Mako: Speculative Distributed Transactions with Geo-Replication

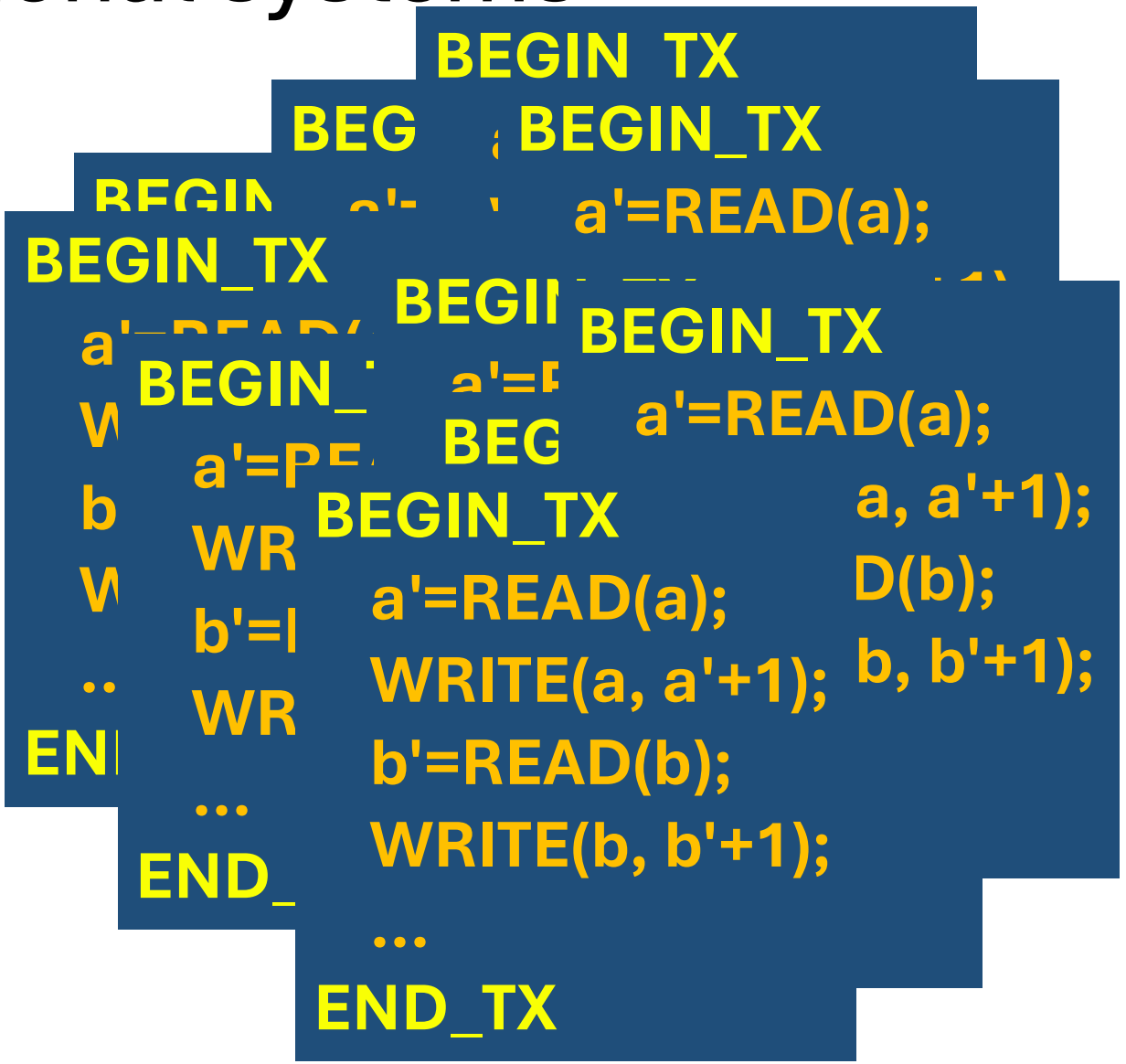
Weihai Shen[1], Yang Cui[2], Siddhartha Sen[3], Sebastian Angel[4], Shuai Mu[1]

[1]Stony Brook University, [2]Google, [3]Microsoft Research, [4]UPenn

Transactional systems

```
BEGIN_TX
   a'=READ(a);
   WRITE(a, a'+1);
   b'=READ(b);
   WRITE(b, b'+1);
   ...
END_TX
```

Transactional systems



Transactional systems

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```


```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

Transactional systems

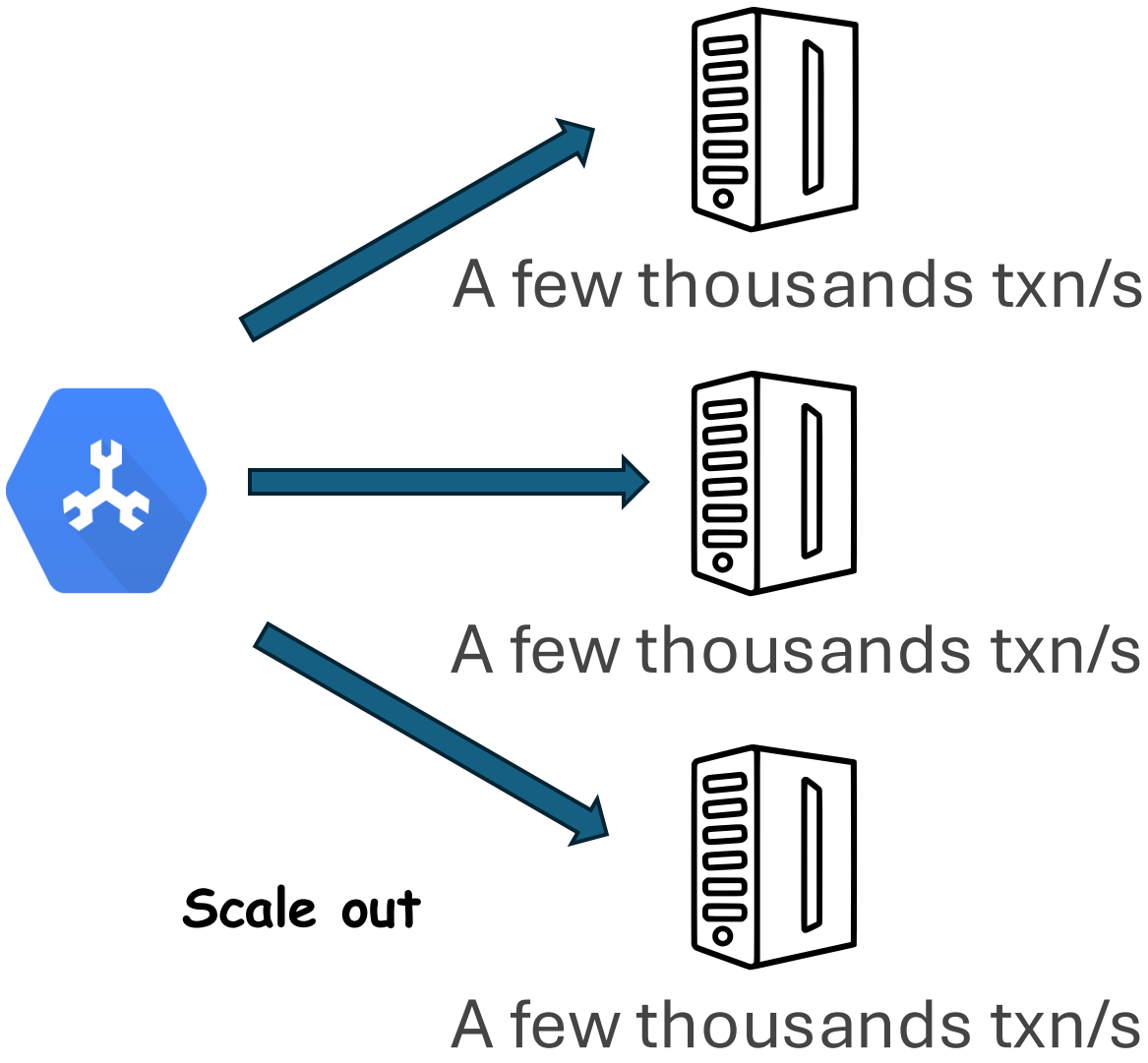
```
BEGIN_TX
a'=READ(a);
WRITE(a, a'+1);
b'=READ(b);
WRITE(b, b'+1);
...
END_TX
```

Transactions make concurrent programming much easier!

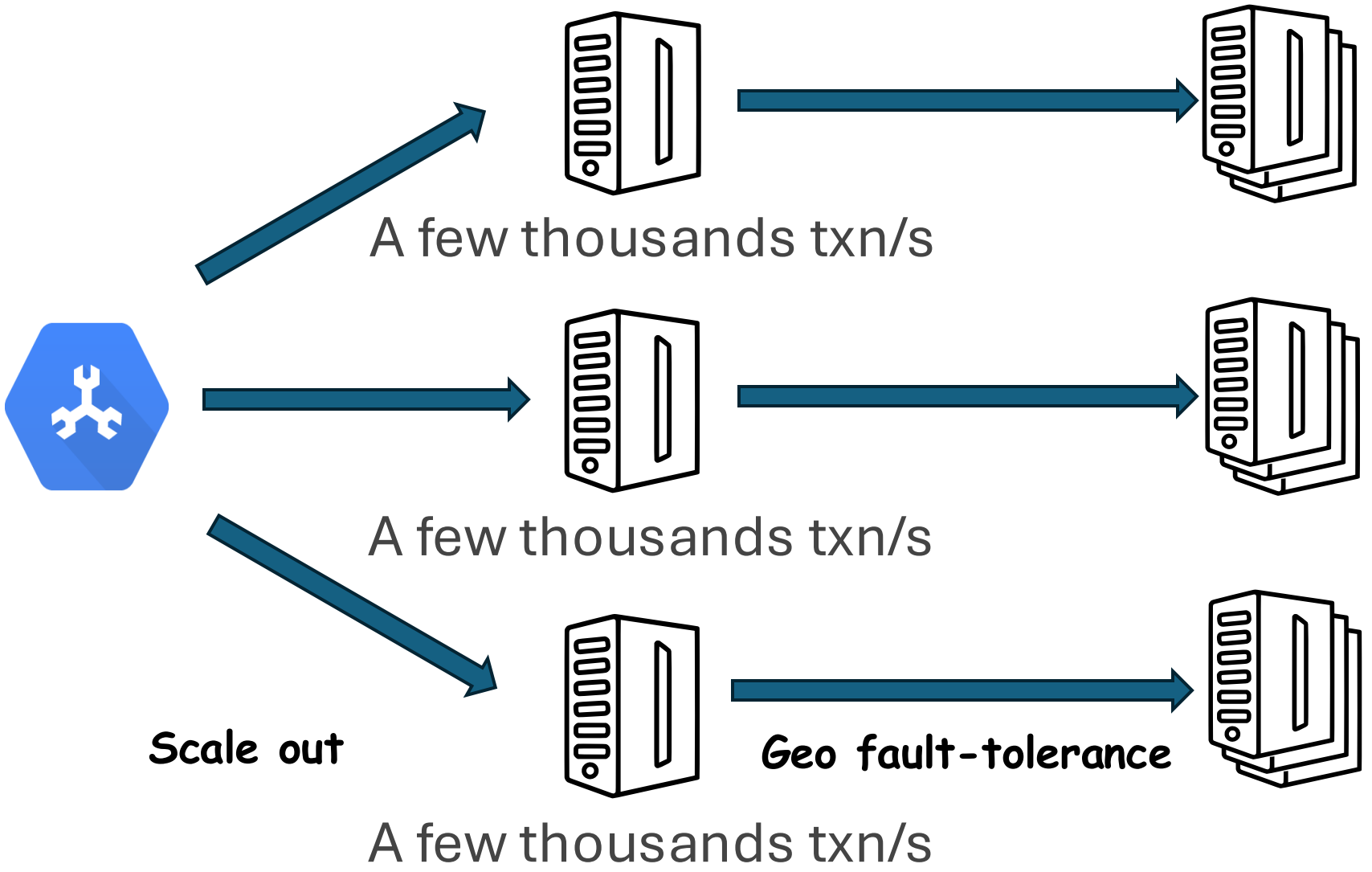
Distributed transactional systems

A few thousands txn/s

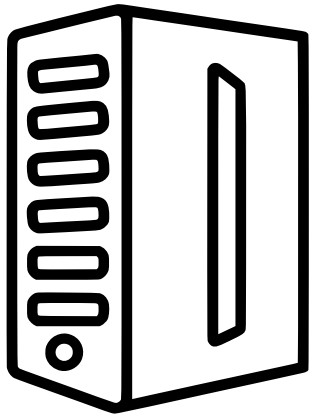
Distributed transactional systems



Distributed transactional systems



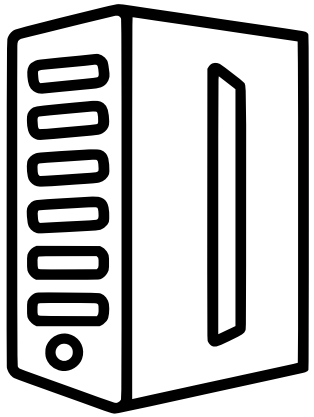
Single-server transactional systems



A few millions txn/s

No networking overhead!

Single-server transactional systems



A few millions txn/s

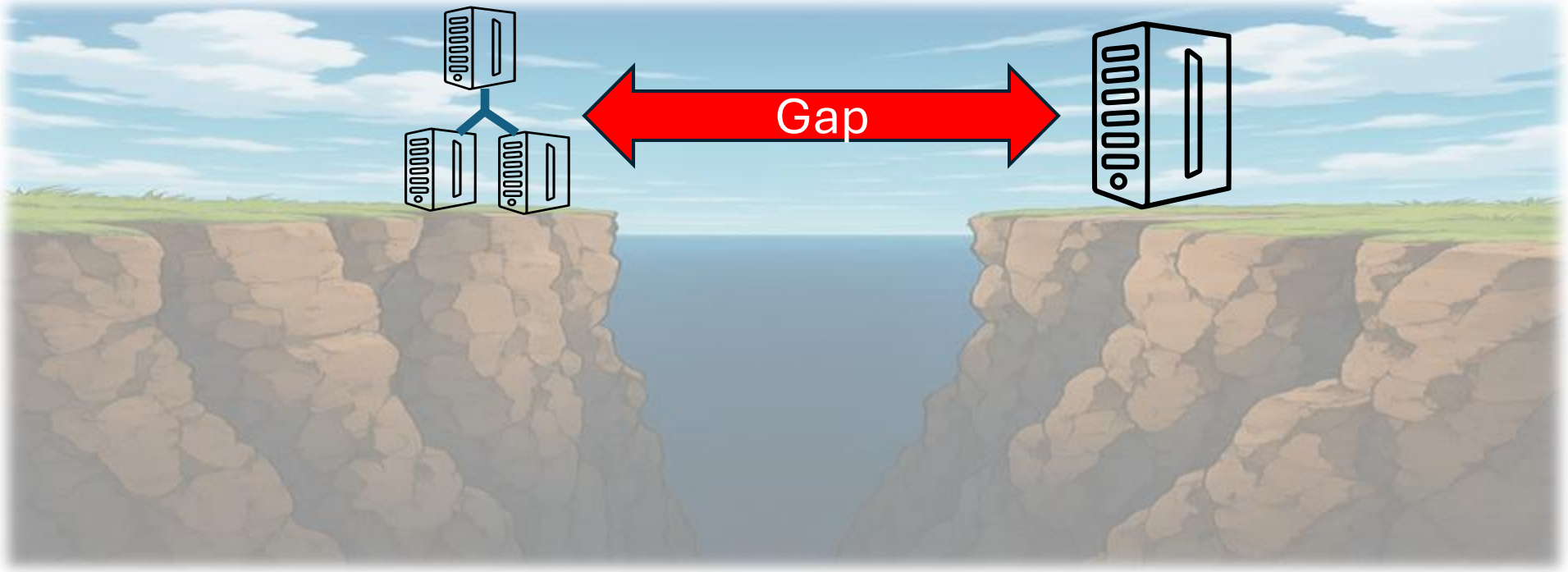
Cannot scale out

No replicas

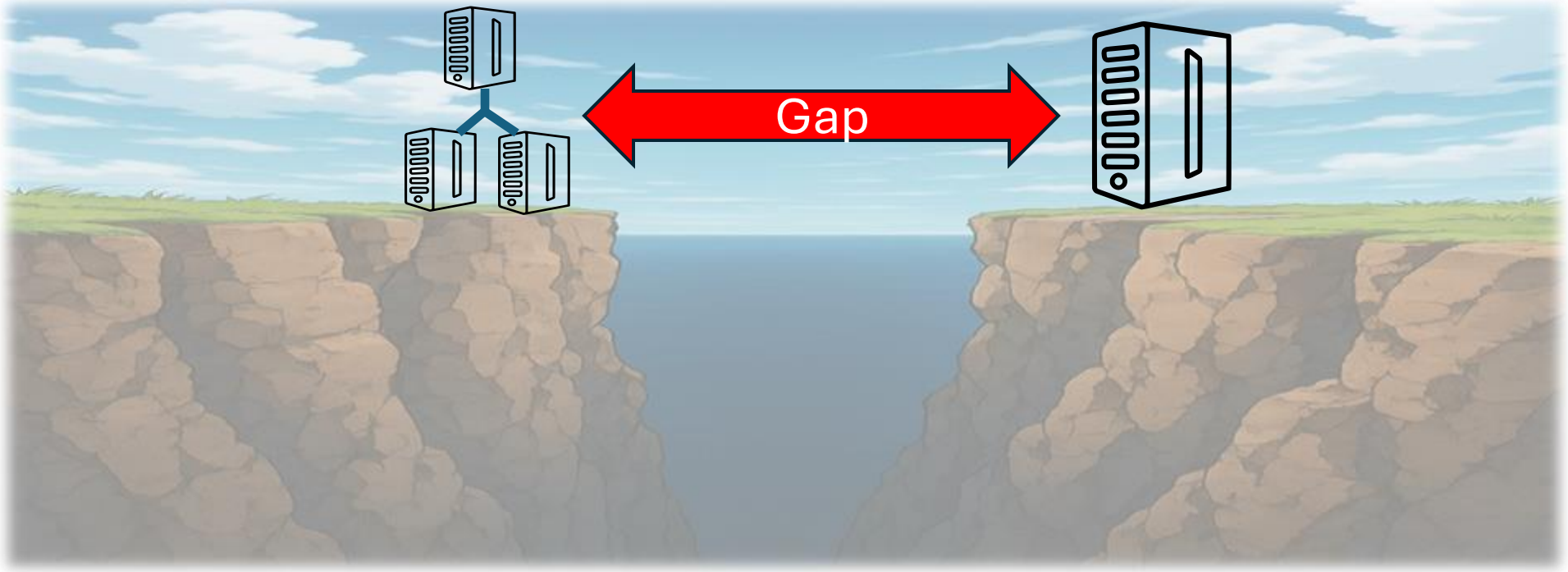
No networking overhead!

Significant networking overhead!

A huge gap between two systems!

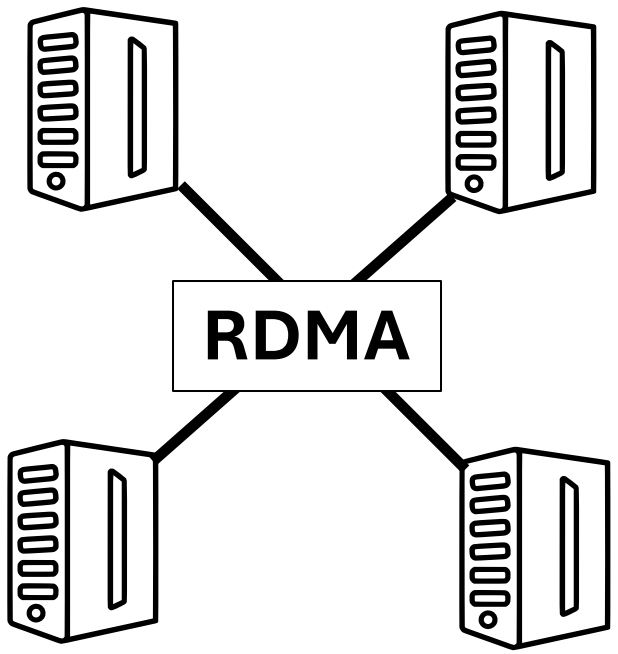


A huge gap between two systems!



