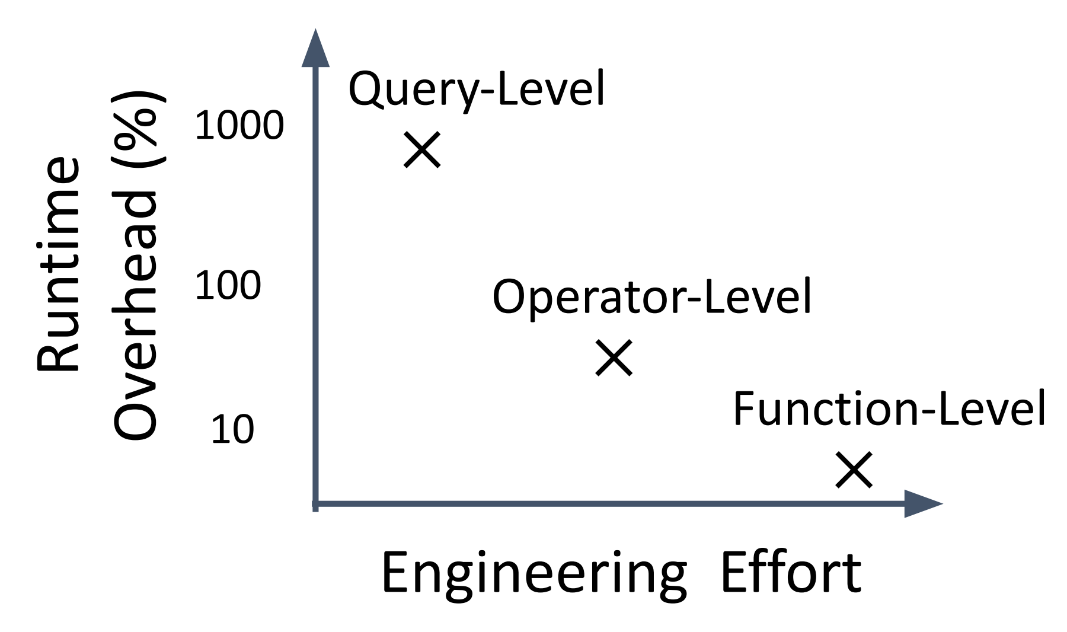
Lineage Capture Trade-offs: A Case Study in DuckDB

Haneen Mohammed | Columbia University, ham2156@columbia.edu
Eugene Wu | Columbia University, ewu@cs.columbia.edu

Best Lineage Capture Method in High Performance Systems?

