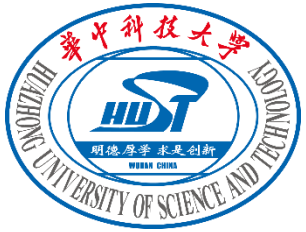
CATS: A Computation-Aware Transaction Processing System with Proactive Unlocking

Bolun Zhu, Yu Hua, Ziyin Long, Xue Liu

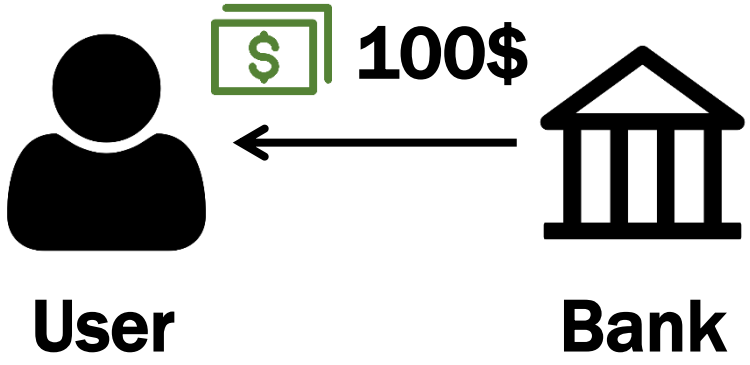


IWQoS 2023

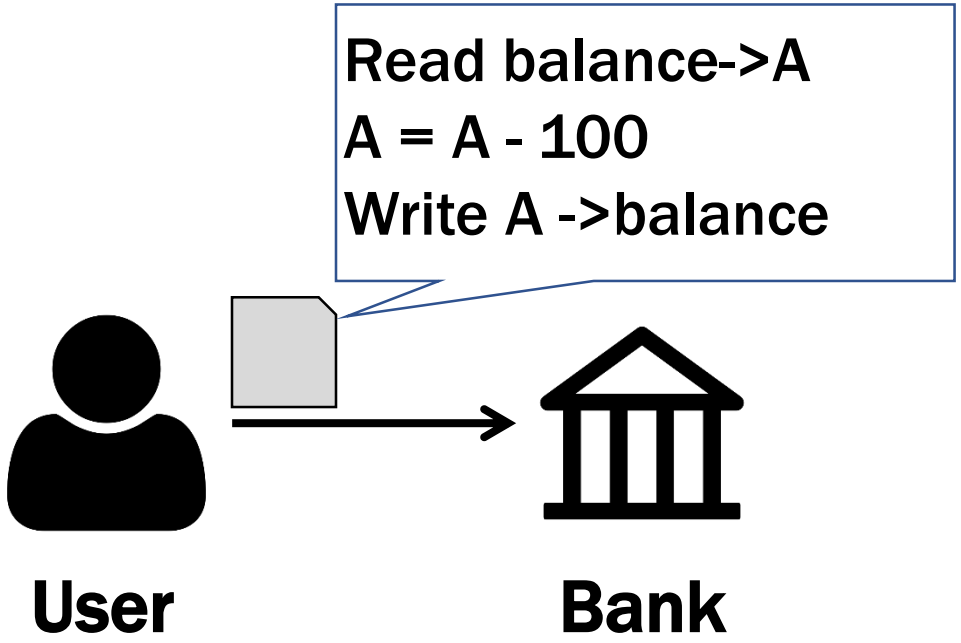
Artifact available at: https://github.com/BolunZhu/CATS

Concurrency Control

- **Concurrency control**: a basic problem for concurrent apps.



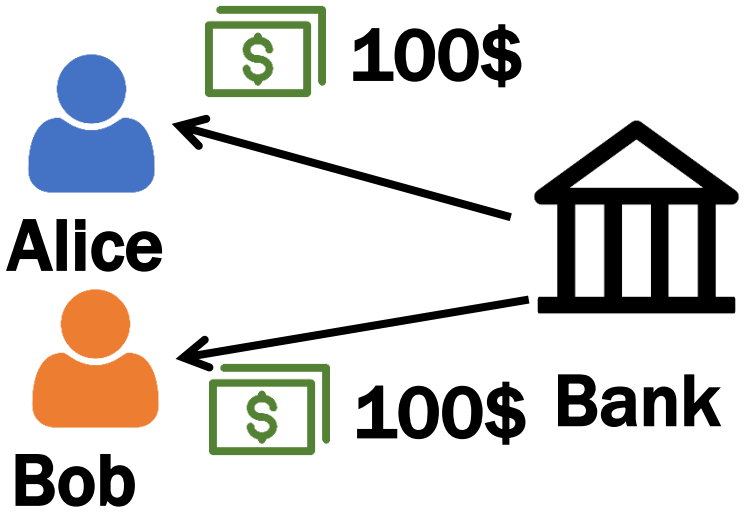
User's view



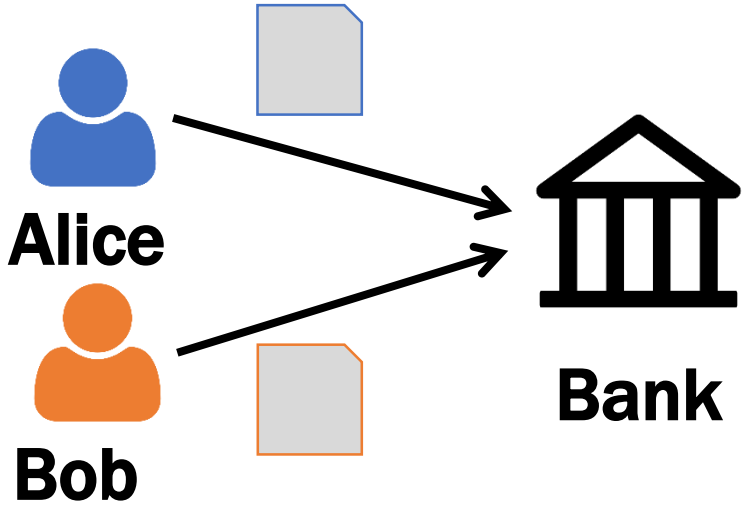
Server's view

Concurrency Control

- **Concurrency control**: a basic problem for concurrent apps.