Question: *Can we have a system that achieves the best of both worlds—super-high per-node throughput, high scalability and fault-tolerance?*

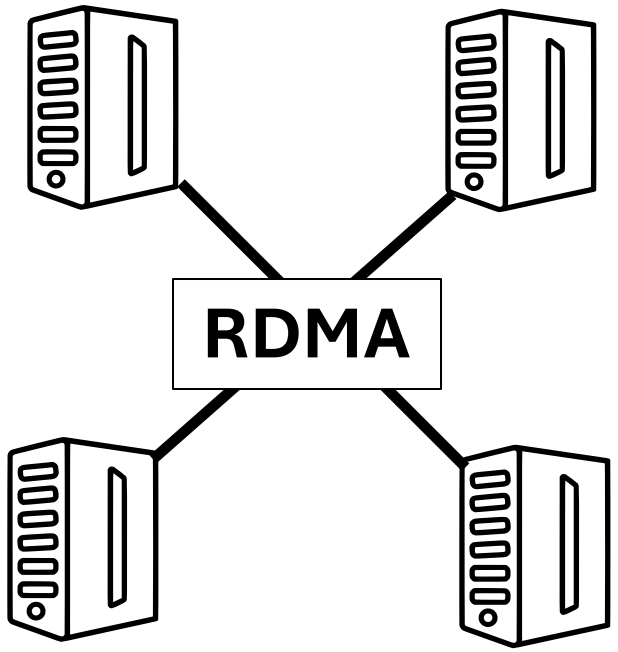
An existing solution: use ultra-fast network



A "single" machine

Existing systems via RDMA:
FaRM [SIGMOD'19], DrTM [SOSP'15] and others

An existing solution: use ultra-fast network

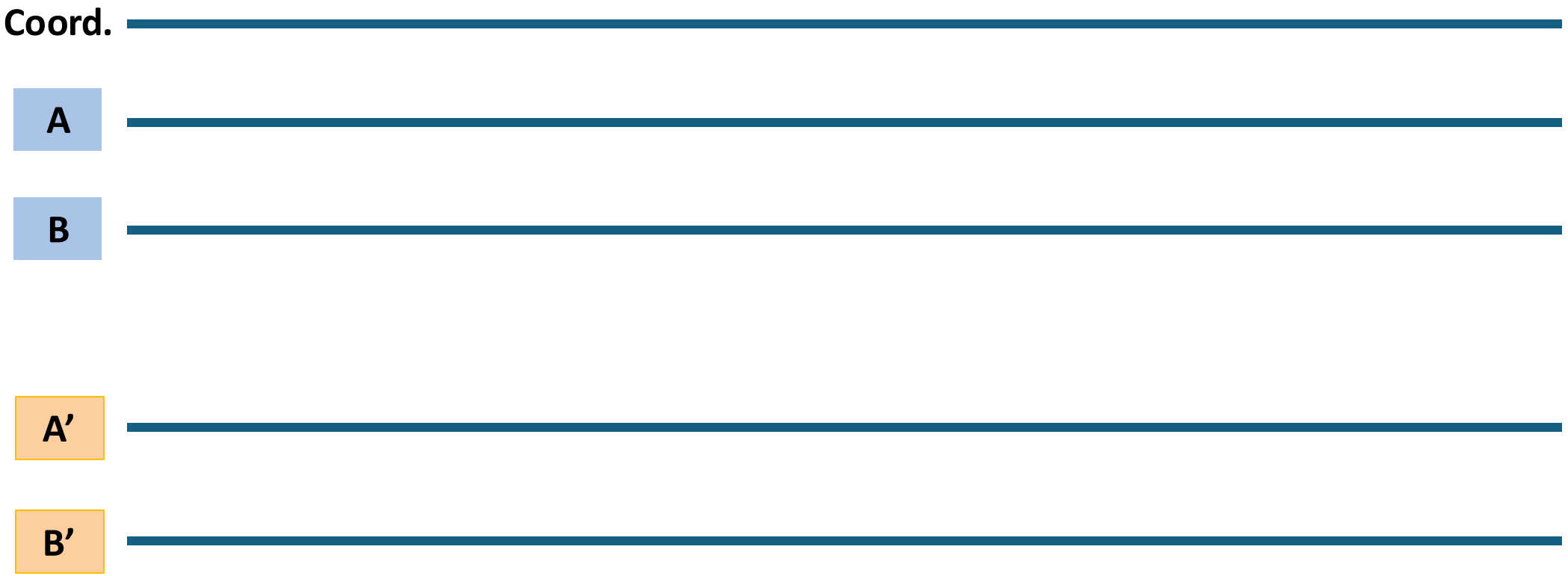


A "single" machine

Existing systems via RDMA:
FaRM [SIGMOD'19], DrTM [SOSP'15] and others

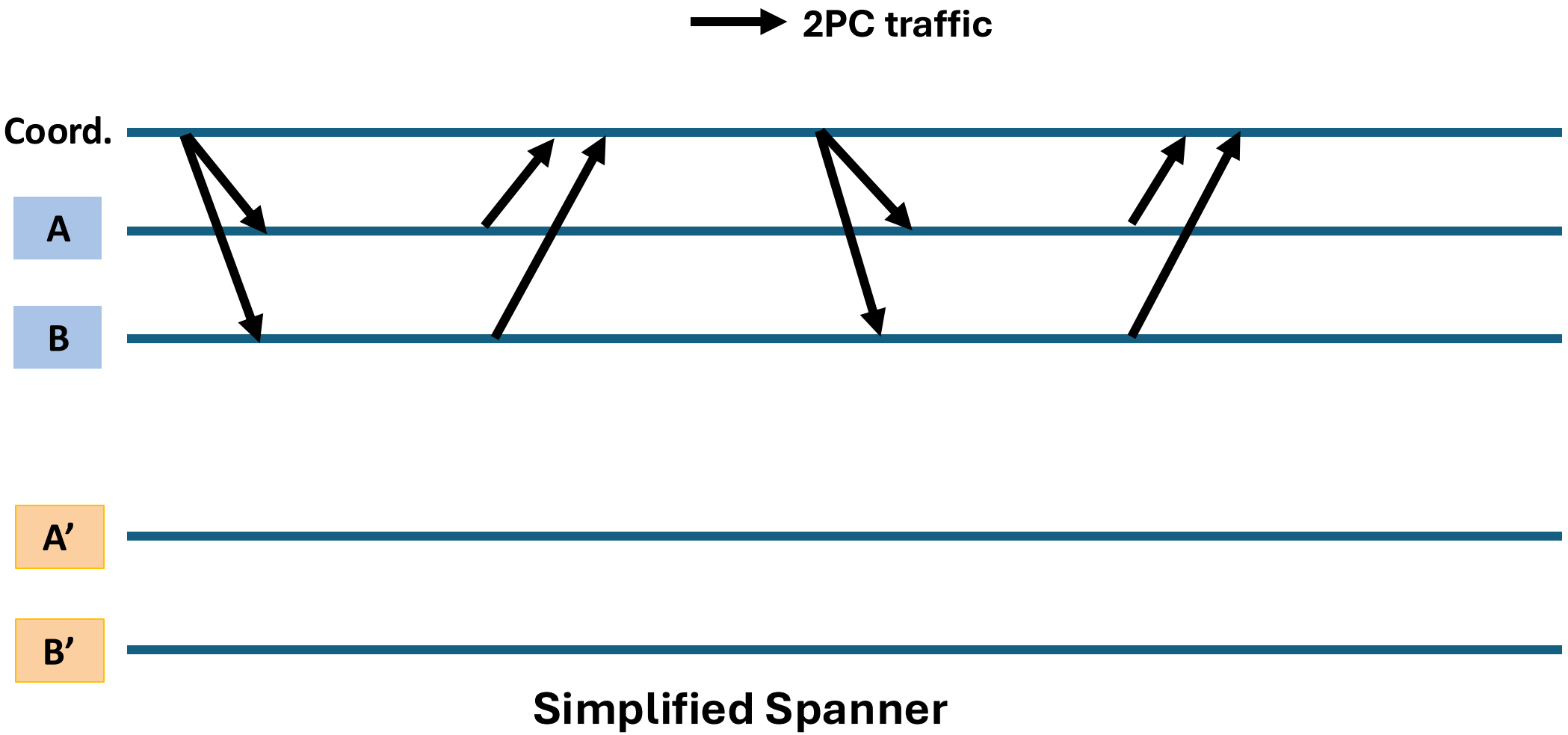
RDMA does not work in geo-replicated setups!

Classic 2PC+Paxos

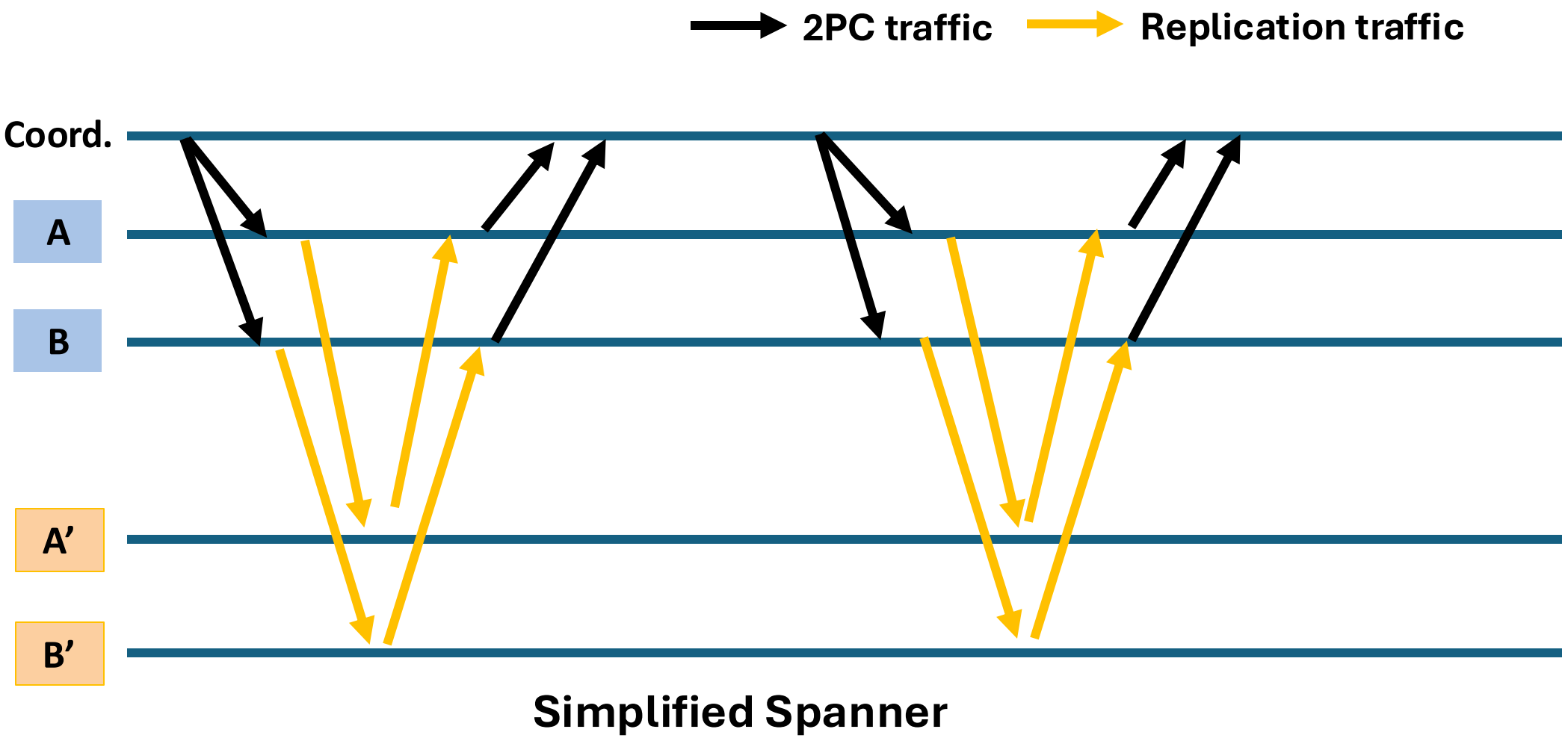


Simplified Spanner

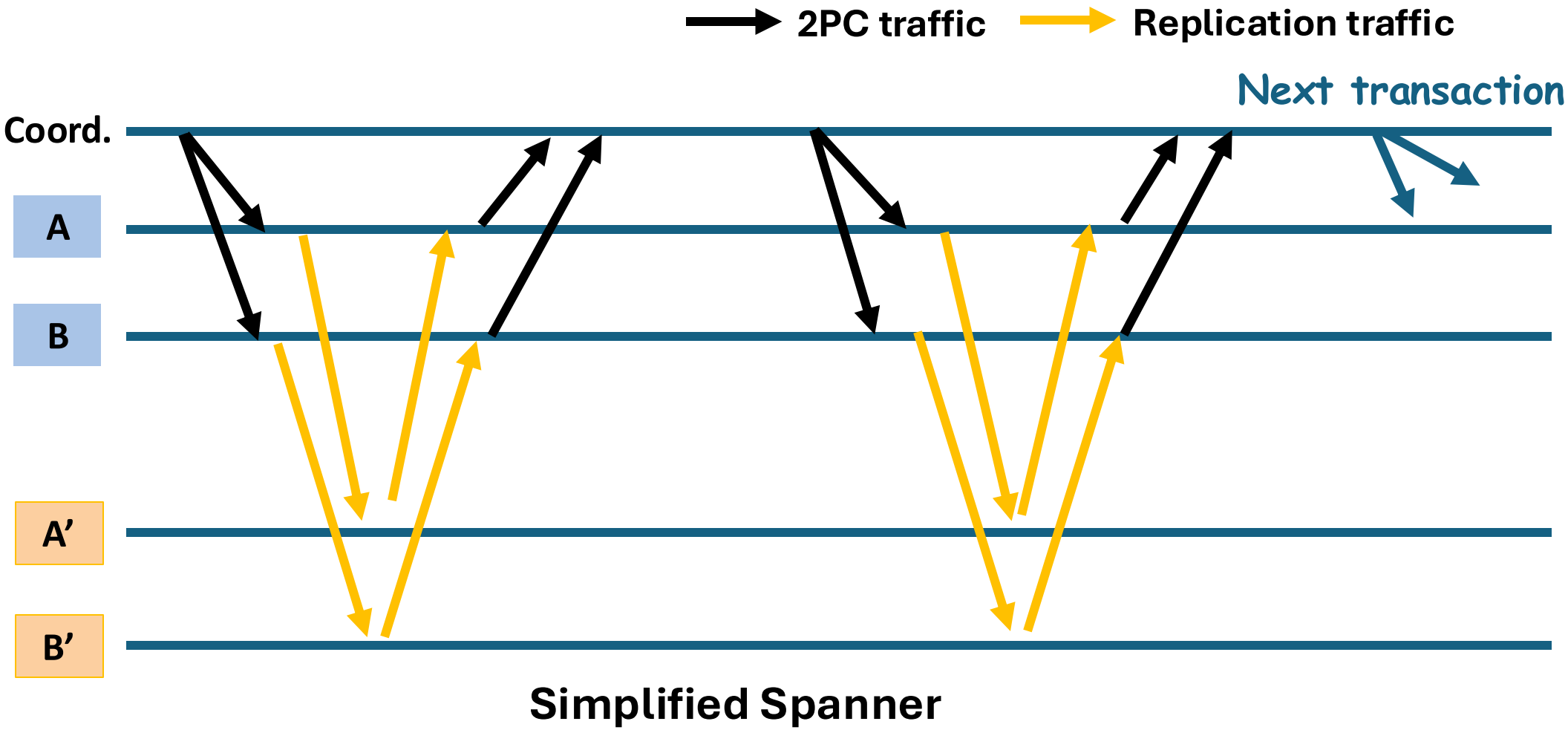
Classic 2PC+Paxos



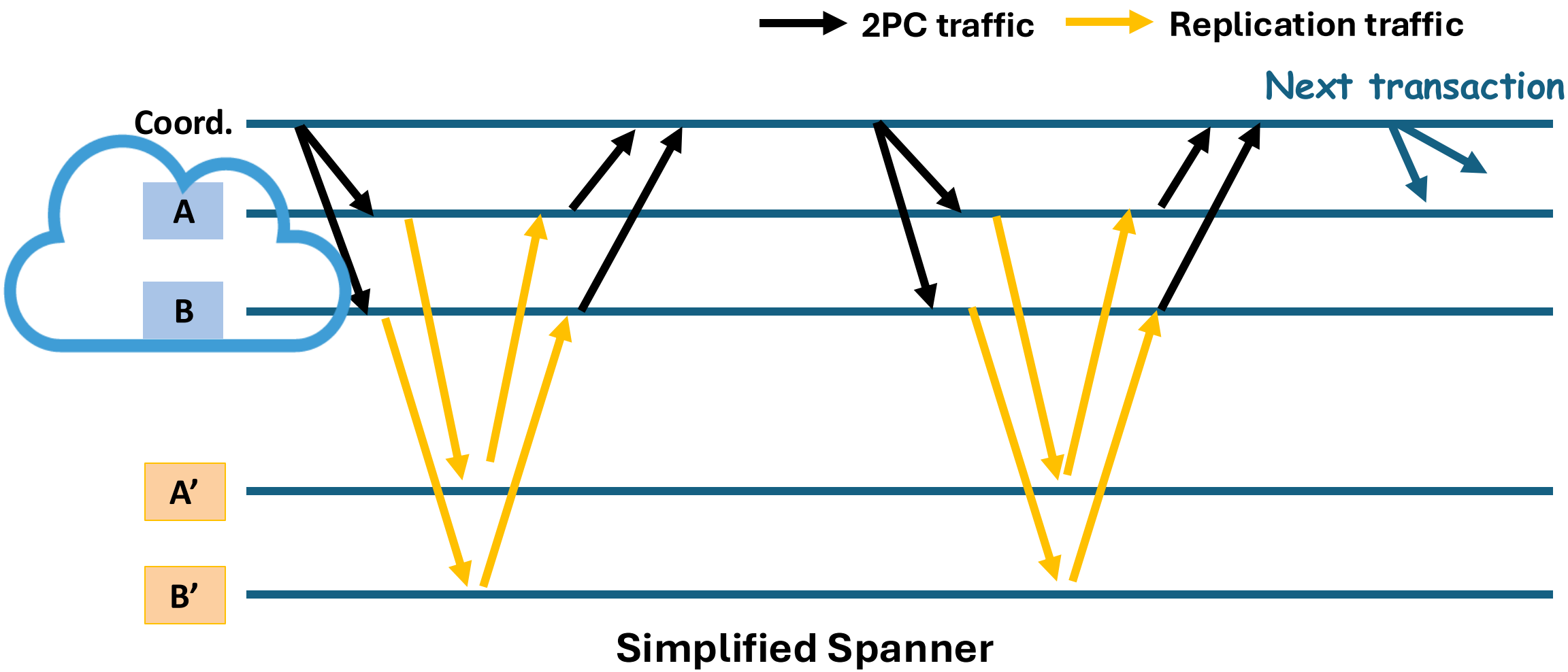
Classic 2PC+Paxos



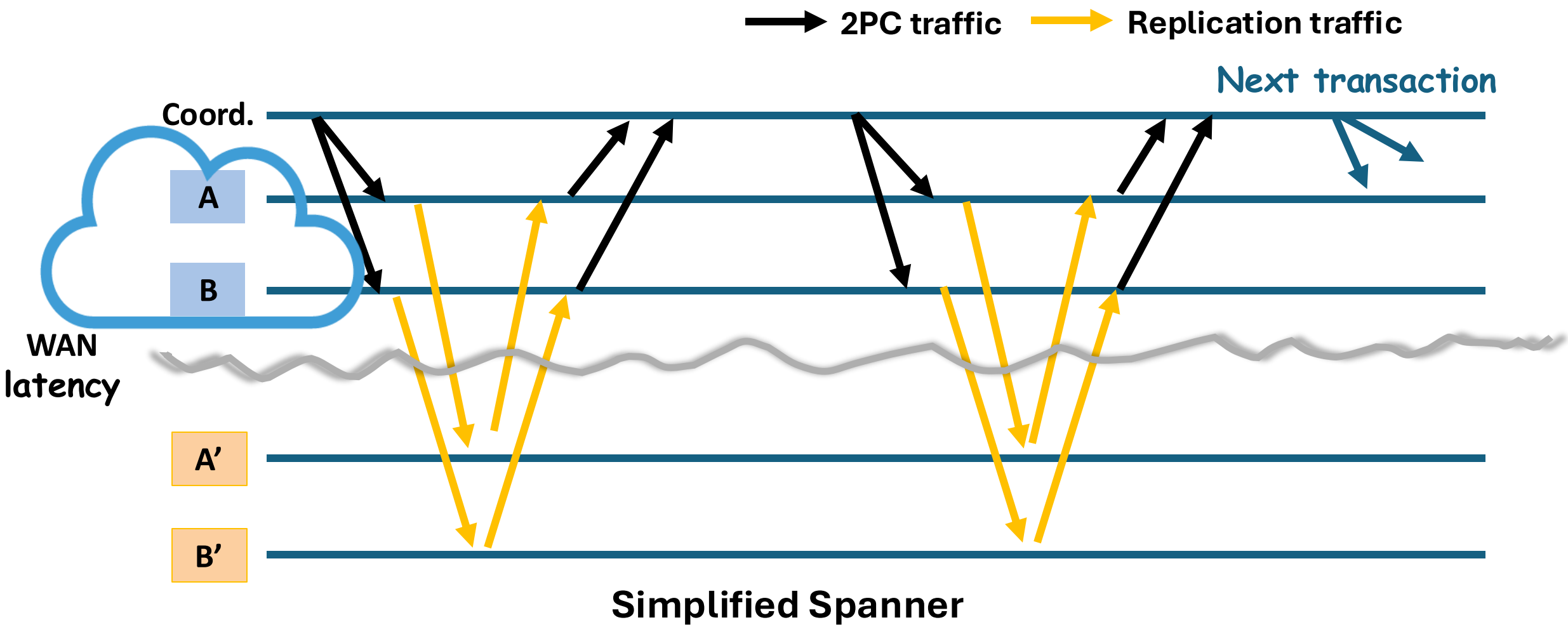
Classic 2PC+Paxos



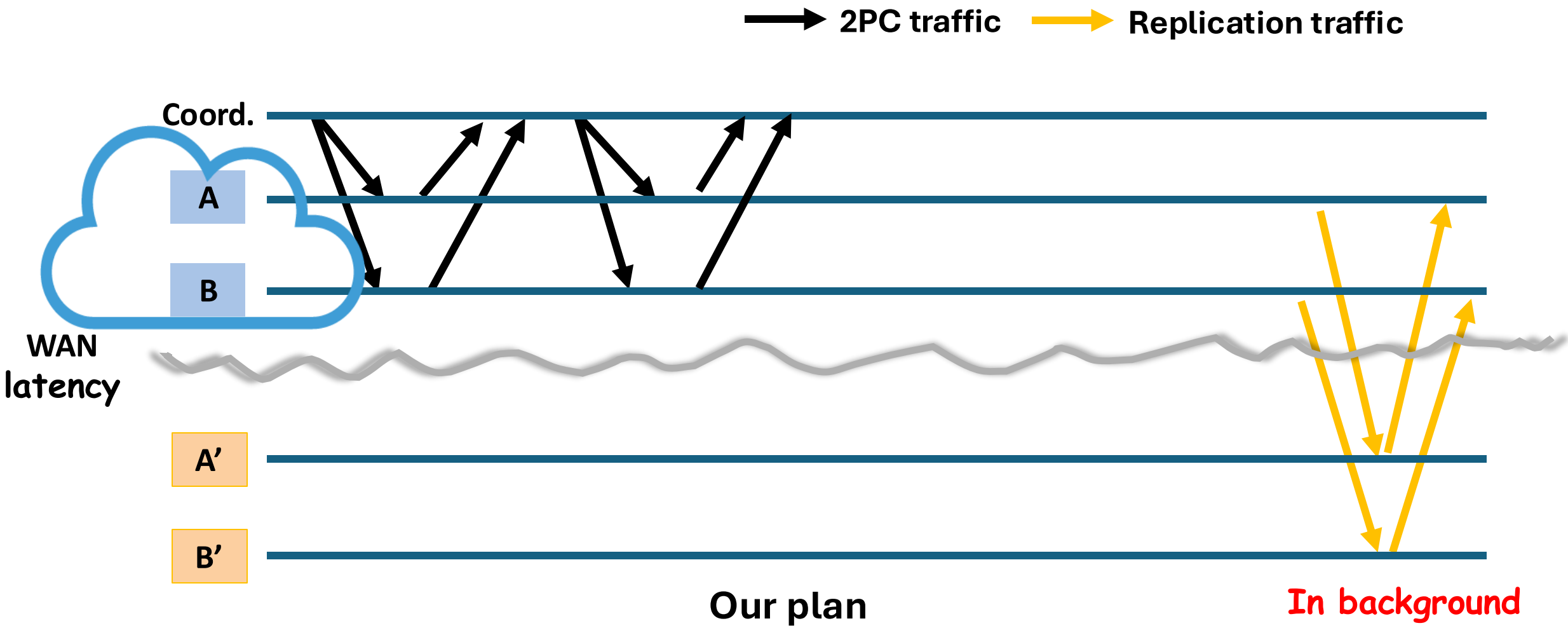
Classic 2PC+Paxos



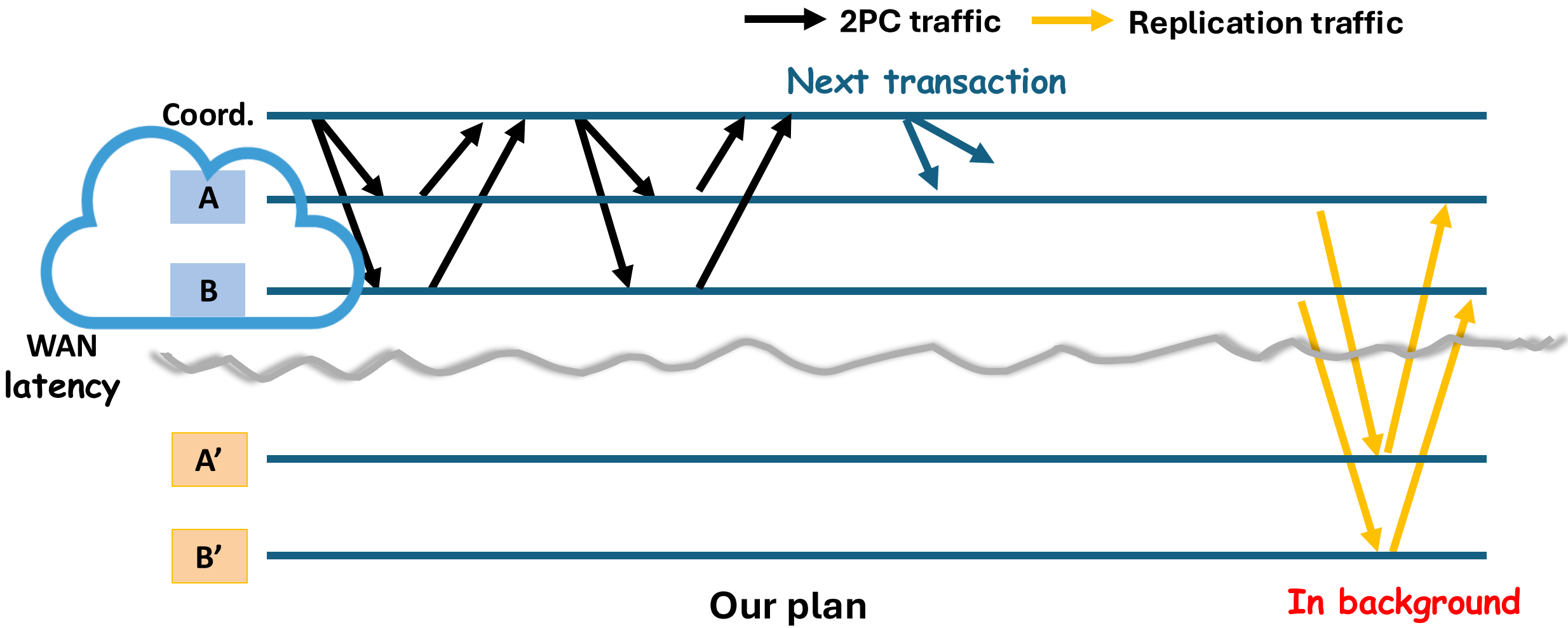
Classic 2PC+Paxos



Our plan: decouple replication!



