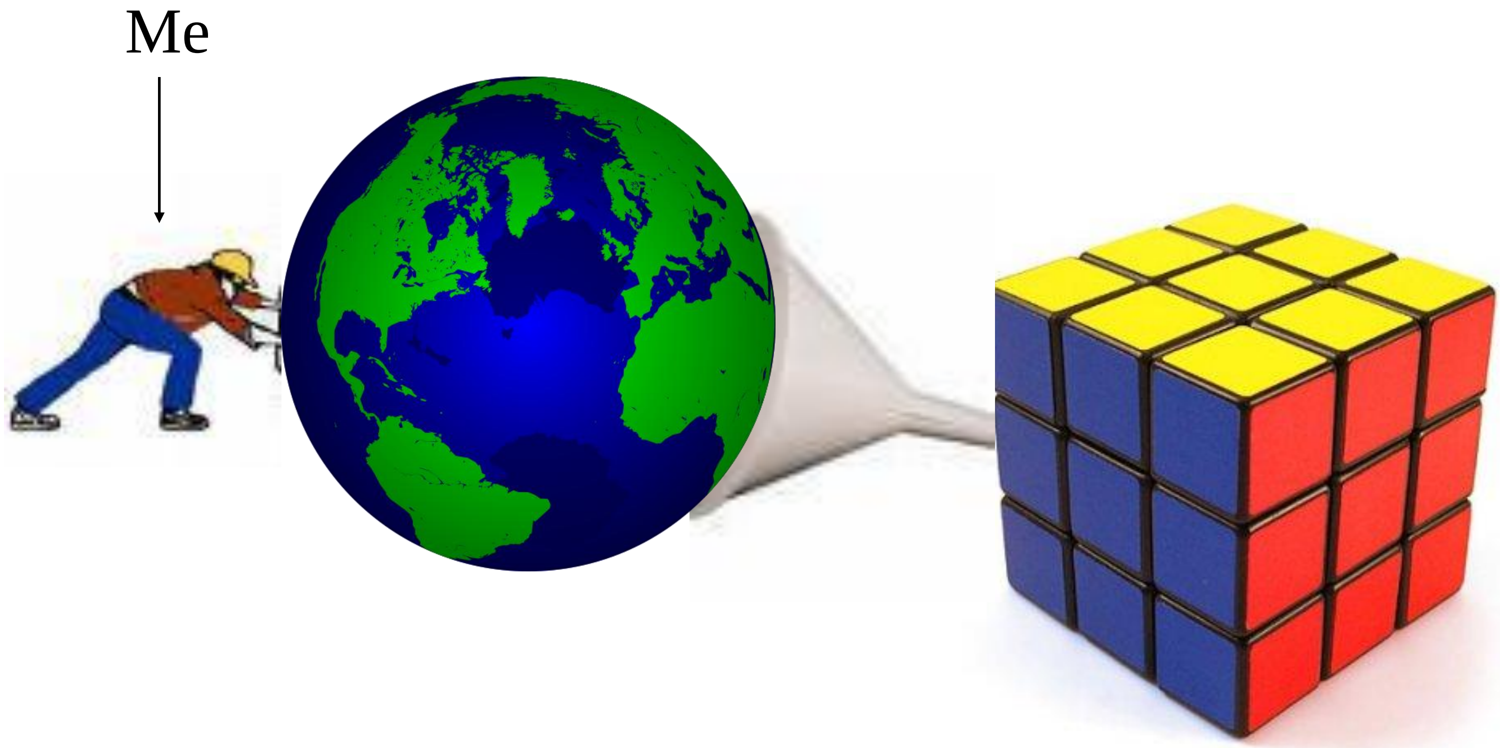
AALBORG UNIVERSITY

Managing Complex Multidimensional Data

Professor Torben Bach Pedersen

What do I do ?

- Well, I try to squeeze the world into cubes…



A Bit More Detail

- M.Sc. Computer science, Aarhus University 1994
- Database administrator Kommunedata (now CSC Scandihealth) 1994-97
  - Hospital information systems
  - Started working on data warehousing in 1995
  - And that was quite exciting, so:
- Industrial Ph.D. Fellow Kommunedata+Aalborg University 1997-2000
  - Research on clinical data warehousing
  - Research was kinda funny, so:
- Associate Professor CS, Aalborg University 2000-2007
  - Research in business intelligence, data warehousing, OLAP, data mining, integrating complex data (web,space+time,…)
- Professor CS Aalborg University 2008-(2039?)

Talk Overview

- **Multidimensional modeling recap**
  - **Cubes, dimensions, measures, …**
- Complex multidimensional data
  - Modeling
  - Performance techniques
- Complex spatial multidimensional data
- Integrating cubes and XML
- Semantic web warehousing
- Integrating cubes and text
- Multidimensional music data

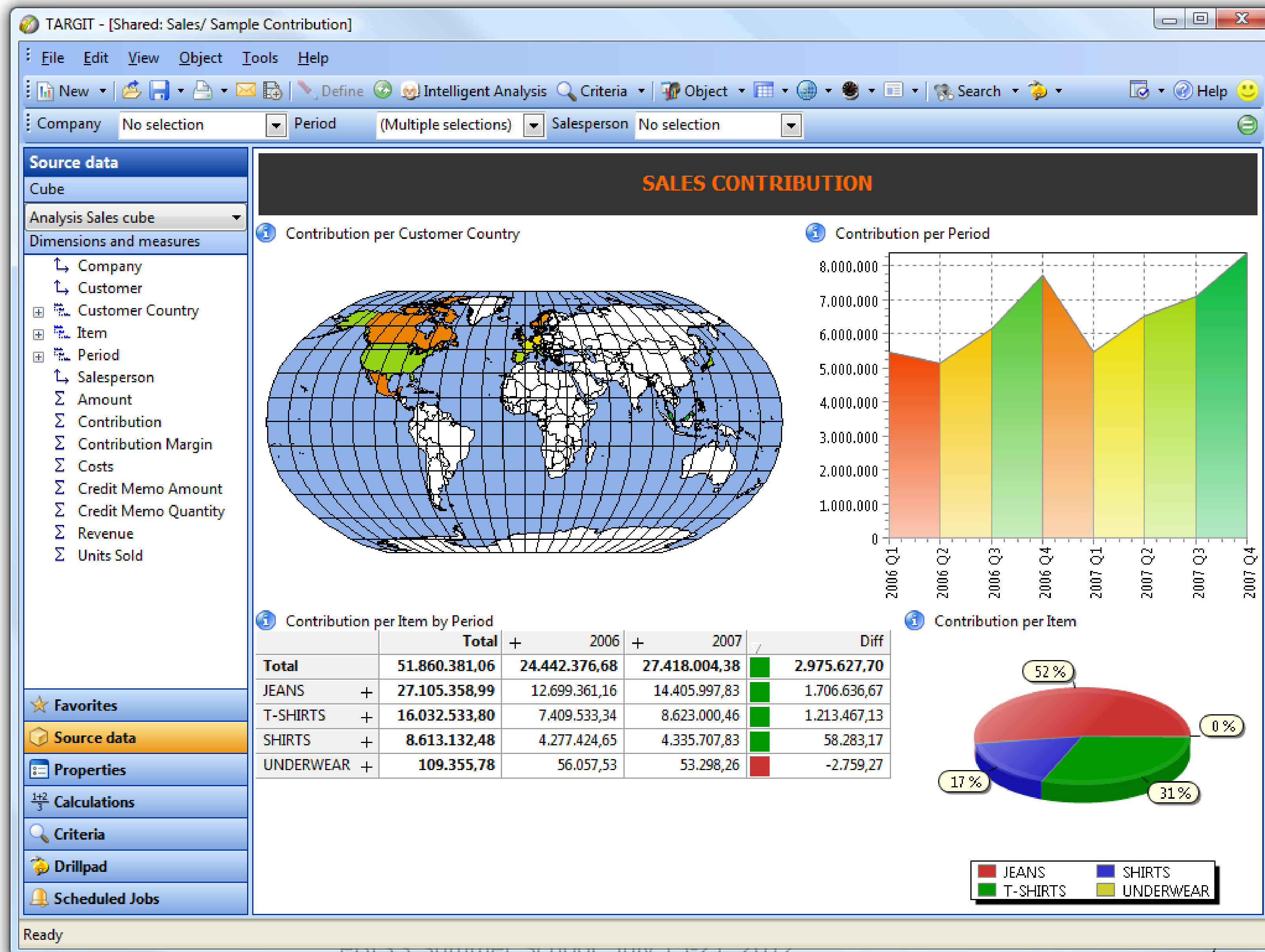
Why a new (MD) model?

- We know E/R and OO modeling
- All types of data are “equal”
- E/R and OO models: many purposes
  - ***Flexible***
  - ***General***
- No difference between:
  - What **is** important
  - What just **describes** the important
- ER/OO models are **large**
  - 50-1000 entities/relations/classes
  - Hard to get an overview
- ER/OO models implemented in RDBMSes
  - Normalized databases **spread** information
  - When analyzing data, the information must be **integrated** again

The multidimensional model

- One purpose
  - **Data analysis**
- Better at **that** purpose
  - Less flexible
  - Not suited for OLTP systems
- More **built in** “meaning”
  - What **is** important
  - What **describes** the important
  - What we want to **optimize**
  - Automatic aggregations means easy querying
- Recognized by OLAP/BI tools
  - Tools offer powerful query facilities based on MD design
  - Examples: TARGIT, Qlikview, BO,…

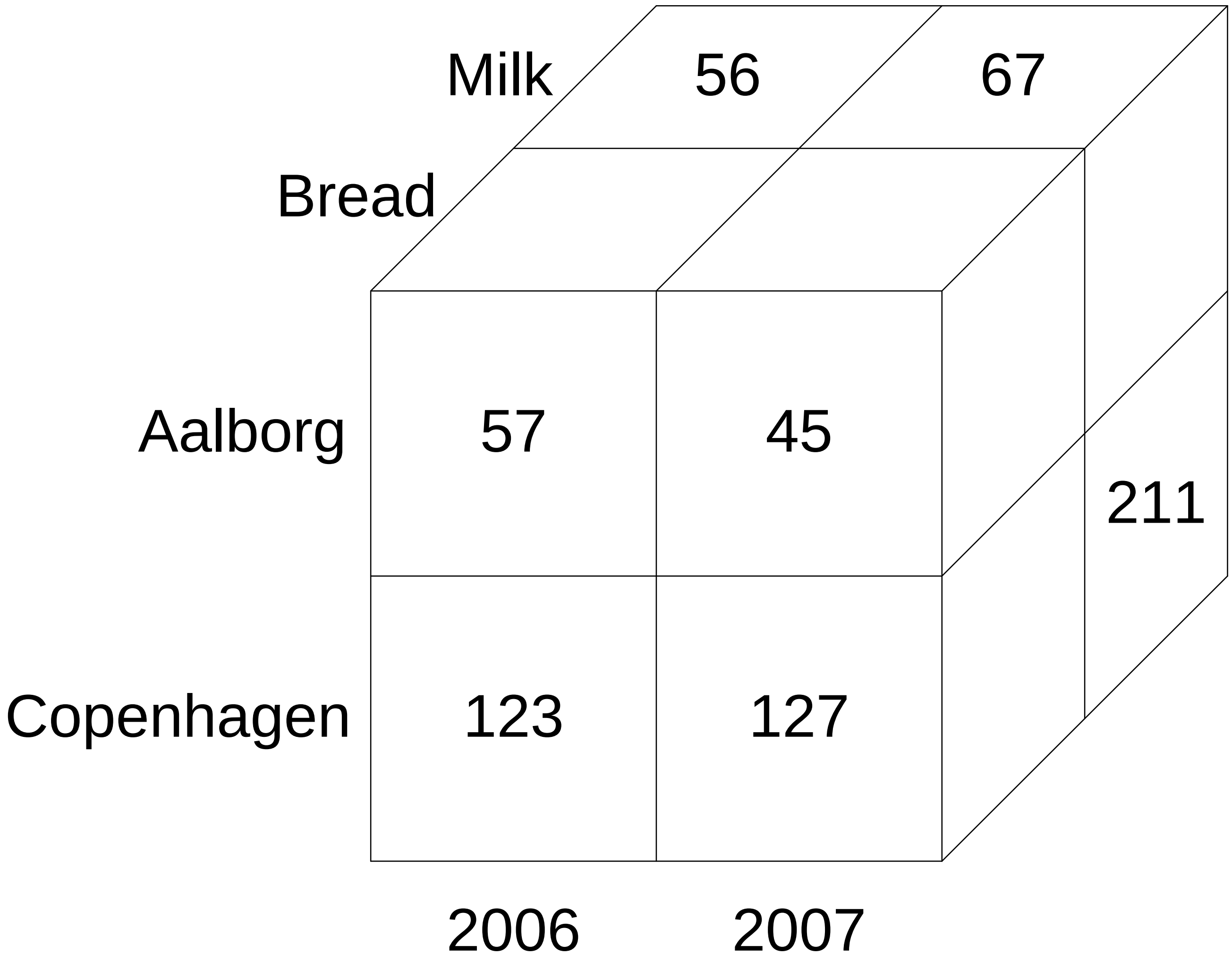
Example tool: TARGIT BI Suite



The multidimensional model

- Data is divided into:
  - **Facts**
  - **Dimensions**
- Facts are the **important** entity: a sale
- Facts have **measures** that can be aggregated: sales price
- Dimensions **describe** facts
  - A sale has the dimensions Product, Store and Time
- Facts “live” in a multidimensional **cube** (dice)
  - Think of an array from programming languages
- Goal for dimensional modeling:
  - Surround facts with as much context (dimensions) as possible
  - Hint: redundancy may be ok (in well-chosen places)
  - But you should **not** try to model **all** relationships in the data (unlike E/R and OO modeling!)

Cube Example



Cubes

- A "cube" may have **many** dimensions!
  - More than 3 - the term "hypercube" is sometimes used
  - Theoretically no limit for the number of dimensions
  - Typical cubes have 4-12 dimensions
- But only 2-3 dimensions can be viewed at a time
  - Dimensionality reduced by queries via projection/aggregation
- A cube consists of **cells**
  - A given combination of dimension values
  - A cell can be empty (no data for this combination)
  - A **sparse** cube has few non-empty cells
  - A **dense** cube has many non-empty cells
  - Cubes become sparser for many/large dimensions

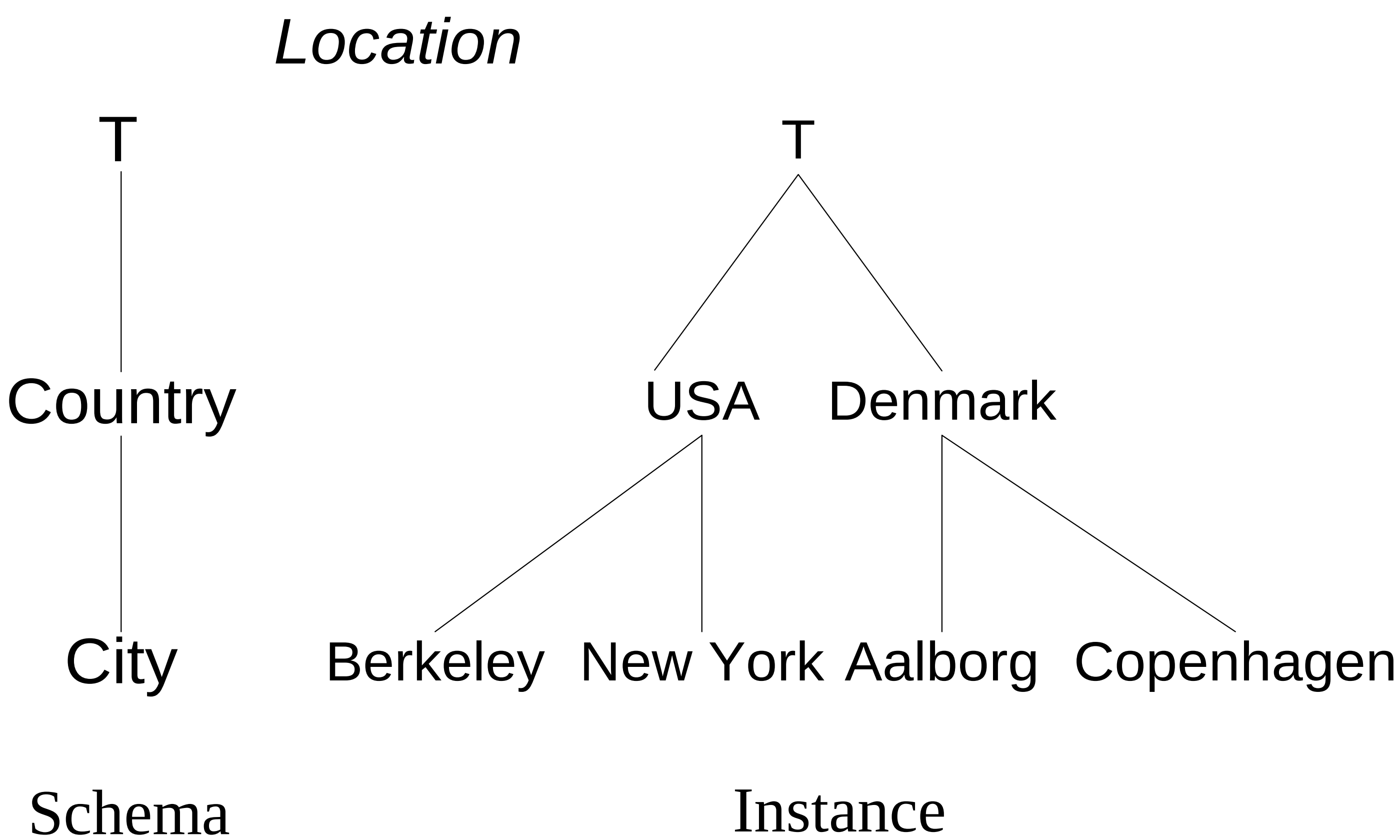
Dimensions

- Dimensions are the core of multidimensional databases
  - Other types of databases do not support dimensions
- Dimensions are used for
  - **Selection** of data
  - **Grouping** of data at the right level of detail
- Dimensions consist of **dimension values**
  - Product dimension have values ”milk”, ”cream”, …
  - Time dimension have values ”1/1/2001”, ”2/1/2001”,…

Dimensions

- Dimensions have **hierarchies** with **levels**
  - Typically 3-5 levels (of detail)
  - Dimension values are organized in a **tree structure**
  - **Product**: Product->Type->Category
  - **Store**: Store->Area->City->County
  - **Time**: Day->Month->Quarter->Year
  - Dimensions have a **bottom level** and a **top level** (ALL)
- Levels may have **attributes**
  - Simple, non-hierarchical information
  - Day has Workday as attribute
- Dimensions should contain much information
  - Time dimensions may contain holiday, season, events,…
  - Good dimensions have 50-100 or more attributes/levels

Dimension Example



Facts

- Facts represent the **subject** of the desired analysis
  - The ”important” in the business that should be analyzed
- A fact is most often identified via its dimension values
  - A fact is a non-empty cell

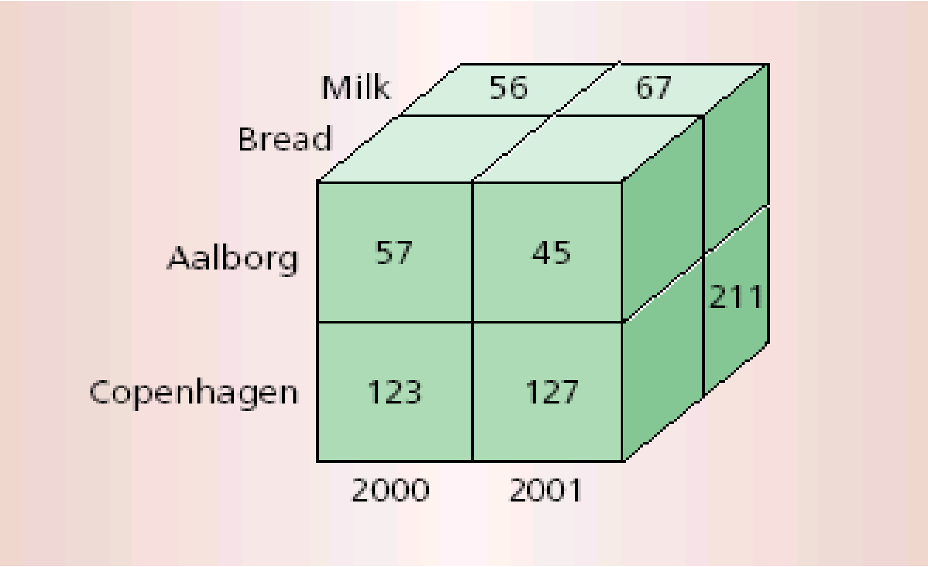
Measures

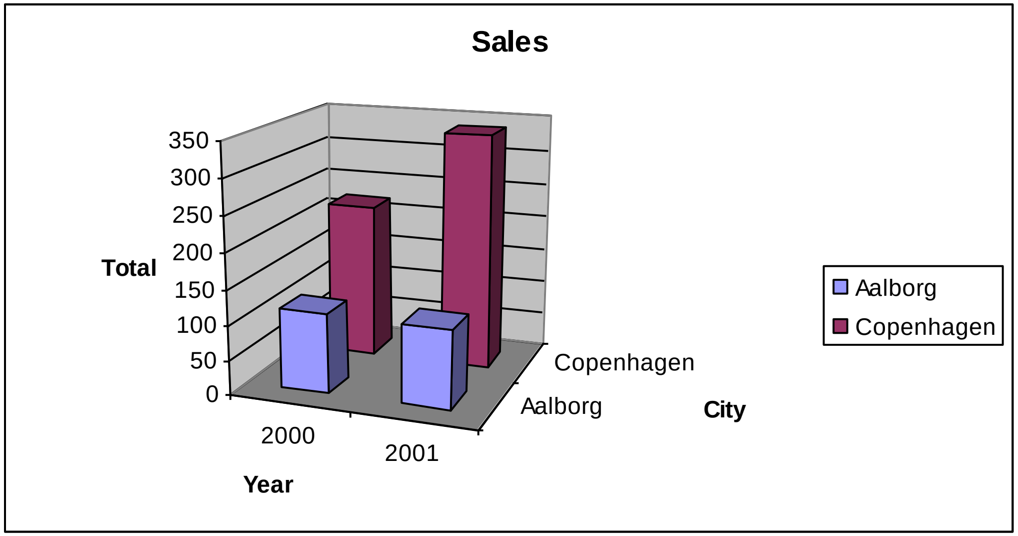
- Measures represent the fact property that the users want to **study and optimize**
  - Example: total sales price
- A measure has two components
  - **Numerical value**: (sales price)
  - **Aggregation formula** (SUM): used for aggregating/combining a number of measure values into one

  - Measure value determined by dimension value combination
  - Measure value is meaningful for all aggregation levels

On-Line Analytical Processing

- Fast, interactive analysis of large amounts of data
  - Sales, web, …
- "Spreadsheets on stereoids"
- Aggregation queries
  - Per City and Year
- Roll up - get overview
- Drill down – more detail
- Fast answers required
  - A few seconds response time even for many gigabytes of data
  - Achieved by pre-computation (pre-aggregation)



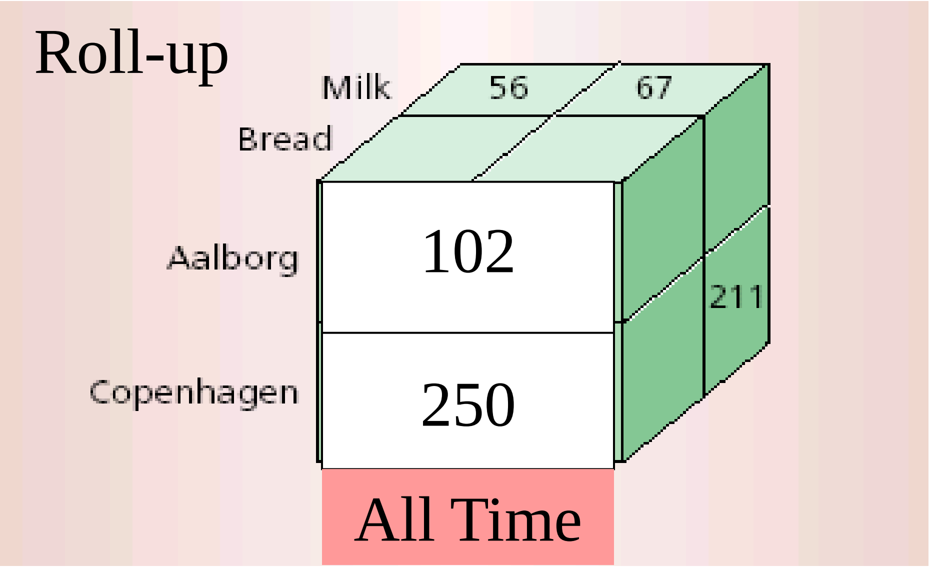


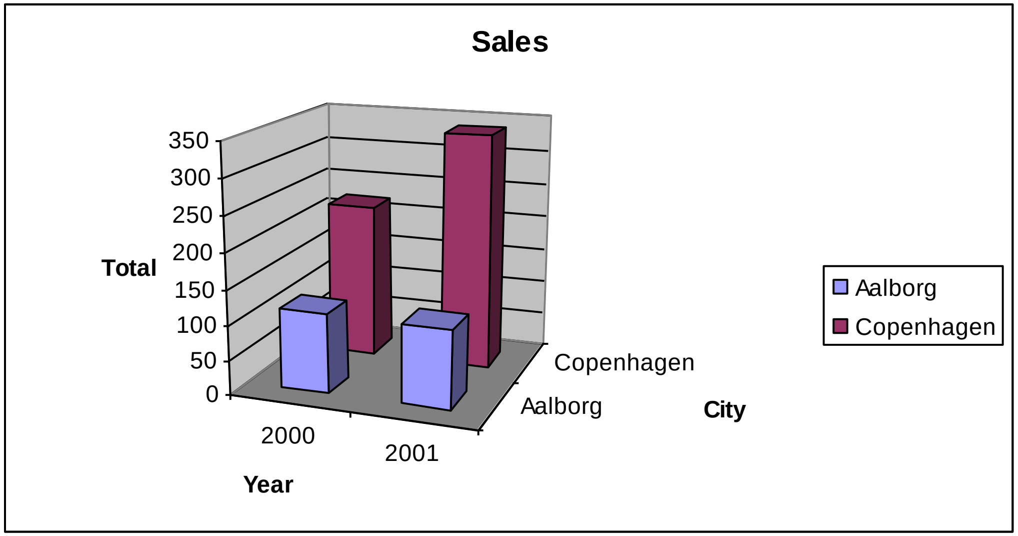
On-Line Analytical Processing

- Fast, interactive analysis of large amounts of data
  - Sales, web, …
- “Spreadsheets on stereoids”

- Aggregation queries
  - Per City and Year
- Roll up - get overview
- Drill down – more detail

- Fast answers required
  - A few seconds response time even for many gigabytes data
  - Achieved by pre-computation (pre-aggregation)



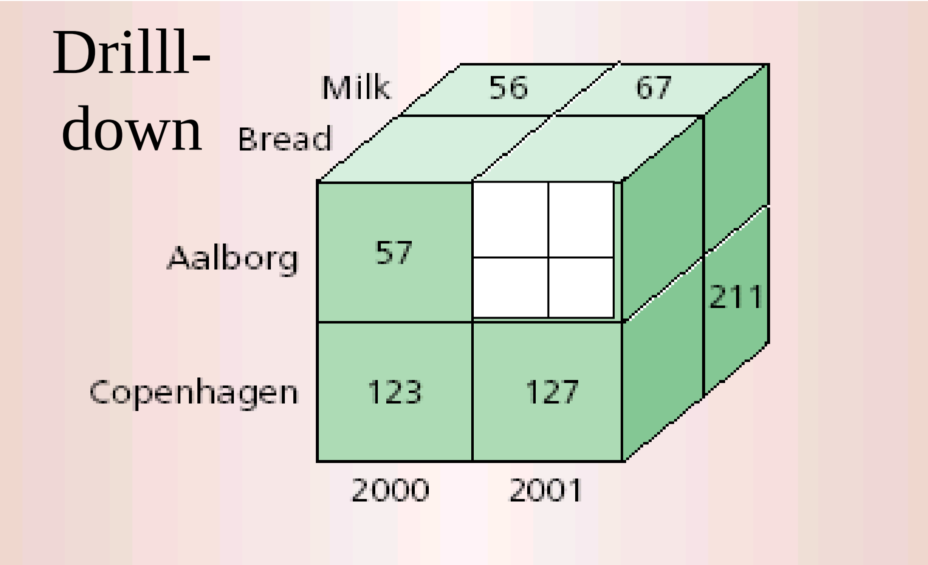


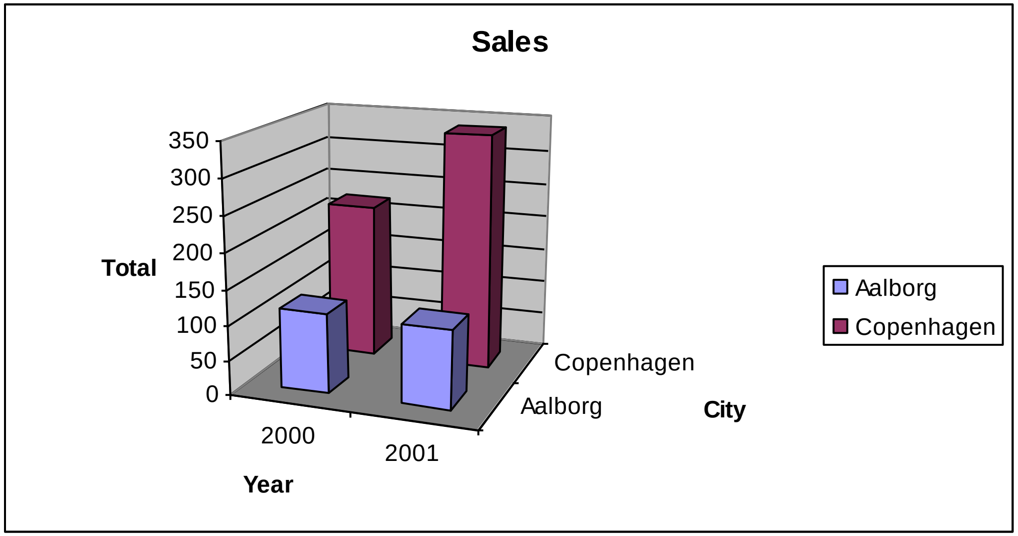
On-Line Analytical Processing

- Fast, interactive analysis of large amounts of data
  - Sales, web, …
- “Spreadsheets on stereoids”

- Aggregation queries
  - Per City and Year
- Roll up - get overview
- Drill down – more detail

- Fast answers required
  - A few seconds response time even for terabytes of data
  - Achieved by pre-computation (pre-aggregation)





Advanced Multidimensional Modeling

- Changing dimensions
  - Some dimensions are not static. They change over time.
    - A store moves to a new location with more space
    - The name of a product changes
    - A customer moves from Aalborg Øst to Hasseris
  - How do we handle these changes?
- Large-scale dimensional modeling
  - How do we coordinate the dimensions in different data cubes and data marts?

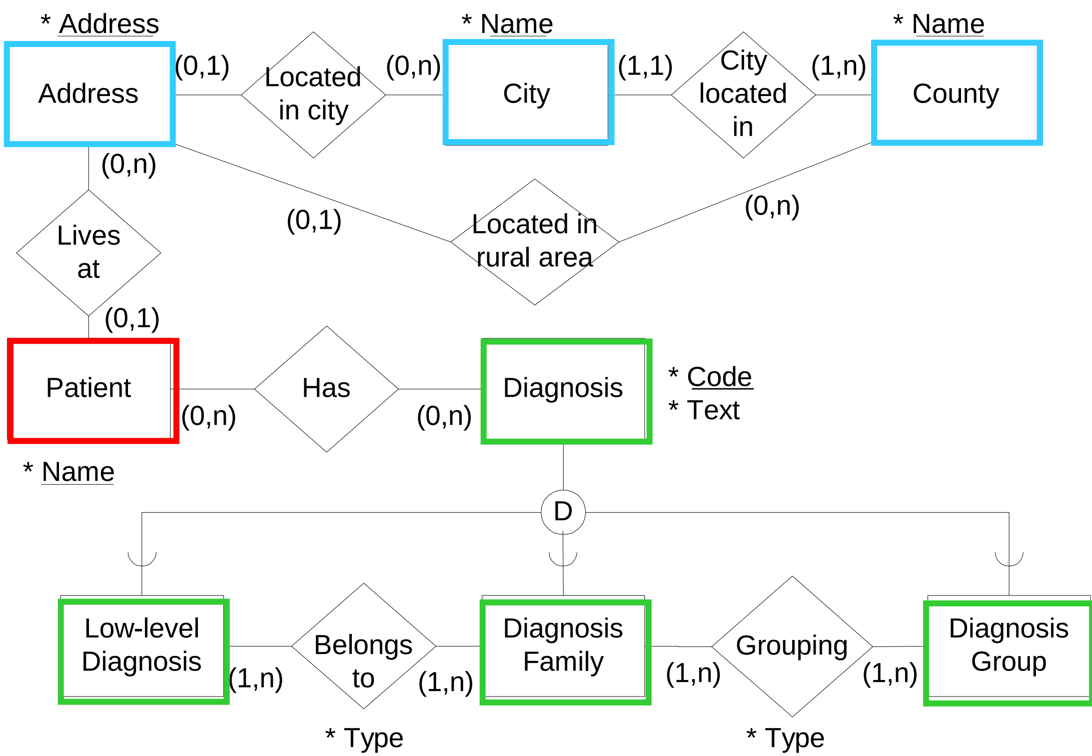
Bus architecture

Dimensions

Data marts	Time	Customer	Product	Supplier
Sales	+	+	+	
Costs			+	+
Profit	+	+	+	+

Patient Case Study

- Patients-diagnoses-addresses
- Patients are the *facts* in multidimensional terms.
- Each patient has zero or more diagnoses (many-to-many).
- The diagnoses may be registered at any level (Low-level, Family, or Group), i.e., the granularity is *varying*.
- A diagnosis may have no children (non-onto) or several parents (non-strict).
- Each patient has one address, in a city or a rural area (non-covering). Cities are located in counties.

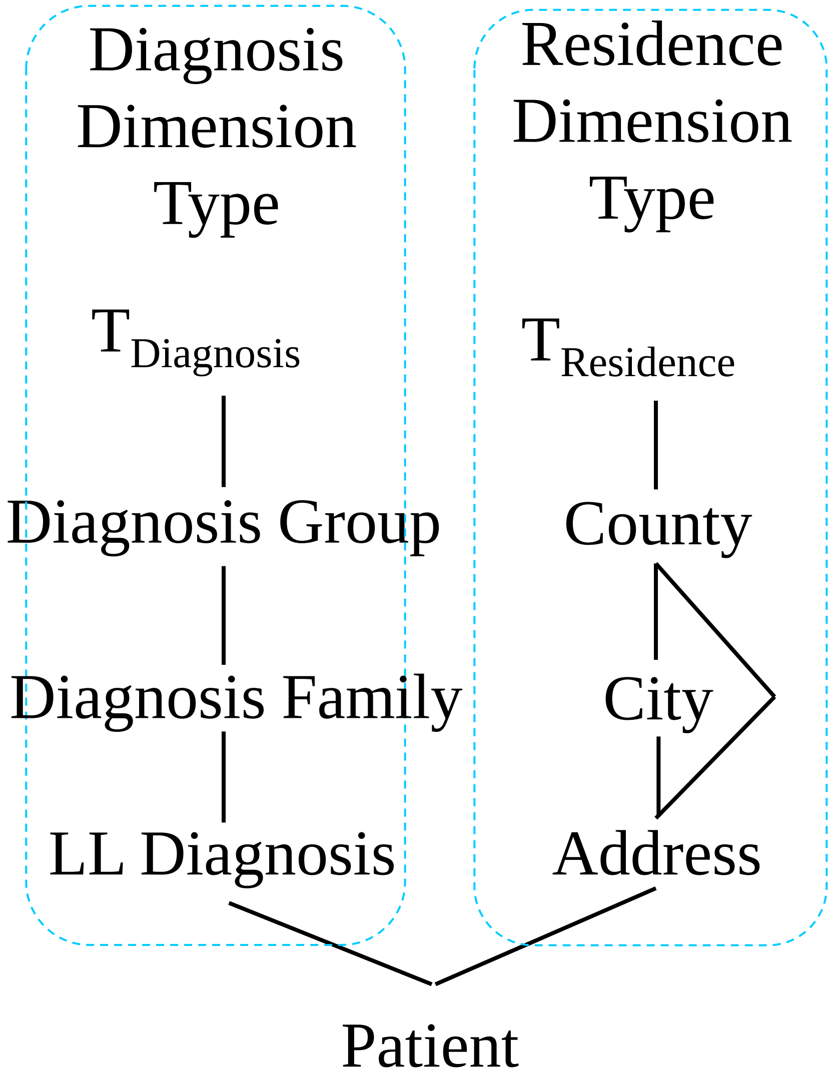


Patient Case Study

- There are other complexities in the case study.
- Data is imprecise
  - The patients' HbA1c% (long term blood sugar) is measured
    - Different methods used for measuring
    - Some data is missing
    - => Varying degree of precision
  - Diagnoses for patients are at different levels
    - Some diagnoses are precise (Diabetes Type 1 during pregnancy)
    - Some diagnoses are imprecise (Diabetes)
    - => Varying degree of precision
- Data change over time
  - Addresses change
  - Diagnoses change
  - HbA1c% change

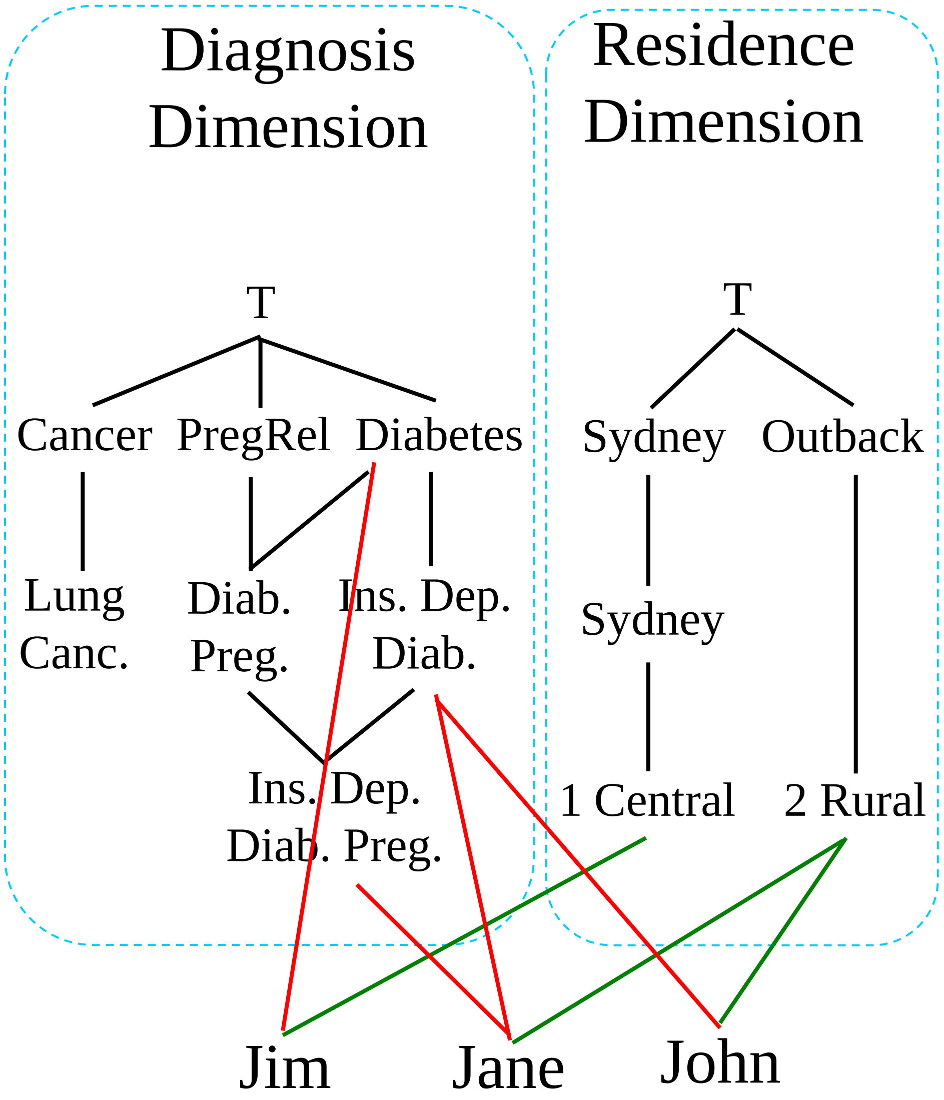
Data Model - Schema

- Fact type: Patient
- Dimension types: Diagnosis and Residence
- There are no “measures”, all data are dimensions.
- Category types: Low-level Diagnosis, Diagnosis Family, Diagnosis Group, Address, City, County
- Top category types: corresponds to ALL of the dimension.
- Bottom category types: the lowest level in each dimension
- The category types of a dimension type form a lattice.