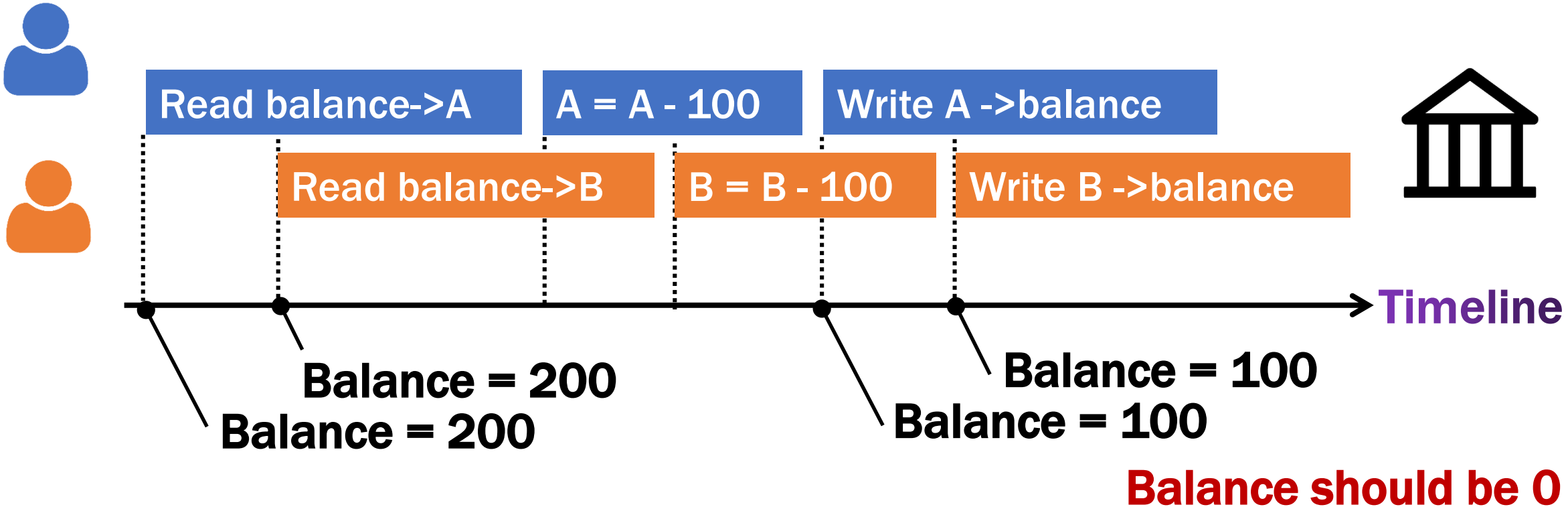


User's view



Server's view

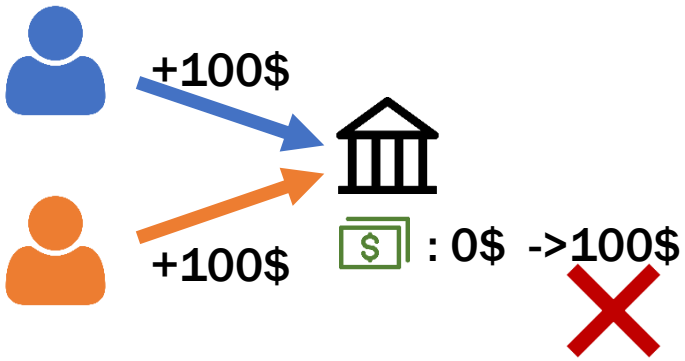
Concurrency Control

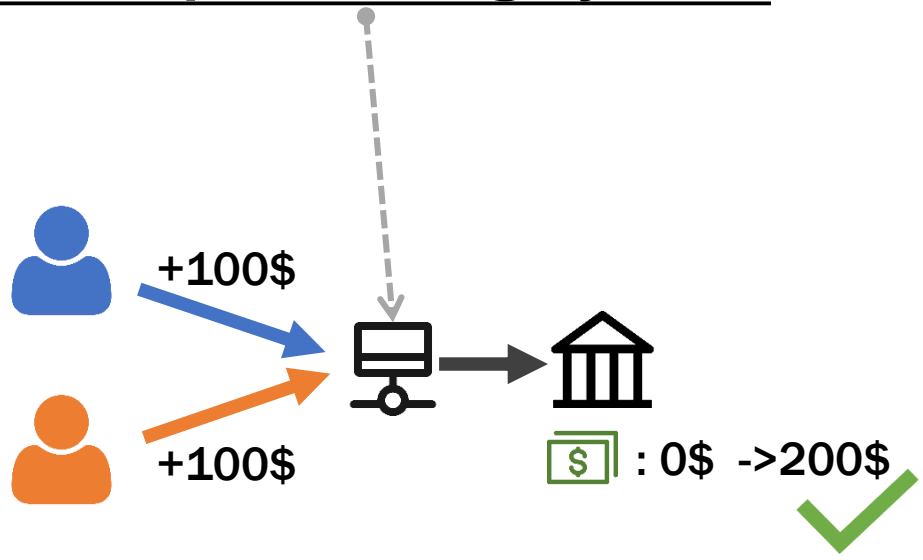


Concurrent operations should be ordered to avoid conflicts

Transaction System:

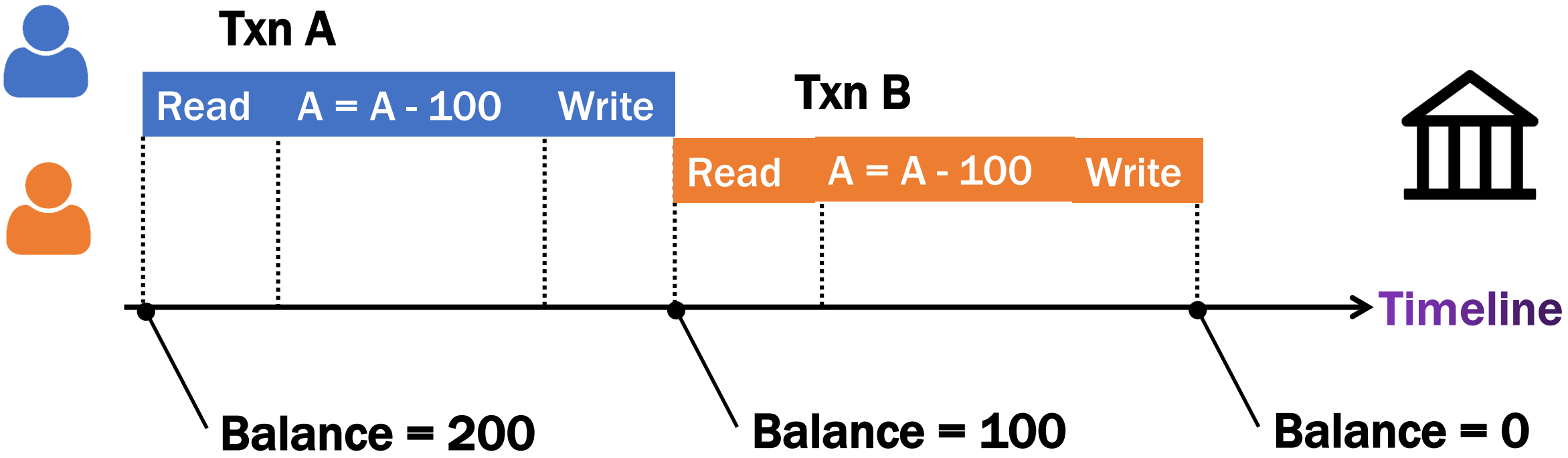
- Concurrency control is a basic problem
- Concurrent programming is hard and error-prone
- Concurrency control with transaction processing system:
  - High efficiency
  - Ease of use





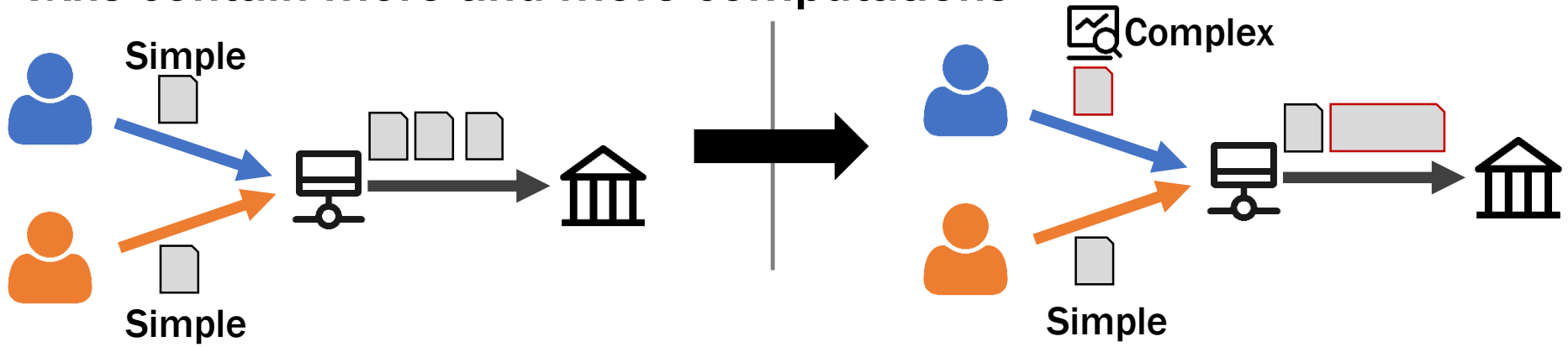
Transaction

- A transaction (txn) is
  a sequence of operations that is performed atomically



Limitations of Prior Systems

- Most prior systems are designed for simple txns:
  - Most ops are read and write ops
  - Computations are negligible
- Real-world apps become more and more complex
  - Txns contain more and more computations