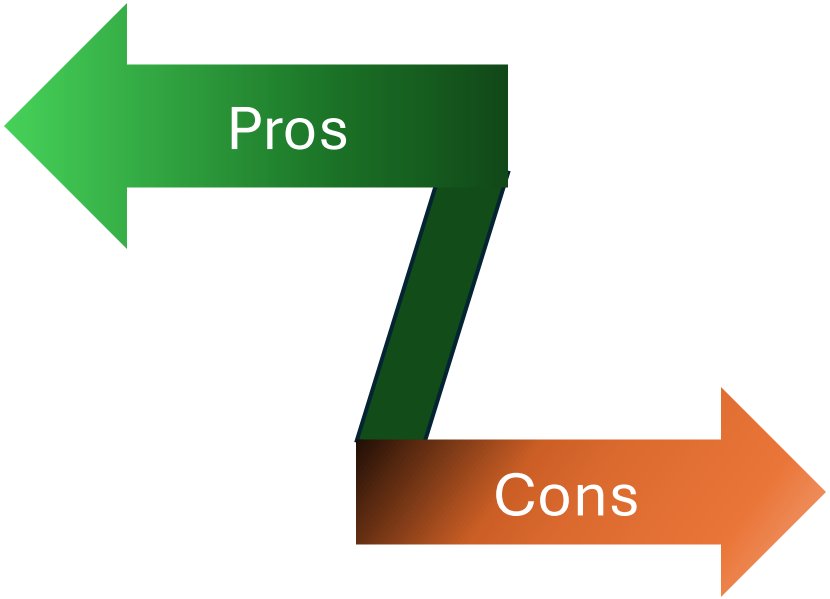
Our plan: decouple replication!



Our plan: decouple replication!

Very high throughput!

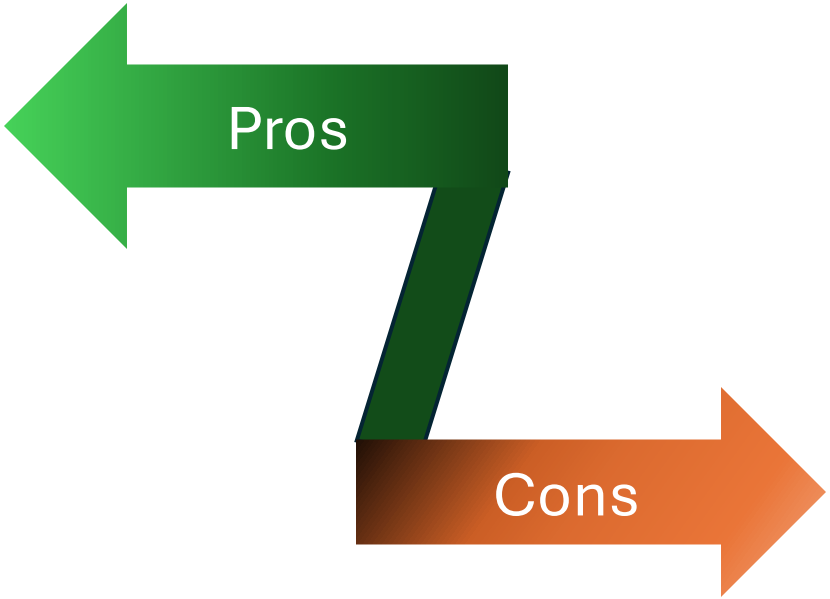
Lower cost for cloud databases!



Our plan: decouple replication!

Very high throughput!

Lower cost for cloud databases!



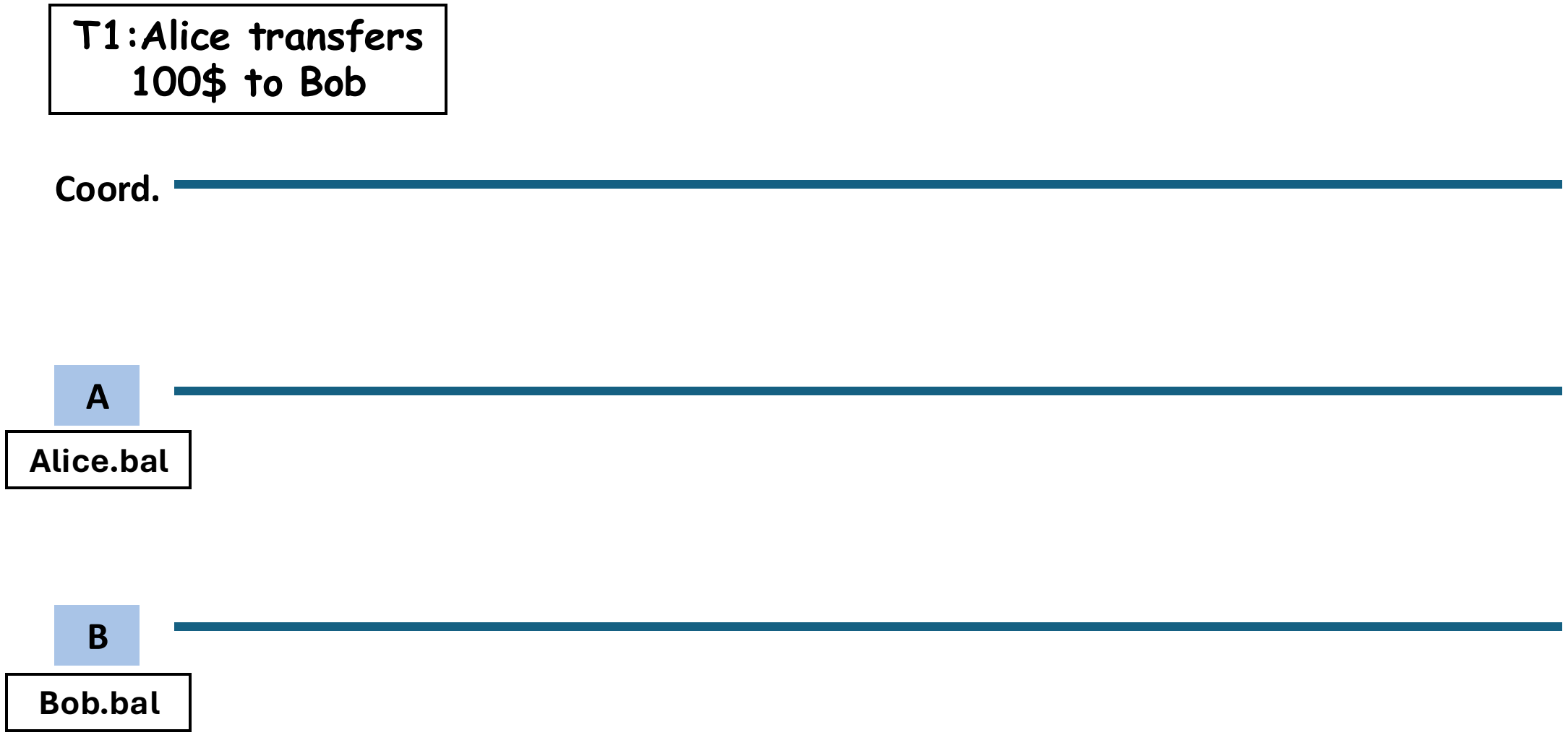
Modest extra latency

Not our goal

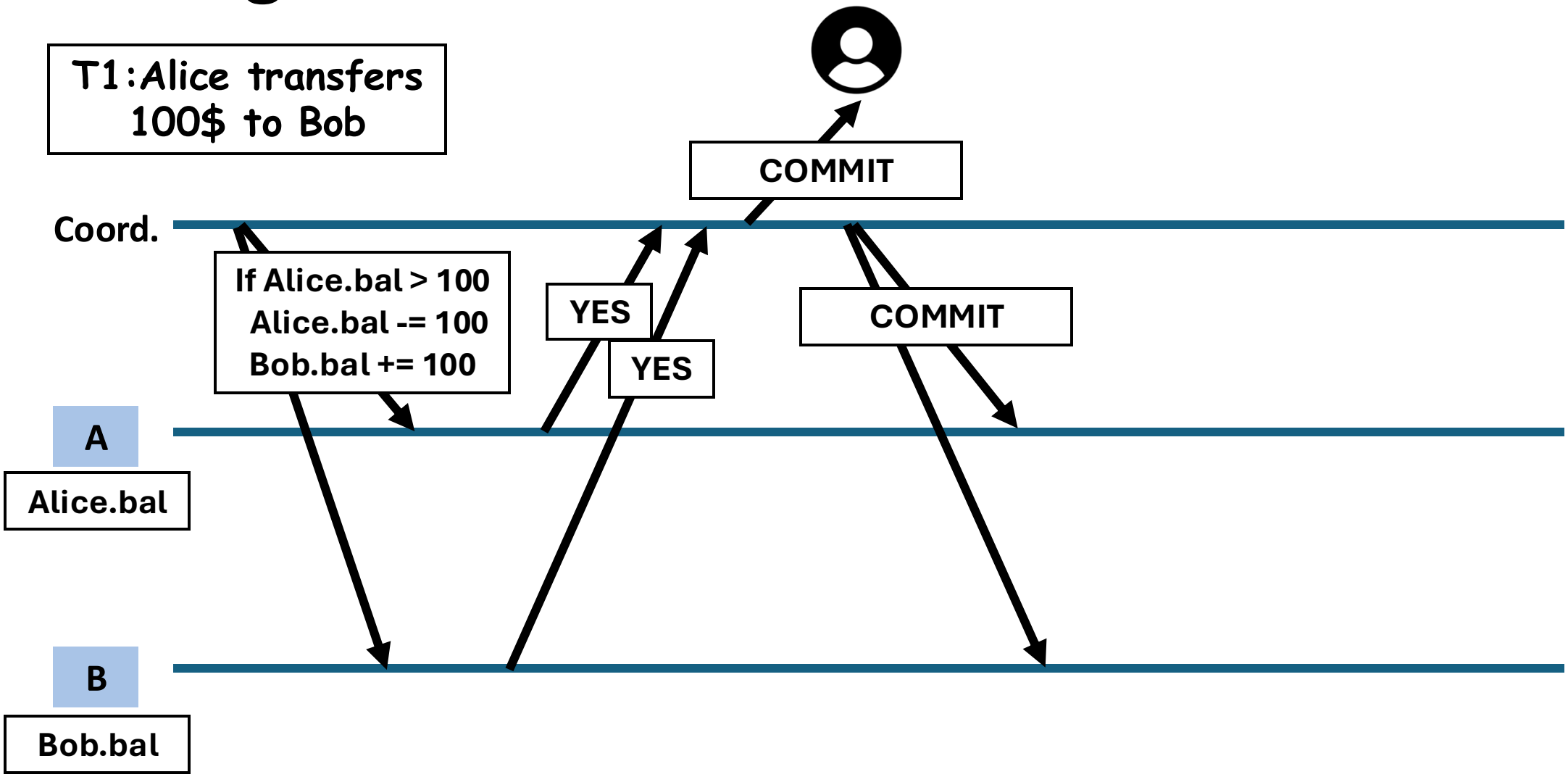
Prior works with a similar issue: rollback too many unnecessary transactions during failures

Our key technical contribution addresses this issue

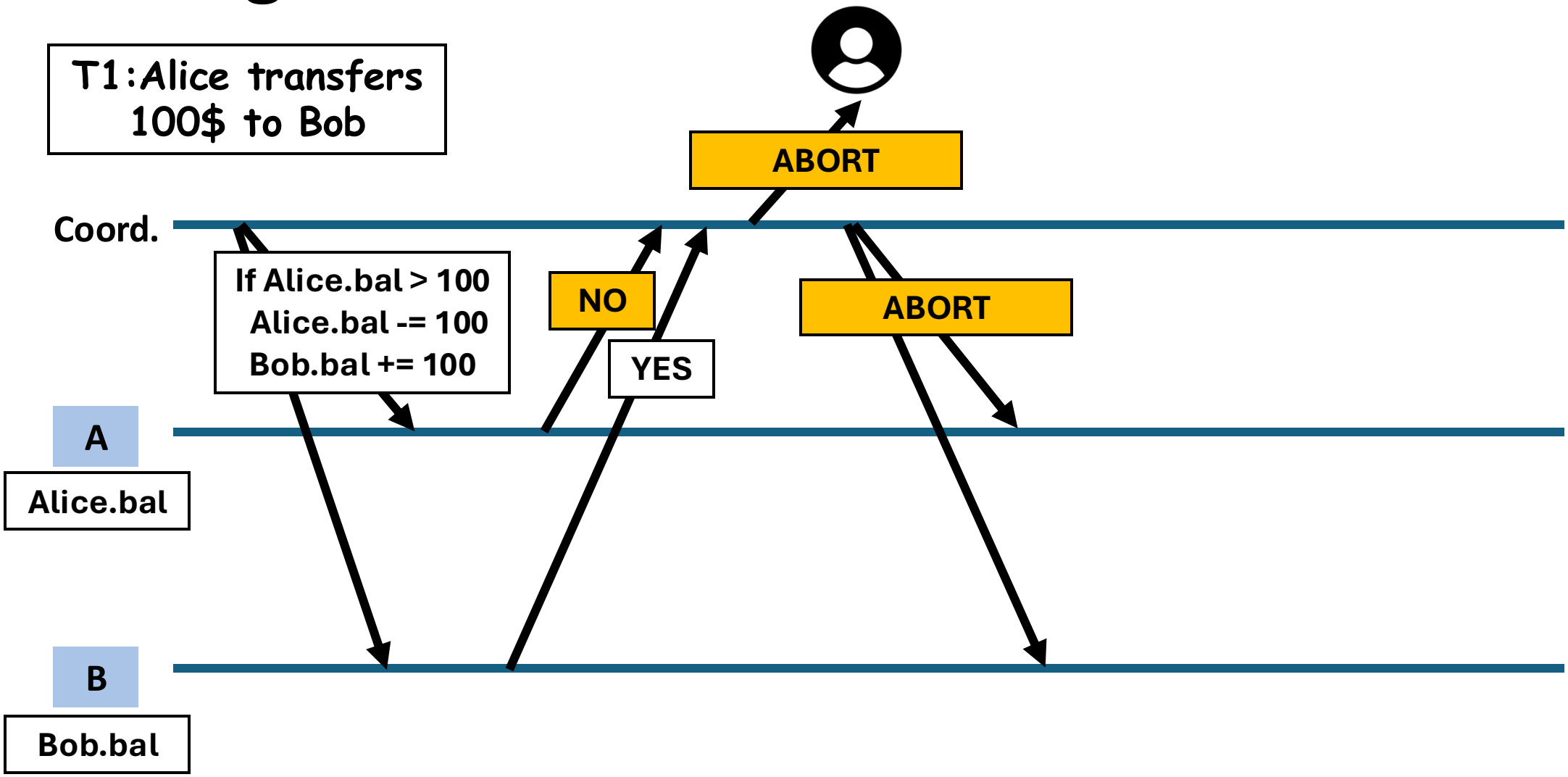
