SAARLAND UNIVERSITY

COMPUTER SCIENCE

Information Systems Group

OctopusDB

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

Alekh Jindal

Supervisor: Prof. Dr. Jens Dittrich

May 31, 2010

Database Landscape

OLTP

Streaming System

OLAP

Archival System

Search Engine

Database Landscape

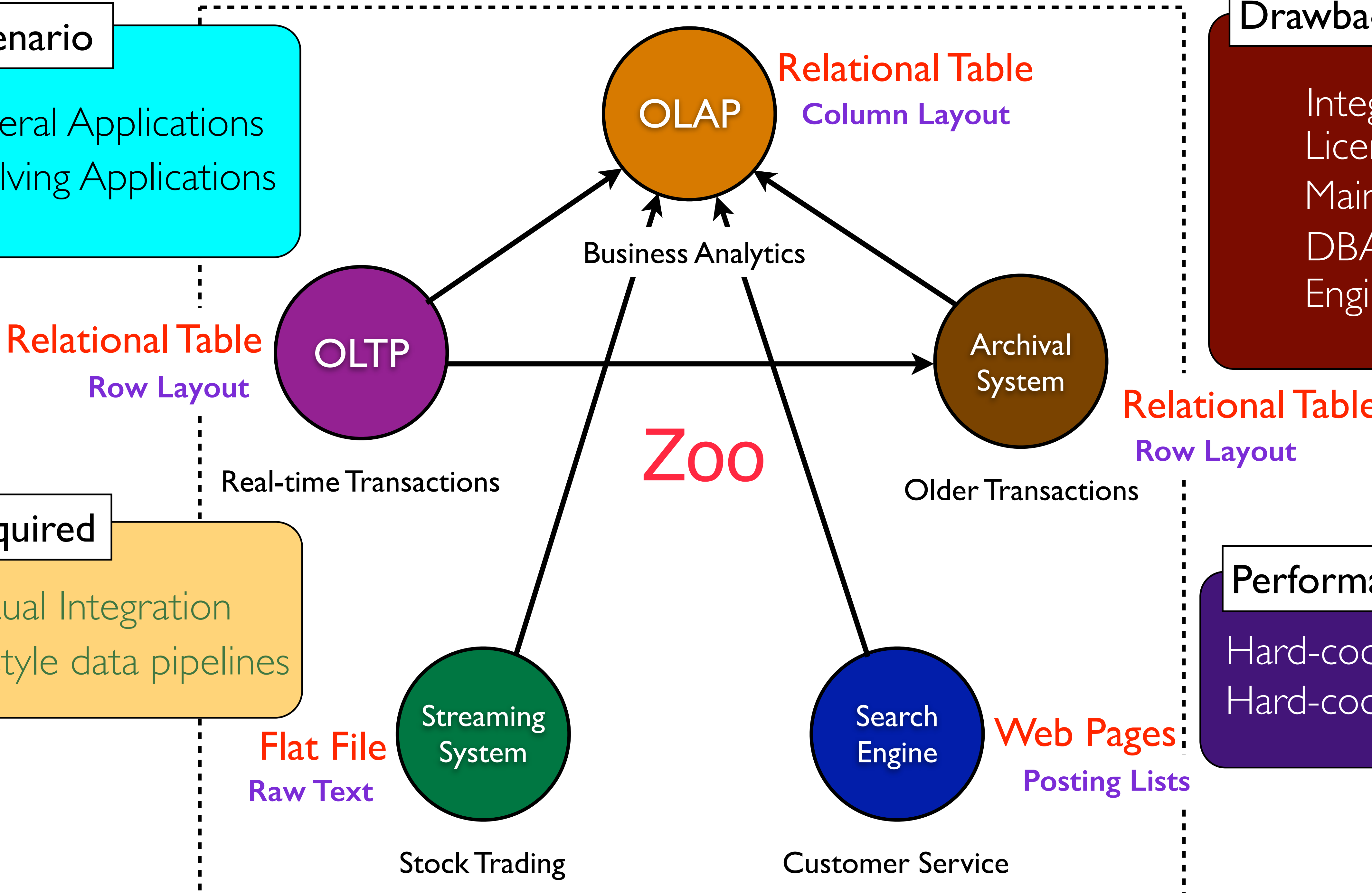
Example: Banking

Scenario

Several Applications
Evolving Applications

Required

Eventual Integration
ETL style data pipelines



Drawbacks

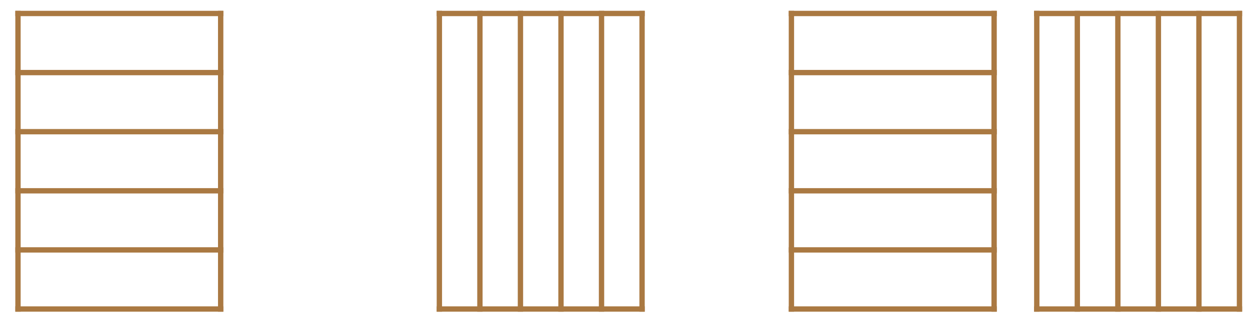
Integration Cost
Licensing Cost
Maintenance Cost
DBA Cost
Engineering Cost

Performance

Hard-coded optimizations
Hard-coded data layouts

Hard-coded Data Layouts

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group



Workload

Type	Fraction of Attribute	Tuple Selectivity	Row	Column	Fractured Mirrors
Query	0.2	0.001	Bad	Good	Good
Query	1.0	0.1	Good	Bad	Good
Query	0.75	1.0	Bad	Bad	Bad
Update	1.0	0.1	Good	Bad	Bad

Hard-coded Data Layouts

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

Workload

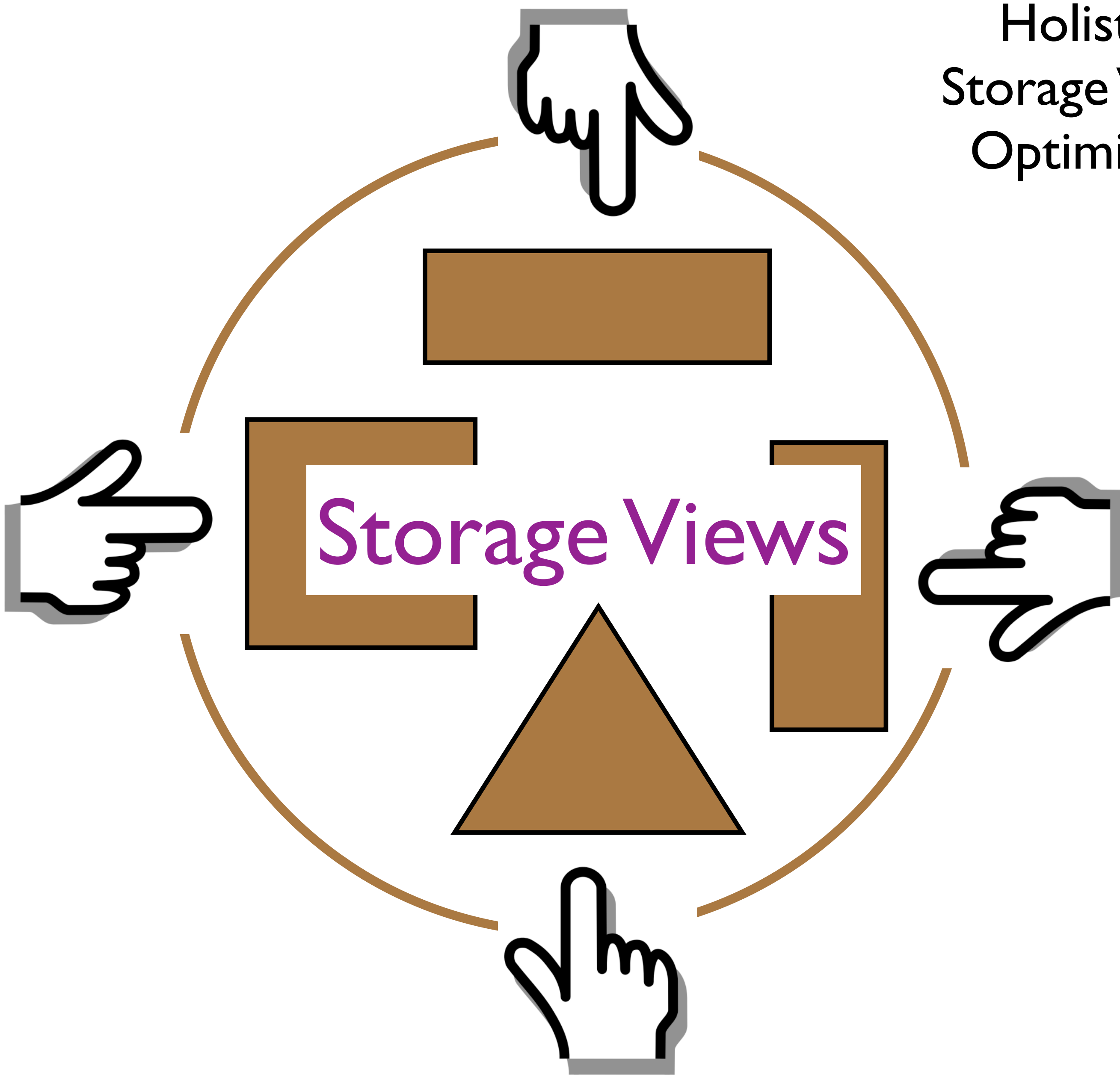
Type	Fraction of Attribute	Tuple Selectivity	Row	Col[illegible]	Fractured Mirrors
Query	0.2	[illegible]	[illegible]	[illegible]	Good
Query	[illegible]	[illegible]	[illegible]	Bad	Good
Query	[illegible]	[illegible]	Bad	Bad	Bad
Update	[illegible]	0.1	Good	Bad	Bad

Inflexibility

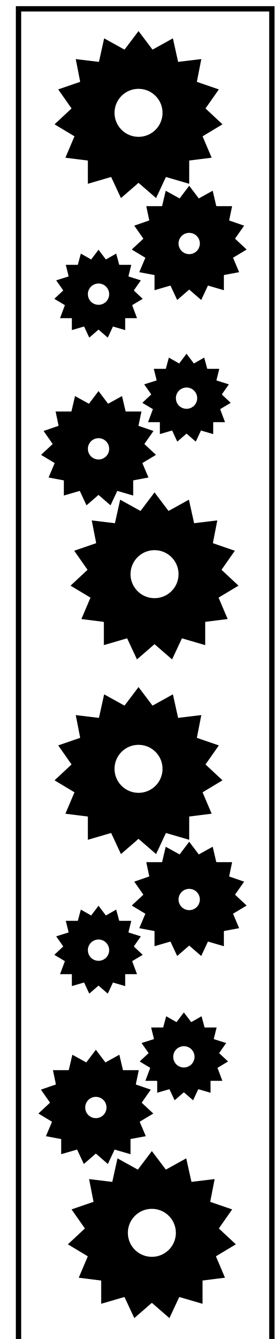
OctopusDB Core Idea

- No fixed store
- Why not have a flexible storage depending on the workload
- Pick the storage appropriate for the use-case
- Emulate a variety of systems