Data Model - Instances

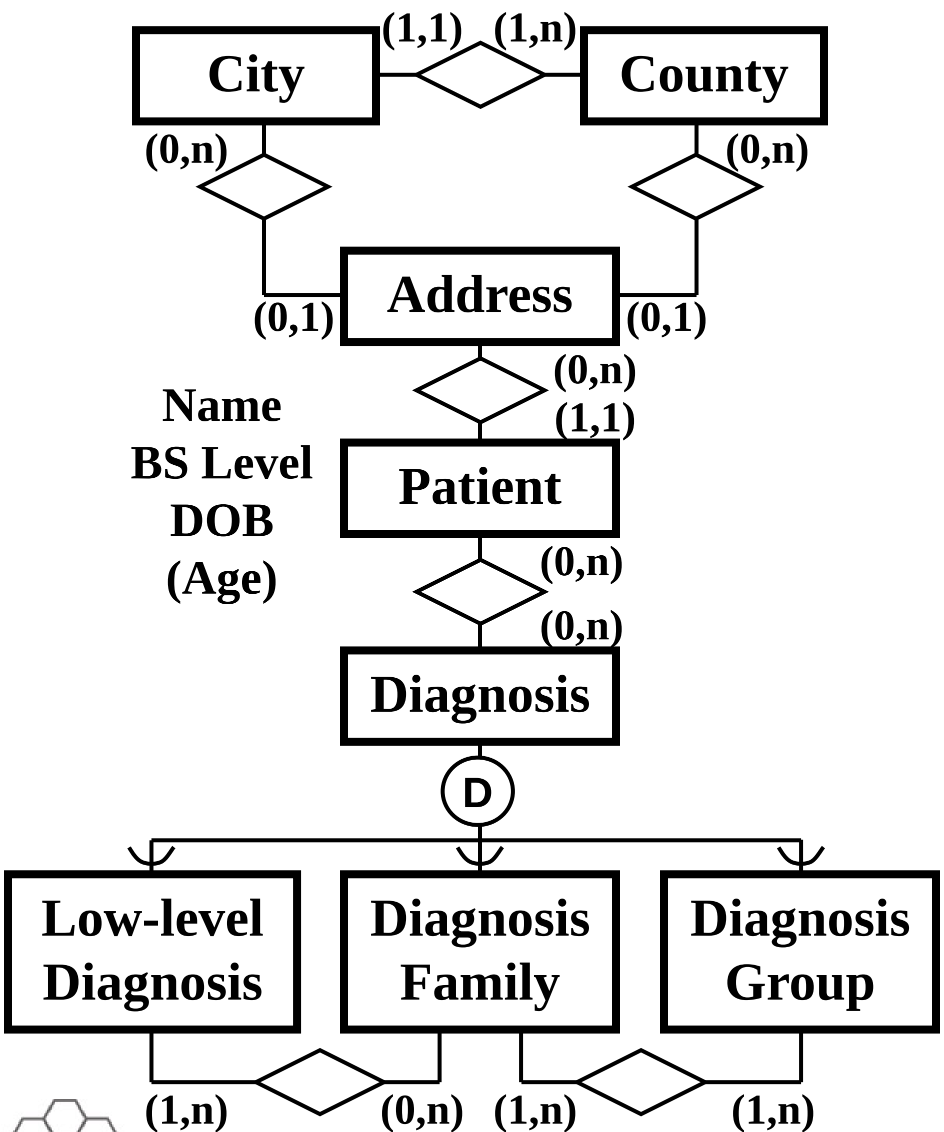
- Categories: instances of category types, consist of dimension values.
- Top categories contain only one "T" value.
- Dimensions = categories + partial order on category values, hierarchy may be non-strict.
- Facts: instances of fact type with separate identity.
- Fact-dimension relations: links facts to dimensions, may map to *several* values of *any* granularity in each dimension.
- Multidimensional object (MO) = schema + dimension + facts + fact-dimension relations



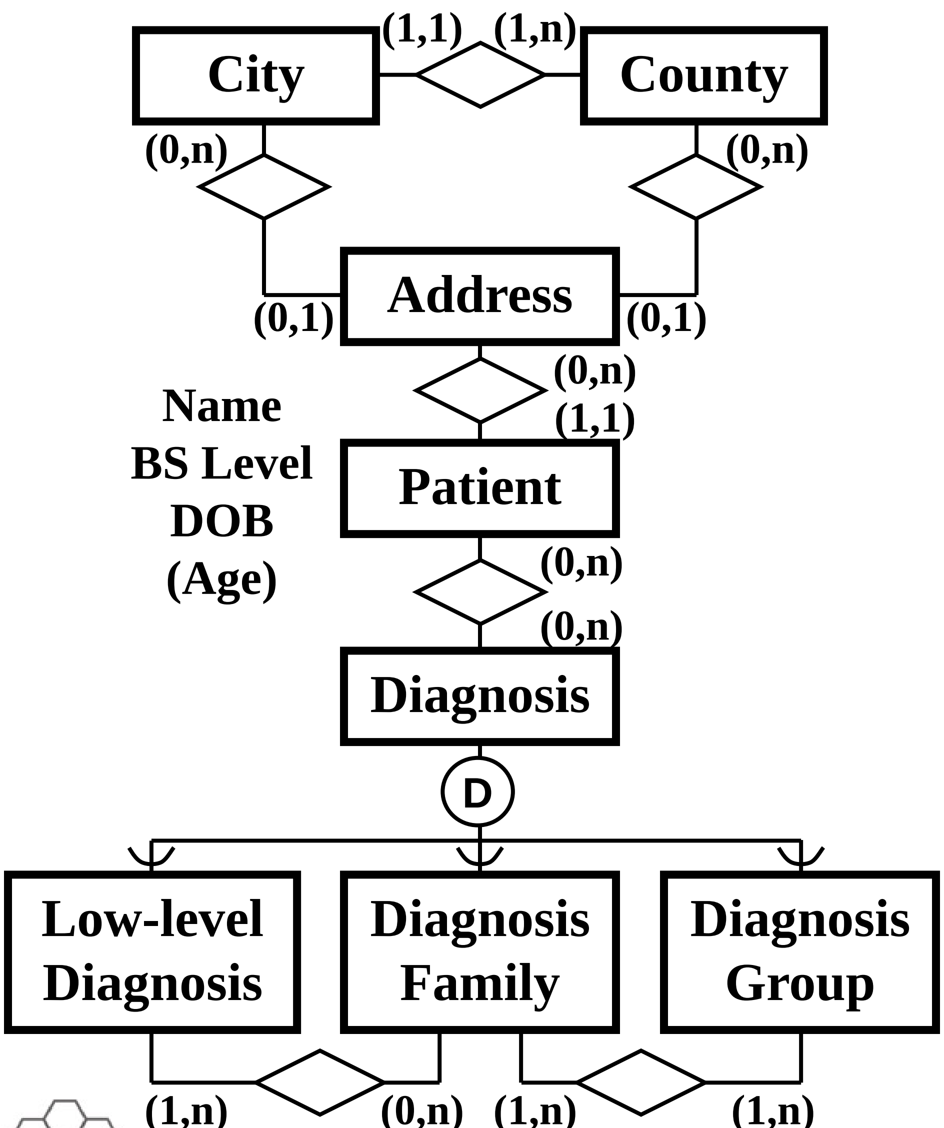
Imprecise Data

- Imprecision is handled using the *granularity* of data
- Separate “Precision MO” captures data granularity
  - Answers the question “is data precise enough for this query?”
  - Used for suggesting *alternative queries* that can be answered precisely
- Imprecision in grouping, e.g., imprecise diagnoses:
  - *Conservative grouping* - only include what’s known to belong
  - *Liberal grouping* - include everything that might belong
  - *Weighted grouping* - include everything, weighted by probability
- Imprecision in computation, e.g., imprecise HbA1c%:
  - Substitution of *expected values* during computation
  - Separate *precision computation* carried out concurrently
- The result *and its precision* is presented to the user
  - Graphical representation, etc.

Requirements

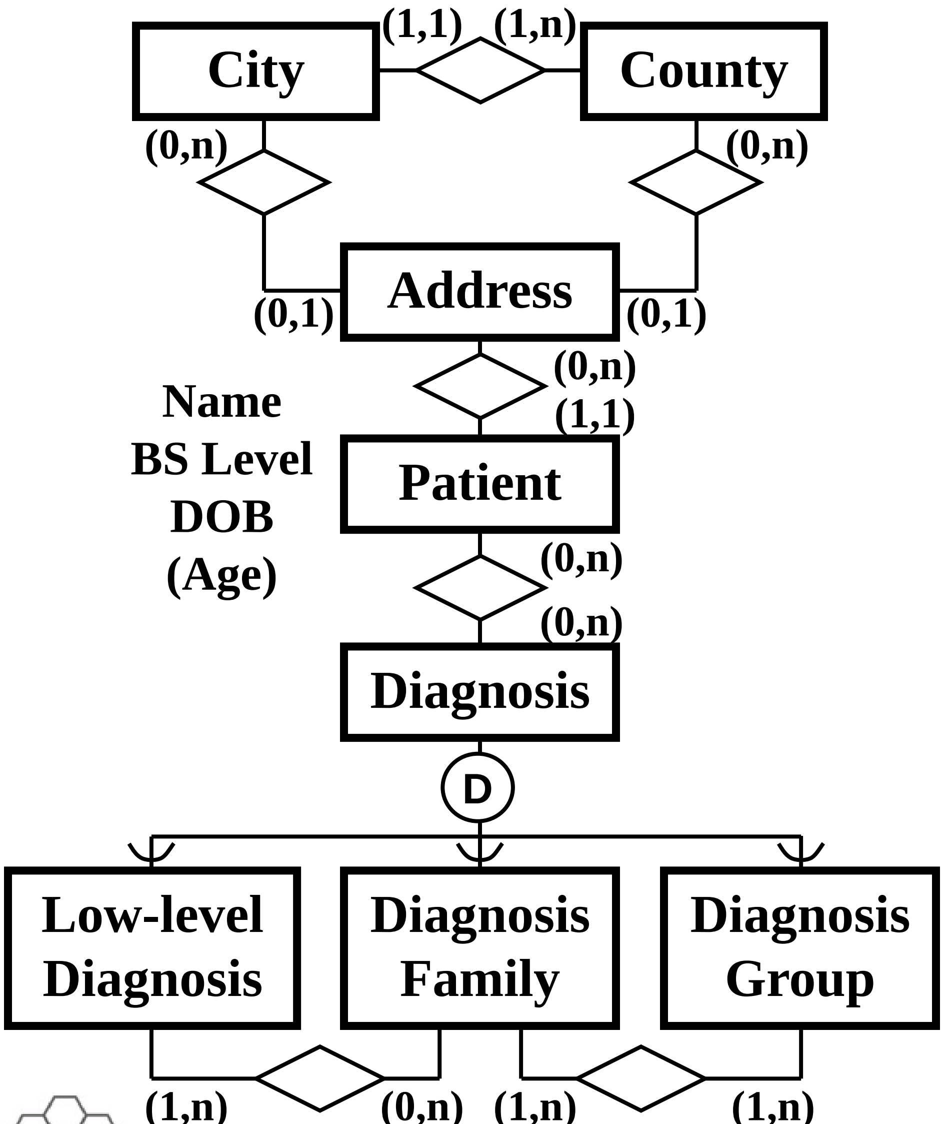


Requirements



- **Explicit hierarchies**

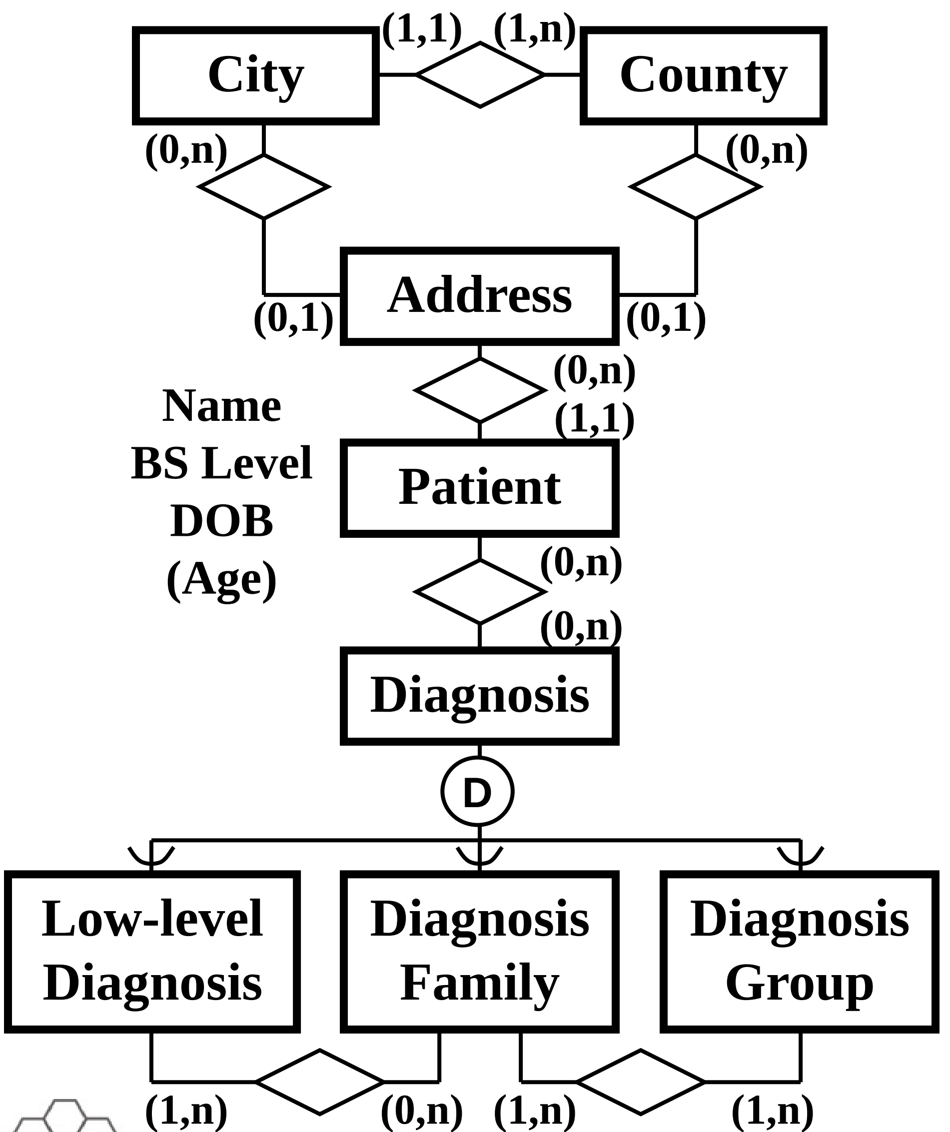
Requirements

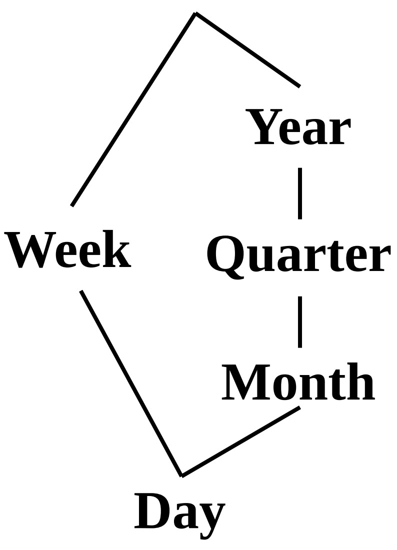


- Explicit hierarchies
- **Dimensions = measures**

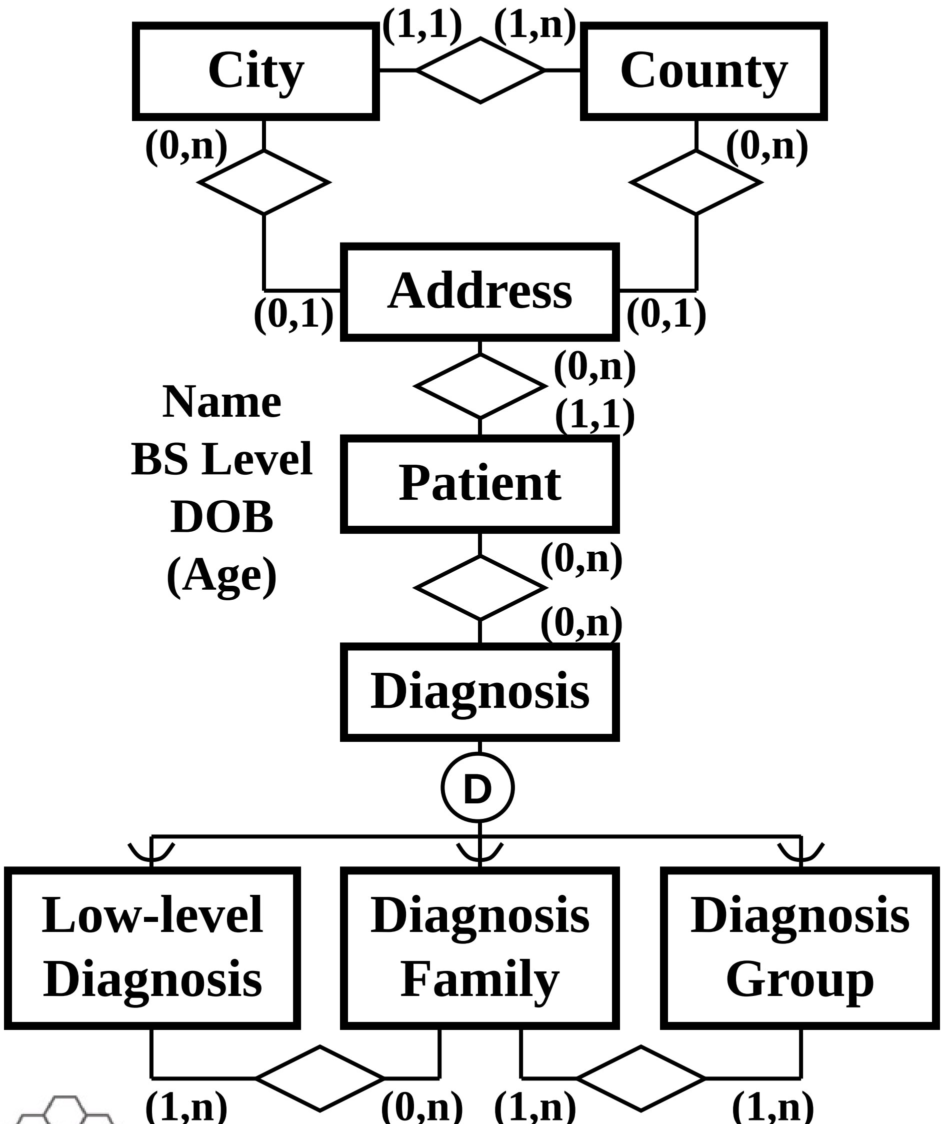
Requirements

- Explicit hierarchies
- Dimensions = measures
- **Multiple hierarchies**



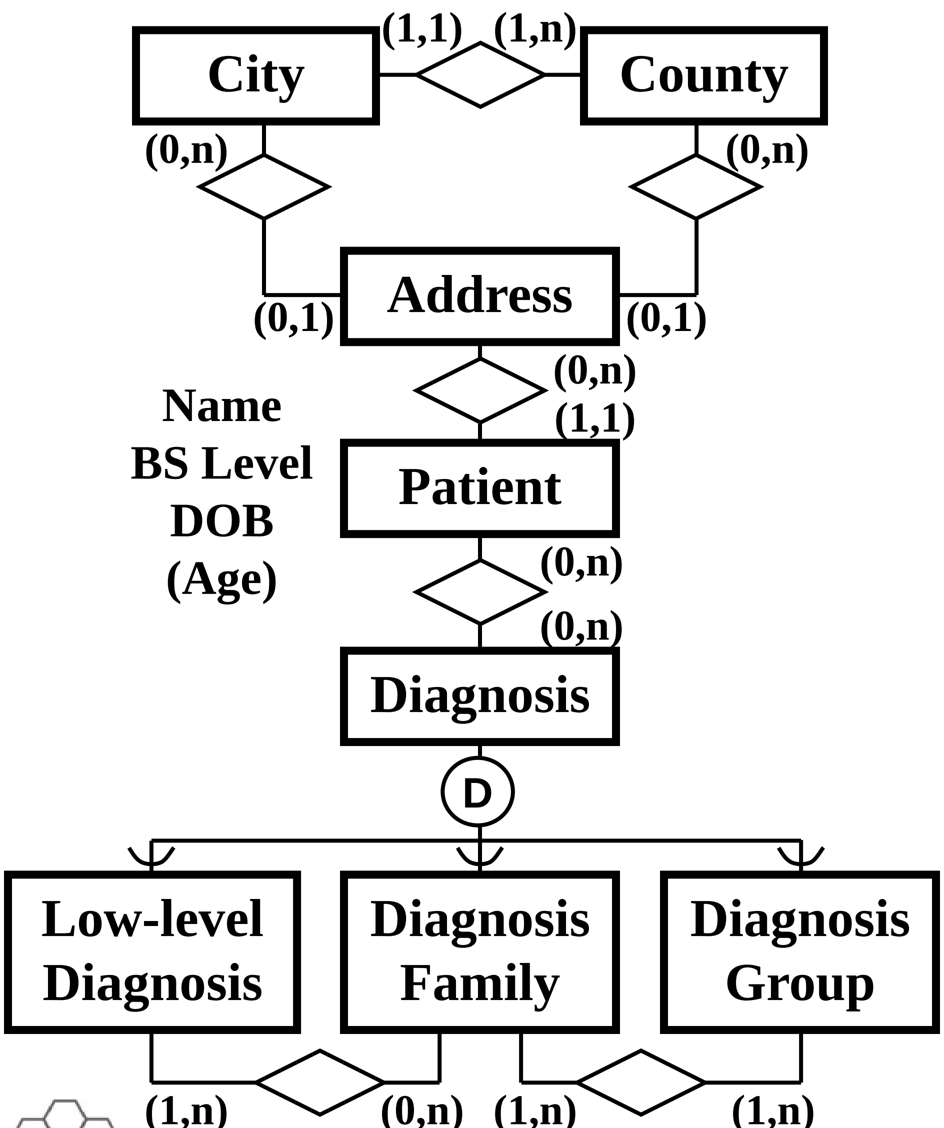


Requirements



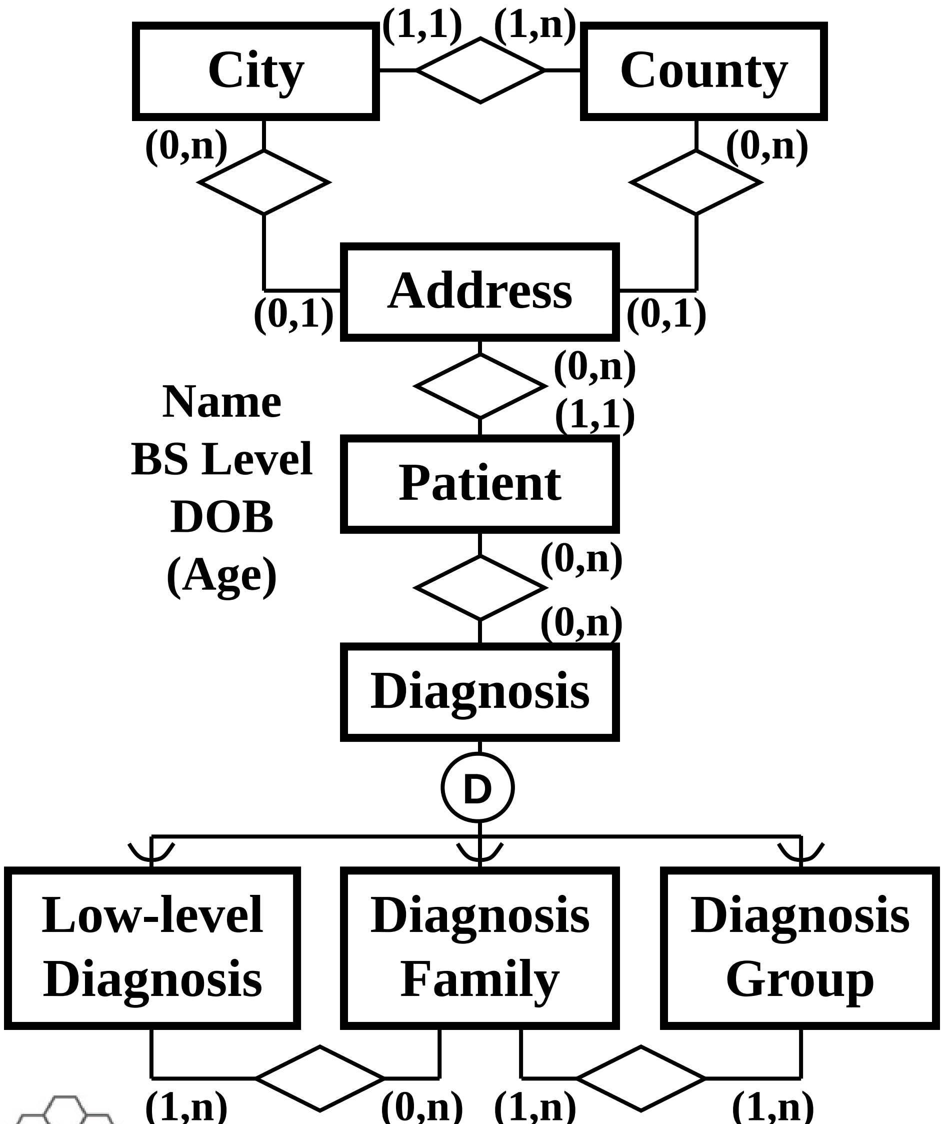
- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- **Aggregation semantics**
  - **Double-counting**
  - **Valid aggregations**

Requirements



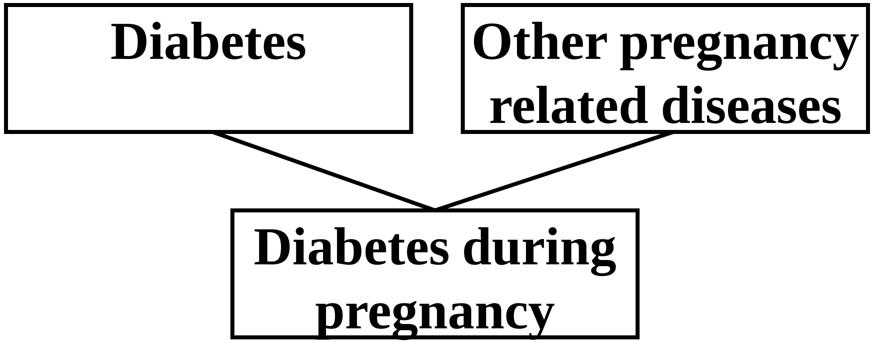
- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- **Facts/dimensions: n-n**

Requirements



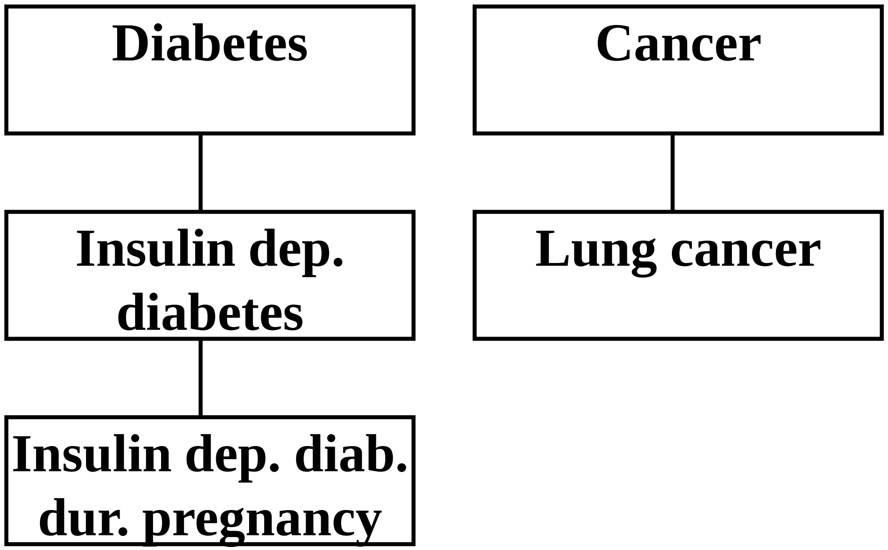
- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- **Different granularities**

Requirements



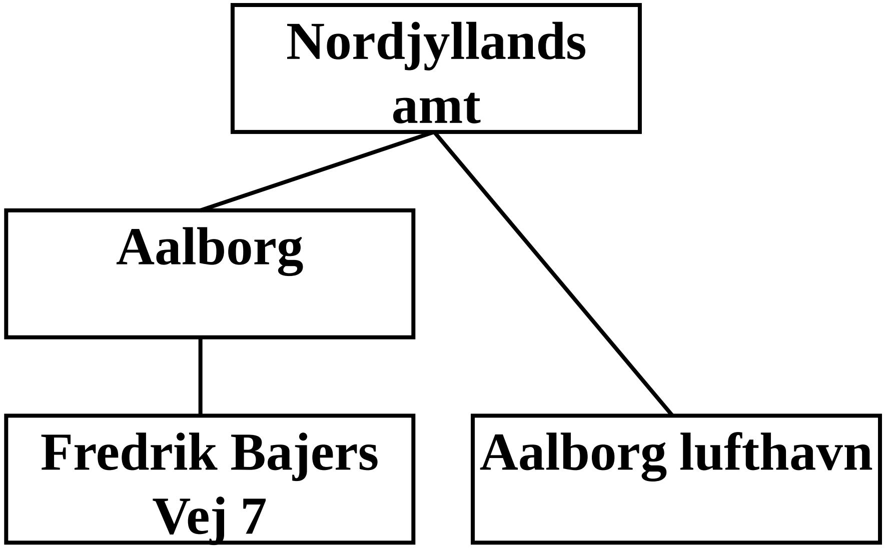
- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- **Non-strict hierarchies**

Requirements



- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- **Non-onto hierarchies**

Requirements



- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- Non-onto hierarchies
- **Non-covering hierarchies**

Requirements

- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- Non-onto hierarchies
- Non-covering hierarchies

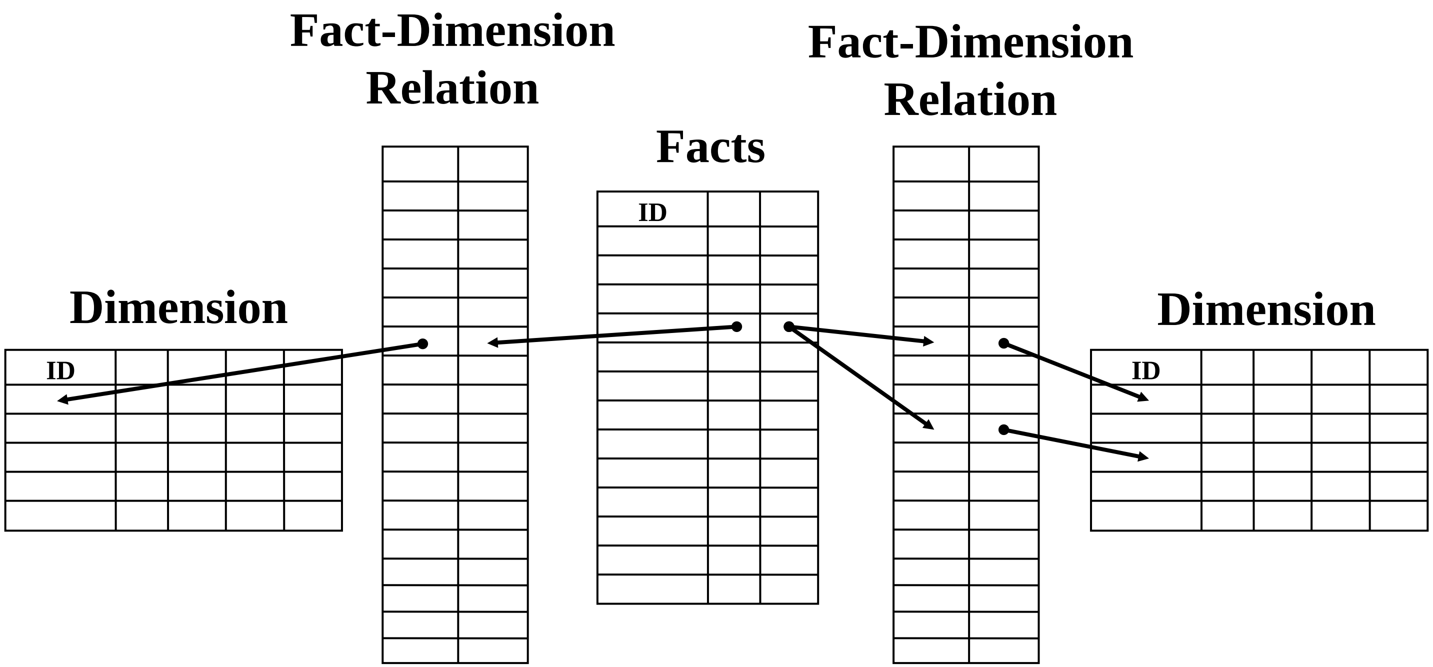
Only a few are supported in existing models

The Model

Multidimensional Object:

$$M = (\mathcal{S}, F, D, R)$$

Schema **Facts** **Dimensions** **Fact-Dimension Relations**

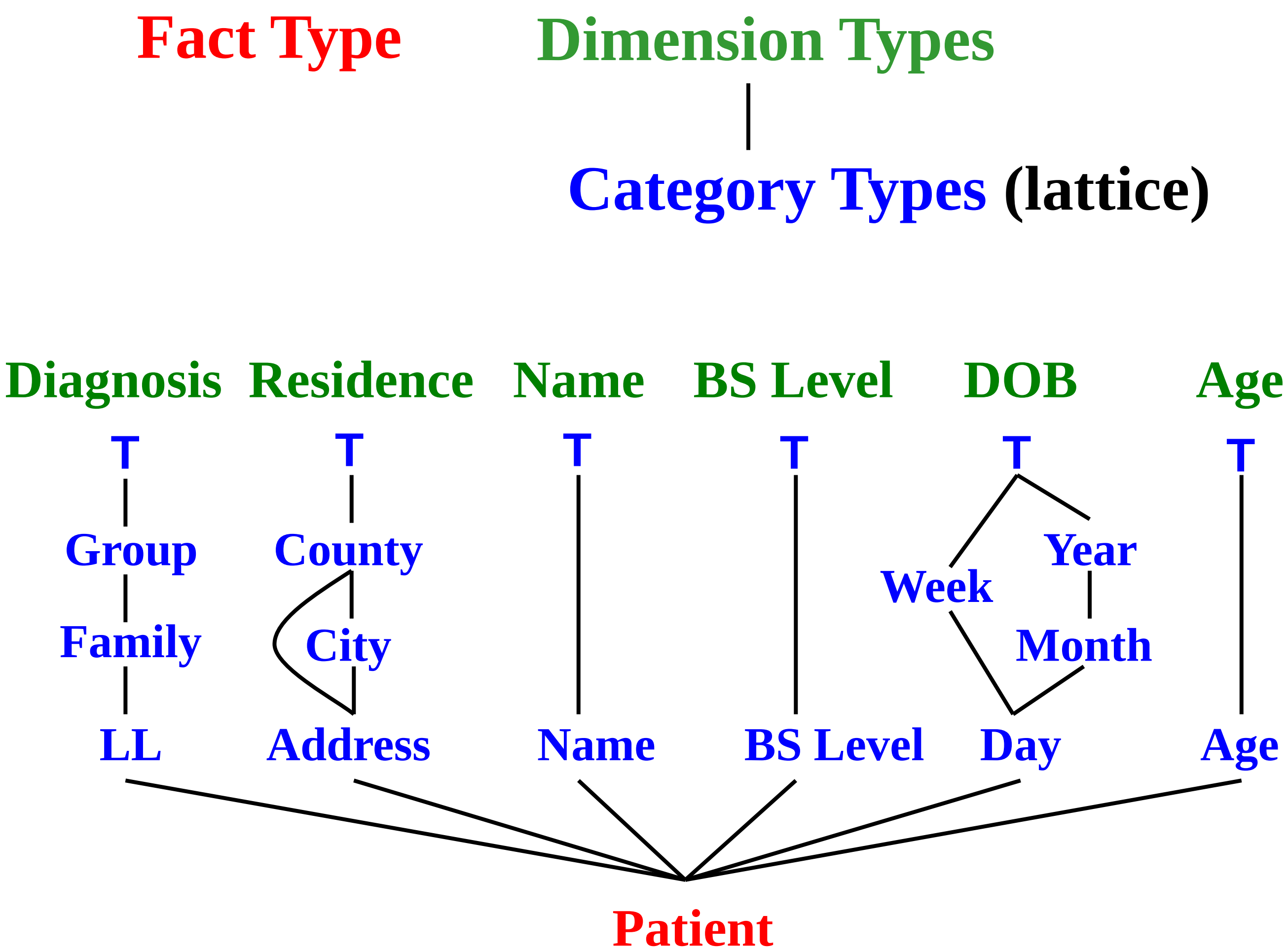


Requirements

- Explicit hierarchies
- **Dimensions = measures**
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- Non-onto hierarchies
- Non-covering hierarchies

The Model

Schema: $S = (F, D)$



Requirements

- **Explicit hierarchies**
- Dimensions = measures
- **Multiple hierarchies**
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- Non-onto hierarchies
- Non-covering hierarchies

The Model

Facts F

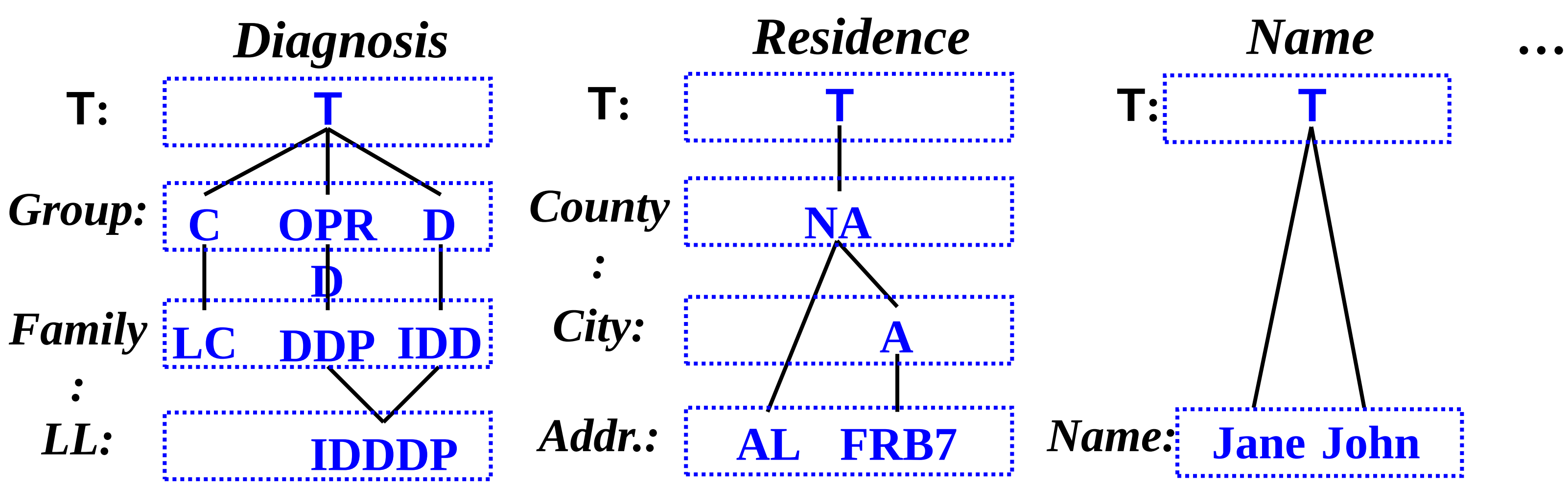
Fact

Dimensions D

Categories

Dimension values

(*partially ordered*)



Patient: **1** **2**

Requirements

- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- Facts/dimensions: n-n
- Different granularities
- **Non-strict hierarchies**
- **Non-onto hierarchies**
- **Non-covering hierarchies**

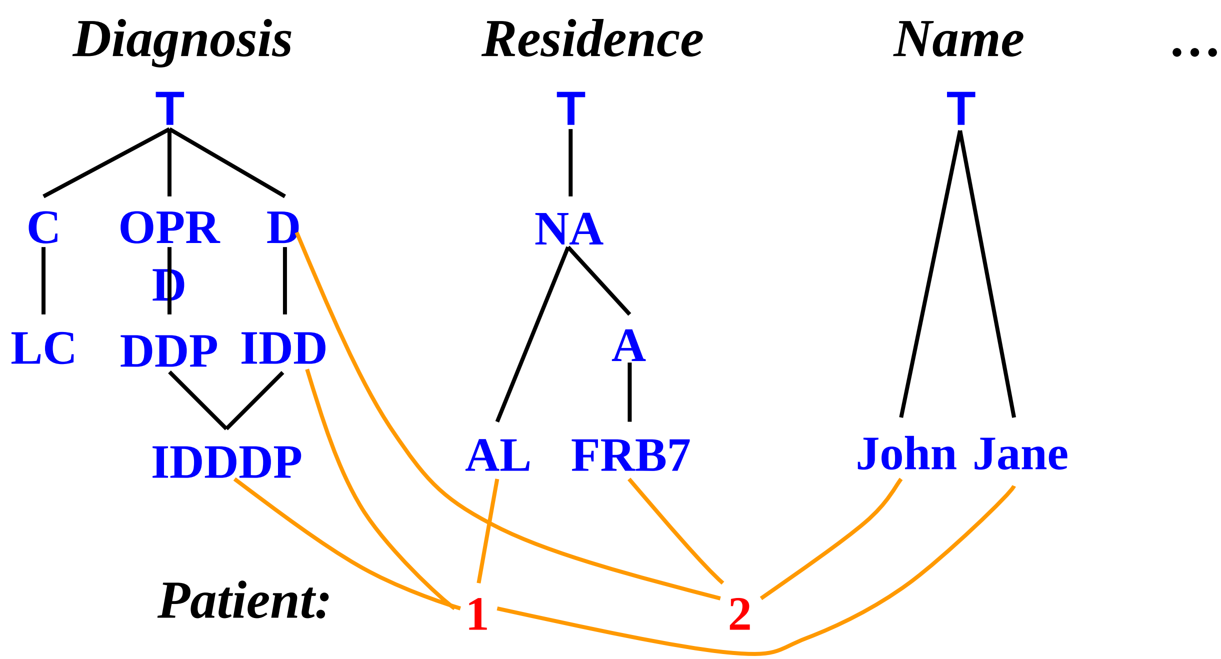
The Model

Fact-dimension relations

| R

Fact-dimension relation:

- **Fact**
- **Dimension value (any category)**



Requirements

- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- Aggregation semantics
- **Facts/dimensions: n-n**
- **Different granularities**
- Non-strict hierarchies
- Non-onto hierarchies
- Non-covering hierarchies

The Model

Distinguish between 3 types of aggregate functions:

- Applicable to data that can be *added* (Σ)
  - sales amounts
- Applicable to data that can be *averaged,* but not added (ϕ)
  - date of birth
- Applicable to data that can only be *counted* (c)
  - diagnosis

Each category type is assigned one of these types

The Algebra

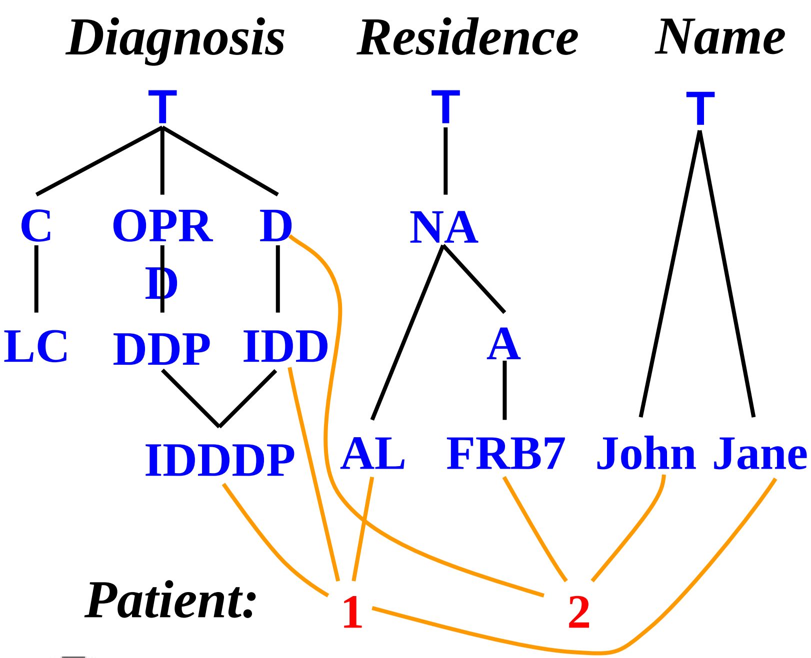
- Fundamental operators: σ, π, ρ, $\cup$, $\setminus$, , $\bowtie\alpha$
- Closed
- At least as strong as RA with aggregation

The Algebra

Aggregate operator:

$$\alpha[\textit{Result}, \text{set-count}, \{C, OPRD, D\}, \{T\}, \{T\}](MO_{input}) = MO_{output}$$

MO_{input} | **MO_{output}**

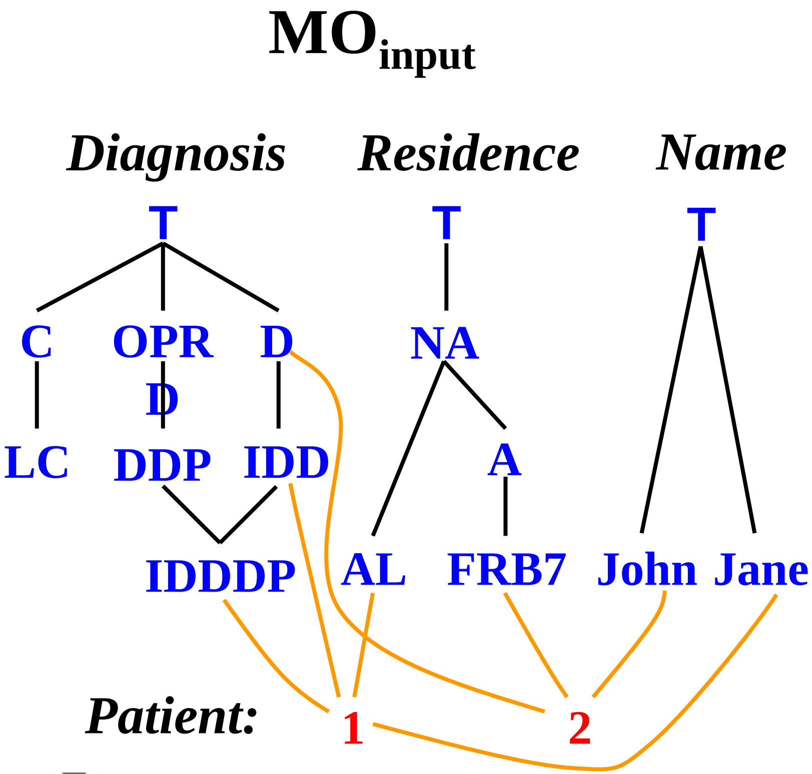


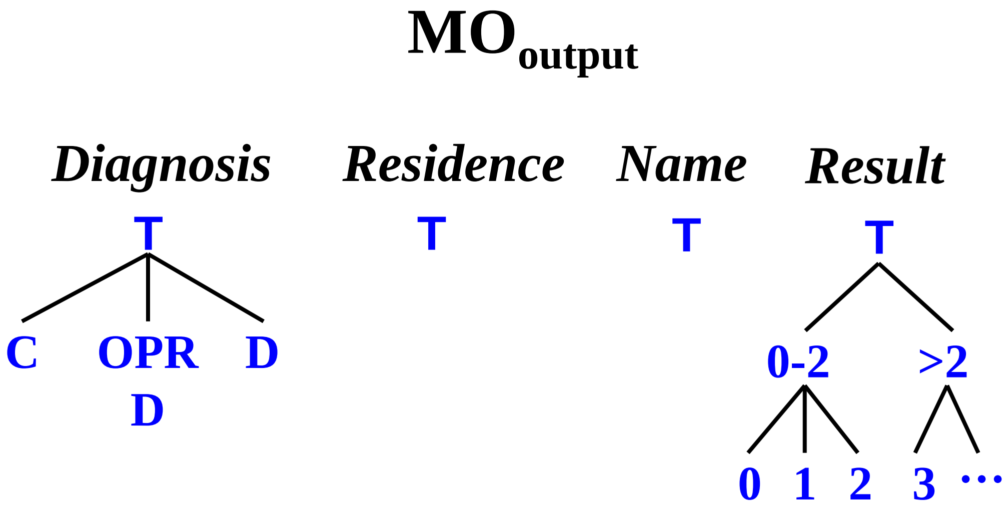
?

