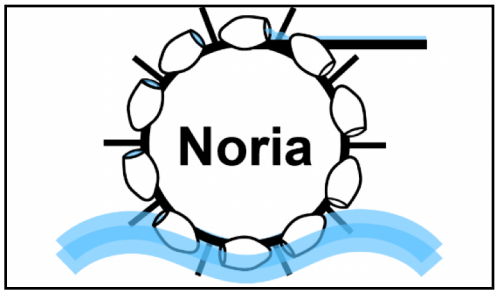
Noria: Dynamic Partially-stafeful data flow for high-performance web applications

Gjengjet et al. (OSDI'18)



Presentation by **Yannis Marketakis**

In the context of HY559 (Computer Science Department, UOC)

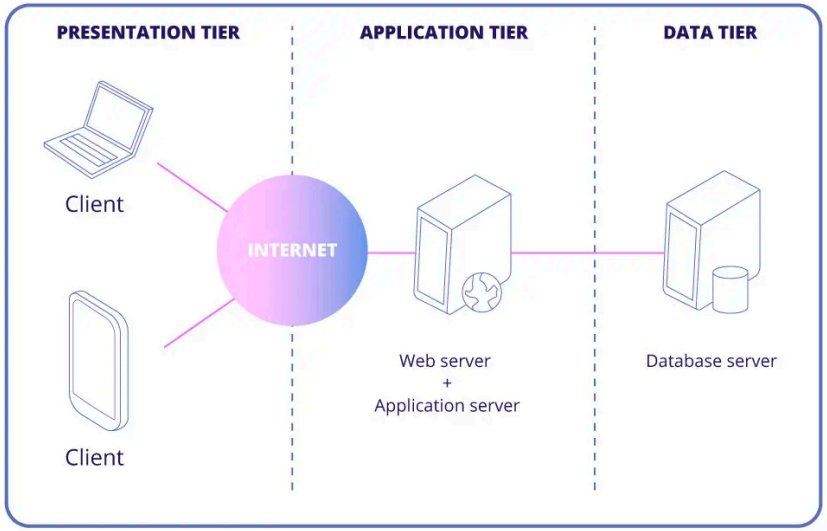
Outline

- Motivation - Background [3’] (sl. 3-4)
- Noria [10’] (sl. 5-12)
  - Introduction
  - Partially-stateful data flow
  - Dynamic data flow
- Implementation Details [2’] (sl. 13-14)
- Evaluation [6’] (sl. 15-20)
- Conclusion [1’] (sl. 18)

