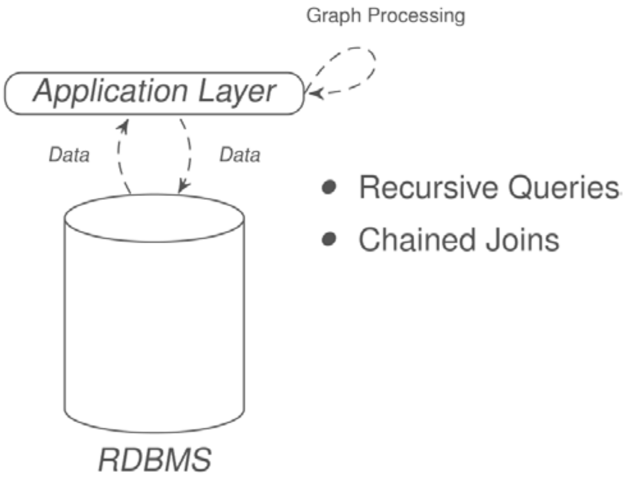
GRAPHITE: An Extensible Graph Traversal Framework for Relational Database Management Systems

Marcus Paradies*, Wolfgang Lehner*, and Christof Bornhoevd° | SSDBM' 15

*TU Dresden °Risk Management Solutions, Inc.

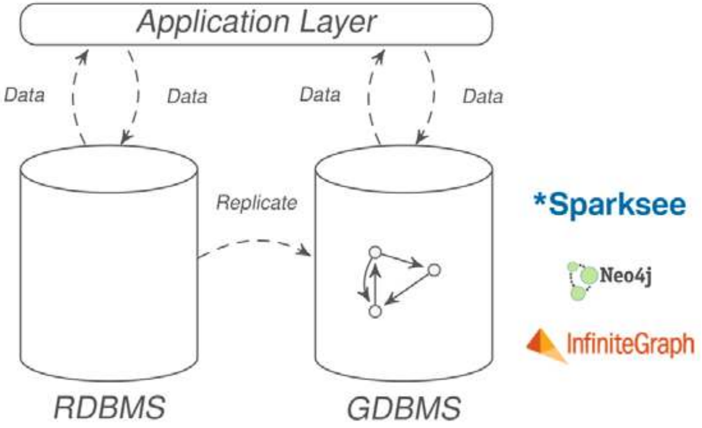
Graph Processing on Enterprise Data

Relational + Application Logic



- ✓ Data already in RDBMS
- ✗ SQL as the only interface/no graph abstraction
- ✗ Data transfer to application

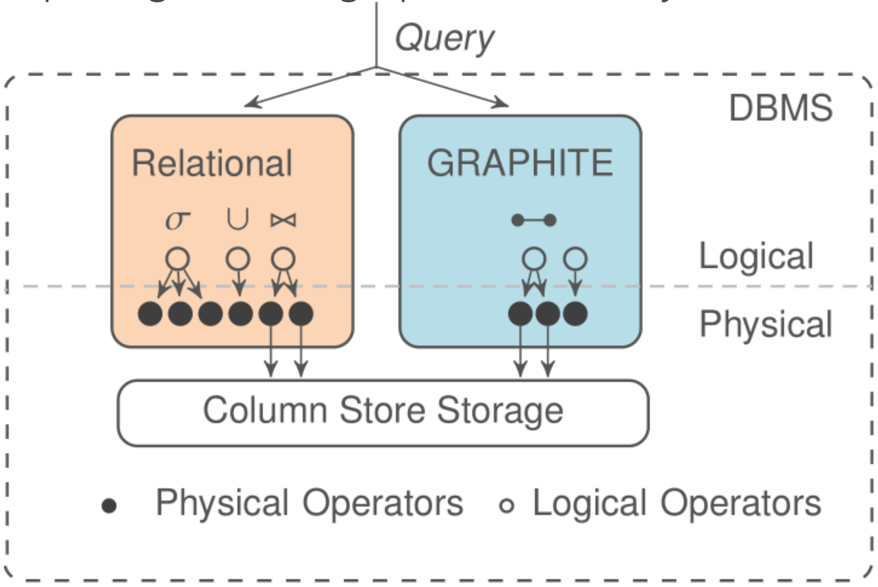
Relational + Graph + Application Logic

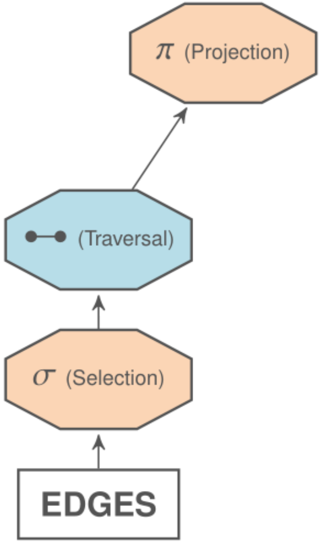


- ✓ Efficient processing in GDBMS
- ✗ Processing on replicated data
- ✗ Data transfer to application
- ✗ No combination with other data models possible