The Algebra

Aggregation:

$$\alpha[\textit{Result}, \text{set-count}, \{C, OPRD, D\}, \{T\}, \{T\}](MO_{input}) = MO_{output}$$

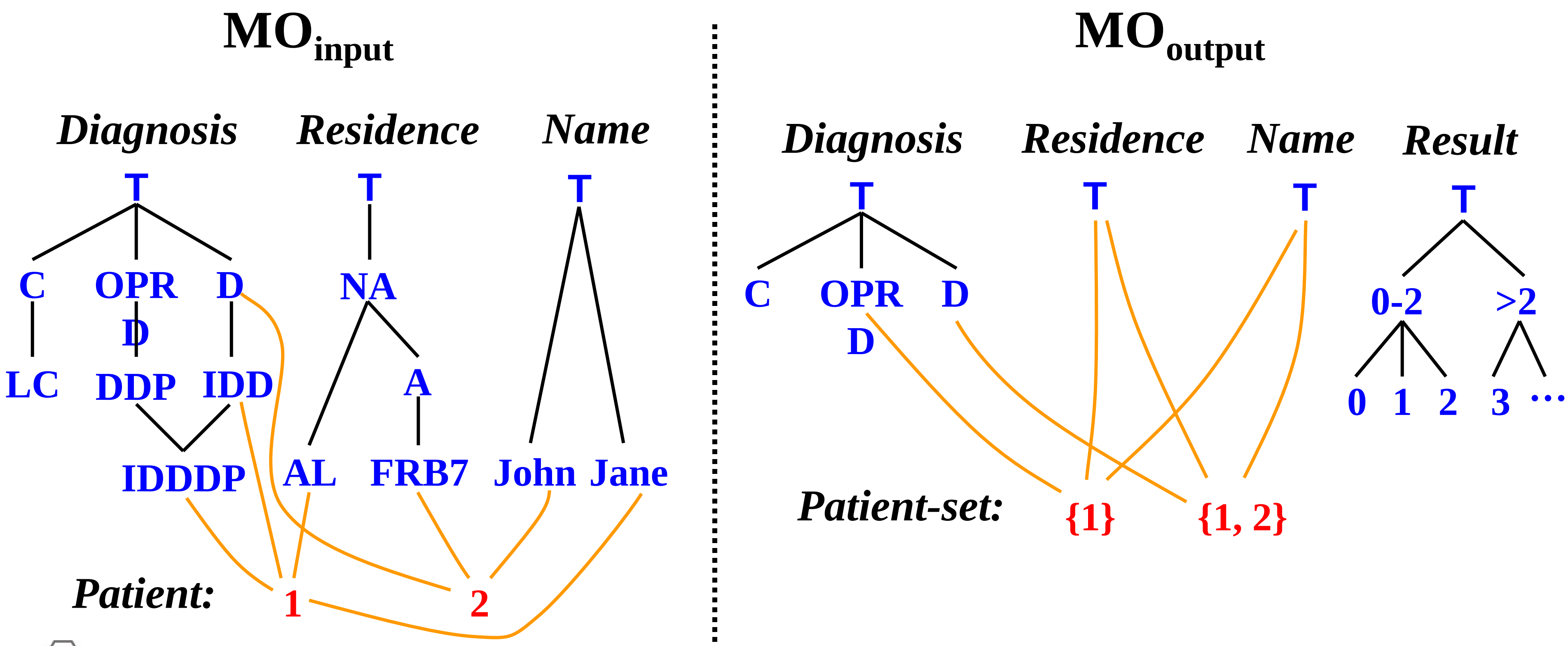




The Algebra

Aggregation:

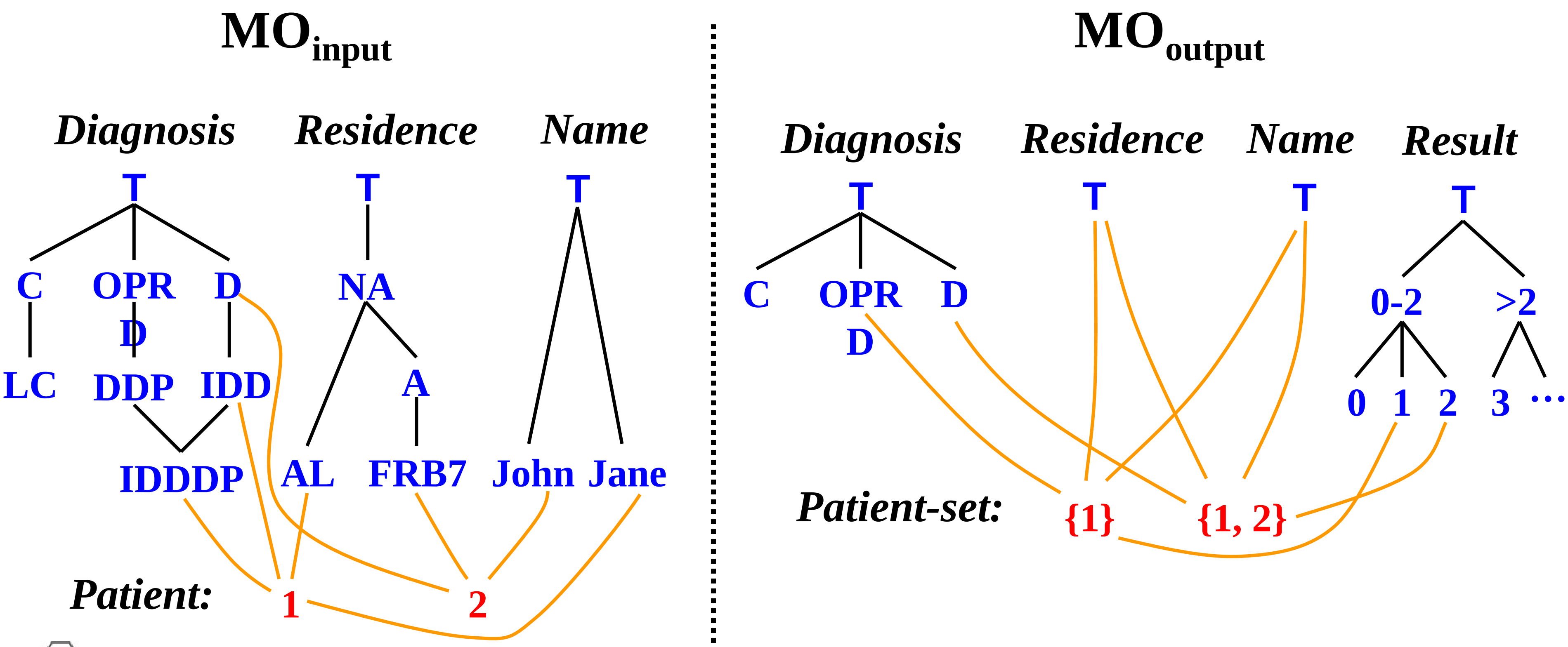
$$\alpha[\textit{Result}, \text{set-count}, \{C, OPRD, D\}, \{T\}, \{T\}](MO_{input}) = MO_{output}$$



The Algebra

Aggregation:

$$\alpha[\textit{Result}, \text{set-count}, \{C, OPRD, D\}, \{T\}, \{T\}](MO_{input}) = MO_{output}$$



The Algebra

Which aggregation type do we choose for the result?

Summarizable: The least capable from the input dimensions

	Dim. 1	*Dim. 2*		*Result*
	$\top$	$\top$		$\top$
	$\vert$	$\vert$		$\vert$
	$\vdots$	$\vdots$		$\vdots$
	$\vert$	$\vert$		$\vert$
	Sales amount	**Date of birth**		**Aggregation**
Aggtype:	Σ	ϕ	$\Rightarrow$	ϕ

Non-summarizable: c

Requirements

- Explicit hierarchies
- Dimensions = measures
- Multiple hierarchies
- **Aggregation semantics**
- Facts/dimensions: n-n
- Different granularities
- Non-strict hierarchies
- Non-onto hierarchies
- Non-covering hierarchies

Performance Optimization

Sales

tid	pid	locid	sales
1	1	1	10
2	1	1	20
3	2	3	40
…	…	…	…

1 billion rows

- The data warehouse contains GBytes or even TBytes of data!
- OLAP users require fast query response time
  - They don’t want to wait for the result for 1 hour!
  - Acceptable: answer within 10 seconds
- Idea: precompute some partial result in advance and store it
  - At query time, such partial result can be utilized to derive the final result very fast

Materialization Example

- Imagine 1 billion sales rows, 1000 products, 100 locations
- CREATE VIEW TotalSales (pid, locid, total) AS
  SELECT s.pid, s.locid, SUM(s.sales)
  FROM Sales s
  GROUP BY s.pid, s.locid
- The **materialized** view has 100,000 rows

Sales

tid	pid	locid	sales
1	1	1	10
2	1	1	20
3	2	3	40
…	…	…	…

1 billion rows

- Wish to answer the query:
  - SELECT p.category, SUM(s.sales)
    FROM Products p, Sales s WHERE p.pid=s.pid
    GROUP BY p.category
- Rewrite the query to use the view:
  - SELECT p.category, SUM(t.total)
    FROM Products p, **TotalSales t**
    WHERE p.pid=t.pid GROUP BY p.category
  - Query becomes 10,000 times faster!

VIEW TotalSales

pid	locid	sales
1	1	30
2	3	40
…	…	…

100,000 rows

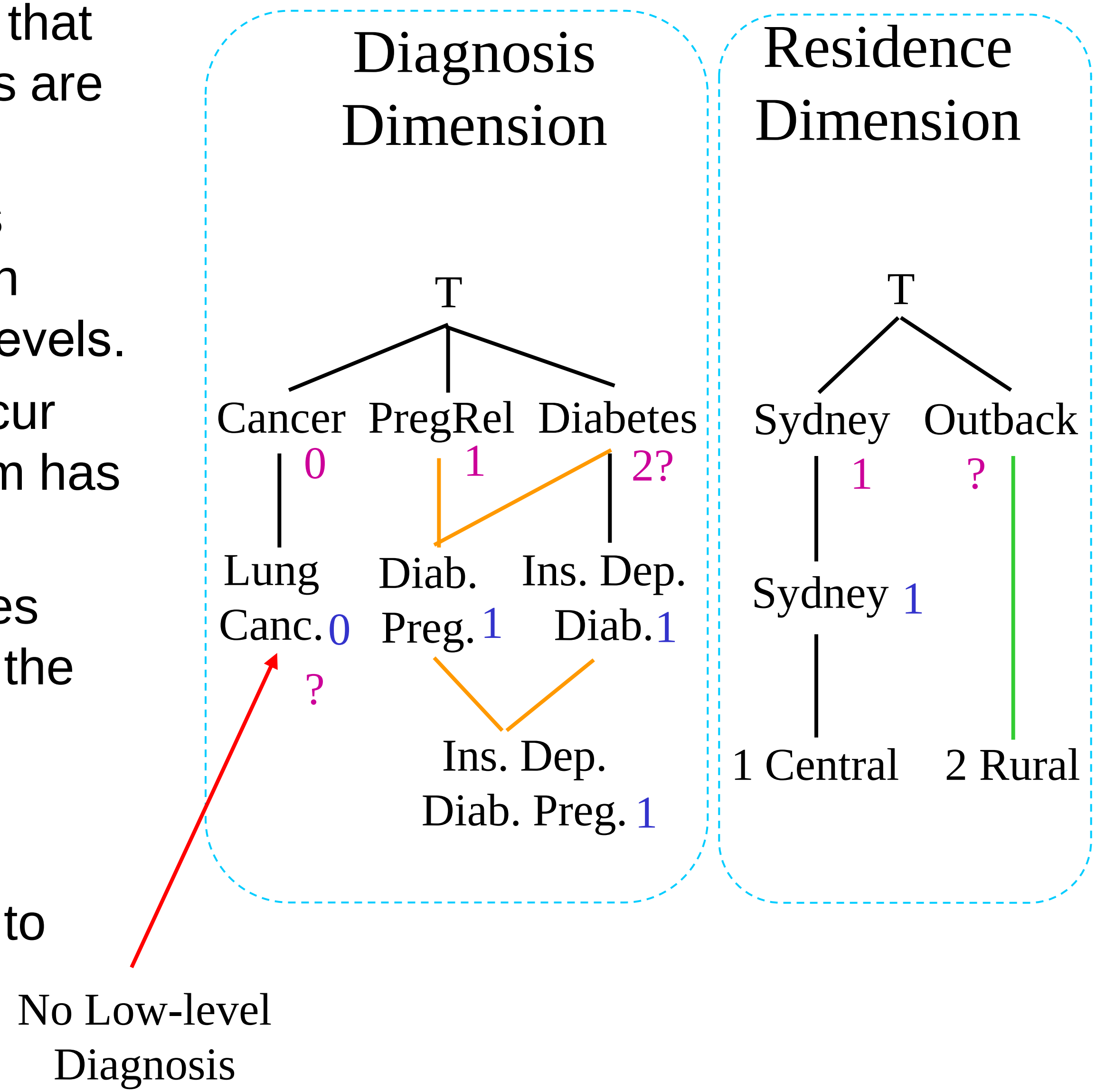
Complex DW Data

- The patient data is more complex than normal DW data
- Complexity may cause problems
  - Getting wrong query results
  - Not possible to use pre-aggregation for performance

- The data must obey certain properties
  - Summarizability: ”always get correct results”
- Careful data modeling needed to find potential problems

Hierarchy Properties

- Summarizability requires that the dimension hierarchies are *covering*, *onto,* and *strict.*
- *Non-covering* hierarchies occur when links between dimension values “skip” levels.
- *Non-strict* hierarchies occur when one lower-level item has several parents.
- *Non-onto* (into) hierarchies occur when the height of the hierarchy is varying.
- These properties cause **problems** when re-using stored counts of patients to compute new values.

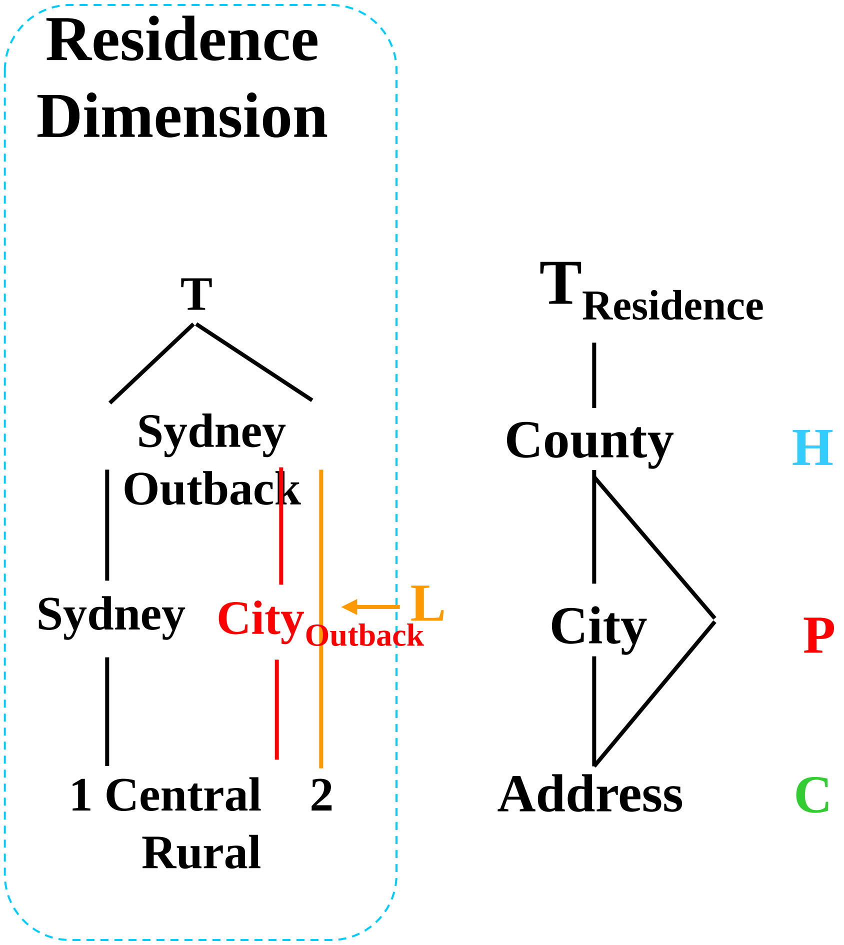


Hierarchy Transformations

- The overall task is to transform non-summarizable hierarchies to summarizable hierarchies automatically.
- A hierarchy is transformed in three steps:
  - 1) Transform the hierarchy to be *covering*.
  - 2) Transform the result from 1) to be *onto.*
  - 3) Transform the result from 2) to be *strict.*
- We give an algorithm for each transformation. Each algorithm assumes that the previous algorithm(s) has been applied.

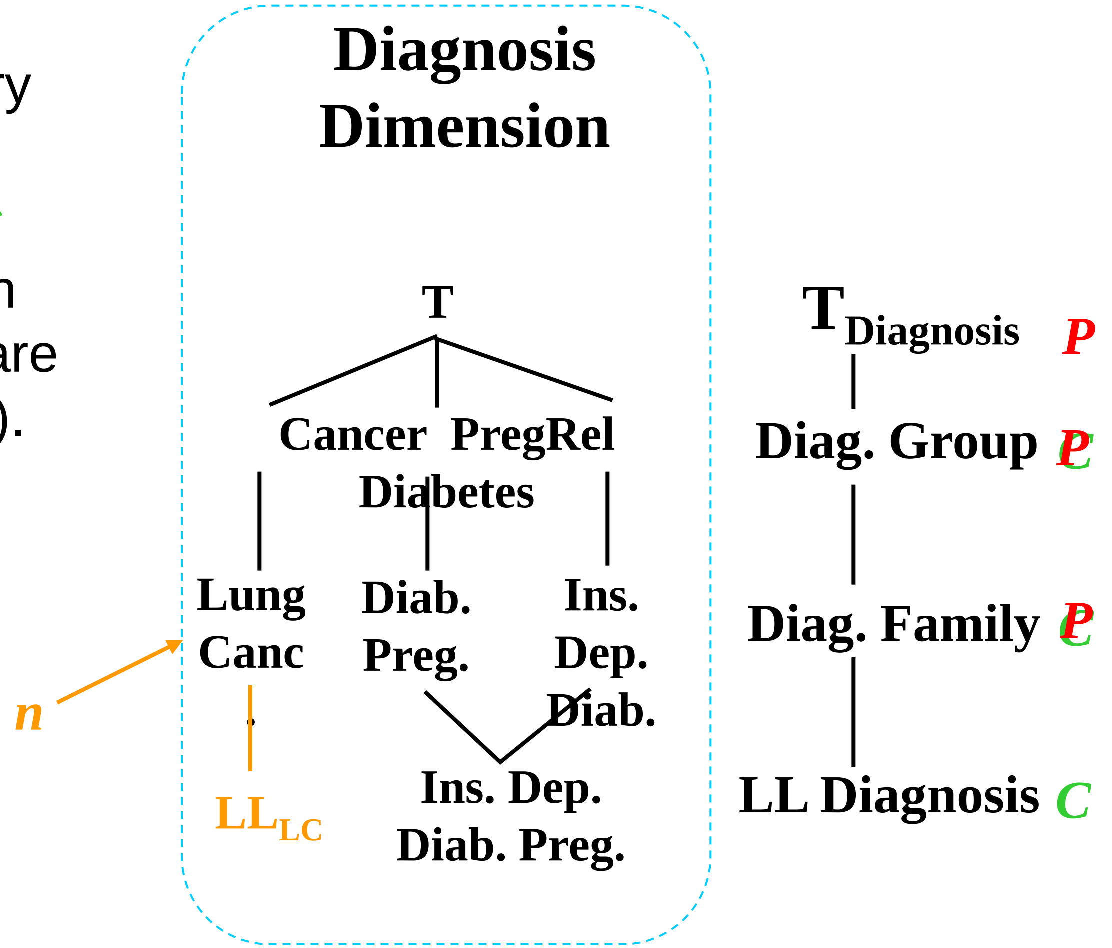
Non-covering Hierarchies

- The MakeCovering algorithm starts with the bottom category *C* (Address).
- For each parent category *P* of *C* (City and County) it looks for parent categories *H* that are "higher than" *P* (City < County).
- The algorithms finds all the links *L=(h,c)* from *H* to *C* that are not covered by going through *P*.
- For each link *L* an intermediate value is inserted into *P* and linked to *h* and *c*.
- Finally, the algorithm is applied recursively to *P* (nothing changes in this case).



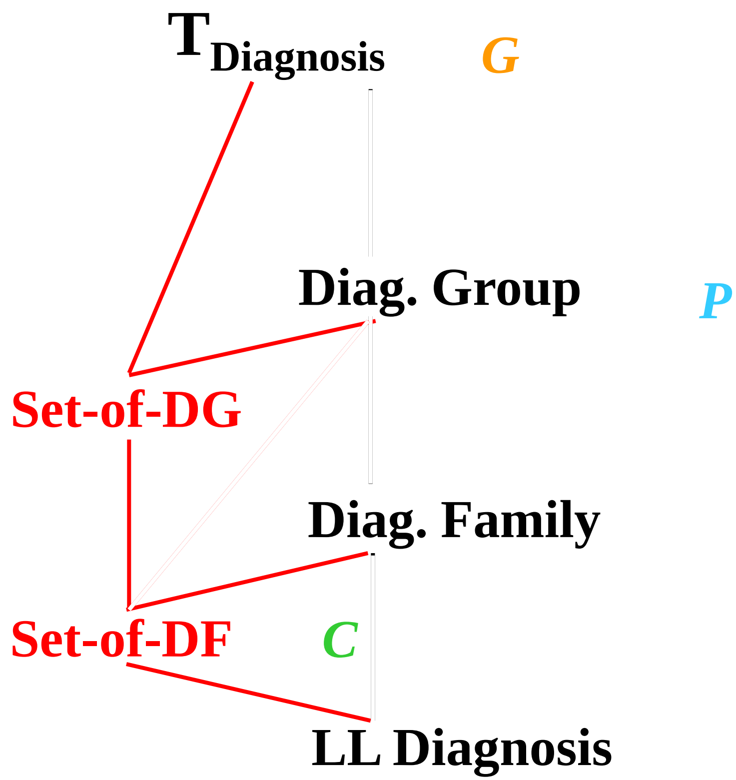
Non-onto Hierarchies

- The MakeOnto algorithm starts with the top category *P* ($T_{diagnosis}$).
- For each child category *C* of *P* it finds the values *n* in *P* with no children (none are found in the first two calls).
- For each childless *n* in *P*, the algorithms inserts a placeholder value c_n in *C* and links it to *n*.
- Finally, the algorithm is called recursively on *C*.



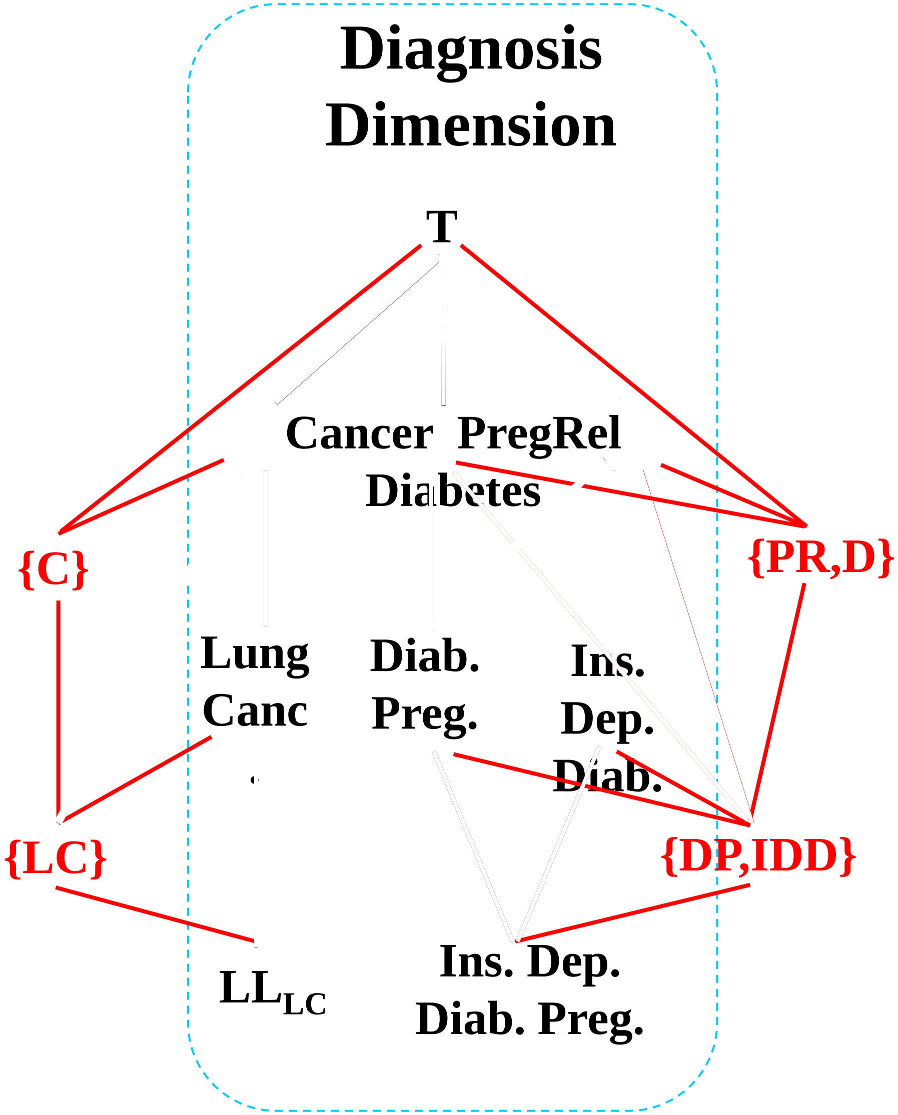
Non-strict Hierarchies

- Schema changes shown first
- Idea: “fuse” sets into one value.
- The MakeStrict algorithm starts with a child category C.
- For each parent category P of C, it looks for non-strictness between C and P **if** P *has parents*.
- If non-strictness occurs, a new category N (holding *sets of* P) is inserted between C and P.
- Grandparents G of C are linked to the new category.
- “Unsafe” links are removed.
- Finally, the algorithm is called recursively on N.



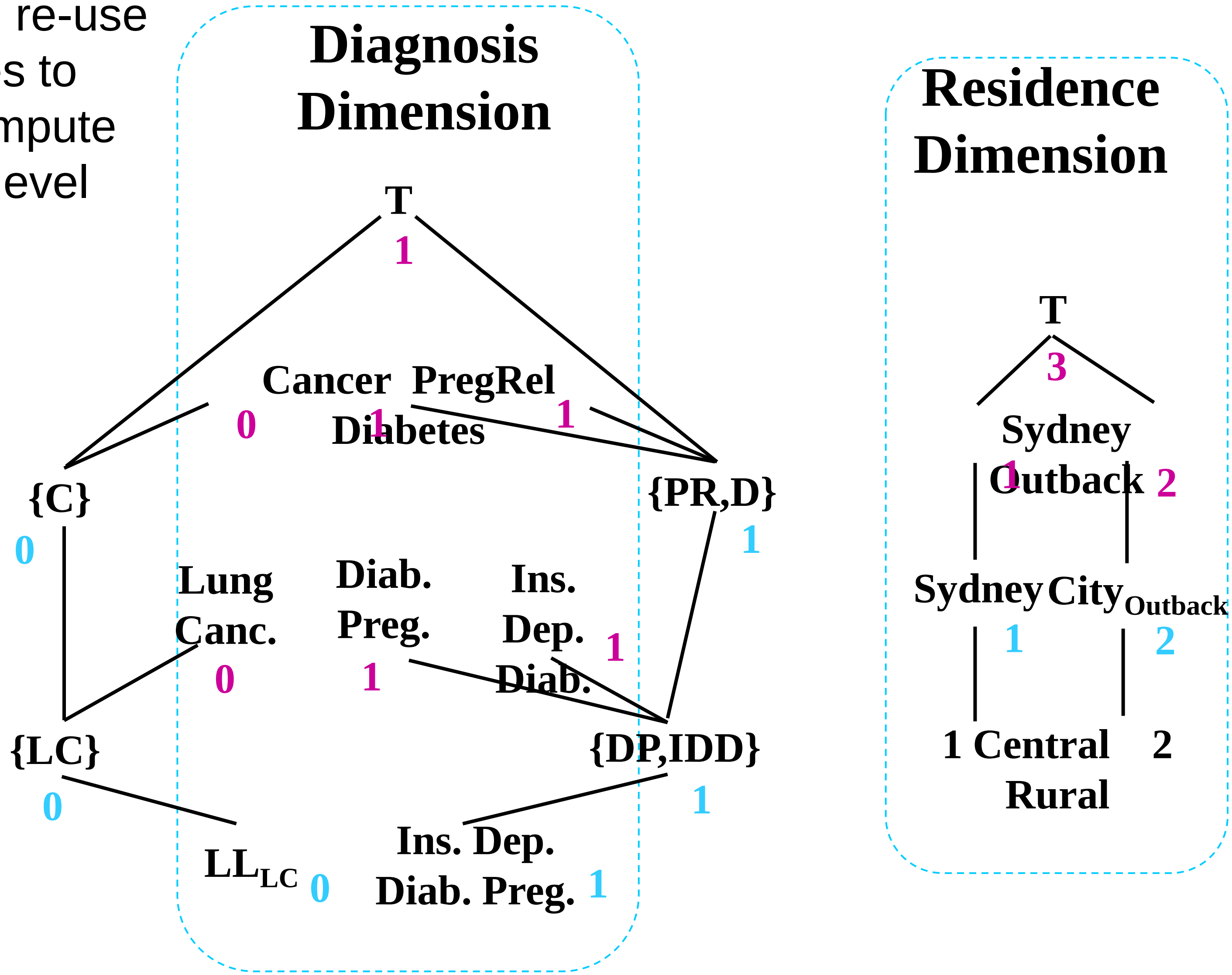
Non-strict Hierarchies

- Instance changes shown now.
- The MakeStrict algorithm starts with a child category *C*.
- For each parent category *P* of *C*, it looks for non-strictness between *C* and *P* **if** *P has parents*.
- If non-strictness occurs, new "fused" values (*sets of P*) are inserted into *N*.
- Values in grandparents *G* of *C* are linked to the new values.
- "Unsafe" links are removed.
- Finally, the algorithm is called recursively on *N*.



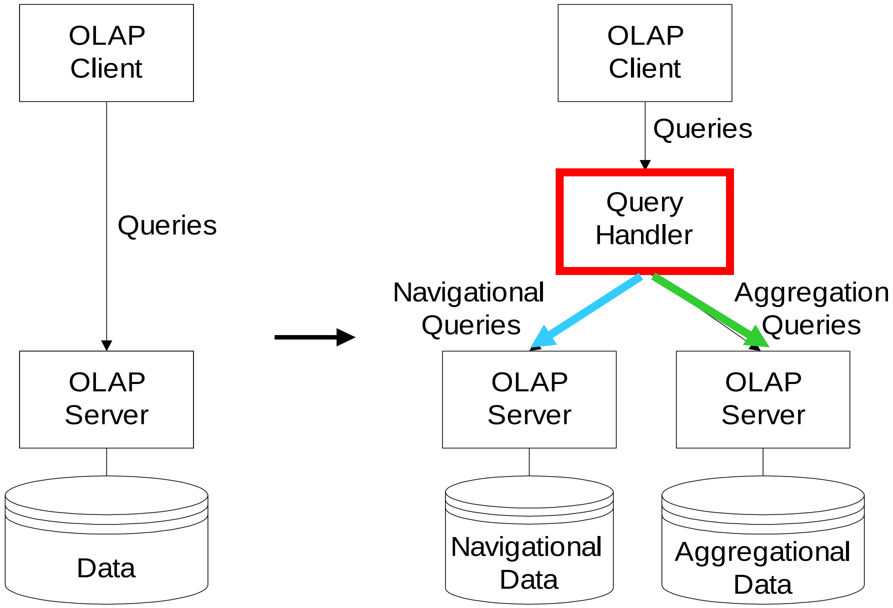
Did We Solve The Problem ?

- Now we can re-use stored values to *correctly* compute new higher-level values.



Integration in Current Systems

- The transformations should be *transparent* to the user.
- Modern OLAP systems have client and server components, communicating using, e.g., OLE DB for OLAP or MD-API.
- OLAP queries consist of 80% *navigation* and 20% *aggregation* queries [Kimball].
- Integration is achieved using a Query Handler that sends navigation queries to the original DB and aggregation queries to the transformed DB.
- Small space overhead (the hierarchies only are stored twice).



Real-World Hierarchy Examples

- Diagnosis hierarchy from the British “Read Codes” hierarchy of medical concepts:
  - 14 levels
  - 29030 nodes
  - 32274 edges
  - Height from 3 to 14 levels (non-onto)
    - 5484 nodes at level 5 (largest level)
    - 32 nodes at level 13
  - 2258 edges jump 2 or more levels (non-covering)
  - 3067 nodes have more than one parent (non-strict)

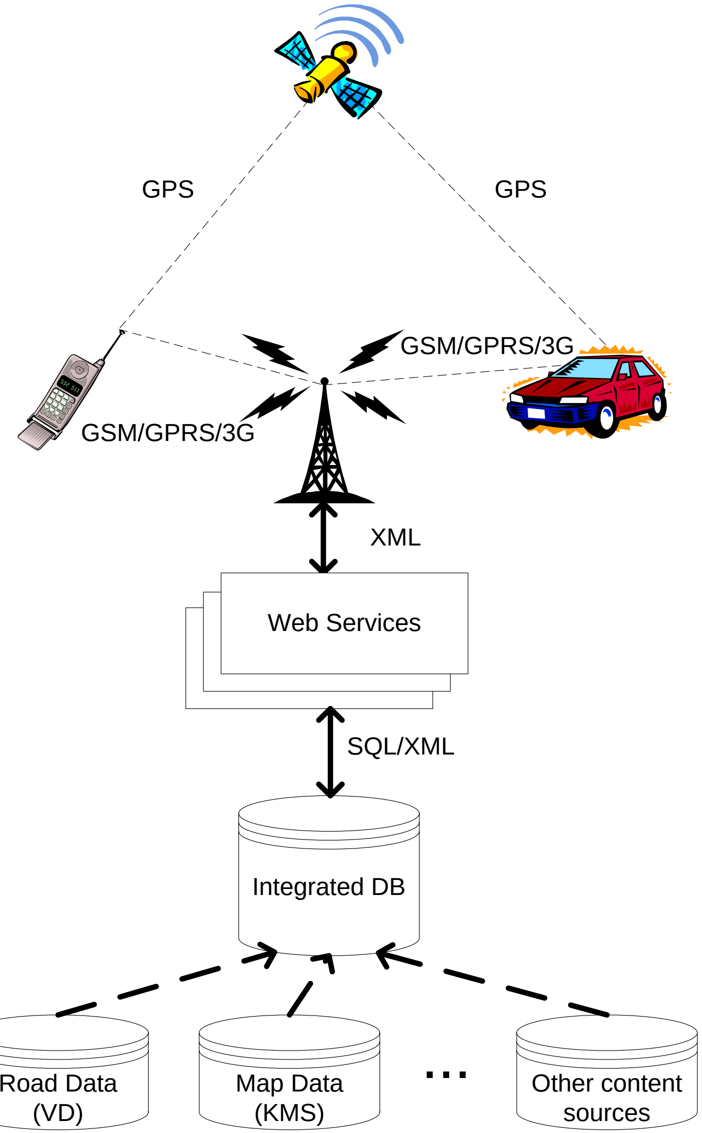
- Web portal concept hierarchies, e.g., Yahoo
  - ~100,000 nodes, similar characteristics

Talk Overview

- Multidimensional modeling recap
  - Cubes, dimensions, measures, …
- Complex multidimensional data
  - Modeling
  - Performance techniques
- **Complex spatial multidimensional data**
- Integrating cubes and XML
- Semantic web warehousing
- Integrating cubes and text
- Multidimensional music data

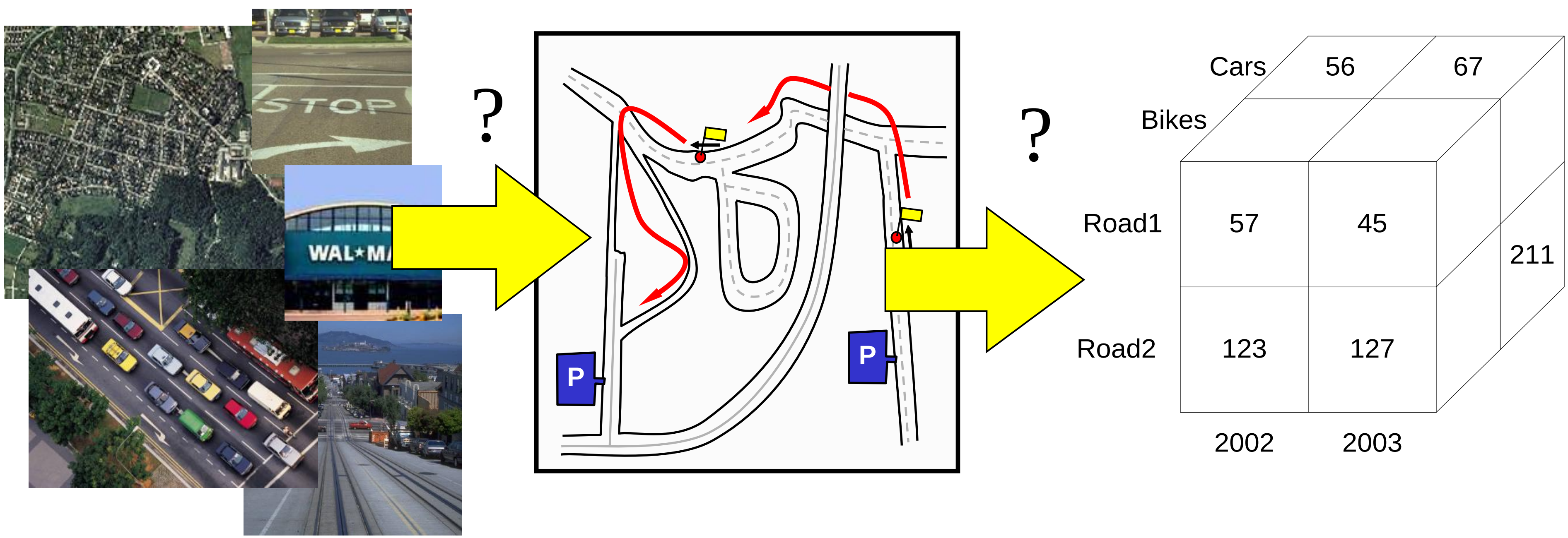
Location-Based Services (LBS)

- Mobile devices are powerful
  - Always on-line (4G)
  - Knows position (GPS)
- These devices enable a whole new type of internet service
  - **Location-Based Services**
  - Based on time, location, and previous user behavior
- LBS service types
  - Traffic, tourists, advertising, safety, games, car insurance,…
  - Example: ”show bus” means ”when is the next bus from the nearest bus stop to where I usually go around this time”
  - Architecture seen right



MD LBS Research

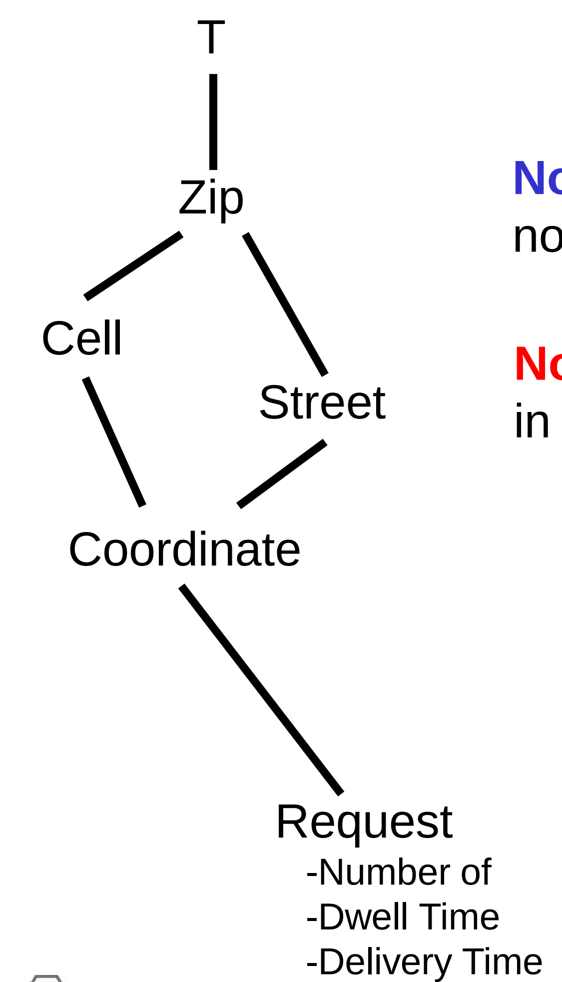
- Two lines of research
  - How do we capture (the for LBS relevant parts of) the ”real world” in a correct and efficient database model ? (left question mark)
  - How do we capture and **analyze** this data using multidimensional ”cube” technology ? (right question mark)



daisy

Non-Standard Dimensions

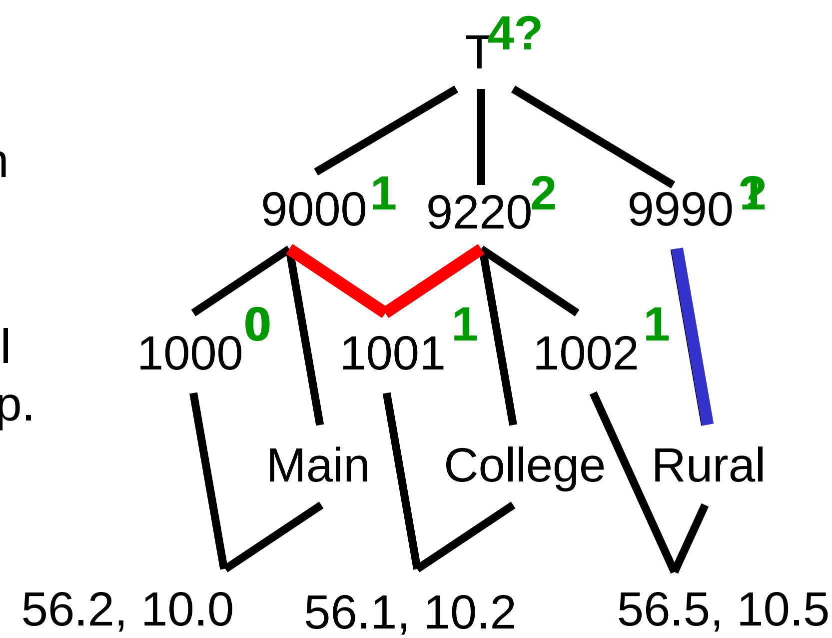
Location dimension



Non-onto: Zips with no cell.

