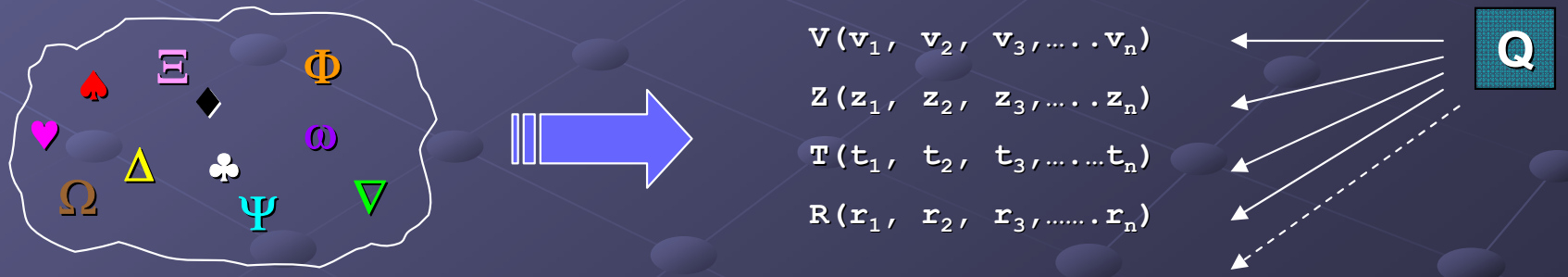
Strutture Dati Efficienti per la Ricerca della Similarità

Introduzione

- Alberi B e B+
- Clustering
- Alberi B+ Multidimensionali
- Alberi K-d
- Grid Files
- R Tree

Introduzione

- Per un insieme di oggetti multimediali, la fase di indicizzazione produce un insieme di vettori di caratteristiche (**feature vector**)
- Ciascun vettore V, a seconda della complessità dell'oggetto indicizzato, può contenere un **numero grande di componenti** (caratteristiche testo, audio, immagini, video,…)
- La **fase di Retrieving** è, pertanto, fondamentalmente caratterizzata da un gran numero di confronti di caratteristiche tra la query Q e le caratteristiche degli oggetti precedentemente memorizzate.
- **Non è quindi pensabile che questi confronti siano eseguiti in modo "lineare"**
- Le strategie proposte sono basate sulla **suddivisione dello spazio multidimensionale** delle caratteristiche in sottospazi.



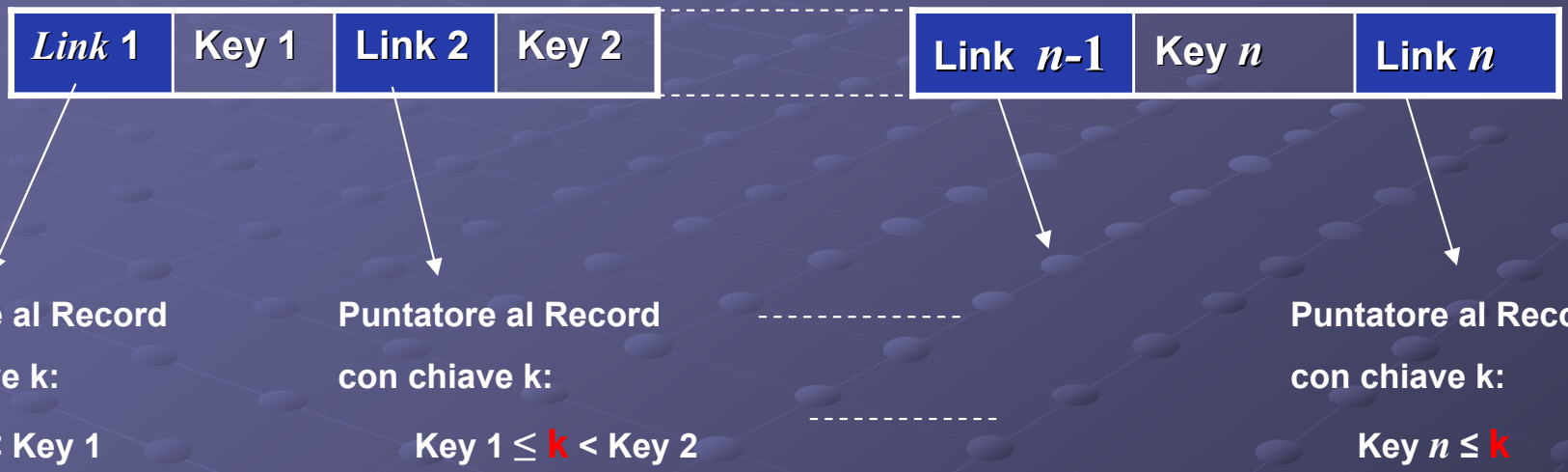
Alberi B

Un albero B di ordine **m** (in cui m rappresenta il **massimo numeri di figli che ogni nodo può avere**) è un albero di ricerca generico con le seguenti proprietà:

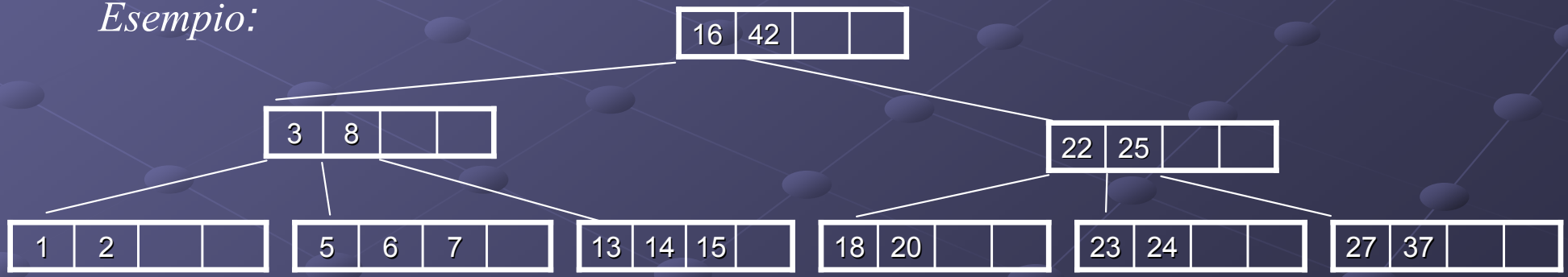
1. La radice ha almeno 2 sottoalberi, a meno che non sia una foglia
2. Ogni nodo che non sia la radice e che non sia una foglia contiene k – 1 chiavi e k riferimenti a sottoalberi, in cui $\lceil m/2 \rceil \leq k \leq m$
3. Ogni nodo foglia contiene k – 1 chiavi , in cui $\lceil m/2 \rceil \leq k \leq m$
4. Tutte le foglie si trovano sullo stesso livello

Secondo tali definizioni un albero B è sempre almeno pieno per metà, ha pochi livelli ed è perfettamente bilanciato

Alberi B (struttura nodo)



Esempio:



Alberi B (Operazioni e Proprietà)

Le operazioni fondamentali per gli alberi B sono:

- Creazione
- Inserimento
- Cancellazione
- Ricerca

Principali proprietà strutturali di un albero B:

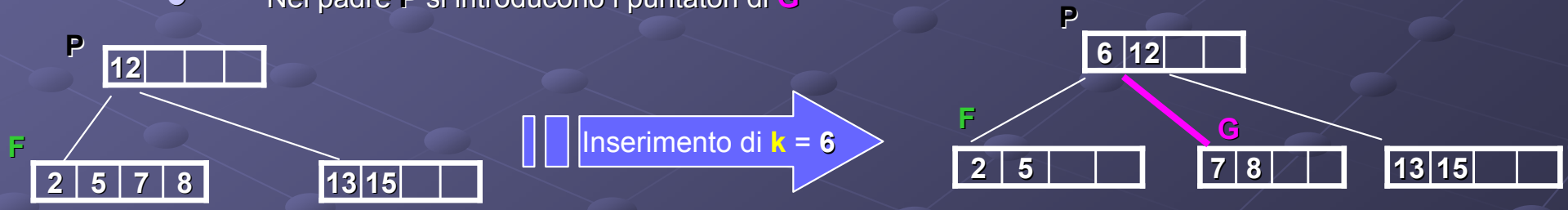
- E' un albero **bilanciato**
- **Prevedibilità per la complessità delle operazioni** di **ricerca** e/o **attraversamento** dell'albero

Alberi B (Inserimento)

Per l'inserimento di una nuova chiave K nell'albero B si hanno 3 casi:

1. **SE** la foglia **F** in cui deve essere piazzata **K** ha ancora spazio **ALLORA** si piazza **K**.

2. **SE** la foglia **F** in cui deve essere piazzata **K** è piena **ALLORA**
   - Si crea una nuova foglia **G** e metà delle chiavi di **F** vengono spostate in **G**;
   - Una chiave di **F** (per es. l'ultima oppure la mediana) si sposta nel proprio padre **P**;
   - Nel padre **P** si introducono i puntatori di **G**

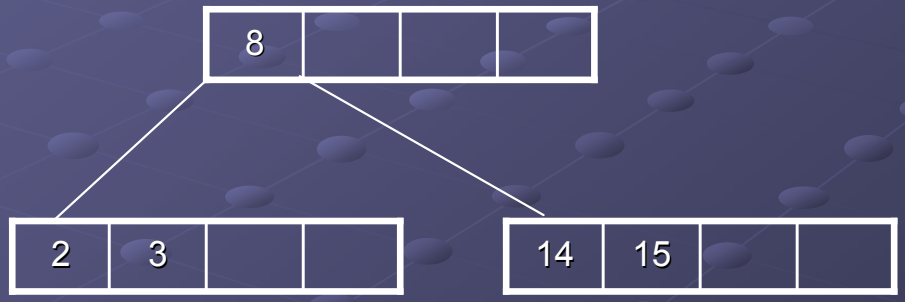


3. **SE** la radice **R** dell'albero è piena **ALLORA** si crea una nuova radice **S** ed un nuovo fratello **M** della radice **R**.

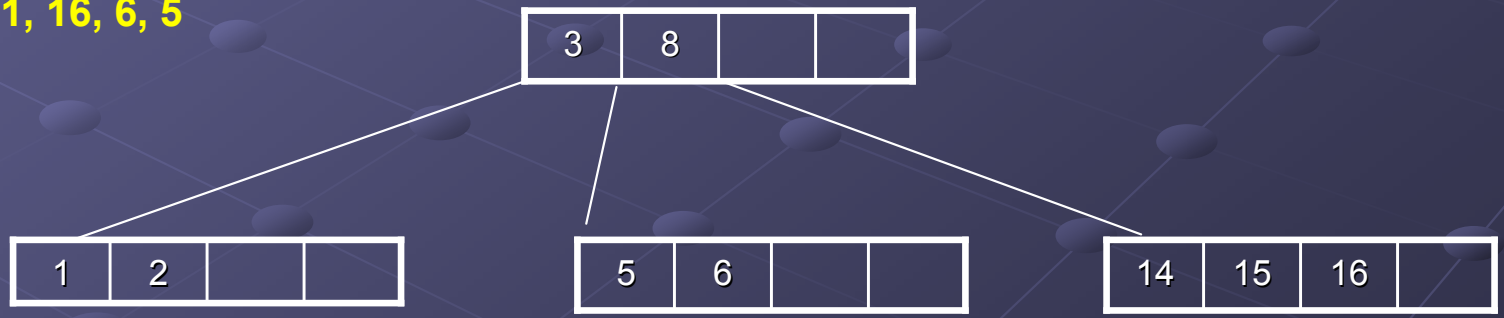
Alberi B (esempio di costruzione ed inserimento)

Inserisci 8, 14, 2, 15

Inserisci 3

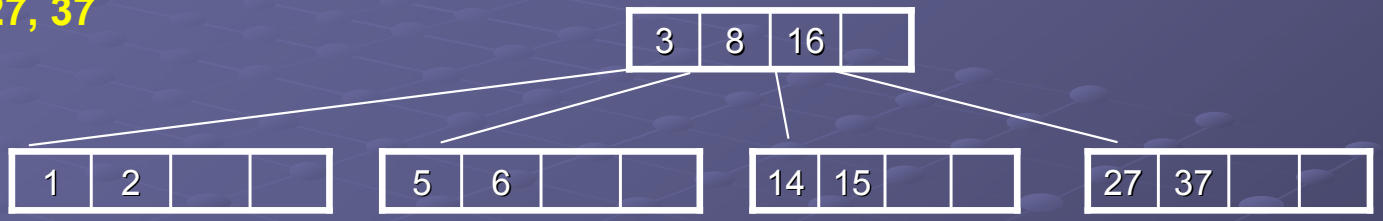


Inserisci 1, 16, 6, 5

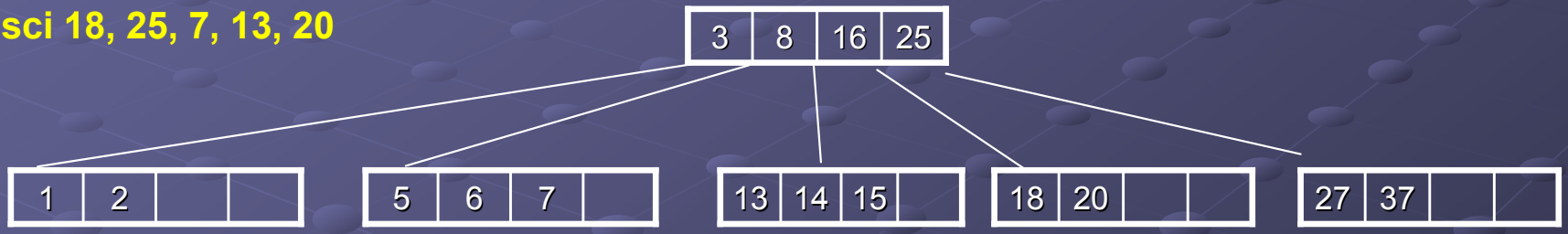


Alberi B (esempio di costruzione ed inserimento)

