

Data Warehousing and OLAP

Toon Calders
t.calders@tue.nl

TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Motivation

- « Traditional » relational databases are geared towards online transaction processing:
 - bank terminal
 - flight reservations
 - student administration
- Decision support systems have different requirements

Transaction Processing

Transaction processing

- Operational setting
- Up-to-date = critical
- Simple data
- Simple queries; only « touch » a small part of the database

Flight reservations

- ticket sales
- do not sell a seat twice
- reservation, date, name
- Give flight details of X, List flights to Y

Transaction Processing

- **Database must support**
 - **simple data**
 - **tables**
 - **simple queries**
 - **select from where ...**
 - **consistency & integrity CRITICAL**
 - **concurrency**
- **Relational databases, Object-Oriented, Object-Relational**

Decision Support

Decision support

- Off-line setting
- « Historical » data
- Summarized data
- Integrate different databases
- Statistical queries

Flight company

- Evaluate ROI flights
- Flights of last year
- # passengers per carrier for destination X
- Passengers, fuel costs, maintenance info
- Average % of seats sold/month/destination



PART I Concepts

Outline

Online Analytical Processing

- **Data Warehouses**
- Conceptual model: Data Cubes
- Query languages for supporting OLAP
 - SQL extensions
 - MDX
- Database Explosion Problem

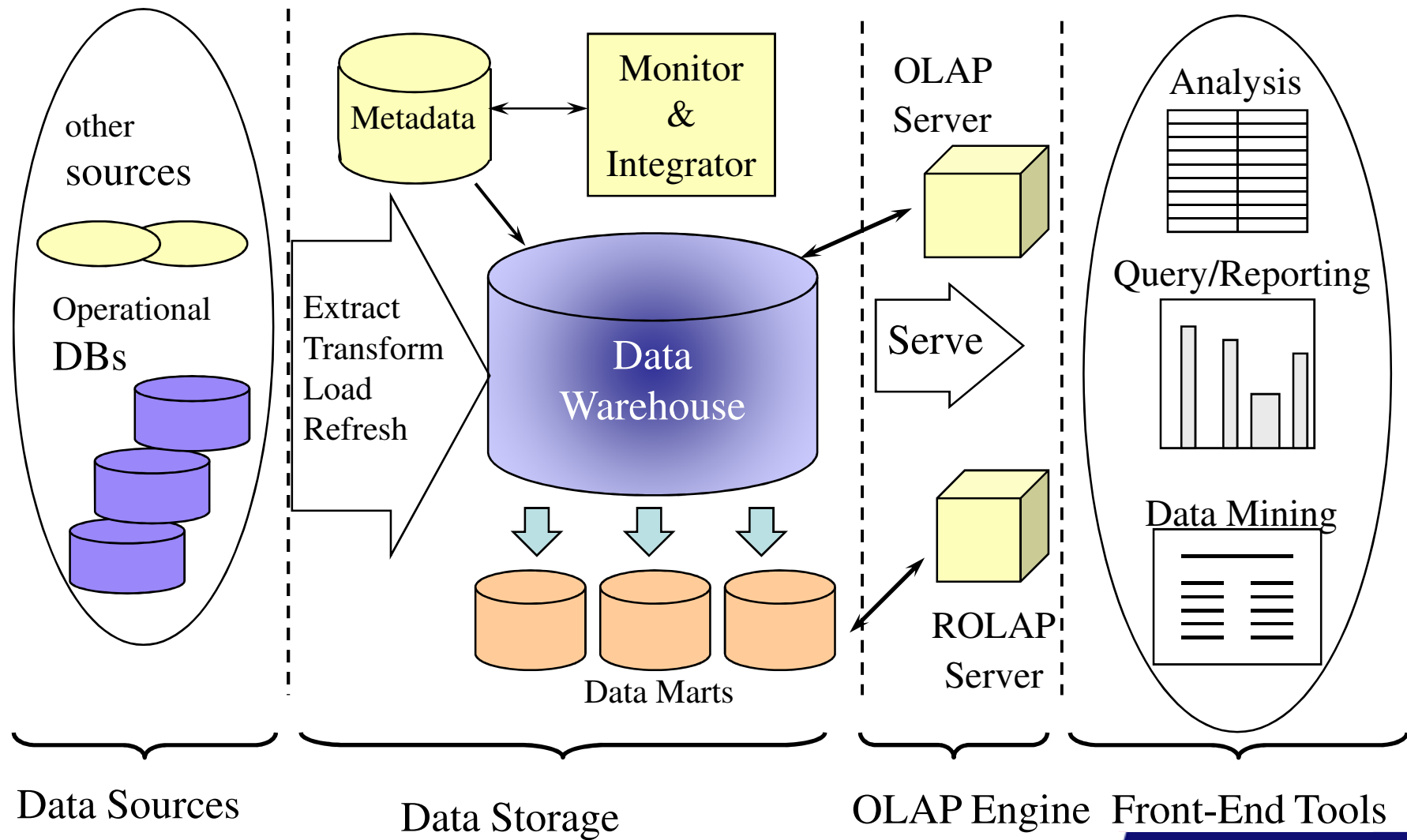
Data Warehouse

A decision support DB maintained **separately** from the operational databases.

Why Separate Data Warehouse?

- Different functions
 - DBMS— tuned for OLTP
 - Warehouse—tuned for OLAP
- Different data
 - Decision support requires historical data
- Integration of data from heterogeneous sources

Three-Tier Architecture



Three-Tier Architecture

- **Extract-Transform-Load**
 - Get data from different sources; data integration
 - Cleaning the data
 - Takes significant part of the effort (up to 80%!)
- **Refresh**
 - Keep the data warehouse up to date when source data changes

Three-Tier Architecture

- **Data storage**
 - Optimized for OLAP
 - Specialized data structures
 - Specialized indexing structures
- **Data marts**
 - common term to refer to “less ambitious data warehouses”
 - Task oriented, departmental level

OLAP

- OLAP = OnLine Analytical Processing
 - Online = no waiting for answers
- OLAP system = system that supports *analytical queries* that are *dimensional* in nature.

Outline

Online Analytical Processing

- Data Warehouses
- **Conceptual model: Data cubes**
- Query languages for supporting OLAP
 - SQL extensions
 - MDX
- Database Explosion Problem

Examples of Queries

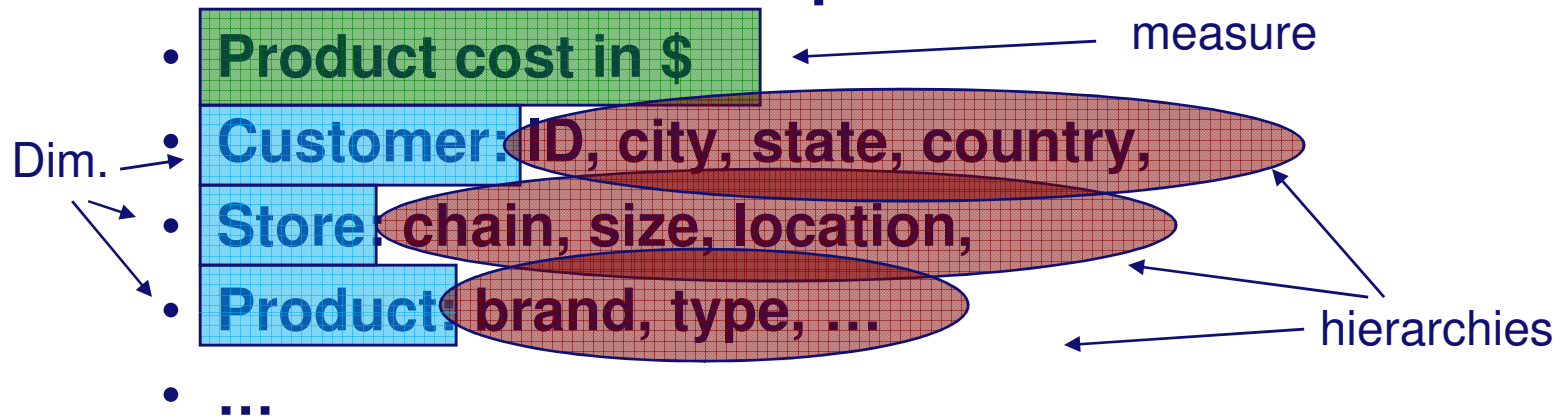
- **Flight company: evaluate ticket sales**
 - give total, average, minimal, maximal amount
 - per date: week, month, year
 - by destination/source port/country/continent
 - by ticket type
 - by # of connections
 - ...

Common Characteristics

- **One (or few) special attribute(s): amount**
→ measure
- **Other attributes: select relevant *regions***
→ dimensions
- **Different levels of generality (month, year)**
→ hierarchies
- **Measurement data is summarized: sum, min, max, average**
→ aggregations

Supermarket Example

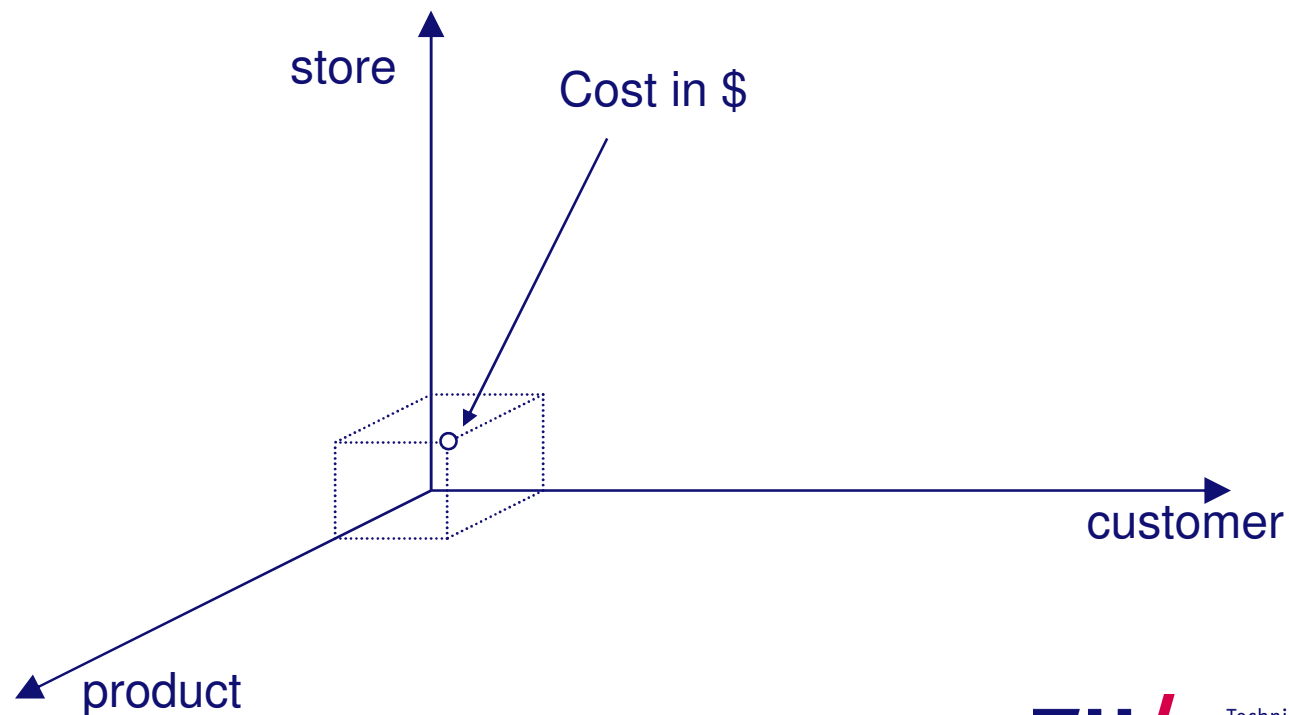
- Evaluate the sales of products



- What are the measure and dimensional attributes, where are the hierarchies?

Why Dimensions?

- Multidimensional view on the data



Cross Tabulation

- **Cross-tabulations are highly useful**
 - **Sales of clothes June→August '06**

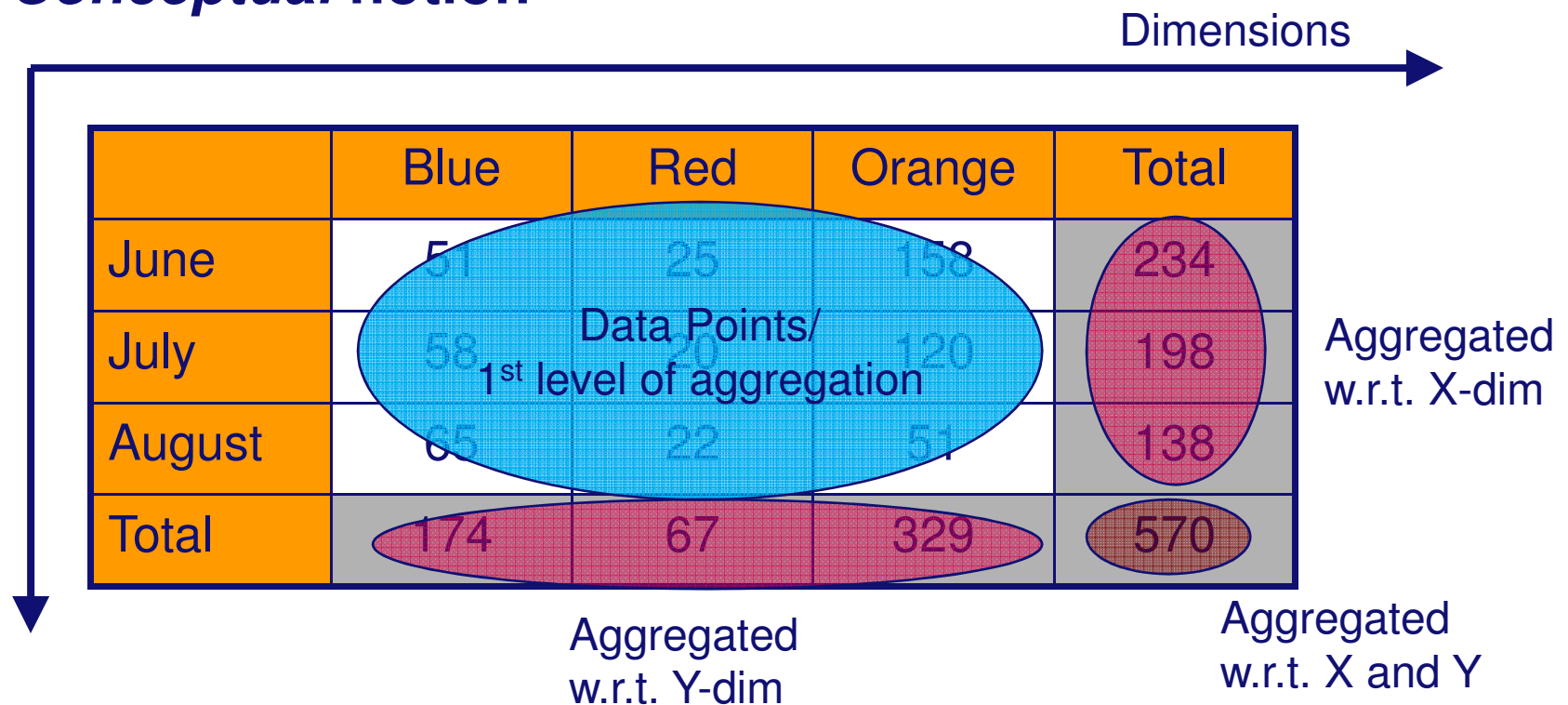
Product: color

Date:month,
June→August
2006

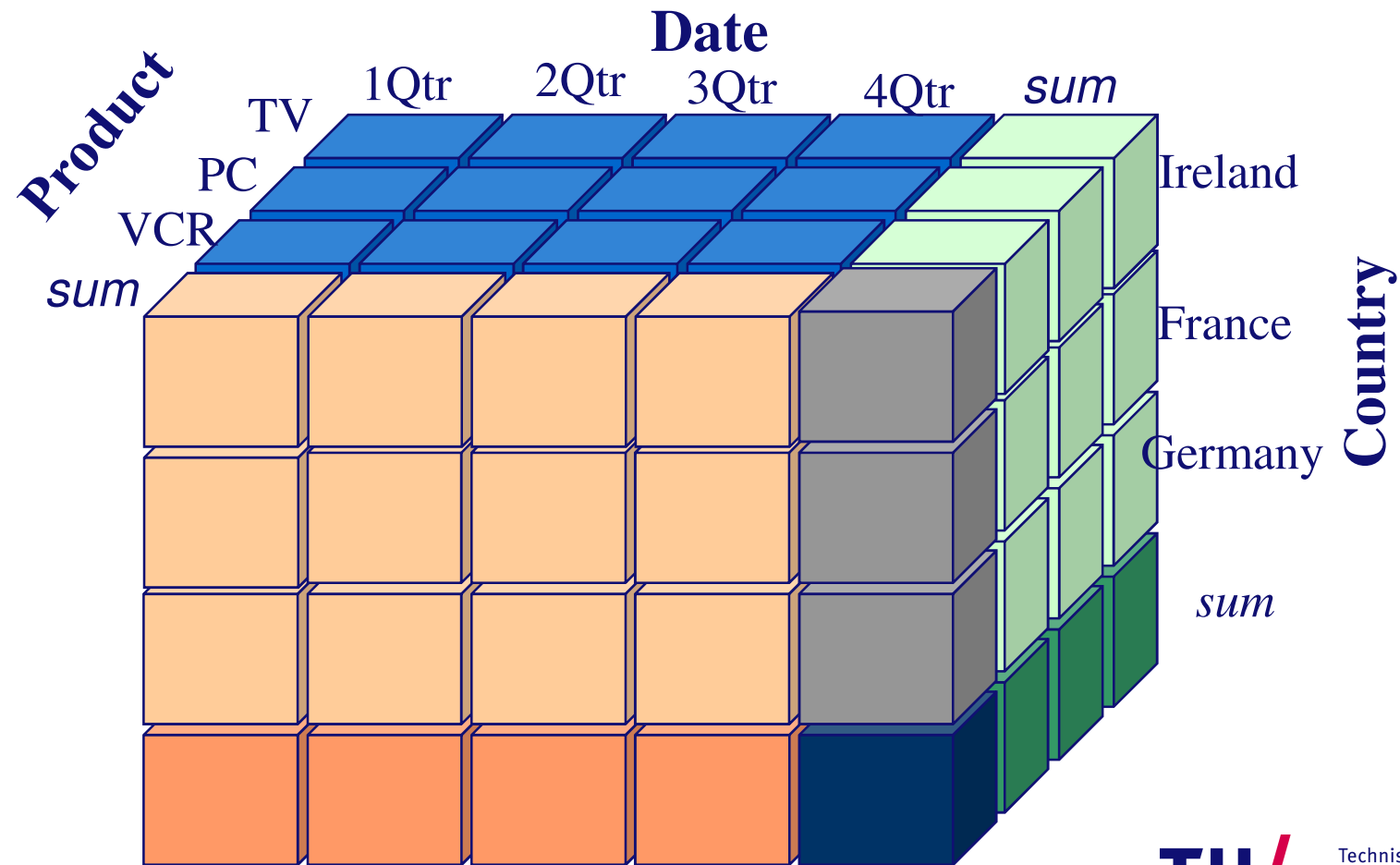
	Blue	Red	Orange	Total
June	51	25	158	234
July	58	20	120	198
August	65	22	51	138
Total	174	67	329	570

Data Cubes

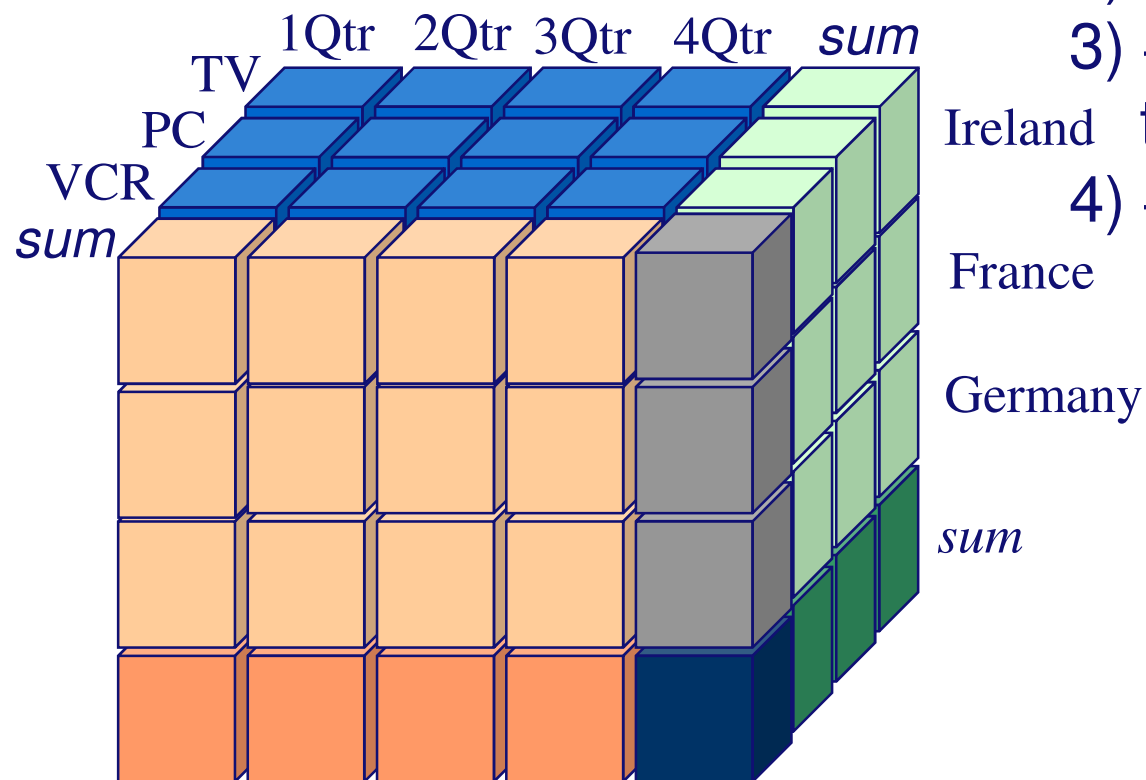
- Extension of Cross-Tables to multiple dimensions
- *Conceptual* notion



Data Cubes



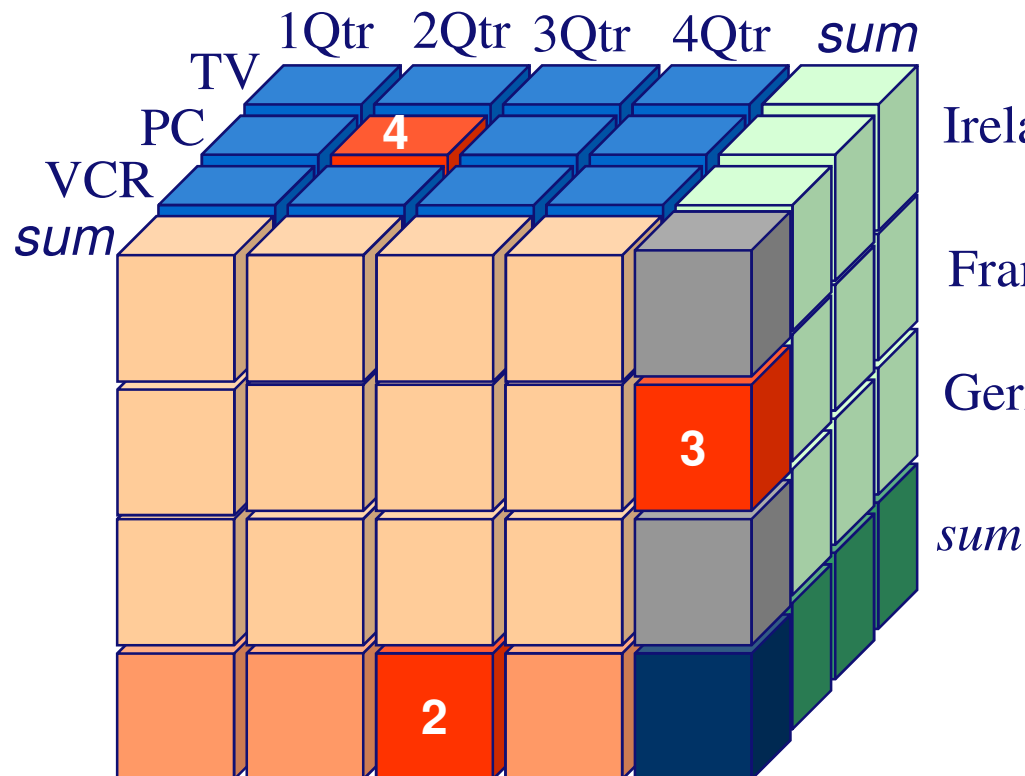
Data Cubes



- 1) #TV's sold in the 2Qtr?
- 2) Total sales in 3Qtr?
- 3) #products sold in France this year
- 4) #PC's in Ireland in 2Qtr?

