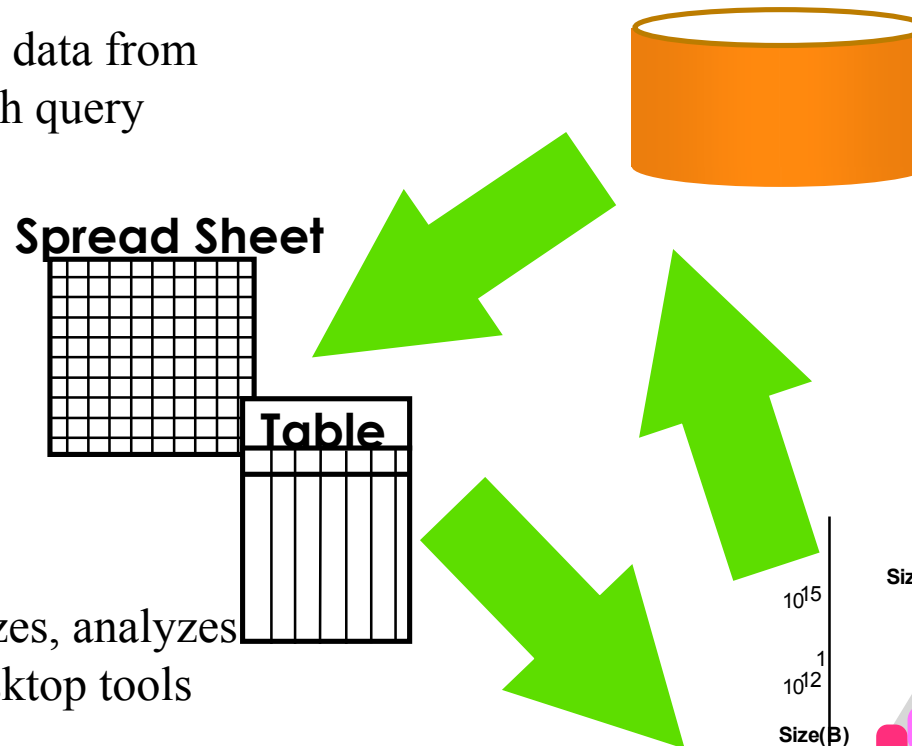

OLAP & Data Cubing

Spring 2007

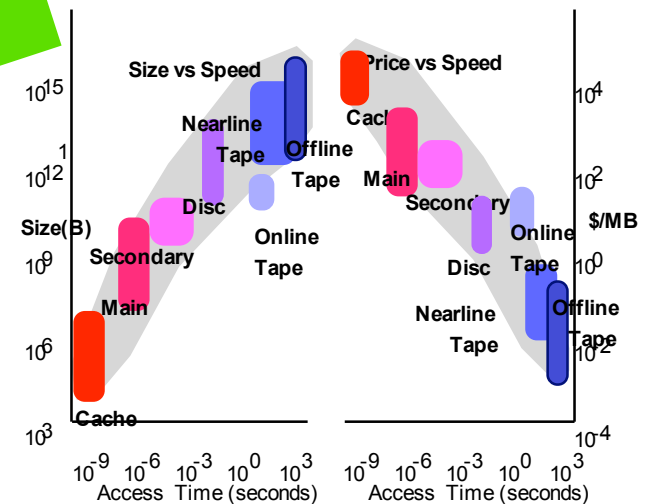
Nick Roussopoulos
nick@cs.umd.edu

OLAP-The Data Analysis Cycle

- User extracts data from database with query



- Then visualizes, analyzes data with desktop tools



The Data Cube

[Gray, Bosworth, Layman, Pirahesh ICDE 96]

- summarize multidimensional data for trend analysis

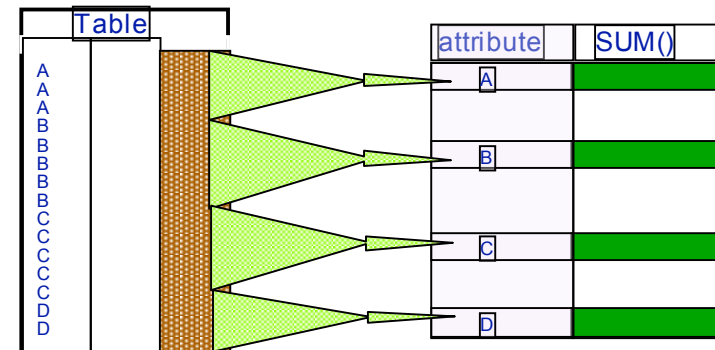
Time (UCT)	Latitude	Longitude	Altitude (m)	Temp (c)	Pres (mb)
27/11/94:1500	37:58:33N	122:45:28W	102	21	1009
27/11/94:1500	34:16:18N	27:05:55W	10	23	1024

- groupby with statistical functions (avg,min,max,count,sum) aggregates over table sub-groups

```
select avg(temp) from weather
select time,altitude from
weather
groupby time,altitude
```

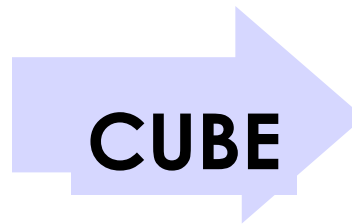
- results in a new table

```
select      location, sum(units)
from        inventory
group by   location
having     nation = "USA";
```



An Example

SALES			
Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

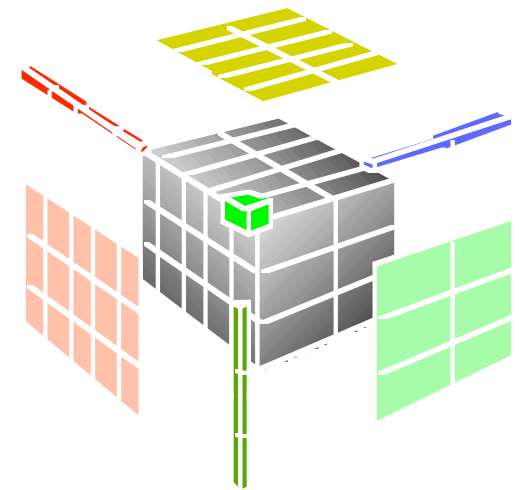


DATA CUBE			
Model	Year	Color	Sales
ALL	ALL	ALL	942
chevy	ALL	ALL	510
ford	ALL	ALL	432
ALL	1990	ALL	343
ALL	1991	ALL	314
ALL	1992	ALL	285
ALL	ALL	red	165
ALL	ALL	white	273
ALL	ALL	blue	339
chevy	1990	ALL	154
chevy	1991	ALL	199
chevy	1992	ALL	157
ford	1990	ALL	189
ford	1991	ALL	116
ford	1992	ALL	128
chevy	ALL	red	91
chevy	ALL	white	236
chevy	ALL	blue	183
ford	ALL	red	144
ford	ALL	white	133
ford	ALL	blue	156
ALL	1990	red	69
ALL	1990	white	149
ALL	1990	blue	125
ALL	1991	red	107
ALL	1991	white	104
ALL	1991	blue	104
ALL	1992	red	59
ALL	1992	white	116
ALL	1992	blue	110

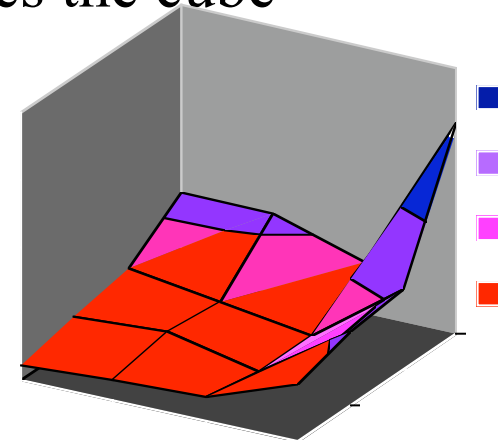
Division of labor

Computation vs Visualization

- Relational system builds CUBE relation
 - ◆ aggregation best done close to data
 - ◆ filtering of data is possible
 - ◆ Cube computation may be recursive
 - ➔ (e.g., percent of total, quartile,)



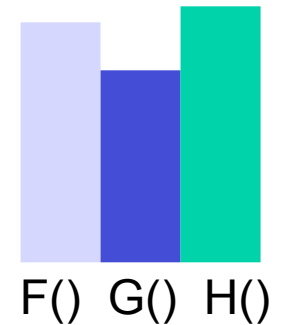
- Visualization System displays/explores the cube



Problems with SQL Groupbys

- Histograms (aggregation over computed categories)

```
SELECT day, nation, MAX(Temp)
FROM Weather
GROUP BY CUBE Day(Time) AS day,
Country(Latitude, Longitude)
AS nation;
```



Weather					
Time (UCT)	Latitude	Longitude	Altitude (m)	Temp. (c)	Pres. (mb)
96/6/1:1500	37:58:33N	122:45:28W	102	21	1009
Many more rows like the ones above and below					
96/6/7:1500	34:16:18N	27:05:55W	10	23	1024

Problems with SQL Groupbys

- drill-down and roll-up

Table 3: Sales Roll Up by Model by Year by Color

Model	Year	Color	Sales by Model by Year by Color	Sales by Model by Year	Sales by Model
Chevy	1994	black	50		
		white	40		
				90	
	1995	black	85		
		white	115		
				200	
					290

Not relational
(null values in the keys)

Table 4: Sales Summary

Model	Year	Color	Units
Chevy	1994	black	50
Chevy	1994	white	40
Chevy	1994	ALL	90
Chevy	1995	black	85
Chevy	1995	white	115
Chevy	1995	ALL	200
Chevy	ALL	ALL	290

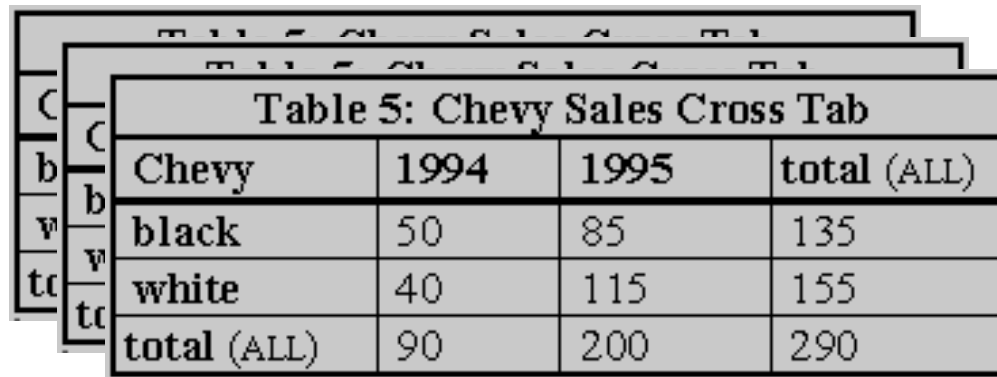
```

SELECT Model, ALL, ALL, SUM(Sales)
  FROM Sales
 WHERE Model = 'Chevy'
 GROUP BY Model
UNION
SELECT Model, Year, ALL, SUM(Sales)
  FROM Sales
 WHERE Model = 'Chevy'
 GROUP BY Model, Year
UNION
SELECT Model, Year, Color, SUM(Sales)
  FROM Sales
 WHERE Model = 'Chevy'
 GROUP BY Model, Year, Color;

```

More problems with Groubys

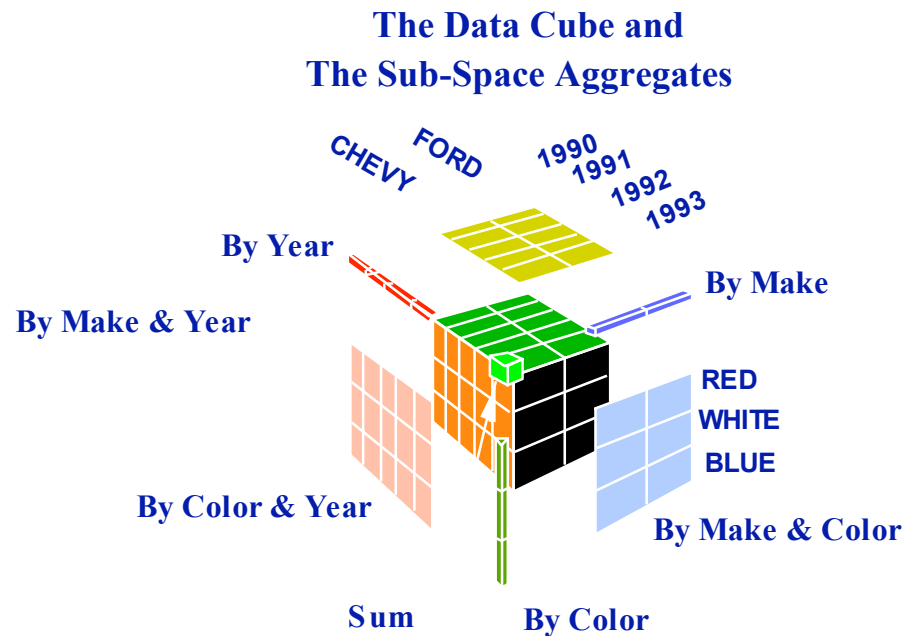
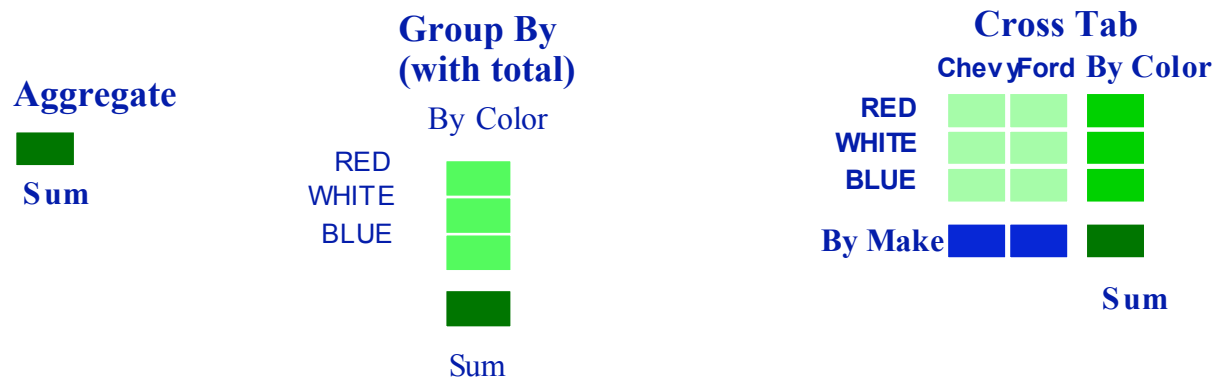
- roll-up is asymmetric (e.g. does not aggregate by year or by color alone)
- cross-tabulation (spreadsheets)



Chevy	1994	1995	total (ALL)
black	50	85	135
white	40	115	155
total (ALL)	90	200	290

- even if SQL syntax can be devised, a 6D cross-tab requires 64 groupby queries to generate it and 64 scans and sorts of the data
- ◆ most of these are not relational expressions but are in many report writers

CUBE: A Relational Aggregate Operator Generalizing Group By



Idea: N-dimensional Cube Each Attribute is a Dimension

- N-dimensionalAggregate (sum(), max(),...)
 - ◆ fits relational model exactly:
 - ➔ $a_1, a_2, \dots, a_N, f(*)$
- Super-aggregate over $N-1$ Dimensional sub-cubes
 - ➔ ALL, $a_2, \dots, a_N, f(*)$
 - ➔ $a_3, \text{ALL}, a_3, \dots, a_N, f(*)$
 - ➔ ...
 - ➔ $a_1, a_2, \dots, \text{ALL}, f(*)$
 - ◆ this is the $N-1$ Dimensional cross-tab.
- Super-aggregate over $N-2$ Dimensional sub-cubes
 - ➔ ALL, ALL, $a_3, \dots, a_N, f(*)$
 - ➔ ...
 - ➔ $a_1, a_2, \dots, \text{ALL}, \text{ALL}, f(*)$

Summary of the Cube

- CUBE operator generalizes relational aggregates
- Needs ALL value to denote sub-cubes
 - ◆ ALL values represent aggregation sets
- Needs generalization of user-defined aggregates
- Decorations and abstractions are interesting
- Computation has interesting optimizations
- Relationship to “rest of SQL” not fully worked out.

