CLUSTER OF EXCELLENCE

SAARLAND UNIVERSITY
Information Systems Group

The Mimicking Octopus

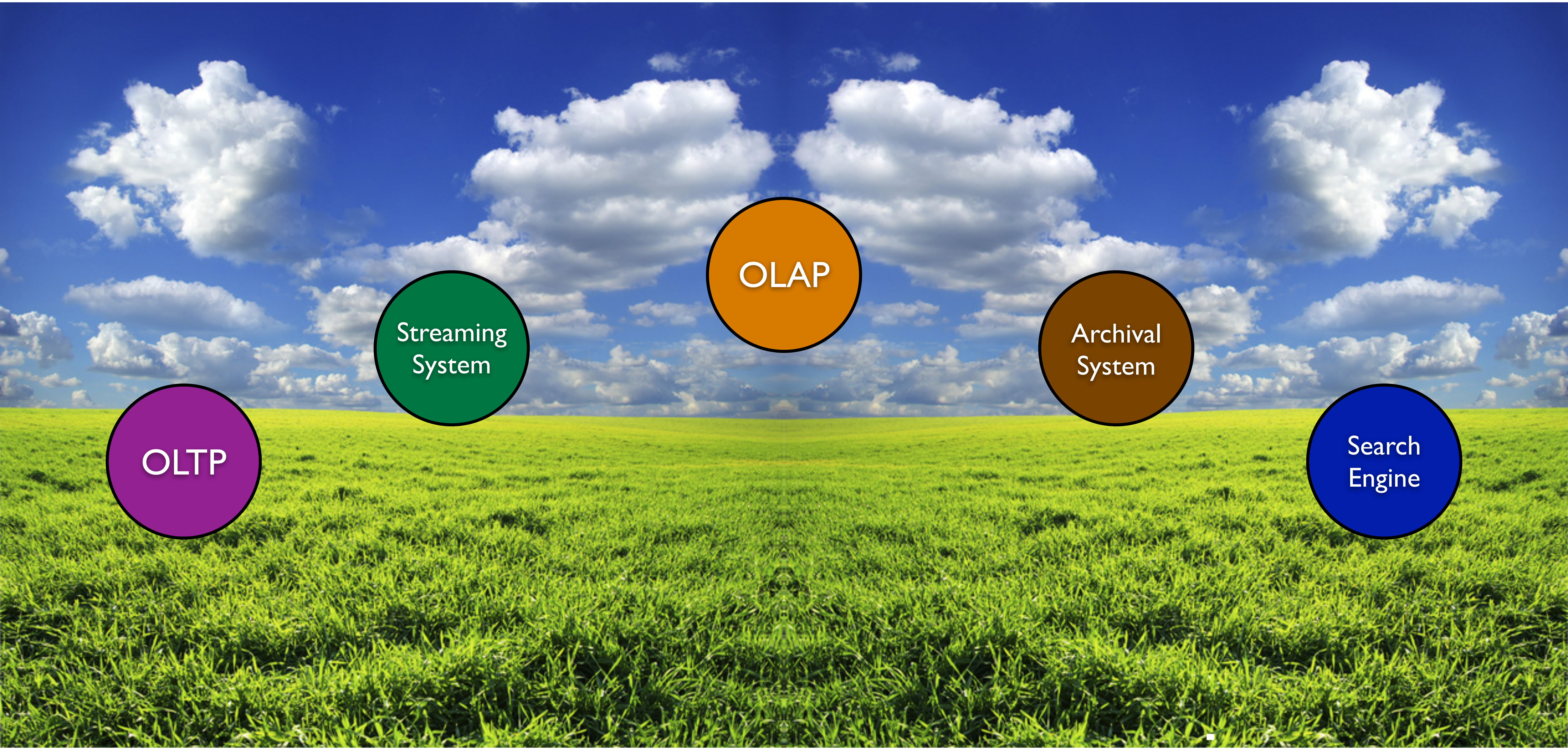
Towards a one-size-fits-all Architecture for Database Systems

Alekh Jindal

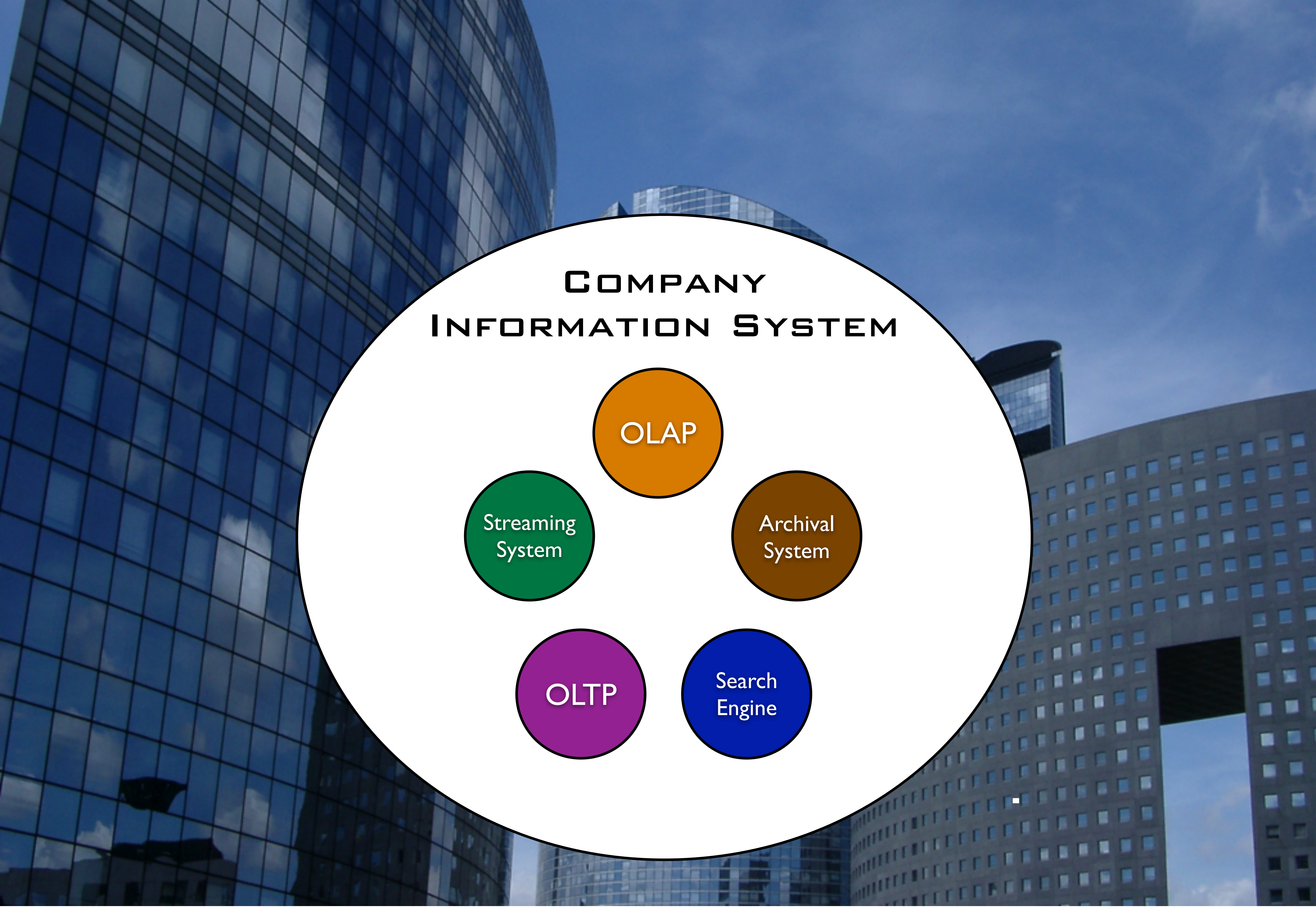
Supervisor: Prof. Dr. Jens Dittrich

VLDB PhD Workshop
September 13, 2010

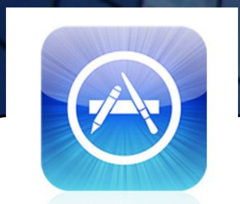
Database Landscape



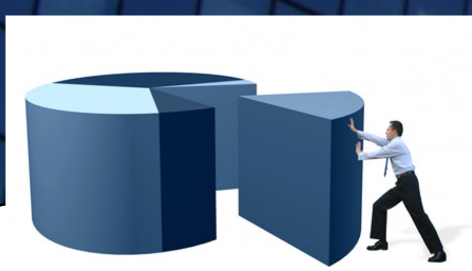
Motivation


Company
Information System
OLAP
Streaming
System
Archival
System
OLTP
Search
Engine

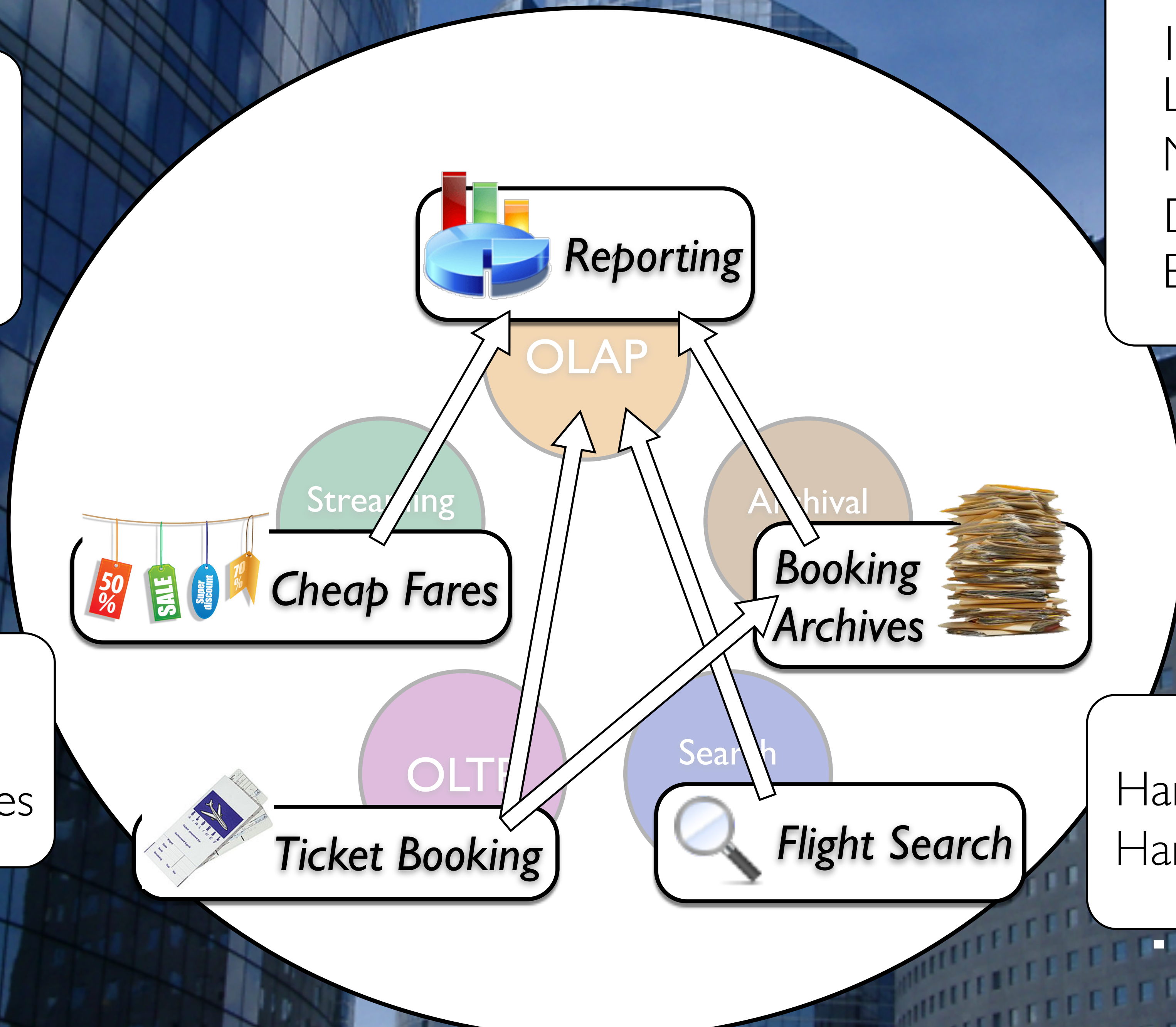
Airline Company



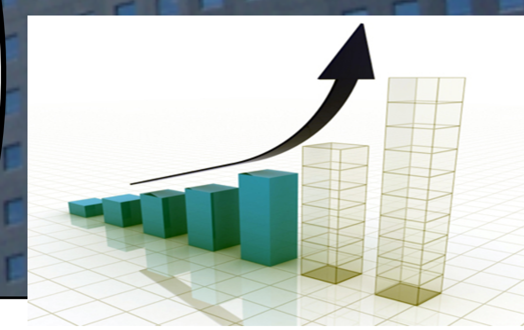
Several Applications
Evolving Applications