Integration of Graph Processing into an RDBMS

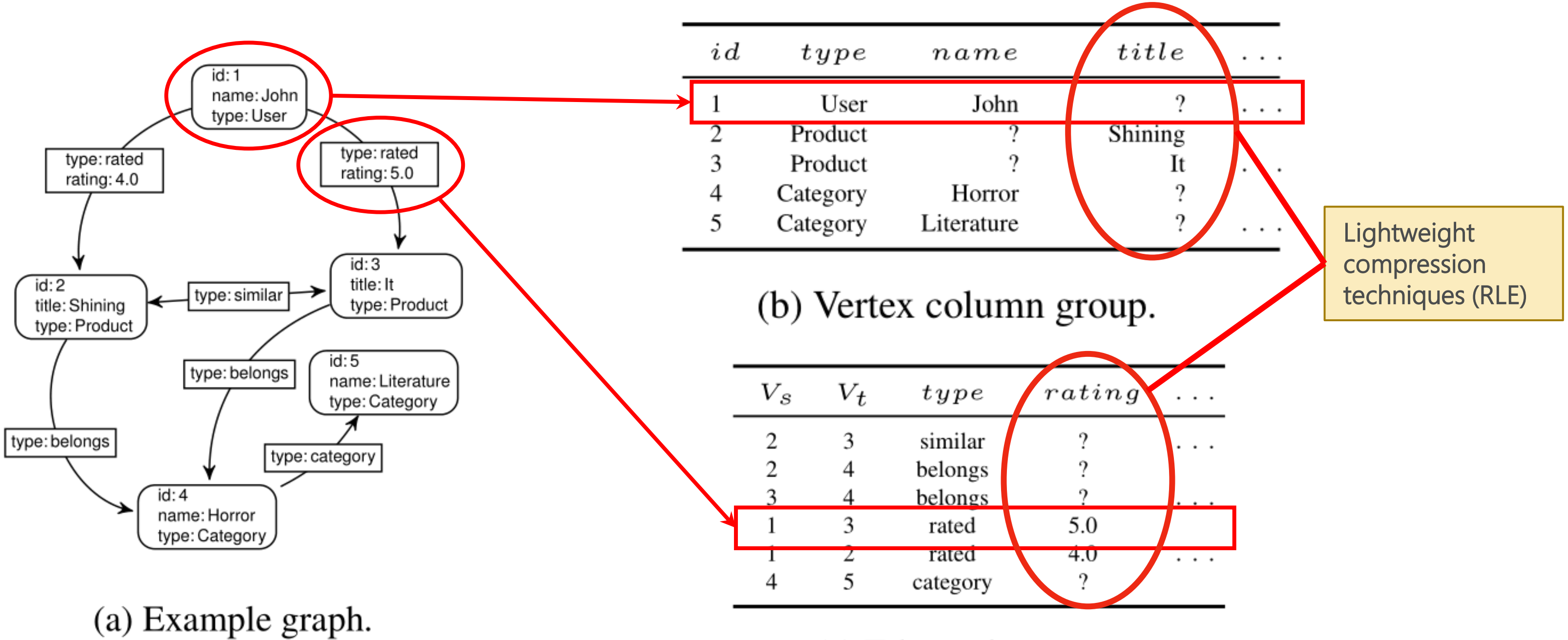
How could a deep integration of graph functionality into an RDBMS look like?





Graph operators can be seamlessly combined with other plan operators

Columnar Graph Storage



(a) Example graph.

id	*type*	*name*	*title*	...
1	User	John	?	...
2	Product	?	Shining	
3	Product	?	It	...
4	Category	Horror	?	
5	Category	Literature	?	...

(b) Vertex column group.

