TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Adaptive Query Processing on Vectorized Hardware

2nd TAB-Talk, 30.10.2020, Johannes Pietrzyk

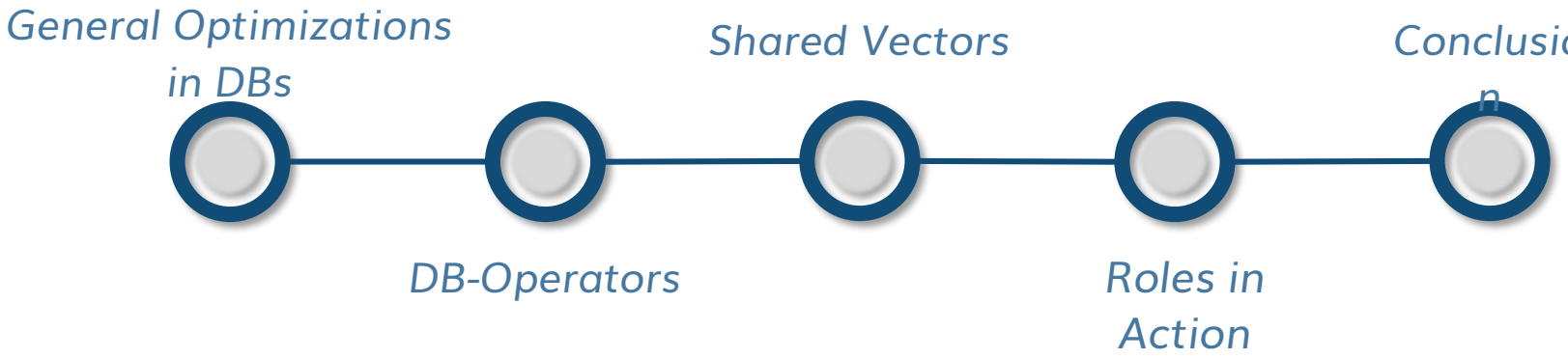
ROSI

Outline

Dresden Database Systems Group

Research Hypothesis

Roles enrich the process of designing and developing adaptive software systems.

