VLDB 2025
WELCOME TO LONDON

Concurrency Control as a Service (CCaaS)

Research Track – Distributed Transaction I

Weixing Zhou, *Yanfeng Zhang, Xinji Zhou, Zhiyou Wang, Zeshun Peng,*

Yang Ren, Sihao Li, Huanchen Zhang, Guoliang Li, and Ge Yu.

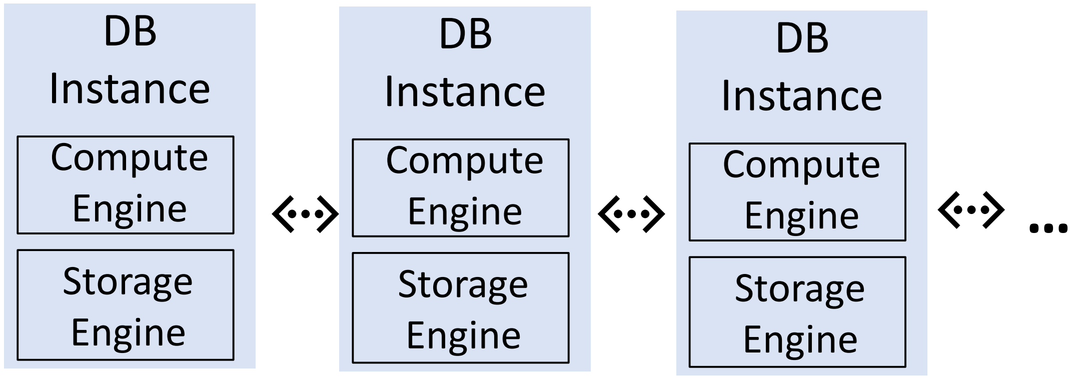
Northeastern University, China

Huawei Technology Co., Ltd

Tsinghua University, China

Databases are evolving to Cloud-Native Arc.

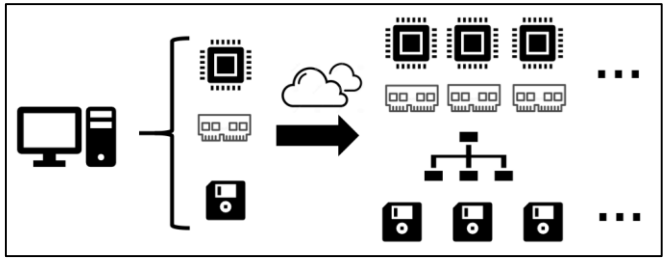
- **Compute-Storage disaggregated two-layer architecture**

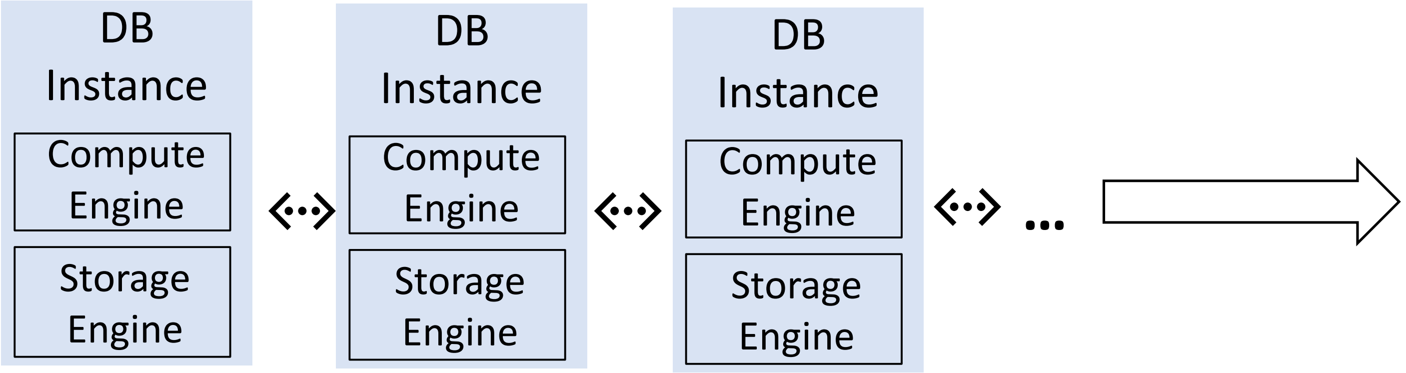


Distributed Database System Architecture

Databases are evolving to Cloud-Native Arc.

- **Compute-Storage disaggregated two-layer architecture**

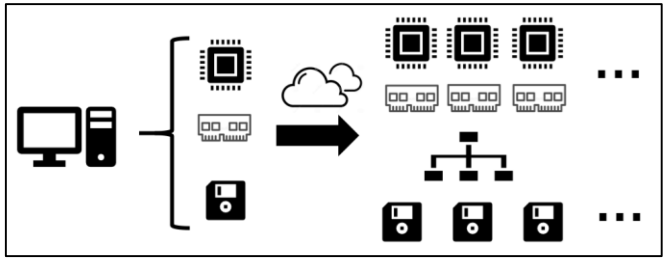


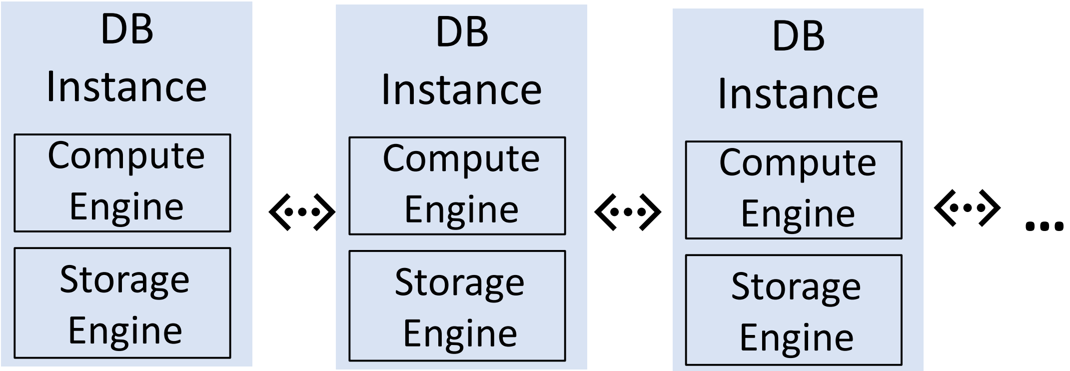


Distributed Database System Architecture

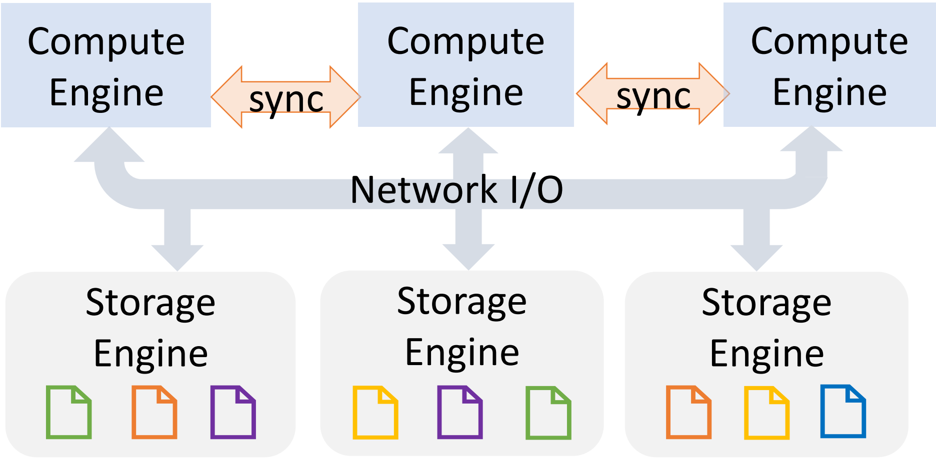
Databases are evolving to Cloud-Native Arc.

- **Compute-Storage disaggregated two-layer architecture**





Distributed Database System Architecture



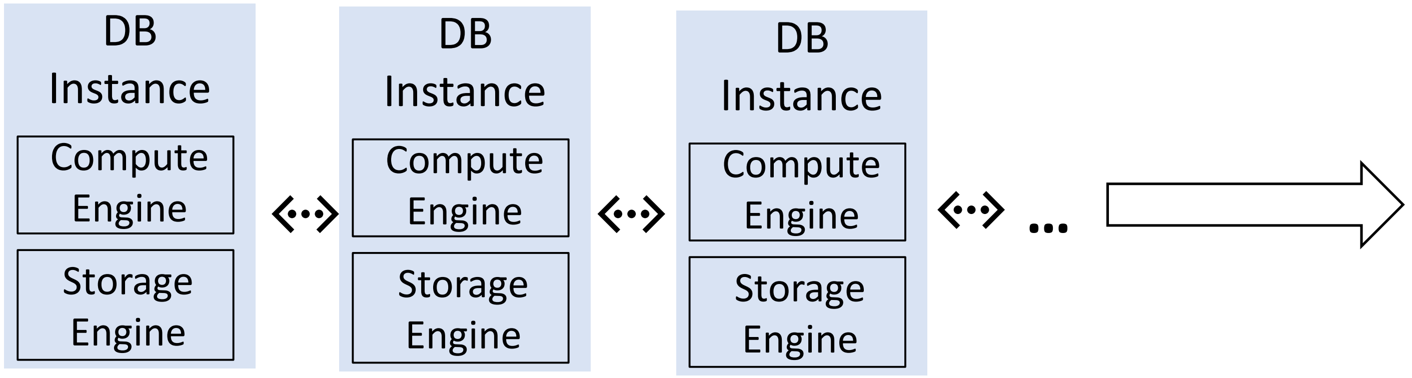
Compute-Storage disaggregated Architecture

Databases are evolving to Cloud-Native Arc.

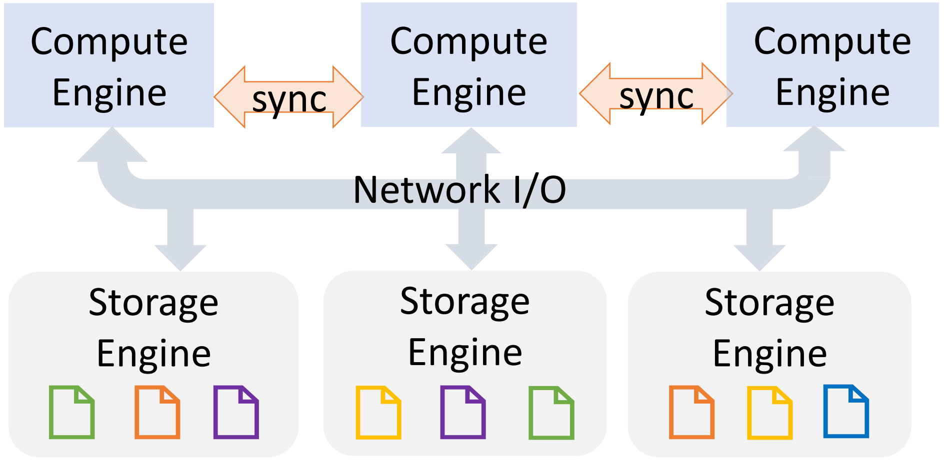
- **Compute-Storage disaggregated two-layer architecture**

➢ **Benefits**:

- Scalability and Elasticity
- Cost-effectiveness
- High Availability



Distributed Database System Architecture



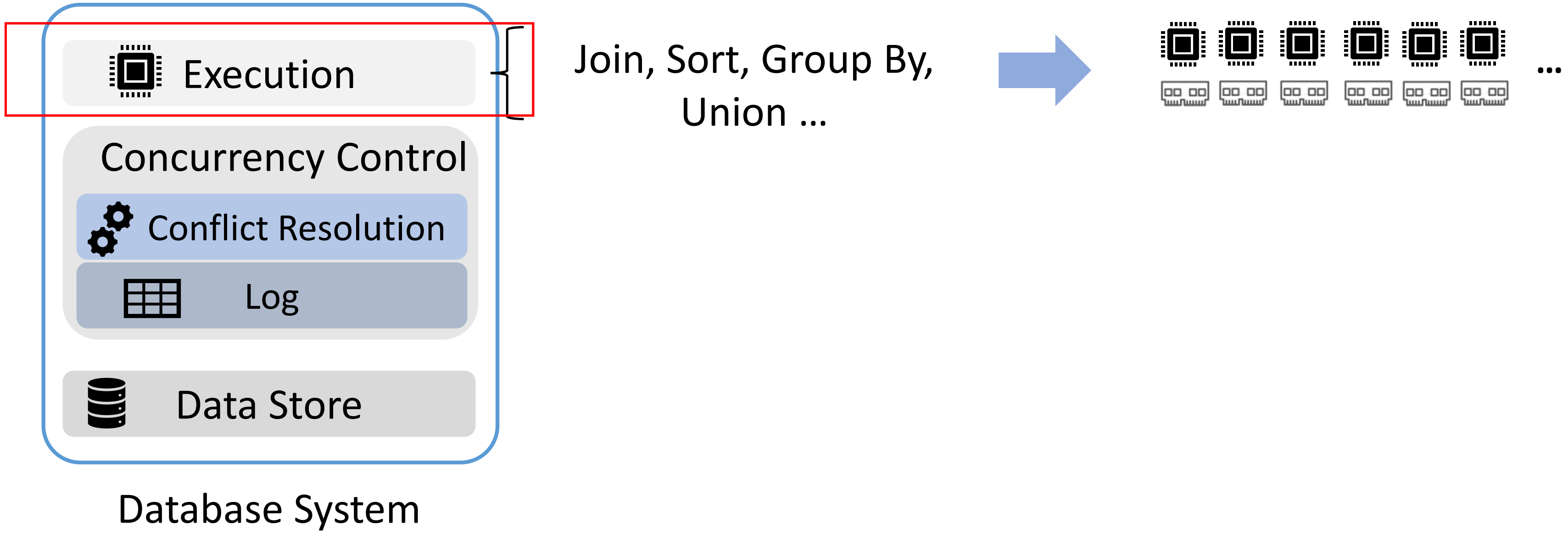
Compute-Storage disaggregated Architecture

Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**

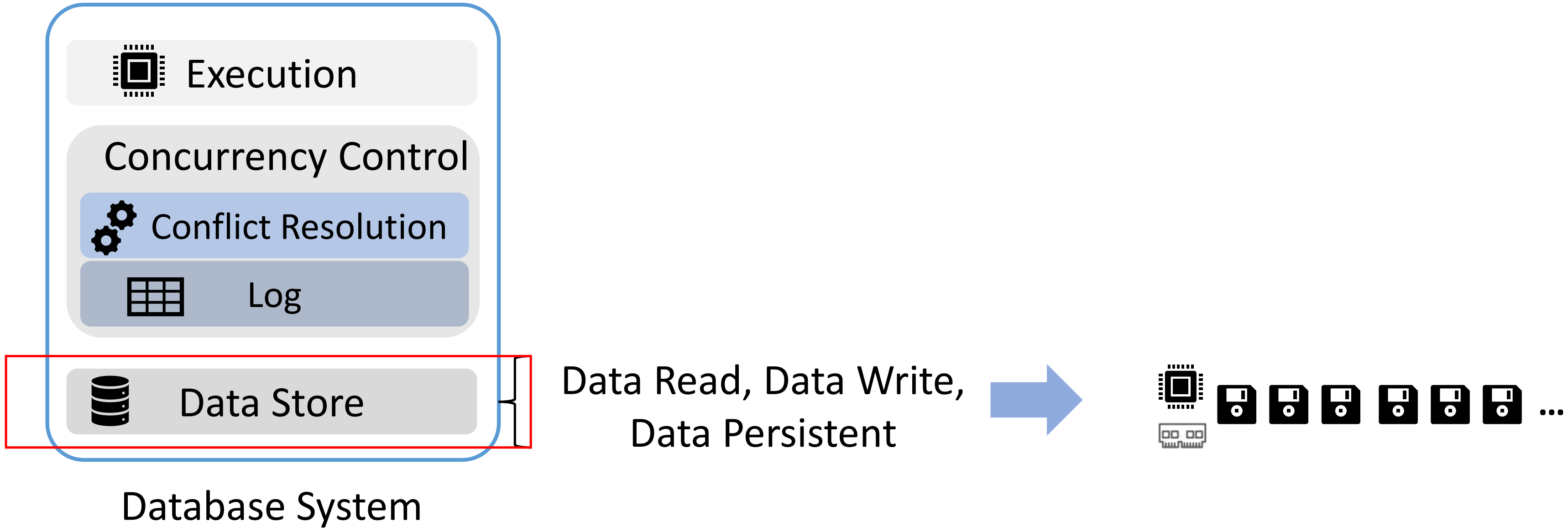
Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**
  - Execution: High computation tasks, free scale



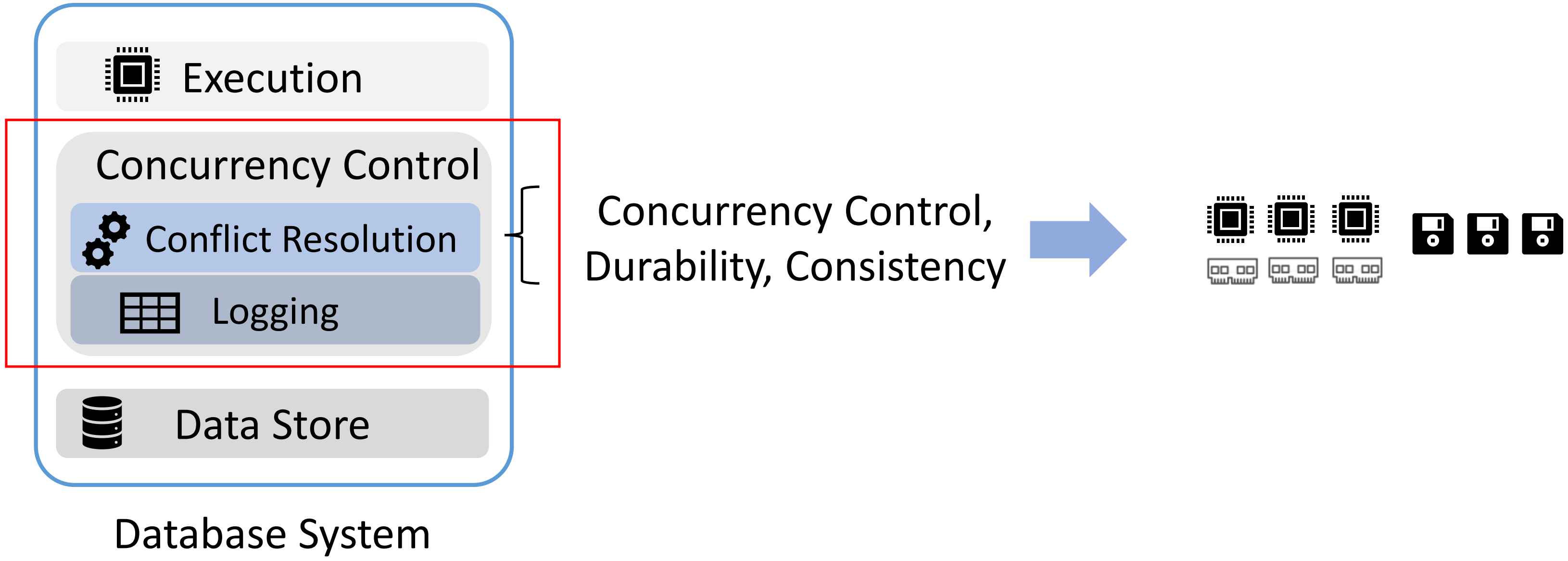
Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**
  - Execution: High computation tasks, free scale
  - Data Storage: Large storage space



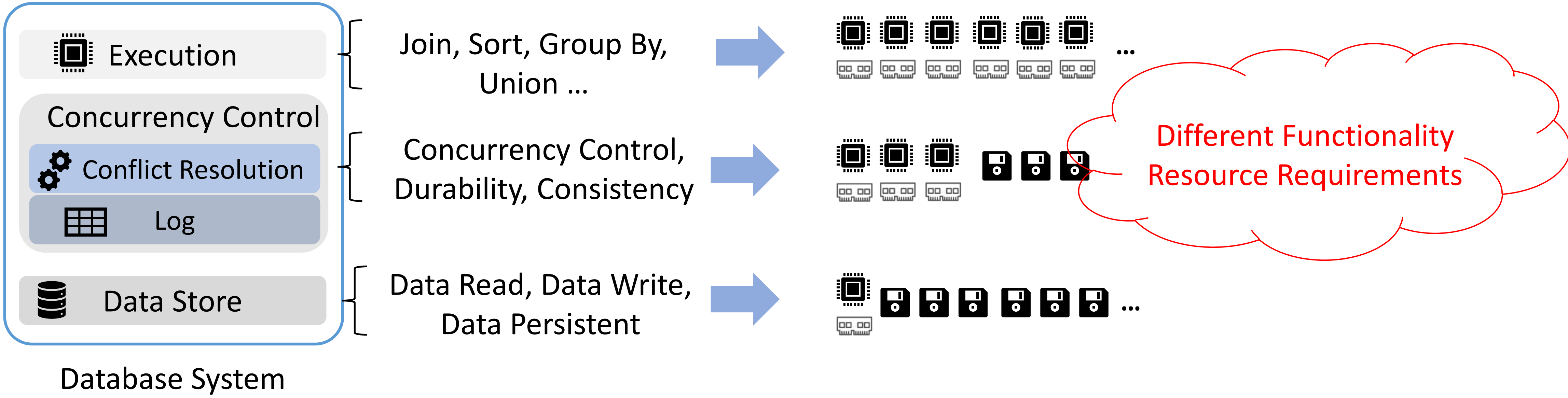
Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**
  - Execution: high computation tasks, free scale
  - Data Storage: large storage space
  - Concurrency Control: high concurrency, data consistency, log durability



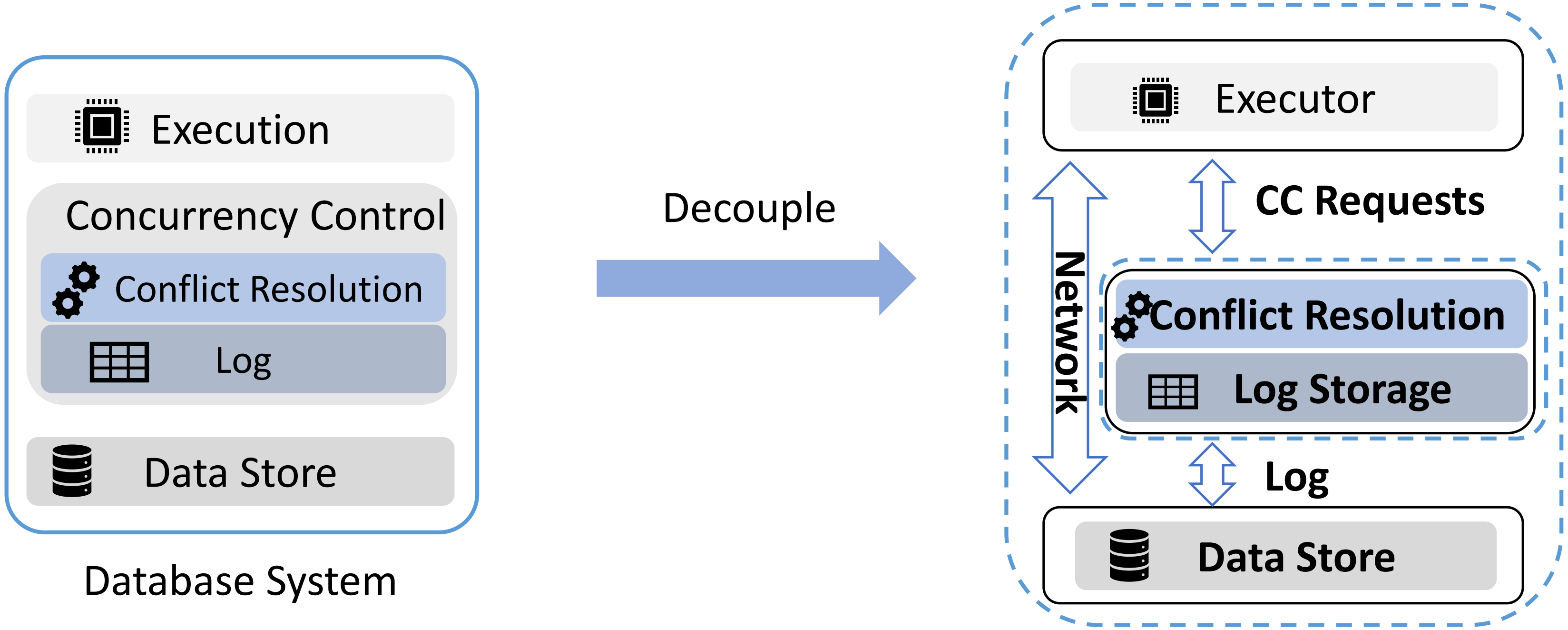
Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**
  - Execution: high computation tasks, free scale
  - Data Storage: large storage space
  - Concurrency Control: high concurrency, data consistency, log durability



Decoupling, the spirit of Cloud-Native Arc.

- **Each module has specific functionality and distinct resource requirements**
  - Decoupling based on functions and resource requirements
    - *fully utilize the **heterogeneous resources***, avoid resource wasting.
    - scale each module independently, high flexibility.

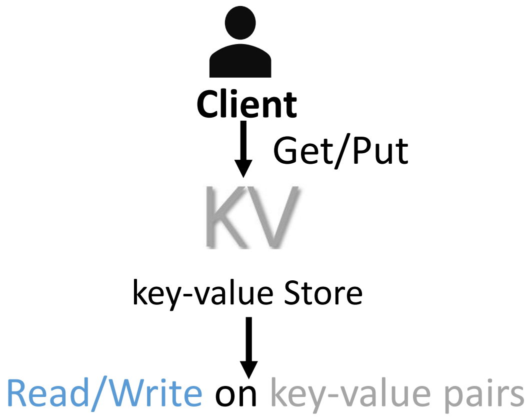


The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**

The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**

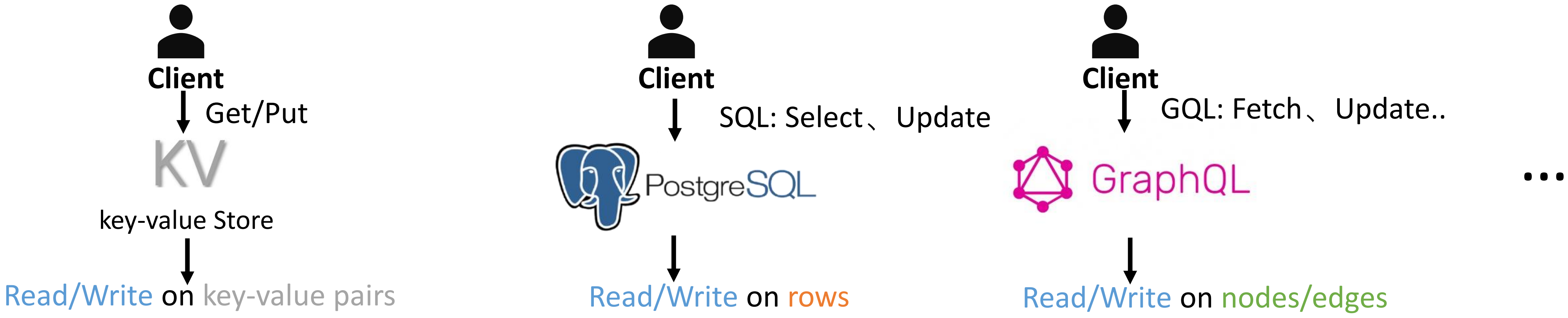


The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**

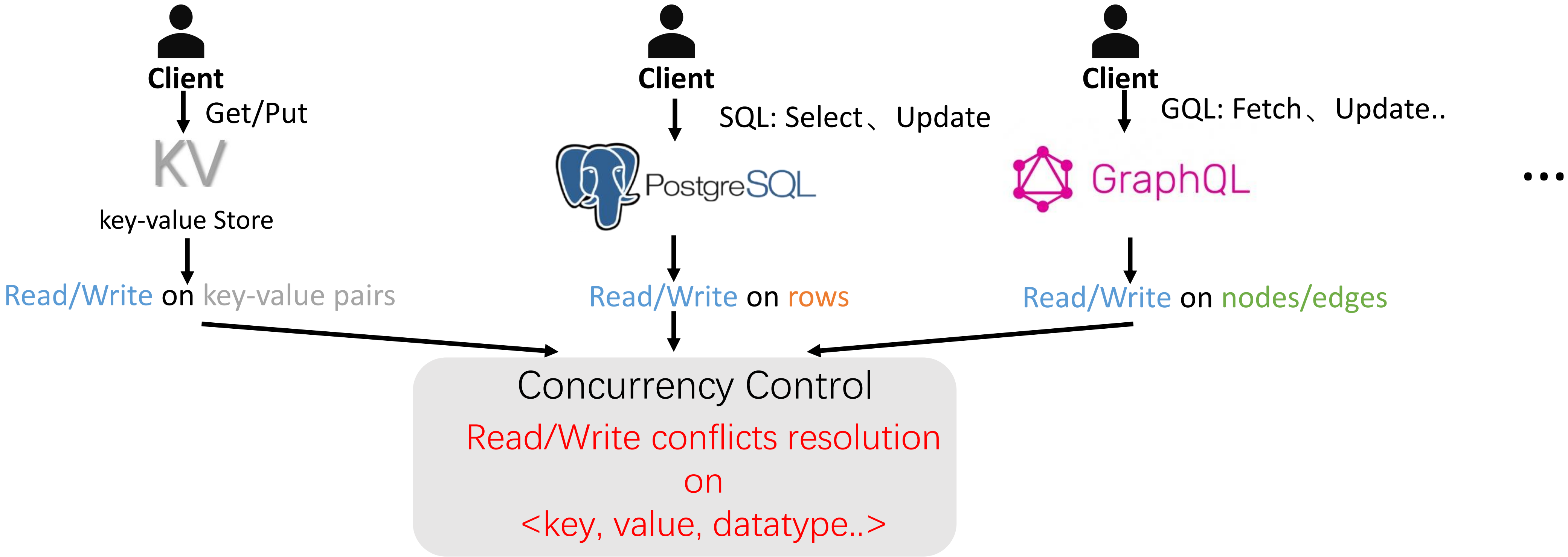
The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**



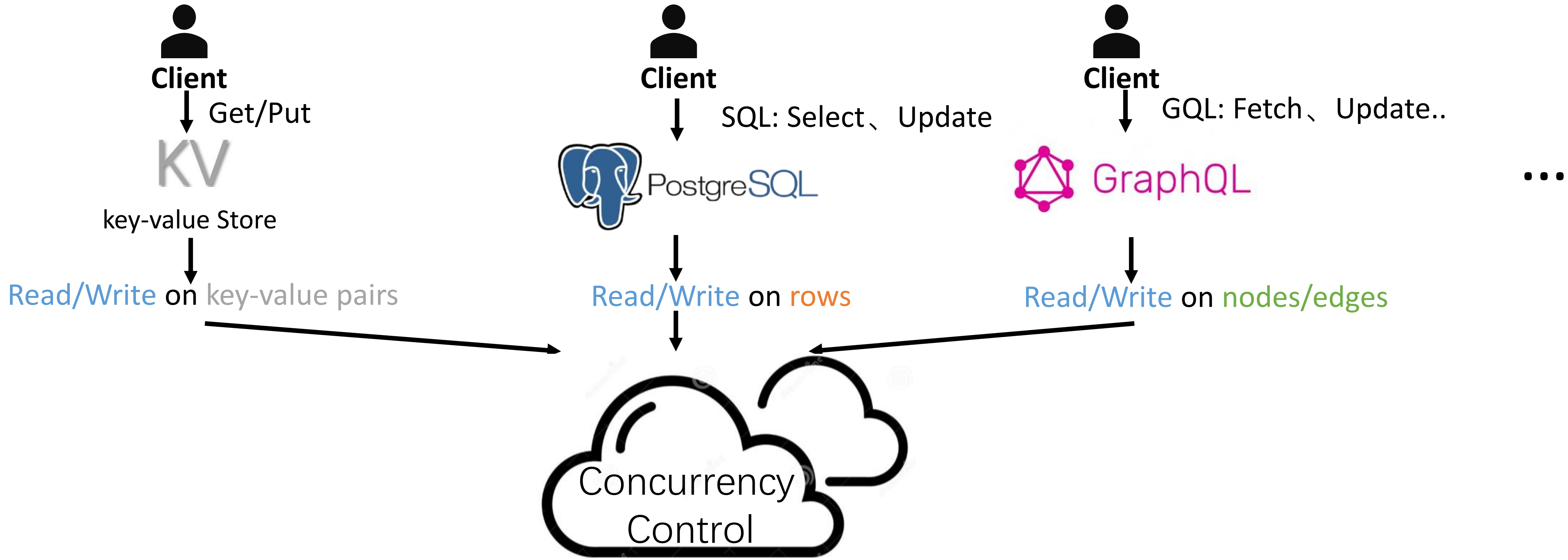
The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**
  - The core of Concurrency Control is handling concurrent read/write conflicts