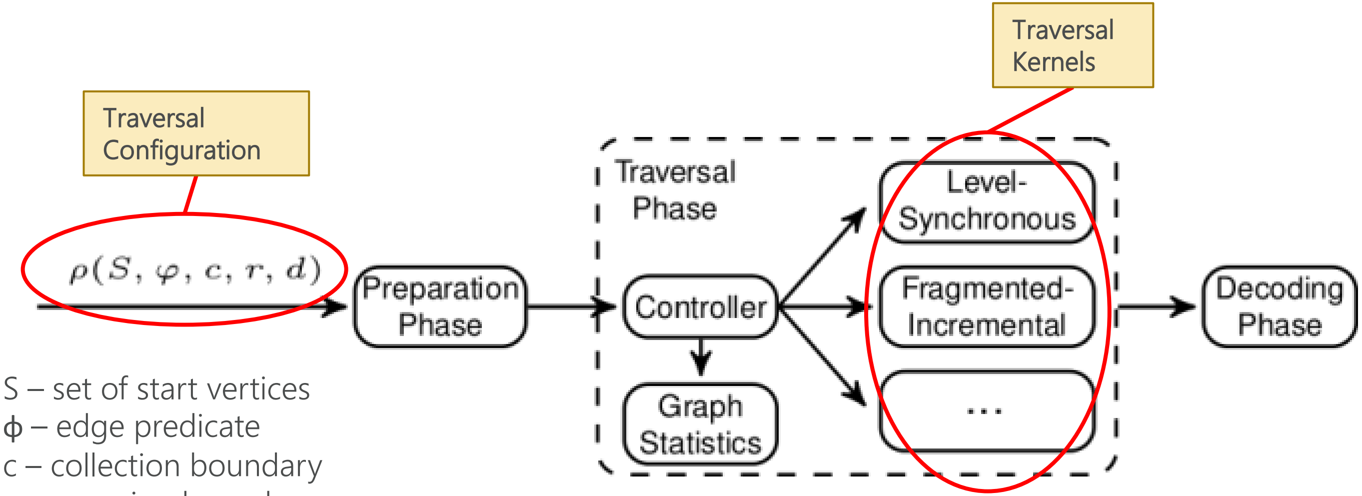
V_s	V_t	*type*	*rating*	...
2	3	similar	?	...
2	4	belongs	?	
3	4	belongs	?	...
1	3	rated	5.0	
1	2	rated	4.0	...
4	5	category	?	

(c) Edge column group.

Lightweight compression techniques (RLE)

All available data types can be used as vertex/edge attributes

Graph Traversal Workflow



Traversal Configuration

Traversal Kernels

- S – set of start vertices
- ϕ – edge predicate
- c – collection boundary
- r – recursion boundary
- d – traversal direction

Pluggable physical traversal kernels as implementations of a logical traversal operator

Graph Traversal Formalism

FORMAL DESCRIPTION (SET-BASED)

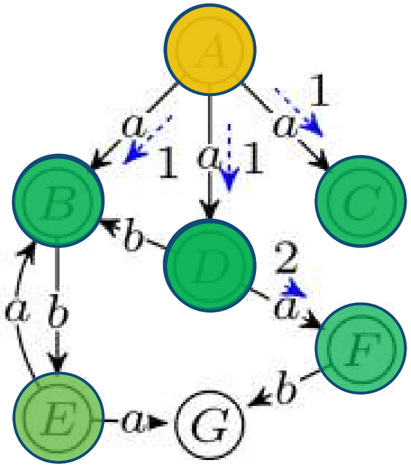
- A traversal operation is a totally ordered set P of path steps
- Each path step p_i receives a vertex set D_{i-1} discovered at level $(i-1)$ and returns a set of adjacent vertices D_i $(1 \leq i \leq r$, r is recursion boundary)
- Initally, $D_0 = \{ S \}$

$$D_i = \{ v \mid \exists u \in D_{i-1\,:}\; e = (u,v) \in E \wedge eval(e, \varphi) \}$$

- The final output R is defined as

$$R = \underbrace{\left(\bigcup\nolimits_{i=c}^{r} D_i\right)}_{\text{Target vertices}} \setminus \underbrace{\left(\bigcup\nolimits_{i=0}^{c-1} D_i\right)}_{\text{Visited vertices}}$$

Graph Traversals by Example



Traversal Configuration	Result
{ {A}, “type = a” OR “type = b”, 2, 2, → }	{ B, C, D, E }