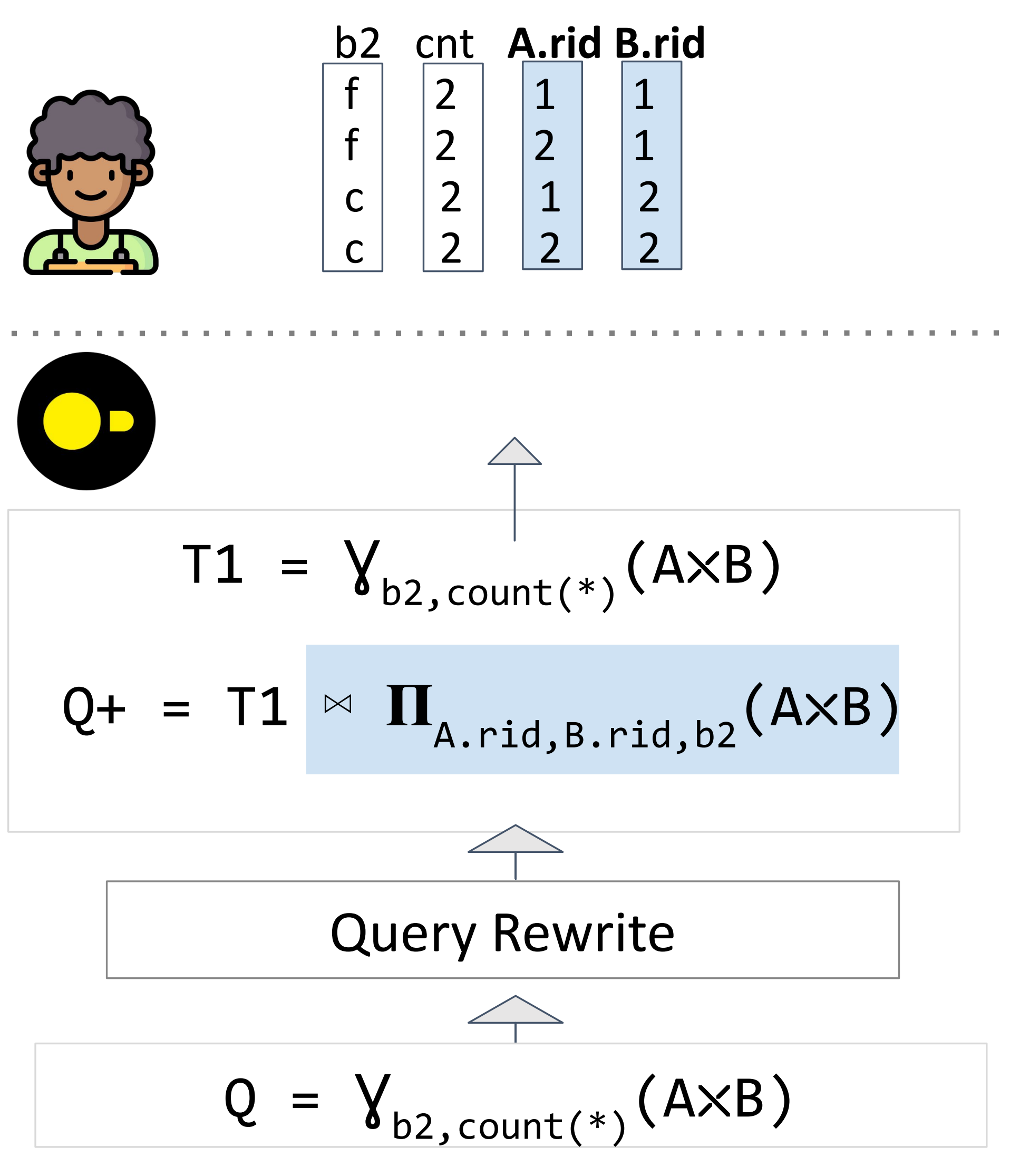
We compare three main methods that instrument queries at different granularities



Experiment Design

- Lineage capture methods implemented in different systems, not comparable
- Implemented three methods in DuckDB for apples-to-apples comparison
- Engineering effort estimated by number of files modified

