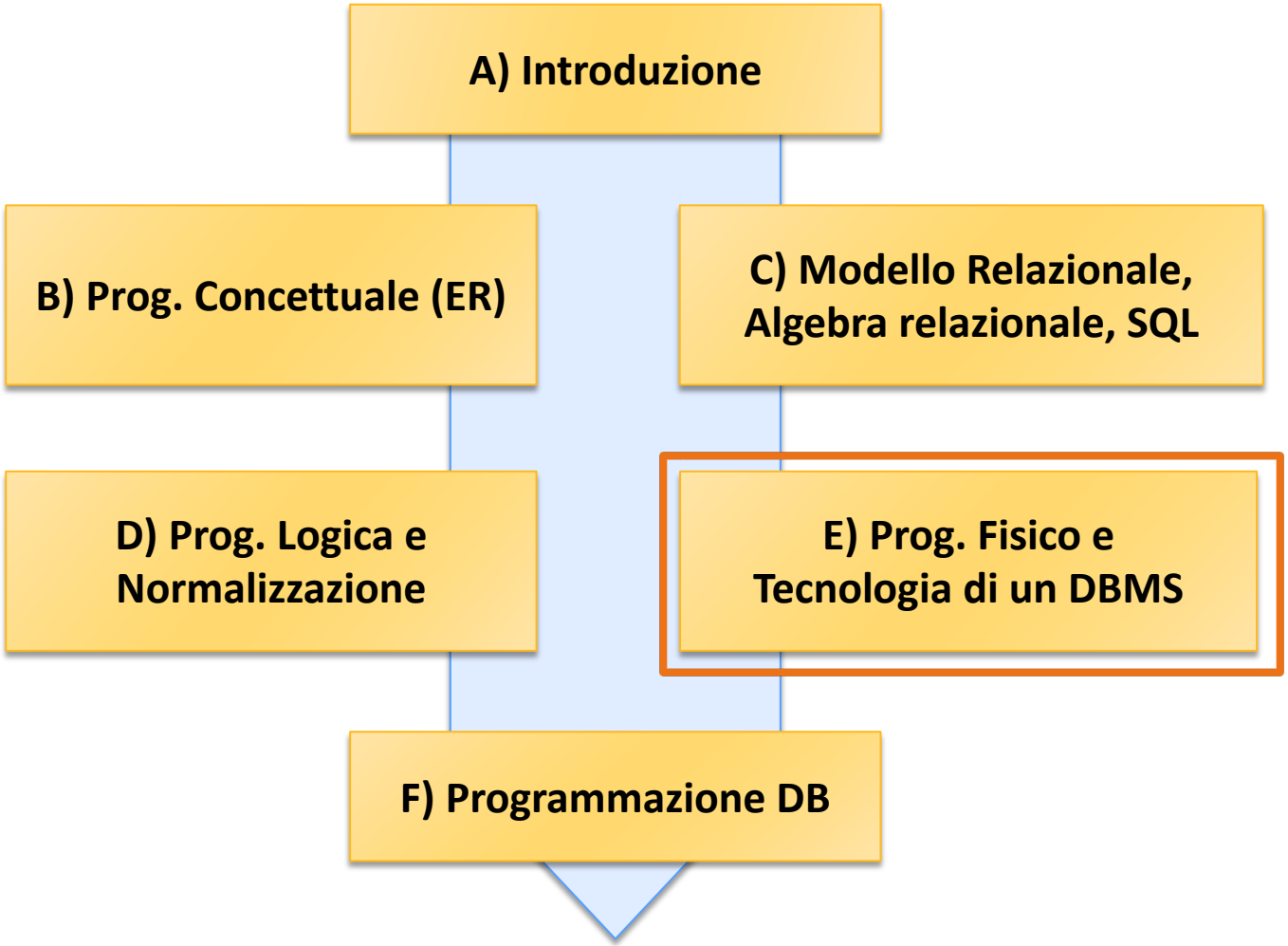
# Basi di Dati

## Metodi di Accesso (Hash e Indici)

# Basi di Dati – Dove ci troviamo?



# Metodi di accesso

## Organizzazioni hash

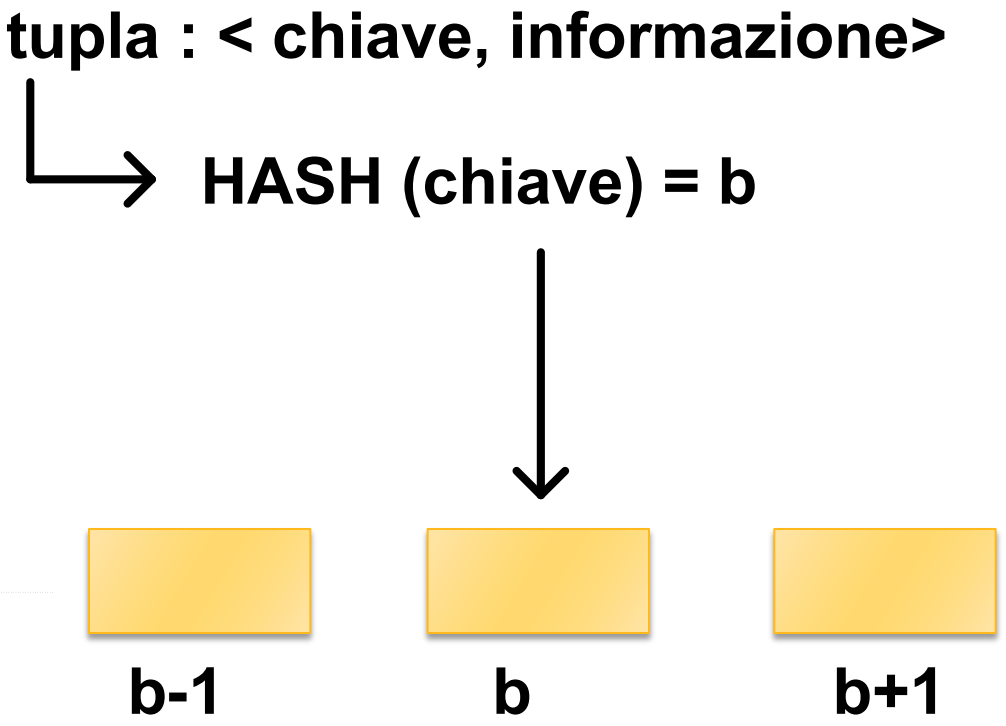
# Organizzazioni hash

- **IDEA DI BASE : associare ad un file di NB blocchi una funzione che trasformi un valore di chiave $k_i$ in un numero di blocco $b_i$ tra 0 ed NB-1**
  - **una tupla viene memorizzata nel blocco $b_i$**
  - **ad ogni blocco è associato un indirizzo nel disco**
  - **un blocco puo' contenere 1 o più tuple**

  - **Nel caso di organizzazione hash, il blocco (o pagina) prende anche il nome di bucket**

# Organizzazioni hash

**La regola con cui ad una chiave k viene associato un numero di blocco b si chiama funzione hash o algoritmo di hashing.**



# Organizzazioni hash

**Il file è ben utilizzato se:**

- **ci sono pochi spazi vuoti, cioè se il fattore di "packing" (caricamento) è elevato:**

