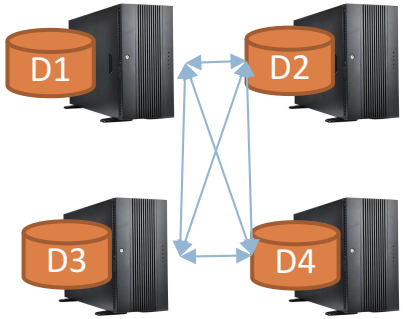
Solar: Towards a Shared-Everything Database on Distributed Log-Structured Storage

Tao Zhu, Zhuoyue Zhao, Feifei Li, Weining Qian,
Aoying Zhou, Dong Xie, Ryan Stutsman, Haining Li, Huiqi Hu

Background

- Single-Node In-Memory DBMS
  - ✓ High xact processing performance
  - × Limited scalability

- Shared-nothing DBMS
  - ✓ Scale out via horizontal partitioning
  - × Poor performance w/ distributed xact



- Shared-everything DBMS
  - ✓ Scalable storage and xact via fast inter-node communication
  - × Expensive network infrastructure

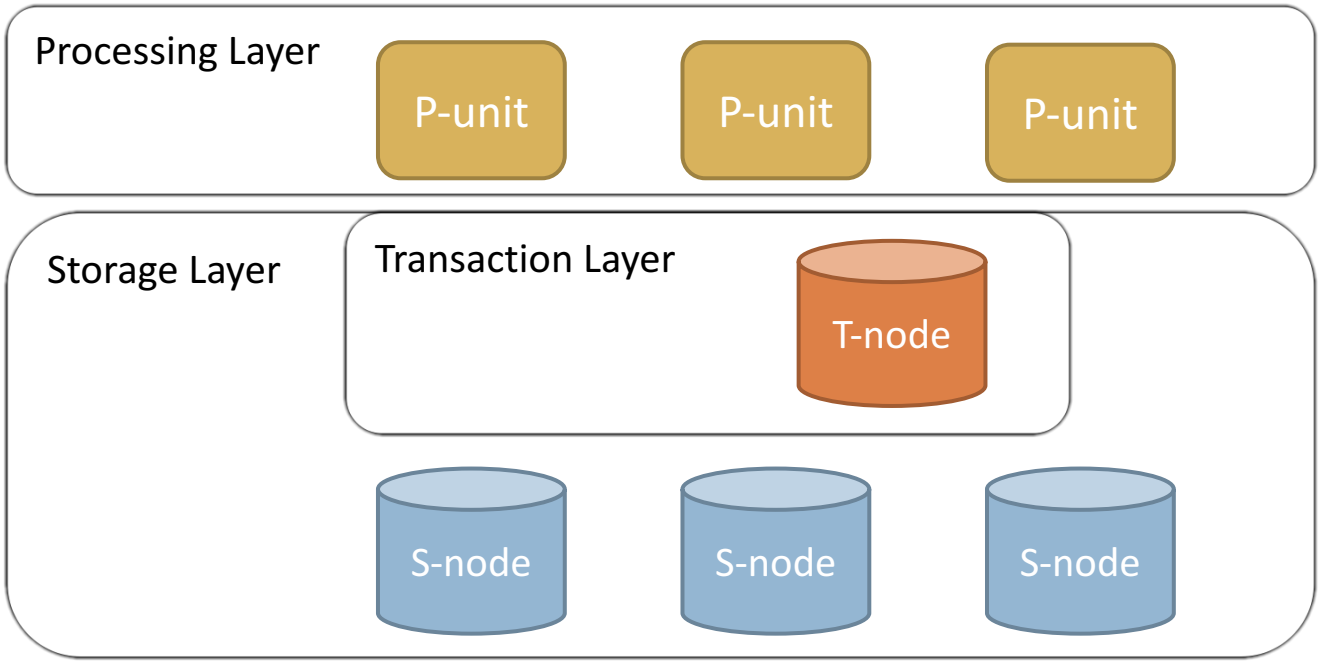
Architecture

- Design considerations
  - General workloads w/ distributed transactions
  - Storage scalability
  - Commodity machines

Goal: high performance OLTP DBMS
w/o assumption on workloads or hardware

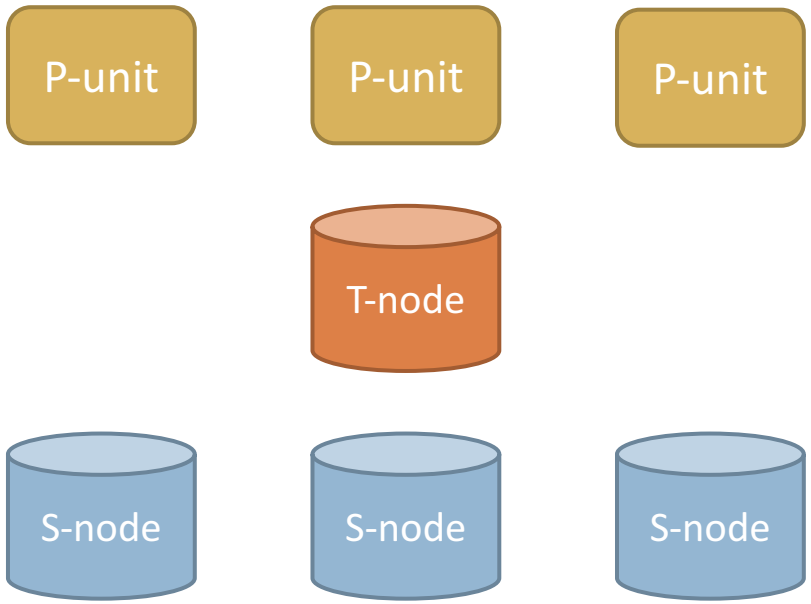
Architecture

- Overview
  - Several ***S-nodes*** (*storage & snapshot read*)
  - A ***T-node*** (*transaction validation/commit & delta read*)
  - Several ***P-units*** (business logic processing)



Architecture

- S-nodes
  - Distributed storage engine
  - Role: storing a consistent database snapshot (SSTable)
  - Feature: supporting scalable data storage



Tablet 1

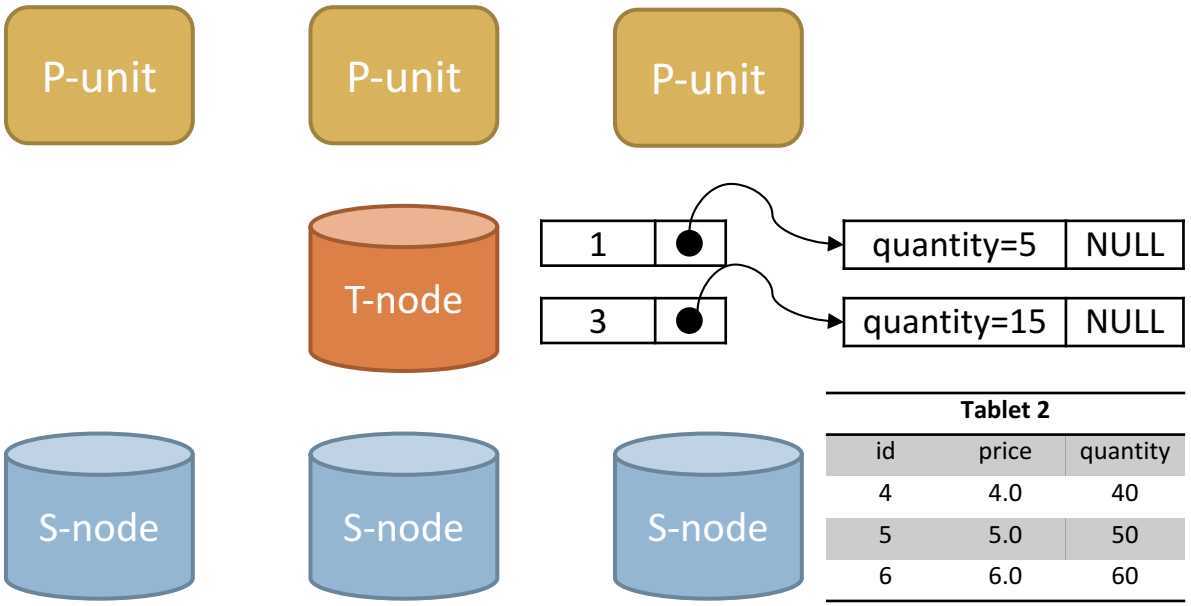
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30

Tablet 2

id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

Architecture

- T-node
  - In-memory transaction engine
  - Role: managing newly committed data since the last snapshot (Memtable)
  - Feature: servicing performant transaction writes



Tablet 1

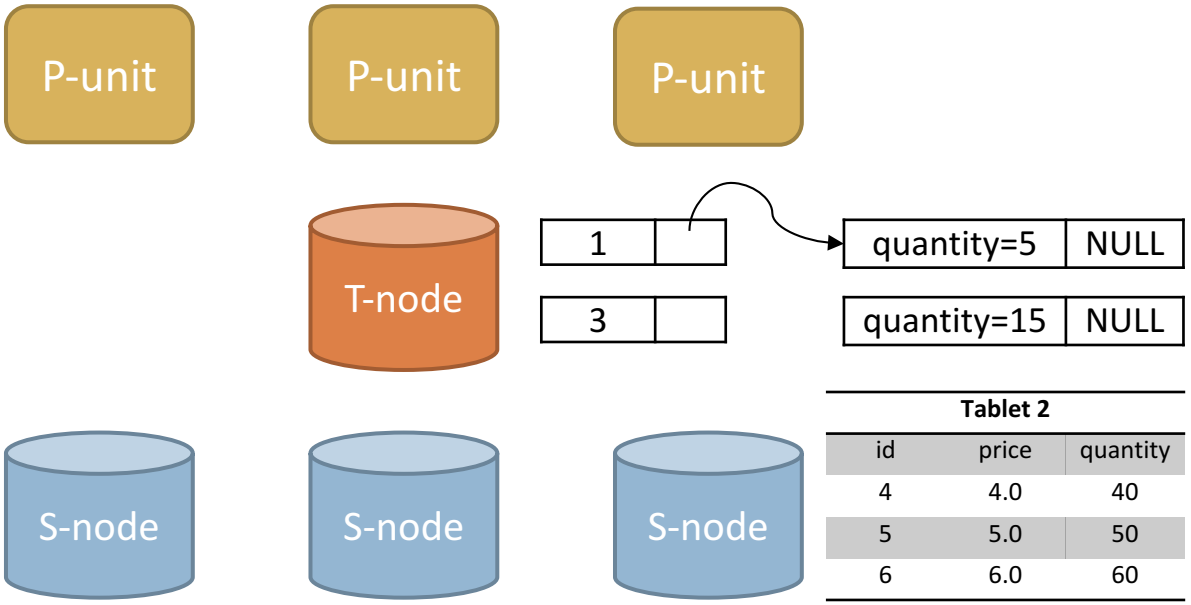
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30

Tablet 2

id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

Architecture

- P-units
  - Distributed query processing engine
  - Role: SQL, stored procedure, query processing, remote data access
  - Feature: scalable computation power



Tablet 1

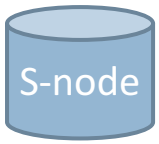
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30

Tablet 2

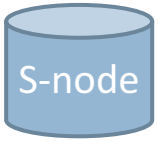
id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

LSM-Tree style storage

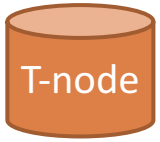
- SSTable
  - A consistent snapshot
  - Data partitioned into tablets (ranges over tables)
  - Tablets replicated over S-nodes
- Memtable
  - Newly committed data
  - Stored in memory on T-node
  - Multi-version storage
  - Replicated to backup T-nodes

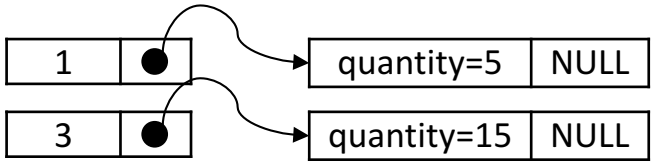


Tablet 1		
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30



Tablet 2		
id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60



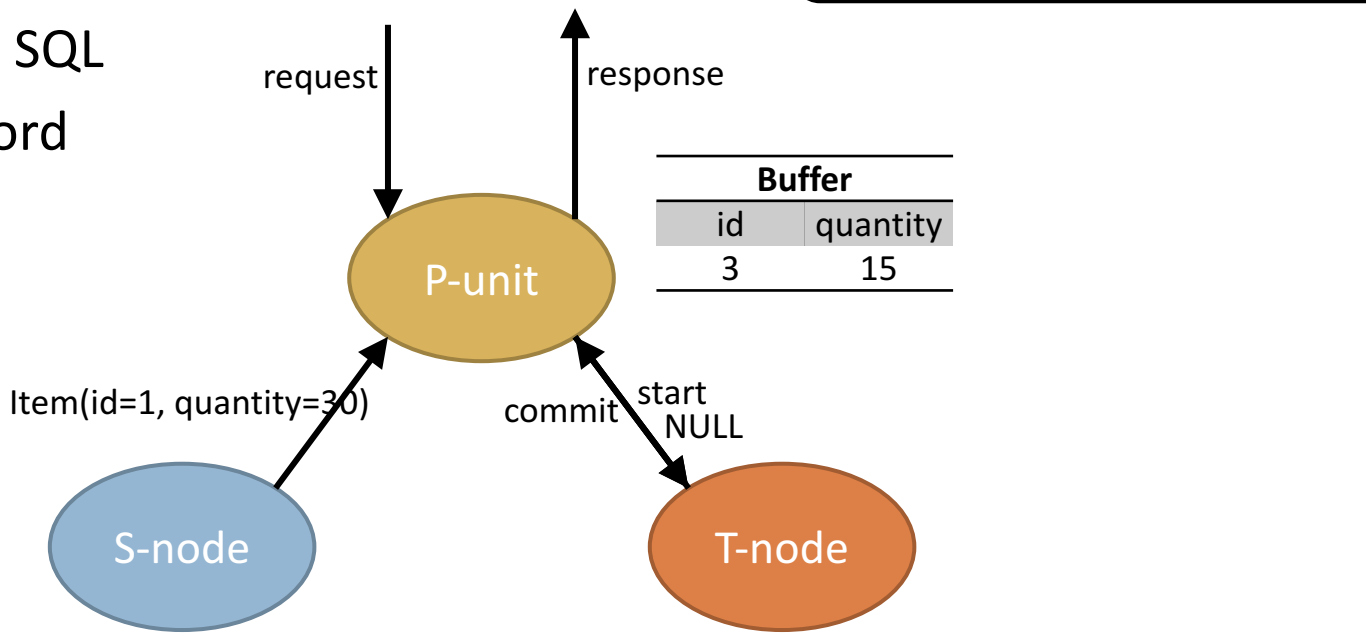


Item Table		
id	price	quantity
1	1.0	5
2	2.0	20
3	3.0	15
4	4.0	40
5	5.0	50
6	6.0	60

