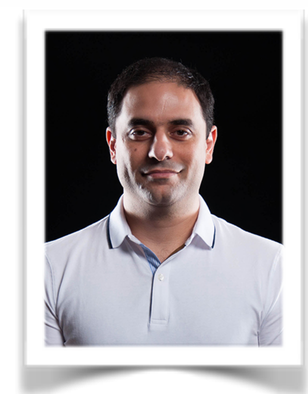
Q-Store: Distributed, Multi-Partition Transactions via Queue-Oriented Execution and Communication



Thamir M. Qadah*
School of Electrical and Computer Engineering
PURDUE UNIVERSITY

Suyash Gupta
Department of Computer Science
UC DAVIS UNIVERSITY OF CALIFORNIA

Mohammad Sadoghi
Department of Computer Science
UC DAVIS UNIVERSITY OF CALIFORNIA

* Also With Umm Al-Qura University, Makkah, Saudi Arabia

Cloud Computing Trends

The rise of cloud computing

Large core counts

Large main-memory

	RAM	vCPUs
AWS[1]	24 TB	448
MSA[2]	12 TB	416
GC[3]	12 TB	416

[1] https://aws.amazon.com/ec2/instance-types/high-memory/

[2] https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/

[3] https://cloud.google.com/compute/docs/machine-types

Distributed Commit Protocols

- Two-phase Commit (2PC)
- **Very good general solution and widely used**
- Adds **overhead per transaction**
- **Can we avoid using it?**

Challenge ???

Distributed Deterministic Transaction Processing

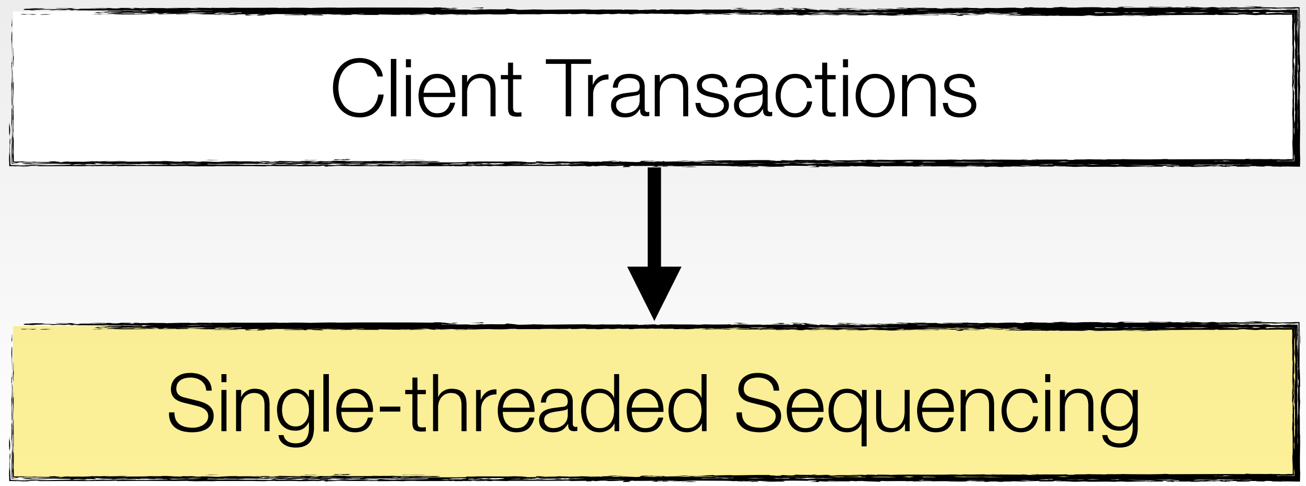
- Provides strict serializability
- **Avoids** non-deterministic transaction **aborts** due to concurrency control
- Removes the **coordination** for transaction-commit from the **critical path**
- Key limitations: requires **knowledge of full read/write sets of transactions prior to execution**

Calvin Overview

Thomson et al. SIGMOD’12

Calvin Overview

Thomson et al. SIGMOD'12



Calvin Overview

Thomson et al. SIGMOD'12

Client Transactions

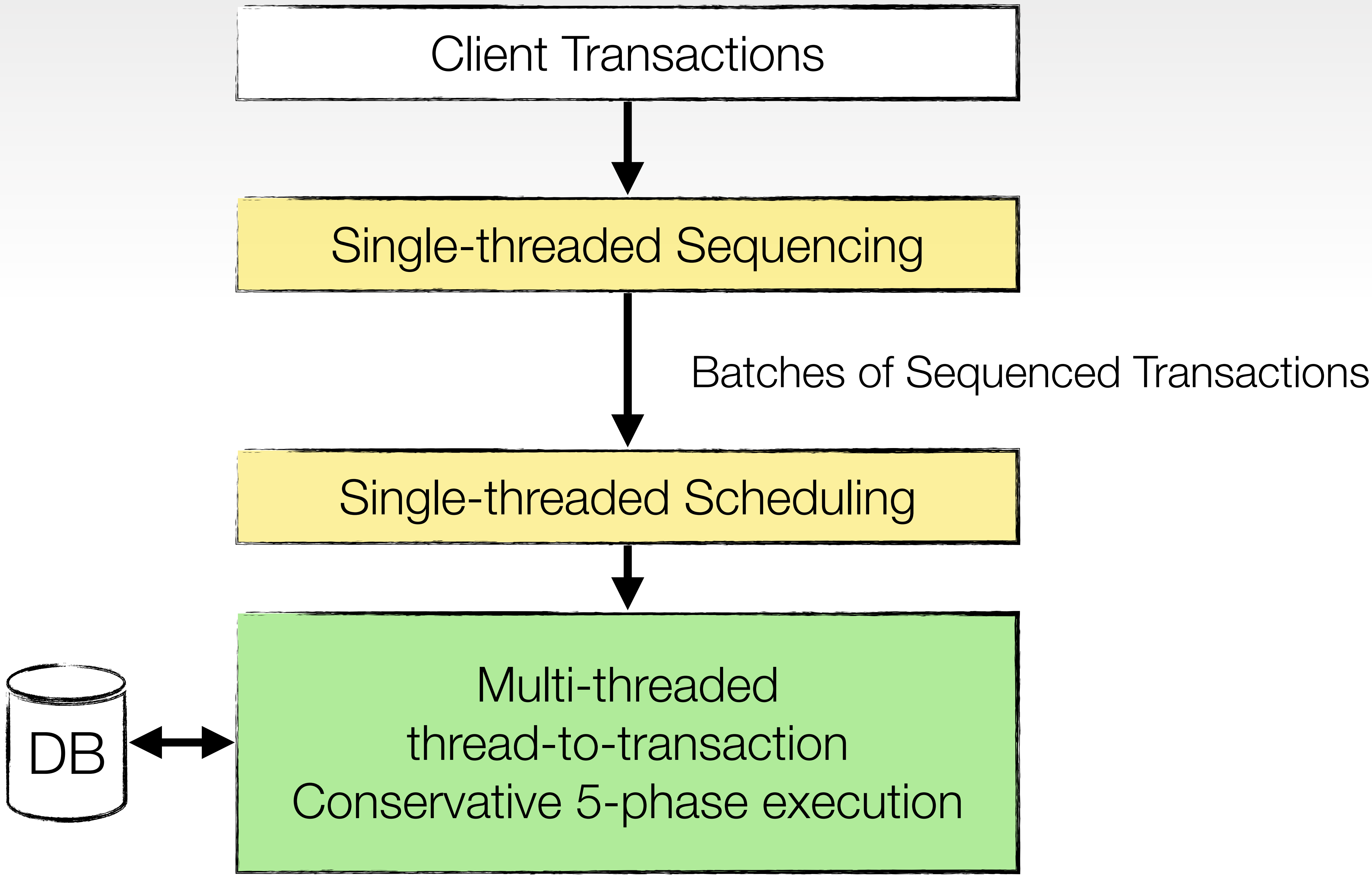
Single-threaded Sequencing

Batches of Sequenced Transactions

Single-threaded Scheduling

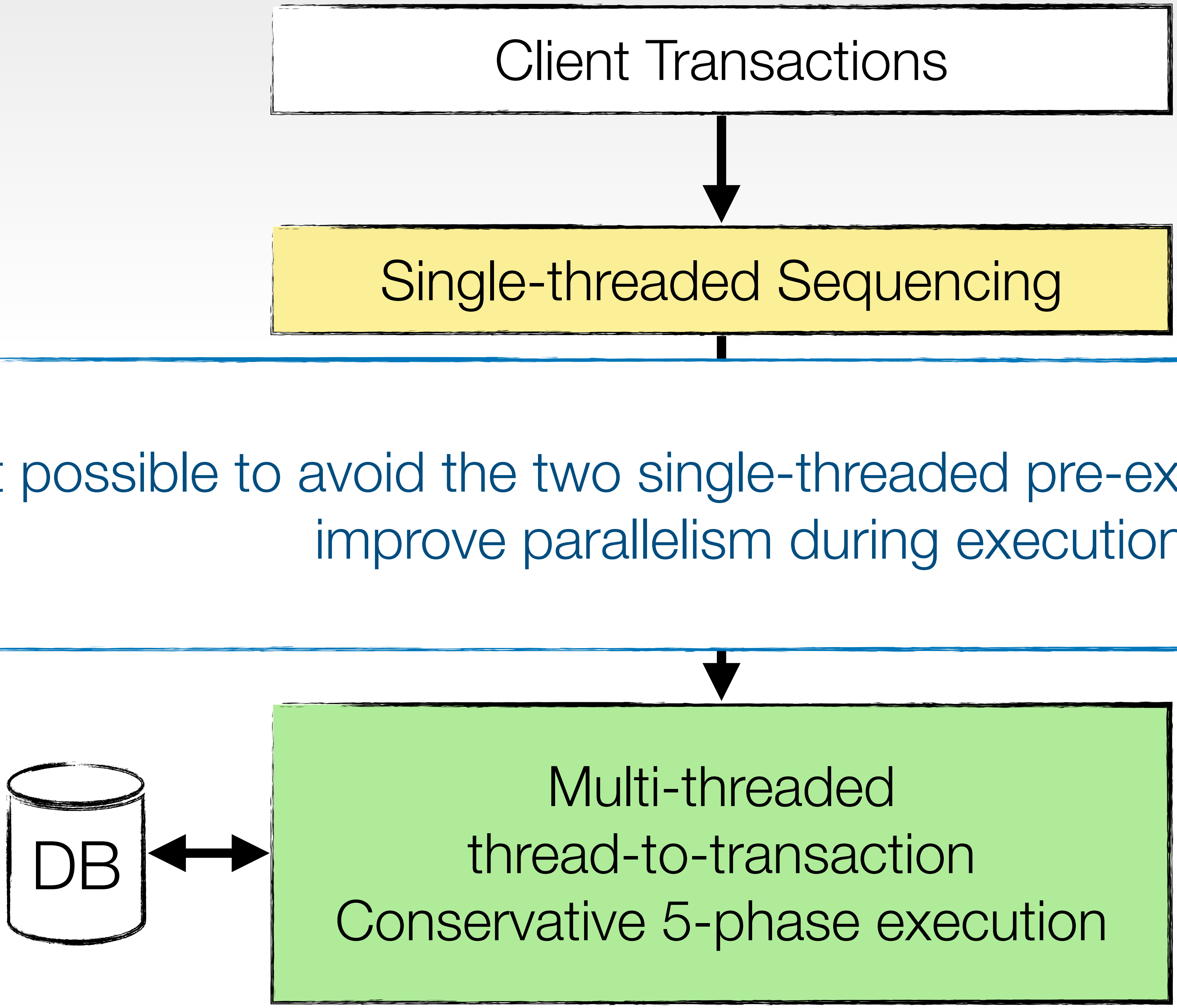
Calvin Overview

Thomson et al. SIGMOD'12



Calvin Overview

Thomson et al. SIGMOD'12



Is it possible to avoid the two single-threaded pre-execution steps and improve parallelism during execution?

Key Ideas in Q-Store

- ✓ Combine sequencing and scheduling into a single step
- ✓ Unified queue-oriented processing paradigm
- ✓ Global execution priority invariant
- ✓ Support speculative and conservative executions of queues
- ✓ Support multiple isolation levels

Key Ideas in Q-Store

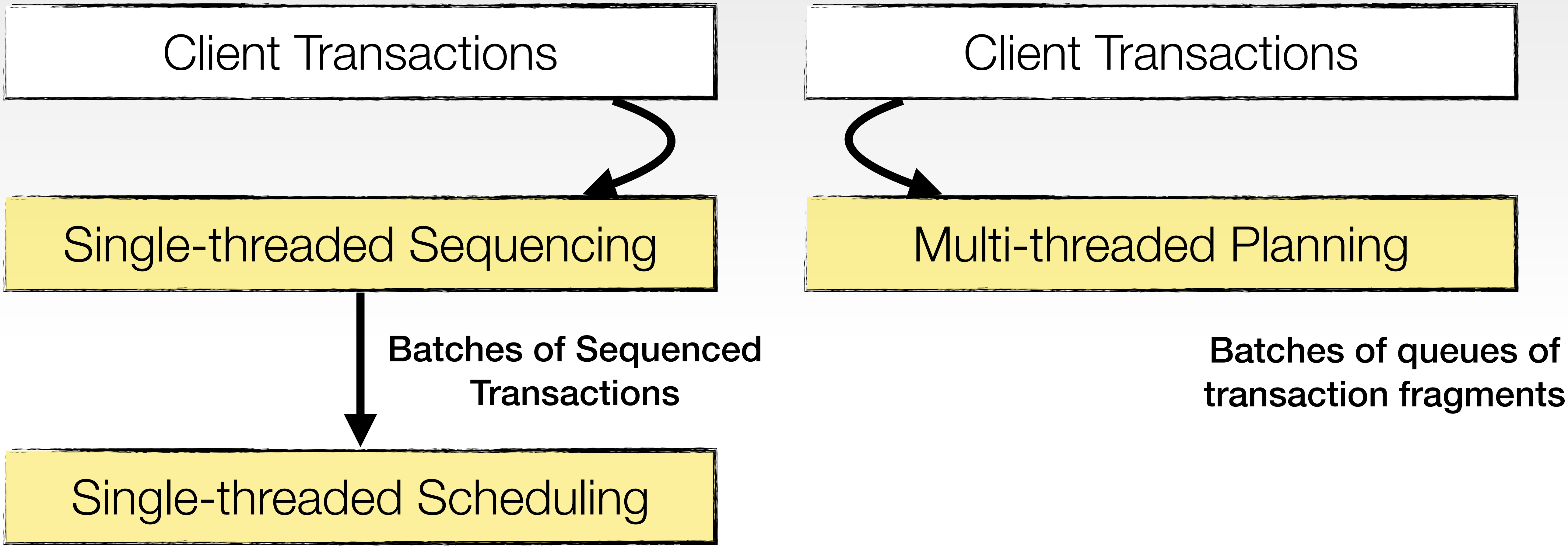
- ✓ Combine sequencing and scheduling into a single step
- ✓ Unified queue-oriented processing paradigm
- ✓ Global execution priority invariant
- ✓ Support speculative and conservative executions of queues
- ✓ Support multiple isolation levels

