Skeena: Efficient and Consistent Cross-Engine Transactions

Jianqiu Zhang, **Kaisong Huang,** Tianzheng Wang - Simon Fraser University
King Lv - Huawei Cloud Database Innovation Lab

SIGMOD 2022

https://github.com/sfu-dis/skeena

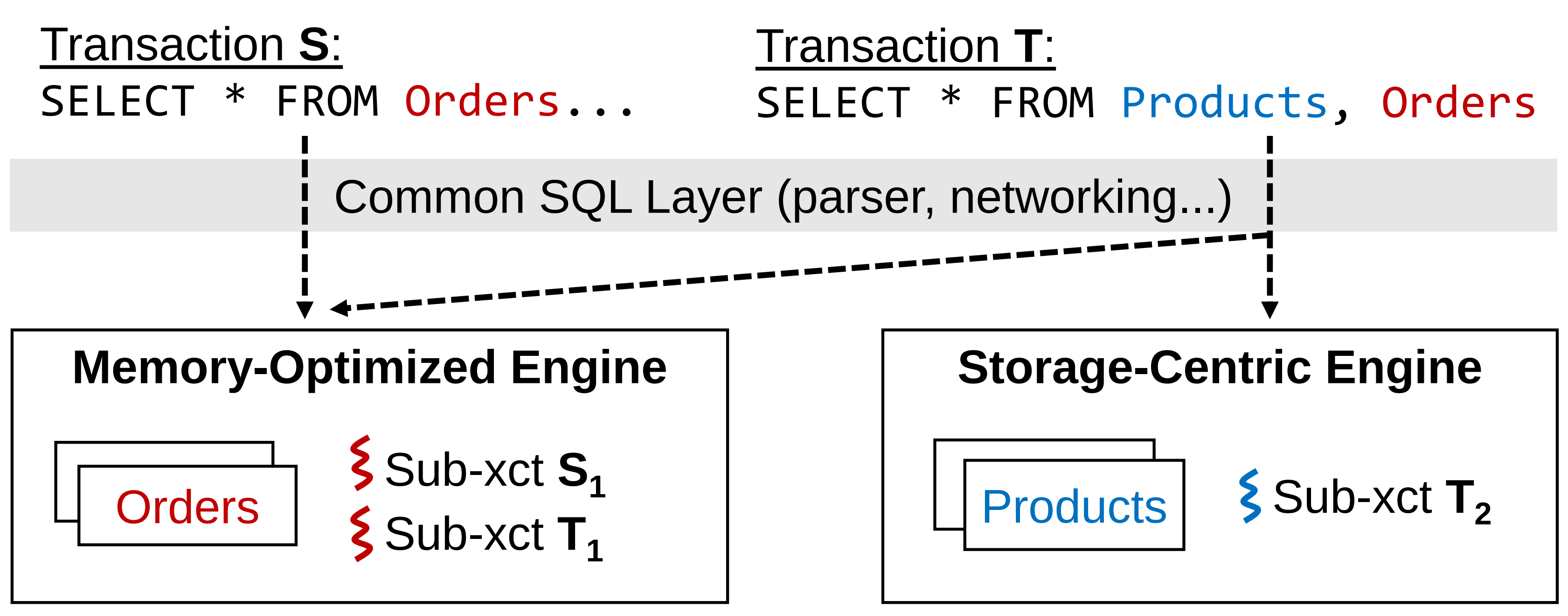
What? Modern DBMSs support multiple engines, but applications can't cross engine boundaries.
Why? Lack of cross-engine support in terms of correctness, performance and programmability.
How? Devise a lightweight snapshot tracking structure and an atomic commit protocol.