  **Fp = m/M <1, dove:**

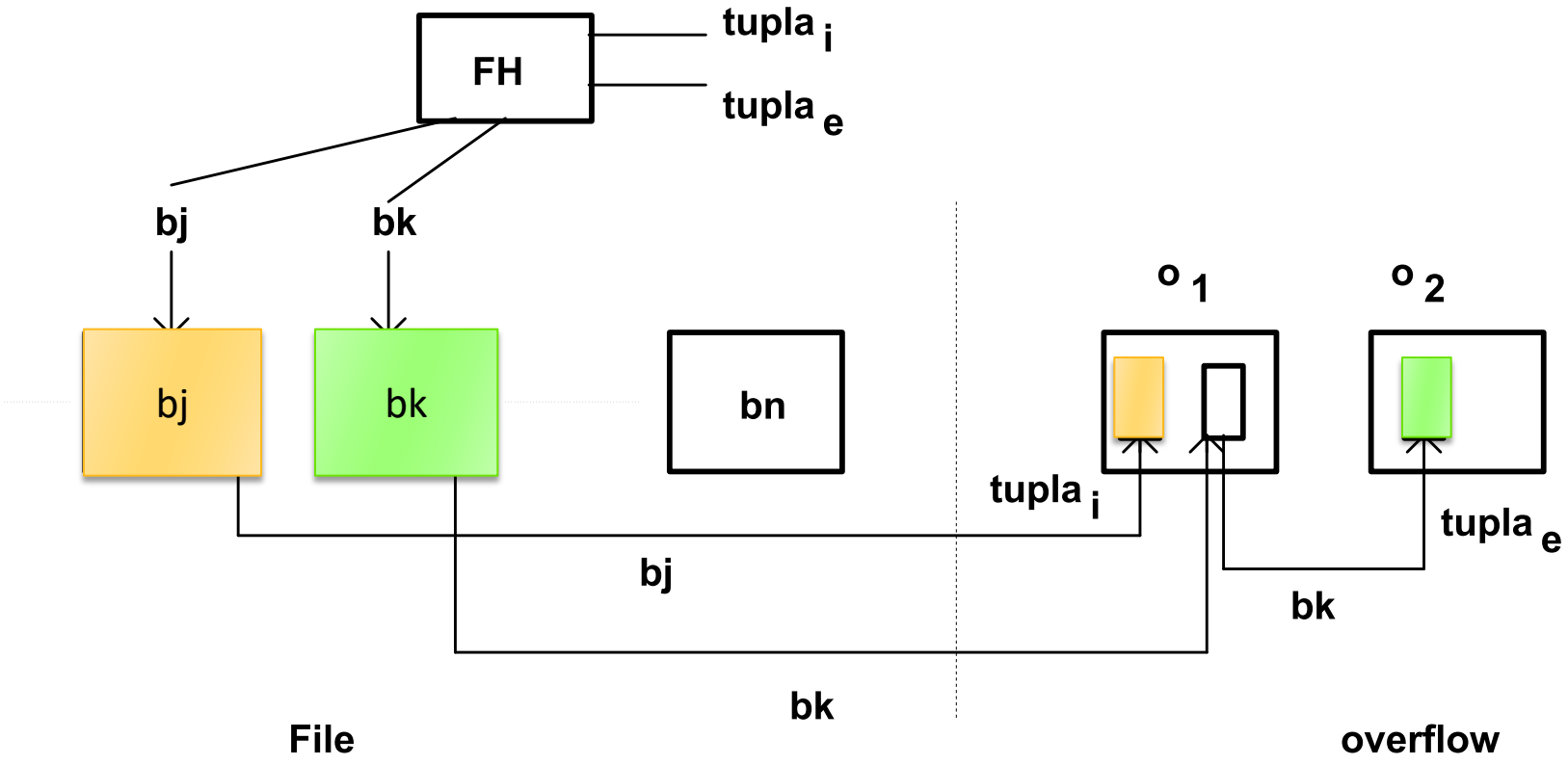
  **M = numero massimo di record per blocco,**

  **m = numero medio di record per blocco**
- **l'occupazione effettiva dei blocchi è vicina alla media**

**Fp basso sta a significare che il file è vuoto,**
**Fp vicino ad 1 da significare che molti blocchi sono pieni. Con i blocchi pieni si creano problemi di OVERFLOW**

# Organizzazioni hash

**Quando un blocco (es. $b_j$) è pieno ed una nuova tupla deve essere inserita poiché FH (f. hash) gli ha assegnato quel blocco ($b_j$) allora si deve creare una lista (catena) di overflow.**



# Organizzazioni hash

**L'OVERFLOW avviene per due motivi:**

- **ci sono più tuple con lo stesso valore di chiave per cui FH le manda nello stesso blocco**
- **FH non è perfetta ed assegna a due o più valori di chiave diversi lo stesso blocco del file (collisione)**

**con lunghe liste di overflow si degrada l'efficienza**
**FH deve distribuire le tuple il più uniformemente possibile nei blocchi. Osserviamo che:**

- **più piccolo è il blocco rispetto alle tuple più è alto Fp ma più probabile è l'overflow,**
- **più grande è il blocco rispetto alle tuple più è basso Fp ma è meno probabile l'overflow.**

# Metodi per la funzione Hash

**1) MID-SQUARE:**

- **La chiave (convertita in numero se alfanumerica) è moltiplicata per se stessa ed i numeri centrali del quadrato vengono presi e normalizzati per rientrare nel "range" NB del numero di blocchi del file.**
- **ES: se un record ha una chiave di 6 cifre ed il file ha NB=7000 blocchi, il quadrato ha 12 cifre e si prendono le quattro dalla 5ª alla 8ª, se il numero è 172148, il quadrato è 029634933904 e le cifre centrali sono 3493.**

# Metodi per la funzione Hash

- **Per evitare che il numero così ottenuto sia maggiore di 7000, lo si normalizza moltiplicandolo per 7000/10000, cioè il numero del blocco che risulta è 0.7 x 3493 = 2445.**
- **I due numeri immediatamente precedenti e successivi danno:**
  **172146 ⇒ 3224 ⇒ 2397**
  **172147 ⇒ 3450 ⇒ 2415**
  **172149 ⇒ 3527 ⇒ 2469**
  **172150 ⇒ 3562 ⇒ 2493**
- **Due valori di K successivi non danno luogo a blocchi successivi**

# Metodi per la funzione Hash

**2) DIVISIONE**

- **La chiave K viene divisa per il numero primo più vicino ad NB ed il resto viene preso come numero del blocco**

**Es. 172148 viene diviso per 6997 (numero primo più vicino < NB) ed il resto è 4220**

**172146 ⇒ 4218**
**172147 ⇒ 4219**
**172149 ⇒ 4221**
**172150 ⇒ 4222**

- **Il metodo della divisione non è un "randomizzatore", ma assegna valori successivi di K a blocchi successivi.**

# Metodi per la funzione Hash

**3) SHIFTING**

- **Le cifre della chiave K vengono divise in frammenti, ognuna costituita da un numero di cifre pari al numero di cifre che rappresentano NB-1 blocchi**
- **i frammenti di K vengono sommati ed il risultato viene normalizzato.**

**Es: : 17207329 ⇒ 1720 | 7329 ⇒**

**1720 +**
**7329 =**
**9049**

**8 cifre ⇒ 4 cifre per NB**

# Metodi per la funzione Hash

**4) FOLDING**

- **I frammenti vengono "ripiegati" (fold) su se stessi e poi sommati, le cifre in eccesso vengono eliminate**
- **ES.: 17207329 ⇒ 17 | 207 | 329**

