

Transactional Libraries

Alexander Spiegelman^{*}, Guy Golan-Gueta[†], and Idit Keidar^{†*}

^{*}Technion

[†]Yahoo Research

Multi-Threading is Everywhere

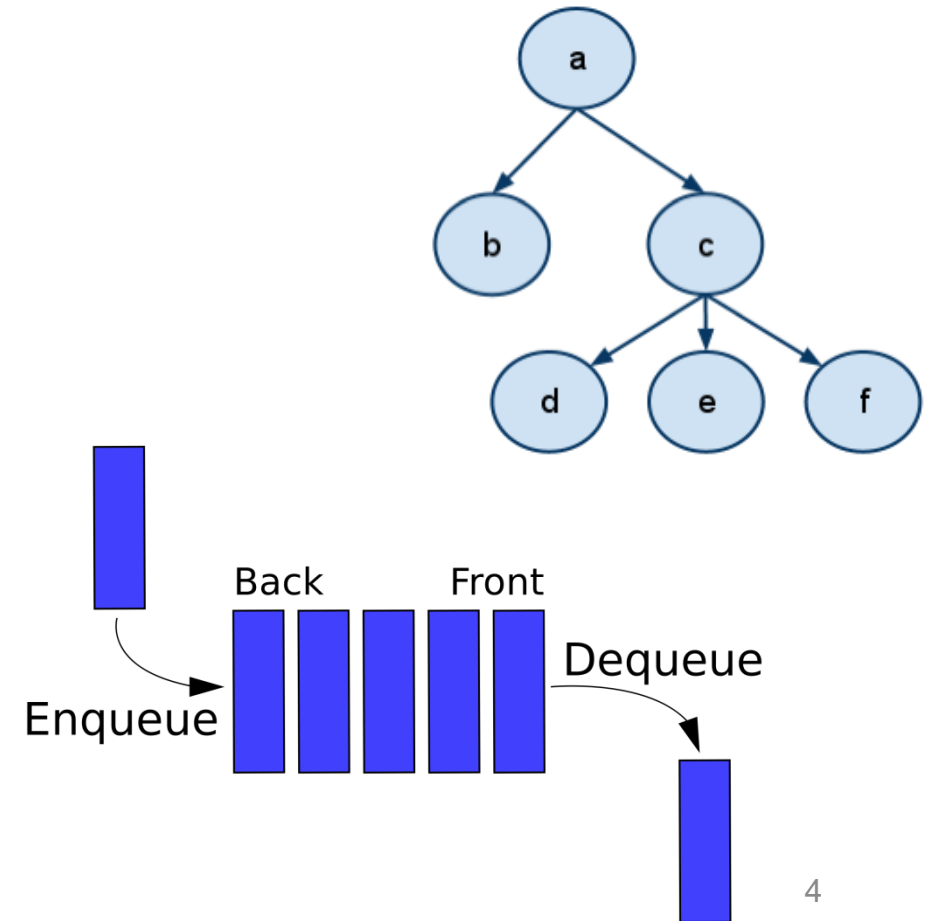
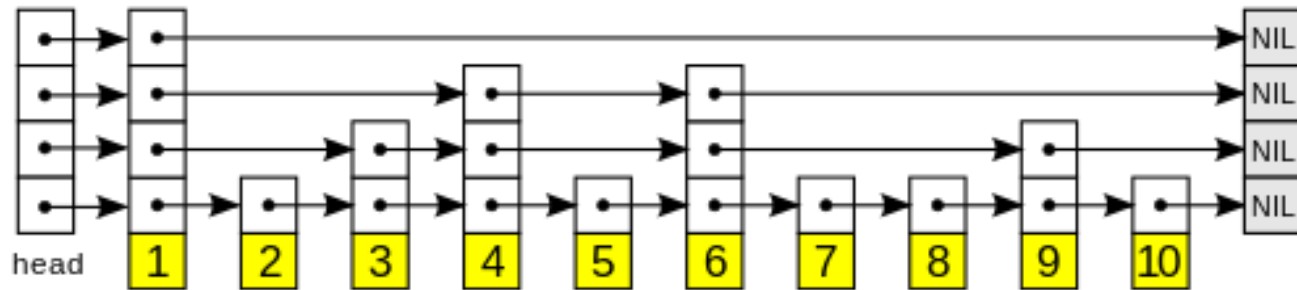


Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

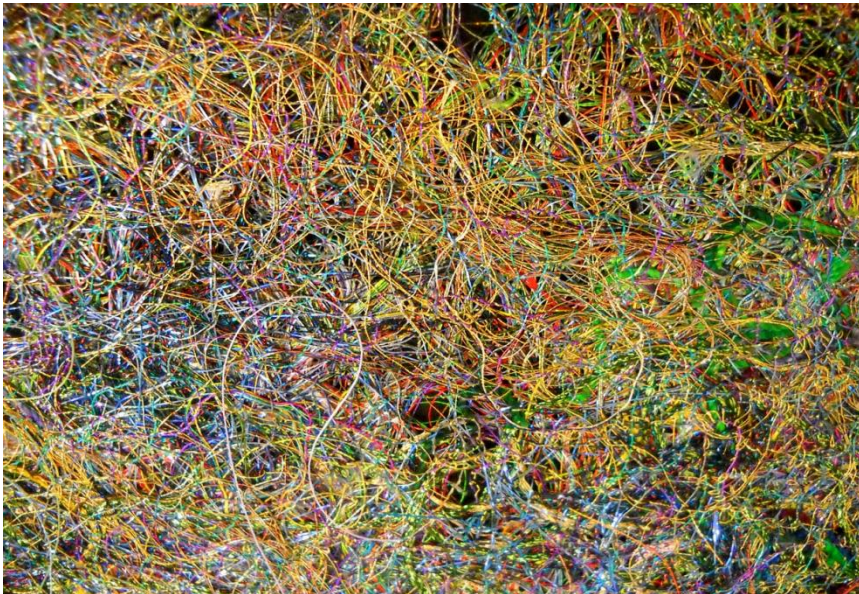
Data Structures (DS)

- Essential building blocks in modern SW
 - Map, skiplist, queue, etc.



