भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

MICHIGAN STATE
UNIVERSITY

Achieving Starvation-Freedom with Greater Concurrency in Multi-Version Object-based Transactional Memory Systems

Chirag Juyal[1] Sandeep Kulkarni[2] Sweta Kumari[1] **Sathya Peri[1]**
Archit Somani[1]

[1]CSE Dept, IIT Hyderabad [2]CSE Dept, Michigan State University

21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2019)

Outline of Today's Presentation

Outline

Transaction

Sequence of instructions guaranteed to execute atomically.

Transaction

Sequence of instructions guaranteed to execute atomically.

History

Concurrent execution of transactions.

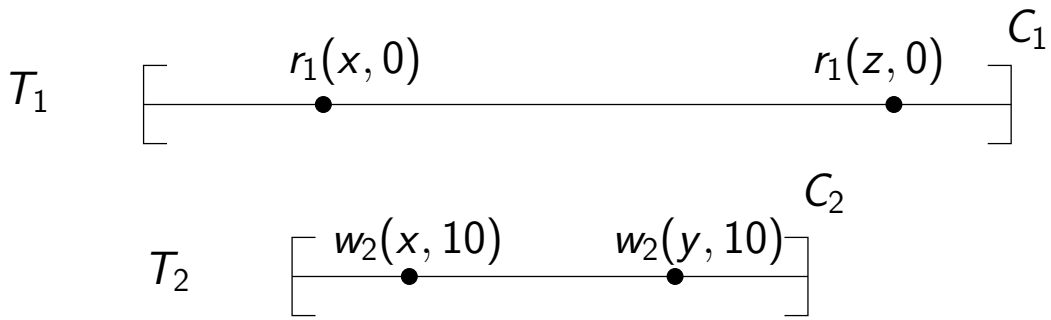


Figure 1: History of an STM

Introduction to STMs Cont'd..

Definition of STMs and its methods

- **Software Transaction Memory Systems (STMs)** are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues.

Introduction to STMs Cont'd..

Definition of STMs and its methods

- **Software Transaction Memory Systems (STMs)** are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues.
- Traditionally, STMs export the following methods:
  - t_begin()
  - t_read()
  - t_write()
  - tryC() and tryA()

  We refer to these as Read-Write STMs (or RWSTMs).

Introduction to STMs Cont'd..

Definition of STMs and its methods

- **Software Transaction Memory Systems (STMs)** are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues.
- Traditionally, STMs export the following methods:
  - t_begin()
  - t_read()
  - t_write()
  - tryC() and tryA()

  We refer to these as Read-Write STMs (or RWSTMs).

▸ Working of STMs

Outline

Correctness of STMs: Opacity [a]

Opacity

1. It is a popular correctness-criteria for STMs which is stronger than serializability.
2. Opacity like serializability requires that the concurrent execution including the aborted transactions be equivalent to some serial execution.

[a] Guerraoui, R., Kapalka, M.: On the Correctness of Transactional Memory. PPoPP, 2008

Correctness of STMs: Opacity [a]

Opacity

1. It is a popular correctness-criteria for STMs which is stronger than serializability.
2. Opacity like serializability requires that the concurrent execution including the aborted transactions be equivalent to some serial execution.

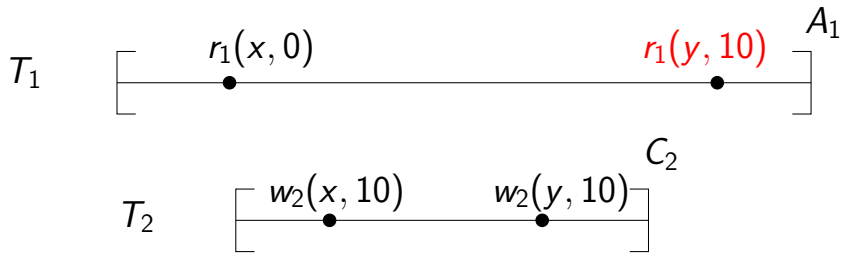


Figure 2: History H is not Opaque

[a] Guerraoui, R., Kapalka, M.: On the Correctness of Transactional Memory. PPoPP, 2008

Correctness of STMs: Opacity [a]

Opacity

1. It is a popular correctness-criteria for STMs which is stronger than serializability.
2. Opacity like serializability requires that the concurrent execution including the aborted transactions be equivalent to some serial execution.

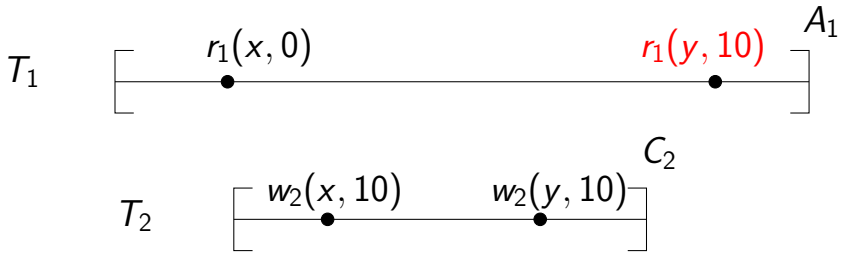


Figure 2: History H is not Opaque

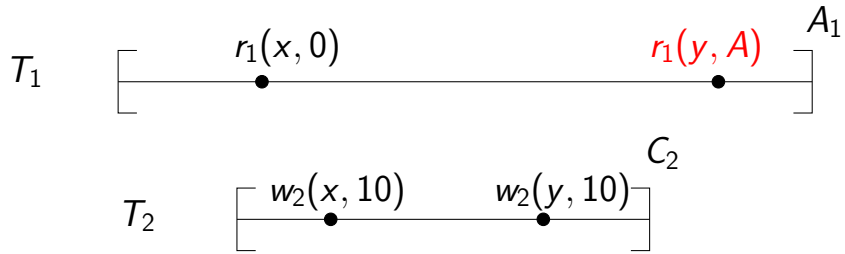


Figure 3: Opaque History H (T_1, T_2)

[a] Guerraoui, R., Kapalka, M.: On the Correctness of Transactional Memory. PPoPP, 2008

Outline

Object-based STM (OSTM) [b] [c] [d]

Introduction

- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read and writes which act upon memory locations.

[b] Herlihy, M., Koskinen, E., Transactional boosting: a methodology for highly-concurrent transactional objects. PPoPP'08

[c] Hassan, Ahmed and Palmieri, Roberto and Ravindran, Binoy, Optimistic Transactional Boosting, PPoPP'14

[d] Peri, S., Singh, A., Somani, A., Efficient means of Achieving Composability using Transactional Memory, NETYS'18

Object-based STM (OSTM) [b] [c] [d]

Introduction

- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read and writes which act upon memory locations.
- OSTM for sets export *t_begin()*, *t_insert()*, *t_delete()*, *t_lookup()*, *and tryC()*.

[b] Herlihy, M., Koskinen, E., Transactional boosting: a methodology for highly-concurrent transactional objects. PPoPP'08

[c] Hassan, Ahmed and Palmieri, Roberto and Ravindran, Binoy, Optimistic Transactional Boosting, PPoPP'14

[d] Peri, S., Singh, A., Somani, A., Efficient means of Achieving Composability using Transactional Memory, NETYS'18

Object-based STM (OSTM) [b] [c] [d]

Introduction

- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read and writes which act upon memory locations.
- OSTM for sets export *t_begin()*, *t_insert()*, *t_delete()*, *t_lookup()*, *and tryC()*.
- OSTM for queue exports *t_begin()*, *t_enqueue()*, *t_dequeue()*, *and tryC()*.

[b] Herlihy, M., Koskinen, E., Transactional boosting: a methodology for highly-concurrent transactional objects. PPoPP'08

[c] Hassan, Ahmed and Palmieri, Roberto and Ravindran, Binoy, Optimistic Transactional Boosting, PPoPP'14

[d] Peri, S., Singh, A., Somani, A., Efficient means of Achieving Composability using Transactional Memory, NETYS'18

Object-based STM (OSTM) [b] [c] [d]

Introduction

- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read and writes which act upon memory locations.
- OSTM for sets export *t_begin()*, *t_insert()*, *t_delete()*, *t_lookup()*, *and tryC()*.
- OSTM for queue exports *t_begin()*, *t_enqueue()*, *t_dequeue()*, *and tryC()*.
- OSTM for stack exports *t_begin()*, *t_push()*, *t_pop()*, *and tryC()*.

[b] Herlihy, M., Koskinen, E., Transactional boosting: a methodology for highly-concurrent transactional objects. PPoPP'08

[c] Hassan, Ahmed and Palmieri, Roberto and Ravindran, Binoy, Optimistic Transactional Boosting, PPoPP'14

[d] Peri, S., Singh, A., Somani, A., Efficient means of Achieving Composability using Transactional Memory, NETYS'18

Outline

Motivation towards OSTMs

Problem with read-write STMs (RWSTMs)

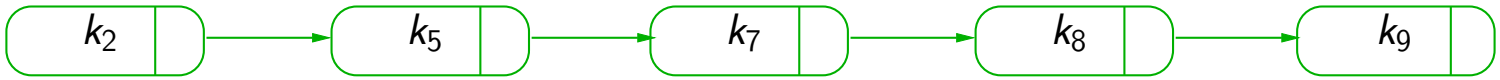


Figure 4: A concurrent list

Motivation towards OSTMs

Problem with read-write STMs (RWSTMs)

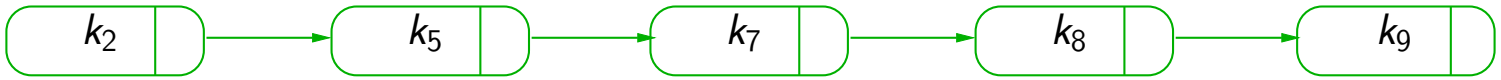


Figure 4: A concurrent list

T_1: lookup(k_5), lookup(k_8) and T_2: delete(k_7)

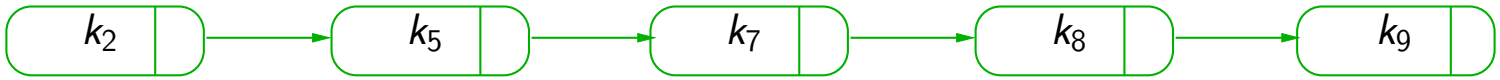


Figure 4: A concurrent list

T_1: lookup(k_5), lookup(k_8) and T_2: delete(k_7)

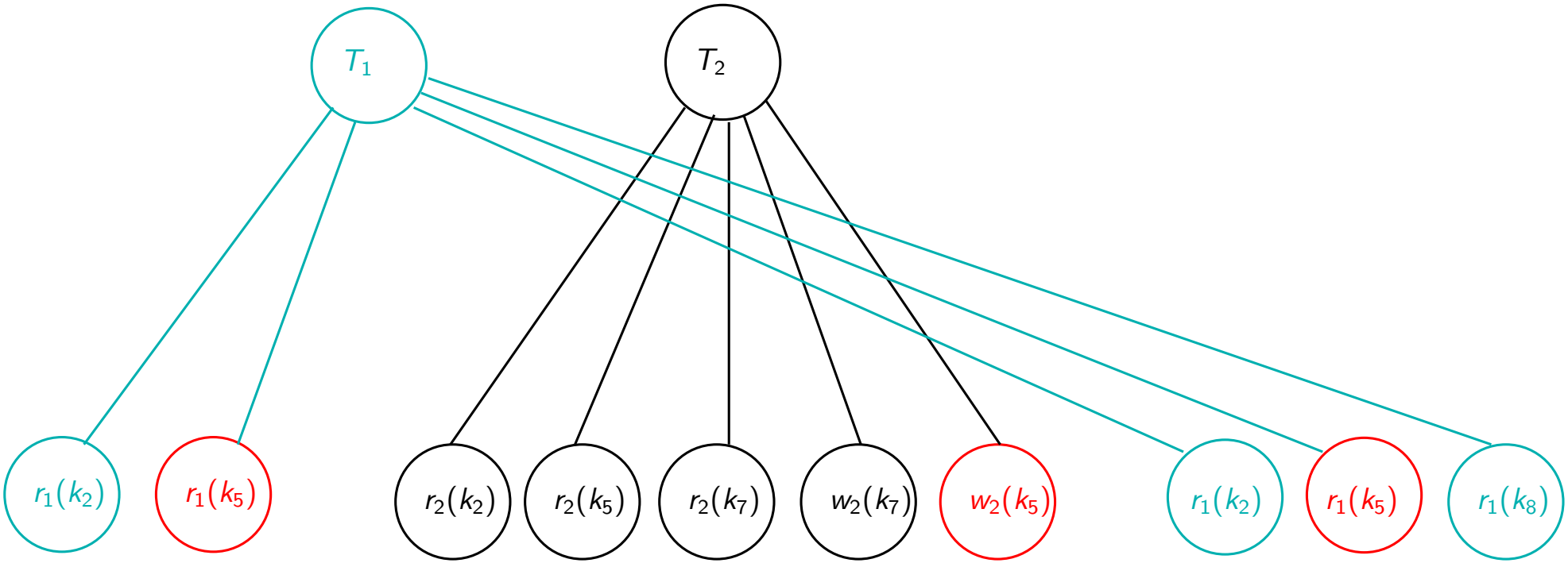


Figure 5: Tree Structure : conflicts are ($r_1(k_5)$, $w_2(k_5)$) and ($w_2(k_5)$, $r_1(k_5)$)

Motivation towards OSTMs Cont'd..

Problem with read-write STMs (RWSTMs)

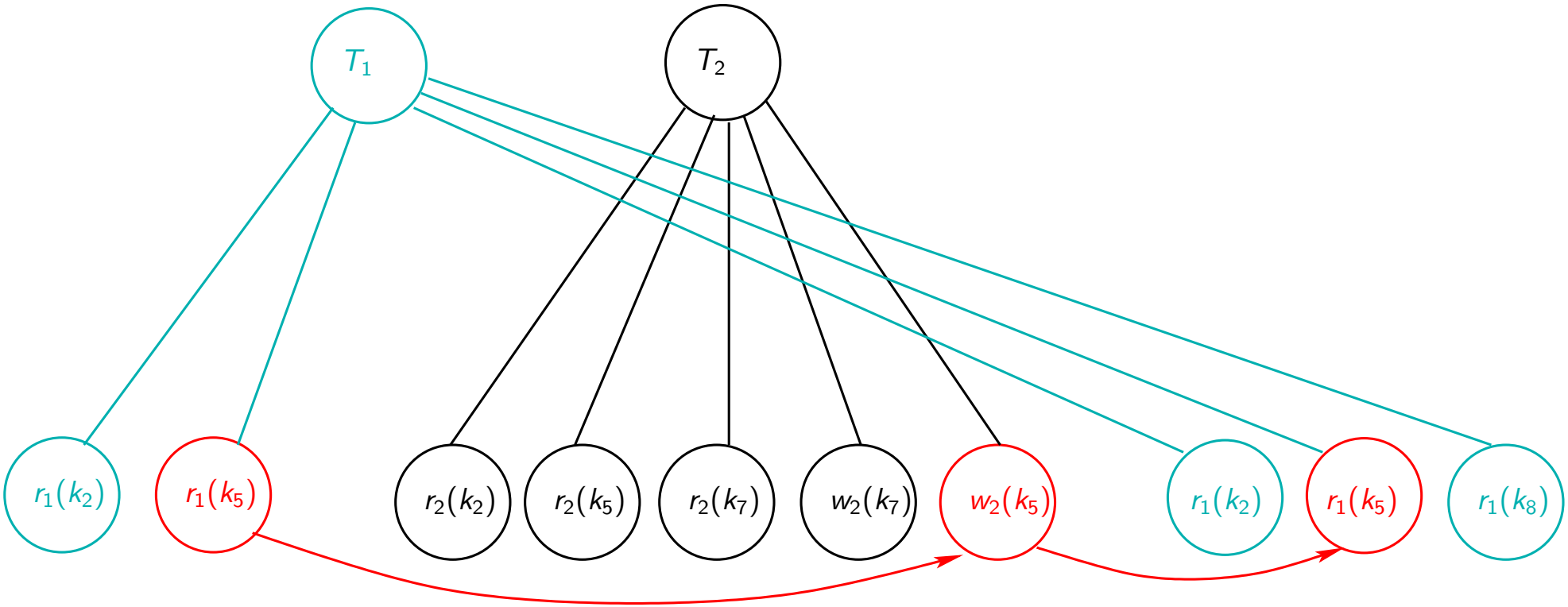


Figure 6: Tree Structure

Motivation towards OSTMs Cont'd..

Problem with read-write STMs (RWSTMs)

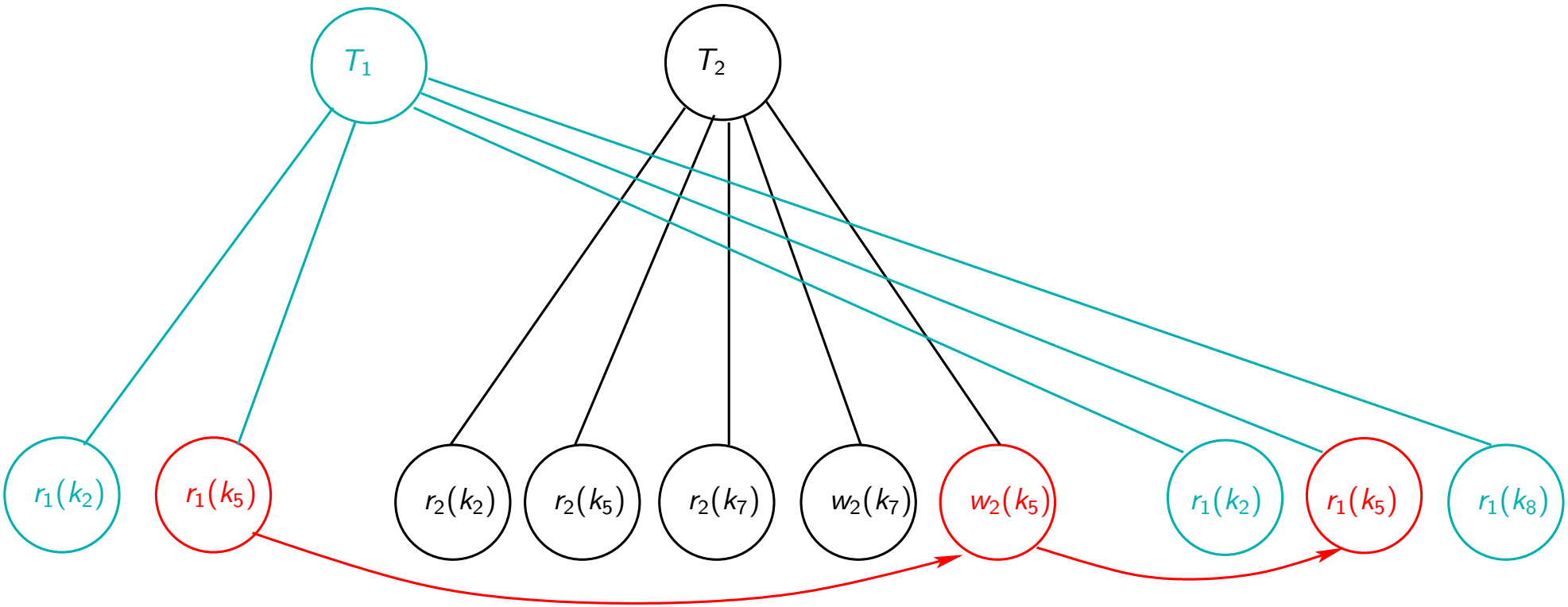


Figure 6: Tree Structure

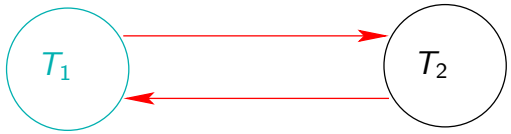


Figure 7: Cycle (Not Serial)

Motivation towards OSTMs Cont'd..

Problem with read-write STMs (RWSTMs)

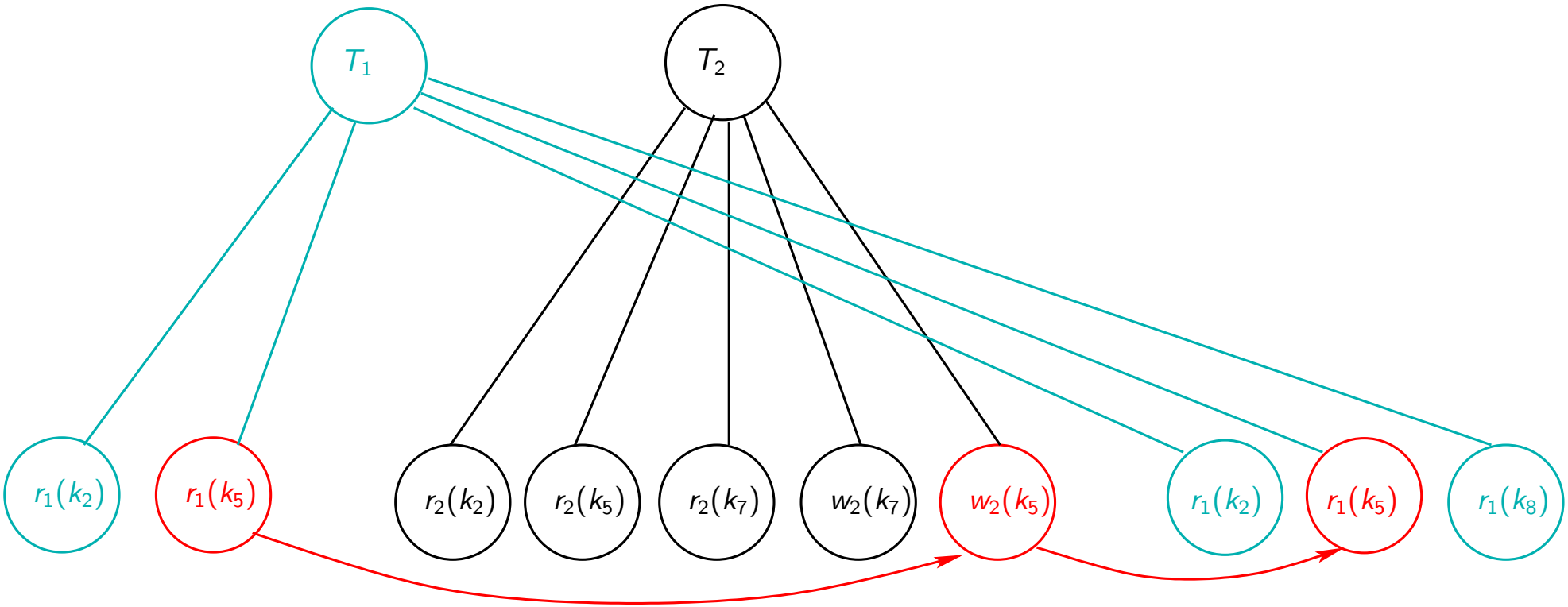


Figure 6: Tree Structure

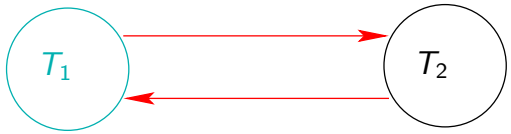


Figure 7: Cycle (Not Serial)

Schedule cannot be accepted by a RWSTM.

Motivation towards OSTMs Cont'd..

Execution at layer-1

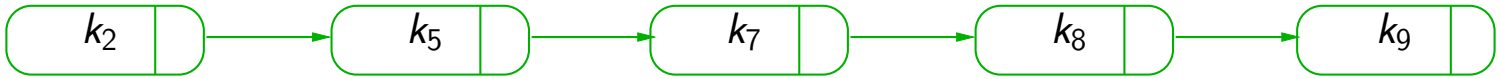


Figure 8: A concurrent list

[e] Gerhard Weikum and Gottfried Vossen. 2001. Transactional Information Systems Book

Motivation towards OSTMs Cont'd..

Execution at layer-1

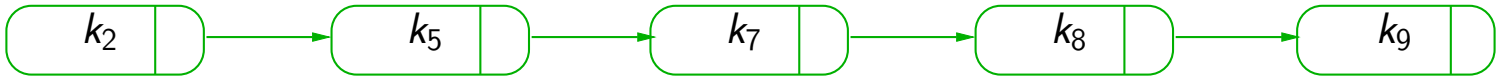


Figure 8: A concurrent list

T1: lookup(k_5), lookup(k_8) and T_2: delete(k_7)

[e] Gerhard Weikum and Gottfried Vossen. 2001. Transactional Information Systems Book

Motivation towards OSTMs Cont'd..

Execution at layer-1

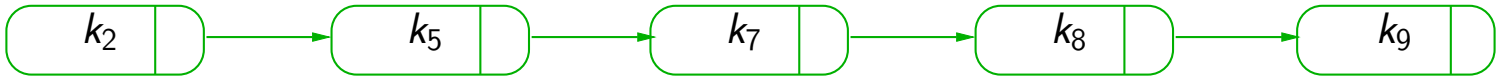


Figure 8: A concurrent list

T1: lookup(k_5), lookup(k_8) and T_2: delete(k_7)

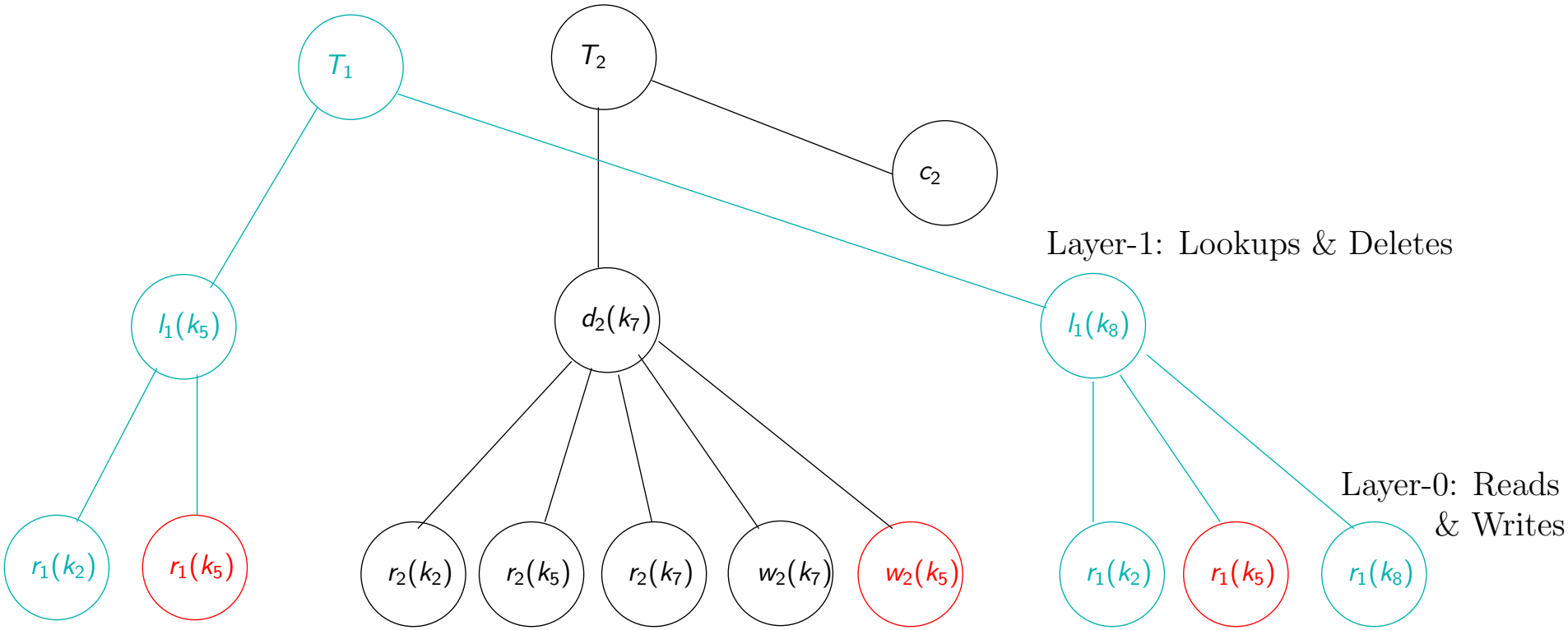


Figure 9: Tree Structure : no conflict at Layer-1[e]

[e] Gerhard Weikum and Gottfried Vossen. 2001. Transactional Information Systems Book

Motivation towards OSTMs Cont'd..

