UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Progettazione fisica

Strutture e metodi d’accesso ai dati su disco

Prof. Stefano Montanelli

DBMS e sistema operativo

- I dati sono memorizzati su **memoria secondaria** per questioni di persistenza
- I dati in memoria secondaria sono organizzati in **blocchi** di dimensione fissa
- L'elaborazione dei dati avviene in **memoria principale**
- Il **buffer** i) permette la gestione dei dati in memoria principale, ii) applica strategie per minimizzare il trasferimento dei dati da elaborare da/verso la memoria secondaria
- Il buffer è organizzato in **pagine** che hanno dimensione pari a X blocchi

Fattore di blocco (blocking factor)

- Il fattore di blocco (***bfr***) è il numero di record contenuti in un blocco

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor$$

- **B** è la dimensione del blocco, **R** è la dimensione media del record
- Se i record hanno tutti la stessa dimensione in byte, allora parliamo di *record a lunghezza fissa*, altrimenti di *record a lunghezza variabile*

Strutture primarie dei file

- La struttura primaria stabilisce il criterio che determina la disposizione delle tuple/record nei file
- Le strutture primarie possono essere suddivise in tre tipologie in base al metodo d'accesso ai dati:
  - Sequenziale (non ordinato, ad array, ordinato)
  - Calcolato (hash)
  - Albero

Strutture ad accesso sequenziale

- Nelle strutture sequenziali, i file sono costituiti da blocchi logicamente consecutivi e le tuple sono inserite rispettando un criterio sequenziale:

  A. Struttura non ordinata (file **heap**)

     - Sequenza indotta dall'ordine di inserimento
     - Inserimento efficiente (ma attenzione ai vincoli di chiave)
     - Ricerca lineare (migliorabile con strutture secondarie)
     - Cancellazione logica con periodiche ristrutturazioni

Strutture ad accesso sequenziale

- Nelle strutture sequenziali, i file sono costituiti da blocchi logicamente consecutivi e le tuple sono inserite rispettando un criterio sequenziale:

  B. Struttura sequenziale ad array

  - Possibile solo con record a lunghezza fissa
  - Il file occupa *n* blocchi e ciascun blocco ospita *m* posizioni dell'array
  - Ogni tupla ha un indice *i* che determina la posizione della tupla nel file (condizione raramente soddisfatta dai dati)
  - Inserimenti e ricerche efficienti

Strutture ad accesso sequenziale

- Nelle strutture sequenziali, i file sono costituiti da blocchi logicamente consecutivi e le tuple sono inserite rispettando un criterio sequenziale:

  C. Struttura sequenziale ordinata

  - Ordinamento fisico dei dati nel file coerente con l'ordinamento di un campo detto chiave (*pseudochiave*)
  - Operazioni efficienti sul campo chiave (sia per *ricerche puntuali* sia per *selezioni su intervallo*)
  - Richiede l'uso di indici per ricerche efficienti (i.e., dicotomiche)
  - Inserimenti e cancellazioni possono i) essere costose, ii) avvenire su un file di overflow, iii) richiedere periodiche ristrutturazioni

Campi di ordinamento (pseudochiavi)

- Il campo di ordinamento di una struttura ordinata può essere costituito da uno o più attributi della relazione
- L'ordinamento avviene sul primo attributo, quindi sui successivi a parità di valore sul primo attributo (e successivamente sui precedenti)
- Il campo di ordinamento della struttura NON è necessariamente la chiave primaria della relazione

Strutture ad accesso calcolato (hash)

- Nelle strutture ad accesso calcolato, la posizione di una tupla nel file dipende dal valore assunto da un campo chiave
- Si utilizza una **funzione hash** *h* per trasformare il valore del campo chiave *k* in un indice di posizione nel file (e.g., h(k) = k mod N, dove *N* è la numerosità delle posizioni a disposizione)
- La soluzione è applicabile solo con record a lunghezza fissa
- Ricerca puntuale efficiente, ricerca per intervallo non efficiente
- Richiede strategie di gestione delle collisioni (*catene di overflow*)

Alberi (indici)

- L’organizzazione ad albero può essere impiegata sia per realizzare strutture primarie (i.e., strutture contenenti i dati) sia strutture secondarie (i.e., strutture ausiliarie mirate a favorire l’accesso ai dati memorizzati in altre strutture)

- Quindi distinguiamo:

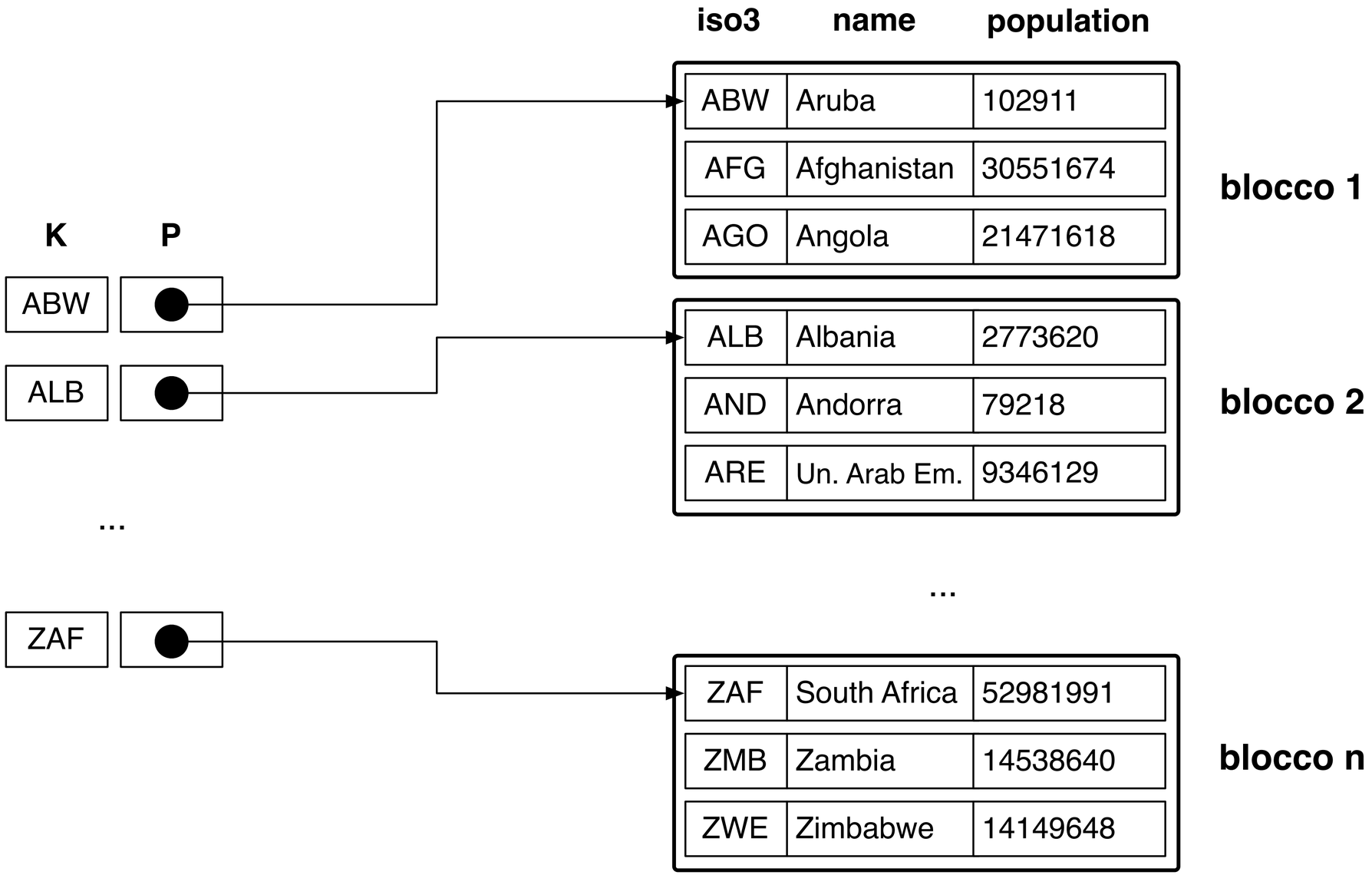
  - **Indici primari**

  - **Indici secondari**

Indici primari

- Un indice primario contiene i dati della relazione (i.e., i record) al suo interno
- Un indice primario garantisce accesso ai dati in base alla pseudochiave usata come campo di ordinamento dei record e ne determina la posizione
- Ciascuna voce dell'indice ha la forma $< k, p >$ dove k è il valore della pseudochiave e p è un puntatore a un'area di memoria
- Il puntatore p può fare riferimento all'inizio del blocco dove il record con chiave k è memorizzato, oppure può tenere conto dell'offset all'interno del blocco

Indici primari



Indici primari

- Il numero delle voci dell'indice primario è uguale al numero di blocchi che costituiscono il file

- In genere, gli indici possono essere **densi**, ovvero contenere una voce per ogni valore della pseudochiave, oppure **sparsi**, ovvero contenere voci solo per alcuni dei valori della pseudochiave

