Data Warehousing
View Materialization

Esteban Zimányi

ezimanyi@ulb.ac.be

Slides by Toon Calders

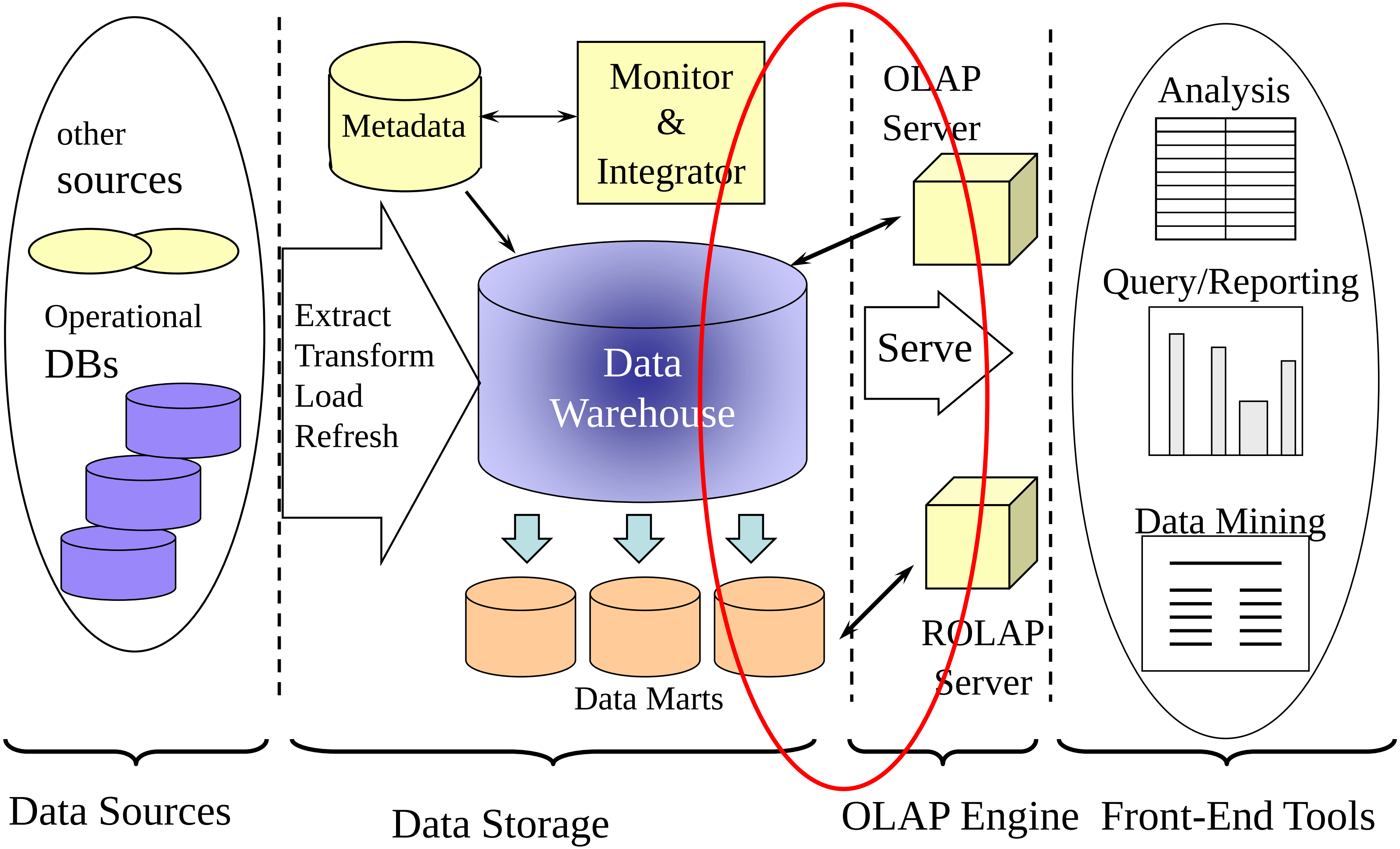
ULB

ECOLE
POLYTECHNIQUE
DE BRUXELLES

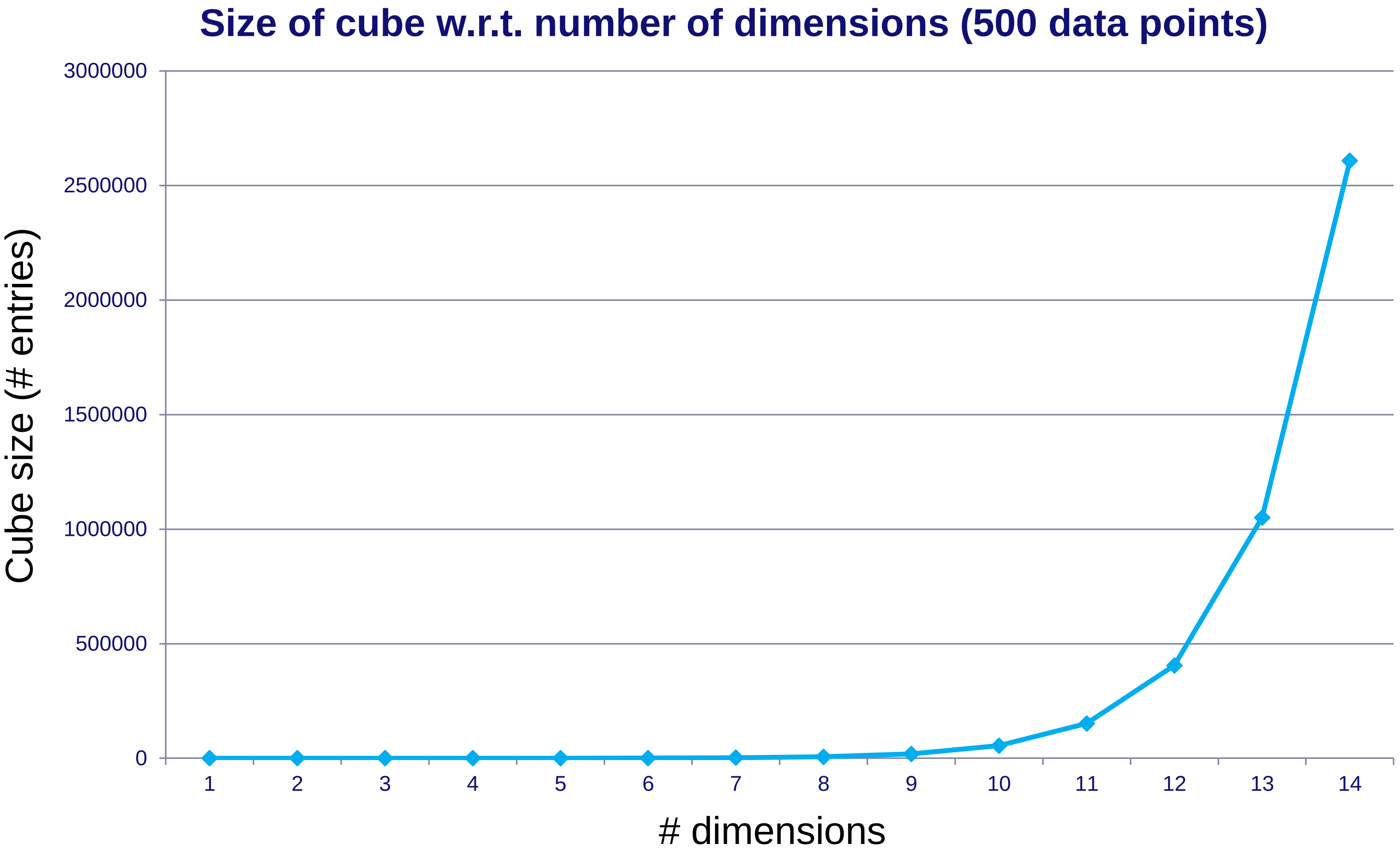
Outline

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

Previous Lecture

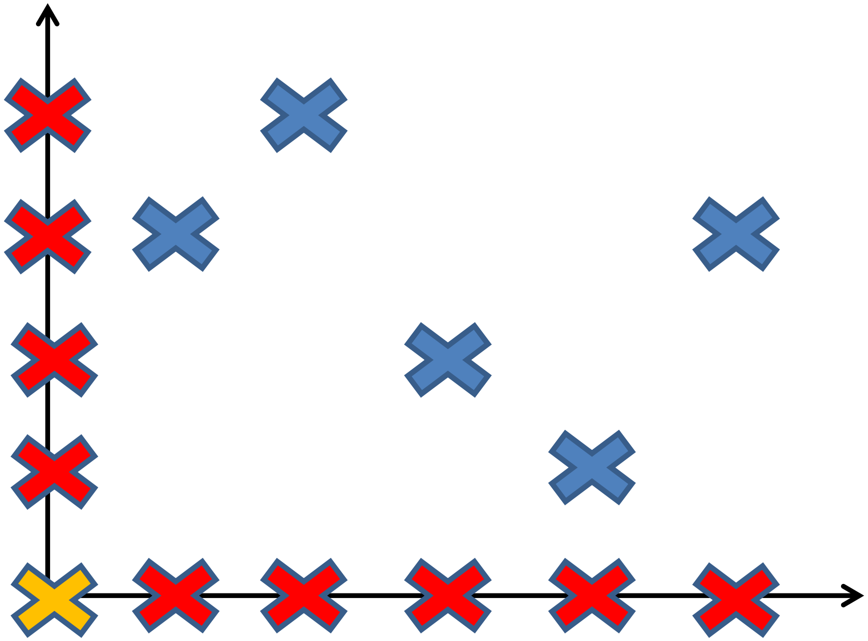


Data Explosion Problem



Data Explosion Problem

Country	Brand	Sales
FR	A	123
NL	B	456
BE	C	678
US	D	254
US	E	134



View materialization

- One of the major optimization techniques for OLAP
- Main question: Which parts to materialize to optimize performance?
  - Explosion problem: Pre-computing everything is impossible

