MSCIT 5210/MSCBD 5002: Knowledge Discovery and Data Mining

Acknowledgement: Slides modified by Dr. Lei Chen based on the slides provided by Jiawei Han, Micheline Kamber, and Jian Pei

1

Chapter 4: Data Warehousing, On-line Analytical Processing and Data Cube

- Data Warehouse: Basic Concepts 
- Data Warehouse Modeling: Data Cube and OLAP
- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Summary

Aspects of SQL

- Most common Query Language – used in all commercial systems
- Discussion is based on the SQL92 Standard. Commercial products have different features of SQL, but the basic structure is the same
- Data Manipulation Language
- Data Definition Language
- Constraint Specification
- Embedded SQL
- Transaction Management
- Security Management .....

Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

  select A_1, A_2, …, A_n
  from R_1, R_2, …, R_m
  where P

  - A_i represent attributes
  - R_i represent relations
  - P is a predicate.
- This query is equivalent to the relational algebra expression:

$$\Pi_{A1, A2, \ldots, An}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

- The result of an SQL query is a relation (but may contain duplicates). SQL statements can be nested.

Projection

- The select clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- Find the names of all branches in the *loan* relation

  select branch-name
  from loan

  Equivalent to: $\Pi_{\text{branch-name}}(\text{loan})$

- An asterisk in the select clause denotes "all attributes"

  select *
  from loan

- Note: for our examples we use the tables:
  - Branch (branch-name, branch-city, assets)
  - Customer (customer-name, customer-street, customer-city)
  - Loan (loan-number, amount, *branch-name*)
  - Account (account-number, balance, *branch-name*)
  - Borrower (*customer-name, loan-number*)
  - Depositor (*customer-name, account-number*)

Duplicate Removal

- SQL allows duplicates in relations as well as in query results. Use select distinct to force the elimination of duplicates.
  Find the names of all branches in the loan relation, and remove duplicates

```
select distinct branch-name
from loan
```

force the DBMS to remove duplicates

- The keyword all specifies that duplicates are not removed.

```
select all branch-name
from loan
```

force the DBMS not to remove duplicates

Arithmetic Operations on Retrieved Results

- The select clause can contain arithmetic expressions involving the operators,+,−,÷ and ×, and operating on constants or attributes of tuples.
- The query:

```
select branch-name, loan-number, amount * 100
from loan
```

would return a relation which is the same as the loan relations, except that the attribute amount is multiplied by 100

The where Clause

- The where clause specifies conditions that tuples in the relations in the from clause must satisfy.
- Find all loan numbers for loans made at the Perryridge branch with loan amounts greater than $1200.

```
select loan-number
from loan
where branch-name="Perryridge" and amount >1200
```

- SQL allows logical connectives and, or, and not. Arithmetic expressions can be used in the comparison operators.
- Note: attributes used in a query (both select and where parts) must be defined in the relations in the from clause.

The where Clause (Cont.)

- SQL includes the between operator for convenience.
- Find the loan number of those loans with loan amounts between \$90,000 and \$100,000 (that is, $\geq$ \$90,000 and $\leq$ \$100,000)

```
select loan-number
from loan
where amount between 90000 and
100000
```

The from Clause

- The from clause corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product borrower × loan

```
select *
from borrower, loan
```

  It is rarely used without a where clause.

- Find the name and loan number of all customers having a loan at the Perryridge branch.

```
select distinct customer-name, borrower.loan-number
from borrower, loan
where borrower.loan-number = loan.loan-number and
branch-name = "Perryridge"
```

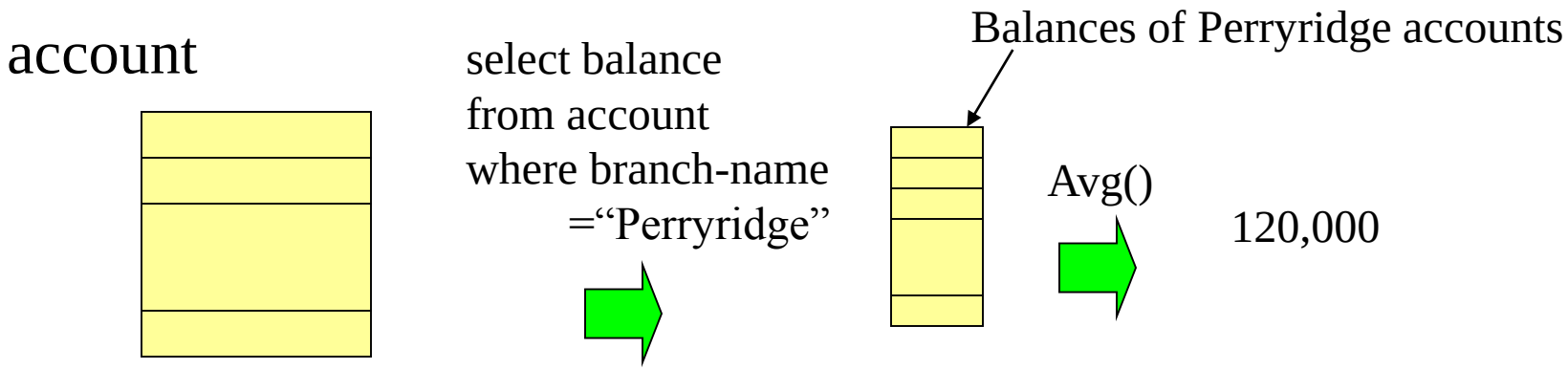
Aggregate Functions

- Operate on a column of a relation, and return a value
  - avg: average value
  - min: minimum value
  - max: maximum value
  - sum: sum of values
  - count: number of values
- Note: for our examples we use the tables:
  - Branch (branch-name, branch-city, assets)
  - Customer (customer-name, customer-street, customer-city)
  - Loan (loan-number, amount, *branch-name*)
  - Account (account-number, balance, *branch-name*)
  - Borrower (*customer-name, loan-number*)
  - Depositor (*customer-name, account-number*)

Aggregate Function Computation

- Find the average account balance at the Perryridge branch.

```
select avg(balance)
from account
where branch-name="Perryridge"
```



Examples of Aggregate Functions

- Find the numbers of tuples in the customer relation.

```
select count(*)
from customer
```

  - remember * stands for all attributes
  - Same as:

```
select count(customer-city)
from customer
```

  - Different from:

```
select count(distinct customer-city)
from customer
```

  - Because customer-city is not a key

Group by

- Find the number of accounts for *each* branch.

```
select branch-name, count(account-number)
from account
group by branch-name
```

- For each group of tuples with the same branch-name, apply aggregate function count and distinct to account-number

branch-name	account-number	balance
Perryridge	a-102	400
Brighton	a-217	750
Perryridge	a-201	900
Brighton	a-215	750
Redwood	a-222	700

account table

branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

branch-name	count-account-no
Perryridge	2
Brighton	2
Redwood	1

Group by Attributes

- Attributes in select clause outside of aggregate functions must appear in group by list, why?

```
select branch-name, balance, count( distinct account-number)
from account
group by branch-name
```

correct

```
select … from account
group by branch-name, balance
```

OR

```
select branch-name, sum(balance), count(…)
from account  group by branch-name
```

branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

Group by with Join

- Find the number of depositors for each branch.

```
select branch-name, count( distinct customer-name)
from depositor, account
where depositor.account-number = account.account-number
group by branch-name
```

- Perform Join then group by then count (distinct ())

  depositor (customer-name, account-number)
  account (branch-name, account-number, balance)
  Join $\Rightarrow$ (customer-name, account-number, branch-name, balance)

- Group by and aggregate functions apply to the Join result

Group by Evaluation

```
select branch-name, customer-name
from depositor, account
where depositor.account-number
= account.account-number
```

branch-name	cust-name
Perryridge	John Wong
Downtown	Pat Lee
Uptown	John Wong
Perryridge	Jacky Chan
Uptown	Mary Kwan
Downtown	John Wong
Perryridge	John Wong
Downtown	May Cheung

group by

branch-name	cust-name
Perryridge	John Wong
Perryridge	Jacky Chan
Perryridge	John Wong
Uptown	John Wong
Uptown	Mary Kwan
Downtown	John Wong
Downtown	Pat Lee
Downtown	May Cheung

distinct

branch-name	cust-name
Perryridge	John Wong
Perryridge	Jacky Chan
Uptown	John Wong
Uptown	Mary Kwan
Downtown	John Wong
Downtown	Pat Lee
Downtown	May Cheung

count

branch-name	count
Perryridge	2
Uptown	2
Downtown	3

Having Clause

- Find the names of all branches where the average account balance is more than $700

```
select branch-name, avg(balance)
from account
group by branch-name
having avg (balance) >700
```

- predicates in the **having** clause are applied to *each group* after the formation of groups

branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

Group-by

- Motivation: Group-by permits us to display aggregate results (e.g., max, min, sum) for groups. For instance, if we have GROUP-BY X, we will get a result for every different value of X.
- Recall that aggregate queries without group-by return just a single number.
- If we put an attribute in SELECT, the attribute must also appear in GROUP-BY. The opposite is not true: there may be attributes in GROUP-BY that do not appear in SELECT.
- Any condition that appears in WHERE, is applied before the formation of groups – in other words, records that do not pass the WHERE condition are eliminated **before** the formation of groups.
- Any condition that appears in HAVING refers to the groups and is applied **after** the formation of the groups. The condition must involve aggregate functions, or attributes that appear in the SELECT or GROUP-BY lines.

Query 1: Find the total number of copies in stock for each poet

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	sum
Douglas Livingstone	23
Mongane Wally	13

```
SELECT poet, SUM (copies_in_stock) as sum
FROM writer
GROUP BY poet
```

Query 2: For each poet, find the max, min, avg and total number of copies in stock

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	max	min	avg	sum
Douglas Livingstone	21	2	11.5	23
Mongane Wally	8	2	4.33	13

```
SELECT poet, MAX(copies_in_stock) AS max, MIN(copies_in_stock) AS
    min, AVG(copies_in_stock) AS avg, SUM(copies_in_stock) AS sum
FROM writer
GROUP BY poet
```

Query 3: For each poet, find the max, min, avg and total number of copies in stock – take into account only books that have > 5 copies in stock

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	max	min	avg	sum
Douglas Livingstone	21	21	21	21
Mongane Wally	8	8	8	8

```
SELECT poet, MAX(copies_in_stock) AS max, MIN(copies_in_stock) AS
    min, AVG(copies_in_stock) AS avg, SUM(copies_in_stock) AS sum
FROM writer
WHERE copies_in_stock > 5
GROUP BY poet
```

Query 4: Find the total number of copies in stock for each poet who has a total of more than 20 copies in stock

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	sum
Douglas Livingstone	23
Mongane Wally	13

```
SELECT poet, SUM(copies_in_stock) AS sum
FROM writer
GROUP BY poet
HAVING sum>20
```

Query 5: Find the total number of copies in stock for each poet who has a total of more than 20 copies in stock – take into account only books that have more than 5 copies in stock

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	sum
Douglas Livingstone	21
Mongane Wally	8

```
SELECT poet, SUM(copies_in_stock) AS sum
FROM writer
WHERE copies_in_stock > 5
GROUP BY poet
HAVING sum>20
```

Query 6: Find the total number of copies in stock for each poet whose name starts with any letter after "E"

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	sum
Mongane Wally	13

```
SELECT poet, SUM(copies_in_stock)
    as sum
FROM writer
WHERE poet > "E"
GROUP BY poet
```

```
SELECT poet, SUM(copies_in_stock)
    as sum
FROM writer
GROUP BY poet
HAVING poet > "E"
```

Query 7: Find the total number of copies in stock for each poet who has more than 2 books

poet	book	copies_in_stock
Douglas Livingstone	The Skull	21
Douglas Livingstone	A Littoral Zone	2
Mongane Wally	Tstetlo	3
Mongane Wally	Must Weep	8
Mongane Wally	A Tough Tale	2

poet	sum
Mongane Wally	13

```
SELECT poet, SUM(copies_in_stock)
    as sum
FROM writer
GROUP BY poet
HAVING count(*)>2
```

DATA WAREHOUSES and OLAP

- *On-Line Transaction Processing* (OLTP) Systems manipulate operational data, necessary for day-to-day operations. Most existing database systems belong this category.
- *On-Line Analytical Processing* (OLAP) Systems support specific types of queries (based on group-bys and aggregation operators) useful for decision making.
- *Data Mining* tools discover interesting patterns in the data

Why OLTP is not sufficient for Decision Making

Lets say that Welcome supermarket uses a relational database to keep track of sales in all of stores simultaneously

SALES table			
product id	**store id**	**quantity sold**	**date/time of sale**
567	17	1	1997-10-22 09:35:14
219	16	4	1997-10-22 09:35:14
219	17	1	1997-10-22 09:35:17
...			

Example (cont.)

PRODUCTS table

Prod. id	product name	product category	Manufact. id
567	Colgate Gel Pump 6.4 oz.	toothpaste	68
219	Diet Coke 12 oz. can	soda	5
...			

STORES table

store id	city id	store location	phone number
16	34	510 Main Street	415-555-1212
17	58	13 Maple Avenue	914-555-1212

CITIES table

id	name	state	Popul.
34	San Francisco	California	700,000
58	East Fishkill	New York	30,000

Example (cont.)

- An executive, asks "I noticed that there was a Colgate promotion recently, directed at people who live in small towns. How much Colgate toothpaste did we sell in those towns yesterday? And how much on the same day a month ago?"

```
select sum(sales.quantity_sold)
from sales, products, stores, cities
where products.manufacturer_id = 68   -- restrict to Colgate-
and products.product_category = 'toothpaste '
and cities.population < 40000
and sales.datetime_of_sale::date = 'yesterday'::date
and sales.product_id = products.product_id
and sales.store_id = stores.store_id
and stores.city_id = cities.city_id
```

Example (cont.)

- You have to do a 4-way JOIN of some large tables. Moreover, these tables are being updated as the query is executed.
- Need for a separate RDBMS installation (i.e., a Data Warehouse) to support queries like the previous one.
- The Warehouse can be tailor-made for specific types of queries: if you know that the toothpaste query will occur every day then you can denormalize the data model.

Example (cont.)

- Suppose Welcome acquires ParknShop which is using a different set of OLTP data models and a different brand of RDBMS to support them. But you want to run the toothpaste queries for both divisions.
- Solution: Also copy data from the ParknShop Database into the Welcome Data Warehouse (data integration problems).
- One of the more important functions of a data warehouse in a company that has disparate computing systems is to provide a view for management as though the company were in fact integrated.

Motivation

- In most organizations, data about specific parts of business is there -- lots and lots of data, somewhere, in some form.
- Data is available but *not information -- and not the right information at the right time.*
- To bring together information from multiple sources as to provide a consistent database source for decision support queries.
- To off-load decision support applications from the on-line transaction system.

What is a Data Warehouse?

- Defined in many different ways, but not rigorously.
  - A decision support database that is maintained separately from the organization's operational database
  - Support information processing by providing a solid platform of consolidated, historical data for analysis.
- "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."—W. H. Inmon
- Data warehousing:
  - The process of constructing and using data warehouses

Data Warehouse—Subject-Oriented

- Organized around major subjects, such as customer, product, sales
- Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing
- Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process

Data Warehouse—Integrated

- Constructed by integrating multiple, heterogeneous data sources
  - relational databases, flat files, on-line transaction records
- Data cleaning and data integration techniques are applied.
  - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources
    - E.g., Hotel price: currency, tax, breakfast covered, etc.
  - When data is moved to the warehouse, it is converted.

Data Warehouse—Time Variant

- The time horizon for the data warehouse is significantly longer than that of operational systems
  - Operational database: current value data
  - Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)
- Every key structure in the data warehouse
  - Contains an element of time, explicitly or implicitly
  - But the key of operational data may or may not contain "time element"

Data Warehouse—Nonvolatile

- A physically separate store of data transformed from the operational environment
- Operational update of data does not occur in the data warehouse environment
  - Does not require transaction processing, recovery, and concurrency control mechanisms
  - Requires only two operations in data accessing:
    - *initial loading of data* and *access of data*

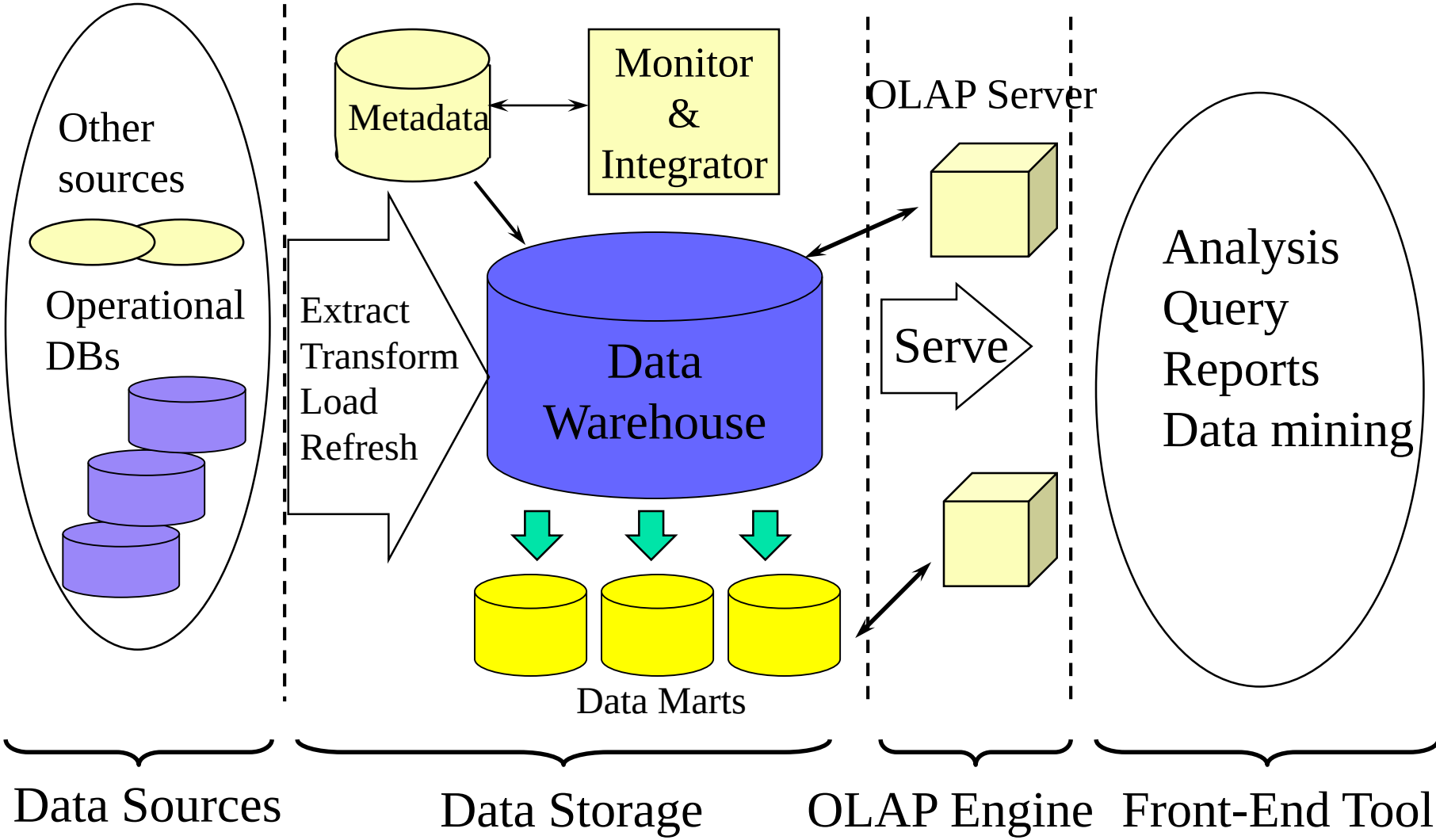
OLTP vs. OLAP

	OLTP	OLAP
users	clerk, IT professional	knowledge worker
function	day to day operations	decision support
DB design	application-oriented	subject-oriented
data	current, up-to-date detailed, flat relational isolated	historical, summarized, multidimensional integrated, consolidated
usage	repetitive	ad-hoc
access	read/write index/hash on prim. key	lots of scans
unit of work	short, simple transaction	complex query
# records accessed	tens	millions
#users	thousands	hundreds
DB size	100MB-GB	100GB-TB
metric	transaction throughput	query throughput, response

Why a Separate Data Warehouse?

- High performance for both systems
  - DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery
  - Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation
- Different functions and different data:
  - missing data: Decision support requires historical data which operational DBs do not typically maintain
  - data consolidation: DS requires consolidation (aggregation, summarization) of data from heterogeneous sources
  - data quality: different sources typically use inconsistent data representations, codes and formats which have to be reconciled
- Note: There are more and more systems which perform OLAP analysis directly on relational databases

Data Warehouse: A Multi-Tiered Architecture



Chapter 4: Data Warehousing and On-line Analytical Processing

- Data Warehouse: Basic Concepts
- Data Warehouse Modeling: Data Cube and OLAP
- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Summary

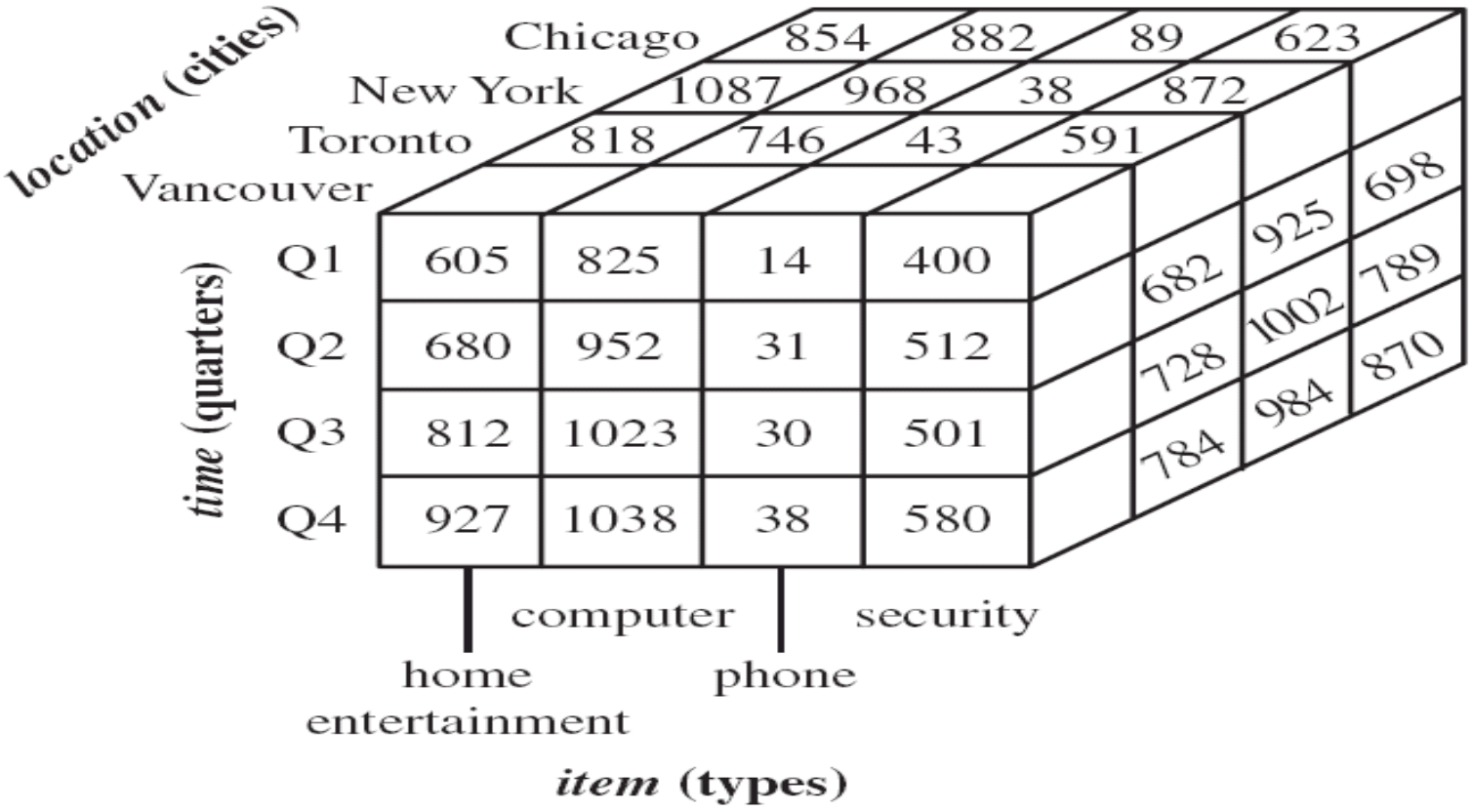
From Tables and Spreadsheets to Data Cubes

- A **data warehouse** is based on a multidimensional data model which views data in the form of a data cube
- 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
- In data warehousing literature, an n-D base cube is called a base cuboid. The top most 0-D cuboid, which holds the highest-level of summarization, is called the apex cuboid. The lattice of cuboids forms a data cube.