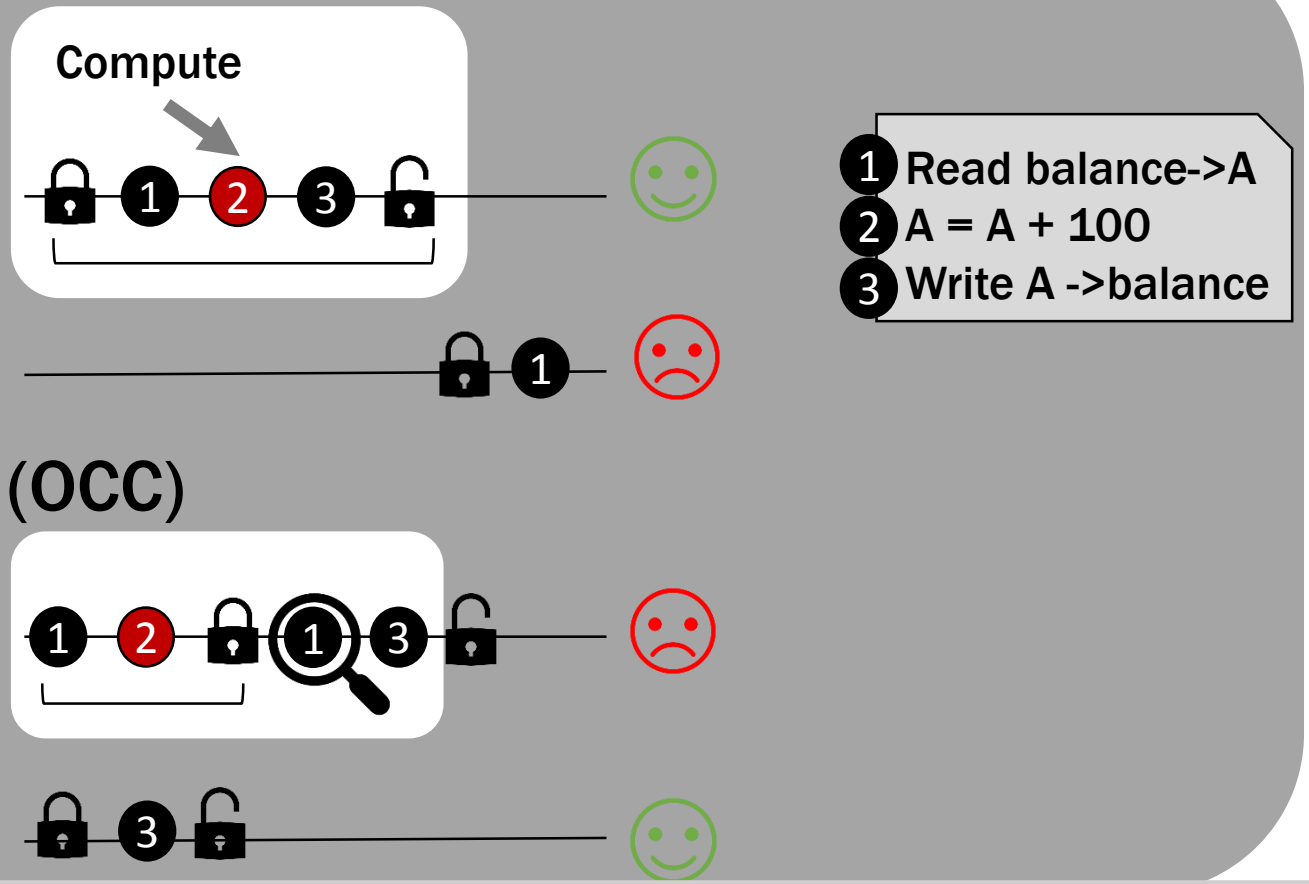
The concurrency control mechanism becomes inefficient

Limitations of Prior Systems

- Two phase locking (2PL)
  - Conflict causes blocking

Compute operations are critical for concurrency control

- C (OCC)
  - Conflict causes re-execution



Prior systems are unaware of computation operations

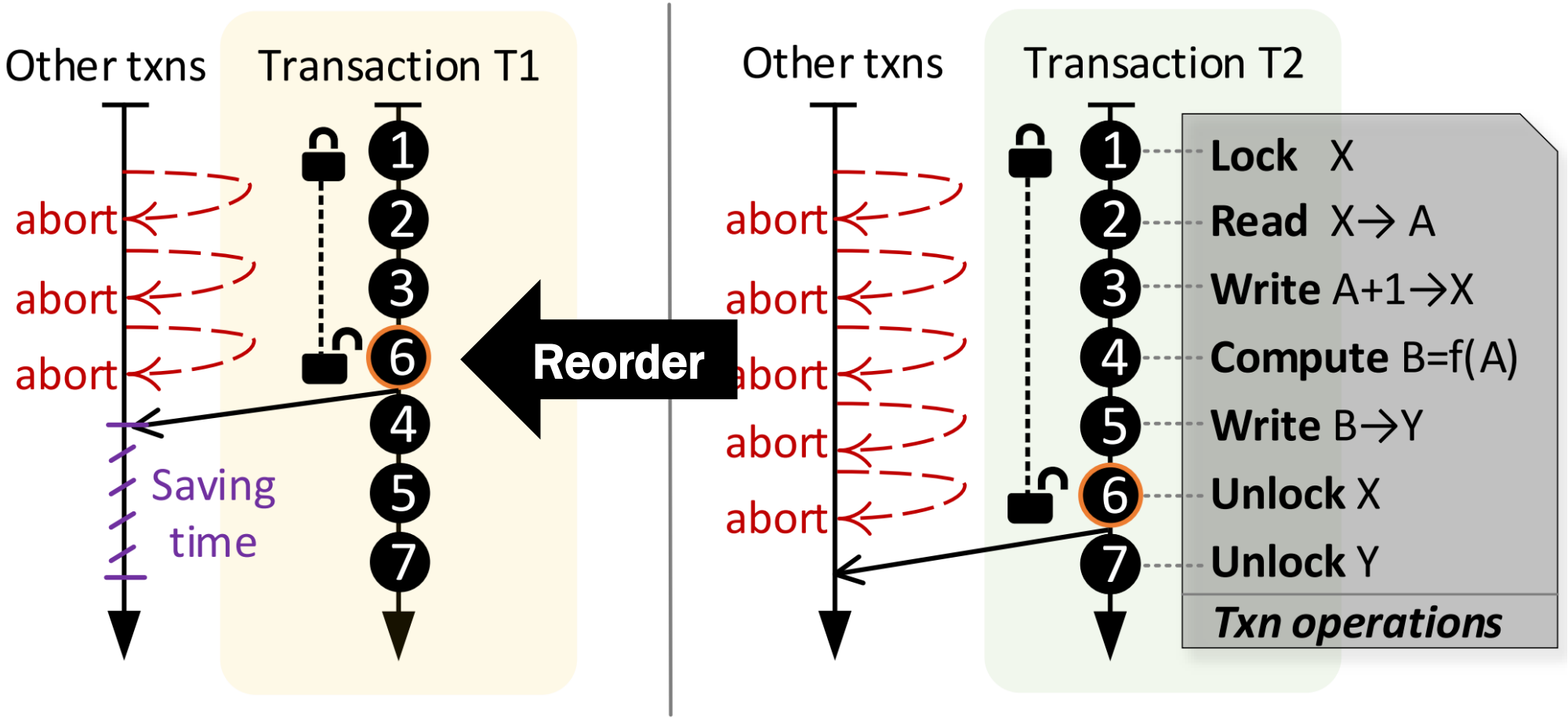
Become Computation-Aware

- Transaction processing system is a significant infrastructure in real-world applications.

