Entrepôts de données

Federico Ulliana

QUERY OPTIMIZATION FOR ON-LINE ANALYTICS (OLAP)

DW Need Optimization !

- At this stage of the DW design we should have a good model to support information extraction
  - at least one problem is adressed.

- Next step is to deal with data that may be so voluminous (OR queries so resource-demanding) that response times take minutes or hours

OLAP

ON-**L**INE **A**NALYTICAL QUERY **P**ROCESSING

Although queries may touch millions of rows (= ANALYTICAL),

the system should answer within seconds (= ON-LINE).

- We will see some interesting optimizations
  - Materialized Views
  - Bitmap Indexes
  - Join Indexes

Snapshot Book Sales

Book_Dim

Book	Genre
Winnie	Children
C	Inform.

Fact_Book_Sales

Book	Shop	Date	TotPrice
Winnie	854	11/11	20
C	876	12/11	10
C	854	12/11	5
Winnie	876	13/11	60

- Hypothesis : size fact table ~10^9

Analytics : book selling by genre

```
SELECT Genre, SUM(TotPrice) as Profit

FROM   Fact_Book_Sales F, Book_Dim D

WHERE  F.BookID = D.BookID

GROUP  BY D.Genre
```

- Touches all rows in the fact table (can be ~10^9)

Analytics : book selling by genre

Genre	Profit
Children	1600
C	400
	:

Materialized Views

- Goal : pre-compute some queries so as to speed up query answering

- Note : the previous query is independent of time

Materialized Views

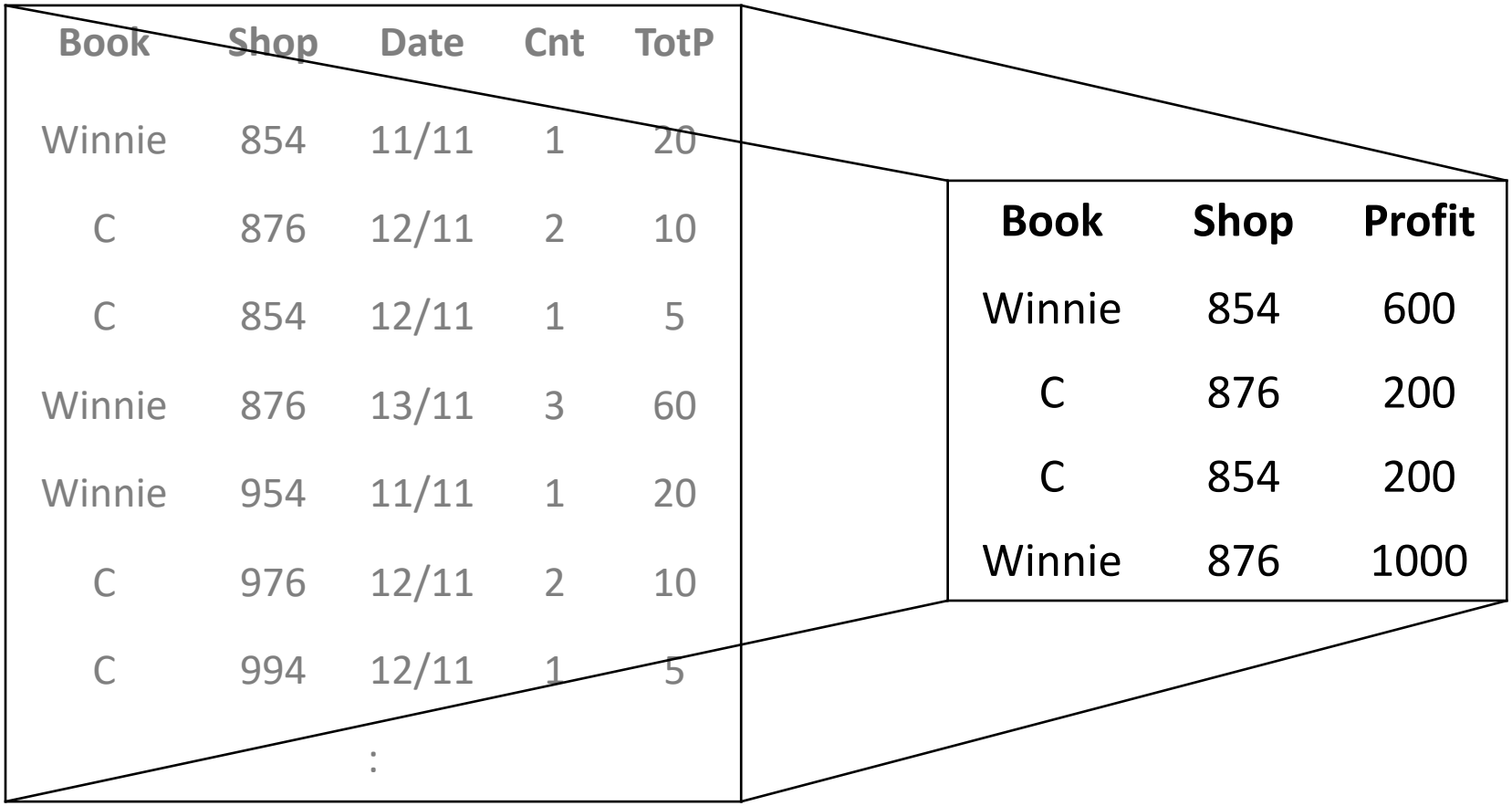
```
CREATE MATERIALIZED VIEW
SalesView (BookID, ShopID, Profit)
AS
    SELECT   BookID,ShopID,SUM(TotPrice)
      FROM   Fact_Book_Sales
 GROUP BY   BookID, ShopID
```

Materialized Views

Book	Shop	Date	Cnt	TotP
Winnie	854	11/11	1	20
C	876	12/11	2	10
C	854	12/11	1	5
Winnie	876	13/11	3	60
Winnie	954	11/11	1	20
C	976	12/11	2	10
C	994	12/11	1	5
		:		

F

Materialized Views

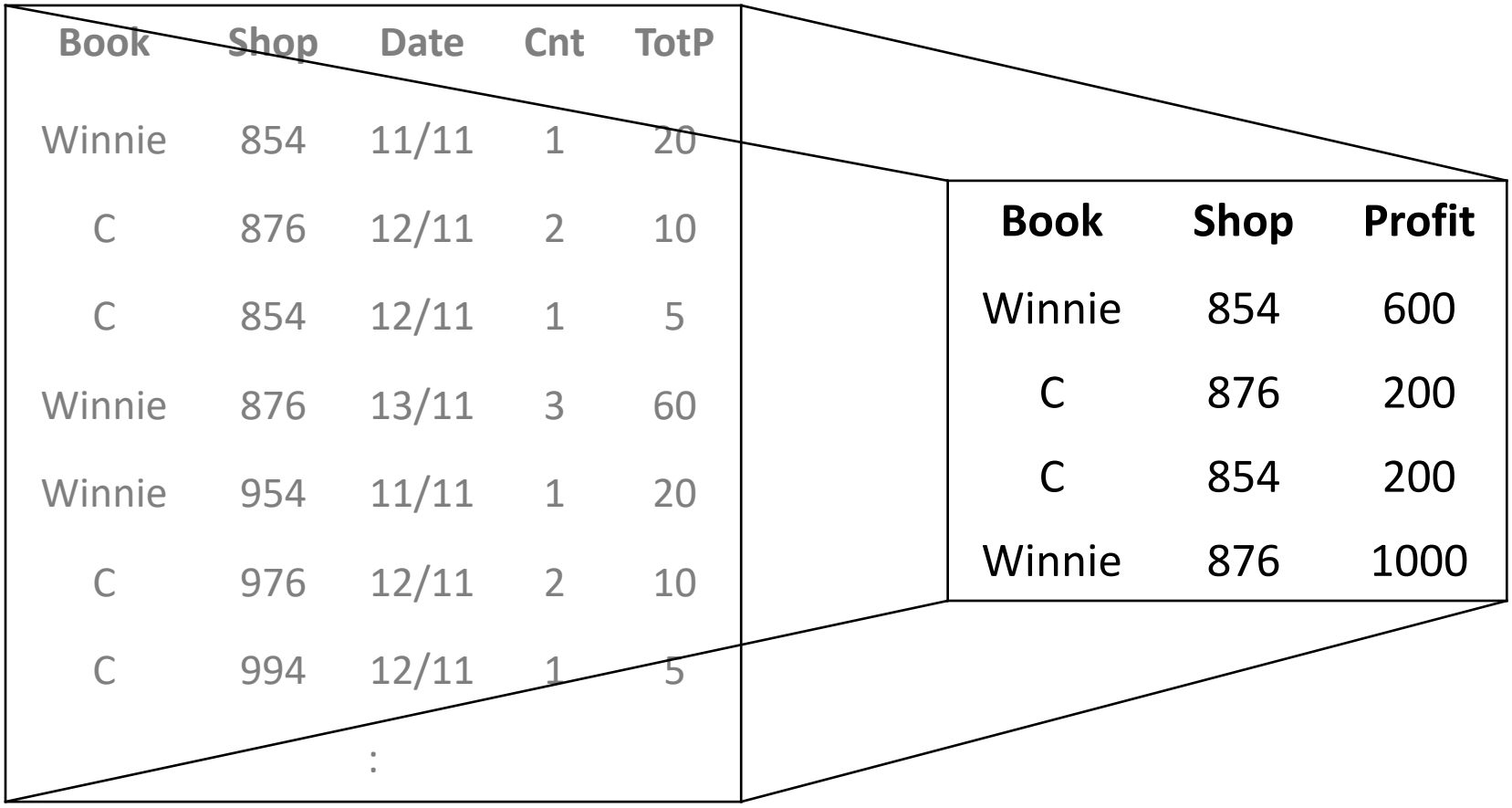


Book	Shop	Date	Cnt	TotP
Winnie	854	11/11	1	20
C	876	12/11	2	10
C	854	12/11	1	5
Winnie	876	13/11	3	60
Winnie	954	11/11	1	20
C	976	12/11	2	10
C	994	12/11	1	5
		:		

Book	Shop	Profit
Winnie	854	600
C	876	200
C	854	200
Winnie	876	1000

F

Materialized Views

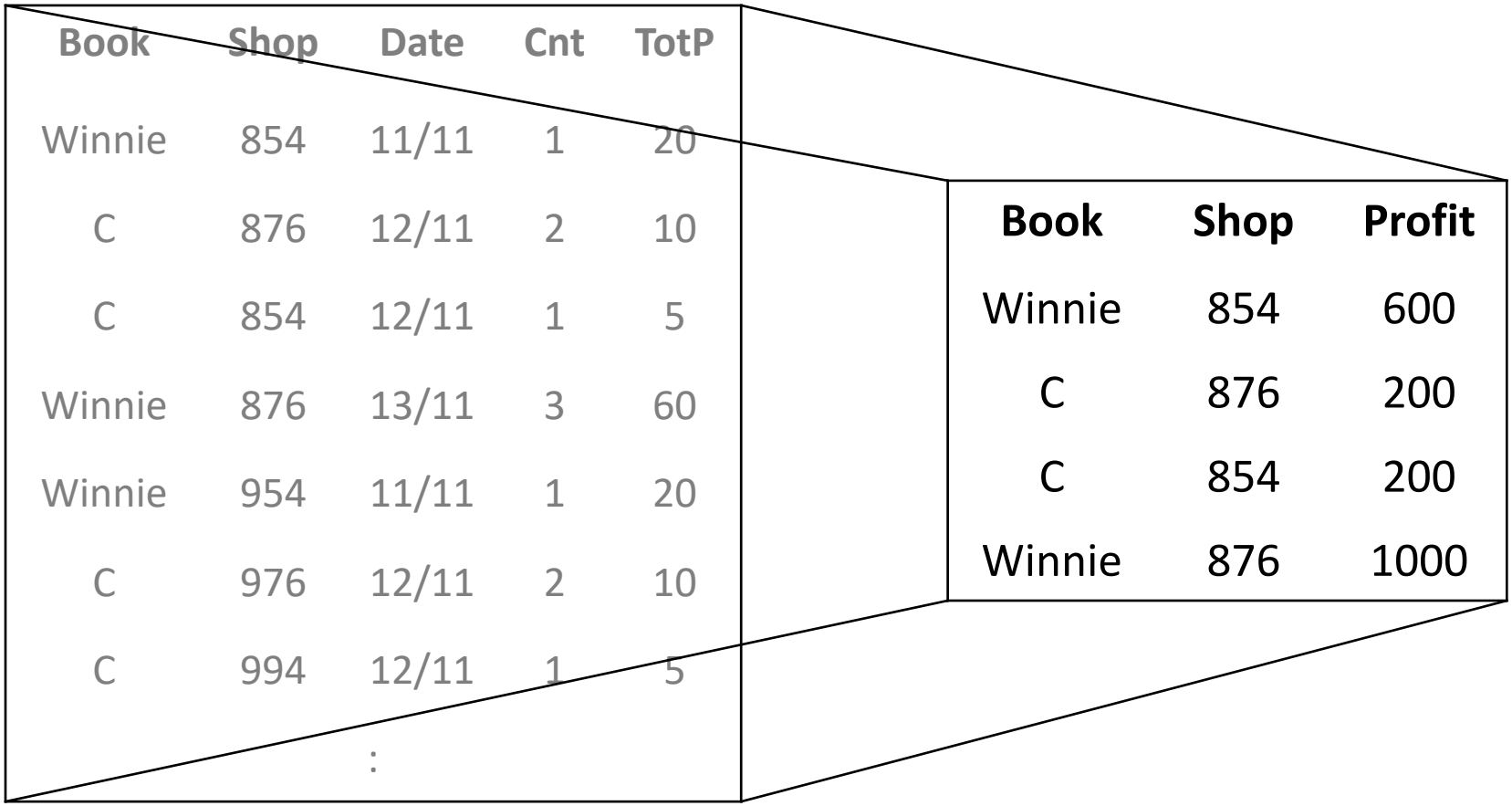


Book	Shop	Date	Cnt	TotP
Winnie	854	11/11	1	20
C	876	12/11	2	10
C	854	12/11	1	5
Winnie	876	13/11	3	60
Winnie	954	11/11	1	20
C	976	12/11	2	10
C	994	12/11	1	5
		:		

Book	Shop	Profit
Winnie	854	600
C	876	200
C	854	200
Winnie	876	1000

F **V**

Materialized Views



Book	Shop	Date	Cnt	TotP
Winnie	854	11/11	1	20
C	876	12/11	2	10
C	854	12/11	1	5
Winnie	876	13/11	3	60
Winnie	954	11/11	1	20
C	976	12/11	2	10
C	994	12/11	1	5
		:		

Book	Shop	Profit
Winnie	854	600
C	876	200
C	854	200
Winnie	876	1000

Size Fact Table >> Size View

Without Materialization **Q(F)**

```
SELECT    Genre, SUM(TotPrice)

FROM      FactSales F, BookDim D

WHERE     F.BookID = D.BookID

GROUP BY D.Genre
```

With Materialization $Q^{ref}(V)$

```
SELECT    Genre, SUM(Profit)

FROM      SalesView V, DimBook D

WHERE     V.BookID = D.BookID

GROUP BY  D.Genre
```

Query Evaluation

Book	Shop	Date	Cnt	TotP
Winnie	854	11/11	1	20
C	876	12/11	2	10
C	854	12/11	1	5
Winnie	876	13/11	3	60
Winnie	954	11/11	1	20
C	976	12/11	2	10
C	994	12/11	1	5
		:		

Book	Shop	Profit
Winnie	854	600
C	876	200
C	854	200
Winnie	876	1000

Q(F) $>>_{time}$  **$Q^{ref}(V)$**

Materialized Views

Pro :

- can reduce query time
- computation done once
- reused by multiple queries

Cons :

- needs storage space
- needs to be maintained
- how to choose the best views ?
  - DB-admin or automatic?
- needs reformulation algorithms

Query Reformulation Using Views
Problem

Input

- A (set of) relations **F**
- A (set of) views **V**
- A (set of) user queries **Q**

Output

- $\mathbf{Q}^{\mathbf{ref}}$ such that $\mathbf{Q}^{\mathbf{ref}}(\mathbf{V}) = \mathbf{Q}(\mathbf{F})$

View **Relevance** (for Query Reformulation)

- A view has sum(sales) by **genre** and by **year** for books introduced after 1991

Genre	Year	SumSales
Children	1991	600
Cooking	1992	800

- is it useful for sum(sales) by genre for products introduced after 1992 ?