Data cube

- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
- Suppose ALLELETRONICS create a *sales* data warehouse with respect to dimensions
  - Time
  - Item
  - Location

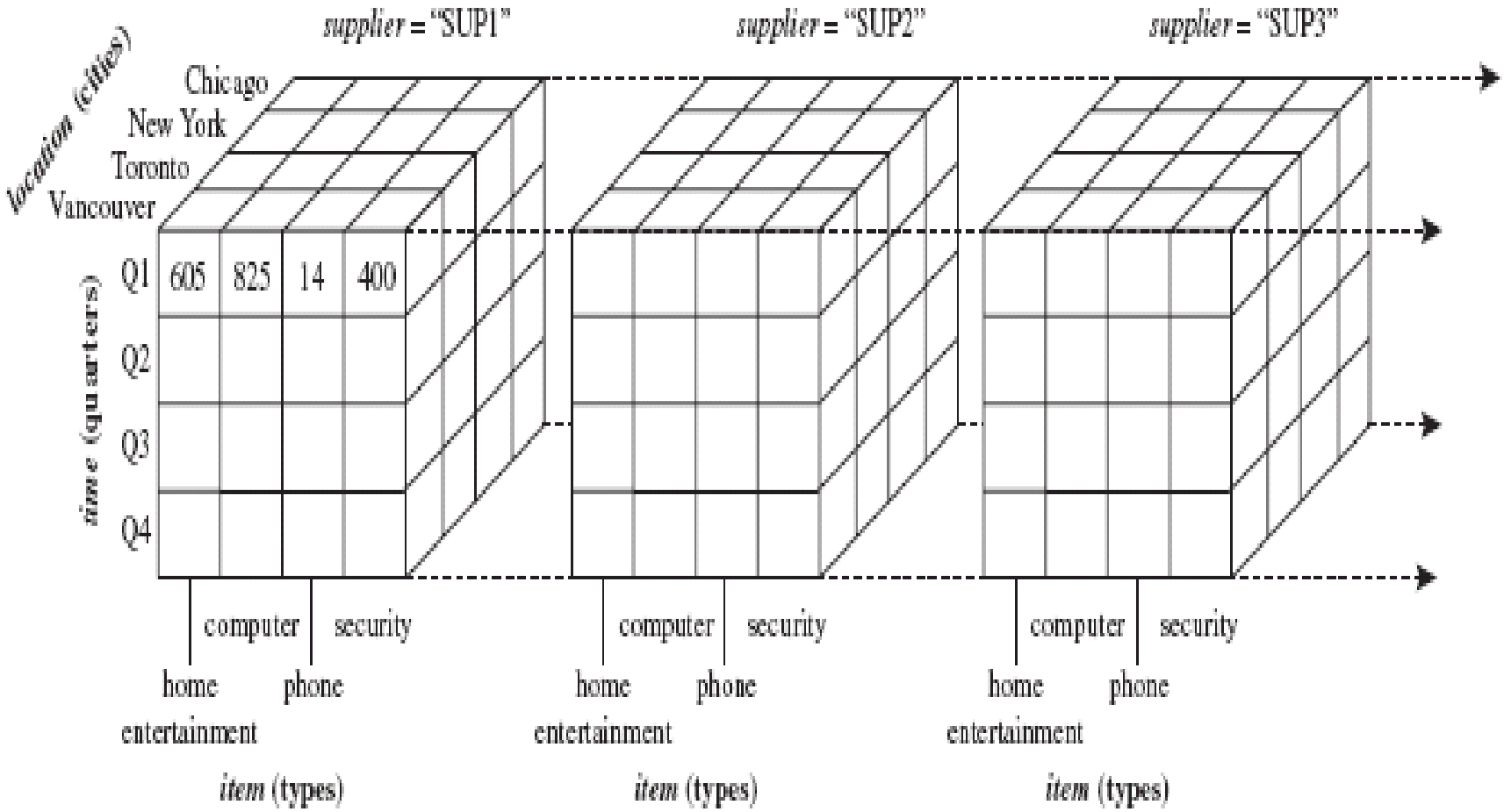
3D Data cube Example



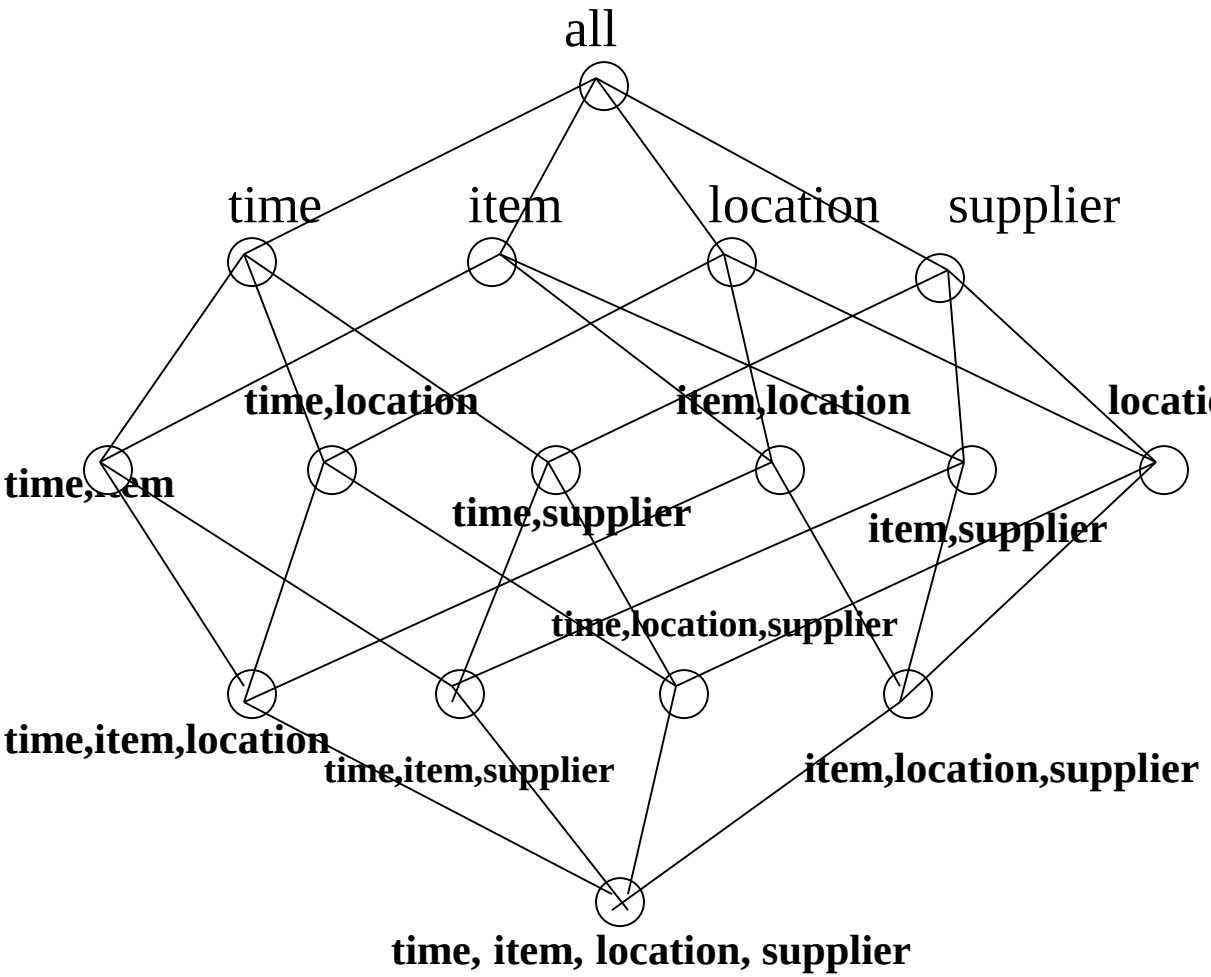
Data cube

- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
- Suppose ALLELETRONICS create a *sales* data warehouse with respect to dimensions
  - Time
  - Item
  - Location
  - Supplier

4D Data cube Example



Cube: A Lattice of Cuboids



0-D (*apex*) cuboid

1-D cuboids

2-D cuboids

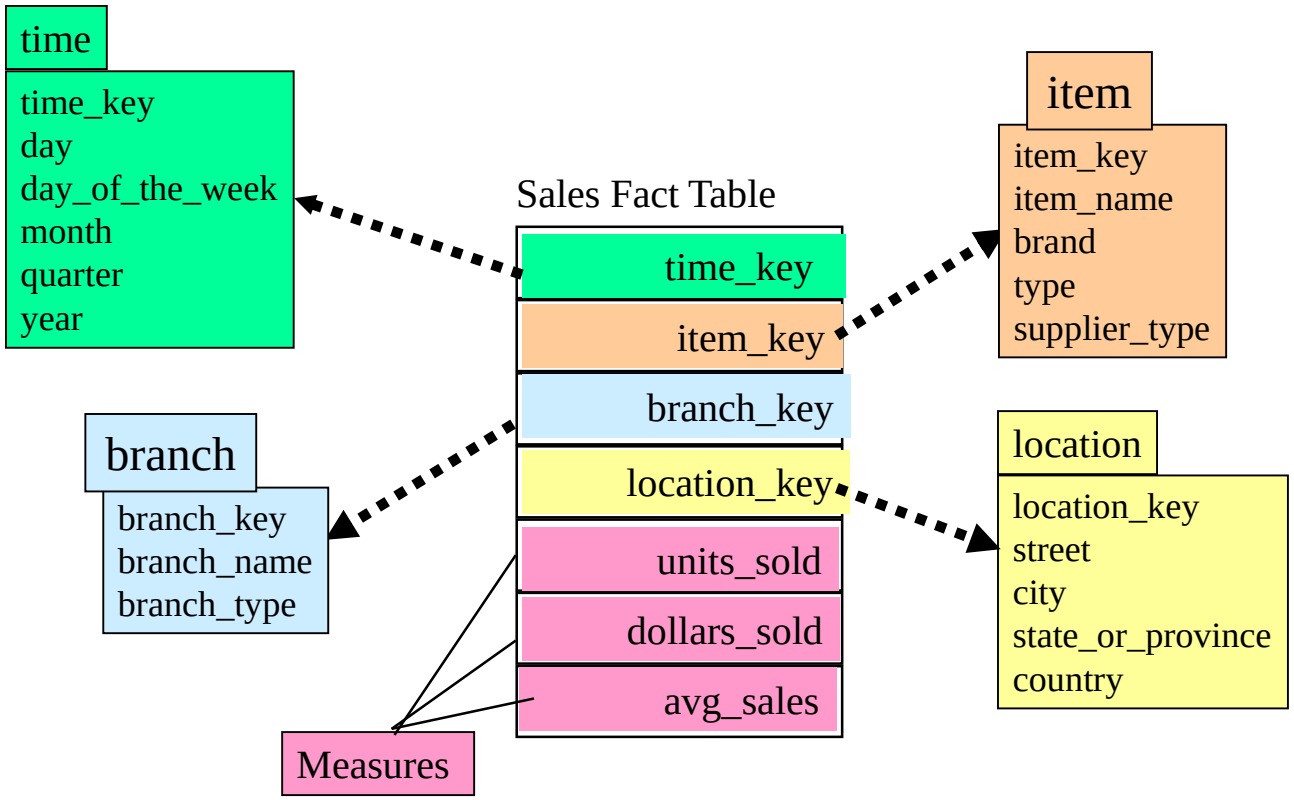
3-D cuboids

4-D (*base*) cuboid

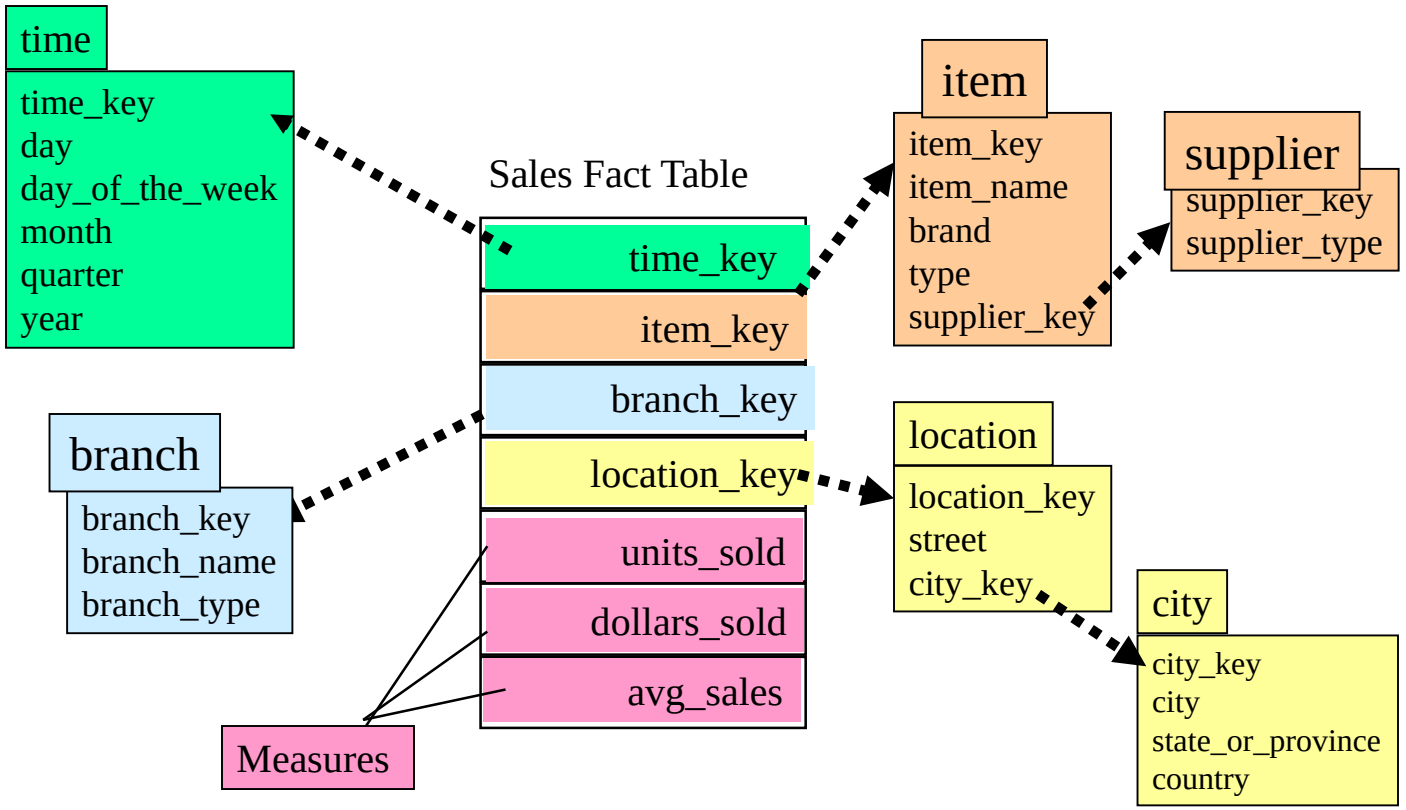
Conceptual Modeling of Data Warehouses

- Modeling data warehouses: dimensions & measures
  - Star schema: A fact table in the middle connected to a set of dimension tables
  - Snowflake schema: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
  - Fact constellations: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation

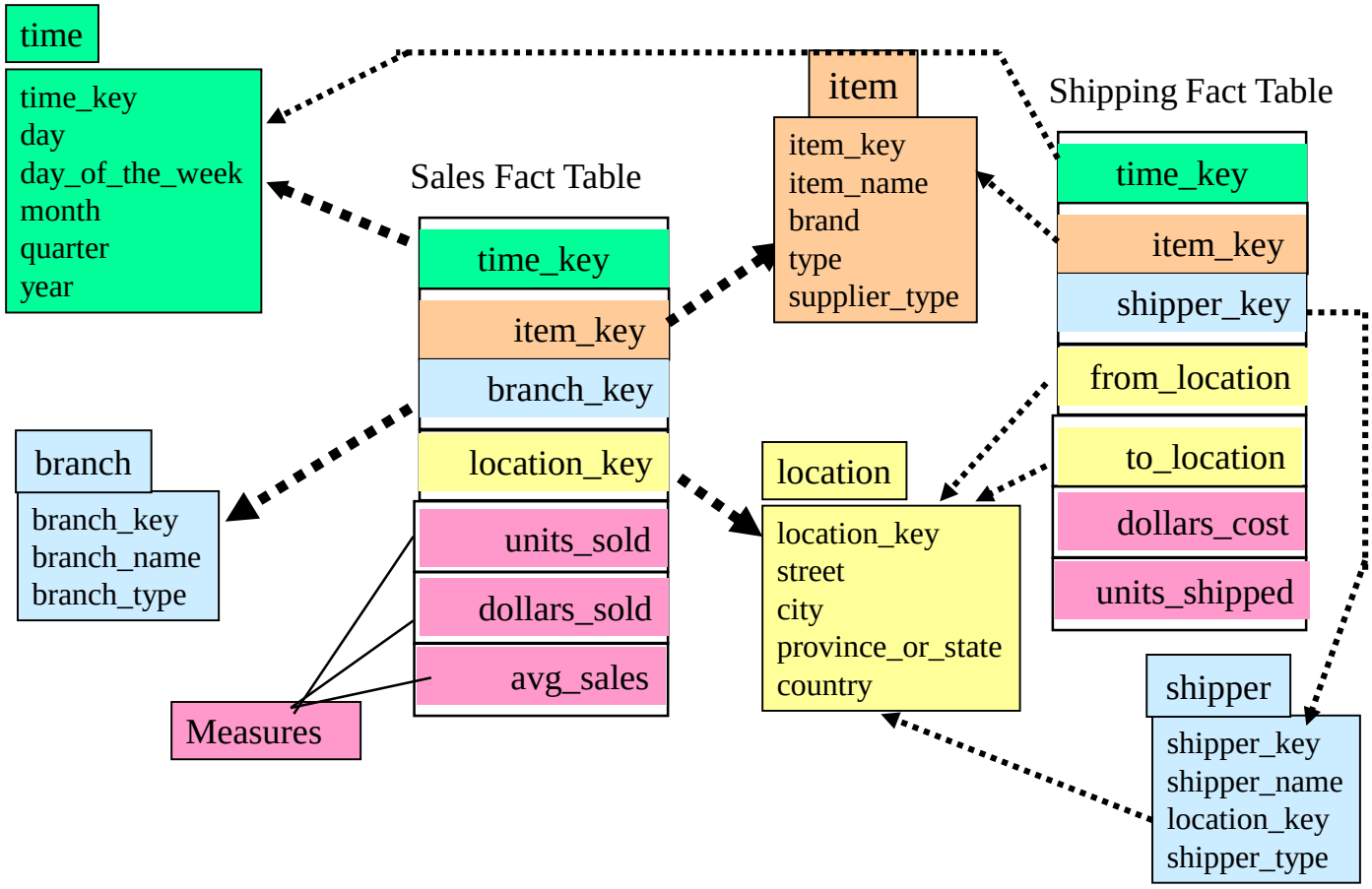
Star Schema: An Example



Snowflake Schema: An Example



Fact Constellation: An Example

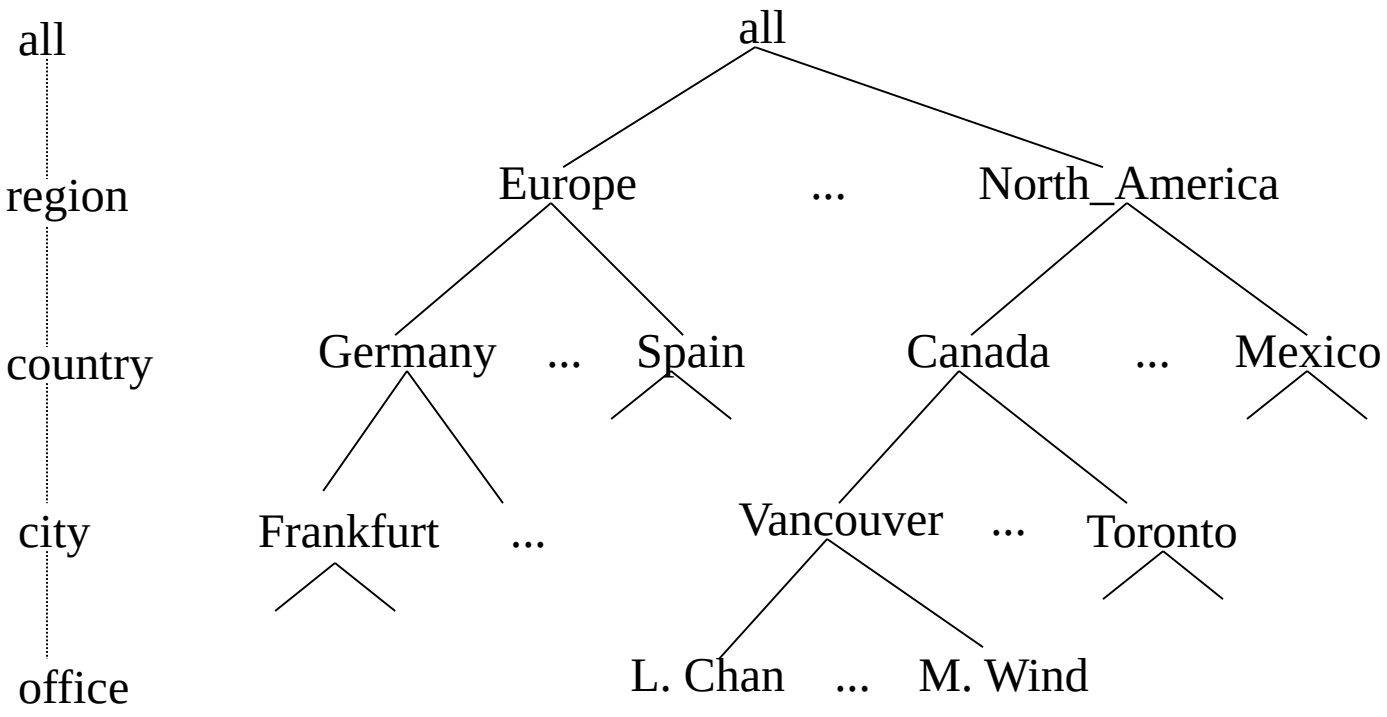


Concept Hierarchies

- A **Concept Hierarchy** defines a sequence of mappings from a set of low-level concepts to high-level