Challenge#1: 2PC isn't fault-tolerant



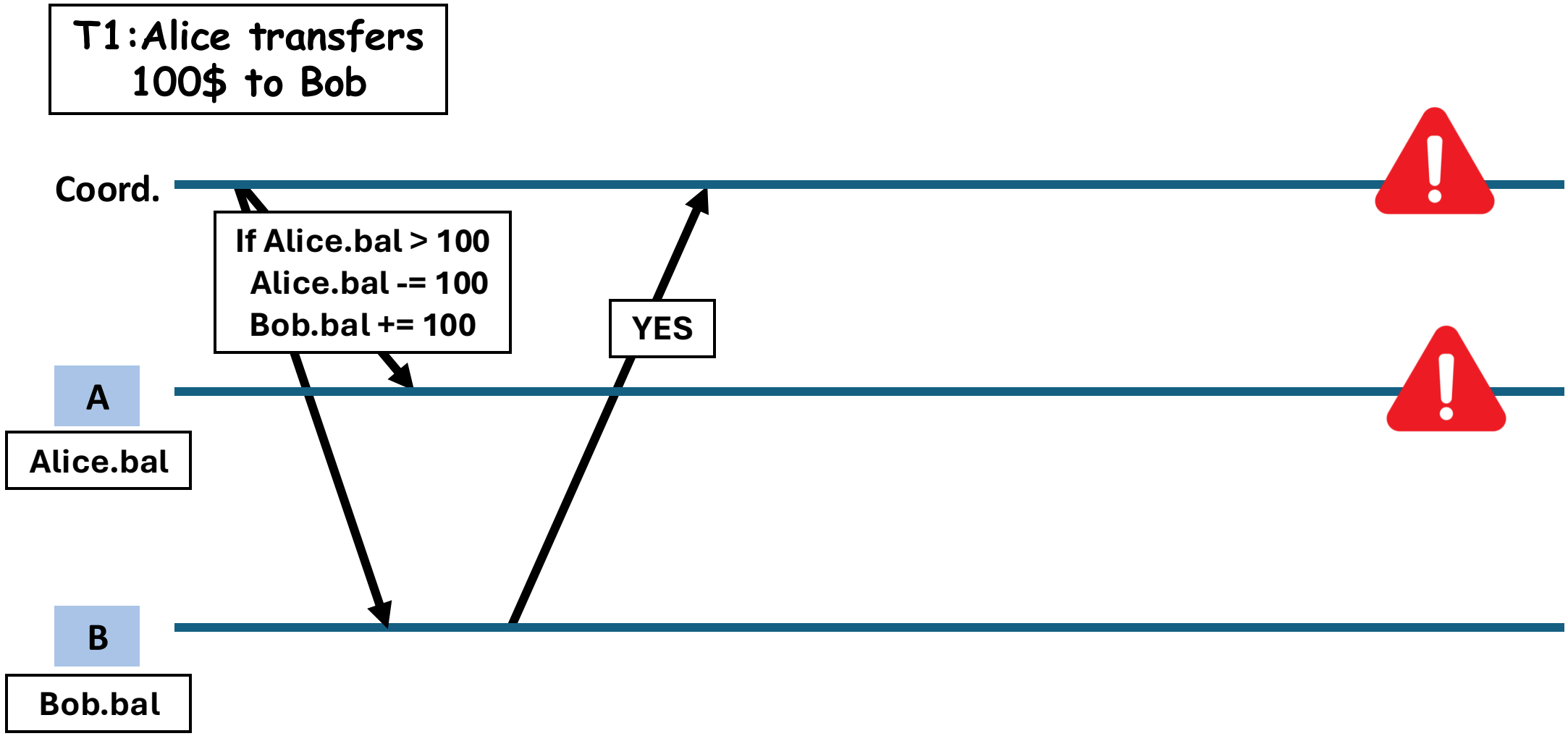
Challenge#1: 2PC isn't fault-tolerant



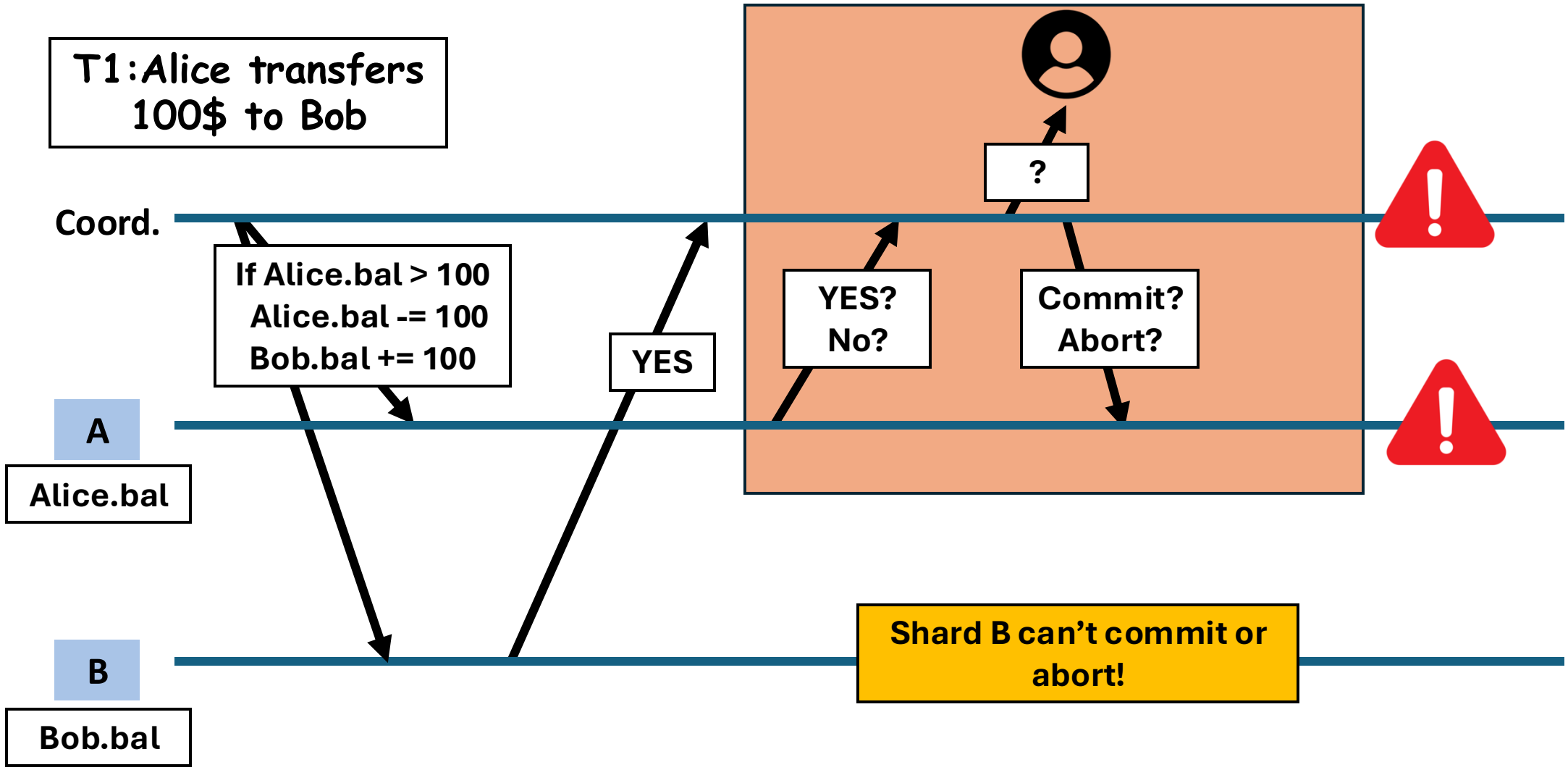
Challenge#1: 2PC isn't fault-tolerant



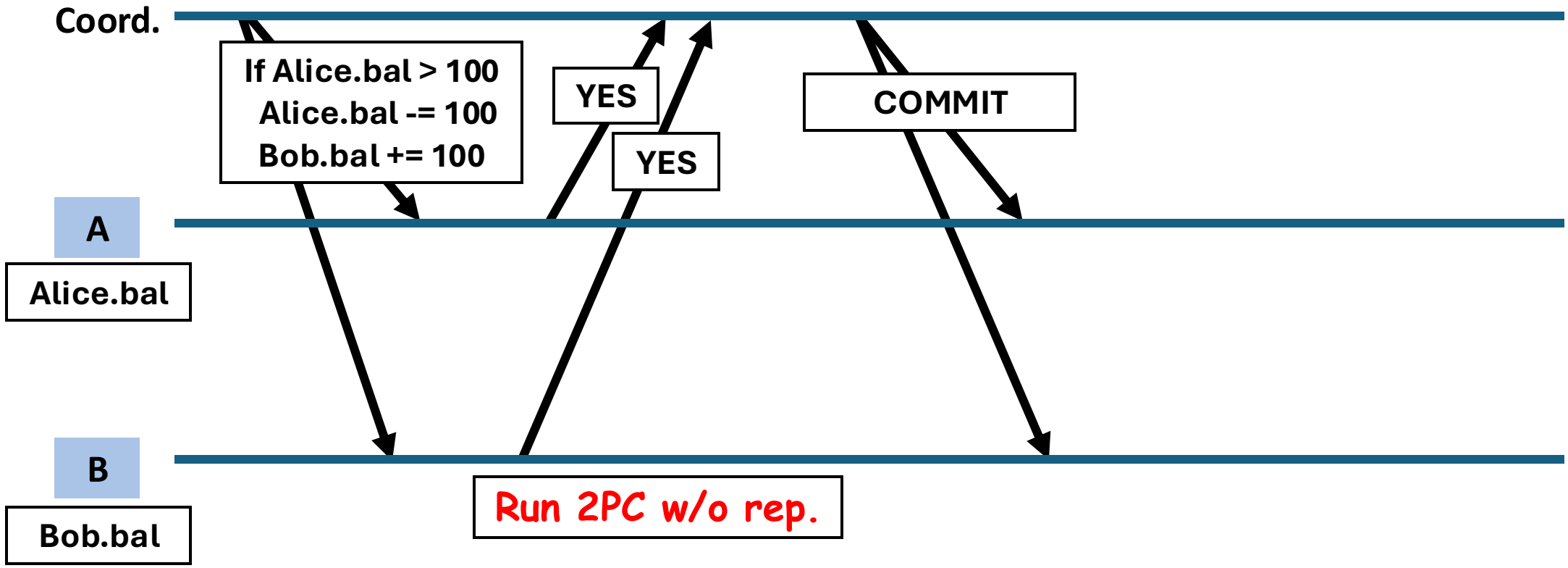
Challenge#1: 2PC isn't fault-tolerant



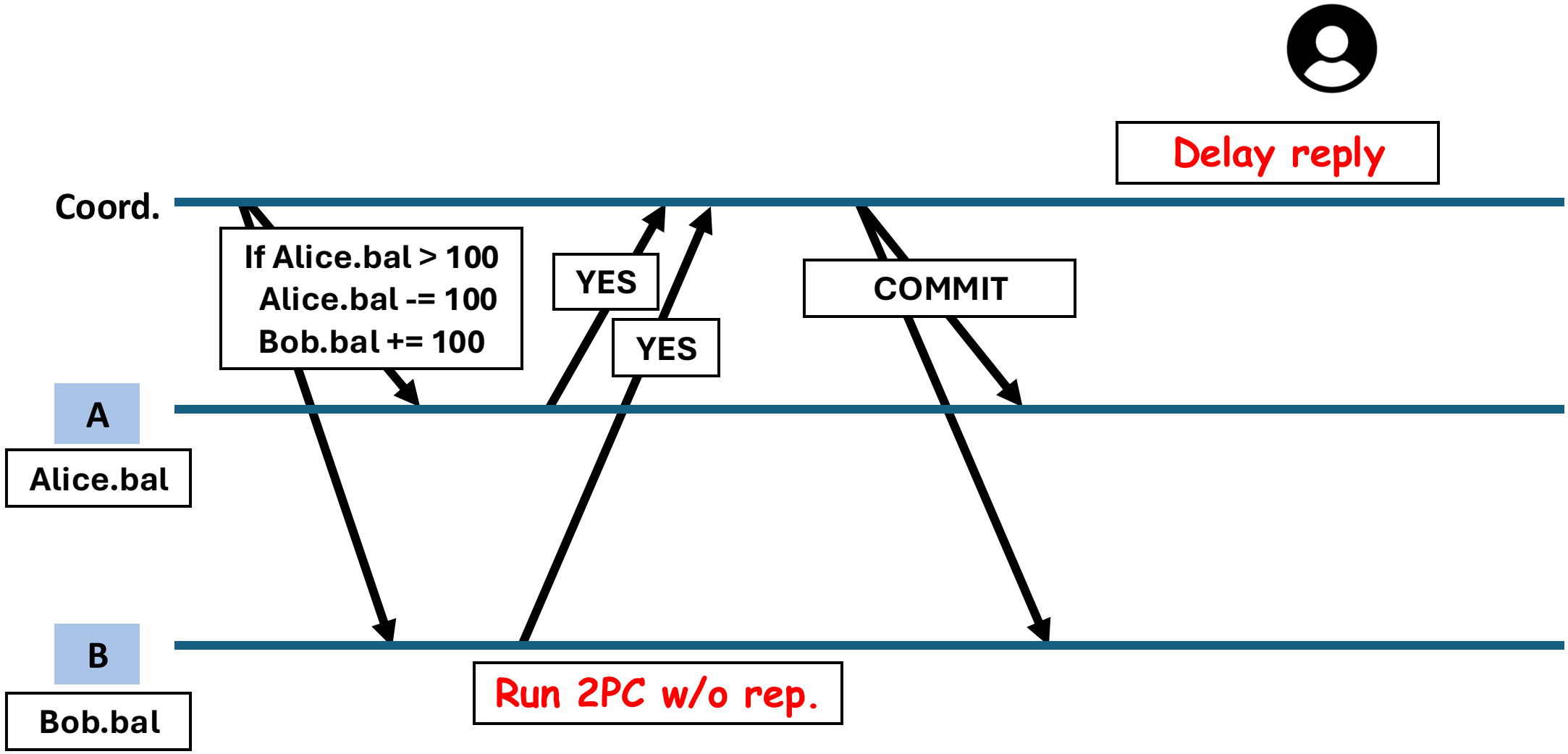
Challenge#1: 2PC isn't fault-tolerant



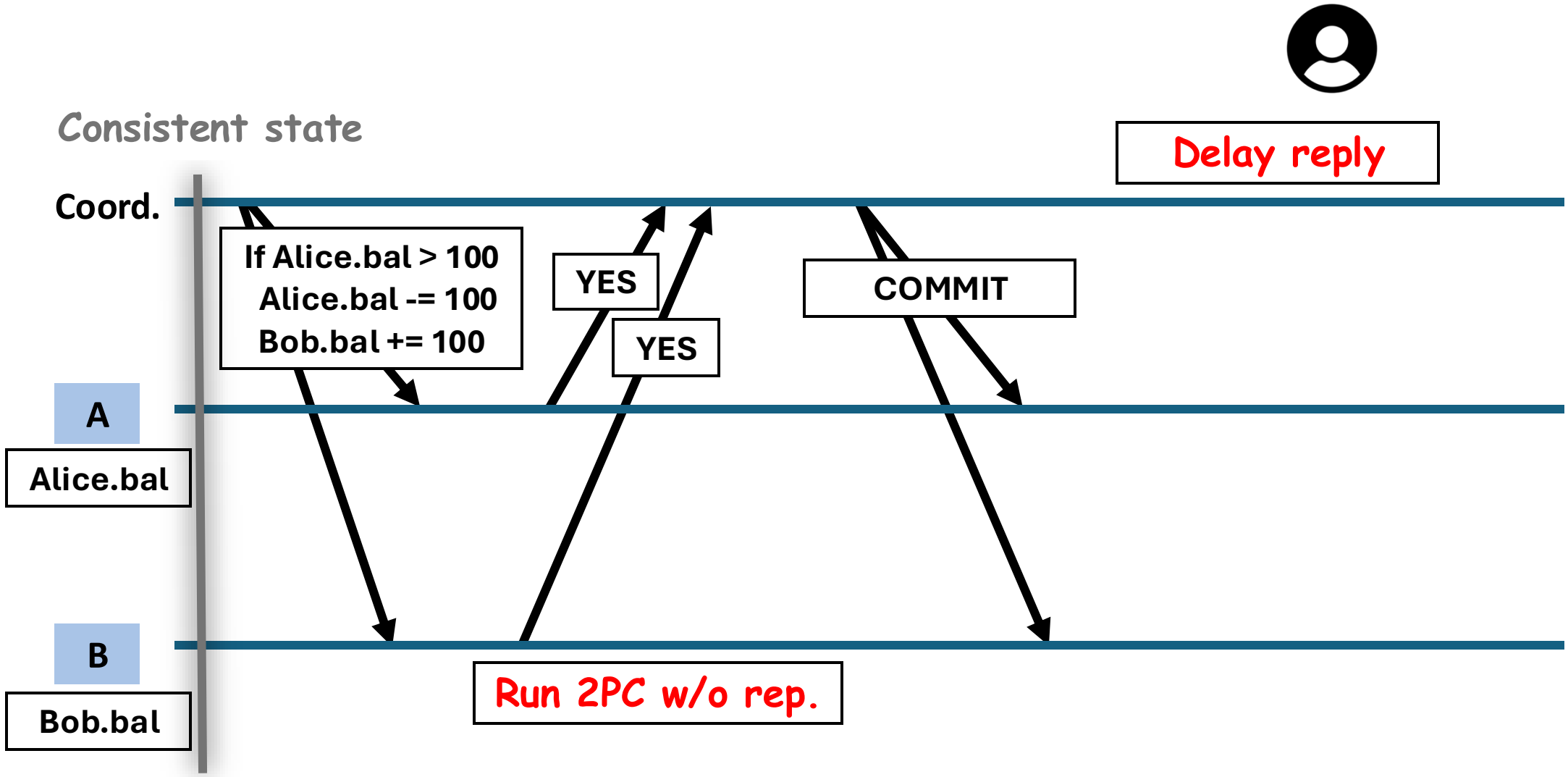
Solution#1: Rollback failed 2PC results



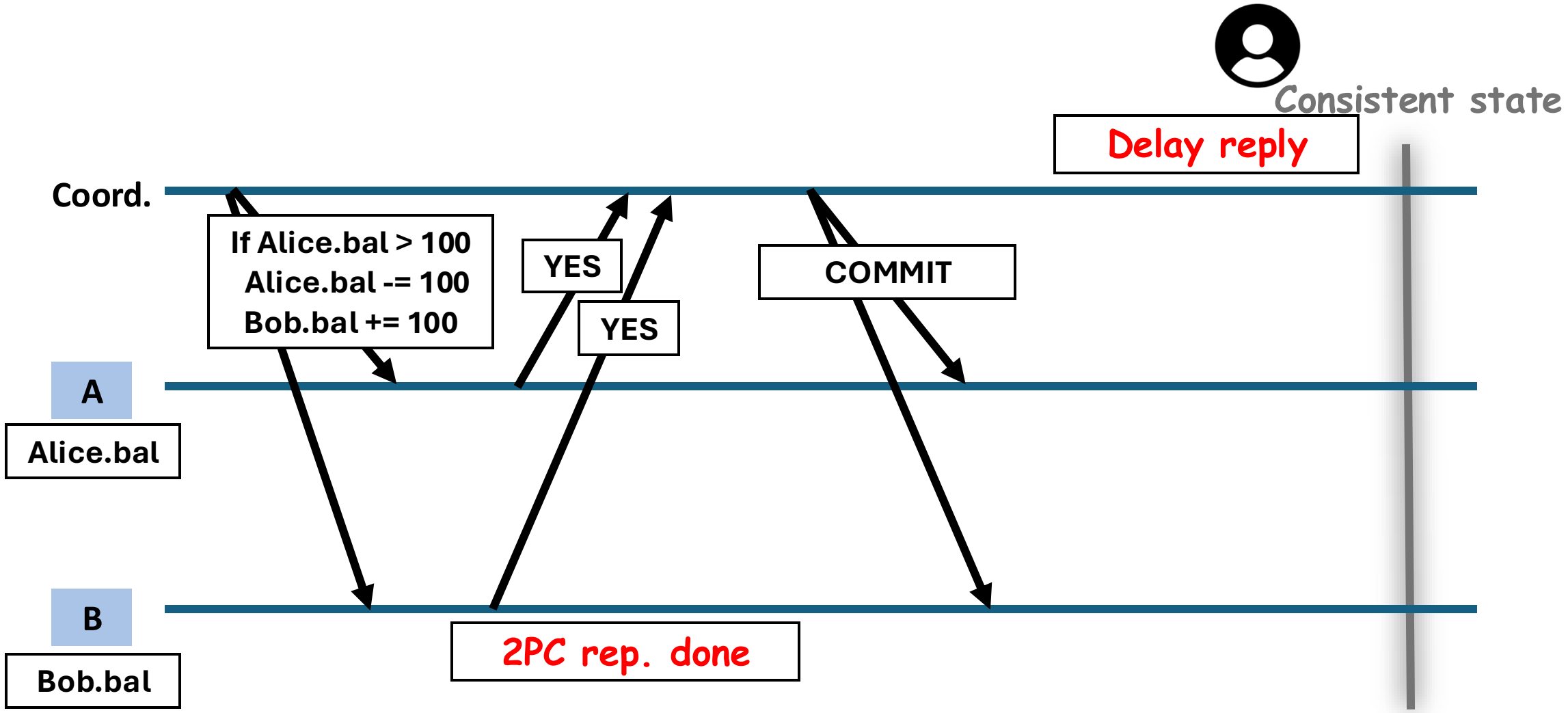
Solution#1: Rollback failed 2PC results



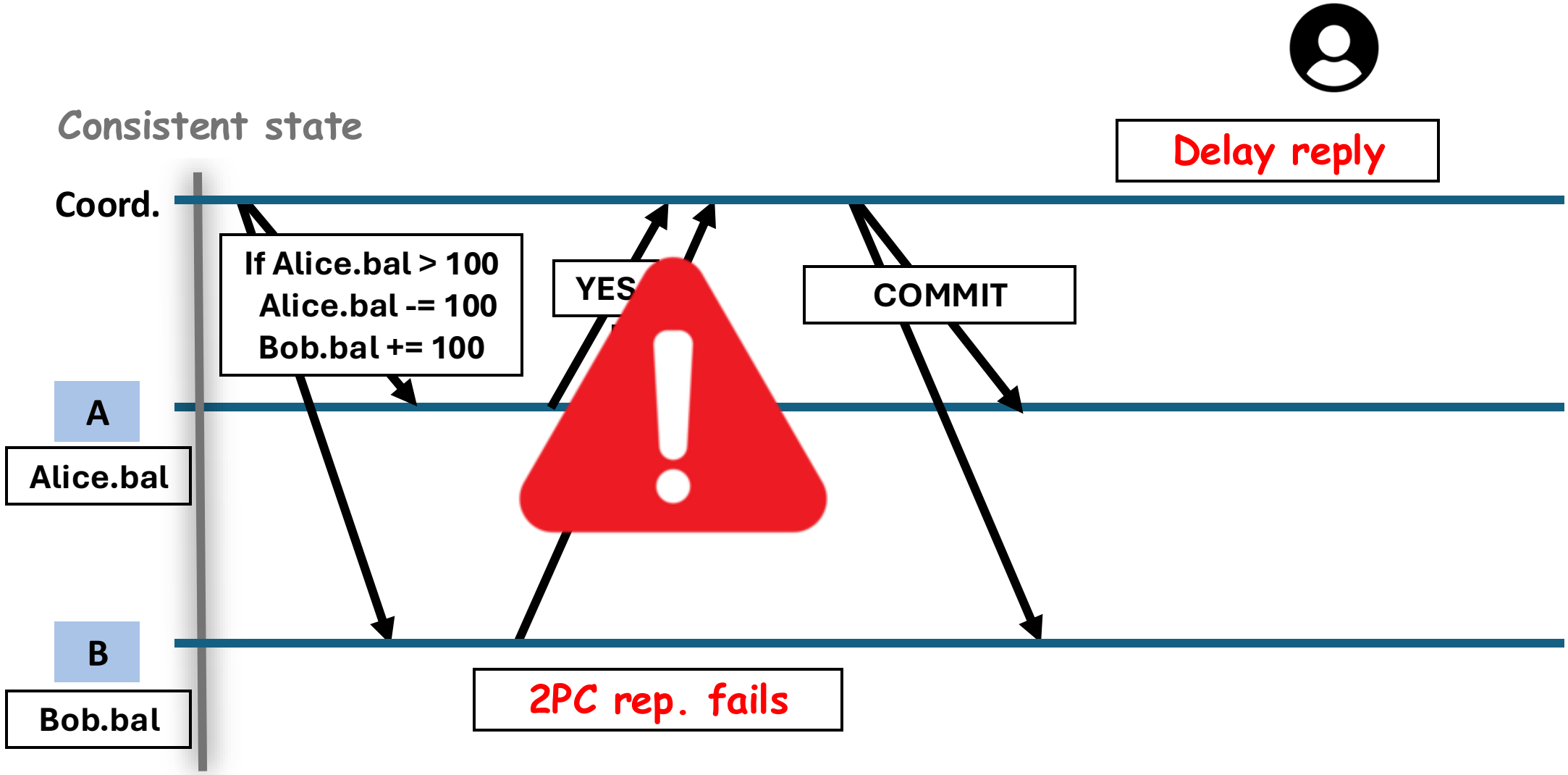
Solution#1: Rollback failed 2PC results