View **Relevance** (for Query Reformulation)

- A view has sum(sales) by **genre** and by **year** for books introduced after 1991

Genre	Year	SumSales
Children	1991	600
Cooking	1992	800

- is it useful for sum(sales) by genre for products introduced before 1991 ?

View **Relevance** (for Query Reformulation)

- A view has sum(sales) by **genre** and by **year** for books introduced after 1991

Genre	Year	SumSales
Children	1991	600
Cooking	1992	800

- is it useful for sum(sales) by region ?

View **Relevance** (for Query Reformulation)

- A view has sum(sales) by **genre** and by **year** for books introduced after 1991

Genre	Year	SumSales
Children	1991	600
Cooking	1992	800

- is it useful for sum(sales) by genre ?

Query Reformulation

Possible to reformulate a query with a view when :

- The view has selection-conditions
  that are compatible with those of the queries

- The view has dimensions
  that are compatible with those of the queries

- In this case, we say that the view is **relevant**.

Materialized-View Selection

Problem

- DW administrator knows typical workload

  W = { Q_1 … Q_n } n=20,50,100

  of the datawarehouse **D**.

Example :

- Q_1 = SELECT … GROUP BY Product, Date
- Q_2 = SELECT … GROUP BY Product, Shop

Materialized-View Selection
Problem

- Problem : how to select the **best**-set-of-views **V** able to answer **W** (via query reformulation)?

- Unfeasible by hand, needs automatic solution.

Materialized-View Selection

Problem

Input

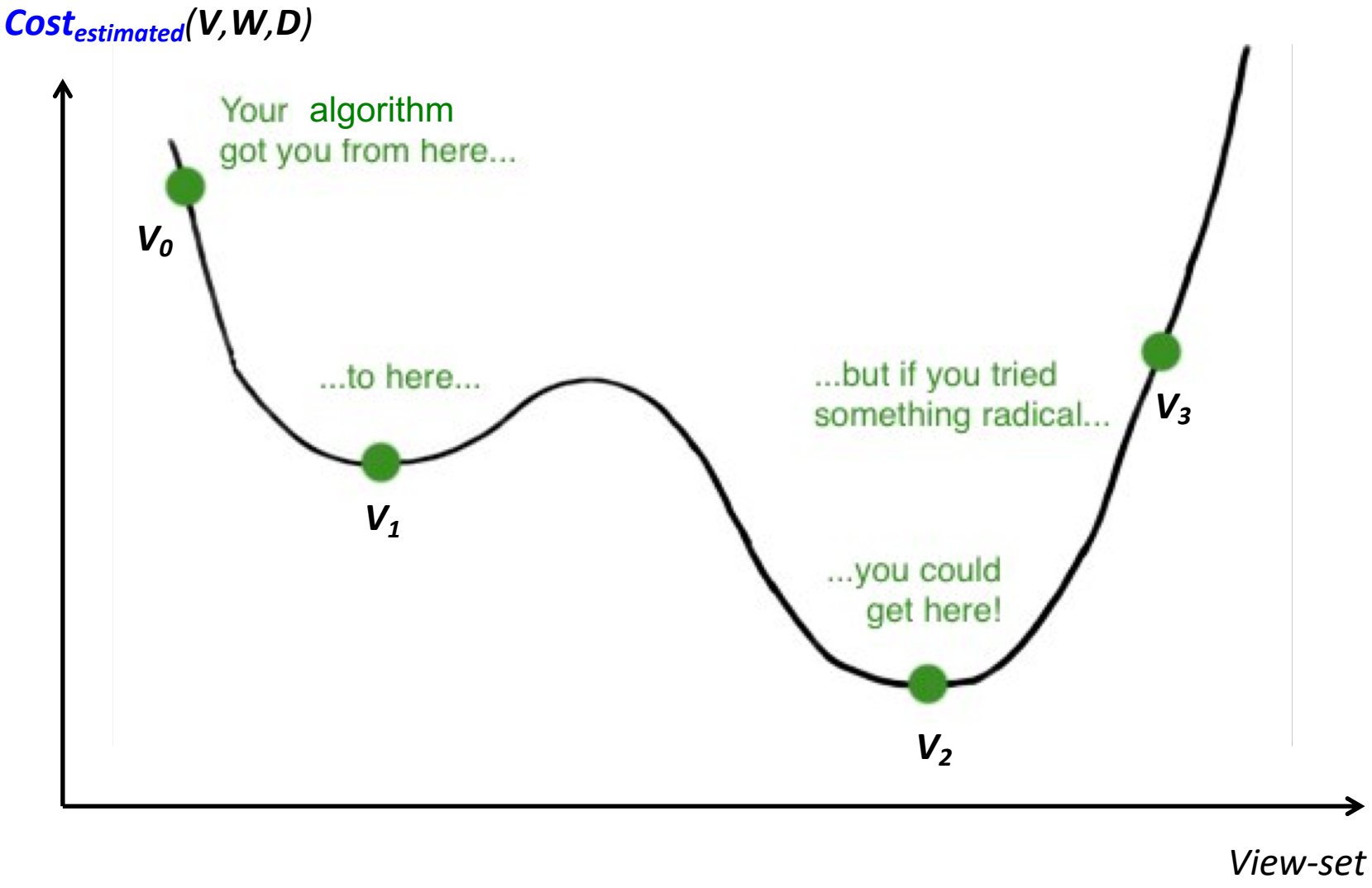
- workload **W**
- database **D**

$$Cost_{estimated}(V,W,D) = \alpha_{ref} * reformulation_{est} + \alpha_{occ} * occupancy_{est} + \alpha_{maint} * maintenance_{est}$$

Output

- A set of views **V** = { V_1 … V_m } minimizing (a linear combination of)
  1. query reformulation + execution time
  2. view space occupancy
  3. maintenance cost

Challenge : cost function is unpredictable



Materialized View Selection
Naïve Algorithm

1. For all possible view set **V**

2. Compute *Cost(**V,W,D**)*

3. Keep **V** with lowest cost

Finds optimal solution, but in an infinite time.

Materialized View Selection
Naïve Algorithm

1. For all candidate view set **V**
2. Compute *Cost(**V,W,D**)*
3. Keep **V** with lowest cost

Candidate View-Set

Definition.

A set of views **V** is *candidate* for **W** and **D** if

(1) it allows to answer all queries in **W** by means of reformulation

(2) every query of **W** admit a unique rewriting with **V**

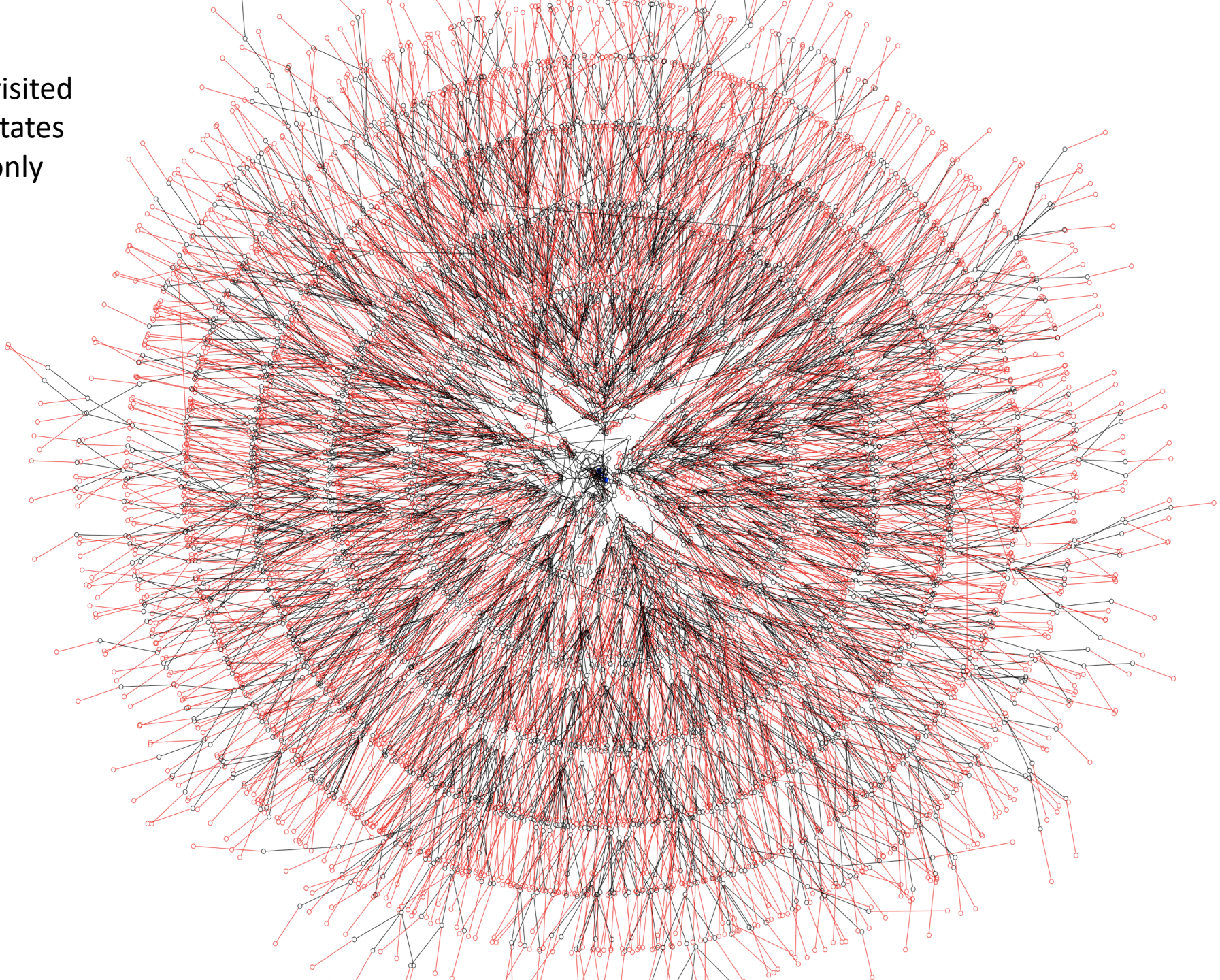
(3) **V** does not contain “useless” views

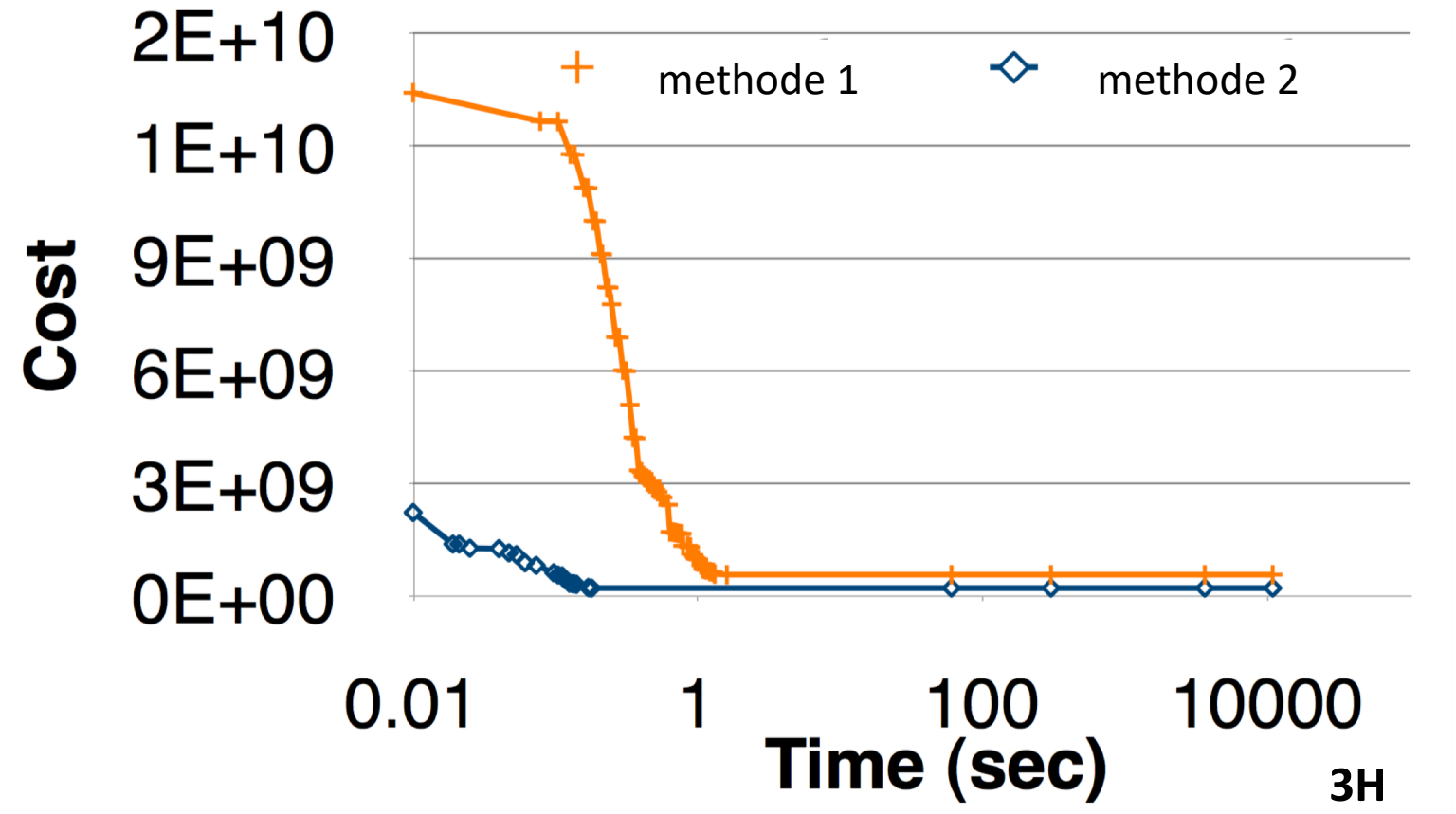
Materialized View Selection
Naïve Algorithm

1. For all candidate view set **V**
2. Compute *Cost(**V,W,D**)*
3. Keep **V** with lowest cost

We still need to explore a very large space of solutions.

