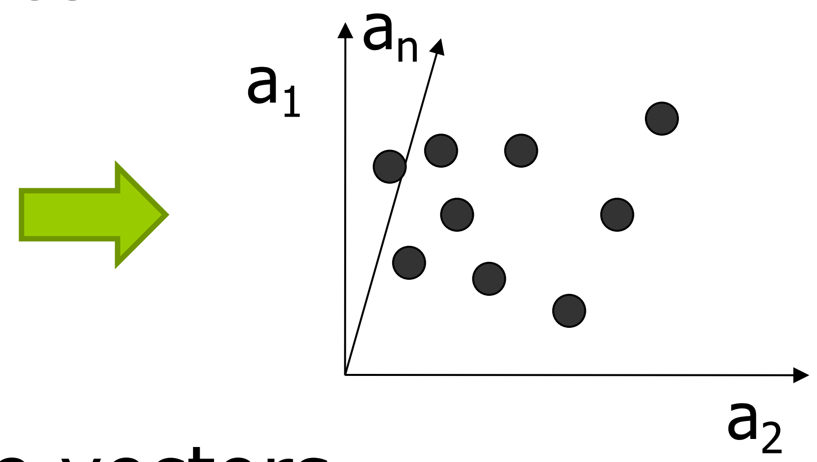
Beyond Similarity Search

- Top-k queries
- Skyline queries
- OLAP
- Time-travel search
- Cluster analysis

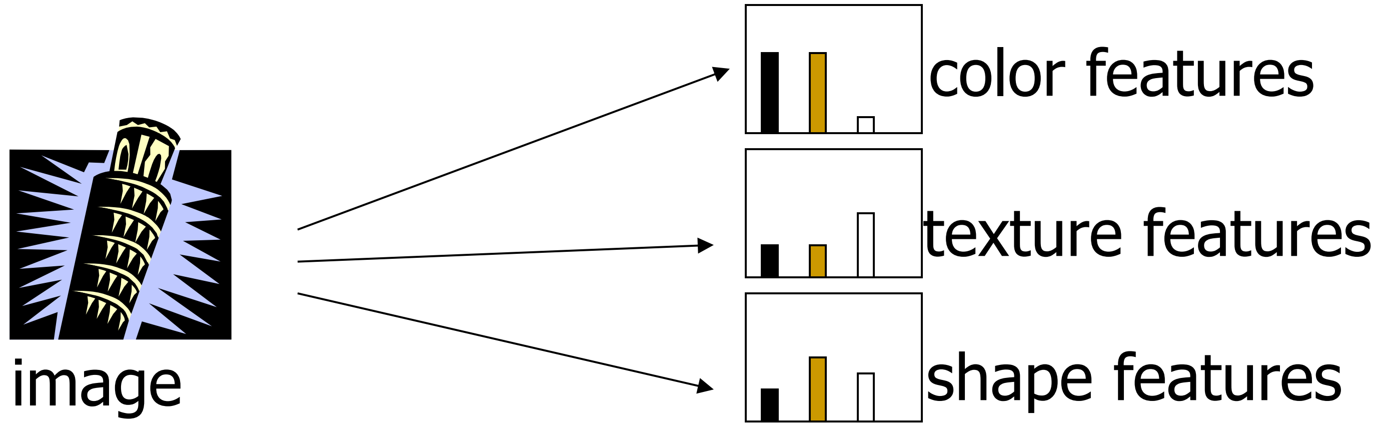
Multidimensional Data

- Flat relational tables

a_1	a_2	…	a_n
v_{11}	v_{12}	…	v_{1n}
v_{11}	v_{12}	…	v_{1n}
v_{11}	v_{12}	…	v_{1n}
…	…	…	…
v_{m1}	v_{m2}	…	v_{mn}

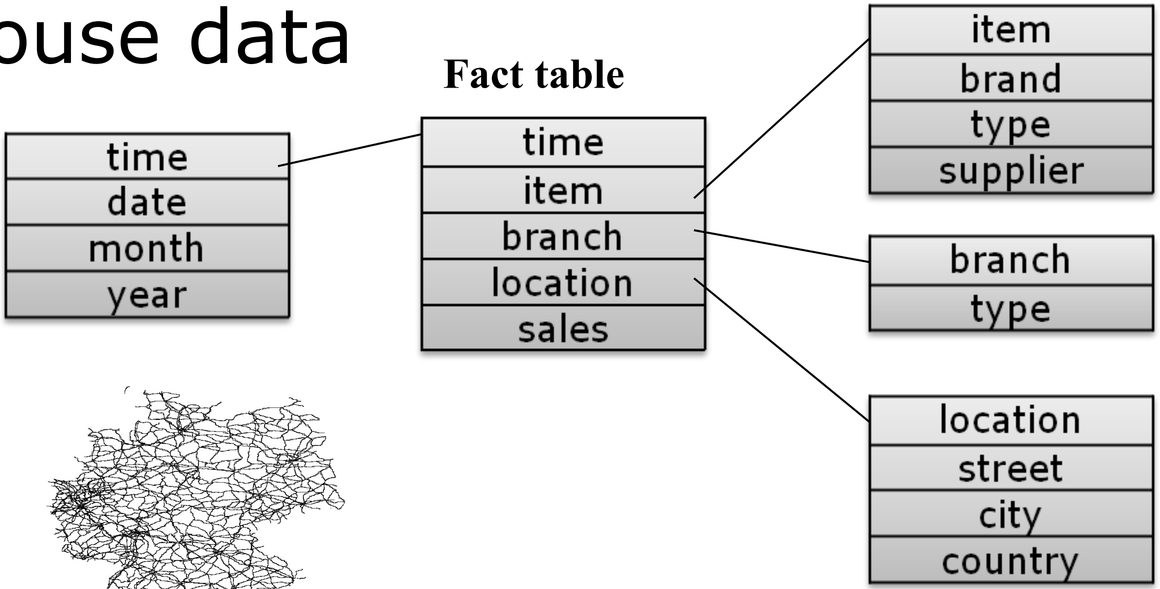


- Multimedia feature vectors

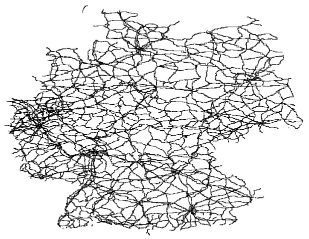


Multidimensional Data (cont'd)

- Data warehouse data



- Spatial data


German roads

- Text documents

	term$_1$	term$_2$	…	term$_n$
d$_1$	f_{11}	f_{12}	…	f_{1n}
d$_2$	f_{11}	f_{12}	…	f_{1n}
…	…	…	…	…
d$_m$	f_{m1}	f_{m2}	…	f_{mn}

Attribute types

- Attributes of multidimensional tuples may have various types
  - Ordinal (e.g., age, salary)
  - Nominal categorical values (e.g., color, religion)
  - Binary (e.g., gender, owns_property)
- Distance between multidimensional ordinal tuples can be computed by an L_p norm
  - e.g., Euclidean distance
- Categorical values are reduced to binary or ordinal ones. Distance between multidimensional binary tuples can be computed by information theory measures
  - e.g., Jaccard coefficient, Hamming distance, etc.

Distance/similarity-based Object Retrieval

- (Range) selection query
  - Returns the records that qualify a (multidimensional) range predicate
  - Example:
    - Return the employees of age between 45 and 50 and salary above $100,000
- Distance (similarity) query
  - Returns the records that are within a distance from a reference record.
  - Example:
    - Find images with feature vectors of distance at most ε with the feature vector of a given image
  - Distance usually defined by an L_p norm
- Nearest neighbor (similarity) query
  - Replaces distance bound by ranking predicate

Top-k Queries

Top-k query

- Given a set of objects (e.g., relational tuples),
- Returns the k objects with the highest combined score, based on an aggregate function f.
- Example:
  - Relational table containing information about restaurants, with attributes
    - Price
    - Quality
    - Location
  - f: sum(-price, quality, -dist(location,my_hotel))
  - attribute value ranges are usually normalized
    - E.g., all values in a [0,1] range
    - otherwise some attribute may be favored in f

Top-k query variants

- Apply on single table, or ranked lists of tuples ordered by individual attributes

	price	quality	location
a	0.1	0.9	$<x_a,y_a>$
b	0.6	0.8	$<x_b,y_b>$
c	0.9	0.2	$<x_c,y_c>$
d	0.8	0.4	$<x_d,y_d>$
e	0.3	0.6	$<x_e,y_e>$

vs.

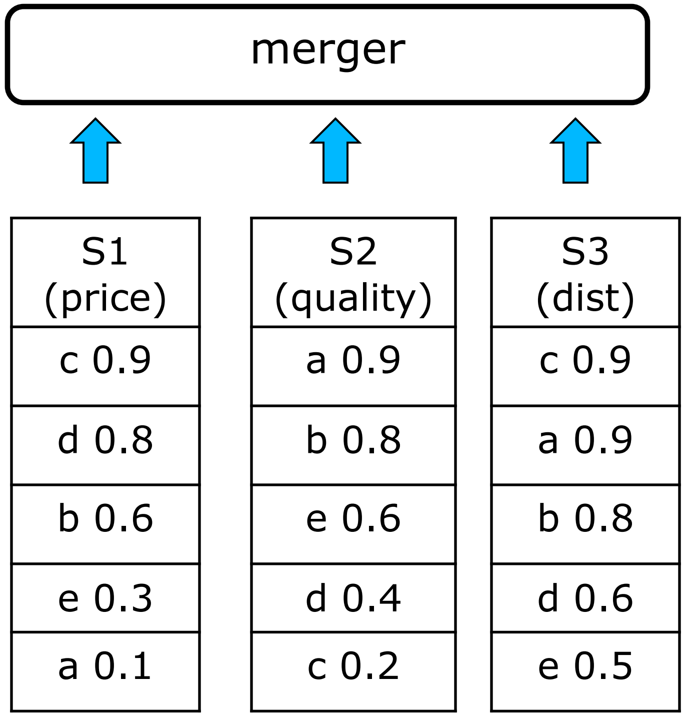
S1 (price)	S2 (quality)	S3 (dist. to hotel)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

Top-k query variants (cont'd)

- Ranked inputs in the same or different servers (centralized or distributed data)

S1 (price)	S2 (quality)	S3 (dist. to hotel)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

vs.



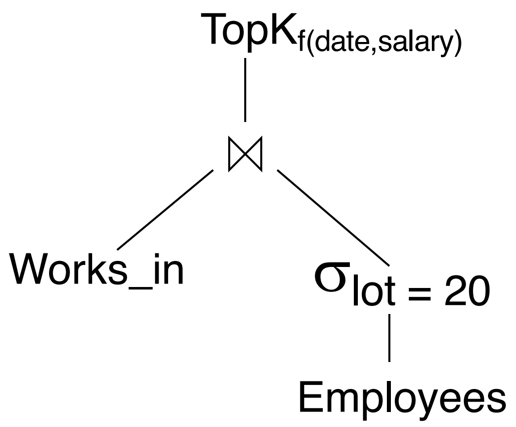
Top-k query variants (cont'd)

- Standalone query or operator in a more complex query plan

simple top-k query:

```
SELECT name
FROM Employee
ORDER BY 0.5*salary+0.5*age
LIMIT 10
```

complex plan with top-k query:



Top-k query variants (cont'd)

- Incremental retrieval of objects with highest scores (k is not predefined)
  - similar to incremental NN search
- Top-k joins (join key is not object-id)