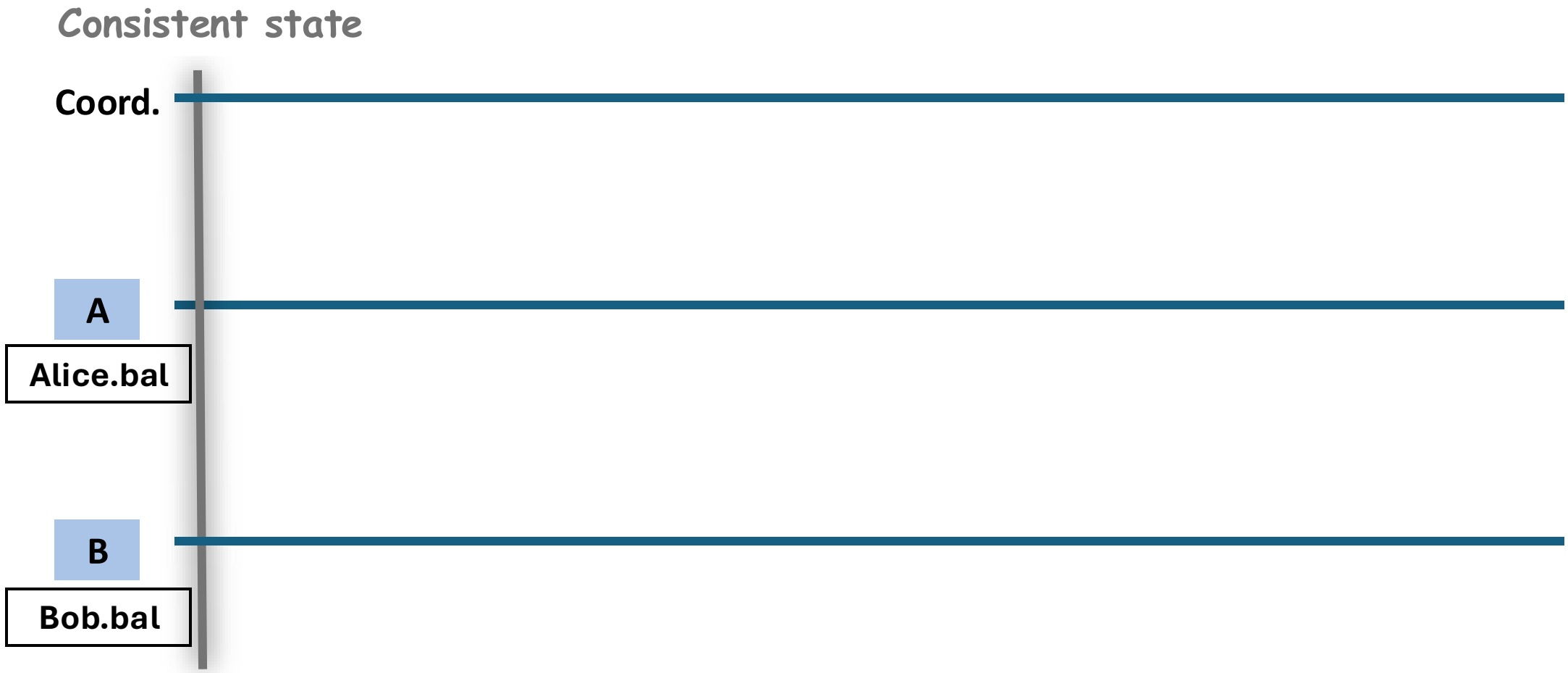
Solution#1: Rollback failed 2PC results



Solution#1: Rollback failed 2PC results



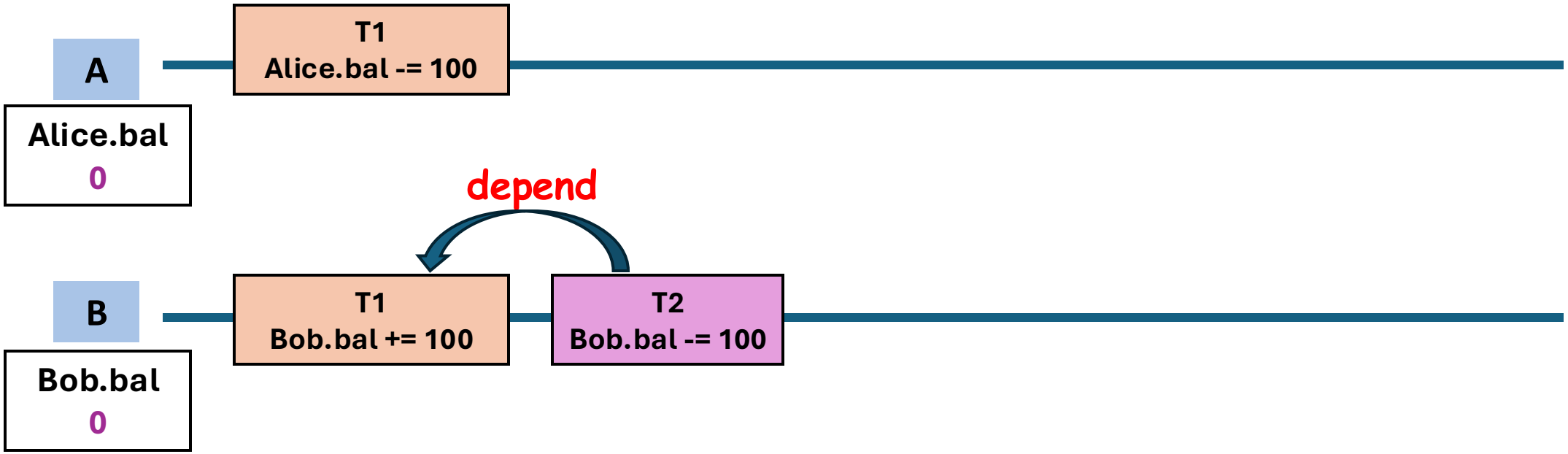
Solution#1: Rollback failed 2PC results



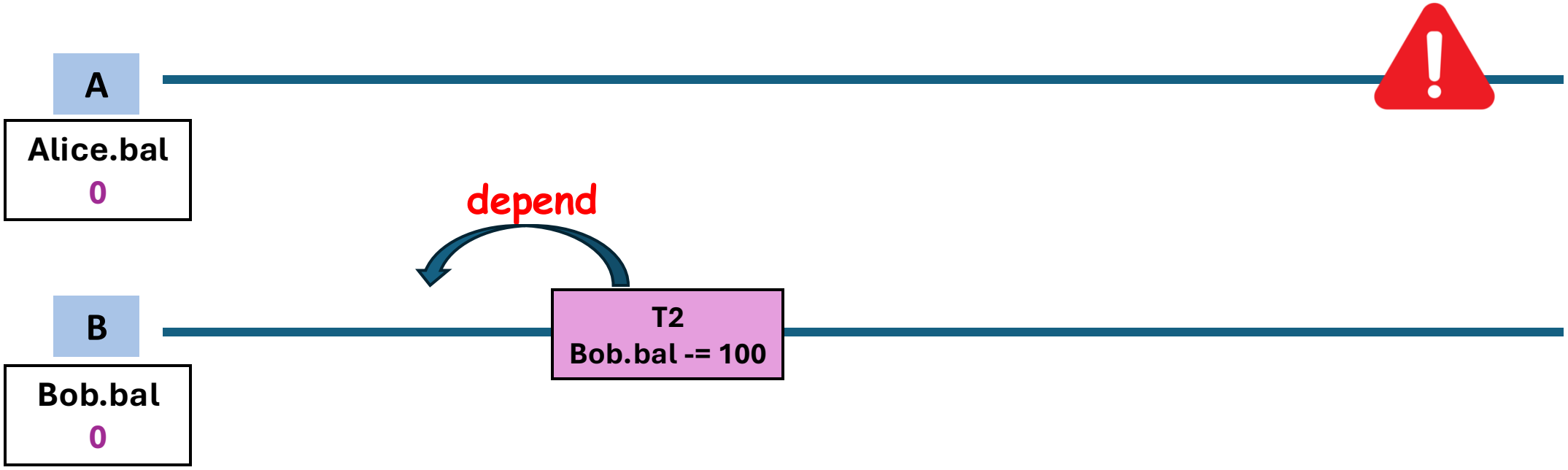
Challenge#2: How about dependent transactions?

Challenge#2: How about dependent transactions?

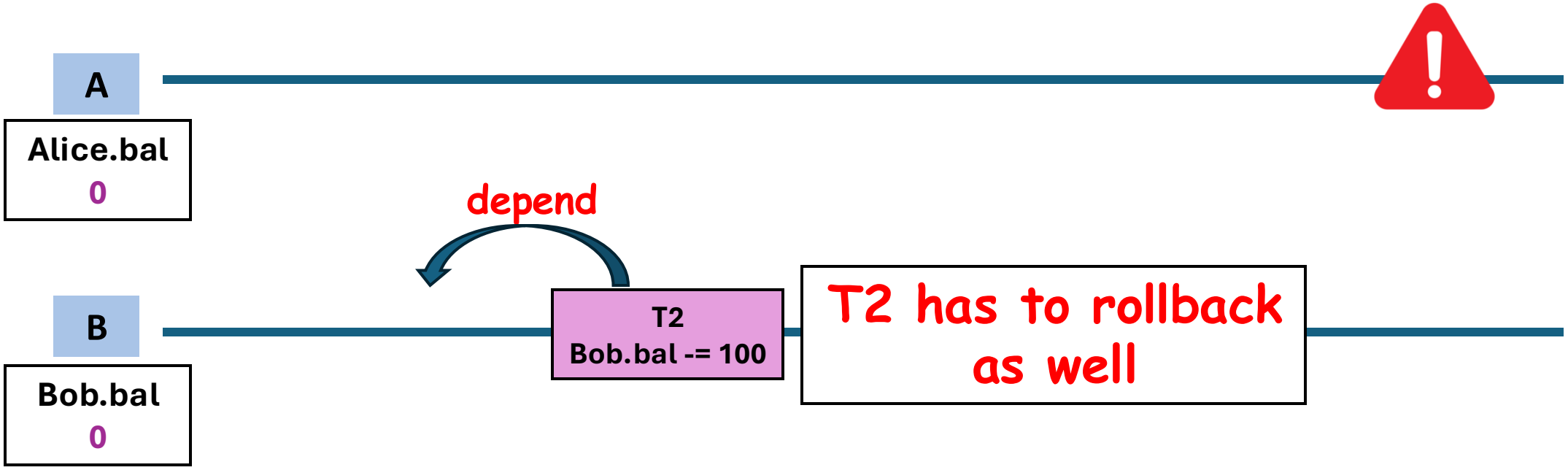
Challenge#2: How about dependent transactions?



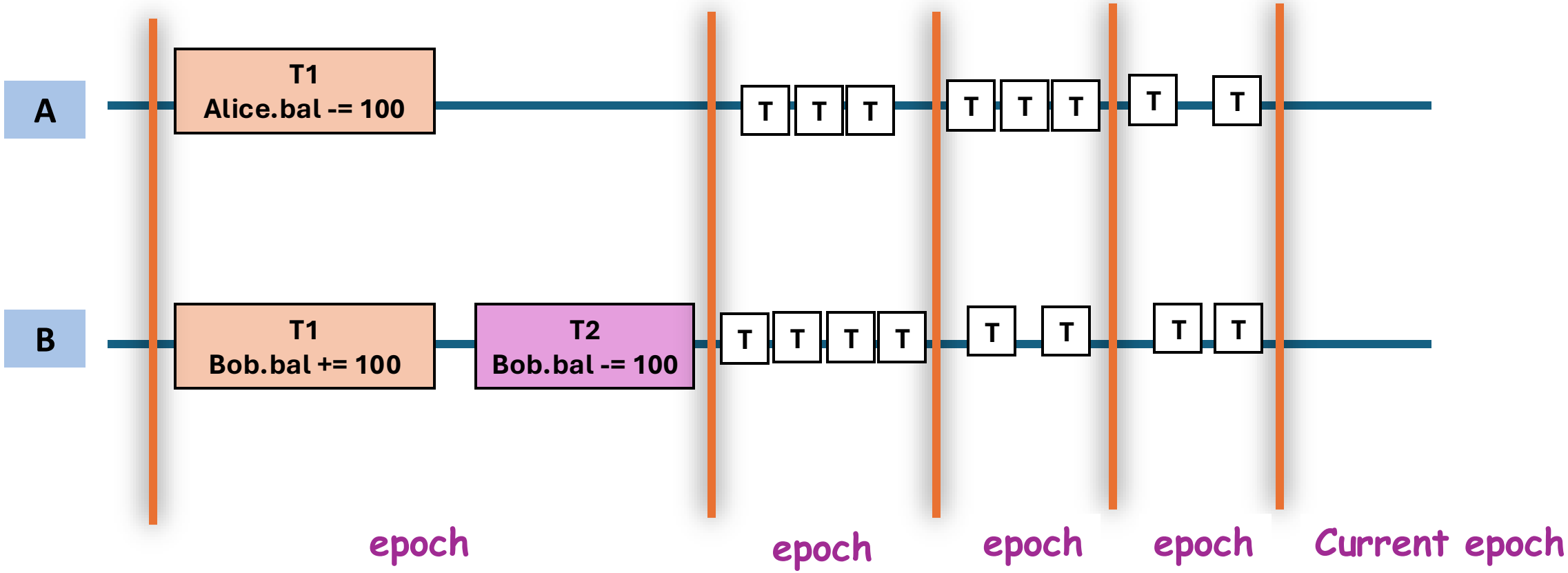
Challenge#2: How about dependent transactions?



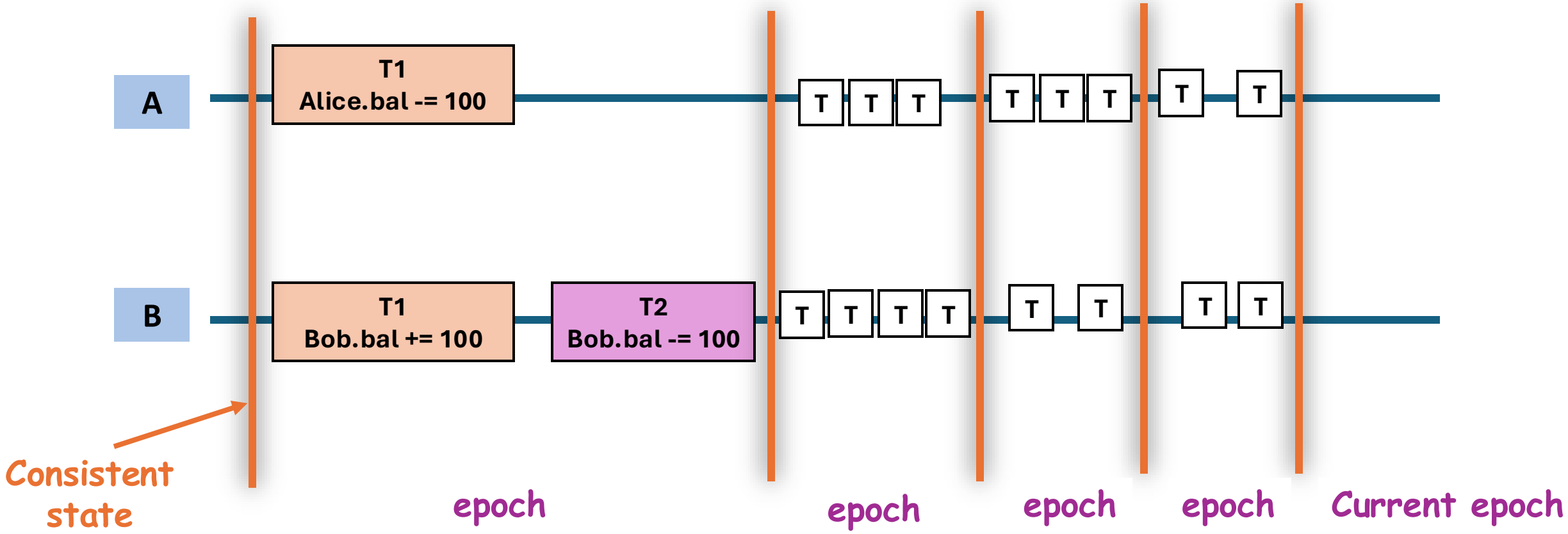
Challenge#2: How about dependent transactions?