Execution at layer-1

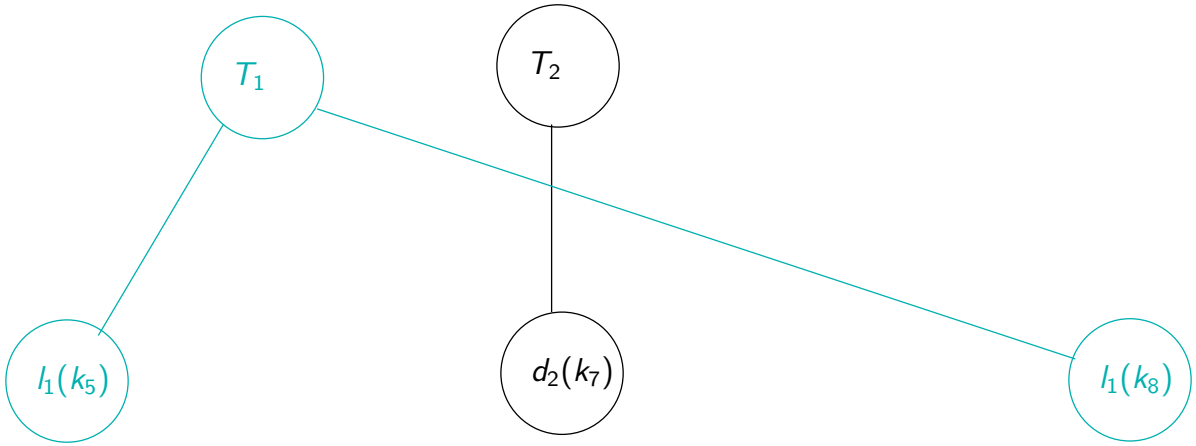


Figure 10: Pruned Tree

Motivation towards OSTMs Cont'd..

Execution at layer-1

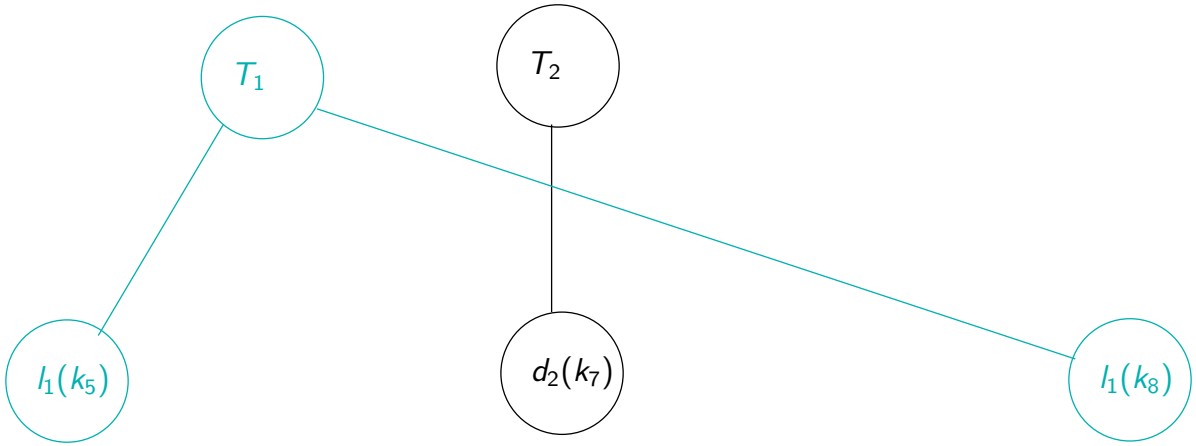


Figure 10: Pruned Tree

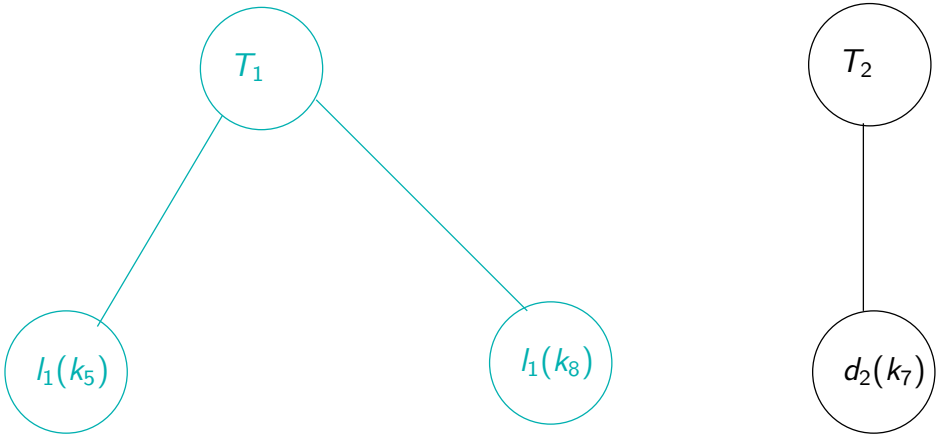


Figure 11: Sequential Schedule

Motivation towards OSTMs Cont'd..

Execution at layer-1

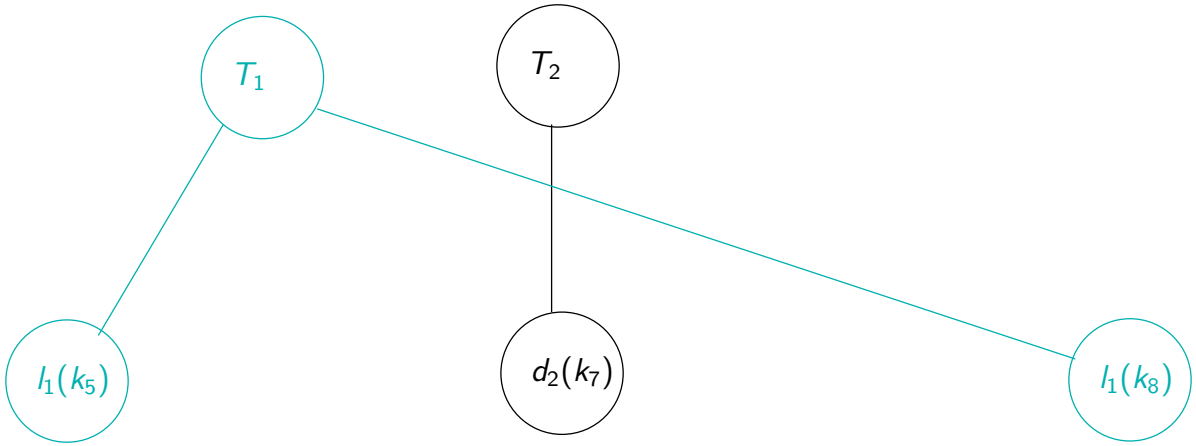


Figure 10: Pruned Tree

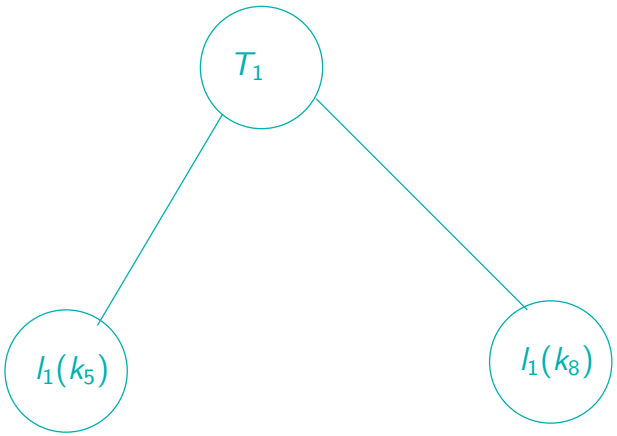


Figure 11: Sequential Schedule

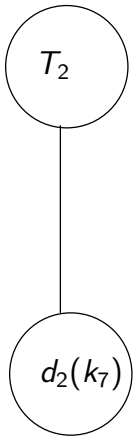


Figure 12: Serial History

Outline

Motivation towards Multi-Version OSTMs (MV-OSTMs)

Limitation of OSTMs

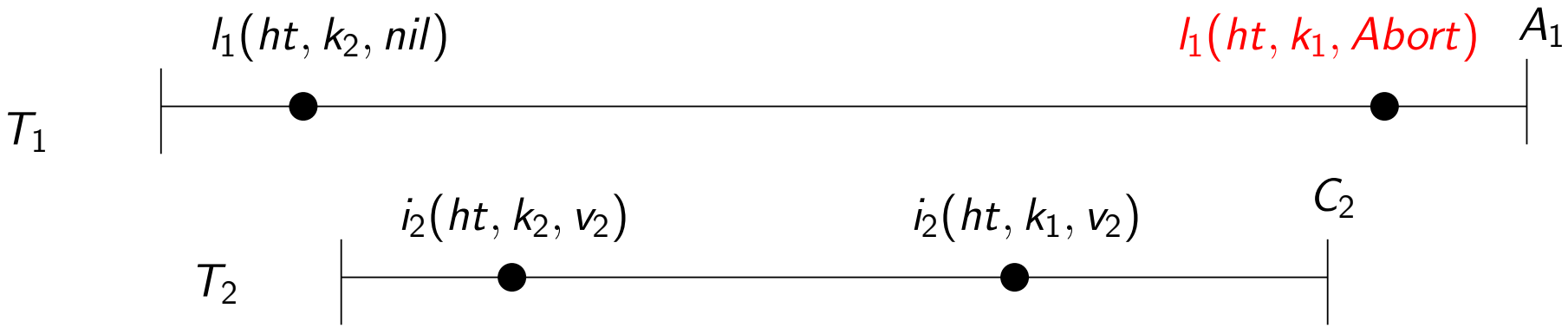


Figure 13: Single-version OSTM (SV-OSTM)

Motivation towards MV-OSTMs Cont'd..

Advantages of multi-version[f] over single-version RWSTMs

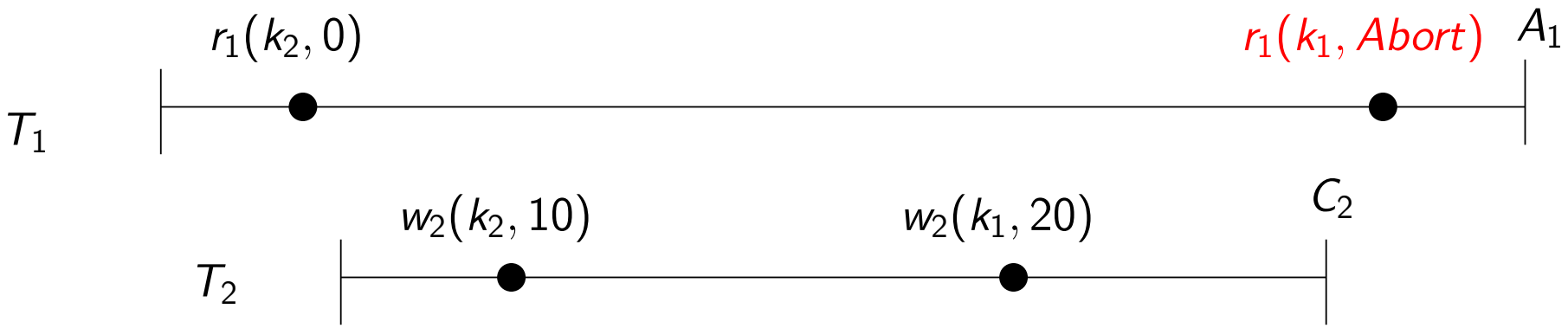


Figure 14: Single-version RWSTM (SV-RWSTM)

[f] Kumar P., Peri S., and K. Vidyasankar. A TimeStamp Based Multi-version STM Algorithm. ICDCN, 2014

Motivation towards MV-OSTMs Cont'd..

Advantages of multi-version[f] over single-version RWSTMs

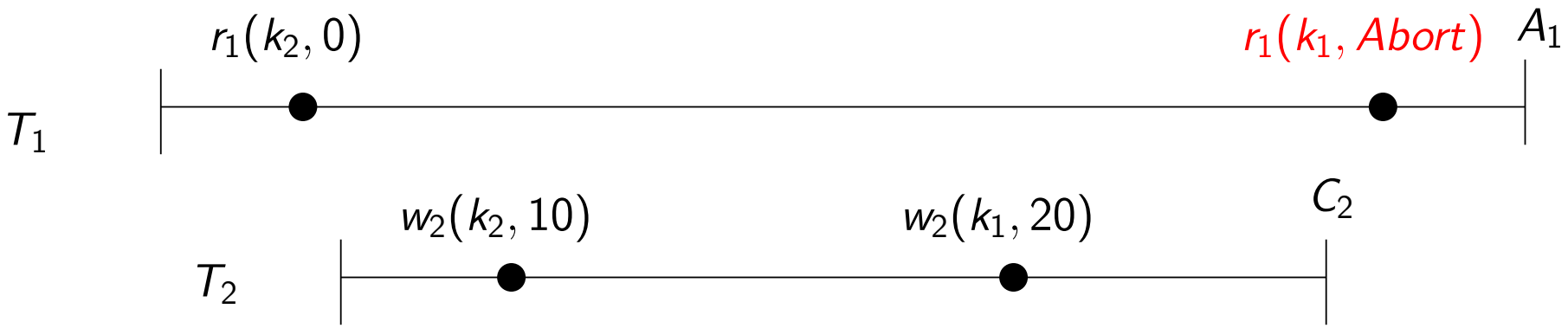


Figure 14: Single-version RWSTM (SV-RWSTM)

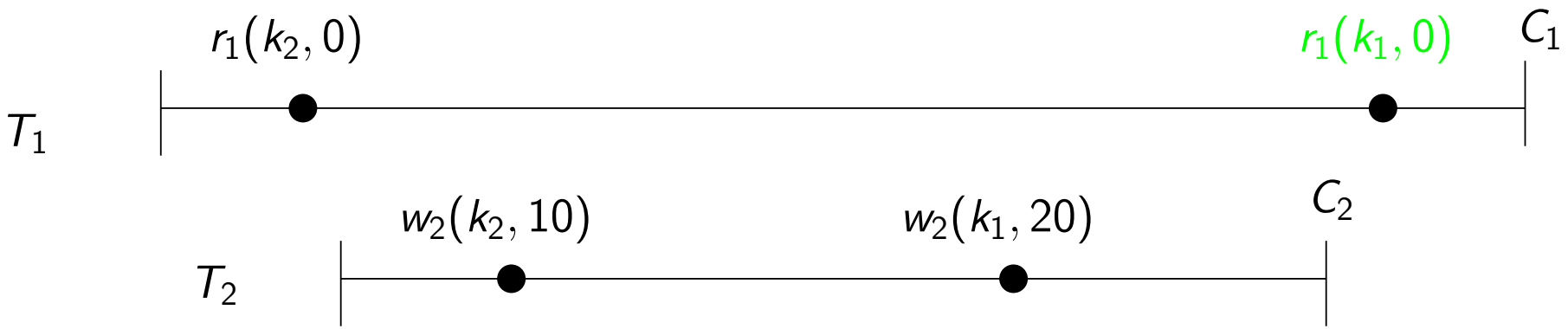


Figure 15: Multi-version RWSTM (MV-RWSTM) : (T_1, T_2)

[f] Kumar P., Peri S., and K. Vidyasankar. A TimeStamp Based Multi-version STM Algorithm. ICDCN, 2014

Motivation towards MV-OSTMs Cont'd..

Advantages of multi-version[g] over single-version OSTM

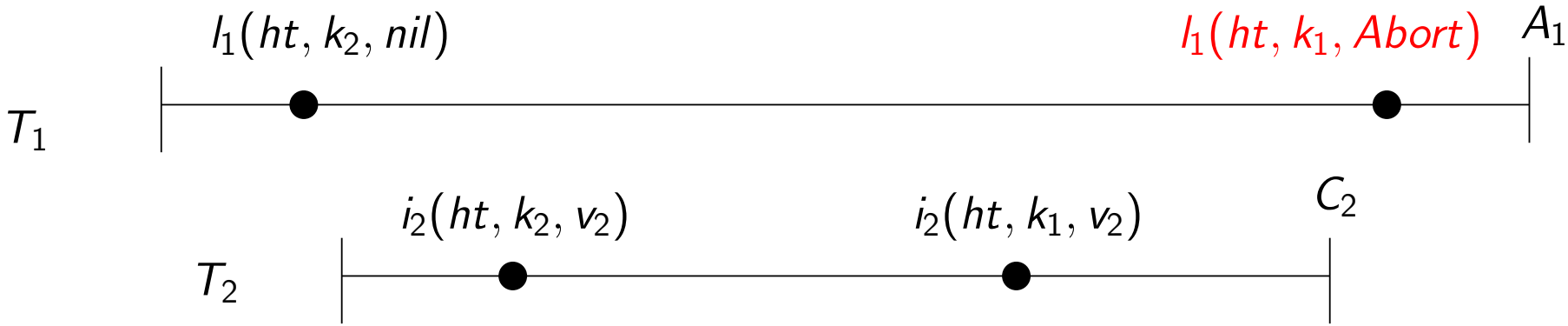


Figure 16: Single-version OSTM (SV-OSTM)

[g] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A., An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems, SSS'18.

Motivation towards MV-OSTMs Cont'd..

Advantages of multi-version[g] over single-version OSTM

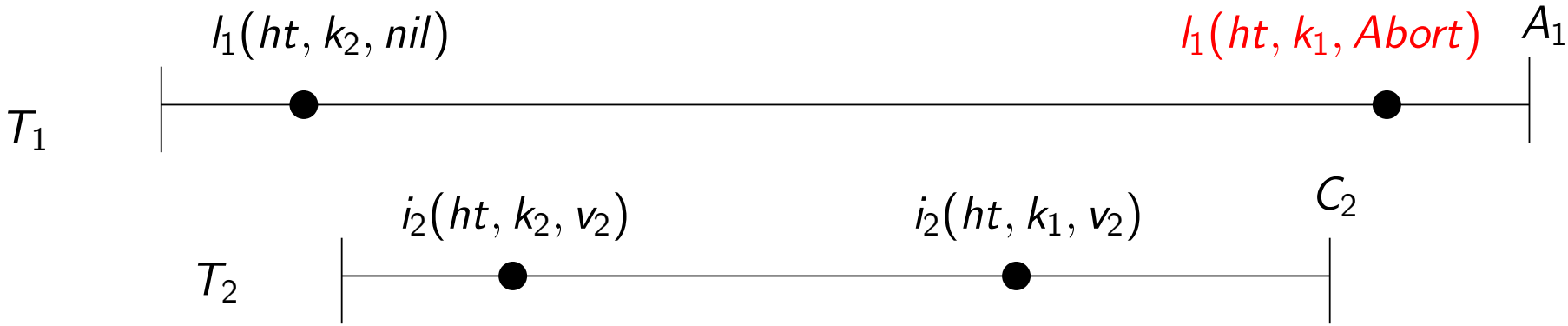


Figure 16: Single-version OSTM (SV-OSTM)

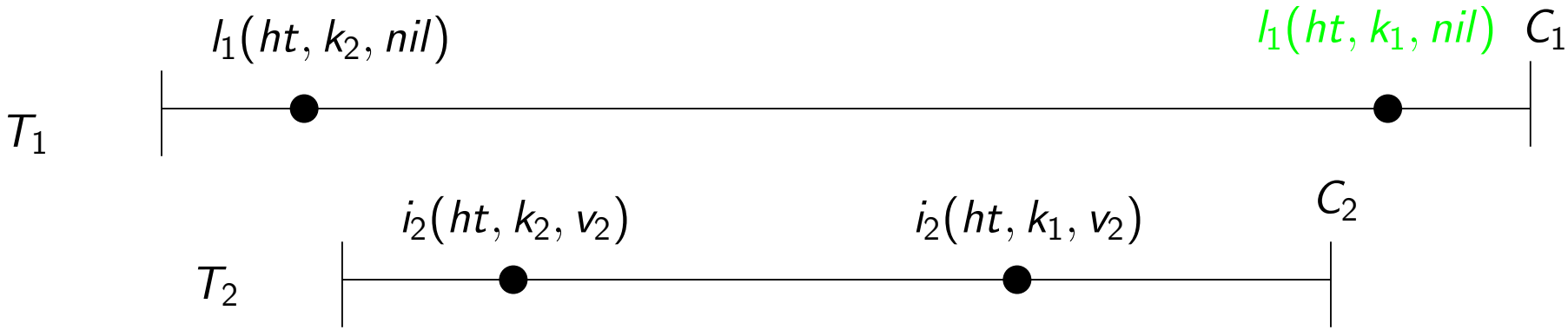


Figure 17: Multi-version OSTM (MV-OSTM) : (T_1, T_2)

[g] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A., An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems, SSS'18.

Outline

Progress Guarantee in SV-OSTMs and MV-OSTMs

Starvation in STMs

- None of the Single and Multi-Version OSTMs provide starvation-freedom.

Progress Guarantee in SV-OSTMs and MV-OSTMs

Starvation in STMs

- None of the Single and Multi-Version OSTMs provide starvation-freedom.

Algorithm Insert(LL, v): Invoked by a thread to insert a value v into a linked-list LL. This method is implemented using transactions.

```
1:  while (true) do
2:      id = t_begin();
3:      ...
4:      v = t_lookup(id, x);
5:      ...
6:      t_insert(id, x, v');
7:      ...
8:      ret = tryC(id);
9:      if (ret == success) then
10:         break;
11:     end if
12: end while
```

Progress Guarantee in SV-OSTMs and MV-OSTMs

Starvation in STMs

- None of the Single and Multi-Version OSTMs provide starvation-freedom.

Algorithm Insert(LL, v): Invoked by a thread to insert a value v into a linked-list LL. This method is implemented using transactions.

```
1:  while (true) do
2:      id = t_begin();
3:      ...
4:      v = t_lookup(id, x);
5:      ...
6:      t_insert(id, x, v');
7:      ...
8:      ret = tryC(id);
9:      if (ret == success) then
10:         break;
11:     end if
12: end while
```

Issue: Starvation can occur.

Progress Guarantee in SV-OSTM and MV-OSTM Cont'd..

Definition of Starvation-Freedom in STMs

- **Definition of Starvation-Freedom:** An STM system is said to be *starvation-free* if a thread invoking a non-parasitic[h] transaction T_i gets opportunity to retry T_i on every abort, due to the presence of a fair scheduler, then T_i will eventually commit.

[h] Bushkov, V., Guerraoui, R., Kapalka, M.: On the liveness of transactional memory. In: PODC'12.Bushkov, V., Guerraoui, R., Kapalka, M.: On the liveness of transactional memory. In: PODC'12.

Progress Guarantee in SV-OSTM and MV-OSTM Cont'd..

Assumptions of our paper

Assumption (1)

We assume that in the absence of other concurrent conflicting transactions, every transaction will commit. In other words, (a) if a transaction T_i is executing in a system where other concurrent conflicting transactions are not present then T_i will not self-abort. (b) Transactions are not parasitic.

Progress Guarantee in SV-OSTM and MV-OSTM Cont'd..

Assumptions of our paper

Assumption (1)

We assume that in the absence of other concurrent conflicting transactions, every transaction will commit. In other words, (a) if a transaction T_i is executing in a system where other concurrent conflicting transactions are not present then T_i will not self-abort. (b) Transactions are not parasitic.

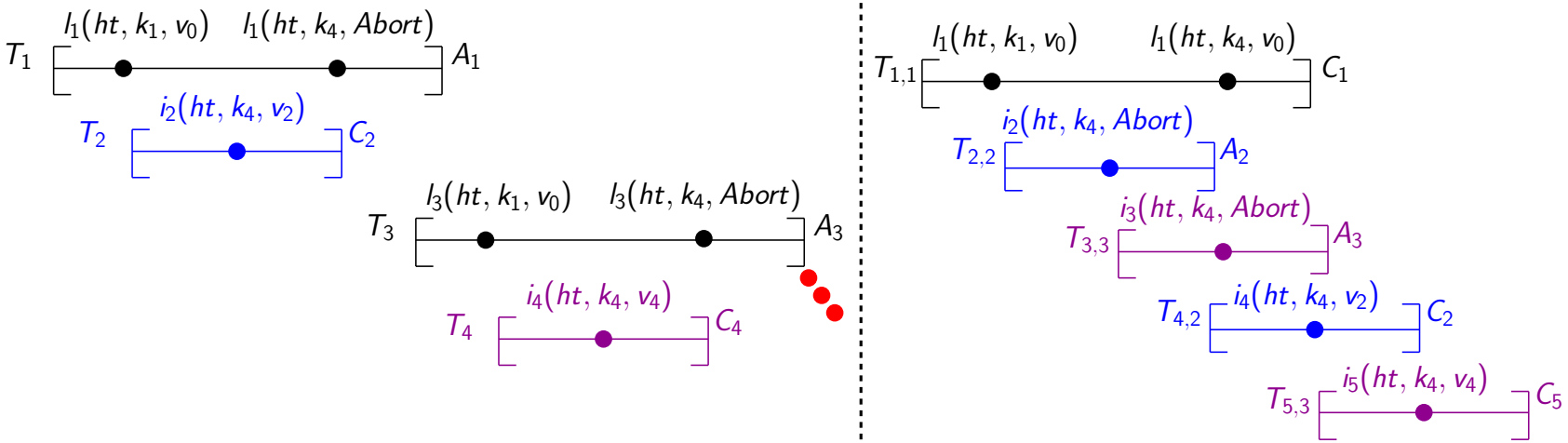
Assumption (2)

Bounded-Termination: *For any transaction T_i, invoked by a thread Th_x, the fair system scheduler ensures, in the absence of deadlocks, Th_x is given sufficient time on a CPU (and memory etc.) such that T_i terminates (either commits or aborts) in bounded time, L.*

Outline

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

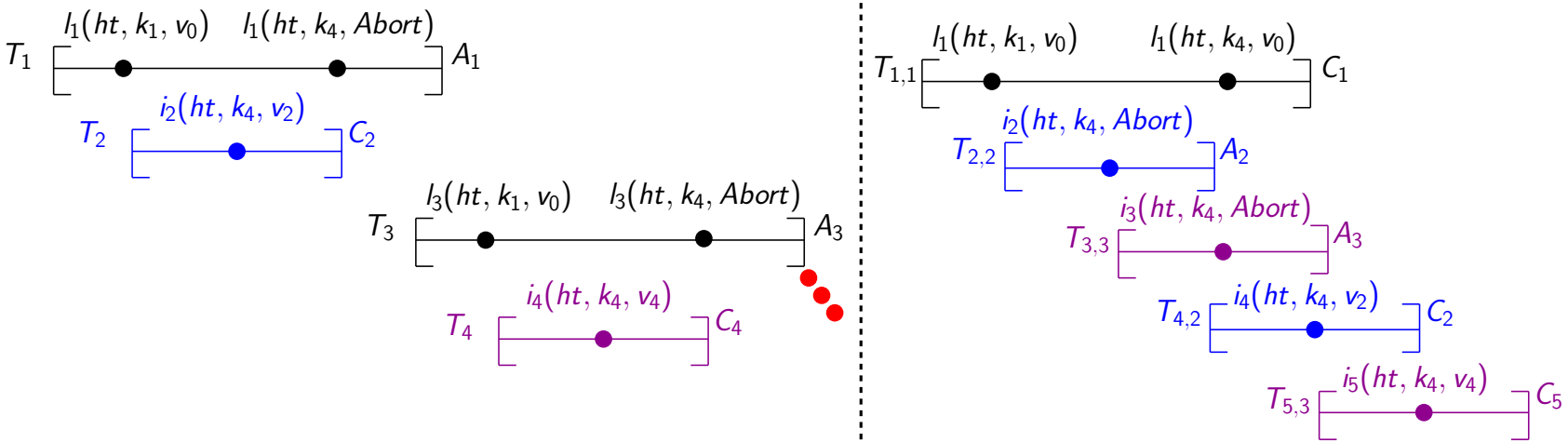
(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

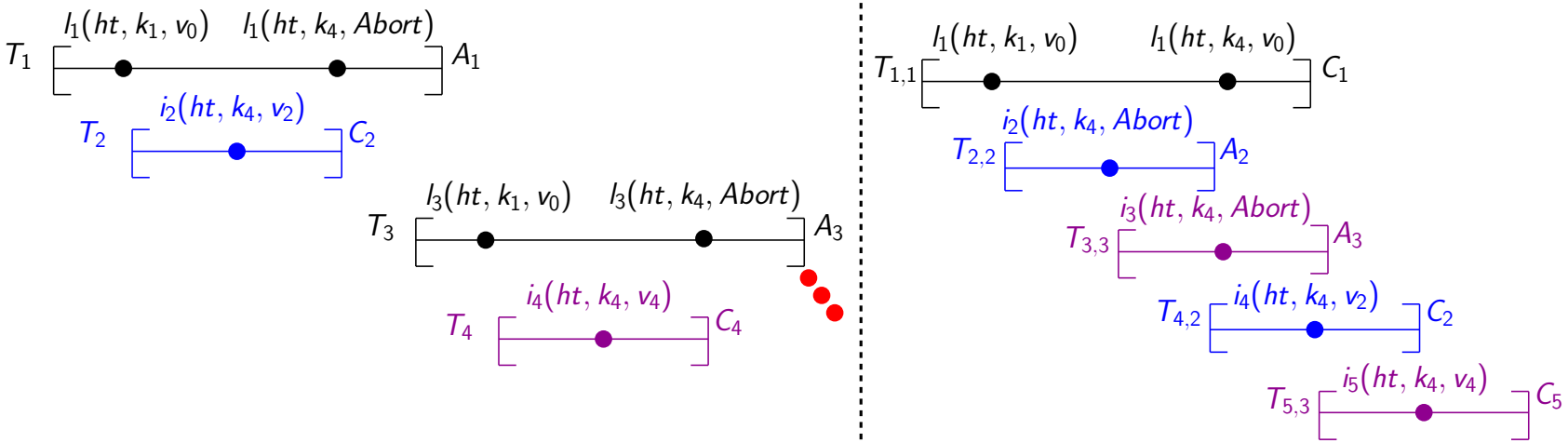
(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:
  1. *Current Timestamp (cts):* This is a unique timestamp alloted to T_i when it begins.

