Leveraging Lock Contention to Improve Transaction Applications

Cong Yan Alvin Cheung
University of Washington

Background

- Database transactions
    - Airline ticket reservation, banking, online shopping...

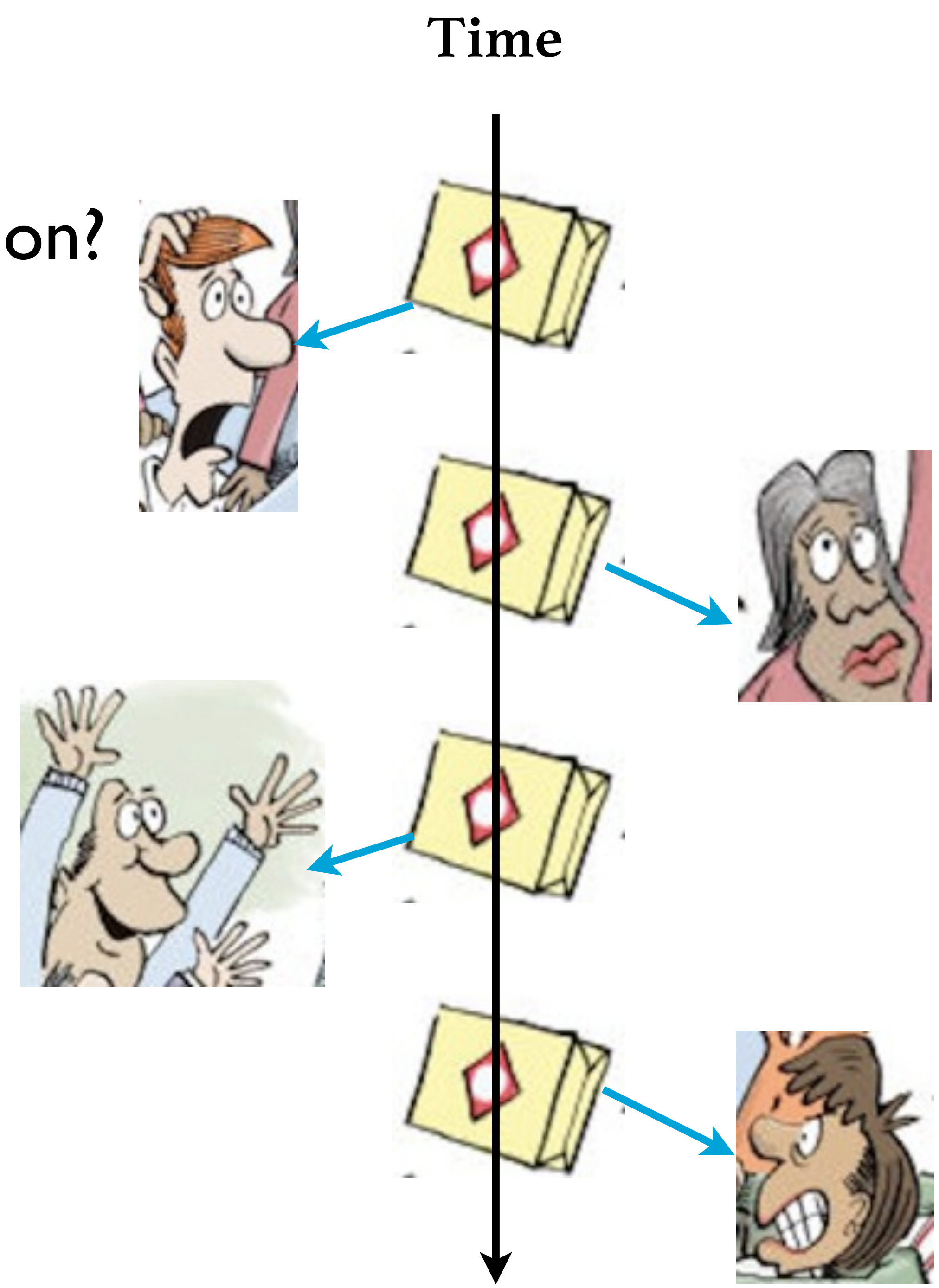
Background

- Database transactions
    - Airline ticket reservation, banking, online shopping...

"AND TO THINK, JUST THE OTHER DAY I WAS WORRIED ABOUT STRANGERS TOUCHING MY JUNK AT THE AIRPORT."

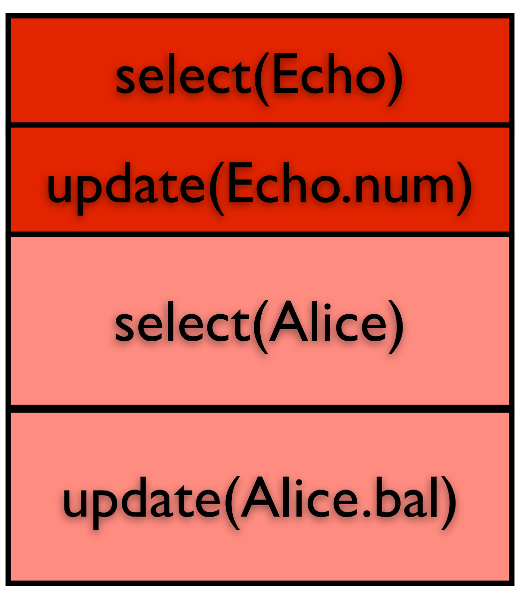
Background

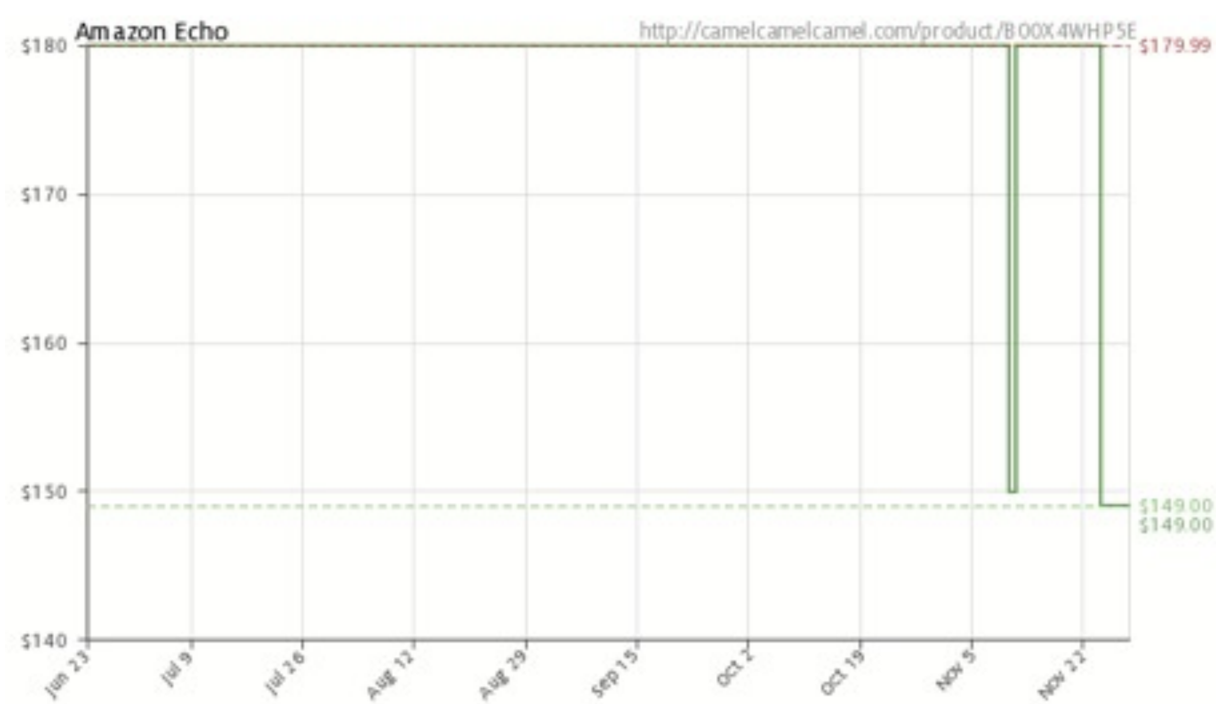
- Database transactions
  - Parallelism under high data contention? atomicity and consistency
  - Concurrency protocols:



Two-phase Locking

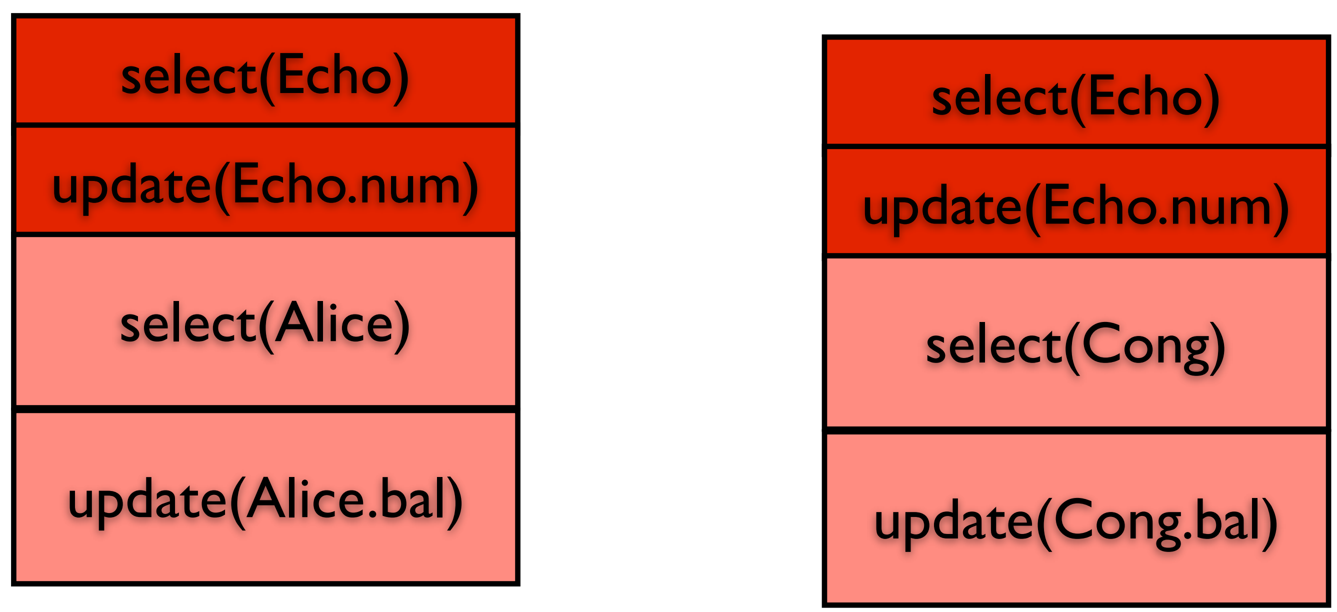
- Example transaction: online shopping
  - Alice(A) and Cong(C) buying Echo at the same time