Non-strict: One cell in more than one Zip.

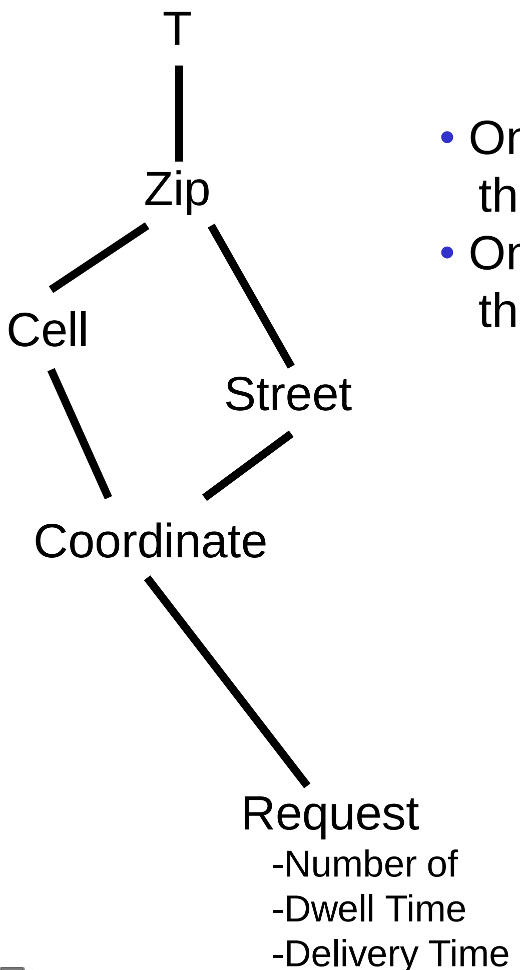
Location dimension instance



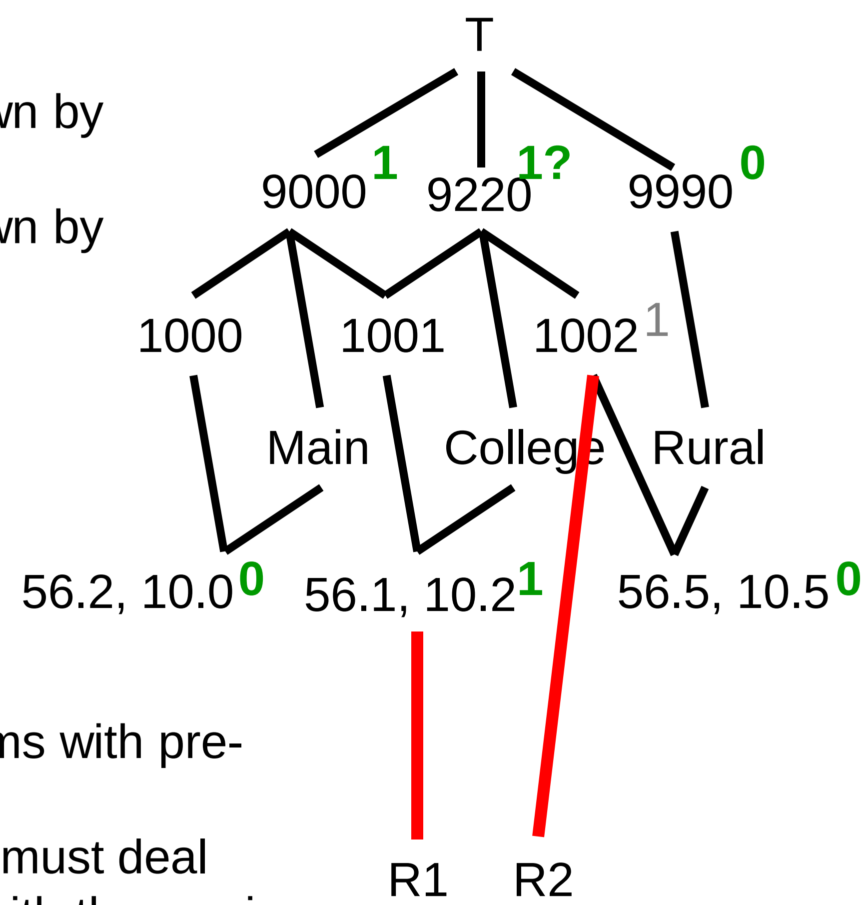
- We have assumed that the relationship among levels is "contained in."
- Can not be captured using standard multi-dimensional data models.
- Summarizability, not always possible.
- One solution is **hierarchy normalization**

Imprecision and Varying Precision

Location dimension



Location dimension instance



- One location is known by the coordinate.
- One location is known by the cell.

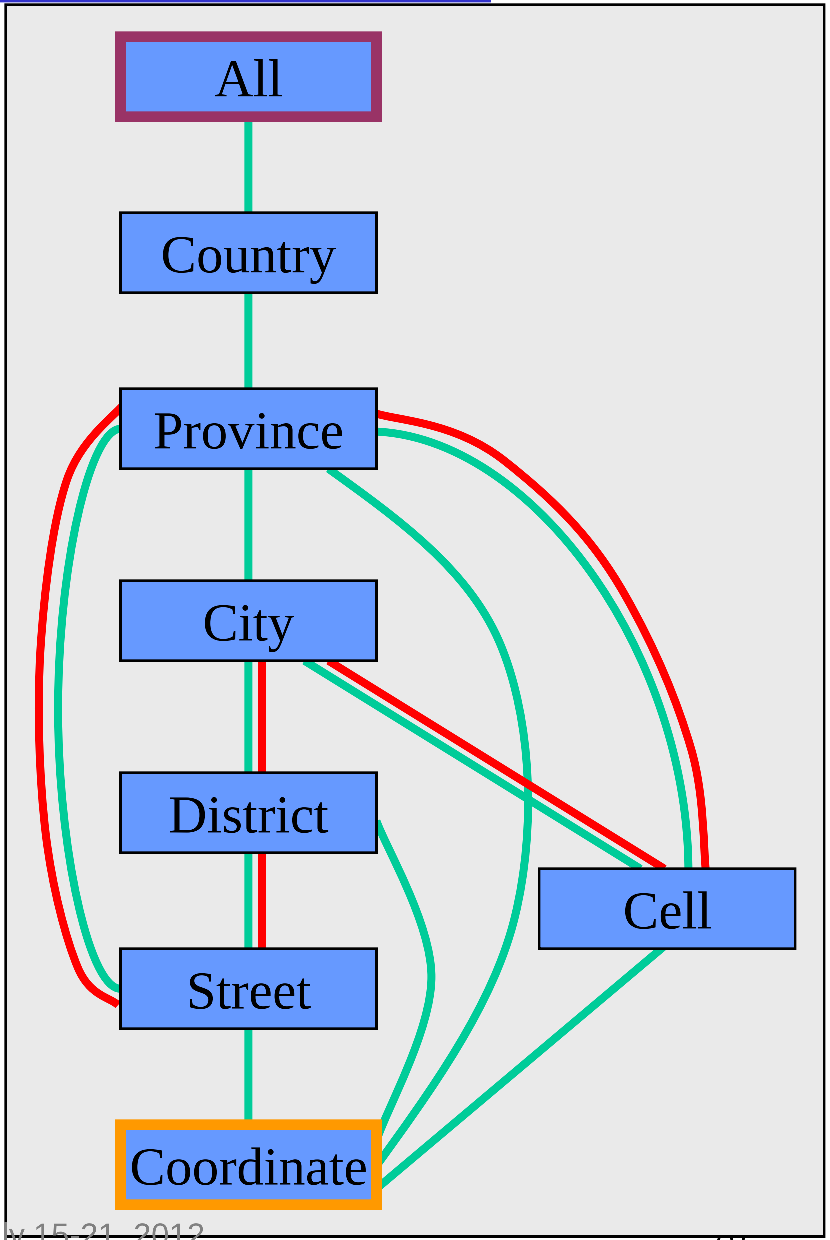
- Again problems with pre-aggregration.
- The systems must deal intelligently with the varying precisions.

MD Model For Spatial Data

- Must handle many non-standard requirements
  - Explicit hierarchies in dimensions
  - Multiple hierarchies in each dimension
  - **Partial containment hierarchies**
  - Non-strict hierarchies
  - Non-onto hierarchies
  - Non-covering hierarchies
  - Different levels of granularity
  - Many-to-many fact-dimension relationships
  - Handling of imprecision

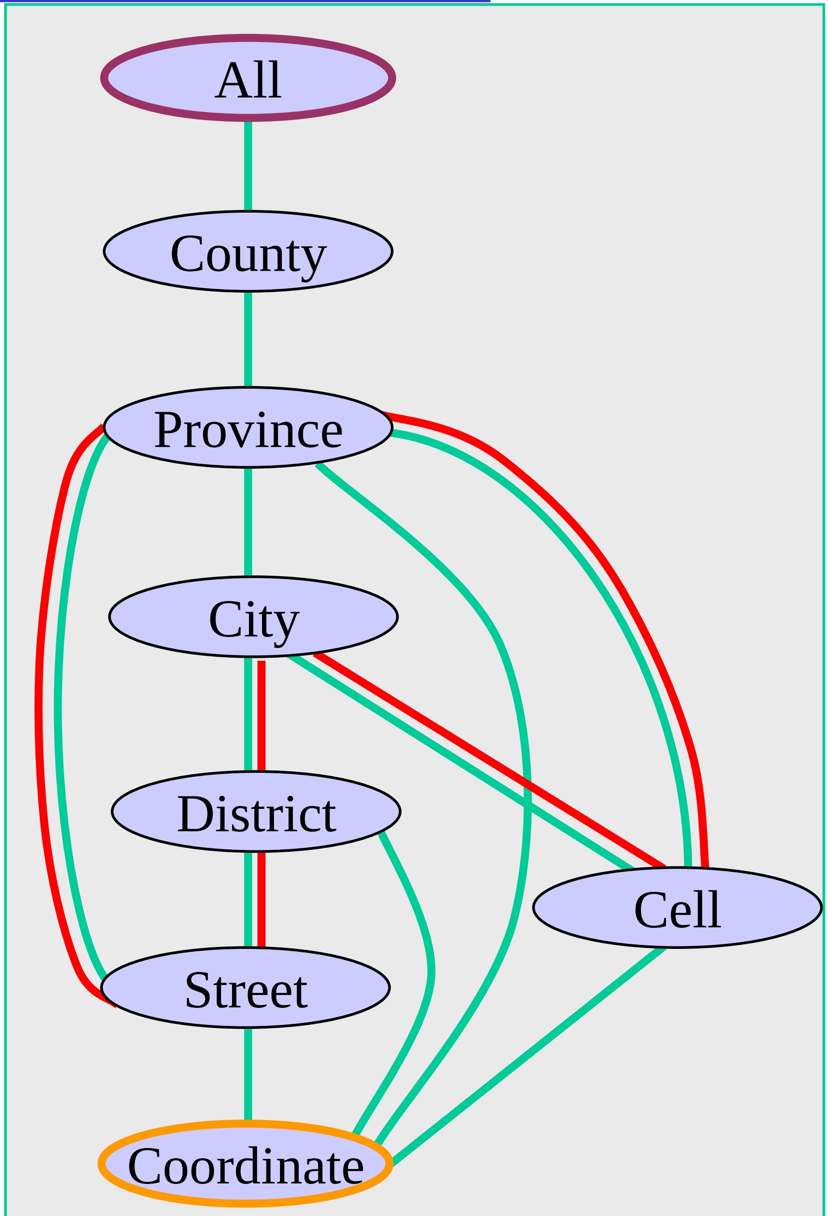
Data Model Schema: Dimension Type

- *Dimension types* refines the schema of dimensions
- A dimension type is given by
  - category types
  - full containment order on types
  - partial containment order on types
  - *bottom element*
  - *top element*
- *Category types* name "levels" of the dimension
- *Partial containment order* defines partial containment hierarchies of levels
- *Full containment order* defines full containment hierarchies of levels
- The combination of orders defines "mixed" containment hierarchies (may be mixed freely)



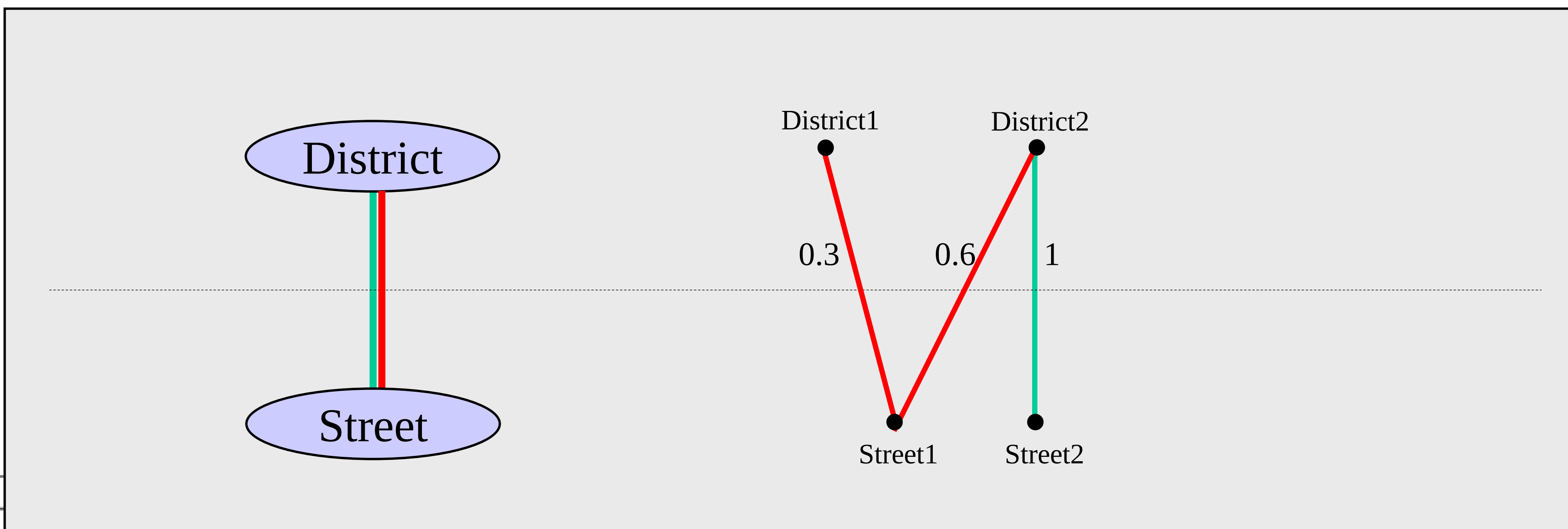
Data Model Instance: Dimension

- *Dimension* has a schema defined by a corresponding dimension type
- *Categories* are **sets** of members of the corresponding "levels"
- *Partial containment order* and *full containment order* on the *categories* are analog to the orders on the category *types*



Data Model Instance: Dimension Values

- Partial order on the values assigns *degrees of containment* to the relationships
- The degree of containment indicates how much of the first value that is contained in the second value
  - Exception: “0 degree” indicates that the first value *may* be contained in the second value

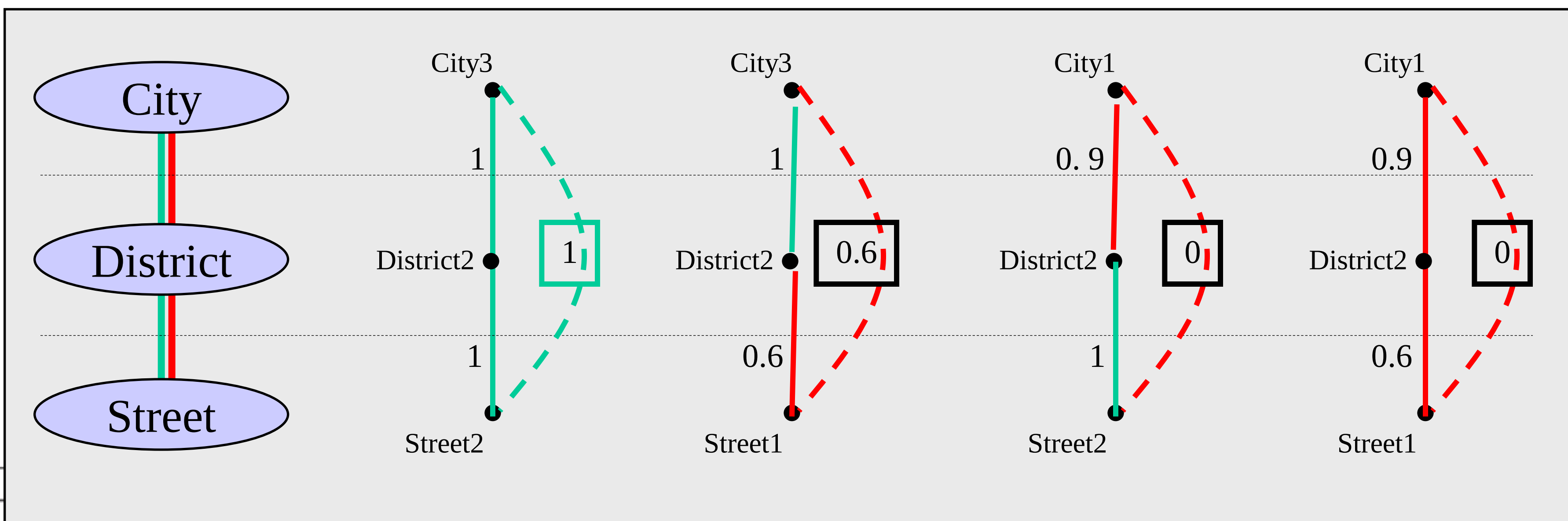


Transitivity of partial containment

- Four different combinations of degrees of containment for transitive case

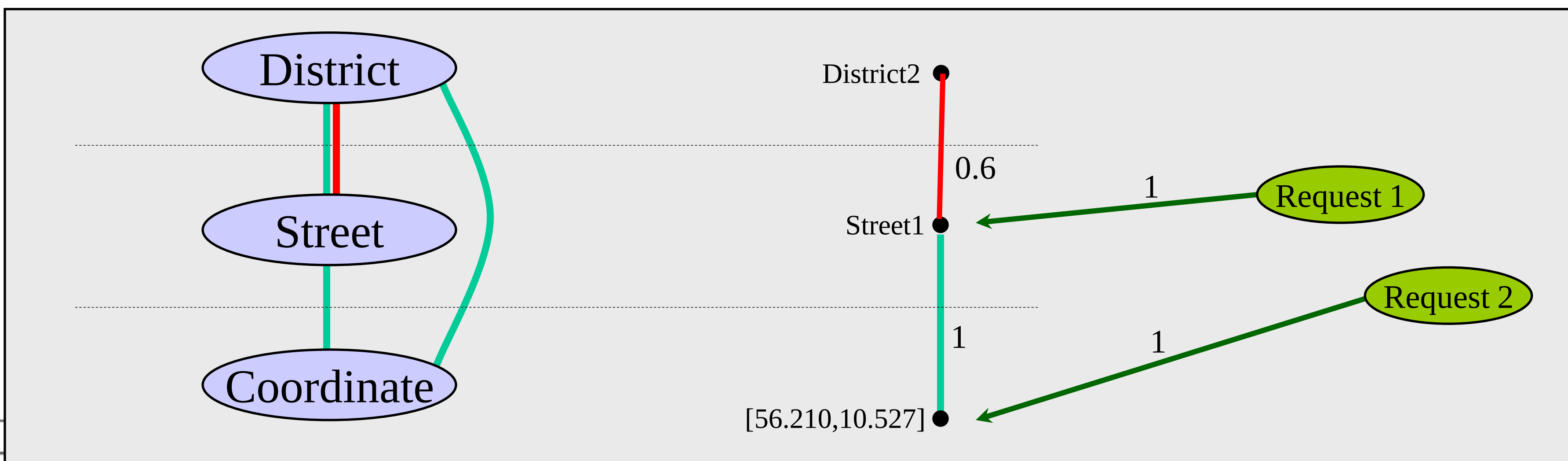
Transitivity of partial containment

- Degrees of containment among indirectly related values are deduced by rules of *transitivity of partial containment*
- The "*safe approach*"
  - *"Infer the maximum degrees of containment that we can guarantee"*
- The four cases lead to the four rules of transitivity



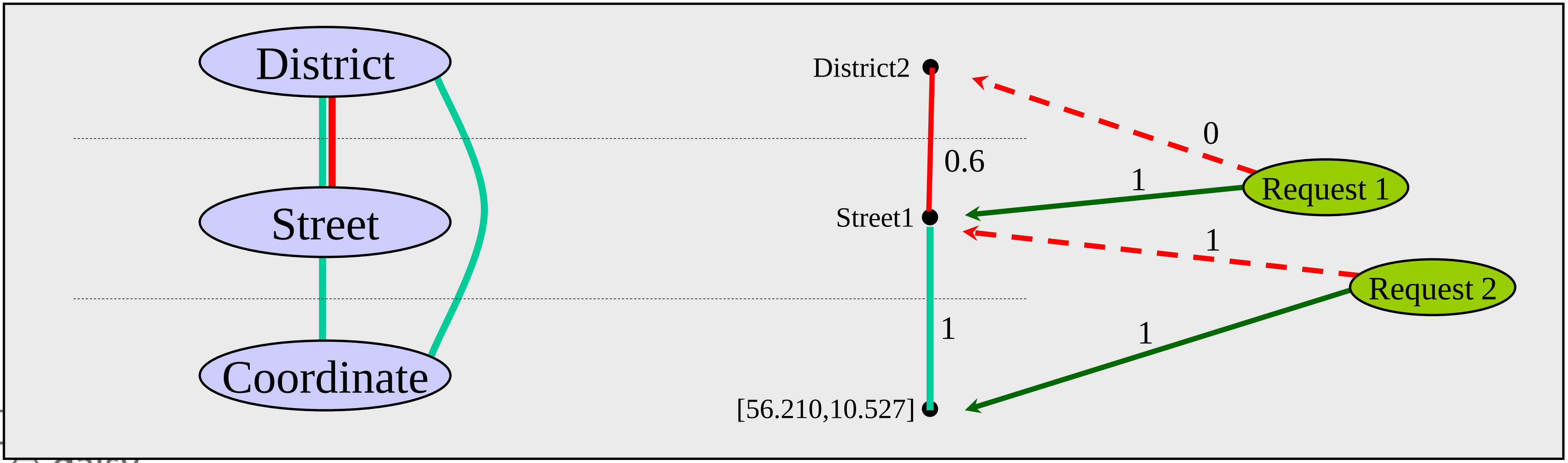
Data Model Instance: Facts

- The *Facts* are a set of entities of a given fact type
- A f*act-dimension relationship* maps a fact to a dimension value
- *A fact-dimension relation* gathers all fact-dimension relationships in a dimension
- Each fact-dimension relationship is seen as a containment relationship among dimension values with the degree of 1



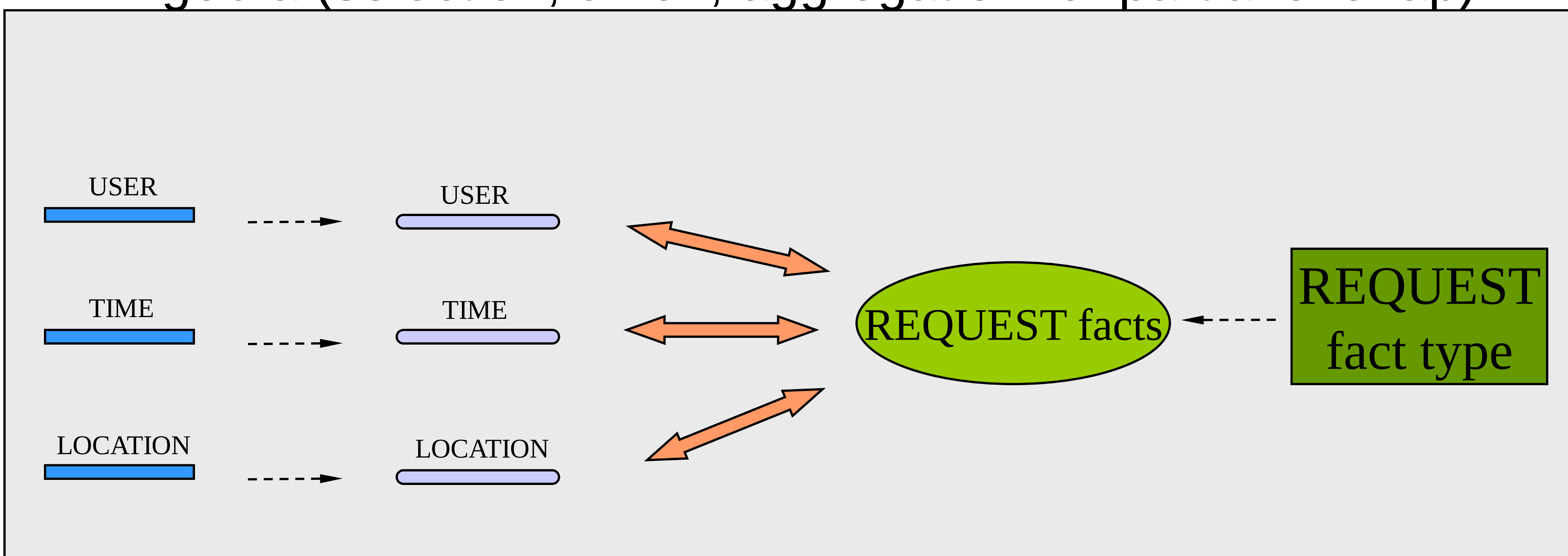
Data Model Instance: Facts

- Fact-dimension relationships are propagated up along the hierarchies of dimension values
- The propagation is done in accordance with the rules of transitivity of partial containment
- Each propagated fact-dimension relationship receives degree of 1 or 0



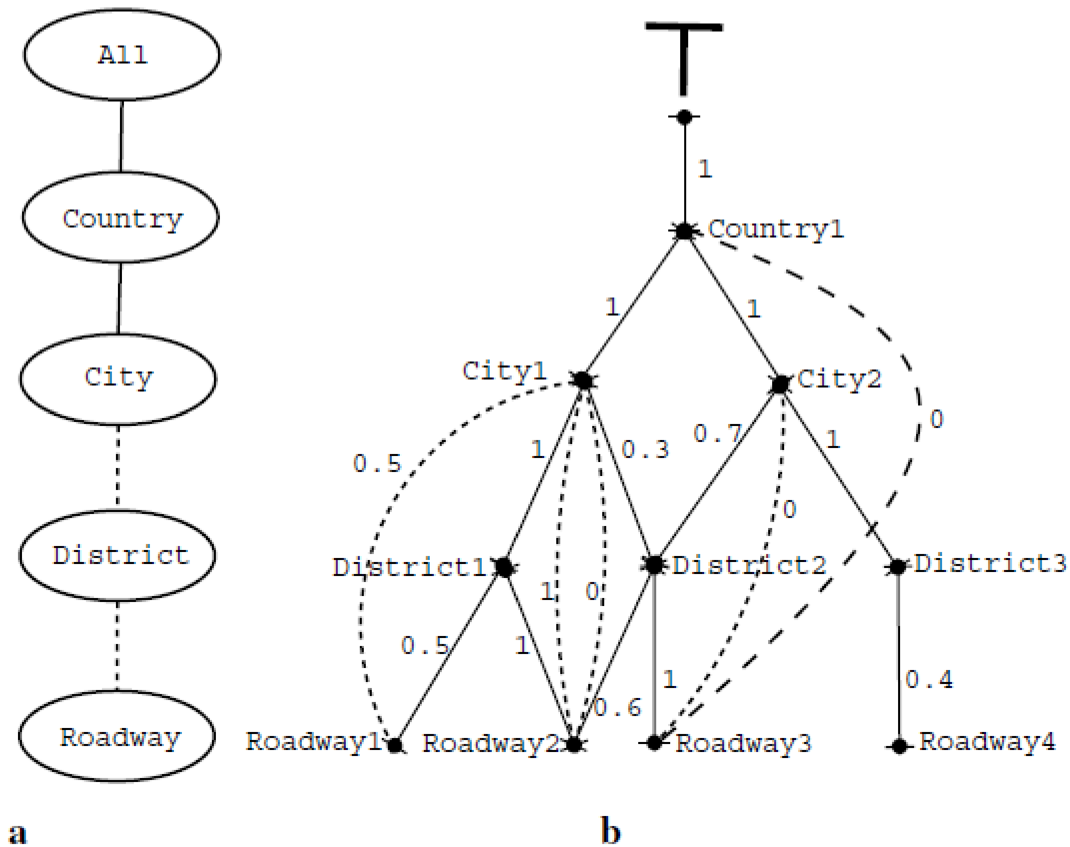
Multidimensional object

- *Multidimensional object* is a multidimensional cube
- The object consists of
  - a fact type and a set of dimension types
  - a set of facts
  - a set of dimensions
  - a set of fact-dimension relations
- Algebra (selection, union, aggregation for partial overlap)



Enabling Pre-Aggregation

- Hierarchy normalization extended to partial overlaps



Transformations

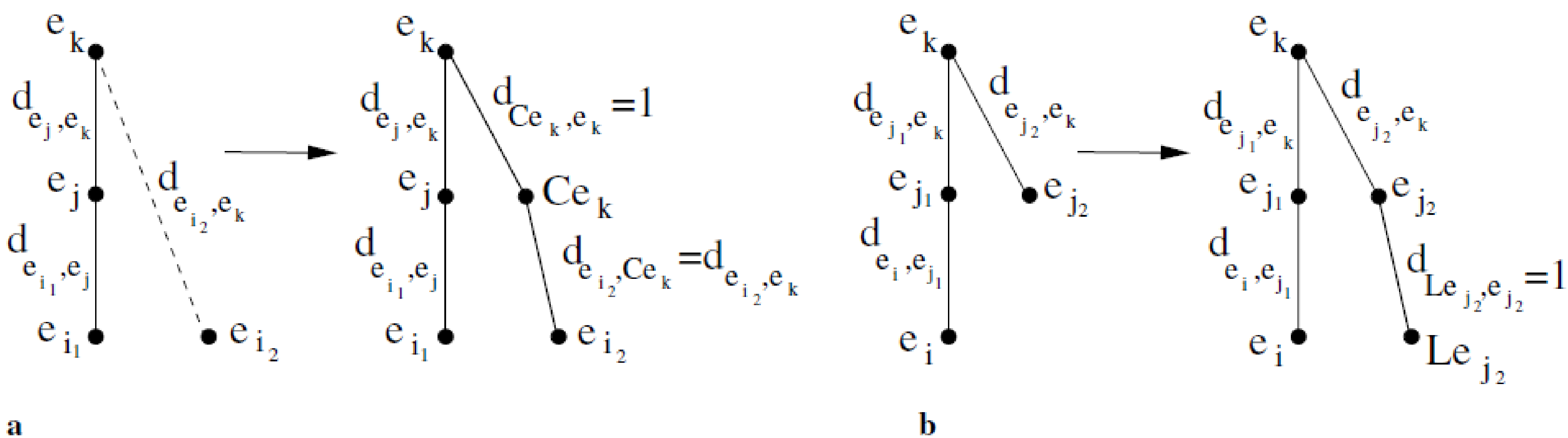


Fig. 9a,b. Transformations by the **a** PMakeCovering and **b** PMakeOnto algorithms

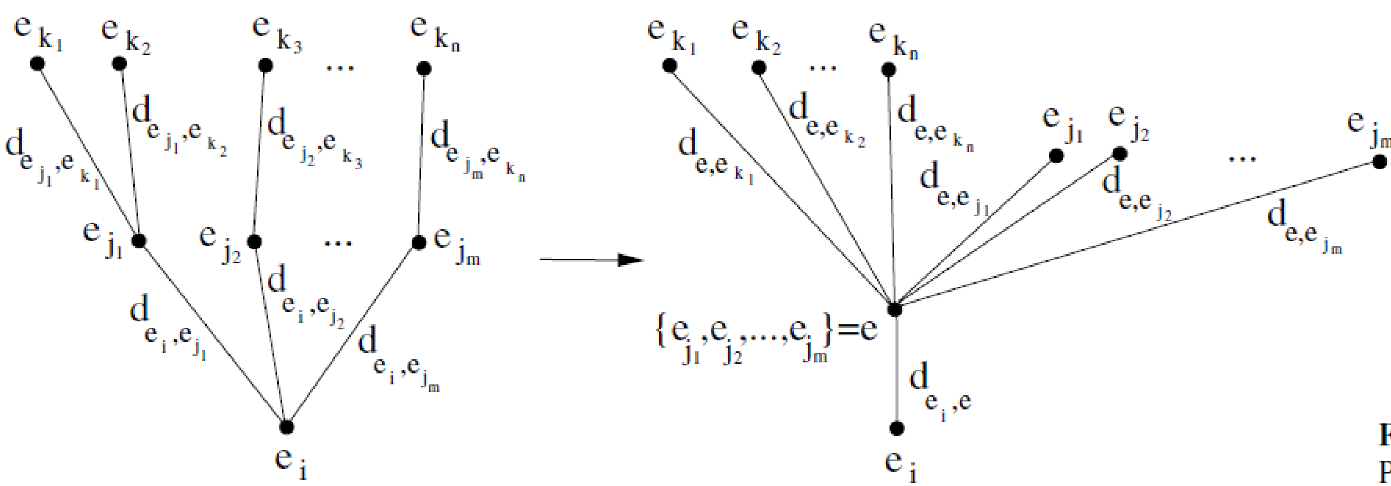
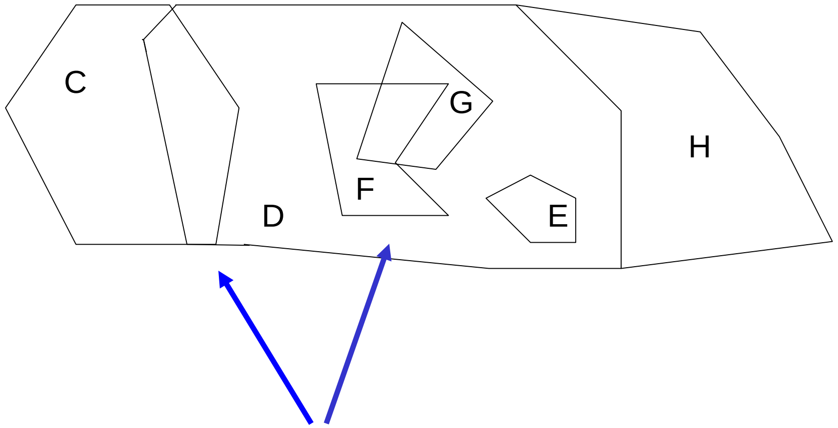


Fig. 10. Transformations by the PMakeStrict algorithm

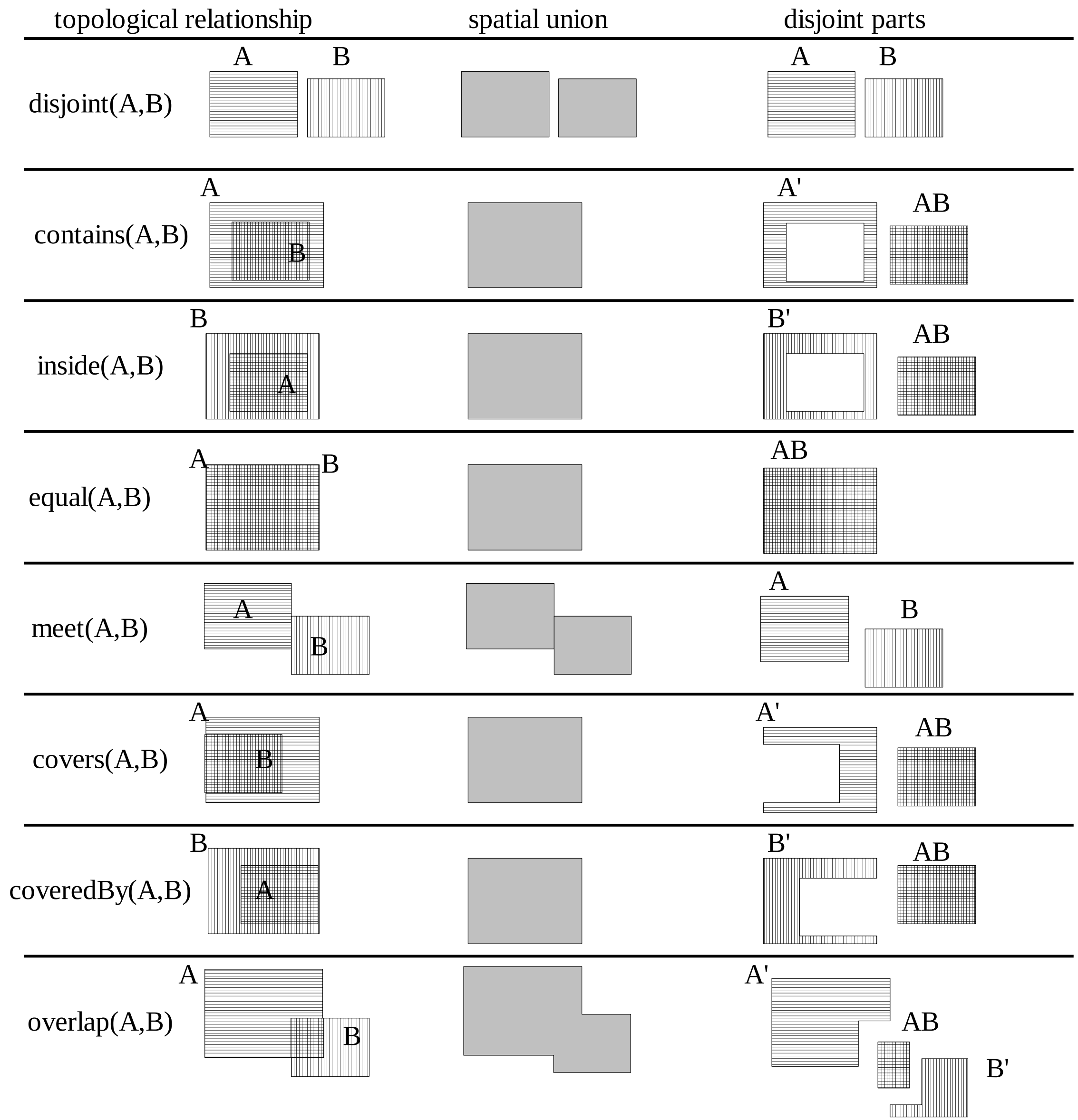
Complex Spatial Facts

- Facts are *spatial objects* (SOs)
  - *Example: areas*
- SOs have *spatial properties* (SPs)
- Aggregation in SDWs is computing (aggregate) values of SPs over the *spatial union* of facts
- Facts *overlap*
- This means that SPs are generally *not distributive* over any aggregation function
- However, SPs are generally *disjoint distributive* over some aggregation function
- Example: size is disjoint distributive over SUM
- Fact overlap may also cause non-strict hierarchies



Overlapping areas with Land Cover

2D Topological Relationships



Correct Aggregation ?