Transaction Processing

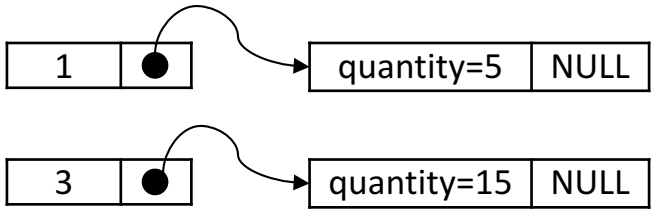
- Start a transaction
- Read a record
- Process the SQL
- Write a record
- Commit

```
UPDATE Item
    SET  quantity = quantity - 15
    WHERE id = 3;
```



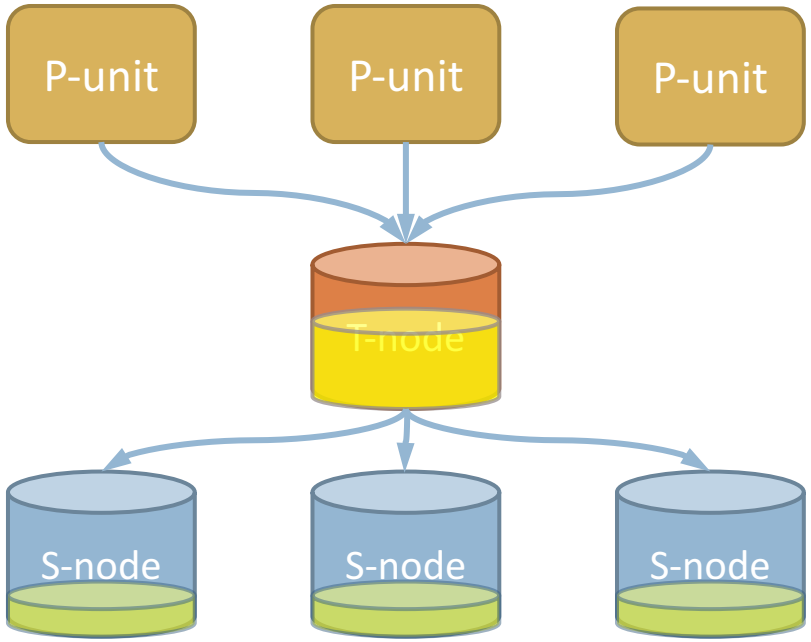
Buffer	
id	quantity
3	15

Tablet 1		
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30



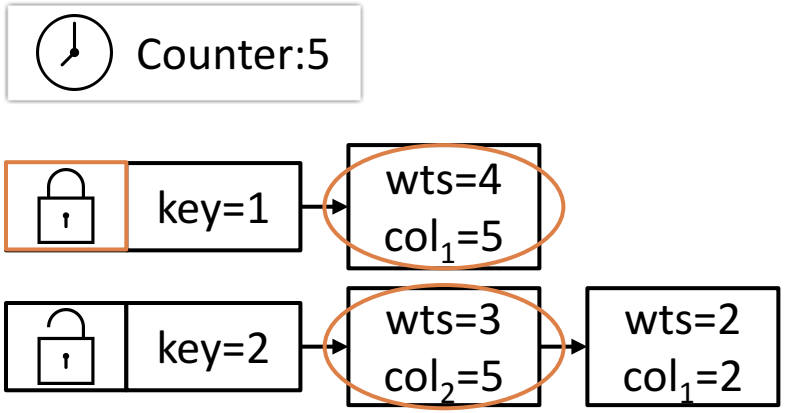
Data Compaction

- All data are firstly written into the T-node
- Data compaction moves committed data into S-nodes
  - Does not block on-going and future transactions



Concurrency Control

- Use MVOCC to support Snapshot Isolation (SI)
  - Prevent lost update anomaly
- Data structures on the T-node
  - A timestamp counter(MVCC)
  - Row-level latch (OCC)
- Snapshot Acquisition
- Transaction Validation



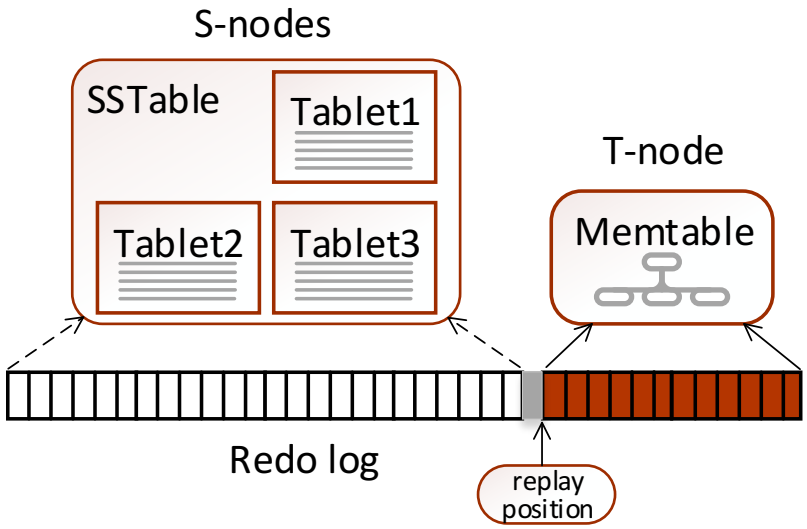
Txn t_x

read-timestamp: $rts = 5$

Write(key=1, col_1 = 3)

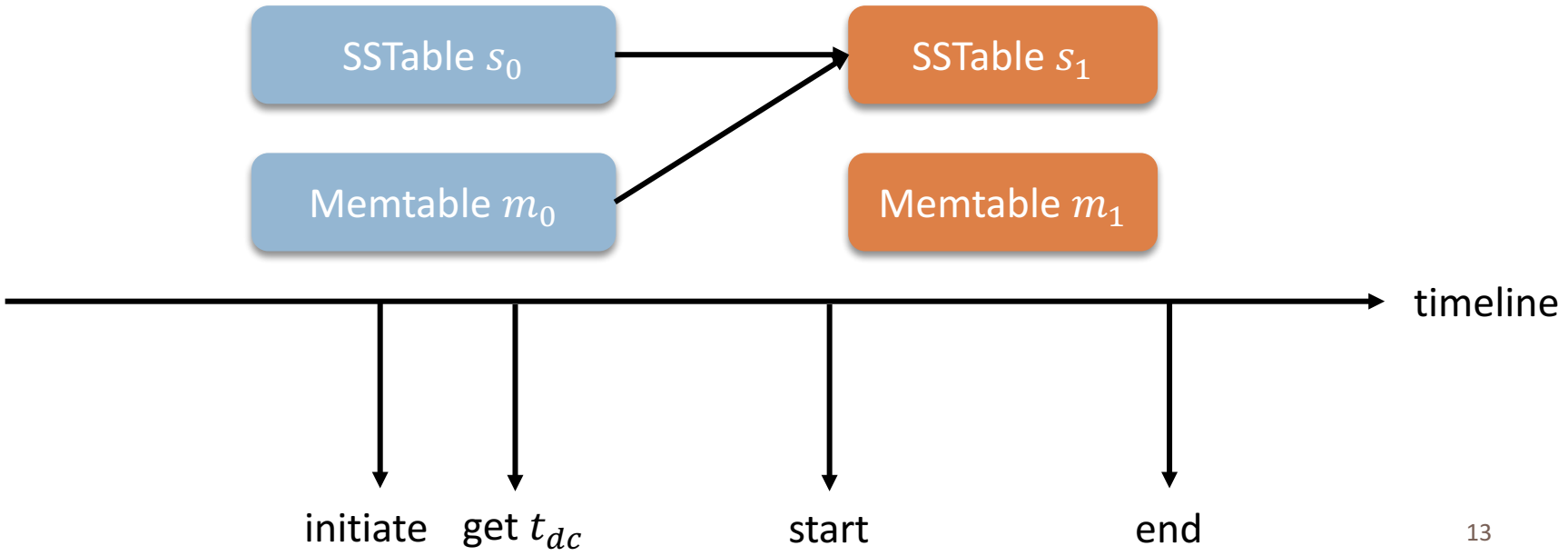
Recovery

- Write ahead log
  - Generate redo log entries
  - Group commit
- T-node recovery
  - Replay redo log entries
  - The replay position is determined by the last finished data compaction
- S-nodes do not lose data
- P-units do not store data



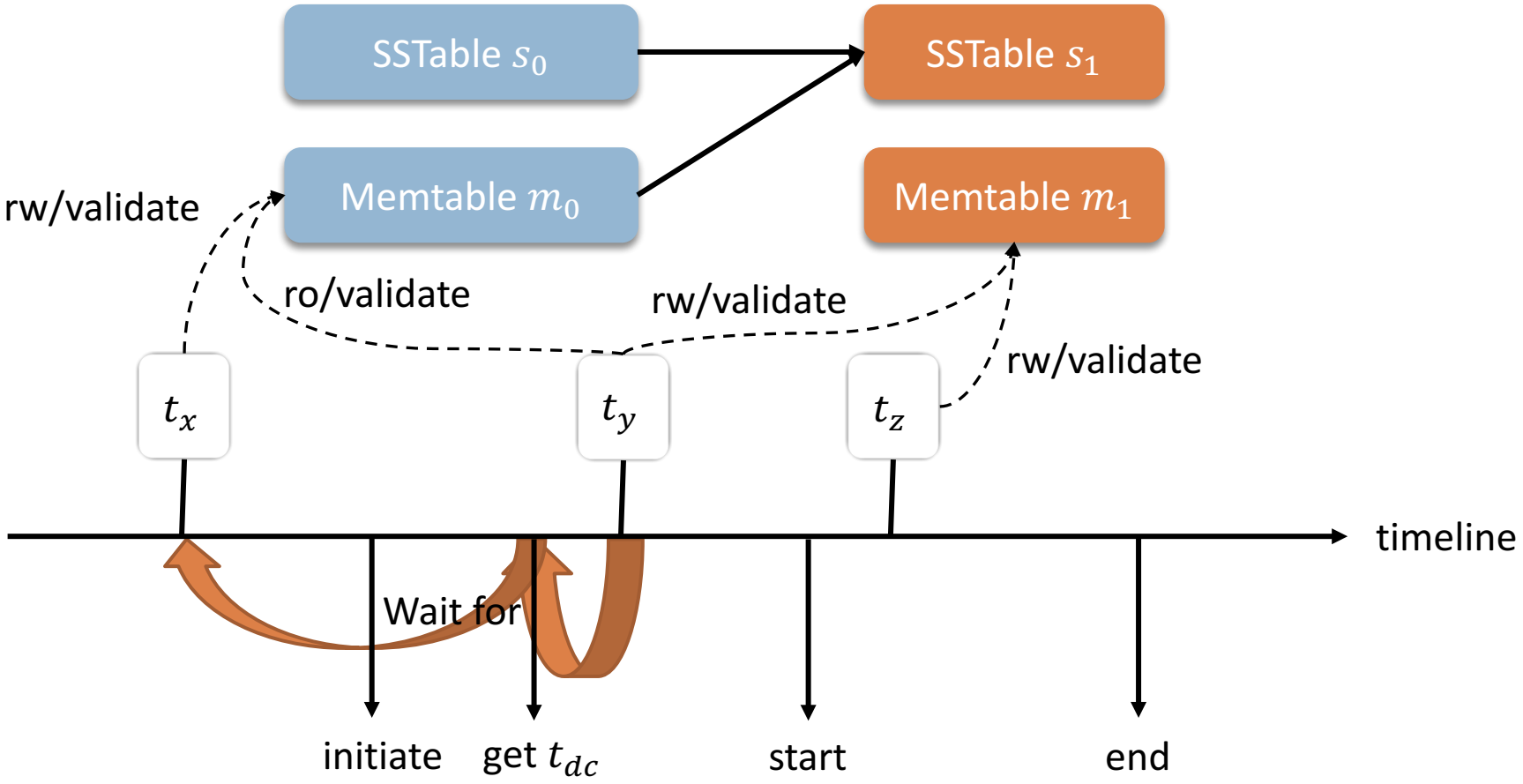
Data Compaction

- Data compaction (DC) starts when the T-node runs out of memory
  - New Memtable m_1 to accept transactions after DC initiation
  - Memtable m_0 is frozen and merged into SSTable



Transaction and CC during Data Compaction

- Goal: minimize blocking of transaction processing

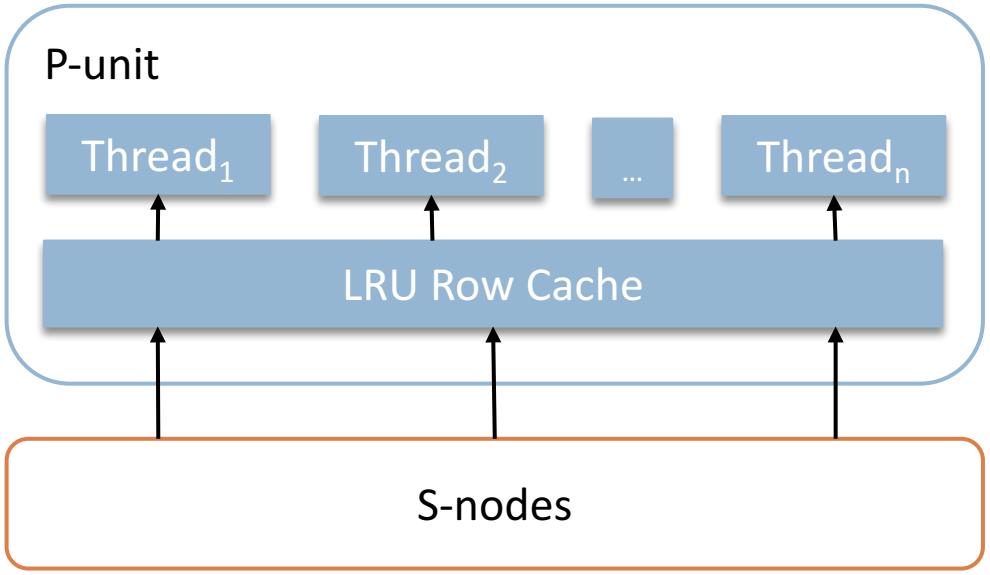


Remote Data Access Optimization

- Shared-Everything architecture
  - Latency bounded by remote data access between
    - P-unit and T-node
    - P-unit and S-node
  - Reducing remote data access cost
    - => more concurrent transactions
    - => higher throughput