Eventual Integration
ETL style data pipelines



Integration Cost
Licensing Cost
Maintenance Cost
DBA Cost
Engineering Cost



Hard-coded optimizations
Hard-coded data layouts

Problem Statement

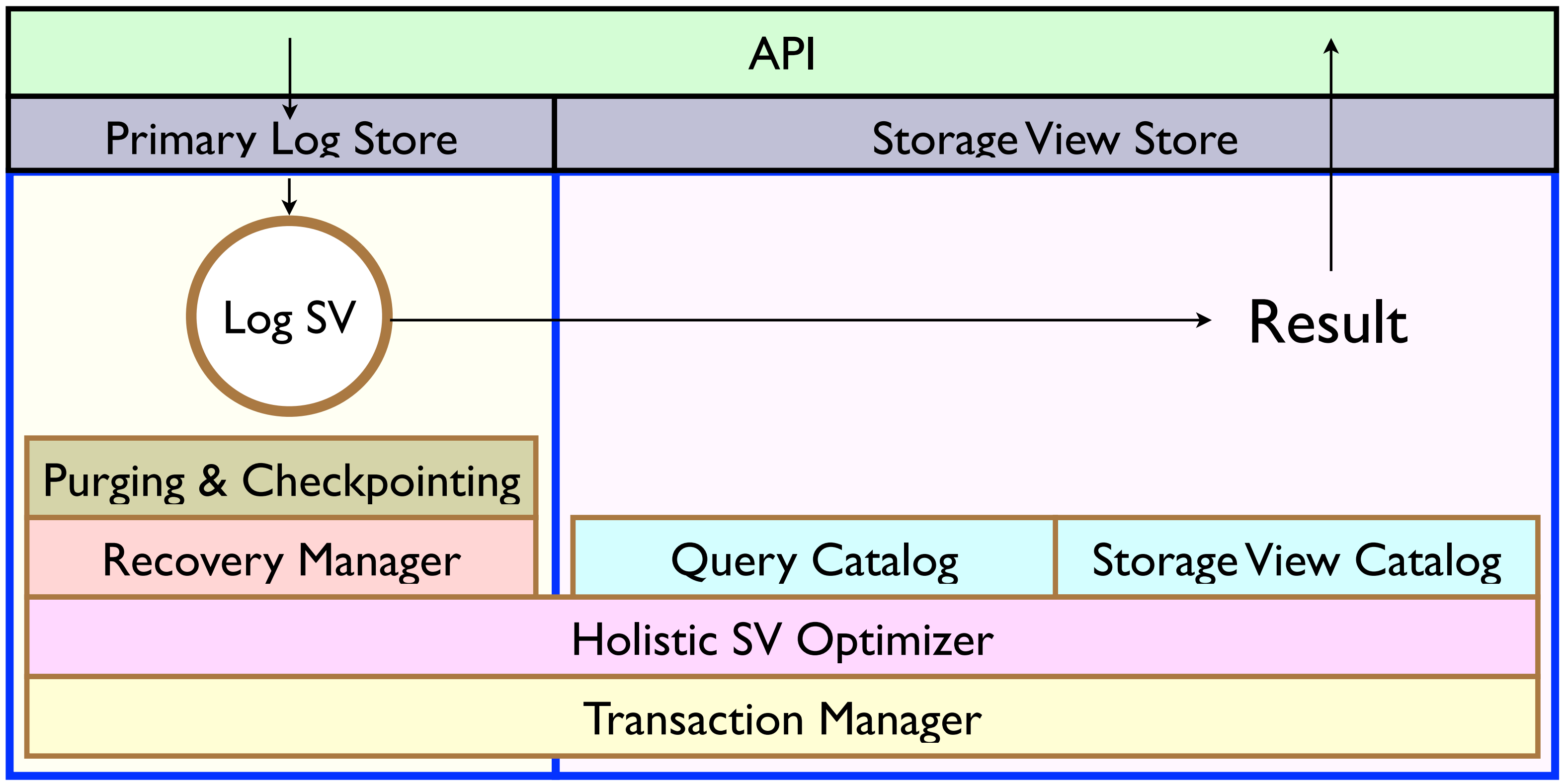
- Single database system
- Automatic adaption
- Improved performance
- Lower cost
- Better maintainability

OctopusDB Overview

- One-size-fits-all architecture
- Abstract storage concept: Storage Views(SV)
- Single optimization problem: SV Selection
- Holistic SV optimizer

System Architecture

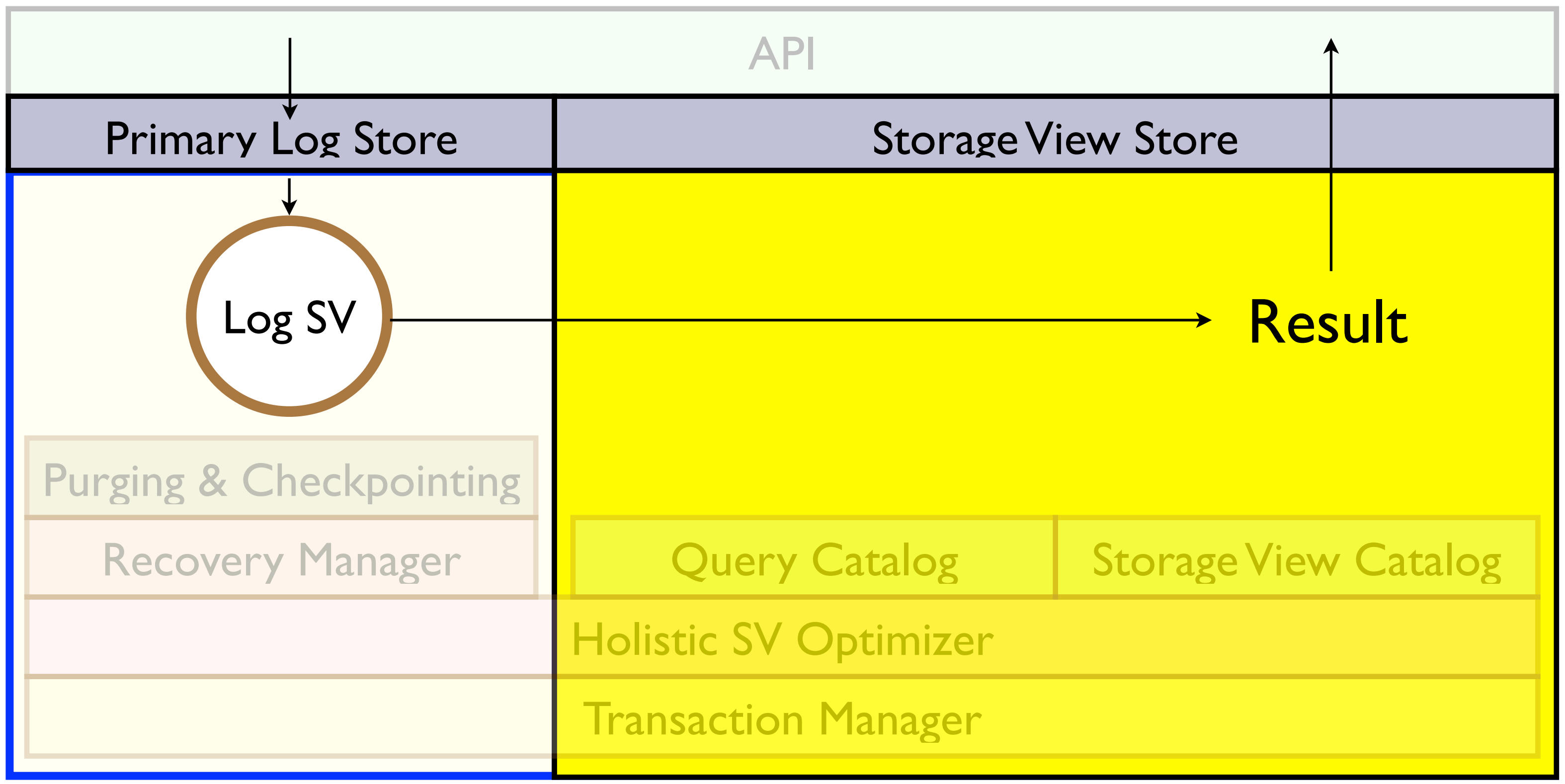
SAARLAND UNIVERSITY
Information Systems Group



- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL

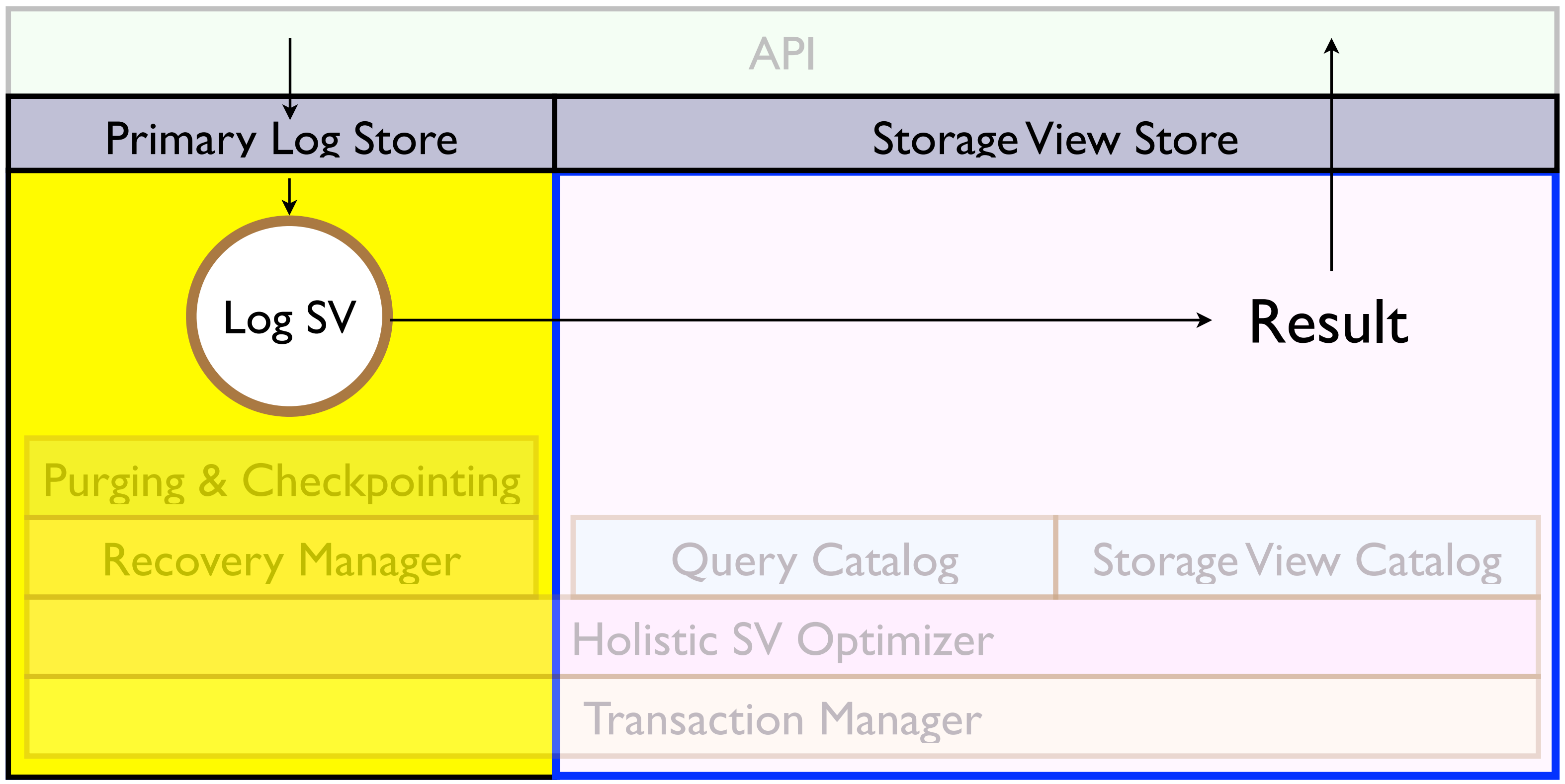
OctopusDB

System Architecture



- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL

System Architecture



- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL

OctopusDB

Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella

Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella

Primary

Log SV

Row SV

Column SV

Index SV

Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella

Primary	Secondary
Log SV	Partial Index SV
Row SV	Bag-partitioned SV
Column SV	Key-consolidated SV
Index SV	Vertically/Horizontally Partitioned SV

Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella

Primary	**Secondary**
Log SV	Partial Index SV
Row SV	Bag-partitioned SV
Column SV	Key-consolidated SV
Index SV	Vertically/Horizontally Partitioned SV

... any hybrid combination of the above

Use-case Scenario*

- Flight booking system
- Tables: **Tickets**, **Customers**
- **Tickets**: several attributes, frequently updated
- **Customers**: fewer attributes
- Queries:

```
SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id AND T.a1=x1 AND T.a2=x2 ... AND T.an=xn
```

* Inspired from Unterbrunner et al. in PVLDB, 2009.

