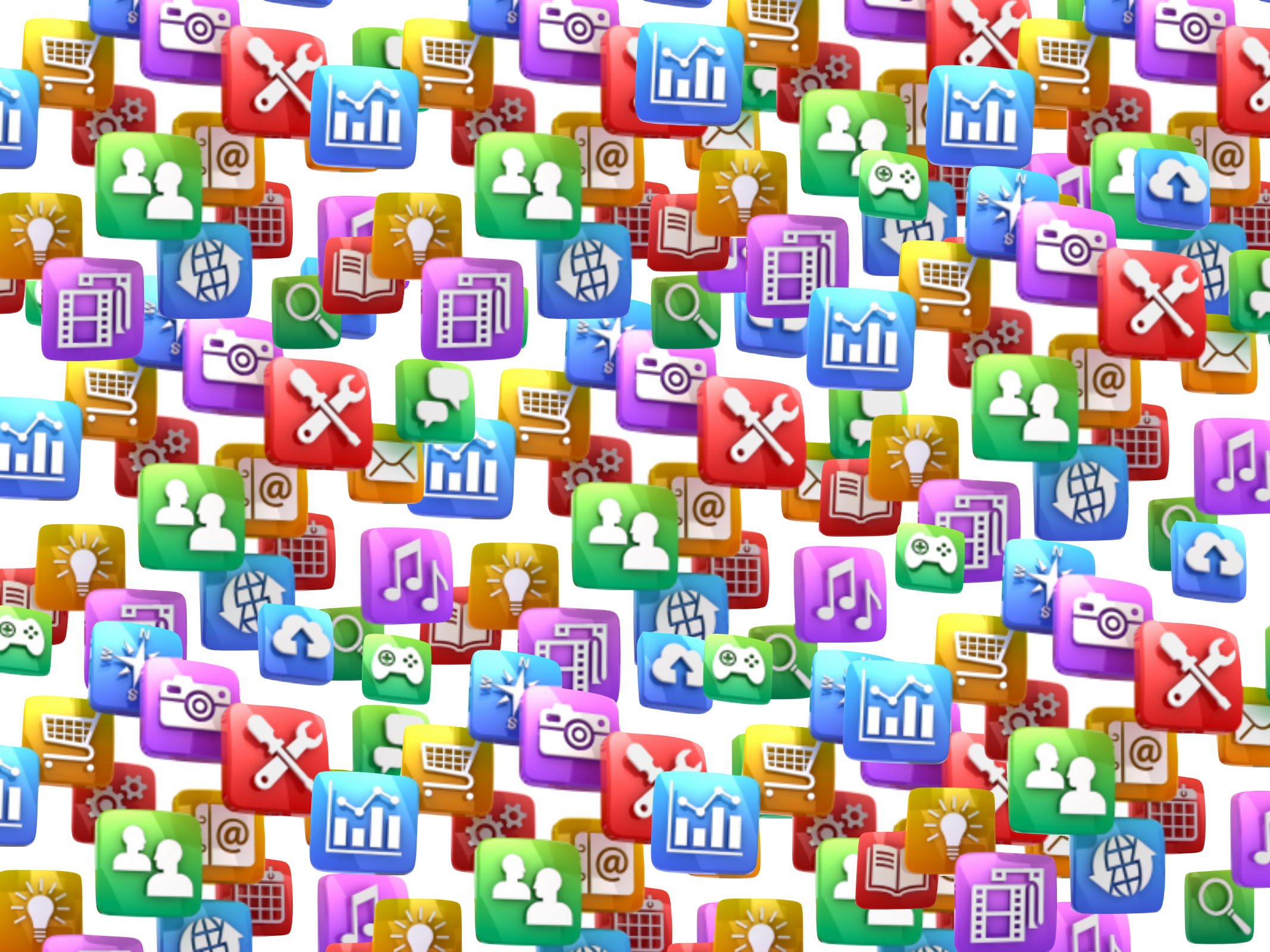


Workload?

OLTP

OLAP

Streaming

Scan-oriented

Archiving

Log-processing

Web-search

Streaming

OLAP

OLTP

Archiving

Log-processing

Web-search

Scan-oriented

Streaming

amazon.com

ebay

OLAP

OLTP

Archiving

Log-processing

Web-search

Scan-oriented

Streaming

amazon.com

ebay

OLAP

OLTP

Archiving

facebook

Log-processing

Web-search

Scan-oriented

Streaming

amazon.com

ebay

OLAP

OLTP

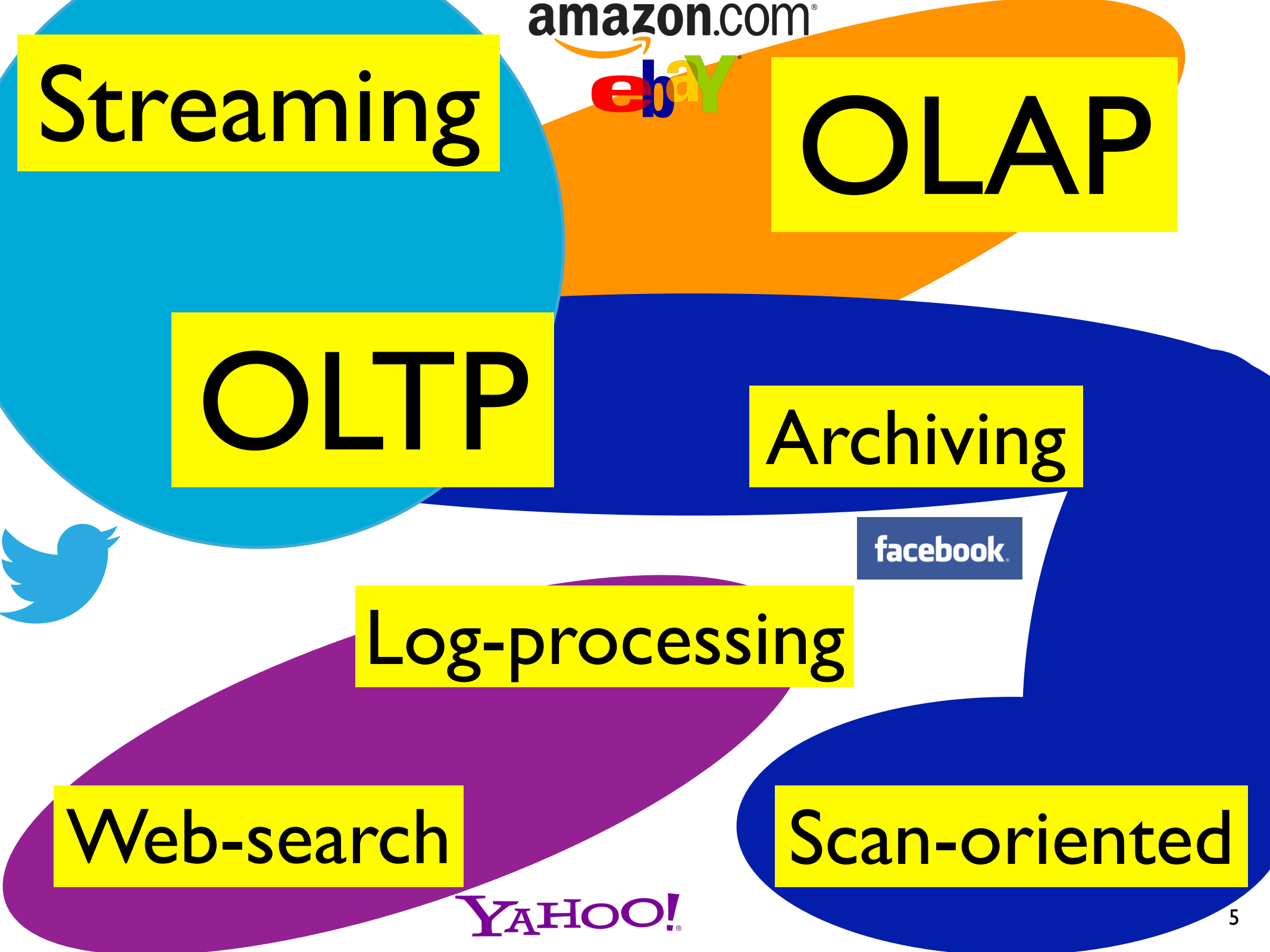
Archiving

facebook

Log-processing

Web-search

Scan-oriented


amazon.com
ebay
Streaming
OLAP
OLTP
Archiving
facebook
Log-processing
Web-search
Scan-oriented
YAHOO!

Streaming

OLAP

OLTP

Archiving

Log-processing

Web-search

Scan-oriented

Streaming

OLAP

OLTP

Archiving

Log-processing

Web-search

Scan-oriented

Streaming

OLAP

OLTP

Archiving

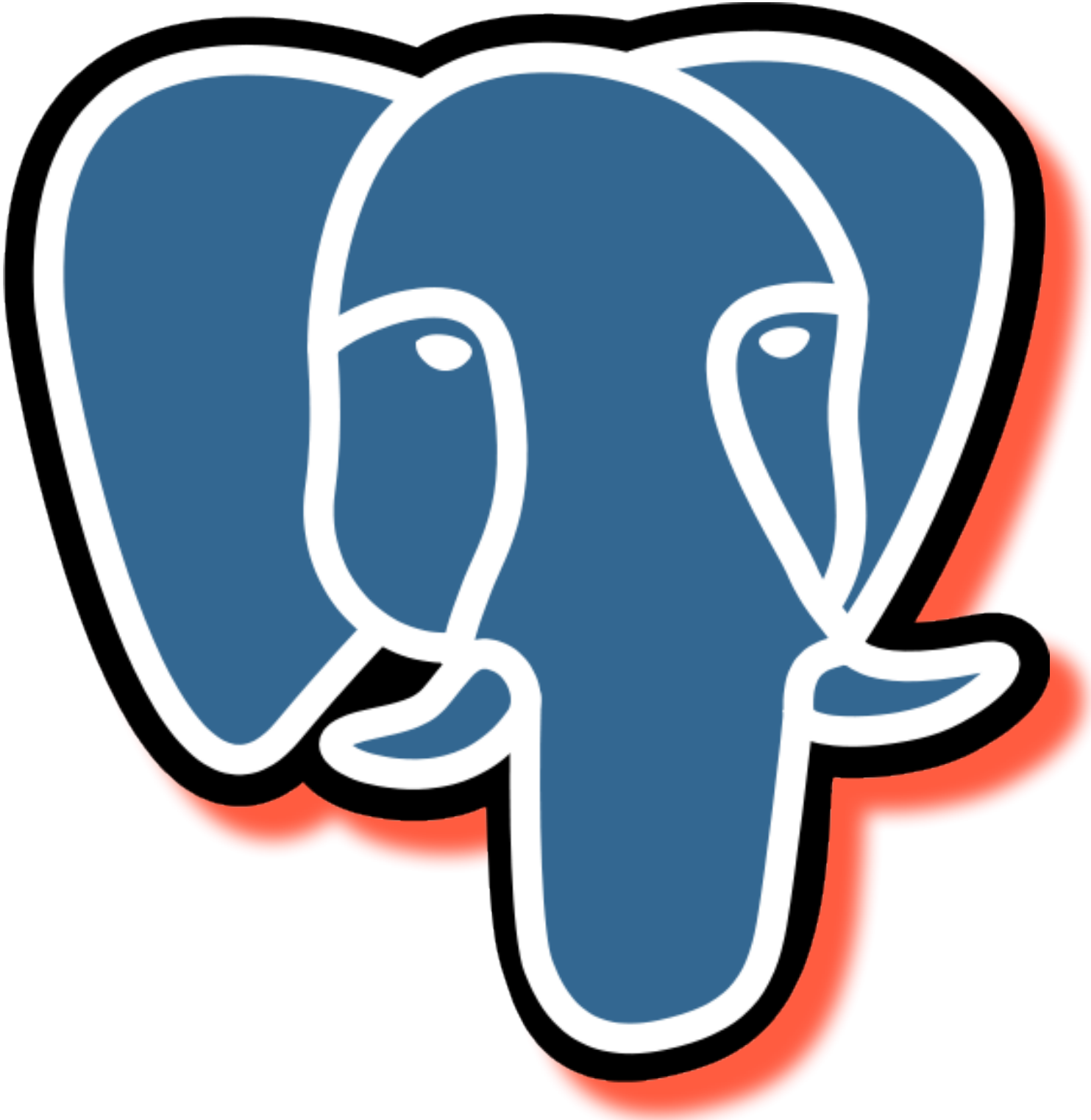
Log-processing

Web-search

Shared-scans


BIG
DATA

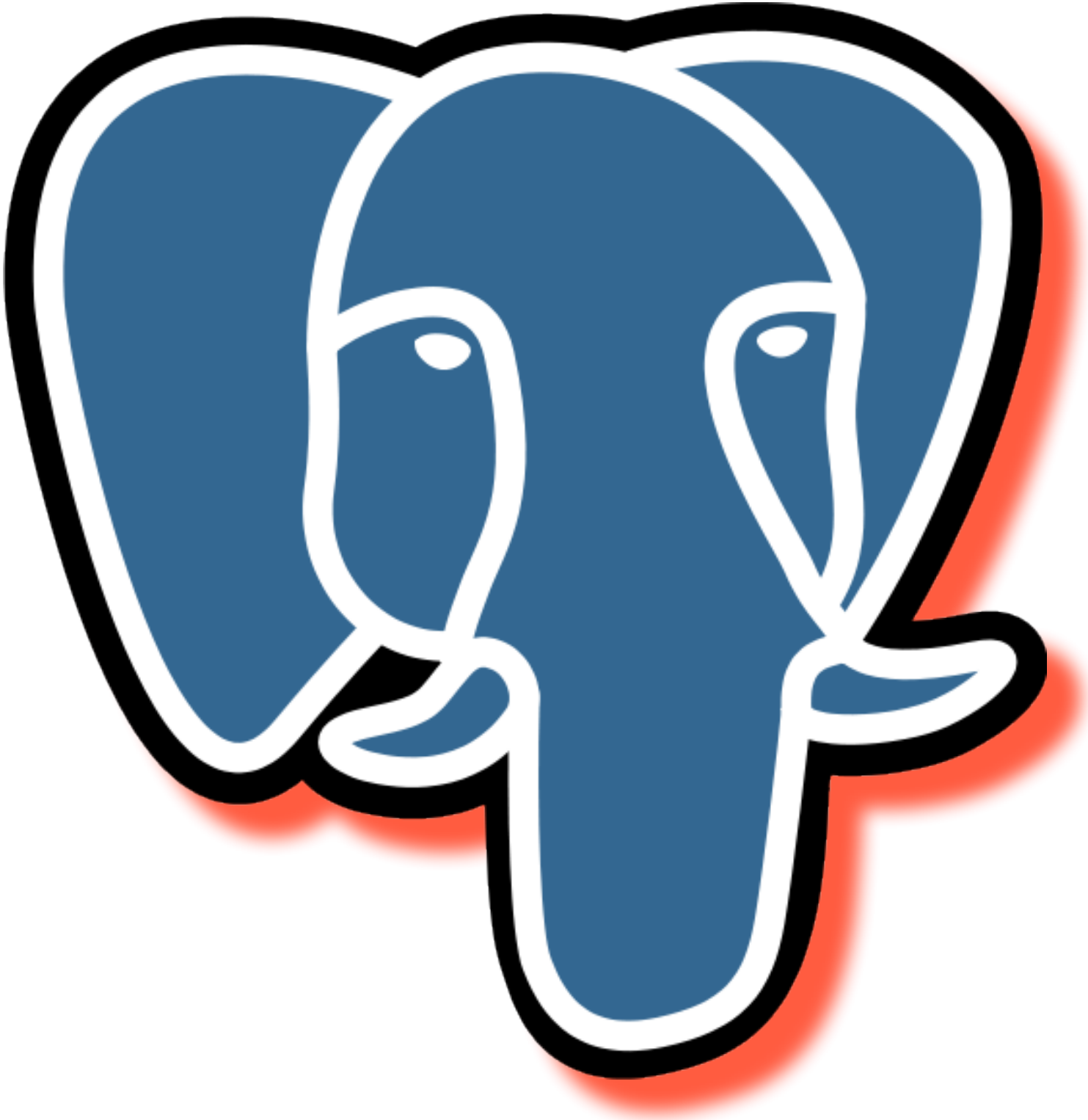

OOPs



- Primary/Secondary Index
- Materialized Views
- Vertical Partitioning
- Horizontal Partitioning

- Primary/Secondary Index
- Materialized Views
- Vertical Partitioning
- Horizontal Partitioning

- Human Intervention
- Slow Response Time
- Inefficient



- Online Indexes
- Dynamic Materialized Views
- Database Cracking
- Adaptive Merging

- Online Indexes
- Dynamic Materialized Views
- Database Cracking
- Adaptive Merging

- Still, does not change core system features
- Several physical designs at schema level
- No true physical data independence
- Physical design not effective

PLAN B?

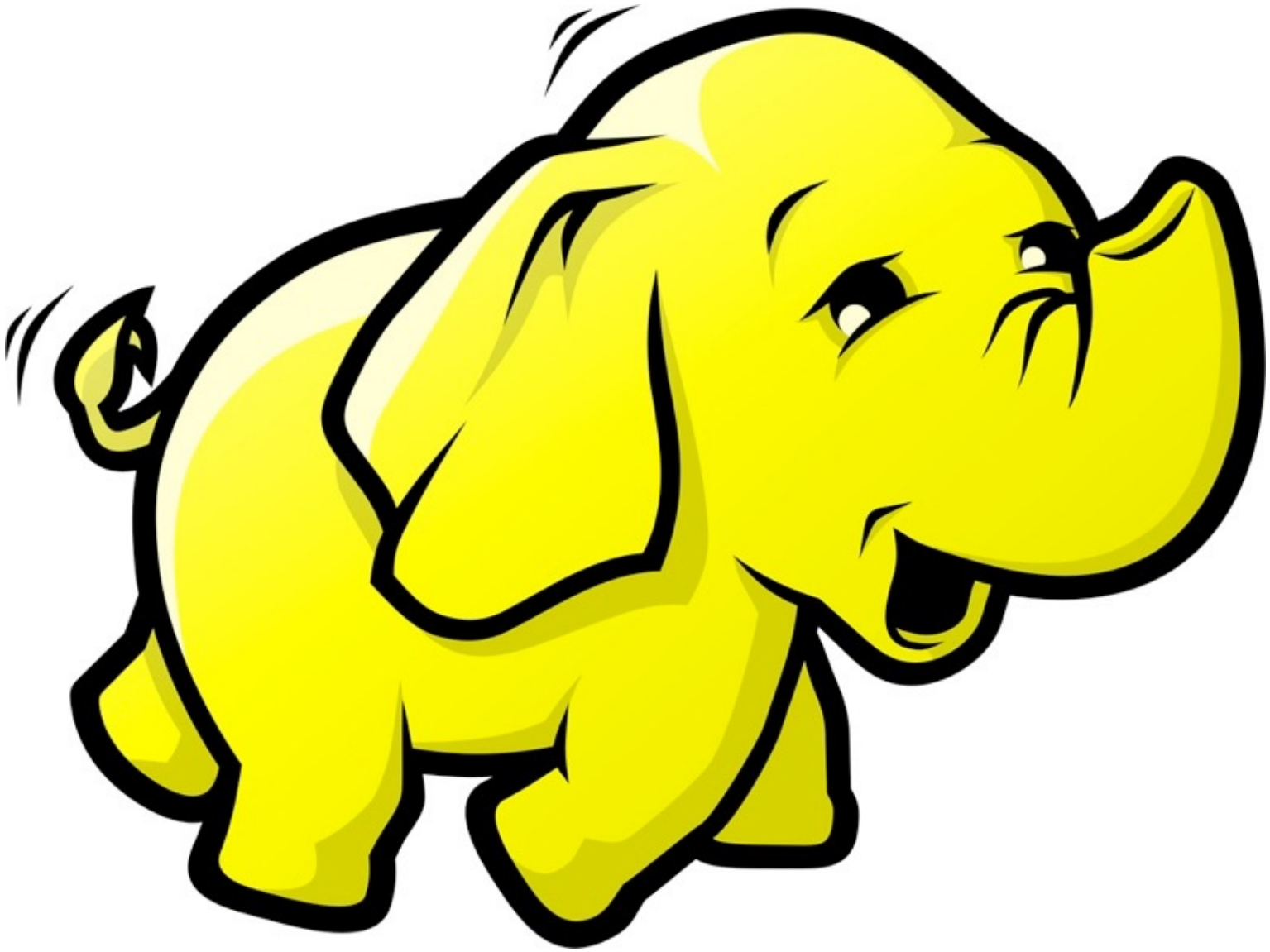
OLTP

OLAP

Archive

Streaming

Log-processing



Streaming

OLAP

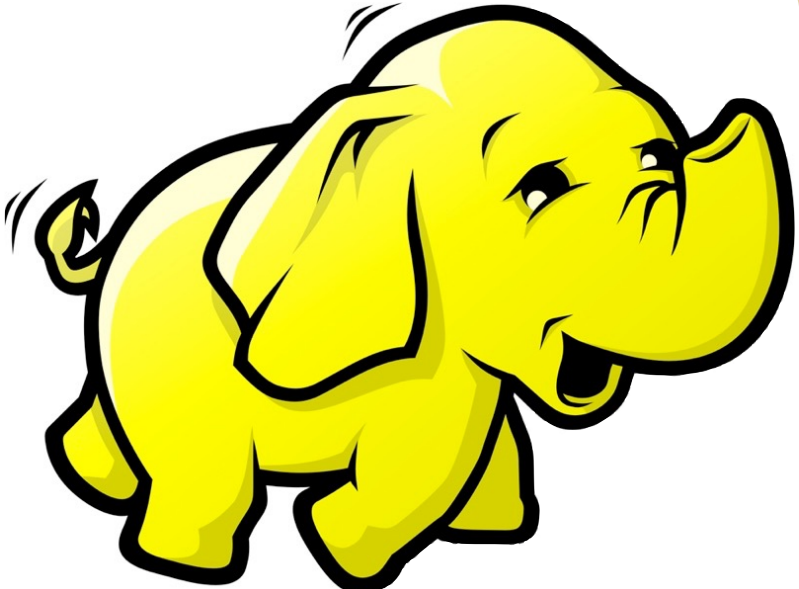
OLTP

Archiving

Log-processing

Web-search

Scan-oriented



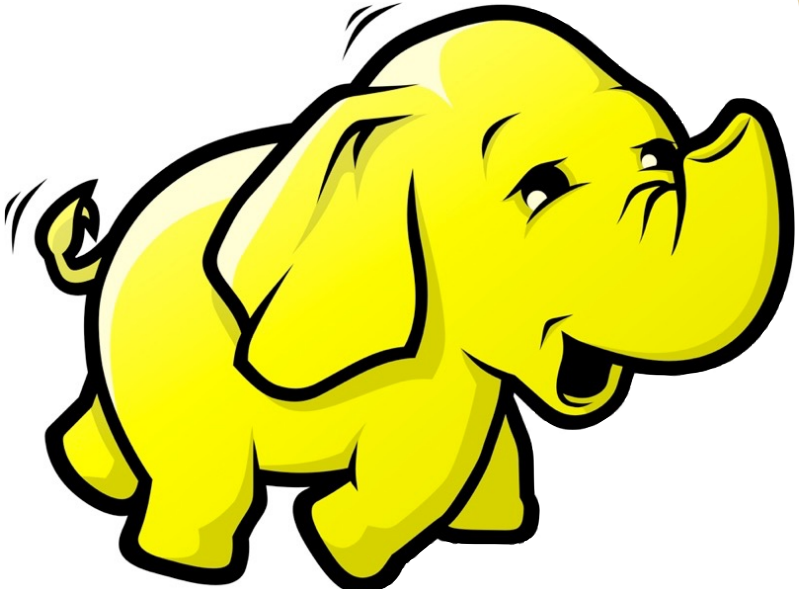
- Tedious
- Expensive
- Complex ETL
- Inefficient