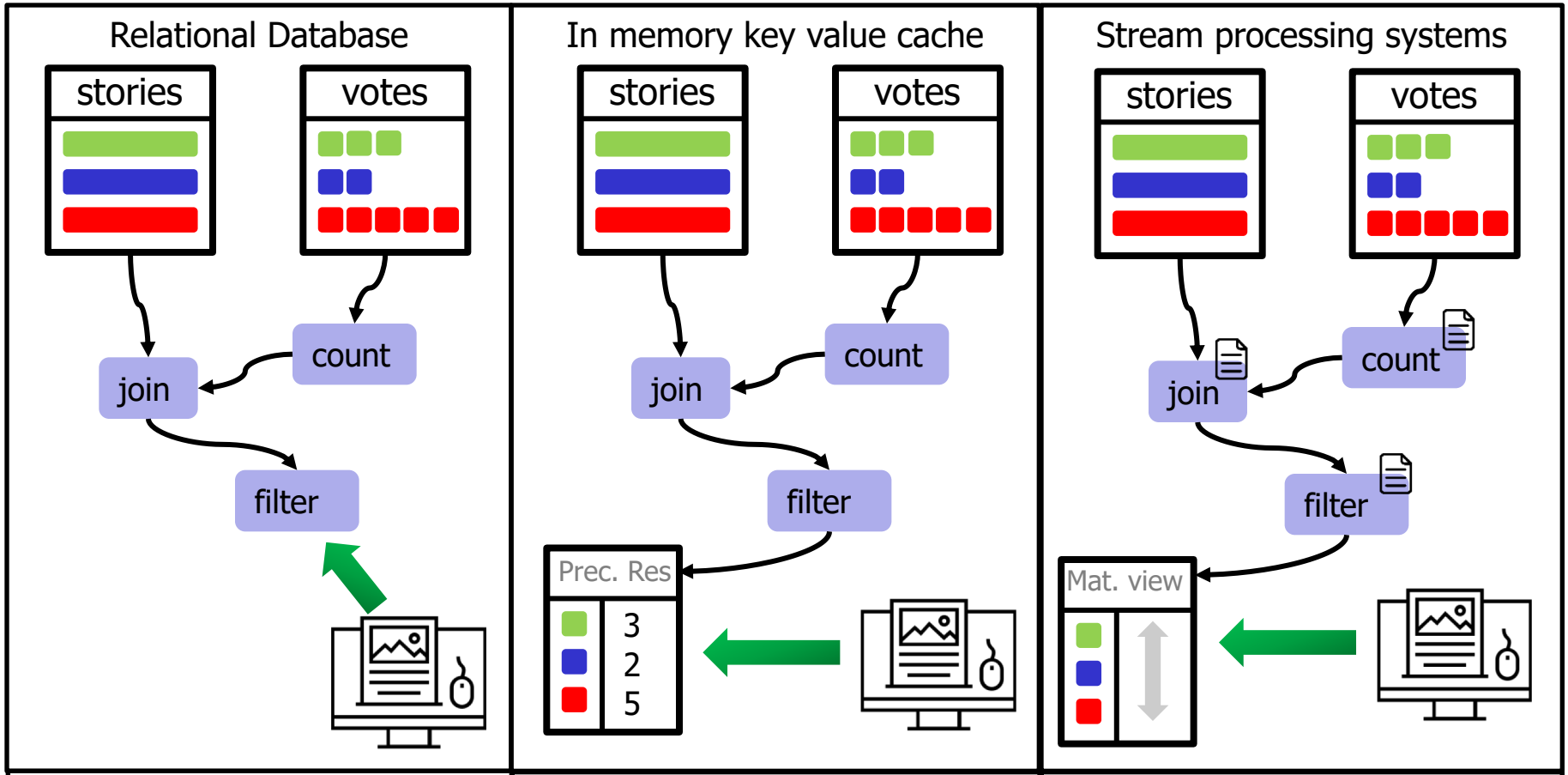
Motivation - Background

- Web applications must serve users at low latency
- Responses are computed by querying data from backend stores (i.e. relational databases)

- Typically, web applications are read-heavy
  - Lobste.rs: 97% reads
  - HotCRP: 88% reads
  - Facebook: 99.8 % reads
    - Bronson et. al. (2013)



Motivation & Background – cont'd

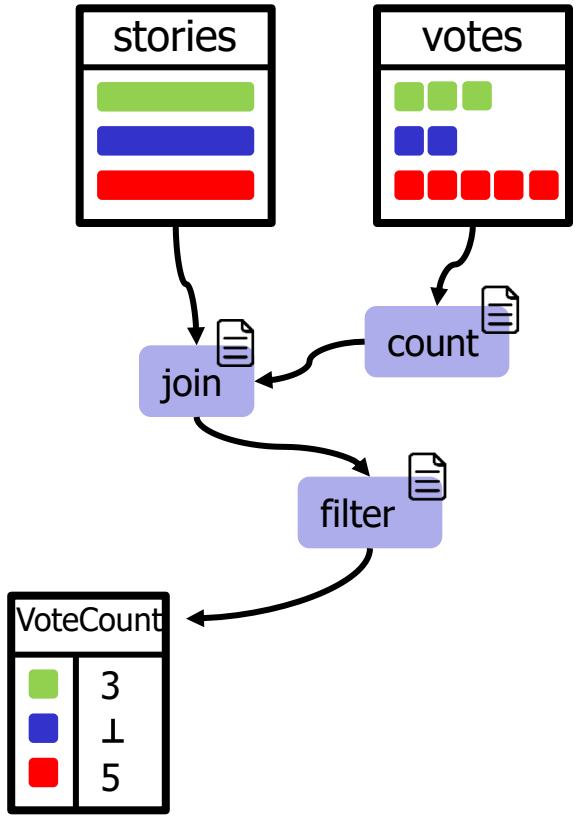


Increased query execution time Repeated work (same queries)	Updates → manual modification, or cache invalidation Complex application-side logic	State can grow rapidly New queries requires restarting

Noria

- Noria combines the best of the three alternatives
  - Fast reads (in-memory key-value cache)
  - Efficiency against updates and parallelism (stream-processing systems)
  - Changing queries and view without downtime (relational databases)

- Key Characteristics
  - Bound memory footprint
  - Live addition/updates of new queries/views
  - Without global coordination



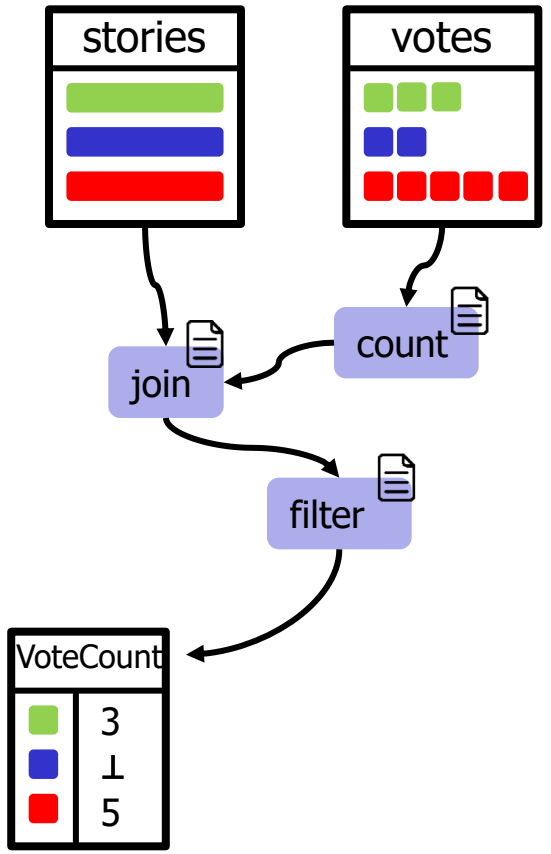
Noria – Challenges

- Limit the size of state and views
  - Partially-stateful data-flow

- Changes to Noria program must adapt the data-flow without downtime
  - Dynamic data-flow