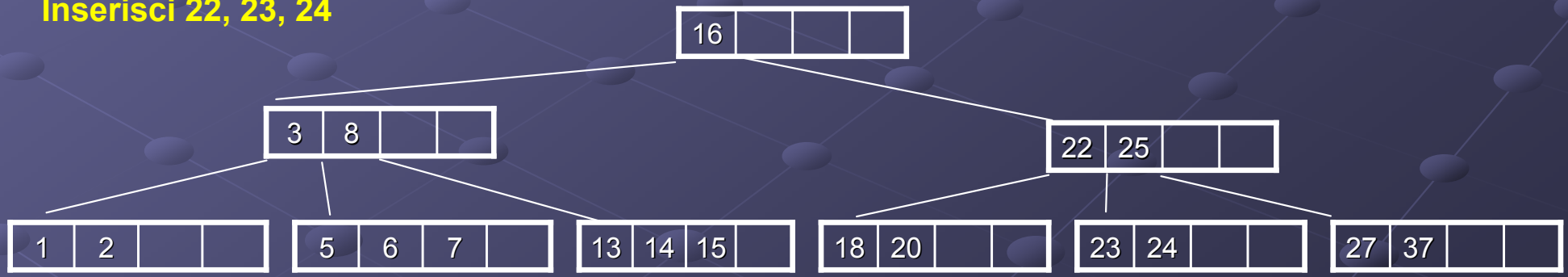
Inserisci 27, 37



Inserisci 18, 25, 7, 13, 20



Inserisci 22, 23, 24

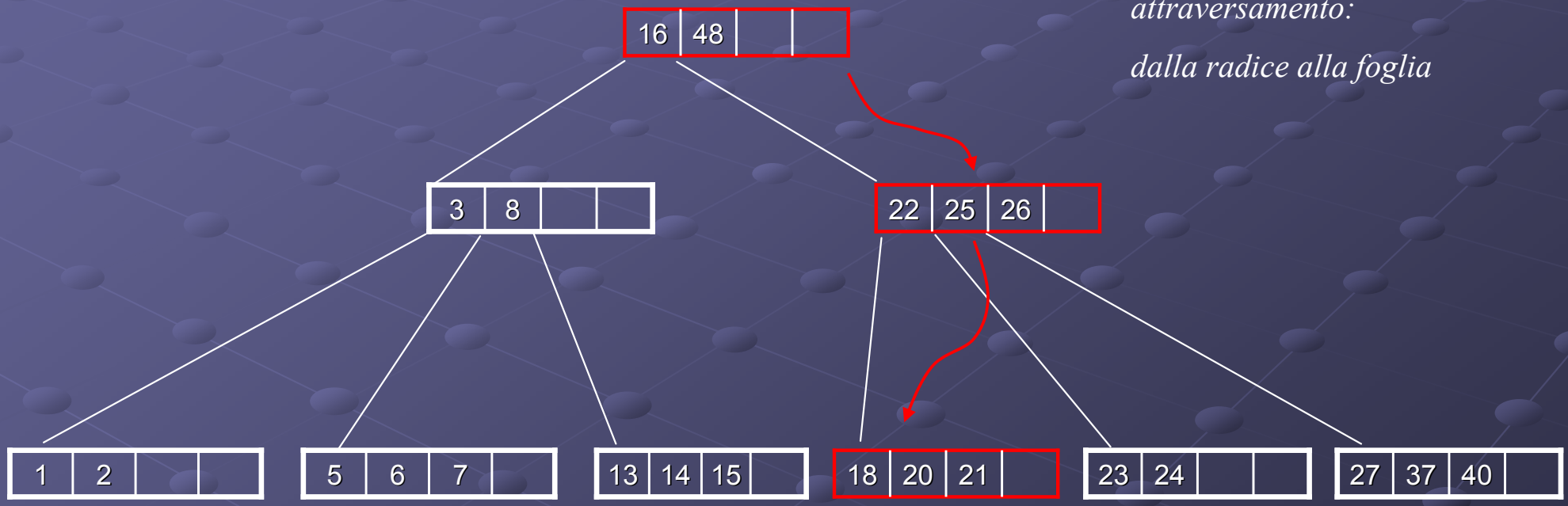


Alberi B (Ricerca)

La ricerca sugli alberi B è particolarmente **efficiente** in quanto essi sono intrinsecamente bilanciati

Esempio di attraversamento:

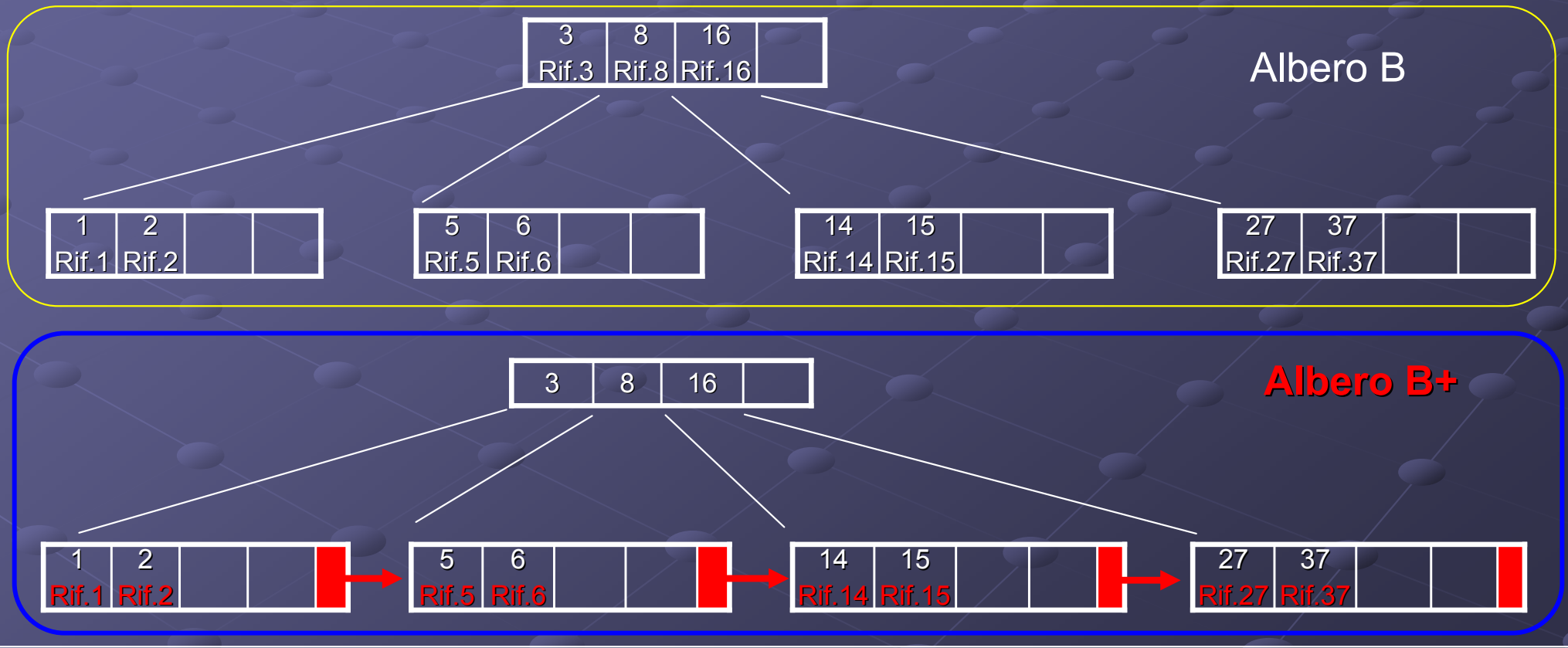
dalla radice alla foglia



Alberi B+

Gli Alberi B+ possono essere considerati come una **variante degli alberi B**. In particolare, le caratteristiche salienti di un B+ Albero sono:

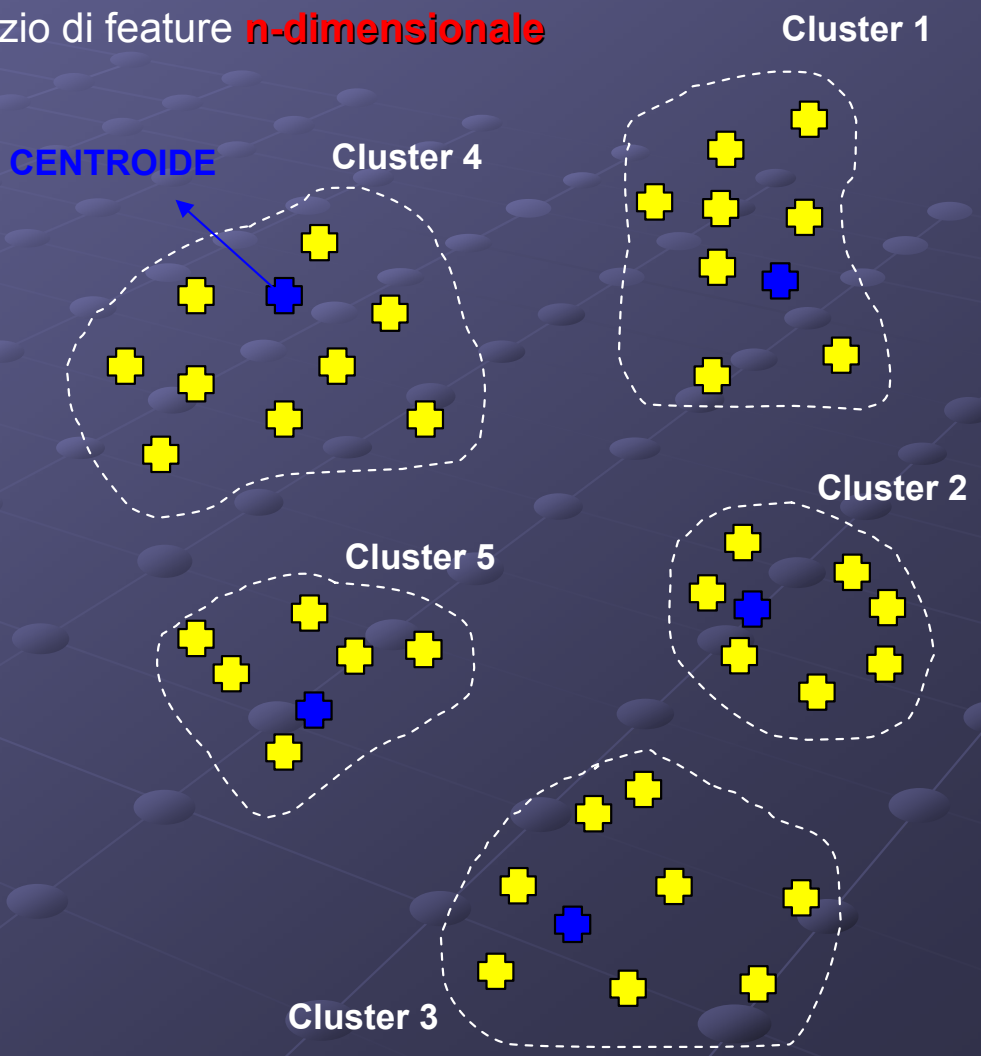
- In un albero B+ i riferimenti ai dati sono contenuti solo nelle foglie (invece in un albero B i riferimenti ai dati sono contenuti in un nodo qualsiasi dell'albero)
- Le foglie di un albero B+ contengono anche un **campo puntatore aggiuntivo che permette di "navigare" tra le foglie** come una linked list.



Clustering

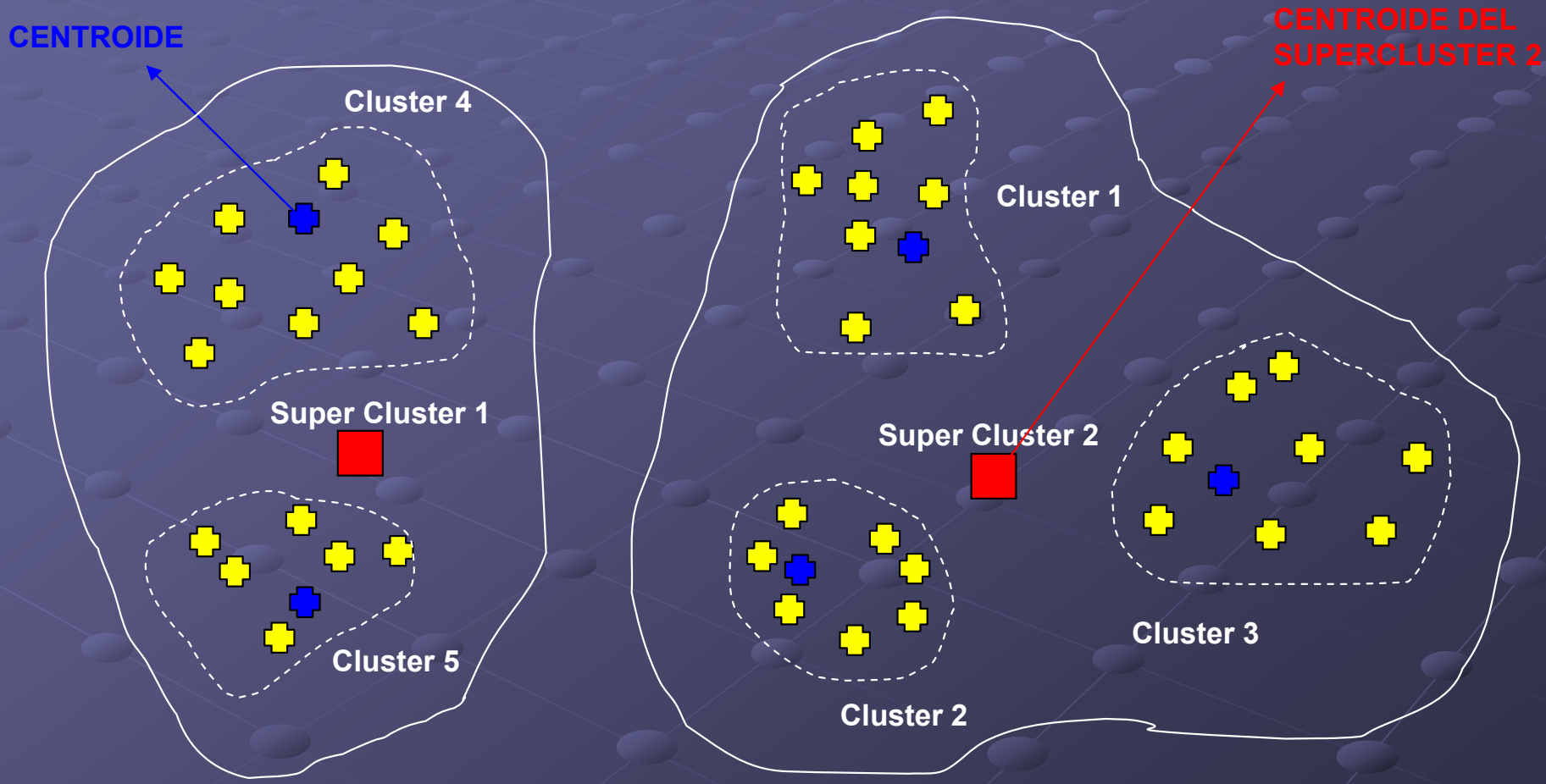
Tecnica per ottimizzare i tempi di ricerca nello spazio di feature **n-dimensionale**