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

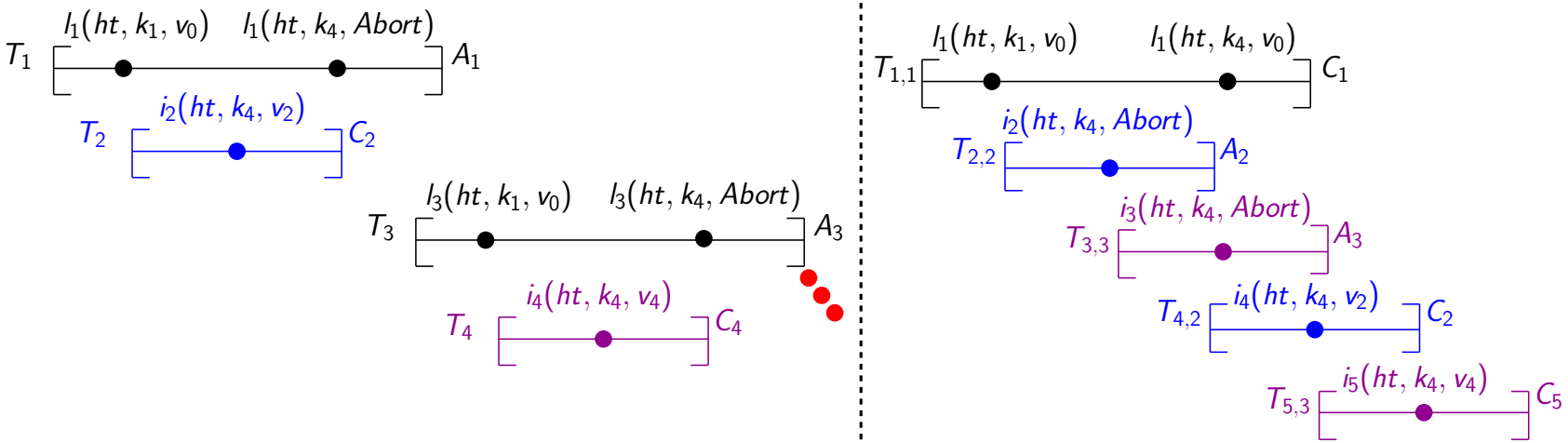
(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:
  1. *Current Timestamp (cts):* This is a unique timestamp alloted to T_i when it begins.
  2. *Initial Timestamp (its):* This is same as *cts* when a transaction T_i starts for the first time.

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

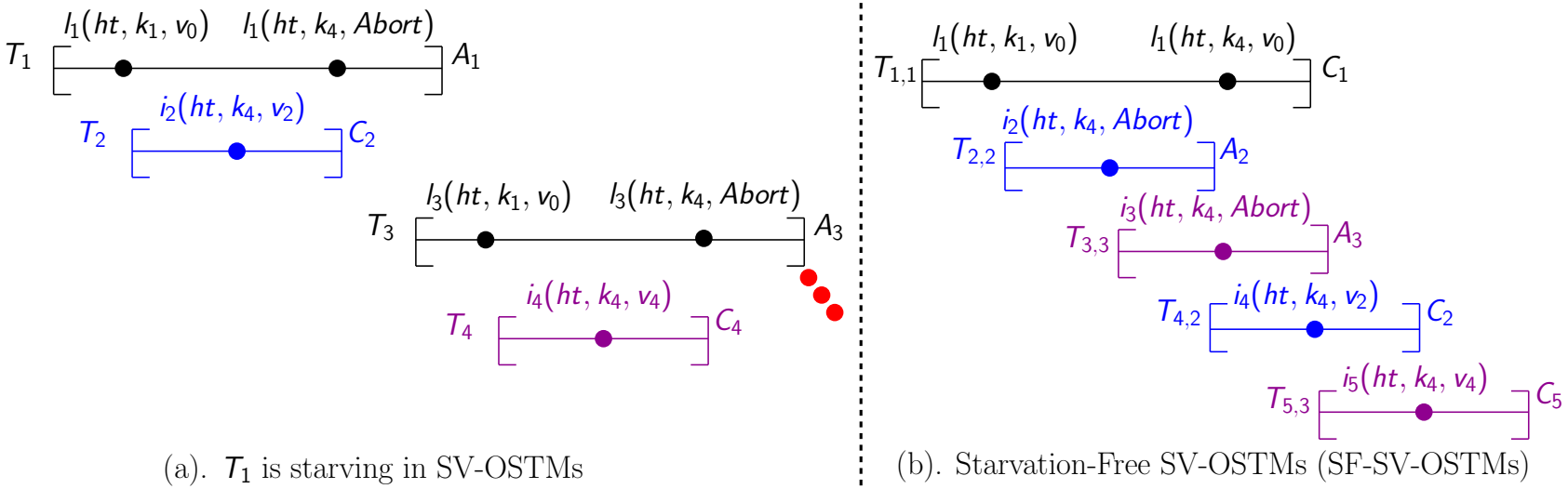
(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:
  1. *Current Timestamp (cts):* This is a unique timestamp alloted to T_i when it begins.
  2. *Initial Timestamp (its):* This is same as *cts* when a transaction T_i starts for the first time.
- When T_i aborts and restarts later, it gets a new *cts*.

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

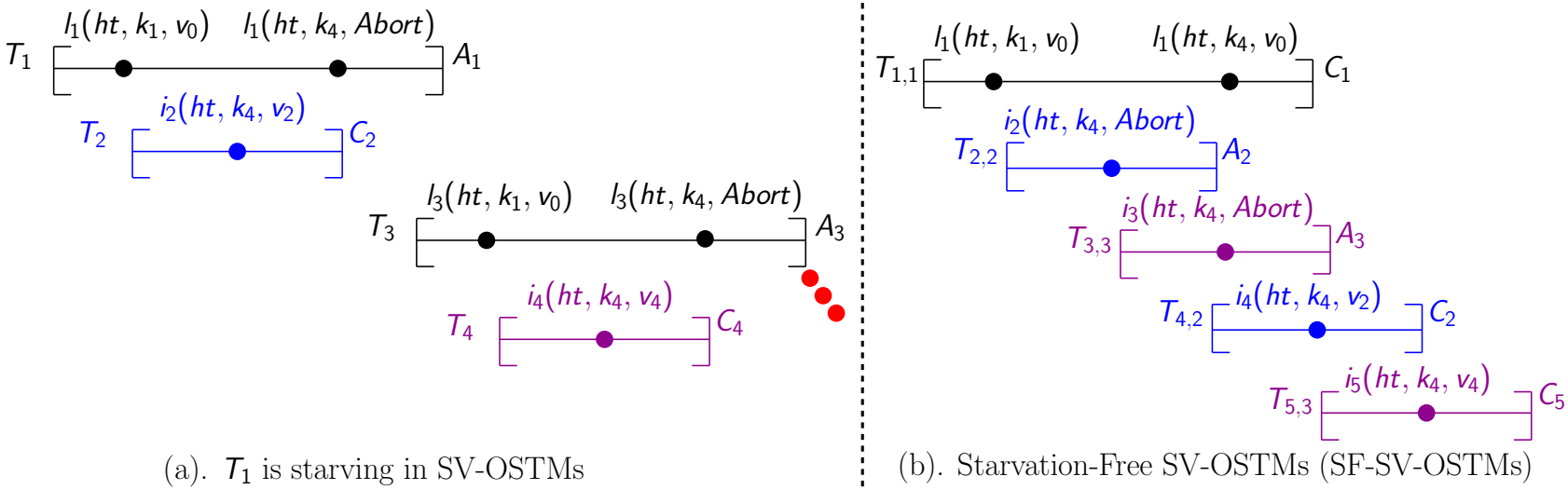
(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:
  1. *Current Timestamp (cts):* This is a unique timestamp alloted to T_i when it begins.
  2. *Initial Timestamp (its):* This is same as *cts* when a transaction T_i starts for the first time.
- When T_i aborts and restarts later, it gets a new *cts*.
- But it retains its original *cts* as *its*.

Starvation-Freedom in Single-Version OSTMs

SF-SV-OSTMs



(a). T_1 is starving in SV-OSTMs

(b). Starvation-Free SV-OSTMs (SF-SV-OSTMs)

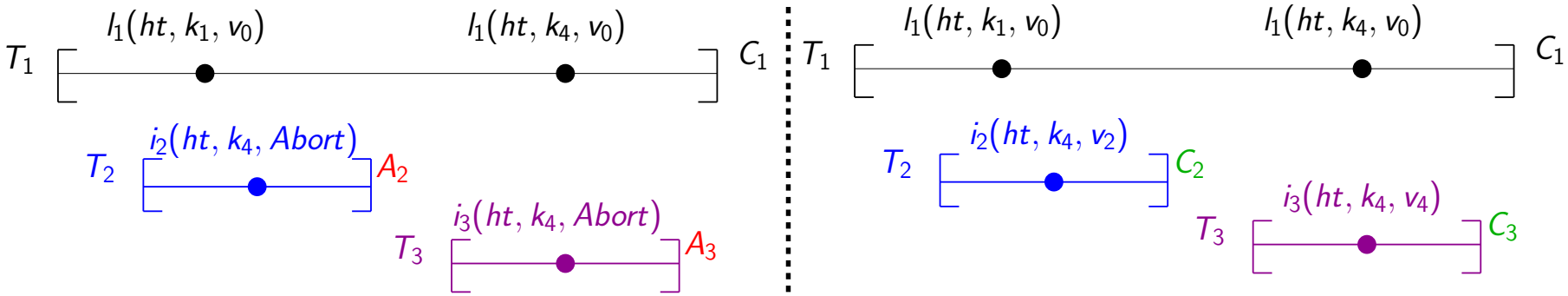
Figure 18: Advantage of SF-SV-OSTM over SV-OSTMs

- Each transaction T_i has two timestamps:
  1. *Current Timestamp (cts):* This is a unique timestamp alloted to T_i when it begins.
  2. *Initial Timestamp (its):* This is same as *cts* when a transaction T_i starts for the first time.
- When T_i aborts and restarts later, it gets a new *cts*.
- But it retains its original *cts* as *its*.
- The above procedure will repeat until T_i gets the lowest *its* when it will commit. Hence, achieves SF.

Outline

Starvation-Freedom in Multi-Version OSTMs

SF-MV-OSTMs

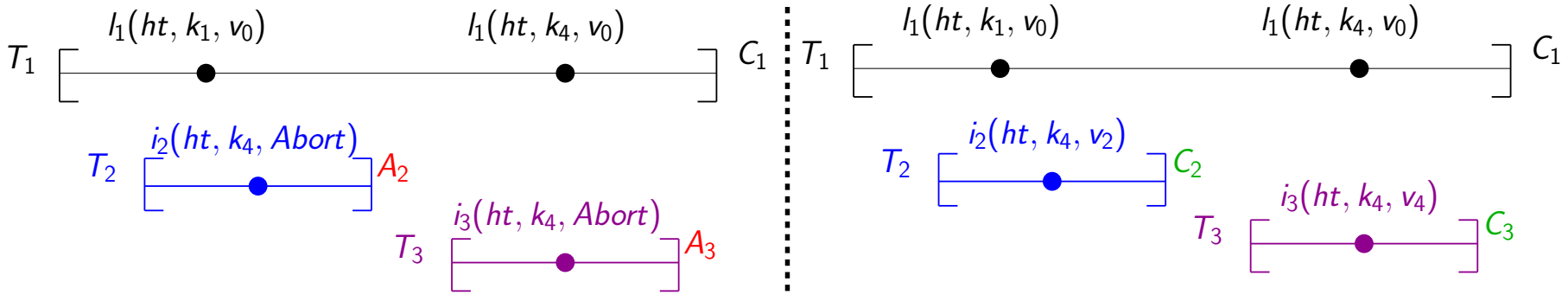


(a). Starvation-Free Single-Version OSTM (SF-SV-OSTM) (b). Starvation-Free Multi-Version OSTM (SF-MV-OSTM)

Figure 19: Benefits of Starvation-Free MV-OSTM over SF-SV-OSTM

Starvation-Freedom in Multi-Version OSTMs

SF-MV-OSTMs



(a). Starvation-Free Single-Version OSTM (SF-SV-OSTM) (b). Starvation-Free Multi-Version OSTM (SF-MV-OSTM)

Figure 19: Benefits of Starvation-Free MV-OSTM over SF-SV-OSTM

Takeaways

- Starvation-Free Multi-version OSTMs reduce number of aborts.
- Increase throughput.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Challenges for K-Version SF-MV-OSTMs

- Although multi-version OSTMs provide greater concurrency, they suffer from the cost of garbage collection.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Challenges for K-Version SF-MV-OSTMs

- Although multi-version OSTMs provide greater concurrency, they suffer from the cost of garbage collection.
  Solution: Maintaining bounded multi-version OSTMs.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Challenges for K-Version SF-MV-OSTMs

- Although multi-version OSTMs provide greater concurrency, they suffer from the cost of garbage collection.
  Solution: Maintaining bounded multi-version OSTMs.

- Achieving starvation-freedom while using only bounded versions is especially challenging given that a transaction may rely on the oldest version that has been removed.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Challenges for K-Version SF-MV-OSTMs

- Although multi-version OSTMs provide greater concurrency, they suffer from the cost of garbage collection.
  Solution: Maintaining bounded multi-version OSTMs.

- Achieving starvation-freedom while using only bounded versions is especially challenging given that a transaction may rely on the oldest version that has been removed.
  Solution: So, we propose Starvation-Free K-Version OSTM (SF-K-OSTM).

Outline

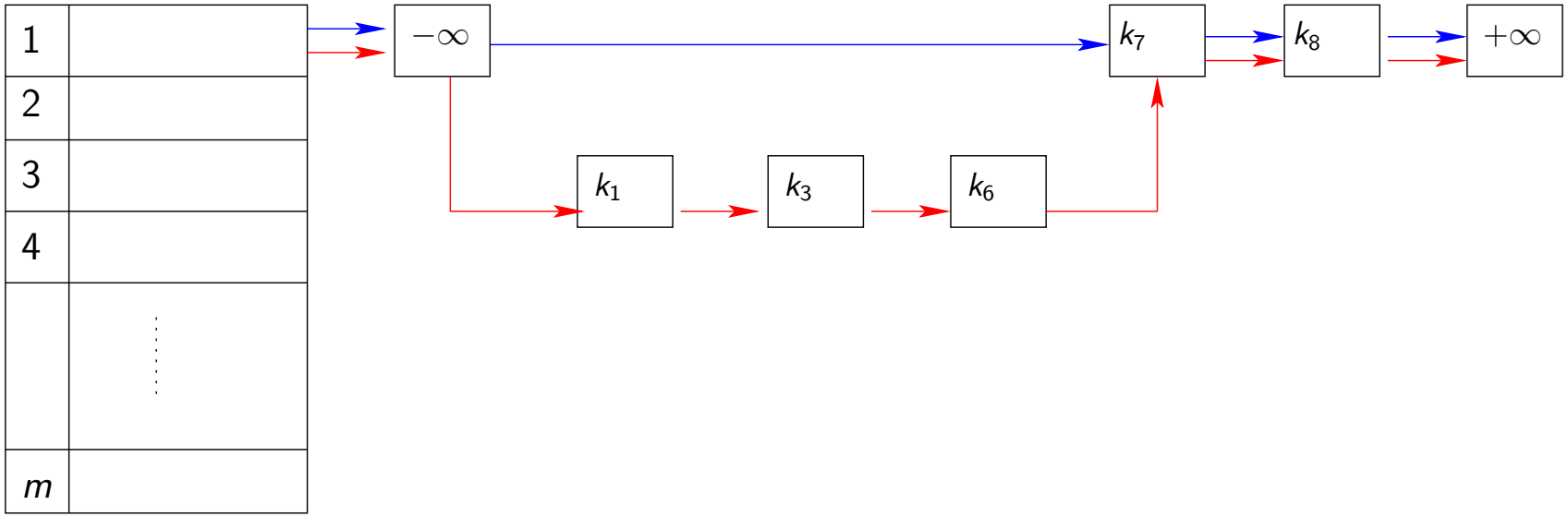


Figure 20: SF-K-OSTM design for Hash Table

Design and Data Structure of SF-K-OSTMs

High-level Design

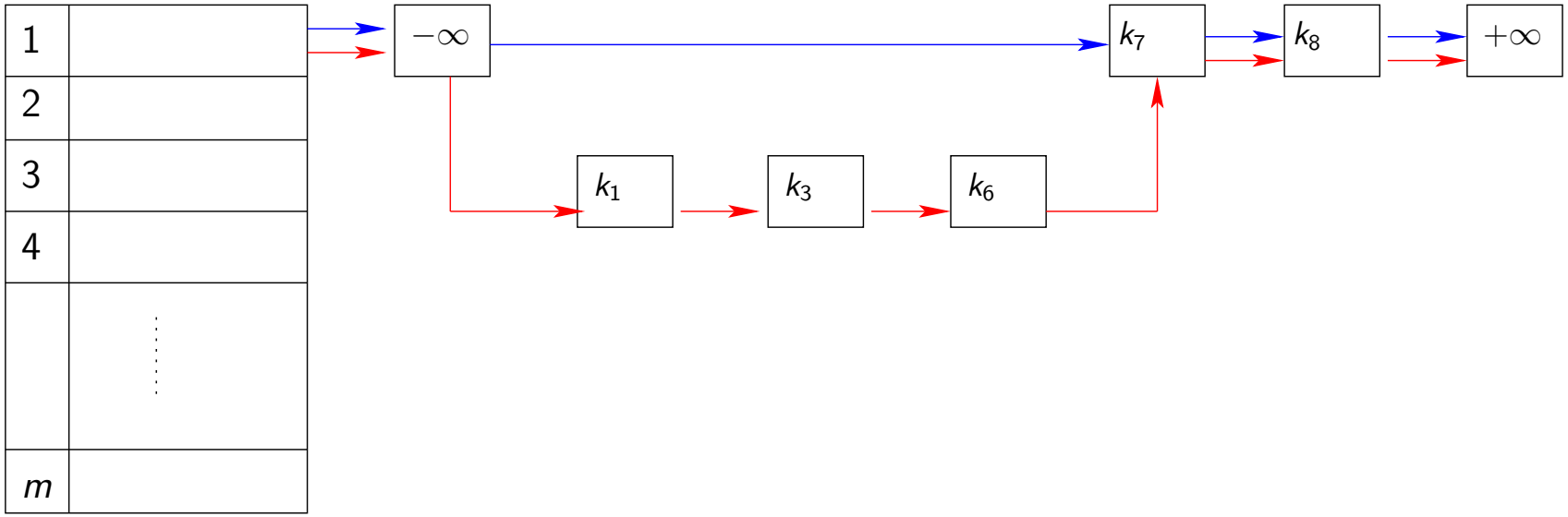


Figure 20: SF-K-OSTM design for Hash Table

SF-K-OSTMs export following methods:

t_begin(), t_lookup(), t_insert(), t_delete(), and tryC()

Design and Data Structure of SF-K-OSTMs

High-level Design

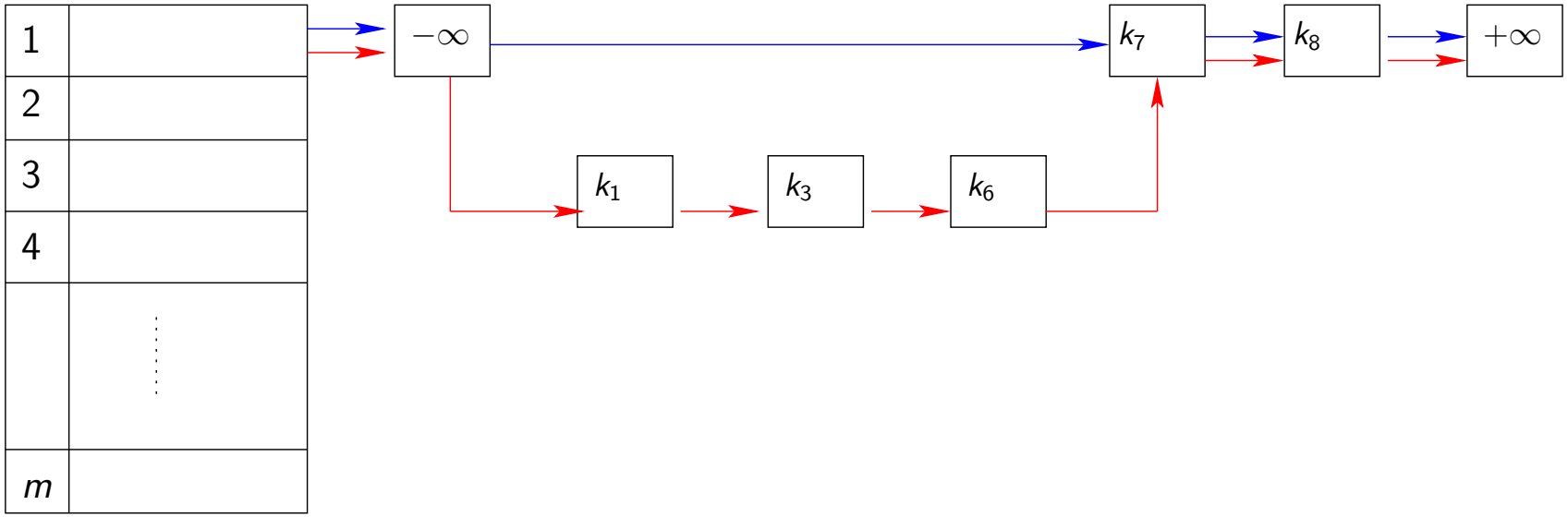


Figure 20: SF-K-OSTM design for Hash Table

SF-K-OSTMs export following methods:

t_begin(), t_lookup(), t_insert(), t_delete(), and tryC()

Design and Data Structure of SF-K-OSTMs Cont'd..

Maintains multiple version corresponding to each key

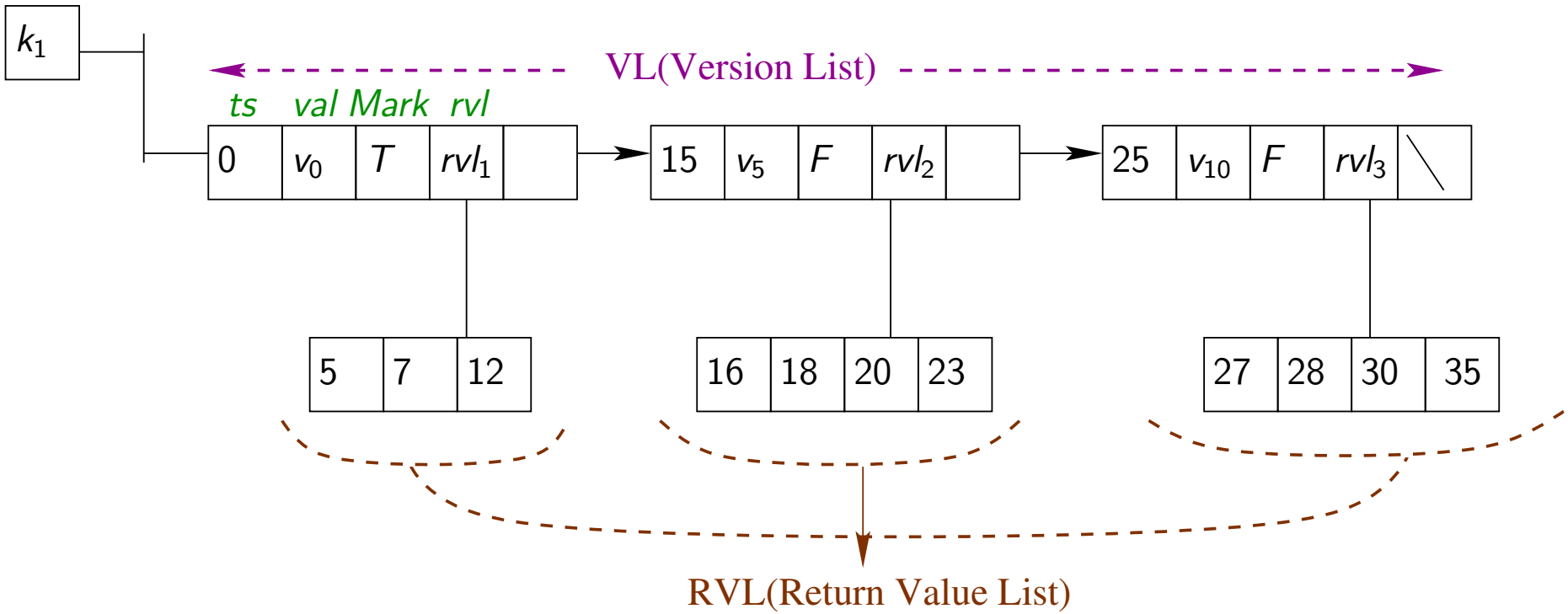


Figure 21: Data Structure for maintaining versions

Outline

Analysing K-MVOSTM

t_lookup() method

K-MVOSTM: K-Version Multi Version OSTM [i]

[i] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. In: SSS'18.