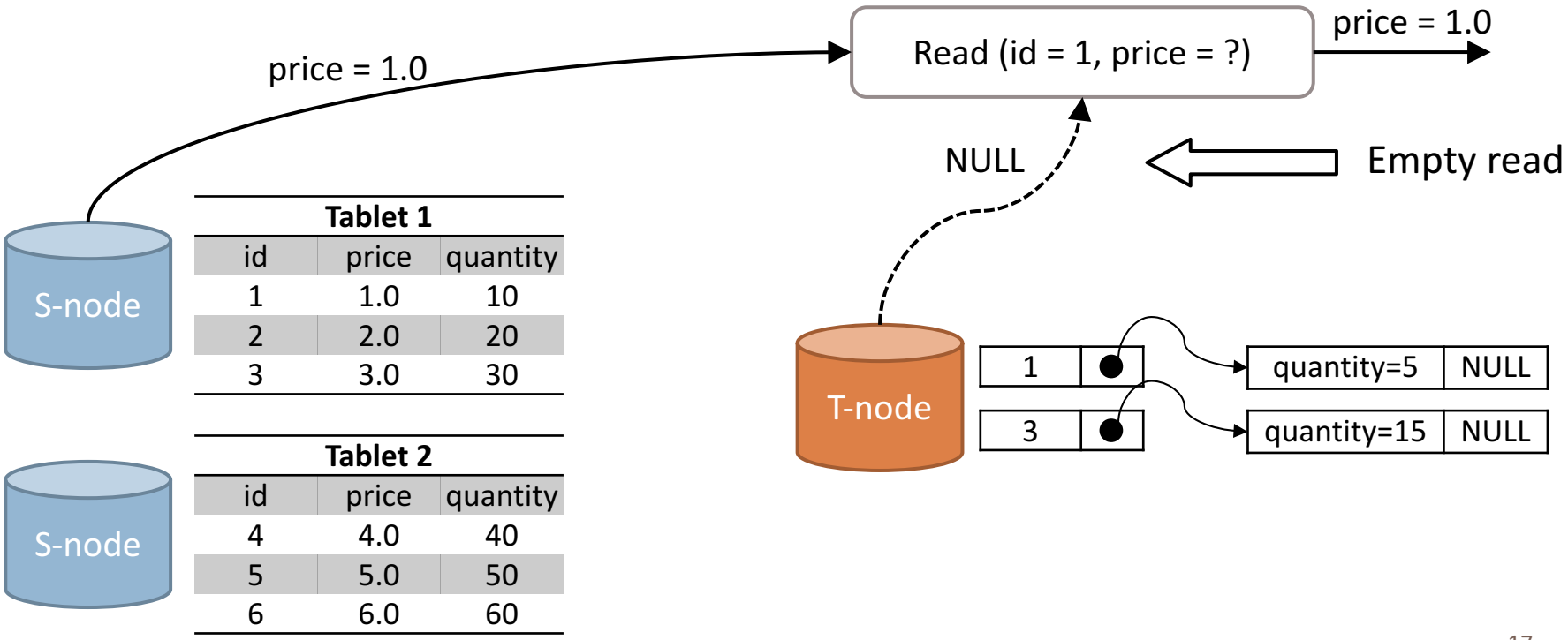
Local SSTable Cache

- Build SSTable Cache on P-unit
  - SSTable is immutable
  - P-unit examines its local cache before communicating with S-nodes



Asynchronous Bit Array

- Empty reads on the T-node
  - The T-node stores a small part of data
  - Reading non-existing data items results in useless empty reads



Tablet 1

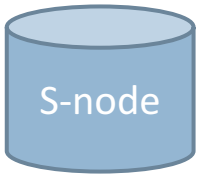
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30

Tablet 2

id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

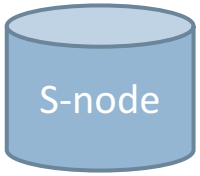
Asynchronous Bit Array

- Asynchronous Bit Array
  - Encode whether any row in Tablet x has its column y modified
  - Periodically synchronized to P-units
    - False positive => empty read (corrected after the first access)
    - False negative => validating empty reads and retry



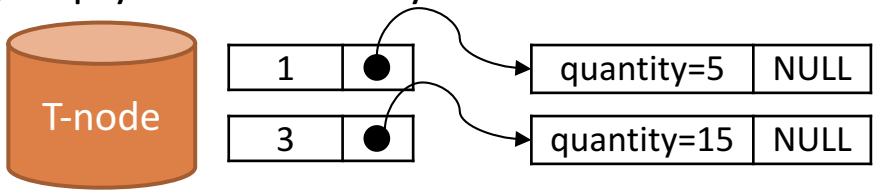
Tablet 1

id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30



Tablet 2

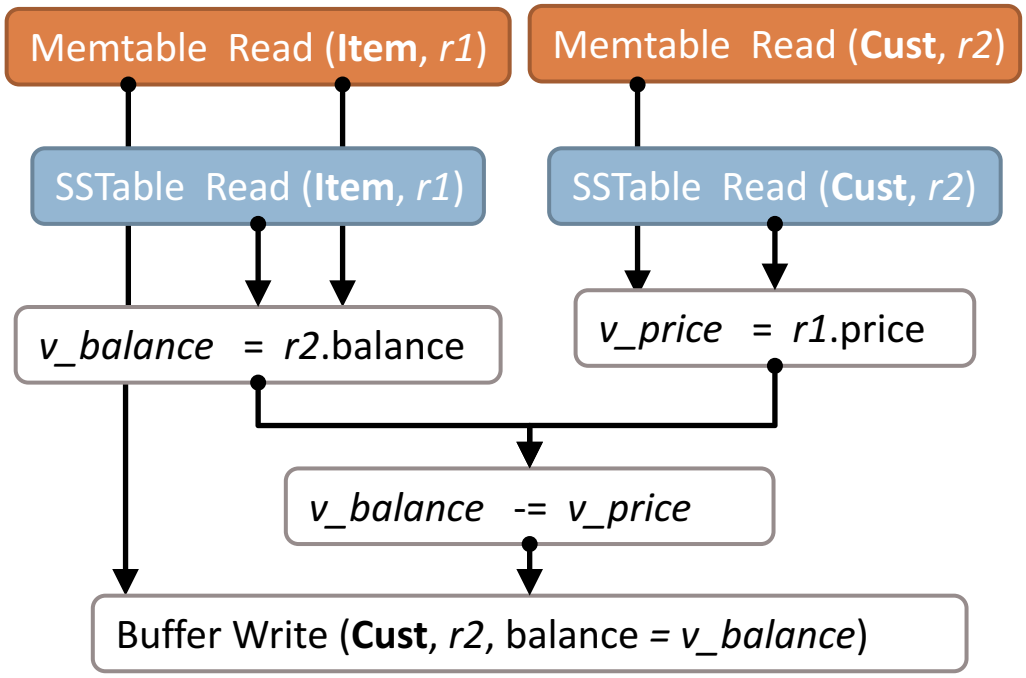
id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

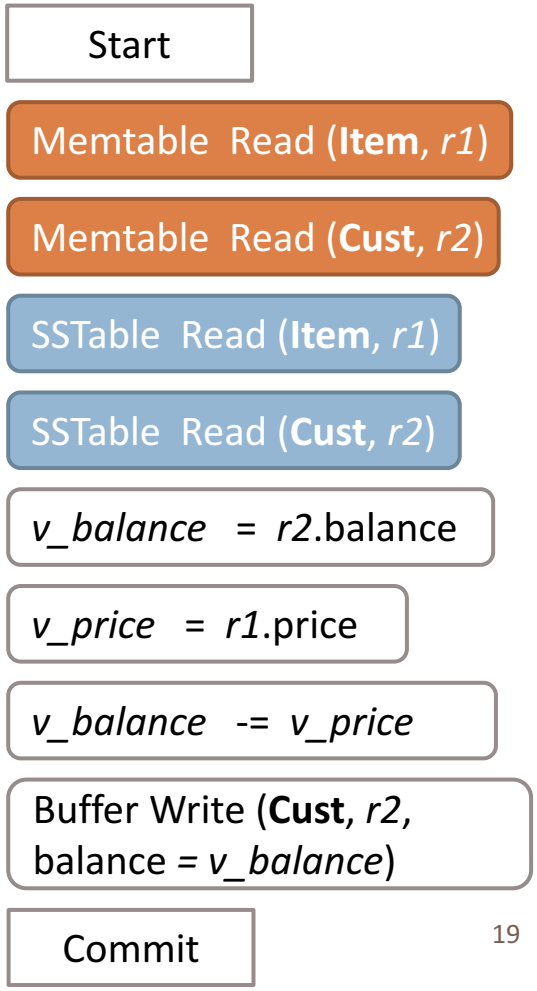


Any data in the T-node		
	price	quantity
Tablet 1	0	1
Tablet 2	0	0

Transaction Compilation

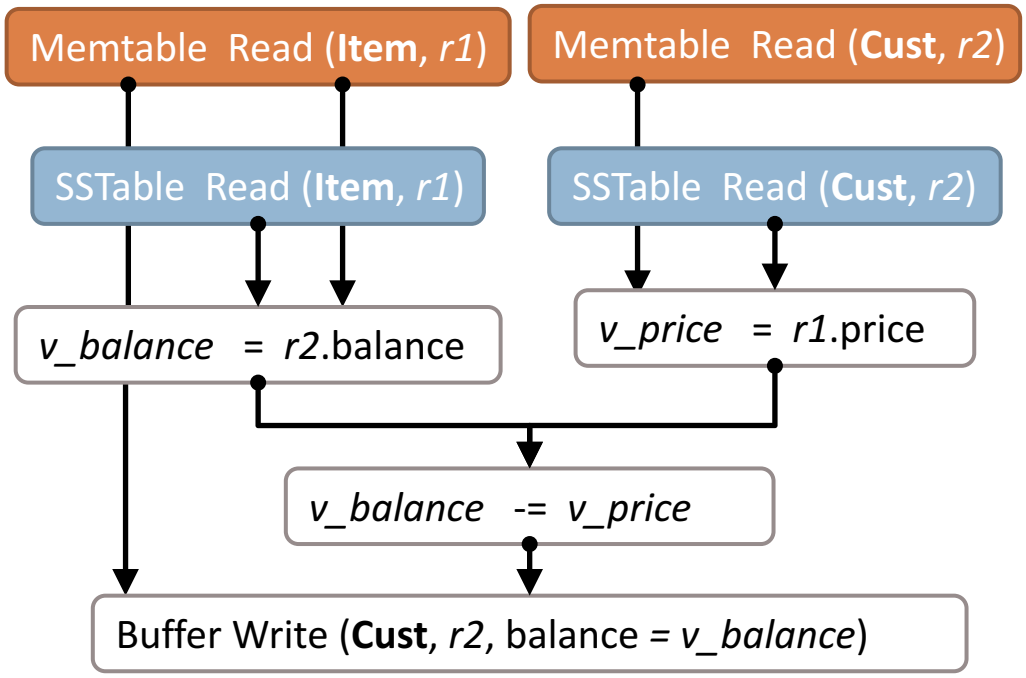
- Model a transaction as a directed acyclic graph
- Move reads to start if possible

