi).

# Chiavi duplicate



10	
10	

10	
20	

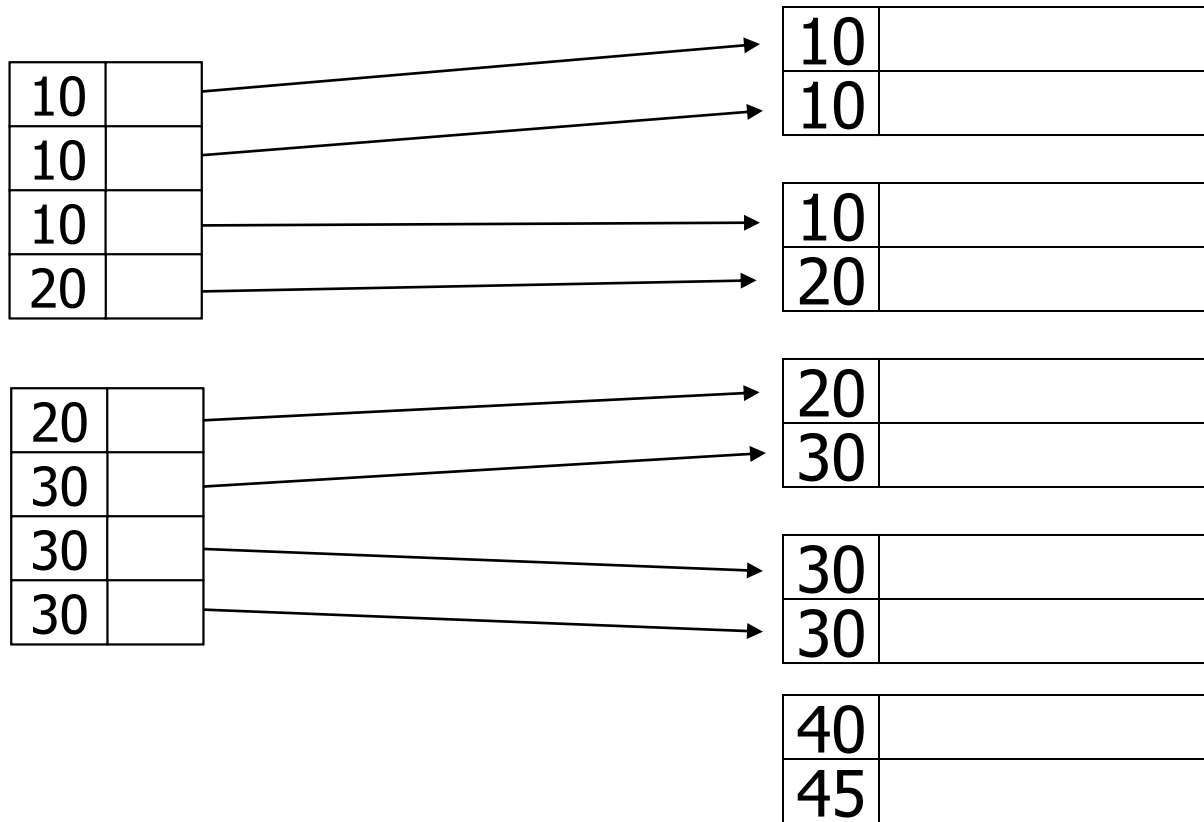
20	
30	

30	
30	

40	
45	

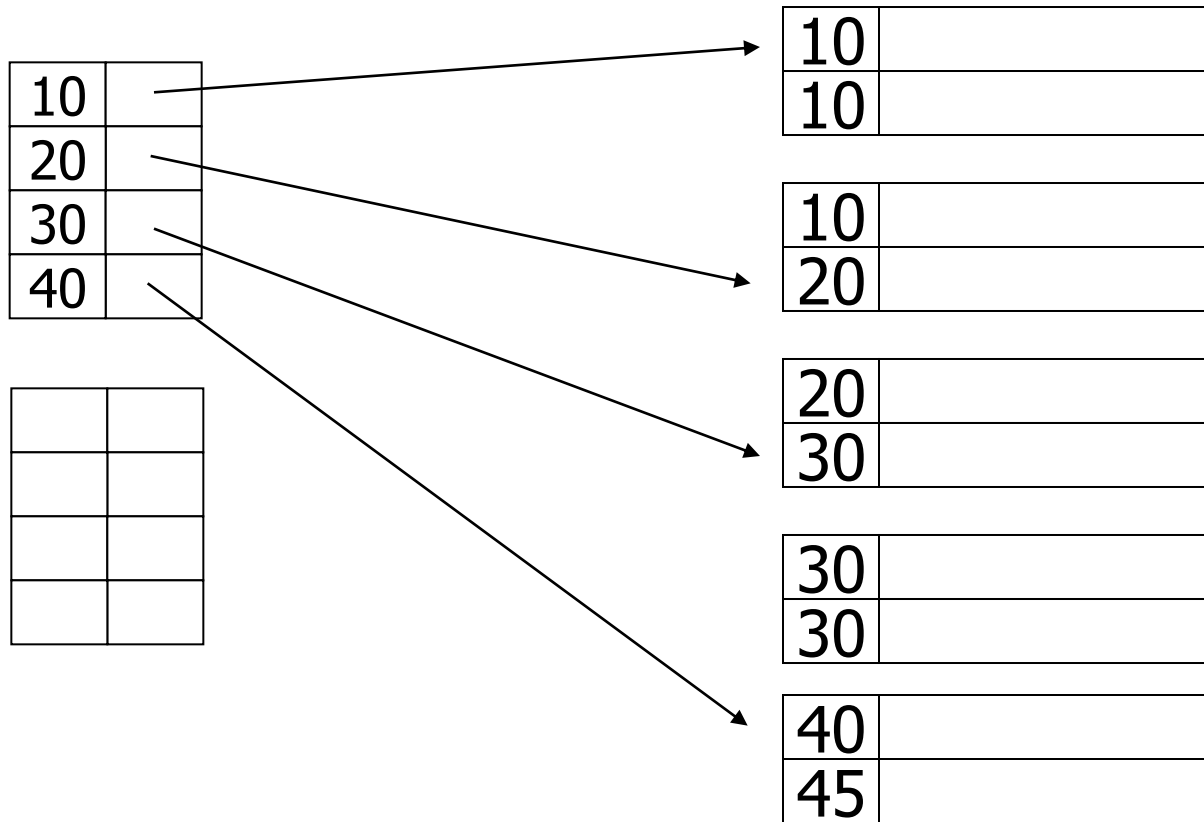
# Chiavi duplicate

## Indice denso



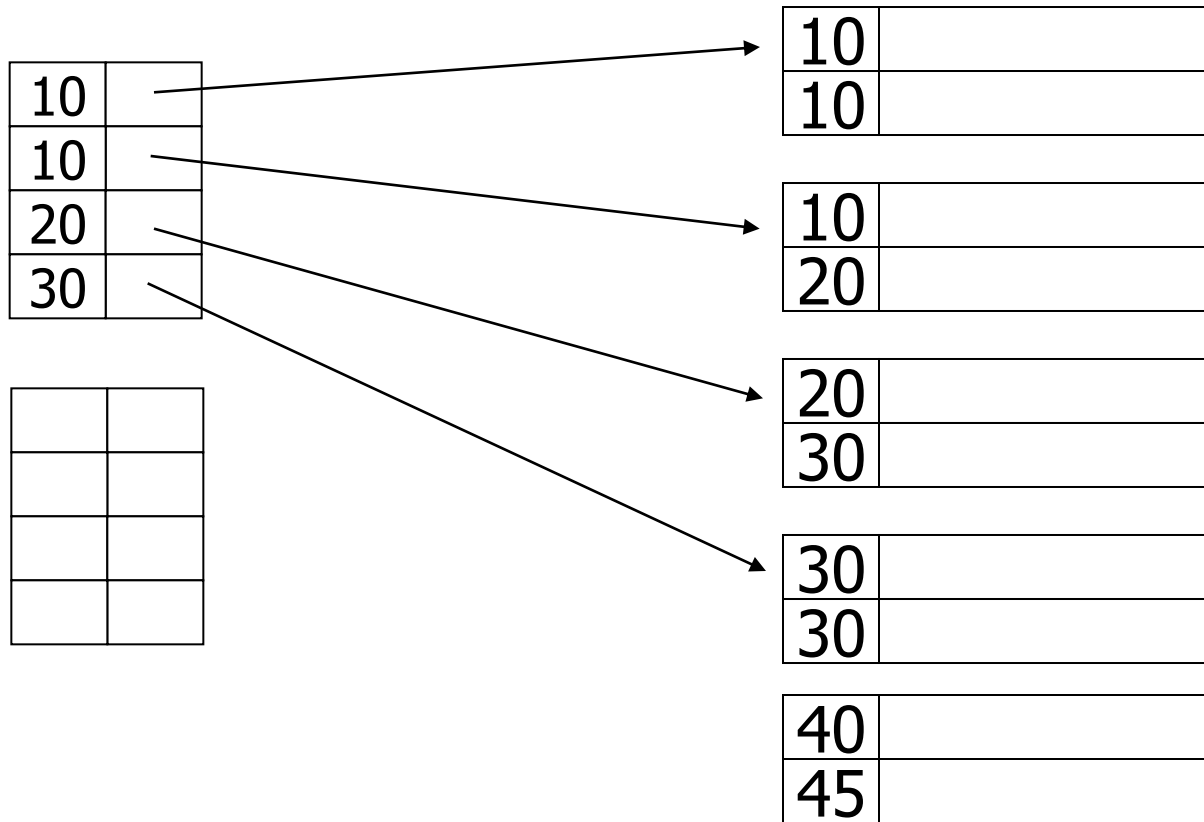
# Chiavi duplicate

Indice denso, metodo migliore

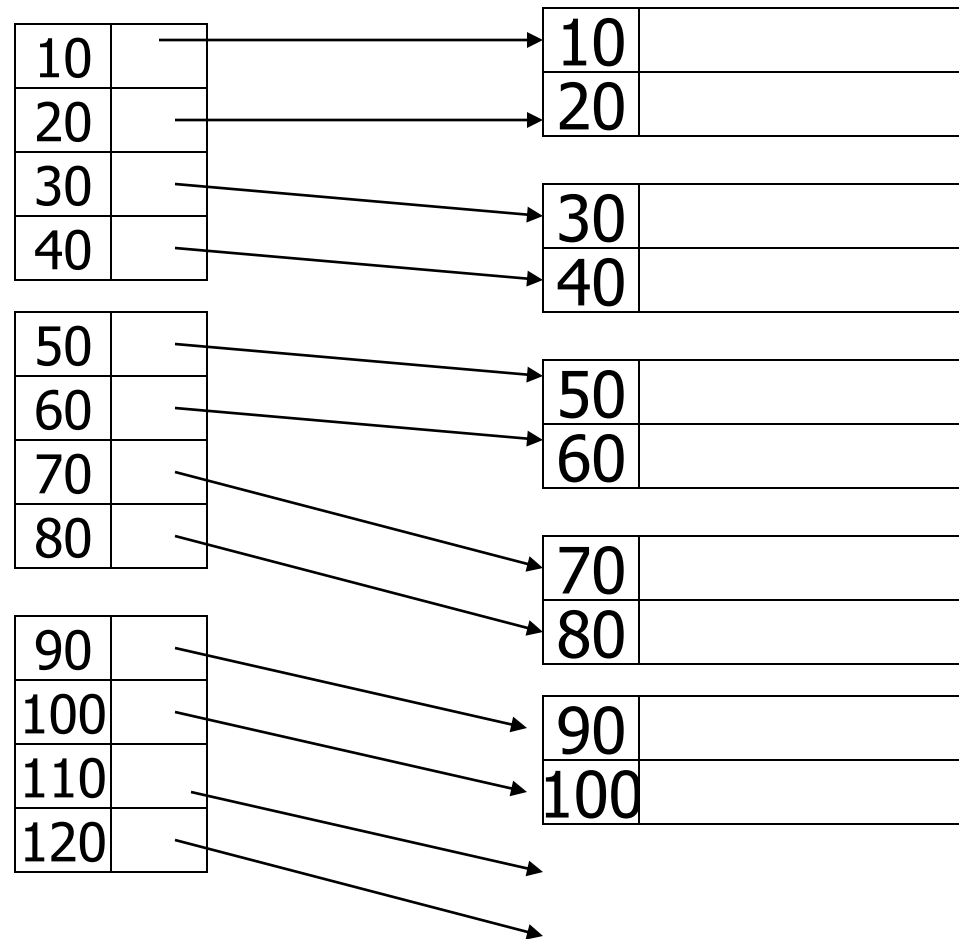


# Chiavi duplicate

## Indice sparso

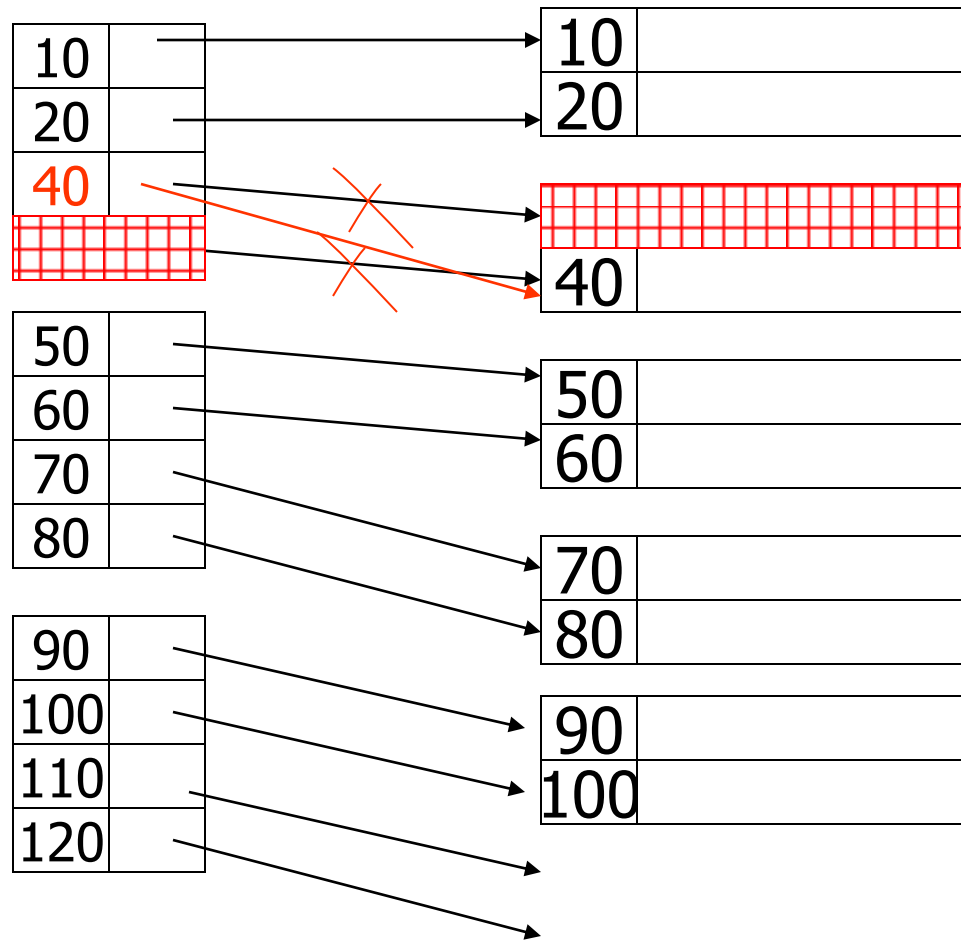


# Cancellazione da un indice denso

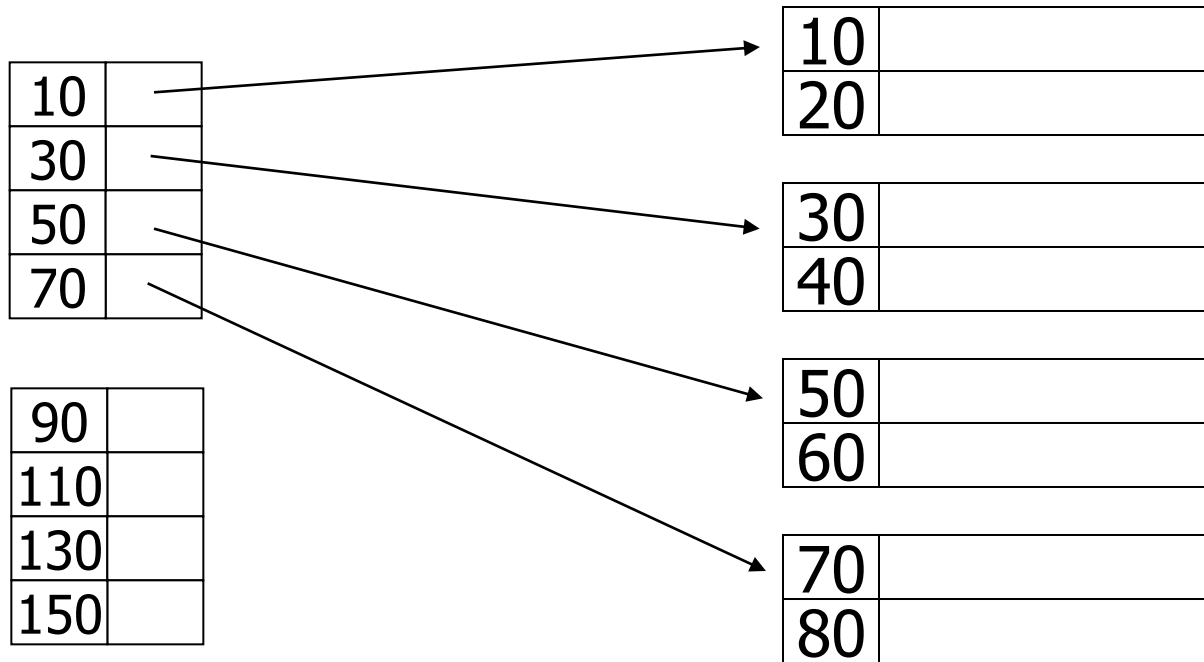


# Cancellazione da un indice denso

– cancellazione del record 30