Techniques in this slide set are based upon the following seminal work:

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

The problem: informally

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

Queries

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

(part, supplier)

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

(part)

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

(supplier)

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

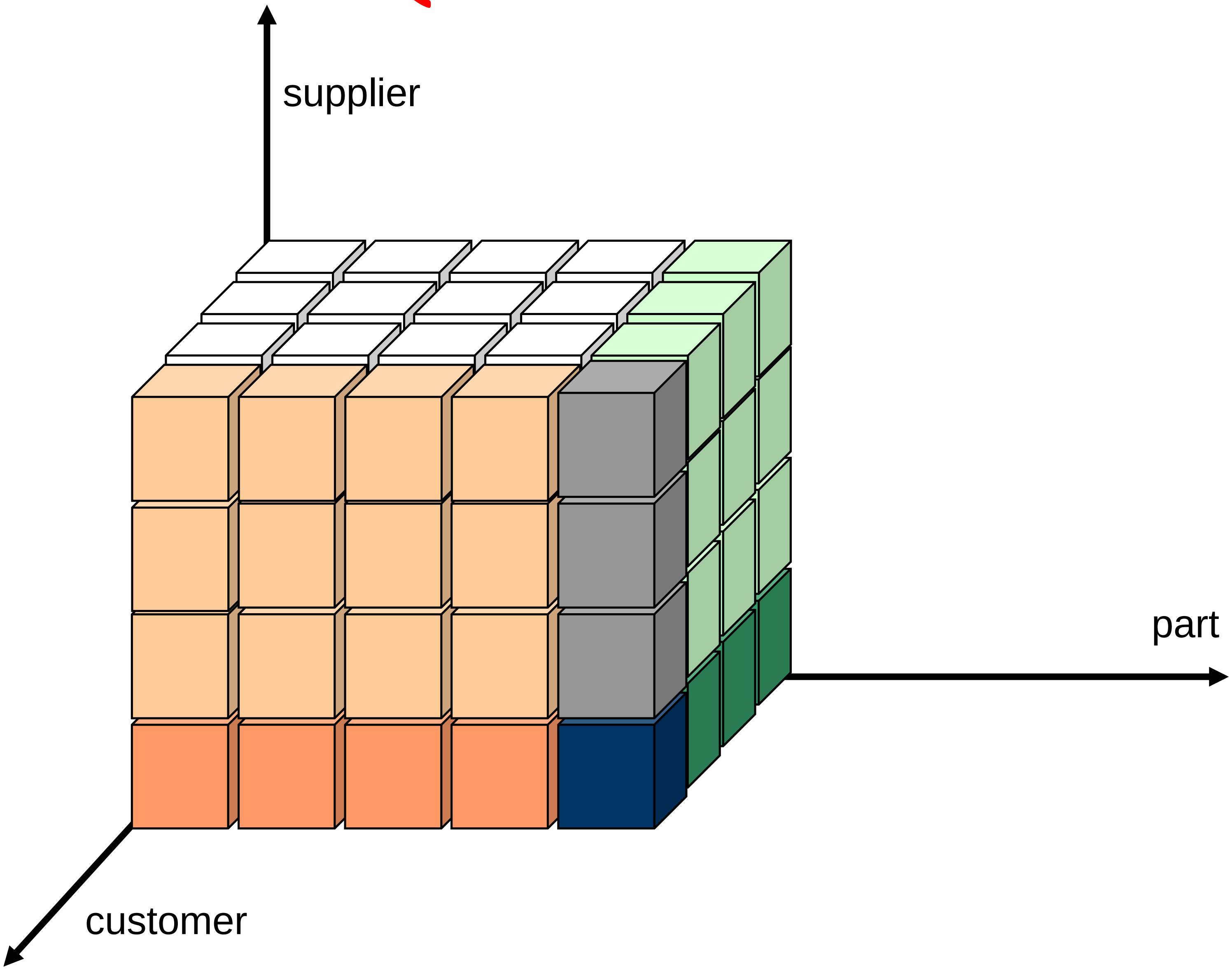
(part, customer)