DBMSs Going Multi-Engine

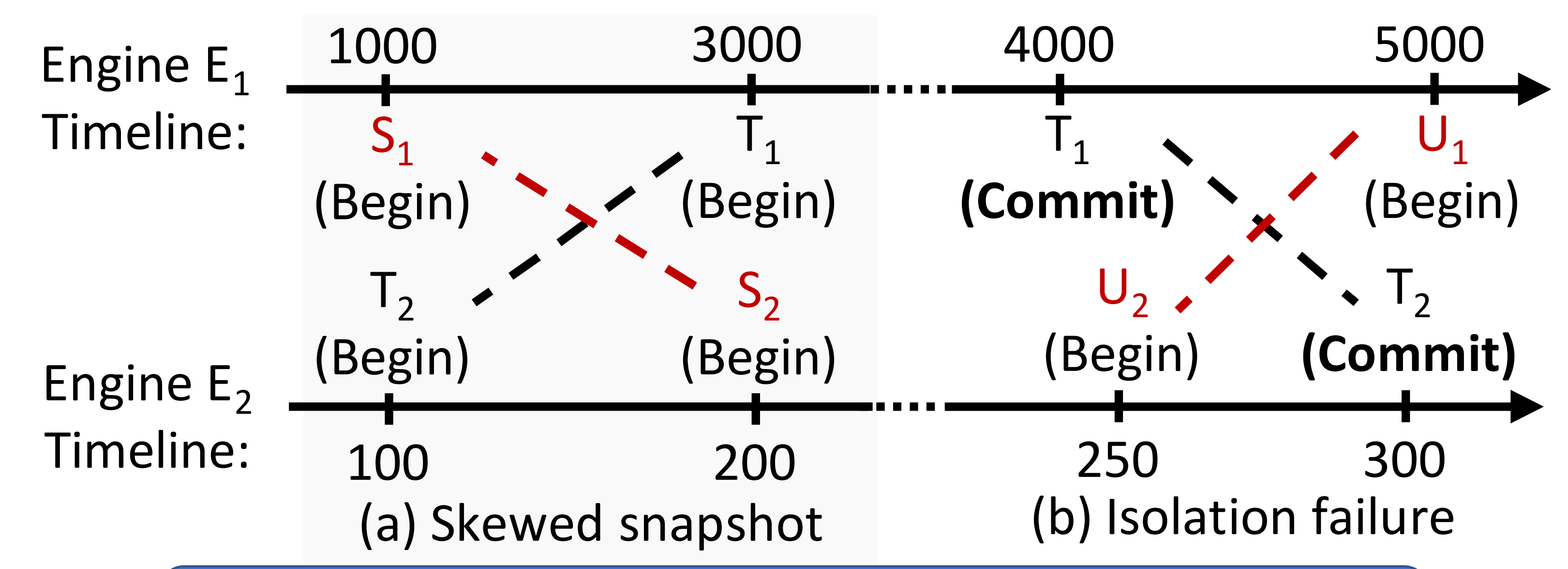
- **Memory-optimized OLTP engines**
  - Orders of magnitude better perf.
- **Storage-centric engines still useful**
  - Cost-effective, backward compatibility

Desirable: multiple engines in one system + use the right engine for the right data and workloads