visited
states
only



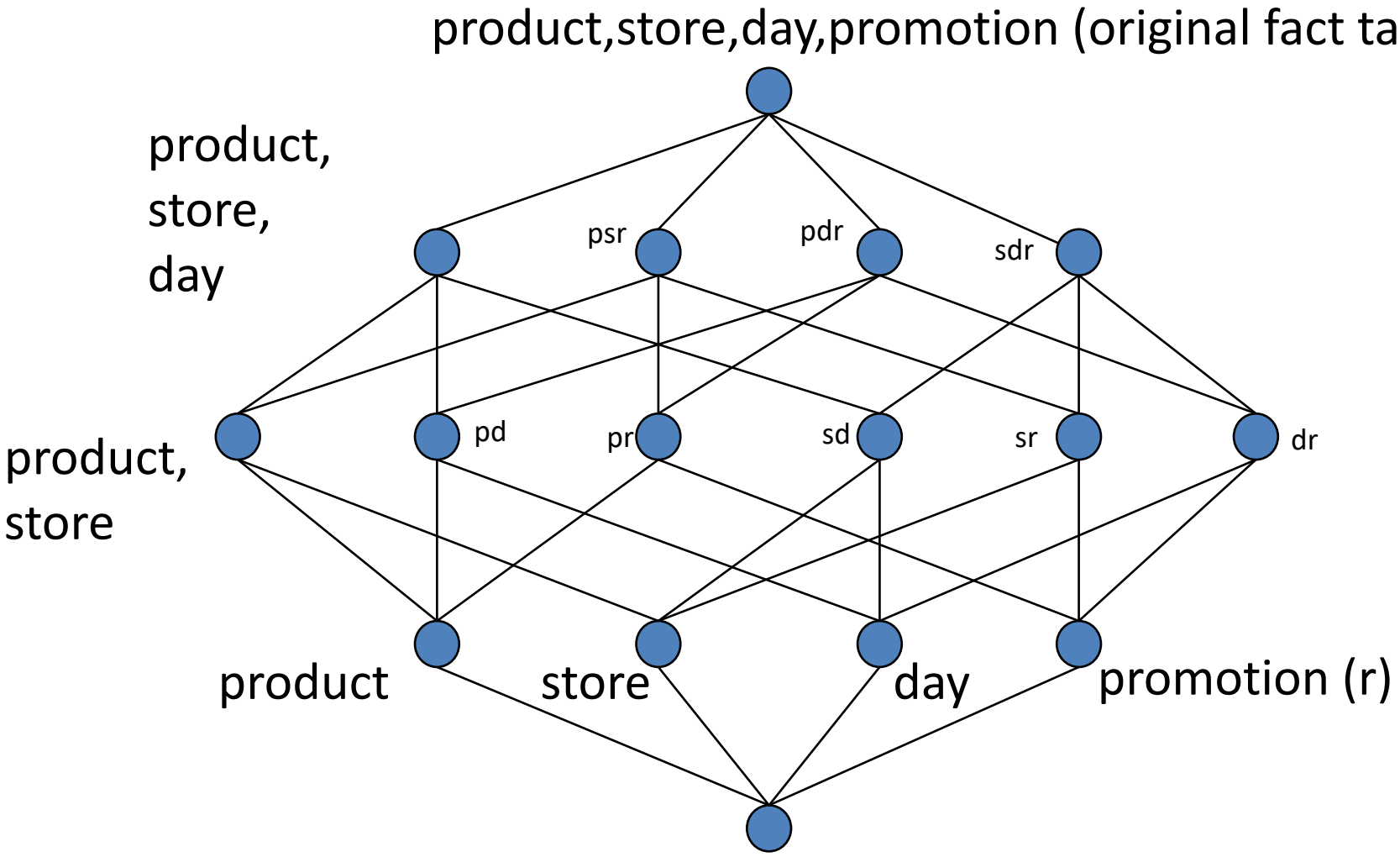

2E+10
1E+10
9E+09
6E+09
3E+09
0E+00
Cost
methode 1
methode 2
0.01
1
100
10000
Time (sec)
3H

Solution : split the problem in two

- Consider a simplified view-selection problem where one considers only group-by clauses

- All filtering conditions (like year>1991) are considered when the former subproblem is solved.

Aggregation Lattice

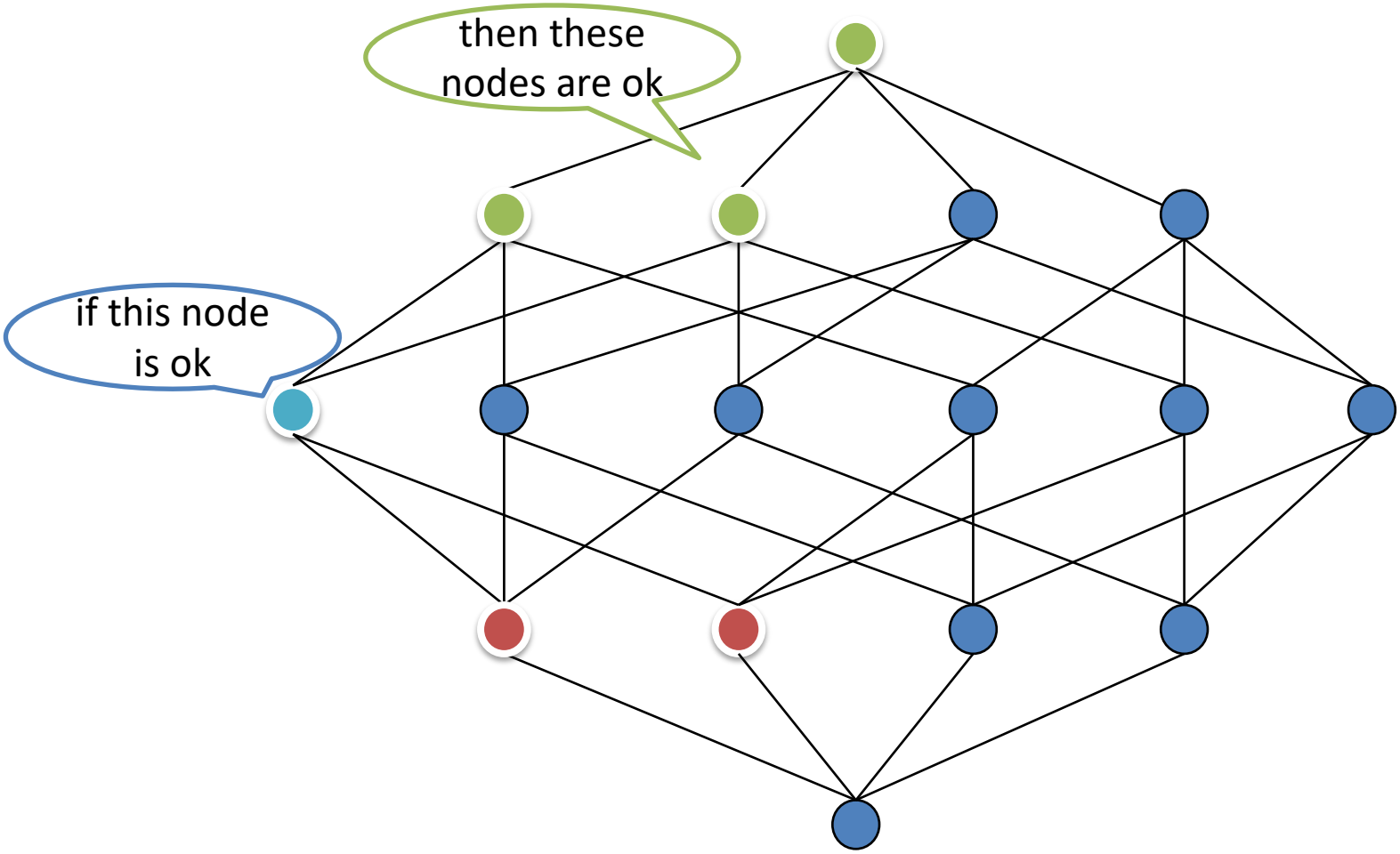


Aggregation Lattice

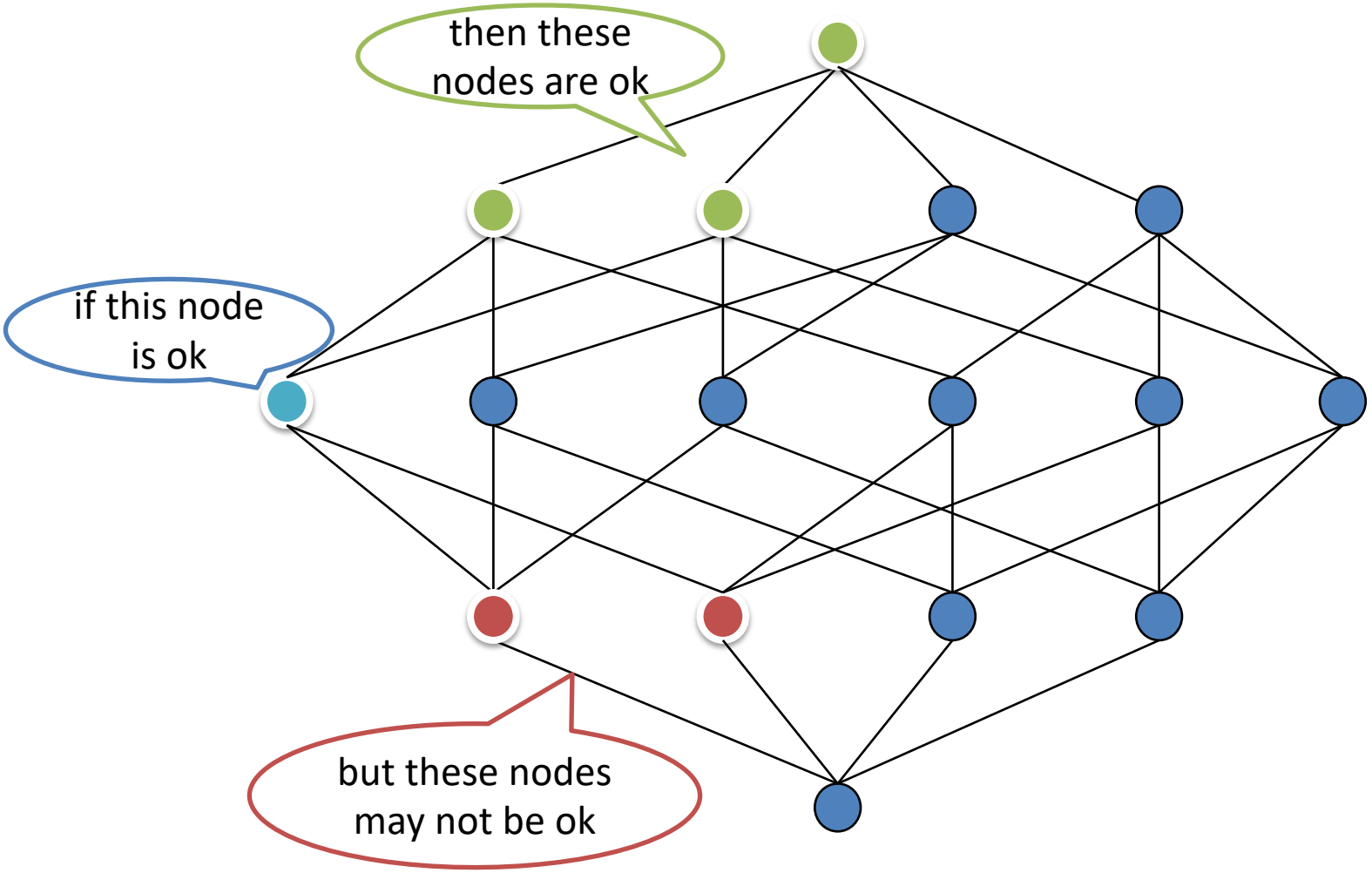
Property

A group-by query that can be answered by a node **n** of the lattice, can be answered by any ancestor of **n**.