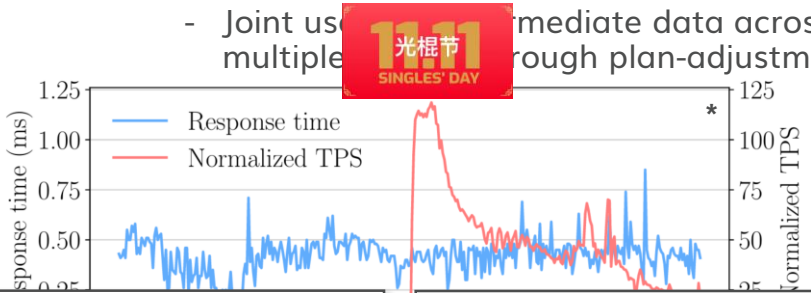


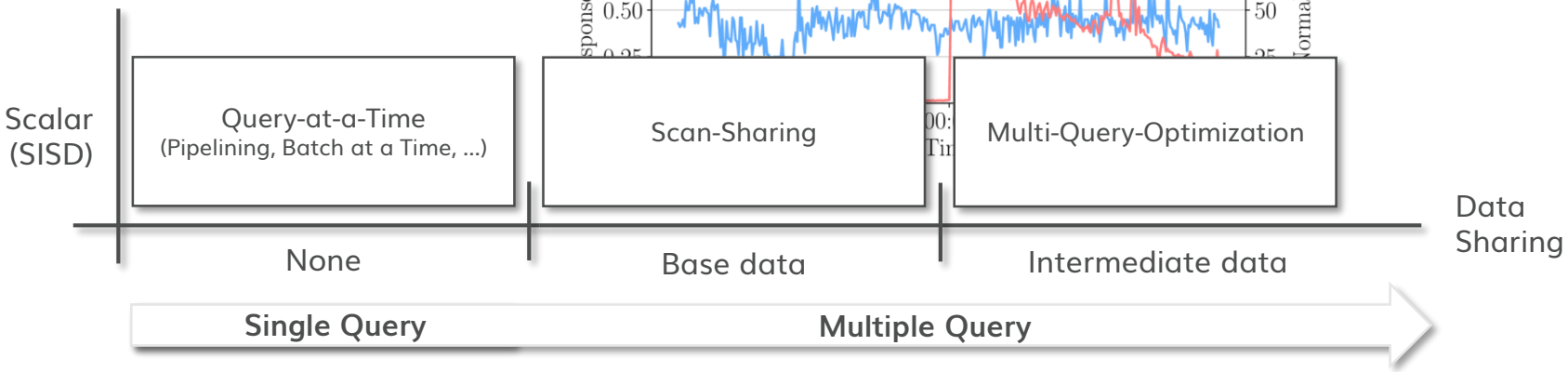
General Overview (State-of-the-Art)

Dresden Database Systems Group

Optimization Techniques

- Share Data:
  - Share base data through specialized operators
  - Joint us[illegible]mediate data across multiple[illegible]rough plan-adjustments





* Huang, Gui, et al. "X-Engine: An optimized storage engine for large-scale E-commerce transaction processing." Proceedings of the 2019 International Conference on Management of Data. 2019.
https://assets.t3n.sc/news/wp-content/uploads/2018/10/singles-day-alibaba-online-handel.jpg?auto=format&fit=crop&h=348&ixlib=php-2.3.0&w=620

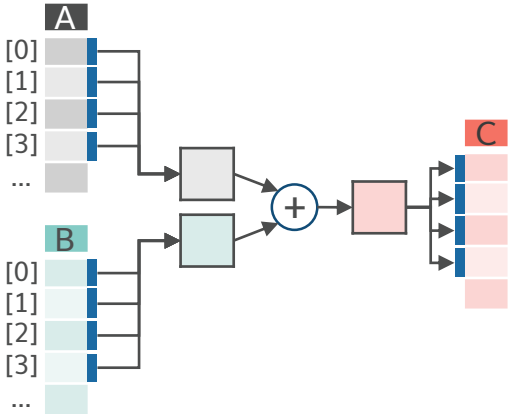
From Scalar to Vectorized Processing

Dresden Database Systems Group

Example:

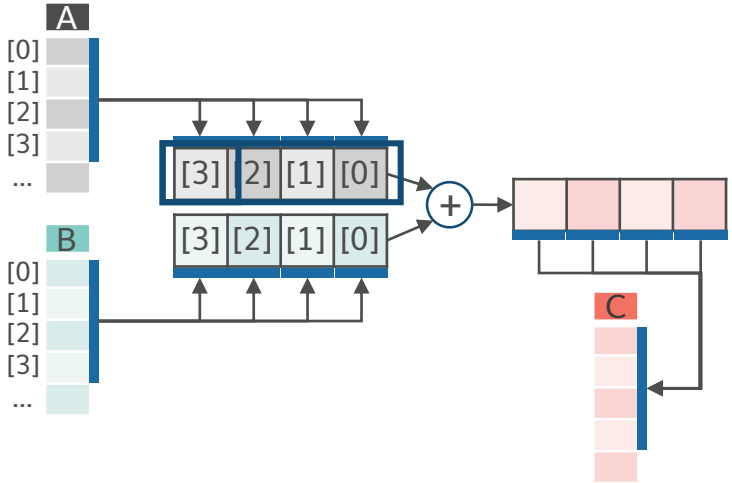
Single Instruction Single Data (SISD)

"Scalar"



Single Instruction Multiple Data (SIMD)

"Vectorized"