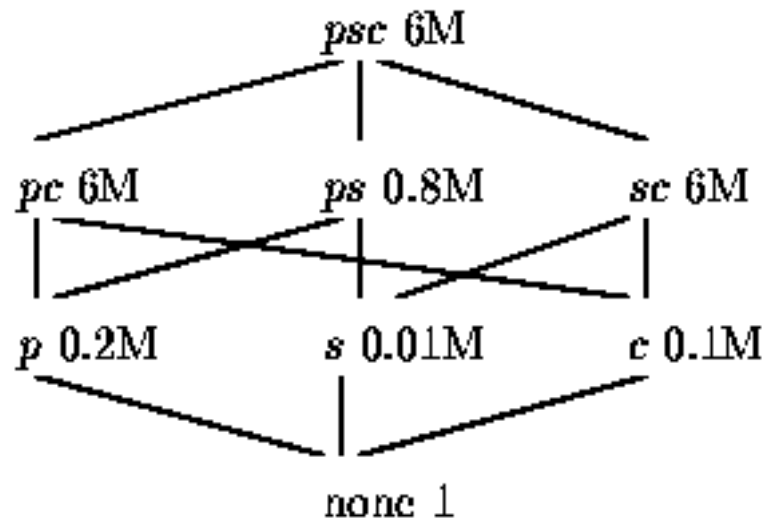
Cube={Materialized Views}

[Harinarayan, Rajaraman, Ullman 96]

- each groupby creates a “summary table” which is a materialized view with some dressing
- storing these summary tables speed up cube queries
- what to store and what not
- TPC-D example for sale analysis

1. `part, supplier, customer` (6M, i.e., 6 million rows)
2. `part, customer` (6M)
3. `part, supplier` (0.8M)
4. `supplier, customer` (6M)
5. `part` (0.2M)
6. `supplier` (0.01M)
7. `customer` (0.1M)
8. `none` (1)

The Lattice Organization



- the query sales groupby part will be answered at
 - p - cost of scanning 0.2M records
 - pc - -"- 6.0M -"-
 - psc - -"- 6.0M -"-
- select the views that minimize overall query performance
 - need a good query model
 - need a good optimization criterion

Views grow exponentially

- in general $2^{**}N$ subspaces

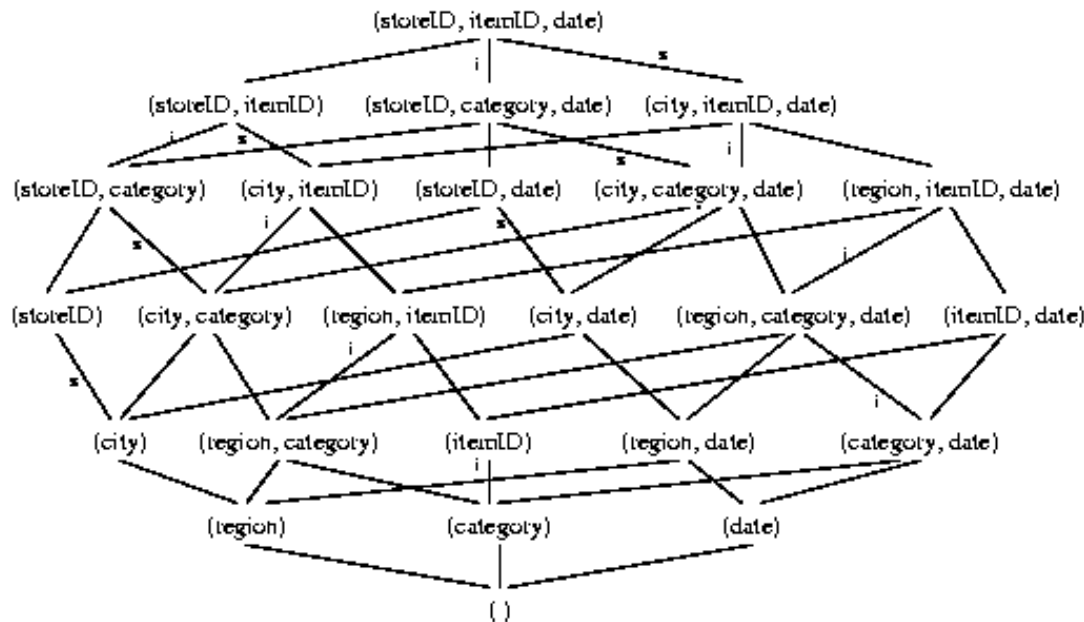
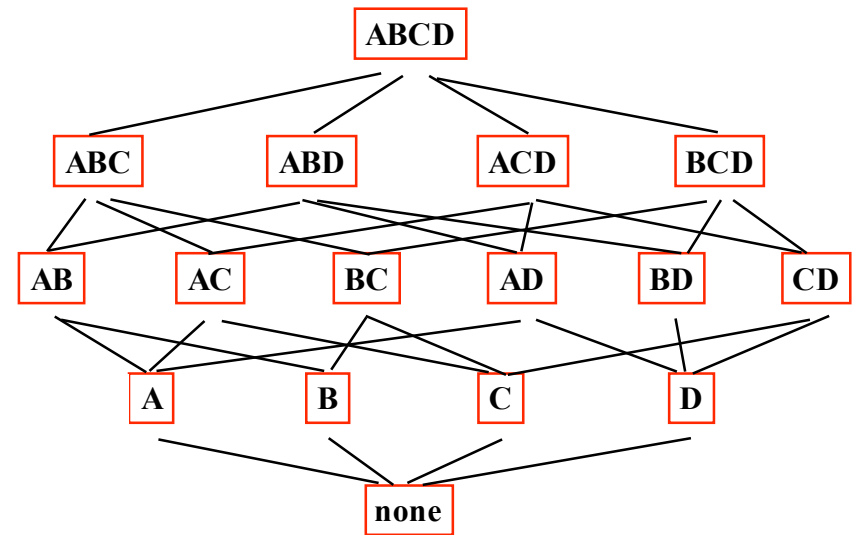


Figure 5: Combined lattice.

Greedy Allocation Algorithm

- optimization criterion:
 - storage S (total capacity)
 - query model (query frequencies to all views)
 - find the best views to materialize
- linear cost model:
 - cost of answering Q from a materialized view A generated by QA (Ancestor) is the size of the table A
 - cost of accessing part of a view is equal to cost of accessing all the view
- for each view v in a subset of views S
 - $C(v)$ is the storage cost
 - $B(v,S)$ is the benefit of v wrt S
 - for each $w \leq v$:
 - u is the min cost in S st $w \leq u$
 - if $C(v) < C(u)$ then $B_w = C(v) - C(u)$ else $B_w = 0$

computes the benefit of v by considering how much it helps other views that it covers- if the cost of answering thru v is better than their's competitors, then it adds this to the total benefit of v



Index Selection for OLAP

[Gupta, Harinarayan, Rajaraman, Ullman 96]

- 2^n sub-cubes or materialized views

$\binom{n}{r}$ r -dimensional subcubes.

- Slice queries (all queries are such)

```
select c, sum(sales)
from trans-table
where p = widget
groupby c
```

- possible queries: $\sum_{r=0}^n \binom{n}{r} 2^r = 3^n$

- Indexes $I_s(ps)$, $I_{sp}(ps)$, ...
 - possible indexes: $\sim 2n!$ (!)

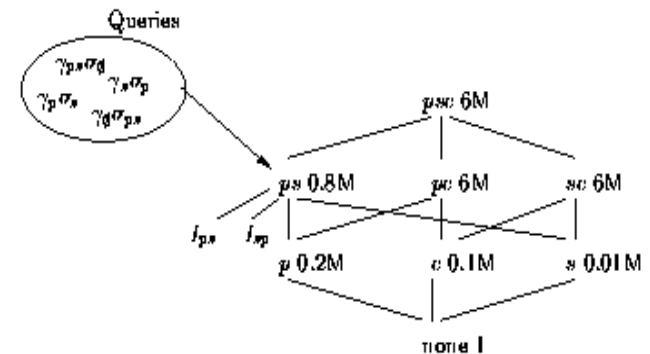


Figure 1: Subcubes, Queries and Indexes in the TPC-D database.

Estimating Sizes & Greedy Algorithms

- views
- indexes
- 1-,2-,...,r-Greedy

LIMITATIONS:

- estimates are very rough
- storage constraint is not the right criterion for deciding what to materialize and what to index- instead, refresh time is the right one

DynaMat

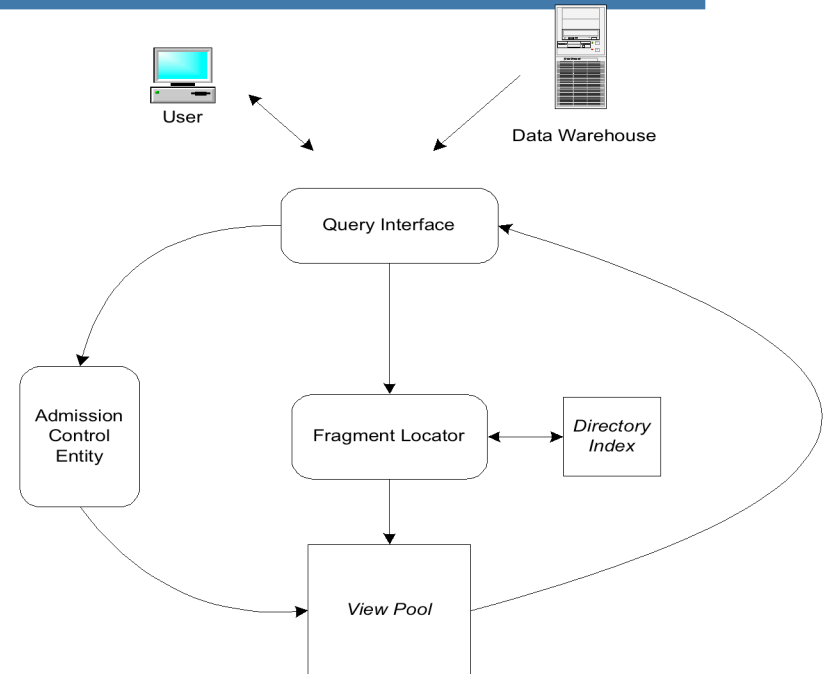
Yannis Kotidis, Nick Roussopoulos (Sigmod 1999)

- Conventional Data Warehouse
 - pre-computed set view is static (too hard to select and adjust)
 - usually selected by an administrator
- DynaMat proposed a framework for automatic management of views
 - Unifies view selection & view refresh
 - Amortizes generation and maintenance cost over multiple uses of cached results
- Techniques
 - DynMat caches the results of every query
 - Each incoming query is evaluated against the cached results to see if any of those can be used
 - The captured set is updated within an update cycle to the extent possible

DynaMat Architecture

Online Operation

- Try to match each query from the view pool (Fragment Locator)
 - Fragments are either single value predicates or complete ranges
 - A Directory Index is maintained for efficient searches
- On the fly decide whether to cache the result in the pool (Admission Control Entity)



Materialized Range Fragments

- Materialized Results are restricted to one of
 - a) a full Range $R_i = \{min_d, max_d\}$
 - b) a single value for d_i
 - c) an empty range denotes a dimension that is not present in the query
- SQL queries are mapped to MR queries that are answered by cached MRFs
- MRFs are Coarser than query results (expanded when necessary)
- No combination of MRFs are used to answer a query (more costly especially when MRFs are too small and/or overlap)
- An R-tree based index is used to identify possible MRFs that can answer the query- among those, the best fit is chosen
- The use of MRFs makes matching efficient.

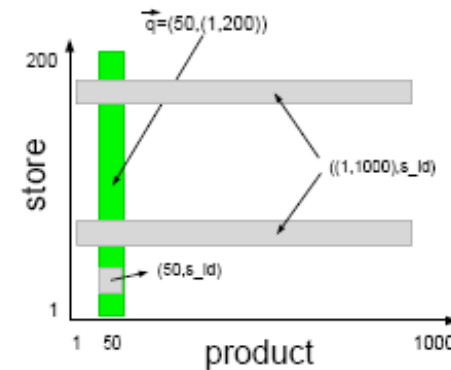


Figure 5: Querying stored MRFs

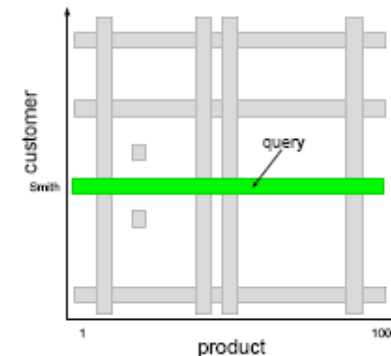
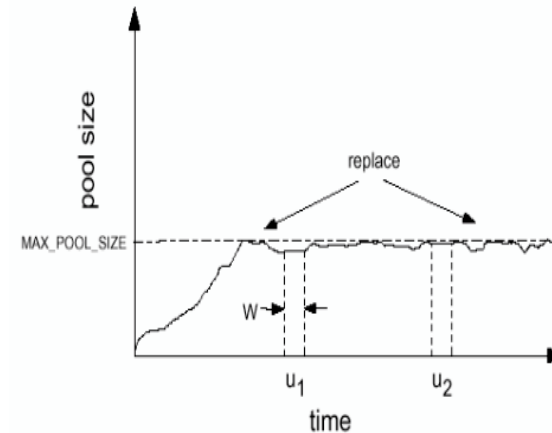


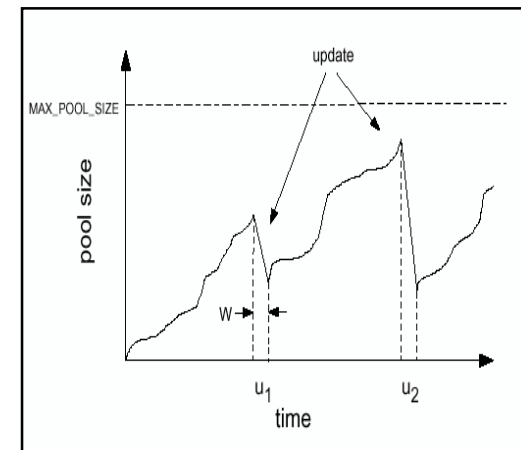
Figure 6: Directory for view (*product*, *customer*)

Space vs. Time Bound

- Space bound (online)
 - pool size is the constraining Factor
 - Several replacement strategies based on several “goodness” criteria
 - LRU
 - LFU,
 - Smallest Fragment First (SFF)
 - Smaller Penalty First (SPF)



- Time bound
 - offline window time is the constraining factor
 - Select the subset of the views to refresh



Storage Structures & Construction of Cubes

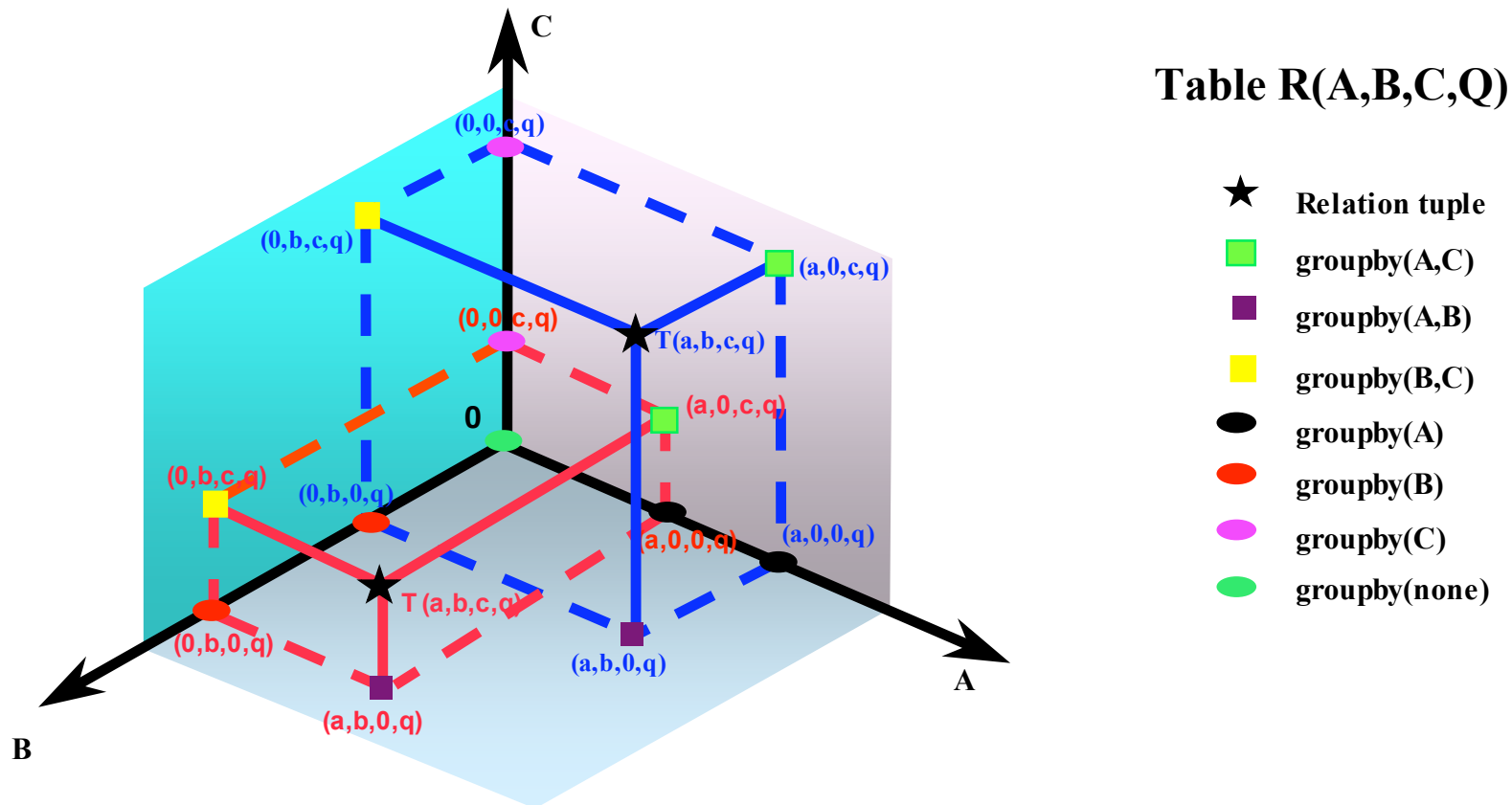
- Subcubes vs. Full cubes
 - Subcube selection
- Cost of construction and indexing
- Maintenance