Query-Level

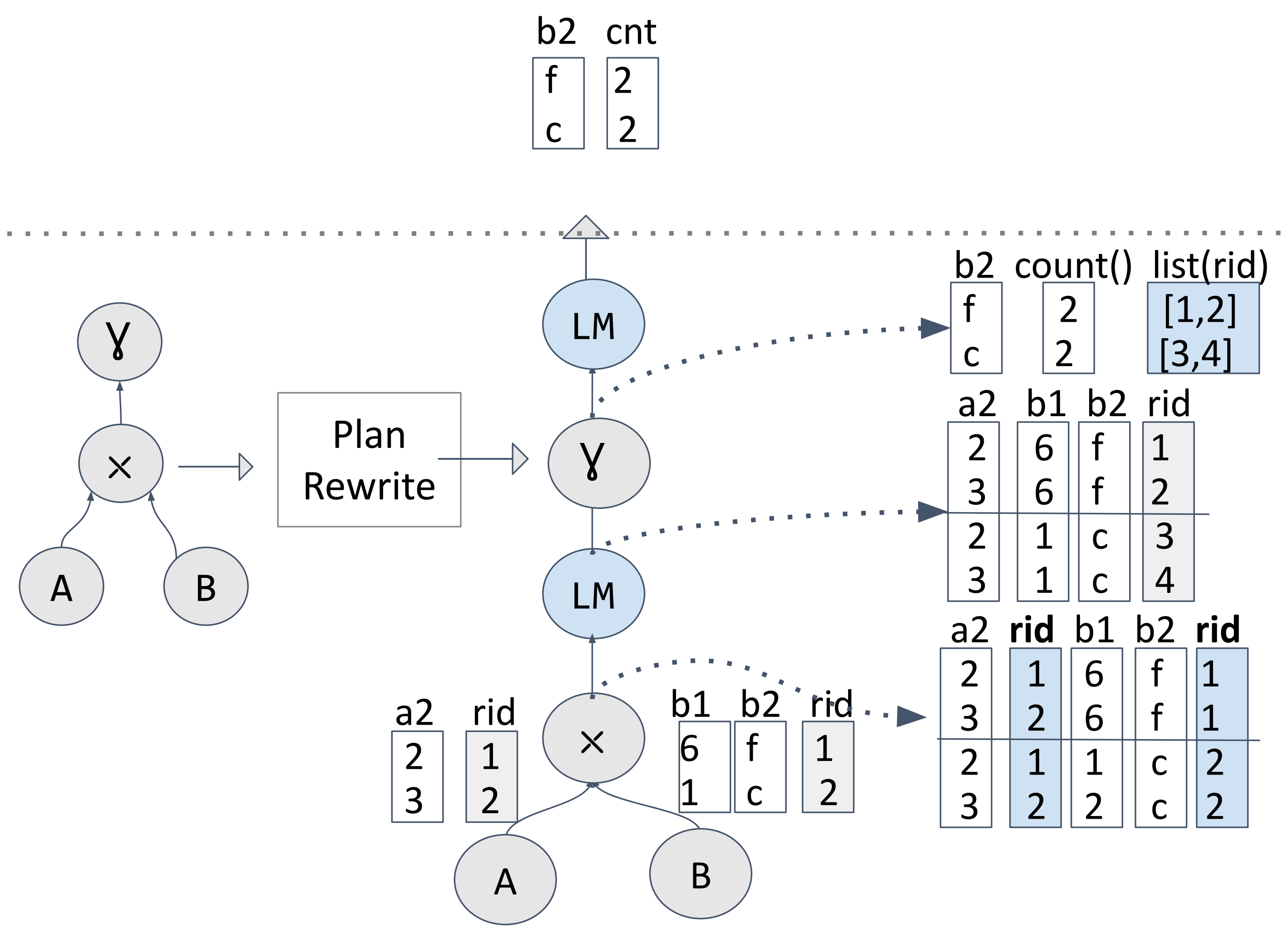


What? PERM-style query rewriting

Pros: DBMS agnostic

Cons: Logically annotates Q with prov annotations
Accumulated annotations slow down exec