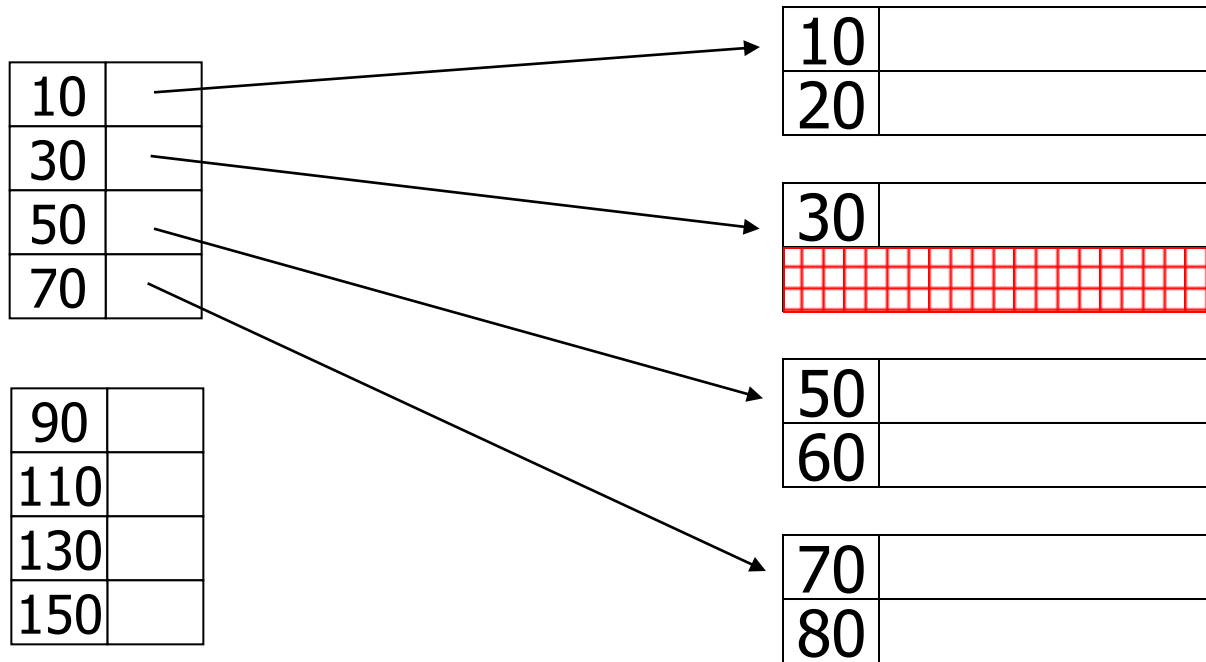
# Cancellazione da un indice sparso





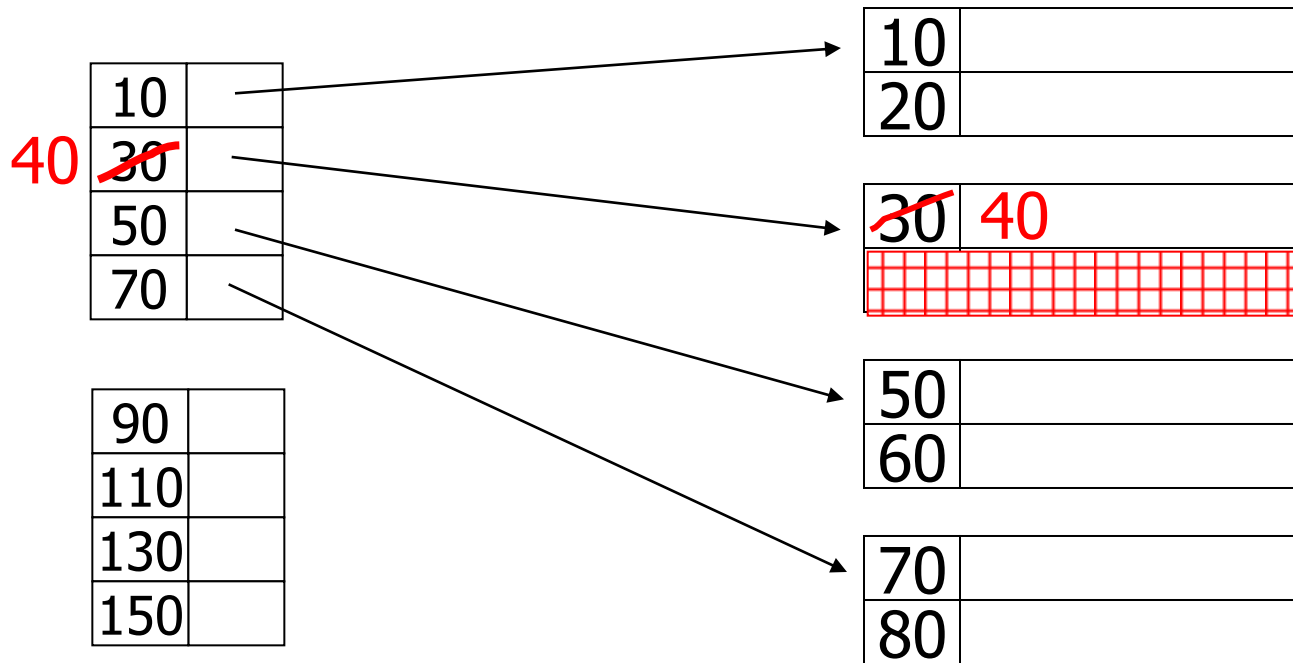
# Cancellazione da un indice sparso

– cancellazione del record 40



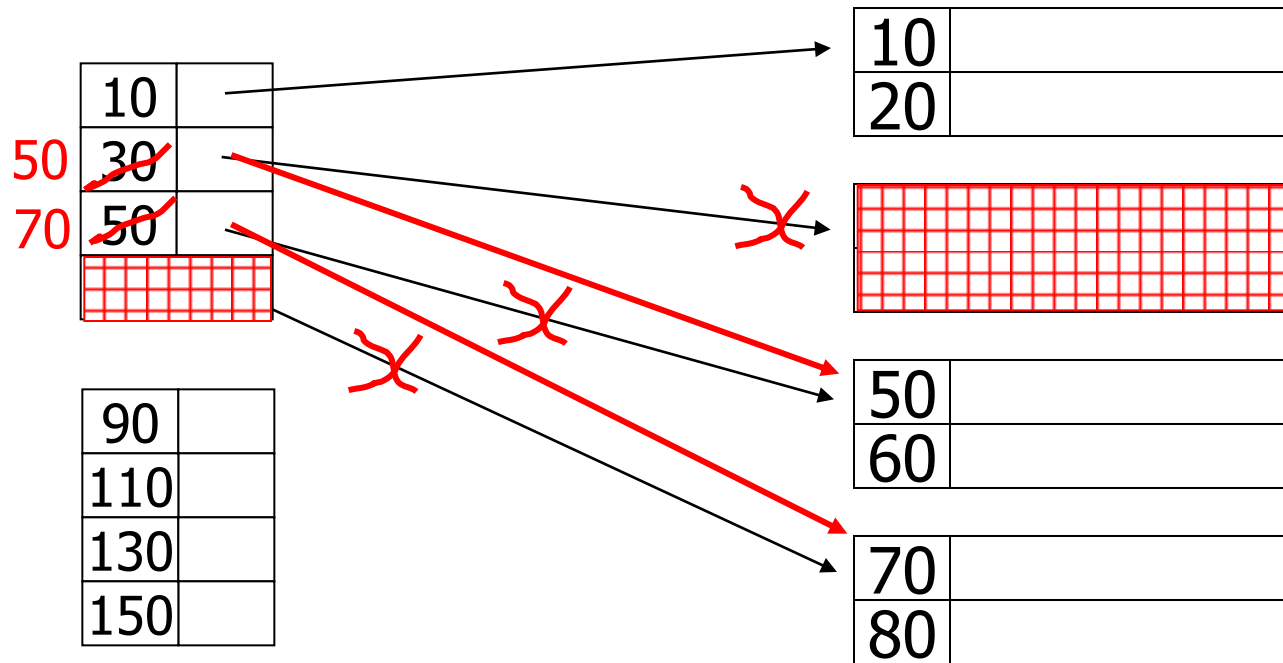
# Cancellazione da un indice sparso

– cancellazione del record 30

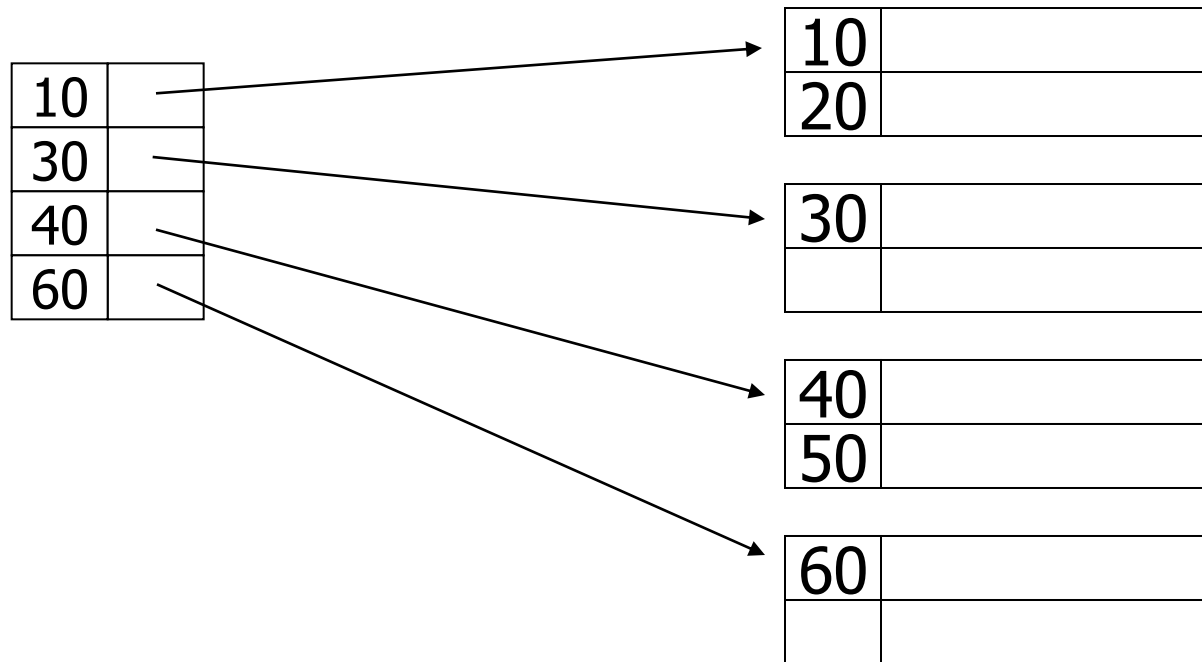


# Cancellazione da un indice sparso

– cancellazione dei records 30 & 40



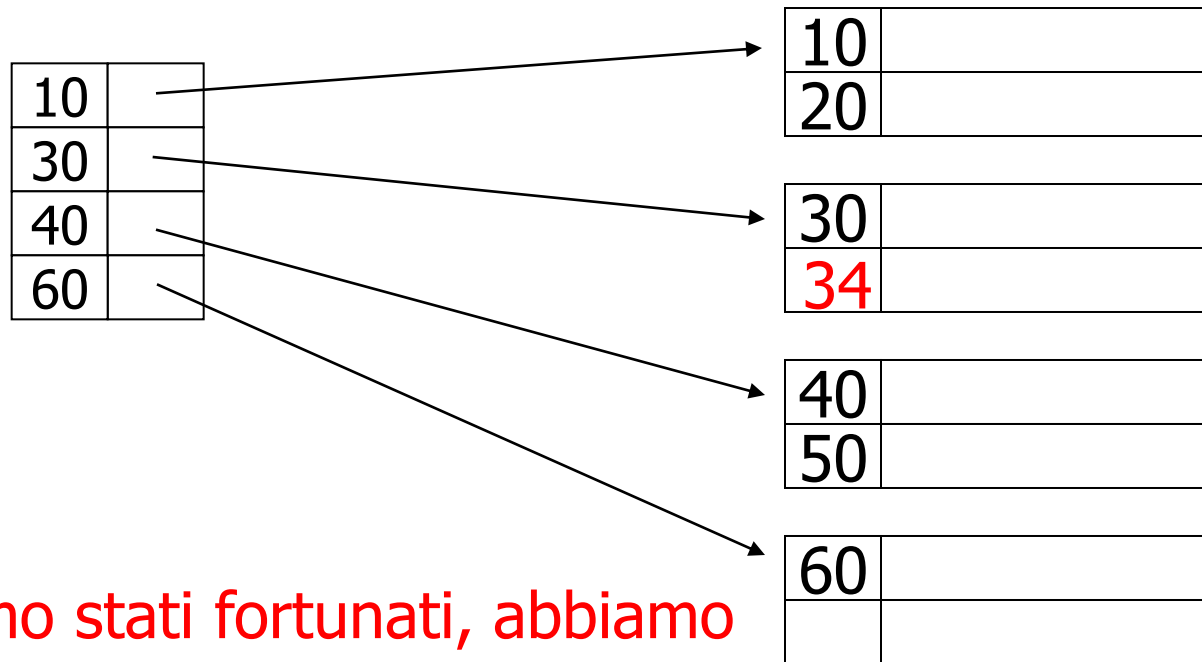
# Inserimento, indice sparso



Il file viene creato all'inizio lasciando un po' di spazio vuoto (tipicamente il 20%)