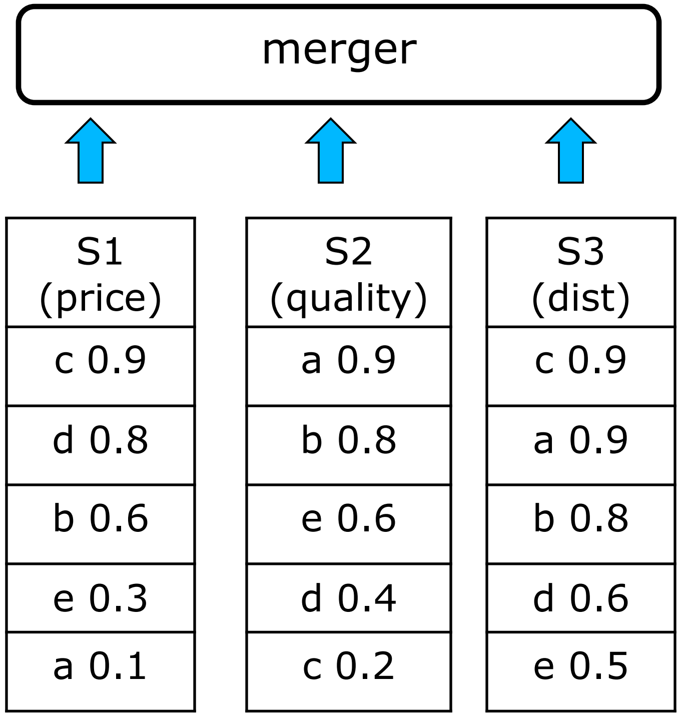
  -
    ```
    SELECT h.id, s.id
    FROM House h, School s
    WHERE h.location=s.location
    ORDER BY h.price + 10 * s.tuition
    LIMIT 5
    ```
- Probabilistic/approximate top-k retrieval
- Random and/or sorted accesses at ranked inputs

Top-k query evaluation

- Most solutions assume distributive, monotone aggregate functions (e.g. f=sum)
  - distributive: f(x,y,z,w)= f(f(x,y),f(z,w))
  - monotone: if x<y and z<w, then f(x,z)<f(y,w)
- Solutions based on 1-D ordering and merging sorted lists (rank aggregation)
- Solutions based on multidimensional indexing

Rank Aggregation

- Solutions based on 1-D ordering and merging sorted lists
- Assume that there is a total ranking of the objects for each attribute that can be used in top-k queries
- These sorted inputs can be accessed sequentially and/or by random accesses

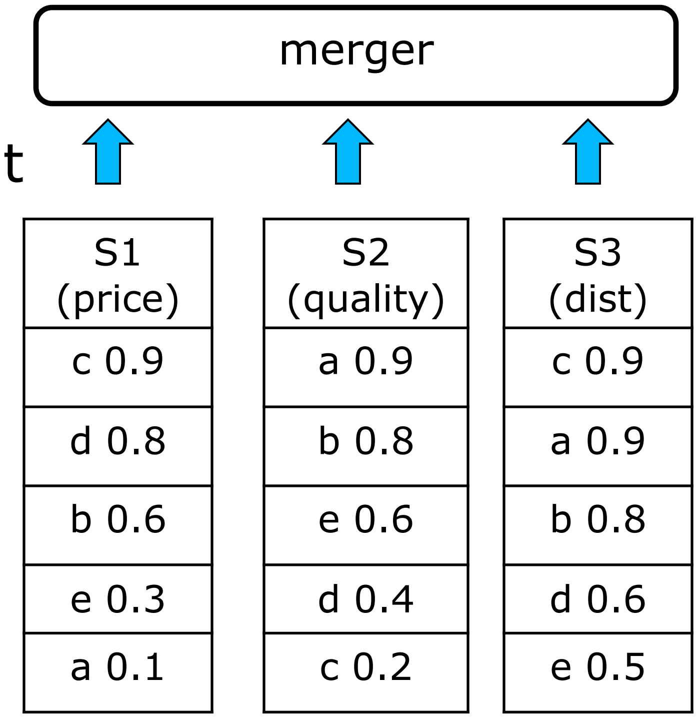


S1 (price)	S2 (quality)	S3 (dist)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

Rank Aggregation

- Advantages:
  - can be applied on any subset of inputs (arbitrary subspace)
  - appropriate for distributed data
  - appropriate for ad-hoc top-k joins
  - easy to understand and implement

- Drawbacks:
  - slower than index-based methods
  - require inputs to be sorted



S1 (price)	S2 (quality)	S3 (dist)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

Rank Aggregation: TA

- TA = Threshold Algorithm
- Iteratively retrieves objects and their atomic scores from the ranked inputs in a round-robin fashion.
- For each encountered object x, perform random accesses to the inputs where x has not been seen.
- Maitain top-k objects seen so far.
- $T = f(l_1, \ldots, l_m)$ is the score derived when applying the aggregation function to the last atomic scores seen at each input. If the score of the k-th object is no smaller than T, terminate.

Example of TA (k=1, f=sum)

S1	S2	S3
🟩 c 0.9	🟩 a 0.9	🟩 c 0.9
d 0.8	b 0.8	🟩 a 0.9
b 0.6	e 0.6	b 0.8
…	…	…
e 0.3	d 0.4	d 0.6
🟩 a 0.1	🟩 c 0.2	e 0.5

🟩 accessed data

- top-1 is c, with score 2.0
- T=sum(0.9,0.9,0.9)=2.7
- T>top-1, we proceed to another round of accesses

Example of TA (k=1, f=sum)

S1	S2	S3
🟩 c 0.9	🟩 a 0.9	🟩 c 0.9
🟩 d 0.8	🟩 b 0.8	🟩 a 0.9
🟩 b 0.6	e 0.6	🟩 b 0.8
…	…	…
e 0.3	🟩 d 0.4	🟩 d 0.6
🟩 a 0.1	🟩 c 0.2	e 0.5

🟩 accessed data

- top-1 is b, with score 2.2
- T=sum(0.8,0.8,0.9)=2.5
- T>top-1, we proceed to another round of accesses

Example of TA (k=1, f=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
…	…	…
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- top-1 is b, with score 2.2
- T=sum(0.6,0.6,0.8)=2.0
- T≤top-1, terminate and output (b,2.2)

Properties of TA

- Used as a standard module for merging ranked lists in many applications
- Usually finds fast the result
- Depends on random accesses
    - random accesses can be expensive
    - random accesses are impossible in some cases
        - e.g., an API allows to access objects incrementally by ranking score, but does not provide the score of a given object

Rank Aggregation: NRA

- NRA: No Random Accesses
- Iteratively retrieves objects and their atomic scores from the ranked inputs in a round-robin fashion.
- For each object x seen so far at any input maintain:
  - f_x^{ub} : upper bound for x's aggregate score (f_x)
  - f_x^{lb} : lower bound for x's aggregate score (f_x)
- W_k = k objects with the largest f^{lb}.
- If the smallest f^{lb} in W_k is at least the largest f_x^{ub} of any object x not in W_k, then terminate and report W_k as top-k result.

Example of NRA (k=1, f=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{ub} = 2.7$, $f_c^{lb} = 1.8$
- $f_a^{ub} = 2.7$, $f_a^{lb} = 0.9$
- $W_k = \{c\}$
- Since $f_c^{lb} < f_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, f=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{ub} = 2.6$, $f_c^{lb} = 1.8$
- $f_a^{ub} = 2.6$, $f_a^{lb} = 1.8$
- $f_d^{ub} = 2.5$, $f_d^{lb} = 0.8$
- $f_b^{ub} = 2.5$, $f_b^{lb} = 0.8$
- $W_k = \{c\}$
- Since $f_c^{lb} < f_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, f=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{ub} = 2.4$, $f_c^{lb} = 1.8$
- $f_a^{ub} = 2.4$, $f_a^{lb} = 1.8$
- $f_d^{ub} = 2.2$, $f_d^{lb} = 0.8$
- $f_b^{ub} = f_b^{lb} = f_b = 2.2$
- $f_e^{ub} = 2.0$, $f_e^{lb} = 0.6$
- $W_k = \{b\}$
- Since $f_b^{lb} < f_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, f=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{ub} = 2.2$, $f_c^{lb} = 1.8$
- $f_a^{ub} = 2.1$, $f_a^{lb} = 1.8$
- $f_d^{ub} = f_d^{lb} = f_d = 1.8$
- $f_b^{ub} = f_b^{lb} = f_b = 2.2$
- $f_e^{ub} = 1.5$, $f_e^{lb} = 0.9$
- $W_k = \{b\}$
- $f_b^{lb} \geq f_c^{ub}$, thus NRA terminates and reports W_k

NRA Properties

- More generic than TA, since it does not depend on random accesses
- Can be cheaper than TA, if random accesses are very expensive
- NRA accesses objects sequentially from all inputs and updates the upper bounds for all objects seen so far unconditionally.
  - Cost: O(n) per access (the expected distinct number of objects accessed so far is O(n))
  - No input list is pruned until the algorithm terminates

LARA: An efficient NRA implementation

- LARA: LAttice-based Rank Aggregation
- Based on 3 observations about the top-k candidates
- Operates differently in the two (growing, shrinking) phases
- Takes its name from the lattice used in the shrinking phase
- Extendable to various top-k query variants

Observation 1

- Let t be the k-th highest score in W_k and $T = f(l_1, . . . , l_m)$ be the score derived when applying the aggregation function to the last atomic scores seen at each input.
  - Example
    - k=2
    - t=1.8
    - T=2.5

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
.	.	.
.	.	.

🟩 accessed data

- While t < T, any objects never seen so far at any input can end up in the top-k result

Example of Observation 1 (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
x 0.8	x 0.8	x 0.9
.	.	.
.	.	.

🟩 accessed data

- $T = 2.5$, $W_k = \{c\}$, $t = f_c^{lb} = 1.8$
- There could be an object x not seen yet at any input with scores (0.8, 0.8, 0.9)

Observation 2

- While t < T, any of the objects seen so far can end up in the top-k result.
- Example (k=1):
  - $T = 2.5$, $W_k = \{c\}$, $t = f_c^{lb} = 1.8$
  - any object {a,b,c,d} could end up in the result

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
.	.	.
.	.	.
.	.	.

🟩 accessed data (rows c 0.9 / a 0.9 / c 0.9 and d 0.8 / b 0.8 / a 0.9)

Implication of observations 1,2

- While $t < T$ the set of candidate top-k objects can only grow and there is nothing we can do about it. Objects which have not been seen so far at any input can end up in the top-k result
- Thus, while $t < T$, *we should only update* W_k *and* T *while accessing objects from the sources and need not apply expensive updates and comparisons using upper bounds*.

Observation 3

- If $t \geq T$, no object which has not been seen at any input can end up in the top-k result.
- Example:
  - $T = 2.0$, $W_k = \{b\}$, $t = f_b^{lb} = 2.2$
  - no object x never seen so far can end up in the result

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
x 0.6	x 0.6	x 0.8
.	.	.

🟩 accessed data

Implication of observation 3

- As soon as $t \geq T$ the set of candidate top-k objects can only shrink and there is no need to process any newly seen objects.