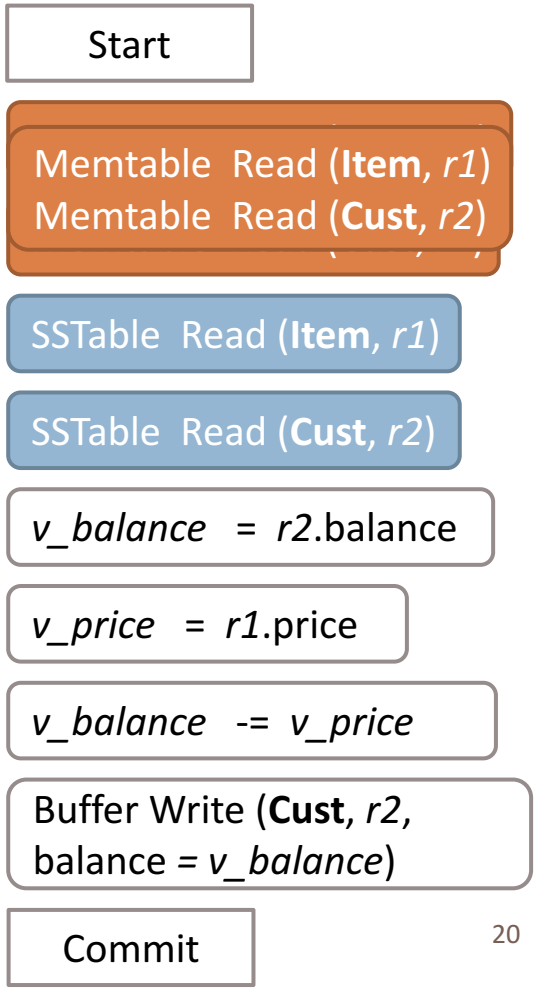




Transaction Compilation

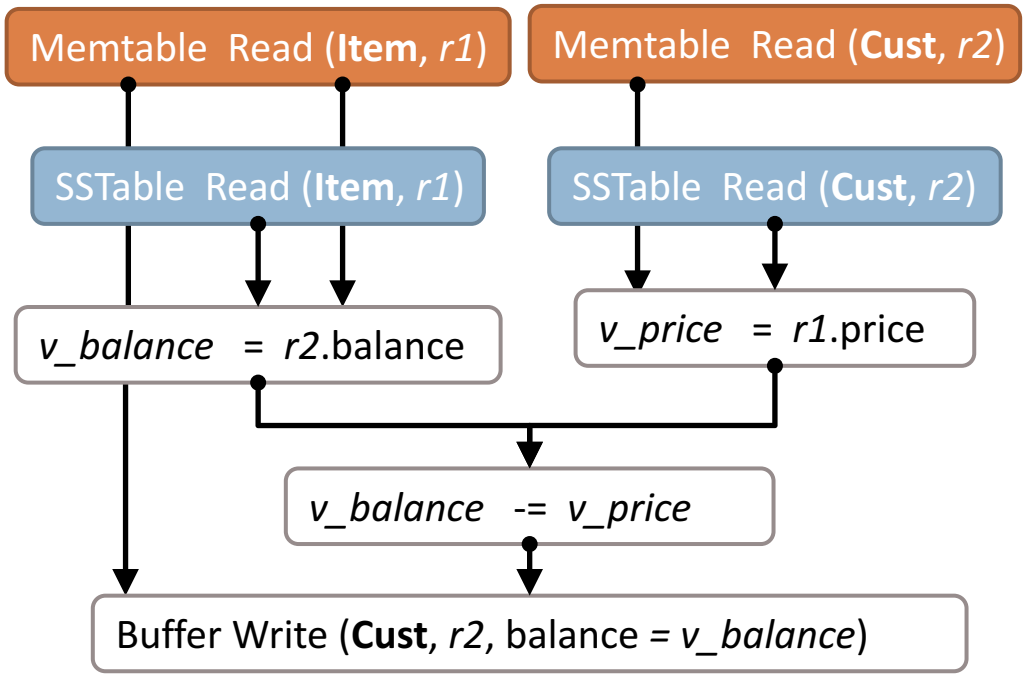
- Group T-node access

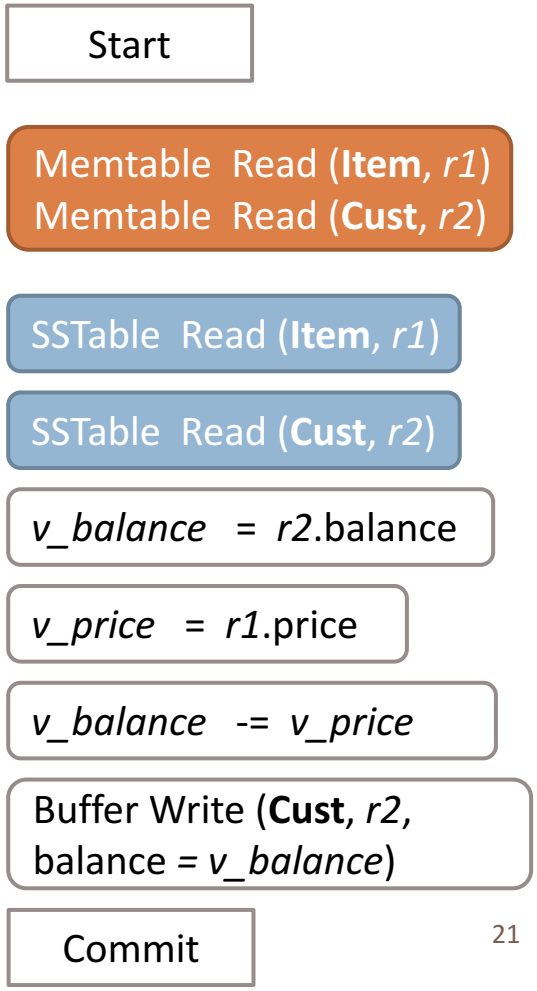




Transaction Compilation

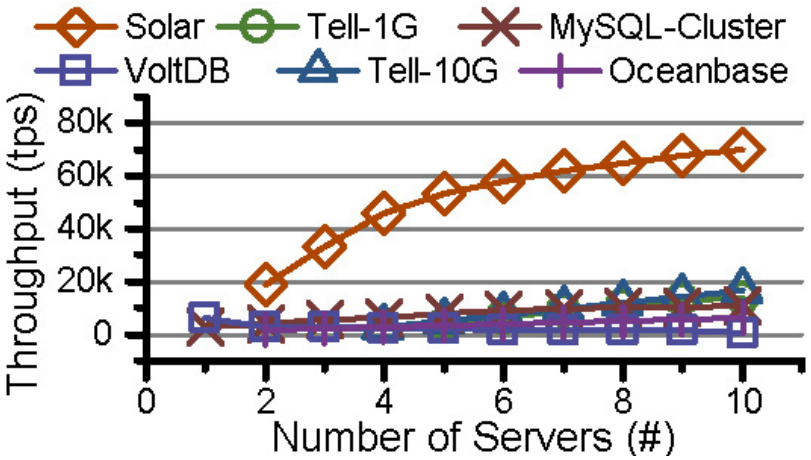
- Pre-execute S-node access



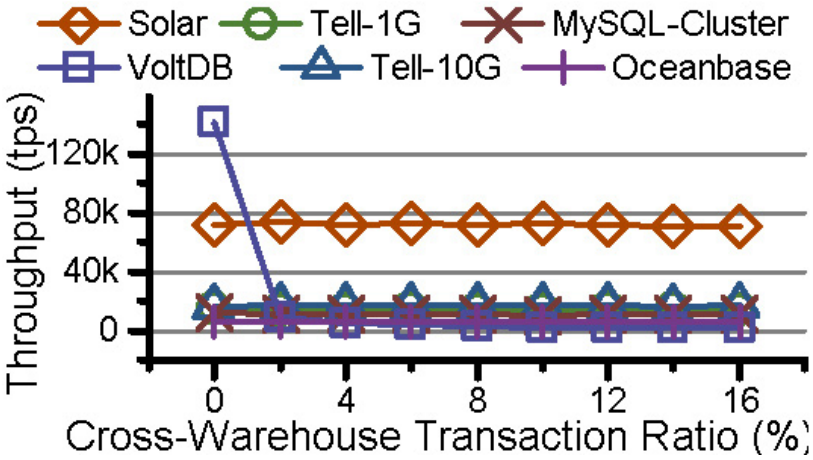


Experiment

- Setting
  - CPU: 2.4G Hz 16-Core
  - Memory: 64GB
  - 10 servers
  - Connected by 1 Gigabits Network
- Benchmark: TPC-C
- Systems
  - Workload: TPC-C
  - MySQL Cluster
  - VoltDB
  - Tell



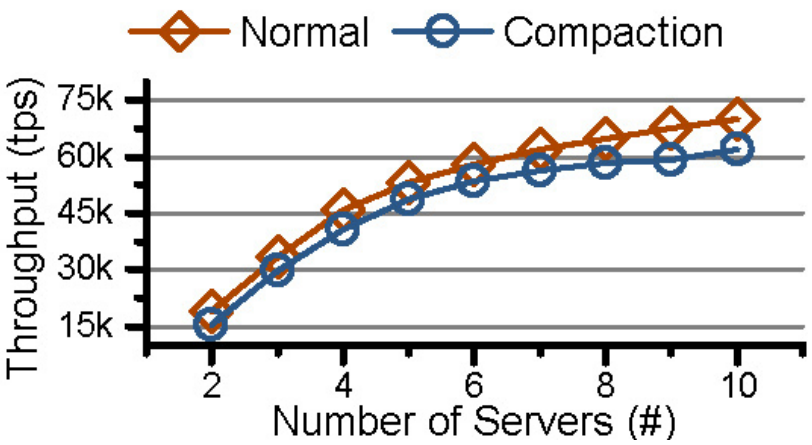
Scalability



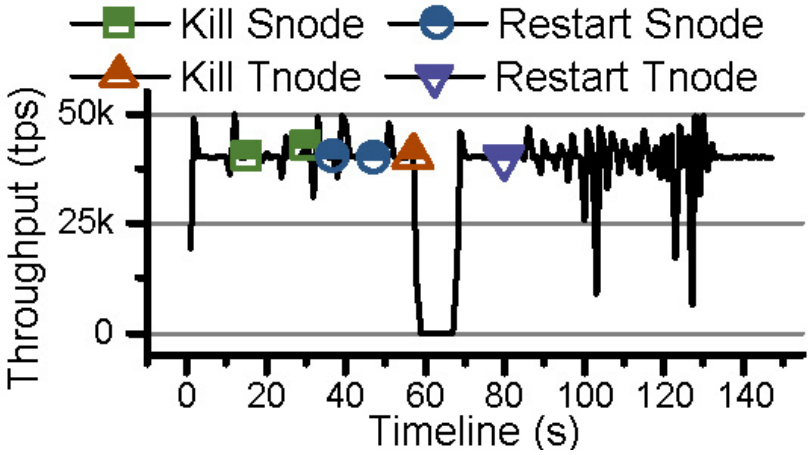
Cross-Partition Transactions

Experiment

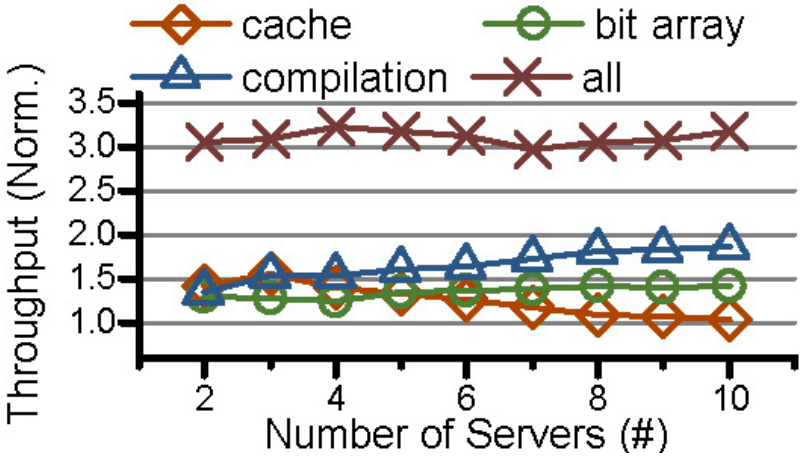
- Data compaction



- System recovery



- Remote data access optimization

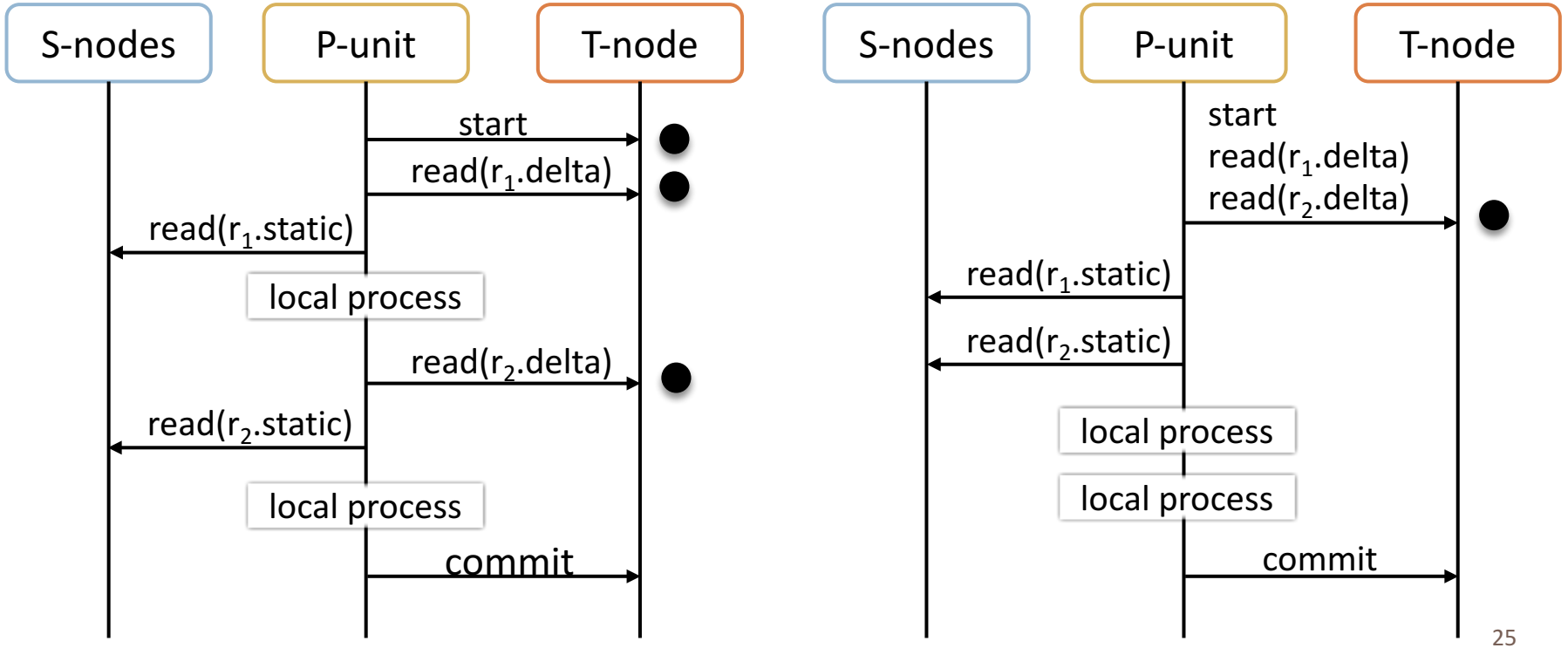


Summary

- Solar
  - A shared-everything OLTP DBMS on Commodity hardware
    - High performance transaction processing
    - Scalable data storage capacity
  - Several novel optimization to improve performance
  - Empirical evaluation shows great performance and scalability

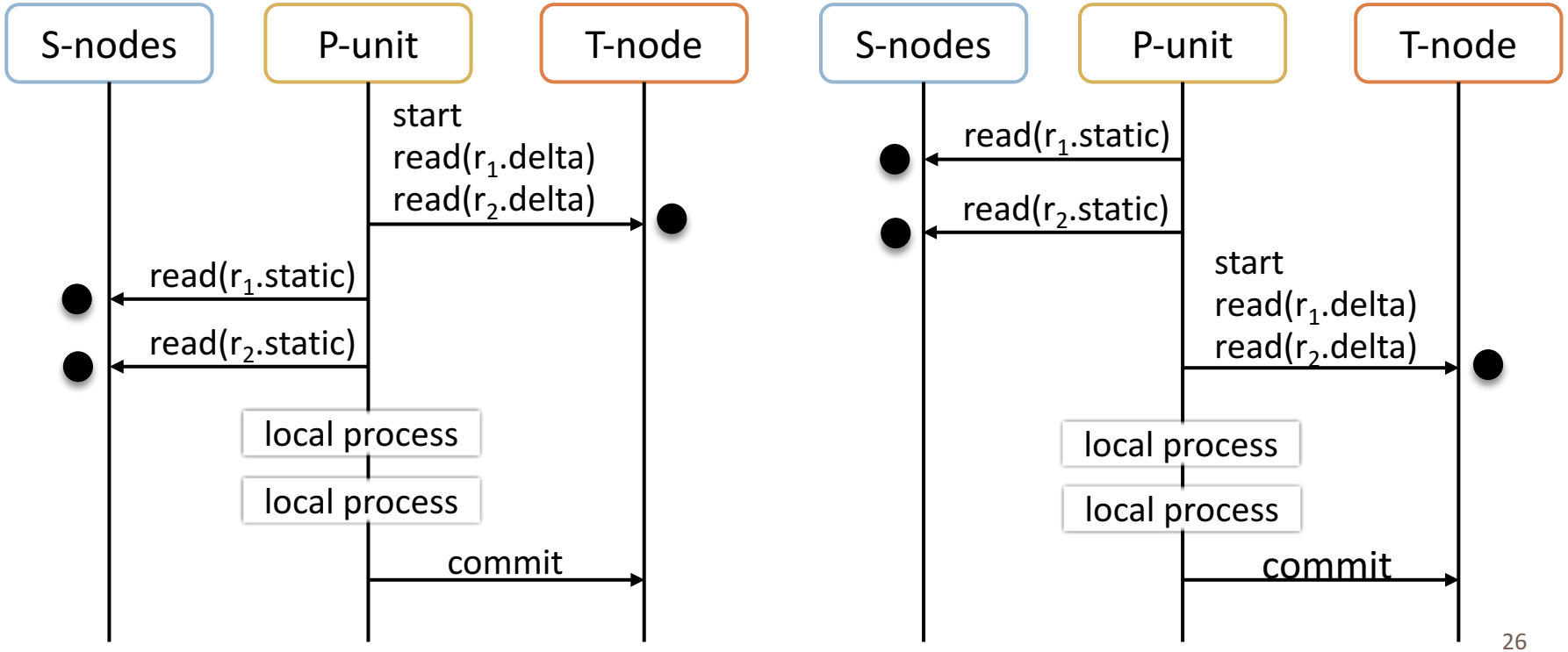
Transaction Compilation

- Group T-node access
  - Normal execution issues T-node access one-by-one
  - Try to batch multiple T-node communications together



Transaction Compilation

- Pre-execute S-node access
  - Normal execution issues S-node access after transaction is started
  - Try to pre-execute S-node reads