- Vettori di features simili vengono raggruppati in **cluster** in base a **misure di similarità**
- Ogni cluster è rappresentato dal proprio **centroide**
- Il calcolo della similarità avviene tra la **query** ed il **centroide** di ogni cluster
- I cluster il cui centroide è più simile alla query vengono utilizzati per la **ricerca completa** sui vettori di features che contengono



Clustering a più livelli

Quando il numero di cluster è comunque alto si utilizzano cluster a livelli multipli per ridurre il numero di calcoli di similarità

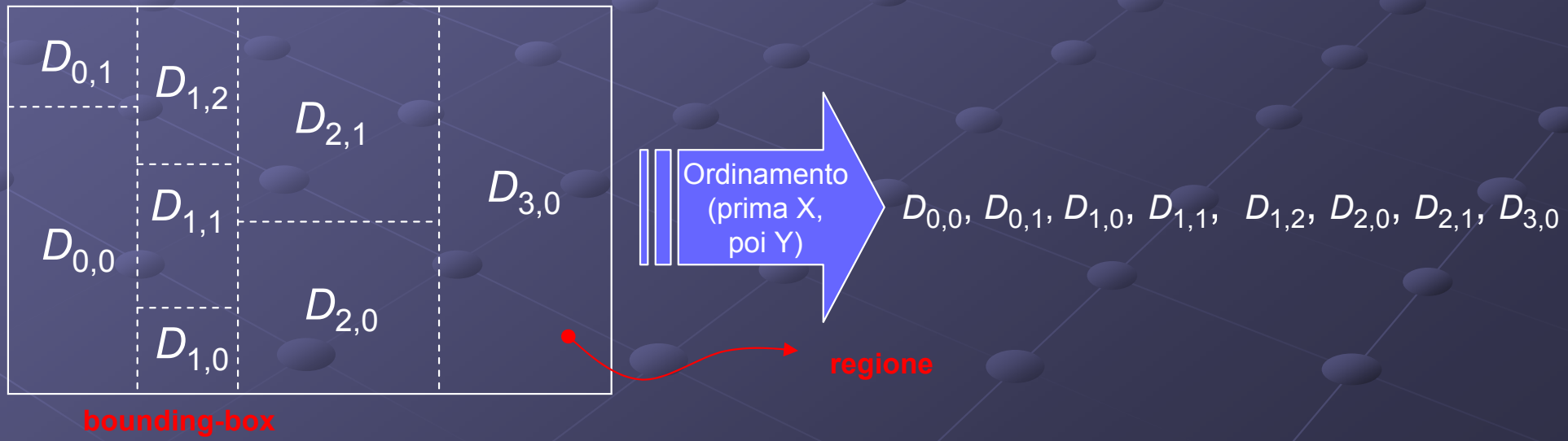


B+ tree MultiDimensionali (MB+ tree)

- Il **B**+ tree **M**ultidimensionale (**MB**+ tree) è una estensione del B+ Standard (da MonoDimensionale a MultiDimensionale)
- L'MB+ tree supporta le "Similarty Query" (Query per intervalli e per prossimità)

Esempio in 2D:

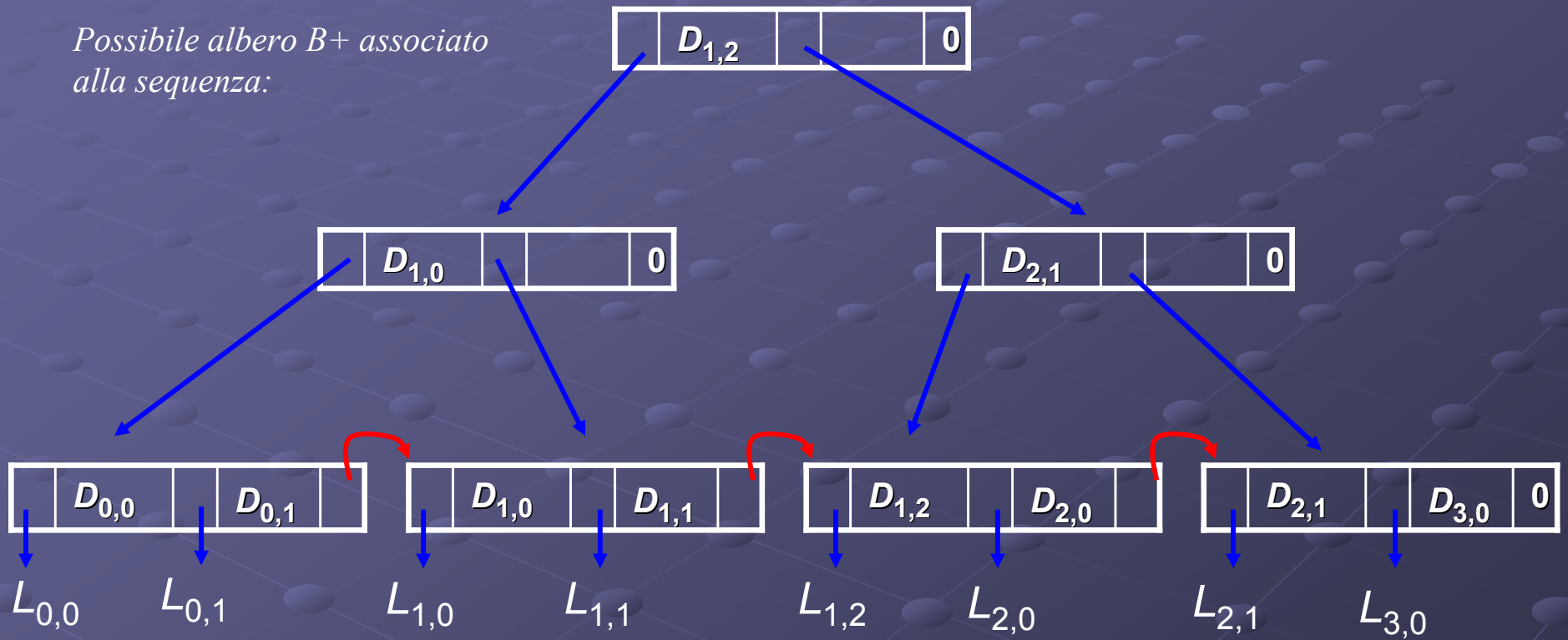
- Ogni feature vector è un punto nello spazio 2D.
- L'intero spazio delle feature "bounding-box" che contenente tutti i punti è il rettangolo identificato dallo spigolo in basso a sinistra (x_{min}, y_{min}) e dallo spigolo in alto a destra (x_{max}, y_{max}).
- Dividiamo il bounding-box in regioni contenenti feature simili.
- **Ordiniamo le regioni secondo un criterio (Prima X, poi Y)**. Ogni regione contiene i puntatori ai feature-vector che ricadono all'interno della regione
- Ogni feature-vector ha un link con il dato multimediale di cui è una rappresentazione



