SEATTLE
WA, USA
SIGMOD
PODS
2023

GeoGauss： Strongly Consistent and Light-Coordinated OLTP for Geo-Replicated SQL Database

WEIXING ZHOU, QI PENG, ZIJIE ZHANG, YANFENG ZHANG, YANG REN, SIHAO LI, GUO FU, YULONG CUI, QIANG LI, CAIYI WU, SHANGJUN HAN, SHENGYI WANG, GUOLIANG LI, AND GE YU.

NORTHEASTERN UNIVERSITY

HUAWEI TECHNOLOGY CO., LTD

TSINGHUA UNIVERSITY

Replicated Databases

Benefits:

- Data locality
- High availability
- High read throughput

Huawei’s Global Data Centers

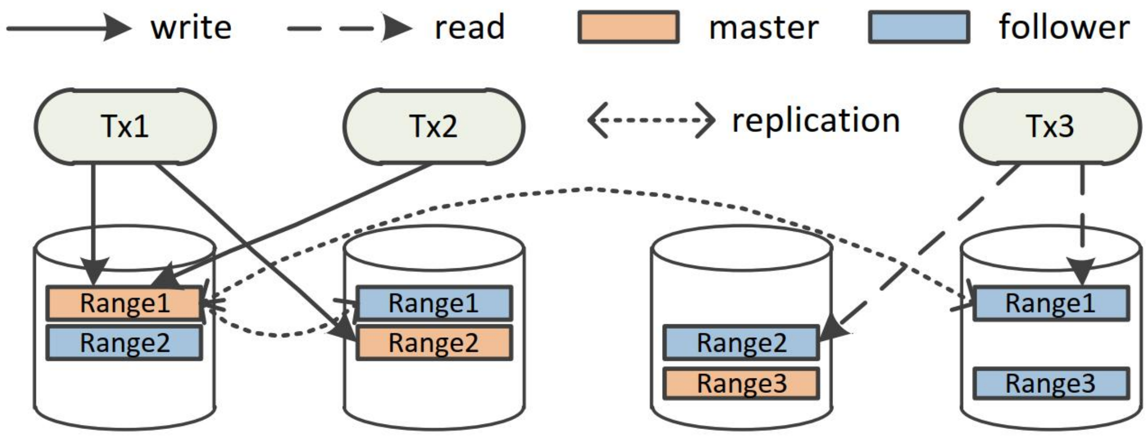
Sharded Master-Follower Replication

Method：

- Data sharding, e.g., Spanner and CockroachDB

Advantage：

- Scalability performance
- Balancing hotspots
- Reducing computing resource contention



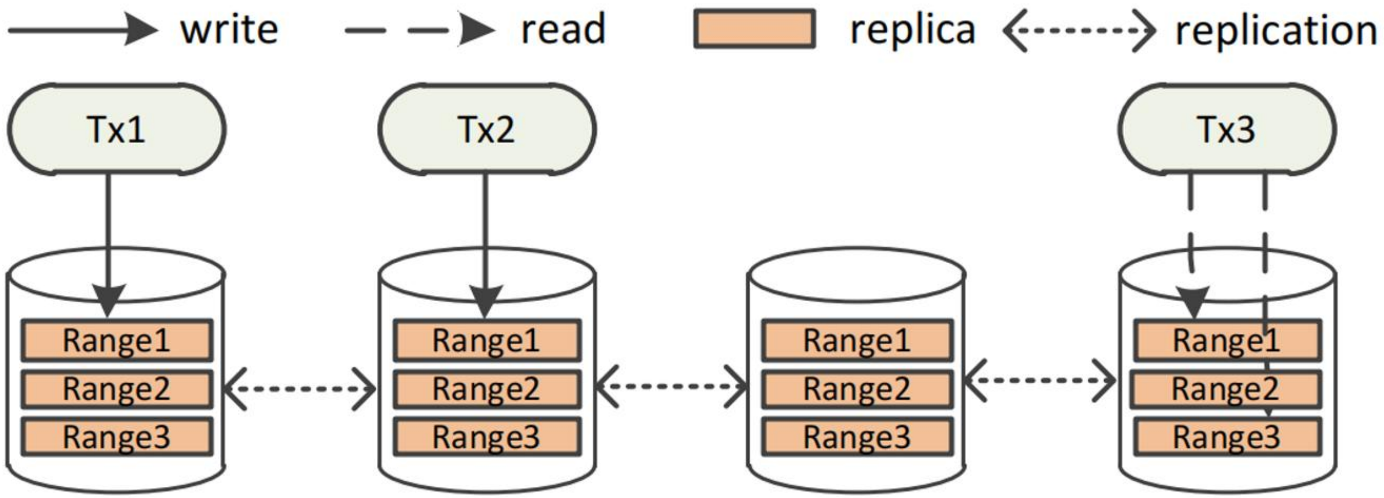
Disadvantage:

- Requests route to single master node
- Heavy coordination cost in cross-region scenarios

Multi-Master Architecture

Advantages:

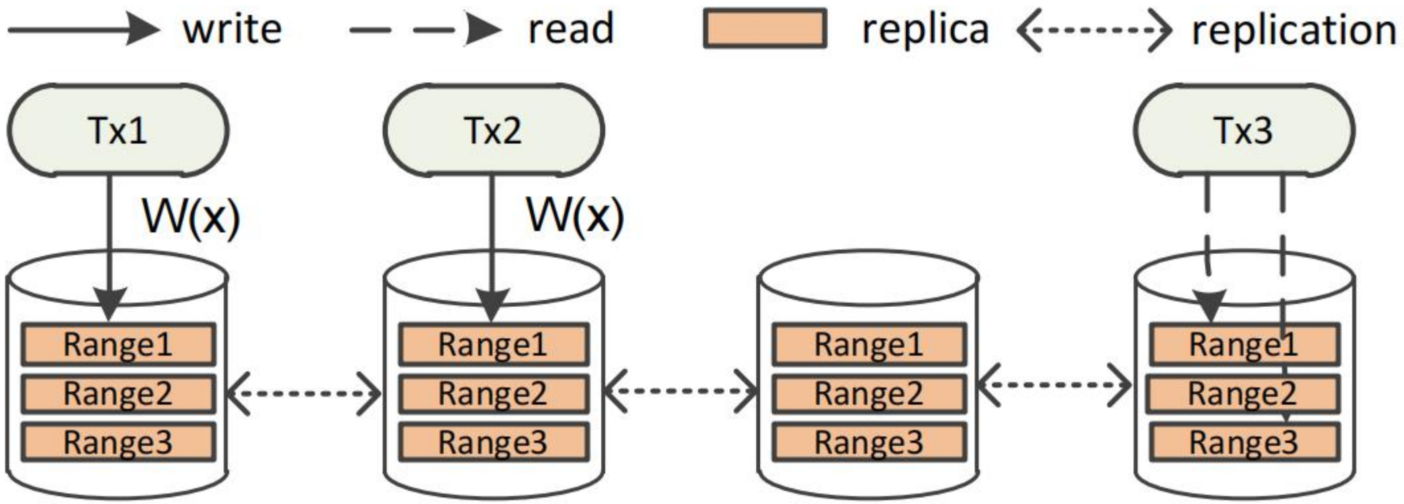
- All server nodes serve both read and write requests
- High availability
- Scalable read performance



Challenge 1 & Solution

Cross-Node Write-Write Conflicts

- Concurrent updates to multiple replicas of the same data
- Expensive coordination：
  - Heavy communication cost in geo-distributed databases



Challenge 1 & Solution

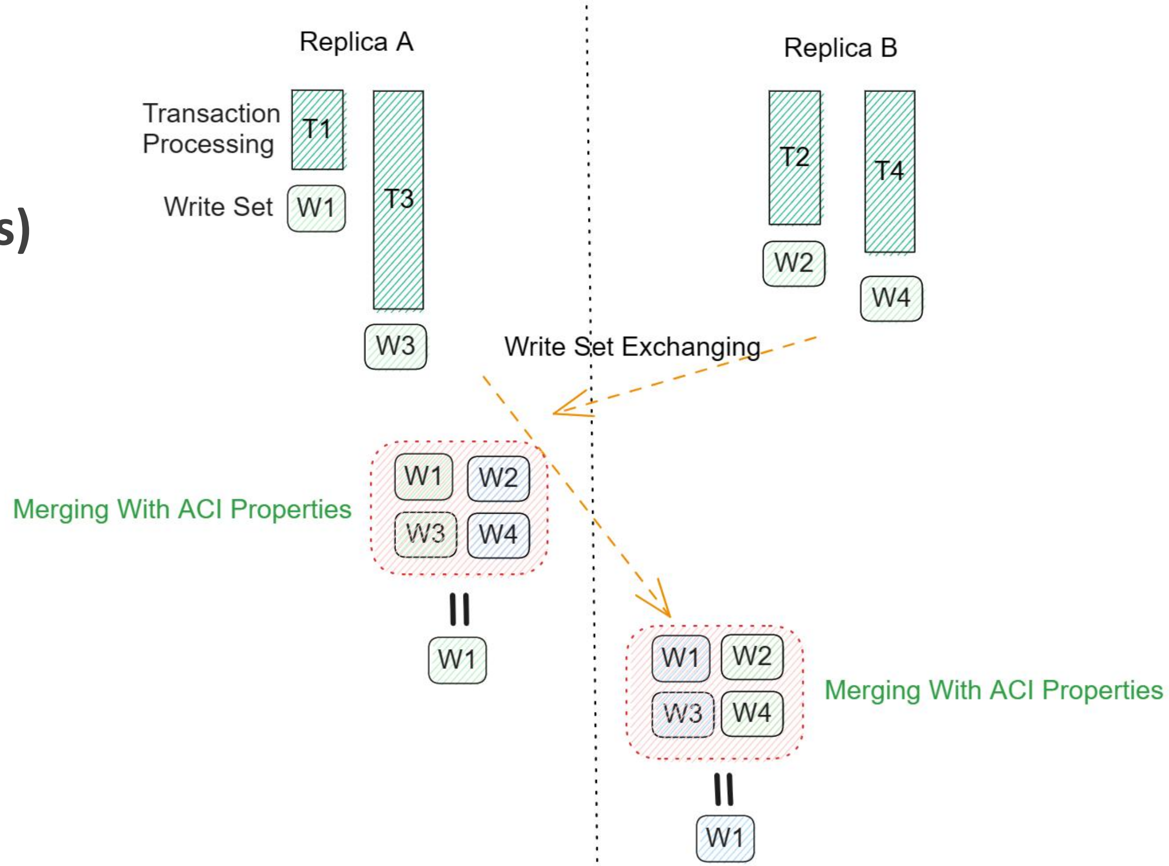
Cross-Node Write-Write Conflicts

Method： ***Conflict-Free Replicated Datatypes*** **(CRDTs)**

- Multi-Master architecture
- Exchange write sets
- Merge function with ACI properties :
  - (*associative, commutative, idempotent*)

Effect：

- Coordination-Free
- Eventual consistency



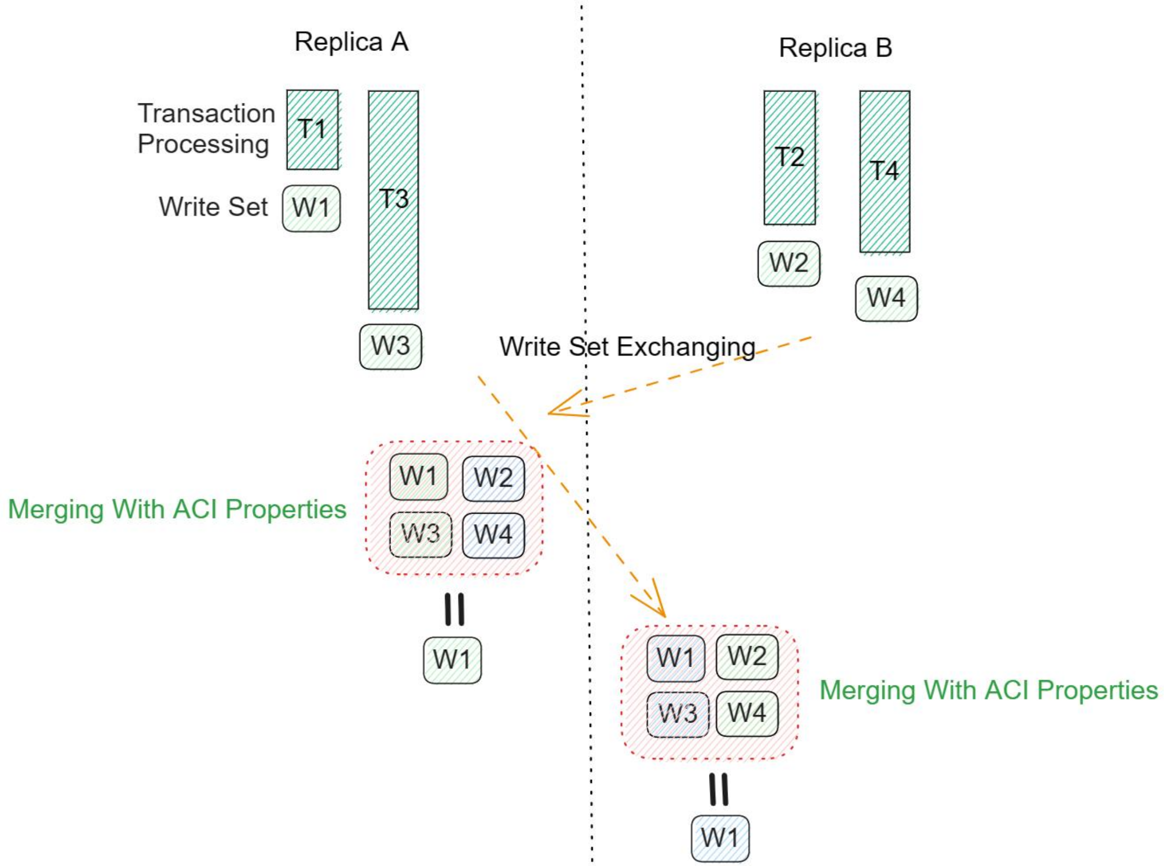
Coordination-Free Conflicts Processing Under Multi-Master architecture by Using ACI Properties

Challenge 2 & Solution

Eventual Consistency:

- Data inconsistent
- Not suitable for OLTP

Require strong consistency



Coordination-Free Conflicts Processing Under Multi-Master architecture by Using ACI Properties

Challenge 2 & Solution

Strong Consistency:

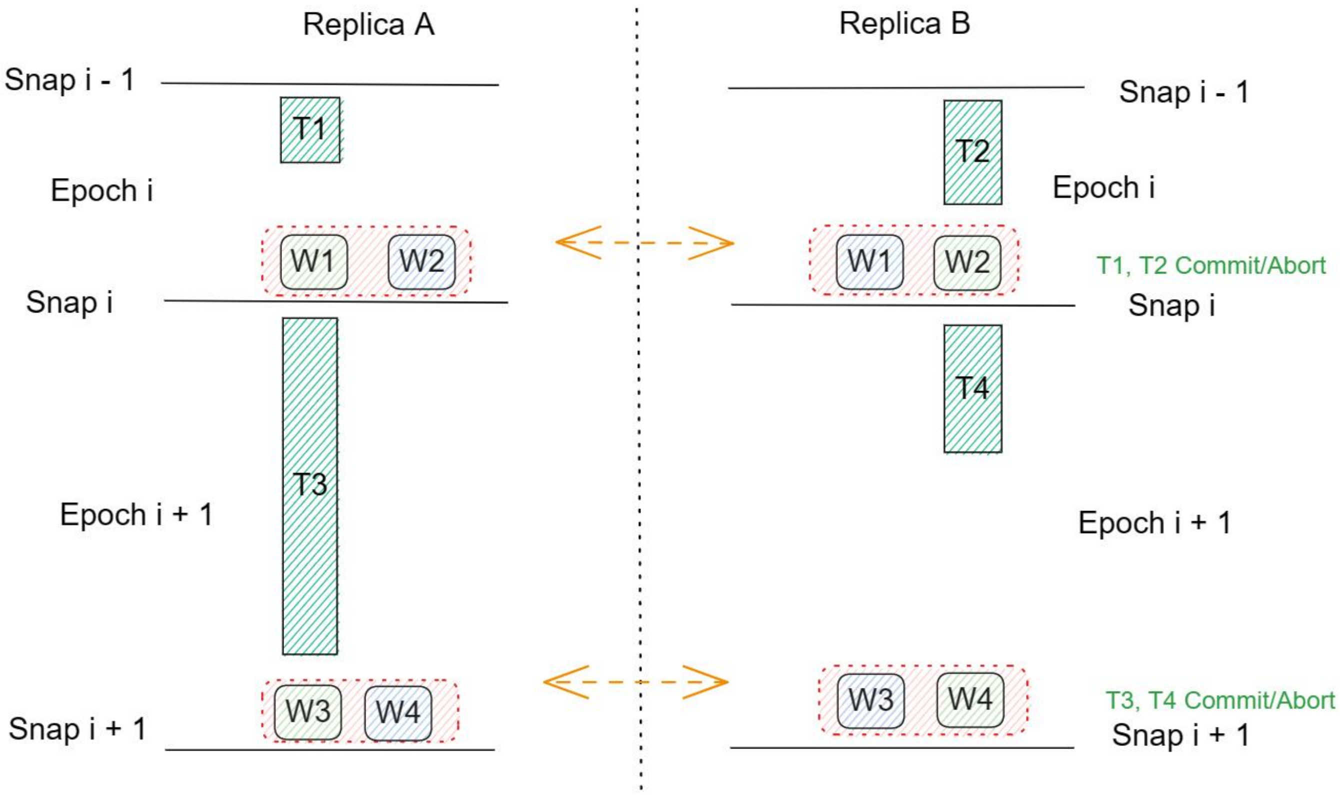
- Eventual consistency is not suitable in OLTP

Method: Epoch-Based processing

- Epoch-Based synchronization
- Execute transactions epoch by epoch

Effect:

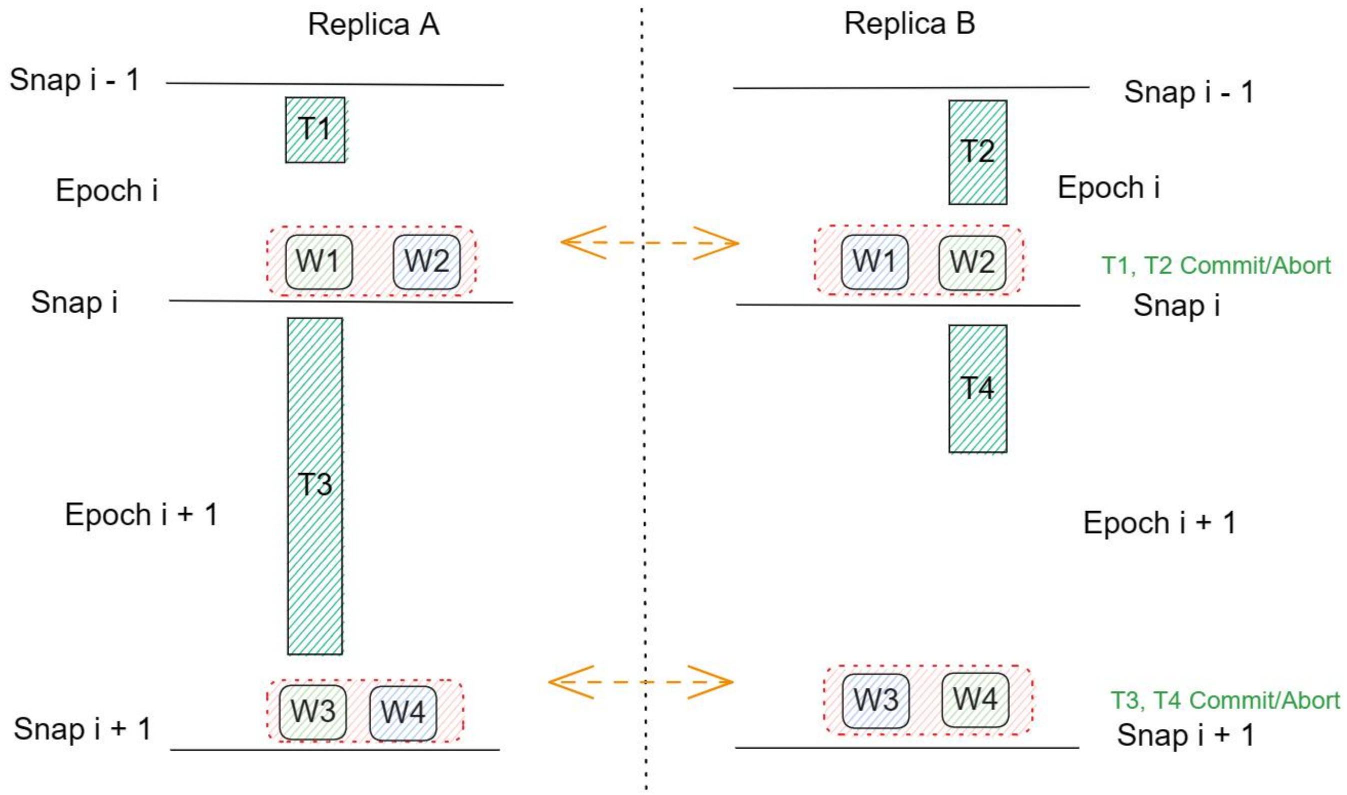
- Sequential consistency per-epoch basis



Challenge 3 & Solution

Performance:

- Epoch-Based execution
- Imbalanced workload e.g. long transaction



Challenge 3 & Solution

Performance:

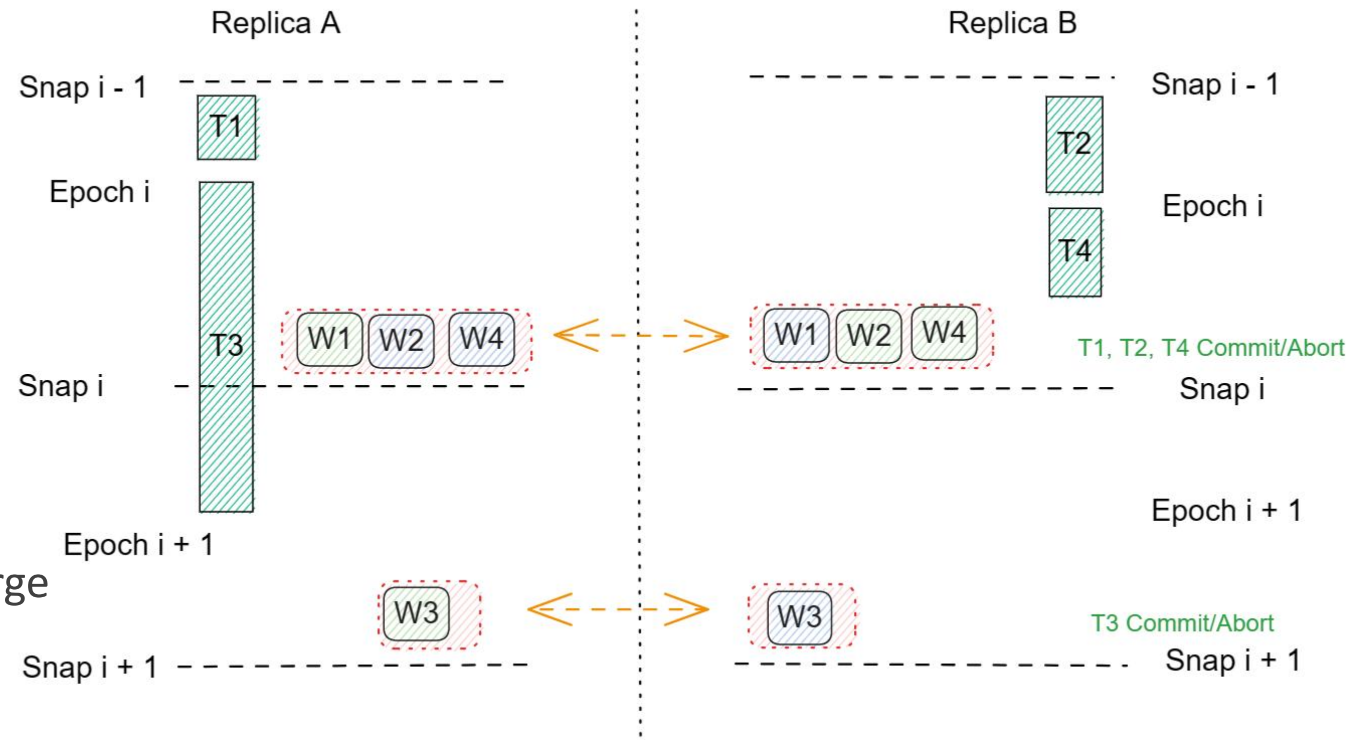
- Epoch-Based execution
- Imbalanced workload e.g. long transaction

Method:

- Multi-Master Epoch-Based OCC

Effect:

- High throughput, low latency
- long transaction does not affect epoch merge



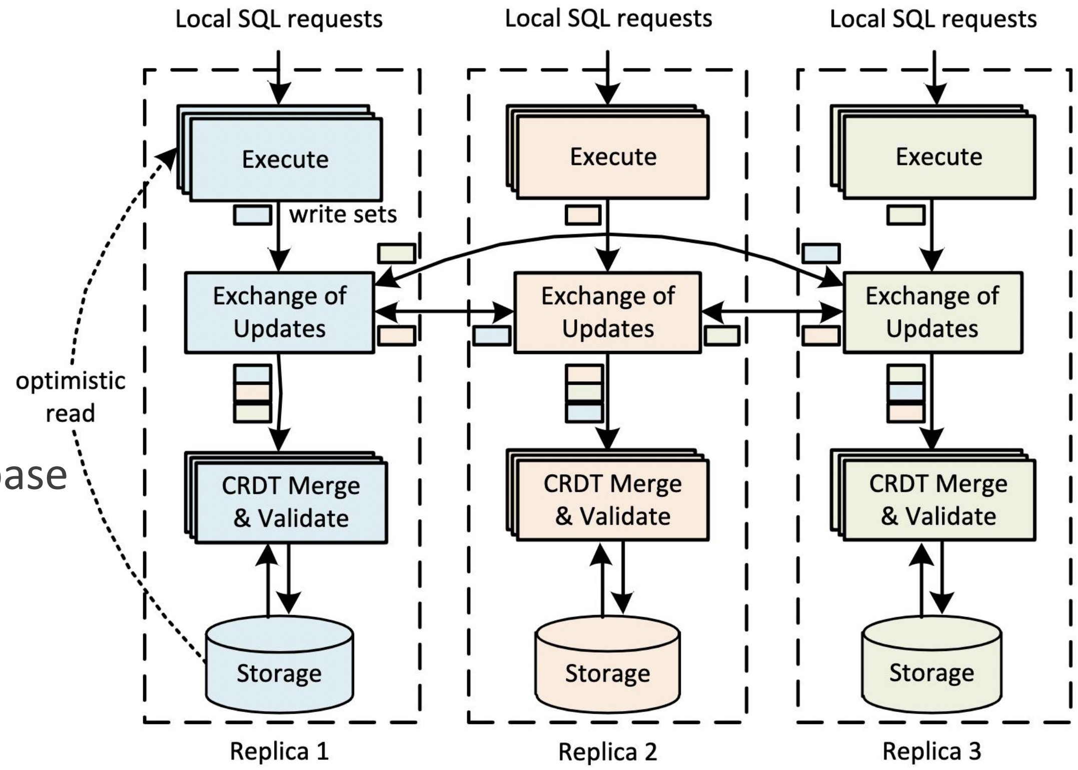
GeoGauss A Cross-Region Multi-master Distributed Database

Epoch-based multi-master OCC:

- Coordination-Free transaction processing
- Epoch-Based merge
- Optimistic execution

GeoGauss:

- Cross-Region Multi-Master Distributed Relational Database
- Based on openGauss MOT[vldb2020]