Cubetrees

[Roussopoulos, Kotidis, Roussopoulos 97]

- better storage organization needed
 - materialized views and indexes are not different
 - single storage organization for both
- bulk load techniques are very important
 - rates should be in the order of GB/hour (industrial strength)
- incremental bulk updates is the MOST important issue
- we had lots of experience with spatial access methods: mainly with all possible variations of R-trees (handy)
- packed R-trees

Extended Data Cube Model



- ◆ relation tuples: points in the N-d space
- ◆ groupby projections: also points
- ◆ point data is very efficient for multidimensional indexing

Dataless Cubetree

- separate the fact table (relation) points
- keep only the aggregate projection points in the cubetree to reduce the size

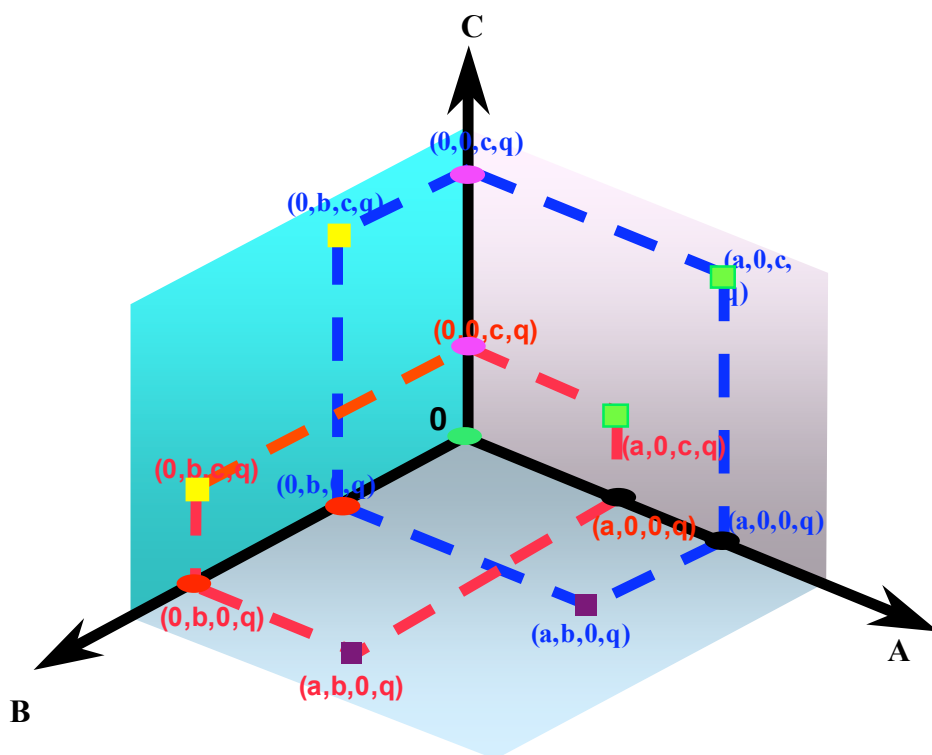
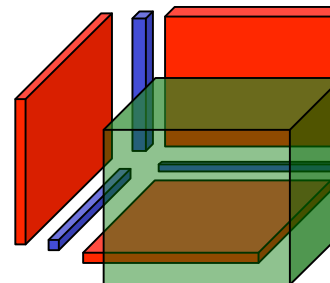
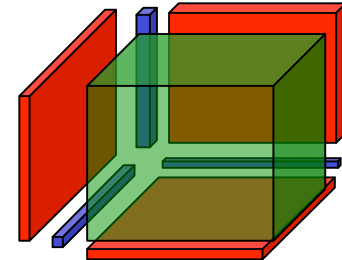


Table R(A,B,C,Q)

- groupby(A,C)
- groupby(A,B)
- groupby(B,C)
- groupby(A)
- groupby(B)
- groupby(C)
- groupby(none)

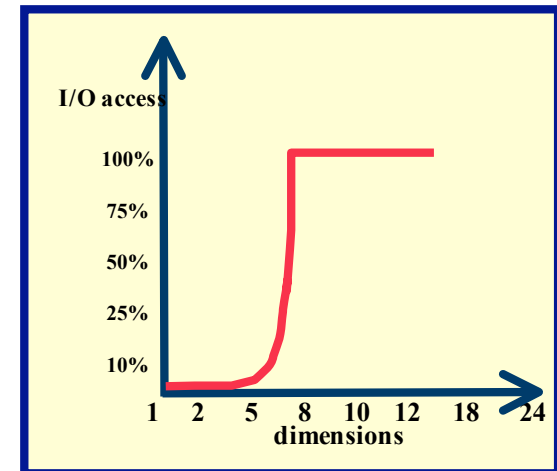
Cube Query Performance

- “*Slice and dice*” queries access data projections scattered in N-d space with lots of false drops of data interleaved amongst desirable records
- Cause significant I/O increases due to increased number of seek times on disk
- performance metric Sequential I/O Ratio (SIR)



$$\text{SIR} = \frac{\text{Smallest \# of blocks required to store the result of a query}}{\text{Total \# of blocks retrieved during a query}}$$

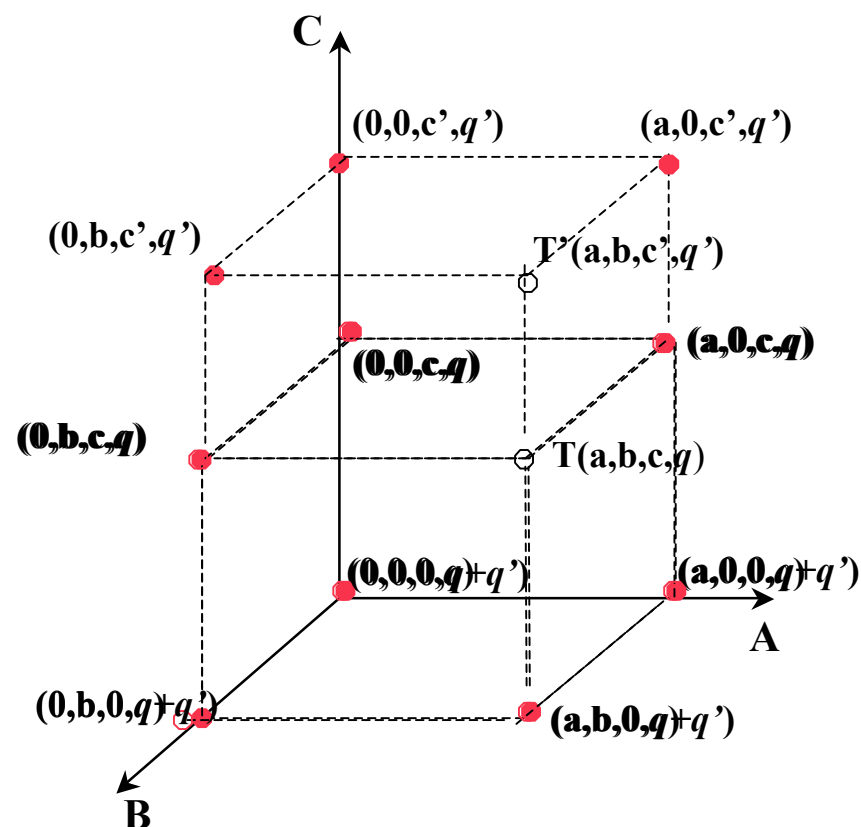
SIR = 100% is perfect
= 50% implies every other block is unnecessary



All COTS in 5-6d give SIR values < 2% unless ...

BIGGEST CHALLENGE: In-place Update problem

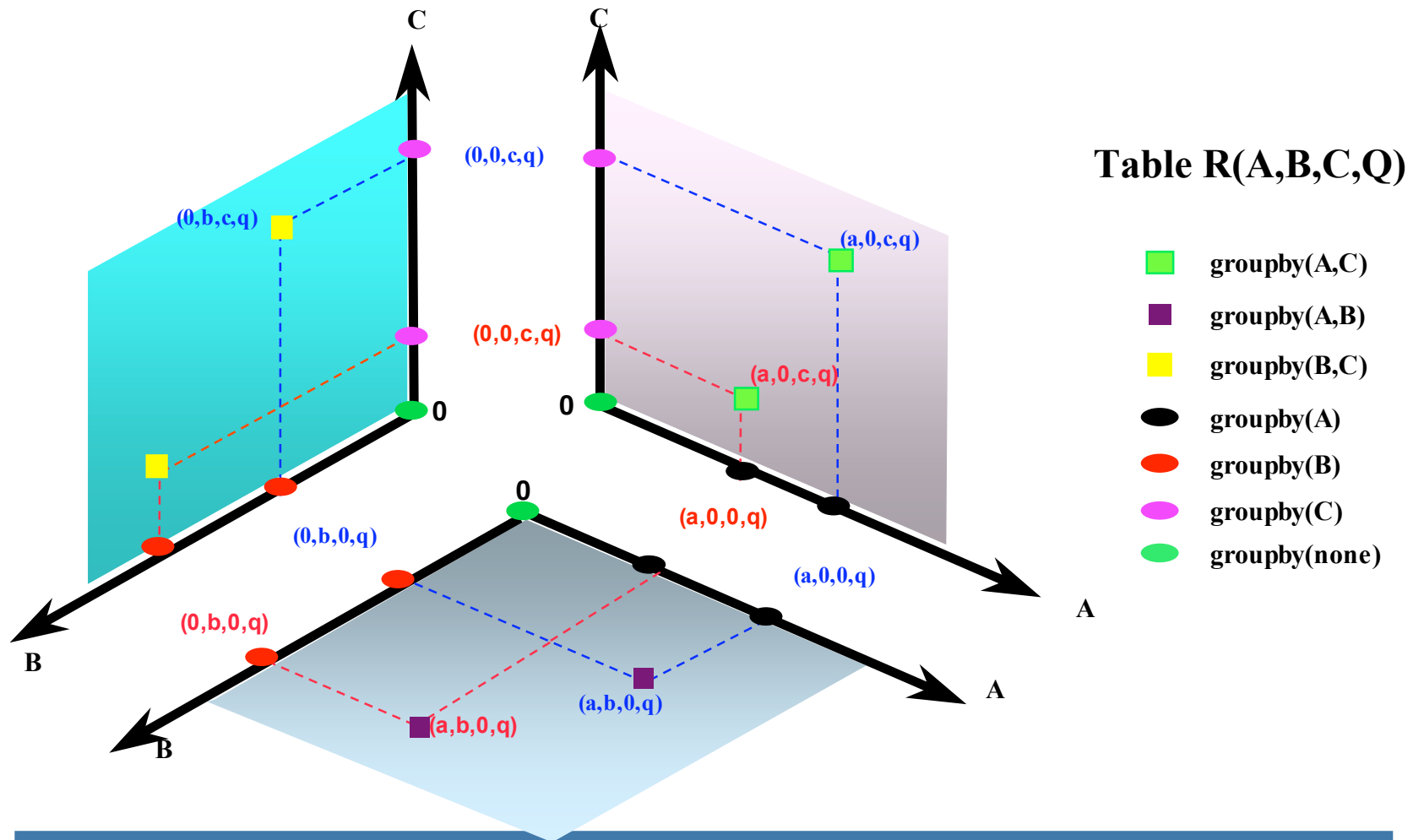
- each record in the fact table may update exponential number of other points (in 3-d $2^3 = 8$ points)
- **record-at-a-time updates are**
 - too expensive in terms of I/O
 - destroys clustering of data points
 - **Kills indexes**
 - main reason for **SIR < 2%**



No COTS offers bulk incremental update

Projections on Hyperplanes Dimensionality Reduction to N-1-d

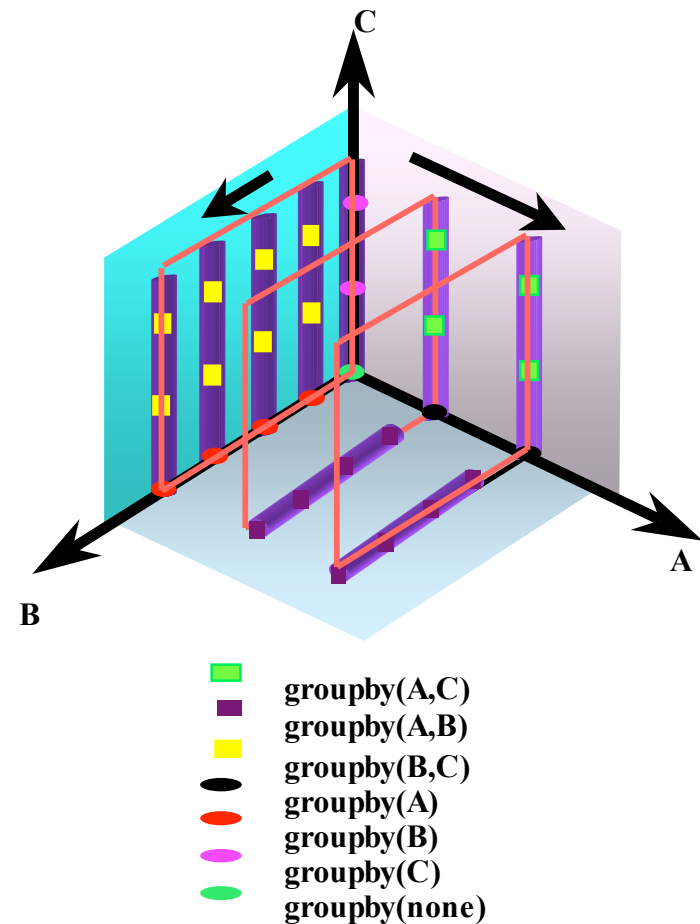
- ◆ Reduced R-trees result in better clustering



The Cubetree Storage Organization (CSO)

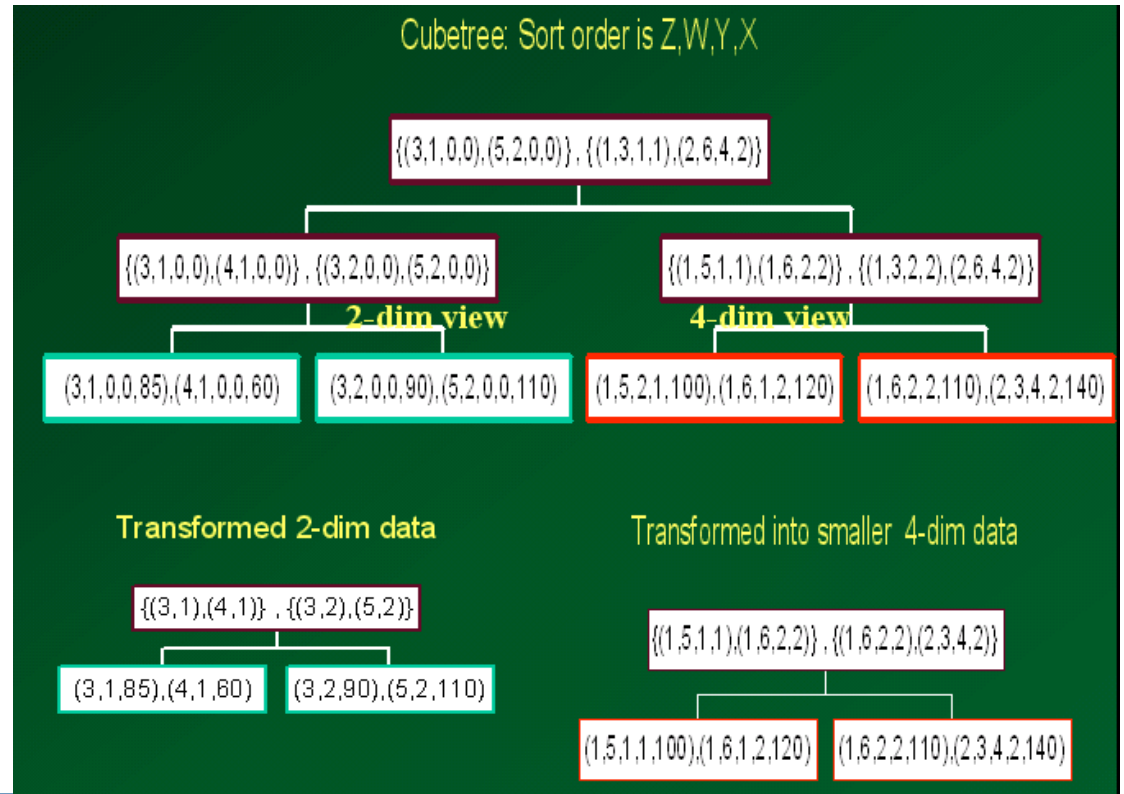
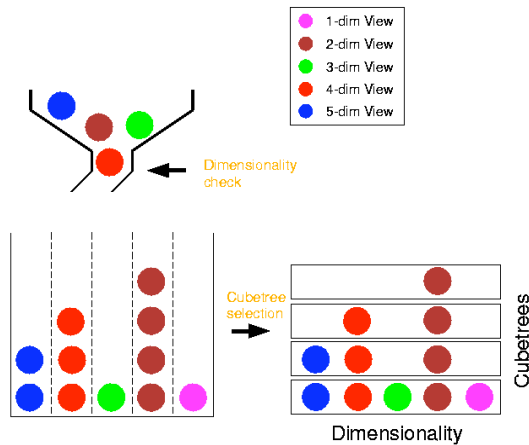
1. Sort Order