OctopusDB

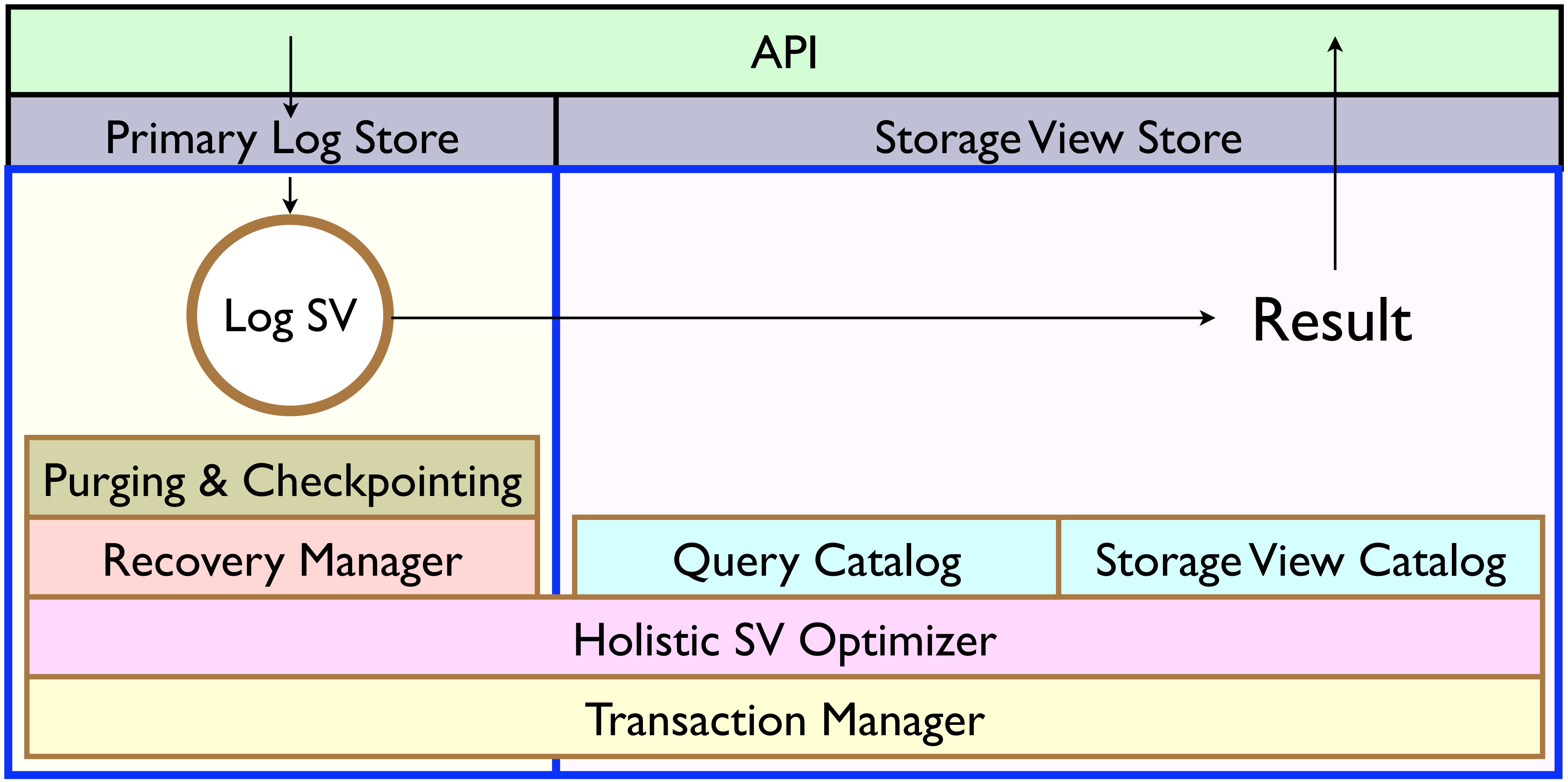


Holistic Storage View Optimizer



Mimic Several Systems

System Architecture



Storage Views

- No hard-coded store in OctopusDB
- All operations recorded in a primary log on stable storage using WAL
- Storage Views: arbitrary physical representations
- Different storage layouts under a single umbrella

Storage View Examples

Primary

- Log SV
- Row SV
- Column SV
- Index SV

Secondary

- Partial Index SV
- Bag-partitioned SV
- Key-consolidated 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 $T.a_1=x_1$ AND $T.a_2=x_2$ ... AND $T.a_n=x_n$

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

Flight Booking System

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

Customers

customers, 01, <tom, 25, tom@abc.com, ...>
customers, 02, <marc, 23, marc@abc.com, ...>
customers 03, <felix, 20, felix@abc.com, ...>
customers, 03, <felix, 20, felix@xyz.com, ...>
.....

Tickets

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

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

Log SV

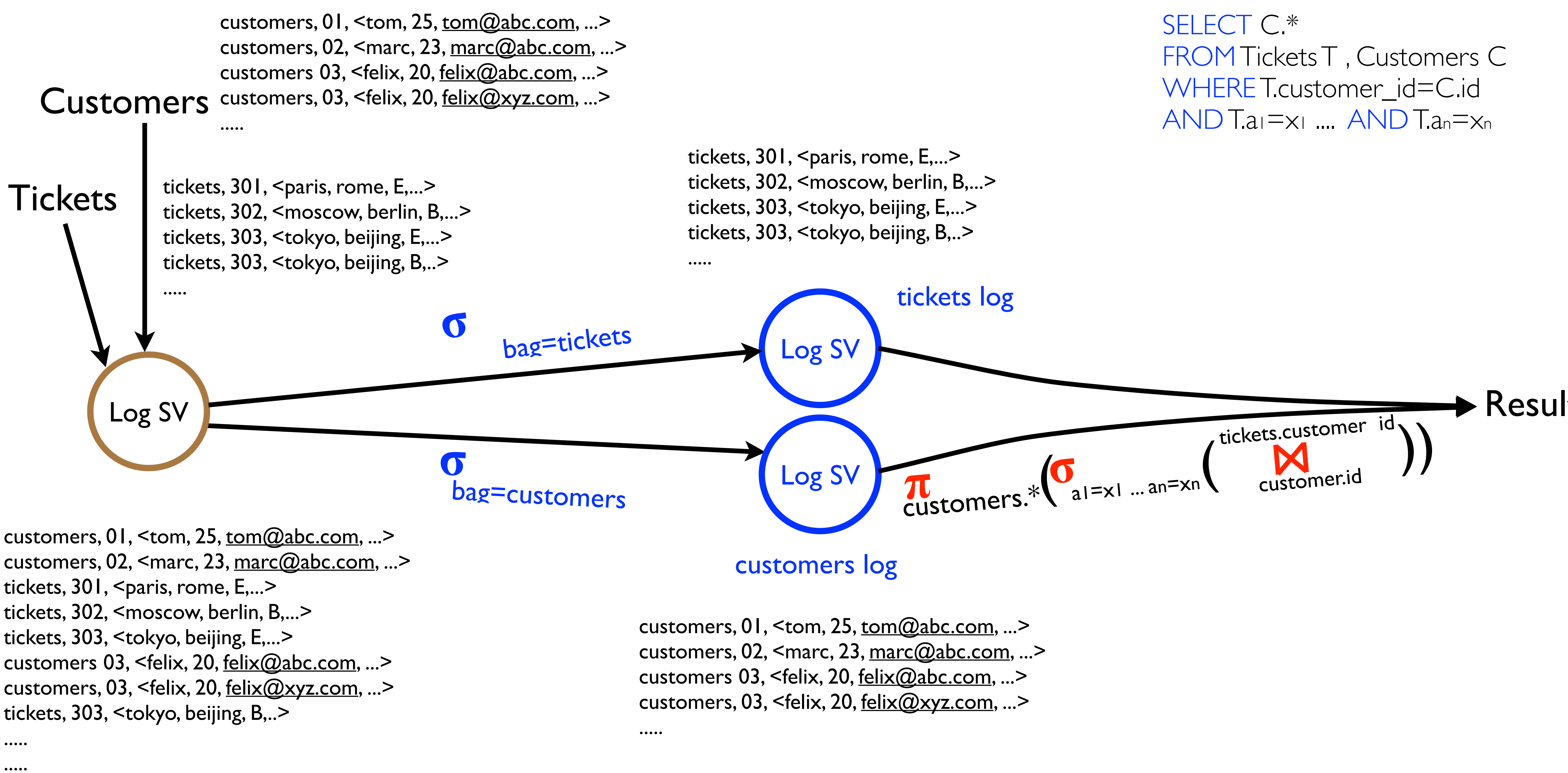
$$\pi_{customer.*}\left(\sigma_{a_1=x_1 \;....\; a_n=x_n}\left(\bowtie_{customers.id}^{tickets.customer\ id}\right)\right)$$

Result

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,..>
.....
.....

