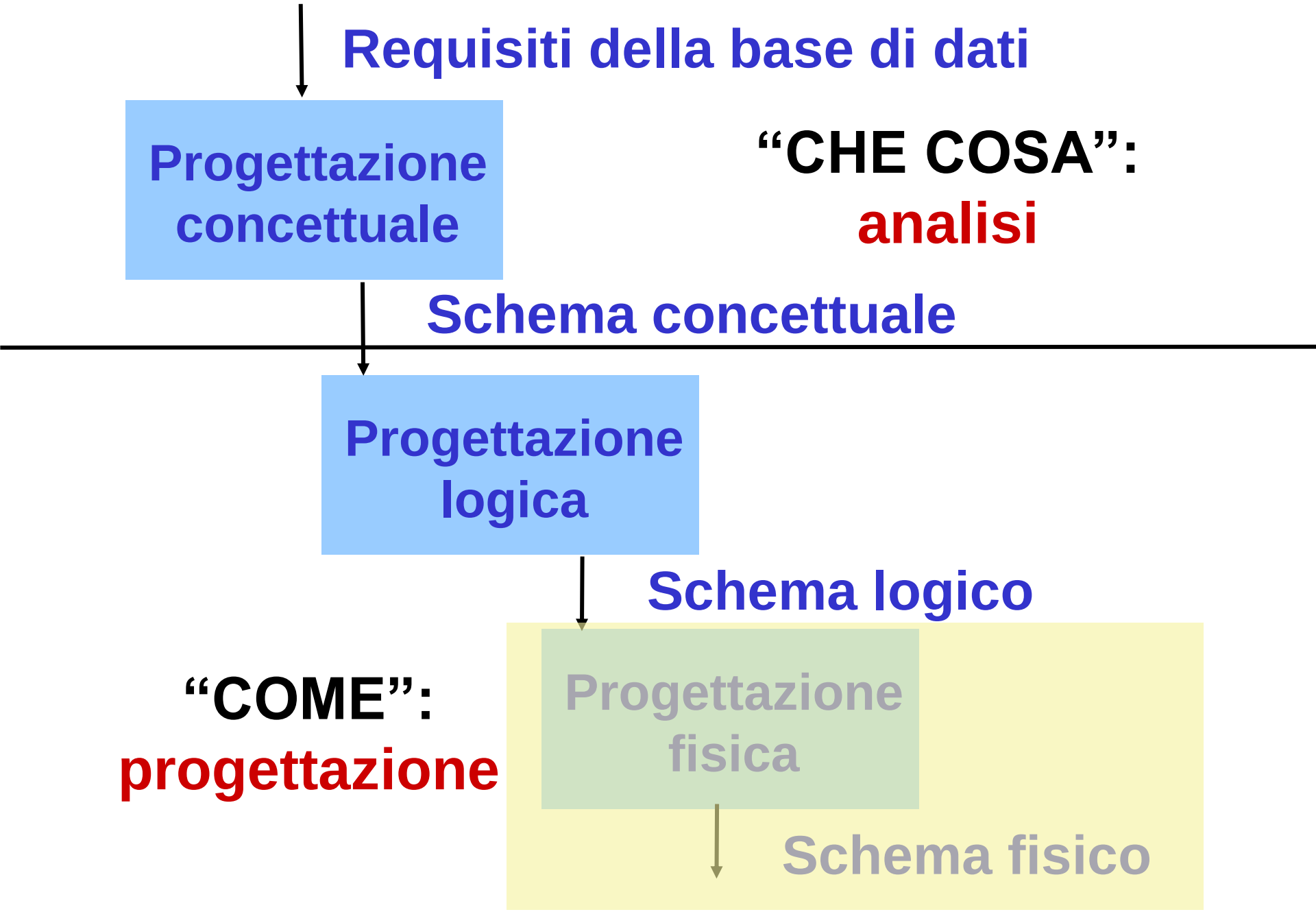
Laurea in Informatica
A.A. 2021-2022

Corso "Base di Dati"

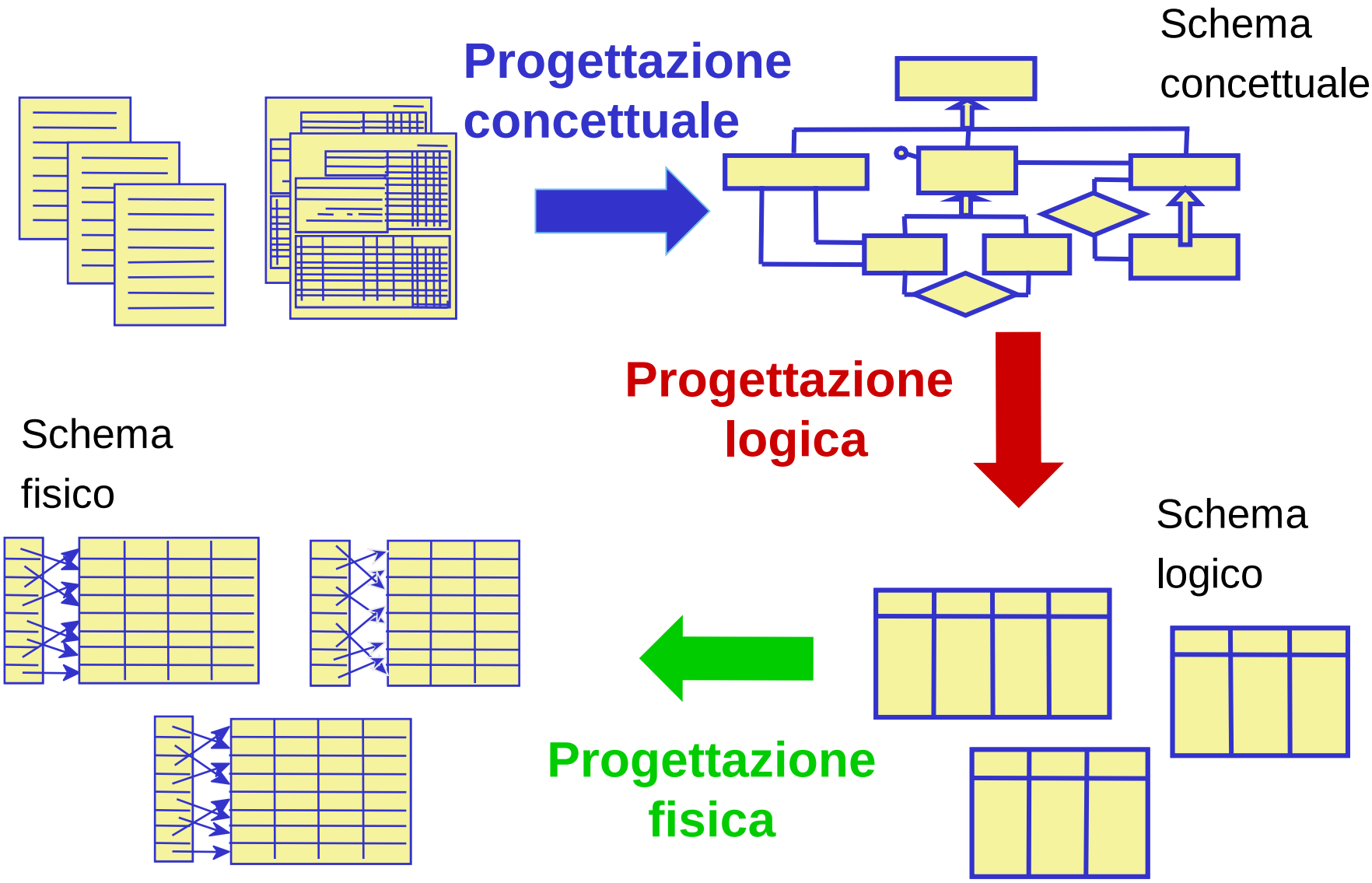
Creazione e Gestione degli Indici

Prof. Massimiliano de Leoni

Università degli Studi di Padova

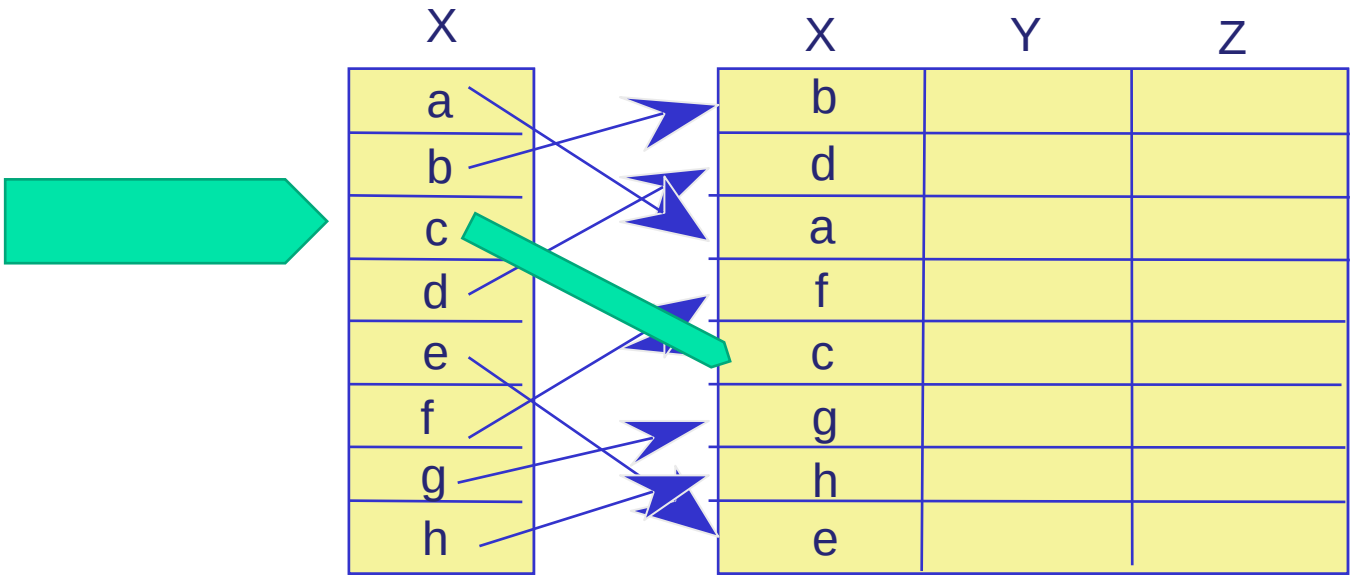

Requisiti della base di dati
Progettazione concettuale
“CHE COSA”:
analisi
Schema concettuale
Progettazione logica
Schema logico
“COME”:
progettazione
Progettazione fisica
Schema fisico

I prodotti delle varie fasi



Concetti di Base / 1

- Gli indici favoriscono l'accesso in base al valore di uno o più attributi
- L'obiettivo degli indici nel DBMS è di velocizzare l'accesso a 0+ tuple del database.
- Si definisce una "chiave di ricerca", 1+ attributi della relazione



Concetti di Base / 2

- Un indice è un file (o una struttura dati) che consiste di un insieme di record (chiamate “index entries”) della forma

- La dimensione dell’indice è molto più piccolo di quella della tabella-relazione associate