Calvin Vs. Q-Store

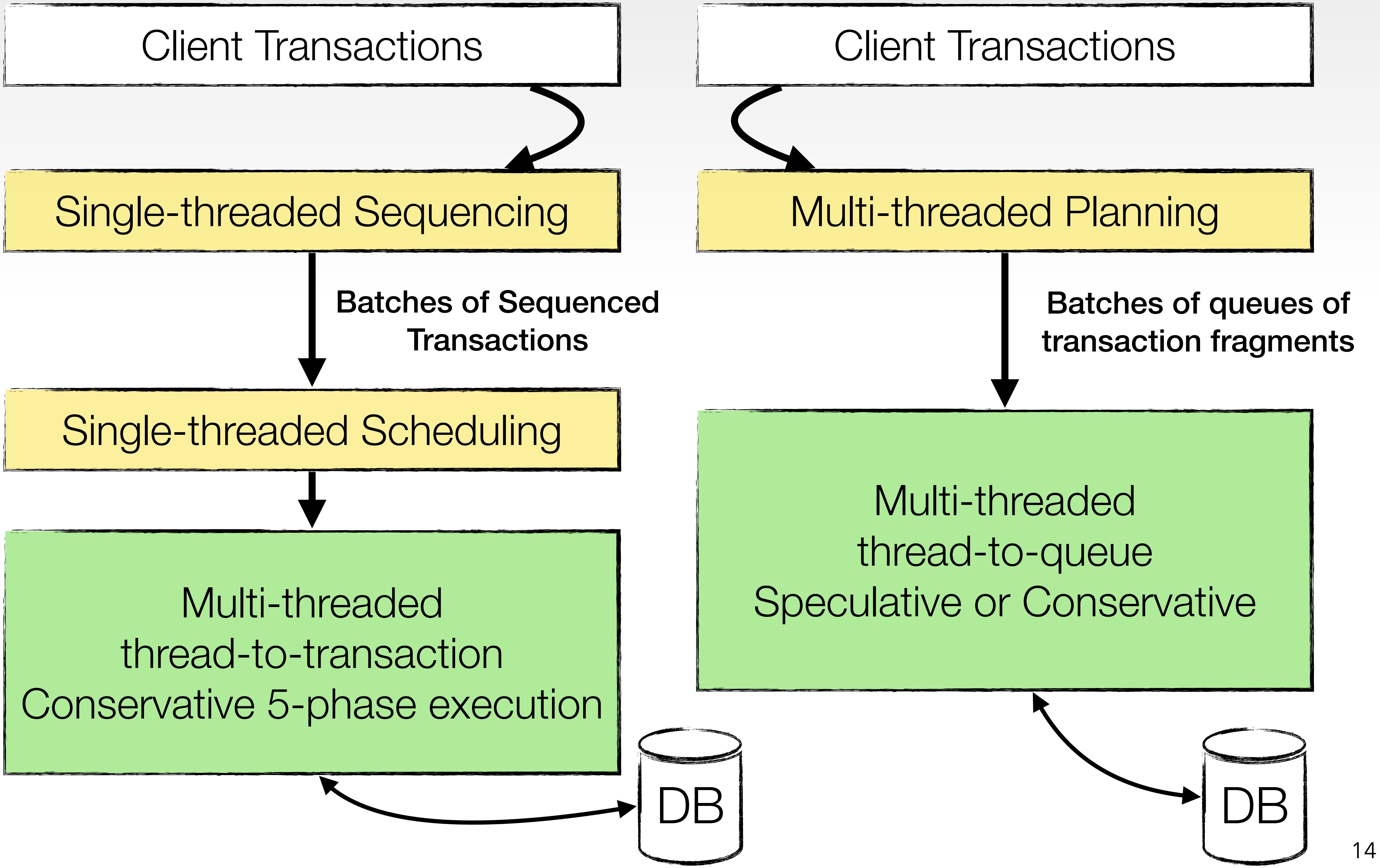
Client Transactions

Client Transactions

Calvin Vs. Q-Store



Calvin Vs. Q-Store



Processing Transactions in Q-Store

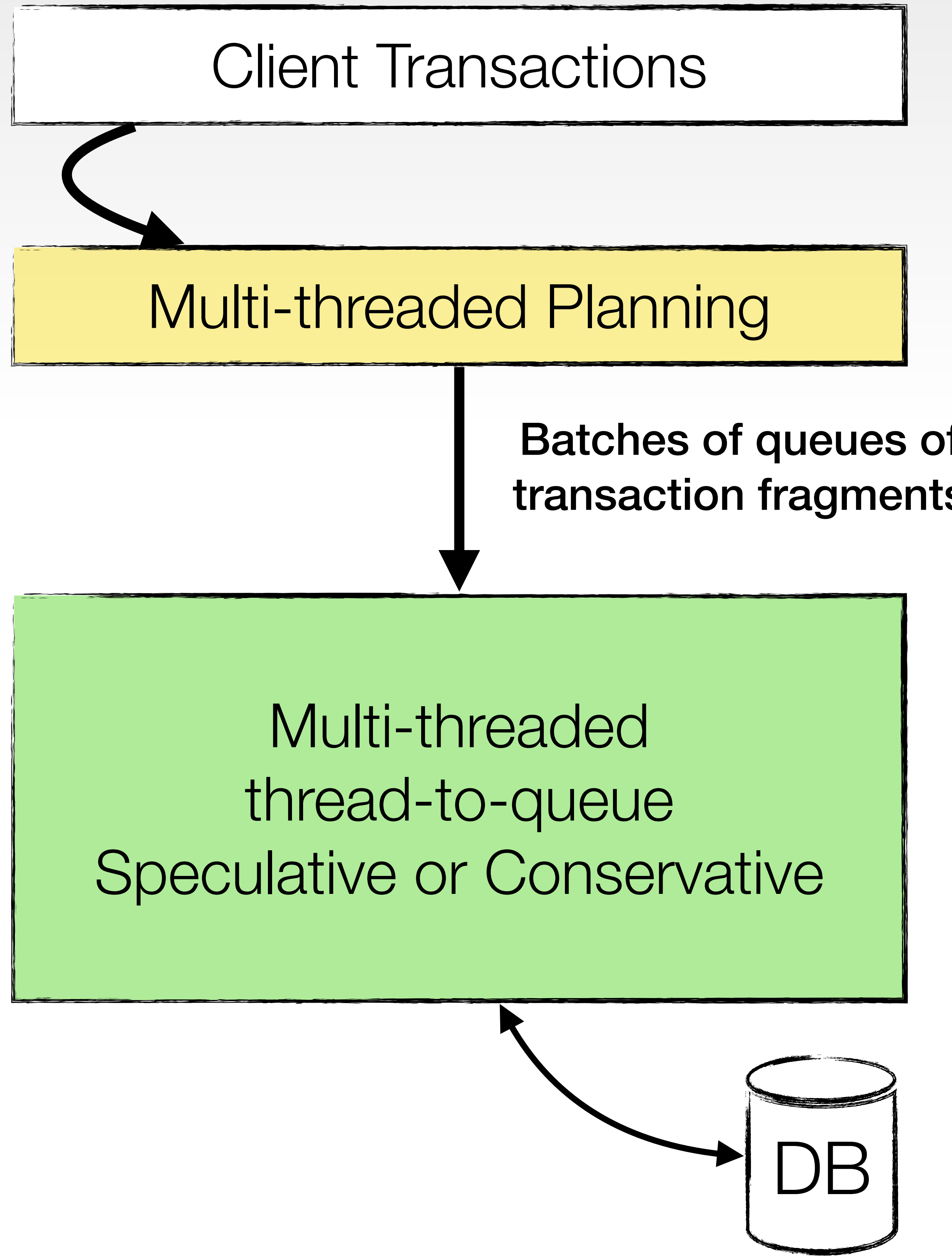
Planning Phase

1. Breakdown transactions into fragments
2. Create prioritized execution-queues of transaction fragments
3. Enforce a strict serial order of conflicting fragments within an execution-queue

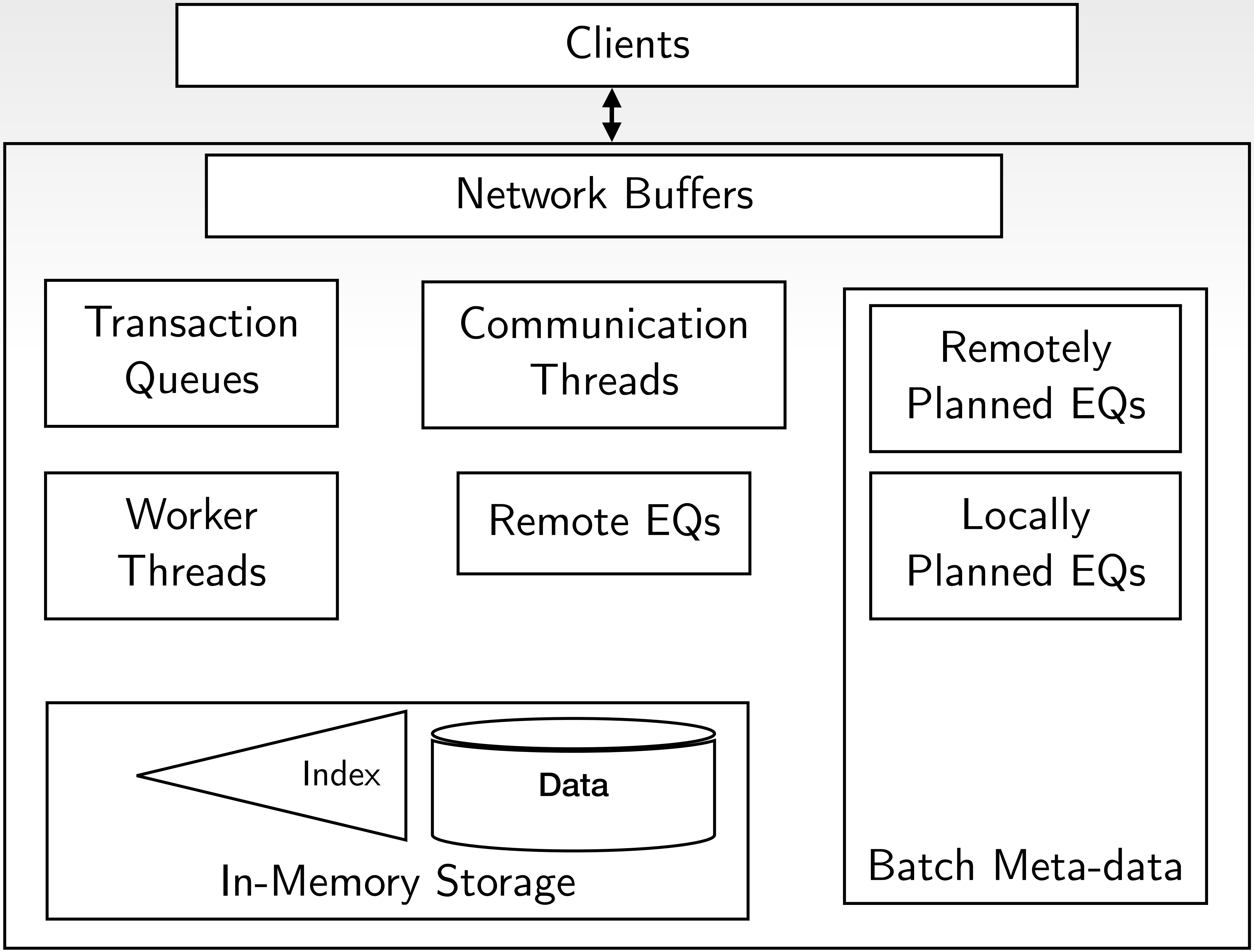
Execution Phase

Process queues while maintaining the **global execution priority invariant**:

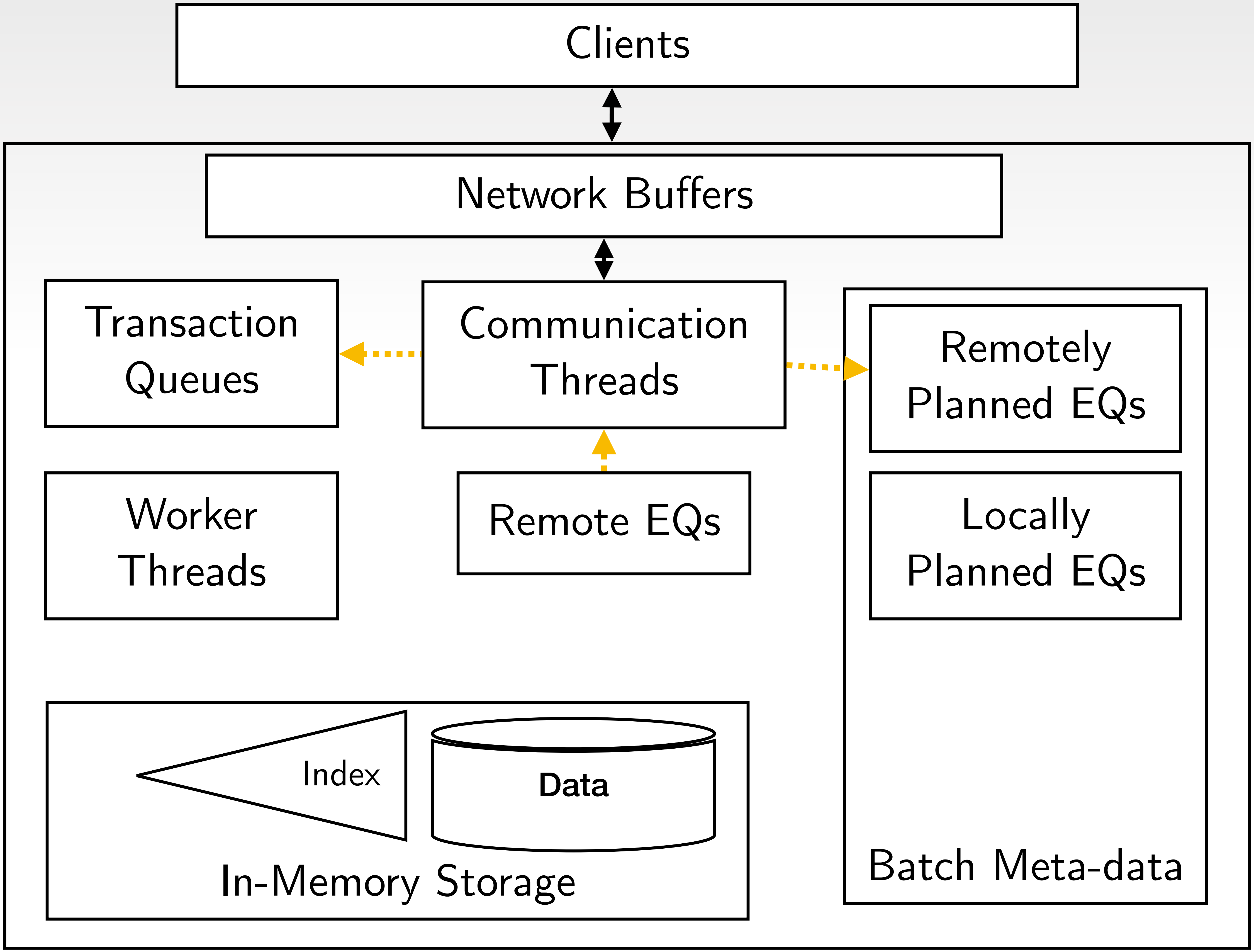
Operations belonging to higher priority execution-queues must always be executed before executing any conflicting lower priority operations.