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

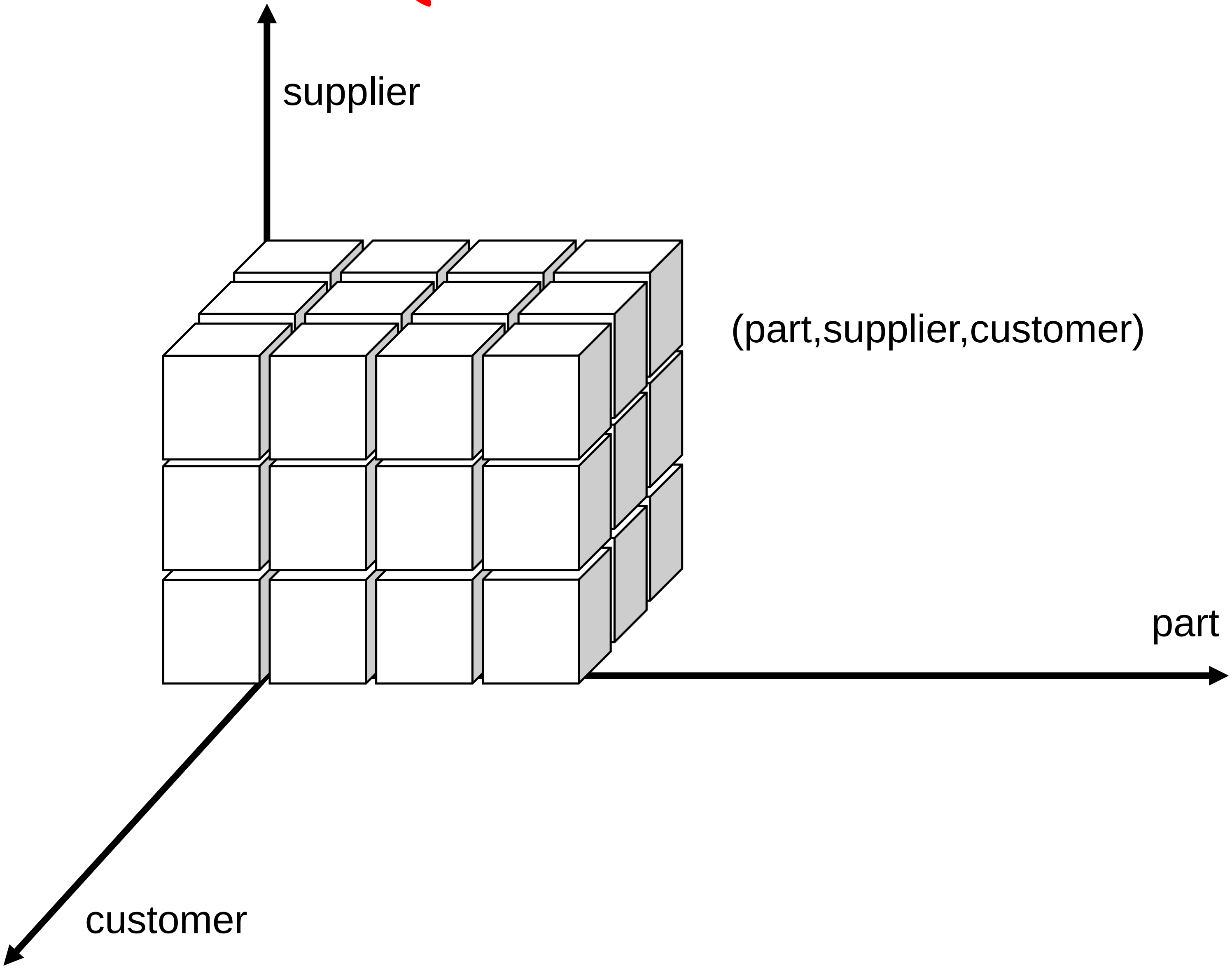
Queries

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

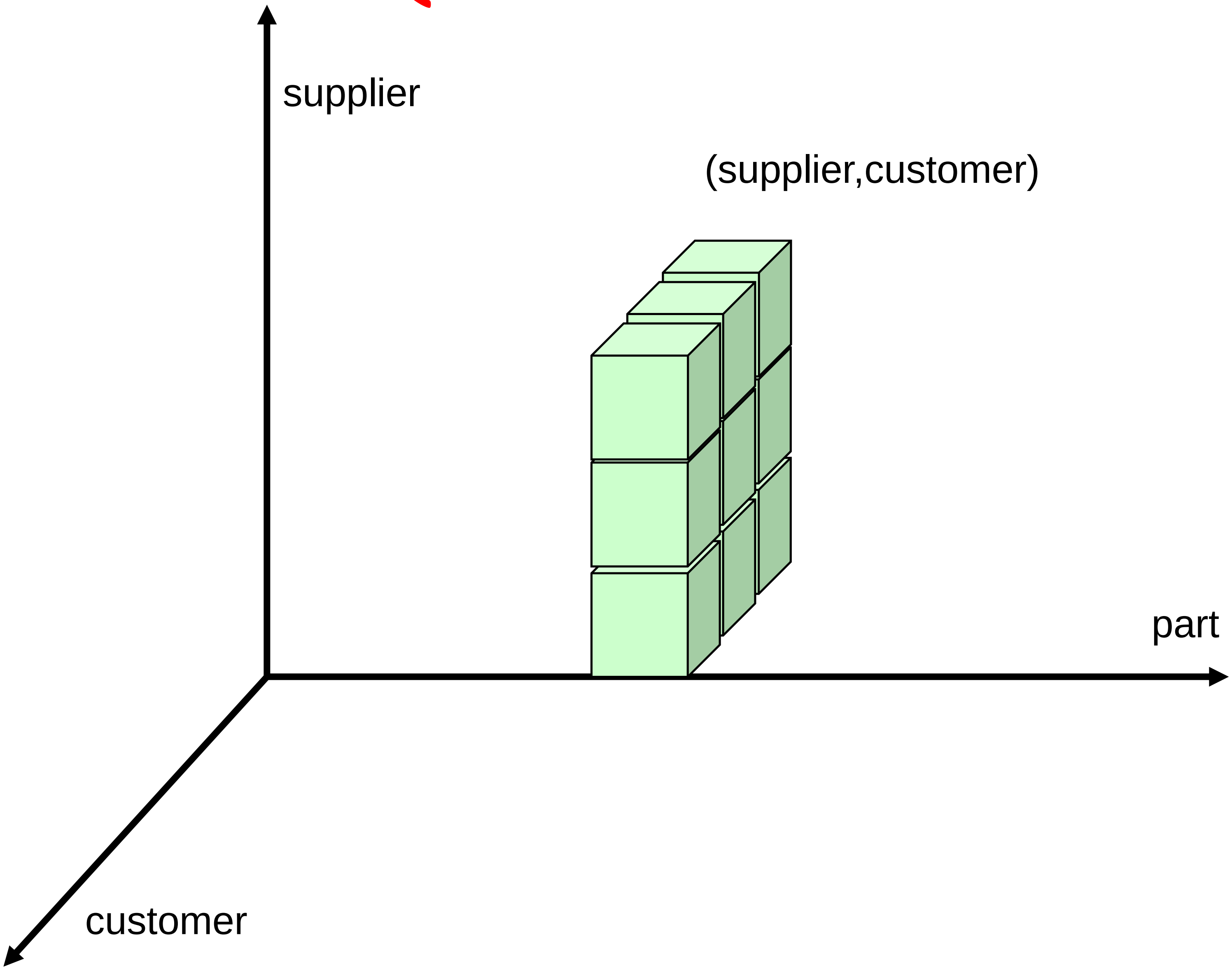
Queries



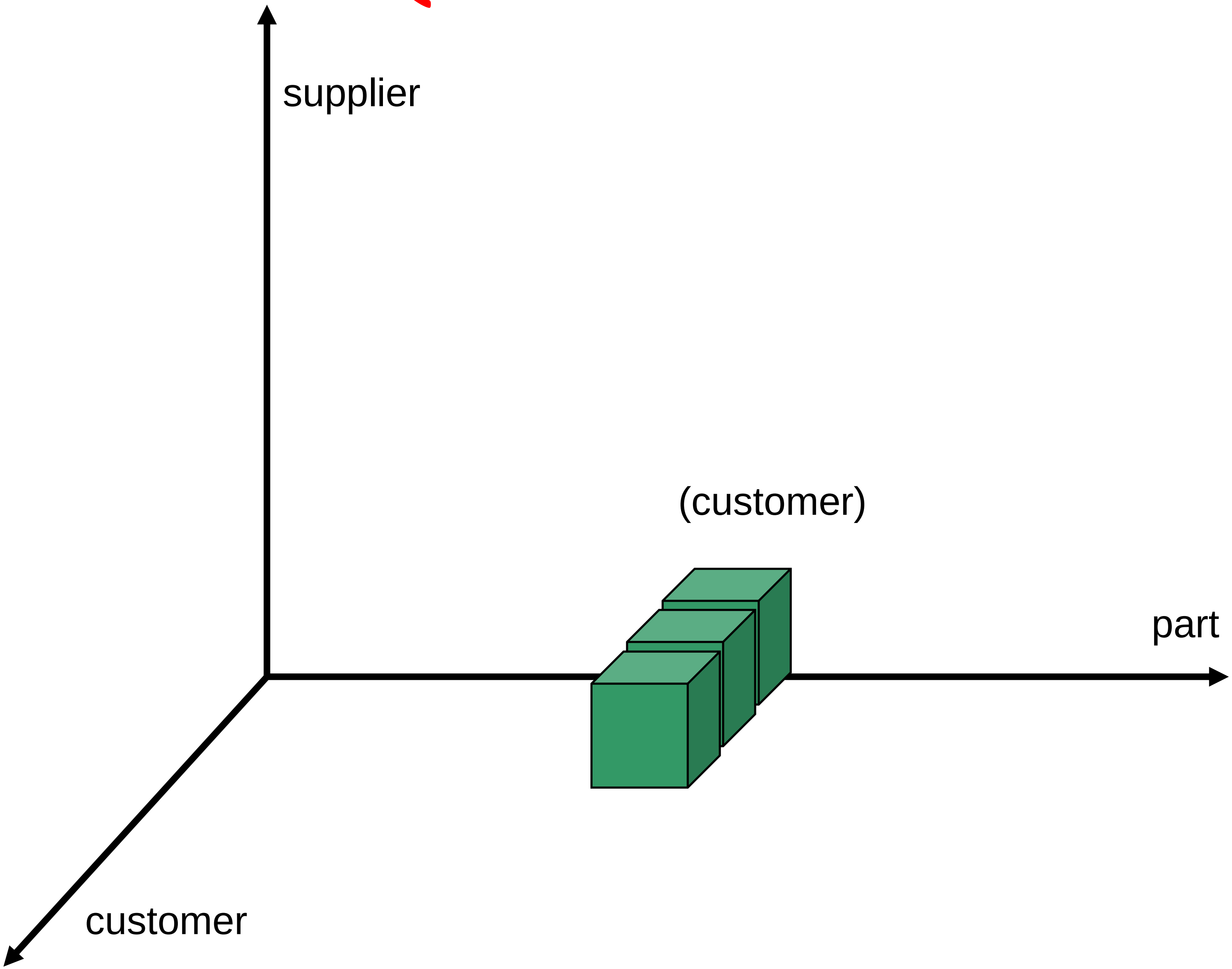
Queries



Queries



Queries



Outline

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

Materialization

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

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

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

Materialization

Example:

(part, customer)

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

| Sales |

(part)

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

| Sales |

Materialization

Example:

(part, customer)

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

materialized as PC

|PC|

(part)

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

|PC|

Caveat!

- For *additive measures-dimension* pairs and *distributive aggregation operators*
  - Measures that can be aggregated meaningful in any way

  E.g., amount of sales, number of orders
- Careful with semi-additive
  - E.g., inventory level
- For algebraic aggregation operators: store support measures