But Are They “Thread Safe”?

Correct under concurrency?



OR



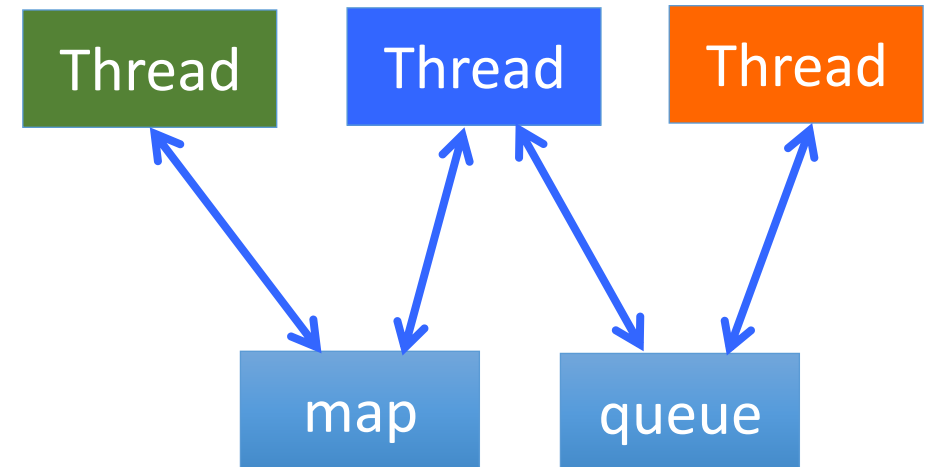
?

“Thread-Safe” Concurrent DS Libraries

- Widely used in real life software
- Numerous research papers
 - Concurrent Skipklist
[Herlihy, Lev, Luchangco, Shavit: A simple optimistic skiplist algorithm]
[Fraser: Practical lock freedom] ...
 - Concurrent queue
[Michael, Scott: Simple, fast, and practical nonblocking and blocking concurrent queue algorithms] [Gramoli, Guerraoui: Reusable concurrent data types]
 - Concurrent binary tree
[Bronson, Casper, Chafi, Olukotun: A practical concurrent binary search tree] [Drachsler, Vechev, Yahav: Practical concurrent binary search trees via logical ordering]

Concurrent Data Structure Libraries (CDSLs)

- Each operation executes atomically
- Custom-tailored implementation preserves semantics



Are They Really Thread Safe??

```
balance ←map.get (key=Yoni)
```

```
newBalance← balance+deposit
```

```
map.set(key=Yoni,newBalance)
```

```
repQ.enq("Yoni's balance=new")
```

```
balance ←map.get(key=Yoni)
```

```
newBalance← balance+deposit
```

```
map.set(key=Yoni,newBalance)
```

```
repQ.enq("Yoni's balance=new")
```

- Oops! Atomic operations are not enough

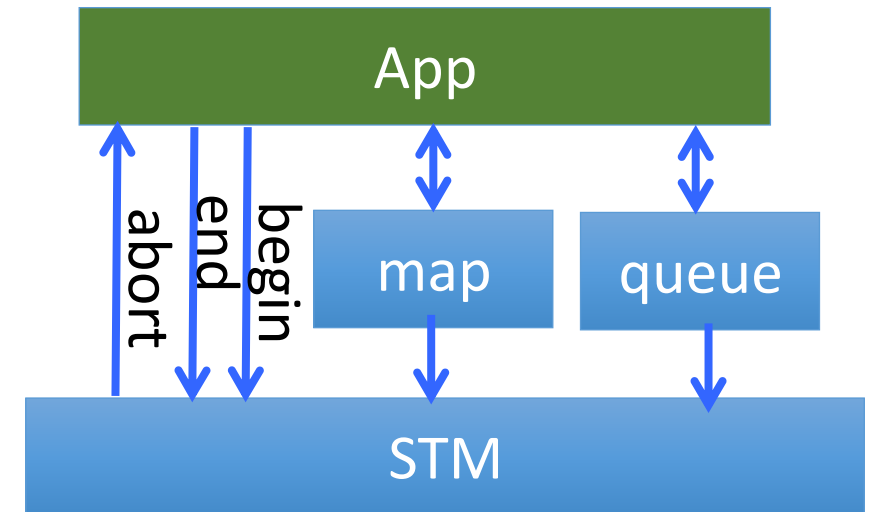
Software Transactional Memory (STM)

- TL2, TinySTM, SwissTM,...
- Transactions (TXs) = atomic sections including multiple DS operations

Begin_TX

```
balance ← map.get(key=Yoni)
newBalance ← balance + deposit
map.set(key=Yoni, newBalance)
repQ.enq("Yoni's balance=new")
```