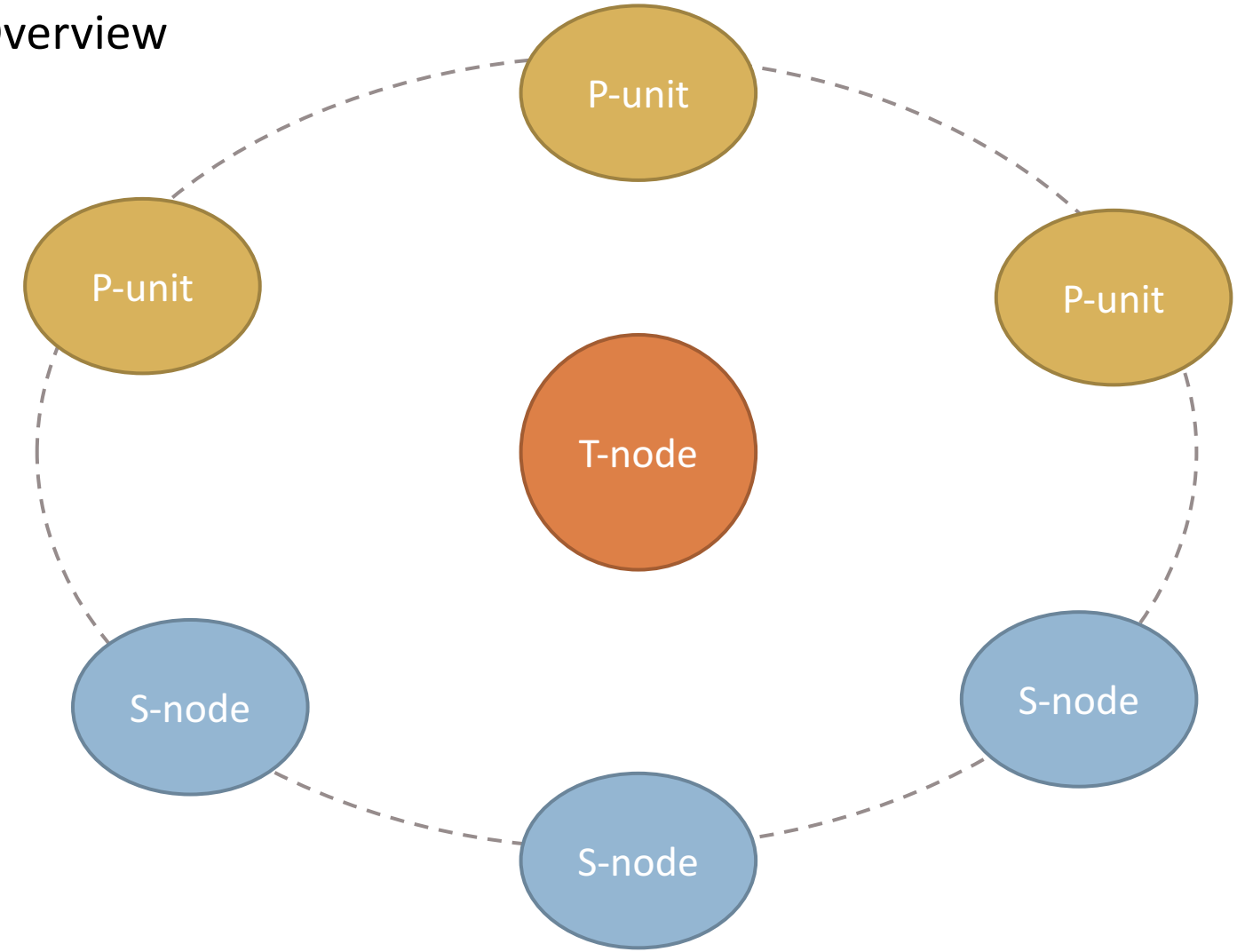
Architecture

- Design considerations
  - A shared-everything architecture
    - 2-Layer LSM-Tree style storage
    - Decouple computation from storage
  - High performance in-memory transaction processing
    - MVOCC, combining the OCC and the MVCC
    - A non-blocking data compaction algorithm
  - Fine-grained remote data access
    - Data cache
    - Asynchronous bit array
    - Transaction compilation

Goal: high performance OLTP DBMS
without assuming a partitionable workload or advanced hardwares

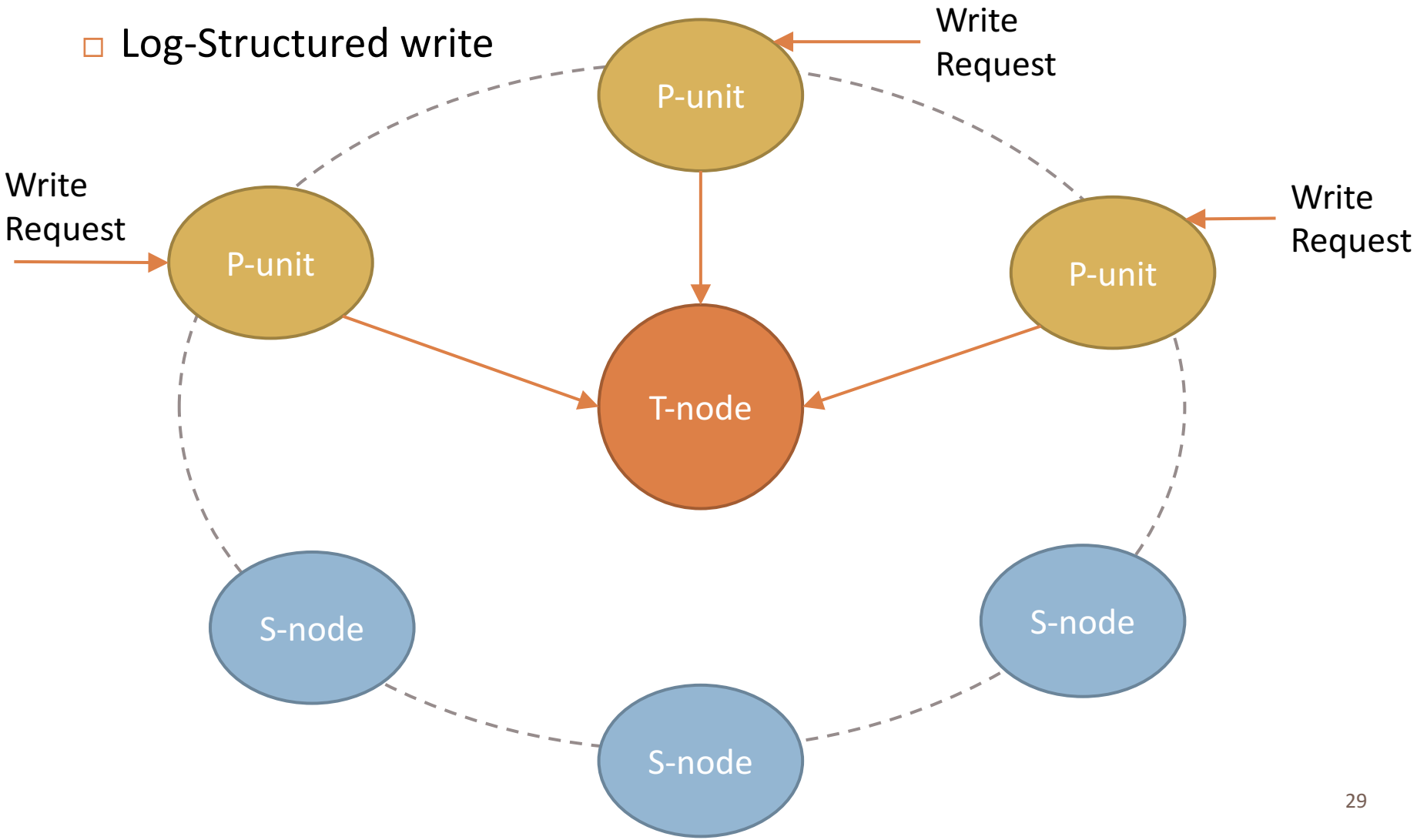
Architecture

- Overview



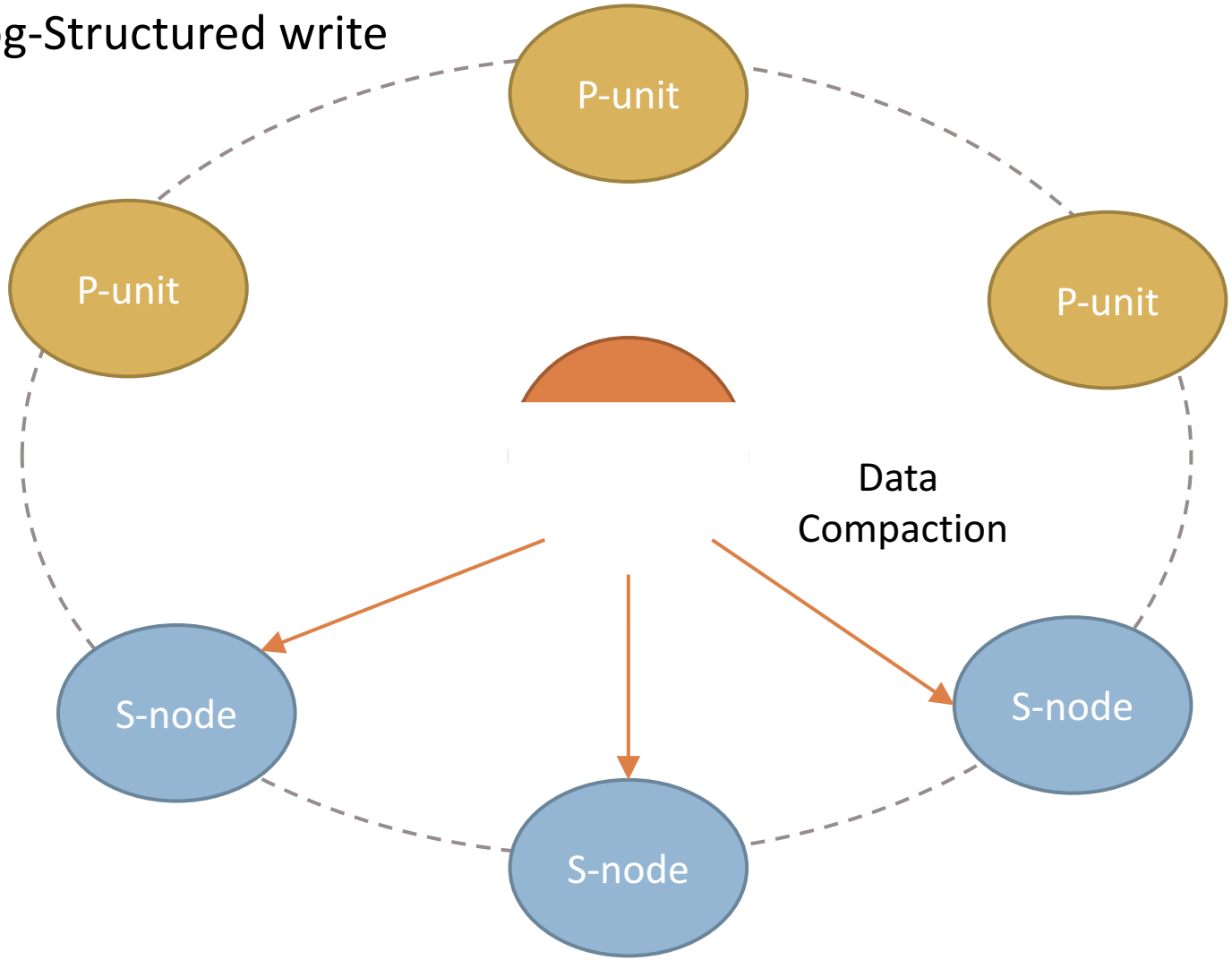
Architecture

- Log-Structured write



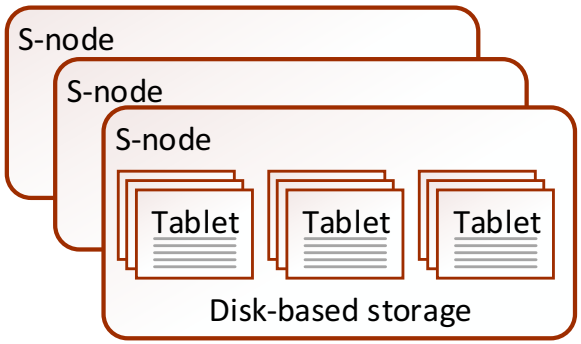
Architecture

- Log-Structured write



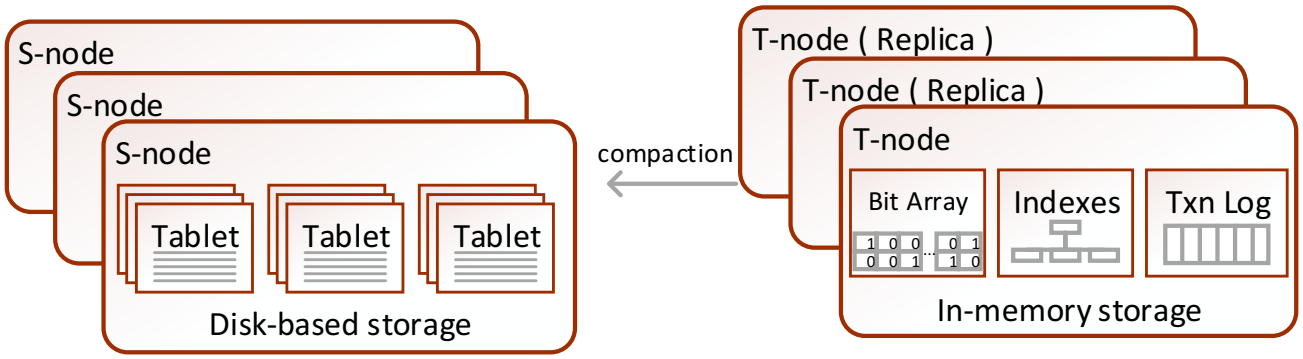
Architecture

- S-nodes
  - Distributed storage engine
  - Role: storing a consistent database snapshot (SSTable)
  - Feature: supporting scalable data storage



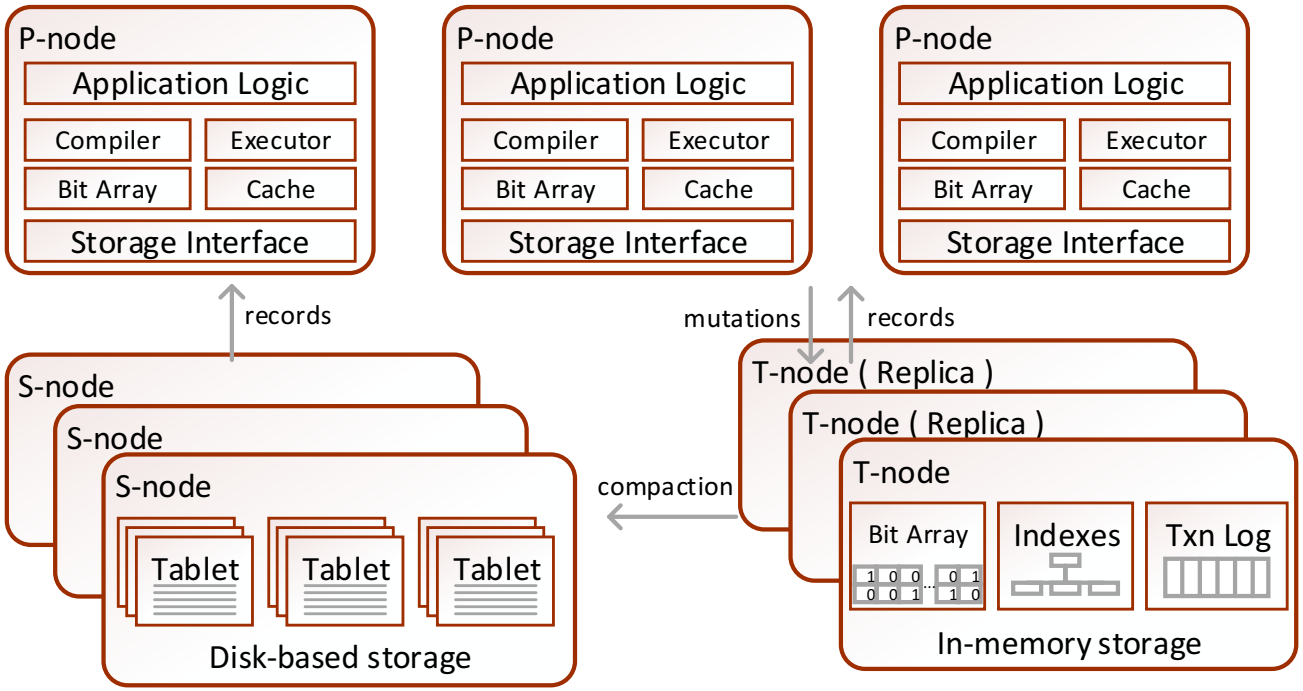
Architecture

- T-node
  - In-memory transaction engine
  - Role: managing the rest recently committed data (Memtable)
  - Feature: providing performant transactional writes