- Prior systems are unaware of computation operations

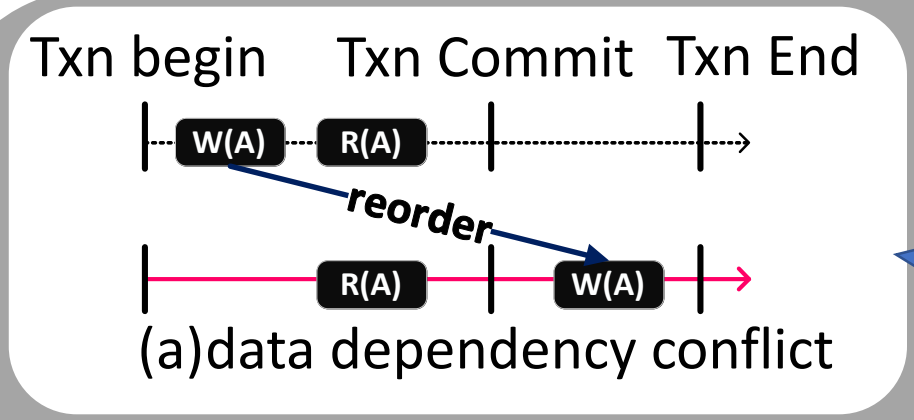
- In this work, we ask:
  - Can txn processing system become computation-aware?
  - How to do this?