Example

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

Example: nothing materialized

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

Total cost: 48M

Example: some materialized

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

Example: some materialized

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

Total cost: 20.6M

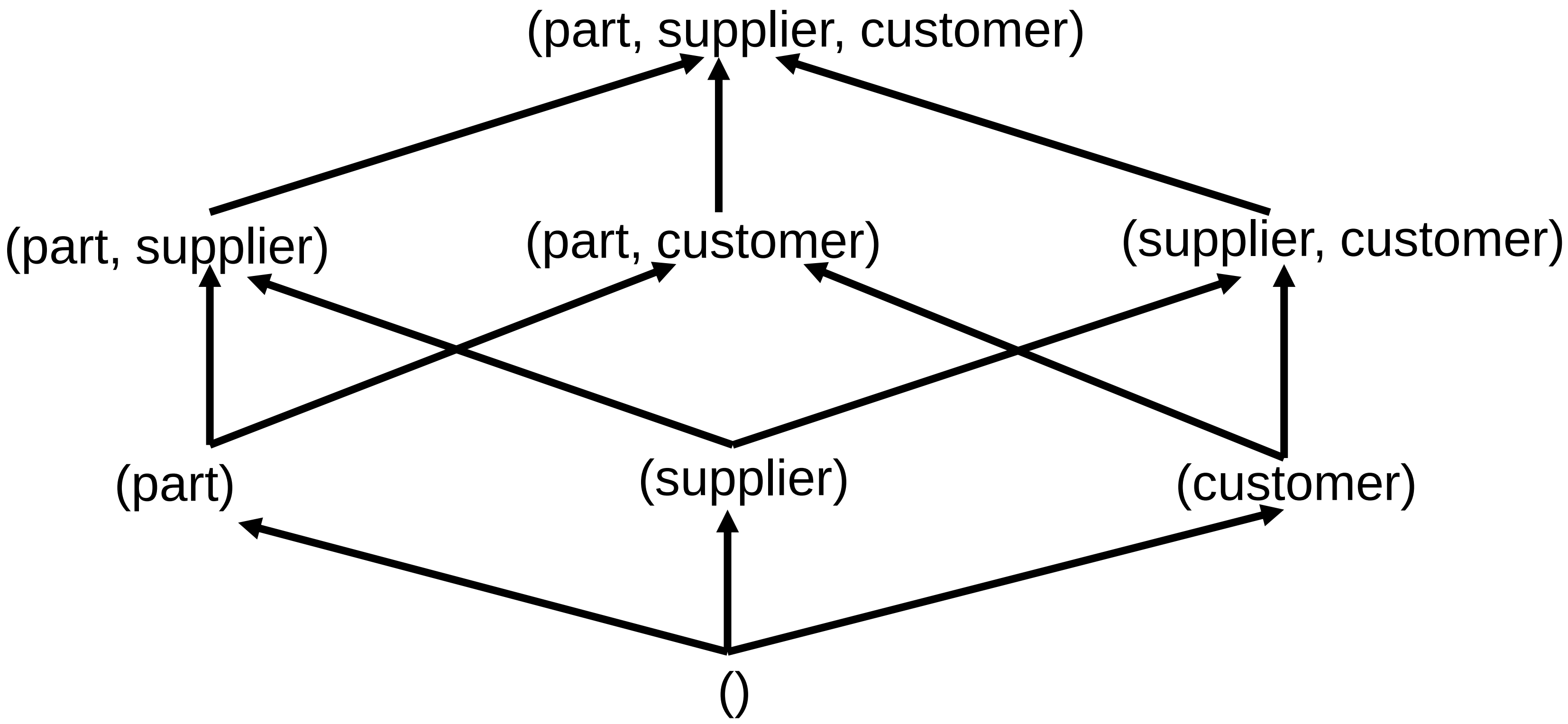
Research Questions

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

- First important step towards a solution: cost model
  - *Estimate* the benefit of materializing a view – associated with an evaluation strategy
  - Often *impossible to know* the sizes of the views without actually computing them

Partial Order on Queries

- partial order on the views: Q1 ≤ Q2 :
  - Q1 can be answered using the query results of Q2
  - *materialized view* of Q2 can be used to answer Q1



Partial order on queries

- Can be generalized to hierarchies:

  (product, country, -) $\leq$ (product, city,-)

  (-, country, -) $\leq$ (product, city, -)

  (-, -, -) $\leq$ (product, city, -)

  (-, country, year) $\leq$ (-, city, month)

- This is actually the normal specialization relation of the roll-up lattice

Cost Model

- Hence, given a set of materialized views S
  $S = \{\text{BaseRelation}, V_1,\dots,V_n\}$,

Unique top element in the lattice

- The *cost of query Q given S* is

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

- The *total cost given S* is:

$$\text{cost}(S) := \textstyle\sum_Q \text{cost}(Q|S)$$

- The *benefit of S* is:

$$B_S := \text{cost}(\{\text{BaseRelation}\}) - \text{cost}(S)$$

Outline

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

Greedy Solution for One Variant

- Class of *view selection problems*:

minimize cost(S) s.t.
$|S| \leq k$ (bound # views)
$\sum_{V \in S} |V| \leq M$ (space bound)
…

- We will treat one particular case:
  - Given: for every view its size, k
    Find: *k* views giving the highest

This problem is NP-complete

  - greedy algorithm with good guarantees

Greedy algorithm

- Input:
  - Size for every view
  - Relation $\leq$ between views
  - Constant k

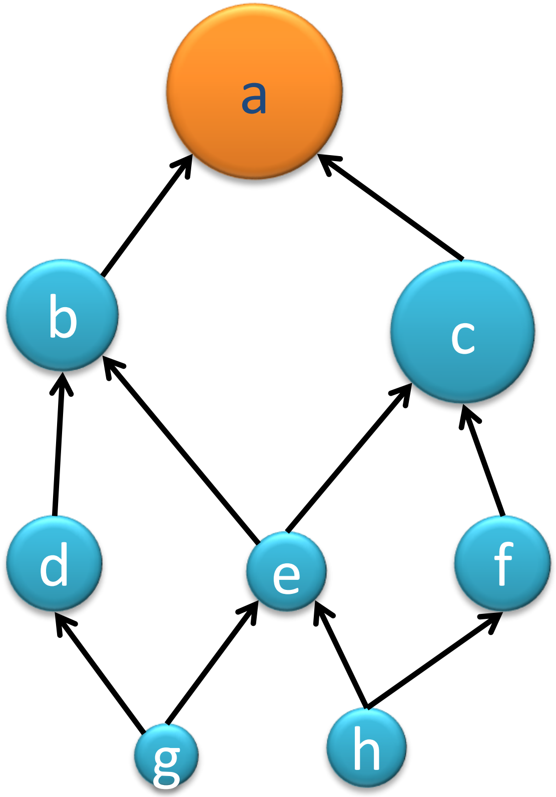
```
Greedy:
    $S = \{ \text{top level view} \}$
    Repeat k times:
        Select W that minimizes $cost(S \cup \{W\})$
        $S := S \cup \{W\}$
    Return S
```

“Benefit” of W given S:

$$
\begin{aligned}
B(W|S) &= Cost(S) - Cost(S \cup \{W\}) \\
&= \sum_Q cost(Q|S) - \sum_Q cost(Q|S \cup \{W\}) \\
&= \sum_Q [cost(Q|S) - cost(Q|S \cup \{W\})] \\
&= \sum_Q B_Q(W|S)
\end{aligned}
$$

Example

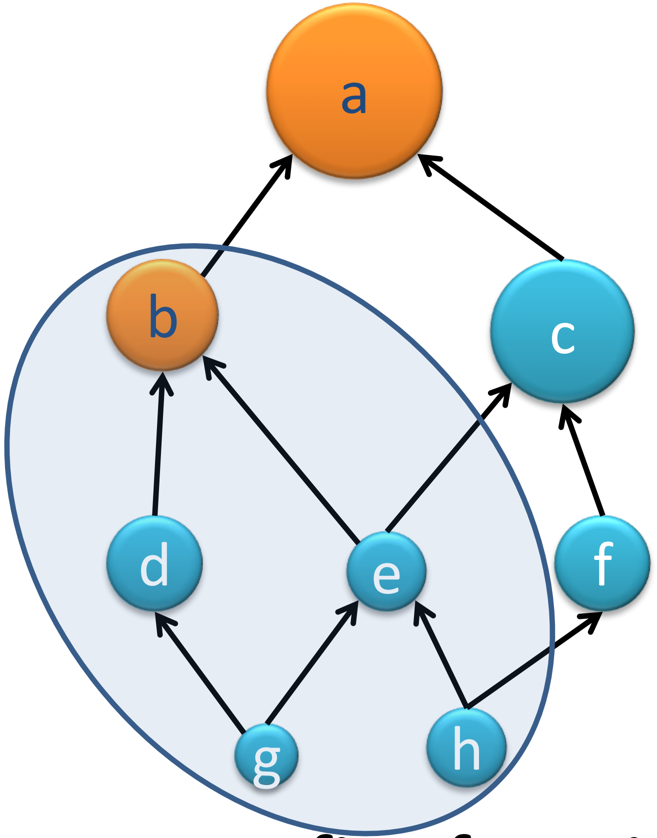
- Only base table a is materialized
  - Total: 8 x 100 = 800



Query	Size	Uses?	Cost
a	100	a	100
b	50	a	100
c	75	a	100
d	20	a	100
e	30	a	100
f	40	a	100
g	1	a	100
h	10	a	100

Example

- Base table a and table b are materialized
  - Total: 3 x 100 + 5 x 50 = 550



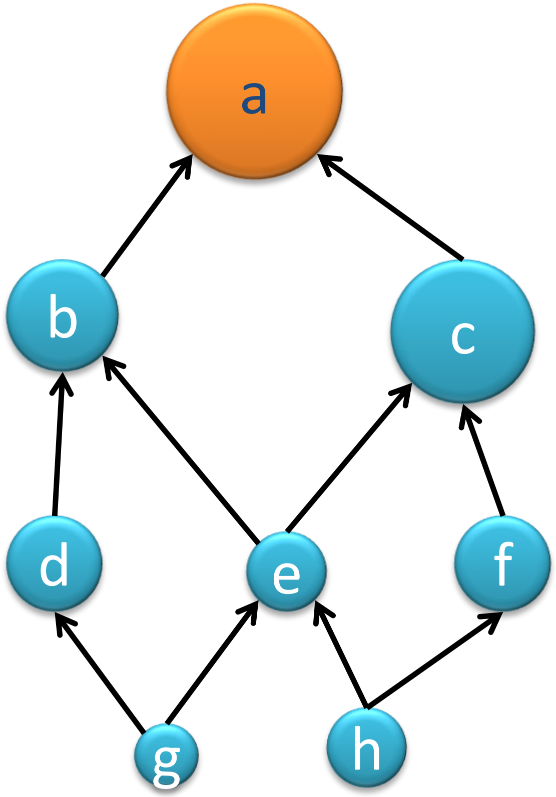
Query	Size	Uses?	Cost	Benefit
a	100	a	100	-
b	50	**b**	**50**	**50**
c	75	a	100	-
d	20	**b**	**50**	**50**
e	30	**b**	**50**	**50**
f	40	a	100	-
g	1	**b**	**50**	**50**
h	10	**b**	**50**	**50**