**8 cifre ⇒ 3 cifre**

**702 +**
**923 +**
**710 =**
**2335 ⇒ 335**

**SHIFTING e FOLDING vengono generalmente adoperati per chiavi alfa- numeriche molto lunghe**

# Esempio di funzione Hash

- **Esempio: sequenza di 22 registrazioni di cui si evidenzia solo la chiave, da memorizzare in un archivio di 13 blocchi la cui capacità è 2.**
- **funzione hash: i caratteri della chiave vengono convertiti in numeri prendendo i primi 4 bit della codifica BCD, successivamente se la stringa numerica così ottenuta è più lunga di 5 la si spezza in gruppi di 5 a partire da sinistra e poi si sommano i gruppi, infine si prende il resto della divisione per 13 (mod 13).**

# Metodi per la funzione Hash

| DATI | NUMERO | RESTO (mod 13) |
|---|---|---|
| 1 LUISA | 34921 | 4 |
| 2 EDVIGE | 54602 | 3 |
| 3 ADELE | 14535 | 2 |
| 4 AGNESE | 17557 | 8 |
| 5 FRANCESCA | 74384 | 12 |
| 6 ROBERTA | 96290 | 13 |
| 7 NOEMI | 56549 | 13 |
| 8 ELENA | 53551 | 5 |
| 9 LAURA | 31491 | 6 |
| 10 BEATRICE | 26074 | 10 |
| 11 DIANA | 49151 | 12 |
| 12 IRENE | 99555 | 2 |
| 13 PAOLA | 71631 | 2 |
| 14 BERENICE | 26890 | 7 |
| 15 VANESSA | 51573 | 3 |
| 16 SARA | 02191 | 8 |
| 17 CARLA | 31931 | 10 |
| 18 OLGA | 06371 | 2 |
| 19 GIORGIA | 79788 | 8 |
| 20 LARA | 03191 | 7 |
| 21 CAROLA | 31964 | 11 |
| 22 GIOVANNA | 80202 | 6 |

# Metodi per la funzione Hash

| indirizzo | ARCHIVIO | | puntatore |
|---|---|---|---|
| 1 | | | |
| 2 | ADELE | IRENE | 14 |
| 3 | EDVIGE | VANESSA | |
| 4 | LUISA | | |
| 5 | ELENA | | |
| 6 | LAURA | GIOVANNA | |
| 7 | BERENICE | LARA | |
| 8 | AGNESE | SARA | 15 |
| 9 | | | |
| 10 | BEATRICE | CARLA | |
| 11 | CAROLA | | |
| 12 | FRANCESCA | DIANA | |
| 13 | ROBERTA | NOEMI | |
| 14 | PAOLA | OLGA | overflow record |
| 15 | GIORGIA | | overflow record |

**Come si può vedere i blocchi 1 e 9 non vengono riempiti, mentre 2 ed 8 vanno in overflow.**

# COSA VUOL DIRE "METODO MIGLIORE"?

- **Il metodo migliore è quello che distribuisce il più uniformemente possibile le tuple, evitando di riempire troppo alcuni blocchi lasciandone vuoti altri. In generale bisogna fare esperimenti e poi eventualmente cambiare funzione Hash e/o il numero di blocchi NB.**
- **Il metodo Mid-square e quello che più si avvicina ad un generatore di valori random. Il metodo della divisione è in generale il migliore perché "distribuisce" meglio le tuple.**

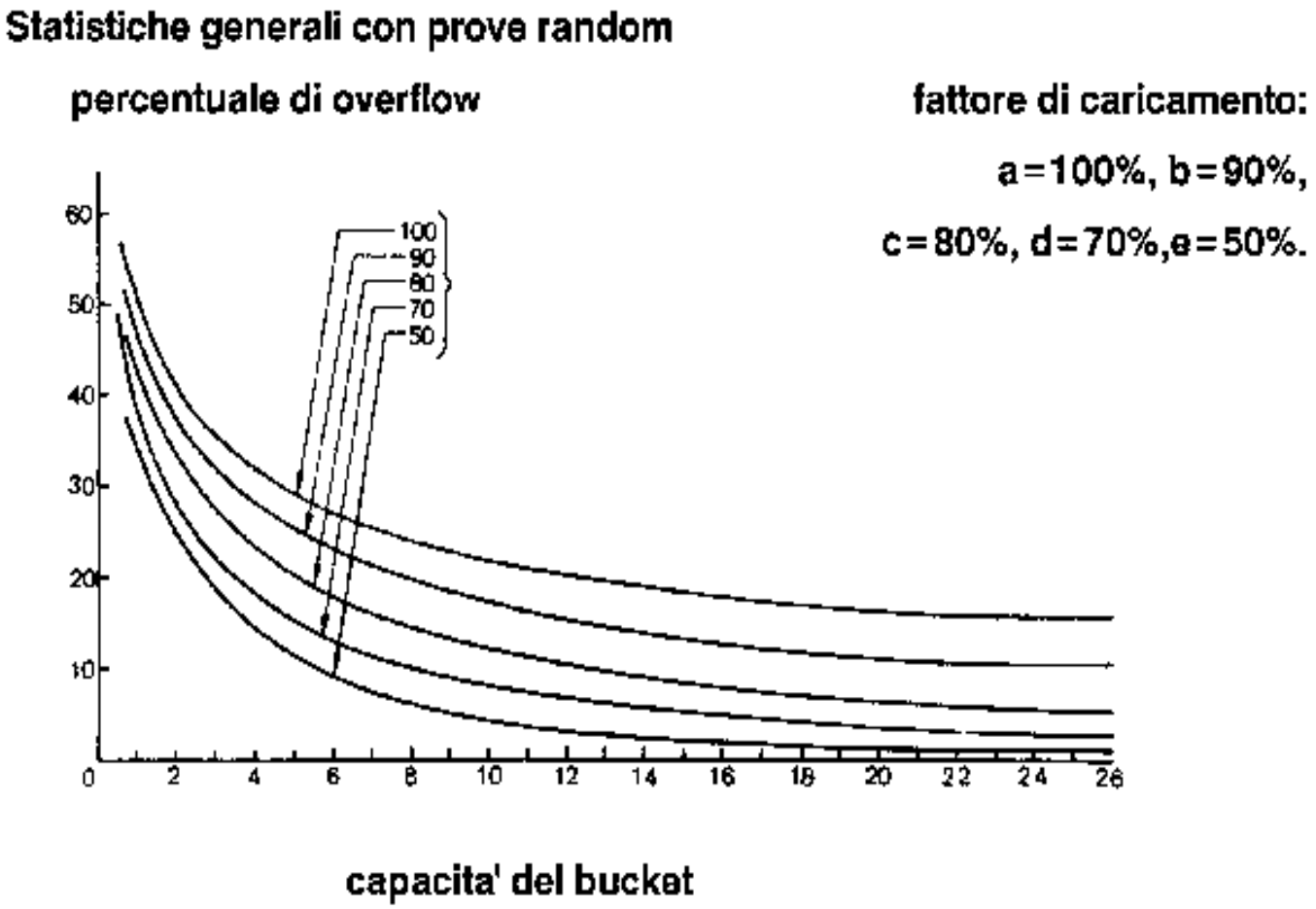
# DIMENSIONAMENTO E PRESTAZIONI

- **Per un FILE HASH si devono definire:**
  - **un numero NB di blocchi che costituiscono la "PRIME AREA"**
  - **un numero NO di blocchi per la "OVERFLOW AREA"**

- **Per effettuare il dimensionamento si utilizzano dati statistici sulla percentuale di overflow**

- **Il costo di accesso $C_{hash}$ (es. ricerca di una tupla) sarà**
  - **1** **in caso standard**
  - **>=2** **in caso di overflow**