- **Un indice primario è sempre un indice sparso**, poiché un blocco contiene più record

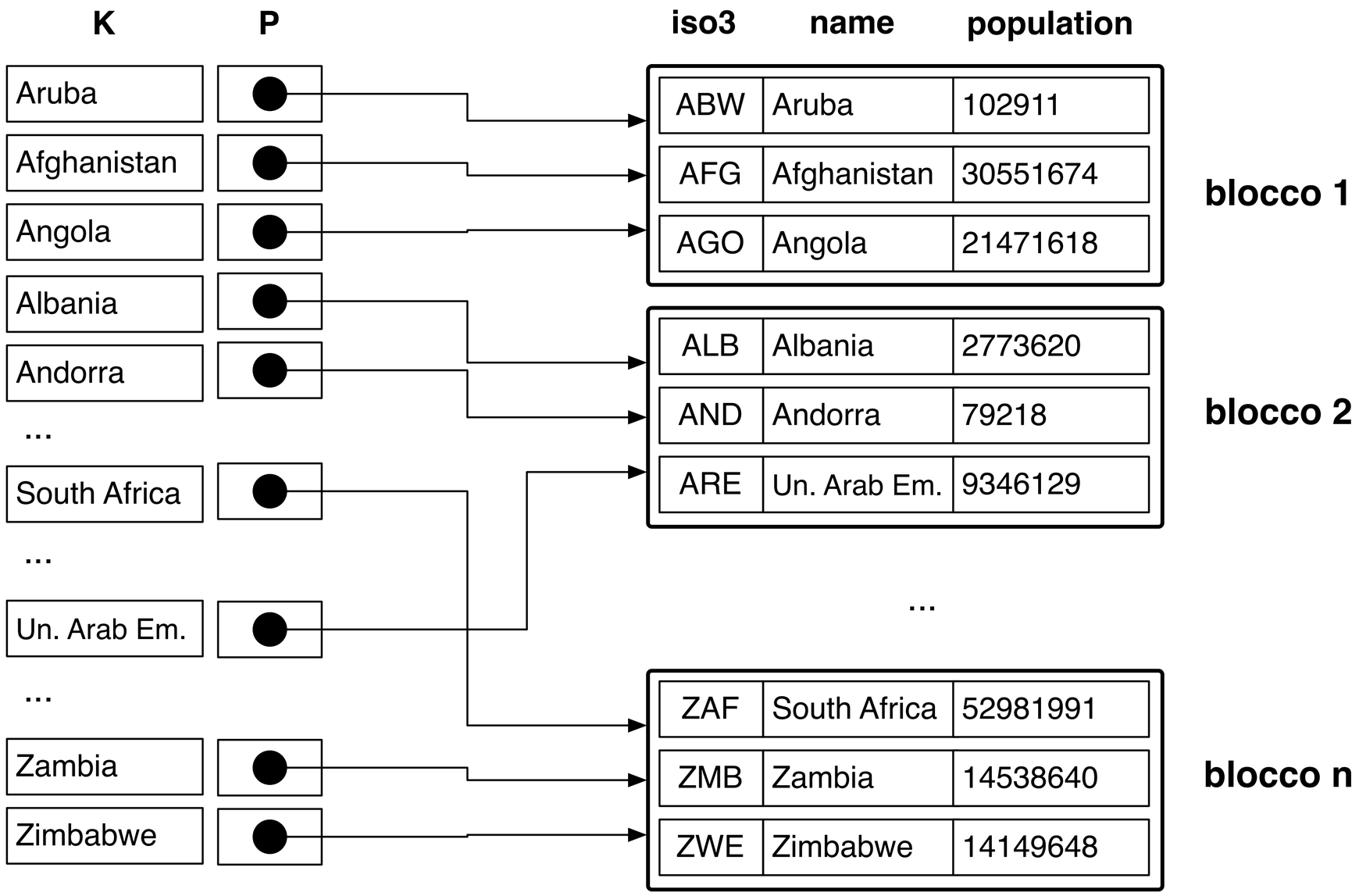
Esempi su indici primari

- **Esempio 1**
  - Record di dimensione fissa R = 100 byte
  - File con r = 30.000 record e blocchi B = 1024 byte
  - Quante voci conterrà l'indice?
- **Esempio 2**
  - I valori della pseudochiave dell'indice occupano 3 byte
  - Il puntatore al blocco è un indirizzo di 6 byte
  - Quanti blocchi occupa l'indice?