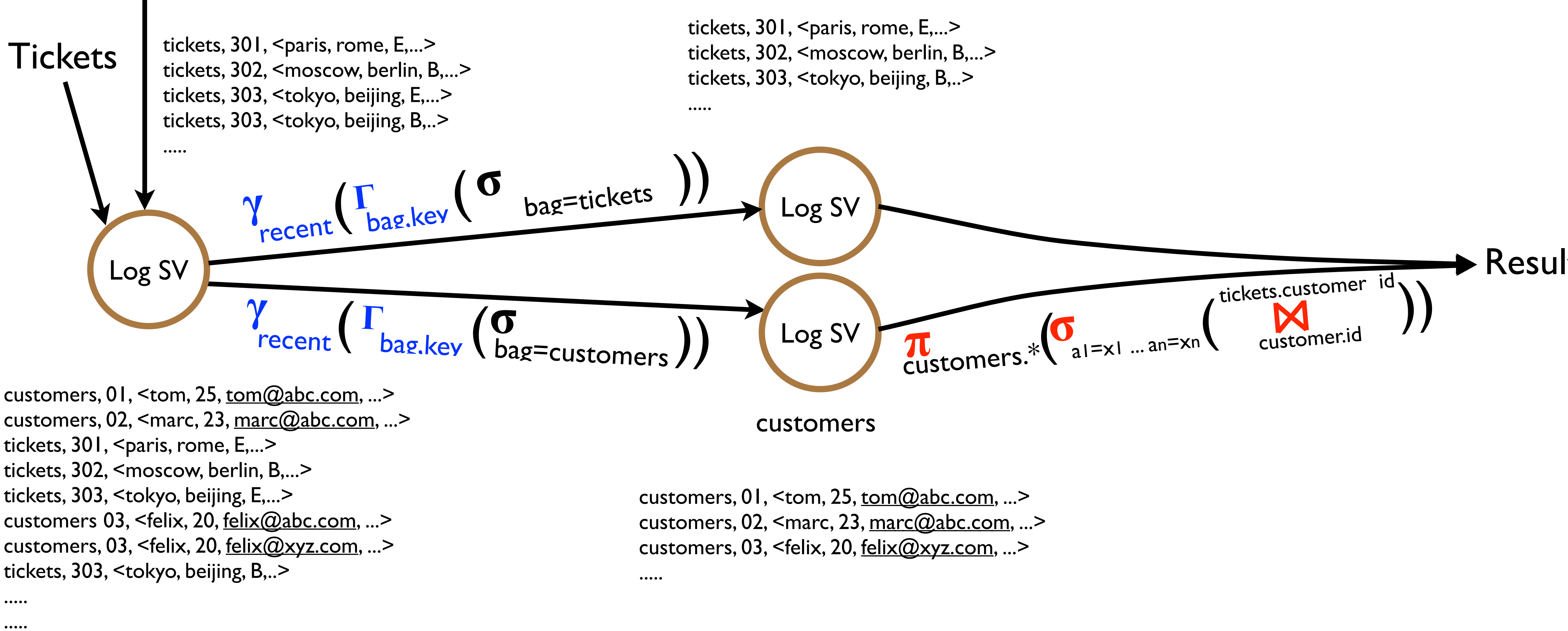
Bag-partitioning

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



Key-consolidation

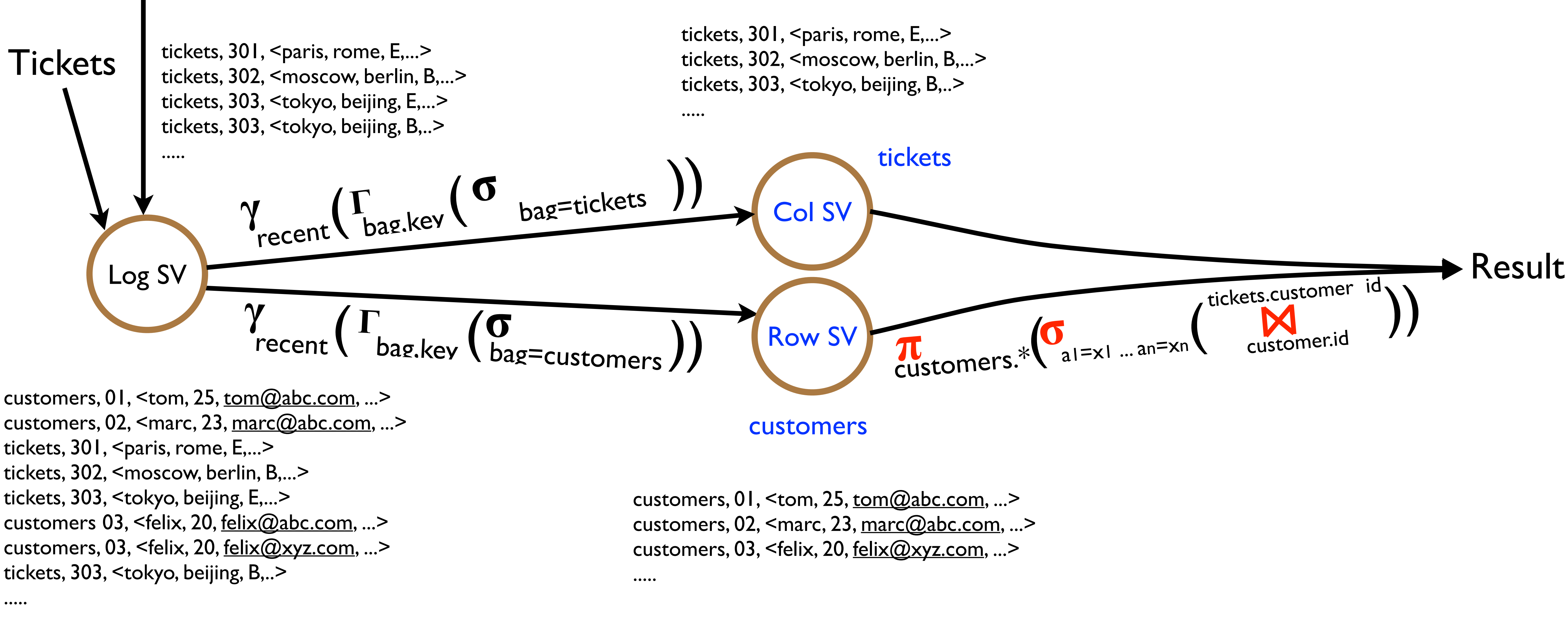
SELECT C.*
FROM Tickets T , Customers C
WHERE T.customer_id=C.id
AND T.a_1=x_1 .... AND T.a_n=x_n