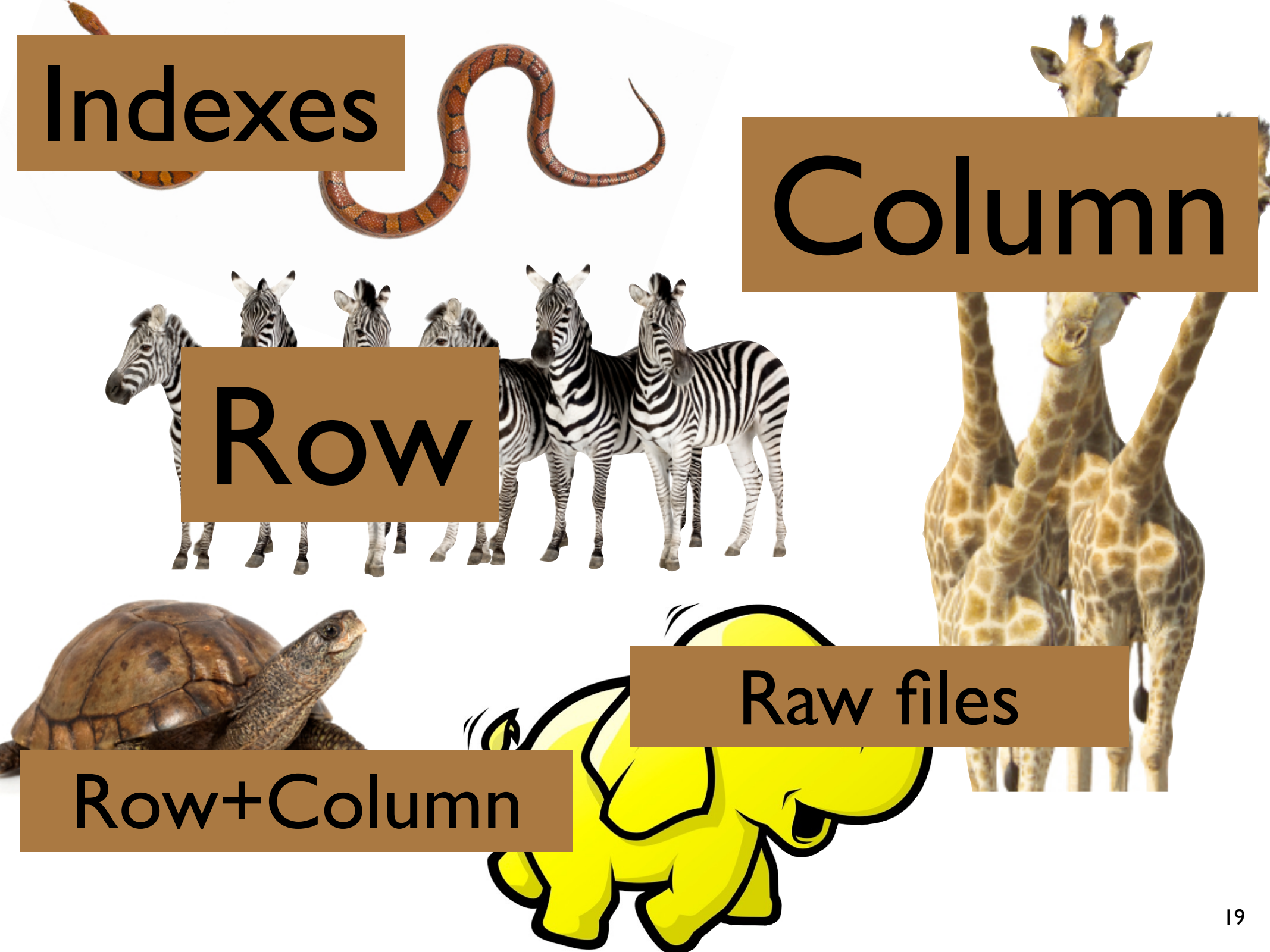
Mixed Workloads

Dynamic Workloads

‘Zoo’ of systems


HELP




Indexes
Column
Row
Raw files
Row+Column

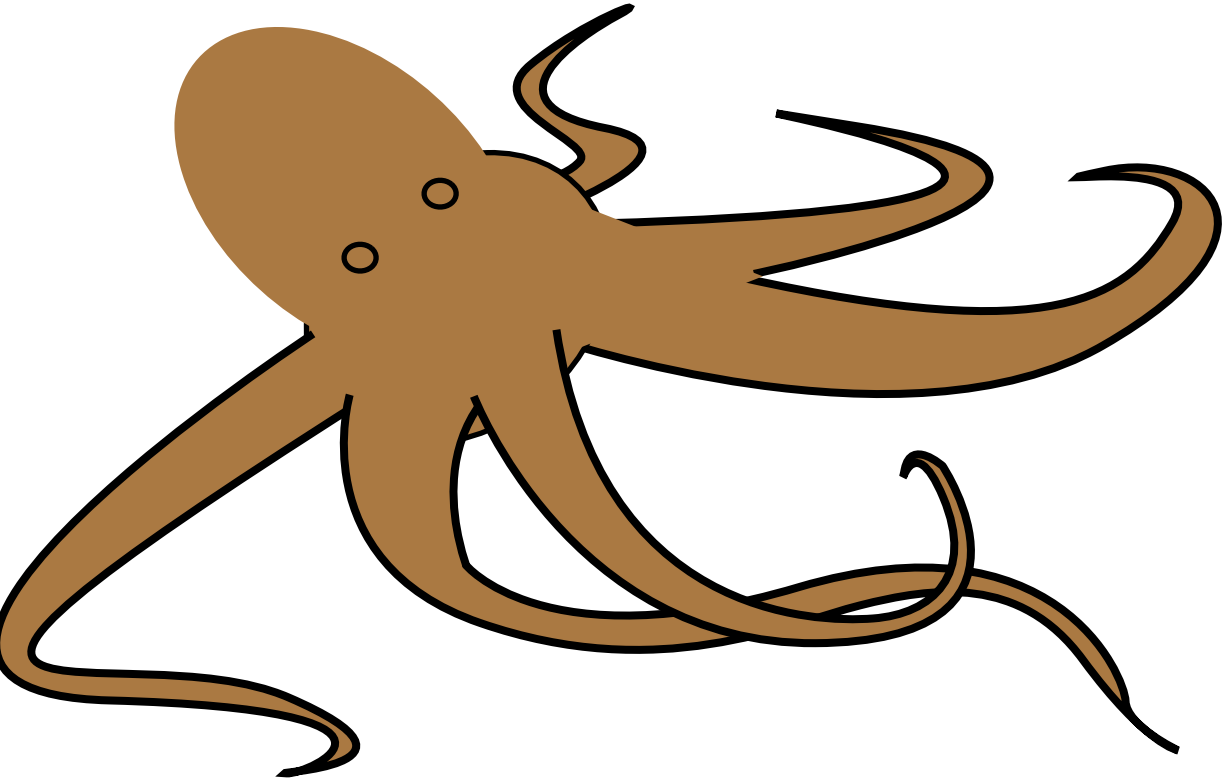
Indexes

Column

Row

Raw files

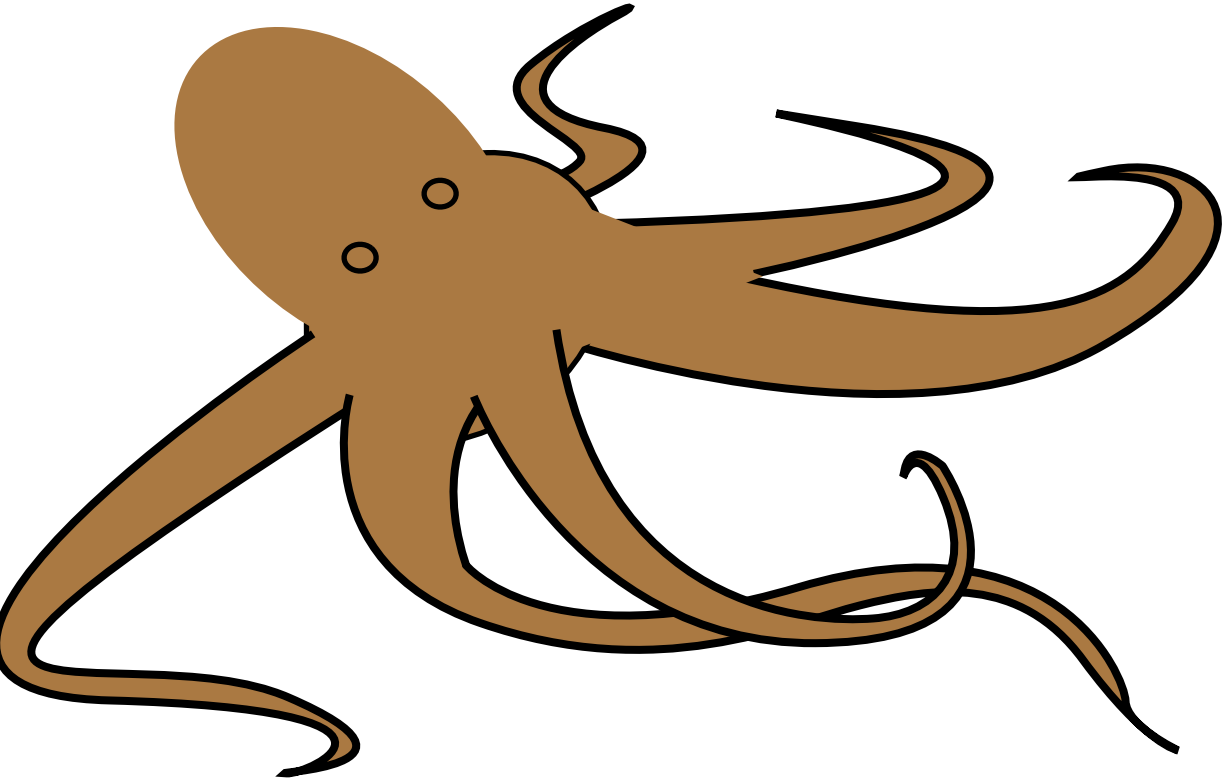
Row+Column



OctopusDB

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

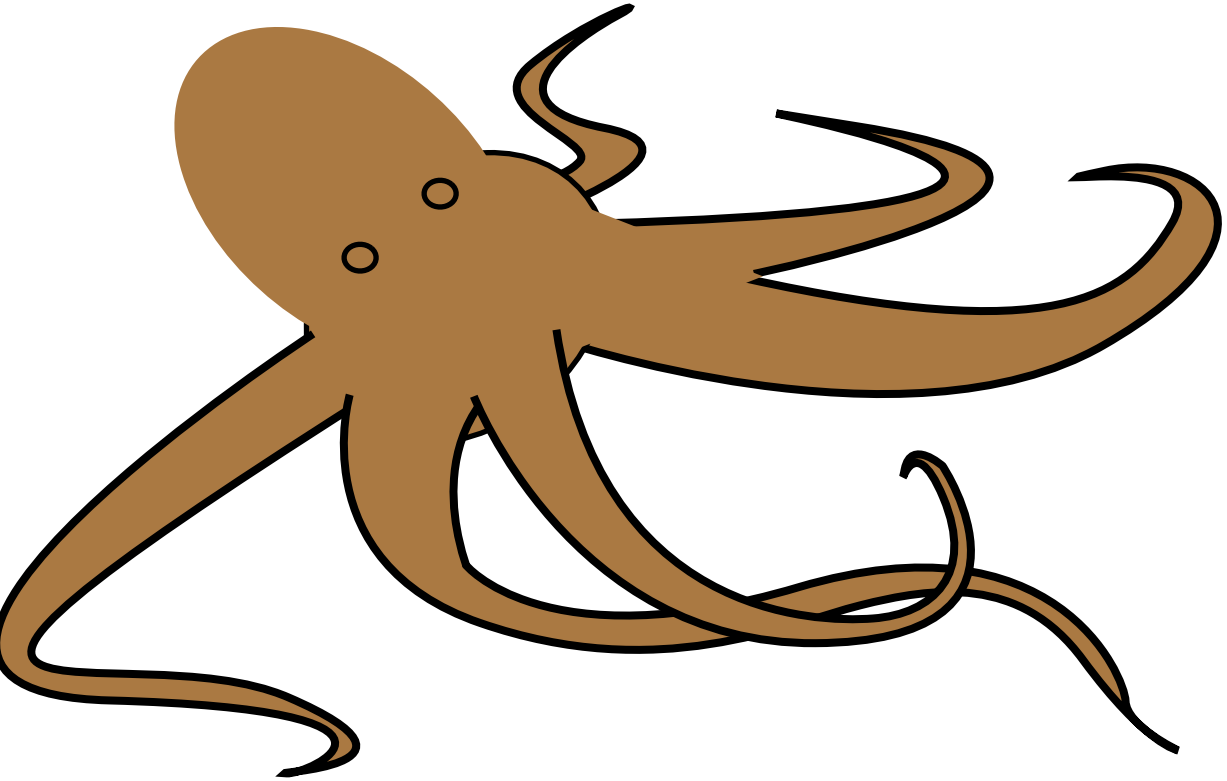
OctopusDB



1	abc	56	887.9
2	fdg	89	445.35
3	poe	67	234.67
4	lkj	12	385.92
5	yui	17	612.13
6	omg	90	148.9

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

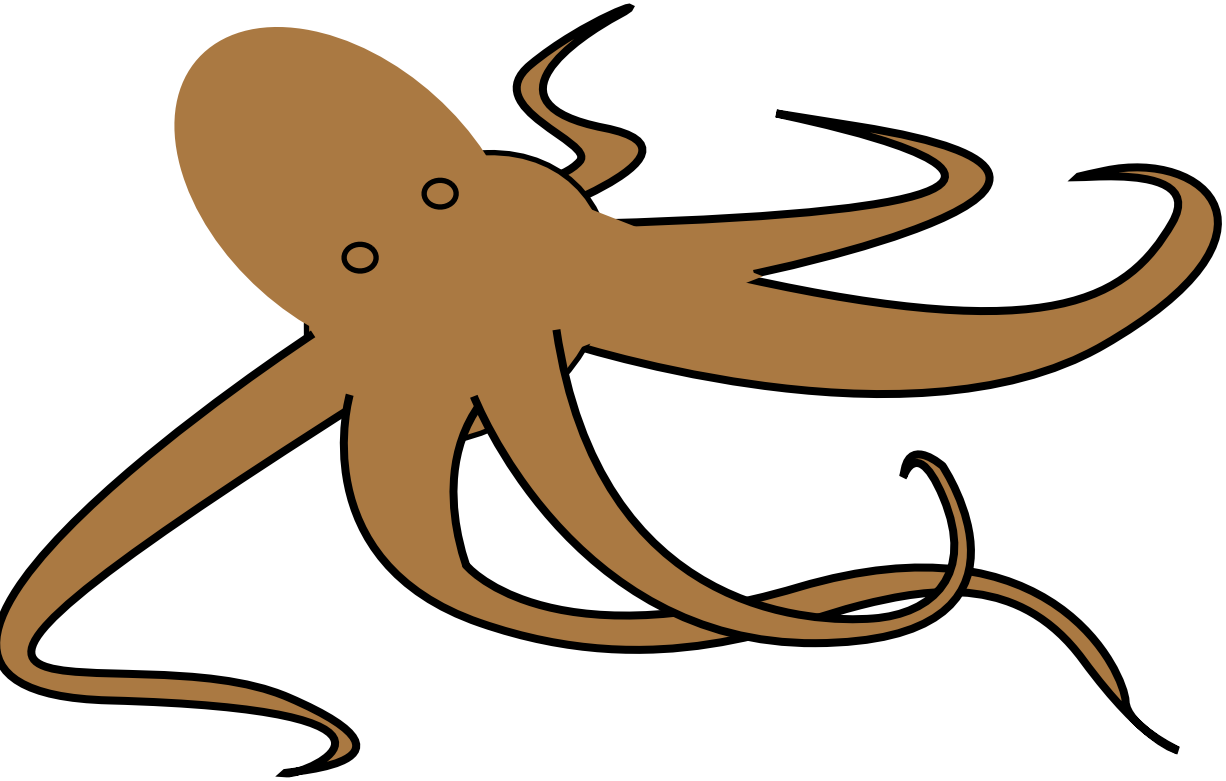
OctopusDB



Log			
		56	887.9
		89	445.35
		67	234.67
4	lkj	12	385.92
5	yui	17	612.13
[illegible]	omg	90	148.9

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

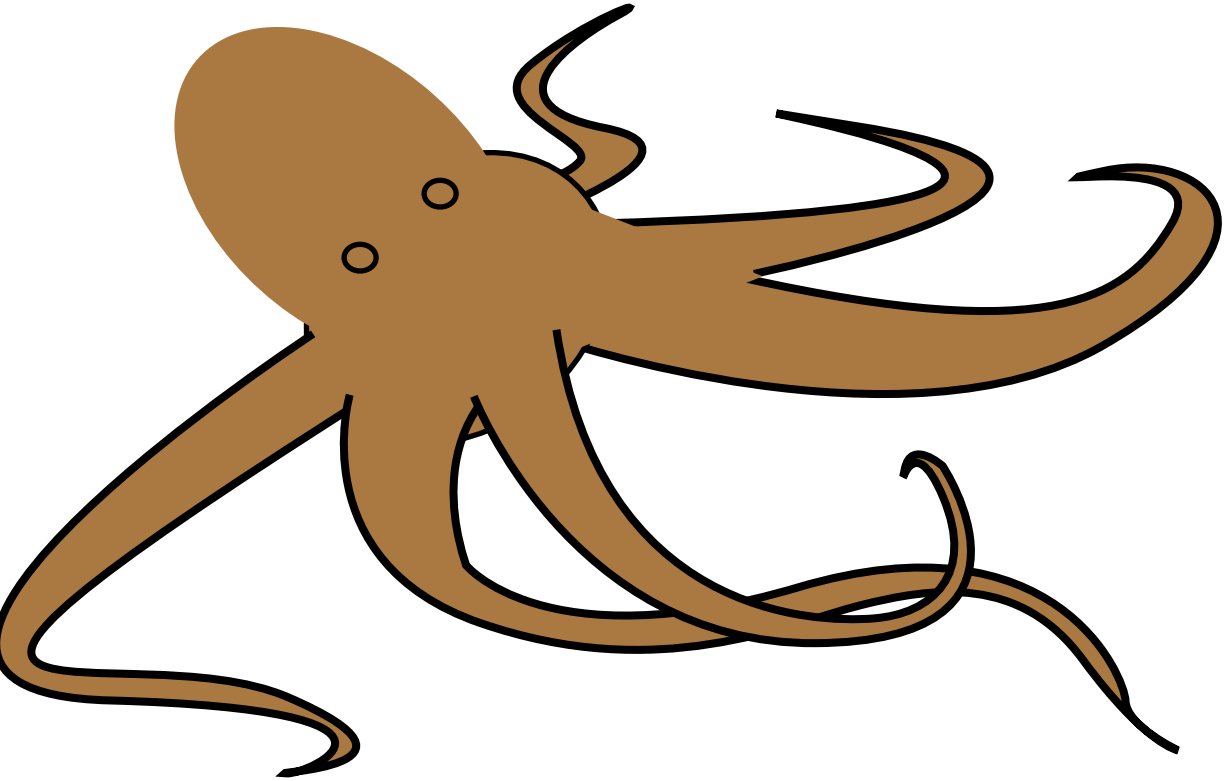
OctopusDB



Log		56	887.9
		89	445.35
		67	234.67
4	lkj	Row	
5	yui	17	612.13
[illegible]	omg	90	148.9

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

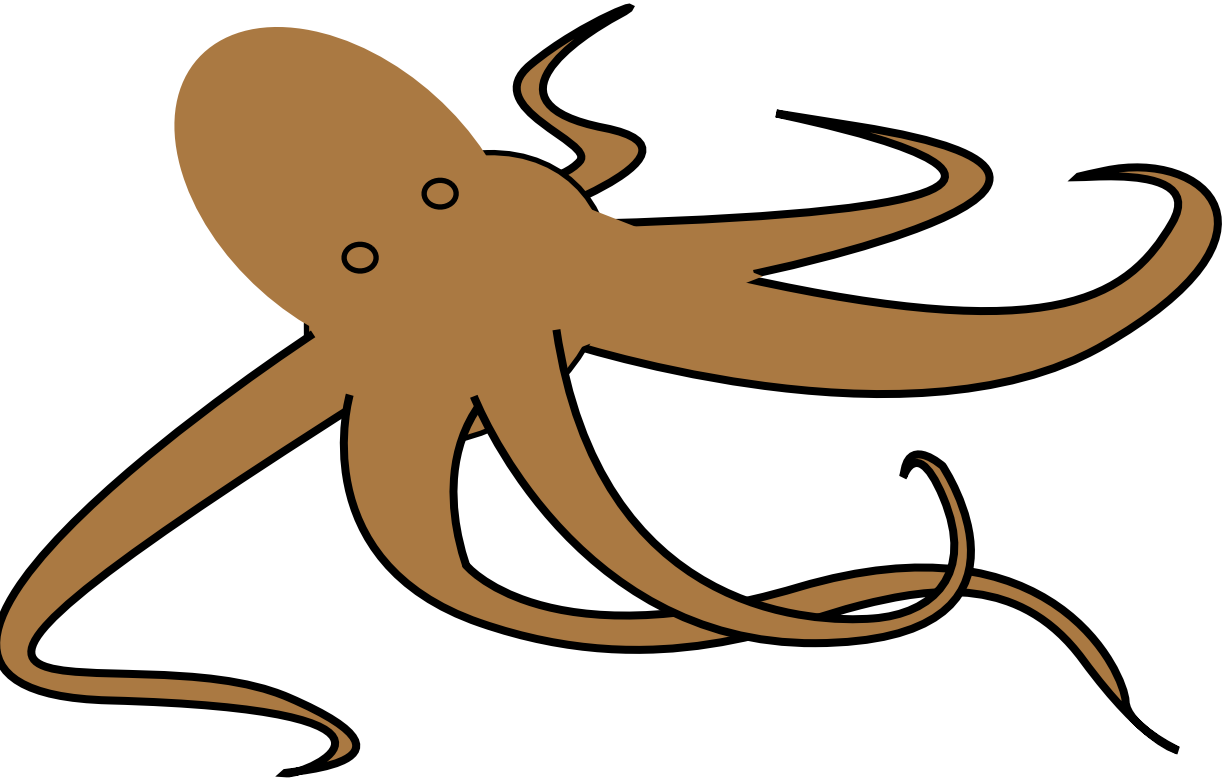
OctopusDB

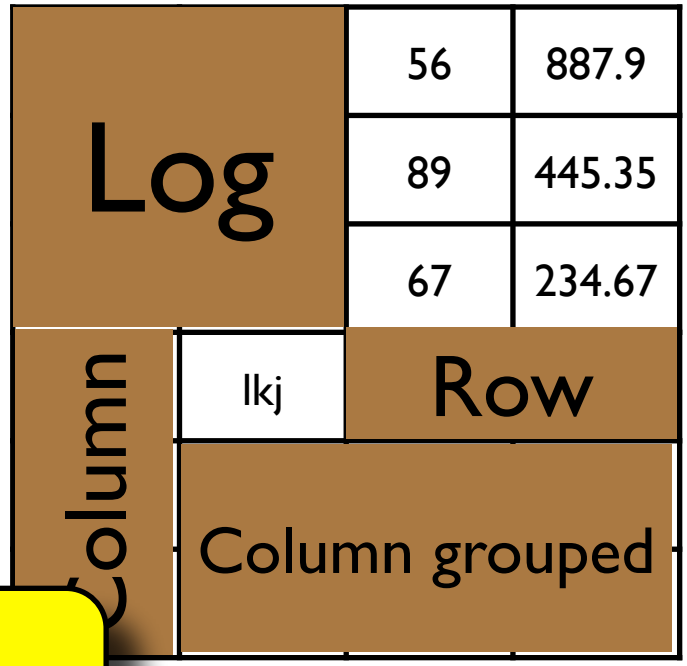


Log		56	887.9
		89	445.35
		67	234.67
Column	lkj	Row	
	yui	17	612.13
	omg	90	148.9