End_TX



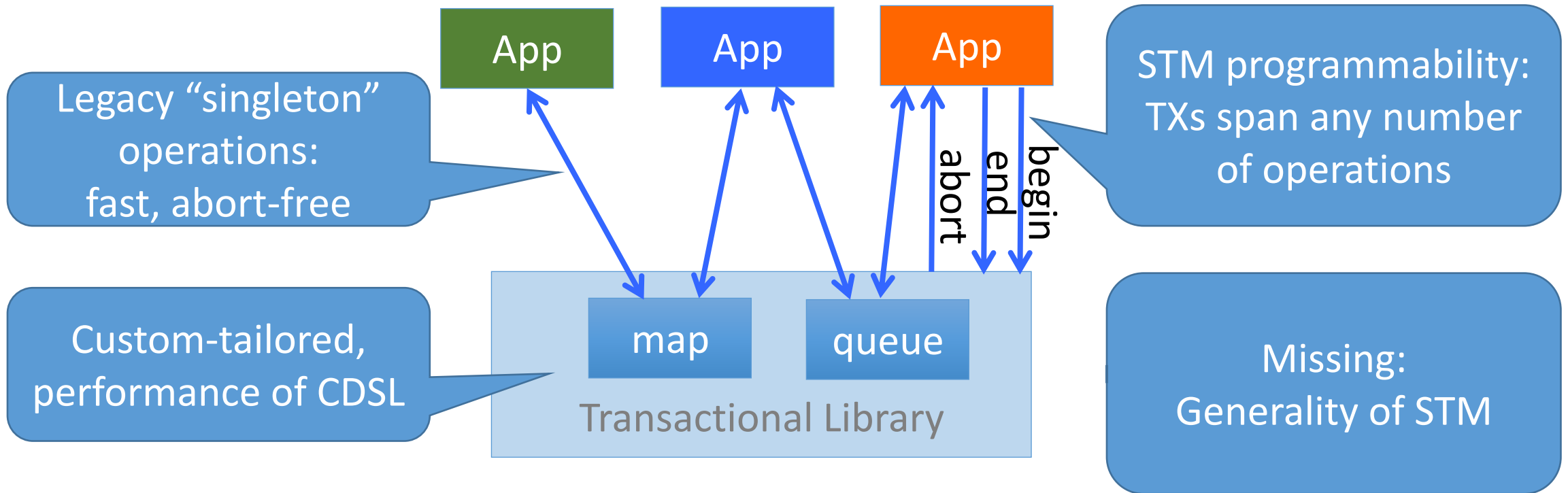
CDSL vs STM

	CDSL	STM
Performance	✓	✗
Exploit DS semantics & structures	✓	✗
Used in practice	✓	✗
Generality	✗	✓
Composability	✗	✓

Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- **Introducing: Transactional Data Structure Libraries (TDSL)**
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

TDSL: Bringing Transactions into CDSL



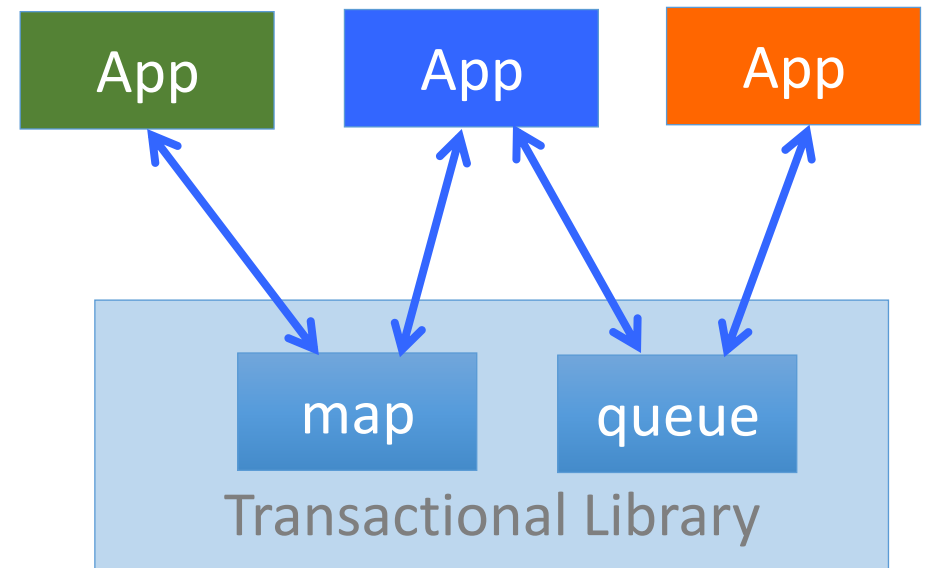
TDSL Benefit 1: Programmability

- Support for legacy code
 - Fast, abort-free singletons
- Power of transactions

Begin_TX

```
val ← map.get(key=Yoni)
new ← val + deposit
map.set(key=Yoni, new)
repQ.enq("Yoni's balance=new")
```

End_TX

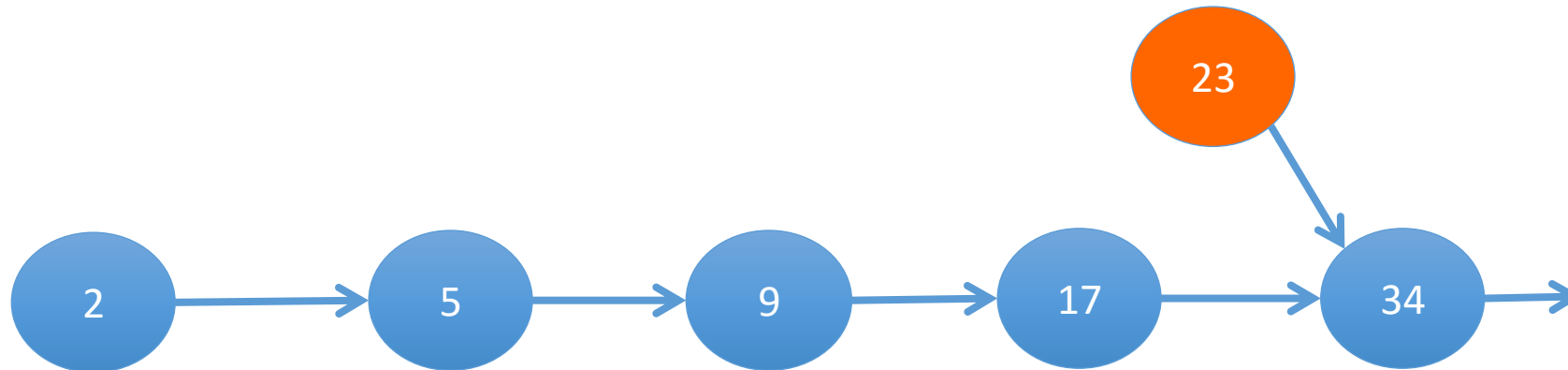


TDSL Benefit 2: Semantic Optimization

- Use known transactional solutions
- But, take advantage of semantics & structure of each DS to reduce aborts & overhead
 - Reduce read-set
 - Do some of the work non-transactionally