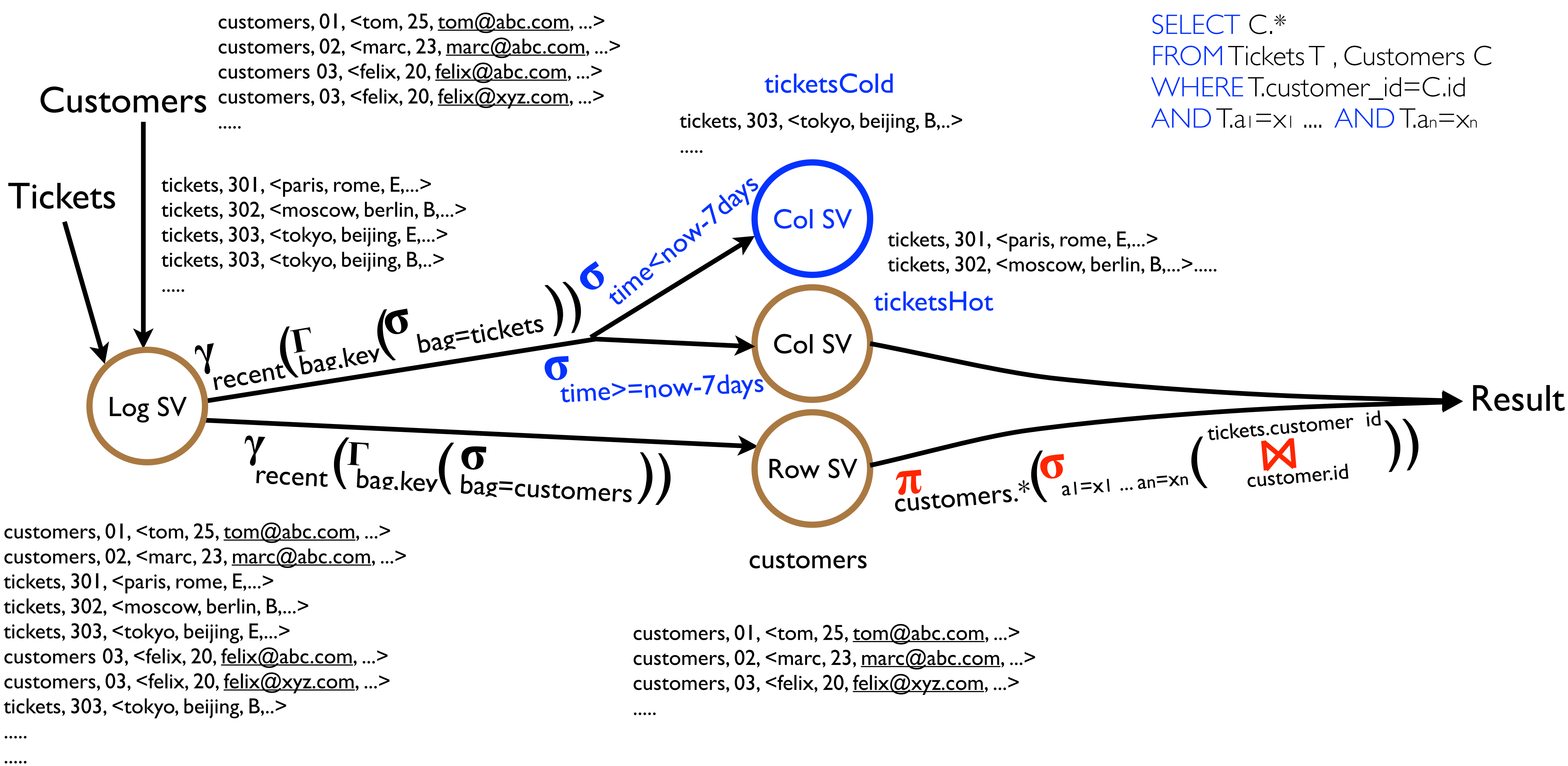
Storage View Transformation

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

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



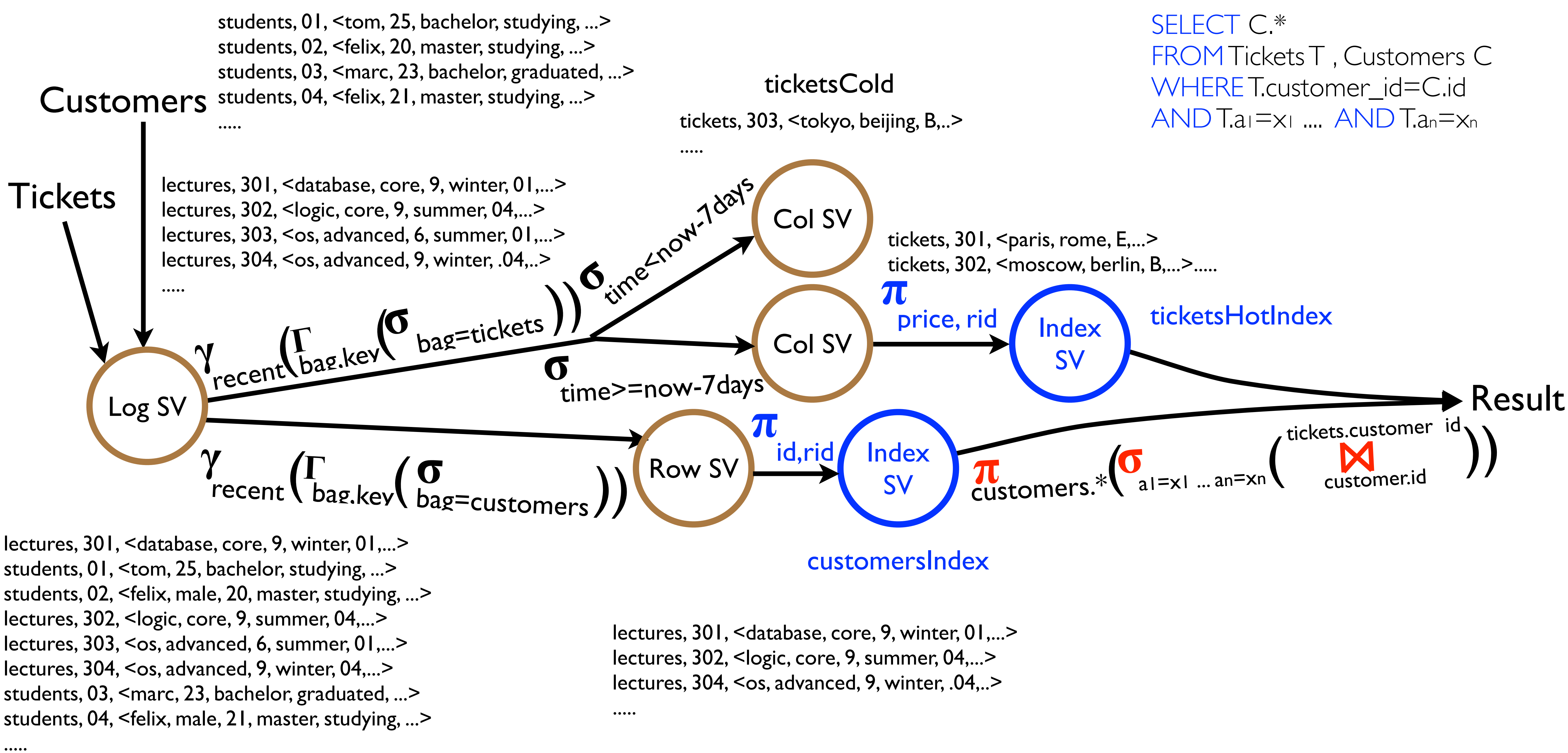
Hot-Cold Storage Views



Index Storage Views

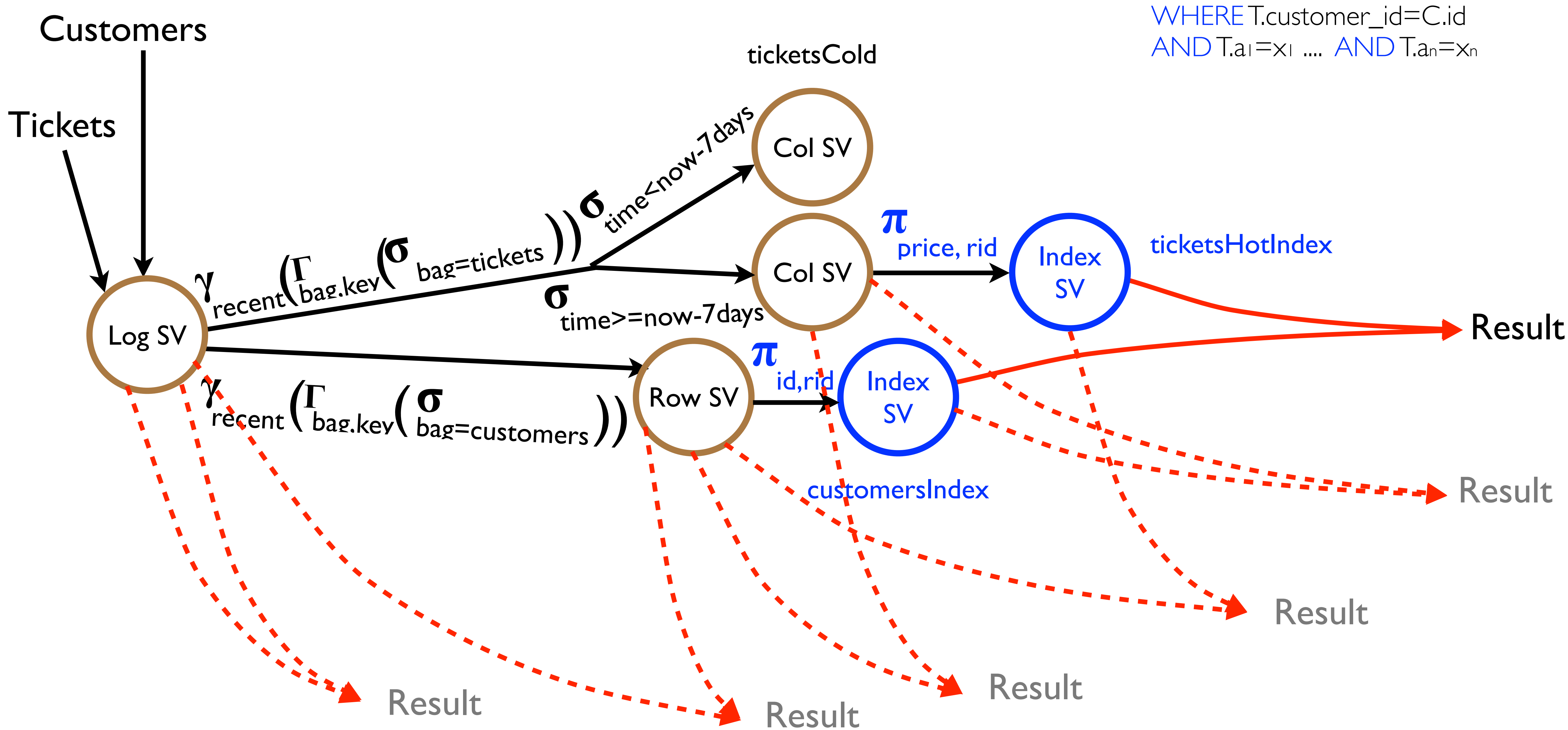
SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