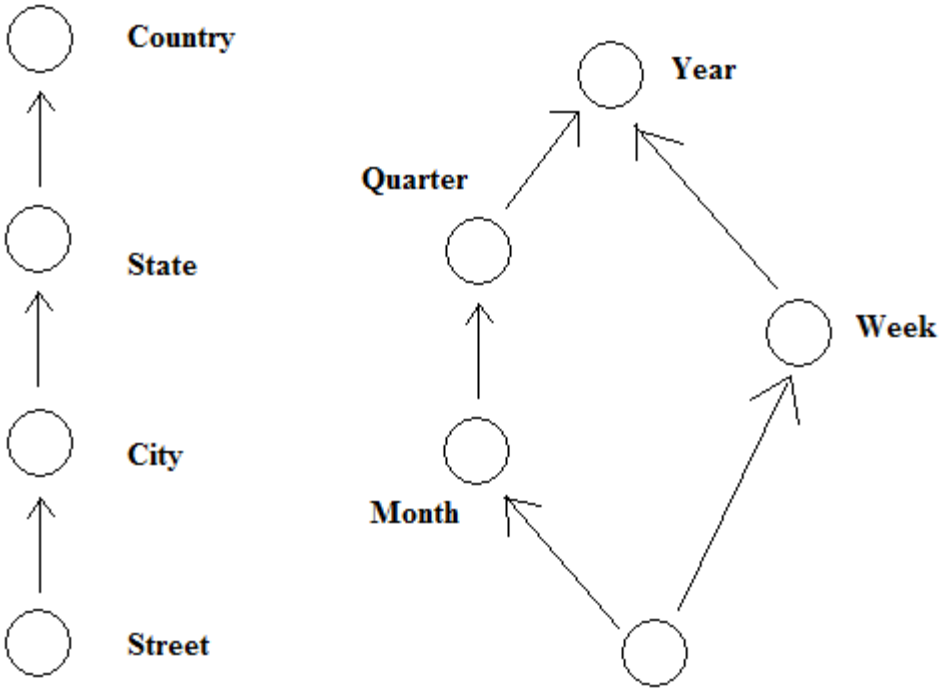
- Consider a concept hierarchy for the dimension **"Location"**

A Concept Hierarchy for a Dimension (location)



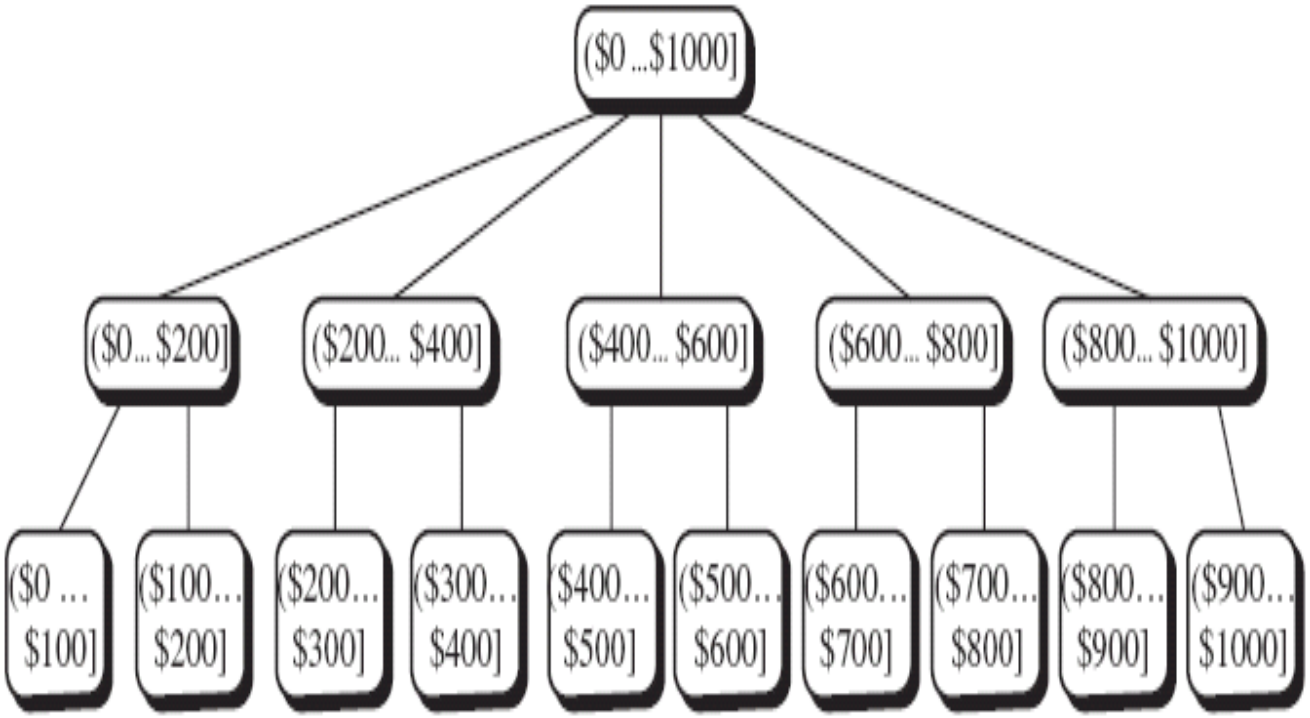
Concept Hierarchies

- Many concept hierarchies are implicit within the database system



Concept Hierarchies

- Concept hierarchies may also be defined by grouping values for a given dimension or attribute, resulting in a **set-grouping hierarchy**



OLAP Operation

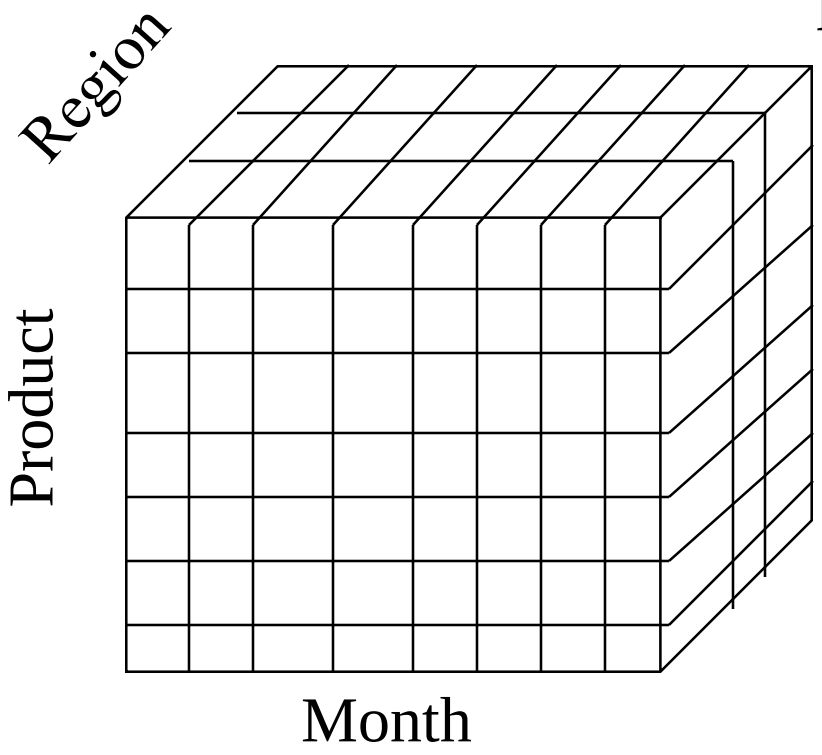
- So, how are *concept hierarchies* useful in OLAP?
- In the multidimensional model, data are organized into multiple dimensions,
- And each dimension contains multiple levels of abstraction defined by concept hierarchies

Data Cube Measures: Three Categories

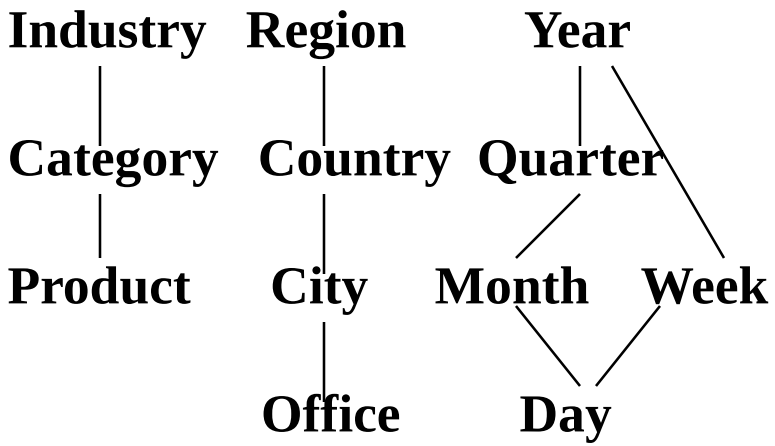
- Distributive: if the result derived by applying the function to *n* aggregate values is the same as that derived by applying the function on all the data without partitioning
  - E.g., count(), sum(), min(), max()
- Algebraic: if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded integer), each of which is obtained by applying a distributive aggregate function
  - E.g., avg(), min_N(), standard_deviation()
- Holistic: if there is no constant bound on the storage size needed to describe a subaggregate.
  - E.g., median(), mode(), rank()

Multidimensional Data

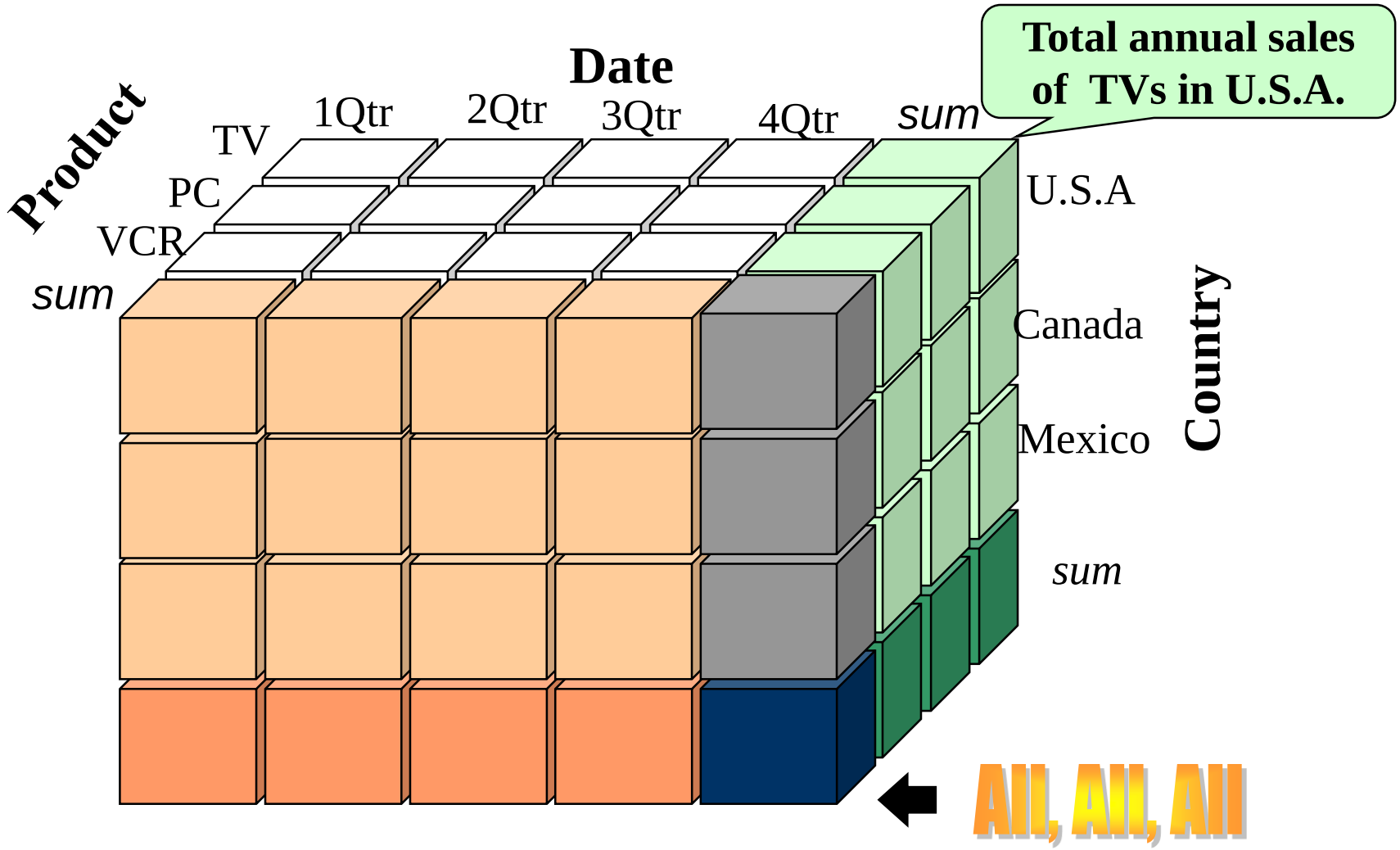
- Sales volume as a function of product, month, and region



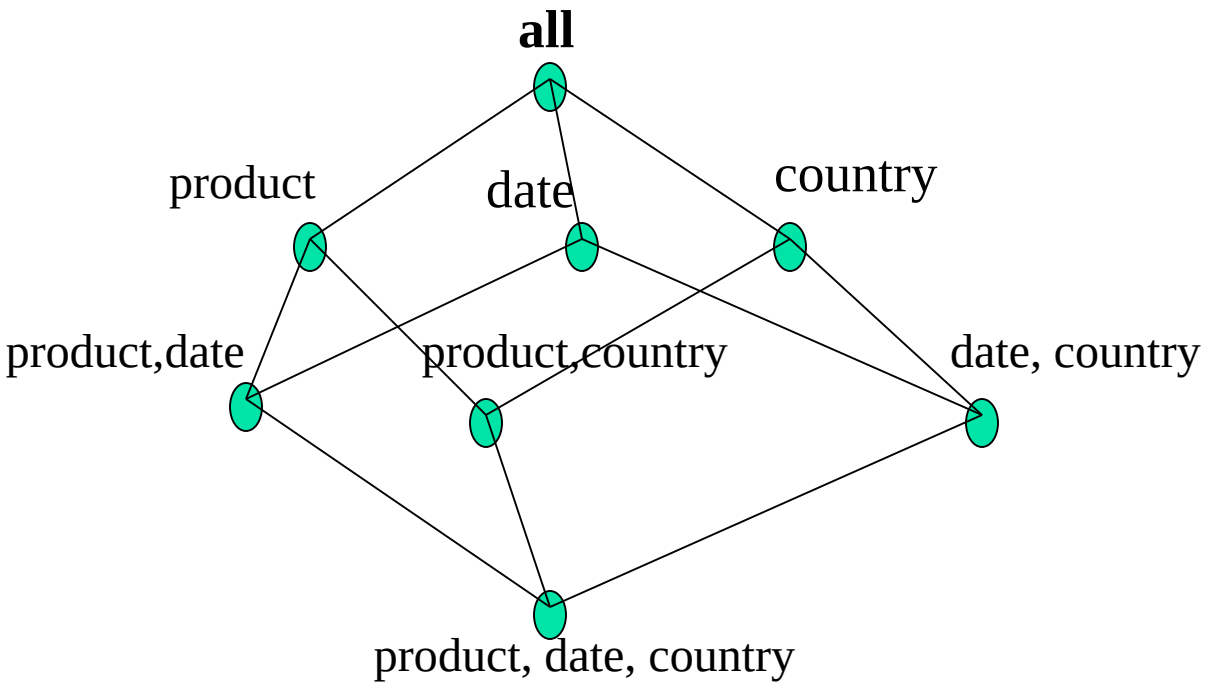
Dimensions: *Product, Location, Time*
Hierarchical summarization paths



A Sample Data Cube



Cuboids Corresponding to the Cube



0-D (*apex*) cuboid

1-D cuboids

2-D cuboids

3-D (*base*) cuboid

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*
- 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 (using SQL)*

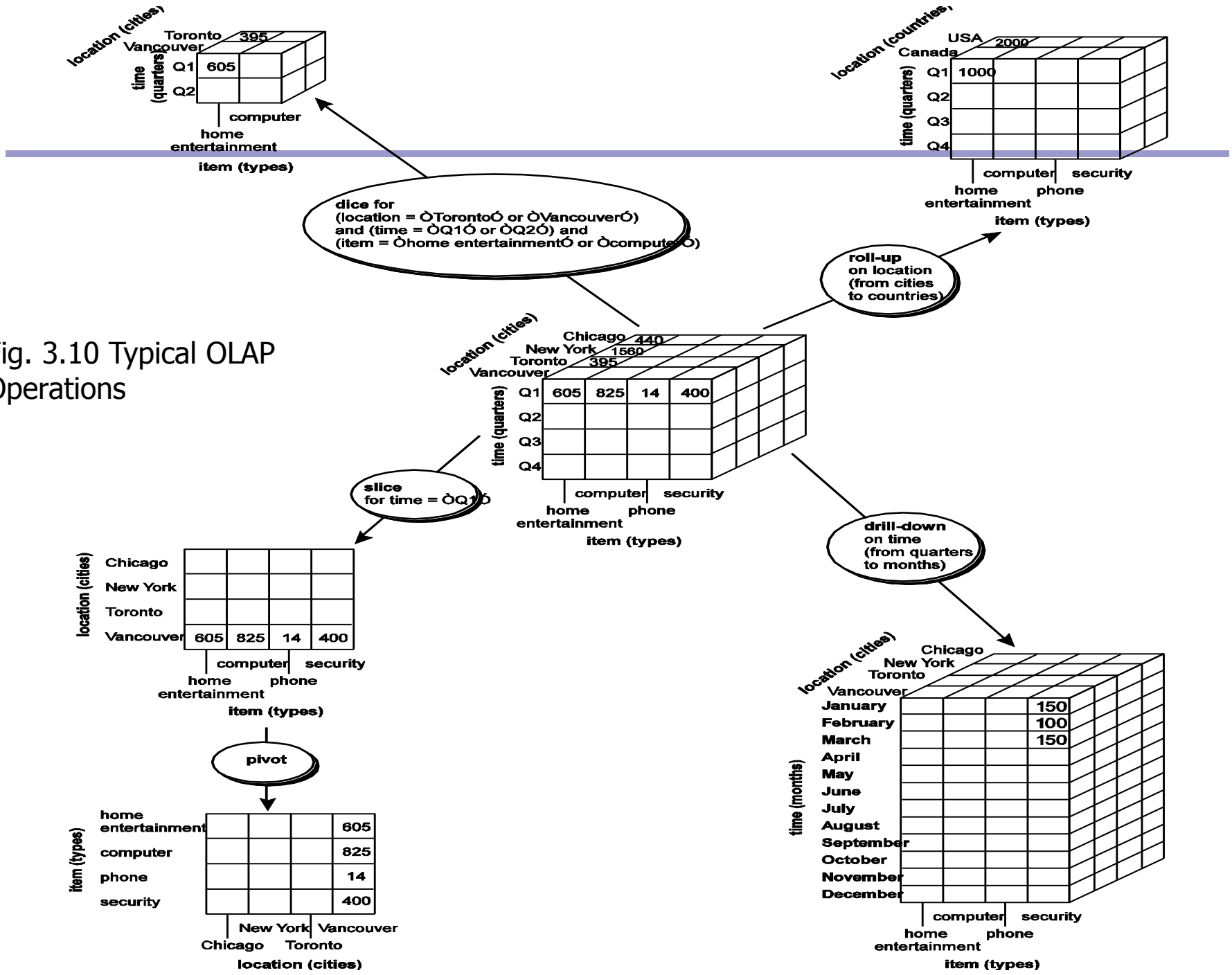
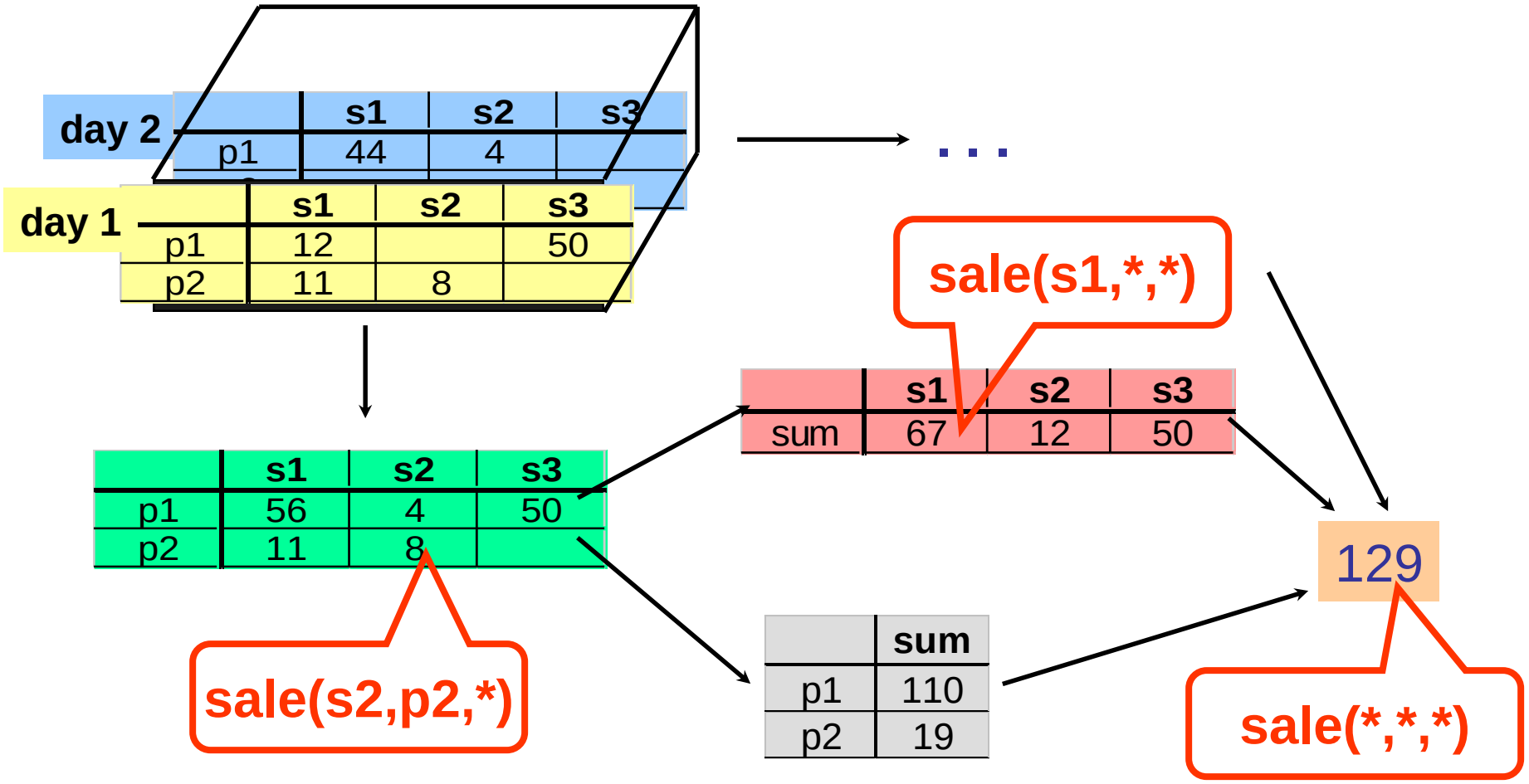
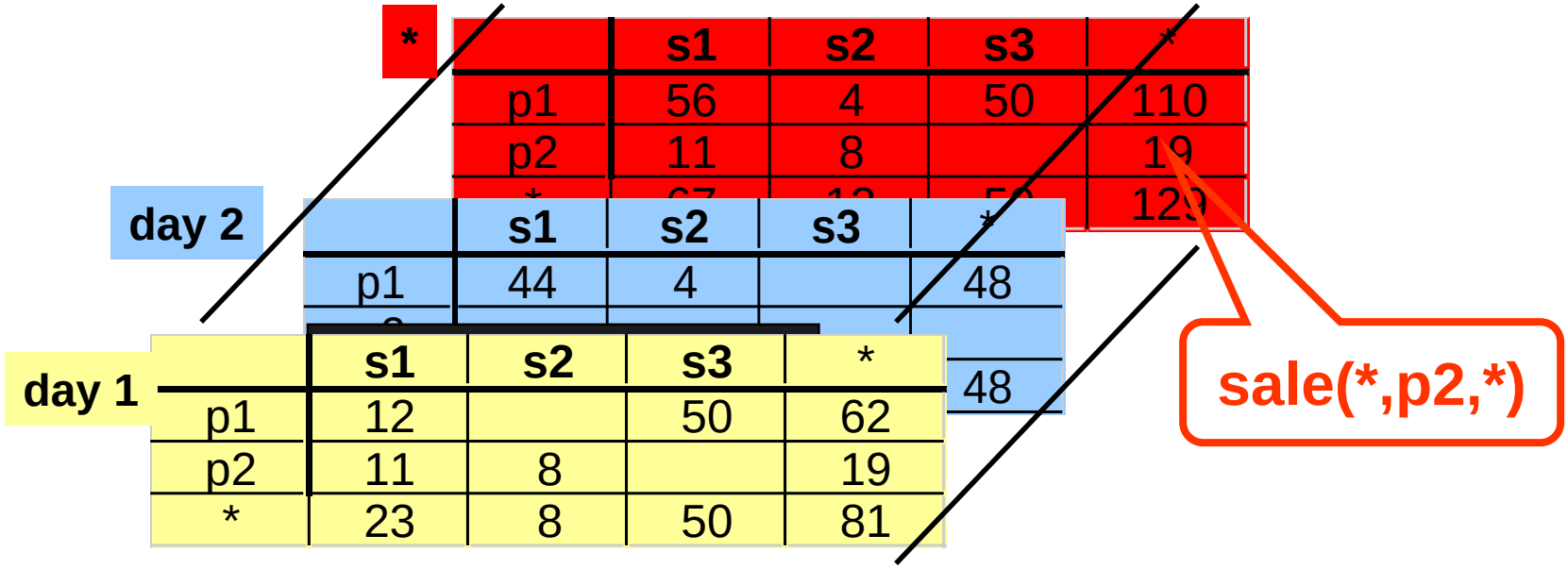