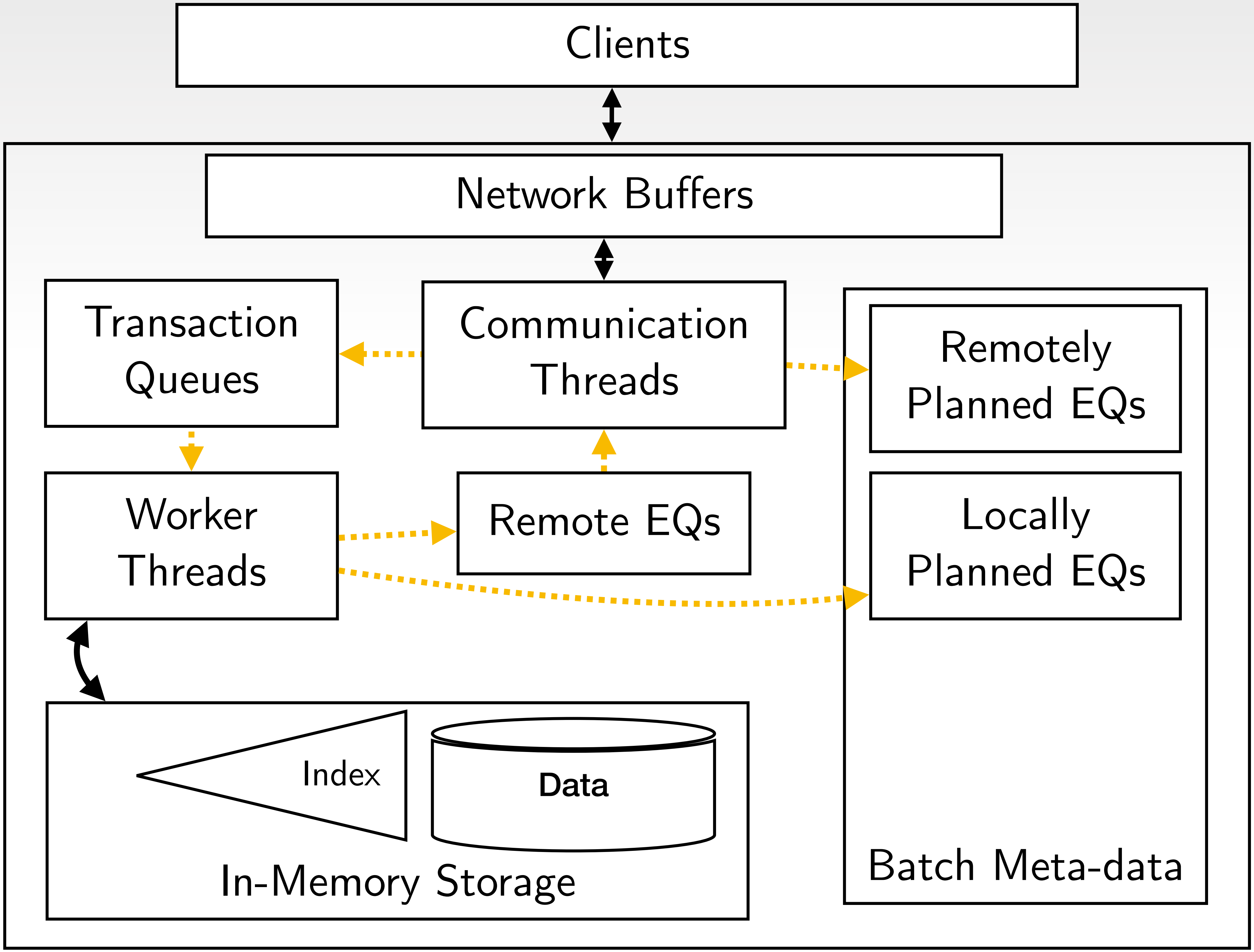
Unified Queue-Oriented Transaction Processing



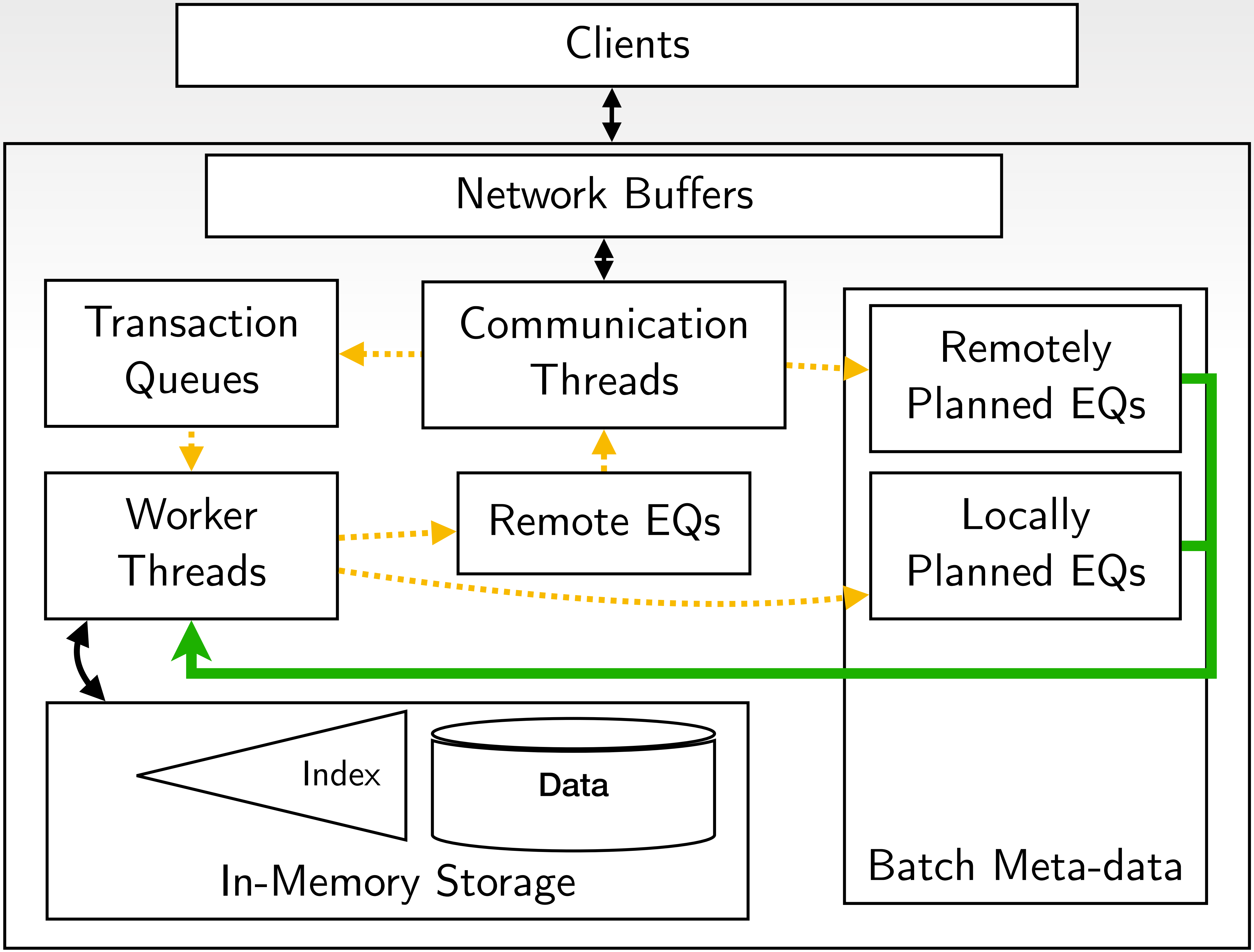
Unified Queue-Oriented Transaction Processing

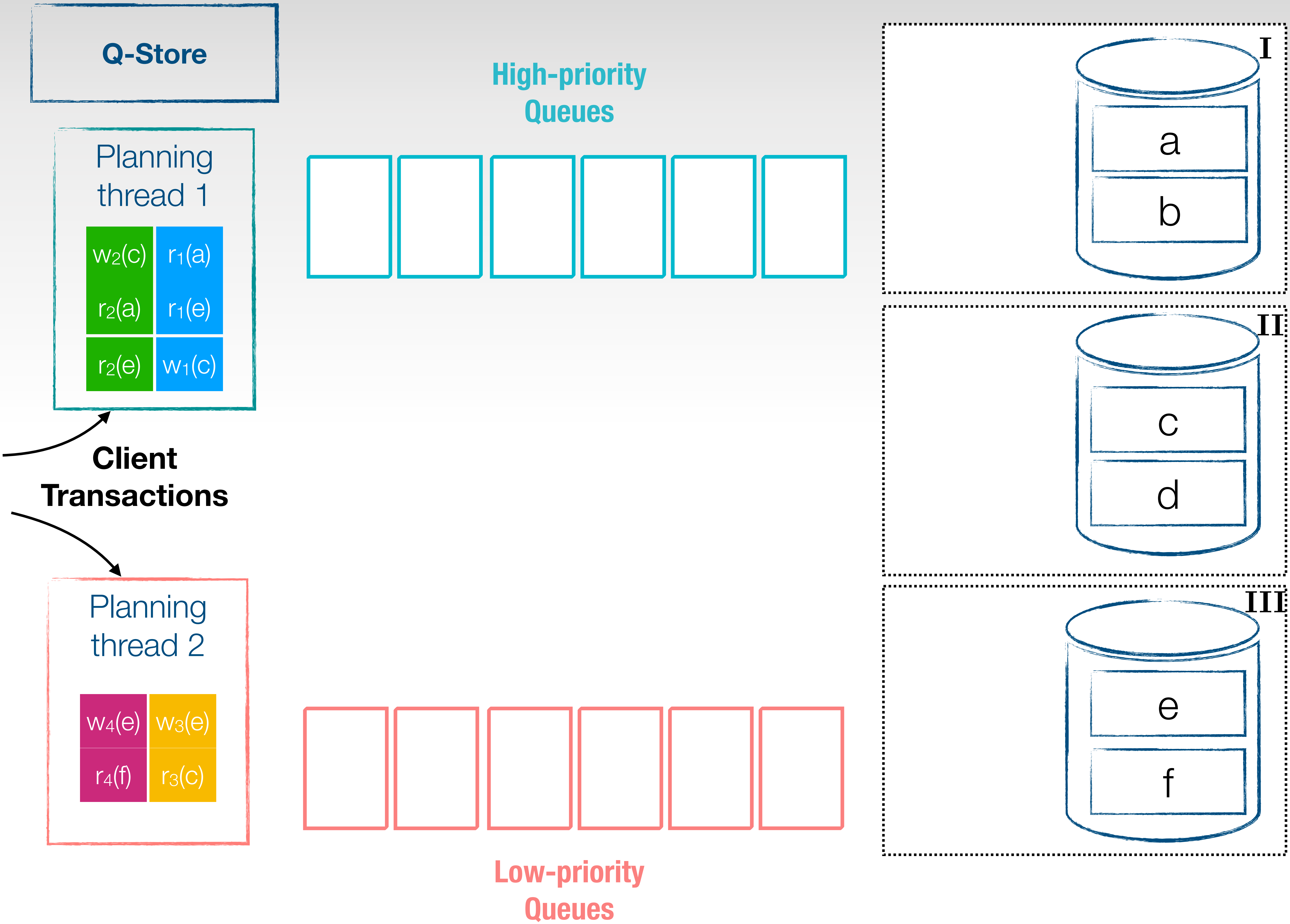


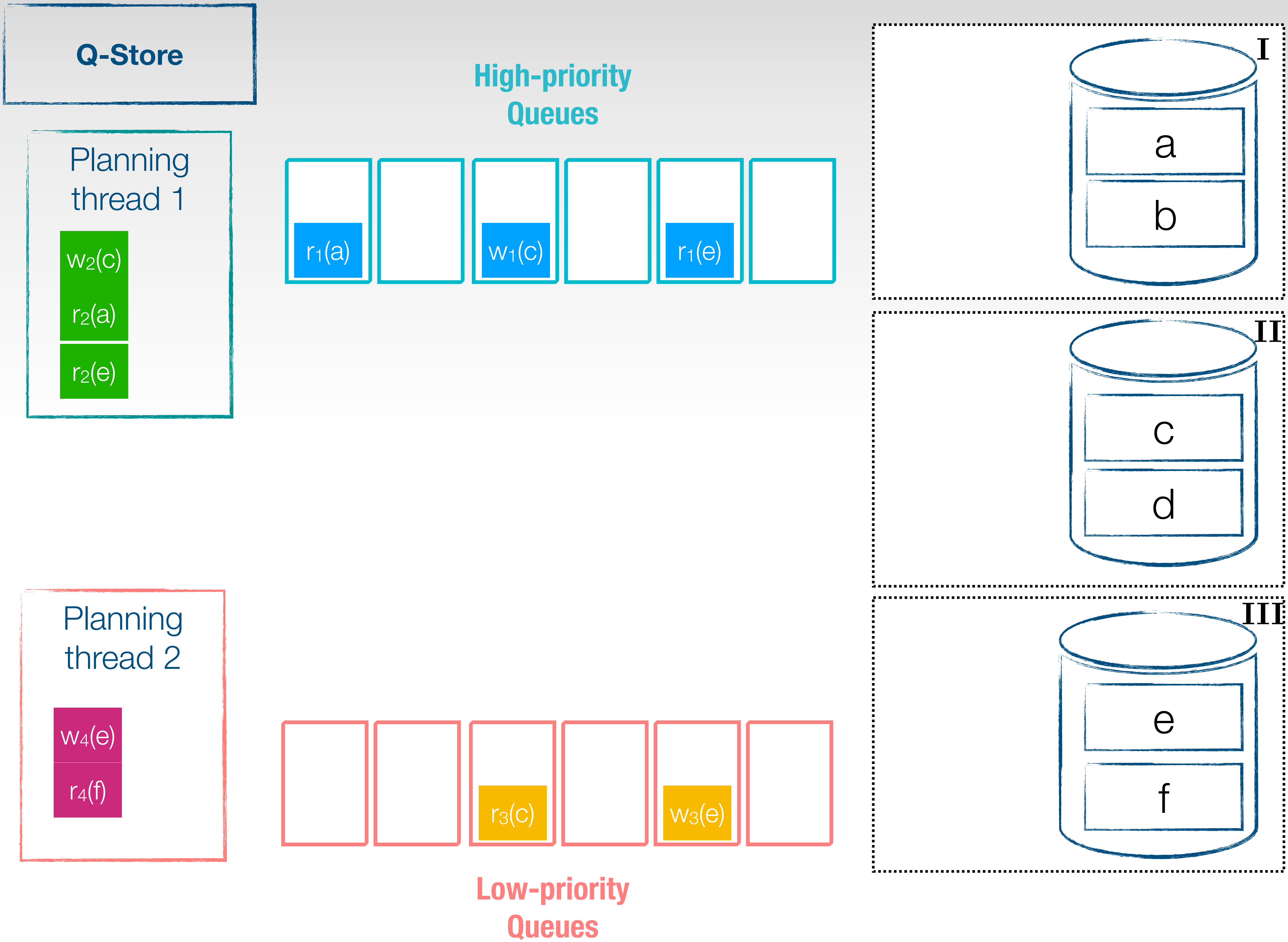
Unified Queue-Oriented Transaction Processing



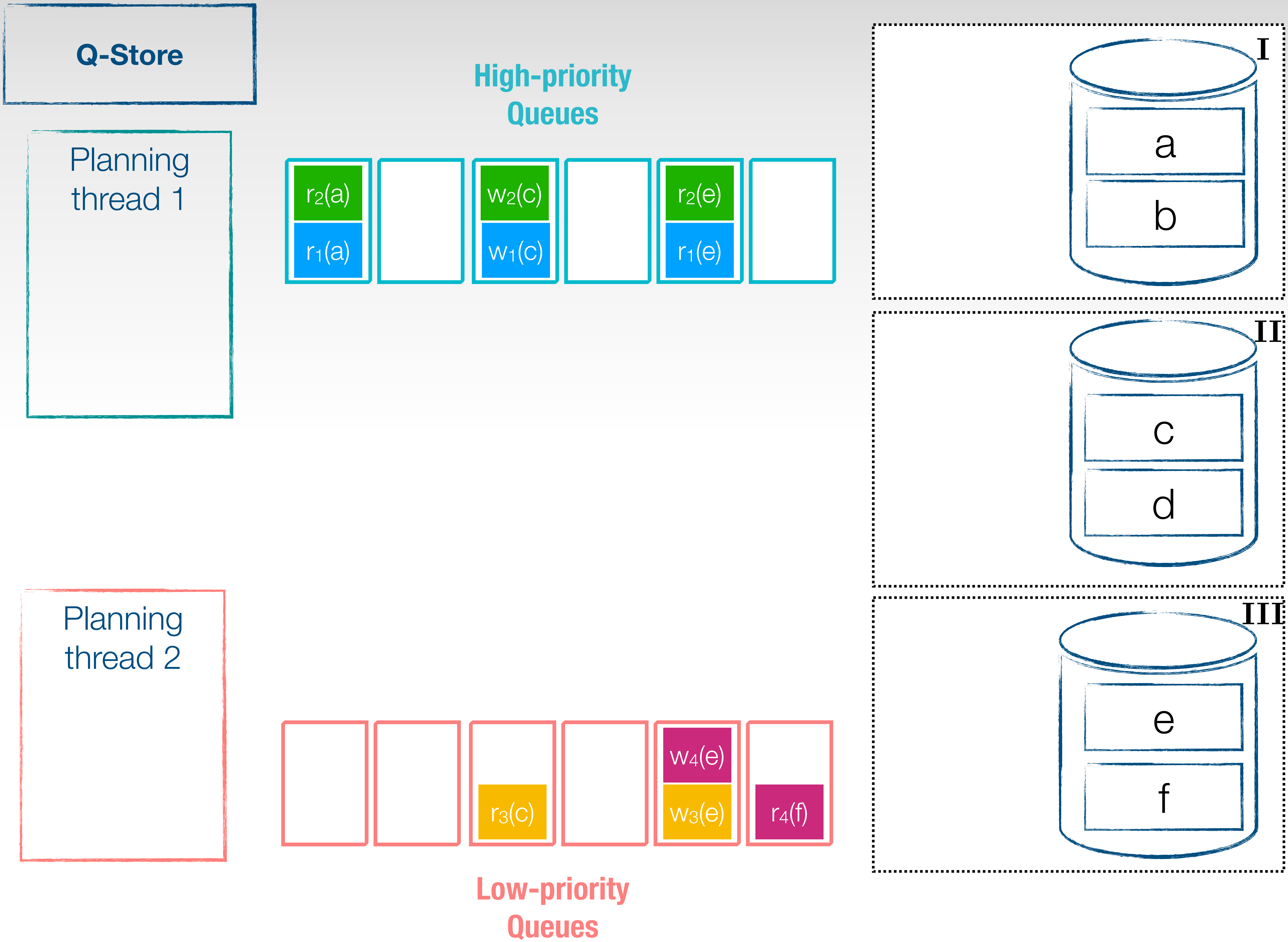
Unified Queue-Oriented Transaction Processing