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

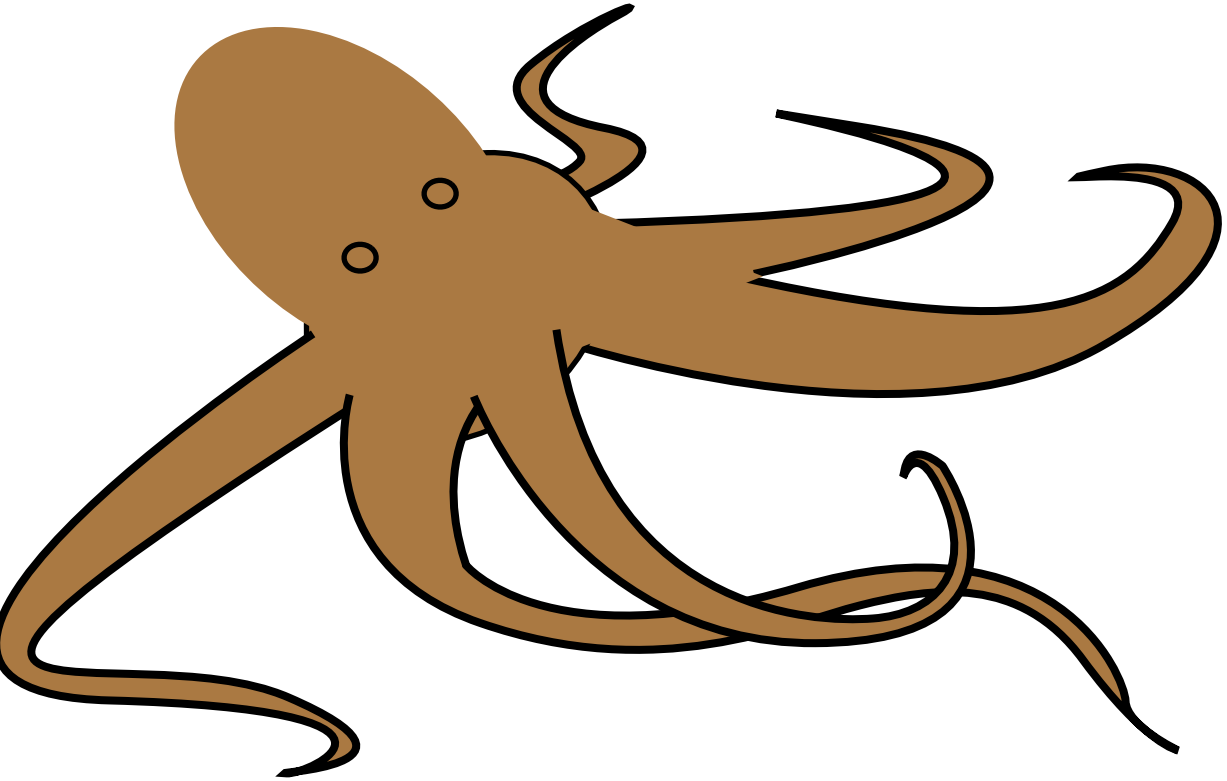
OctopusDB

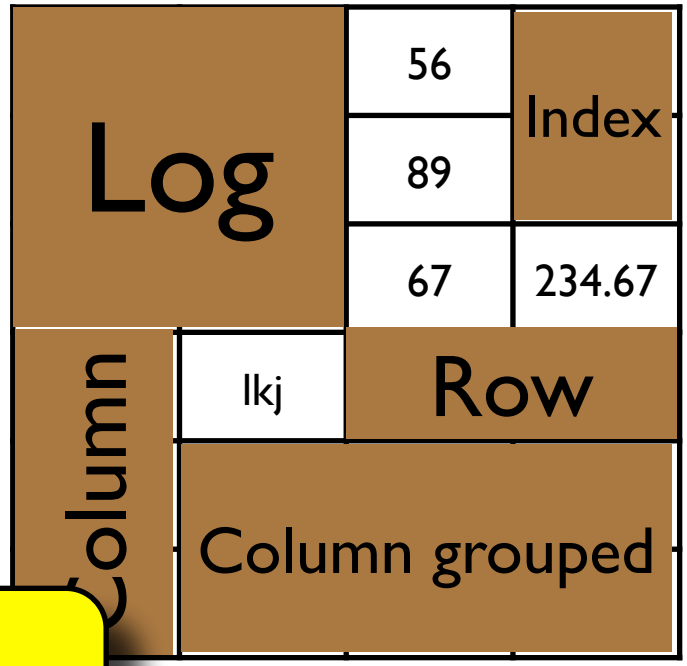




- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

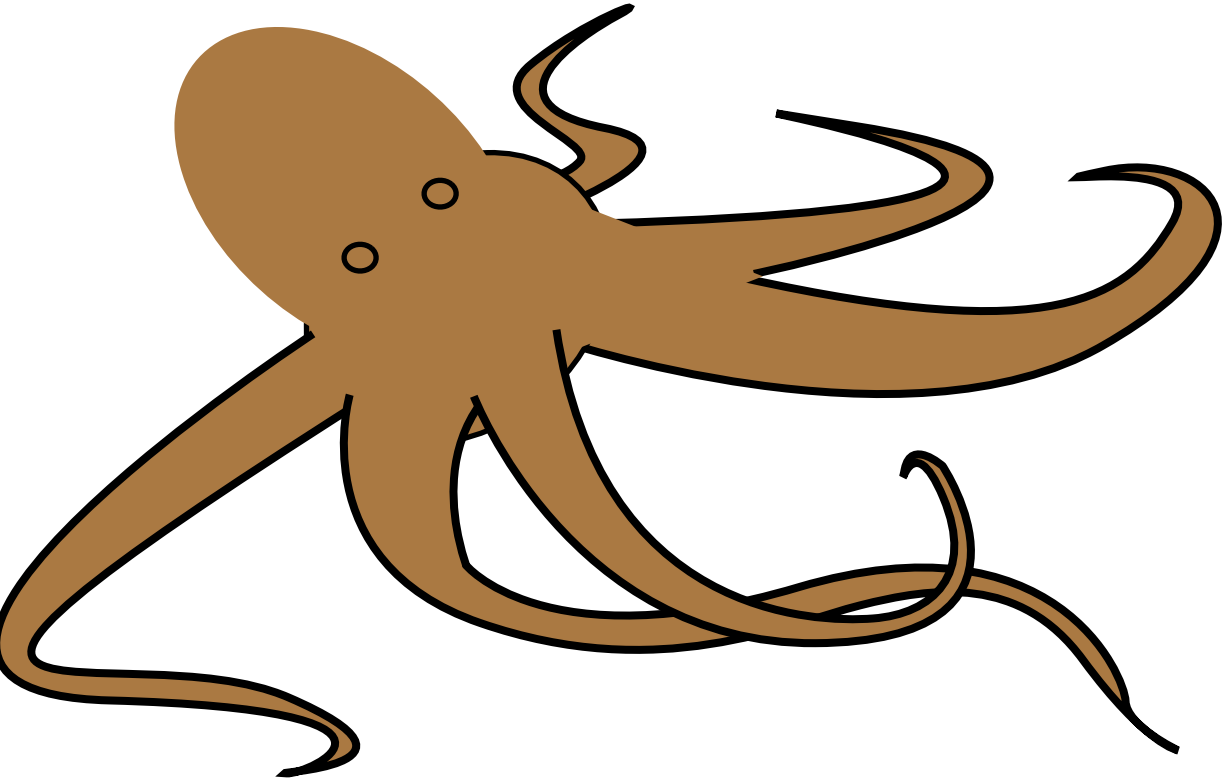
OctopusDB

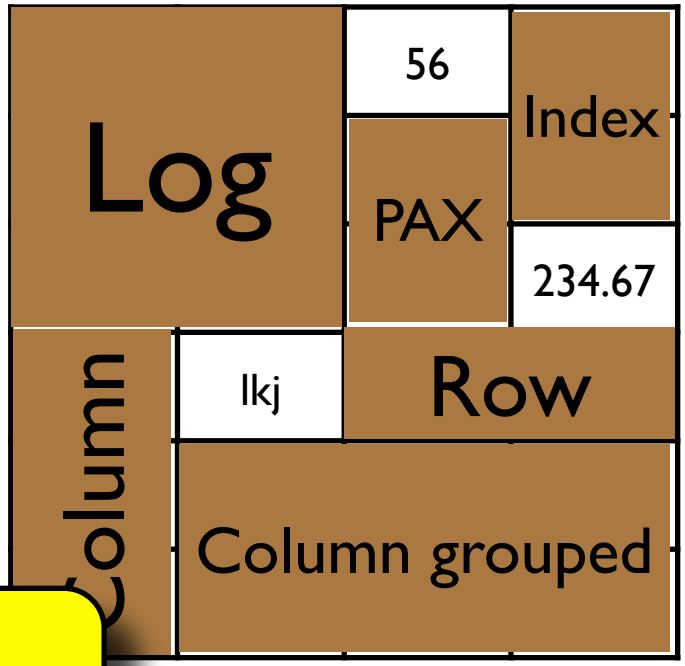




- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

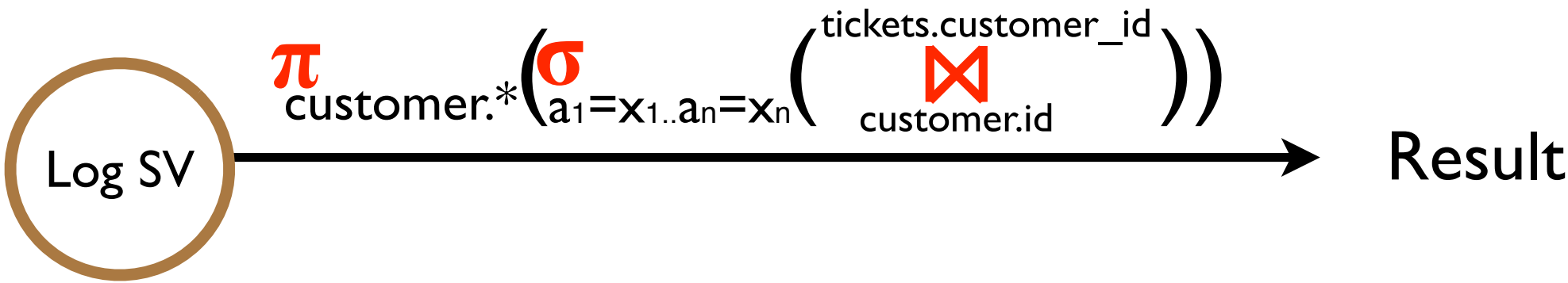
OctopusDB



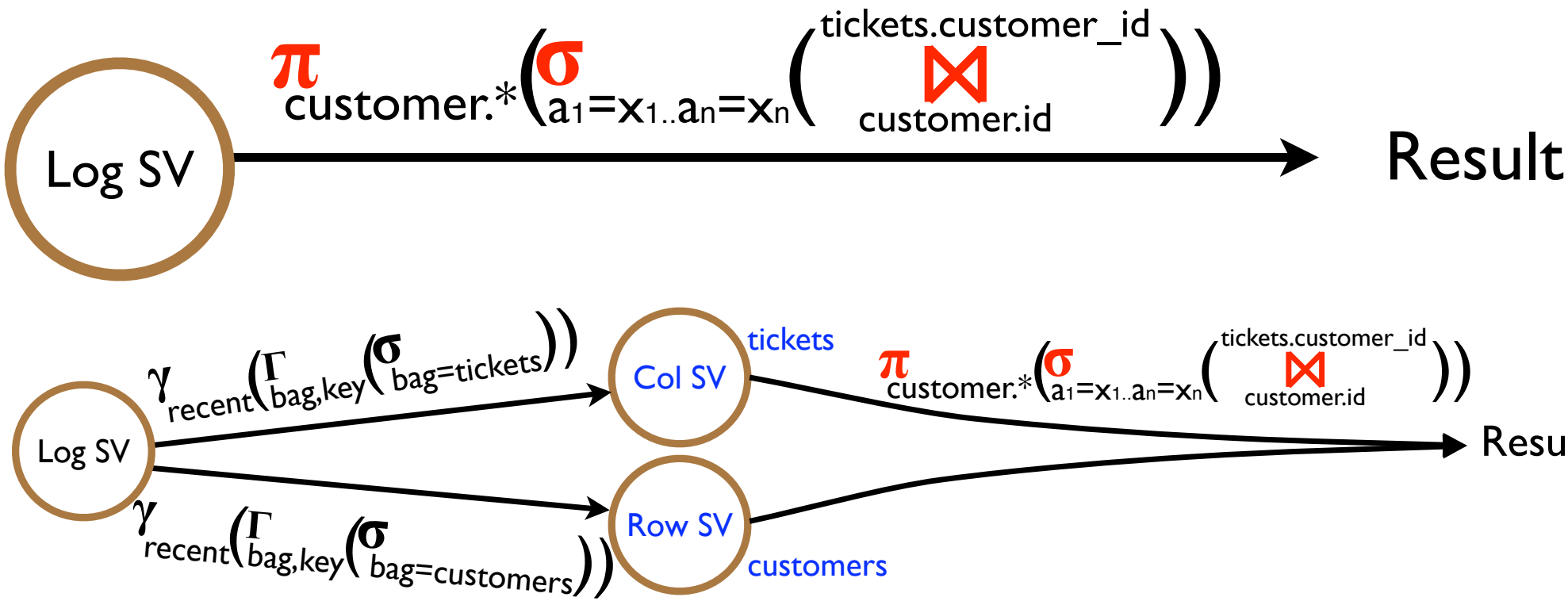


- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views

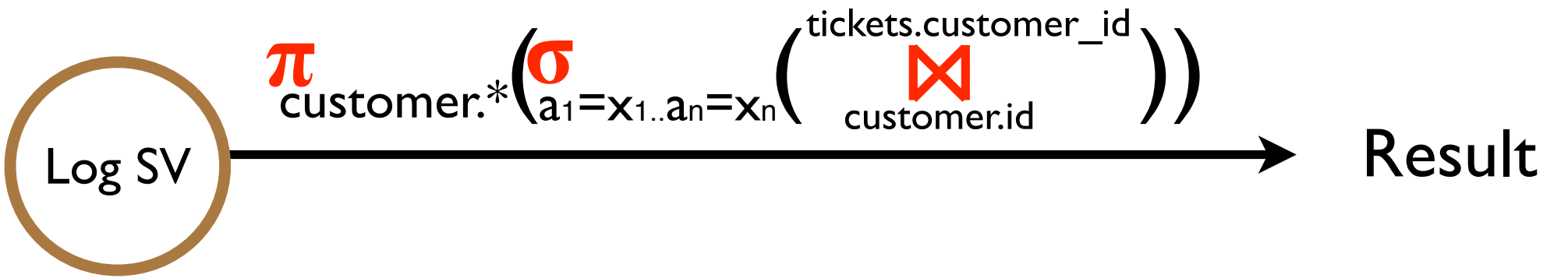
Example: Flight Tickets

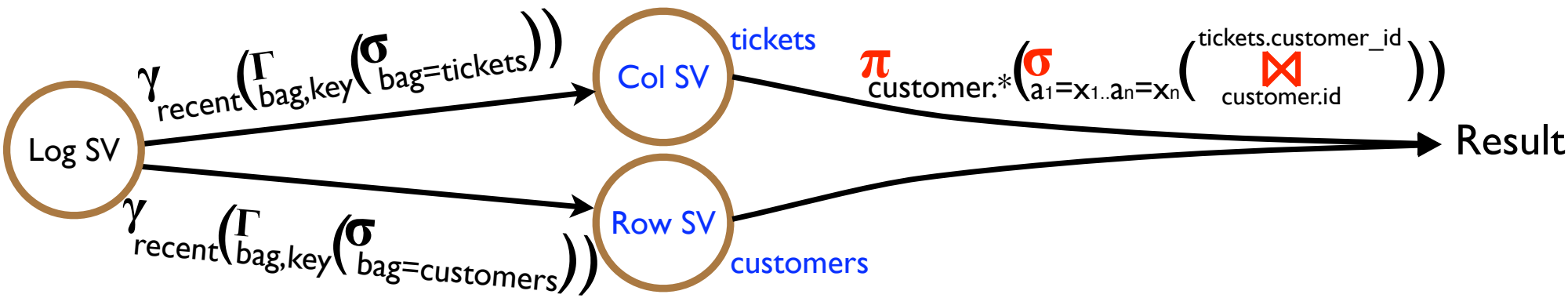


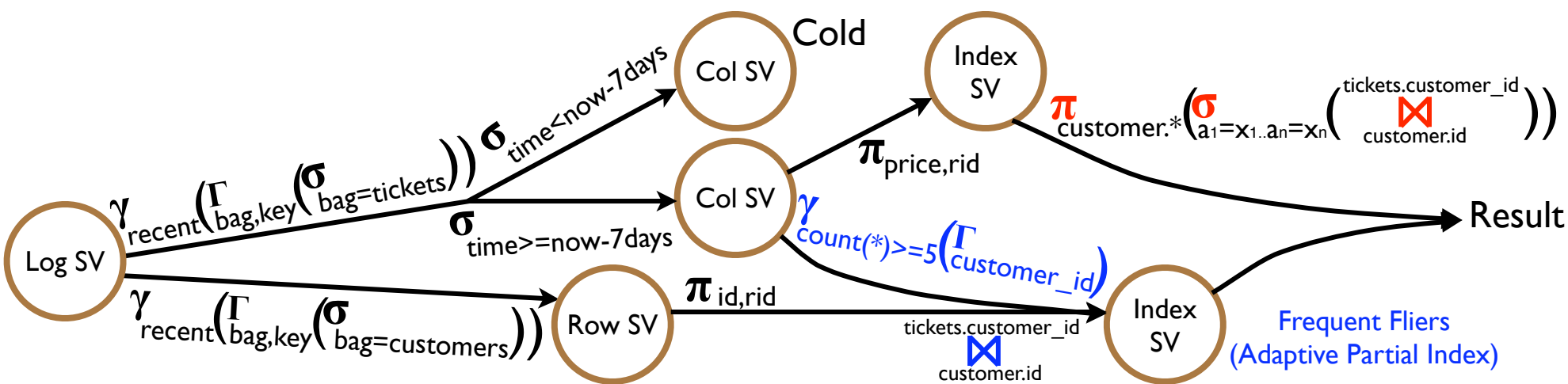
Example: Flight Tickets



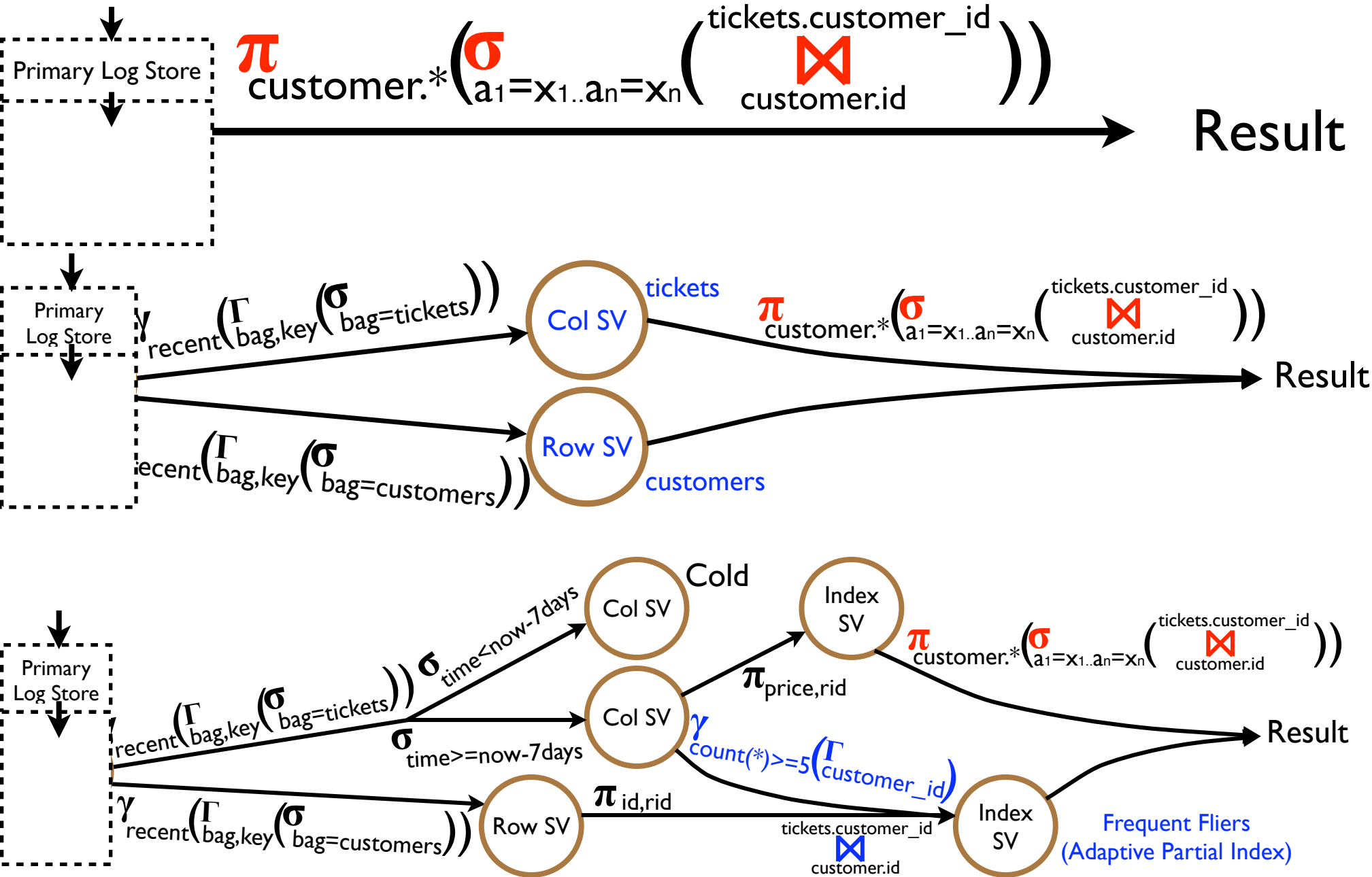
Example: Flight Tickets





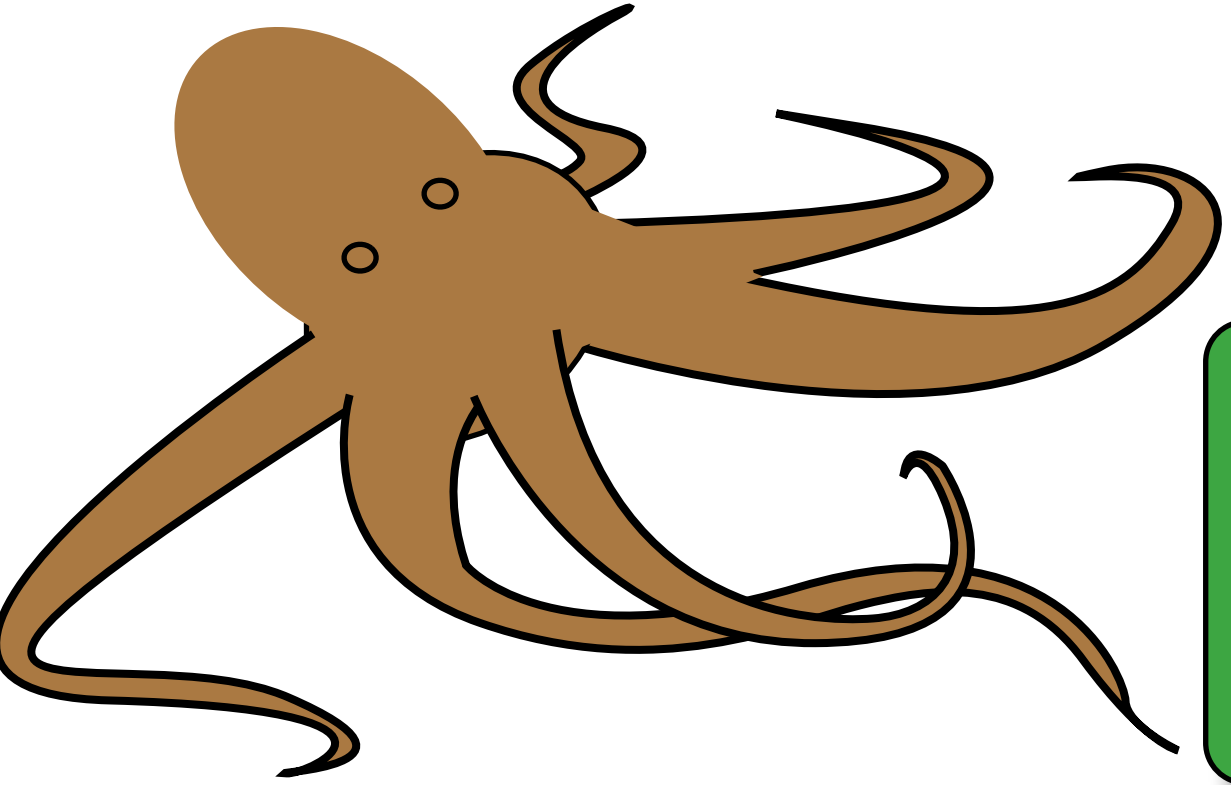


Example: Flight Tickets



Use-Case (traditional systems)	Storage view definition type	example query
row store	Row SV	any
column store	Col SV	any
PAX	PAX SV	any
fractured mirrors	Row SV and Col SV	same query for both
column groups	Row SV and Col SV	$\pi_{a_1,\ldots,a_k}$ $\pi_{a_{k+1},\ldots,a_m}$
index	Index SV	any
indexed row store	Index SV(Row SV)	any
indexed column store	Index SV(Col SV)	any
read-optimized indexed column store + differential write-optimized row store	Index SV(Col SV) Row SV	$\sigma_{t<\text{now}()-1\text{day}}$ $\sigma_{t\geq\text{now}()-1\text{day}}$
partial index	Index SV	$\sigma_{420\leq a_k\leq 42000}$
projection index	Col SV	π_{a_k}
partial projection index	Index SV(Col SV)	$\pi_{a_k}(\sigma_{420\leq a_k\leq 42000})$
DSMS	Index SV	$\sigma_{t\geq\text{now}()-5\text{min}}$
DSMS + archive	Index SV and Col SV	$\sigma_{t\geq\text{now}()-5\text{min}}$ $\sigma_{t<\text{now}()-5\text{min}}$
snapshot	any	any
replicated row store	Row SV Row SV	same query for both
query	any	any
dynamic view	any	any
materialized view	any	any

Use-Case (new system)	**Storage view definition**	
	type	**example query**
OLTP	Row SV	$\sigma_{t \geq \text{now}()-1\text{day}}$
+ OLAP	Col SV	$\sigma_{t < \text{now}()-1\text{day}}$
DSMS	Index SV	$\sigma_{t \geq \text{now}()-5\text{min}}$
+ OLTP	Row SV	$\sigma_{t < \text{now}()-5\text{min}}$
DSMS	Index SV	$\sigma_{t \geq \text{now}()-5\text{min}}$
+ archive OLTP	Row SV	$\sigma_{\text{now}()-1\text{day} \leq t < \text{now}()-5\text{min}}$
+ archive OLAP	Col SV	$\sigma_{t < \text{now}()-1\text{day}}$
other hybrid	any combination of the above	any



OctopusDB

- 'Mimic' several systems
- No 'zoo' overheads
- One-size-fits-all

- Flexible data storage layer
- Adapt layout to workload
- Logical journal of data operations
- Arbitrary physical representations
- New concept: Storage Views


Interesting


19

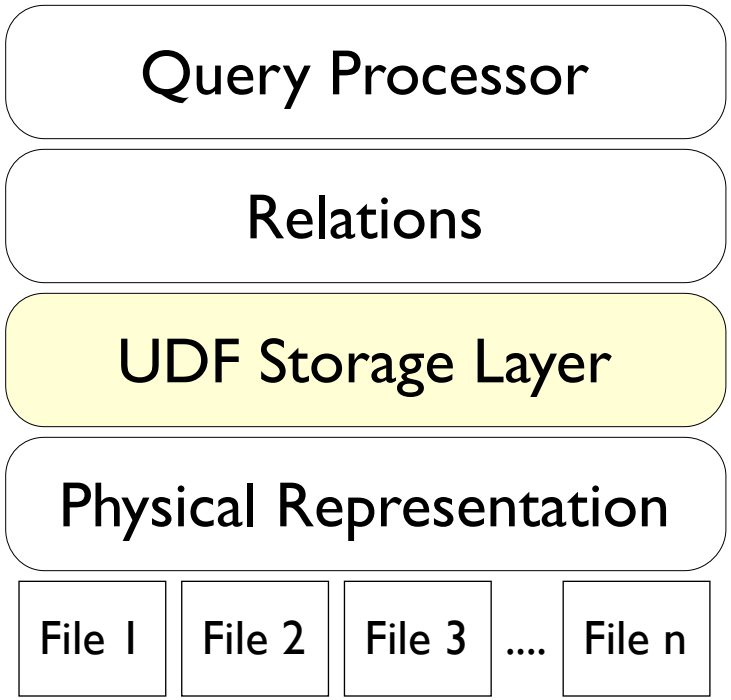
Trojan Techniques

- Good Trojans
- Existing system
- Source-code not required
- Inject additional layouts

How does it work?