B+ tree MultiDimensionali (MB+ tree)

Ordinamento (prima X, poi Y) $D_{0,0}, D_{0,1}, D_{1,0}, D_{1,1}, D_{1,2}, D_{2,0}, D_{2,1}, D_{3,0}$

Possibile albero B+ associato alla sequenza:



In cui $L_{i,j}$ rappresenta la lista degli elementi relativi alla regione $D_{i,j}$

Differenze con B+ standard:
- **L'albero ottenuto dipende dall'ordinamento delle regioni (e non dalle chiavi)**
- **Ogni regione corrisponde alla lista di vettori di caratteristiche (anziché i record dei dati)**

Ricerca su MB trees MultiDimensionali

Point query

- Ricerca di **un** vettore dato (x,y)
- Si parte dalla root e si trova la regione che contiene il vettore da ricercare
- Si scorre la lista di feature-vector associata alla regione

Range query

- Ricerca di **tutti i vettori che ricadono in un rettangolo**
- Partendo dalla root troviamo tutte le regioni che si sovrappongono al rettangolo di ricerca
- Si scorre la lista di feature-vector associata alla regione **K**

Nearest-Neighbor query

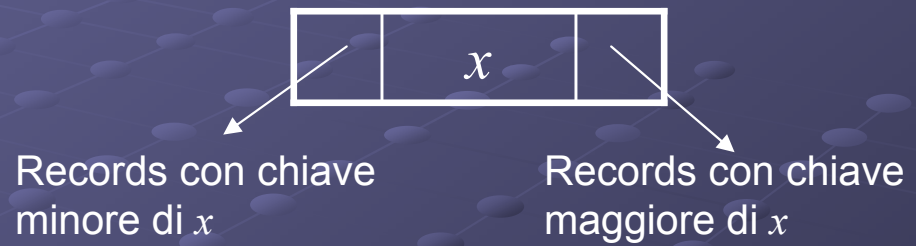
- Ricerca dei **k vettori più vicini ad un vettore dato**
- Si usa un procedimento iterativo, basato sulla ripetizione di Range query finché non si trova un numero sufficiente di vettori candidati.
- Si usa il calcolo della distanza euclidea tra il vettore da ricercare ed i vettori candidati

K-d trees

I **K**-d trees sono considerati una estensione degli ALBERI BINARI:

ALBERO BINARIO

- ogni nodo ha tre elementi:
  - un valore della chiave X
  - un puntatore ai record con chiave < x
  - un puntatore ai record con chiave > x
- e' normalmente non bilanciato
- in fase di inserimento si applicano **metodi di bilanciamento** per mantenere i tempi di ricerca in O(log n)



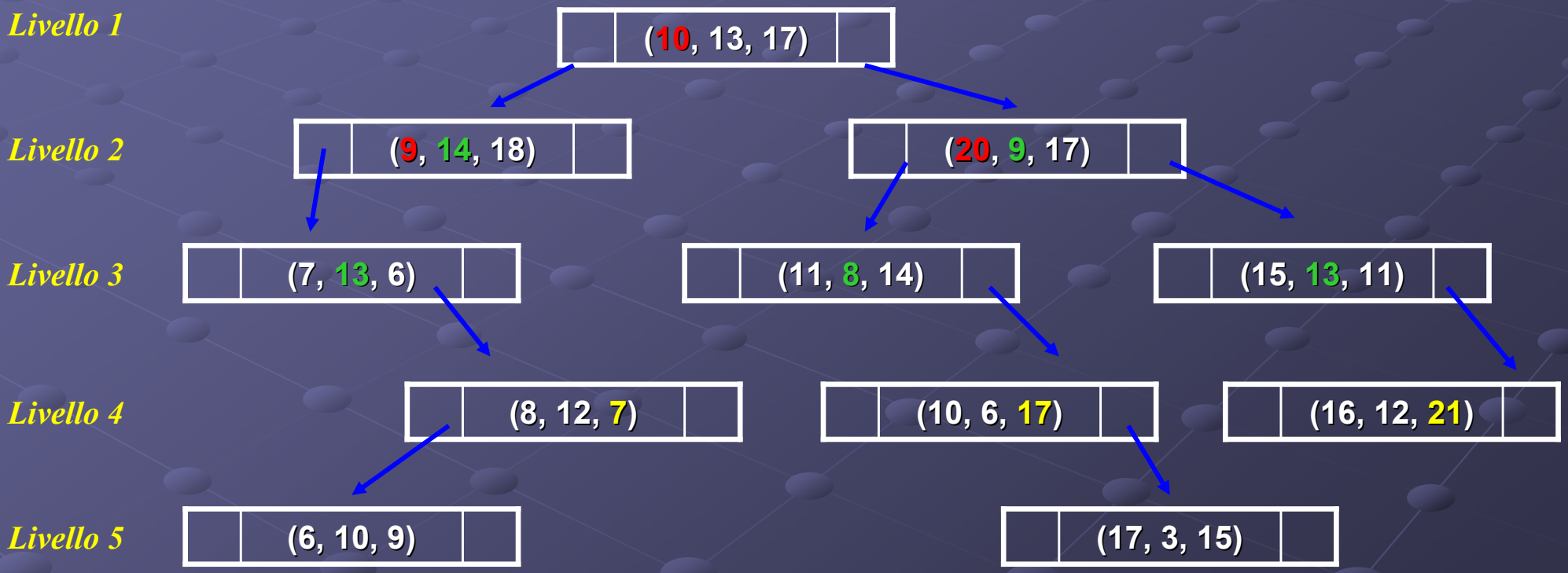
ALBERO K-D TREE

- ogni chiave è costituita dal vettore **K**-dimensionale invece che da un solo valore
- per generare l'albero occorre regolamentare la modalità di inserimento di un nuovo elemento (sinistra oppure destra):
  - **Al primo livello si decide basandosi sulla prima componente del vettore**
  - Al secondo livello si decide basandosi sulla seconda componente del vettore
  - **ecc…**
  - **Esaurite le K dimensioni si ricomincia dalla prima componente del vettore**