# Inserimento, indice sparso

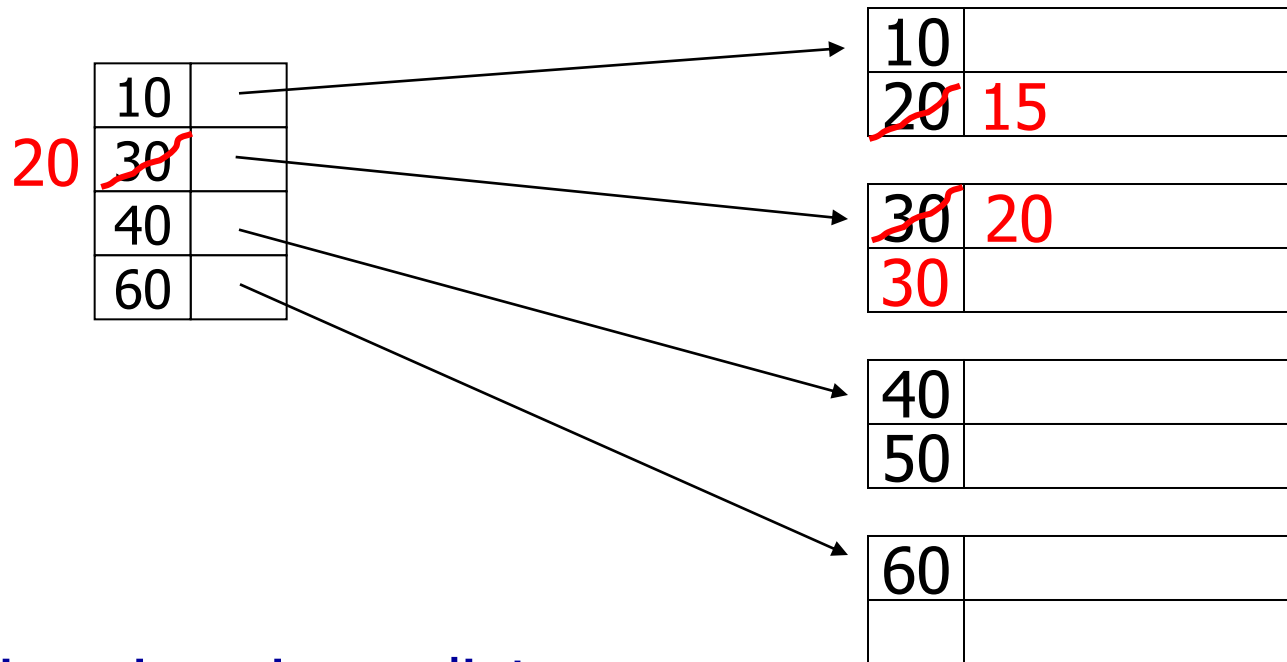
– inserisci record 34



• siamo stati fortunati, abbiamo trovato spazio dove serviva

# Inserimento, indice sparso

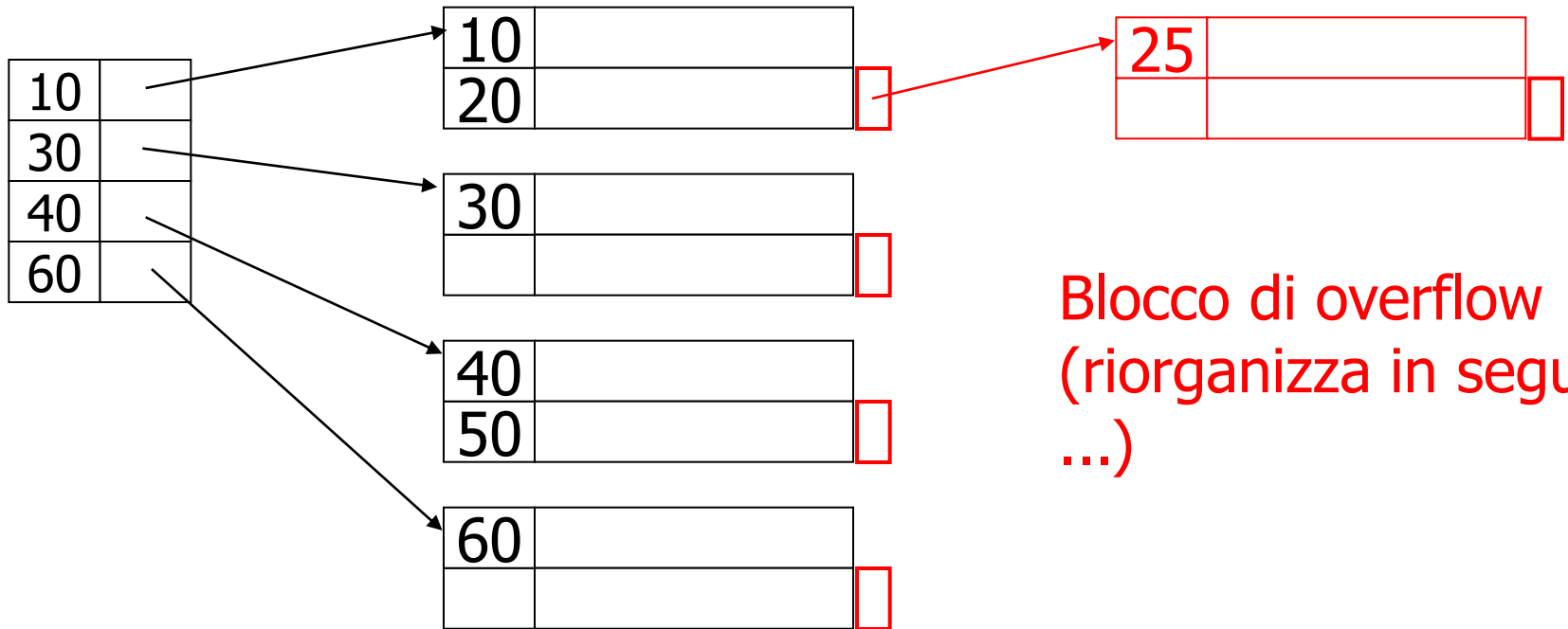
– inserisci il record 15



- Riorganizzazione immediata

# Inserimento, indice sparso

– inserisci il record 25



Blocco di overflow  
(riorganizza in seguito  
...)

# Inserimento, indice denso

- simile

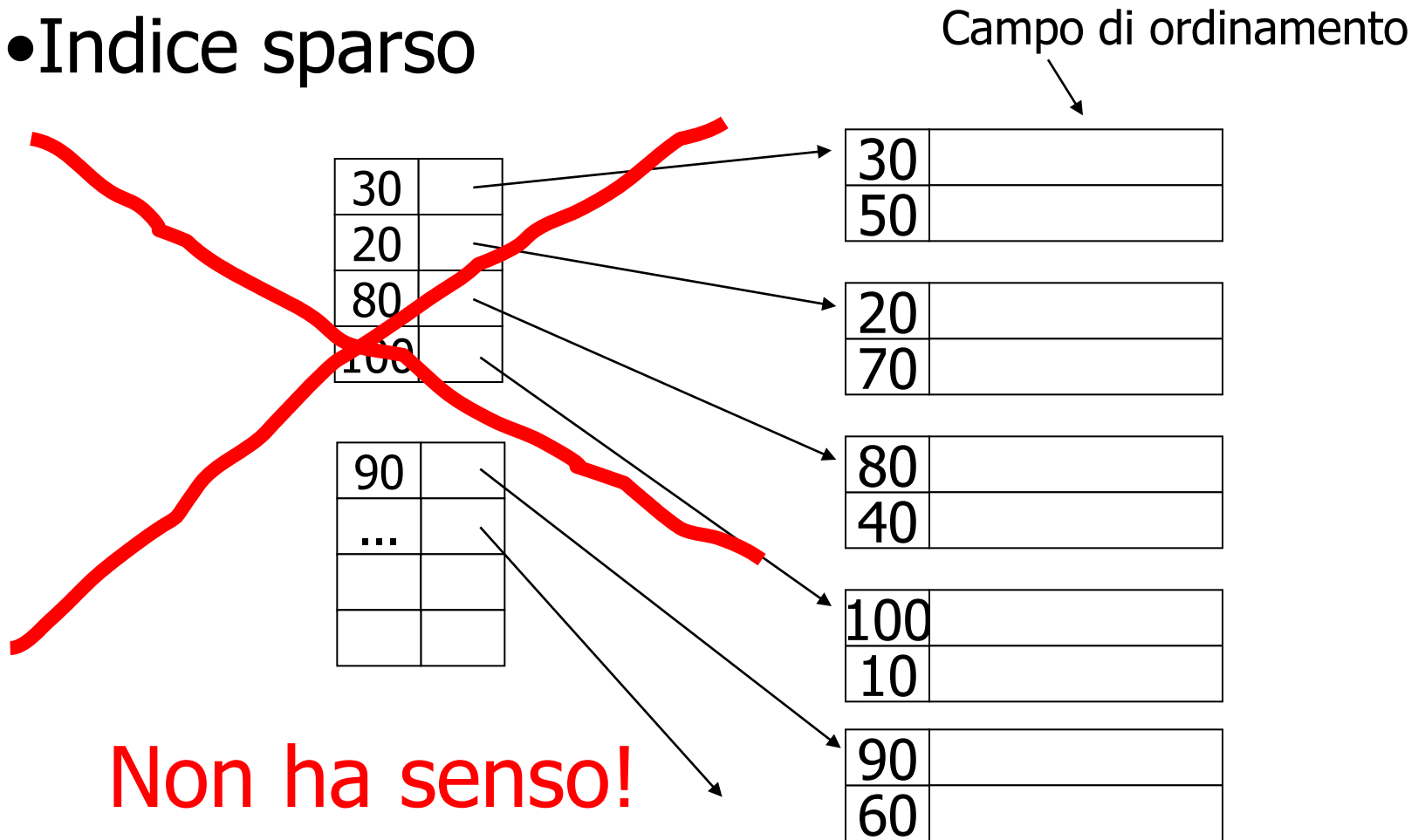


# Indici primari ISAM, osservazioni

- Adatto per ricerche su una chiave o un intervallo di chiavi
- Scansione sequenziale ordinata efficiente
- Gli inserimenti e le eliminazioni sono inefficienti

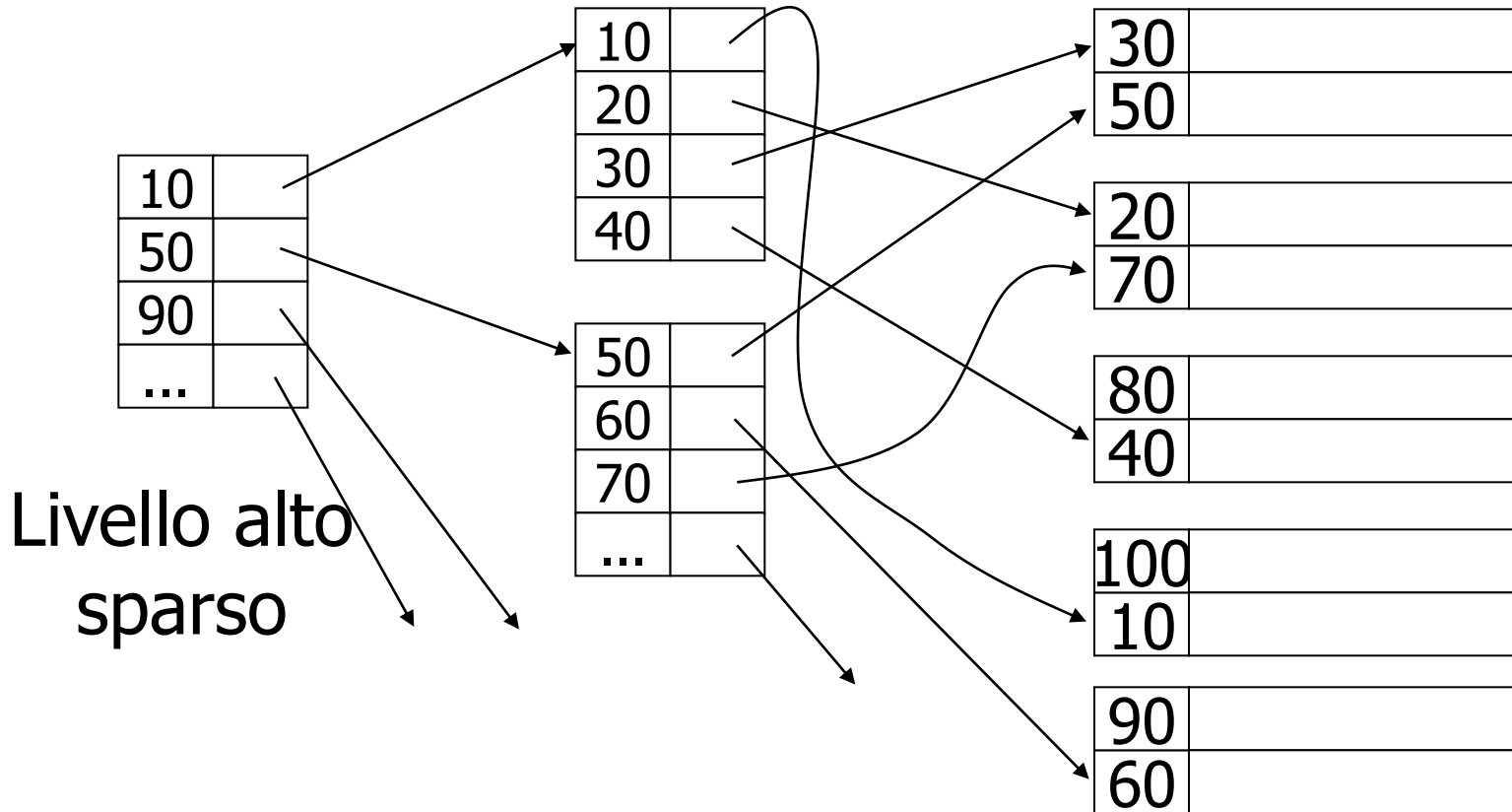
# Indici secondari ISAM

- Indice sparso



# Indici secondari

- Indice denso



# Indici secondari

- Il livello pu' basso e' denso
- Gli altri sono sparsi

**Inoltre: i puntatori sono puntatori a record**

(non puntatori a blocchi)

