Approximately Opaque Multi-version Transactional Memory

Basem Assiri

Costas Busch

Louisiana State University

Outline

- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- Transactional Memory and Consistency
- Approximately Opaque TM for Read-only Transactions
- Results
- Garbage Collector
- Conclusion

Outline

- **Challenges of Distributed Computing and Performance**
- **Objectives and Motivation**
- **Transactional Memory and Consistency**
- **Approximately Opaque TM for Read-only Transactions**
- **Results**
- **Garbage Collector**
- **Conclusion**

The Challenges of Distributed Computing and Performance

A multi-core revolution aims to improve the performance through concurrent computing. However, synchronizing memory accesses makes writing concurrent applications much harder than sequential ones.

Transactional Memory

Transactional memory is an important way to cope with such challenge!

Transactional memory allows concurrent accesses by using the concept of transaction

Transaction is a piece of code or a finite sequence of instructions that access local and shared memory - read-only and update.

Transactions **c**ommit or abort

Outline

- Challenges of Distributed Computing and Performance
- **Objectives and Motivation**
- Transactional Memory and Consistency
- Approximately Opaque TM for Read-only transaction
- Results
- Garbage Collector
- Conclusion

Objectives and Motivation

- **Improve TM performance** (increase throughput).

The precision level in consistency.

- 1- **In the large scale network systems**
- 2- **long read-only transactions**
- 3- **In real life, some types of systems do not require the precise computations.**

Inventory queries, Decision Support Systems, Advertising and Recommendation Systems and Sensors Systems.

None-sensitive data and frequent changes.

Outline

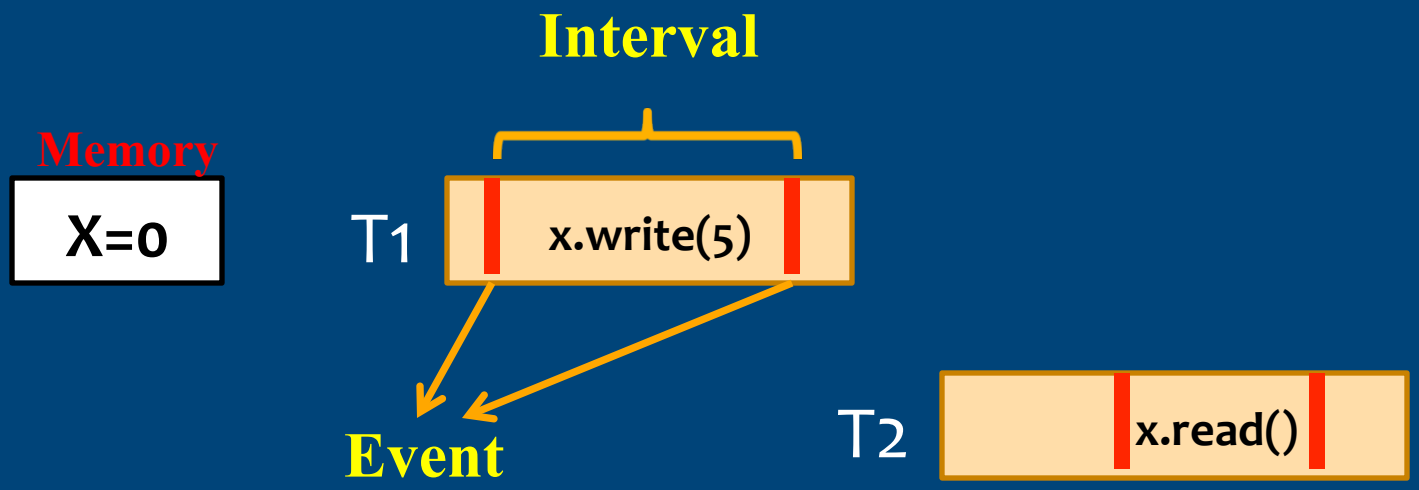
- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- **Transactional Memory and Consistency**
- Approximately Opaque TM for Read-only Transactions
- Results
- Garbage Collector
- Conclusion

TM Consistency

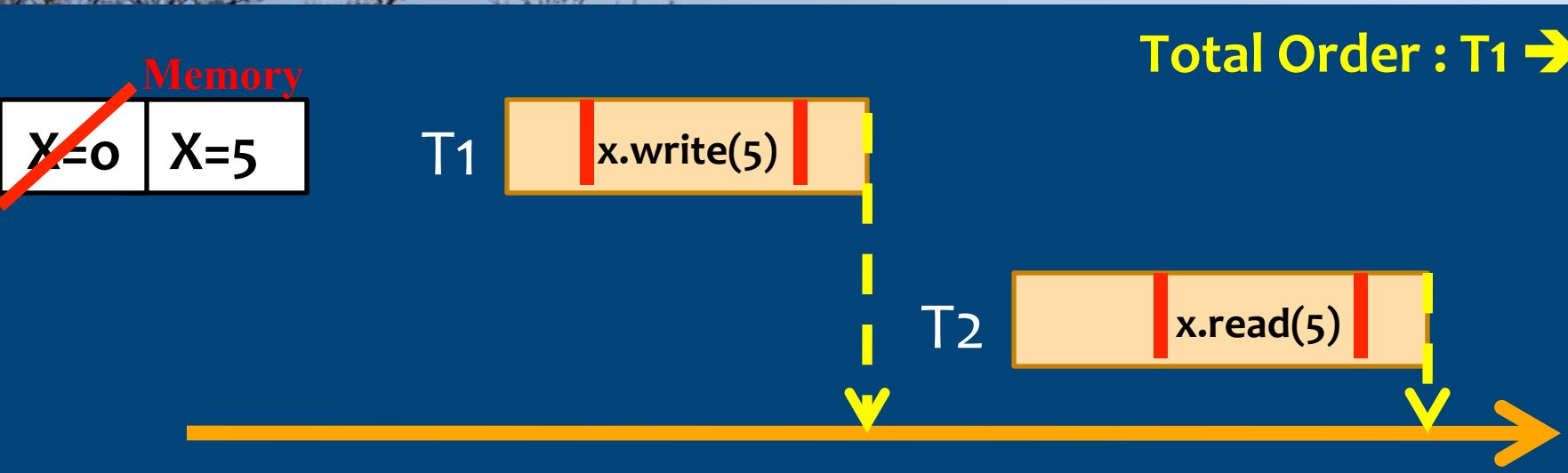
- We can easily prove the correctness of sequential execution. So, we prove the correctness of any concurrent execution if we can match it with a sequential one.

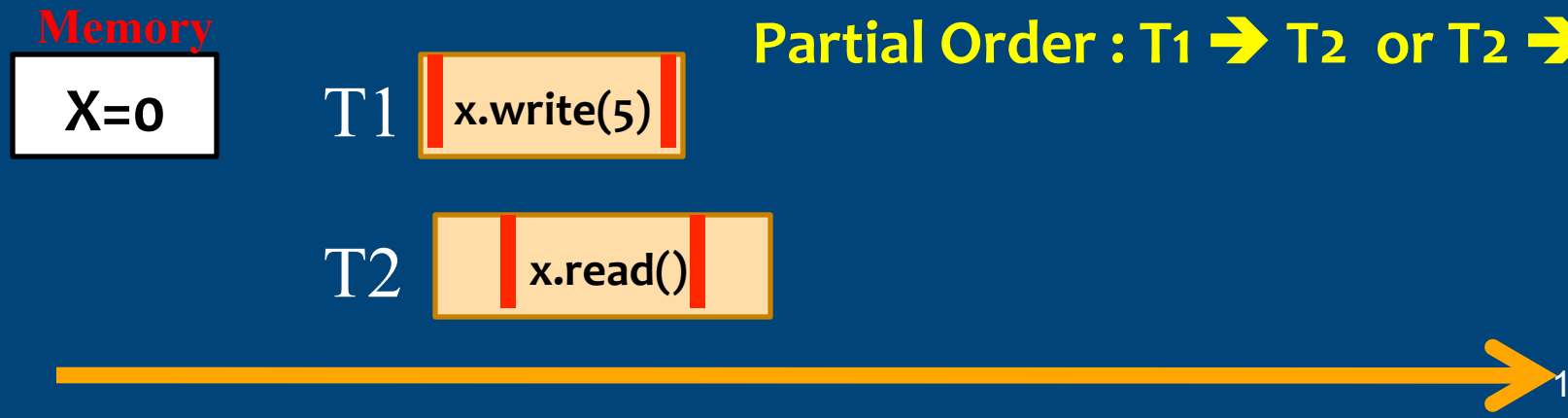
TM Consistency

- Event: "x.write()" = invocation + response
- History (H): a sequence of events.



TM Consistency (Order)

