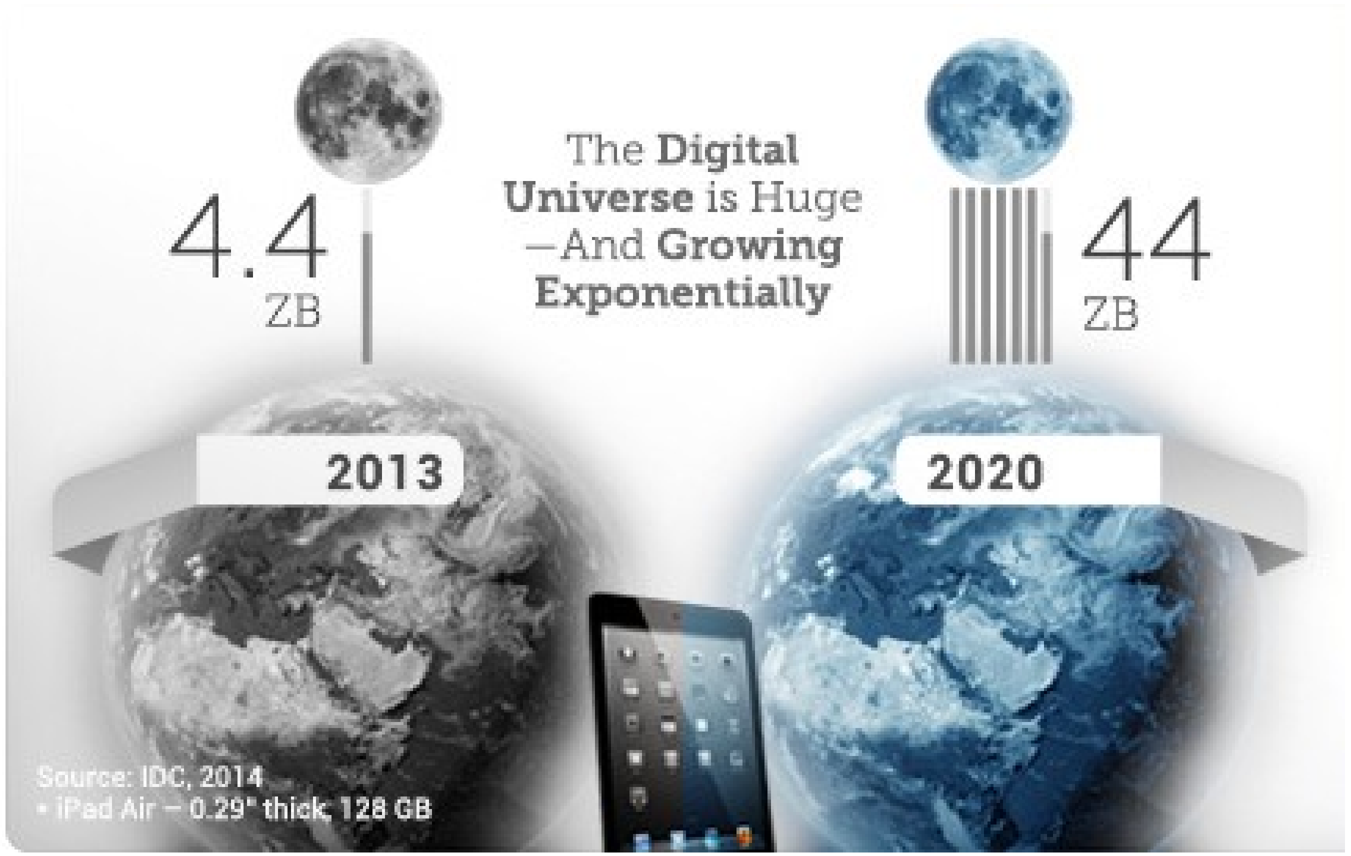
Skipping-oriented Partitioning for Columnar Layouts

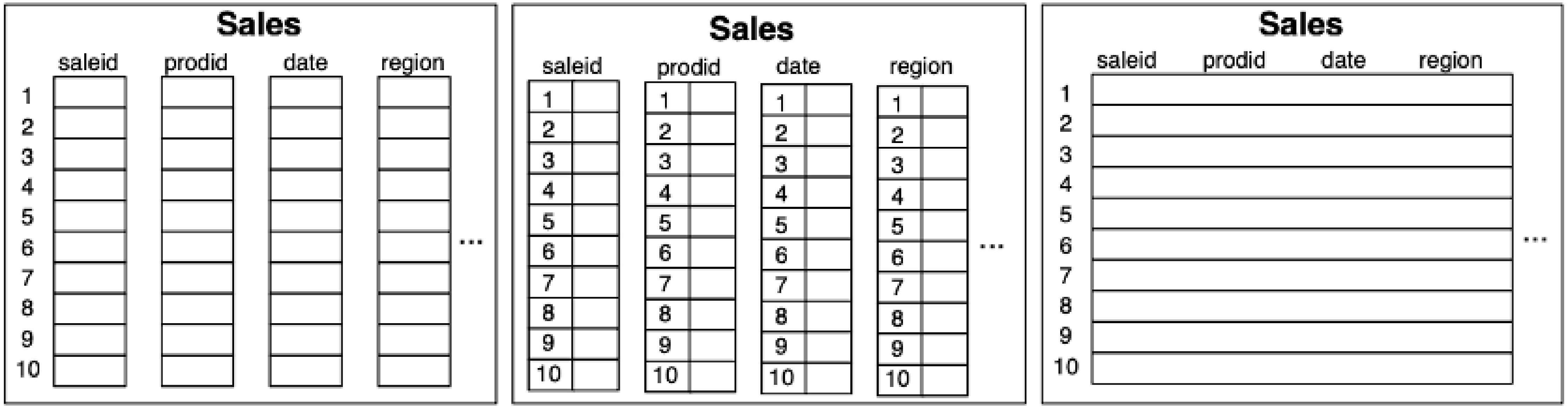
Agenda

- Motivation
- Background Knowledge
- GSOP
- Experiments
- Conclusion

Data Volumes


The Digital Universe is Huge —And Growing Exponentially
4.4 ZB
44 ZB
2013
2020
Source: IDC, 2014
• iPad Air — 0.29" thick, 128 GB

Columnar Layouts



(a) Column Store with Virtual Ids (b) Column Store with Explicit Ids (c) Row Store

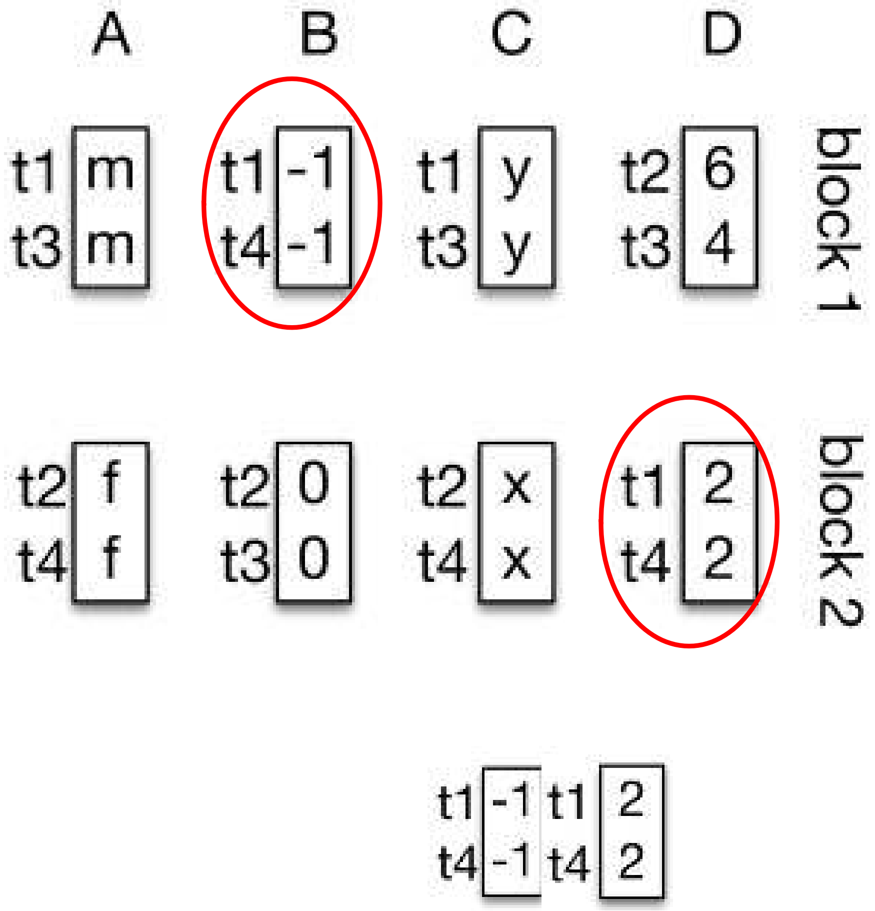
- Pros: no need to access unnecessary data, good for OLAP workloads
- Cons:
  - More operations required to complete a data update (bad for OLTP)
  - Tuple reconstruction cost (CPU)