Indici secondari

- Un indice secondario fornisce un'ulteriore struttura di accesso a un file per il quale ci sia già un indice primario
- Il file contenente i record può essere non ordinato, ad accesso calcolato (hash) o ordinato, ma non rispetto al campo di indicizzazione secondaria
- Il campo di ordinamento dell'indice secondario può essere una chiave (ovvero avere valori univoci) o un qualsiasi attributo della relazione (anche non univoco)
- Un indice secondario deve necessariamente contenere tutti i valori del campo di ordinamento, quindi **gli indici secondari sono sempre indici densi**

Indici secondari



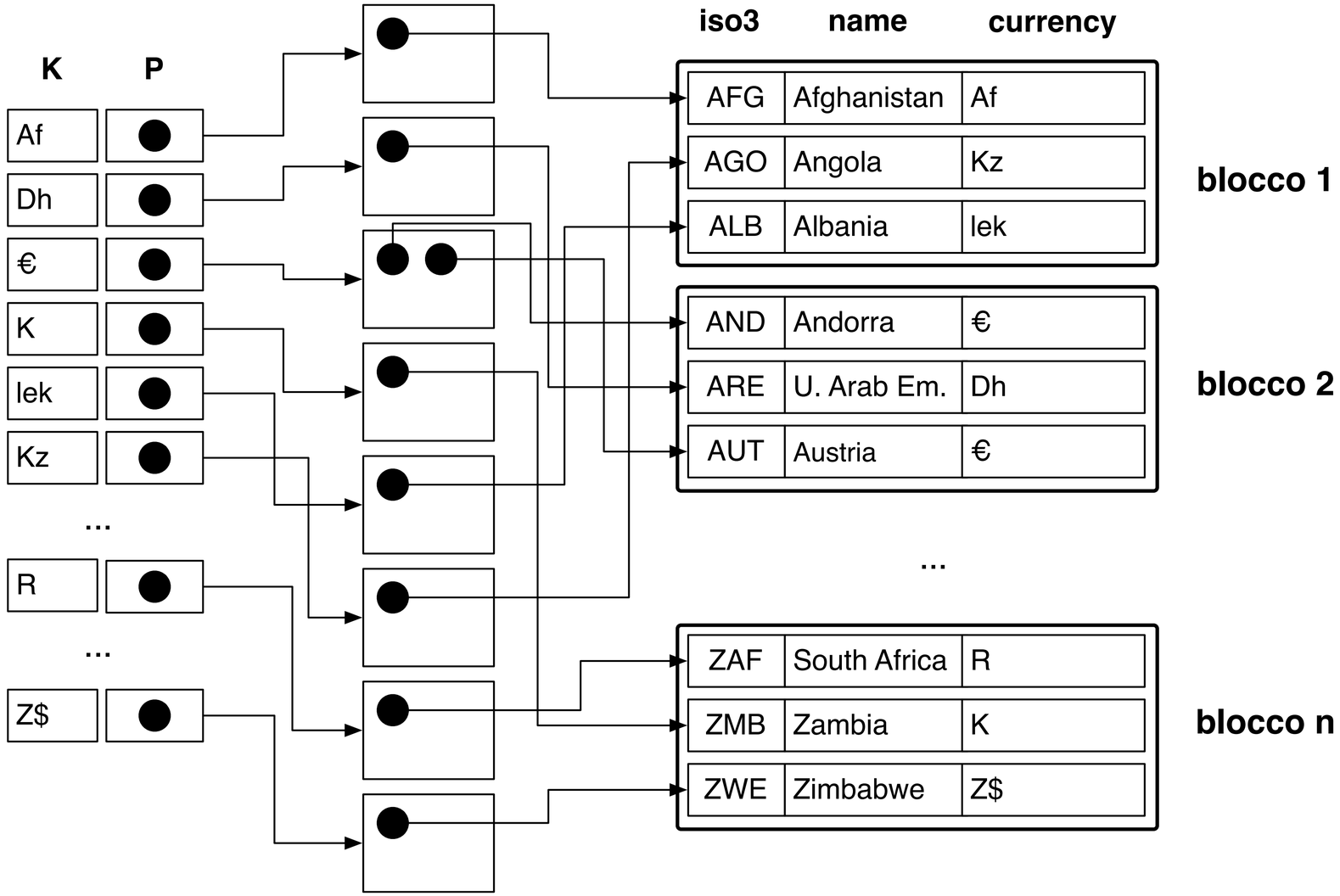
Esempi su indici secondari

- **Esempio**
  - Consideriamo un file che abbia r = 30.000 record di dimensione R = 100 byte e B = 1024 byte per blocco
  - Quanti accessi sarebbero mediamente necessari con una ricerca lineare su un file non ordinato?
  - Quanti accessi sarebbero necessari con una ricerca binaria assumendo una dimensione di R = 15 byte per ciascuna voce dell'indice?