- Assume *g* is a spatial property, *f* an aggregation function
- Examples: *g* = size, *f* = SUM
- +,\,* denotes spatial union, difference, and intersection
- Column 2 is *distributive,* column 3 is *disjoint distributive*

Case	**g(A + B)=f(g(A),g(B))**	**g(A+B)=f(g(A\B),g(B\A),g(A*B))**
Disjoint	Yes	Yes
Contains	No	Yes
Inside	No	Yes
Equal	No	Yes
Meet	Yes	Yes
Covers	No	Yes
Coveredby	No	Yes
Overlaps	No	Yes

Spatially Extended Normalization

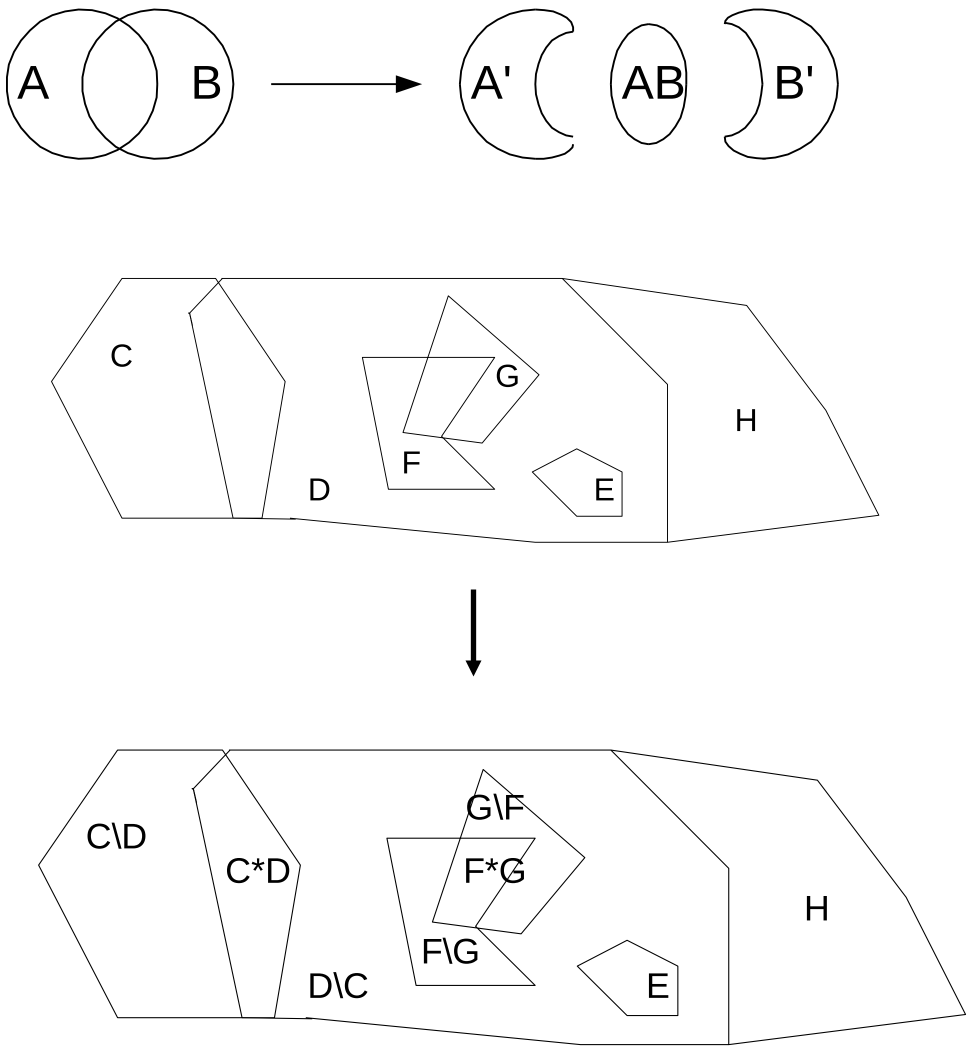
- Idea:
  - Compute only on disjoint parts
  - ”Fix” hierarchies so practical pre-aggregation can be used
- Based on previous work for irregular OLAP hierarchies
- Independent of representation type for spatial objects
- For parts 3)+5), facts may be considered bottom category

1) Compute disjoint parts of fact areas
2) Map new (disjoint) fact areas to dimension values
3) Make the dimension hierarchies covering
4) Make the dimension hierarchies onto
5) Make the dimension hierarchies aggregation strict

daisy

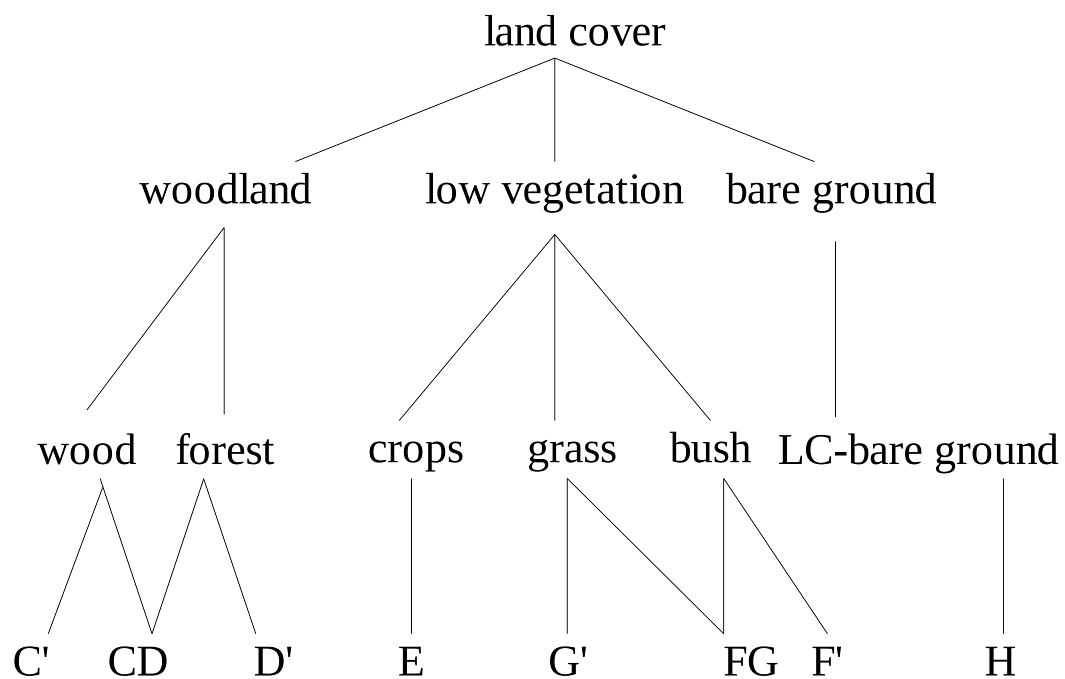
Computing Disjoint Fact Parts

- Idea: it does not matter for the *aggregate* result whether facts (areas) are in one or more ”pieces”
- *Split* overlapping areas into disjoint parts to support SPs that are only disjoint distributive, e.g., over SUM
- Example: size



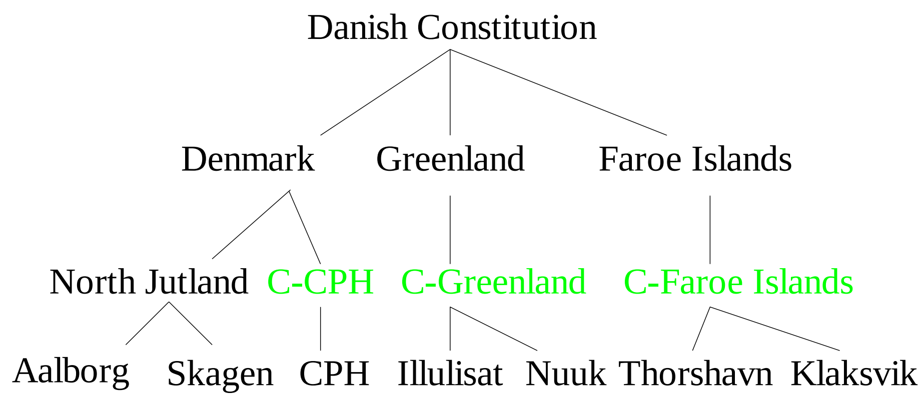
Map Fact Parts To Dimensions

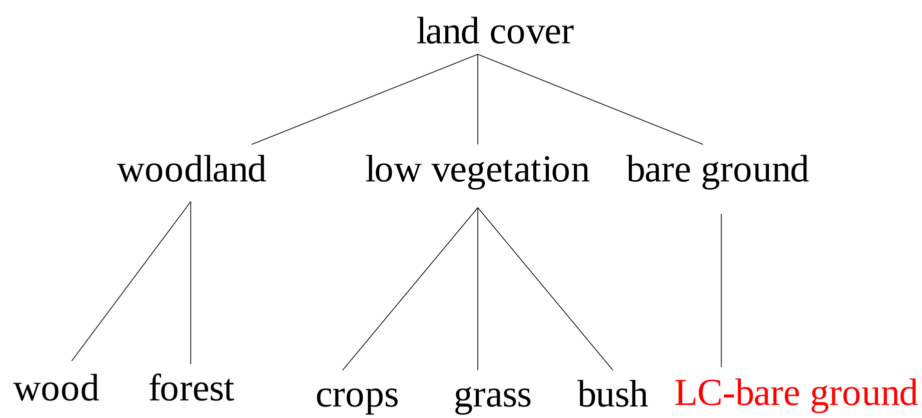
- Next, the new disjoint fact parts are *mapped* to the relevant dimension values
- C´=C\D, CD=C*D, D´=D\C, G`=G\F, FG=F*G, F´=F\G
- Facts are now the bottom category
- Introduces non-strictness



MakeCovering and MakeOnto

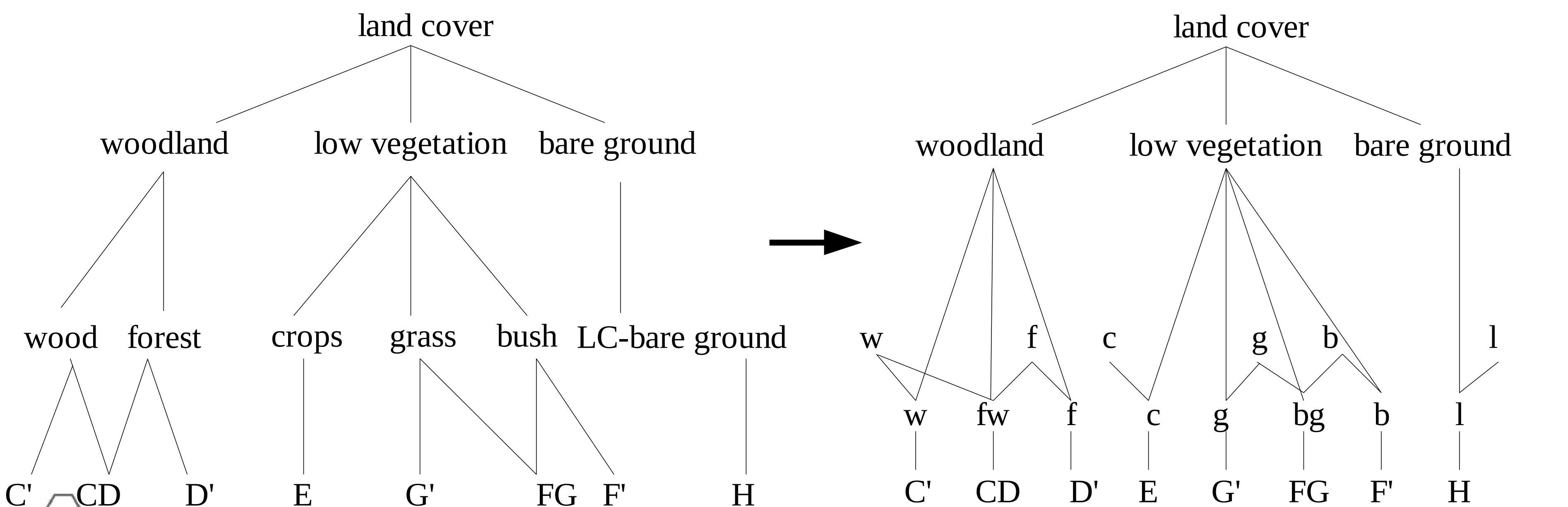
- MakeCovering: ”pad” hierarchies with *intermediate* dimension values that can hold the values of the ”missing” parts
- MakeOnto: ”pad” hierarchies with ”placeholder” children for nodes with no children (no children ok for fact level)
- Algorithms do this automatically
- The hierarchies are now balanced





MakeStrict

- ”Fuse” *sets of parents* into one node in new category
- *Link* children, parents, and grandparents to new nodes
- *Unlink* children and parents => no double-counting
- Result: ”safe” hierarchy + ”unsafe” parts (aggregation strict)
- Below: C´=C\D, CD=C*D, D´=D\C, G`=G\F, FG=F*G, F´=F\G



Talk Overview

- Multidimensional modeling recap
  - Cubes, dimensions, measures, …
- Complex multidimensional data
  - Modeling
  - Performance techniques
- Complex spatial multidimensional data
- **Integrating cubes and XML**
- Semantic web warehousing
- Integrating cubes and text
- Multidimensional music data

Motivation

- OLAP-systems are good for complex analysis queries
  - Easy-to-use
  - Fast
  - *Business, science ...*
- More and more data is needed for the analyses
- Problems with physical integration in existing OLAP systems
  - Integrating new data requires (partial) cube rebuild => too slow
- Problems arise with
  - Short term and varying data needs
    - *Demographic info, disease info...*
  - Dynamic data
    - *Stock quotes, competitors prices, disease info...*
  - Data with limited access
    - *Product information, public databases, scientific data banks*

Motivation

- Data will often (in the future) be available in XML format
  - Info syndicators, public institutions
  - SWISS-PROT, TrEMBL, GenBank,…

```
<Diseases>
    <Disease>
        <Diagnosis>Pneumonia</ Diagnosis >
        <Code>N12</Code>
        <Symptom>Cough</Symptom>
        <Symptom>Fever</Symptom>
        :
    </Disease>
    :
</Diseases>
```

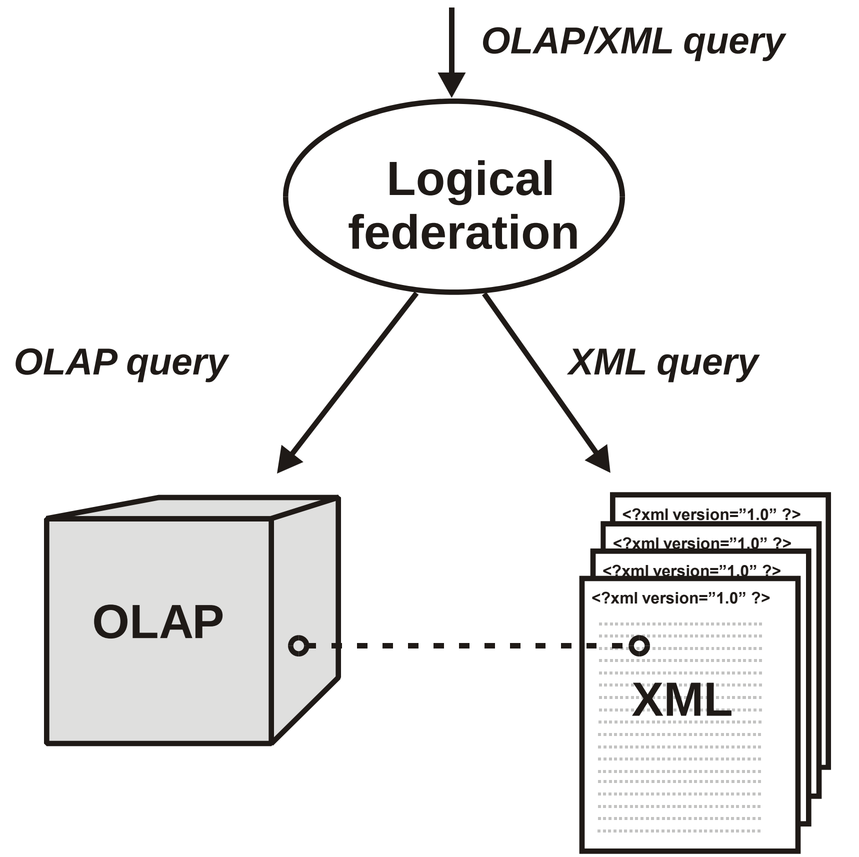
- **Hierarchical**
- **Elements**
- **Start-tag, end-tag**
- **Describes the *semantics* of the data**
- **Irregular structure**

- ***Goal: flexible access to XML data from OLAP systems***

daisy

OLAP-XML Federation

- Allows the use of external XML data as *virtual dimensions*
  - Decoration (extra info)
  - Grouping
  - Selection
- Loosely-coupled federation
  - Data stored in the ”best” place
  - Supports ad-hoc integration
  - Rapid prototyping of DWs
  - High degree of component autonomy
  - Data is always up-to-date
  - Performance problem?
- Recent term: situational BI



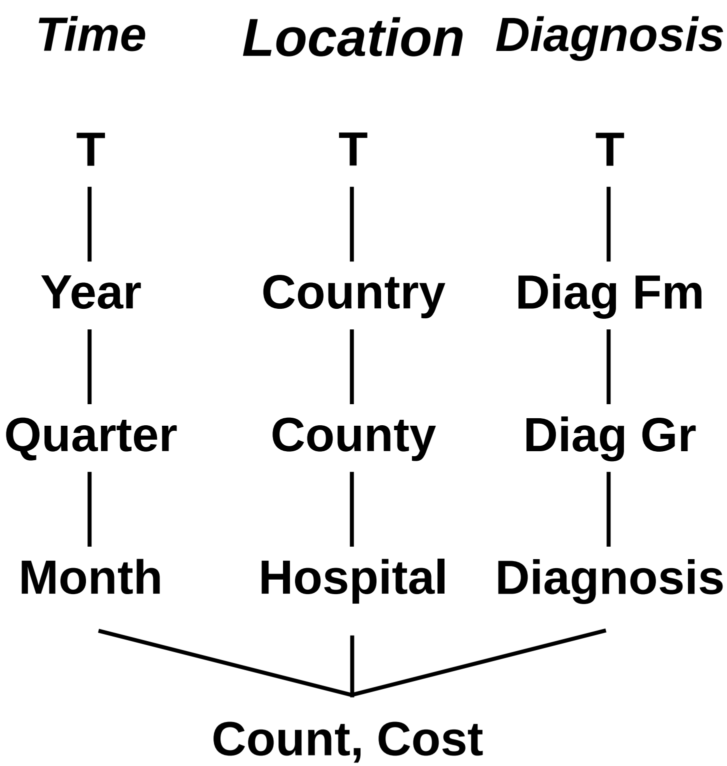
OLAP Component

- *Cubes*
  - *Dimensions*, having partially ordered set of of *levels*
  - Partially ordered set of *dimension values*
  - *Fact tables* contain dimension keys and *measure values*
- Irregular hierarchies
  - Non-strict, non-onto, non-covering
  - Affects the *summarizability* of the data (double-counting, etc.)
  - Aggregation types keep track of problems
- Formal algebra
  - Selection σ_{Cube}, generalized projection Π_{Cube}
- SQL_M query language
  - SQL with roll-up functions, etc.
  - **SELECT SUM (Cost),Hospital,Year(Time) FROM Patients GROUP BY Hospital, Year(Time) HAVING SUM(Cost)>1000**

XML Component

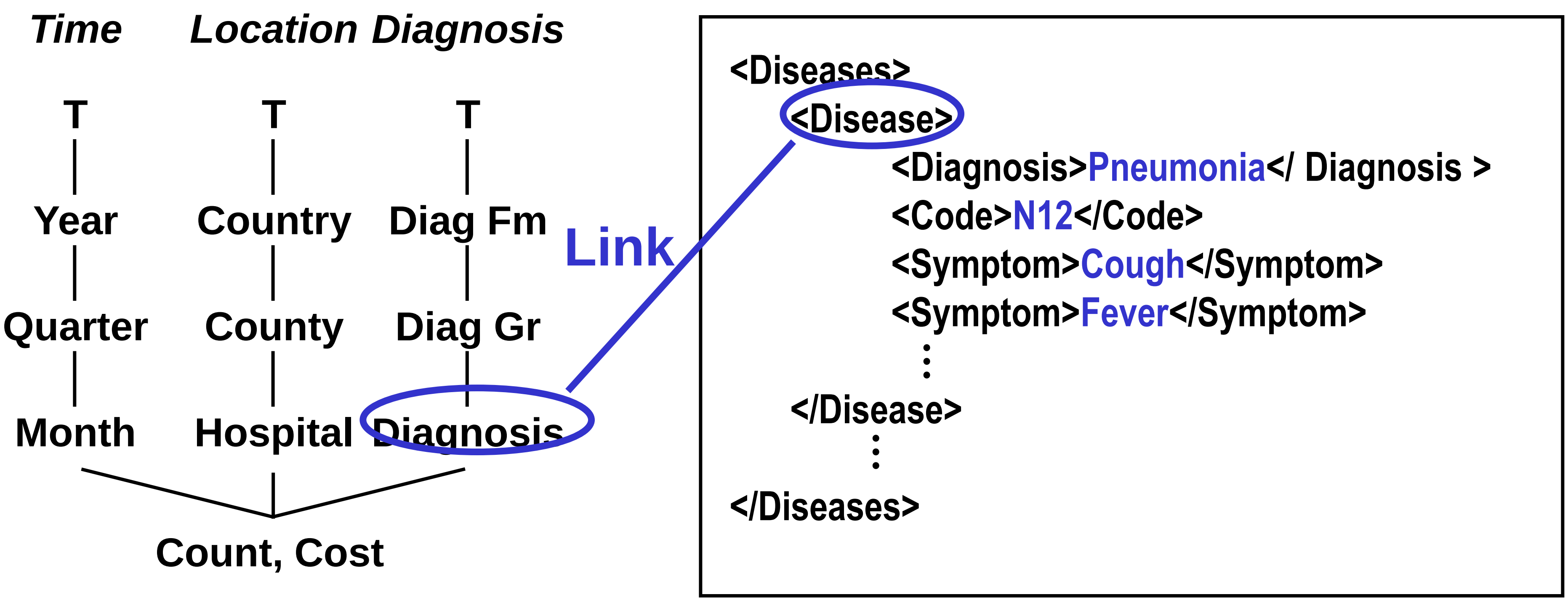
- XPath data model
  - XML document: Labeled Ordered Tree (LOT)
  - Tree consists of *nodes*
  - Seven types of nodes (root, element, text,…)
- XPath language
  - Similar to Unix file paths
  - /locationstep_1/../locationstep_n returns a set of nodes
  - Many forms of predicates
  - /Diseases/Disease[Code=”N12”]
- Our view: XPath expression = *function over a set of nodes*

OLAP-XML Federation




```
<Diseases>
    <Disease>
        <Diagnosis>Pneumonia</ Diagnosis >
        <Code>N12</Code>
        <Symptom>Cough</Symptom>
        <Symptom>Fever</Symptom>
          ⋮
    </Disease>
       ⋮
</Diseases>
```

OLAP-XML Federation



Links and federations

- Link
  - *Relation* between set of dimension values and set of XML nodes
  - Link *cardinality* (1-1, n-1, 1-n, n-n) will affect summarizability
- Enumerated link
  - List of connected (value,node) pairs
  - Very general, but hard to work with
- Natural link
  - A predicate specifies the connections
  - Diagnosis links to www.diseases.org/disease.xml/Diseases/Disease
  - Diagnosis=www.diseases.org/disease.xml/Diseases/Disease/Code
- Level expression
  - <level>/<link>/<XPath expression> specifies a concrete link *usage*
  - Default link allowed
  - *Diagnosis/Symptom* links Diagnosis level to Symptoms
- Federation: cube+XML docs+links

Decoration

- Add new dimension to cube, based on level expression
  - **Select Diagnosis[ALL]/Symptom,…, Cost FROM Patient**
- Problem: no nodes link to particular dimension value
  - Creates non-covering hierarchy
  - Solution: add ”N/A” values to dimension
- Problem: more than one nodes link to one dimension value
  - Creates non-strict hierarchy
- Three often-used semantics
  - ANY: Pick an arbitrary node, summarizability ok
  - CONCAT: Concatenate string values into one, summarizability ok
  - ALL: Use all nodes, good for grouping and selection, duplicates facts, summarizability violated
  - User chooses between these in query
  - More can be imagined

Grouping and Selection

- Grouping
  - Use level expressions in GROUP BY clause
  - **Select Diagnosis[ALL]/Symptom, Cost FROM Patient GROUP BY Diagnosis[ALL]/Symptom**
  - Semantics: 1) decorate w. new dimension(s), 2) perform aggregation
  - Non-strictness may need to be handled
- Selection
  - Use level expression in WHERE clause
  - **Select Diagnosis, Cost FROM Patient**
    - **WHERE Diagnosis[ALL]/Symptom='Cough'**
  - Semantics: 1) decorate w. new dimension(s), 2) perform selections over new cube, 3) remove new dimensions
  - No problem with ALL semantics
  - *"any"* (<>"ANY") selection semantics is used

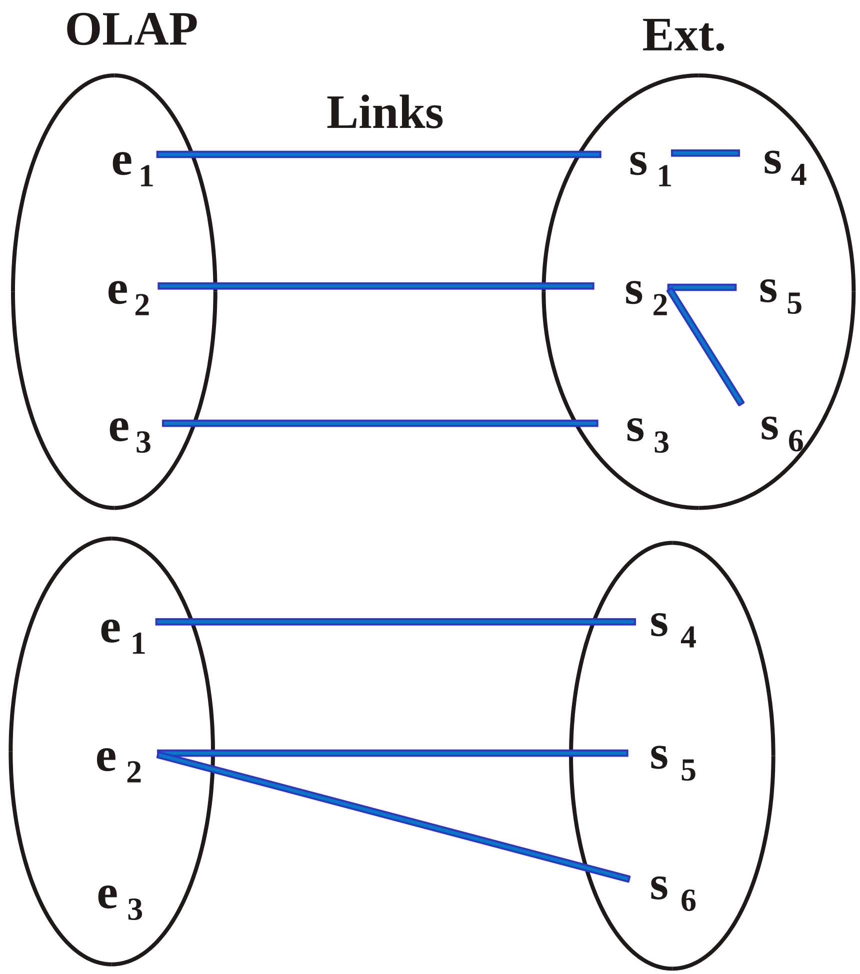
Decoration – How ?

Outside the cube:

Before:

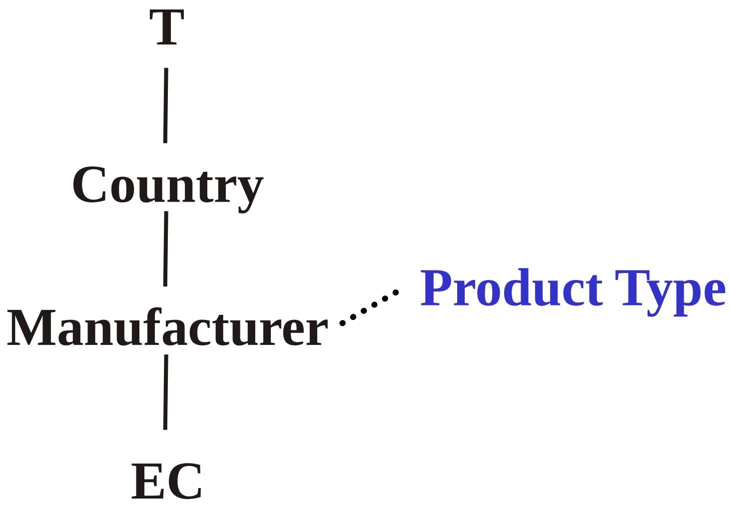
After:

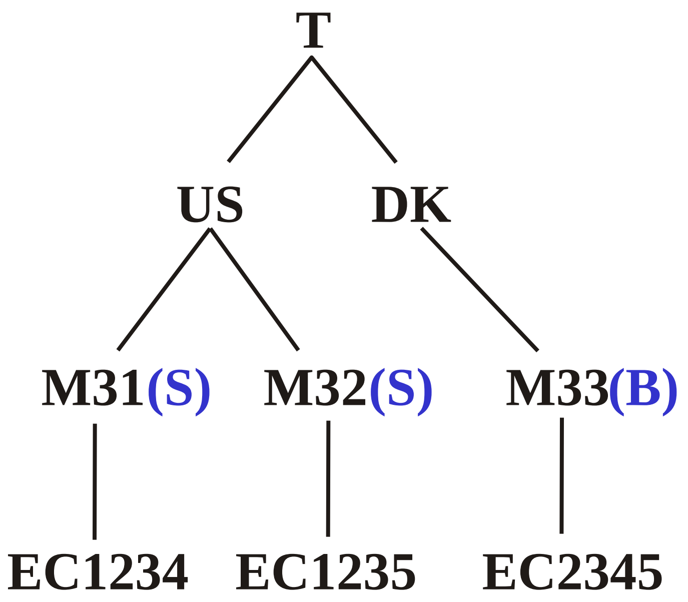
Too cumbersome to use



Decoration – How ?

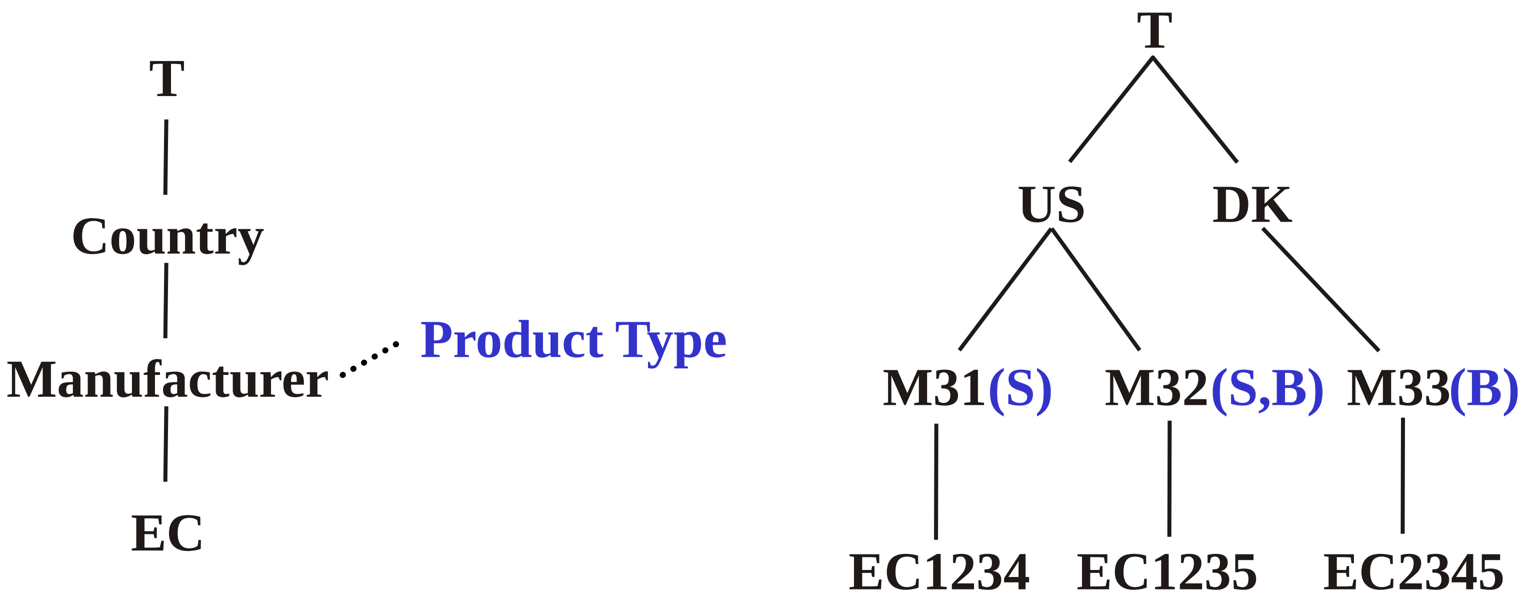
Decoration using attributes:





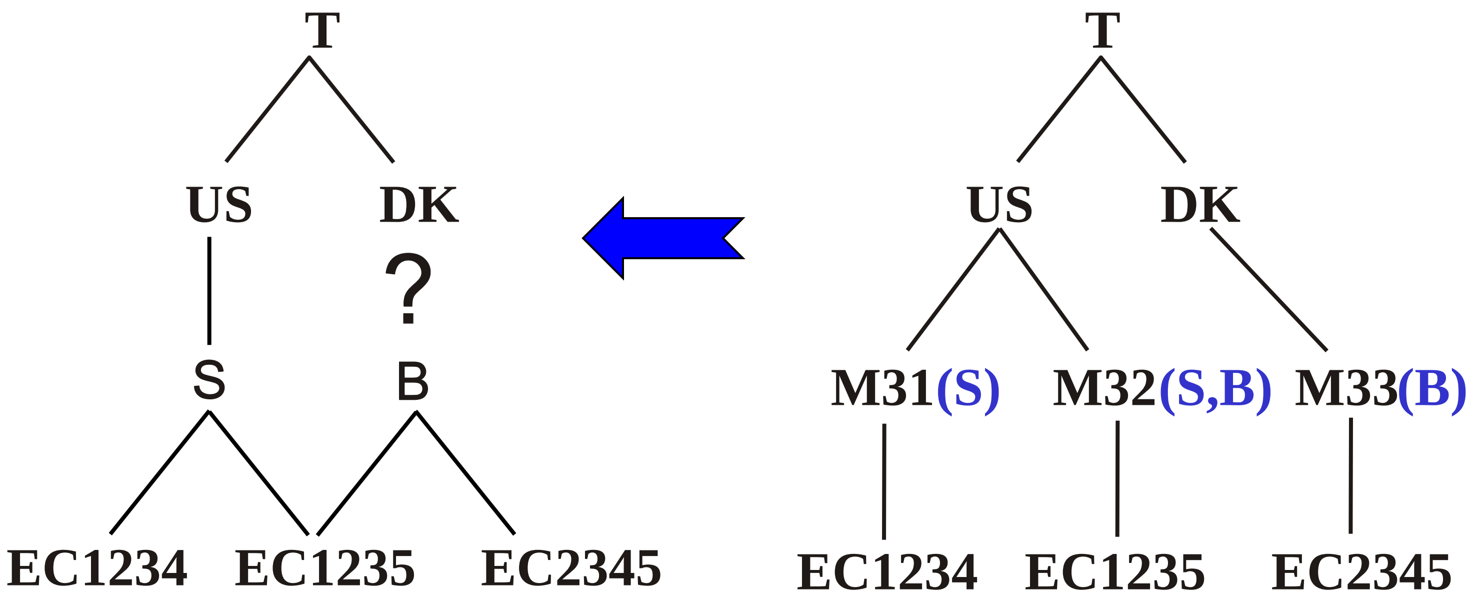
Decoration – How ?

Decoration using attributes:



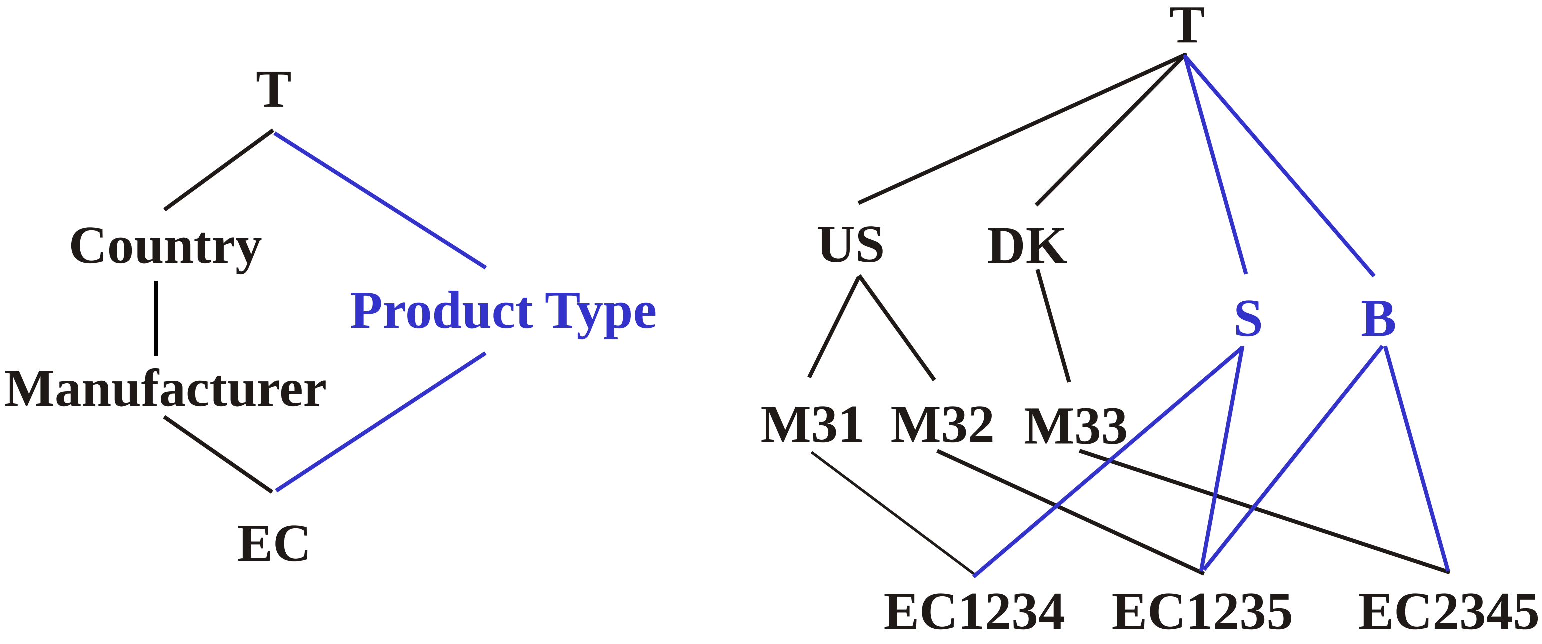
Decoration – How ?

Decoration using attributes: not powerful enough for complex data



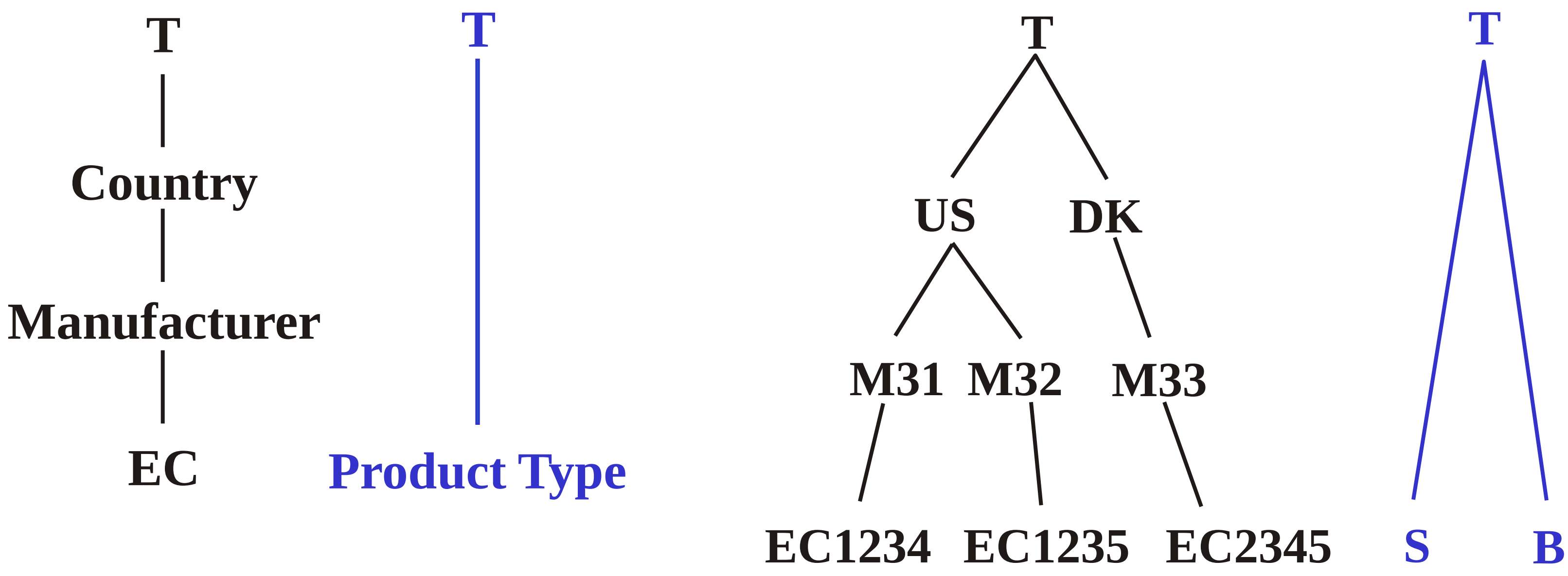
Decoration – How ?

New dimension hierarchy: dimensions cannot (always) be altered



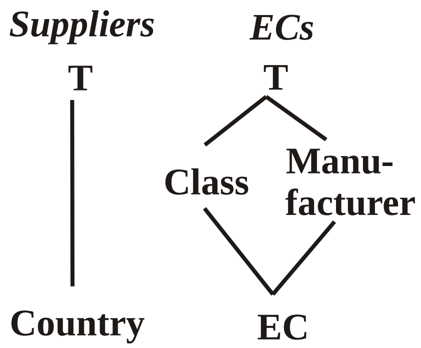
Decoration – How ?

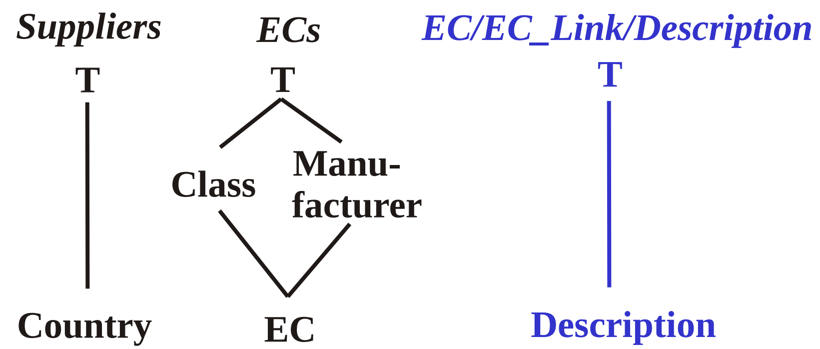
New dimension !



Decoration

Before *After*





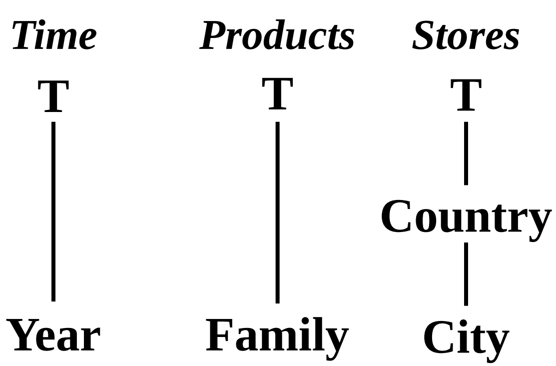
Cost	Country	EC
9840	US	EC1234
9480	UK	EC2345
32050	US	EC1235

Cost	Country	EC	Description
9840	US	EC1234	D-type flip-flop
9480	UK	EC2345	16-bit latch
32050	US	EC1235	16-bit flip-flop

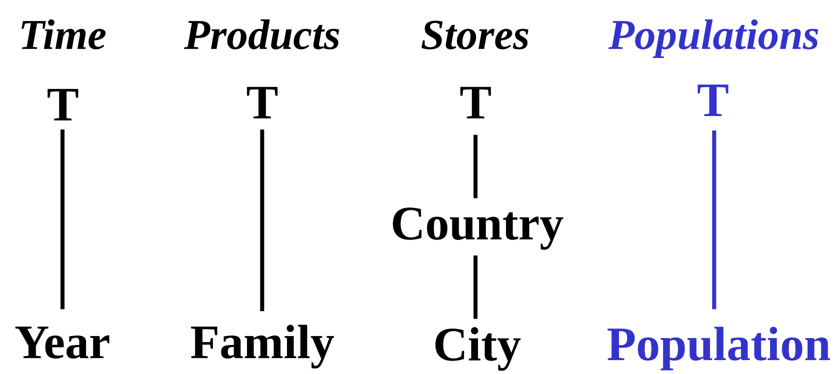
The Decoration Operator δ

City-Population-Cube = $\delta_{[\text{City, CityLink, Population}]}$(City-Cube)

City-Cube



City-Population-Cube



Price	Year	Family	City
3 mil.	2000	Jeans	København
2.7 mil.	1999	Shirts	Århus
4 mil.	2000	Jeans	Århus

Price	Year	Family	City	Population
3 mil.	2000	Jeans	København	1.300.000
2.7 mil.	1999	Shirts	Århus	210.000
4 mil.	2000	Jeans	Århus	210.000

Decoration Issues

Two main problems with external data

- No decoration values

Price	Year	Family	City	Population
2.7 mil.	1999	Shirts	Århus	210.000
4 mil.	2000	Jeans	Århus	210.000
1.2 mil.	2000	Shirts	Snave	N/A

- Several decoration values per dimension value

Price	Year	Family	City	Category
3 mil.	2000	Jeans	København	Industrial City
3 mil.	2000	Jeans	København	Capital
2.7 mil.	1999	Shirts	Århus	Industrial City
4 mil.	2000	Jeans	Århus	Industrial City

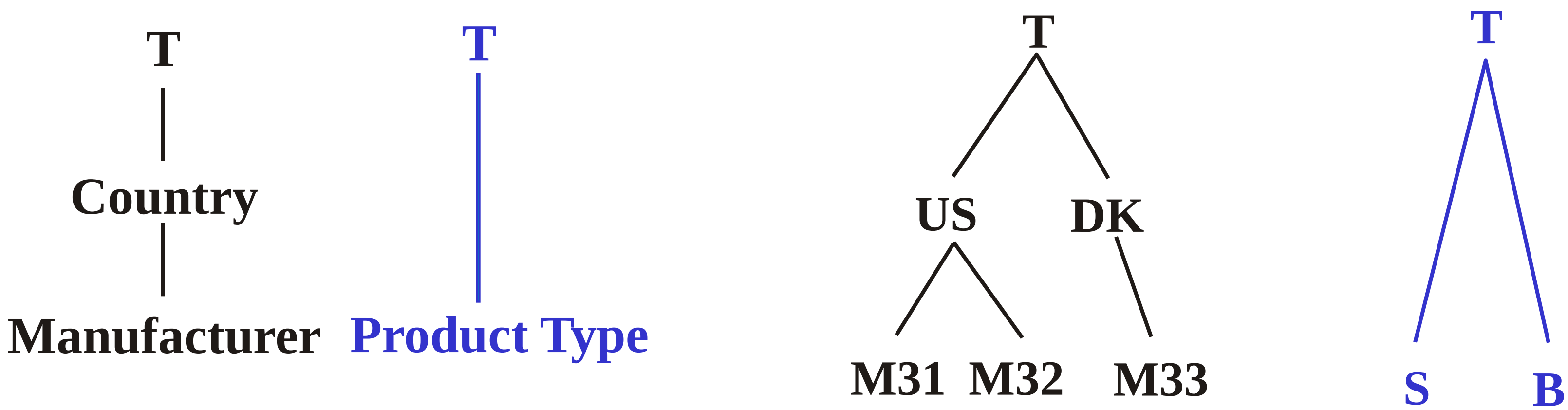
All?
One?
Combine?