D.Abadi et al. The design and implementation of modern column-oriented database systems. Foundations and Trends in Databases,

Columnar Layouts (tuple reconstruction)

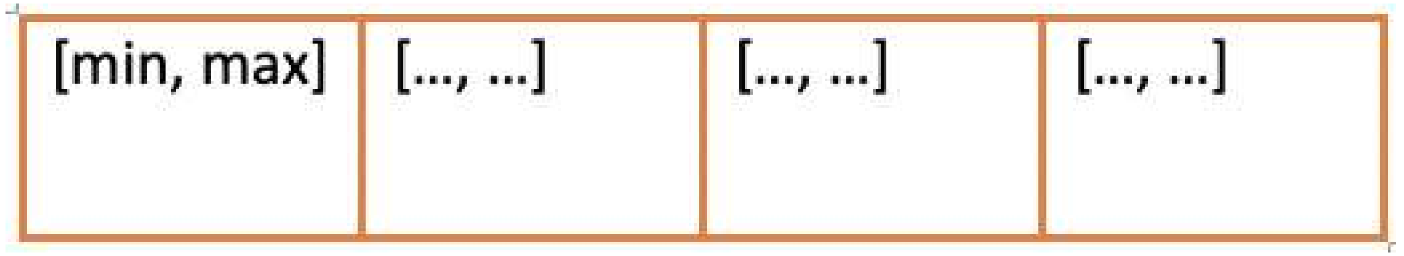
- 4 data tuples
- 4 attributes stored separately
- Divided to 2 blocks

- SELECT B, D FROM T WHERE B<0 and D=2



Data Skipping

- Data is organized into blocks

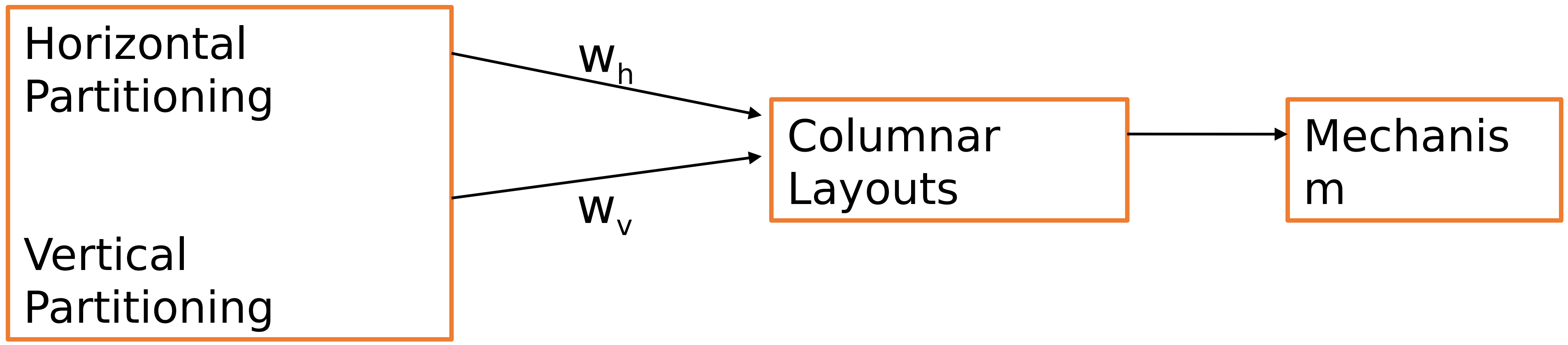


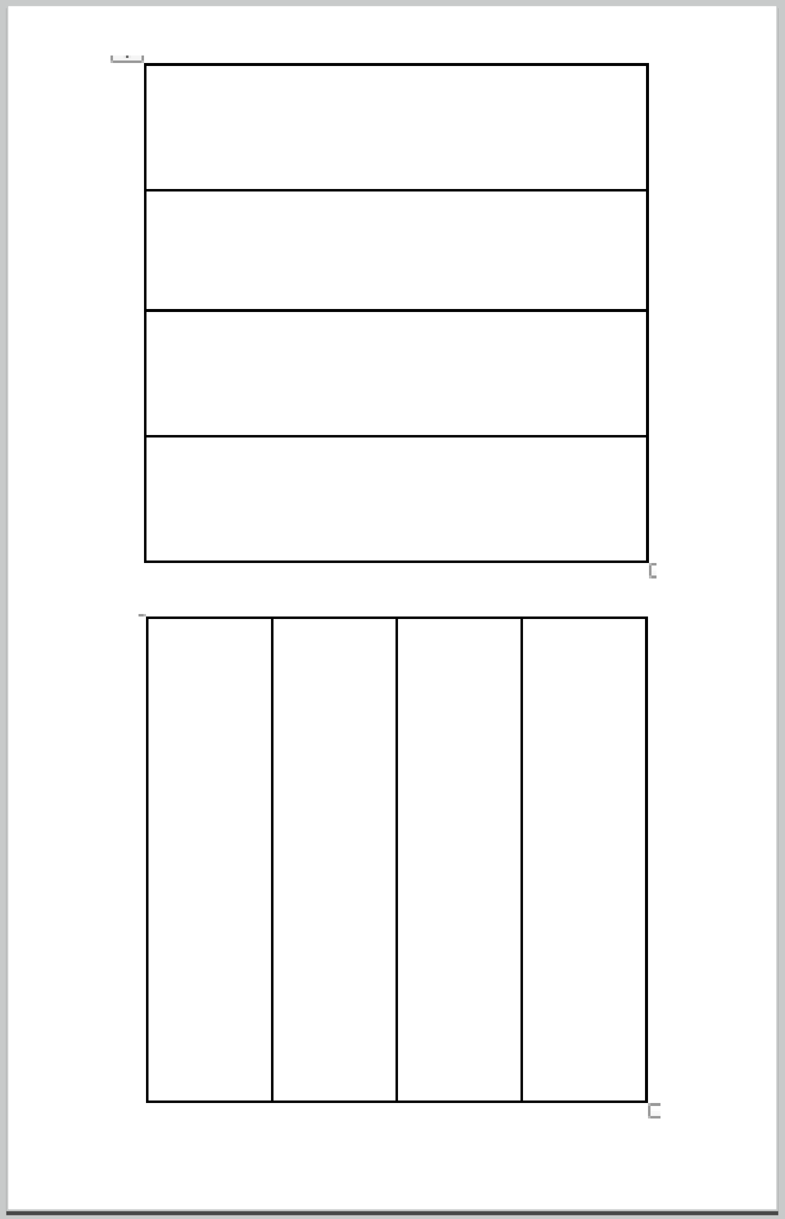
skipped. qualified skipped skipped

- Given a query q, evaluate its filter predicates against the metadata
- Save I/O, CPU work

Target Question

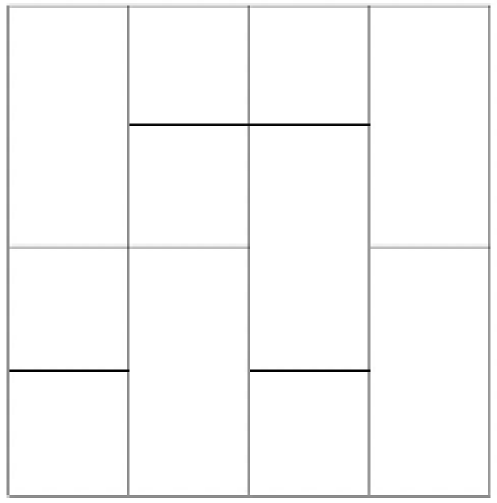
- How to design a data skipping mechanism that exploits the properties of columnar layouts?