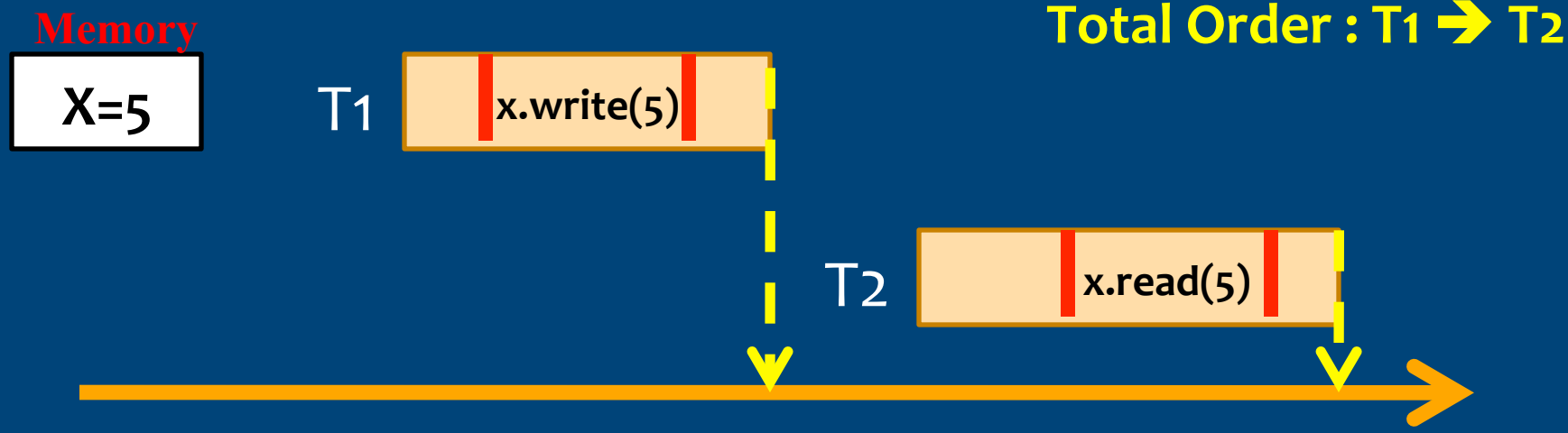




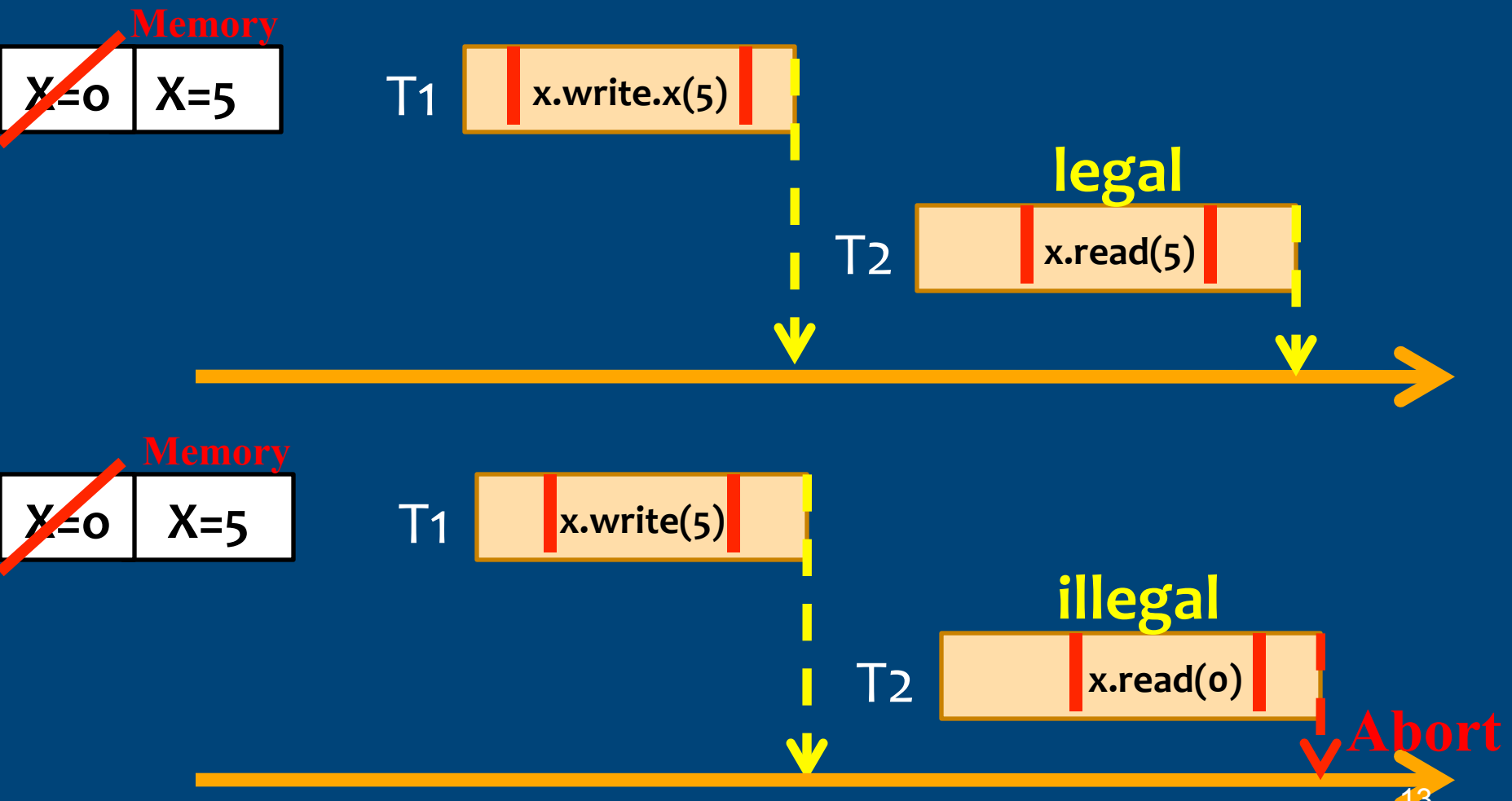
TM Consistency (Order)

Complete History (H'): **all transactions are either committed or aborted.**

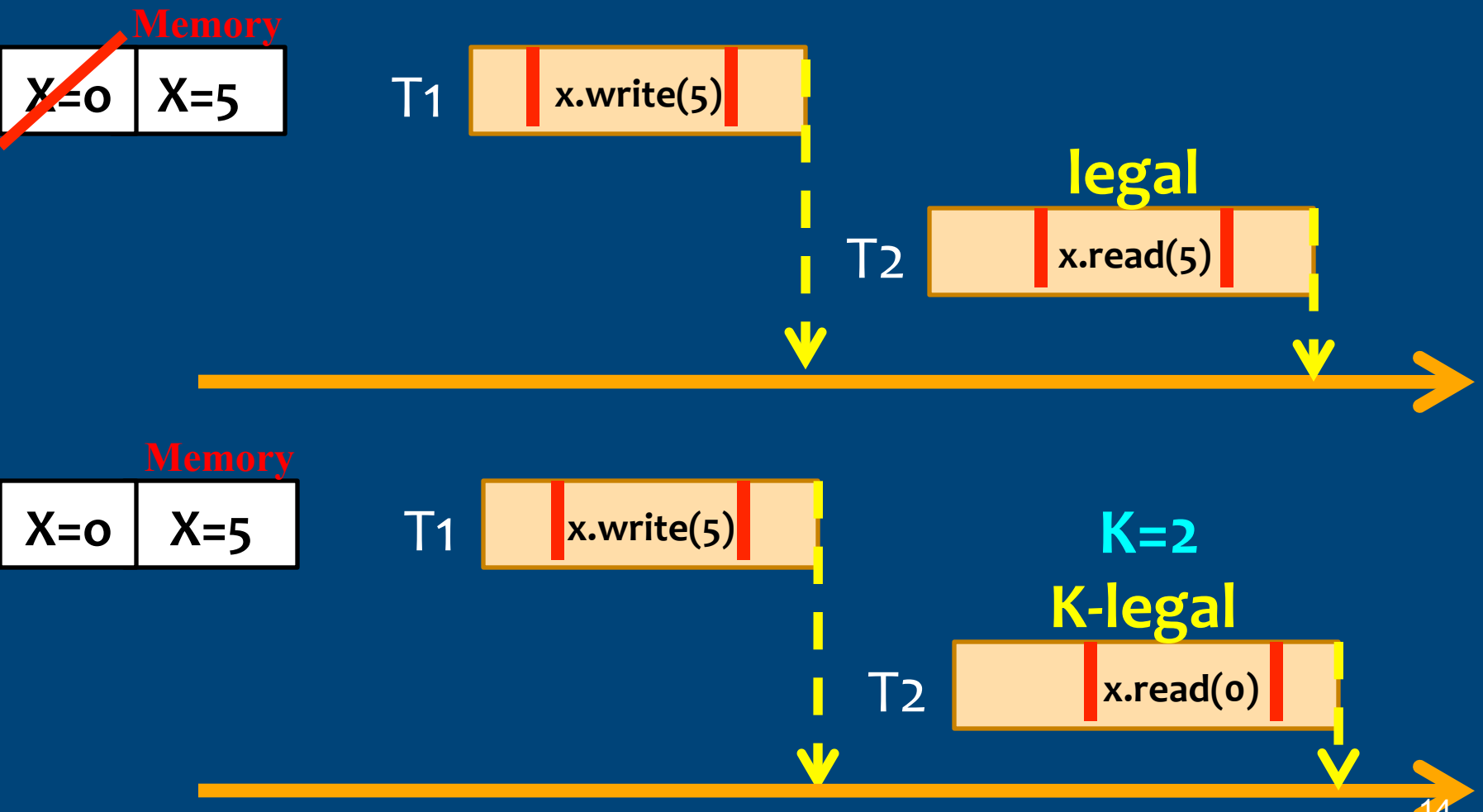
Sequential History (S): **is complete, and is a total order.**



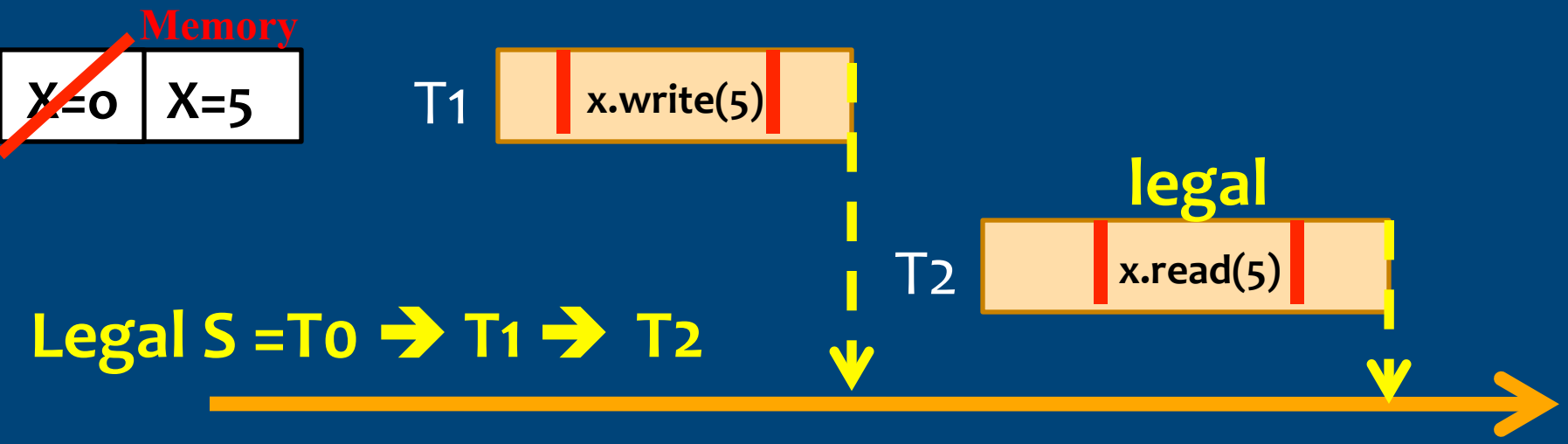
TM Consistency (Legality)