Decoration Semantics

- *Selection* semantics
  - Predicate ”Category=Industrial City”
  - Should data for København be included ?
  - *all*: predicate must hold for all values (false for Kbh.)
  - ***any***: predicate must hold for at least one value (true for Kbh.)
- Three often-used *decoration* semantics
  - ANY: Pick an arbitrary node, summarizability ok
  - CONCAT: Concatenate string values into one, summar. ok
  - ALL: Use all nodes, good for grouping and selection, duplicates facts, summarizability violated
  - Formal semantics given in the paper
  - User chooses between these in query
  - More can be imagined

ANY Semantics

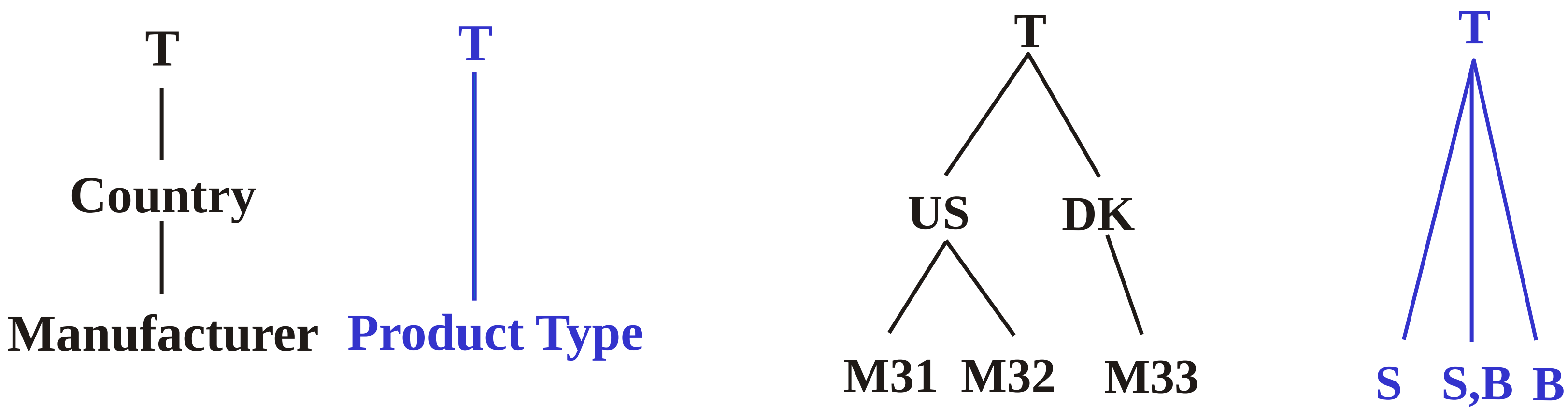
- M32 has two product types (S,B): choose S (randomly)



…	Cost	Manufacturer	Product Type	…
	1000	M31	S	
	2000	M32	S	
	3000	M33	S	

CONCAT Semantics

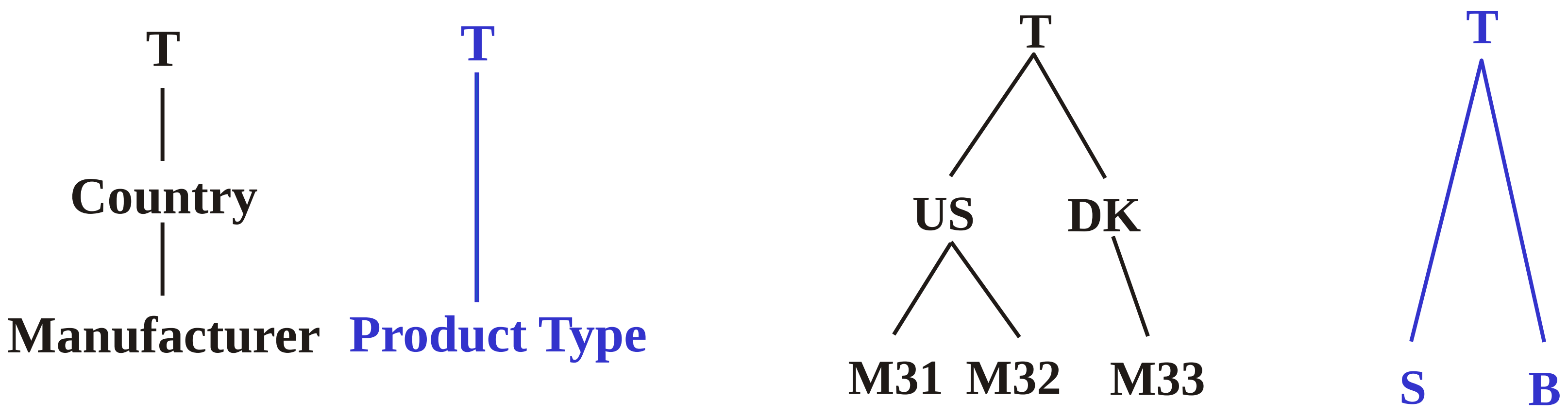
- M32 has two product types (S,B): use ”S,B”



…	Cost	Manufacturer	Product Type	…
	1000	M31	S	
	2000	M32	S,B	
	3000	M33	S	

ALL Semantics – Duplicate Facts ?

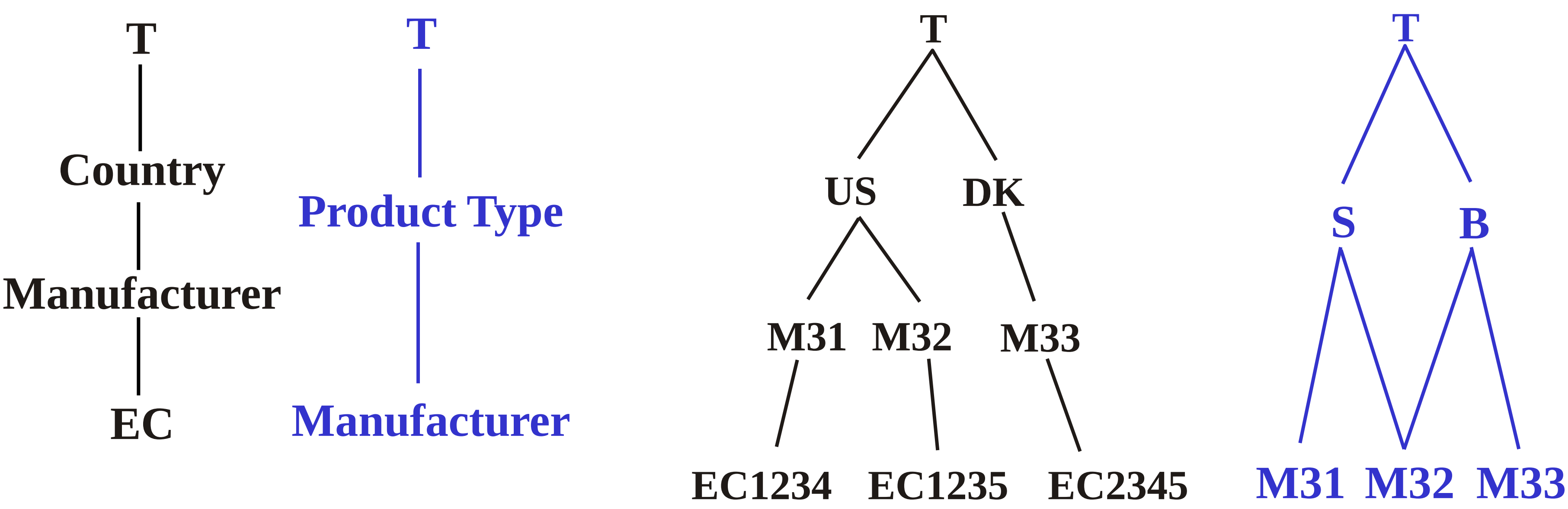
Destroys summarizability, bad behavior for decoration op.



…	Cost	Manufacturer	Product Type	…
	1000	M31	S	
	2000	M32	S	
	2000	M32	B	
	3000	M33	S	

ALL Sem.: Use Deco. Base Level !

Using Manufacturer as the base level in the new dimension means that **both** summarizability and flexibility is preserved

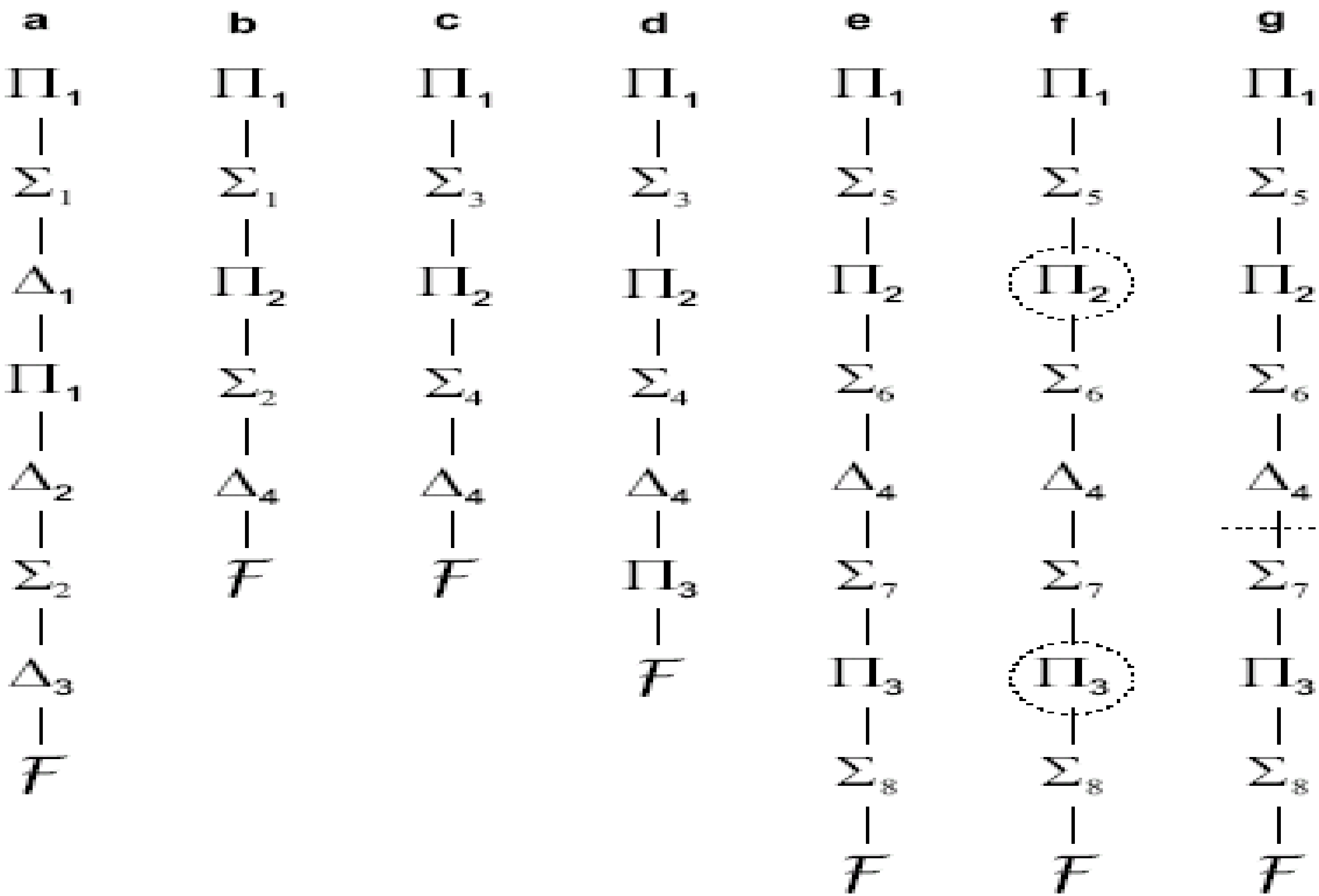


Transformation Rules

- 16 transformation rules given for algebra
- Allows query trees to be transformed (optimization)
- Focus on rules for decoration operator (1-8)
- Other rules similar to rel. algebra with GP
- Rule 9-16 not discussed due to space constraints
- Proof sketches in the paper

No.	Description
1	Remove redundant decoration above GP
2	Remove redundant decoration below GP
3	Commutativity of decoration and GP
4	Pushing GP below decoration
5	Commutativity of selection and decoration
6	Inlining of decoration in selection
7	Commutativity of decorations
8	Cascade of decorations
9	Commutativity of selection and GP
10	Pushing GP below selection
11	Commutativity of GP and selections with measure references
12	Cascade of selections
13	Commutativity of selections
14	Cascade of GPs
15	Redundant GP
16	Pushing GP below decorations and selections

Rewriting Queries: Push to OLAP



Query Optimization – Partitioning

$\Pi_{[Year,\ Family,\ Category]<SUM(Price)>}$

$\delta_{ALL[City,\ CityLink,\ Category]}$

$\sigma_{Population\ >\ 200.000}$

$\delta_{ALL[City,\ CityLink,\ Population]}$

Sales

$\longrightarrow$

$\Pi_{[Year,\ Family,\ Category]<SUM(Price)>}$

$\delta_{ALL[City,\ CityLink,\ Category]}$

$\sigma_{Population\ >\ 200.000}$

$\delta_{ALL[City,\ CityLink,\ Population]}$

$\Pi_{[Year,\ Family,\ City]<SUM(Price)>}$

Sales

Partitioning

Algorithm 1 Partitioning a Query Tree

1 PartitionQueryTree(RootNode)
2 RemoveAndPushDecorations(RootNode, $\emptyset$)
3 SplitSelections(RootNode)
4 LowerGP := FindLowerGP(RootNode)
5 PushGPDown(LowerGP.Child, LowerGP)
6 PushSelectionsDown(RootNode, $\emptyset$)
7 RemoveRedundantGPs(RootNode)

Rules for Decoration and GP

- Redundant decoration above GP
  - A *repeated* decoration above a GP containing it already can be removed or added
- Redundant decoration below GP
  - A decoration below a GP not including it as a GROUP BY level can be removed
- Commutativity of decoration and GP
  - Decoration and GP can be switched freely if the GP contains the starting level (or lower) of the decoration
- Pushing GP (partly) below decoration
  - Parts of a GP may be pushed below a decoration if the decoration can still be applied to the GP result

Rules for Decoration and Selection

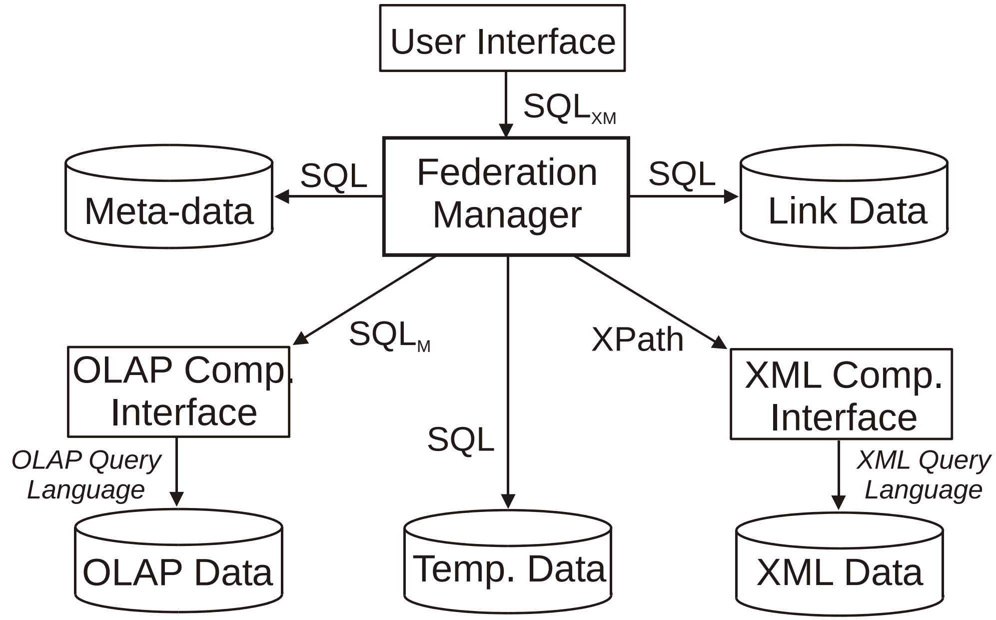
- Commutativity of selection and decoration
  - Selection commutes with decoration
    - if the selection does not refer to the decoration
    - if the cube has already been decorated with the same decoration
- Inlining of decoration in selection
  - A decoration can be integrated into a predicate by *inlining* literal data values from the base level
    - Creating a new predicate referencing only literals values from a dimension level
  - Example:
    - “[SupplierCities] IN ('Los Angeles','New York')” is translated to
    - “[Supplier] IN ('A.A', 'B.B')”
    - Which can be evaluated entirely in the OLAP component,

Rules for Decoration Only

- Commutativity of decorations
  - Decoration operators commute if one operator does not decorate the other
- Cascade of decorations
  - If a decoration is applied to an identical decoration, one of them can be removed

Prototype Architecture

- SQL_{XM} queries handled by Federation Manager (FM)
- XML Data stored in XML Component (Tamino)
- OLAP data stored in OLAP Component (MS Analysis)
- Link, Temp, and Meta-data stored in RDBMS
- FM fetches relevant data from components and processes SQL_{XM} queries using Temp data



Query Processing and Optimization

- Naive strategy
  - Fetch necessary XML data and store in Temp DB
  - Fetch necessary OLAP data and store in Temp DB
  - Join XML and OLAP data + perform XML-specific part of SQL_{XM} query
  - Problem: too much data transferred to Temp DB
- Rule-based optimization
  - Based on *algebraic transformation rules*
  - Partition query tree into OLAP part, XML part, and Temp part
  - Push as much evaluation to components as possible (always valid)
- Cost-based optimization
  - Inline literal XML data values in OLAP queries
  - Efficient fetch of XML data over XPath interfaces
  - Caching/Prefetching in this particular setting
  - Focus of this paper

Inlining

- Problem: decorations used for selections are expensive
  - A lot of low-level OLAP data must be transferred to Temp
- Solution: inlining
  - Get level expression predicate data from XML component
  - Transform predicate to refer to literal data values
  - Allows selection+aggregation to take place in OLAP component
- Predicate length will be linear/quadratic in no of dim values
- Queries may get too long
  - Split into partial queries
- Inlining is particularly effective for OLAP systems
  - Allows more aggregation in OLAP component

$\Pi_{\text{[Year, Family, Category]<SUM(Price)>}}$

$\delta_{\text{ALL[City, CityLink, Category]}}$

$\sigma_{\text{Population > 200.000}}$

$\delta_{\text{ALL[City, CityLink, Population]}}$

$\Pi_{\text{[Year, Family, City]<SUM(Price)>}}$

Sales

$\Pi_{\text{[Year, Family, Category]<SUM(Price)>}}$

$\delta_{\text{ALL[City, CityLink, Category]}}$

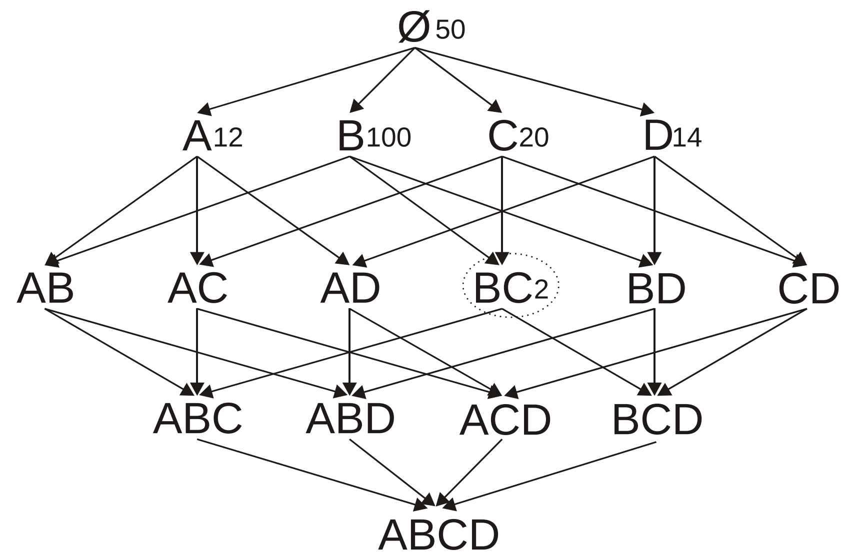
$\sigma_{\text{City IN (København, Århus)}}$

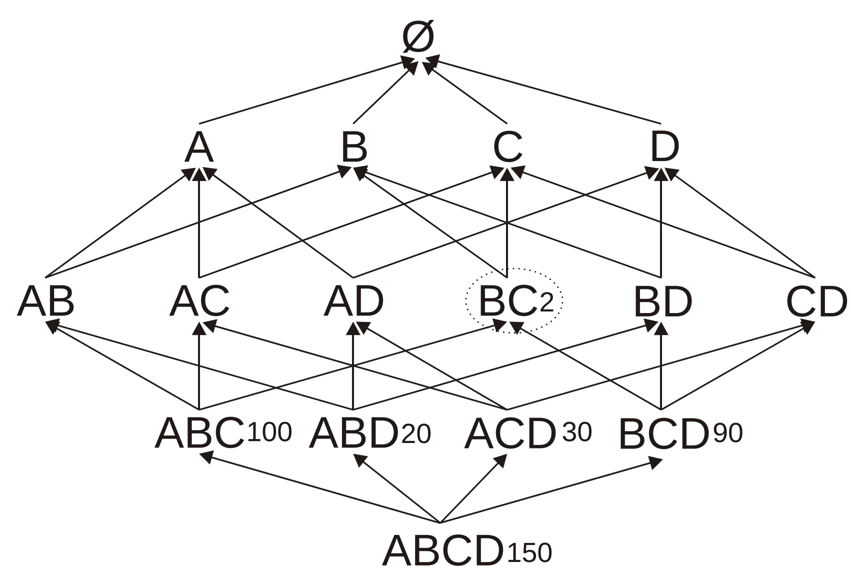
$\Pi_{\text{[Year, Family, City]<SUM(Price)>}}$

Sales

Inlining

- Problem: what predicates to inline?
  - Exponential number of possibilities
  - Theorem: choosing the best inlining strategy is NP-hard
  - No obvious best choice
  - Often, an exhaustive search is feasible
  - Otherwise, two heuristics are used
- Top down (see right above)
  - Start from no inlining and add
  - Will often fail
- Bottom up
  - Start with all inlined and remove
  - Less likely to fail
- General strategy
  - Generate all bottom up, except if cost goes up very much





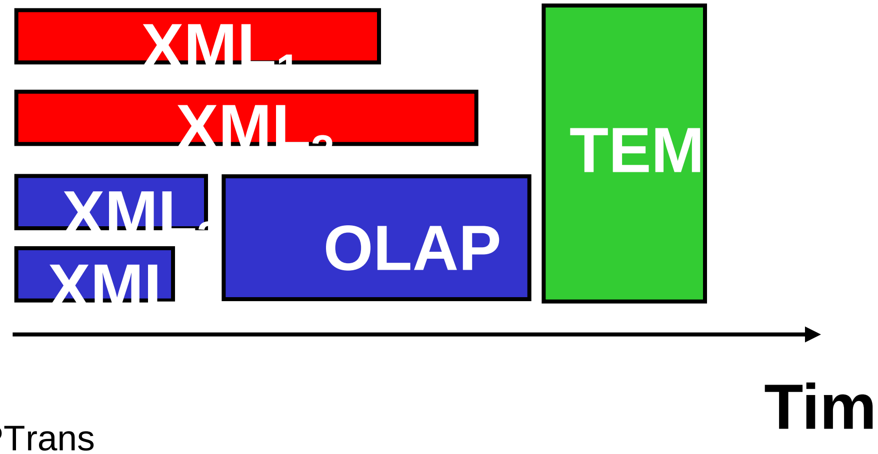
Optimizing XML Retrieval

- XML component queries retrieve (locator,level exp) tables
  - Example tuple: (Aalborg, 160,000)
- Easy to do in XQuery, but not in XPath
  - XPath can fetch only entire *existing* nodes
  - Many systems provide only an XPath interface
- One query for *every* dimension value?
  - Often too much overhead because of many queries
  - Combining with OR/IN not possible as XPath result is *unordered*
- Common parent of locator and level exp can be fetched
  - Larger data volume may have to be returned
- Data for *several* level exps may be fetched together by *combining* XPath expressions
  - Algorithm for doing this in full paper
- Cost analysis chooses between these options

Caching and pre-fetching

- General technique specialized for this particular domain
  - Caching/prefetching not feasible for very *dynamic* data
- Given a new query tree the cache is checked
  - Identical query tree found: use cache (always fastest)
  - Lower part of query tree found: compare cost of using cache to not using it (not using cache may be faster due to OLAP pre-aggregation)
- Caching and prefetching is done for *component* queries only
  - Entire federation queries take up too much space and has too little probability of reuse
- XML cache
  - Raw XML data + temporary XML mapping tables
- OLAP cache
  - Intermediate OLAP result tables
- LRU cache replacement strategy used

Cost Model



- Federation cost model
  - Parallel exec of queries 1-4
  - OLAP waits for 3+4 (inlined)
  - OLAP query not split
  - Results combined in Temp
- OLAP cost model
  - $Cost_{OLAP} = t_{OLAPOH} + t_{OLAPEval} + t_{OLAPTrans}$
  - Based on network+disk rate, selectivity, fact size, rollup fraction, cube size, and possibilities for using pre-aggregation
- XML cost model
  - Based on assumption of *constant data rate* from a given document
  - $Cost_{XML} = t_{XMLOH} + t_{XMLProc}$
- Temp component used standard relational cost model
- All models adapt based on real execution times
- Details + parameter estimation in another paper

Introduction

- Web Ontology Language (OWL) has gained popularity for publishing and sharing ontologies.
- The OWL/RDF data takes the form of (subject, predicate, object)triple.
- The triples typically are stored in specialized storage engines called *triple-stores*.
- A need is to insert and retrieve triples efficiently (bulk-operation), but not advanced features such as logical inference.
- For the basic storage of triple data, even the subset of OWL-Lite is enough.

We implement 3XL, a DBMS-based triple-store, which supports bulk operations for OWL-Lite data.

Overview of 3XL System

- Automate database schema generation based on OWL-Lite ontology.
- Use the features of object-relational DBMS such as inheritance.
- Extensively use bulk techniques to speed up the data loading.
- Support different query patterns, and efficient bulk retrieval.
- Provide application programming interface (API), graphical and command line interfaces.

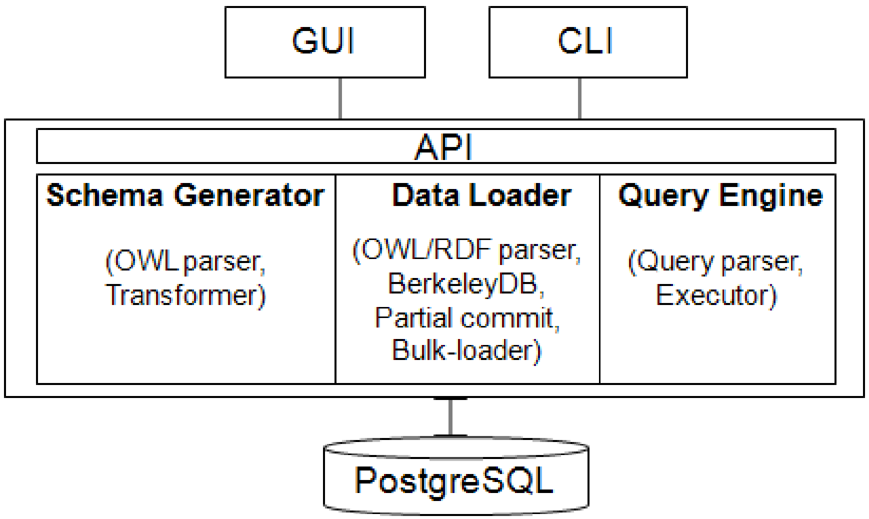


Fig. 1: Architecture

Specialized Data-dependent Schema

- Inheritance database schema is generated based on the hierarchical ontology classes.
- A table (called *class table*) is create for each OWL class, and the table attributes are for the OWL properties.
- A multi-valued property is generated as a separate table, or "in-line" array attribute in a class table.
- Overflow table holds triples not described by ontology
- Map table: find id+class table

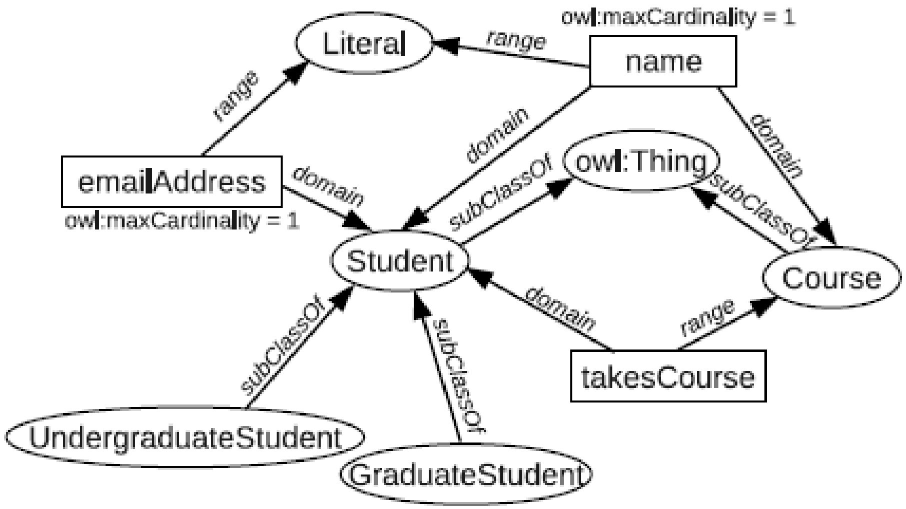


Fig. 2: OWL-Lite Ontology

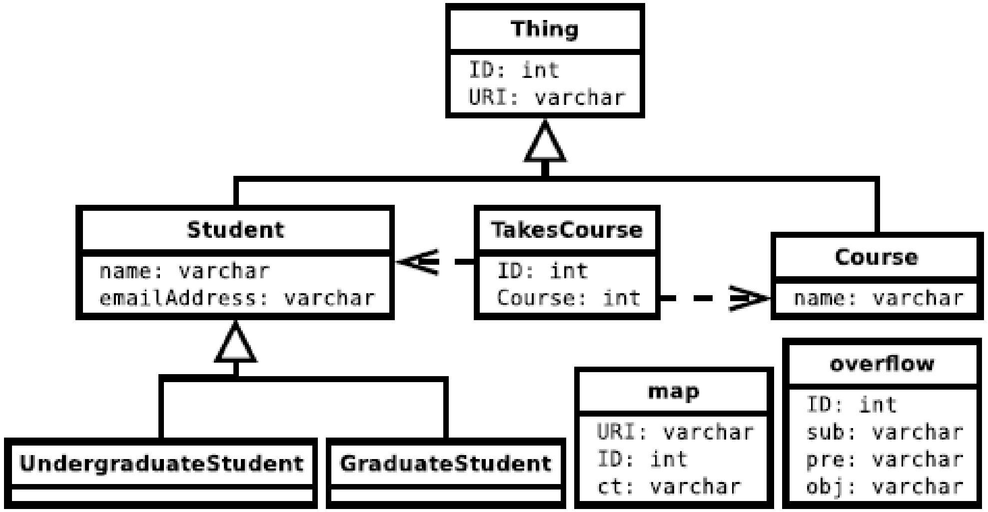


Fig. 3: Generated database schema

Triple Loading

- Triples with a common subject are gathered in a value holder.
- Berkeley DB speeds up value holder generation.
- Value holders are kept temporarily in a buffer, and loaded to the database when the buffer is full.
- Data is bulk-loaded, instead INSERTed.
- Considering data locality, only the LRU value holders in the buffer are loaded, called *partial commit* .

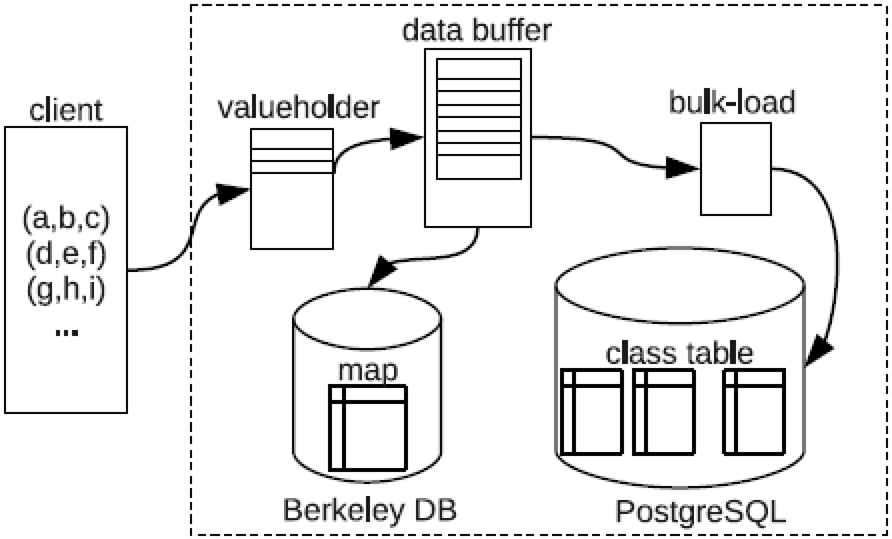


Fig. 4: Data flow of triple loading

Triple Loading - An Example

An example of value holder generation, and the loading:

- The triples with a same resource are made into a value holder, which is corresponding a record in a class table.
- The value holders for a class table are exported to a CSV le, and loaded to the class table by bulk-loader.
- As the triples for a class table are loaded together, it is more efficient.

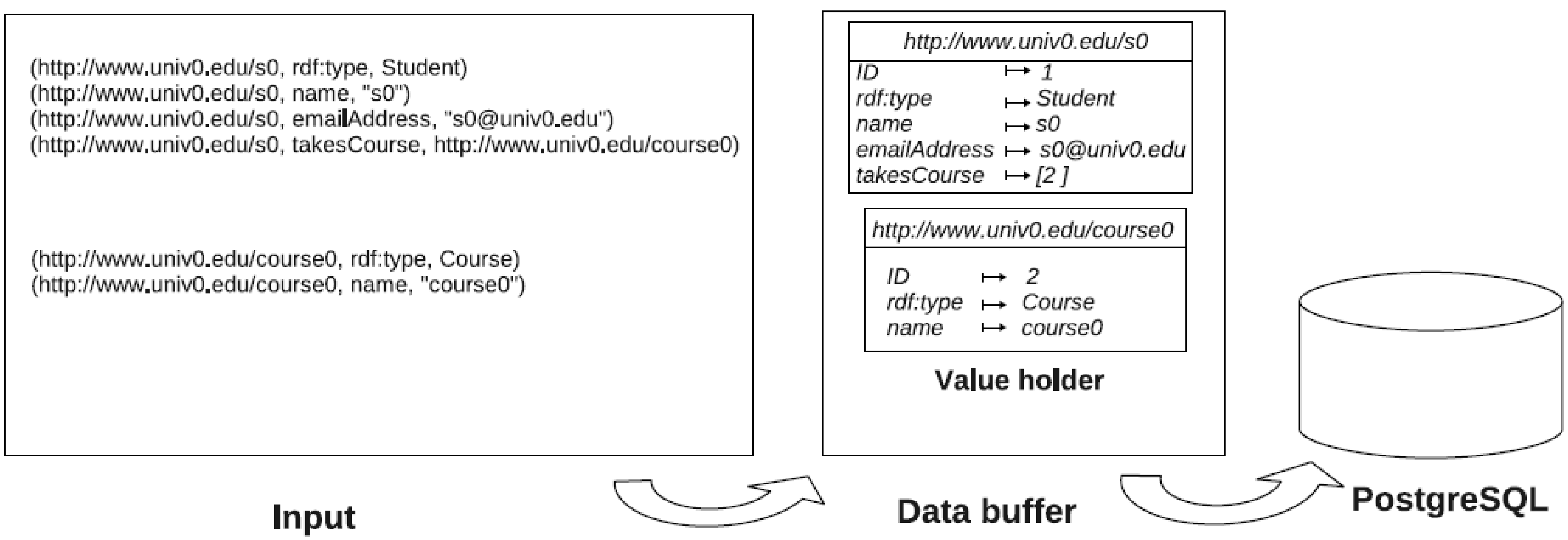


Fig. 5: Data flow of triple loading

Triple Queries

3XL supports two classes of queries:

1) *Point-wise query*, which is expressed in triple format. It supports the following eight query patterns (* matches any): (s, p, o), (s, p, *), (s, *, o), (s, *, *), (*, p, o), (*, p, *), (*, *, o) and (*, *, *).

2) *Composite queries*, which consist of several linked query triples. It supports chain, star and their combined patterns.

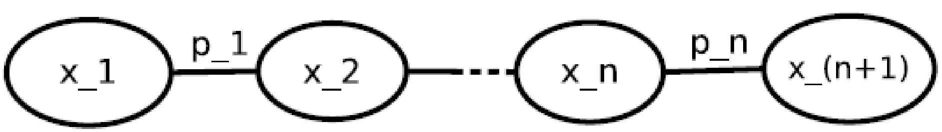


Fig. 6: C**hain query pattern**

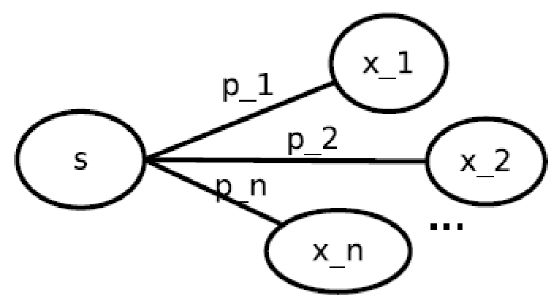


Fig. 4: Star query pattern

daisy

Triple Queries - An example

To answer a query, the query triples are first translated into a SQL statement, then executed by the query engine.

Composite Query	SQL Statement
(?X; rdf : type; GraduateStudent) (?Y; rdf : type; course) (?X; name; s0) (?X; takeCourse; ?Y)	SELECT GraduateStudent.uri AS x, Course.uri AS y FROM GraduateStudent, TakesCourse, Course WHERE GraduateStudent.ID = TakesCourse.ID AND TakesCourse.takesCourse = Course.id AND GraduateStudent.name = s0

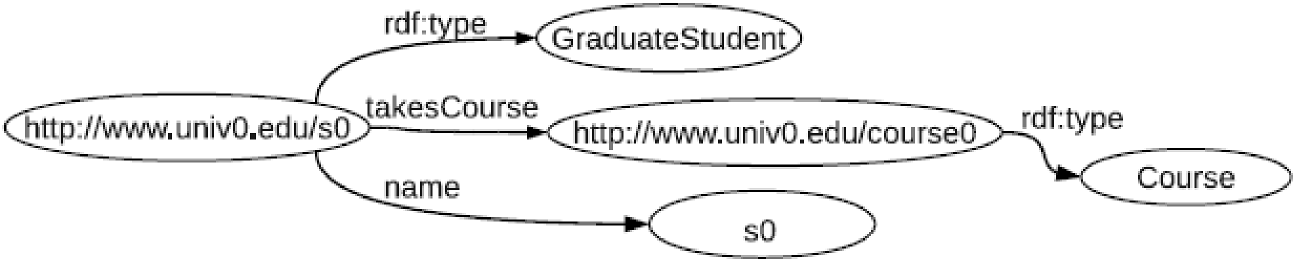


Fig. 7: Query results

3XL Summary

- 3XL is DBMS-based and uses a specialized data-dependent schema derived from an OWL-Lite ontology.
- 3XL uses advanced object-relational features of the underlying database (in this case PostgreSQL), such as table inheritance and arrays as "in-lined" attribute values.
- 3XL makes extensive use of bulk loading techniques that speed up bulk operations significantly.
- 3XL supports very efficient bulk retrieval for point-wise queries where the subject and/or the predicate is known.
- 3XL has loading and query performance comparable to the best file-based solution, and provides the flexibility as it is DBMS-based, such as integrating with the data from other sources.