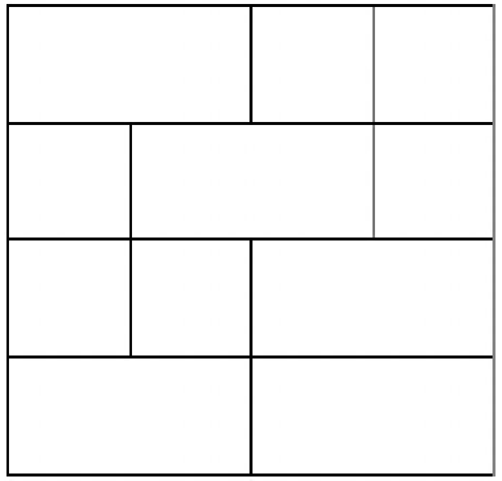




- When $w_h = 1$ and $w_v = 0$, **pure horizontal partitioning**
  - No need to retrieve unnecessary data, a property of columnar layouts
  - Feature conflicts

- When $w_h = 0$ and $w_v = 1$, **pure vertical partitioning**
  - Tuple reconstruction

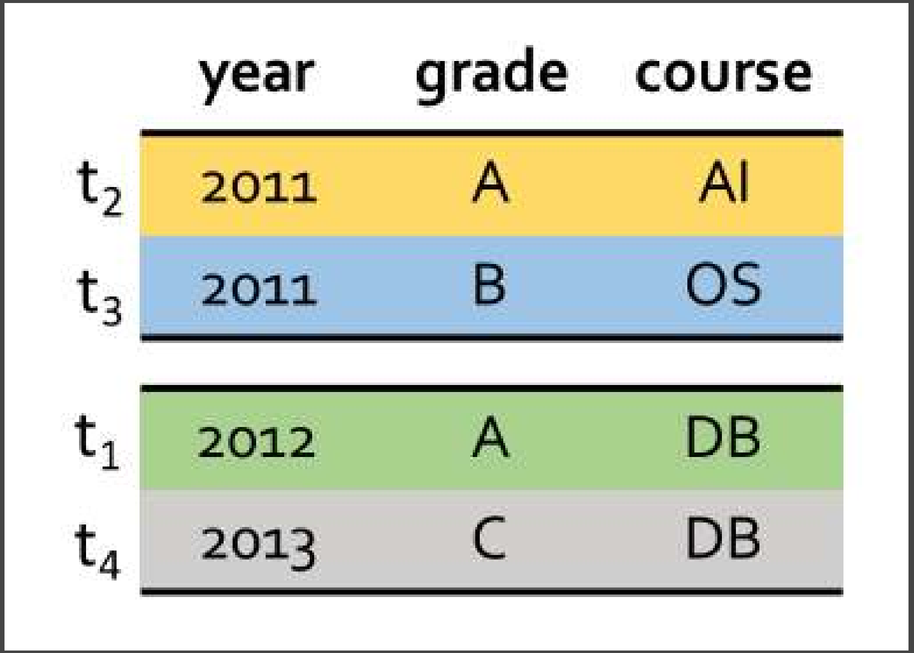




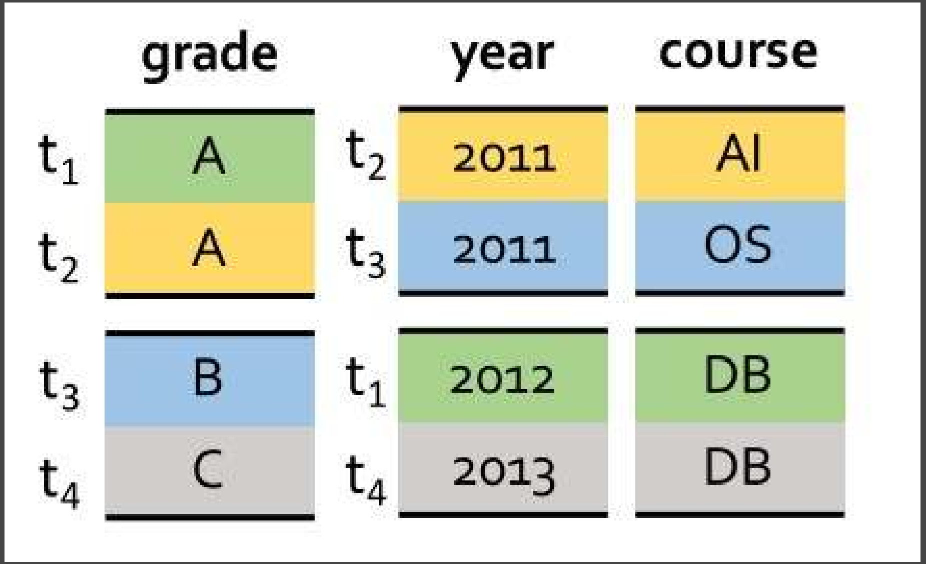
- When $w_h > 0$, $w_v > 0$ and $w_v > w_h$, **prioritizing vertical partitioning**
  - Generalized Skipping-Oriented Partitioning (GSOP)
- When $w_h > 0$, $w_v > 0$ and $w_h > w_v$, **prioritizing horizontal partitioning**

Feature Conflicts

- F1: grade=‘A’; F2:year>2011∧course=‘DB’
- The best partitioning scheme for F1 ***t1t2|t3t4***
- The best partitioning scheme for F2 ***t1t4|t2t3***
- But you have to choose one



	year	grade	course
t_2	2011	A	AI
t_3	2011	B	OS
t_1	2012	A	DB
t_4	2013	C	DB



	grade		year	course
t_1	A	t_2	2011	AI
t_2	A	t_3	2011	OS
t_3	B	t_1	2012	DB
t_4	C	t_4	2013	DB

Background

Generalized Skipping Oriented Partitioning (GSOP)

- Workload Analysis
- Augmentation
- Column Grouping (vertical partitioning)
- Local Feature Selection
- Partitioning (horizontal)

Workload Analysis

- A workload is a collection of queries

- Each query is associated with a filter
  - Filter can be seen as a conjunction of features
  - SELECT B, D FROM T WHERE B<0, D=2

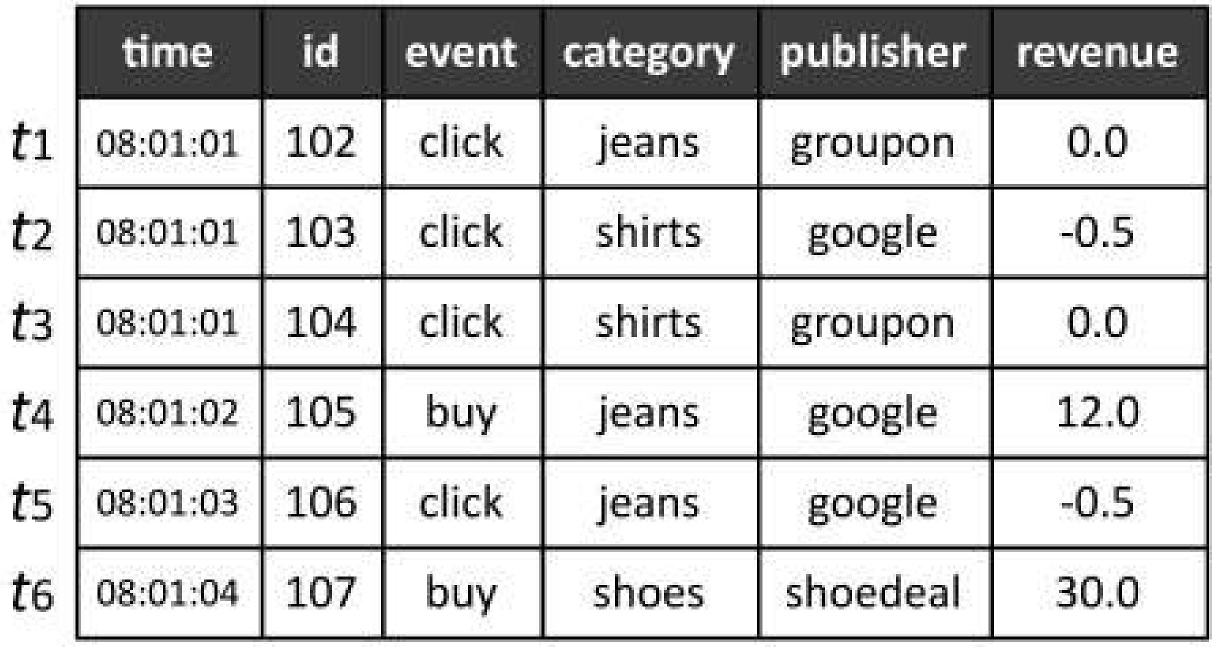
- Find a set of features that occurs in the workload
  - Subsume as many queries as possible

- Q1: prod.=‘shoes’, prod. in (‘shoes’, ‘shirts’)
- Q2: prod. in (‘shoes’, ‘shirts’), revenue>32, revenue> 21
- Q3: prod.=‘shirts’, revenue>21, prod. in (‘shoes’, ‘shirts’)

- F1: {revenue > 21} **subsumes** Q2 and Q3
- F2: {product in (‘shoes’, ‘shirts’)} **subsumes** both Q1 and Q2