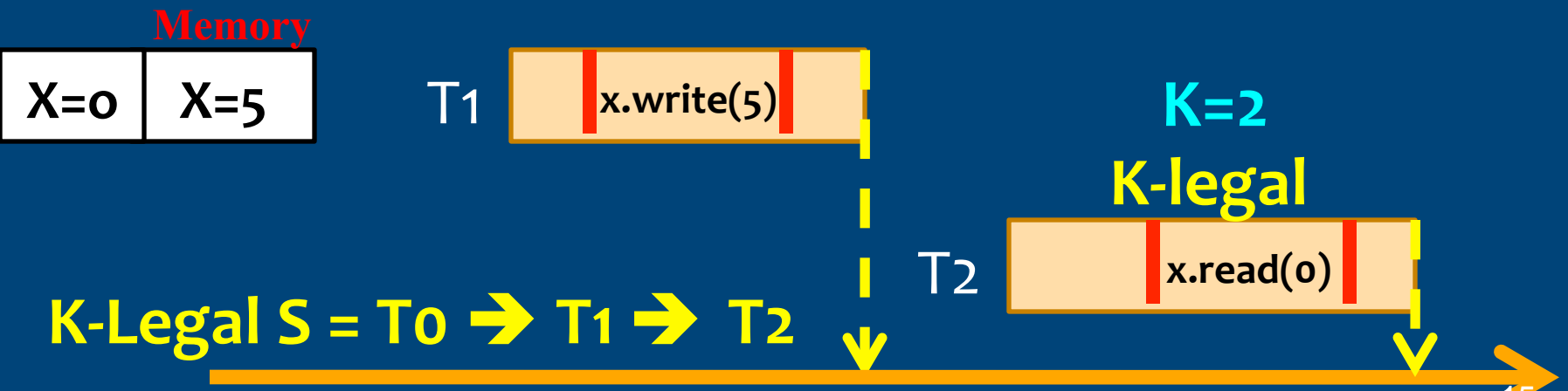


TM Consistency (K-legality)



TM Consistency (K-legality)





TM Consistency (Opacity)

- Opacity:

Concurrent H -> H' -> *legal* S where S respects the *real time order* of transactions in H.

- K-opacity:

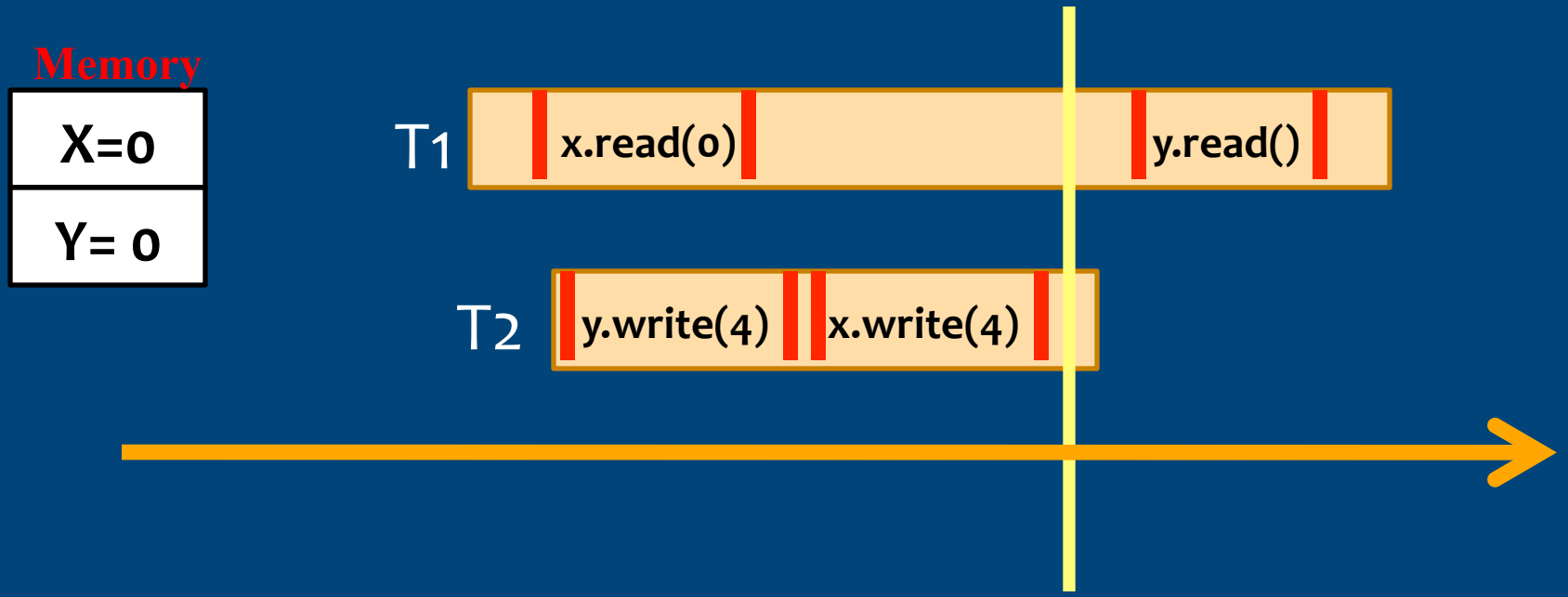
Concurrent H -> H' -> *K-legal* S where S respects the *real time order* of transactions in H.

Outline

- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- Transactional Memory and Consistency
- **Approximately Opaque TM for Read-only Transactions**
- Results
- Garbage Collector
- Conclusion

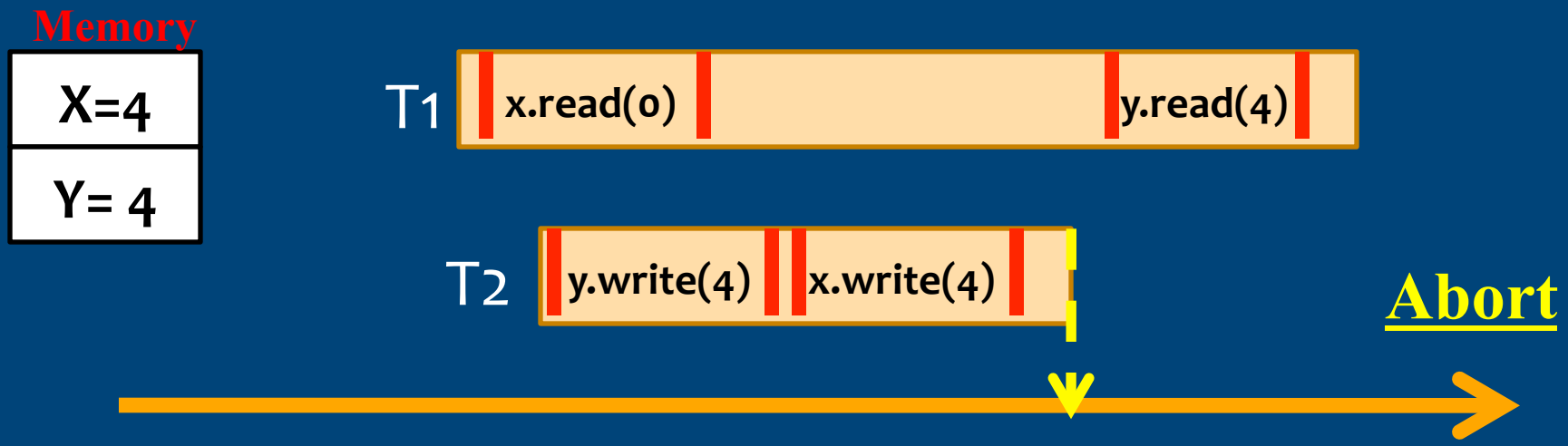
Approximately Opaque TM for Read-only Transactions

- Two concurrent read-only and update transactions sharing two objects (x and y)



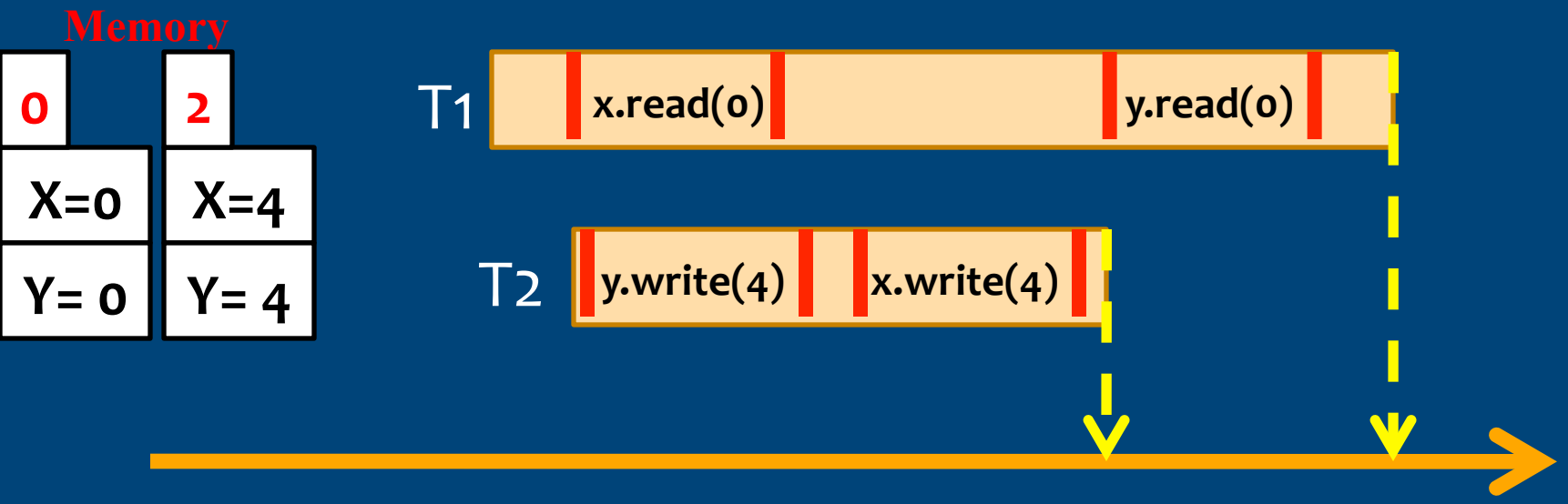
Approximately Opaque TM for Read-only Transactions

- Two concurrent read-only and update transactions sharing two objects (x and y)



Multi-version

- Two concurrent read-only and update transactions sharing two objects (x and y)