- CSO has a Unique sort order that maximizes SIR and organizes the data layout
- packed R-trees based on the CSO order
- incremental bulk load



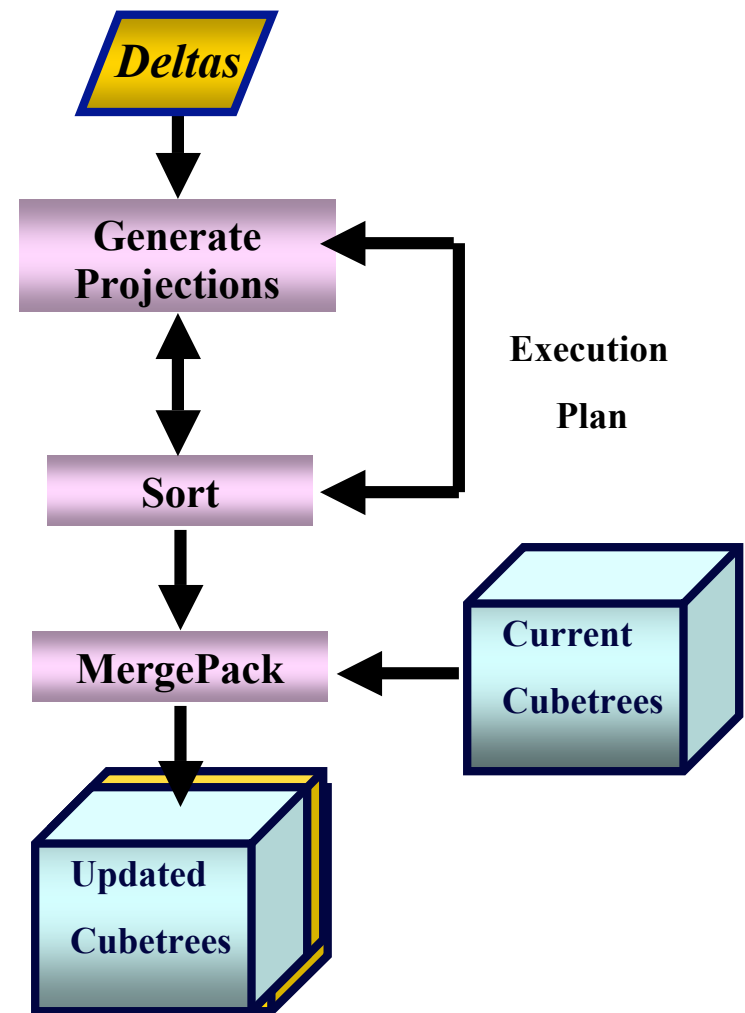
2. Multi-resolution and Storage Compression

- Each view occupies a distinct and continuous string of leaves
- *Multi-resolution* Packed Cubetrees: only useful coordinates are stored
- Shared container tree structure (more buffer hits)
- More than 90% compression



3. Incremental Bulk Updates

- The CSO sort order permits MergePack
- 100 times faster than bitmap indexes
- A rewrite disk strategy
 - 1 sequential read for deltas
 - 1 sequential read for current cubetrees
 - 1 sequential write for updated cubetrees
 - 100% SIR
- Current cubetrees are used up to the point of the switch
- No update down time



CSO Performance Data

- CubeTree Storage Organization (SIGMOD 97-98)

- 2-1 space reduction over relational tables
- 10-1 faster queries



- Merge-Packing

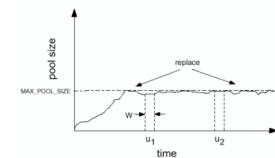
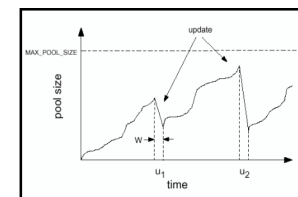
- Updating the views translates to sort & merge the deltas with the current data
- Serial I/O up to using 3 disks: **32GB/Hour**
(RedBricks on a RAID less than 14GB/Hour)



No COTS can match CSO bulk incremental update

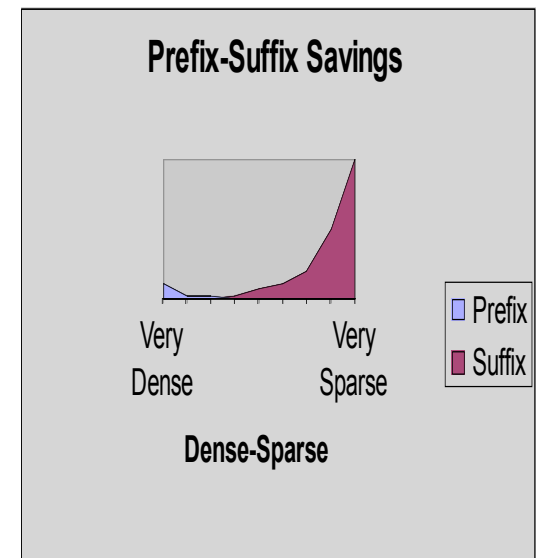
- DynaMat (SIGMOD 99, Best paper award)

- Dynamic management of query results- saves and reuses
- Computes other views from existing views
- dynamic update window (time or space)



Dwarf

- Stores and indexes full Data Cubes Lilliputian footprint
- Eliminates the “dimensionality-curse”
- Combines all views in a single DAG
 - Index comes for free during creation
- Factors out inter-view redundancies
 - Prefix: high in dense areas (becomes smaller than the fact table)
 - Suffix: extremely high in sparse areas (exponential savings with sparsity)
 - Removing redundancy prior to building the Dwarf Cube
- Complete solution with queries, incremental updates



Prior Techniques for Full Cubes

- Precompute only a subset of the cube
 - [HRU96, GHRU97, Gup97, SDN98,...]
- Estimate the values of group-bys using approximation techniques
 - [GM98, VWI98, SFB99, AGP00, ...]

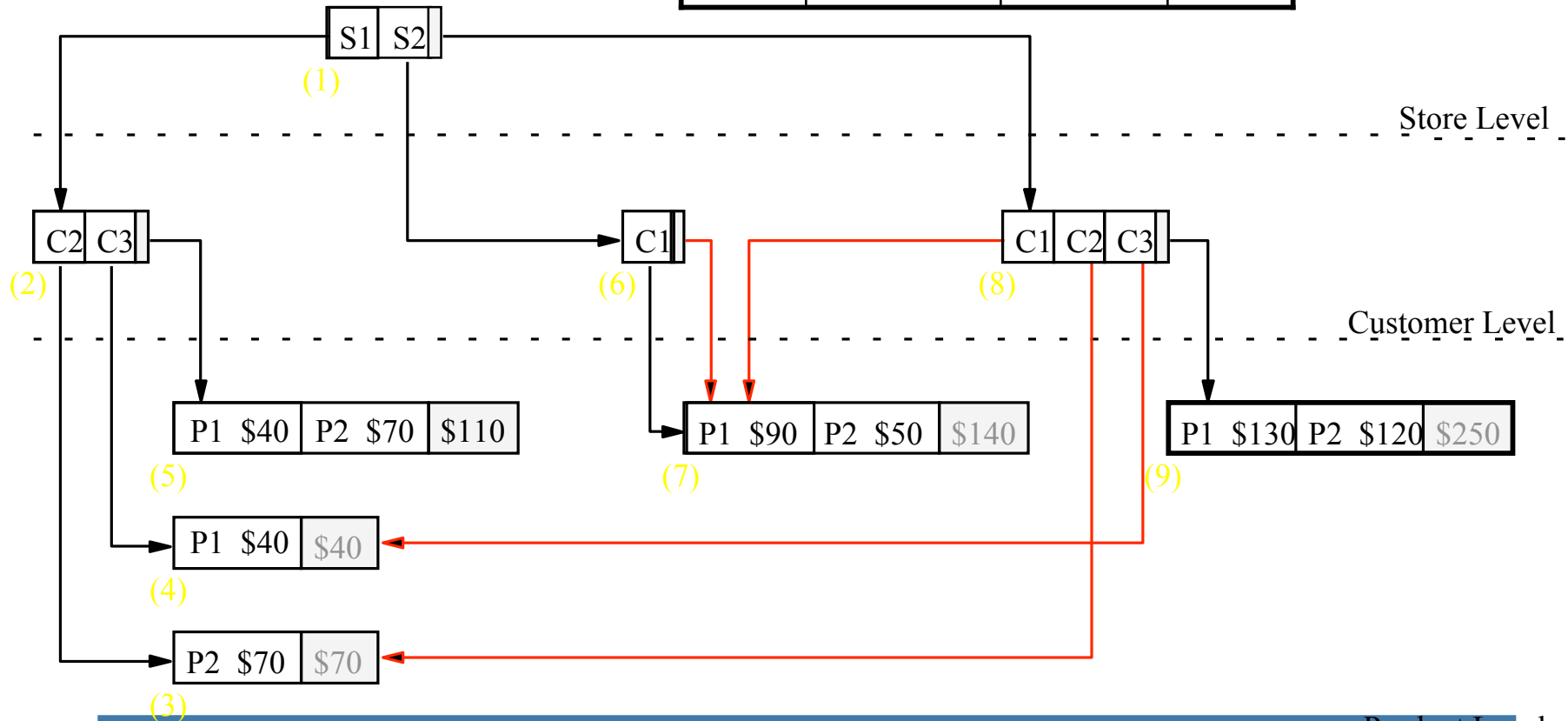
Dwarf Properties

- DAG w/ d levels ($d = \text{num of Dims}$)
 - Leaf nodes contain cells [key,aggr]
 - Non-leaf nodes contain cells [key,ptr]
 - *Special ALL cell*
- Important properties:
 - Unique prefixes are stored once
 - Identical nodes and cells are *coalesced*

A Dwarf Example

Fact Table →

Store	Customer	Product	Price
S1	C2	P2	\$70
S1	C3	P1	\$40
S2	C1	P1	\$90
S2	C1	P2	\$50

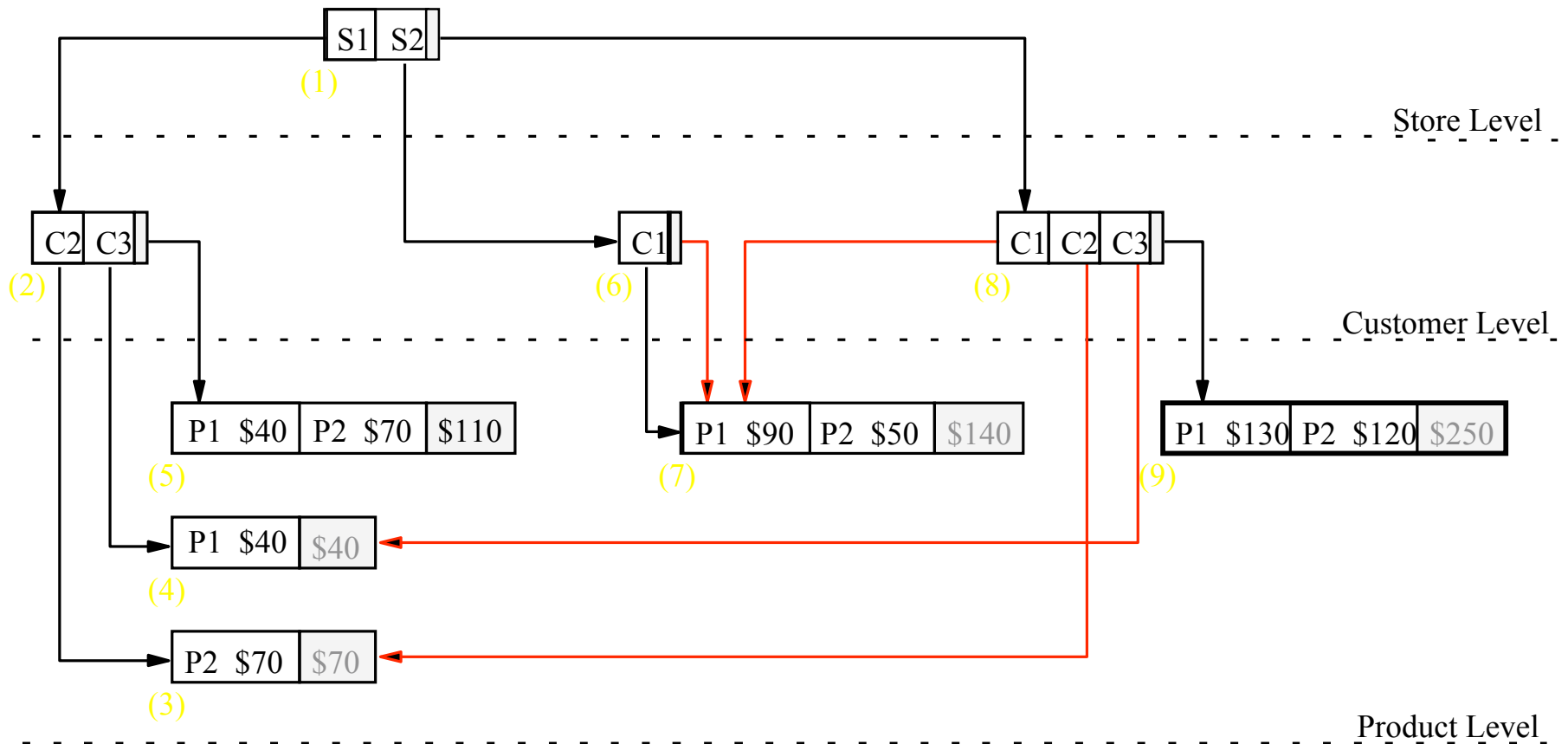


Dwarf Construction

- Governed by two interleaved processes:
 - Prefix expansion
 - Suffix Coalescing
- Single pass over the sorted fact table
 - If the processes were not interleaved, they would need enormous temporary space and time