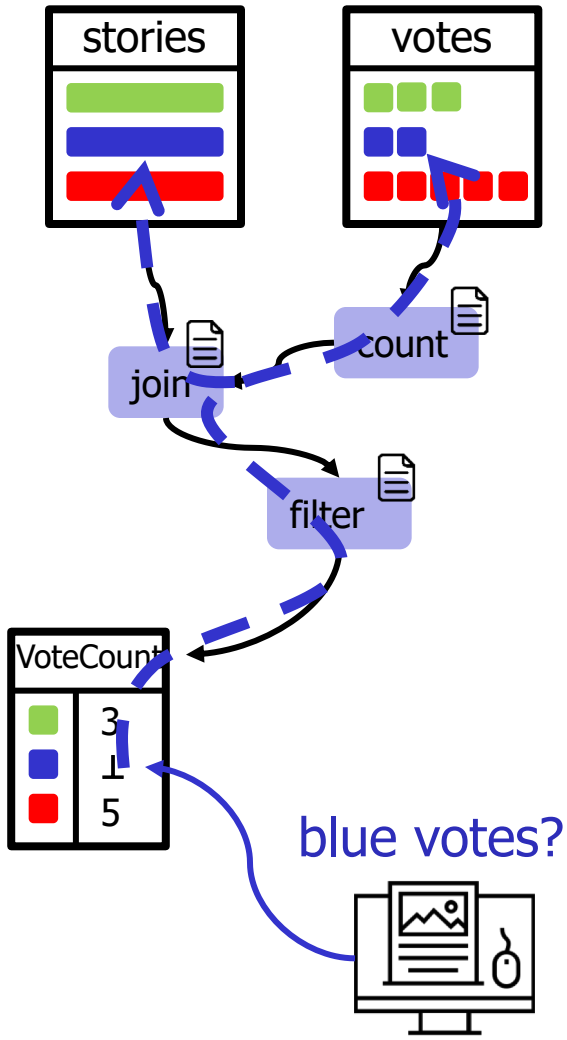
Noria – Partially-stateful data-flow

- Operators maintain only a subset of their state
  - Absent entries (⊥)
- Why absent?
  - They were never filled in (i.e. not requested)
  - They were evicted by the system (i.e. rarely used)
- Benefits
  - It reduces the memory footprint efficiently
    - It does not rely on a “window cache”



Noria – Partially-stateful data-flow

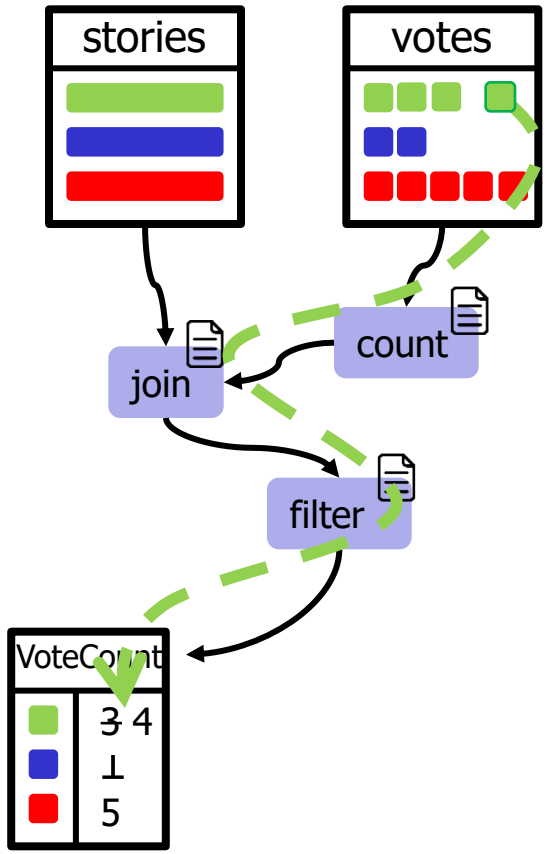
- Read from evicted state?
  - Upquery the data flow stream
- Fill the evicted state
  - In the operators
  - In the external views



Noria – Partially-stateful data-flow

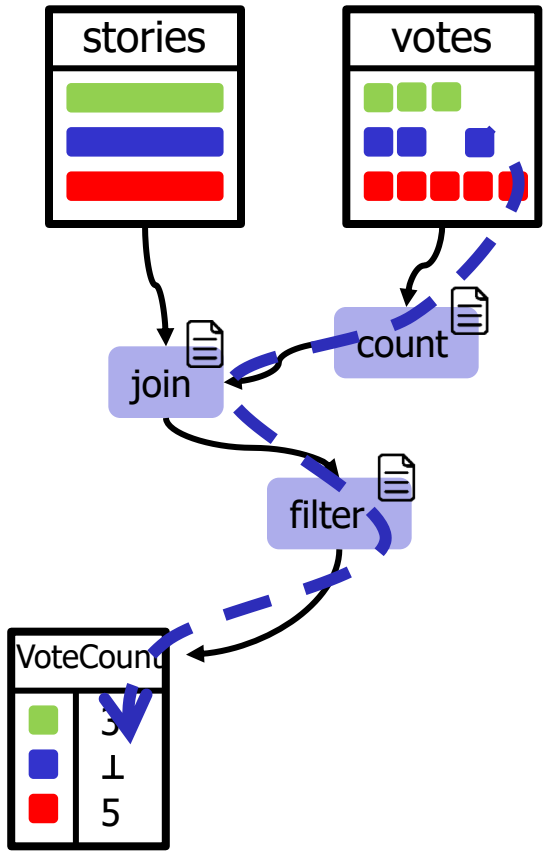
- Writes
  - Applied to a base table and injected in the data flow as an update
  - Eventually updates read and modify external views