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

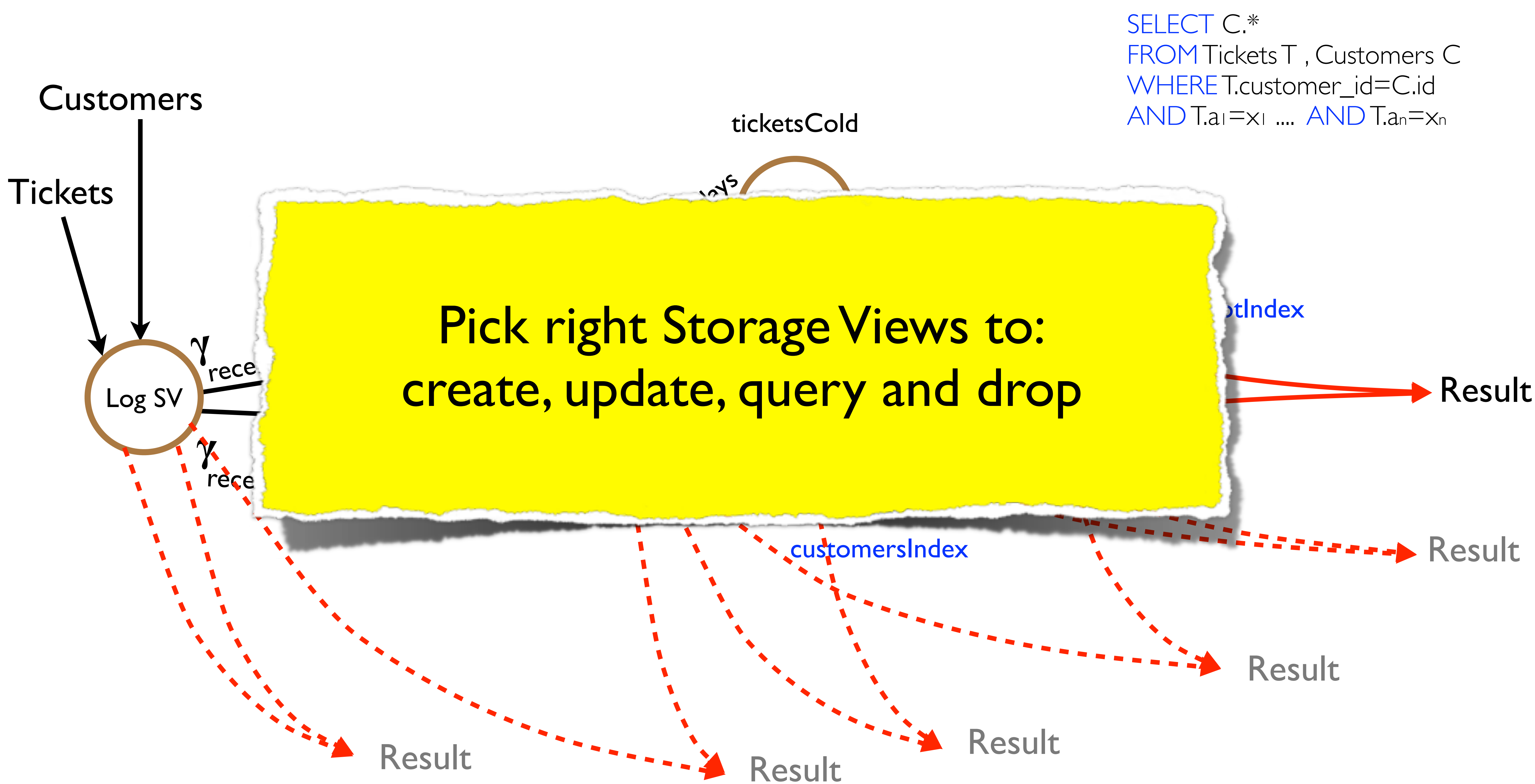


Storage View Selection

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



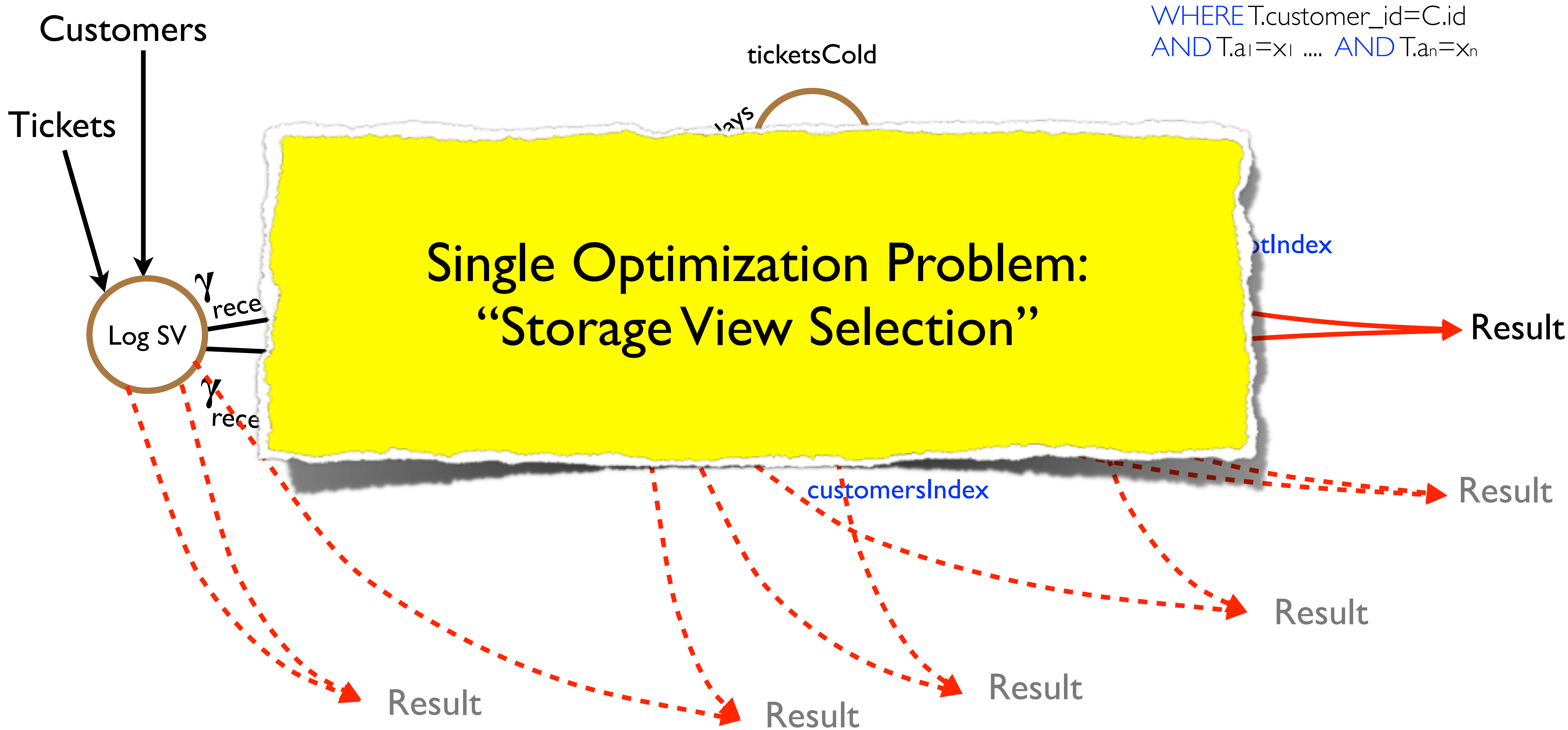
Storage View Selection



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

Storage View Selection

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



Holistic Storage View Optimizer

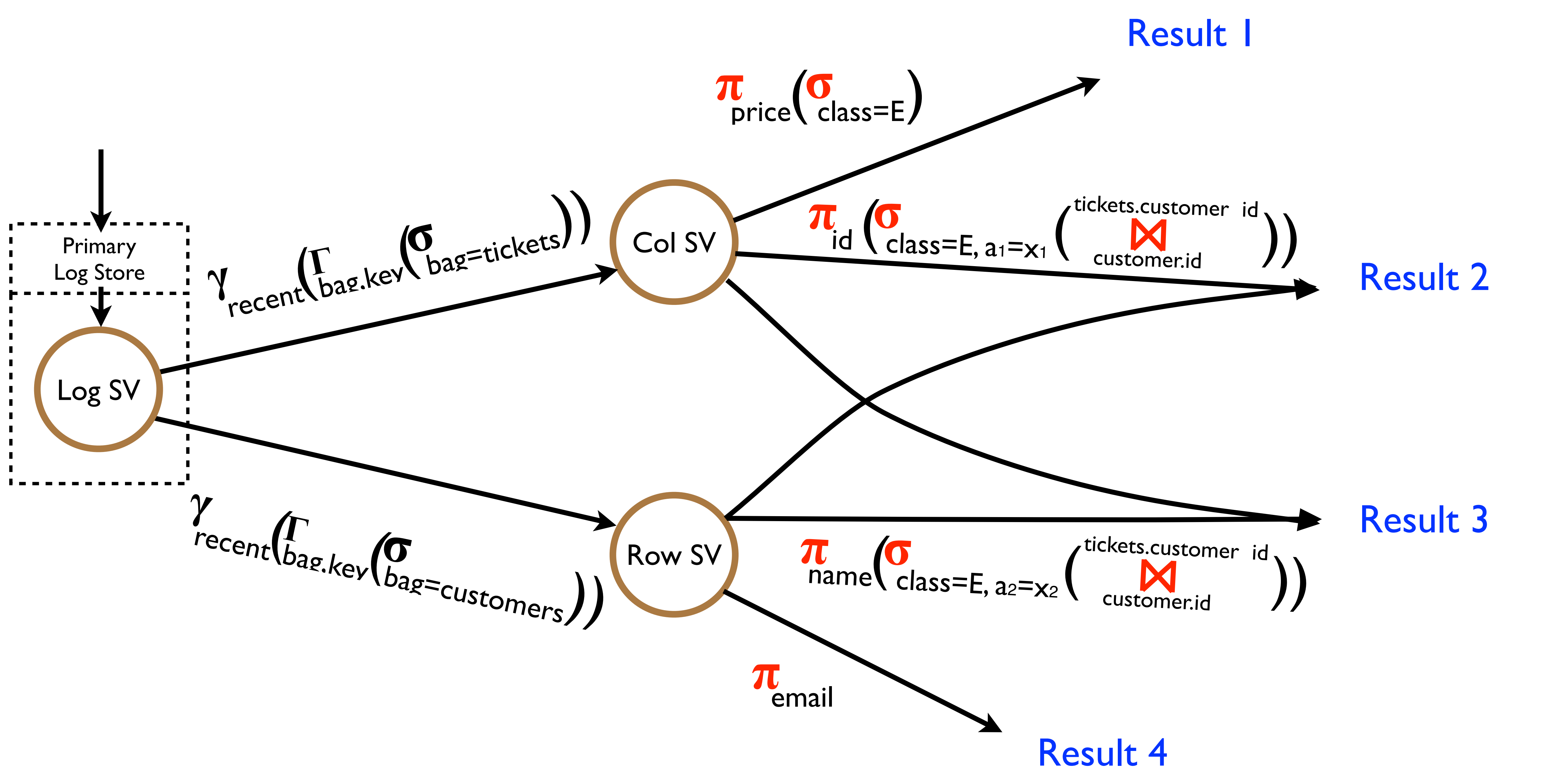
- Storage totally dynamic:
  Any subset of data in ***any*** storage structure
- Storage View selection
- Storage View update maintenance
- Pick physical execution plan
- Combine results spanning several Storage Views

Adaptive SV Optimization

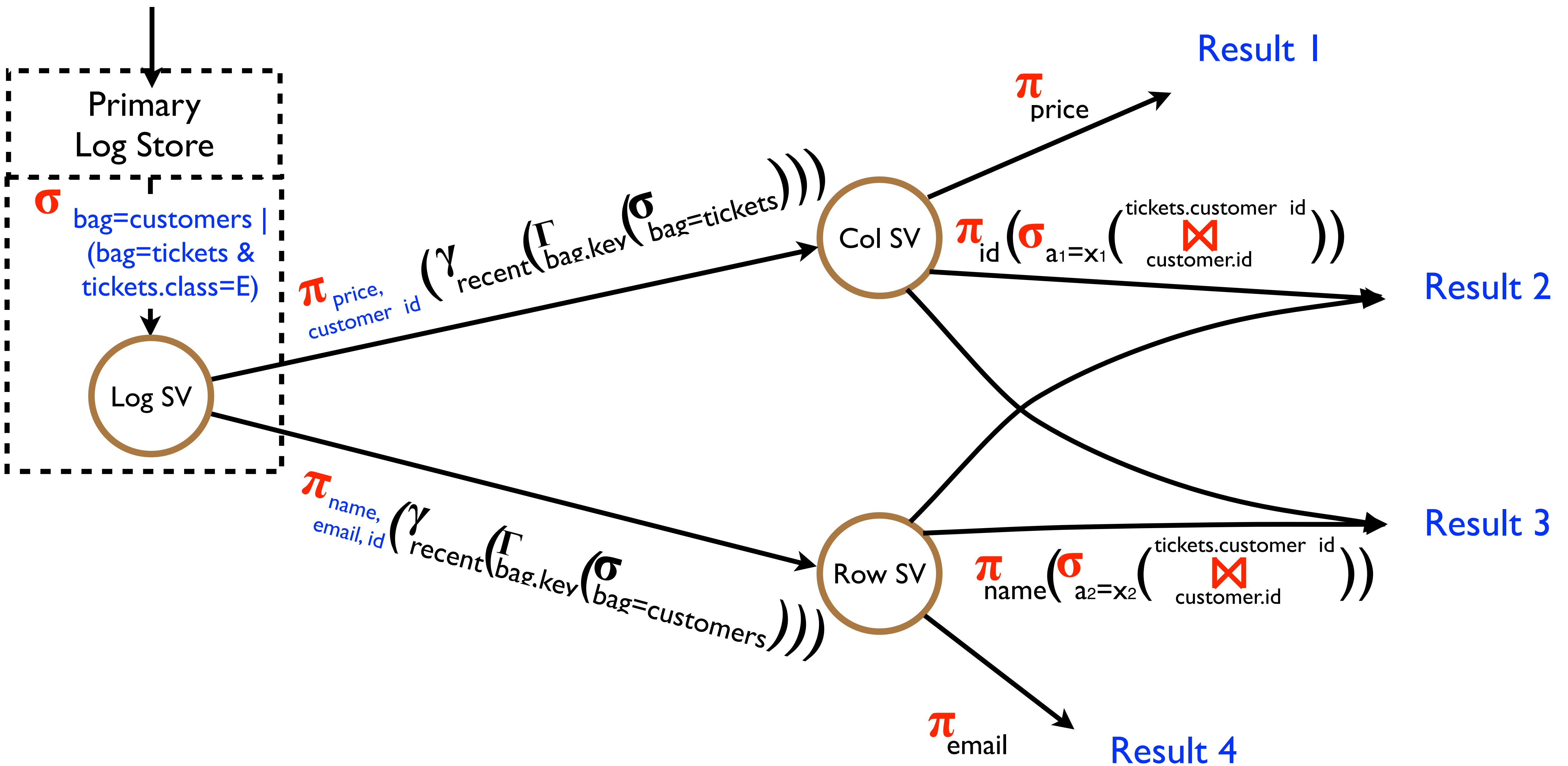
SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

- SV Rearrangement
  - e.g. Operator log-pushdown
- Adaptive partial SVs
  - e.g. Partial Indexes
- Stream transformation
  - e.g. OLTP to Streaming System