- Can load 100 mio. EIAO triples in 1 hour (*days* on 3store)

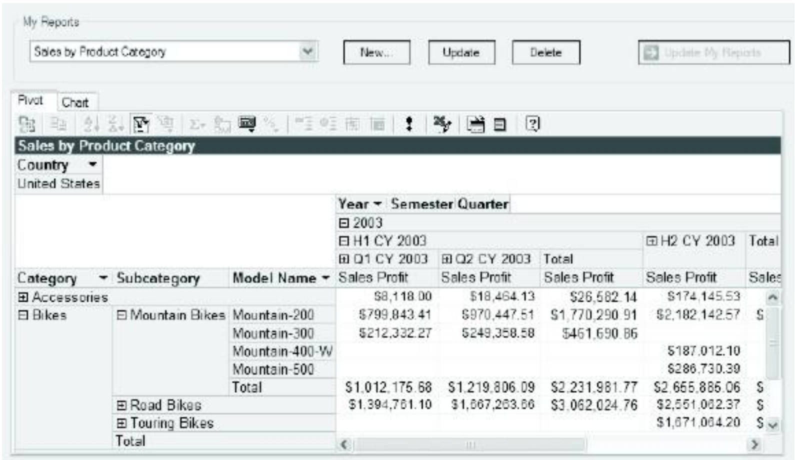
Talk Overview

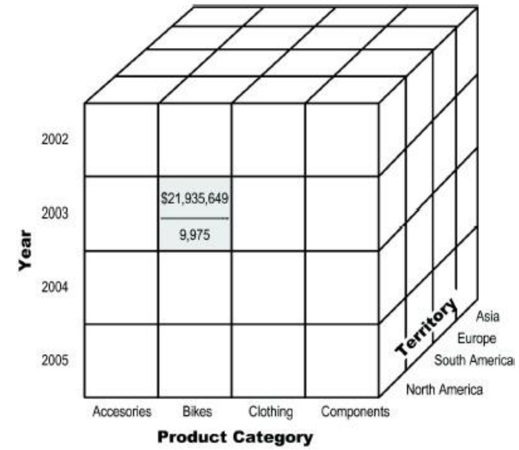
- Multidimensional modeling recap
  - Cubes, dimensions, measures, …
- Complex multidimensional data
  - Modeling
  - Performance techniques
- Complex spatial multidimensional data
- Integrating cubes and XML
- Semantic web warehousing
- **Integrating cubes and text**
- Multidimensional music data

Motivation

- Most of the world's information exist in *text*
- *Structured data* is only a small part
- This even more true for the data on the web
- But how to get hold of text data ?
- How to analyze text data ?
  - Text cubes
- How to integrate text data with multidimensional DW data?
  - Contextualized data warehouses

Data Warehousing + OLAP

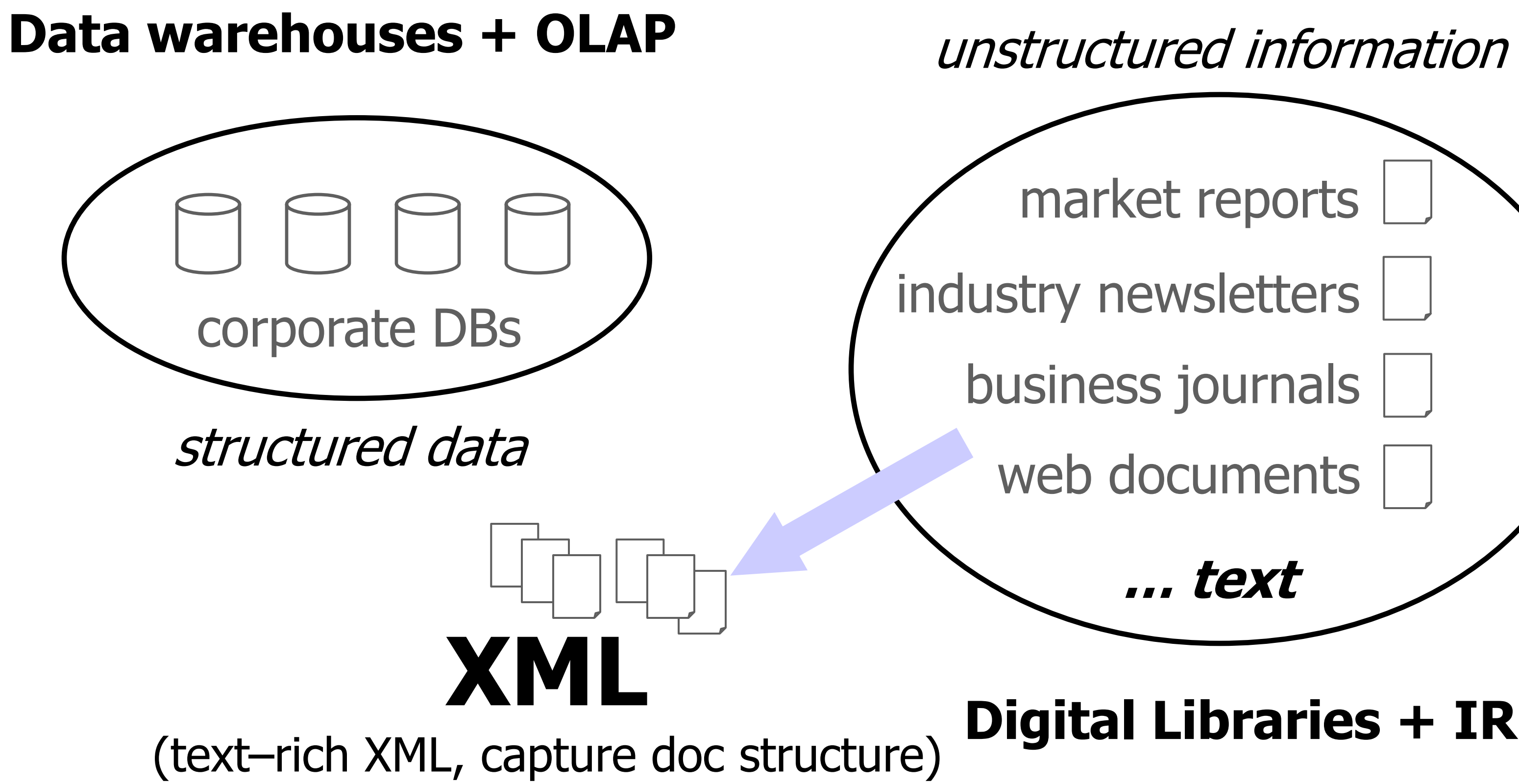




How to combine??

Information Retrieval

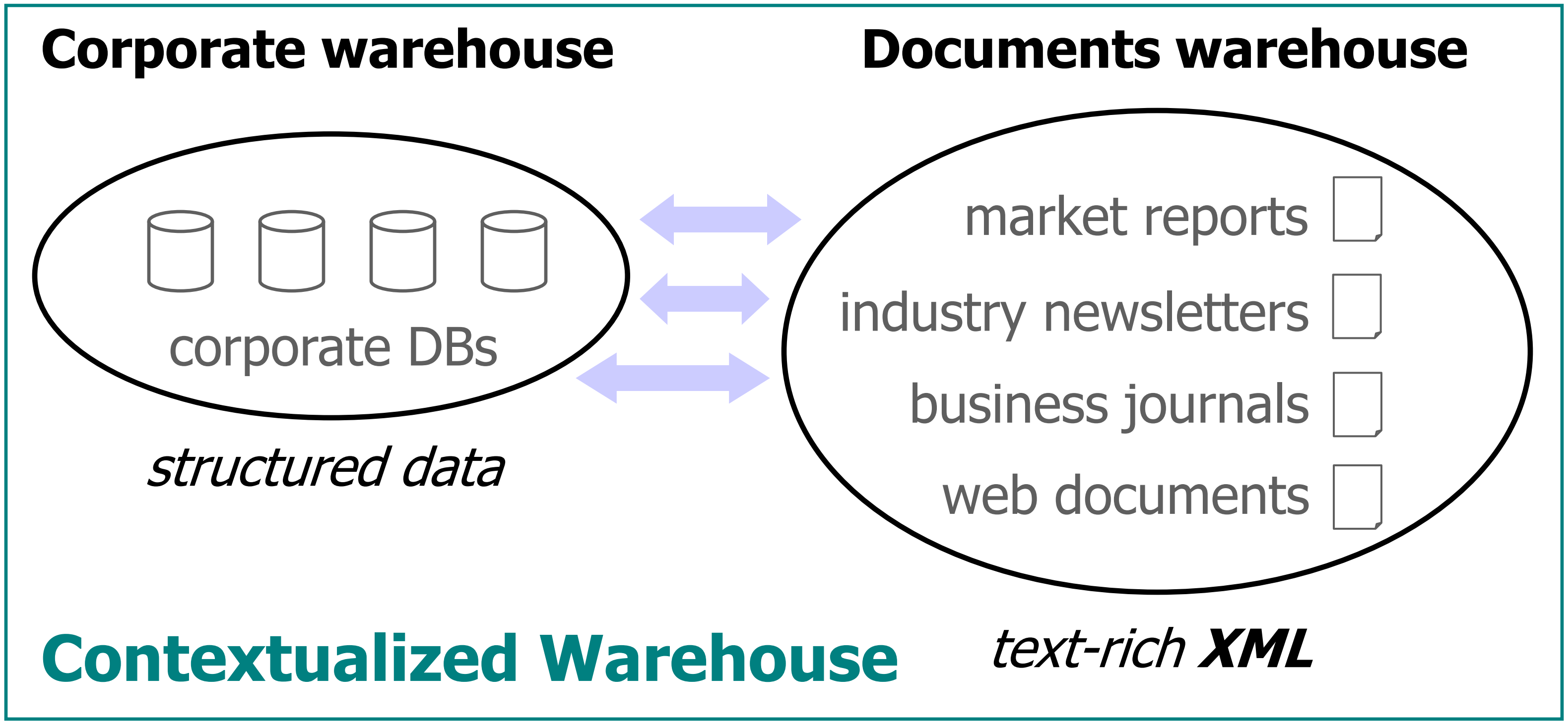
The Contextualized Warehouse



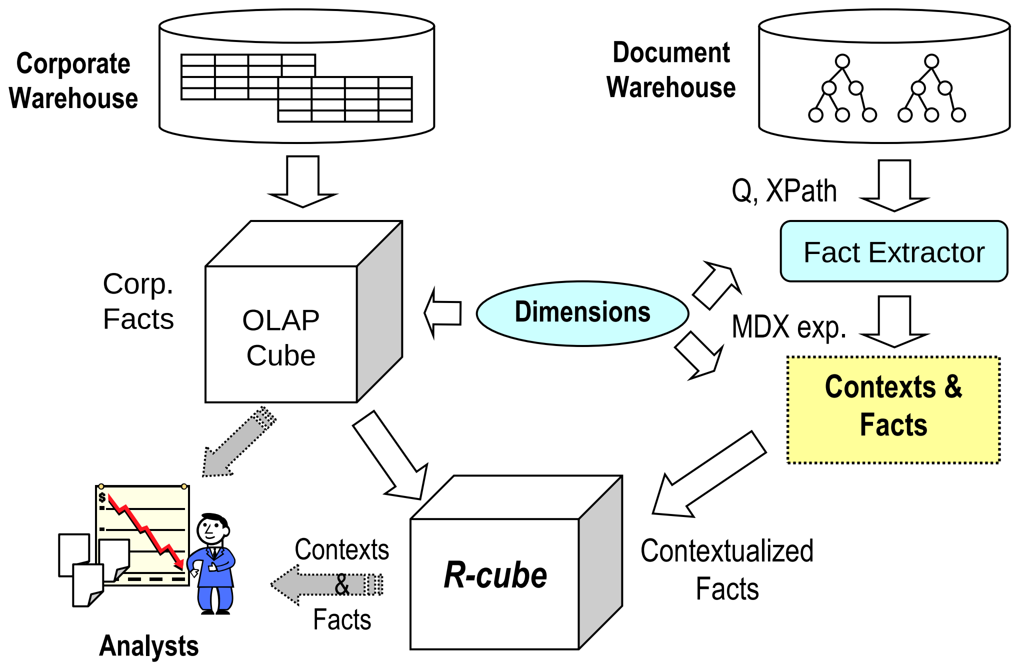
The documents explain the circumstances (the *context*) of the corporate facts

What is a Contextualized Warehouse?

A *Contextualized Warehouse* is a new kind of decision support system that allows users to obtain strategic information by combining all their sources of structured data and unstructured documents, and by analyzing the integrated data under different contexts.



Contextualized Warehouse Architecture

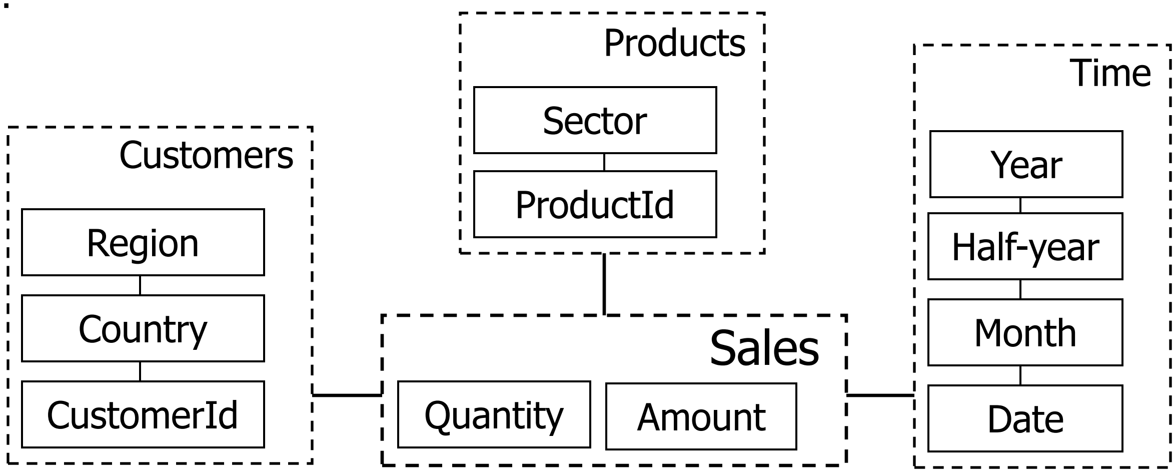


R-cube: relevance cube

Building R-Cubes

<u>Example</u>: vegetable oil by products provider

- Corporate warehouse: Sales database
  - *Products* : *fo1*, *fo2* (*food* sector) *he1* and *he2* (*healthcare*)
  - *Customers,* organized into *Countries* and *Regions* (eg. *Southeast Asia*)
  - …



- Document warehouse: XML business newspapers gathered from the Internet

Building R-Cubes

<u>Example</u>: Analysis of the of the sales of food products in 1998, under the context of a financial crisis

- *"Q = financial, crisis" (IR query)*
- *"XPath = /business_newspaper/economy/article//" (doc part)*
- *"MDX = (Products.[food], Customers.Country,*
  Time.[1998].Month, SUM(Amount))" (DW query)

Building R-Cubes

Example: Analysis of the of the sales of food products in 1998, under the context of a financial crisis

- ***Q = financial, crisis***
- ***XPath = /business_newspaper/economy/article//***
- *MDX = (Products.[food], Customers.Country, Time.[1998].Month, SUM(Amount))*

1.- **Document Warehouse.** *Q* and *XPath* are evaluated, resulting in a set of document fragments and their relevance to *Q*

$d_7^{0.08}$

```
<business_newspaper date="Dec.1,1998">
<economy>
<article> ...
<paragraph>
The financial crisis in Southeast Asian countries, has mainly affected
companies in the food market sector. Particularly, Chicken SPC Inc.
has reduced total exports to $1.3 million during this half of the year, from
$10.1 million in 1997. </paragraph> ...
```

Building R-Cubes

2.- **Fact Extractor.** parses document fragments and returns the set of facts described by each document, along with their frequency

```
<business_newspaper date="Dec.1,1998">
<economy>
<article> ...
<paragraph>
The financial crisis in Southeast Asian countries, has mainly affected
companies in the food market sector. Particularly, Chicken SPC Inc.
has reduced total exports to $1.3 million during this half of the year, from
$10.1 million in 1997. </paragraph> ...
```

$d_7^{0.08}$

- Applies Information Extraction techniques
- Dimension values are identified in the text
  - *Customer.Region = Southeast Asia*
  - *Products.Sector = food*
  - *Time.Half-year = 1998/2nd half*
- … and grouped into facts

(*Products.Sector = food, Customer.Region = Southeast Asia, Time.Half-year = 1998/2nd half*)

Building R-Cubes

- Dimension value frequency is also recorded (determines the importance of the fact in the document)

  (*Products.Sector* = *food*, *Customer.Region* = *Southeast Asia*, *Time.Half-year* = *1998/2nd half*)

  frequency = 3

- Characteristics of the identified facts:
  - Imprecise dimension values:

    Customer.Region = *Southeast Asia*, *Time.Half-year* = *1998/2nd half*

  - Incomplete facts (some dimensions are missing)

Building R-Cubes

- ***MDX = (Products.[food], Customers.Country, Time.[1998].Month, SUM(Amount) > 0)***

3.- **Corporate Warehouse.** (possibly in parallel to 1 and 2) *MDX*

Products.ProductId	***Customers.CustomerId***	***Time.Date***	***Quantity***	***Amount***
(he1	*Y Corp.*	*5th Jul.1998*	*7,300*	*200,000$)*
(fo1	*X Ltd.*	*11th Nov.1998*	*3,000*	*20,000$)*
(fo1	*X Ltd.*	*7th Dec.1998*	*2,500*	*15,000$)*
…				

Products.ProductId	***Customers.Country***	***Time.Month***	***SUM(Amount)***
(fo1	*Japan*	*1998/10*	*300,000$)*
(fo2	*Korea*	*1998/11*	*400,000$)*
…			

Building R-Cubes

4.- Each document fragment is related to those facts of the corporate warehouse whose dimension values can be *rolled-up* or *drilled-down* to some (possibly imprecise or incomplete) fact described by this document

Products.ProductId,	**Customers.Country**,	**Time.Month** ,	**SUM(Amount)**
(*fo1*	, *Japan*	, *1998/10*	, *300,000$*)
(*fo2*	, *Korea*	, *1998/11*	, *400,000$*)

can be rolled-up to

(*Products.Sector = food, Customer.Region = Southeast Asia, Time.Half-year = 1998/2*[nd] *half*)

So, they are related with the example document ($d_7^{0.08}$)

Products.ProductId,	**Customers.Country**,	**Time.Month** ,	**SUM(Amount)**,	**Ctxt**
(*fo1*	, *Japan*	, *1998/10*	, *300,000$*	, $d_7^{0.08}$)
(*fo2*	, *Korea*	, *1998/11*	, *400,000$*	, $d_7^{0.08}$)

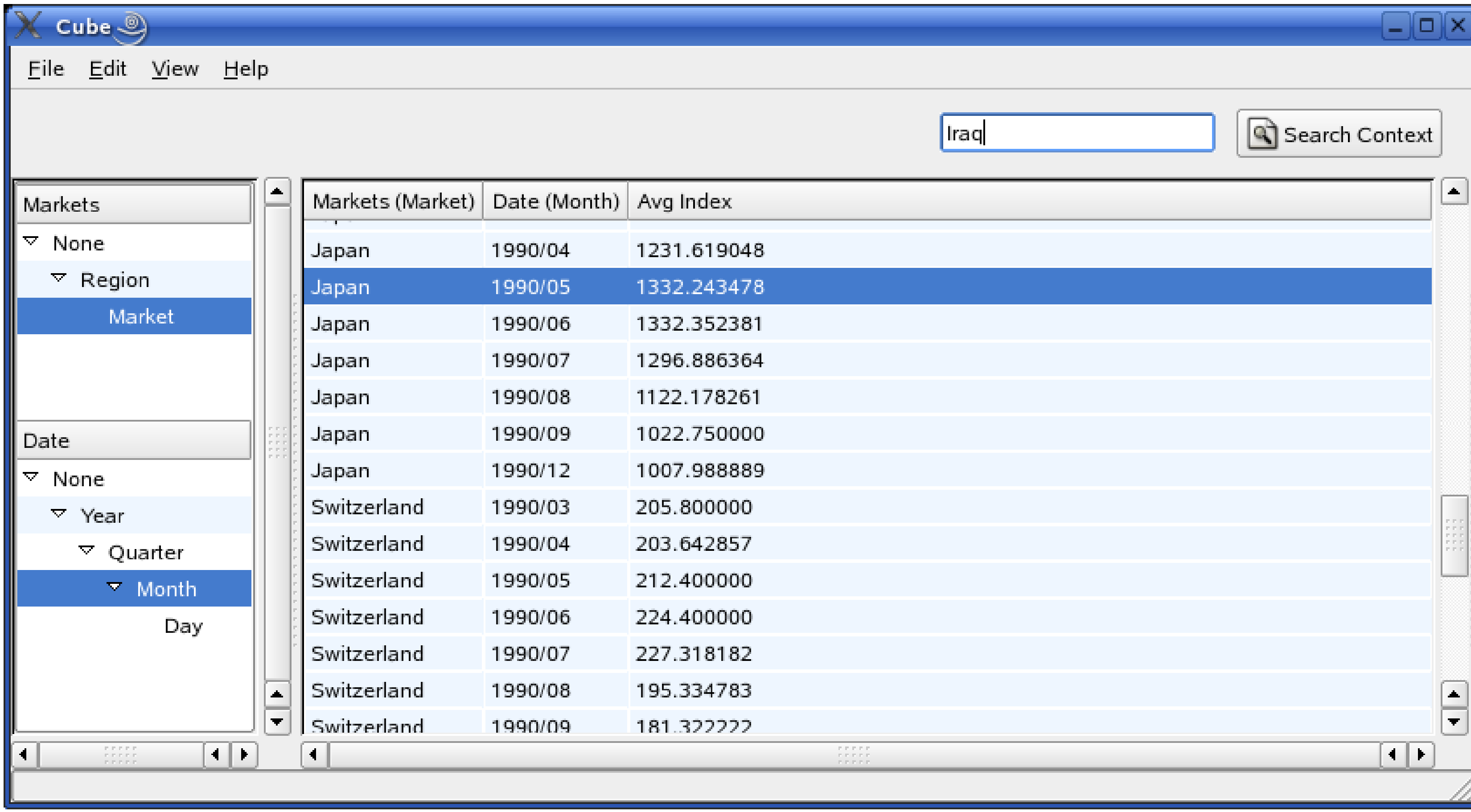
Building R-Cubes

5.- Apply formal "fact relevance calculus" … and the result is an *R-cube*

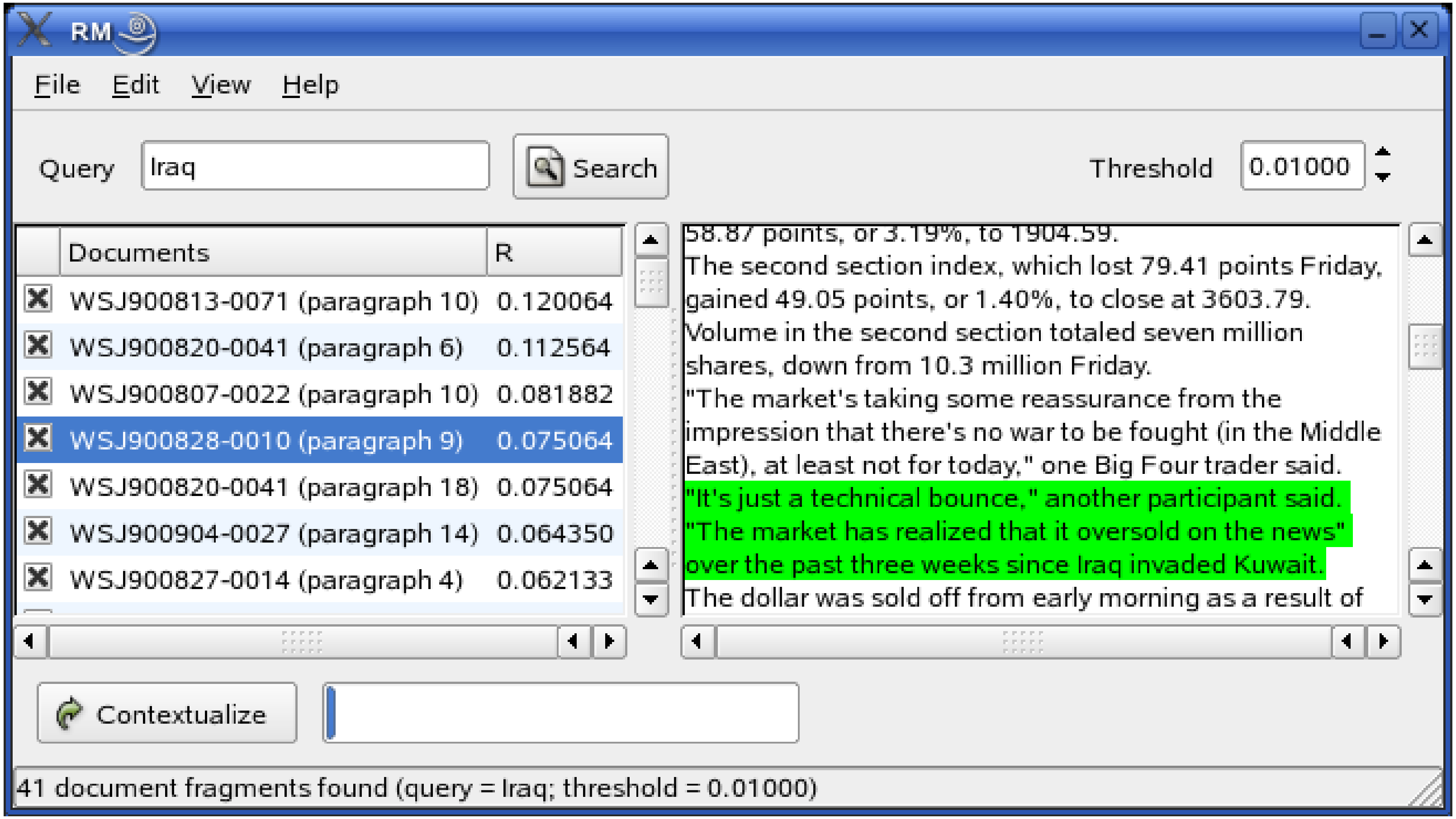
Products.ProductId	***Customers.Country***	***Time.Month***	***SUM(Amount)***	***R***	***Ctxt***
f_1 = (*fo1*	, *Cuba*	, *1998/03*	, *4,300,000$*	, *0.05*,	$d_{23}^{0.005}\, d_{47}^{0.005}$)
f_2 = (*fo2*	, *Japan*	, *1998/02*	, *3,200,000$*	, *0.1* ,	$d_{50}^{0.02}$)
f_3 = (*fo2*	, *Korea*	, *1998/05*	, *900,000$*	, *0.2* ,	$d_{84}^{0.04}$)
f_4 = (*fo1*	, *Japan*	, *1998/10*	, *300,000$*	, *0.4* ,	$d_{123}^{0.04}\, d_7^{0.08}$)
f_5 = (*fo2*	, *Korea*	, *1998/11*	, *400,000$*	, *0.25*,	$d_7^{0.08}\, d_{69}^{0.01}$)

f_4 has the highest relevance, and the context is d_{123} and d_7

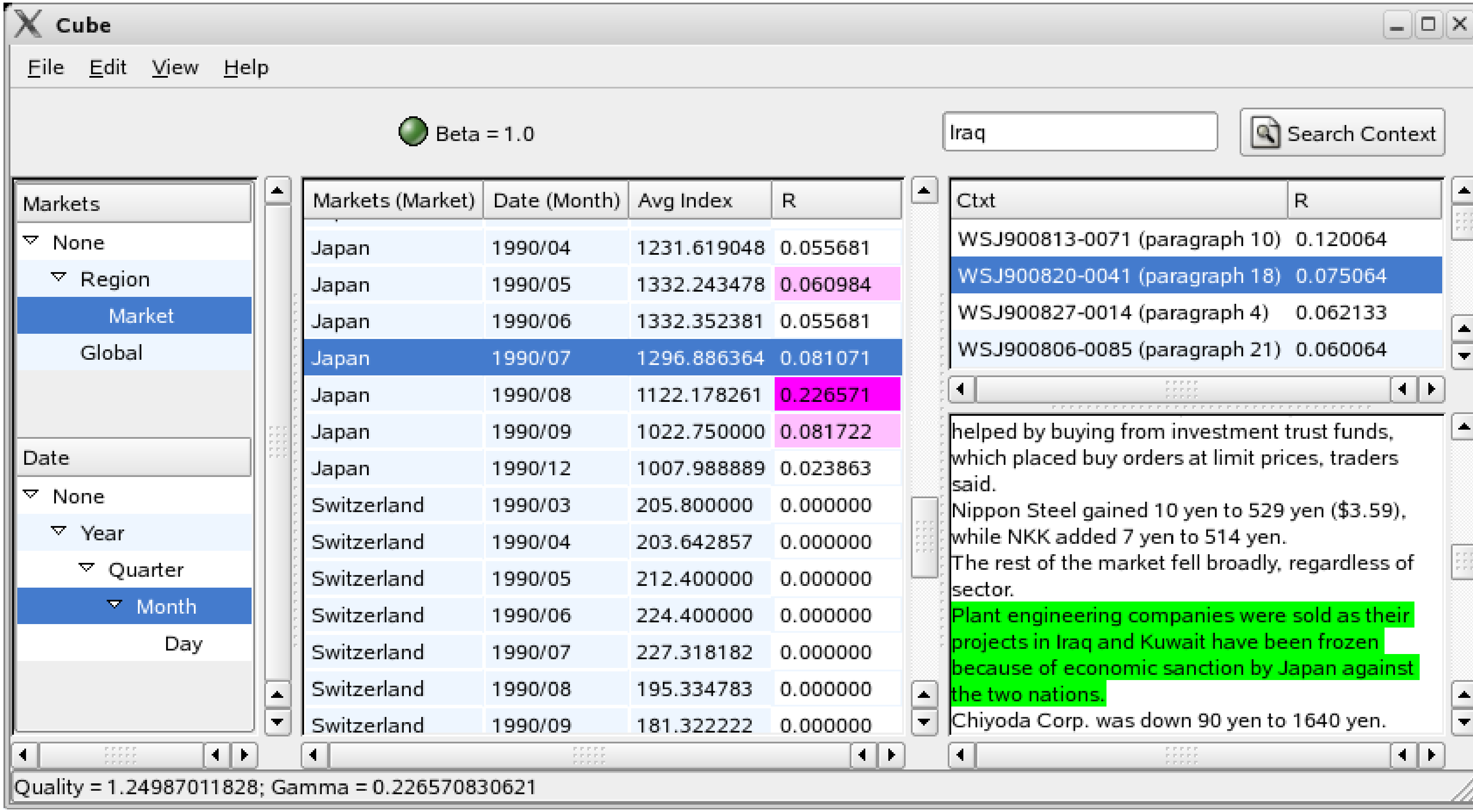
Prototype: issue IR query



Prototype: choose IR results



Prototype: Explore R-Cube



Talk Overview

- Multidimensional modeling recap
  - Cubes, dimensions, measures, …
- Complex multidimensional data
  - Modeling
  - Performance techniques
- Complex spatial multidimensional data
- Integrating cubes and XML
- Semantic web warehousing
- Integrating cubes and text
- **Multidimensional music data**

Multidimensional Bitmap Indices

- A B-tree index stores a list of RowIDs for each value
  - A RowID takes ~8 bytes
  - **Large** space use for columns with **low cardinality** (gender, color)
  - Example: Index for 1 bio. rows with gender takes 8 GB
  - Not efficient to do ”index intersection” for these columns
- Idea: make a ”position bitmap” for each value (only two)
  - Female: 01110010101010…
  - Male: 10001101010101…
  - Takes only (no. of values)*(no. of rows)*1 bit
  - Example: bitmap index on gender (as before) takes only 256 MB
  - **Very** efficient to do”index intersection” (AND/OR) on bitmaps
  - Can be improved for for higher cardinality using compression
- Supported by some RDBMSer (DB2, Oracle)

Using Bitmap Indices

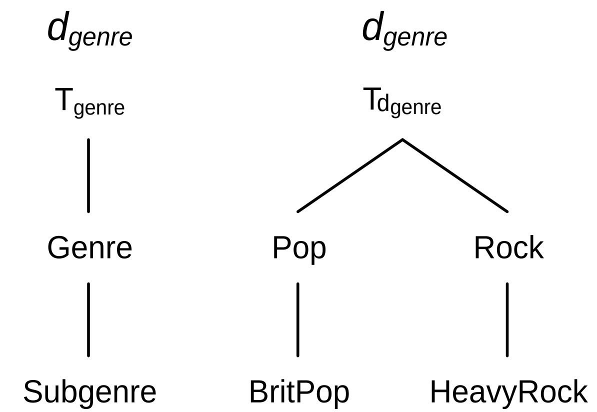
- Query example
  - Find female customers in Jutland with blond hair and blue eyes
  - Female: 01010101010
  - Jutland: 00000011111
  - Blond: 10110110110
  - Blue: 01101101111
  - Result 00000000010 – use AND, only one such customer
- Range queries can also be handled
  - …and Salary BETWEEN 200,000 AND 300,000
  - 200-250,000: 001001001
  - 250-300,000: 010010010
  - OR together: 011011011
  - Use as regular bitmap

Compressed Bitmaps

- Problem: space use
  - With *m* possible values and *n* records: n*m bits required
  - However, probability of a 1 is 1/*m* => very few 1's
- Solution: compressed bitmaps
  - Run-length encoding
  - A *run* is *i* 0's followed by a 1
  - Concatenating binary numbers won't work – no unique decoding
  - Instead, determine *j* – number of bits in binary representation of *i*
  - Encode as "<*j*-1 1's>"+"0"+"<*i* in binary>"
  - Encode next run similarly, trailing 0's not encoded
  - Example: 000000010000 encoded as 110111
  - *j>1* => first bit of *i* is 1 – this bit can be saved – encoded as 11011
  - Decoding: scan bits to find *j*, scan next j-1 bits to find *i,* find next 1
  - Example: *j* = 3, *i* = 7 (11) => bitmap = 00000001 + 0000 (trailing)

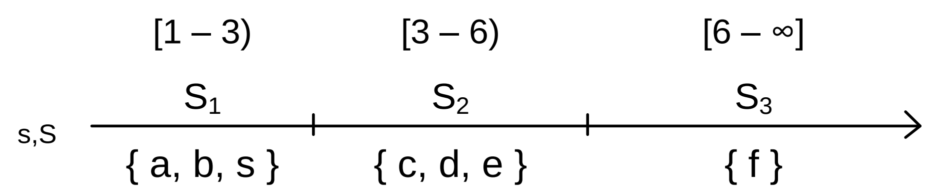
Multidimensional Music Data

- Metadata dimension
  - Posets
  - Support for irregular hierarchies



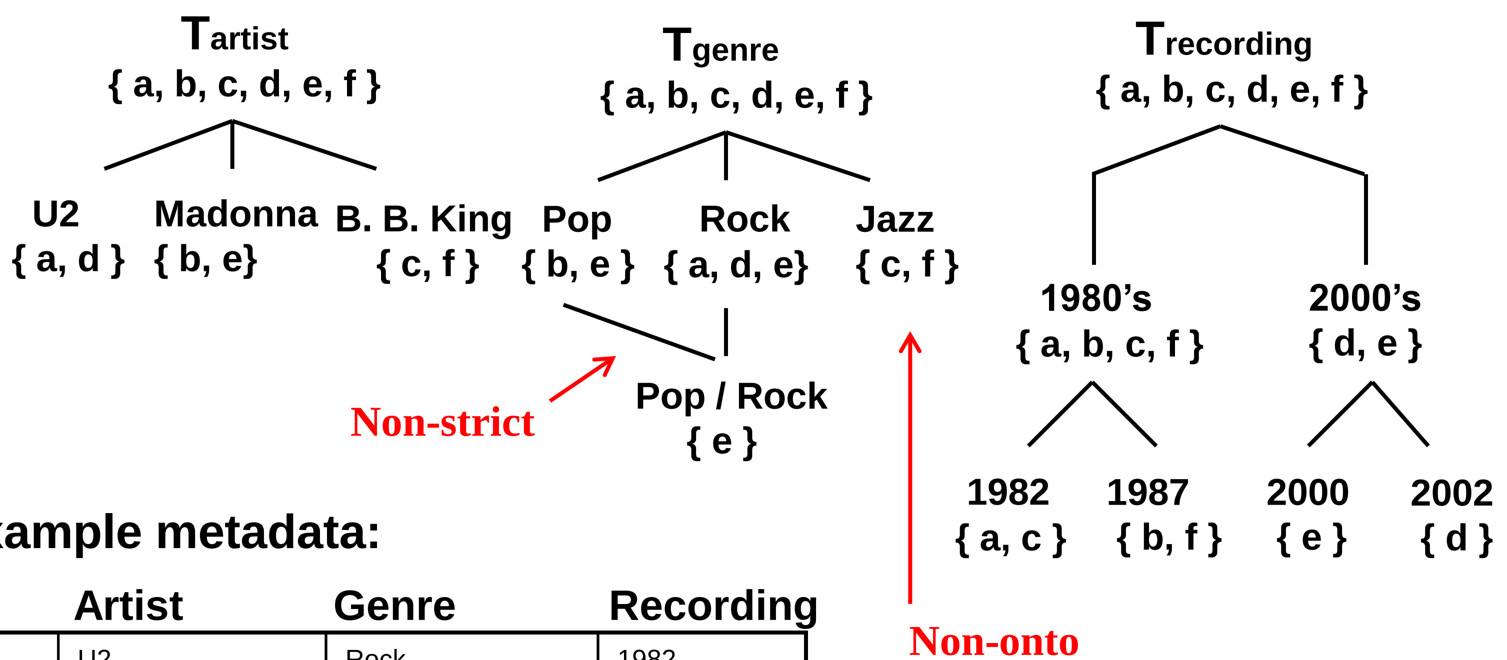
- Song **facts** defined by:
  - Metadata (artist name, genre, …)
  - Music content (features)
  - Non-metric distance function

    dist (x,x) = 0



- Distance store
  - Complete disjoint partitioning of distance domain for seed song *s*
  - Each partition is unique
  - Non-overlapping distance intervals
  - All songs within a partition are ”equally good”

Metadata Dimension Instances

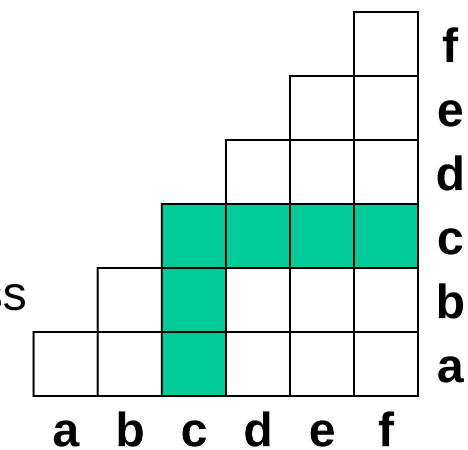


Example metadata:

	Artist	Genre	Recording
a	U2	Rock	1982
b	Madonna	Pop	1987
c	B. B. King	Jazz	1982
d	U2	Rock	2002
e	Madonna	Pop / Rock	2000
f	B. B. King	Jazz	1987

Similarity between Songs

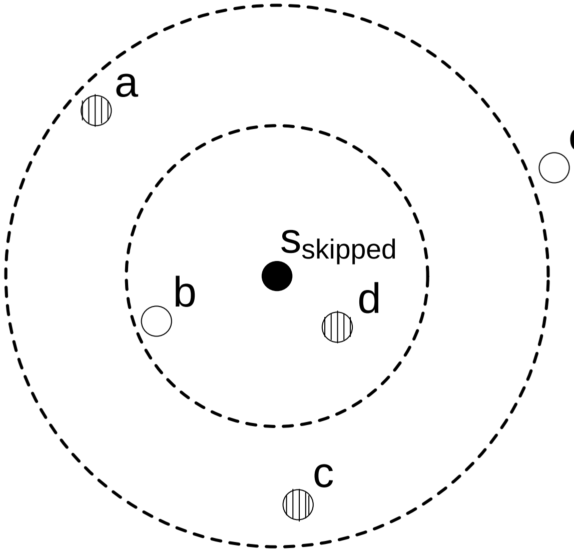
- Needed for several types of queries
  - Find a similar song
  - Find a random song avoiding the songs similar to disliked songs
- Requires a similarity measure
  - Similarity between songs is subjective
- Metric properties are most often not obeyed
  - Identity property ok
  - Most often symmetry
  - Almost **never** triangular inequality
    - Hard to index ☹
  - A distance matrix may be inappropriate and needless



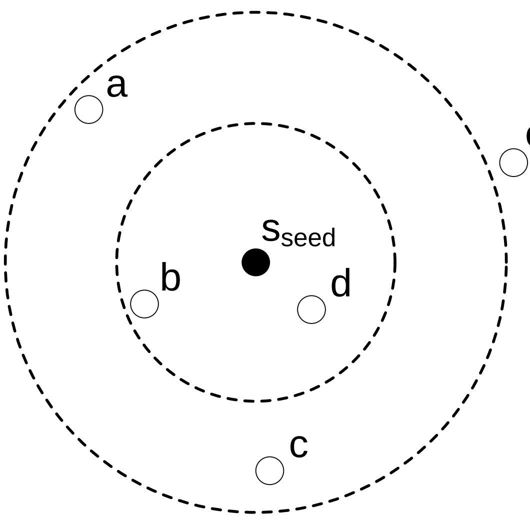
- Many-to-many relationship between **facts**

Query Operators

- *RandomSong*
  - Find a subset of randomly chosen songs from which the song *least similar* to *any* of the skipped songs is returned



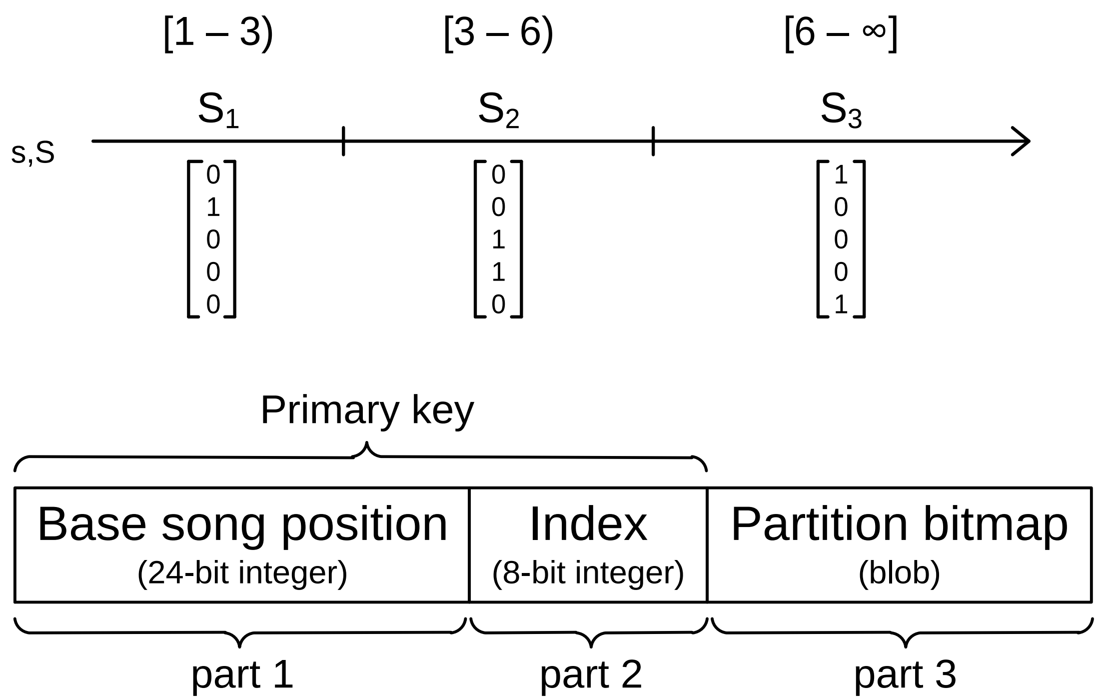
- *SimilarSong*
  - Retrieves the song *most similar* to a given *seed song*
  - Only valid (non-skipped) songs returned
  - The inner-most distance partition containing valid songs is considered



Distance Store Index

Complete disjoint partitioning of the distance domain

- Each song is assigned a unique *base position*
- Unique partition number (*index*)
- Each partition is represented with a (compressed) *bitmap*



Metadata Index

- Building bitmap index on hierarchical data (cube/dimensions)
  - The top hierarchical level:

Primary key: ID

ID (integer)	Dimension value (varchar)	Bitmap (blob)

  - The sub-levels:

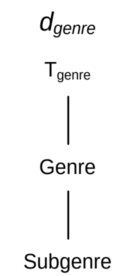
Primary key: ID; Secondary key (clustered): SuperID

ID (integer)	Dimension value (varchar)	SuperID (integer)	Bitmap (blob)

Metadata Index

- Bitmaps allows effective indexing of hierarchical data

  Example: (using the title and genre dimensions)
  - Snowflake schema
  - Bitmap for higher level (*Genre*) is OR of lower level (*Subgenre)*



Fact table

TitleID	TitleLevelDepth	SubgenreID	GenreLevelDepth
1	1	1	2
2	1	2	2
3	1	3	2
4	1	4	2

Title (depth 1)

TitleID	Title	Bitmap
1	”The Fine Art”	1000
2	”T.N.T.”	0100
3	”Wonder Wall”	0010
4	”Twentysomething”	0001

Subgenre (depth 2)

SubgenreID	Subgenre	GenreID	Bitmap
1	”Soft Pop”	1	1000
2	”Hard Rock”	2	0100
3	”Brit Pop”	1	0010
4	”Modern Jazz”	3	0001

Genre (depth 1)

GenreID	Genre	Bitmap
1	”Pop”	1010
2	”Rock”	0100
3	”Jazz”	0001

Query Examples

Music Player Application

Current song: s
History
Skipped songs
Collection

Nine Inch Nails from 1994 and songs by Corona

Query Examples

Music Player Application

Current song: s
History
Skipped songs
Collection

Nine Inch Nails from 1994 and songs by Corona

Fact table			
ArtistID	ArtistLevelDepth	ReleaseID	ReleaseLevelDepth
1	1	1	2
1	1	2	2
1	1	2	2
2	1	3	2
3	1	3	2
4	1	4	2

Artist (depth 1)		
ArtistID	Artist	Bitmap
1	"Nine Inch Nails"	111000
2	"Corona"	000100
3	"Oasis"	000010
4	"Jamie Cullum"	000001

Year(depth 2)			
YearID	Year	DecadeID	Bitmap
1	"1987"	1	100000
2	"1994"	2	011000
3	"1995"	2	000110
4	"2004"	3	000001

Decade (depth 1)		
DecadeID	Decade	Bitmap
1	"1980's"	100000
2	"1990's"	011110
3	"2000's"	000001

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{AND} \quad \textbf{OR} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Query Examples

Music Player Application

Current song: s
History
Skipped songs
Collection 011100

Nine Inch Nails from 1994 and songs by Corona

	$[1 - 3)$	$[3 - 6)$	$[6 - \infty]$
	S_1	S_2	S_3
$ds_{s,S}$	$\begin{bmatrix}0\\1\\1\\1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\\1\\0\end{bmatrix}$

daisy

Bitmap Index

- Example:

TID	X	Y	X0	X1	X2	X3	X4	Ya	Yb
1	**3**	a	0	0	0	1	0	1	0
2	**1**	a	0	1	0	0	0	1	0
3	**1**	b	0	1	0	0	0	0	1
4	**2**	b	0	0	1	0	0	0	1
5	**4**	b	0	0	0	0	1	0	1
6	**0**	a	1	0	0	0	0	1	0
7	**1**	a	0	1	0	0	0	1	0
8	**3**	a	0	0	0	1	0	1	0

- Range queries

  X0 OR X1

- Multidimensional queries

  (X0 OR X1) AND Ya

Why Bitmap Compression?