Data Cubes

In the back, at the bottom, second column



- 1) #TV's sold in the 2Qtr?
- 2) Total sales in 3Qtr?
- 3) #products sold in France this year
- 4) #PC's in Ireland in 2Qtr?

Outline

Online Analytical Processing

- Data Warehouses
- Conceptual model: Data cubes
- Query languages for supporting OLAP
 - SQL extensions
 - MDX
- Database Explosion Problem

Operations with Data Cubes

- What operations can you think of that an analyst might find useful? (e.g., store)

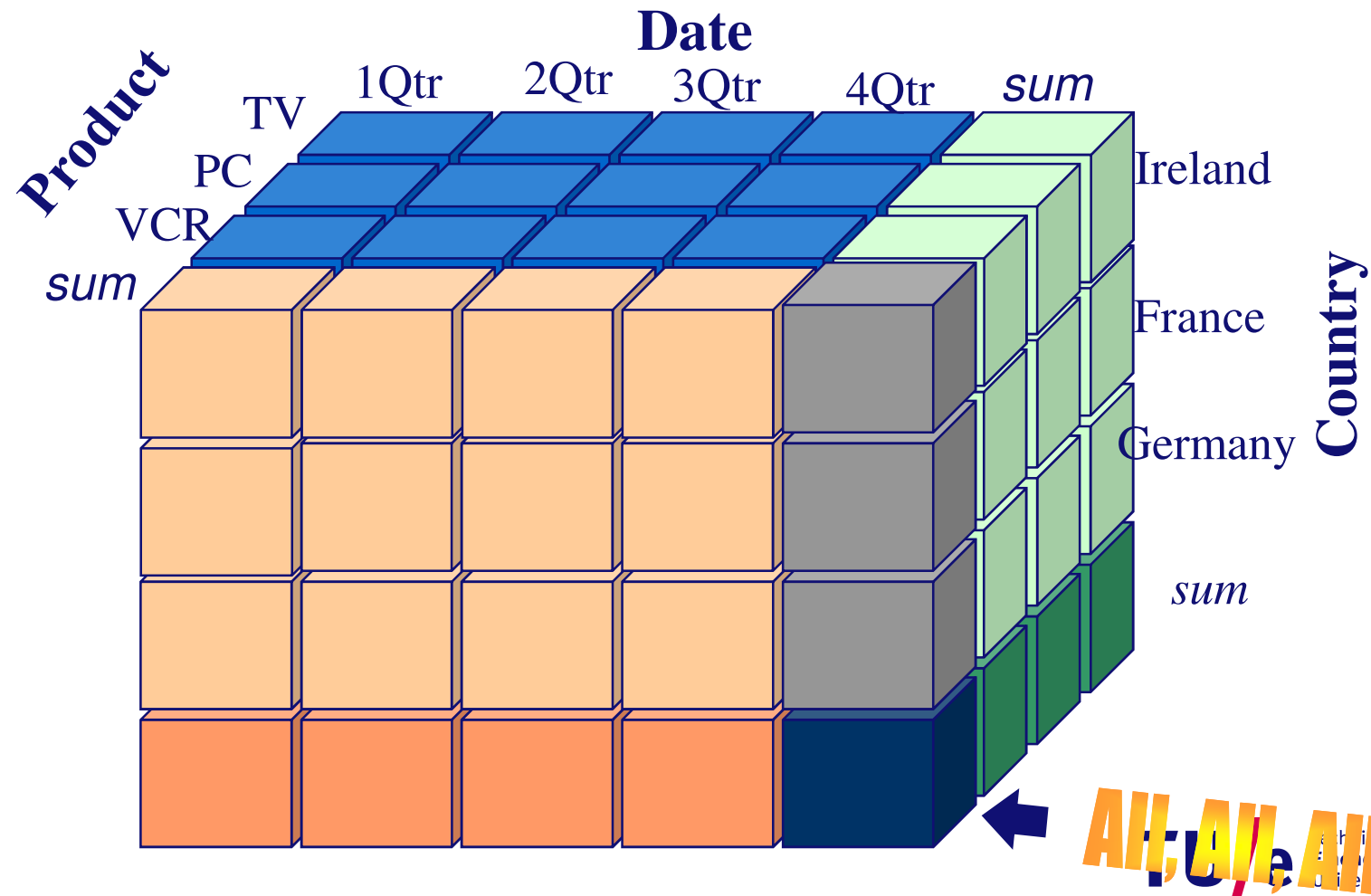
Operations with Data Cubes

- What operations can you think of that an analyst might find useful? (e.g., store)
 - only look at stores in the Netherlands
 - look at cities instead of individual stores
 - look at the cross-table for product-date
 - restrict analysis to 2006, product O1
 - go back to a finer granularity at the store level

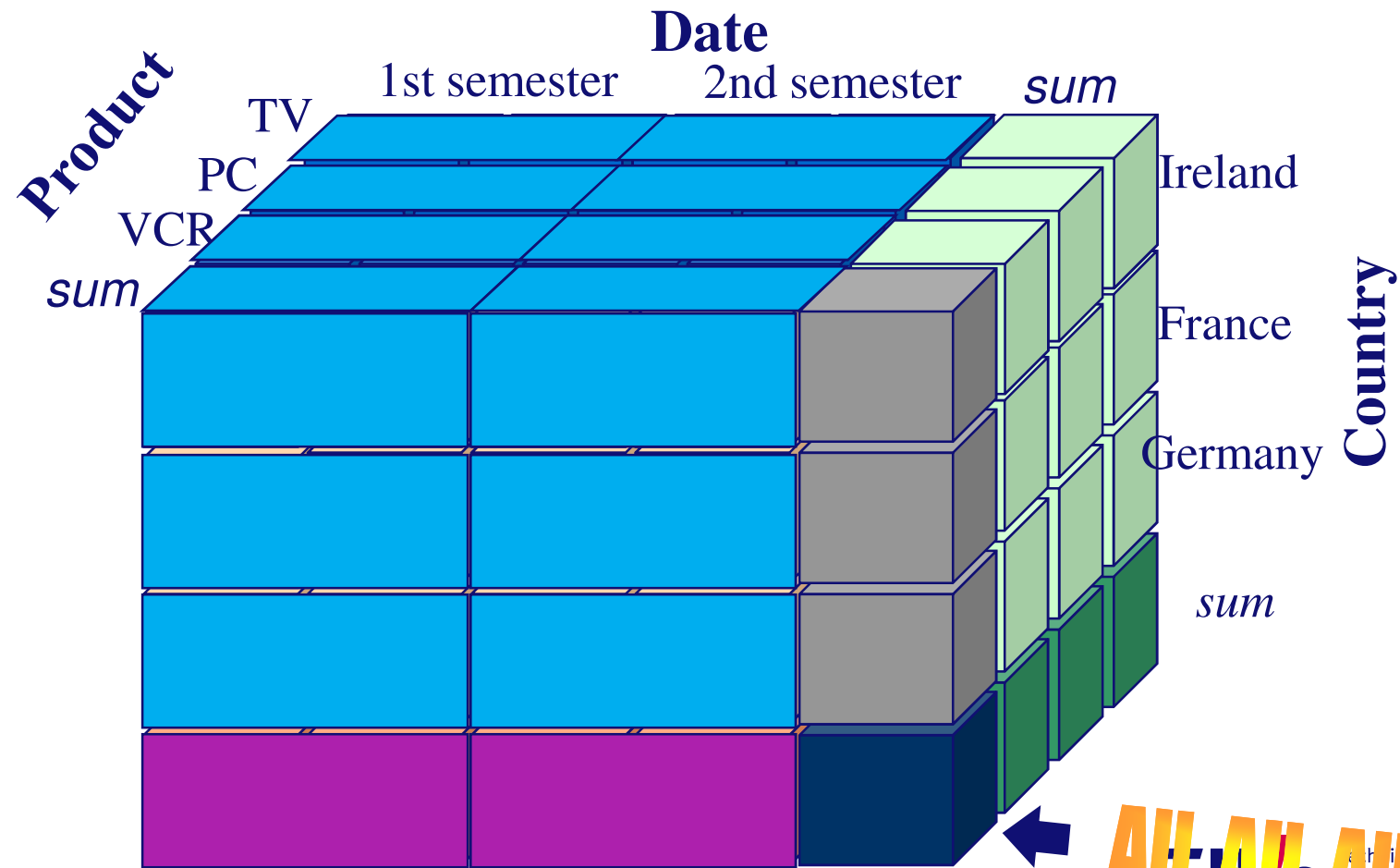
Roll-Up

- **Move in one dimension from a lower granularity to a higher one**
 - store → city
 - cities → country
 - product → product type
 - Quarter → Semester

Roll-Up



Roll-Up



All, All, All

Drill-down

- **Inverse operation**
- **Move in one dimension from a higher granularity to a lower one**
 - city → store
 - country → cities
 - product type → product
- **Drill-through:**
 - go back to the original, individual data records

Pivoting

- **Change the dimensions that are “displayed”; select a cross-tab.**
 - look at the cross-table for product-date
 - display cross-table for date-customer

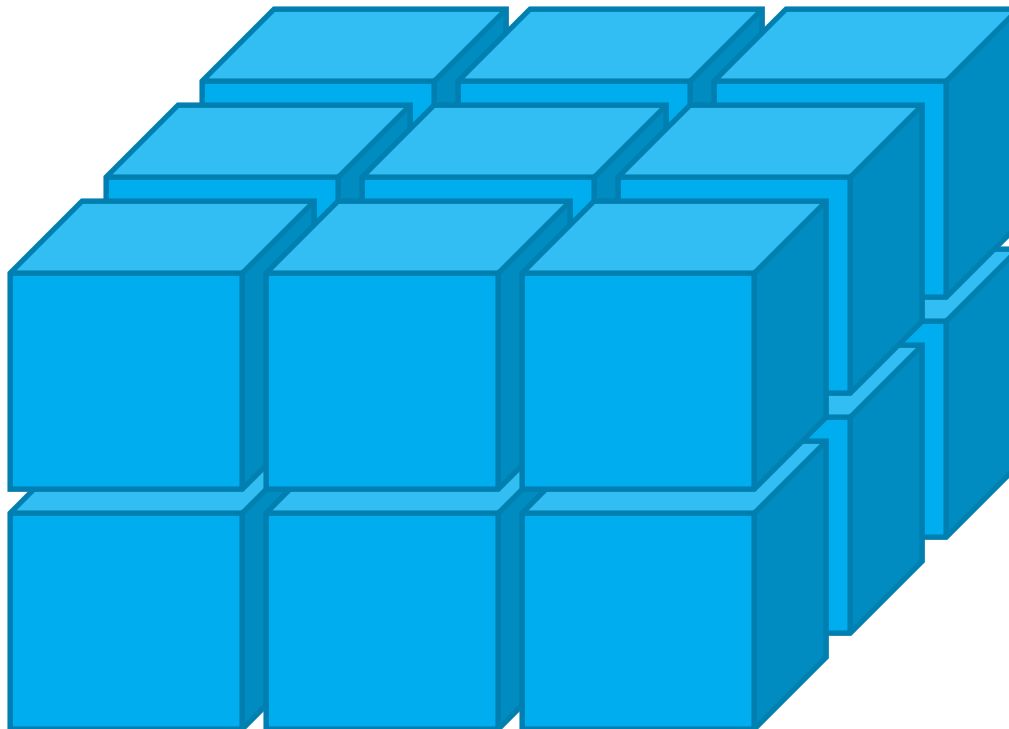
Pivoting

- Change the dimensions that are “displayed”; select a cross-tab.
 - look at the cross-table for product-date
 - display cross-table for date-customer

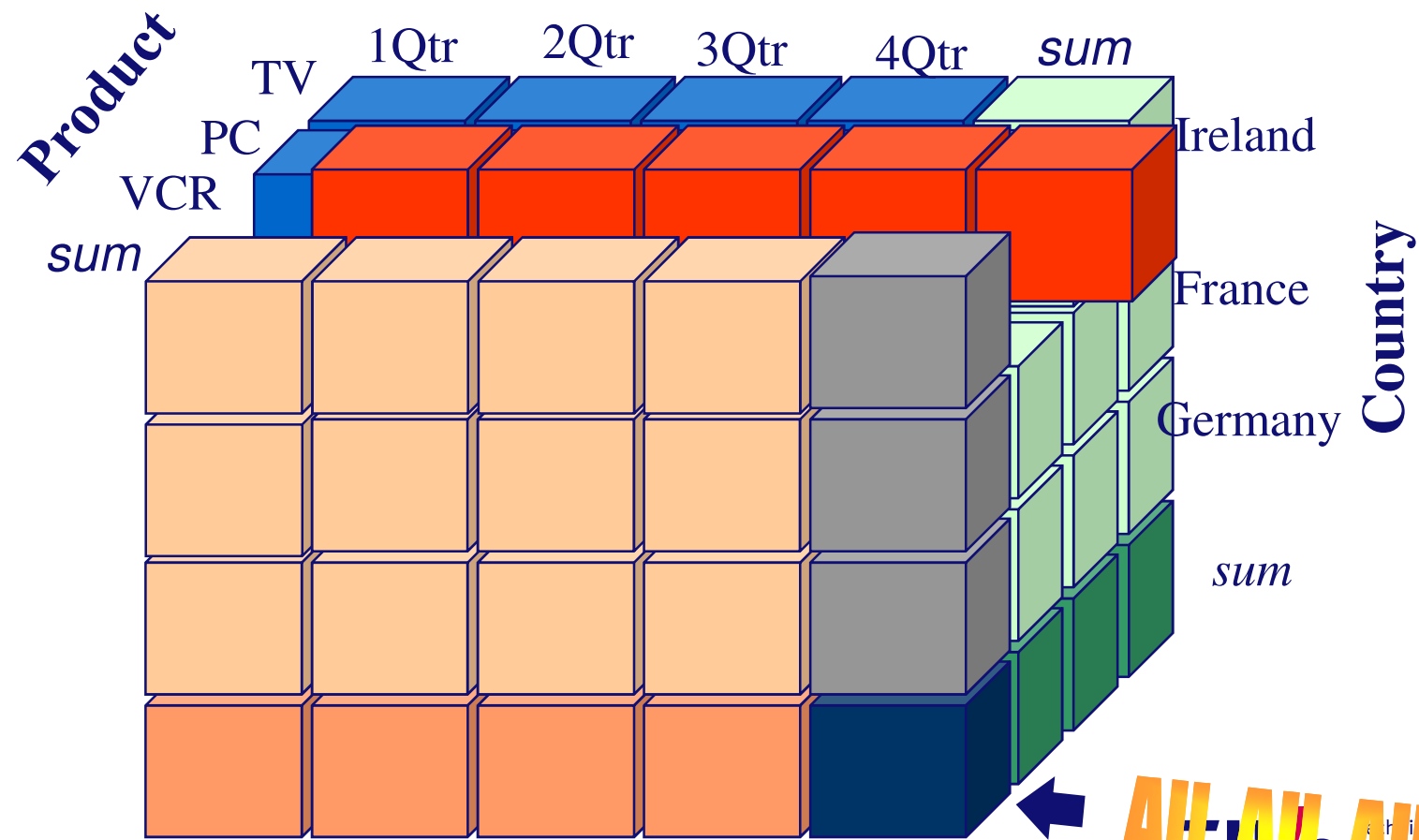
Sales		Date		
Country		1st sem	2 nd sem	Total
	Ireland	20	23	43
	France	126	138	264
	Germany	56	48	104
	Total	202	209	411

Slice & dice

- Roll-up on multiple dimensions at once



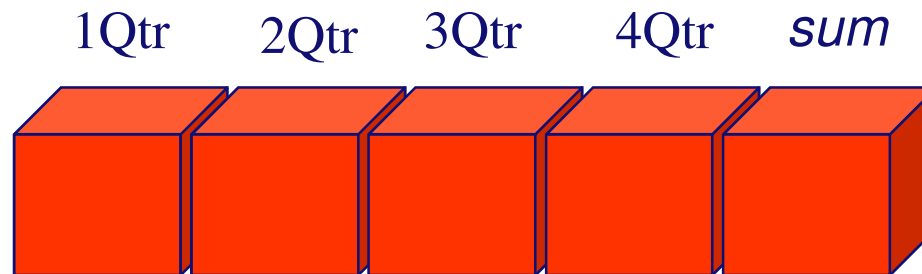
Select



All, All, All

Select

- **Select a part of the cube by restricting one or more dimensions**
 - **restrict analysis to Ireland and VCR**



Outline

Online Analytical Processing

- Data Warehouses
- Conceptual model: Data cubes
- Query languages for supporting OLAP
 - **SQL extensions**
 - MDX
- Database Explosion Problem

Extended Aggregation

- **SQL-92 aggregation quite limited**
 - **Many useful aggregates are either very hard or impossible to specify**
 - **Data cube**
 - **Complex aggregates (median, variance)**
 - **binary aggregates (correlation, regression curves)**
 - **ranking queries (“assign each student a rank based on the total marks”)**
- **SQL:1999 OLAP extensions**

Representing the Cube

- Special value « null » is used:

Sales	Date			
Country		1st sem	2 nd sem	Total
	Ireland	20	23	43
	France	126	138	264
	Germany	56	48	104
	Total	202	209	411

Representing the Cube

- Special value « null » is used:

Date	Country	Sales
1st semester	Ireland	20
1st semester	France	126
1st semester	Germany	56
1st semester	null	202
2nd semester	Ireland	23
2nd semester	France	138
2nd semester	Germany	48
2nd semester	null	209
null	Ireland	43
null	France	264
null	Germany	104
null	null	411

Group by Cube

- group by cube:

```
select item-name, color, size, sum(number)  
from sales  
group by cube(item-name, color, size)
```

Computes the union of eight different groupings of the *sales* relation:

```
{ (item-name, color, size), (item-name, color),  
(item-name, size), (color, size), (item-name),  
(color), (size), ( ) }
```

Group by Cube

- Relational representation of the date-country-sales cube can be computed as follows:

```
select semester as date, country, sum(sales)
from sales
group by cube(semester, country)
```

- grouping() and decode() can be applied to replace “null” by other constant:
 - decode(grouping(semester), 1, ‘all’, *semester*)

Group by Rollup

- rollup construct generates union on every prefix of specified list of attributes

-

```
select item-name, color, size, sum(number)  
from sales  
group by rollup(item-name, color, size)
```

Generates union of four groupings:

```
{ (item-name, color, size), (item-name, color),  
  (item-name), ( ) }
```

Group by Rollup

- Rollup can be used to generate aggregates at multiple levels.
- E.g., suppose *itemcategory(item-name, category)* gives category of each item.

```
select category, item-name, sum(number)
from sales, itemcategory
where sales.item-name = itemcategory.item-name
group by rollup(category, item-name)
```

gives a hierarchical summary by *item-name* and by *category*.