Replica a

x=1, cen=0, csn=0	y=0, cen=0, csn=0

cen: commit epoch number

csn: commit sequence number

Replica b

x=1, cen=0, csn=0	y=0, cen=0, csn=0

Replica a	x=1, cen=0, csn=0	y=0, cen=0, csn=0

T1 {$R_x(1)$ $W_x(2)$}

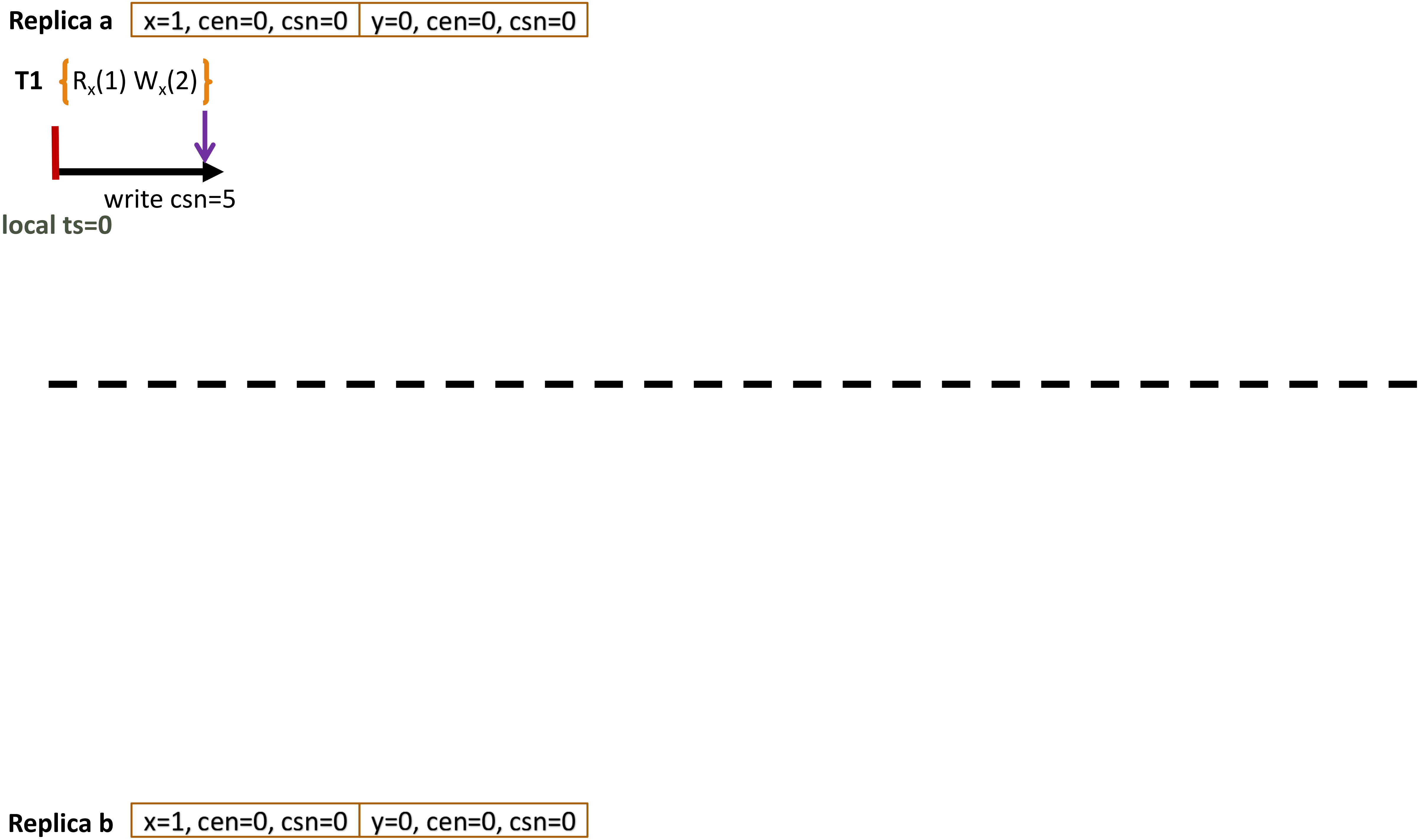
T1 execution,

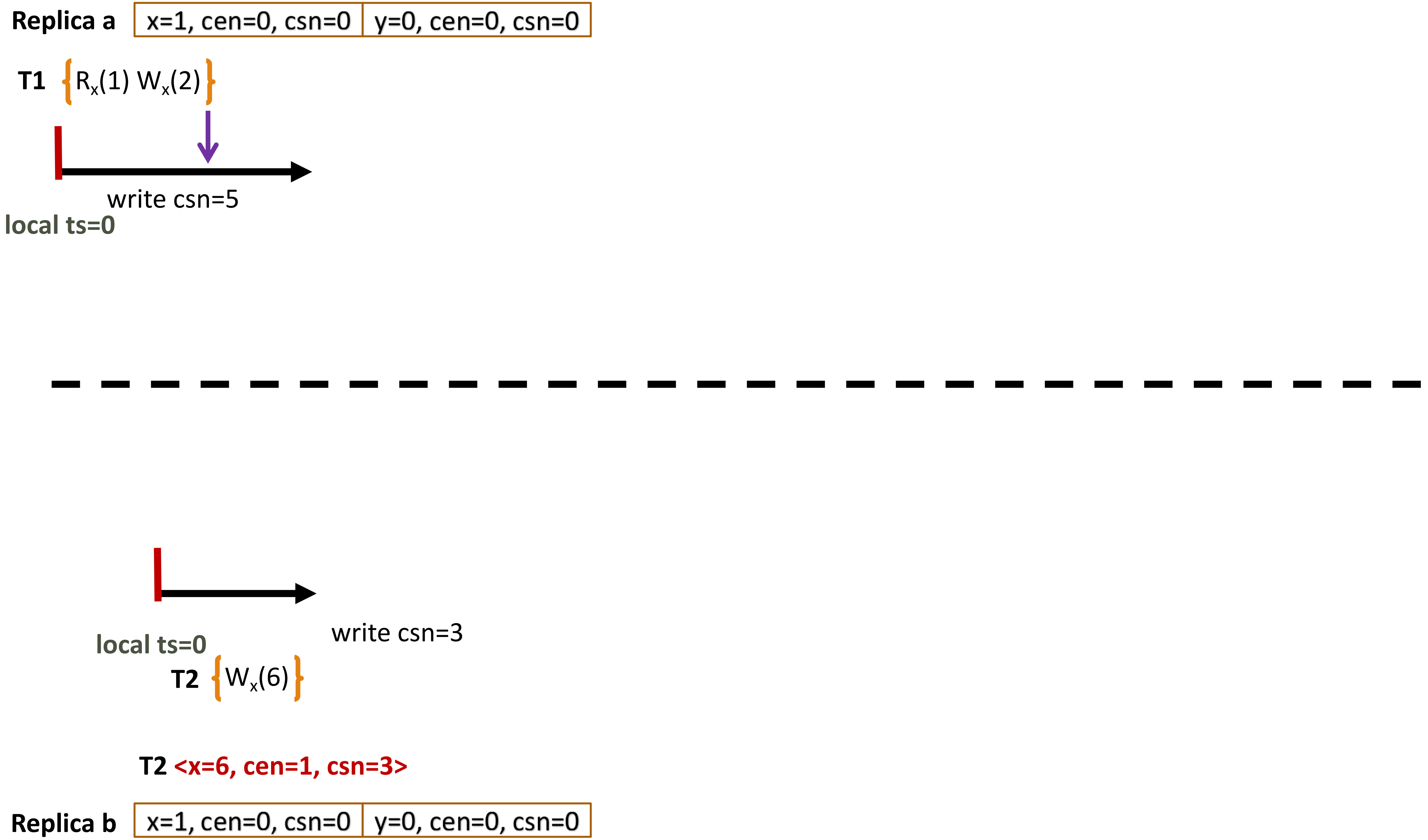
T1’s updates <x=2, cen=1, csn=5>

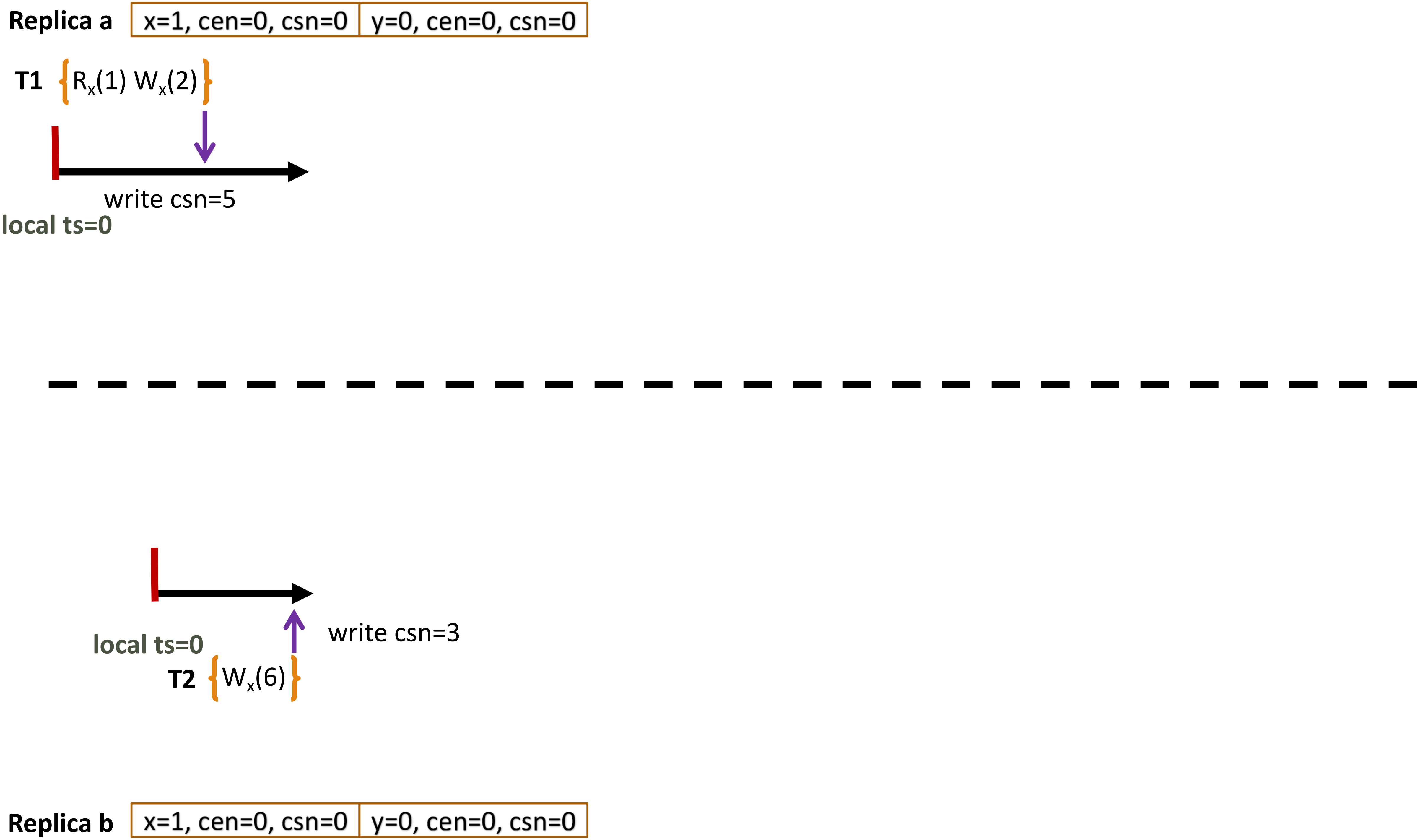
write csn=5

local ts=0

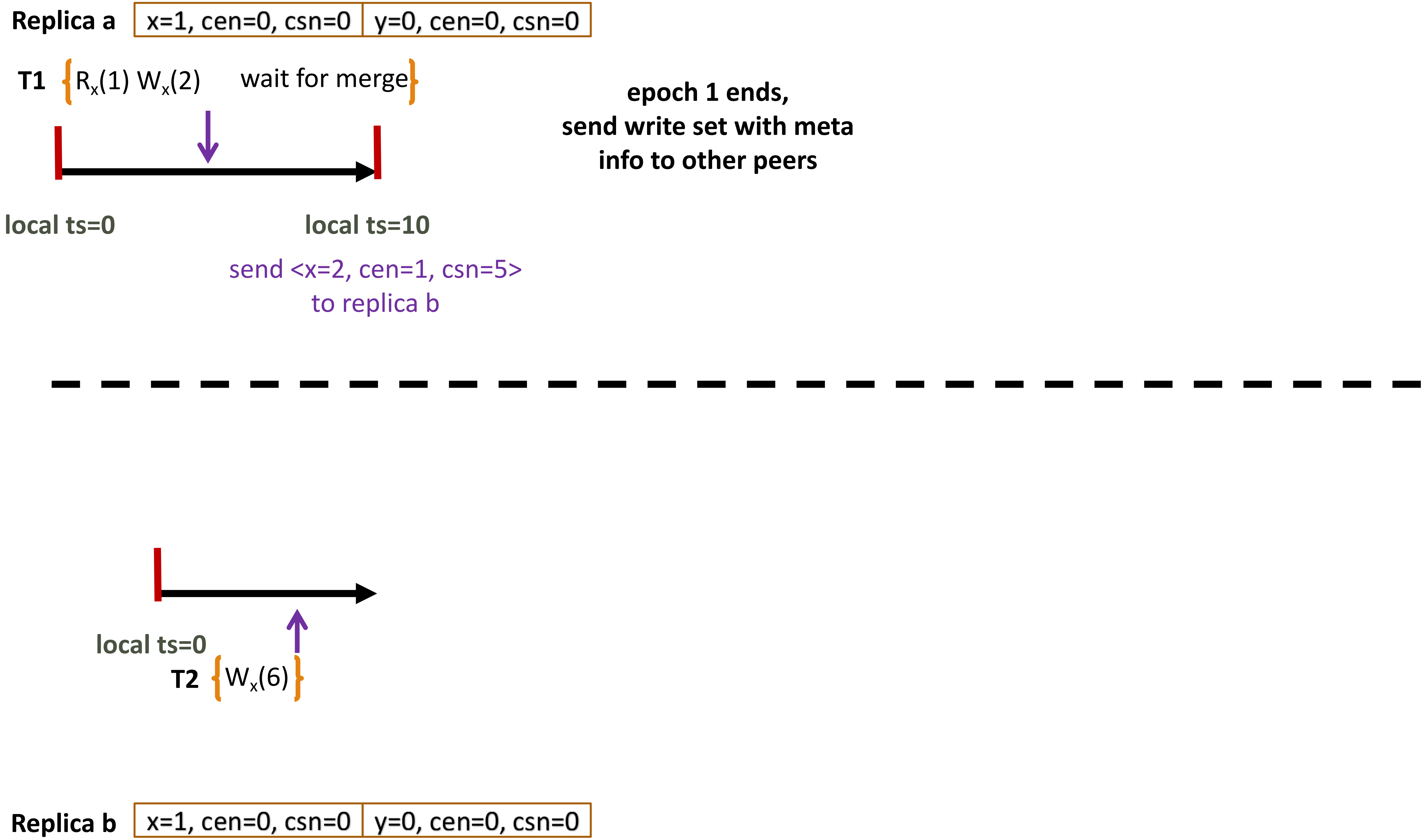
Replica b	x=1, cen=0, csn=0	y=0, cen=0, csn=0

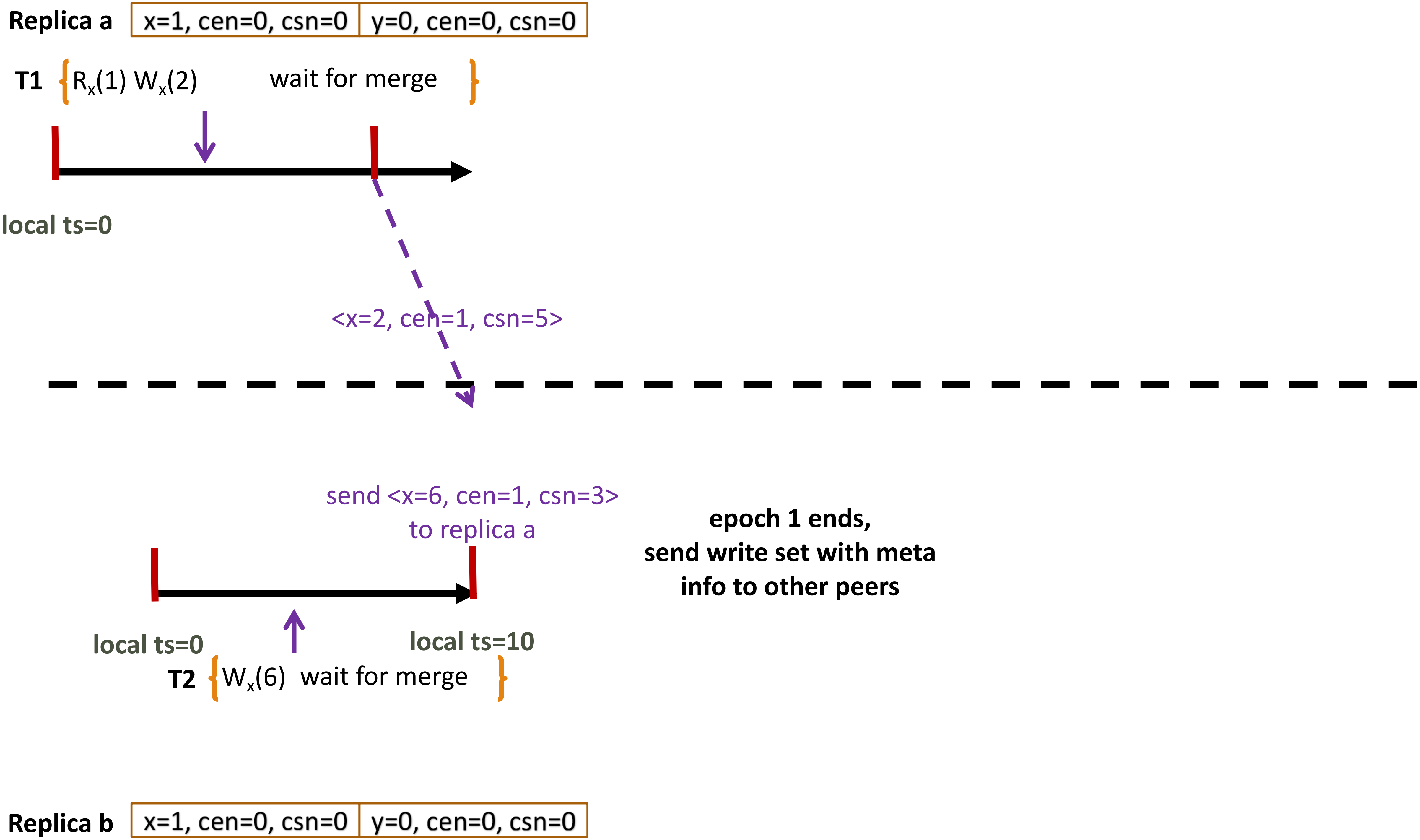

Replica a
x=1, cen=0, csn=0
y=0, cen=0, csn=0
T1
$R_x(1)$ $W_x(2)$
write csn=5
local ts=0
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0

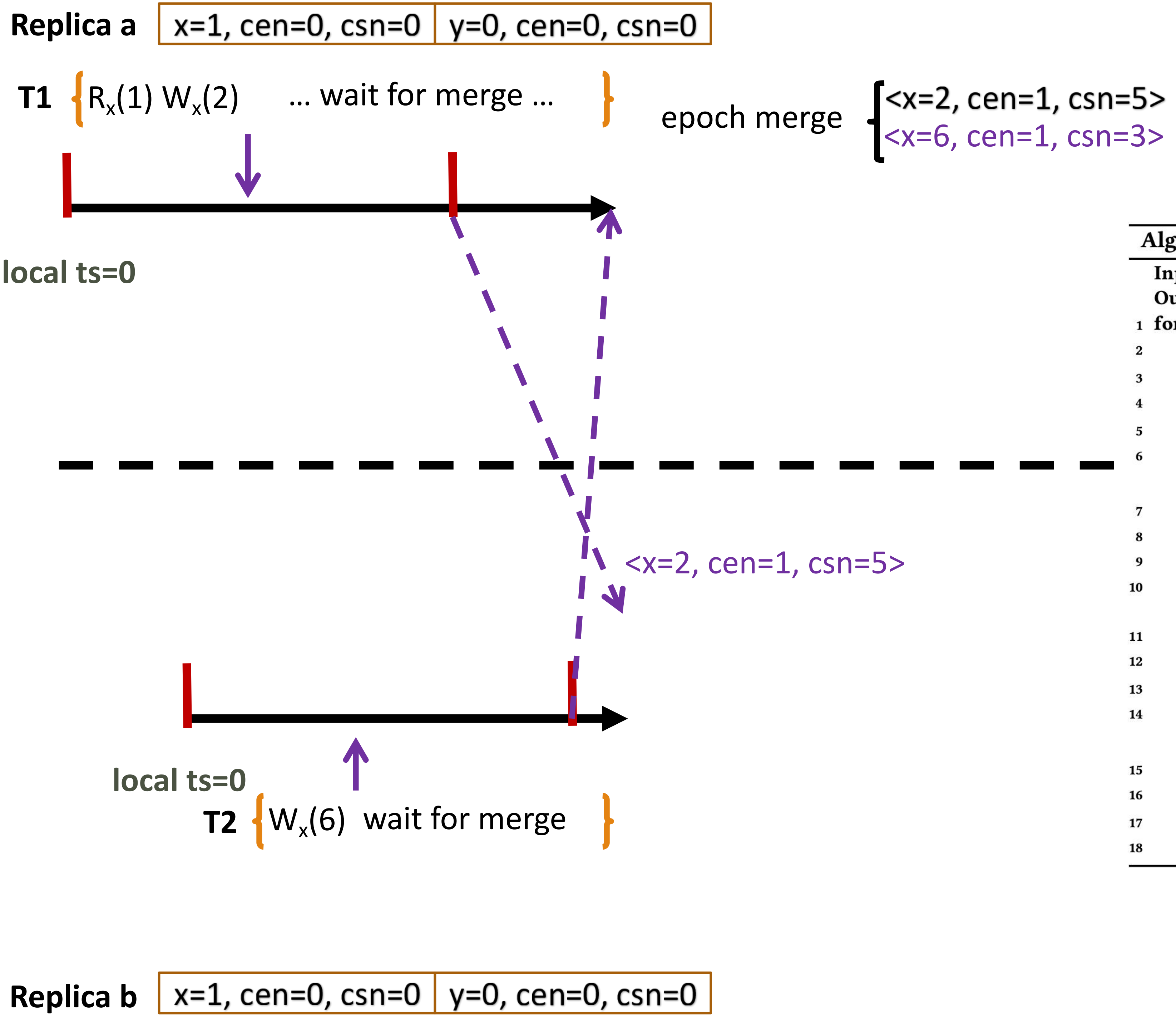

Replica a
x=1, cen=0, csn=0
y=0, cen=0, csn=0
T1 {$R_x(1)$ $W_x(2)$}
write csn=5
local ts=0
local ts=0
write csn=3
T2 {$W_x(6)$}
T2 <x=6, cen=1, csn=3>
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0



Replica a
x=1, cen=0, csn=0
y=0, cen=0, csn=0
T1
$R_x(1)$ $W_x(2)$
write csn=5
local ts=0
local ts=0
write csn=3
T2
$W_x(6)$
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0


Replica a
x=1, cen=0, csn=0
y=0, cen=0, csn=0
T1
R_x(1) W_x(2)
wait for merge
epoch 1 ends,
send write set with meta
info to other peers
local ts=0
local ts=10
send <x=2, cen=1, csn=5>
to replica b
local ts=0
T2
W_x(6)
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0

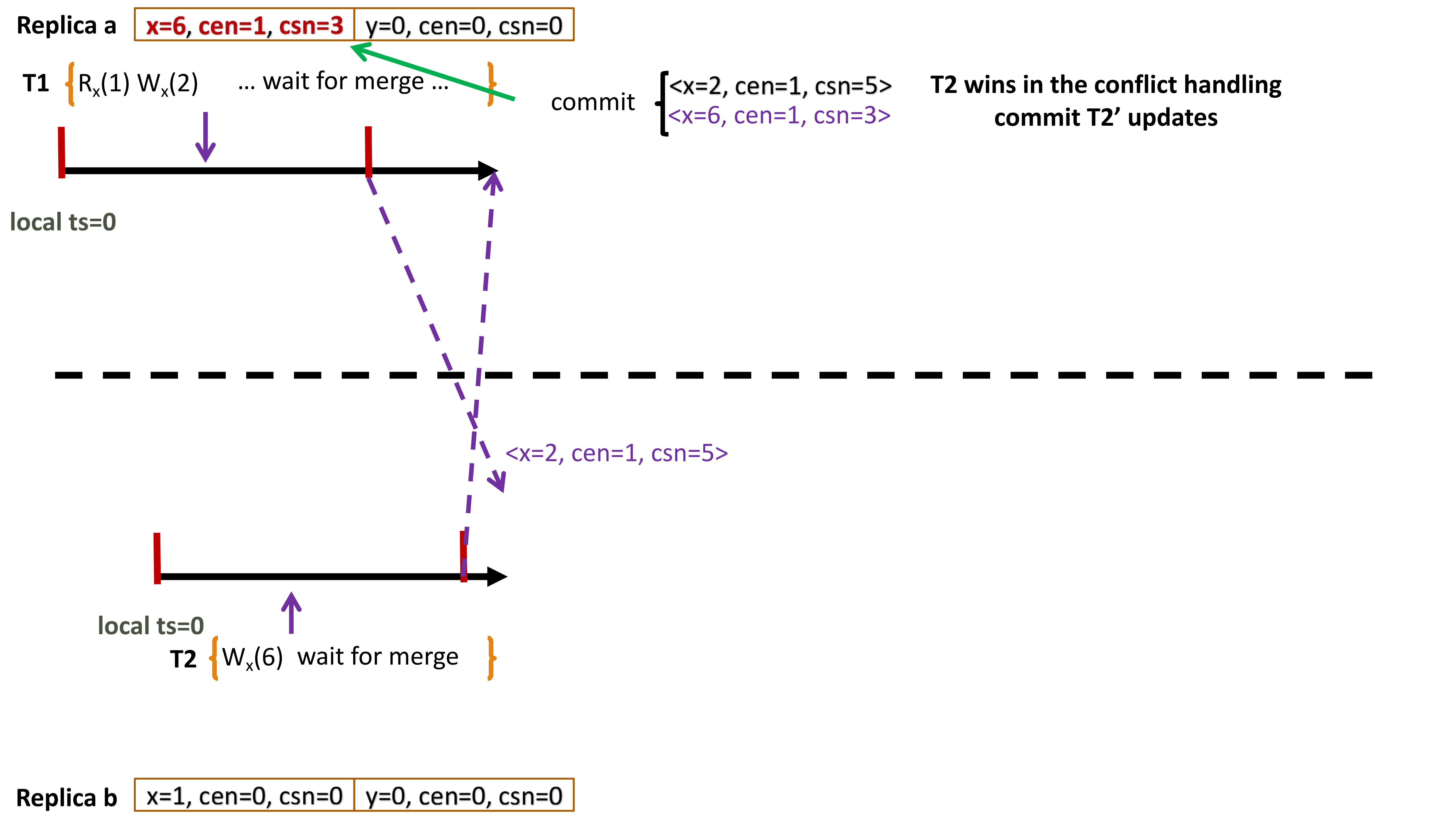

Replica a
x=1, cen=0, csn=0
y=0, cen=0, csn=0
T1
R_x(1) W_x(2)
wait for merge
local ts=0
<x=2, cen=1, csn=5>
send <x=6, cen=1, csn=3>
to replica a
epoch 1 ends,
send write set with meta
info to other peers
local ts=0
local ts=10
T2
W_x(6) wait for merge
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0