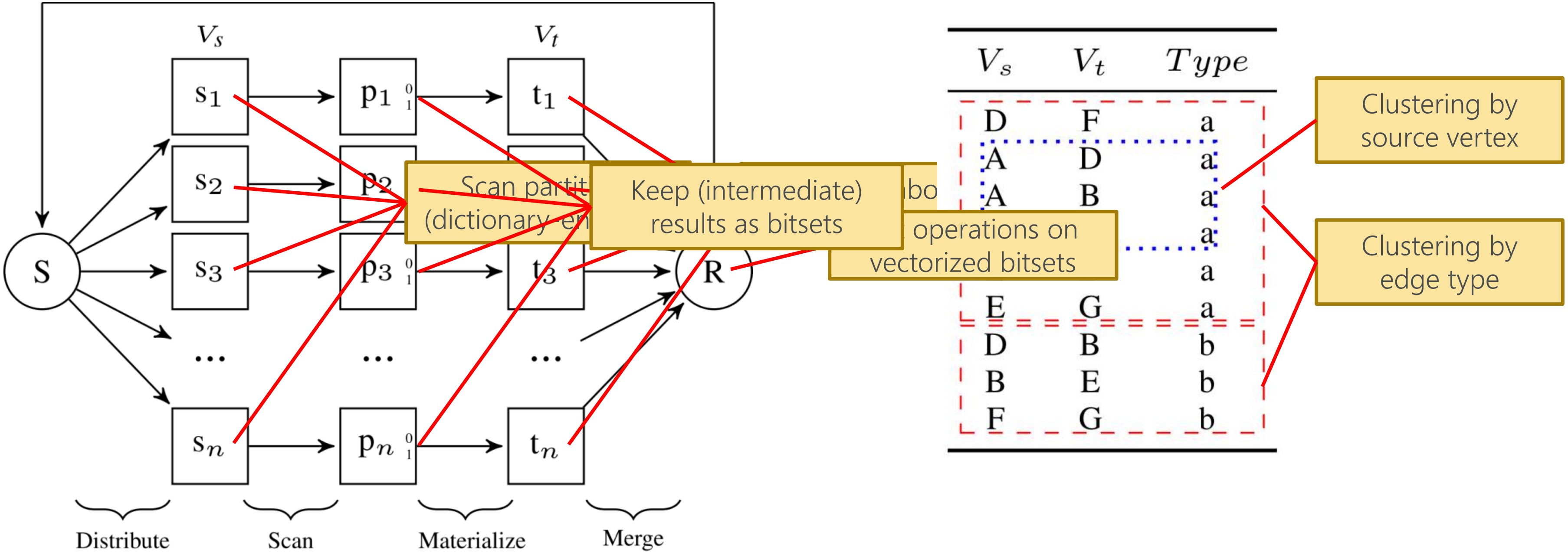
🟡 Root vertex

🟢 Discovered vertex

Level-Synchronous (LS) Traversal

SCAN-BASED GRAPH TRAVERSAL

EDGE CLUSTERING



Scan partit
(dictionary en

Keep (intermediate) results as bitsets

bo

operations on vectorized bitsets

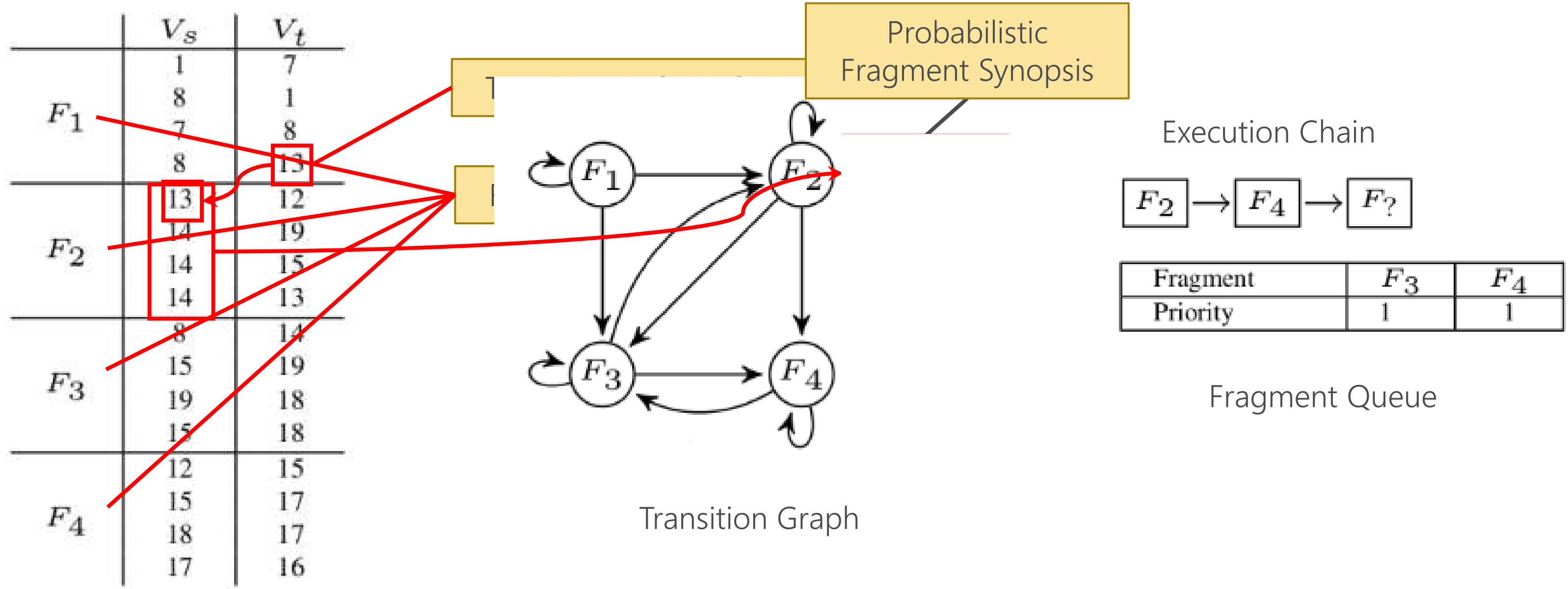
V_s	V_t	$Type$
D	F	a
A	D	a
A	B	a
[illegible]	[illegible]	a
[illegible]	[illegible]	a
E	G	a
D	B	b
B	E	b
F	G	b