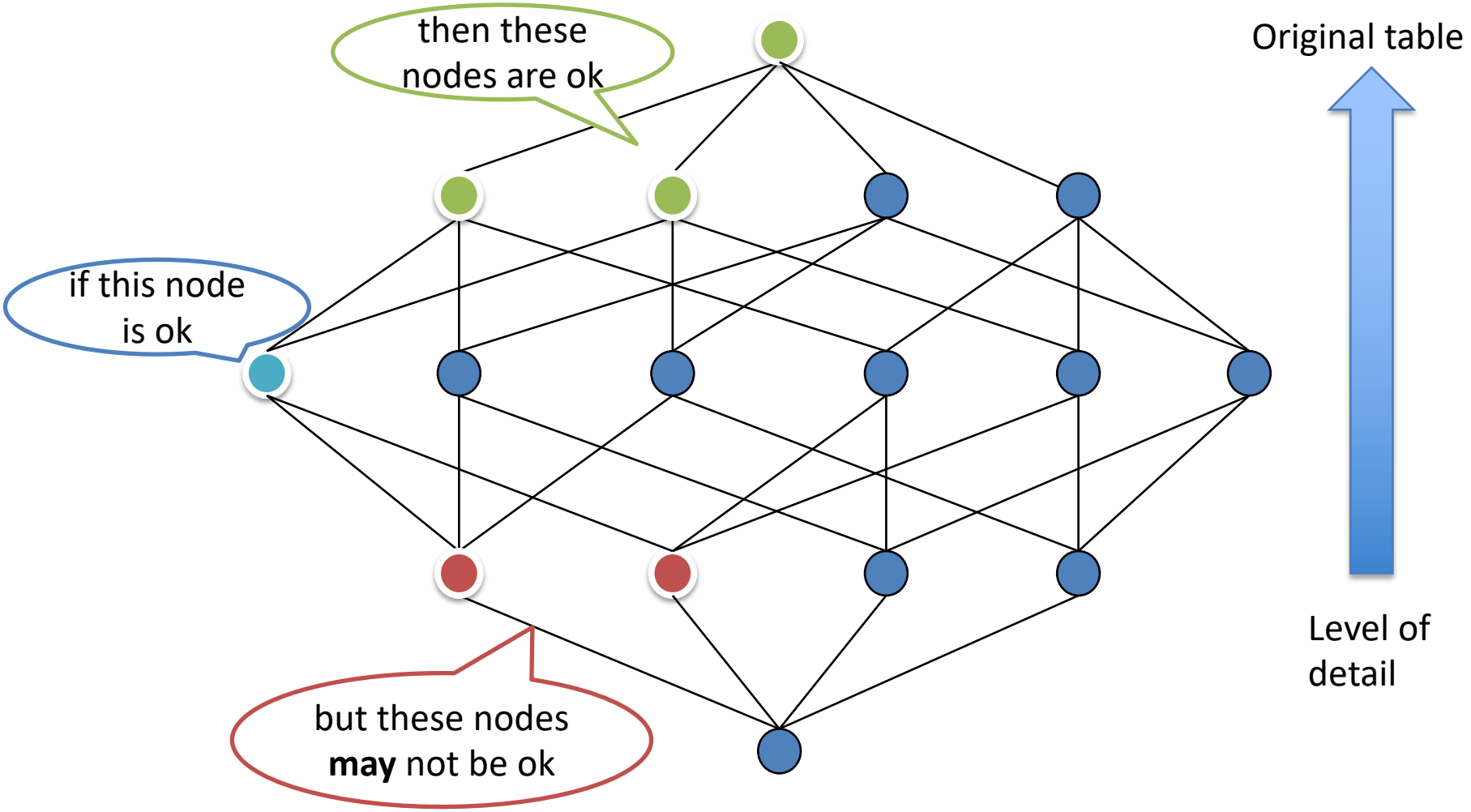
Aggregation lattice



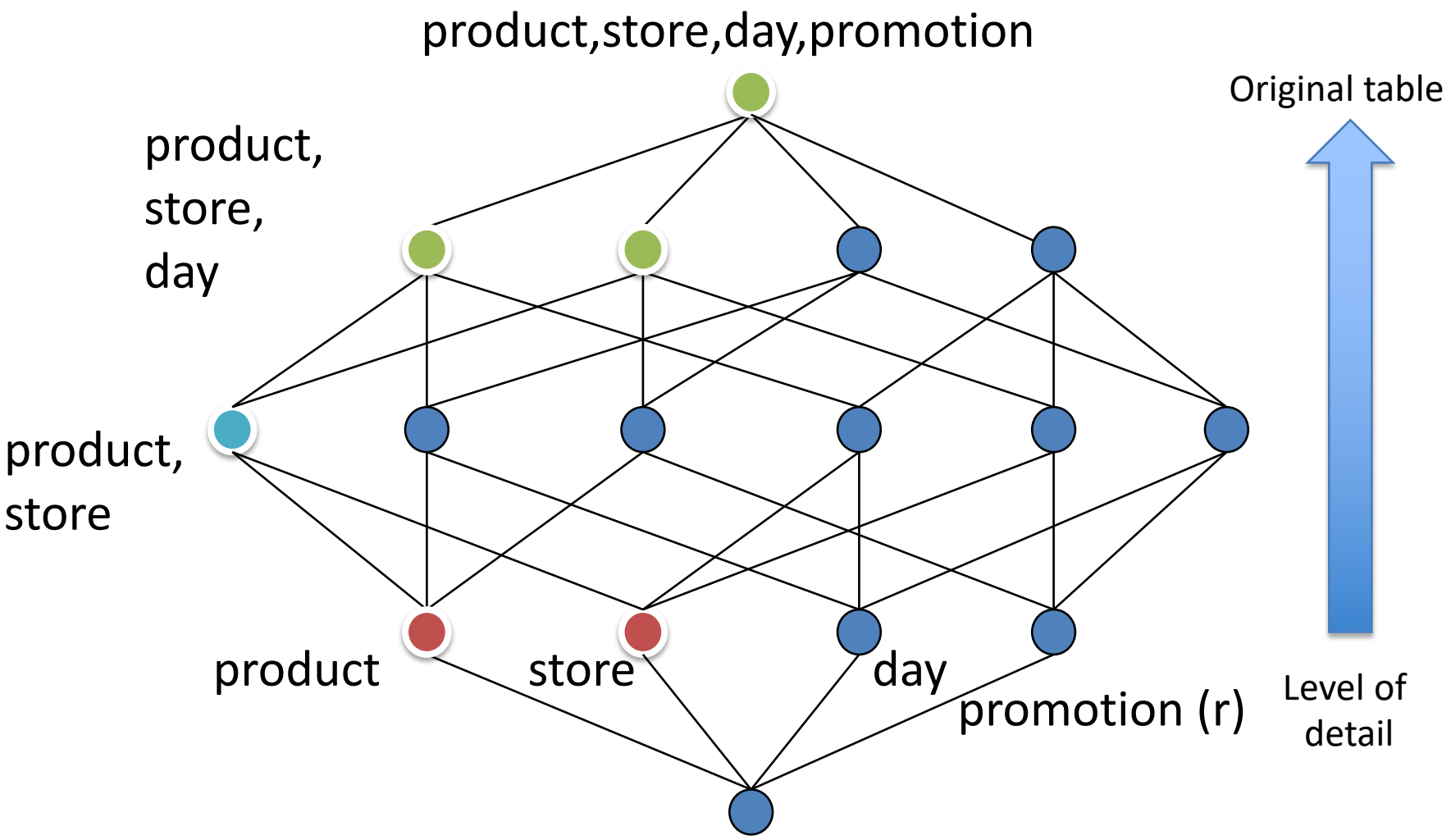
Aggregation lattice



Aggregation lattice



Aggregation lattice



Materialized-View Selection Algorithm

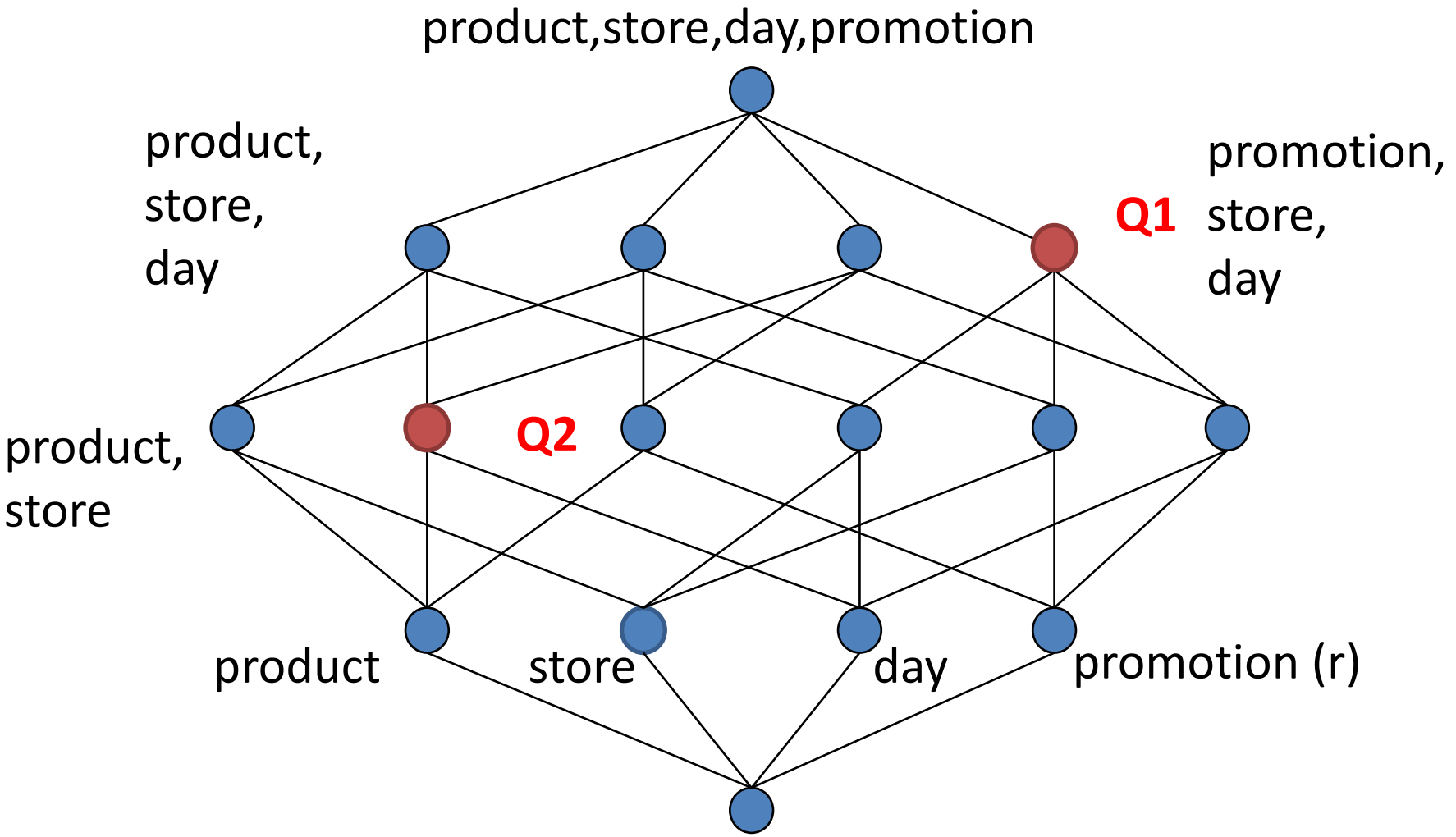
1. For each Q in **W**, find "lowest" node allowing one to answer Q

2. Prune irrelevant sub-lattice

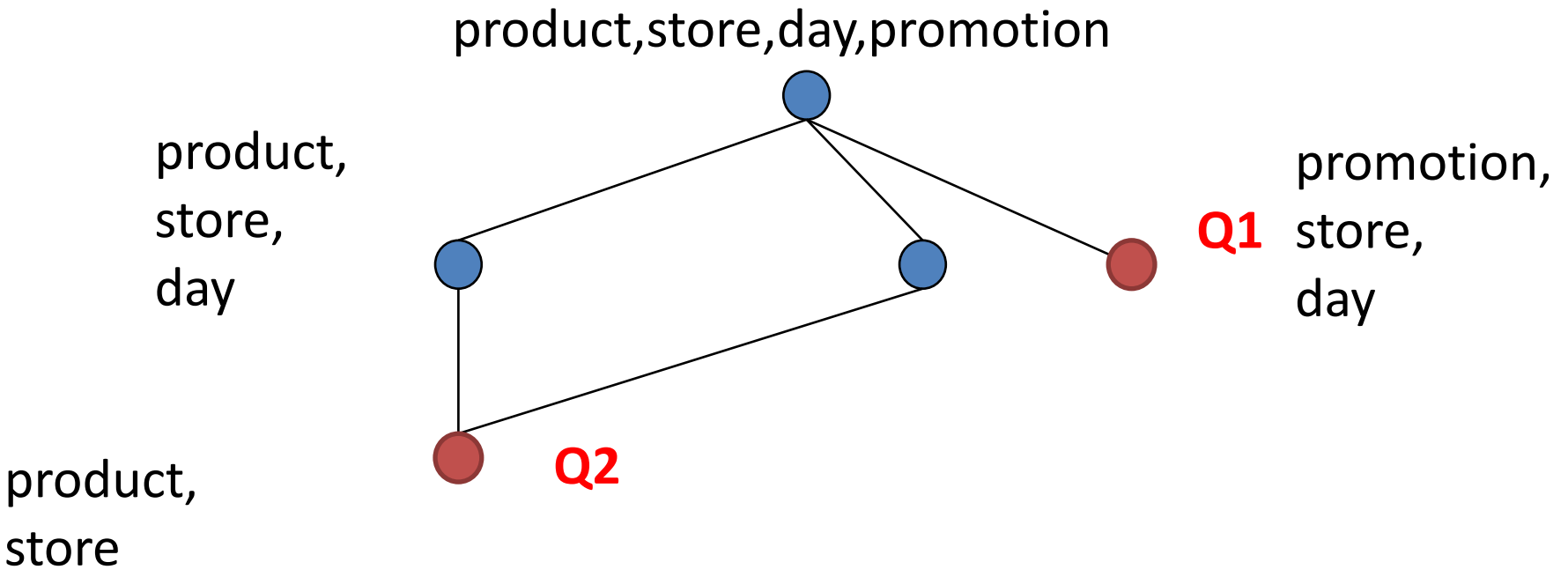
3. Evaluate cost function on every solution

Find lowest node for each Q

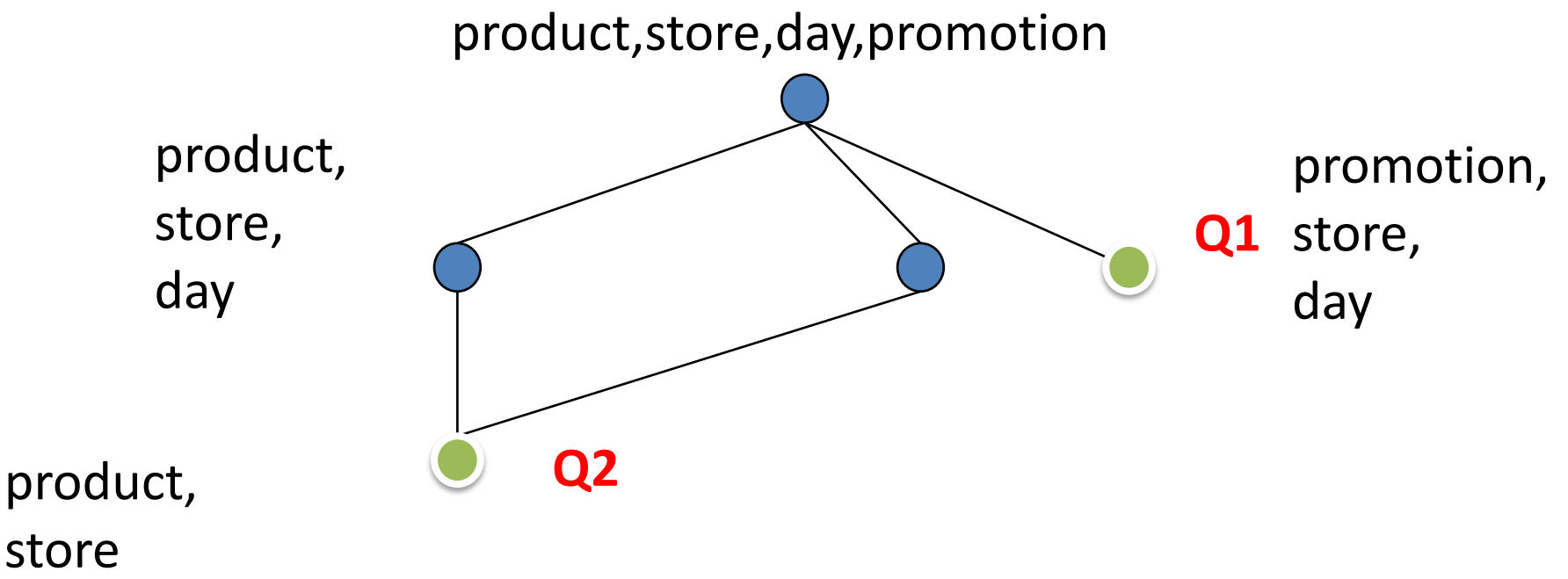
Find lowest node for each Q



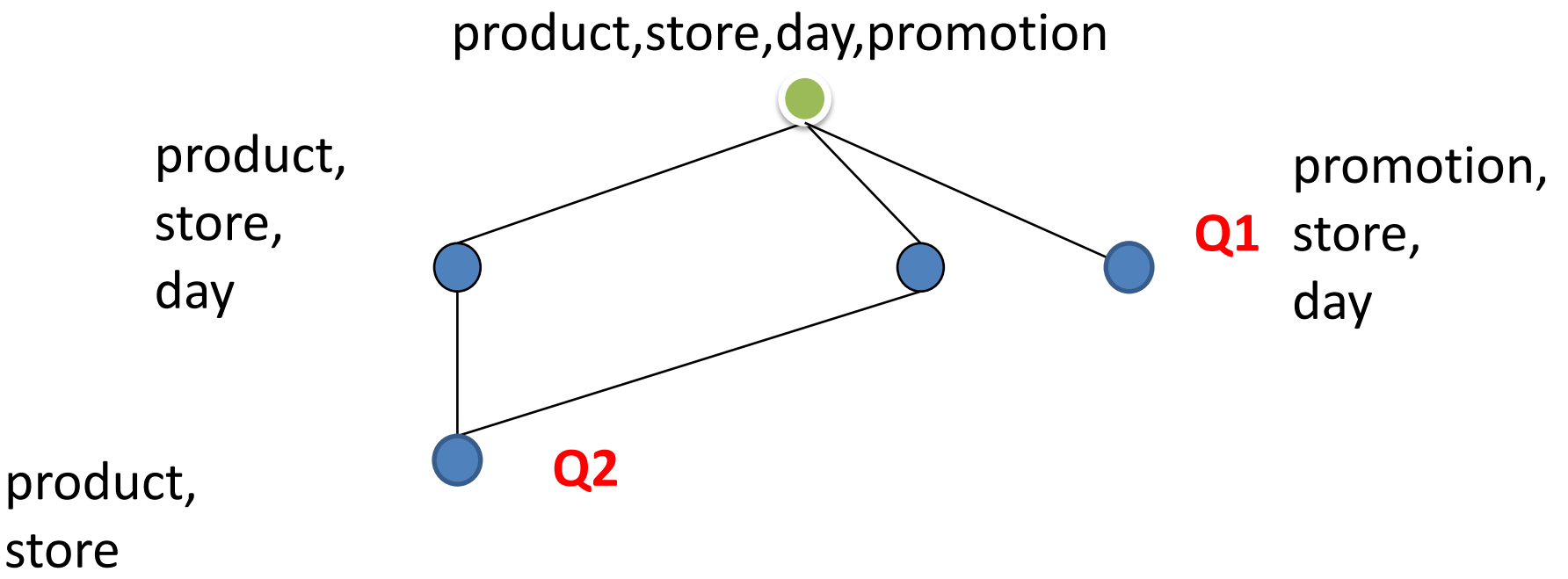
Prune irrelevant sub-lattice



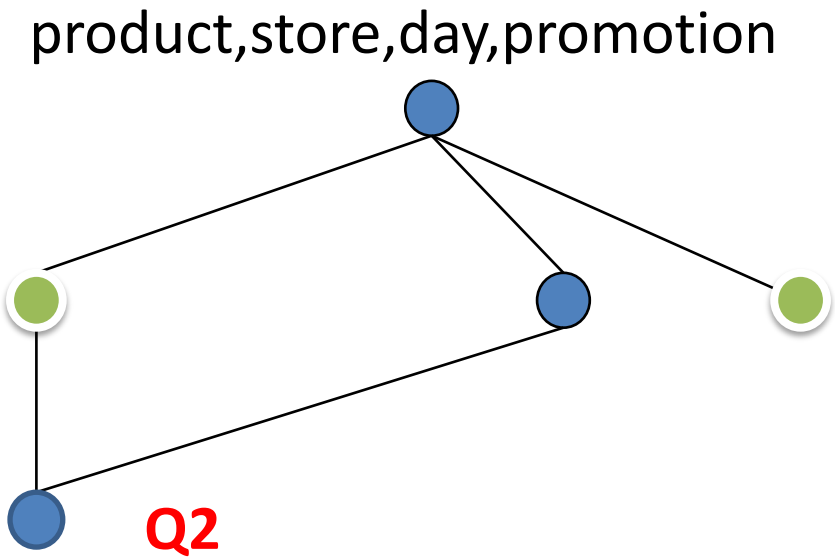
Evaluate cost function on every solution (1)