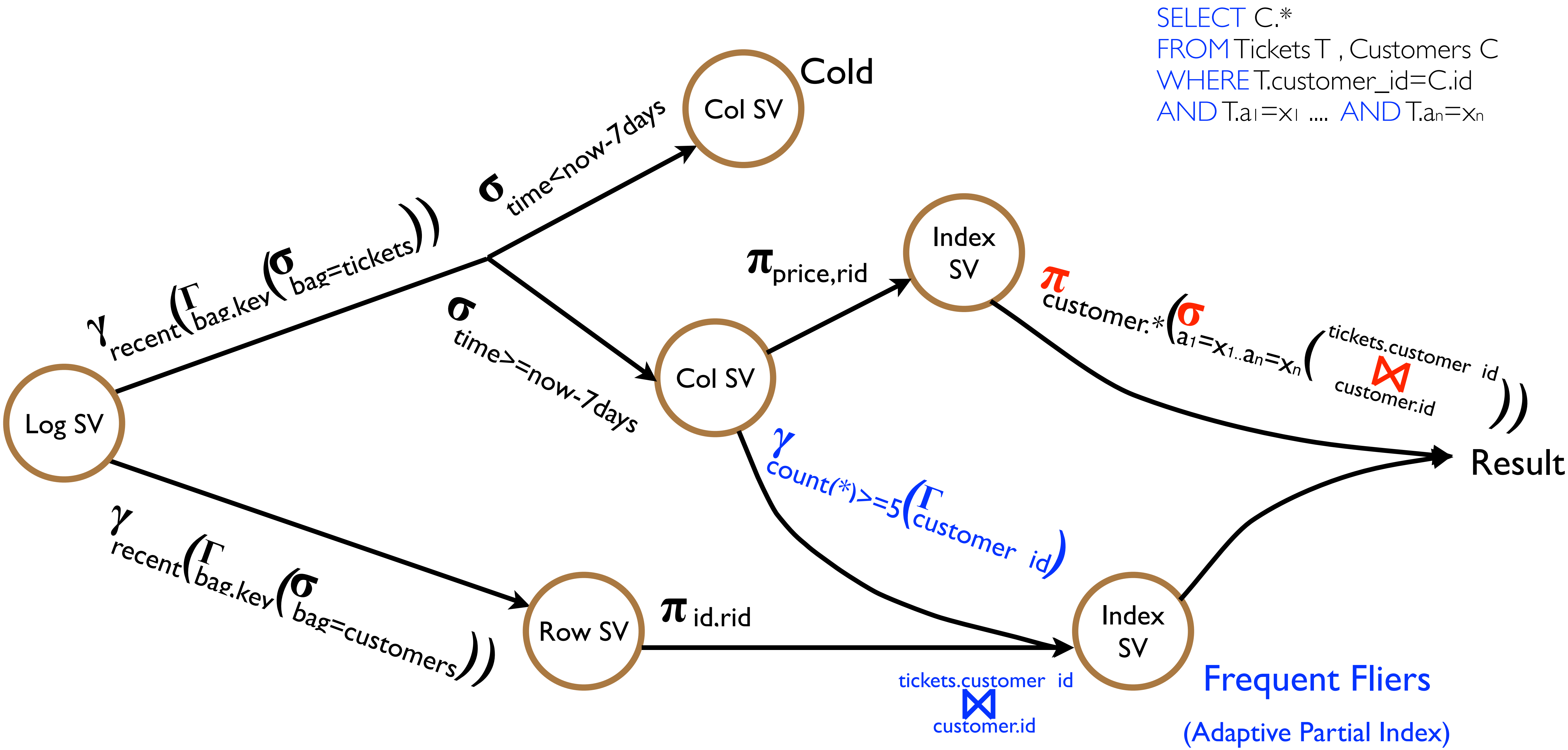
Operator Log-Pushdown



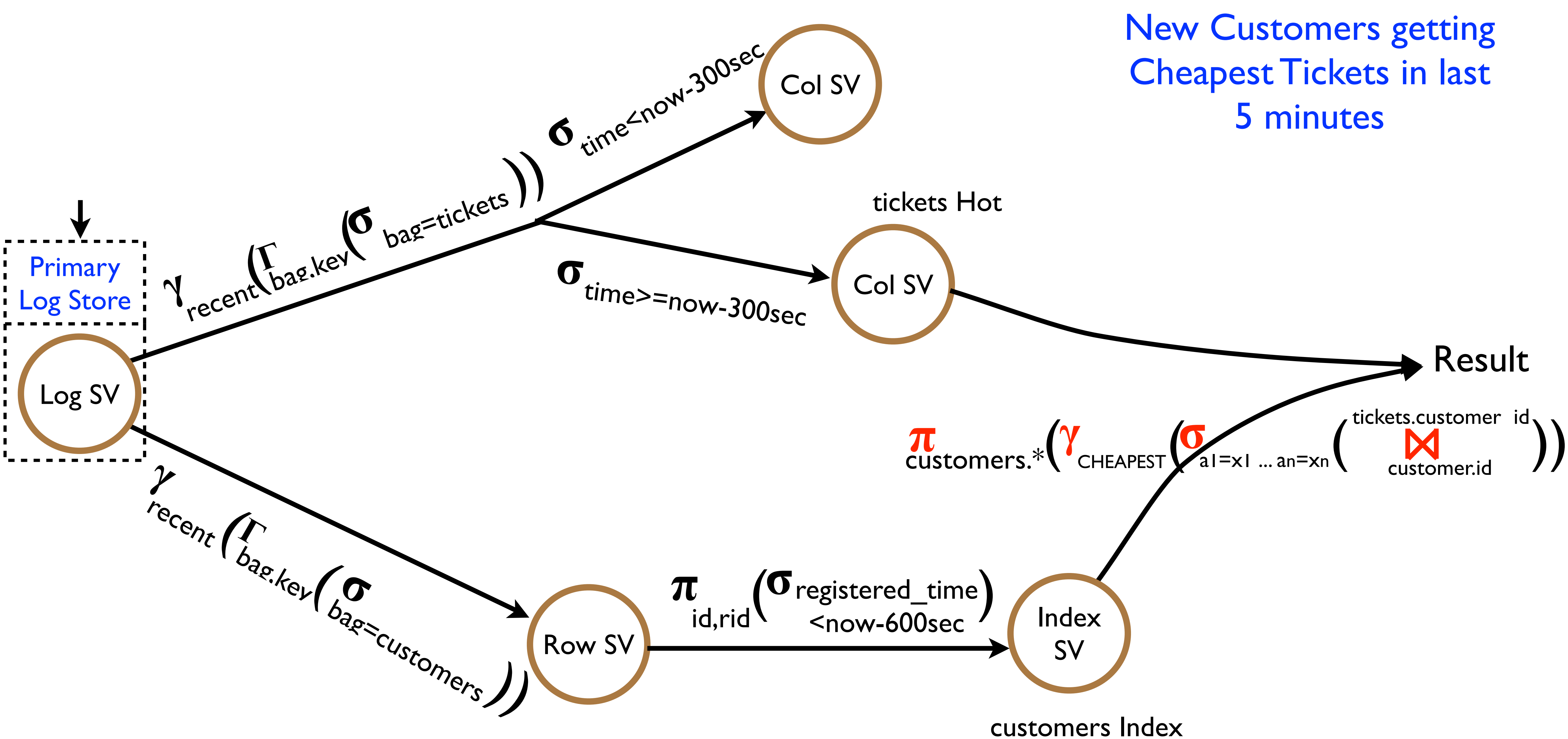
Operator Log-Pushdown



Adaptive Partial SVs

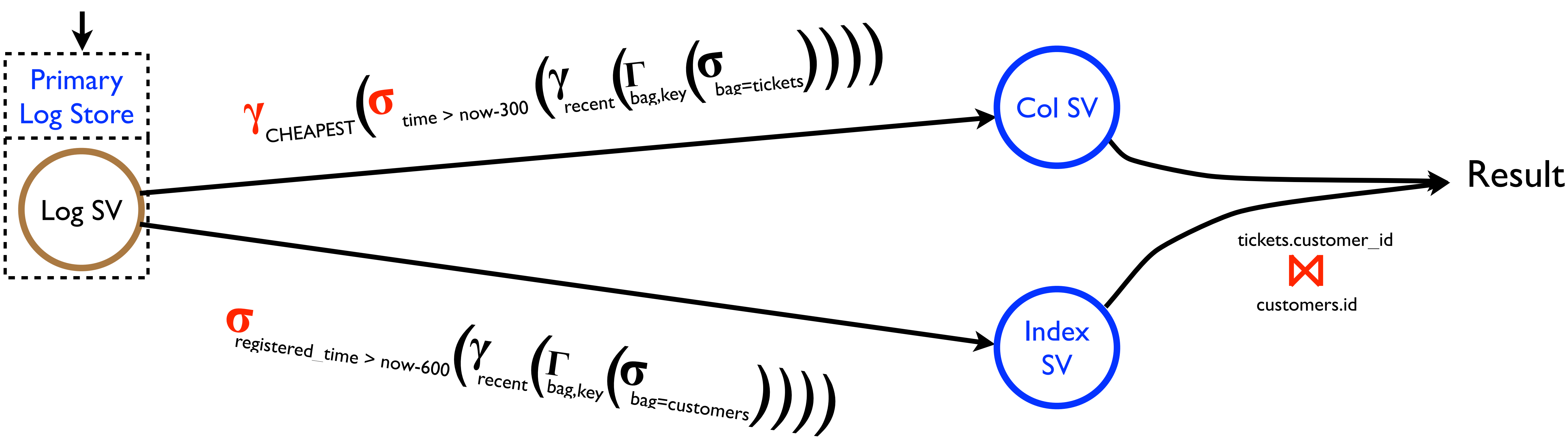


Stream Transformation

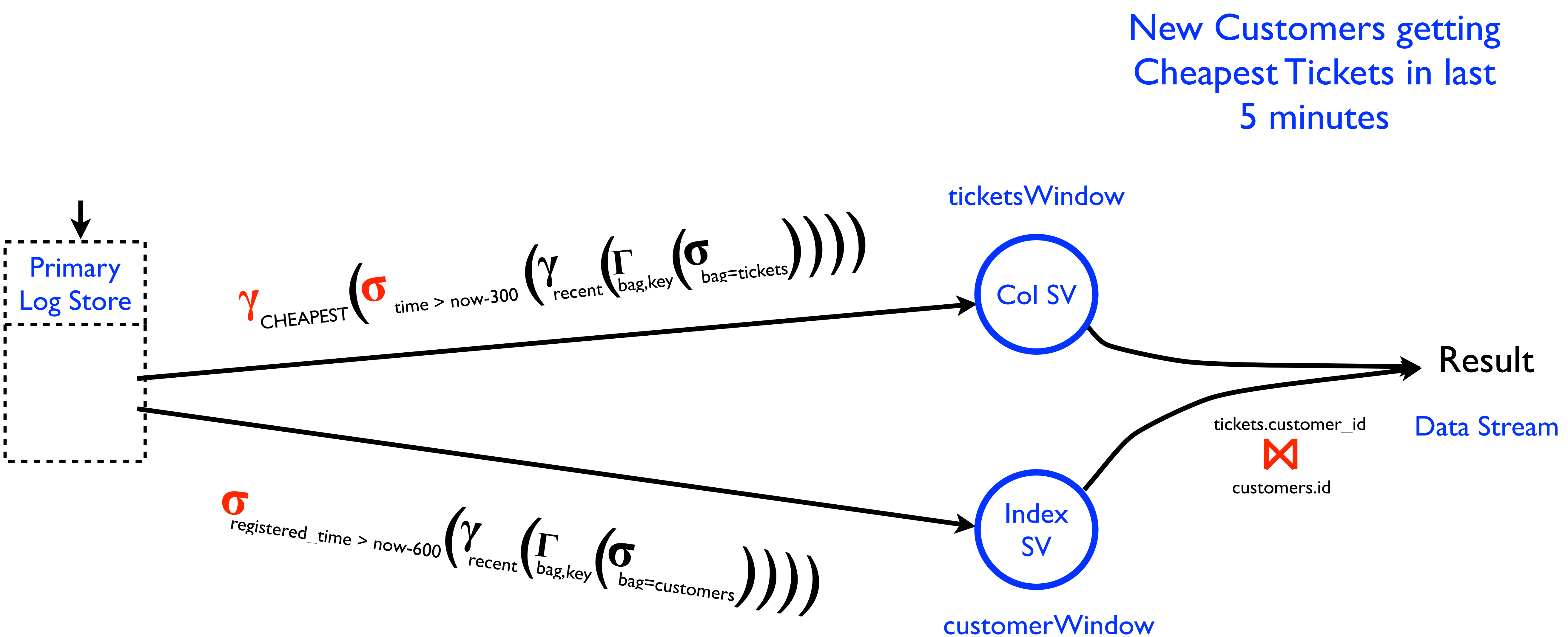


Stream Transformation

New Customers getting Cheapest Tickets in last 5 minutes



Stream Transformation



Related Work

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

- Storage Views different from Materialized Views
  - do not always replicate
  - consider different layouts
- Work on view matching, query containment etc. operate on a higher level
- Still, much of it could be adapted in OctopusDB

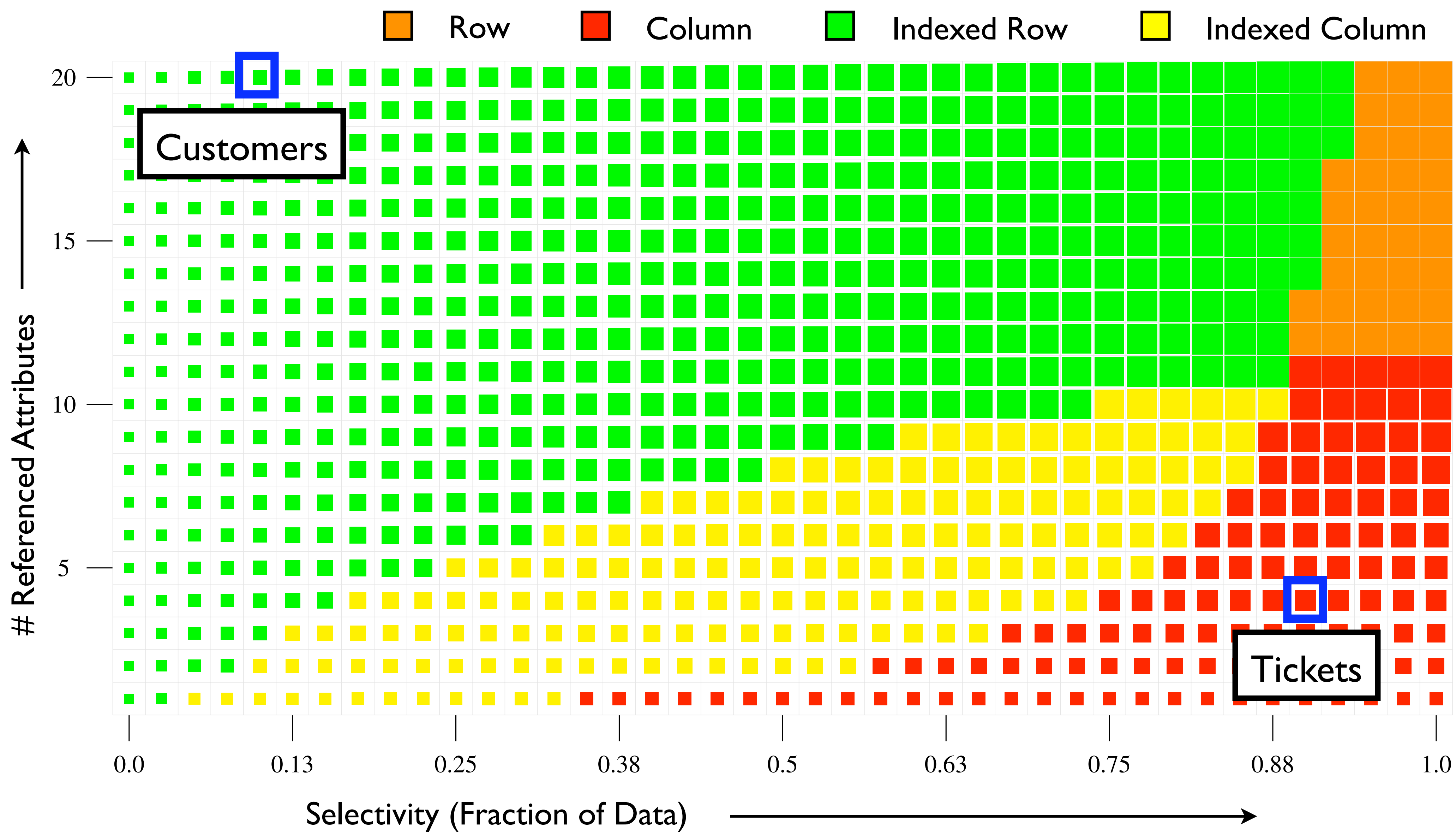
Simulations & Experiment

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

- Goals
  - To show the viability of our approach
  - To show adaptability in OctopusDB
- Method
  - Cost model
  - Prototype Implementation