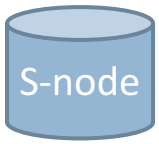
Architecture

- P-units
  - Distributed query processing engine
  - Role: SQL, stored procedure, query processing, remote data access
  - Feature: providing scalable computation power

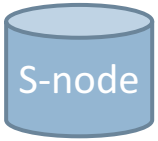


LSM-Tree style storage

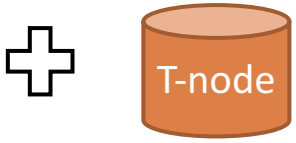
- SSTable
  - A consistent snapshot
  - Partitioned into tablets
  - Replicated over S-nodes
- Memtable
  - Newly committed data
  - In-memory stored in the T-node
  - Multiple version storage
  - Replicated to backup T-nodes



Tablet 1		
id	price	quantity
1	1.0	10
2	2.0	20
3	3.0	30



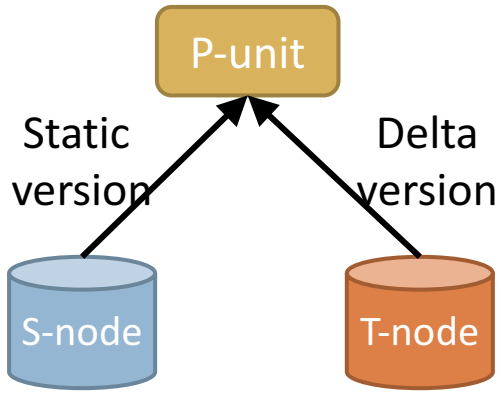
Tablet 2		
id	price	quantity
4	4.0	40
5	5.0	50
6	6.0	60

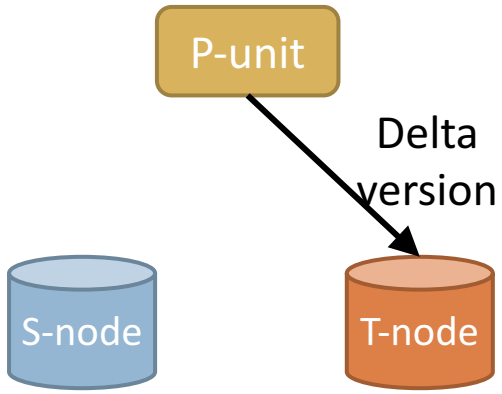


Item Table		
id	price	quantity
1	1.0	5
2	2.0	20
3	3.0	15
4	4.0	40
5	5.0	50
6	6.0	60

Read & Writes

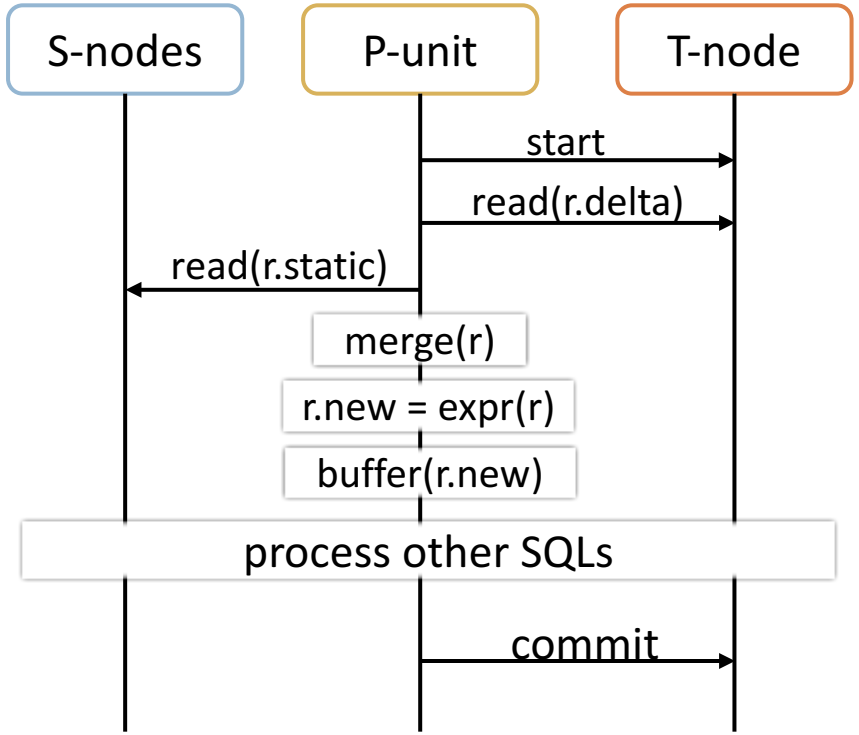
- Read
  - read and merge versions from both T-node and one of S-node
- Write
  - directly write into the T-node





Transaction Processing

- P-unit execute transactions
  - Start a transaction
  - Fetch records from remote
  - Execute user-defined logics
  - Buffer data writes
  - Commit the transaction



Background

- Single-Node In-Memory DBMS
  - Hekaton, HyPer
  - Features
    - No disk I/O during transaction processing (In-memory storage)
    - Transaction compilation
    - Lightweight concurrency control (OCC, MVCC, determinism)
    - Simplified write-ahead logging
    - Very high performance transaction processing
  - Limitations
    - Database size should be smaller than memory capacity

Background

- Shared-Nothing DBMS
  - VoltDB/HStore, Spanner
  - Features
    - Use horizontal partition
    - Reply on two phase commit
    - Scalable transaction processing and storage
  - Limitations
    - Partitionable workload
    - Low percentage of distributed transactions

Background

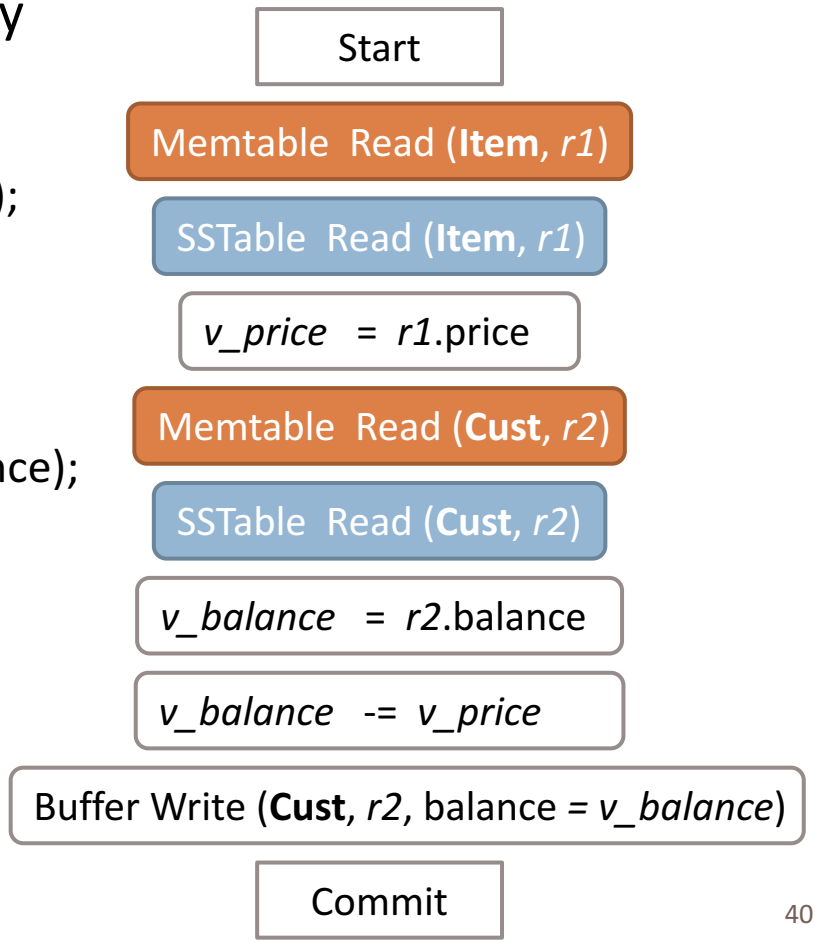
- Shared-Everything DBMS
  - Oracle RAC, Tell
  - Features
    - Share data/cache among nodes
    - Rely on fast inter-node communication
    - Scalable transaction processing and storage
  - Limitations
    - Require advanced network infrastructure
      - InfiniBand switch with 43TB/s, 216 ports costs about $60,000

Transaction Compilation

- Many remote data access between *start* and *commit*
- Group reads to reduce read latency

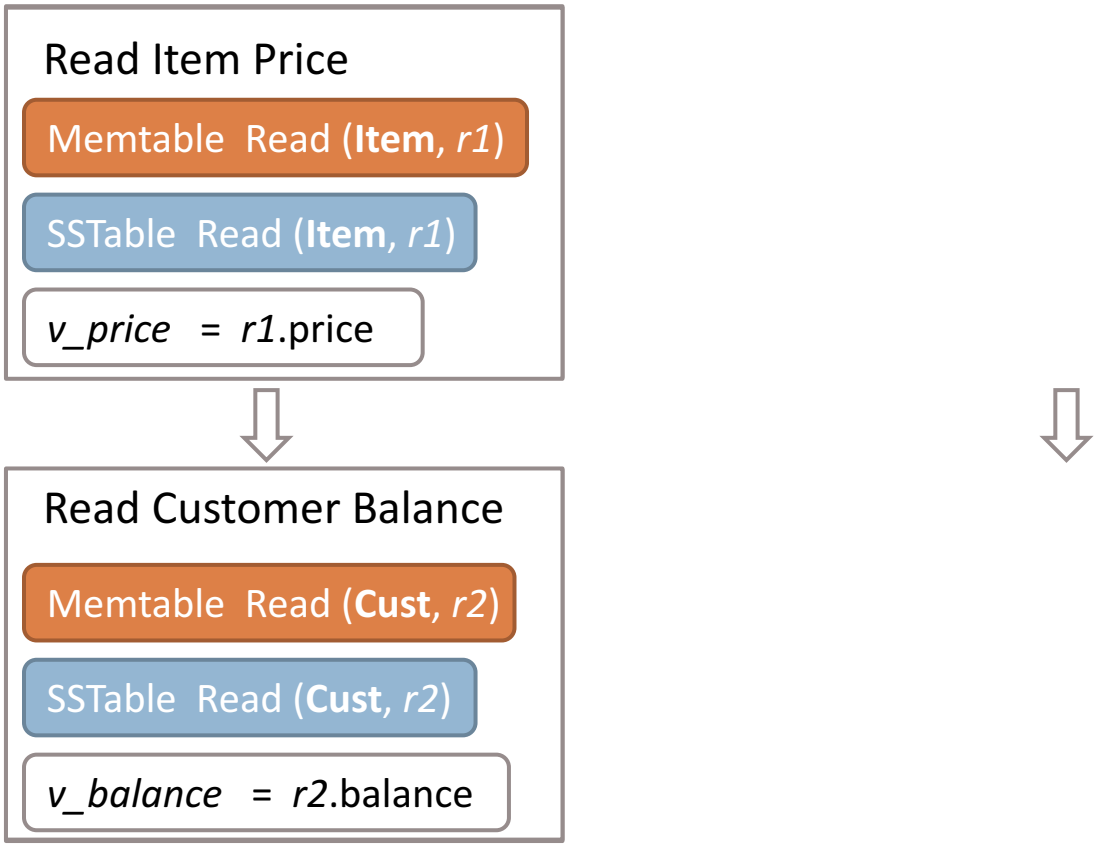
```
v_price     = Read (Item, id = 1,  price);

v_balance = Read (Cust,  id = 5,  balance);
```