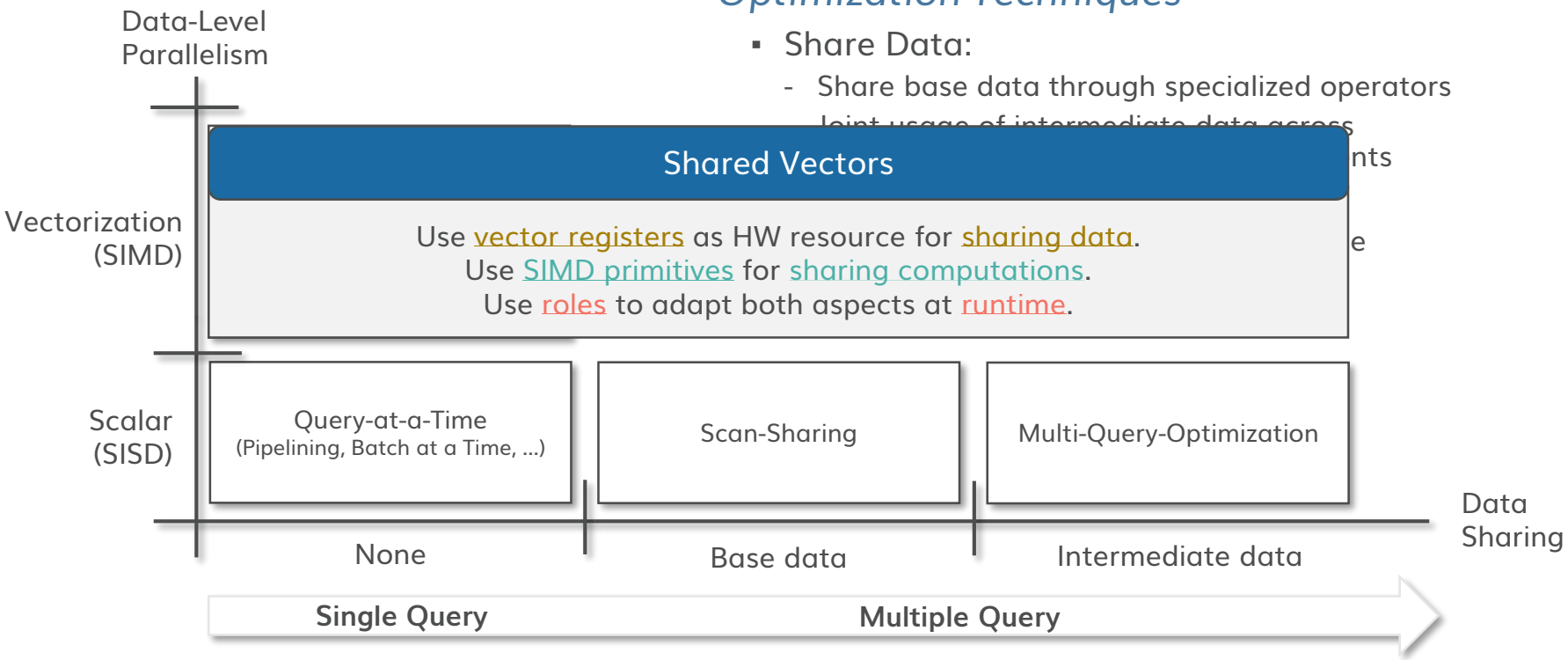


General Overview (State-of-the-Art)