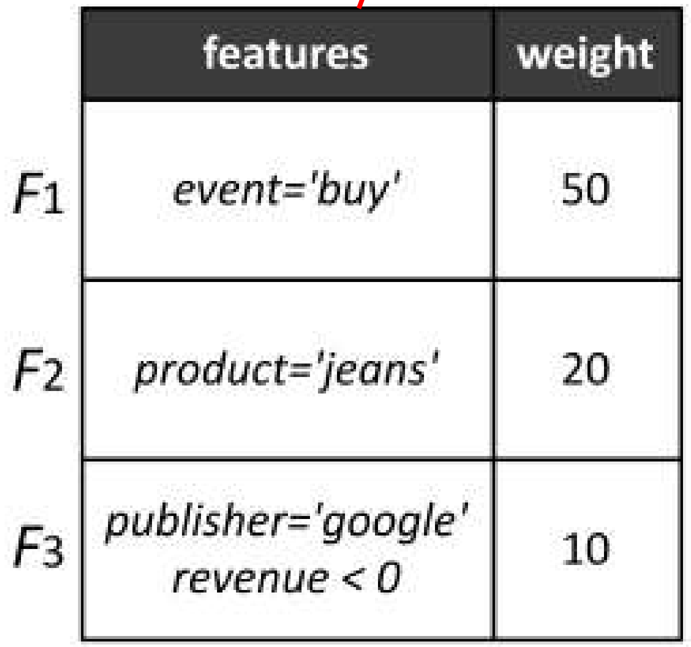
Augmentation

Selected Features



	time	id	event	category	publisher	revenue
t_1	08:01:01	102	click	jeans	groupon	0.0
t_2	08:01:01	103	click	shirts	google	-0.5
t_3	08:01:01	104	click	shirts	groupon	0.0
t_4	08:01:02	105	buy	jeans	google	12.0
t_5	08:01:03	106	click	jeans	google	-0.5
t_6	08:01:04	107	buy	shoes	shoedeal	30.0

(a) tuples



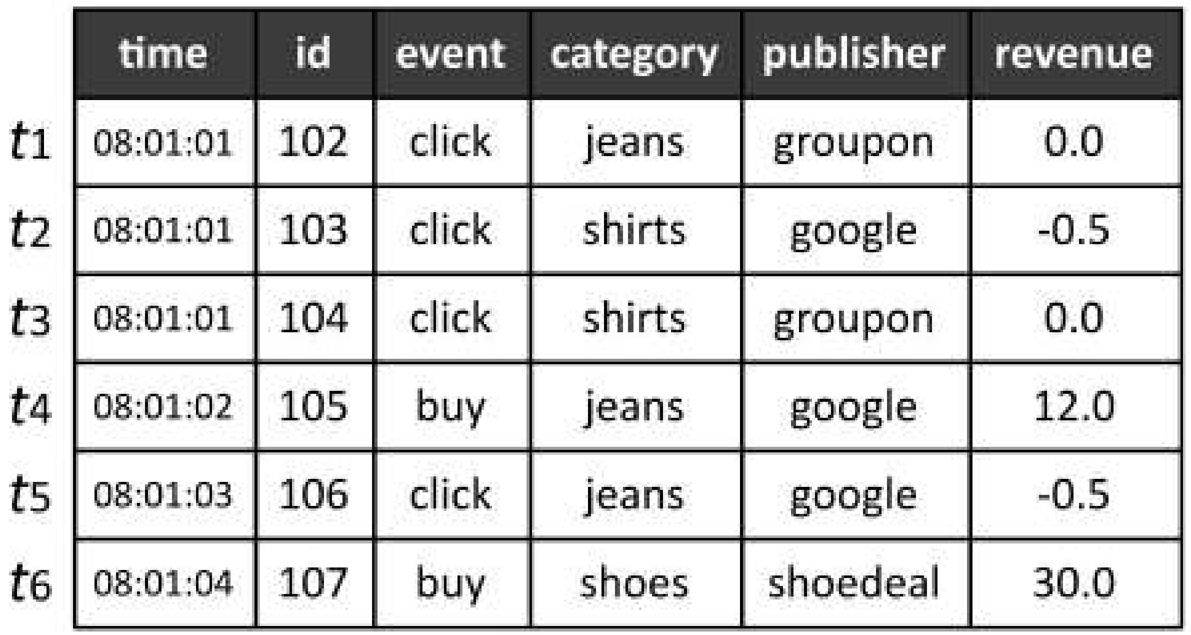
	features	weight
F_1	*event='buy'*	50
F_2	*product='jeans'*	20
F_3	*publisher='google'* *revenue < 0*	10

(b) features

Augmentation

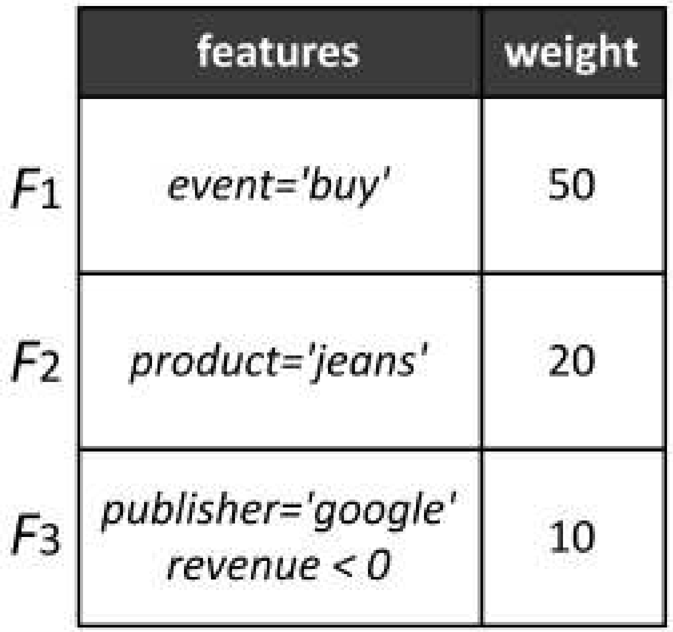
Store the evaluation results
as a bit vector

Batch evaluate these features against each tuple



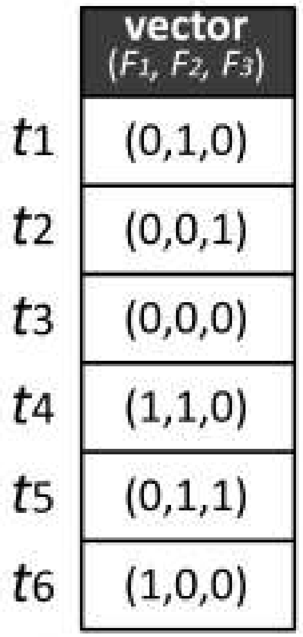
	time	id	event	category	publisher	revenue
t_1	08:01:01	102	click	jeans	groupon	0.0
t_2	08:01:01	103	click	shirts	google	-0.5
t_3	08:01:01	104	click	shirts	groupon	0.0
t_4	08:01:02	105	buy	jeans	google	12.0
t_5	08:01:03	106	click	jeans	google	-0.5
t_6	08:01:04	107	buy	shoes	shoedeal	30.0

(a) tuples



	features	weight
F_1	*event='buy'*	50
F_2	*product='jeans'*	20
F_3	*publisher='google'* *revenue < 0*	10

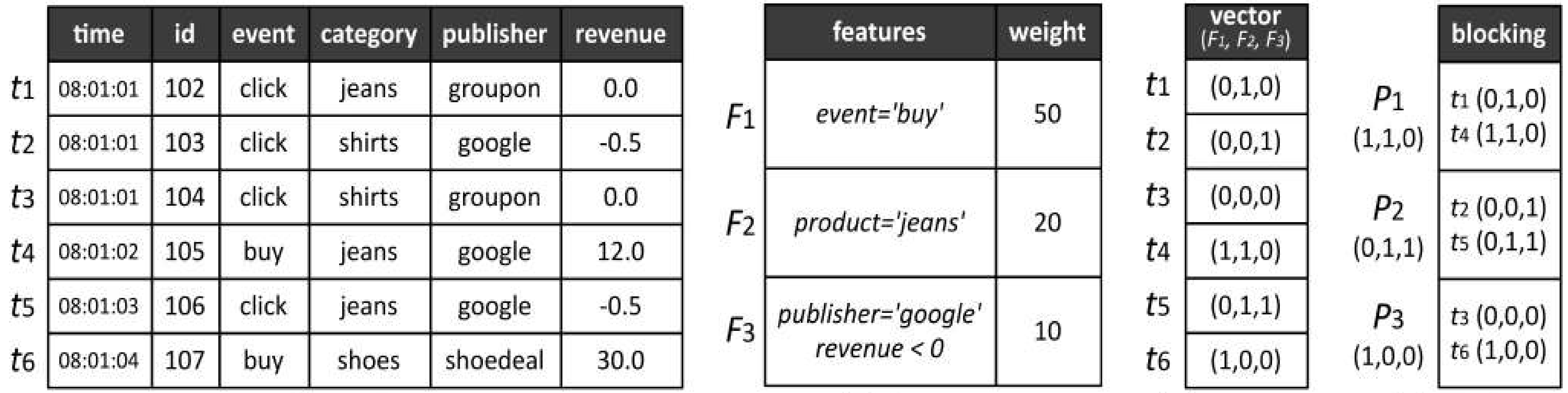
(b) features



	vector (F_1, F_2, F_3)
t_1	(0,1,0)
t_2	(0,0,1)
t_3	(0,0,0)
t_4	(1,1,0)
t_5	(0,1,1)
t_6	(1,0,0)