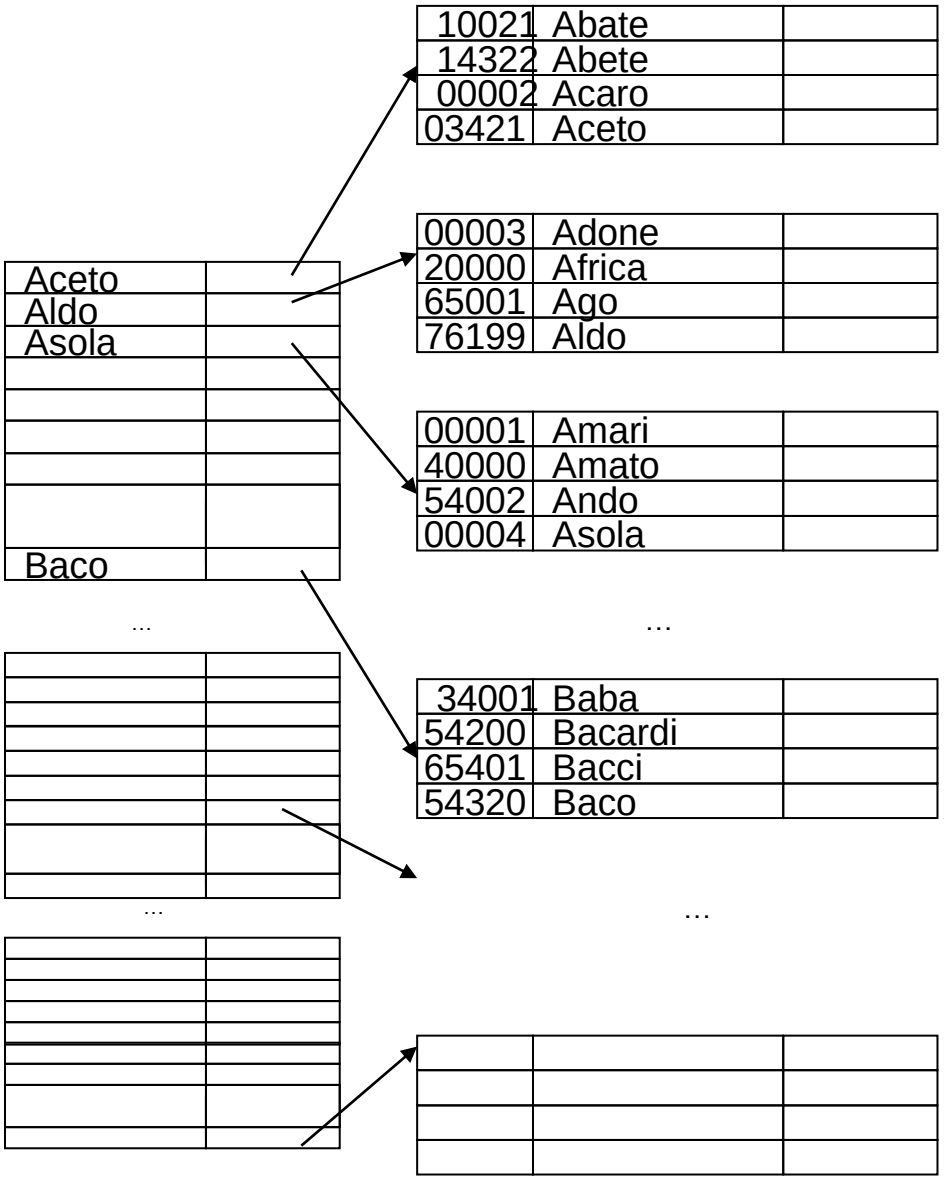
- Pro degli indici: Indici offrono benefici sostanziali quando si cercano dei record
- Contro: Aggiornare una relazione fornisce un “overhead” perchè ogni indice della relazione deve essere aggiornato

Metriche di Valutazione degli Indici

1. Tempo di Accesso (via queries):
   - ai record con uno specifico valore per un attributo,
   - ai record con valori all’interno di uno specific intervallo
2. Tempo di Inserimento
3. Tempo di Cancellazione
4. Occupazione di Spazio

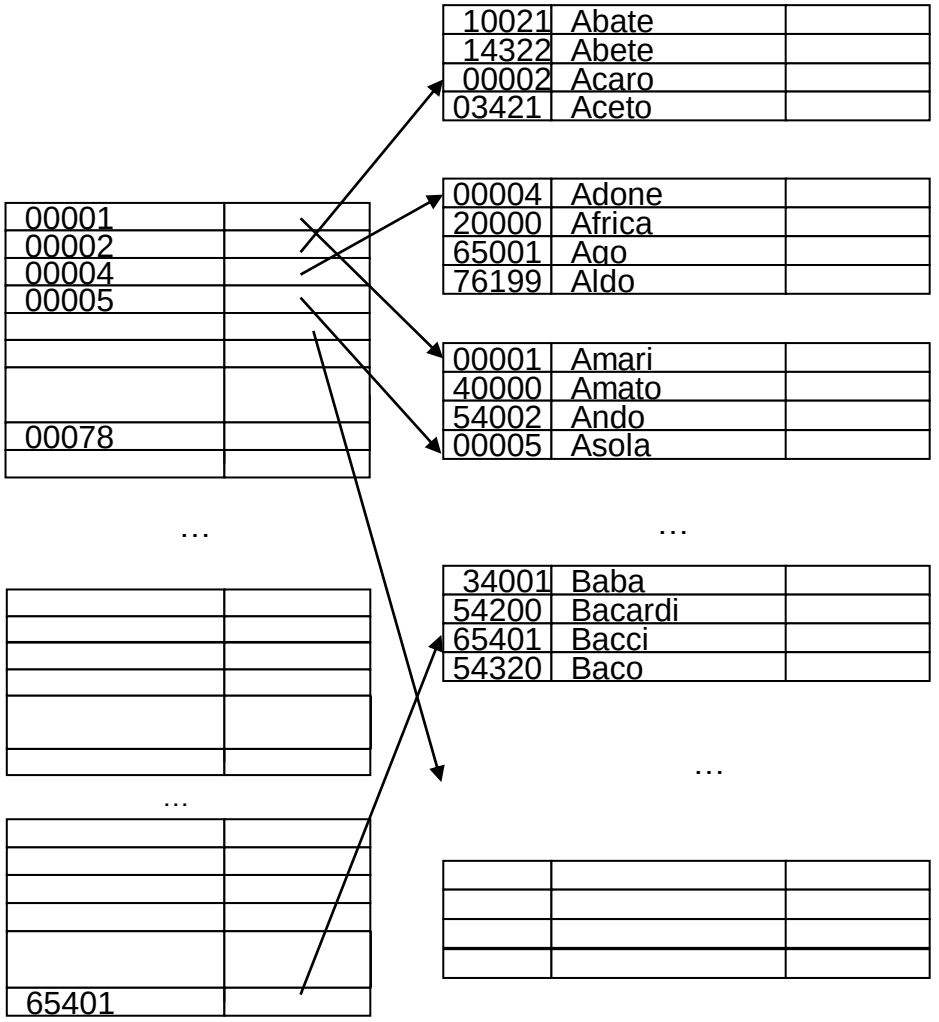
Indice Primario

- Le tuple sono mantenute ordinate per i valori delle chiavi primarie.
- Tipicamente chiave di ricerca = chiave primaria
  - ma non è necessario
- Al più 1 indice primario



Indice Secondario

- Le tuple sono mantenute in ordine diverso rispetto a quello della chiave di ricerca
- Possibili 0+ indici secondari

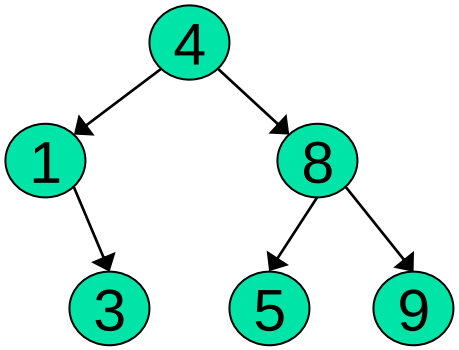


Osservazioni su Indici Primari e Secondari

- Iterare è efficiente su un indice primario:
  I blocchi dei file del database vengono letti in sequenza
  - *Punto negativo:* gli inserimenti posso essere costosi poiché occorre dover “spostare” alcuni elementi.
- Iterare è costoso su un indice secondario
  - Occorre saltare potenzialmente avanti ed indietro nel file del database
  - Quindi: i blocchi del file degli indice viene letto più volte

Per fare un indice:
Albero binario di ricerca

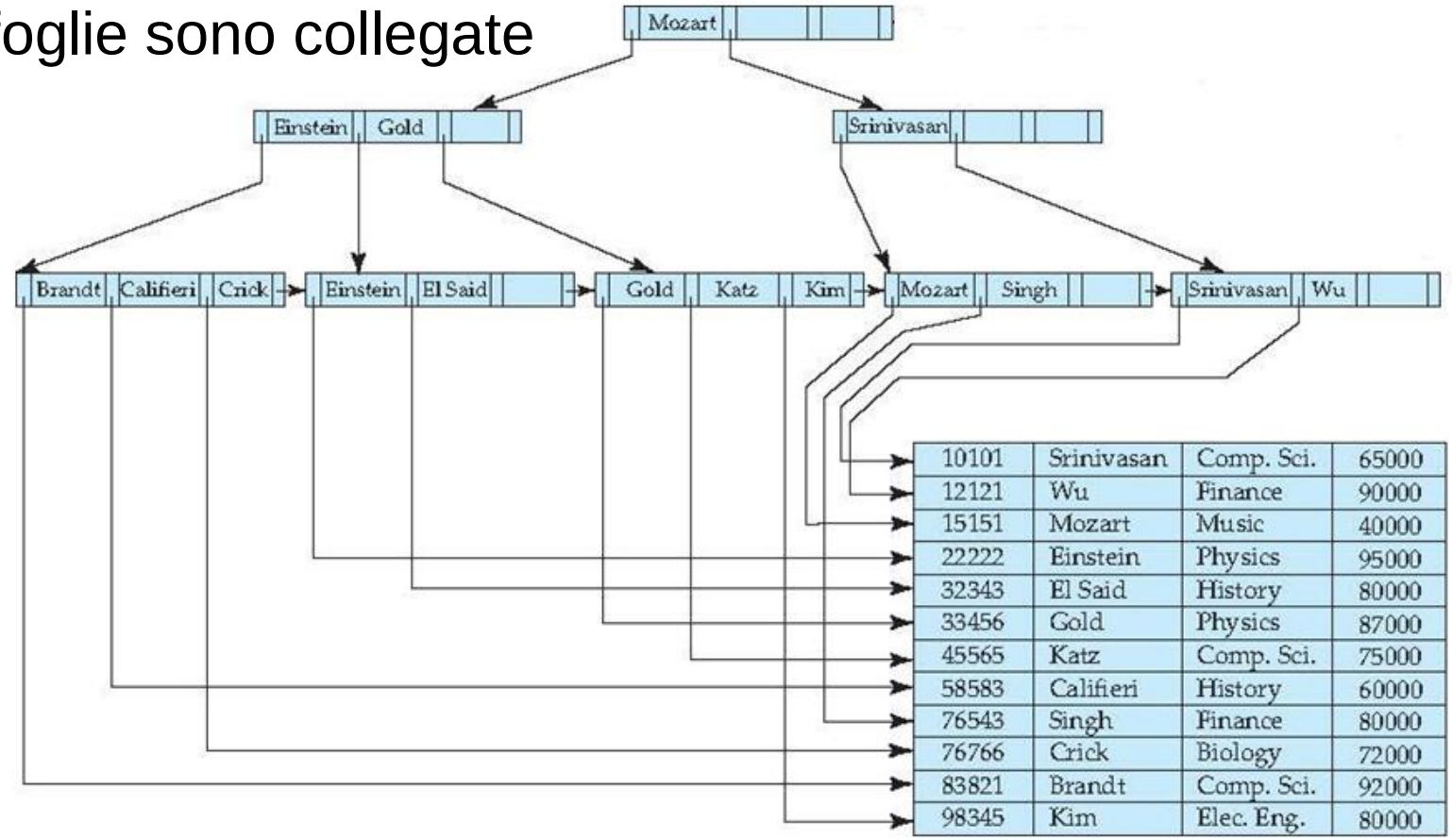
- Per ogni nodo:
  - Il sottoalbero sinistro contiene solo etichette minori di quella del nodo
  - Il sottoalbero destro etichette maggiori