Clustering by source vertex

Clustering by edge type

Fragmented-Incremental (FI) Traversal

- Partition column into fragments
- Track dependencies between fragments in index structure
- Goal: Minimize number of fragment reads



	V_s	V_t
F_1	1	7
	8	1
	7	8
	8	13
F_2	13	12
	14	19
	14	15
	14	13
F_3	8	14
	15	19
	19	18
	15	18
F_4	12	15
	15	17
	18	17
	17	16

Probabilistic Fragment Synopsis

Transition Graph

Execution Chain

$F_2 \rightarrow F_4 \rightarrow F_?$

Fragment	F_3	F_4
Priority	1	1

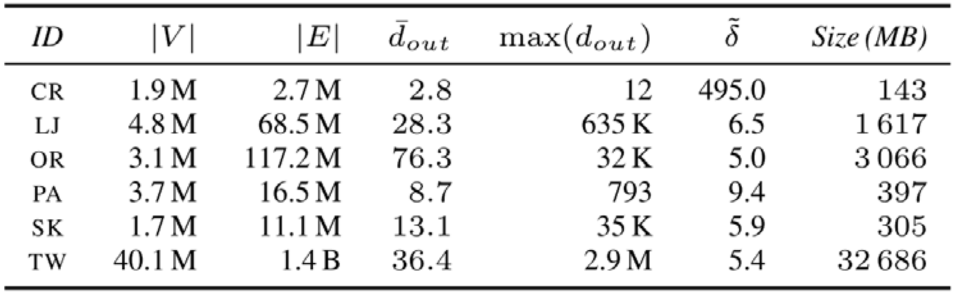
Fragment Queue

For a fragment size equal to |E|, FI-traversal degenerates to LS-traversal

Experimental Evaluation

EVALUATED REAL-WORLD DATA SETS

- Six real-world data sets with different topology characteristics



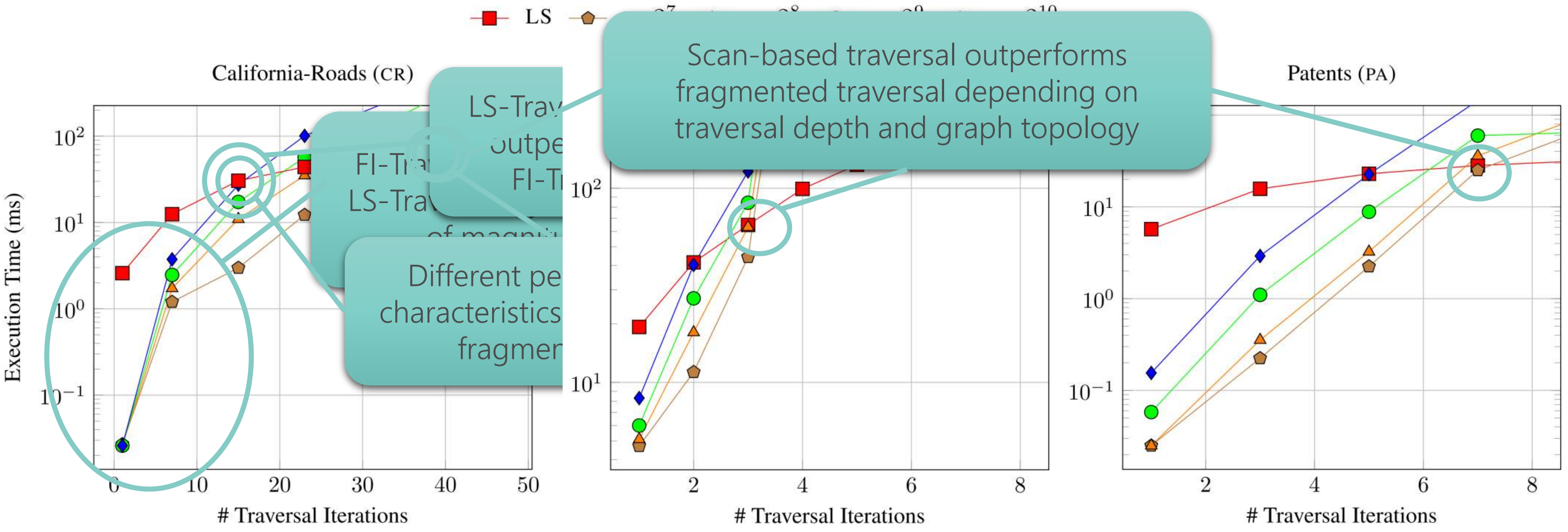
ID	$\lvert V \rvert$	$\lvert E \rvert$	$\bar{d}_{out}$	$\max(d_{out})$	$\tilde{\delta}$	*Size (MB)*
CR	1.9 M	2.7 M	2.8	12	495.0	143
LJ	4.8 M	68.5 M	28.3	635 K	6.5	1 617
OR	3.1 M	117.2 M	76.3	32 K	5.0	3 066
PA	3.7 M	16.5 M	8.7	793	9.4	397
SK	1.7 M	11.1 M	13.1	35 K	5.9	305
TW	40.1 M	1.4 B	36.4	2.9 M	5.4	32 686

EVALUATED SYSTEMS

- Implementation in main-memory column store prototype (C++)
- Graph database (Neo4j)
- RDF DBMS (Virtuoso 7.0 with columnar storage layout)
- Commerical columnar RDBMS (via chained self-joins, with and without index support)

Experimental Evaluation

COMPARISON OF LS-TRAVERSAL AND FI-TRAVERSAL



Scan-based traversal outperforms fragmented traversal depending on traversal depth and graph topology