# DIMENSIONAMENTO E PRESTAZIONI



# DIMENSIONAMENTO E PRESTAZIONI

- **Es.: 10000 tuple di 200 bytes con blocchi di 4096, capacità = 20**

  **usando le curve del grafico si ha che:**
- **caricamento del 100% overflow al 17%,**

  **NB= 10000/20= 500, NO = 85, tot.= 585**

  **$C_{hash}$ = 1 nel 83% dei casi ≥ 2 nel 17%**
- **caricamento del 70% overflow al 3%,**

  **NB= 10000/14=715, NO= 22, tot. = 737**

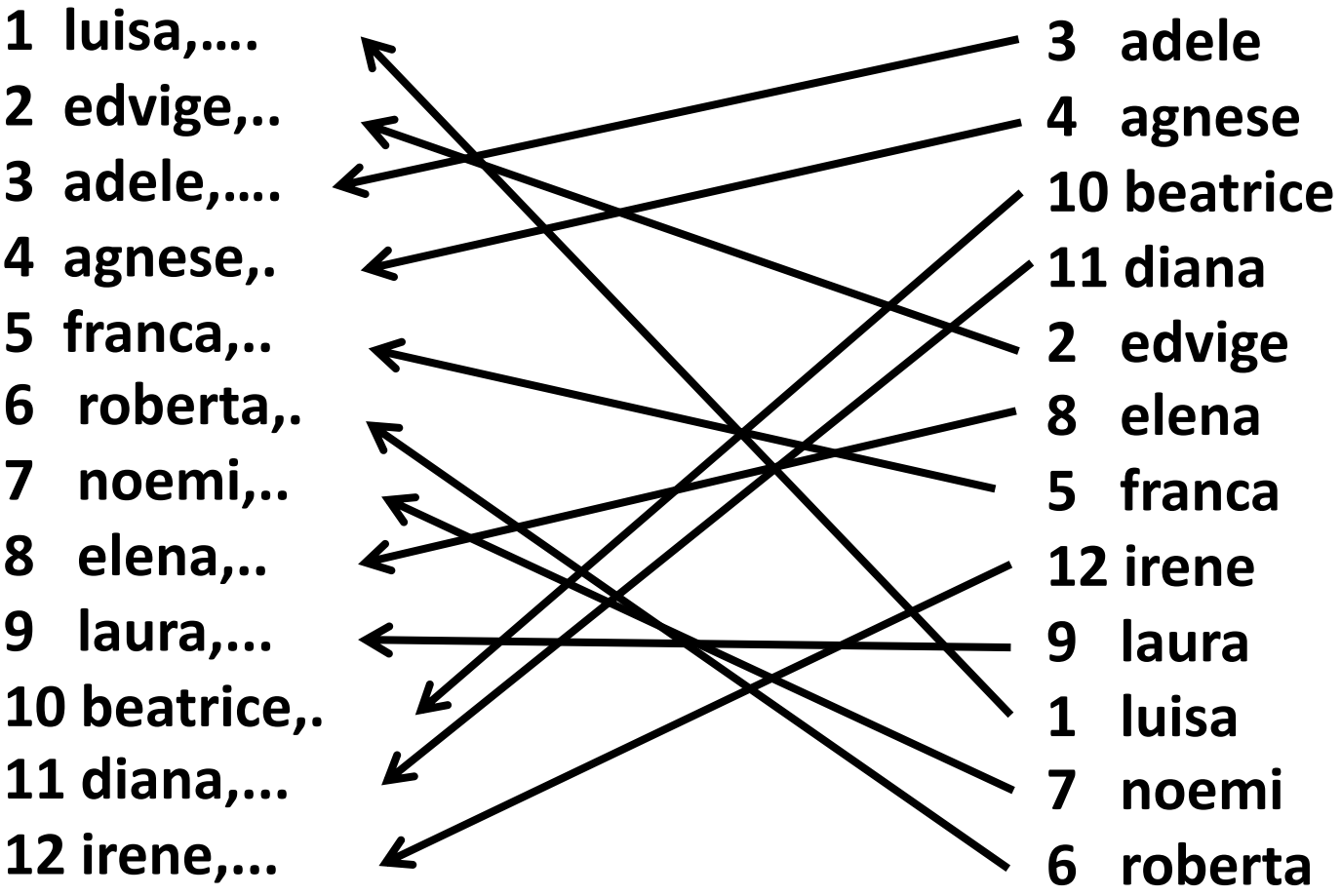
  **$C_{hash}$ = 1 nel 97% dei casi ≥ 2 nel 3%**

# Organizzazioni con indice

# Organizzazioni con indice

- **IDEA DI BASE : associare ad un file una tabella nella quale l'entrata i-esima memorizza una coppia del tipo ($k_i$,$p_i$) dove :**
  - **$k_i$ è un valore di chiave**
  - **$p_i$ è un riferimento al record (tupla) con valore di chiave $k_i$**
  - **l'indice è ordinato: le coppie ($k_i$ ,$p_i$) sono ordinate in base ai valori di k**

# Organizzazioni con indice

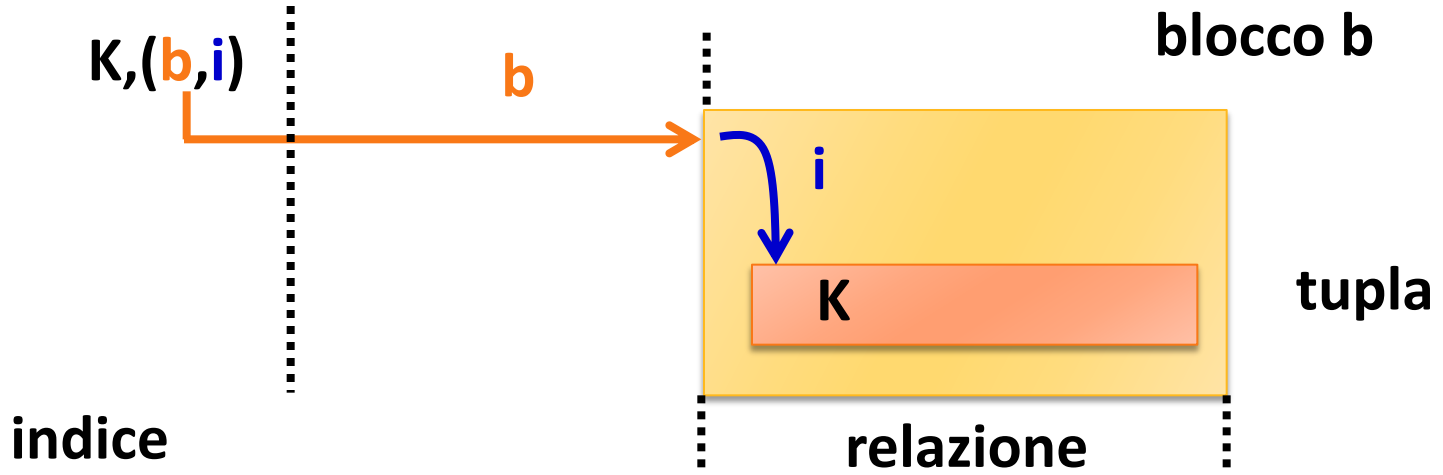


# Organizzazioni con indice

- **Nell'esempio abbiamo mostrato solamente la colonna chiave e la posizione della tupla, mentre l'indice è un file costituito da mini-tuple ed occupa generalmente uno spazio di memoria di circa 5÷20% della relazione**
- **ordinare il file indice è quindi più veloce**
- **l'indice può essere costruito su attributi qualsiasi (anche non chiave nel senso relazionale)**
- **l'indice fornisce una visione ordinata della relazione anche su colonne non di ordinamento**