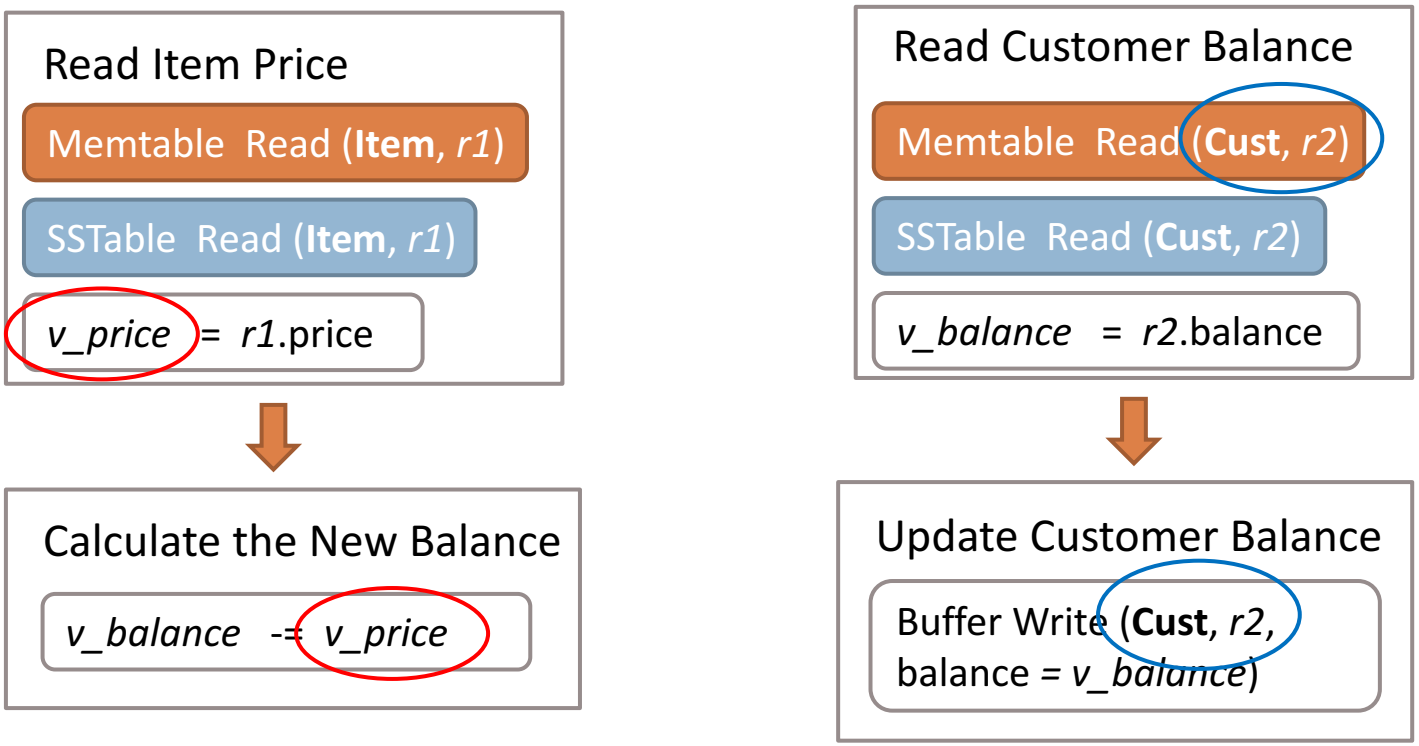
Transaction Compilation

- Reorder ops w/o data dependency does not change semantics



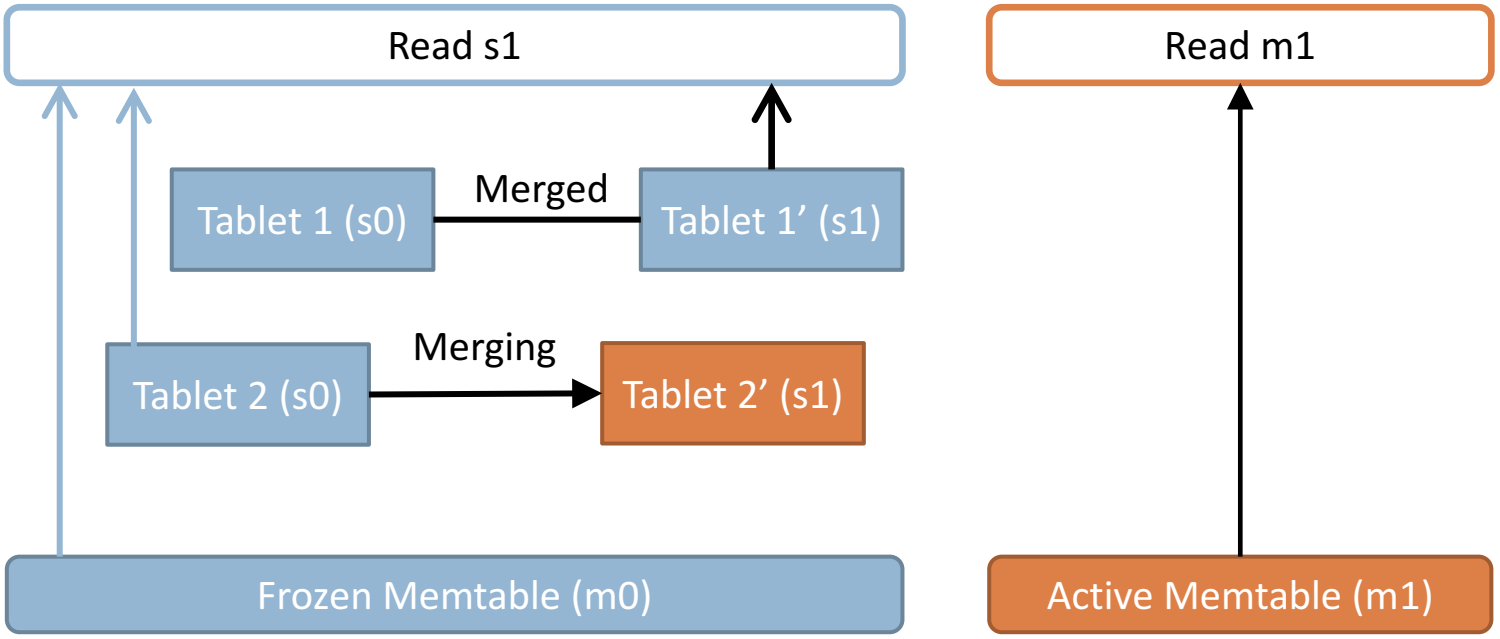
Transaction Compilation

- Only ops w/ data dependencies cannot be reordered
  - Use the same variable, and one is write (identify by variable name)
  - Use the same record, and one is write (identify by table name)



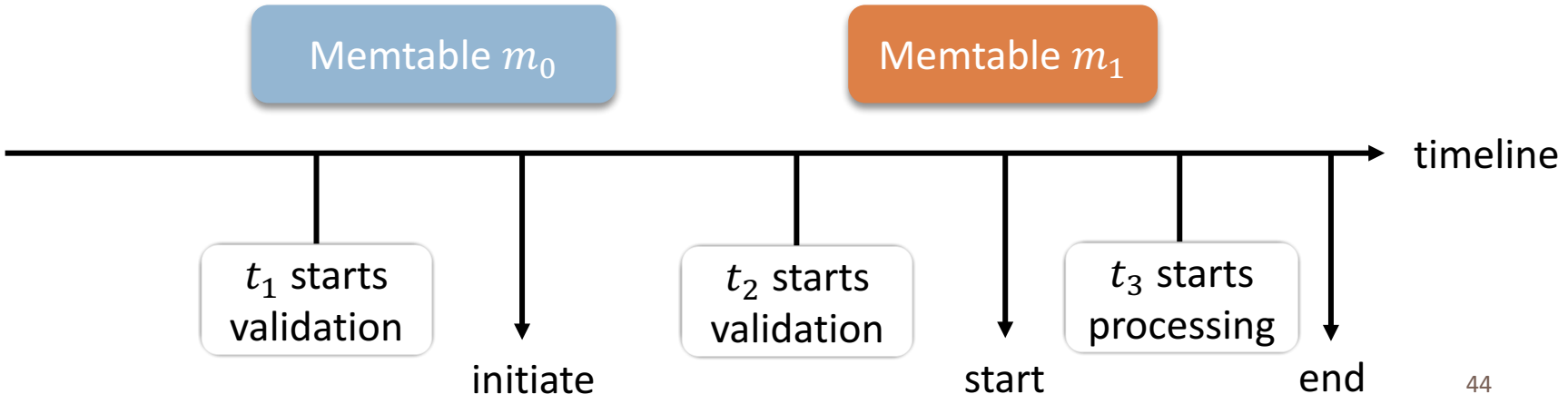
Data Access During Compaction

- MemTable Read: always read the new MemTable m_1
- SSTable Read
  - Merged data ($Tablet\ 1$): read from s_1
  - Merging data ($Tablet\ 2$): read from s_0 and the frozen Memtable m_0



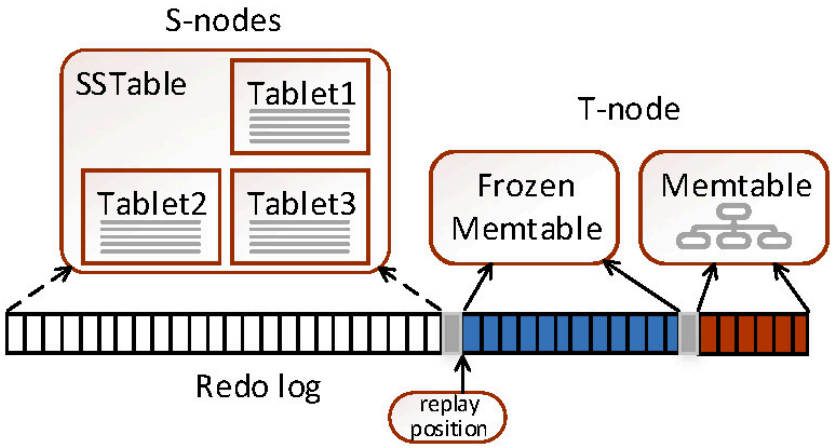
Snapshot Isolation During Data Compaction

- Classify transactions into three types:
  - Type 1: start validation before the compaction is initialized
    - validate on m_0, write on m_0
  - Type 2: start validation after the compaction is initialized
    - validate on m_0 and m_1, write on m_1
  - Type 3: starts processing after the compaction is started
    - validate on m_1, write on m_1



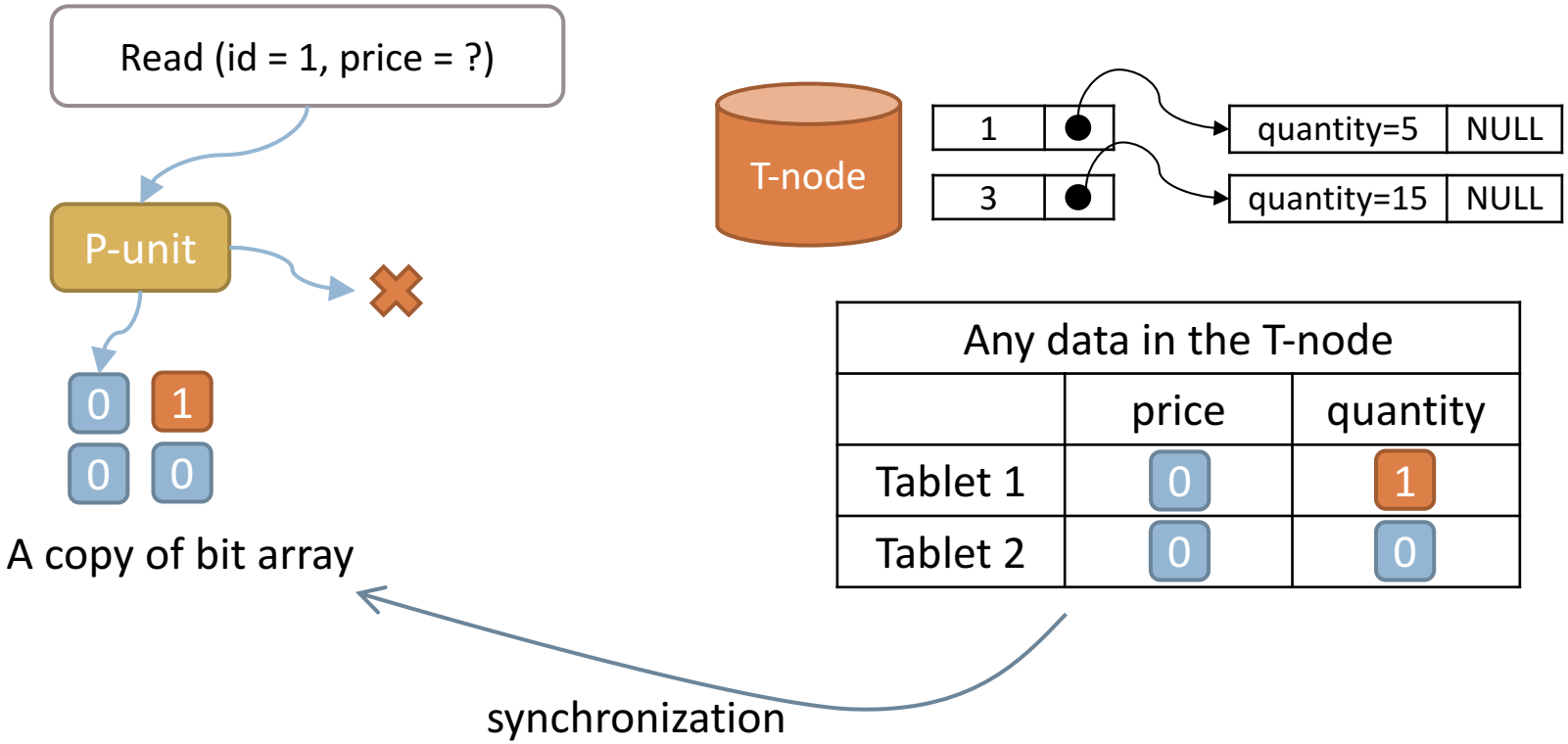
Recovery during Data Compaction (DC)

- Compaction start log entry (CSLE)
  - Persist when the DC is started
  - Acts as a border of redo log entries
- Compaction end log entry (CELE)
  - Persist when the DC is ended
  - Save the position of the CSLE of the DC
- Recovery procedure
  - Read CELE to find the position of CSLE
  - Replay the redo log from CSLE
  - At first, replay data into m_0
  - Once CSLE is encountered, repay data into m_1



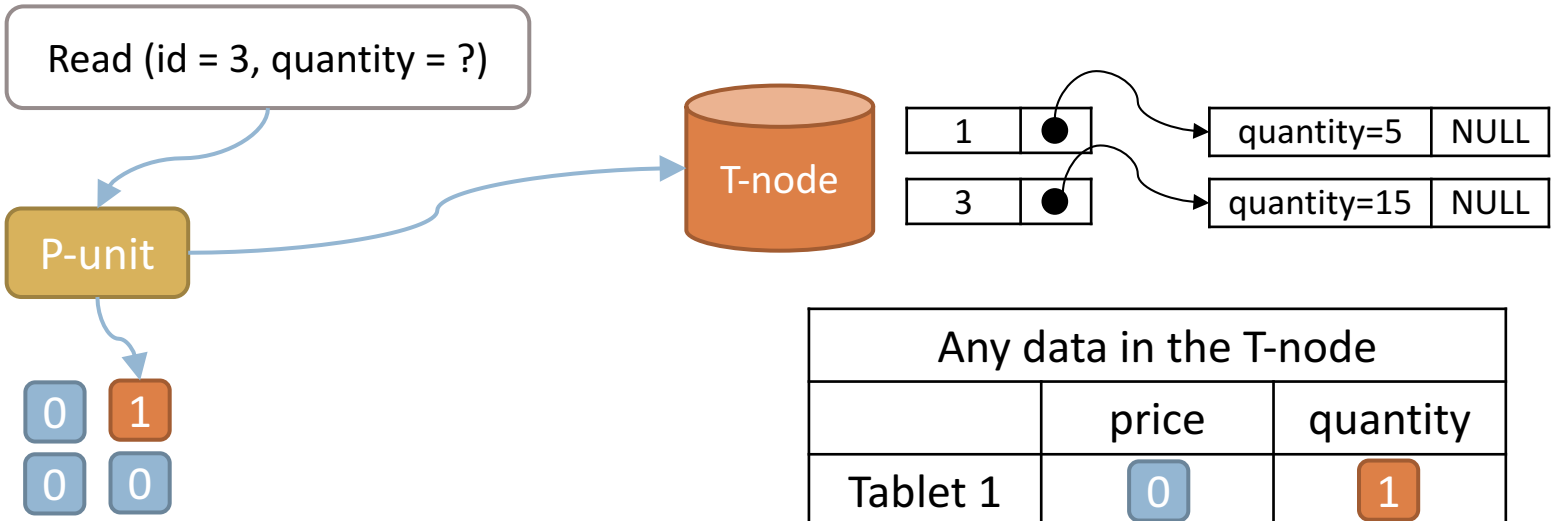
Asynchronous Bit Array

- Synchronization & usage
  - Periodically synchronized to P-units
  - A P-unit check its local copy to filter useless T-node access