Group by Cube & Rollup

- **Multiple rollups and cubes can be used in a single group by clause**
 - **Each generates set of group by lists, cross product of sets gives overall set of group by lists**

Example

```
select item-name, color, size, sum(number)  
from sales  
group by rollup(item-name), rollup(color, size)
```

generates the groupings

$$\{ \textit{item-name}, () \} \times \{ (\textit{color}, \textit{size}), (\textit{color}), () \}$$

=

$$\{ (\textit{item-name}, \textit{color}, \textit{size}), (\textit{item-name}, \textit{color}),$$
$$(\textit{item-name}), (\textit{color}, \textit{size}), (\textit{color}), () \}$$

MDX

- **Multidimensional Expressions (MDX)** is a query language for cubes
 - Supported by many data warehouses
 - Input and output are cubes

```
SELECT { [Measures].[Store Sales] } ON  
COLUMNS, { [Date].[2002], [Date].[2003] } ON  
ROWS FROM Sales  
WHERE ( [Store].[USA].[CA] )
```

Outline

Online Analytical Processing

- Data Warehouses
- Conceptual model: Data cubes
- Query languages for supporting OLAP
 - SQL extensions
 - MDX
- Database Explosion Problem

Implementation

- To make query answering more efficient: consolidate (materialize) all aggregations
- Early implementations used a multidimensional array.
 - Fast lookup: cell(prod. p, date d, prom. pr):
 - look up index of p1, index of d, index of pr:
$$\text{index} = (p \times D \times PR) + (d \times PR) + pr$$

Implementation

- **Multidimensional array**
 - **obvious problem: sparse data can easily be solved, though.**

Example:

**binary search tree, key on index
hash table.**

Implementation

- However: very quickly people were confronted with the *Data Explosion Problem*

Consolidating the summaries blows the data enormously !

Reasons are often misunderstood and confusing.

Data Explosion Problem

- Why?

Suppose:

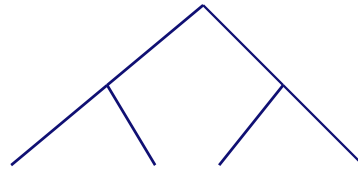
- n dimensions, every dimension has d values
- d^n possible tuples.
- Number of cells in the cube: $(d+1)^n$
- So, this is not the problem

Data Explosion Problem

- Why?

Suppose

- n dimensions, every dimension has d values
- every dimension has a hierarchy



- most extreme case: **binary tree**
→ **2^d possibilities/dimension**

Data Explosion Problem

- Why?

Suppose

- n dimensions, every dimension has d values
- every dimension has a hierarchy
- most extreme case: binary tree
 - 2d possibilities/dimension → $2^n \times d^n$ cells

Only partial explanation (factor 2^n comes from an extremely pathological case)

Data Explosion Problem

- **Why?**
 - The problem is that most data is not dense, but *sparse*.
 - Hence, not all d^n combinations are possible.

Example: 10 dimensions with 10 values

- 10 000 000 000 possibilities

Suppose « only » 1 000 000 are present

Data Explosion Problem

Example: 10 dimensions with 10 values

- 10 000 000 000 possibilities

Suppose « only » 1 000 000 are present

Every tuple increases count of 2^{10} cells !

With hierarchies: effect even worse!

If every hierarchy has 5 items:

$$5^{10} = 9\,765\,625 \text{ cells!}$$

Summary Part I

- Datawarehouses supporting OLAP for *decision support*
- Data Cubes as a *conceptual* model
 - Measurement, dimensions, hierarchy, aggregation
- Queries
 - Roll-up, Drill-down, Slice and dice, pivoting...
 - SQL:1999 extensions for supporting OLAP
- Straightforward implementation is problematic



PART II Technology

II.1 View materialization

II.2 Data Storage and Indexing

View materialization

Is it possible to get performance gains when only partially materializing the cube?

Which parts should we materialize in order to get the best performance?

V. Harinarayan, A. Rajaraman, and J. D. Ullman: Implementing data cubes efficiently. In: ACM SIGMOD 1996.

Overview

- **Problem statement**
- **Formal model of computation**
 - Partial order on Queries
 - Cost model
- **Greedy solution**
- **Performance guarantee**
- **Conclusion**

The problem: informally

- **Relation Sales**
 - **Dimensions: part, supplier, customer**
 - **Measure: sales**
 - **Aggregation: sum**
- **The data cube contains all aggregations on part, supplier, customer; e.g.:**
 - **(p1,s1,<all>,165)**
 - **(p1,s2,<all>,145)**

Queries

- We are interested in answering the following *type* of queries efficiently:

(part, supplier)

```
SELECT part, supplier, sum(sales)
FROM Sales
GROUP BY part, supplier
```

(part)

```
SELECT part, sum(sales)
FROM Sales
GROUP BY part
```

(supplier)

```
SELECT supplier, sum(sales)
FROM Sales
GROUP BY supplier
```

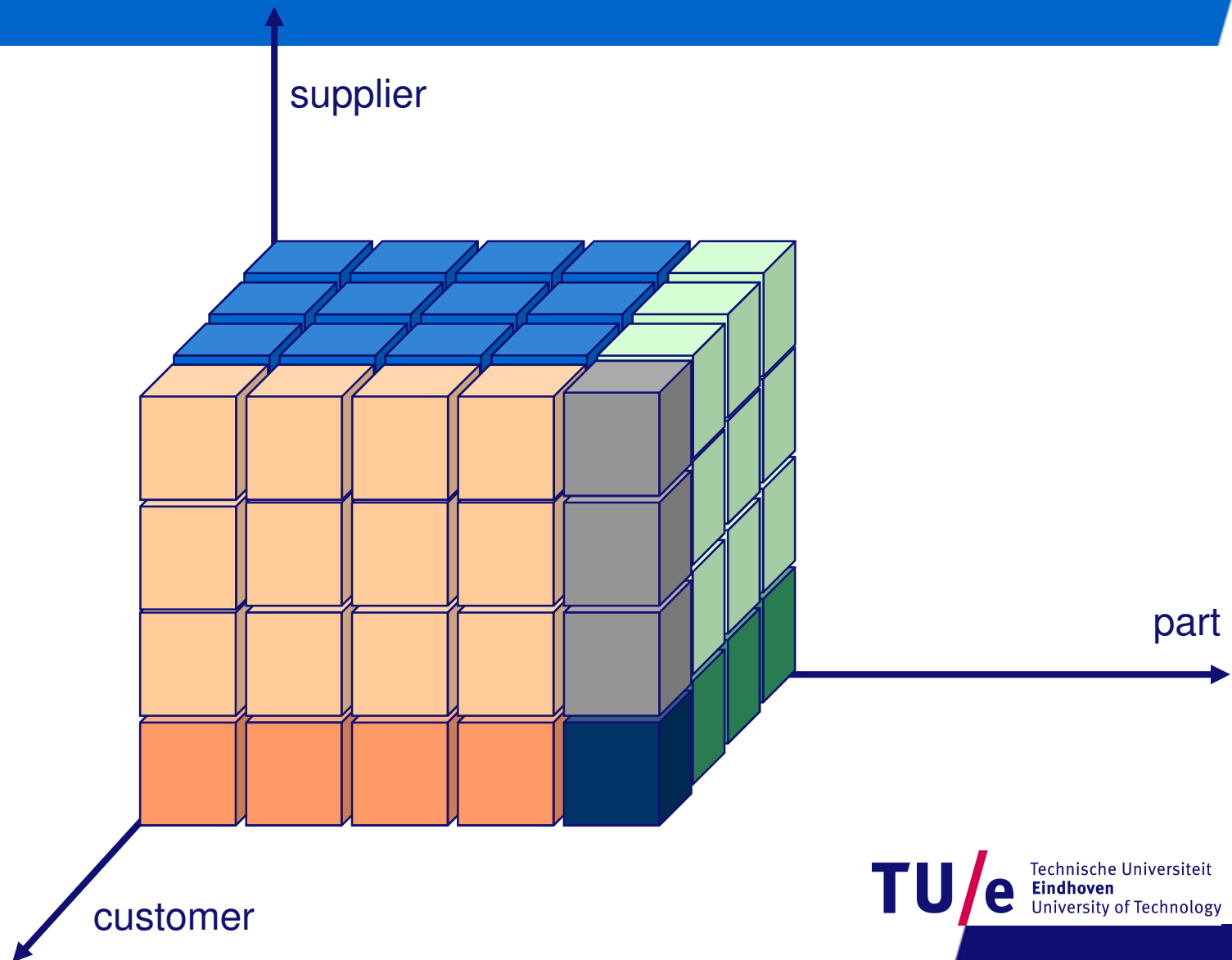
(part, customer)

```
SELECT customer, part, sum(sales)
FROM Sales
GROUP BY customer, part
```

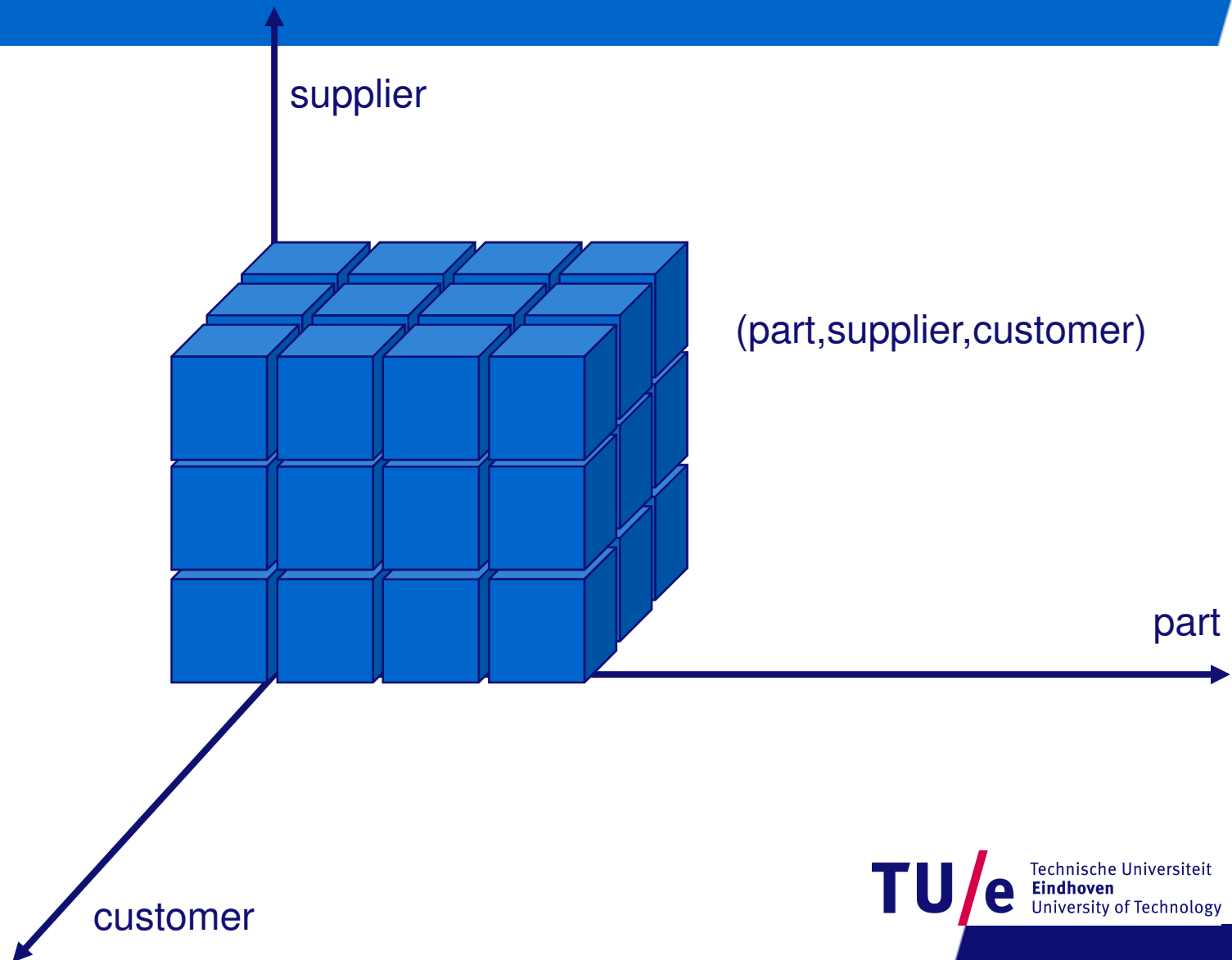
Queries

- The queries are quantified by their *grouping attributes*
- These queries select *disjoint* parts of the cube.
- The *union* of all these queries is the complete cube.

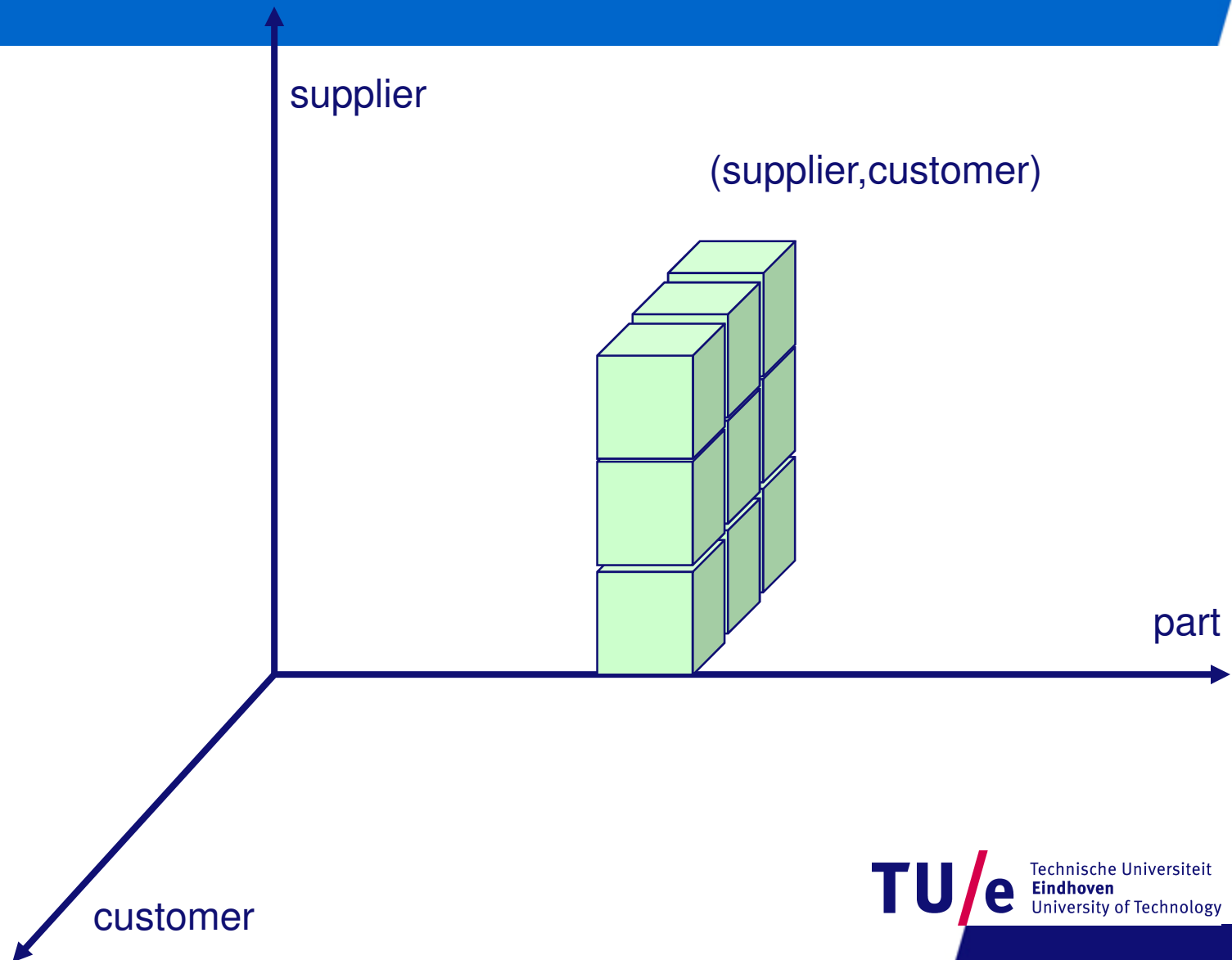
Queries



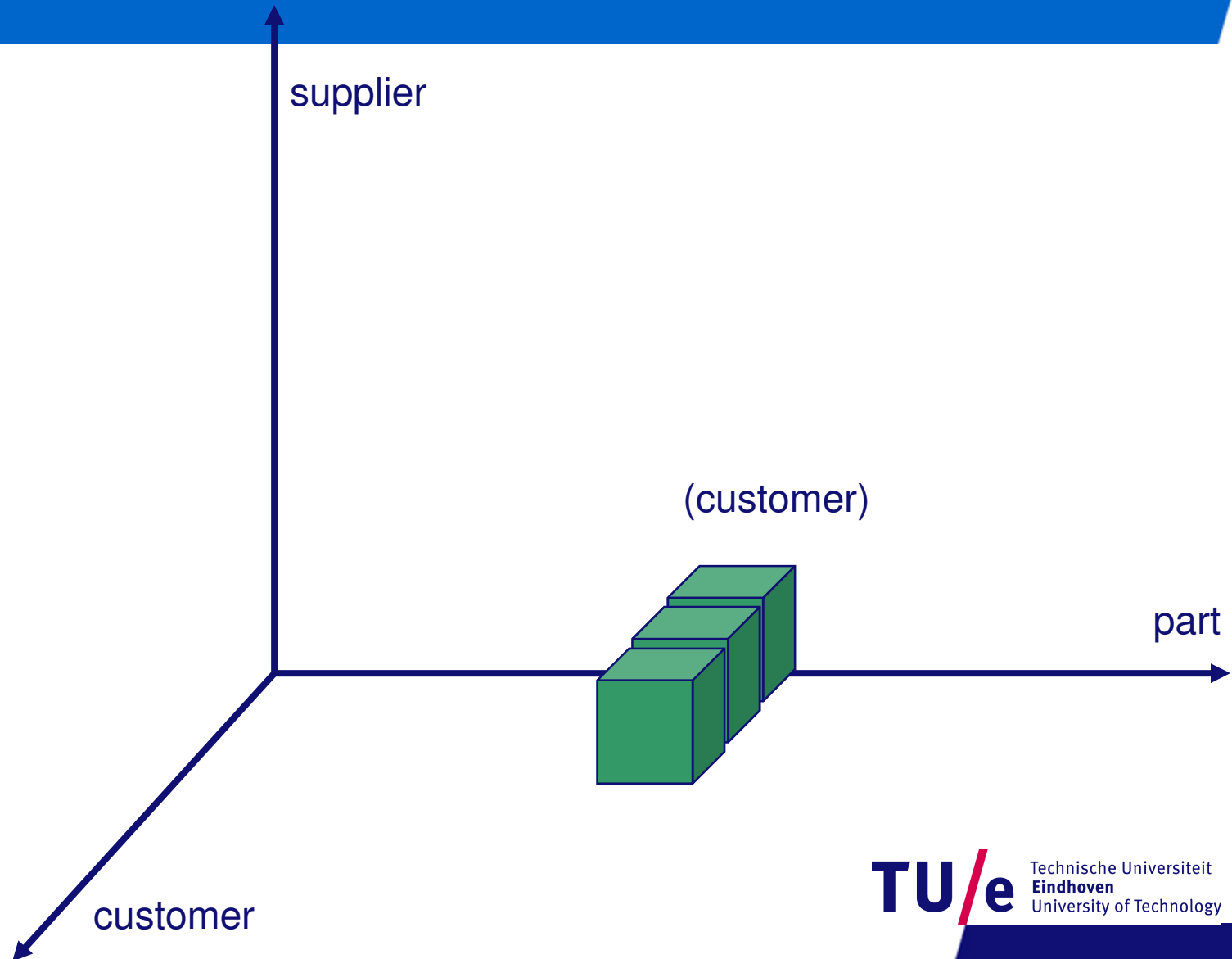
Queries



Queries



Queries



Materialization

- Materializing everything is impossible
- Cost of evaluating following query directly:

```
SELECT D1, ..., Dk, sum(M)
FROM R
GROUP BY D1, ..., Dk
```

- in practice roughly linear in the size of R
(worst case: $|R| \log(|R|)$)
- Materializing *some* queries can help the other queries too

Materialization

Example:

(part, customer)

```
SELECT customer, part, sum(sales)
FROM Sales
GROUP BY customer, part
```

| Sales |

(part)

```
SELECT part, sum(sales)
FROM Sales
GROUP BY part
```

| Sales |

Materialization

Example:

```
( part, customer )  
SELECT customer, part, sum(sales)  
FROM Sales  
GROUP BY customer, part  
  
materialized as PC
```

|PC|

```
( part )  
SELECT part, sum(sales)  
FROM PC  
GROUP BY part
```

|PC|

Example

Query	Answer
• (part,supplier,customer)	6M
• (part,customer)	6M
• (part,supplier)	0.8M
• (supplier,customer)	6M
• (part)	0.2M
• (supplier)	0.01M
• (customer)	0.1M
• ()	1

Example: nothing materialized

Query	Answer	Cost
• (part,supplier,customer)	6M	6M
• (part,customer)	6M	6M
• (part,supplier)	0.8M	6M
• (supplier,customer)	6M	6M
• (part)	0.2M	6M
• (supplier)	0.01M	6M
• (customer)	0.1M	6M
• ()	1	6M

Total cost: 48M

Example: some materialized

Query	Answer	Cost
• <u>(part,supplier,customer)</u>	6M	
• (part,customer)	6M	
• <u>(part,supplier)</u>	0.8M	
• (supplier,customer)	6M	
• (part)	0.2M	
• (supplier)	0.01M	
• <u>(customer)</u>	0.1M	
• ()	1	

Example: some materialized

Query	Answer	Cost
• <u>(part,supplier,customer)</u>	6M	<u>6M</u>
• (part,customer)	6M	6M
• <u>(part,supplier)</u>	0.8M	<u>0.8M</u>
• (supplier,customer)	6M	6M
• (part)	0.2M	<u>0.8M</u>
• (supplier)	0.01M	<u>0.8M</u>
• <u>(customer)</u>	0.1M	<u>0.1M</u>
• ()	1	<u>0.1M</u>

Total cost: 20.6M

Research Questions

- What is the *optimal set of views* to materialize?
- *How many views* must we materialize to get reasonable performance?
- Given bounded space S , what is the *optimal choice* of views to materialize?