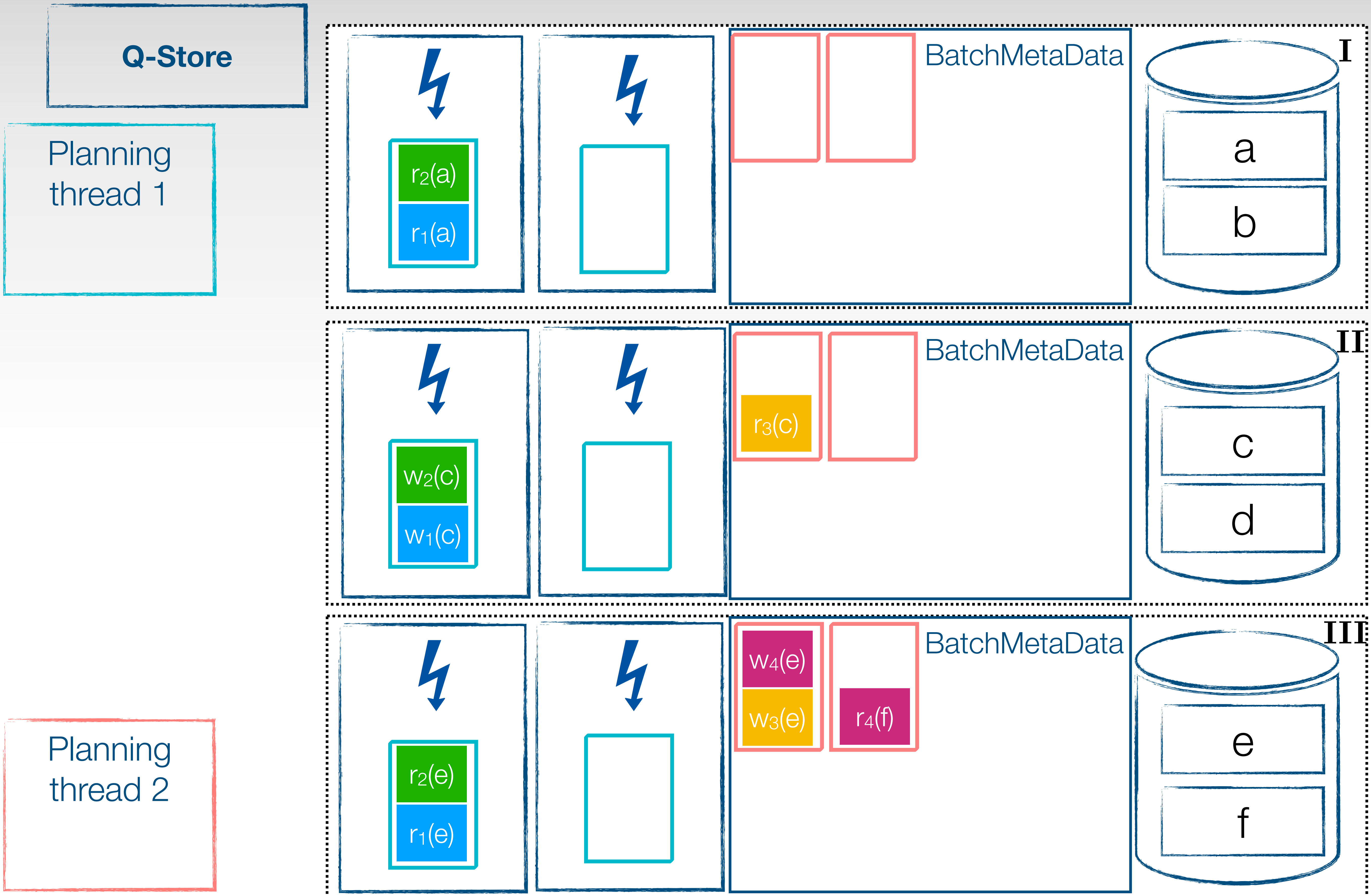

Q-Store
High-priority Queues
Planning thread 1
$w_2(c)$
$r_2(a)$
$r_2(e)$
$r_1(a)$
$r_1(e)$
$w_1(c)$
Client Transactions
Planning thread 2
$w_4(e)$
$r_4(f)$
$w_3(e)$
$r_3(c)$
Low-priority Queues
I
a
b
II
c
d
III
e
f

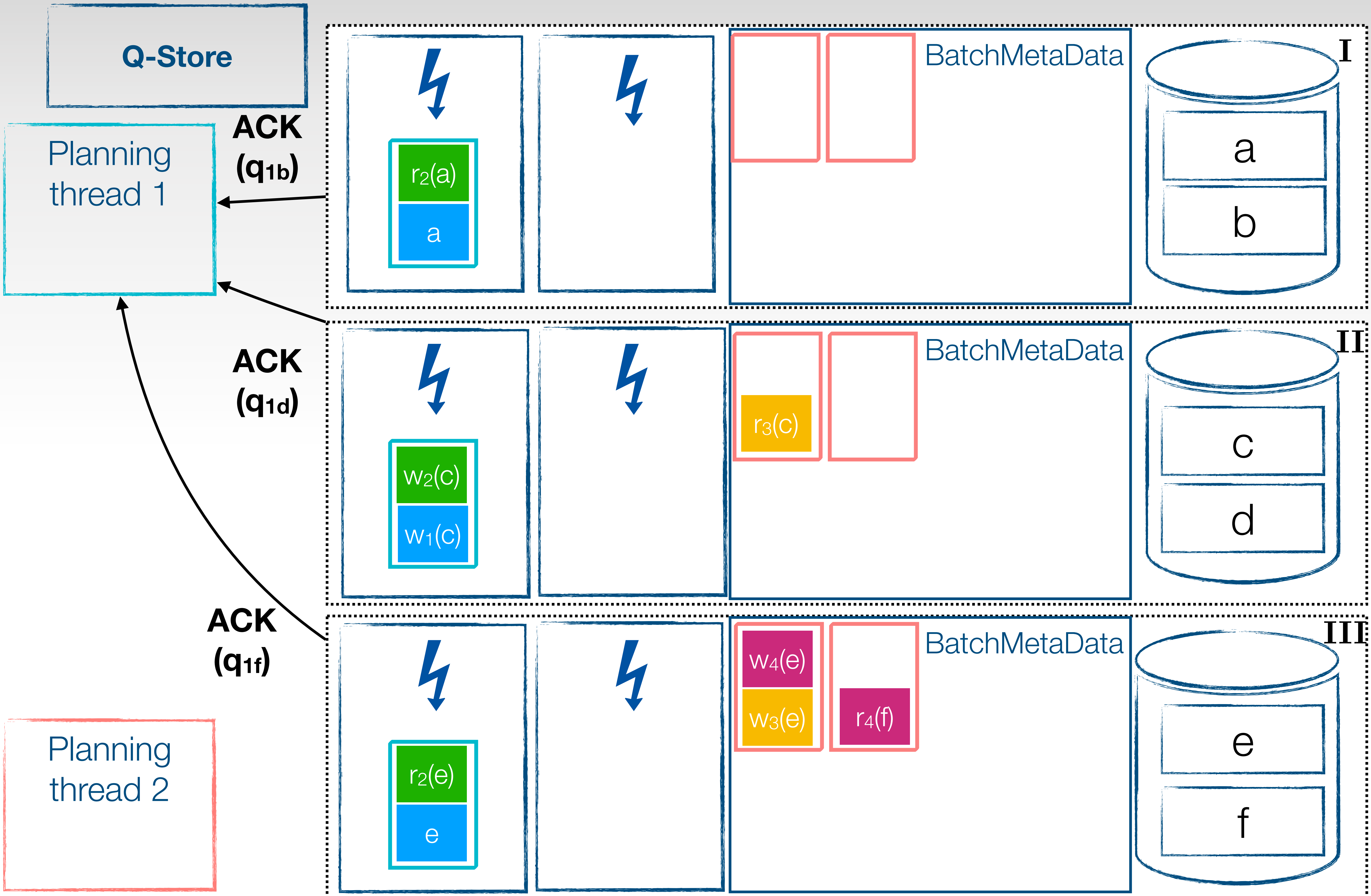


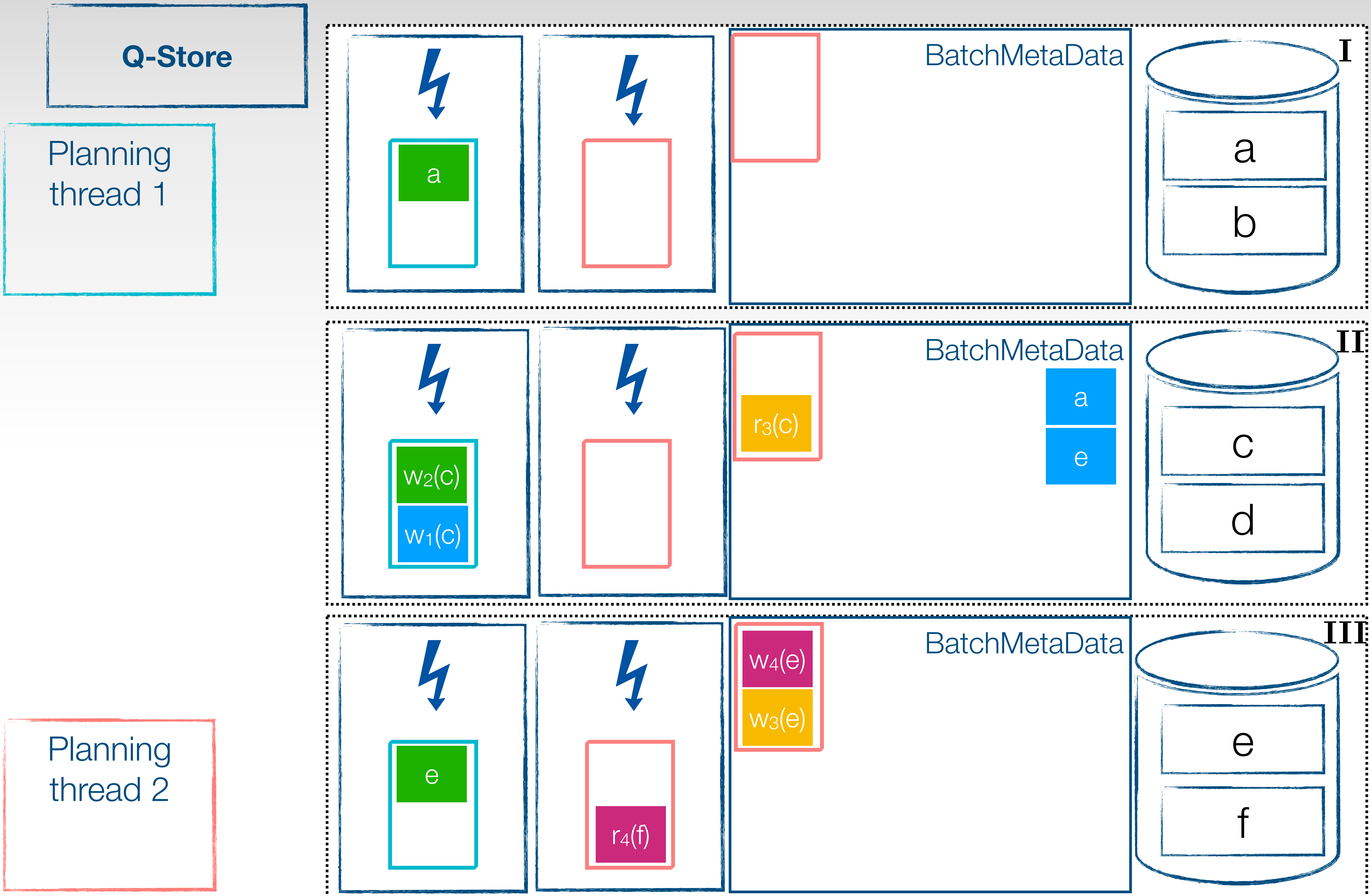
Q-Store
Planning thread 1
$w_2(c)$
$r_2(a)$
$r_2(e)$
High-priority Queues
$r_1(a)$
$w_1(c)$
$r_1(e)$
I
a
b
II
c
d
III
e
f
Planning thread 2
$w_4(e)$
$r_4(f)$
$r_3(c)$
$w_3(e)$
Low-priority Queues


Q-Store
High-priority Queues
Planning thread 1
$r_2(a)$
$r_1(a)$
$w_2(c)$
$w_1(c)$
$r_2(e)$
$r_1(e)$
I
a
b
II
c
d
Planning thread 2
$r_3(c)$
$w_4(e)$
$w_3(e)$
$r_4(f)$
III
e
f
Low-priority Queues