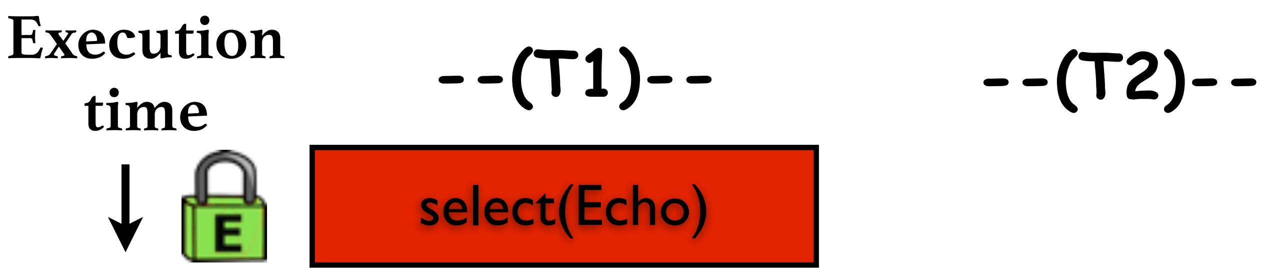


Two-phase Locking

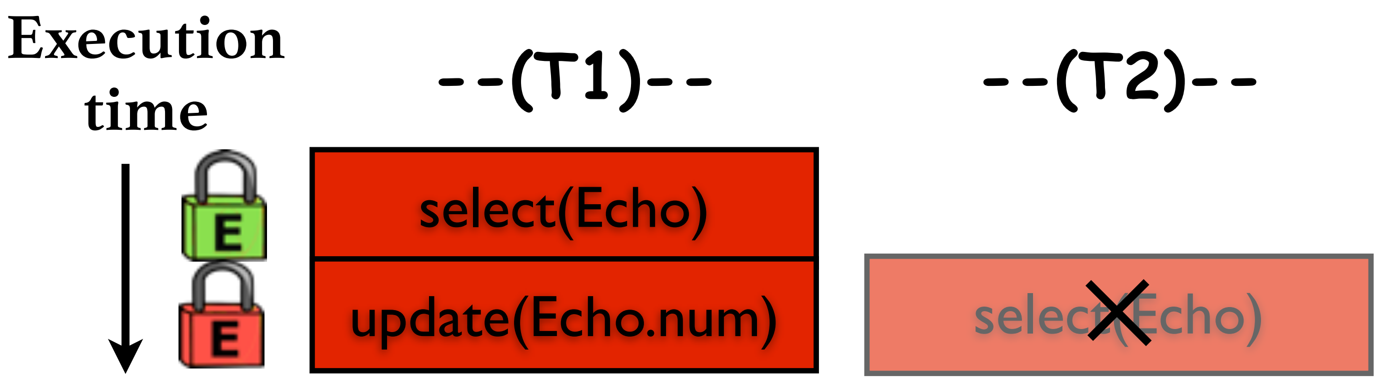
- Example transaction: online shopping
    - Alice(A) and Cong(C) buying Echo at the same time