(c) vectors

Data Skipping with Bit Vectors



	time	id	event	category	publisher	revenue
t_1	08:01:01	102	click	jeans	groupon	0.0
t_2	08:01:01	103	click	shirts	google	-0.5
t_3	08:01:01	104	click	shirts	groupon	0.0
t_4	08:01:02	105	buy	jeans	google	12.0
t_5	08:01:03	106	click	jeans	google	-0.5
t_6	08:01:04	107	buy	shoes	shoedeal	30.0

(a) tuples

	features	weight
F_1	*event='buy'*	50
F_2	*product='jeans'*	20
F_3	*publisher='google' revenue < 0*	10

(b) features

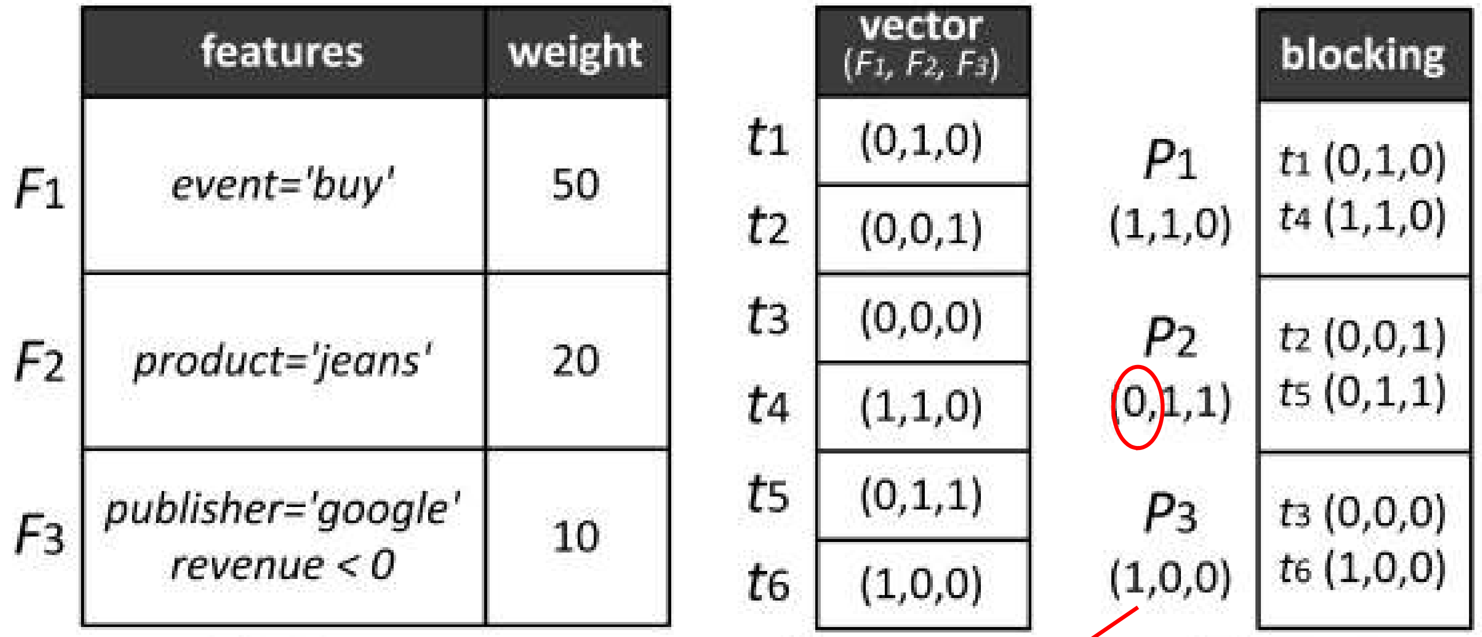
	vector (F_1, F_2, F_3)
t_1	(0,1,0)
t_2	(0,0,1)
t_3	(0,0,0)
t_4	(1,1,0)
t_5	(0,1,1)
t_6	(1,0,0)

(c) vectors

	blocking
P_1 (1,1,0)	t_1 (0,1,0) t_4 (1,1,0)
P_2 (0,1,1)	t_2 (0,0,1) t_5 (0,1,1)
P_3 (1,0,0)	t_3 (0,0,0) t_6 (1,0,0)

(d) blocks

Data Skipping with Bit Vectors



	features	weight
F_1	*event='buy'*	50
F_2	*product='jeans'*	20
F_3	*publisher='google'* *revenue < 0*	10

(b) features

	vector (F_1, F_2, F_3)
t_1	(0,1,0)
t_2	(0,0,1)
t_3	(0,0,0)
t_4	(1,1,0)
t_5	(0,1,1)
t_6	(1,0,0)

(c) vectors

	blocking
P_1 (1,1,0)	t_1 (0,1,0) t_4 (1,1,0)
P_2 (0,1,1)	t_2 (0,0,1) t_5 (0,1,1)
P_3 (1,0,0)	t_3 (0,0,0) t_6 (1,0,0)

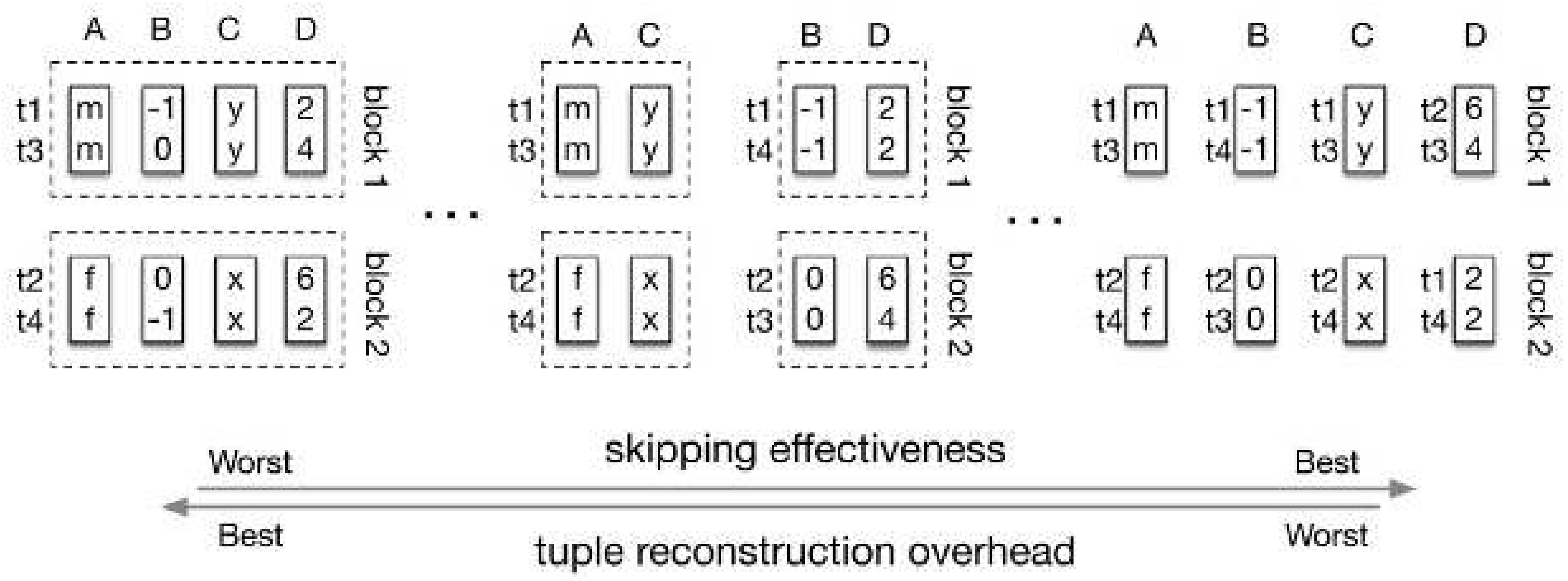
(d) blocks

Union Vector(OR)

Query: SELECT *publisher* FROM *table* WHERE *F1*

P2 can be safely skipped

Spectrum of Partitioning


A B C D
t1 t3 m m -1 0 y y 2 4 block 1
t2 t4 f f 0 -1 x x 6 2 block 2
A C
t1 t3 m m y y
t2 t4 f f x x
B D
t1 t4 -1 -1 2 2 block 1
t2 t3 0 0 6 4 block 2
A B C D
t1 t3 m m; t1 t4 -1 -1; t1 t3 y y; t2 t3 6 4 block 1
t2 t4 f f; t2 t3 0 0; t2 t4 x x; t1 t4 2 2 block 2
skipping effectiveness
Worst
Best
Best
Worst
tuple reconstruction overhead