Querying the Dwarf

Store	Customer	Product	Sum(Price)
S1	ALL	ALL	\$110
S2	ALL	ALL	\$250

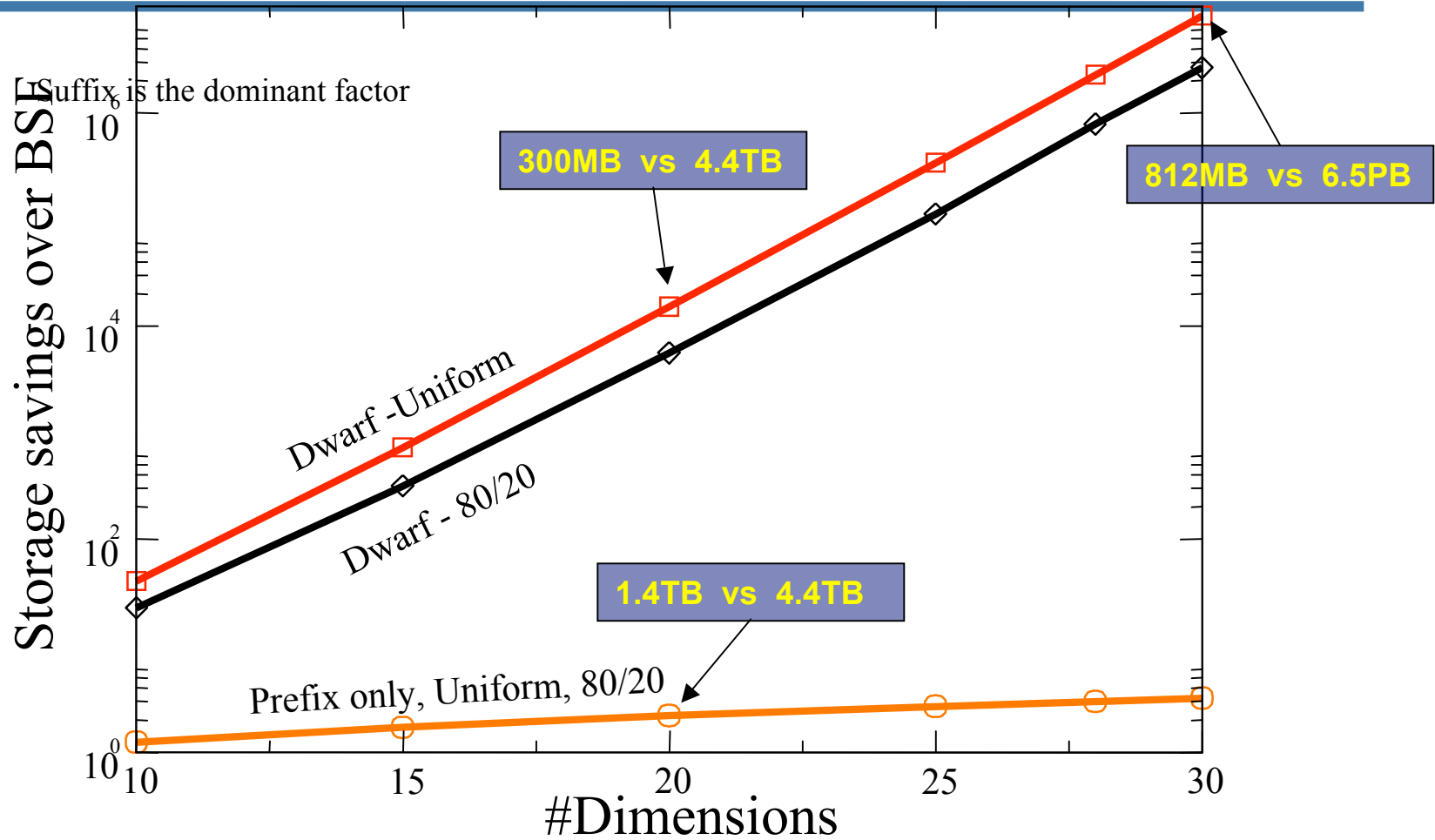


Experiments

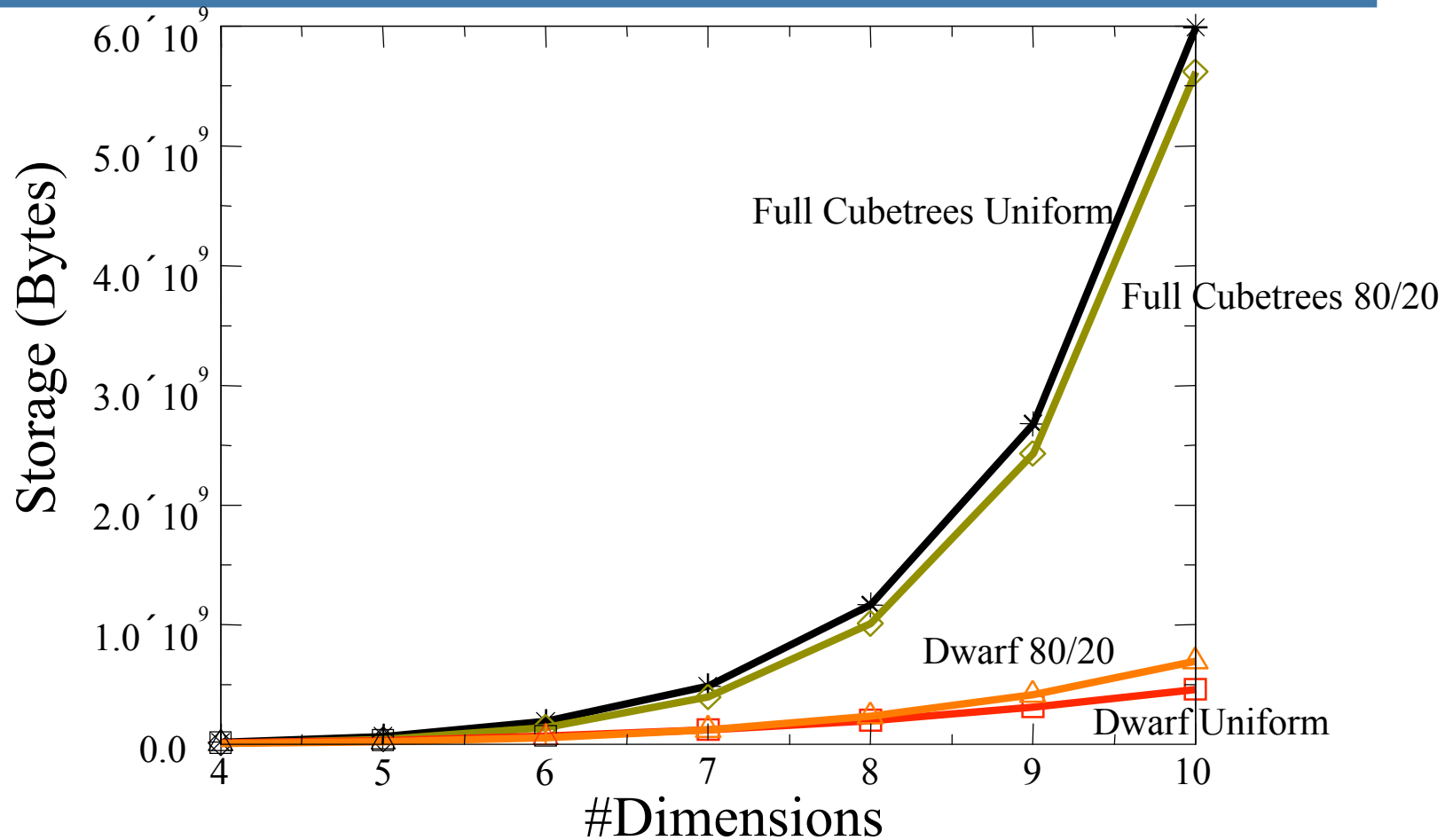
- Measurement:
 - Prefix and suffix savings
 - Storage
 - Creation time
 - Query time
 - Update
- Comparison with:
 - Full Cubetrees
 - Reduced Cubetrees

Prefix and Suffix Savings

- Suffix is the dominant factor

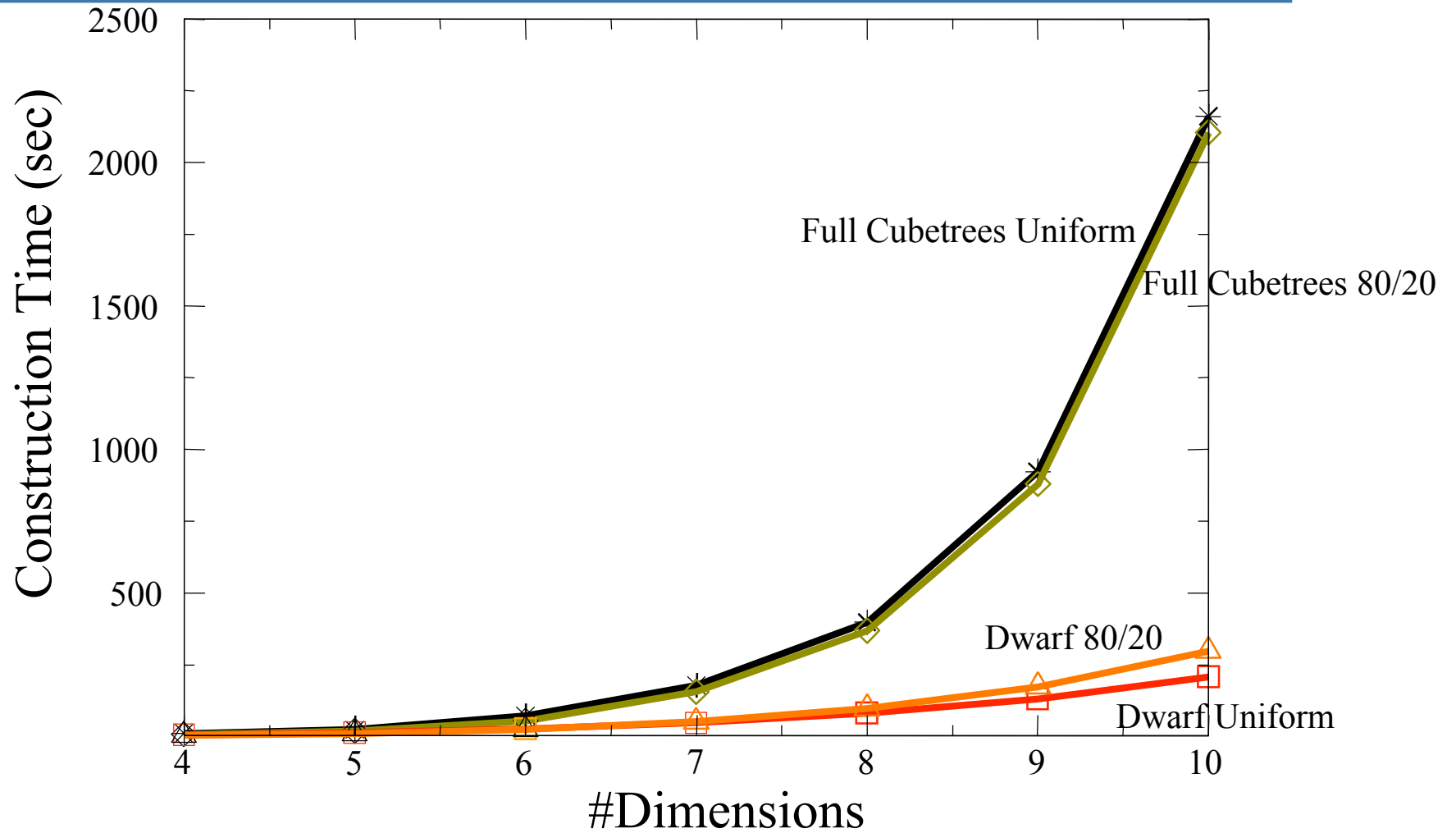


Storage Comparison



- Dwarf scales much better to many more dimensions

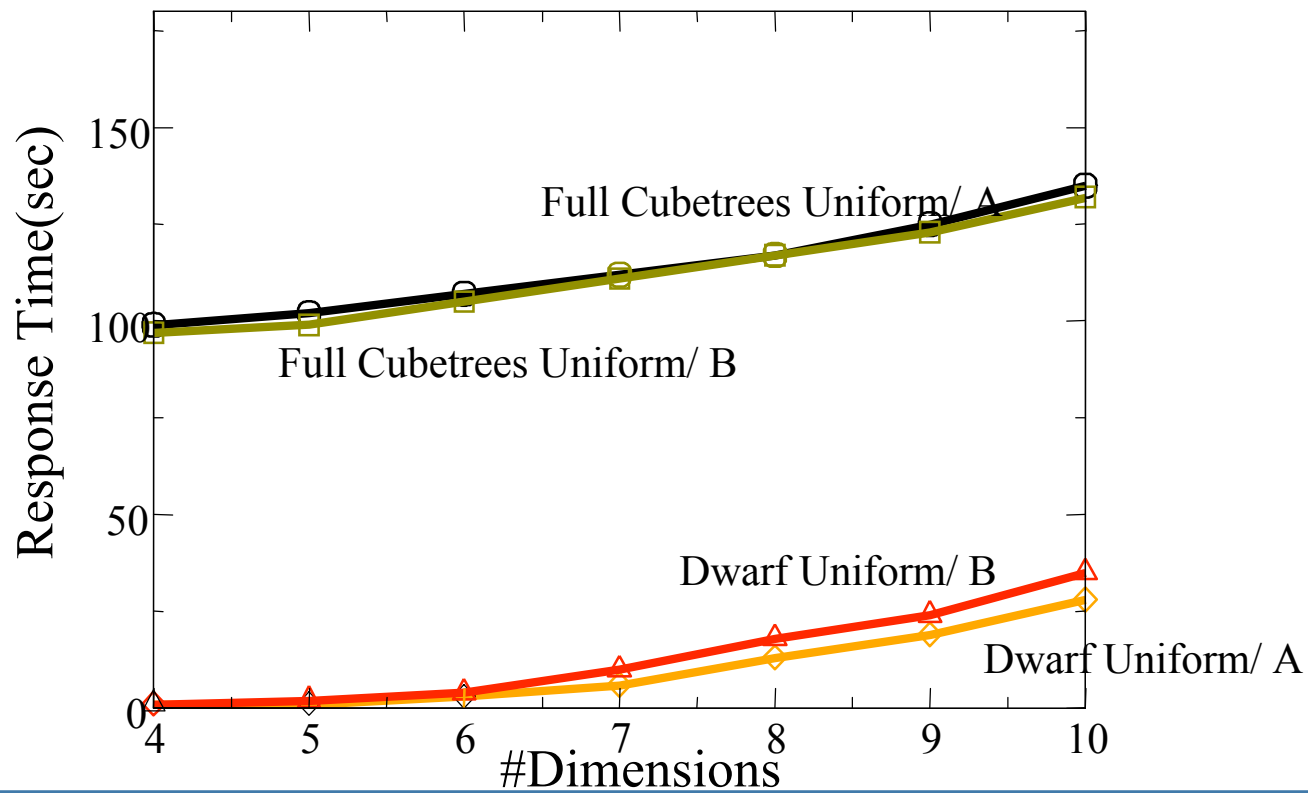
Creation Comparison



- Construction time scales equally well

Query Response

WL	#Queries	Probabilities			Range	
		NewQ	Dim	Point	Min	Max
A	1000	0.34	0.4	0.2	1	20%
B	1000	1	0.4	0.2	1	20%

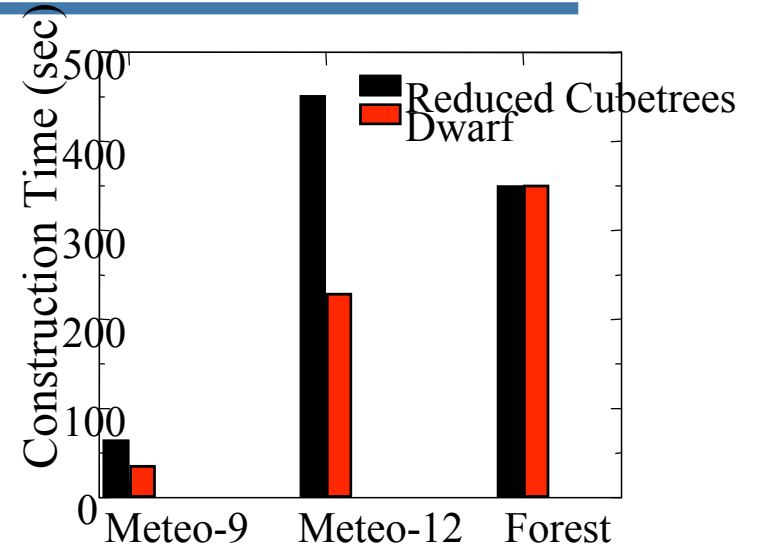


Dwarfs vs Reduced Cubetrees

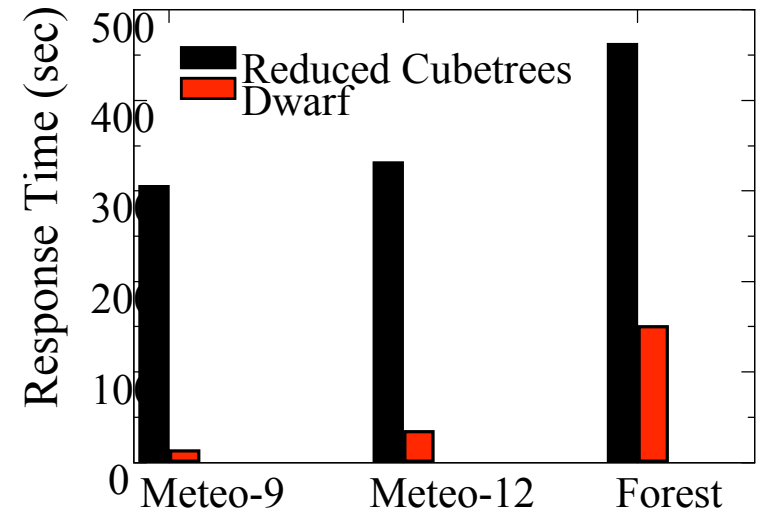
- Real datasets
- More pragmatic approach
 - Materialize parts of cube for Cubetrees
- PBS
 - Materialize “most” aggregated views first
- Same storage size

Real Datasets

Dataset	d	Size(MB)	PBS
Meteo-9	9	66	63/512
Meteo-12	12	358	310/4096
Forest	10	594	113/1024