LS-Trav
outpe
FI-T

FI-Tr
LS-Tra

of magni

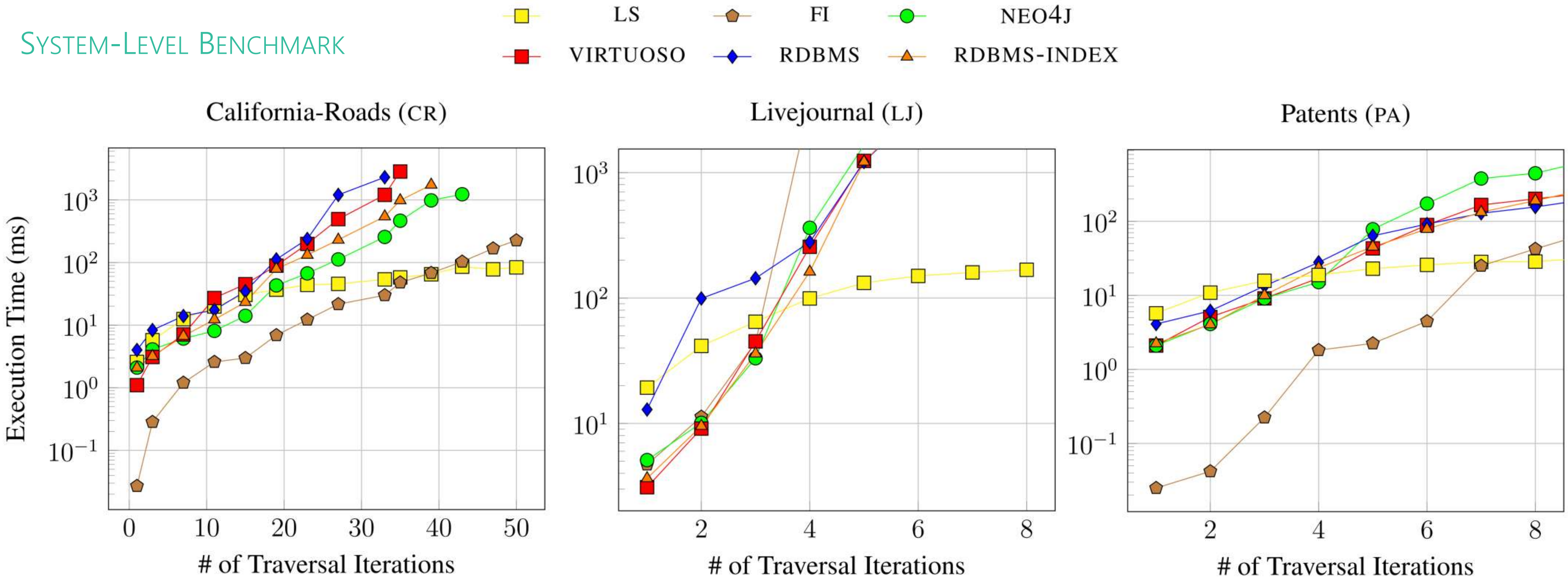
Different pe
characteristics
fragmen

Traversal performance depends on the traversal depth and the topology

Experimental Evaluation

Dresden Database Systems Group

SYSTEM-LEVEL BENCHMARK



Combination of LS and FI-traversal outperforms native graph systems by up to two orders of magnitude

TECHNISCHE UNIVERSITÄT DRESDEN

Summary

GRAPHITE

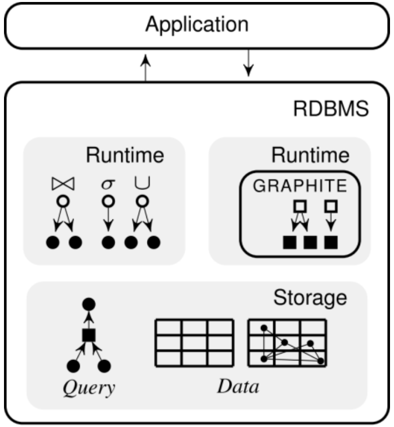
- Graph processing tightly integrated into RDBMS
- Extensions of core components by graph extensions (operators, cost model, index structures)