- Example
  - A new vote for the green story
  - Stored in the base table (votes)
  - State updated
    - Operators of the data flow
    - External view



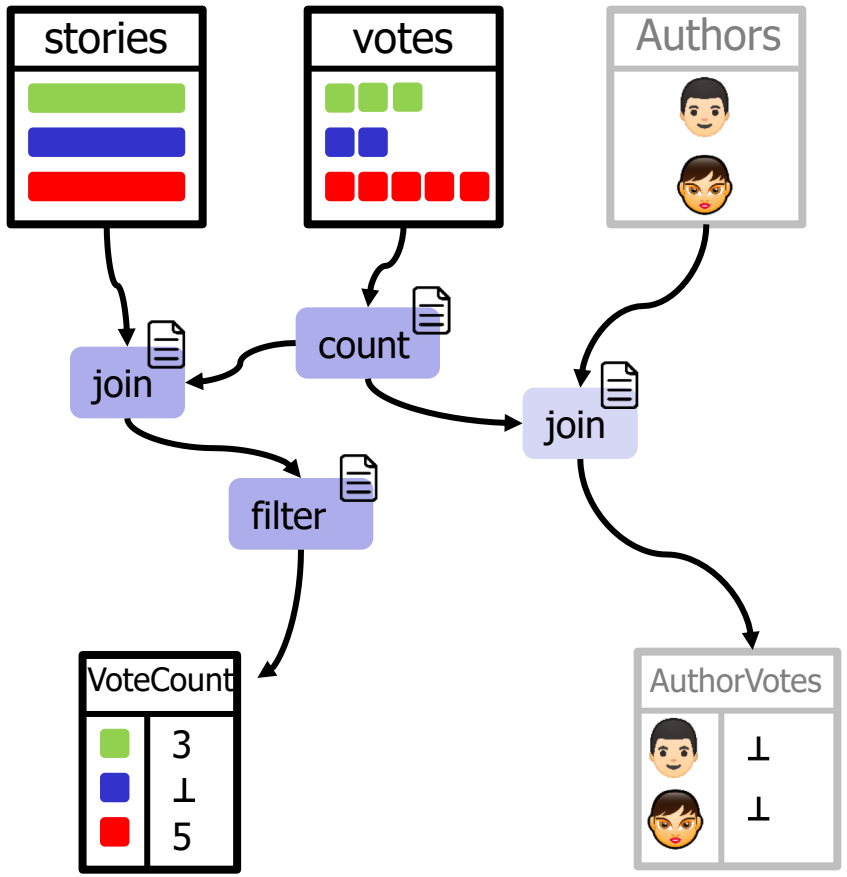
Noria – Partially-stateful data-flow

- If a request for updating an evicted state arrives (⊥) → drop it
- Benefits
  - There is no need to update missing entries
  - They will be updated as soon as they are requested (via an upquery)
- Example
  - A new vote for the green story
  - Stored in the base table (votes)



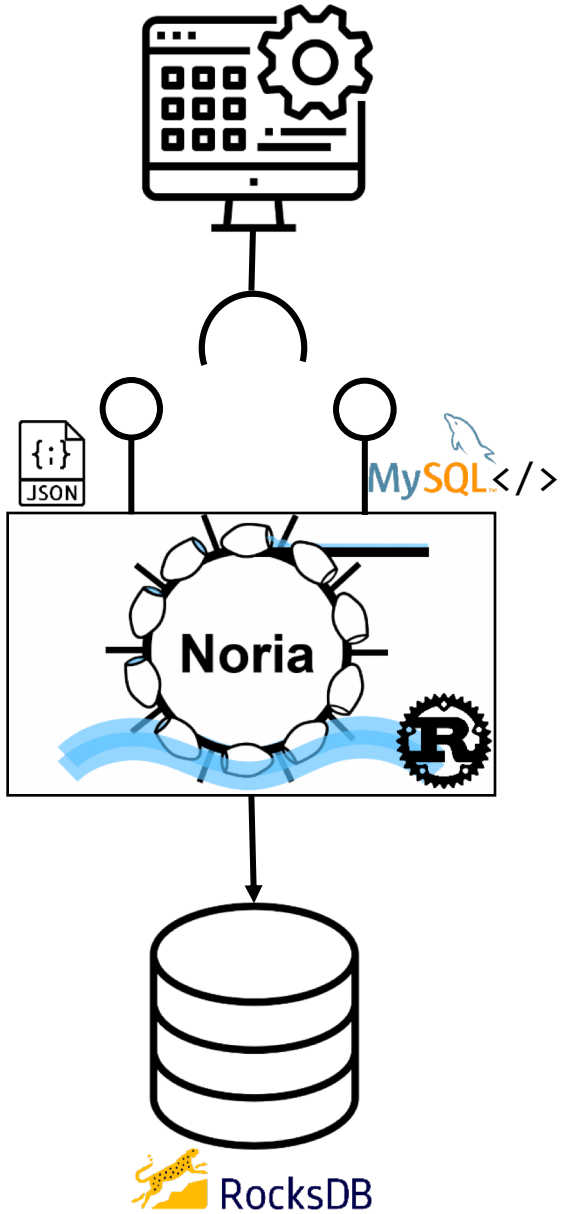
Noria – Dynamic data-flow

- 1st step: transition planning
  - The app provides added and removed expressions
  - Noria computes the required changes (this allows re-use of operators)
- 2nd step: data flow transition
  - Adds new operators
  - Originally their state is empty
  - Remove obsolete operators
- Challenges
  - Avoid state duplication
  - Continuous supports of reads and writes