Our Approach: Proactive Unlocking



Removing compute operations from the critical path

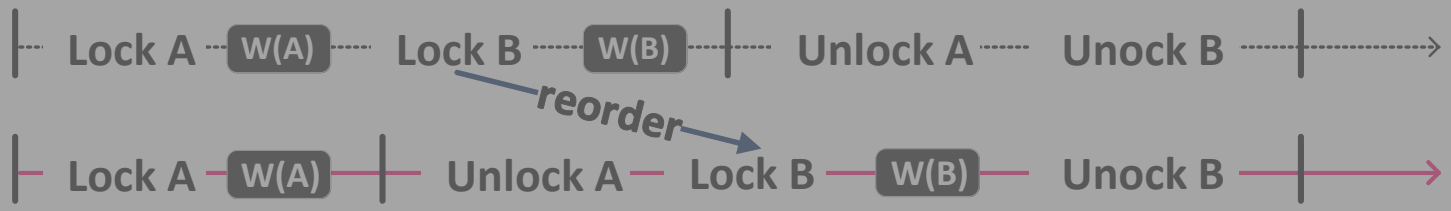
Challenge #1: Data Dependency Conflict



(a)data dependency conflict

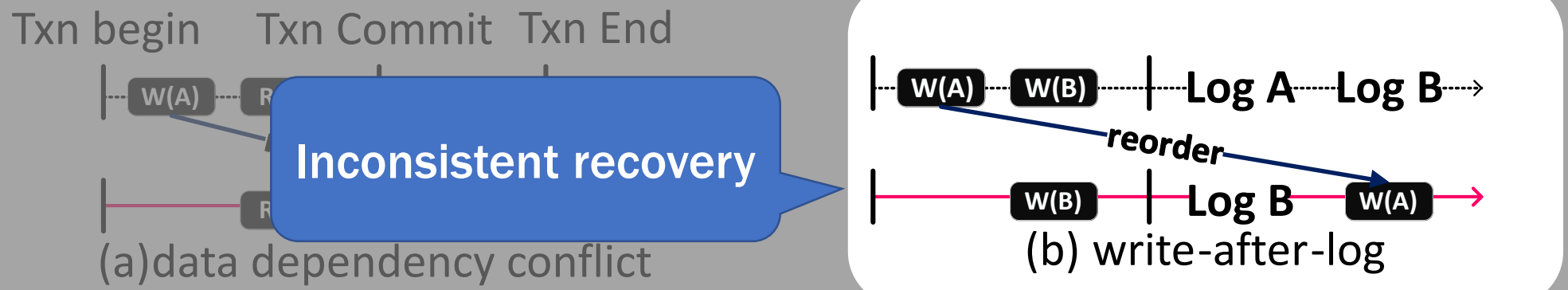
Incorrect execution results

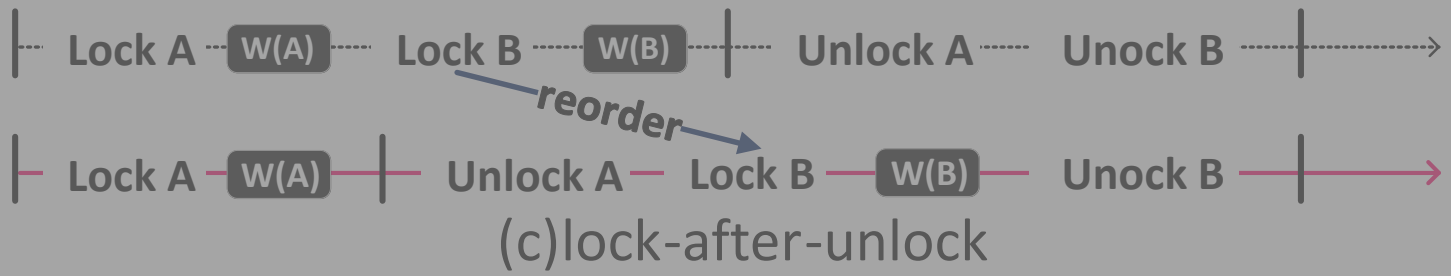
(b) write-after-log



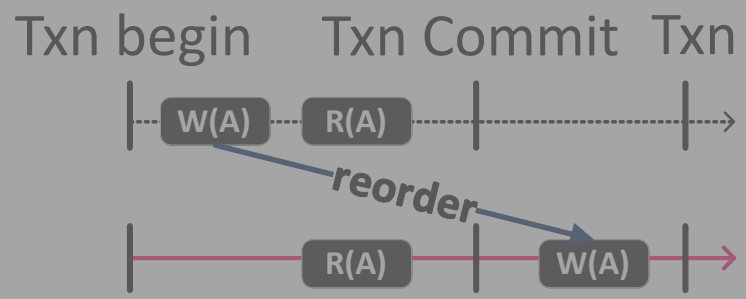
(c)lock-after-unlock

Challenge #2: Write-after-log

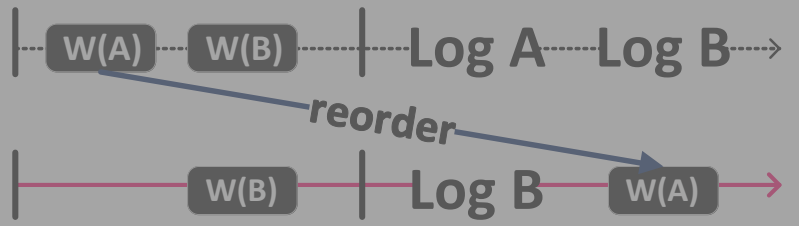




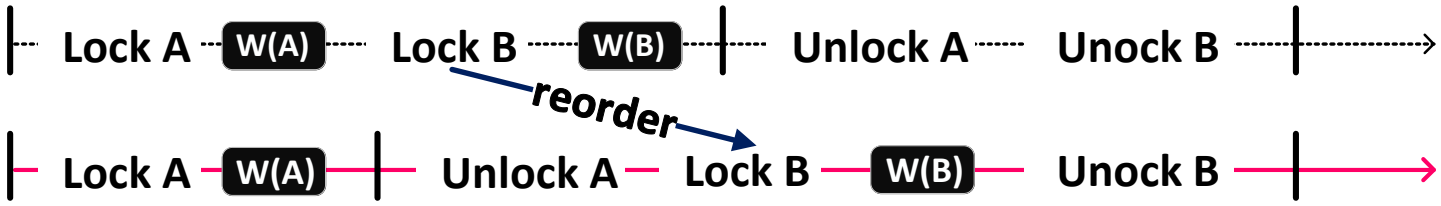