- Benefit of storing b = 250 = 5 x (100-50)

Example

a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

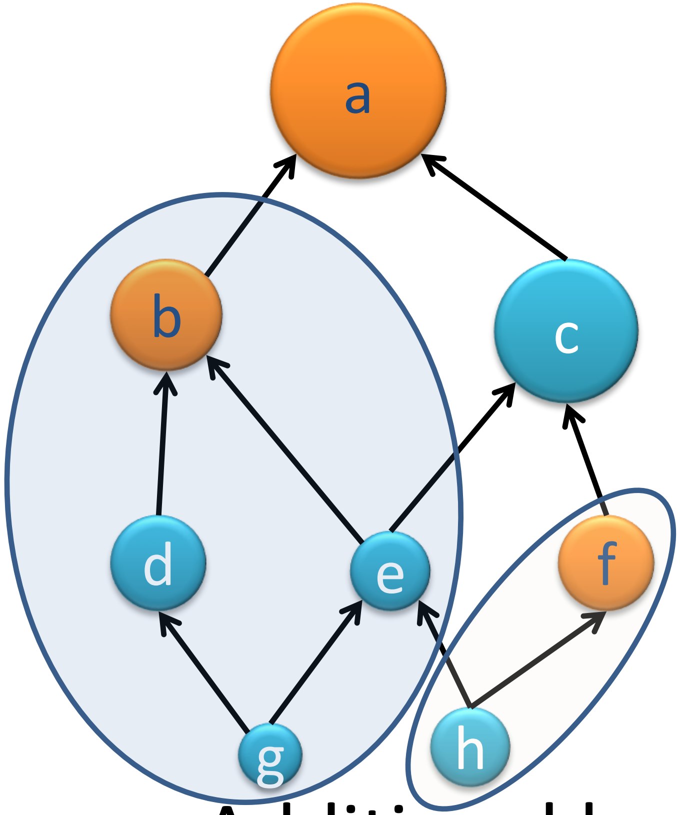
- Benefit for materializing the other tables:



query	Materialized view						
	b	c	d	e	f	g	
a	-	-	-	-	-	-	-
b	**50**	-	-	-	-	-	-
c	-	25	-	-	-	-	-
d	**50**	-	**80**	-	-	-	-
e	**50**	25	-	**70**	-	-	-
f	-	25	-	-	**60**	-	-
g	**50**	25	**80**	**70**	-	**99**	-
h	**50**	25	-	**70**	**60**	-	**90**
Total	**250**	**125**	**160**	**210**	**120**	**99**	**90**

Example

- Base table a, table b, and f are materialized
  - Total: 2 x 100 + 4 x 50 + 2 x 40 = 480



Query	Size	Uses?	Cost	Benefit
a	100	a	100	-
b	50	b	50	-
c	75	a	100	-
d	20	b	50	-
e	30	b	50	-
f	40	**f**	**40**	**60**
g	1	b	50	-
h	10	**f**	**40**	**10**

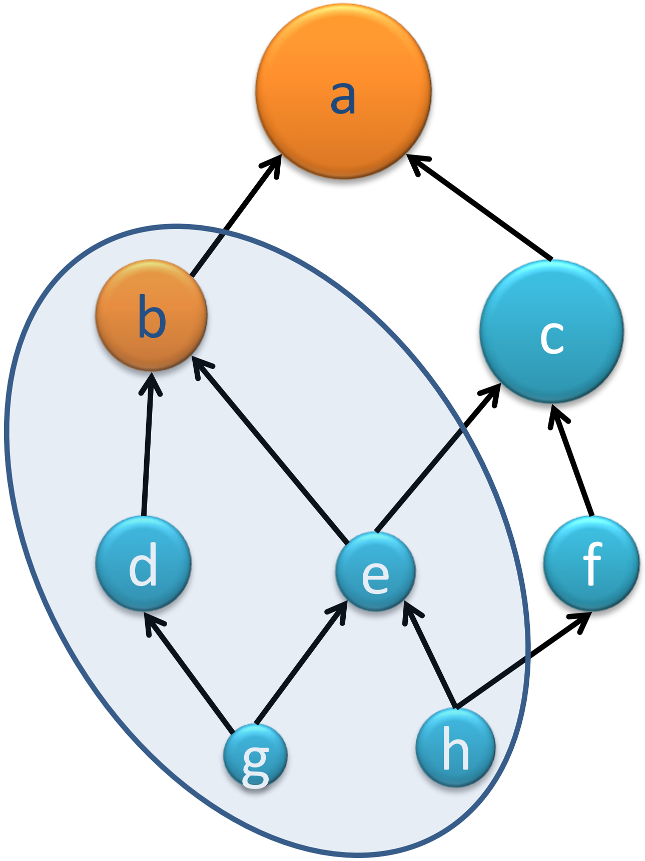
- Additional benefit of materializing f

  = 70 = 1 x (100-40) + 1 x (50-40)

Example

- Benefit for materializing the other tables:

a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

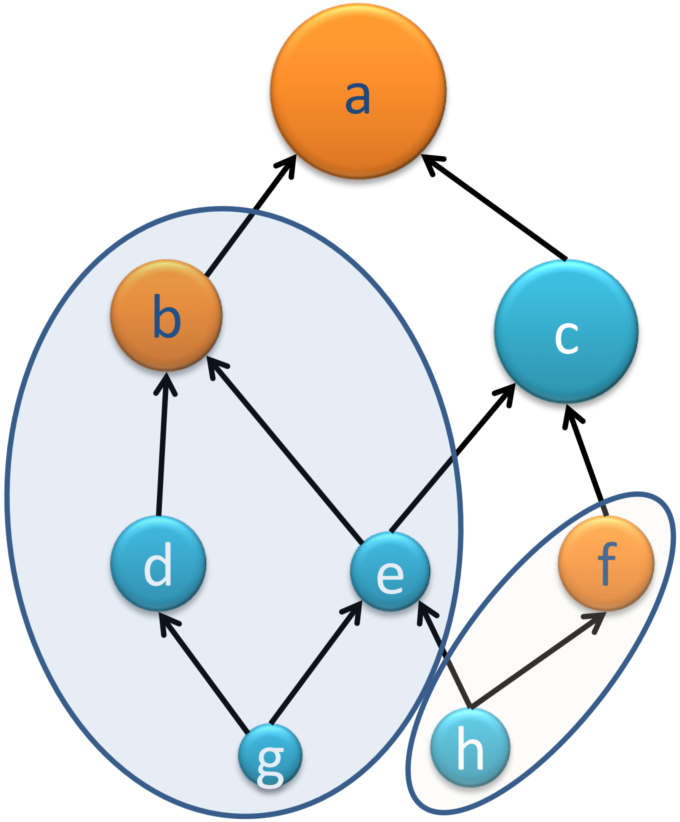


query	Materialized view					
	c	d	e	f	g	h
a	-	-	-	-	-	-
b	-	-	-	-	-	-
c	**25**	-	-	-	-	-
d	-	**30**	-	-	-	-
e	-	-	**20**	-	-	-
f	**25**	-	-	**60**	-	-
g	-	**30**	**20**	-	**49**	-
h	-	-	**20**	**10**	-	**40**
Total	**50**	**60**	**60**	**70**	**49**	**40**

a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

Example

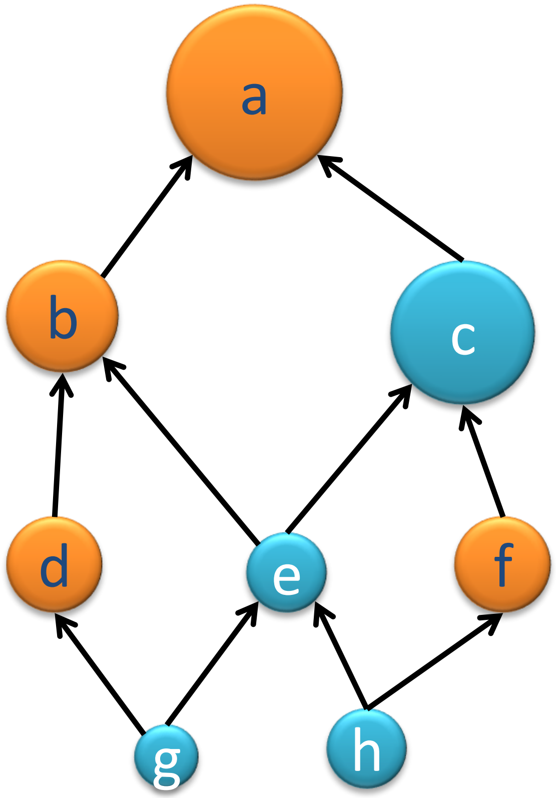
- Benefit for materializing a third table:



query	Materialized view				
	c	d	e	g	h
a	-	-	-	-	-
b	-	-	-	-	-
c	25	-	-	-	-
d	-	**30**	-	-	-
e	-	-	**20**	-	-
f	-	-	-	-	-
g	-	**30**	**20**	**49**	-
h	-	-	**10**	-	**30**
Total	**25**	**60**	**50**	**49**	**30**