S= T1 → T2

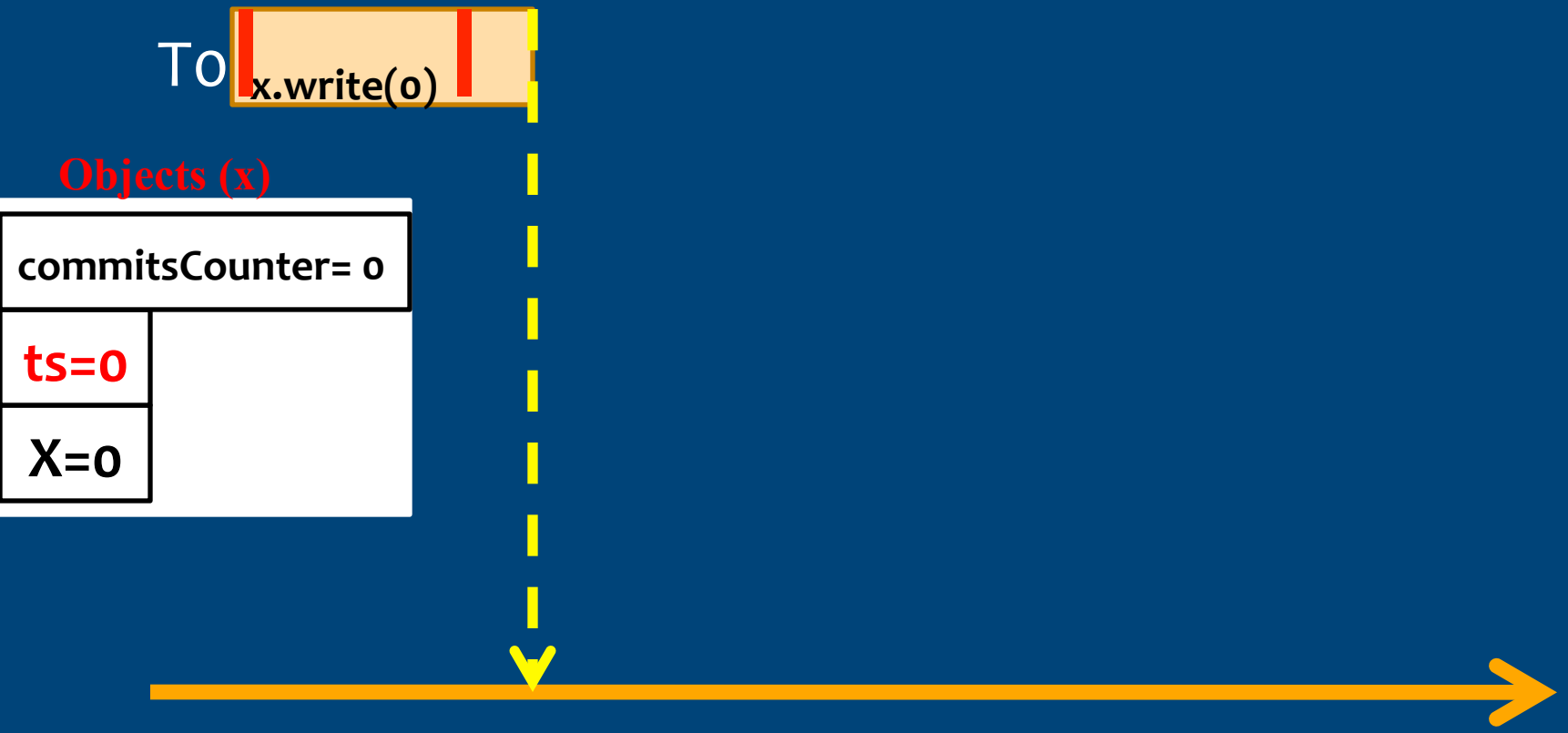
Approximately Opaque TM for Read-only Transactions

We relax the opacity to K-opacity

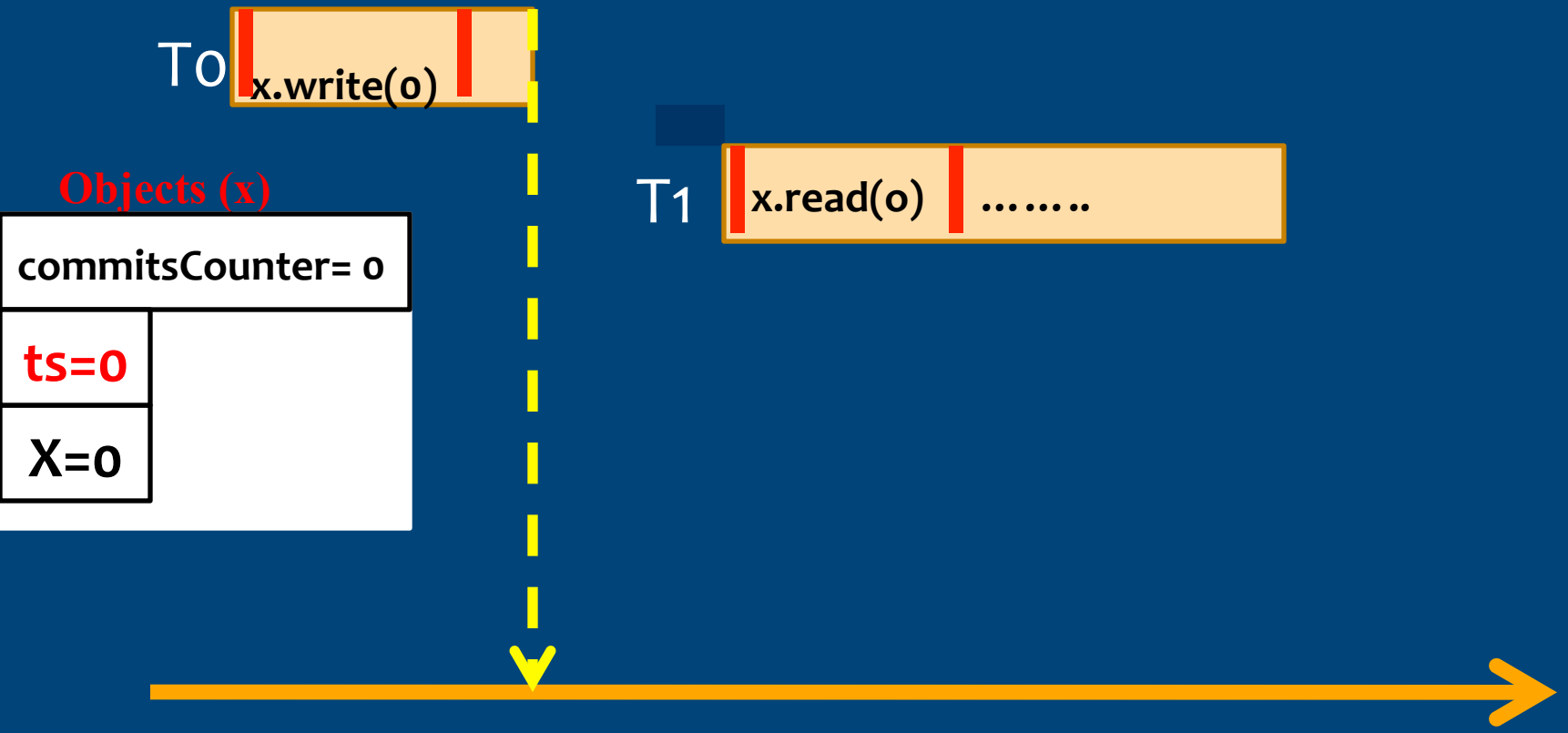
(i) It reduces the number of aborts since there is smaller chance for read-only transactions to abort update transactions.

(ii) It reduces space requirements, since a new version is saved once every K object updates, which reduces the total number of saved object versions by a factor of K

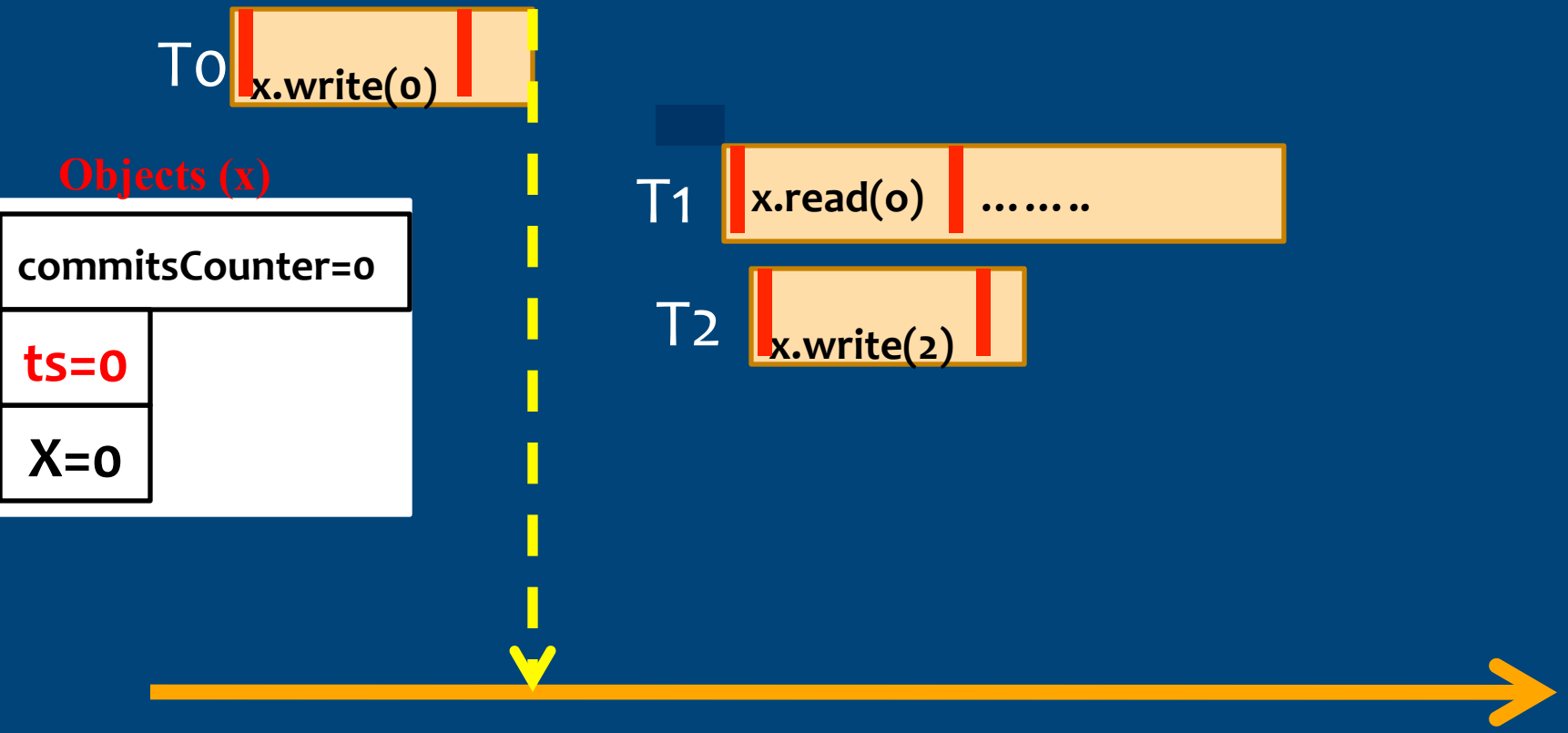
Approximately Opaque TM for Read-only Transactions