Challenge #3: Lock-after-unlock



(a)data dependency conflict



(b) write-after-log

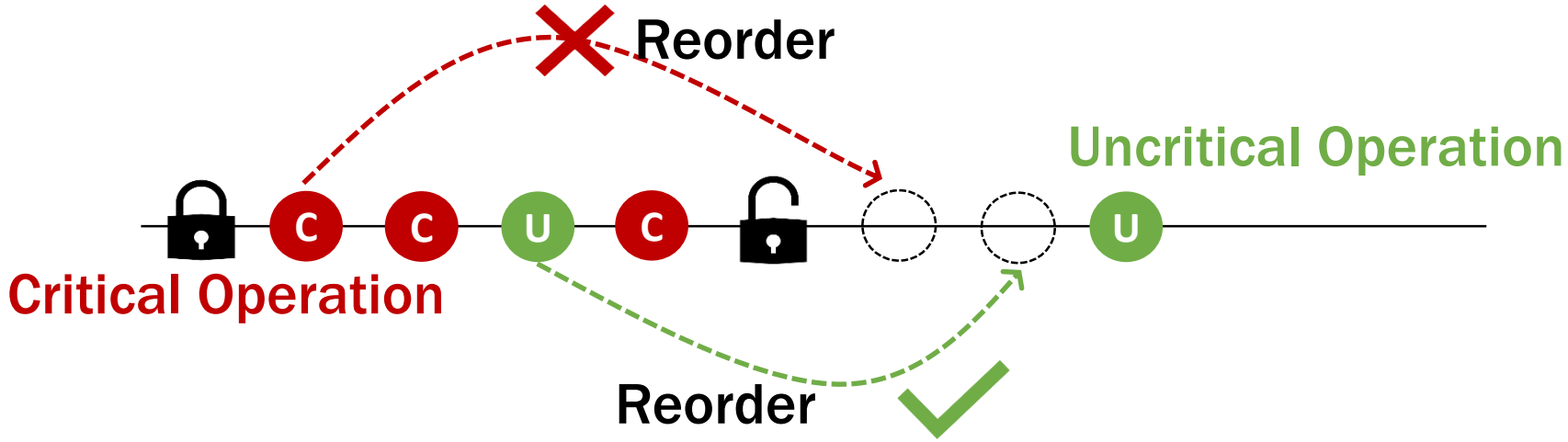


(c)lock-after-unlock

Violates the atomicity

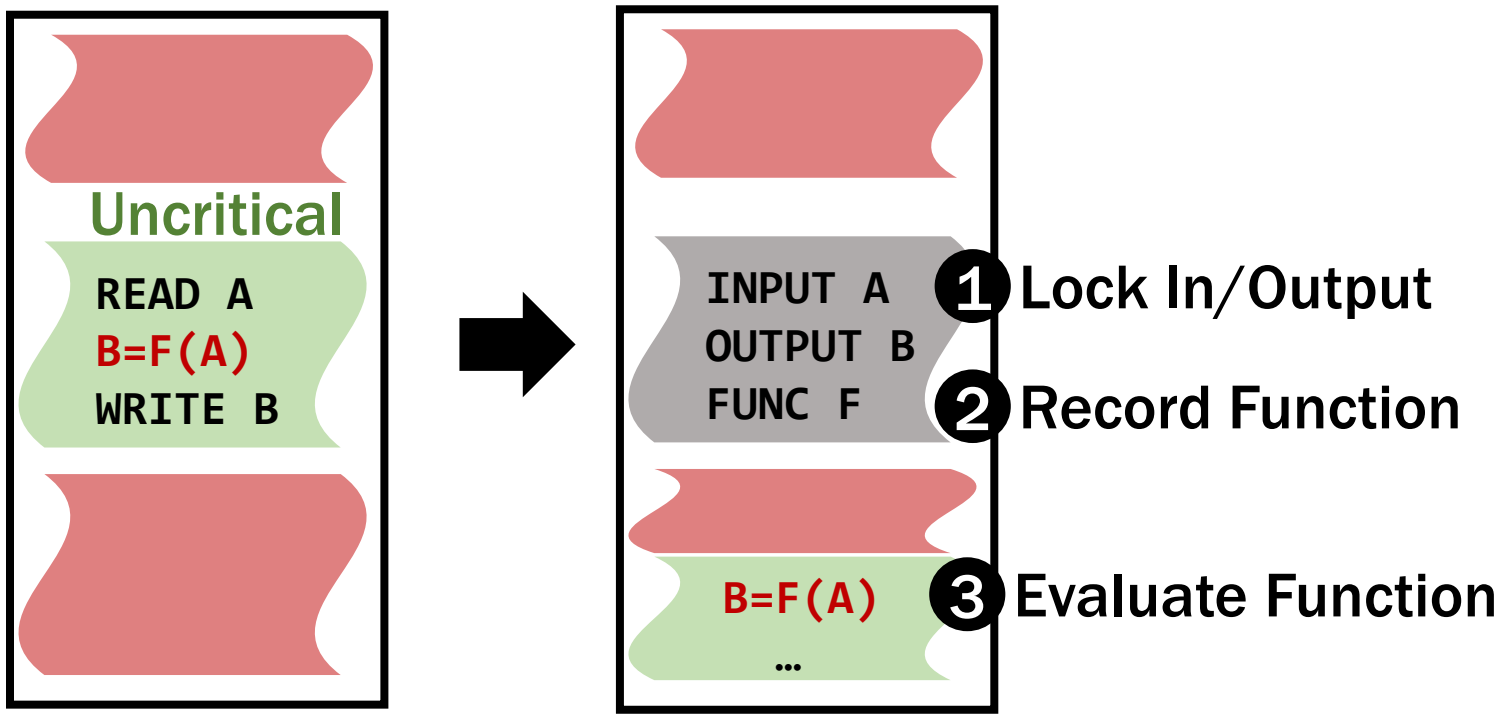
Design #1: Critical Operations

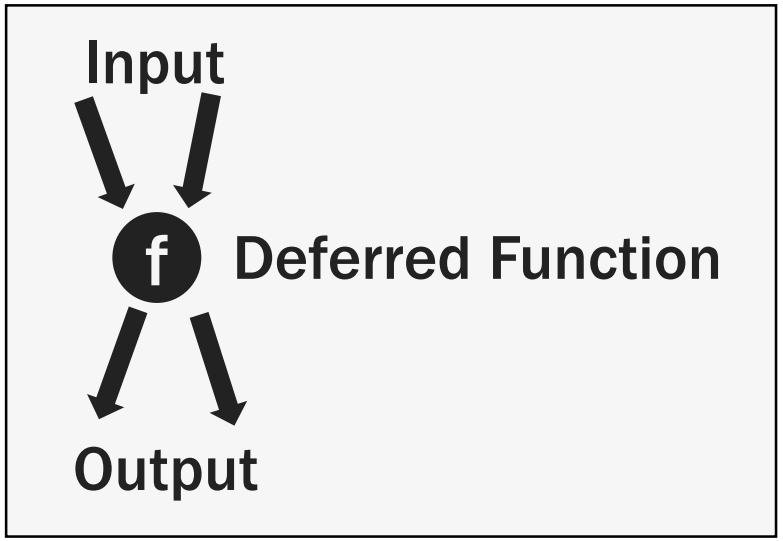
- We identify two categories of transaction operations
  - Critical Operation
  - Uncritical Operation



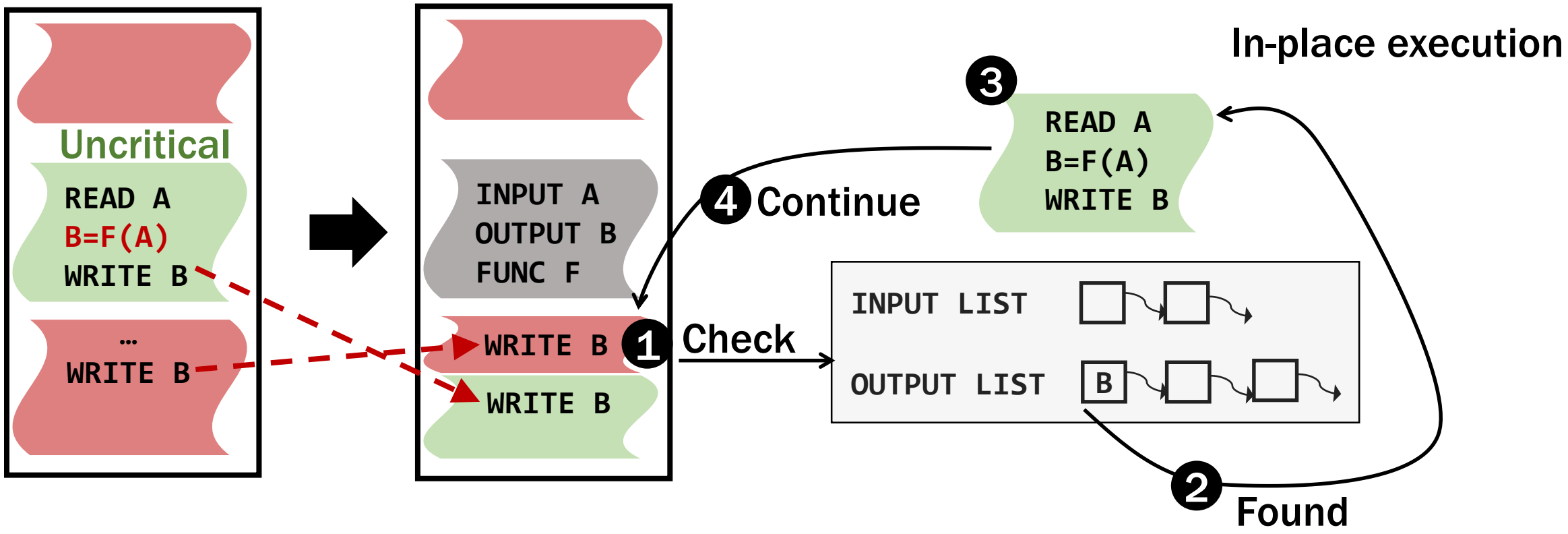
Uncritical operations can be removed from the critical path