Cross-Engine Transactions

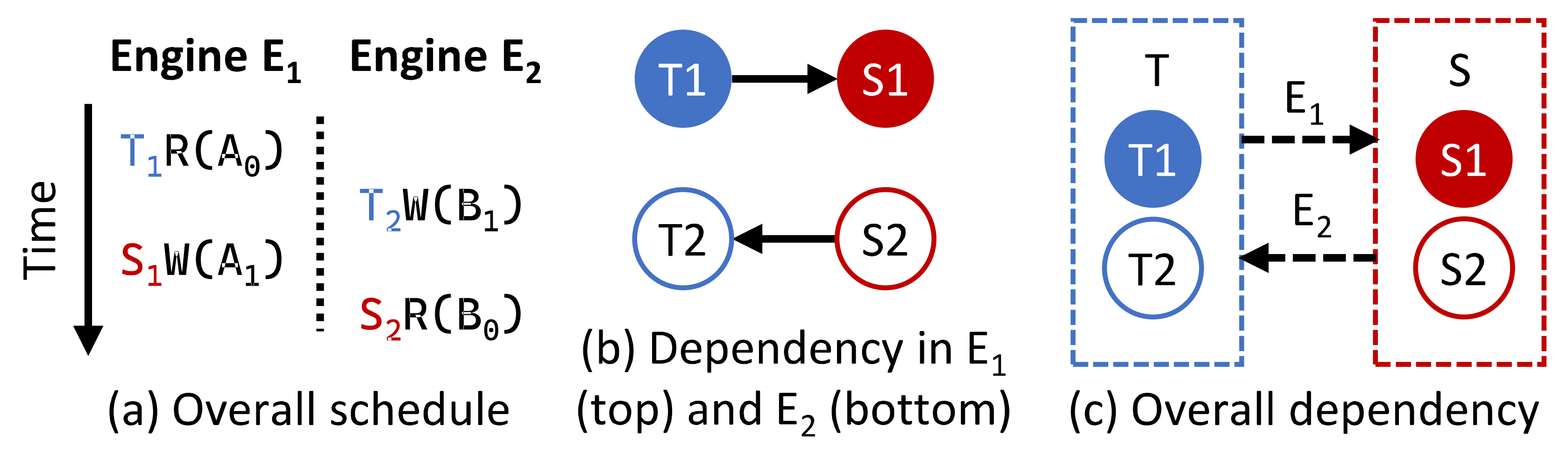


Anomaly 1: Inconsistent Snapshots



(a) S uses an older (newer) snapshot in E_1 (E_2).
(b) U sees T_1's results, but does not see T_2's.