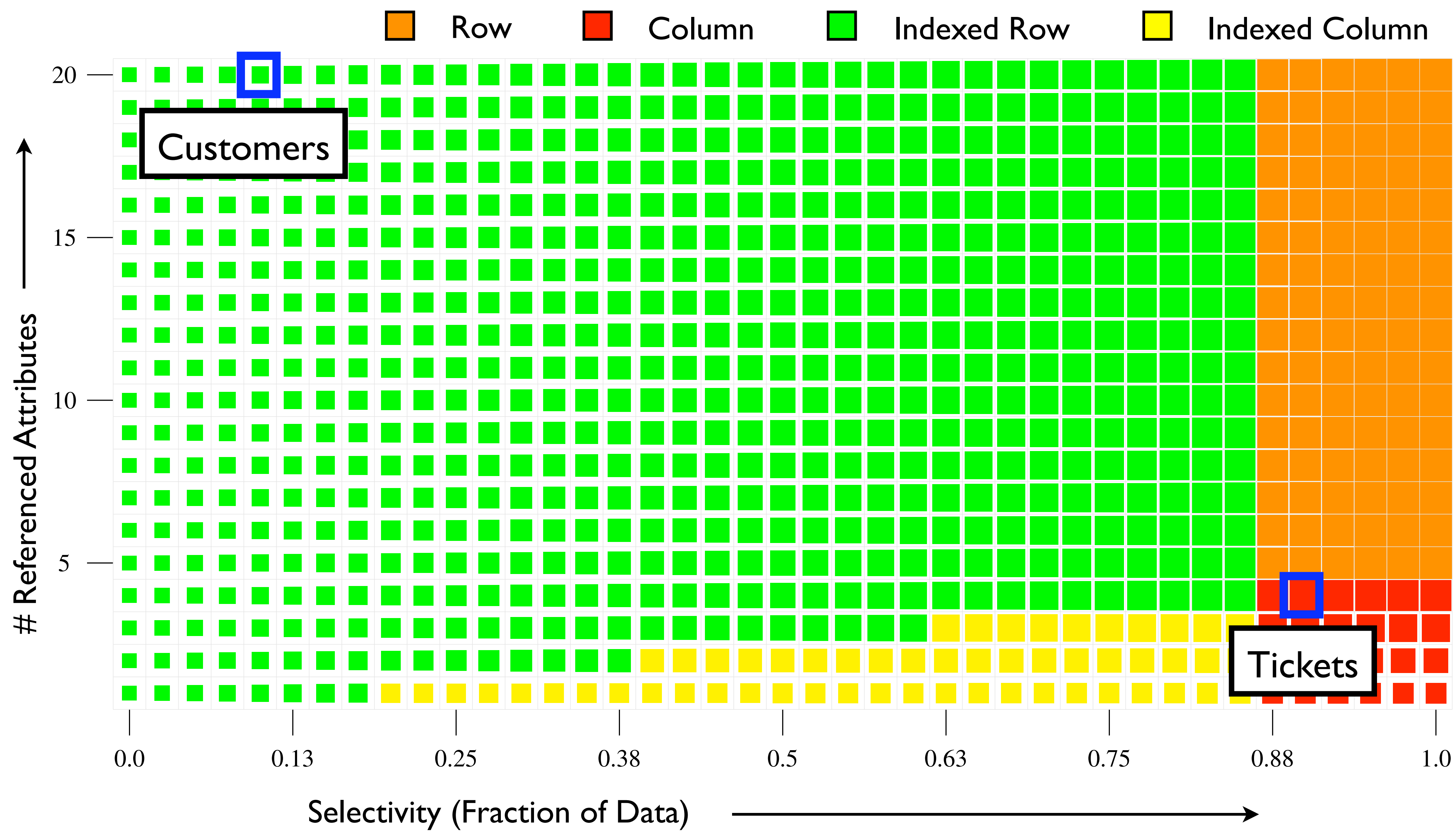
Simulation for Query-based Layout

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group



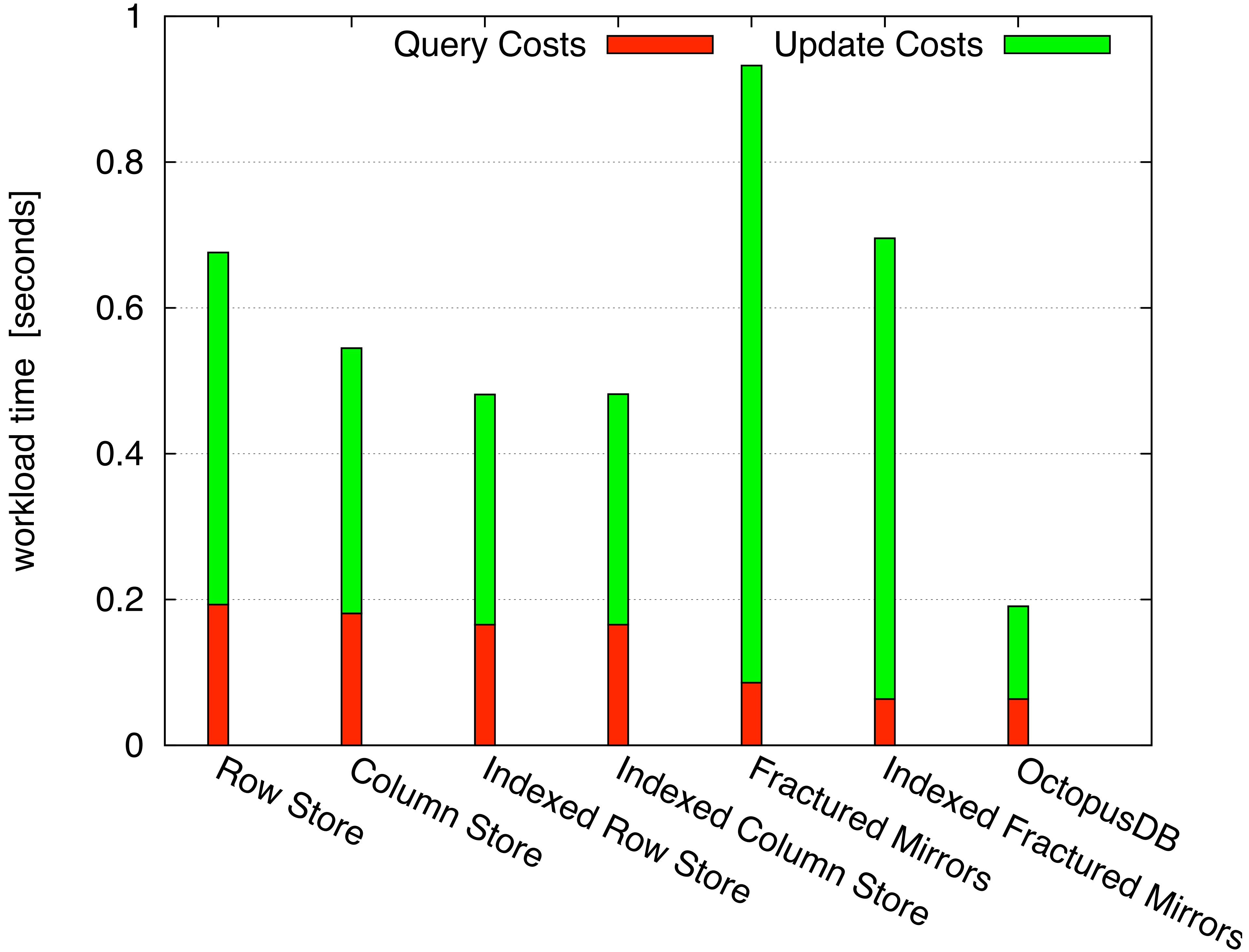
Simulation for Update-based Layout

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

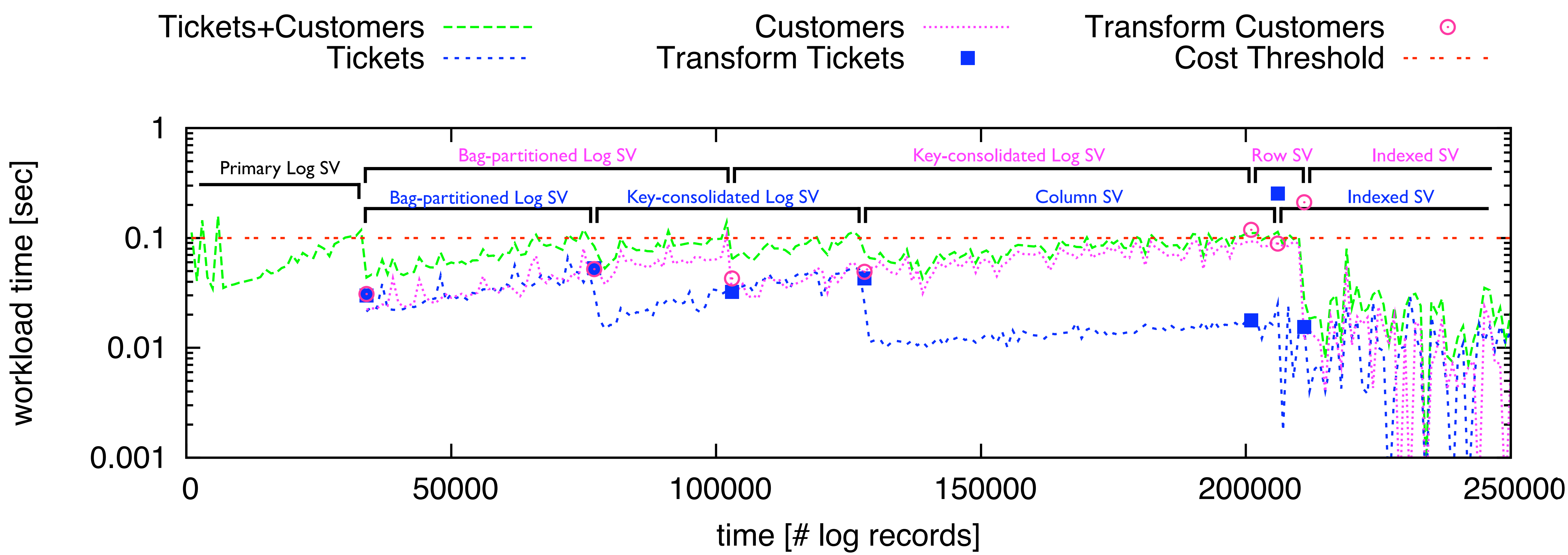


Simulation for Comparing DBMS Stores

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group



Experiment for Automatic Adaption



Main memory prototype implementation

Tickets, Customers have 40 attributes each

10 update queries

30 scan queries with selectivity 0.01, projecting random attributes with skewness 4