# Valori duplicati e indici secondari

20	
10	

20	
40	

10	
40	

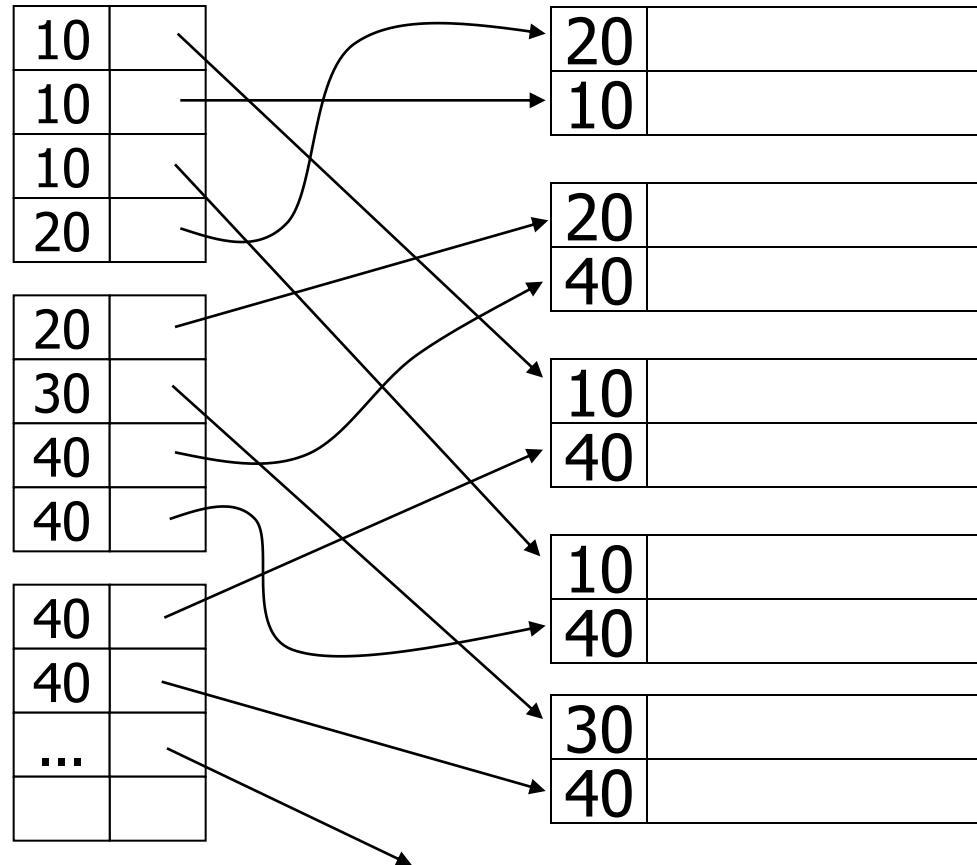
10	
40	

30	
40	

# Valori duplicati e indici secondari

Una possibilita'

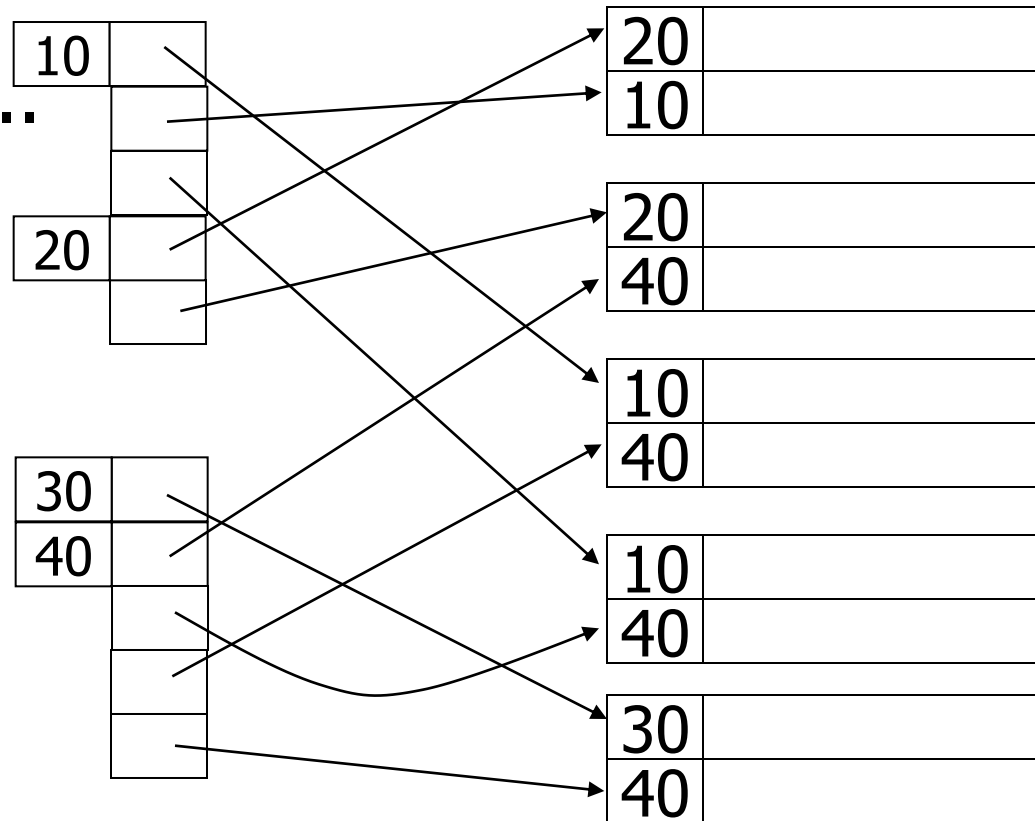
Problema:  
Spreco di  
Spazio su disco  
Tempo di ricerca



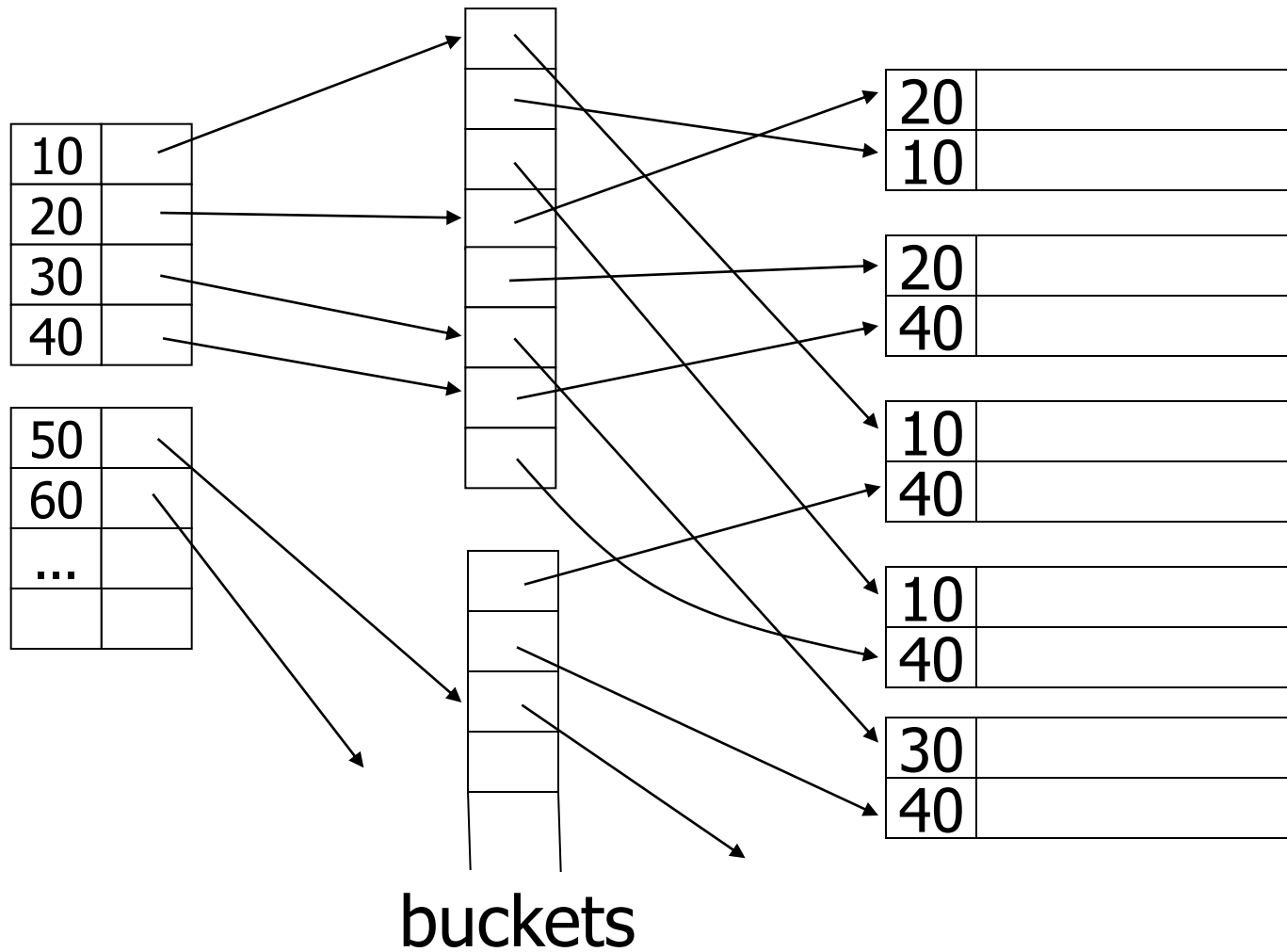
# Valori duplicati e indici secondari

Un'altra possibilita'...

Problema:  
I record dell'  
indice hanno  
dimensione  
variabile



# Valori duplicati e indici secondari





# Perche' l'idea dei bucket e' utile

Indici

Records

Name: primario

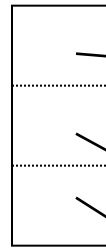
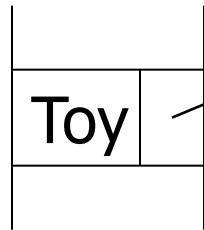
EMP (name,dept,floor,...)

Dept: secondario

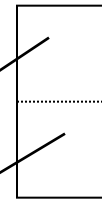
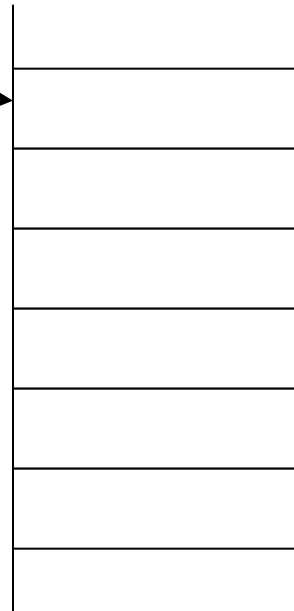
Floor: secondario

# Query: ottieni gli impiegati in (Toy Dept) $\wedge$ (2nd floor)

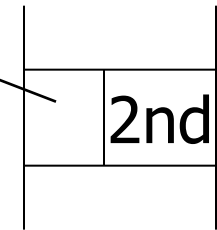
Indice su Dept.



EMP

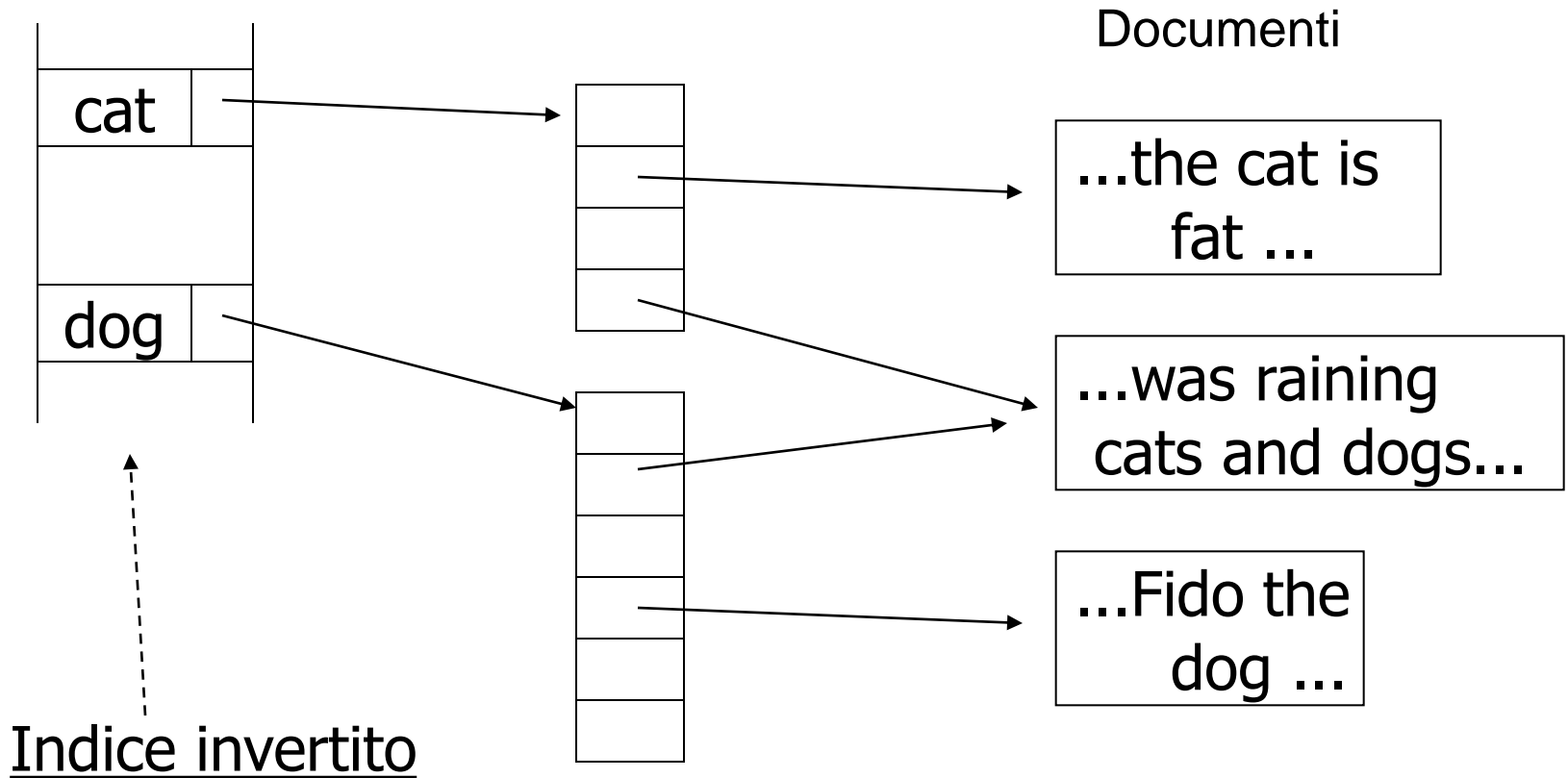


index su floor



→Interseca i bucket per toy e per 2nd Floorper ottenere l'insieme degli impiegati richiesti