- Summarizing, observations 1 through 3 imply two phases that all NRA algorithms go through;
  - a *growing* phase during which $t < T$ and the set of top-k candidates can only grow and
  - a *shrinking* phase during which $t \geq T$ and the set of candidate objects can only shrink, until the top-k result is finalized.

LARA: Growing phase

- While ($t<T$) // growing phase
  - Iteratively retrieve objects and their atomic scores from the ranked inputs in a round-robin fashion.
  - Update lower bound (f_x^{lb}) for each newly accessed object x.
  - Update W_k (the set of the k objects with the largest f^{lb}) and compute t from it.
  - Update T from the last atomic scores seen at each input.
- W_k is updated at O(logk) cost after each access (heap implementation). T's update cost is O(1).

LARA: Shrinking phase

- As soon as $t \geq T$, LARA enters the shrinking phase.
- Accessed objects that have not been seen during the growing phase are immediately pruned (observation 3)
- Instead of explicitly maintaining f_x^{ub} for each candidate x, LARA reduces the computations by distributing the candidates using a lattice.

Example of LARA (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{lb} = 1.8$, $f_a^{lb} = 0.9$
- $W_k = \{c\}$, $t = 1.8$, $T=2.7$, $t<T$
- Still in growing phase

Example of LARA (k=1)

S1	S2	S3
🟩 c 0.9	🟩 a 0.9	🟩 c 0.9
🟩 d 0.8	🟩 b 0.8	🟩 a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{lb} = 1.8$, $f_a^{lb} = 1.8$, $f_d^{lb} = 0.8$, $f_b^{lb} = 0.8$
- $W_k = \{c\}$, $t = 1.8$, $T=2.5$, $t<T$
- Still in growing phase

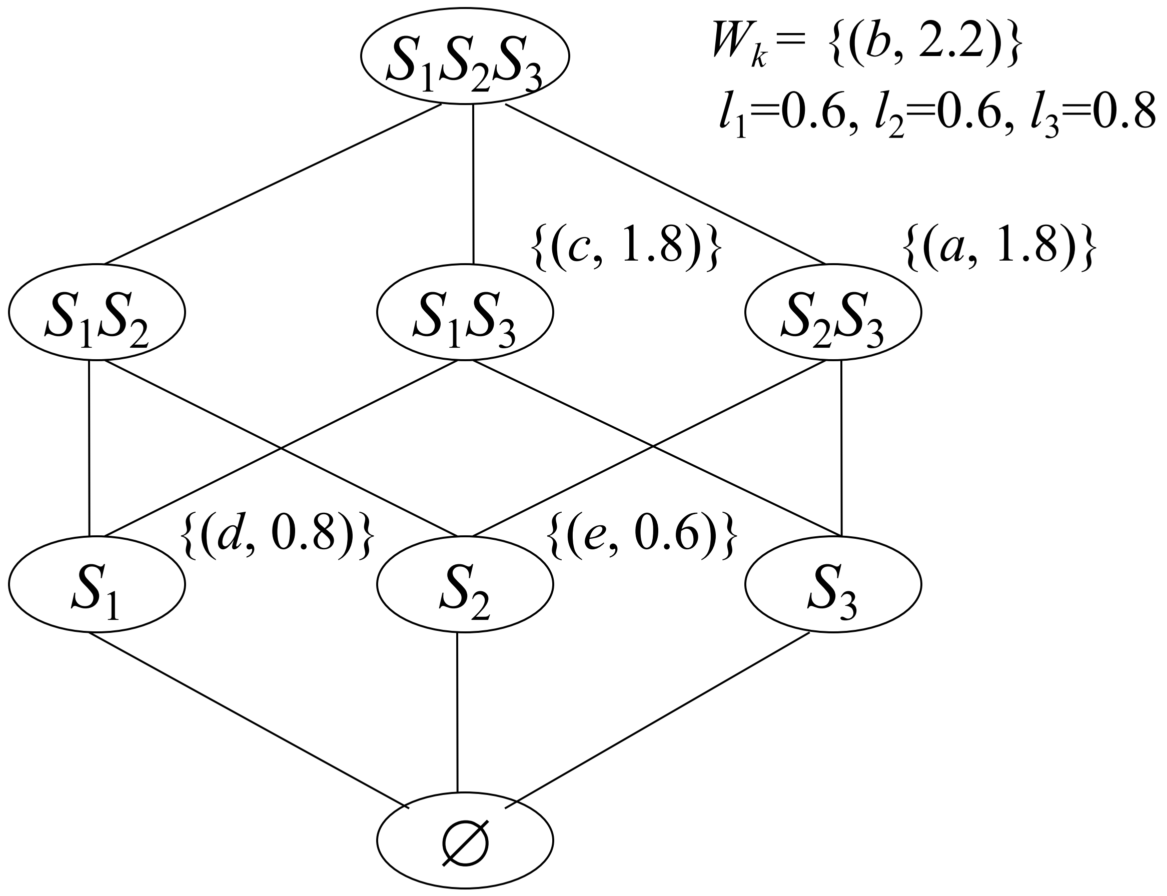
Example of LARA (k=1)

S1	S2	S3
🟩 c 0.9	🟩 a 0.9	🟩 c 0.9
🟩 d 0.8	🟩 b 0.8	🟩 a 0.9
🟩 b 0.6	🟩 e 0.6	🟩 b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

🟩 accessed data

- $f_c^{lb} = 1.8$, $f_a^{lb} = 1.8$, $f_d^{lb} = 0.8$, $f_b^{lb} = 2.2$, $f_e^{lb} = 0.6$
- $W_k = \{b\}$, $t = 2.2$, $T=2.0$, $t \geq T$
- Entering shrinking phase

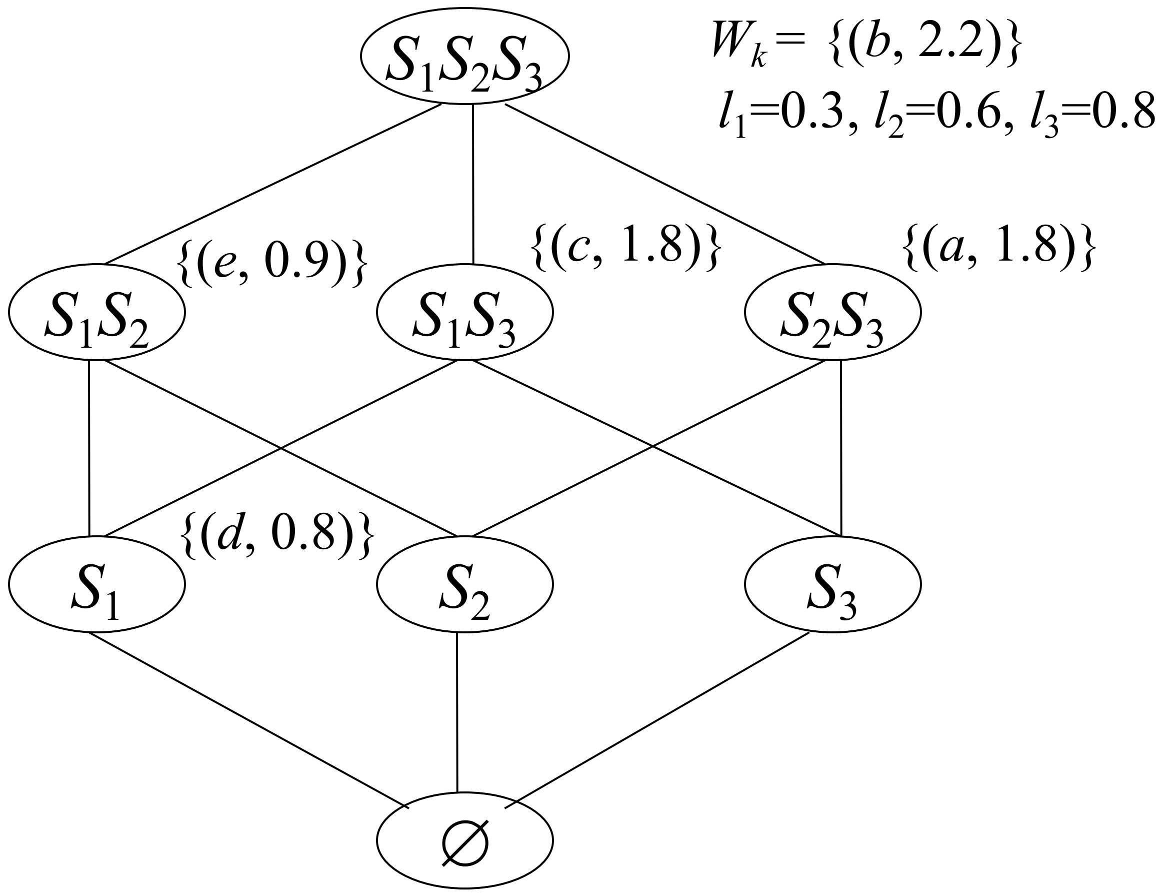
Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

$f_c^{ub} = 2.4$ is the greatest upper bound

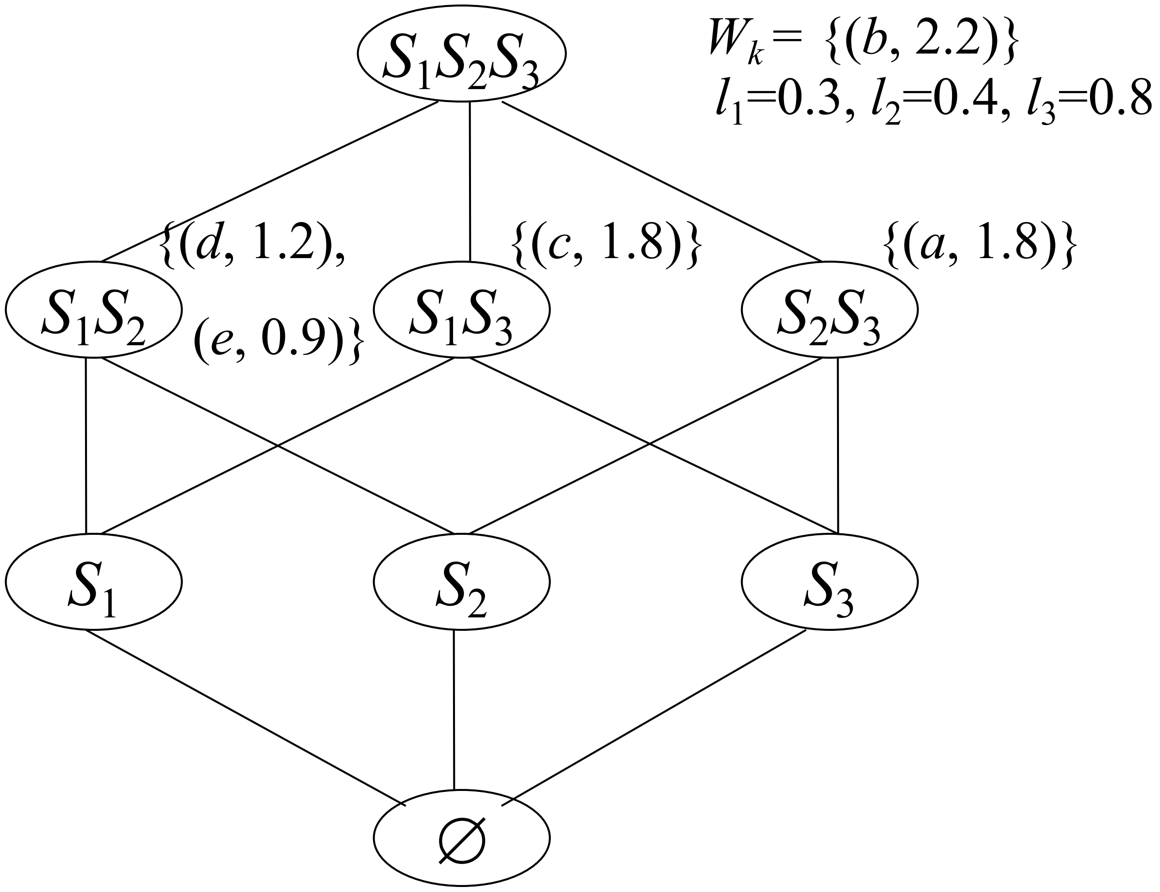
Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

$f_c^{ub} = 2.4$ is the greatest upper bound

Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

f_c^{ub} = 2.2 is the greatest upper bound → LARA stops!

LARA: Computational cost analysis

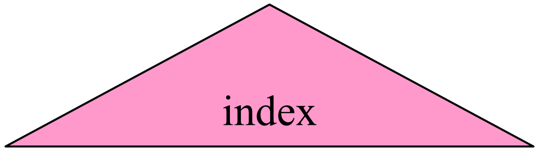
- Growing phase:
  - $O(\log k)$ per access
- Shrinking phase:
  - $O(2^m+\log k)$ per access
- Computational cost is much lower than the $O(n)$ cost of NRA

Issues in Top-k Rank Aggregation

- Is round-robin the best access order?
  - Quick-Combine is a variant of TA, which accesses the source with the largest decrease rate in score
  - this way, T is reduced faster and the algorithm terminates earlier
- What if some inputs are not sorted?
  - Sorted accesses are only applied in sorted inputs
  - Random accesses are applied in remaining inputs
  - Objective is to minimize cost, assuming that the cost of sorted/random accesses to different inputs varies

Indexing for top-k queries

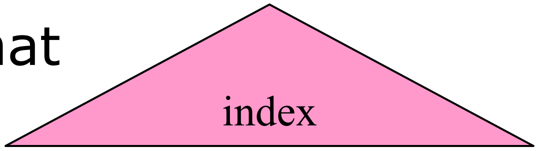
- Solutions based on indexing or materialized views
- Construct an index for the data or pre-compute some information, which is used for top-k query evaluation



S1 (price)	S2 (quality)	S3 (dist. to hotel)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

Indexing for top-k queries

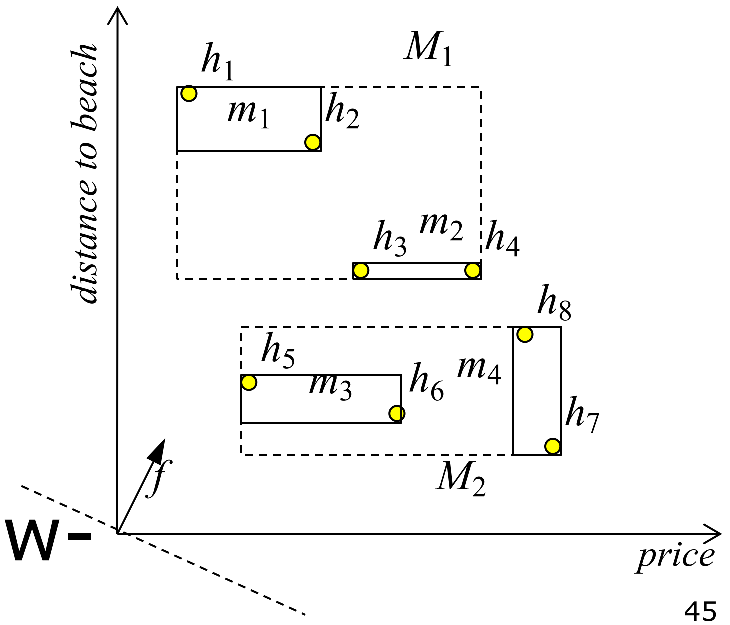
- Advantages:
  - more efficient than rank-aggregation methods
  - apply directly on data in record format
  - can make use of general-purpose multi-dimensional indexes
- Drawbacks
  - may not be applicable for top-k queries in arbitrary subspaces
  - hard to apply on distributed data
  - cannot be used on top of other query operations
  - some methods have high preprocessing and storage requirements



S1 (price)	S2 (quality)	S3 (dist. to hotel)
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

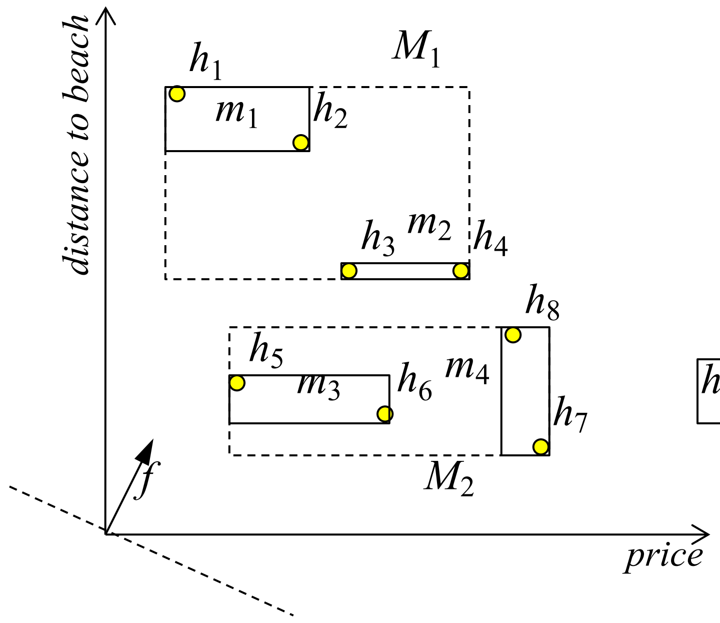
Multi-dimensional index search

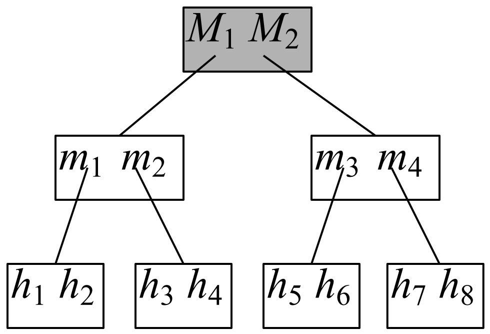
- Each attribute a dimension, each tuple a multidimensional point
- Aggregate function is a geometric shape "sweeping" the space from the (0,0,…,0) point
- First k points hit by function are top-k results
- Branch-and-bound search can be applied on multi-d index, using low-most corners as bounds



Multi-dimensional index search

- Best-First search algorithm for top-k
  - f = ax+by
  - objective: find k hotels with min(f(h_i))
  - heap Q: prioritize based on {min(f(p)):p in M}

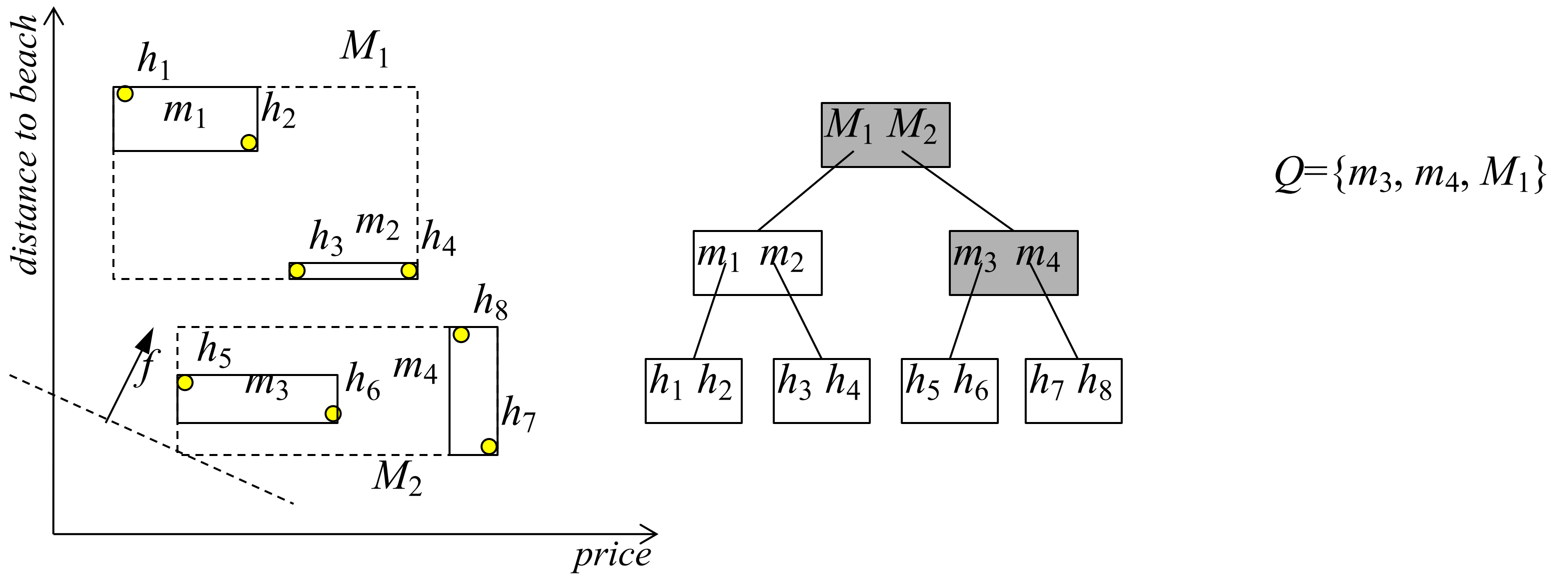




$Q=\{M_2, M_1\}$

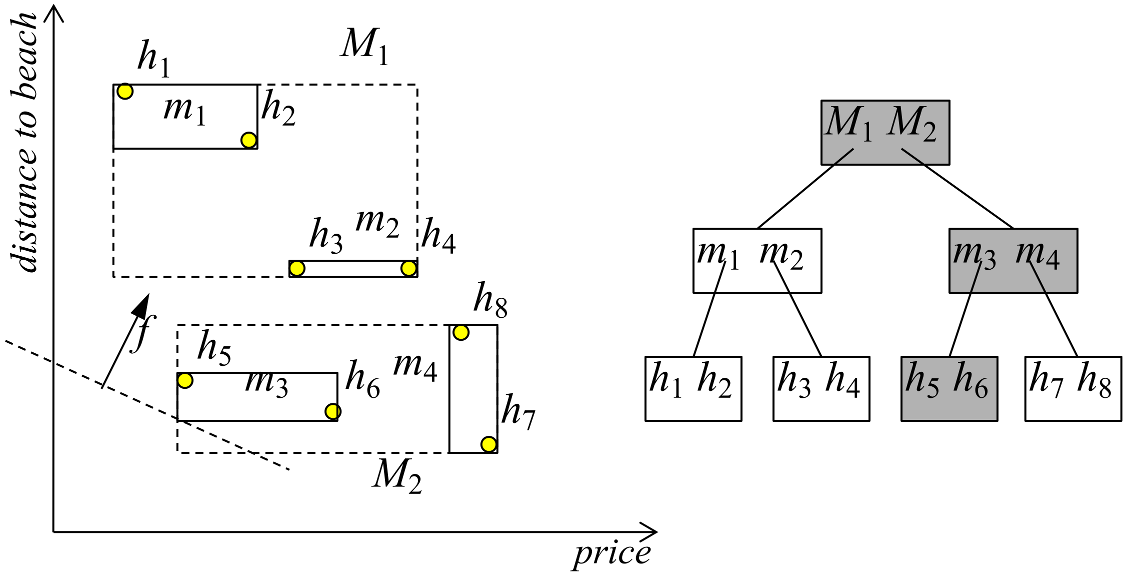
Multi-dimensional index search

- Best-First search algorithm for top-k
  - f = ax+by
  - objective: find k hotels with min(f(h_i))
  - heap Q: prioritize based on {min(f(p)):p in M}



Multi-dimensional index search

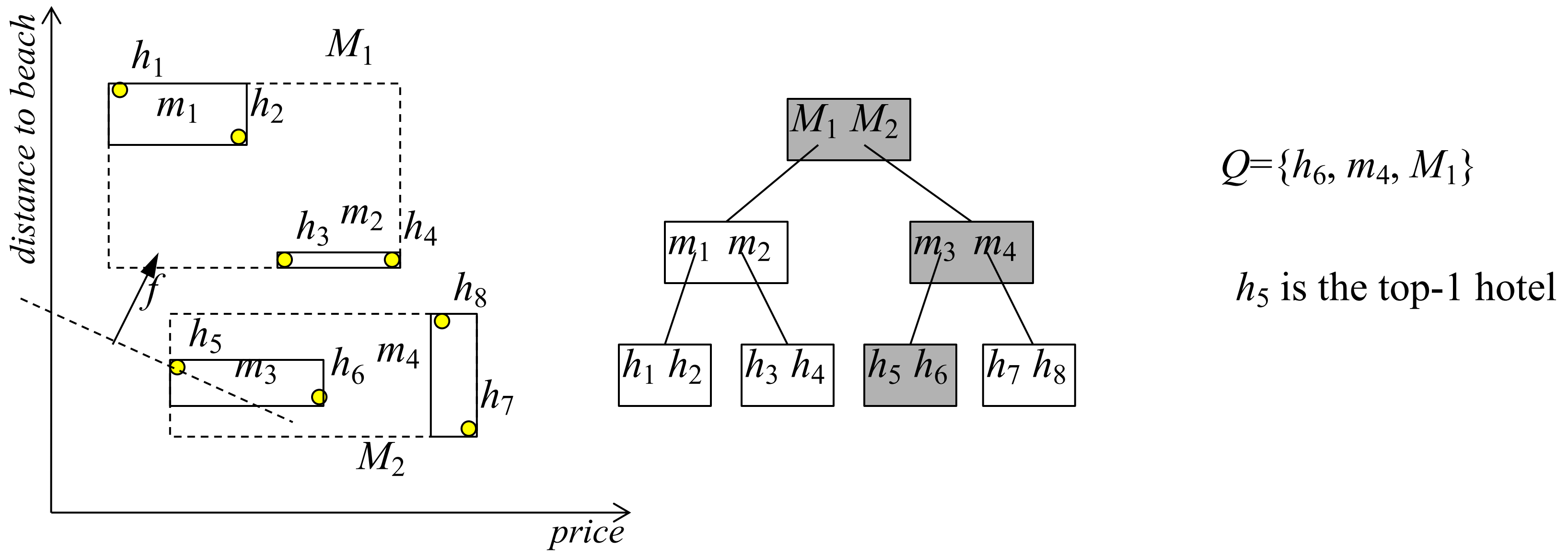
- Best-First search algorithm for top-k
  - f = ax+by
  - objective: find k hotels with min(f(h_i))
  - heap Q: prioritize based on {min(f(p)):p in M}



$Q=\{h_5, h_6, m_4, M_1\}$

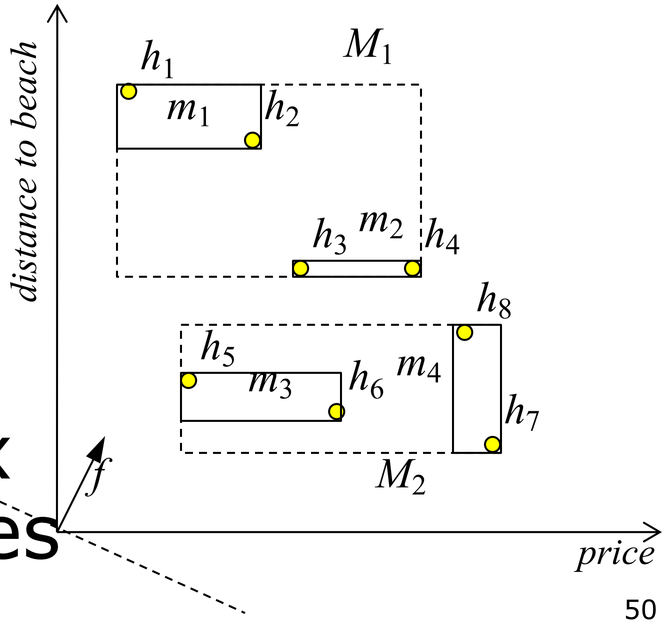
Multi-dimensional index search

- Best-First search algorithm for top-k
  - f = ax+by
  - objective: find k hotels with min(f(h_i))
  - heap Q: prioritize based on {min(f(p)):p in M}

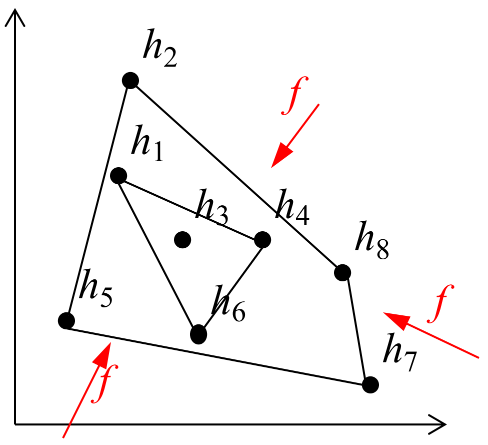


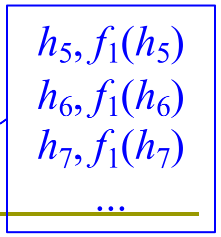
Multi-dimensional index search

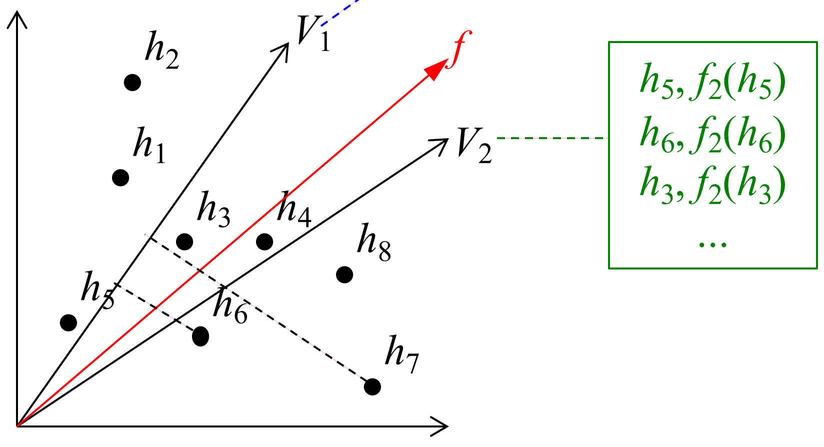
- Method is I/O optimal, given an R-tree that indexes the data
- Incrementally finds the top objects according to any monotone aggregate function
- Relies on existence of the index for the set of attributes that are aggregated by f
- Can also be applied if index is on a superset of attributes
  - but not as efficient



Other index/view methods







ONION: indexing convex hulls

- top-1 must be in outer layer
- top-k in at most k layers

disadvantages: slow in high-dimensional spaces, works for linear functions only, shares problems of index-based methods

Materialized top-k views:

- cache results of previous top-k queries
- use materialized views in TA to replace dimension-wise rankings (e.g., f can be computed by linearly combining f_1 and f_2)

disadvantages: works for linear functions only, hard to determine best views to use for a query

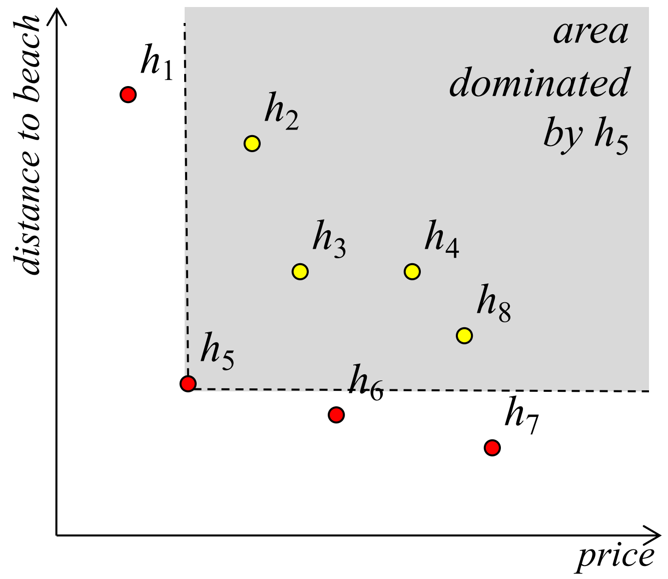
Skyline Queries

Skyline Queries

- Definition
- Skyline evaluation methods
  - Direct computation methods
  - Index-based methods
- Skyline variants

Skyline Queries

- In multi-dimensional spaces with ordinal attributes
- A point p dominates another point p′ if it is better than p′ w.r.t. at least one attribute and no worse than p′ w.r.t. the remaining attributes
- The skyline is the subset of points not dominated by others
    - Top-1 point for any monotone function must belong to the skyline



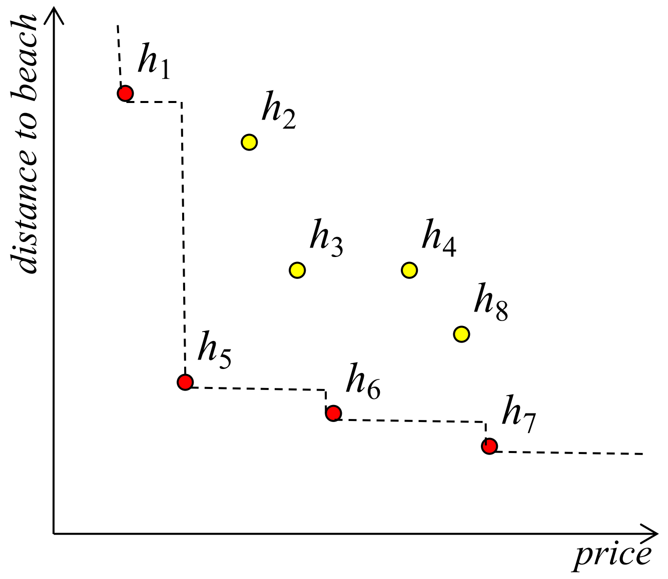
Skyline Queries

- Name comes from skyline
- A building appears in a skyline if it is not dominated by another building (with the same x-position) with respect to height and distance to coastline

Skyline Queries

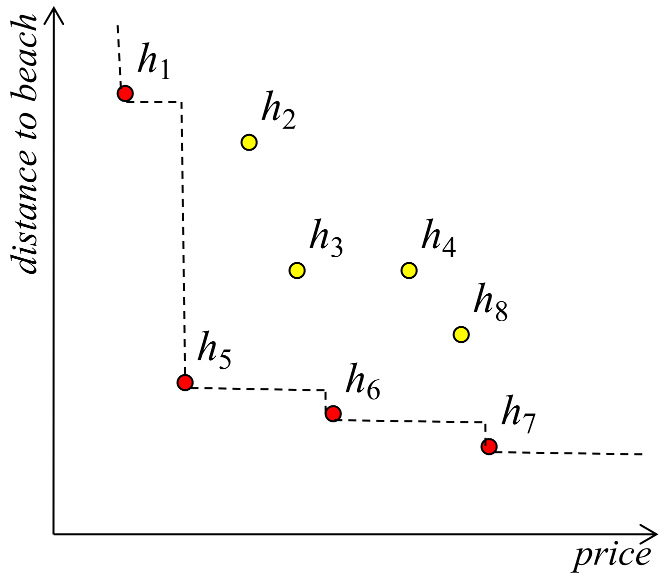
- In SQL:

```
SELECT *
FROM Hotels
WHERE city = ’Miami’
SKYLINE OF price MIN, distance_to_beach MIN;
```



Skyline Queries: Applications

- Advantage: User can find the best objects according to some criteria, without specifying parameters (ranking function, k)
- Caching top-k results for all functions
  - k-skyband

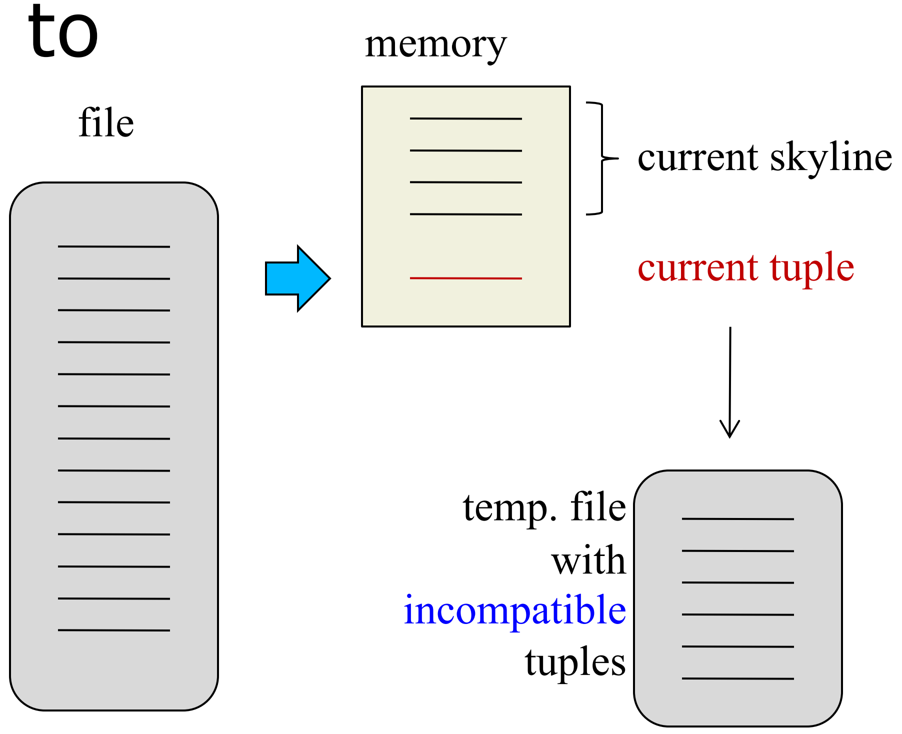


Skyline Queries: Evaluation

- Categories of skyline computation methods:
  - methods that apply on raw data (no index exists for them)
  - methods that apply on a multi-dimensional index (e.g., R-tree)

Skyline evaluation: Block-nested loops on raw data

- Operates on unordered, non-indexed data.
- Read file and maintain as many skyline points in memory as possible
- Use timestamps to compare and prune
- Additional passes to temp. file as necessary



Skyline evaluation: Block-nested loops on raw data

- F=input file, W=initially empty memory window
- While ((o=getNext(F))!=NULL)
  - compare o with all skyline objects in W
  - if o is dominated by any object in W
    - prune o (cannot be in skyline)
  - else
    - prune from W any objects dominated by o
    - add o in W
    - if no space for o in W
      - write o to temporary file T
- if T=empty, then output W as the skyline S
- else
  - S = S + all objects in W with timestamp smaller than first object written to T
  - set F=T and restart

Skyline evaluation: Block-nested loops on raw data

W

h_1, 20, 100
h_2, 50, 90
h_5, 40, 40

~~h_2, 50, 90~~
~~h_8, 100, 50~~

T

h_3, 60, 60
h_4, 80, 60
h_6, 80, 30
h_7, 120, 20

F

hotel	**price**	**distance**
h_1	20	100
h_2	50	90
h_3	60	60
h_4	80	60
h_5	40	40
h_6	80	30
h_7	120	20
h_8	100	50

S = {h_1}

Skyline evaluation: Block-nested loops on raw data

W

h_7, 120, 20
h_6, 80, 30

~~h_3, 60, 60~~

~~h_4, 80, 60~~

T

F

hotel	price	distance
h_3	60	60
h_4	80	60
h_6	80	30
h_7	120	20

S = $\{h_1\}$

=> S = $\{h_1,h_5\}$ (h_5 already checked with h_i, $i>5$)

=> S = $\{h_1,h_5,h_6,h_7\}$ (after adding W)

Skyline evaluation: Block-nested loops on raw data

- Notes on BNL
  - Can do multiple passes over the data
    - for high dimensional data the domination probability is low
  - Works well if skyline is small
    - if S smaller than window size only one loop is needed
  - timestamps are used to avoid checking pairs twice in multiple rounds and release tuples from W
  - can organize tuples in W well to optimize checking each new tuple against all tuples in W
    - e.g., put tuples with high pruning power in the beginning of W

Skyline evaluation: Sort-Filter-Skyline on raw data

- Similar to BNL
- Before BNL, sort objects such that if o_i before o_j in the sorted order, o_j cannot dominate o_i
  - example sorting: sum of all attribute values
- Sorting ensures that an object is in the skyline as soon as it enters W
  - complete W is guaranteed to be in S after each pass

Skyline evaluation: Sort-Filter-Skyline on raw data

first pass

W	T
h_5, 40, 40 h_6, 80, 30	h_1, 20, 100 h_7, 120, 20

S = {h_5,h_6}

second pass

W
h_1, 20, 100 h_7, 120, 20

S = {h_5,h_6,h_1,h_7}

sorted F

hotel	price	distance
h_5	40	40
h_6	80	30
h_1	20	100
~~h_3~~	~~60~~	~~60~~
~~h_2~~	~~50~~	~~90~~
~~h_4~~	~~80~~	~~60~~
h_7	120	20
~~h_8~~	~~100~~	~~50~~

Skyline evaluation: Other methods on raw data

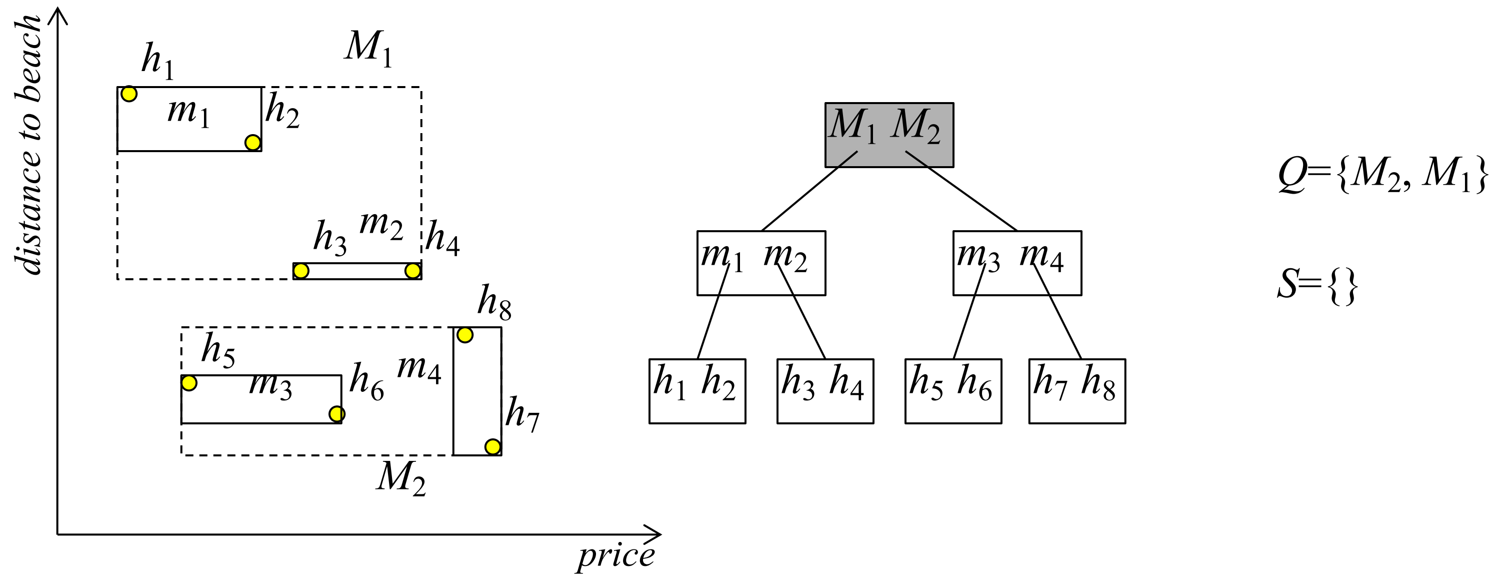
- Divide and Conquer
  - partition the data recursively by taking medians in alternate dimensions
    - first partition based on m_p in dimension d_p
    - then, each partition on m_g in dimension d_g
    - …
  - if partitions become small enough, compute skyline S_i in each partition P_i
  - hierarchically merge skylines of partitions, eliminating objects that are dominated
- worst-case optimal, but slower than BNL on typical data

Skyline evaluation: Index-based evaluation

- Branch-and-bound Skyline (BBS)
  - Operates on multi-dimensional indexes (like R-tree)
  - Extends best-first algorithm for NN search
  - Points are retrieved in increasing distance from origin point (0,0,…,0)
  - Points or entries (sub-trees) that are dominated by found skyline points are pruned
  - Skyline points are buffered to prune entries later or output as soon as no other entries can be pruned by them
  - I/O optimal, given the index

Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S

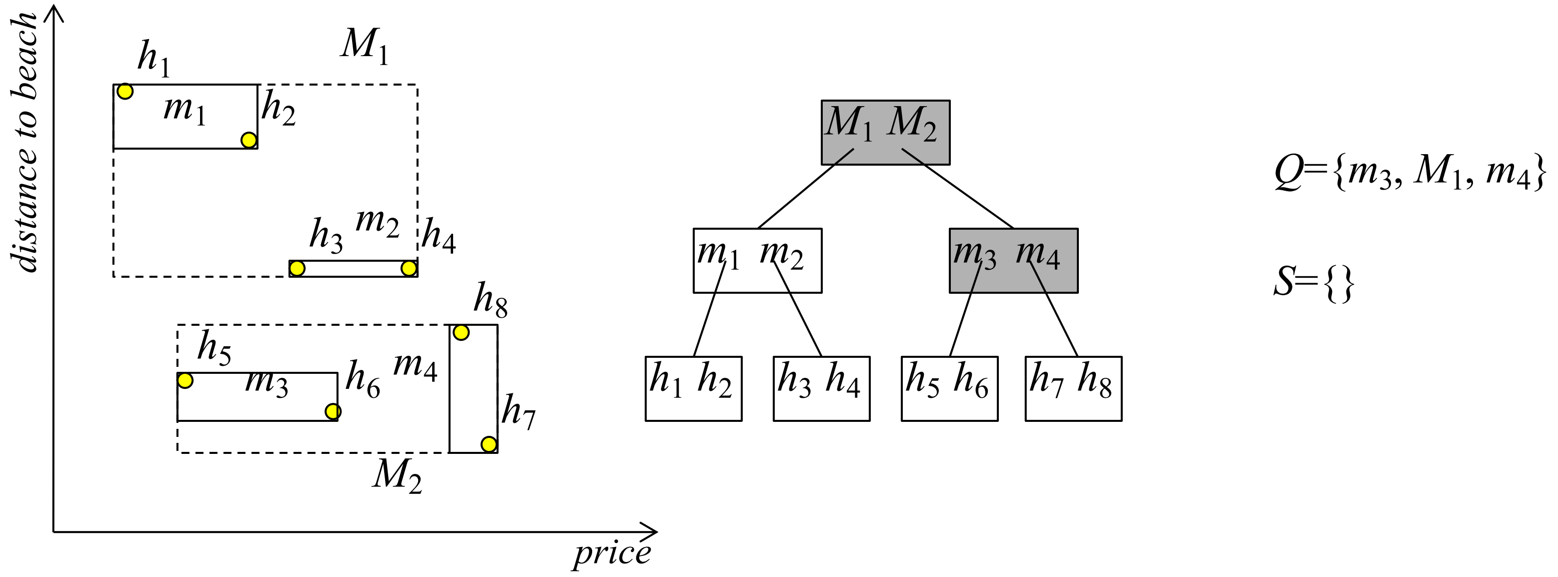


$Q=\{M_2, M_1\}$

$S=\{\}$

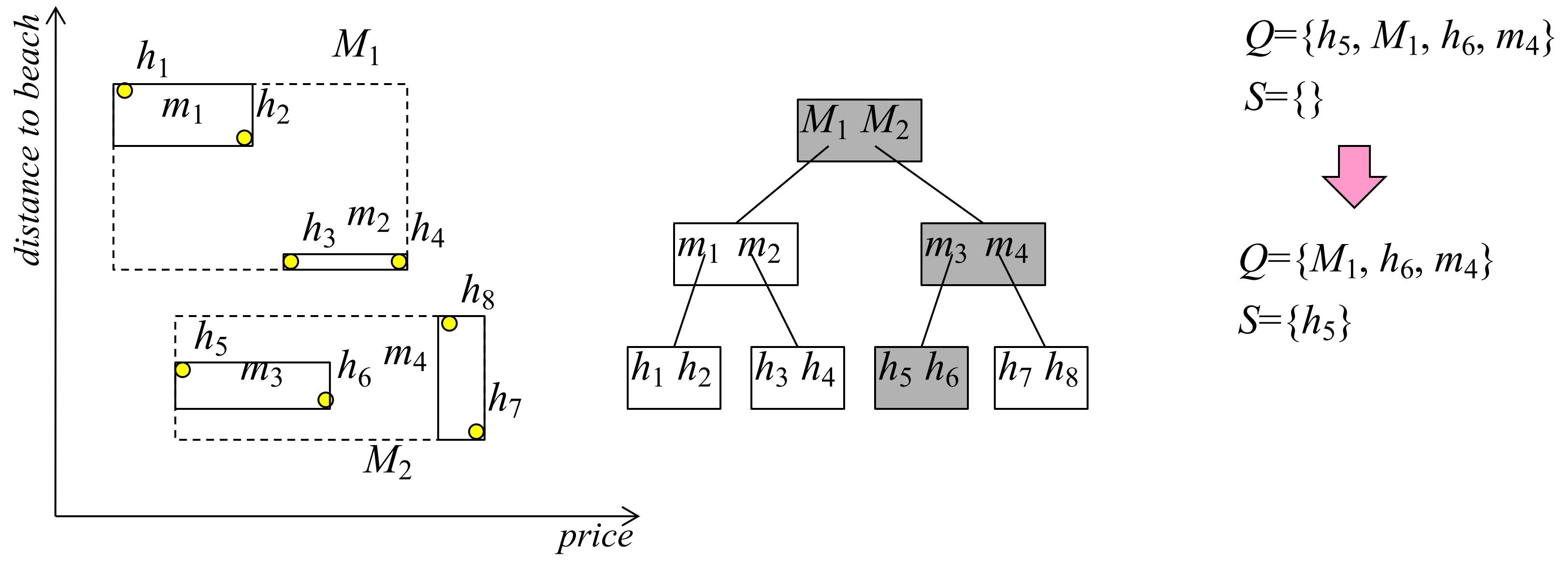
Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S



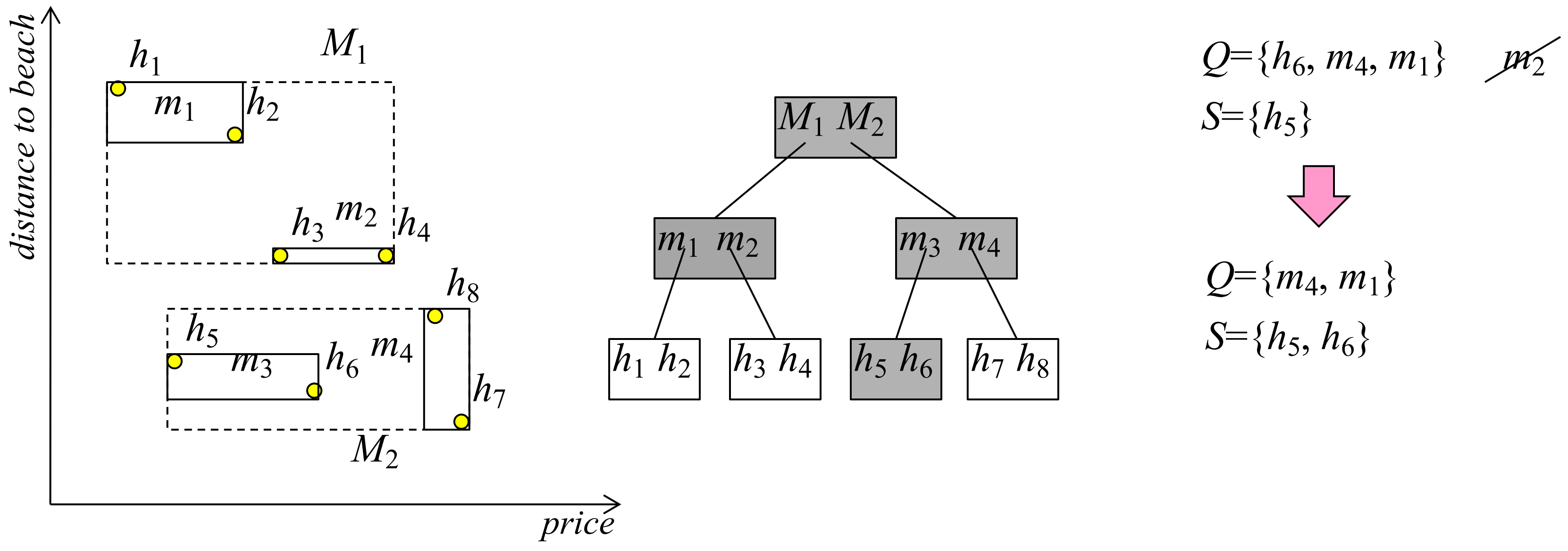
Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S



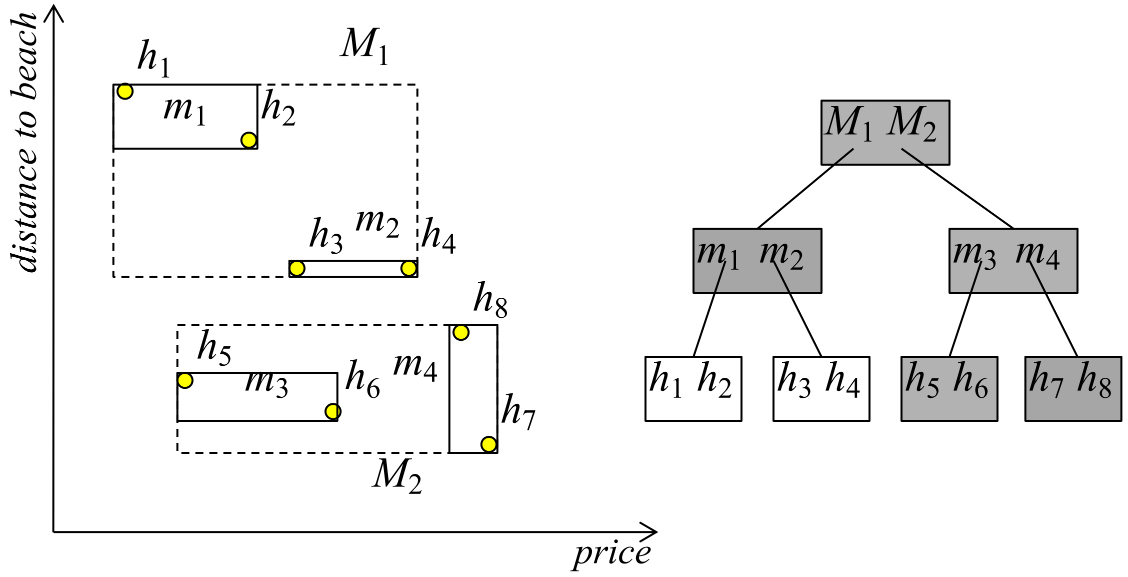
Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S



Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S

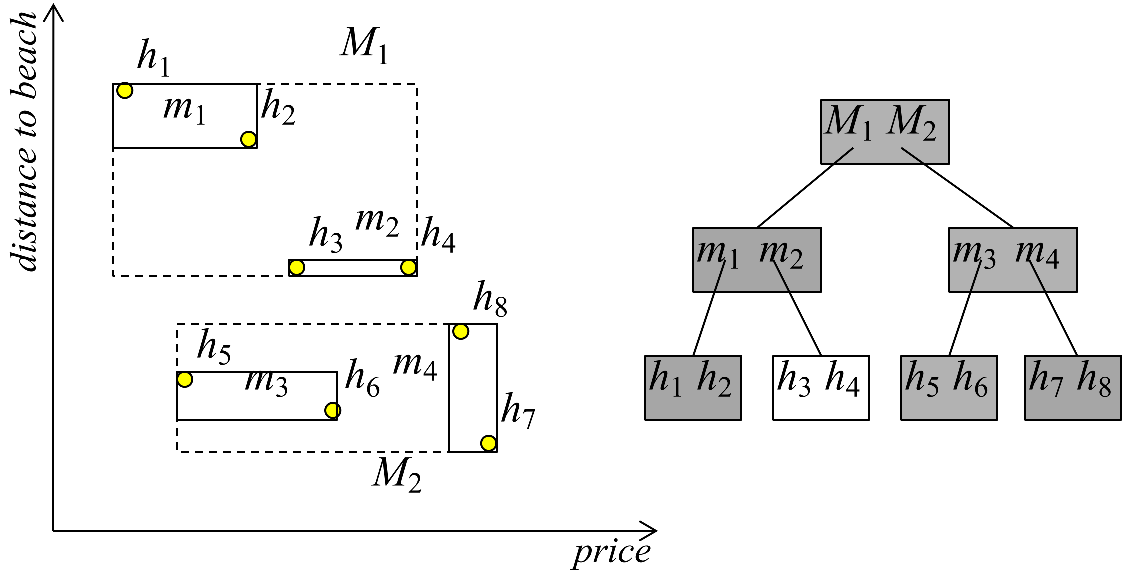


$Q=\{m_1, h_7\}$ ~~h_8~~

$S=\{h_5, h_6\}$

Skyline evaluation: Branch-and-Bound Skyline (BBS)

- Best-First search algorithm for top-k
  - access entries in increasing distance to (0,0)
  - prune dominated entries by some point in S
  - unpruned visited points guaranteed to be in S



$Q=\{h_7, h_1\}$ ~~h_2~~
$S=\{h_5, h_6\}$

$\Downarrow$

$Q=\{h_1\}$
$S=\{h_5, h_6, h_7\}$

$\Downarrow$

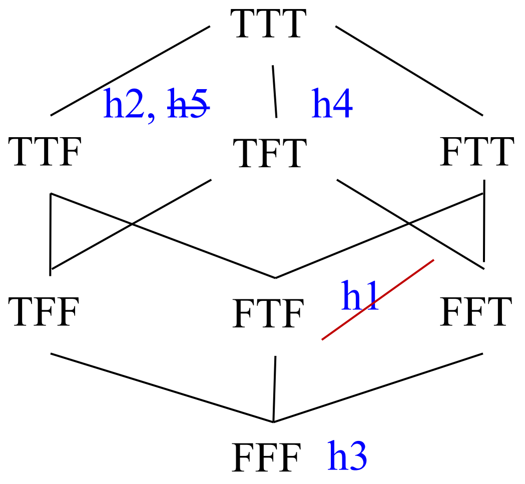
$Q=\{\}$
$S=\{h_1, h_5, h_6, h_7\}$

Skyline evaluation: Skylines for low-cardinality domains

1. Build lattice based on low-cardinality domains
2. Objects in the same node are compared based on non-lattice attributes
3. Dominating-Dominated Pairs of nodes are compared

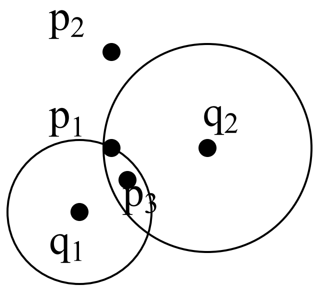
low-cardinality attributes

hotel	parking	pool	gym	price
h1	F	T	F	100
h2	T	T	F	90
h3	F	F	F	60
h4	T	F	T	120
h5	T	T	F	110



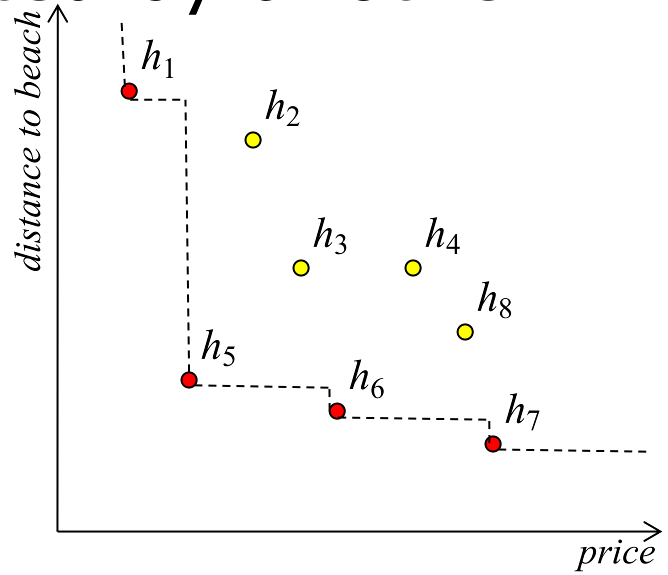
Skyline variants: Spatial skyline

- Input: set of query points Q, set P
- p_i dominates p_j, if $dist(p_i,q) \leq dist(p_j,q)$ for all q in Q and there exists a q in Q, such that $dist(p_i,q) < dist(p_j,q)$
  - p_1 dominates p_2, p_3 dominates p_1
- Skyline should contain:
  - the NN of each q
  - any point in the convex hull of Q
  - skyline depends only on CH(Q)
  - points whose VoronoiCell(p) intersects CH(Q)
- Algorithms extend BBS, or traverse the Voronoi diagram



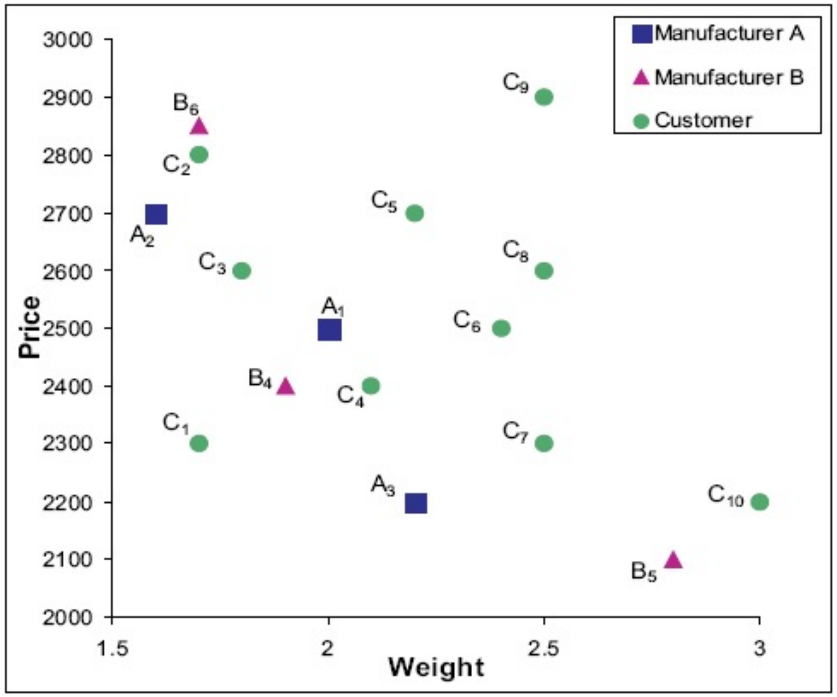
Skyline variants: Skylines in subspaces

- An object may be in the skyline considering all dimensions, but not part of the skyline if we consider a subspace of dimensions
- Reversely, an object in the skyline of a subspace may be dominated by another in a superspace
- Q: how to find the skyline points and their decisive subspaces
- Q: index to support subspace skyline queries



Skyline variants: Domination Relationship Analysis

- Products + requirements of customers
- Product is good if it dominates many customers and dominated by few
- Analysis queries:
  - Lin. optim. query
    - Make laptop lighter & more expensive better?
  - Subspace analysis
    - For which subspaces is product better?
  - Comparative analysis
    - Which manufacturer is better?

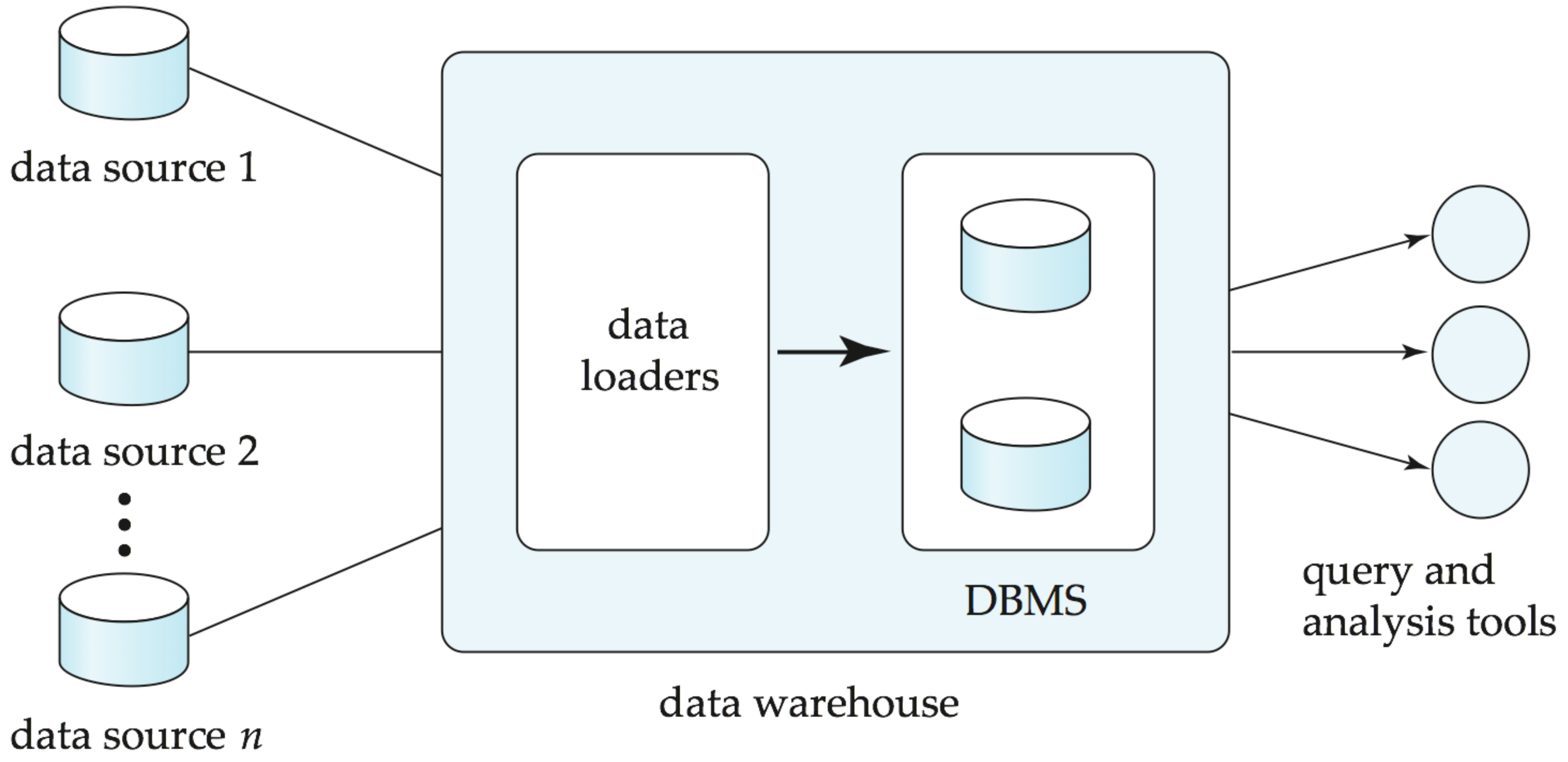


Data Warehousing and OLAP

Data Warehousing

- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
  - Greatly simplifies querying, permits study of historical trends
  - Shifts decision support query load away from transaction processing systems

Data Warehousing



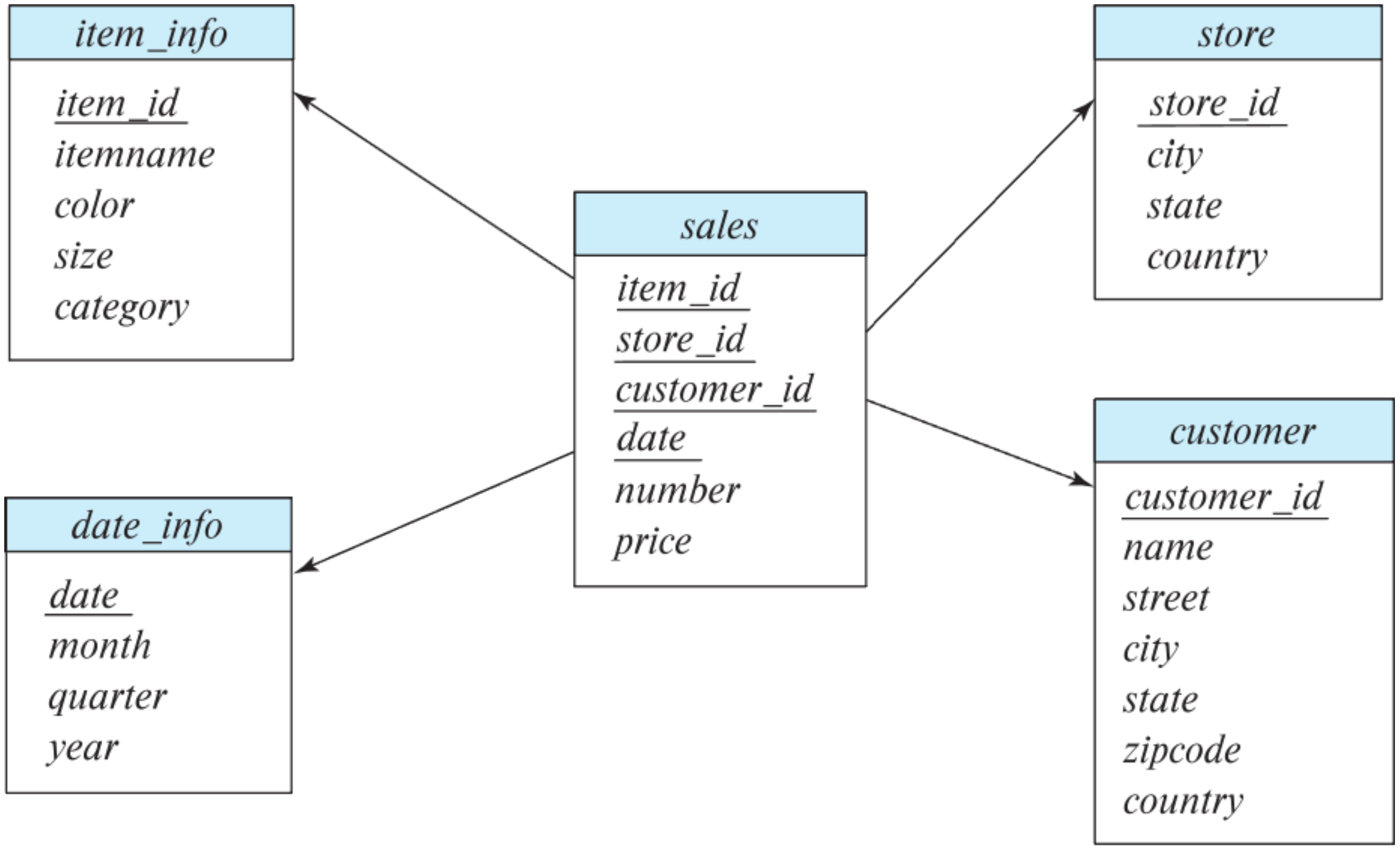
Warehouse Design Issues

- **Data transformation** and **data cleansing**
  - E.g., correct mistakes in addresses (misspellings, zip code errors)
  - Merge address lists from different sources and purge duplicates
- *How to propagate updates*
  - Warehouse schema may be a (materialized) view of schema from data sources
    - View maintenance
- *What data to summarize*
  - Raw data may be too large to store on-line
  - Aggregate values (totals/subtotals) often suffice
  - Queries on raw data can often be transformed by query optimizer to use aggregate values

Multidimensional Data and Warehouse Schemas

- Data in warehouses can usually be divided into
  - **Fact tables**, which are large
    - E.g, *sales*(*item_id, store_id, customer_id, date, number, price*)
  - **Dimension tables**, which are relatively small
    - Store extra information about stores, items, etc.