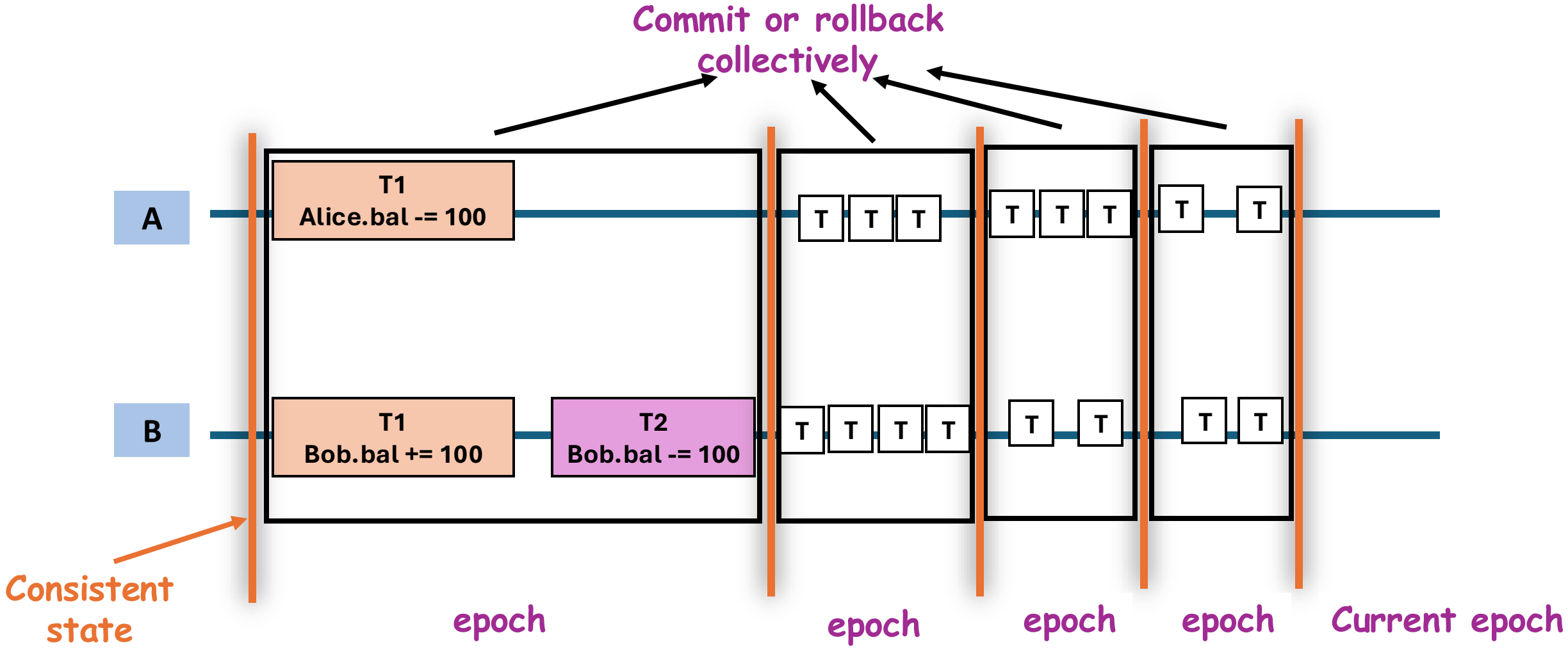
Solution#2: Epoch-based rollbacks



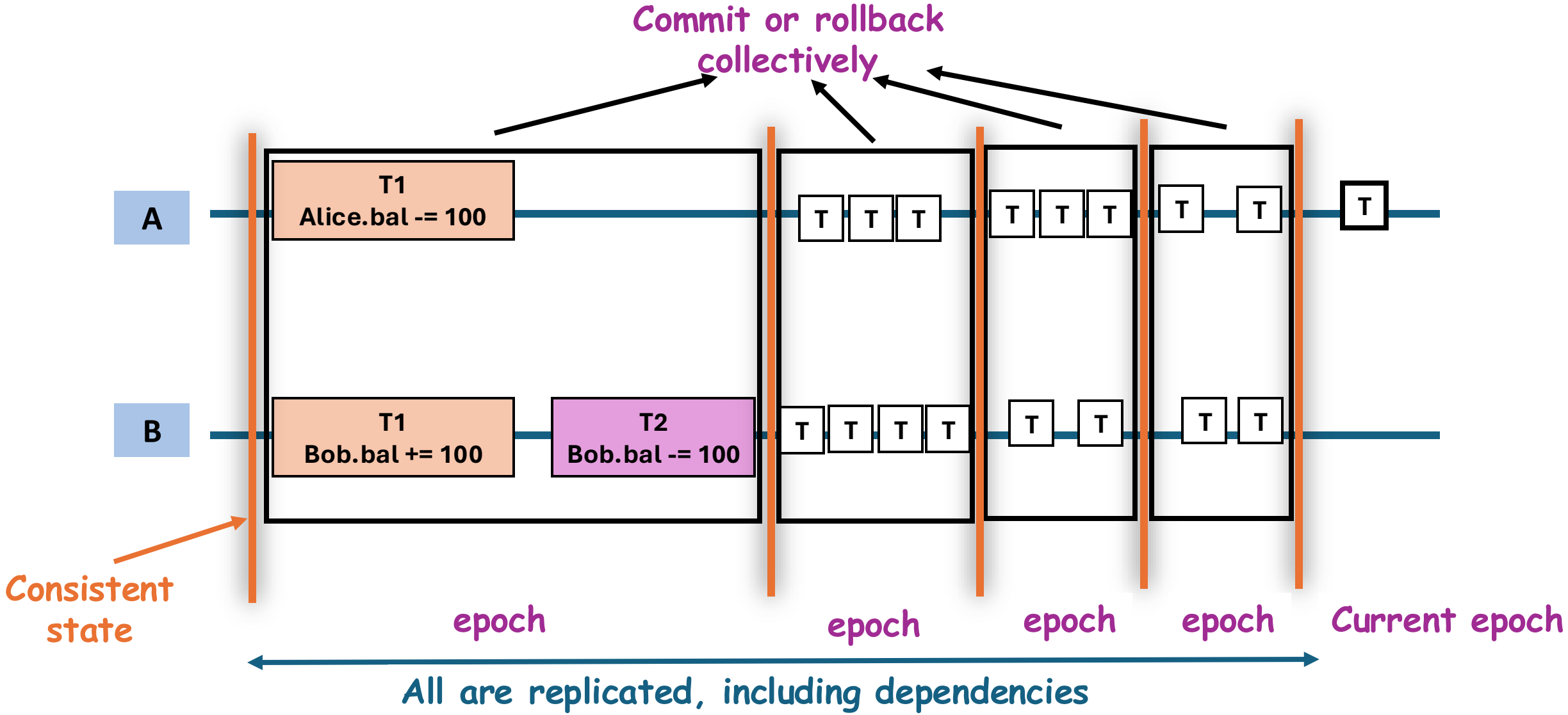
Solution#2: Epoch-based rollbacks



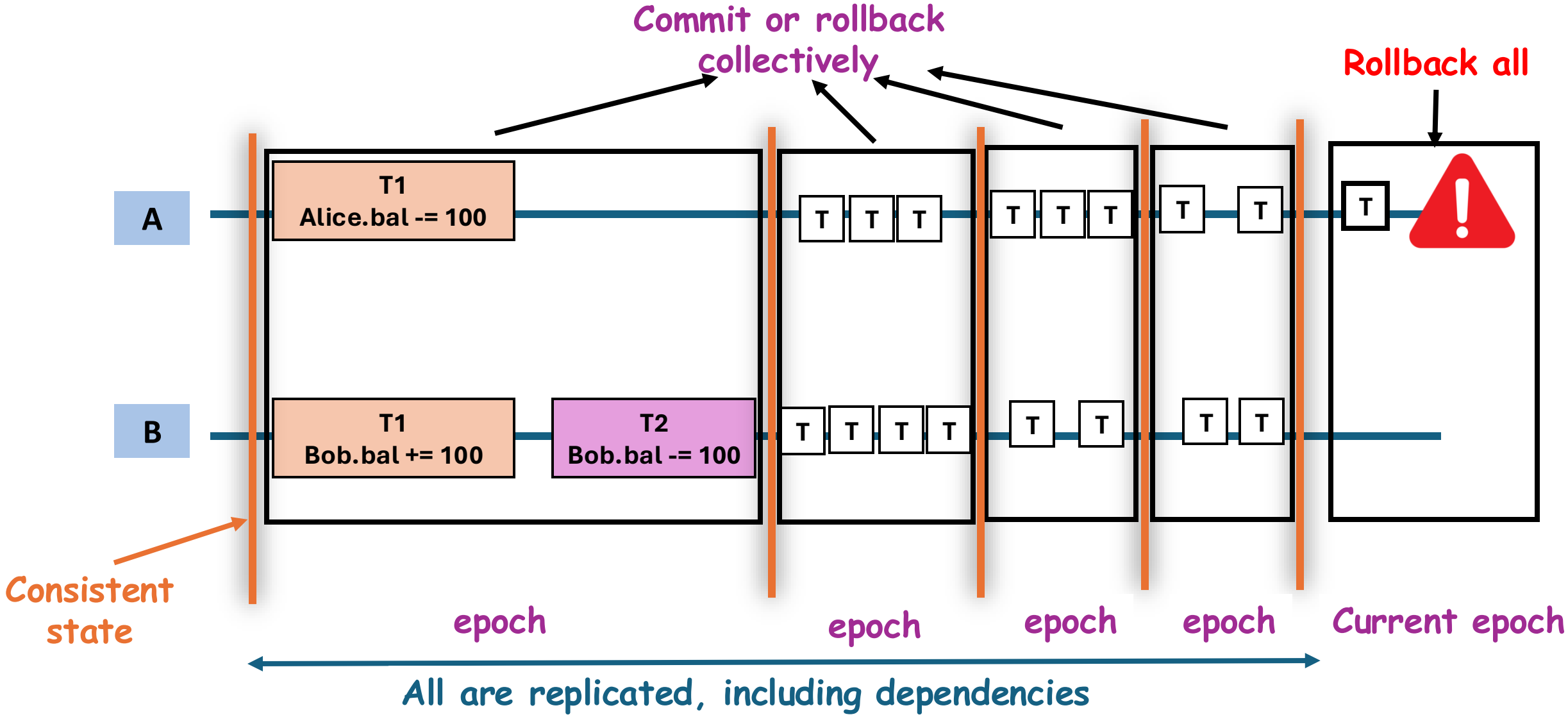
Solution#2: Epoch-based rollbacks



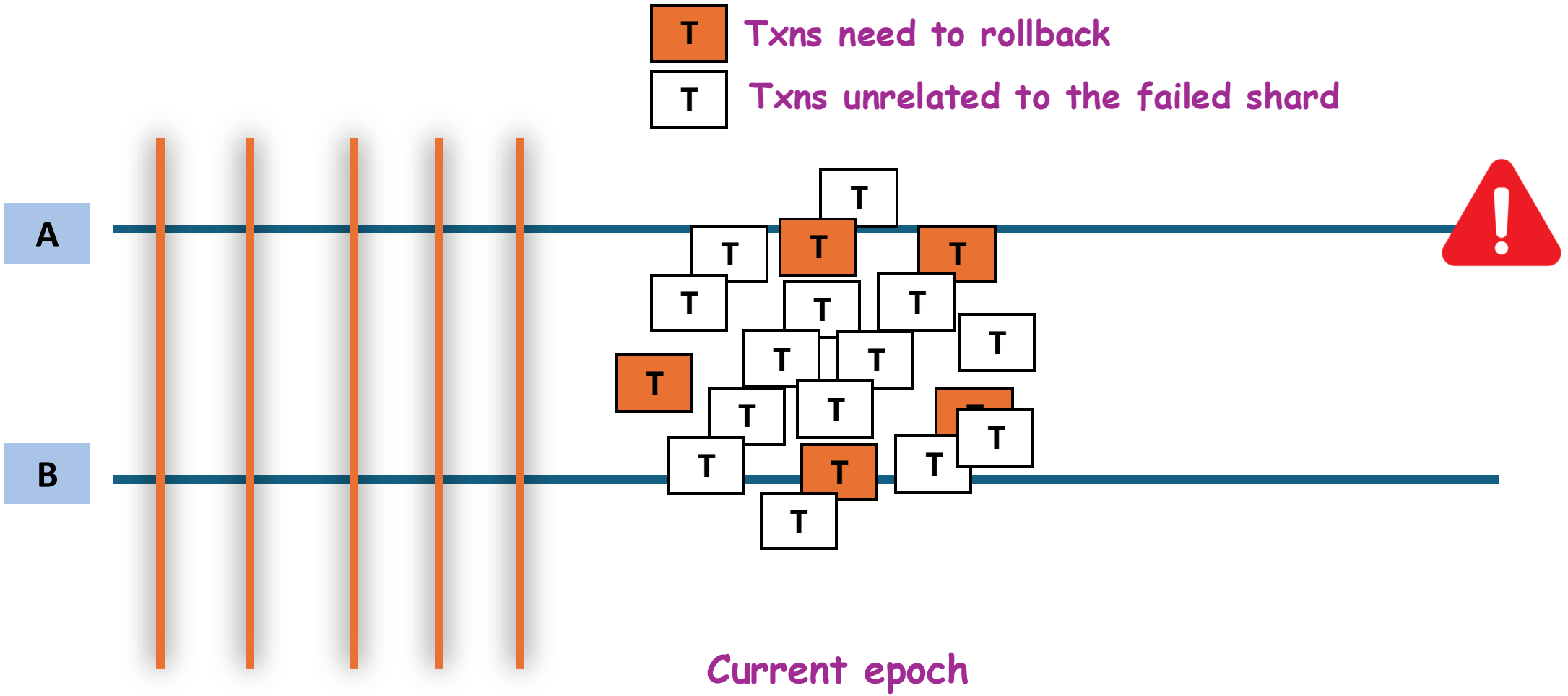
Solution#2: Epoch-based rollbacks



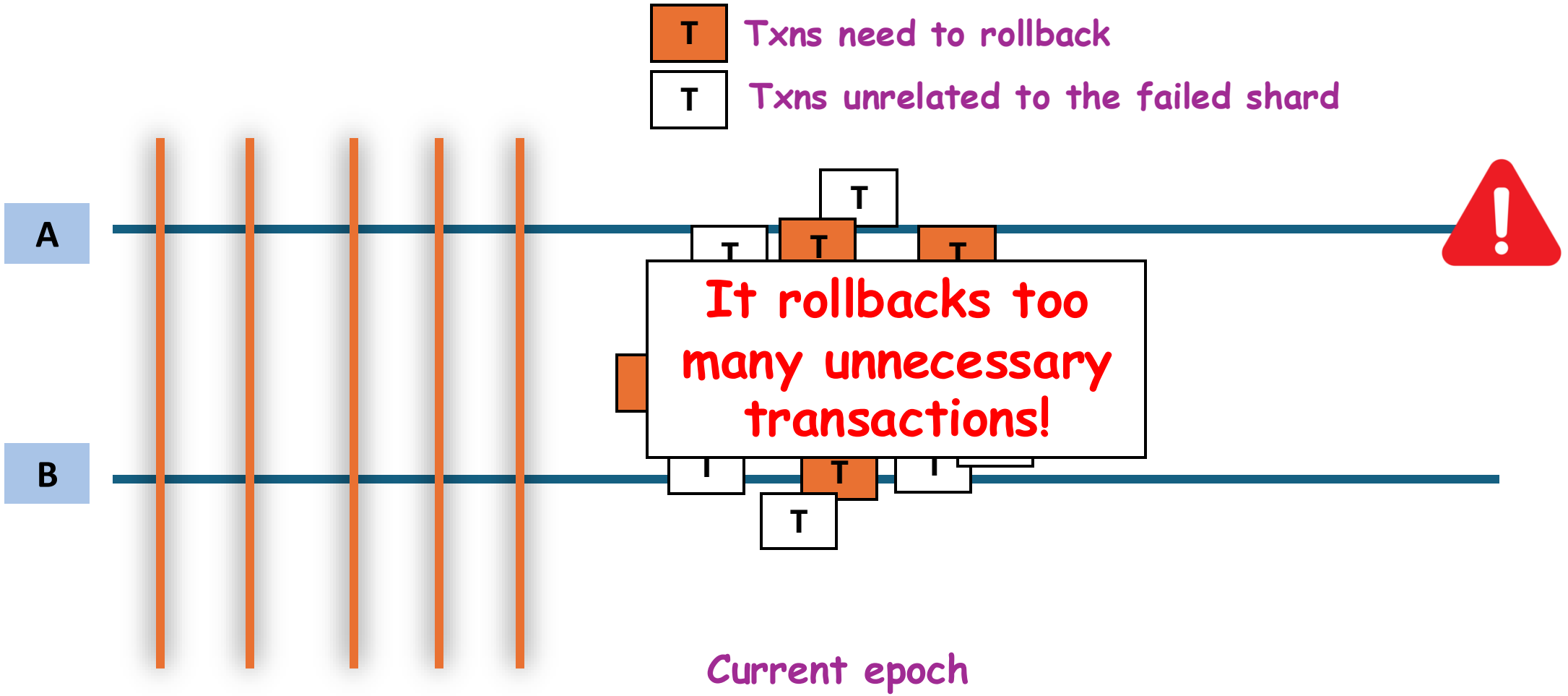
Solution#2: Epoch-based rollbacks



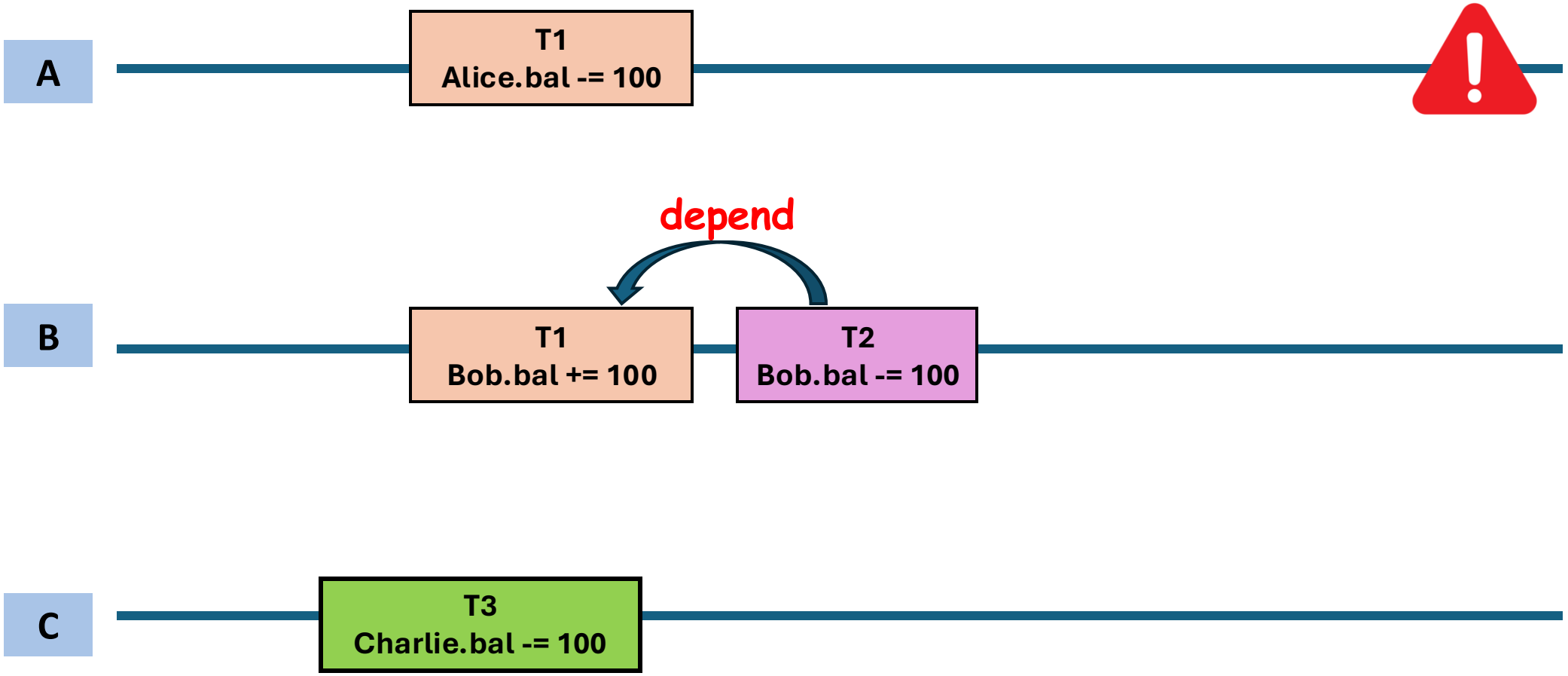
Challenge#3: Excessive rollbacks



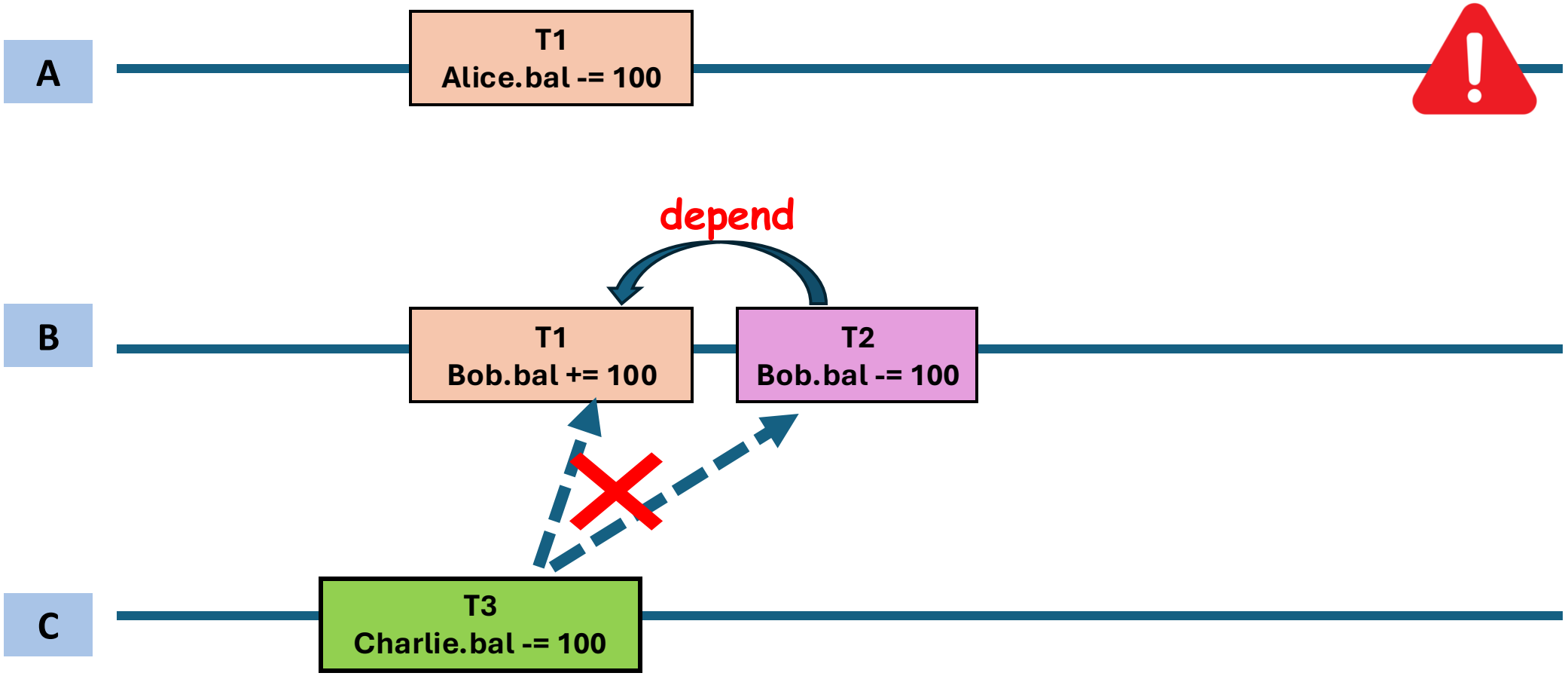
Challenge#3: Excessive rollbacks



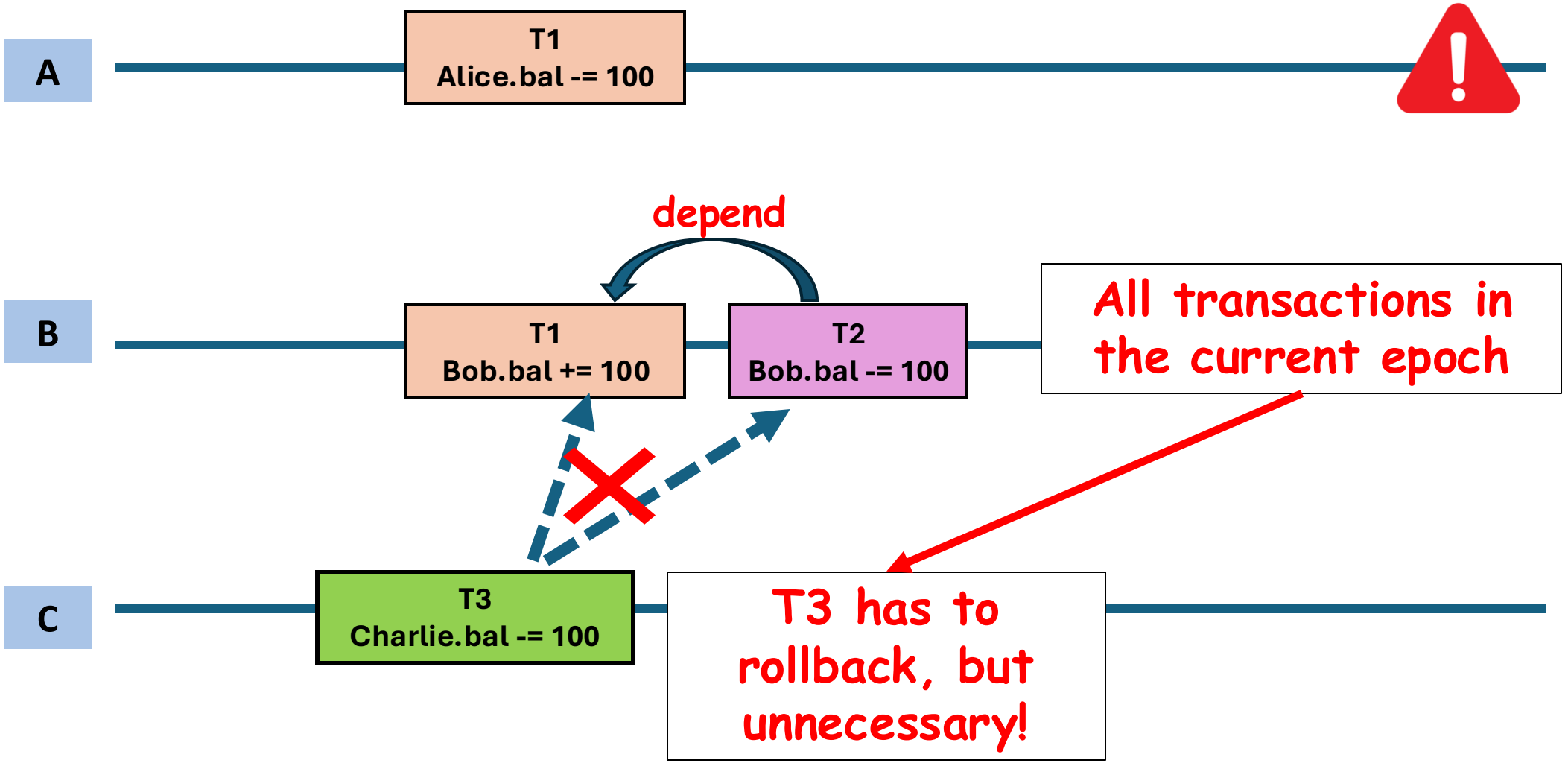
Challenge#3: Excessive rollbacks



Challenge#3: Excessive rollbacks

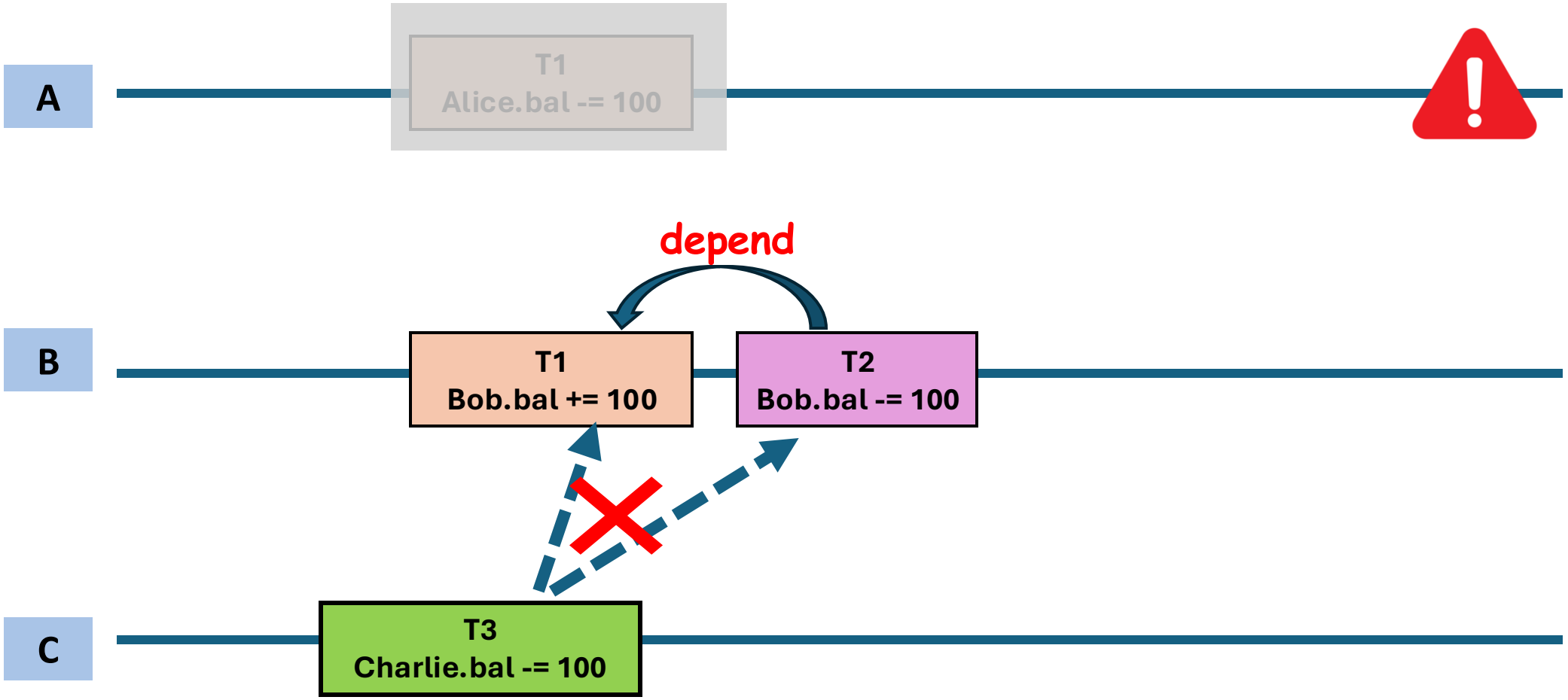


Challenge#3: Excessive rollbacks



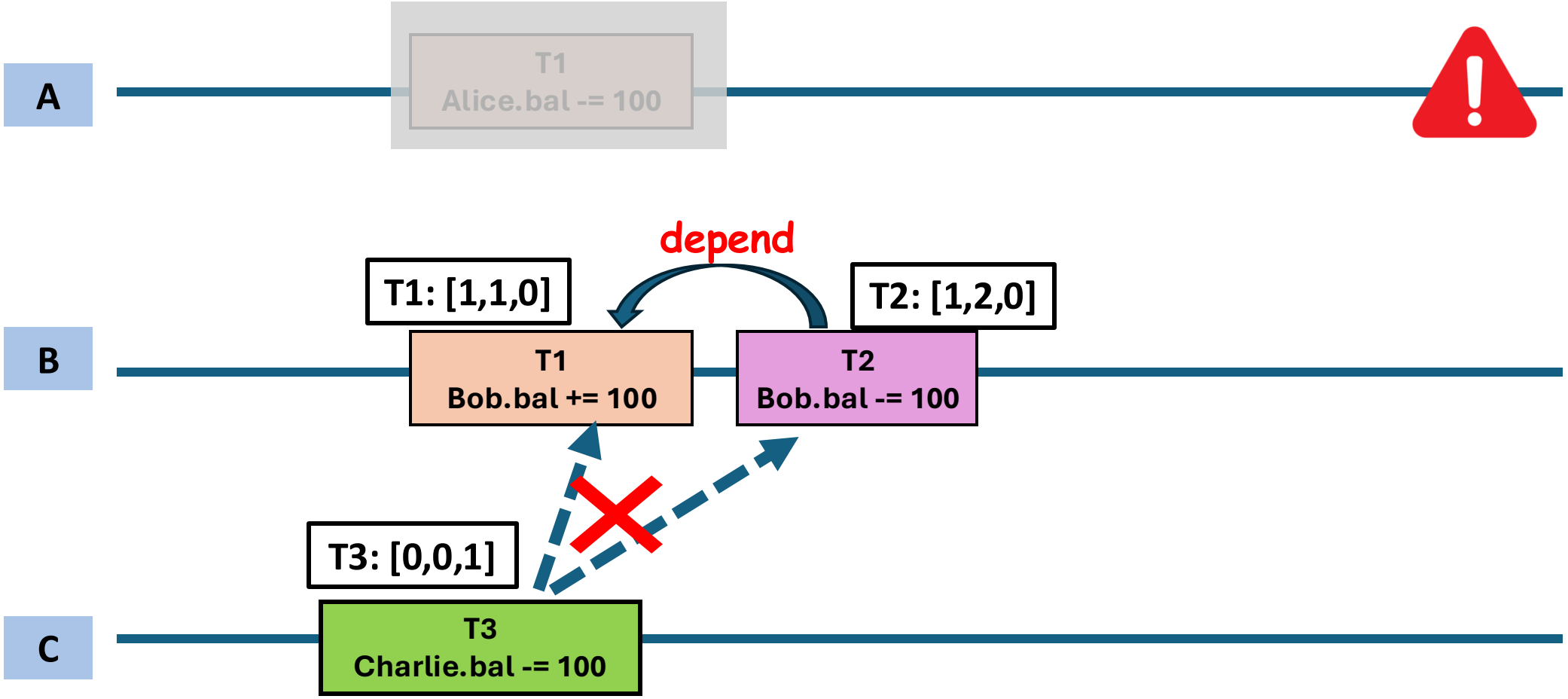
Solution#3: Selective rollbacks

Insight: Vector clock to track dependencies



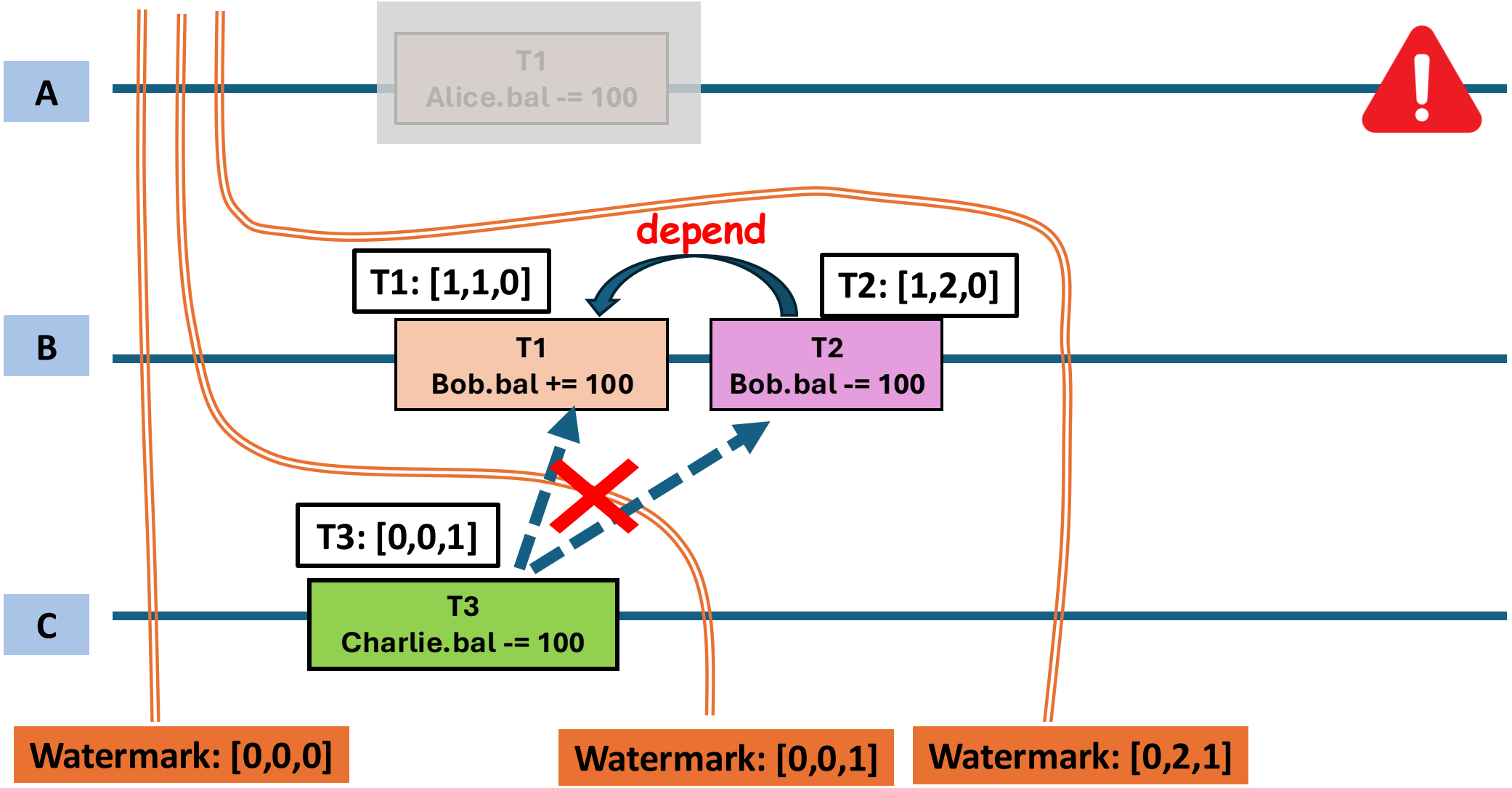
Solution#3: Selective rollbacks

Insight: Vector clock to track dependencies



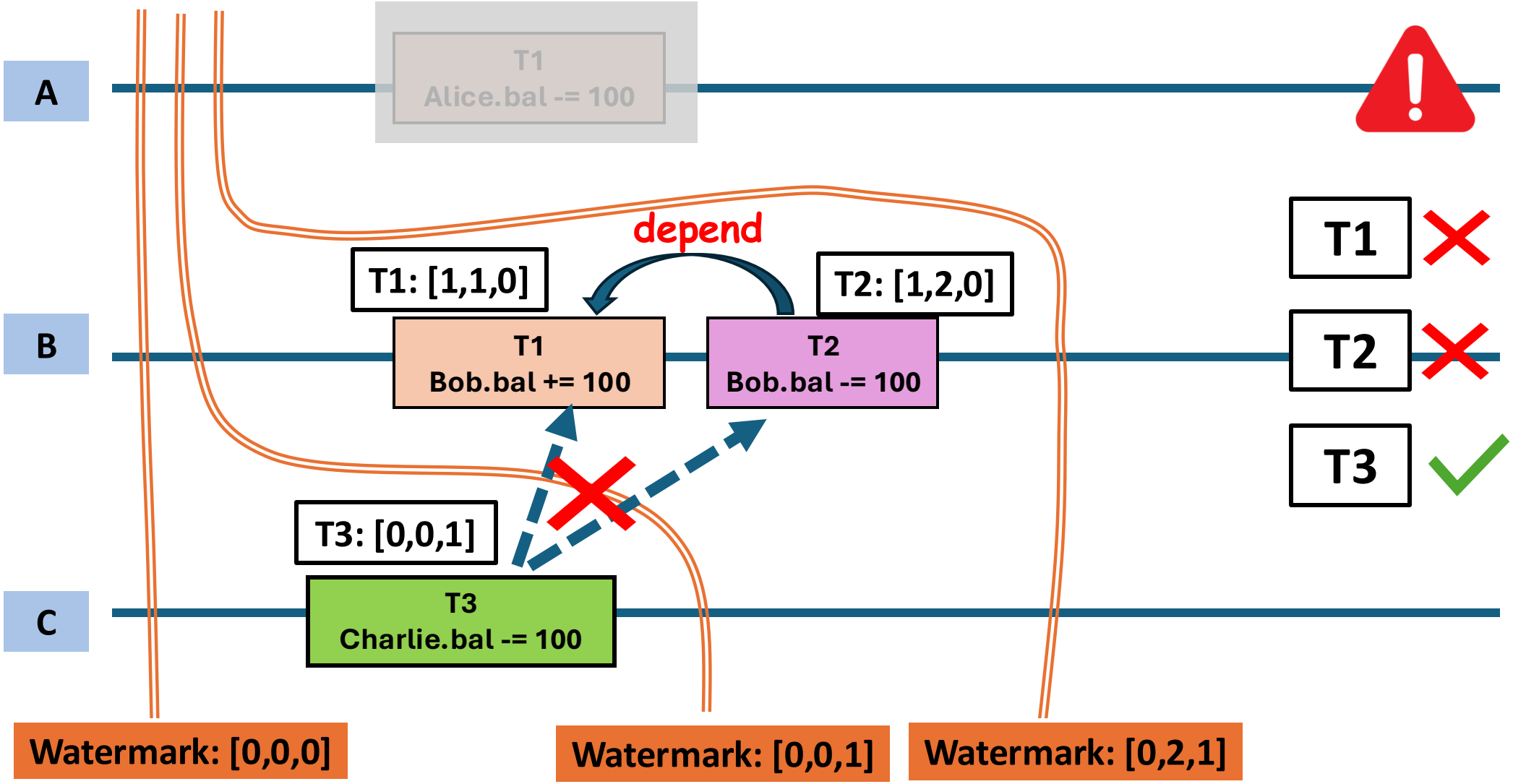
Solution#3: Selective rollbacks

Insight: Vector clock to track dependencies