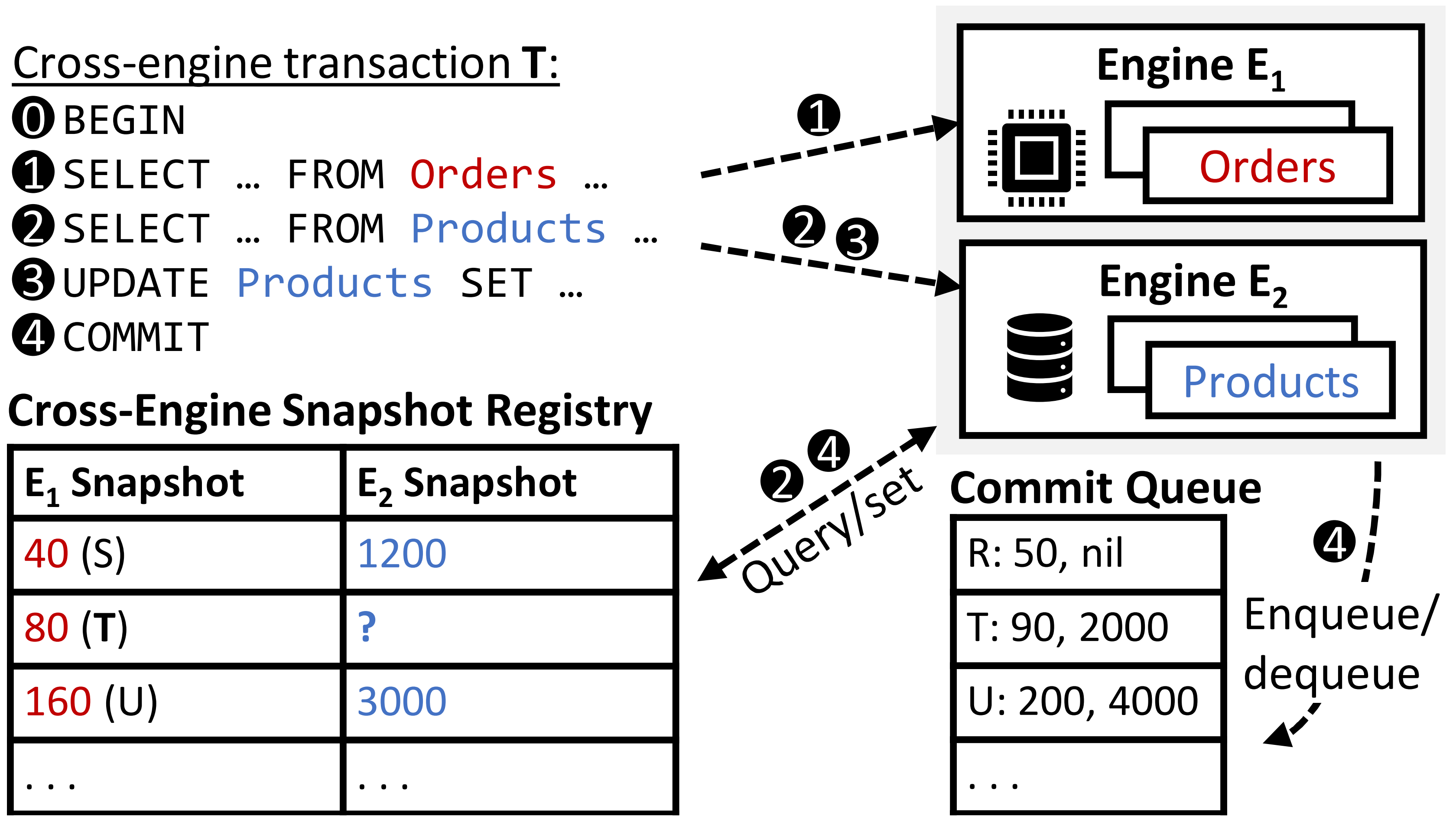
Anomaly 2: Serializability



(a) Each engine executes a serializable schedule (b) without cyclic dependencies. (c) Overall cyclic dependency between *T* and *S* .

Skeena Overview

Design principles: 1) low overhead, 2) engine autonomy, 3) full functionality, 4) transparent adoption



Cross-engine transaction **T**:

```
⓿BEGIN
❶SELECT … FROM Orders …
❷SELECT … FROM Products …
❸UPDATE Products SET …
❹COMMIT
```

Cross-Engine Snapshot Registry

E_1 Snapshot	E_2 Snapshot
40 (S)	1200
80 (**T**)	?
160 (U)	3000
. . .	. . .