- Two Extremes:
  - All columns follow the same horizontal partitioning scheme
  - Each column can have its own partitioning scheme
- Which one is better?
  - Depends on the workload and data characteristics

Column Grouping

Column Grouping

- Divide columns into column groups
- An objective function
    - Tradeoff (skipping effectiveness, tuple reconstruction)
    - The opportunities of skipping horizontal blocks within each column group

C: the set of columns in the table

G = {G₁, G₂, . . . , Gm}: a column grouping scheme of the

= *C* ; for

Given query *q*

: the set of columns *q* needs to access

: the column groups *q* needs to access

: # rows that *q* needs to scan in

Objective Function

- Skipping Effectiveness:
  - The overall scanning cost for query *q* is:

$$\sum_{G_i \in \mathbb{G}^q} |G_i \cap C^q| \cdot r_i^q. \longrightarrow \text{\# cells}$$

# columns q accesses

rows, depends on horizontal partitioning

Objective Function

C: the set of columns in the table

G = {*G*$_1$, *G*$_2$, . . . , *Gm*}: a column grouping scheme of the

= *C* ; for

Given query *q*

: the set of columns *q* needs to access

: the column groups *q* needs to access

: # rows that *q* needs to scan in

- Tuple Reconstruction Overhead
  - Store tuple-id for each row
  - Assume that we use sort-merge join to do tuple reconstruction

$$\text{overhead}(q, \mathbb{G}) = \begin{cases} \sum_{G_i \in \mathbb{G}^q} (r_i^q + \text{sort}(r_i^q)) & \text{if } |\mathbb{G}^q| > 1 \\ 0 & \text{otherwise} \end{cases}$$

Objective Function

C: the set of columns in the table

$G = \{G_1, G_2, \ldots, Gm\}$: a column grouping scheme of the

= *C* ; for

Given query *q*

: the set of columns *q* needs to access

: the column groups *q* needs to access

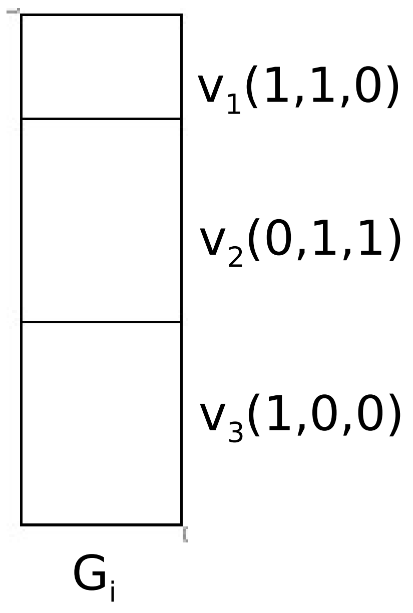
: # rows that *q* needs to scan in

$$\mathsf{COST}(q, \mathbb{G}) = \sum_{G_i \in \mathbb{G}^q} |G_i \cap C^q| \cdot r_i^q + \mathsf{overhead}(q, \mathbb{G})$$

$$\mathsf{COST}(W, \mathbb{G}) = \sum_{q \in W} \mathsf{COST}(q, \mathbb{G})$$

Efficient Estimation of

- : # rows that *q* needs to scan in
- Exact computation of is very expensive



$V = \{(1,0,0), (0,1,1), (1,0,1)\}$

The set of distinct vectors in G_i

$\text{count}(v_1) = 10$

# rows whose feature vector is v

b: the size of a block (skipping granularity)

, an upperbound

total # tuples

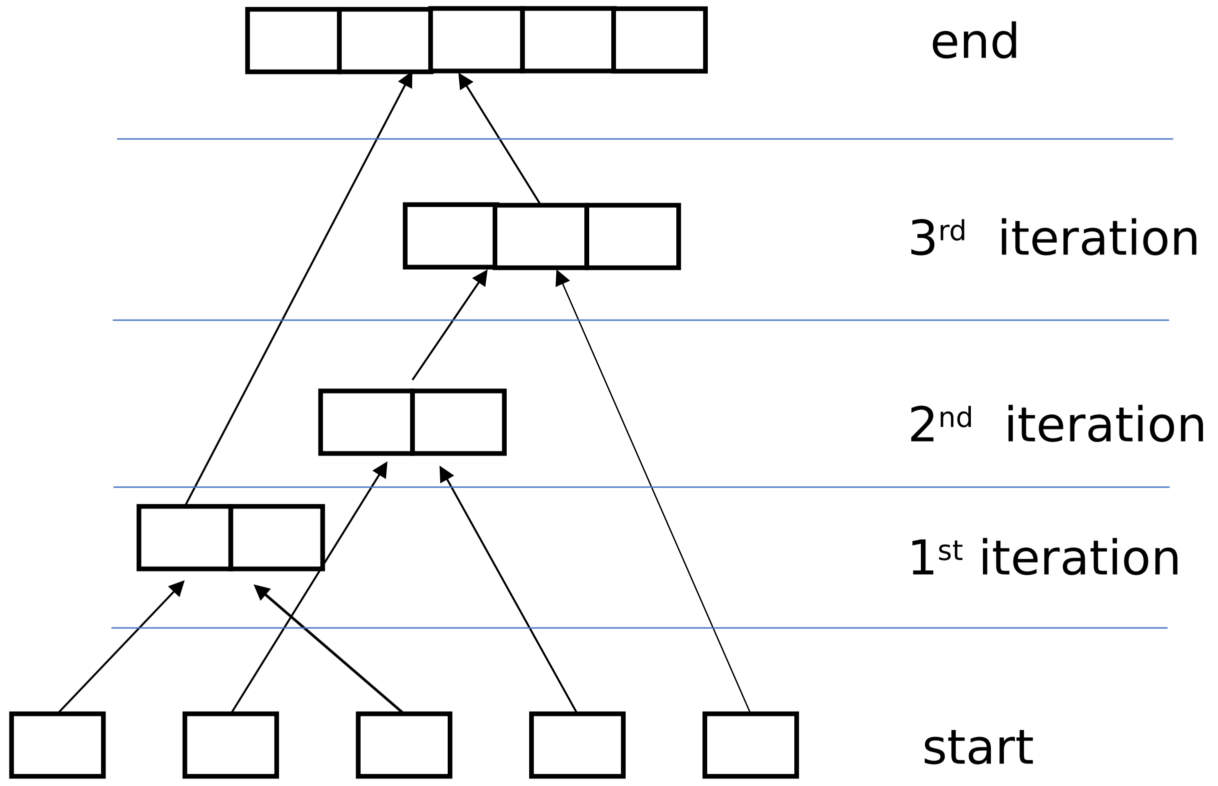
min # blocks to skip

Efficient Estimation of

- Exact computation of is very expensive
- Estimation:
  - Group the rows that have the same feature vectors
  - Let be the set of distinct vectors after grouping in G_i
  - For , is the number of rows whose feature vector is
  - is the size of a block
  - Given a query , divide into and
  - , an upperbound