Example

- Choice of 3 views according to the Greedy algorithm:
  - Benefit = 250 + 70 + 60 = 380
  - Cost = 420



Outline

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

Some theoretical results

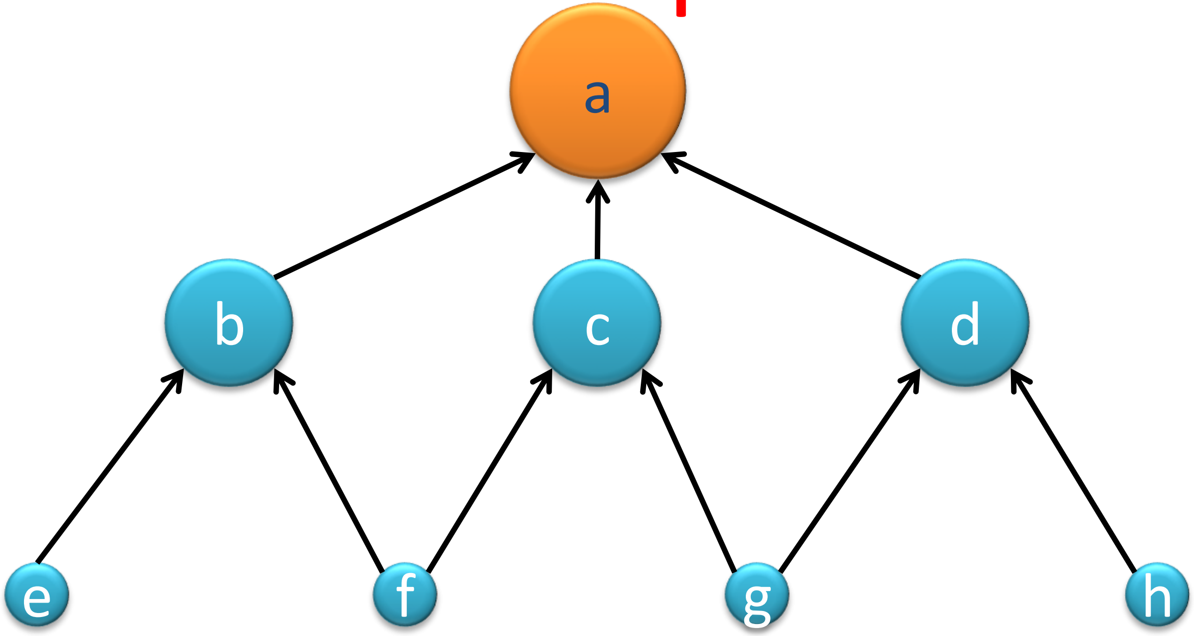
- It can be proved that we can get at least (e – 1) / e (which is about 63%) of the benefit of the optimal algorithm

- When selecting k views, bound is:

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

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

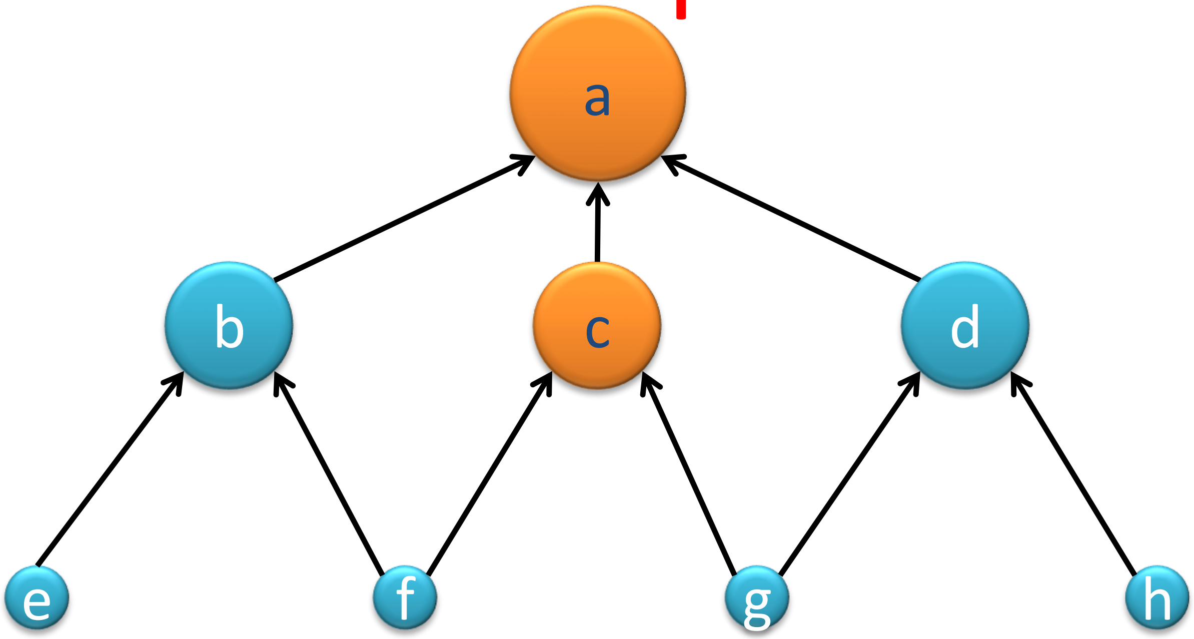
Example



a	200
b	100
c	99
d	100
e	50
f	50
g	50
h	50

Node	Benefit
b, d	3 x (200-100) = 300
c	3x(200-99) = 303
e, f, g, h	1x(200-50) = 150