Fig. 3.10 Typical OLAP Operations

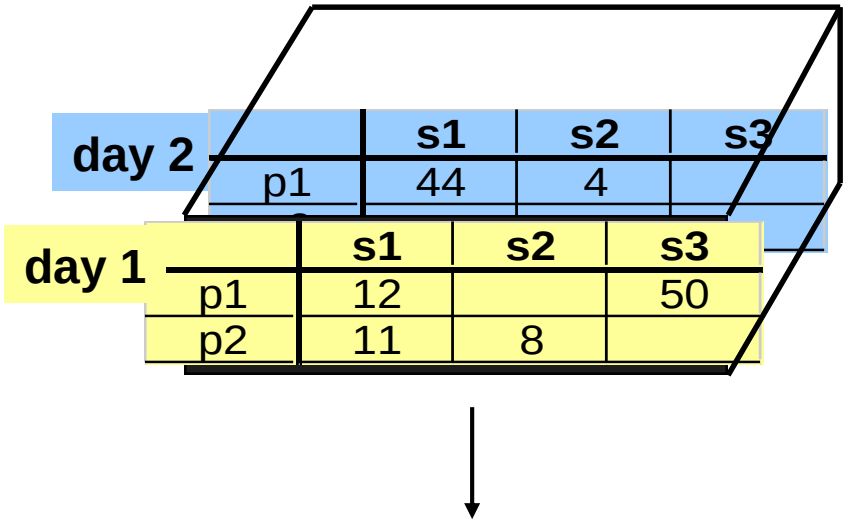
Cube Operators for Roll-up



Extended Cube



Aggregation Using Hierarchies



	region A	region B
p1	56	54
p2	11	8

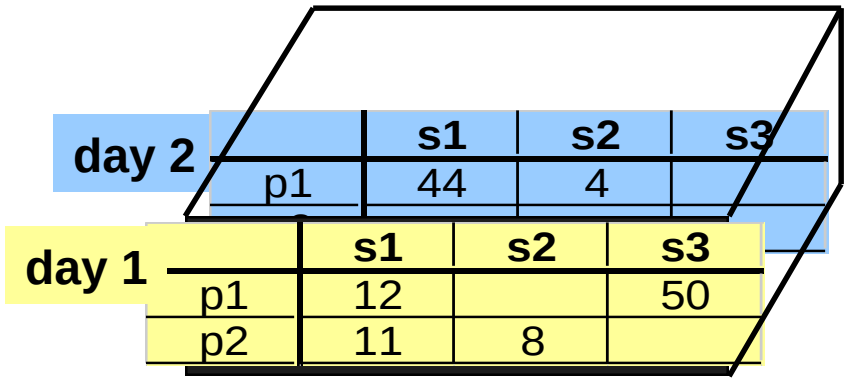
store

region

country

(store s1 in Region A;
stores s2, s3 in Region B)

Slicing



TIME = day 1

	s1	s2	s3
p1	12		50
p2	11	8	

Slicing & Pivoting

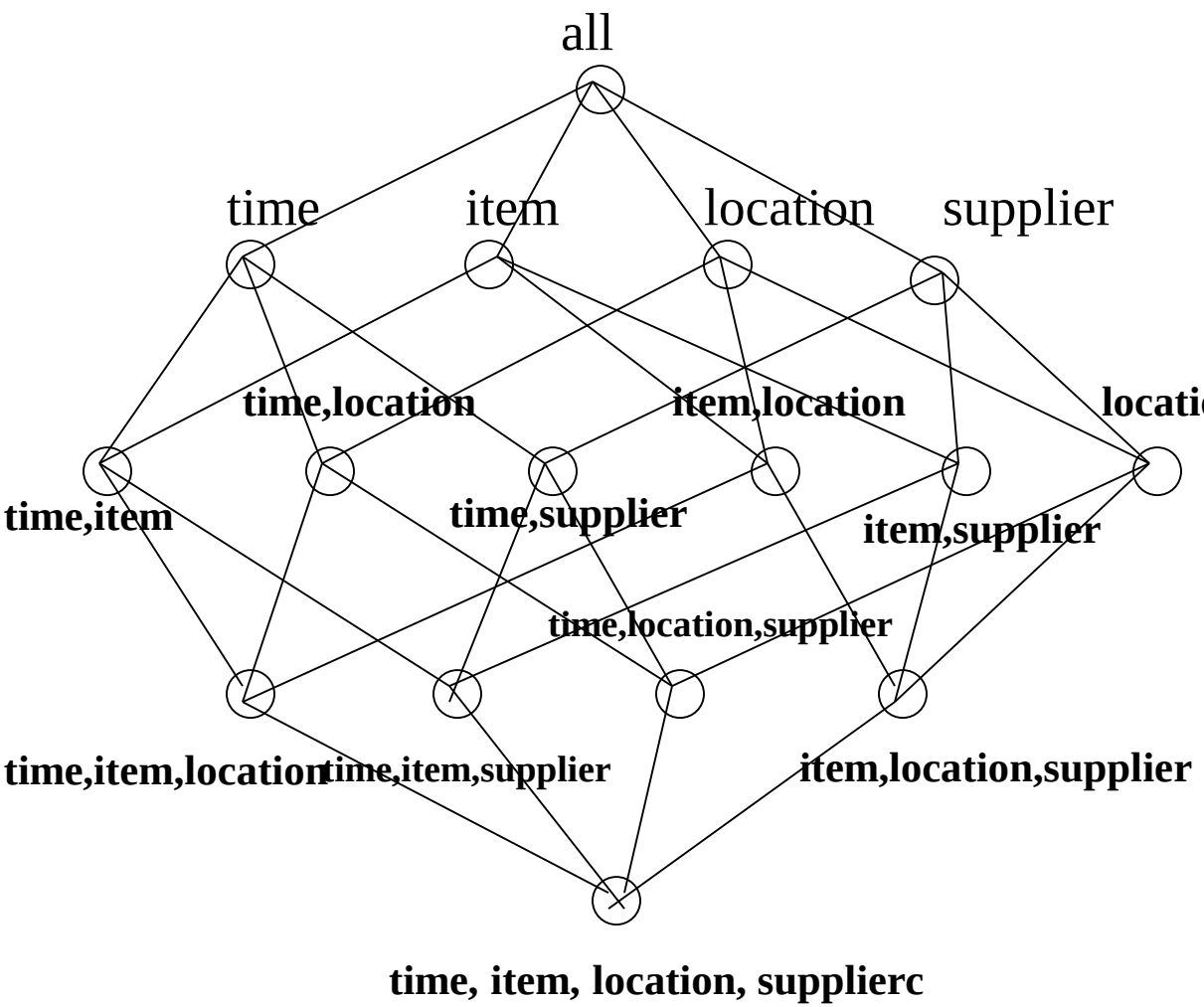
		Sales ($ millions)		
	Products	Time		
		d1	d2	
Store s1	Electronics	$5.2		
	Toys	$1.9		
	Clothing	$2.3		
	Cosmetics	$1.1		
Store s2	Electronics	$8.9		
	Toys	$0.75		
	Clothing	$4.6		
	Cosmetics	$1.5		

		Sales ($ millions)		
	Products	d1		
		Store s1	Store s2	
Store s1	Electronics	$5.2	$8.9	
	Toys	$1.9	$0.75	
	Clothing	$2.3	$4.6	
	Cosmetics	$1.1	$1.5	
Store s2	Electronics			
	Toys			
	Clothing			

Chapter 5: Data Cube Technology

- Data Warehouse: Basic Concepts
- Data Warehouse Modeling: Data Cube and OLAP
- Data Cube Computation: Preliminary Concepts 
- Data Cube Computation Methods

Data Cube: A Lattice of Cuboids



0-D(apex) cuboid

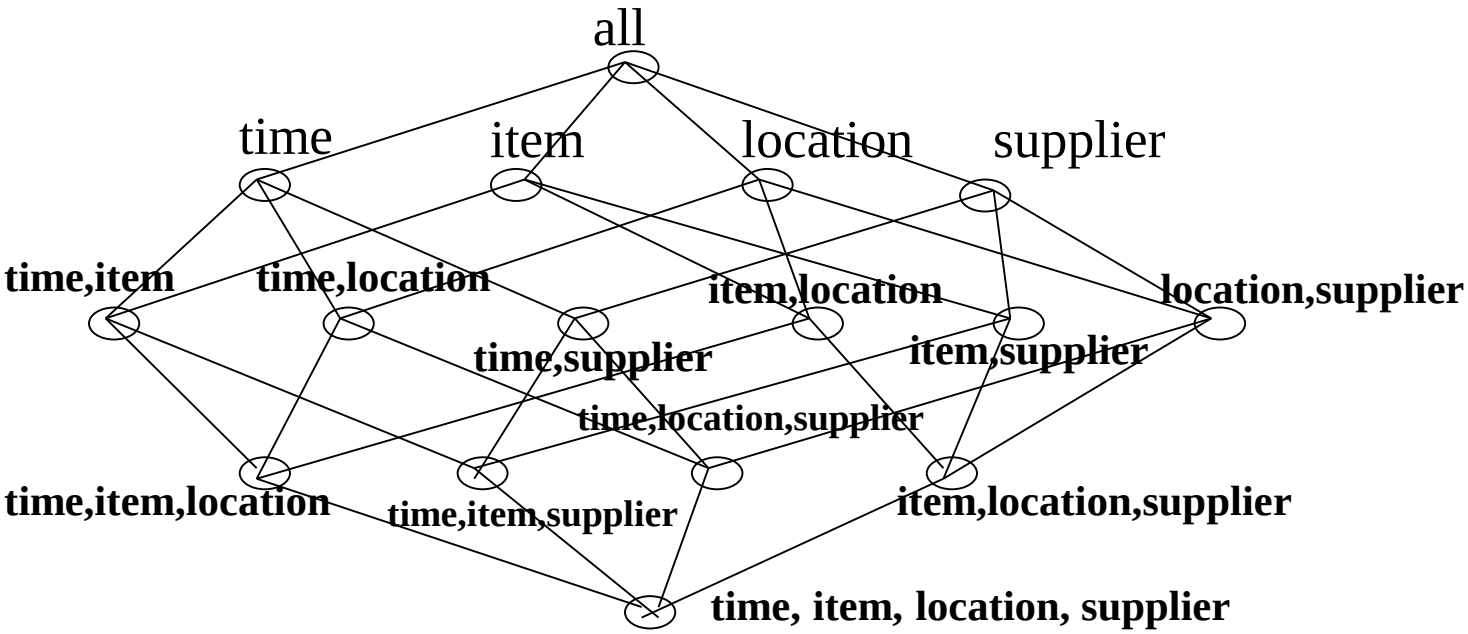
1-D cuboids

2-D cuboids

3-D cuboids

4-D(base) cuboid

Data Cube: A Lattice of Cuboids



0-D(apex) cuboid

1-D cuboids

2-D cuboids

3-D cuboids

4-D(base) cuboid

- Base vs. aggregate cells; ancestor vs. descendant cells; parent vs. child cells
  1. (9/15, milk, Urbana, Dairy_land)
  2. (9/15, milk, Urbana, *)
  3. (*, milk, Urbana, *)
  4. (*, milk, Urbana, *)
  5. (*, milk, Chicago, *)
  6. (*, milk, *, *)

Cube Materialization: Full Cube vs. Iceberg Cube

- Full cube vs. iceberg cube

```
compute cube sales iceberg as
select month, city, customer group, count(*)
from salesInfo
cube by month, city, customer group
having count(*) >= min support
```

iceberg condition

- Compute *only* the cuboid cells whose measure satisfies the iceberg condition
- Only a small portion of cells may be "above the water'' in a sparse cube
- Ex.: Show only those cells whose count is no less than 100

Why Iceberg Cube?

- Advantages of computing iceberg cubes
  - No need to save nor show those cells whose value is below the threshold (iceberg condition)
  - Efficient methods may even avoid computing the un-needed, intermediate cells
  - Avoid explosive growth
- Example: A cube with 100 dimensions
  - Suppose it contains only 2 base cells: $\{(a_1, a_2, a_3, \ldots, a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$
  - How many aggregate cells if "having count >= 1"?
    - Answer: $2^{101} - 4$
  - What about the iceberg cells, (i,e., with condition: "having count >= 2")?
    - Answer: 4

Is Iceberg Cube Good Enough? Closed Cube & Cube Shell

- Let cube P have only 2 base cells: {(a_1, a_2, a_3 . . . , a_{100}):10, (a_1, a_2, b_3, . . . , b_{100}):10}
  - How many cells will the iceberg cube contain if "having count(*) ≥ 10"?
    - Answer: $2^{101} - 4$ (still too big!)
- **Close cube:**
  - A cell c is ***closed*** if there exists no cell *d*, such that *d* is a descendant of *c*, and *d* has the same measure value as *c*
    - Ex. The same cube P has only 3 closed cells:
  - {(a_1, a_2, *, …, *): 20, (a_1, a_2, a_3 . . . , a_{100}): 10, (a_1, a_2, b_3, . . . , b_{100}): 10}
  - A ***closed cube*** is a cube consisting of only closed cells
- **Cube Shell:** The cuboids involving only a small # of dimensions, e.g., 2
  - Idea: Only compute cube shells, other dimension combinations can be computed on the fly

Q: For (A_1, A_2, … A_{100}), how many combinations to compute?

General Heuristics (Agarwal et al. VLDB'96)

- Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples
- Aggregates may be computed from previously computed aggregates, rather than from the base fact table
  - **Smallest-child:** computing a cuboid from the smallest, previously computed cuboid
  - **Cache-results:** caching results of a cuboid from which other cuboids are computed to reduce disk I/Os
  - **Amortize-scans:** computing as many as possible cuboids at the same time to amortize disk reads
  - **Share-sorts:** sharing sorting costs cross multiple cuboids when sort-based method is used
  - **Share-partitions:** sharing the partitioning cost across multiple cuboids when hash-based algorithms are used

Chapter 5: Data Cube Technology

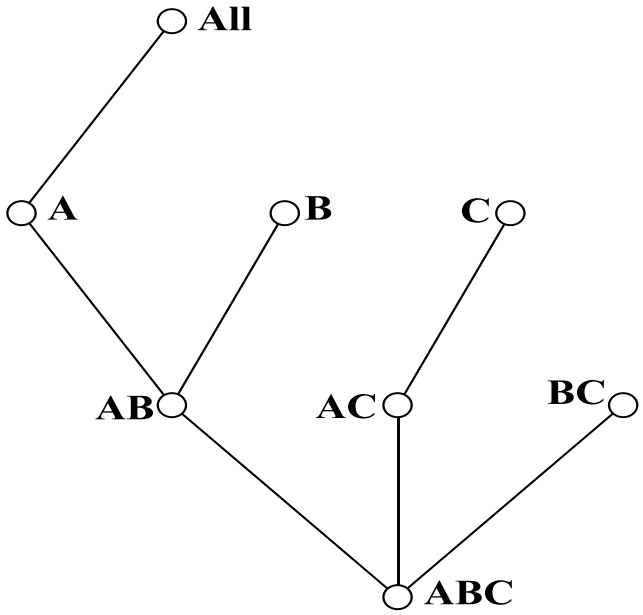
- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods 
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Data Cube Computation Methods

- Multi-Way Array Aggregation 
- BUC

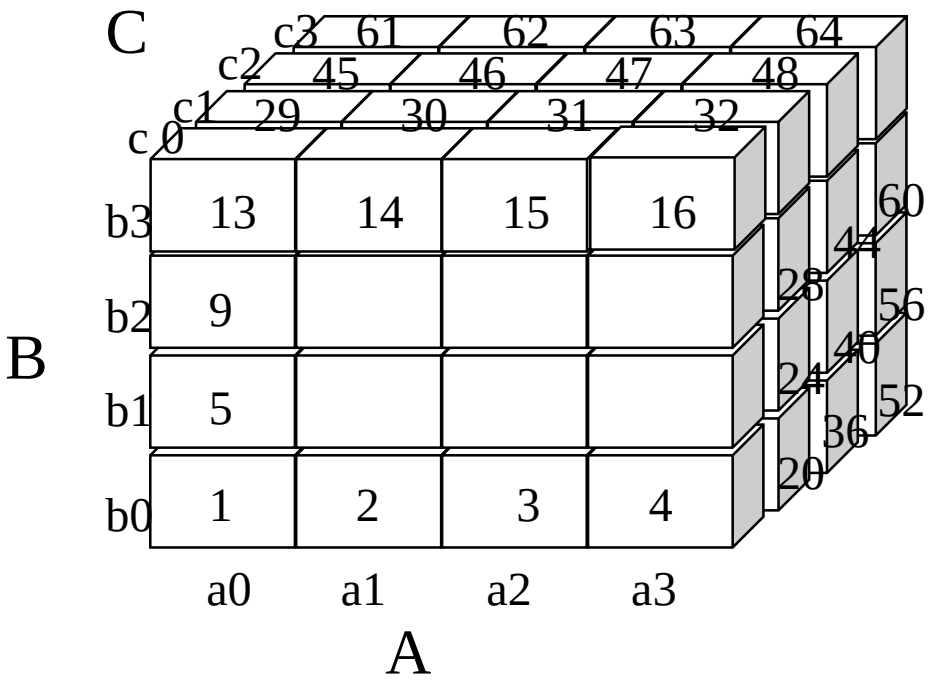
Multi-Way Array Aggregation

- Array-based "bottom-up" algorithm
- Using multi-dimensional chunks
- No direct tuple comparisons
- Simultaneous aggregation on multiple dimensions
- Intermediate aggregate values are re-used for computing ancestor cuboids
- Cannot do *Apriori* pruning: No iceberg optimization