K-MVOSTM: K-Version Multi Version OSTM [i]

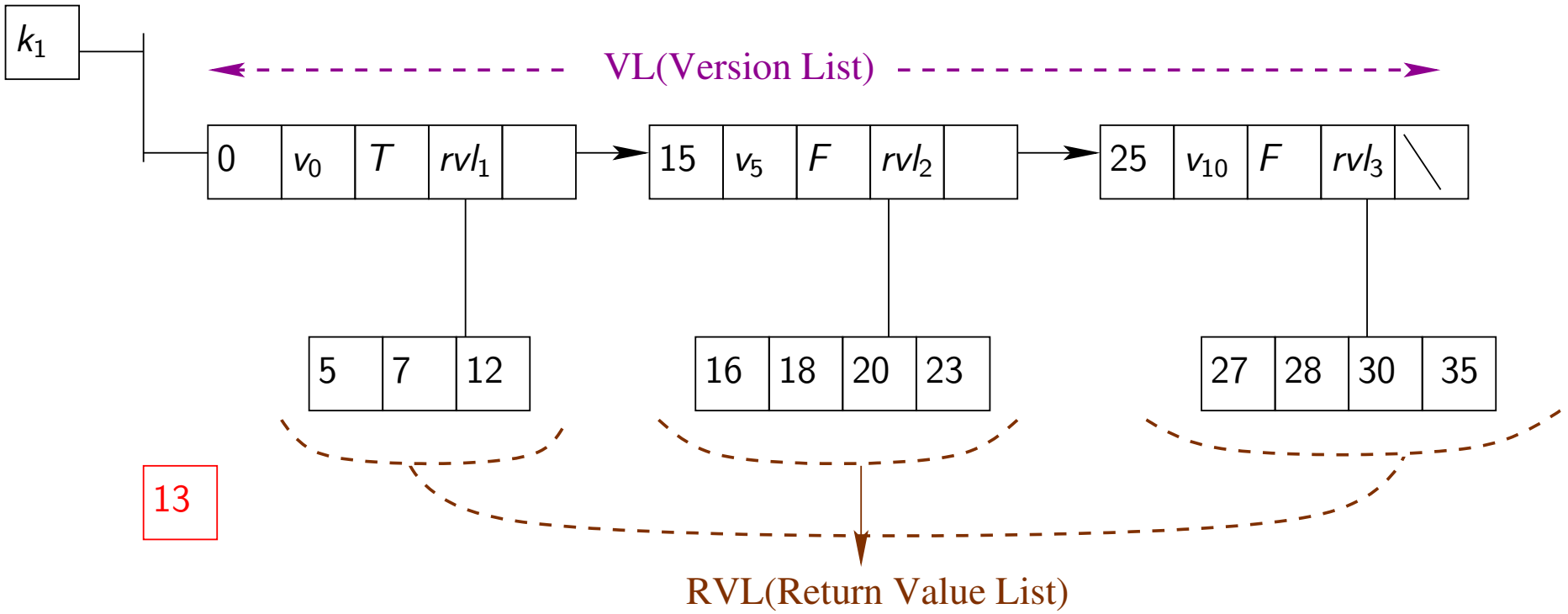


Figure 22: Lookup on key k_1 by T_{13}

[i] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. In: SSS'18.

Analysing K-MVOSTM

t_lookup() method

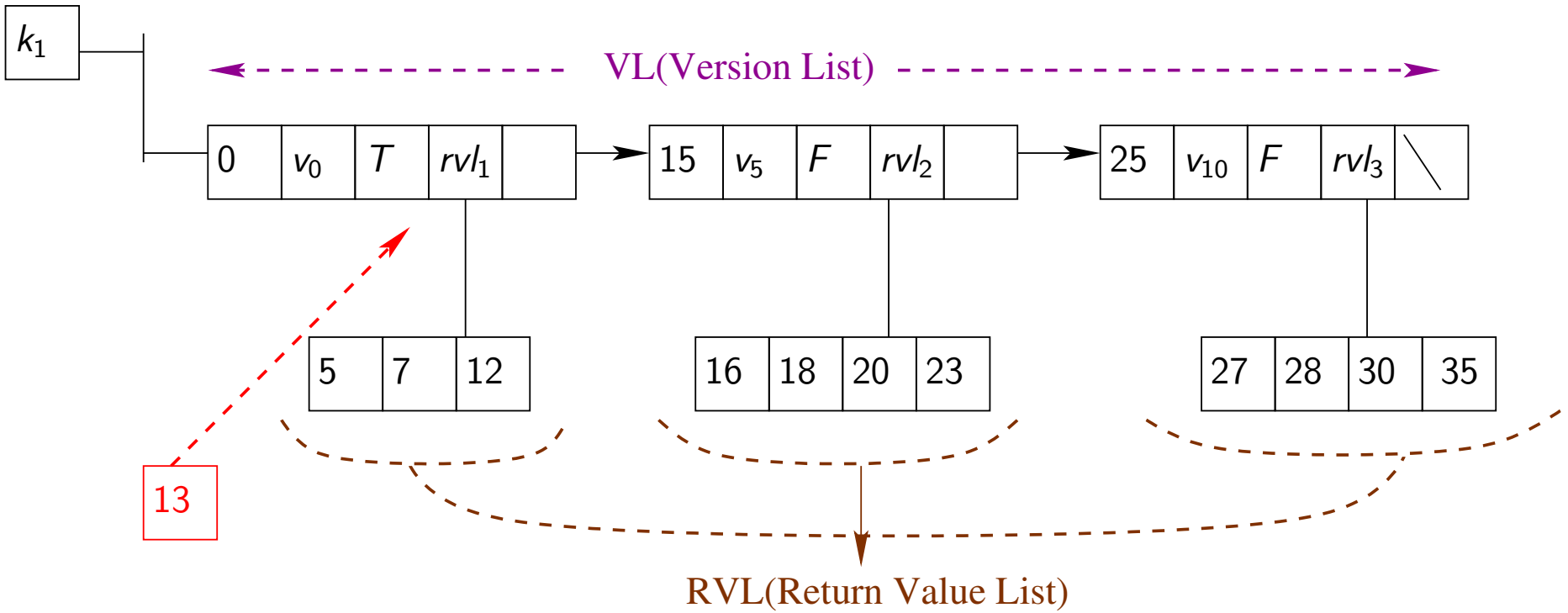


Figure 23: T_{13} searching appropriate place in version list of k_1

Analysing K-MVOSTM Cont'd...

t_lookup() method

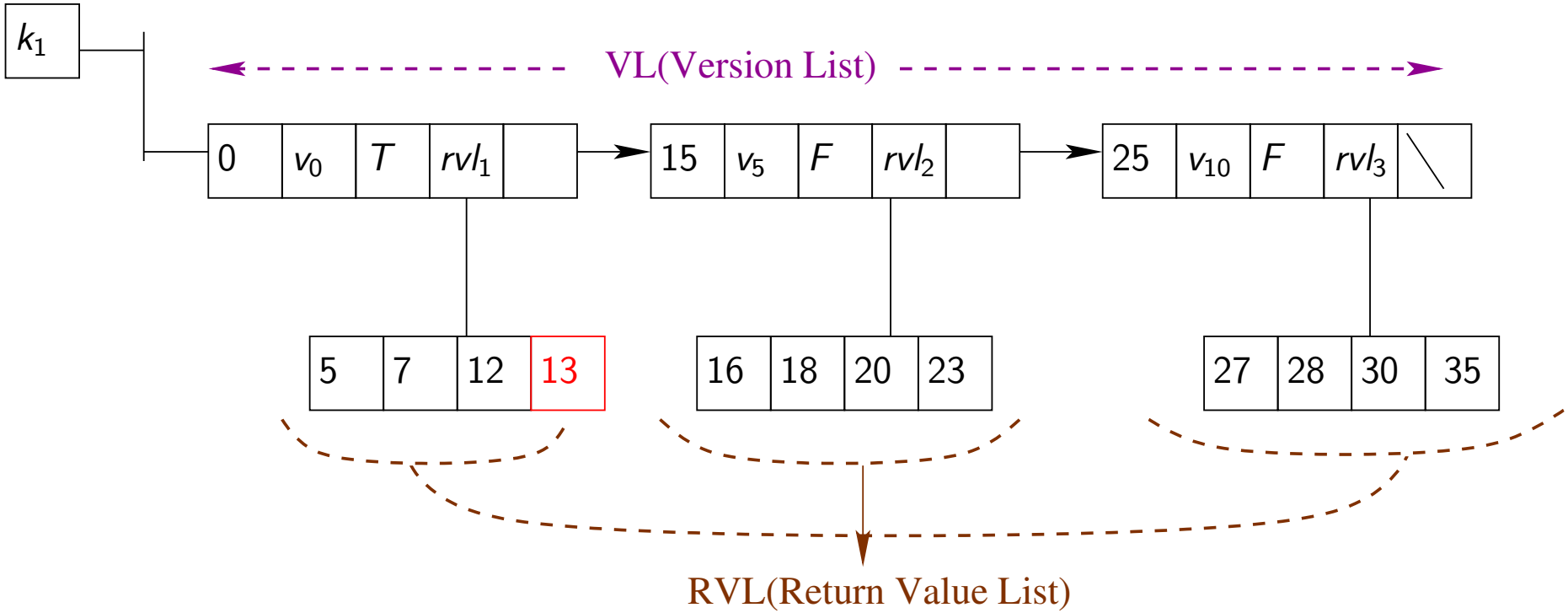


Figure 24: T_{13} successfully added into rvl_1

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

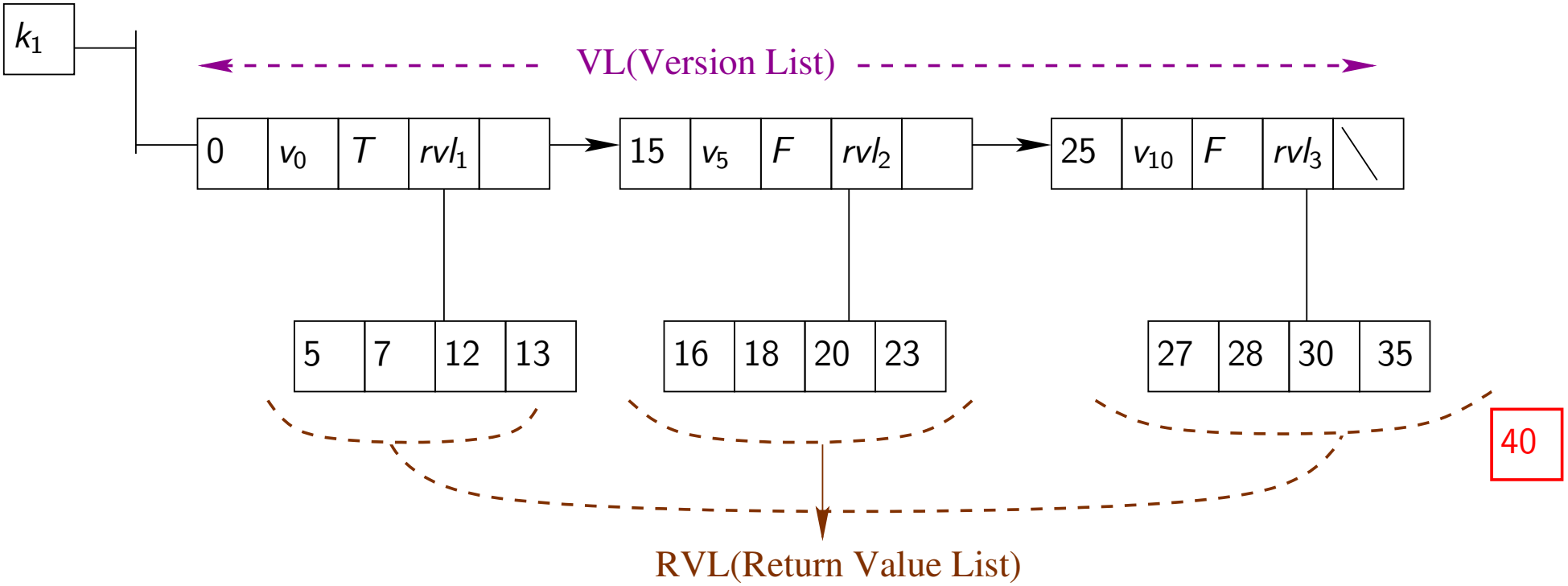


Figure 25: Insert a version of key k_1 by T_{40}

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

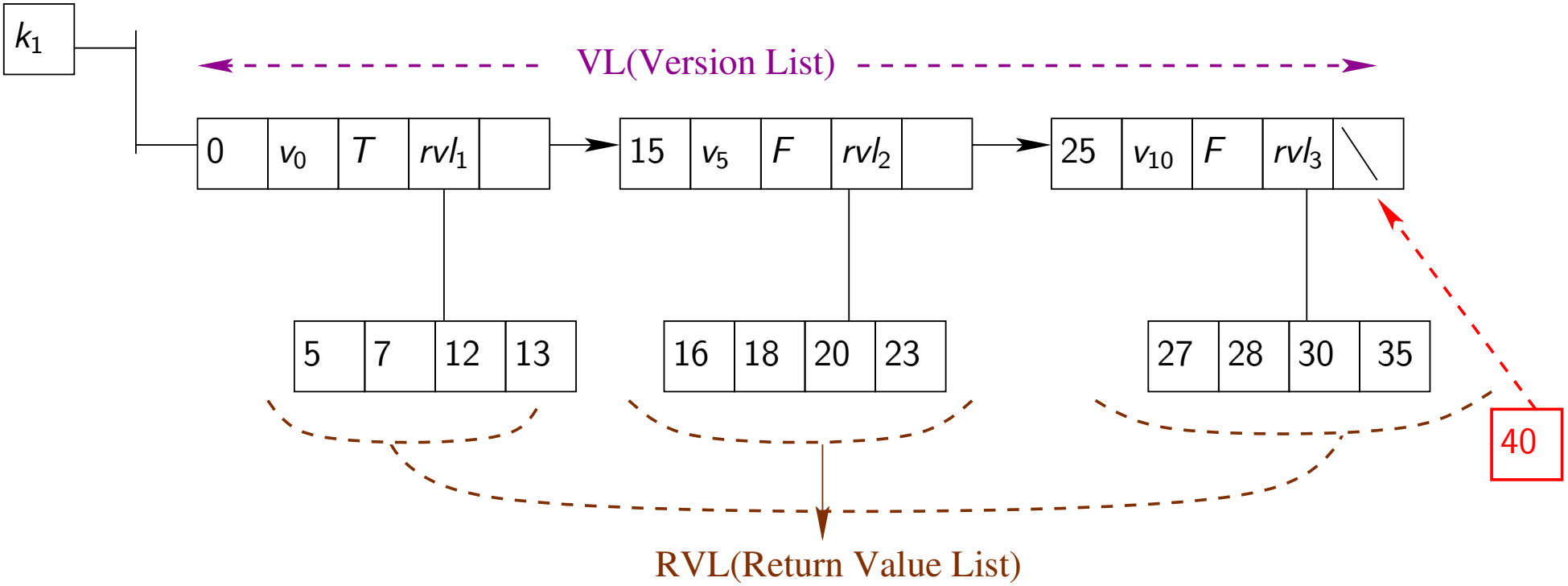


Figure 26: T_{40} searching appropriate place in version list of k_1

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

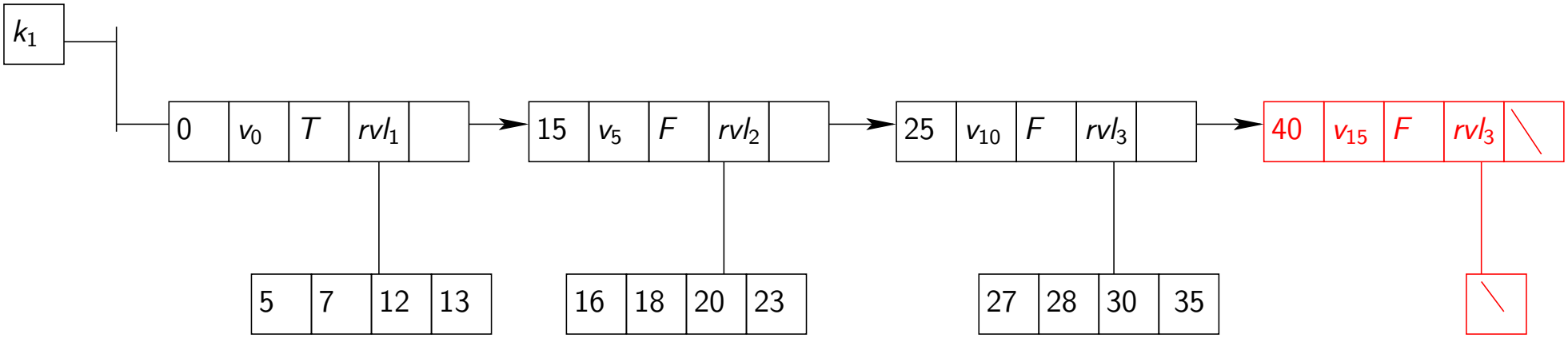


Figure 27: T_{40} successfully created a new version of k_1

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

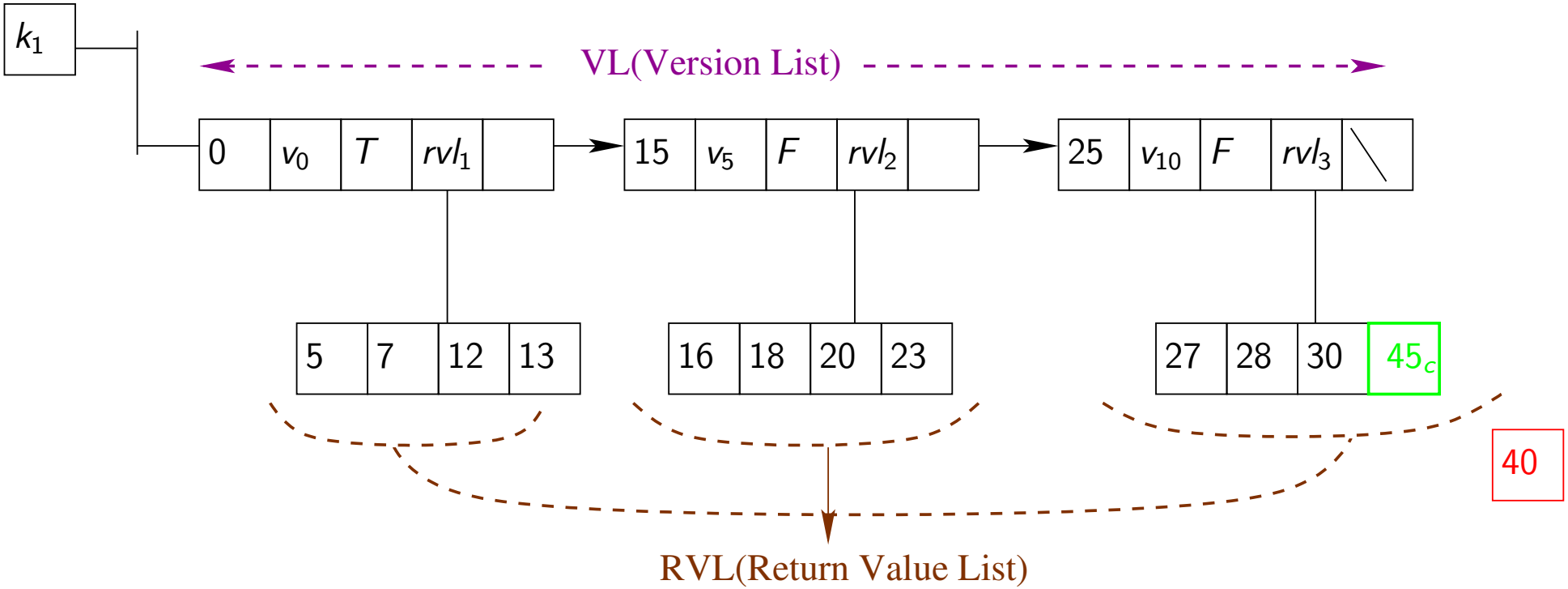


Figure 28: Insert a version of key k_1 by T_{40}

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

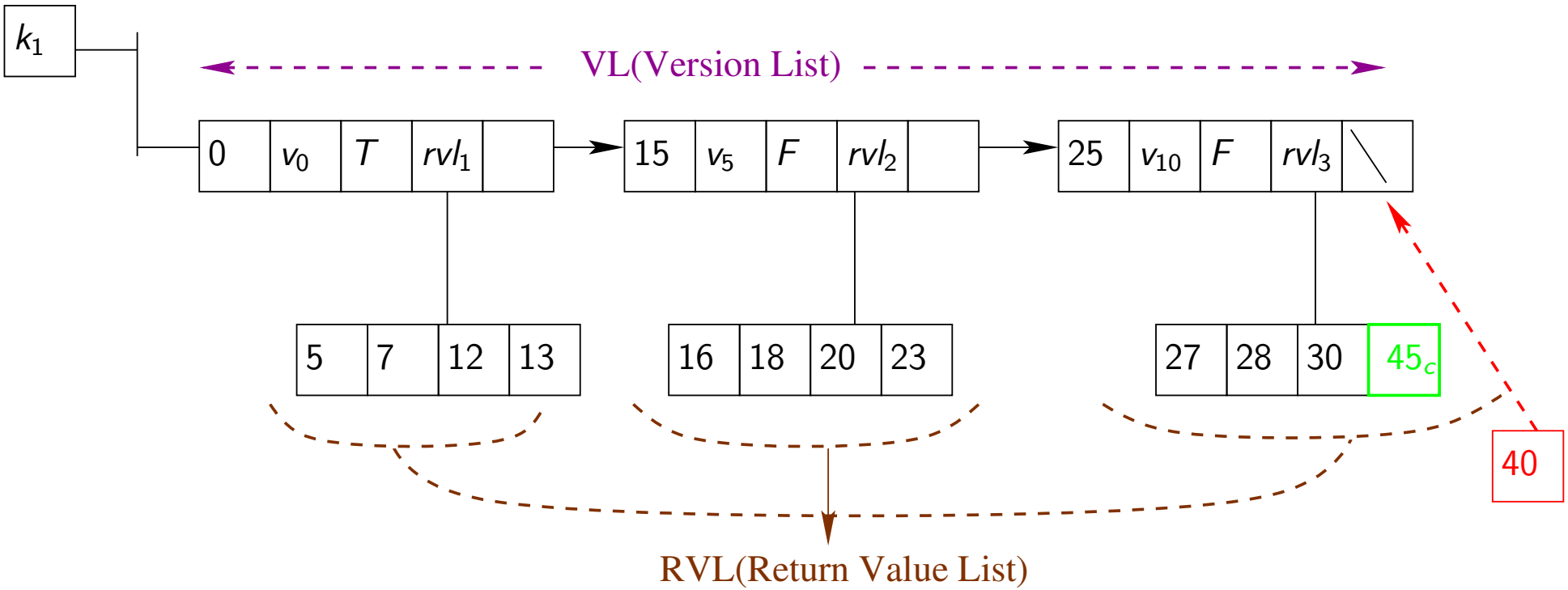


Figure 29: T_{40} searching appropriate place in version list of k_1

Analysing K-MVOSTM Cont'd..