#Queries	P_{newQ}	Dims	P_{pointQ}	Range _{Max}
2000	0.34	4	0.1	15%



Extensions

- Granularity GMin
 - Only compute the ALL aggregates if the number of tuples per group by is above GMin
 - Further reduces the size at minor expense of querying (not a linear tradoff)
- Clustering
 - Essential for good query performance
 - Speeds up creation as well
- Hierarchies
 - See next paper

Hierarchical Dwarfs For the Rollup Cube

Sismanis, Deligiannakis, Kotidis, Roussopoulos DOLAP 2003

Hierarchies			Declared Metadata	
Store	Product	Customer	Dimension	Metadata
ALL	ALL		Store	S1 → R1
↑	↑		Store	S2 → R1
Retailer	Group	ALL	Store	S3 → R2
↑	↑	↑	Product	C1 → G2
StoreId	Code	Name	Product	C2 → G1
			Product	C3 → G2
			Customer	N1
			Customer	N2

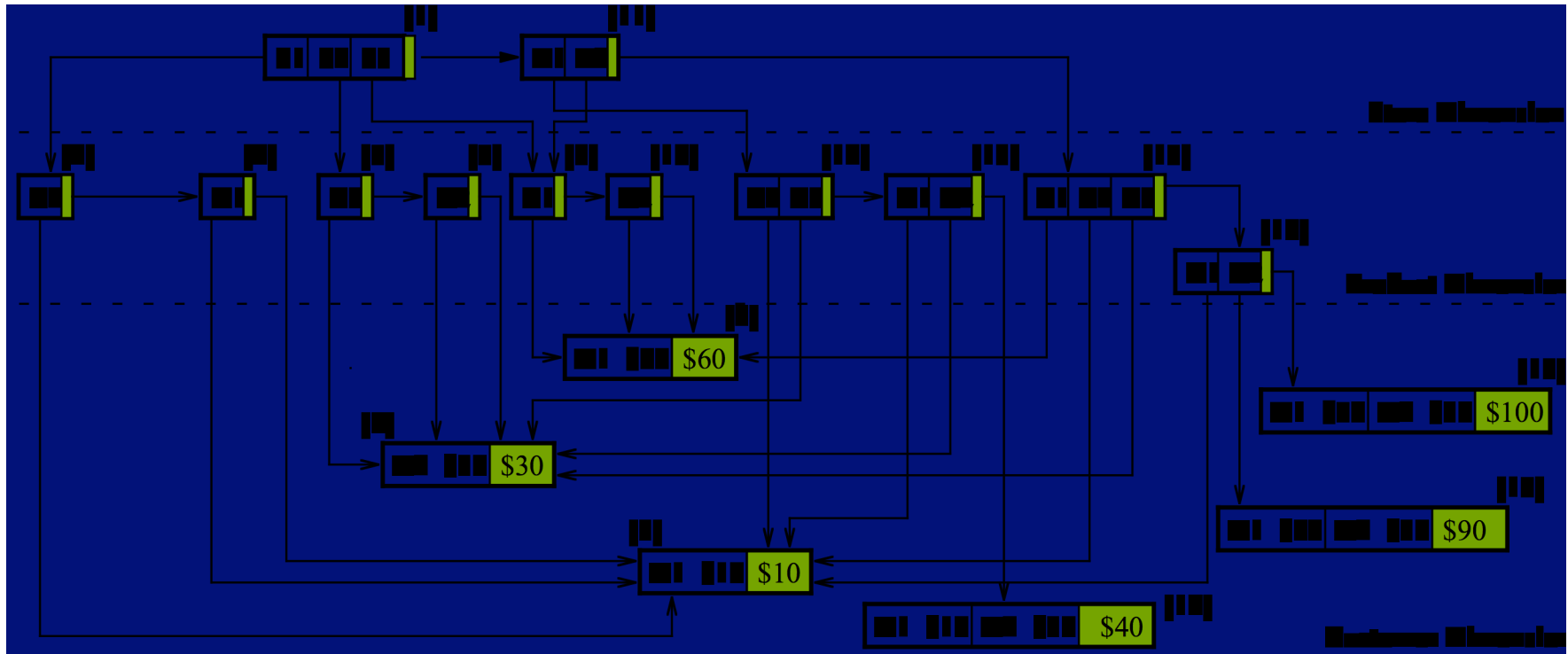
Table 1: Example of declared Hierarchies

Hierarchical Dwarfs

Store	Code	Name	Sales
S1	C2	N1	\$10
S2	C3	N2	\$30
S3	C1	N1	\$60

Store	Code
S1→R1	C1→G2
S2→R1	C2→G1
S3→R2	C3→G2

Table 3: Sample Fact Table



Revolutionary Technology

- Highly compressed storage
 - Full Cubes: ALL views answerable
 - Extends to high dimensionality
 - Deep hierarchies
 - 100% Precision answers on all views including the fact table
 - Stores a subset of the views in very tight space
- Tremendous savings
 - Storage
 - Construction time
- Efficient Query Retrieval
 - Sub-second response
 - Queries the full cube- any dimension & level
- Complete solution
 - Incremental updates
 - Indexing is inherent – all in one structure
- No gotchas
 - No expensive preprocessing (just a single sort)
 - No TEMP space required for construction
 - No hidden post-construction costs

APB-1 Benchmark

- Density 1 (**1.3M**)
 - Dwarf (Thinkpad): **18 s** **57 MB**

- Density 5 (**65M**)
 - Oracle's best benchmark (4 CPU, RAID) **4.5 hrs,** **30.0+ GB**
 - Dwarf **65 min** **2.4 GB**
(Single CPU Pentium 4)

- Density 40 (**496M**)
 - Dwarf: **10.3 hrs** **8.2 GB**
(Single CPU Pentium 4)

Never done before

NOTE: fact table is **32GB in ASCII, 11.8GB in Binary**

Real Data Sets

- Real data set (13,449,327):
 - **Dimensions:** 8
 - **Views:** 11,200
 - **Creation time:** 100 min
 - **Size:** 6.7 GB
 - **1000 Queries*:** 15.8 sec

Dimension	Level Cardinalities
A	7458 → 2265 → 737 → 188 → 32 → 11
B	2765 → 91 → 31 → 8
C	3857 → 841 → 111 → 16
D	213 → 68 → 8
E	3247
F	660
G	4
H	4

Table 4: Real Dataset Hierarchies

- **Challenge by XYZ**
 - 48 hrs for a “wizard” to decide what to materialize
 - Several more hrs to create and index summary tables
 - Huge storage

* Each query asks for 10 different values for 3 randomly selected dimensions (e.g. $v1 | v2 | \dots | v10$) and “all” for a 4th dimension- $10*10*10$ point query

Dream DataCube

- **Fact table (5,000,000):**
 - **Dimensions:** *10 (3x9L, 4x4L, 3x2L)*
 - **Views:** **16,875,000**
 - **Creation:** **123 min**
 - **Size:** **6.3 GB**
 - **1000 Queries*:** **325 sec**

- **Challenge by XYZ**
 - **This cube can never be built!**

Data Driven Tuning

- GMin:
 - Min # of records per aggregate group to be explicitly stored
 - Reduces storage in lower levels of the hierarchies
- The Knob:
 - Max # of aggregations to be computed on the fly (not stored)
 - Reduces storage in the higher levels of the hierarchies

Data Driven Tuning

- Gmin

G_{min}	Space(MB)	Construction(sec)	Queries(sec)
0	490	202	154
100	400	74	110
1000	312	59	317
5000	166	29	408
20,000	151	25	476

Patented

- “The Knob”

			Workloads	
Knob	Computation	Storage	A	B
0	4860s	6.6GB	282s	340s
100	3388s	3.1GB	209s	249s
500	2038s	2.1GB	198s	238s
1,000	1794s	1.5GB	186s	222s
10,000	768s	806MB	191s	229s
Base Dwarf				
N/A	552s	764MB	1331s	1706s

Patented

Table 9: Knob Evaluation with 13,5 million tuples

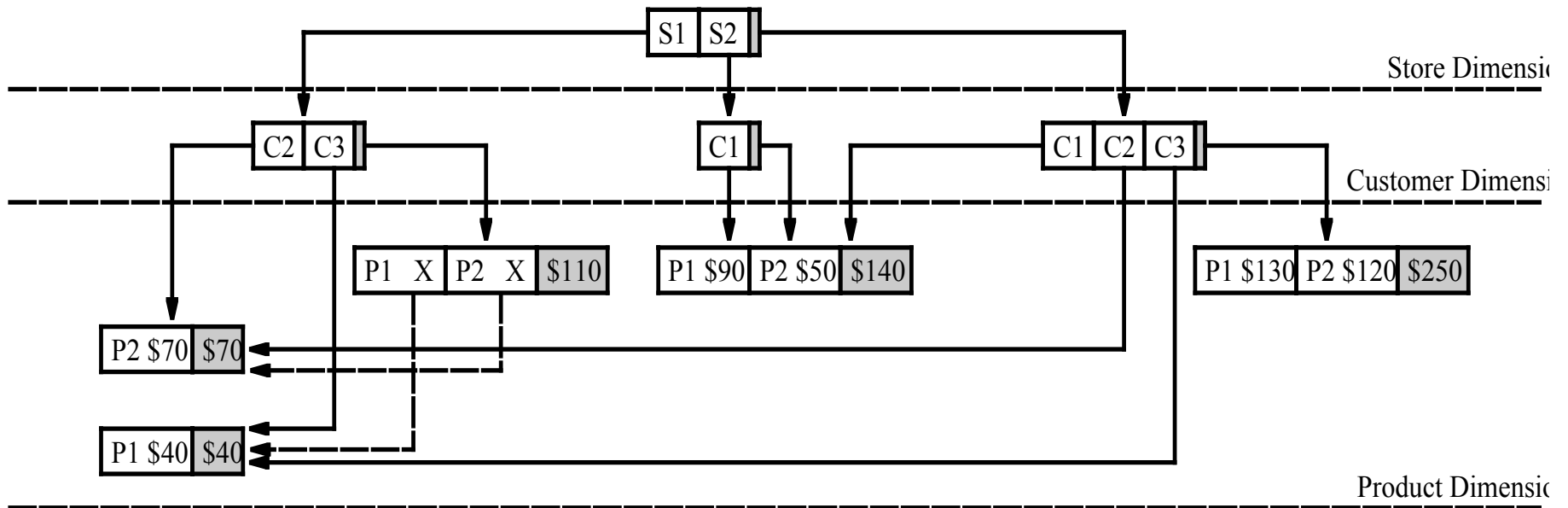
The Polynomial Complexity of Fully Materialized Coalesced Cubes

Sismanis & Roussopoulos VLDB 2004

- Analytical framework for estimating the size of coalesced cubes (Dwarfs)
- Prior work using probabilistic techniques [SDNR VLDB96] was very inaccurate and did not measure coalescing

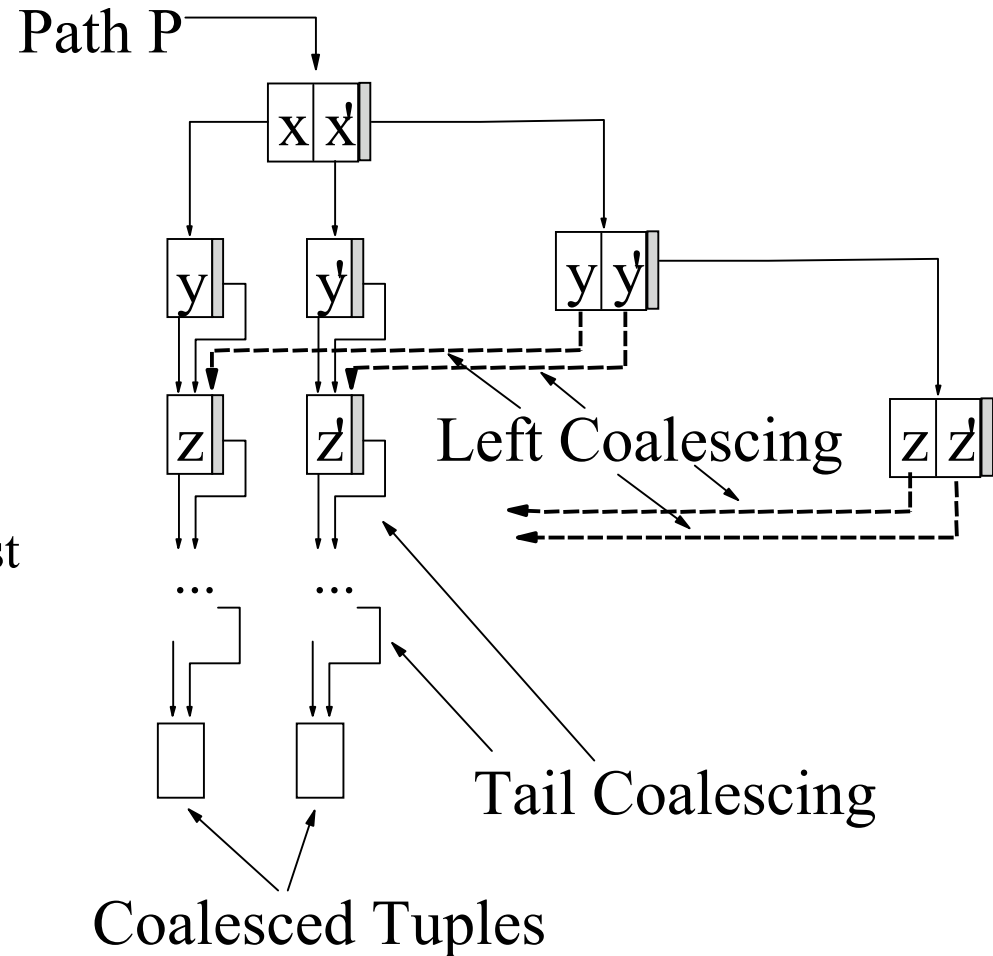
Dwarf Example - Visualization

<i>Store</i>	<i>Customer</i>	<i>Product</i>	<i>Price</i>
S1	C2	P2	\$70
S1	C3	P1	\$40
S2	C1	P1	\$90
S2	C1	P2	\$50



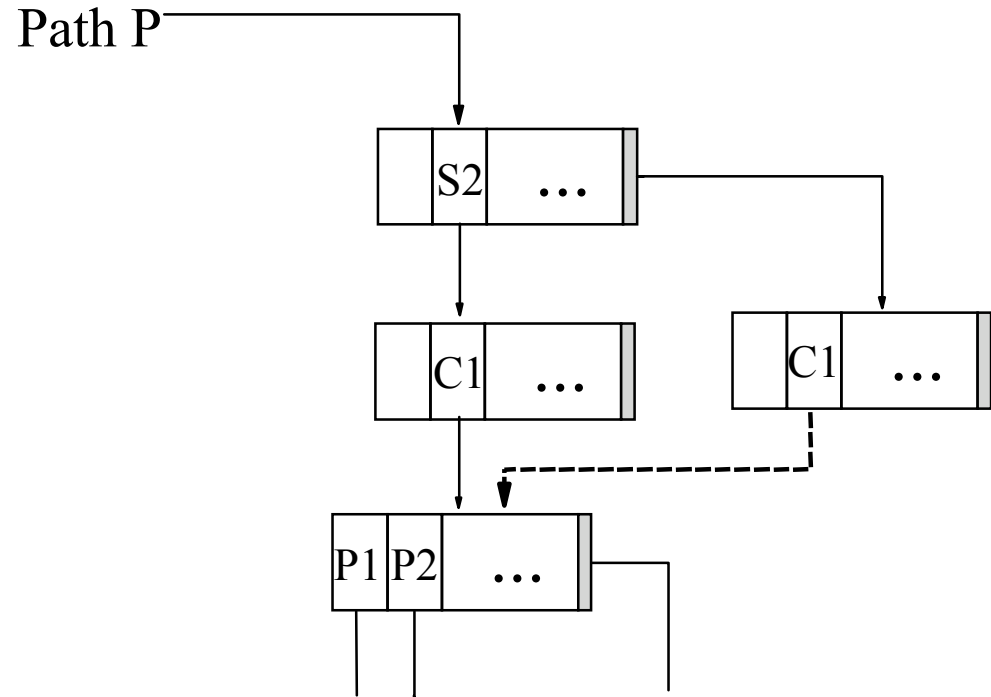
Suffix Redundancies/Sparsity Coalescing

- Example:
Path P points to a sparse area with two tuples
 - $\langle x, y, z, \dots \rangle$
 - $\langle x', y', z', \dots \rangle$
where $x < x', y < y', z < z', \dots$
- Tail Coalescing introduces just two aggregates
- Left Coalescing introduces no new aggregates