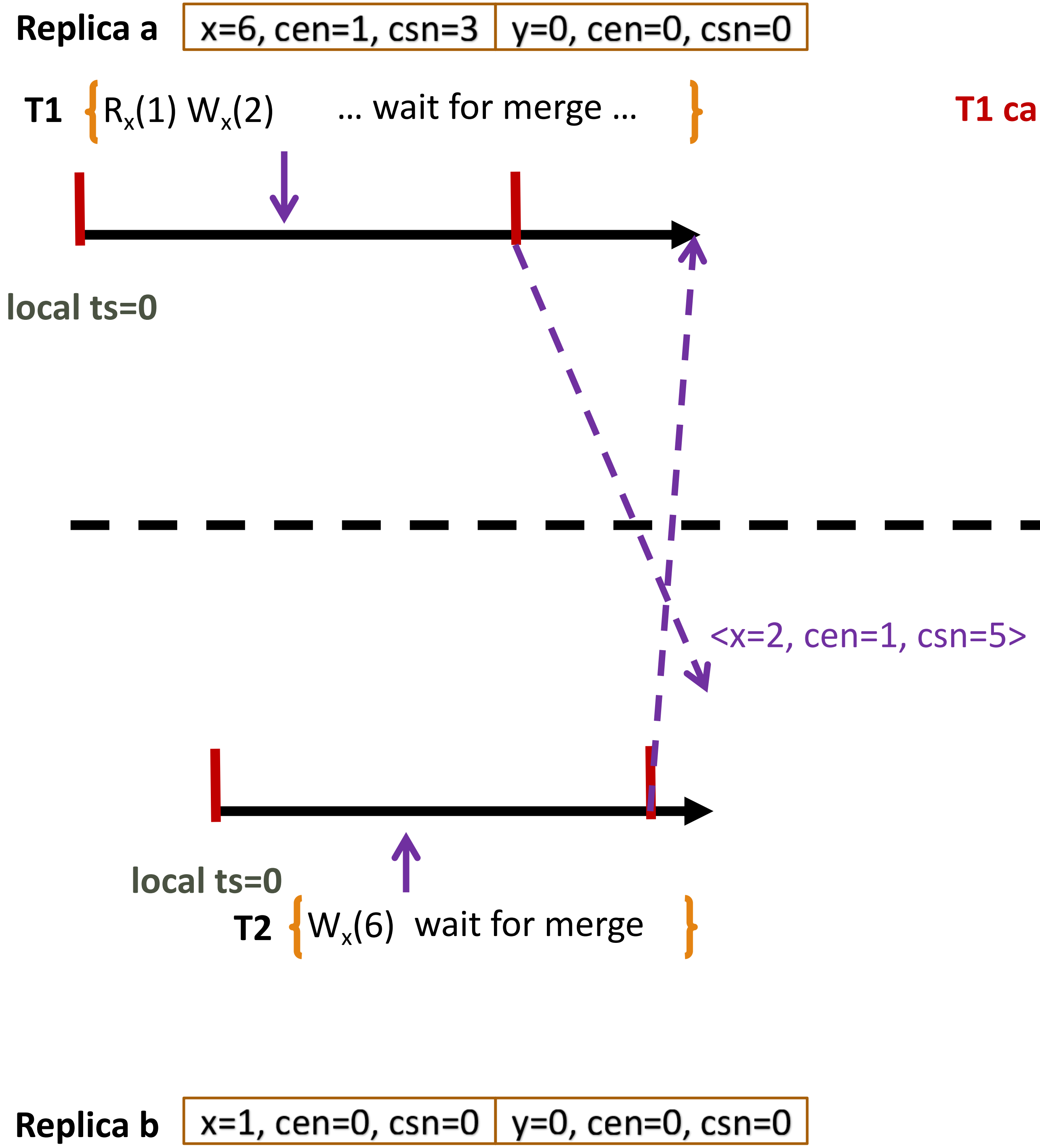
remote updates arrive, merge with local updates

Algorithm 2: DeltaCRDTMerge

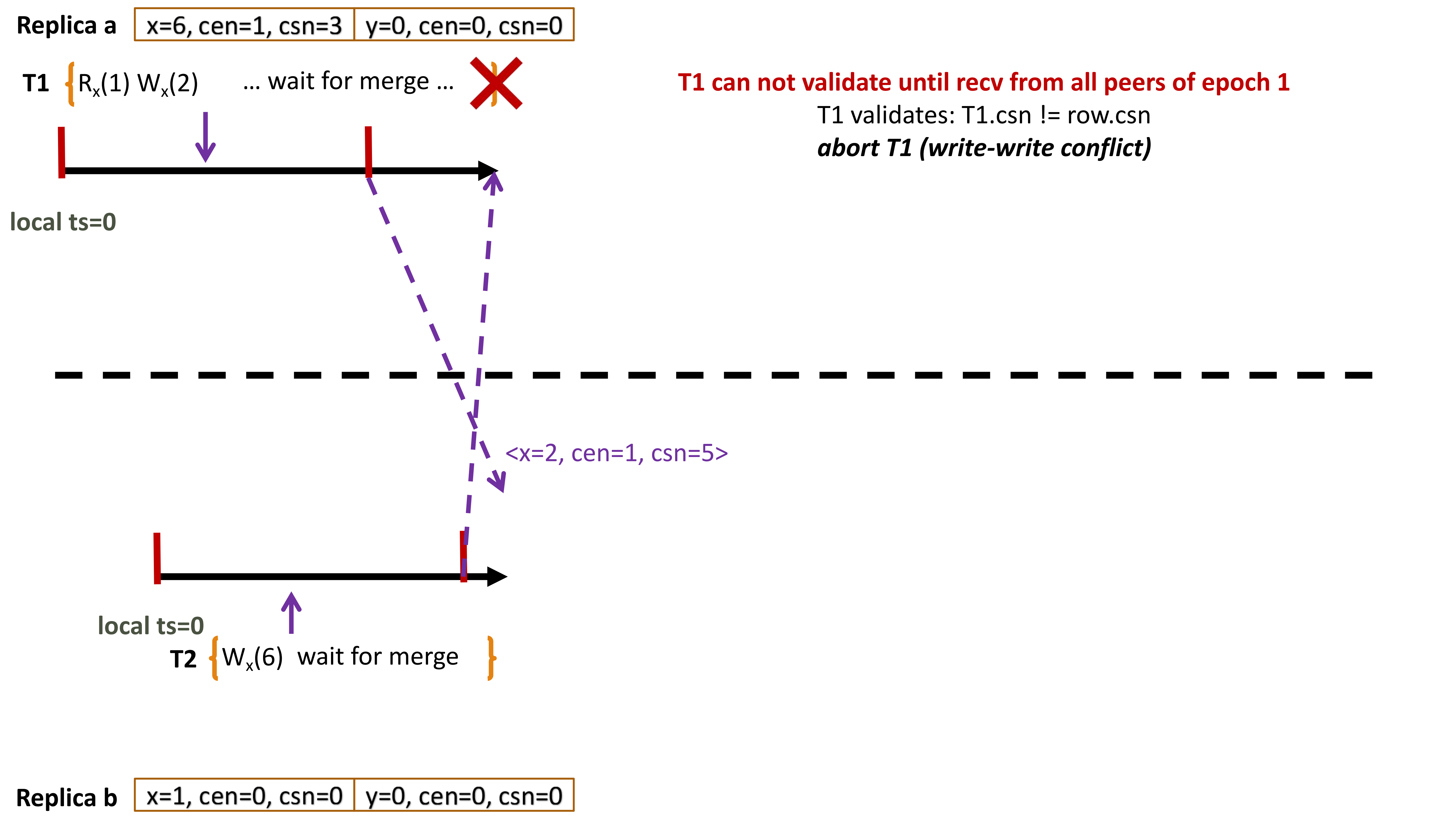
```
Input: a transaction's T.{sen, csn, cen, WS}, current row headers
Output: updated row headers
for each record in T.WS do
    row = FindRow(record.key);
    if row == null then
        Abort T; //row is deleted by other threads
    if row.cen < T.cen then
        //row is not pre-written in current epoch
         row.{sen, csn, cen} = T.{sen, csn, cen};
    else if row.cen == T.cen then
        if row.sen == T.sen then
            if row.csn > T.csn then
                //first write wins
                 row.{sen, csn, cen} = T.{sen, csn, cen};
            else
                Abort T;
        else if row.sen < T.sen then
            //shorter transaction wins
             row.{sen, csn, cen} = T.{sen, csn, cen};
        else
            Abort T;
    else
        //row.cen > T.cen will never happen
```

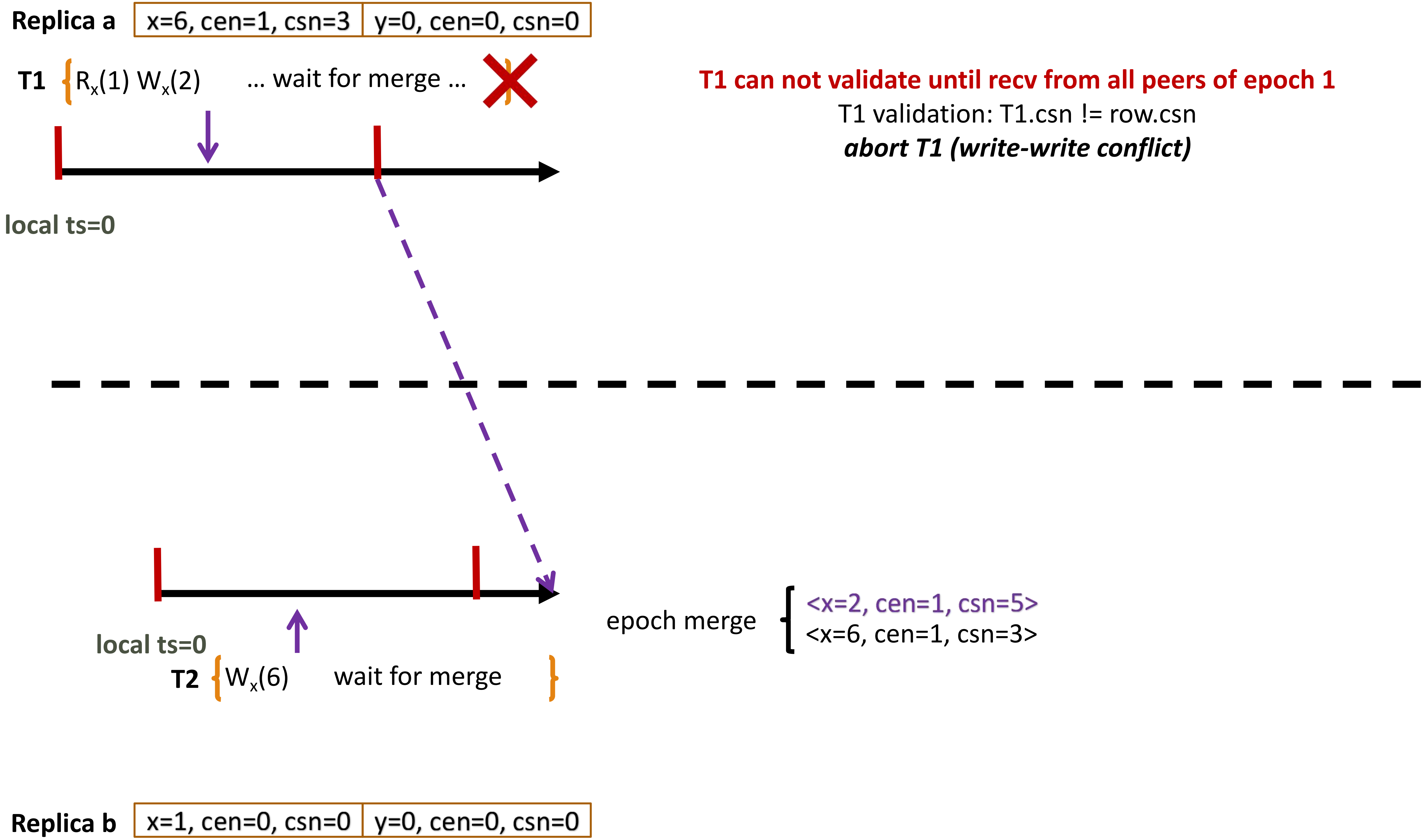


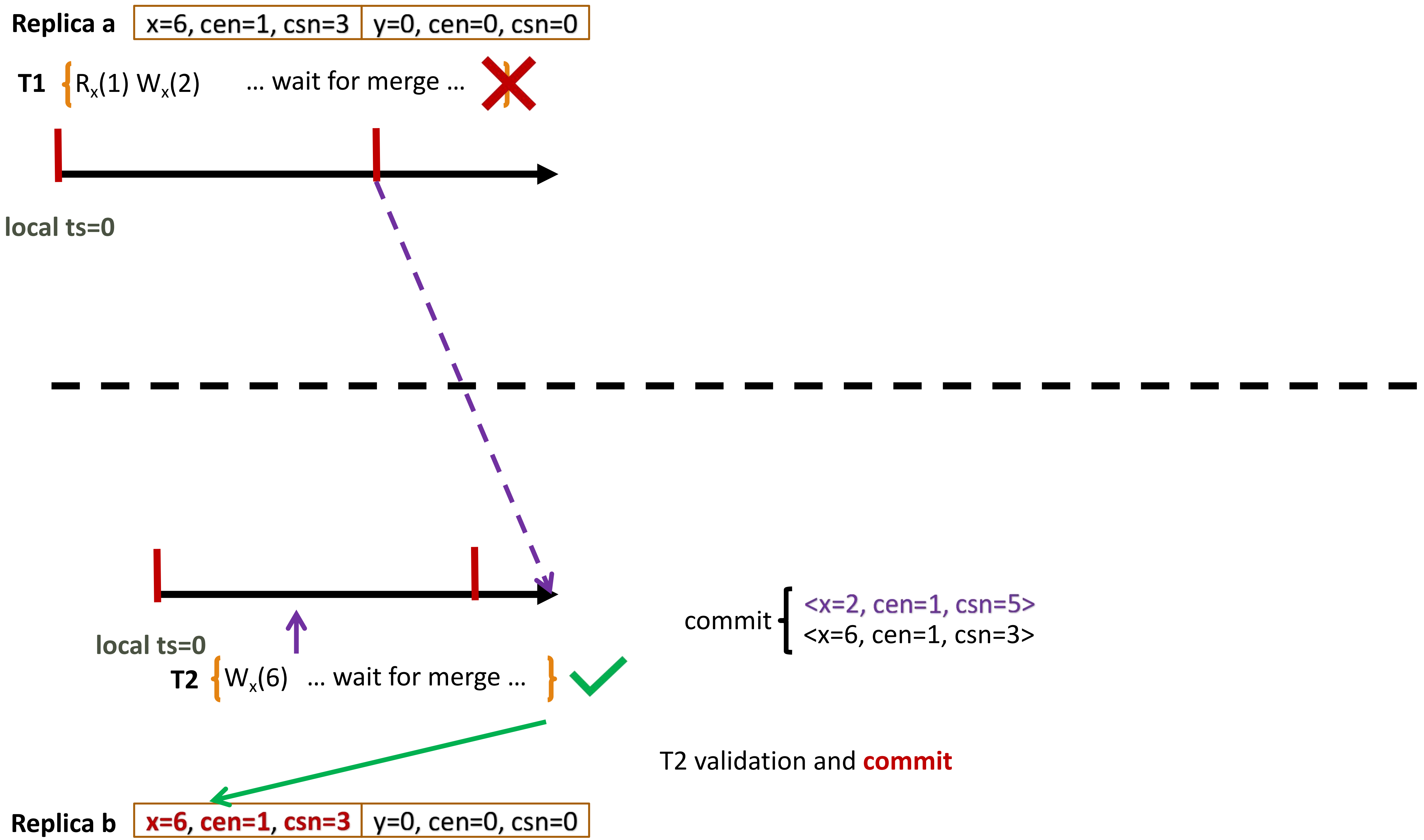
Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T1
$R_x(1)$ $W_x(2)$
… wait for merge …
commit
<x=2, cen=1, csn=5>
<x=6, cen=1, csn=3>
T2 wins in the conflict handling
commit T2’ updates
local ts=0
<x=2, cen=1, csn=5>
local ts=0
T2
$W_x(6)$ wait for merge
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0



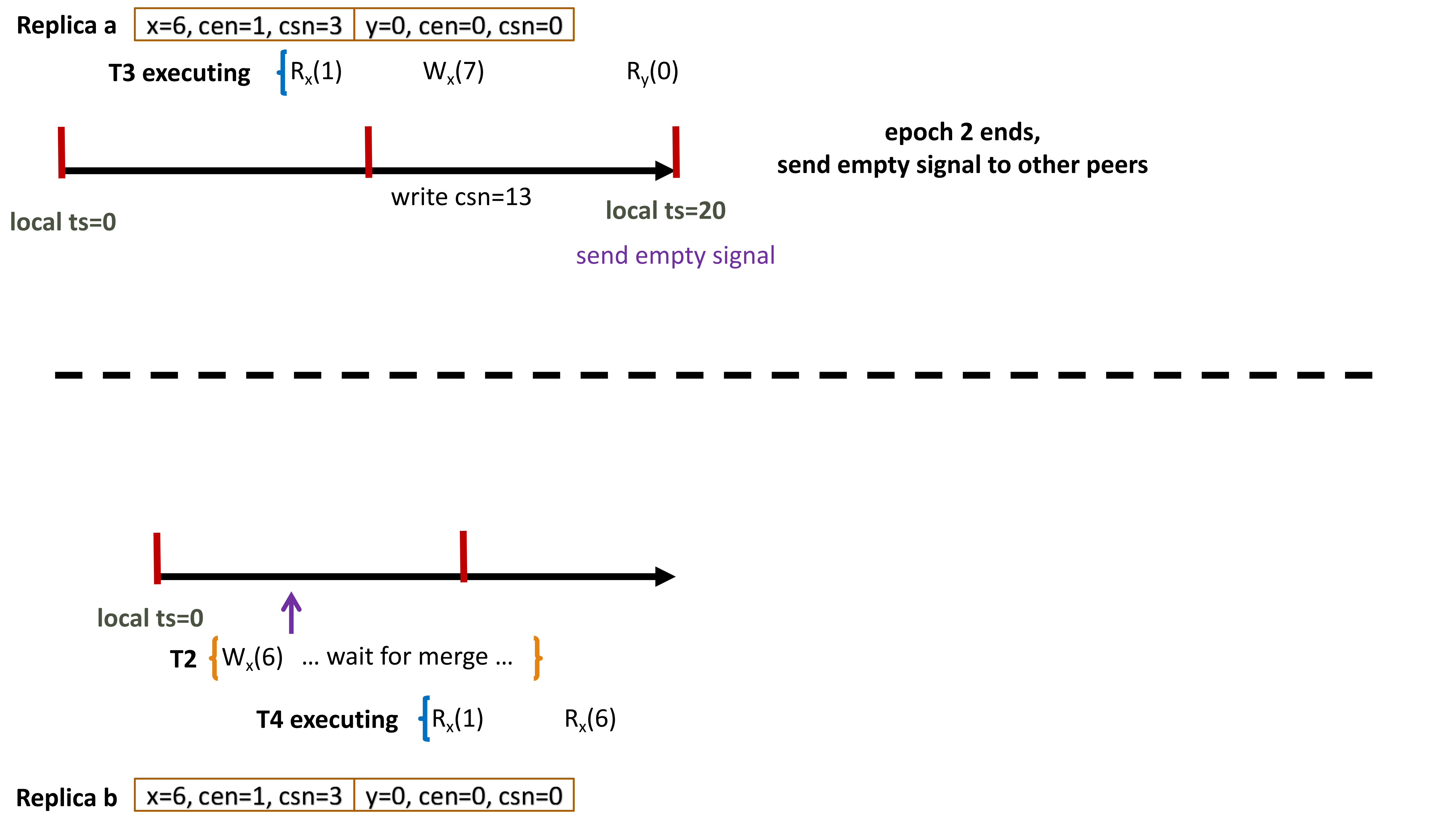
T1 can not validate until recv from all peers of epoch 1