Indici secondari su campi non chiave

- Se il campo di indicizzazione secondario può avere valori duplicati, vi sono tre opzioni di memorizzazione dell'indice:
  - Inserire più voci dell'indice con medesimo valore di K
  - Usare un record a lunghezza variabile per l'indice in modo da inserire più puntatori per ogni voce dell'indice
  - Mantenere voci a lunghezza fissa, ma inserendo un ulteriore livello per i puntatori (si veda lo schema della slide successiva per un esempio)

Indici secondari su campi non chiave



Considerazioni

- Un file può avere un solo indice primario (che determina la posizione dei record nei blocchi di memoria secondaria)
- Un file organizzato con accesso sequenziale non ordinato può essere affiancato da un indice primario per favorire le ricerche puntuali sulla pseudochiave
- Un file organizzato con accesso hash o sequenziale ordinato NON può avere un indice primario
- Un file può avere numerosi indici secondari

Considerazioni

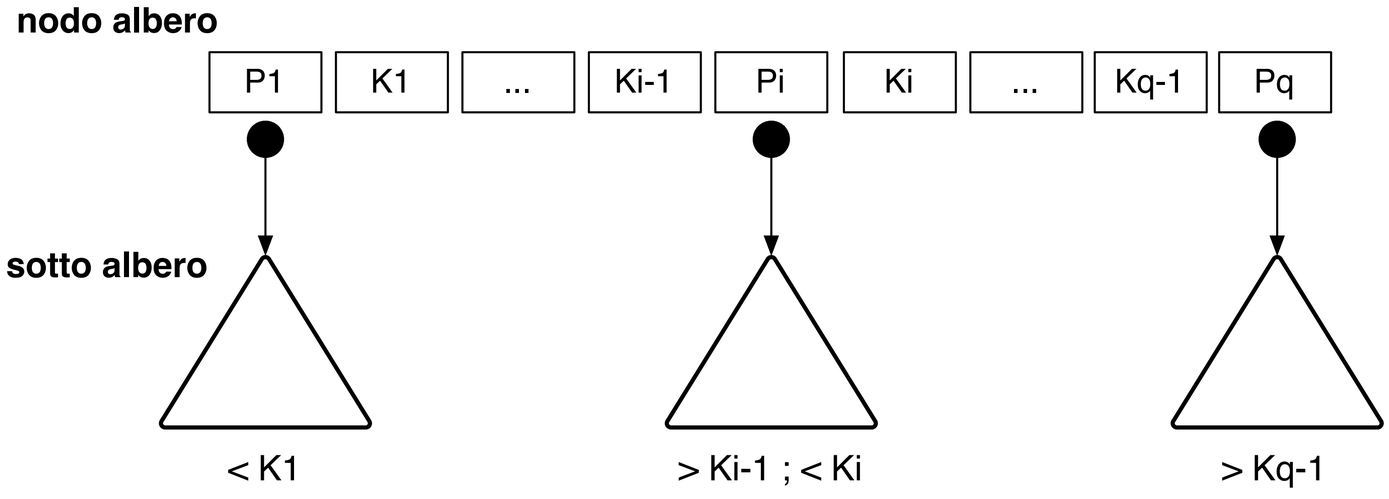
- Gli indici sono file di piccole dimensioni
- Le ricerche sui file di indice sono efficienti (occupano poche pagine e possono essere interamente caricati nel buffer)
- Essendo ordinati, gli indici rendono efficienti sia le ricerche puntuali sia le ricerche per intervallo
- Gli indici hanno tempi di accesso logaritmico in funzione del numero di blocchi occupati

Alberi di ricerca

- Un albero di ricerca di ordine *p* è un albero tale per cui ogni nodo contiene al massimo *p – 1* valori di ricerca e i *p* puntatori sono definiti come segue:

$$\langle P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q \rangle$$

- Vincoli:
  - in ciascun nodo: $K_1 < K_2 < \ldots < K_{q-1}$
  - per tutti i valori X del sottoalbero a cui si riferisce P_i si ha $K_{i-1} < X < K_i$



Alberi di ricerca

- La caratteristica più importante nella gestione di un albero di ricerca è **mantenerne il bilanciamento** in modo che:
  - i nodi siano distribuiti uniformemente e la profondità dell'albero sia minimizzata
  - rendere uniforme la velocità di ricerca in modo che il tempo medio per trovare una qualsiasi chiave sia lo stesso

Albero di ricerca di ordine p=3