- Topology characteristics-aware traversal operators

GRAPH-SPECIFIC DATA STATISTICS AND ALGORITHMS

- Diverse graph topologies demand different algorithmic design decisions
- Index scan versus full column scan decision also applies for graph traversals

FUTURE WORK

- Integration with temporal, spatial, and text data
- Language extensions for custom code executed during graph traversal



Integration of GRAPHITE into RDBMS

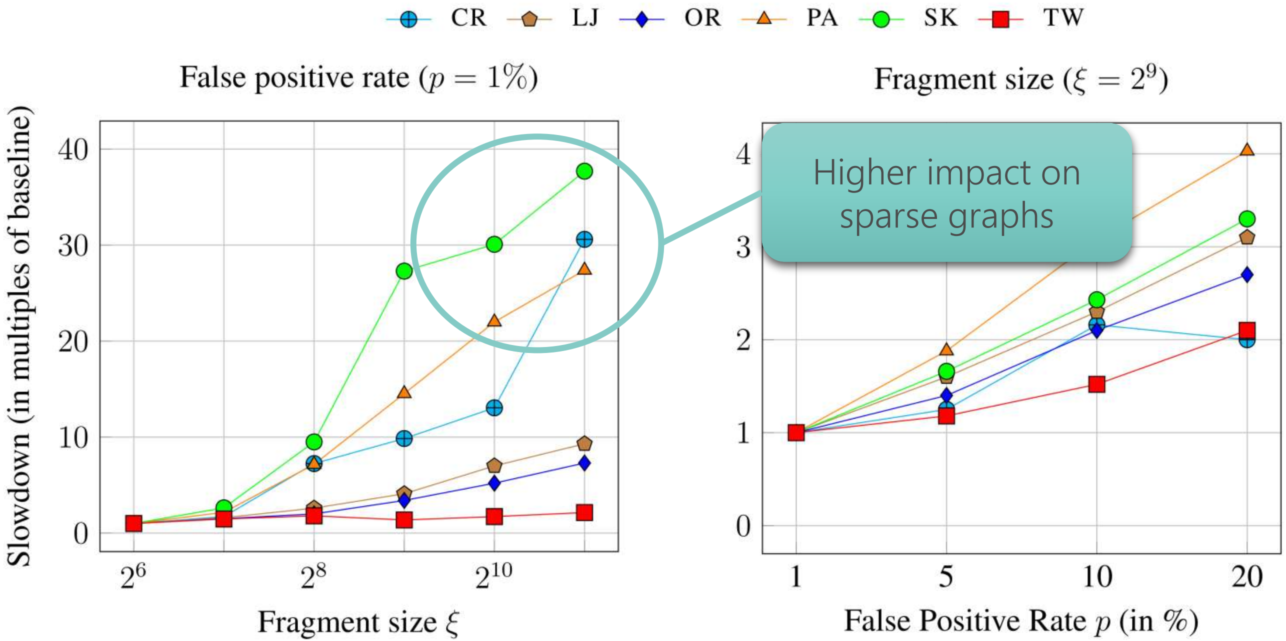
Contact

Marcus Paradies, TU Dresden
m.paradies@sap.com
https://wwwdb.inf.tu-dresden.de/team/external-members/marcus-paradies/