- Attributes of fact tables can be usually viewed as
  - **Measure attributes**
    - measure some value, and can be aggregated upon
    - e.g., the attributes *number* or *price* of the *sales* relation
  - **Dimension attributes**
    - dimensions on which measure attributes are viewed
    - e.g., attributes *item_id, color,* and *size* of the *sales* relation
    - Usually small ids that are foreign keys to dimension tables

Data Warehouse Schema



Multidimensional Data and Warehouse Schemas

- Resultant schema is called a **star schema**
  - More complicated schema structures
    - **Snowflake schema**: multiple levels of dimension tables
    - May have multiple fact tables
- Typically
  - fact table joined with dimension tables and then
  - group-by on dimension table attributes, and then
  - aggregation on measure attributes of fact table
- Some applications do not find it worthwhile to bring data to a common schema
  - **Data lakes** are repositories which allow data to be stored in multiple formats, without schema integration
  - Less upfront effort, but more effort during querying

Database Support for Data Warehouses

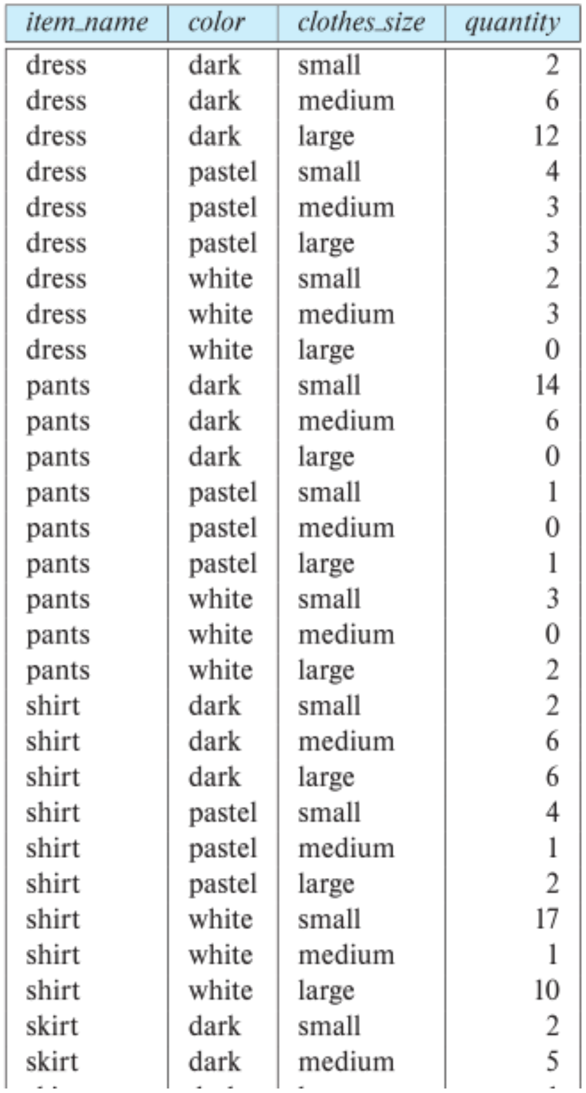
- Data in warehouses usually append only, not updated
  - Can avoid concurrency control overheads
- Data warehouses often use **column-oriented storage**
  - E.g., a sequence of *sales* tuples is stored as follows
    - Values of item_id attribute are stored as an array
    - Values of store_id attribute are stored as an array,
    - And so on
  - Arrays are compressed, reducing storage, IO and memory costs significantly
  - Queries can fetch only attributes that they care about, reducing IO and memory cost
- Data warehouses often use parallel storage and query processing infrastructure
  - Distributed file systems, Map-Reduce, Hive, …

Data Analysis and OLAP

- **Online Analytical Processing (OLAP)**
  - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- We use the following relation to illustrate OLAP concepts
  - *sales* (*item_name*, *color*, *clothes_size*, *quantity*)

  This is a simplified version of the *sales* fact table joined with the dimension tables, and many attributes removed (and some renamed)

Example sales relation



item_name	*color*	*clothes_size*	*quantity*
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
pants	dark	small	14
pants	dark	medium	6
pants	dark	large	0
pants	pastel	small	1
pants	pastel	medium	0
pants	pastel	large	1
pants	white	small	3
pants	white	medium	0
pants	white	large	2
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
skirt	dark	small	2
skirt	dark	medium	5

Cross Tabulation of sales by item_name and color

clothes_size **all**

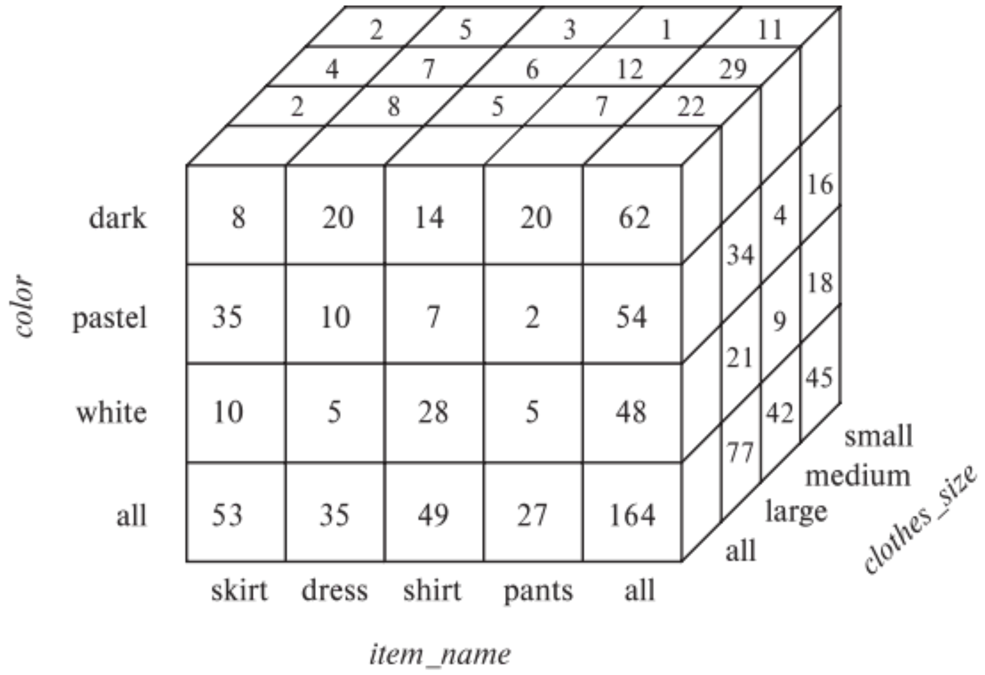
color

item_name	dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
  - Values for one of the dimension attributes form the row headers
  - Values for another dimension attribute form the column headers
  - Other dimension attributes are listed on top
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have n dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube



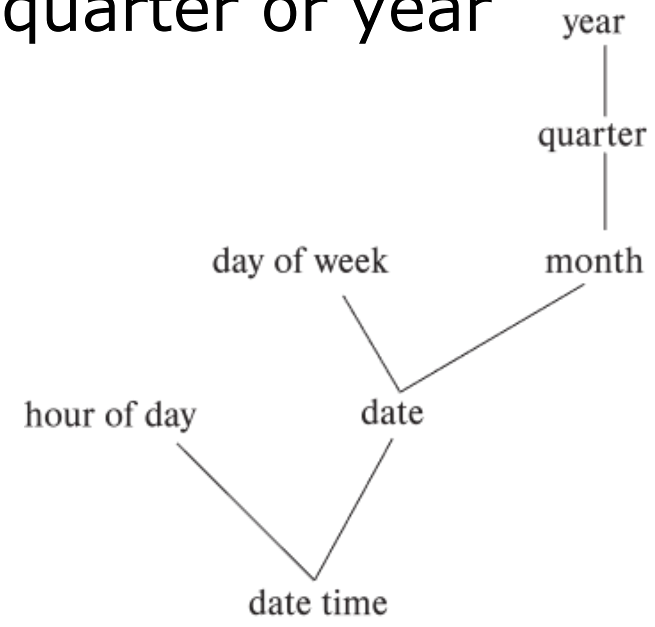
color \ item_name	skirt	dress	shirt	pants	all
dark	8	20	14	20	62
pastel	35	10	7	2	54
white	10	5	28	5	48
all	53	35	49	27	164

Online Analytical Processing Operations

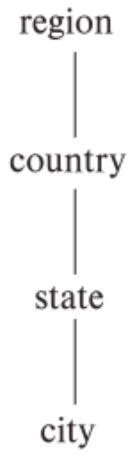
- **Pivoting:** changing the dimensions used in a cross-tab
  - E.g., moving colors to column names
- **Slicing:** creating a cross-tab for fixed values only
  - E.g., fixing color to white and size to small
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** moving from finer-granularity data to a coarser granularity
  - E.g., aggregating away an attribute
  - E.g., moving from aggregates by day to aggregates by month or year
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data

Hierarchies on Dimensions

- **Hierarchy** on dimension attributes: lets dimensions be viewed at different levels of detail
- E.g., the dimension *datetime* can be used to aggregate by hour of day, date, day of week, month, quarter or year



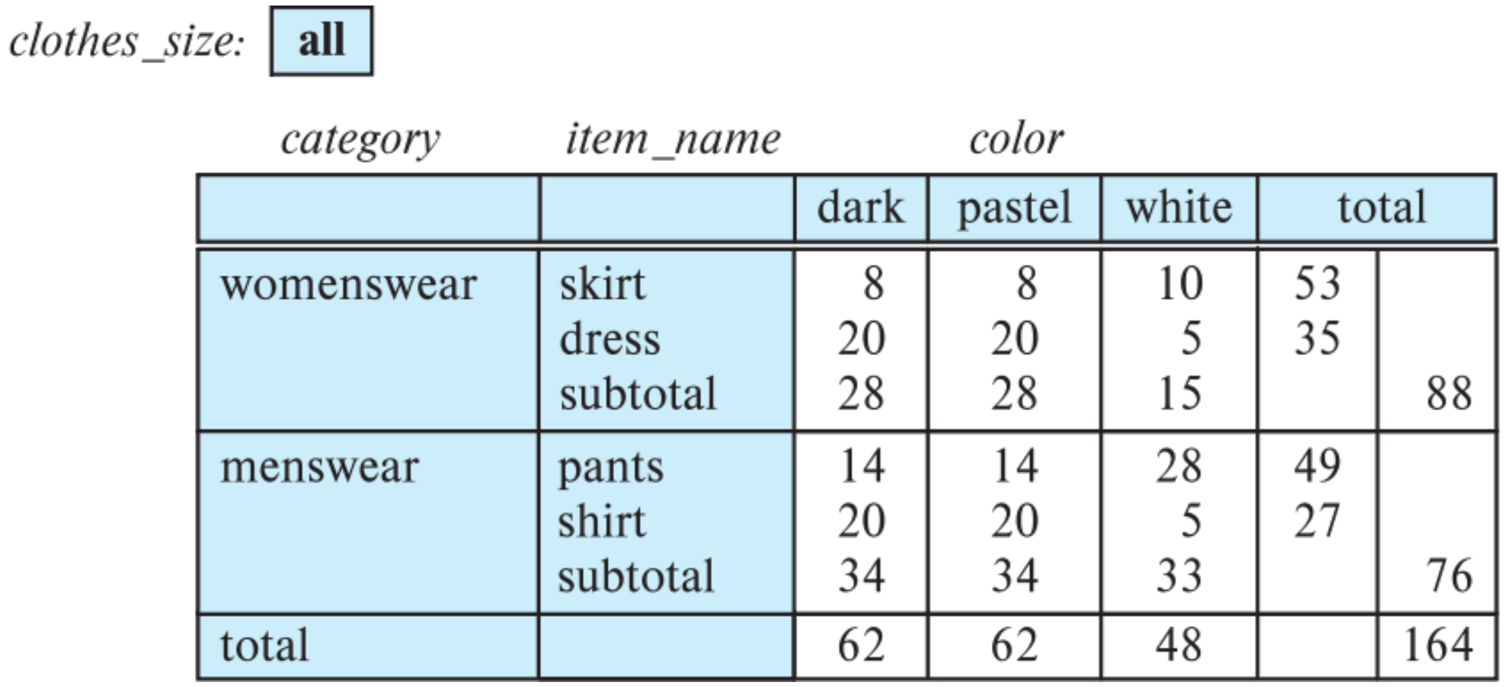
(a) time hierarchy



(b) location hierarchy

Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
- Can drill down or roll up on a hierarchy
- E.g. hierarchy: *item_name* → *category*



clothes_size: **all**

category	*item_name*	*color*				
		dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

OLAP Implementation

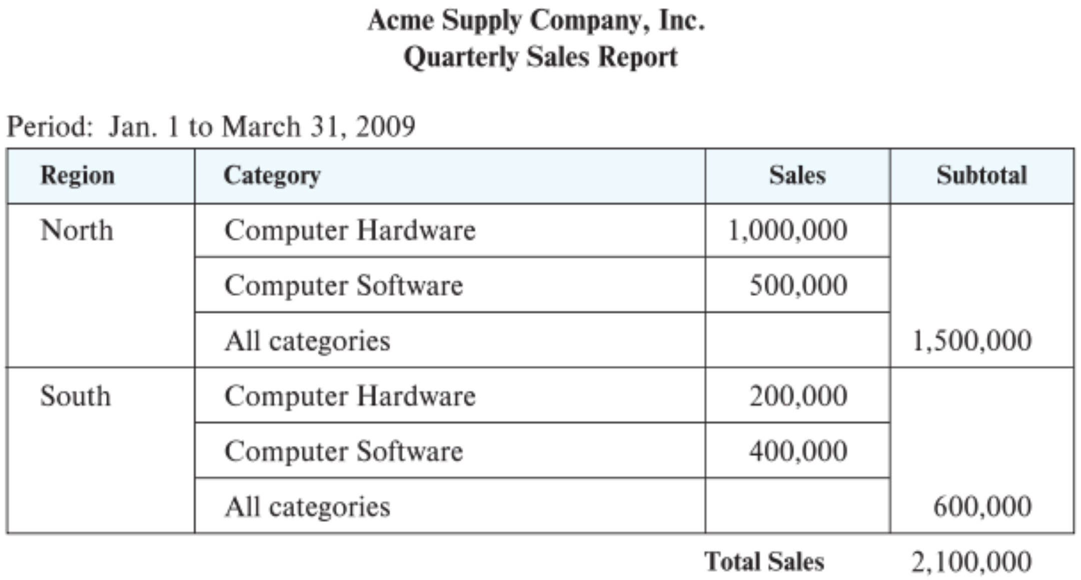
- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **relational OLAP (ROLAP)** systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **hybrid OLAP (HOLAP)** systems.

OLAP Implementation (Cont.)

- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
  - Space and time requirements for doing so can be very high
    - 2^n combinations of **group by**
  - It suffices to precompute some aggregates, and compute others on demand from one of the precomputed aggregates
    - Can compute aggregate on (*item_name, color*) from an aggregate on (*item_name, color, size*)
      - For all but a few "non-decomposable" aggregates such as *median*
      - is cheaper than computing it from scratch
- Several optimizations available for computing multiple aggregates
  - Can compute aggregate on (*item_name, color*) from an aggregate on (*item_name, color, size*)
  - Can compute aggregates on (*item_name, color, size*), (*item_name, color*) and (*item_name*) using a single sorting of the base data

Reporting and Visualization

- **Reporting tools** help create formatted reports with tabular/graphical representation of data
  - E.g., SQL Server reporting services, Crystal Reports
- **Data visualization** tools help create interactive visualization
  - E.g., Tableau, FusionChart, plotly, Datawrapper, Google Charts, etc.
  - Frontend typically based on HTML+JavaScript



Acme Supply Company, Inc.
Quarterly Sales Report

Period: Jan. 1 to March 31, 2009

Region	Category	Sales	Subtotal
North	Computer Hardware	1,000,000	
	Computer Software	500,000	
	All categories		1,500,000
South	Computer Hardware	200,000	
	Computer Software	400,000	
	All categories		600,000
		Total Sales	2,100,000

Temporal Databases and Time-travel Search

Temporal Data Management

- Temporal Data Models: extension of relational model by adding temporal attributes to each relation
- Modeling time in a Temporal DB
- Temporal Queries
- Temporal Indexing Methods and Query Processing

Temporal DBs – Motivation

- Conventional databases model reality at a point in time (at the “current” time)
  - They represent the state of an enterprise at a single moment of time
- Many applications need information about the past
  - Financial (payroll)
  - Medical (patient history)
  - Government
  - Versioned document collections (Wikipedia)
- Temporal DBMS: a system that manages the current and the past states of the data

Comparison

- Conventional DBs:
  - Evolve through transactions from one state to the next
  - Changes are viewed as modifications to the state
  - No information about the past
  - Snapshot of the enterprise
- Temporal DBs:
  - Maintain historical information
  - Changes are viewed as additions to the information stored in the database
  - Incorporate notion of time in the system
  - Efficient access to past states

Time In Databases

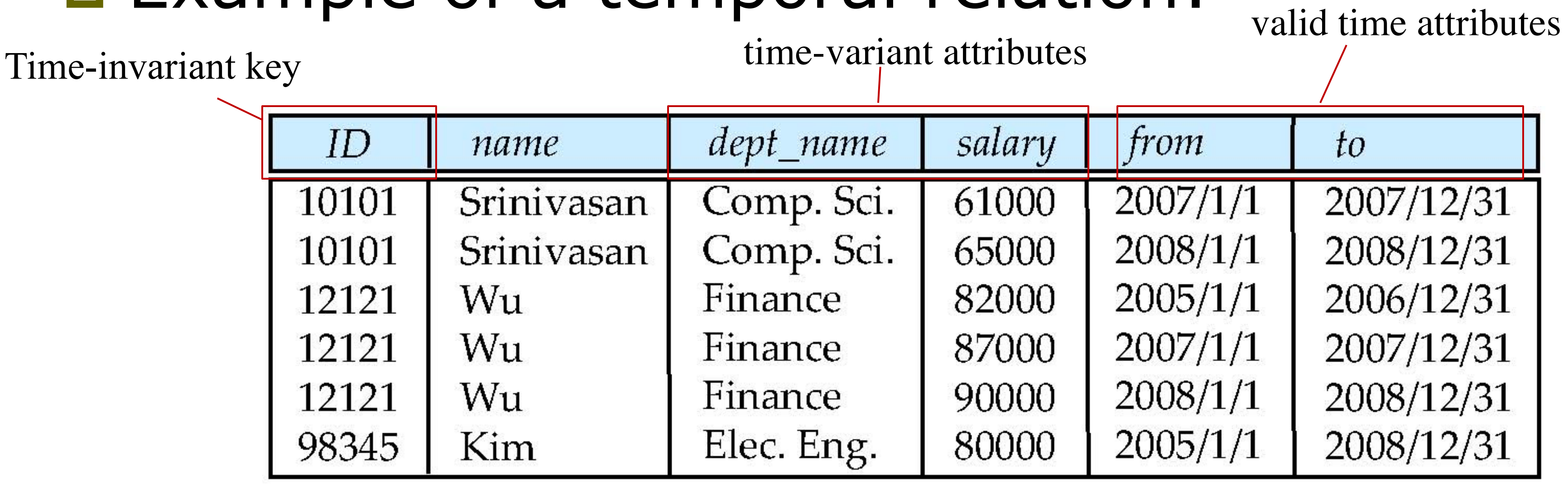
- Facts in temporal relations have associated times when they are *valid*, which can be represented as a union of intervals.
- The **transaction time** for a fact is the time interval during which the fact is current within the database system.
- In a **temporal relation**, each tuple has an associated time when it is true; the time may be either valid time or transaction time.
- A **bi-temporal relation** stores both valid and transaction time.

Example

- Sales example: data about sales are stored at the end of the day
  - Transaction time is different than valid time!
  - Valid time can refer to the future also!
    - Credit card: 03/01-04/06

Time In Databases (Cont.)

- Example of a temporal relation:


Time-invariant key
time-variant attributes
valid time attributes