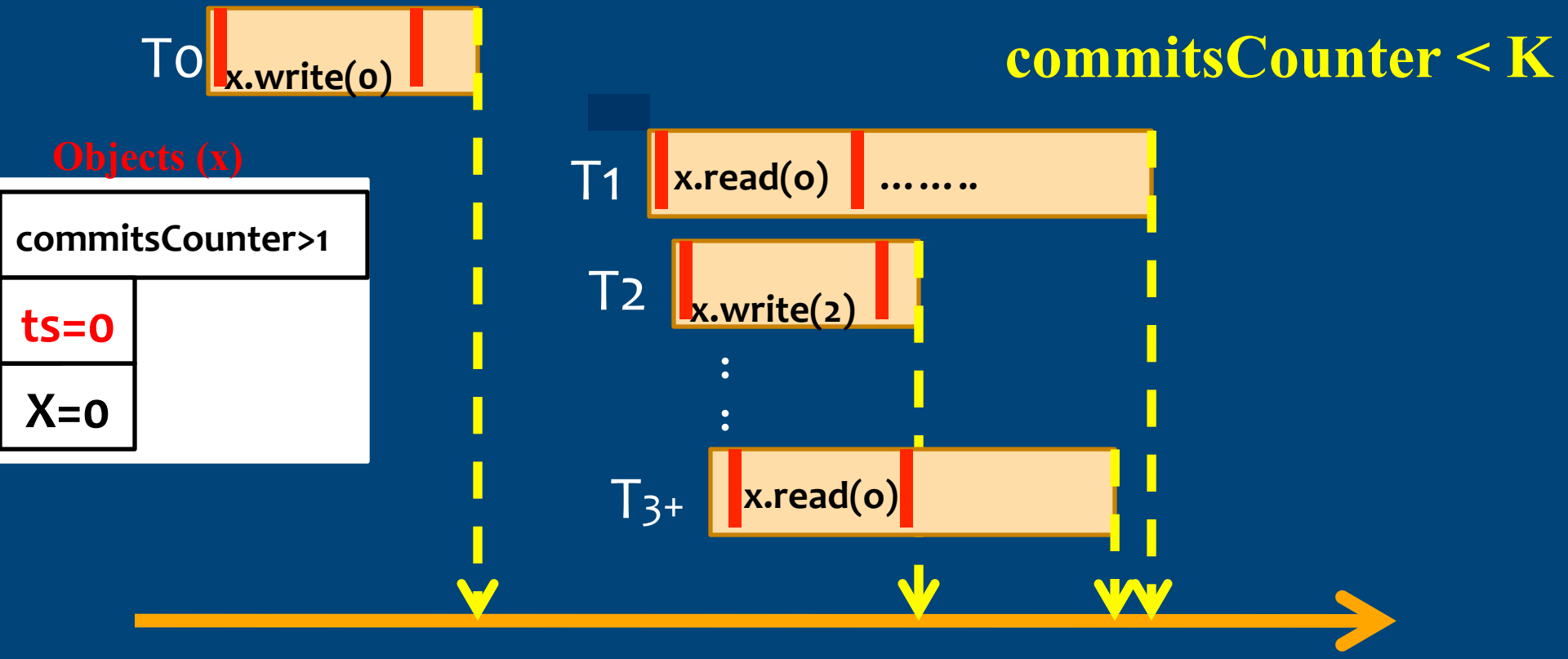
Approximately Opaque TM for Read-only Transactions



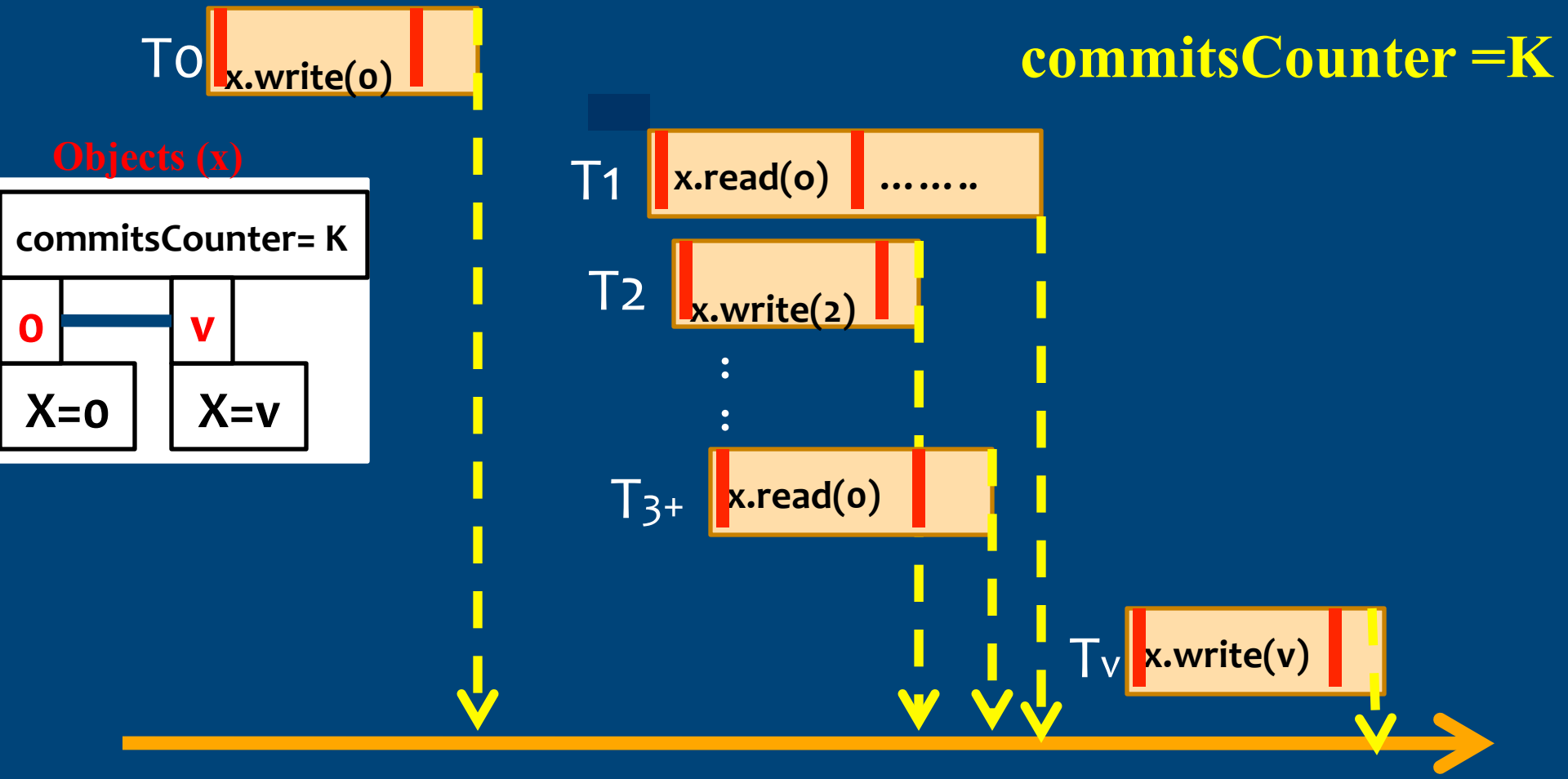
Approximately Opaque TM for Read-only Transactions



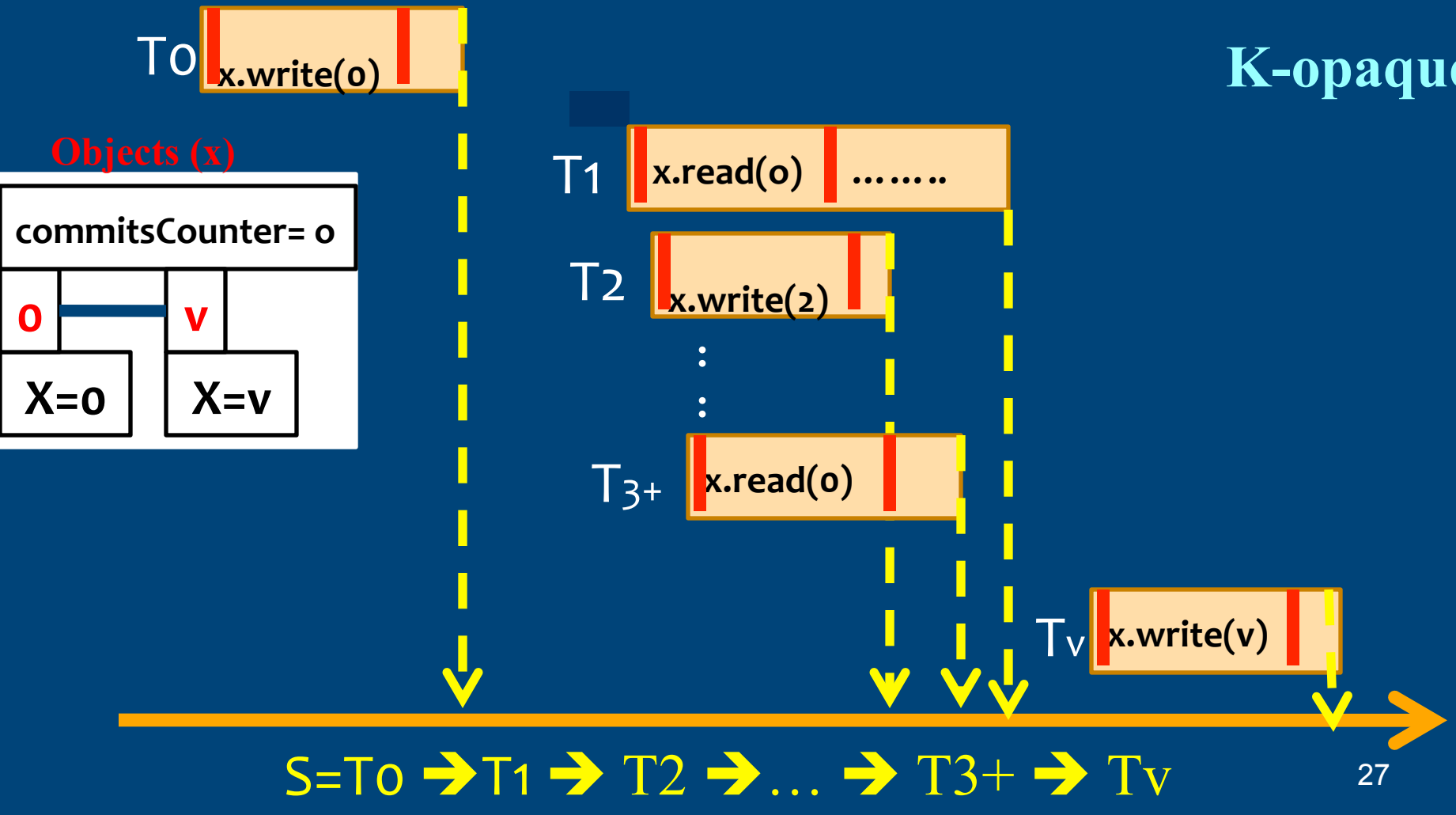
Approximately Opaque TM for Read-only Transactions