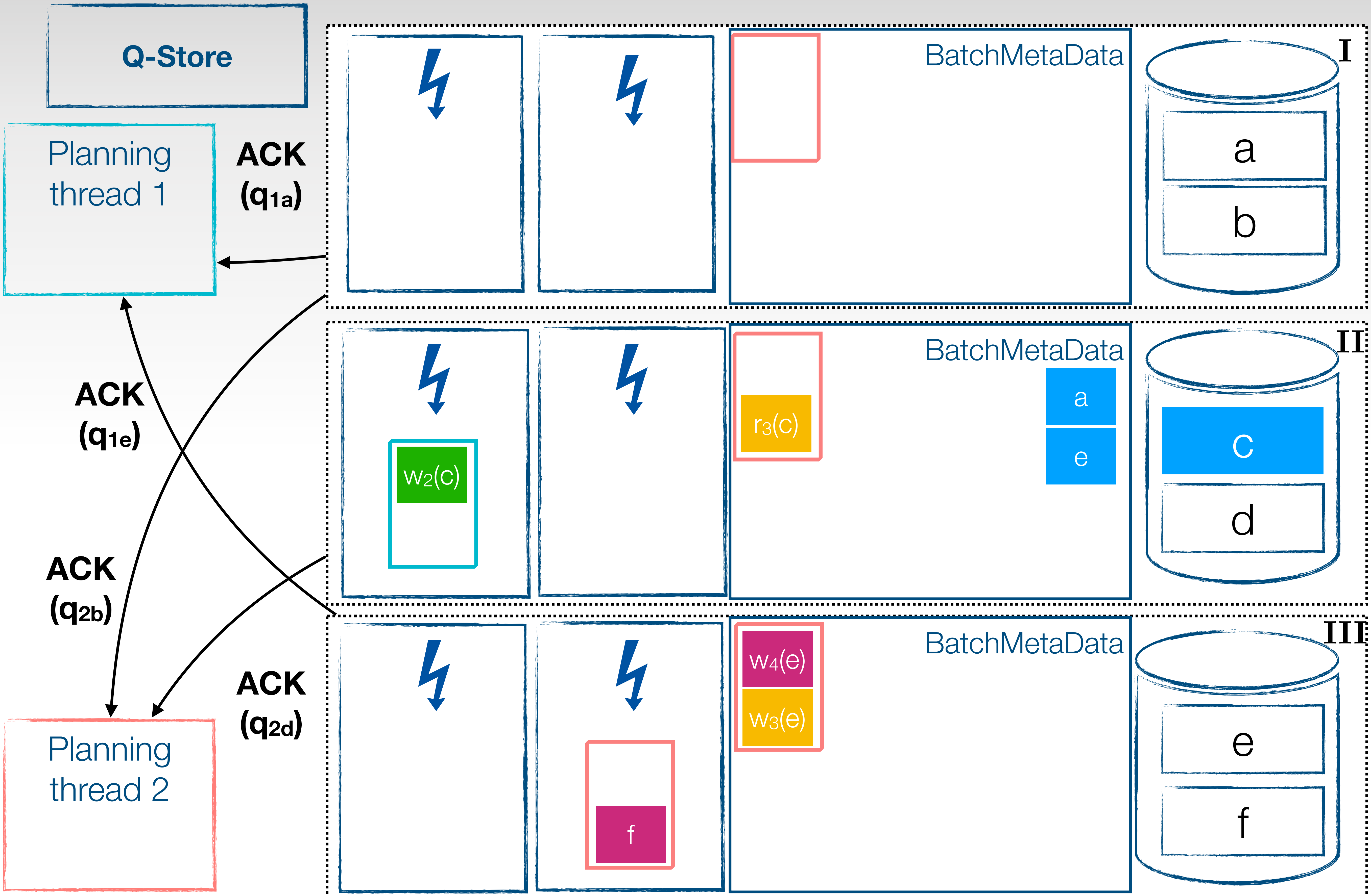

Q-Store
Planning thread 1
Planning thread 2
BatchMetaData
r2(a)
r1(a)
a
b
I
BatchMetaData
r3(c)
w2(c)
w1(c)
c
d
II
BatchMetaData
w4(e)
w3(e)
r4(f)
r2(e)
r1(e)
e
f
III

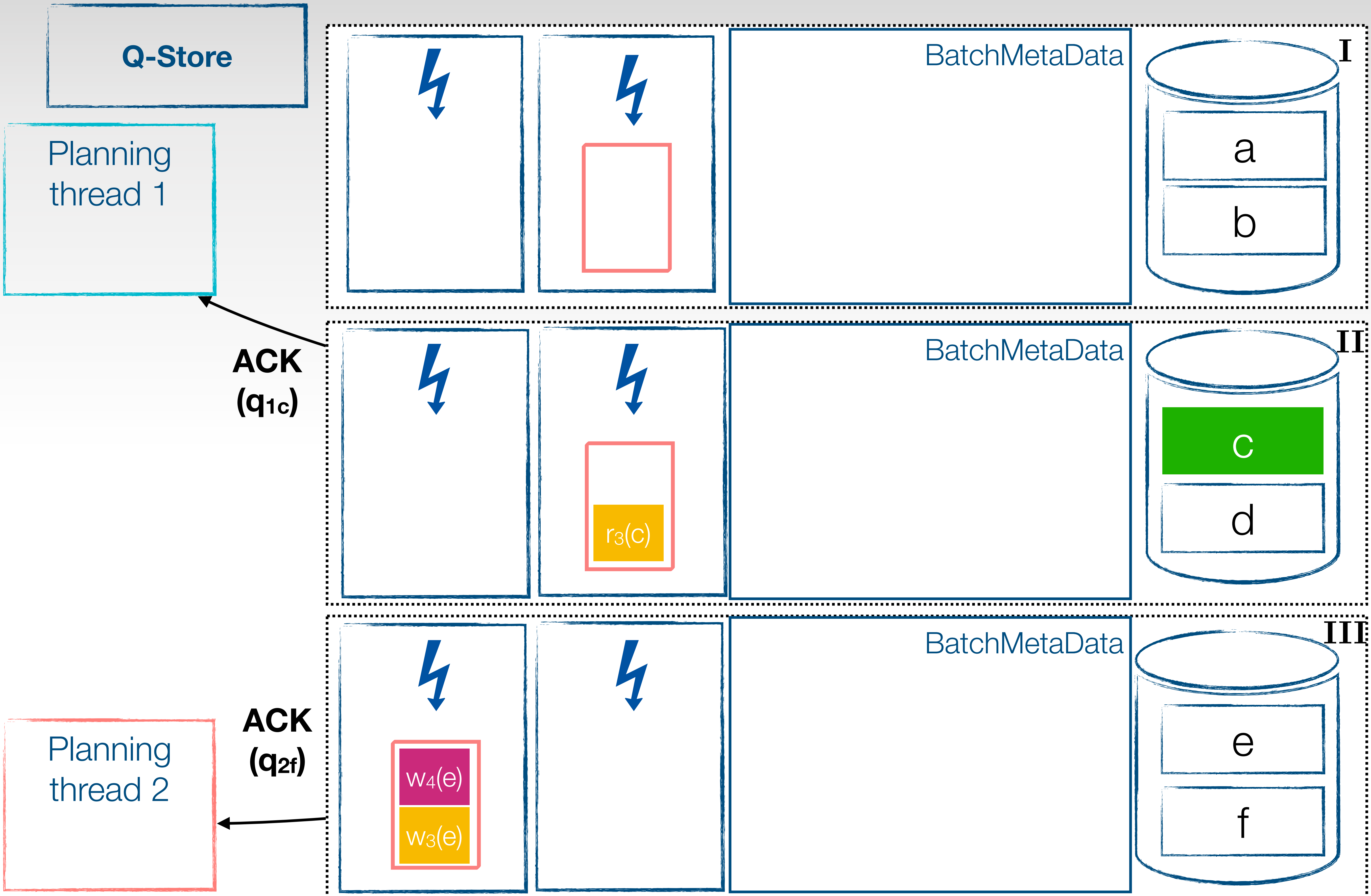

Q-Store
Planning thread 1
Planning thread 2
I
II
III
BatchMetaData
BatchMetaData
BatchMetaData
$r_2(a)$
$r_1(a)$
$w_2(c)$
$w_1(c)$
$r_2(e)$
$r_1(e)$
$r_3(c)$
$w_4(e)$
$w_3(e)$
$r_4(f)$
a
b
c
d
e
f

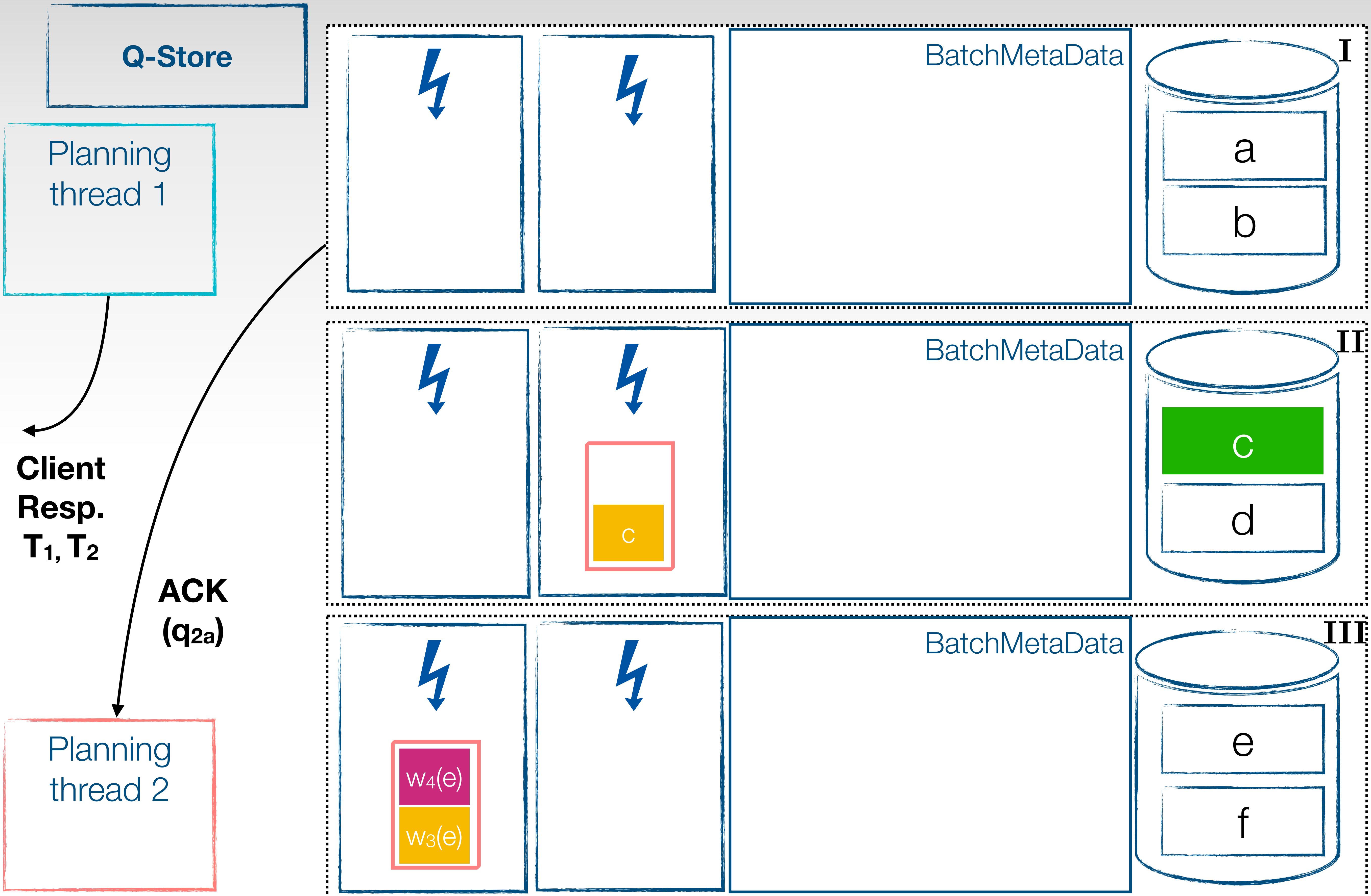

Q-Store
Planning thread 1
ACK (q_{1b})
ACK (q_{1d})
ACK (q_{1f})
Planning thread 2
$r_2(a)$
a
BatchMetaData
I
a
b
$w_2(c)$
$w_1(c)$
$r_3(c)$
BatchMetaData
II
c
d
$r_2(e)$
e
$w_4(e)$
$w_3(e)$
$r_4(f)$
BatchMetaData
III
e
f