Commit Queue

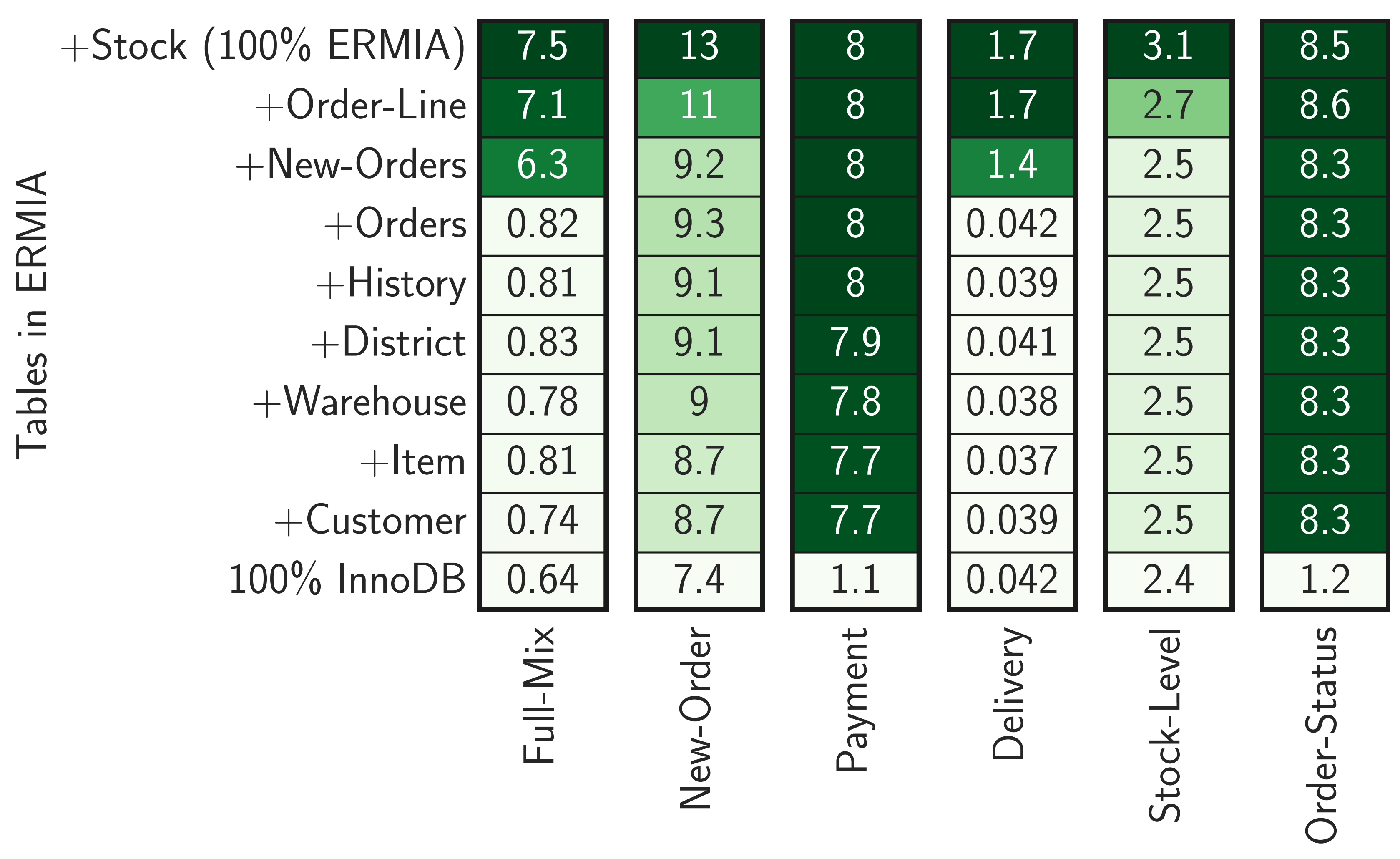
R: 50, nil
T: 90, 2000
U: 200, 4000
. . .

❶ Transactions access data without explicitly declaring whether they are cross-engine.

❷ Upon accessing an additional engine, the transaction
❸ consults CSR to obtain a proper snapshot.

❹ Cross-engine transactions use CSR for commit check and if passed, goes through the pipelined commit protocol.

Recommended End-to-End Cross-Engine TPC-C



Tables in ERMIA	Full-Mix	New-Order	Payment	Delivery	Stock-Level	Order-Status
+Stock (100% ERMIA)	7.5	13	8	1.7	3.1	8.5
+Order-Line	7.1	11	8	1.7	2.7	8.6
+New-Orders	6.3	9.2	8	1.4	2.5	8.3
+Orders	0.82	9.3	8	0.042	2.5	8.3
+History	0.81	9.1	8	0.039	2.5	8.3
+District	0.83	9.1	7.9	0.041	2.5	8.3
+Warehouse	0.78	9	7.8	0.038	2.5	8.3
+Item	0.81	8.7	7.7	0.037	2.5	8.3
+Customer	0.74	8.7	7.7	0.039	2.5	8.3
100% InnoDB	0.64	7.4	1.1	0.042	2.4	1.2

Implementation

* ERMIA (main-memory) + InnoDB (traditional) in MySQL

Three recommended table placement schemes:

- ***New-Order-Opt***: Customer and Item in ERMIA to optimize the New-Order transaction.
- ***Payment-Opt***: Only Customer in ERMIA to optimize the Payment transaction
- ***Archive***: All the tables but History in InnoDB for lower storage cost