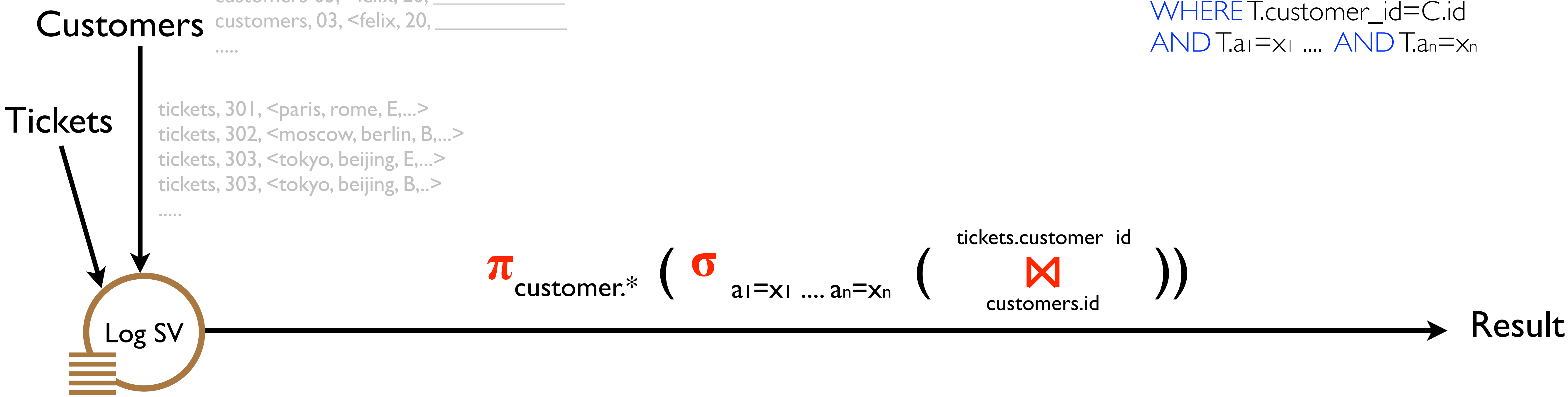
Storage Views

Flight Booking System

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$

customers, 01, <tom, 25, ____________
customers, 02, <marc, 23, ____________
customers 03, <felix, 20, ____________
customers, 03, <felix, 20, ____________
.....

tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, E,...>
tickets, 303, <tokyo, beijing, B,..>
.....



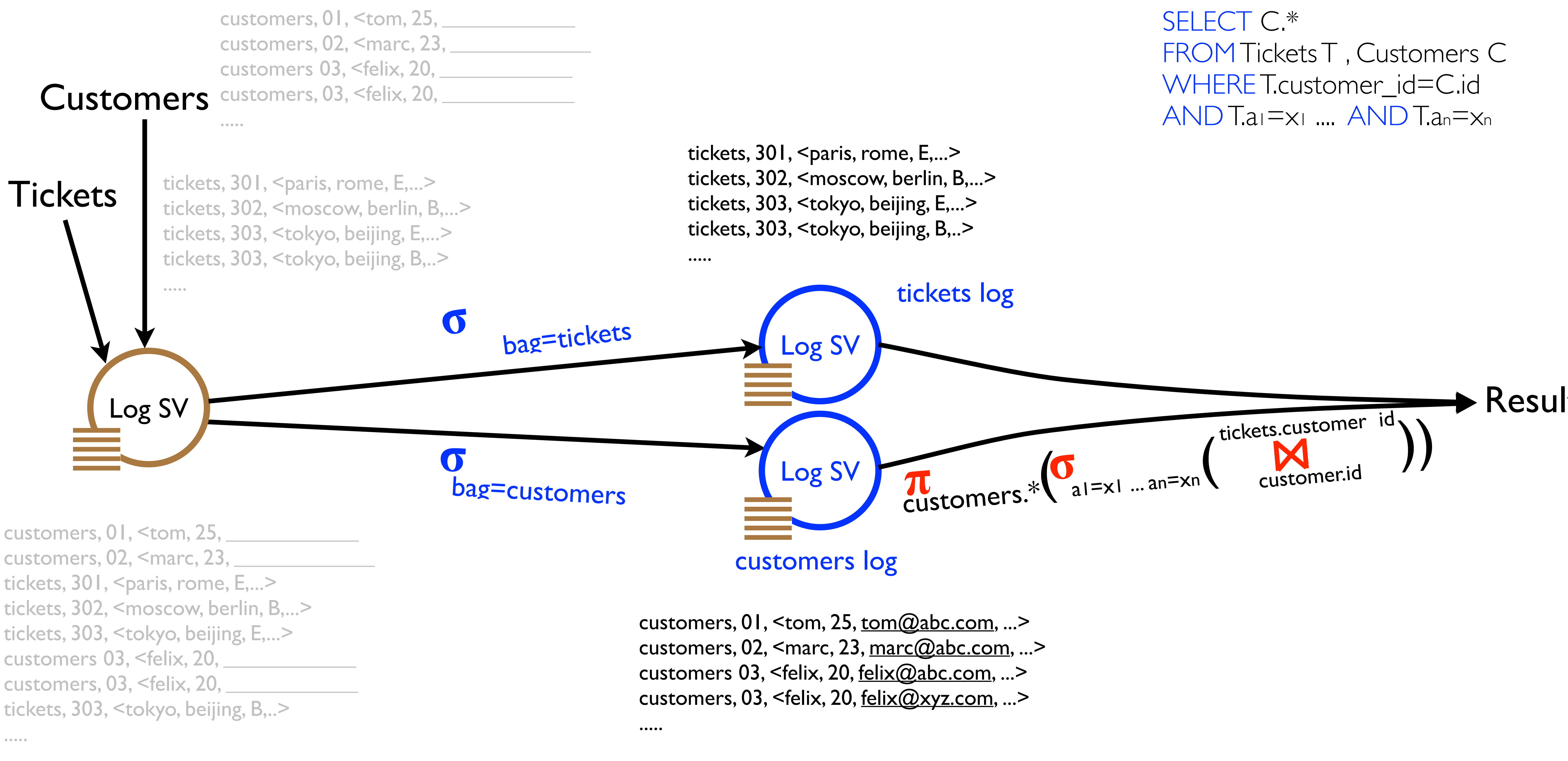
$$\pi_{customer.*}\left(\sigma_{a_1=x_1 \ldots a_n=x_n}\left(\bowtie^{tickets.customer\_id}_{customers.id}\right)\right)$$

customers, 01, <tom, 25, tom@abc.com, ...>
customers, 02, <marc, 23, marc@abc.com, ...>
tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, E,...>
customers 03, <felix, 20, felix@abc.com, ...>
customers, 03, <felix, 20, felix@xyz.com, ...>
tickets, 303, <tokyo, beijing, B,..>
.....
.....