Central Question

- **Given:**
 - for every view its size
 - a number k
- **Find:**
 - which k materialized views give the most *gain*

This problem is NP-complete

- greedy algorithm

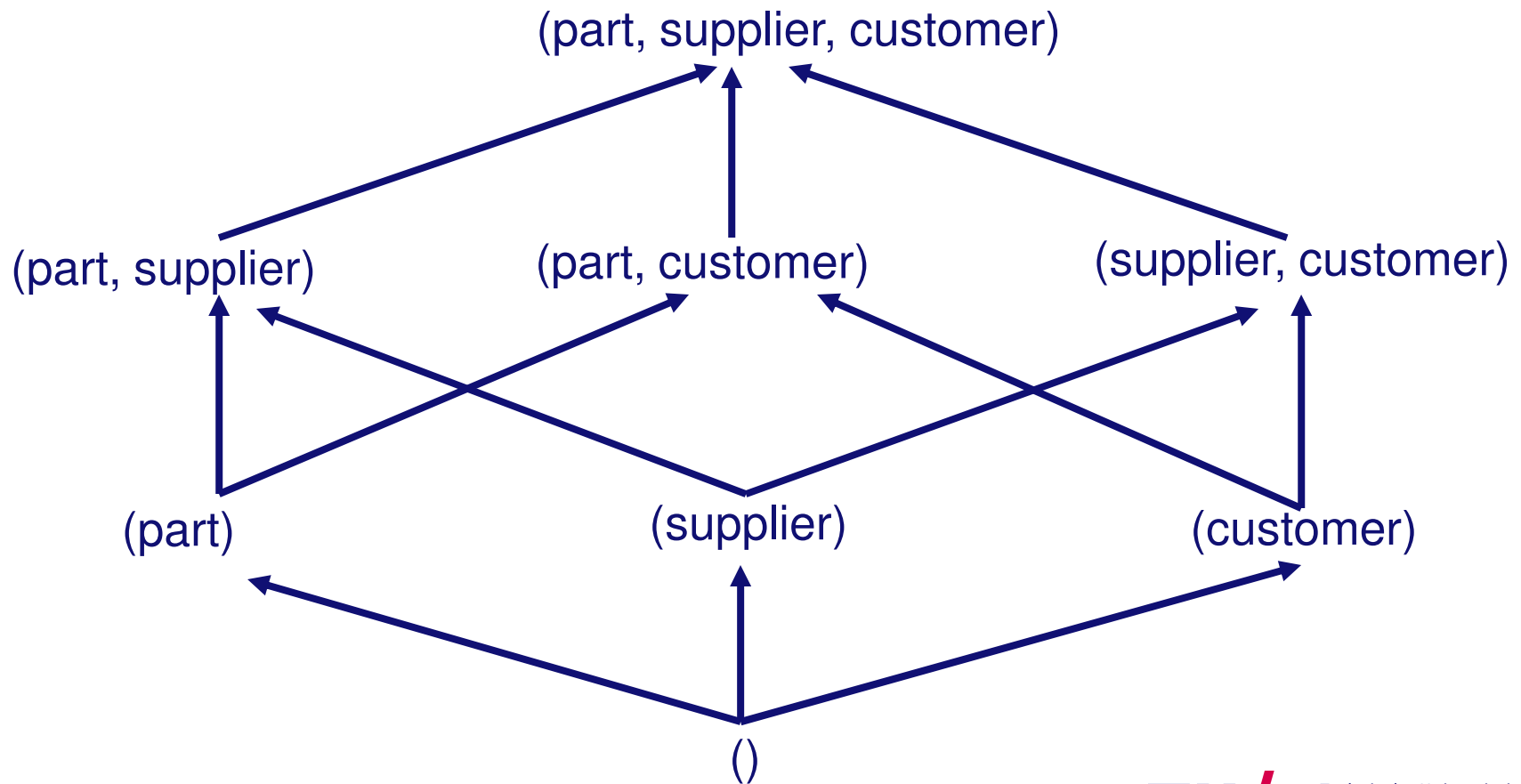
Overview

- Problem statement
- **Formal model of computation**
 - Partial order on Queries
 - Cost model
- Greedy solution
- Performance guarantee
- Conclusion

Partial order on queries

- $Q1 \leq Q2$:
 - $Q1$ can be answered using the query results of $Q2$
 - A *materialized view* of $Q2$ can be used to answer $Q1$
- \leq is a partial order on the views

Partial order on queries



Partial order on queries

- Can be generalized to hierarchies:
 $(\text{product}, \text{country}, -) \leq (\text{product}, \text{city}, -)$
 $(-, \text{country}, -) \leq (\text{product}, \text{city}, -)$
 $(-, -, -) \leq (\text{product}, \text{city}, -)$
 $(-, \text{country}, \text{year}) \leq (-, \text{city}, \text{month})$

$(a_1, \dots, a_n) \leq (b_1, \dots, b_n)$ if for all $i = 1 \dots n$,
 a_i higher than or equal to b_i in the hierarchy of the i th
dimension (a_i more general than b_i)

Cost Model

- Given a set of materialized views $S = \{V_1, \dots, V_n\}$, the *cost of query Q* is

$$\text{cost}_S(Q) := \min(\{ |V_i| : i = 1 \dots n, Q \leq V_i \})$$

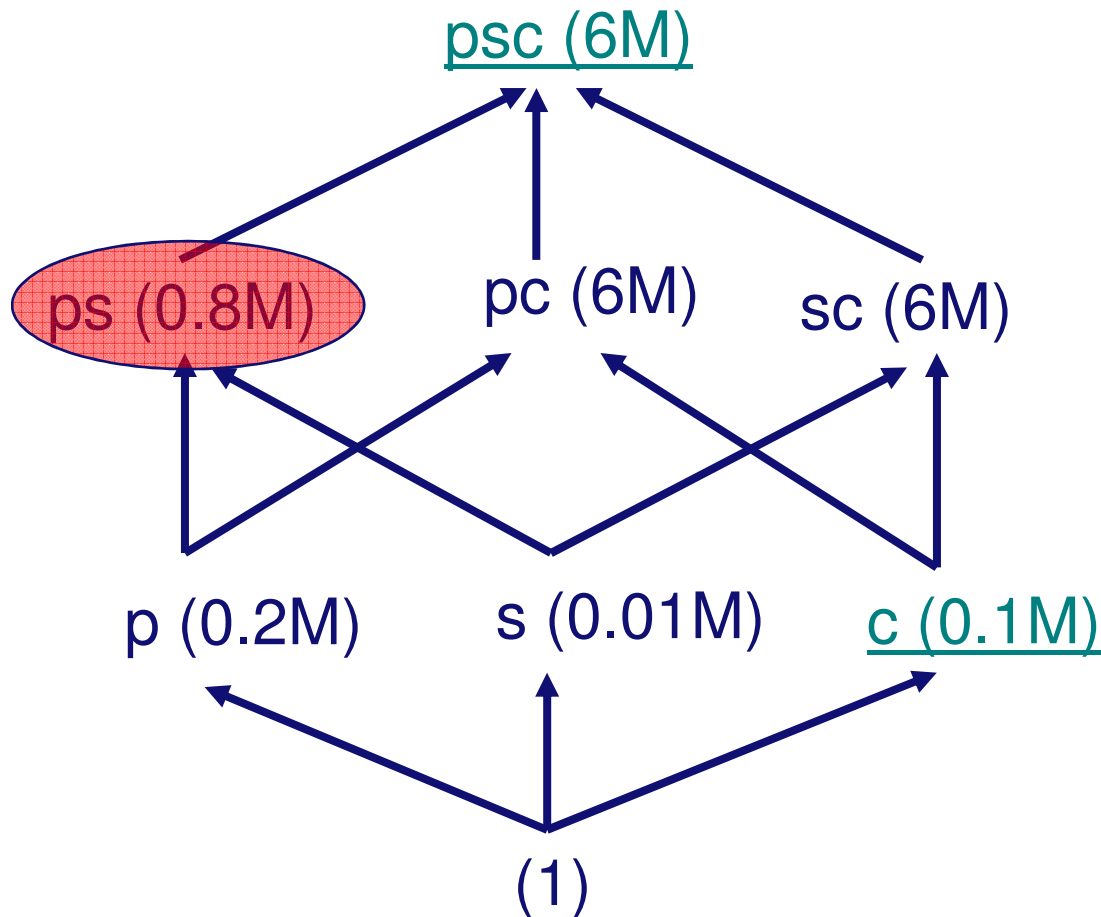
- The *benefit* of view W for computing Q w.r.t. S

$$B_W(Q | S) = \text{cost}_{S \cup \{W\}}(Q) - \text{cost}_S(Q)$$

- Total benefit* of W w.r.t. S

$$B_W(S) = \sum_Q B_W(Q | S)$$

Cost Model: Example



Q	C_s	$C_{SU\{w\}}$	B_w
psc	6M	6M	0
Ps	6M	0.8M	5.2M
Pc	6M	6M	0
p	6M	0.8M	5.2M
s	6M	0.8M	5.2M
c	0.1	0.1	0
()	0.1M	0.1M	0

Overview

- Problem statement
- Formal model of computation
 - Partial order on Queries
 - Cost model
- Greedy solution
- Performance guarantee
- Conclusion

Greedy algorithm

- **Input:**
 - Size for every view and constant k
- **In each step**
 - Select the view with the most benefit
 - Add it to the result
 - Until we have k views

Greedy Algorithm

Algorithm

$S := \{ \text{top view} \};$

for $i:=1$ to k {

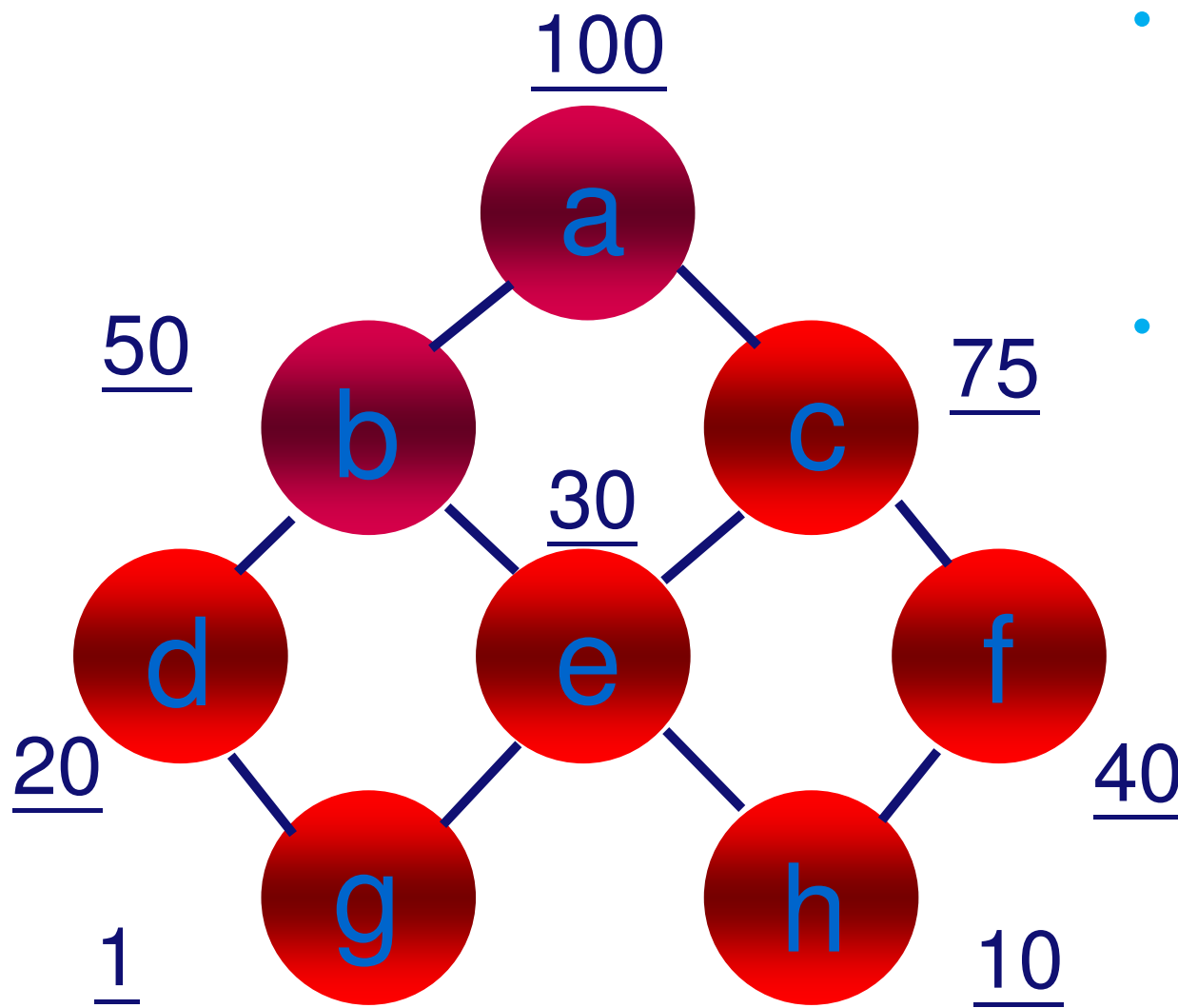
select view W such that $B_W(S)$ is maximal

$S := S \cup \{W\}$

}

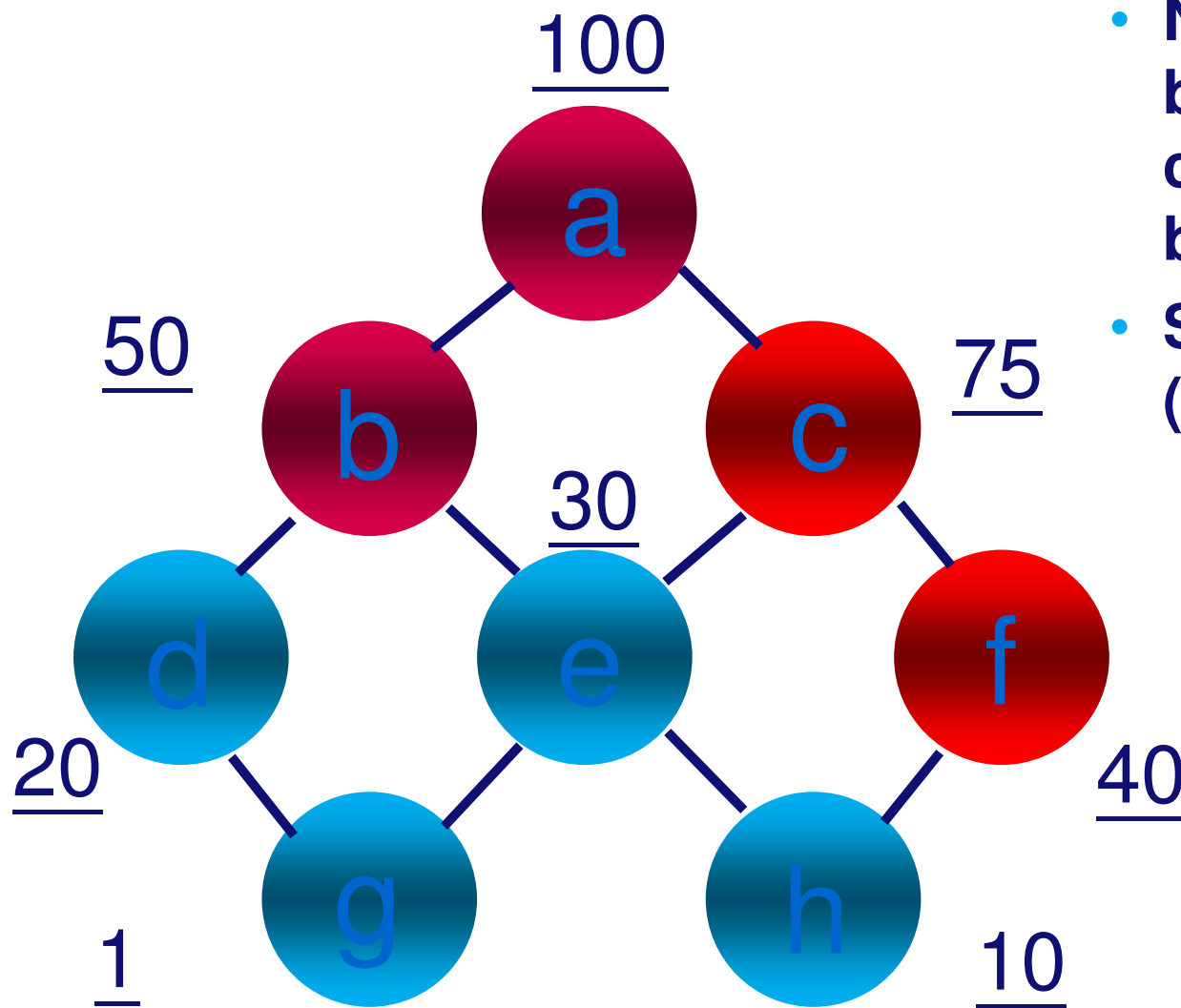
return S ;

Example



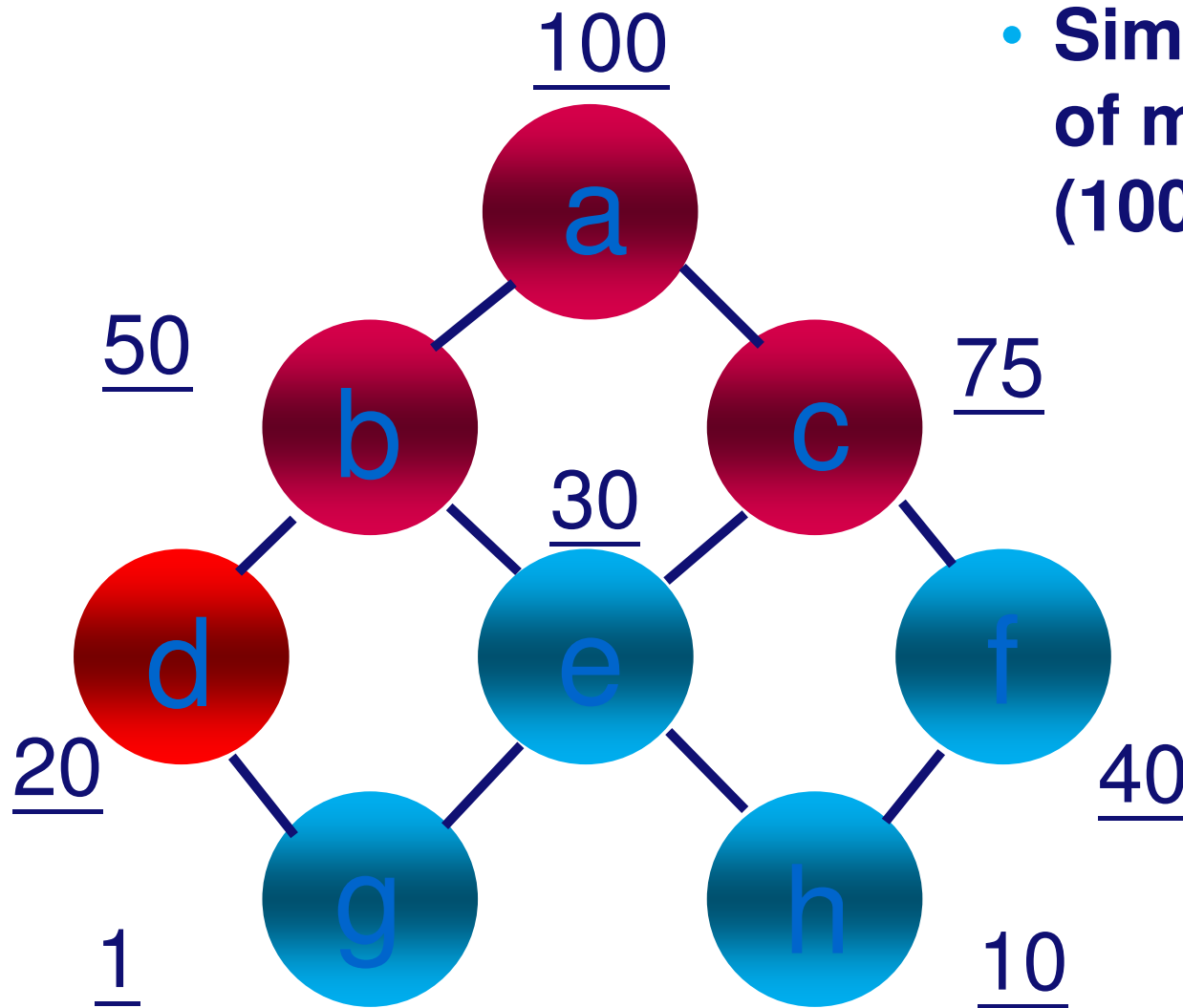
- E.g. The cost of constructing view b given the view A is 100
- If we choose b to materialize, the new cost of constructing view b is 50.