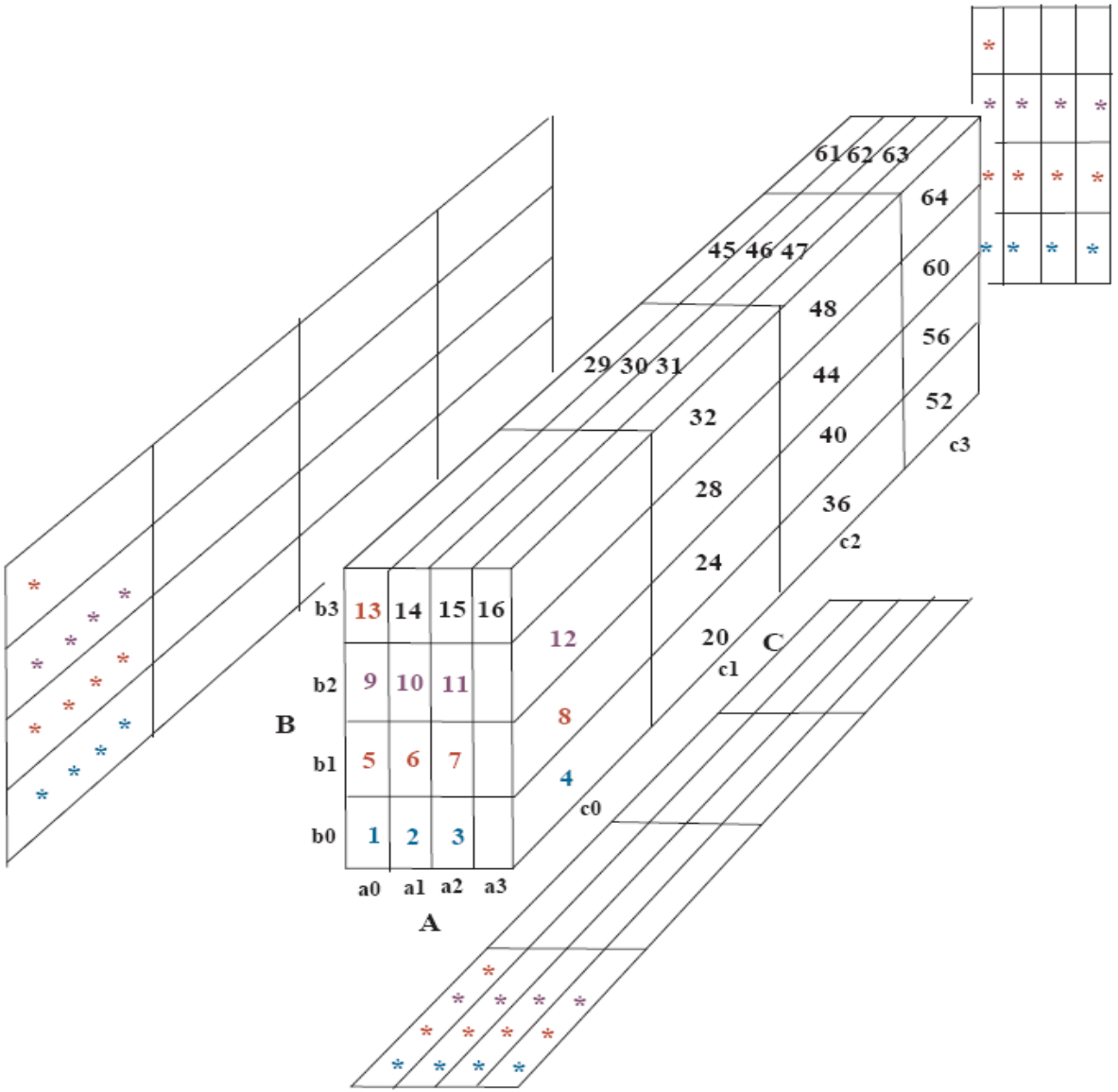
Multi-way Array Aggregation for Cube Computation (MOLAP)

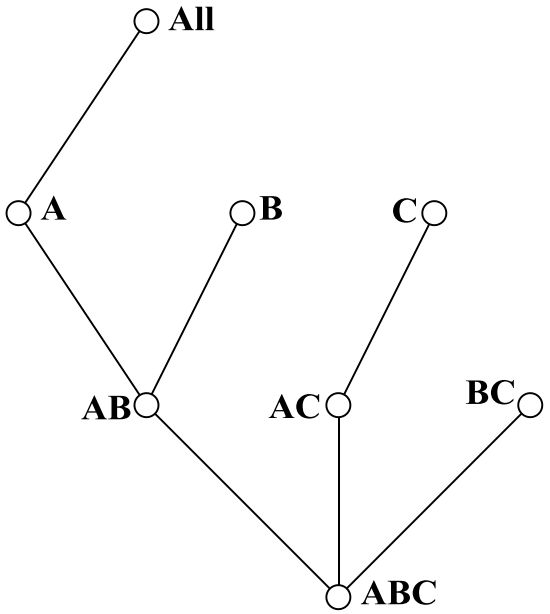
- Partition arrays into chunks (a small subcube which fits in memory).
- Compressed sparse array addressing: (chunk_id, offset)
- Compute aggregates in "multiway" by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces memory access and storage cost.



What is the best traversing order to do multi-way aggregation?

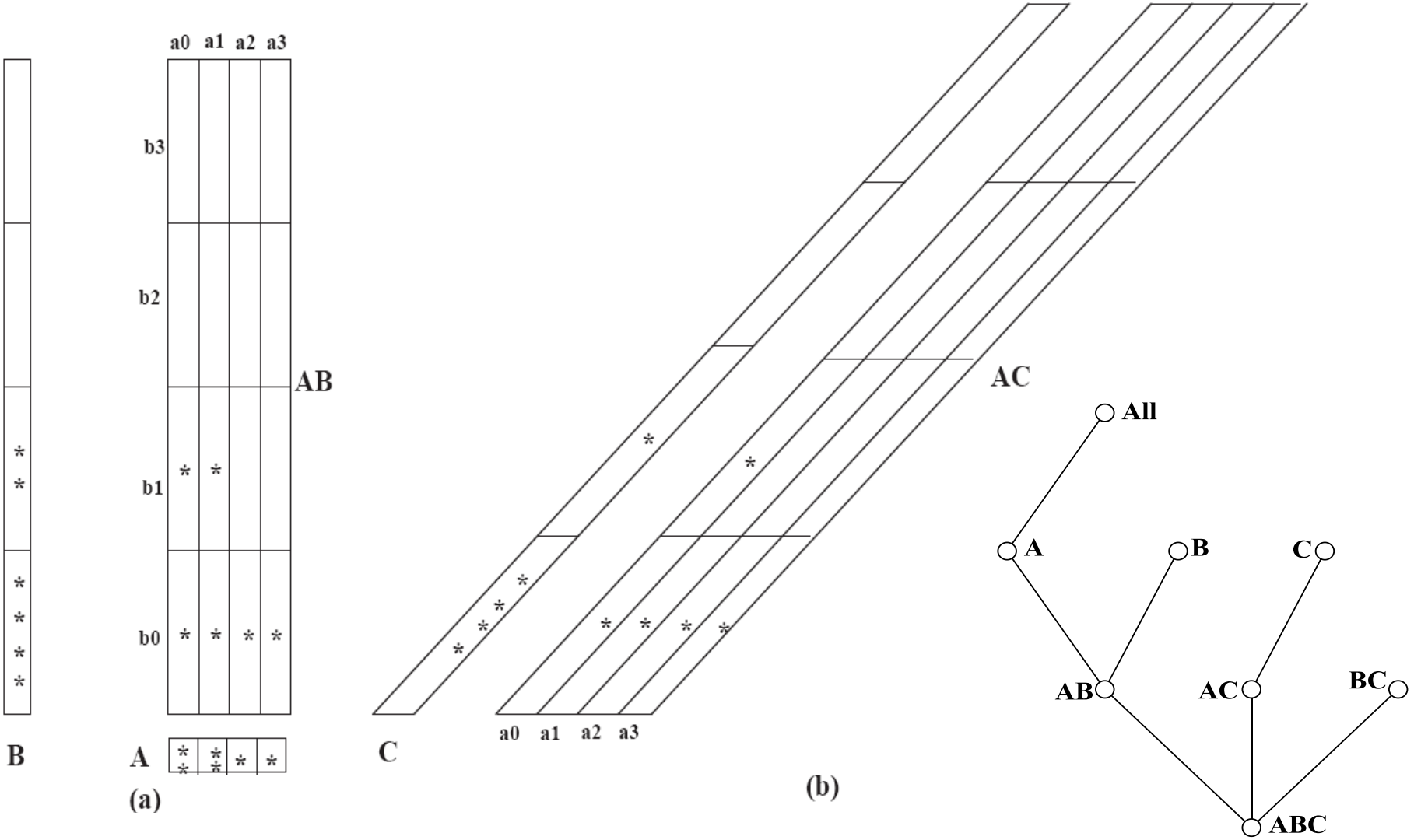
Multi-way Array Aggregation for Cube Computation (3-D to 2-D)





- The best order is the one that minimizes the memory requirement and reduced I/Os

Multi-way Array Aggregation for Cube Computation (2-D to 1-D)



Multi-Way Array Aggregation for Cube Computation (Method Summary)

- Method: the planes should be sorted and computed according to their size in ascending order
  - Idea: keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane
- Limitation of the method: computing well only for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods can be explored

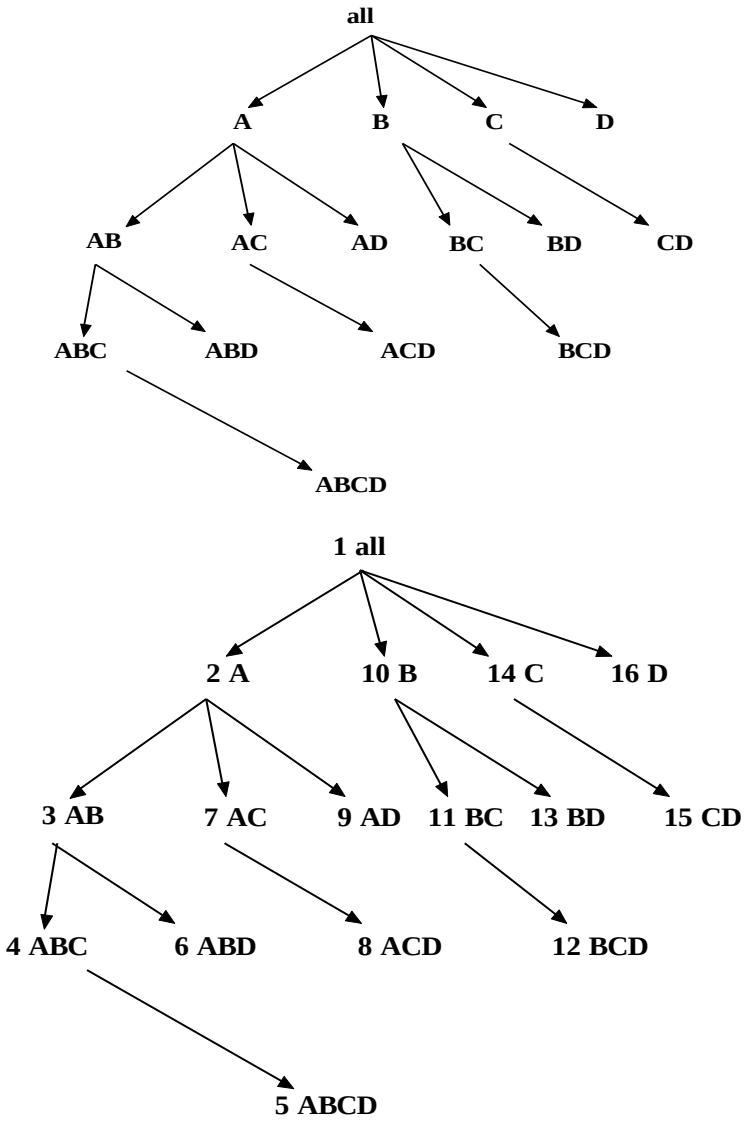
Data Cube Computation Methods

- Multi-Way Array Aggregation
- BUC 

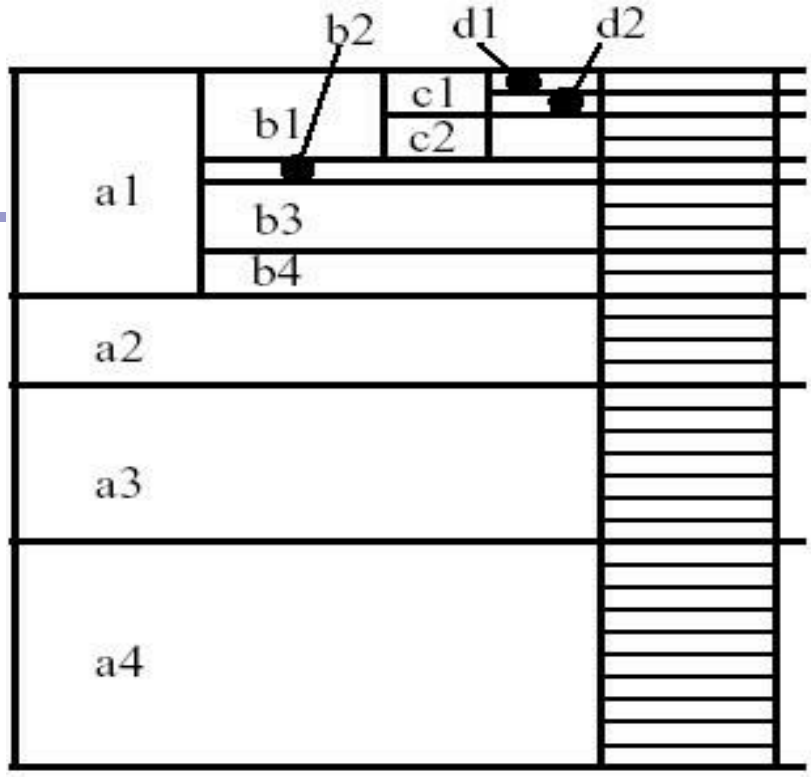
Bottom-Up Computation (BUC)

- BUC (Beyer & Ramakrishnan, SIGMOD'99)
- Bottom-up cube computation

  (Note: top-down in our view!)
- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *minsup* = 1 $\Rightarrow$ compute full CUBE!
- No simultaneous aggregation

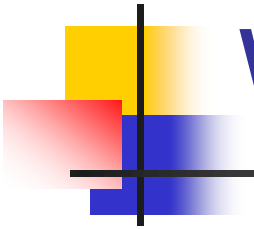


BUC: Partitioning



- Usually, entire data set can't fit in main memory
- Sort *distinct* values
  - partition into blocks that fit
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
  - Collapsing duplicates
    - Can't do holistic aggregates anymore!

Attribute-Based Data Warehouse



- Problem
  - Data Warehouse
  - NP-hardness
- Algorithm
- Performance Study

Data Warehouse

Parts are bought from suppliers and then sold to customers at a sale price SP

Table T

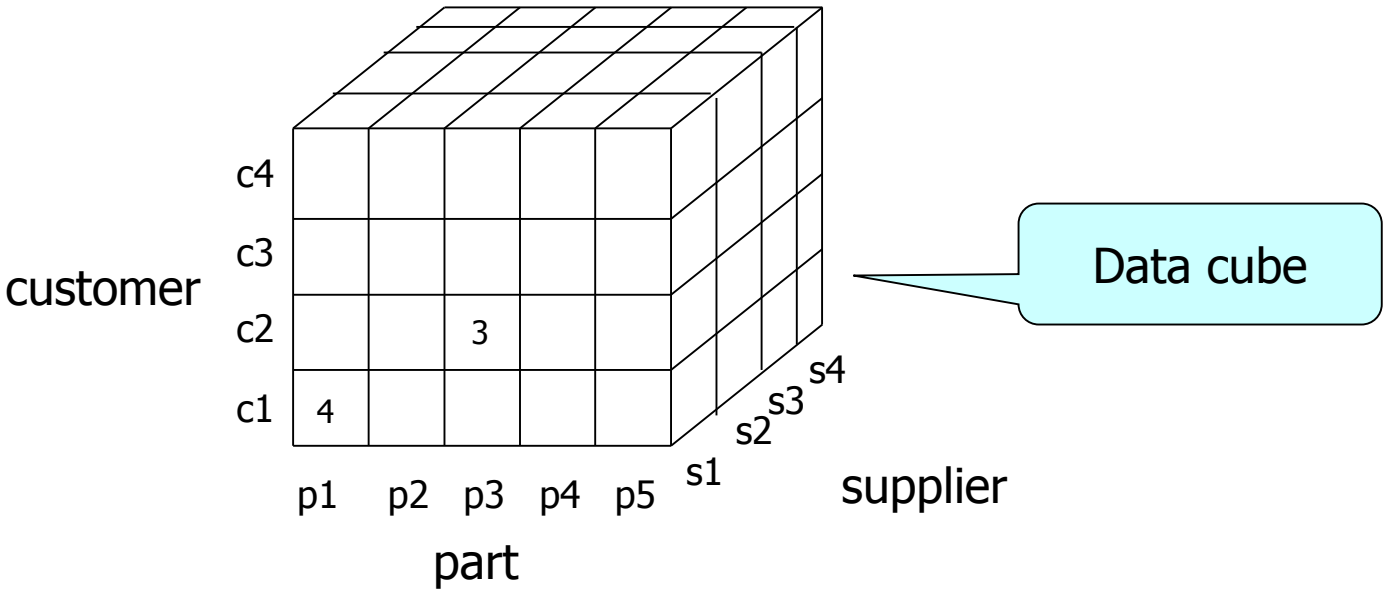
part	supplier	customer	SP
p1	s1	c1	4
p3	s1	c2	3
p2	s3	c1	7
…	…	…	…