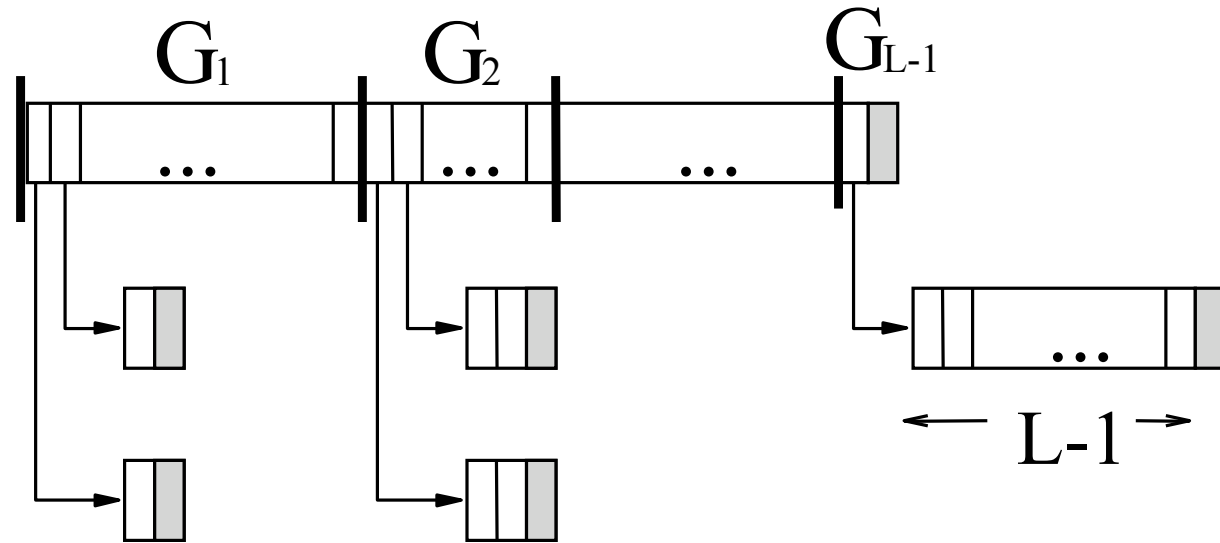


Suffix Redundancies/Implication Coalescing

- Example:
Path P points to an area where customer C1 *implies* store S2
- Identical aggregates for
 - $\langle \dots, S2, C1, \dots \rangle$ and
 - $\langle \dots, ALL, C1, \dots \rangle$
- Real datasets exhibit a large number of such correlation patterns



Basic Partitioned Node

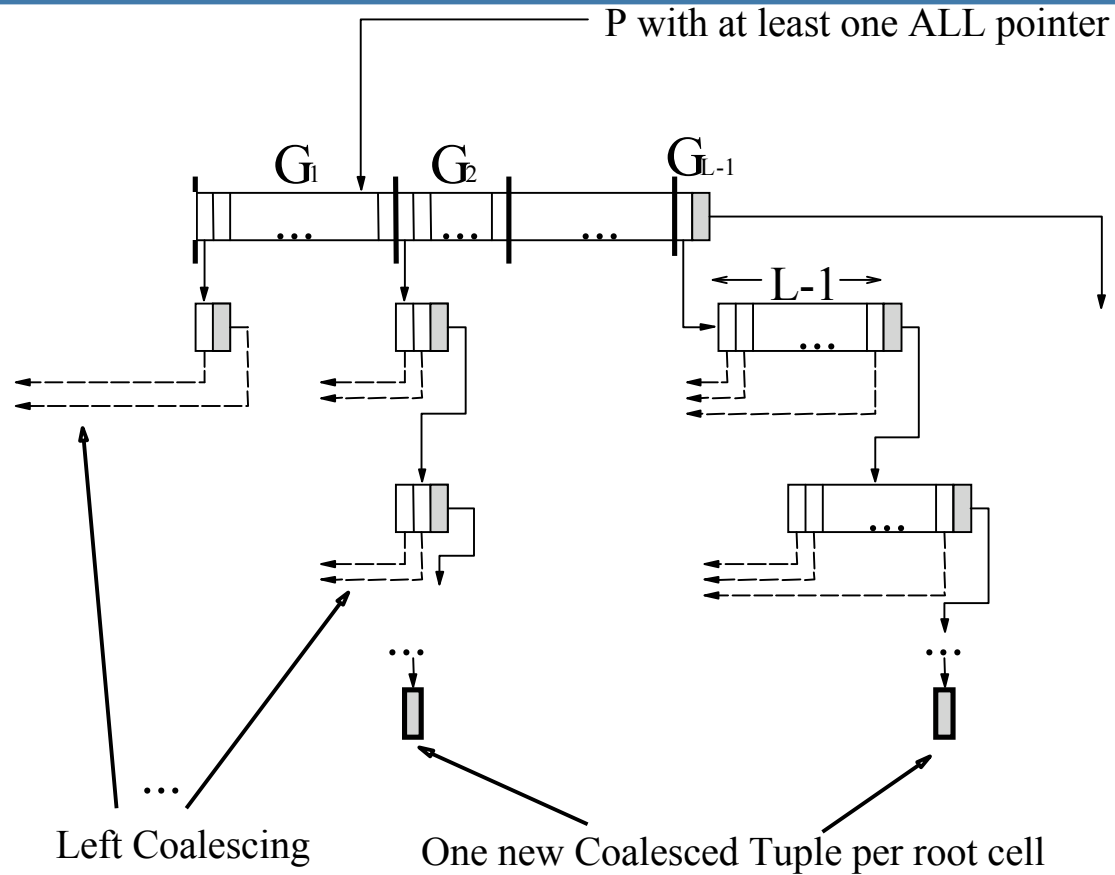


Cells in G_z point to nodes with exactly z cells

C card

$$P_z(C, T) = \frac{\binom{T}{z}}{(C-1)^z} e^{-T/C} \text{ s, } z \leq L-1$$

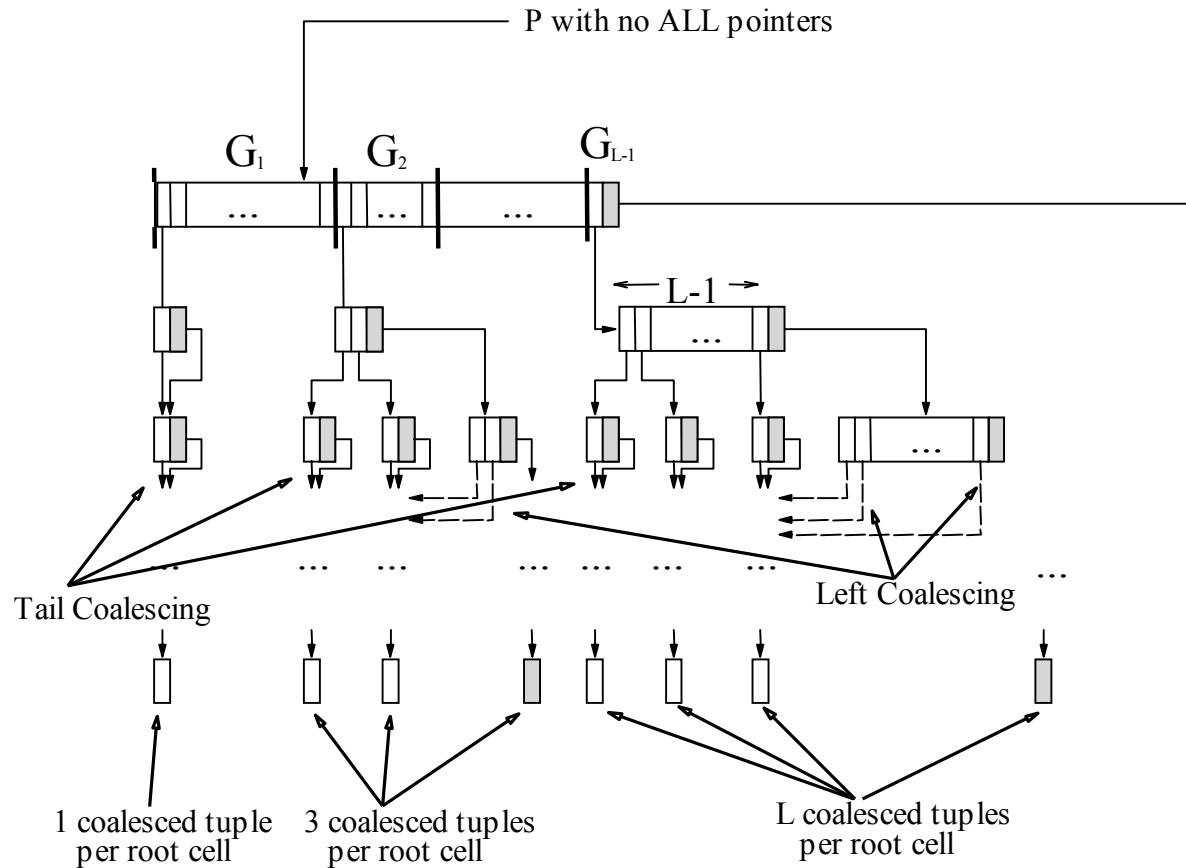
Left Coalesced Areas



$$\text{Coalesced Tuples} = a_0 C d + 1,$$

Where $a_0 = (e-2)/e$, $T=C$ is the cardinality, d the number of dimensions

Tail Coalesced Areas



$$\text{Coalesced Tuples} = b_0 C + a_0 C (d-1) + 1,$$

Where $a_0 = (e-2)/e$, $b_0 = (2e-2)/e$, $T=C$ is the cardinality, d the number of dimensions

Coalesced Size & Time Complexity

- d dimensions
- C cardinality for each dimension
- T tuples in the fact table

$$\#CoalescedTuples = O\left(T \frac{d^{\log_C T}}{(\log_C T)!}\right) = O\left(T^{1+1/\log_d C}\right)$$

$$\#TotalCells = O\left(T \frac{d^{\log_C T+1}}{(\log_C T)!}\right)$$

$$\text{Dwarf Computation Time} = O\left(T \frac{d^{\log_C T+1}}{(\log_C T)!}\right)$$

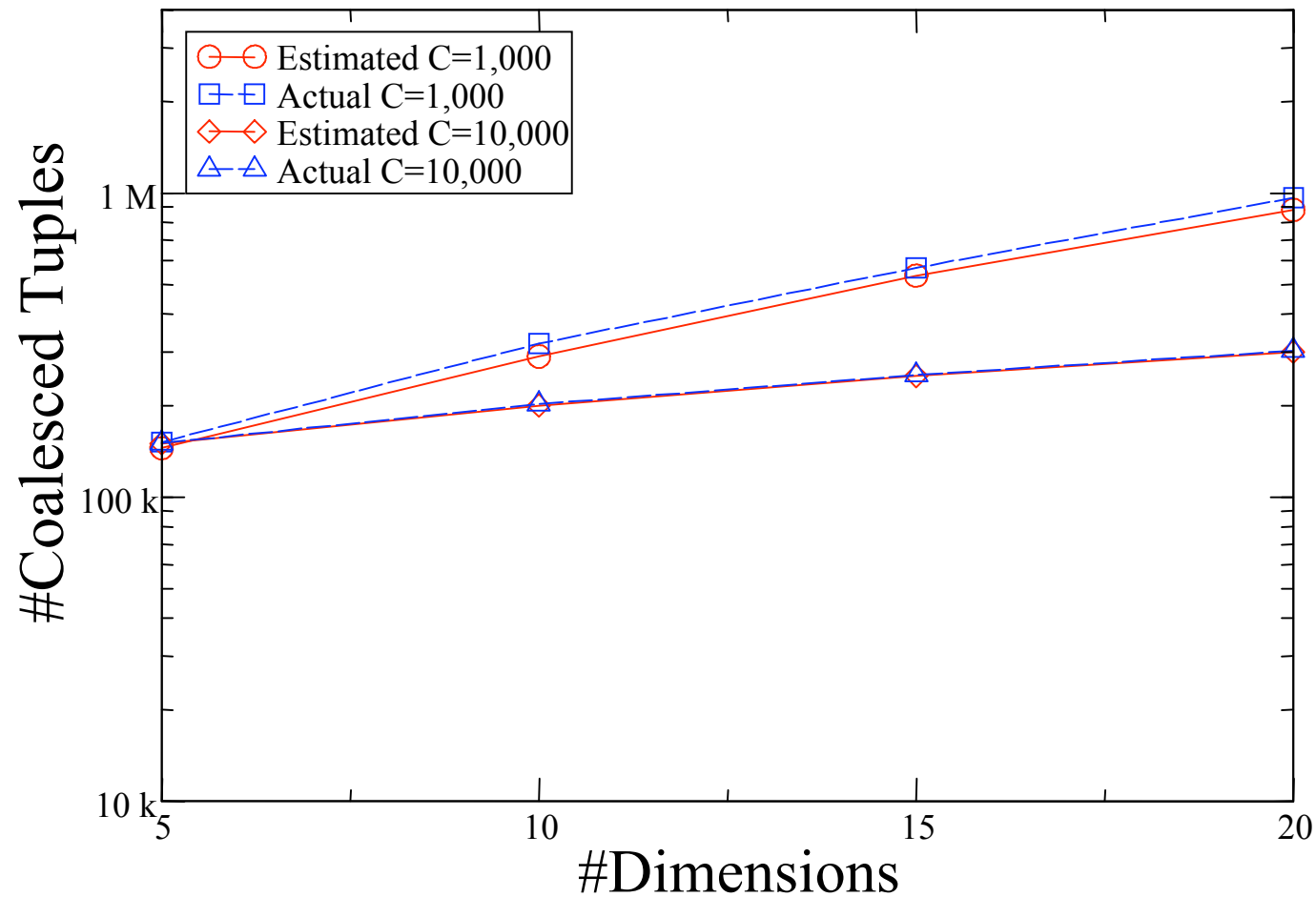
- $d = 100$
- $C = 1,000,000$
- $T = 100,000,000$
- $CT = O(T d^{2.33})/2$

Experiments

- Used Dwarf as reference platform
- Compared with the analytical framework
 - Required Size
 - Time
- Analytical Datasets
 - Uniform
 - Zipfian
- Real Datasets
 - Actual Data from OLAP company