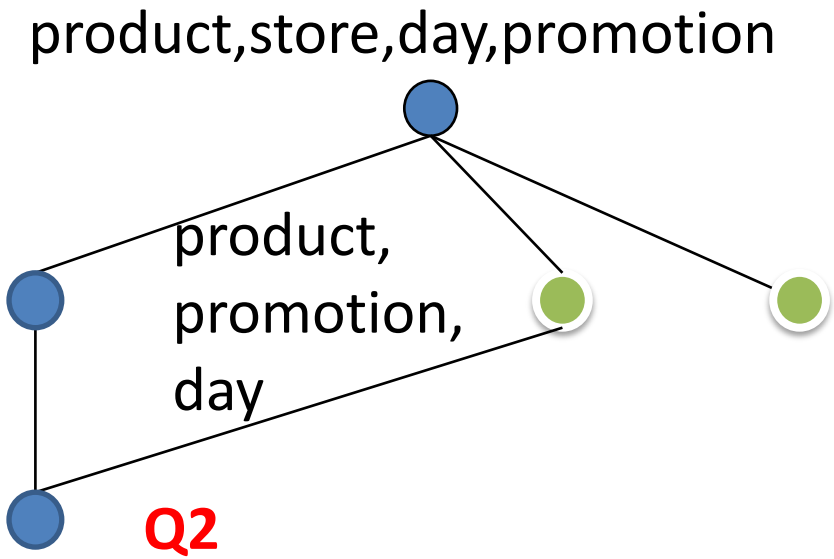
Evaluate cost function on every solution (2)



Evaluate cost function on every solution (3)



Evaluate cost function on every solution (4)

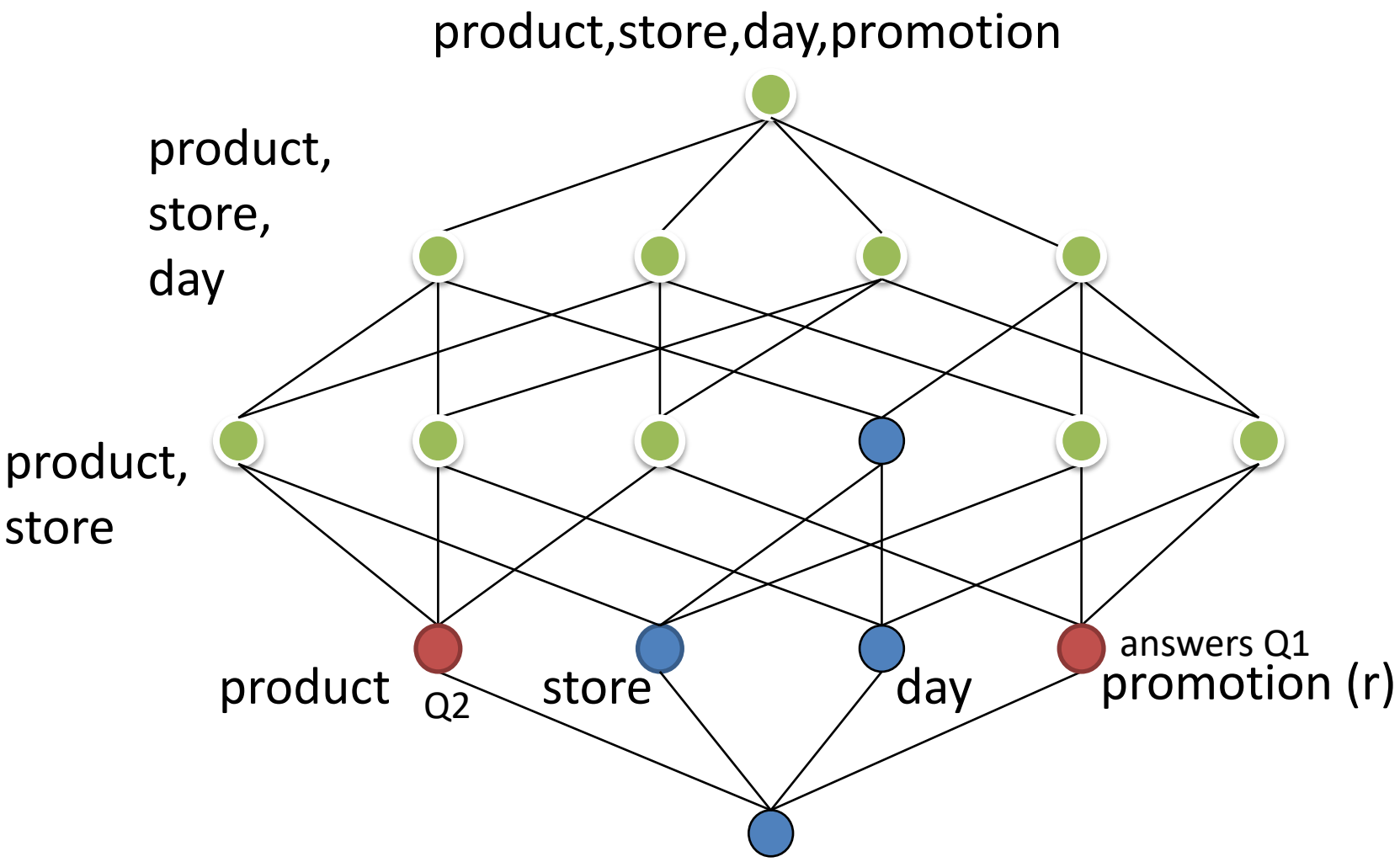


Materialized View Selection Algorithm

1. Find lowest node for each Q in **W**
2. Prune irrelevant sub-lattice
3. Evaluate cost function on any possible solution

Worst case : $O(2^{d})$ time (**d** = #dimensions of the fact)

Lattice has exponential size (in **d**)



Materialized View Selection Algorithm

1. Find lowest node for each Q in **W**
2. Prune irrelevant sub-lattice
3. Evaluate cost functions with an **approximated strategy**

PTIME approximation, but not guarantee of “optimality”.

INDEXING

Index Structures

- Traditional Access Methods
  - B+ trees
  - inverted lists
- Popular in Warehouses (for **dimensional** tables)
  - inverted lists
  - bit map indexes
  - join indexes

The tuple acces problem

- Input
  - a set of attributes
  - and value for each attribtue (eg employée_id = 17)
- Output
  - the tuples containg all input (attribute,value) pairs
    - sets of row_id offsets
    - key IDs
- Index : a data structure that solves this problem in constant or logarithmic time

B+ Trees

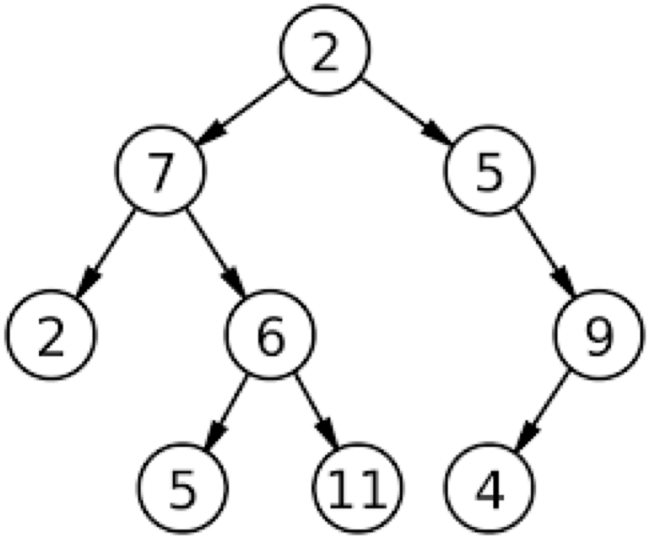
- Basic query optimization tasks must tackle queries like :

```
SELECT   Employee.name
  FROM   Employee
 WHERE   Employee.id=17
```

- access the tuple of employee 17 in constant time

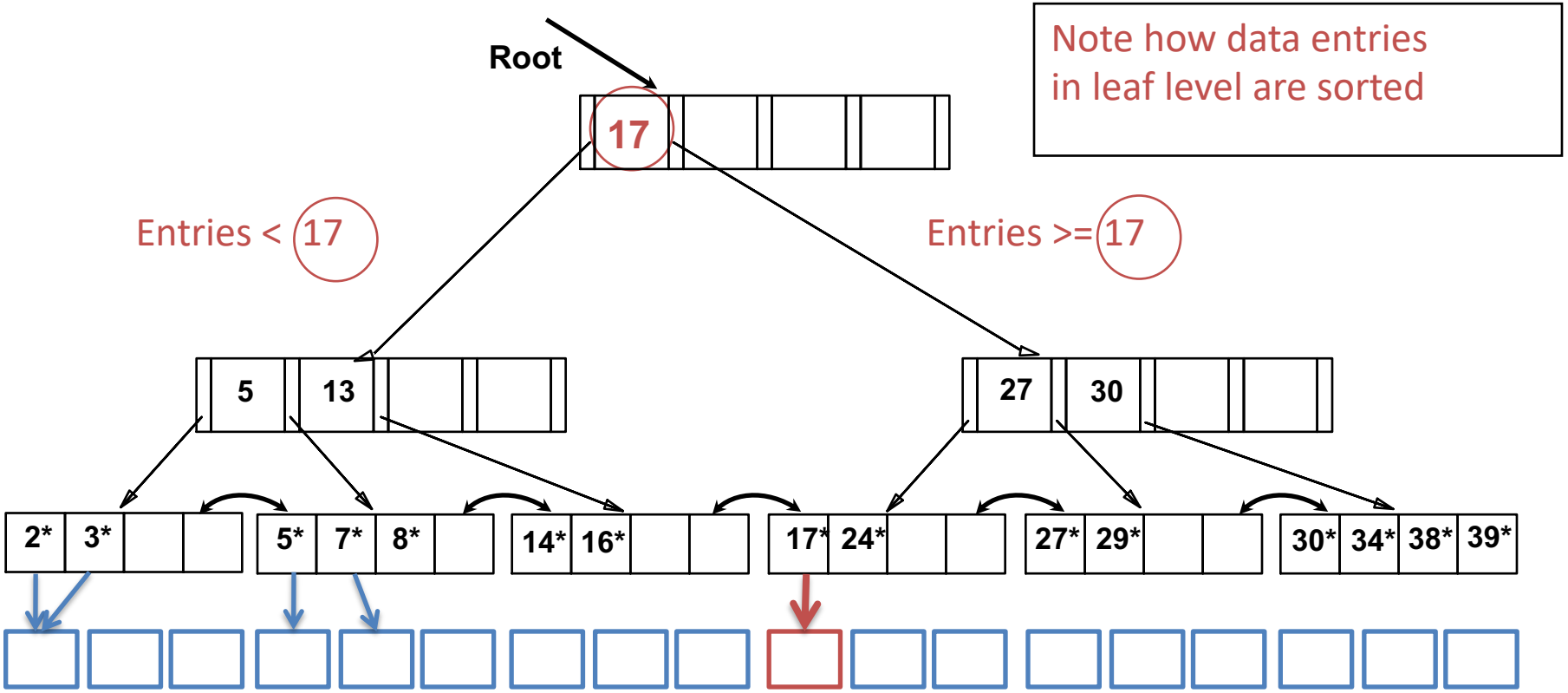
id	name
1	Alice
2	Bob
3	Charles
4	Bob
…	
17	Dylan
…	

Binary trees are not enough



- reduce deepnees → cut index access time cost

B+ Tree on Employee.id



Inverted Lists

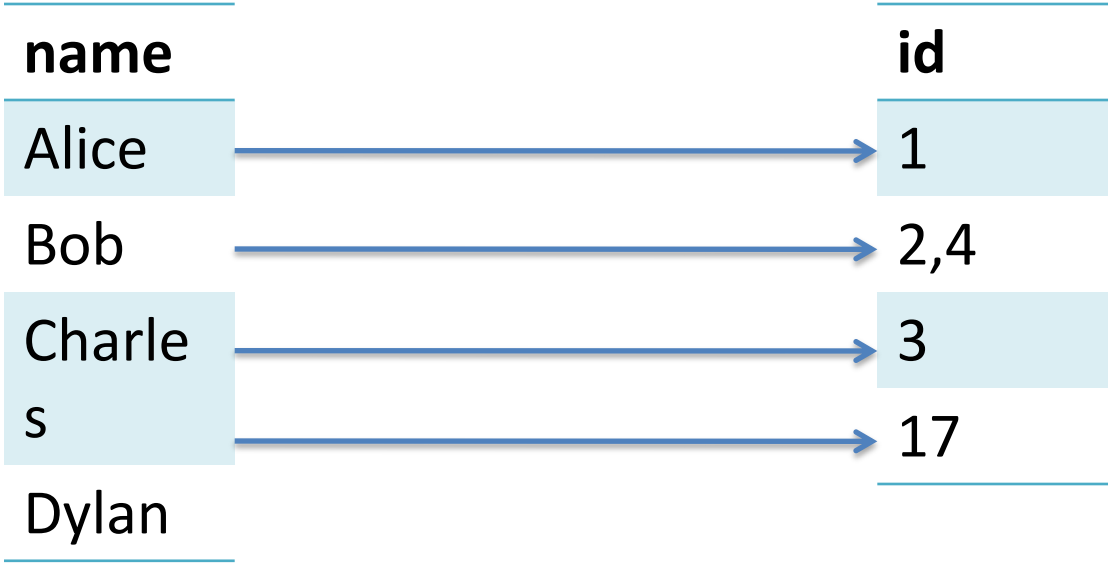
- Basic query optimization tasks on datawarehouse dimensions must tackle queries like :

```
SELECT   Employee.id
  FROM   Employee
 WHERE   name="Bob"
```

- access all tuples in constant time

id	name
1	Alice
2	Bob
3	Charles
4	Bob
…	
17	Dylan
…	

Inverted Lists



Combining Inverted Lists

- Query:
  - Get people with age = 20 and name = “Bob”
- List for age = 20
  - r2, r5, r18
- List for name = “Bob”
  - r2, r4
- Answer is intersection: r2

Bit Map Index

- Alternative representation of RecordID-list

Bitmap Index Example

custid	name	gender	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
116	Woo	M	4

Query : Find all customers with gender='M'

Bitmap Index Example

	custid	name	gender	rating
true	112	Joe	M	3
true	115	Ram	M	5
false	119	Sue	F	5
true	116	Woo	M	4

Query : Find all customers with gender='M'

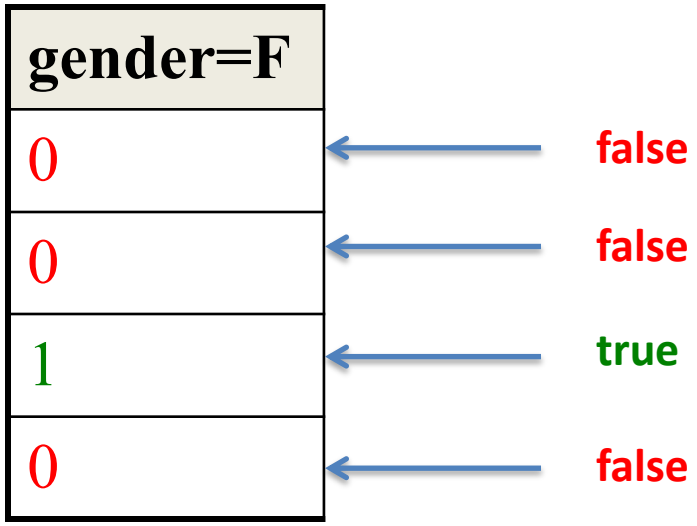
Bitmap Index Example

gender=M	
1	true
1	true
0	false
1	true

custid	name	gender	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
116	Woo	M	4

Query : Find all customers with gender='M'