Dresden Database Systems Group

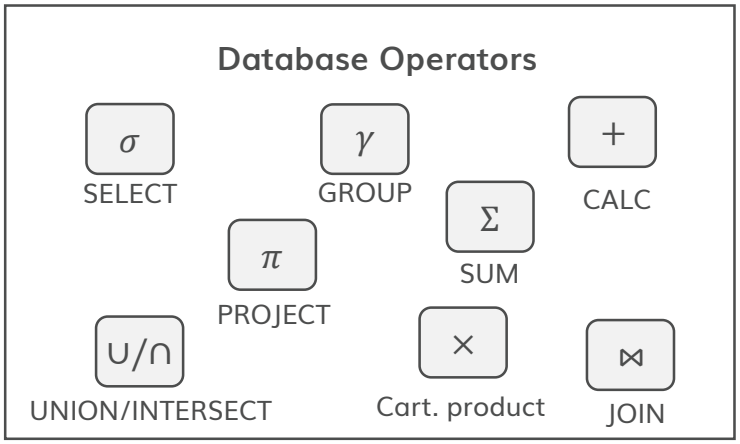
Optimization Techniques

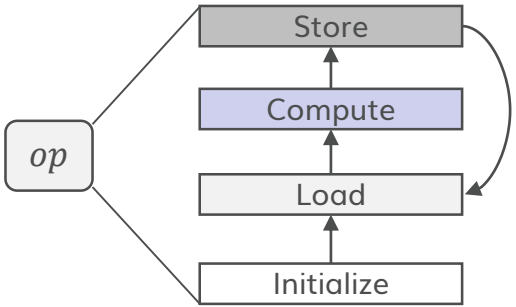
- Share Data:
  - Share base data through specialized operators
  - Joint usage of intermediate data across
  - nts
  - e



Operators in DBMS

Dresden Database Systems Group





Research Prototype – MorphStore

- Vectorization and Compression as first class citizen

Operator	Load-Phase	Compute-Phase	Store-Phase
σ	load	compare	store
π	load, gather	/	store
$+$	load	add, mul, div	store
Σ	load	$+$	store
$\gamma/\bowtie$	load, gather	$+\,\sigma$	store, scatter

Manageable number of operators

Operators have primitives in common

An Example

Dresden Database Systems Group

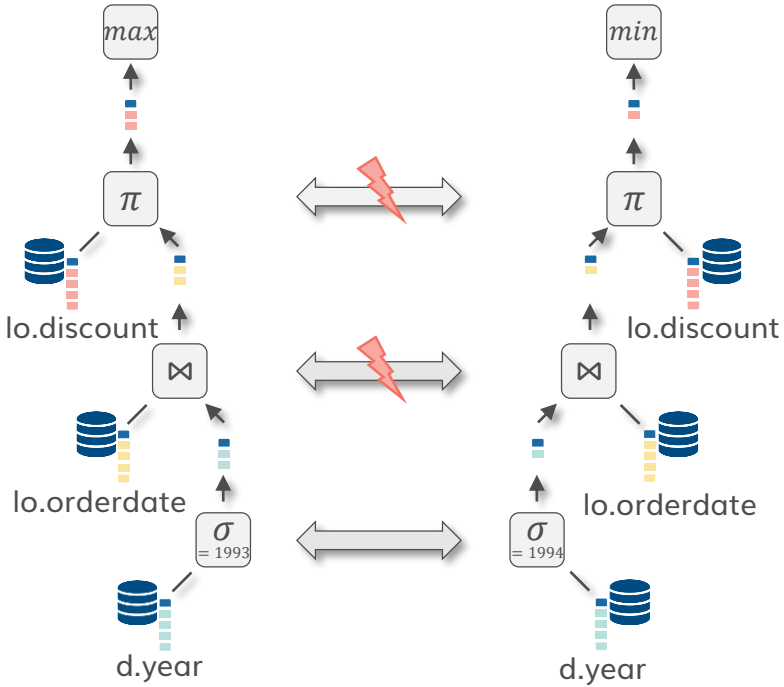
Inspired by SSB Query 1.1

```
SELECT MAX(lo.discount)
FROM   lineorder as lo, date as d
WHERE  lo.orderdate = d.datekey
AND    d.year = 1993
```

```
SELECT MIN(lo.discount)
FROM   lineorder as lo, date as d
WHERE  lo.orderdate = d.datekey
AND    d.year = 1994
```