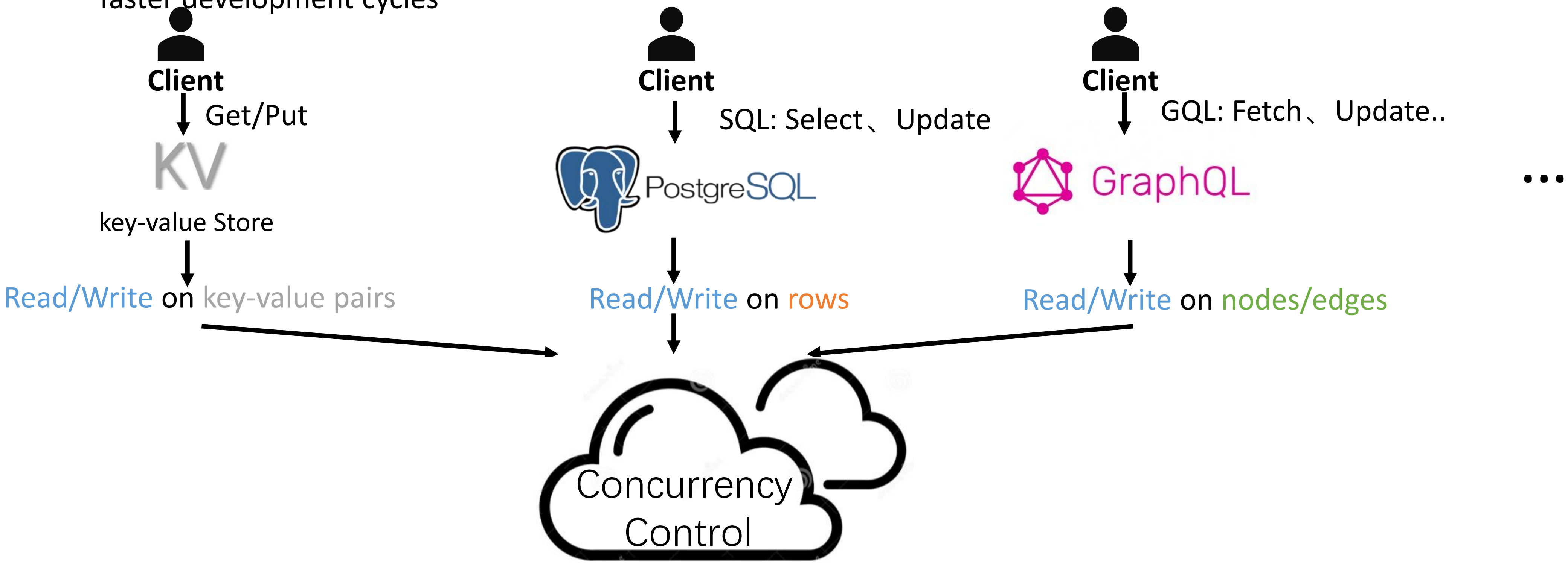
The Core of CC is handling conflicts

- **Databases with different data models process different types of requests:**
  - The core of Concurrency Control is handling concurrent read/write conflicts
  - Decoupled function modules → independent **services**



The Core of CC is handling conflicts

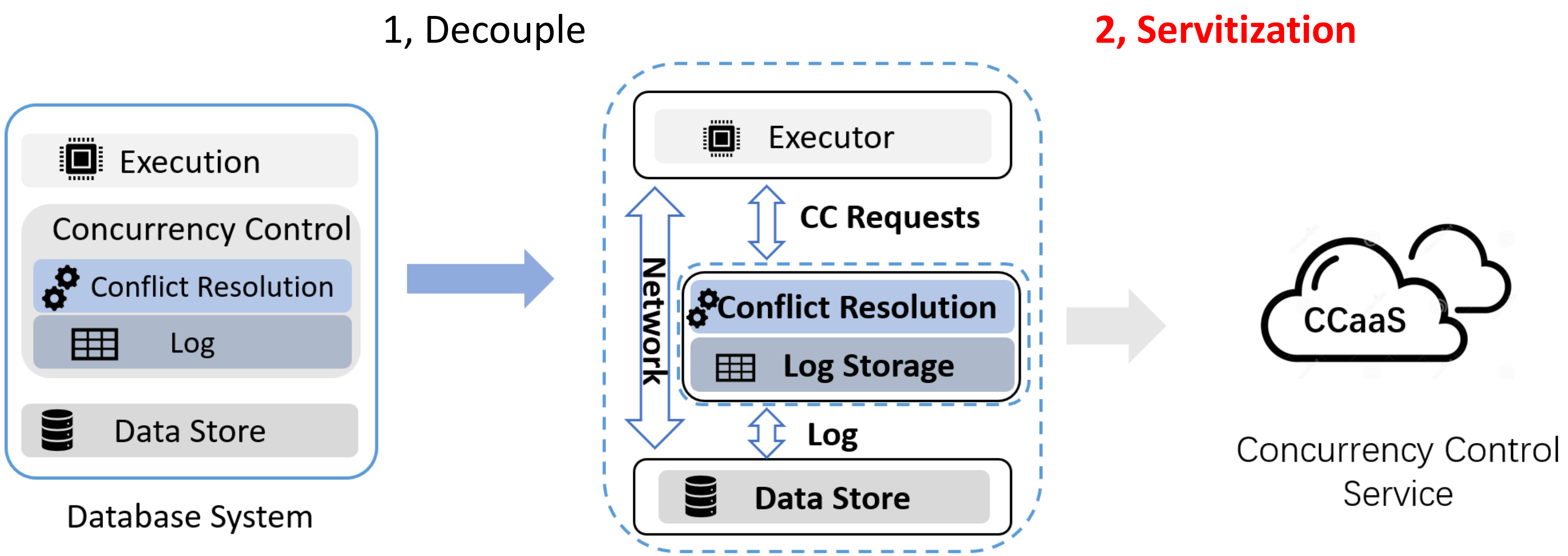
- **Databases with different data models process different types of requests:**
  - The core of Concurrency Control is handling concurrent read/write conflicts
  - Decoupled function modules → independent **services**
    - Improved agility, different services can be reused easily and adapt quickly to changing business needs, enabling faster development cycles



Concurrency Control as a Service (CCaaS)

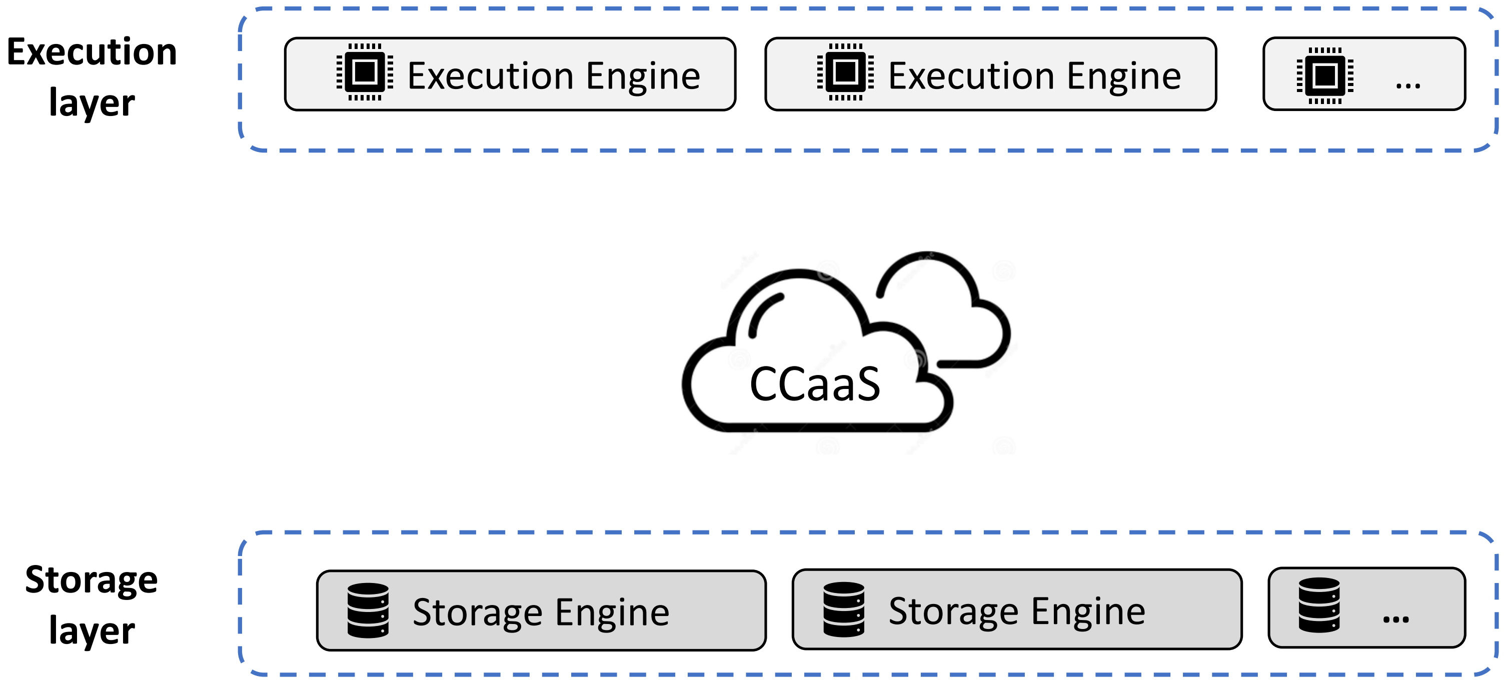
- **Decouple** the concurrency control module
  high scalability and *resource utilization*

- Move a step forward: **concurrency control service**
  development *agility*, different databases can *reuse easily*



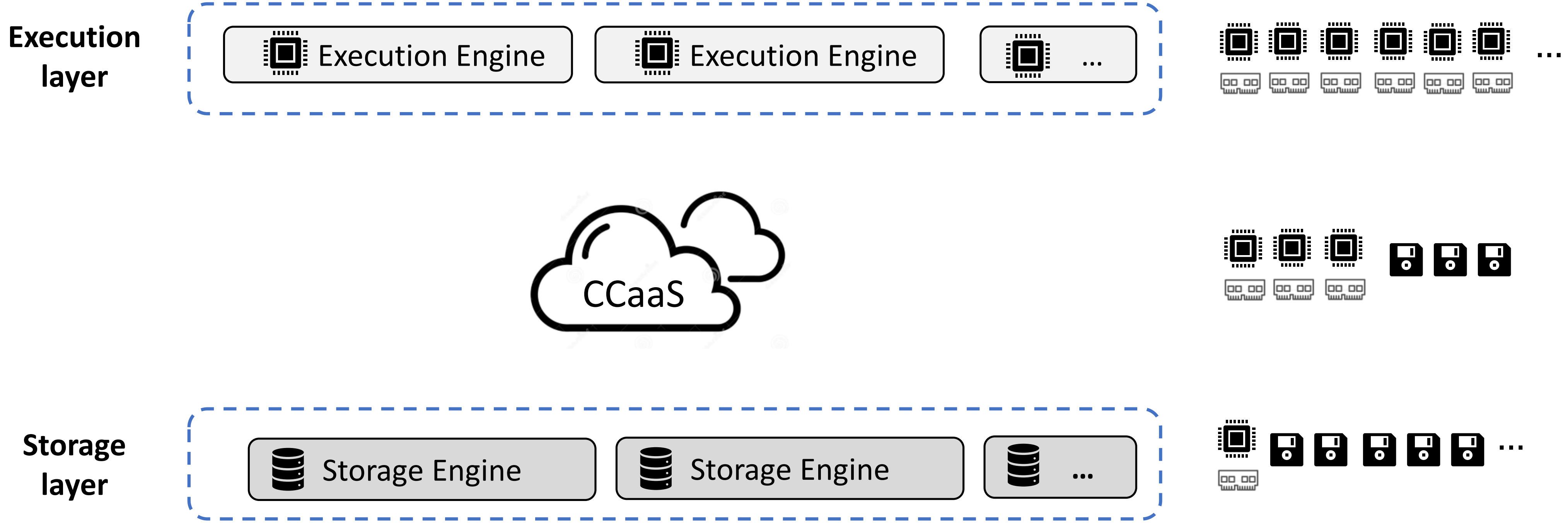
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture



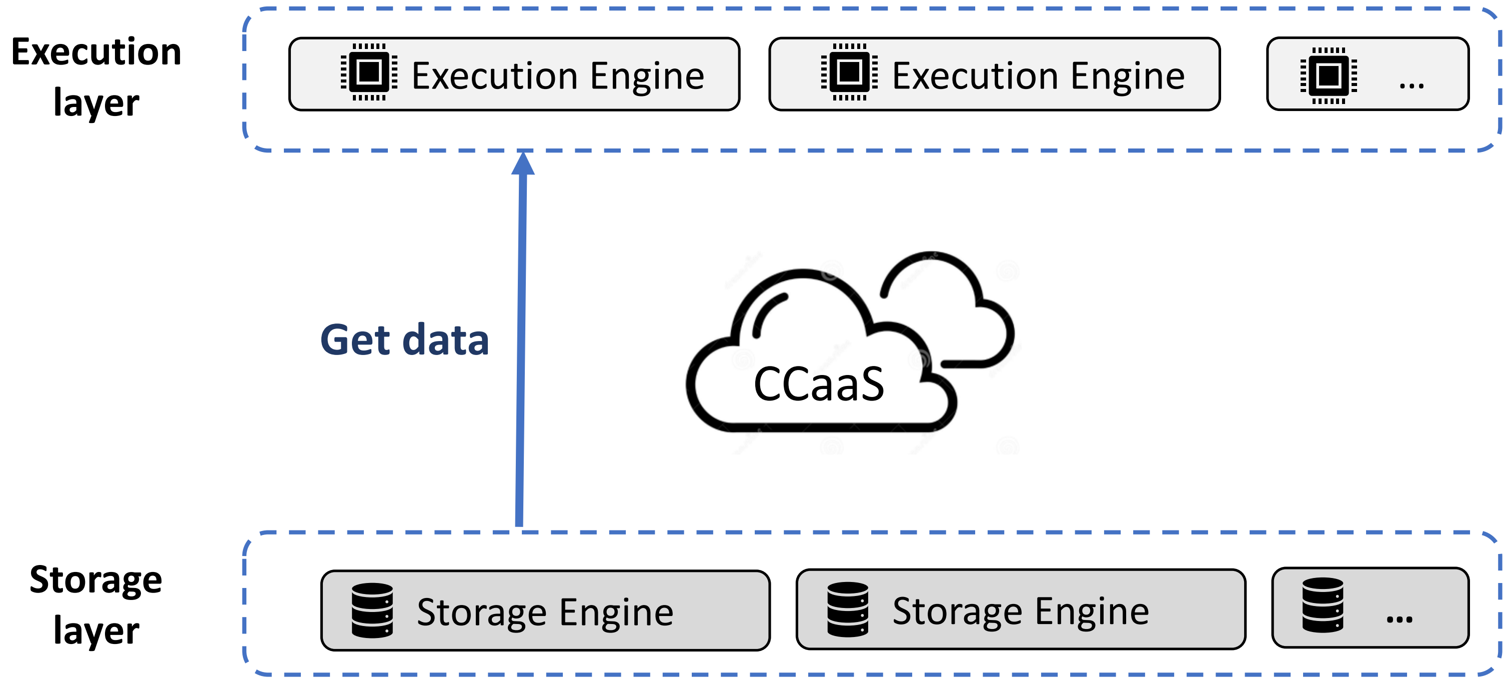
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*



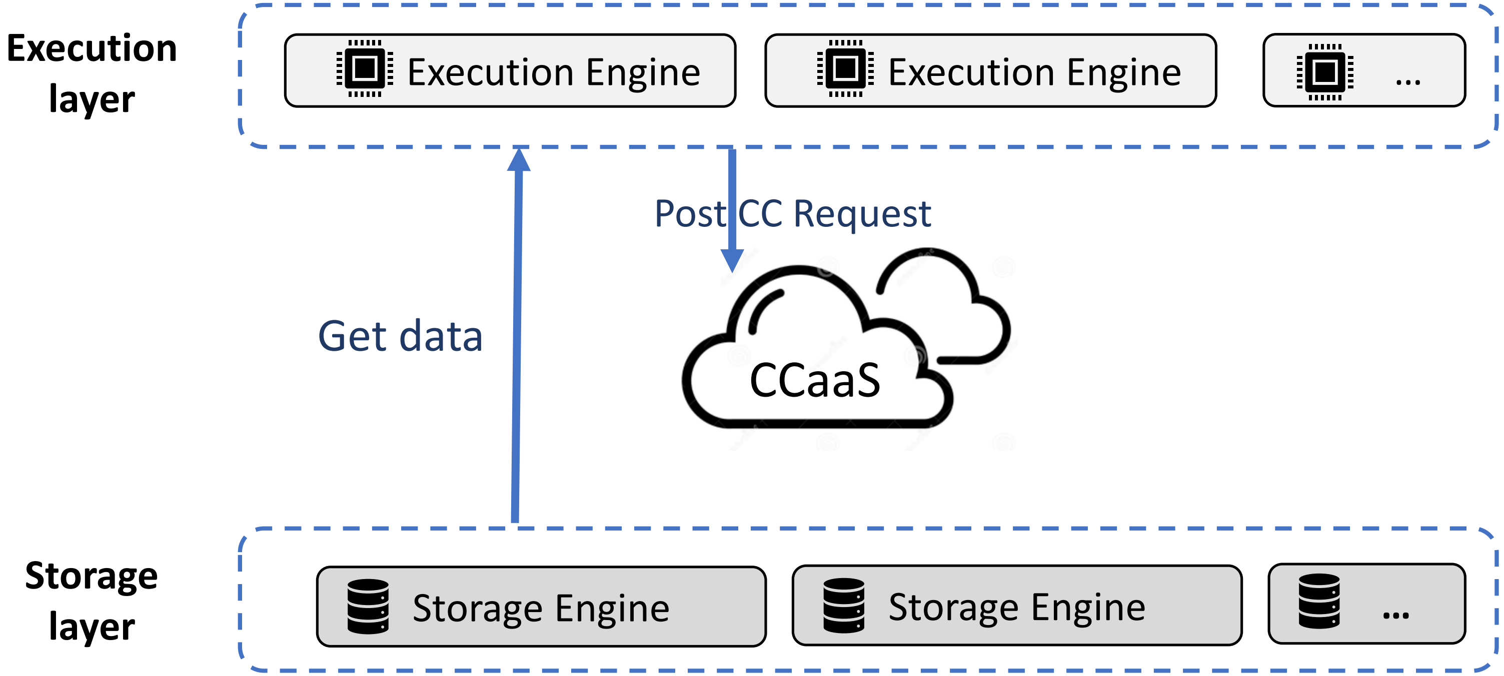
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*



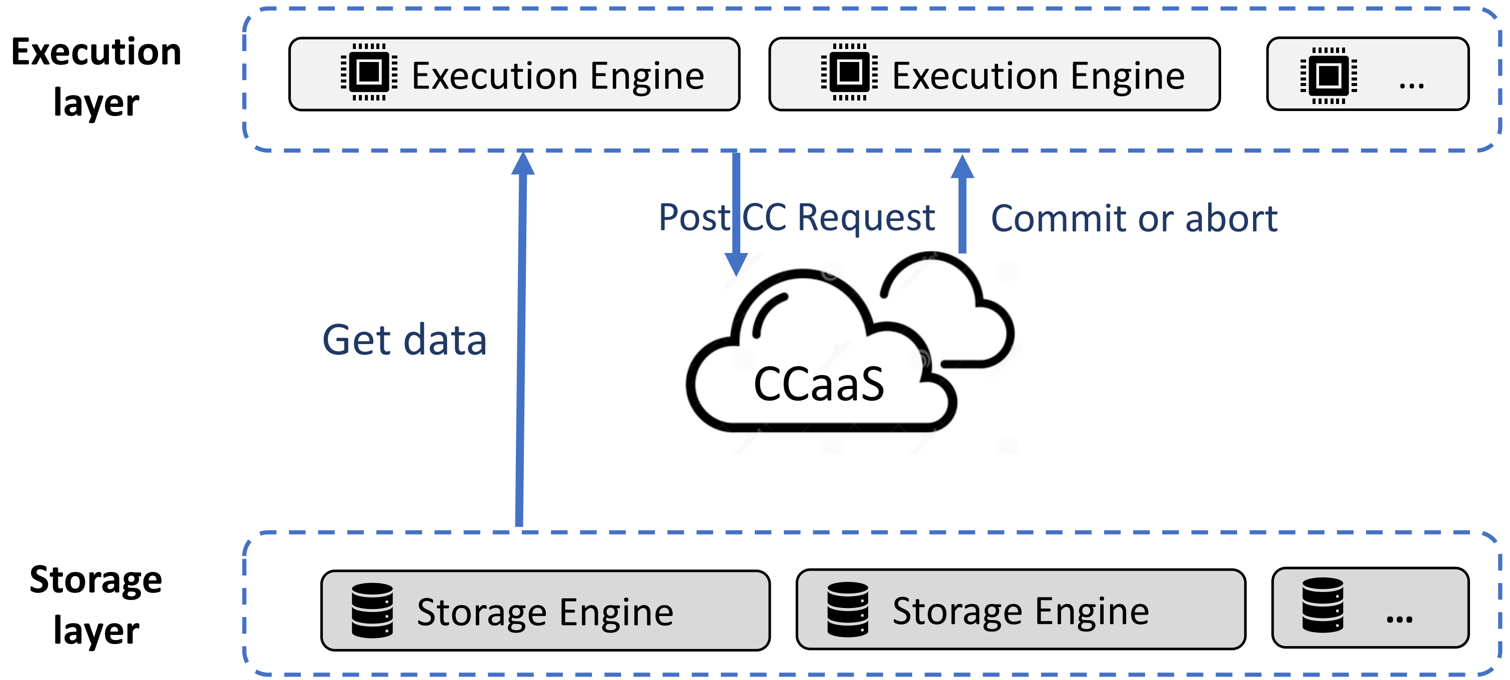
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*



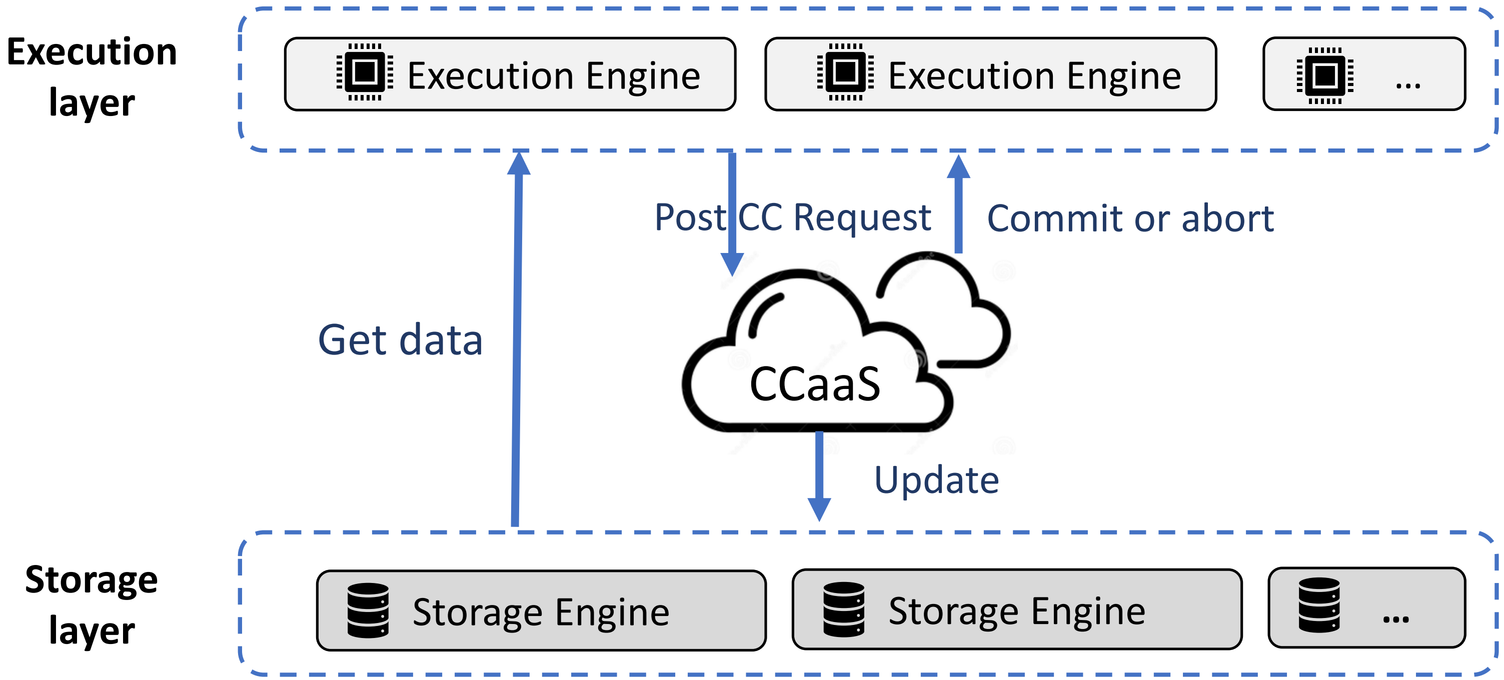
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*



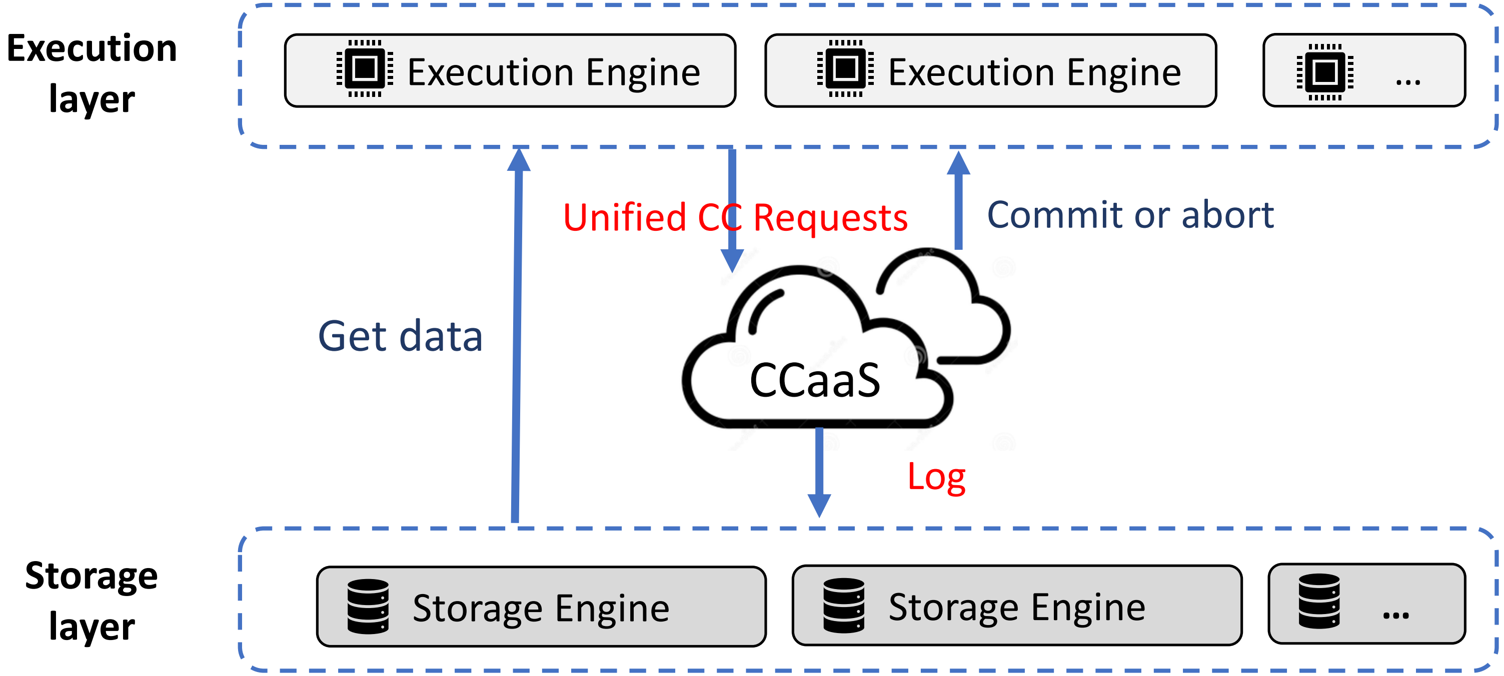
Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*



Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*
  - **concurrency control service**



Interfaces for the execution layer:

Begin(ExecutionInfo)

TxnCommit(*TxnID, RS, WS*)

Lock(*TxnID, Key[ ], Value[ ] ,OpType[ ]*)

Commit(*TxnID*)

Abort(*TxnID*)

Interfaces for the storage layer:

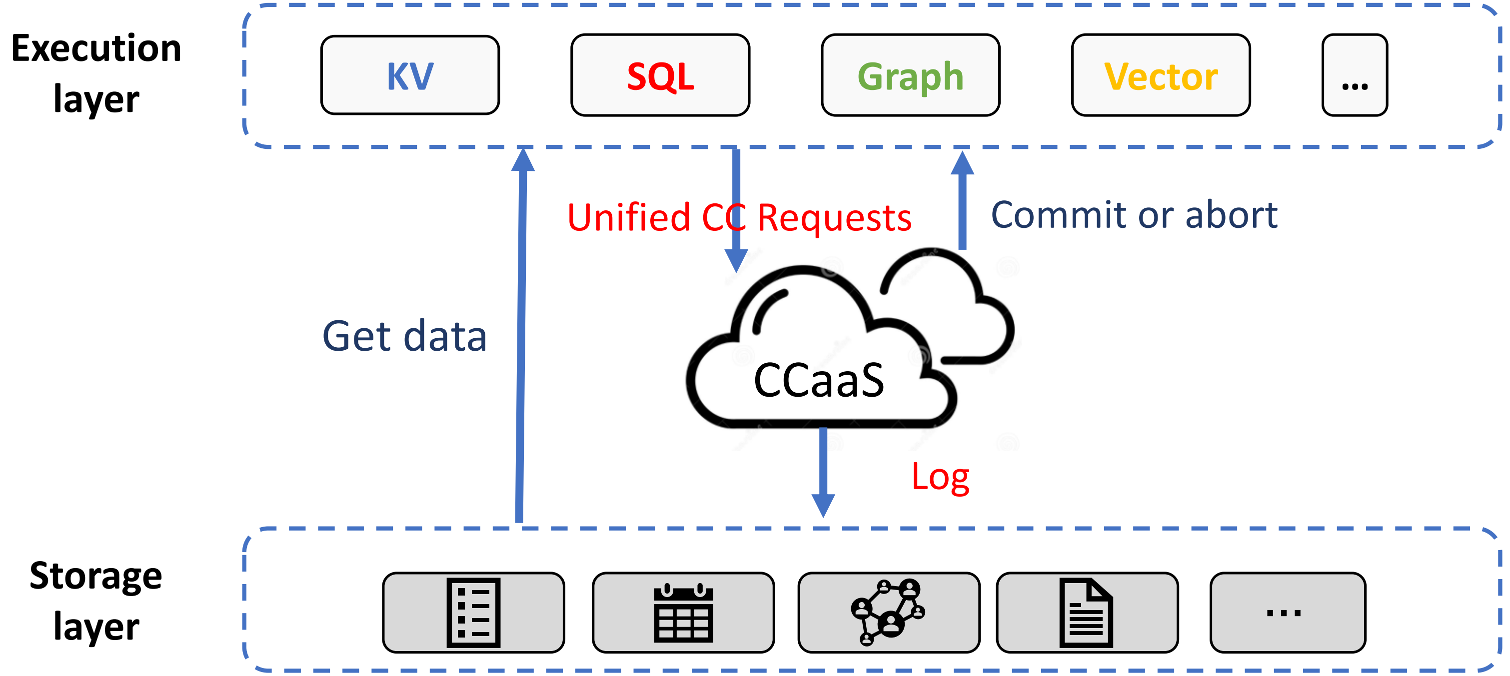
LogPush(LogAdaptorID)

LogPull(LSN[])

CCaaS Interfaces

Concurrency Control as a Service (CCaaS)

- **Decoupled concurrency control module**
  - **Execution-Concurrency Control-Storage** three layer architecture
    - *high scalability* and *resource utilization*
  - **concurrency control service**
    - development *agility*, different databases can *reuse easily*



Interfaces for the execution layer:

Begin(ExecutionInfo)

TxnCommit(*TxnID, RS, WS*)

Lock(*TxnID, Key[ ], Value[ ],OpType[ ]*)

Commit(*TxnID*)

Abort(*TxnID*)

Interfaces for the storage layer:

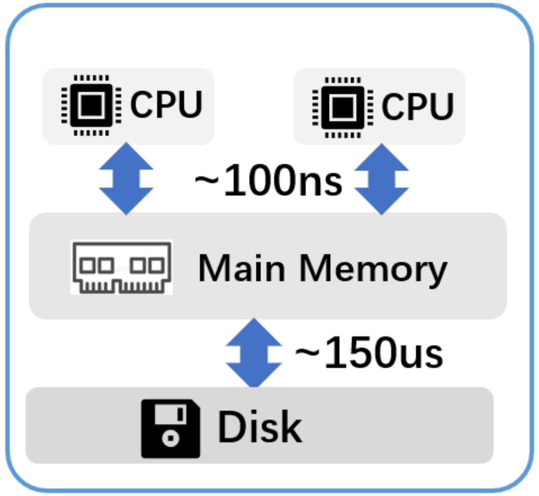
LogPush(LogAdaptorID)

LogPull(LSN[])

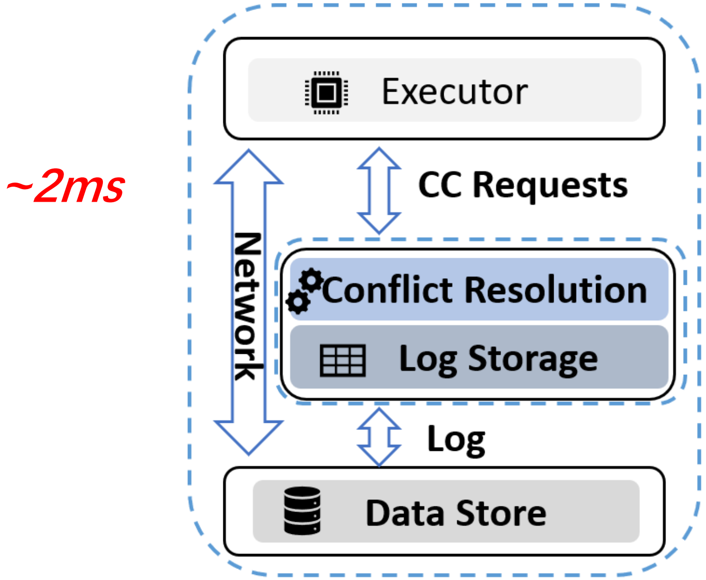
CCaaS Interfaces

Problem1: Decoupled CC brings more Network Communication

- **Decoupled CC increasing latency**
  - *Communication with hardware VS communication with network.*

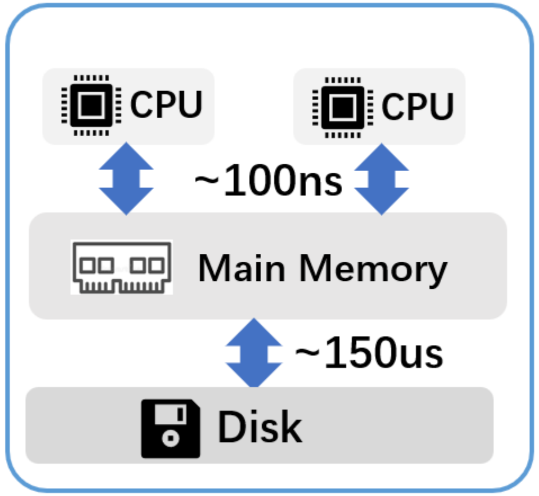


VS

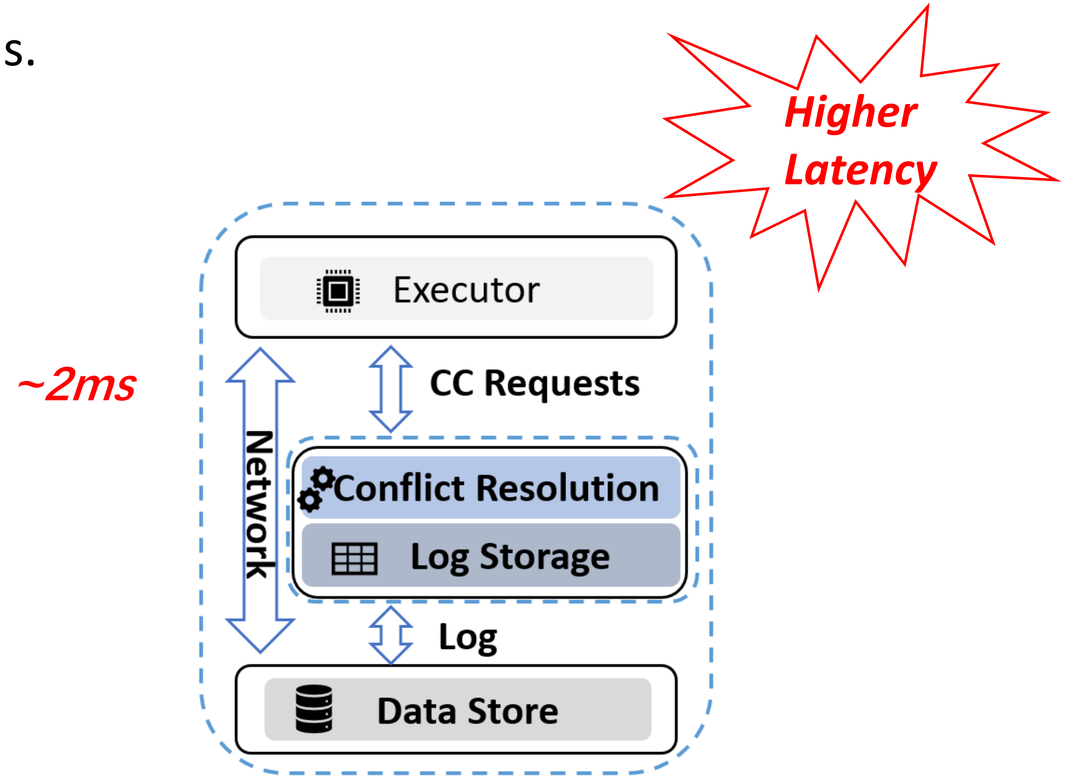


Problem1: Decoupled CC brings more Network Communication

- **Decoupled CC increasing latency**
  - *Communication with hardware VS communication with network.*
  - **Higher latency**, not suitable for OLTP workloads.