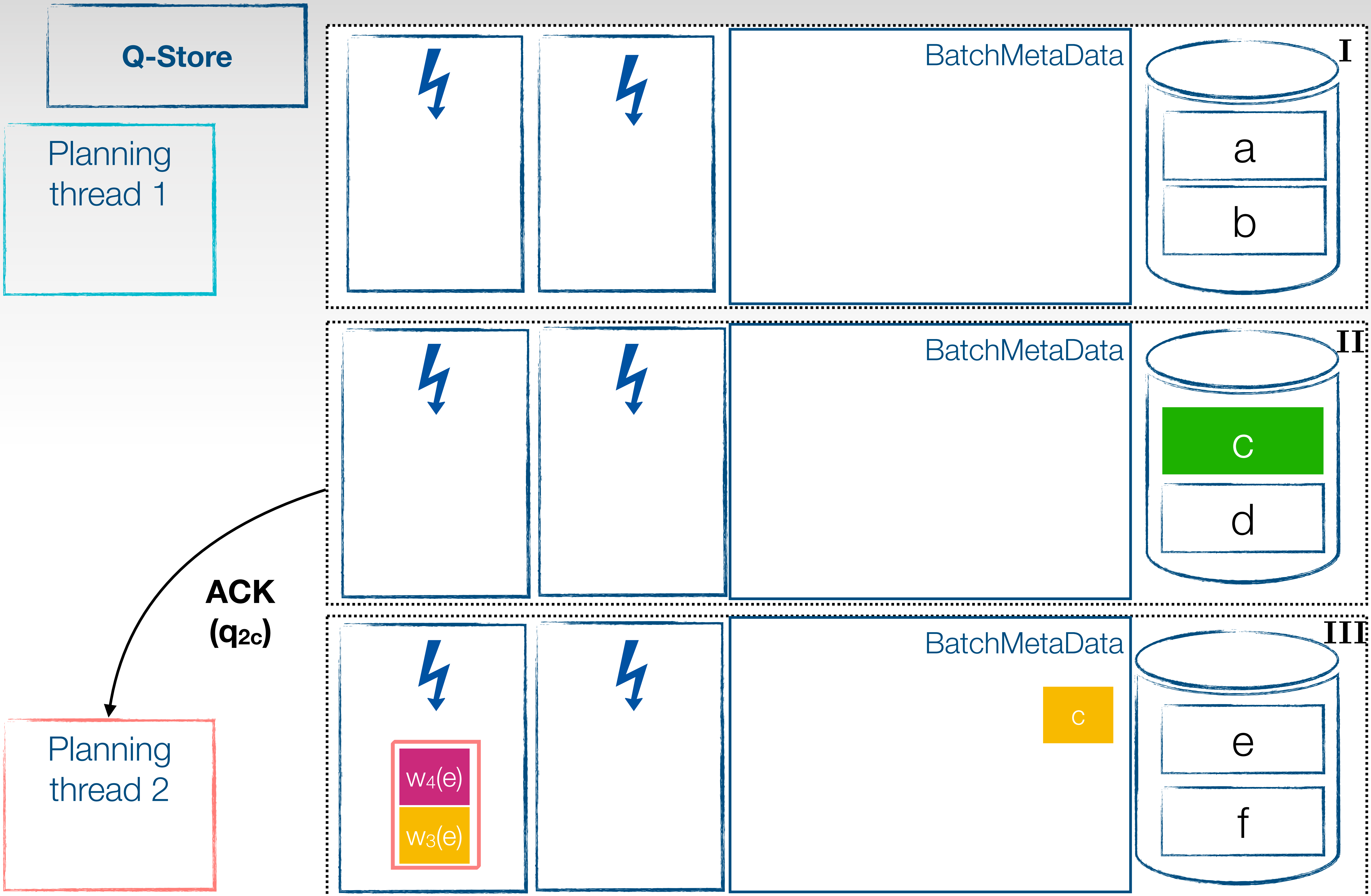


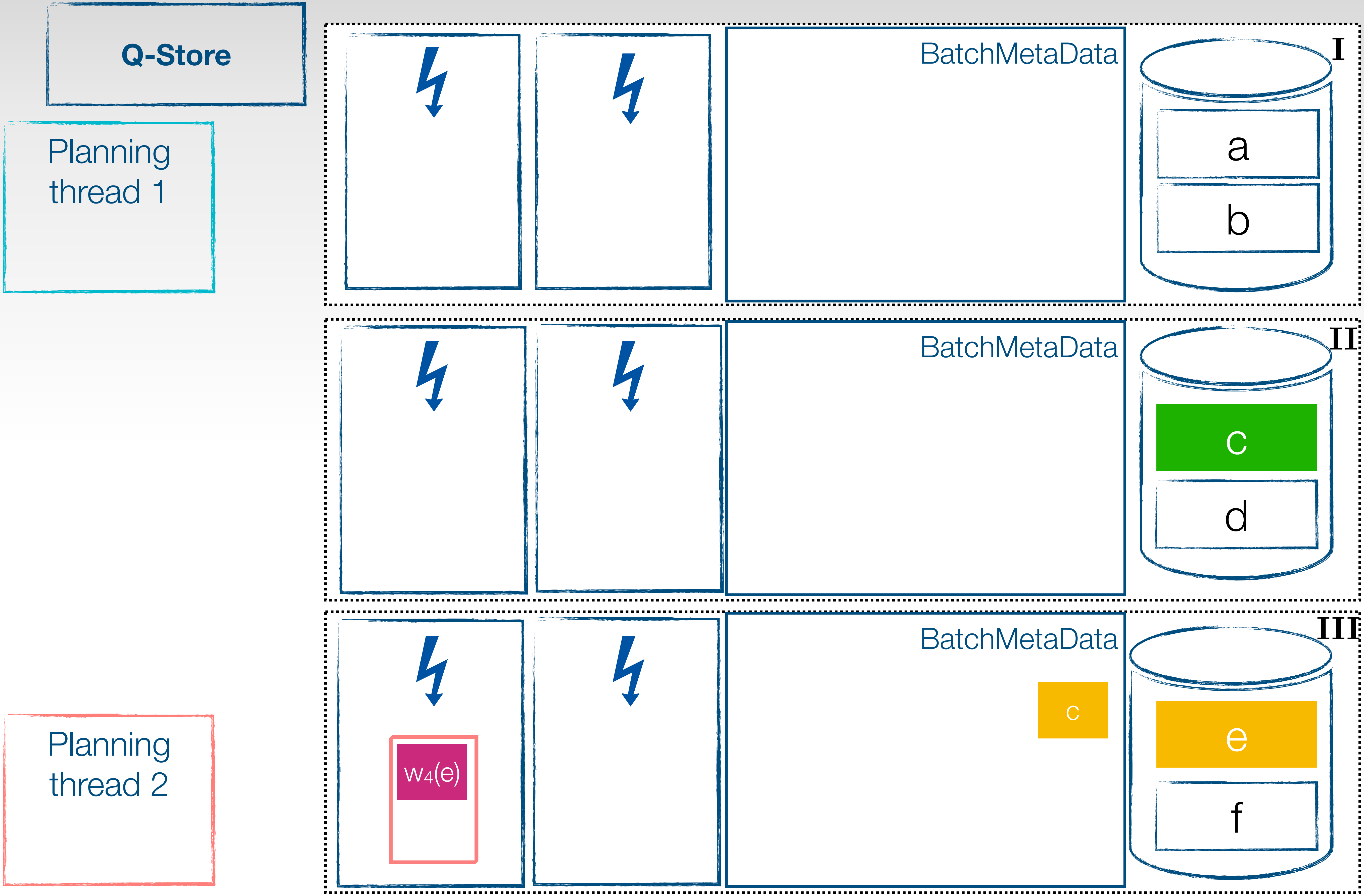
Q-Store
Planning thread 1
Planning thread 2
I
a
BatchMetaData
a
b
II
$w_2(c)$
$w_1(c)$
$r_3(c)$
BatchMetaData
a
e
c
d
III
e
$r_4(f)$
$w_4(e)$
$w_3(e)$
BatchMetaData
e
f

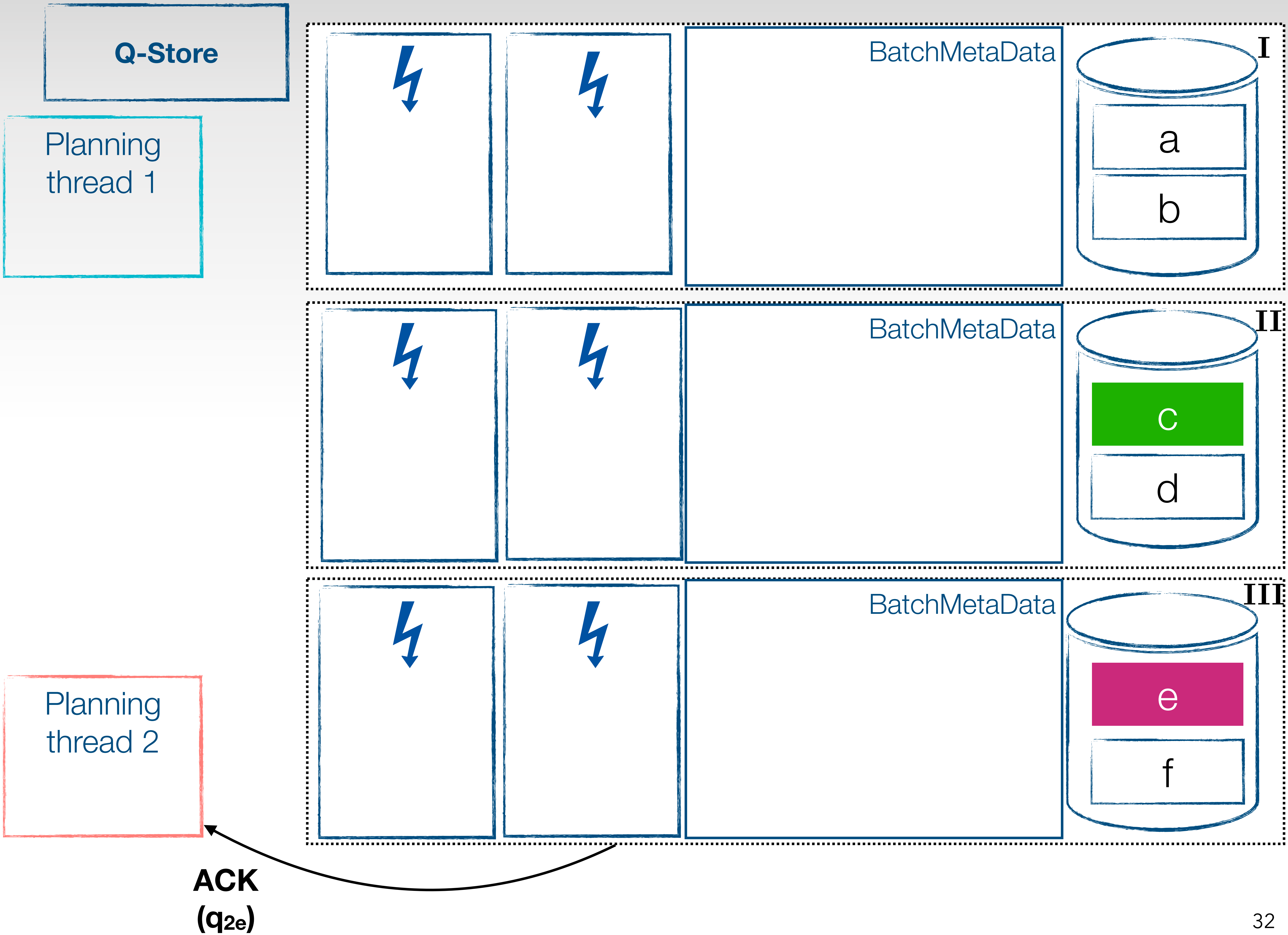

Q-Store
Planning thread 1
ACK (q1a)
BatchMetaData
I
a
b
ACK (q1e)
BatchMetaData
II
a
e
r3(c)
w2(c)
c
d
ACK (q2b)
ACK (q2d)
BatchMetaData
III
w4(e)
w3(e)
f
e
f
Planning thread 2

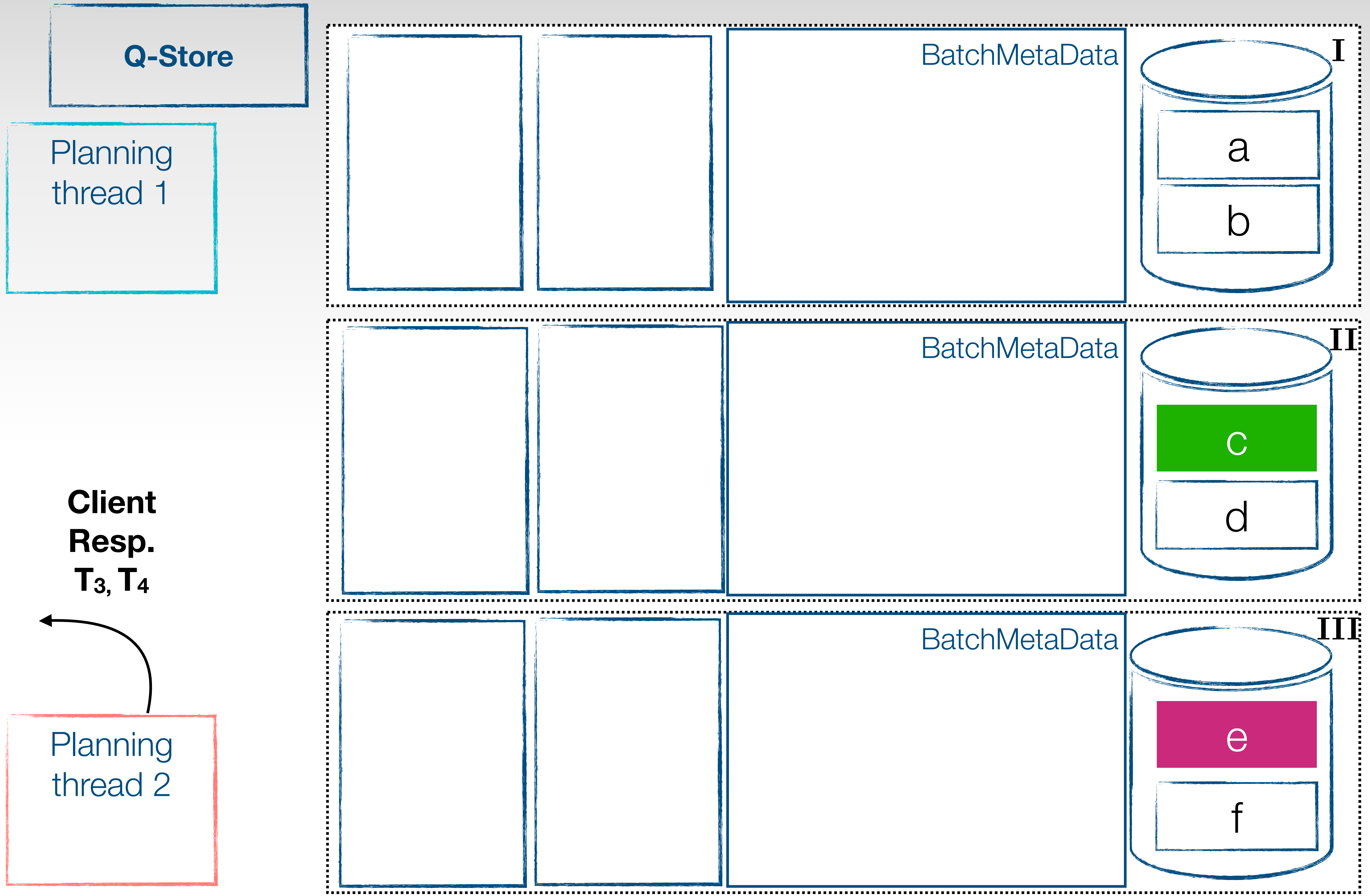

Q-Store
Planning thread 1
Planning thread 2
ACK (q_{1c})
ACK (q_{2f})
BatchMetaData
BatchMetaData
BatchMetaData
I
II
III
a
b
c
d
e
f
$r_3(c)$
$w_4(e)$
$w_3(e)$


Q-Store
Planning thread 1
Client Resp. T_1, T_2
ACK (q_{2a})
Planning thread 2
BatchMetaData
I
a
b
BatchMetaData
II
c
c
d
BatchMetaData
III
$w_4(e)$
$w_3(e)$
e
f


Q-Store
Planning thread 1
BatchMetaData
I
a
b
BatchMetaData
II
c
d
ACK (q2c)
Planning thread 2
BatchMetaData
III
w4(e)
w3(e)
c
e
f


Q-Store
Planning thread 1
Planning thread 2
BatchMetaData
BatchMetaData
BatchMetaData
I
II
III
a
b
c
d
e
f
c
w4(e)


Q-Store
Planning thread 1
BatchMetaData
I
a
b
BatchMetaData
II
c
d
BatchMetaData
III
e
f
Planning thread 2
ACK
(q_{2e})



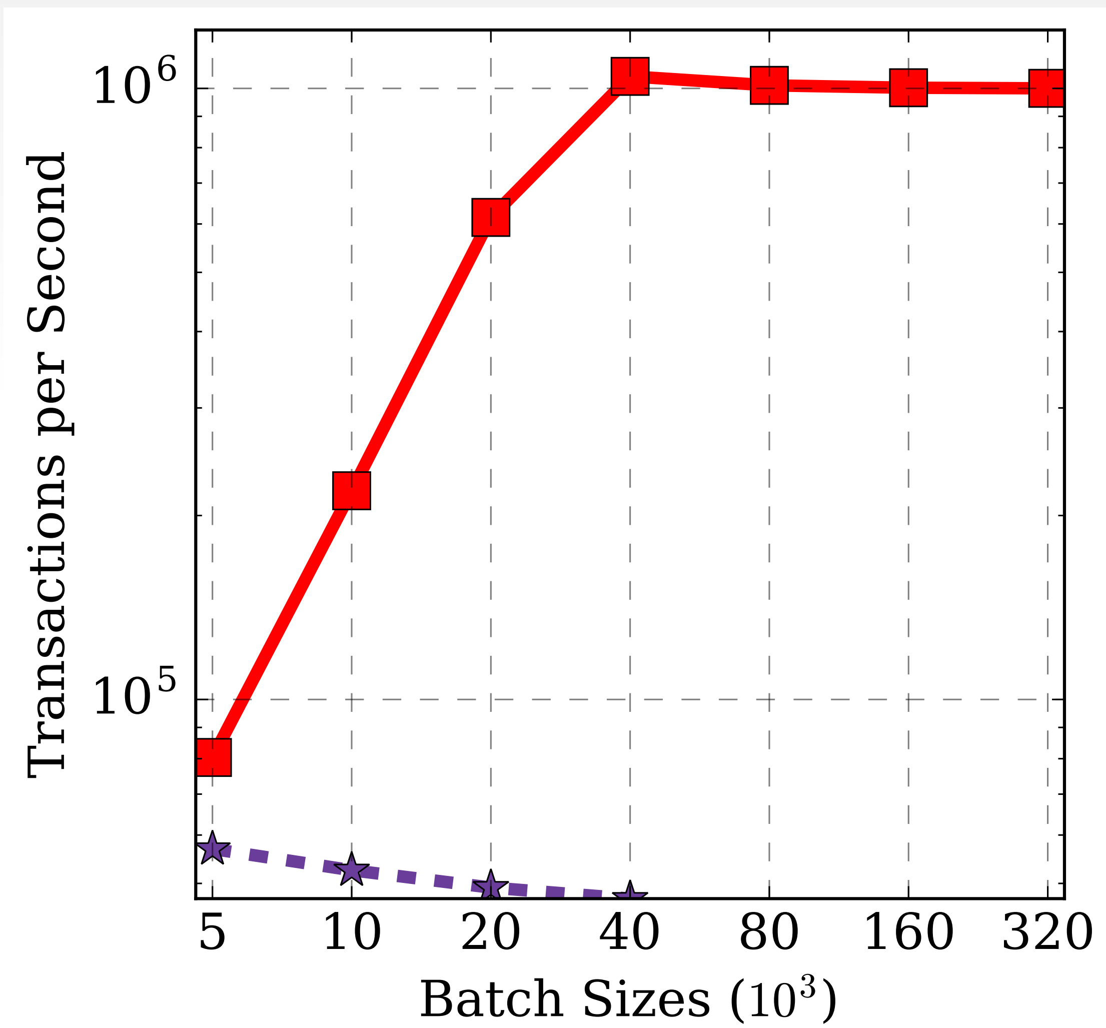
Q-Store
Planning thread 1
Client Resp. T_3, T_4
Planning thread 2
BatchMetaData
I
a
b
BatchMetaData
II
c
d
BatchMetaData
III
e
f