Operator-Level

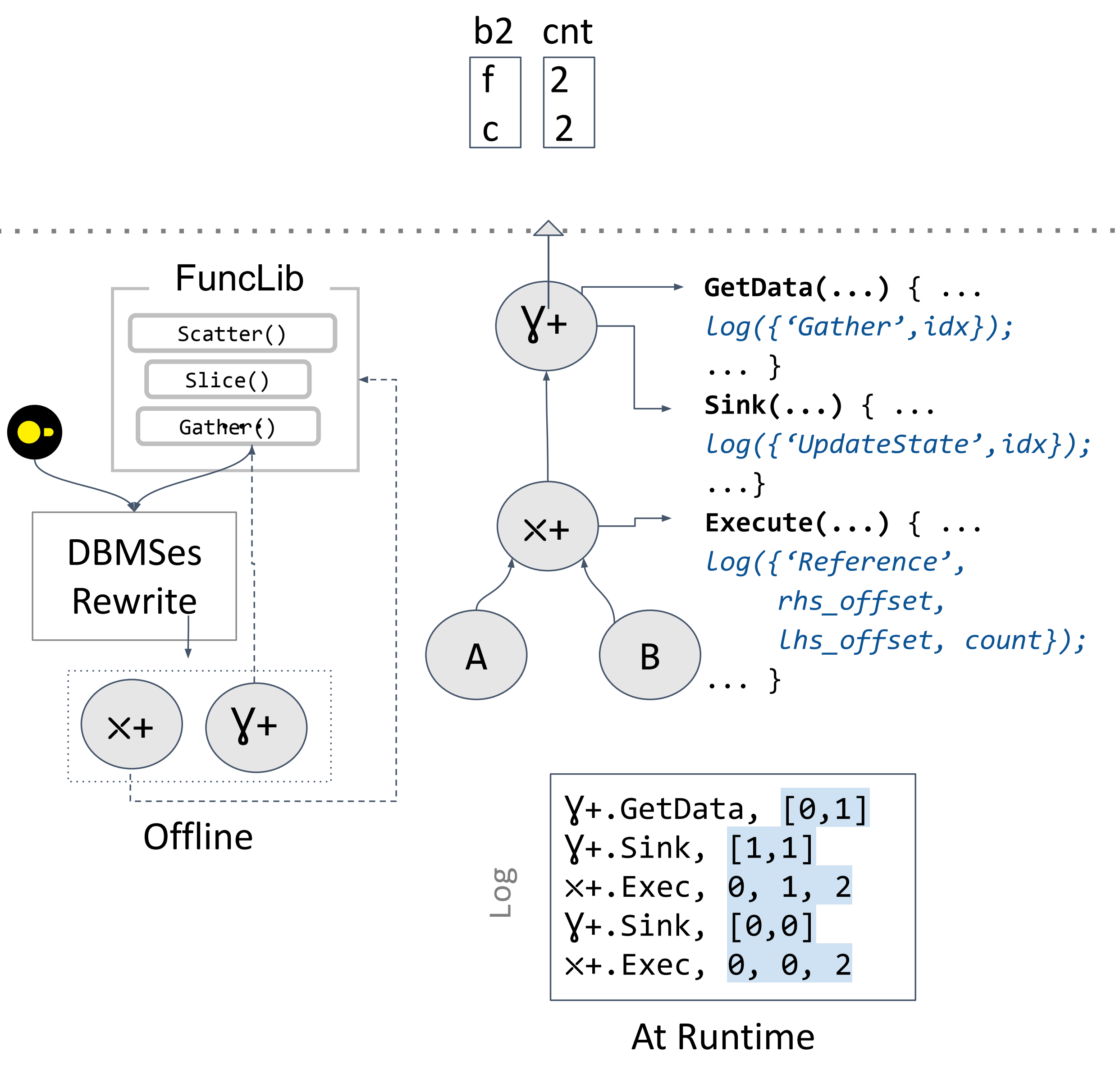


What? PERM-style rewrites *per-operator* + new LM operator to strip away annotations.

Pros: Doesn't accumulate annotations during execution

Cons: Creating annotations (blue columns) still expensive at pipeline breakers
Must modify query planner

Function-Level



```
GetData(...) { ...
Log({'Gather',idx});
... }
Sink(...) { ...
Log({'UpdateState',idx});
...}
Execute(...) { ...
Log({'Reference',
     rhs_offset,
     lhs_offset, count});
... }
```

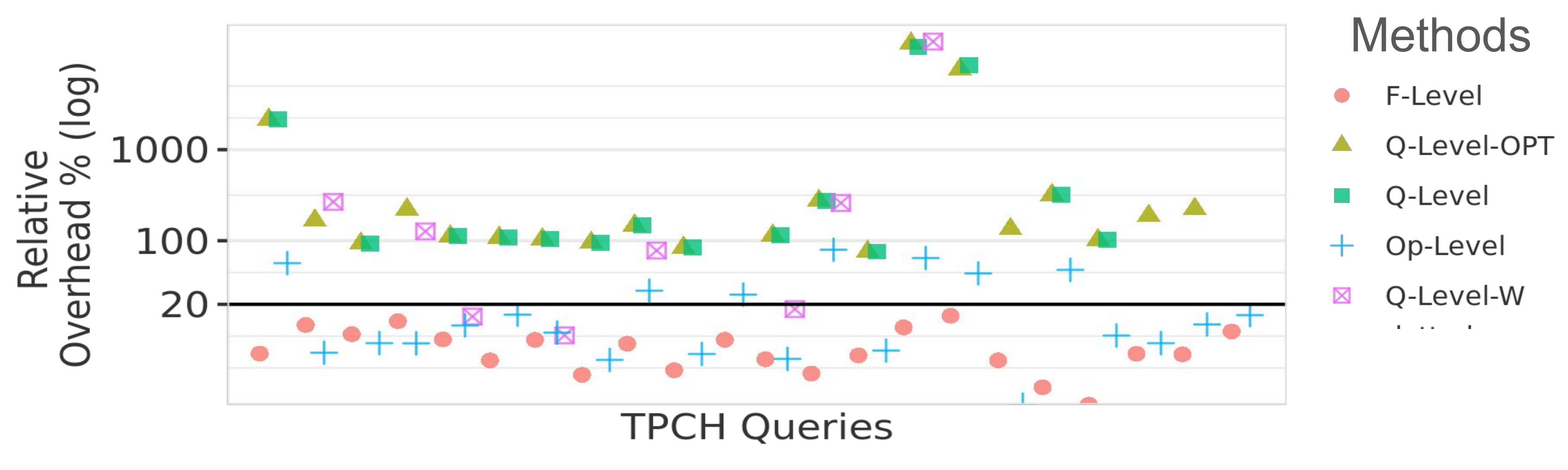
```
γ+.GetData, [0,1]
γ+.Sink, [1,1]
×+.Exec, 0, 1, 2
γ+.Sink, [0,0]
×+.Exec, 0, 0, 2
```