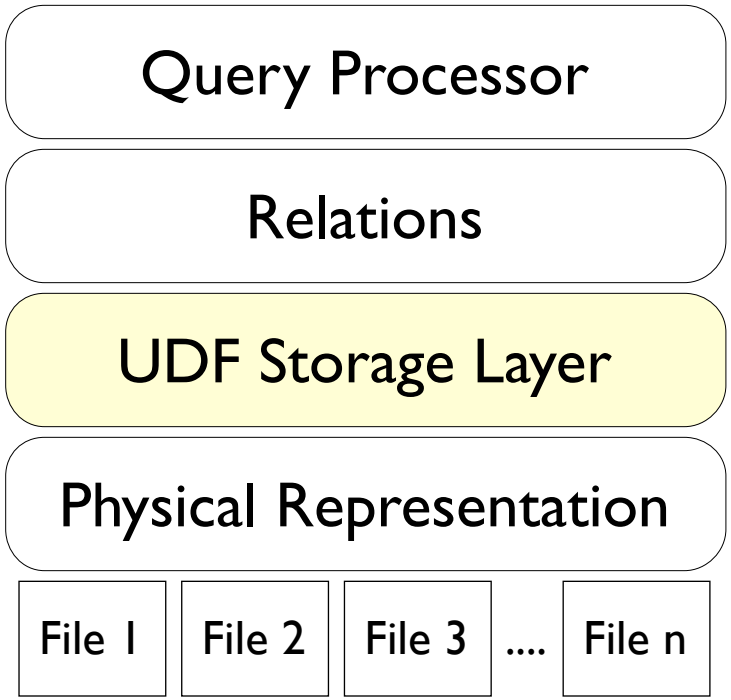
- Exploit UDFs provided by existing systems
- Inject pieces of code
- Hack layouts into the UDFs
- UDF as mapping between logical and physical view of data

How does it work?



- Exploit UDFs provided by existing systems
- Inject pieces of code
- Hack layouts into the UDFs
- UDF as mapping between logical and physical view of data

How does it work?



- Exploit UDFs provided by existing systems
- Inject pieces of code
- Hack layouts into the UDFs
- UDF as mapping between logical and physical view of data
- Novel use of UDFs

Use Case 1:
OLAP in Row-stores

OLTP

OLAP

OLTP

OLAP

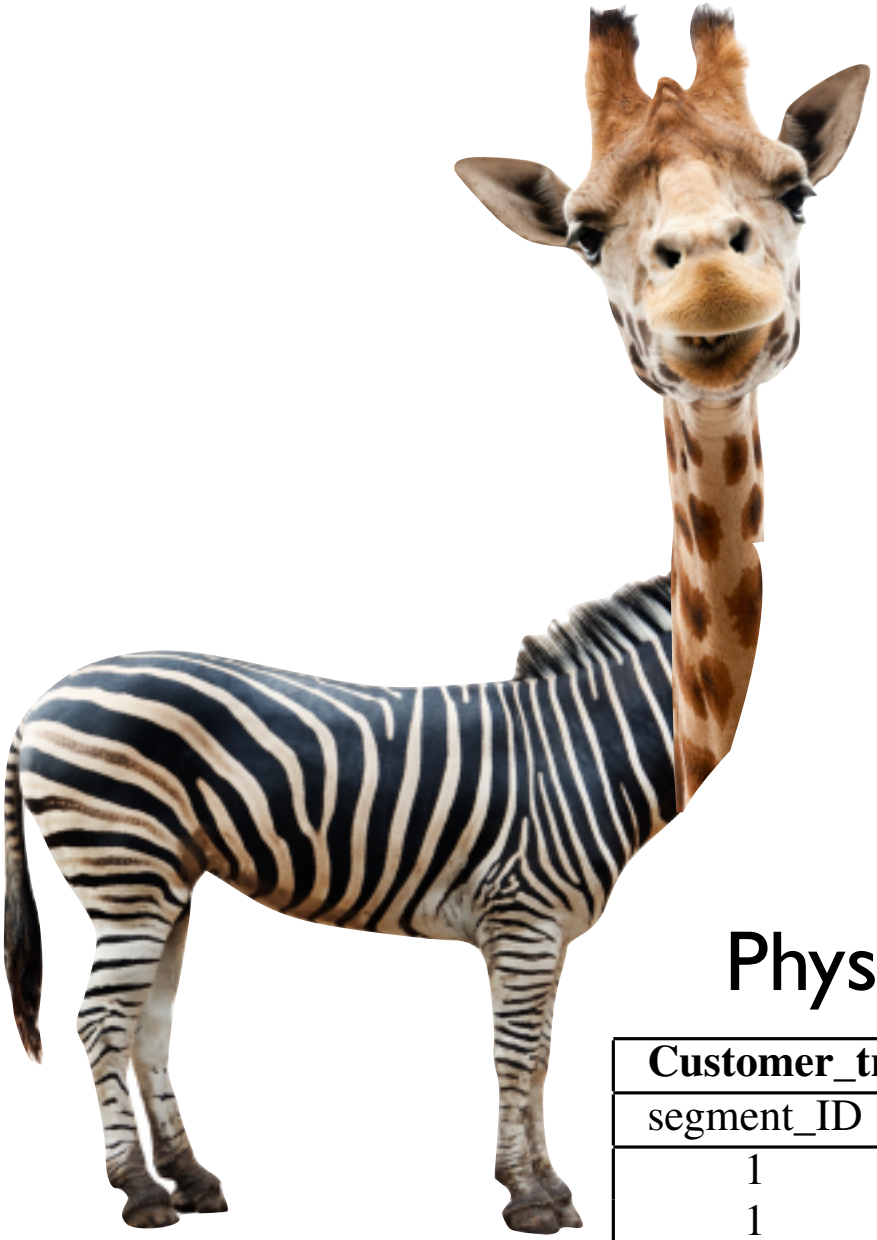
OLTP
OLAP?

- Can we push the limits of row stores?

Trojan Columns

Relation

Customer		
name	phone	market_segment
smith	2134	automobile
john	3425	household
kim	6756	furniture
joe	9878	building
mark	4312	building
steve	2435	automobile
jim	5766	household
ian	8789	household



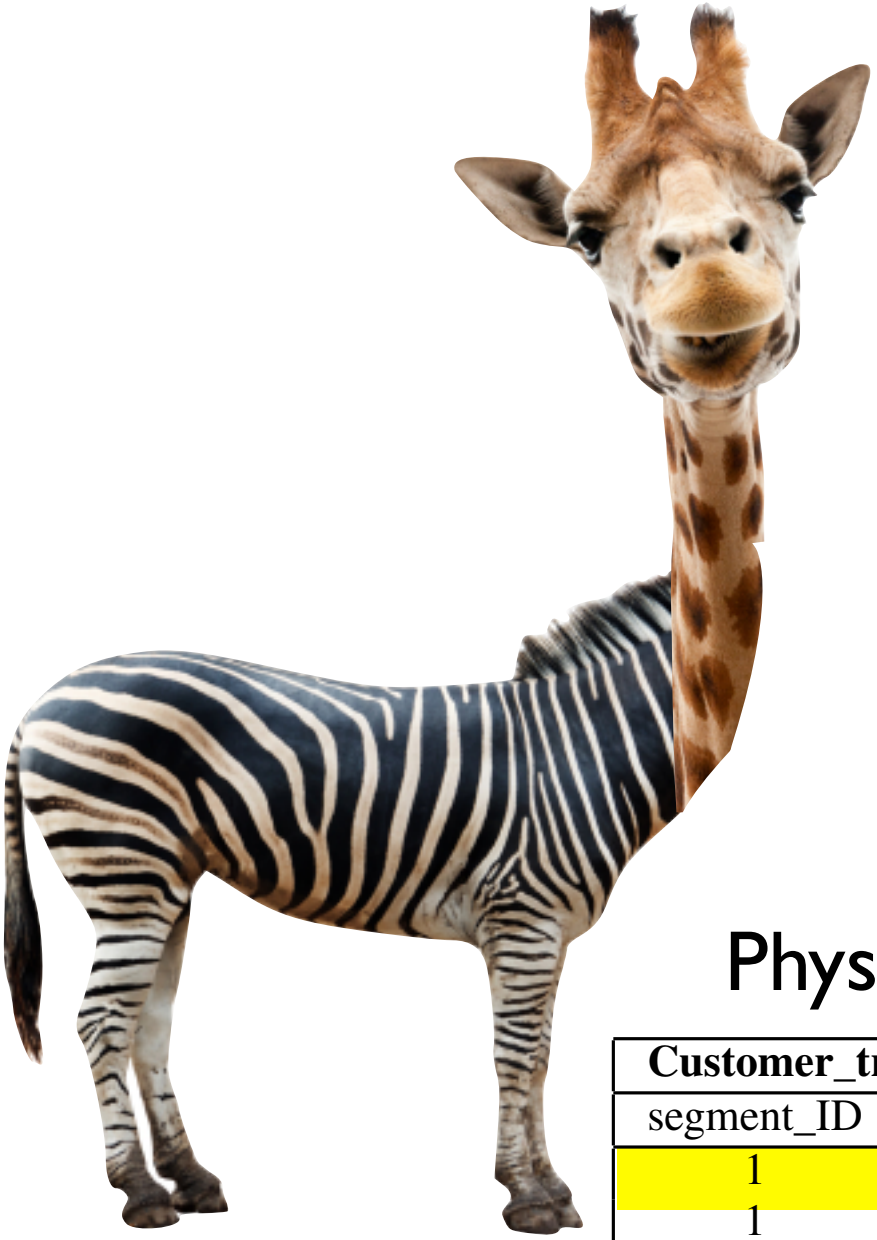
Trojan Columns

Relation

Customer		
name	phone	market_segment
smith	2134	automobile
john	3425	household
kim	6756	furniture
joe	9878	building
mark	4312	building
steve	2435	automobile
jim	5766	household
ian	8789	household

Physical Table

Customer_trojan		
segment_ID	attribute_ID	blob_data
1	name	smith, john, kim, joe
1	phone	2134, 3425, 6756, 9878
1	market_segment	automobile, household, furniture, building
2	name	mark, steve, jim, ian
2	phone	4312, 2435, 5766, 8789
2	market_segment	building, automobile, household, household



Trojan Columns

Relation

Customer		
name	phone	market_segment
smith	2134	automobile
john	3425	household
kim	6756	furniture
joe	9878	building
mark	4312	building
steve	2435	automobile
jim	5766	household
ian	8789	household

Physical Table

Customer_trojan		
segment_ID	attribute_ID	blob_data
1	name	smith, john, kim, joe
1	phone	2134, 3425, 6756, 9878
1	market_segment	automobile, household, furniture, building
2	name	mark, steve, jim, ian
2	phone	4312, 2435, 5766, 8789
2	market_segment	building, automobile, household, household

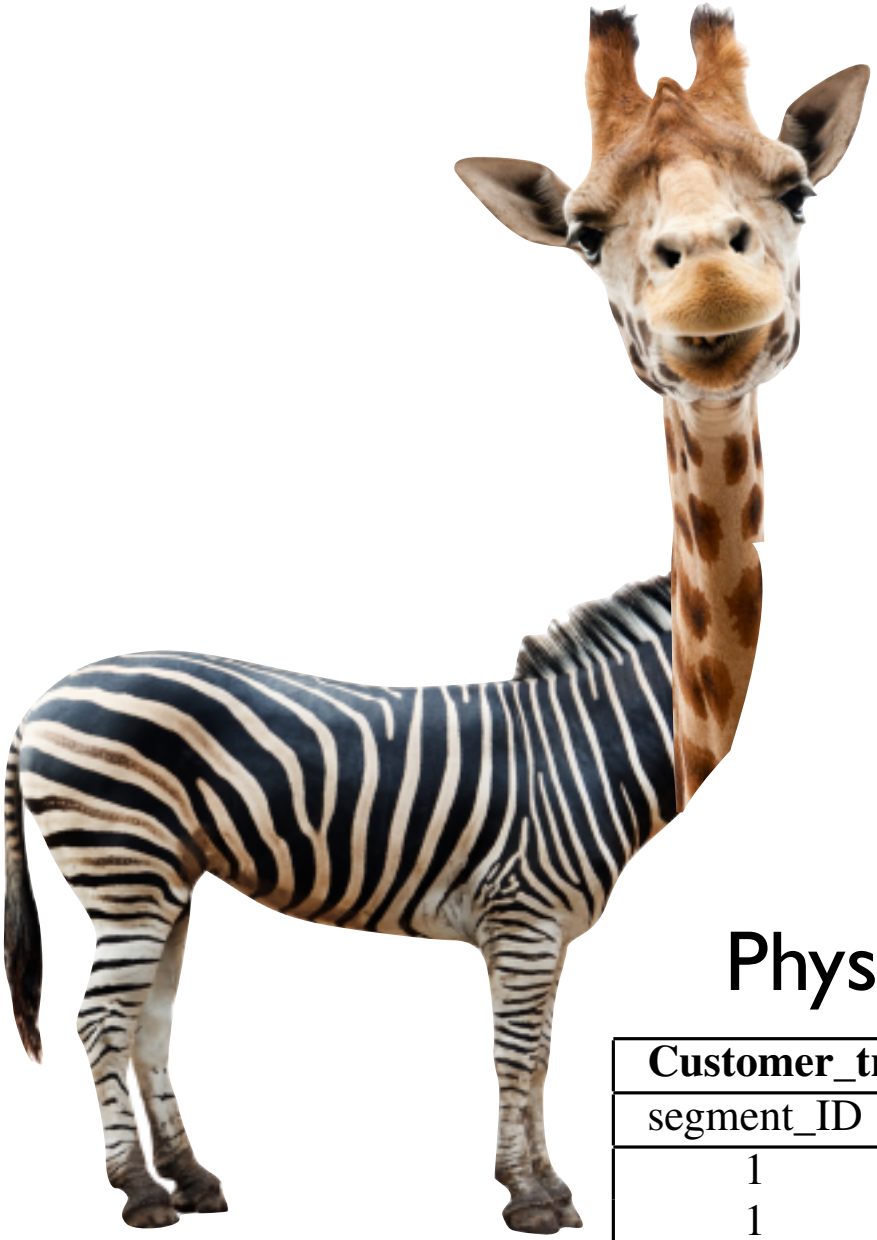
Trojan Columns

Relation

Customer		
name	phone	market_segment
smith	2134	automobile
john	3425	household
kim	6756	furniture
joe	9878	building
mark	4312	building
steve	2435	automobile
jim	5766	household
ian	8789	household

Physical Table

Customer_trojan		
segment_ID	attribute_ID	blob_data
1	name	smith, john, kim, joe
1	phone	2134, 3425, 6756, 9878
1	market_segment	automobile, household, furniture, building
2	name	mark, steve, jim, ian
2	phone	4312, 2435, 5766, 8789
2	market_segment	building, automobile, household, household



Trojan Columns

Relation

Customer		
name	phone	market_segment
smith	2134	automobile
john	3425	household
kim	6756	furniture
joe	9878	building
mark	4312	building
steve	2435	automobile
jim	5766	household
ian	8789	household

Physical Table

Customer_trojan		
segment_ID	attribute_ID	blob_data
1	name	smith, john, kim, joe
1	phone	2134, 3425, 6756, 9878
1	market_segment	automobile, household, furniture, building
2	name	mark, steve, jim, ian
2	phone	4312, 2435, 5766, 8789
2	market_segment	building, automobile, household, household

Example: TPC-H Query 6

Result

↑

γ_{agg} (extendedprice * discount)

↑

σ shipdate BETWEEN
'1994-01-01' AND '1995-01-01'
AND discount BETWEEN
0.05 AND 0.07
AND quantity < 24