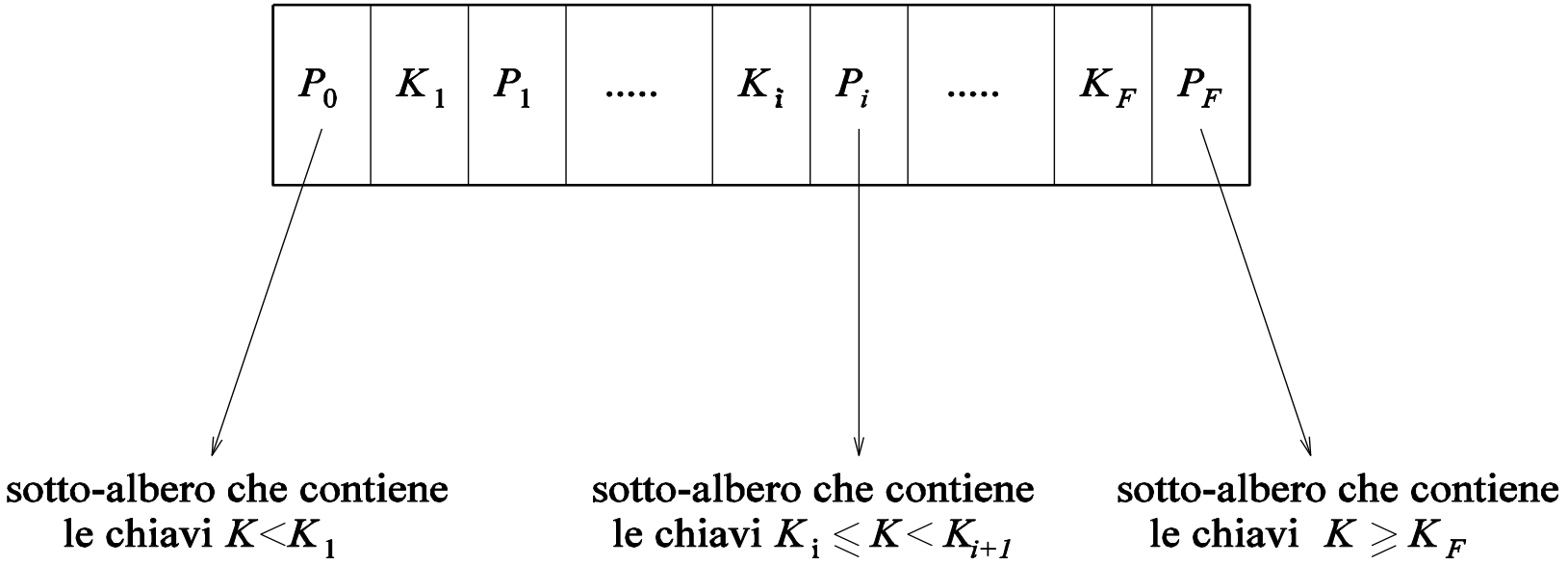
- «Trovare» un nodo costa quanto la sua profondità:
  - Per minizzare il costo medio, bilanciare l'albero: ogni nodo è in uno dei due casi (salvo impossibilità)
    - 2 figli
    - Nessun figlio
  - Se l'albero è bilanciato e contiene **n** nodi, la profondità è ca. $\mathbf{\log_2 n}$

B+ Tree

- Alberi di ricerca bilanciati
- I valori (=puntatore ai dati) sono solo nei nodi "foglia"
- I nodi hanno più di un valore
- Le foglie sono collegate



Struttura di un nodo non foglia di un B+ Tree

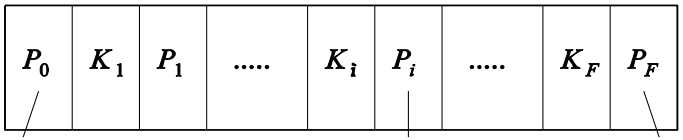


I valori K_i delle chiavi di ricerca sono ordinate:

$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

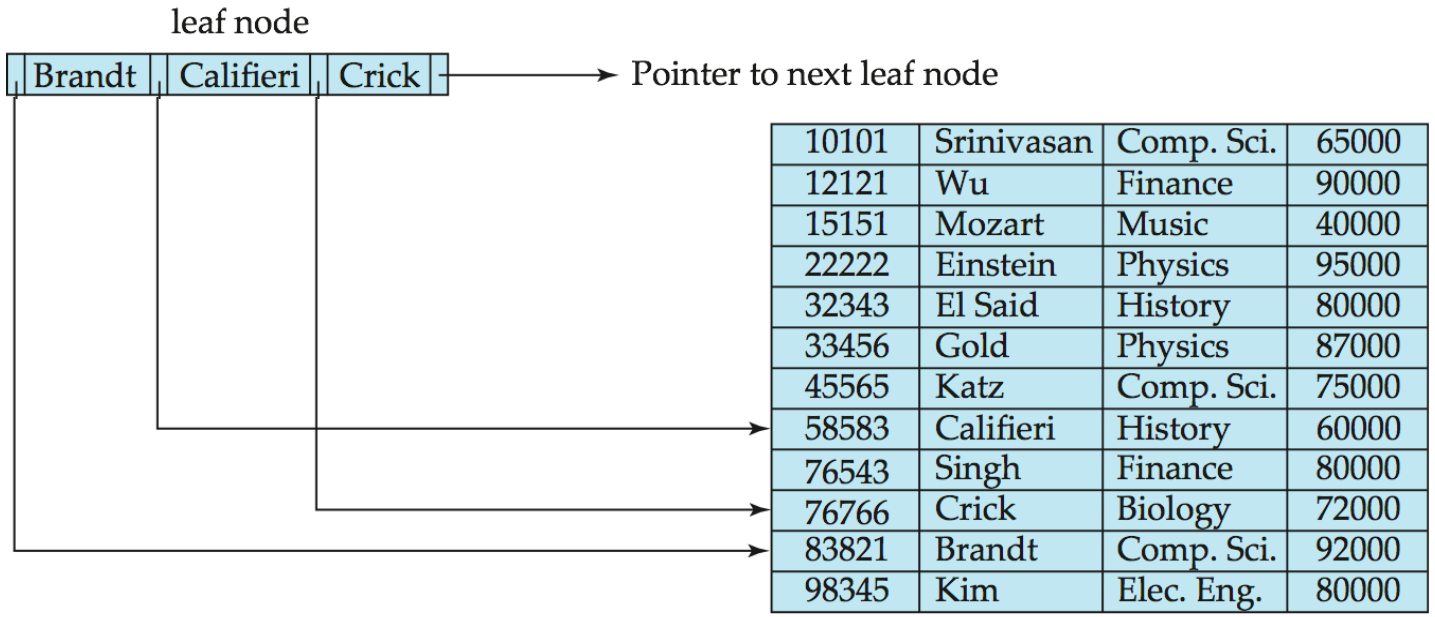
(Assumiamo che non ci siano duplicati)

Nodi foglia dei B+Trees / 1



Assumendo F valori per foglia:

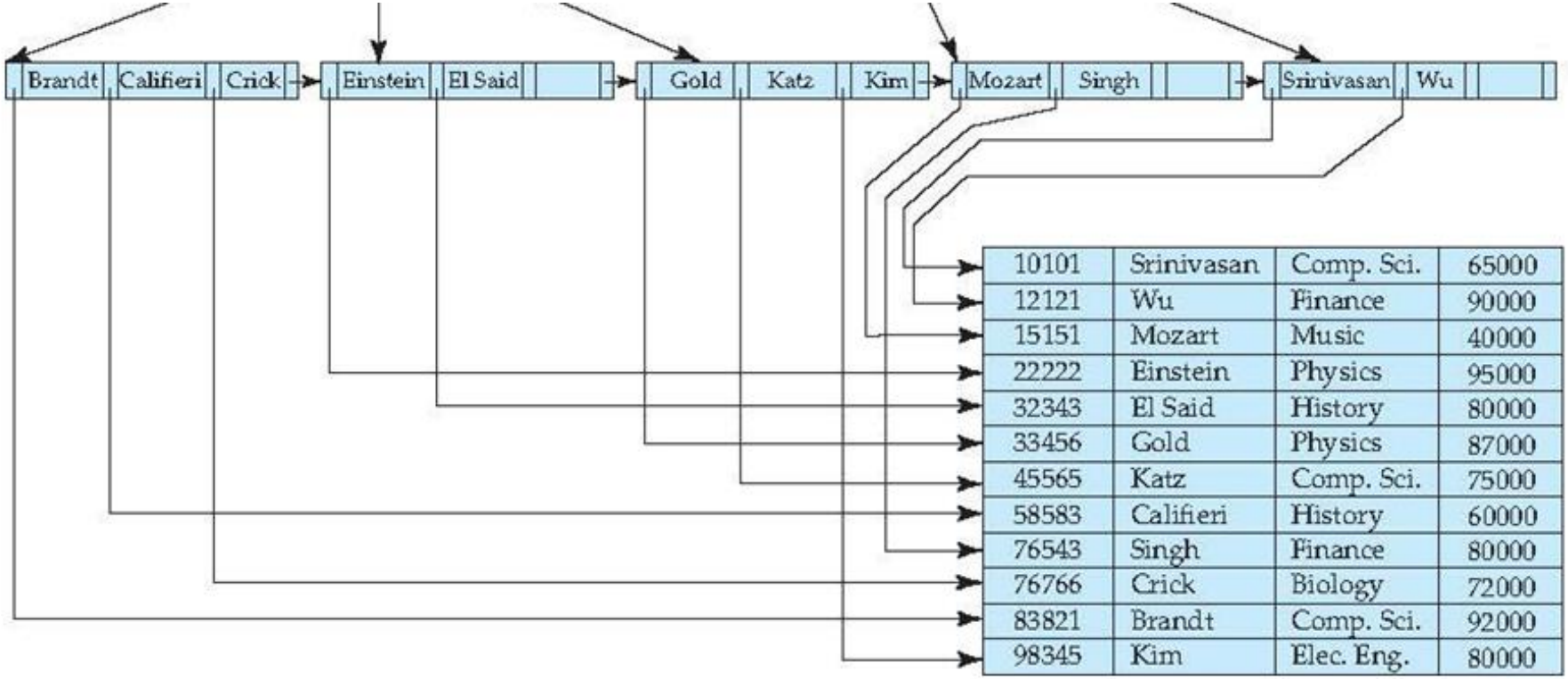
- $\forall$ i < F–1, il puntatore P_i si riferisce alla tuple con valore K_i per la chiave di ricerca
- P_F punta alla successiva foglia nell'ordine della chiave di ricerca