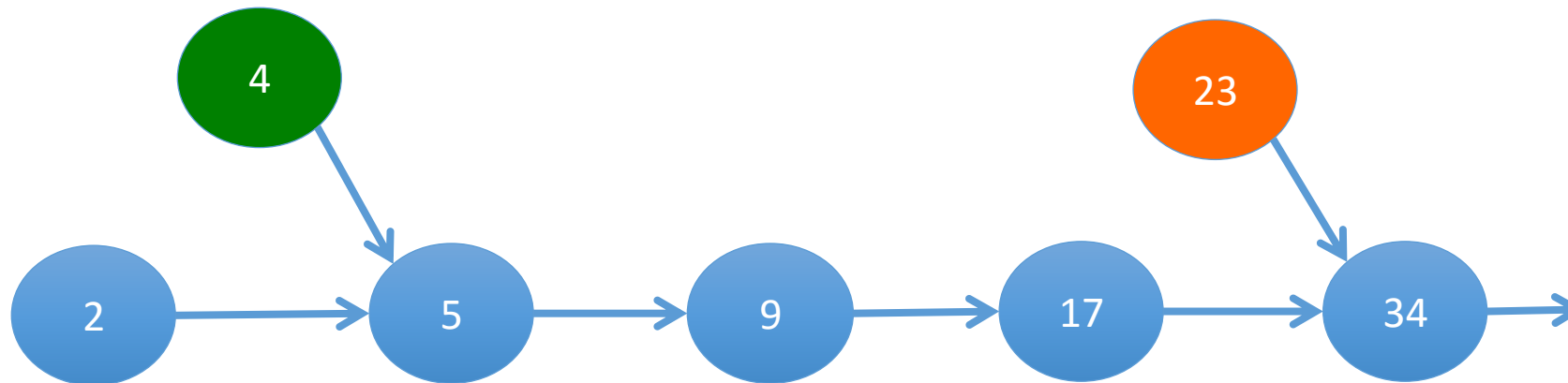
Example of Abort Reduction

- Two concurrent put operations
- Put(23) traverses the list, reads nodes that put(4) updates
 - Conflict, STM would abort at least one
 - But no semantic conflict



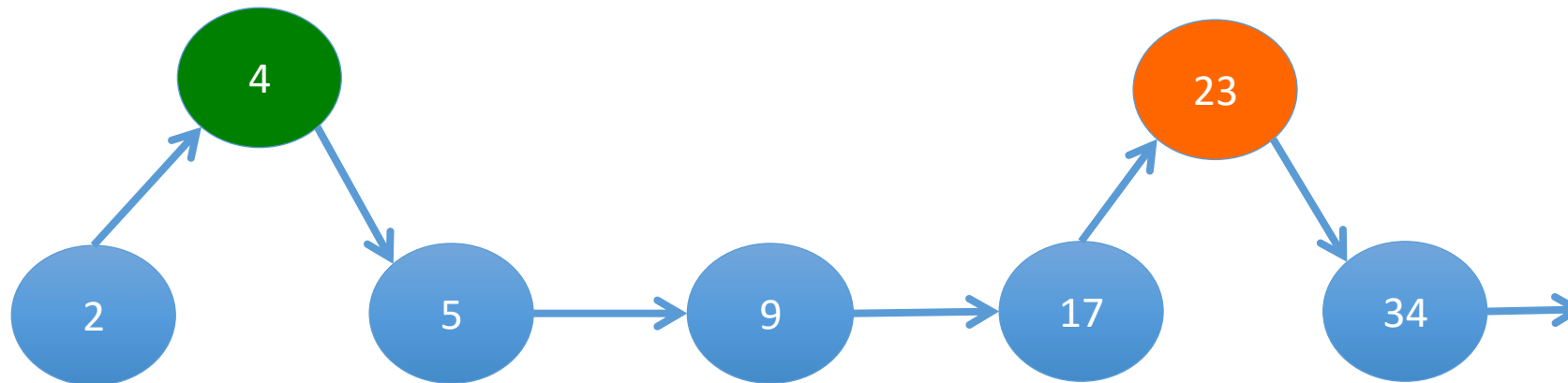
Example of Abort Reduction

- Two concurrent put operations
- Put(23) traverses the list, reads nodes that put(4) updates
 - Conflict, STM would abort at least one
 - But no semantic conflict



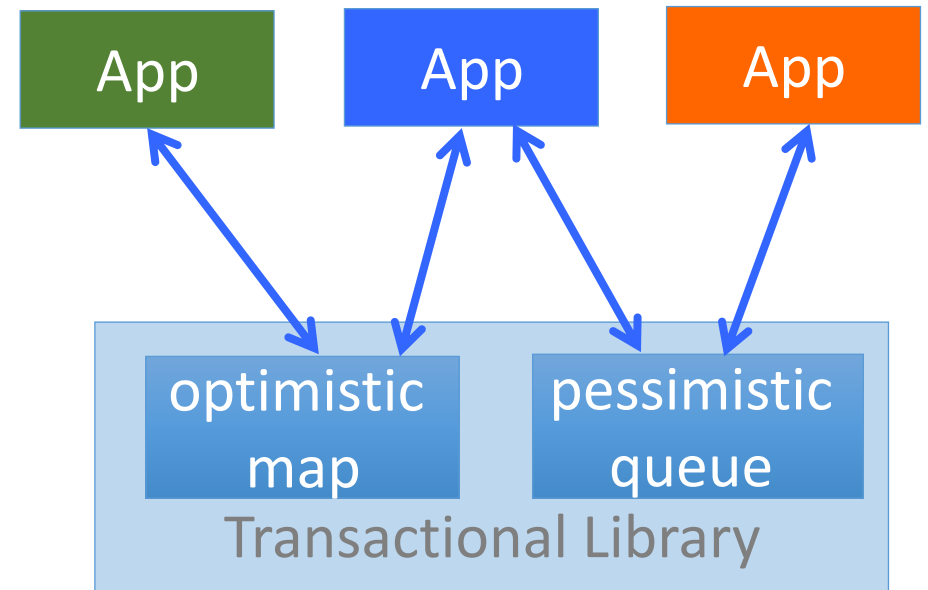
Example of Abort Reduction

- Two concurrent put operations
- Put(23) traverses the list, reads nodes that put(4) updates
 - Conflict, STM would abort at least one
 - But no semantic conflict