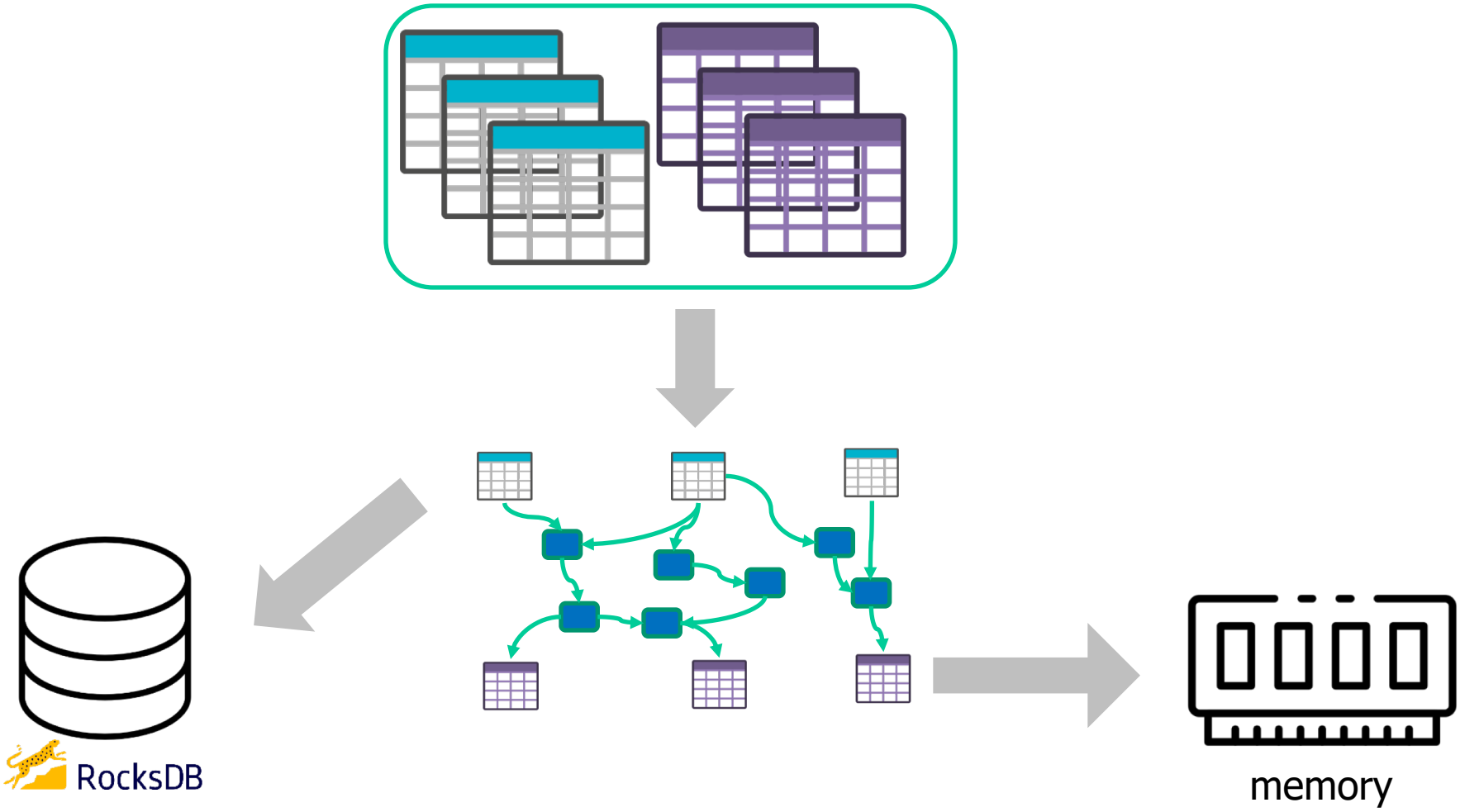


Implementation details

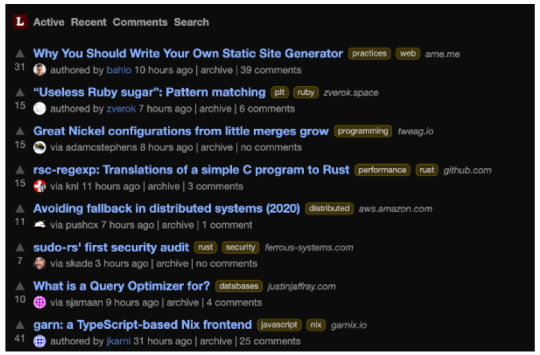
- Implemented in Rust
- Operation mode
  - Single server
  - Cluster of servers
- 2-way parallelism
  - Multiple shards - multiple Noria instances
  - Each instance: multiple threads to handle its shard
- Each Noria instance maintains a complete copy of the data-flow graph, but holds only the state of its operator shards
- Zookeeper: Controller election
  - Changes in data flow, shards assignment