First round



- Notice that not only b, but also d, e, g and h can be calculated from b
- So the total benefit is $(100 - 50) \times 5 = 250$

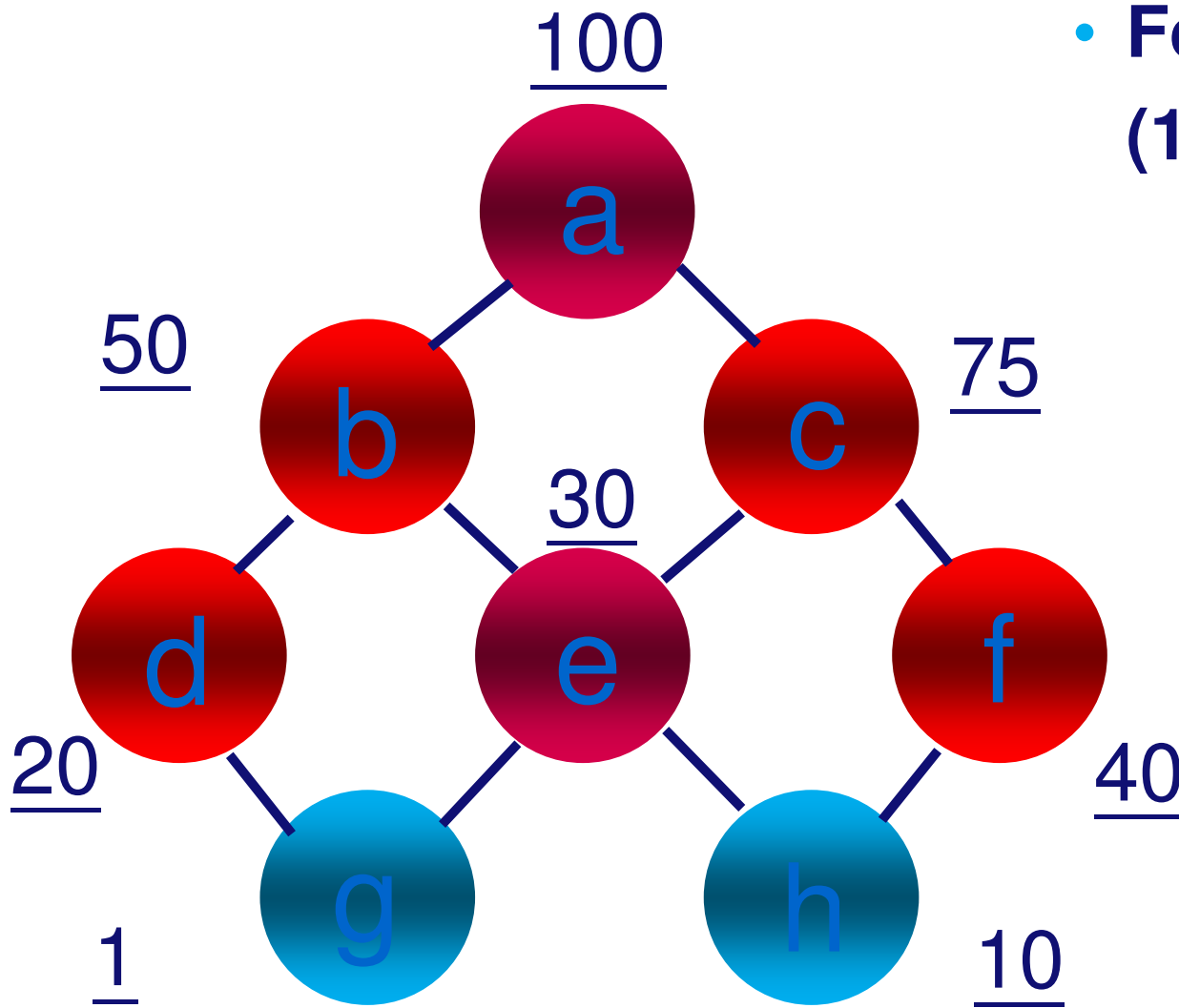
Continue...



- Similarly, the benefit of materializing c is $(100 - 75) \times 5 = 125$

	Benefit
b	250
c	125

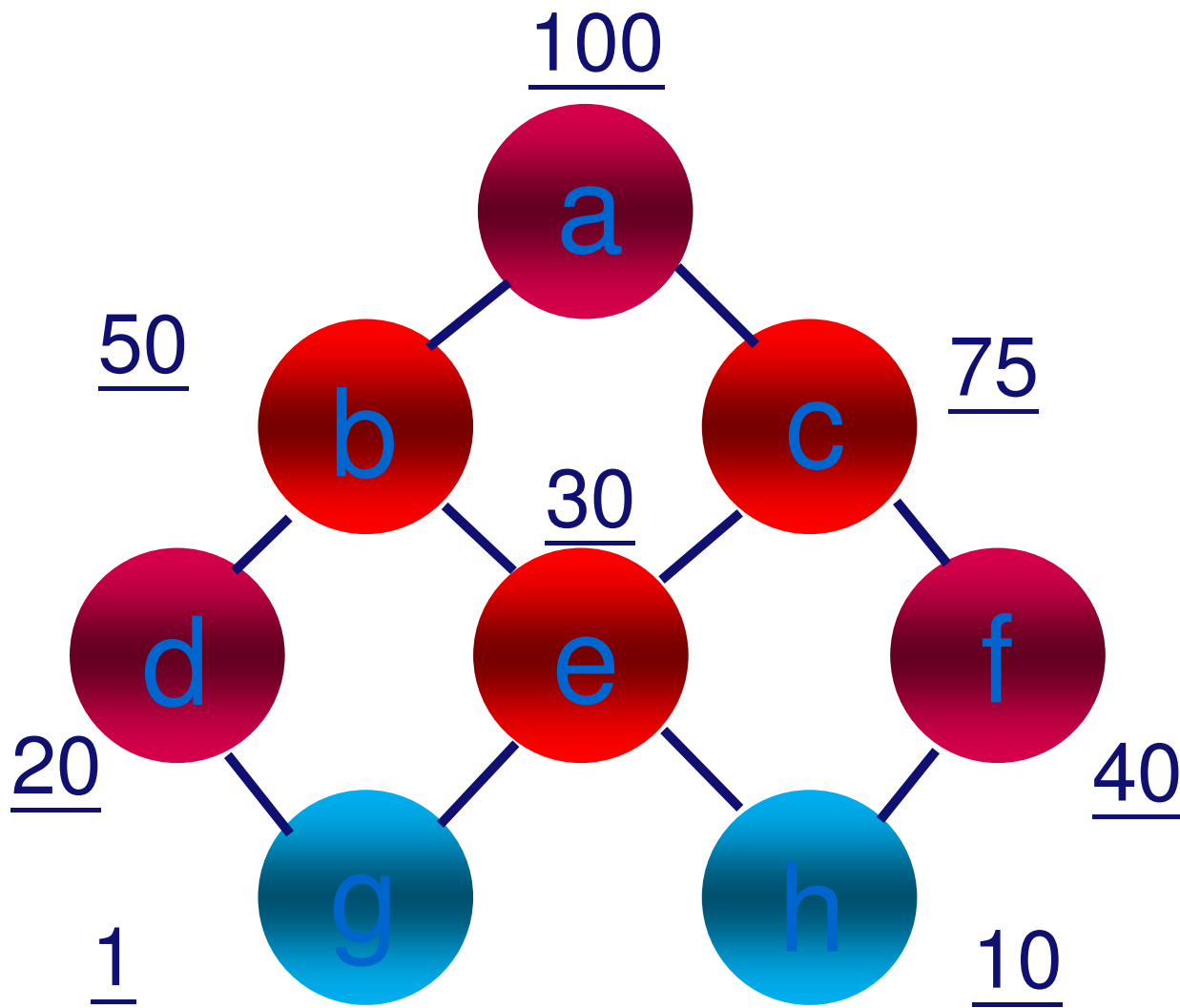
Not yet finish...



- For e, Benefit = $(100-30) \times 3 = 210$

	Benefit
b	250
c	125
e	210

Let's choose b!



- For d and f ,
Benefit =
 $(100-20) \times 2 = 160$
and
 $(100-40) \times 2 = 120$

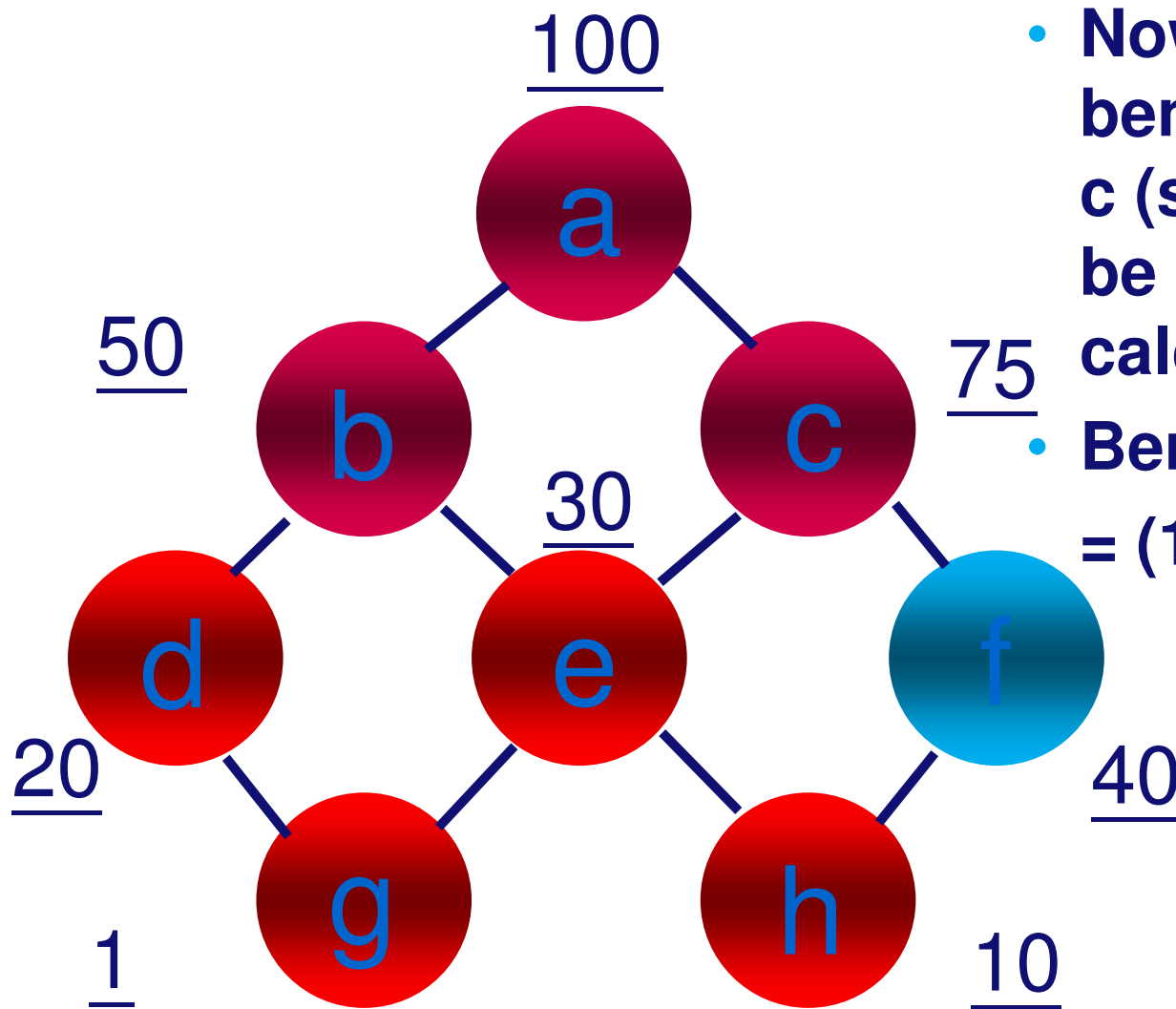
	Benefit
b	250
c	125
d	160
e	210
f	120

Next round?

- Seems we should choose e, as it has the second largest benefit.
- Let's see what will happen in the second round.

	Benefit
b	250
c	125
d	160
e	210
f	120

Second round!



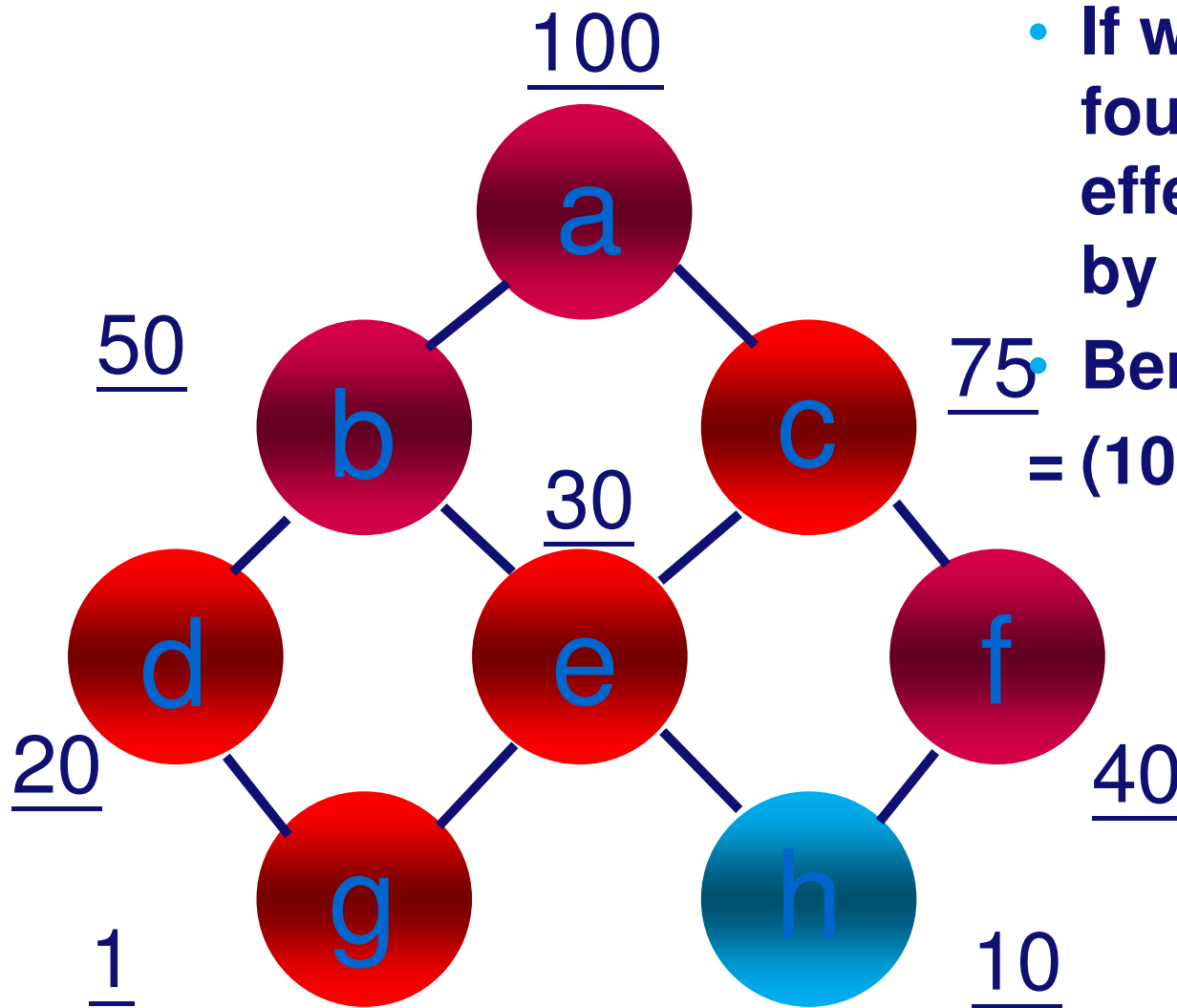
- Now, only c and f get benefit if we materialize c (since e, g and h can be more efficiently calculated by using b)

• Benefit

$$= (100 - 75) \times 2 = 50$$

	Benefit
c	50

How about choosing f?



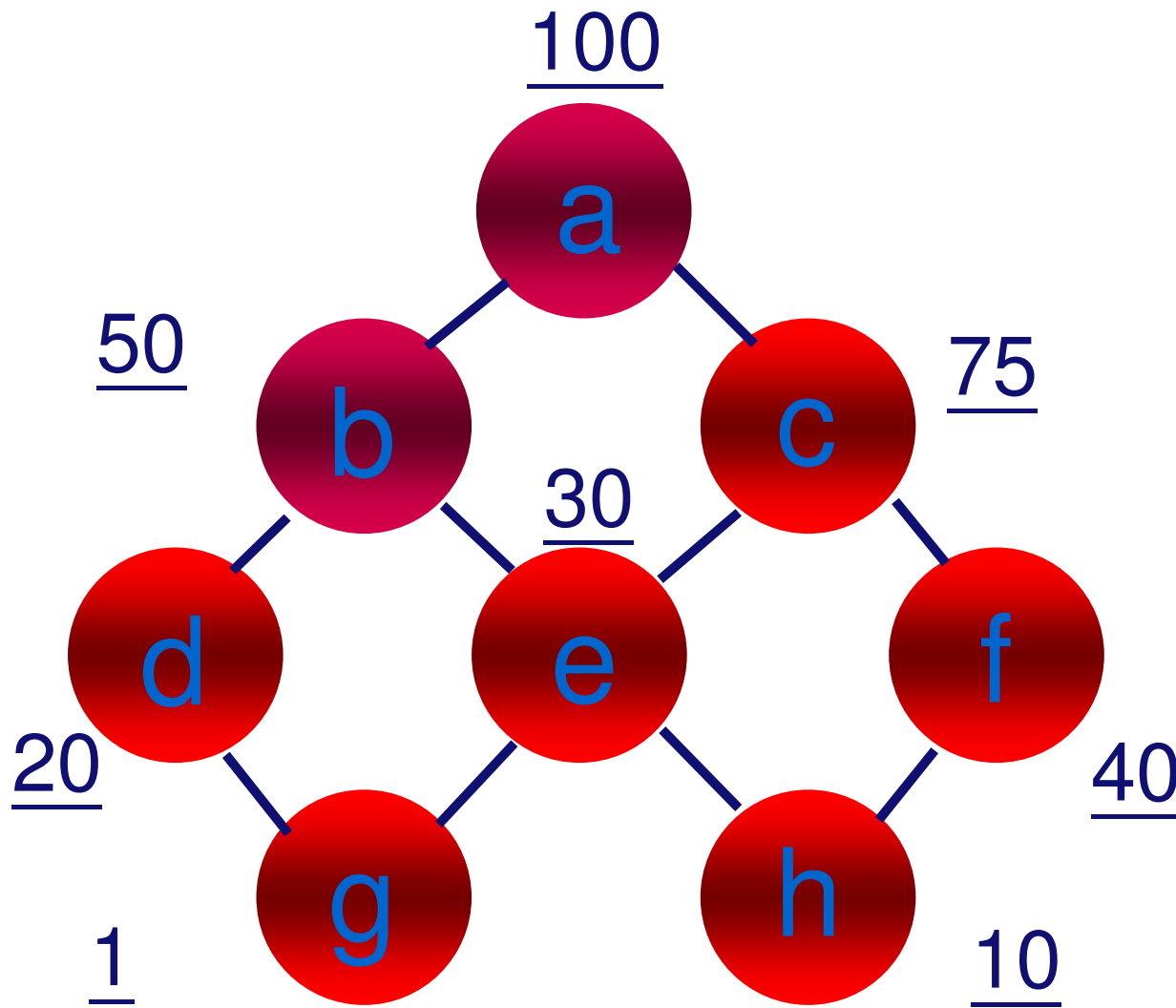
- If we choose f, we found that h can be effectively calculated by using f instead of b.

• Benefit

$$= (100 - 40) + (50 - 40)$$

	Benefit
c	50
f	70

Easy to work out others



- **Benefit of d**
 $= (50 - 20) \times 2 = 60$
- **Benefit of e**
 $= (50 - 30) \times 3 = 60$
- **Benefit of g**
 $= 50 - 1 = 49$
- **Benefit of h**
 $= 50 - 10 = 40$

Observation

- In the first round, the benefit of choosing f (only 120) is far from the best choice (250)
- But in second round, choosing f gives the maximum benefit!