tryC() : t_insert() method

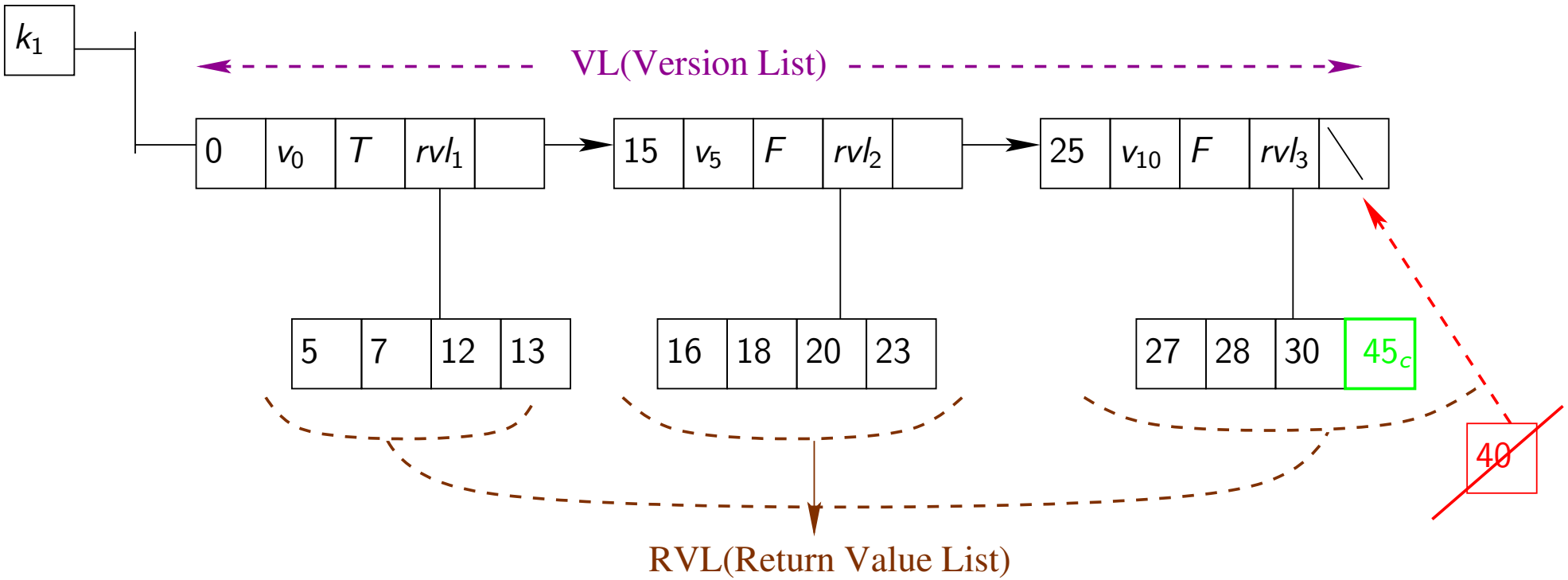


Figure 30: Abort T_{40} : T_{45} committed before T_{40}

Problem with K-MVOSTM

Drawback

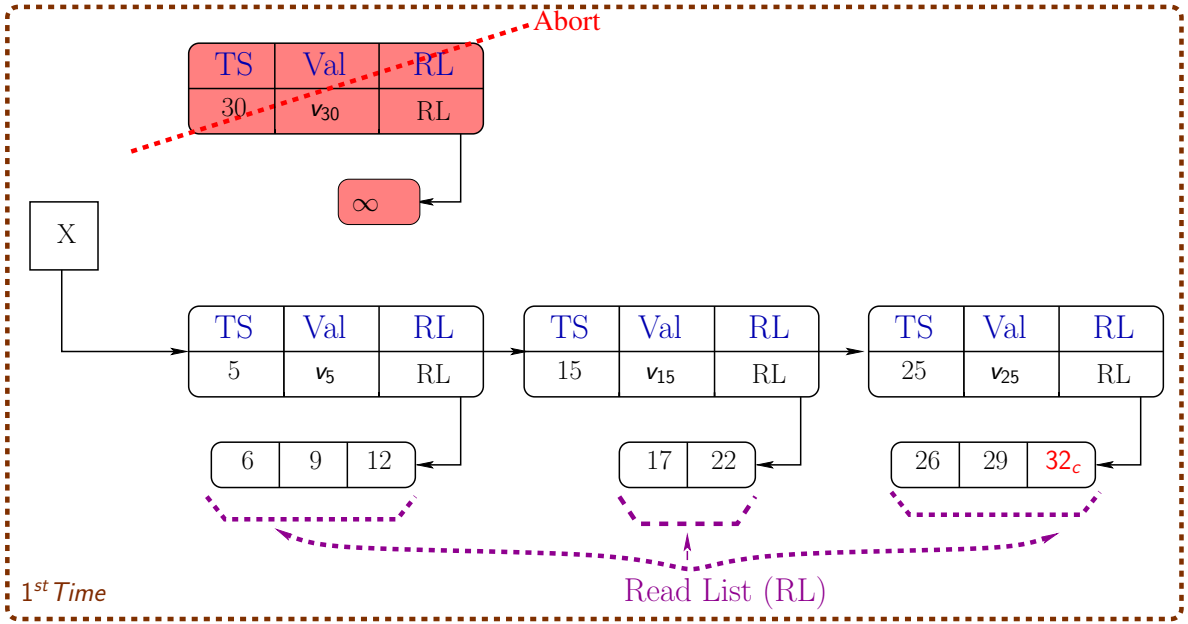


Figure 31: Starvation in K-MVOSTM

Problem with K-MVOSTM

Drawback

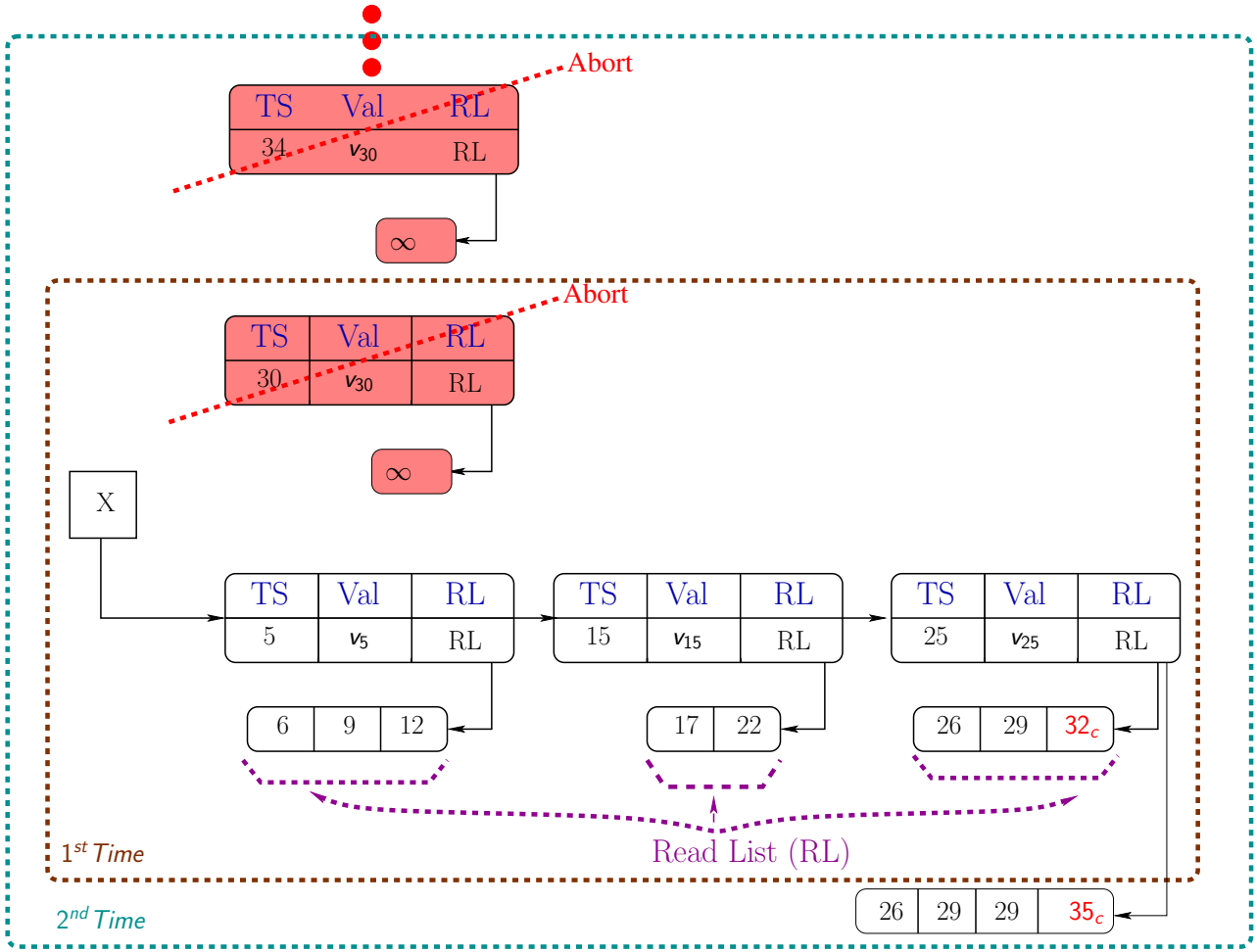


Figure 32: Starvation in K-MVOSTM

Adding ITS, CTS to K-MVOSTM

Solution Direction

Adding ITS, CTS to K-MVOSTM

Solution Direction

- We can add the notion of *its*, *cts* to K-MVOSTM algorithm.

Adding ITS, CTS to K-MVOSTM

Solution Direction

- We can add the notion of *its*, *cts* to K-MVOSTM algorithm.
- Key Observation: Any transaction with lowest *its* and highest *cts* will never abort. ▸ Detailed Description
- First observed in the context of RWSTMs by Chaudhary, V.P., Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S.: Achieving starvation freedom in multi-version transactional memory systems. NETYS 2019.

Key Insight for Achieving Starvation

Requirement of WTS

- A transaction T_i instead of using the current time as cts_i, uses a potentially a unique higher timestamp, *Working Timestamp* - *wts* or wts_i.

Key Insight for Achieving Starvation

Requirement of WTS

- A transaction T_i instead of using the current time as cts_i, uses a potentially a unique higher timestamp, *Working Timestamp* - *wts* or wts_i.
- Specifically, it adds $C * (cts_i - its_i)$ to cts_i, i.e.,

$$wts_i = cts_i + C * (cts_i - its_i);$$

where, C is any constant greater than 0.

Key Insight for Achieving Starvation

Requirement of WTS

- A transaction T_i instead of using the current time as cts_i, uses a potentially a unique higher timestamp, *Working Timestamp* - *wts* or wts_i.
- Specifically, it adds $C * (cts_i - its_i)$ to cts_i, i.e.,

$$wts_i = cts_i + C * (cts_i - its_i);$$

  where, C is any constant greater than 0.
- In other words, when the transaction T_i is issued for the first time, wts_i is same as $cts_i = its_i$.

Key Insight for Achieving Starvation

Requirement of WTS

- A transaction T_i instead of using the current time as cts_i, uses a potentially a unique higher timestamp, *Working Timestamp* - *wts* or wts_i.
- Specifically, it adds $C * (cts_i - its_i)$ to cts_i, i.e.,

$$wts_i = cts_i + C * (cts_i - its_i);$$

  where, C is any constant greater than 0.
- In other words, when the transaction T_i is issued for the first time, wts_i is same as $cts_i = its_i$.
- However, as transaction keeps getting aborted, the drift between cts_i and wts_i increases. The value of wts_i increases with each retry.

Key Insight for Achieving Starvation

Requirement of WTS

- A transaction T_i instead of using the current time as cts_i, uses a potentially a unique higher timestamp, *Working Timestamp* - *wts* or wts_i.
- Specifically, it adds $C * (cts_i - its_i)$ to cts_i, i.e.,

$$wts_i = cts_i + C * (cts_i - its_i);$$

  where, C is any constant greater than 0.
- In other words, when the transaction T_i is issued for the first time, wts_i is same as $cts_i = its_i$.
- However, as transaction keeps getting aborted, the drift between cts_i and wts_i increases. The value of wts_i increases with each retry.
- Eventually, a transaction with lowest *its* will finally have the highest *wts*.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Drawback with *wts*

- As *wts* may be significantly larger than *cts* so it may not correspond to the real-time order.

[j] Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM 26(4) (1979).

[k] Kuznetsov, P., Peri, S.: Non-interference and local correctness in transactional memory. Theor. Comput. Sci. 2017

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Drawback with *wts*

- As *wts* may be significantly larger than *cts* so it may not correspond to the real-time order.
- Hence, the history generated using *wts* ensures starvation-freedom but does not satisfies strict-serializability[j] and local opacity [k].

[j] Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM 26(4) (1979).

[k] Kuznetsov, P., Peri, S.: Non-interference and local correctness in transactional memory. Theor. Comput. Sci. 2017

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Drawback with *wts*

- As *wts* may be significantly larger than *cts* so it may not correspond to the real-time order.
- Hence, the history generated using *wts* ensures starvation-freedom but does not satisfies strict-serializability[j] and local opacity [k].

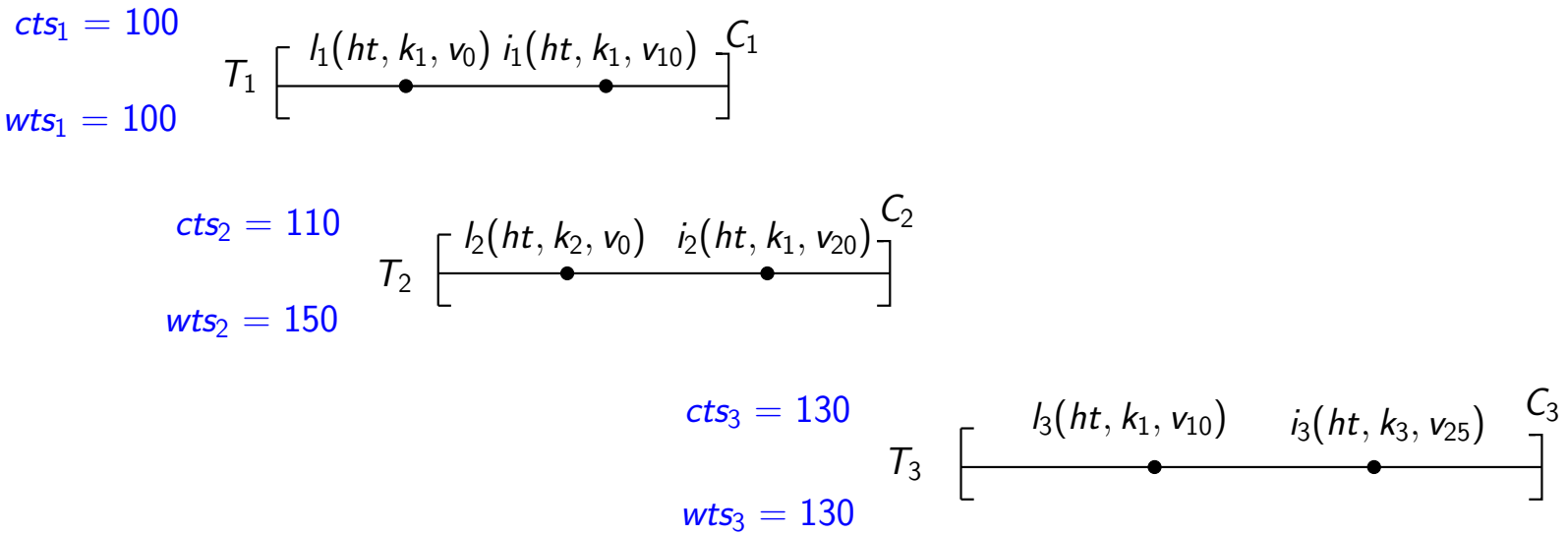


Figure 33: Violating the *real-time order* by *wts*

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Drawback with *wts*

- As *wts* may be significantly larger than *cts* so it may not correspond to the real-time order.
- Hence, the history generated using *wts* ensures starvation-freedom but does not satisfies strict-serializability[j] and local opacity [k].

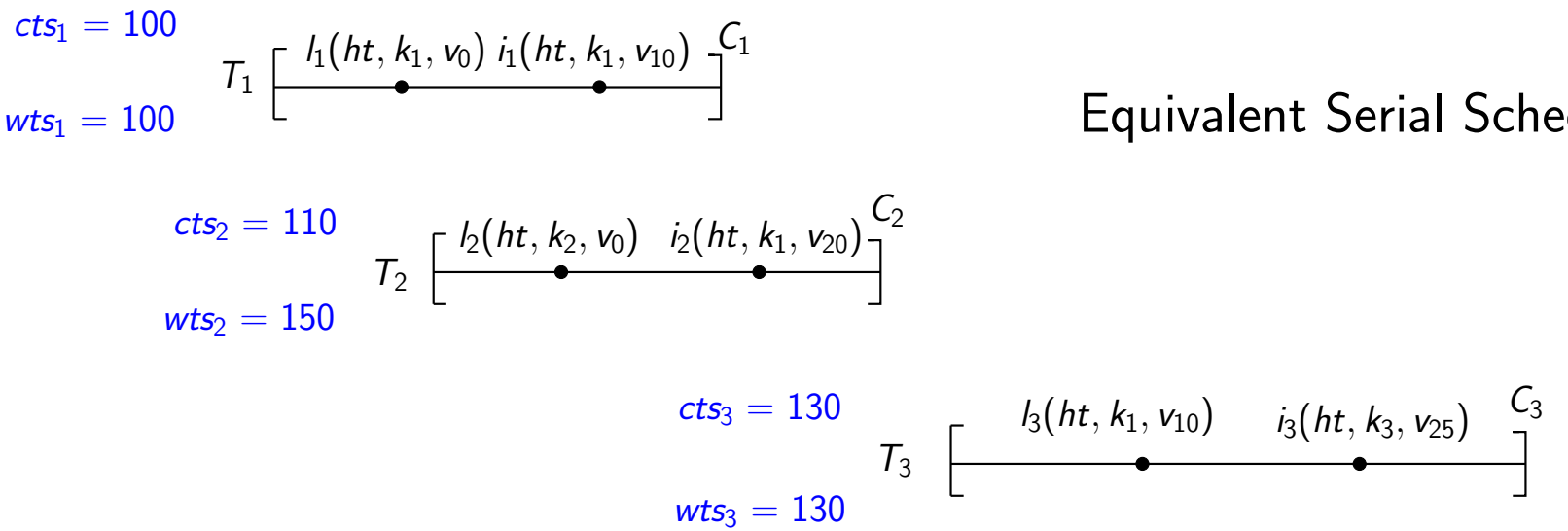
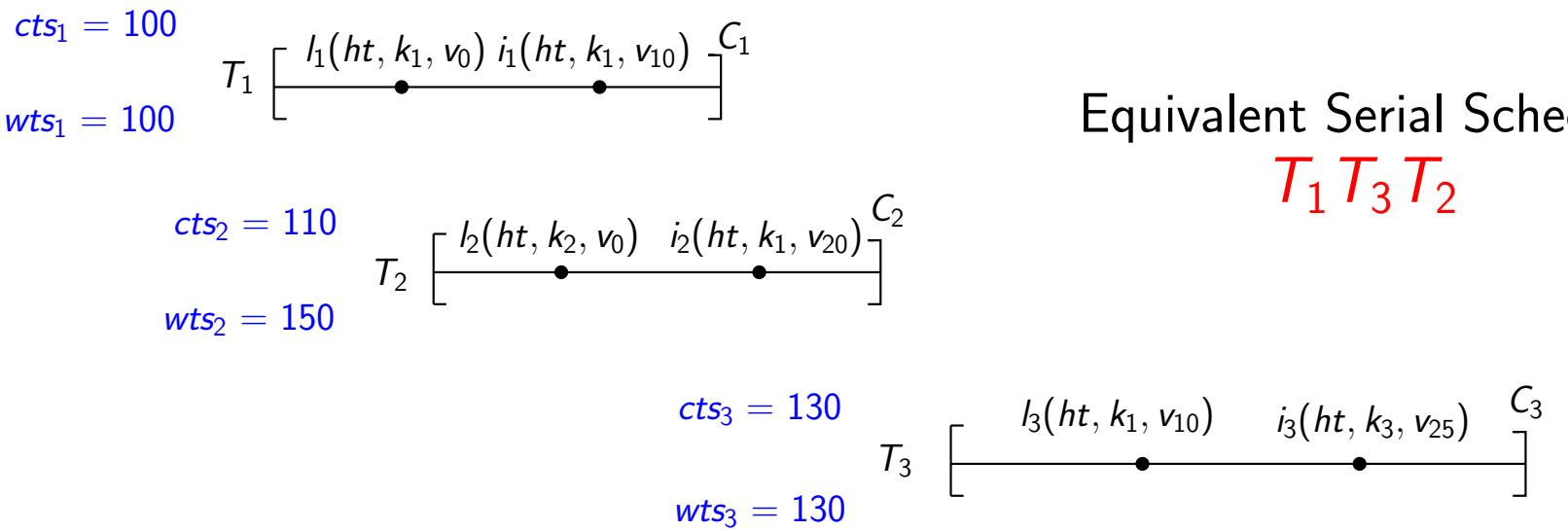


Figure 33: Violating the *real-time order* by *wts*

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Drawback with *wts*

- As *wts* may be significantly larger than *cts* so it may not correspond to the real-time order.
- Hence, the history generated using *wts* ensures starvation-freedom but does not satisfies strict-serializability[j] and local opacity [k].



Equivalent Serial Schedule is $T_1 T_3 T_2$

Figure 33: Violating the *real-time order* by *wts*

[j] Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM 26(4) (1979).

[k] Kuznetsov, P., Peri, S.: Non-interference and local correctness in transactional memory. Theor. Comput. Sci. 2017

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Regaining the real-time order

- We use the idea of timestamp ranges[1]: *Transaction Lower Timestamp Limit* or $tltl_i$, and *Transaction Upper Timestamp Limit* or $tutl_i$ to ensure real-time order.

[1] Riegel, T., Felber, P., Fetzer, C.: A lazy snapshot algorithm with eager validation. In: DISC 2006.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Regaining the real-time order

- We use the idea of timestamp ranges[1]: *Transaction Lower Timestamp Limit* or $tltl_i$, and *Transaction Upper Timestamp Limit* or $tutl_i$ to ensure real-time order.
- A transaction T_i invokes the methods as follows:
  - *t_begin()*:
    1. Initialize $\langle its, wts, tltl \rangle$ = cts
    2. tutl $= \infty$
  - *t_lookup()* or *t_delete()* or *tryC()*:
    - Increments the tltl
    - Decrements the tutl
    - **If**(tltl $>$ tutl) **then** T_i returns abort.

[1] Riegel, T., Felber, P., Fetzer, C.: A lazy snapshot algorithm with eager validation. In: DISC 2006.

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Correct Execution

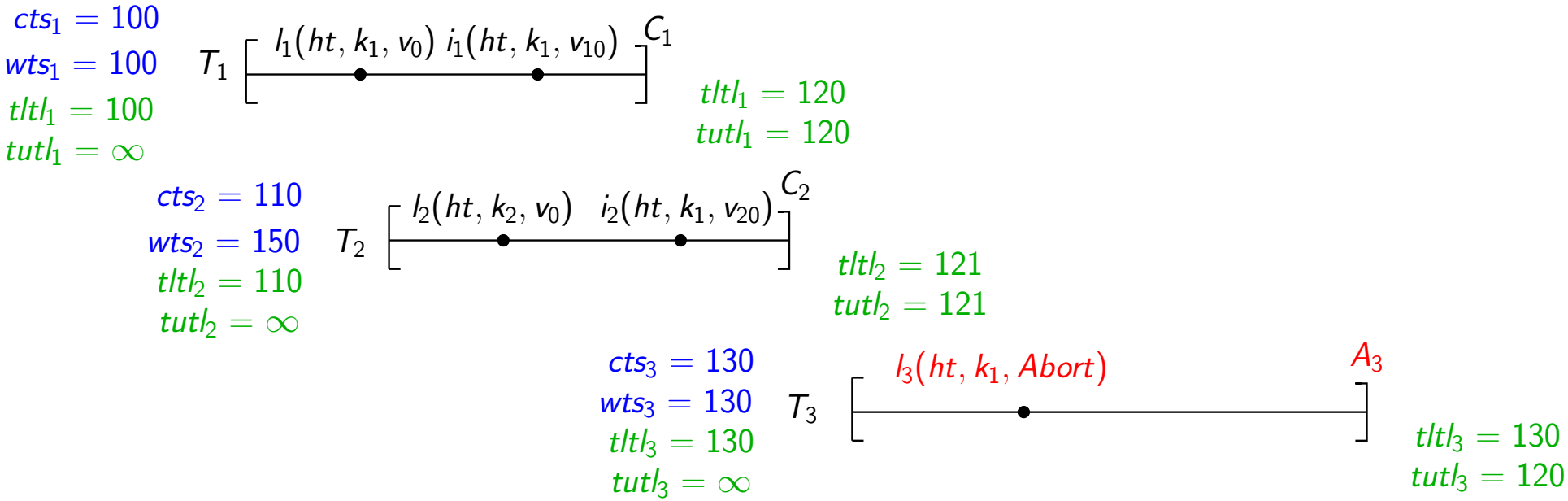


Figure 34: Regaining the *real-time order* using Timestamp Ranges along with *wts*

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Correct Execution

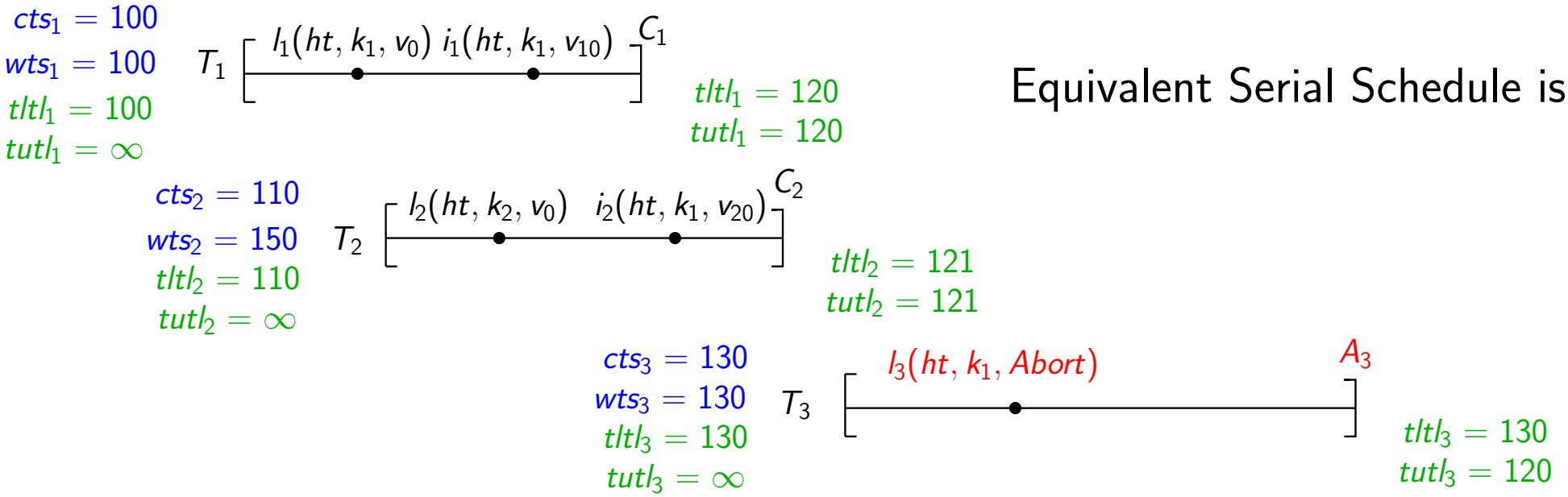


Figure 34: Regaining the *real-time order* using Timestamp Ranges along with *wts*

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Correct Execution

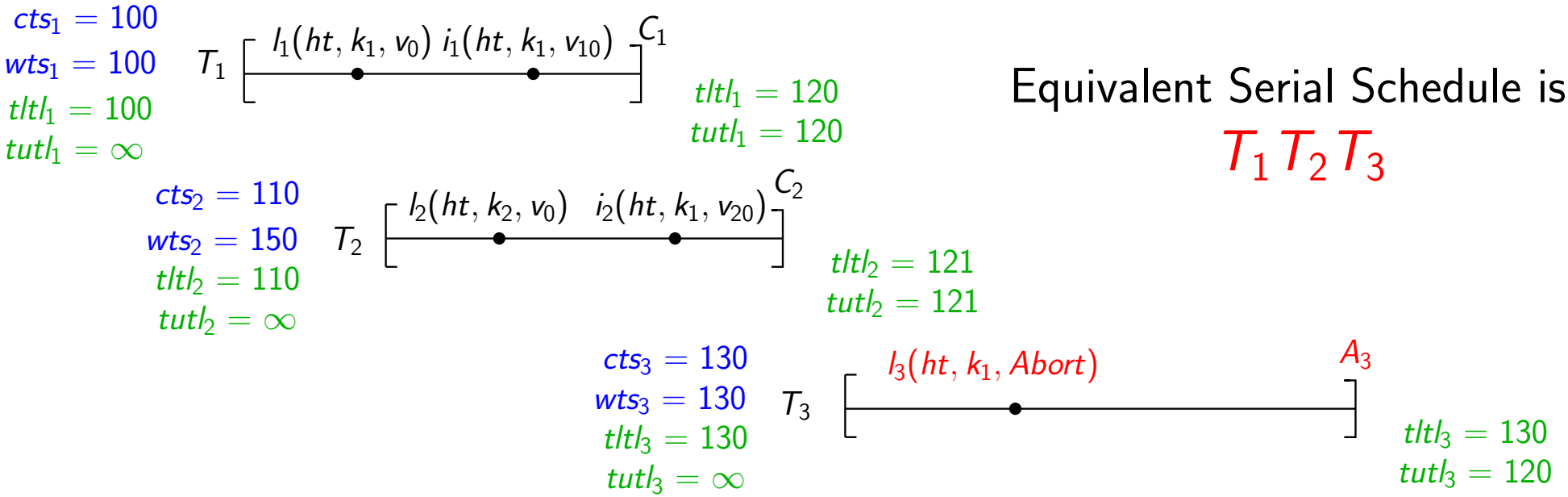


Figure 34: Regaining the *real-time order* using Timestamp Ranges along with *wts*

Outline

Proof Idea of SF-K-OSTMs

Safety and Liveness Proof

Theorem (1)

History generated by SF-K-OSTMs is strict-serializable and local-opaque.

Proof Idea of SF-K-OSTMs

Safety and Liveness Proof

Theorem (1)

History generated by SF-K-OSTMs is strict-serializable and local-opaque.

Theorem (2)

SF-K-OSTMs ensure starvation-freedom in presence of a fair scheduler that satisfies bounded-termination.

Proof Idea of SF-K-OSTMs

Safety and Liveness Proof

Theorem (1)

History generated by SF-K-OSTMs is strict-serializable and local-opaque.

Theorem (2)

SF-K-OSTMs ensure starvation-freedom in presence of a fair scheduler that satisfies bounded-termination.

Proof.

Eventually, every transaction T_i of SF-K-OSTMs will get the lowest *its* and highest *wts*. Hence, T_i will never abort and SF-K-OSTMs ensures starvation-freedom. □

Outline

Experimental Evaluations

Experimental Setup

- The experimental system is a 2-socket Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz with 14 cores per socket and 2 hyper-threads (HTs) per core, for a total of 56 threads.

Experimental Evaluations

Experimental Setup

- The experimental system is a 2-socket Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz with 14 cores per socket and 2 hyper-threads (HTs) per core, for a total of 56 threads.
- The machine has 32GB of RAM and runs Ubuntu 16.04.2 LTS.

Experimental Evaluations

Experimental Setup

- The experimental system is a 2-socket Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz with 14 cores per socket and 2 hyper-threads (HTs) per core, for a total of 56 threads.
- The machine has 32GB of RAM and runs Ubuntu 16.04.2 LTS.
- We consider three workloads as follows:
  1. (W1) Lookup Intensive: lookup:90%, insert:5% & delete:5%
  2. (W2) Mid Intensive: lookup:50%, insert:25% & delete:25%
  3. (W3) Update Intensive: lookup:10%, insert:45% & delete:45%

Experimental Evaluations

Experimental Setup

- The experimental system is a 2-socket Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz with 14 cores per socket and 2 hyper-threads (HTs) per core, for a total of 56 threads.
- The machine has 32GB of RAM and runs Ubuntu 16.04.2 LTS.
- We consider three workloads as follows:
  1. (W1) Lookup Intensive: lookup:90%, insert:5% & delete:5%
  2. (W2) Mid Intensive: lookup:50%, insert:25% & delete:25%
  3. (W3) Update Intensive: lookup:10%, insert:45% & delete:45%
- Shared data-items: 30 and Operations/transaction: 10

Experimental Evaluations Cont'd..