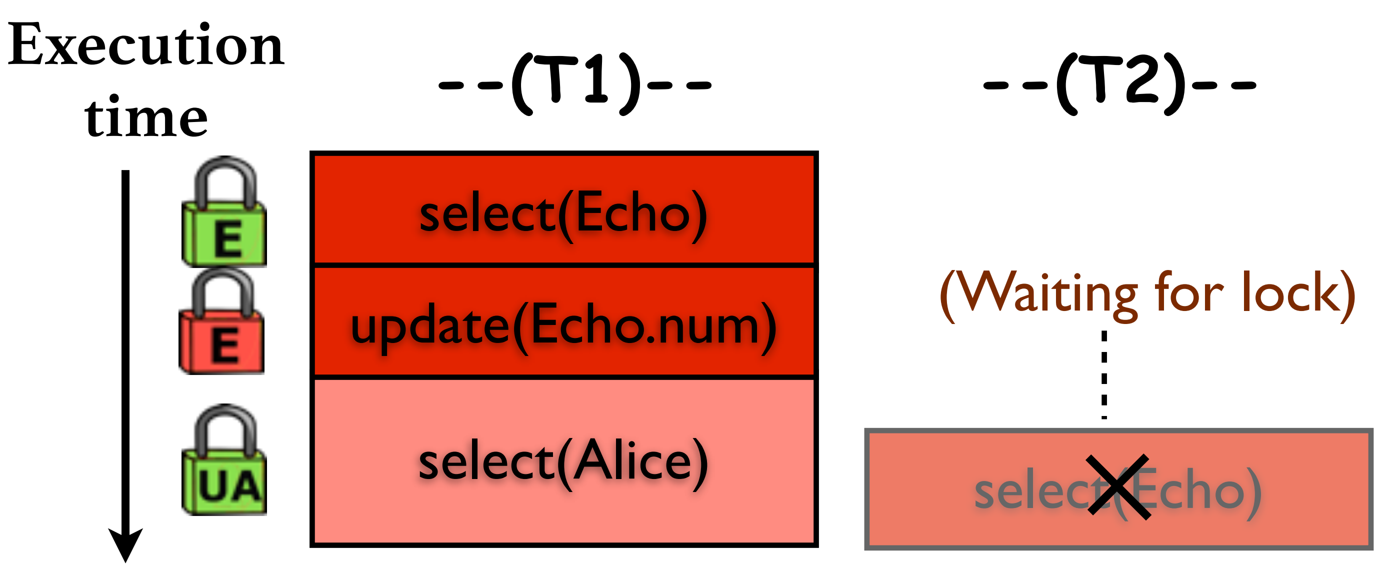
Two-phase Locking



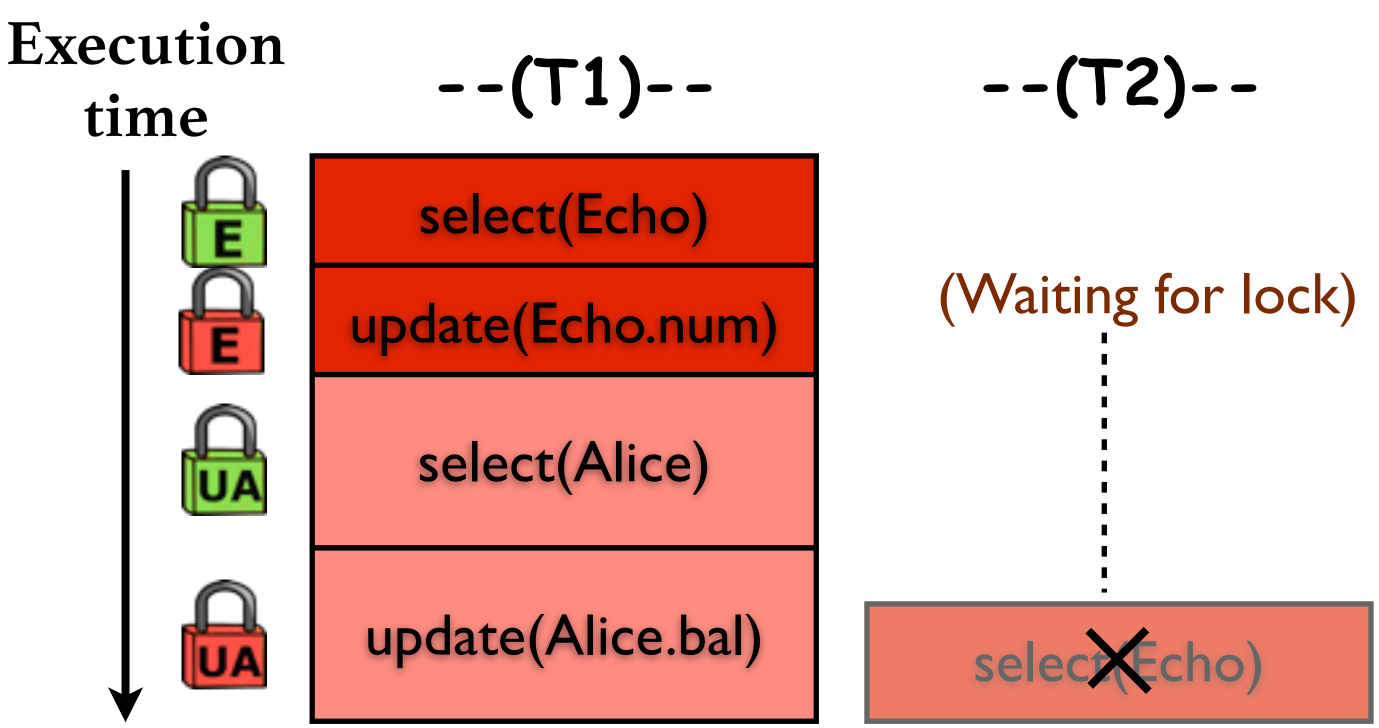
Two-phase Locking



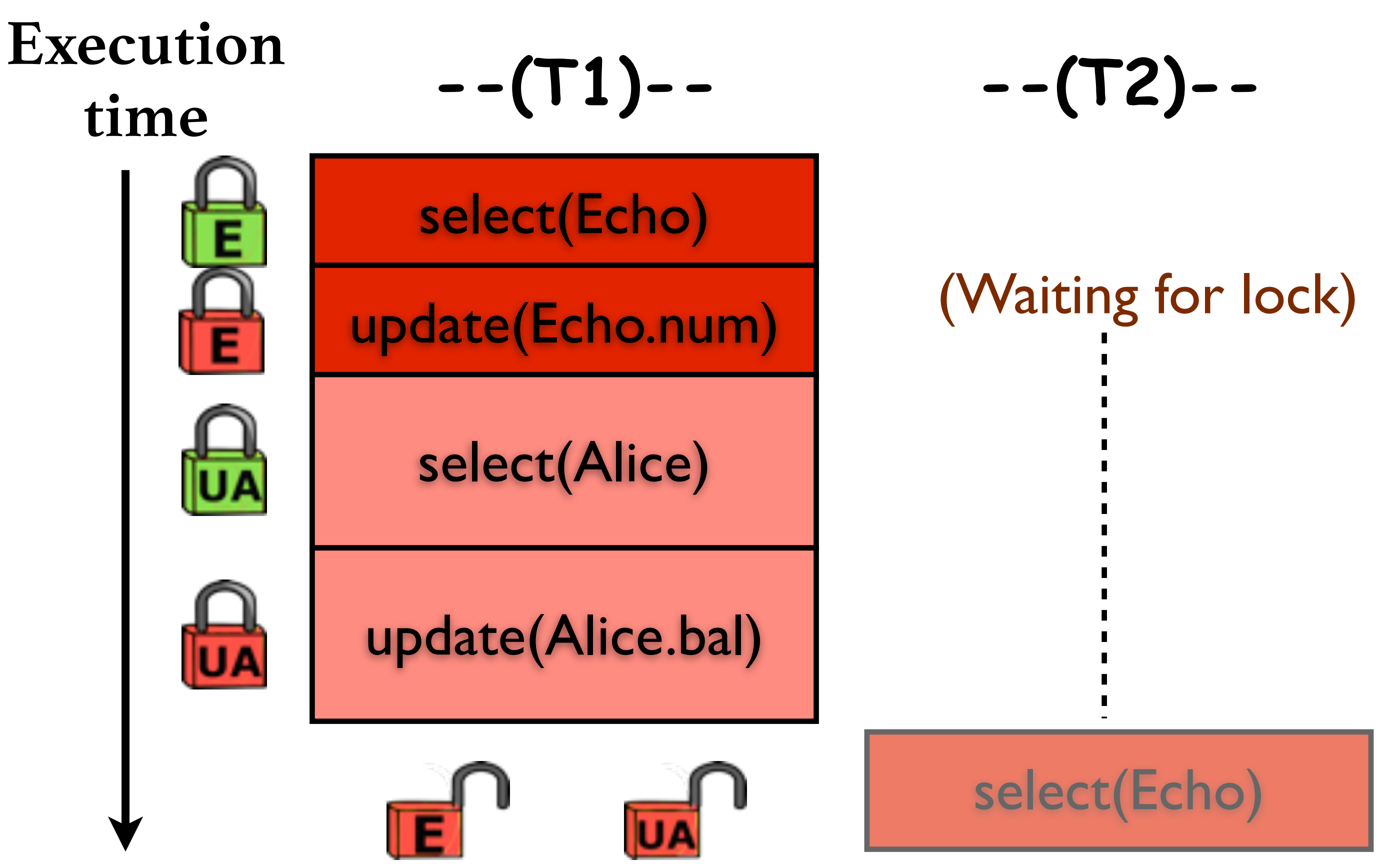
Two-phase Locking



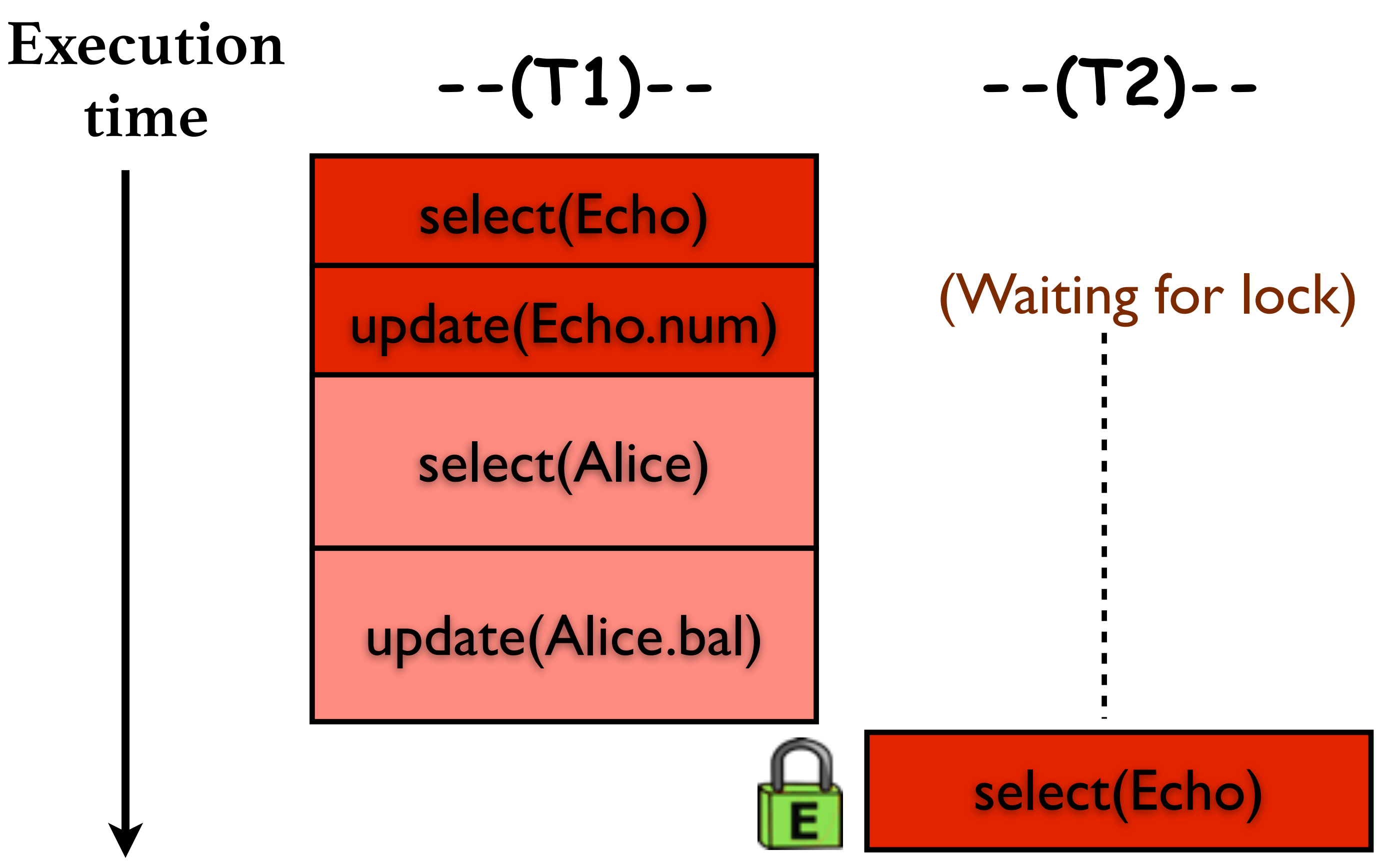
Two-phase Locking



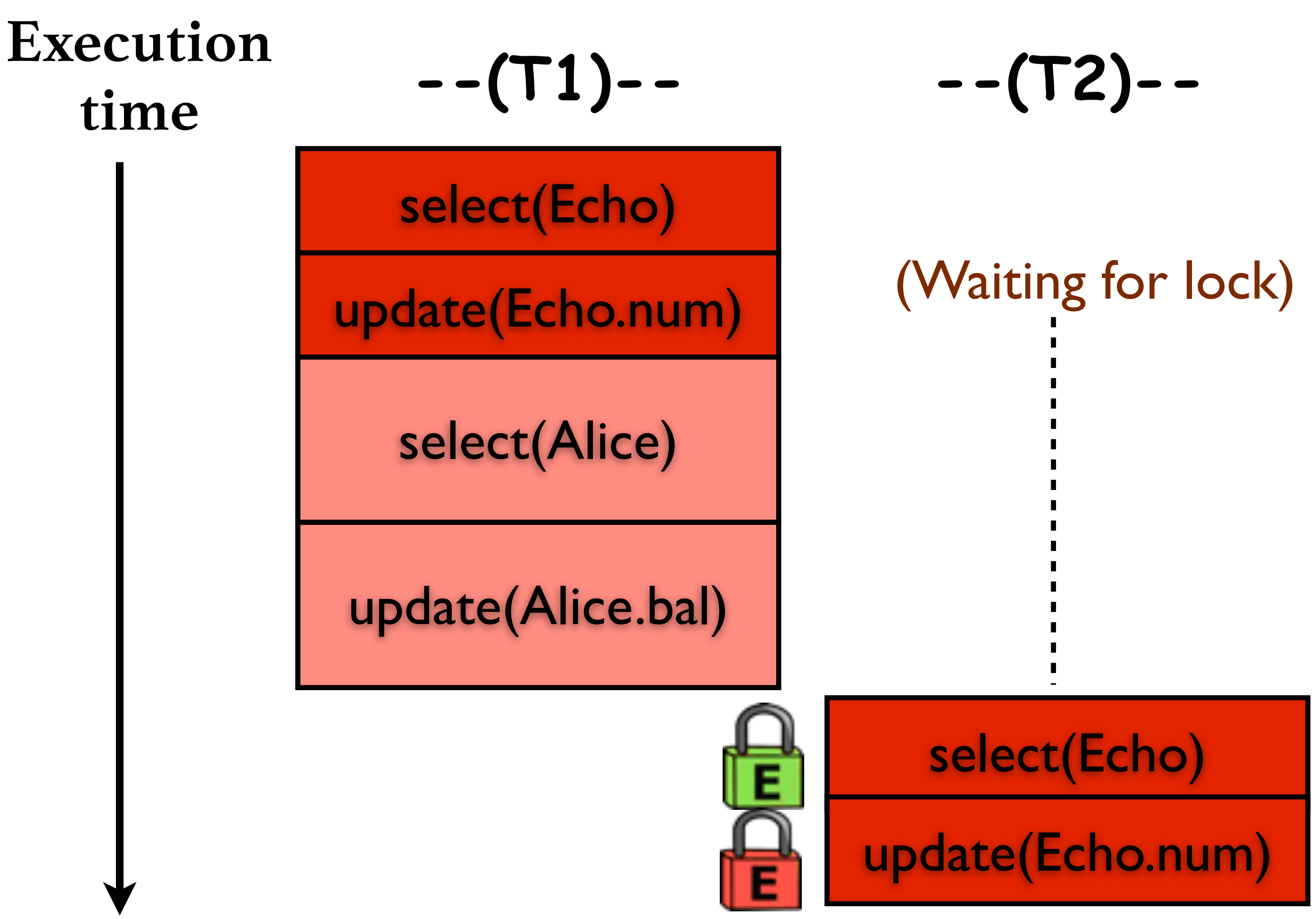
Two-phase Locking



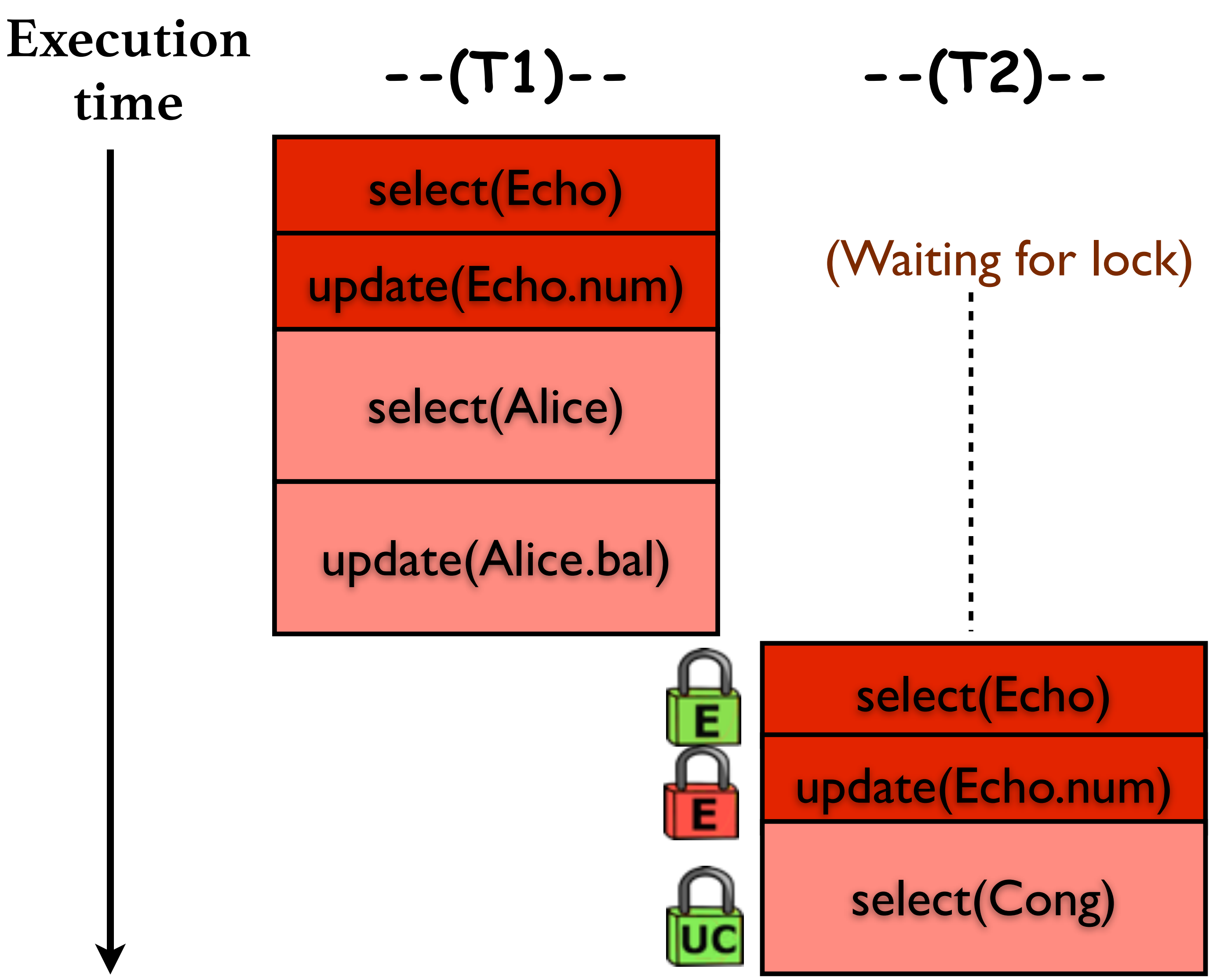
Two-phase Locking



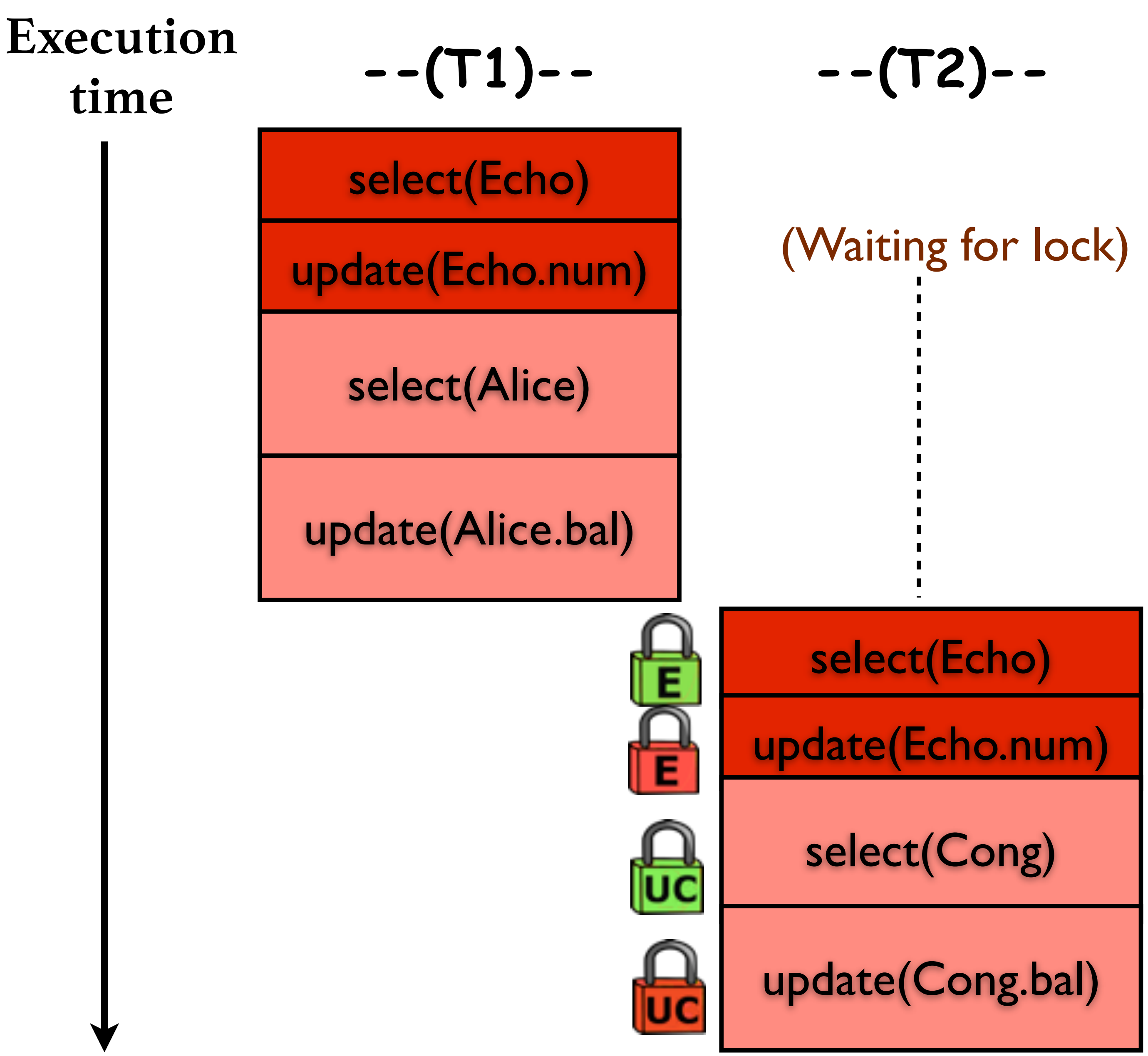
Two-phase Locking



Two-phase Locking