Bitmap Index Example

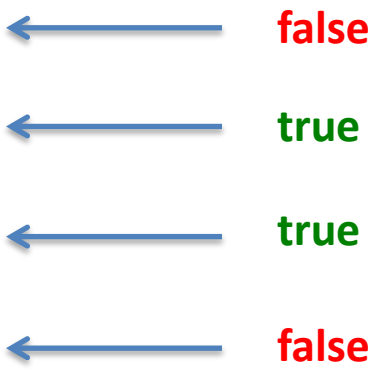


custid	name	gender	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
116	Woo	M	4

Query : Find all customers with gender='F'

Bitmap Index Example

rating=5
0
1
1
0



custid	name	gender	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
116	Woo	M	4

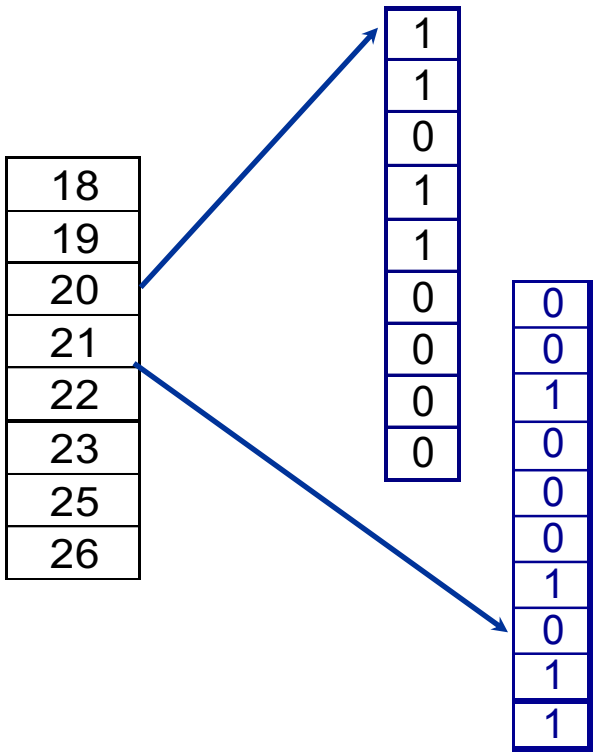
Query : Find all customers with rating = 5

Bitmap Index Example

custid	name	gender	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
116	Woo	M	4

rating=1	rating=2	rating=3	rating=4	rating=5
0	0	**1**	0	0
0	0	0	0	**1**
0	0	0	0	**1**
0	0	0	**1**	0

Bit Maps



age list

bit maps

id	name	age
1	joe	20
2	fred	20
3	sally	21
4	nancy	20
5	tom	20
6	pat	25
7	dave	21
8	jeff	26

data records

The magic of Booleans

- Query:
  - Get people with age = 20 and name = “Bob”

- List for age = 20: 1101100000
- List for name = “Bob”: 0100000001
- Answer is logical-& 0100000000

The magic of Booleans

- Query:
  - Get people with age = 20 OR age = 21

- List for age = 20: **11**0**11**00000
- List for age = 21: 00**1**0000000**1**
- Answer is logical-OR **1111**00000**1**

Bitmap indexes

- Advantageous for **low-cardinality domains**
  - Significant reduction in space and I/O
    Compression (e.g., run-length encoding) exploited
- Comparison, join and aggregation operations are reduced to *bit arithmetic*

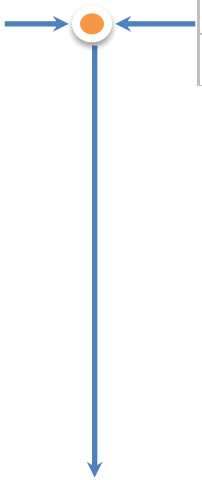
Joins

Fact_table_sales

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

Dim_table_product

product	id	name	price
	p1	bolt	10
	p2	nut	5



joinTb	prodId	name	price	storeId	date	amt
	p1	bolt	10	c1	1	12
	p2	nut	5	c1	1	11
	p1	bolt	10	c3	1	50
	p2	nut	5	c2	1	8
	p1	bolt	10	c1	2	44
	p1	bolt	10	c2	2	4

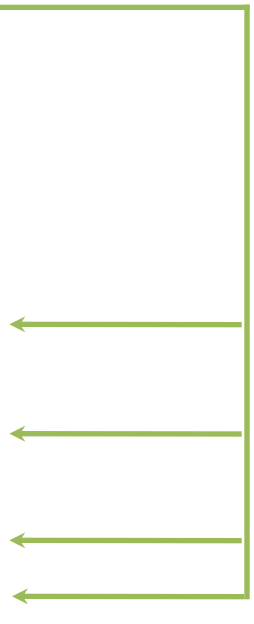
Join Indexes

Dim_table_product

join index

product	id	name	price	jIndex
	p1	bolt	10	r1,r3,r5,r6
	p2	nut	5	r2,r4

sale	rId	prodId	storeId	date	amt
	r1	p1	c1	1	12
	r2	p2	c1	1	11
	r3	p1	c3	1	50
	r4	p2	c2	1	8
	r5	p1	c1	2	44
	r6	p1	c2	2	4

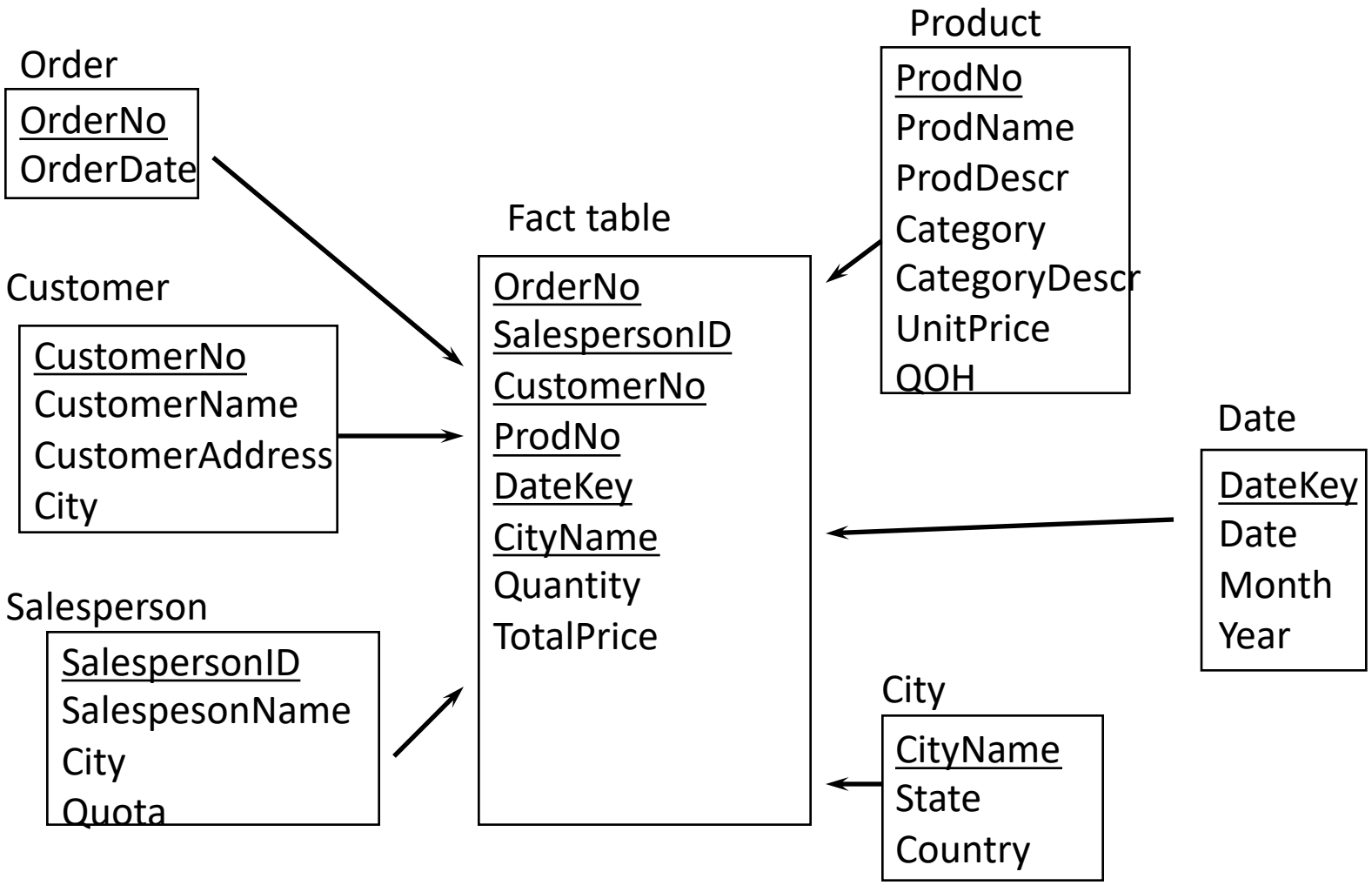


Join Index

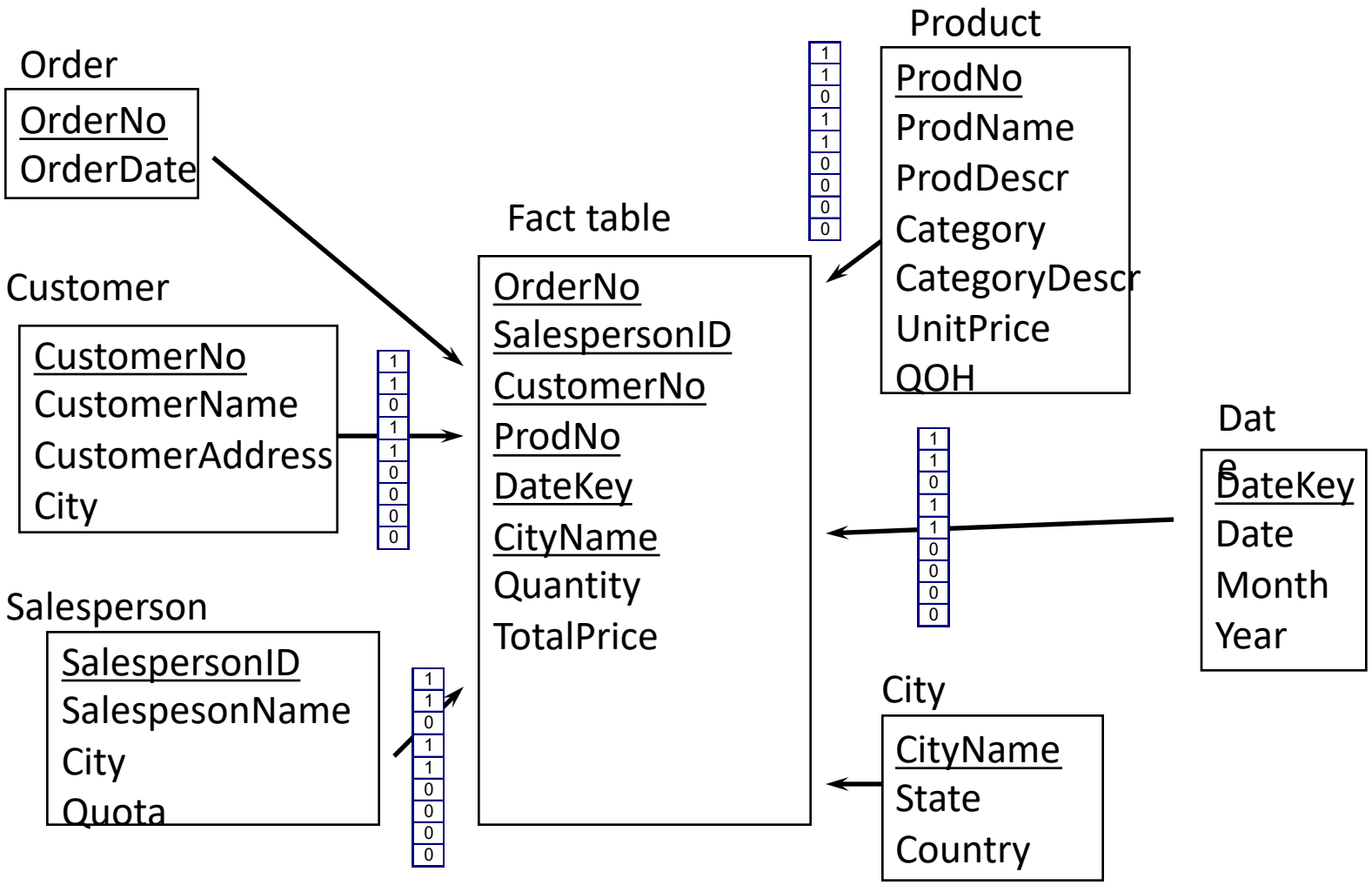
- Relationships between attribute value of a dimension and the matching rows in fact table

- Good news : it can be bitmapped !