# Organizzazioni con indice

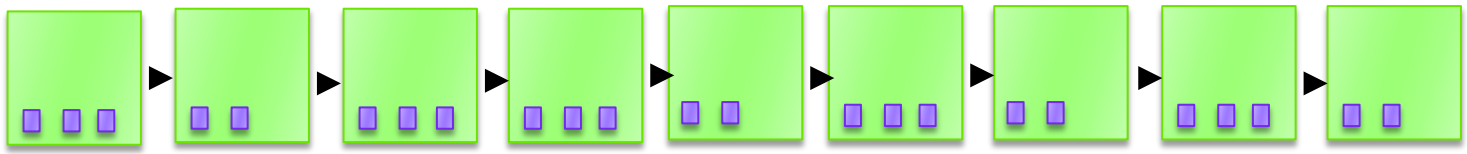
- I riferimenti (4÷8 byte) vengono denominati: indirizzi, puntatori o, più comunemente, TID (tuple identifier)
- sono costituiti da una coppia: id. di blocco (pagina), indirizzo nel blocco:




L'indirizzo nel blocco è a sua volta indiretto

# Organizzazioni con indice

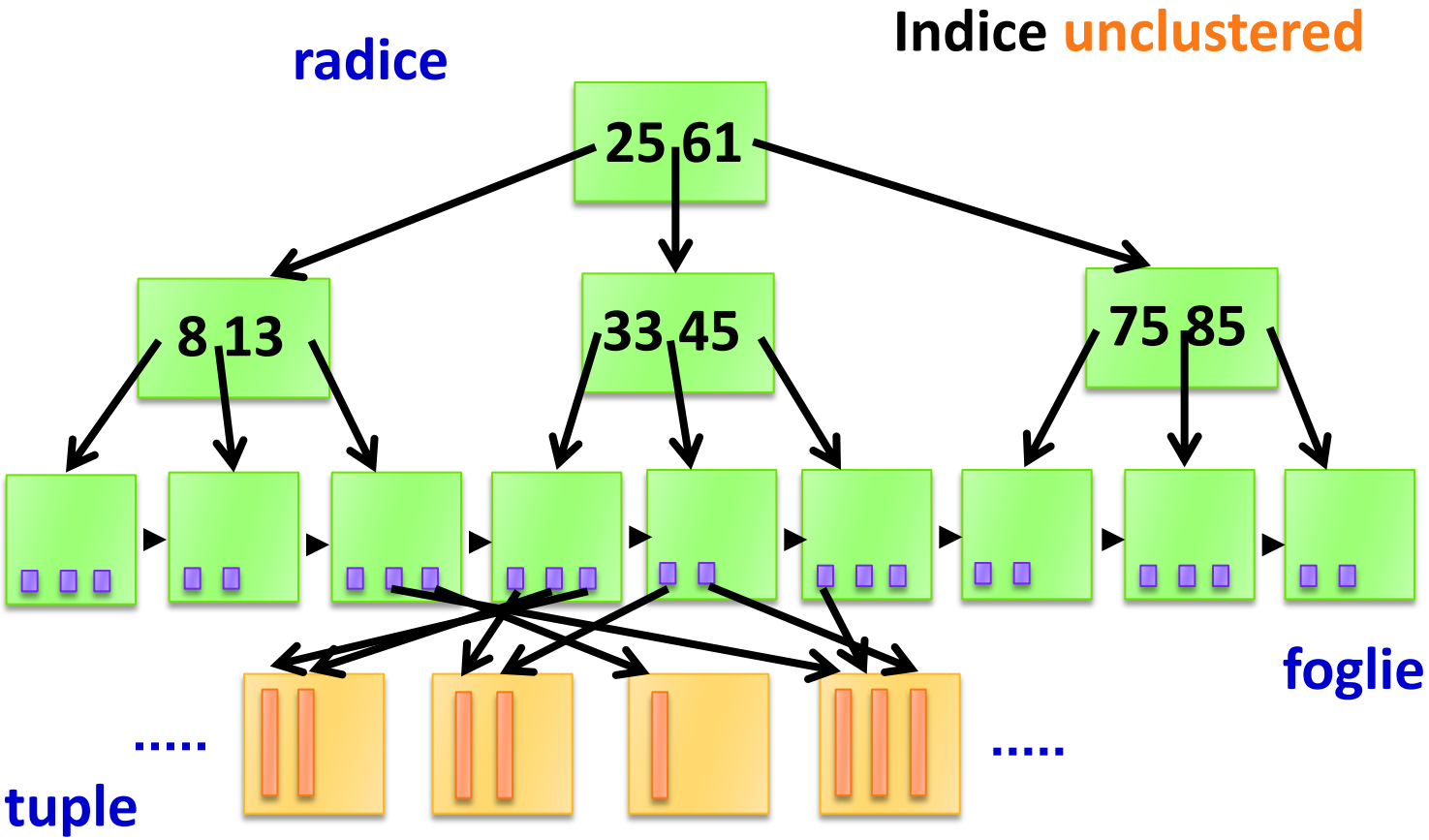
- **Metodo di accesso**
  **data una chiave K :**
  **1. accesso all'indice**
  **2. ricerca della coppia (K,(b,i))**
  **3. accesso al blocco dati b**
  **4. accesso alla tupla contenente K (indirizzo i)**
- **le coppie (K,(b,i)) vengono collezionate in blocchi ed ordinate, questi blocchi si chiamano FOGLIE dell'indice e sono tra loro collegati da puntatori.**