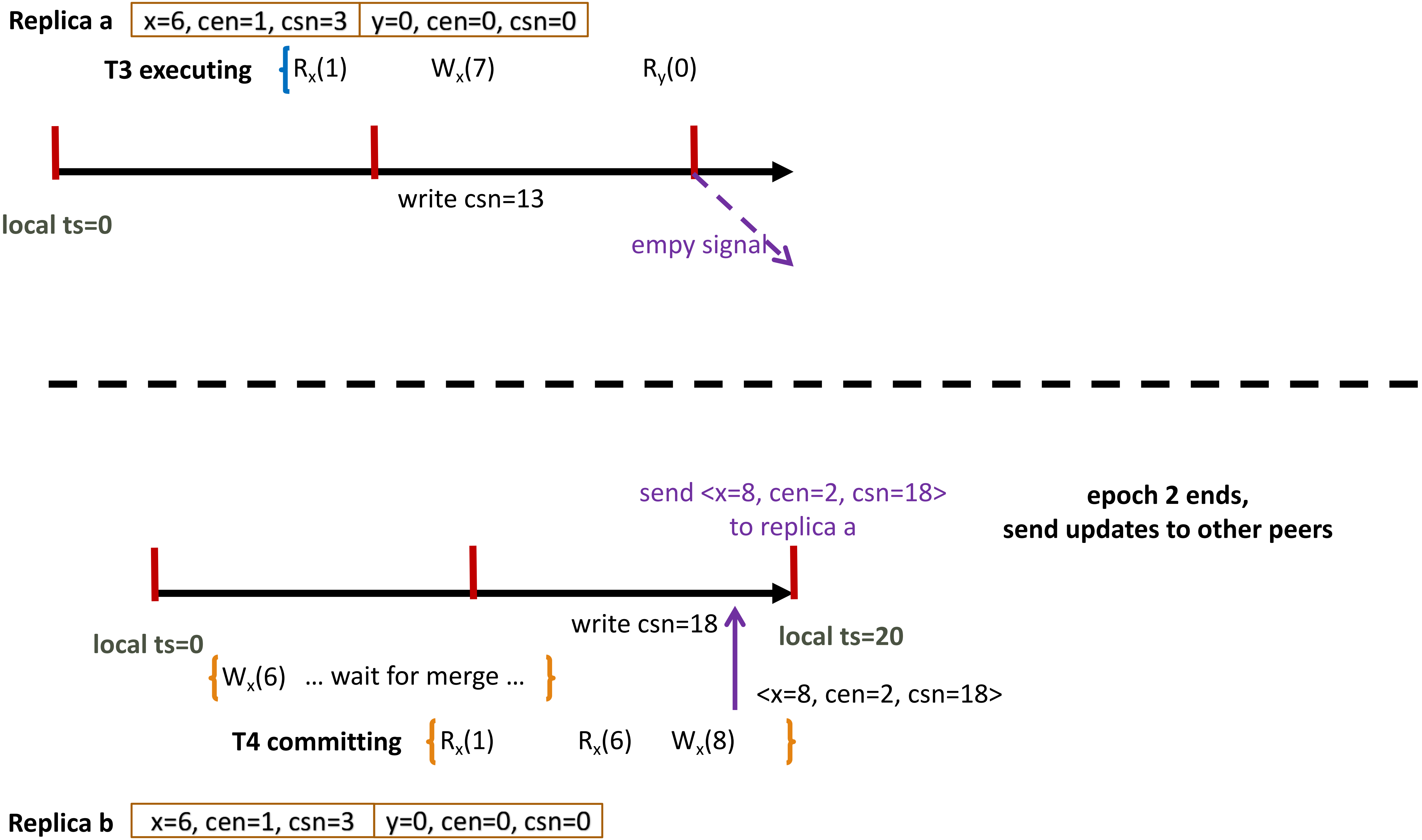

Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T1
Rx(1) Wx(2)
… wait for merge …
T1 can not validate until recv from all peers of epoch 1
T1 validates: T1.csn != row.csn
abort T1 (write-write conflict)
local ts=0
<x=2, cen=1, csn=5>
local ts=0
T2
Wx(6) wait for merge
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0


Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T1
$R_x(1)$ $W_x(2)$
… wait for merge …
T1 can not validate until recv from all peers of epoch 1
T1 validation: T1.csn != row.csn
abort T1 (write-write conflict)
local ts=0
local ts=0
T2
$W_x(6)$
wait for merge
epoch merge
<x=2, cen=1, csn=5>
<x=6, cen=1, csn=3>
Replica b
x=1, cen=0, csn=0
y=0, cen=0, csn=0

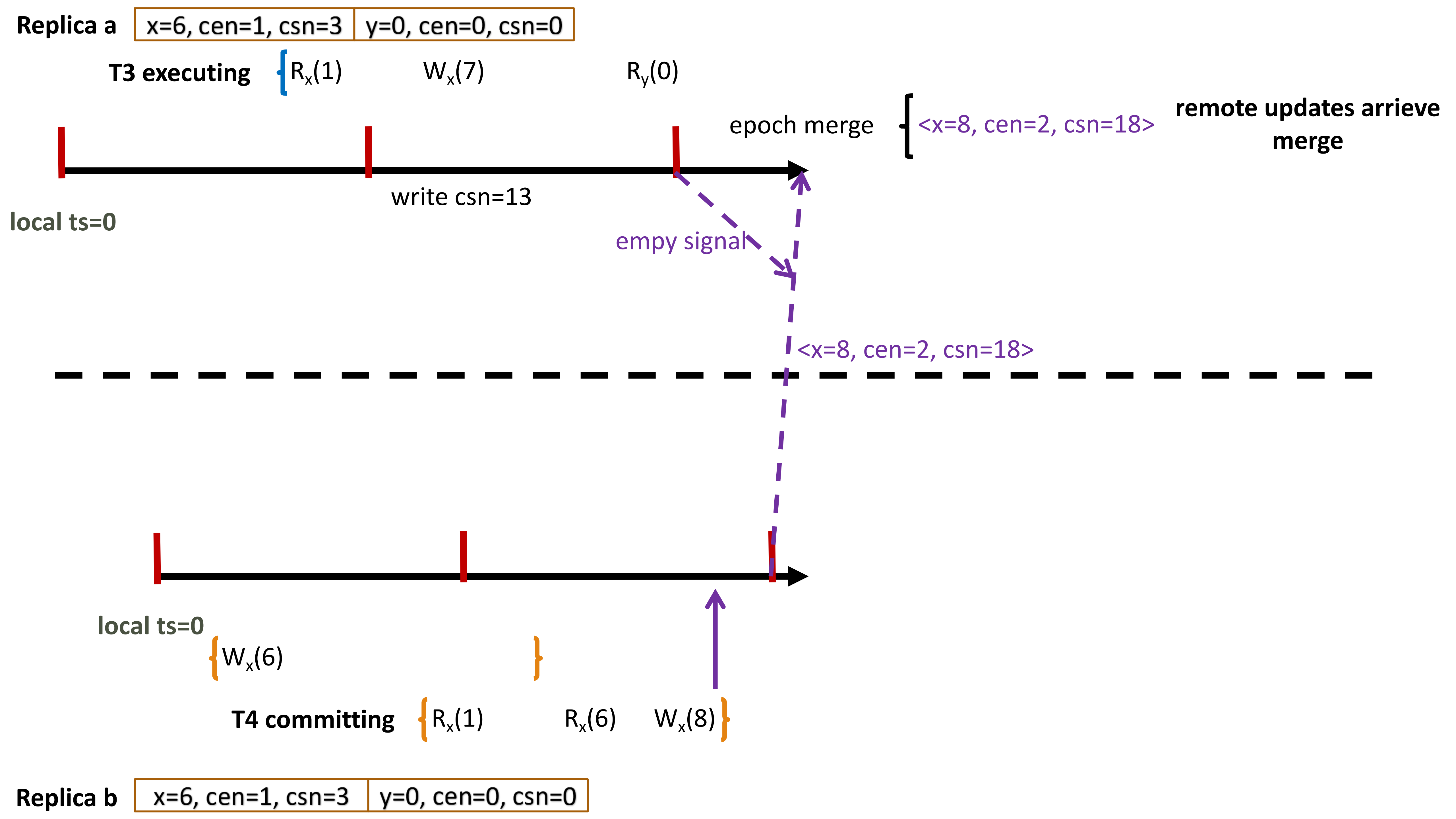


Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T1
Rx(1) Wx(2)
… wait for merge …
local ts=0
local ts=0
T2
Wx(6)
… wait for merge …
commit
<x=2, cen=1, csn=5>
<x=6, cen=1, csn=3>
T2 validation and commit
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

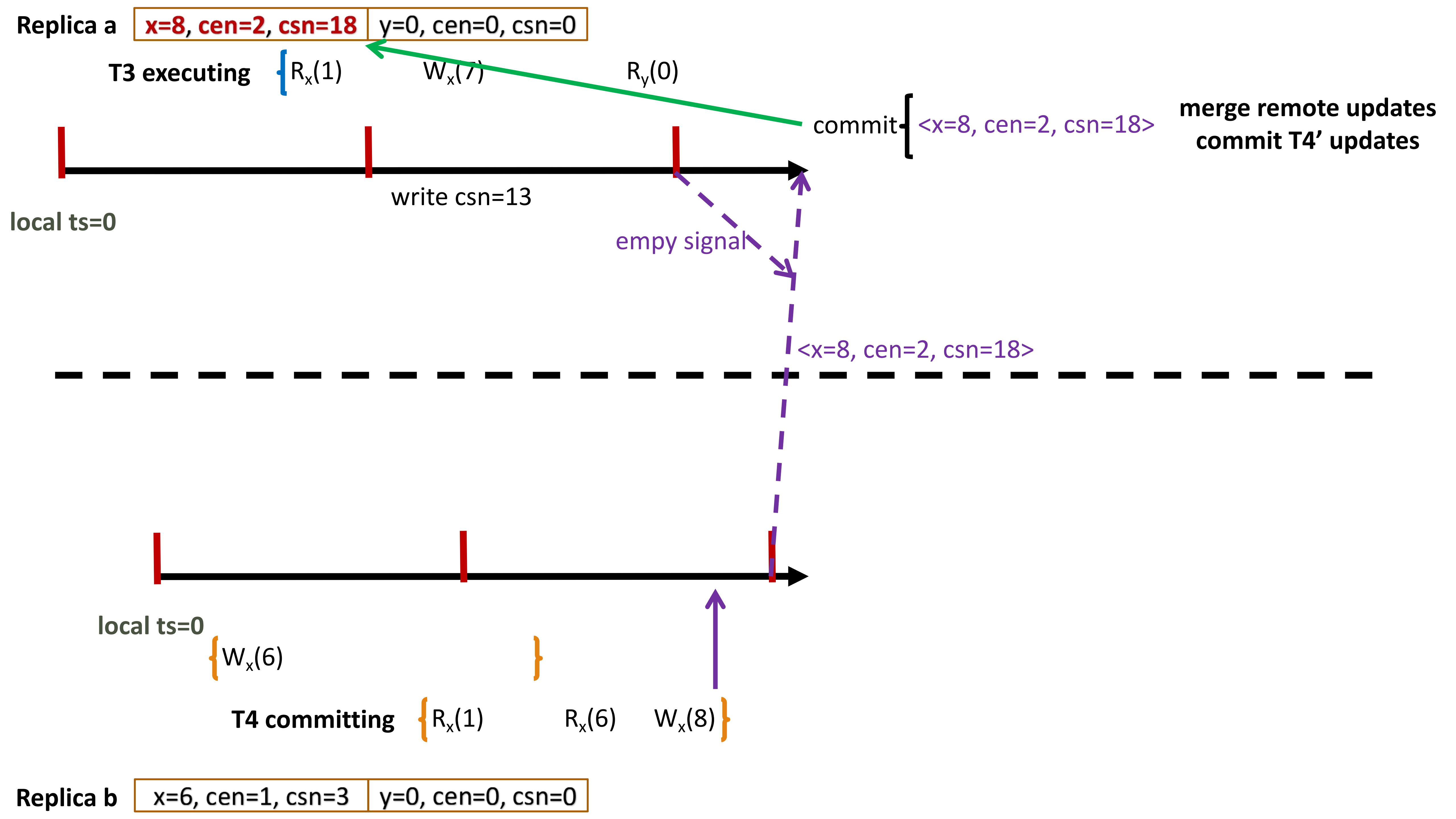

Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T3 executing
$R_x(1)$
$W_x(7)$
$R_y(0)$
epoch 2 ends,
send empty signal to other peers
write csn=13
local ts=0
local ts=20
send empty signal
local ts=0
T2
$W_x(6)$
… wait for merge …
T4 executing
$R_x(1)$
$R_x(6)$
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

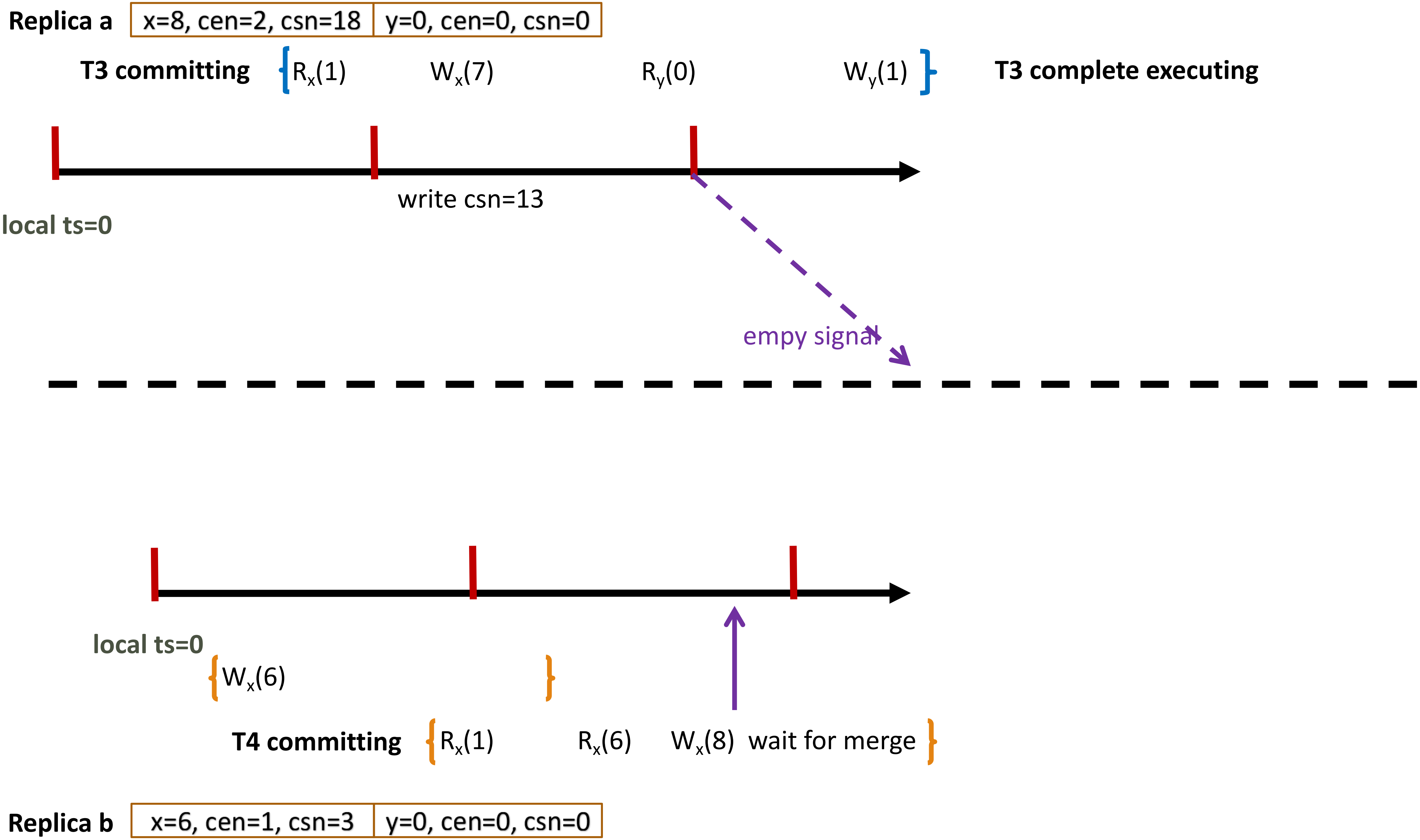


Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T3 executing
$R_x(1)$
$W_x(7)$
$R_y(0)$
write csn=13
local ts=0
empy signal
send <x=8, cen=2, csn=18>
to replica a
epoch 2 ends,
send updates to other peers
write csn=18
local ts=20
local ts=0
$W_x(6)$ … wait for merge …
<x=8, cen=2, csn=18>
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