Two-phase Locking

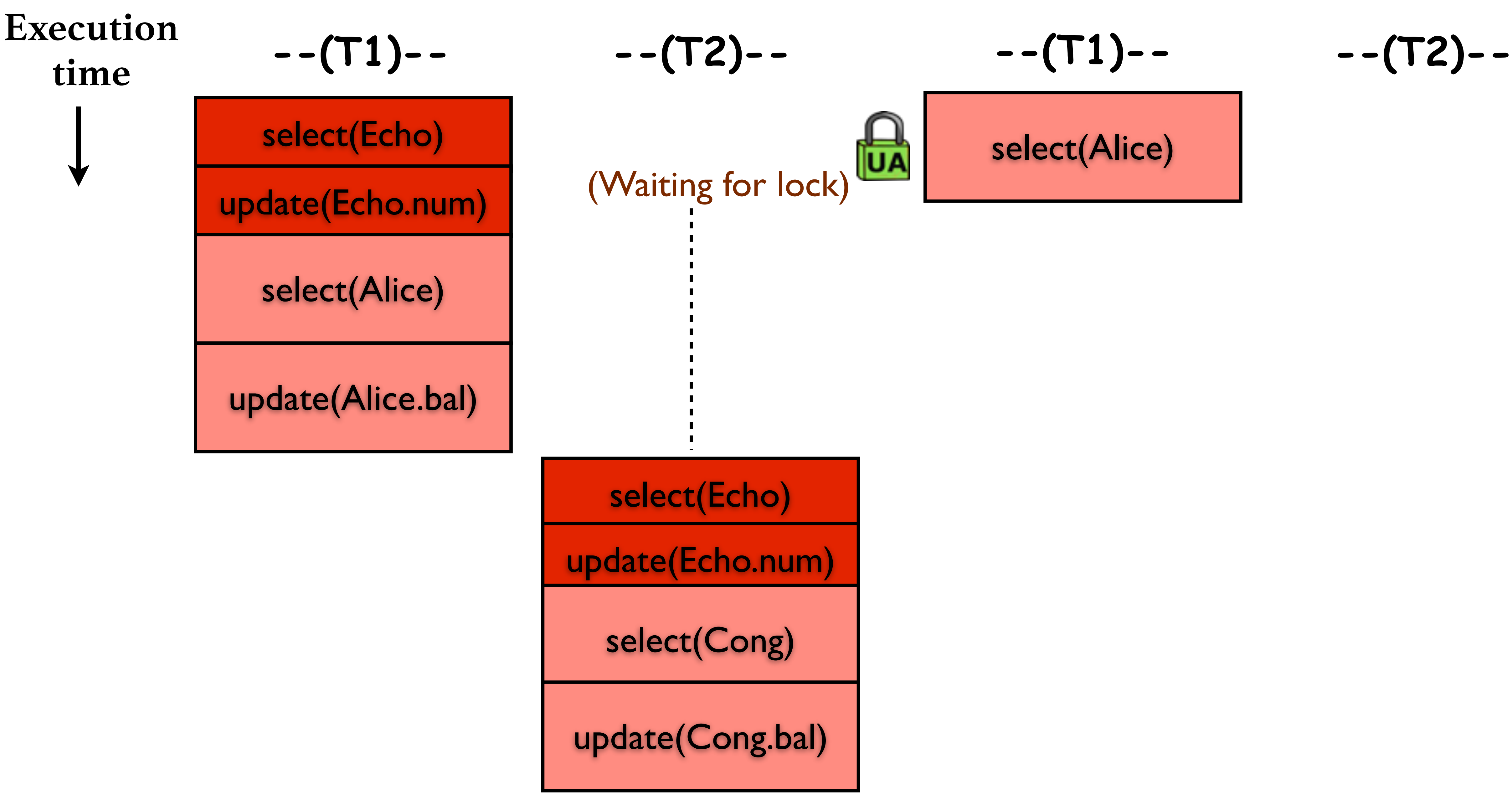


Two-phase Locking

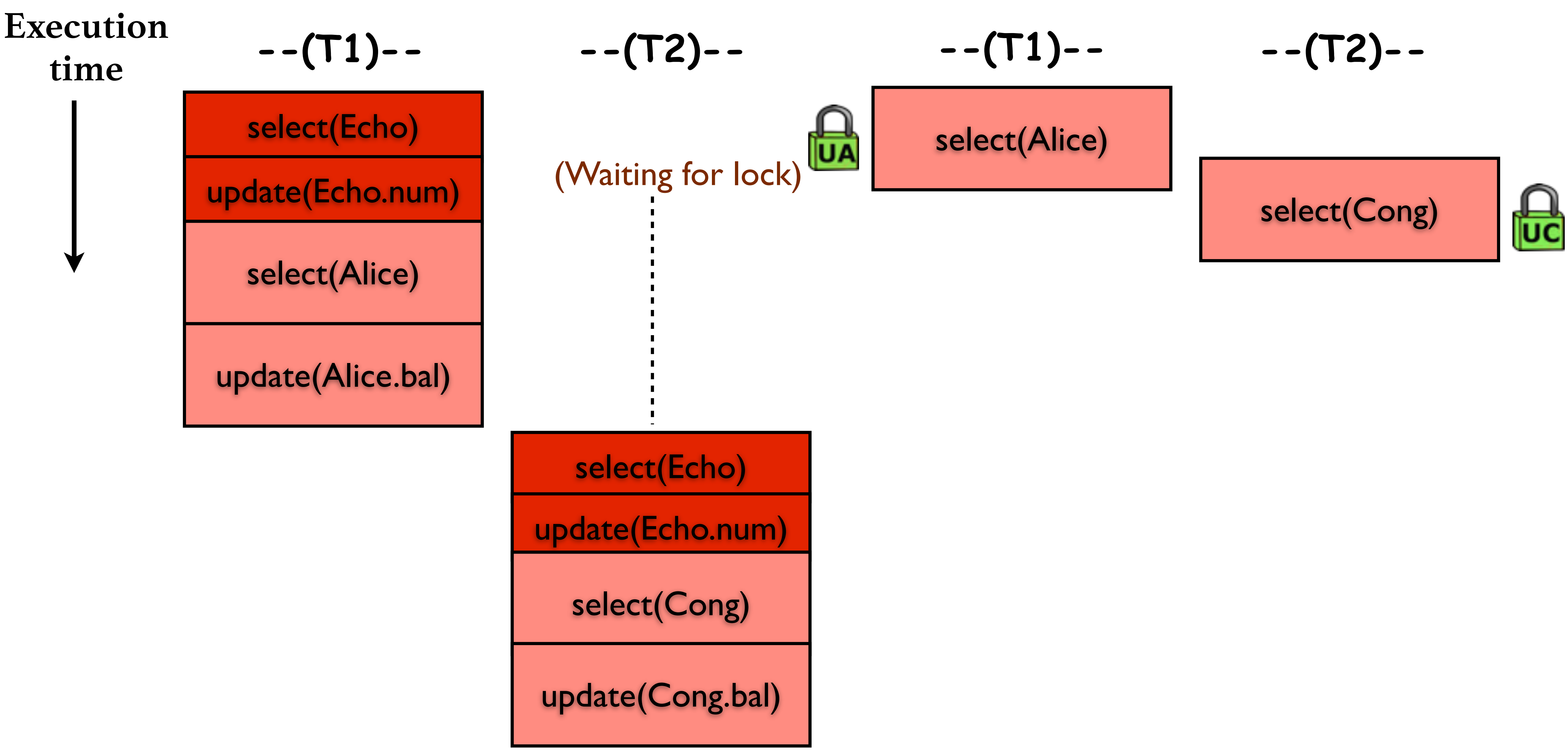
- Problem: serialization of all transactions
  - Changing the order of queries within transactions shortens lock waiting time
- Other concurrency control protocols: OCC, MVCC
  - 2PL is more efficient under high data contention(*)

*: *"Staring into the abyss, An Evaluation of Concurrency Control with One-thousand Cores", VLDB-14*