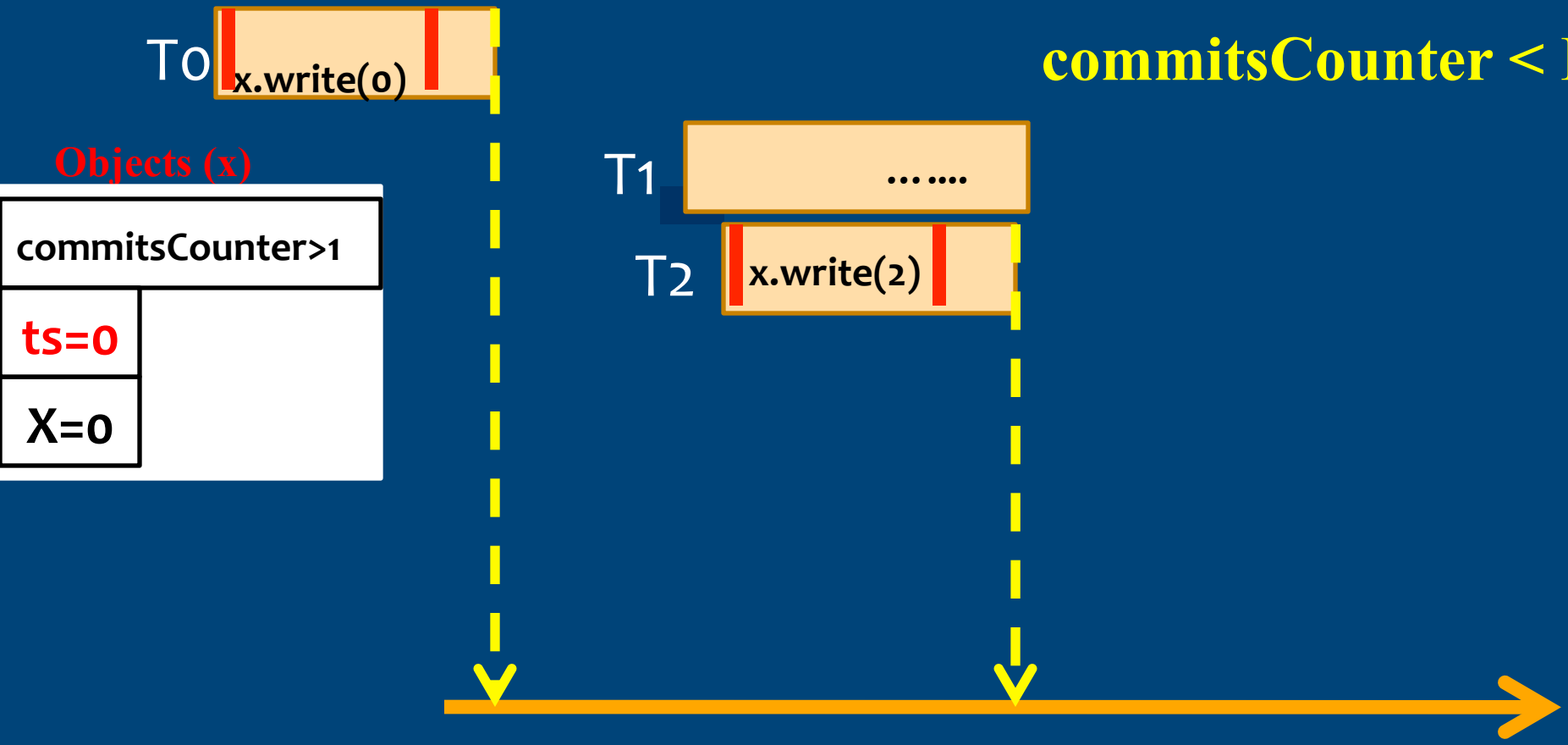
Approximately Opaque TM for Read-only Transactions



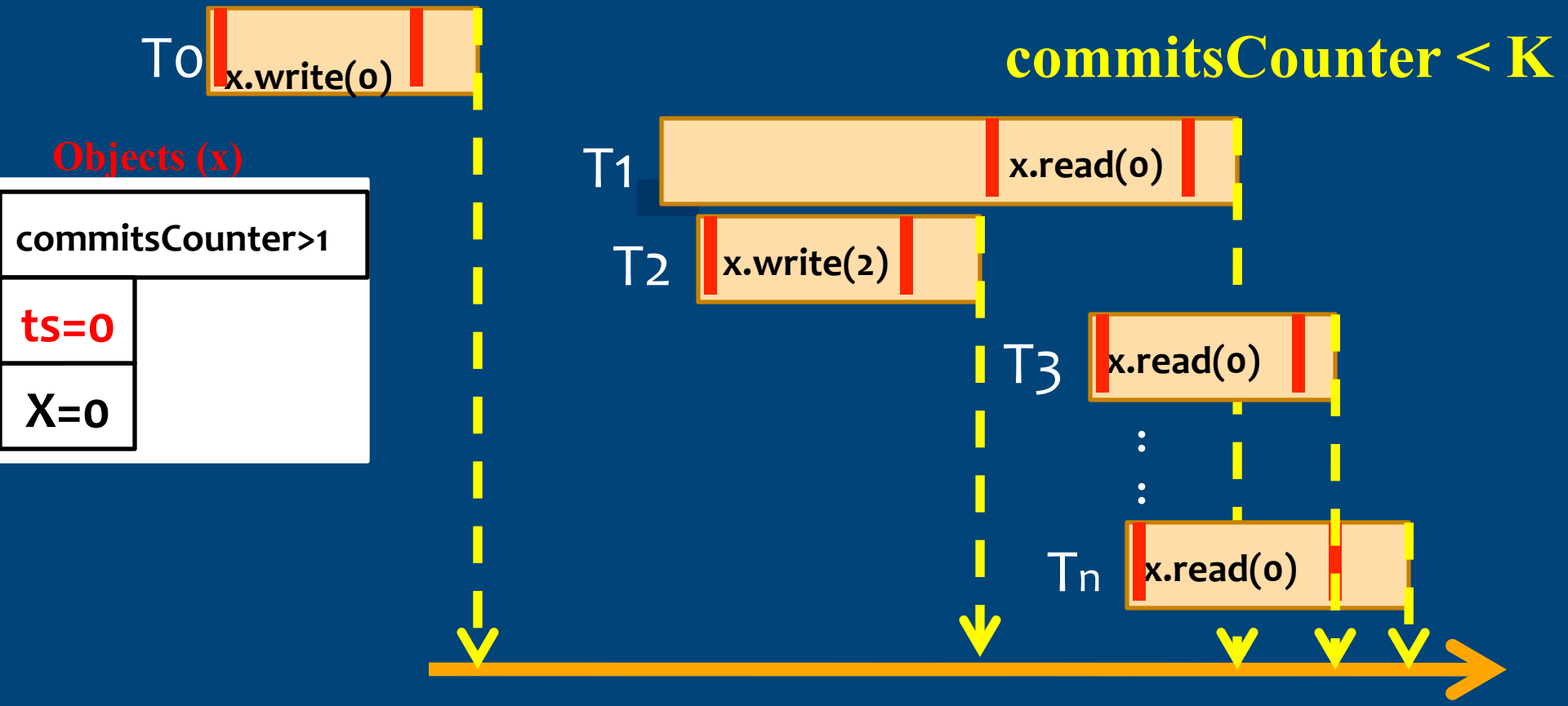
Approximately Opaque TM for Read/write Object