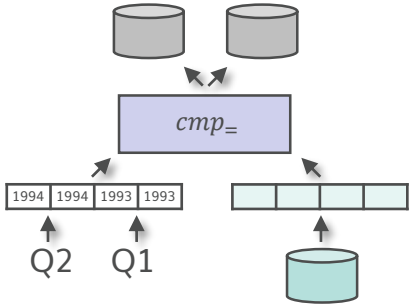
- Varying predicates for filters
- Different aggregation functions
- Base data can be shared
- No sharing of intermediates possible
- BUT: Sharing on primitive level possible

Execution Plans



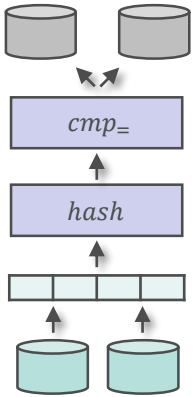
Sharing on primitive level – using vectorization

Dresden Database Systems Group



Filter-Operator

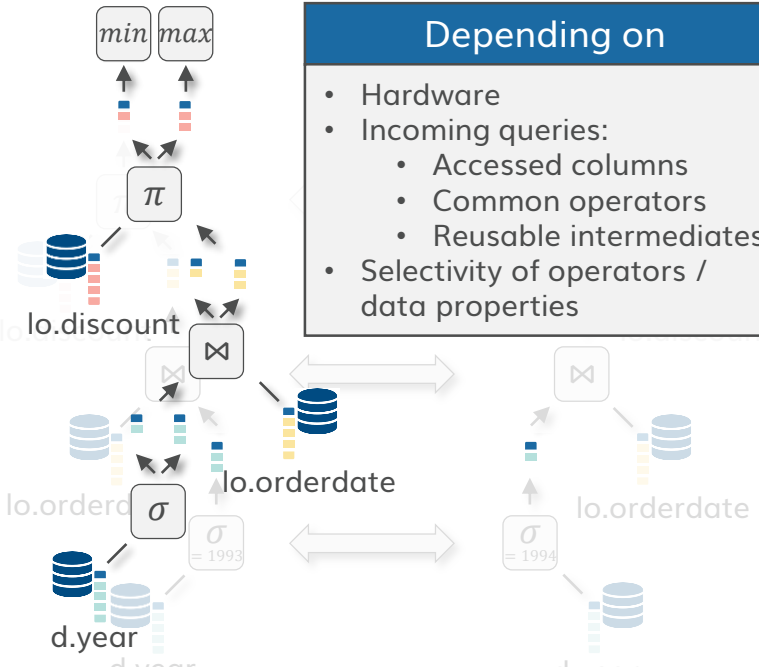
- **Primitive**-sharing (compare) and **Data**-sharing possible
- Query-dependent (predicate) initialization phase
- Query-dependent (operators) load and store phase
  - Load-Phase: 1 Column
  - Store-Phase: 2 Columns



Join-Operator

- **Primitive**-sharing but no Data-sharing possible
- Query-dependent load and store phase
  - Load-Phase: 2 Columns
  - Store-Phase: 2 Columns
- Even the compare primitive can be shared across operators

Sharing potential



Depending on

- Hardware
- Incoming queries:
  - Accessed columns
  - Common operators
  - Reusable intermediates
- Selectivity of operators / data properties