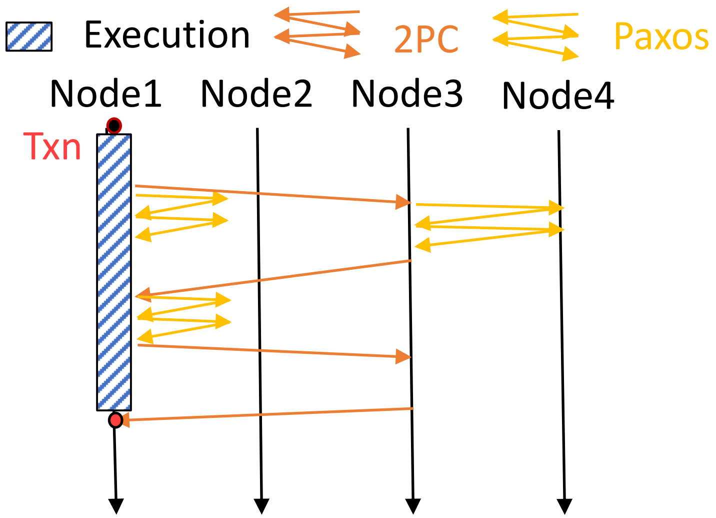


VS



Problem2: High Communication in CC

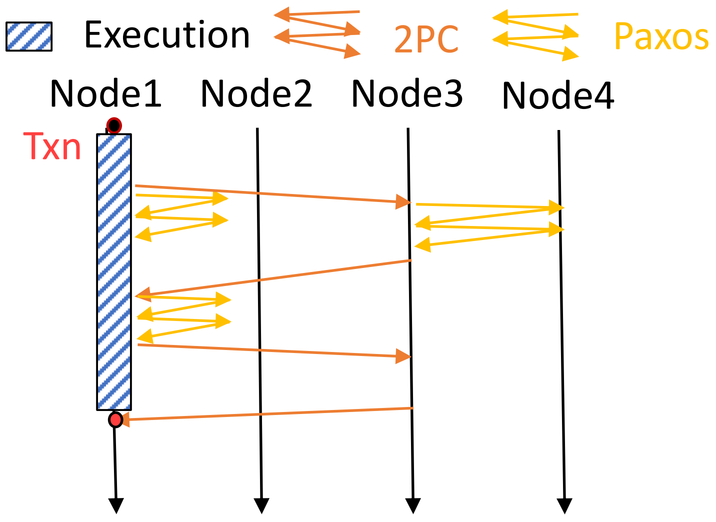
- **Decoupled CC increasing latency**
  - *Communication with hardware VS communication with network.*
  - **Higher latency**, not suitable for OLTP workloads.
- **High Coordination overhead**



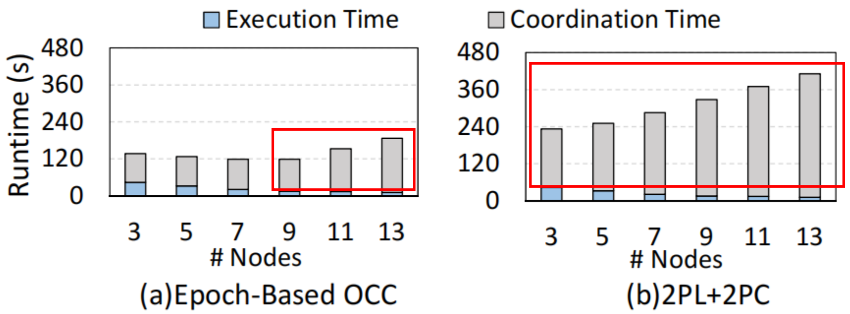
Communications in Distributed Concurrency Control

Problem2: High Communication in CC

- **Decoupled CC increasing latency**
  - *Communication with hardware VS communication with network.*
  - **Higher latency**, not suitable for OLTP workloads.
- **High Coordination overhead**
  - the coordination overhead increases when adding CC nodes
  - ***Limited scalability***



Communications in Distributed Concurrency Control



Breakdown of total processing time for distributed transactions with a changing number of nodes.

CCaaS Design

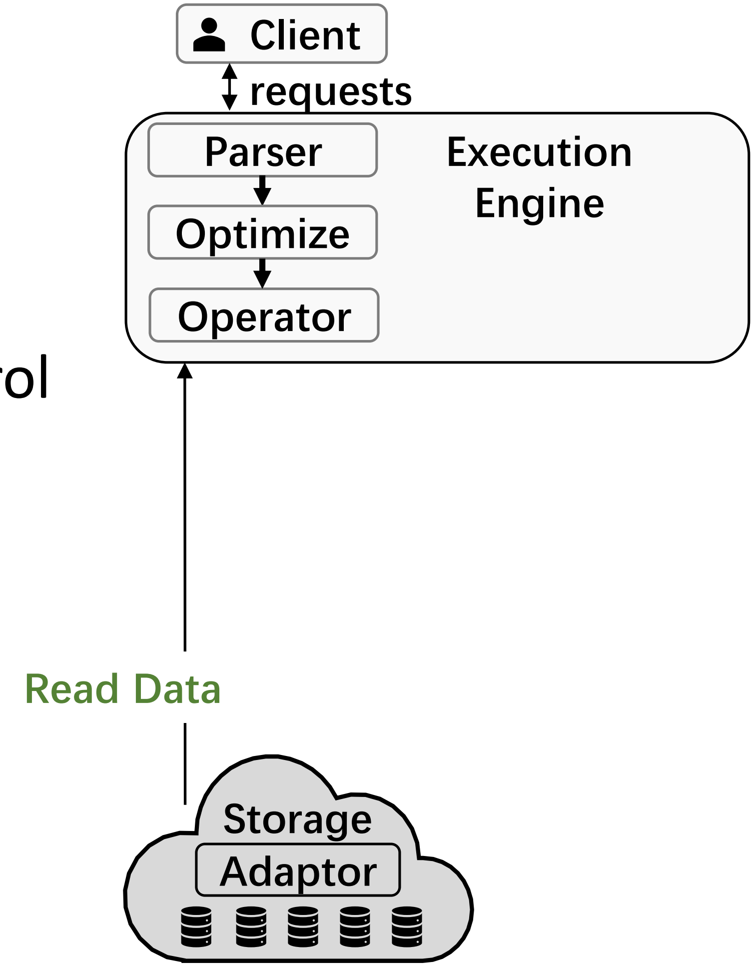
- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - *Limited scalability*

➢ Sharded Multi-Write Optimistic Concurrency Control (SM-OCC)

CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - *Limited scalability*

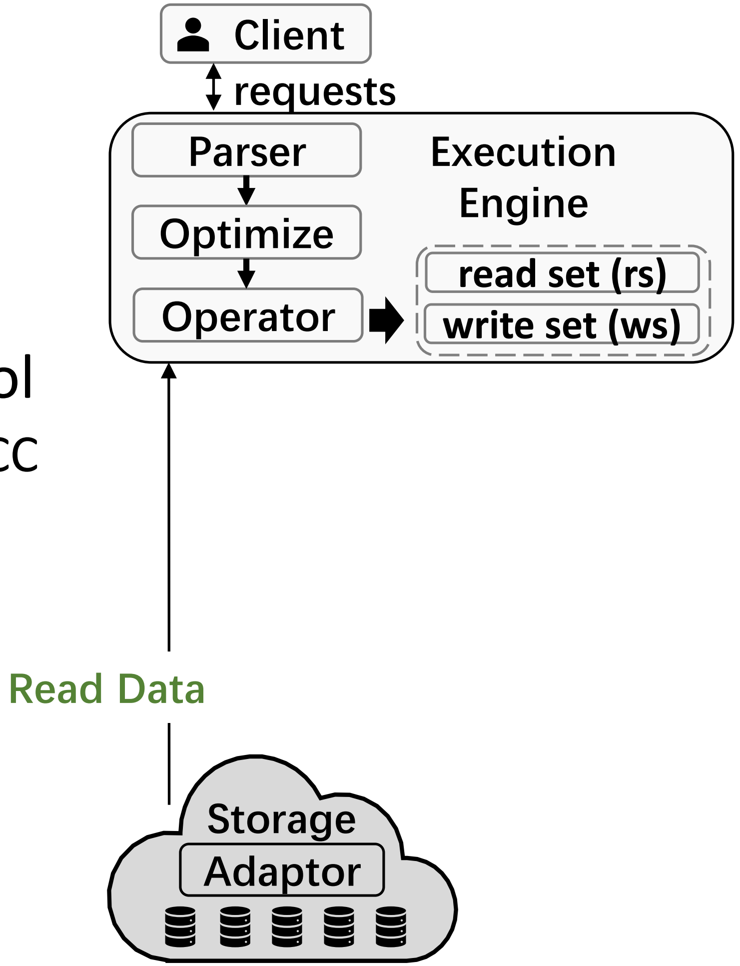
➢ Sharded Multi-Write Optimistic Concurrency Control
  ➢ ***Optimistic*** for fewer communication between execution-CC



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - ***Limited scalability***

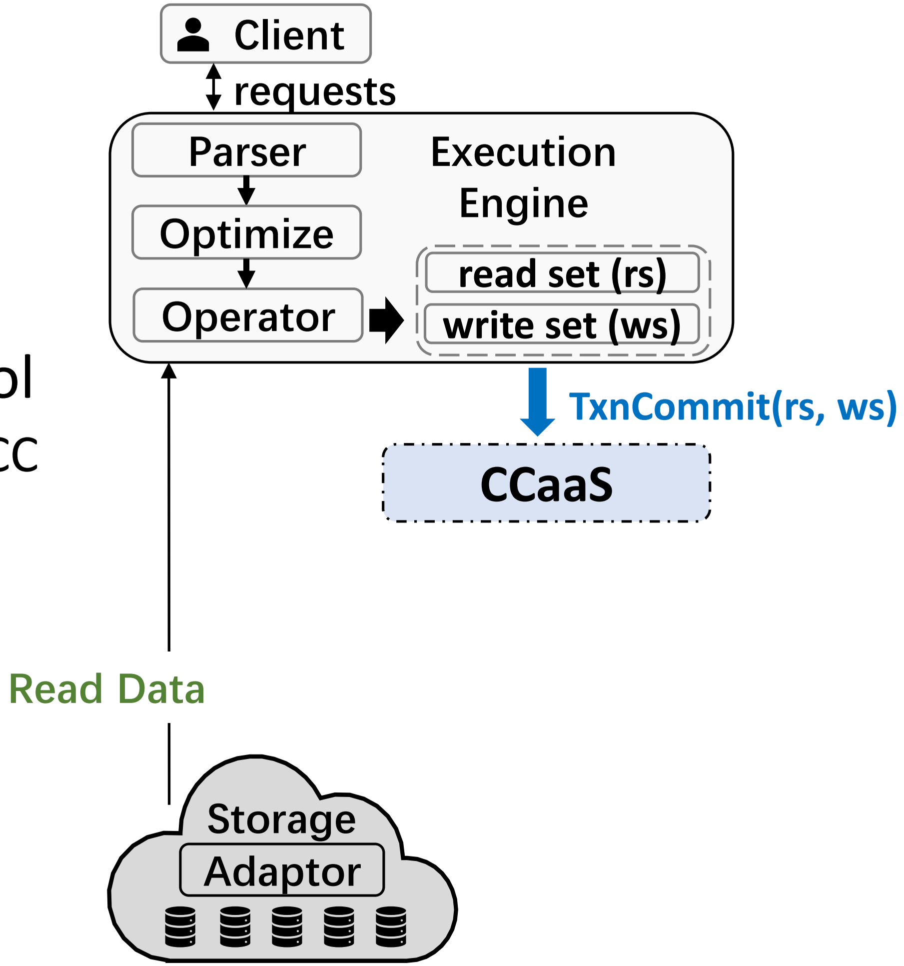
➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - *Limited scalability*

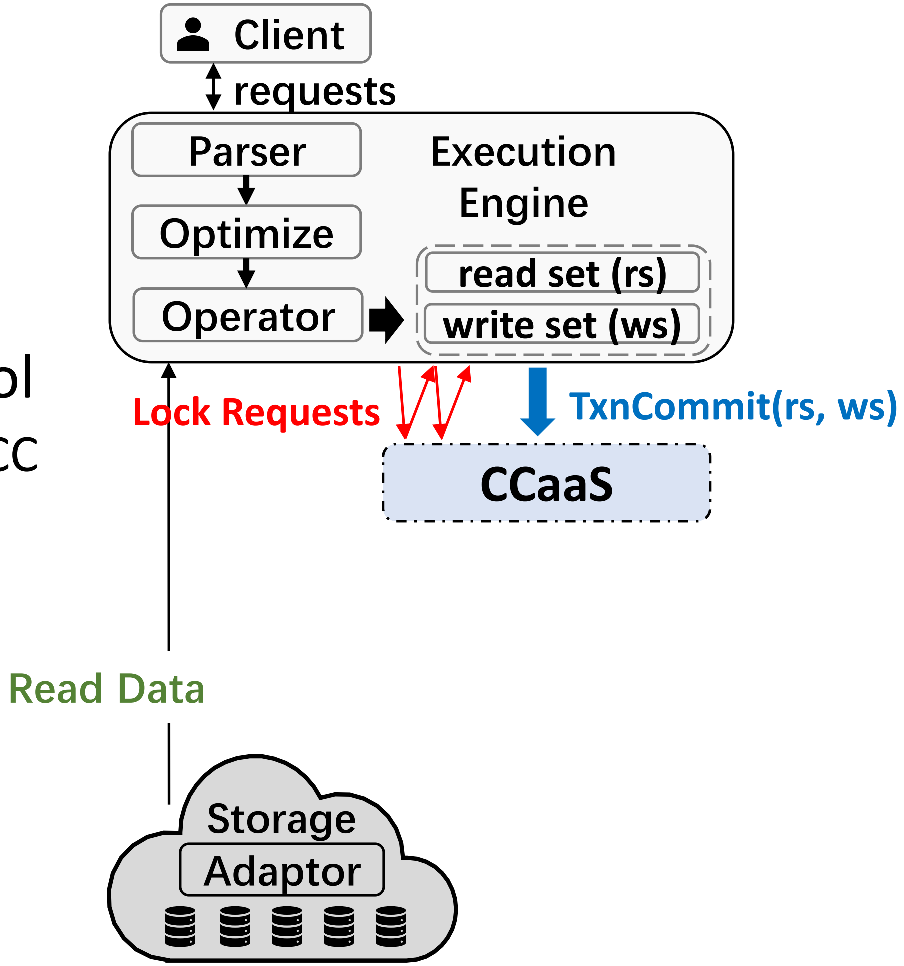
➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - *Limited scalability*

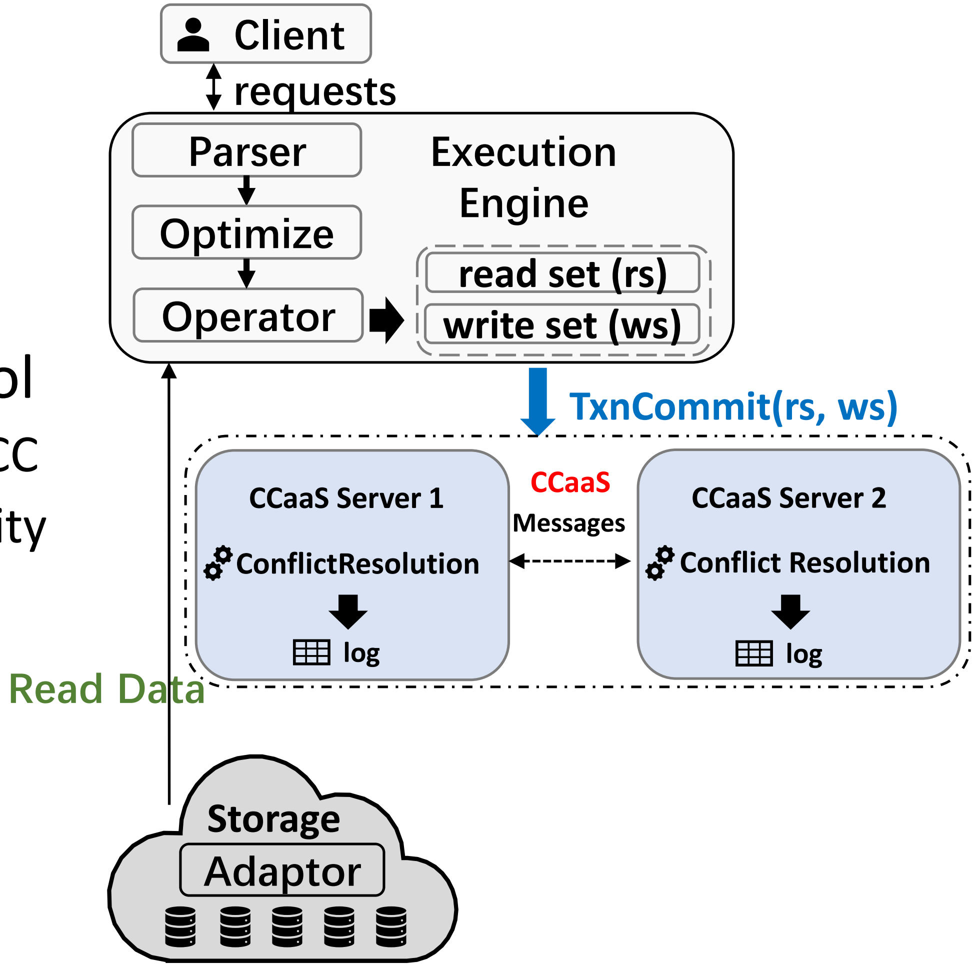
➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - ***Limited scalability***

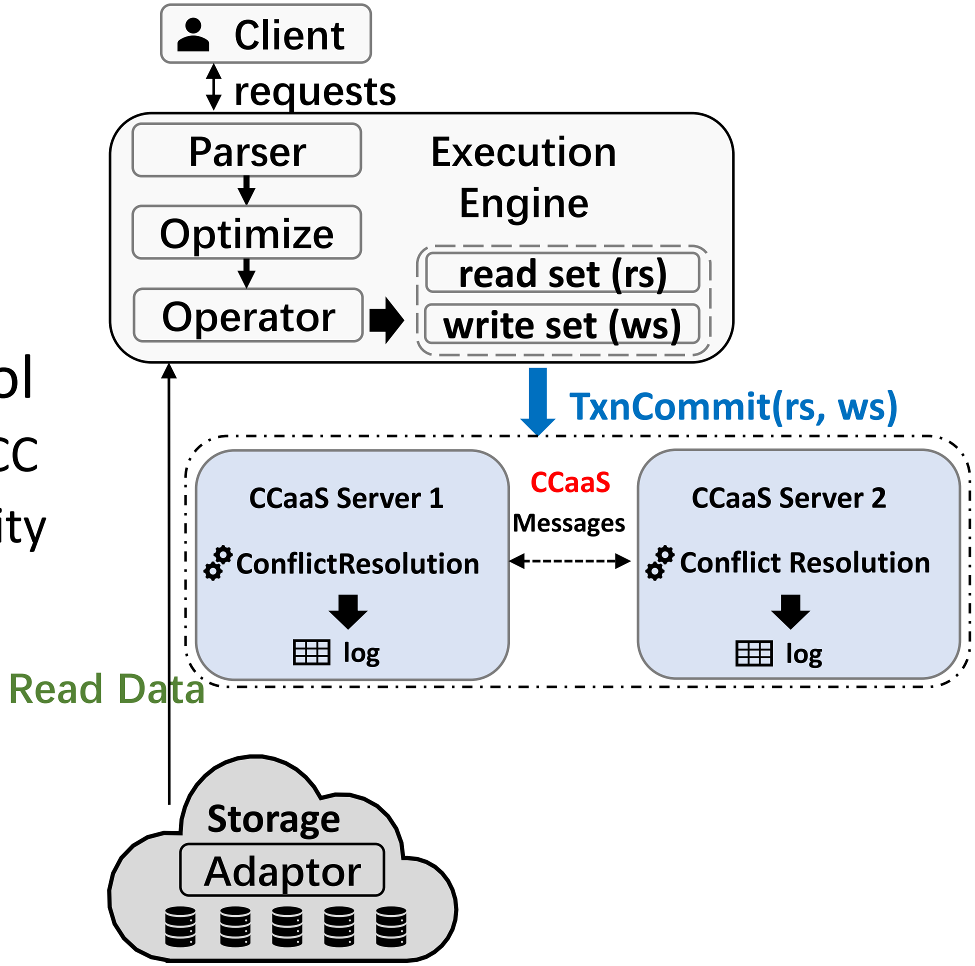
➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC
  - ➢ ***Multi-Write*** (Multi-Master) architecture for High availability



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - ***Limited scalability***

➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC
  - ➢ ***Multi-Write*** (Multi-Master) architecture for High availability
  - ➢ ***Sharding*** for high scalability



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - ***Limited scalability***

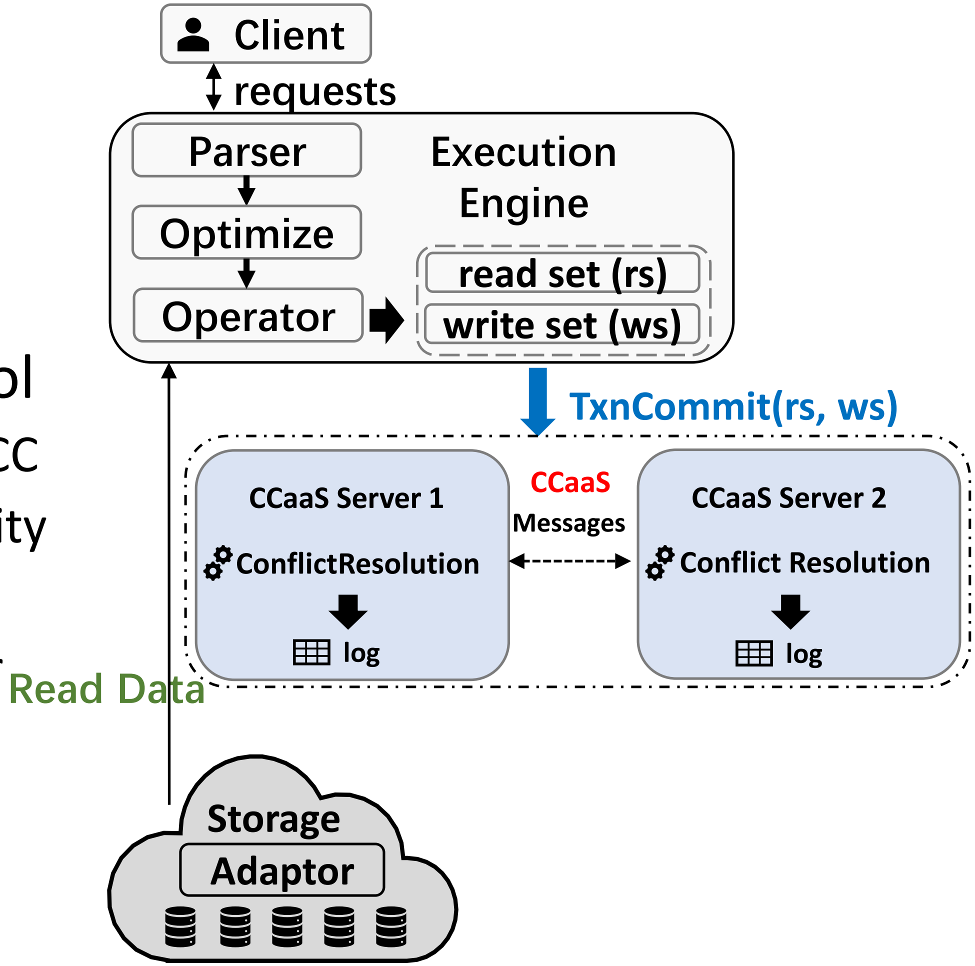
➢ Sharded Multi-Write Optimistic Concurrency Control

  ➢ ***Optimistic*** for fewer communication between execution-CC

  ➢ ***Multi-Write*** (Multi-Master) architecture for High availability

  ➢ ***Sharding*** for high scalability

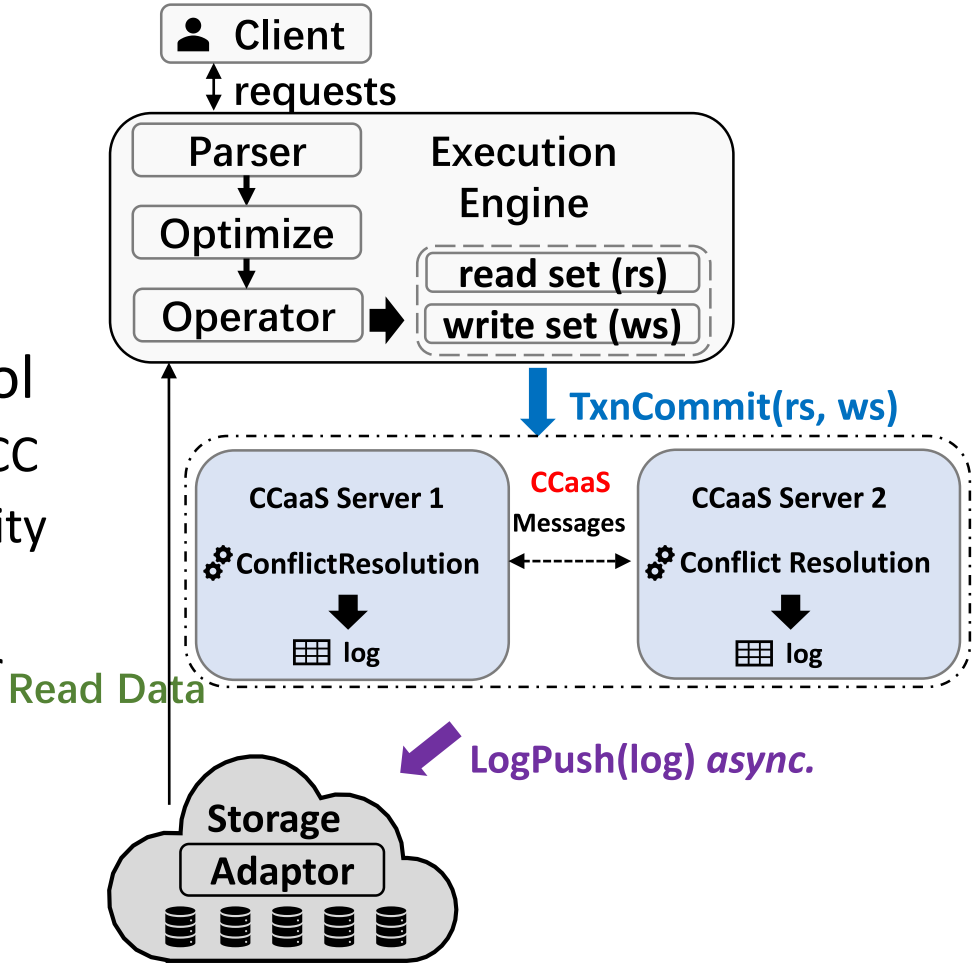
  ➢ ***Epoch-Based & Deterministic Conflict Resolving*** for fewer communication between CCaaS nodes



CCaaS Design

- **Decoupled CC increasing latency**
  - **Higher latency**
- **High Coordination overhead**
  - ***Limited scalability***

➢ Sharded Multi-Write Optimistic Concurrency Control
  - ➢ ***Optimistic*** for fewer communication between execution-CC
  - ➢ ***Multi-Write*** (Multi-Master) architecture for High availability
  - ➢ ***Sharding*** for high scalability
  - ➢ ***Epoch-Based & Deterministic Conflict Resolving*** for fewer communication between CCaaS nodes
  - ➢ ***Async*** Log push



SM-OCC

Snapshot X=1, Y=1

Each node maintains ***sharded*** meta-data, achieving high scalability.