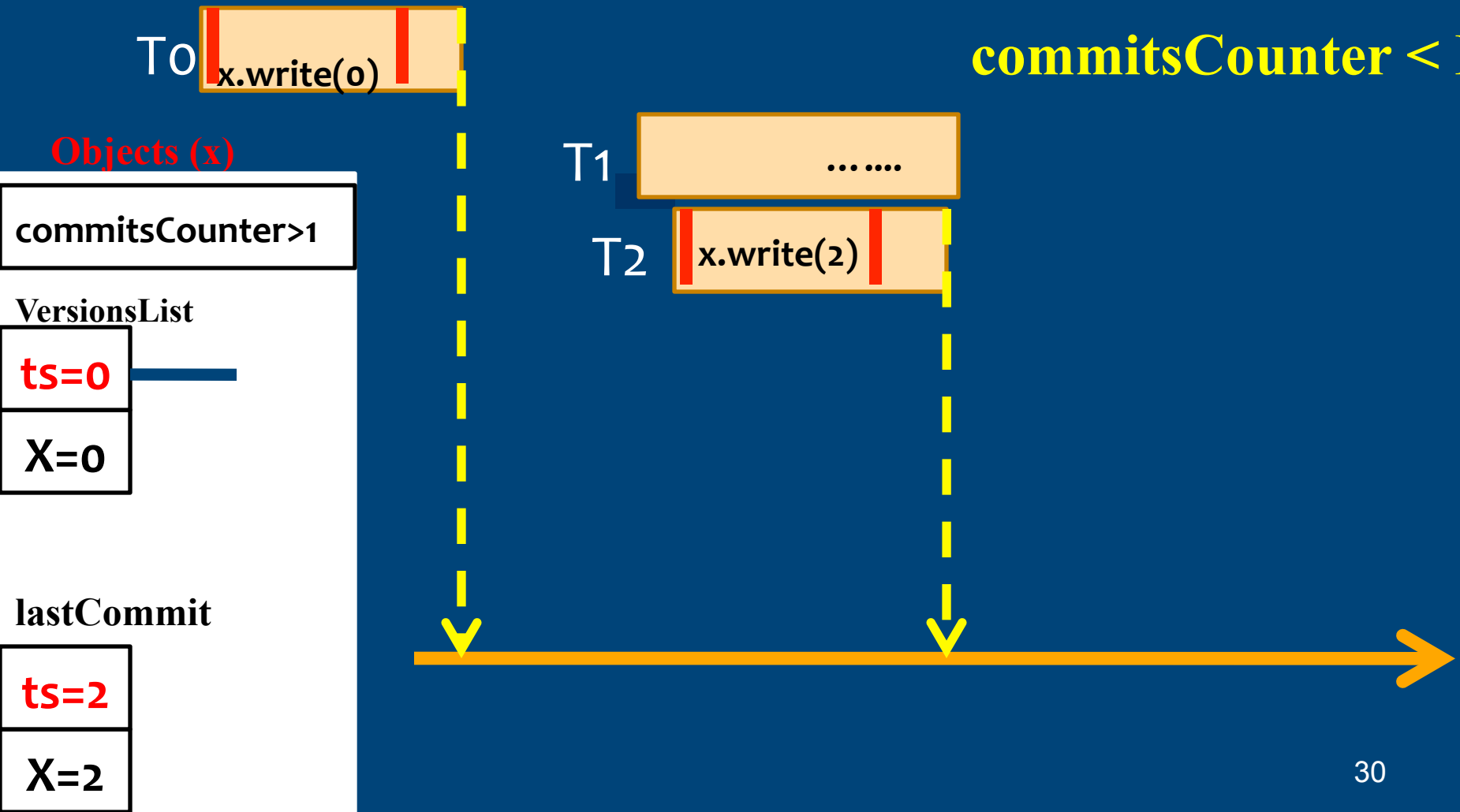
Another Issue



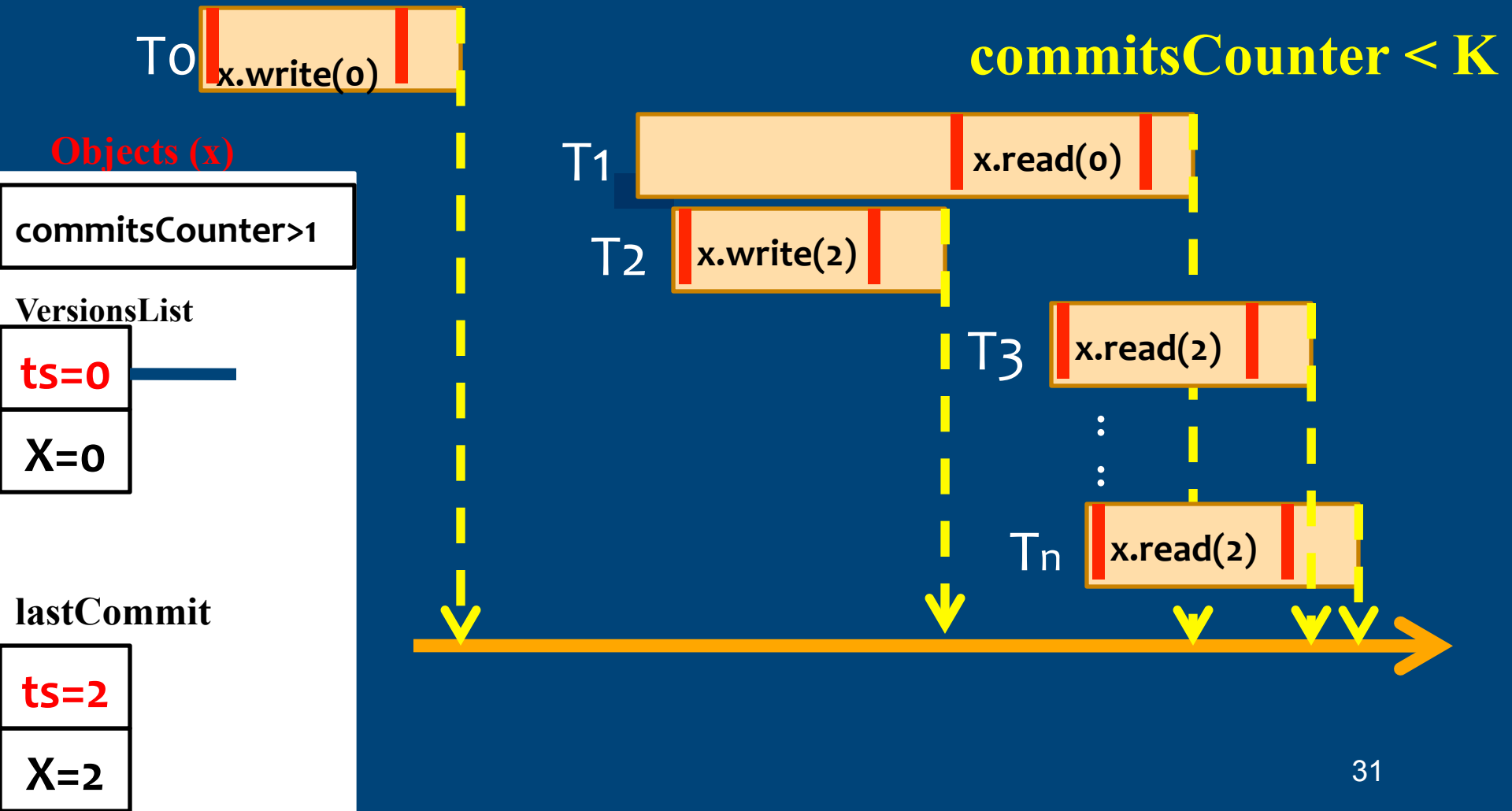
Another Issue



Solution


T0
x.write(0)
commitsCounter < K
T1
……
T2
x.write(2)
Objects (x)
commitsCounter>1
VersionsList
ts=0
X=0
lastCommit
ts=2
X=2

Solution


T0
x.write(0)
commitsCounter < K
Objects (x)
commitsCounter>1
VersionsList
ts=0
X=0
lastCommit
ts=2
X=2
T1
x.read(0)
T2
x.write(2)
T3
x.read(2)
Tn
x.read(2)

Outline

- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- Transactional Memory and Consistency
- Approximately Opaque TM for Read-only Transactions
- **Results**
- Garbage Collector
- Conclusion

Results

Micro-benchmarks : (tinySTM-1.0.5)