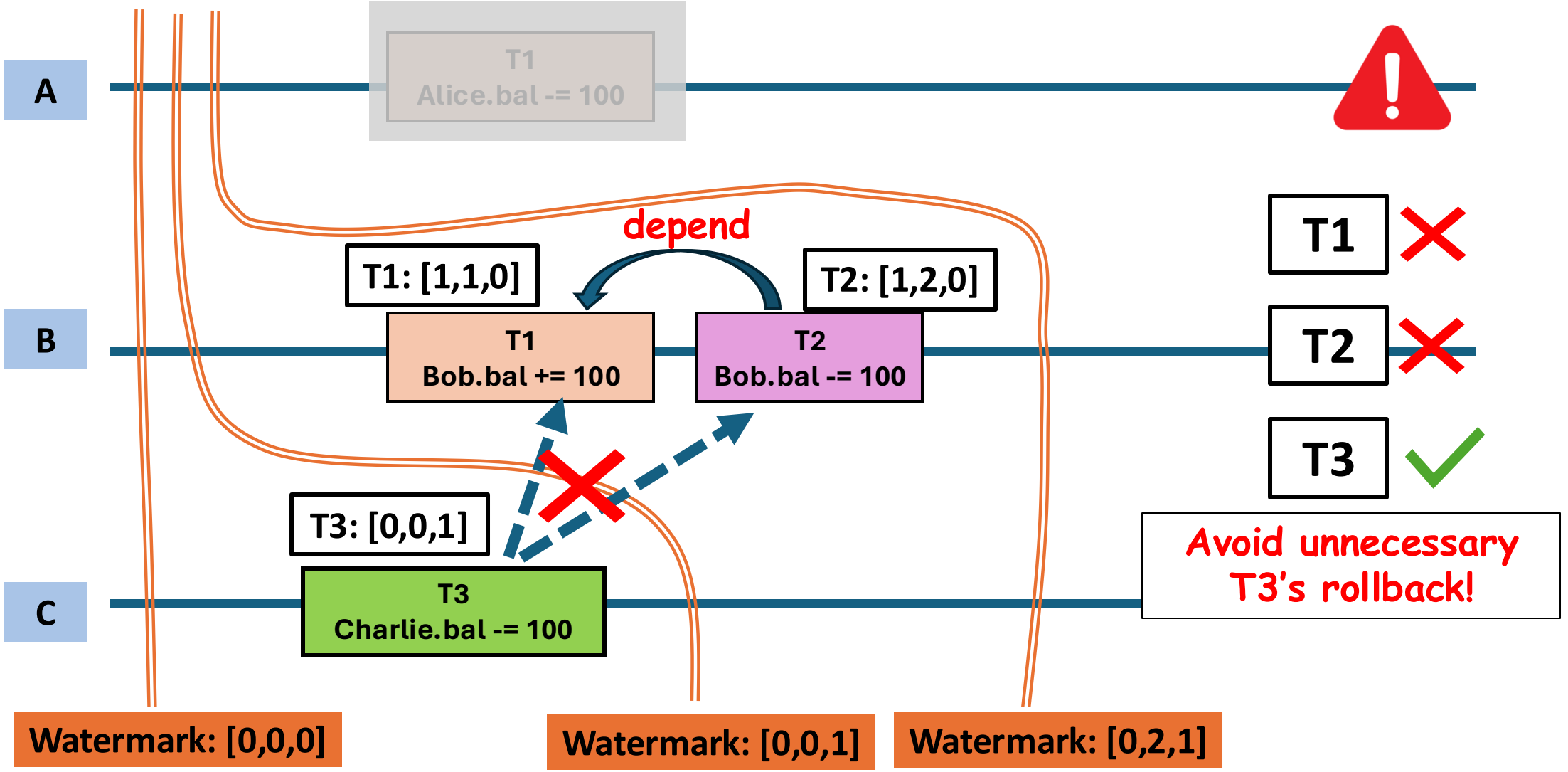
Solution#3: Selective rollbacks

Insight: Vector clock to track dependencies



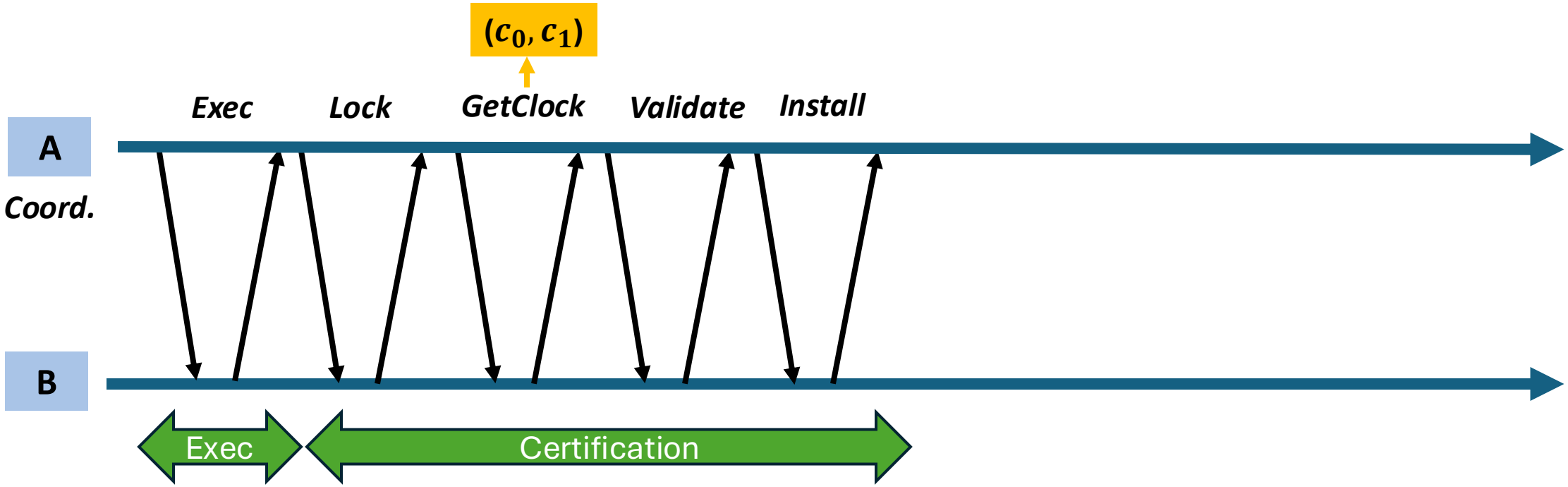
Solution#3: Selective rollbacks

Insight: Vector clock to track dependencies

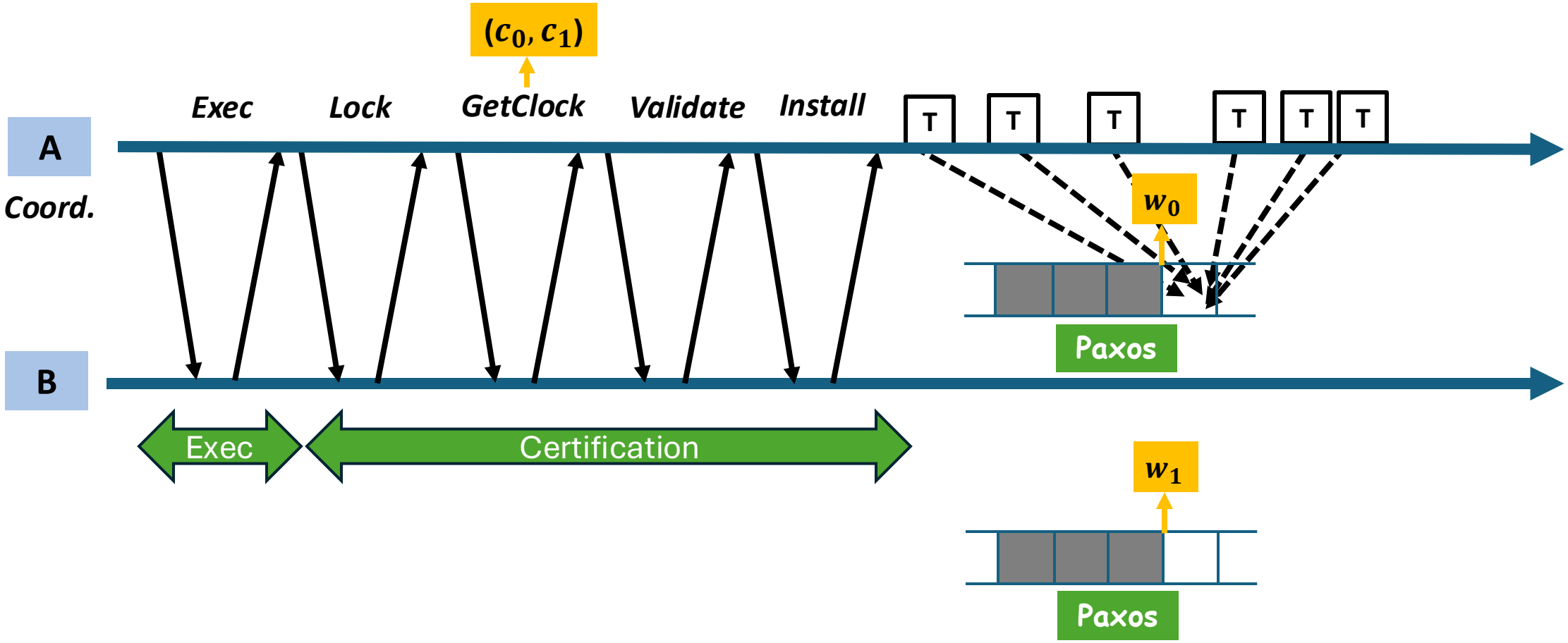


Mako: A new design for geo-replicated transactions

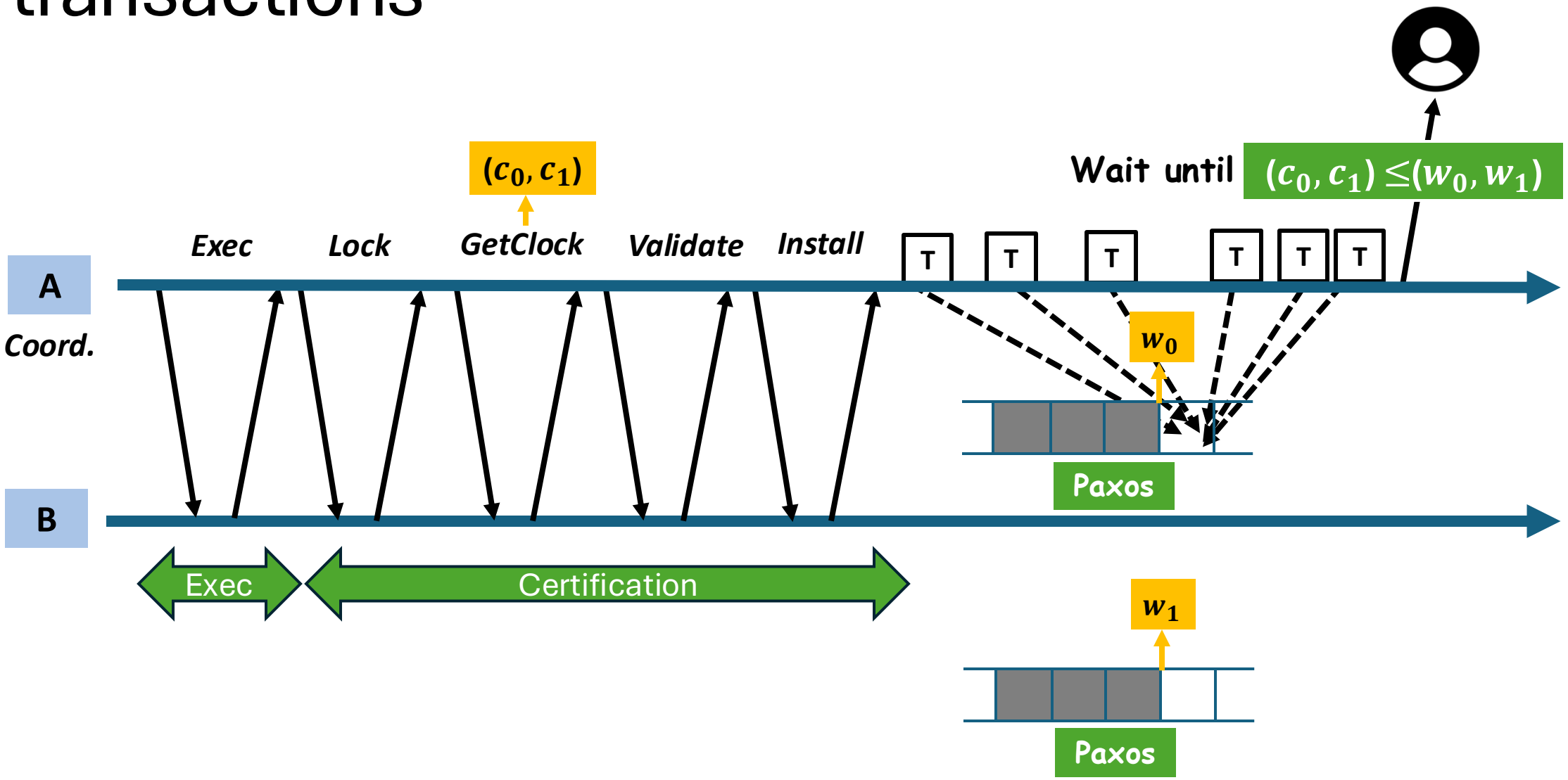
Mako: A new design for geo-replicated transactions



Mako: A new design for geo-replicated transactions



Mako: A new design for geo-replicated transactions

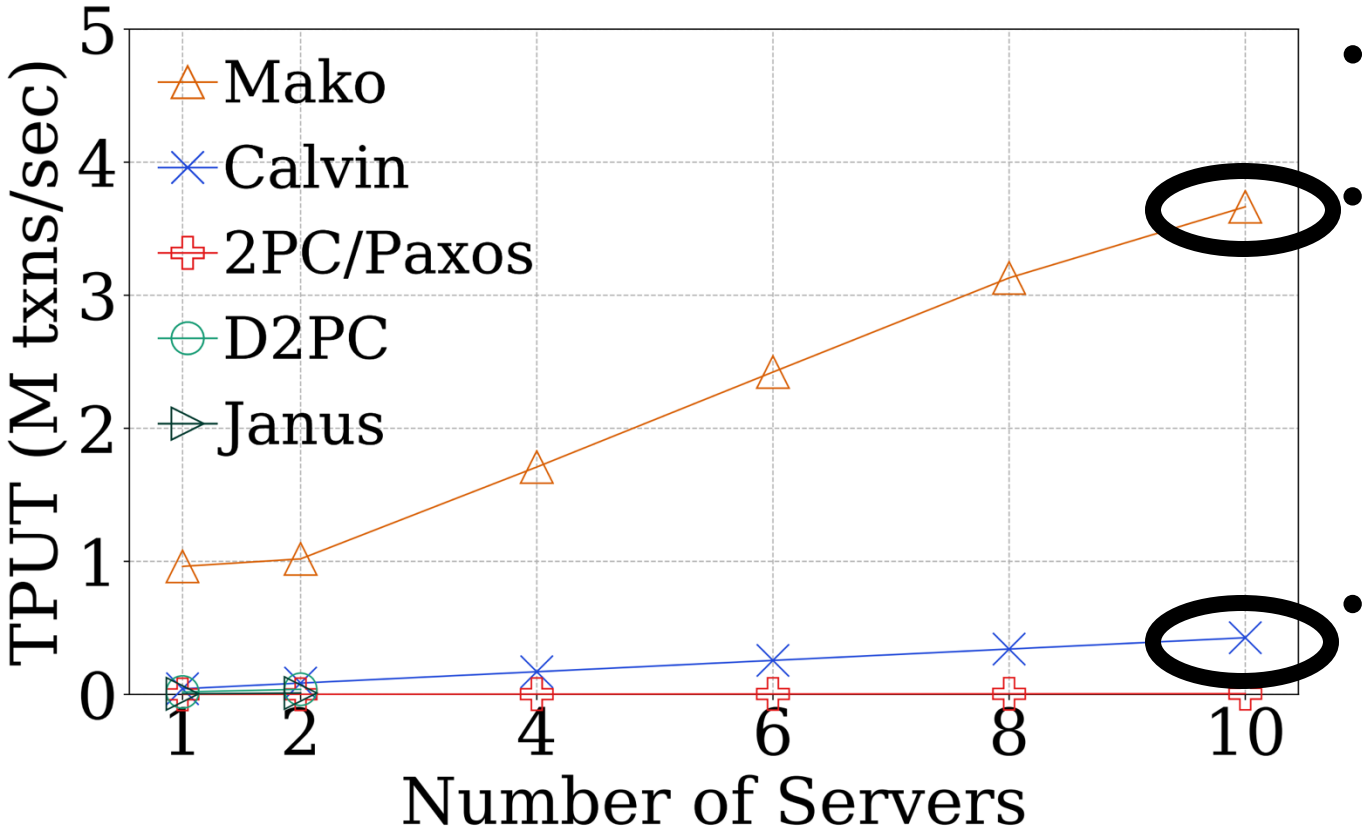


Evaluation

- Implementation
  - Built on: Silo [SOSP'13], Janus [OSDI'16] and eRPC [NSDI'19]
  - ~10k new lines of C++
- Azure
  - Simulate 3 DCs with 50ms injected latency[1]
  - Each datacenter: 10 servers; each server has 24 worker threads
- Benchmarks
  - Complex TPC-C benchmark with its default configuration
  - Microbenchmark with several RW operations

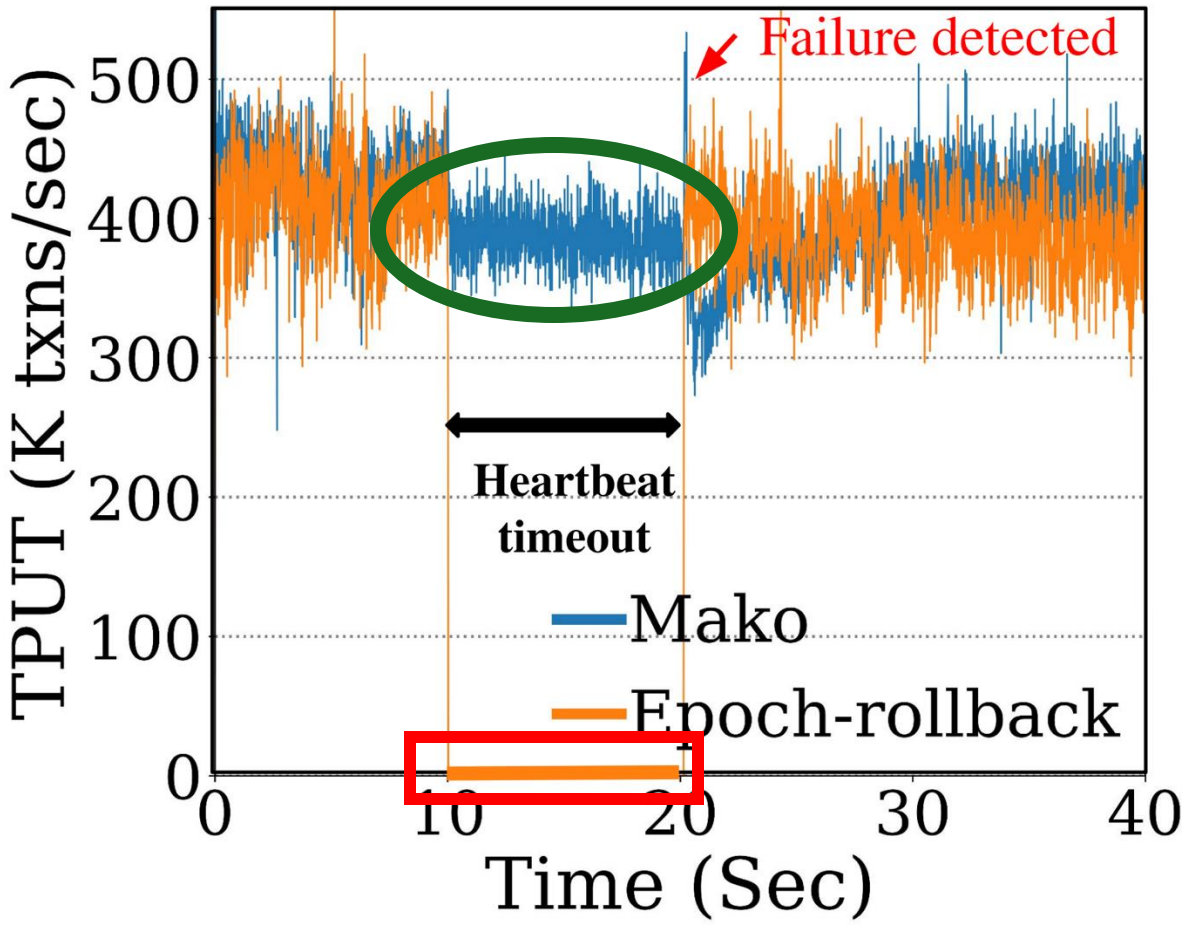
[1] We inject latency instead of deploying in multiple datacenters since we were limited by Azure quotas.