CCaaS Node1

CCaaS Node2

Snapshot X=1, ,Z=1

SM-OCC

Snapshot X=1, Y=1

Different nodes maintains some of ***the same shard,*** building a ***multi-master arc.***

CCaaS Node1

CCaaS Node2

Snapshot X=1, ,Z=1

SM-OCC

Snapshot X=1,Y=1

T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5

CCaaS Node1

CCaaS Node2

Snapshot X=1, ,Z=1

SM-OCC

Snapshot X=1,Y=1

T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5

Note: csn = local time : node_id

CCaaS Node1

①Tag Commit Info {csn = 1:1, cen = 1}

CCaaS Node2

Snapshot X=1, ,Z=1

SM-OCC

Snapshot X=1,Y=1

T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5

CCaaS Node1

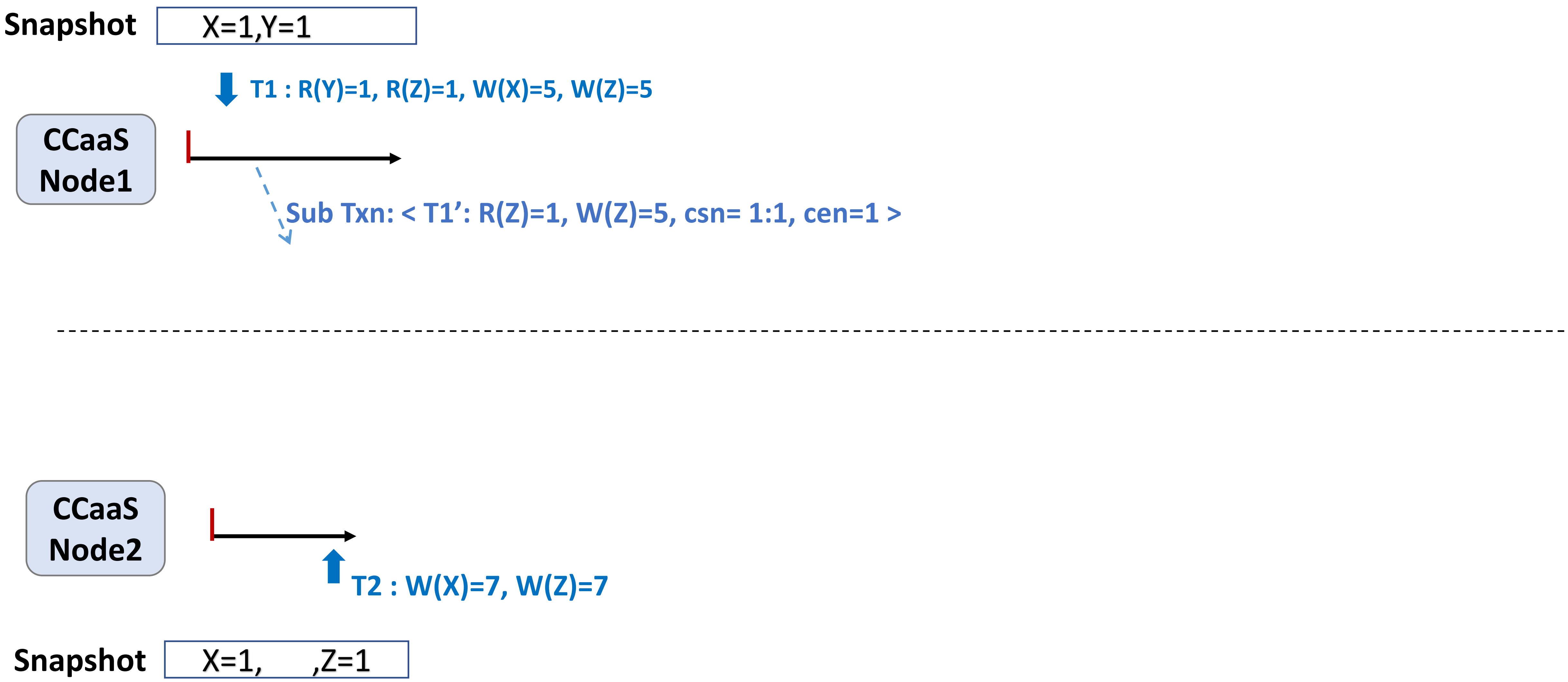
②Replicate Sharded Txn{R(Z) W(Z)} to CCaaS Node 2

Sub Txn: <T1’: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>

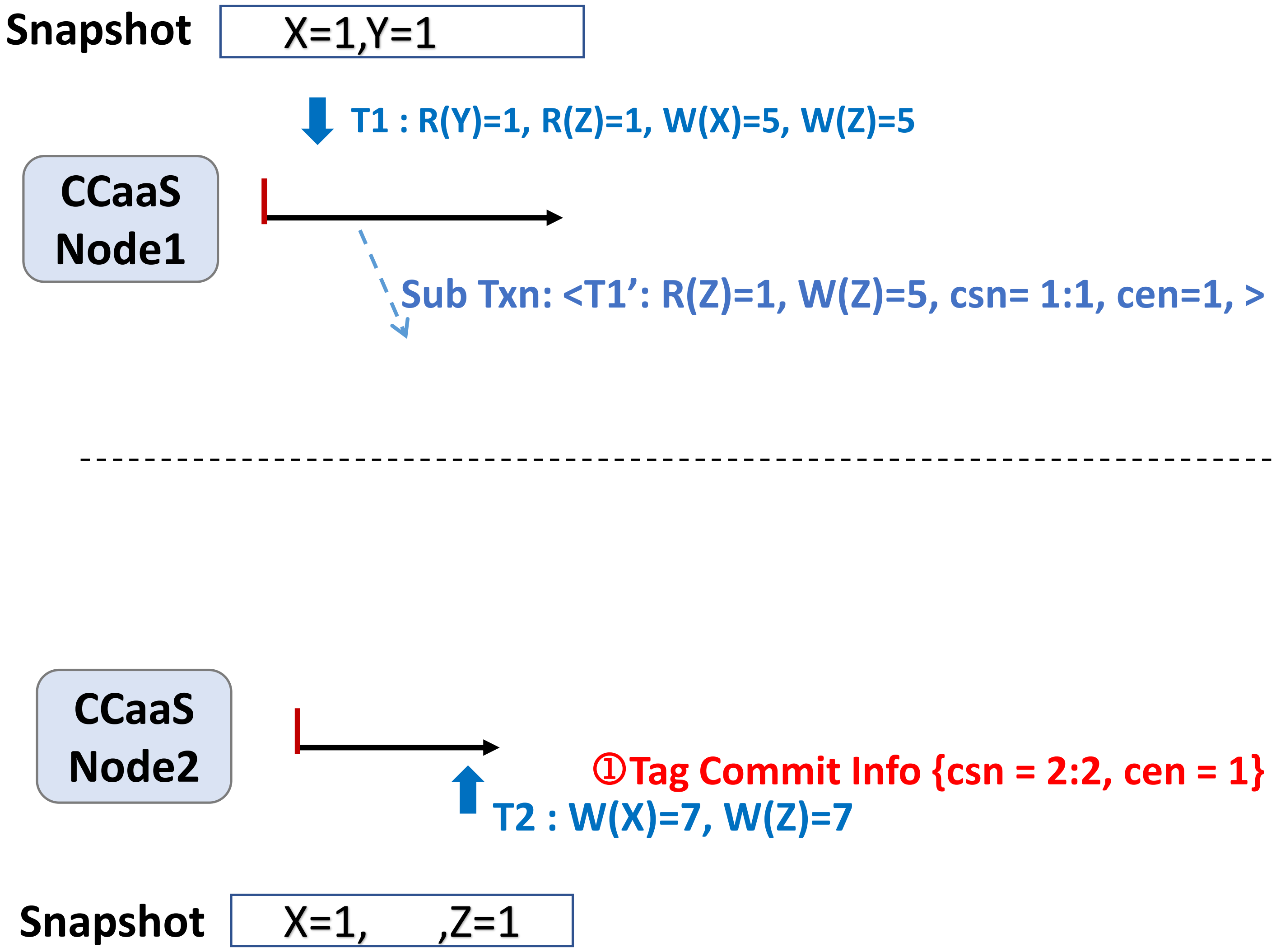
CCaaS Node2

Snapshot X=1, ,Z=1

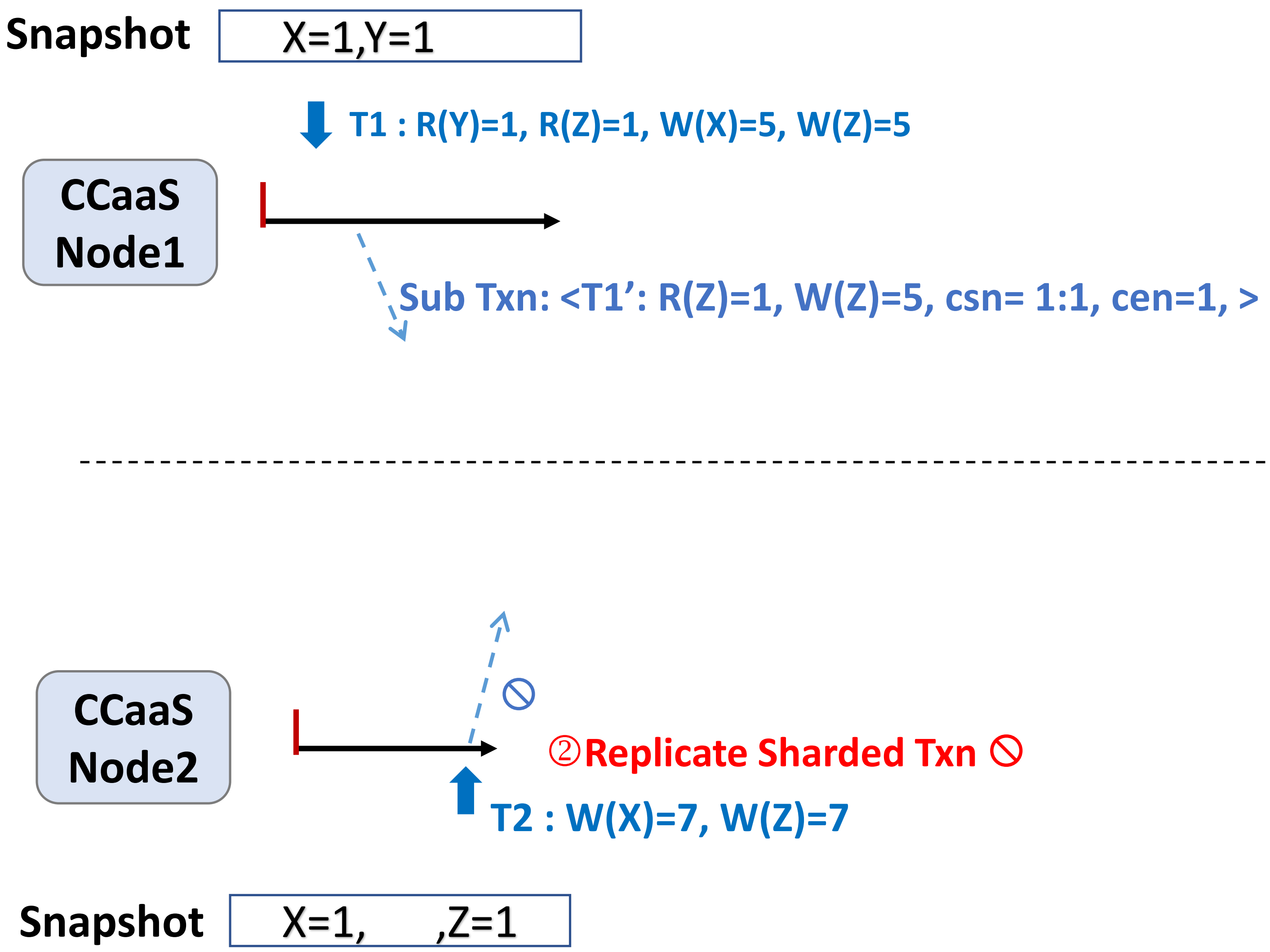
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Sub Txn: < T1’: R(Z)=1, W(Z)=5, csn= 1:1, cen=1 >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

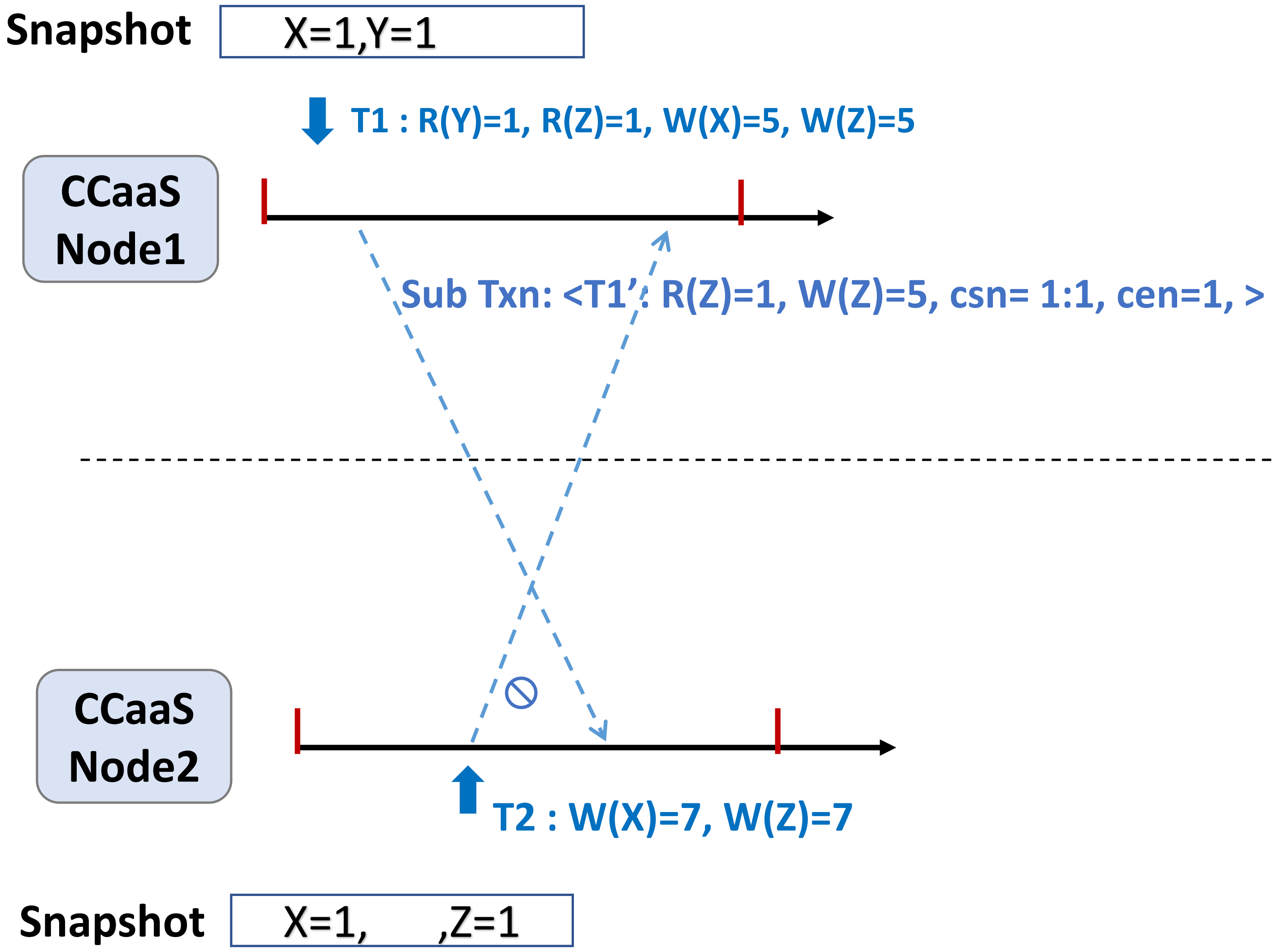

Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Sub Txn: <T1’: R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
CCaaS Node2
①Tag Commit Info {csn = 2:2, cen = 1}
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

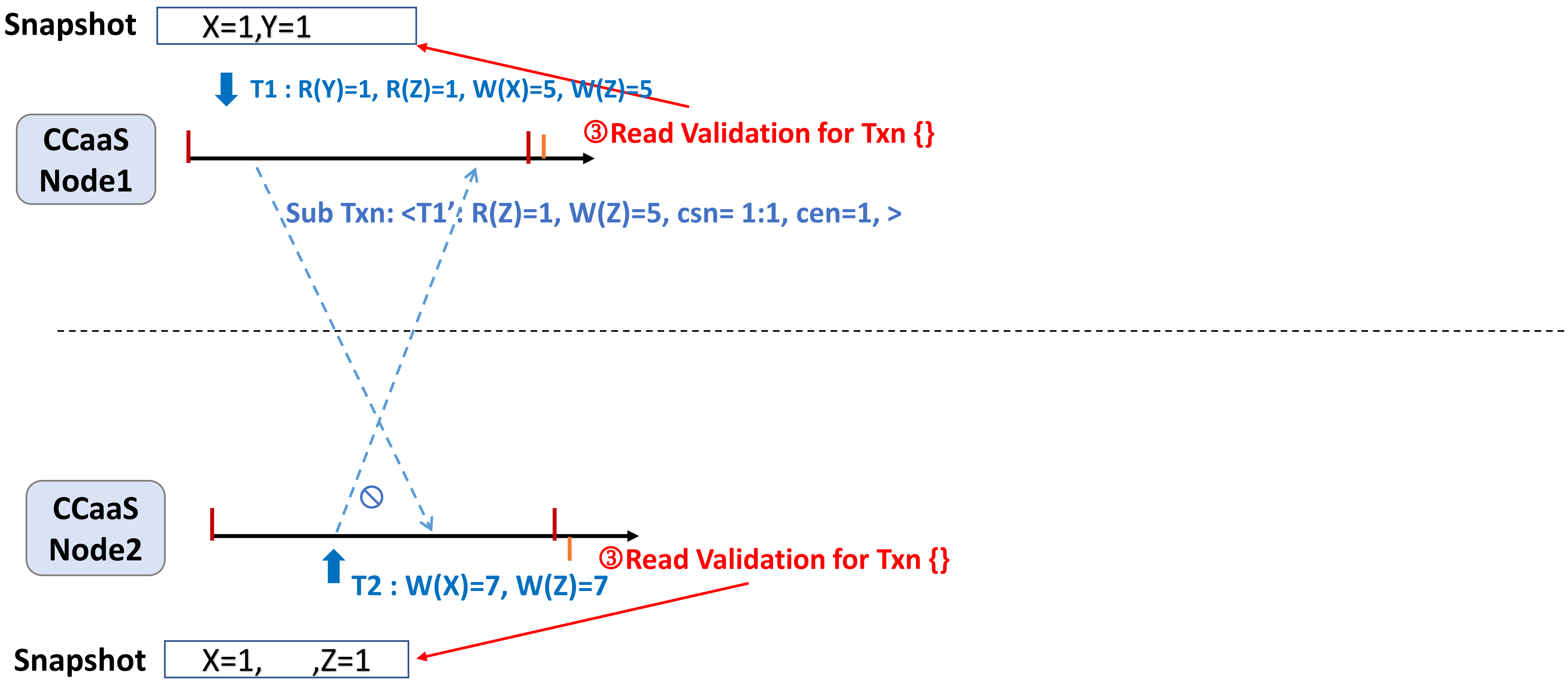


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Sub Txn: <T1': R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
CCaaS Node2
②Replicate Sharded Txn
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

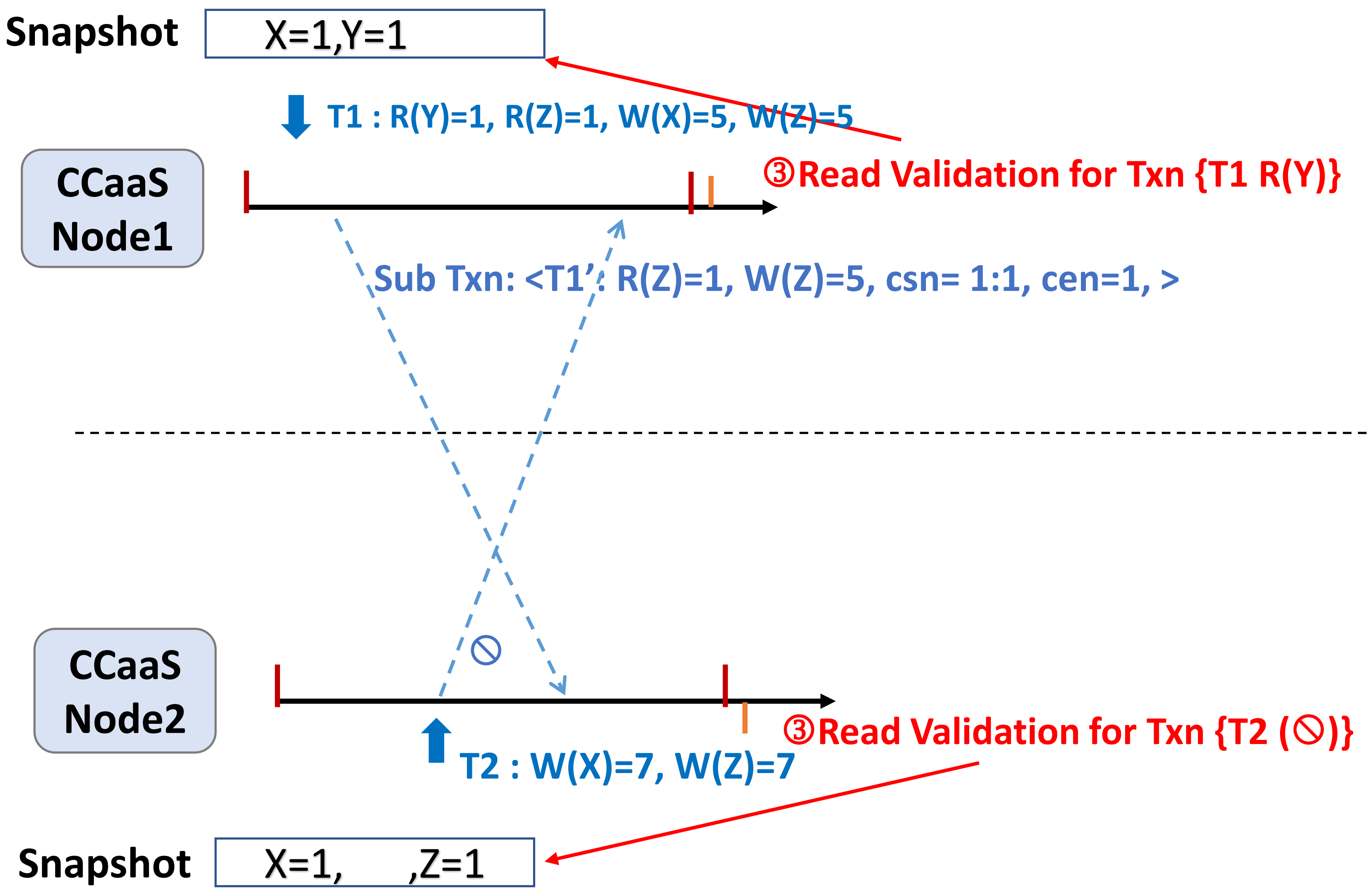
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Sub Txn: <T1': R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

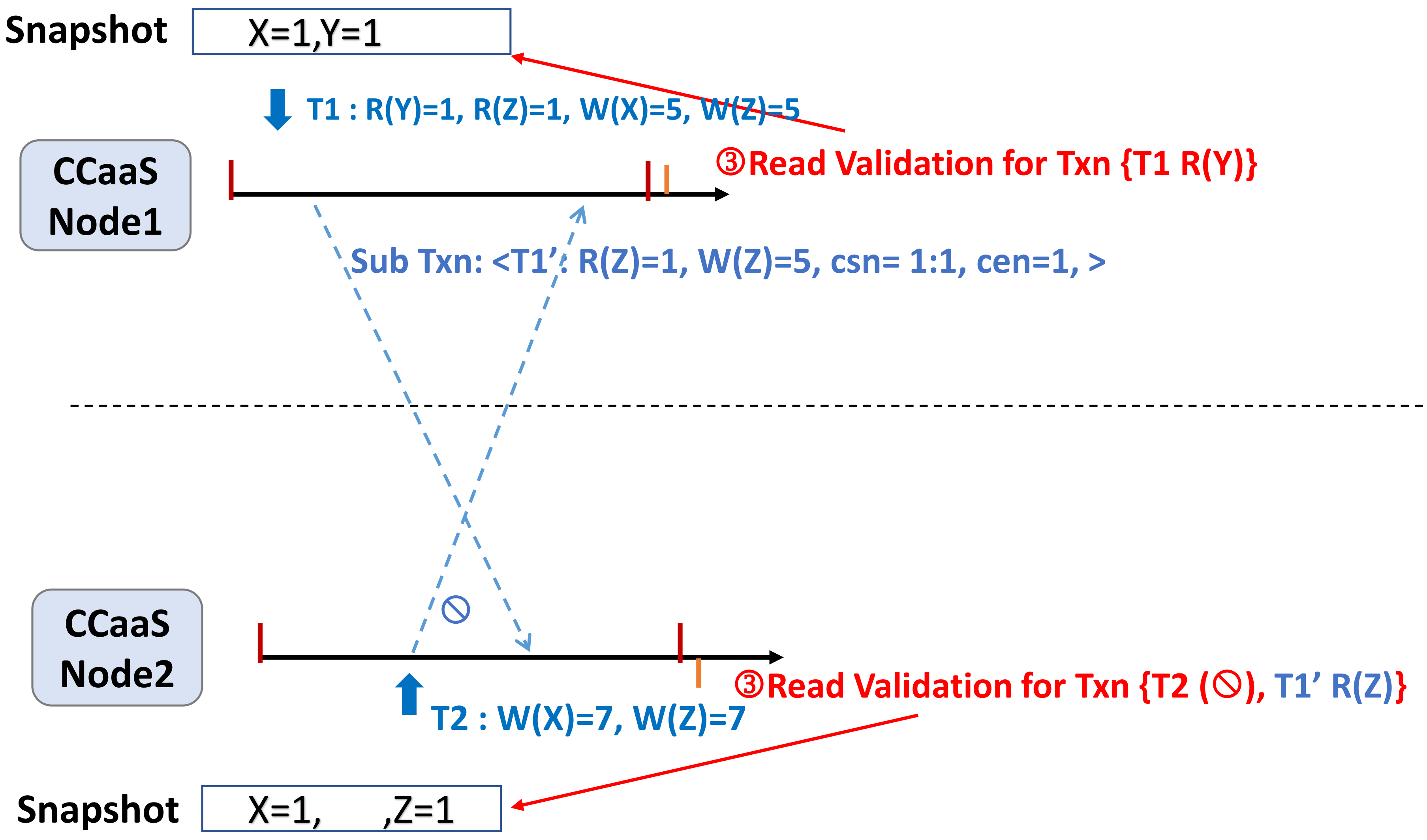
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
③Read Validation for Txn {}
Sub Txn: <T1': R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
CCaaS Node2
③Read Validation for Txn {}
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

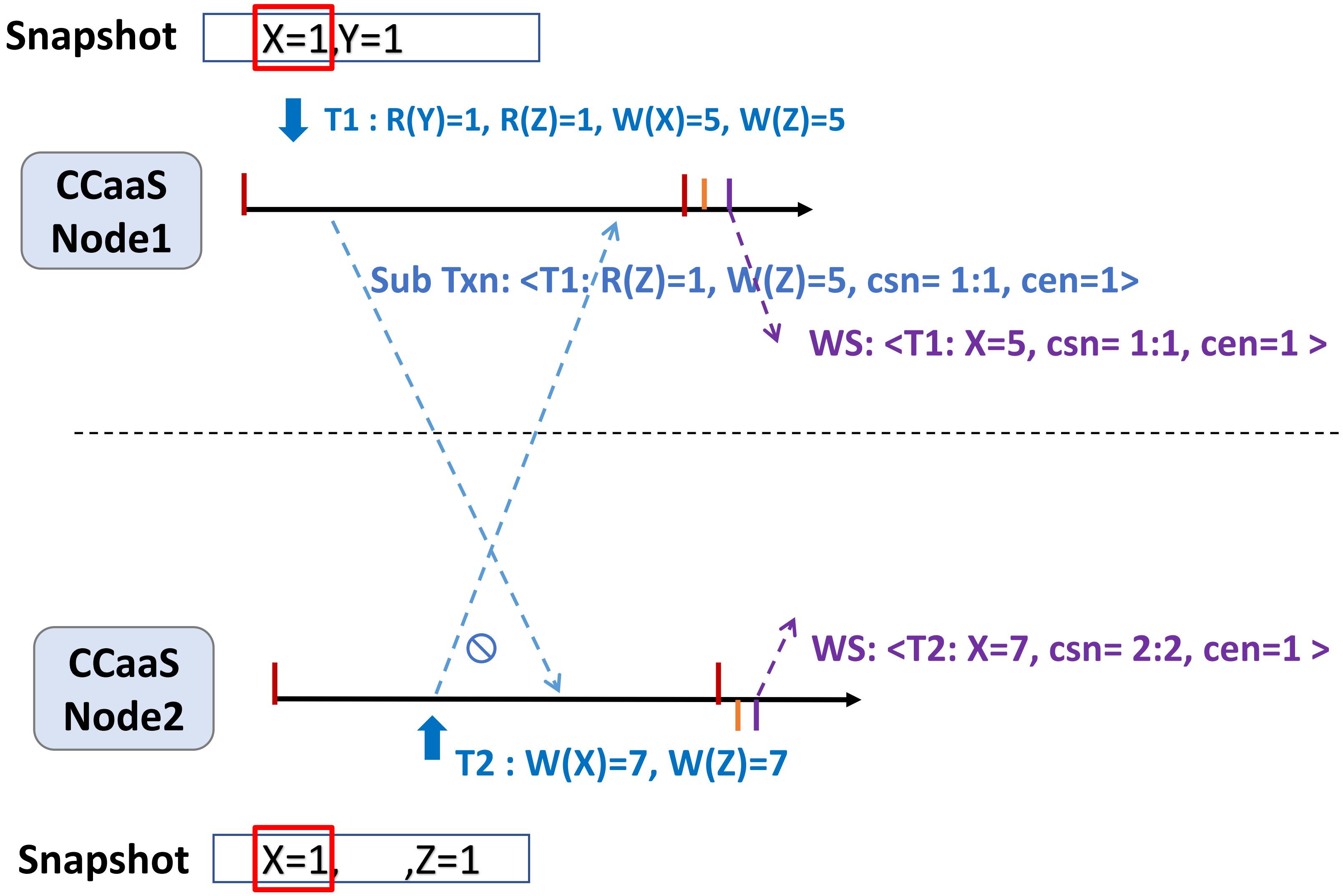

Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
③Read Validation for Txn {T1 R(Y)}
Sub Txn: <T1': R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
⃠
CCaaS Node2
③Read Validation for Txn {T2 (⃠)}
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

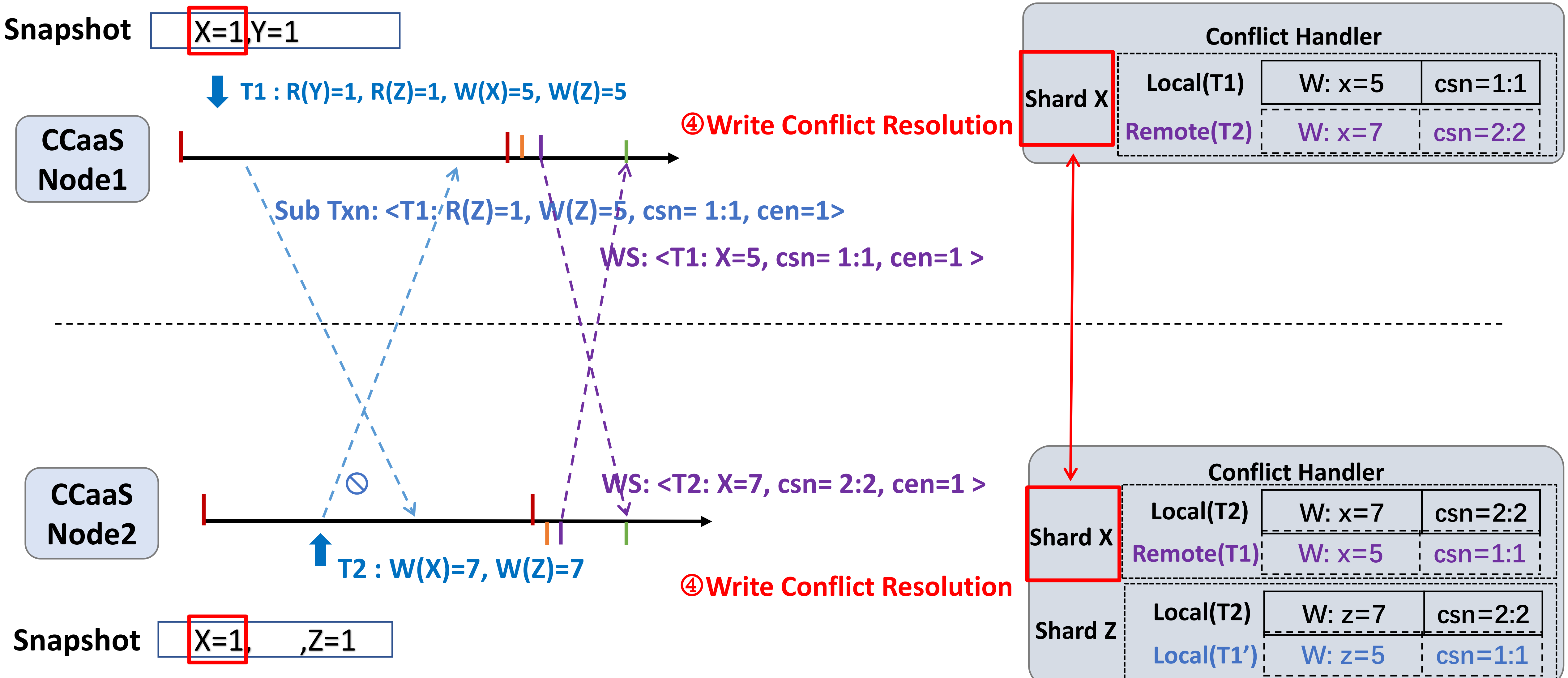


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
③Read Validation for Txn {T1 R(Y)}
Sub Txn: <T1': R(Z)=1, W(Z)=5, csn= 1:1, cen=1, >
CCaaS Node2
③Read Validation for Txn {T2 (⃠), T1' R(Z)}
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