Bottom-up Search Strategy

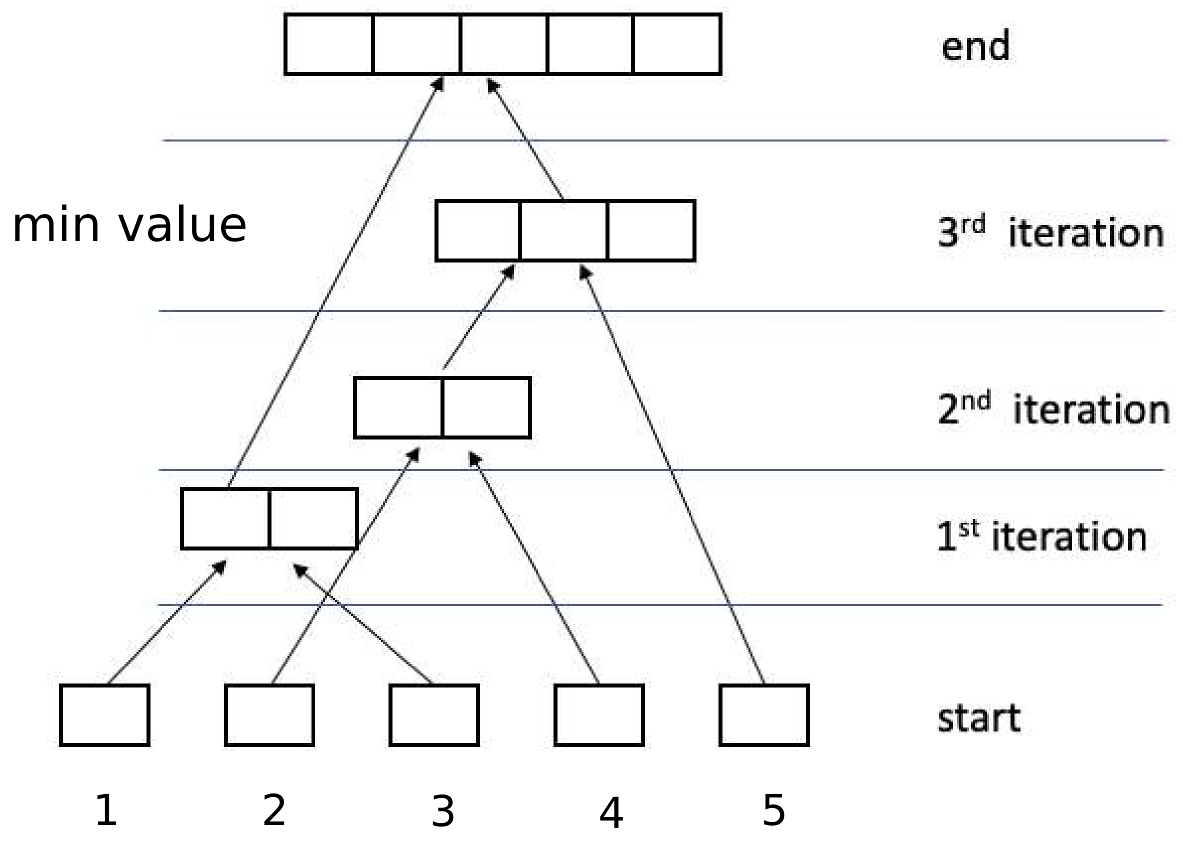
- Initially, each column itself forms a group
- Iteratively choose two groups to merge until all columns are in one group
- The merge should lead to the minimum value of the obj function



Bottom-up Search Strategy

- Pick the iteration where the objective function has the minimum value and return the corresponding grouping scheme

- Evaluate the obj function (estimation) times



: the set of queries that need to access column group
: the features that subsume

Local Feature Selection

- For each column group, the set of features that are most helpful in block data skipping is different

- How do we decide the set of features that is the most helpful?

$\downarrow$

The set of features that are subsumed by at least one query in the workload

: the set of queries that need to access column group
: the features that subsume

Local Feature Selection

$$\text{weight}(G, f) = \left|\{q \mid f \in F^q \text{ and } q \in W^G\}\right|$$

- Create a ranked list of local features for each column group

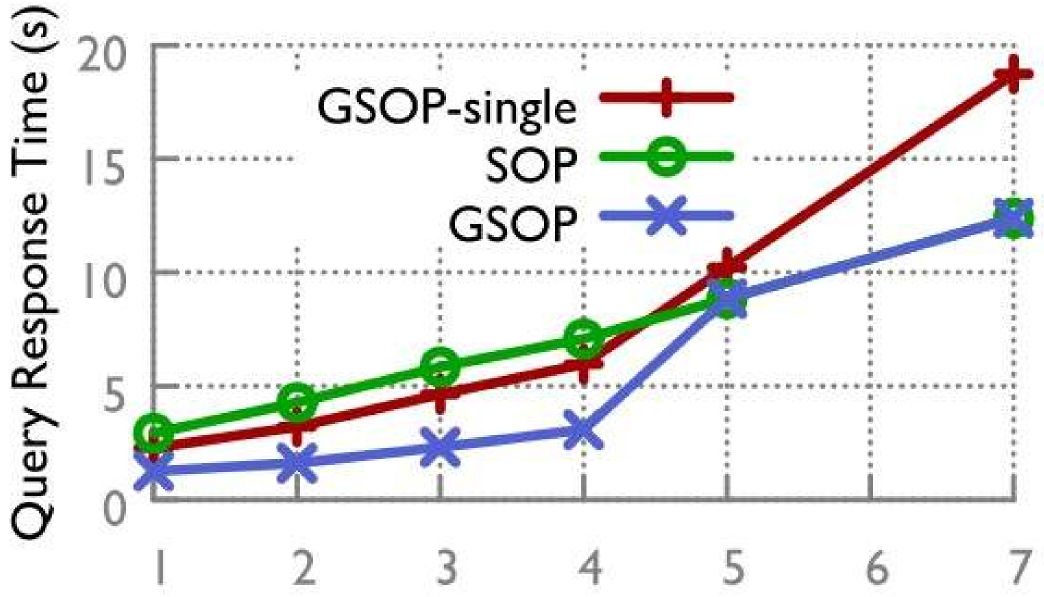
- Determine how many features to use for partitioning
  - Set a heuristic number

Horizontal Partitioning

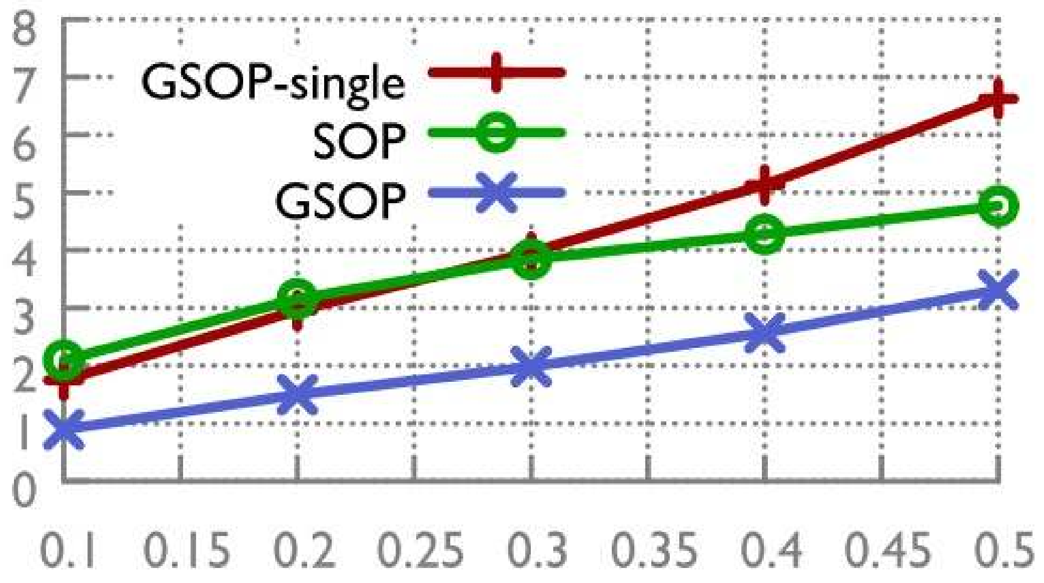
- L. Sun, M. J. Franklin, S. Krishnan, and R. S. Xin. Fine-grained partitioning for aggressive data skipping. In SIGMOD, pages 1115–1126, 2014.