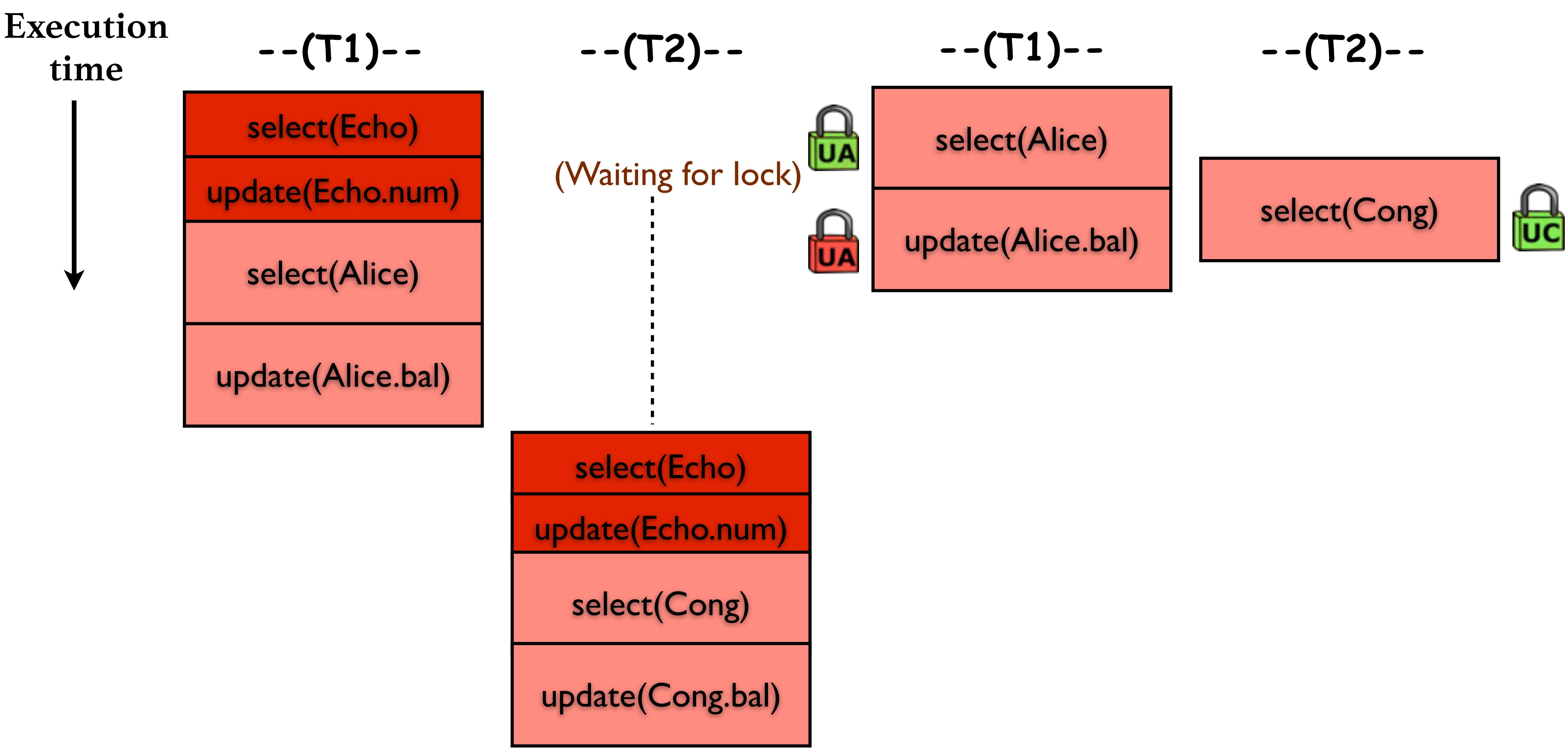
Shortening Lock Waiting



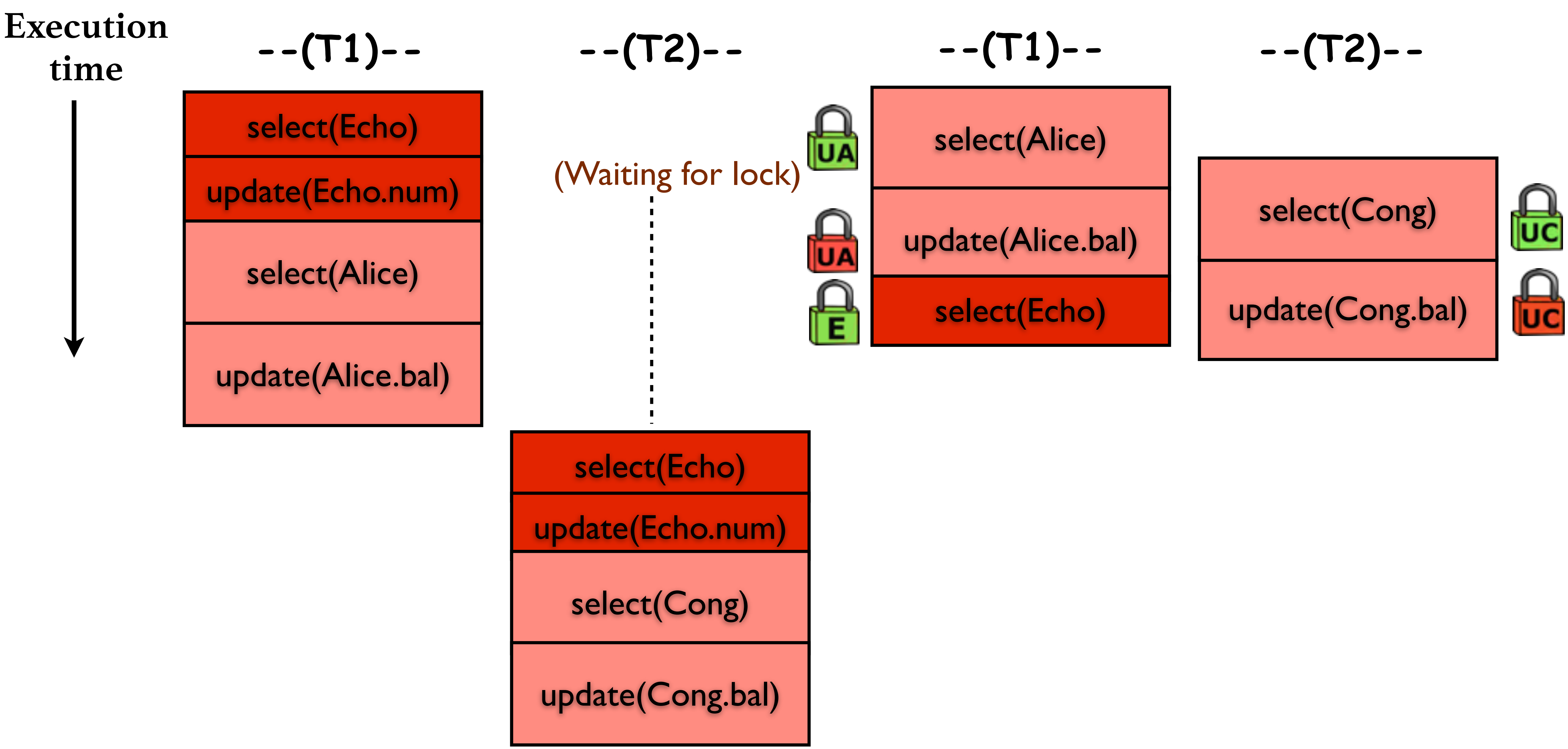
Shortening Lock Waiting



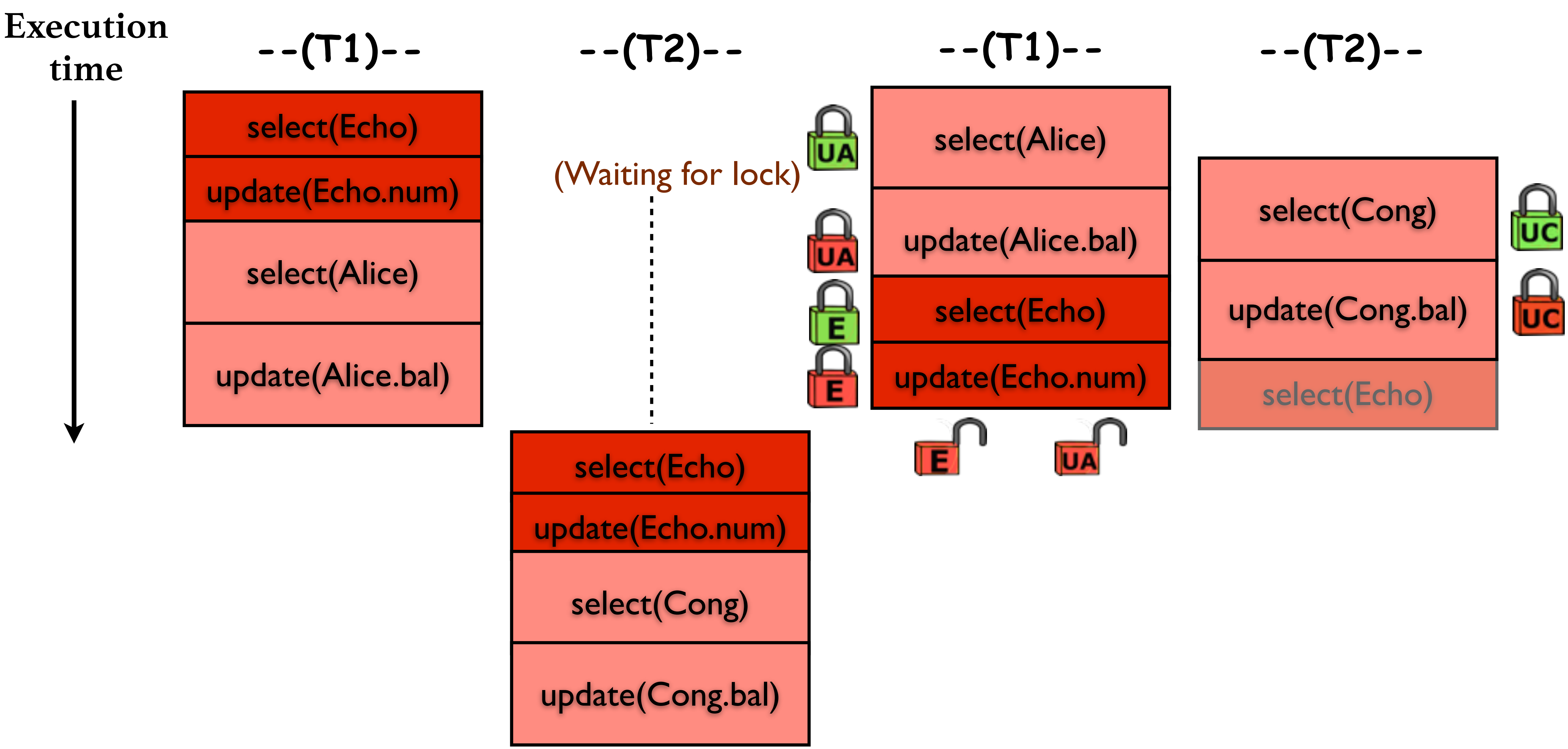
Shortening Lock Waiting



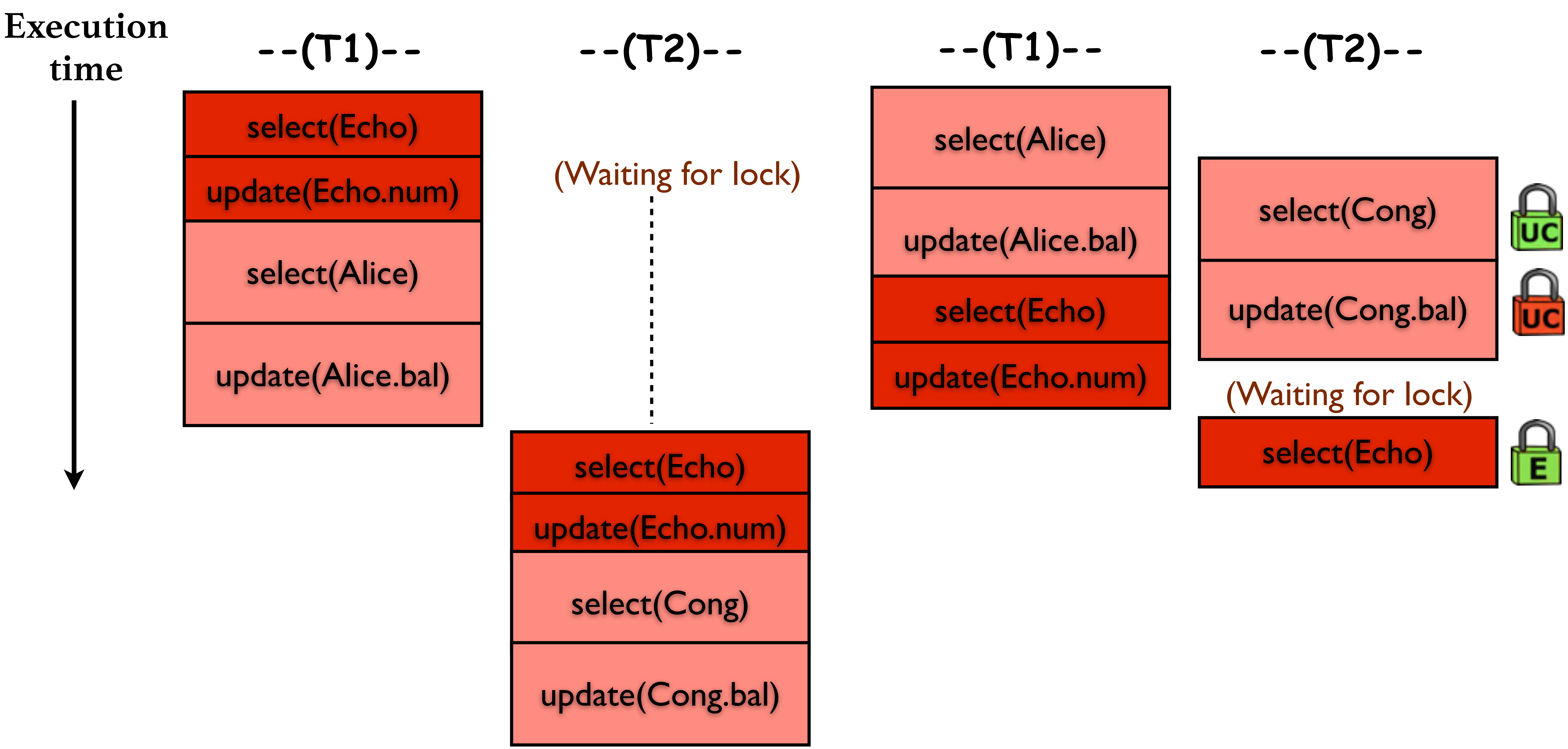
Shortening Lock Waiting