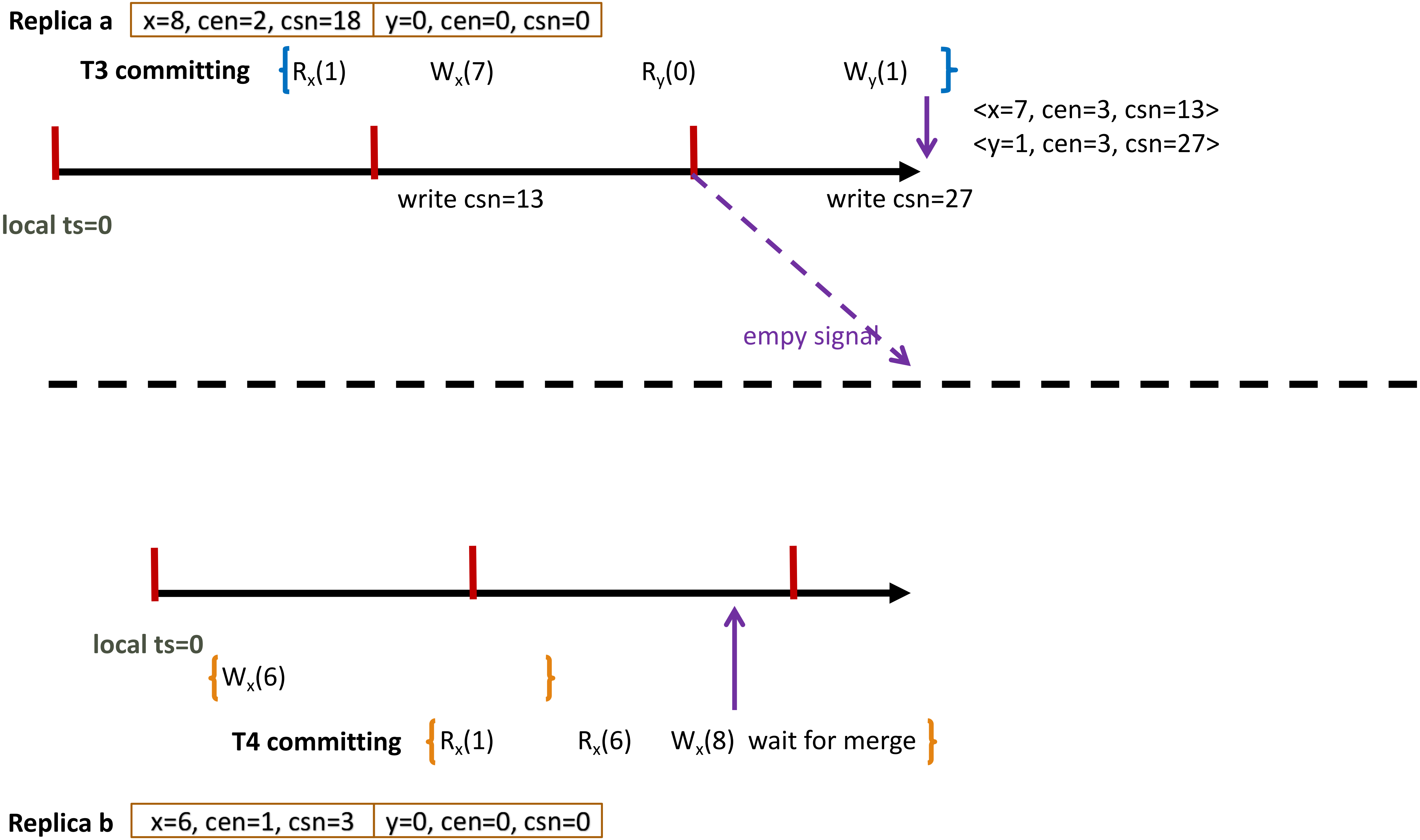


Replica a
x=6, cen=1, csn=3
y=0, cen=0, csn=0
T3 executing
$R_x(1)$
$W_x(7)$
$R_y(0)$
epoch merge
<x=8, cen=2, csn=18>
remote updates arrieve merge
write csn=13
local ts=0
empy signal
<x=8, cen=2, csn=18>
local ts=0
$W_x(6)$
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

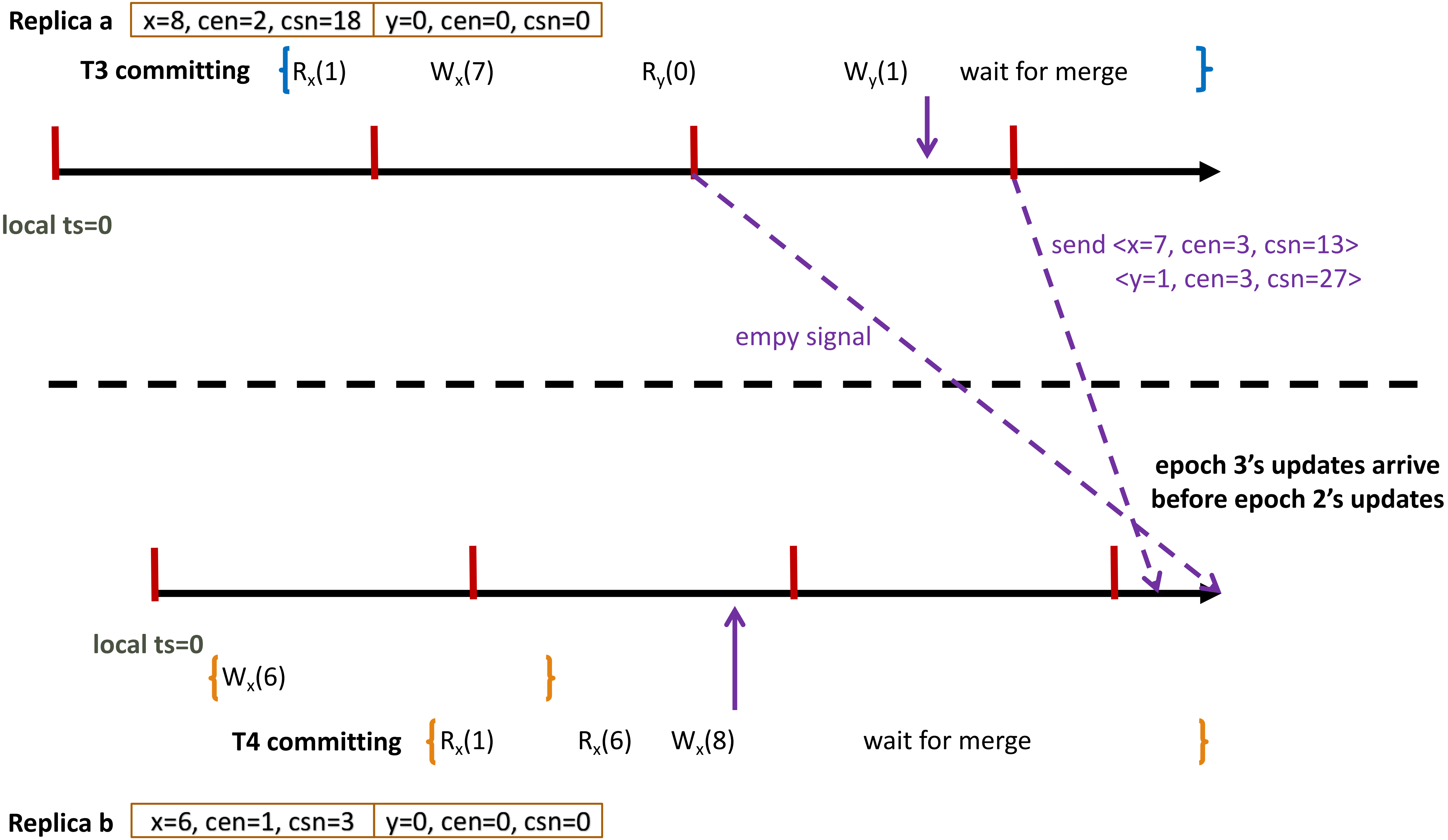

Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 executing
$R_x(1)$
$W_x(7)$
$R_y(0)$
commit
<x=8, cen=2, csn=18>
merge remote updates
commit T4' updates
write csn=13
local ts=0
empy signal
<x=8, cen=2, csn=18>
local ts=0
$W_x(6)$
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

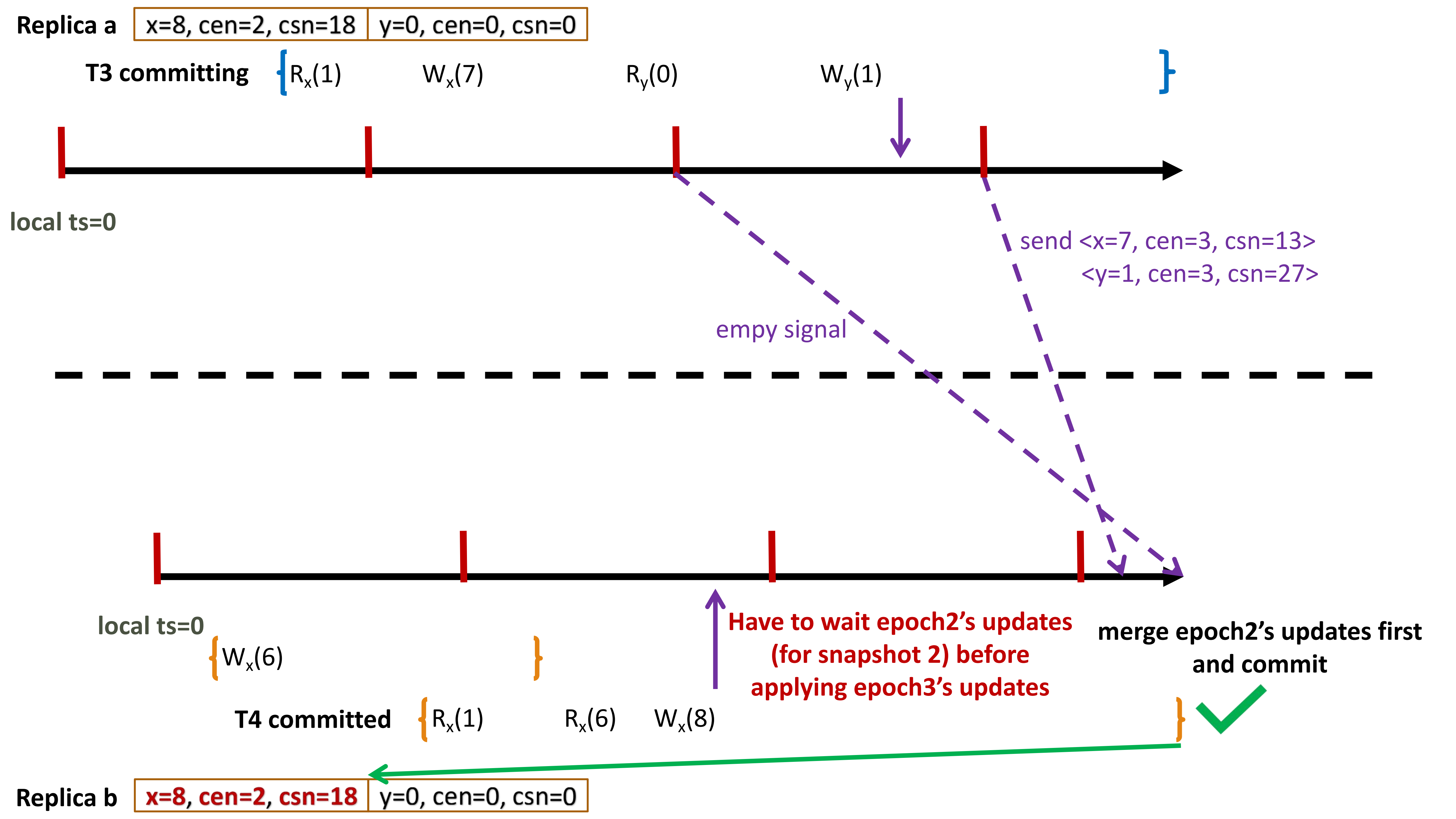


Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
T3 complete executing
write csn=13
local ts=0
empy signal
local ts=0
$W_x(6)$
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
wait for merge
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

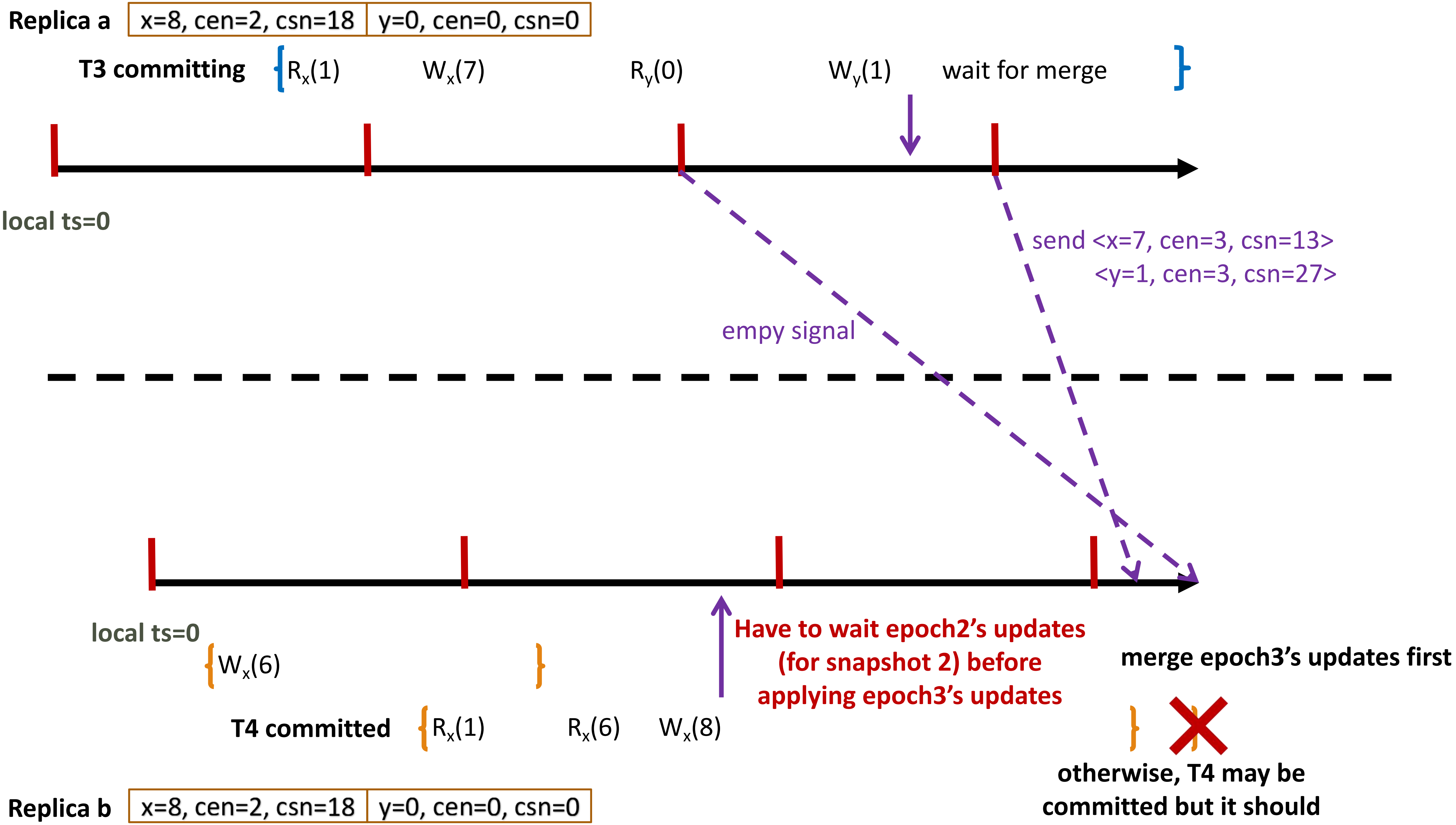


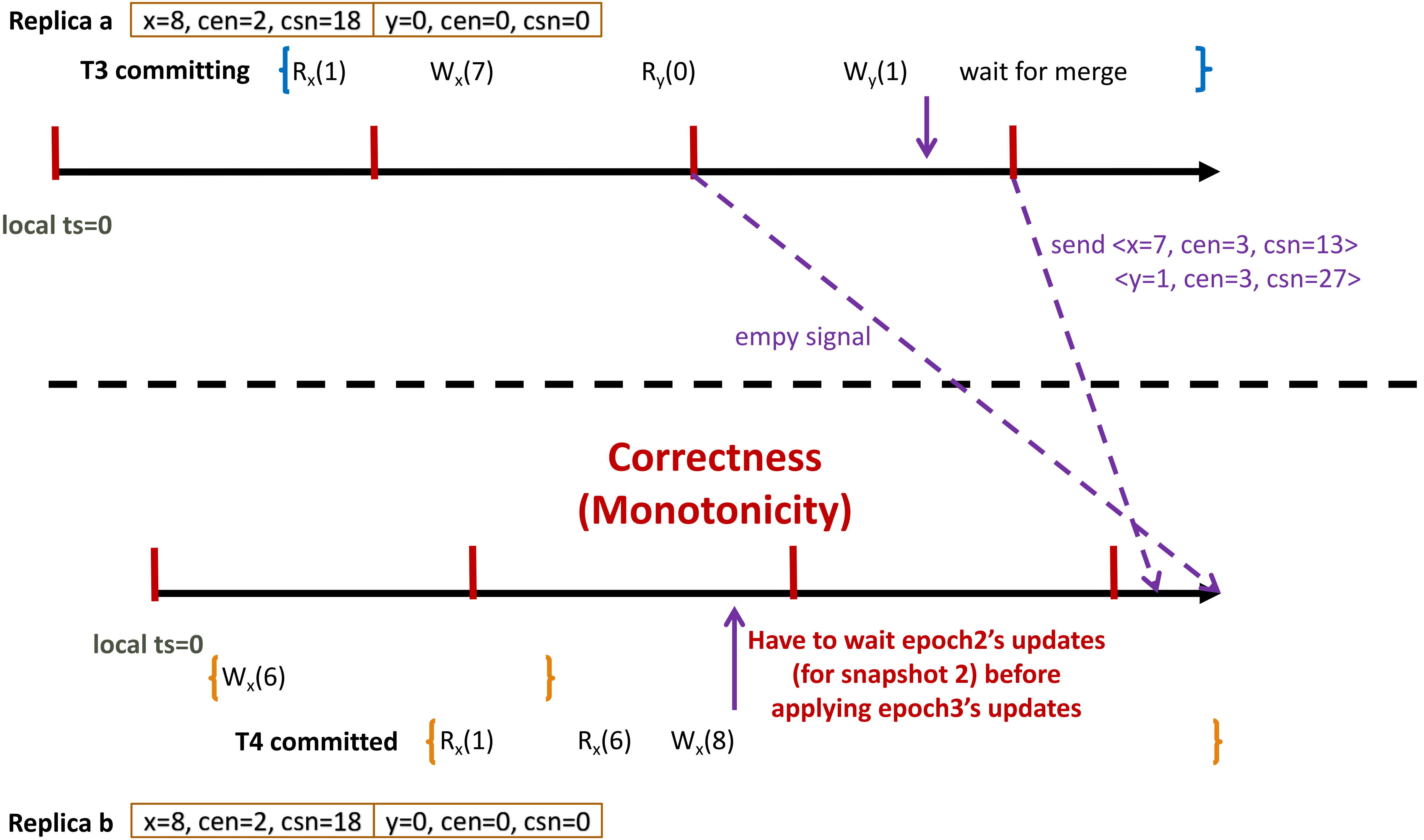
Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
<x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
write csn=13
write csn=27
local ts=0
empy signal
local ts=0
$W_x(6)$
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
wait for merge
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0