Storage Views

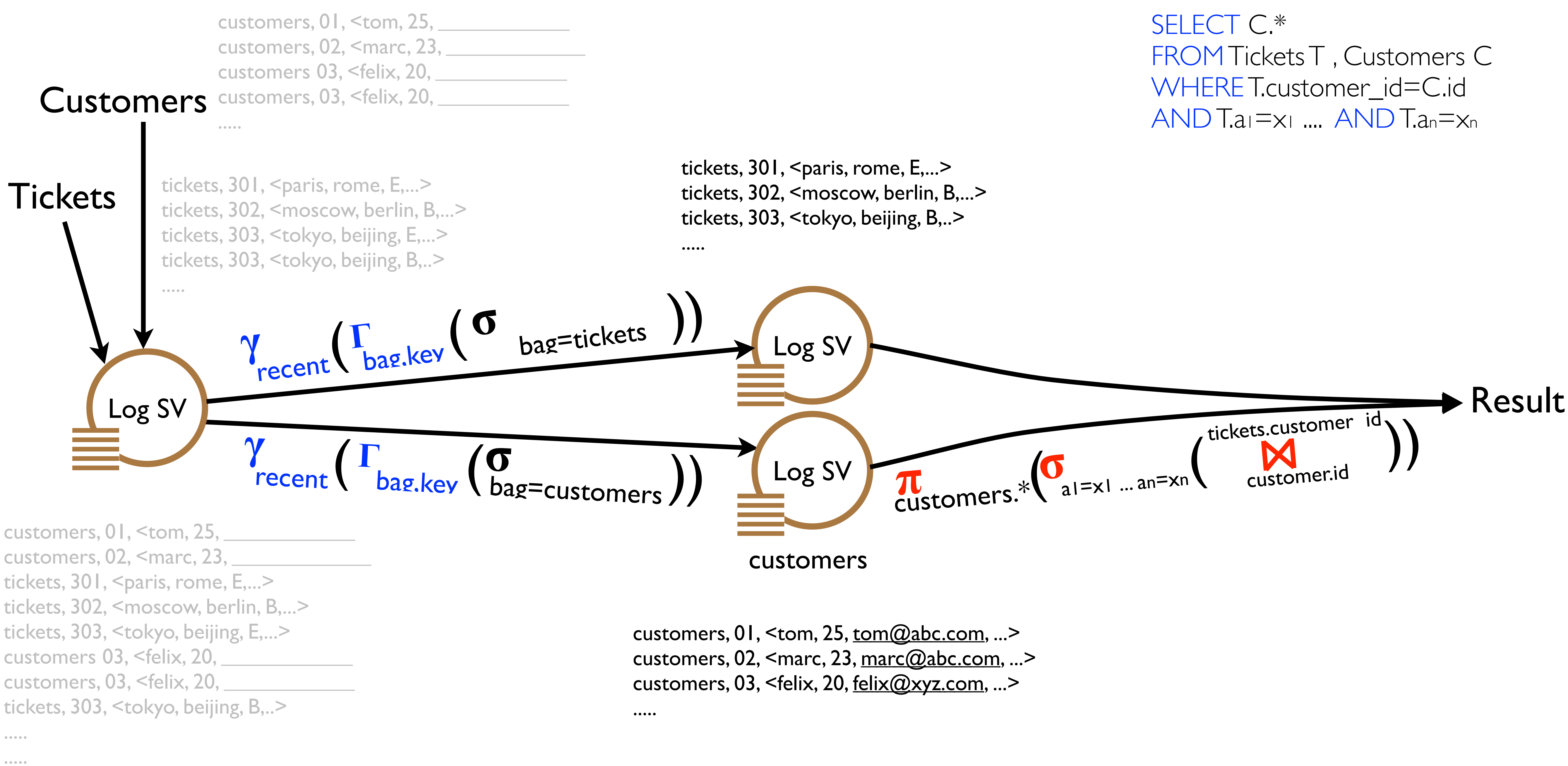
Bag-partitioning

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$



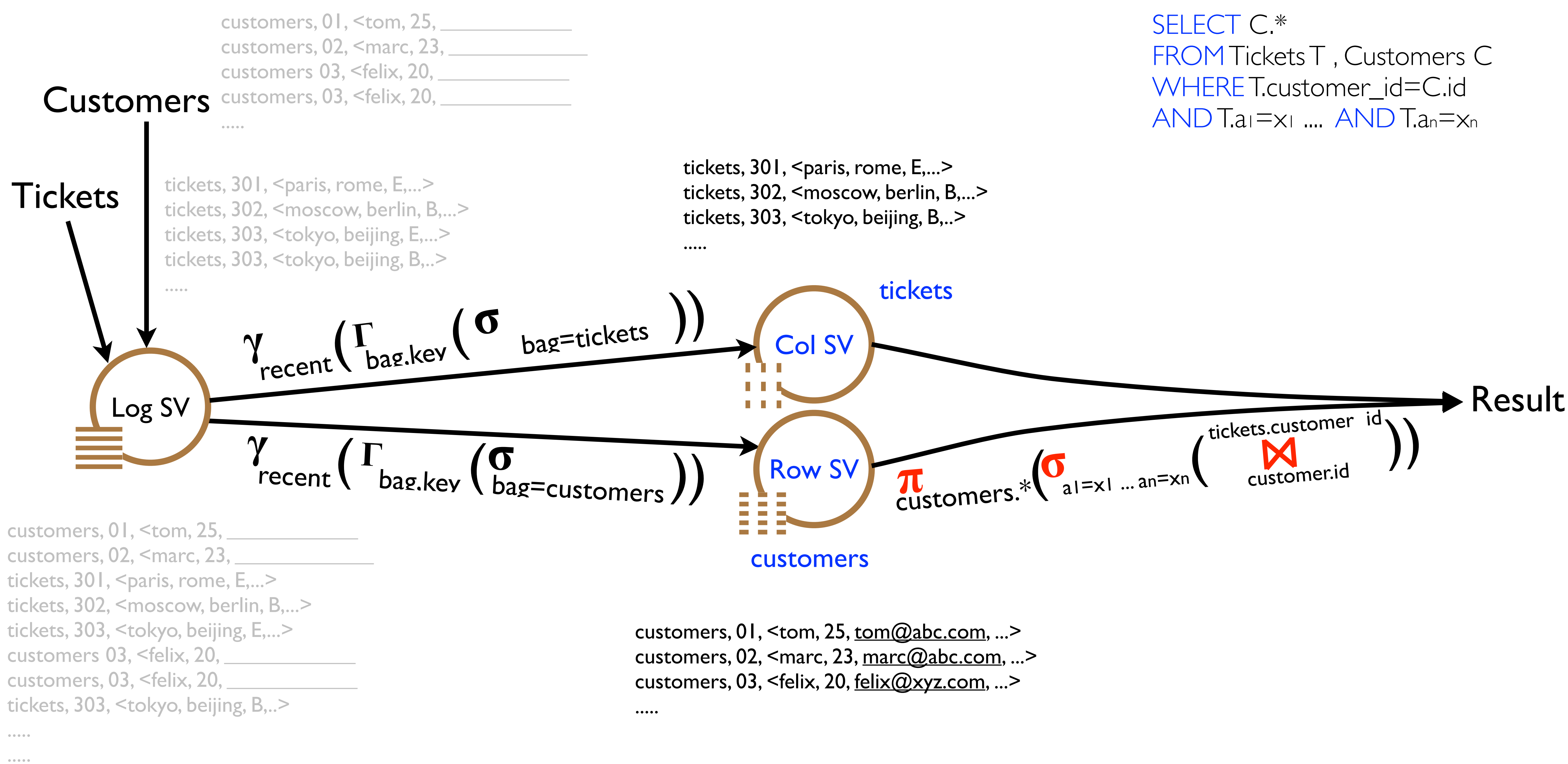
Storage Views

Key-consolidation



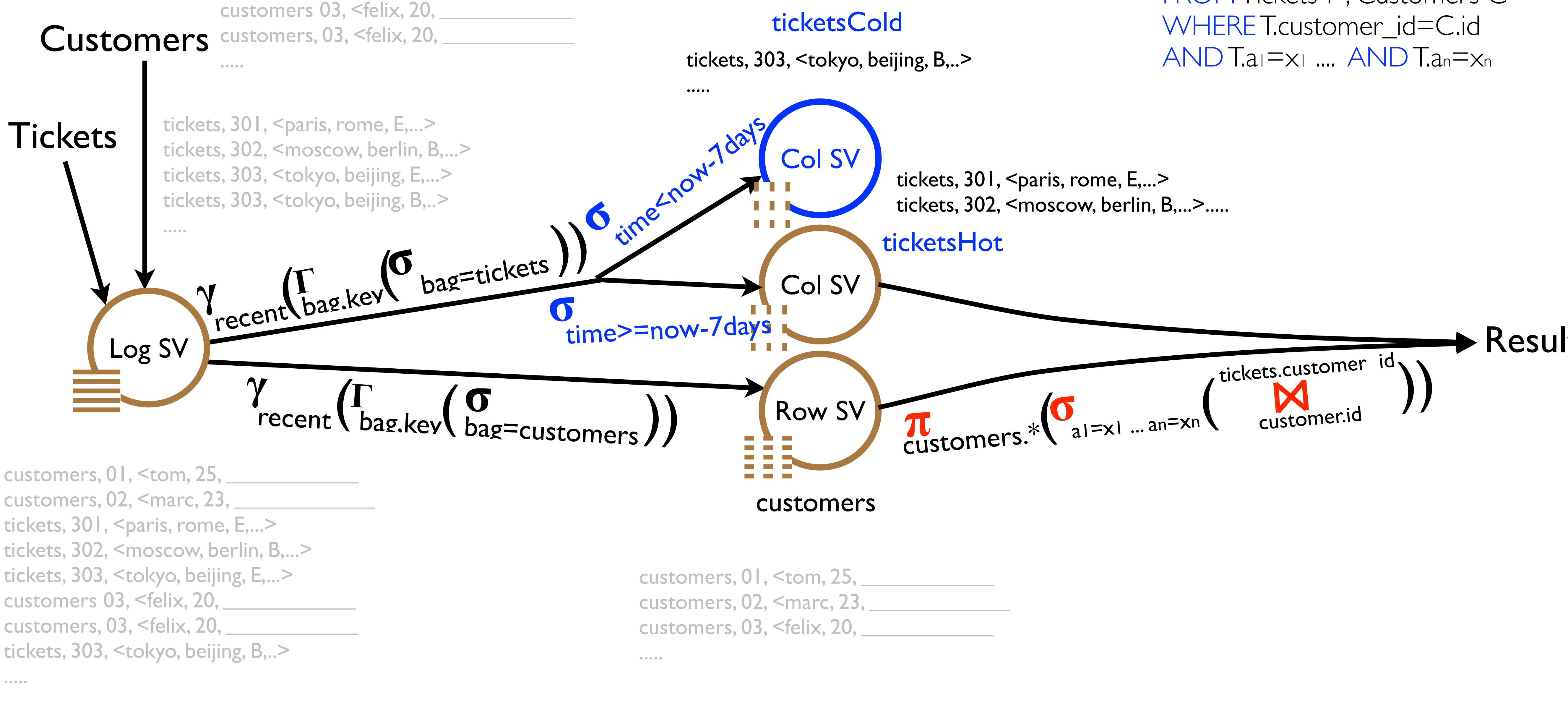
Storage Views

Storage View Transformation



Hot-Cold Storage Views

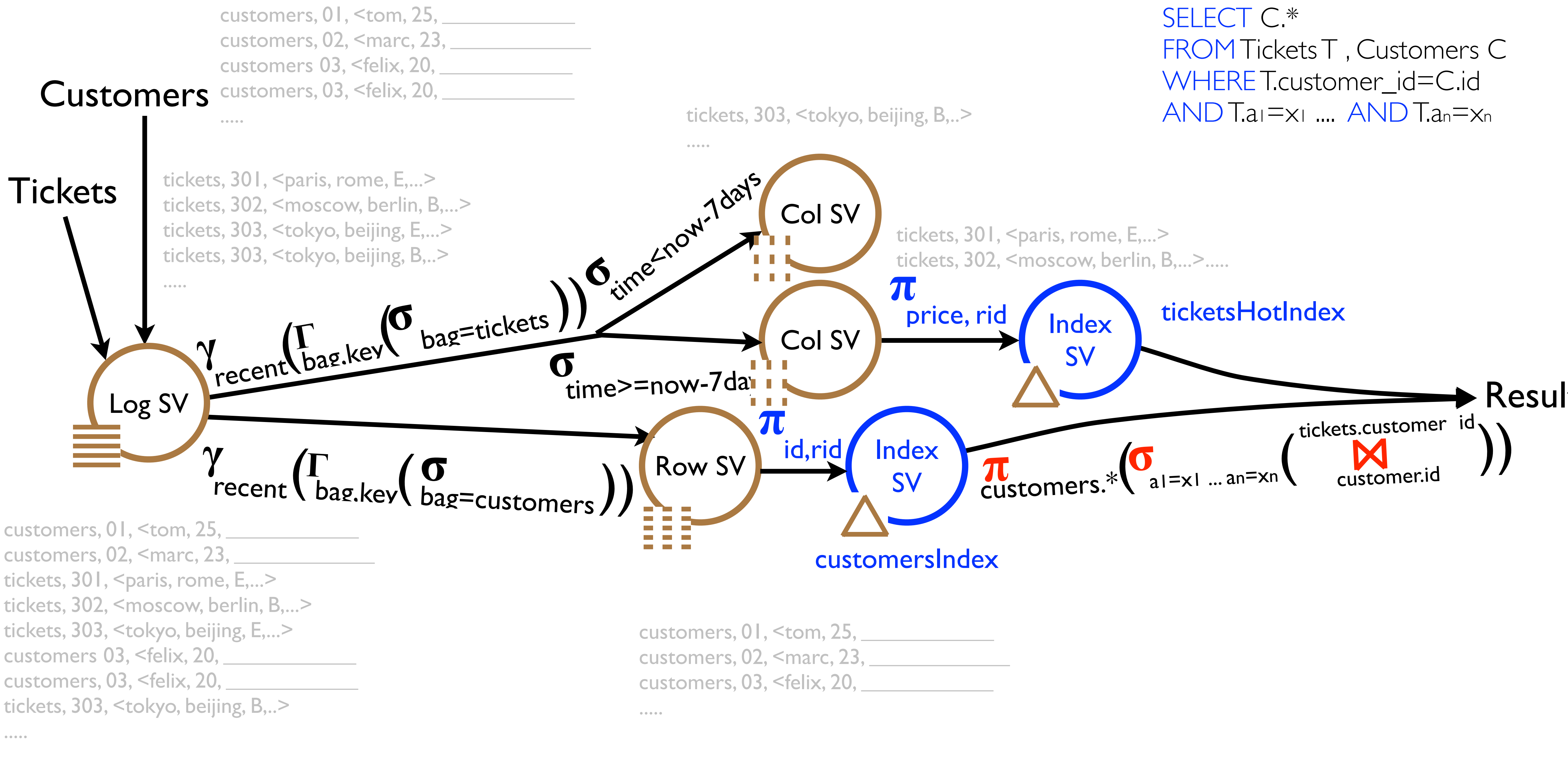
SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$