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Nodi foglia dei B+Trees / 2

Se la foglia F_i e seguita dalla foglia F_j tutti i valori nella foglia F_i sono inferiori a tutti quelli della foglia F_j

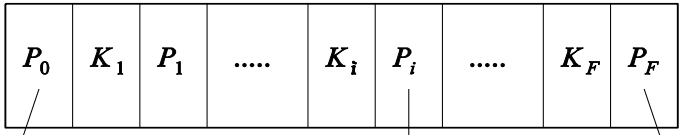


B+ Tree

- Vantaggi di file ad indici B+-tree:
  - Si riorganizzano con cambiamenti “locali” in caso di inserimenti e cancellazioni.
  - La tabella non deve essere riorganizzata per le garantire le performance quando tuple vengono aggiunte o cancellate
- Svantaggi:
  - Extra occupazione di spazio, tempi extra per inserire e rimuovere
  - Le query con “=” vengono solo parzialmente ottimizzate

Ricerca di un record (assumendo senza duplicati)

```
V è il valore della chiave di ricerca di cui trovare il record
C=Puntatore a Radice del B+Tree
while (C non è un foglia)
{
     /*C.Pj indica il campo Pj del blocco puntato da C*/
     Prendi j tale che C.Pj<>null e C.Pj+1=null
     /*Se valore dell'ultima chiave non nulla è maggiore di V, prendi l'ultimo puntatore*/
     if (V > C.Kj-1) C=Pj
     else {
        Prendi i tale che V ≤ C.Ki e (V > C.Ki+1 o C.Pi+1=null)
        if (V = C.Ki) C=Pi else C = Pi-1
     }
}
```



Osservazioni

1. Se ci sono N valori nell'albero B+Tree,
   la profondità ≤ $\lceil \log_{\lceil F/2 \rceil}(N) \rceil$ dove F è il numero massimo di valori in un nodo.
2. Un nodo è della stessa dimensione di un blocco/cluster del file system, tipicamente 4 kilobytes
   ➔ Se ca. 40 bytes per index entry, F~100 index entry per nodo
3. Se N=1.000.000 valori and F = 100
   ➔ $\log_{50}(1{,}000{,}000) = 4$ nodi

Inserimento in B+Trees

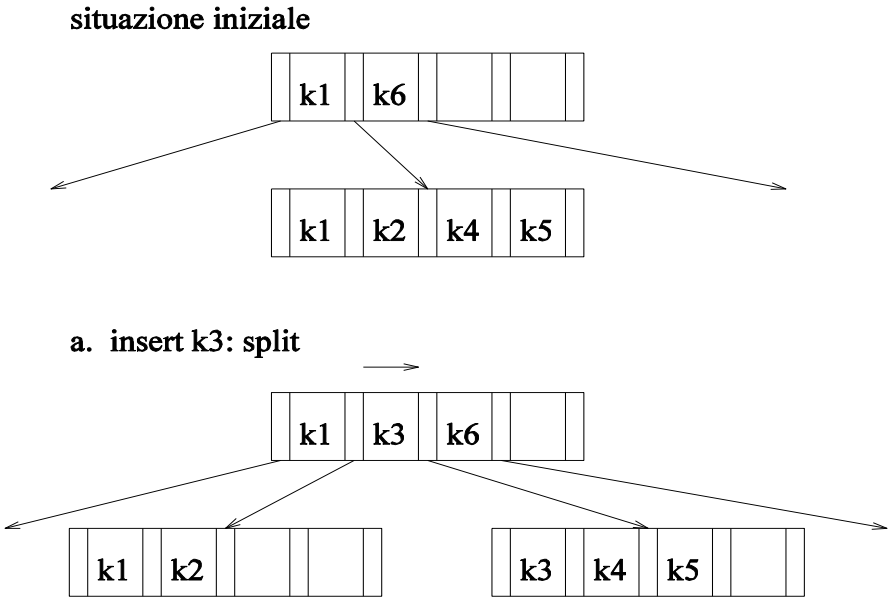
1. Trova la foglia dove la chiave di ricerca dovrebbe apparire
2. Se il valore è presente, aggiorna il puntatore.
3. Se il valore non è presente, allora
   A. Se c’è spazio, aggiungi la coppia (chiave, puntatore al record)
   B. Se non c’è spazio, occorre creare un nodo

Inserimento in B+Trees: Creare un nuovo nodo

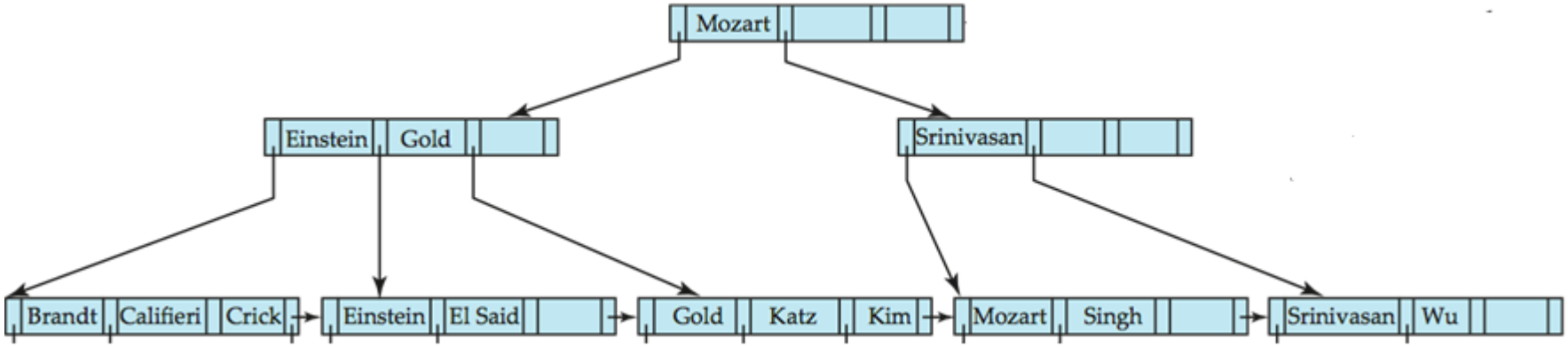
Si vuole aggiungere una coppia (k,p) dove p è puntatore alla tupla di dato.

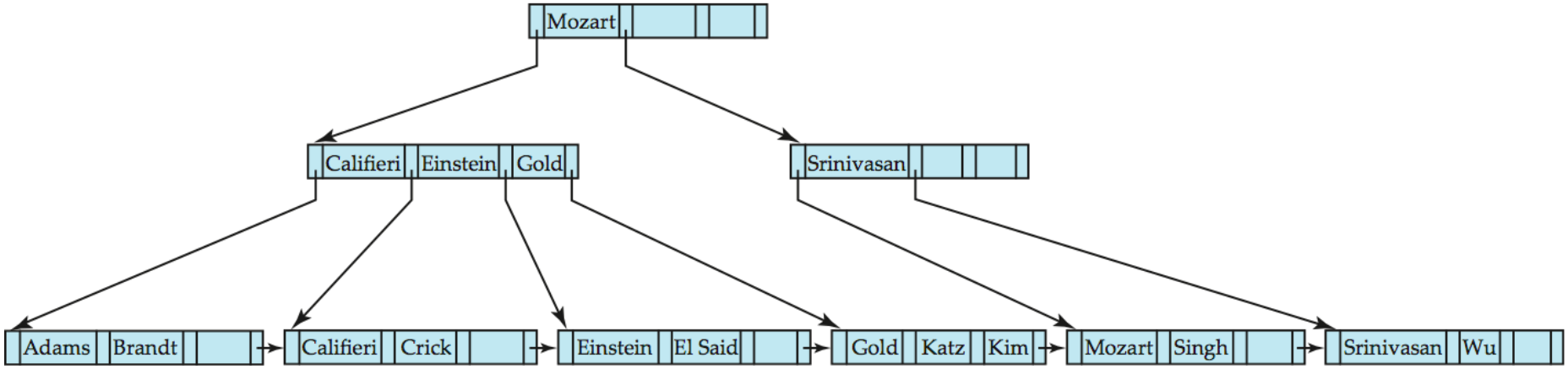
Dividere la foglia con F coppie (valore chiave ricerca, puntatore):

A. Tenere $\lceil F/2 \rceil$ nel nodo originale, e il resto in un nuovo nodo incluso la coppia (k,p).
B. Modificare il nodo padre come da figura.
C. Se il nodo padre è pieno, ripetere la procedura per il nodo padre



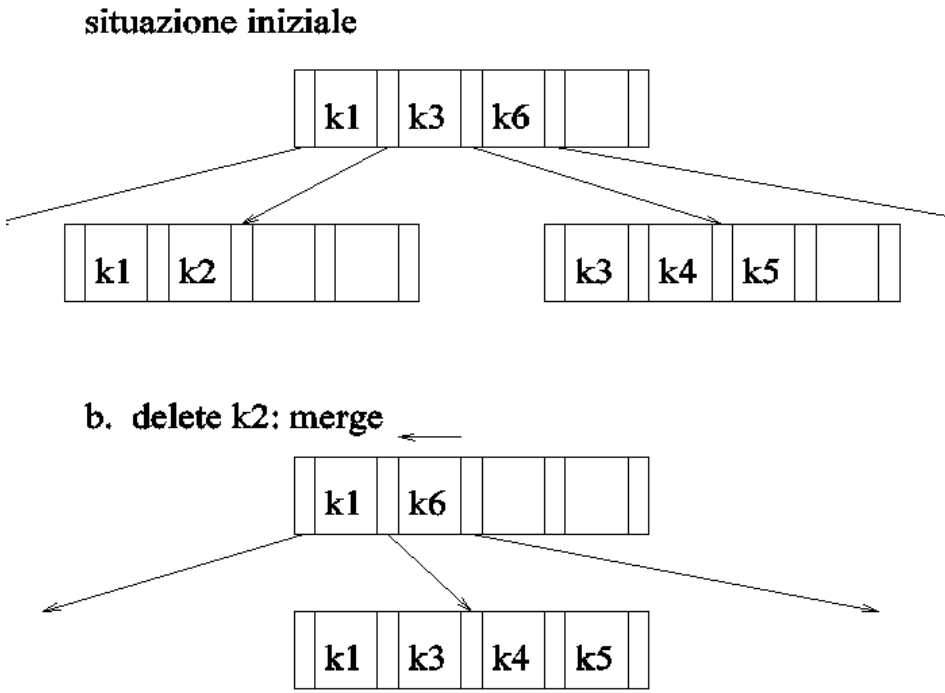
Esempio: B+Trees prima e dopo l'inserimento di "Adams"