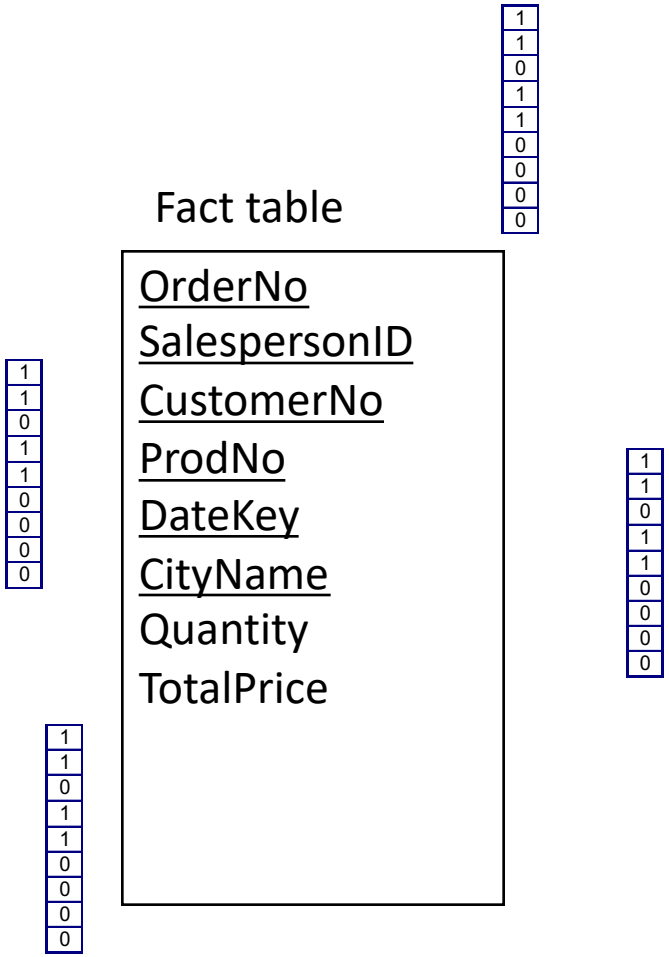
Join Index over Star Schema



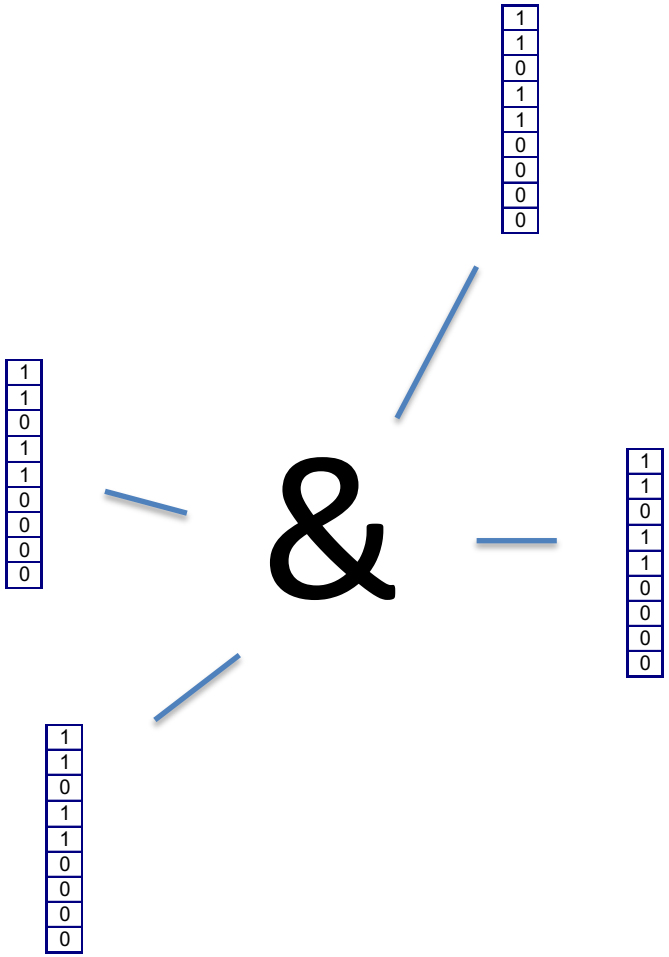
Join Index over Star Schema



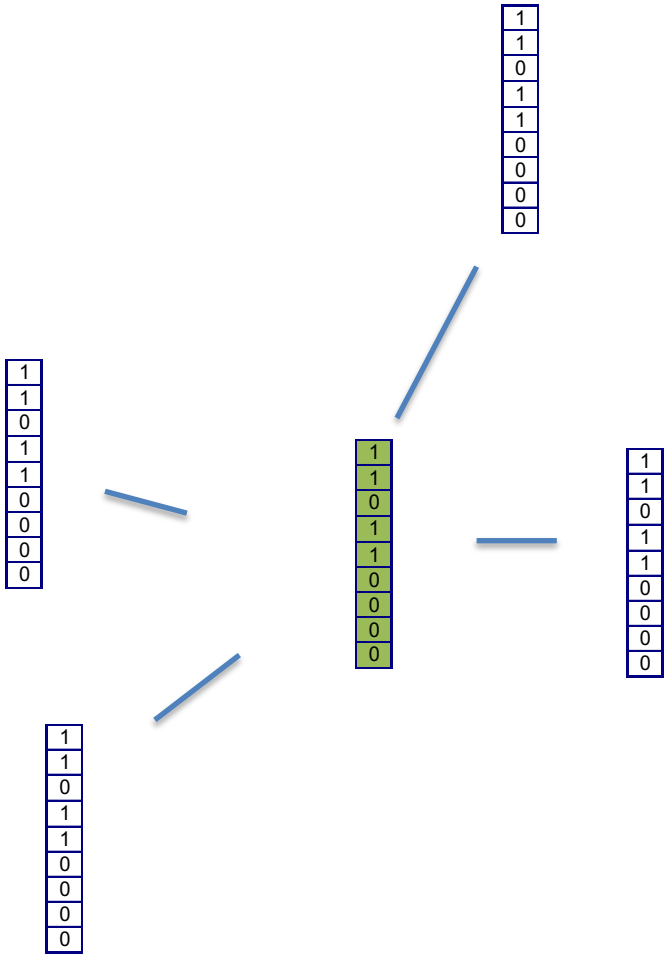
Join Index over Star Schema



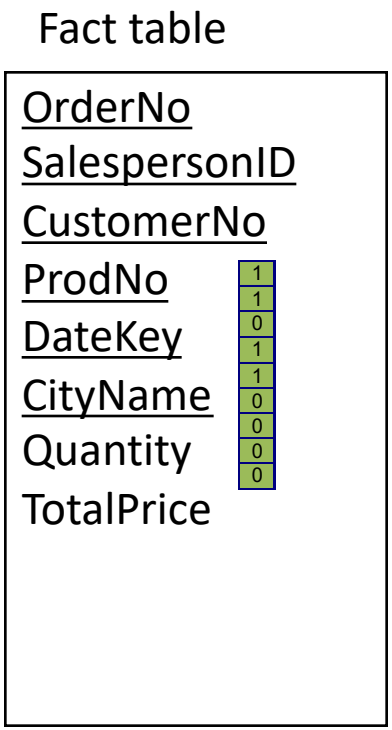
Join Index over Star Schema



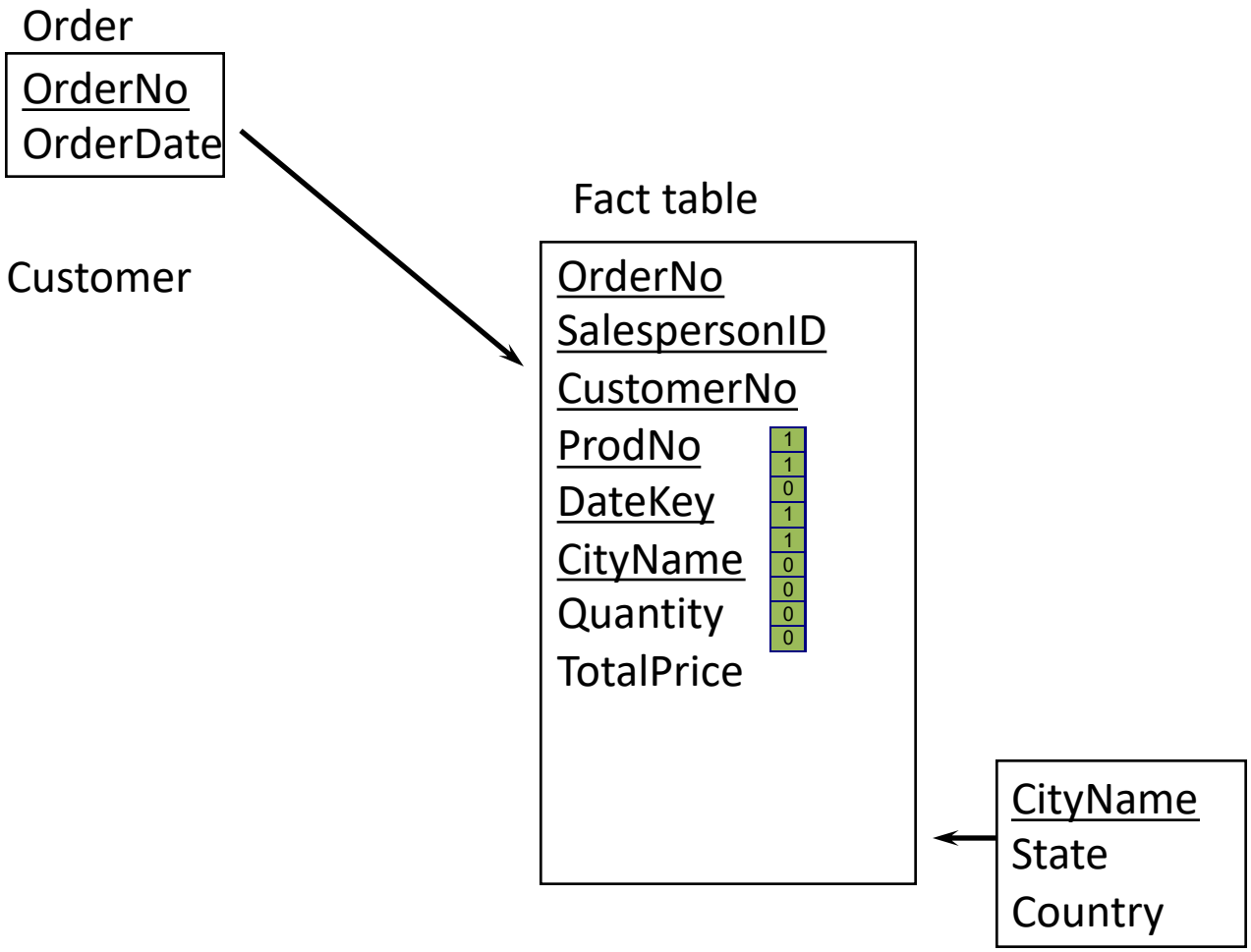
Join Index over Star Schema



Join Index over Star Schema



Join Index over Star Schema



Join-Algorithm Using Bitmapped Join-Indexes

1. Take intersection of join-indexes for all available dimensions so as to filter as many facts as possible

2. Join the remaining dimension on the resulting facts

3. Aggregate on the fact table

MULTIDIMENSIONAL DATABASES

From Tables to (hyper-)Cubes

- Datamarts are multidimensional models which views data in the form of a data (hyper-)cube

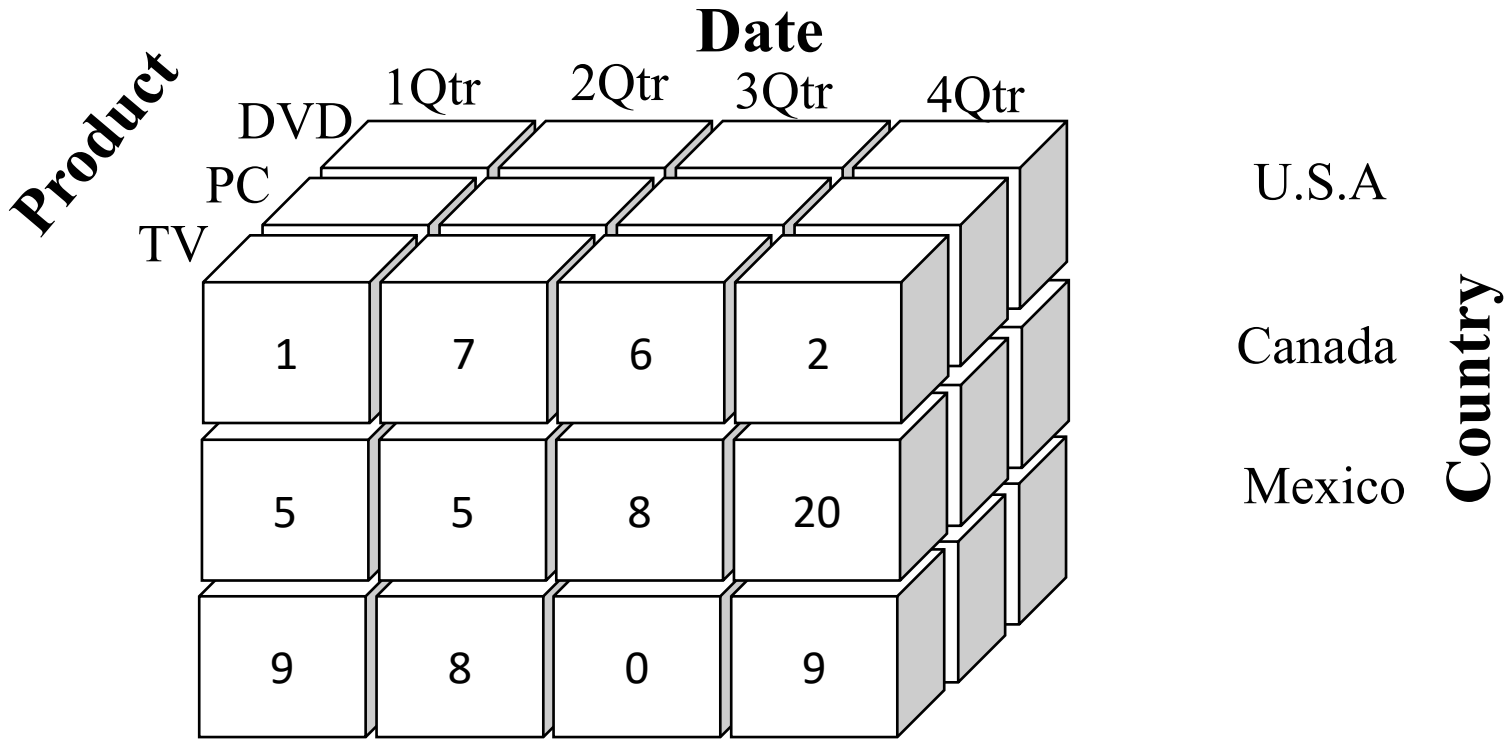
- We have seen relational datawarehouses, just one kind of “implementation” of this model

- Other implementations, may differ in the technological solution
  - for example, how to store sparse arrays (lots of zeros) with a matrix-based approach

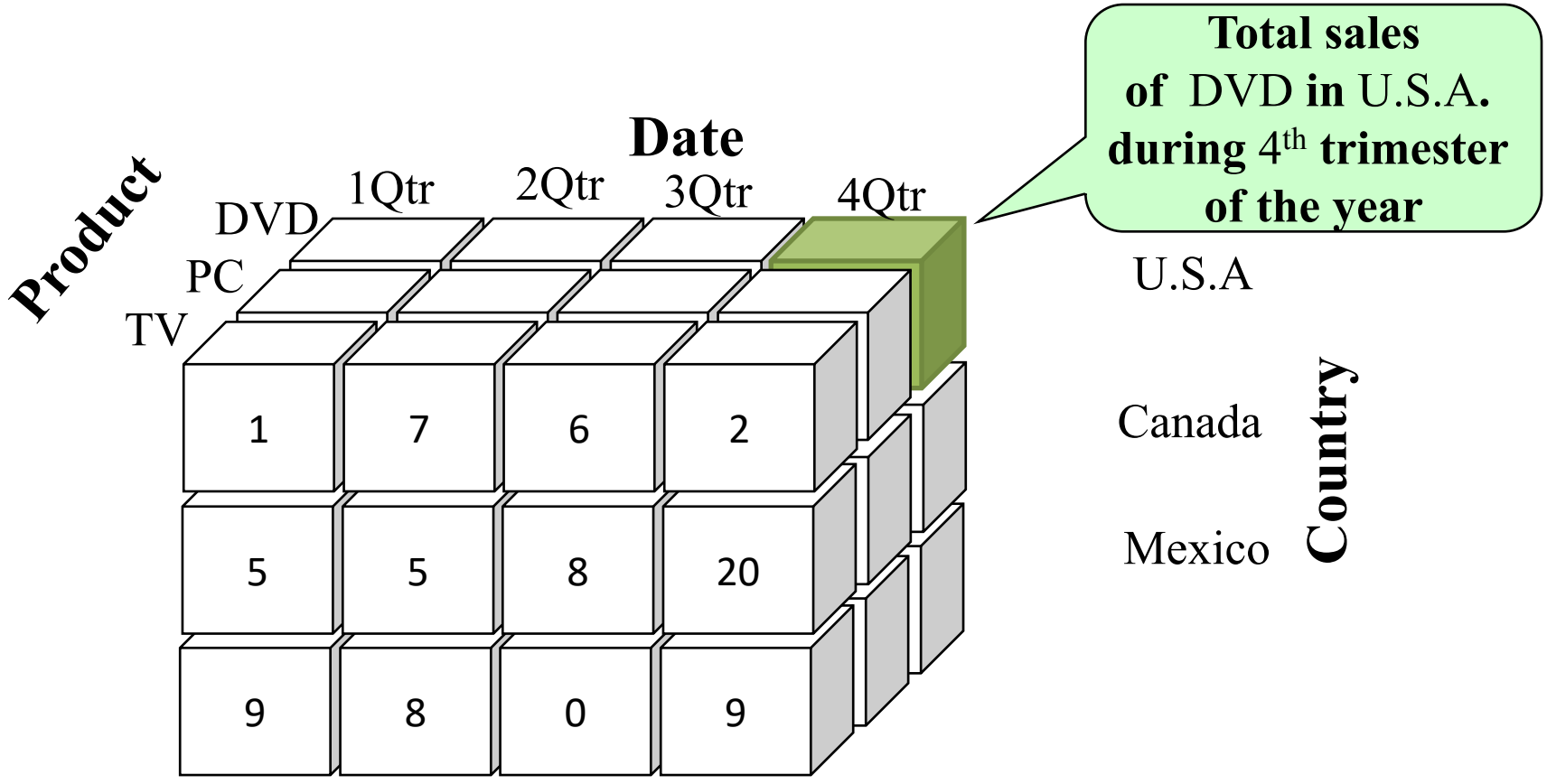
From Tables to (hyper-)Cubes

- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
  - Dimension tables, such as item (item_name, brand, type), or time(day, week, month, quarter, year)
  - Fact table contains measures (such as dollars_sold) and keys to each of the related dimension tables