Index Storage Views

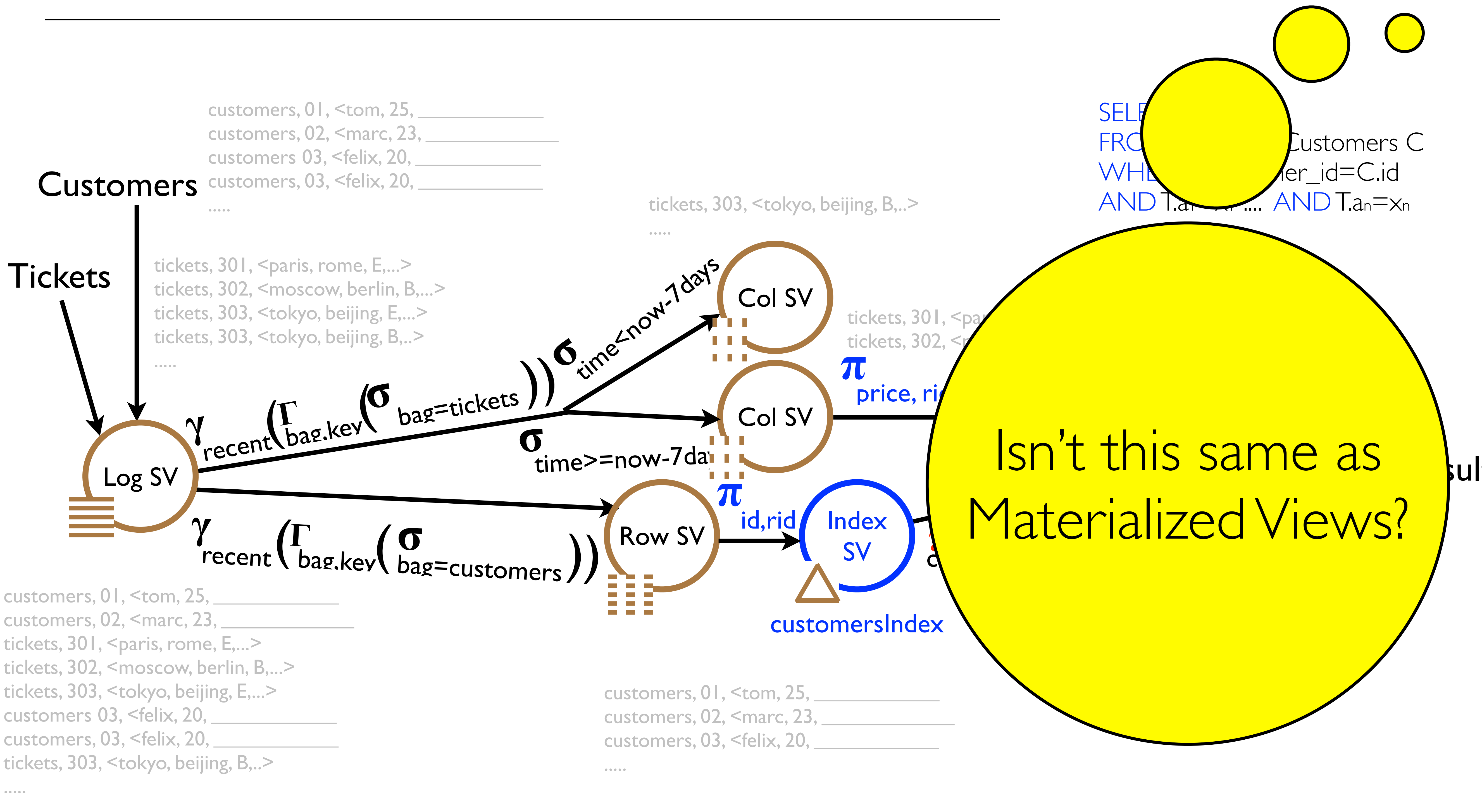
SAARLAND UNIVERSITY
Information Systems Group

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 .... AND T.an=xn

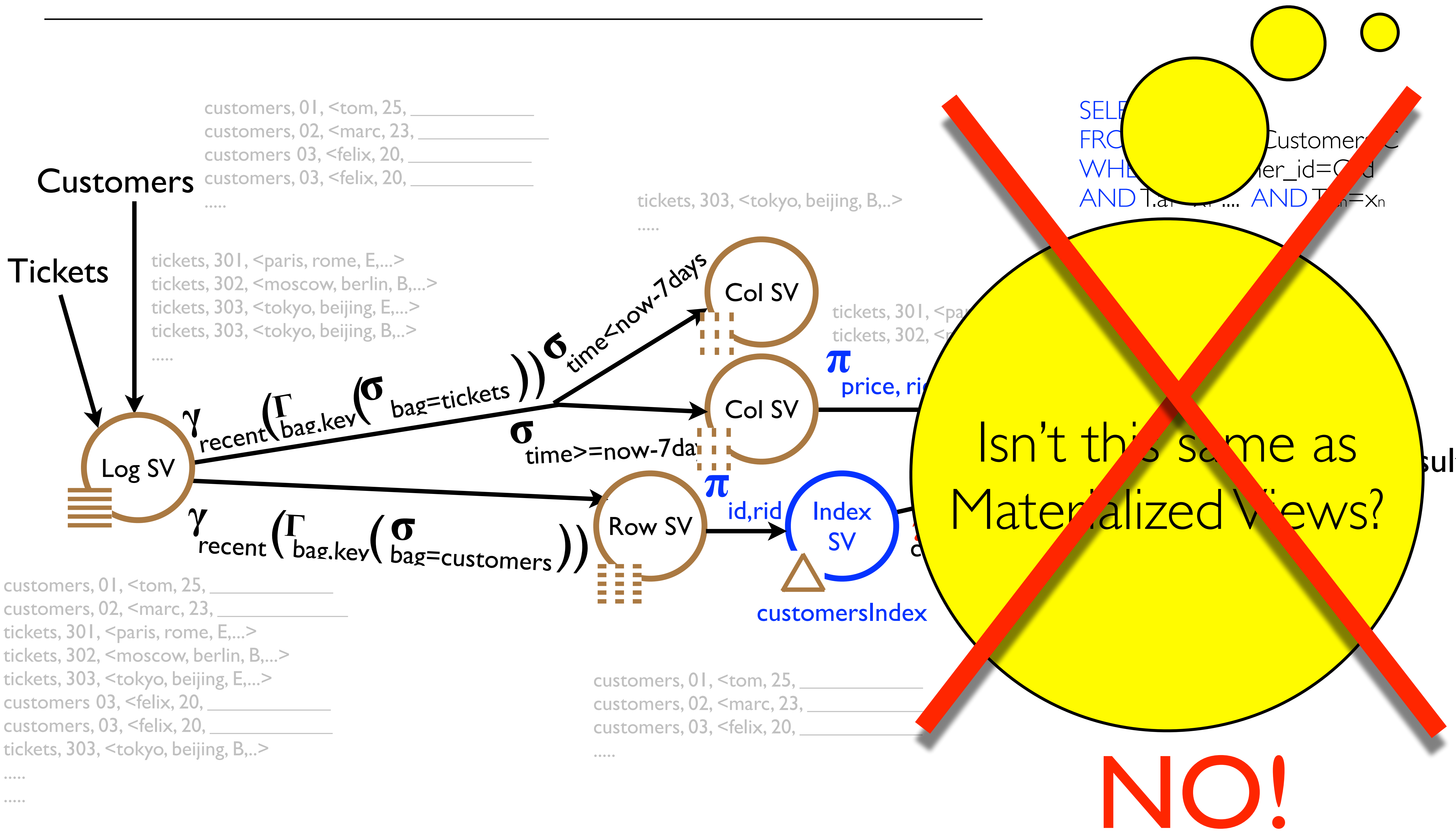


Storage Views

Index Storage Views



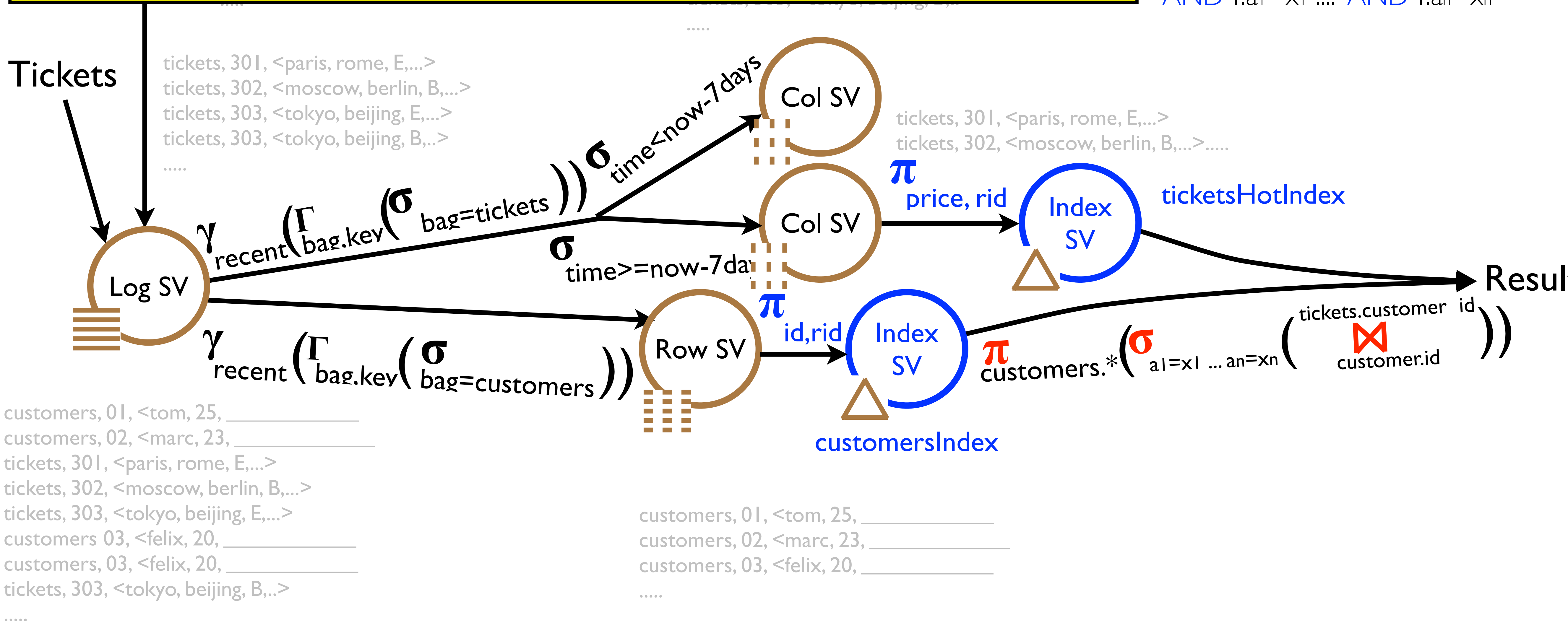
Index Storage Views



Index Storage Views

Materialized View knows what to materialize

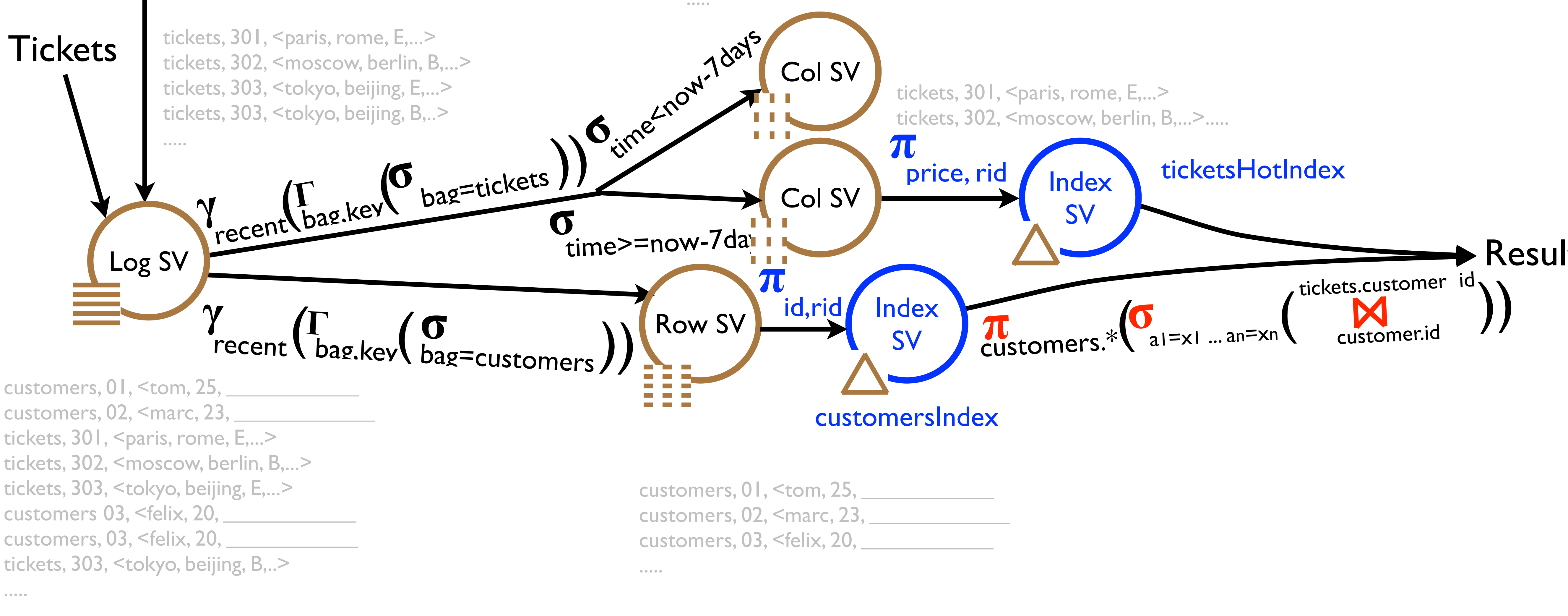
SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND T.a_1=x_1 .... AND T.a_n=x_n