ID	*name*	*dept_name*	*salary*	*from*	*to*
10101	Srinivasan	Comp. Sci.	61000	2007/1/1	2007/12/31
10101	Srinivasan	Comp. Sci.	65000	2008/1/1	2008/12/31
12121	Wu	Finance	82000	2005/1/1	2006/12/31
12121	Wu	Finance	87000	2007/1/1	2007/12/31
12121	Wu	Finance	90000	2008/1/1	2008/12/31
98345	Kim	Elec. Eng.	80000	2005/1/1	2008/12/31

- Temporal query languages have been proposed to simplify modeling of time as well as time related queries.
- Queries may refer to valid time, transaction time, or both

Examples of Temporal Queries

- Predicates *precedes, overlaps,* and *contains* on time intervals.
- *Intersect* can be applied on two intervals, to give a single (possibly empty) interval; the union of two intervals may or may not be a single interval.
- A **snapshot** of a temporal relation at time t consists of the tuples that are valid at time t, with the time-interval attributes projected out.
- **Temporal selection**: involves time attributes
- **Temporal projection**: the tuples in the projection inherit their time-intervals from the tuples in the original relation.
- **Temporal join**: the time-interval of a tuple in the result is the intersection of the time-intervals of the tuples from which it is derived. If intersection is empty, tuple is discarded from join.

Transaction Time DBs

- Time evolves discretely, usually is associated with the transaction number:

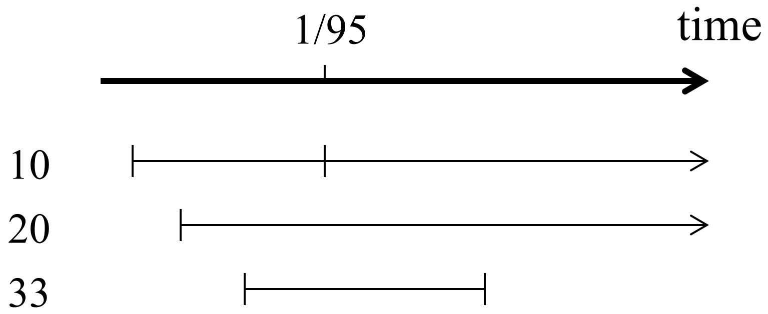
  $$T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \ldots.$$

- A record R is extended with an interval [t.start, t.end). When we insert an object at t1 the temporal attributes are updated to become [t1, now)
- Updates can be made only to the current state!
  - Past cannot be changed

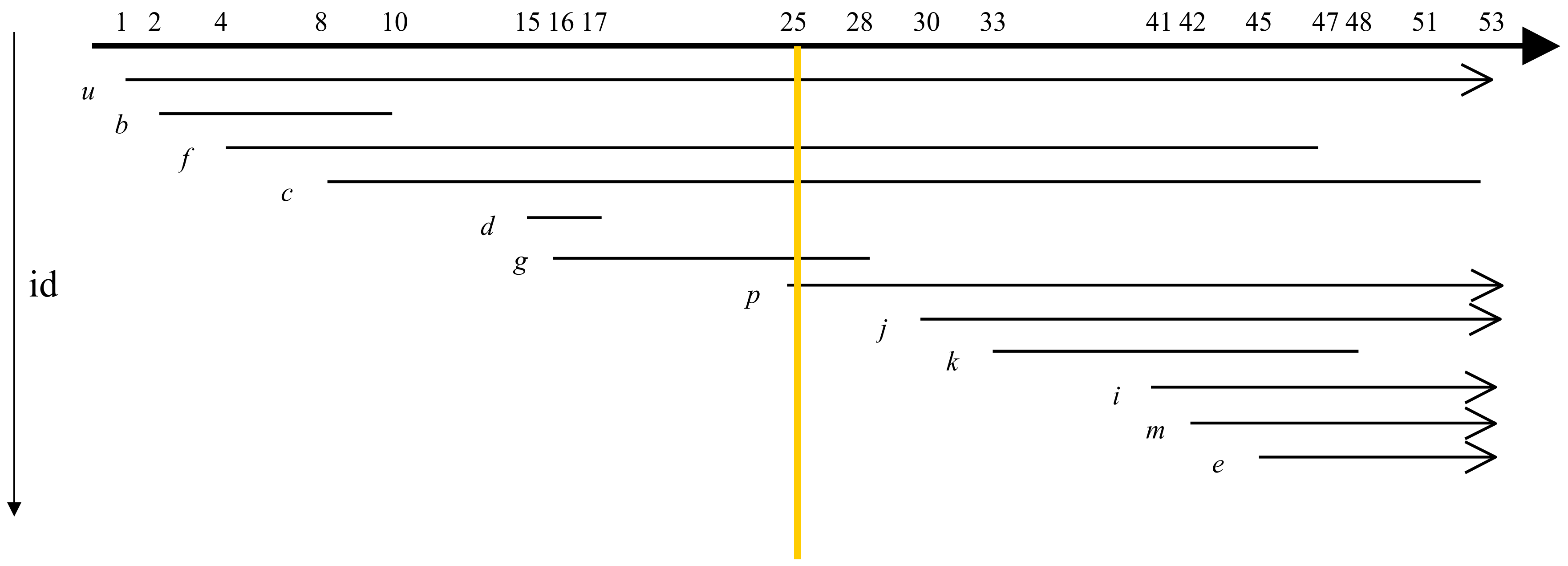
Transaction Time DBs

- Deletion is <u>logical</u> (never physical deletions!)
  - When an object is deleted at t2, its temporal attribute changes from [t1, now) → [t1, t2) (i.e. it updates its interval)
  - Object is "alive" from insertion to deletion time, ex. t1 to t2. If the value is "now" then the object is still alive

eid	salary	start	end
10	20K	9/93	1/95
20	50K	4/94	*
33	30K	5/94	6/95
10	50K	1/95	*



Transaction Time DBs



Database evolves through insertions and deletions

Transaction Time DBs

- Requirements for index methods:
  - Store past logical states
  - Support addition/deletion/modification changes on the objects of the current state
  - Efficiently access and query any database state

Transaction Time DBs

- Queries:
  - Timestamp (timeslice) queries: ex. "Give me all employees at 05/94"
  - Range-timeslice: "Find all employees with id between 100 and 200 that worked in the company on 05/94"
  - Interval (period) queries: "Find all employees with id in [100,200] from 05/94 to 06/96"
- Also known as time-travel queries

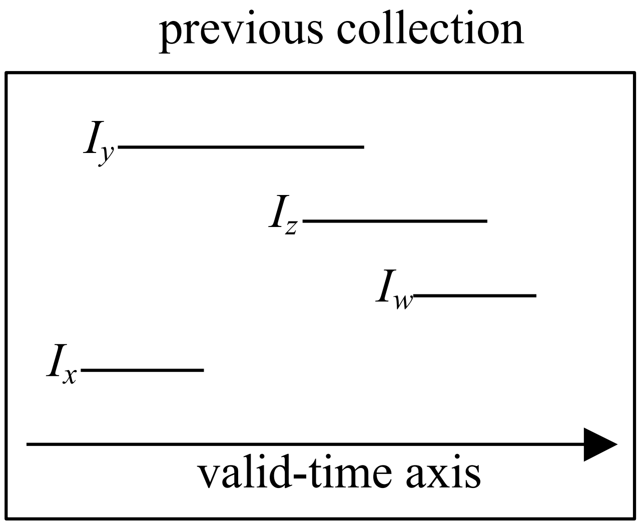
Valid Time DBs

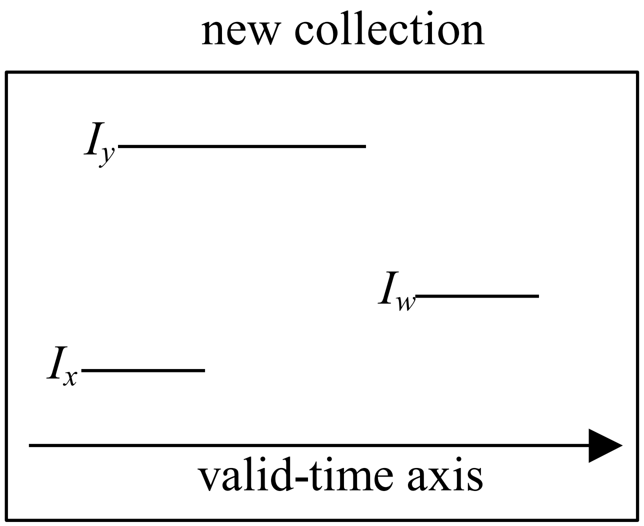
- Time evolves continuously
- Each object is a line segment representing its time span (e.g. Credit card valid time)
- Support full operations on interval data:
  - Deletion at any time
  - Insertion at any time
  - Value change (modification) at any time (no ordering)

Valid Time DBs

- Deletion is physical:
  - No way to know about the previous states of intervals
- The notion of “future”, “present” and “past” is relative to a certain timestamp t

Valid Time DBs





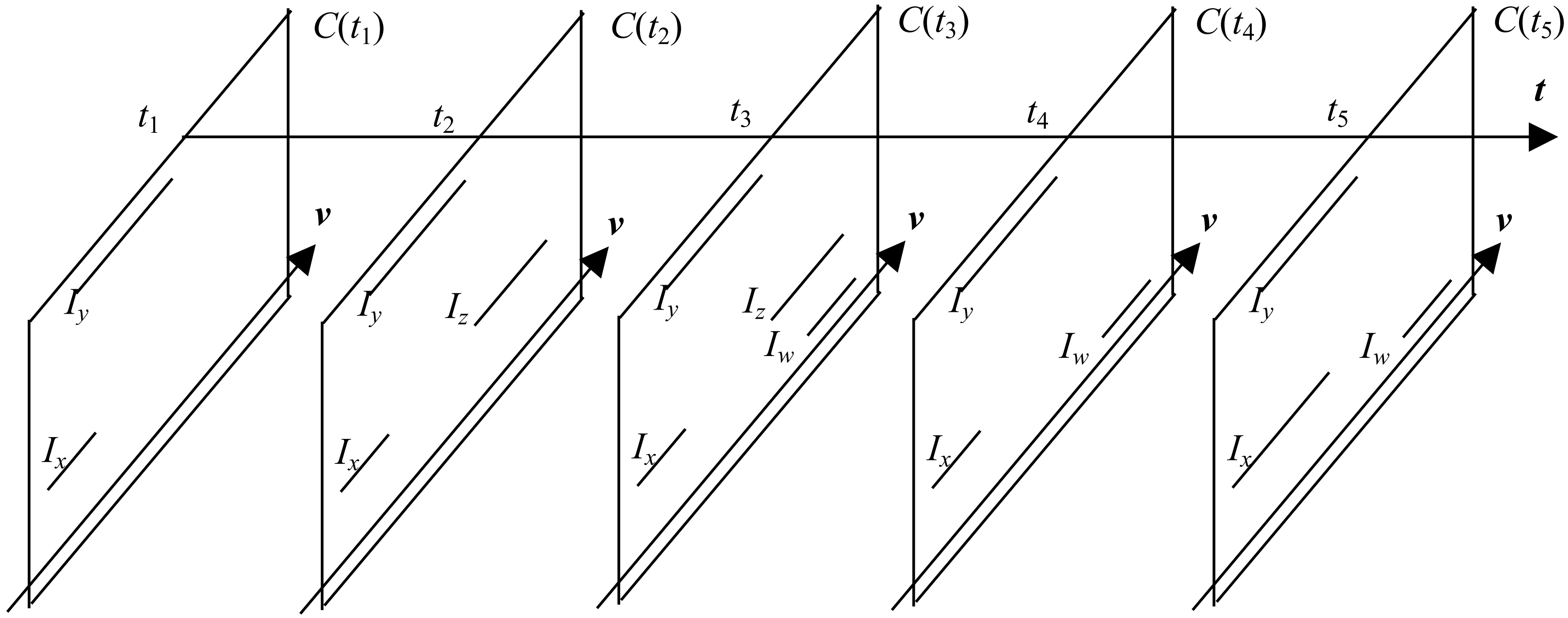
Valid Time DBs

- Requirements for an Index method:
  - Store the latest collection of interval-objects
  - Support add/del/mod changes to this collection
  - Efficiently query the intervals in the collection
    - Timestamp query
    - Interval (period) query

Bitemporal DBs

- A transaction-time Database, but each entity has a validity interval (plus the other attributes of the record)
- Keeping the evolution of a dynamic collection of interval-objects
- At each timestamp, it is a valid time database

Bitemporal DBs



Bitemporal DBs

- Requirements for access methods:
  - Store past/logical states of collections of objects
  - Support additions/deletions/modifications of interval objects of the current logical state
  - Efficient query answering

Temporal Indexing

- Straightforward approach:
  - One B$^+$-tree for each timestamp
  - Problems?
- Transaction time:
  - Snapshot Index, TSB-tree, MVB-tree, MVAS
- Valid time:
  - Interval structures: Interval tree, Segment tree, even R-tree
- Bitemporal:
  - Bitemporal R-tree

Temporal Indexing

- Lower bound on answering timeslice and range-timeslice queries:
  - Space $O(n/B)$, search $O(\log_B n + s/B)$
- n: number of changes, s: answer size, B page capacity

MVB-tree

- Suitable for transaction-time temporal databases
- Insertions and deletions only happen at current time (now)
- A DAG of B-tree nodes (compresses multiple B-trees, one per version/timestamp)
  - Leaf nodes store entries of the form <key,t_{st},t_{end},A>
  - Non-leaf nodes store entries of the form <key,t_{st},t_{end},ptr>
  - Key: time-invariant key (same for all versions of the same object
  - [t_{st},t_{end}): validity interval
  - A: remaining attributes; ptr: pointer to another node

MVB-tree (cont'd)

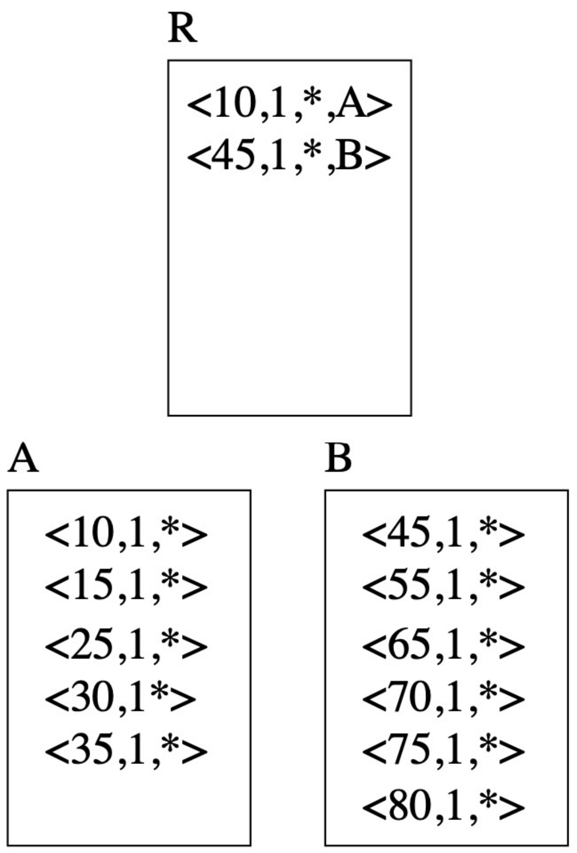
- Each root node stands for an interval of versions (timestamps)
  - A query for a version searches from the corresponding root
- Each update (insert/delete) at time i creates a new version of i the data
  - An entry for which $[t_{st}, t_{end})$ contains i is said to be of version i
- A block that has not been copied is live, otherwise it is dead
- In a live block, if an entry has $t_{end}=*$, this means that the entry is alive at present; in a dead block, if an entry has $t_{end}=*$, this means that the entry was not deleted when the block died
- For each version i and each block A (except roots) the number of entries of version i in A is either 0 or at least d (=k% of block capacity b)

MVB-tree (insertion)

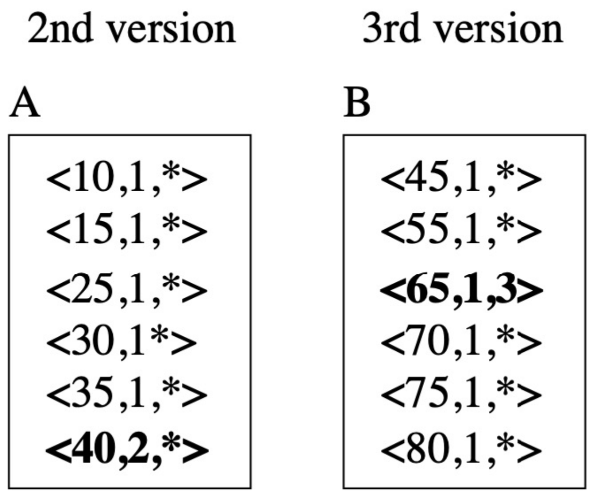
- Newly inserted entries at time i have lifespan [i,*)
- Deleted entries at time i change their lifespan from [t_s,*) to [t_s,i)
- Structural changes:
  - Block overlflow at insert if target node is full
  - Block underflow after deletion if version-i entries less than d
    - Also: if root with version-i has exactly 1 live entry
- Overflow handling:
  - Copy block and remove all but current entries from the copy (version split)
- Underflow handling:
  - Merge with sibling node

MVB-tree updates (example)

Initial MVB-tree

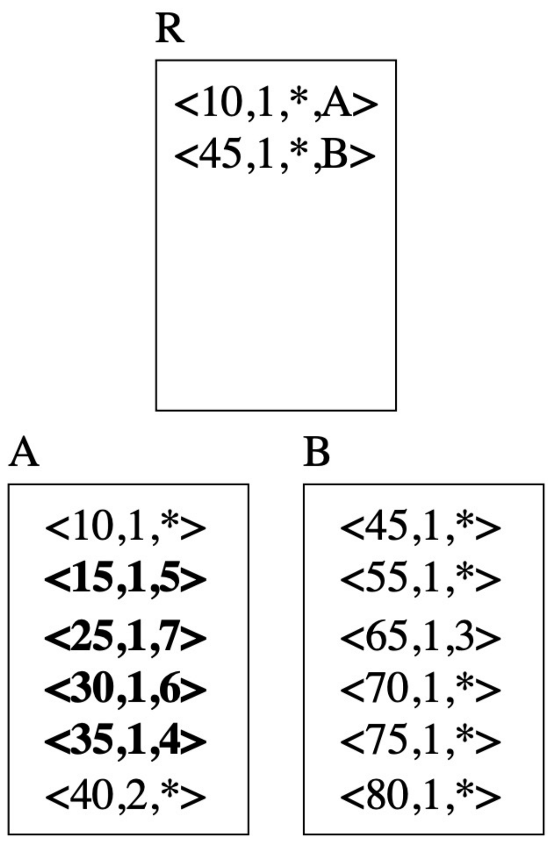


Changes after
insert(40) and delete(65)

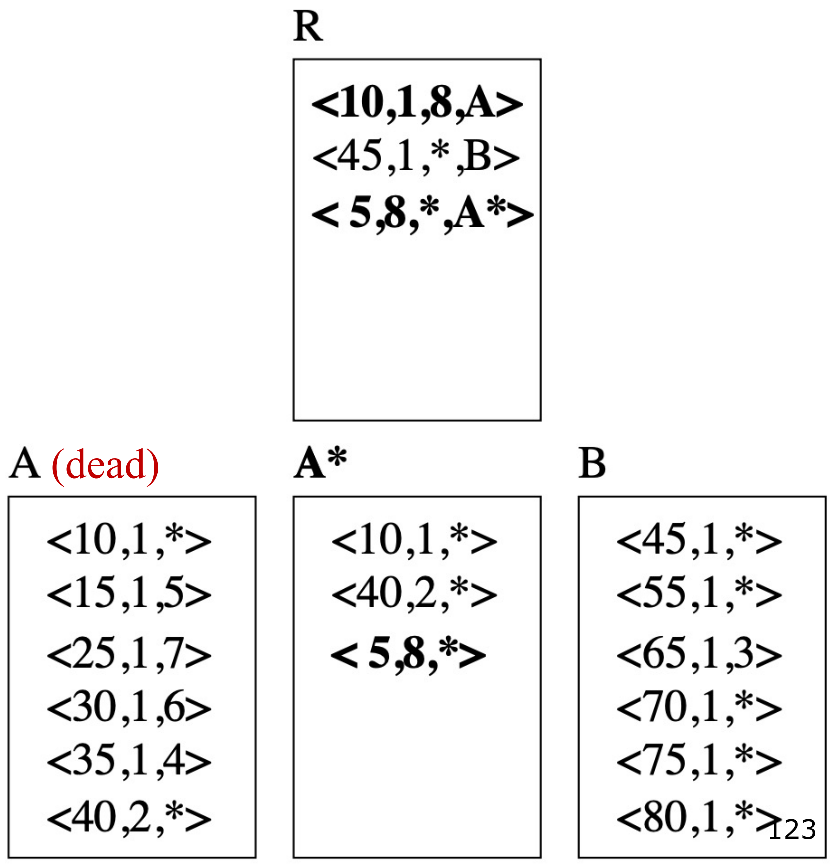


MVB-tree updates (example)

Changes after 4 deletions



Overflow handling after insert(5)



MVB-tree (search)

- Timestamp (timeslice) queries: ex. “Give me all employees at 05/94”
  - Find root that includes query timestamp and recursively search tree to find relevant leaf wherefrom results are drawn
- Interval (period) queries: “Find all employees who worked from 05/94 to 06/96”
  - Find root nodes whose time interval overlap with the query interval and recursively search tree from them to find relevant leaves and results

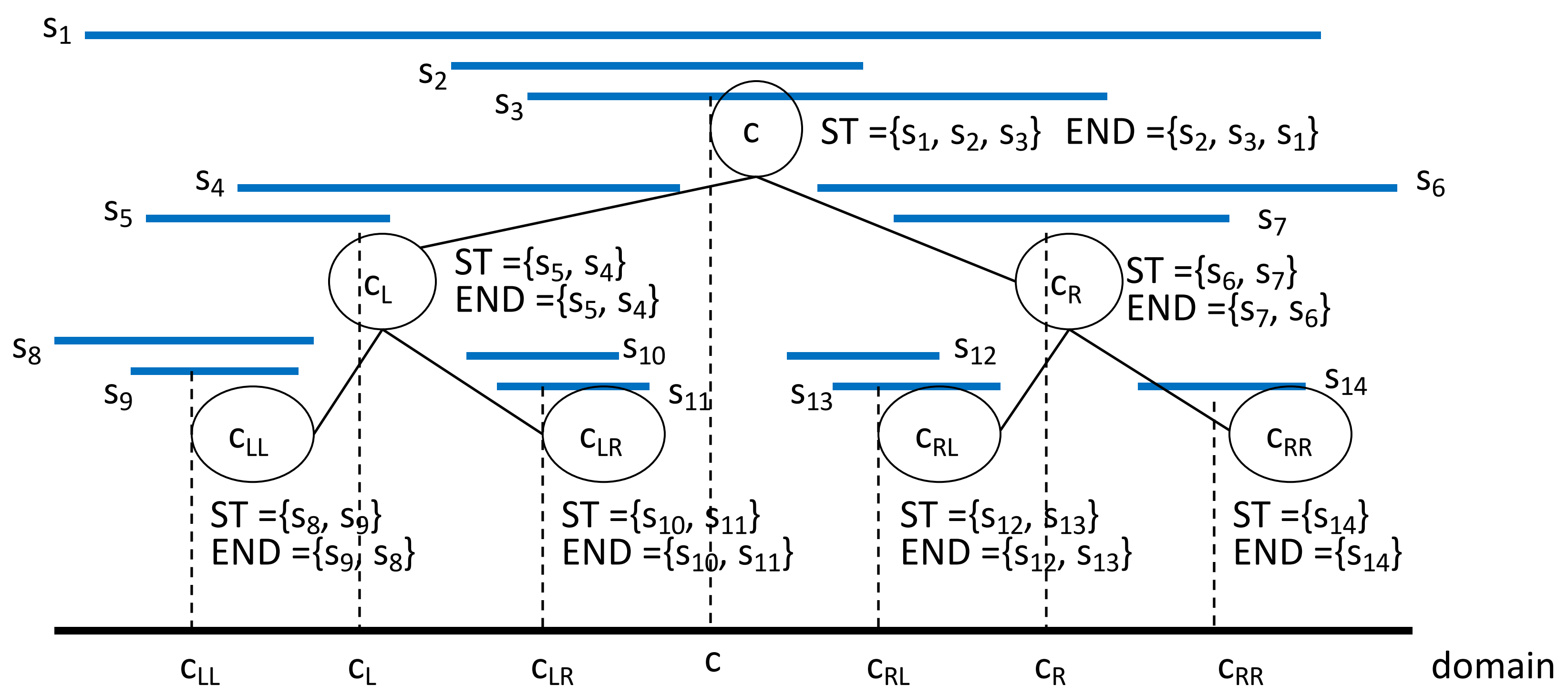
Indexing Intervals (valid time)

- Data structures for main memory
  - Interval Tree
  - Segment Tree
  - Priority Search Tree
- Indexes for secondary memory
  - RI-tree
  - Timeline Index (also used for transaction time DBs)

Interval tree

- Proposed for managing intervals in memory
  - Output sensitive search time: O(logn+k)
  - Space complexity: O(n)
  - Construction cost: O(nlogn)
- Binary tree:
  - c = domain’s center
  - Divide intervals into three sets:
    - S_c = Intervals that include c
    - S_{before} = Intervals before c
    - S_{after} = Intervals after c
  - Root node c includes value c and two sorted arrays
    - Intervals in S_c ordered by start point
    - Intervals in S_c ordered by end point
    - Left (right) child constructed recursively from S_{before} (S_{after})

Interval tree (example)

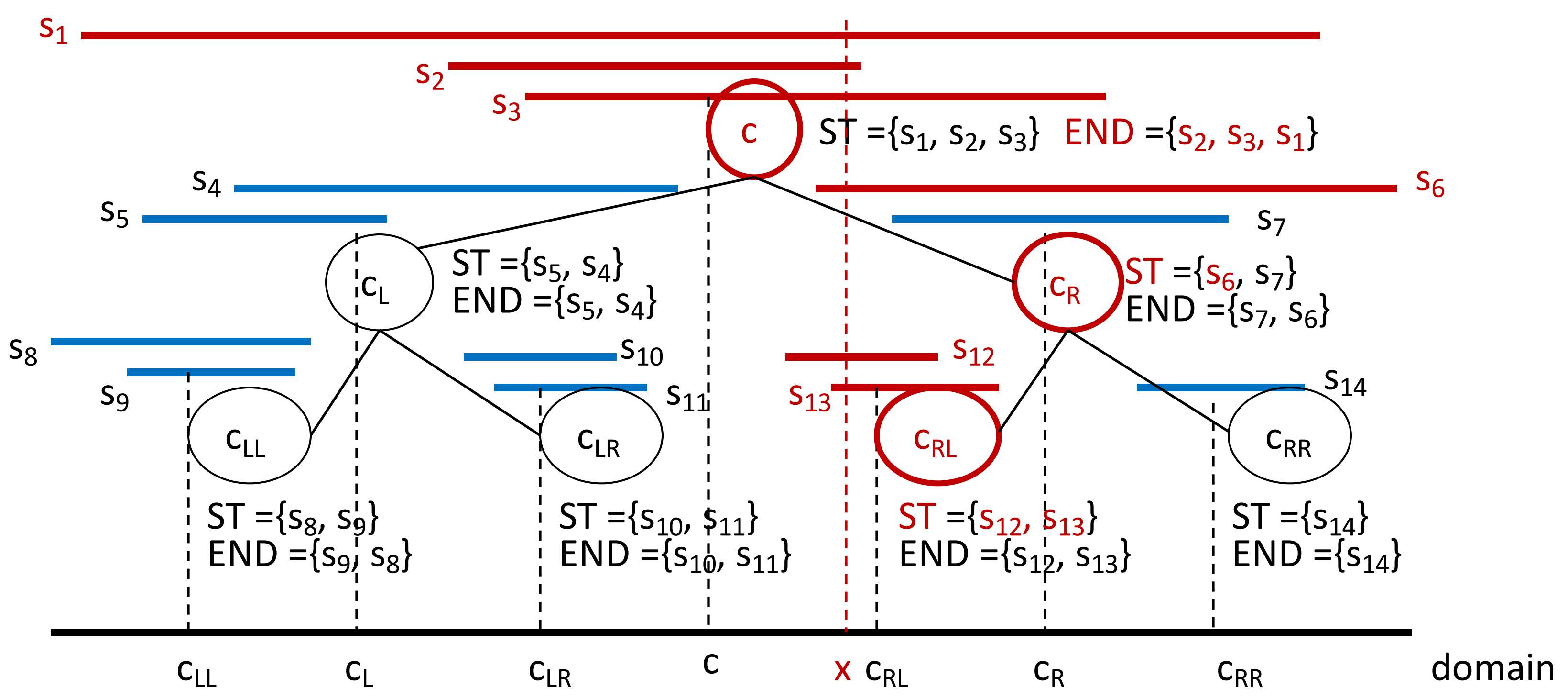


Interval tree (snapshot queries)

- Input: a value x
- Output: intervals that contain x
- function valueSearch(x,node)