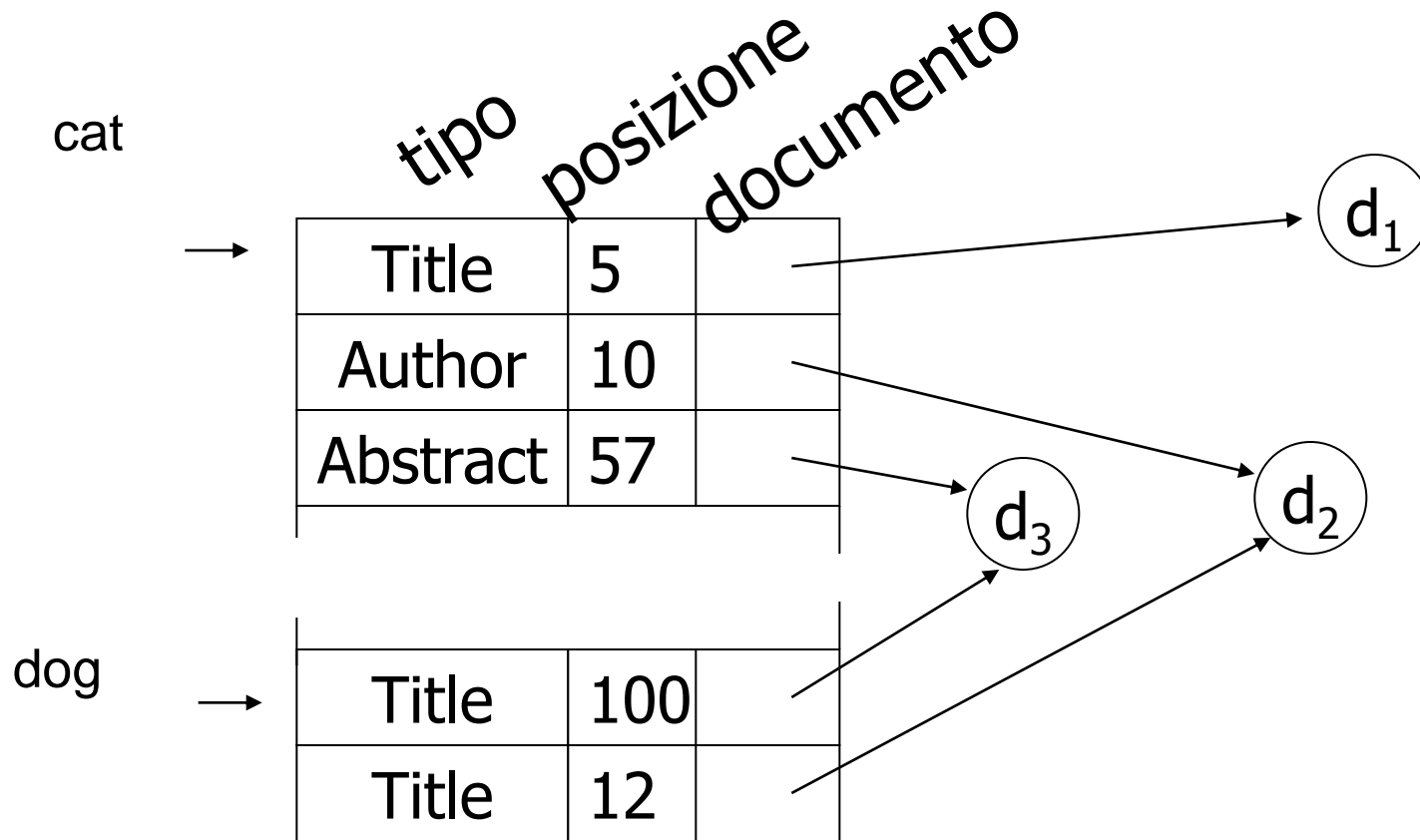
# Questa idea e' utile nell'information retrieval



# Queries di Information retrieval

- Trova gli articoli con “cat” e “dog”
  - Trova gli articoli con “cat” o “dog”
  - Trova gli articoli con “cat” e non “dog”
- 
- Trova gli articoli con “cat” nel titolo
  - Trova gli articoli con “cat” e “dog” entro 5 parole

# Tecnica comune: piu' informazione nei bucket



# Indici secondari, osservazioni

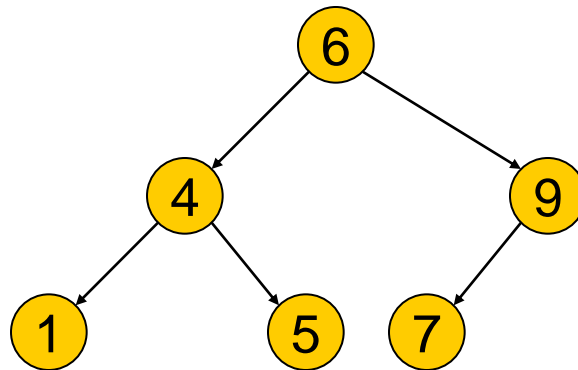
- Adatto per ricerche su una chiave o un intervallo di chiavi
- E' possibile la scansione del file nell'ordine della chiave di ricerca ma il numero di accessi e' pari al numero di record del file (un po' meno grazie alla bufferizzazione)
  - A volte costa di meno leggere il file nell'ordine fisico e poi riordinarlo con il merge sort
- Gli inserimenti e le eliminazioni sono inefficienti

# Indici ISAM, problemi

- Gli indici ISAM visti finora sono basati su strutture ordinate e quindi sono poco flessibili in presenza di elevata dinamicità
  - Gli inserimenti sono costosi
  - Gli alberi possono diventare sbilanciati
  - Lo spazio di overflow puo' diventare significativo
- Gli indici utilizzati dai DBMS sono più sofisticati:
  - indici dinamici multilivello: B+-tree (intuitivamente: alberi di ricerca bilanciati)
    - Arriviamo ai B+-tree per gradi
      - Alberi binari di ricerca
      - Alberi n-ari di ricerca
      - Alberi n-ari di ricerca bilanciati

# Albero binario di ricerca

- Albero binario etichettato in cui per ogni nodo il sottoalbero sinistro contiene solo etichette minori di quella del nodo e il sottoalbero destro etichette maggiori
- tempo di ricerca (e inserimento), pari alla profondità:
  - logaritmico nel caso “medio” (assumendo un ordine di inserimento casuale)

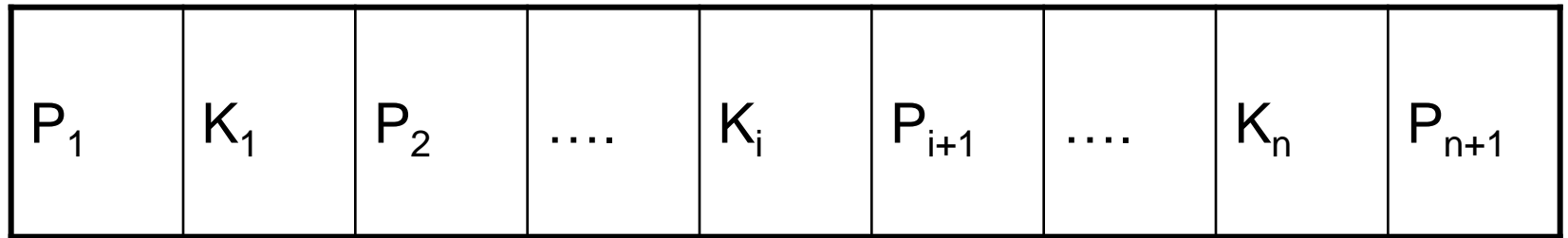




# Albero di ricerca di ordine $n$

- Ogni nodo ha (fino a)  $n+1$  figli e (fino a)  $n$  etichette, ordinate
- $n$  e' detto ordine dell'albero
- Nell' $(i+1)$ -esimo sottoalbero abbiamo tutte etichette maggiori o uguali della  $i$ -esima etichetta e minori della  $(i+1)$ -esima
- Ogni ricerca o modifica comporta la visita di un cammino radice foglia
- In strutture fisiche, un nodo corrisponde a un blocco
- La struttura è ancora (potenzialmente) rigida
- Un B+-tree è un albero di ricerca che viene mantenuto bilanciato, grazie a:
  - Riempimento parziale (mediamente 70%)
  - Riorganizzazioni (locali) in caso di sbilanciamento

# Organizzazione dei nodi del B+-tree: nodo non foglia



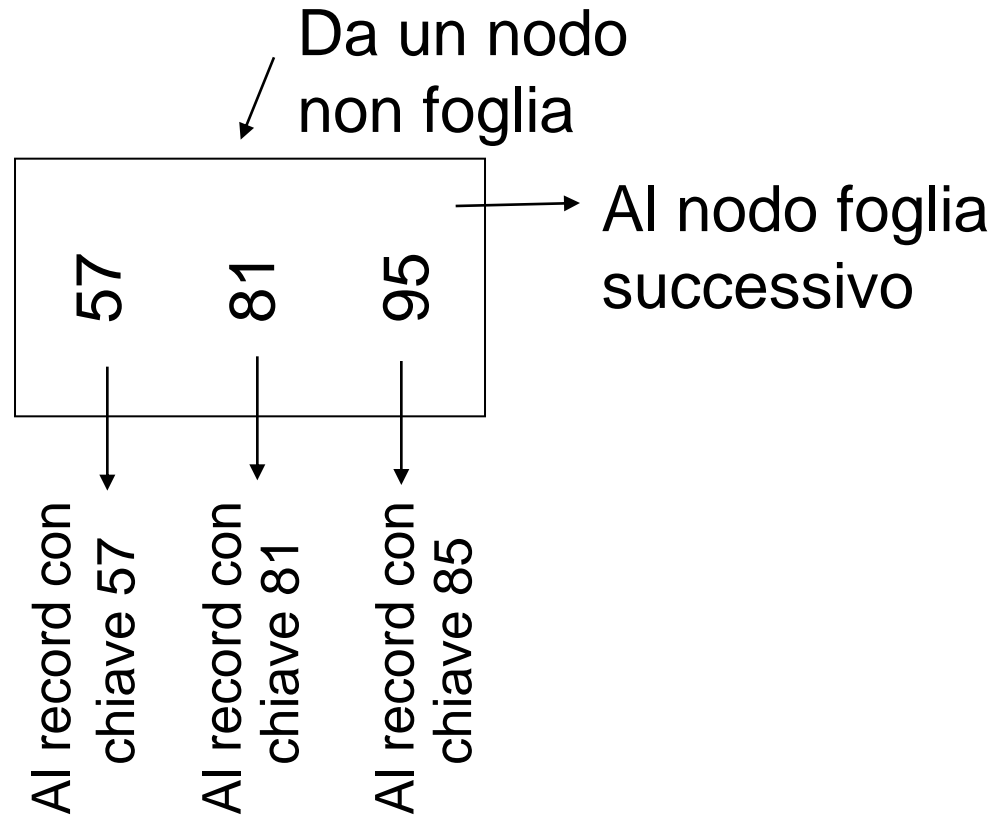
sotto-albero  
che contiene  
le chiavi  
 $K < K_1$

sotto-albero  
che contiene  
le chiavi  
 $K_i \leq K < K_{i+1}$

sotto-albero  
che contiene  
le chiavi  
 $K \geq K_n$

# Nodi foglia

- Esempio



# Bilanciamento

- Si richiede di usare almeno

Non-foglia:  $NF = \lceil (n+1)/2 \rceil$  puntatori

Foglia:  $F = \lfloor (n+1)/2 \rfloor$  puntatori ai dati

- Se  $n$  e' dispari  $NF = F = (n+1)/2$
- Se  $n$  e' pari  $NF = F + 1 = n/2 + 1$

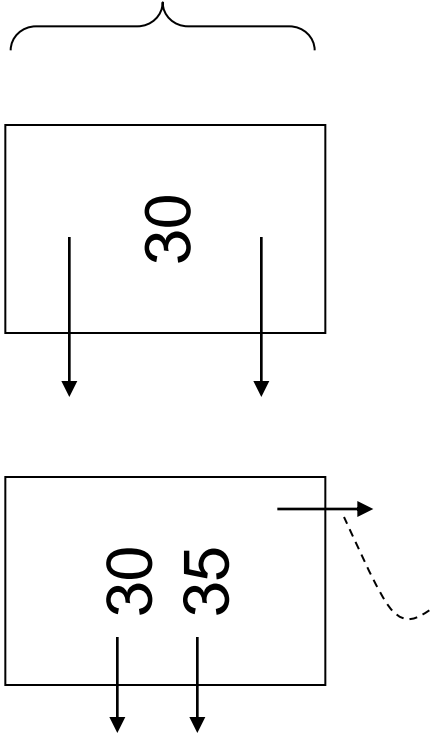
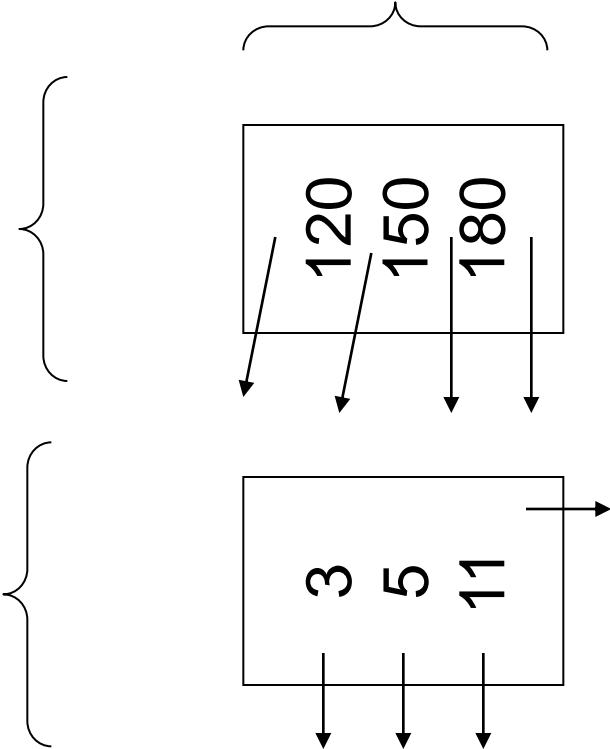
n=3

Non-foglia

Foglia

nodo pieno

nodo minimo



Conta anche se nullo

# Chiavi e puntatori

	Max Pun. →dati	Max ch.	Min pun→dati	Min ch.
Non-foglia (non-radice)	$n+1$	$n$	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Foglia (non-radice)	$n$	$n$	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Radice	$n+1$	$n$	2	1

## Chiavi e puntatori, n=3

	Max Pun. →dati	Max ch.	Min pun→dati	Min ch.
Non-foglia (non-radice)	4	3	2	1
Foglia (non-radice)	3	3	2	2
Radice	4	3	2	1

## Chiavi e puntatori, n=4

	Max Pun. →dati	Max ch.	Min pun→dati	Min ch.
Non-foglia (non-radice)	5	4	3	2
Foglia (non-radice)	4	4	2	2
Radice	5	4	2	1



# Aggiornamenti

- Inserimenti ed eliminazioni sono precedute da una ricerca fino ad una foglia
- Per gli inserimenti, se c'è posto nella foglia, ok, altrimenti il nodo va suddiviso, con necessità di un puntatore in più per il nodo genitore; se non c'è posto, si sale ancora, eventualmente fino alla radice.
- Dualmente, le eliminazioni possono portare a riduzioni di nodi. Il riempimento deve rimanere sempre superiore al 50%
- Modifiche del campo chiave vanno trattate come eliminazioni seguite da inserimenti

# Algoritmo di inserimento

- Si cerca la foglia in cui va la nuova chiave e si mette la chiave nella foglia se c'è posto
- Altrimenti, si divide la foglia in due e si dividono le chiavi tra i due nuovi nodi, in modo che ciascuno sia pieno almeno a metà
- La divisione viene vista al livello superiore come la creazione di una nuova coppia chiave-puntatore da inserire a quel livello. Si applica perciò una strategia ricorsiva: se c'è posto si inserisce, altrimenti si divide e si continua al livello superiore
- L'unica eccezione è la radice: se si cerca di inserire una coppia chiave-puntatore nella radice e non c'è posto, si divide la radice in due e si crea un nuovo nodo al livello superiore che ha i due nodi come figli (nuova radice).