K-d trees (esempio)

Siano assegnati i seguenti vettori in uno spazio 3D:
(10,13,7) (9,14,8) (20,9,17) (7,13,6) (8,12,7) (6,10,9) (11,8,14) (15,13,11) (10,6,17) (16,12,21) (17,3,15)

E' possibile costruire il seguente albero:



K-d trees (operazioni)

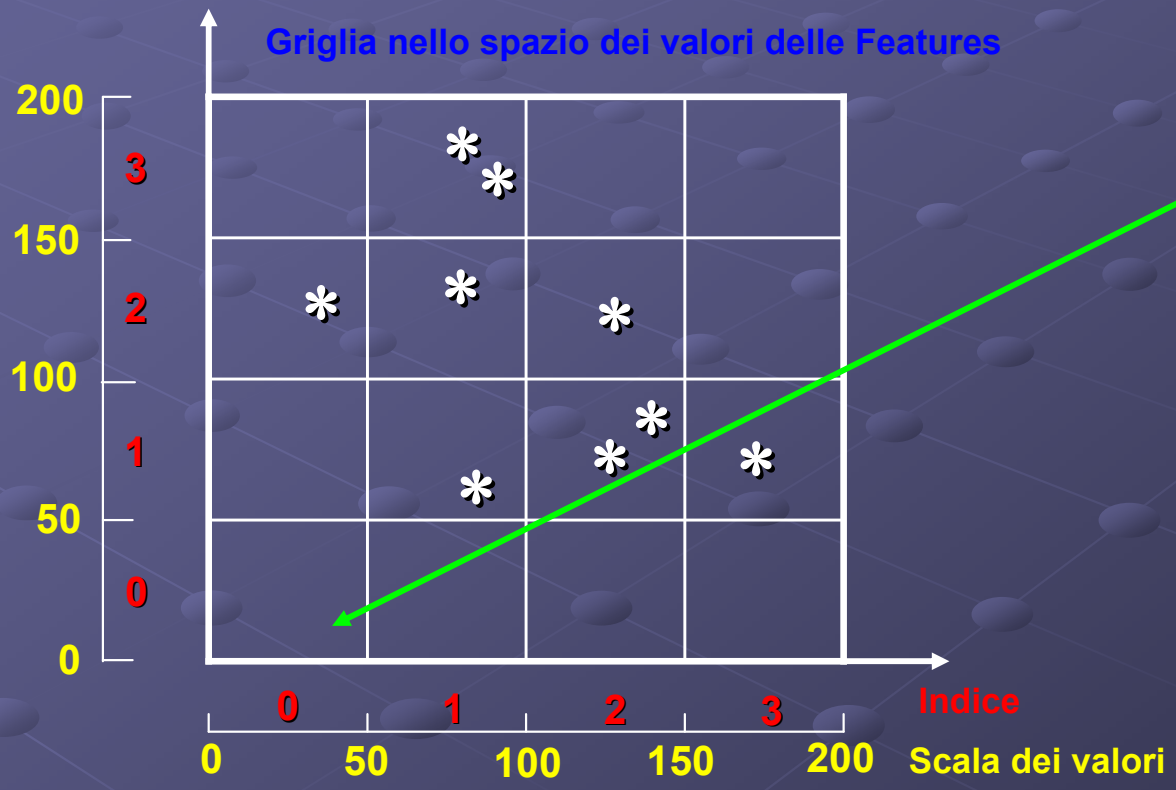
- **Inserimento:**
  - Per ciascun livello vale l'ordinamento relativo alla componente corrispondente del feature vector.
  - Problema: la struttura dell'albero dipende dall'ordine di inserimento dei record.
  - L'albero può diventare sbilanciato e richiedere operazioni di ri-bilanciamento
- **Ricerca:**
  - E' simile al processo di inserimento
  - Per ogni livello la scelta dipende solo dal valore della relativa componente del vettore
- **Eliminazione:**
  - Può risultare complicata quando occorre eliminare un nodo intermedio dell'albero
  - Si possono effettuare delle cancellazioni logiche senza modificare la struttura dell'albero
- **Range query:**

  Facile da implementare e comporta la riduzione del numero dei nodi da visitare

Grid files

I Grid files costituiscono una modalità di indicizzazione e ricerca semplice, spesso impiegata in implementazioni reali.
Consistono nella suddivisione dello spazio n-dimensionale in ipercubi aventi tutti la stessa dimensione.
Ogni ipercubo contiene zero o più feature-vector

Esempio in 2D:



Array 2D di puntatori alle griglie

$$\begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$$

Grid Files (operazioni)

- L'**inserimento** di un valore nella struttura è molto semplice:
  - ***Esempio:*** se nella griglia 2D precedente si deve inserire il vettore di feature {80, 70} allora esso dovrà essere inserito nella griglia con indice (1,1) e quindi sarà puntato dal puntatore $P_{1,1}$

- La **ricerca** di un valore avviene in un modo simile:
  - ***Esempio di point query:*** per trovare il vettore {40, 125} basterà puntare alla cella (0,2) della griglia e quindi scorrere la lista di vettori puntati da questo (tutti i vettori di feature che si trovano nella stessa griglia sono puntati dallo stesso puntatore)
  - ***Esempio di range query:*** è necessario trovare tutte le liste di vettori puntate dai puntatori le cui griglie vengono intersecate dal rettangolo descritto dalla query

Grid files (considerazioni)

SE

i vettori di features sono distribuiti abbastanza uniformemente all'interno dello spazio dei valori

ALLORA

tale metodo da buoni risultati

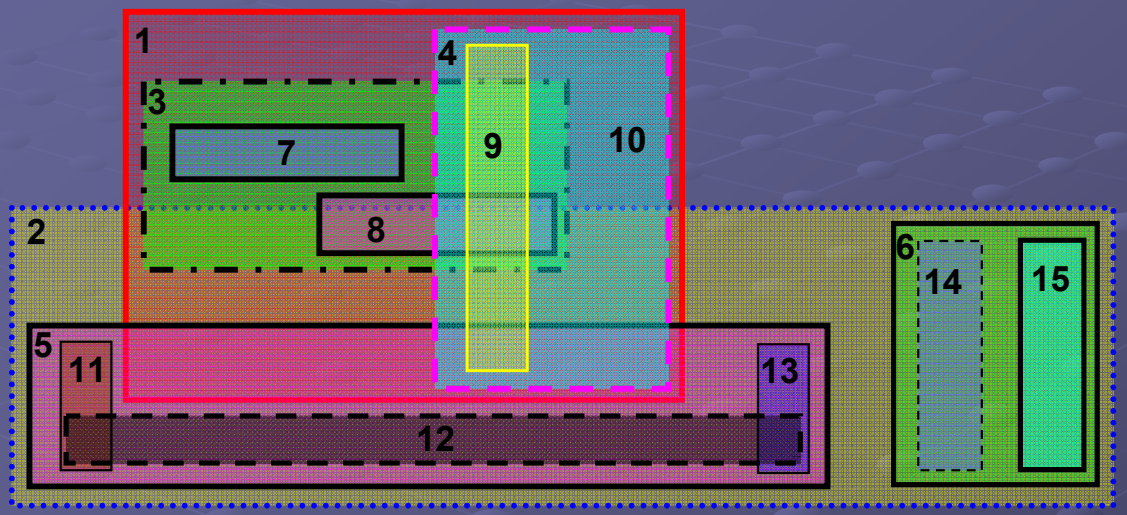
ALTRIMENTI alcune griglie risultano vuote o quasi e altre sovraffollate