1 st round	Benefit
b	250
c	125
d	160
e	210
f	120

2 nd round	Benefit
c	50
d	60
e	70
f	70
g	49

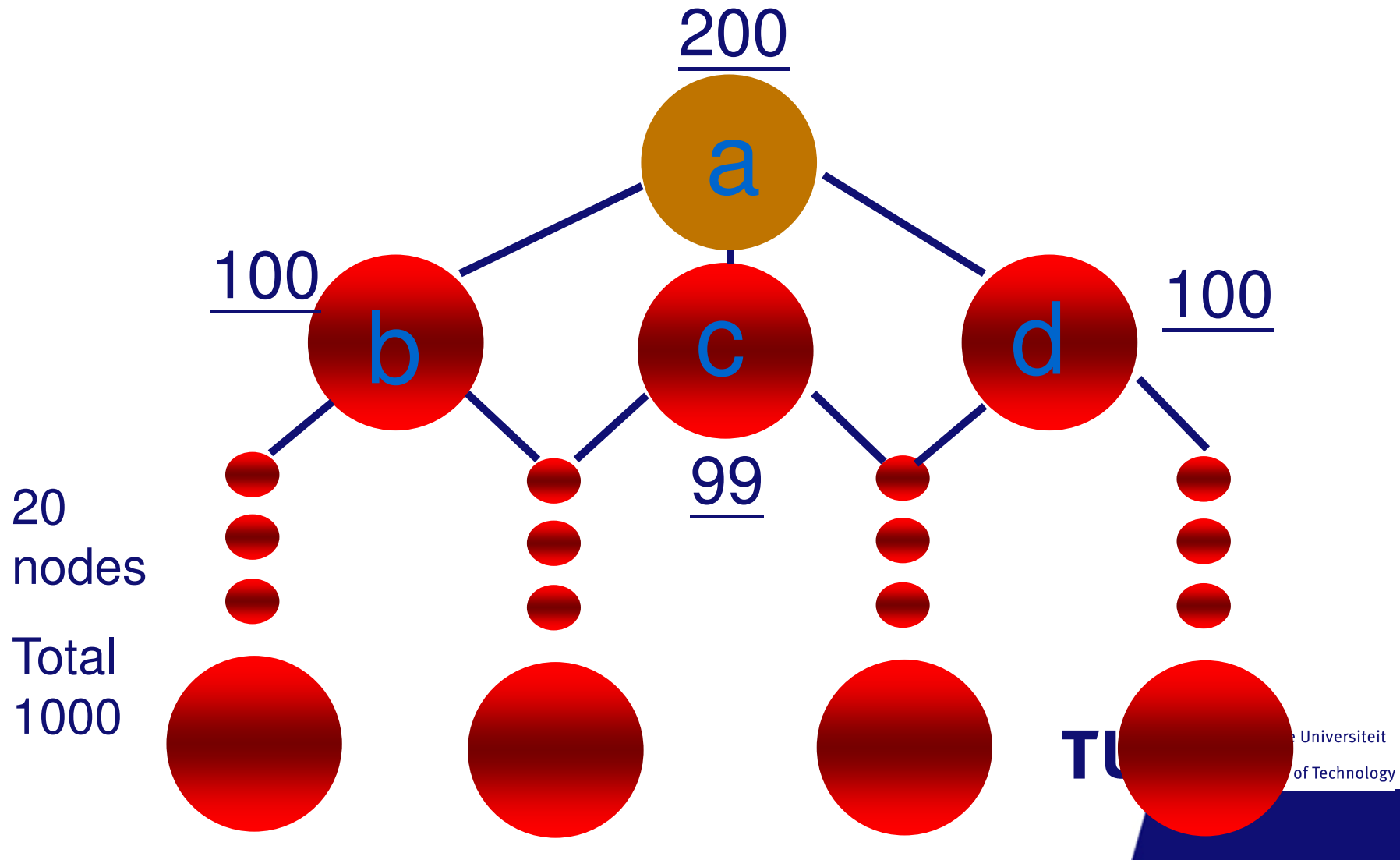
Overview

- Problem statement
- Formal model of computation
 - Partial order on Queries
 - Cost model
- Greedy solution
- Performance guarantee
- Conclusion

Simple? Optimal?

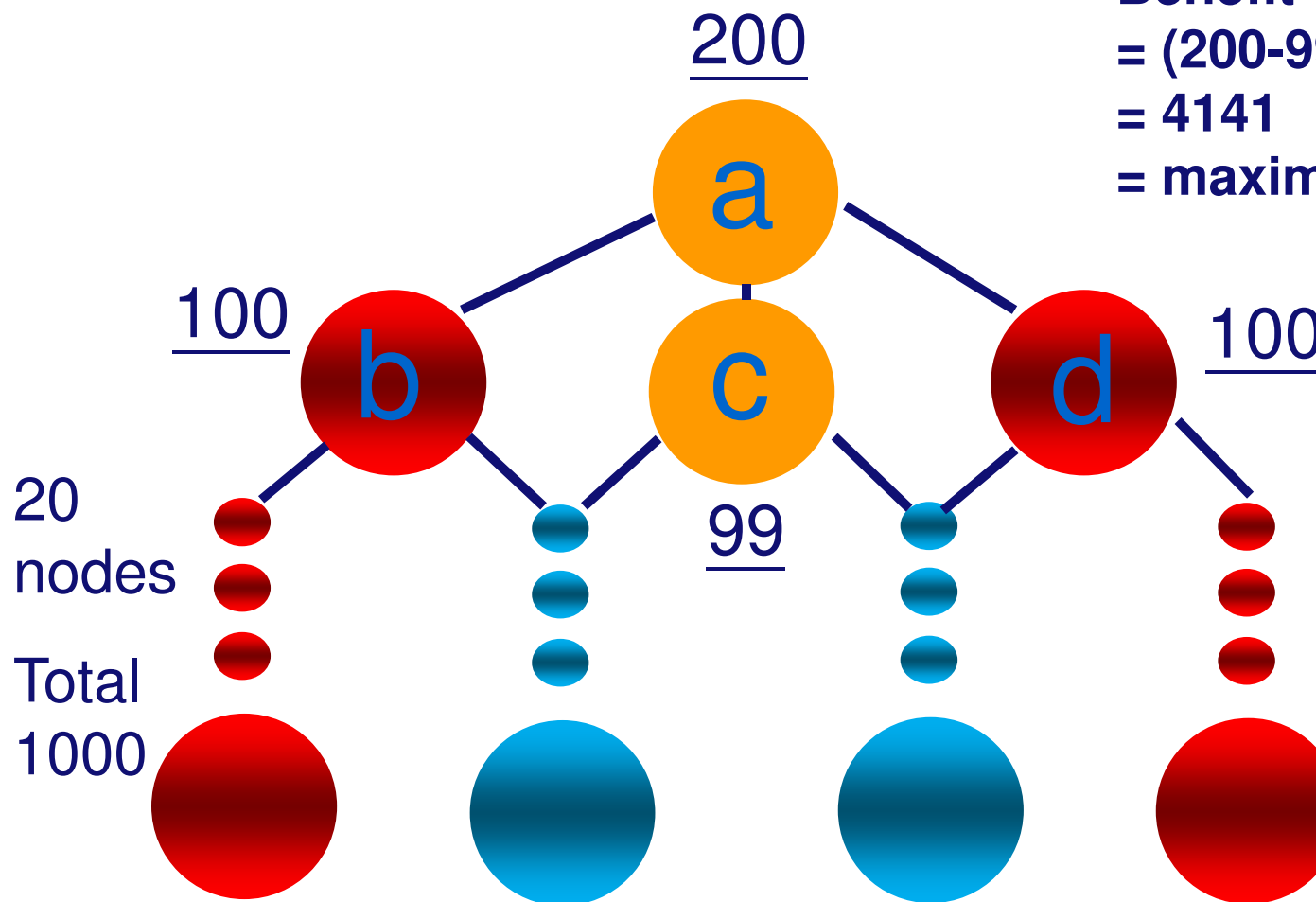
- **Trade off! This simple algorithm is not optimal in all cases!**
- **Consider the following case...**

Bad example

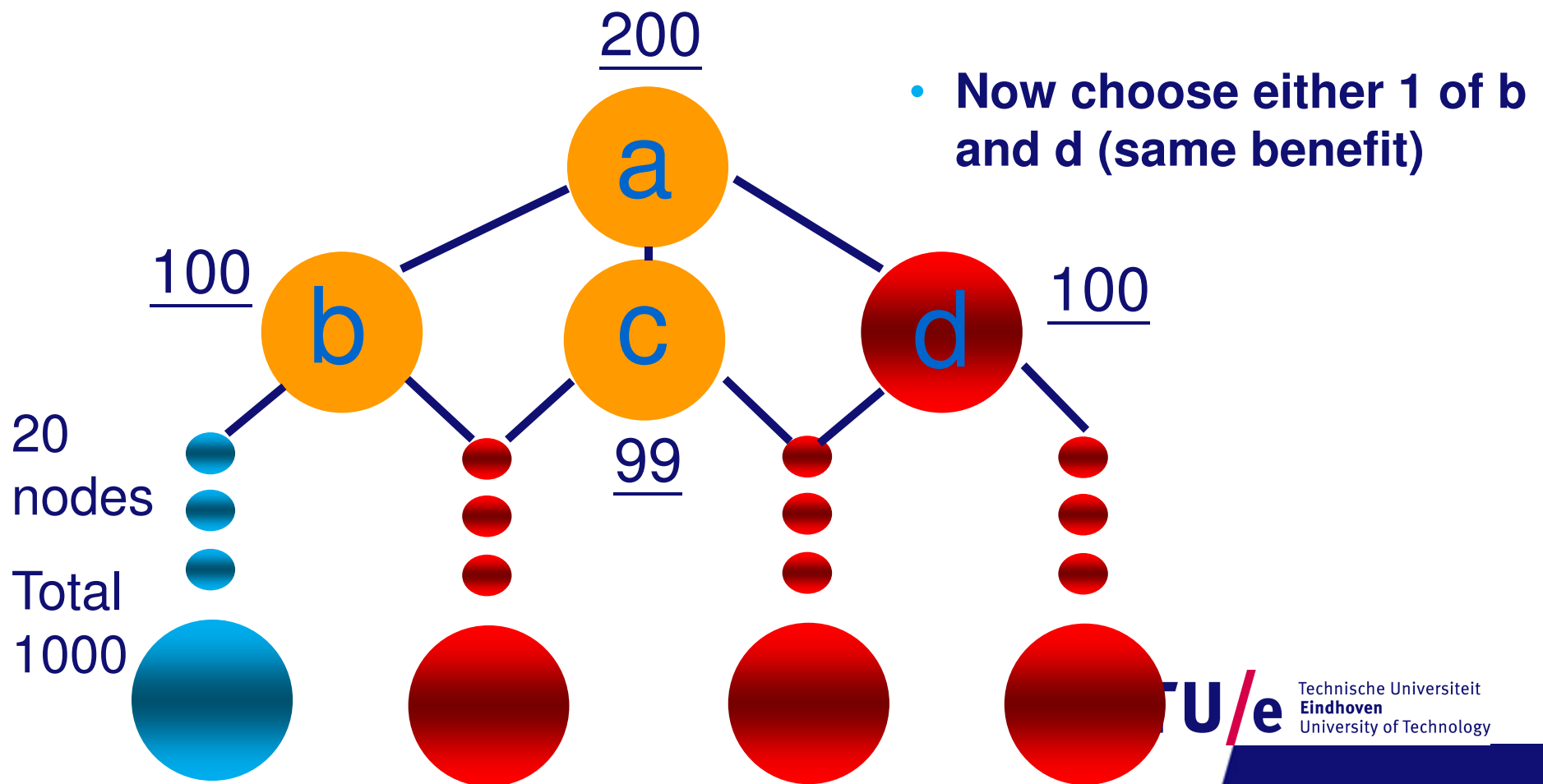


Bad example

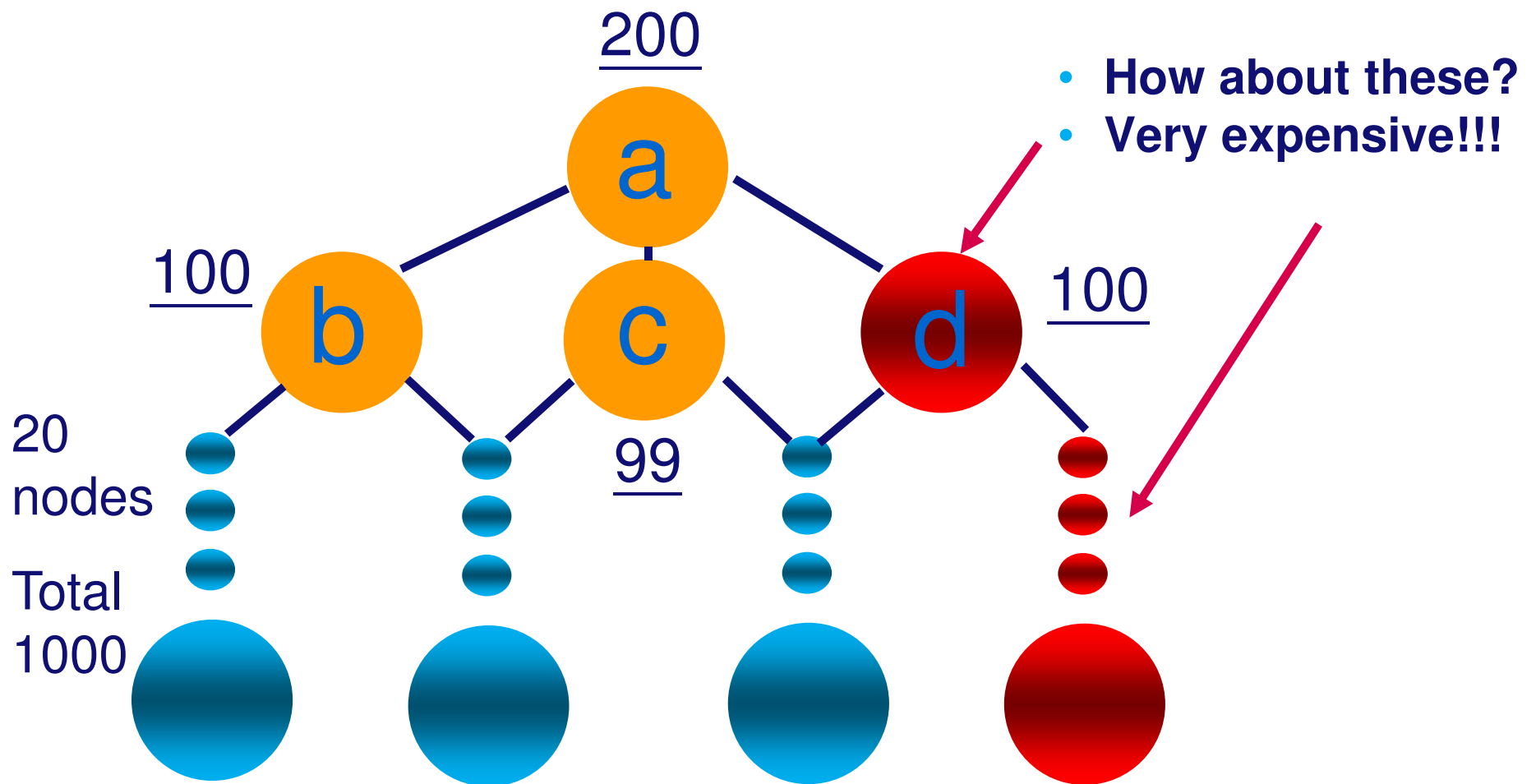
- Choose c
- Benefit
 $= (200 - 99) \times (1 + 20 + 20)$
 $= 4141$
 $= \text{maximum}$



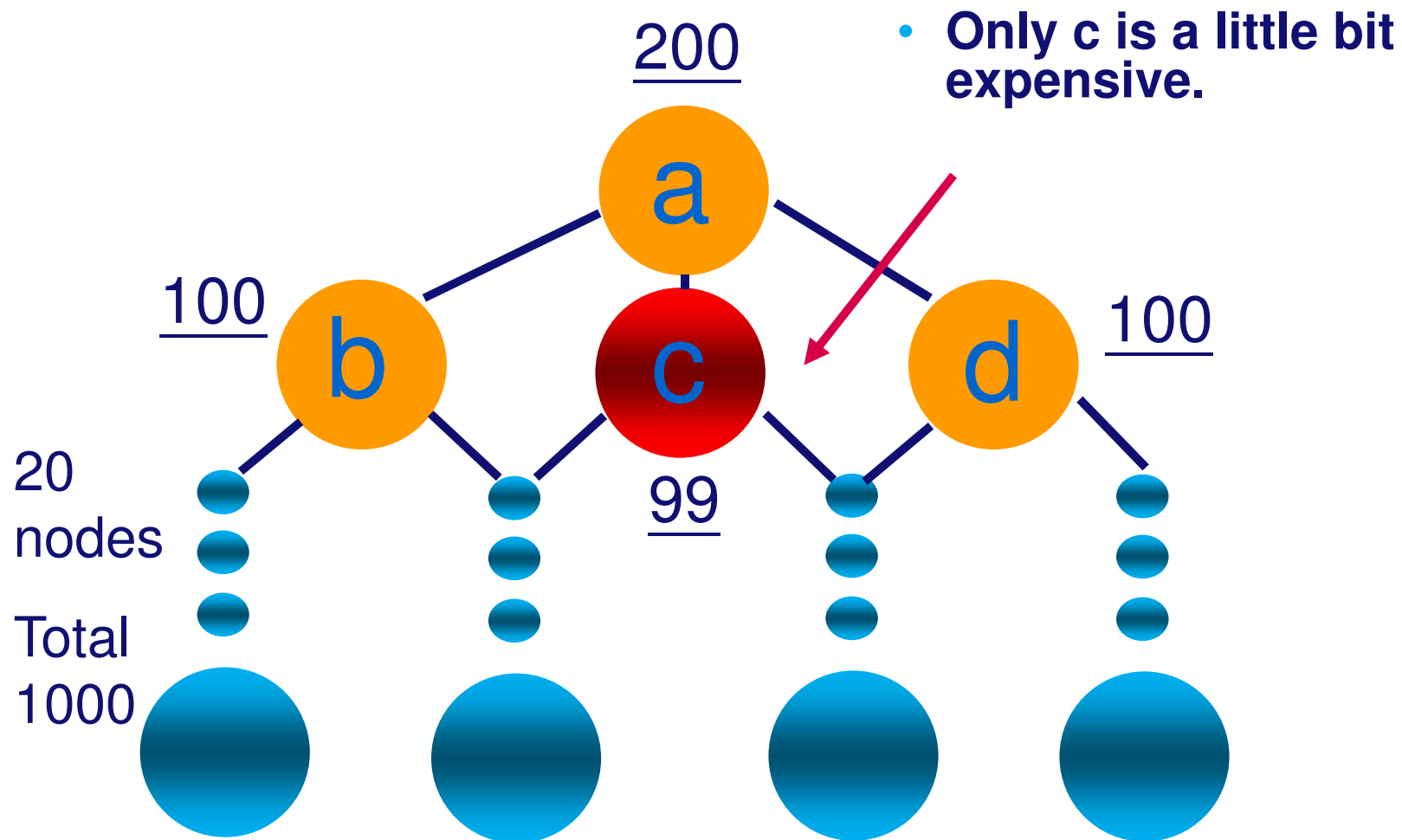
Bad example



Bad example



Optimal solution should be...



Some theoretical results

- It can be proved that we can get at least $(e - 1) / e$ (which is about 63%) of the benefit of the optimal algorithm.
- When selecting k views, bound is:

$$B_{\text{greedy}} / B_{\text{opt}} \geq 1 - \left(\frac{k-1}{k} \right)^k$$

There are lattices for which this ratio is arbitrarily close to this bound.

Extensions (1)

- **Problem**
 - The views in a lattice are unlikely to have the same probability of being requested in a query.
- **Solution:**
 - We can weight each benefit by its probability.

Extensions (2)

- **Problem**
 - Instead of asking for some fixed number (k) of views to materialize, we might instead allocate a fixed amount of space to views.
- **Solution**
 - We can consider the “benefit of each view per unit space”.

Conclusions Cube Materialization

- **Materialization of views is an essential query optimization strategy for decision-support applications.**
- **Reason to materialize some part of the data cube but not all of the cube.**
- **A lattice framework that models multidimensional analysis very well.**

Conclusions Cube Materialization

- Finding optimal solution is NP-hard.
- Introduction of greedy algorithm
- Greedy algorithm works on this lattice and picks the almost right views to materialize.
- There exists cases which greedy algorithm fails to produce optimal solution.
- But greedy algorithm has guaranteed performance
- Expansion of greedy algorithm.