Comparison against State-of-the-art STMs

Hash Table based State-of-the-art STMs

- HT-SF-K-OSTM vs state-of-the-art STMs (HT-K-OSTM[a], HT-SV-OSTM[b], ESTM[c], RWSTM[d], and HT-MVTO[e]).

Experimental Evaluations Cont'd..

Comparison against State-of-the-art STMs

Hash Table based State-of-the-art STMs

- HT-SF-K-OSTM vs state-of-the-art STMs (HT-K-OSTM[a], HT-SV-OSTM[b], ESTM[c], RWSTM[d], and HT-MVTO[e]).

[a] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. In: SSS'18.

[b] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transac- tional Memory. NETYS '18.

[c] Felber, P., Gramoli, V., Guerraoui, R.: Elastic Transactions. J. Parallel Distrib. Comput. 2017.

[d] Weikum, G., Vossen, G.: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery 2002.

[e] Kumar, P., Peri, S., Vidyasankar, K.: A TimeStamp Based Multi-version STM Algorithm. In: ICDCN'14.

List based State-of-the-art STMs

- list-SF-K-OSTM vs state-of-the-art STMs (list-K-OSTM, list-SV-OSTM, Trans-list[a], Boosting-list[b], NOrec-list[c], list-MVTO, and list-K-SFTM[d]).

Experimental Evaluations Cont'd..

Comparison against State-of-the-art STMs

Hash Table based State-of-the-art STMs

- HT-SF-K-OSTM vs state-of-the-art STMs (HT-K-OSTM[a], HT-SV-OSTM[b], ESTM[c], RWSTM[d], and HT-MVTO[e]).

[a] Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. In: SSS'18.

[b] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transac- tional Memory. NETYS '18.

[c] Felber, P., Gramoli, V., Guerraoui, R.: Elastic Transactions. J. Parallel Distrib. Comput. 2017.

[d] Weikum, G., Vossen, G.: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery 2002.

[e] Kumar, P., Peri, S., Vidyasankar, K.: A TimeStamp Based Multi-version STM Algorithm. In: ICDCN'14.

List based State-of-the-art STMs

- list-SF-K-OSTM vs state-of-the-art STMs (list-K-OSTM, list-SV-OSTM, Trans-list[a], Boosting-list[b], NOrec-list[c], list-MVTO, and list-K-SFTM[d]).

[a] Zhang, D., Dechev, D.: Lock-free transactions without rollbacks for linked data structures. SPAA'16.

[b] Herlihy, M., Koskinen, E.: Transactional boosting: a methodology for highly-concurrent transactional objects. PPoPP, 2008

[c] Dalessandro, L., Spear, M.F., Scott, M.L.: NOrec: streamlining stm by abolishing ownership records. PPOPP, 2010.

[d] Chaudhary, V.P., Juyal, C., Kulkarni, S.S., Kumari, S., Peri, S.: Achieving starvation freedom in multi-version transactional memory systems. NETYS 2019

Experimental Evaluations Cont'd..

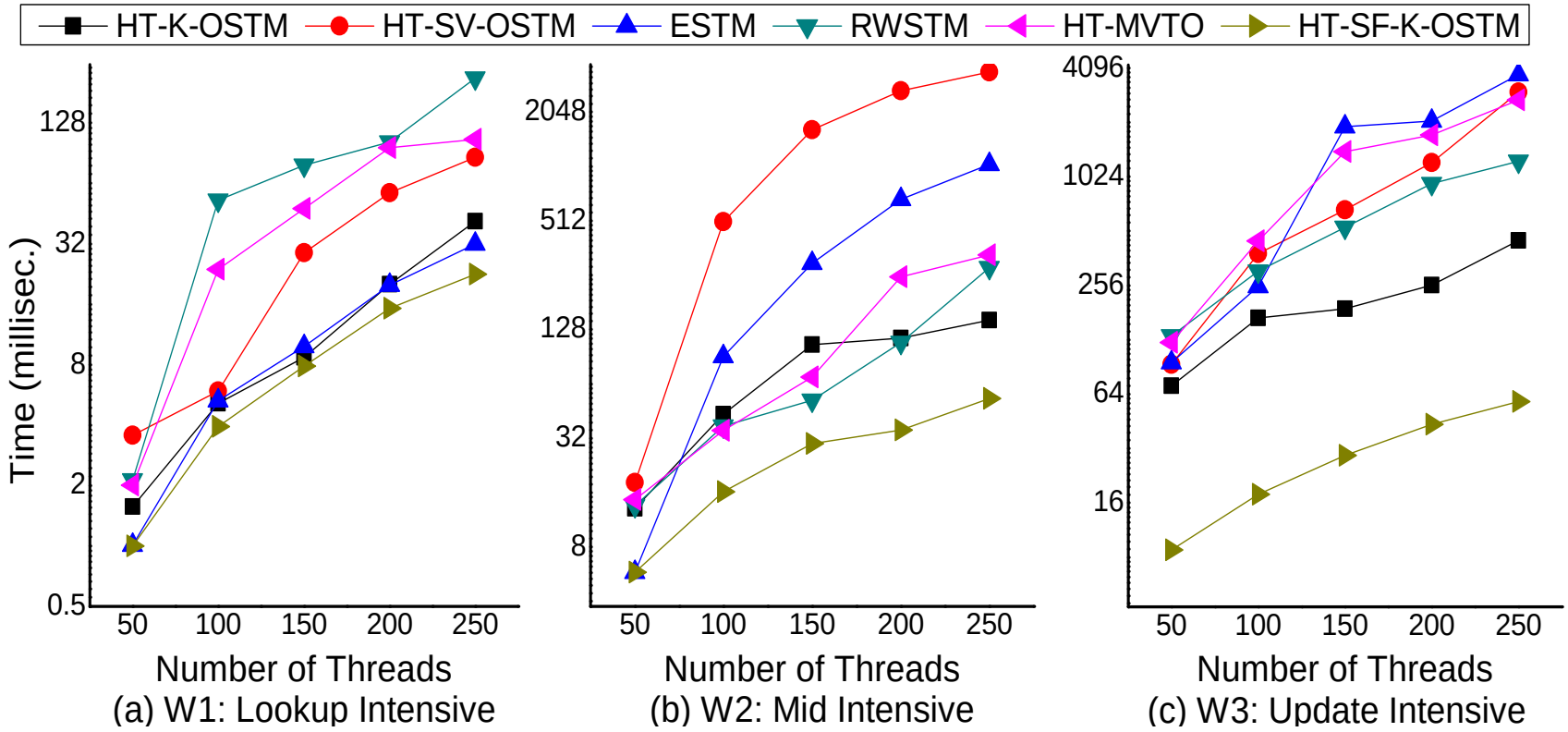


Figure 35: Performance analysis of SF-K-OSTM and State-of-the-art STMs on hash table

Experimental Evaluations Cont'd..

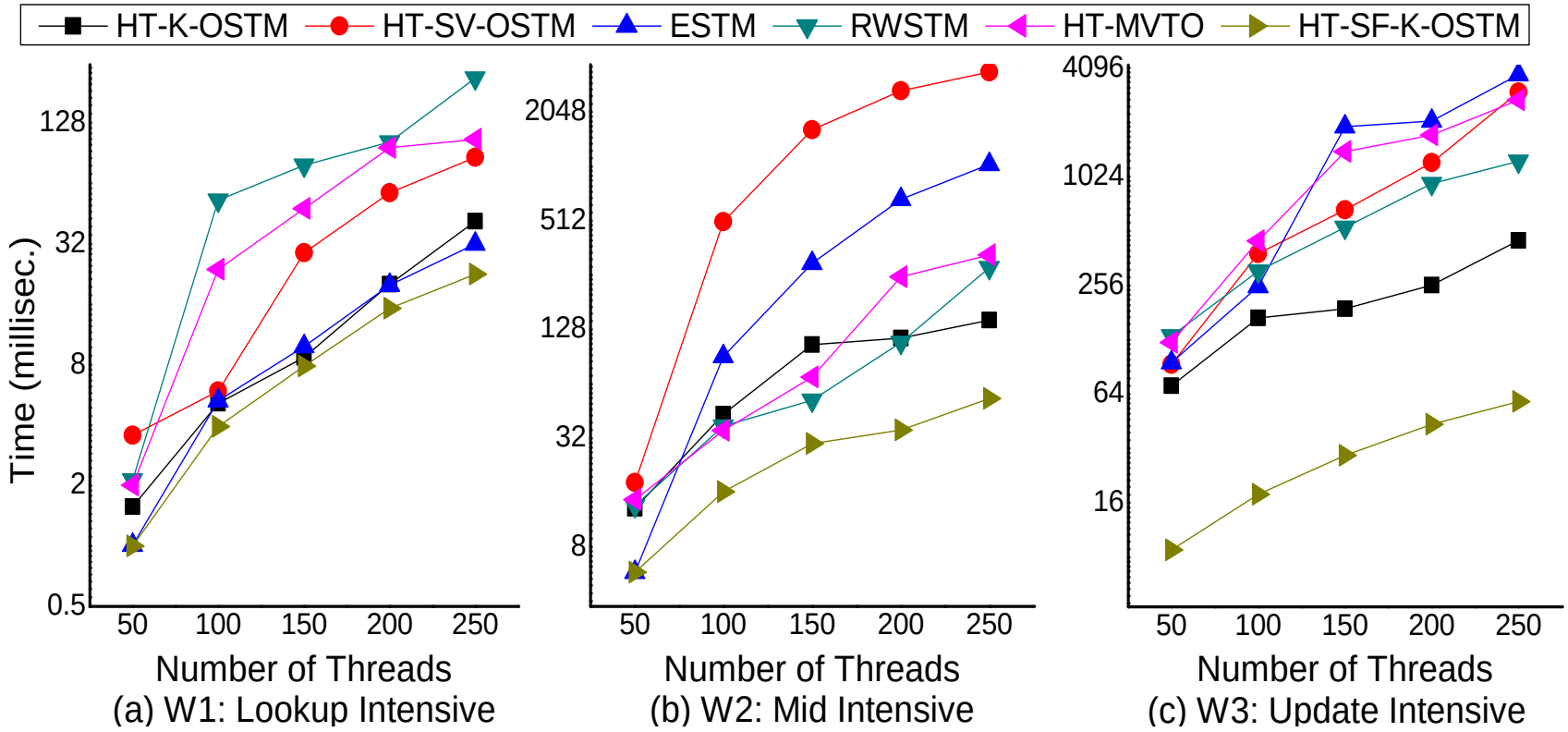


Figure 35: Performance analysis of SF-K-OSTM and State-of-the-art STMs on hash table

- Proposed hash table based SF-K-OSTM (HT-SF-K-OSTM) performs 3.9x, 32.18x, 22.67x, 10.8x, and 17.1x average speedup on *max-time* for a transaction to commit than state-of-the-art STMs (HT-K-OSTM, HT-SV-OSTM, ESTM, RWSTM, and HT-MVTO) respectively.

Experimental Evaluations Cont'd..

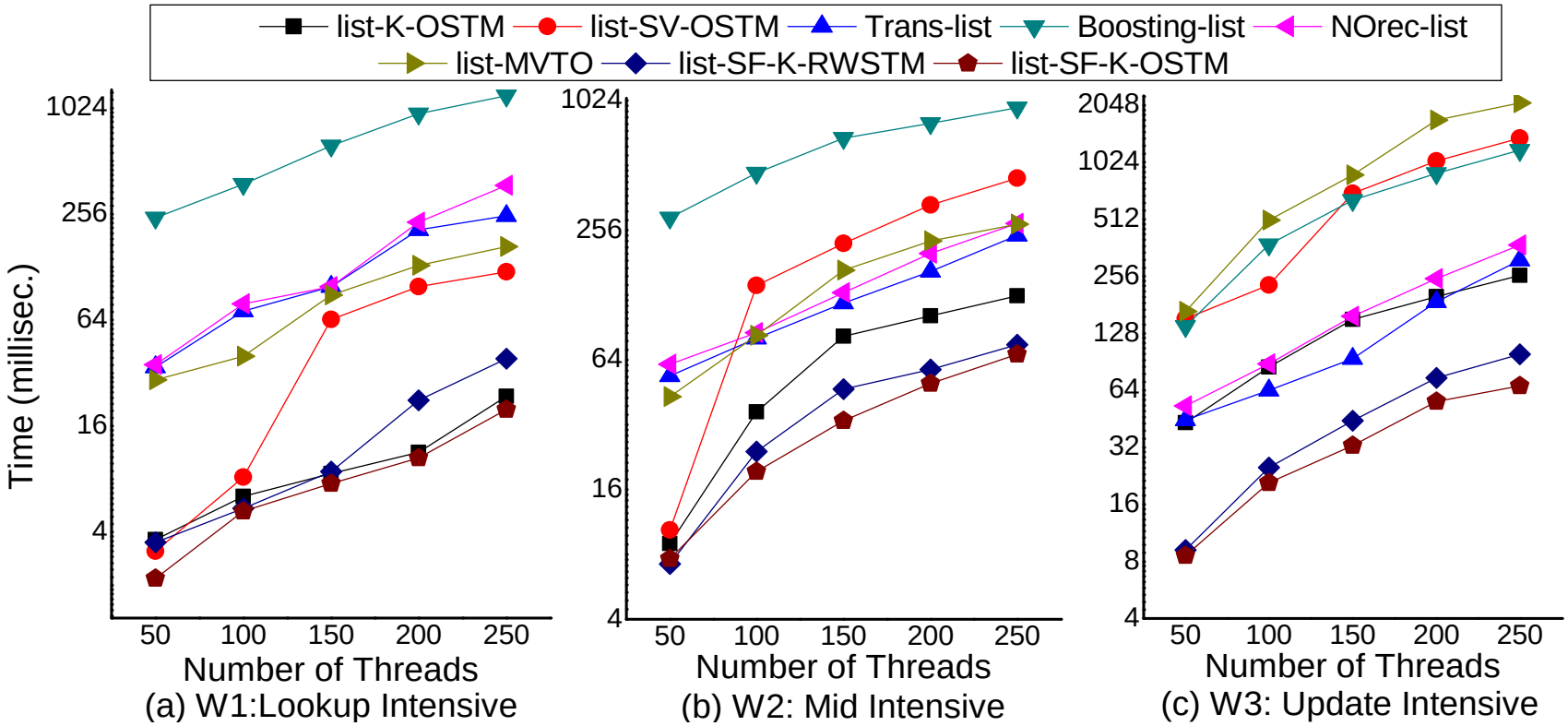


Figure 36: Performance analysis of SF-K-OSTM and State-of-the-art STMs on list

Experimental Evaluations Cont'd..

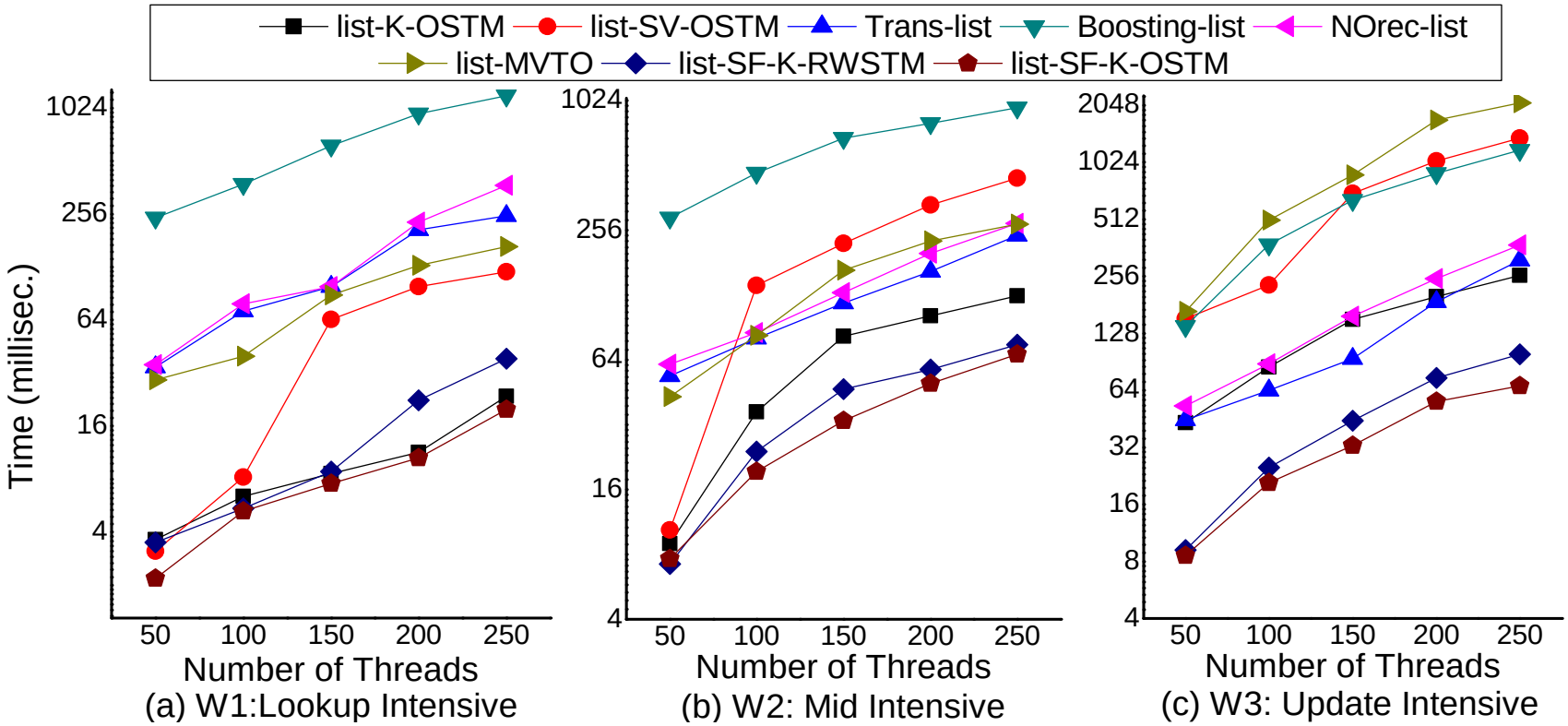


Figure 36: Performance analysis of SF-K-OSTM and State-of-the-art STMs on list

- Proposed list based SF-K-OSTM (list-SF-K-OSTM) performs 2.4x, 10.6x, 7.37x, 36.7x, 9.05x, 14.47x, and 1.43x average speedup on *max-time* for a transaction to commit than state-of-the-art STMs (list-K-OSTM, list-SV-OSTM, Trans-list, Boosting-list, NOrec-list, list-MVTO, and list-SF-K-RWSTM) respectively.

Experimental Evaluations Cont'd..

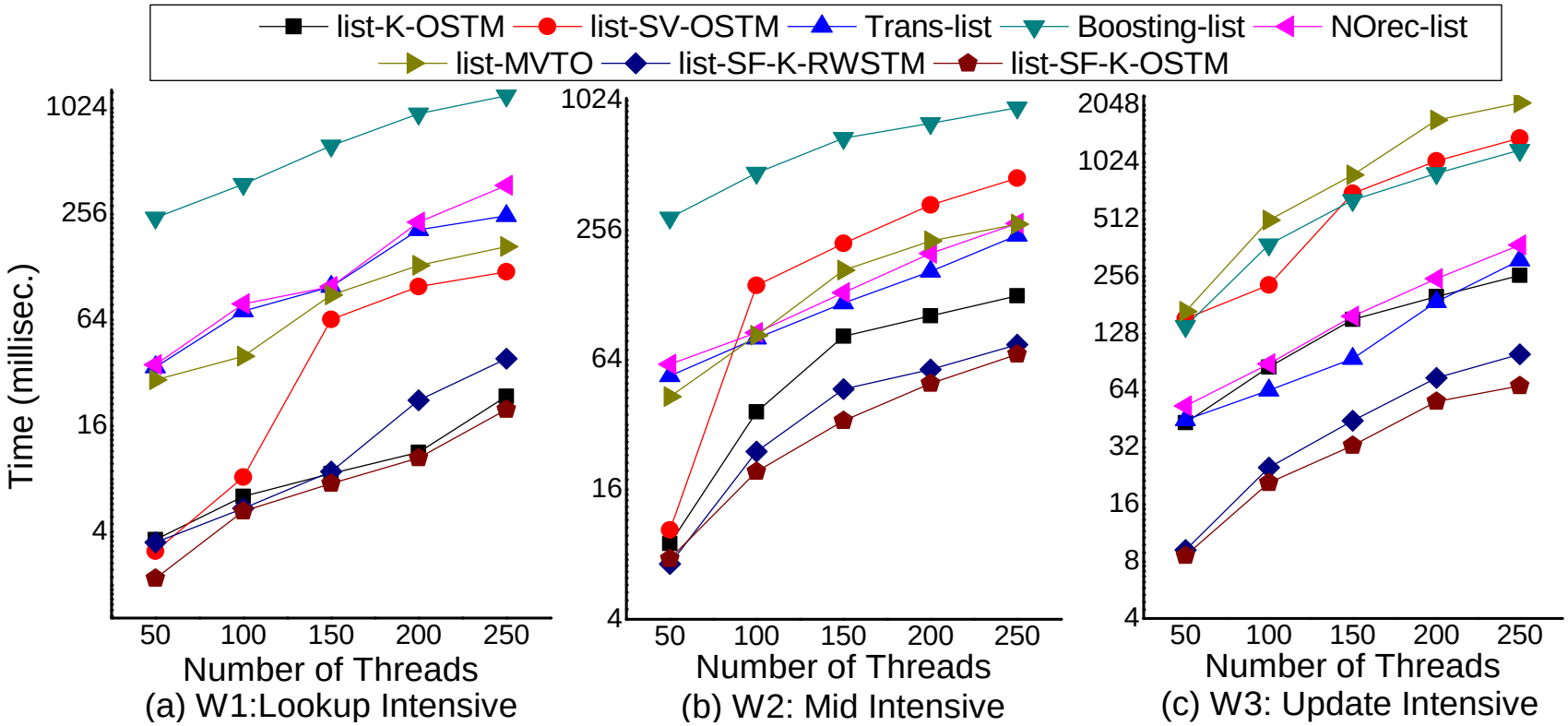


Figure 36: Performance analysis of SF-K-OSTM and State-of-the-art STMs on list

- Proposed list based SF-K-OSTM (list-SF-K-OSTM) performs 2.4x, 10.6x, 7.37x, 36.7x, 9.05x, 14.47x, and 1.43x average speedup on *max-time* for a transaction to commit than state-of-the-art STMs (list-K-OSTM, list-SV-OSTM, Trans-list, Boosting-list, NOrec-list, list-MVTO, and list-SF-K-RWSTM) respectively.

▸ Additional Experiments

Outline

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.
- For harnessing greater concurrency in STMs, we proposed a novel Starvation-Free *K-Version Object-based STM (SF-K-OSTM)*.

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.
- For harnessing greater concurrency in STMs, we proposed a novel Starvation-Free *K-Version Object-based STM (SF-K-OSTM)*.
- SF-K-OSTM ensures the starvation-freedom while maintaining the latest K-versions corresponding to each key and satisfies the correctness criteria as local-opacity.

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.
- For harnessing greater concurrency in STMs, we proposed a novel Starvation-Free *K-Version Object-based STM (SF-K-OSTM)*.
- SF-K-OSTM ensures the starvation-freedom while maintaining the latest K-versions corresponding to each key and satisfies the correctness criteria as local-opacity.
- We proposed the SF-SV-OSTM and SF-K-OSTM for hash-table and list but its generic for other data structures as well.

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.
- For harnessing greater concurrency in STMs, we proposed a novel Starvation-Free *K-Version Object-based STM (SF-K-OSTM)*.
- SF-K-OSTM ensures the starvation-freedom while maintaining the latest K-versions corresponding to each key and satisfies the correctness criteria as local-opacity.
- We proposed the SF-SV-OSTM and SF-K-OSTM for hash-table and list but its generic for other data structures as well.
- The results of SF-K-OSTM shows significant performance gain over state-of-the-art STMs and SF-SV-OSTM.

Conclusion and Future Direction

Conclusion

- We proposed *SF-SV-OSTM* which ensures starvation-freedom along with the correctness-criteria as conflict-opacity.
- For harnessing greater concurrency in STMs, we proposed a novel Starvation-Free *K-Version Object-based STM (SF-K-OSTM)*.
- SF-K-OSTM ensures the starvation-freedom while maintaining the latest K-versions corresponding to each key and satisfies the correctness criteria as local-opacity.
- We proposed the SF-SV-OSTM and SF-K-OSTM for hash-table and list but its generic for other data structures as well.
- The results of SF-K-OSTM shows significant performance gain over state-of-the-art STMs and SF-SV-OSTM.

Future Direction

- Exploring starvation-freedom in **Nested STMs**[a].

[a] Sathya Peri and K.Vidyasankar, Correctness of Concurrent Executions of Closed Nested Transactions in Transactional Memory Systems", Theoretical Computer Science, 2013

Thank You
&
Questions Please!!