- One bitmap per attribute value
  → Index size explodes when dealing with high-cardinality attributes: n * (#X + #Y)

TID	X	Y	X0	X1	X2	X3	X4	Ya	Yb
1	**3**	a	0	0	0	1	0	1	0
2	**1**	a	0	1	0	0	0	1	0
3	**1**	b	0	1	0	0	0	0	1
4	**2**	b	0	0	1	0	0	0	1
5	**4**	b	0	0	0	0	1	0	1
6	**0**	a	1	0	0	0	0	1	0
7	**1**	a	0	1	0	0	0	1	0
8	**3**	a	0	0	0	1	0	1	0

- Tradeoff between I/O and CPU load

- "Ultimate goal":

  Fastest logical operations on bitmaps

  and the best compression ratio

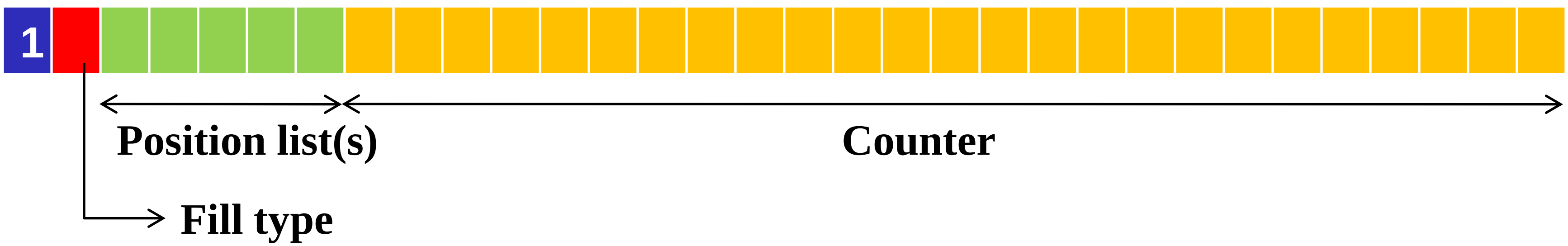
What’s the trick?

- Ability to perform logical operations on compressed or only partially decompressed bitmaps
- Limited conditional branches
- Byte-Aligned Bitmap Compression (BBC)
  [Antoshenkov VLDB ’96]
  - Byte level compression
  - High dependencies between bytes
  - Used in Oracle
- Word Aligned Hybrid WAH
  [Wu et al. CIKM’01,VLDB’04,TODS’06,…]
  - 6-12 times faster than BBC, respects CPU word alignment
  - Lower compression ratio (160% of BBC space)
  - “CPU Optimal”
  - Used in scientific computing, etc.

PLWAH Bitmap Compression

- Literal word

- Fill word



- Four intuitive steps (integrated in practice):
  1. Split bitmap into chunks of w-1 bits (word length w)
  2. Make fill words or literal words
  3. Merge fill words (adapt the counter)
  4. Merge fill words with literal words (if possible)

PLWAH Example (Step 1)

- Original uncompressed bitmap (w = 32)

  0000000000 0000000000 0000000000 00

  0000000000 0000000000 0000000000 00

  0000000000 0000000000 0000000000 00

  0000001000 00

- Form groups of w-1 bits

  0000000000 0000000000 0000000000 0

  0000000000 0000000000 0000000000 0

  0000000000 0000000000 0000000000 0

  0000000001 0000000000 0000000000 0

PLWAH Example (Step 2)

0000000000 0000000000 0000000000 0
0000000000 0000000000 0000000000 0
0000000000 0000000000 0000000000 0
0000000001 0000000000 0000000000 0

- Generate Fill and Literal words

1|0|00000|0000000000 0000000000 00001
1|0|00000|0000000000 0000000000 00001
1|0|00000|0000000000 0000000000 00001
0|0000000001 0000000000 0000000000 0

PLWAH Example (Step 3)

1|0|00000|0000000000 0000000000 00001
1|0|00000|0000000000 0000000000 00001
1|0|00000|0000000000 0000000000 00001
0|0000000001 0000000000 0000000000 0

- Merge Fill words

  1|0|00000|0000000000 0000000000 00011
  0|0000000001 0000000000 0000000000 0

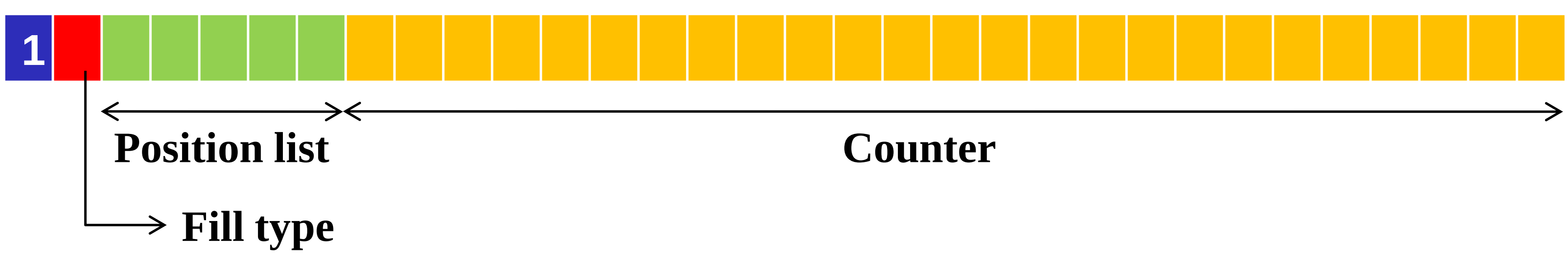
PLWAH Example (Step 4)

1|0|00000|0000000000 0000000000 00011

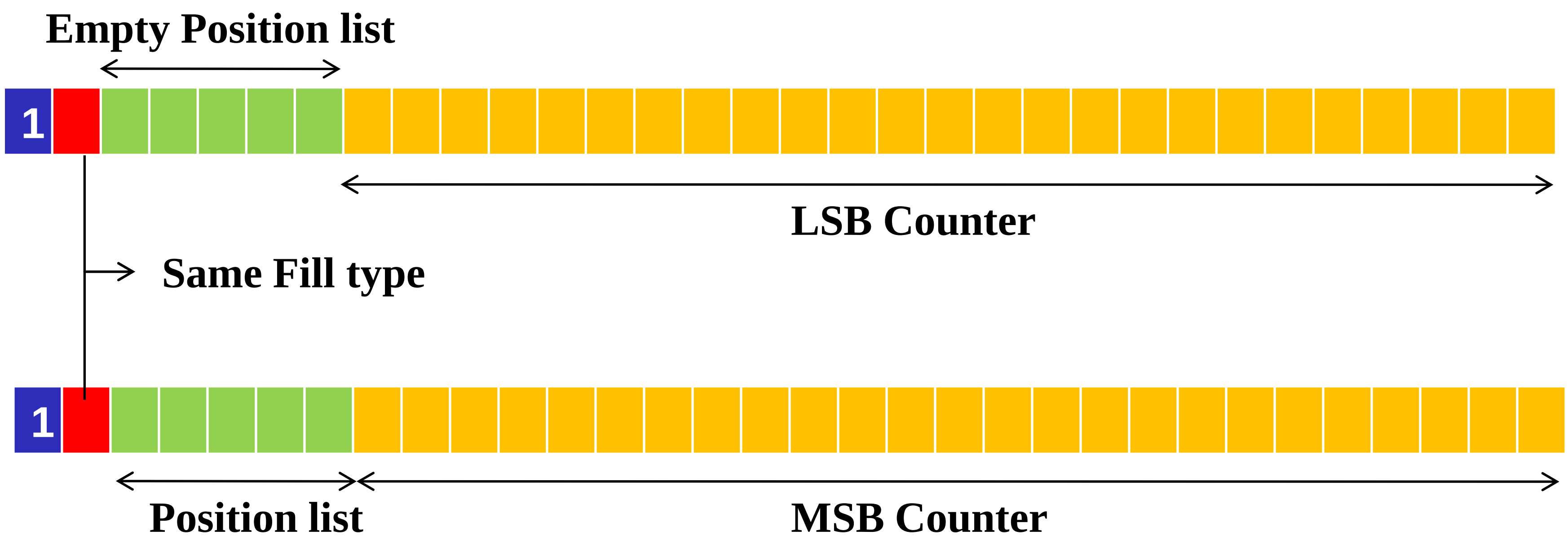
0|0000000001 0000000000 0000000000 0

- Merge Fill words with Literal words

  1|0|01010|00000000000 000000000 00011



Adaptive Counter



- A second Fill Word is used if the counter is too small
  - Two fill words of the same type
  - First fill word has an empty position list

Intuitive Storage Estimates

- High-cardinality attribute uniformly distributed
  → most bitmaps are sparse

0000…0000100000…0000

Fill word of 0s with a non-empty position list → one word

→ c bitmaps, each bitmap has n / c set bits

→ total size = n words

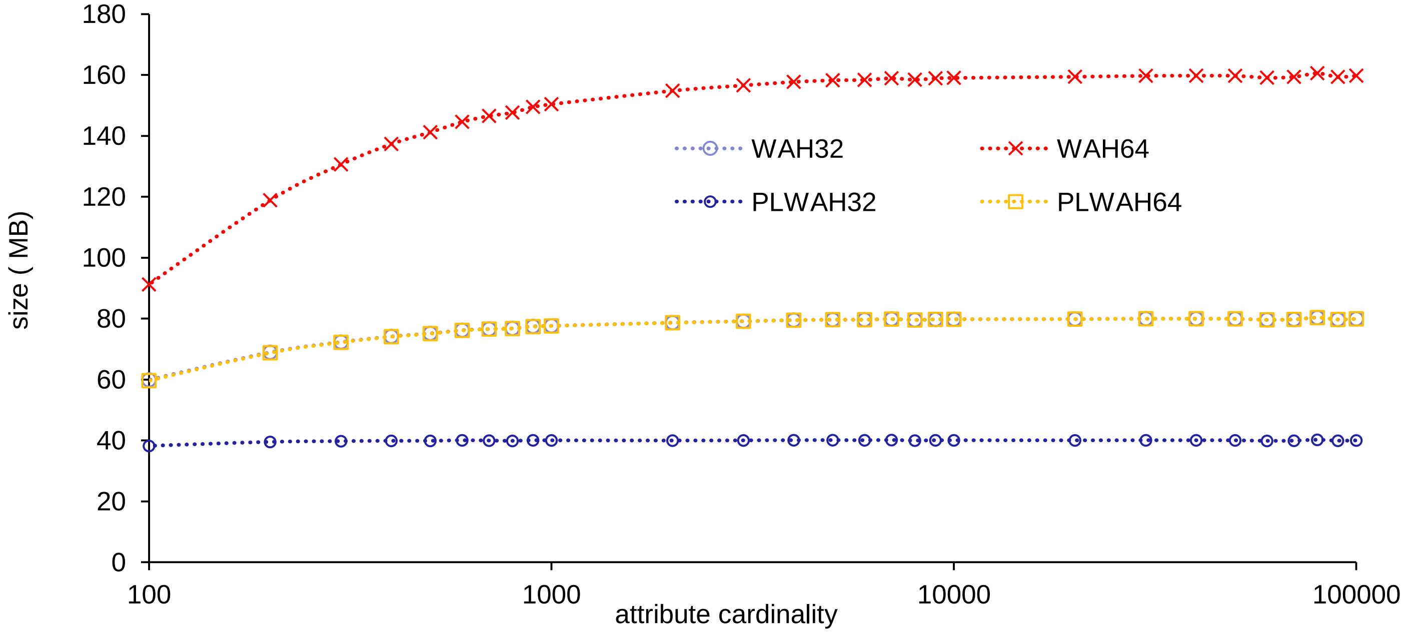
→ Independent from the cardinality (for c >> w)

→ PLWAH compressed bitmaps are half the size of the classical WAH compressed bitmaps
(within the compression limits)

Storage Experiments

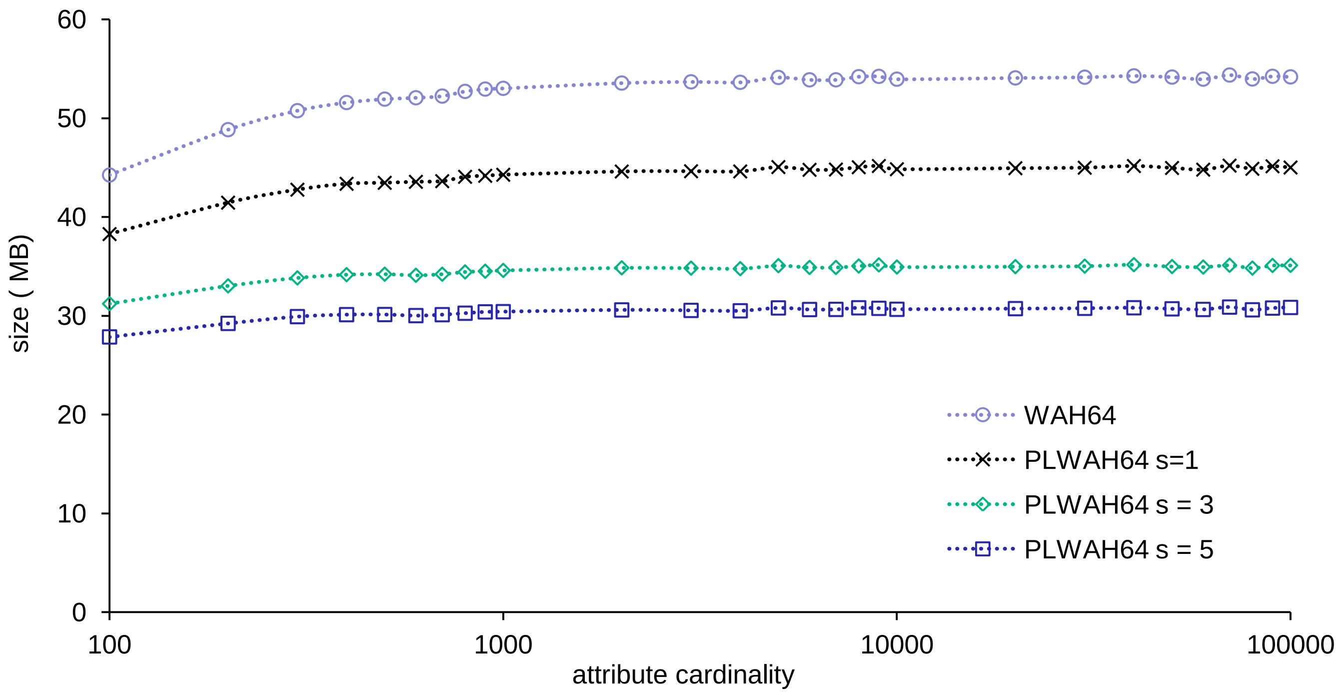
- Uniform distribution

  10 000 000 elements



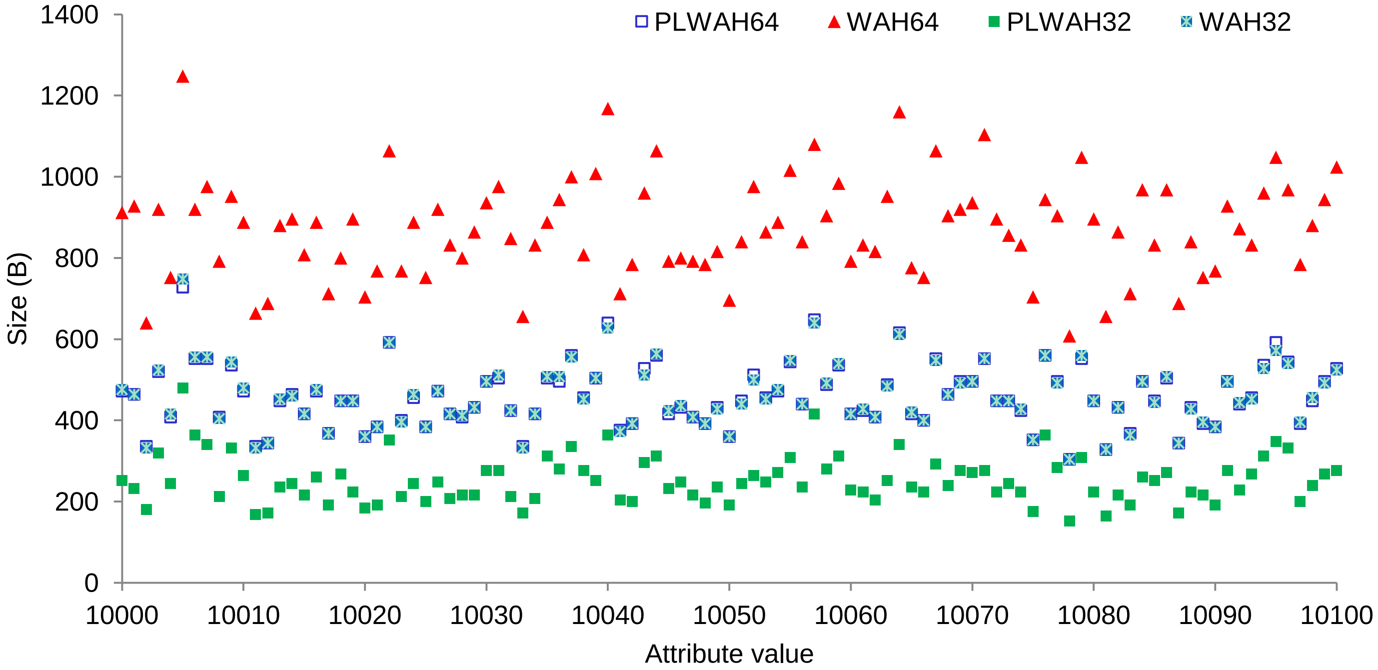
Storage Experiments

- Clustered bitmap
  - 2-state Markov process
  - Number of position lists: s



Storage Experiments

- Real dataset (music metadata)
  - Elements: 15 000 000 music segments
  - 15 attributes, cardinality: 100 000



Read / Write Performance

- Fast Reads
  - Only one branch per word decompression (little CPU stalling)
  - Most operations only require bitmasks (efficient CPU pipelining)
  - Reading the position list done by bit shifting

- Operations iterates over the compressed words
  → half the number of iterations needed compared w. WAH

- Writes take advantage of new CPU instructions (i7)
  - Bitscan: the relative position of a bit in a word
  - PopCount: the number of set bits in a word

- But memory allocation becomes too slow…

Performing Logical Operators

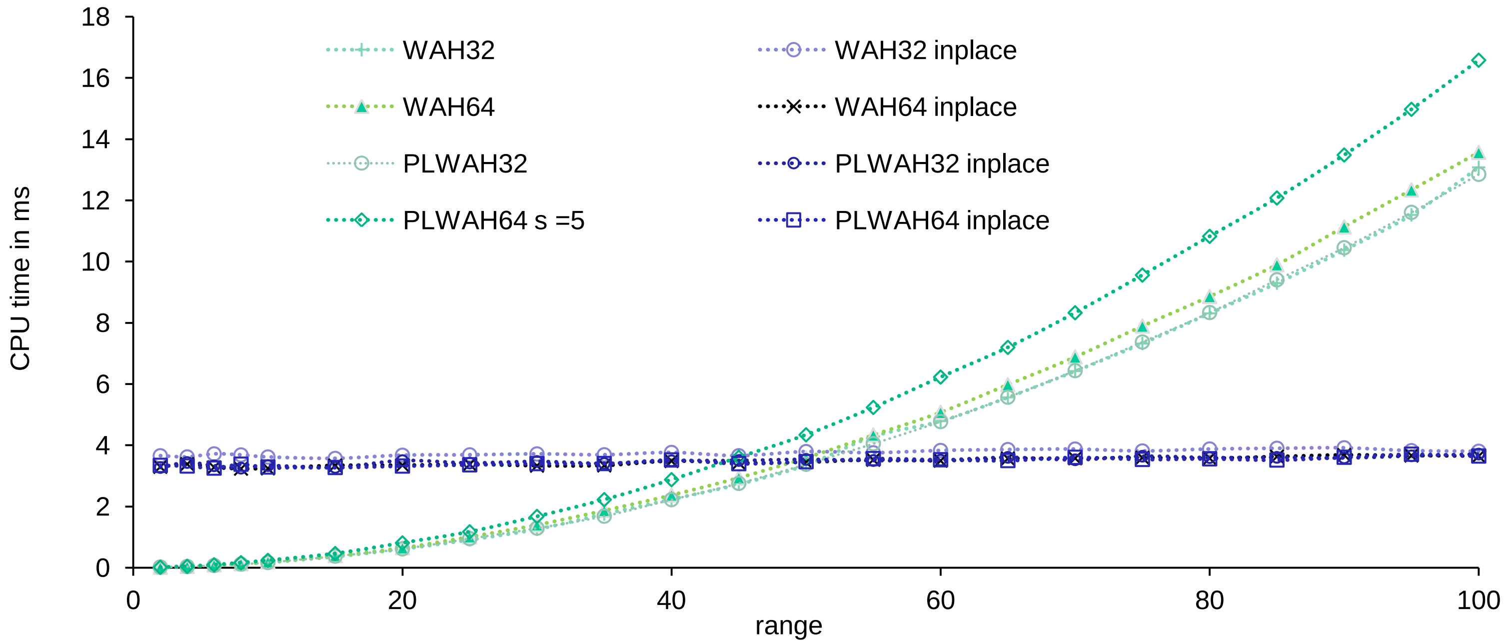
- Operator over compressed bitmaps z = x | y;
  - Memory allocation for each operation
  - Linear complexity in the size of the output
  - Straightforward parallelization

- In-place operator x |= y;
  - Decompressed bitmap
  - High initial memory allocation
  - Linear complexity in the size of compressed input

Experiments

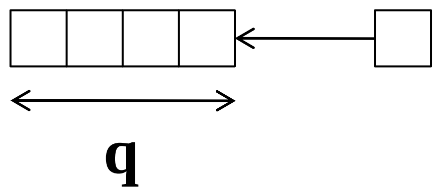
- OR operator

  (Bitmaps become denser w. no. ORs)

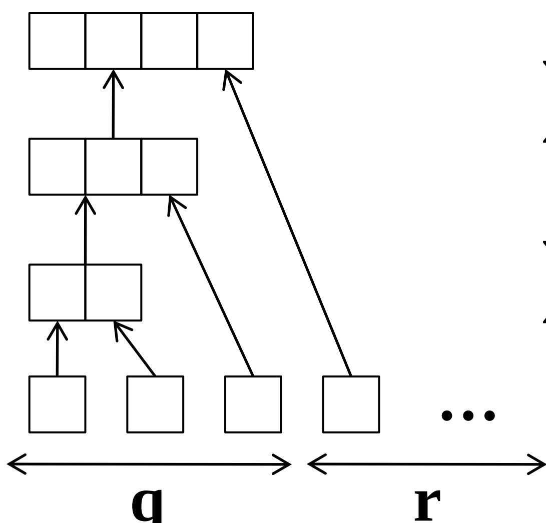


The “quadratic” behavior

On sparse bitmaps, the cost of an OR is proportional to the sum of the size of the two operands



$$q+1$$

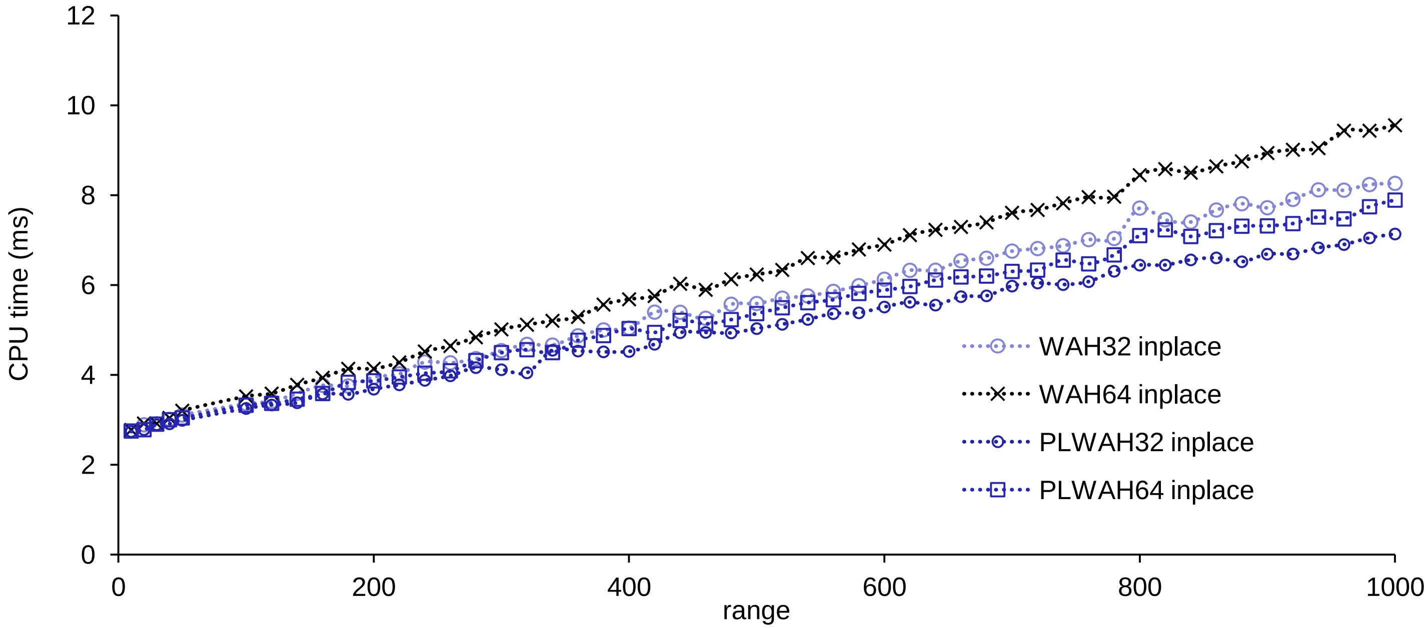


$$\sum_{i=1}^{r} i = \frac{r(r-1)}{2}$$

$$\sum_{i=q}^{r+q} i = \frac{(r+q)(r+q-1)}{2} - \frac{(q-1)(q-2)}{2}$$

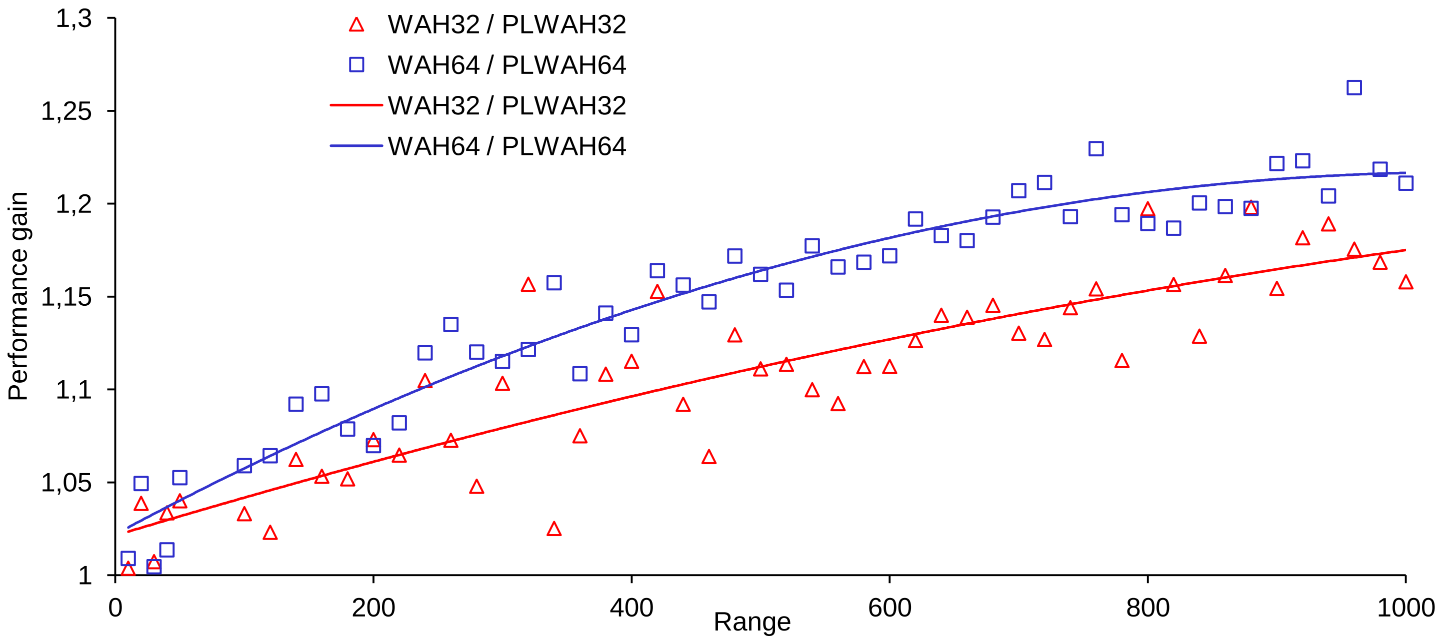
Experiments

- Inplace OR alleviates this
  - High initial memory allocation cost
  - Scales linearly



Experiments

- In-place OR comparison



PLWAH Conclusion

- Up to 50% smaller than WAH
- Up to 40% faster than WAH (CPU time)
- Supports extremely large and sparse bitmaps (Adaptive Counter)

- Optimizations for working in a non-sequential fashion (parallelization to n-cores, priority queues)
- Updates

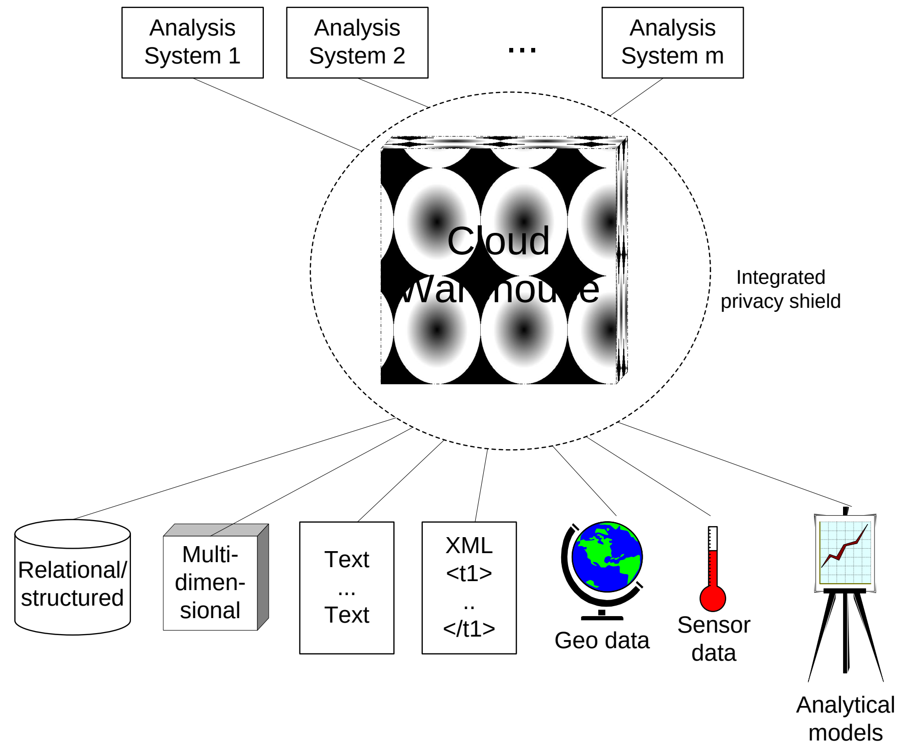
- Indexing irregular dimension hierarchies
- Integration with different systems

Conclusion

- Multidimensional data is everywhere
- In many application domains
- Often too complex to be managed using standard models and tools
- From high-level models to bits…
- Multidimensional data challenges (and solutions) presented for a number of domains
  - Medical data
  - Spatial data
  - XML and Semantic Web
  - Text
  - Music
- What's next ?
  - Cloud Intelligence in the Cloud Warehouse

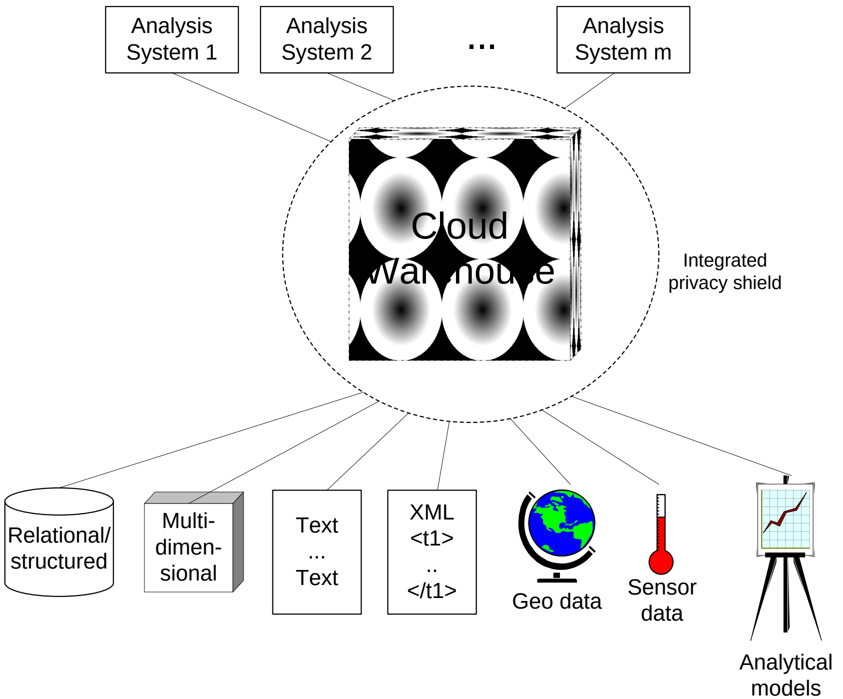
The Cloud Warehouse

- Overall idea: repeat the “data warehouse success” for integrating different types of data
- Data of a particular type should only need to be “integrated” once
- Integrated results put into common, “harmonized” data store (CW)
- CW handles all these types of data (or *derivations*) for data analysis
- The CW is a cube, meaning, i.e., based on MD principles
- CW content has different “shades,” data is “not just black and white.”
- All CW data has a built-in notion of “perfection” (precision/certainty)
- Data may be very precise and totally certain (like ord. DW data)
- Or imprecise and uncertain (sampling errors, data from analytical models)
- The CW is *virtual* (dotted lines) – data is spread throughout the cloud



The Cloud Warehouse

- Sources (different types) connected through only ***one*** "connection"
- Difficult task of integrating particular type of data handled once-and-for-all, by mapping into CW data model (+algs/tools)
- Analysis systems have only one "connection" each to the CW
- Take advantage of all functionality and data available in CW
- No need to perform integration themselves (as the systems mentioned earlier)
- CW has "integrated privacy shield."
- When data comes from sources, shield analyzes data + performs modifications (aggregation, swapping,...) before storing, or data may be encrypted
- When data is requested from an analysis system, CW may perform further modifications of results



The Cloud Warehouse

- The CW approach means that the “complexity” of the integration of all the different types of data for:
  - *n* types of data
  - *m* analysis systems
- Drops to *n+m* (from *n*m*)!
- The “hard” tasks
  - Integrating a new type of data
  - Privacy, scalability, reliability, …
- Are generally handled only once
  - By the CW rather than in the analysis systems
  - Great relief for the development of the analysis systems.
  - The same benefits to all the described data types as is currently available in traditional DWs for structured data
  - CW enables the integration and analysis of all types of data using the developed data model and query language

A New Data Model

- Basis for the CW will be a novel kind of data model
  - Should encompass the best of several worlds
  - Multidimensional modeling concepts (superior for analysis)
  - Flexibility and generality from semi-structured data models
  - Borrow useful Semantic Web concepts
- Support a much wider range of data
  - Physical world data: geo data, sensor data, data streams, ...
    - Missing or incorrect values, etc.
  - Semi-structured and unstructured data
    - Enabling analysis across structured, semi-structured, and unstructured data
  - Imperfect (imprecise, uncertain, etc.) data
- Support for privacy+security, virtualization, scalability, reliability,…

Research Plan

- Develop **complete** “infrastructure”
- Query languages, query processing/optimization,…
- Data integration techniques
  - Allow for virtualization, scalability, …
- Techniques for integrating databases, sensors, and analytical/predictive models of data
- Integrate contributions into a common prototype system
  - Open source project?
- Integrated system enables solutions to be evaluated experimentally using large volumes of real-world data

References

- General complex data
  - T. B. Pedersen and C. S. Jensen. Multidimensional data modeling for complex data. In ICDE, pages 336–345, 1999.
  - T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending practical pre-aggregation in on-line analytical processing. In VLDB, pages 663–674, 1999.
  - T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. Information Systems, 26(5):383–423, 2001.
  - C. E. Dyreson, T. B. Pedersen, and C. S. Jensen. Incomplete information in multidimensional databases. In M. Rafanelli, editor, Multidimensional databases: Problems and Solutions. Idea Group Publishing, 2003.
- Medical data
  - T. B. Pedersen and C. S. Jensen. Clinical data warehousing—a survey. In MEDICON, page 20.3, 1998.
  - T. B. Pedersen and C. S. Jensen. Research issues in clinical data warehousing. In SSDBM, pages 43–52, 1999.

References

- OLAP-XML and semantic web warehousing

  - D. Pedersen, J. Pedersen, and T. B. Pedersen. Integrating XML Data in the TARGIT OLAP System. In ICDE, pp. 778–781, 2004.
  - D. Pedersen, T. B. Pedersen, and K. Riis. The Decoration Operator: A Foundation for On-Line Dimensional Data Integration. In IDEAS, pp. 357–366, 2004.
  - D. Pedersen, K. Riis, and T. B. Pedersen. XML-Extended OLAP Querying. In SSDBM, pp. 195–206, 2002.
  - Xiufeng Liu, Christian Thomsen, Torben Bach Pedersen: 3XL: Supporting efficient operations on very large OWL Lite triple-stores. Inf. Syst. 36(4): 765-781 (2011)

- Text data

  - J. M. Perez, R. Berlanga Llavori, M. J. Aramburu Cabo, and T. B. Pedersen. R-Cubes: OLAP Cubes Contextualized with Documents. In ICDE, pp. 1477–1478, 2007.
  - J. M. Perez, R. Berlanga Llavori, M. J. Aramburu, and T. B. Pedersen. Integrating Data Warehouses with Web Data: A Survey. IEEE TKDE 20(7):940–955, 2008.
  - J. M. Prez-Martnez, R. Berlanga Llavori, M. J. Aramburu, and T. B. Pedersen. Contextualizing data warehouses with documents. Decision Support Systems 45(1):77–94, 2008.

References

- Music data and bitmaps

  - F. Deliege, B. Y. Chua, and T. B. Pedersen. High-Level Audio Features: Distributed Extraction and Similarity Search. In ISMIR, pp. 565–570, 2008.
  - F. Deliege and T. B. Pedersen. Fuzzy Song Sets for Music Warehouses. In ISMIR, pp. 21–26, 2007.
  - F. Deliege and T. B. Pedersen. Using Fuzzy Lists for Playlist Management. In MMM, pp. 198–209, 2008.
  - F. Deliege and T. B. Pedersen. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In EDBT, pp. 228–239, 2010.
  - C. A. Jensen, E. M. Mungure, T. B. Pedersen, and K. Sørensen. A Data and Query Model for Dynamic Playlist Generation. In ICDE Workshops, pp. 65–74, 2007.
  - C. A. Jensen, E. M. Mungure, T. B. Pedersen, and K. Sørensen, F. Deliege. Effective Bitmap Indexing for Non-metric Similarities. In DEXA, Vol. (1), pp. 137–151, 2010.

- Spatial data

  - C. S. Jensen, A. Kligys, T. B. Pedersen, and I. Timko. Multidimensional data modeling for location-based services. VLDB Journal 13(1):1–21, 2004.
  - T. B. Pedersen, N. Tryfona: Pre-aggregation in Spatial Data Warehouses. SSTD 2001: 460-480

References

- Energy data
  - M. Böhm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys, T. Tusar: Data management in the MIRABEL smart grid system. EDBT/ICDT Workshops 2012: 95-102
  - L. Siksnys, M. E. Khalefa, T. B. Pedersen: Aggregating and Disaggregating Flexibility Objects. SSDBM 2012: 379-396
  - L. Siksnys, C. Thomsen, T. B. Pedersen: MIRABEL DW: Managing Complex Energy Data in a Smart Grid. DaWaK 2012, to appear.
- Outlook
  - T. B. Pedersen. Warehousing The World: A Vision for Data Warehouse Research. Annals Of Information Systems, Special Issue: New Trends in Data Warehousing and Data Analysis, pp. 1–17, 2009.
  - T. B. Pedersen. Research challenges for cloud intelligence: invited talk. In EDBT/ICDT Workshops, 2010.

Acknowledgements

- Some slides borrowed from Dennis Pedersen, Juan Manuel Perez Martinez, Igor Timko, Xiufeng Liu, Francois Deliege, Kenneth Sørensen, Ester Mungure, Claus Aage Jensen, MIRABEL consortium