Asynchronous Bit Array

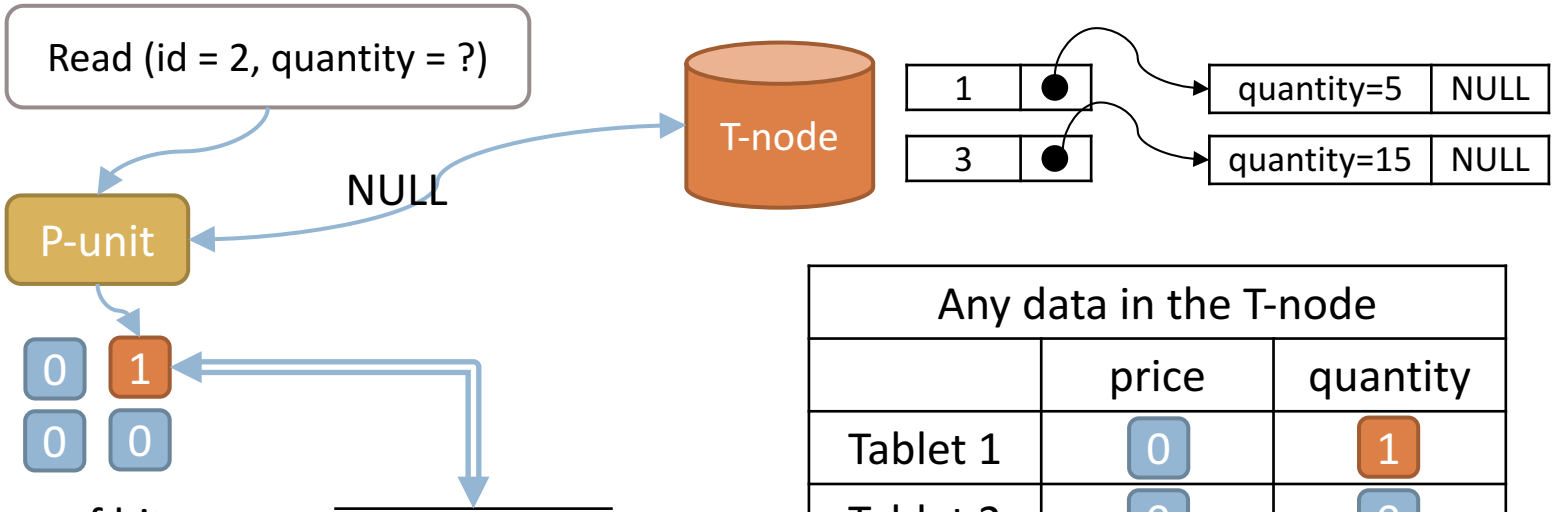
- Synchronization & usage
  - Periodically synchronized to P-units
  - A P-unit check its local copy to filter useless T-node access



Any data in the T-node		
	price	quantity
Tablet 1	0	1
Tablet 2	0	0

Asynchronous Bit Array

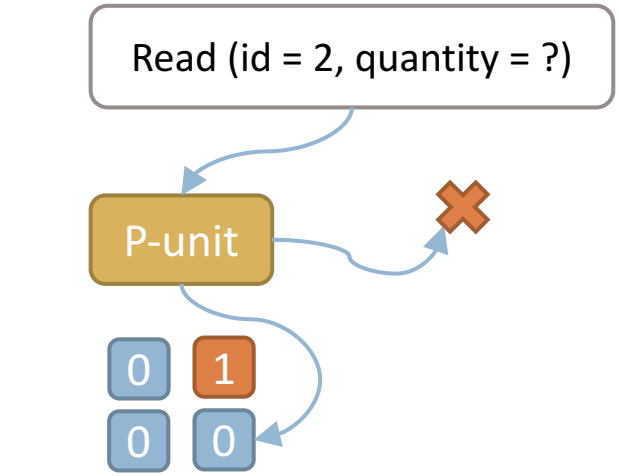
- False positive
  - (row_x, col_y) does not exist on the T-node, but the bit array says yes
    - An empty read
  - Reason: bit array maintained at tablet granularity



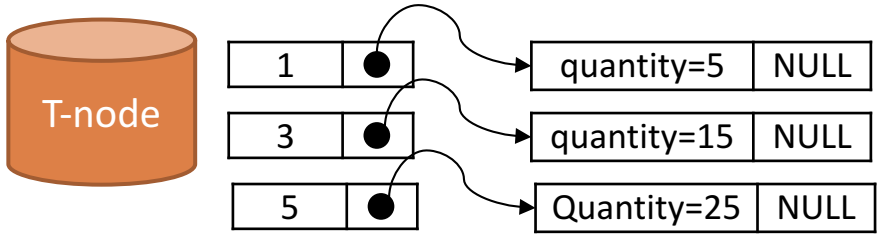
Any data in the T-node		
	price	quantity
Tablet 1	0	1
Tablet 2	0	0

Asynchronous Bit Array

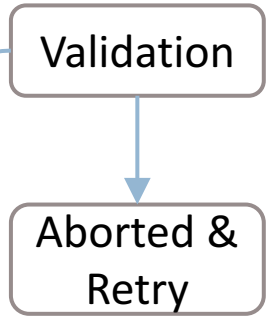
- False negative
  - A bit array copy may fall behind the latest version
  - (row_x, col_y) exists on the T-node, but the bit array says no
  - Transaction re-check all potential empty reads in the validation phase



A copy of bit array

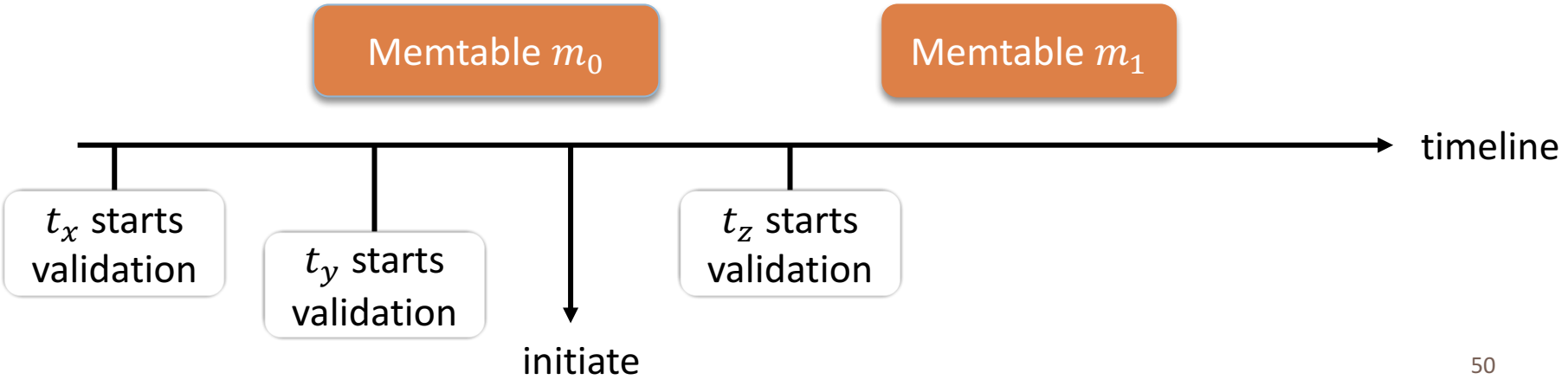


Any data in the T-node		
	price	quantity
Tablet 1	0	1
Tablet 2	0	1



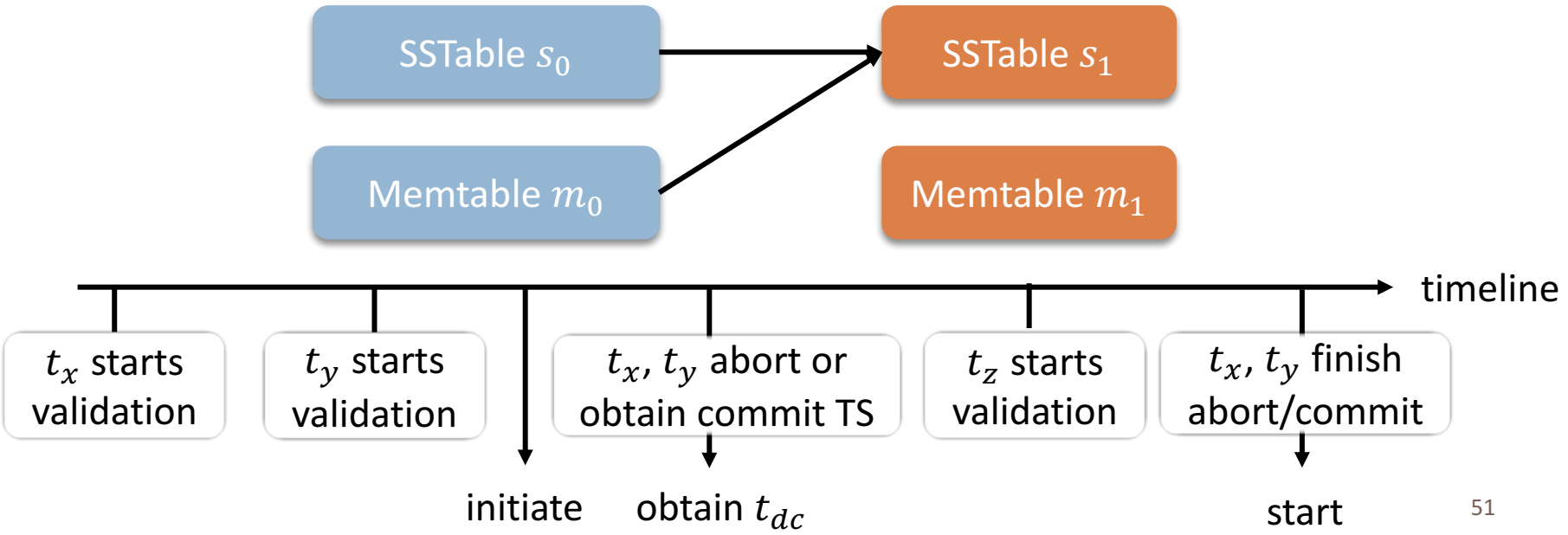
Data Compaction

- Initiate
  - Create a new Memtable
  - Freeze the current Memtable
  - Handling ongoing transactions
    - Case 1: validation starts before the compaction is initiated
      - t_x and t_y are allowed to write data into m_0
    - Case 2: validation starts after the compaction is initiated
      - t_z will write data into m_1 after the data compaction is started



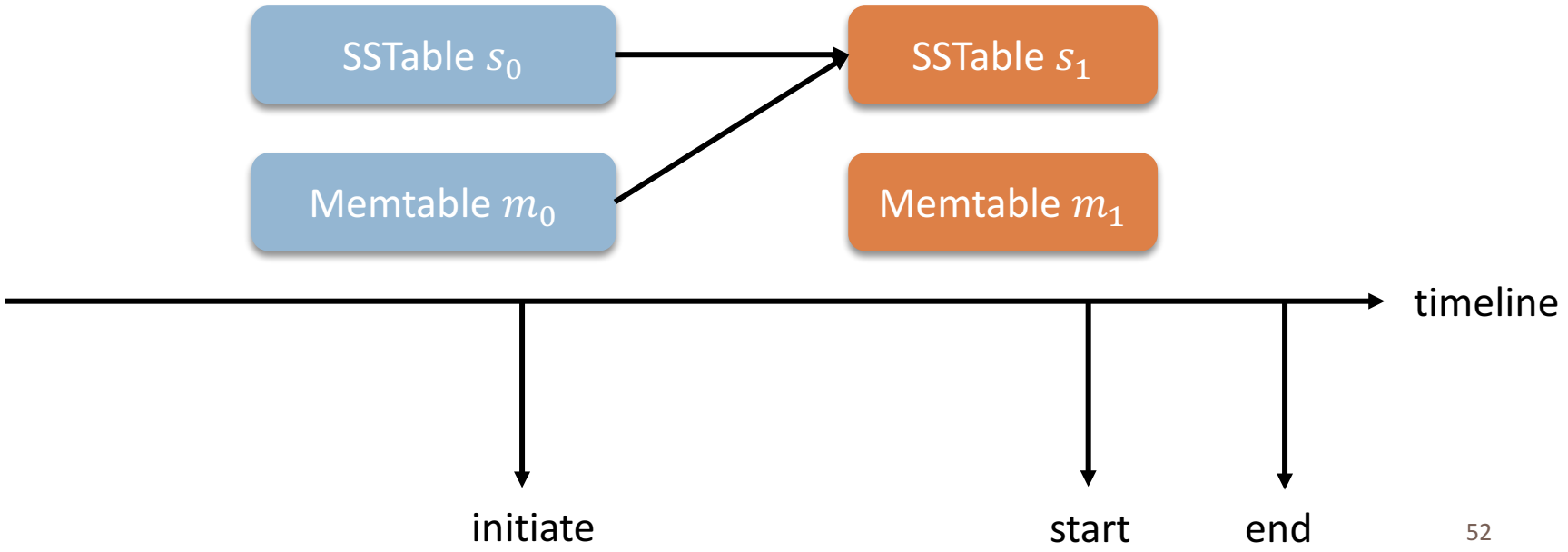
Data Compaction

- Start
  - Get compaction timestamp t_{dc} after t_x and t_y abort or obtain commit TS
    - t_z starts validation only after t_{dc} is obtained
  - Start data compaction after t_x and t_y finish abort/commit
    - Create a new SSTable by merging the old one and the frozen Memtable



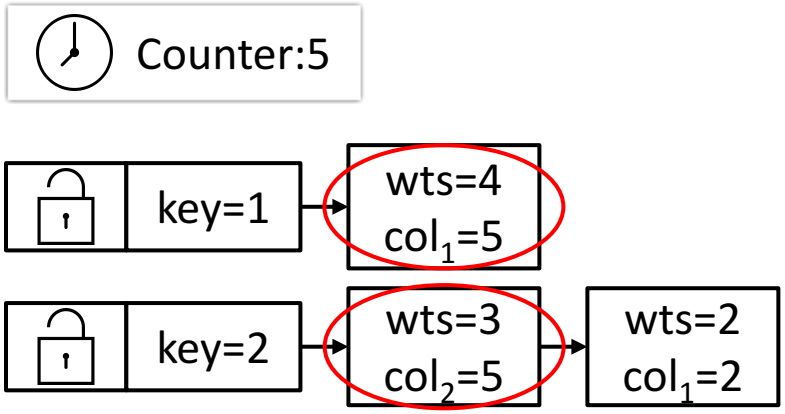
Data Compaction

- End
  - Wait until the s_1 is fully created
  - Release the old Memtable and SSTable



Concurrency Control

- Data structures on the T-node
  - A timestamp counter(MVCC)
  - Row-level latch (OCC)
- Start
  - Acquire read-timestamp rts
- Process
  - Read latest version specified by rts

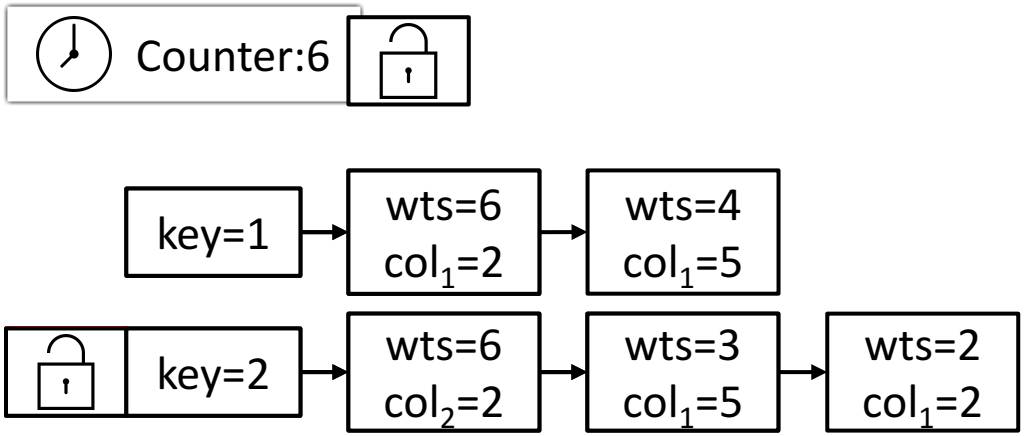


Txn t_x

read-timestamp: $rts = 5$

Concurrency Control

- Commit
  - Acquire latches for records in the write set
  - Verify there is no newer version
  - Acquire write timestamp wts
  - Write and release latches



Txn t_x
read-timestamp: $rts = 5$
write-timestamp: $wts = 6$