Working of STMs

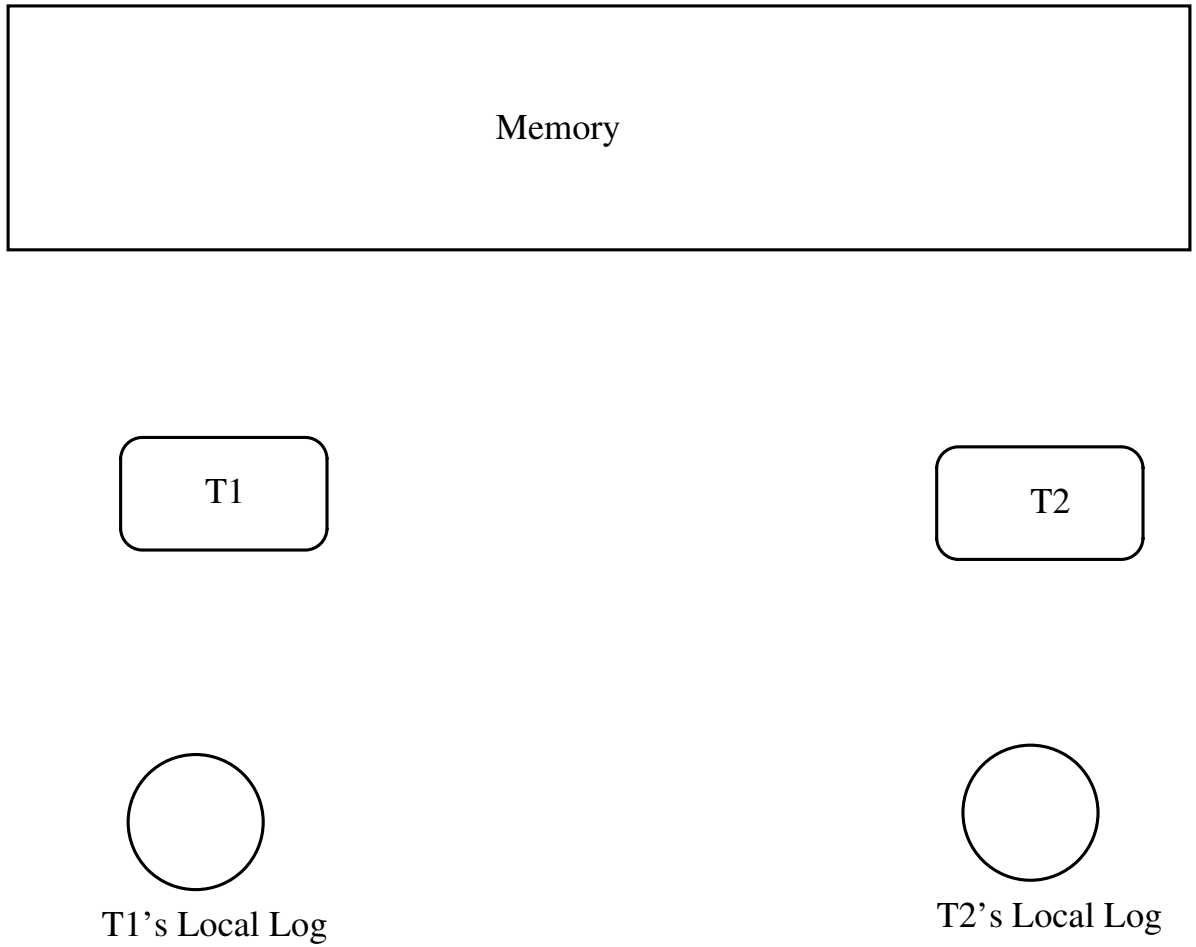


Figure 37: Working of STM System

return

Working of STMs

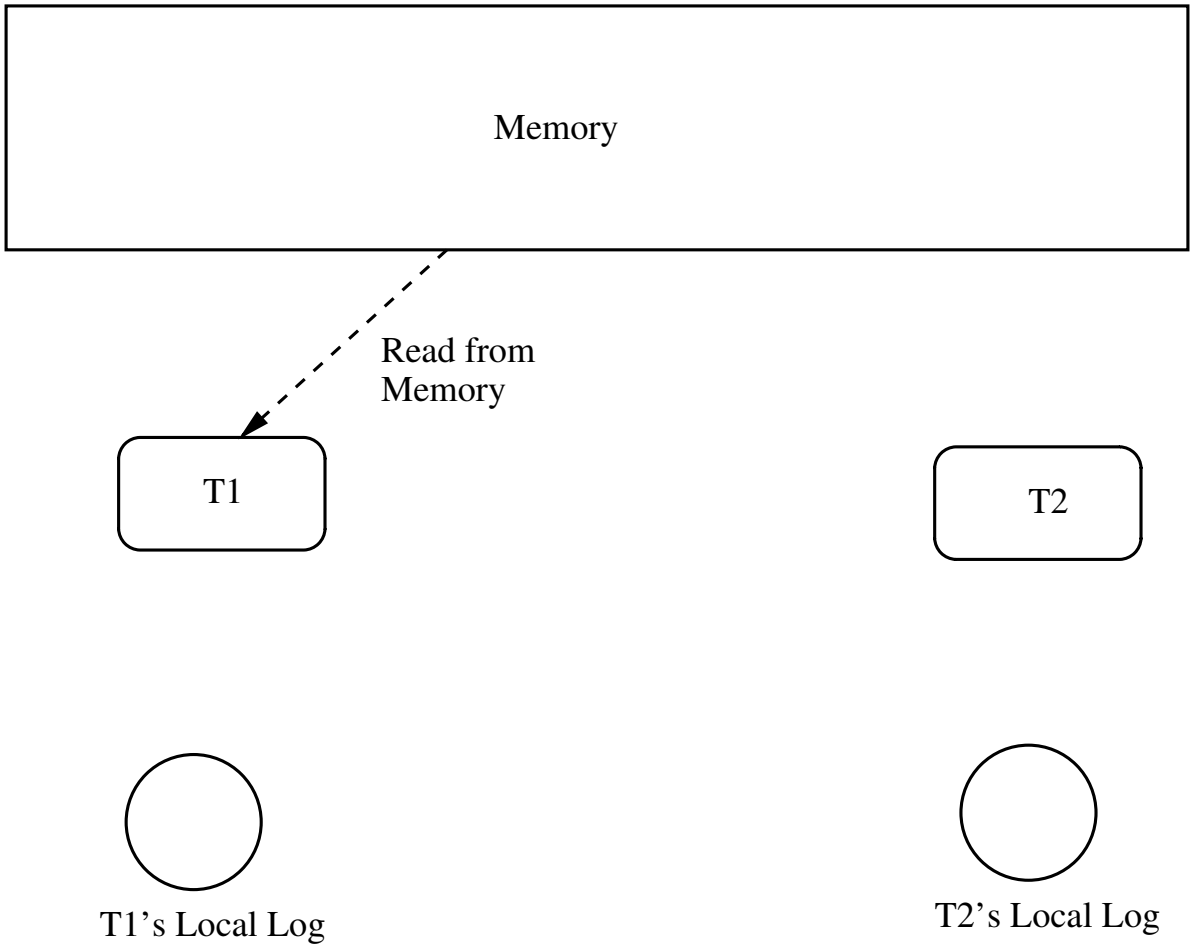


Figure 37: Working of STM System

return

Working of STMs

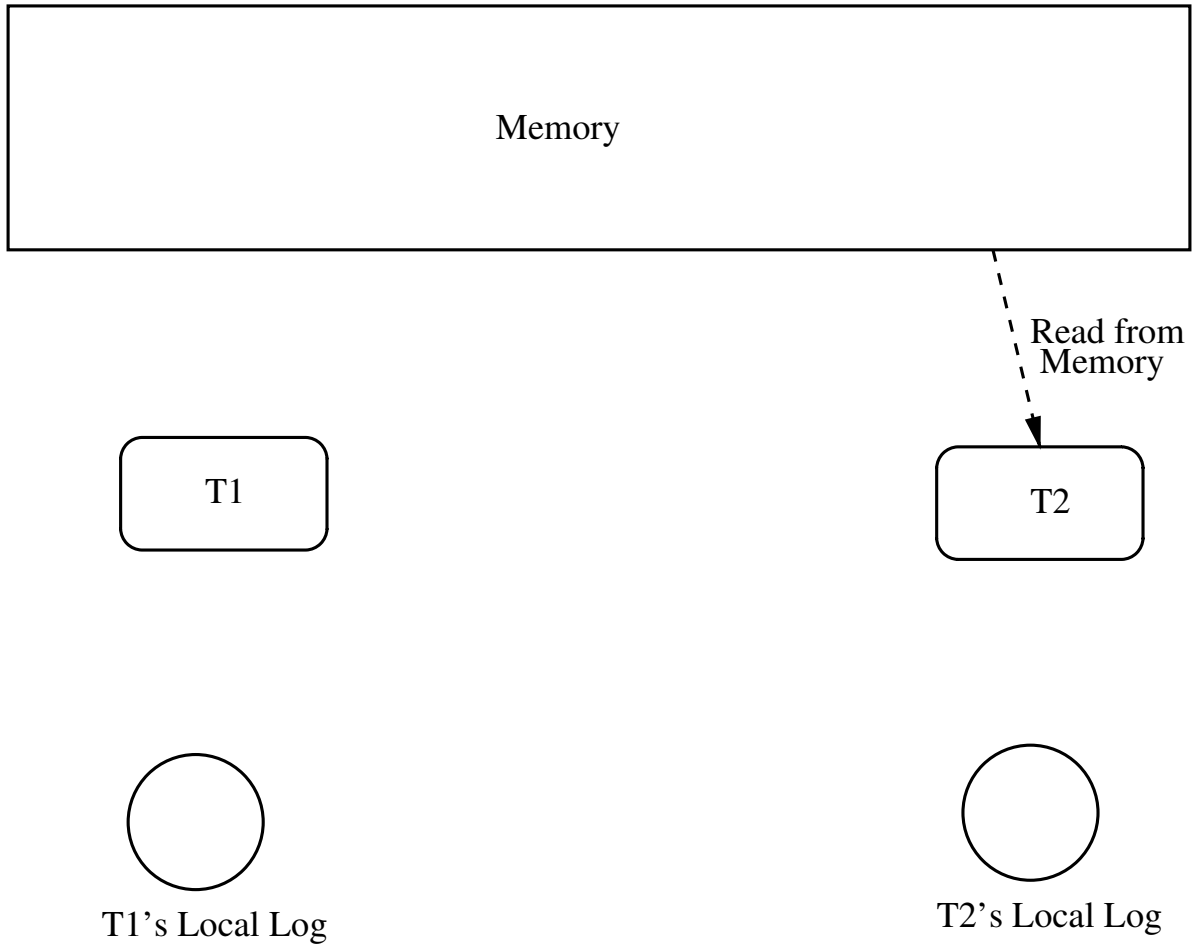


Figure 37: Working of STM System

return

Working of STMs

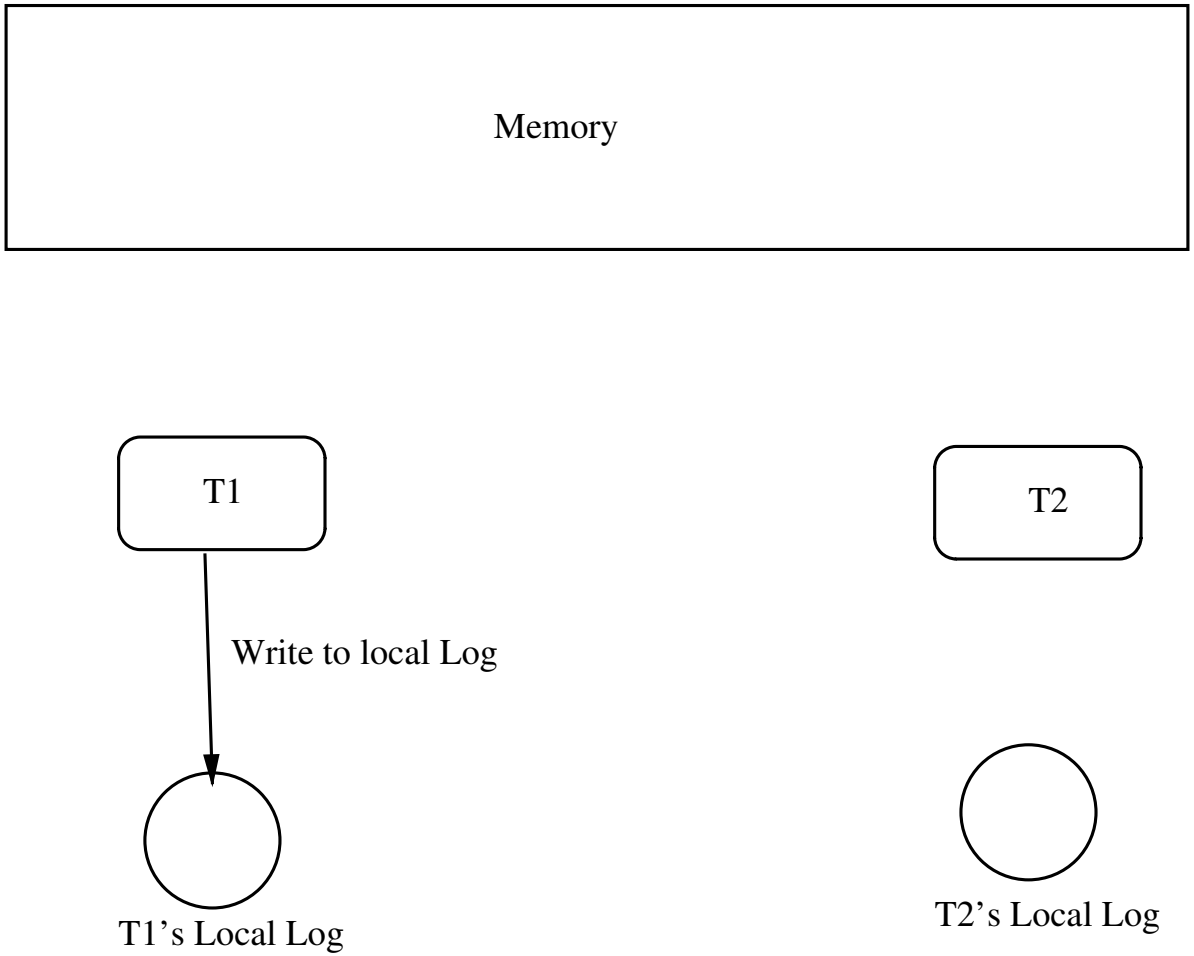


Figure 37: Working of STM System

return

Working of STMs

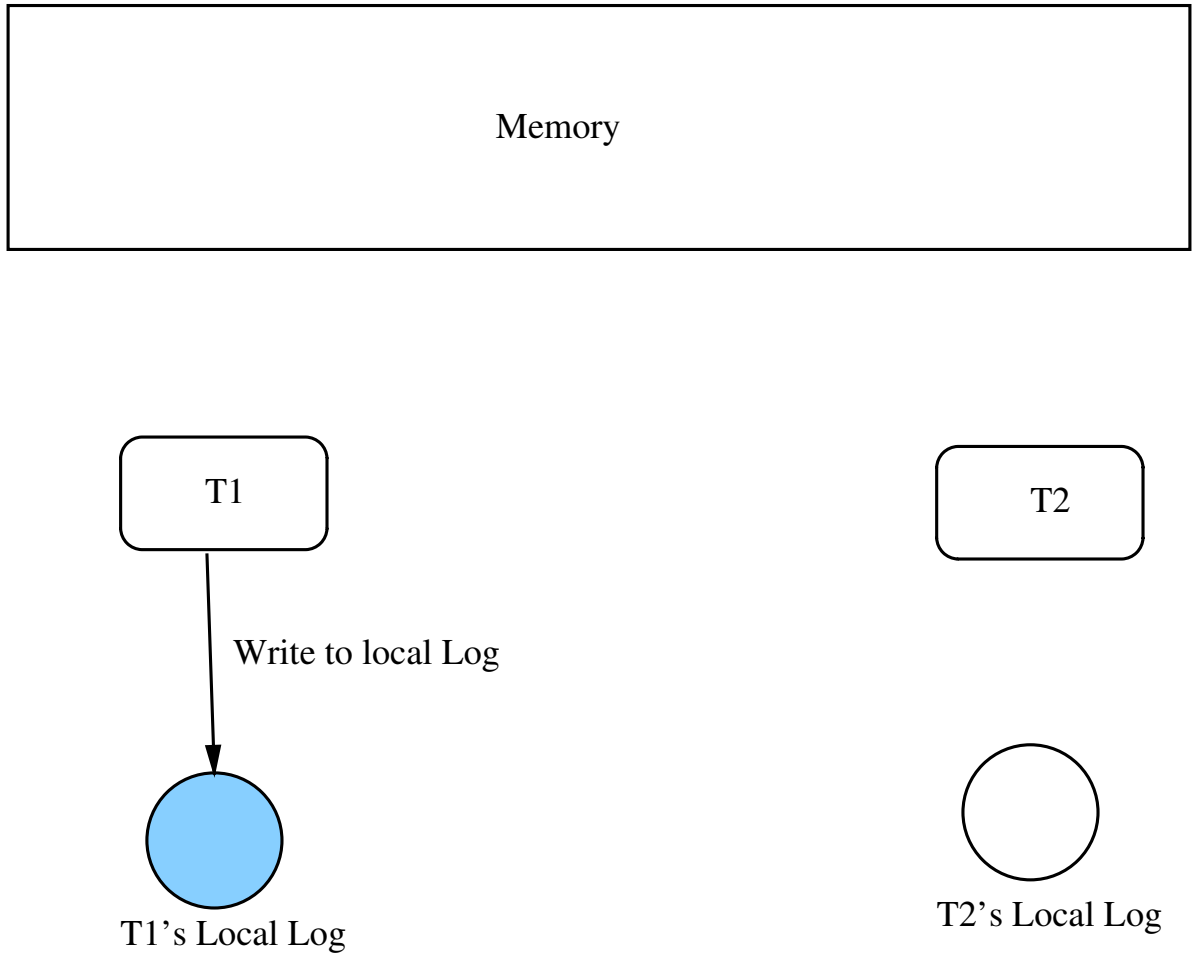


Figure 37: Working of STM System

return

Working of STMs

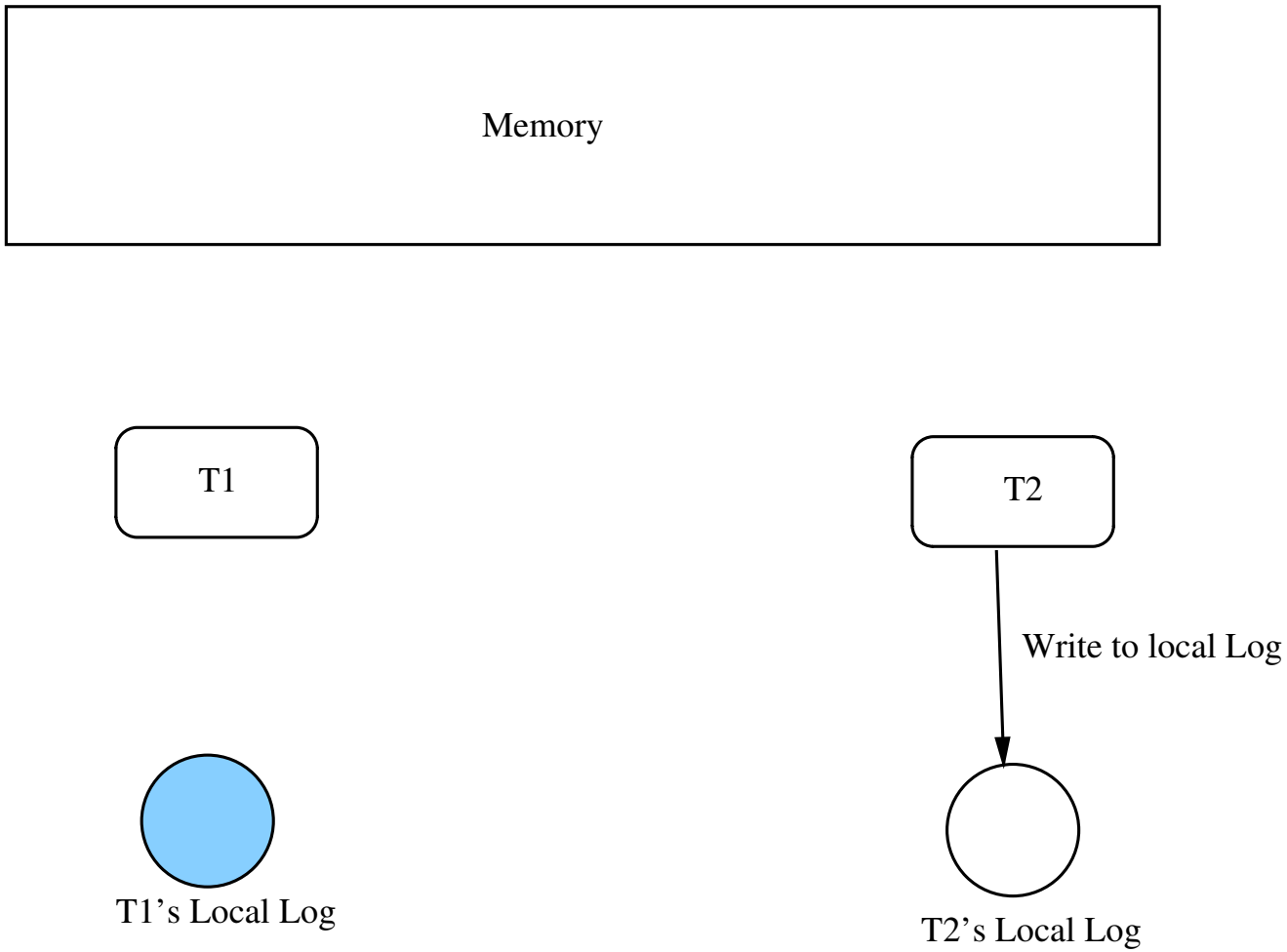


Figure 37: Working of STM System

return

Working of STMs

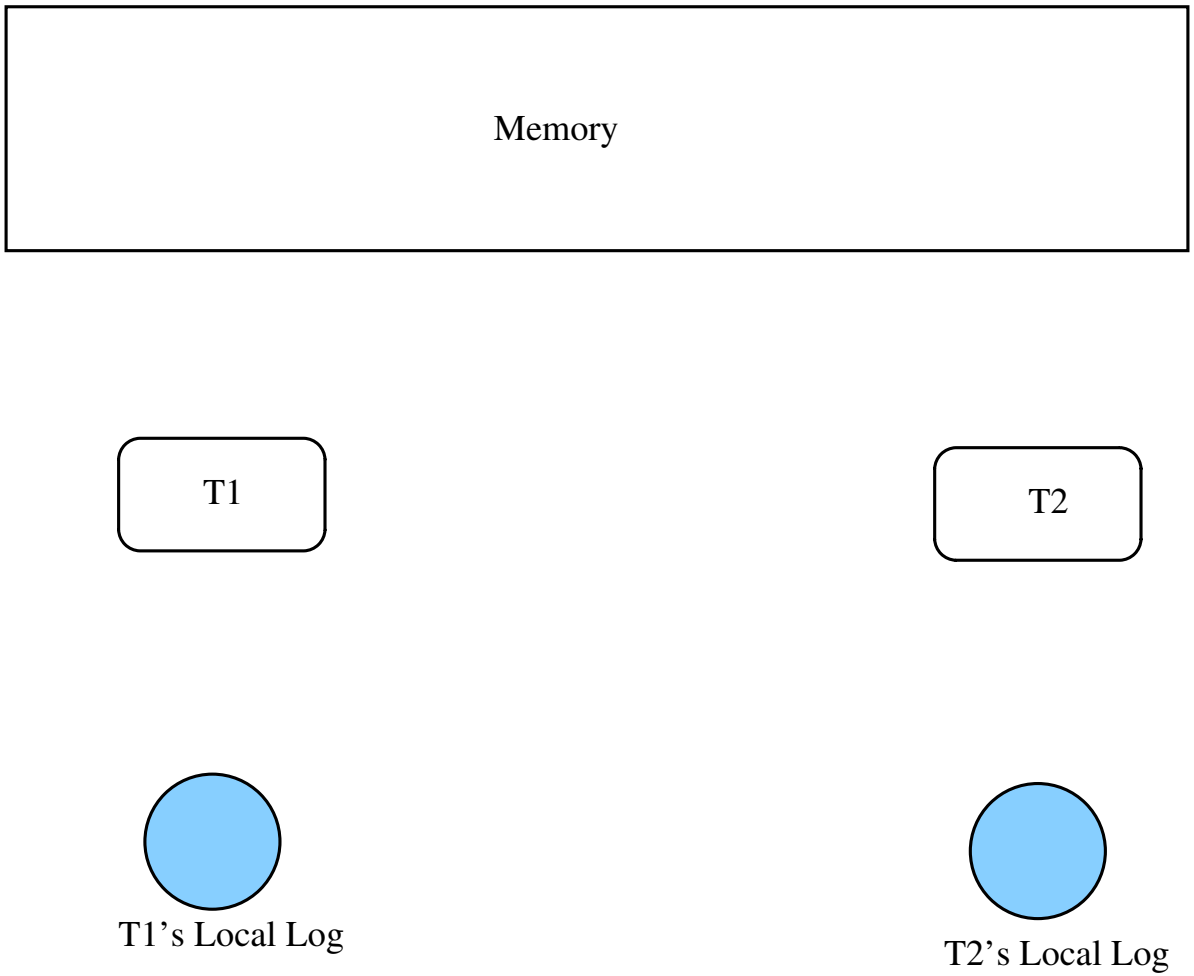


Figure 37: Working of STM System

return

Working of STMs

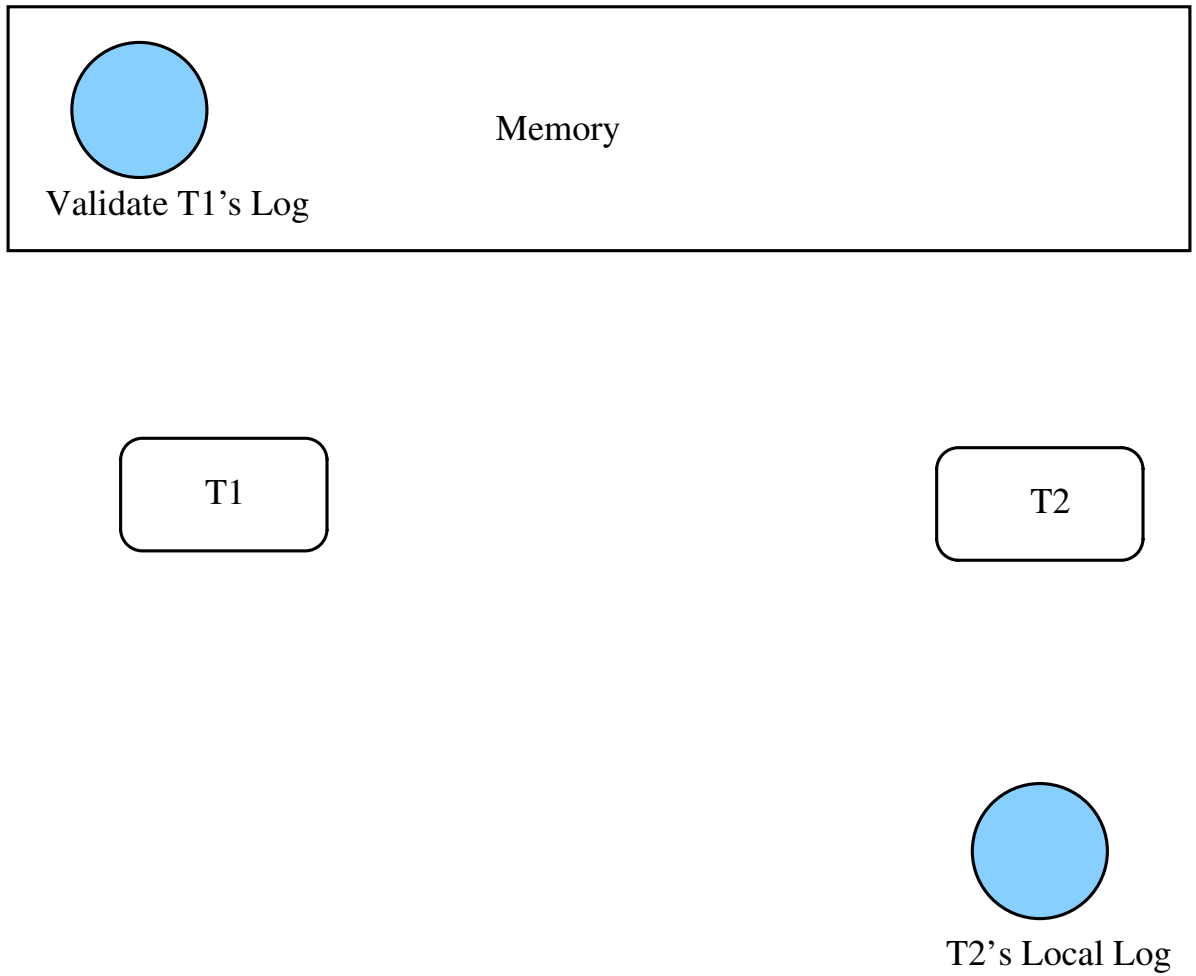


Figure 37: Working of STM System

return

Working of STMs

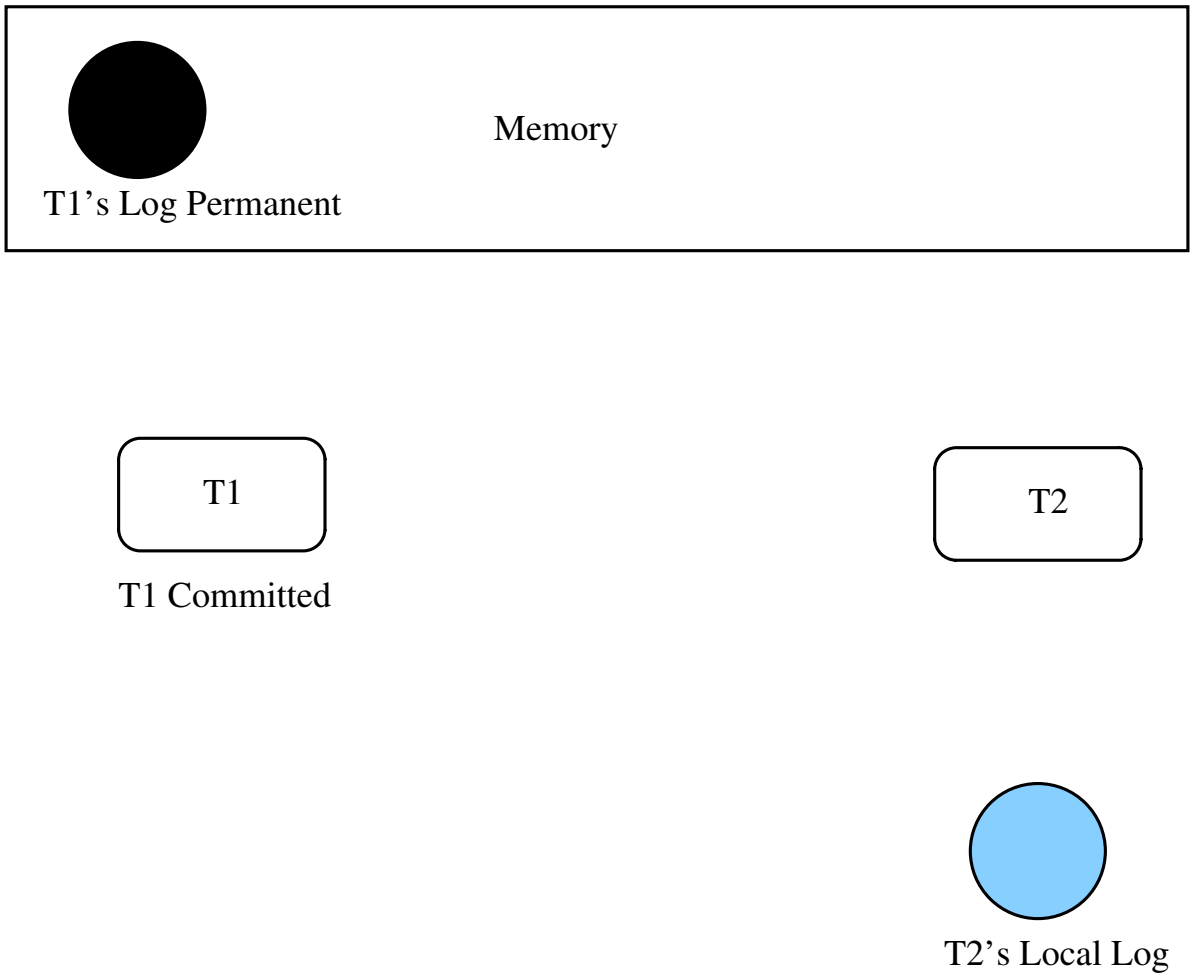


Figure 37: Working of STM System

return

Working of STMs

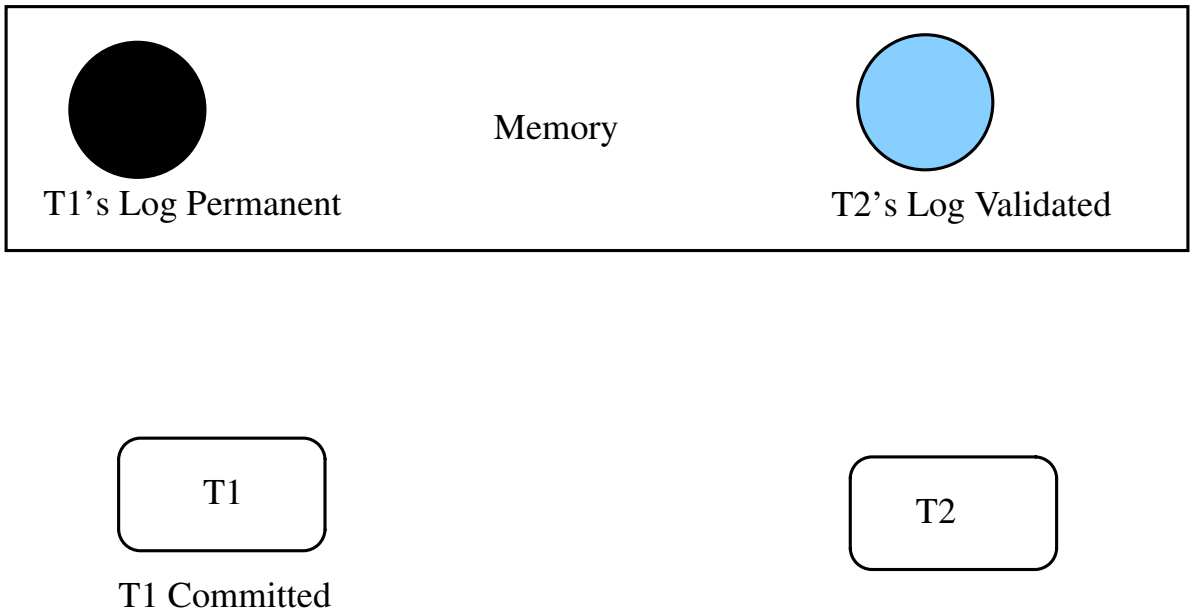


Figure 37: Working of STM System

return

Working of STMs

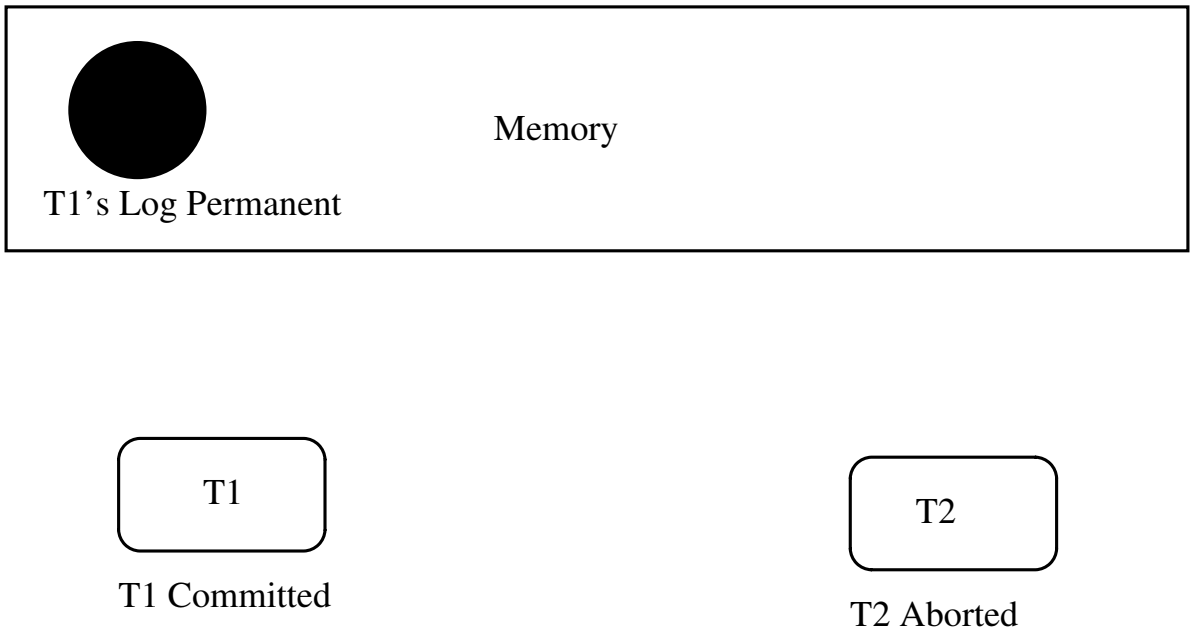


Figure 37: Working of STM System

return

Starvation-Freedom in Multi-Version OSTMs

Key Observation

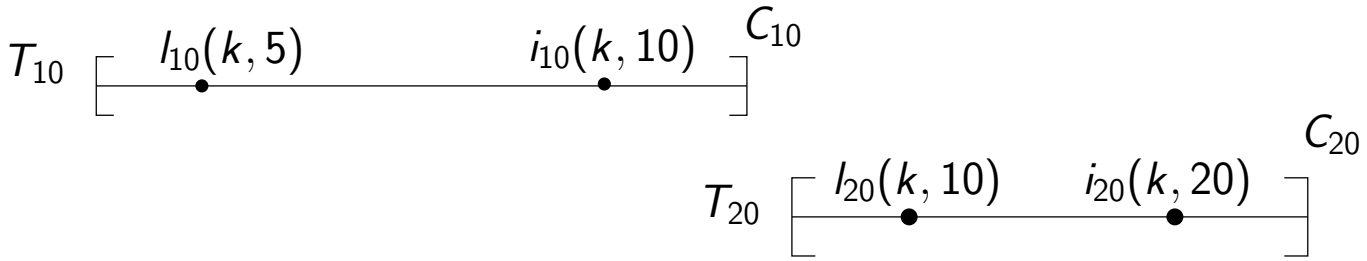


Figure 38: T_{20} looks up the version inserted by T_{10}. No conflict.

Starvation-Freedom in Multi-Version OSTMs

Key Observation

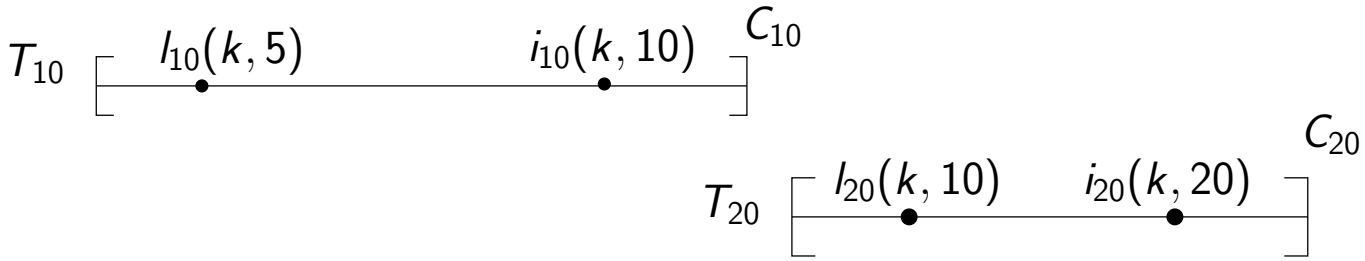


Figure 38: T_{20} looks up the version inserted by T_{10}. No conflict.

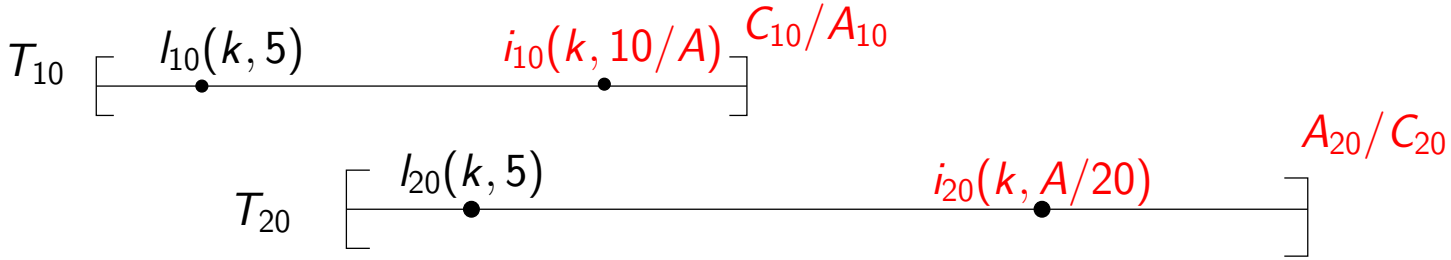


Figure 39: Conflict detected at i_{10}. Either abort T_{10} or T_{20}

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Permutations of operations

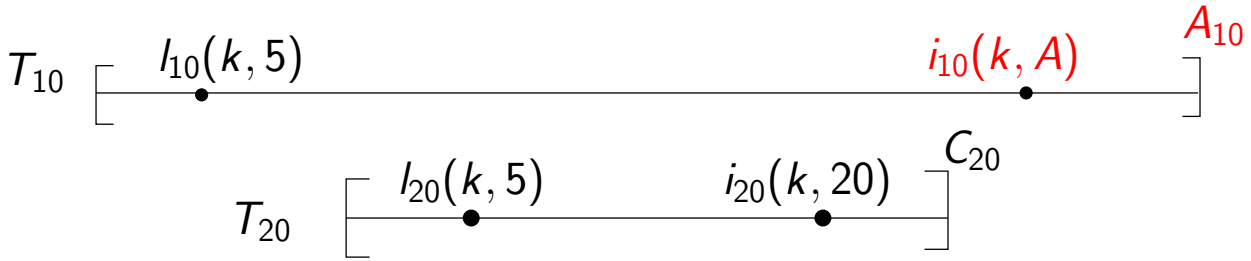


Figure 40: Conflict detected at i_{10}. Hence, abort T_{10}

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Permutations of operations

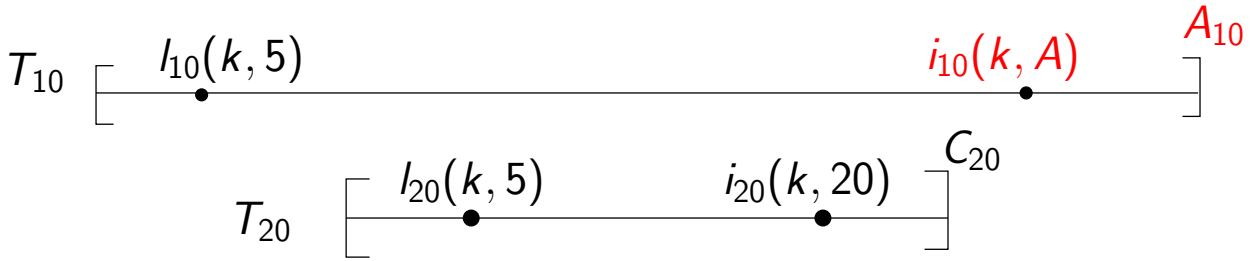


Figure 40: Conflict detected at i_{10}. Hence, abort T_{10}

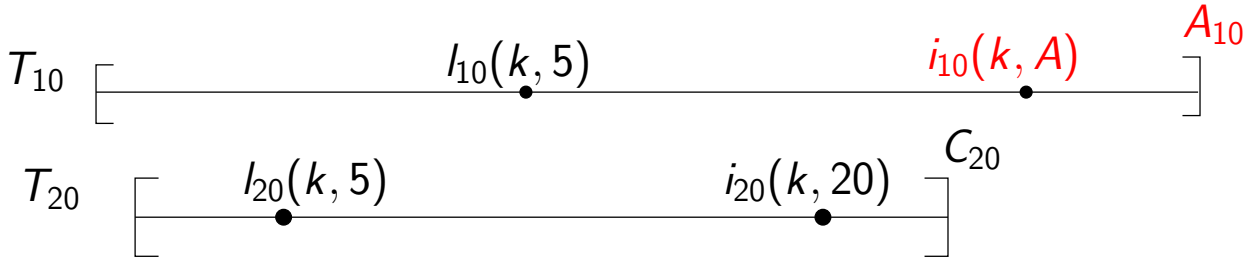


Figure 41: Conflict detected at i_{10}. Hence, abort T_{10}

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Permutations of operations

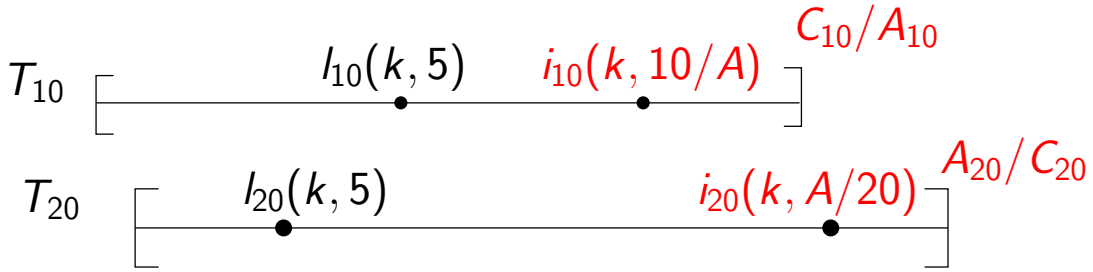


Figure 42: Conflict detected at i_{10}. Abort T_{10}

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Permutations of operations