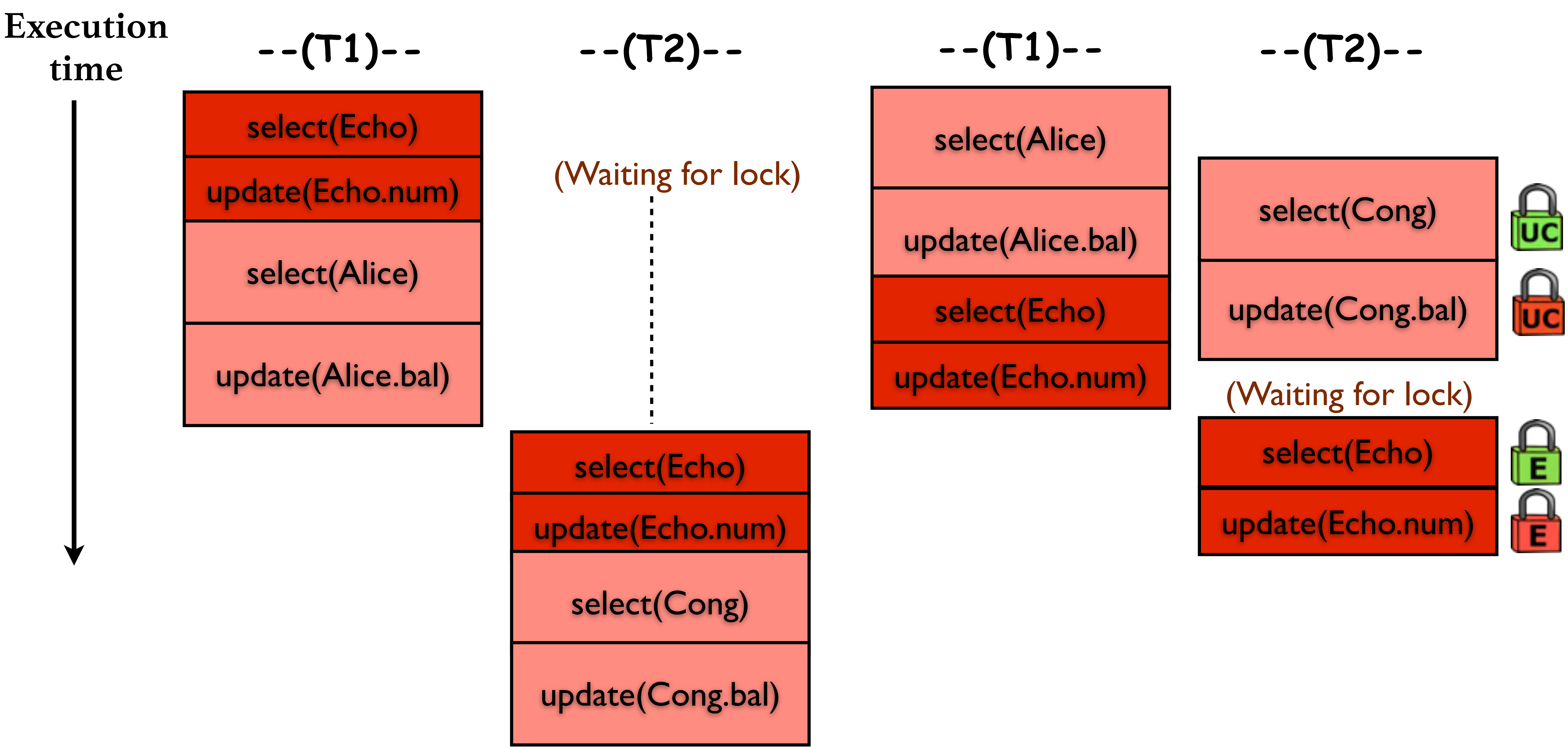
Shortening Lock Waiting



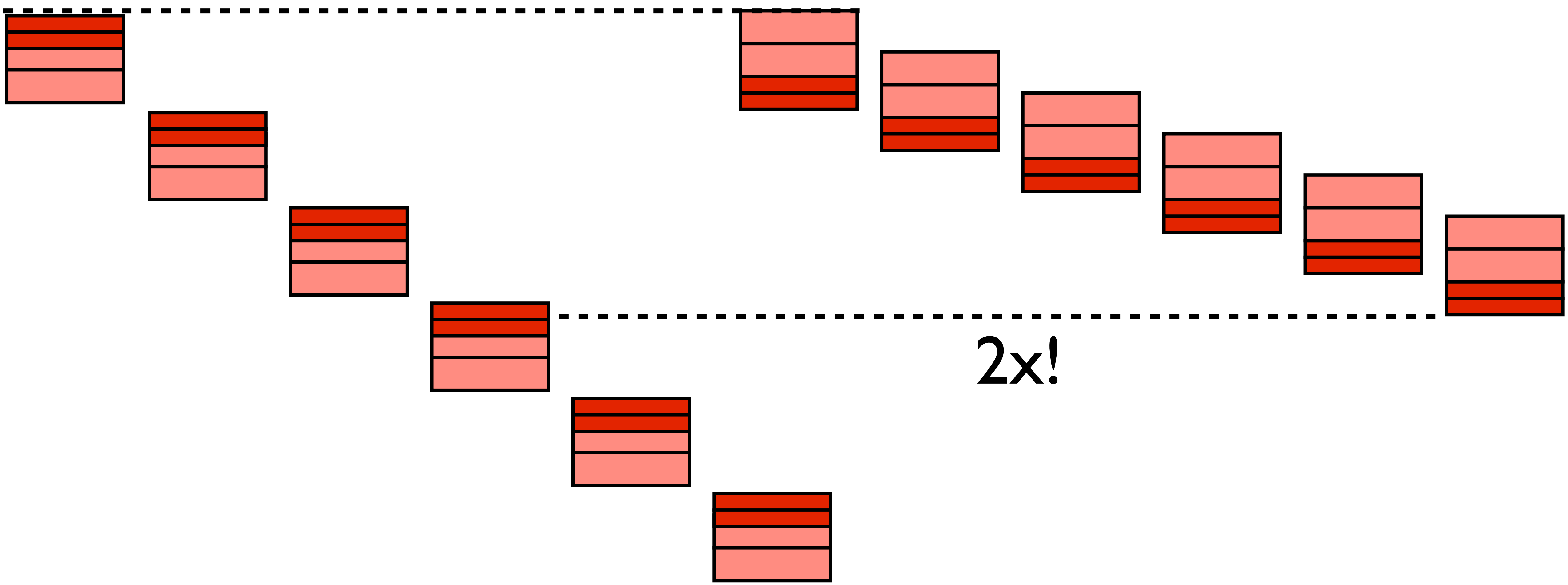
Shortening Lock Waiting



Shortening Lock Waiting

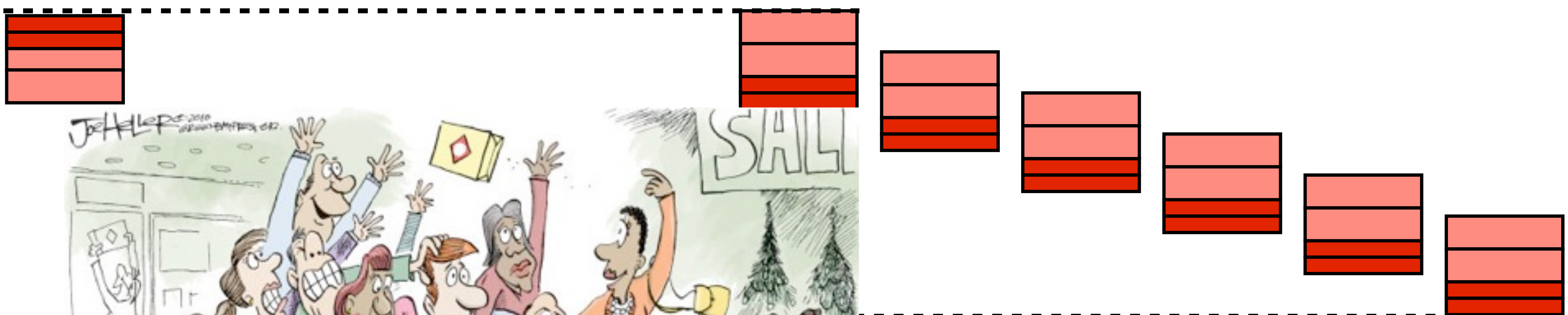


Shortening Lock Waiting



More threads...