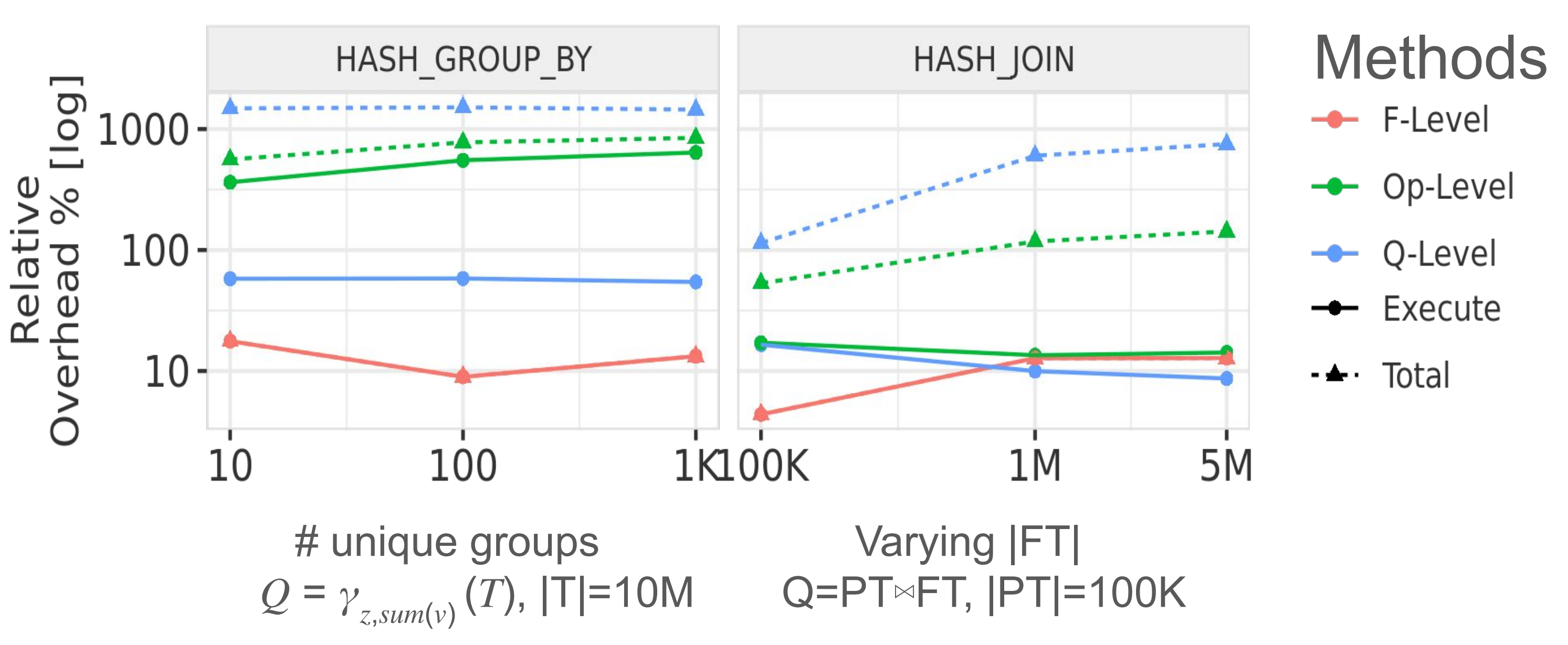
What? Persists program variables that already encode data-movement (lineage) during execution.

Pros: Logs existing variables, avoids annotations

Cons: Must modify engine implementation

Results





Takeaways

- Query-level: DBMS-agnostic but too slow
- Operator-level: Efficient for pipelined operators and integrates cleanly with extensible query planner
- Function-level: Faster but requires invasive DBMS changes
- Hybrid of Function- and Operator-level may offer the best trade-off between performance and engineering effort.