Index Storage Views

Storage View also knows how to materialize

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$

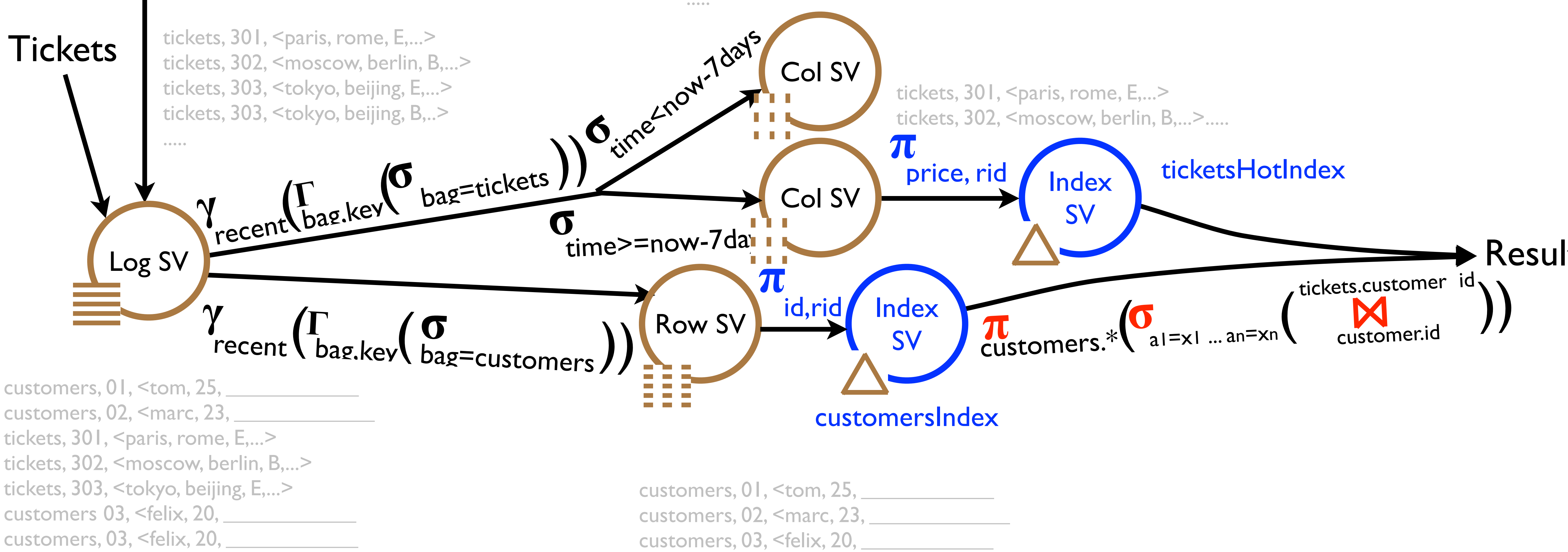


SAARLAND UNIVERSITY
Information Systems Group

Index Storage Views

Storage View also knows how to materialize

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$

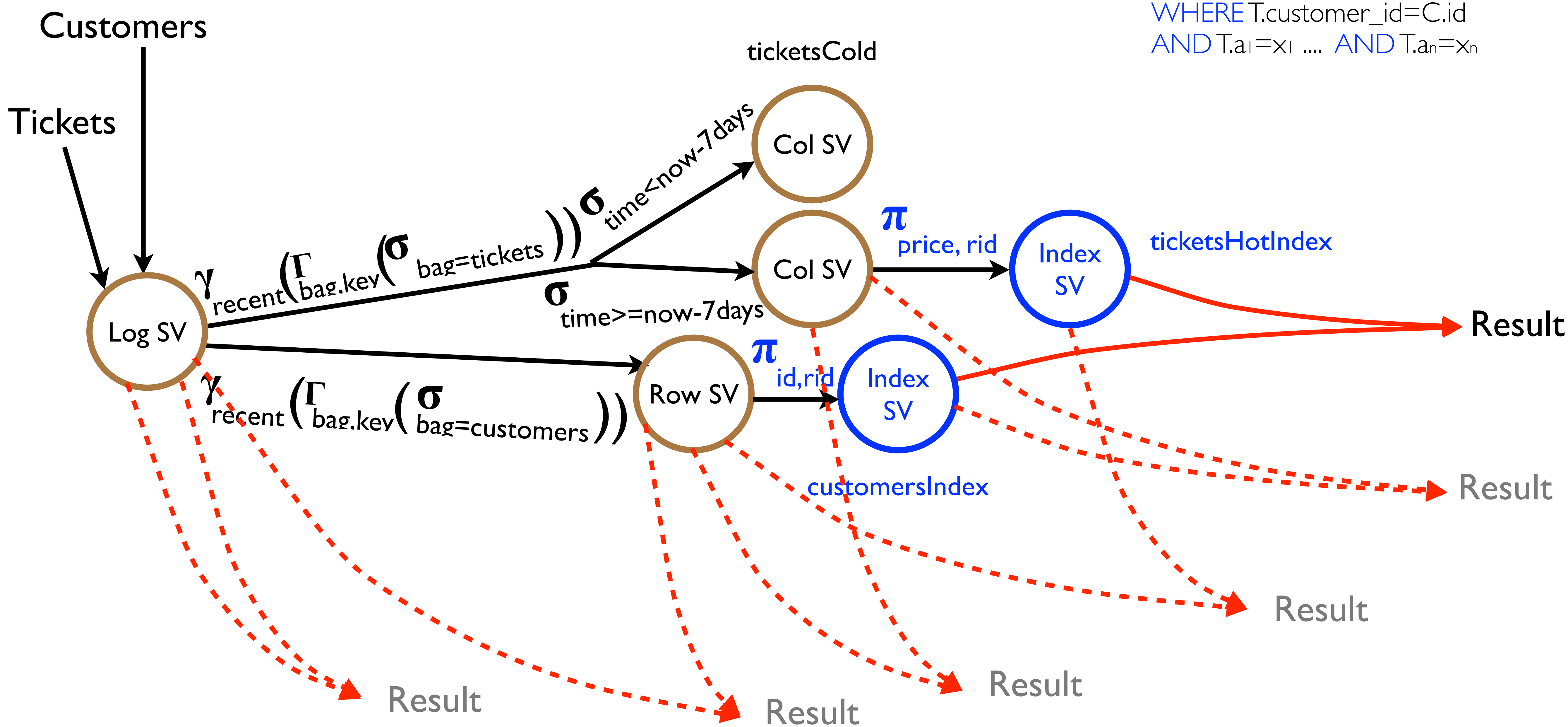


A Materialized View still needs a Storage View

Storage Views

Storage View Selection

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND T.a₁=x₁ .... AND T.aₙ=xₙ

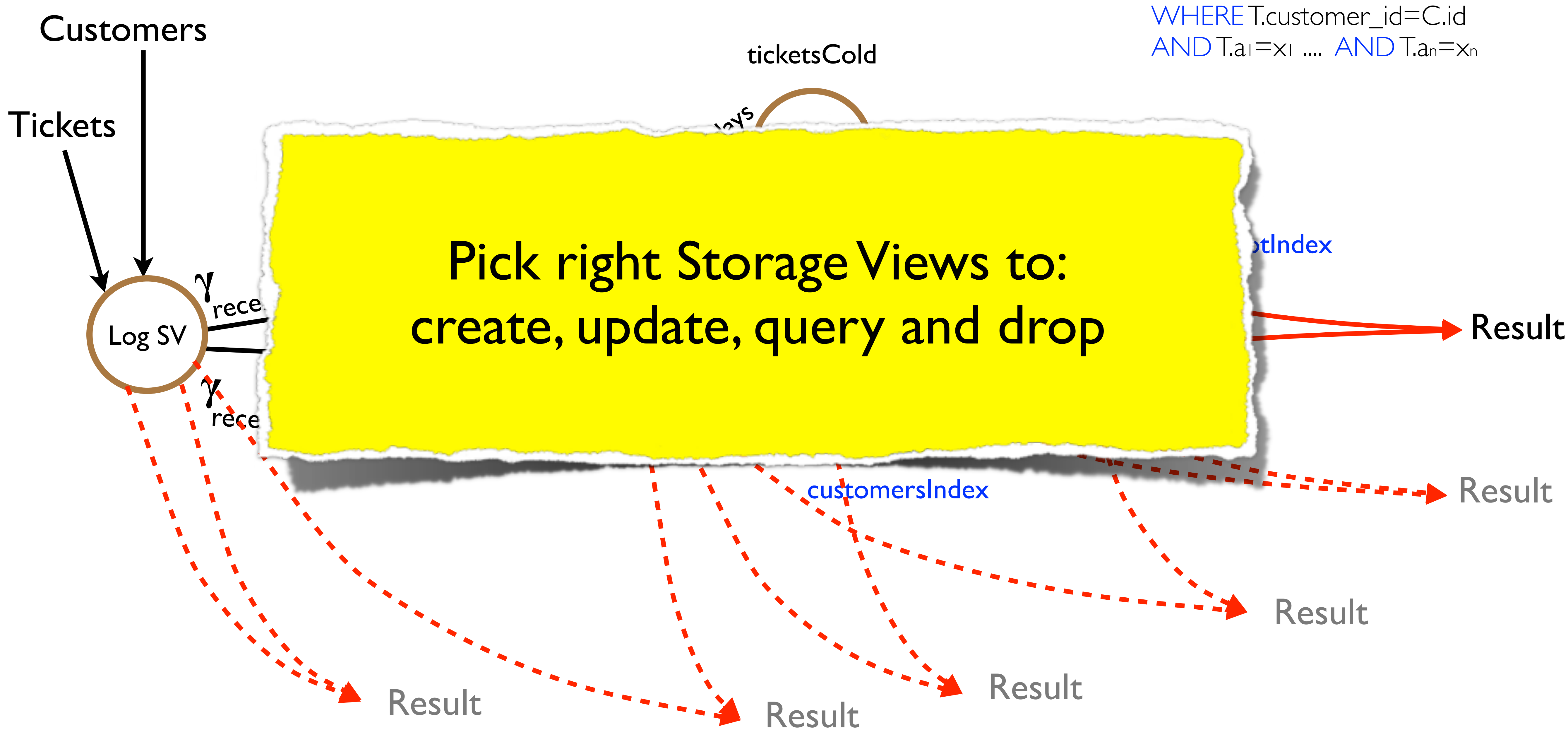


Optimizer

Storage View Selection

SAARLAND UNIVERSITY
Information Systems Group

```
SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 .... AND T.an=xn
```