Research Challenges

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

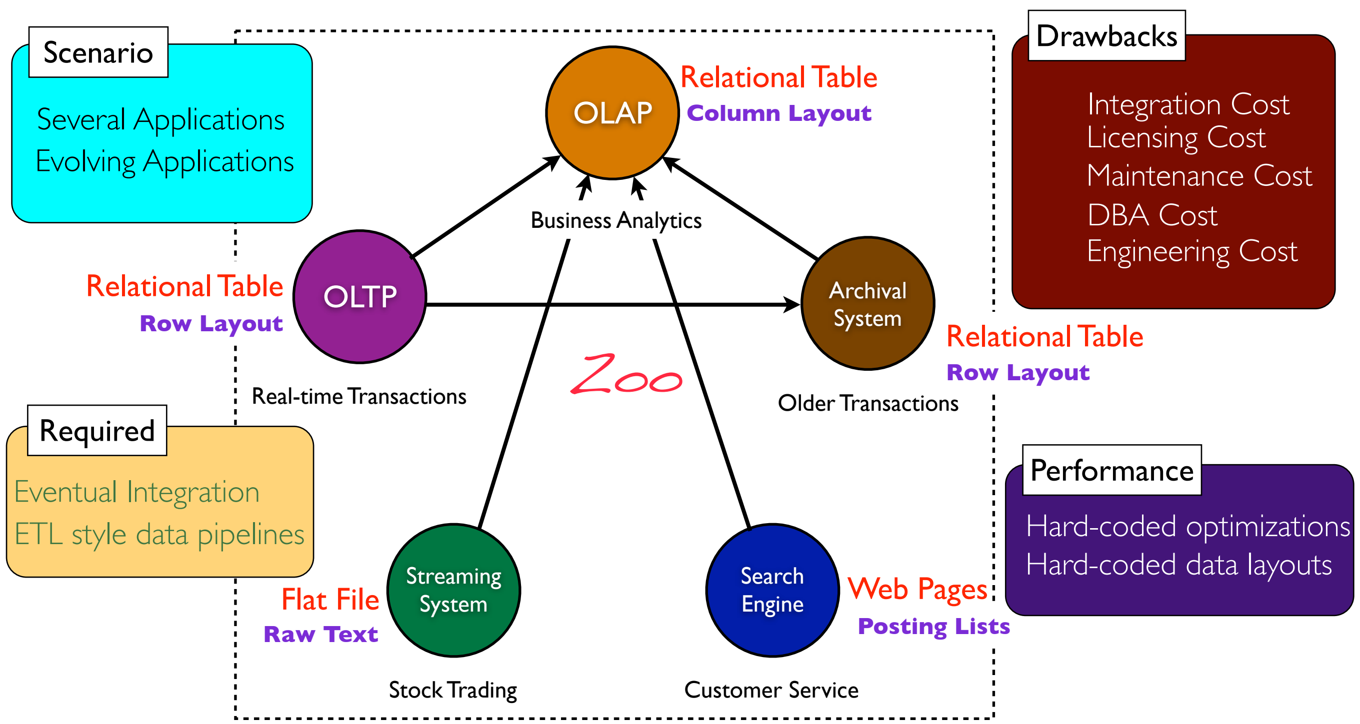
1. Mapping logical schema to physical layout
2. Automatically picking the right layout
   - dynamic partitioning, 2D cracking etc.
3. Storage View selection
4. Storage View update maintenance
5. OctopusDB Benchmarking and Evaluation
6. OctopusDB ideas with MapReduce

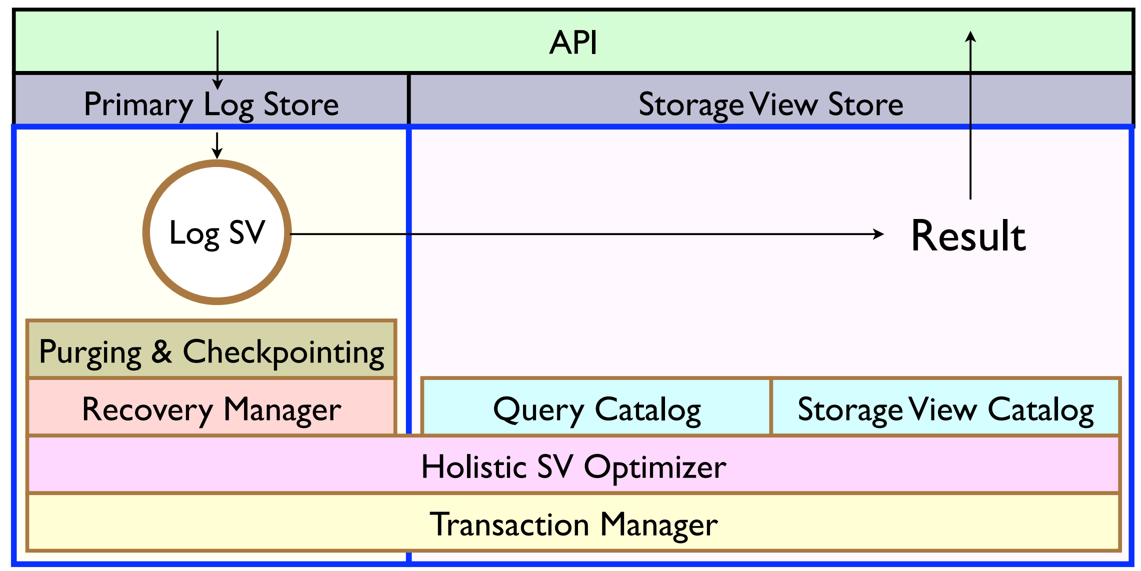
Preliminary Conclusion

- Existing DBMS Engines e.g. OLTP, OLAP are application specific
- We propose a one-size-fits-all database system
- Storage View is a abstract storage concept and gives flexibility to data layout
- Holistic SV Optimizer for the single optimization problem: storage view selection
- Initial results look promising

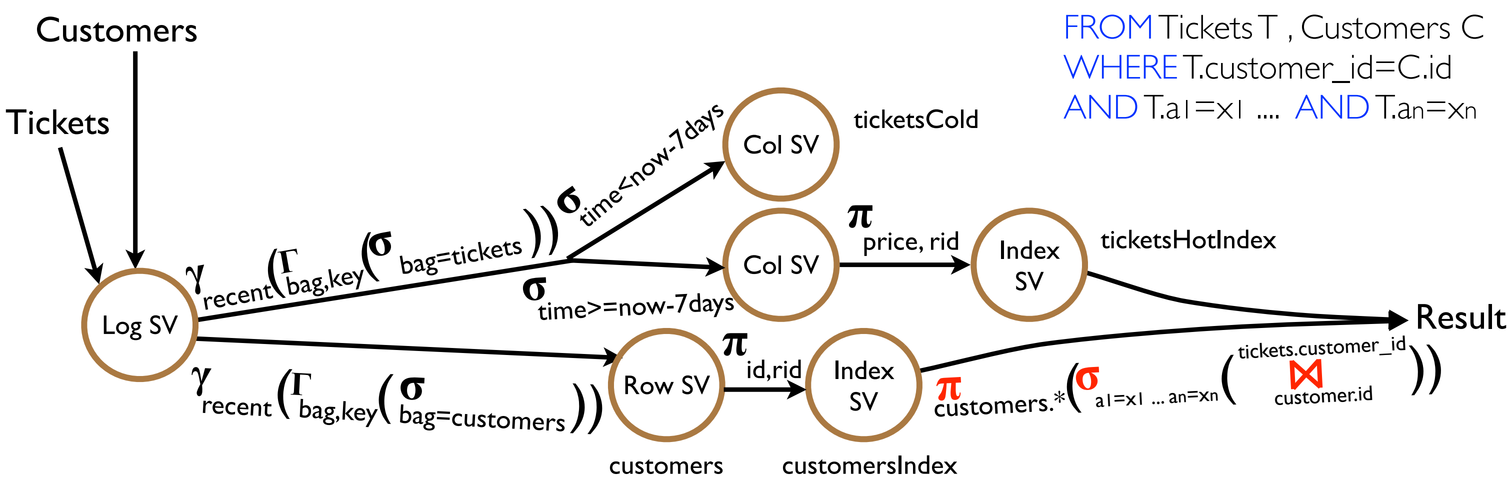
Summary

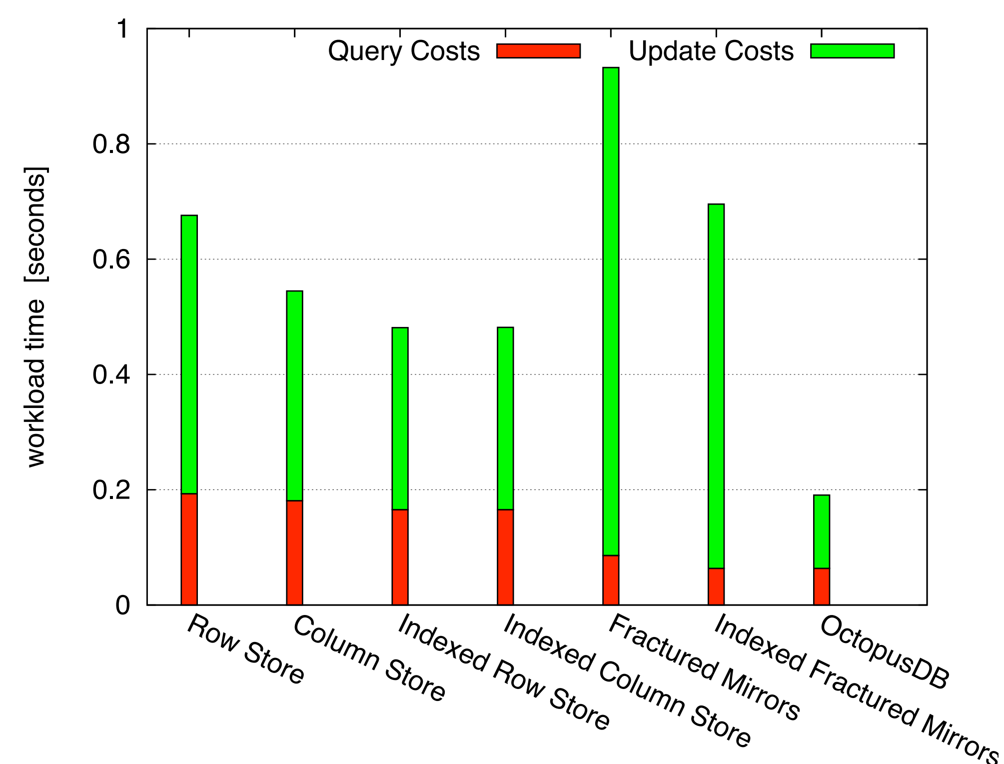
SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group





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





Sources

- http://www.cksinfo.com/signssymbols/pointing/index.html
- http://www.manywallpapers.com/nature-wallpapers/spring/spring-landscape.html