TDSL Benefit 3: Mix & Match

- Maps allow lots of concurrency
 - Amenable to optimistic synchronization
- Queues do not
 - Pessimistic synchronization better
- STM picks one
- Our TDSL can mix & match



Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

Skiplist Roadmap

1. Add STM-like transaction support to simple linked list
 - Based on TL2 STM algorithm
2. Optimization: remove redundant validation and tracking
 - Reduce aborts
 - Use semantics and structure
3. Optimization: shorten transactions
 - Reduce aborts
 - Non-transactional index
 - Lazy GC

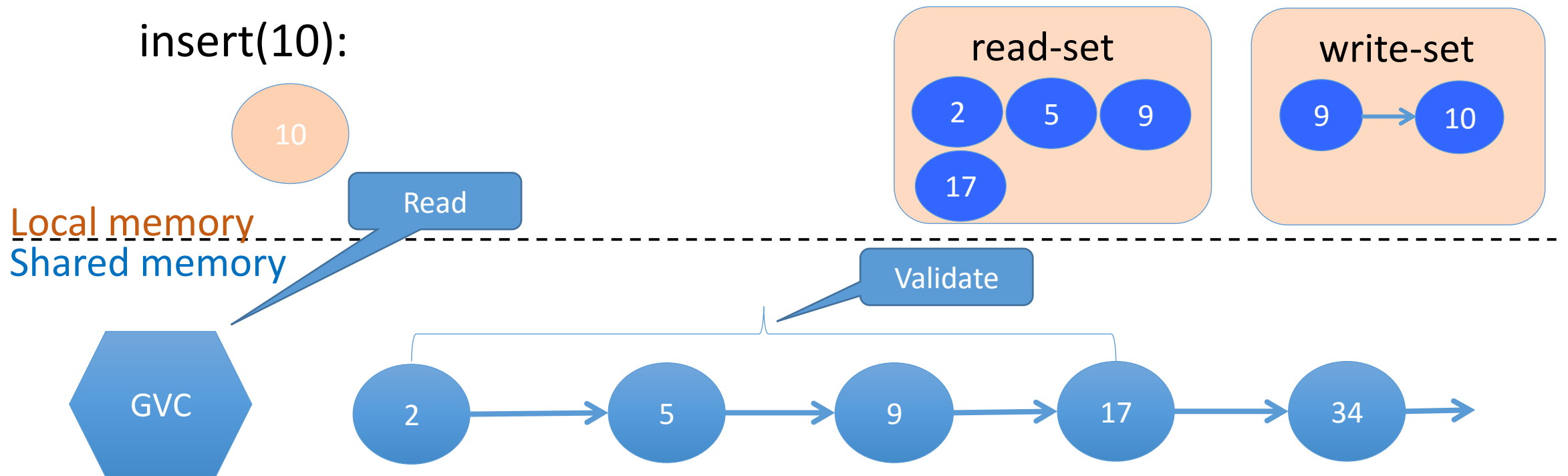
Step 1: Standard STM

- Take a simple linked list



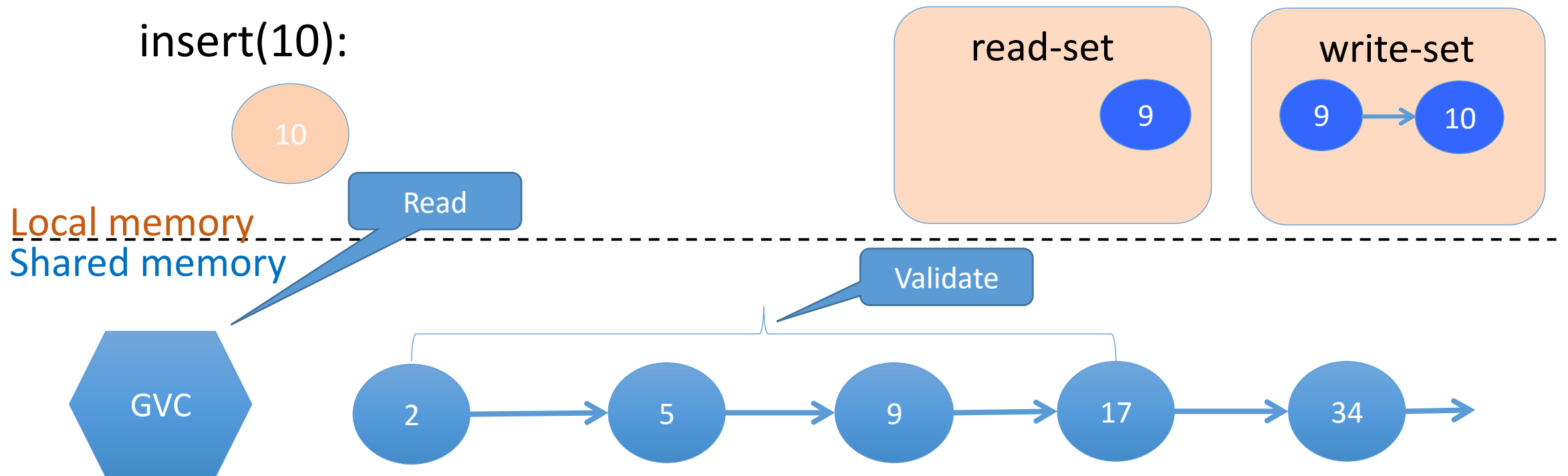
- Add TL2 mechanism to support TXs:
 - Add a version to each node
 - Get versions from global version clock (GVC)
 - Maintain read-set with read versions
 - Defer updates, track in write-set
 - To commit: lock write-set, validate read-set, increment GVC, update