Scalability and its geo-replicated baselines



- **Mako**: 3.66M TPS
- **Combined cc. and rep.**
  - 2PC/Paxos, Janus [OSDI'16], D2PC [VLDB'24]
  - Orders of magnitude slower
- **Decoupled cc. and rep.**
  - Calvin [SIGMOD'12]
  - 8.6x slower

A single shard failure



One healthy shard

- **Epoch-rollback:** an epoch-based solution
- Kill a shard server at 10sec
- Heartbeat timeout: 10sec

Mako:
save most of transactions during heartbeat timeout!

Epoch-rollback:
zero throughput during heartbeat timeout!

Latency experiments

Percentile	Mako	Janus	Calvin
10%	57 ms	50.3 ms	146 ms
50%	60 ms	50.5 ms	166 ms
90%	64 ms	50.7 ms	202 ms
95%	65 ms	50.8 ms	206 ms
99%	66 ms	51.3 ms	212 ms

- A light workload on Microbenchmark with just 1 replicated shard

Mako median latency:
60 ms = ~50 ms WAN + 3.5 ms batching + 6.5 ms watermark advancement

Conclusion

- Mako decouples replication from the execution path
- Mako uses vector clock/watermarks to selectively roll back transactions
- Mako outperforms geo-replicated baselines, and saves most of transactions during failures

Thank you!