↑

π quantity, discount
extendedprice, shipdate

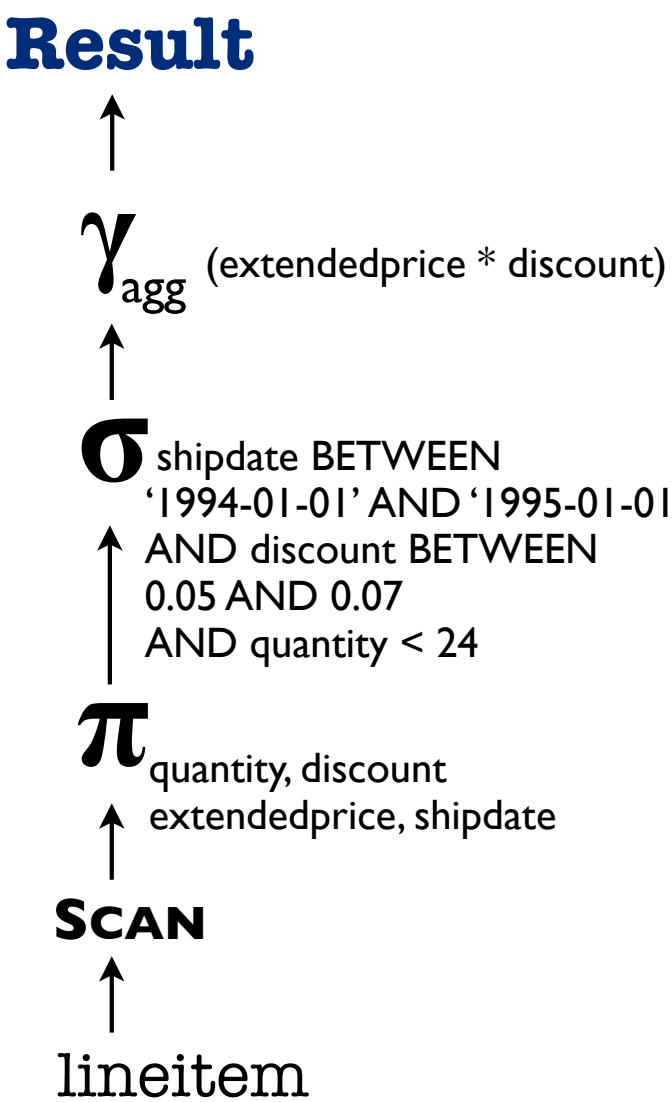
↑

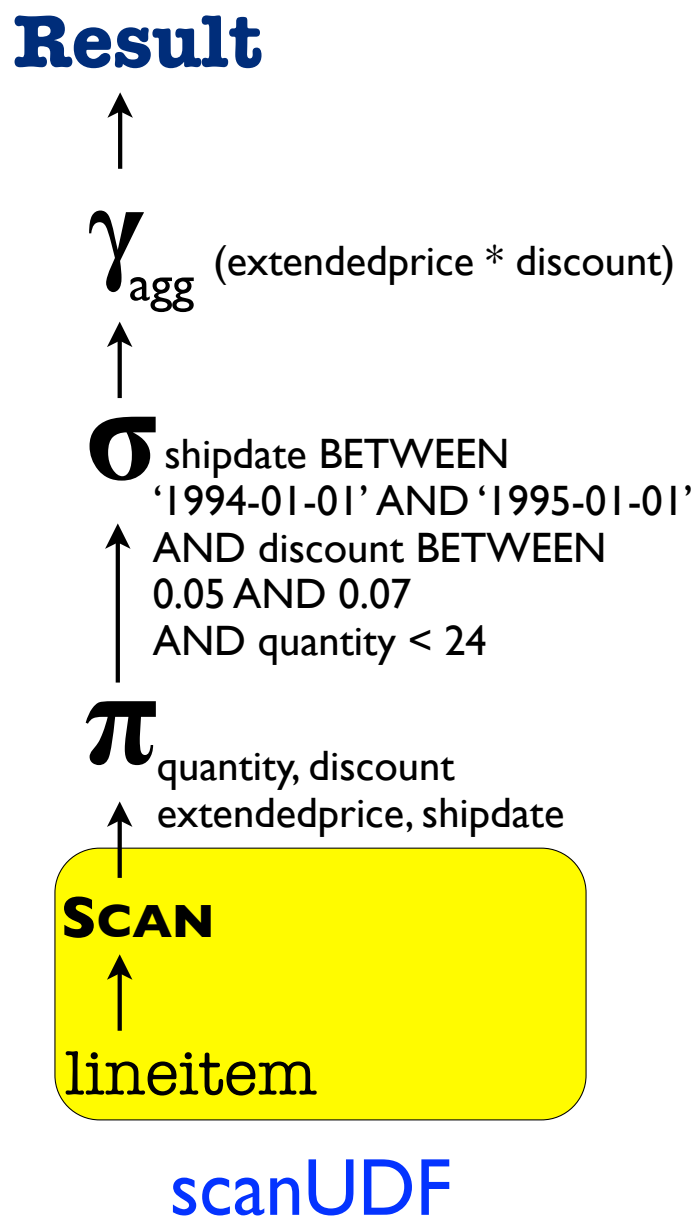
SCAN

↑

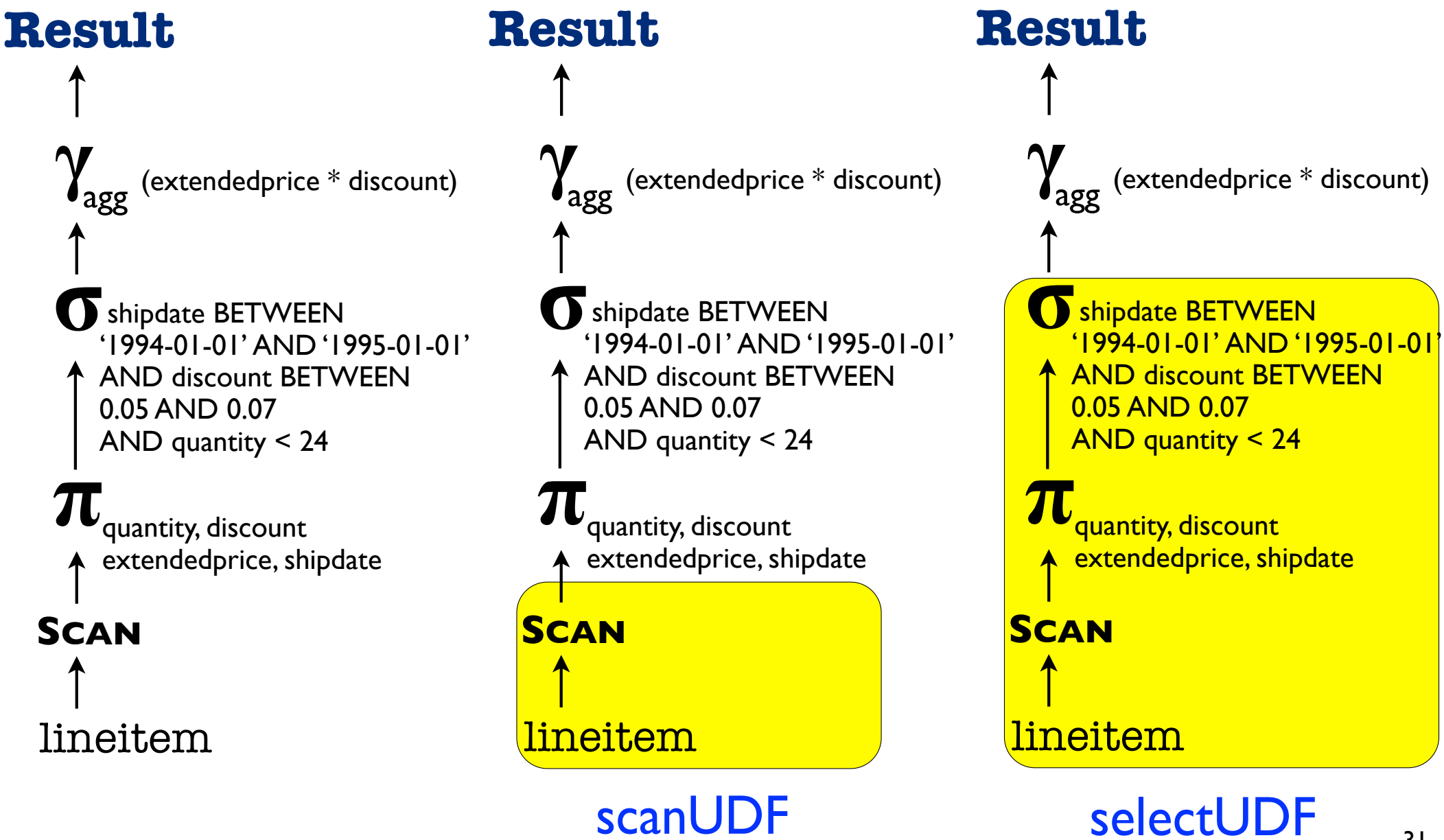
lineitem

Example: TPC-H Query 6

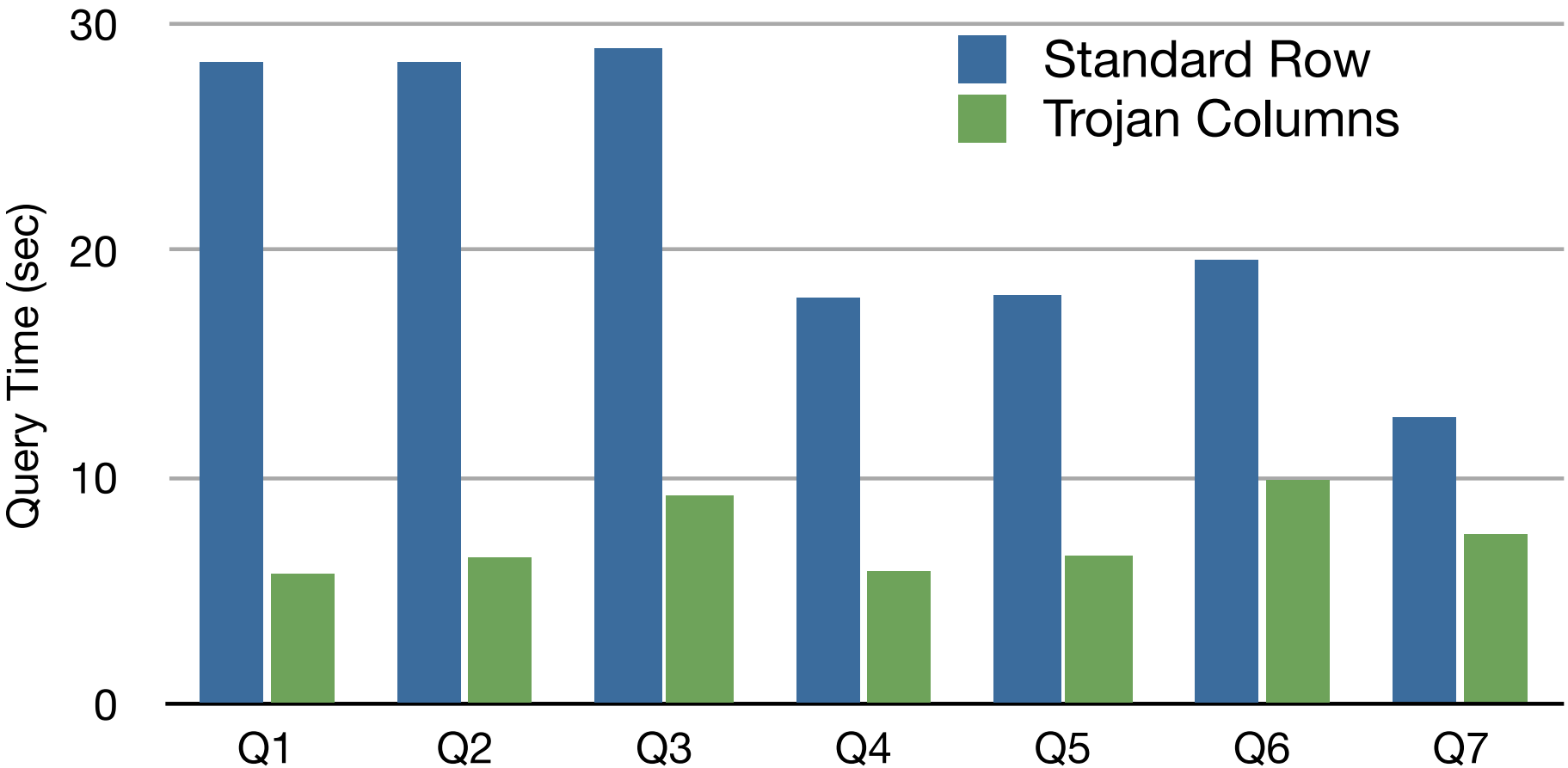




Example: TPC-H Query 6

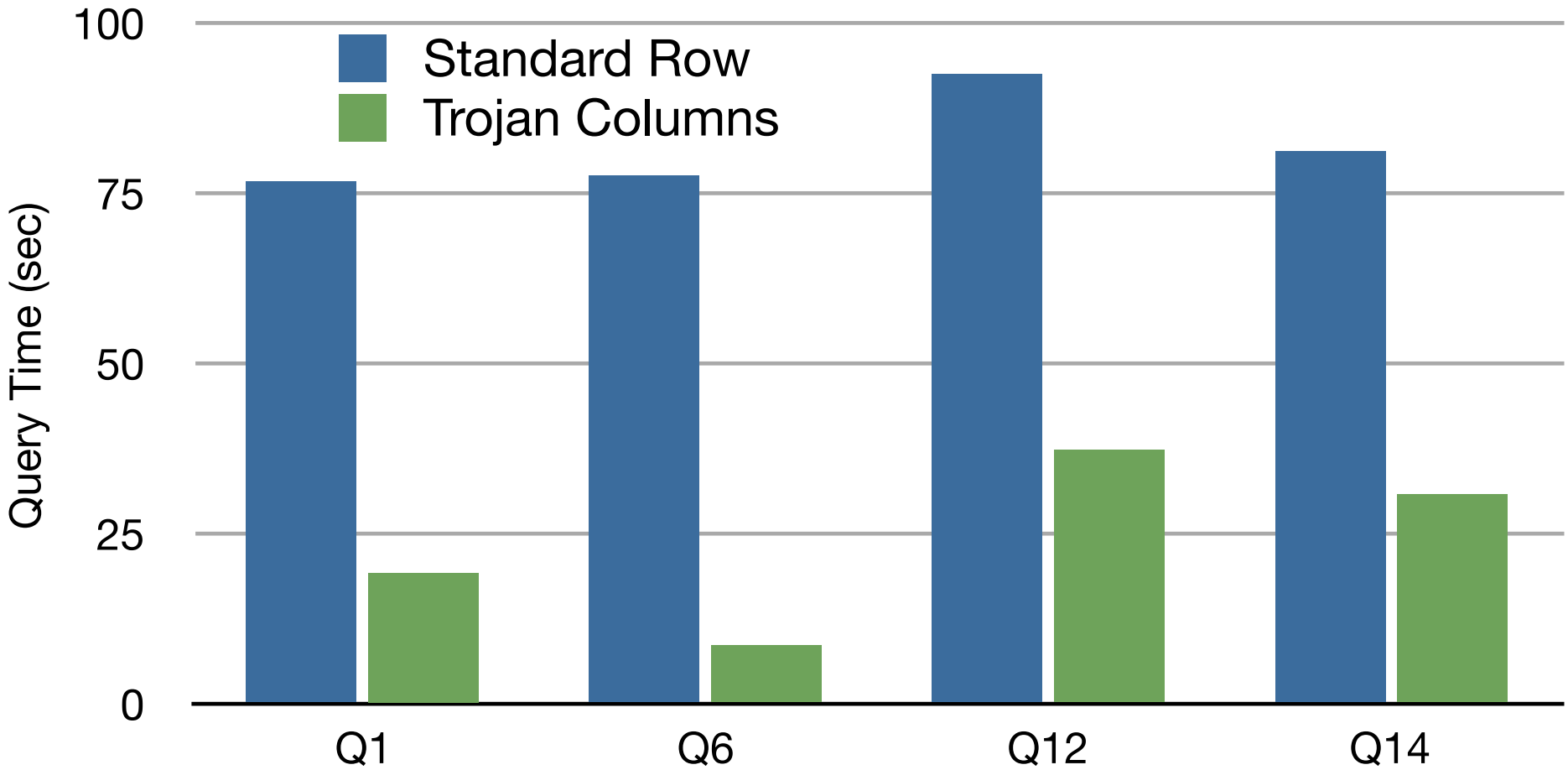


C-Store Benchmark *



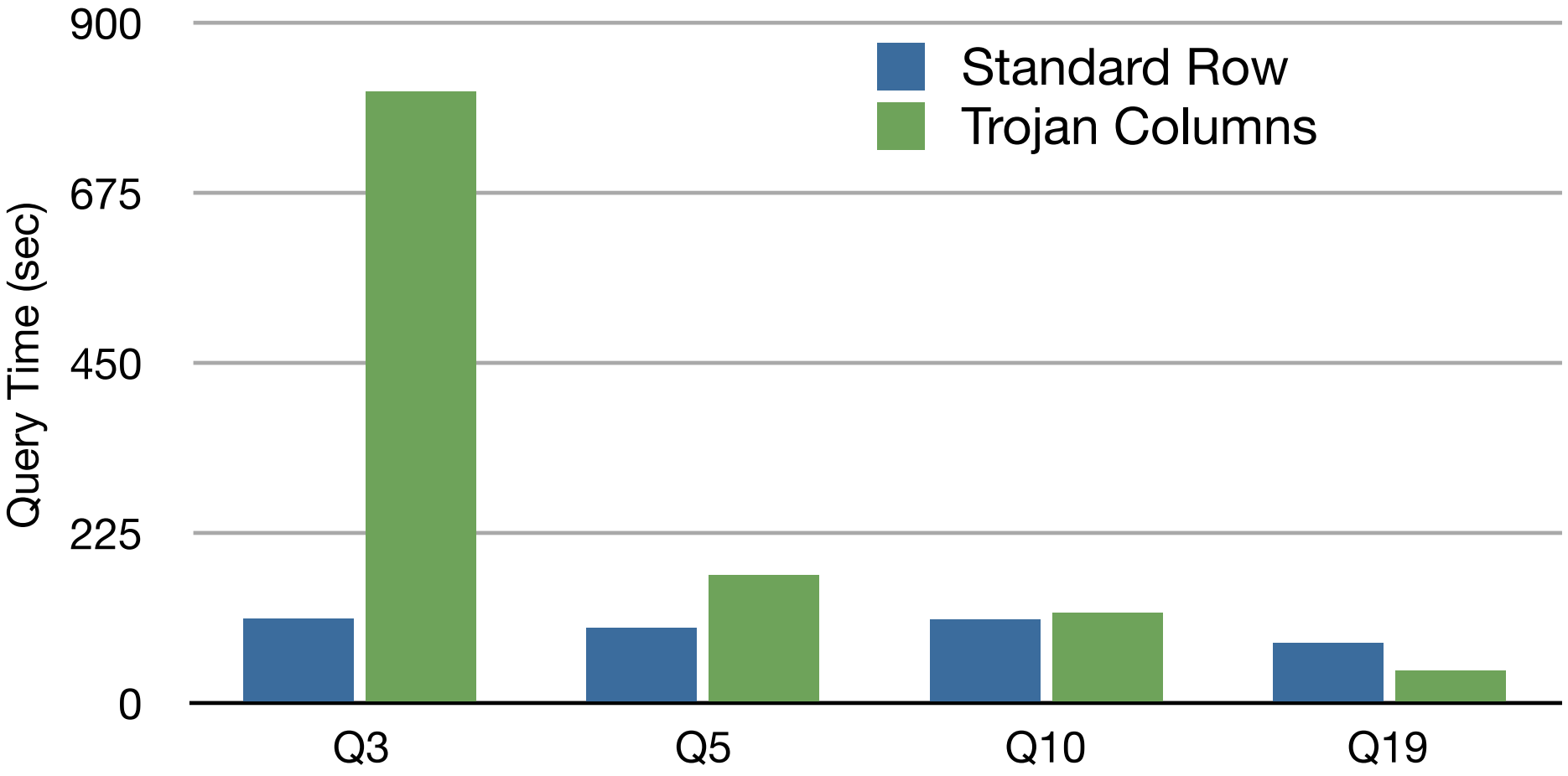
* Mike Stonebraker et. al. C-Store: A Column Oriented DBMS. VLDB 2005

TPC-H Benchmark *



* tpch.org/tpch

TPC-H Benchmark *



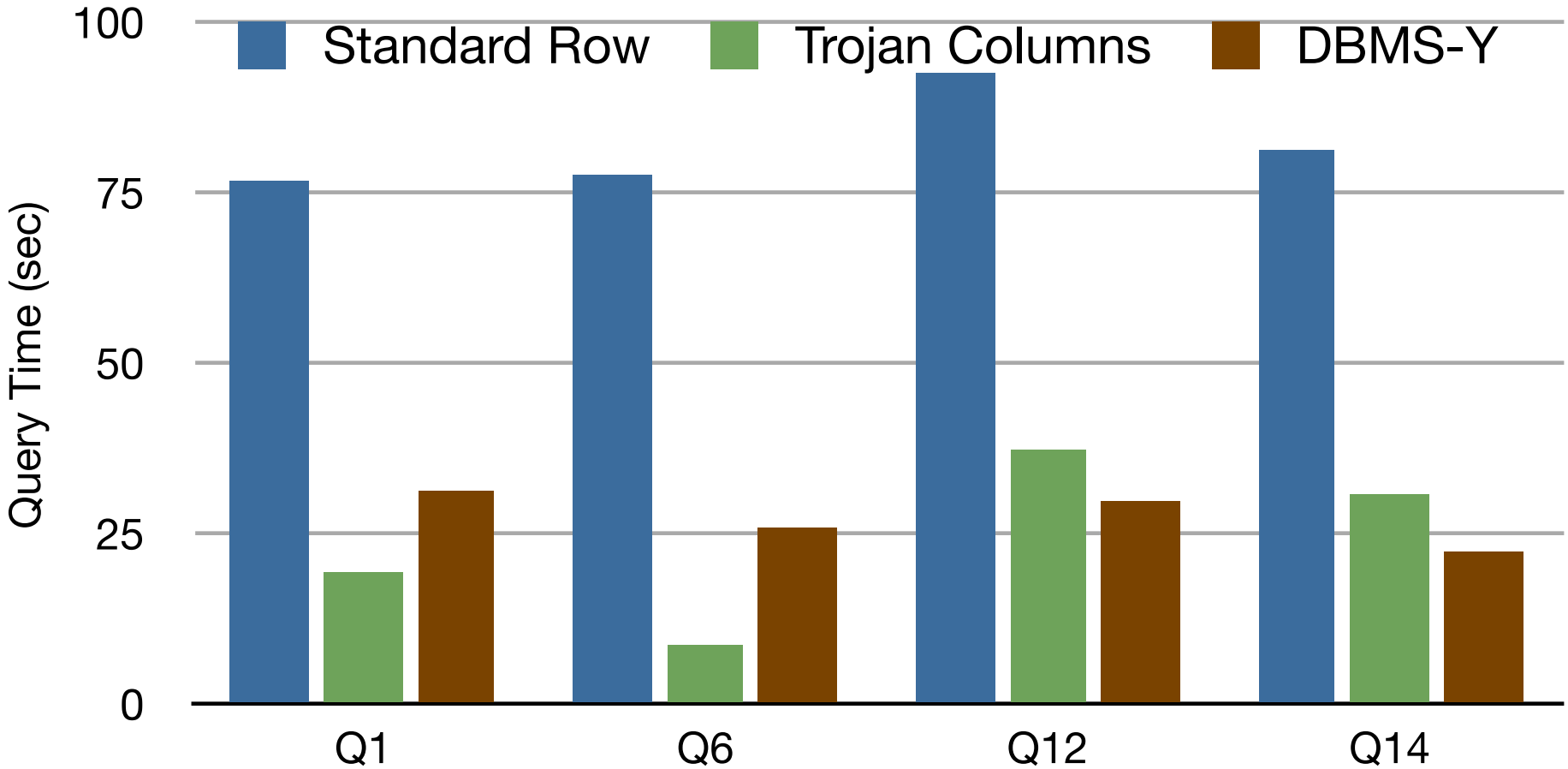
* tpch.org/tpch

Micro-Benchmark

# referenced attributes (r)	1E-06	1E-05	1E-04	1E-03	1E-02	1E-01	1E+00
16	2.13	2.13	2.11	2.06	1.55	0.47	0.06
15	4.64	4.62	4.55	4.27	2.57	0.55	0.06
13	5.00	5.00	4.94	4.61	2.70	0.57	0.06
11	5.79	5.82	5.75	5.24	2.87	0.56	0.06
9	6.39	6.38	6.25	5.79	3.11	0.54	0.06
7	7.00	6.96	6.80	6.23	3.17	0.56	0.06
5	10.96	10.94	10.55	9.27	3.75	0.57	0.06
3	12.86	13.57	13.22	11.03	4.16	0.56	0.06
1	17.43	17.61	16.61	13.57	4.39	0.57	0.06

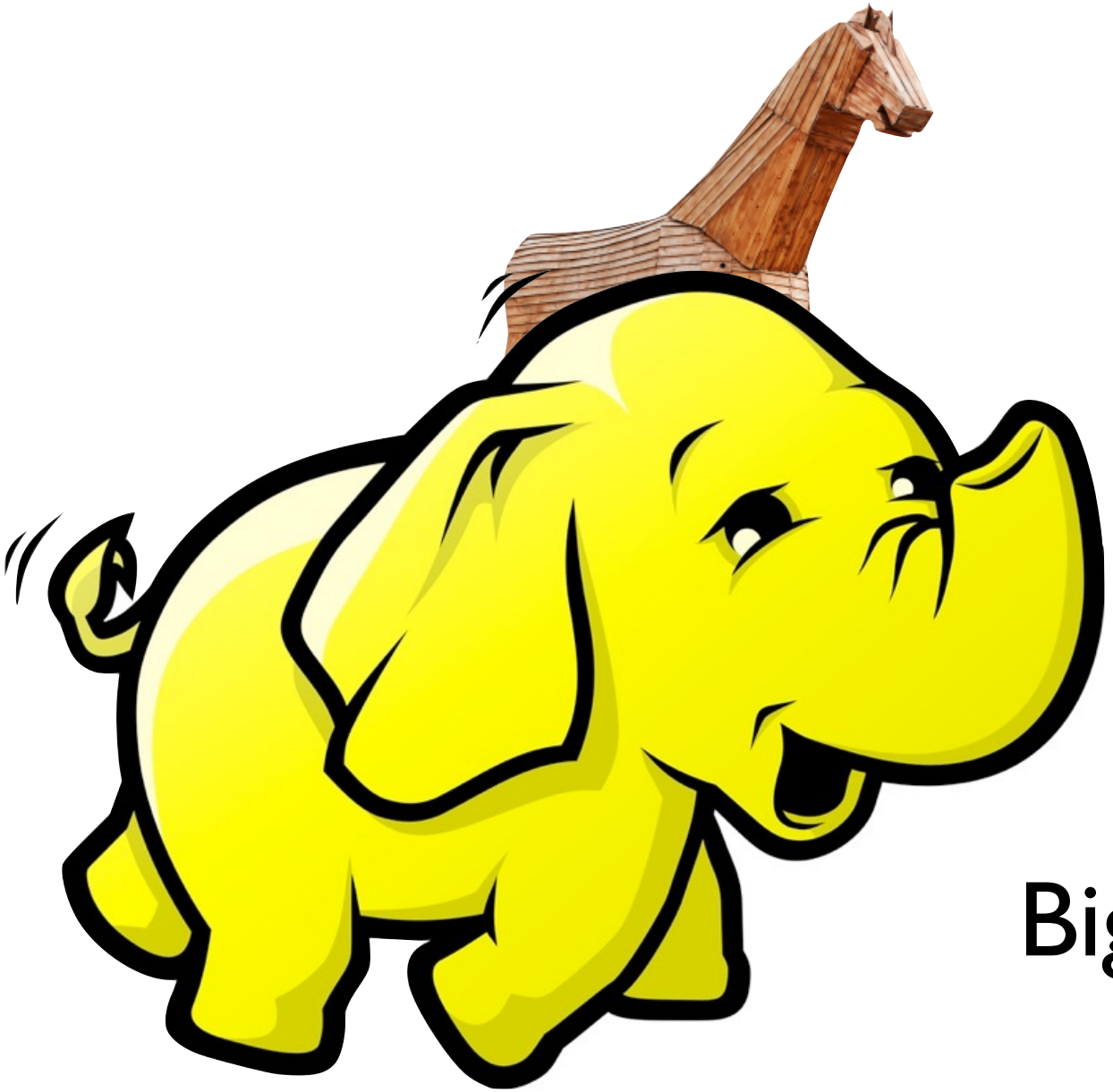
selectivity (fraction of tuples accessed)