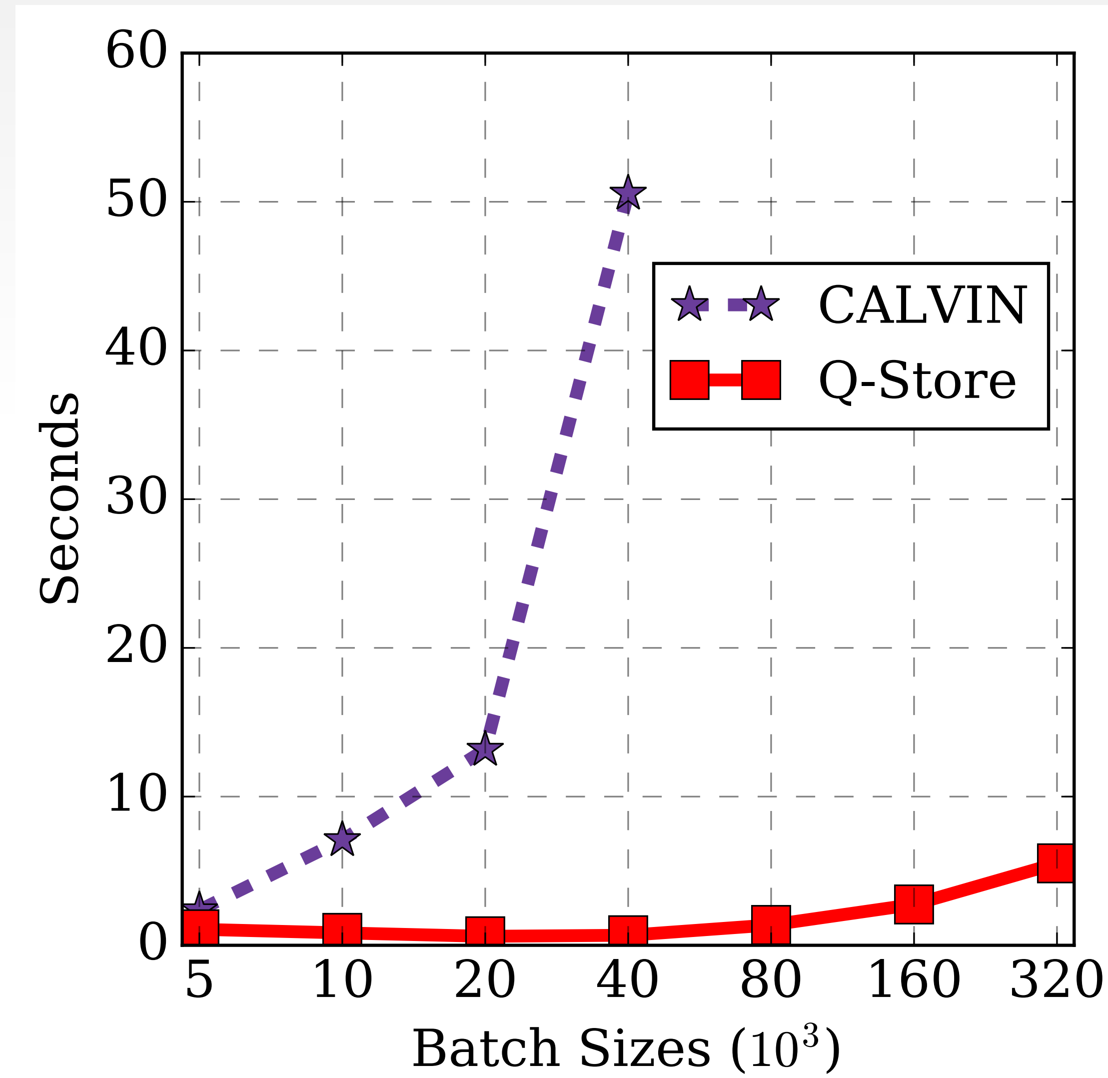
Evaluation Environment

Hardware	32 (16 clients + 16 servers) AWS EC2 c5.2xlarge instances with: CPU: 8 vCPUs RAM: 16GB
Workload	YCSB: 1 table, RMW and Read-only operations, Uniform and Zipfian distribution TPC-C: 9 tables, Payment and NewOrder
Software	Operating System: Ubuntu LTS 16.04.3 Compiler: GCC with –O2 compiler optimizations

Effect of Varying Batch Size

- 8 read and 8 RMW operations per transaction
- 50% multi-partition transactions
- Uniform distribution

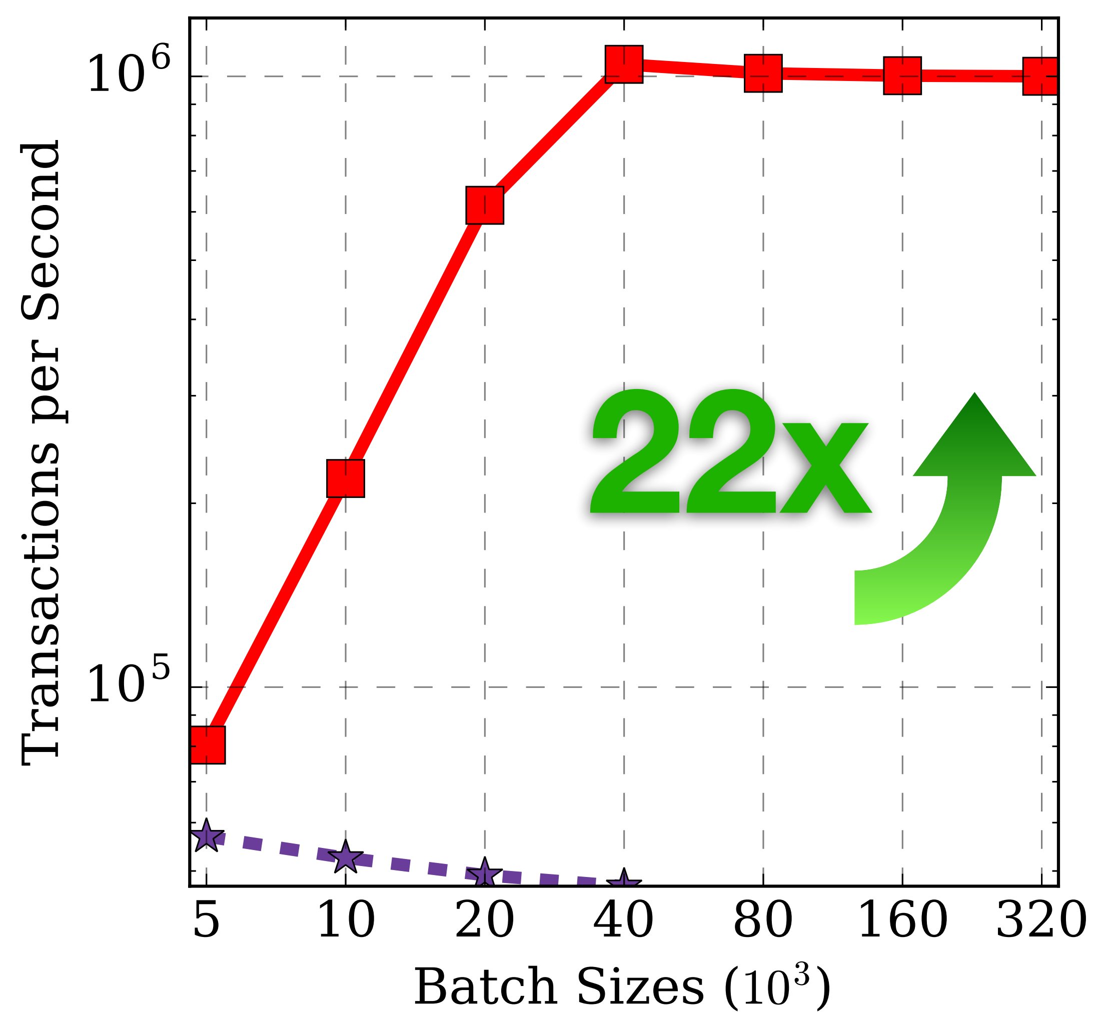


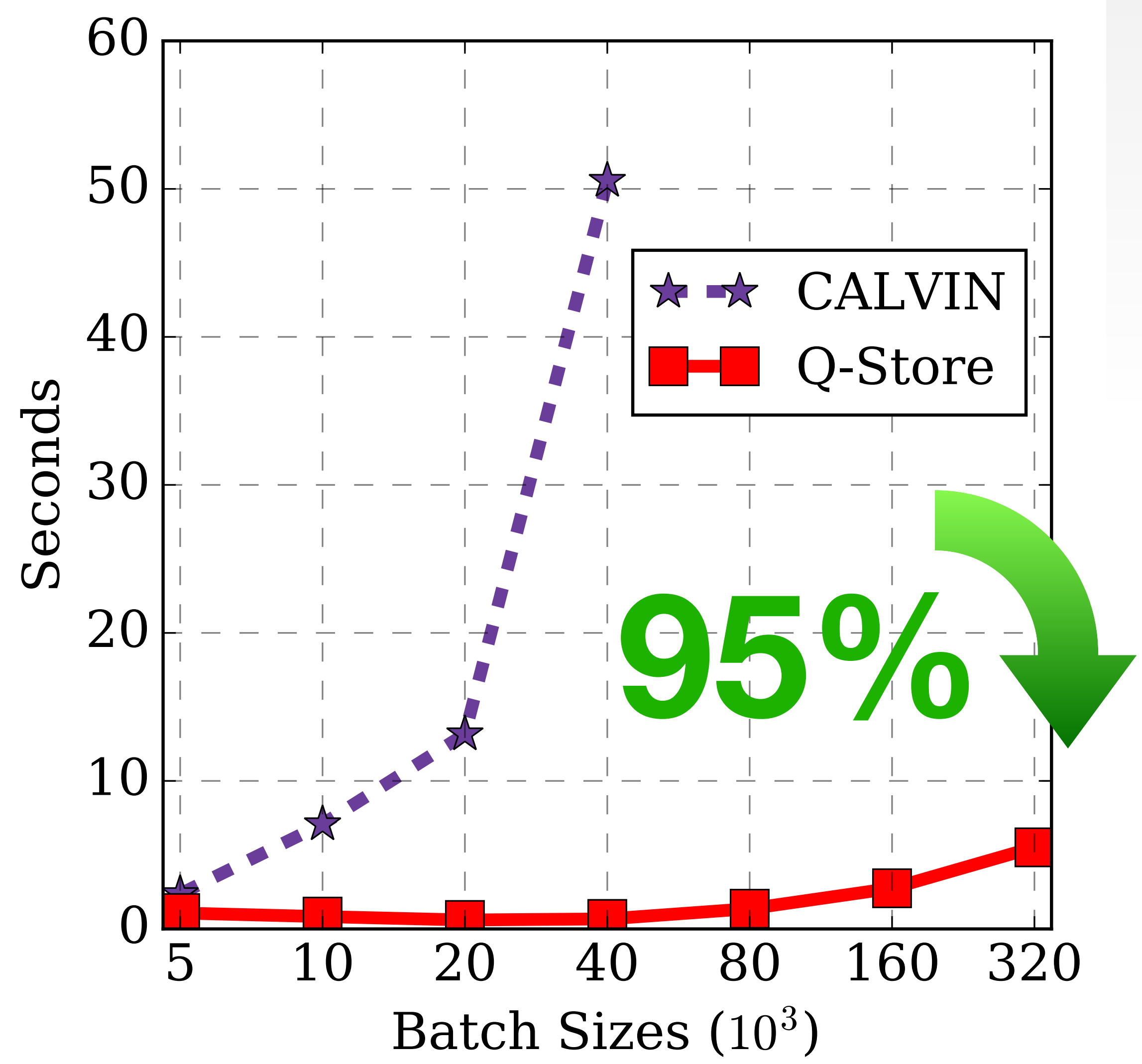


Q-Store eliminates the bottleneck of single-threaded sequencing scheduling and scales well while increasing the batch size

Effect of Varying Batch Size

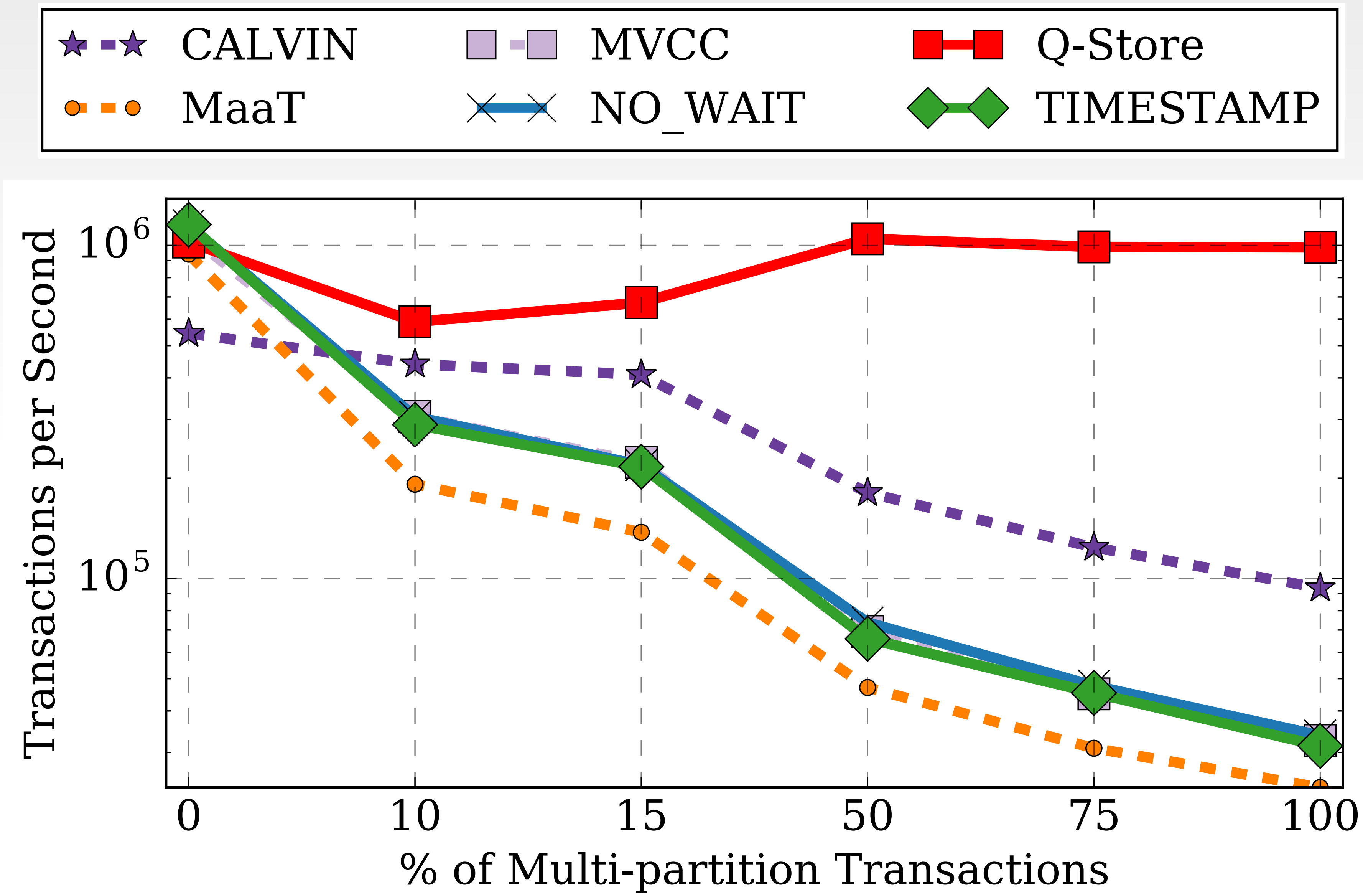
- **8 read and 8 RMW operations per transaction**
- **50% multi-partition transactions**
- **Uniform distribution**