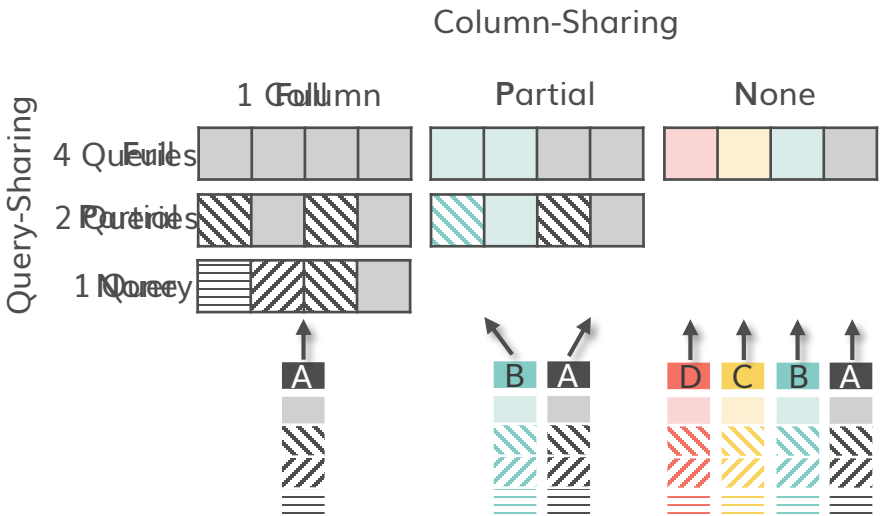
Sharing has to be adapted at workload and query runtime

First Evaluation of Shared Vectors*

Dresden Database Systems Group

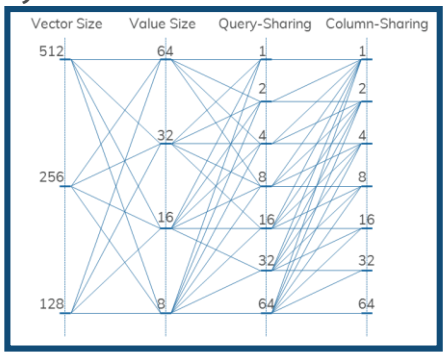
Sharing potential depending on Query and Column count

- Number of avail. vector lanes depends on vector size and value size



Setup

- Simple query consisting of two shared operators
- Systematic evaluation of impact of vector lane, query and column count



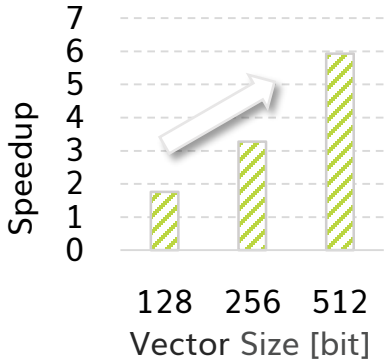
Platform

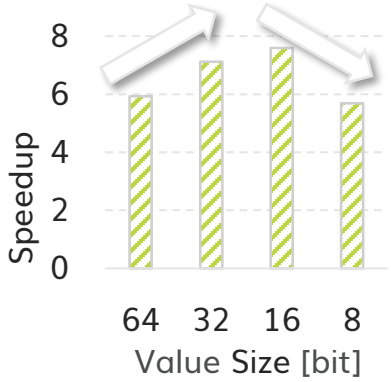
- Intel Xeon Gold 6126
- 92 GB DDR4
- 3.7 Ghz max. core frequency
- 64-bit CentOs (7.7.1908)

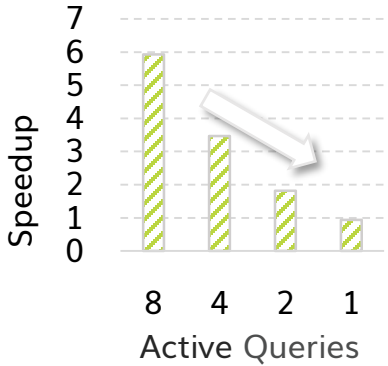
*published at DaMoN'20

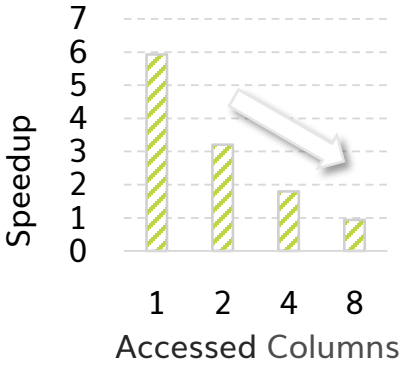
First Evaluation of Shared Vectors

Dresden Database Systems Group









Key Findings:

- If work can be shared, Shared Vectors are beneficial
- Bigger vector registers lead to better speedup
- Value Sizes have significant impact on speedup
- The more queries, the higher the speedup
- The more columns, the lower the speedup

Multiple tuning knobs available

⇩

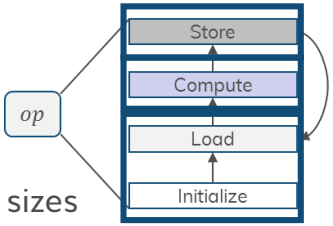
High demand on a flexible, adaptive solution

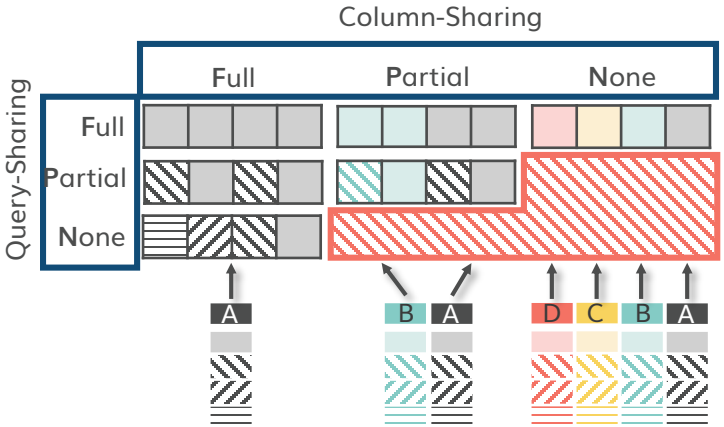
Mapping System Design to CROM Model

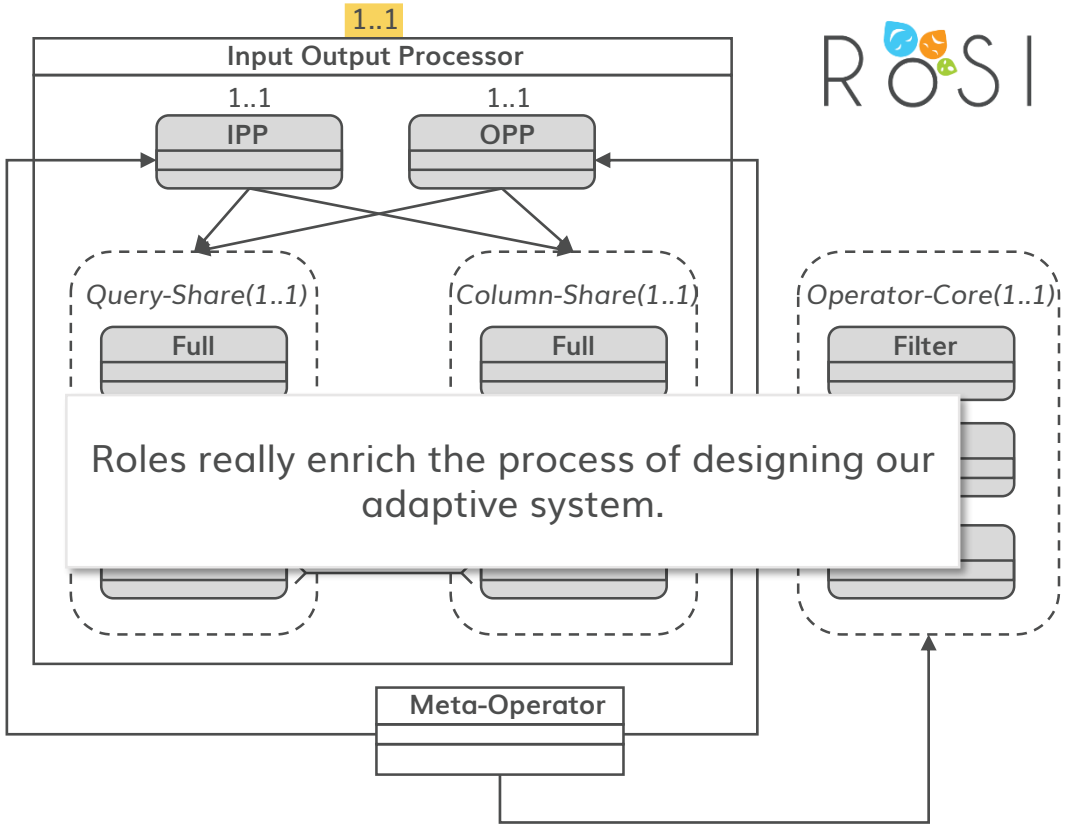
Dresden Database Systems Group

Adaption to

- Hardware
  - Available vector sizes
- Processed data
  - Data size
- Workload-Structure







ROSI

Roles really enrich the process of designing our adaptive system.

Summary

Dresden Database Systems Group

Conclusion

- Presented a novel approach to share data and computations using vectorized hardware (using TVL*)
- First evaluations published at DaMoN'20
  - Systematically evaluated impact of contexts
  - Focused on input preprocessing
  - First results seem very promising
- Proposed concept rely on frequent adaption
- Seems to perfectly fit for Roles as a technique for runtime adaptivity

Future Work

- Open question:
  - How to realize and limit possible role transitions for IPP and OPP at runtime
- Implementation of:
  - Generalized operator model
  - Roles and compartments to enable changes of input and output processing during operator runtime
  - Consistent role transitions
- Evaluation of:
  - Impact of more complex queries
  - Role transition costs at runtime
- Design of a cost-based optimizer to determine which roles to apply
  - Using an Event-Condition-Action Loop

* Ungethüm, A., Pietrzyk, J., Damme, P., Krause, A., Habich, D., Lehner, W., & Focht, E. (2020). Hardware-Oblivious SIMD Parallelism for In-Memory Column-Stores. In *CIDR*.