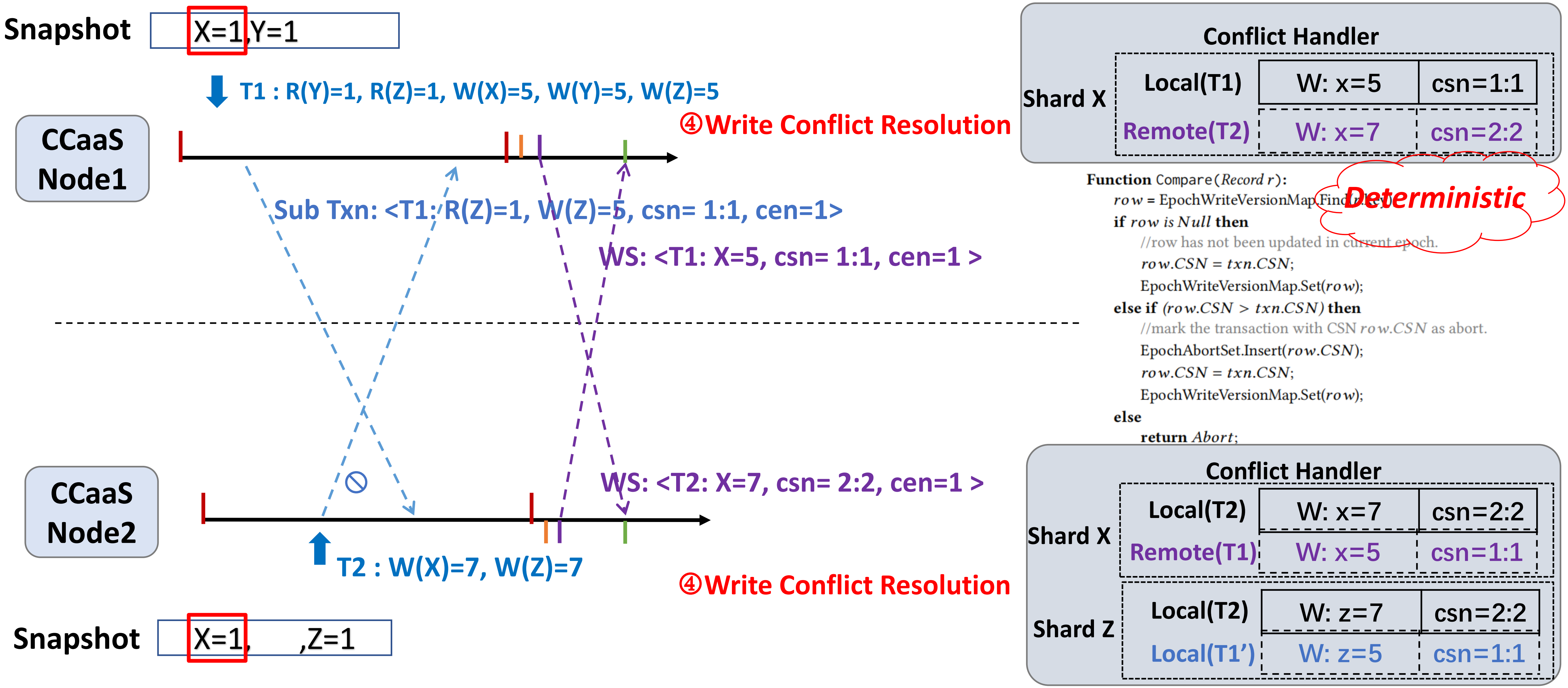

Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC



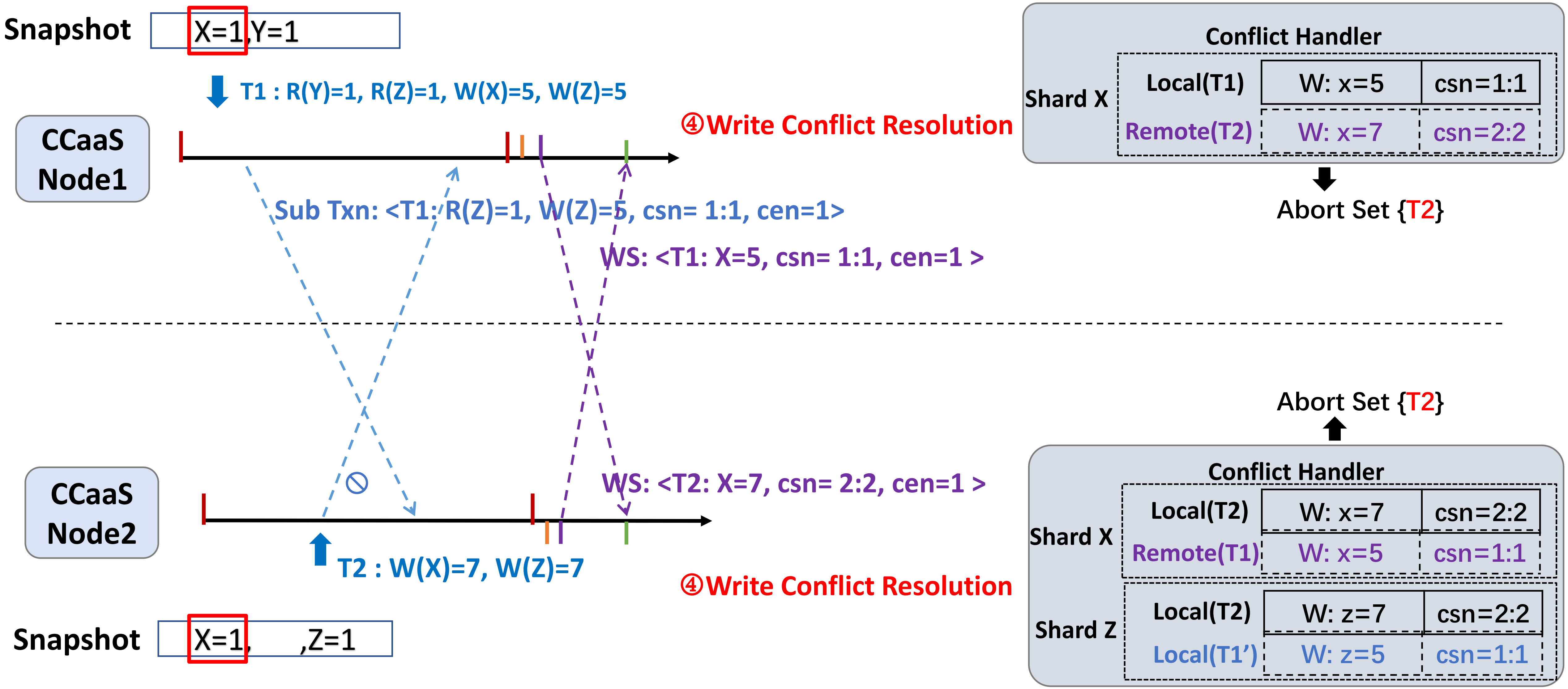
Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
④Write Conflict Resolution
Conflict Handler
Shard X
Local(T1)
W: x=5
csn=1:1
Remote(T2)
W: x=7
csn=2:2
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
④Write Conflict Resolution
Conflict Handler
Shard X
Local(T2)
W: x=7
csn=2:2
Remote(T1)
W: x=5
csn=1:1
Shard Z
Local(T2)
W: z=7
csn=2:2
Local(T1’)
W: z=5
csn=1:1
Snapshot
X=1, ,Z=1

SM-OCC

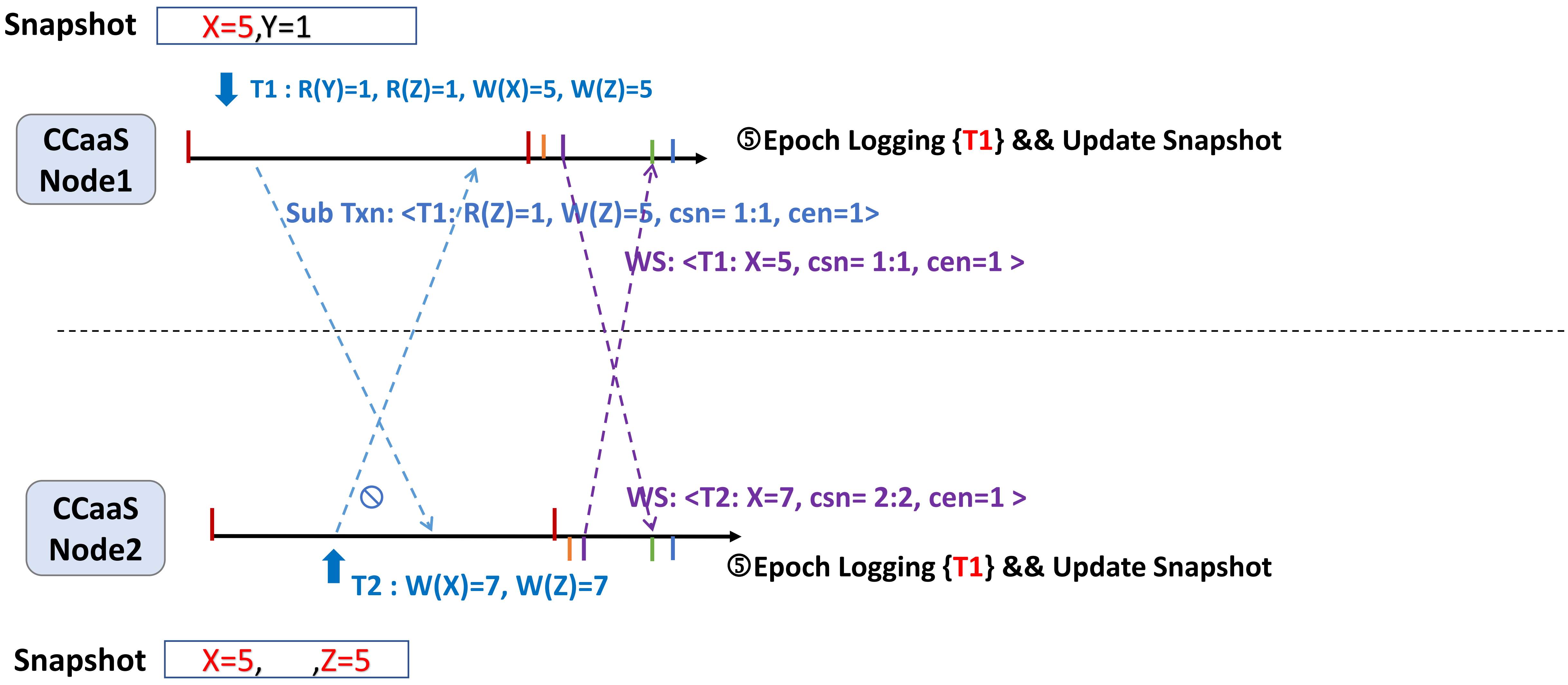


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Y)=5, W(Z)=5
CCaaS Node1
④Write Conflict Resolution
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
Conflict Handler
Shard X
Local(T1)
W: x=5
csn=1:1
Remote(T2)
W: x=7
csn=2:2
Deterministic
Function Compare(Record r):
row = EpochWriteVersionMap.Find(key)
if row is Null then
//row has not been updated in current epoch.
row.CSN = txn.CSN;
EpochWriteVersionMap.Set(row);
else if (row.CSN > txn.CSN) then
//mark the transaction with CSN row.CSN as abort.
EpochAbortSet.Insert(row.CSN);
row.CSN = txn.CSN;
EpochWriteVersionMap.Set(row);
else
return Abort;
CCaaS Node2
WS: <T2: X=7, csn= 2:2, cen=1 >
T2 : W(X)=7, W(Z)=7
④Write Conflict Resolution
Snapshot
X=1, ,Z=1
Conflict Handler
Shard X
Local(T2)
W: x=7
csn=2:2
Remote(T1)
W: x=5
csn=1:1
Shard Z
Local(T2)
W: z=7
csn=2:2
Local(T1')
W: z=5
csn=1:1

SM-OCC

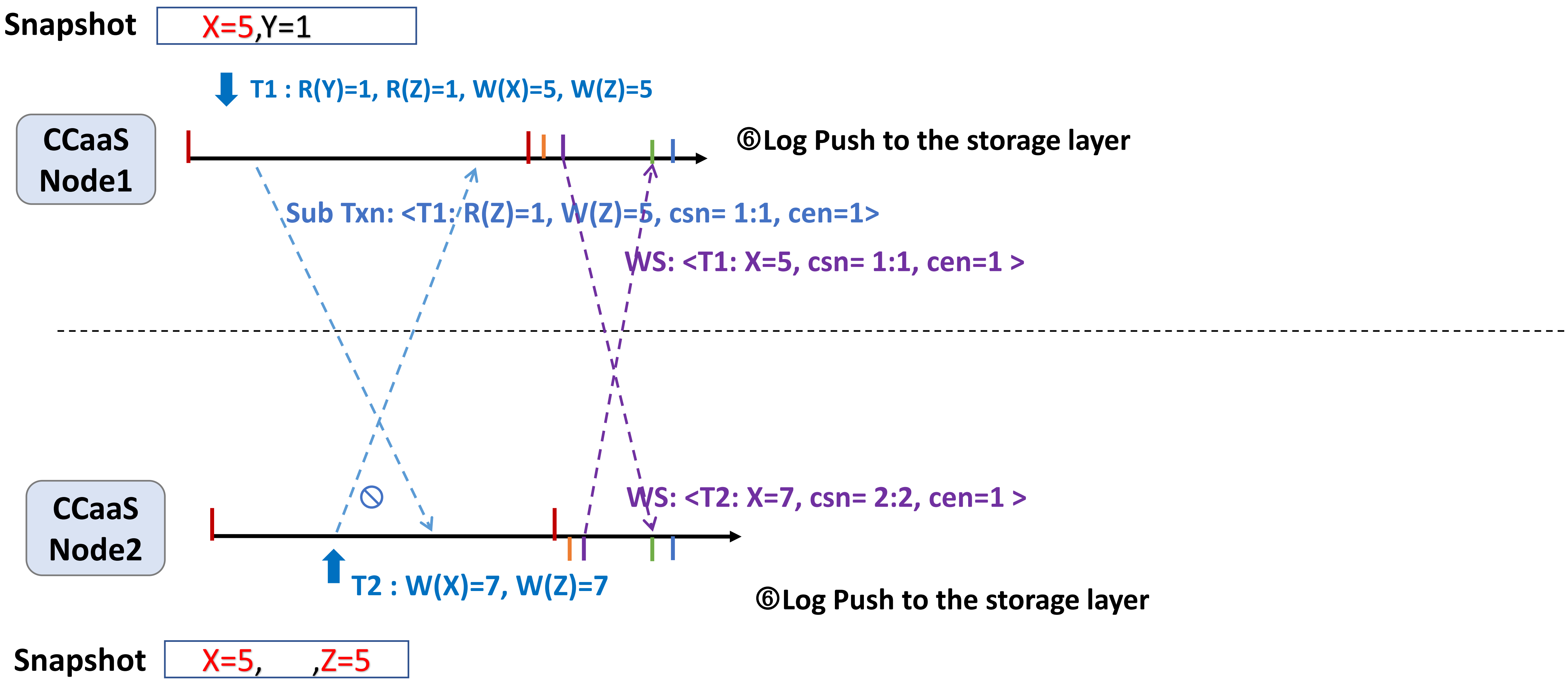

Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
④Write Conflict Resolution
Conflict Handler
Shard X
Local(T1)
W: x=5
csn=1:1
Remote(T2)
W: x=7
csn=2:2
Abort Set {T2}
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
Abort Set {T2}
Conflict Handler
CCaaS Node2
WS: <T2: X=7, csn= 2:2, cen=1 >
Shard X
Local(T2)
W: x=7
csn=2:2
Remote(T1)
W: x=5
csn=1:1
T2 : W(X)=7, W(Z)=7
④Write Conflict Resolution
Shard Z
Local(T2)
W: z=7
csn=2:2
Local(T1')
W: z=5
csn=1:1
Snapshot
X=1, ,Z=1

SM-OCC



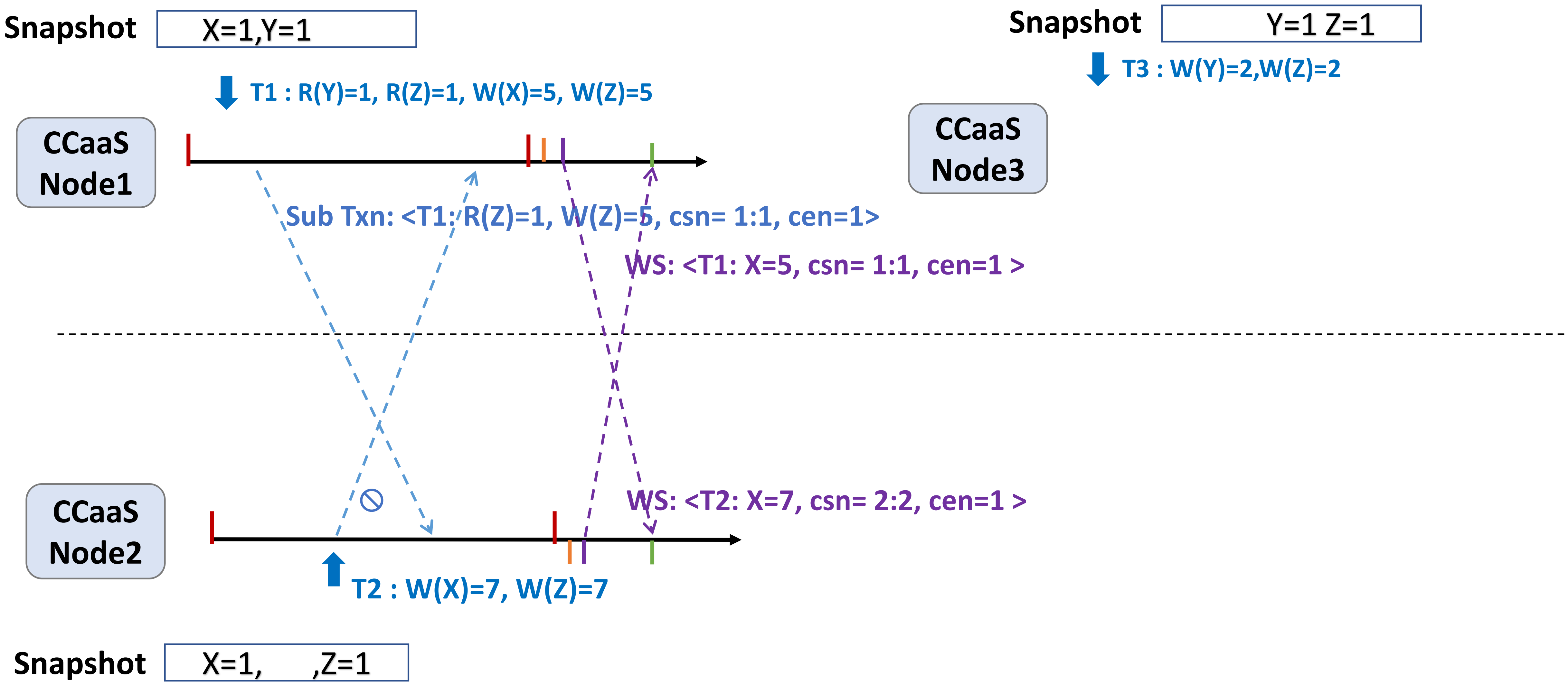
Snapshot
X=5,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
⑤Epoch Logging {T1} && Update Snapshot
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
⑤Epoch Logging {T1} && Update Snapshot
Snapshot
X=5, ,Z=5

SM-OCC

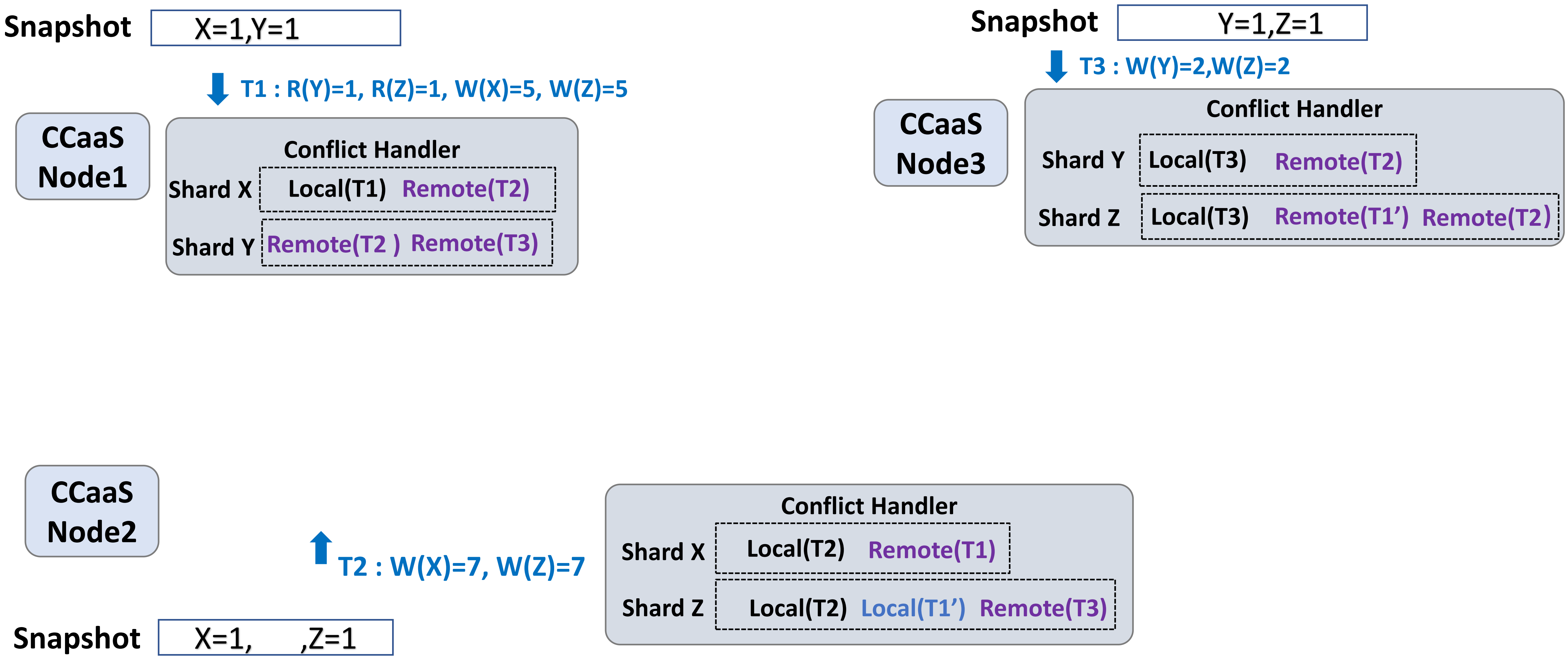


Snapshot
X=5,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
⑥Log Push to the storage layer
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS Node2
T2 : W(X)=7, W(Z)=7
⑥Log Push to the storage layer
Snapshot
X=5, ,Z=5

SM-OCC

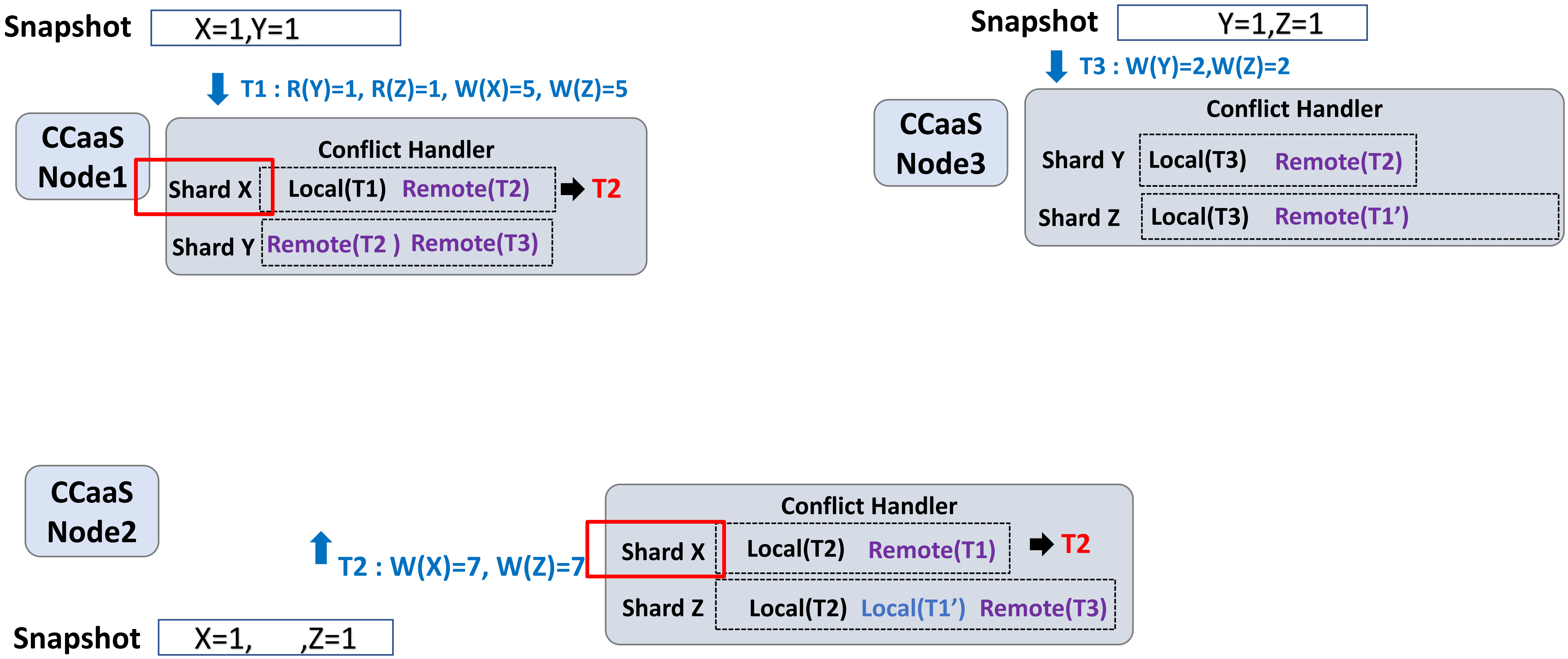

Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS
Node1
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
Snapshot
Y=1 Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS
Node3
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS
Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1

SM-OCC

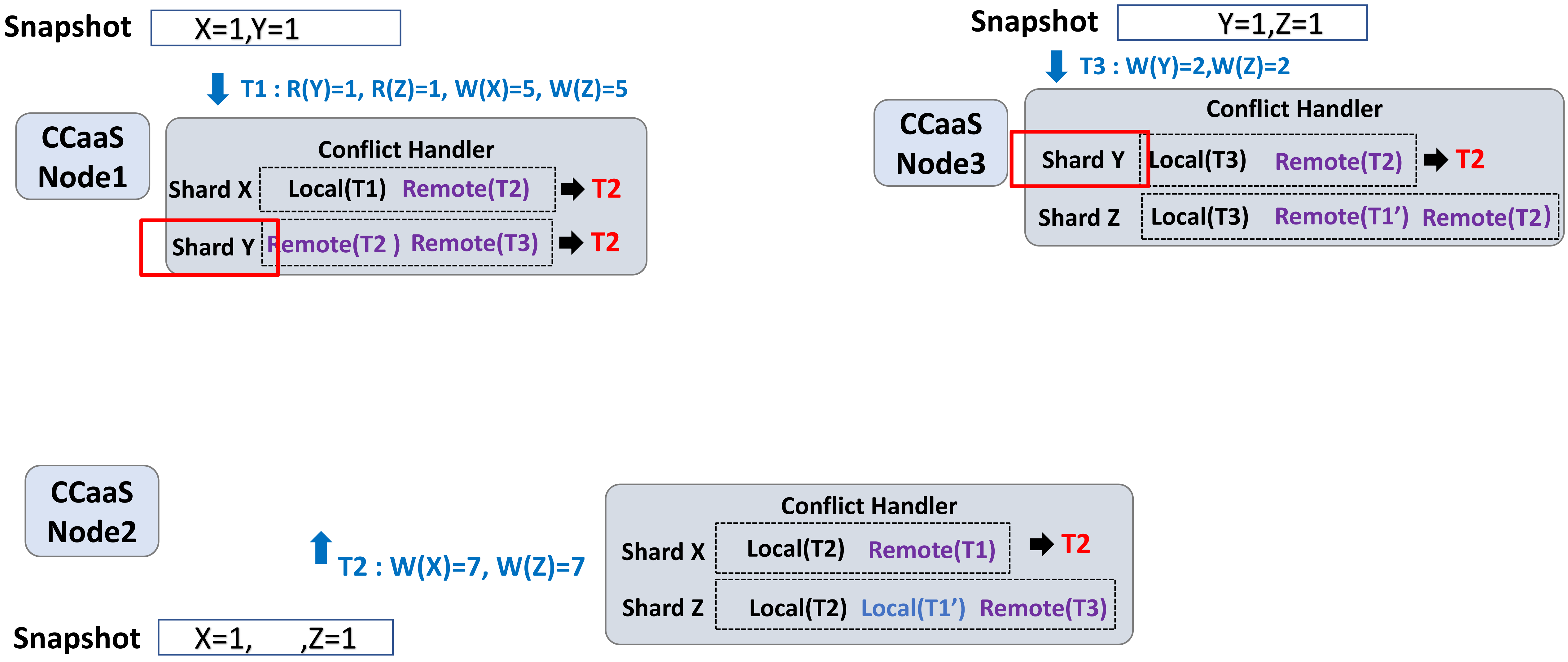


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1) Remote(T2)
Shard Y
Remote(T2) Remote(T3)
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3) Remote(T2)
Shard Z
Local(T3) Remote(T1’) Remote(T2)
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Conflict Handler
Shard X
Local(T2) Remote(T1)
Shard Z
Local(T2) Local(T1’) Remote(T3)
Snapshot
X=1, ,Z=1

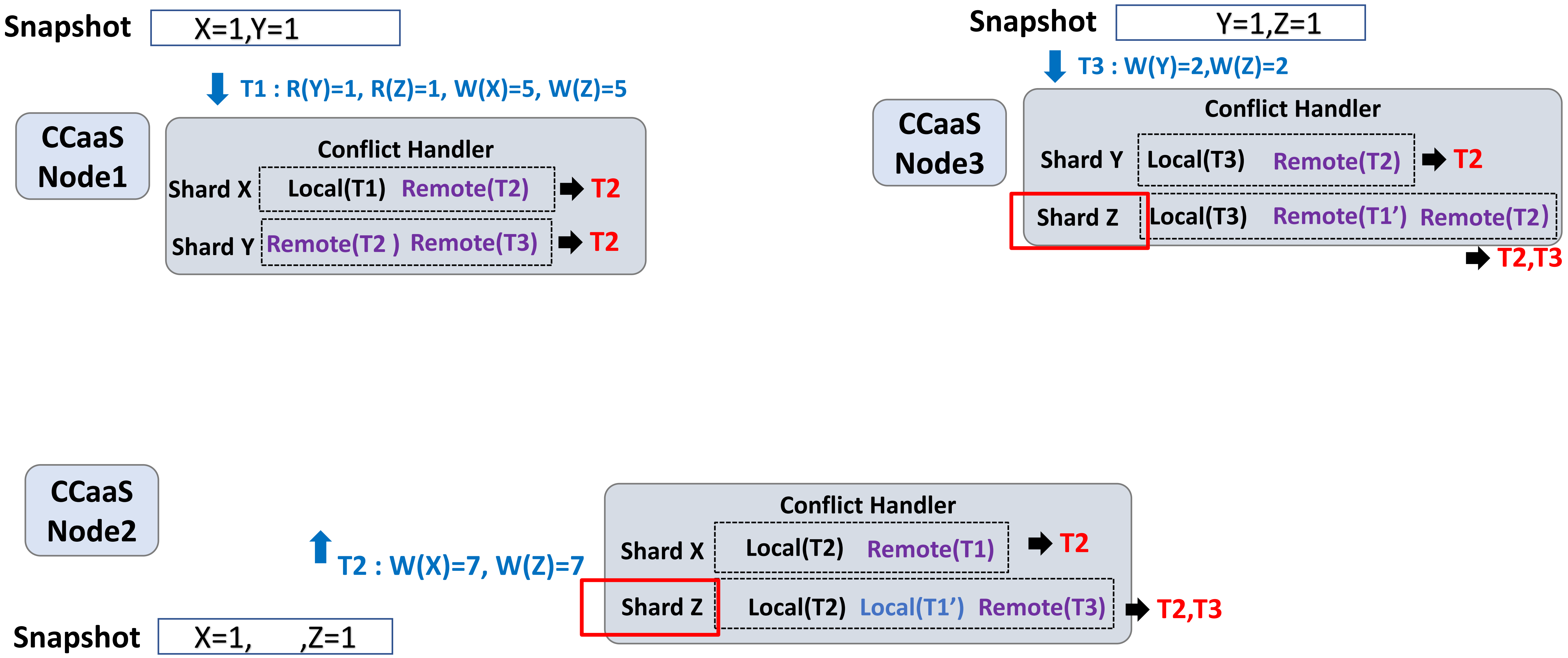
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1)
Remote(T2)
T2
Shard Y
Remote(T2)
Remote(T3)
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3)
Remote(T2)
Shard Z
Local(T3)
Remote(T1')
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Conflict Handler
Shard X
Local(T2)
Remote(T1)
T2
Shard Z
Local(T2)
Local(T1')
Remote(T3)
Snapshot
X=1,
,Z=1