versus Column Stores



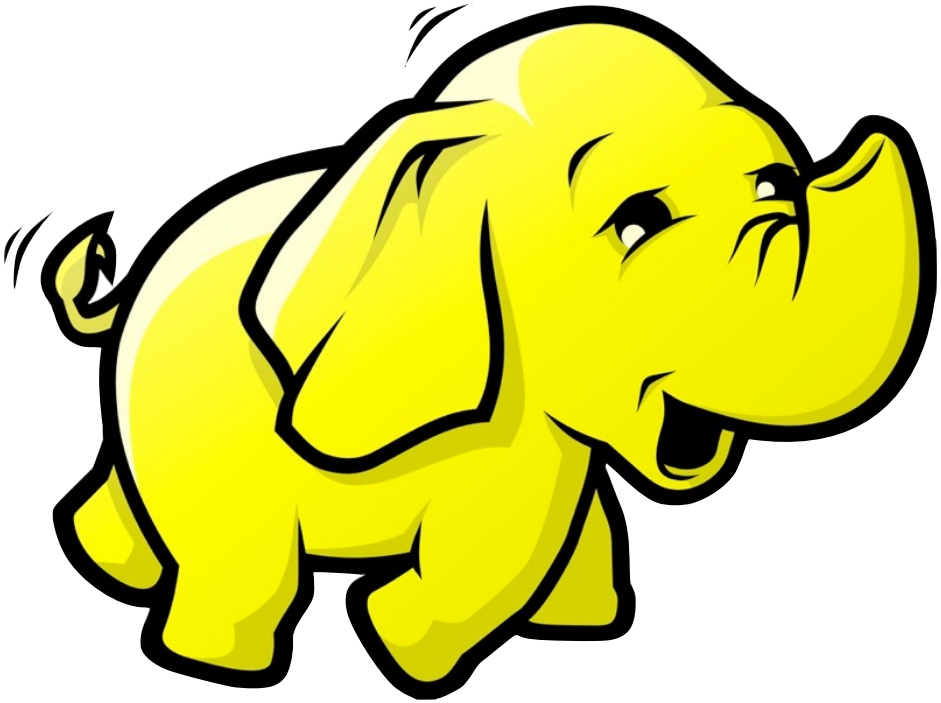
Trojan Columns Advantages

- Column + Row storage
- Much better performance
- Closed source system



Use Case 2:
Big Data Analytics

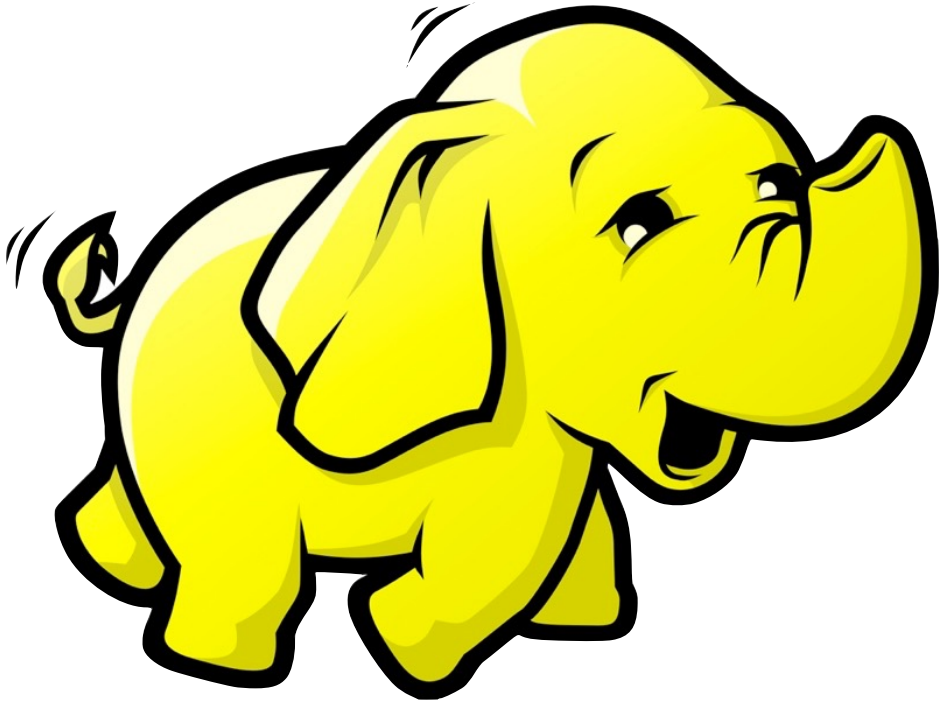
Web-log Processing



- Scan Tasks

User visits from different countries

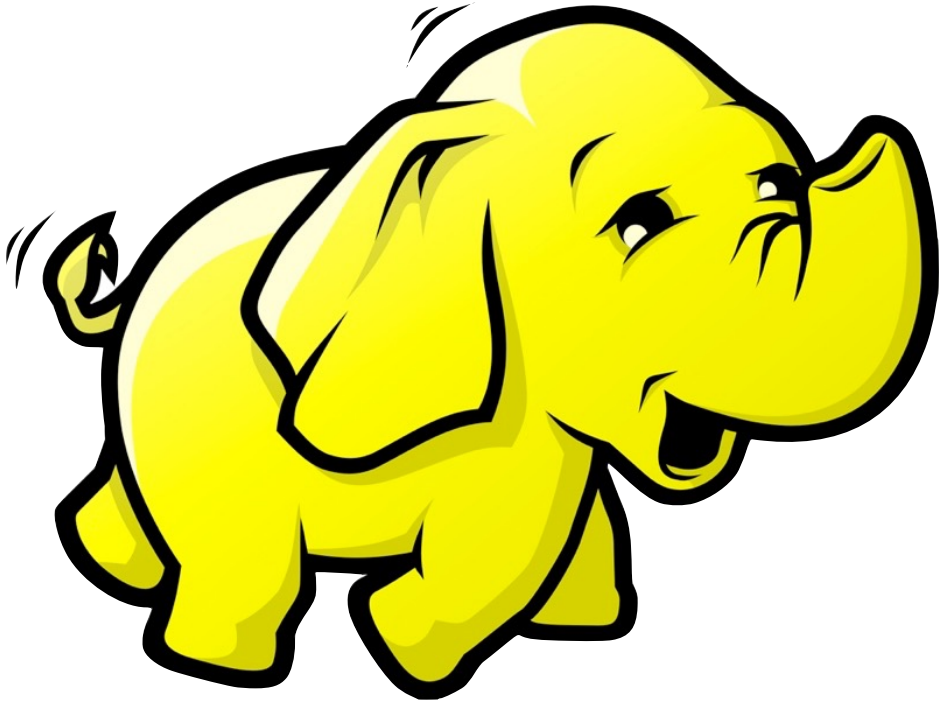
Web-log Processing



- Scan Tasks
- Selection Tasks

User visits with duration greater than 10s

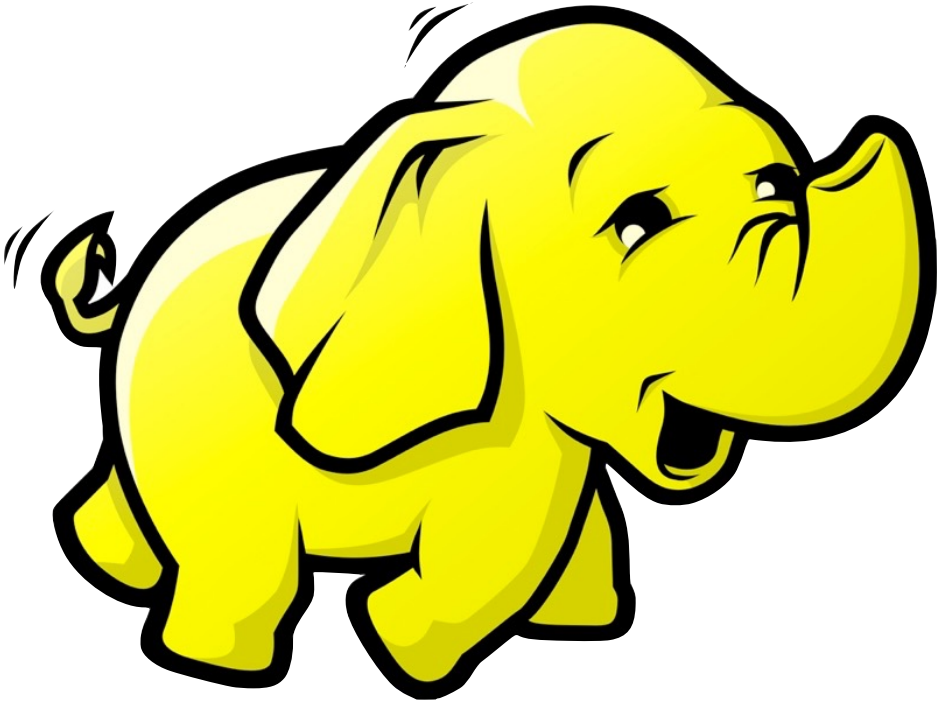
Web-log Processing



- Scan Tasks
- Selection Tasks
- Projection Tasks

URL and duration of each user visit

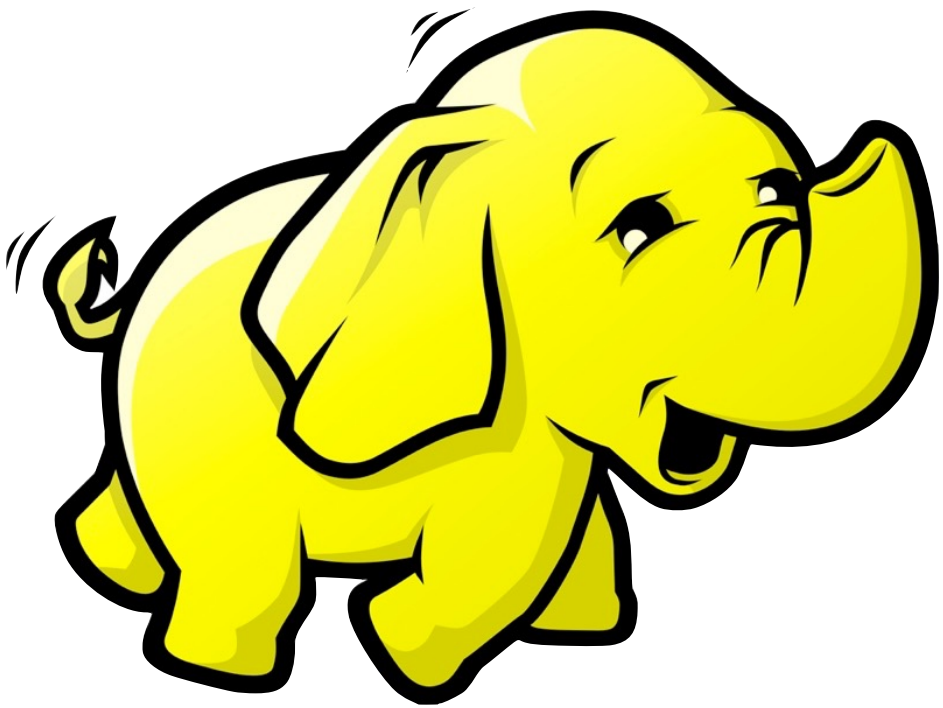
Web-log Processing



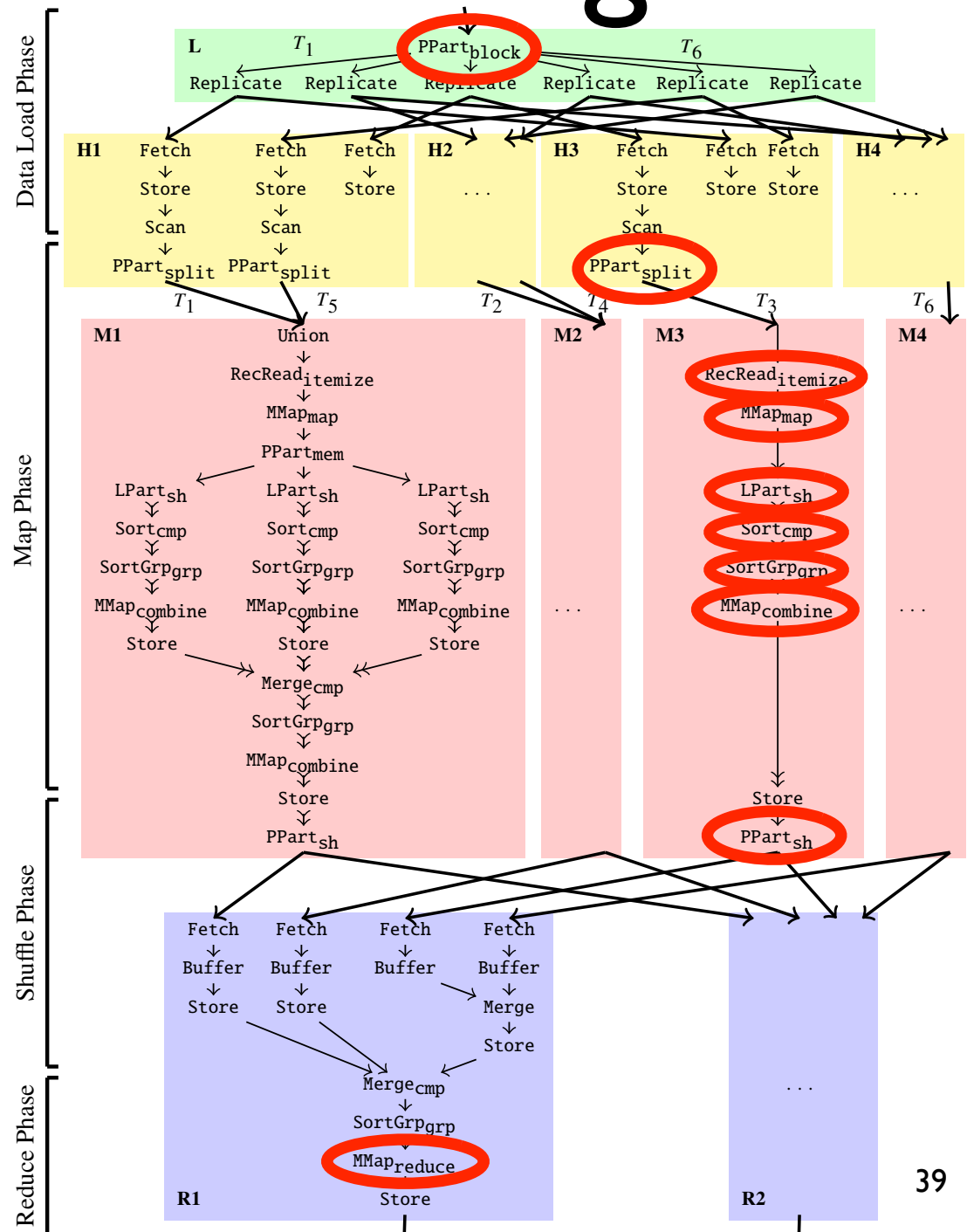
- Scan Tasks
- Selection Tasks
- Projection Tasks
- Join Tasks