Design #2: Deferred Execution





Design #3: Dependency Conflict Avoidance Mechanism



Methodology

Platform

CPU	Intel Xeon Gold 2*26 cores
DRAM	4x16GB DDR4
OS	Ubuntu 18.04 Linux kernel 4.15
Tool	RSTM (word-based transactional memory system)

Compared Target

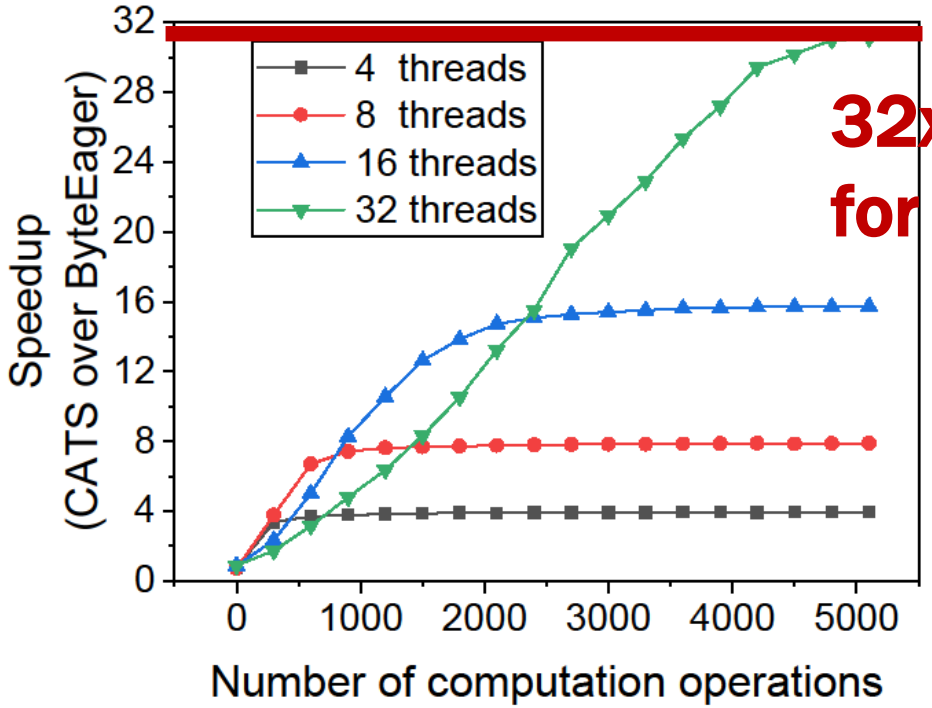
ByteEager (2PL-based system)
LLT (OCC-based system)

Test Programs

Compute operations
Hotspot position
Percentage of uncritical operations

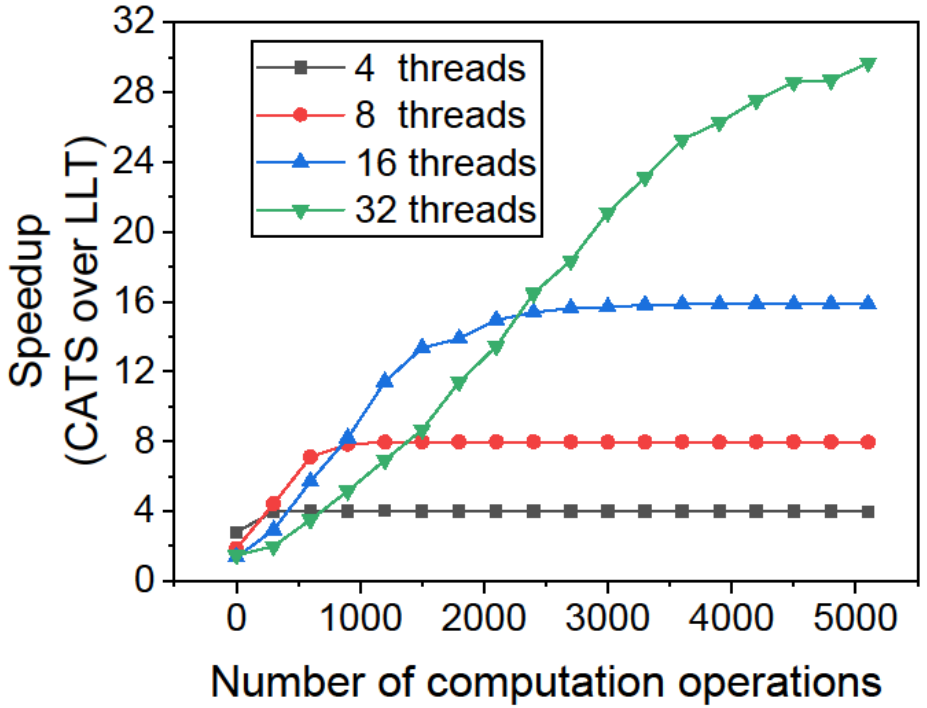
Evaluation

Speedup over 2PL-based system



32x speedup for 32 threads

Speedup over OCC-based system



CATS scales linearly as number of computation operations increases

Summary

- Transaction processing system is a significant infrastructure in real-world applications.
- Prior systems are unaware of computation operations
- We present *CATS* that can remove computation operations from the critical path of concurrency control
- CATS defines critical operations and uncritical operations
- CATS defers the execution of uncritical operations
- CATS maintains data dependencies of critical operations at runtime
- CATS is open-sourced at: https://github.com/BolunZhu/CATS

See our paper for more details