Implementation details



Evaluation

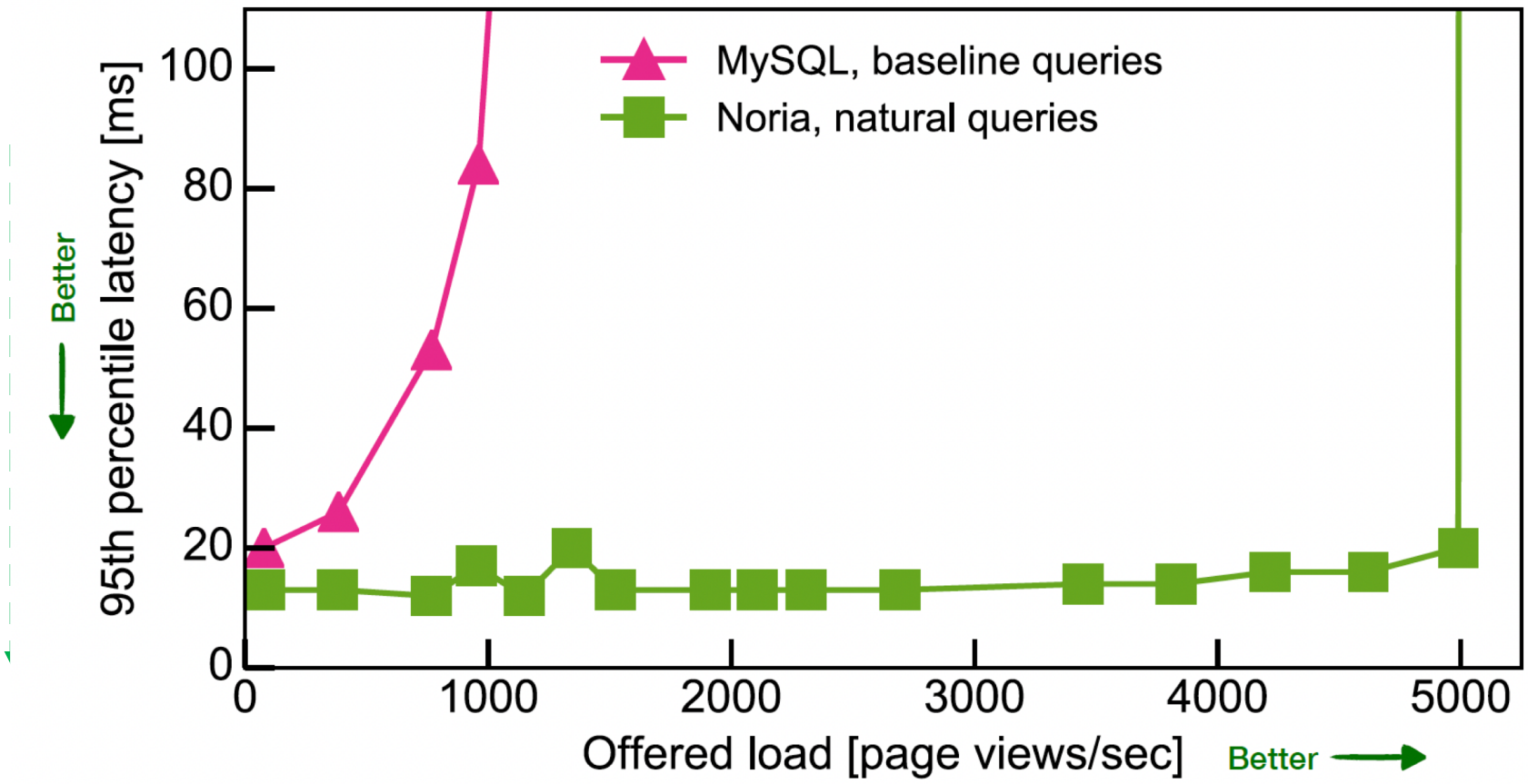


- Lobsters (http://lobste.rs)
  - 9.2K users & 40K Stories & 120K Comments
  - Emulating the production load
- Ruby-on-Rails web application with MySQL backend
- Hand-optimized SQL queries by developers to precompute various aggregations
- Noria data flow: 235 operators & 35 views

- Experiments
  - Server: Amazon EC2 c5.4xlarge instance - 16 vCPUs
  - Client: Amazon EC2 c5.4xlarge instance