II.2 Data storage and indexing

- **How is the data stored?**
 - **relational database (ROLAP)**
 - **Specialized structures (MOLAP)**
- **How can we speed up computation?**
 - **Indexing structures**
 - **bitmap index**
 - **join index**

Implementation

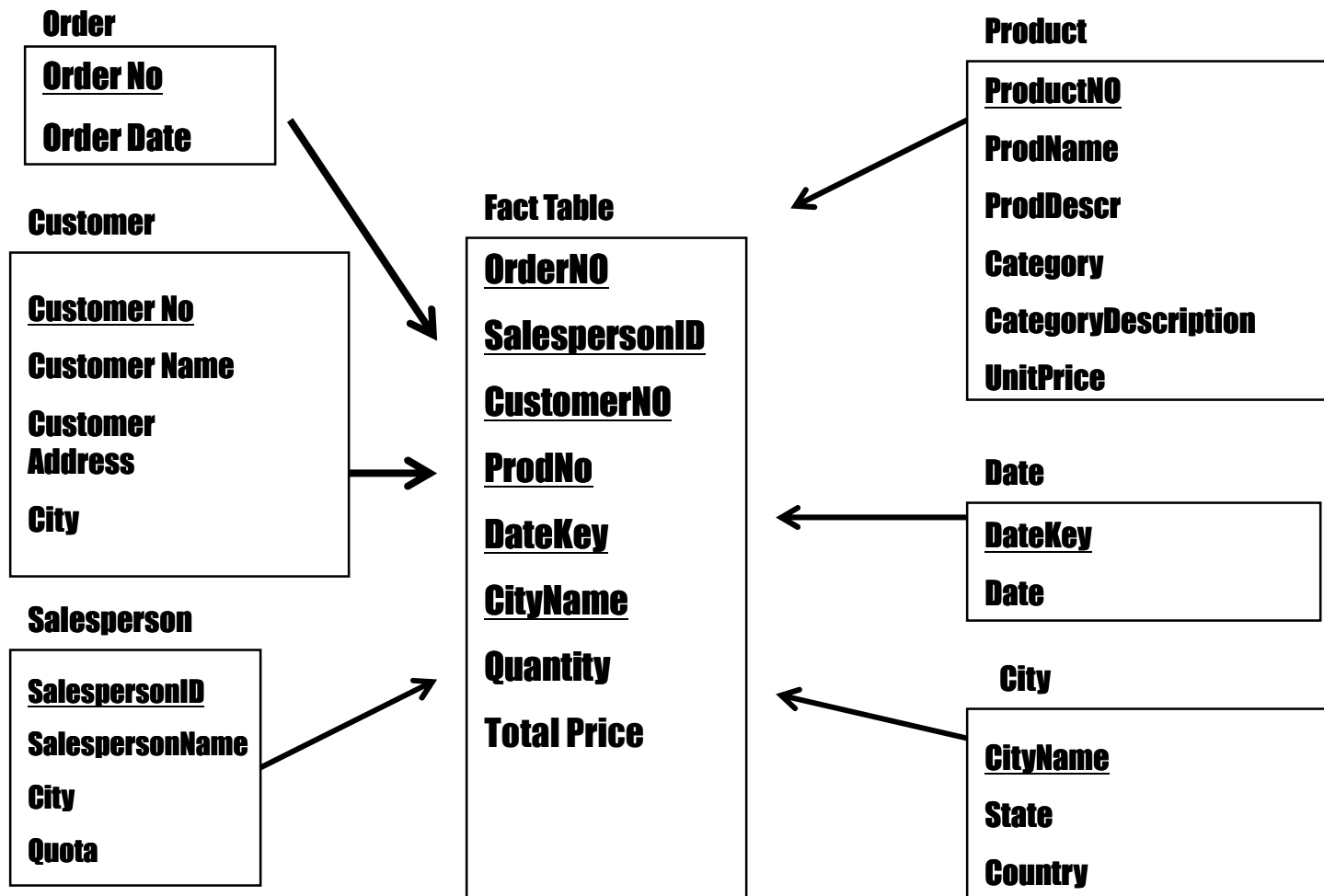
Nowadays systems can be divided in three categories:

- **ROLAP (Relational OLAP)**
 - OLAP supported on top of a relational database
- **MOLAP (Multi-Dimensional OLAP)**
 - Use of special multi-dimensional data structures
- **HOLAP: (Hybrid)**
 - combination of previous two

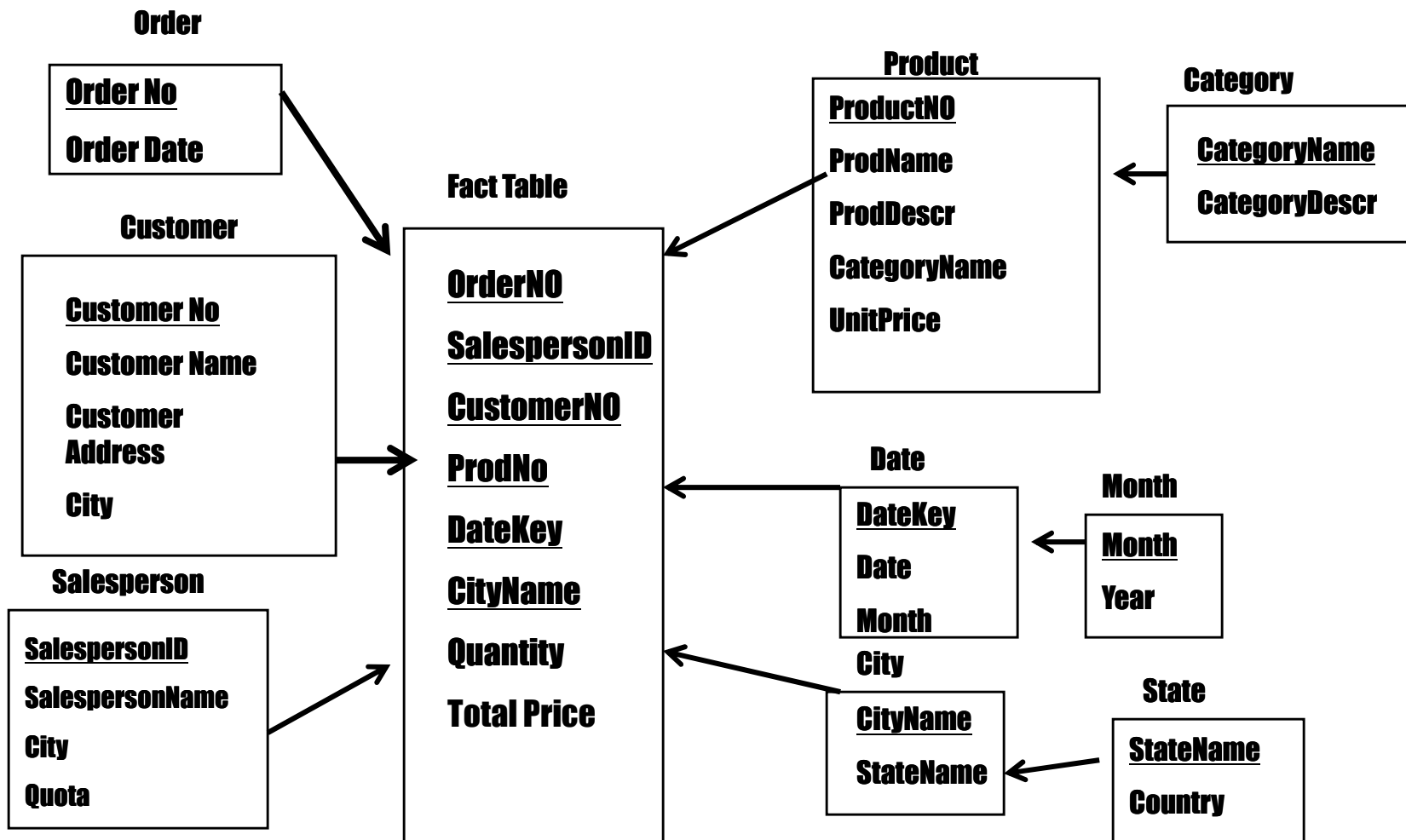
ROLAP

- **Typical database scheme:**
 - **star schema**
 - fact table is central
 - links to dimensional tables
 - **Extensions:**
 - snowflake schema
 - dimensions have hierarchy/extra information attached
 - Star constellation
 - multiple star schemas sharing dimensions

Example of a Star Schema

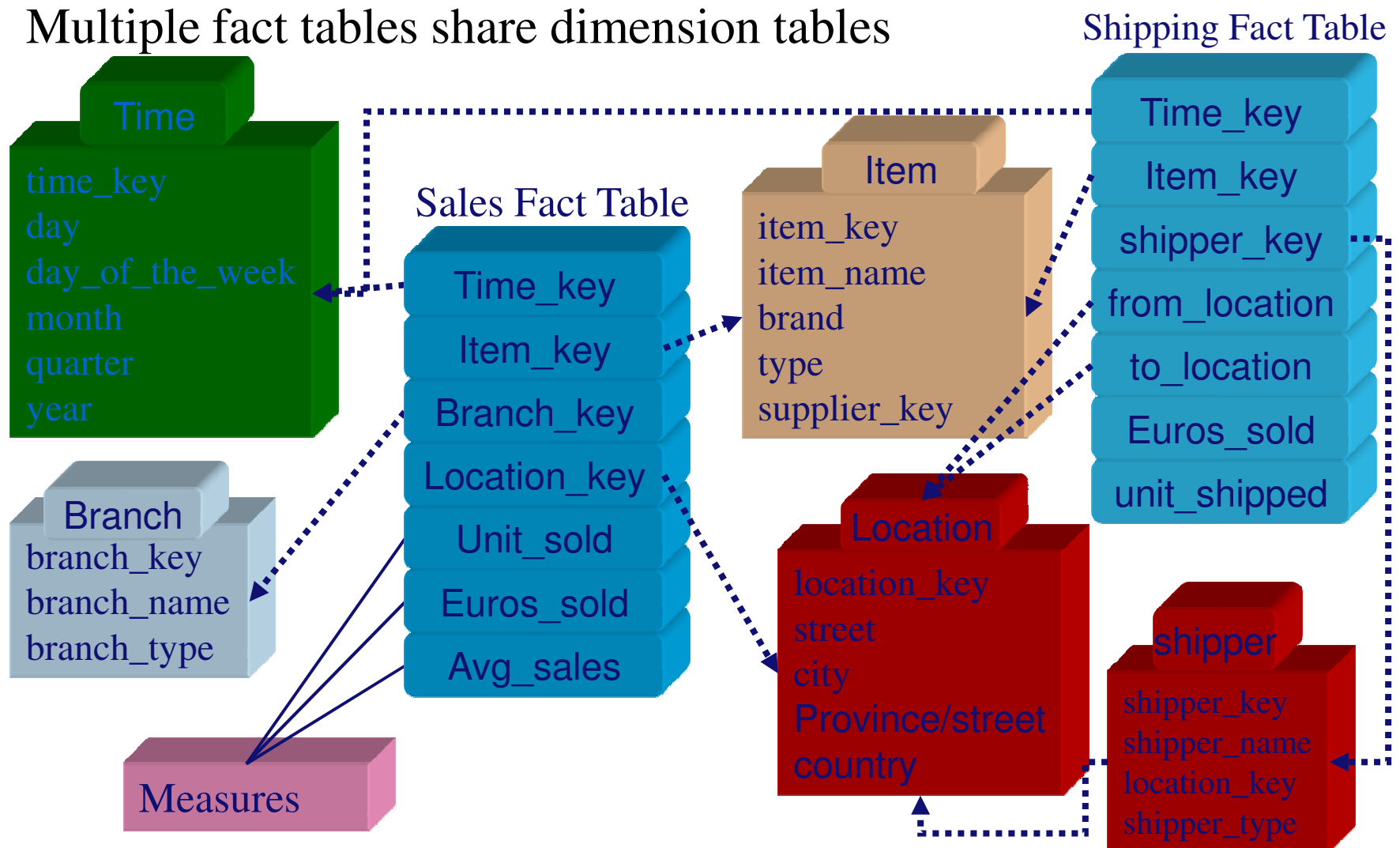


Example of a Snowflake Schema



Example of Fact Constellation

Multiple fact tables share dimension tables



This Lecture

- How is the data stored?
 - Relational database (ROLAP)
 - **Specialized structures (MOLAP)**
- How can we speed up computation?
 - Indexing structures
 - bitmap index
 - join index

MOLAP

- **Not on top of relational database**
 - most popular design
 - specialized data structures
 - Multicubes vs Hypercubes
 - Not all subcubes are materialized

Storing the cube

- User identifies set of *sparse* attributes S , and a set of *dense* attributes D .
- Index tree is constructed on sparse dimensions.
- Each leaf points to a multidimensional array indexed by D .

Example

- product, store are sparse dimensions
- date and customer-type are dense

prod. p

prod. p store s1

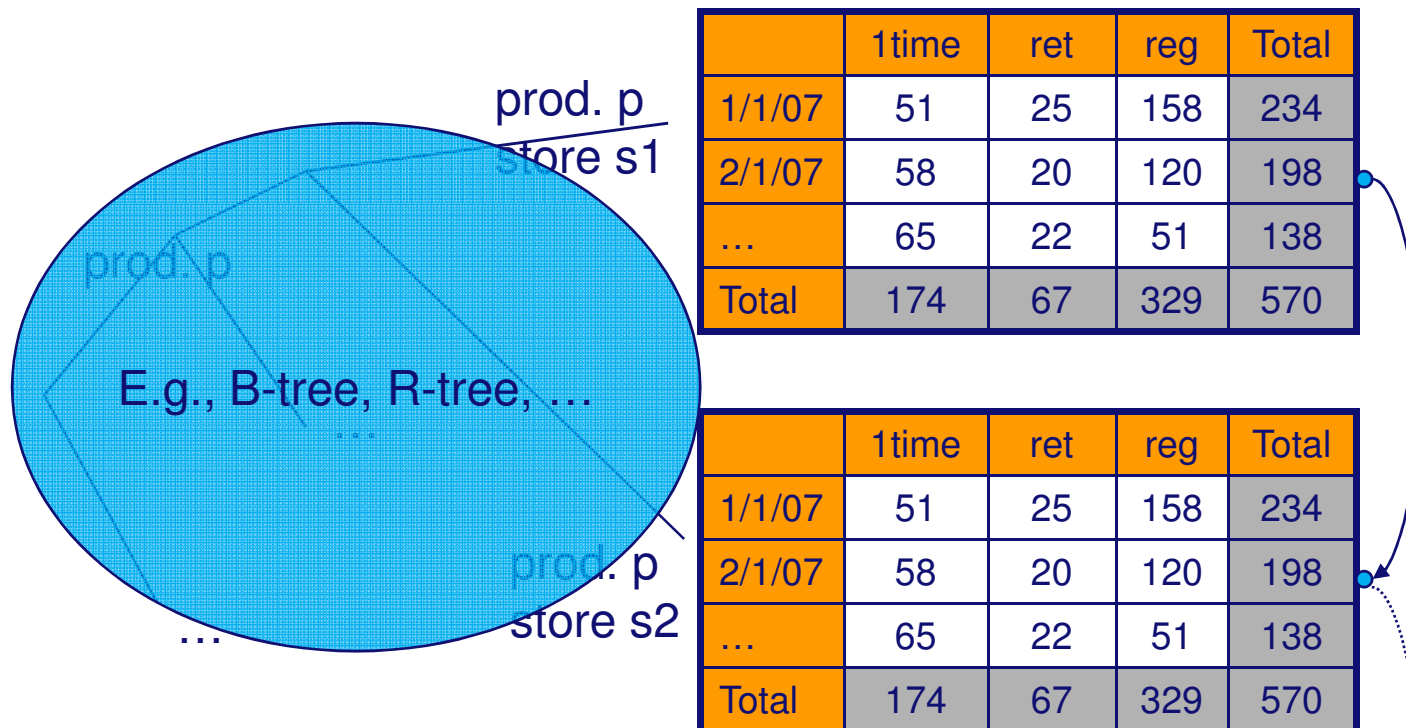
prod. p store s2

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
...	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
...	65	22	51	138
Total	174	67	329	570

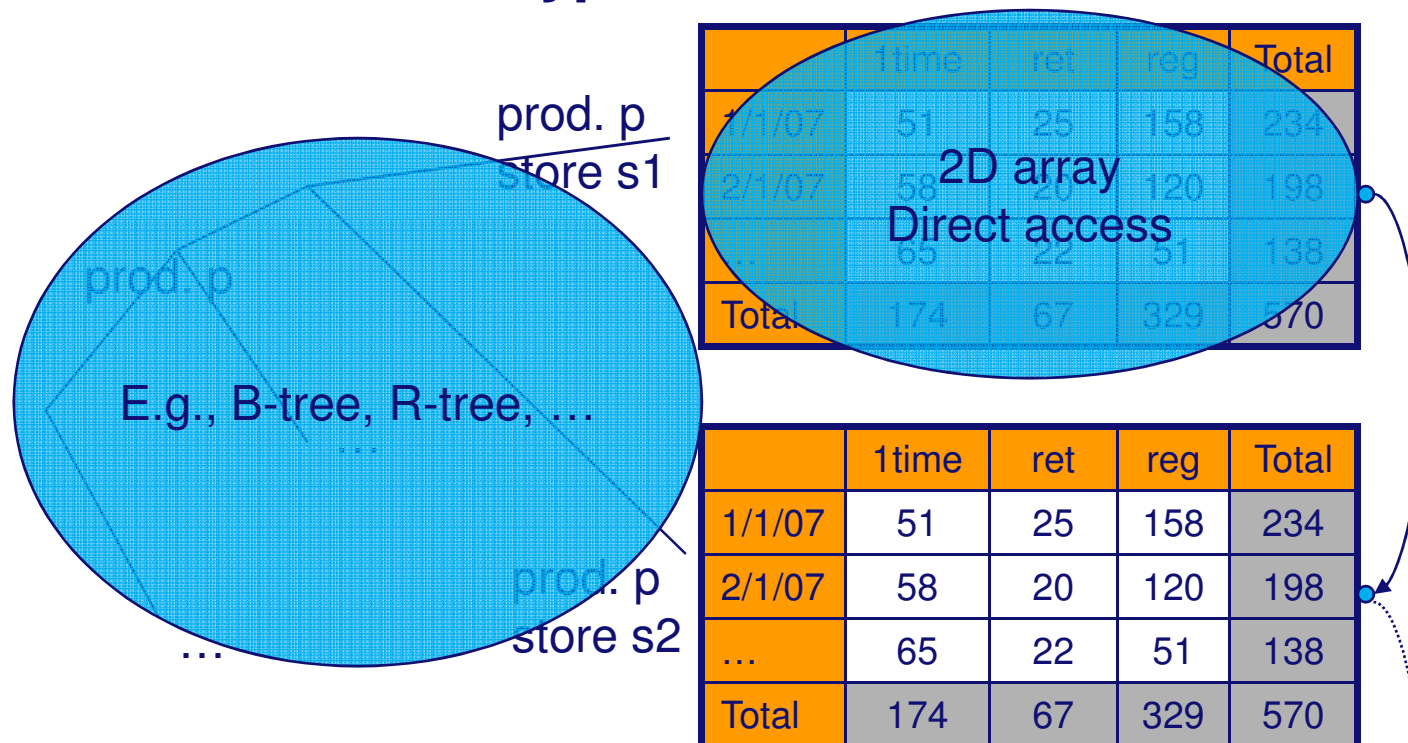
Example

- product, store are sparse dimensions
- date and customer-type are dense



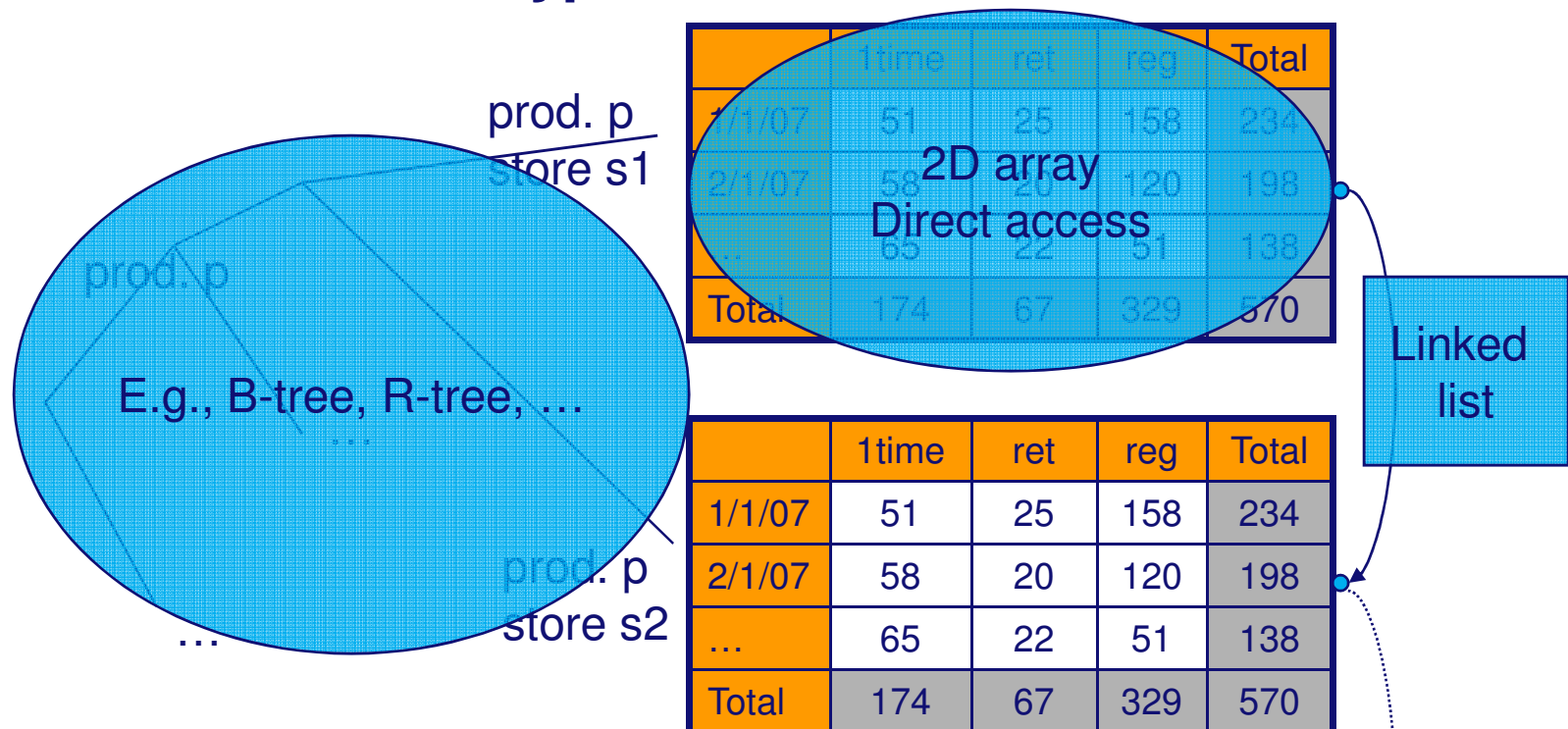
Example

- product, store are sparse dimensions
- date and customer-type are dense



Example

- product, store are sparse dimensions
- date and customer-type are dense

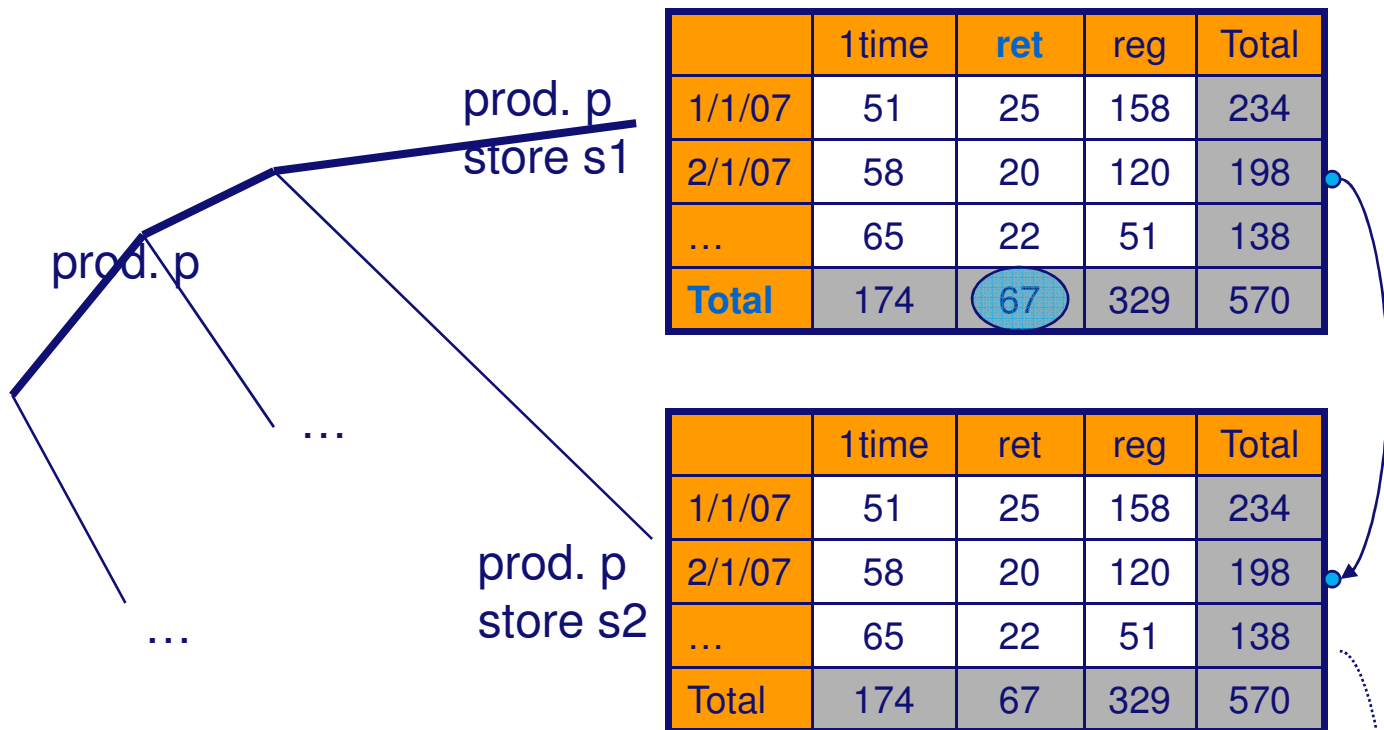


Queries

- **Efficiency depends on:**
 - **does index on sparse dimensions fit into memory?**
- **Type of queries:**
 - **Restrictions on all dimensions**
 - **Restrictions only on dense**
 - **Restrictions only on some sparse and dense**