Cancellazione da un B+Tree

- Rimuovere la coppia (valore chiave ricerca, puntatore) dalla foglia.
- Se la foglia F ha meno di metà coppie utilizzate:
  - Se una foglia adiacente G ha sufficiente spazio per ospitare le entry di F:
    - Inserisci tutte le entry di F in G
    - Rimuovi la foglia G
    - Rimuovi la coppia (k,p) dal nodo padre dove p è il puntatore a G
    - Aggiorna i puntatori nel nodo padre
  - Se nessuna foglia adiacente ha sufficiente spazio
    - Ridistruisci le coppie con le foglie adiacenti
    - Aggiorna i puntatori nel nodo padre

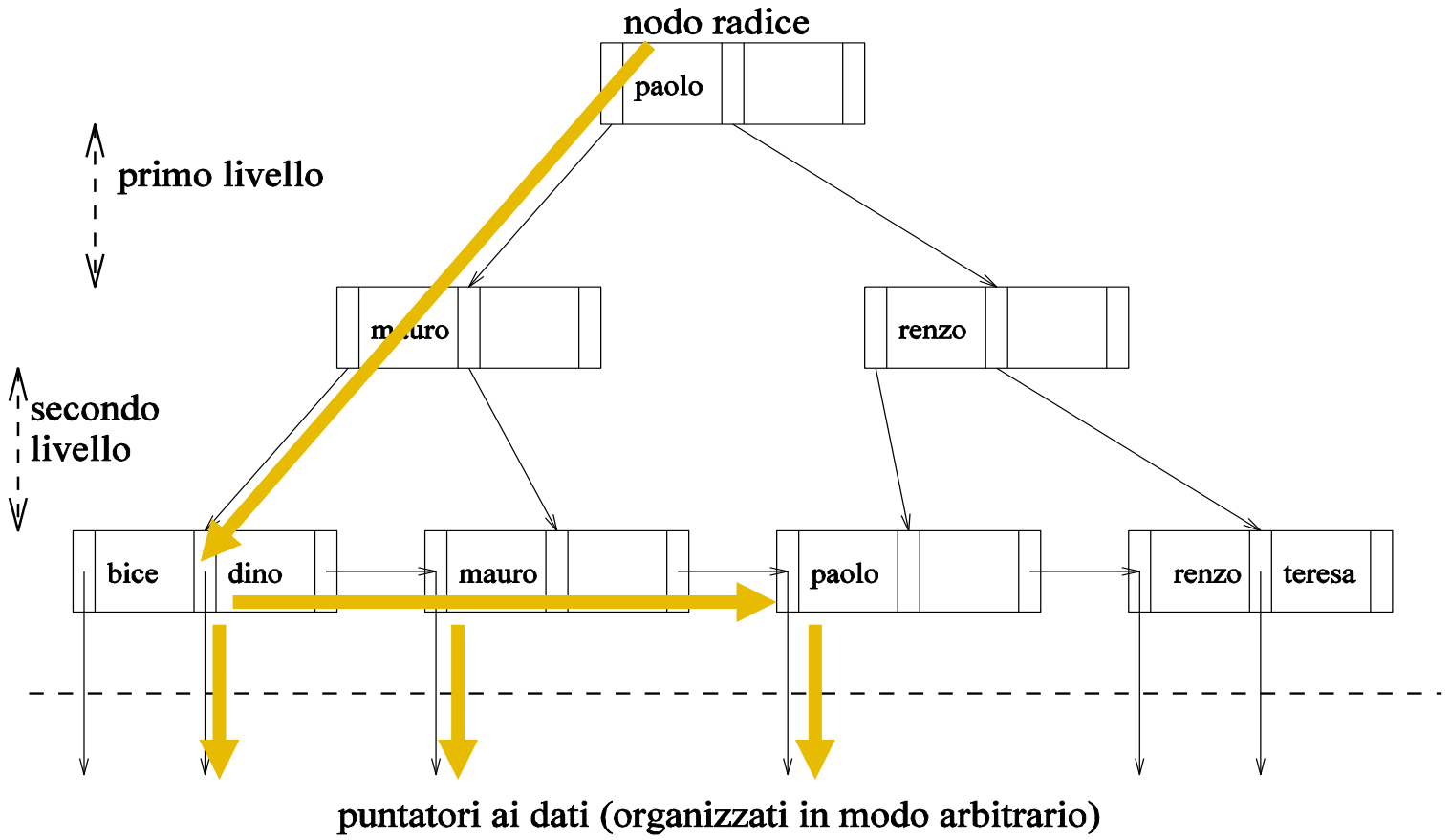


Vantaggio importante dei B+Tree: Query su intervalli

Supponiamo di voler fare

```
select * from Persone where Nome >= 'Dino' and Nome <='Paolo'
```

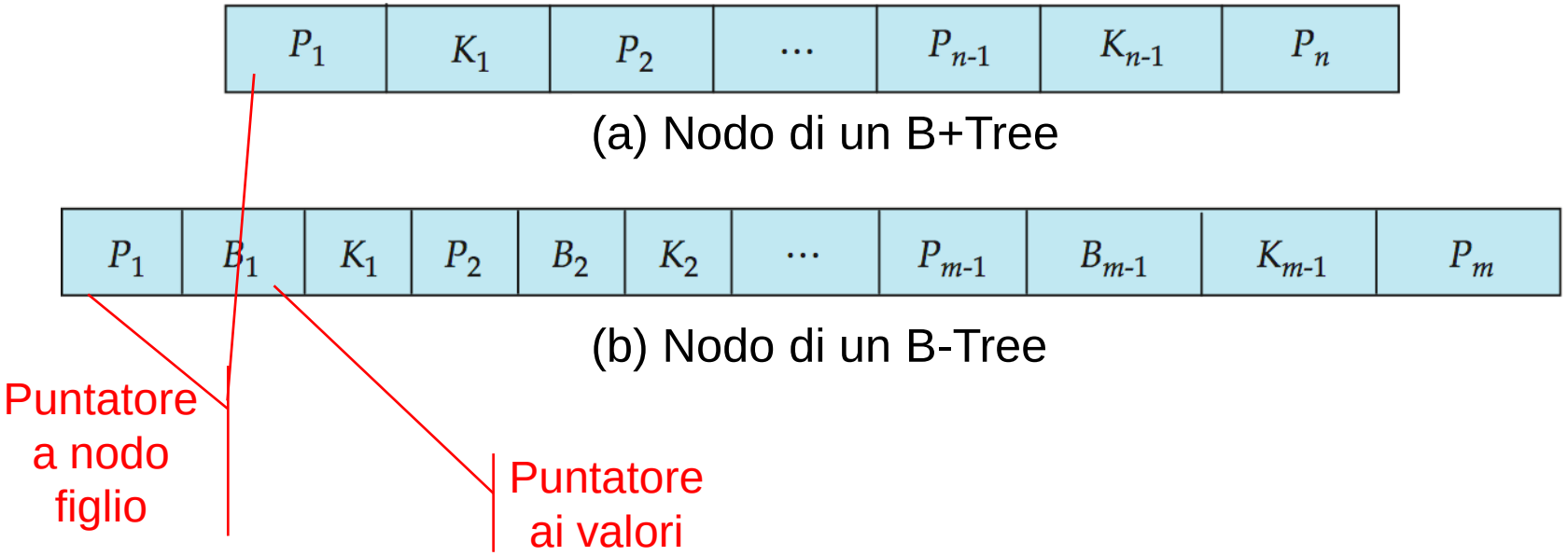
L'indice verrà navigato:



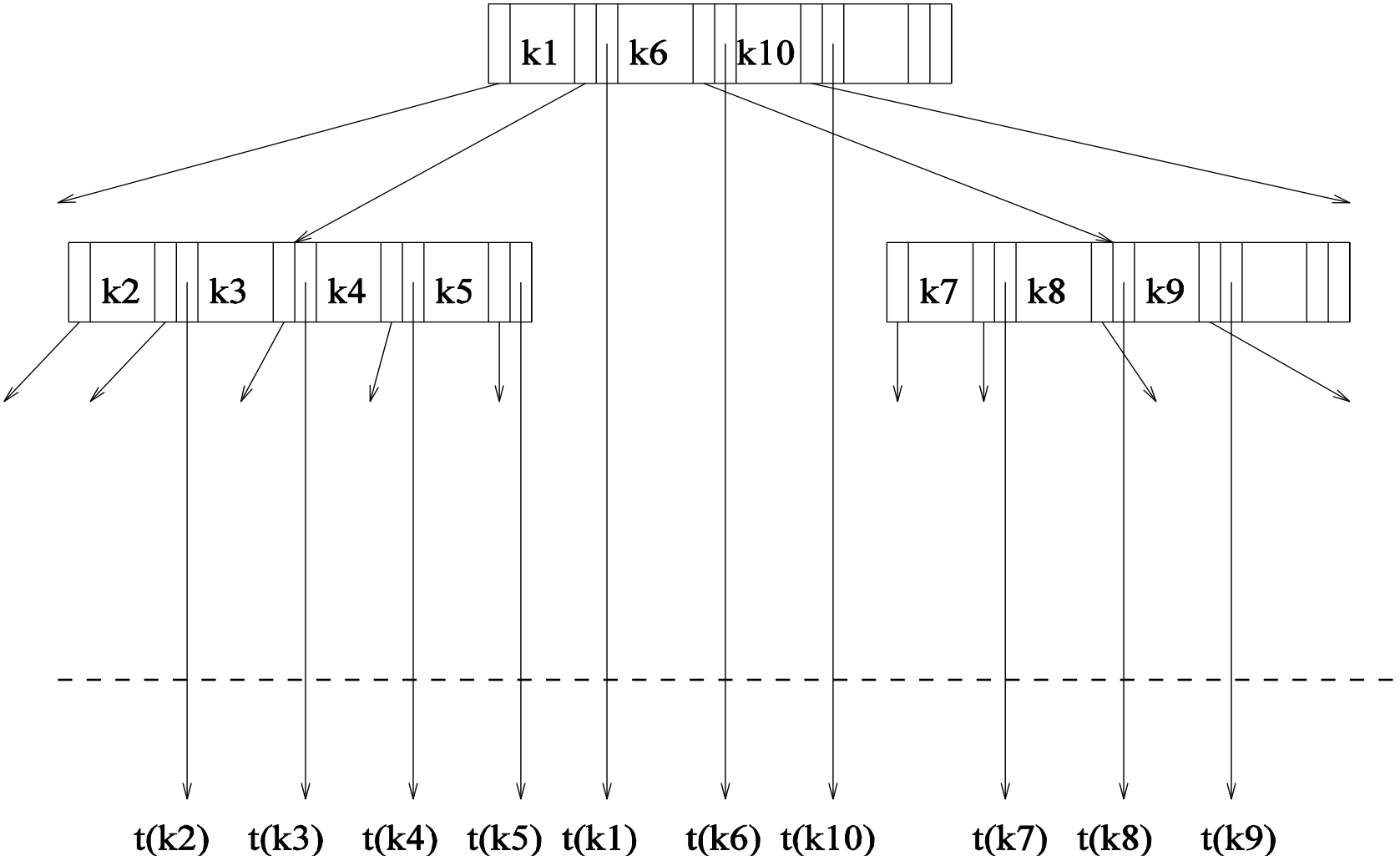
Indici B-Tree

Simili a B+tree, ma:

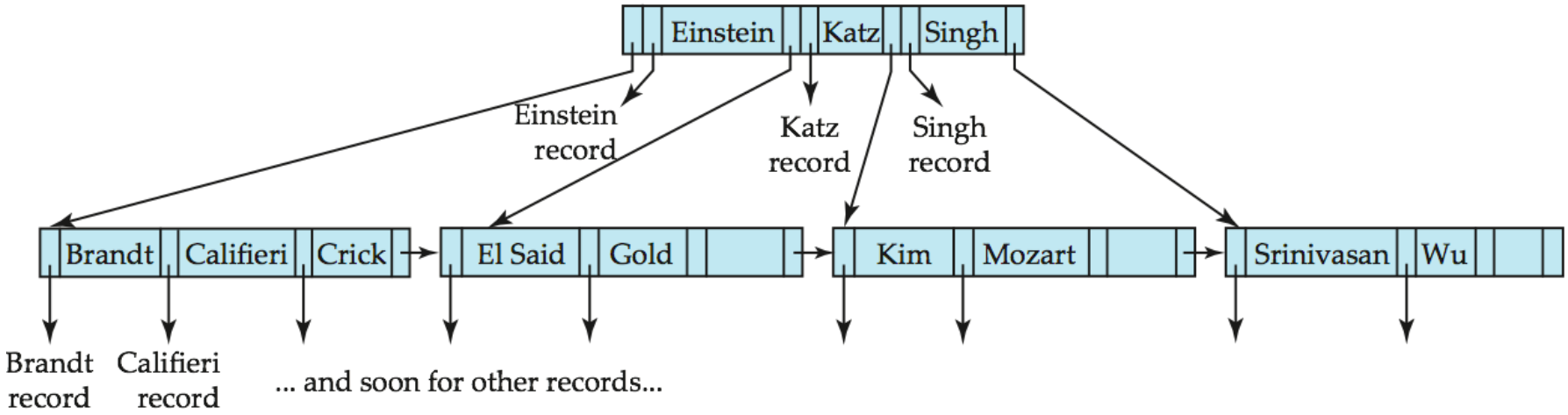
- Anche i nodi interni contengono valori
- Le foglie non sono connesse.

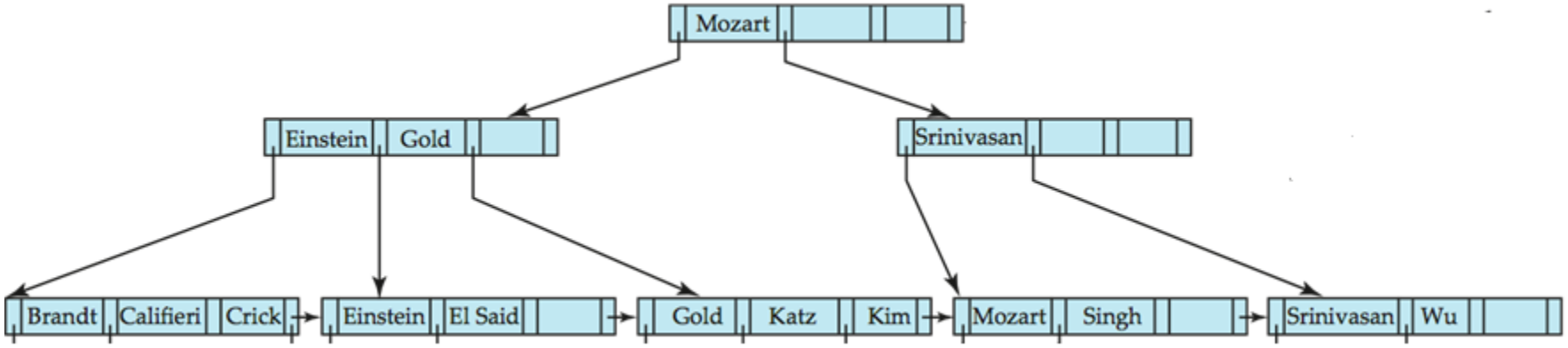


Un Esempio di B-Tree



Esempi: B-tree (sopra) e B+tree (sotto) sugli stessi dati





B-Tree vs B+Tree

- Vantaggi di un B-Tree:
  - In genere, tende ad usare meno nodi di un B+Tree
  - È possibile trovare una coppia chiave-valore anche prima di raggiungere una foglia (in verità poche volte)
- Svantaggi di un B-Tree:
  - Nessun vantaggio sostanziale nelle query per intervallo
  - Inserimenti e cancellazioni sono più complessi rispetto ad un B+Tree
  - La manutenzione di B-Tree è più complesso

Ottimizzazione di query con selezione su molteplici attributi

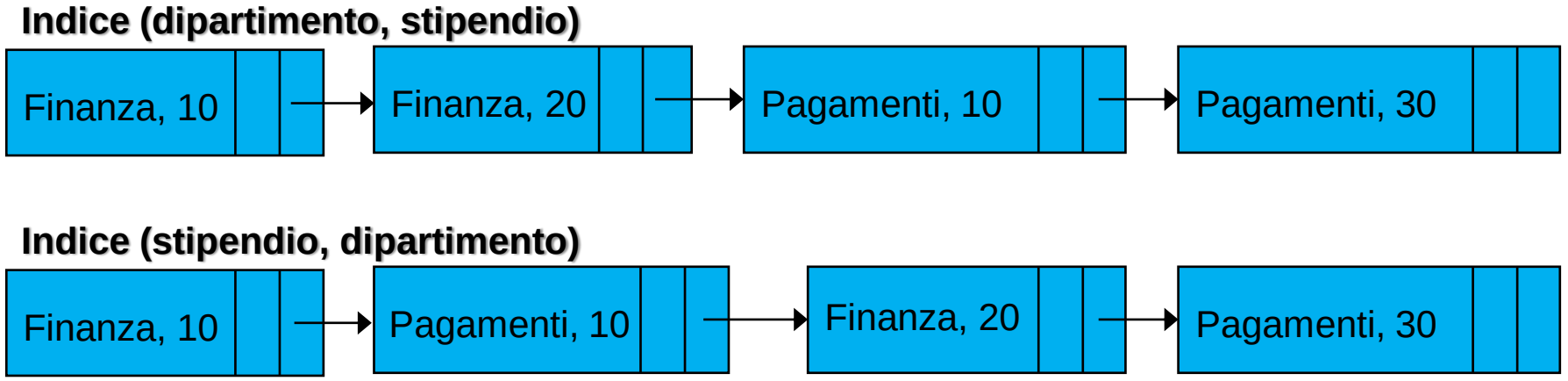
- Esempio:

```
select ID
from Impiegato
where dipartimento = "Finanza" and stipendio = 80000
```

- Possible strategie se ci sono indici su attributi **singoli**:
  1. Usa indice su dipartimento per trovare gli impiegati del Dipartimento "Finanza"; scorri tutti, estraendo quelli con stipendio di € 80000
  2. Usa indice su stipendio per trovare gli impiegati con stipendio di € 80000; scorri tutti, estraendo quelli del Dipartimento Finanza
  3. Usa sia l'indice su Dipartimento che quello su stipendio. Prendi l'intersezione di entrambi

La 4ª alternativa: Chiavi di Ricerca su Due attributi

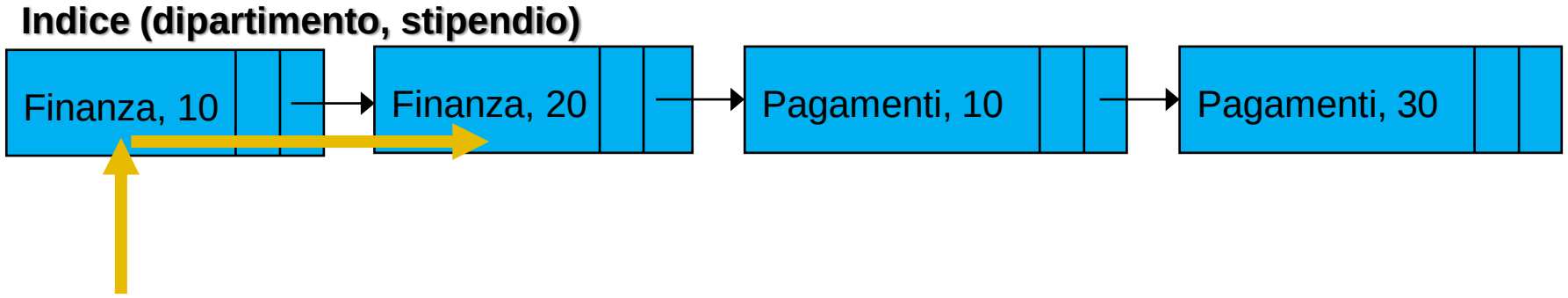
- Più efficiente
- L'ordine delle chiavi di ricerca contano!
  - **(dipartimento,stipendo)** è diverso da **(stipendio,dipartimento)**
  - Ordine lessicografico: (a1, a2) < (b1, b2) se:
    - a1 < b1, o
    - a1=b1 e a2 < b2



Indici su Attributi Multipli / 1

Un indice su una chiave di ricerca multipla `(dipartimento, stipendio)` è efficiente nei casi:

```
where dipartmento = 'Finanza' and stipendio=10
where dipartmento = 'Finanza' and stipendio<=20
```

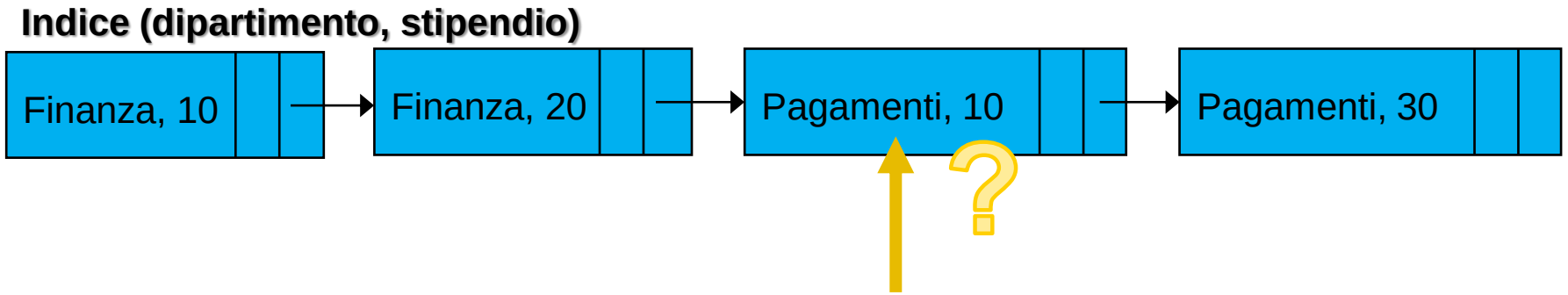


Indici su Attributi Multipli / 2

Un indice su una chiave di ricerca multipla `(dipartimento, stipendio)` NON è efficiente nel caso:

```
where dipartmento > 'Finanza' and stipendio=80
```