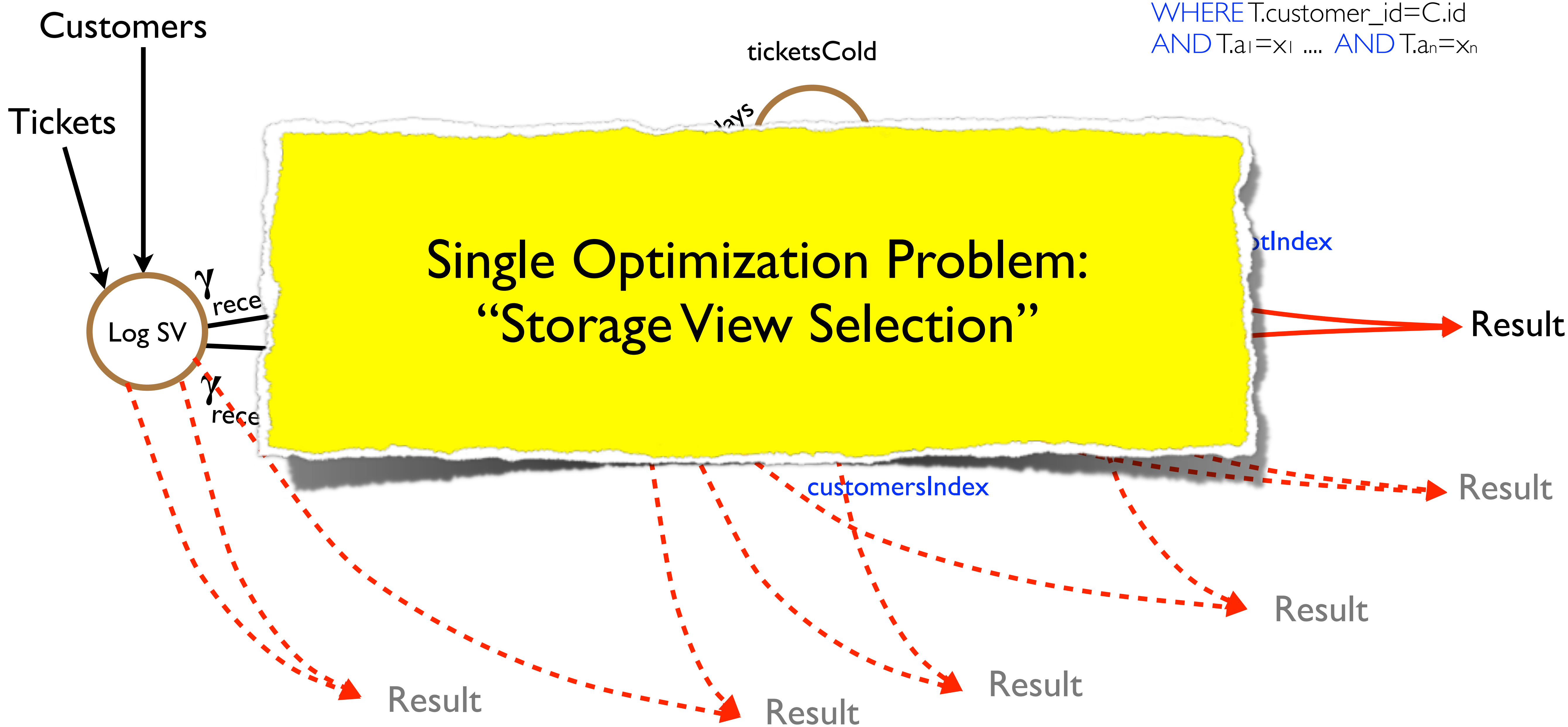


Pick right Storage Views to: create, update, query and drop

Optimizer

Storage View Selection

SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND $T.a_1=x_1$ .... AND $T.a_n=x_n$



Optimizer

Holistic Storage View Optimizer

- Storage totally dynamic:
  Any subset of data in ***Any*** storage structure
- Storage View selection
- Storage View update maintenance
- Pick physical execution plan
- Combine results spanning several Storage Views

Research Challenges

- Single umbrella for different storage layouts
  - storage layer abstraction
  - still layout specific specialization
- Automatic adaptive bifurcation
  - monolithic system
  - right online algorithms
- Simplicity vs Optimization
  - only as complex as required
  - mimic several specialized systems

Related Work

- Materialized Views [Chirkova et. al. VLDBJ 2002]
  - as pointed before different from storage views
- Dynamic materialized views [Zhou et. al. ICDE 2007]
  - horizontal dynamism, storage view still open
- View matching, query containment [A.Y. Halevy VLDBJ 2001]
  - again operate on a higher level
- Cracked databases [Idreos et. al. CIDR 2007]
  - logical partitioning of data, only horizontal
- Rodent store [Cudre-Mauroux et. al. CIDR 2009]
  - still assumes a store
- GMAP [Tsatalos et. al. VLDB 1994]
  - does not adapt the stores

Optimizer Cost Model

SAARLAND UNIVERSITY
Information Systems Group

Query Cost Model

Symbol	Meaning	Model
$C^{\text{log}}_{\text{scan}}(N)$	Log SV scan cost	$\left\lceil \frac{\sum_{i=1}^{N} colsize(log_i)}{m} \right\rceil \cdot C_{\text{random}} + \left\lceil \frac{\sum_{i=1}^{N} colsize(log_i)}{pageSize} \right\rceil / \text{BW}$
$C^{\text{row}}_{\text{scan}}(N)$	Row SV scan cost	$\left\lceil \frac{N \cdot \sum_{A_i \in A} colsize(A_i)}{m} \right\rceil \cdot C_{\text{random}} + \left\lceil \frac{N \cdot \sum_{A_i \in A} colsize(A_i)}{pageSize} \right\rceil / \text{BW}$
$C^{\text{col}}_{\text{scan}}(N, S)$	Col SV scan cost	$\sum_{A_i \in S} \left(\left\lceil \frac{N \cdot \sum_{A_i \in S} colsize(A_i)}{m} \right\rceil \cdot C_{\text{random}} + \left\lceil \frac{N \cdot colsize(A_i)}{pageSize} \right\rceil / \text{BW} \right)$
$C^{\text{index}}_{\text{lookup}}(N)$	Index lookup cost	$C_{\text{random}} \cdot \lceil log_F(N \cdot (colsize(key) + \text{pointerSize})/\text{pageSize}) \rceil$
$C^{\text{row cl. index}}_{\text{scan}}(N, sel)$	Unclustered Indexed Row SV scan cost	$C^{\text{index}}_{\text{lookup}}(N) + C^{\text{row}}_{\text{scan}}(\lceil sel \cdot N \rceil)$
$C^{\text{col. cl. index}}_{\text{scan}}(N, S, sel)$	Unclustered Indexed Col SV scan cost	$C^{\text{index}}_{\text{lookup}}(N) + C^{\text{col}}_{\text{scan}}(\lceil sel \cdot N \rceil, S)$
$C^{\text{row uncl. index}}_{\text{scan}}(N, sel)$	Clustered Indexed Row SV scan cost	$C^{\text{index}}_{\text{lookup}} + \lceil sel \cdot N \rceil \cdot (C_{\text{random}} + \text{pageSize/BW})$
$C^{\text{col. uncl. index}}_{\text{scan}}(N, S, sel)$	Unclustered Indexed Col SV scan cost	$C^{\text{index}}_{\text{lookup}} + \lceil sel \cdot N \rceil \cdot \lvert S \rvert \cdot (C_{\text{random}} + \text{pageSize/BW})$

Update Cost Model

Symbol	Meaning	Model
$C^{\text{log}}_{\text{update}}(N_u)$	Log SV update cost	$C^{\text{log}}_{\text{scan}}(N_u)$
$C^{\text{row}}_{\text{update}}(N, N_u)$	Row SV update cost	$\min \left(C_{\text{random}} + \left\lceil \frac{N}{N_c} \right\rceil \cdot C^{\text{row}}_{\text{scan}}(2 \cdot N_c), \left\lceil \frac{N}{N_c} \right\rceil \cdot C^{\text{row}}_{\text{scan}}(N_c) + N_u \cdot (C_{\text{random}} + \text{pageSize/BW}) \right)$
$C^{\text{col}}_{\text{update}}(N, N_u, S)$	Col SV update cost	$\min \left(C_{\text{random}} + \left\lceil \frac{N}{N_c} \right\rceil \cdot C^{\text{col}}_{\text{scan}}(2 \cdot N_c), \left\lceil \frac{N}{N_c} \right\rceil \cdot C^{\text{col}}_{\text{scan}}(N_c) + N_u \cdot \lvert S \rvert \cdot (C_{\text{random}} + \text{pageSize/BW} \right)$
$C^{\text{index}}_{\text{split}}(d)$	Index split cost	$\left(\sum_{i=1}^{d} (p_{\text{split}})^i \right) \cdot C_{\text{random}}$
$C^{\text{row cl. index}}_{\text{update}}(N, N_u, d)$	Cl. Index Row SV update cost	$C^{\text{index}}_{\text{lookup}}(N) + 2 \cdot C^{\text{row}}_{\text{scan}}(N_u) + C^{\text{index}}_{\text{split}}(d)$
$C^{\text{col. cl. index}}_{\text{update}}(N, N_u, S, d)$	Cl. Index Col SV update cost	$C^{\text{index}}_{\text{lookup}}(N) + 2 \cdot C^{\text{col}}_{\text{scan}}(N_u, S) + C^{\text{index}}_{\text{split}}(d)$
$C^{\text{row uncl. index}}_{\text{update}}(N, N_u, d)$	Uncl. Index Row SV update cost	$C^{\text{index}}_{\text{lookup}} + N_u \cdot (C_{\text{random}} + pageSize/BW) + C^{\text{index}}_{\text{split}}(d)$
$C^{\text{col. uncl. index}}_{\text{update}}(N, N_u, S, d)$	Uncl. Index Col SV update cost	$C^{\text{index}}_{\text{lookup}} + N_u \cdot \lvert S \rvert \cdot (C_{\text{random}} + pageSize/BW) + C^{\text{index}}_{\text{split}}(d)$

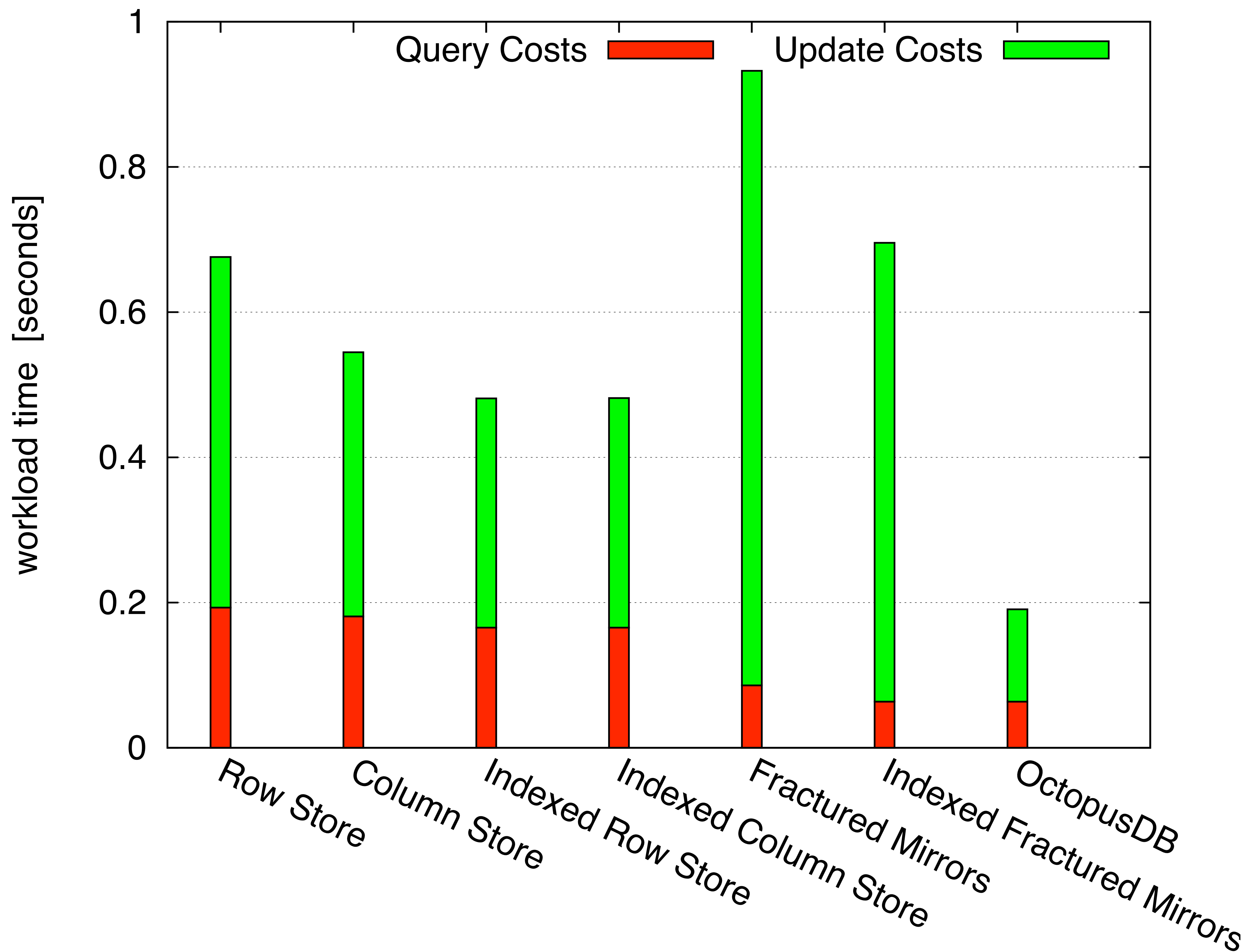
Transform Cost Model

SV Transformation	Cost
Log SV $\rightarrow$ Row SV	$C^{\text{log}}_{\text{scan}}(N) + C^{\text{row}}_{\text{scan}}(N)$
Log SV $\rightarrow$ Col SV	$C^{\text{log}}_{\text{scan}}(N) + C^{\text{col}}_{\text{scan}}(N, A)$
Row SV $\leftrightarrow$ Col SV	$C^{\text{row}}_{\text{scan}}(N) + C^{\text{col}}_{\text{scan}}(N, A)$
Row SV $\rightarrow$ Index SV	$C^{\text{row}}_{\text{scan}}(N) + \left(\frac{F^{d+1} - 1}{F - 1} \right) \cdot C_{\text{random}}$
Col SV $\rightarrow$ Index SV	$C^{\text{col}}_{\text{scan}}(N, \{\text{key,rowID}\}) + \left(\frac{F^{d+1} - 1}{F - 1} \right) \cdot C_{\text{random}}$