Step 1: Standard STM



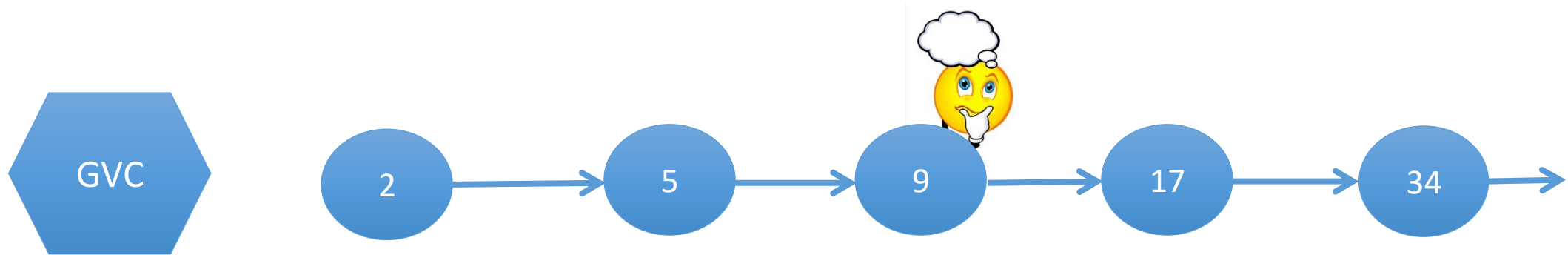
Step 2: Reduce Read-Set

- Exploit structure and semantics



Step 3: Non-Transactional Index

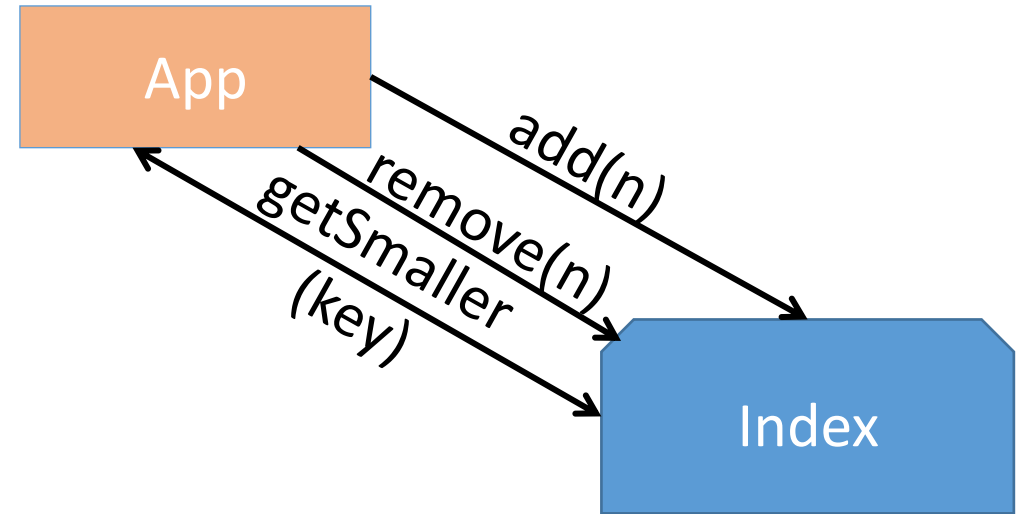
- Imagine we could guess the right node



- So, we could be faster and save aborts during the traversal

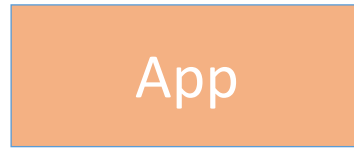
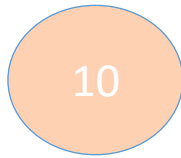
Step 3: Non-Transactional Index

- getSmaller
 - Returns some node with a smaller key
 - For performance, not much smaller
- Implemented as (concurrent) skiplist
 - For example



Step 3: Non-Transactional Index

insert(10):



getSmaller(10)

return 5



Start traverse from 5



Step 3: Non-Transactional Index

- Updated outside the TX (if completes successfully)
 - Reduces aborts, overhead
- But:
 - May return nodes with smaller keys than predecessor
⇒ longer traversals
 - May return removed nodes
 - Subtle, more details in the paper

Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

Composition

- One GVC shared among all objects
- Commit:
 - Lock all write sets
 - Validate all read sets
 - Increase GVC
 - Update objects and release locks

Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

Fast Abort-Free Singletons

- Use the index
- Do not increment GVC
 - No contention
 - As fast as in CDSL
 - Make transactions aware of singletons using designated fields
- Details in the paper

Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

Compose With Existing Libraries

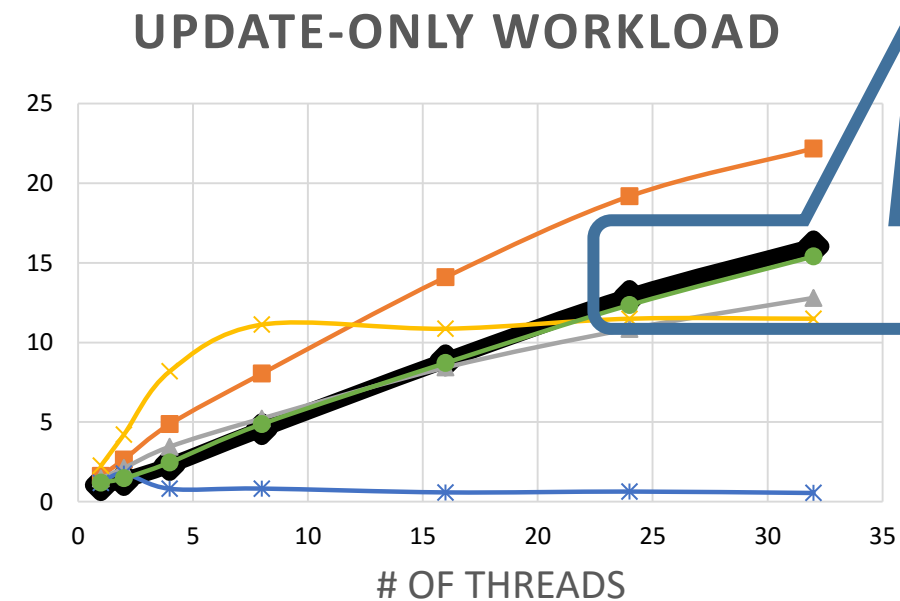
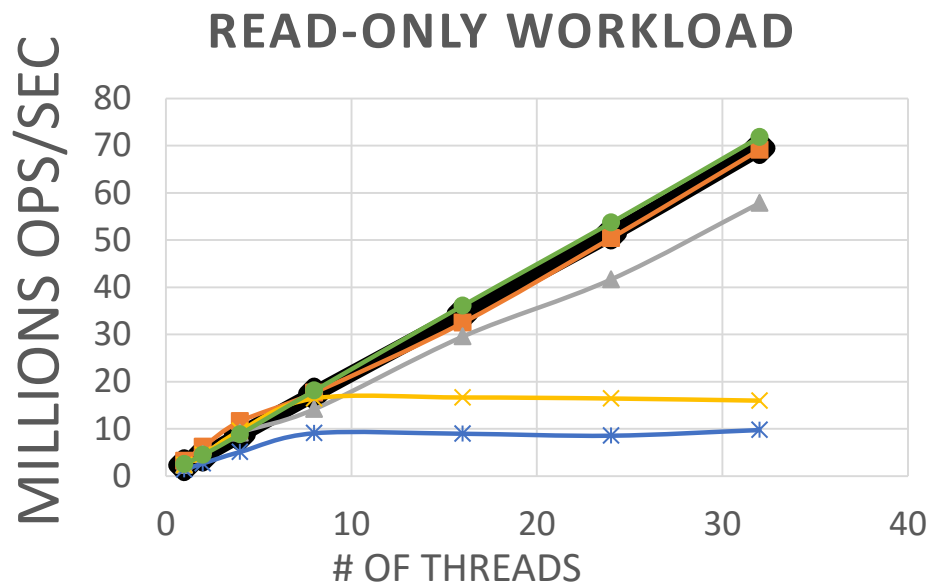
- Modify library to support certain API
 - E.g., TL2 STM naturally supports it
 - E.g., use standard 2 phase locking
- Together we get **general transactions**
 - Ones supported fully by a custom-tailored TDSL are fast
 - Others less so

Agenda

- Motivation
 - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
 - Skiplist
 - Composition of multiple objects (skiplists and queues)
 - Fast abort-free singletons
- Library composition
- Evaluation