Backup Slides

Experimental Evaluation

EFFECT OF FRAGMENT SIZE AND FALSE POSITIVE RATE

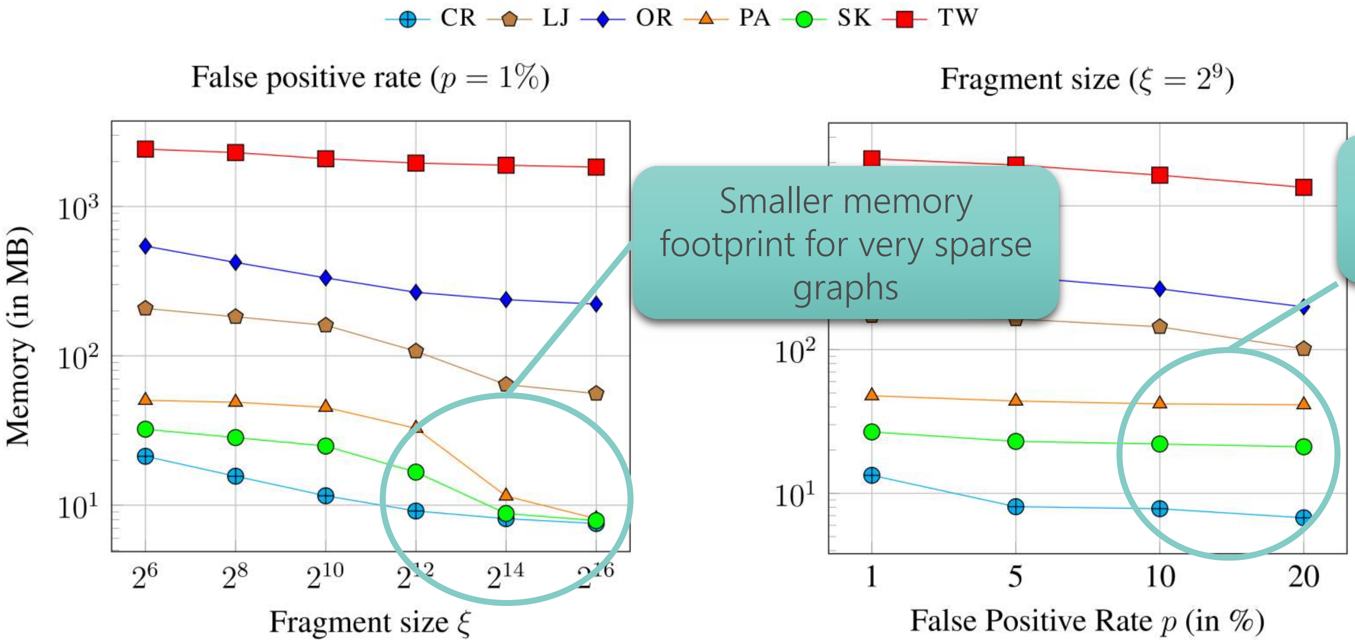


Higher impact on sparse graphs

Elapsed time of FI-traversal more sensible to a change of the fragment size than a change to the false positive rate

Experimental Evaluation

MEMORY CONSUMPTION



Smaller memory footprint for very sparse graphs

No significant improvement for very sparse graphs

Memory footprint of the index can be reduced by increasing the fragment size or the false positive rate

Cost Model

- Models based on the total number of accessed edges including a constant access cost
- Goal is to minimize the number of edges to read

LEVEL-SYNCHRONOUS

$$\mathcal{C}_{\mathrm{LS}} = \min\{r, \tilde{\delta}\} \cdot |E| \cdot \mathcal{C}_e$$

FRAGMENTED-INCREMENTAL

$$\mathcal{C}_{\mathrm{FI}} = \sum_{i=0}^{\min\{r,\tilde{\delta}\}} (1+p)(\bar{d}_{\mathrm{out}})^i \cdot \xi \cdot \mathcal{C}_e$$