```
If x ≤ node.c    // node.c = domain center of node
    I = first interval in node.ST
    While I.start ≤ x
        Add I to result
        I = next interval in node.ST to I
    If node.leftchild!=NULL valueSearch(x,node.leftchild)
else                    // x > node.c
    I = last interval in node.END
    While I.end>x
        Add I to result
        I = previous interval in node.END to I
    If node.rightchild!=NULL valueSearch(x,node.rightchild)
```

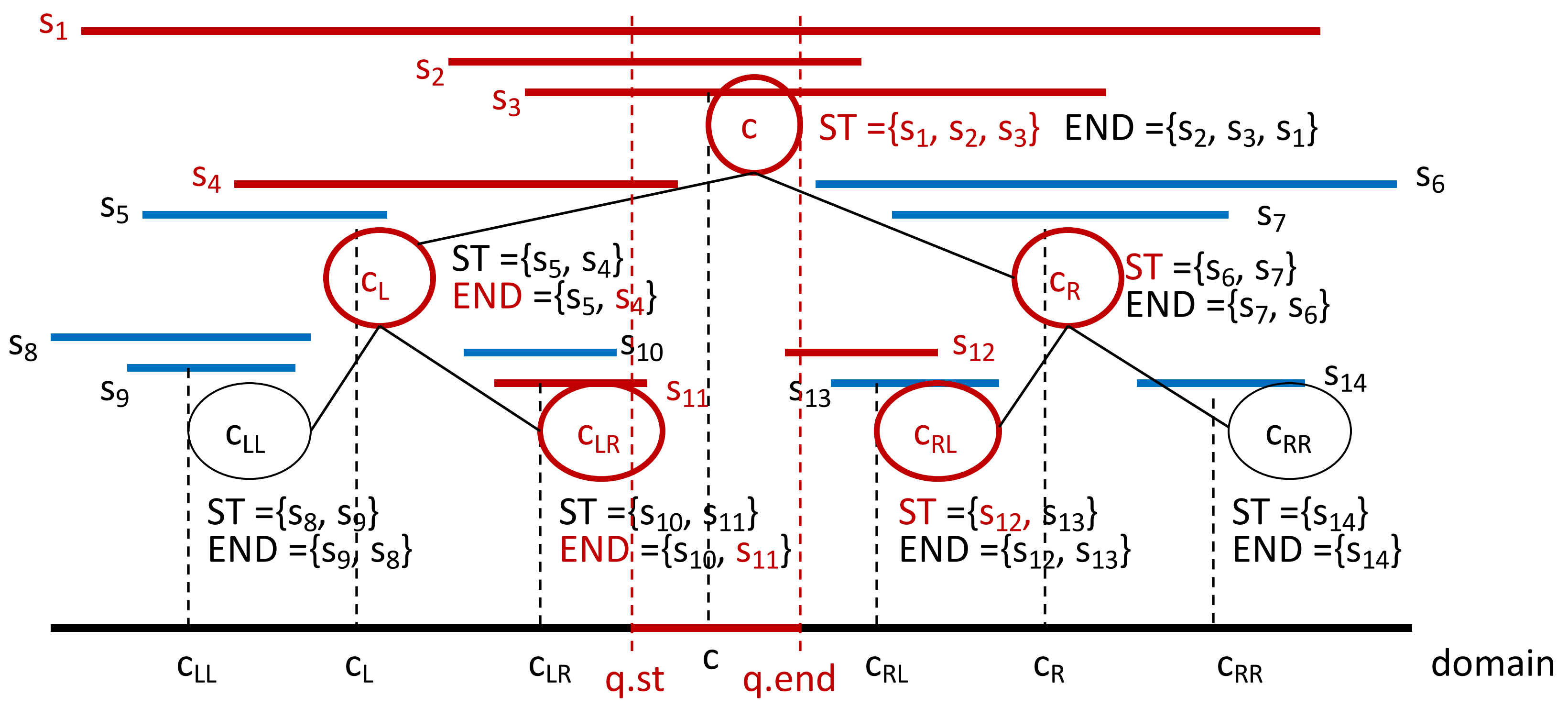
Interval tree (snapshot search)



Interval tree (interval queries)

- Input: an interval q=[q.st,q.end)
- Output: intervals that overlap with q
- Search guidelines:
  - If current node.c ∊ [q.st,q.end)
    - Report all intervals in node
    - Search recursively both children
  - If current node.c < q.st
    - Use node.END to report all intervals I in node.c s.t. I.end>q.st
    - Search recursively right child
  - If current node.c ≥ q.end
    - Use node.ST to report all intervals I in node.c s.t. I.st<q.end
    - Search recursively left child

Interval tree (interval search)

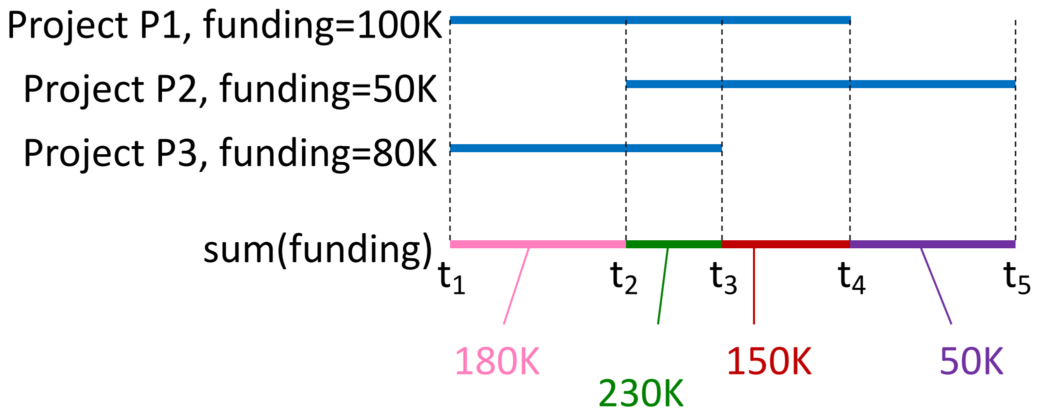


Timeline Index

- Used within SAP HANA
- Designed for temporal aggregation queries
- Can be used for other query types as well
- Originally proposed for transaction time but also suitable for valid time

Temporal aggregation

- For each timestamp t, compute an aggregation over all tuples which are valid at time t
  - Ex: Find the sum of funding from projects at any time
- Observation: aggregation can change only at starting/ending points of intervals
- Aggregation can be restricted to a specific period
  - Ex: Find the sum of funding at each time in 2020



Timeline Index – Event List

- Keeps in sorted order the endpoints of all intervals
- Scanning the table can give us the active intervals at every moment
  - Maintain active set using appropriate data structure
- Aggregating the values relevant to the active intervals at every time moment solves the temporal aggregation problem
- Incremental updates possible for some aggregate functions (e.g., sum, count, average)

Event List

time	**id**	**begin**
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

Timeline Index: Temporal Aggregation

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

0+100K+80K=180K

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

180K+50K=230K

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

230K-80K=150K

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

150K-100K=50K

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

50K-50K=0

Timeline Index – Checkpoints

- Motivation: for a temporal aggregation query confined to a given interval, e.g. [t3,t4) we still have to scan from the beginning of the Event List
- Checkpoint(t) = the complete set of active intervals at time t
- Keep multiple checkpoints (e.g., for the 1st day of each month)
- Given a query q, find the latest check point on or before q.st, get the active set, and scan the Event List from thereon

Event List

time	id	begin
t1	P1	1
t1	P3	1
t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

Temporal Aggregation w/ checkpoints

Example:

Checkpts at t2 and t4

Query:

interval: [t3,t4], aggregate: sum, attribute: funding

time	id	begin
t1	P1	1
t1	P3	1
⚐ **t2**	**P2**	**1**
t3	P3	0
t4	P1	0
t5	P2	0

230K

time	id	begin
t1	P1	1
t1	P3	1
⚐ t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

230K-80K=150K

time	id	begin
t1	P1	1
t1	P3	1
⚐ t2	P2	1
t3	P3	0
t4	P1	0
t5	P2	0

150K-100K=50K

Cluster Analysis

Clustering

- Clustering Problem Overview
- Clustering Techniques
  - Hierarchical Algorithms
  - Partitioning Algorithms
  - Density-based Clustering

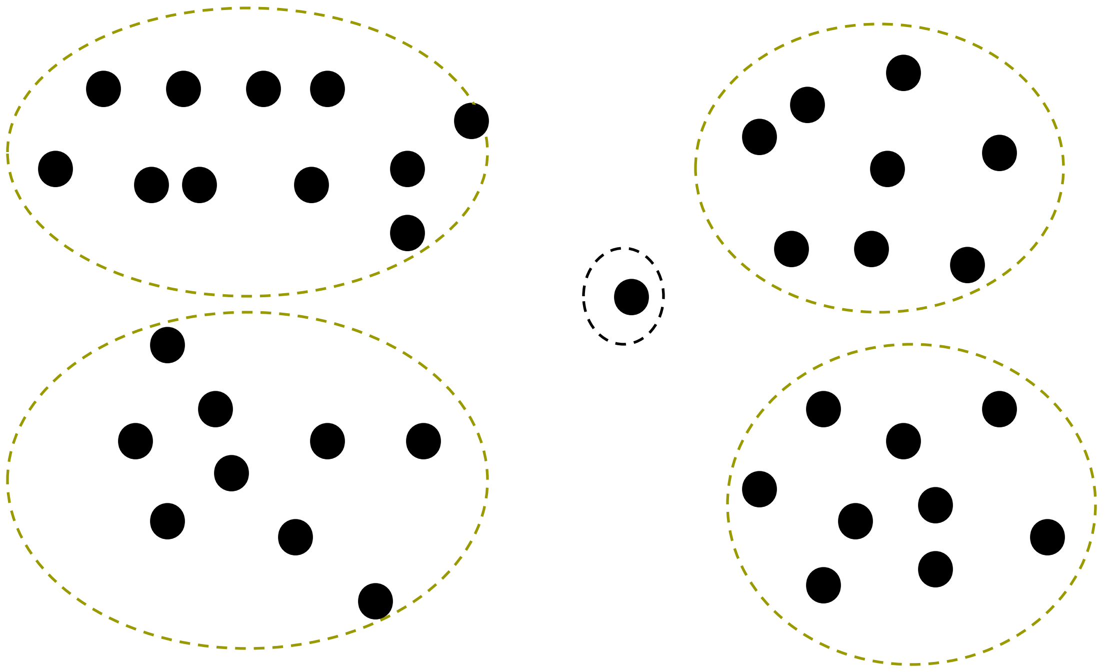
What is Cluster Analysis?

- Cluster: a collection of data objects
  - Objects in the same cluster similar to one another
  - Dissimilar objects in different clusters
- Cluster analysis
  - Grouping a set of data objects into clusters
- Clustering is unsupervised classification:
  - no predefined classes
    - number of clusters unknown
    - Meaning of clusters unknown
  - unsupervised learning
- Clustering is used:
  - As a stand-alone tool to get insight into data distribution
    - Visualization of clusters may unveil important information
  - As a preprocessing step for other algorithms
    - Efficient indexing or compression often relies on clustering

Clustering Examples

- Segment customer database based on similar buying patterns.
- Group houses in a town into neighborhoods based on similar features.
- Identify new plant species
- Identify similar Web usage patterns

Clustering Houses based on Distance



General Applications of Clustering

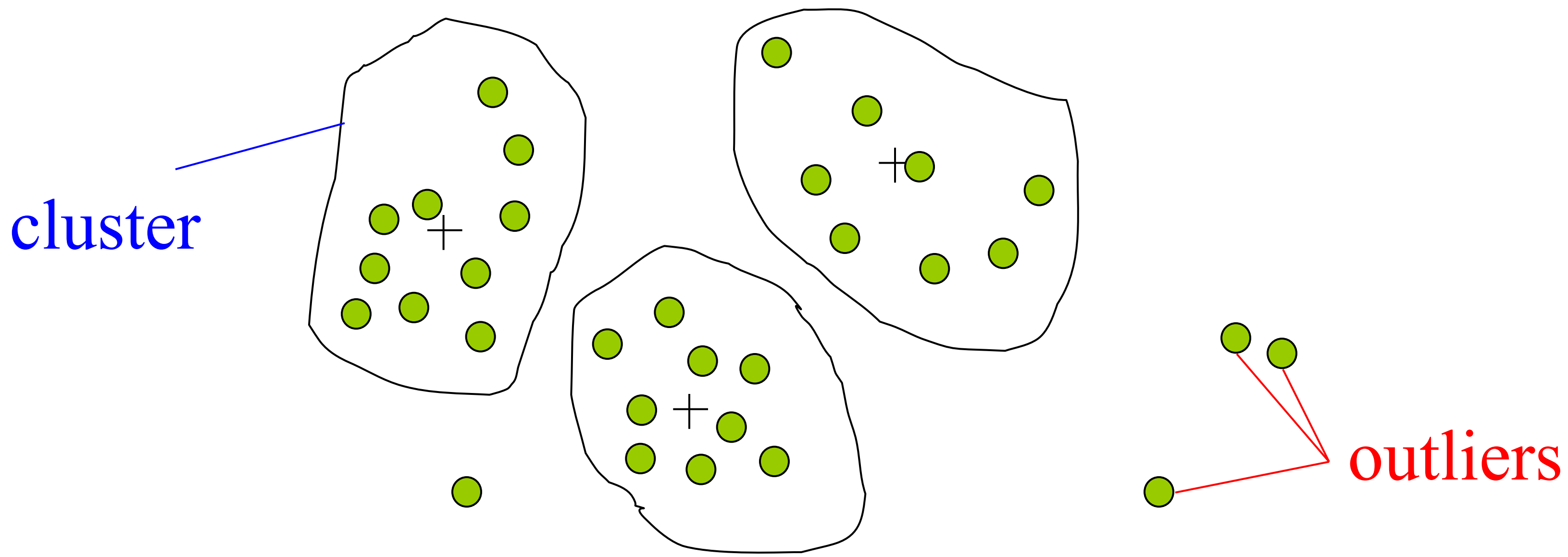
- Pattern Recognition
- Spatial Data Analysis
  - create thematic maps in GIS by clustering feature spaces
  - detect spatial clusters and explain them
- Image Processing
  - cluster images based on their visual content
- Economic Science (especially market research)
- WWW
  - document classification
  - cluster Weblog data to discover groups of similar access patterns

Examples of Clustering Applications

- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- Land use: Identification of areas of similar land use in an earth observation database
- Insurance: Identifying groups of motor insurance policy holders with a high average claim cost
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Earth-quake studies: Observed earth quake epicenters should be clustered along continent faults

Outliers

- Outliers are objects that do not belong to any cluster or form clusters of very small cardinality



- In some applications we are interested in discovering outliers, not clusters (outlier analysis)

Clustering Problem Definition

- Given a database $D=\{t_1,t_2,\ldots,t_n\}$ of objects (or tuples) and an integer value k, the Clustering Problem is to define a mapping $f:D\rightarrow\{1,..,k\}$ where each t_i is assigned to one cluster C_j, $1<=j<=k$.
- A Cluster, C_j, contains precisely those objects mapped to it.

Major Clustering Approaches

- Hierarchical algorithms: Create a hierarchical decomposition of the set of data (or objects) using some criterion
- Partitioning algorithms: Construct random partitions and then iteratively refine them by some criterion
- Density-based: based on connectivity and density functions
- Grid-based: based on a multiple-level granularity structure
- Model-based: A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other

Cluster Parameters

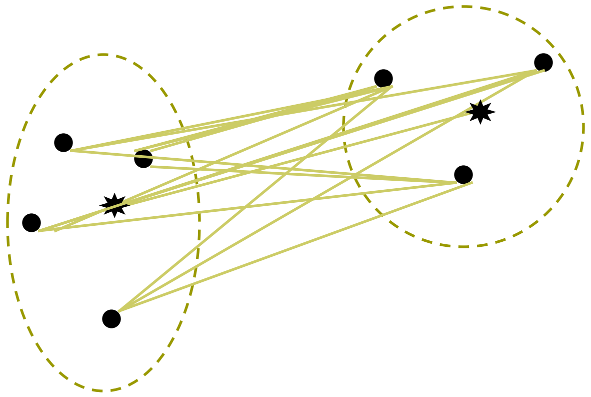
$$centroid = C_m = \frac{\sum_{i=1}^{N}(t_{mi})}{N}$$

$$radius = R_m = \sqrt{\frac{\sum_{i=1}^{N}(t_{mi} - C_m)^2}{N}}$$

$$diameter = D_m = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}(t_{mi} - t_{mj})^2}{(N)(N-1)}}$$

Distance Between Clusters

- Single Link: smallest distance between points
- Complete Link: largest distance between points
- Average Link: average distance between points
- Centroid: distance between centroids

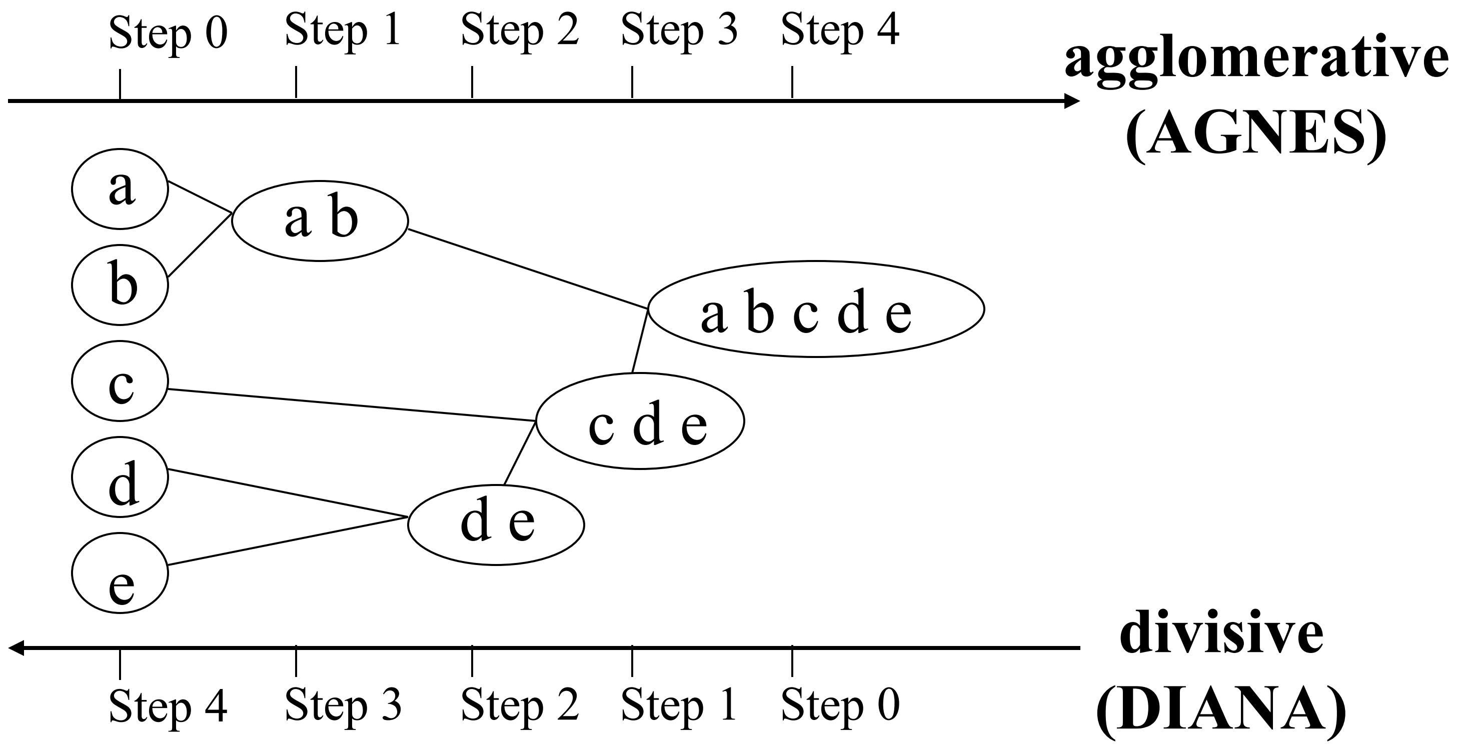


Hierarchical Clustering

- Clusters are created in levels actually creating sets of clusters at each level.
- Agglomerative
  - Initially each item in its own cluster
  - Iteratively clusters are merged together
  - Bottom Up
- Divisive
  - Initially all items in one cluster
  - Large clusters are successively divided
  - Top Down

Hierarchical Clustering

- Hierarchical clustering does not require the number of clusters ***k*** as an input, but needs a termination condition

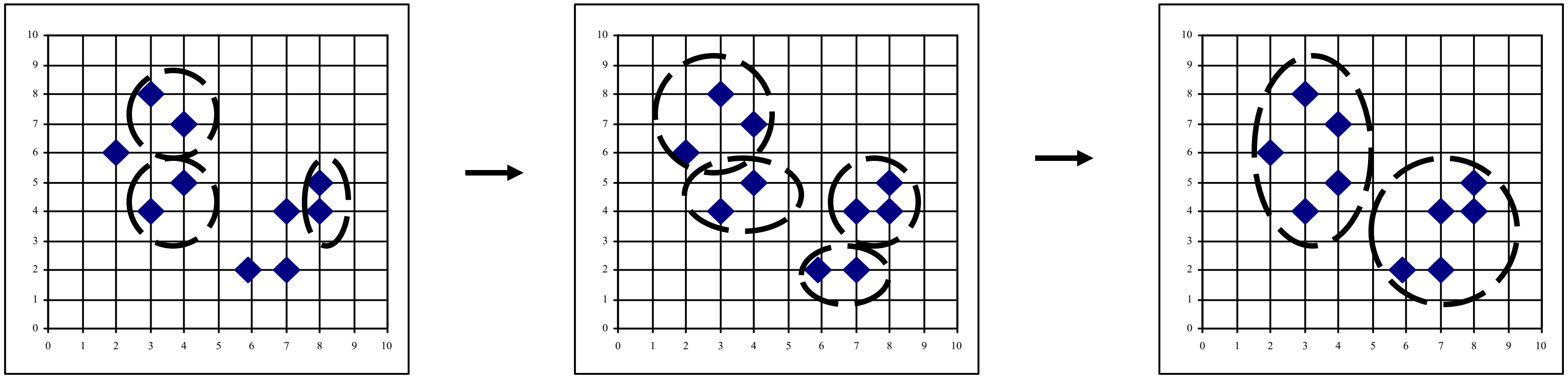


Hierarchical Algorithms

- Single Link
- MST Single Link
- Complete Link
- Average Link

AGNES (Agglomerative Nesting)

- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical analysis packages, e.g., Splus
- Use the Single-Link method and the dissimilarity matrix.
- Merge objects that have the least dissimilarity
- Go on in a non-descending fashion
- Eventually all objects belong to the same cluster



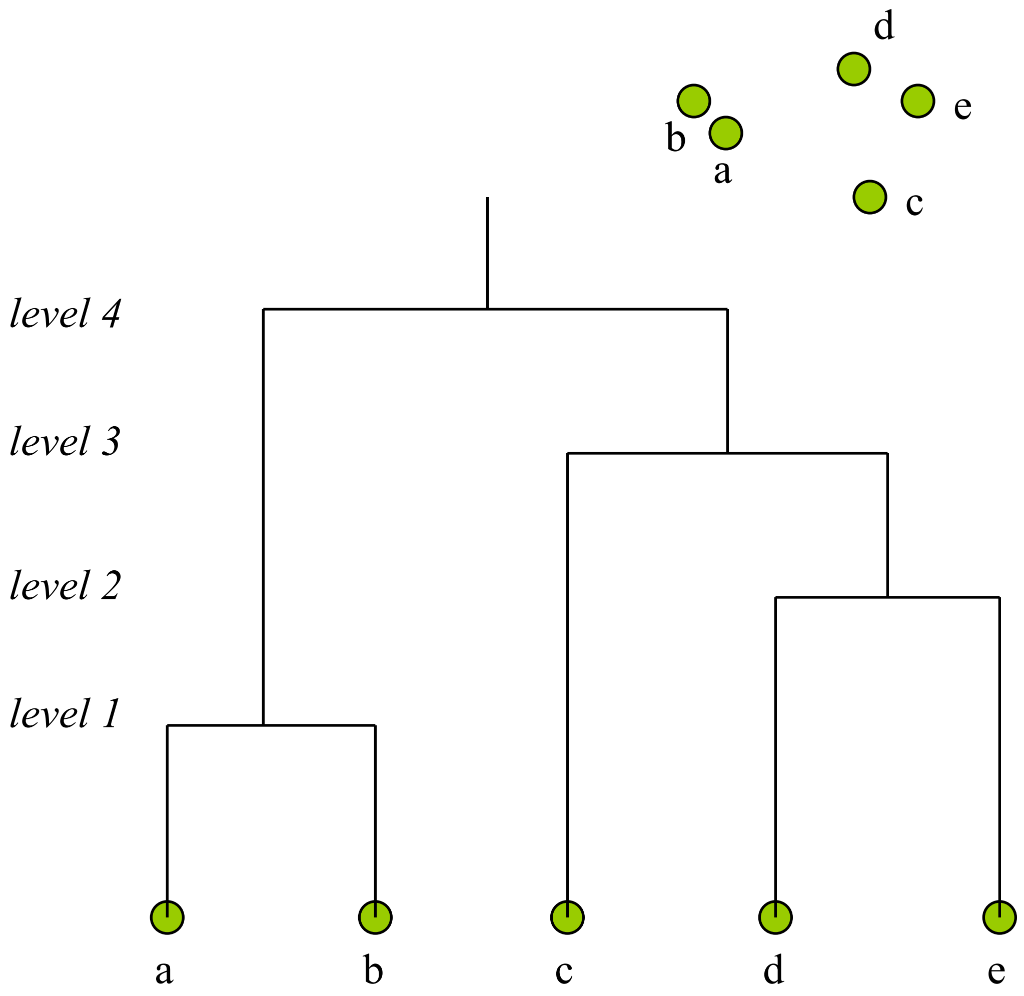
- Single-Link: each time merge the clusters (C_1,C_2) which are connected by the *shortest single link* of objects, i.e., $\min_{p \in C1, q \in C2} dist(p,q)$

A *Dendrogram* Shows How the Clusters are Merged Hierarchically

Decompose data objects into a several levels of nested partitioning (<u>tree</u> of clusters), called a <u>dendrogram</u>.

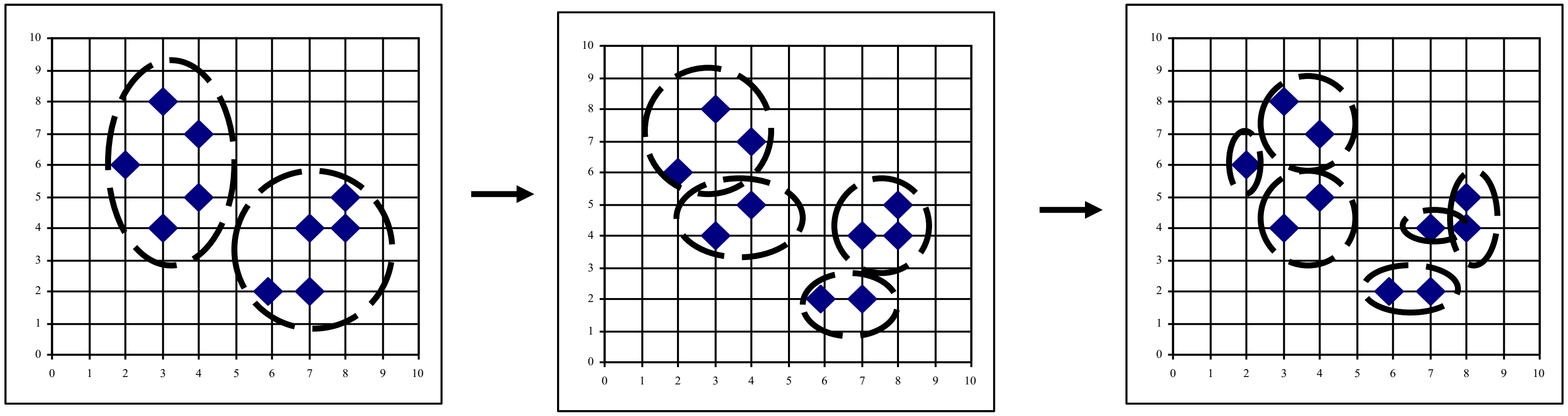
A <u>clustering</u> of the data objects is obtained by <u>cutting</u> the dendrogram at the desired level, then each <u>connected component</u> forms a cluster.

E.g., level 1 gives 4 clusters: {a,b},{c},{d},{e},
level 2 gives 3 clusters: {a,b},{c},{d,e}
level 3 gives 2 clusters: {a,b},{c,d,e},
etc.



DIANA (Divisive Analysis)

- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical analysis packages, e.g., Splus
- Inverse order of AGNES
- Eventually each node forms a cluster on its own



More on Hierarchical Clustering Methods

- Major weakness of agglomerative clustering methods
  - do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
  - can never undo what was done previously
- Integration of hierarchical with distance-based clustering
  - BIRCH (1996): uses CF-tree and incrementally adjusts the quality of sub-clusters
  - CURE (1998): selects well-scattered points from the cluster and then shrinks them towards the center of the cluster by a specified fraction
  - CHAMELEON (1999): hierarchical clustering using dynamic modeling

Partitioning Algorithms: Basic Concepts

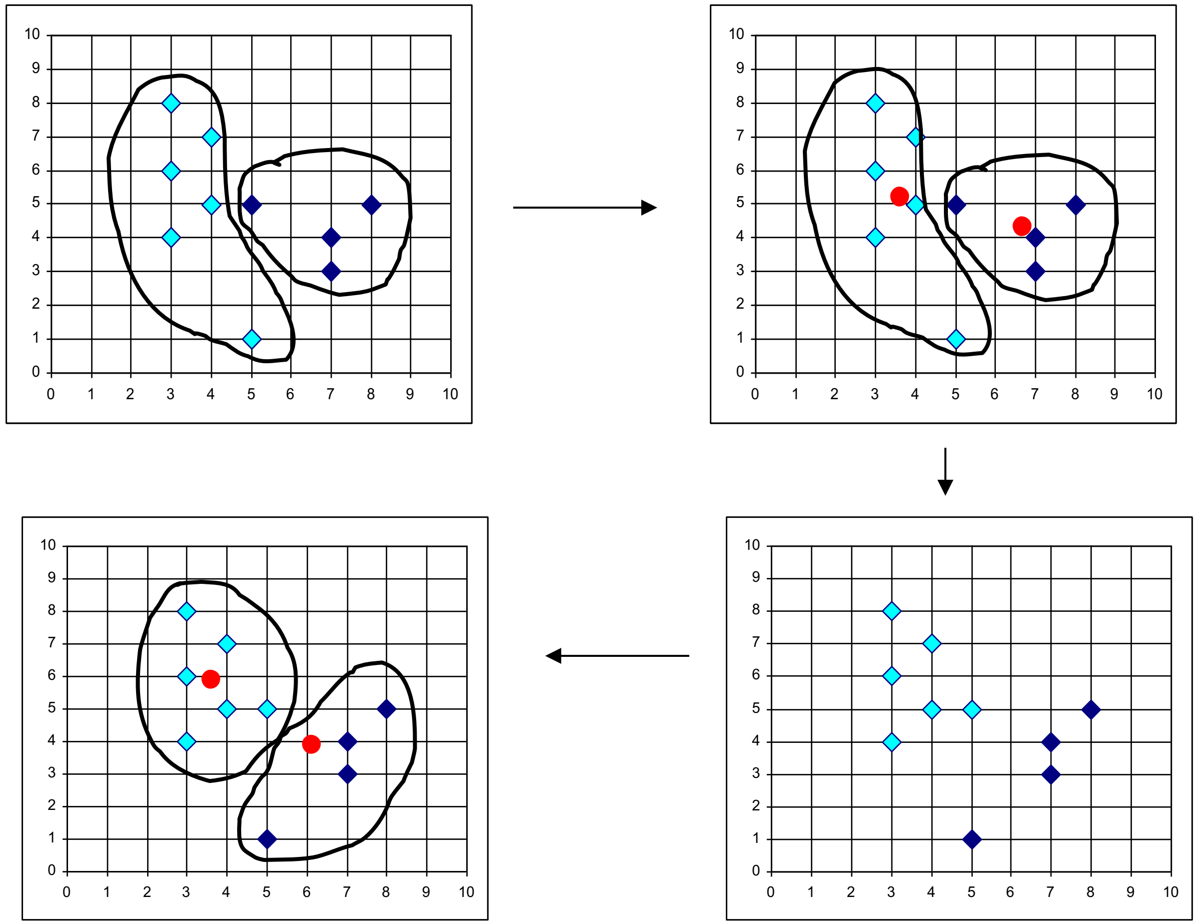
- Partitioning method: Construct a partition of a database ***D*** of ***n*** objects into a set of ***k*** clusters
- Given a *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all partitions
  - Heuristic methods: *k-means* and *k-medoids* algorithms
  - *k-means* (MacQueen'67): Each cluster is represented by the center of the cluster
  - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

The k-means Clustering Method

- Given *k*, the *k-means* algorithm is implemented in 4 steps:
  1. Partition objects into *k* nonempty subsets
  2. Compute seed points as the centroids of the clusters of the current partition. The centroid is the center (mean point) of the cluster.
  3. Assign each object to the cluster with the nearest seed point.
  4. Go back to Step 2, stop when no more new assignment.

The k-means Clustering Method

- Example



Comments on the k-means Method

- <u>Strength</u>
  - *Relatively efficient*: *O*(*tkn*), where *n* is # objects, *k* is # clusters, and *t* is # iterations. Normally, *k*, *t* << *n*.
  - Often terminates at a *local optimum*. The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*
- <u>Weakness</u>
  - Applicable only when *mean* is defined (what about categorical data)?
  - Need to specify *k,* the *number* of clusters, in advance
  - Unable to handle noisy data and *outliers*
  - Not suitable to discover clusters with *non-convex shapes*

Variations of the k-means Method

- A few variants of the *k-means* which differ in
  - Selection of the initial *k* means
  - Dissimilarity calculations
  - Strategies to calculate cluster means

The k-Medoids Clustering Method

- Find *representative* objects, called <u>medoids</u>, in clusters
- *PAM* (Partitioning Around Medoids, 1987)
  - starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering
  - *PAM* works effectively for small data sets, but does not scale well for large data sets
- *CLARA* (Kaufmann & Rousseeuw, 1990)
- *CLARANS* (Ng & Han, 1994): Randomized sampling

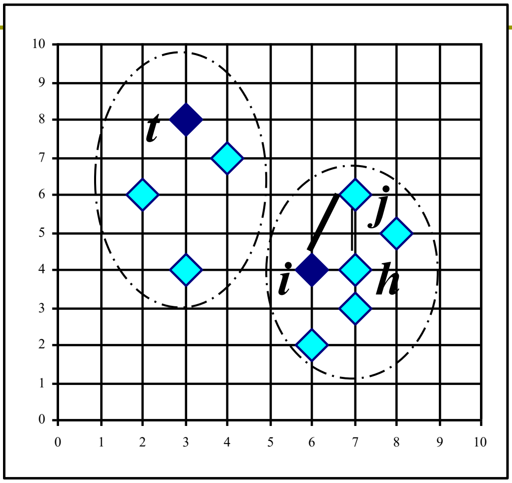
PAM (Partitioning Around Medoids)

- PAM (Kaufman and Rousseeuw, 1987), built in statistical package S+
- Use real object to represent the cluster
  1. Select ***k*** representative objects arbitrarily
  2. For each pair of non-selected object ***h*** and selected object ***i***, calculate the total swapping cost $\boldsymbol{TC_{ih}}$
  3. Find the pair of ***i*** and ***h***, for which TC_{ih} is the smallest
  4. If $TC_{ih} < 0$
     - replace ***i*** by ***h***
     - assign each non-selected object to the closest representative object
     - Goto 3

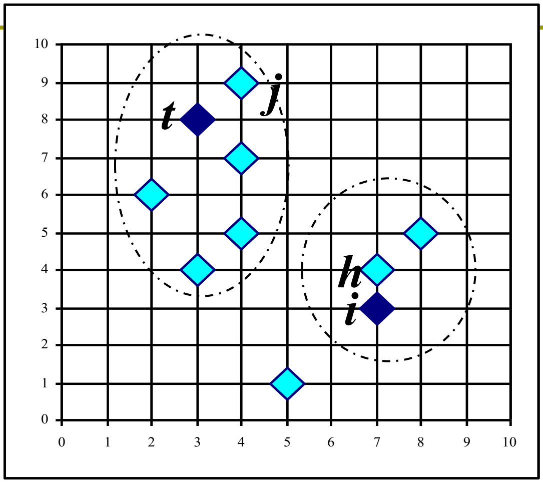
PAM Clustering: Total swapping cost $TC_{ih}=\sum_j C_{jih}$

- i is a current medoid, h is a non-selected object
- Assume that i is replaced by h in the set of medoids
- $TC_{ih} = 0$;
- For each non-selected object $j \neq h$:
  - TC_{ih} += $d(j, new\_med_j) - d(j, prev\_med_j)$:
    - new_med_j = the closest medoid to j after i is replaced by h
    - $prev\_med_j$ = the closest medoid to j before i is replaced by h

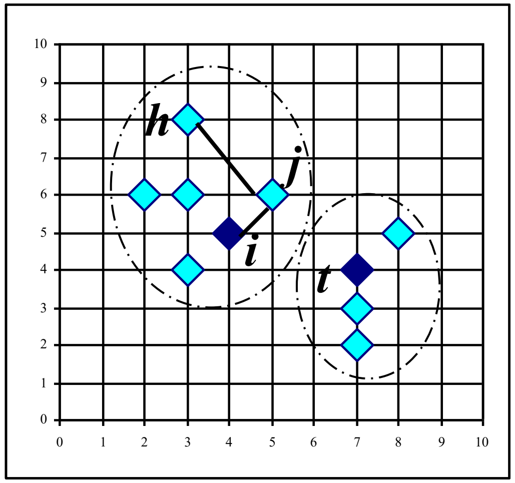
PAM Clustering: Total swapping cost $TC_{ih}=\sum_j C_{jih}$



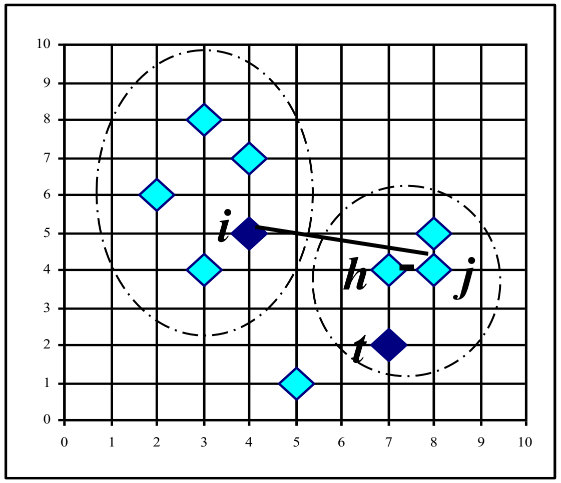
$C_{jih} = d(j, h) - d(j, i)$



$C_{jih} = 0$



$C_{jih} = d(j, t) - d(j, i)$



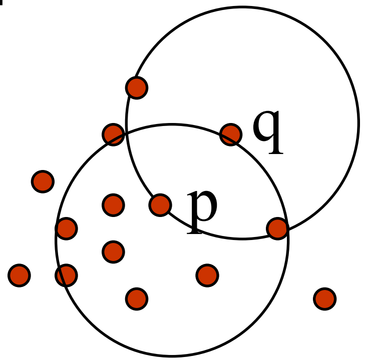
$C_{jih} = d(j, h) - d(j, t)$

Density-Based Clustering Methods

- Clustering based on density (local cluster criterion), such as density-connected points
- Major features:
  - Discover clusters of arbitrary shape
  - Handle noise
  - One scan
  - Need density parameters as termination condition
- Several interesting studies:
  - DBSCAN: Ester, et al. (KDD'96)
  - OPTICS: Ankerst, et al (SIGMOD'99).
  - DENCLUE: Hinneburg & D. Keim (KDD'98)
  - CLIQUE: Agrawal, et al. (SIGMOD'98)

Density-Based Clustering: Background

- Neighborhood of point p=all points within distance Eps from p:
  - $N_{Eps}(p)=\{q \mid dist(p,q) <= Eps\}$
- Two parameters***:***
  - ***Eps***: Maximum radius of the neighborhood
  - ***MinPts***: Minimum number of points in an Eps-neighborhood of that point
- If the number of points in the Eps-neighborhood of p is at least ***MinPts***, then p is called a core object.
- If an object q is not a core point, but it belongs to the Eps-neighborhood of a core point, then q is a border object.

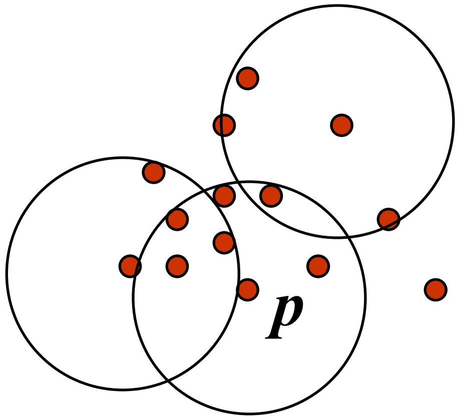


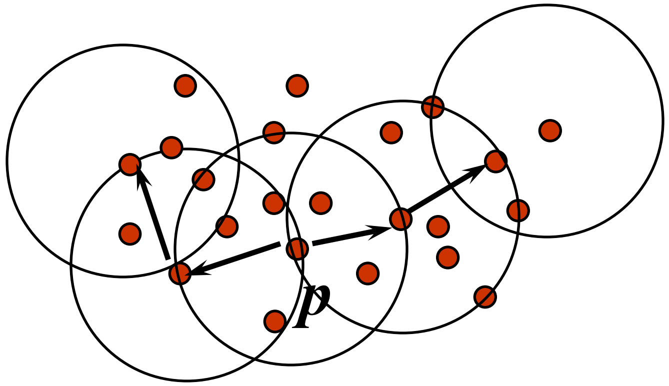
MinPts = 5

Eps = 1 cm

Density-Based Clustering: Background

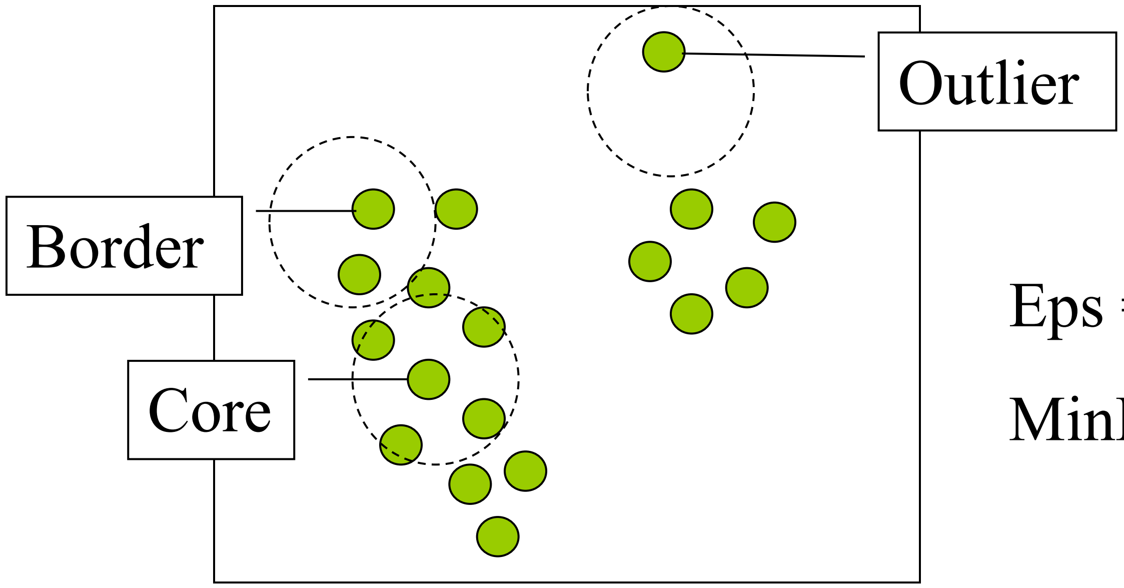
- A core point and its Eps-neighborhood define a cluster.
- If two core points p and q belong to the Eps-neighborhood of each other, the corresponding clusters are merged.





DBSCAN: Density Based Spatial Clustering of Applications with Noise

- Discovers clusters of arbitrary shape in spatial databases with noise



Eps = 1cm

MinPts = 5

DBSCAN: The Algorithm

1. Select an unprocessed point ***p***
2. Find Eps-Neighborhood of ***p*** using parameter ***Eps***.
3. If ***p*** is a core point (based on ***MinPts***), a cluster is formed
   - Put all points in Eps-Neighborhood of ***p*** in a queue Q and examine the points in Q whether they are core points; expand current cluster accordingly
4. Otherwise leave ***p*** unlabeled (***p*** may be included to a cluster later if it is found to be in the Eps-Neighborhood of a core point; if not, becomes an outlier)
5. If there are more unprocessed points goto 1

Summary

- Multi-dimensional data, refer to objects having multiple attributes
- Besides similarity search, many additional queries and analysis tasks can be performed
  - Top-k queries
  - Skyline queries
  - OLAP
  - Time-travel queries
  - Data mining operations (dimensionality reduction, clustering, outlier analysis, association analysis, classification)