Shortening Lock Waiting



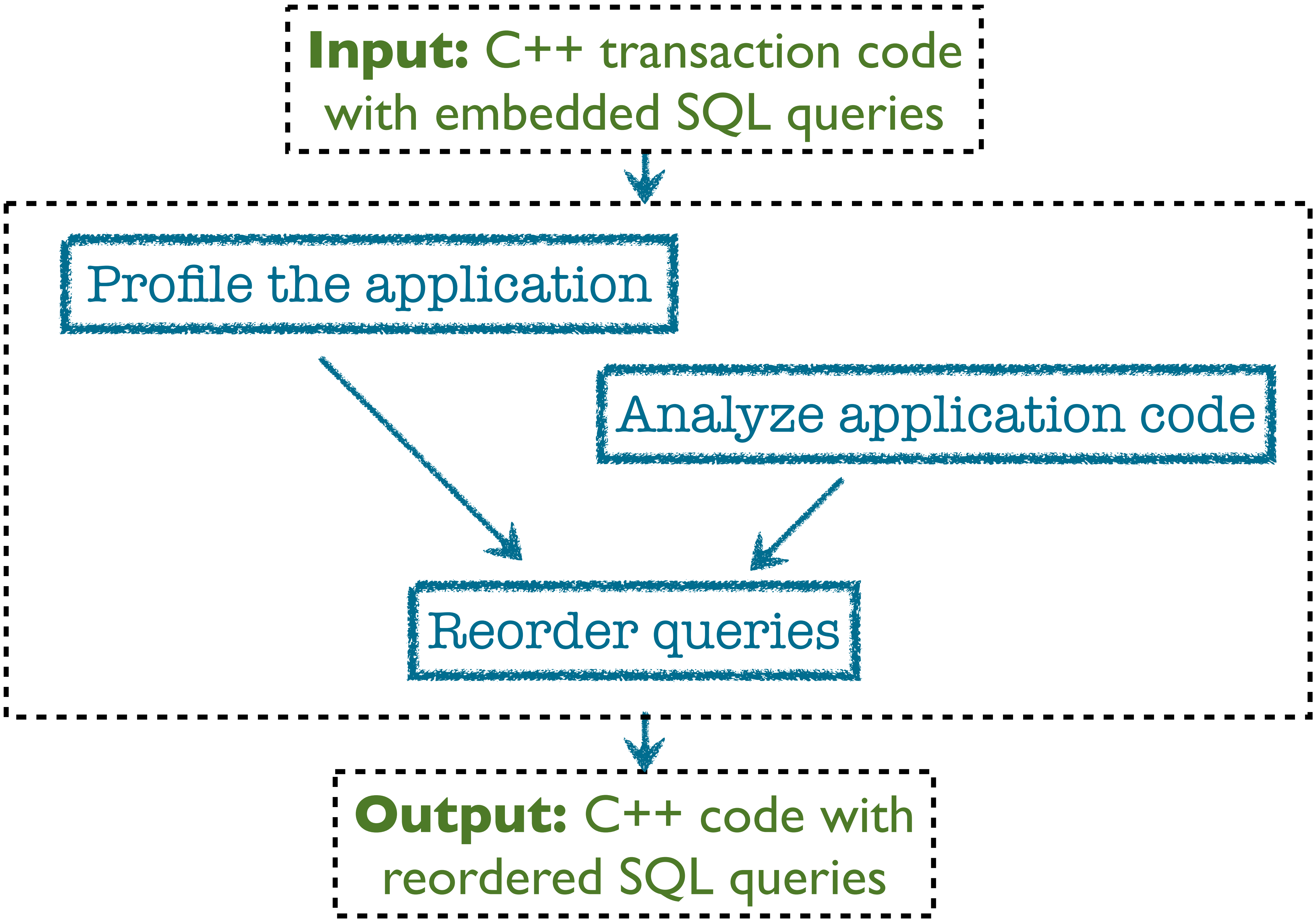
2x!

Shorter latency
Higher throughput

More threads...

- Manually reordering queries is hard
- QURO: a query-aware compiler
  - Automatically reorders queries in transactions based on data contention
  - Preserves original program semantics

QURO


Input: C++ transaction code with embedded SQL queries
Profile the application
Analyze application code
Reorder queries
Output: C++ code with reordered SQL queries

QURO

- Profile the applications
  - Know which queries are likely to access contentious data
    - Calculate the variance of running time for each query

QURO

- Analyze application code
  - Reordering preserves program semantics
    - Data dependency among program variables

```
1. v1 = select("table1");
2. v2 = select("table2", v1);
3. update("table1", input);
```

Statement 1 should appear before statement 2

- Database constraints (same table, view, foreign key...)

```
1. v1 = select("table1");
2. v2 = select("table2", v1);
3. update("table1", input);
```

Statement 1 should appear before statement 3

QURO

- Goal: contentious queries appear as late as possible in transactions
- Constraint: data dependencies & database constraints

QURO

- Goal: contentious queries appear as late as possible in transactions
- Constraint: data dependencies & database constraints

Optimization problem!

QURO

- Goal: contentious queries appear as late as possible in transactions
- Constraint: data dependencies & database constraints

  Optimization problem!

- Formalize into ILP
- Optimizations: reordering long transactions within seconds

Evaluation

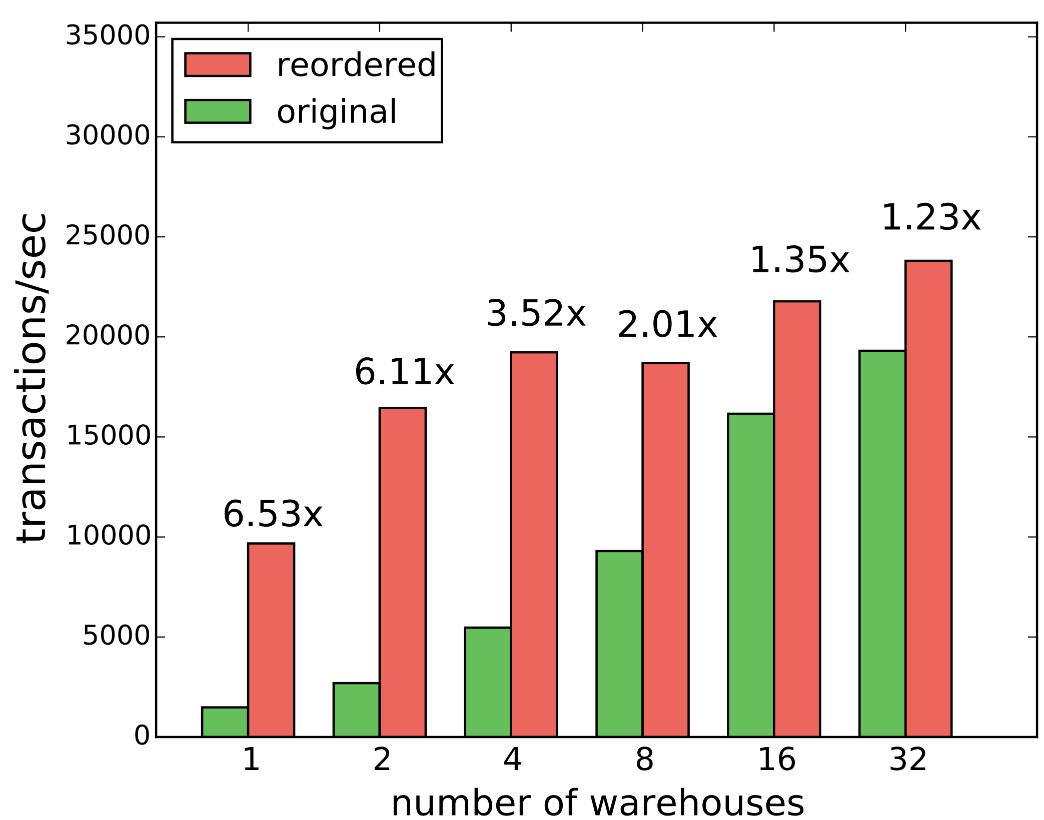
- Experiment overview:
  - Benchmarks: TPC-C, TPC-E
  - Throughput:
    original Vs. reordered implementation (by QURO)
  - Increasing data contention
    - smaller data size
    - more threads