# Algoritmo di inserimento – Gestione delle chiavi nelle foglie

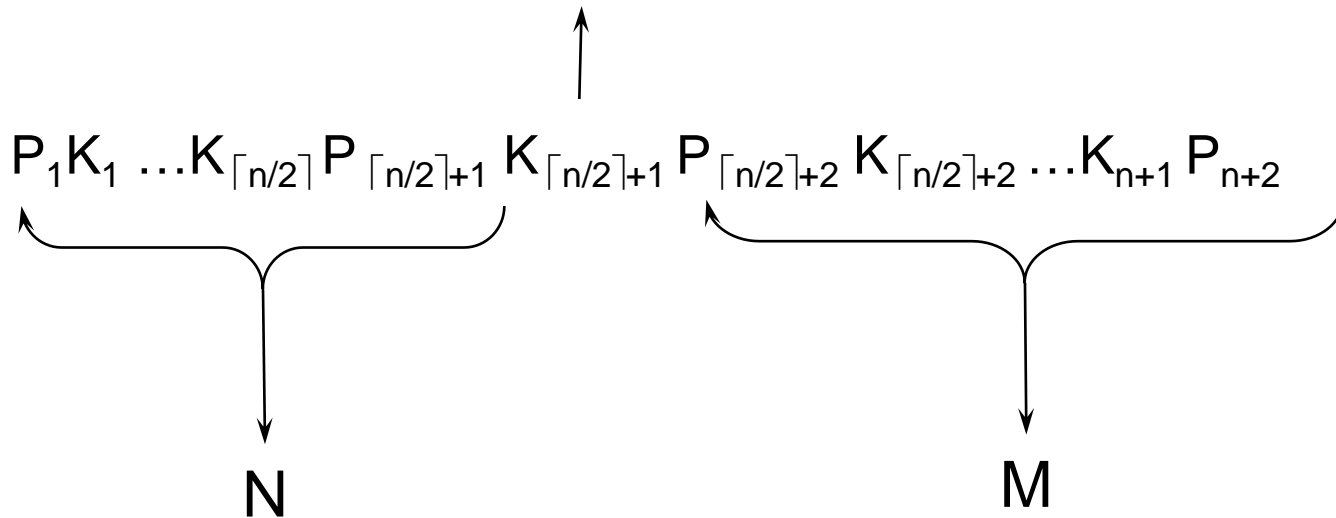
- Sia  $N$  una foglia con  $n$  coppie chiave-puntatore
- Si cerca di inserire la  $n+1$  esima coppia chiave-puntatore
- Si crea un nodo  $M$  fratello di  $N$  direttamente alla sua destra
- Le prime  $\lceil (n+1)/2 \rceil$  coppie, in ordine di chiave, rimangono in  $N$ , le rimanenti ( $\lfloor (n+1)/2 \rfloor$ ) vanno in  $M$
- Sia  $N$  che  $M$  hanno il numero minimo ( $\lfloor (n+1)/2 \rfloor$ ) di coppie
- Si passa al genitore di  $N$  la coppia chiave minima di  $M$ -puntatore ad  $M$

# Algoritmo di inserimento – Gestione delle chiavi nei nodi non foglia

- Sia N un nodo non-foglia con n chiavi e n+1 puntatori a cui è stata assegnata la (n+1)-esima chiave e l'(n+2)-esimo puntatore. Si procede in questo modo:
  - Si inserisce la nuova coppia chiave-puntatore nella giusta posizione nell'elenco di quelle esistenti
  - Si crea un nodo M, fratello di N, alla sua destra
  - Si lasciano in N i primi  $\lceil n/2 \rceil + 1$  puntatori, in ordine e si mettono in M i rimanenti  $\lfloor n/2 \rfloor + 1$
  - Le prime  $\lceil n/2 \rceil$  chiavi rimangono in N, le ultime  $\lfloor n/2 \rfloor$  vanno in M. Rimane sempre una chiave nel mezzo  $K_{\lceil n/2 \rceil + 1}$ : indica la chiave più piccola raggiungibile con il primo puntatore di M. Rappresenta quindi la chiave più piccola raggiungibile da M
  - $K_{\lceil n/2 \rceil + 1}$  sarà usata nel genitore di N ed M per dividere le ricerche tra questi due nodi: si passa al nodo genitore di N la coppia  $K_{\lceil n/2 \rceil + 1}$ -puntatore ad M

# Algoritmo di inserimento – Gestione delle chiavi nei nodi non foglia

Al genitore insieme  
a un puntatore a M



# Algoritmo di inserimento

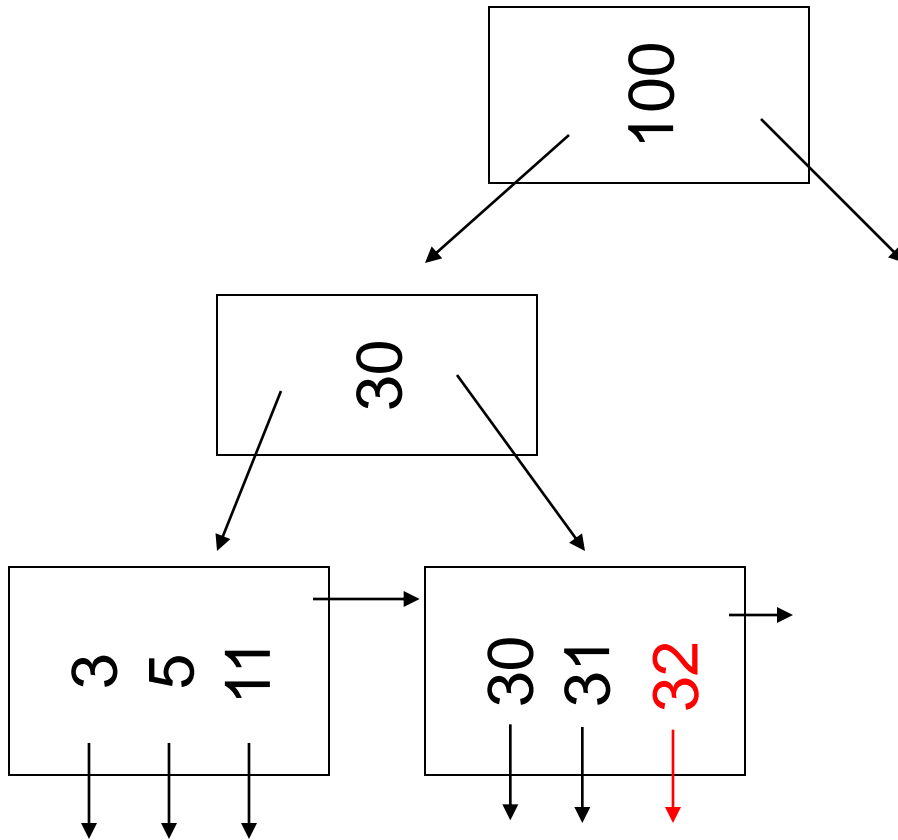
- In alternativa alla suddivisione di un nodo, se uno dei fratelli del nodo in cui si deve inserire la coppia chiave-puntatore ha posto, si possono ridistribuire le chiavi tra il nodo e il fratello

## Esempi di inserimento in un B+-tree

- (a) Caso semplice
  - Spazio disponibile nella foglia
- (b) Overflow in una foglia
- (c) Overflow in una non foglia
- (d) Nuova radice