- È possibile estrarre tutti gli impiegati di dipartimenti "maggiori di" finanza efficientemente
- Non è possibile estrarre efficientemente quelli con stipendio di 80



Indice Hash / 1

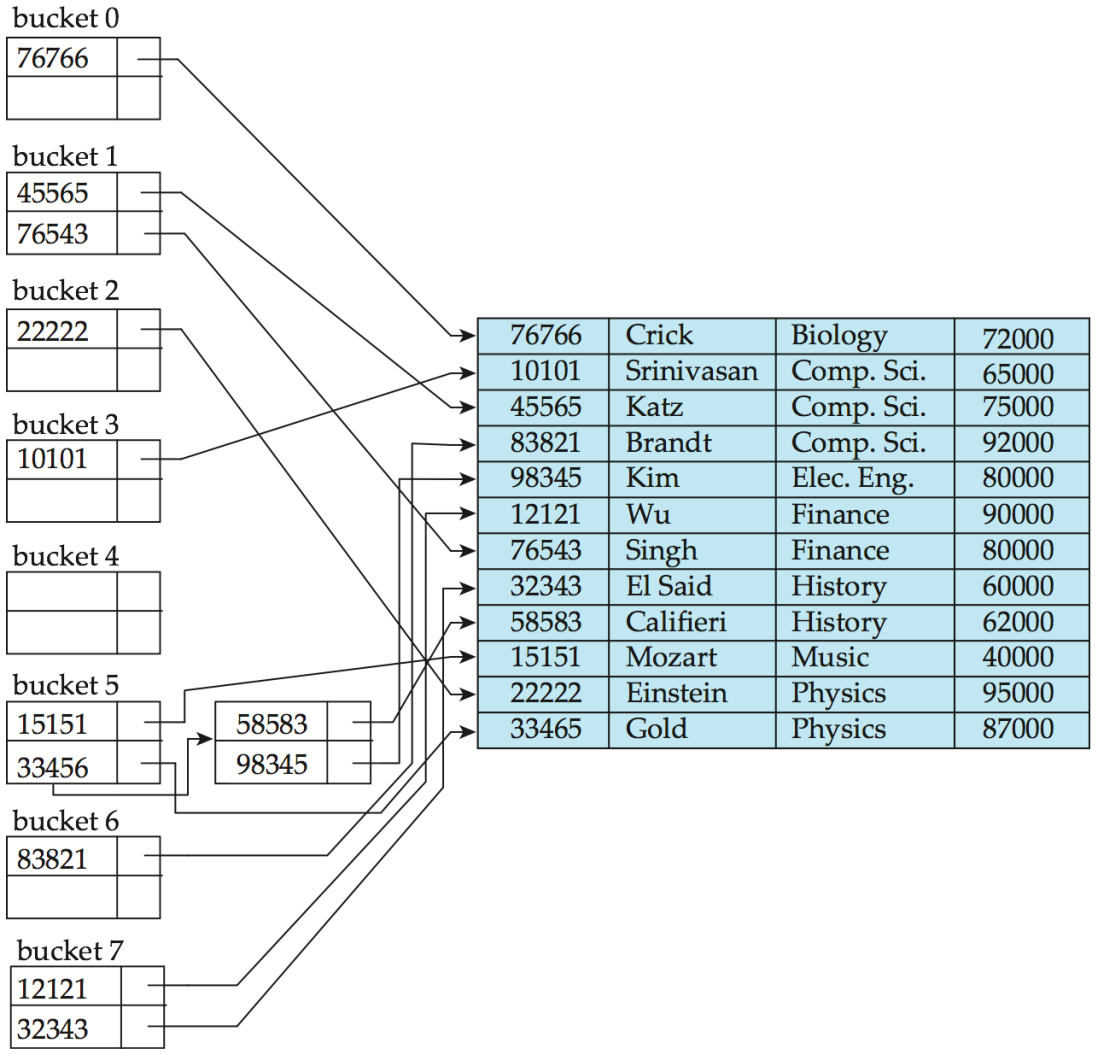
- Si basa sul principio della **funzione di hash**.

- Una funzione $\boldsymbol{f: string \rightarrow string}$ è di hash se
  - fissata un valore $\mathbf{n} \in \mathbb{N}$,
  - per ogni stringa $\boldsymbol{s}$ di lunghezza arbitraria
  - $\boldsymbol{f(s)}$ restituisce una stringa di stringa di lunghezza $\mathbf{n}$
- Un caso particolare di funzione hash è tale che restituisce una stringa esadecimale (cioè un numero).

- La funzione non è necessariamente invertibile, quindi non è vero$\boldsymbol{f^{-1}(f(s)) = s}$

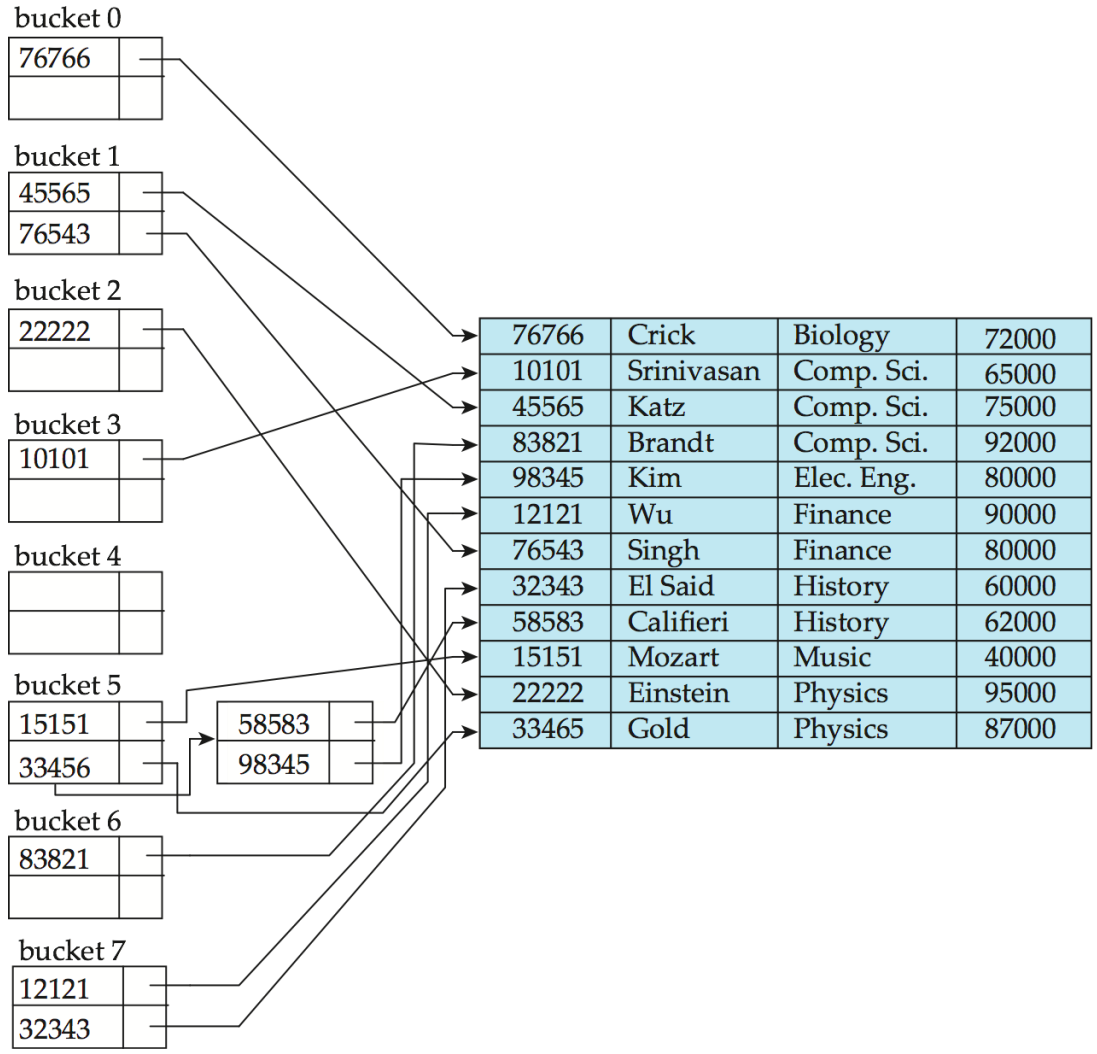
Indici Hash / 2

- Un indice hash organizza le chiavi di ricerca in the file “hash”
- Basato sui concetti di “bucket”, insieme di record (valore di chiave, puntatore).
- Dato un valore K della chiave di ricerca,
- Una funzione hash **h** associa ogni valore K a esattamente un bucket.



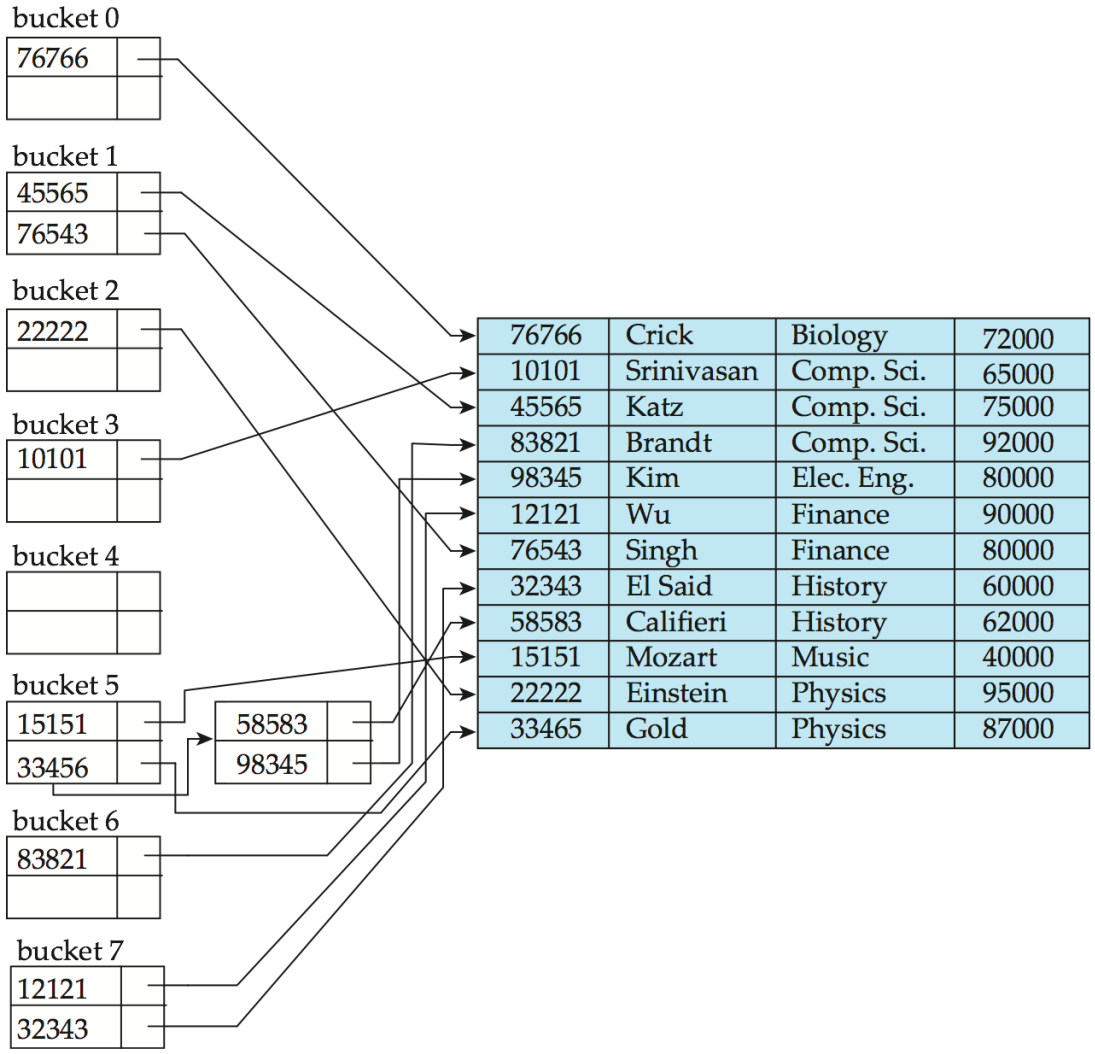
Indici Hash / 3

- Hash function è usato per trovare I record di accesso, inserimento e cancellazione
- Dati due valore K,K', è possibile che **h(K)=h(K')**
  - Collisioni possibili
  - Occorre poi cercare il valore K di interesse iterando nel bucket