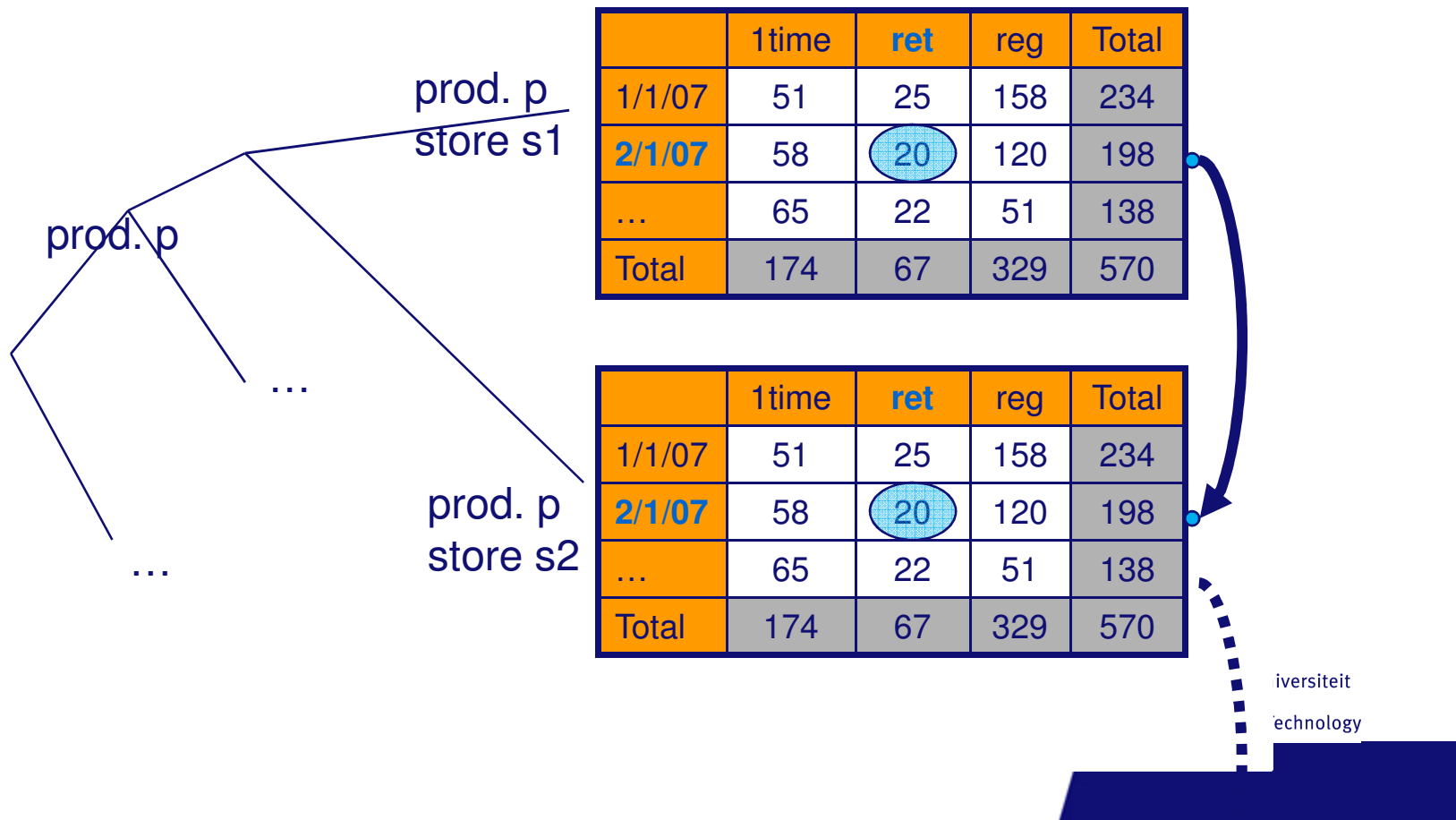
Queries

- Selection on all attributes: (p,s1,ret,all)



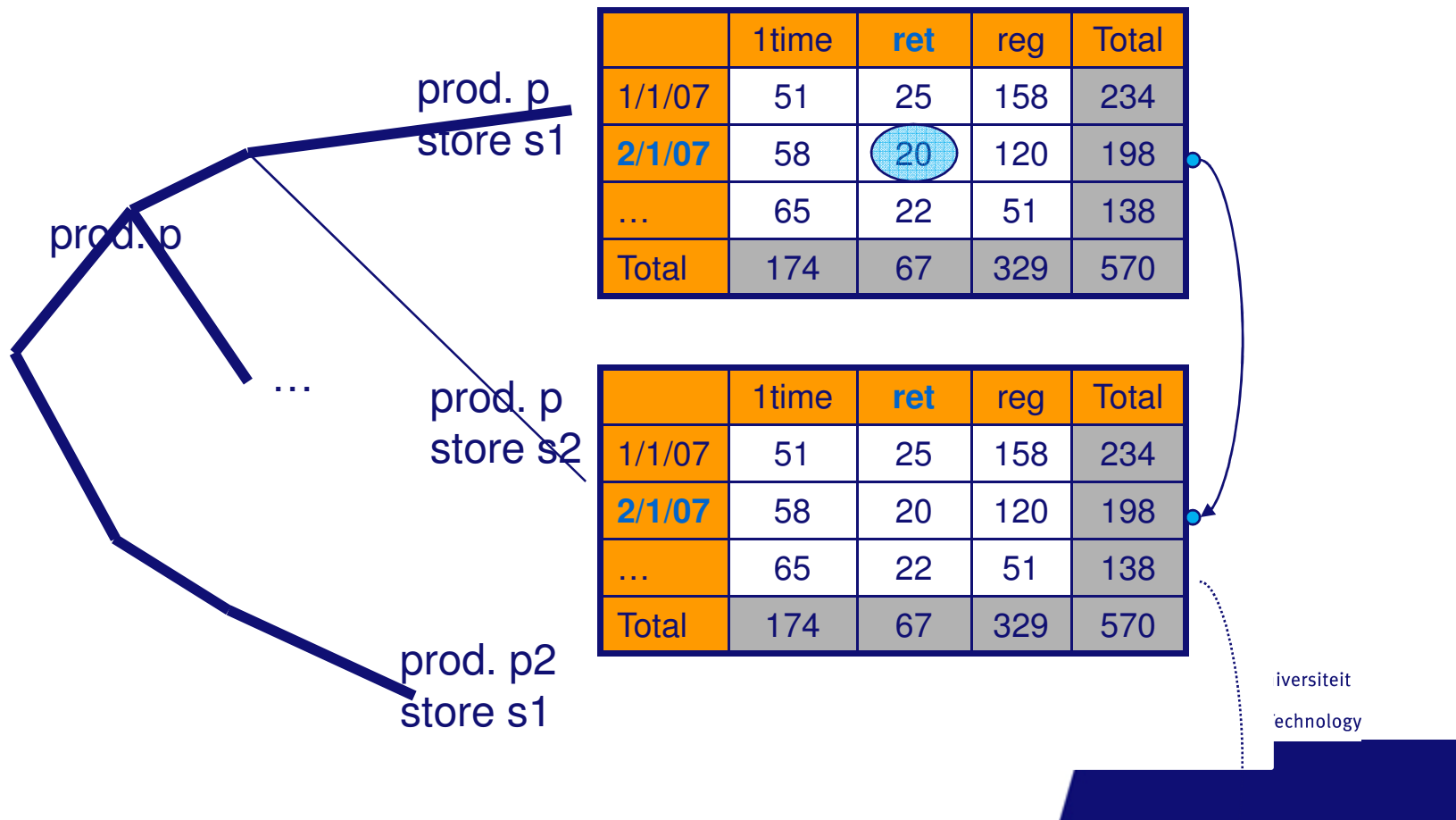
Queries

- Only on dense attributes: (-,-,ret,"2/1/07")



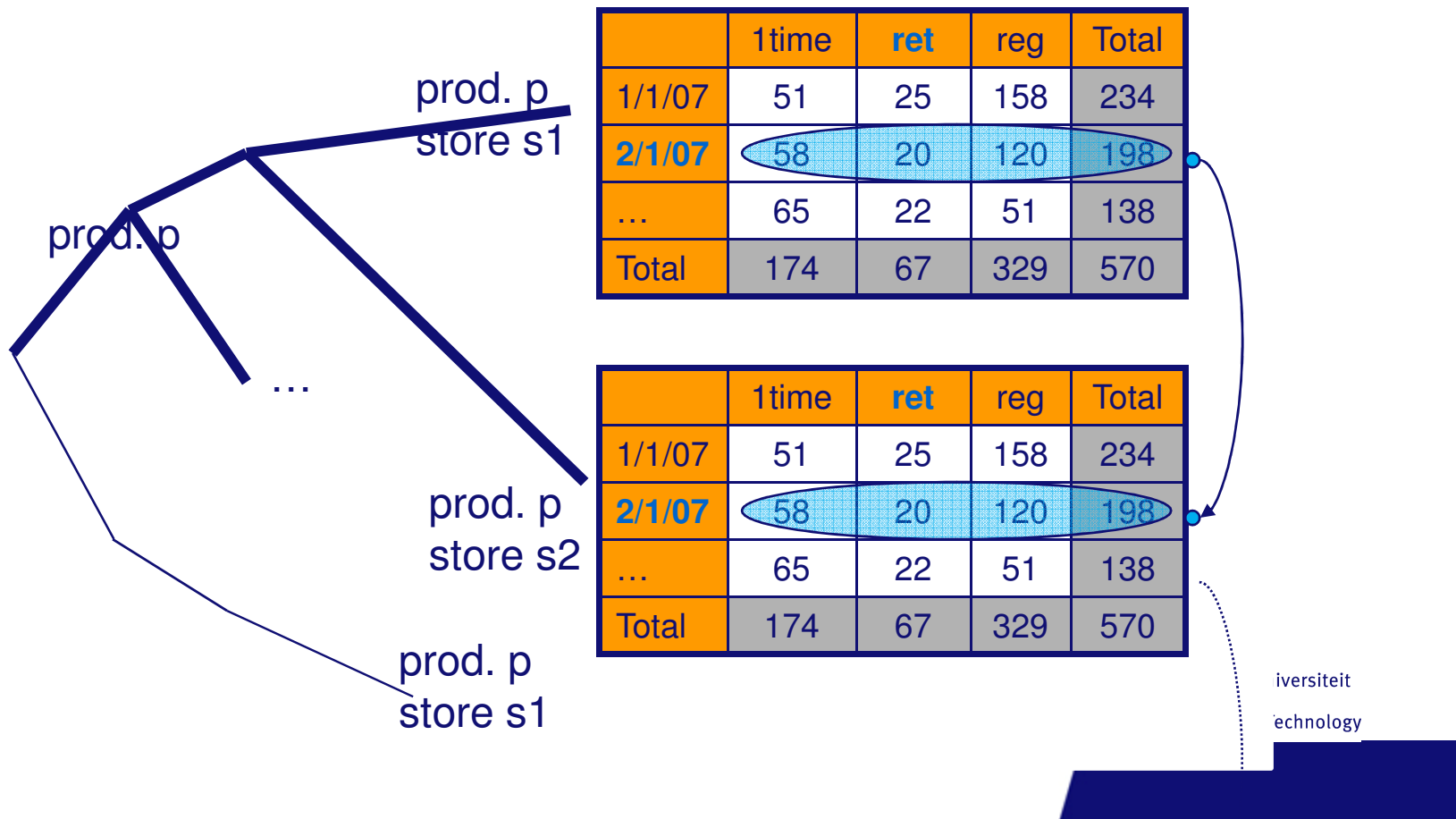
Queries

- Only some sparse and dense attributes:
(-,s1,ret,"2/1/07")



Queries

- Only some sparse and dense attributes:
(p,-,-,"2/1/07")



Storing the Cube

- Dense combinations of dimensions can be stored in multi-dimensional arrays
- For every combination of sparse dimensions
 - one sub-cube
- Sub-cubes indexed by sparse dimensions
 - E.g., B-tree
 - Order of the dimensions plays a role

This Lecture

- **How is the data stored?**
 - relational database (ROLAP)
 - Multi-dimensional structure (MOLAP)
- **How can we speed up computation?**
 - **Indexing structures**
 - bitmap index
 - join index

Specialized Indexing Structures

- B-trees, (covered in other courses)
- **Bitmapped indices,**
- **Join indices,**
- Spatial data structures

Index Structures

- **Indexing principle:**
 - mapping key values to records for associative direct access
- **Most popular indexing techniques in relational database: B+-trees**
- **For multi-dimensional data, a large number of indexing techniques have been developed: R-trees**

Bitmap Indexes

- **Bitmap index:** indexing technique that has attracted attention in multi-dimensional DB implementation

table

Customer	City	Car
c1	Detroit	Ford
c2	Chicago	Honda
c3	Detroit	Honda
c4	Poznan	Ford
c5	Paris	BMW
c6	Paris	Nissan

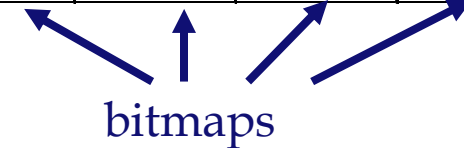
Bitmap Indexes

- The index consists of bitmaps:

ec1	Chicago	Detroit	Paris	Poznan
1	0	1	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1
5	0	0	1	0
6	0	0	1	0



ec1	BMW	Ford	Honda	Nissan
1	0	1	0	0
2	1	0	1	0
3	0	0	1	0
4	0	1	0	0
5	1	0	0	0
6	0	0	0	1

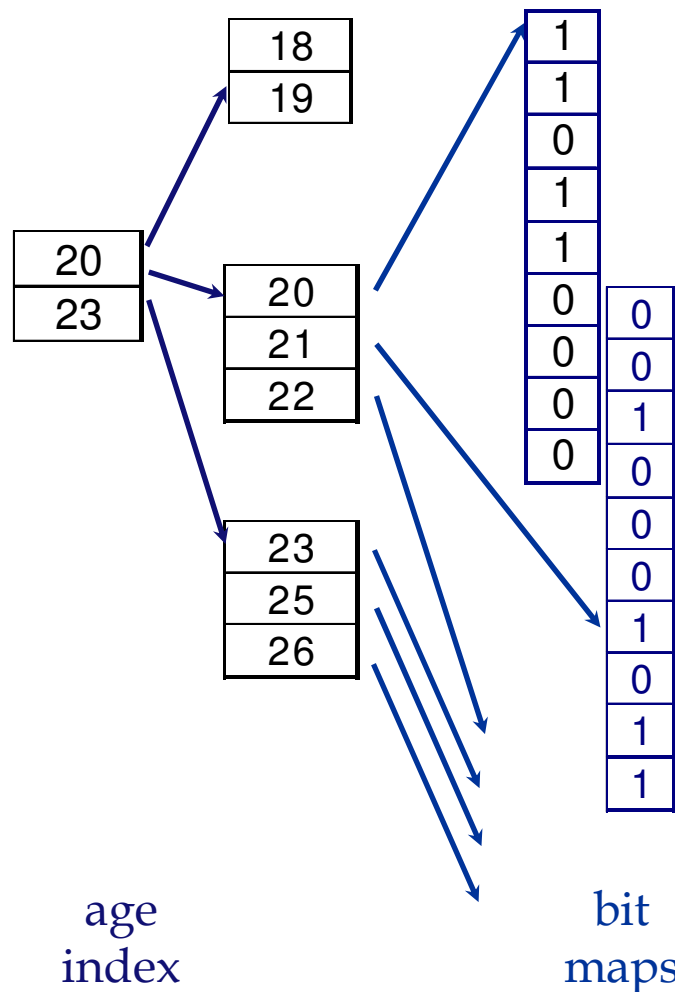


- Index on a particular column
- Index consists of a number of bit vectors - bitmaps
- Each value in the indexed column has a bit vector (bitmaps)
- The length of the bit vector is the number of records in the base table
- The i -th bit is set if the i -th row of the base table has the value for the indexed column

Bitmap Indexes

- Index on a particular column
- Index consists of a number of bit vectors - bitmaps
- Each value in the indexed column has a bit vector (bitmaps)
- The length of the bit vector is the number of records in the base table
- The i -th bit is set if the i -th row of the base table has the value for the indexed column

Bitmap Indexes



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

Query:

Get people with age =
20 and name = "fred"

List for age = 20:

1101100000

List for name = "fred":

0100000001

Answer is intersection:

0100000000

Suited well *for domains
with small cardinality*

Bitmap Index

- Size of bitmaps can be further reduced
 - use run-length encoding

1111000111100000001111000 is encoded as
4x1;3x0;4x1;7x0;4x1;3x0

- Can reduce the storage space significantly
- Logical operations can work directly on the encoding

Bitmap Index – Summary

- With efficient hardware support for bitmap operations (AND, OR, XOR, NOT), bitmap index offers better access methods for certain queries
 - e.g., selection on two attributes
- Some commercial products have implemented bitmap index
- Works poorly for high cardinality domains since the number of bitmaps increases
- Difficult to maintain - need reorganization when relation sizes change (new bitmaps)

This Lecture

- **How is the data stored?**
 - relational database (ROLAP)
 - Specialized structures (MOLAP)
- **How can we speed up computation?**
 - **Indexing structures**
 - bitmap index
 - **join index**

Join Indexes

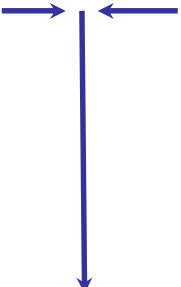
- **Traditional indexes: value \rightarrow rids.**
Join indices: tuples in the join \rightarrow to rids in the source tables.
- **Data warehouse:**
 - **values of dimensions of star schema \rightarrow rows in fact table.**
- **Join indexes can span multiple dimensions**

Join

SELECT * FROM SALE, PRODUCT

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

product	id	name	price
	p1	bolt	10
	p2	nut	5



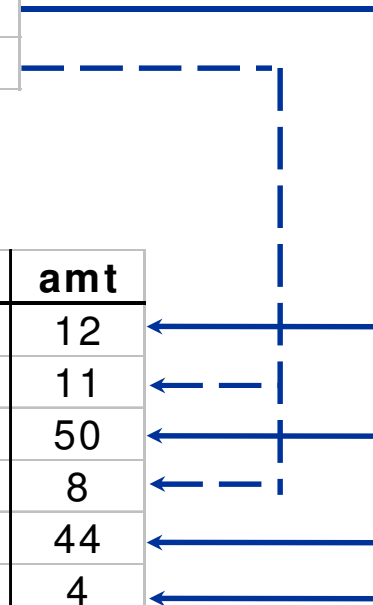
joinTb	prodId	name	price	storeId	date	amt
	p1	bolt	10	c1	1	12
	p2	nut	5	c1	1	11
	p1	bolt	10	c3	1	50
	p2	nut	5	c2	1	8
	p1	bolt	10	c1	2	44
	p1	bolt	10	c2	2	4

Join Indexes

join index

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

sale	rld	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



Summary

- **Data warehouse is a specialized database to support analytical queries = OLAP queries**
- **Data cube as conceptual model**
- **Implementation of Data Cube**
 - **View selection problem**
 - **Explosion problem**
 - **ROLAP vs. MOLAP**
 - **Indexing structures**