Further Directions

Comparing Different Stores

	Tickets	Customers
Tuples	100,000	20,000
Selectivity	0.9	0.1
Attributes Referenced	4/20	20/20



Further Directions

Next Steps

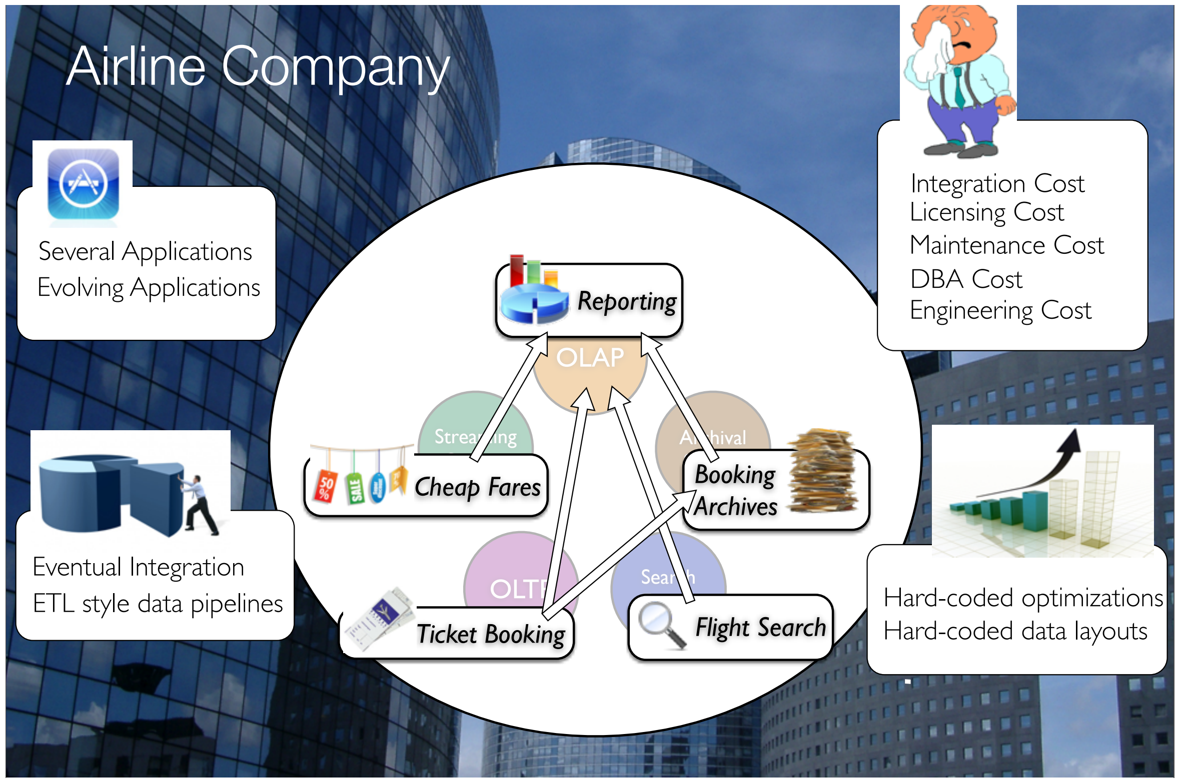
SAARLAND UNIVERSITY
Information Systems Group

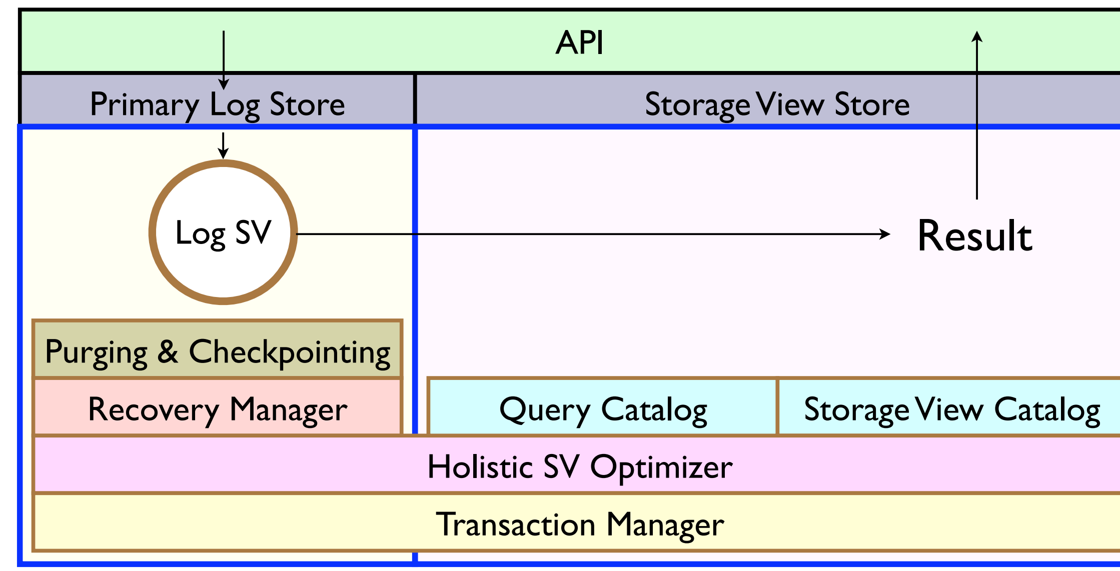
1. Automatically picking the right layout
   - row, column, partitioned, cracked, more?
2. Storage View compression
   - adaptive compression
3. Storage View maintenance
   - maintaining heterogenous SVs
4. OctopusDB benchmarking and evaluation
   - one-size-fits-all benchmark

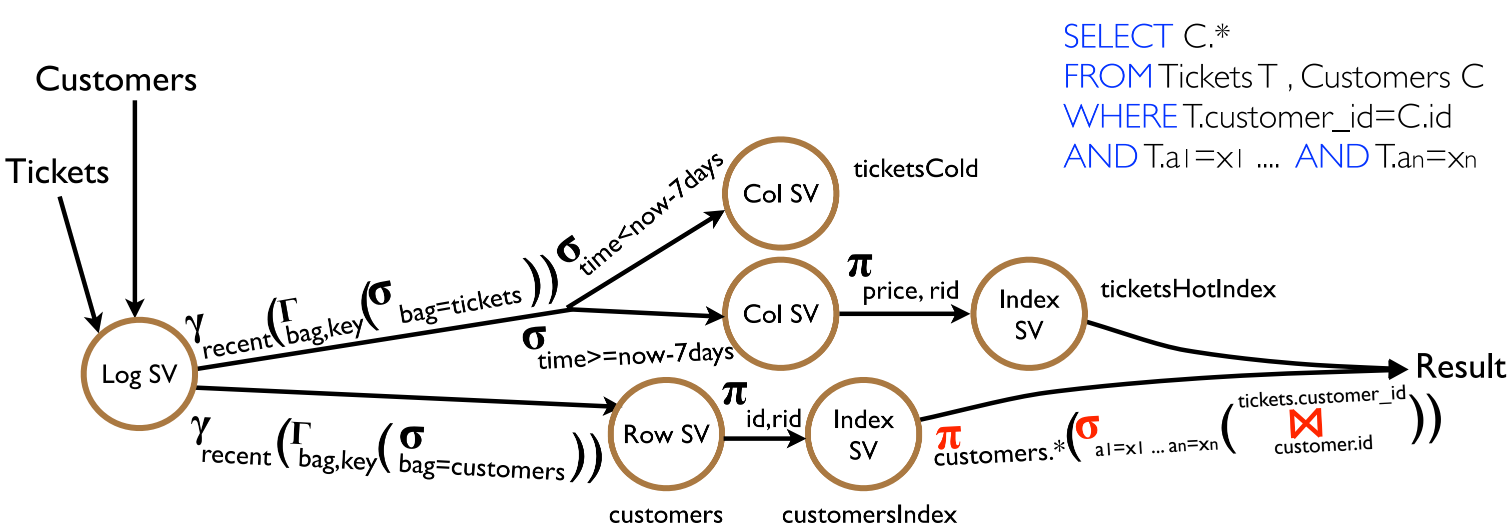
Further Directions

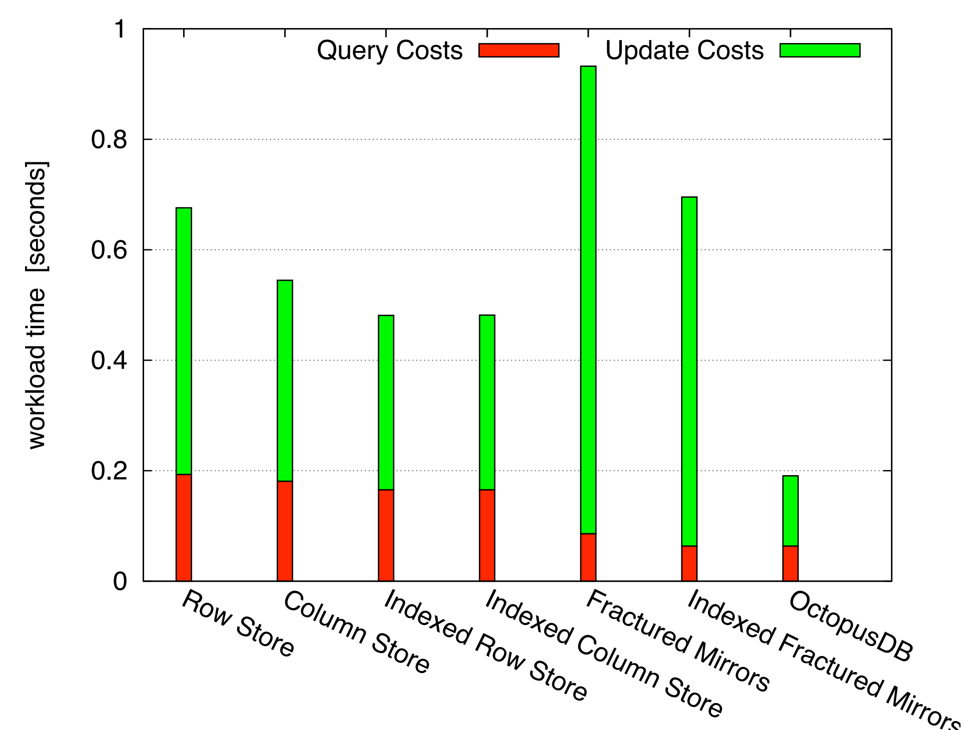
SAARLAND UNIVERSITY
Information Systems Group

Summary









Thanks!