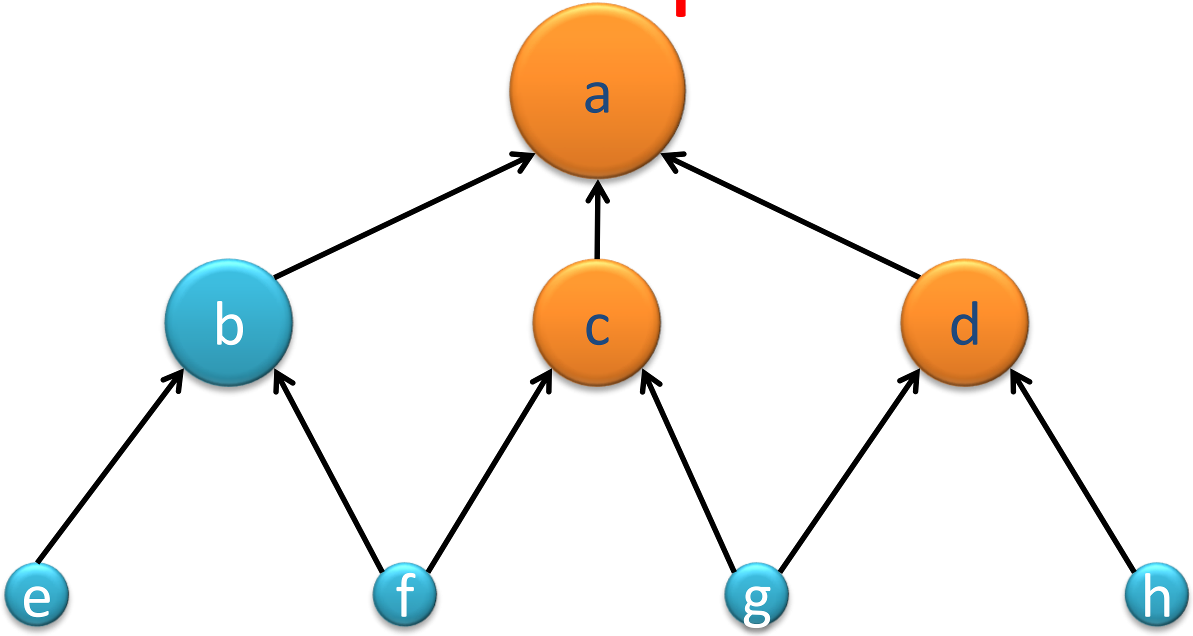
Example



a	200
b	100
c	99
d	100
e	50
f	50
g	50
h	50

Node	Benefit
b, d	2 x (200-100) = 200
e, h	1x(200-50) = 150
f, g	1x(99-50) = 49

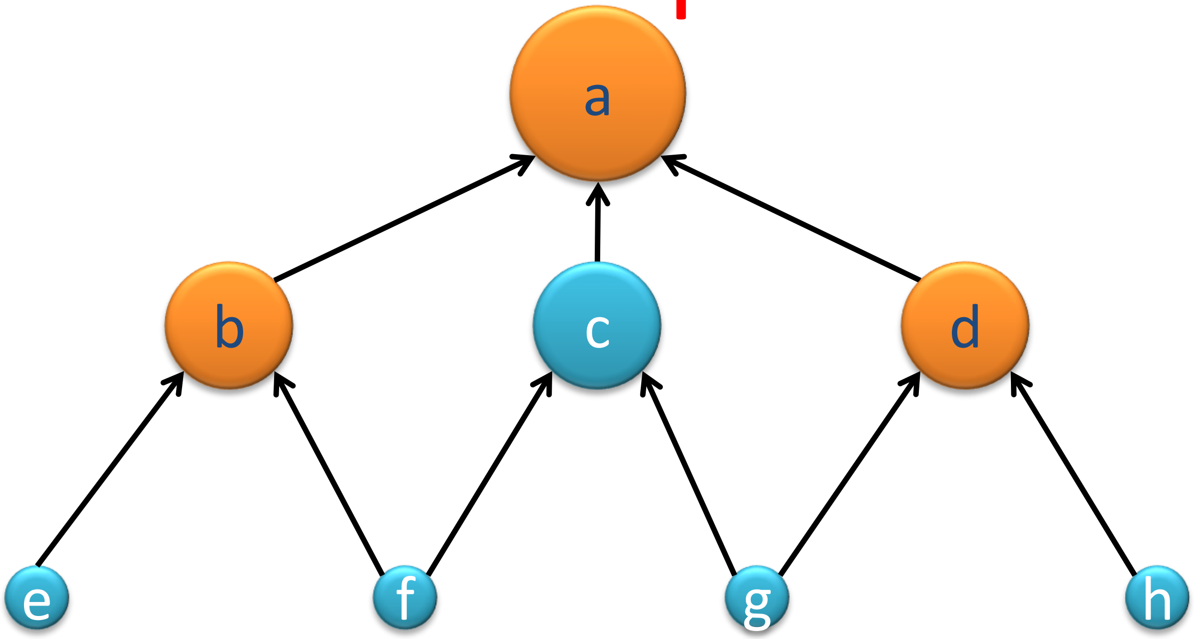
Example



a	200
b	100
c	99
d	100
e	50
f	50
g	50
h	50

- Hence B_{greedy} = 503

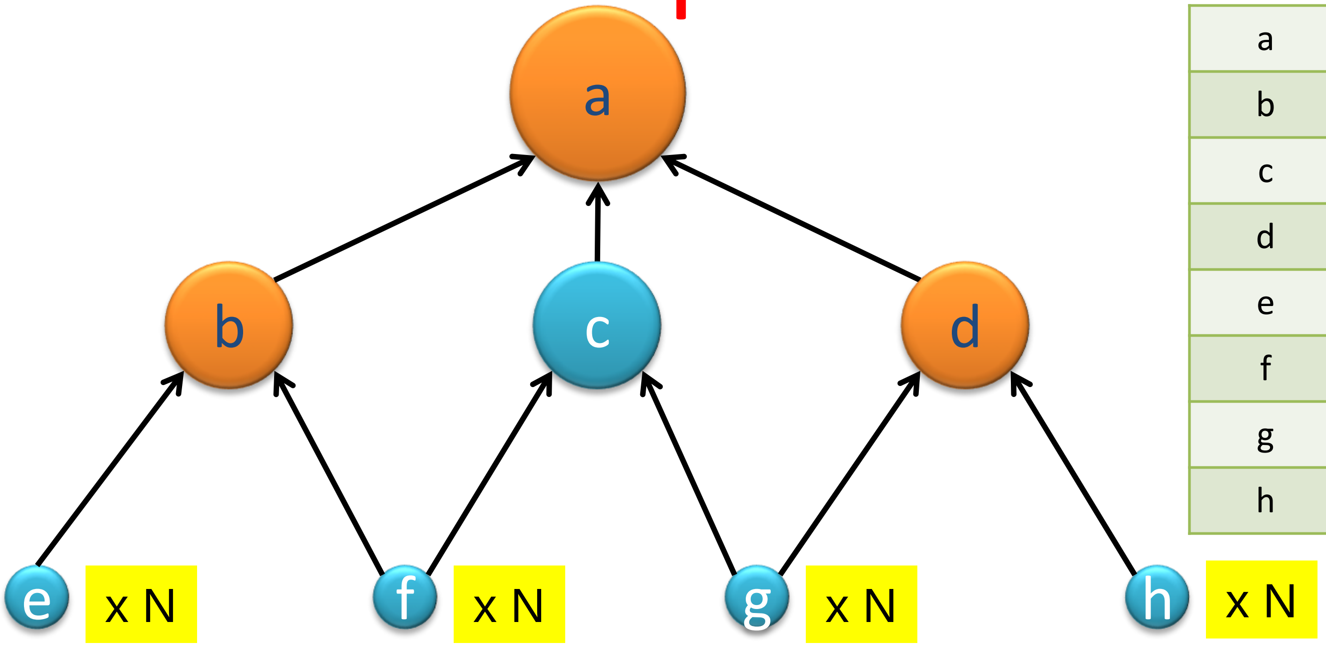
Example



a	200
b	100
c	99
d	100
e	50
f	50
g	50
h	50

- Hence B_{greedy} = 503
- B_{opt} = 6 x (200-100) = 600
- $B_{greedy} / B_{opt} = 503/600 \geq 75\%$

Example



a	200
b	100
c	100-ε
d	100
e	50
f	50
g	50
h	50

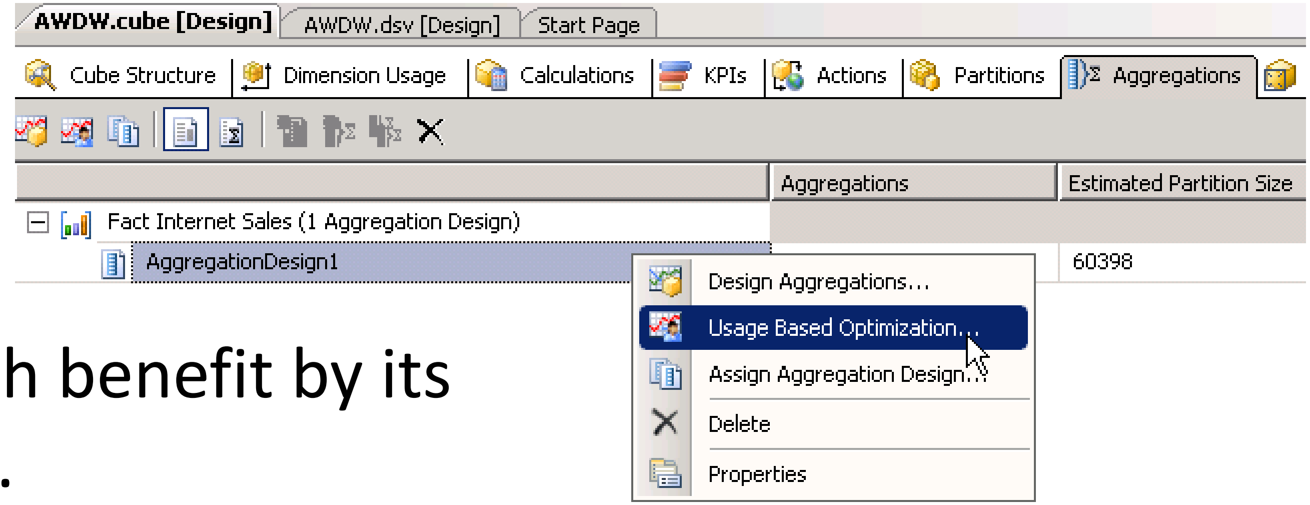
- Hence $B_{greedy} = (2N + 1) \times (100+\varepsilon) + (N+1) \times 100$ → $300N + 200$
- $B_{opt} = (4N+2) \times 100$ → $400N + 200$
- B_{greedy} / B_{opt} → $(3N+2)/(4N+2)$ → 75%

Outline

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

Extensions (1)

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

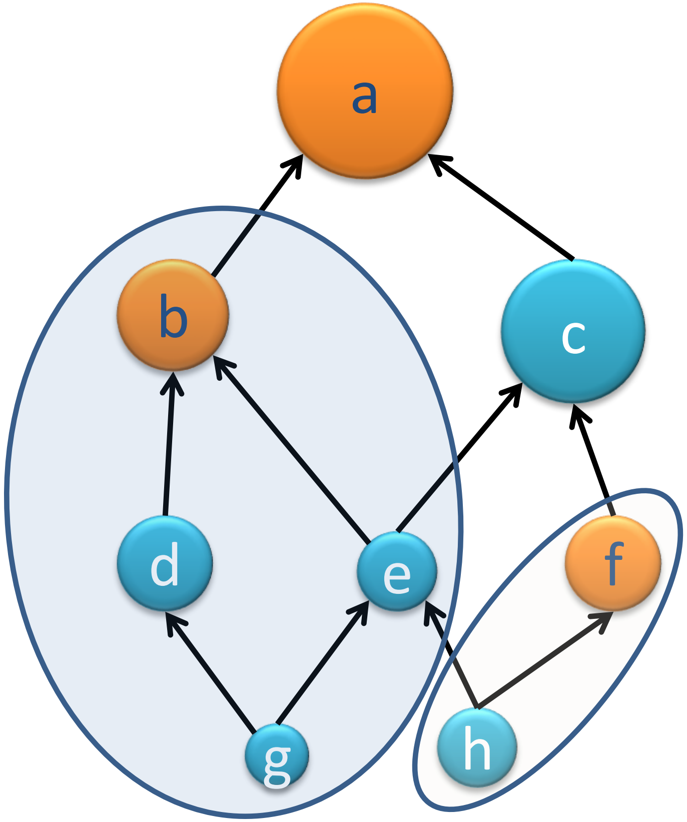


a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

Example

- Suppose following distribution:

a: 0%, b: 10%, c:0%, d:10%, e:5%, f:20%, g: 40% , h:15%



Query	Uses?	Cost	Exp. Cost
a	a	100	0
b	b	50	5
c	a	100	0
d	b	50	5
e	b	50	2.5
f	f	40	8
g	b	50	20
h	f	40	6
Total			**46.5**