Average PageRank visited by each user IP

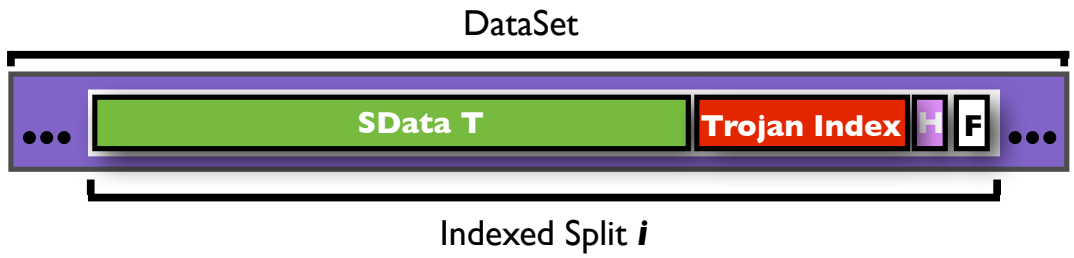
Web-log Processing

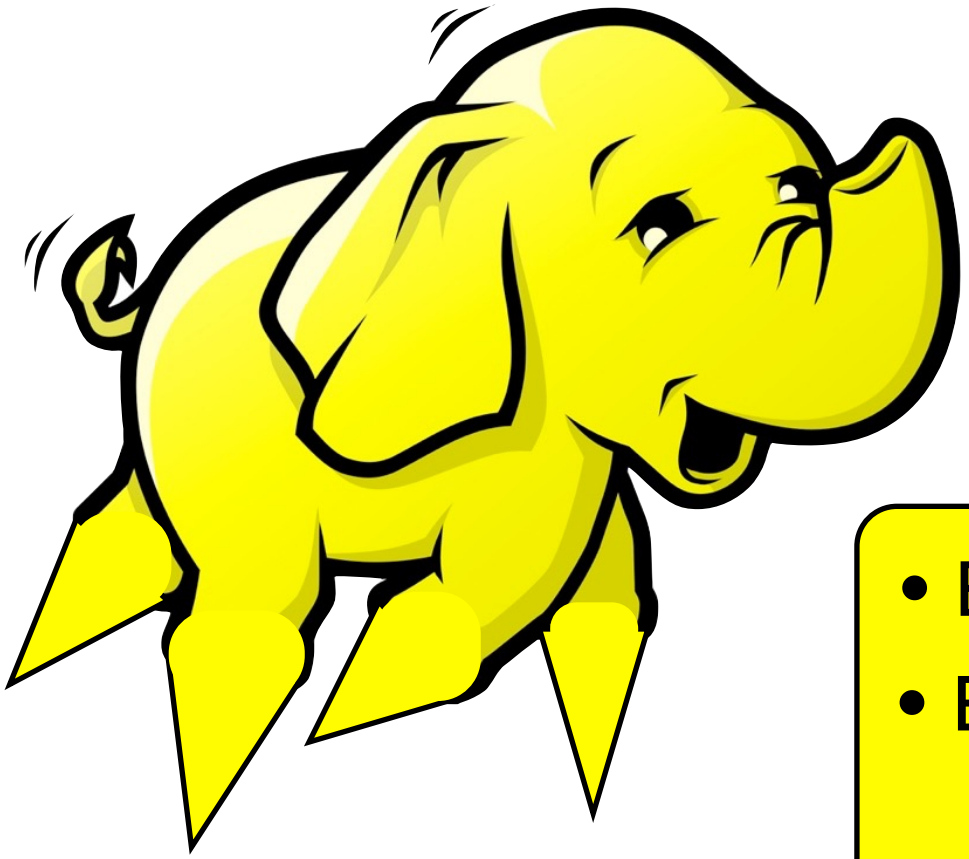


- Scan Tasks
- Selection Tasks
- Projection Tasks
- Join Tasks



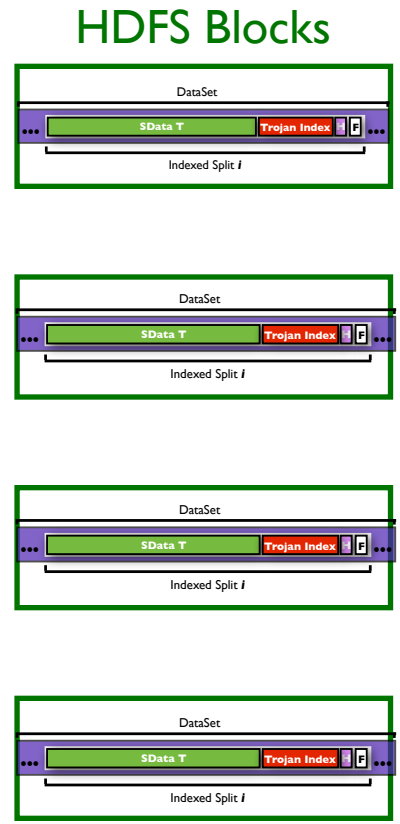
Trojan Index

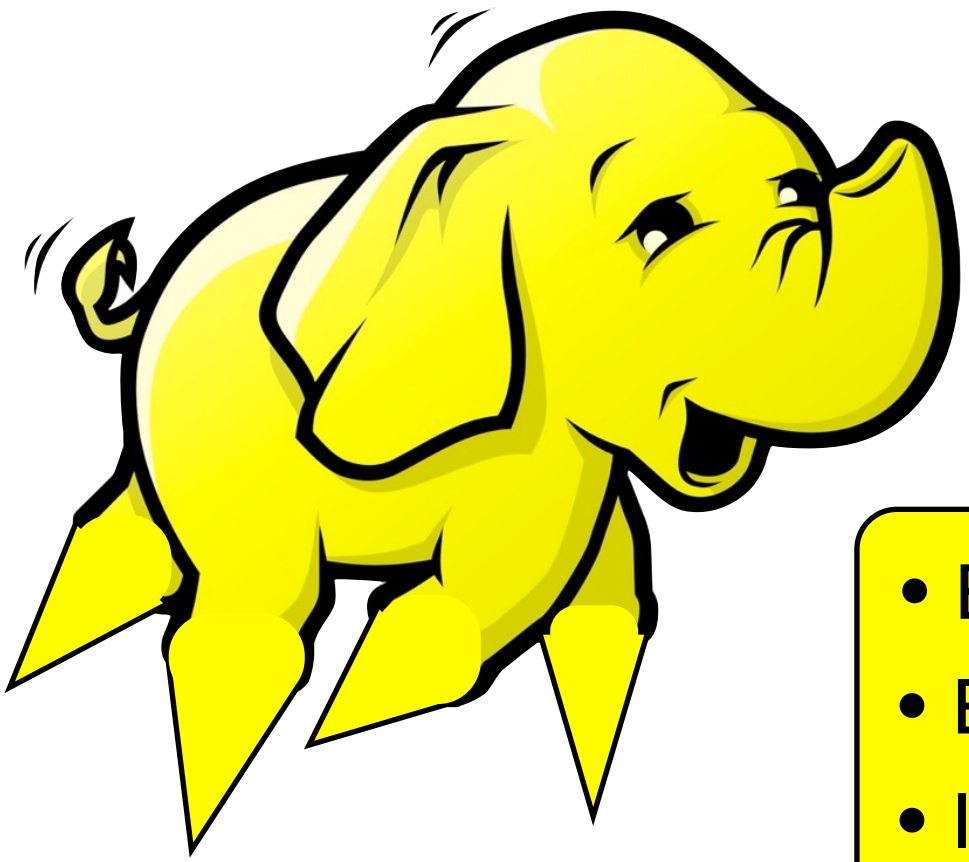




- Each HDFS block sorted
- Each block contains an index

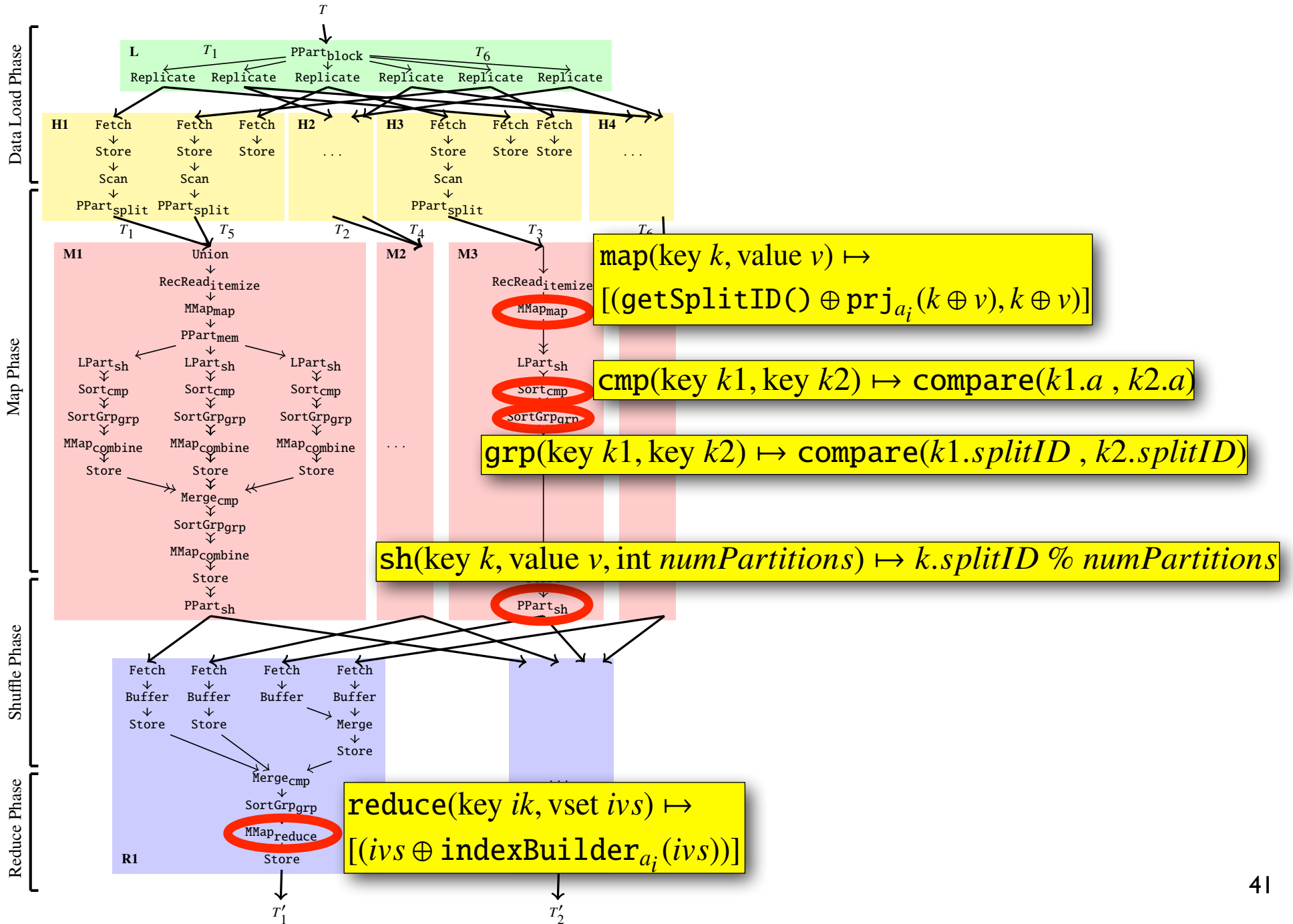
Trojan Index



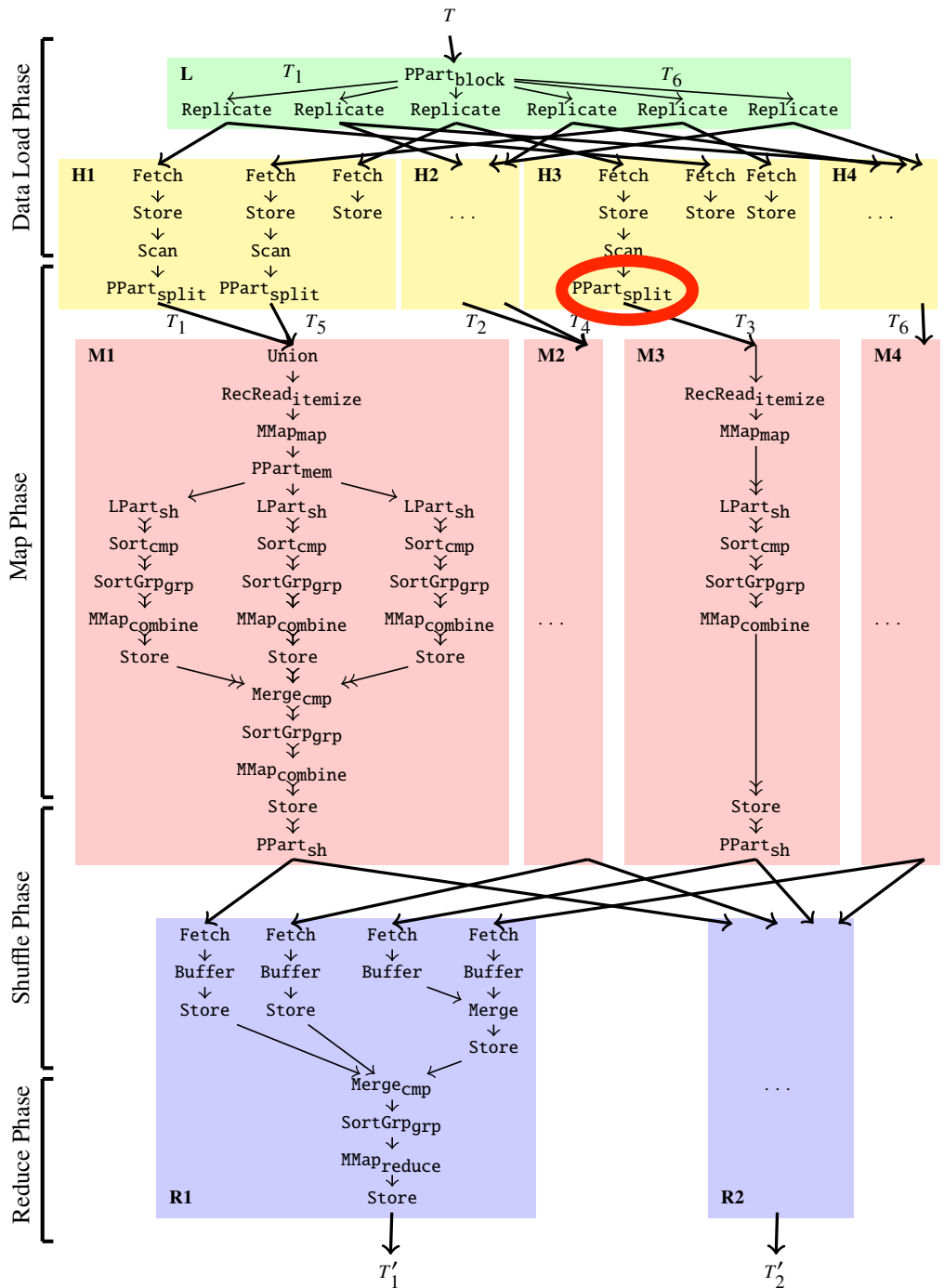


- Each HDFS block sorted
- Each block contains an index
- Index access in UDF

Trojan Index Creation



Trojan Index Access

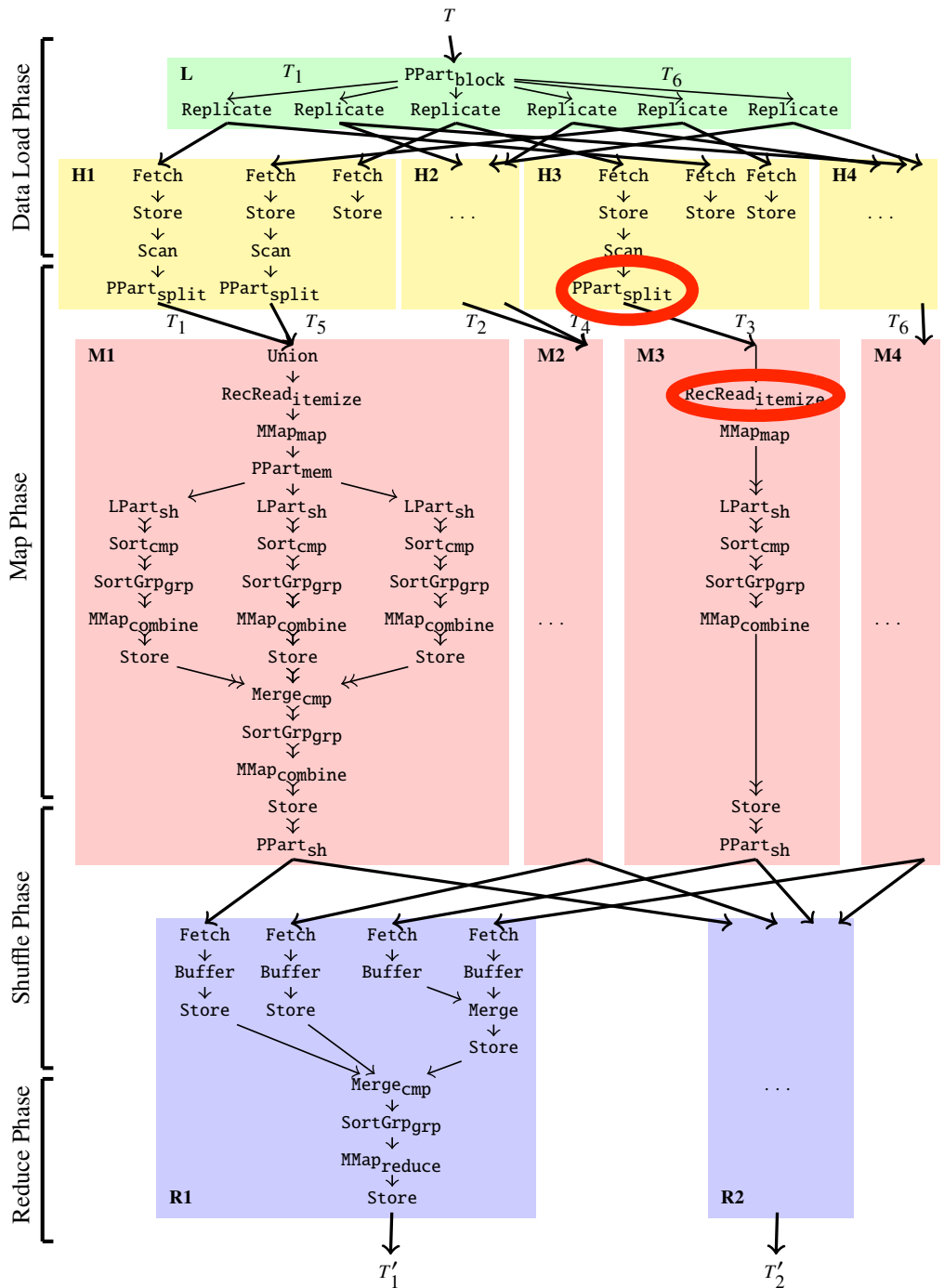


Algorithm 1: Trojan Index/Trojan Join `split` UDF

Input : JobConf job, Int numSplits
Output: logical data splits

```
FileSplit [] splits;
File [] files = GetFiles(job);
foreach file in files do
    Path path = file.getPath();
    InputStream in = GetInputStream(path);
    Long offset = file.getLength();
    while offset > 0 do
        in.seek(offset-FOOTER_SIZE);
        Footer footer = ReadFooter(in);
        Long splitSize = footer.getSplitSize();
        offset -= (splitSize + FOOTER_SIZE);
        BlockLocations blocks = GetBlockLocations(path,offset);
        FileSplit newSplit = CreateSplit(path,offset,splitSize,blocks);
        splits.add(newSplit);
    end
end
return splits;
```

Trojan Index Access

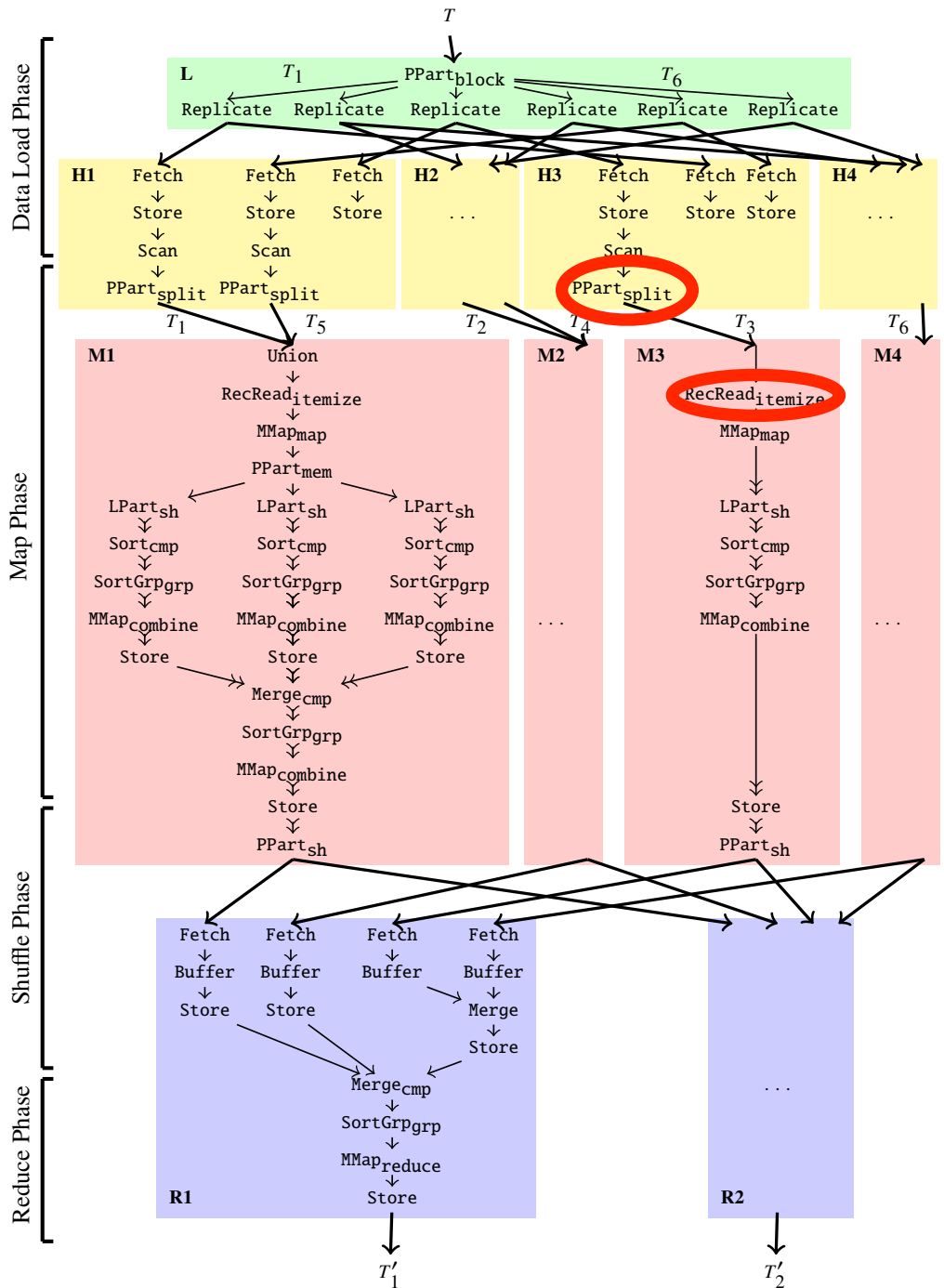


Algorithm 2: Trojan Index `itemize`.initialize UDF

Input: FileSplit split, JobConf job

```
Global FileSplit split = split;
Key lowKey = job.getLowKey();
Global Key highKey = job.getHighKey();
Int splitStart = split.getStart();
Global Int splitEnd = split.getEnd();
Header h = ReadHeader(split);
Overlap type = h.getOverlapType(lowKey,highKey);
Global Int offset;
if type == LEFT_CONTAINED or type == FULL_CONTAINED or type ==
  POINT_CONTAINED then
    Index i = ReadIndex(split);
    offset = splitStart + i.lookup(lowKey);
else if type == RIGHT_CONTAINED or type == SPAN then
    offset = splitStart;
else
    // NOT_CONTAINED, skip the split;
    offset = splitEnd;
end
Seek(offset);
```

Trojan Index Access

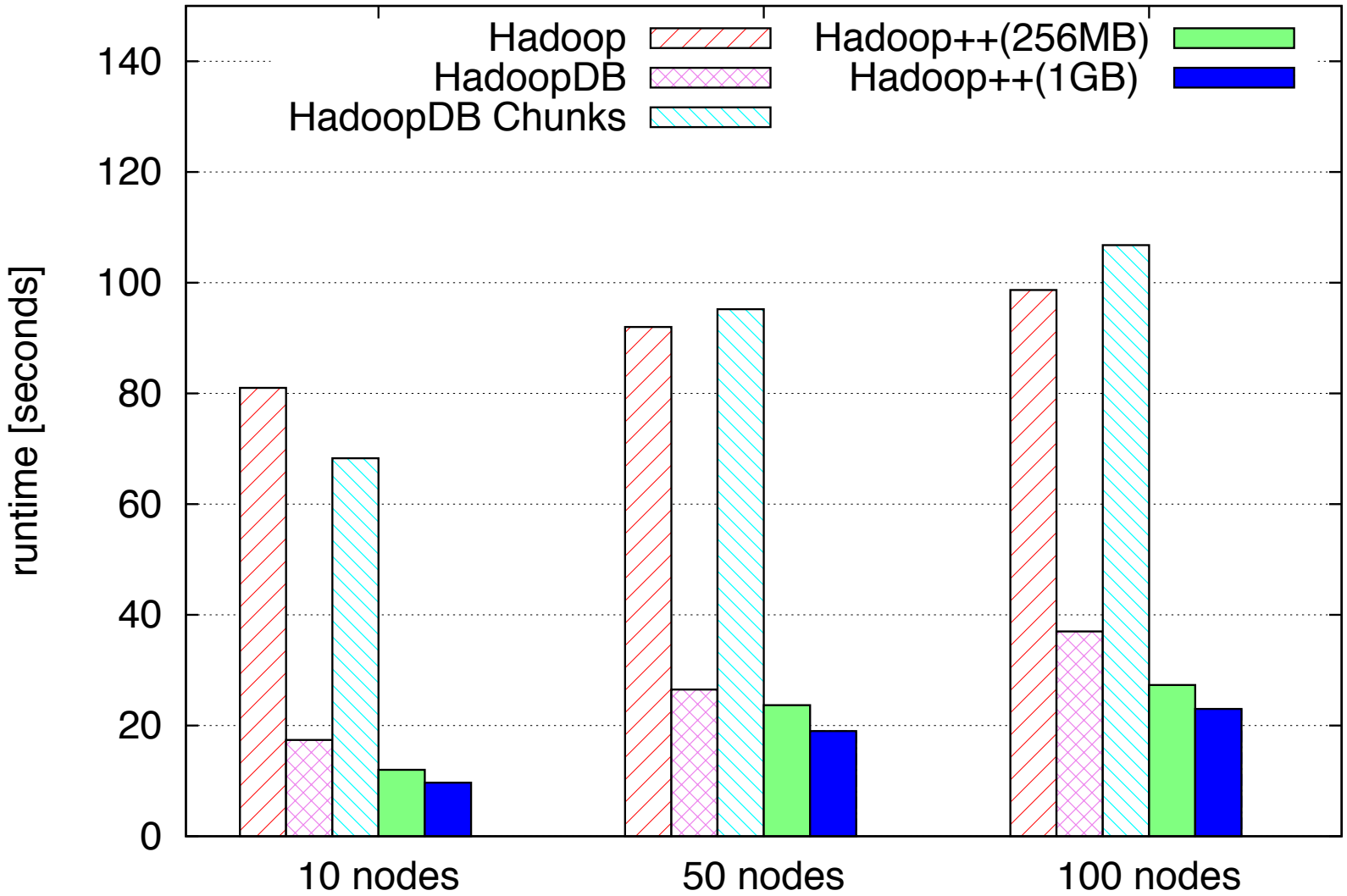


Algorithm 3: Trojan Index `itemize`.next UDF

Input : KeyType key, ValueType value
Output: has more records

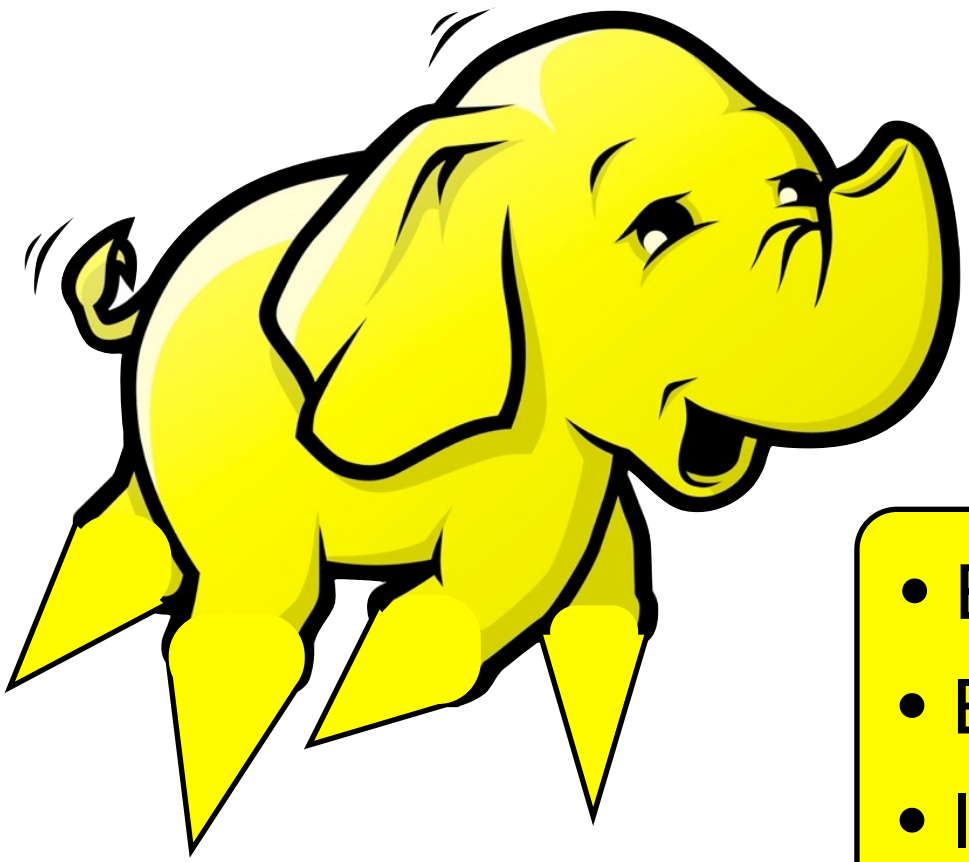
```
if offset < splitEnd then
    Record nextRecord = ReadNextRecord(split);
    offset += nextRecord.size();
    if nextRecord.key < highKey then
        SetKeyValue(key, value, nextRecord);
        return true;
    end
end
return false;
```

Selection Analytical Task *



* Pavlo et. al. A Comparison of Approaches to large-Scale Data Analysis. SIGMOD 2009

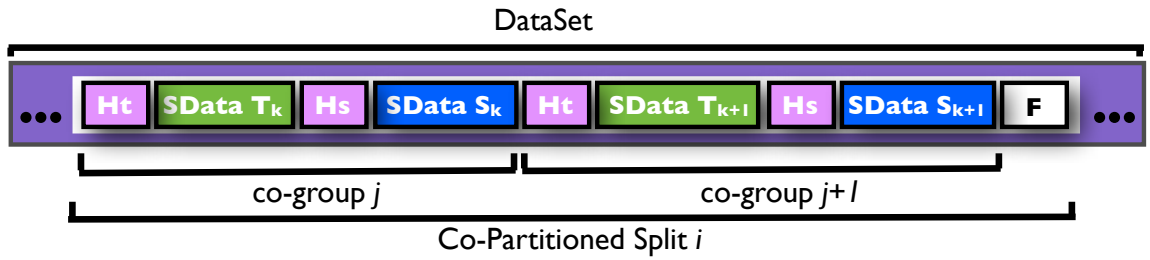
Trojan Index Advantages

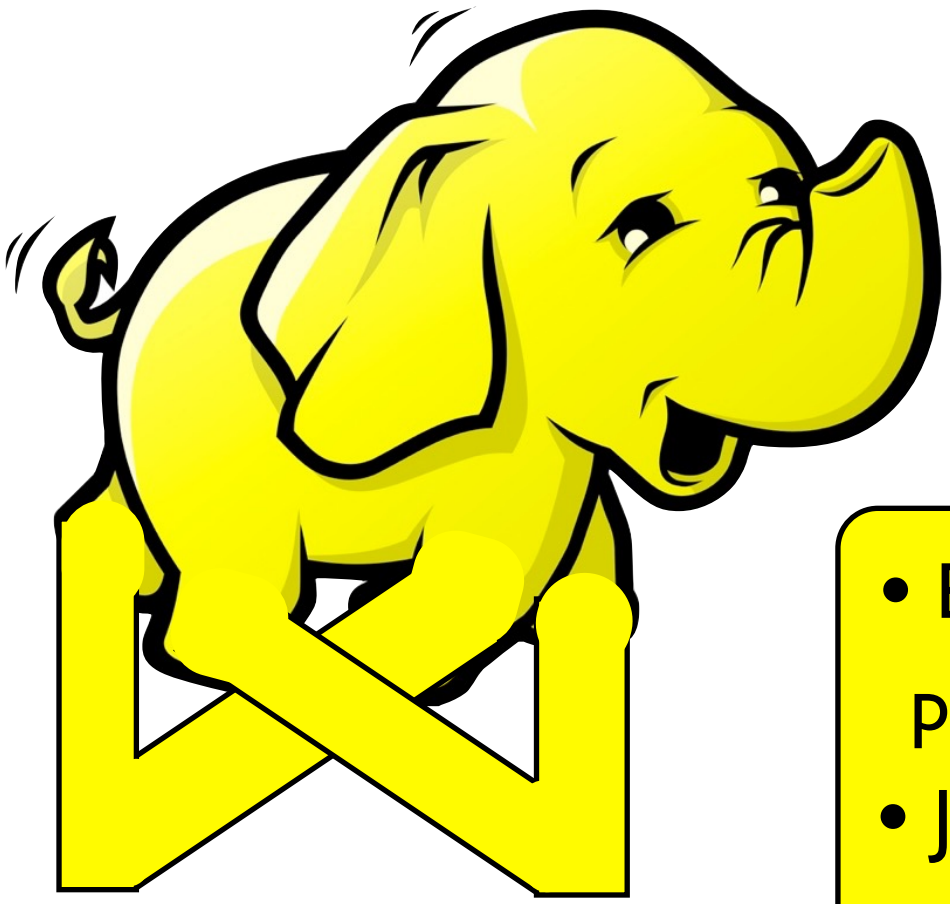


- Scan + Index data accesses
- Parallel index lookups
- Non-invasive system changes
- Much better performance

- Each HDFS block sorted
- Each block contains an index
- Index access in UDF

Trojan Join

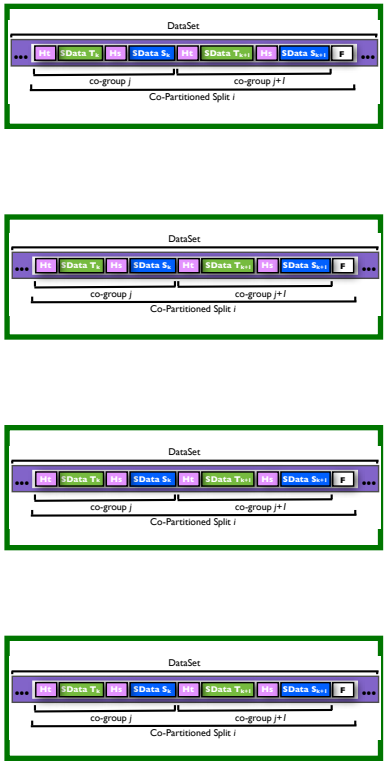


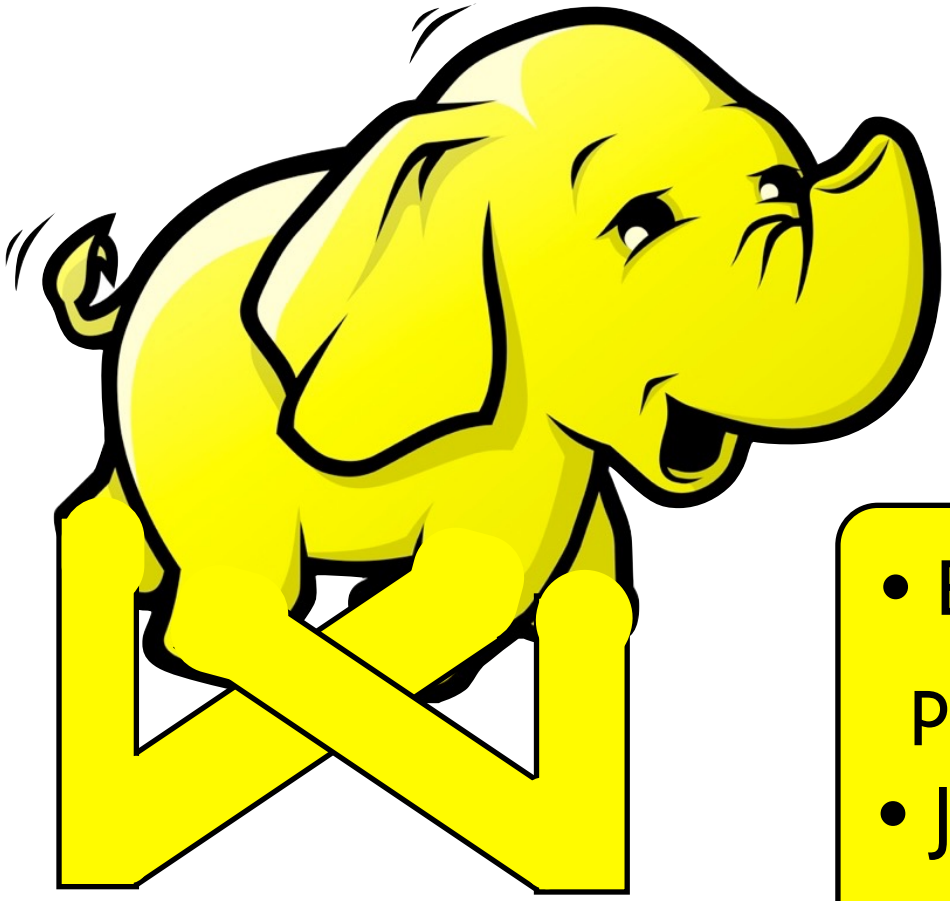


- Each HDFS block co-partitioning over two relations
- Join relations are co-located