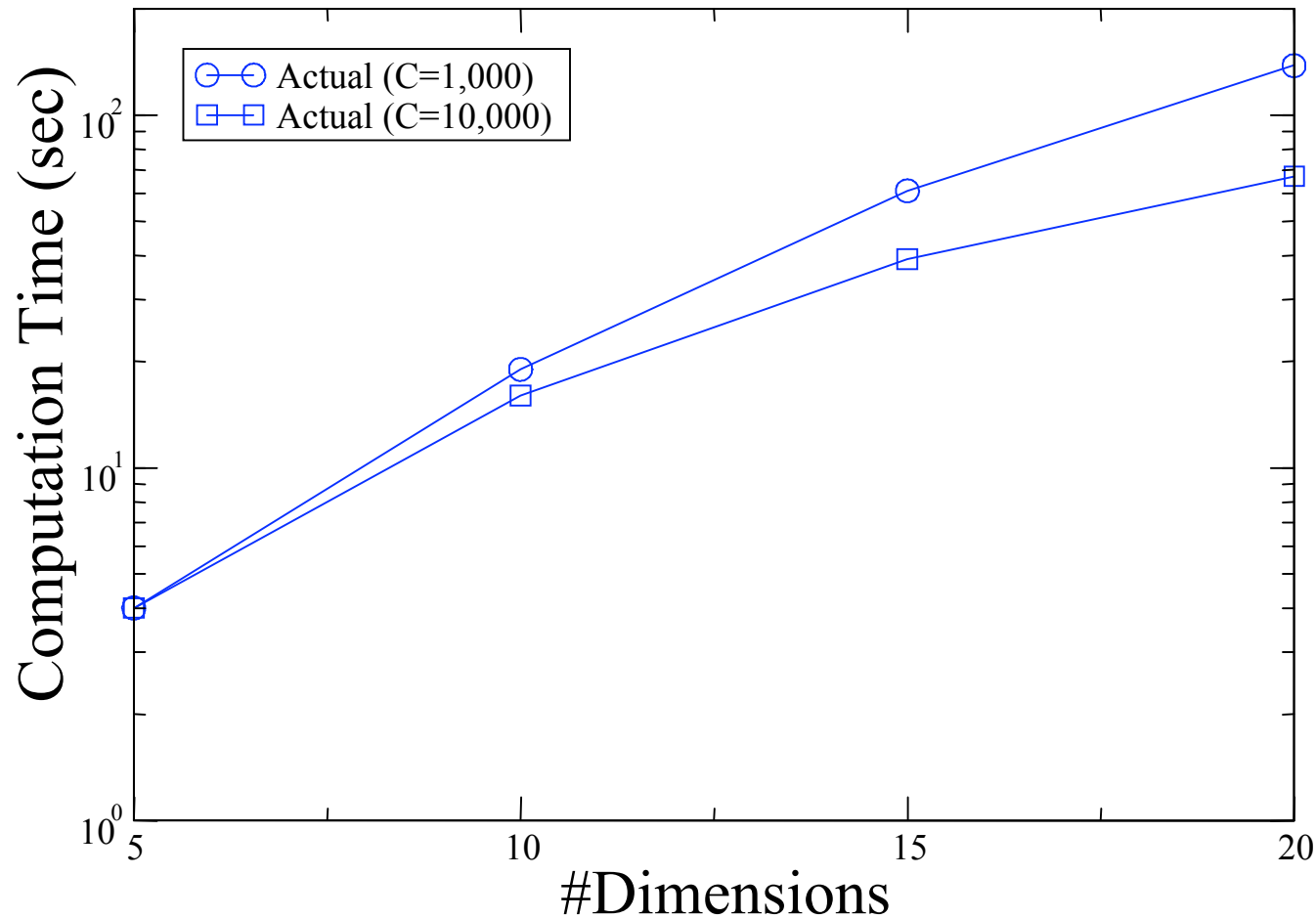
Size Scalability v.s. Dimensionality

Uniform Distribution



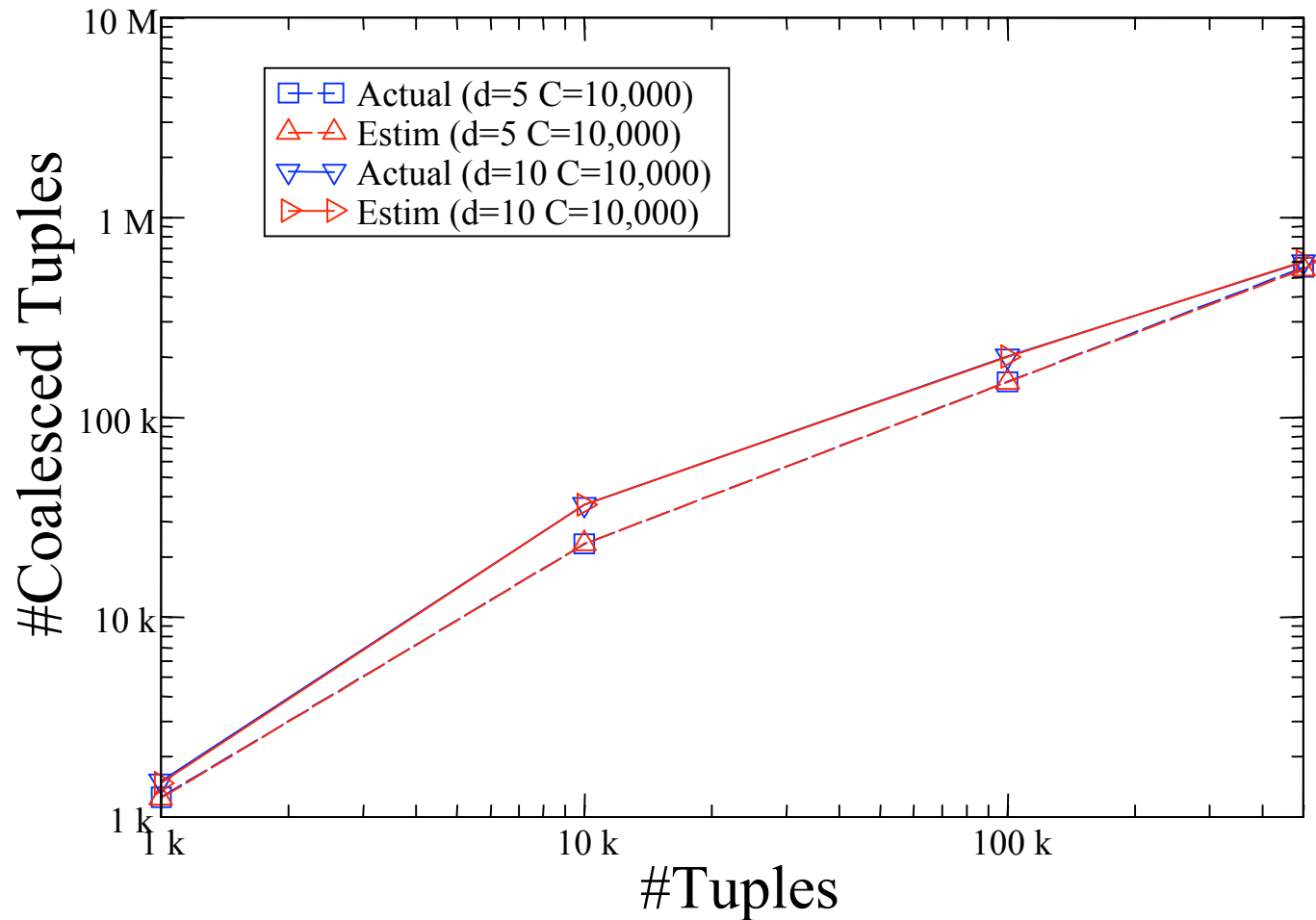
Time Scalability v.s. Dimensionality

Uniform Distribution



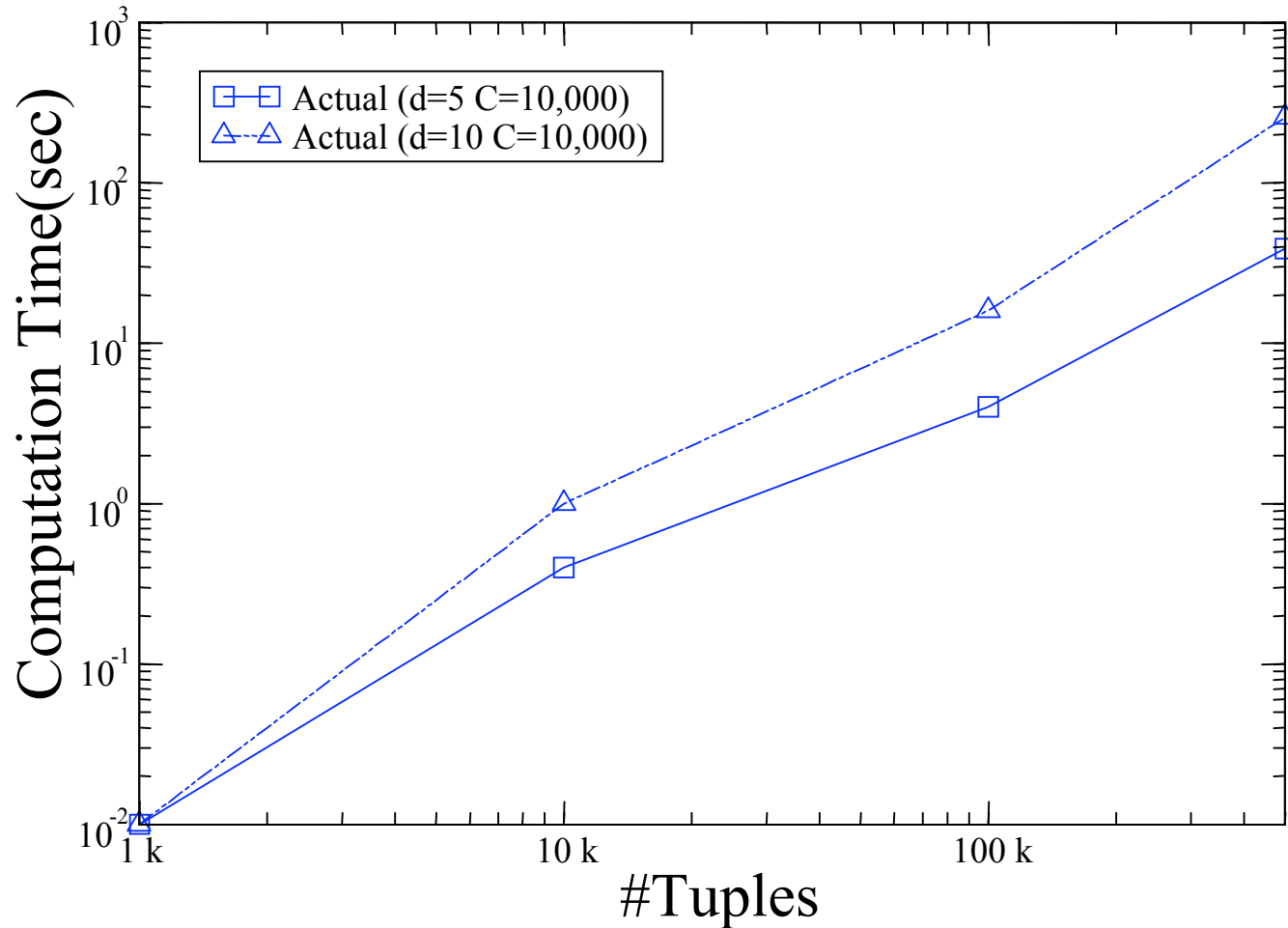
Size Scalability v.s. #Tuples

Uniform Distribution

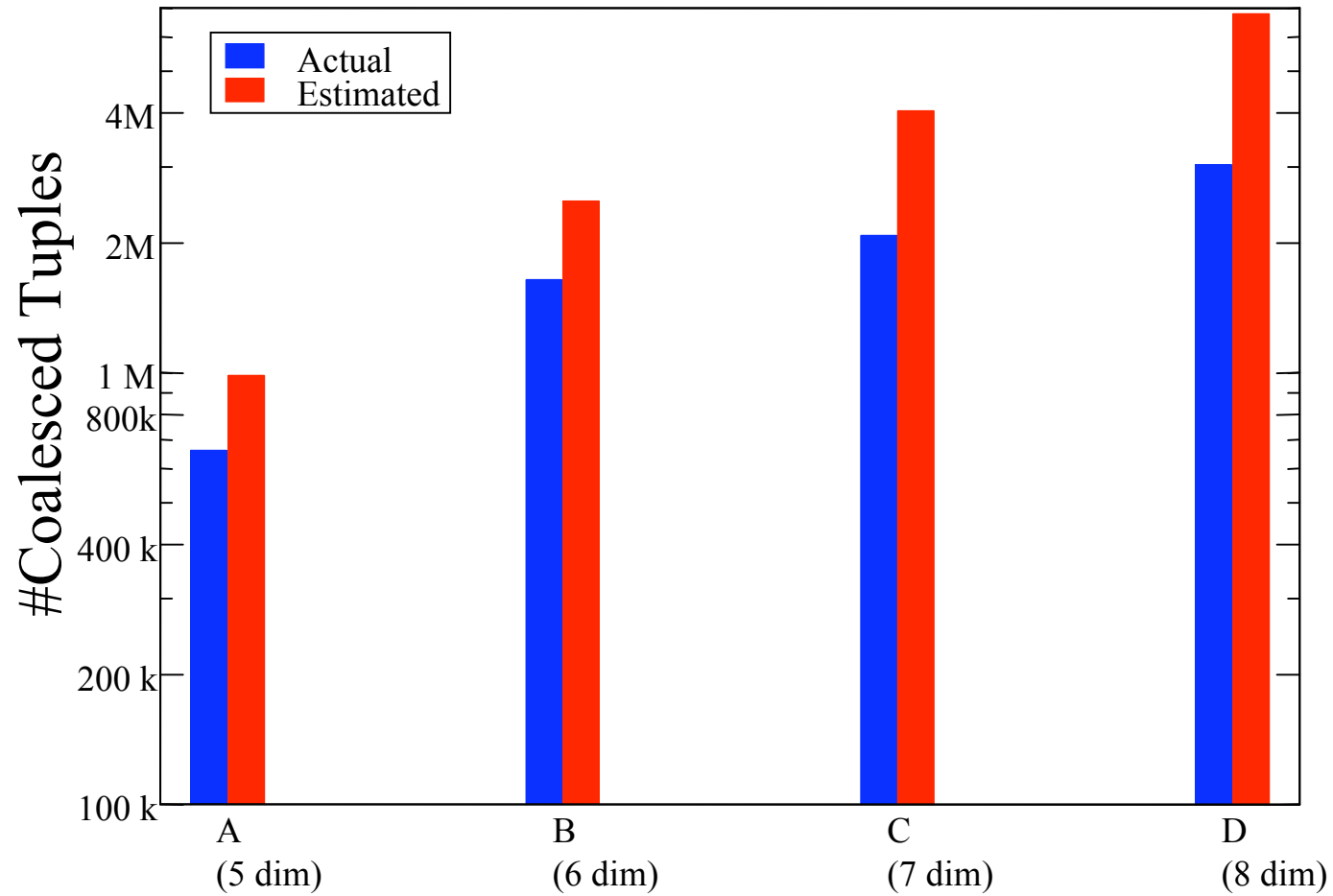


Time Scalability v.s. #Tuples

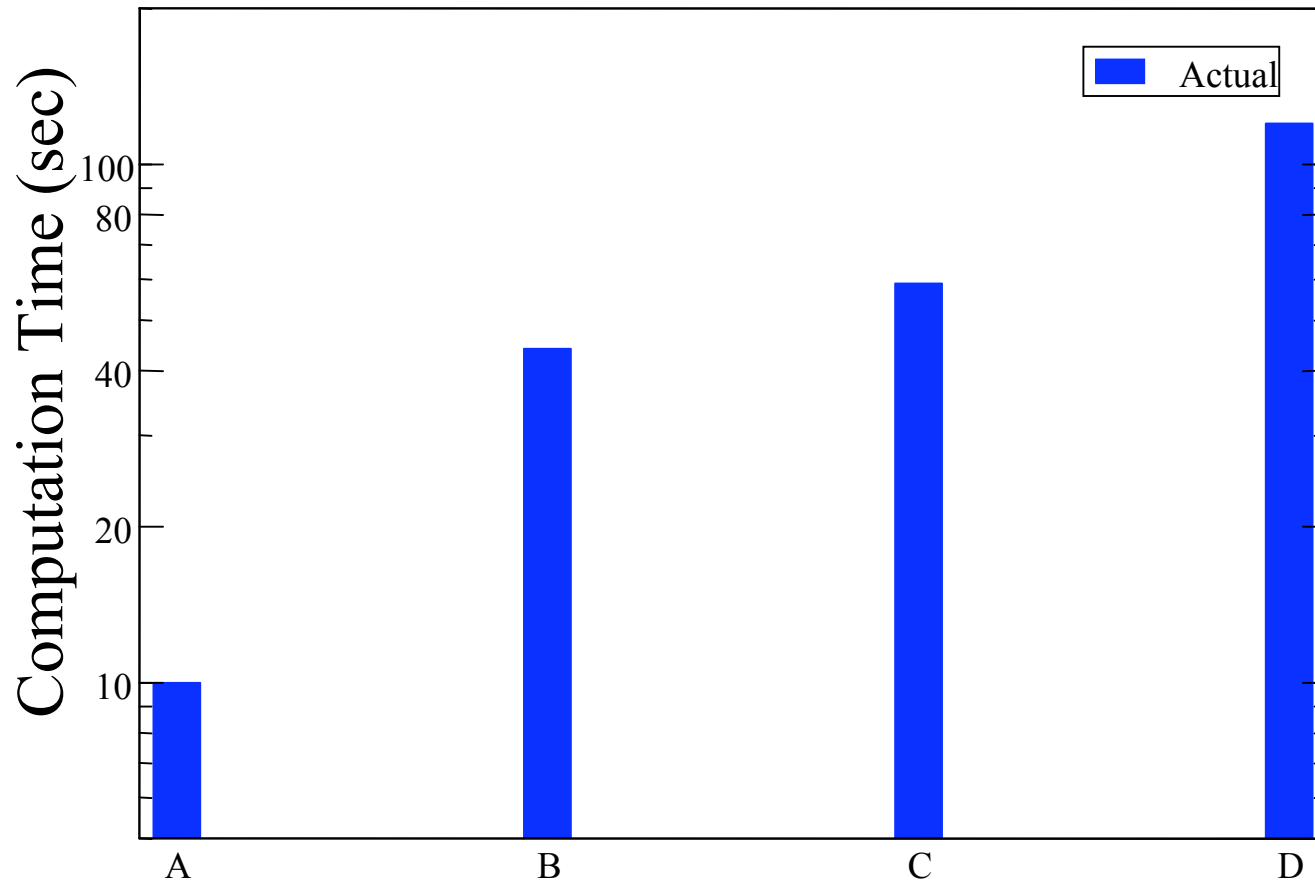
Uniform Distribution



Real Datasets - Size



Real Datasets - Time



Conclusions of Dwarf Cubes

- Analytical framework for representing coalesced cubes
- *Demonstrated that:*
 - *fully materialized*
 - *100% accurate Dwarf**require **polynomial** (w.r.t dimensionality and number of fact table tuples) time and size to compute and store*
- Designed efficient algorithm for the estimation of the coalesced cube size
- Oracle 10g now offers coalesced cubes

Future of OLAP & Data Cubing

- The big rush to data warehousing has passed and left bitter taste
 - Too costly
 - Did not achieve the promised database integration
- New applications with multi-dimensional data are needed
 - Cost with today's technology is much less
 - Data integration is not easier and requires hard brain work
- Promising data areas
 - Scientific
 - Security
 - Web