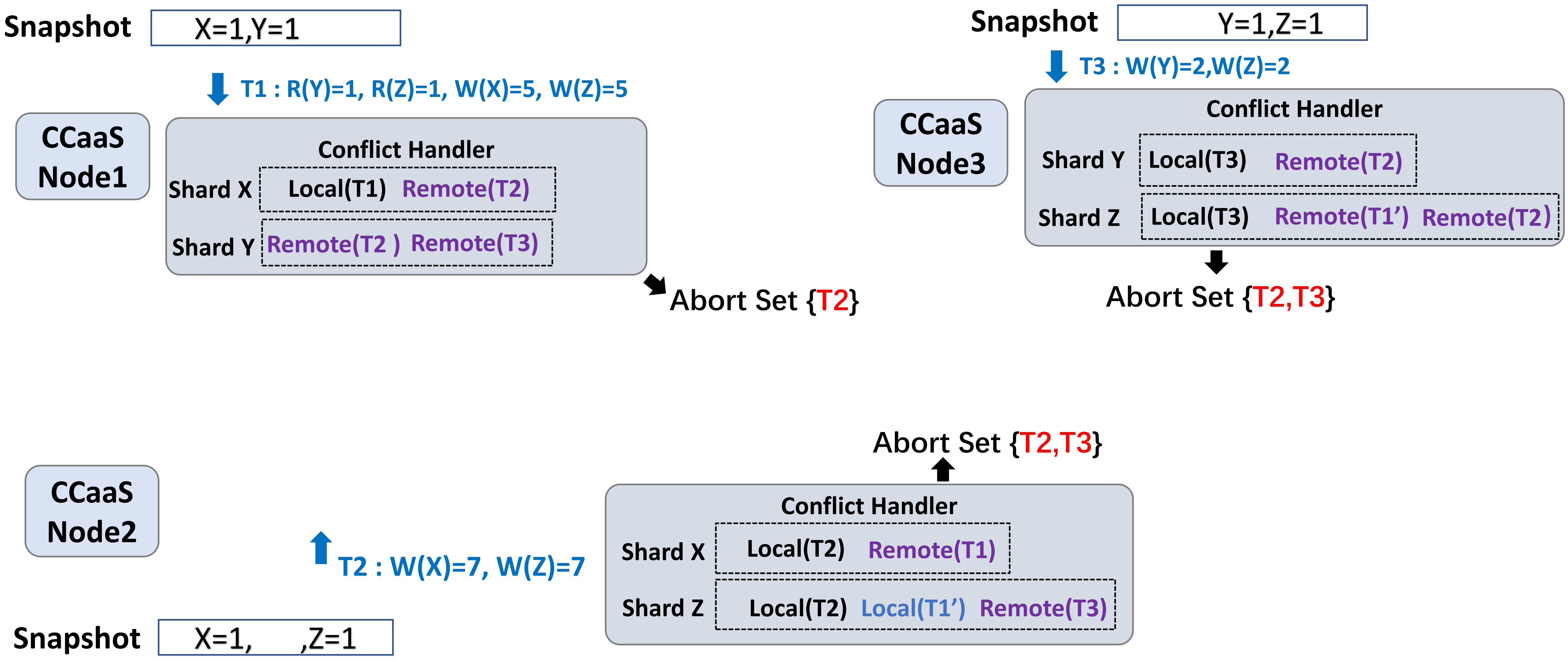
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS
Node1
Conflict Handler
Shard X
Local(T1)
Remote(T2)
T2
Shard Y
Remote(T2)
Remote(T3)
T2
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS
Node3
Conflict Handler
Shard Y
Local(T3)
Remote(T2)
T2
Shard Z
Local(T3)
Remote(T1')
Remote(T2)
CCaaS
Node2
T2 : W(X)=7, W(Z)=7
Conflict Handler
Shard X
Local(T2)
Remote(T1)
T2
Shard Z
Local(T2)
Local(T1')
Remote(T3)
Snapshot
X=1, ,Z=1

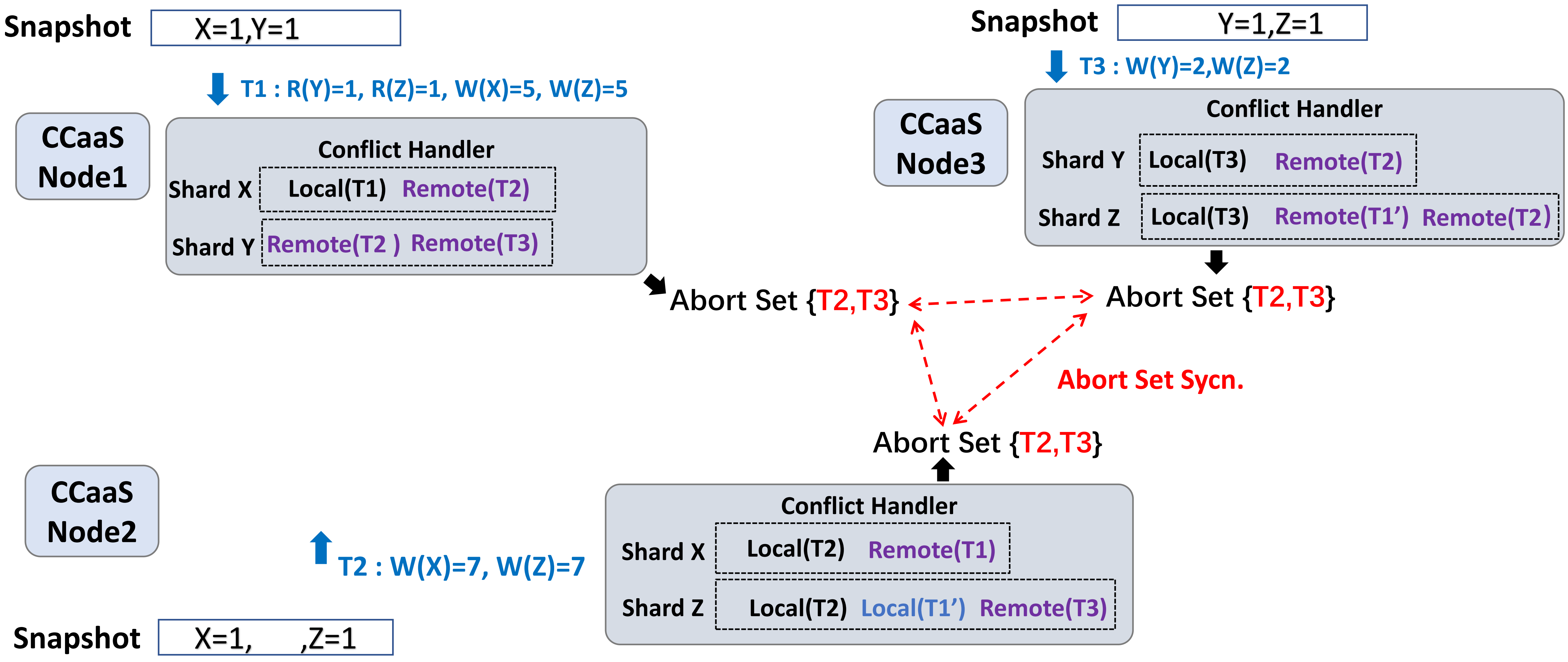
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1)
Remote(T2)
T2
Shard Y
Remote(T2)
Remote(T3)
T2
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3)
Remote(T2)
T2
Shard Z
Local(T3)
Remote(T1')
Remote(T2)
T2,T3
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1
Conflict Handler
Shard X
Local(T2)
Remote(T1)
T2
Shard Z
Local(T2)
Local(T1')
Remote(T3)
T2,T3

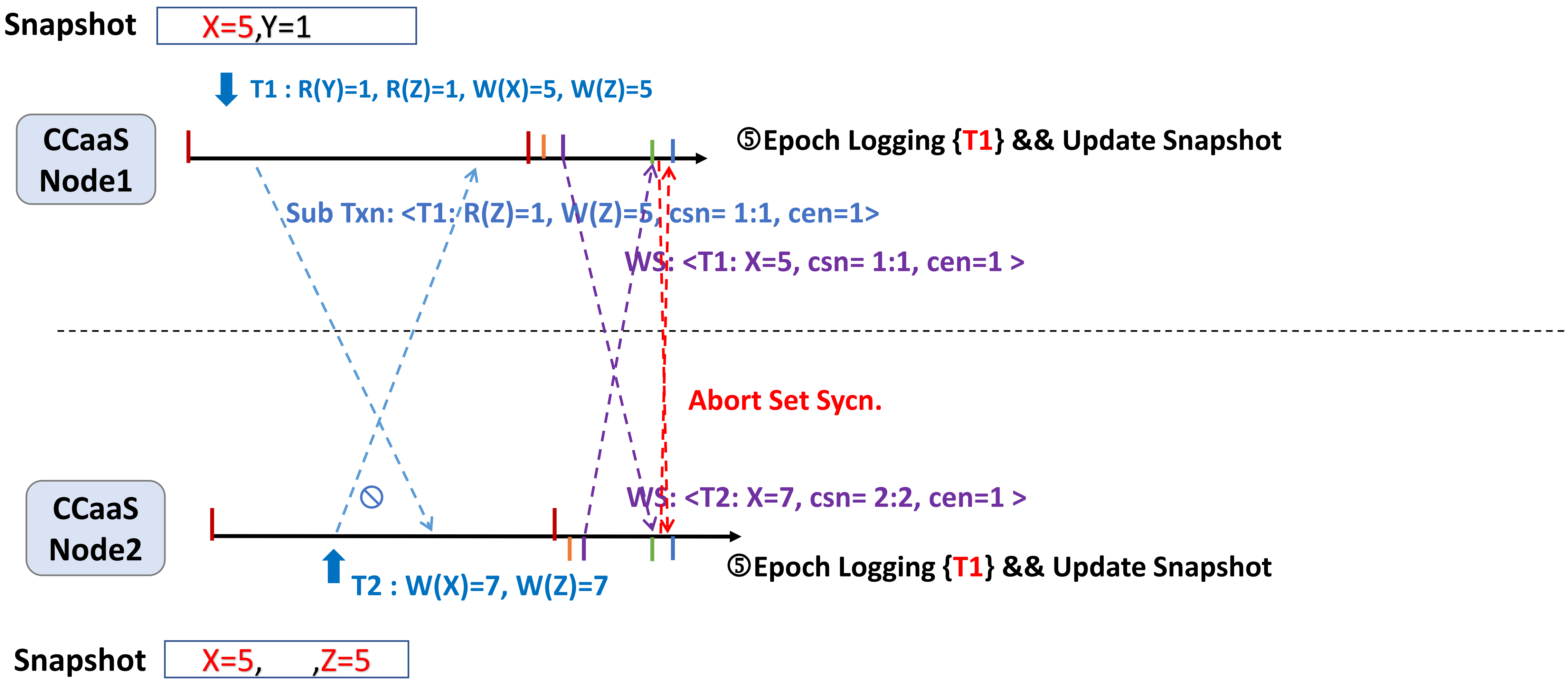
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1) Remote(T2)
Shard Y
Remote(T2) Remote(T3)
Abort Set {T2}
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3) Remote(T2)
Shard Z
Local(T3) Remote(T1’) Remote(T2)
Abort Set {T2,T3}
Abort Set {T2,T3}
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Conflict Handler
Shard X
Local(T2) Remote(T1)
Shard Z
Local(T2) Local(T1’) Remote(T3)
Snapshot
X=1, ,Z=1

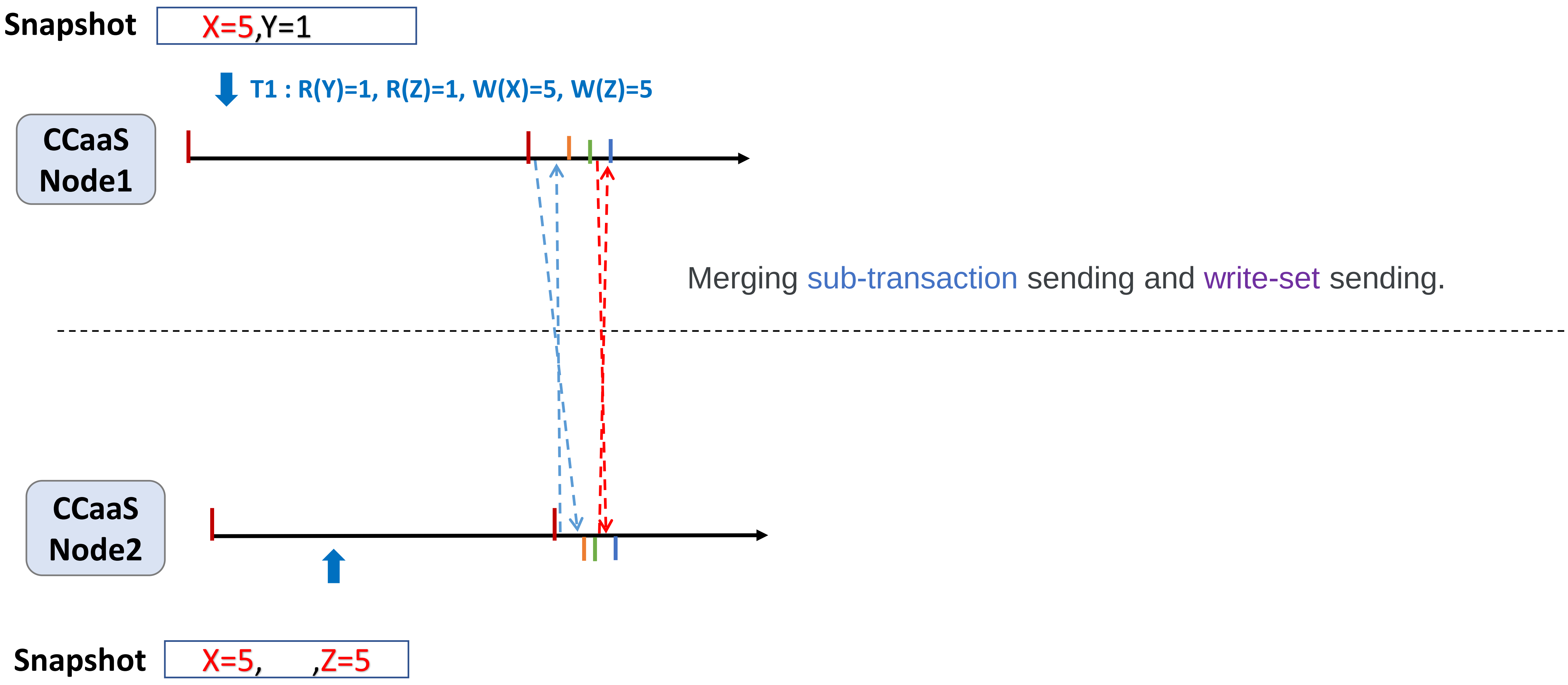
SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1)
Remote(T2)
Shard Y
Remote(T2)
Remote(T3)
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3)
Remote(T2)
Shard Z
Local(T3)
Remote(T1')
Remote(T2)
Abort Set {T2,T3}
Abort Set {T2,T3}
Abort Set Sycn.
Abort Set {T2,T3}
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Conflict Handler
Shard X
Local(T2)
Remote(T1)
Shard Z
Local(T2)
Local(T1')
Remote(T3)
Snapshot
X=1, ,Z=1

SM-OCC

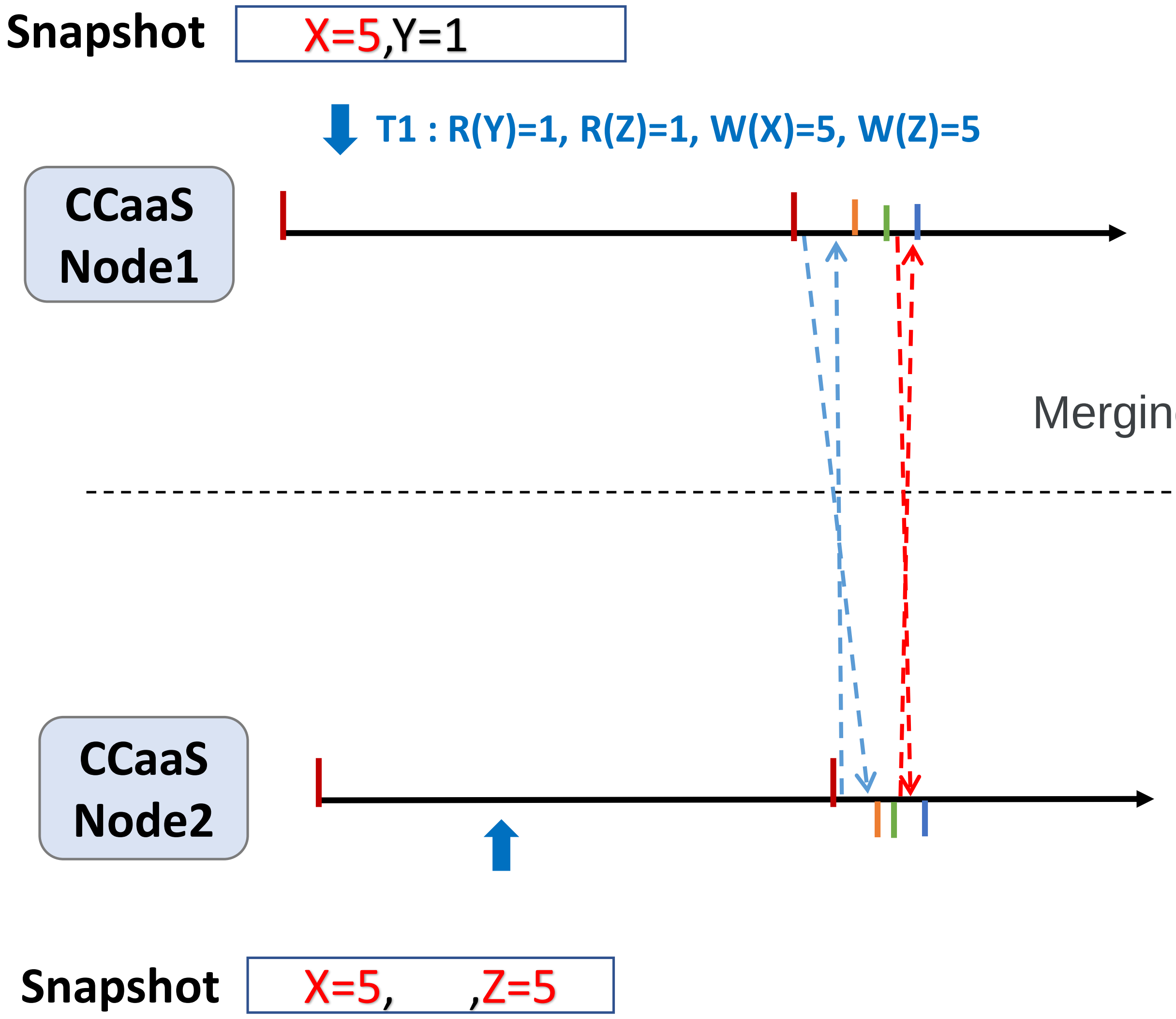

Snapshot
X=5,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
⑤Epoch Logging {T1} && Update Snapshot
Sub Txn: <T1: R(Z)=1, W(Z)=5, csn= 1:1, cen=1>
WS: <T1: X=5, csn= 1:1, cen=1 >
Abort Set Sycn.
WS: <T2: X=7, csn= 2:2, cen=1 >
CCaaS Node2
⑤Epoch Logging {T1} && Update Snapshot
T2 : W(X)=7, W(Z)=7
Snapshot
X=5, ,Z=5

SM-OCC



Merging sub-transaction sending and write-set sending.

SM-OCC



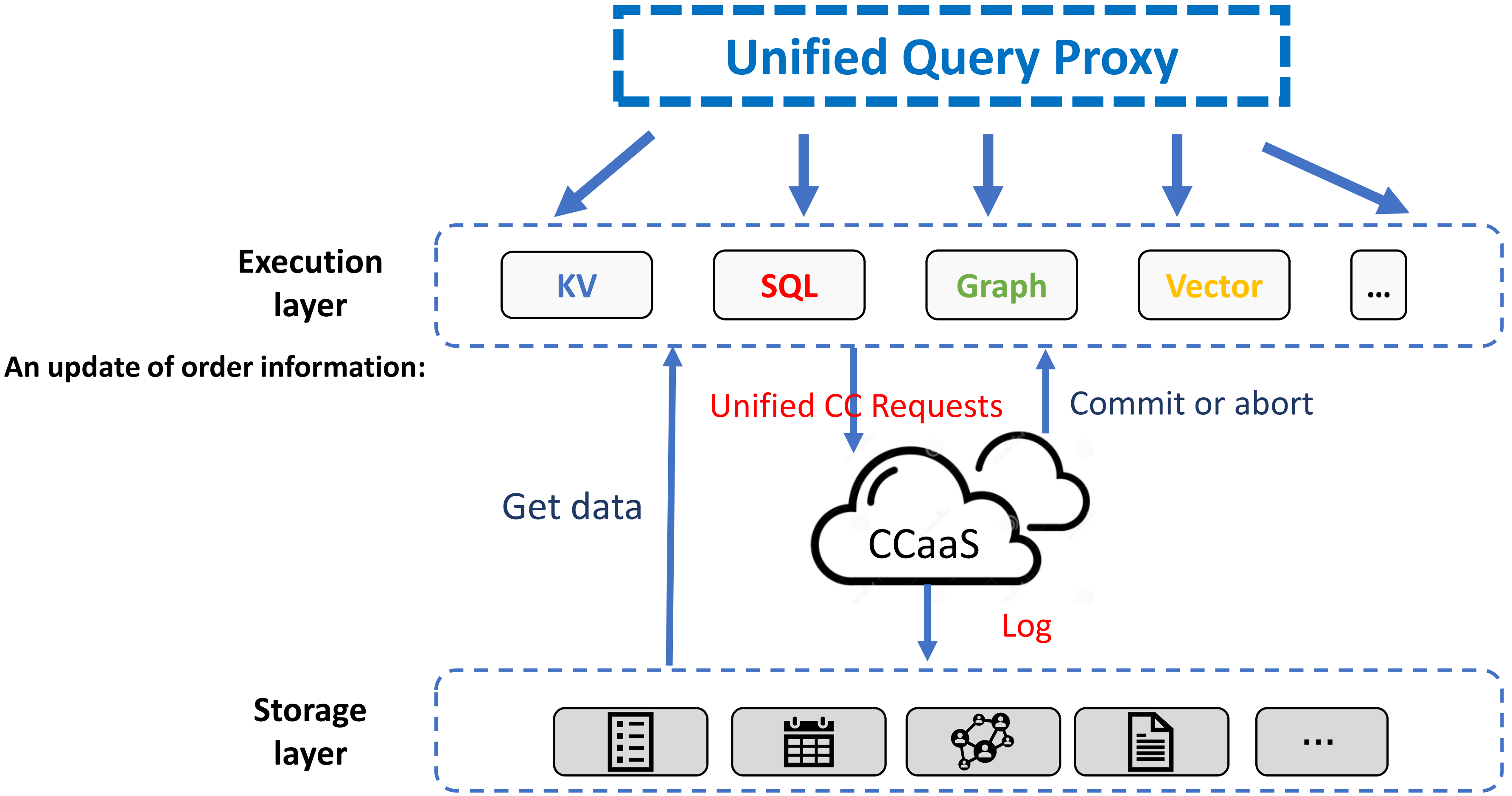
Merging sub-transaction sending and write-set sending.

Less Communication frequency

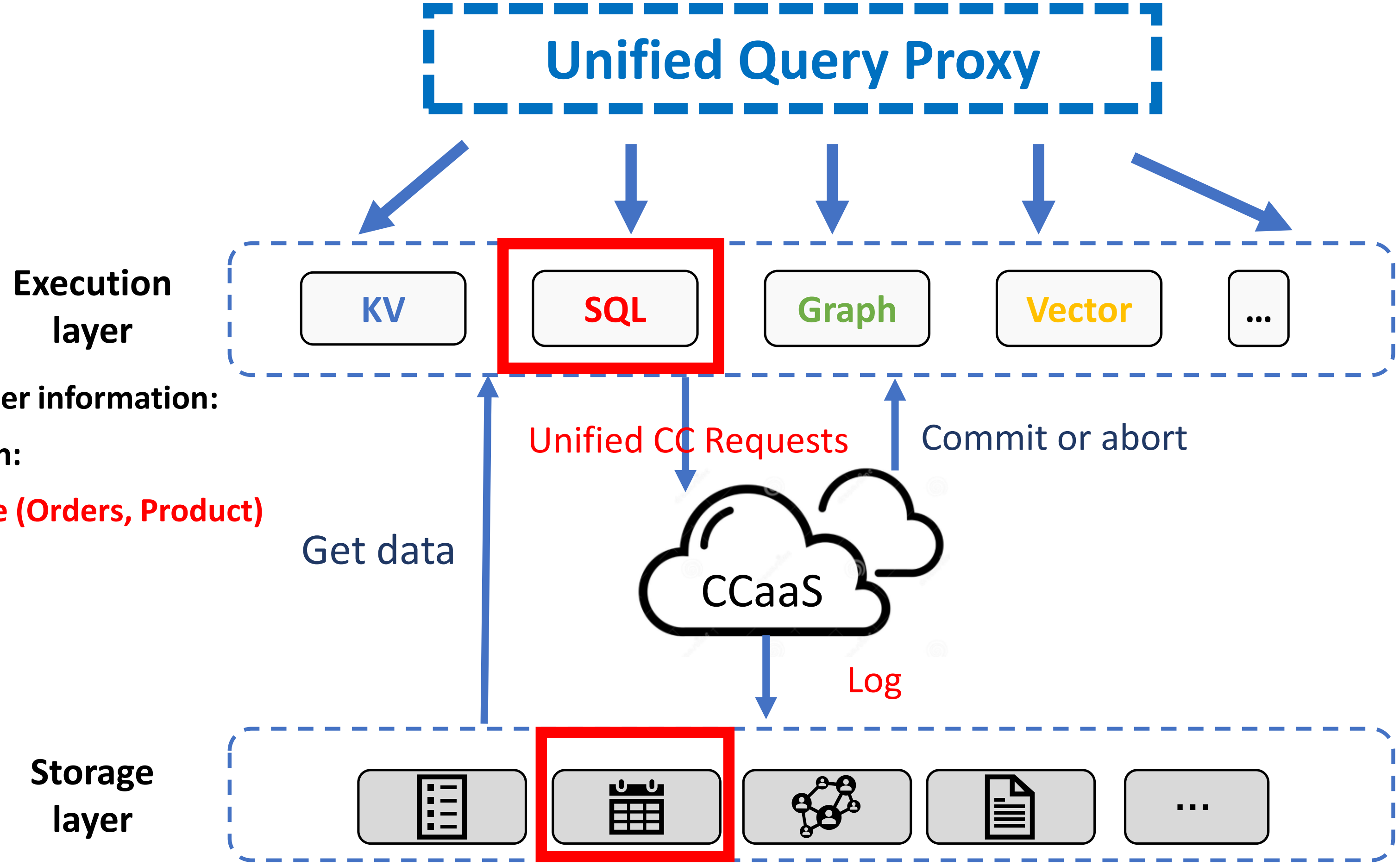
Case Studies

- **Supporting Cross-Model Transactions**

Cross-Model Transaction with CCaaS



Cross-Model Transaction with CCaaS

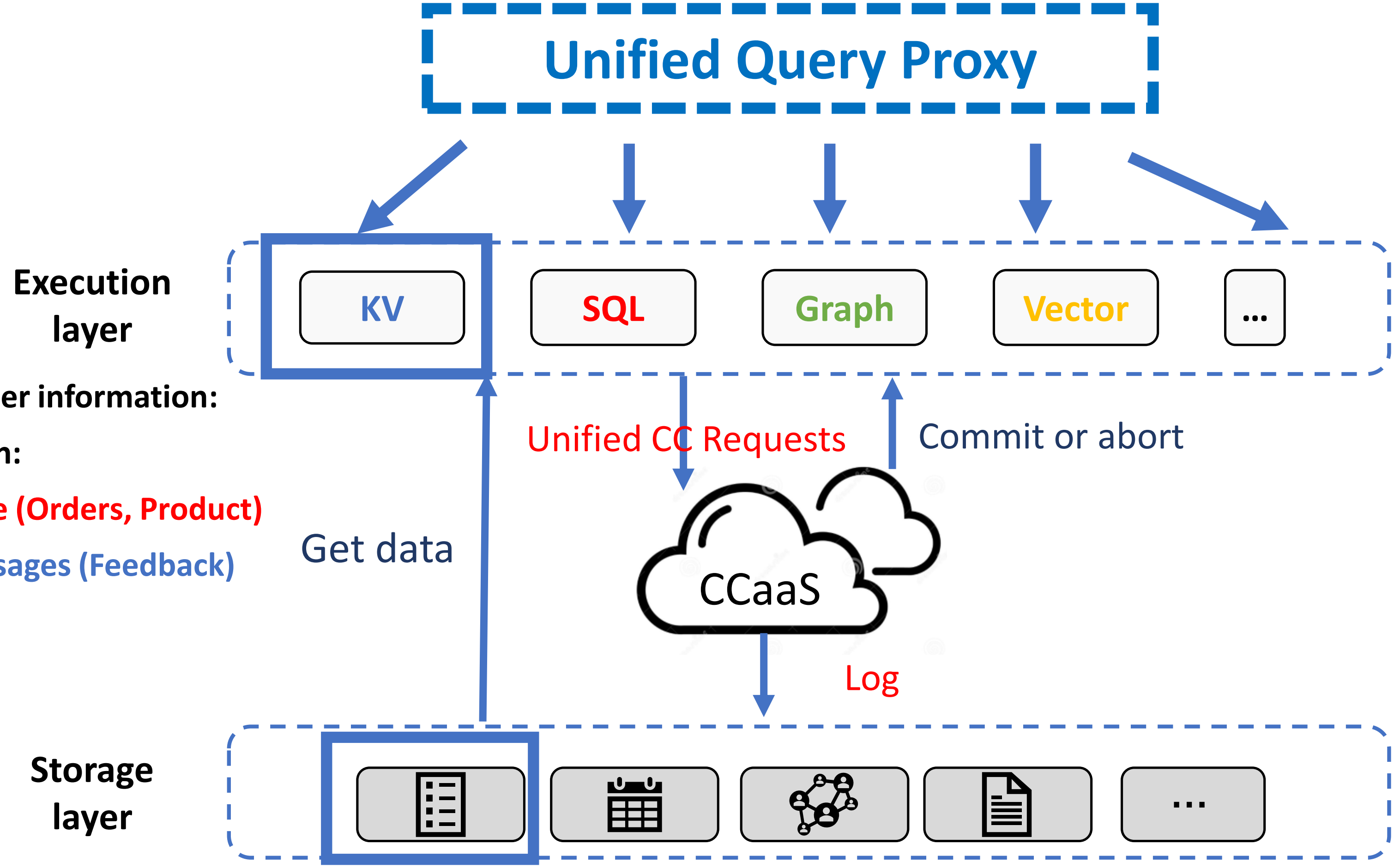


An update of order information:

Begin Transaction:

relational table (Orders, Product)

Cross-Model Transaction with CCaaS



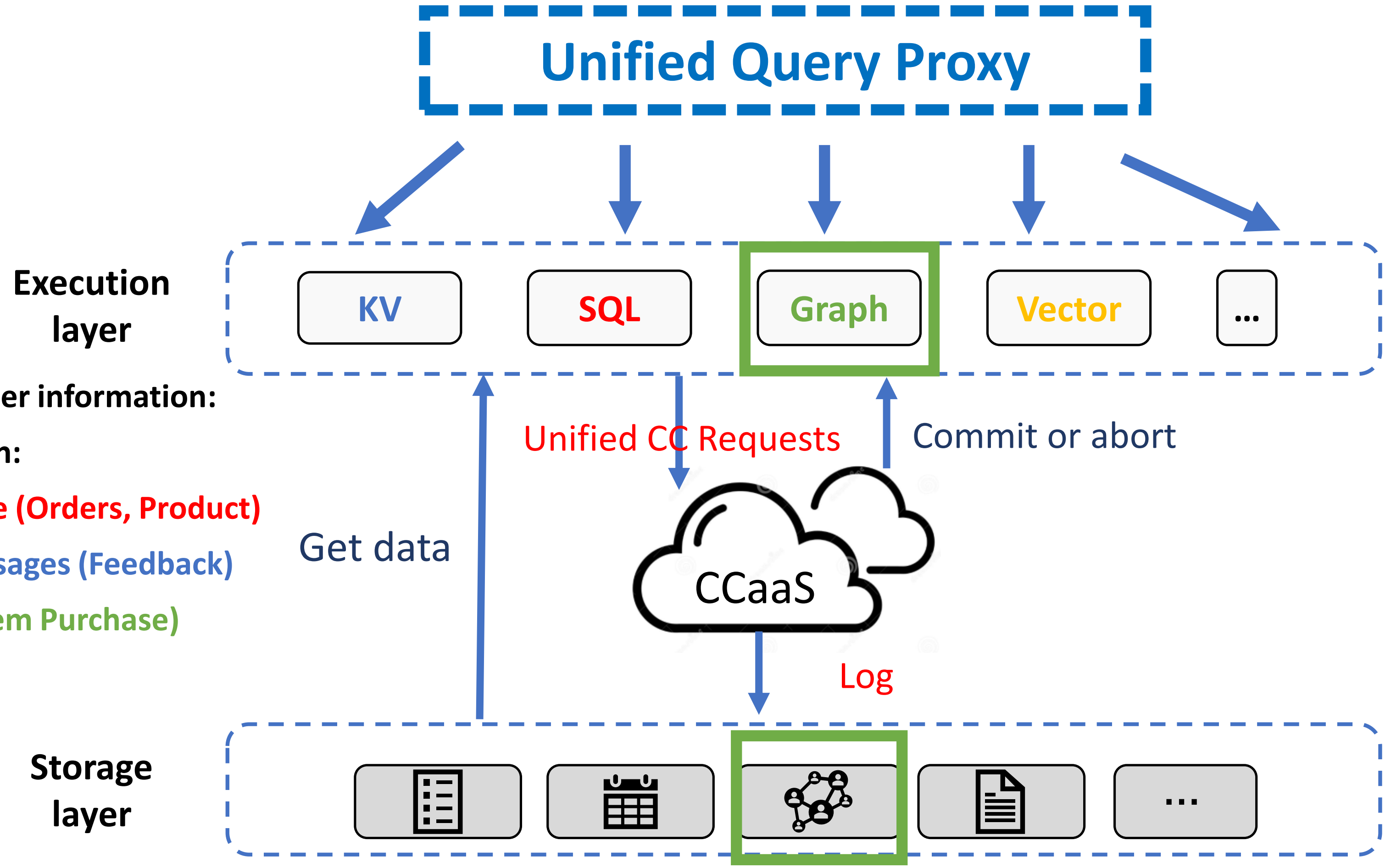
An update of order information:

Begin Transaction:

relational table (Orders, Product)

key-value messages (Feedback)

Cross-Model Transaction with CCaaS



An update of order information:

Begin Transaction:

- **relational table (Orders, Product)**
- **key-value messages (Feedback)**
- **graph (User-Item Purchase)**

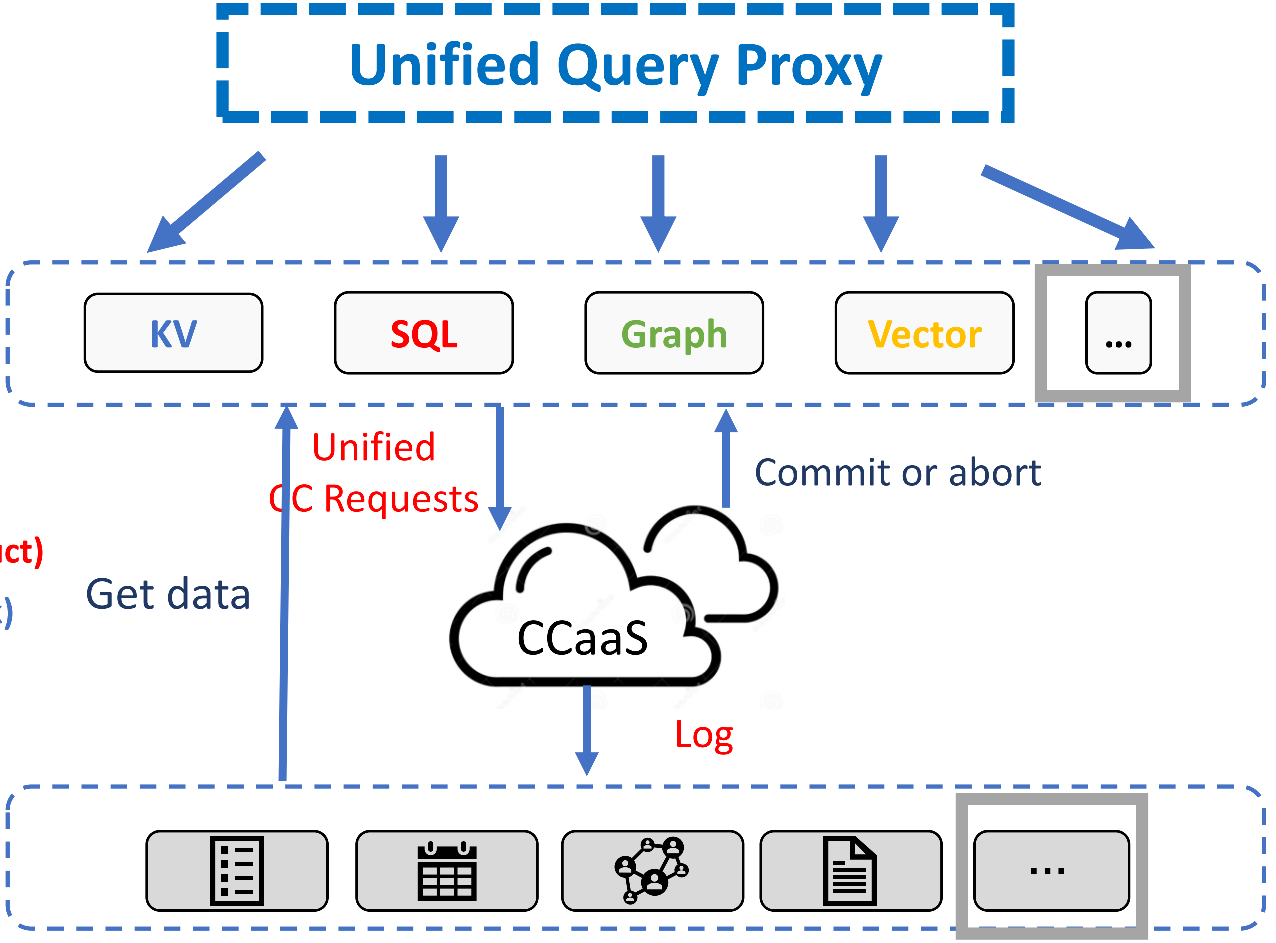
Cross-Model Transaction with CCaaS

Execution layer

An update of order information:

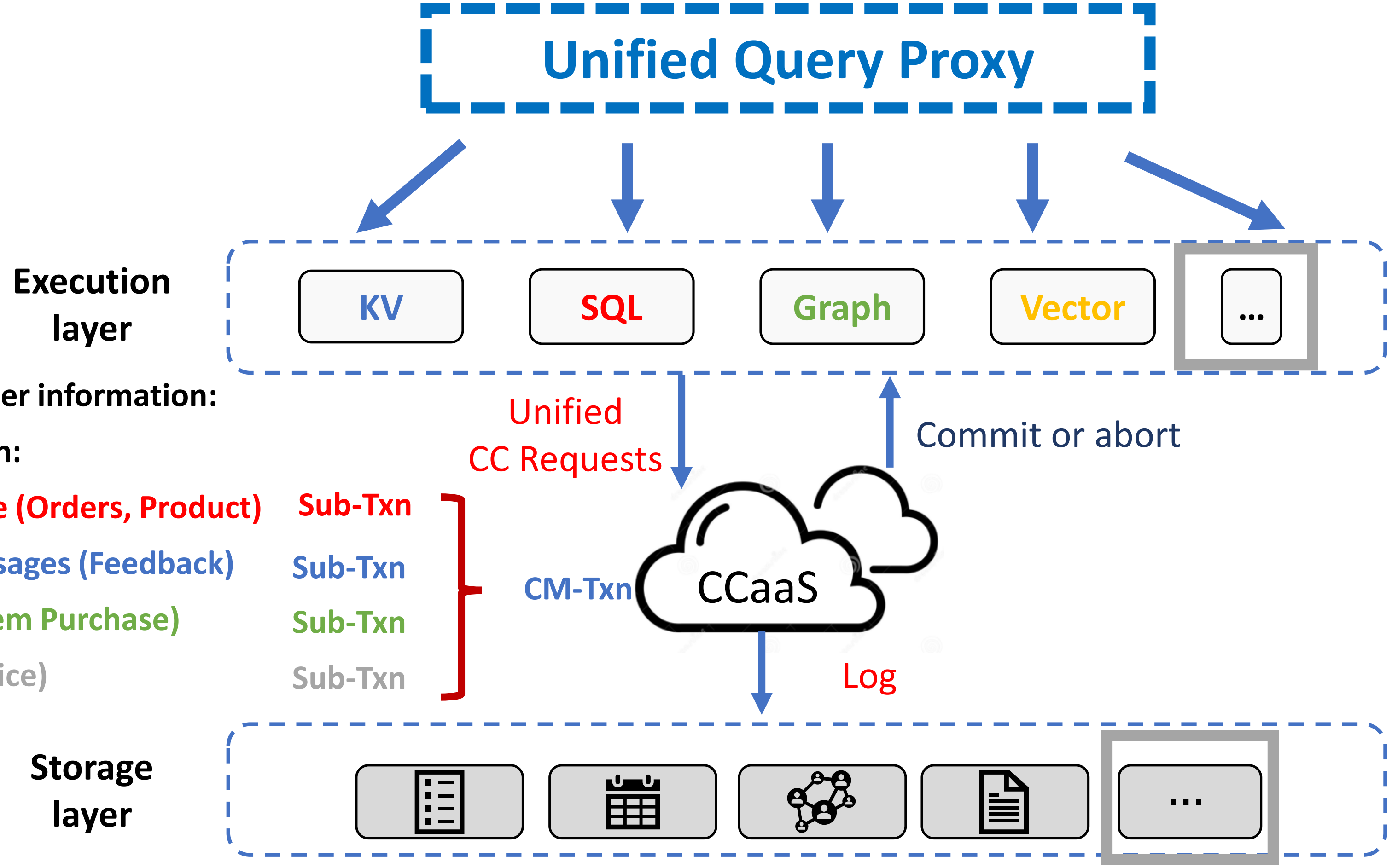
Begin Transaction:

- relational table (Orders, Product)
- key-value messages (Feedback)
- graph (User-Item Purchase)
- XML data (Invoice)



Storage layer

Cross-Model Transaction with CCaaS



An update of order information:

Begin Transaction:

- **relational table (Orders, Product)**
- **key-value messages (Feedback)**
- **graph (User-Item Purchase)**
- XML data (Invoice)

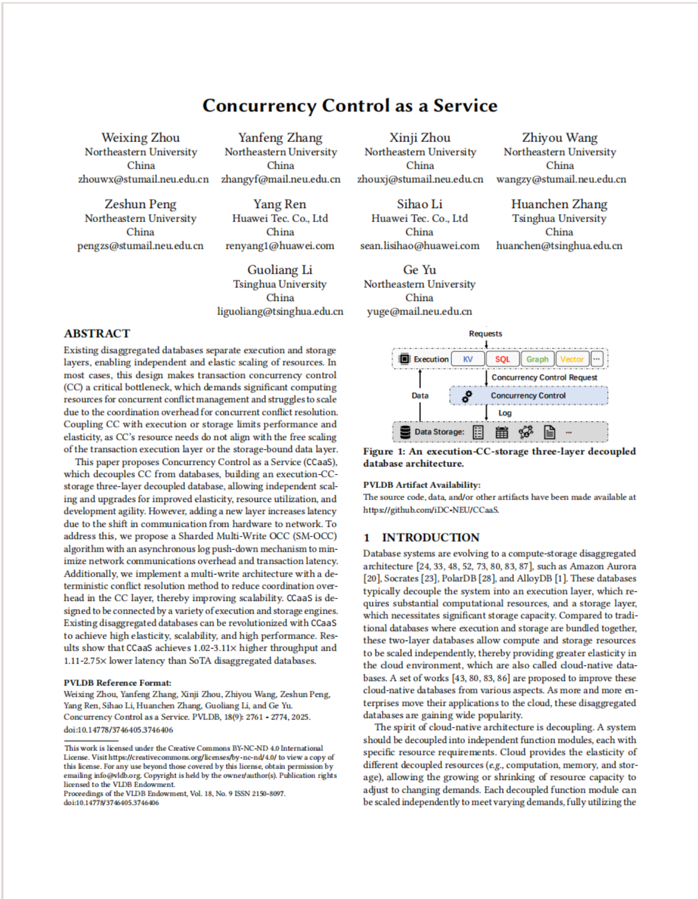
Commit;

Case Studies

- **Supporting Cross-Model Transactions**
- **Empowering NoSQL DBs with TP Capability**
- **Making Standalone TP Distributed**
- **…**

More Details

- **More:**
  - **Sharding**
  - **Isolation**
  - **Logging**
  - **Fault Tolerance**
  - **Cross-Model**
  - **…**



Concurrency Control as a Service

Weixing Zhou, Northeastern University, China, zhouwx@stumail.neu.edu.cn
Yanfeng Zhang, Northeastern University, China, zhangyf@mail.neu.edu.cn
Xinji Zhou, Northeastern University, China, zhouxj@stumail.neu.edu.cn
Zhiyou Wang, Northeastern University, China, wangzy@stumail.neu.edu.cn
Zeshun Peng, Northeastern University, China, pengzs@stumail.neu.edu.cn
Yang Ren, Huawei Tec. Co., Ltd, China, renyang1@huawei.com
Sihao Li, Huawei Tec. Co., Ltd, China, sean.lisihao@huawei.com
Huanchen Zhang, Tsinghua University, China, huanchen@tsinghua.edu.cn
Guoliang Li, Tsinghua University, China, liguoliang@tsinghua.edu.cn
Ge Yu, Northeastern University, China, yuge@mail.neu.edu.cn

ABSTRACT

Existing disaggregated databases separate execution and storage layers, enabling independent and elastic scaling of resources. In most cases, this design makes transaction concurrency control (CC) a critical bottleneck, which demands significant computing resources for concurrent conflict management and struggles to scale due to the coordination overhead for concurrent conflict resolution. Coupling CC with execution or storage limits performance and elasticity, as CC's resource needs do not align with the free scaling of the transaction execution layer or the storage-bound data layer.

This paper proposes Concurrency Control as a Service (CCaaS), which decouples CC from databases, building an execution-CC-storage three-layer decoupled database, allowing independent scaling and upgrades for improved elasticity, resource utilization, and development agility. However, adding a new layer increases latency due to the shift in communication from hardware to network. To address this, we propose a Sharded Multi-Write OCC (SM-OCC) algorithm with an asynchronous log push-down mechanism to minimize network communications overhead and transaction latency. Additionally, we implement a multi-write architecture with a deterministic conflict resolution method to reduce coordination overhead in the CC layer, thereby improving scalability. CCaaS is designed to be connected by a variety of execution and storage engines. Existing disaggregated databases can be revolutionized with CCaaS to achieve high elasticity, scalability, and high performance. Results show that CCaaS achieves 1.02-3.11× higher throughput and 1.11-2.75× lower latency than SoTA disaggregated databases.

PVLDB Reference Format:
Weixing Zhou, Yanfeng Zhang, Xinji Zhou, Zhiyou Wang, Zeshun Peng, Yang Ren, Sihao Li, Huanchen Zhang, Guoliang Li, and Ge Yu. Concurrency Control as a Service. PVLDB, 18(9): 2761 - 2774, 2025.
doi:10.14778/3746405.3746406

Proceedings of the VLDB Endowment, Vol. 18, No. 9 ISSN 2150-8097.
doi:10.14778/3746405.3746406

Figure 1: An execution-CC-storage three-layer decoupled database architecture.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/iDC-NEU/CCaaS.

1 INTRODUCTION

Database systems are evolving to a compute-storage disaggregated architecture [24, 33, 48, 52, 73, 80, 83, 87], such as Amazon Aurora [20], Socrates [23], PolarDB [28], and AlloyDB [1]. These databases typically decouple the system into an execution layer, which requires substantial computational resources, and a storage layer, which necessitates significant storage capacity. Compared to traditional databases where execution and storage are bundled together, these two-layer databases allow compute and storage resources to be scaled independently, thereby providing greater elasticity in the cloud environment, which are also called cloud-native databases. A set of works [43, 80, 83, 86] are proposed to improve these cloud-native databases from various aspects. As more and more enterprises move their applications to the cloud, these disaggregated databases are gaining wide popularity.

The spirit of cloud-native architecture is decoupling. A system should be decoupled into independent function modules, each with specific resource requirements. Cloud provides the elasticity of different decoupled resources (*e.g.*, computation, memory, and storage), allowing the growing or shrinking of resource capacity to adjust to changing demands. Each decoupled function module can be scaled independently to meet varying demands, fully utilizing the

Evaluation

- Cluster Set Up:
  - 3 nodes in each layer,
  - 16 vCPUs, 32G DRAM, Ubuntu 22.04 LTS,
  - 2.5Gbps, ~2ms latency
- Benchmark:
  - YCSB[1]
    - YCSB-A (50% read,50 Write, $\theta = 0.99$)
    - YCSB-B (95% read, 5% write, $\theta = 0.99$)
  - TPC-C[2]

1: YCSB 10 op/txn
2: 50% New-Order – 50% Payment in GeoGauss[SIGMOD 2023]

Evaluation

- Scalability & Elasticity
  - Coupling CC with the execution VS Decoupled CC
  - Dynamically adjust the number of execution and CC nodes

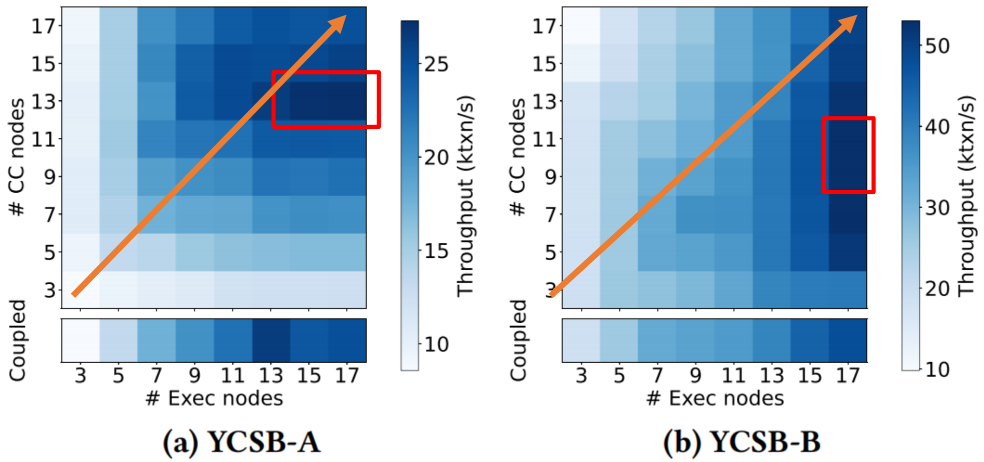


Figure 15: Throughput when scaling the number of execution nodes and CC nodes in CCaaS.

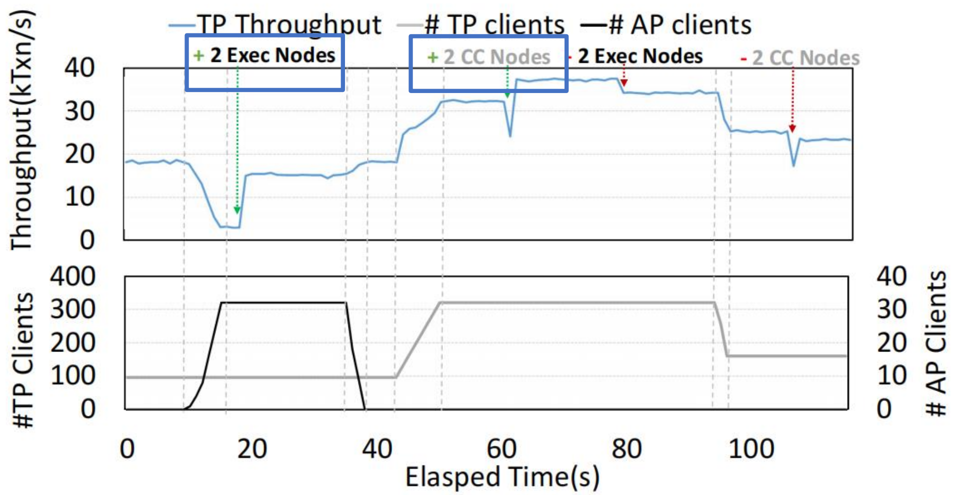


Figure 16: Elasticity performance under changing workloads.

Evaluation

- Performance
  - Throughput 1.02-3.11 X
  - Latency 1.11-2.75 X

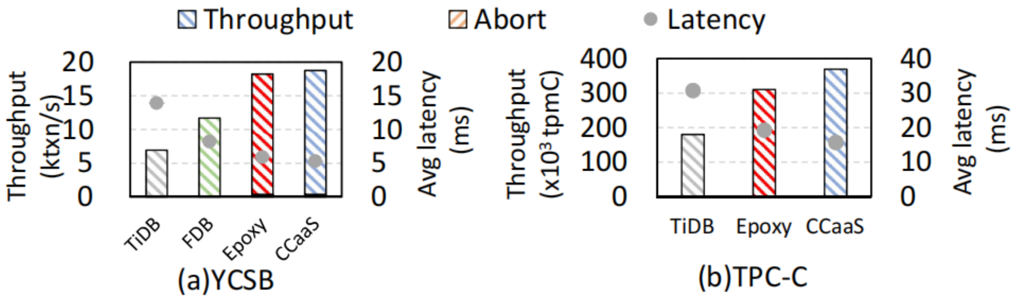


Figure 13: Experiment results with competitors.

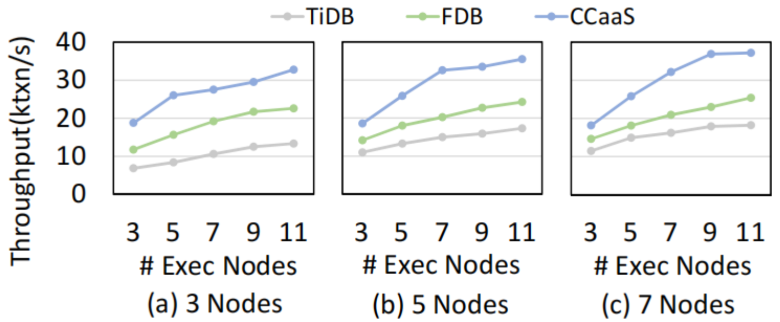


Figure 14: Comparison with existing disaggregated databases under YCSB-B workload.

Evaluation

- Servitization
  - Empowering TP Capability to NoSQL DBs
  - Distributed TP with standalone TP engines
  - Cross-Model Transactions

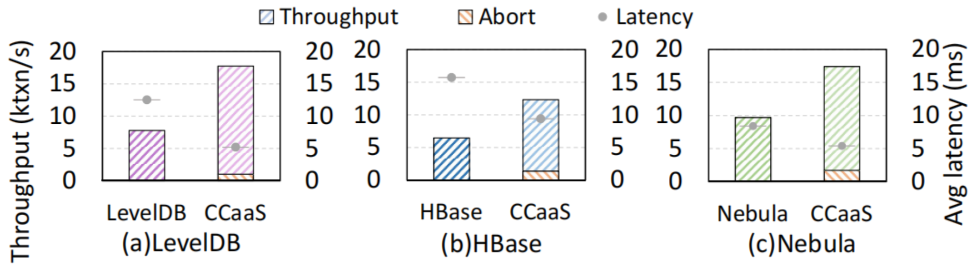


Figure 17: Performance of NoSQL DBs empowered with ACID TP capability.

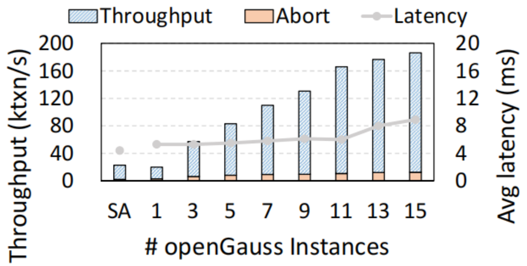


Figure 18: Performance of serving standalone TP engines.

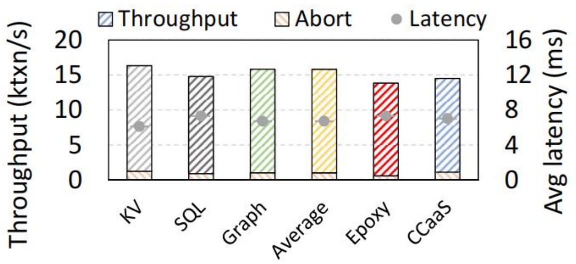


Figure 19: Performance of cross-model (CM) transactions.

Concurrency Control as a Service

- Decoupled **execution-transaction-storage three-layer** cloud-native database architecture
  - High scalability, elasticity
  - High development agility
- Concurrency Control **Service**
  - Empowering NoSQL DBs with TP Capability
  - Making Standalone TP Distributed
  - Supporting Cross-Model Transactions
- **Sharded Multi-Write Optimistic Concurrency Control** Algorithm
  - High availability
  - Low communication costs

https://github.com/iDC-NEU/CCaaS

Wechat

Thank you! Q&A

Concurrency Control as a Service

- Scalability

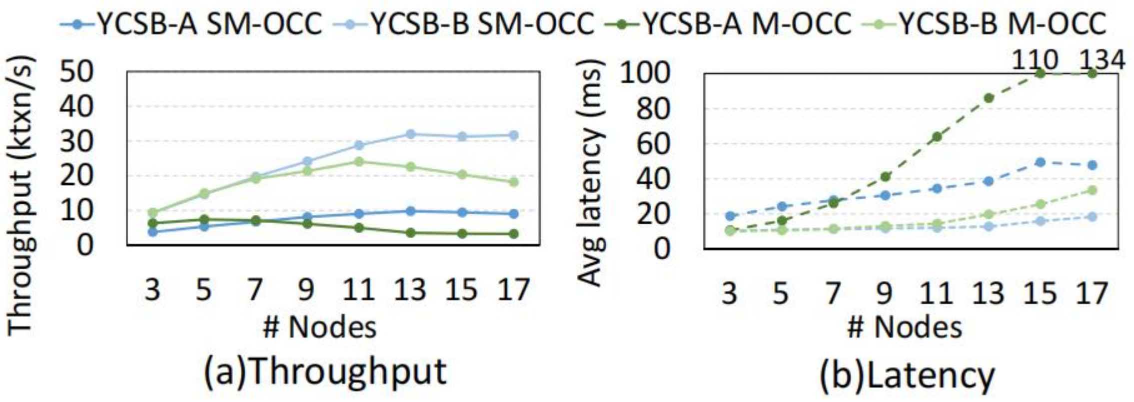


Figure 16: Scaling performance of CCaaS.

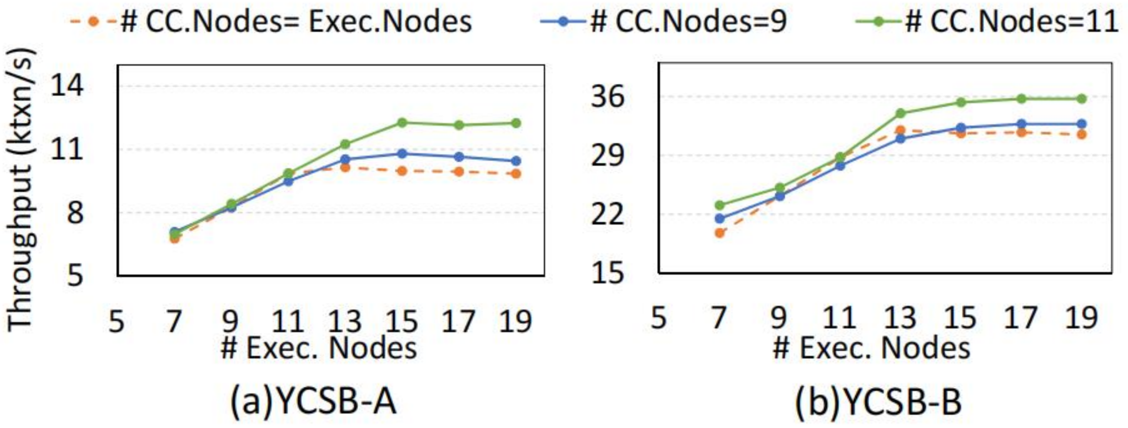


Figure 17: Throughput when varying the number of execution nodes and CCaaS nodes.

Concurrency Control as a Service

- Breakdown

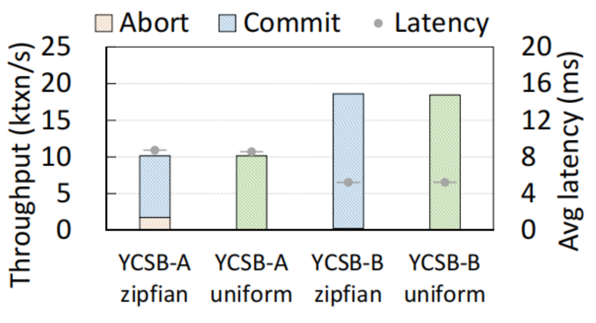


Figure 20: Performance under different contention.

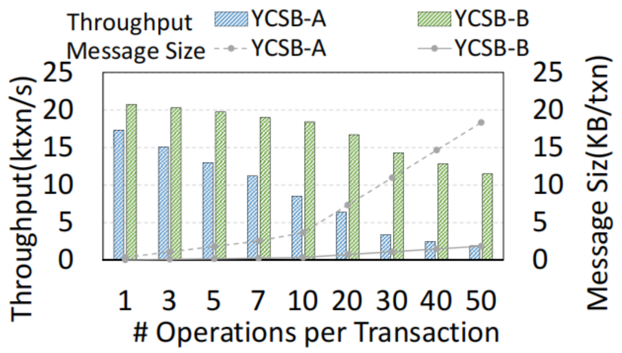


Figure 21: Performance when varying number of operations.

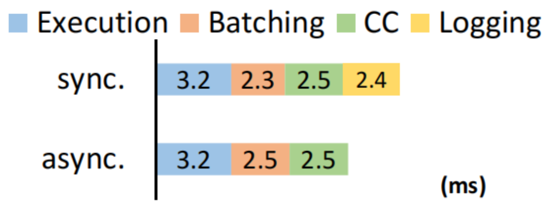


Figure 22: Runtime breakdown (sync./async. logging).