Trojan Join

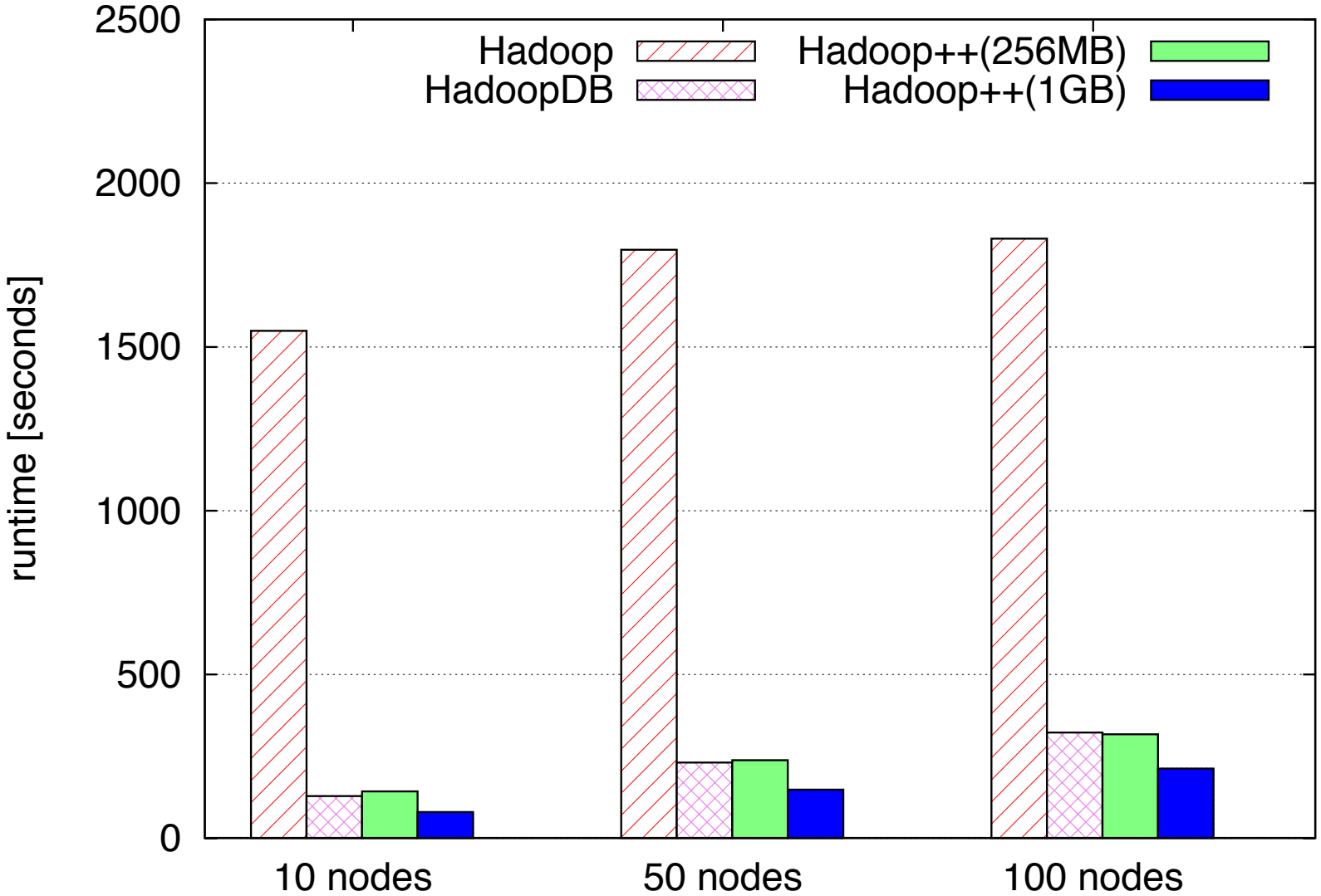
HDFS Blocks





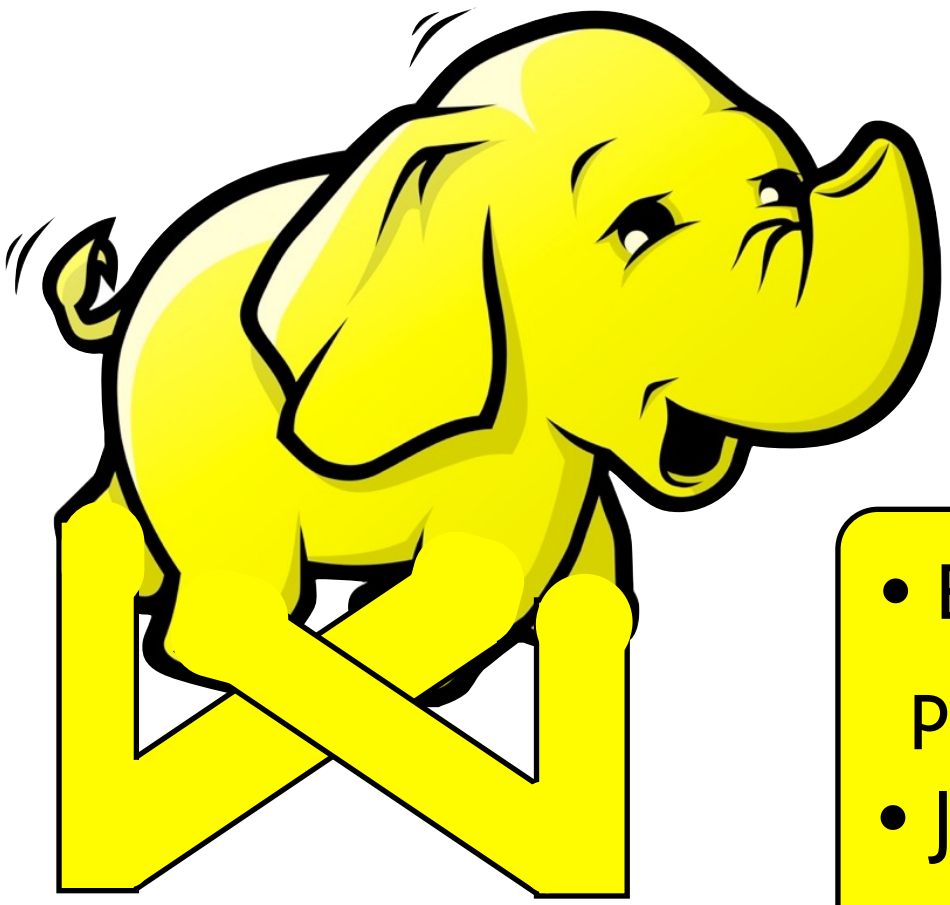
- Each HDFS block co-partitioning over two relations
- Join relations are co-located
- Co-partitioned join in UDF

Join Analytical Task *



* Pavlo et. al. A Comparison of Approaches to large-Scale Data Analysis. SIGMOD 2009

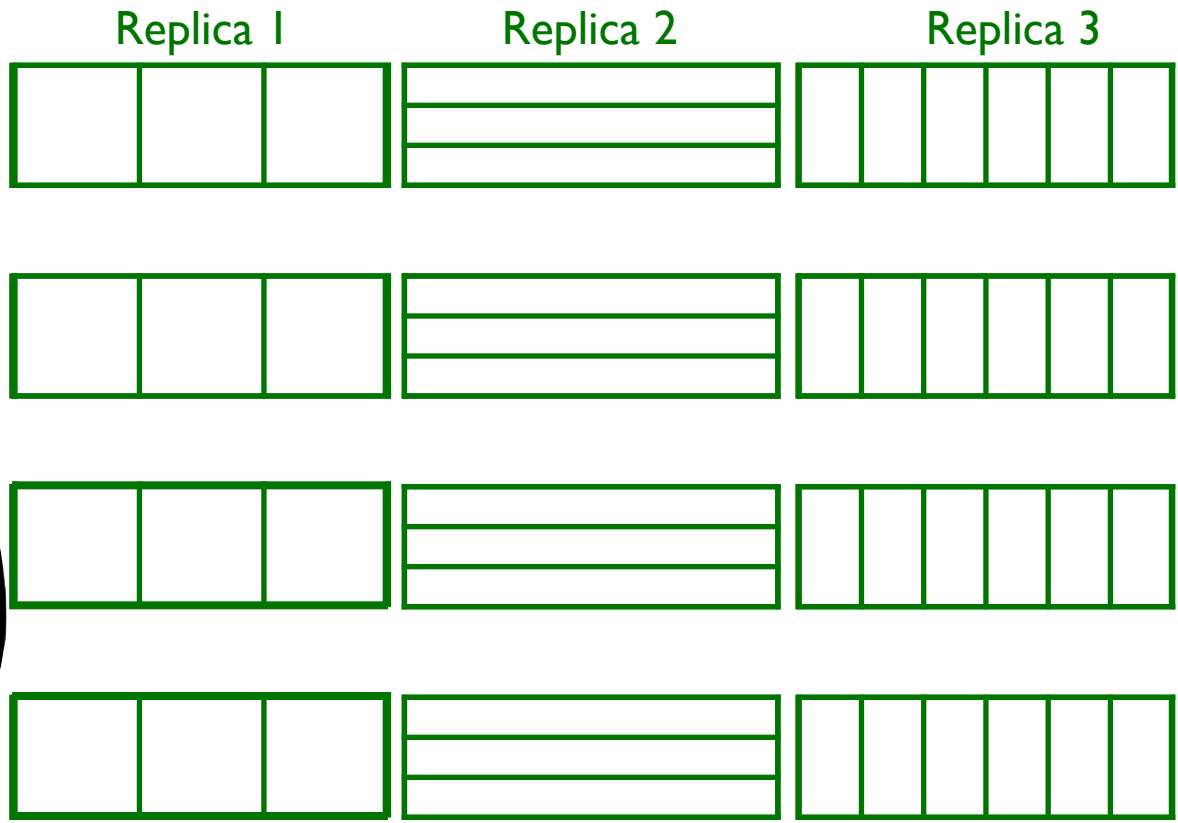
Trojan Join Advantages

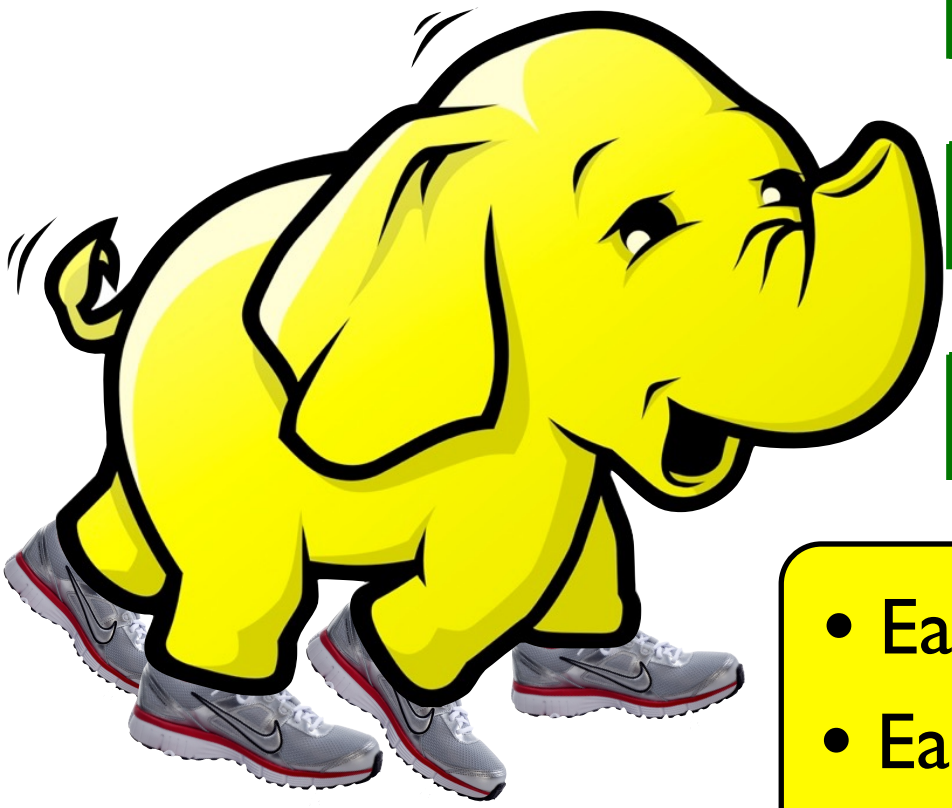


- Re- + Co- partitioned join
- Parallel join processing
- Non-invasive system changes
- Much better performance

- Each HDFS block co-partitioning over two relations
- Join relations are co-located
- Co-partitioned join in UDF

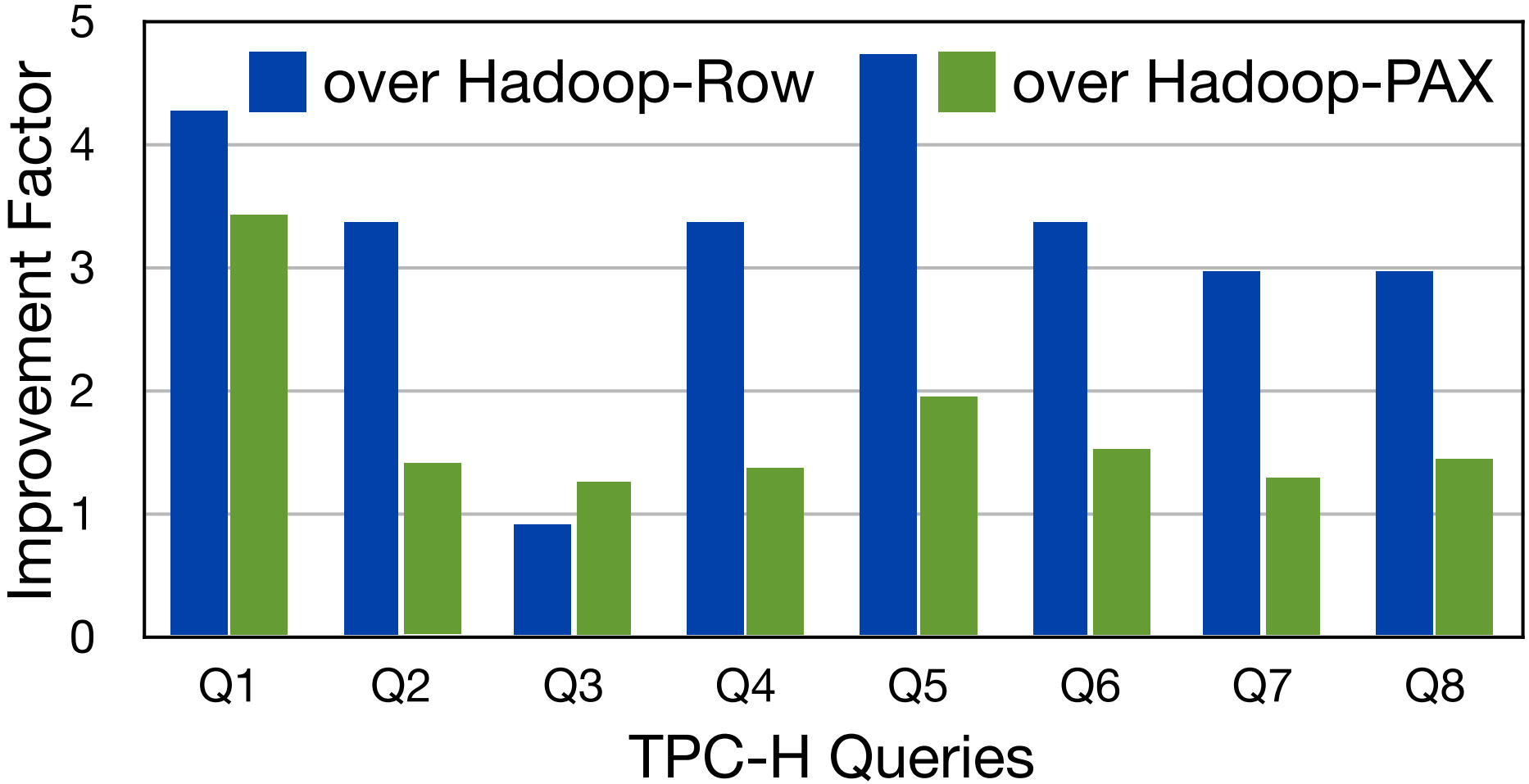
Trojan Layouts



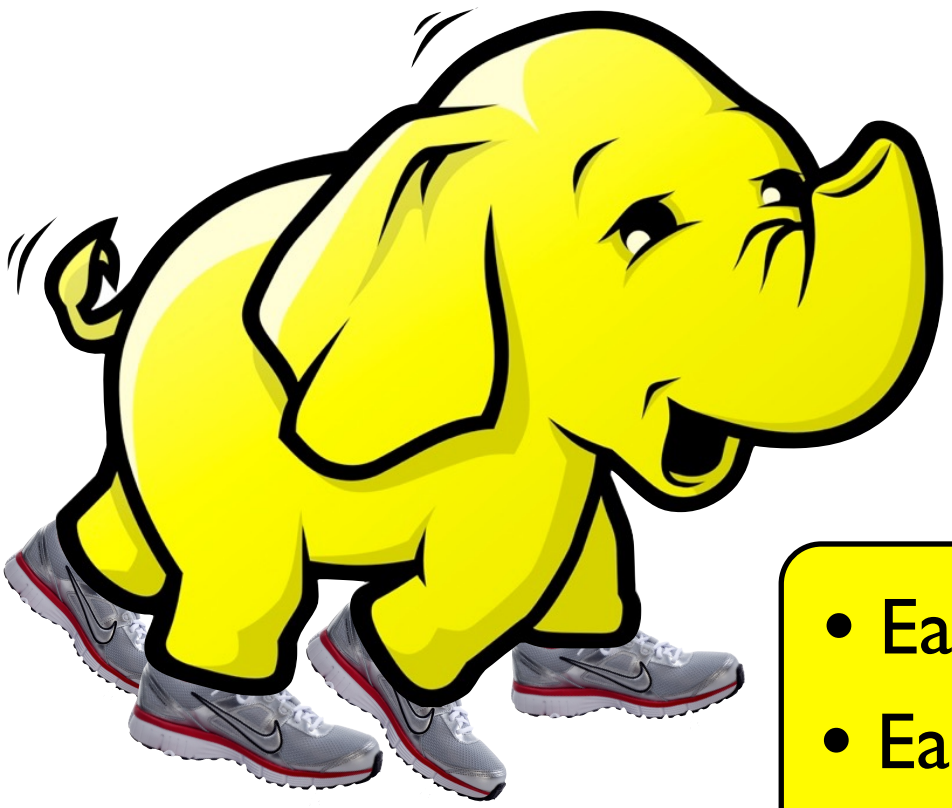


- Each HDFS block in row or column
- Each block replica in different layout
- Pick right layout in UDF

Projection Analytical Task



Trojan Layouts Advantages

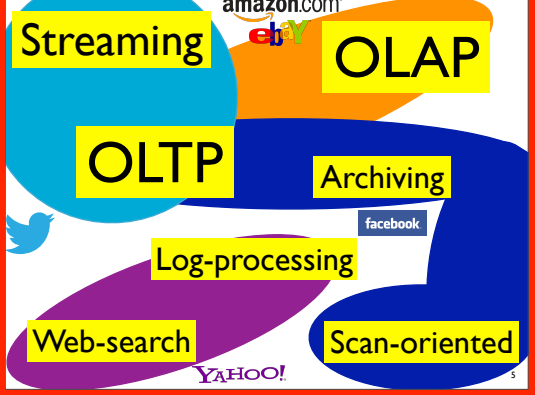


- Row, PAX, Column-group layouts
- Several layouts at the same time
- Non-invasive system changes
- Much better performance

- Each HDFS block in row or column
- Each block replica in different layout
- Pick right layout in UDF

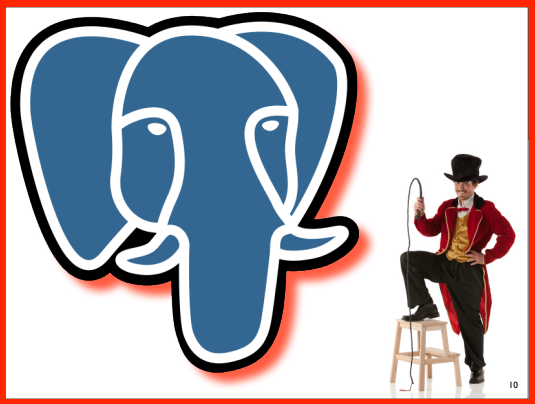
Open Issues

- Automatically rewriting user queries
- Avoiding UDF call overheads for low selectivity
- Putting all Trojan Techniques together in a single system
- What to store, How to store, Where to store
- Trojan Techniques: one way of approaching OctopusDB
- Trojan Techniques: first step towards OctopusDB
- Storage View optimization: selection, transformation, update propagation



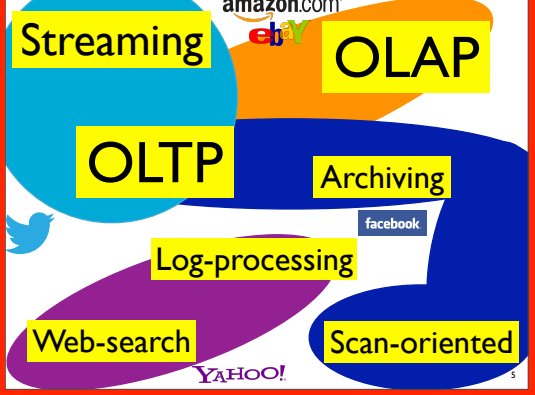
Mixed Workloads

Big Problem



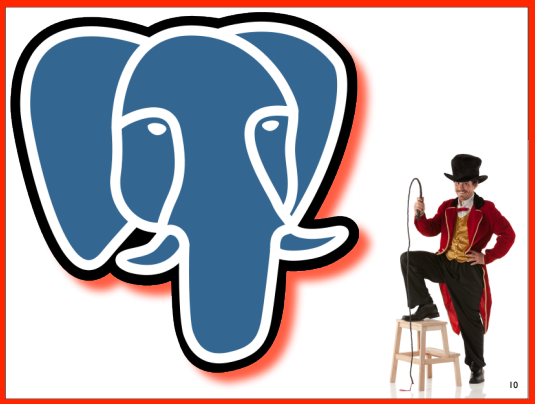
Not Sufficient

Complicated



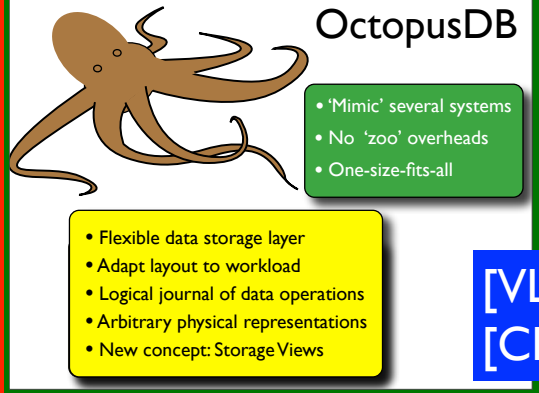
Mixed Workloads

Big Problem



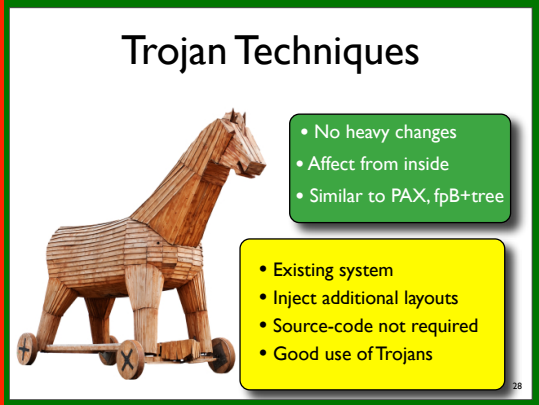
Not Sufficient

Complicated



Our Vision

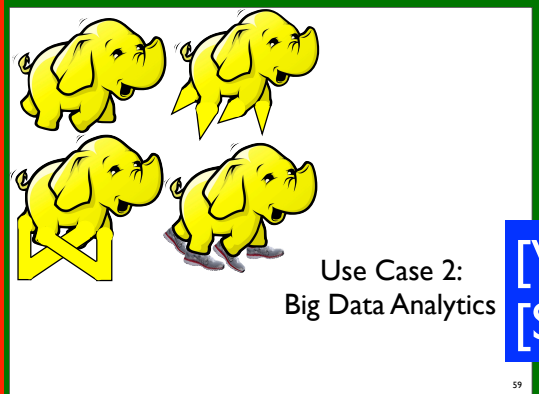
[VLDB PhD Workshop, 2010]
[CIDR, 2011]



First Steps

OLAP in Row-stores

[CIDR, 2013]
(Under Submission)



Big Data Analytics

[VLDB, 2010]
[SOCC, 2011]