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Adaptive Query Processing on Vectorized Hardware

2nd TAB-Talk, 30.10.2020, Johannes Pietrzyk

ROSI

List of relevant publications

Dresden Database Systems Group

[1] Pietrzyk, J., Ungethüm, A., Habich, D., & Lehner, W. (2018). Beyond Straightforward Vectorization of Lightweight Data Compression Algorithms for Larger Vector Sizes. In Grundlagen von Datenbanken (pp. 71-76).

[2] Ungethum, A., Pietrzyk, J., Damme, P., Habich, D., & Lehner, W. (2018, April). Conflict detection-based run-length encoding-AVX-512 CD instruction set in action. In 2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW) (pp. 96-101). IEEE.

[3] Pietrzyk, J., Ungethüm, A., Habich, D., & Lehner, W. (2019). Fighting the duplicates in hashing: conflict detection-aware vectorization of linear probing. BTW 2019.

[4] Pietrzyk, J., Habich, D., Damme, P., Focht, E., & Lehner, W. (2019). Evaluating the vector supercomputer sx-aurora TSUBASA as a co-processor for in-memory database systems. Datenbank-Spektrum, 19(3), 183-197.

[5] Ungethüm, A., Pietrzyk, J., Damme, P., Krause, A., Habich, D., Lehner, W., & Focht, E. (2020). Hardware-Oblivious SIMD Parallelism for In-Memory Column-Stores. In CIDR.

[6] Habich, D., Damme, P., Ungethüm, A., Pietrzyk, J., Krause, A., Hildebrandt, J., & Lehner, W. (2019, June). Morphstore-in-memory query processing based on morphing compressed intermediates LIVE. In Proceedings of the 2019 International Conference on Management of Data (pp. 1917-1920).

[7] Pietrzyk, J., Habich, D., & Lehner, W. (2020, June). To share or not to share vector registers?. In Proceedings of the 16th International Workshop on Data Management on New Hardware (pp. 1-10).