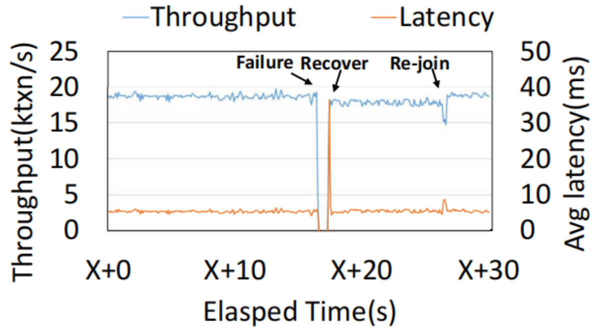
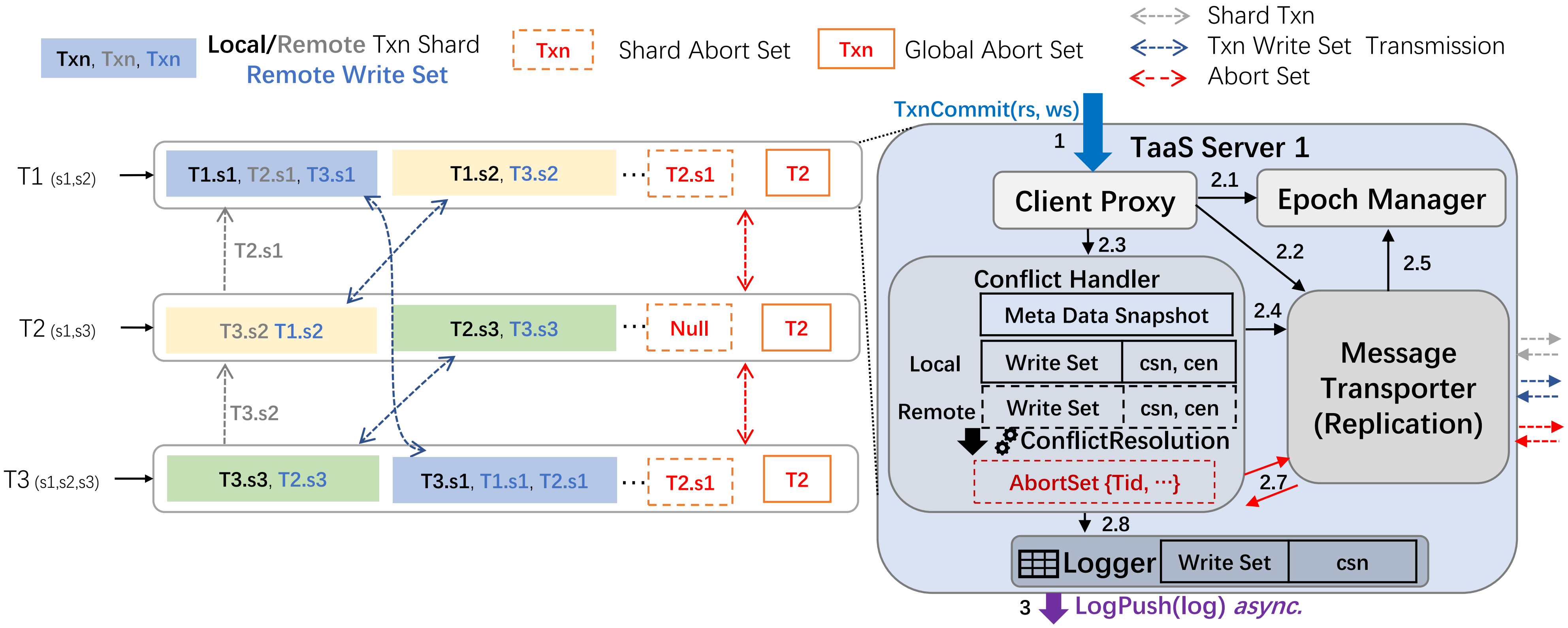
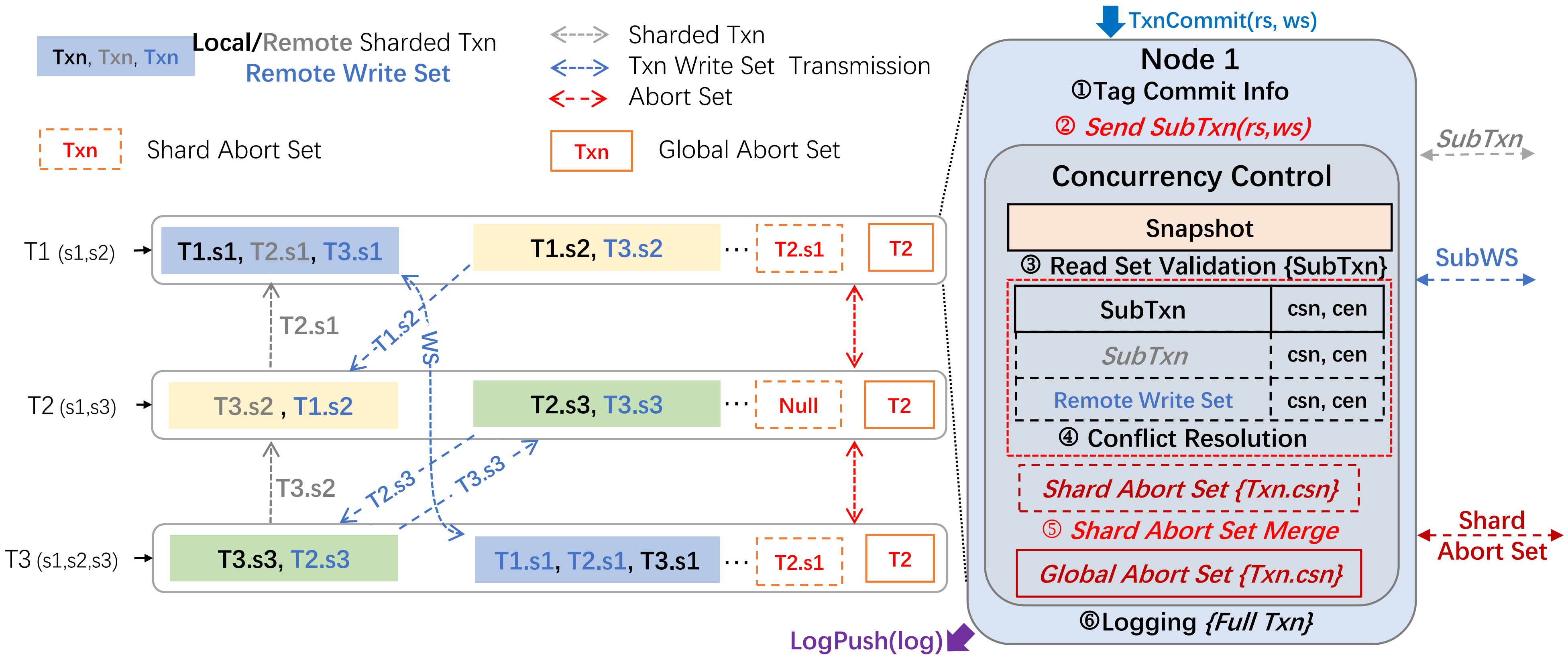


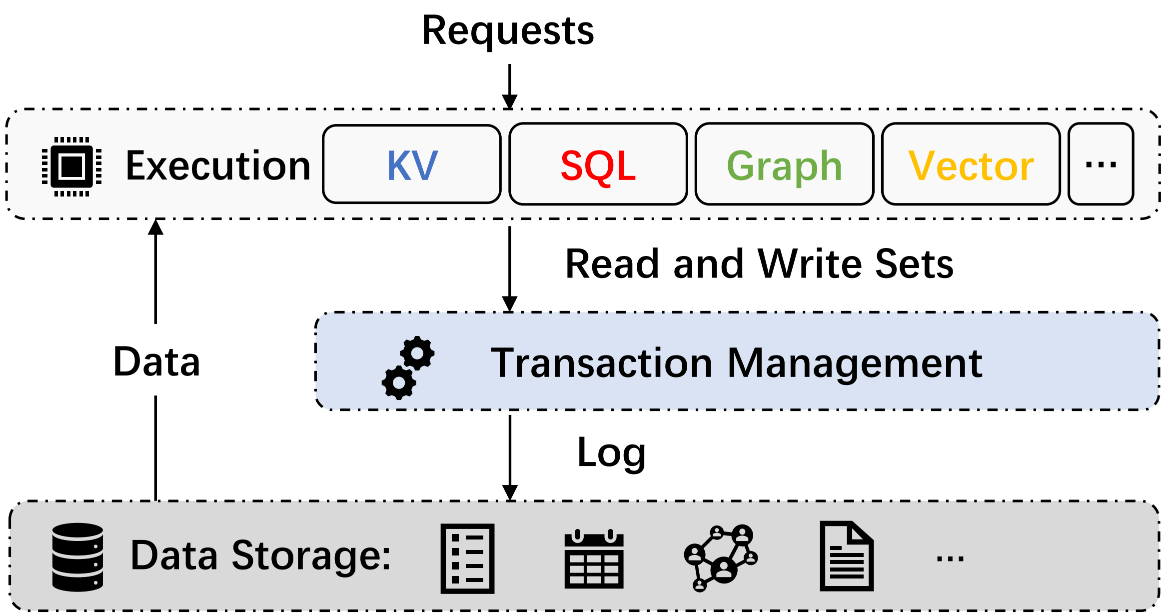
Figure 23: Performance variation when system recovery.

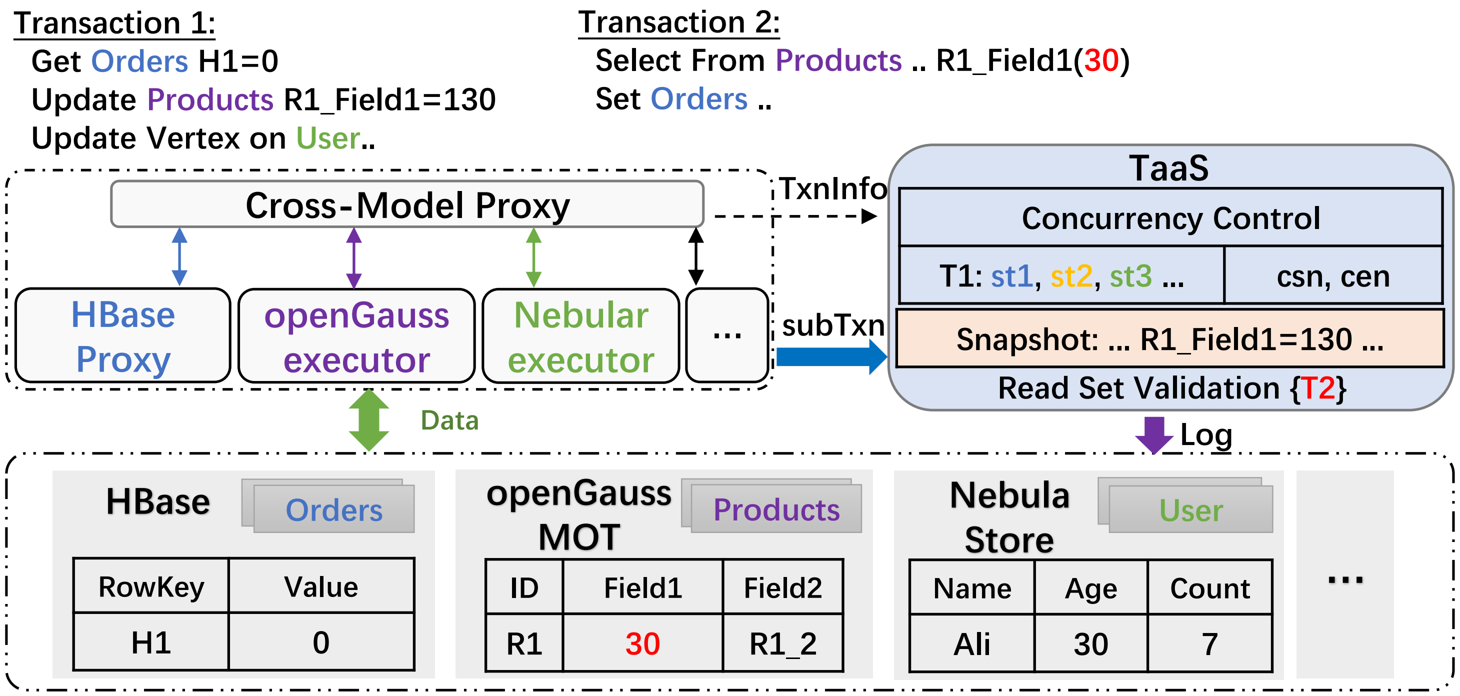


Txn, Txn, Txn
Local/Remote Txn Shard
Remote Write Set
Txn
Shard Abort Set
Txn
Global Abort Set
Shard Txn
Txn Write Set Transmission
Abort Set
TxnCommit(rs, ws)
1
TaaS Server 1
Client Proxy
2.1
Epoch Manager
2.3
2.2
2.5
Conflict Handler
Meta Data Snapshot
2.4
Local
Write Set
csn, cen
Remote
Write Set
csn, cen
ConflictResolution
Message Transporter (Replication)
AbortSet {Tid, ···}
2.7
2.8
Logger
Write Set
csn
3
LogPush(log) async.
T1 (s1,s2)
T1.s1, T2.s1, T3.s1
T1.s2, T3.s2
T2.s1
T2
T2.s1
T2 (s1,s3)
T3.s2 T1.s2
T2.s3, T3.s3
Null
T2
T3.s2
T3 (s1,s2,s3)
T3.s3, T2.s3
T3.s1, T1.s1, T2.s1
T2.s1
T2

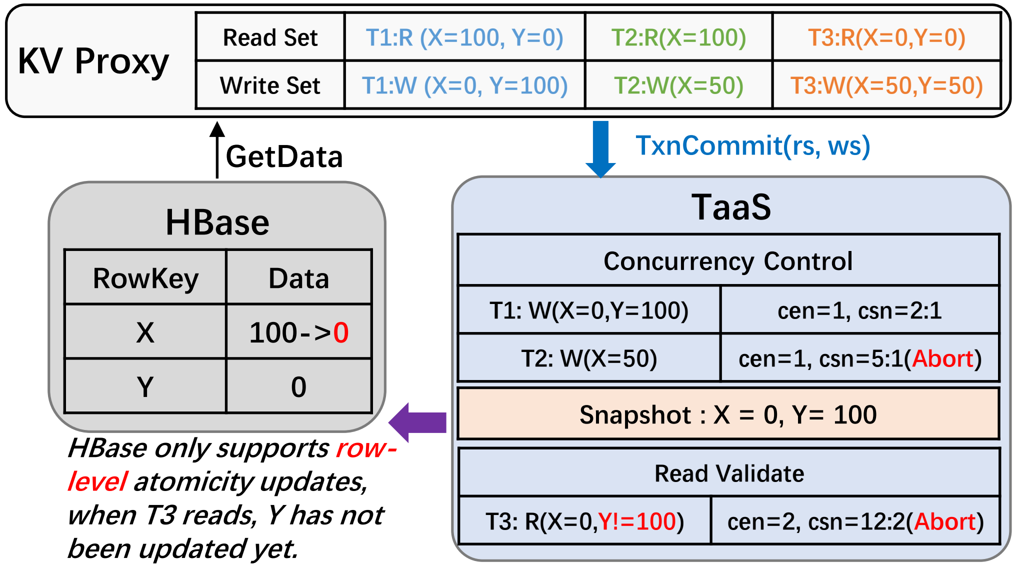


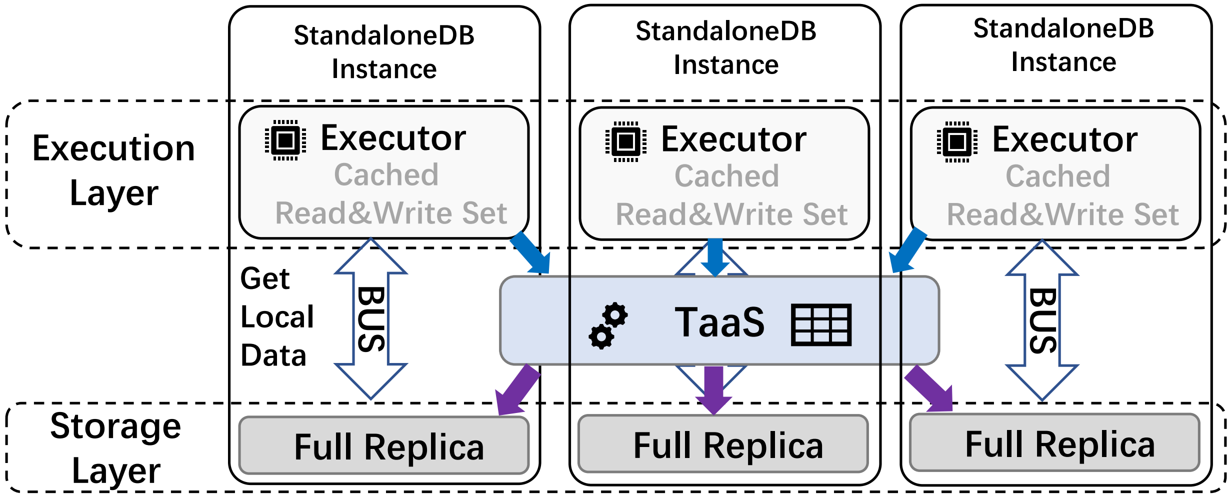
Txn, Txn, Txn
Local/Remote Sharded Txn
Remote Write Set
Sharded Txn
Txn Write Set Transmission
Abort Set
Txn
Shard Abort Set
Txn
Global Abort Set
T1 (s1,s2)
T1.s1, T2.s1, T3.s1
T1.s2, T3.s2
T2.s1
T2
T2.s1
T1.s2
WS
T2 (s1,s3)
T3.s2 , T1.s2
T2.s3, T3.s3
Null
T2
T3.s2
T2.s3
T3.s3
T3 (s1,s2,s3)
T3.s3, T2.s3
T1.s1, T2.s1, T3.s1
T2.s1
T2
TxnCommit(rs, ws)
Node 1
①Tag Commit Info
② Send SubTxn(rs,ws)
Concurrency Control
Snapshot
③ Read Set Validation {SubTxn}
SubTxn
csn, cen
SubTxn
csn, cen
Remote Write Set
csn, cen
④ Conflict Resolution
Shard Abort Set {Txn.csn}
⑤ Shard Abort Set Merge
Global Abort Set {Txn.csn}
⑥Logging {Full Txn}
LogPush(log)
SubTxn
SubWS
Shard
Abort Set


Requests
Execution
KV
SQL
Graph
Vector
···
Read and Write Sets
Data
Transaction Management
Log
Data Storage:
···



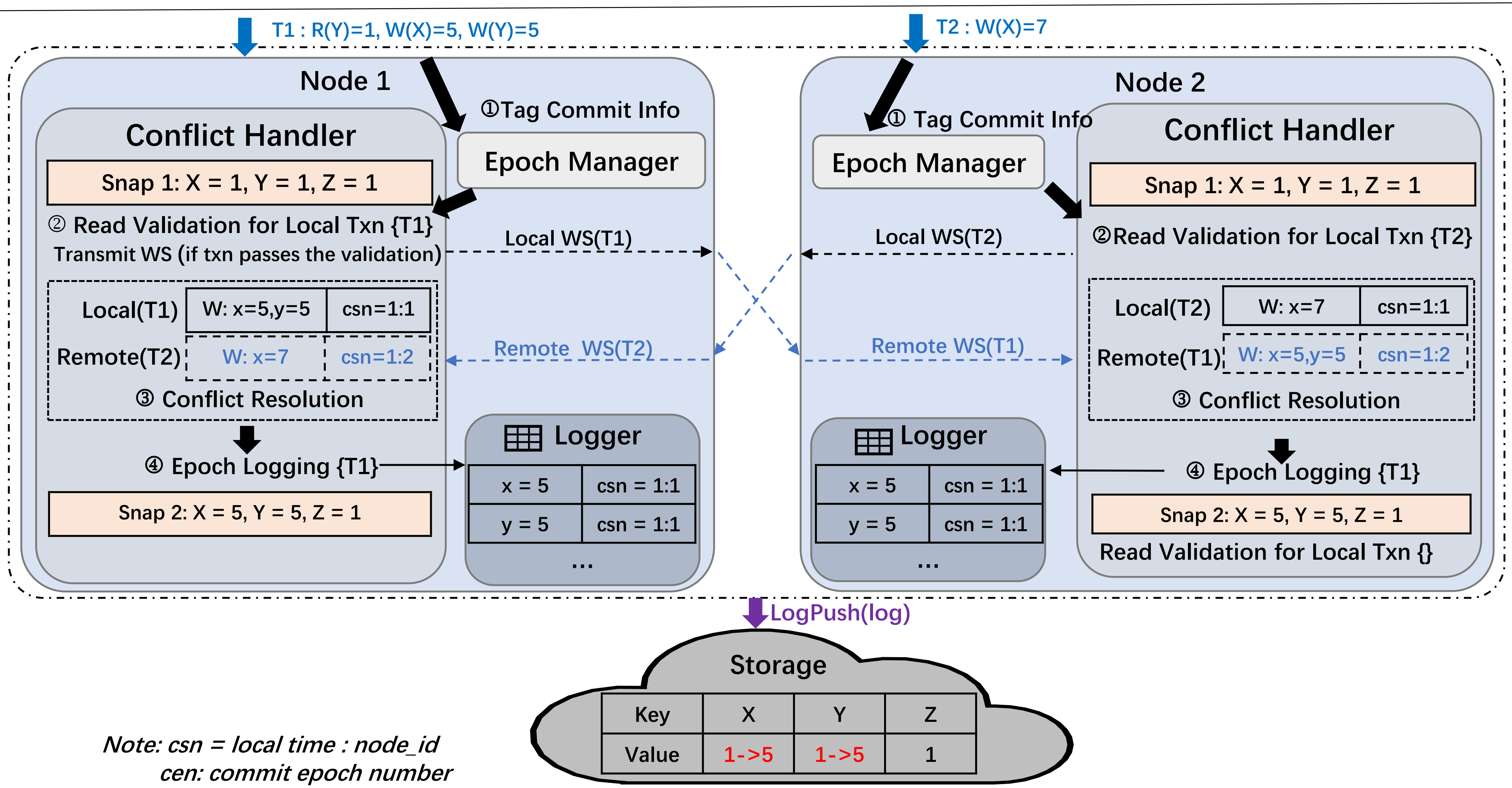
Transaction 1:
Get Orders H1=0
Update Products R1_Field1=130
Update Vertex on User..
Transaction 2:
Select From Products .. R1_Field1(30)
Set Orders ..
Cross-Model Proxy
HBase Proxy
openGauss executor
Nebular executor
...
TxnInfo
subTxn
TaaS
Concurrency Control
T1: st1, st2, st3 ...
csn, cen
Snapshot: ... R1_Field1=130 ...
Read Set Validation {T2}
Data
Log
HBase
Orders
RowKey
Value
H1
0
openGauss MOT
Products
ID
Field1
Field2
R1
30
R1_2
Nebula Store
User
Name
Age
Count
Ali
30
7
...


KV Proxy
Read Set
T1:R (X=100, Y=0)
T2:R(X=100)
T3:R(X=0,Y=0)
Write Set
T1:W (X=0, Y=100)
T2:W(X=50)
T3:W(X=50,Y=50)
GetData
TxnCommit(rs, ws)
HBase
RowKey
Data
X
100->0
Y
0
TaaS
Concurrency Control
T1: W(X=0,Y=100)
cen=1, csn=2:1
T2: W(X=50)
cen=1, csn=5:1(Abort)
Snapshot : X = 0, Y= 100
Read Validate
T3: R(X=0,Y!=100)
cen=2, csn=12:2(Abort)
HBase only supports row-level atomicity updates, when T3 reads, Y has not been updated yet.



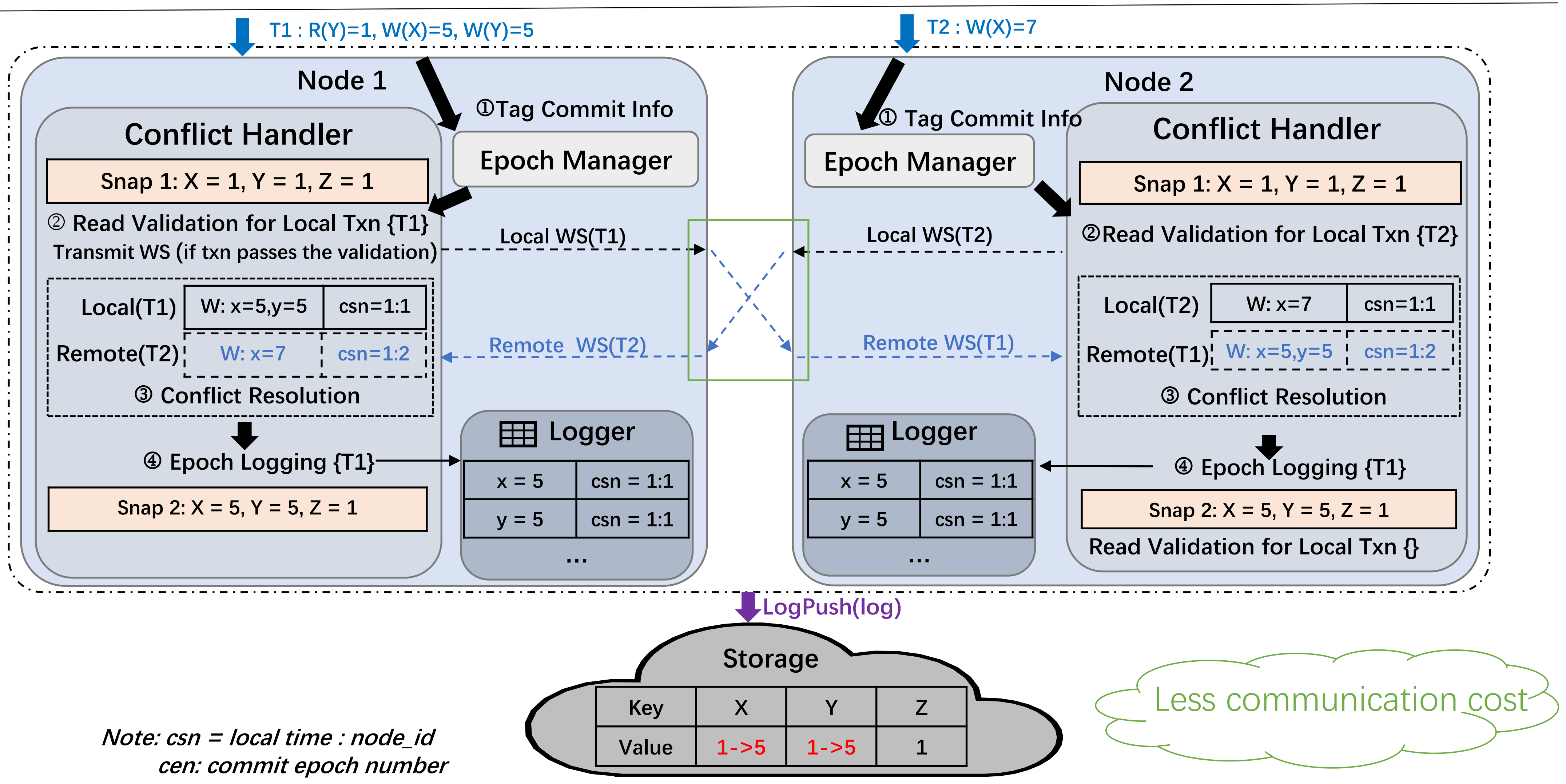
StandaloneDB Instance
StandaloneDB Instance
StandaloneDB Instance
Execution Layer
Executor
Cached Read&Write Set
Executor
Cached Read&Write Set
Executor
Cached Read&Write Set
Get Local Data
BUS
TaaS
BUS
Storage Layer
Full Replica
Full Replica
Full Replica

SM-OCC



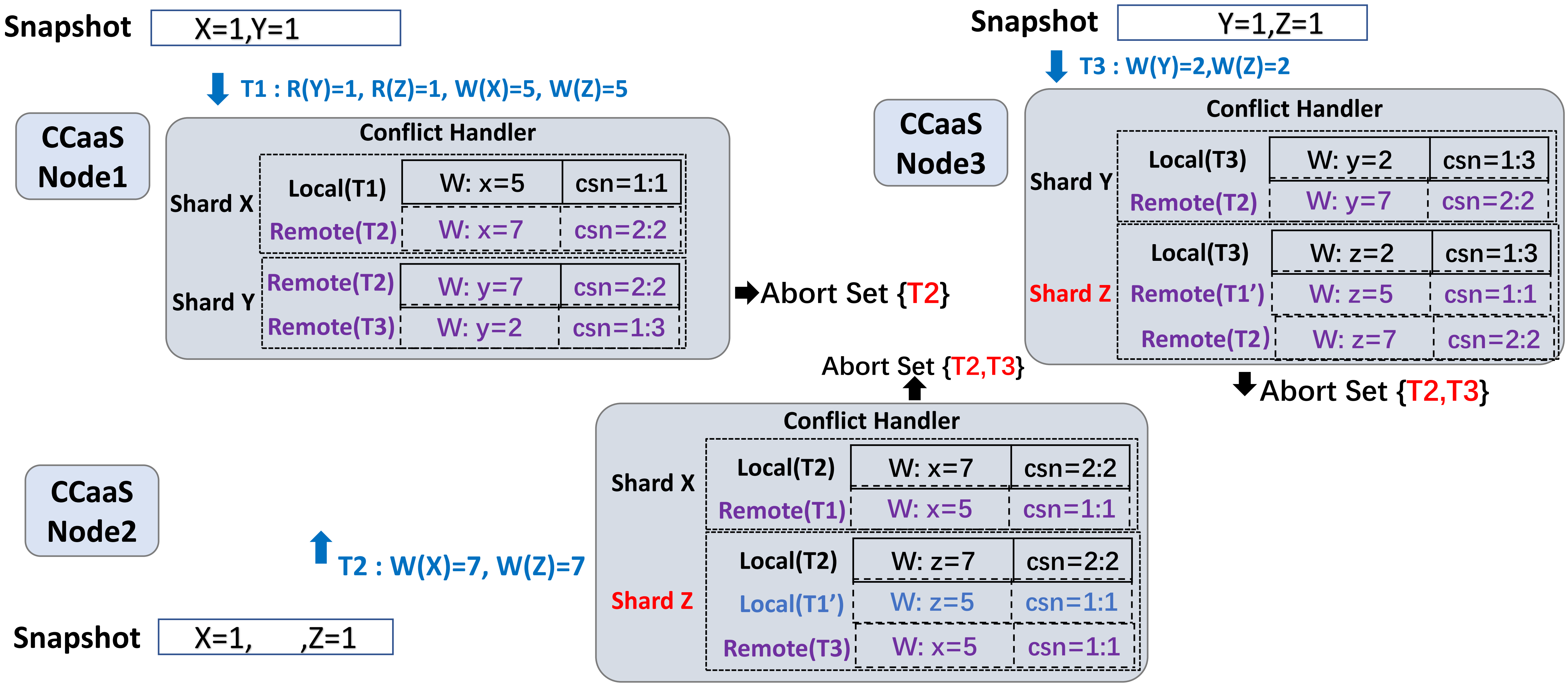
Note: csn = local time : node_id
cen: commit epoch number

SM-OCC

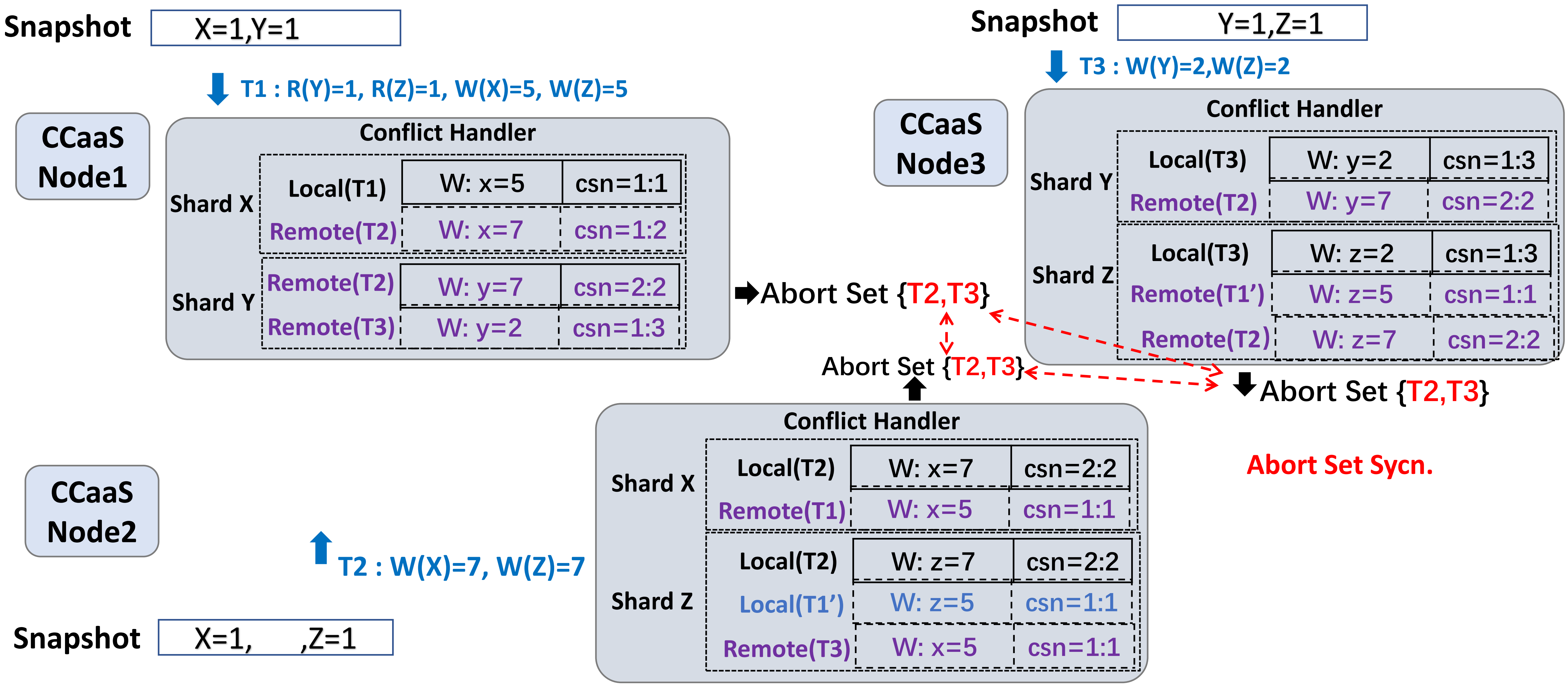


Note: csn = local time : node_id
cen: commit epoch number

SM-OCC


Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1)
W: x=5
csn=1:1
Remote(T2)
W: x=7
csn=2:2
Shard Y
Remote(T2)
W: y=7
csn=2:2
Remote(T3)
W: y=2
csn=1:3
Abort Set {T2}
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3)
W: y=2
csn=1:3
Remote(T2)
W: y=7
csn=2:2
Shard Z
Local(T3)
W: z=2
csn=1:3
Remote(T1')
W: z=5
csn=1:1
Remote(T2)
W: z=7
csn=2:2
Abort Set {T2,T3}
Abort Set {T2,T3}
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1
Conflict Handler
Shard X
Local(T2)
W: x=7
csn=2:2
Remote(T1)
W: x=5
csn=1:1
Shard Z
Local(T2)
W: z=7
csn=2:2
Local(T1')
W: z=5
csn=1:1
Remote(T3)
W: x=5
csn=1:1

SM-OCC



Snapshot
X=1,Y=1
T1 : R(Y)=1, R(Z)=1, W(X)=5, W(Z)=5
CCaaS Node1
Conflict Handler
Shard X
Local(T1)
W: x=5
csn=1:1
Remote(T2)
W: x=7
csn=1:2
Shard Y
Remote(T2)
W: y=7
csn=2:2
Remote(T3)
W: y=2
csn=1:3
Abort Set {T2,T3}
Snapshot
Y=1,Z=1
T3 : W(Y)=2,W(Z)=2
CCaaS Node3
Conflict Handler
Shard Y
Local(T3)
W: y=2
csn=1:3
Remote(T2)
W: y=7
csn=2:2
Shard Z
Local(T3)
W: z=2
csn=1:3
Remote(T1')
W: z=5
csn=1:1
Remote(T2)
W: z=7
csn=2:2
Abort Set {T2,T3}
Abort Set {T2,T3}
Abort Set Sycn.
CCaaS Node2
T2 : W(X)=7, W(Z)=7
Snapshot
X=1, ,Z=1
Conflict Handler
Shard X
Local(T2)
W: x=7
csn=2:2
Remote(T1)
W: x=5
csn=1:1
Shard Z
Local(T2)
W: z=7
csn=2:2
Local(T1')
W: z=5
csn=1:1
Remote(T3)
W: x=5
csn=1:1