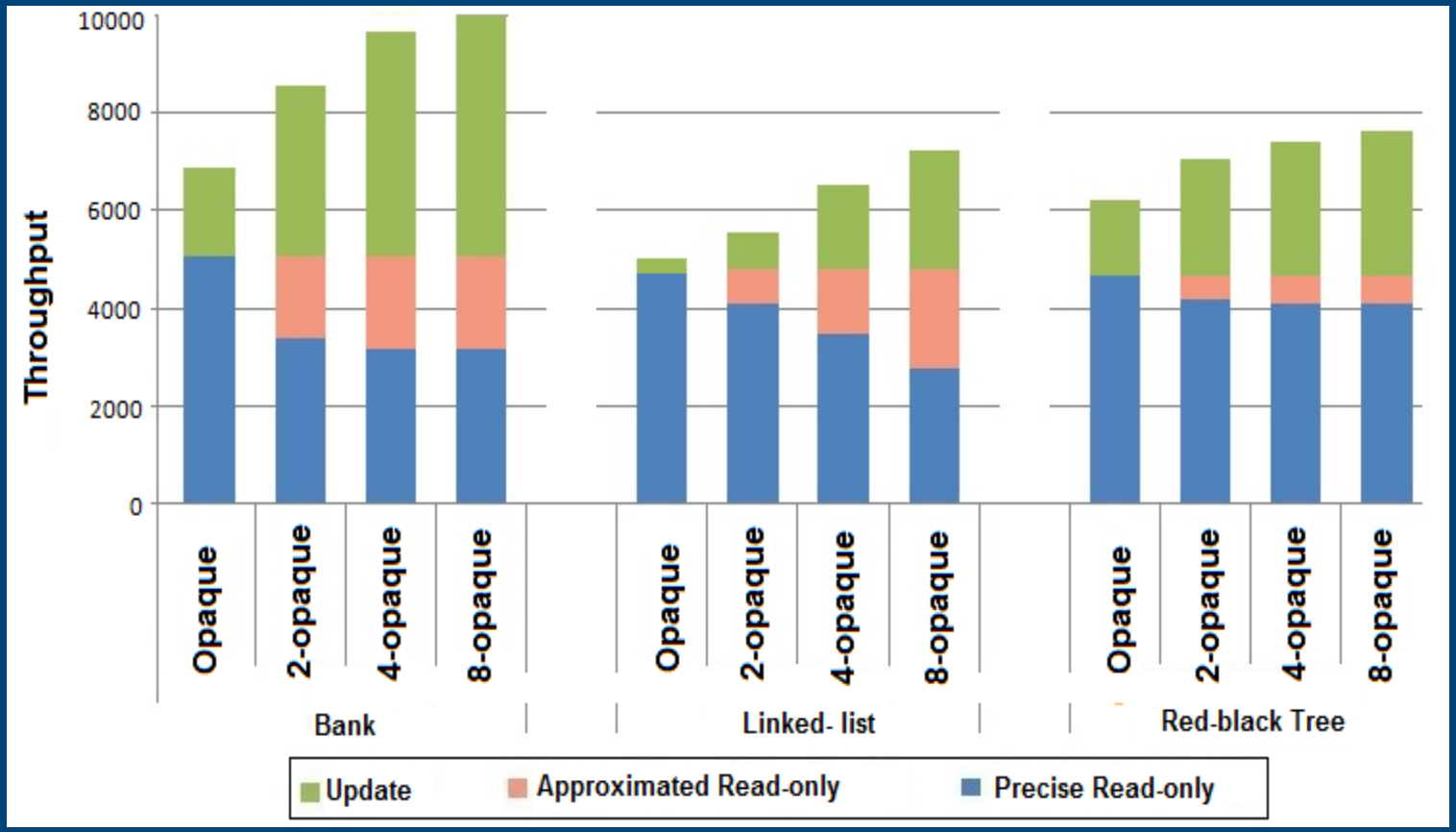
(a) Linked-list

(b) Read-black Tree

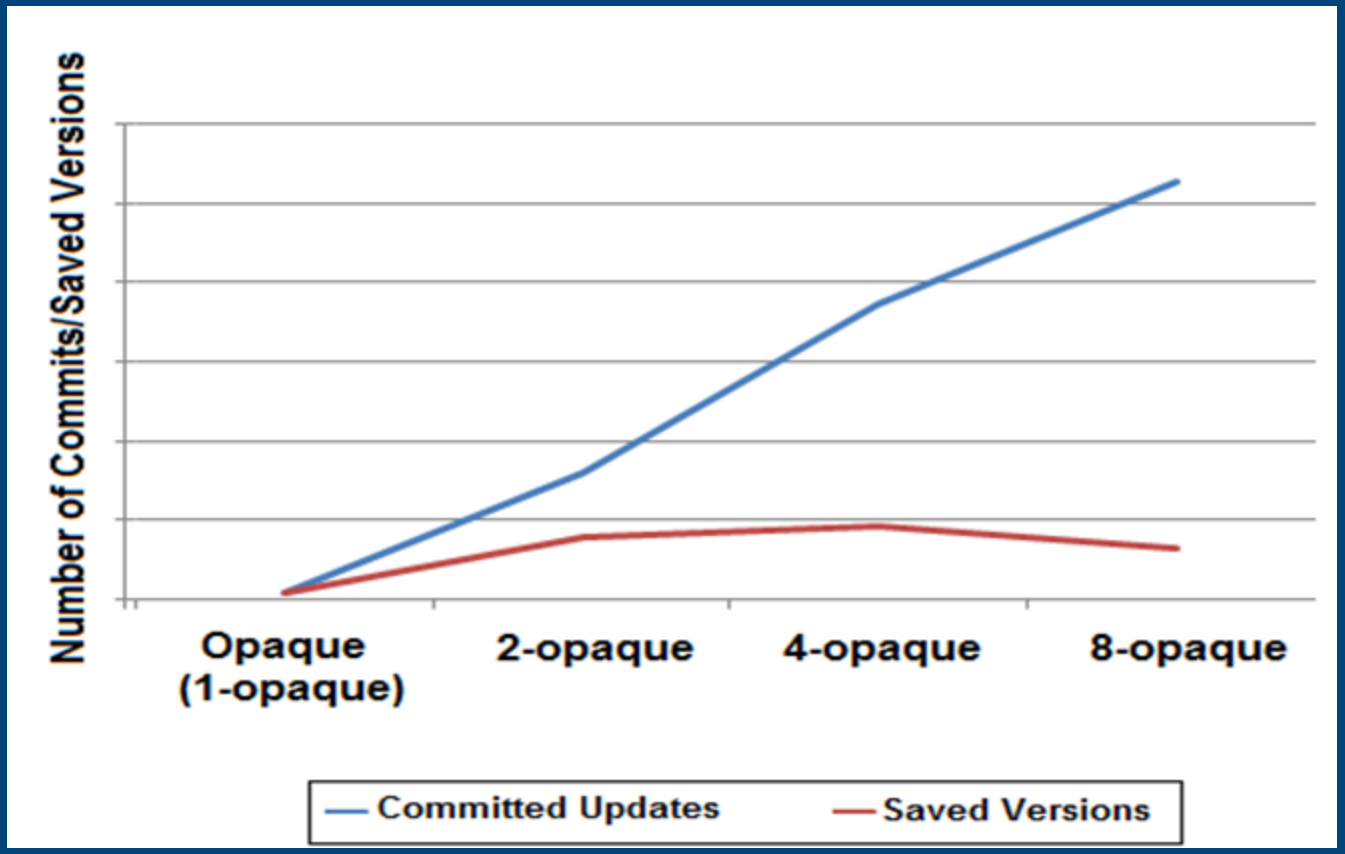
(c) Bank

- We run the experiments on a machine with dual Intel(R) Xeon(R) CPU E5-2630 (6 cores total) clocked at 2.30 GHz. Each run of the benchmark takes about 5500 milliseconds using 10 threads.

Results


Throughput
10000
8000
6000
4000
2000
0
Opaque
2-opaque
4-opaque
8-opaque
Bank
Linked- list
Red-black Tree
Update
Approximated Read-only
Precise Read-only

Results


Number of Commits/Saved Versions
Opaque (1-opaque)
2-opaque
4-opaque
8-opaque
Committed Updates
Saved Versions

Outline

- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- Transactional Memory and Consistency
- Approximately Opaque TM for Read-only Transactions
- Results
- **Garbage Collector**
- Conclusion

Garbage Collector

minT

LiveT
9
11
12
15
16

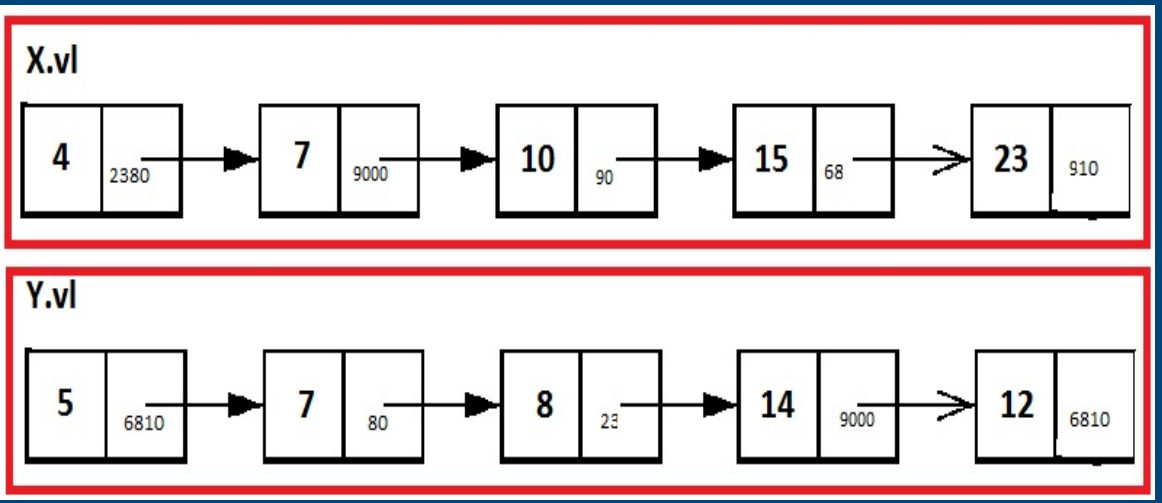
T9

T11

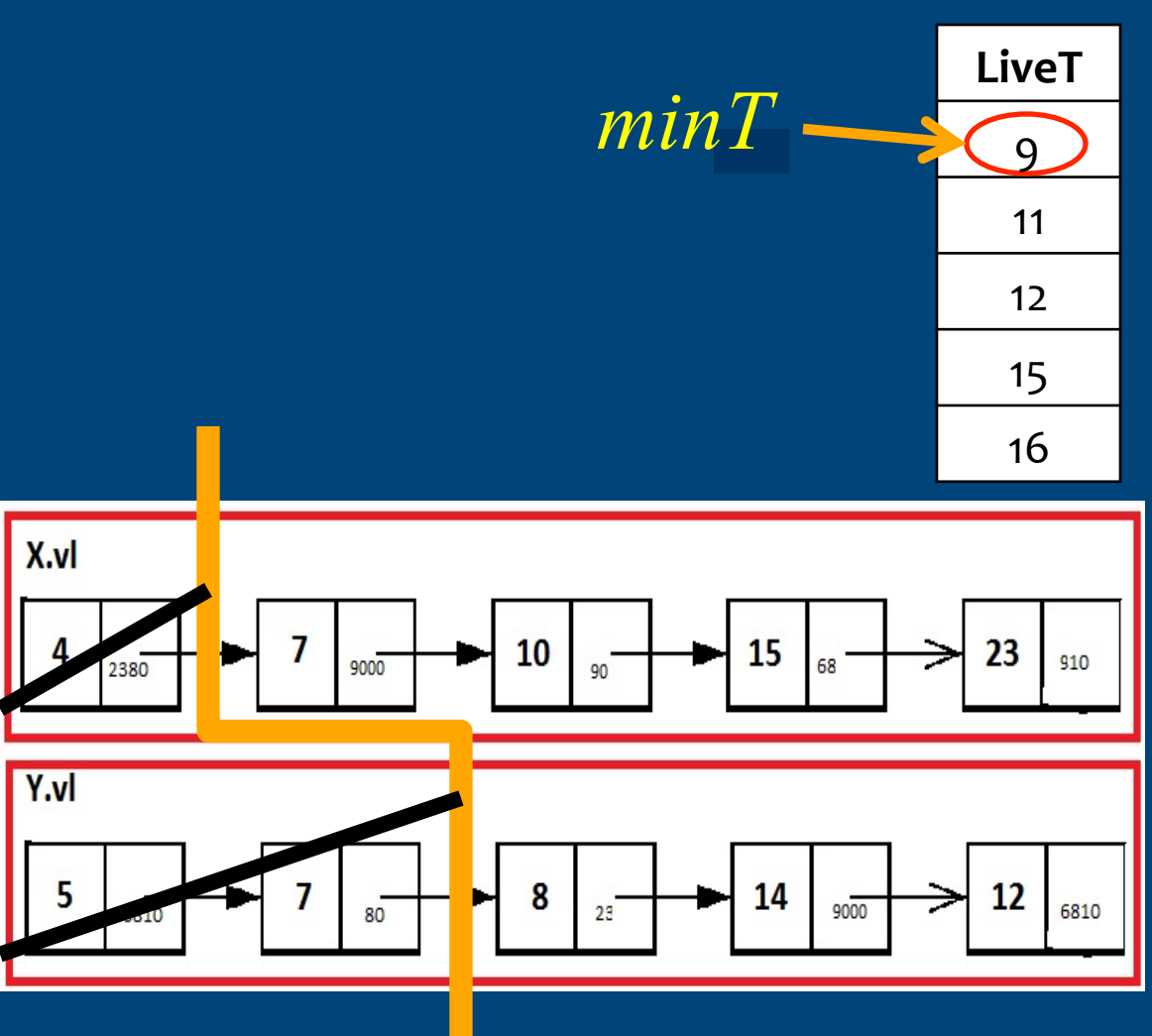
T12

T15

T16



Approximately Opaque TM for Queue Object



Outline

- Challenges of Distributed Computing and Performance
- Objectives and Motivation
- Transactional Memory and Consistency
- Approximately Opaque TM for Read-only Transactions
- Results
- Garbage Collector
- **Conclusion**

Conclusion

- **Improve the performance and the space complexity by relaxing the consistency**

- **We extend this work and apply the K-opacity concept on update transactions and different data structures.**

- **For future direction we can use different kind of relaxation such time-based or value-based relaxation.**

Conclusion

Snapshot Isolation allows older reads, but it still ensures that each transaction sees a consistent snapshot. However, SI does not preserve serializability. While in K-opacity transactions may see a K-consistent snapshot but it preserves a relaxed (Approximate) serializability.

Relaxing Opacity in Pessimistic Transactional Memory allows for update transaction B to update any object that belongs to the read set of a live transaction A, if the live transaction A would not access that object any more.

Thank you!

Questions??