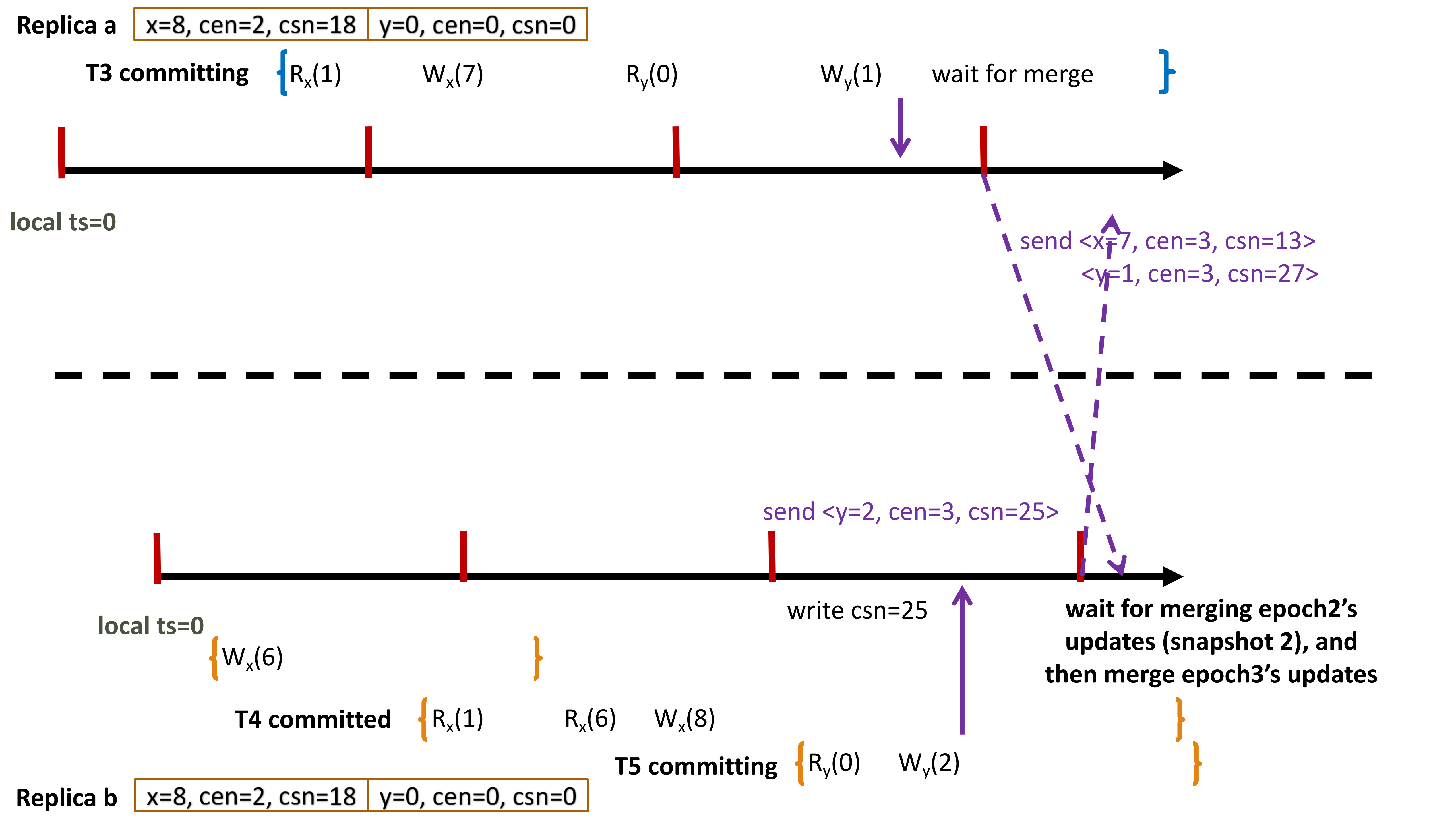

Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
wait for merge
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
empy signal
epoch 3's updates arrive
before epoch 2's updates
local ts=0
$W_x(6)$
T4 committing
$R_x(1)$
$R_x(6)$
$W_x(8)$
wait for merge
Replica b
x=6, cen=1, csn=3
y=0, cen=0, csn=0



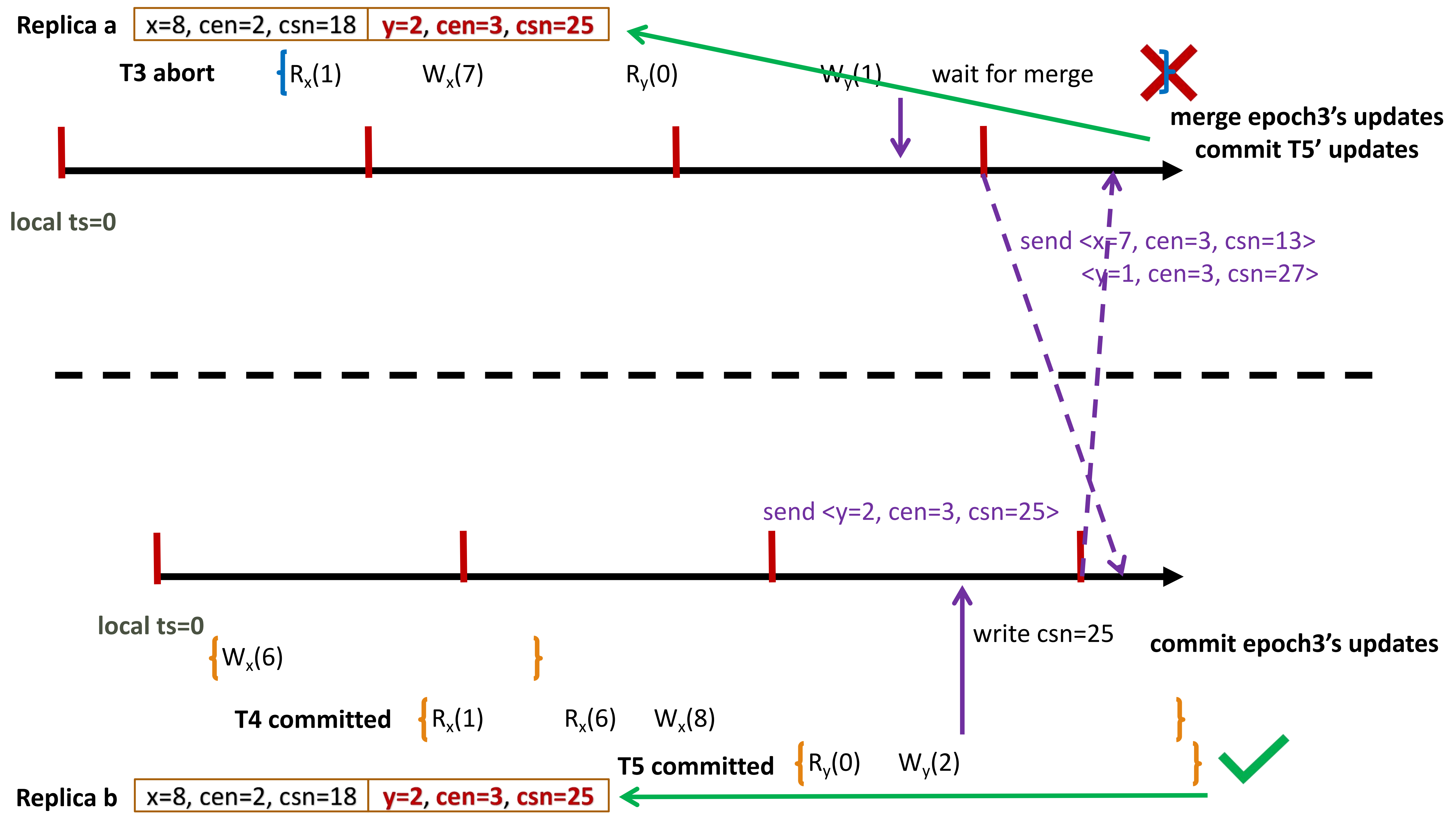
Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
empy signal
local ts=0
$W_x(6)$
Have to wait epoch2's updates
(for snapshot 2) before
applying epoch3's updates
merge epoch2's updates first
and commit
T4 committed
$R_x(1)$
$R_x(6)$
$W_x(8)$
Replica b
x=8, cen=2, csn=18
y=0, cen=0, csn=0


Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
wait for merge
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
empy signal
local ts=0
$W_x(6)$
Have to wait epoch2's updates
(for snapshot 2) before
applying epoch3's updates
merge epoch3's updates first
T4 committed
$R_x(1)$
$R_x(6)$
$W_x(8)$
otherwise, T4 may be
committed but it should
Replica b
x=8, cen=2, csn=18
y=0, cen=0, csn=0

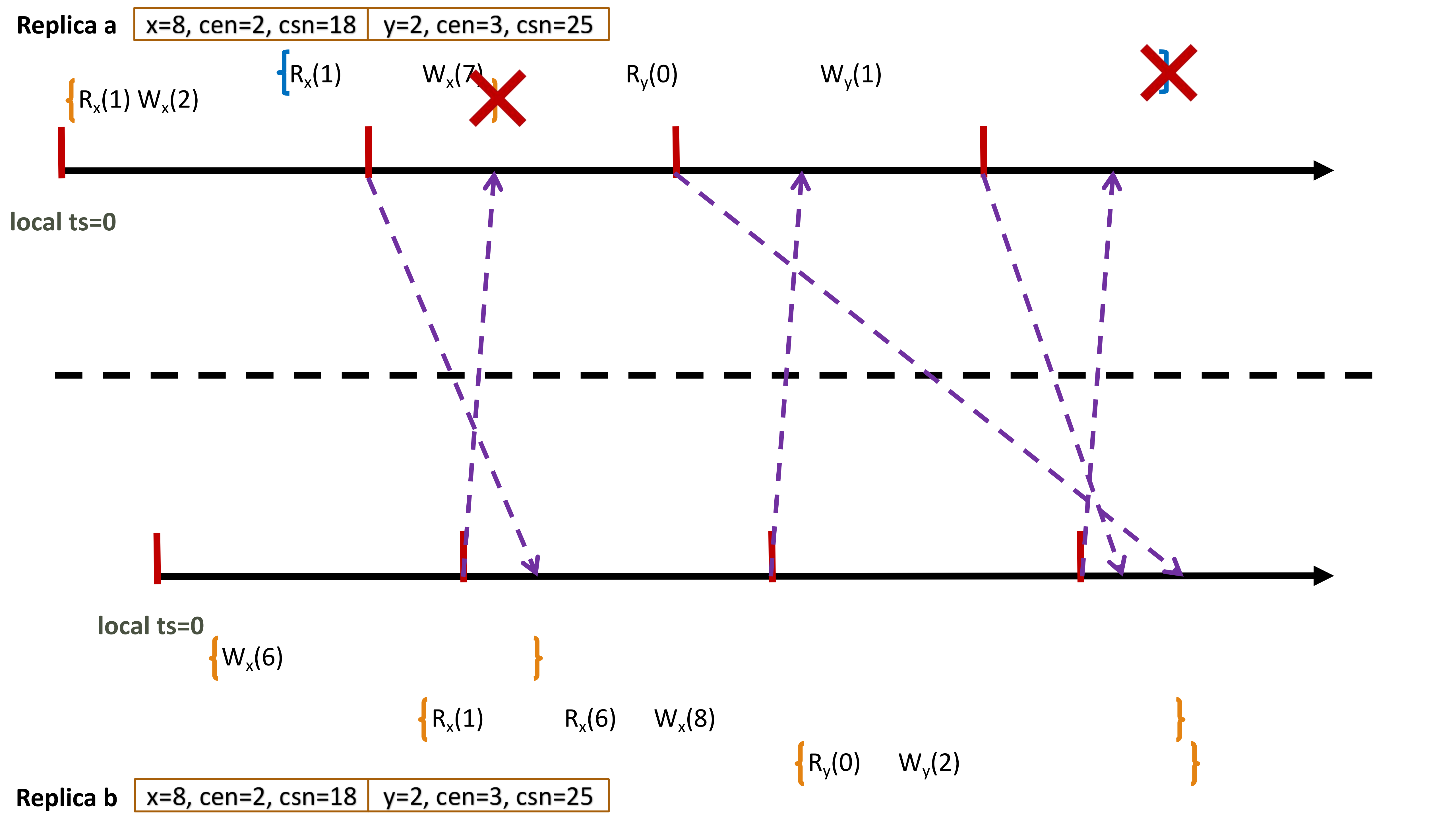

Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
wait for merge
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
empy signal
Correctness
(Monotonicity)
local ts=0
$W_x(6)$
Have to wait epoch2's updates
(for snapshot 2) before
applying epoch3's updates
T4 committed
$R_x(1)$
$R_x(6)$
$W_x(8)$
Replica b
x=8, cen=2, csn=18
y=0, cen=0, csn=0



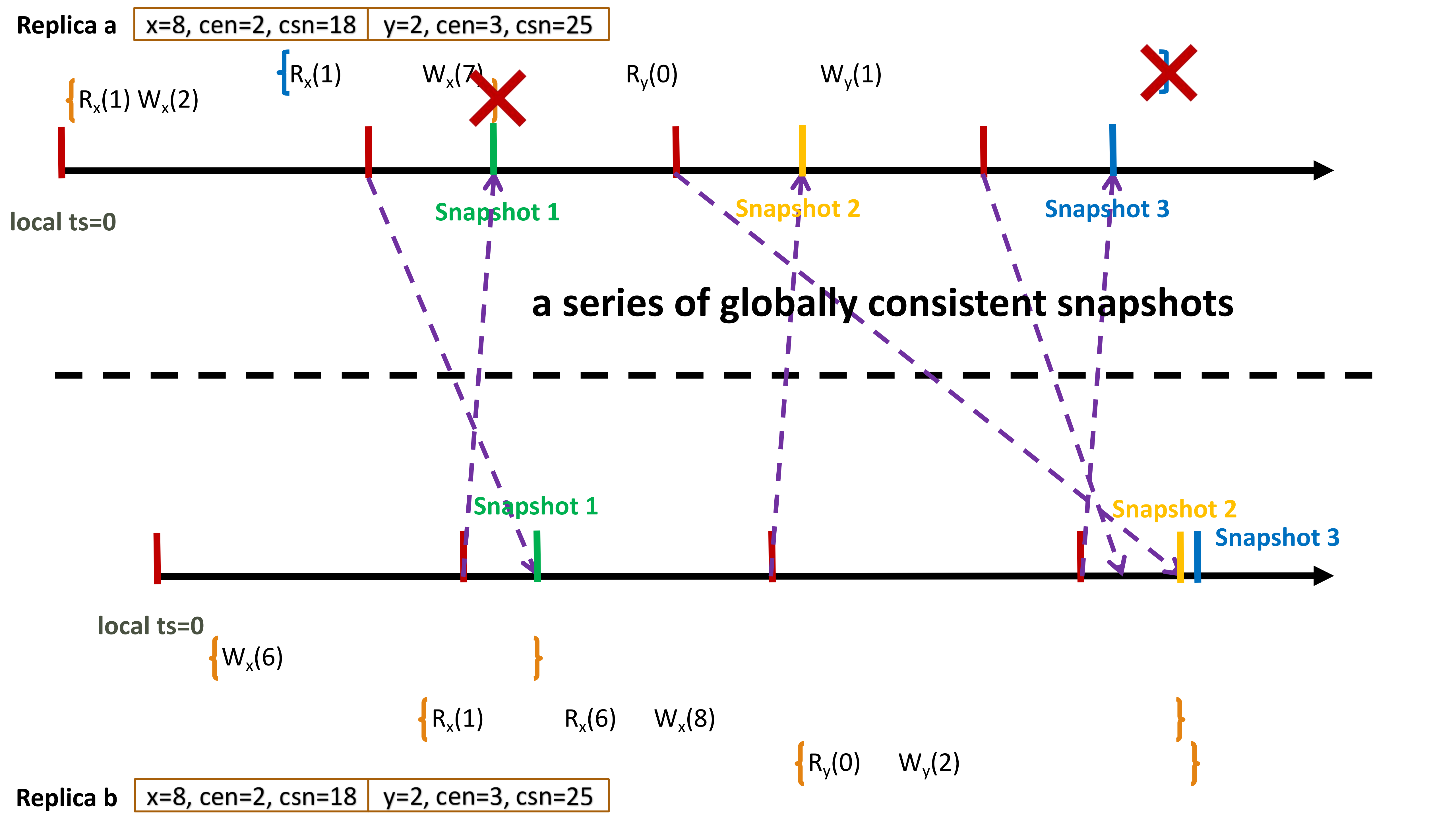
Replica a
x=8, cen=2, csn=18
y=0, cen=0, csn=0
T3 committing
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
wait for merge
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
send <y=2, cen=3, csn=25>
write csn=25
wait for merging epoch2's updates (snapshot 2), and then merge epoch3's updates
local ts=0
$W_x(6)$
T4 committed
$R_x(1)$
$R_x(6)$
$W_x(8)$
T5 committing
$R_y(0)$
$W_y(2)$
Replica b
x=8, cen=2, csn=18
y=0, cen=0, csn=0



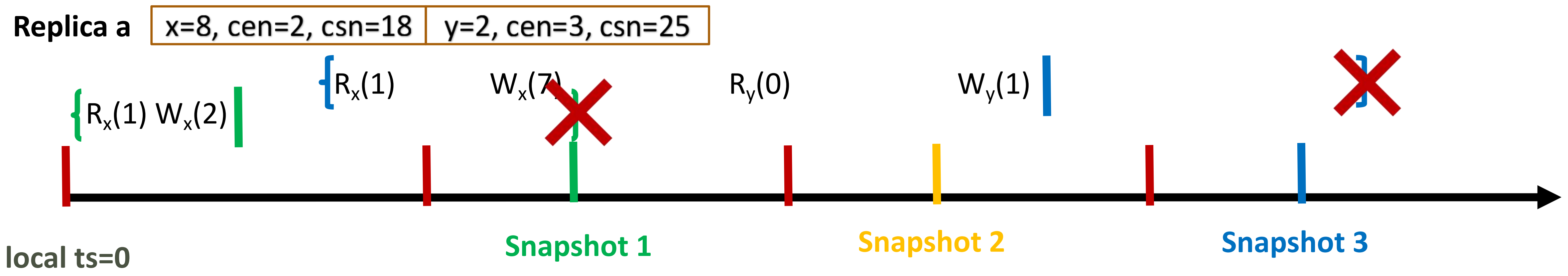
Replica a
x=8, cen=2, csn=18
y=2, cen=3, csn=25
T3 abort
Rx(1)
Wx(7)
Ry(0)
Wy(1)
wait for merge
merge epoch3's updates
commit T5' updates
local ts=0
send <x=7, cen=3, csn=13>
<y=1, cen=3, csn=27>
send <y=2, cen=3, csn=25>
local ts=0
Wx(6)
write csn=25
commit epoch3's updates
T4 committed
Rx(1)
Rx(6)
Wx(8)
T5 committed
Ry(0)
Wy(2)
Replica b
x=8, cen=2, csn=18
y=2, cen=3, csn=25