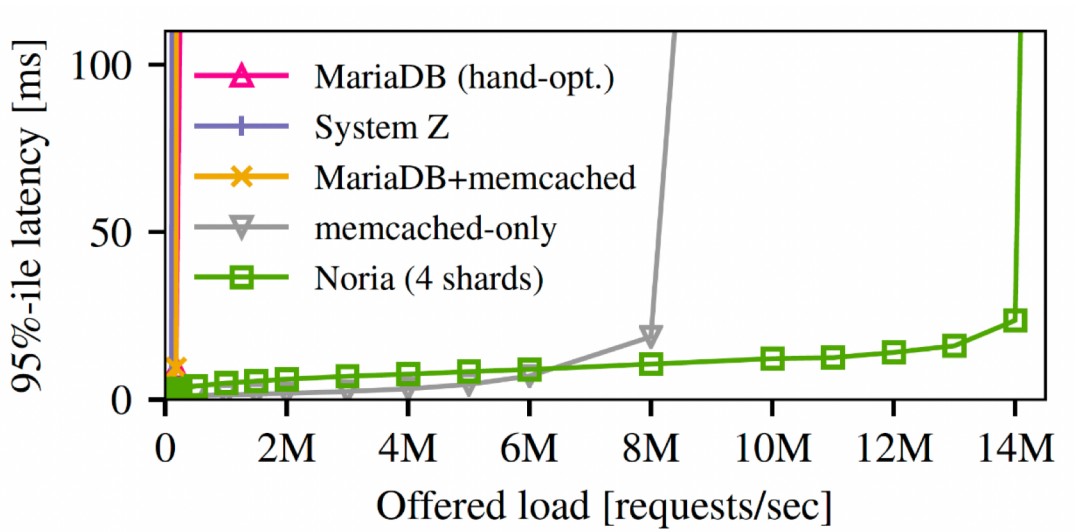
Evaluation - cont'd

- Noria performance improvements for real web applications



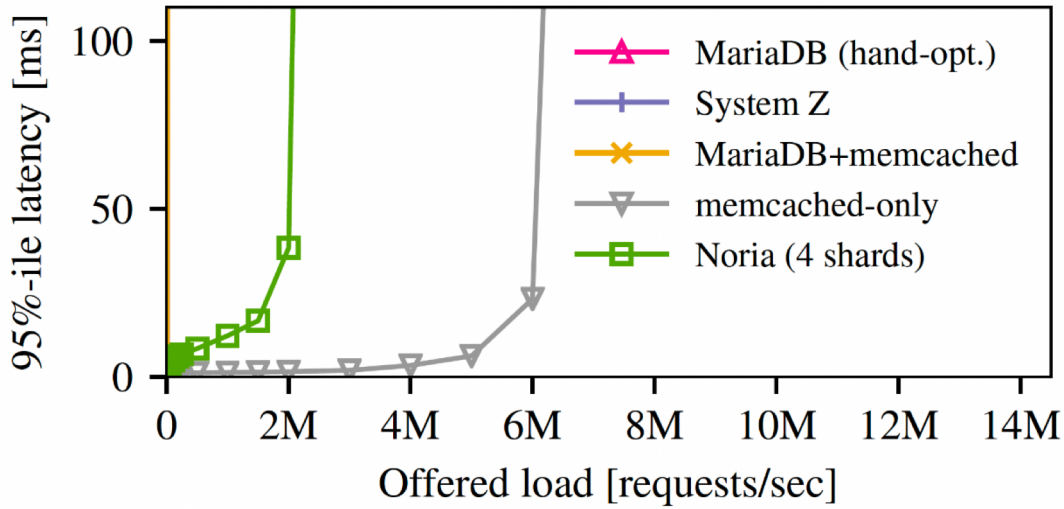
Evaluation - cont'd

- Comparison of Noria with other alternatives



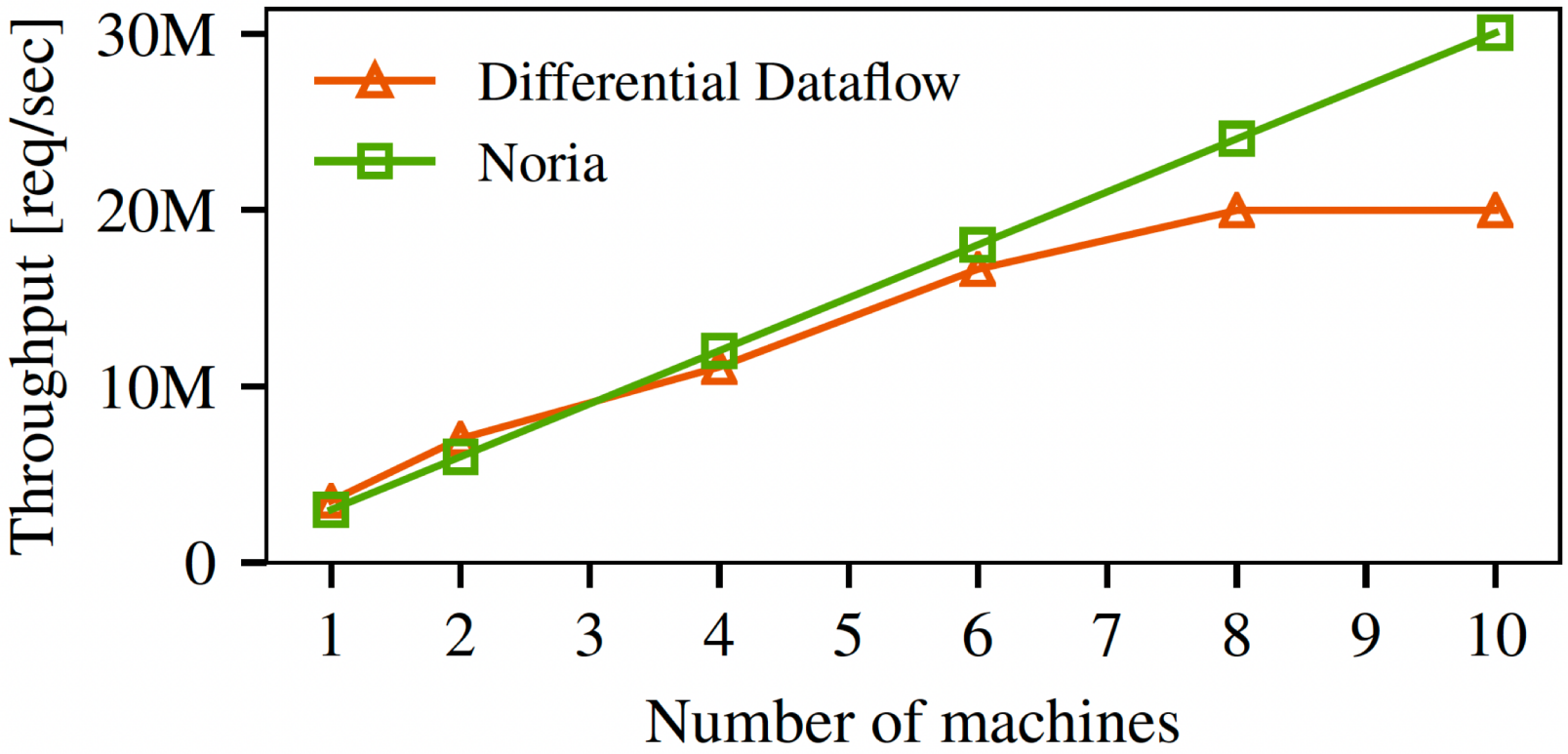
Read-heavy workload
(95% R / 5% W)