- Se una griglia e' sovraffollata il puntatore alla griglia individuerà una lista di vettori molto lunga il cui scorrimento e calcolo della similarità comporta perdita di tempo elevata
- Per far fronte a questo problema (nel caso di distribuzione non uniforme) invece di utilizzare griglie fisse della stessa dimensione si crea una suddivisione adattativa cercando di bilanciare il contenuto delle diverse griglie
- Nelle zone dello spazio densamente popolate si utilizzano griglie di piccole dimensioni mentre nelle zone scarsamente popolate si utilizzano griglie di dimensioni più grandi

R Tree (Rectangle)

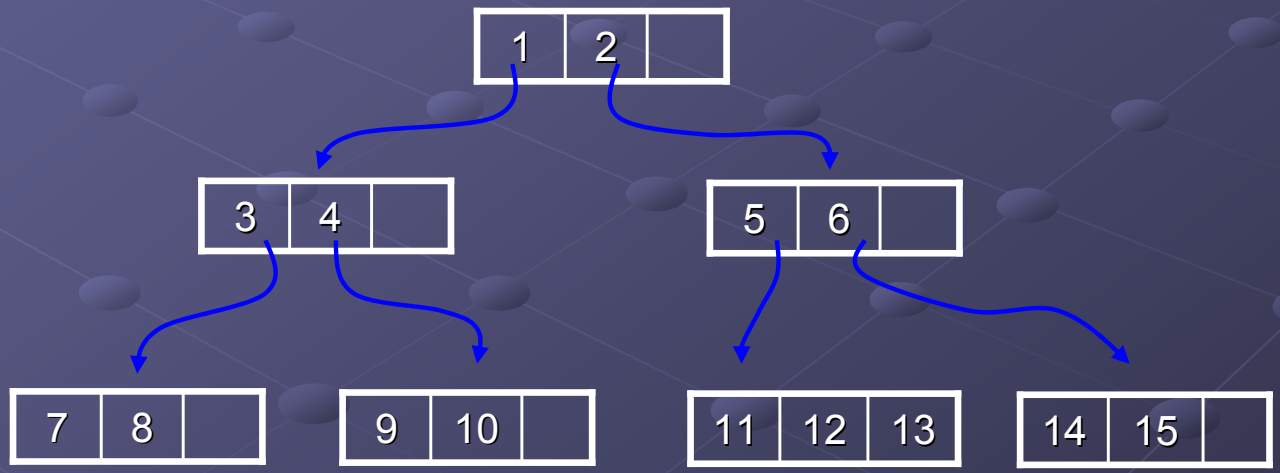
- L' R Tree (**R**ectangle) è una generalizzazione dell' MB+ Tree ed identifica una famiglia di strutture indicizzate molto utilizzate per l'organizzazione dei **dati multidimensionali** (per es. dati spaziali e geografici).
- La struttura dati divide lo spazio in **MBR** (**M**inimum **B**ounding **R**ectangles)
- Ogni nodo dell'R-tree ha un numero variabile di entry (fino ad un massimo predeterminato).
- Ogni entry che non sia un nodo foglia contiene due entità: una identifica il nodo figlio, l'altra l'MBR che contiene tutte le entry del nodo figlio.
- In ogni nodo non foglia viene memorizzato un puntatore che punta ad un nodo di livello più basso nell'albero e un rettangolo che copre tutti i rettangoli associati ai discendenti del nodo.
- Nei nodi foglia viene memorizzata la lista dei vettori che ricadono dentro al singolo rettangolo di livello più basso.
- Sono strutture dati utilizzate sia per memorizzare dati che hanno un "boundingbox" che dati di tipo puntuale

R Tree (Esempio)



Ordine Spaziale

Si Noti come l'area indicizzata da ogni record comprenda l'unione delle aree dei figli



Rappresentazione R Tree

R Tree (Operazioni)

Query:

- La regione da cercare viene caratterizzata dal suo MBR (minimum bounding-box)
- A partire dalla root si attraversa l'albero cercando i rettangoli che intersecano l'MBR (possono essere più di uno ad ognuno dei livelli)
- Raggiunti i nodi foglia, si calcola l'intersezione tra l'MBR e il rettangolo collegato

Insert:

- Si attraversa l'albero selezionando il rettangolo più piccolo che include l'oggetto da inserire o quello che richiederebbe l'allargamento minore per "coprire" il nuovo oggetto.
- L'inserimento comporta l' "allargamento" del nodo padre per fare in modo che il suo rettangolo includa completamente il nuovo oggetto
- Se il nodo nell'albero è già pieno per più di metà, occorre procedere alla operazione di splitting in maniera analoga a quanto avviene sui MB+ tree
- Lo splitting si può ripercuotere ricorsivamente verso l'alto fino a quando l'aggiunta di un nuovo rettangolo non comporta un riempimento eccessivo

Delete:

- Si utilizza un procedimento di attraversamento dell'albero simile a quello della ricerca
- Se l'eliminazione di un oggetto comporta che un nodo dell'albero contiene troppo pochi elementi, il nodo viene eliminato e gli oggetti che conteneva vengono reinseriti nell'albero

R tree (Dati puntuali ed efficienza di ricerca)

Dati Puntuali

- Per la **ricerca**, **l'inserimento** e **l'eliminazione** di dati puntuali in un R TREE gli algoritmi sono simili a quelli precedenti (dati multidimesionali) e l'unica differenza consiste nel fatto che **ogni nodo foglia contiene più di un elemento puntuale di cui viene memorizzato l'MBR**
- E' possibile implementare la **ricerca k nearest neighbor** attraverso la stima di un rettangolo che contiene sicuramente tutti i punti da ricercare e utilizzando la ricerca su oggetti rettangolari descritta in precedenza
- L'inserimento di un punto avviene ricercando il rettangolo dell'albero che deve essere **ampliato di meno** per contenerlo
- In modo analogo al caso di oggetti rettangolari vengono trattati i casi di **splitting** (quando la lista di punti di un nodo
- conterrebbe troppi elementi) o di eliminazione di un nodo a seguito della eliminazione di uno o più punti

Efficienza di ricerca

Dipende da 2 concetti (definiti per ognuno dei livelli dell'albero):

- **COVERAGE:** E' l'area totale di tutti i rettangoli associati ai nodi del livello
- **OVERLAP:** E' l'area totale coperta da due o più nodi

Un R tree è efficiente se **sia la COVERAGE che l'OVERLAP sono minimizzati**; in particolare l'overlap comporta problemi in fase di ricerca.

Inoltre è cruciale l'ordine di inserimento per ottenere un albero maggiormente bilanciato