Evaluation

- Benchmark: TPC-C payment transaction
  - changing data size

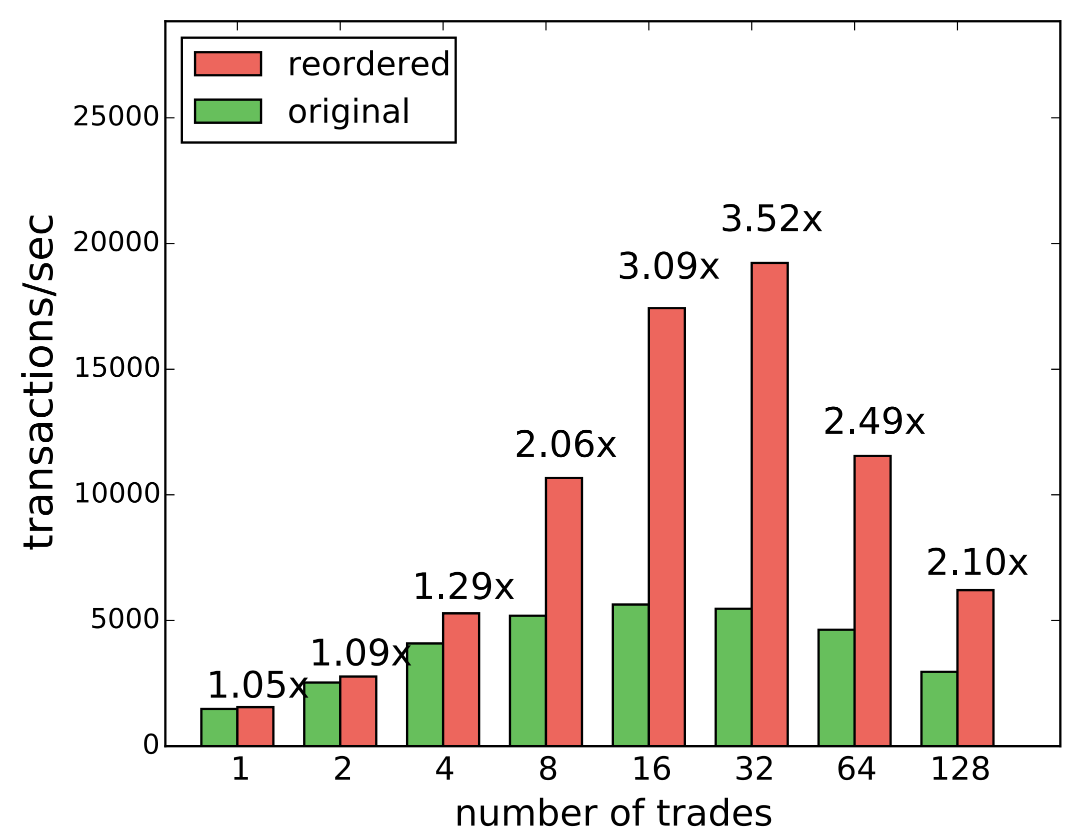
scaling to more threads

contention ←



latency: -83%

→ contention



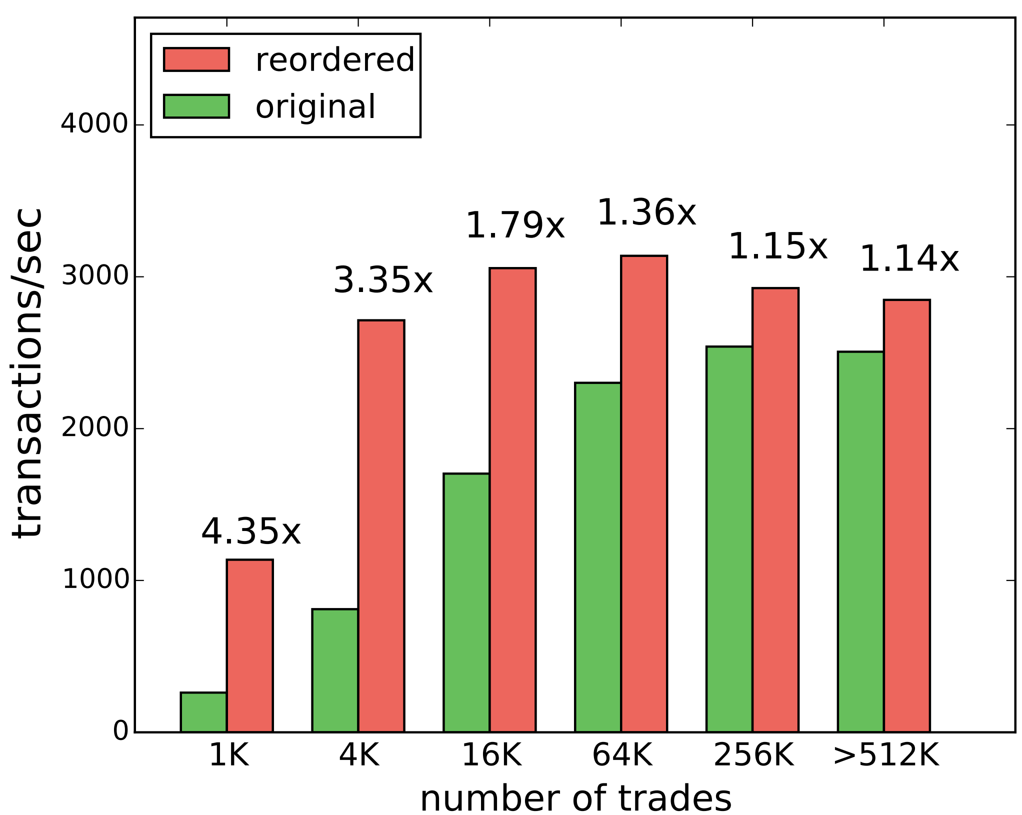
latency: -70%

Evaluation

- Benchmark: TPC-E trade update transaction
  - changing data size

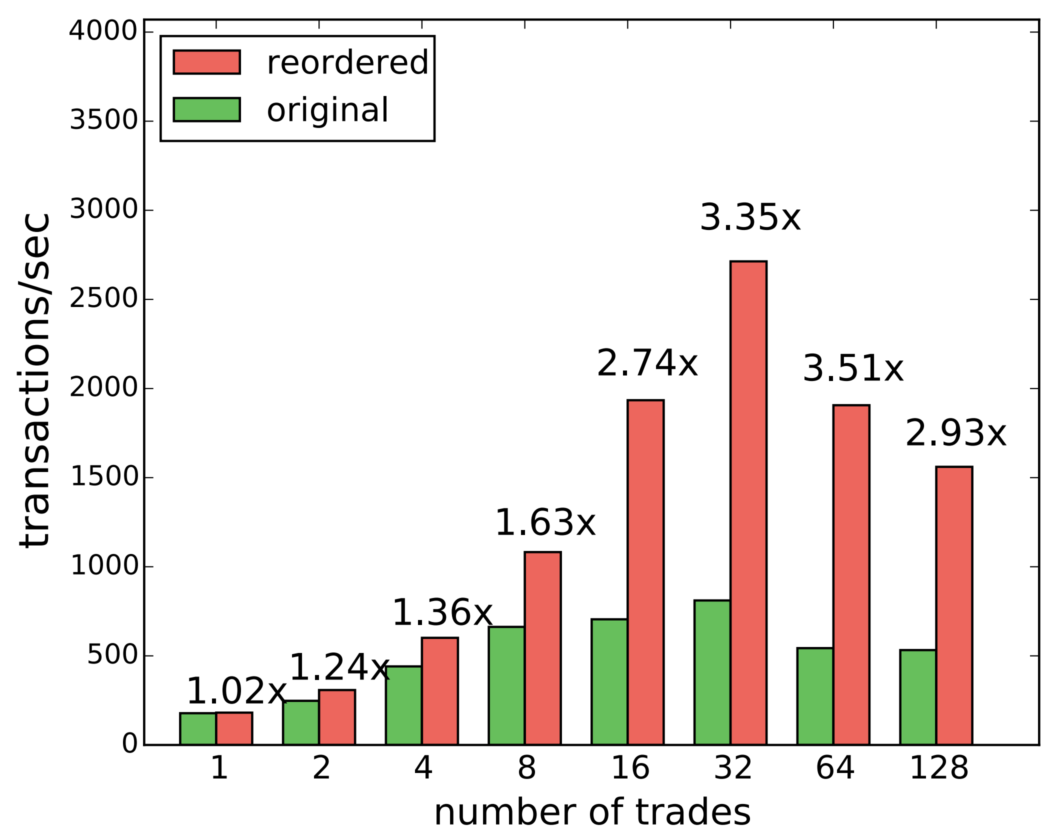
scaling to more threads

contention ←



latency: -75%

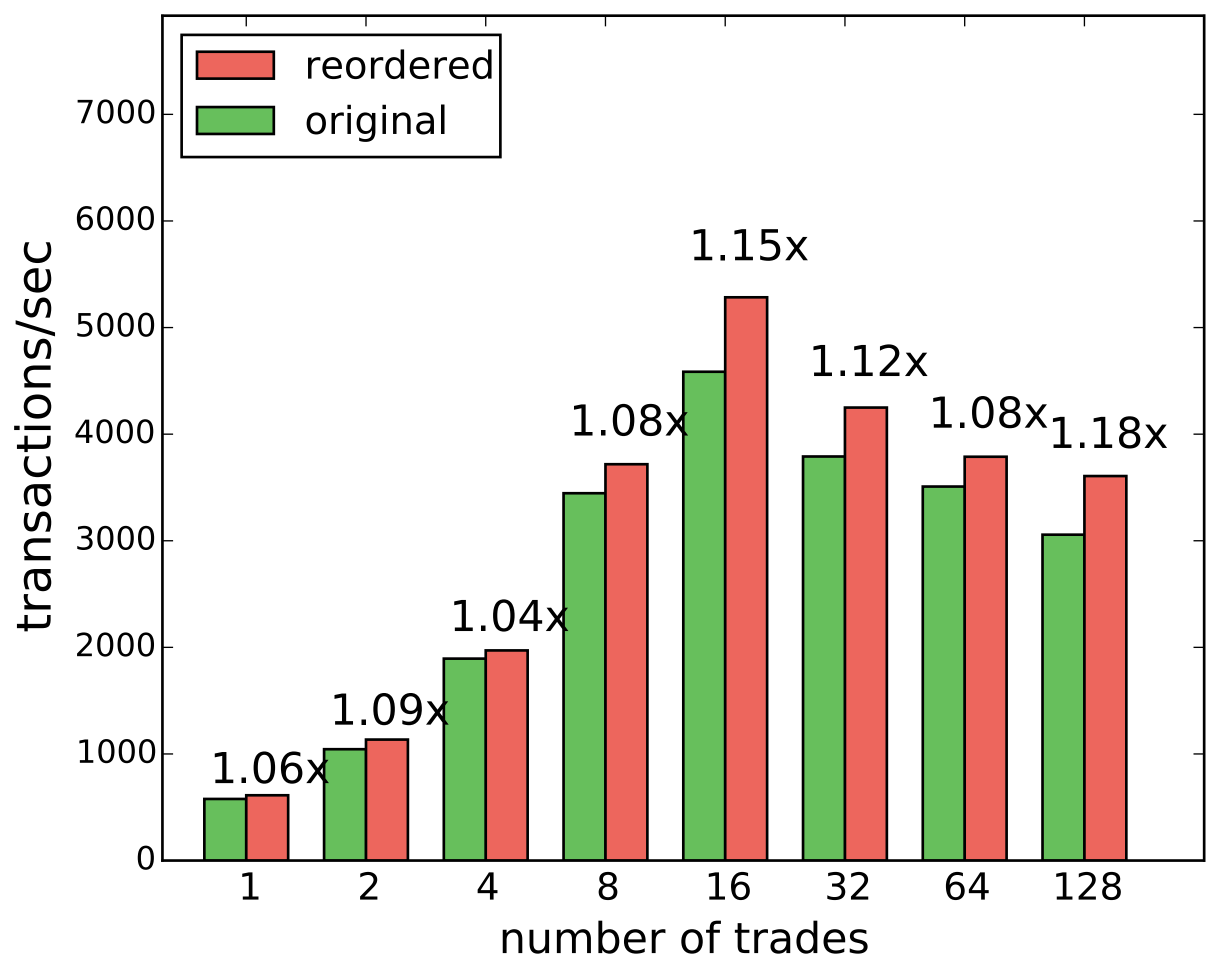
→ contention



latency: -66%

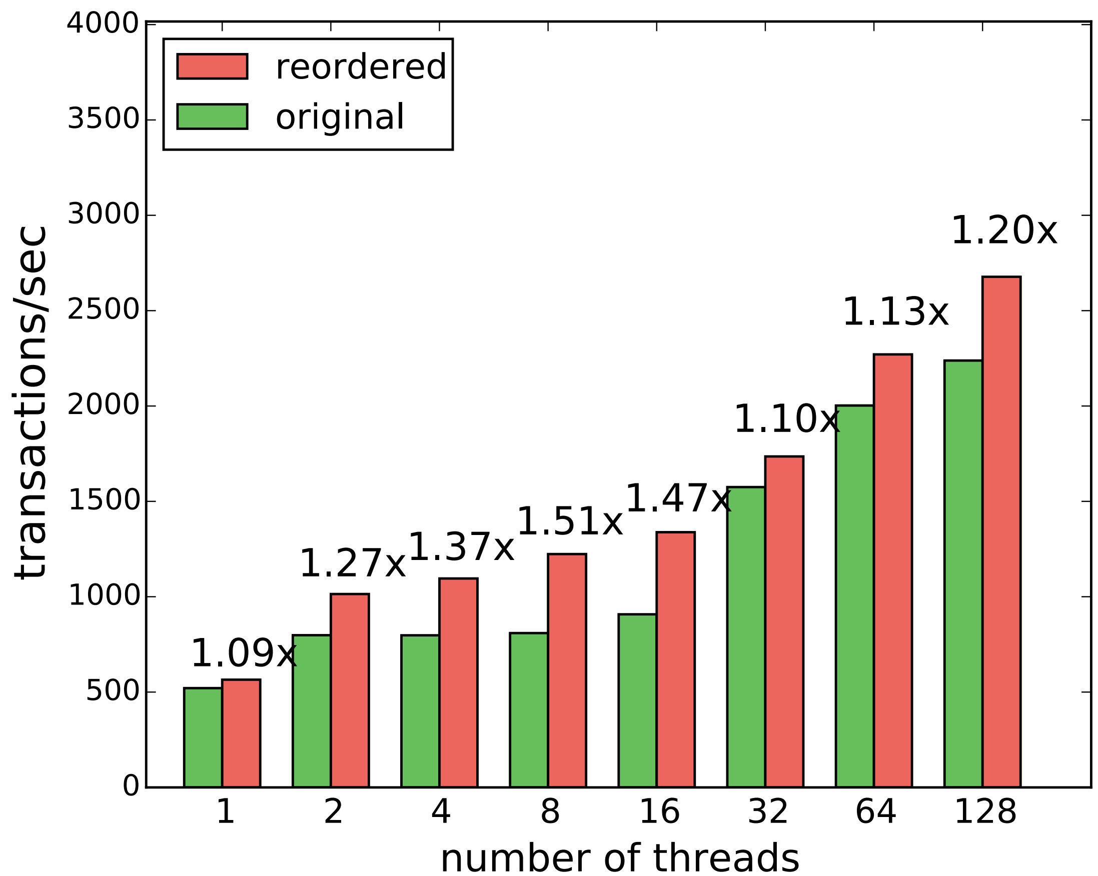
Experiments

- Mix of different transactions
  - TPC-C standard mix: 5 types of transactions



Experiments

- More complicated transactions
  - TPC-E trade order and result
    - Each >500 lines of code, >20 queries, complicated logic



"AND TO THINK, JUST THE OTHER DAY I WAS WORRIED ABOUT STRANGERS TOUCHING MY JUNK AT THE AIRPORT."

"AND TO THINK, JUST THE OTHER DAY I WAS WORRIED ABOUT STRANGERS TOUCHING MY JUNK AT THE AIRPORT."

Better black friday!

Conclusion

- The order of query has large impact on transaction performance.
- QURO leverages information about query contention, and automatically reorders the queries.
- Reordered code generated by QURO can improve throughput up to 6.53x, and can be applied to a wide range of applications.
- We are in the process of releasing code (congy@cs.washington.edu).