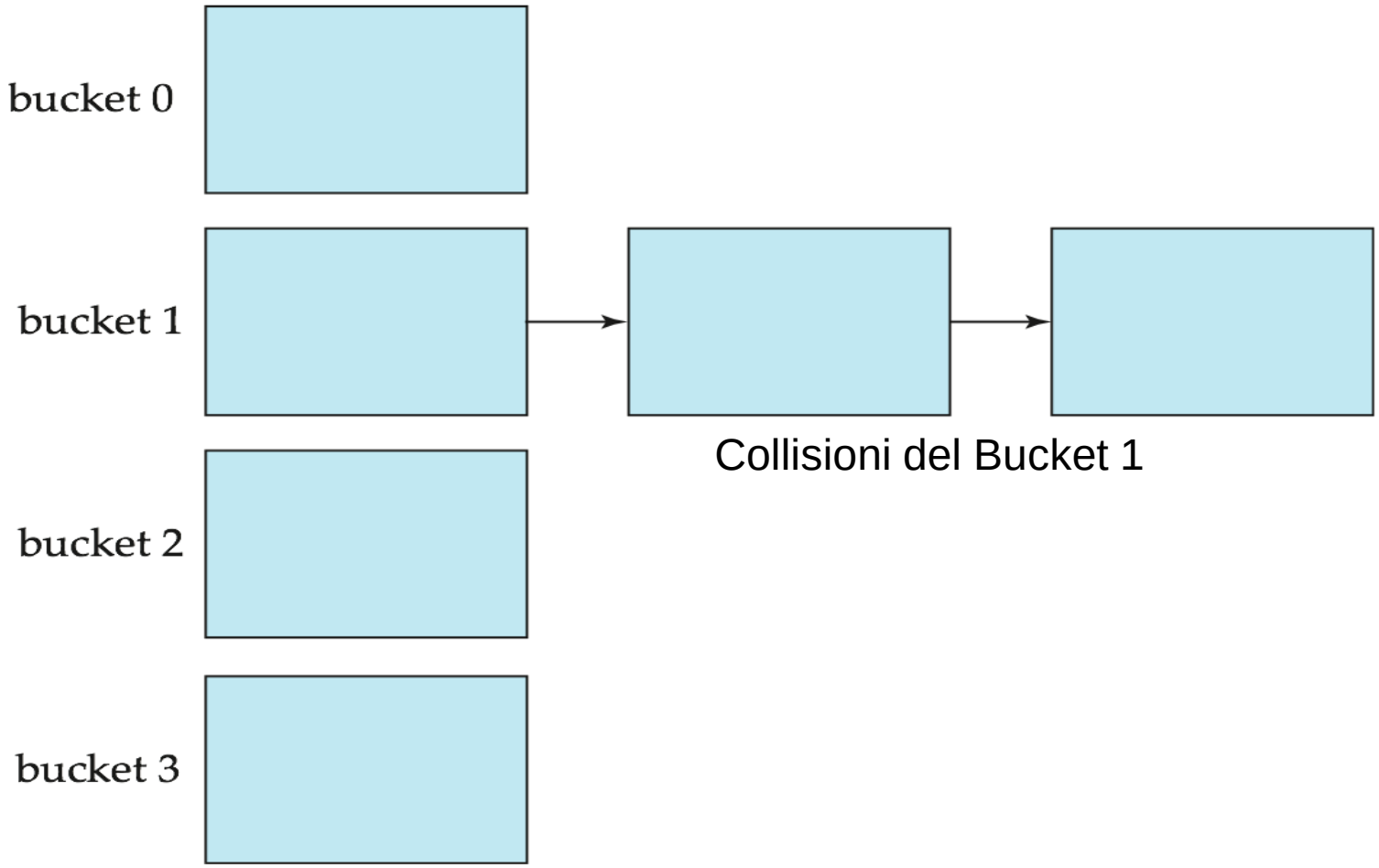


Indici Hash / 4

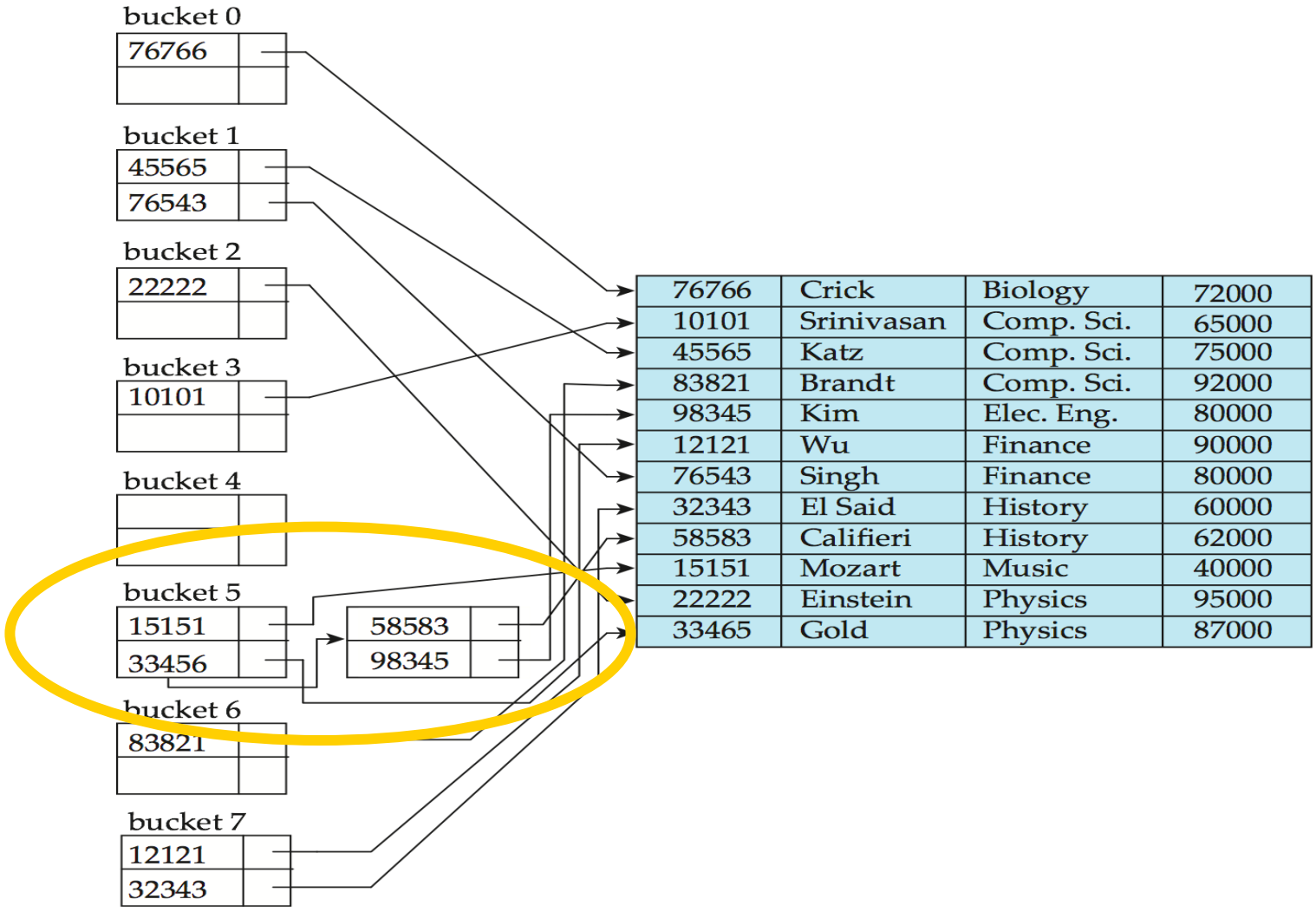
- Sono sempre secondari
- È possibile avere un indice primario separato
- È quindi possibile avere sia hash che B-tree



Gestione delle Collisioni



Gestione delle Collisioni



Vantaggi e Svantaggi di Indici Hash rispetto a B+-tree

Vantaggi

- Ricerca di un valore ha costo ca. 1
  - B+Tree è basso ma non 1
- Insertion ha costo ca. 1
- Cancellazione ha costo di ca. 1
  - Occorre inserire/rimuovere la entry dal bucket, creando/eliminando bucket di Collisione

Svantaggi

- Può essere solo usato per WHERE con atomi `variable=valore`
- Non può essere usato per velocizzare operazioni di ORDER BY.
  - Non è possible cercare la prossima entry in ordine

Definizione degli indici in SQL

- Non è standard, ma presente in forma simile nei vari DBMS
  - `create [unique] index` *IndexName* `on` *TableName*(*AttributeLis*t) [`using` *method*]
    - `unique` è opzionale: forza attributi degli indici ad avere valori unici
    - `using` *method* può essere `btree` o `hash`
    - *method* è opzionale: il default è `btree`.
    - altri *method* esistono ma non visti
  - `drop index` *IndexName*
- Esempio: `create index DipStipendio on Impiegati(Dipartimento,Stipendio)`
- Nota su Postgres:
  - Automaticamente viene creato un indice “unique” B+Tree sulla chiave primaria di ogni tabella
  - Indici hash possono essere creati solamente per indici su un attributo
  - La clausola `unique` è solo per i B+Tree

Riferimento

- Parte del contenuto di questa lezione è ritrovabile sul libro, nel Capitolo 11:
  - Sezione 11.4 (B+Tree)
  - Sezione 11.3.2 (Funzioni Hash)
- Per quanto non coperto sul libro, si faccia riferimento a questo set di slide

- Per indici per PostgreSQL, si veda la documentazione sul sito:
  - Capitolo 11.2 per una discussione sull'uso
  - Capitolo 63 per una discussione sull'implementazione (utile anche per capire i concetti)