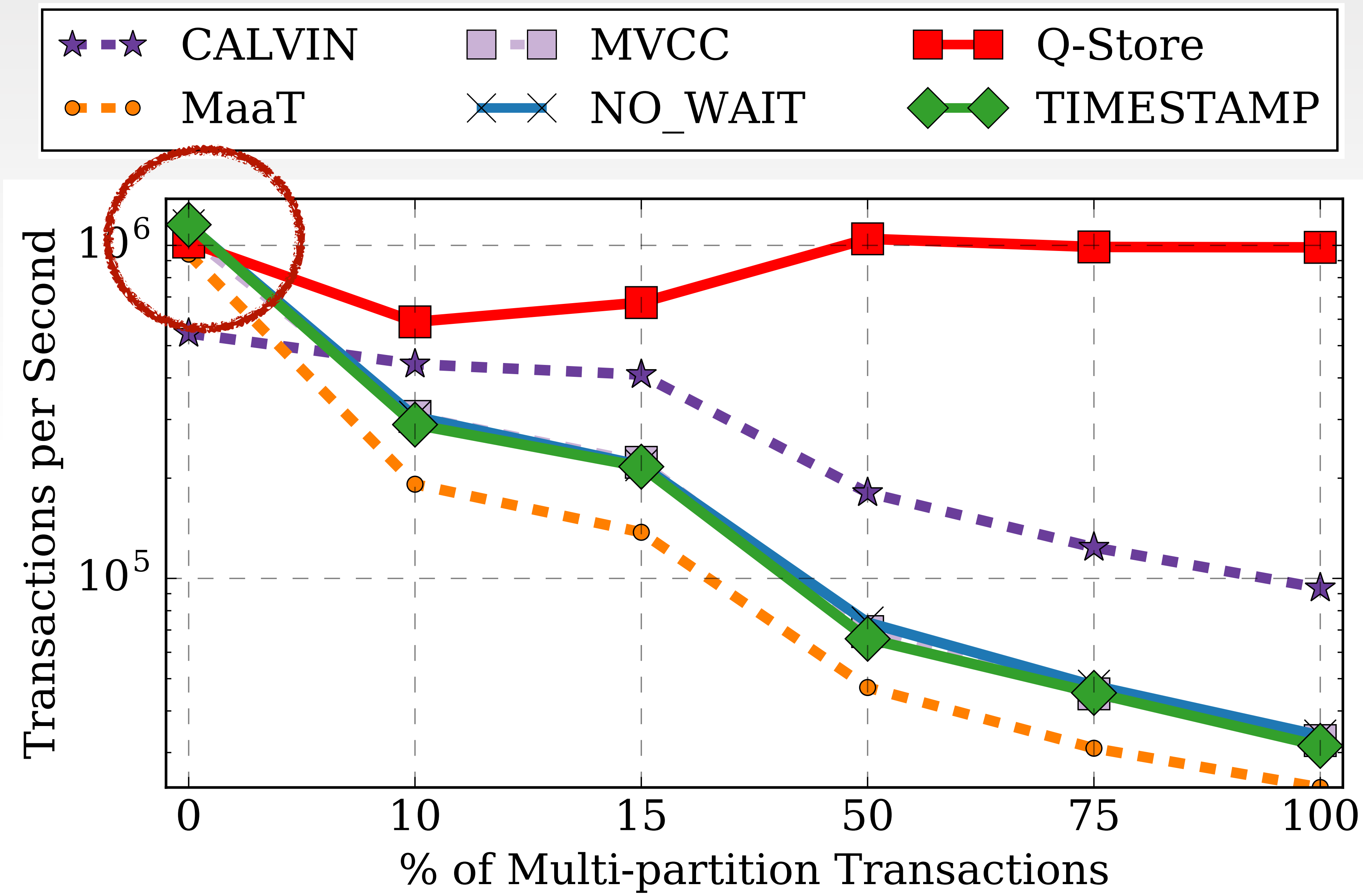
Q-Store eliminates the bottleneck of single-threaded sequencing scheduling and scales well while increasing the batch size

Effect of Multi-Partition Transactions



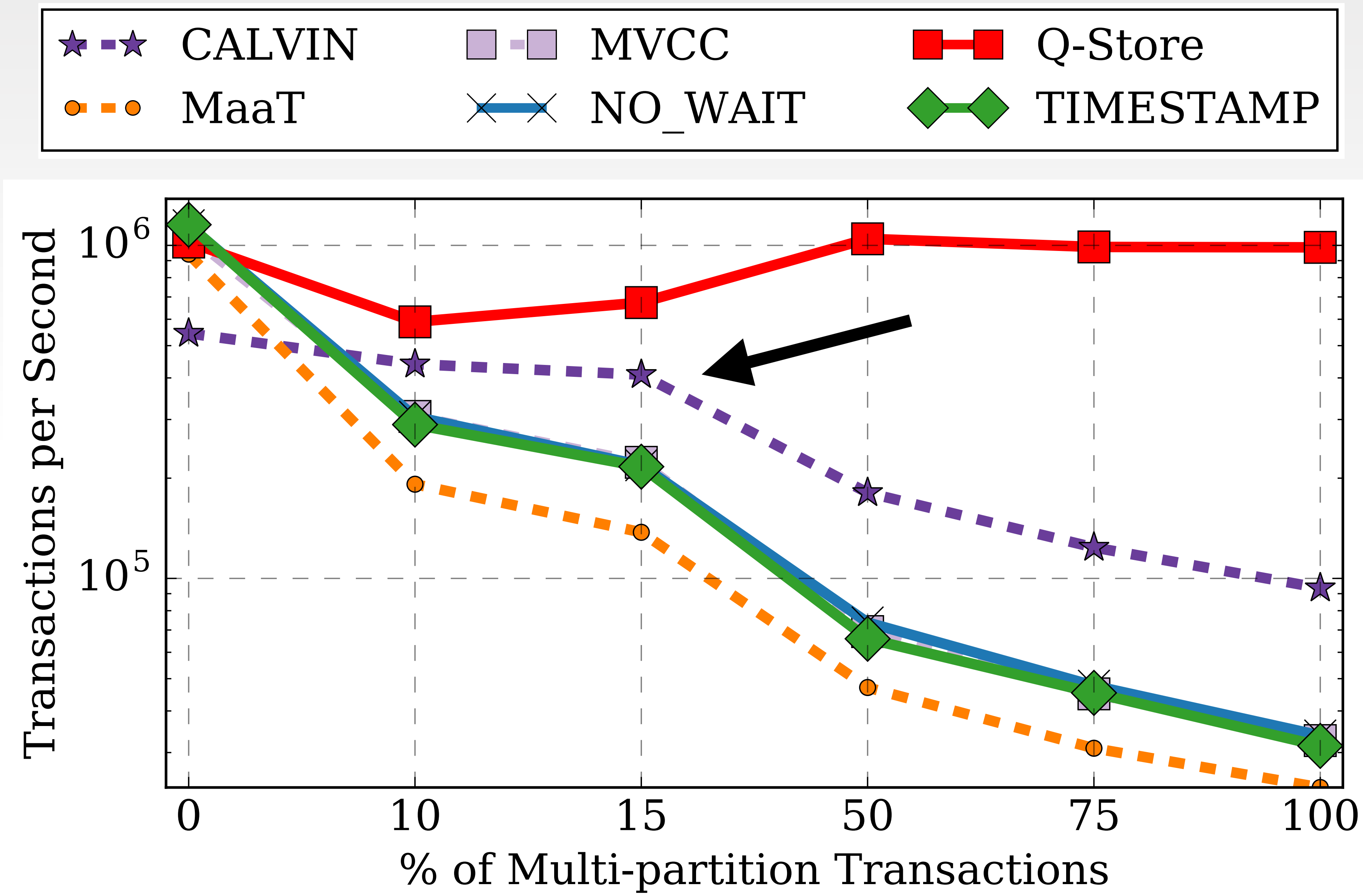
Q-Store's performance is comparable to non-deterministic protocols with 0% MPT

Effect of Multi-Partition Transactions



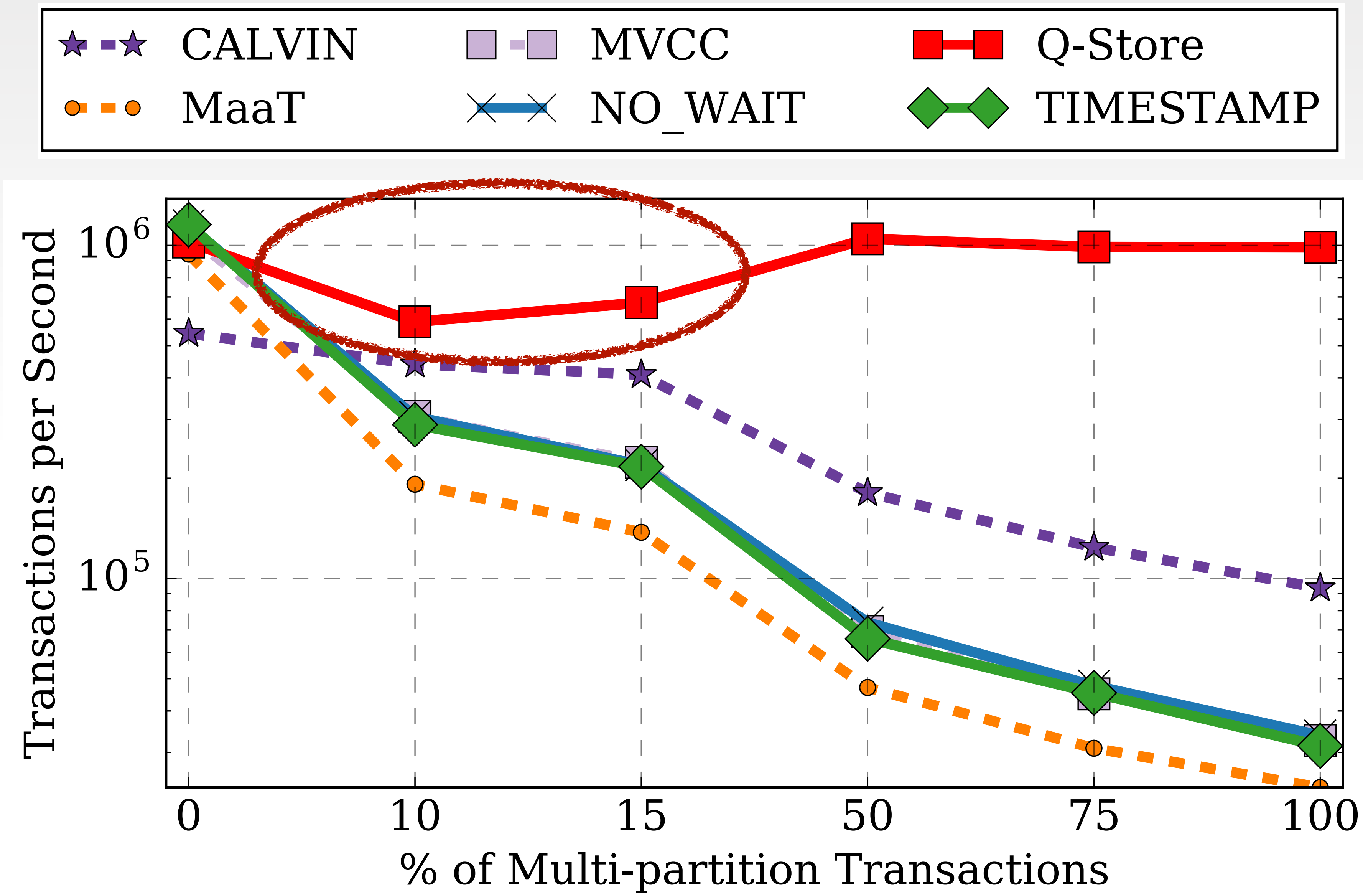
Q-Store's performance is comparable to non-deterministic protocols with 0% MPT

Effect of Multi-Partition Transactions

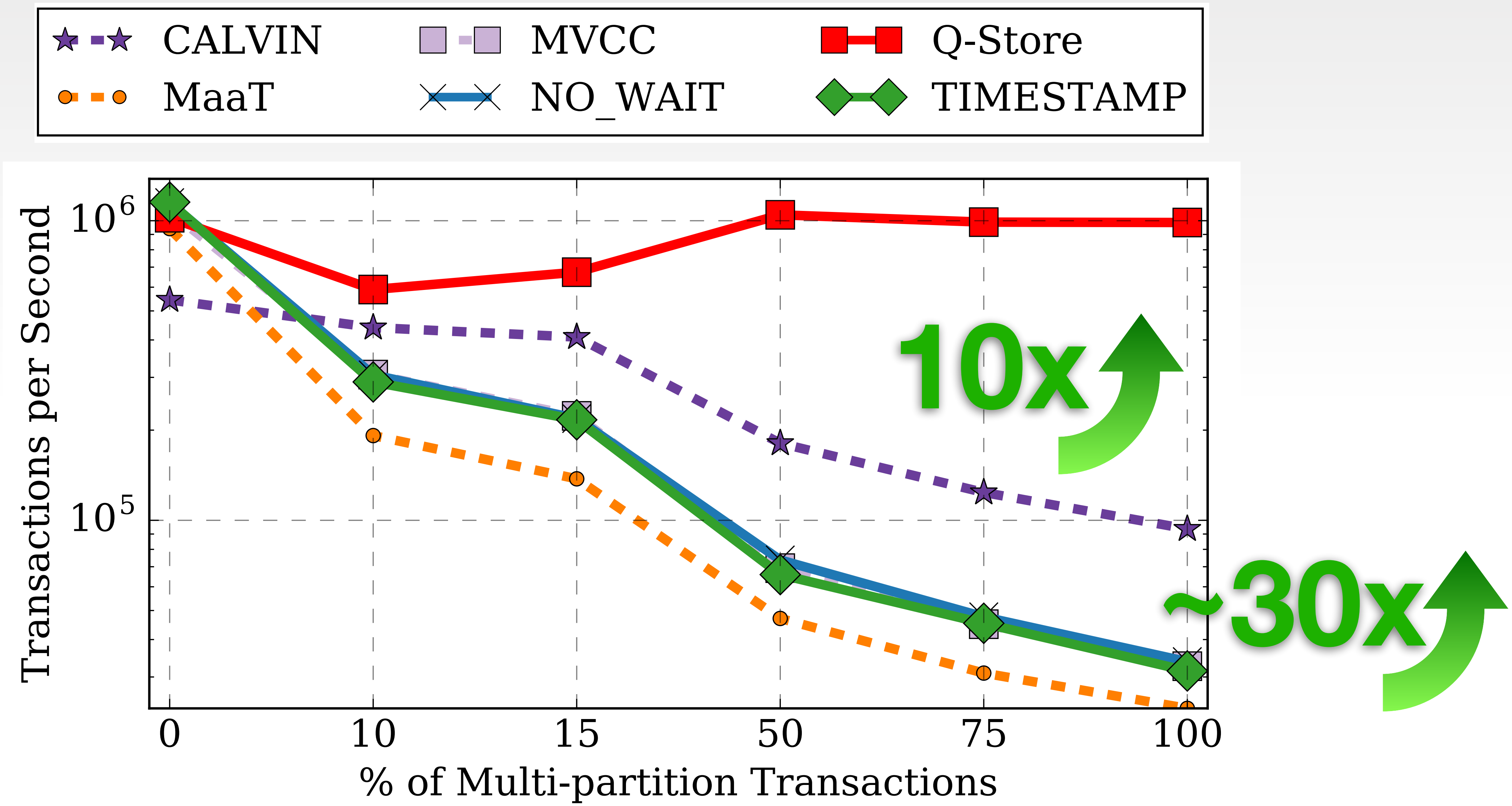


Calvin is sensitive to multi-partition transactions while Q-Store is not

Effect of Multi-Partition Transactions



Effect of Multi-Partition Transactions



Best performance with multi-partition transactional workload

Conclusions and Future Work

- We can **improve the performance and efficiency** of deterministic transaction processing by using **queue-oriented transaction processing principles**
- Q-Store improves system throughput over Calvin by **up to 22x**
- Q-Store improves system throughput over non-deterministic protocols by **up to two orders of magnitude**
- Future work include studying and developing **queue-oriented protocols for byzantine fault-tolerance in database systems**