(a) Inserisci chiave= 32

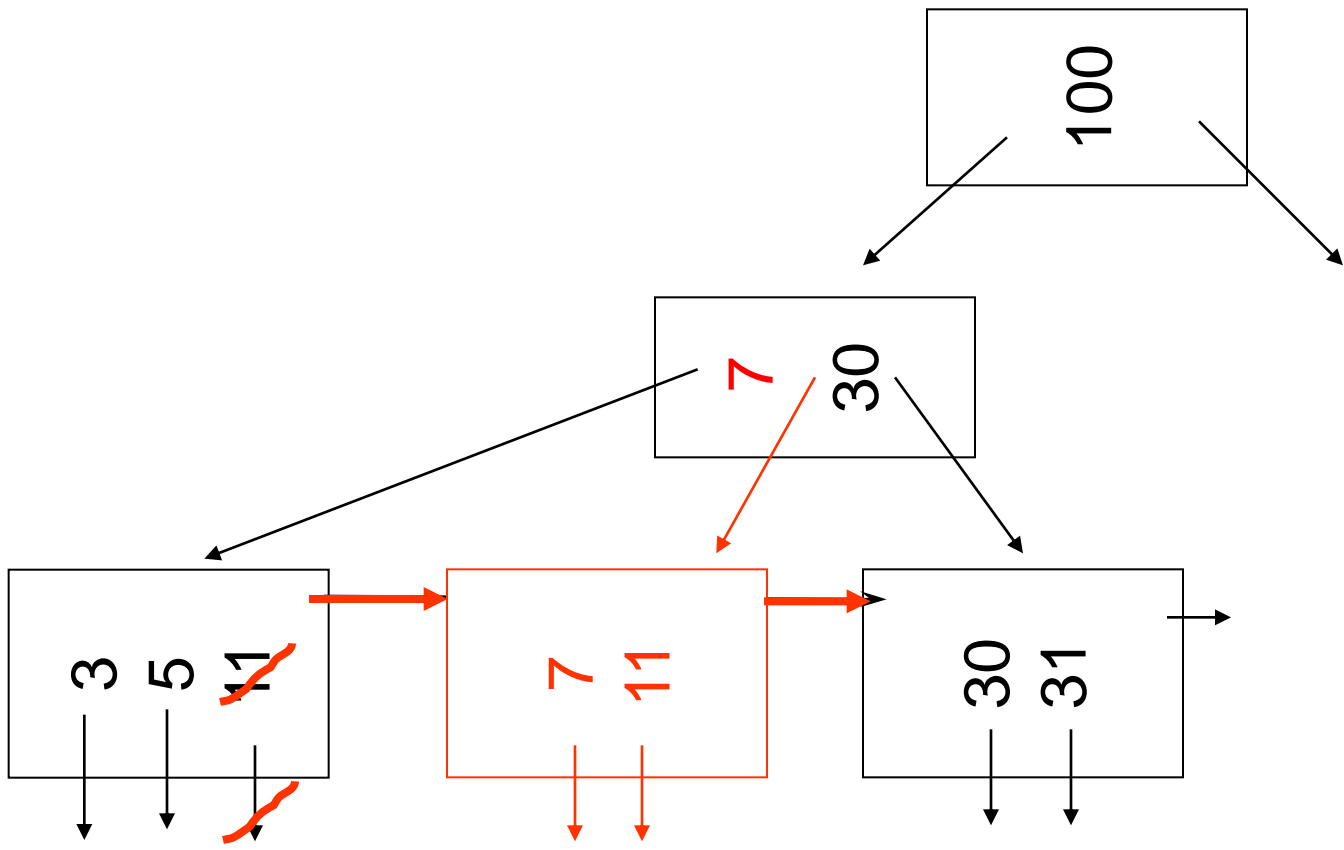
n=3





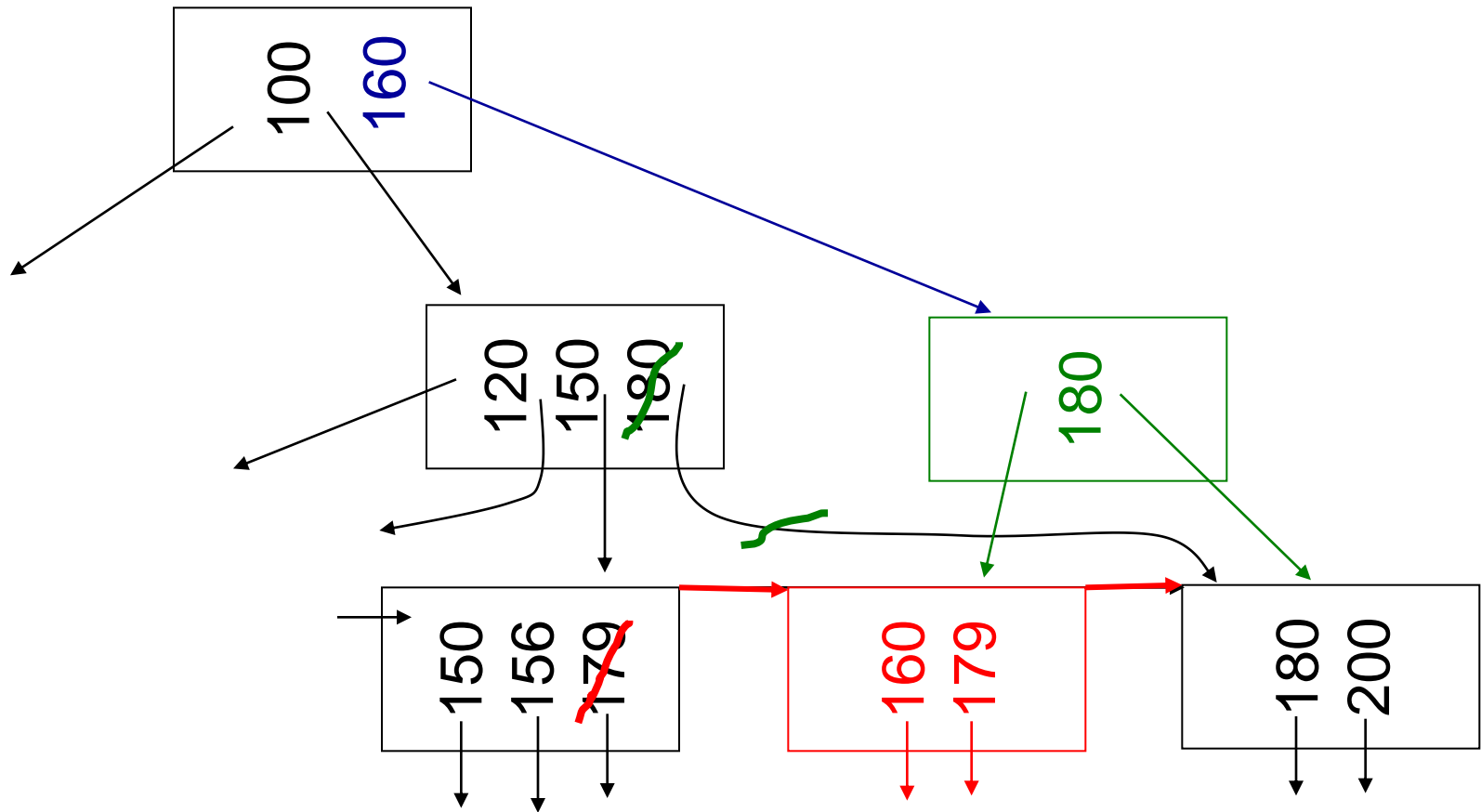
(b) Inserisci chiave = 7

n=3



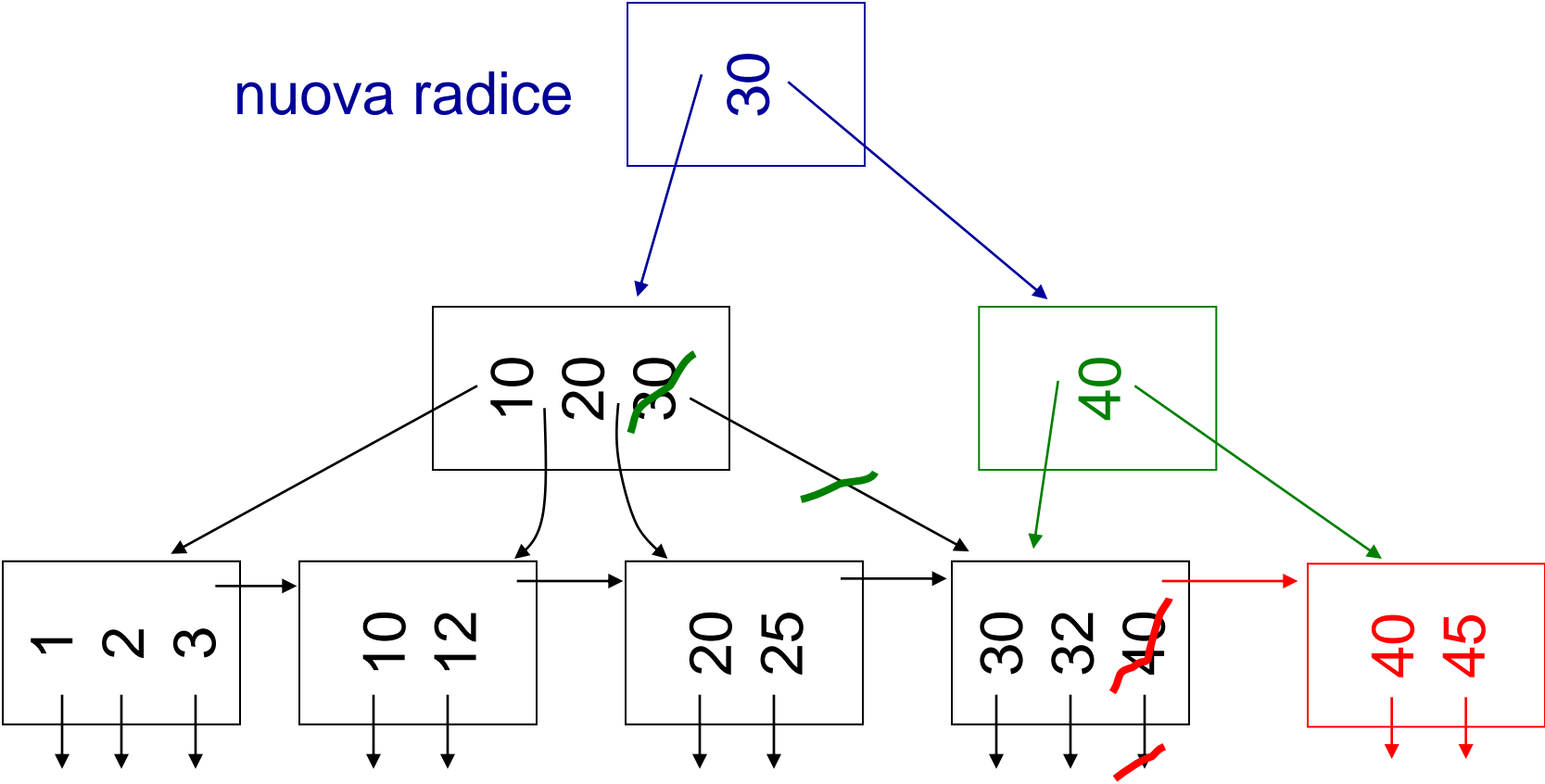
(c) Inserisci chiave = 160

n=3



(d) Inserisci 45

n=3



# Algoritmo di cancellazione in una foglia

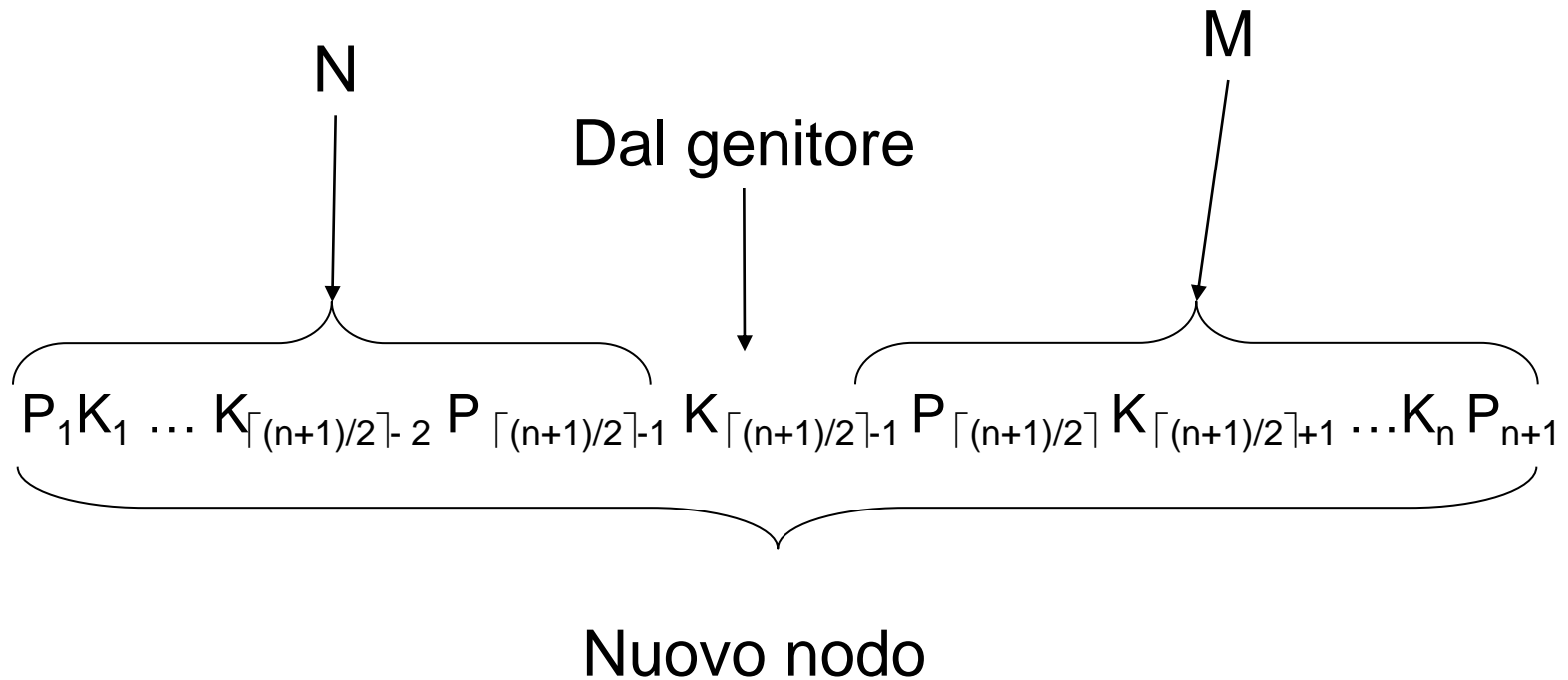
- Si cerca la foglia che contiene la chiave. Se la foglia ha più del numero minimo di chiavi, si rimuove la coppia chiave-puntatore.
- Se la foglia N ha il numero minimo di chiavi
  - Se uno dei fratelli adiacenti M ha più del numero minimo di chiavi, una coppia chiave-puntatore può essere spostata da M ad N.
    - Le chiavi al genitore possono dover essere modificate per riflettere il cambiamento. Ad esempio, se M è a destra, si muove la chiave più piccola in N e bisogna sostituire la chiave nel genitore a sinistra del puntatore ad M con la nuova chiave più piccola di M
  - Altrimenti, N ha meno del numero minimo di chiavi ed esiste un suo fratello M che ne ha il numero minimo. N ed M possono quindi essere fusi. Lo si fa, cancellando uno dei due. Si cancella la relativa coppia chiave-puntatore nel genitore ed, eventualmente, si procede ricorsivamente.

# Algoritmo di cancellazione in una non-foglia

- Se il nodo ha più del numero minimo di chiavi, si rimuove la coppia chiave-puntatore.
- Se il nodo N ha il numero minimo di chiavi
  - Se uno dei fratelli adiacenti M ha più del numero minimo di chiavi, una coppia chiave-puntatore può essere spostata da M ad N.
    - Le chiavi al genitore possono dover essere modificate per riflettere il cambiamento.
  - Altrimenti, N ha meno del numero minimo di chiavi ed esiste un suo fratello M che ne ha il numero minimo. N ed M possono quindi essere fusi. Lo si fa, cancellando uno dei due. Si separa l'ultimo puntatore del nodo di sinistra dal primo puntatore del nodo di destra con la chiave K a sinistra del puntatore al nodo di destra nel padre. Si cancella la coppia K-puntatore al nodo eliminato nel padre.

# Algoritmo di cancellazione in una non-foglia

Nel caso in cui N sia a sinistra



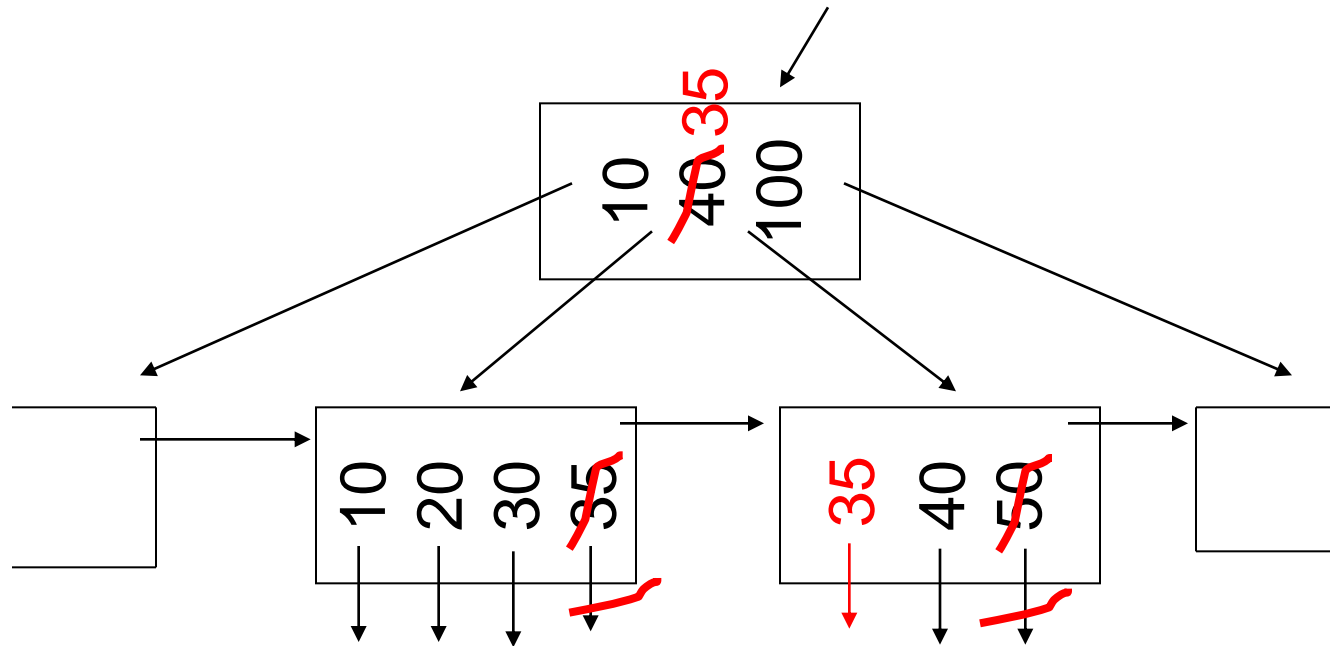
## Esempi di cancellazione da un B+-tree

- (a) Caso semplice – nessun esempio
- (b) Ridistribuisce le chiavi (da sinistra)
- (c) Ridistribuisce le chiavi (da destra)
- (d) Unione con il vicino (fratello)
- (e) Caso (d) a una non foglia