Data Warehous

Table T

part	supplier	customer	SP
p1	s1	c1	4
p3	s1	c2	3
p2	s3	c1	7
…	…	…	…

Parts are bought from suppliers and then sold to customers at a sale price SP



Data Warehous

Table T

part	supplier	customer	SP
p1	s1	c1	4
p3	s1	c2	3
p2	s3	c1	7
…	…	…	…

Parts are bought from suppliers and then sold to customers at a sale price SP

e.g.,
select part, customer, SUM(SP)
from table T
group by part, customer

part	customer	SUM(SP)
p1	c1	4
p3	c2	3
p2	c1	7

pc 3

e.g.,
select customer, SUM(SP)
from table T
group by customer

customer	SUM(SP)
c1	11
c2	3

c 2

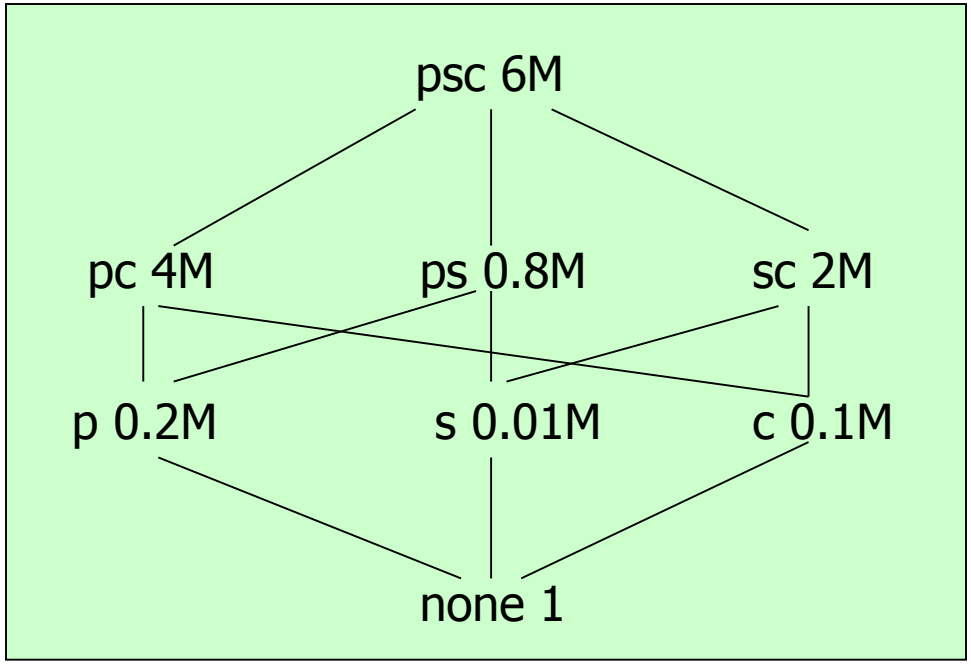
AVG(SP), MAX(SP), MIN(SP), …

Data Warehous

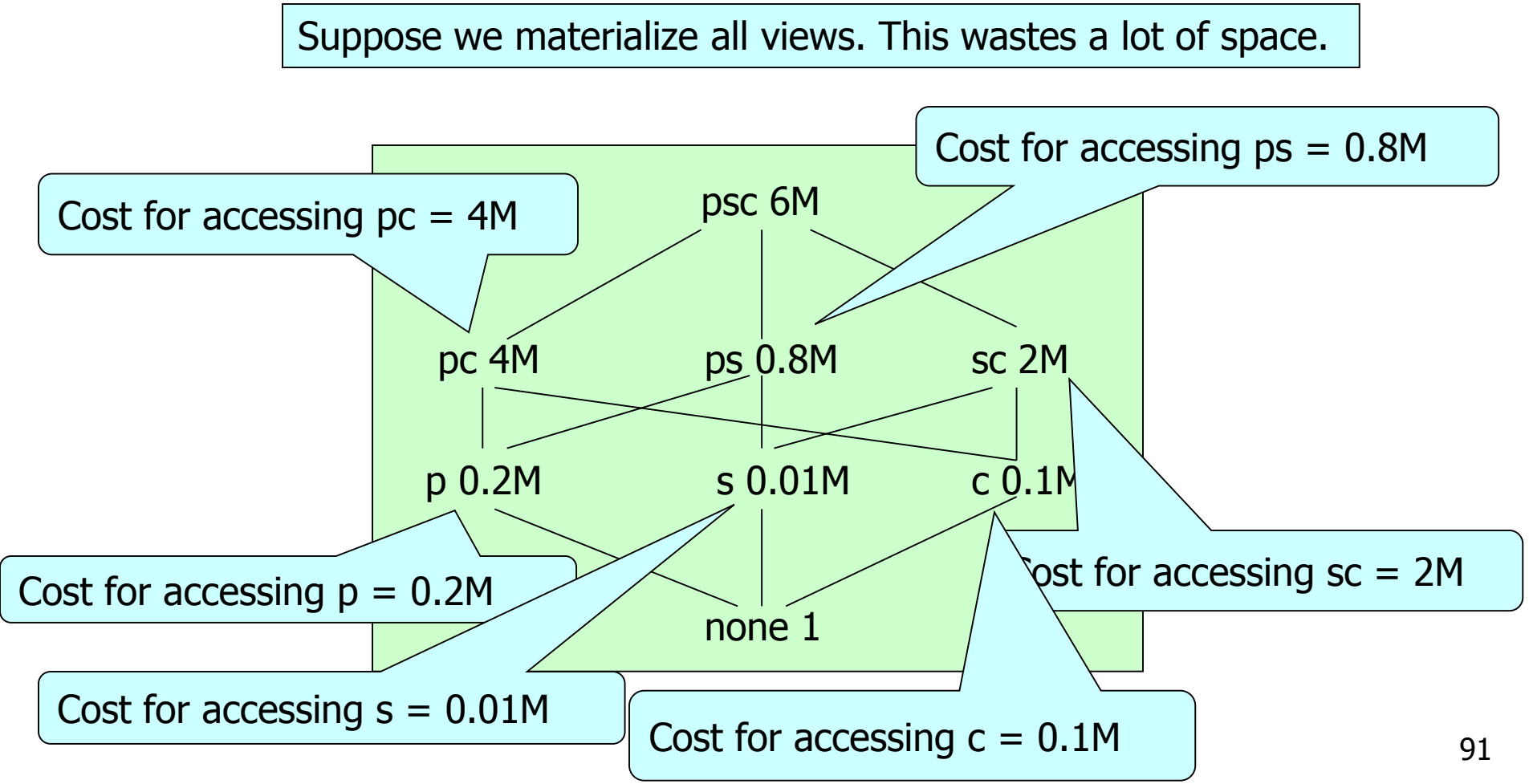
Table T

part	supplier	customer	SP
p1	s1	c1	4
p3	s1	c2	3
p2	s3	c1	7
…	…	…	…

Parts are bought from suppliers and then sold to customers at a sale price SP

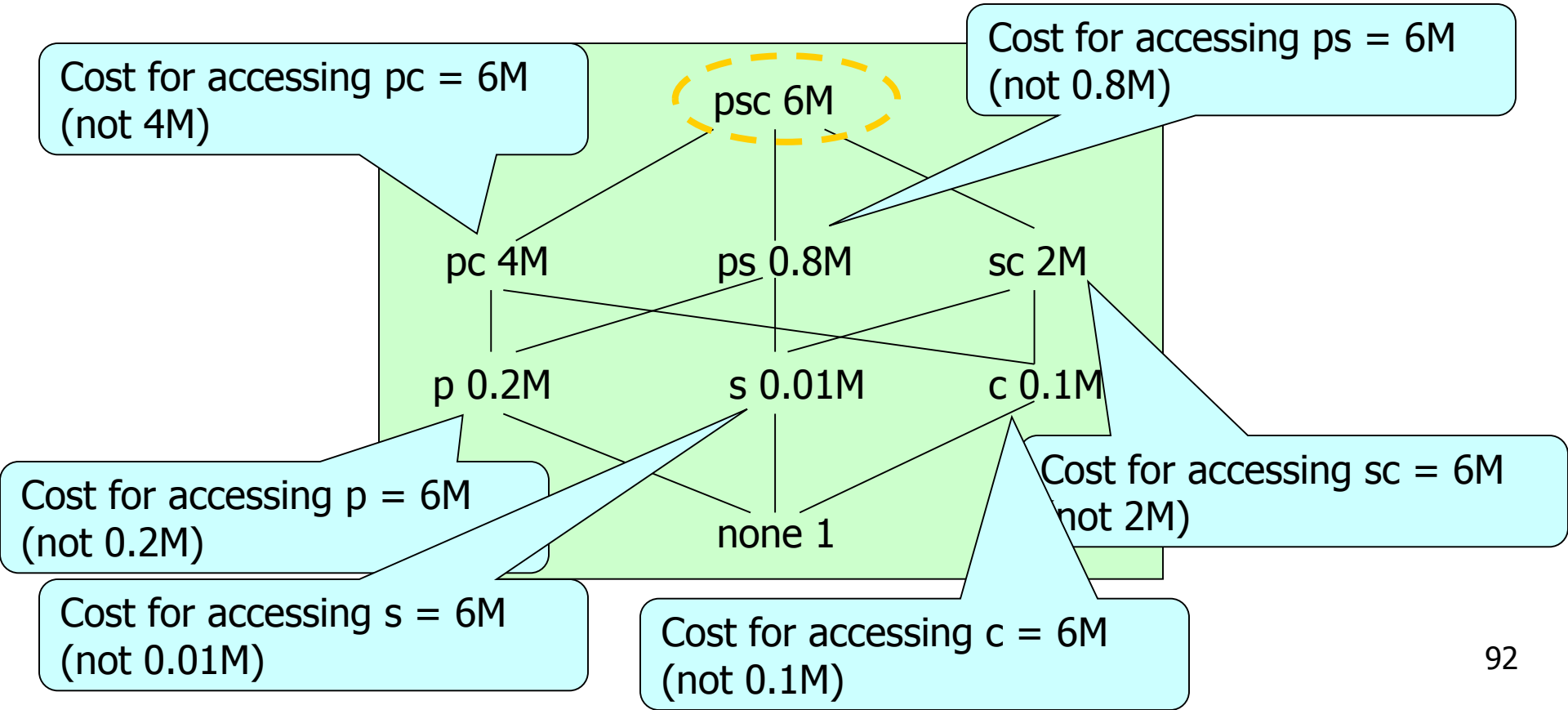


Data Warehouse



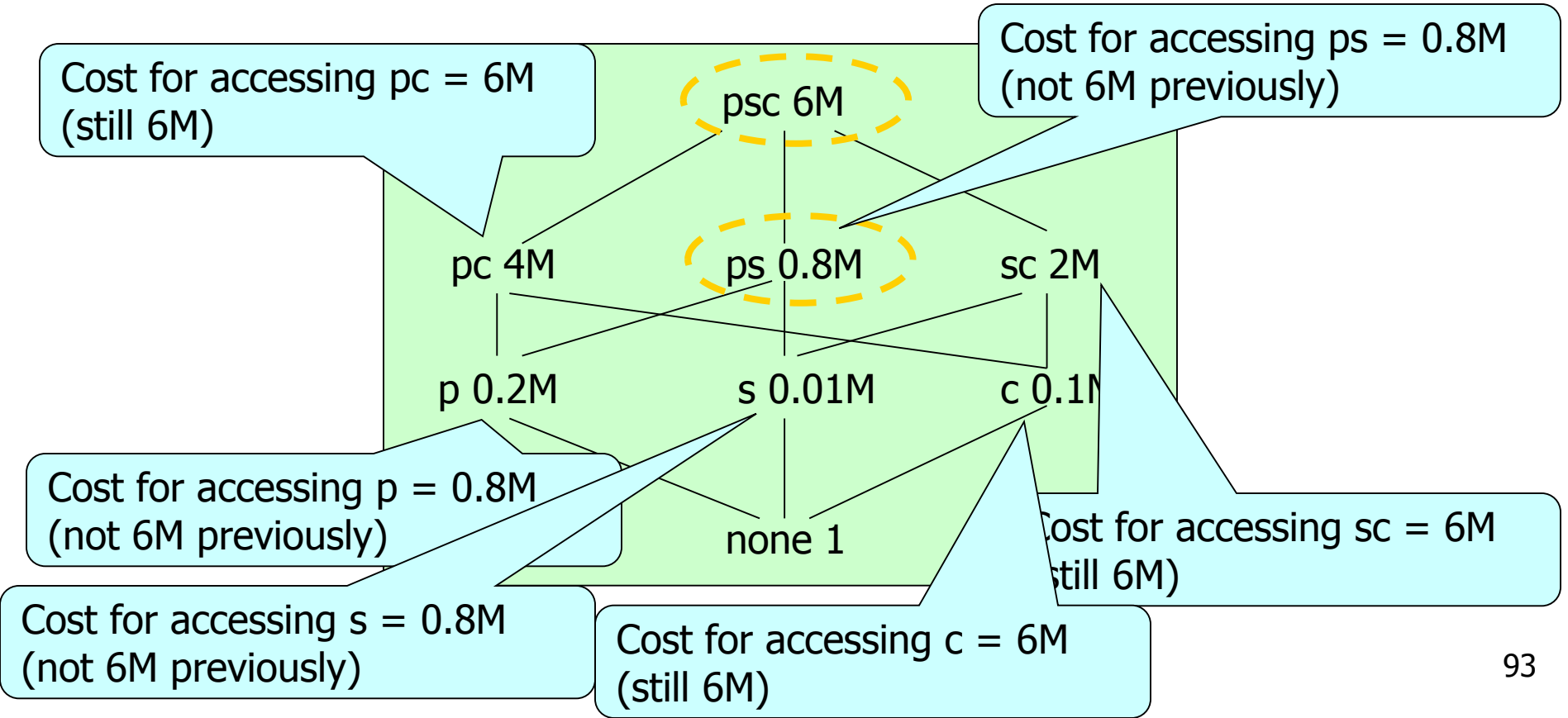
Data Warehouse

Suppose we materialize the top view only.



Data Warehouse

Suppose we materialize the top view and the view for "ps" only.

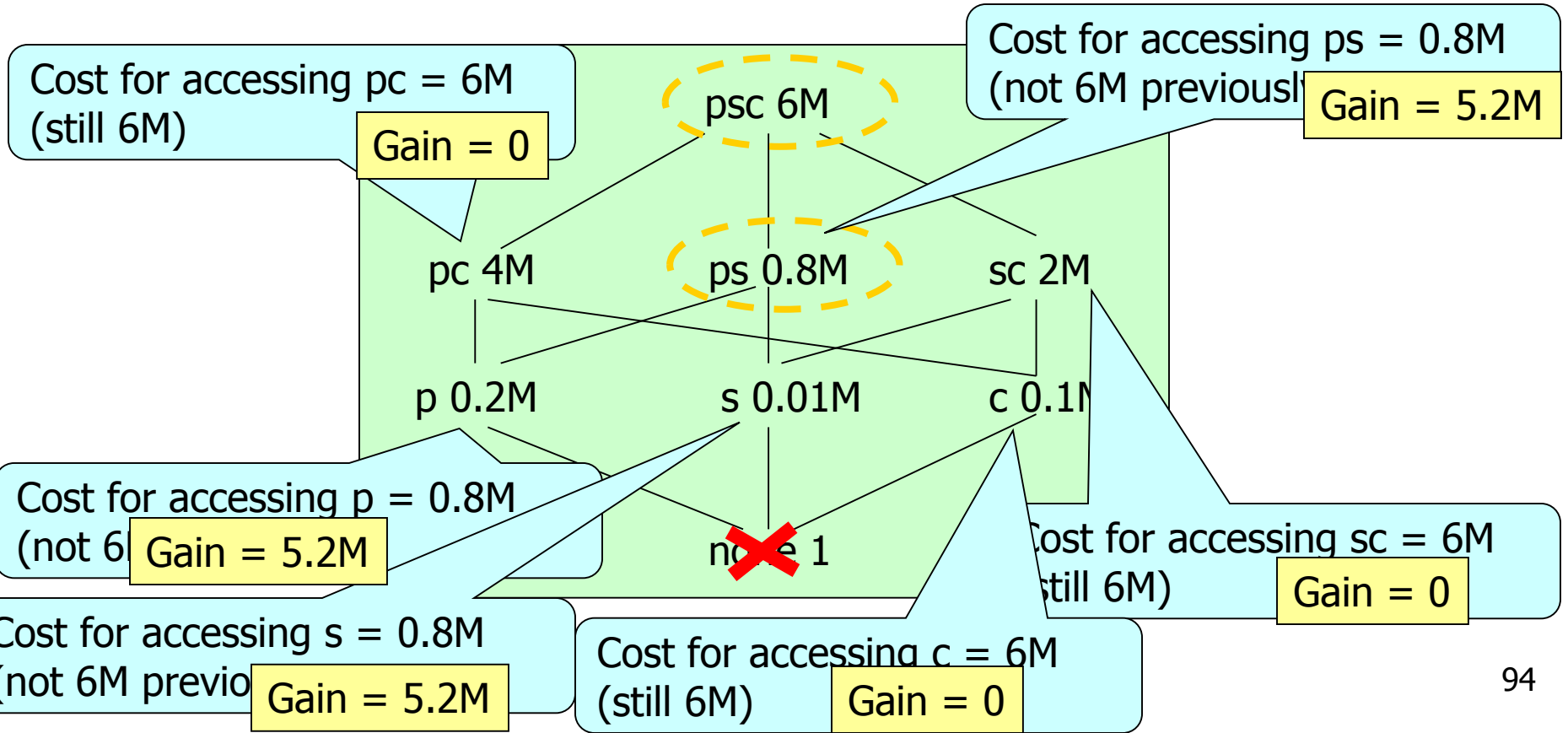


Gain({view for "ps", top view}, {top view}) = 5.2*3 = 15.6

Data

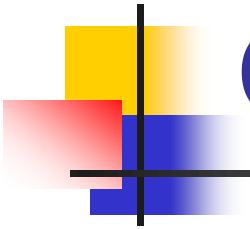
Selective Materialization Problem:
We can select a set V of k views such that
Gain(V U {top view}, {top view}) is maximized.

Suppose we materialize the top view and the view for "ps" only.



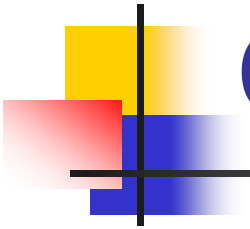
Attribute-Based Data Warehouse

- Problem
  - Date Warehouse
- Algorithm



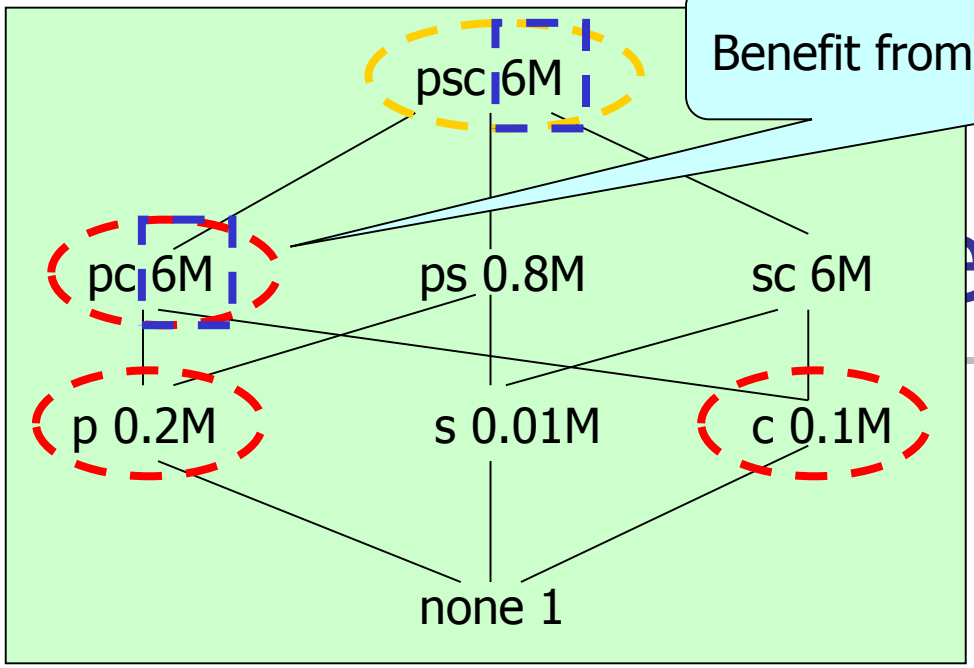
Greedy Algorithm

- k = number of views to be materialized
- Given
  - v is a view
  - S is a set of views which are selected to be materialized
- Define the benefit of selecting v for materialization as
  - B(v, S) = Gain(S U v, S)



Greedy Algorithm

- S ← {top view};
- For i = 1 to k do
  - Select that view v not in S such that B(v, S) is maximized;
  - S ← S U {v}
- Resulting S is the greedy selection

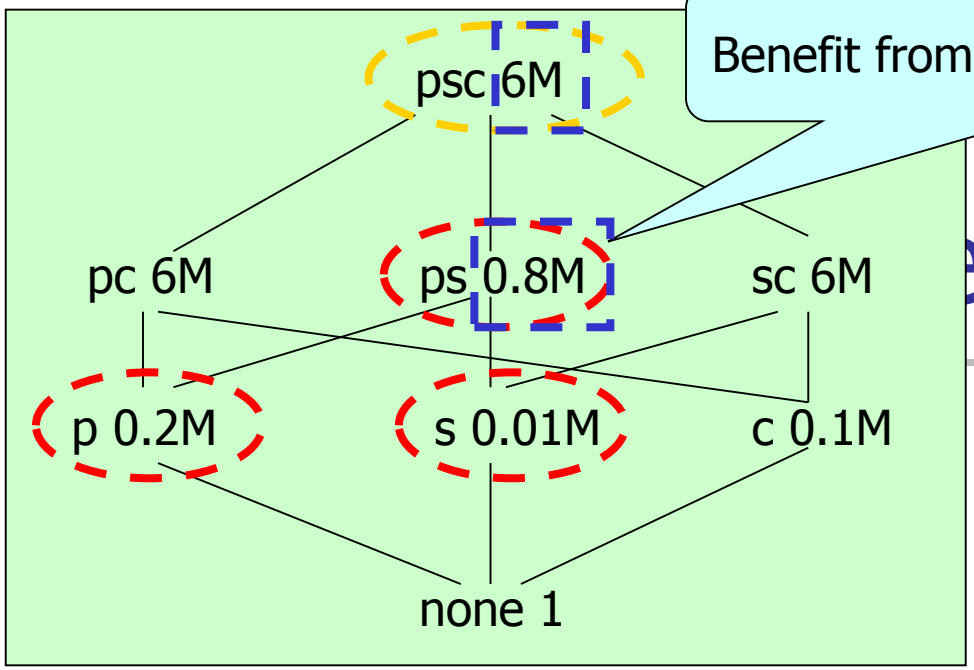


Benefit from pc = 6M-6M = 0

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps		
sc		
p		
s		
c		

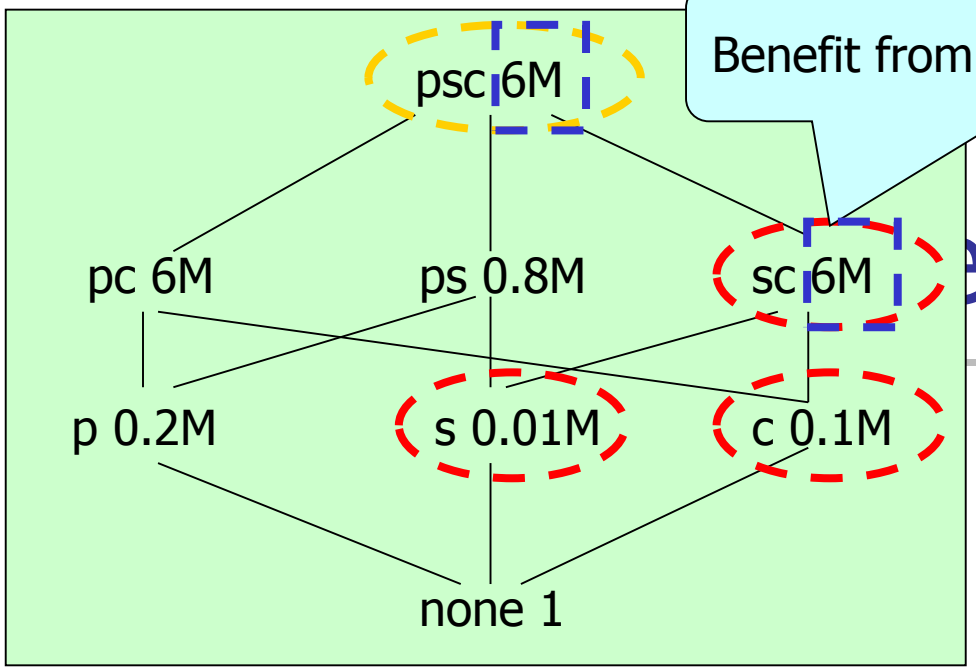


Benefit from ps = 6M-0.8M= 5.2M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps	5.2 x 3 = 15.6	
sc		
p		
s		
c		

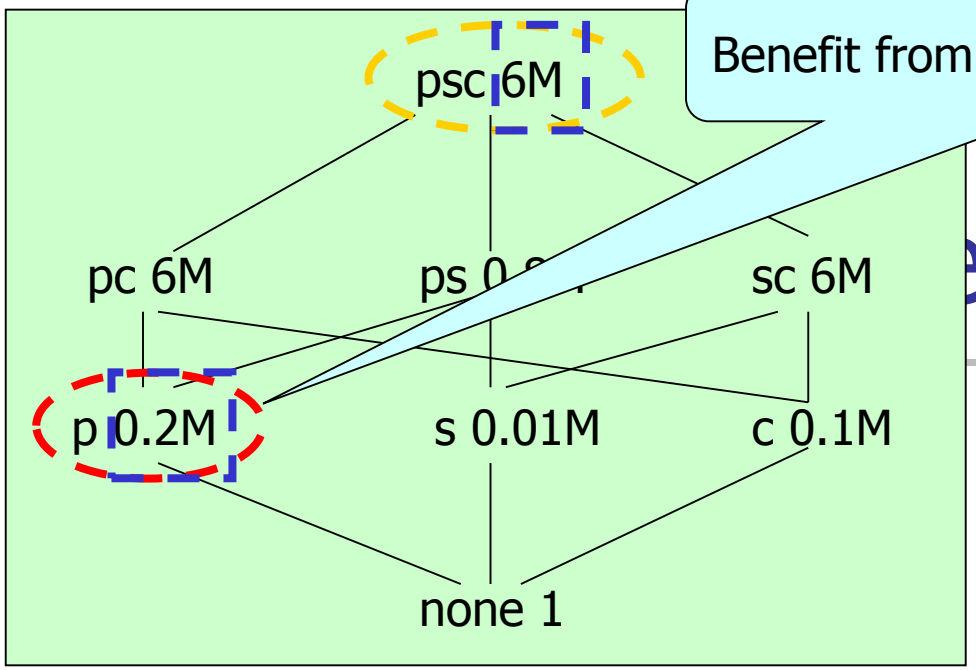


Benefit from sc = 6M-6M= 0

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	
p		
s		
c		

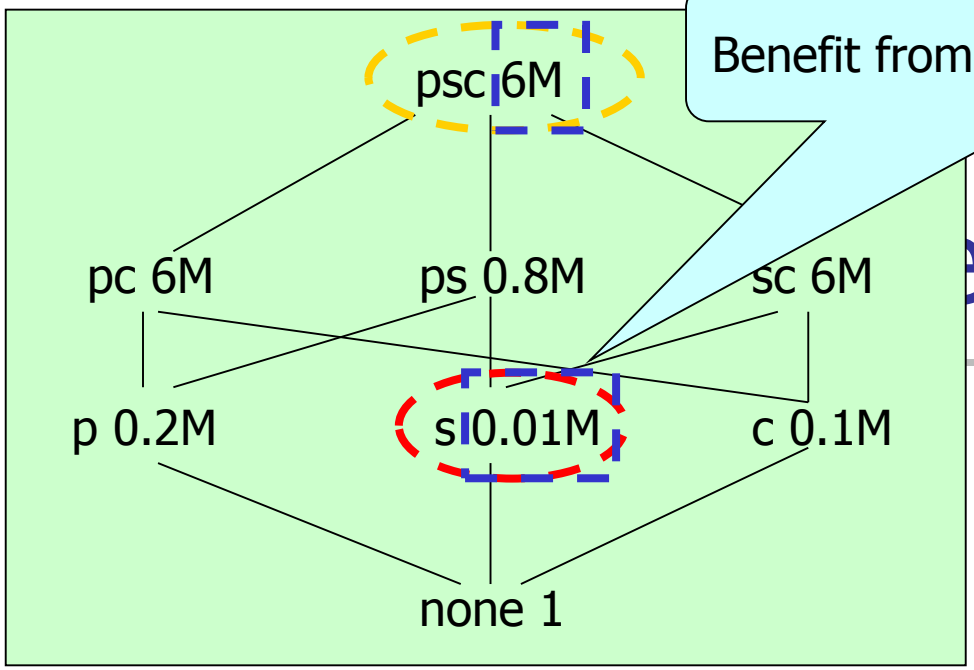


Benefit from p = 6M-0.2M = 5.8M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	
p	5.8 x 1 = 5.8	
s		
c		