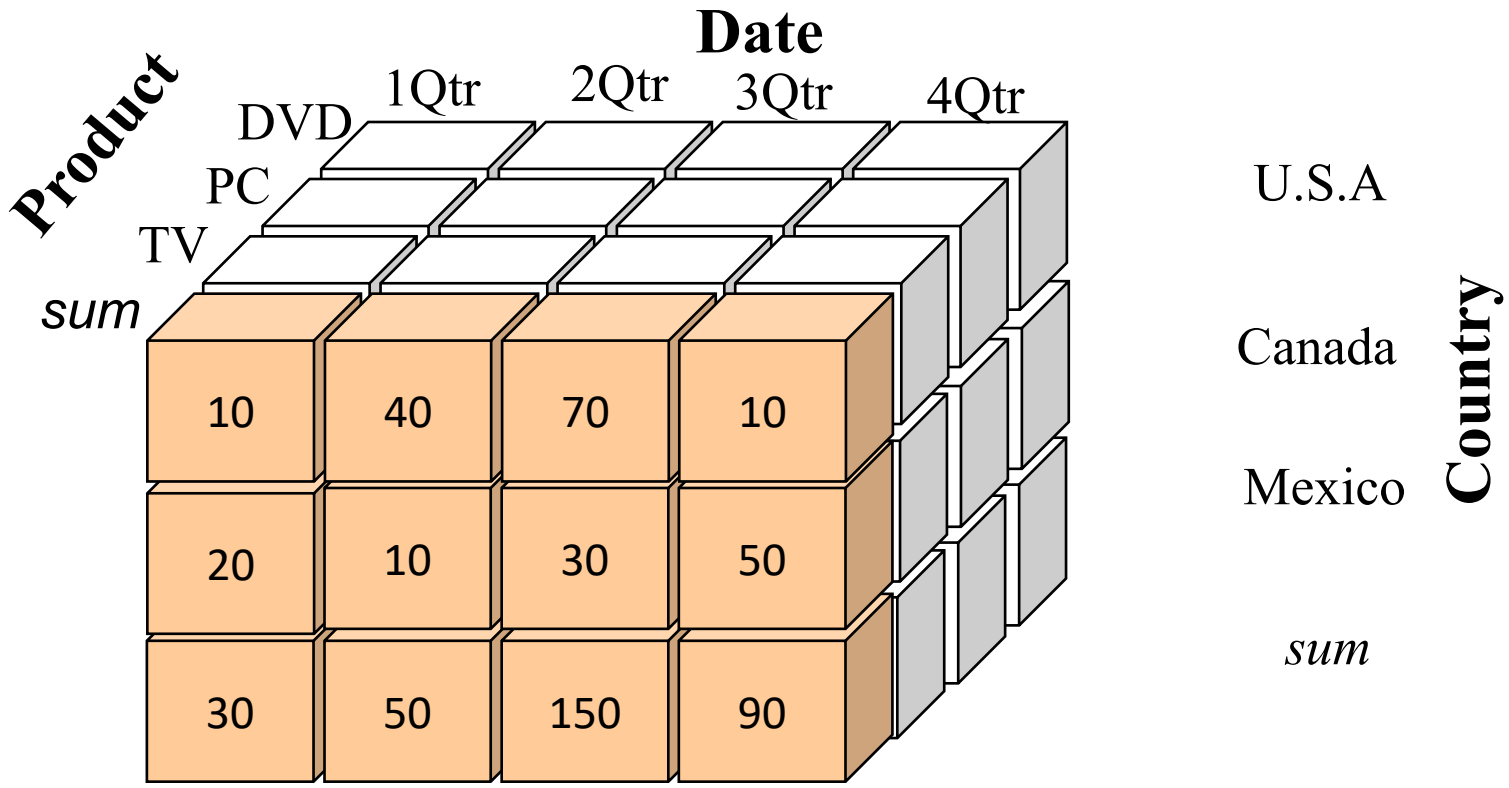
A Sample Data Cube



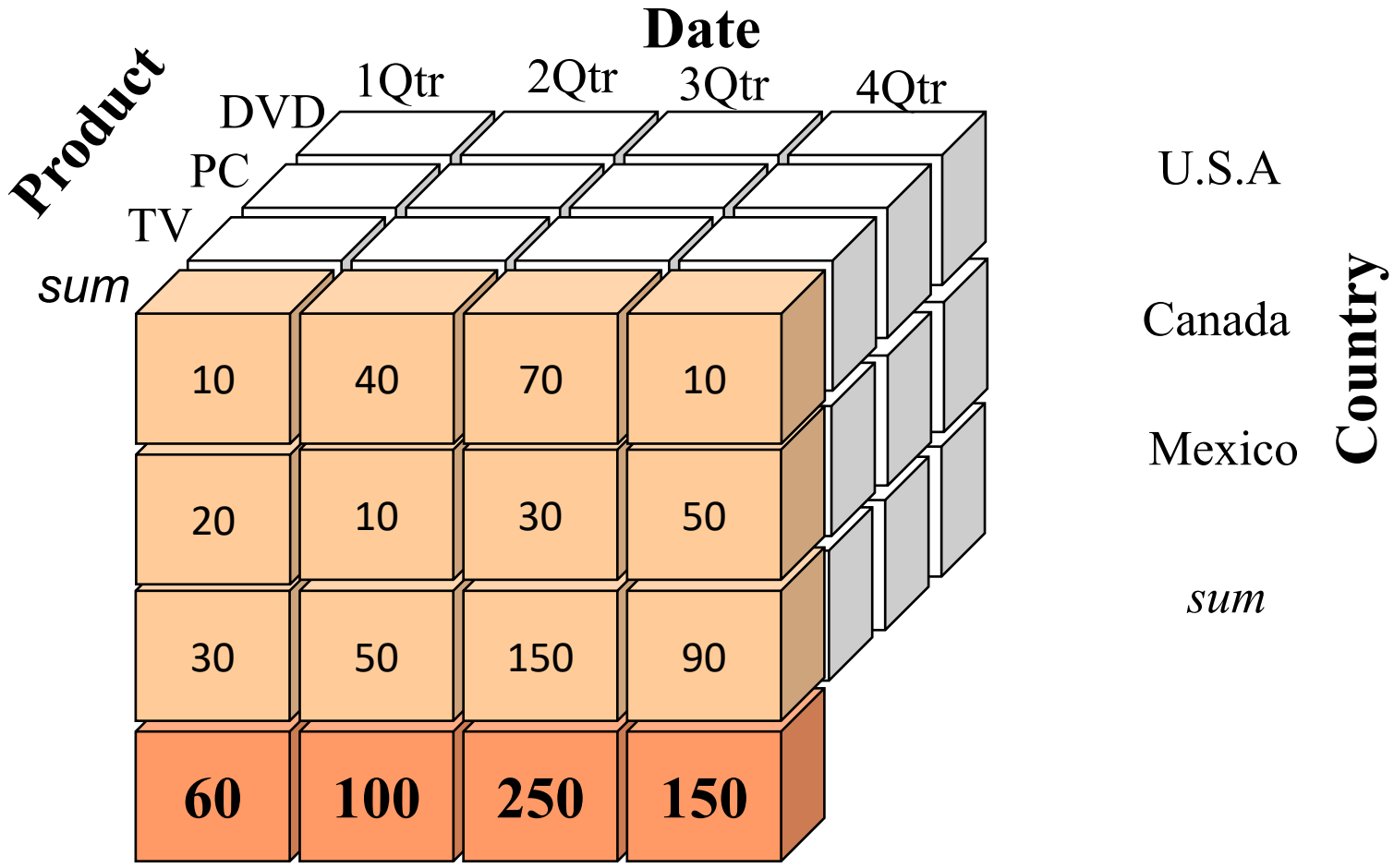
A Sample Data Cube



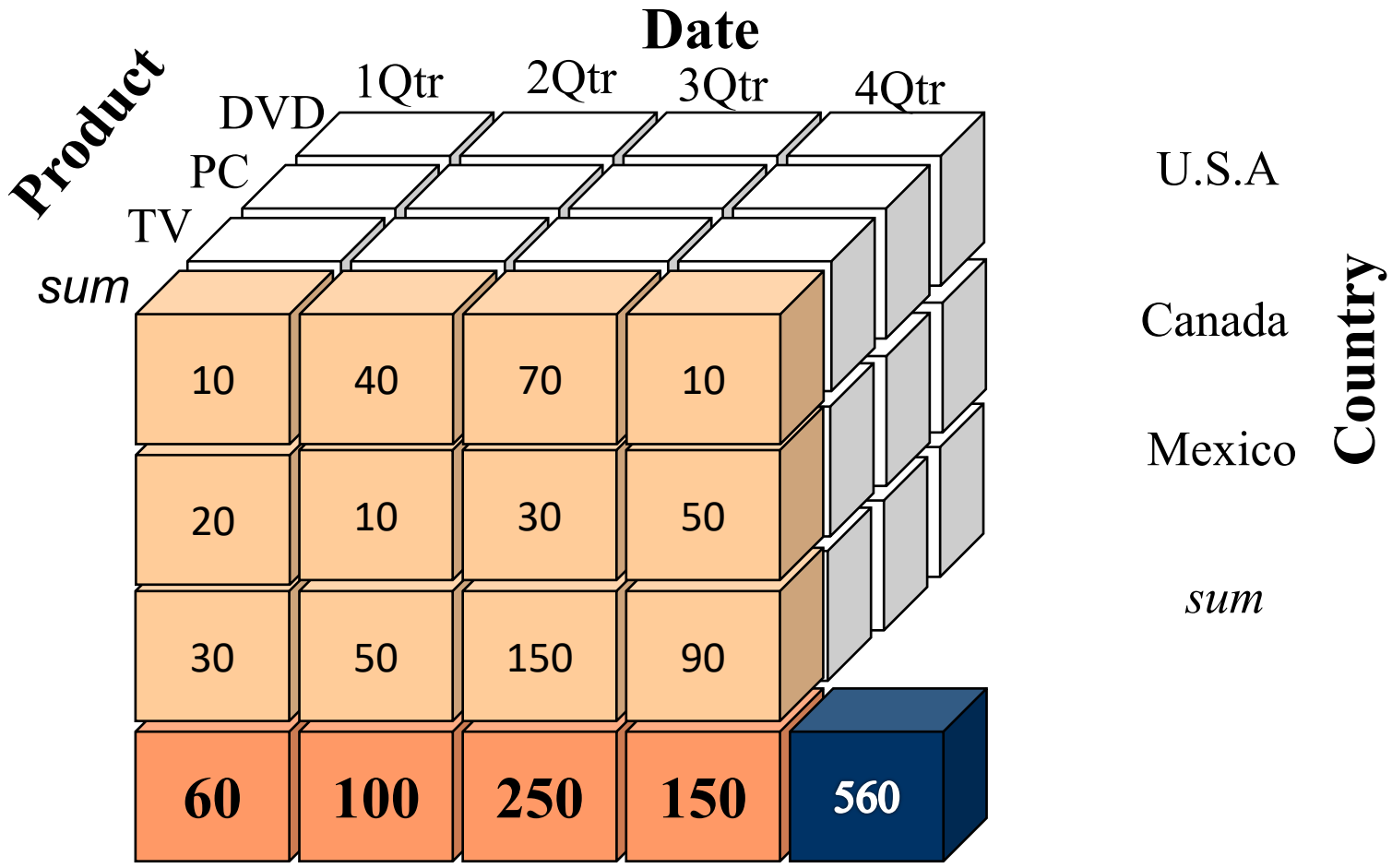
Group By Date, Country



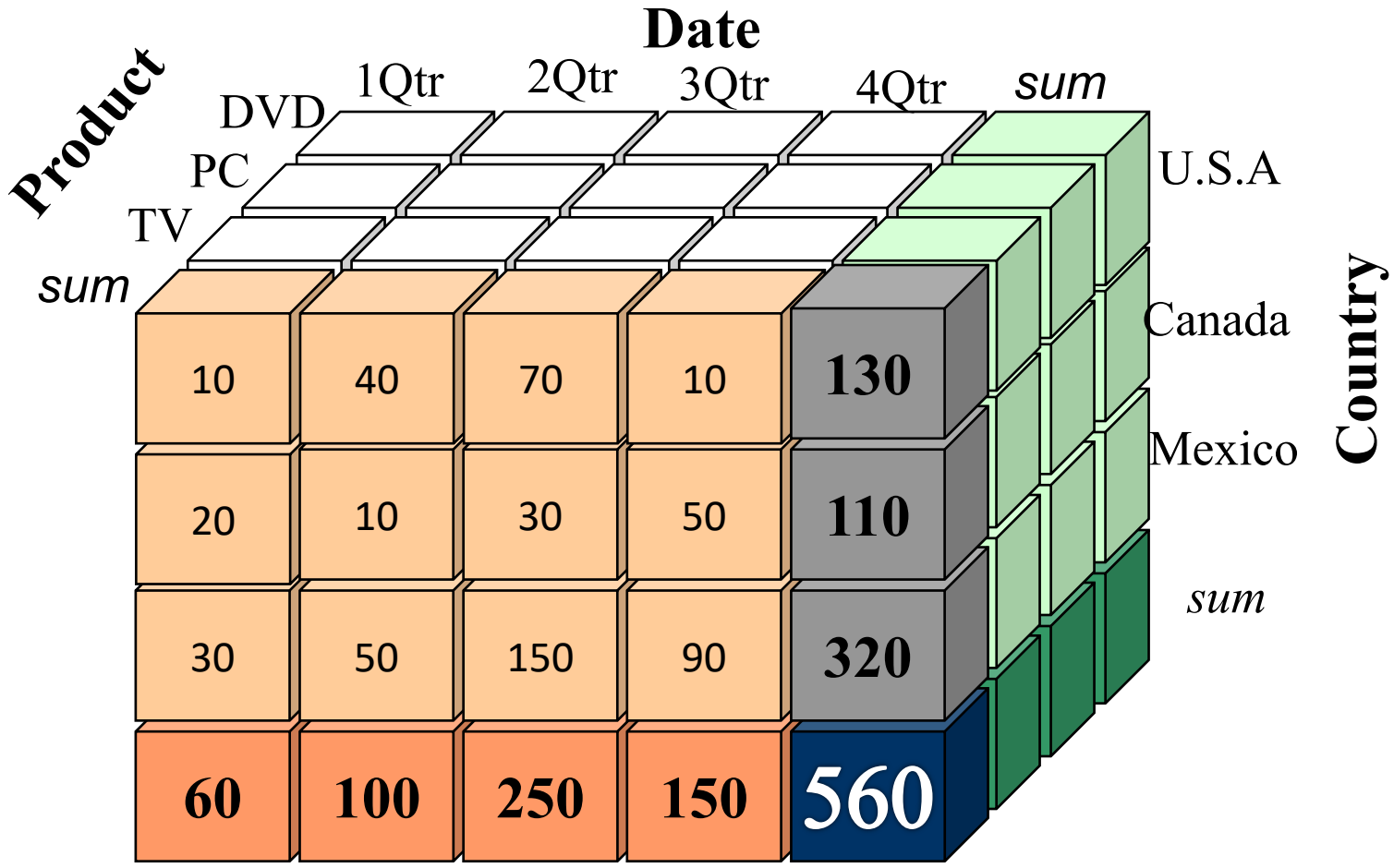
Group By Date



Group By ROLLUP (Date, Country)



Group By CUBE (Period, Date, Country)



Typical OLAP Operations

- **Roll up** (drill-up): summarize data
  - *by climbing up hierarchy or by dimension reduction*
- **Drill down** (roll down): reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*

GROUP-BY
ROLL-UP
CUBE

- **Slice** and **dice**:
  - *project and select*
- **Pivot (rotate)**:
  - *reorient the cube, visualization, 3D to series of 2D planes.*
- Other operations
  - ***drill across**: involving (across) more than one fact table*
  - ***drill through**: through the bottom level of the cube to its back-end relational tables*

Multidimensional DB History

- ’90 large datawarehouses appeared
- ’98 MS OLAP Server, first multidim. database for mass market
- Today: multidimensional analytics part of any commercial DB
- 2000 : Column-stores as the novel data warehouse systems
- 2010+ : Big Data : exploratory analytics