Mixed read-write
workload
(50% R / 50% W)



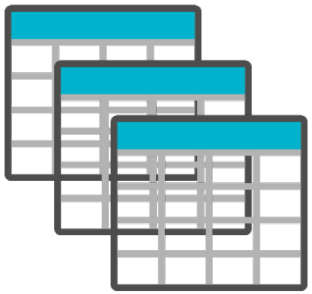
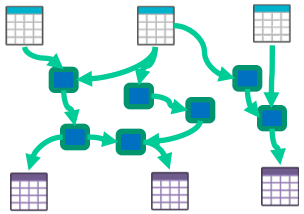
Evaluation - cont'd

- Utilization of multiple servers



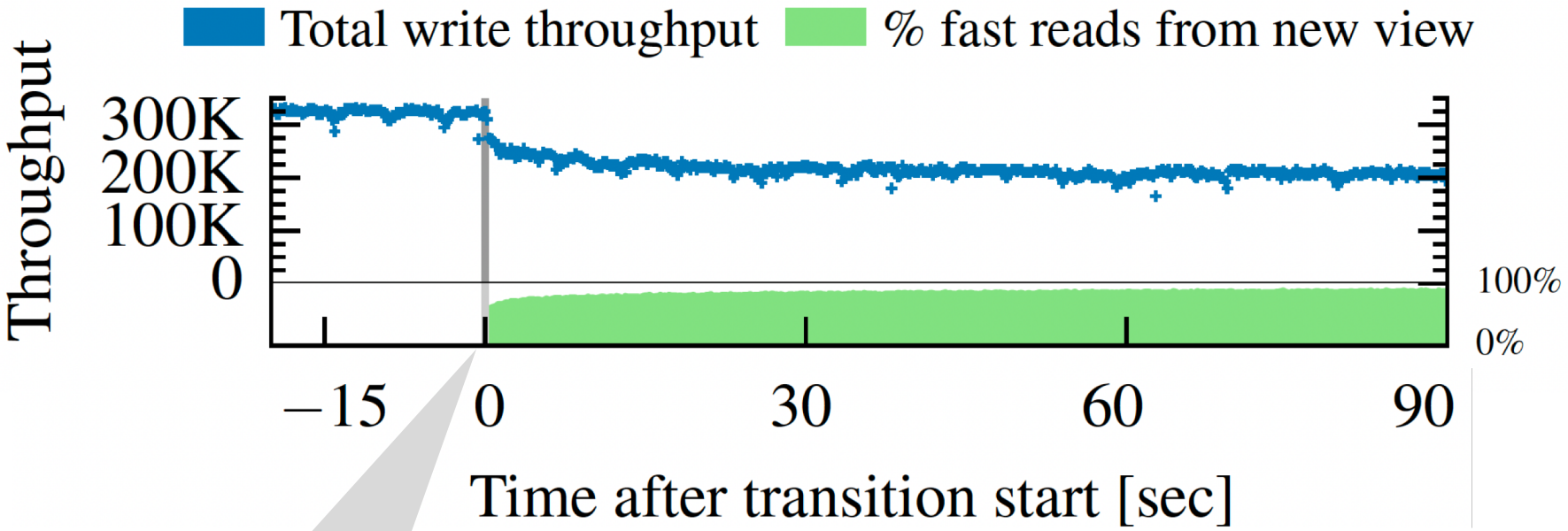
Evaluation - cont'd

- Space overhead of the data-flow and performance with limited resources
- Goal: <100ms latency at 2,300 pages/sec (Lobsters production load)

	Lobsters size	x10 Lobsters size
 base tables	137 MB	866 MB
 data flow	525 MB 60% of all state	2.6 MB 38% of all state

Evaluation - cont'd

- Adaptability of Noria to new queries and schema updates



Instantaneous transition, no downtime for writes

Conclusion

- A partially-stateful data flow model tailored for read-heavy web applications
- Fast reads with restricted memory footprint
- Near instantaneous dynamic transition for data flow graphs, in response to query changes
- Prototype implementation which is backwards-compatible with MySQL protocol interface
- With limitations
  - It only guarantees eventual consistency
  - Lacks support for some SQL keywords

References and Links

- Citation
  - Gjengset, J., Schwarzkopf, M., Behrens, J., Araújo, L.T., Ek, M., Kohler, E., Kaashoek, M.F. and Morris, R., 2018. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 213-231).
- Link to paper and slides
  - https://www.usenix.org/conference/osdi18/presentation/gjengset
- Source code
  - https://github.com/mit-pdos/noria
- Link to these slides
  - https://github