Singletons

As good as baseline



our skiplist

rotating

nohotspot

fraser

optimistic

baseline

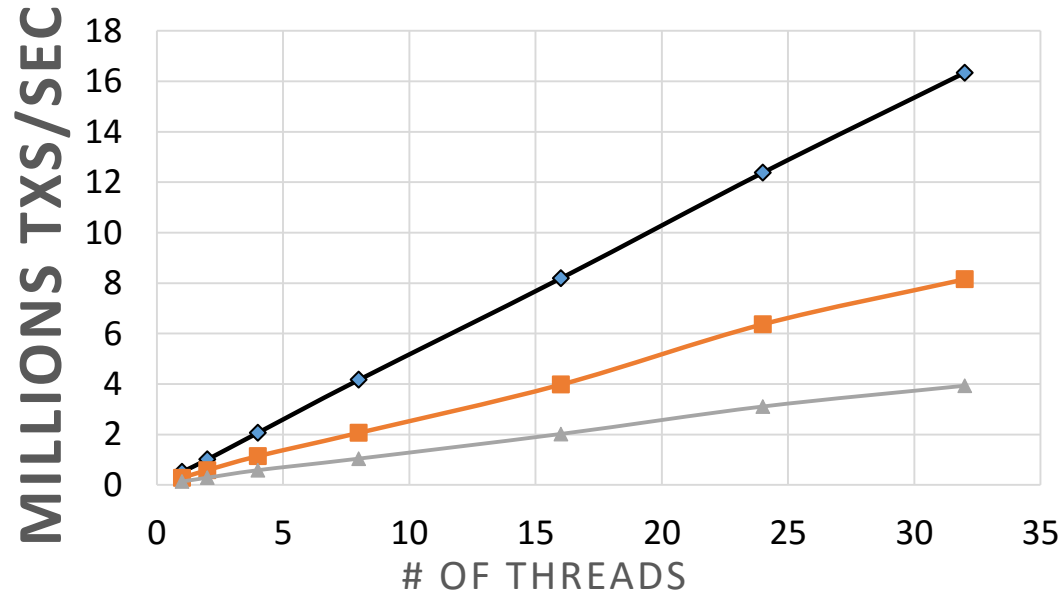
Do not support transactions

Transactions

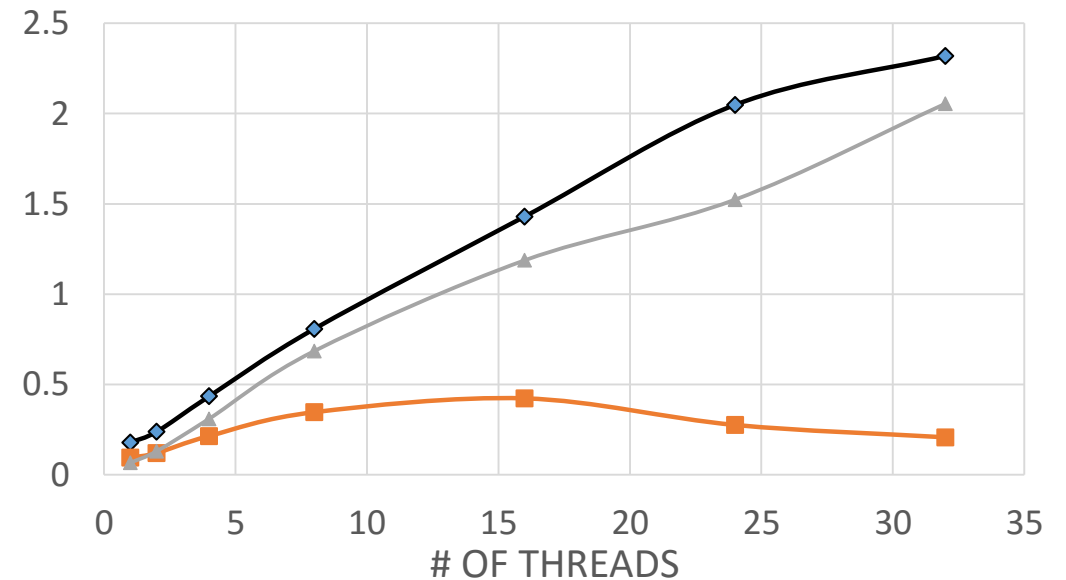
We have less overhead

We have less aborts

READ-ONLY WORKLOAD



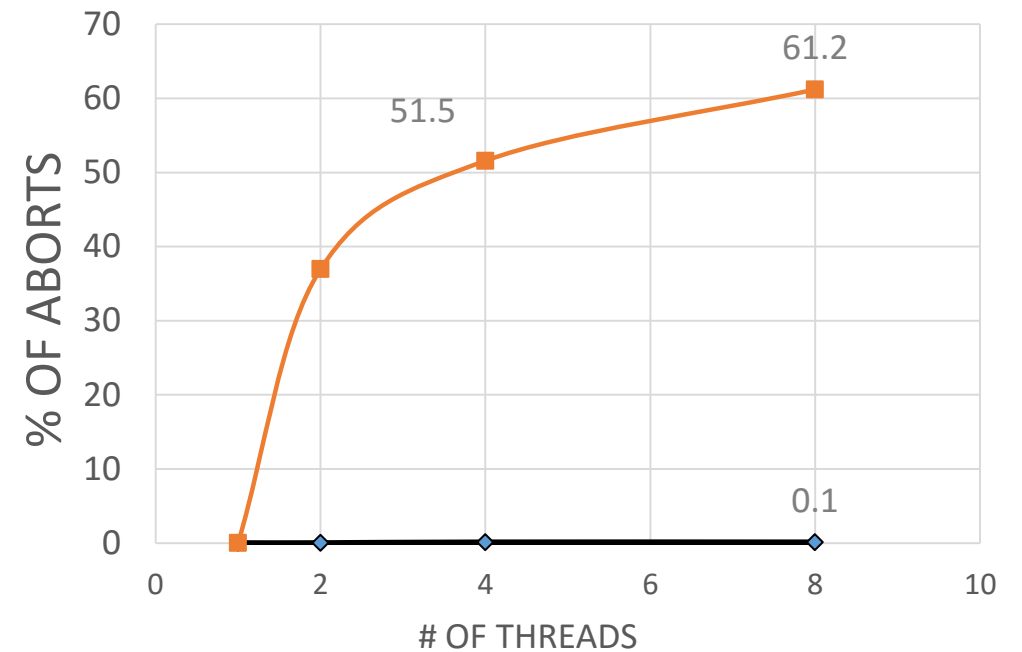
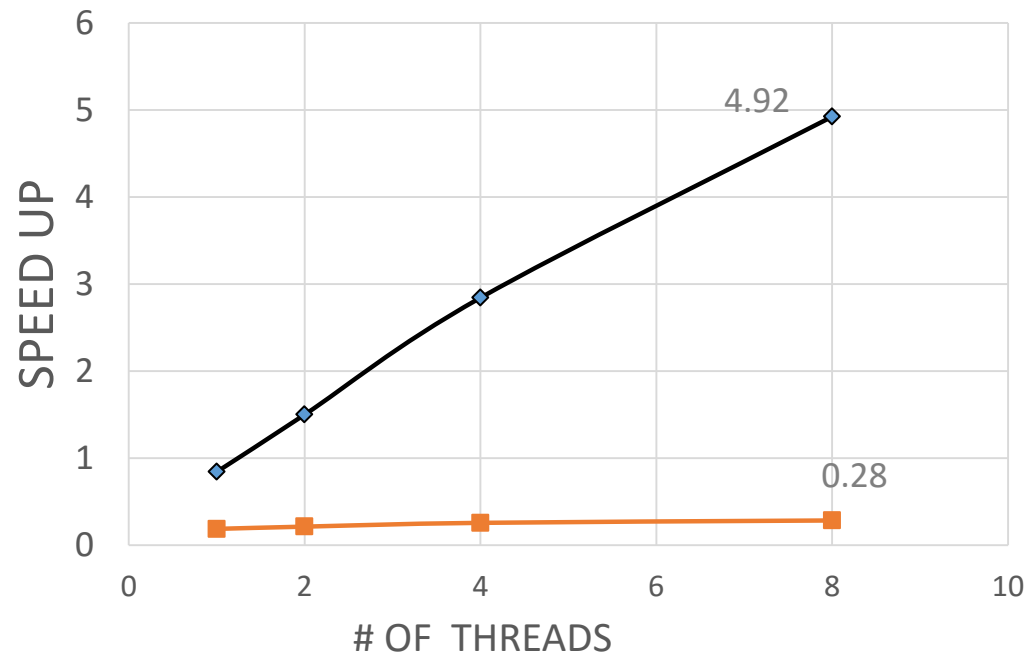
UPDATE-ONLY WORKLOAD



—◆— our skiplist —■— SeqTL2 —▲— FriendlyTL2

Intruder

Testing multiple skiplists and queues in a real application



—◆— our skiplist —■— SeqTL2 —▲— FriendlyTL2

cannot support
intruder

Related Work

Boosting and optimistic boosting

- [M. Herlihy and E. Koskinen: Transactional boosting: methodology for highly-concurrent transactional objects]
- [Ahmed Hassan, Roberto Palmieri, Sebastiano Peluso, and Binoy Ravindran: Optimistic Transactional Boosting]


Open nesting

- [Y. Ni, V. S. Menon, A.-R. Adl-Tabatabai, A. L. Hosking, R. L. Hudson, J. E. B. Moss, B. Saha, and T. Shpeisman. Open nesting in software transactional memory]

Reagents

- [A. Turon. Reagents: expressing and composing fine-grained concurrency]

Conclusion

	CDSL	STM	TDSL
Performance	✓	✗	✓
Exploit DS semantics	✓	✗	✓
Used in practice	✓	✗	
Generality	✗	✓	✓
Composability	✗	✓	✓

We hope that the community will adopt this concept

- Build and use more such libraries