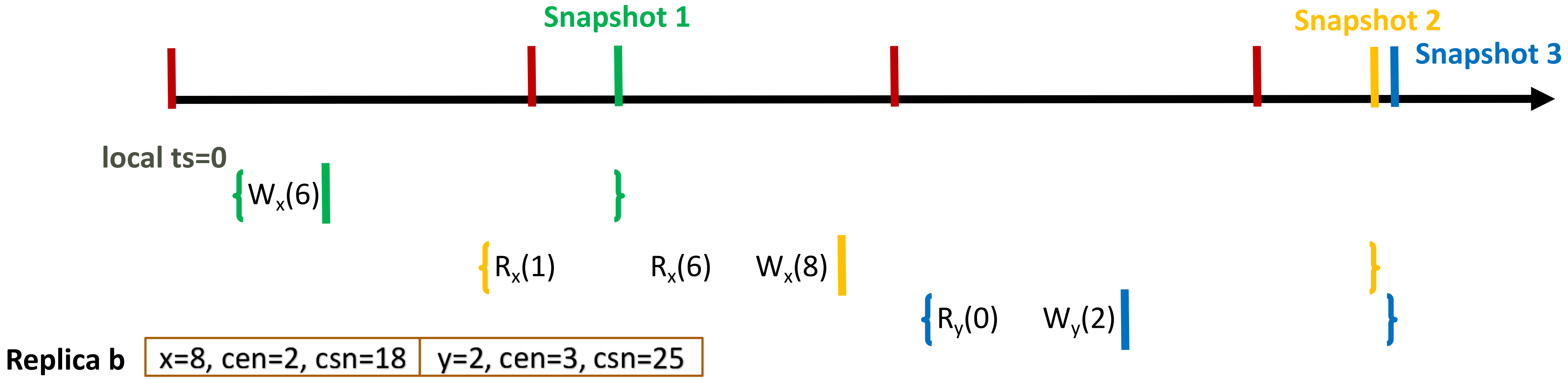
Replica a
x=8, cen=2, csn=18
y=2, cen=3, csn=25
{Rx(1)
Wx(7)
}
Ry(0)
Wy(1)
}
{Rx(1) Wx(2)
local ts=0
local ts=0
{Wx(6)
}
{Rx(1)
Rx(6)
Wx(8)
}
{Ry(0)
Wy(2)
}
Replica b
x=8, cen=2, csn=18
y=2, cen=3, csn=25



Replica a
x=8, cen=2, csn=18
y=2, cen=3, csn=25
$R_x(1)$ $W_x(2)$
$R_x(1)$
$W_x(7)$
$R_y(0)$
$W_y(1)$
local ts=0
Snapshot 1
Snapshot 2
Snapshot 3
a series of globally consistent snapshots
Snapshot 1
Snapshot 2
Snapshot 3
local ts=0
$W_x(6)$
$R_x(1)$
$R_x(6)$
$W_x(8)$
$R_y(0)$
$W_y(2)$
Replica b
x=8, cen=2, csn=18
y=2, cen=3, csn=25



Tx does not knows its commit/abort status until the snapshot of its commit epoch is generated(after merge)



More Details

More:

- DeltaCRDTMerge algorithm
- Isolation levels
- Consistency Proof
- Optimization
- Fault Tolerance
- …

GeoGauss: Strongly Consistent and Light-Coordinated OLTP for Geo-Replicated SQL Database

WEIXING ZHOU and QI PENG, Northeastern University, China
ZIJIE ZHANG, Huawei Technology Co., Ltd, China
YANFENG ZHANG*, Northeastern University, China
YANG REN and SIHAO LI, Huawei Technology Co., Ltd, China
GUO FU and YULONG CUI, Northeastern University, China
QIANG LI, Huawei Technology Co., Ltd, China
CAIYI WU, SHANGJUN HAN, and SHENGYI WANG, Northeastern University, China
GUOLIANG LI, Tsinghua University, China
GE YU, Northeastern University, China

Multinational enterprises conduct global business that has a demand for geo-distributed transactional databases. Existing state-of-the-art databases adopt a sharded master-follower replication architecture. However, the single-master serving mode incurs massive cross-region writes from clients, and the sharded architecture requires multiple round-trip acknowledgments (*e.g.*, 2PC) to ensure atomicity for cross-shard transactions. These limitations drive us to seek yet another design choice. In this paper, we propose a strongly consistent OLTP database GeoGauss with full replica multi-master architecture. To efficiently merge the updates from different master nodes, we propose a multi-master OCC that unifies data replication and concurrent transaction processing. By leveraging an epoch-based delta state merge rule and the optimistic asynchronous execution, GeoGauss ensures strong consistency with light-coordinated protocol and allows more concurrency with weak isolation, which are sufficient to meet our needs. Our geo-distributed experimental results show that GeoGauss achieves 7.06X higher throughput and 17.41X lower latency than the state-of-the-art geo-distributed database CockroachDB on the TPC-C benchmark.

CCS Concepts: • **Information systems → Relational parallel and distributed DBMSs.**

Additional Key Words and Phrases: Geo-distributed; multi-master replication; replica consistency; transaction processing; deterministic databases

ACM Reference Format:
Weixing Zhou, Qi Peng, Zijie Zhang, Yanfeng Zhang, Yang Ren, Sihao Li, Guo Fu, Yulong Cui, Qiang Li, Caiyi Wu, Shangjun Han, Shengyi Wang, Guoliang Li, and Ge Yu. 2023. GeoGauss: Strongly Consistent and Light-Coordinated OLTP for Geo-Replicated SQL Database. *Proc. ACM Manag. Data* 1, 1, Article 62 (May 2023), 27 pages. https://doi.org/10.1145/3588916

62

*Yanfeng Zhang is the corresponding author.

Authors' addresses: Weixing Zhou, Qi Peng, Yanfeng Zhang, Guo Fu, Yulong Cui, Caiyi Wu, Shangjun Han, Shengyi Wang, Ge Yu, Northeastern University, No. 195, Chuangxin Road, Hunnan District, Shenyang, Liaoning, China, 110169, {zhouwx@stumail, ffpengqi@stumail, Zhangyf@mail, yuge@mail}.neu.edu.cn; Zijie Zhang, Yang Ren, Sihao Li, Qiang Li, Huawei Technology Co., Ltd, Xian, Shanxi, China, {zhangzijie9, renyang1, sean.lisihao, liqiang199}@huawei.com; Guoliang Li, Tsinghua University, 30 Shuangqing Road, Haidian District, Beijing, China, liguoliang@tsinghua.edu.cn;

2836-6573/2023/5-ART62 $15.00
https://doi.org/10.1145/3588916

Evaluation

Cluster Set Up:

- 3 geo-distributed nodes

  Chengdu (Southwest China),

  Shenzhen (South China),

  Zhangjiakou (North China).
- 32 vCPUs, 256G DRAM, Centos 7.6
- 100Mbps

Benchmark:

- YCSB[1]
  - YCSB-RO (100% read)
  - YCSB-MC (80% read, 20% write, $\theta = 0.9$)
- TPC-C[2]

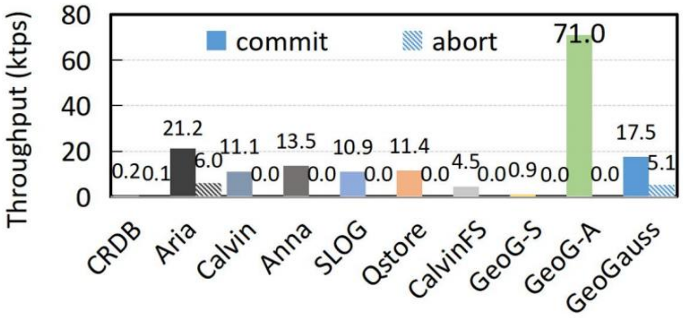
1: YCSB 10 op/txn

2: 50% New-Order – 50% Payment in Aria[VLDB2020]

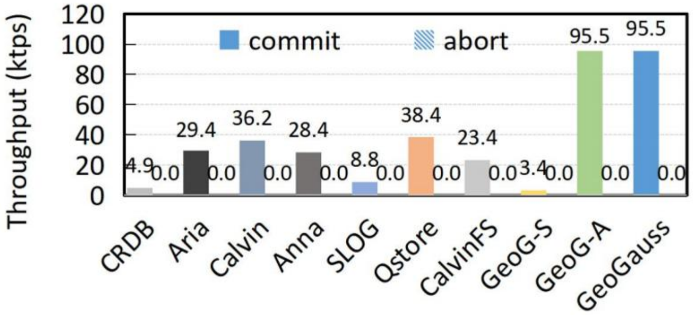
Inter- and intra-cluster ping round-trip times (latency)

	Chengdu	Shenzhen	Zhangjiakou
Chengdu (Southwest China)	0.2 ms	37.5 ms	57.4 ms
Shenzhen (South China)		0.2 ms	38.3 ms
Zhangjiakou (North China)			0.2 ms

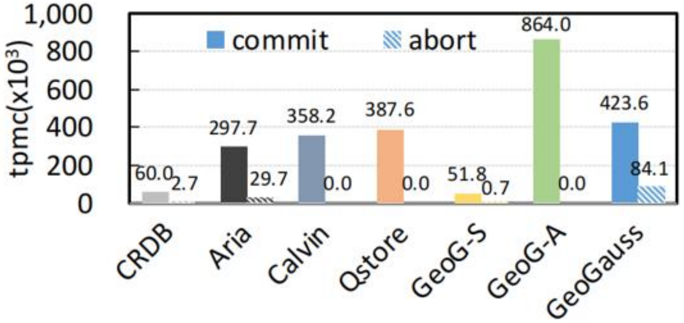
Overall Performance



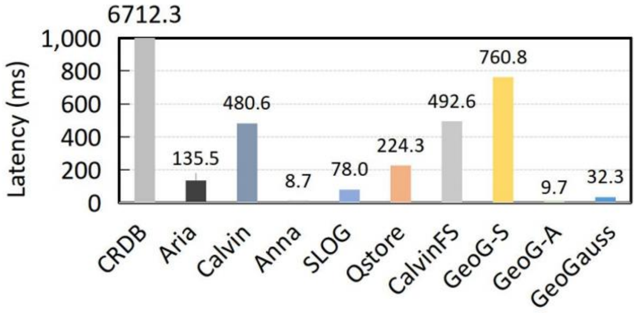
YCSB-MC Throughput



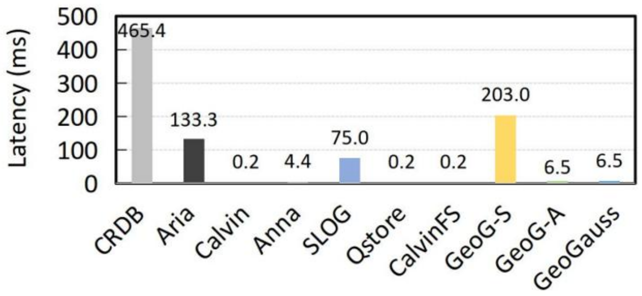
YCSB-RO Throughput



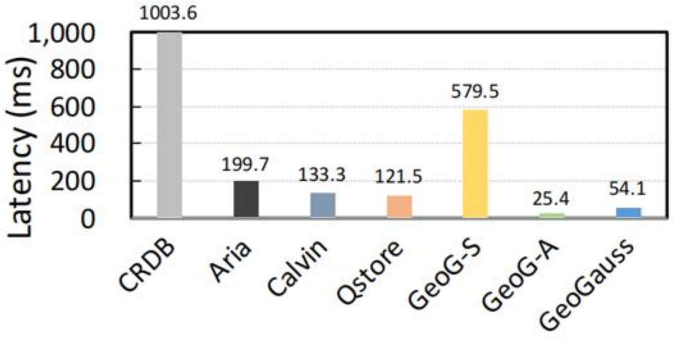
TPC-C Throughput



YCSB-MC Latency



YCSB-RO Latency



TPC-C Latency

System Breakdown

GeoG-S : *synchronous* execution and synchronous validation (*heavy coordination*)

GeoG-A : asynchronous execution and *asynchronous* validation (*eventual consistency*)

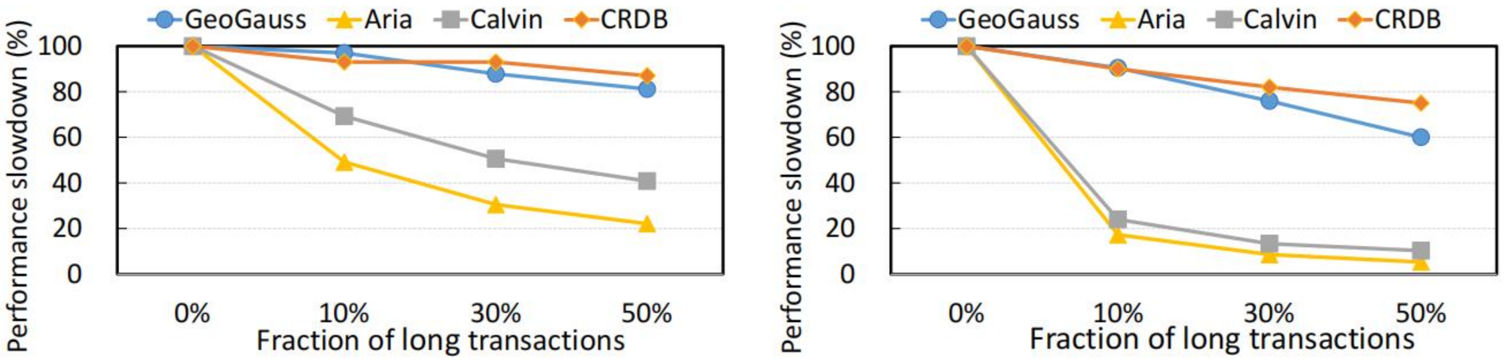
GeoGauss : asynchronous execution and synchronous validation

- Avoid long waits by asynchronous execution
- Achieve sequential consistency by synchronous validation

Table 2. Runtime breakdown of a transaction (TPC-C).

	GeoG-S	GeoG-A	GeoGauss
SQL Parse	4.6 ms	4.6 ms	4.6 ms
Execute	5.8 ms	6.5 ms	4.8 ms
Wait	**564.2 ms**	**0 ms**	**34.1 ms**
Merge	4.0 ms	10.9 ms	9.4 ms
Log	0.8 ms	6.5 ms	4.7 ms

Long Transaction



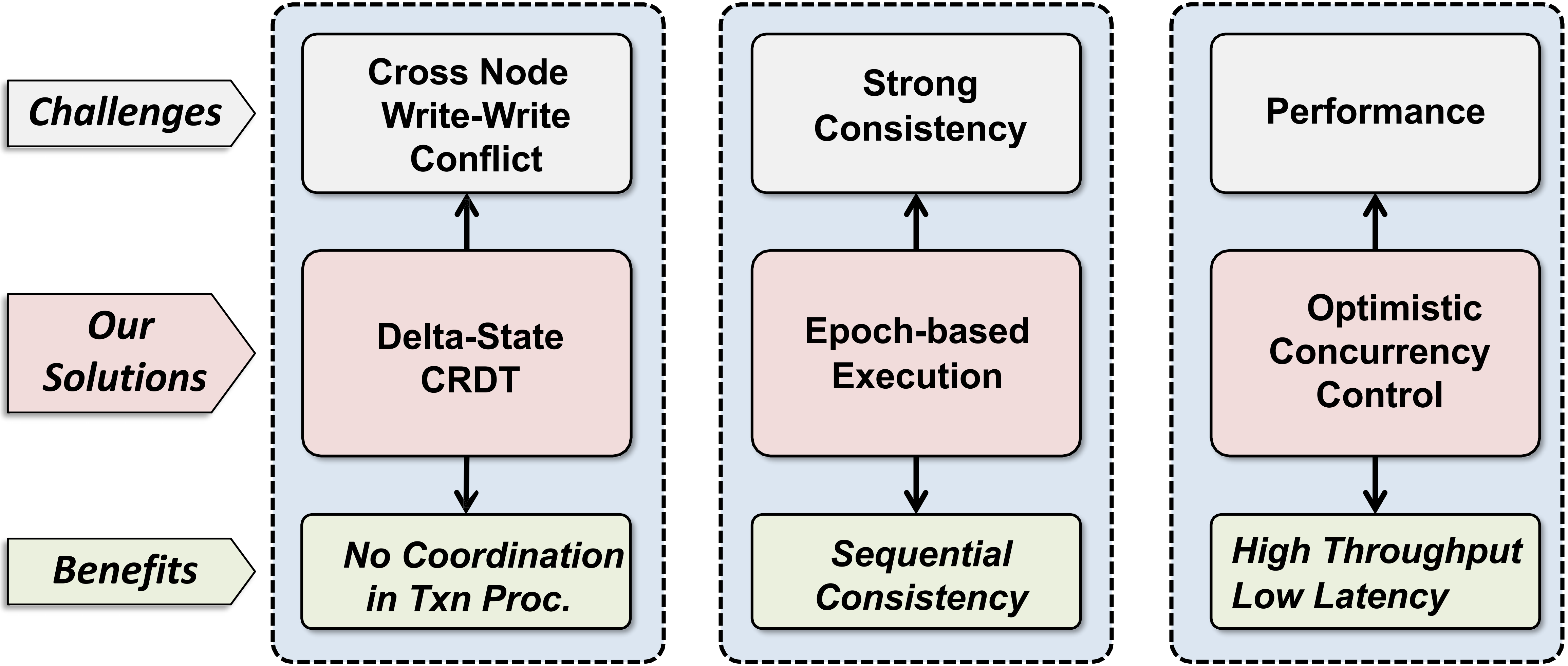
(a) Delay = 20ms (b) Delay = 100ms

Fig. 7. Effect of long transactions (YCSB-MC).

CRDB(CockroachDB) : sharded master-follower DB

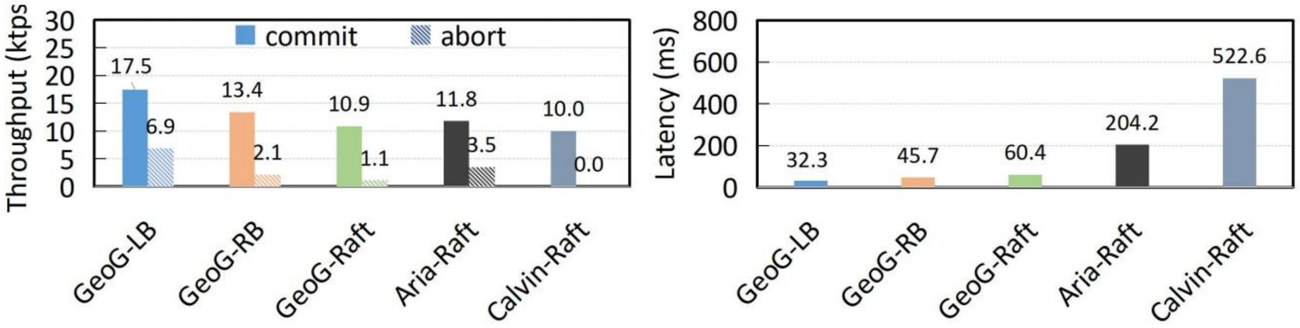
Calvin & Aria : Deterministic DB

Conclusion


Challenges
Our Solutions
Benefits
Cross Node Write-Write Conflict
Delta-State CRDT
No Coordination in Txn Proc.
Strong Consistency
Epoch-based Execution
Sequential Consistency
Performance
Optimistic Concurrency Control
High Throughput Low Latency

Thank you! Q&A

Fault tolerance



(a) throughput (b) latency

Fig. 12. Performance with fault tolerance (YCSB-MC).