(b) Ridistribuisci le chiavi: prendi una chiave dal fratello di sinistra

- Cancella 50

n=4

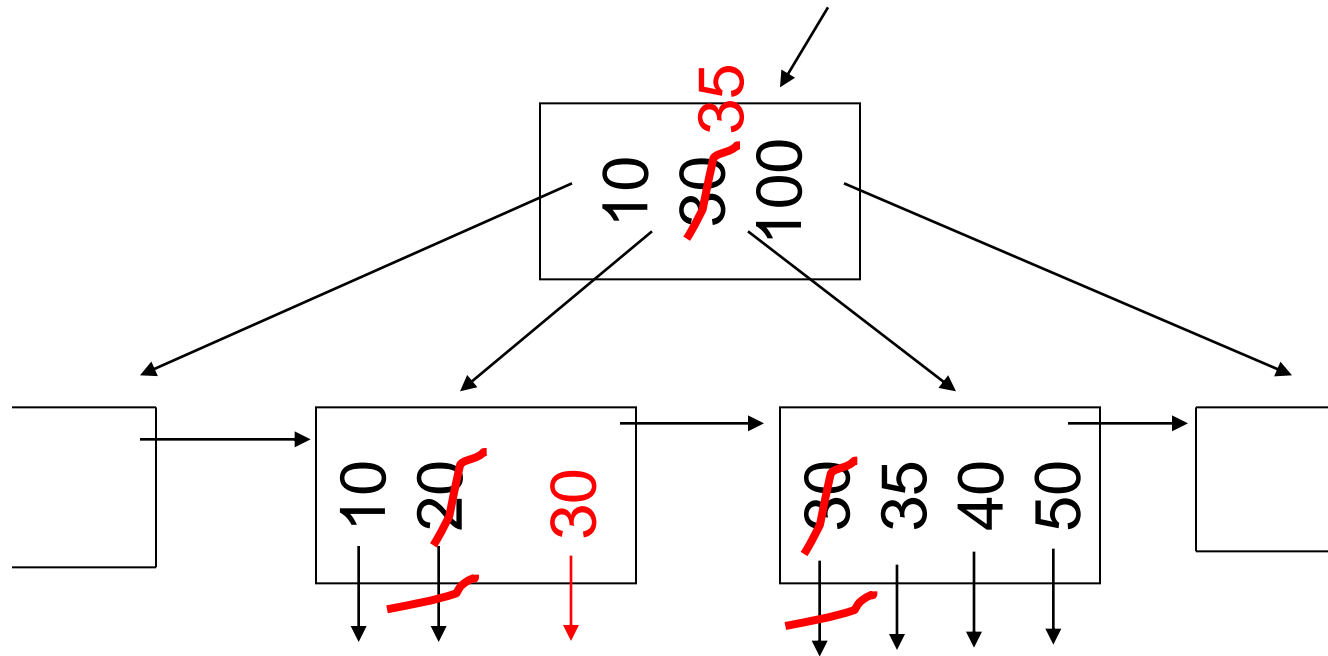




(c) Ridistribuisci le chiavi: prendi una chiave dal fratello di destra

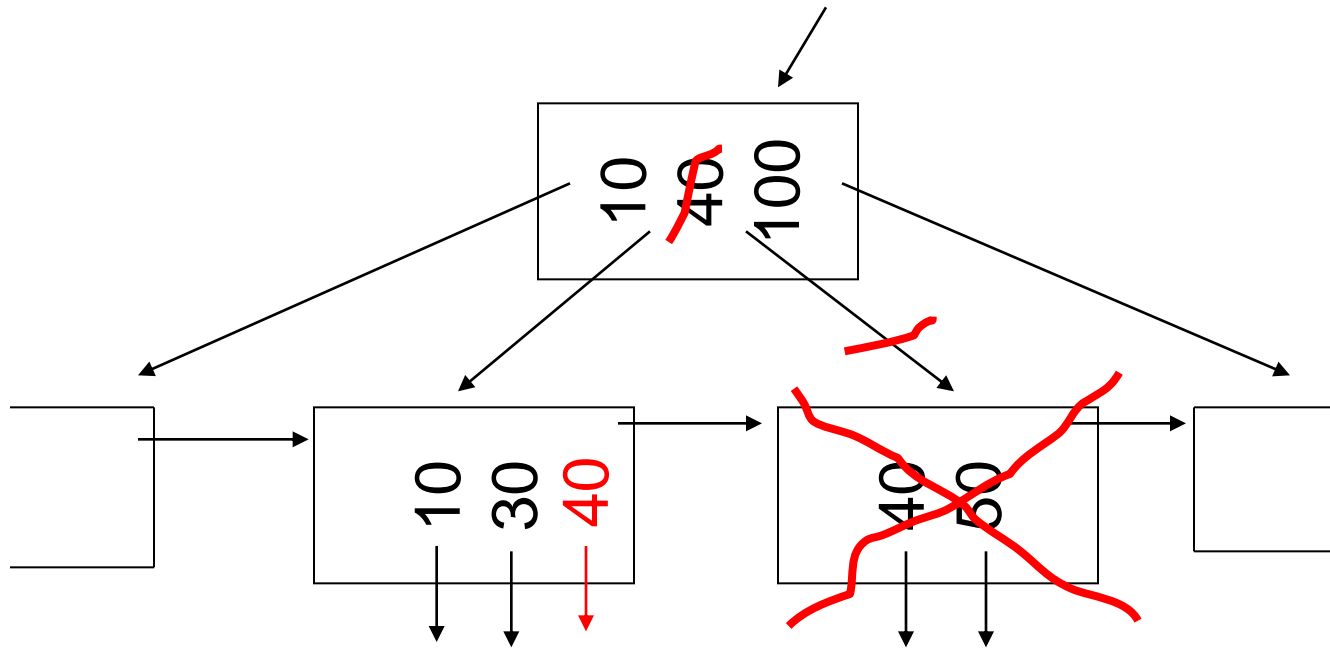
- Cancella 20

n=4



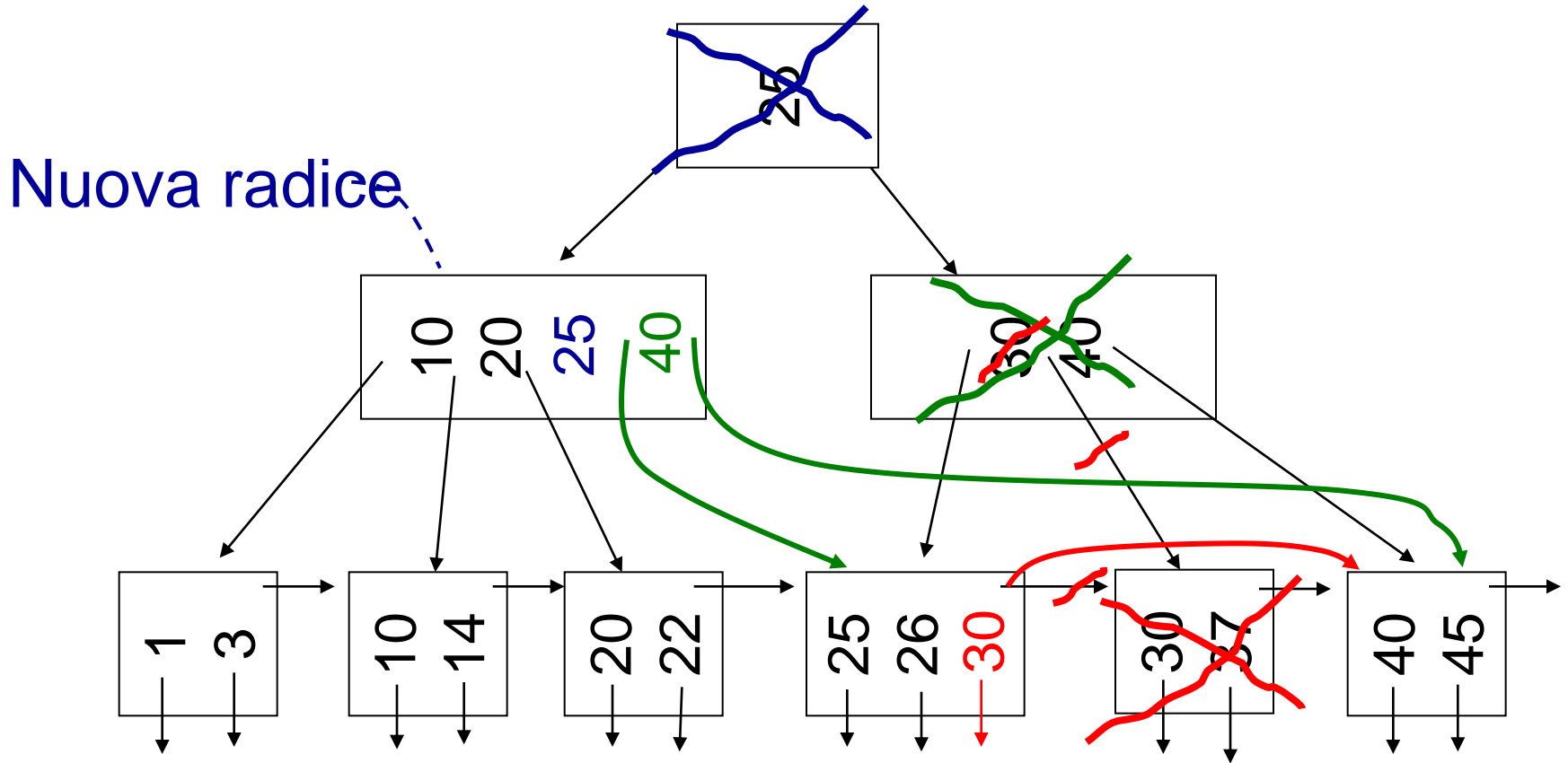
(d) Unisci al fratello  
– Cancella 50

n=4



(e) Unione di non foglie  
– Cancella 37

n=4



# Cancellazione in pratica

- Spesso l'unione non e' implementata
  - Troppo costosa e non necessaria

# Tempi tipici di accesso di un B+-tree

- Ipotesi: blocchi 4096 bytes, chiave intero di 4 bytes, puntatori 8 bytes,  $n$  e' il piu' grande intero tale che
- $4n+8(n+1)\leq 4096 \Rightarrow 12n\leq 4088 \Rightarrow n\leq 340,6 \Rightarrow n=340$
- Tempo di accesso ad una chiave: numero di operazioni di I/O=numero di livelli dell'albero + 1 per la ricerca o 2 per l'inserimento o la cancellazione
- Quanti livelli ha un B+-tree (supponendo che  $n=340$ )?
- Supponiamo che i blocchi siano occupati a mezza via tra il minimo (171 puntatori) e il massimo (341 puntatori): 256 puntatori per nodo
- Con una radice, 256 figli e  $256^2=65536$  nodi foglia, abbiamo  $256^3=16,8$  milioni di record
- Quindi file con record fino a 16,8 milioni di record possono essere indirizzati con un B+-tree a tre livelli

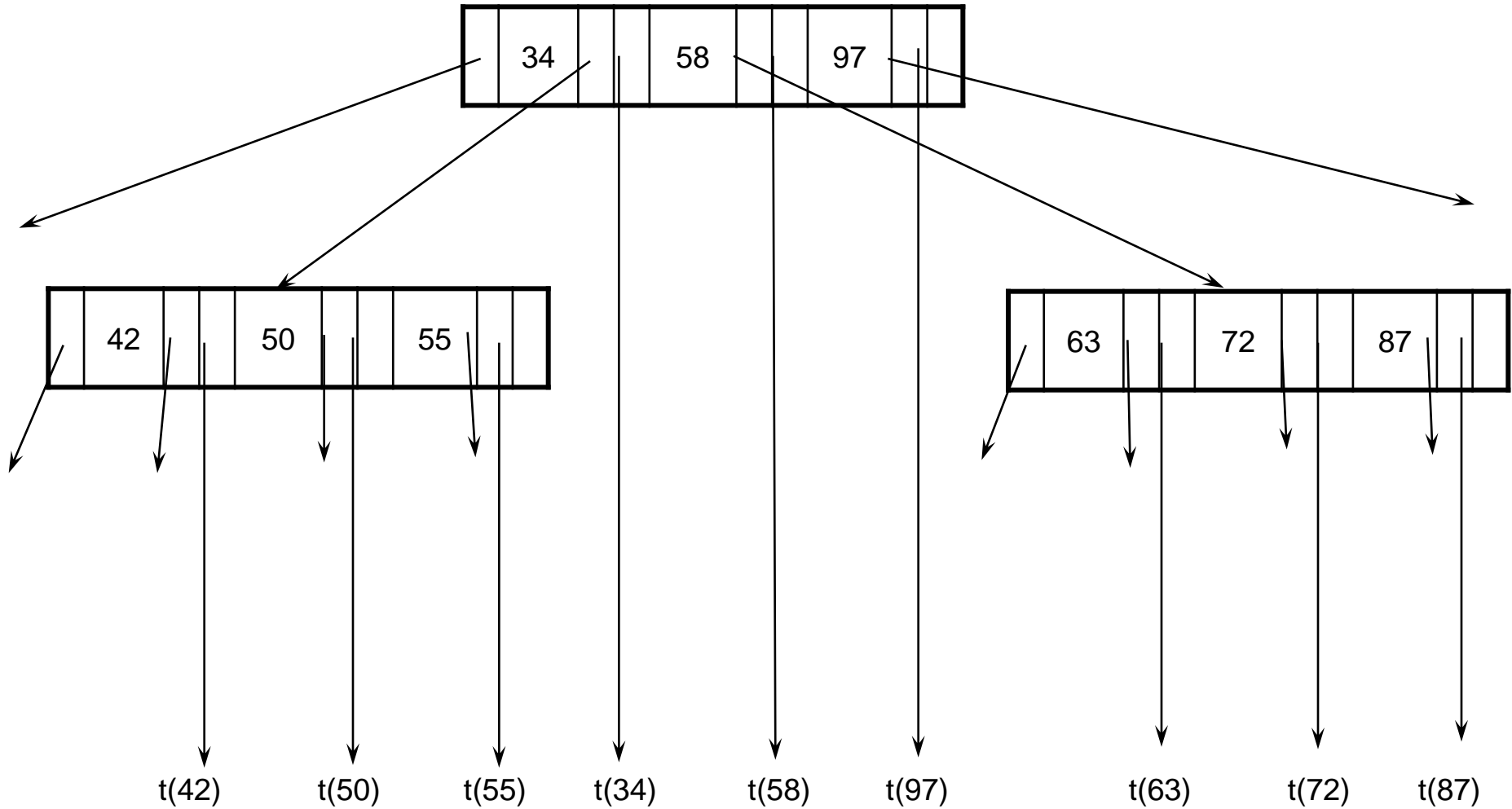
# Tempi tipici di accesso di un B+-tree

- Si possono avere meno di 4 operazioni di I/O per accedere ad una chiave:
  - Se la radice e' in memoria, 3 operazioni
  - Se anche il primo livello e' in memoria, 2 operazioni

# B-tree e B+-tree

- B+-tree:
  - le foglie sono collegate in una lista
  - ottimi per le ricerche su intervalli
  - molto usati nei DBMS
- B-tree:
  - i nodi intermedi hanno puntatori direttamente ai dati
  - le foglie non sono collegate in una lista

# Un B-tree





## B-tree verso B+-tree

- + I B-tree hanno un tempo di ricerca inferiore
- + I B-tree hanno meno nodi nell'indice (perche' le chiavi sono presenti al piu' una volta nell'indice)
- Nei B-tree le cancellazioni sono piu' complicate
- Nei B-tree rispondere a query di range o eseguire una scansione ordinata del file e' piu' costoso

Vengono preferiti i B+-tree