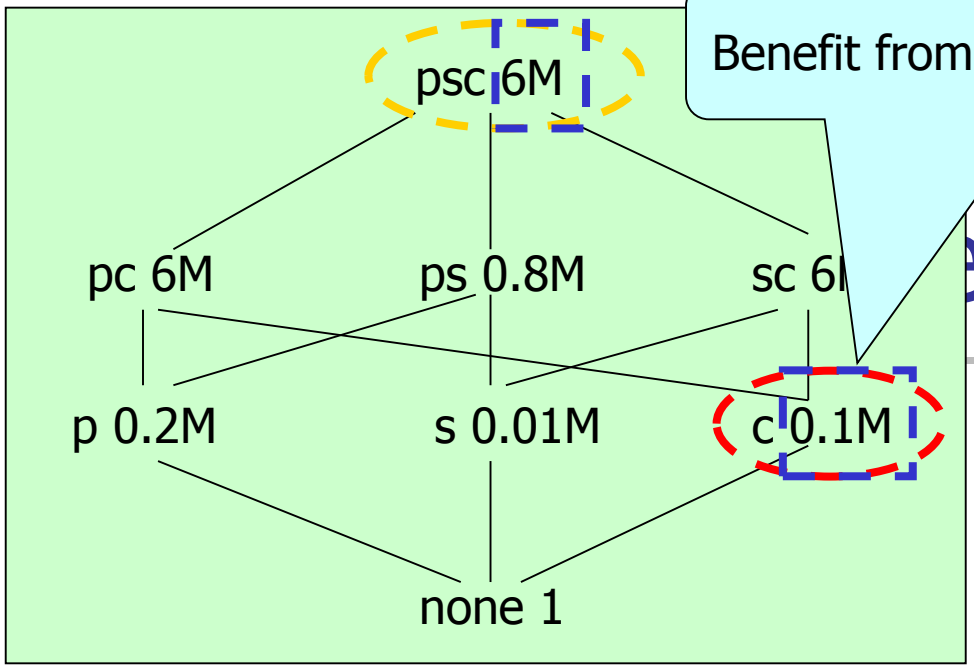


Benefit from s = 6M-0.01M = 5.99M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	
p	5.8 x 1 = 5.8	
s	5.99 x 1 = 5.99	
c		

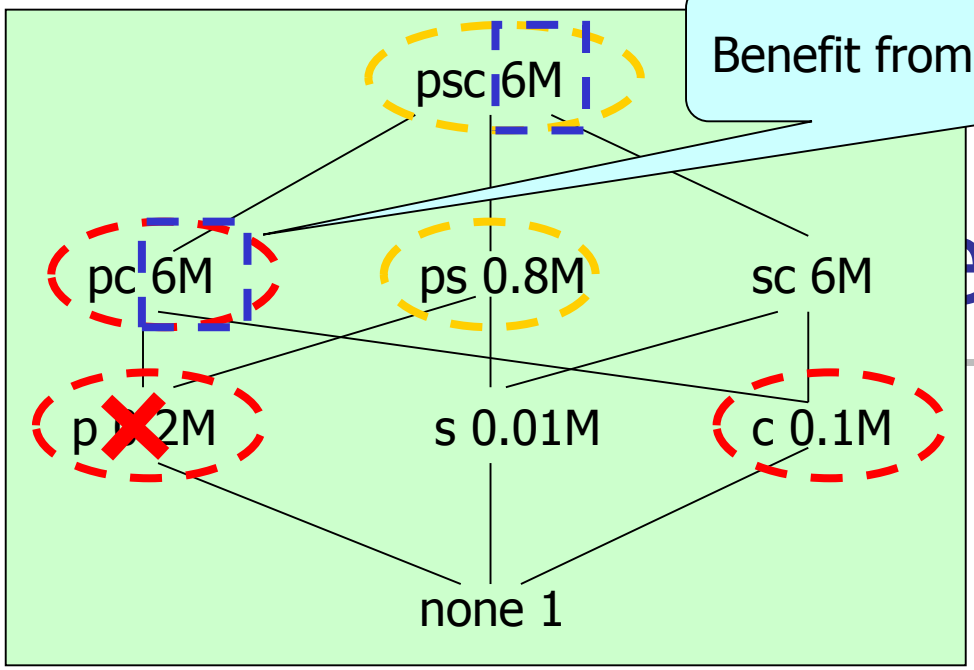


Benefit from c = 6M-0.1M = 5.9M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	
p	5.8 x 1 = 5.8	
s	5.99 x 1 = 5.99	
c	5.9 x 1 = 5.9	

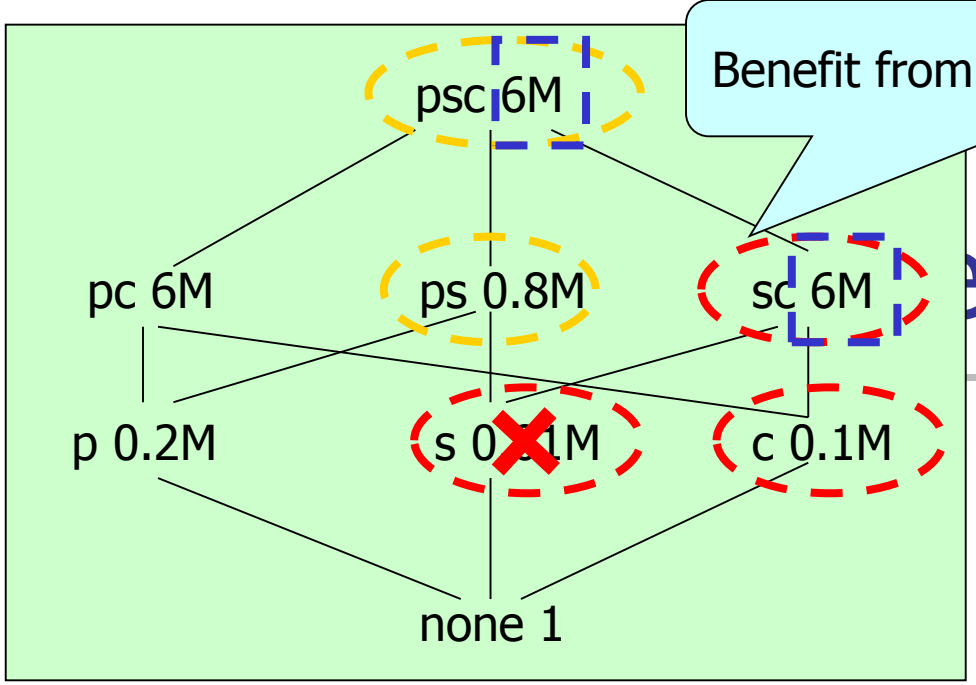


Benefit from pc = 6M-6M = 0

k = 2

Benefit

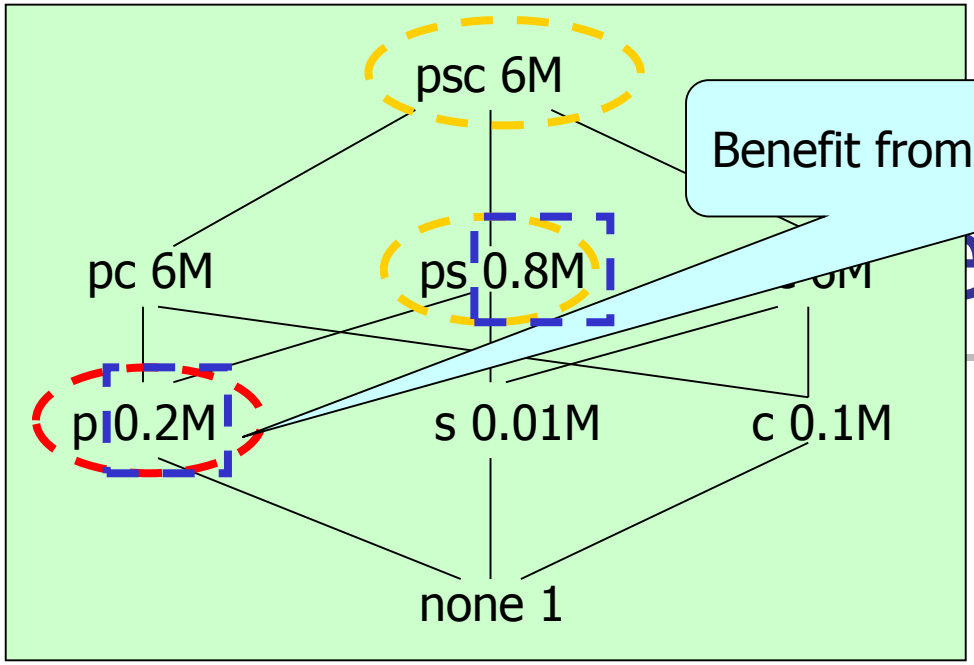
	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	
p	5.8 x 1 = 5.8	
s	5.99 x 1 = 5.99	
c	5.9 x 1 = 5.9	



k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	0 x 2 = 0
p	5.8 x 1 = 5.8	
s	5.99 x 1 = 5.99	
c	5.9 x 1 = 5.9	

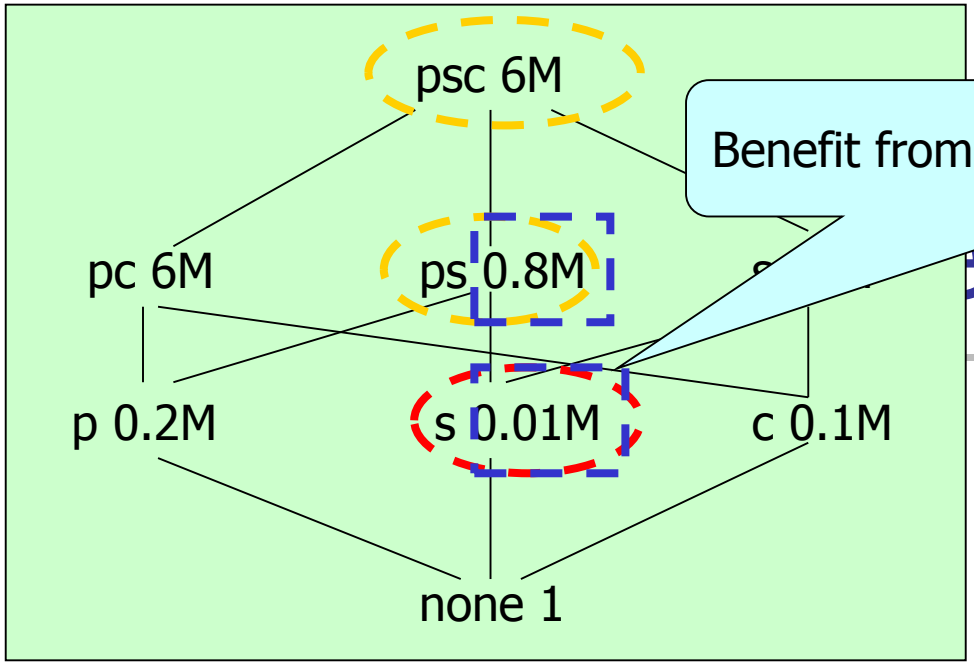


Benefit from p = 0.8M-0.2M = 0.6M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	0 x 2 = 0
p	5.8 x 1 = 5.8	0.6 x 1 = 0.6
s	5.99 x 1 = 5.99	
c	5.9 x 1 = 5.9	

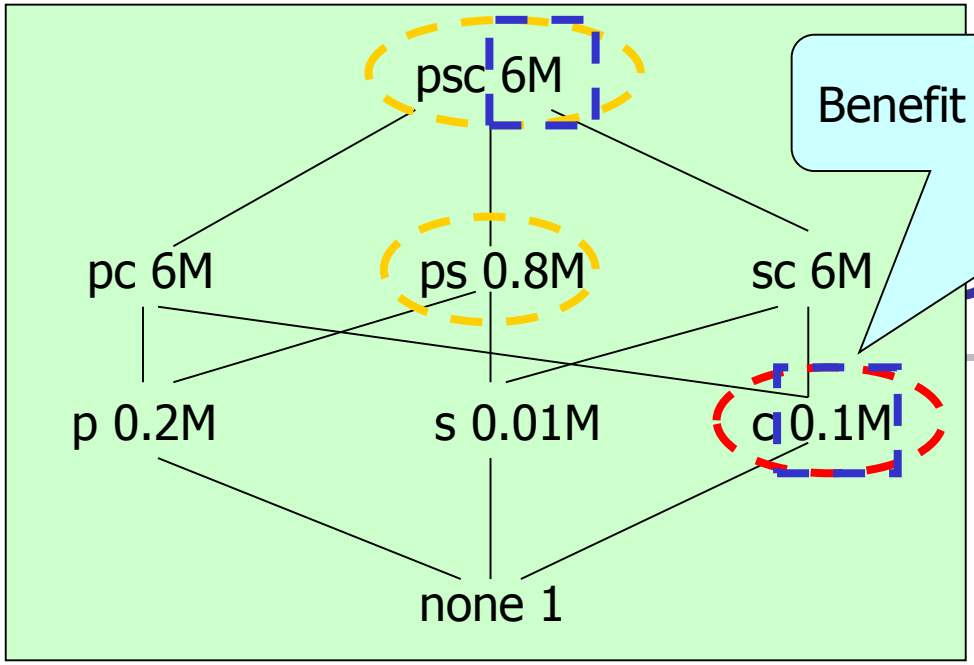


k = 2

Benefit from s = 0.8M-0.01M = 0.79M

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	0 x 2 = 0
p	5.8 x 1 = 5.8	0.6 x 1 = 0.6
s	5.99 x 1 = 5.99	0.79 x 1 = 0.79
c	5.9 x 1 = 5.9	

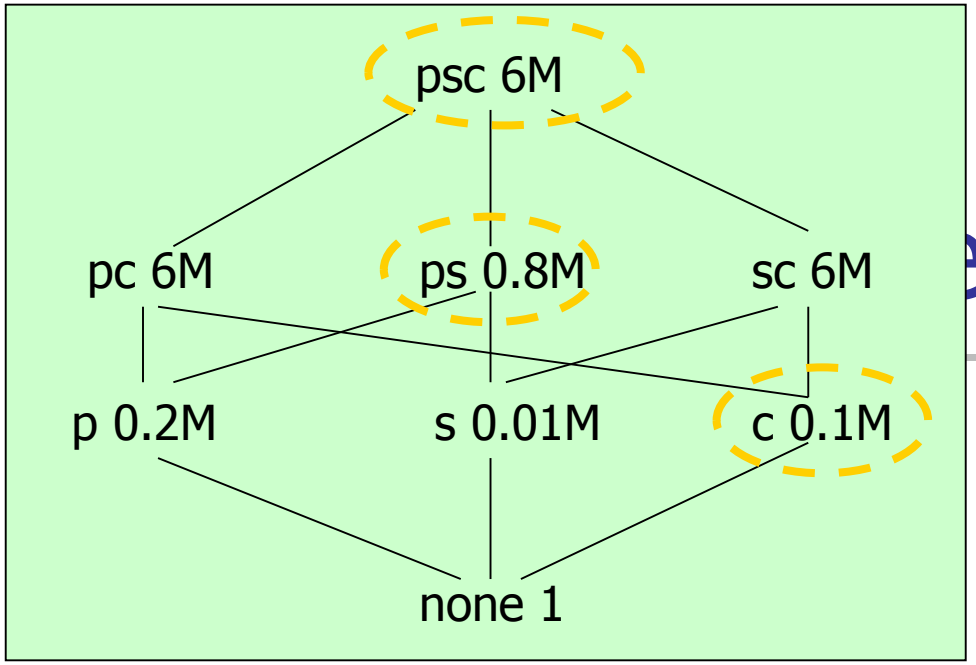


Benefit from c = 6M-0.1M = 5.9M

k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	0 x 2 = 0
p	5.8 x 1 = 5.8	0.6 x 1 = 0.6
s	5.99 x 1 = 5.99	0.79 x 1 = 0.79
c	5.9 x 1 = 5.9	5.9 x 1 = 5.9



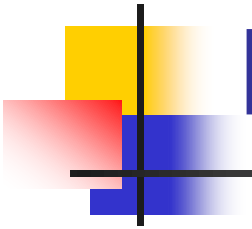
k = 2

Benefit

	1st Choice (M)	2nd Choice (M)
pc	0 x 3 = 0	0 x 2 = 0
ps	5.2 x 3 = 15.6	
sc	0 x 3 = 0	0 x 2 = 0
p	5.8 x 1 = 5.8	0.6 x 1 = 0.6
s	5.99 x 1 = 5.99	0.79 x 1 = 0.79
c	5.9 x 1 = 5.9	5.9 x 1 = 5.9

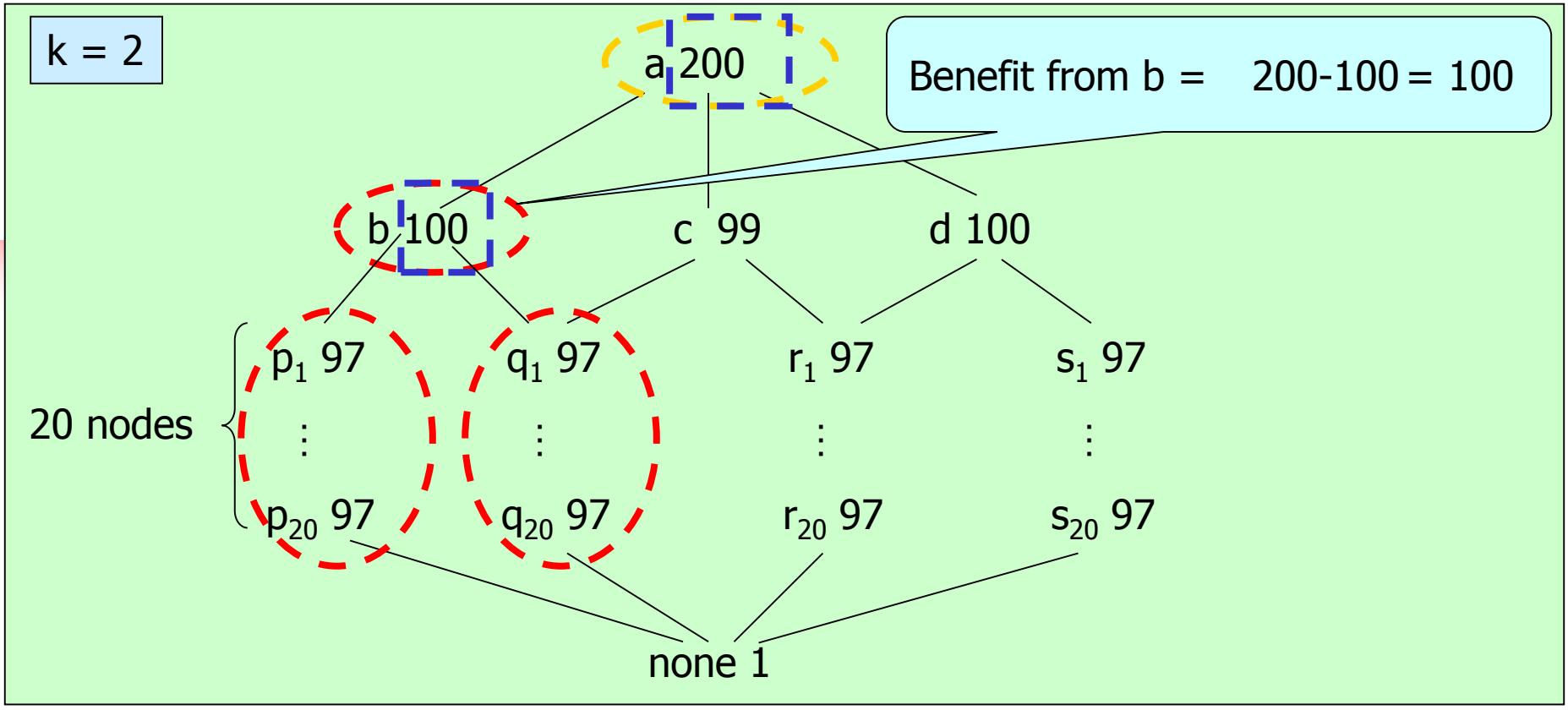
Two views to be materialized are
1. ps
2. c

V = {ps, c}
Gain(V U {top view}, {top view})
= 15.6 + 5.9 = 21.5



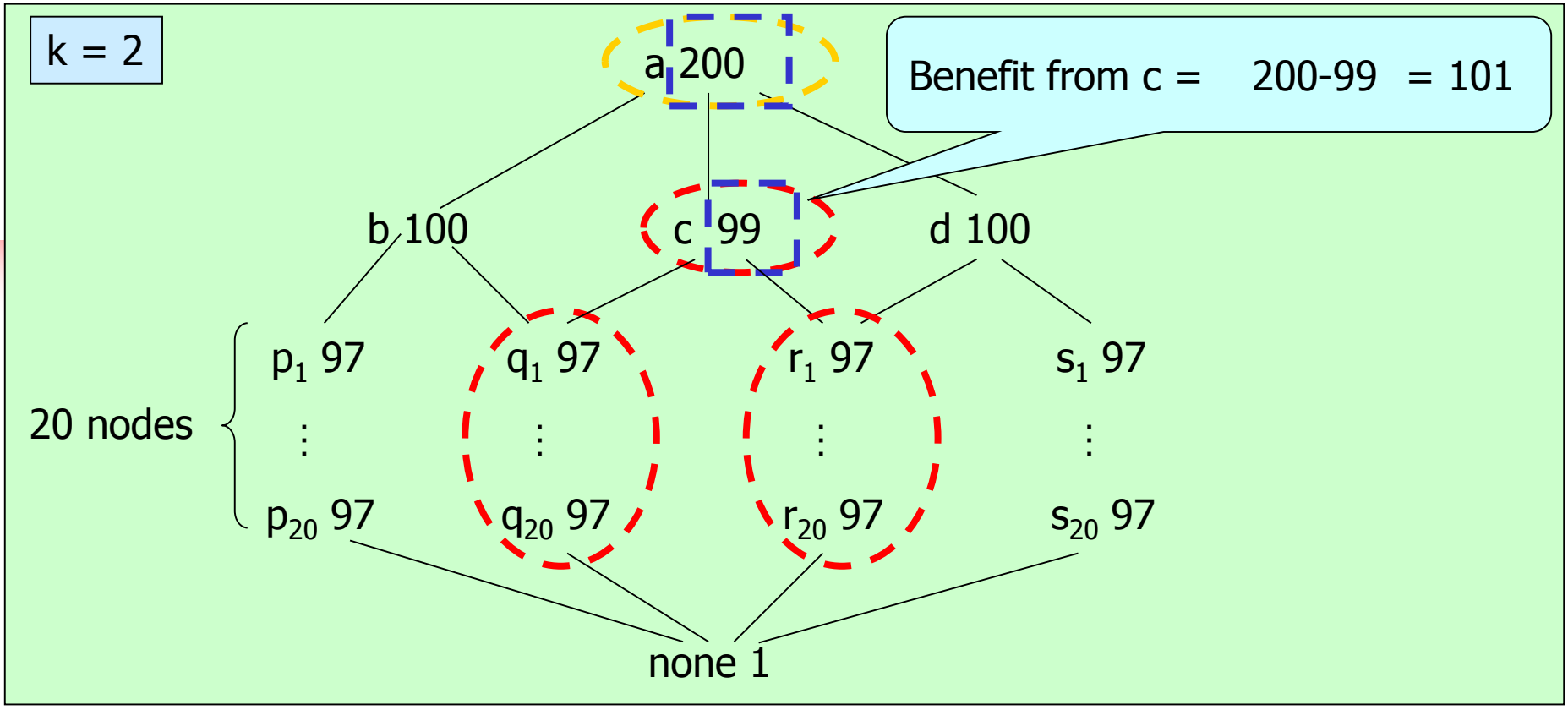
Performance Study

- How bad does the Greedy Algorithm perform?