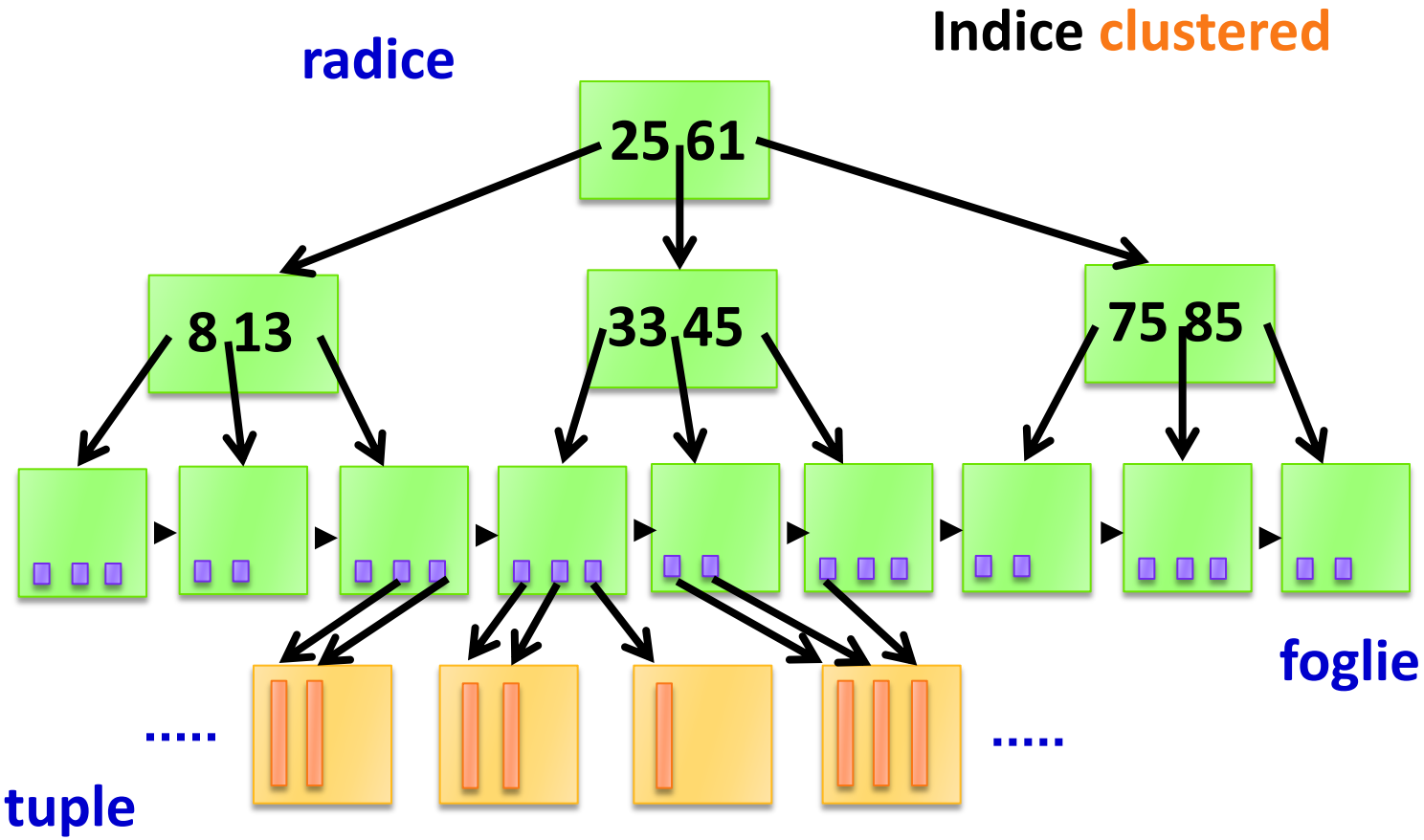
# Organizzazioni con indice

- Al di sopra delle foglie si costruisce un'organizzazione ad albero a più livelli di indirizzamento.
- Organizzazione blocco sopra foglie
  - ogni foglia *i* è rappresentata da una coppia ($K_i$,$P_i$) nel livello immediatamente superiore dove
    - $K_i$ è il valore minore contenuta nella foglia *i*
    - $P_i$ è l'identificativo del blocco foglia *i*
  - $K_1$ è preceduta da puntatore $P_0$ per accedere a valori di chiavi inferiori a $K_1$
- i blocchi del livello sopra le foglie sono a loro volta rappresentati a livello superiore , e così via fino ad avere un solo blocco chiamato radice

# Organizzazioni con indice



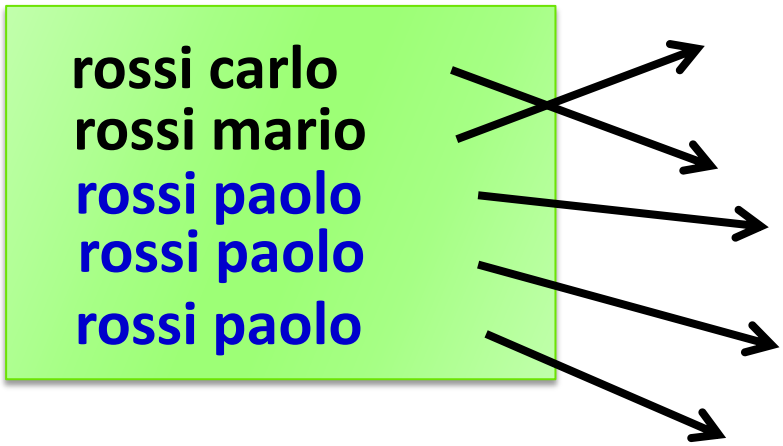
# Organizzazioni con indice



# Indici

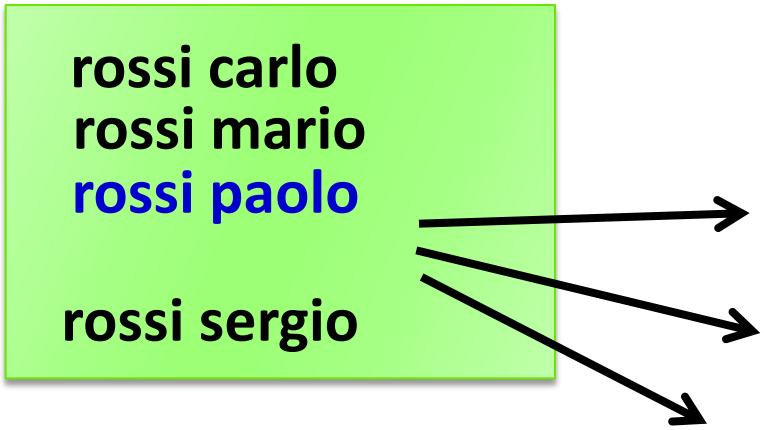
- L'indice si dice clustered se è costruito su un attributo di ordinamento, altrimenti unclustered
- su una relazione si possono costruire un indice clustered e più di un indice unclustered
- l'indice può essere anche multiattributo (multicolonna), ad es. K :<cognome,nome>
- nel caso di clausola UNIQUE l'indice garantisce la non ripetizione dei valori
- l'indice può essere costruito su attributi con valori ripetuti, in tal caso la coppia (K, p) diventa (K,(p1, p2,…. pk)):

# Indici

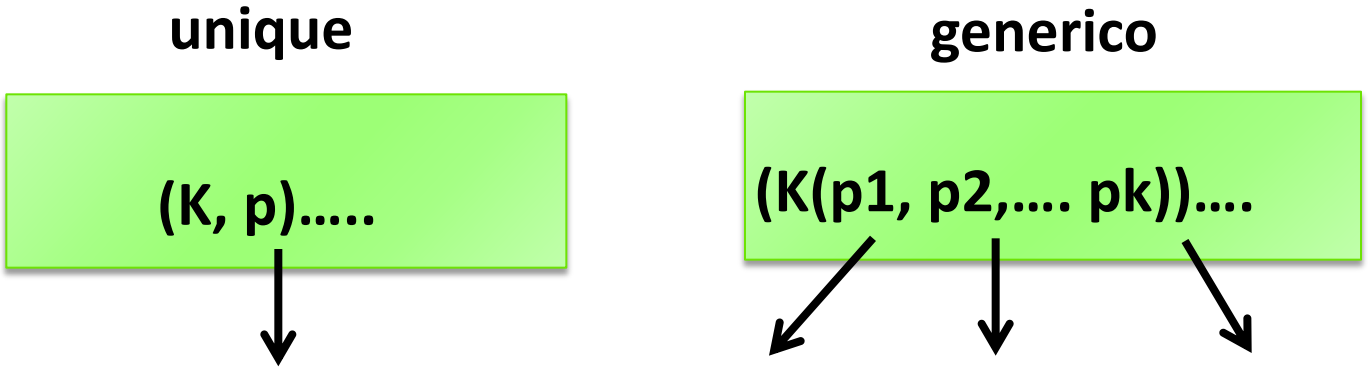



valori ripetuti

eliminazione dei valori ripetuti e riduzione di spazio



# Indici




**In SQL:**

- **CREATE {UNIQUE} INDEX nome-indice**
  **ON nome-relazione**
  **(nome-attributo { ASC | DESC })**
- **DROP INDEX nome-indice**