Experiments

Big Data Benchmark



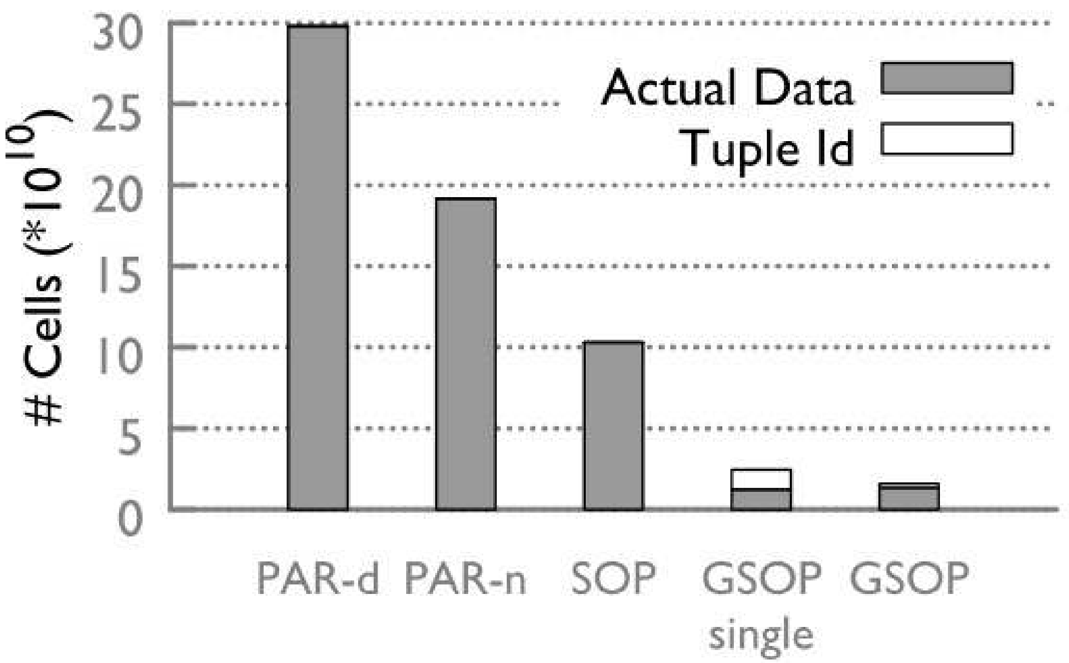
(a) Varying #Cols



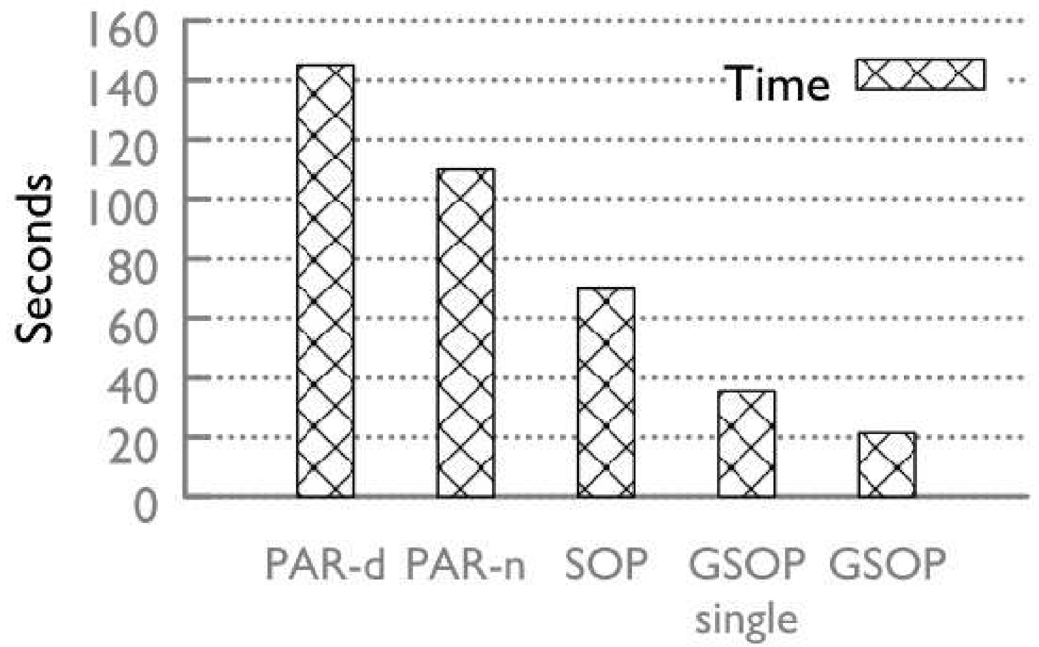
(d) Varying Selectivity

Experiments

TPC-H Benchmark



(a) # Cells Read

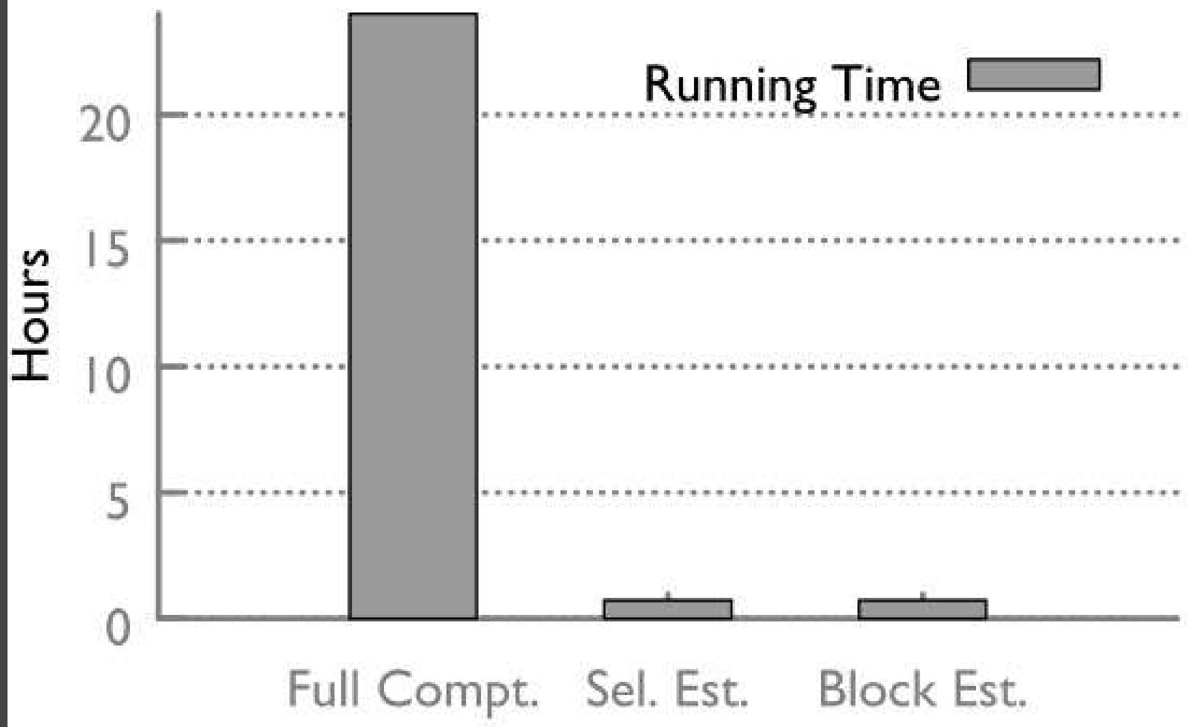


(b) Query Response Time

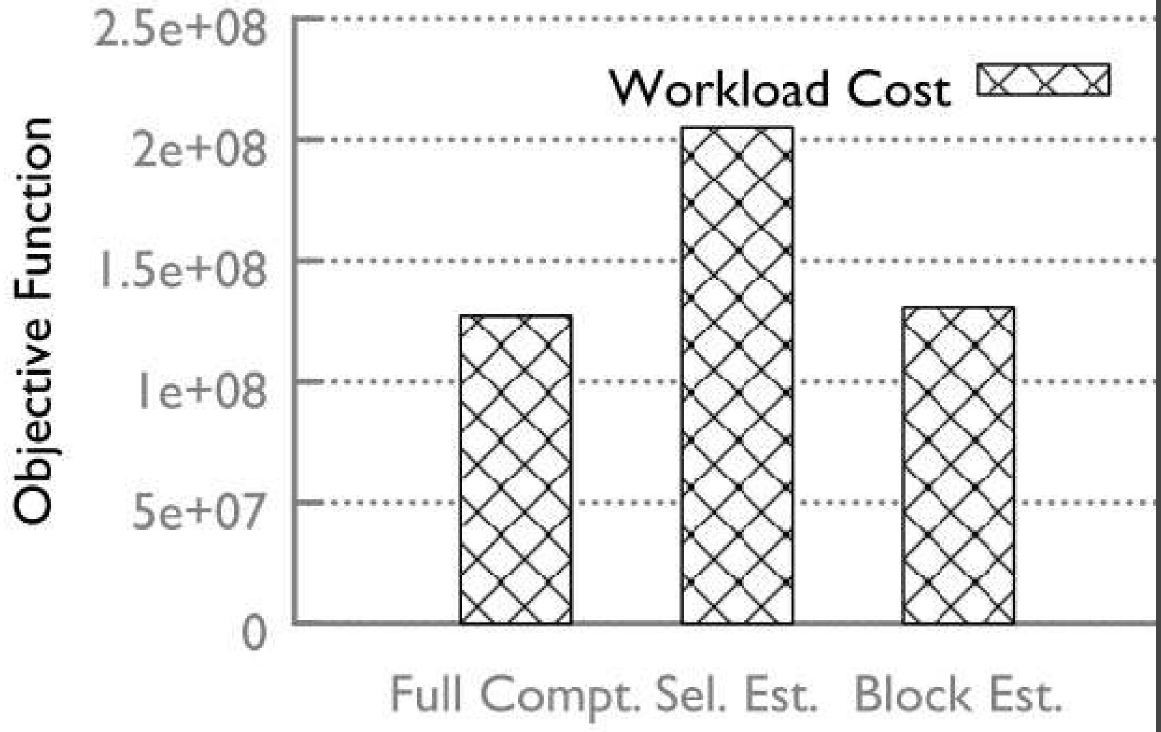
100 test queries, hundreds of millions of rows

Is the estimation good?

- Full Compt: compute the exact value of the obj function
- Sel. Est.: baseline estimation based on traditional selectivity
- Block Est.: proposed block-based estimation
- TPC-H



(a) Efficiency Comparision



(b) Quality Comparison

Conclusion

- Develop a novel hybrid data skipping framework (GSOP)
  - Take into account these row-based and column-based tradeoffs

- GSOP can always find a partitioning layout no worse than SOP
  - Significantly reduce the amount of data scanned
  - Improve end-to-end query response times