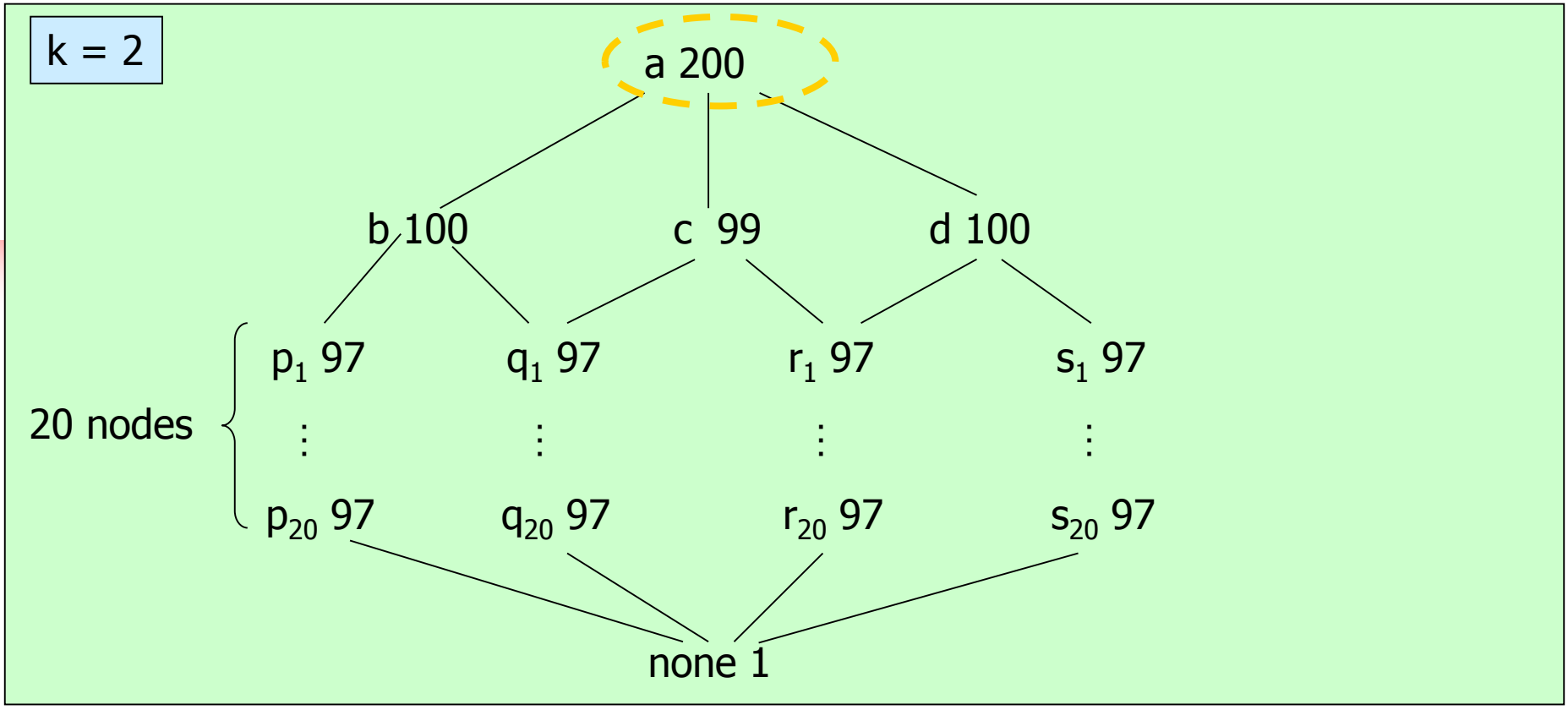
Benefit

	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	
c		
d		
…	…	…



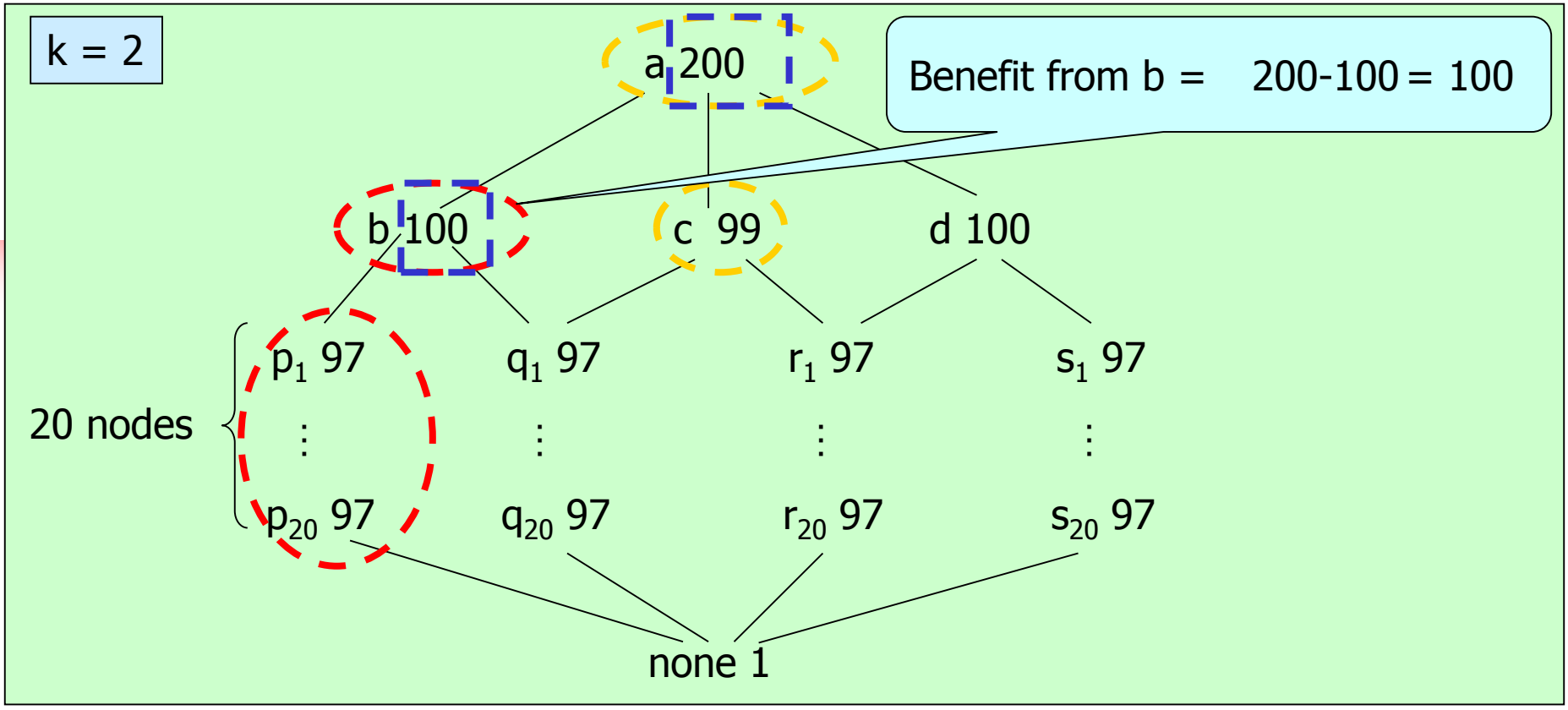
Benefit

	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	
c	41 x 101= 4141	
d		
…	…	…



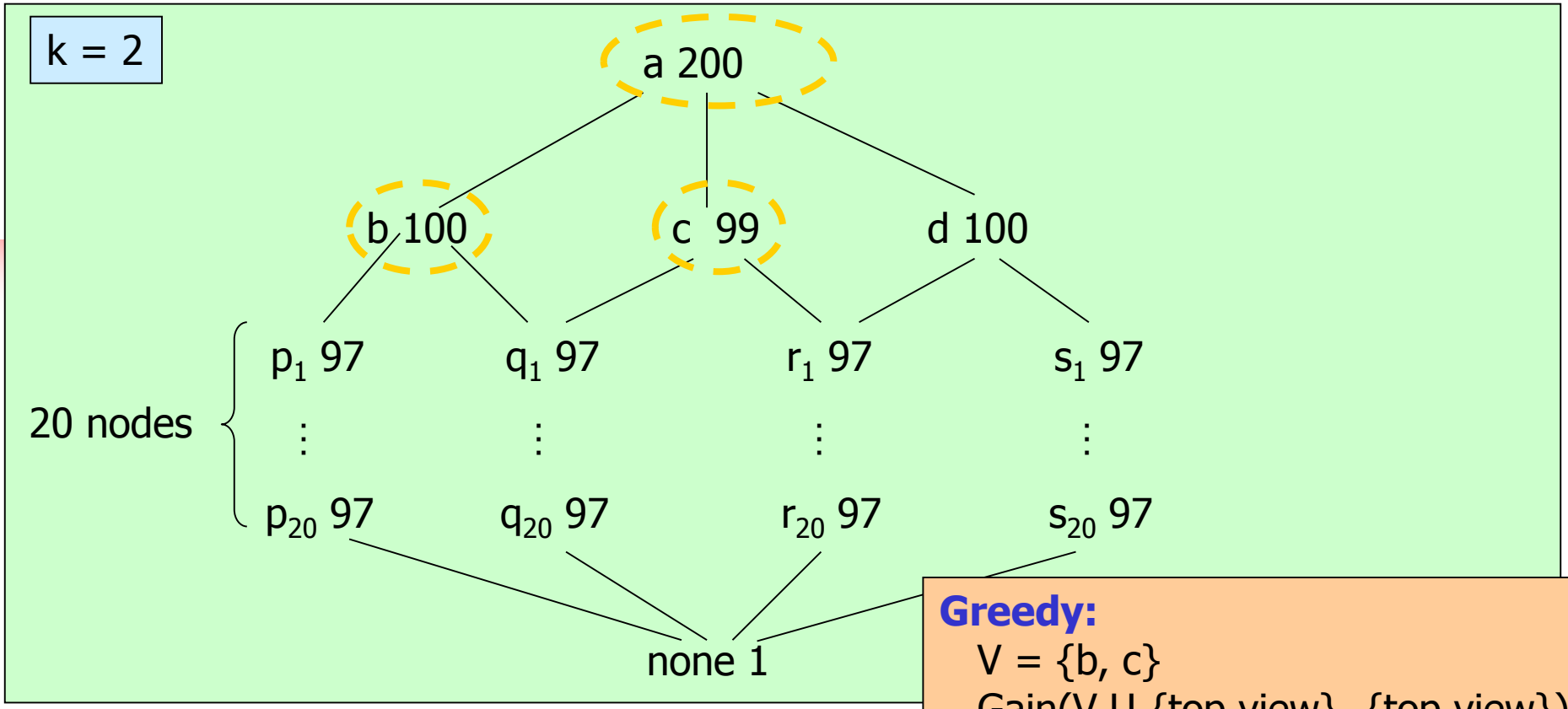
Benefit

	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	
c	41 x 101= 4141	
d	41 x 100= 4100	
…	…	…



Benefit

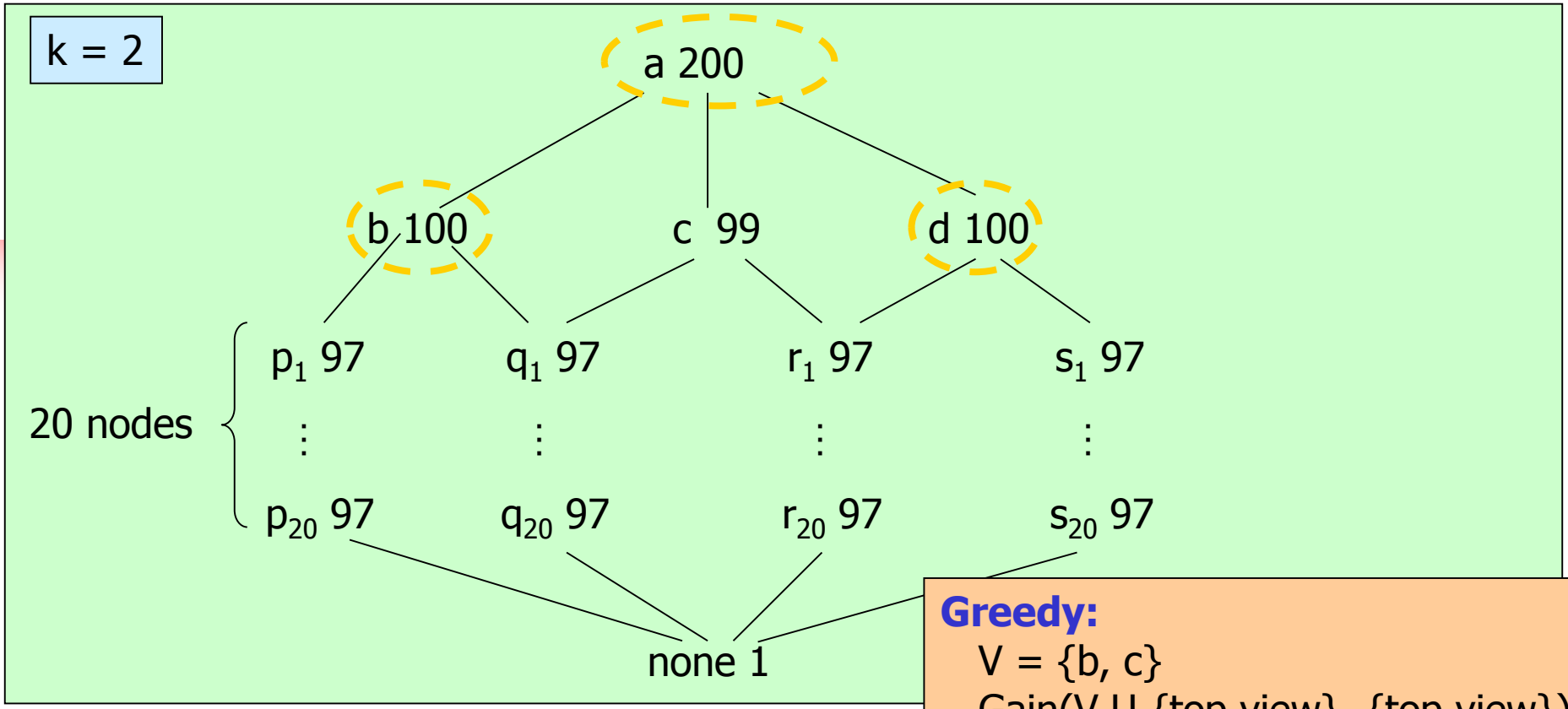
	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	21 x 100= 2100
c	41 x 101= 4141	
d	41 x 100= 4100	
…	…	…



Greedy:
V = {b, c}
Gain(V U {top view}, {top view})
= 4141 + 2100 = 6241

Benefit

	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	21 x 100= 2100
c	41 x 101= 4141	
d	41 x 100= 4100	21 x 100= 2100
…	…	…

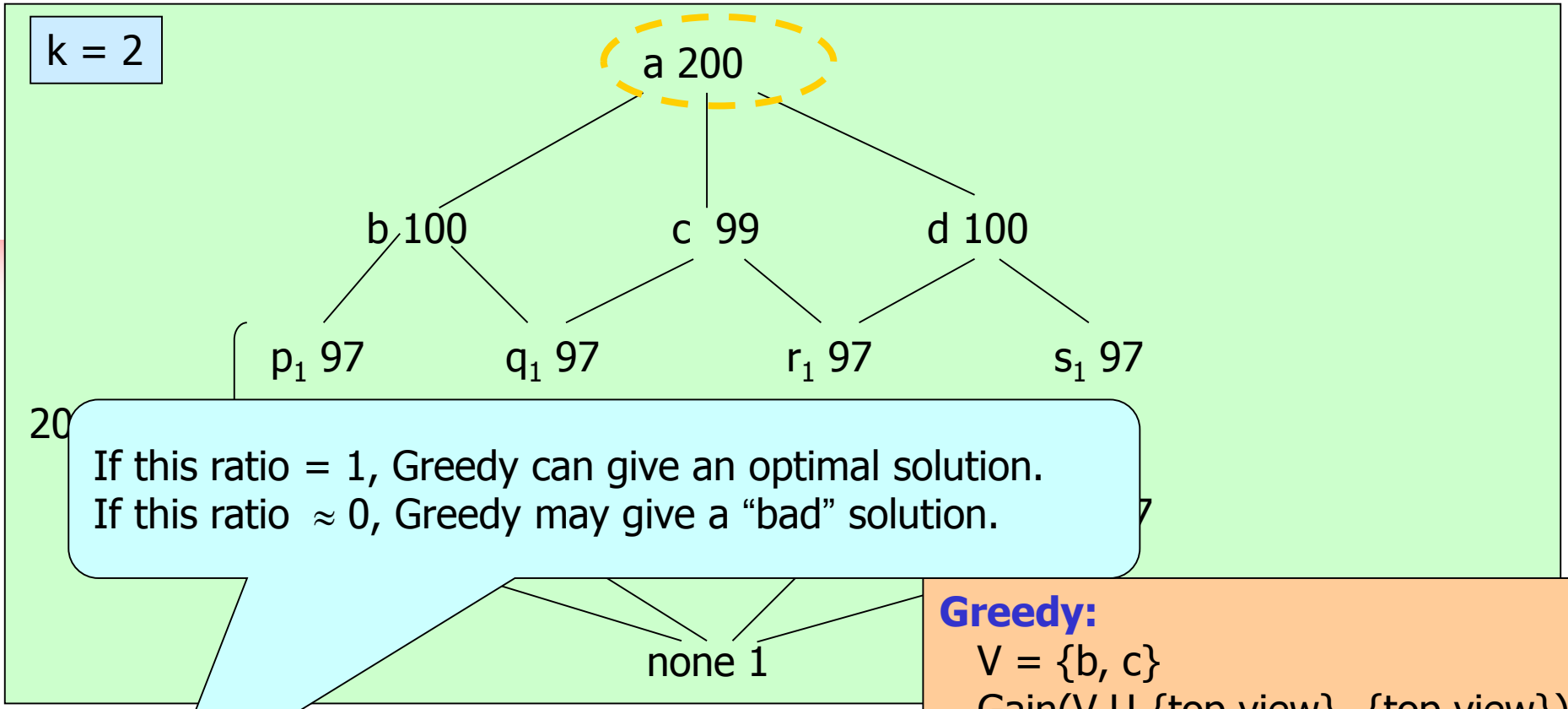


Greedy:
V = {b, c}
Gain(V U {top view}, {top view})
= 4141 + 2100 = 6241

Benefit

	1st Choice (M)	2nd Choice (M)
b	41 x 100= 4100	
c	41 x 101= 4141	21 x 101 + 20 x 1 = 2141
d	41 x 100= 4100	41 x 100= 4100
…	…	…

Optimal:
V = {b, d}
Gain(V U {top view}, {top view})
= 4100 + 4100 = 8200



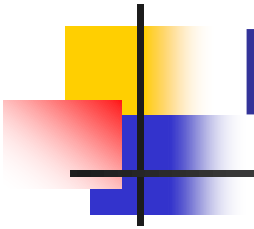
Greedy:
V = {b, c}
Gain(V U {top view}, {top view})
= 4141 + 2100 = 6241

$$\frac{\text{Greedy}}{\text{Optimal}} = \frac{6241}{8200} = 0.7611$$

Does this ratio has a "lower" bound?

It is proved that this ratio is at least 0.63.

Optimal:
V = {b, d}
Gain(V U {top view}, {top view})
= 4100 + 4100 = 8200



Performance Study

- This is just an example to show that this greedy algorithm can perform badly.
- A complete proof of the lower bound can be found in the paper.

Summary

- Data warehousing: A multi-dimensional model of a data warehouse
  - A data cube consists of *dimensions* & *measures*
  - Star schema, snowflake schema, fact constellations
  - OLAP operations: drilling, rolling, slicing, dicing and pivoting
- Data Warehouse Architecture, Design, and Usage
  - Multi-tiered architecture
  - Business analysis design framework
  - Information processing, analytical processing, data mining, OLAM (Online Analytical Mining)
- Implementation: Efficient computation of data cubes
  - Partial vs. full vs. no materialization
  - Indexing OALP data: Bitmap index and join index
  - OLAP query processing
  - OLAP servers: ROLAP, MOLAP, HOLAP
- Data generalization: Attribute-oriented induction

References (I)

- S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96
- D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. SIGMOD'97
- R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. ICDE'97
- S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26:65-74, 1997
- E. F. Codd, S. B. Codd, and C. T. Salley. Beyond decision support. Computer World, 27, July 1993.
- J. Gray, et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Mining and Knowledge Discovery, 1:29-54, 1997.
- A. Gupta and I. S. Mumick. Materialized Views: Techniques, Implementations, and Applications. MIT Press, 1999.
- J. Han. Towards on-line analytical mining in large databases. *ACM SIGMOD Record*, 27:97-107, 1998.
- V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. SIGMOD'96

References (II)

- C. Imhoff, N. Galemmo, and J. G. Geiger. Mastering Data Warehouse Design: Relational and Dimensional Techniques. John Wiley, 2003
- W. H. Inmon. Building the Data Warehouse. John Wiley, 1996
- R. Kimball and M. Ross. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. 2ed. John Wiley, 2002
- P. O'Neil and D. Quass. Improved query performance with variant indexes. SIGMOD'97
- Microsoft. OLEDB for OLAP programmer's reference version 1.0. In http://www.microsoft.com/data/oledb/olap, 1998
- A. Shoshani. OLAP and statistical databases: Similarities and differences. PODS'00.
- S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. ICDE'94
- P. Valduriez. Join indices. ACM Trans. Database Systems, 12:218-246, 1987.
- J. Widom. Research problems in data warehousing. CIKM'95.
- K. Wu, E. Otoo, and A. Shoshani, Optimal Bitmap Indices with Efficient Compression, ACM Trans. on Database Systems (TODS), 31(1), 2006, pp. 1-38.