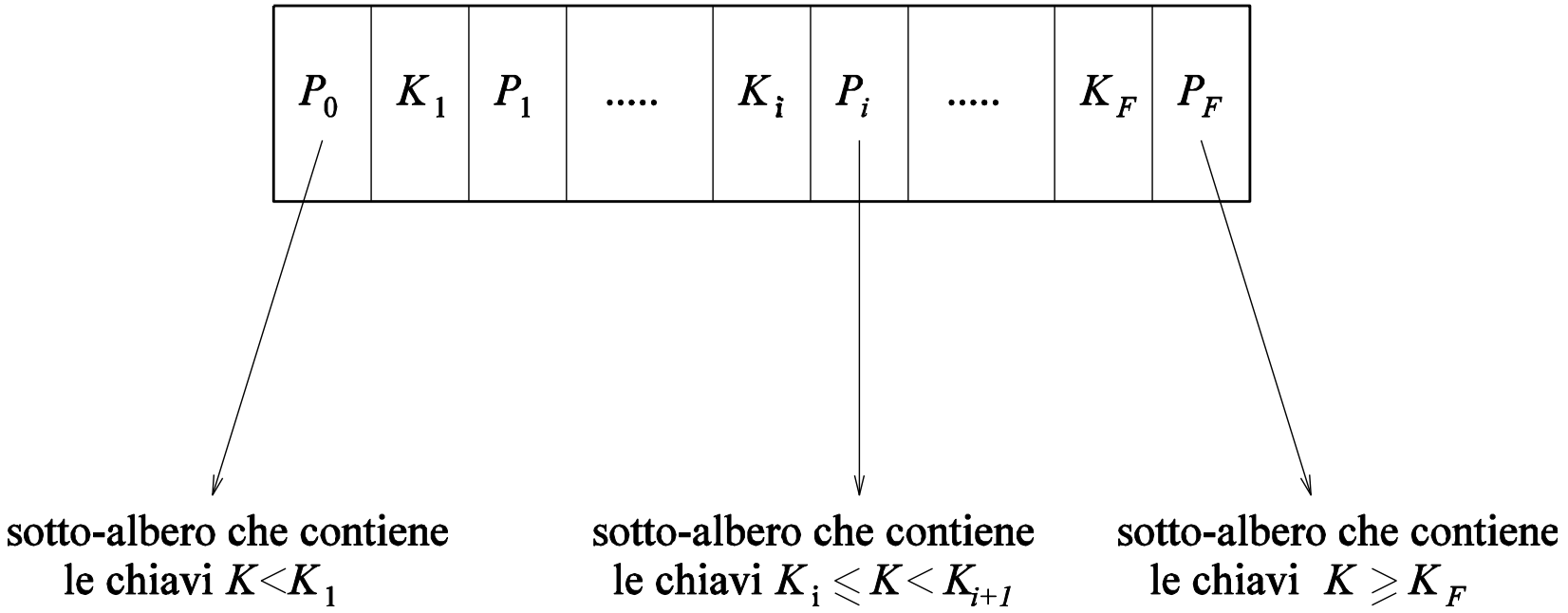
# Indici $B^+$ tree

## Struttura dati ad albero per memoria secondaria, dinamica e bilanciata

- l'albero è associato ad un SW di gestione che lo mantiene **sempre bilanciato**: ogni percorso dalla radice ad una foglia ha sempre la stessa **lunghezza h**, chiamata **altezza** del $B^+$ tree
- Ogni **nodo** corrisponde ad un **blocco** del file system
- **Dimensione nodo O**: ogni nodo intermedio ha sempre almeno **O+1 e non più di 2xO+1** figli, contiene almeno **O e non più di 2xO** valori dell'attributo
- la radice può avere da **2 a non più di 2xO +1** figli, contiene da **1 a non più di 2xO** valori
- **O** si chiama **ordine** del $B^+$ tree
- Le foglie sono riempite dal **50 al 100 %**

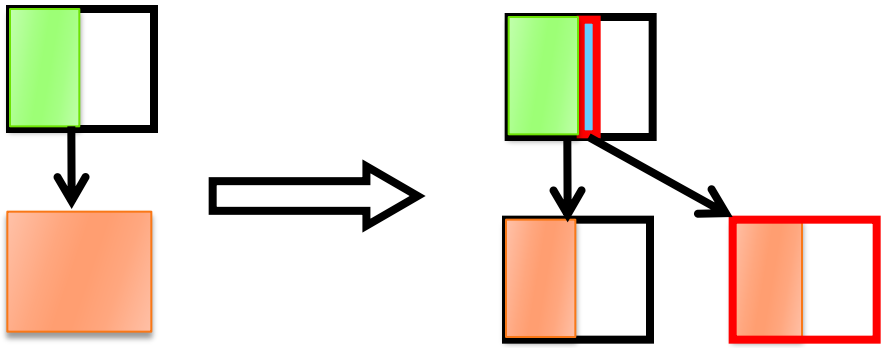
  (valore tipico: 69%)

# Organizzazione dei nodi del B+-tree



# Indici B+ tree

**inserimento**



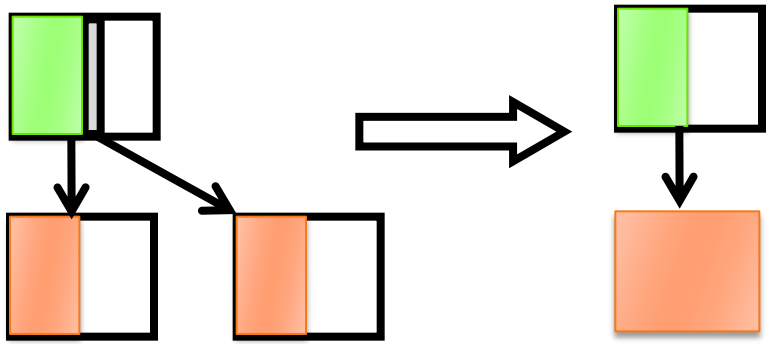
**block split**

**(si può propagare fino alla radice)**

**eliminazione**

**underflow**

**(si può propagare fino alla radice)**



# Prestazioni del $B^+$ tree

**Facciamo riferimento a:**

- **dimensione del blocco D (es. 4096)**
- **Dimensione del puntatore L(p) ai nodi intermedi (es. 4 per entrambi)**
- **dimensione L(k) del valore K**

**Calcoliamo :**

- **l'ordine O**
- **il numero di foglie NF**
- **l'altezza H**

# Prestazioni del $B^+$ tree

**Ordine O:**

**consideriamo il vincolo**

$$2xOxL(k) + (2xO + 1)xL(p) \leq D$$

**da cui**

$$O = \lfloor ( D - L(p)) / (2x(L(k) + L(p)) \rfloor$$

- **con una dimensione L(k) di 10 si ha**

  **O = 146 e 4 byte inutilizzati per nodo**

- **con una dimensione L(k) di 40 si ha**

  **O = 46 e 44 byte inutilizzati per nodo**

# Prestazioni del $B^+$ tree

**Numero di foglie NF:**

- dipende dal numero di tuple NT e dal fattore di riempimento medio dei blocchi = ln 2 ≃ 69 %
- ricordiamo che nelle foglie si avranno un numero di p = NT ed un numero di valori NK da cui

$$NF = \lceil ( NK \times L(k) + NT \times L(p)) / (D \times \ln 2 ) \rceil$$

- con una dimensione L(k) = 10, NT = $10^6$ , NK = $10^4$ si ha:

$$NF = 5776$$

# Prestazioni del $B^+$ tree

**Altezza minima:**

- **l'altezza minima Hmin si può ottenere quando tutti i nodi sono pieni,**

  **poiché al penultimo livello sono necessari NF puntatori alle foglie, si ha:**

$$NF \leq ( 2 \times O + 1)^{Hmin-1}$$ **e quindi**

$$Hmin = 1 + \lceil \text{Log}_{( 2 \times O + 1)} NF \rceil$$

# Prestazioni del $B^+$ tree

**Altezza massima:**

- **l'altezza massima Hmax si può ottenere quando tutti i nodi sono pieni per metà e la radice contiene due soli puntatori, poiché al penultimo livello sono necessari NF puntatori alle foglie, si ha:**

$$NF \leq 2 \times (O + 1)^{Hmax-2}$$ **e quindi**

$$Hmax = 2 + \lceil \log_{(O+1)} (NF / 2) \rceil$$

# Prestazioni del $B^+$ tree

- Esempio

D = 1024, L(p) = 4, NT = $10^6$

| | L(k) = 10 | | | L(k) = 20 | | |
|---|---|---|---|---|---|---|
| NK | NF | Hmax | Hmin | NF | Hmax | Hmin |
| $10^3$ | 5650 | 5 | 4 | 5664 | 5 | 4 |
| $10^4$ | 5776 | 5 | 4 | 5918 | 5 | 4 |
| $10^5$ | 7045 | 5 | 4 | 8454 | 5 | 4 |
| $10^6$ | 19725 | 5 | 4 | 33814 | 6 | 4 |

si tratta in generale di alberi abbastanza bassi

# Prestazioni del $B^+$ tree

- Continuando l'esempio precedente, con NB= $10^5$, $10^6$ tuple, 10 per blocco L(k) = 10
  - $C_{seq} = 5 \times 10^4$
  - $C_{sort} = 1.2 \times 10^6$ e $C_{bin} = 17$
  - $C_{ind} = 4 + 1 = 5$,
    bisogna sommare l'accesso al blocco che contiene la tupla cercata (+1)
- si può anche ipotizzare che se le richieste sono molte, durante l'esercizio, la radice ed i livelli superiori siano già in memoria:
  - $C_{ind} = 1 + 1 = 2$

# Prestazioni del $B^+$ tree

- **costo di costruzione dell'indice:**
- **lettura della relazione,**
- **estrazione delle coppie, costruzione delle foglie**
- **sort/merge delle stesse**

$$C_{indice} = NB + NF + 2 \times NF \times (PM+1)$$

- **continuando l'esempio precedente, con M e NM = 8 ed inglobando il passo di sort nella estrazione delle coppie:**

$$C_{indice} = NB + NF + 2 \times NF \times PM$$

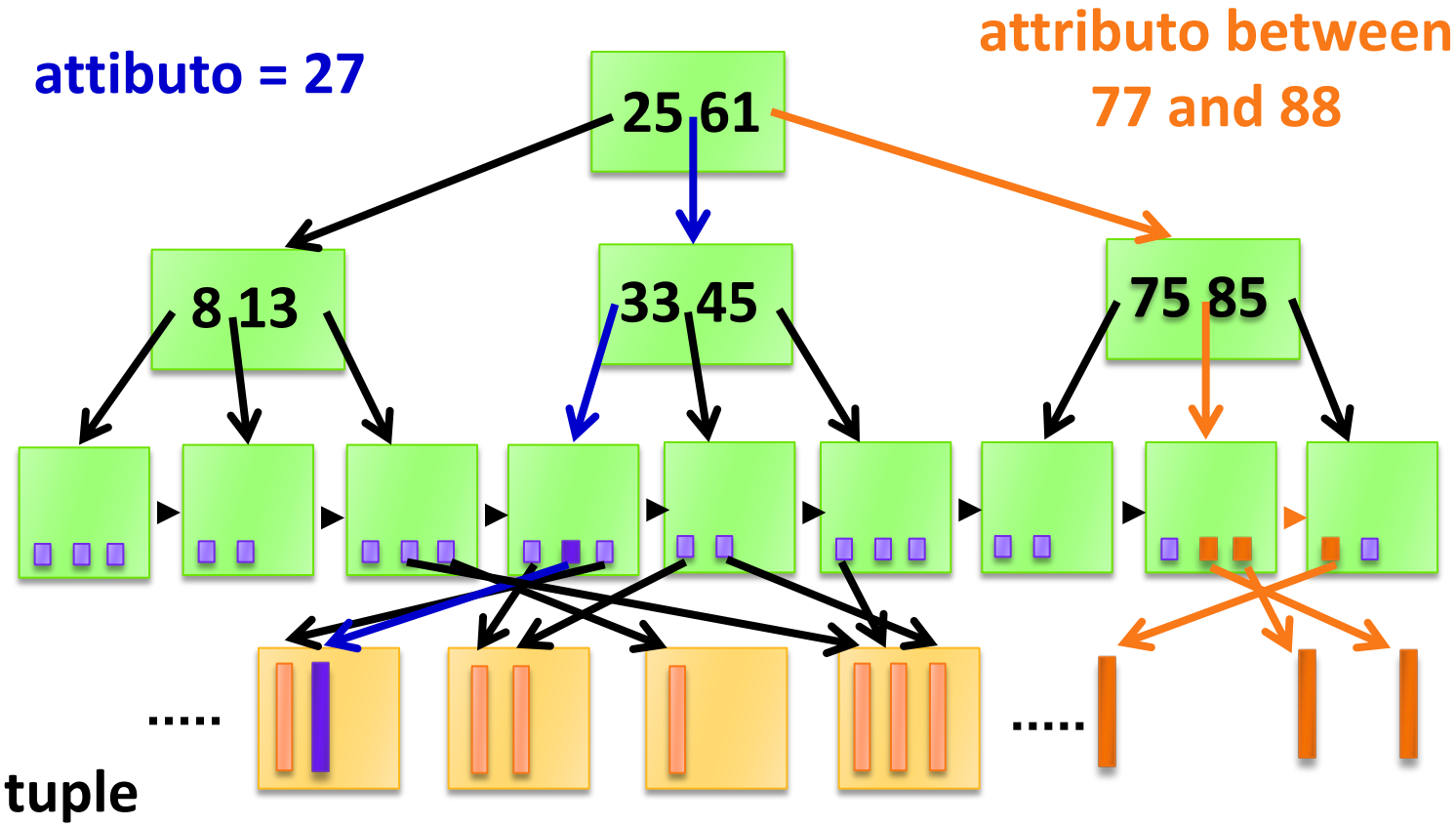
$$C_{indice} = 10^5 + 1.98 \times 10^5 \simeq 3 \times 10^5$$

# Prestazioni del $B^+$ tree

**Utilizzo dell'indice:**

- **accesso veloce alla tupla per K unique**
- **accesso veloce alle tuple per K replicata**
- **accesso veloce alle tuple per range di K**
  **(K < valore, K BETWEEN k1 AND k2)**
- **facilita ORDER BY e GROUP BY**

# Organizzazioni con indice



# Compressione della chiave

- **FORWARD COMPRESSION**
  - **Si registrano: il numero di caratteri uguali ed i caratteri finali che differenziano dal valore precedente.**
  - **Es:**

| | |
|---|---|
| JOHNSON | 0 JOHNSON |
| JOHNSTON | 5 TON |
| JOHNSTOWN | 7 WN |
| JOHNTON | 4 TON |

- **REAR COMPRESSION (Riporta i caratteri iniziali) è inefficace**