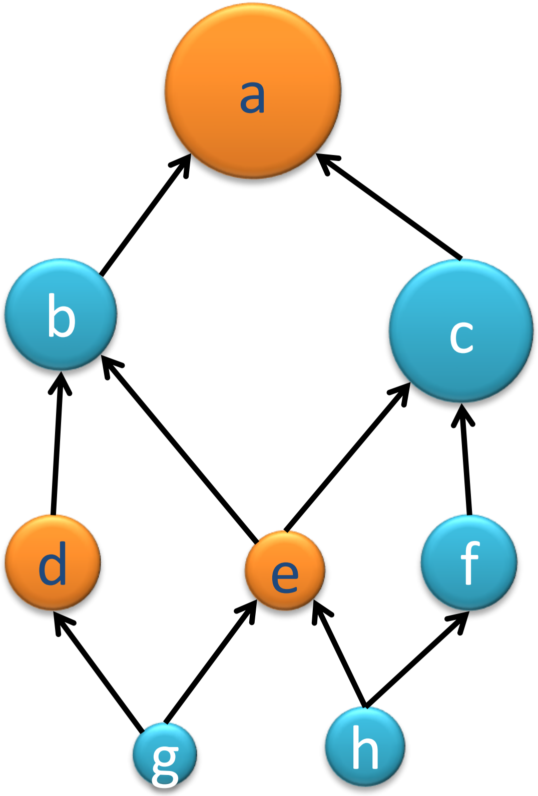
- Expected cost given S = 46.5

a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

Example

- Solution found by greedy:

a: 0%, b: 10%, c:0%, d:10%, e:5%, f:20%, g: 40% , h:15%



Query	Uses?	Cost	Exp. Cost
a	a	100	0
b	a	100	10
c	a	100	0
d	d	20	2
e	e	30	1.5
f	a	100	20
g	d	20	8
h	e	30	4.5
Total			**46**

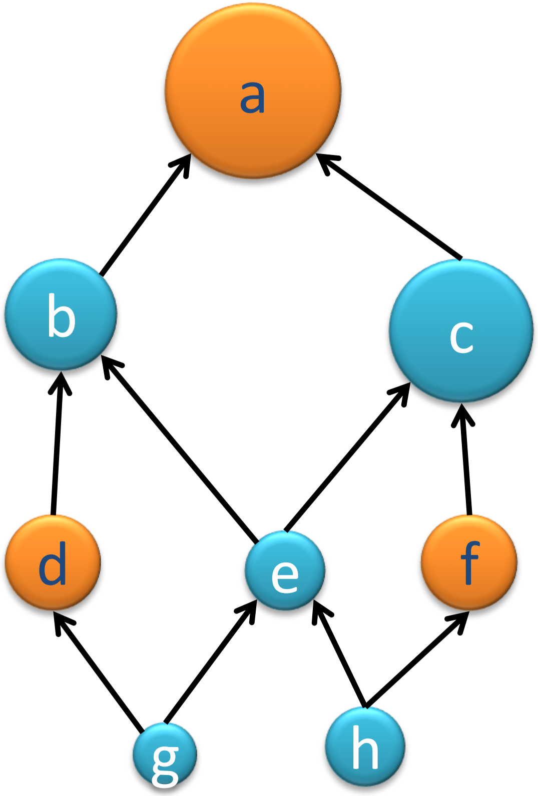
- Expected cost given S = 46

a	100
b	50
c	75
d	20
e	30
f	40
g	1
h	10

Example

- Optimal Solution:

  a: 0%, b: 10%, c:0%, d:10%, e:5%, f:20%, g: 40% , h:15%



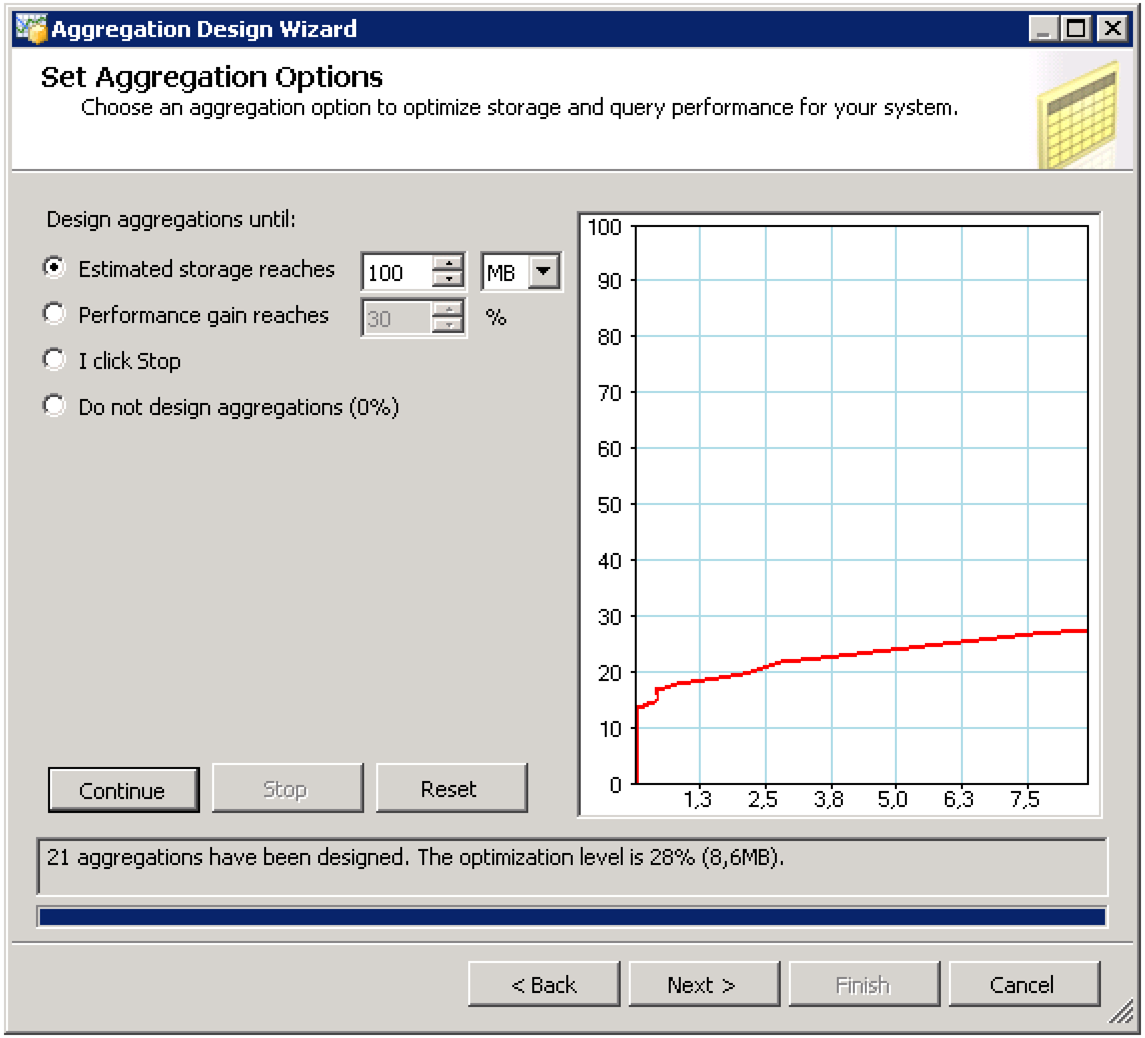
Query	Uses?	Cost	Exp. Cost
a	a	100	0
b	a	100	10
c	a	100	0
d	d	20	2
e	a	100	5
f	f	40	8
g	d	20	8
h	f	40	6
Total			**39**

- Expected cost given S = 39

Extensions (2)

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

View Materialization



Aggregation Design Wizard

Set Aggregation Options

Choose an aggregation option to optimize storage and query performance for your system.

Design aggregations until:

● Estimated storage reaches 100 MB

○ Performance gain reaches 30 %

○ I click Stop

○ Do not design aggregations (0%)

21 aggregations have been designed. The optimization level is 28% (8,6MB).

Conclusion: View Materialization

- Materialization of views is an essential query optimization strategy for decision-support applications.
  - Finding optimal solution is NP-hard.
- Introduction of greedy algorithm
  - There exists cases which greedy algorithm fails to produce optimal solution
  - Yet, greedy algorithm has a good guarantee
- Expansion of greedy algorithm.