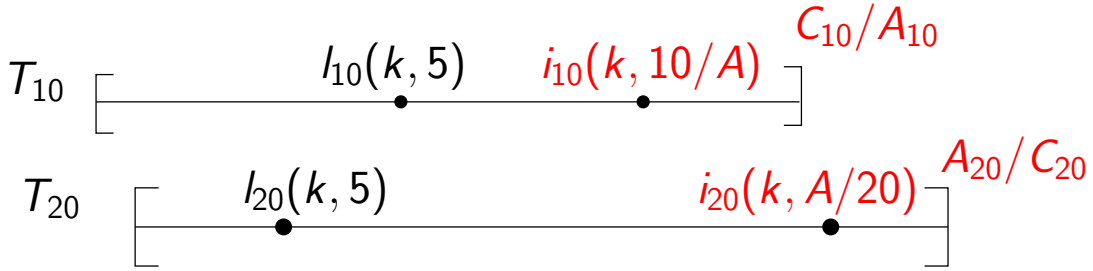


Figure 42: Conflict detected at i_{10}. Abort T_{10}

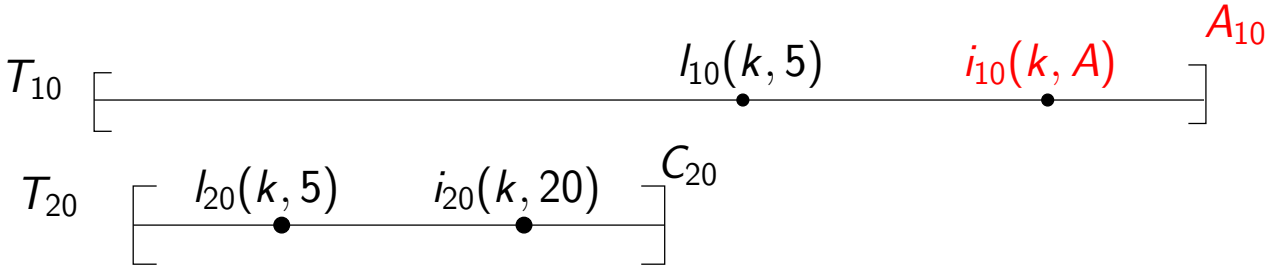


Figure 43: Conflict detected at i_{10}. Hence, abort T_{10}

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Permutations of operations

S. No.	Execution Sequence	Possible actions by Transactions
1.	$l_{10}(k), i_{10}(k), l_{20}(k), i_{20}(k)$	$T_{20}(k)$ lookups the version inserted by T_{10}. No conflict.
2.	$l_{10}(k), l_{20}(k), i_{10}(k), i_{20}(k)$	Conflict detected at $i_{10}(k)$. Either abort T_{10} or T_{20}.
3.	$l_{10}(k), l_{20}(k), i_{20}(k), i_{10}(k)$	Conflict detected at $i_{10}(k)$. Hence, abort T_{10}.
4.	$l_{20}(k), l_{10}(k), i_{20}(k), i_{10}(k)$	Conflict detected at $i_{10}(k)$. Hence, abort T_{10}.
5.	$l_{20}(k), l_{10}(k), i_{10}(k), i_{20}(k)$	Conflict detected at $i_{10}(k)$. Either abort T_{10} or T_{20}.
6.	$l_{20}(k), i_{20}(k), l_{10}(k), i_{10}(k)$	Conflict detected at $i_{10}(k)$. Hence, abort T_{10}.

Table 1: Possible Permutations of Methods

Starvation-Freedom in Multi-Version OSTMs Cont'd..

Key Observation

Takeaway

A transaction cannot be aborted if

- It has the lowest *its*, i.e., it is the earliest transaction and
- It has the highest *cts*

return

Experimental Evaluations

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

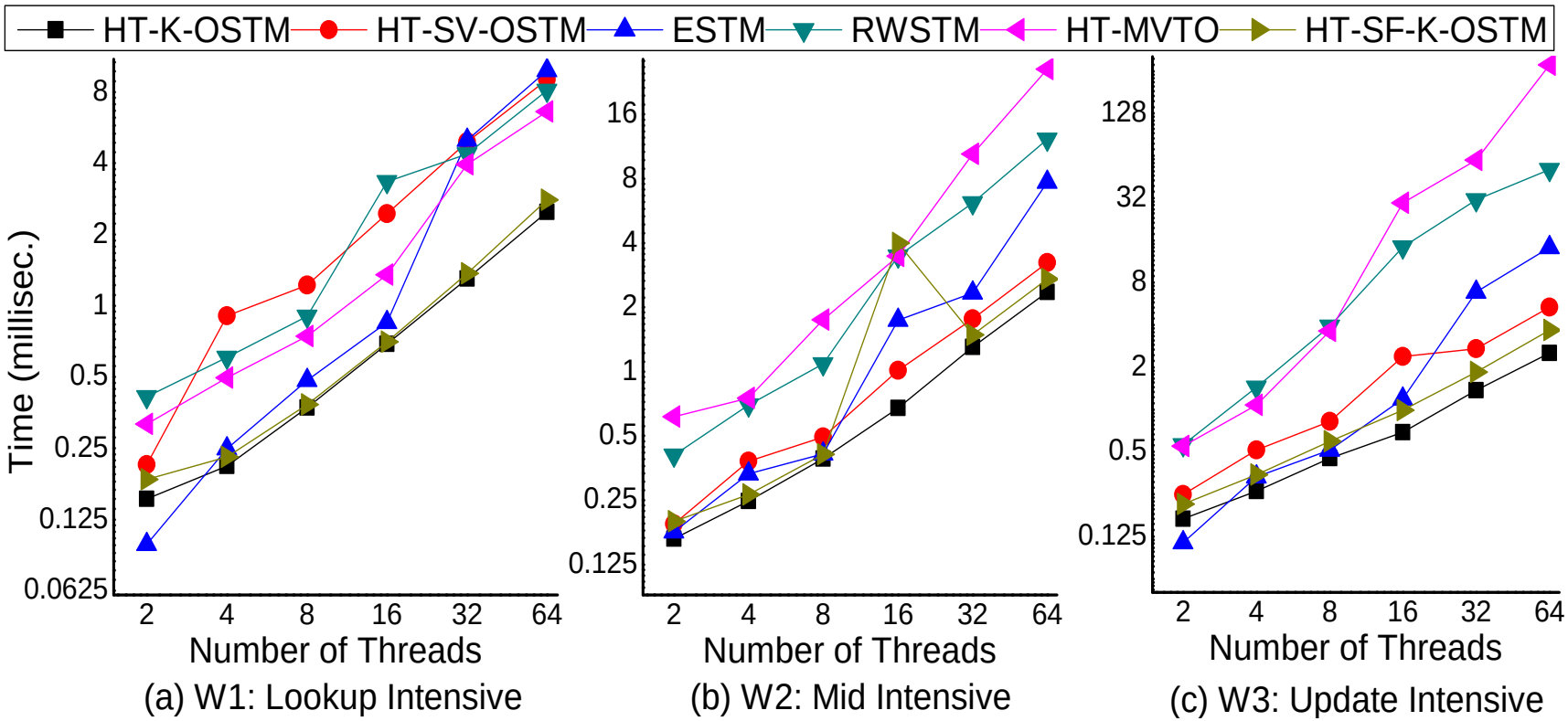


Figure 44: Time comparison of SF-K-OSTM and State-of-the-art STMs on hash table

Experimental Evaluations

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

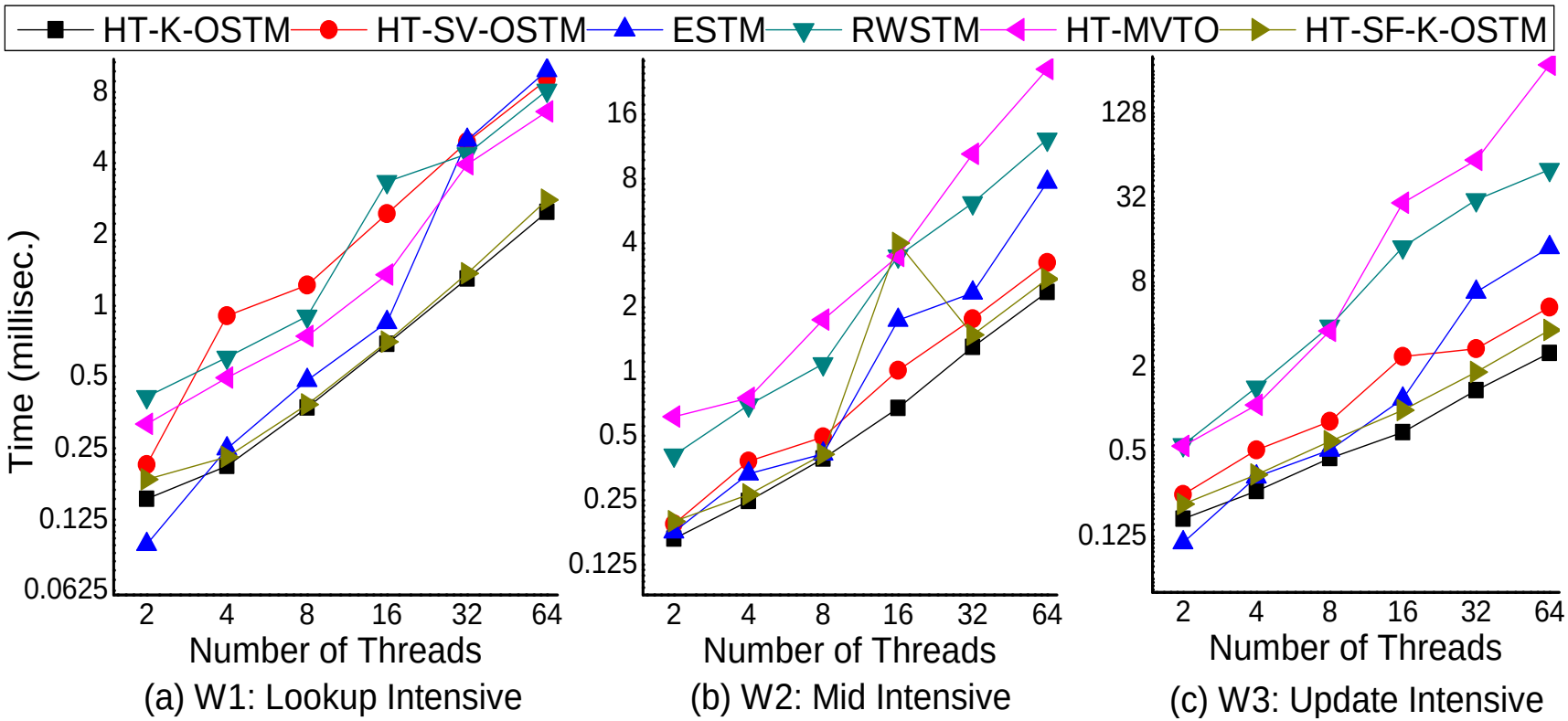


Figure 44: Time comparison of SF-K-OSTM and State-of-the-art STMs on hash table

- Proposed hash table based SF-K-OSTM (HT-SF-K-OSTM) performs better than all other state-of-art-STMs algorithms but slightly lesser than the non starvation-free HT-K-OSTM.

Experimental Evaluations Cont'd..

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

Algorithm	W1	W2	W3
HT-SF-SV-OSTM	1.49	1.13	1.09
HT-SF-MV-OSTM	1.22	1.08	1.04
HT-SF-MV-OSTM-GC	1.13	1.03	1.02
HT-SV-OSTM	3.3	0.77	1.57
ESTM	2.92	1.4	3.03
RWSTM	3.13	2.65	13.36
HT-MVTO	2.37	4.74	49.39
HT-K-OSTM	0.91	0.7	0.8

Table 2: Speedup by HT-SF-K-OSTM

Experimental Evaluations Cont'd..

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

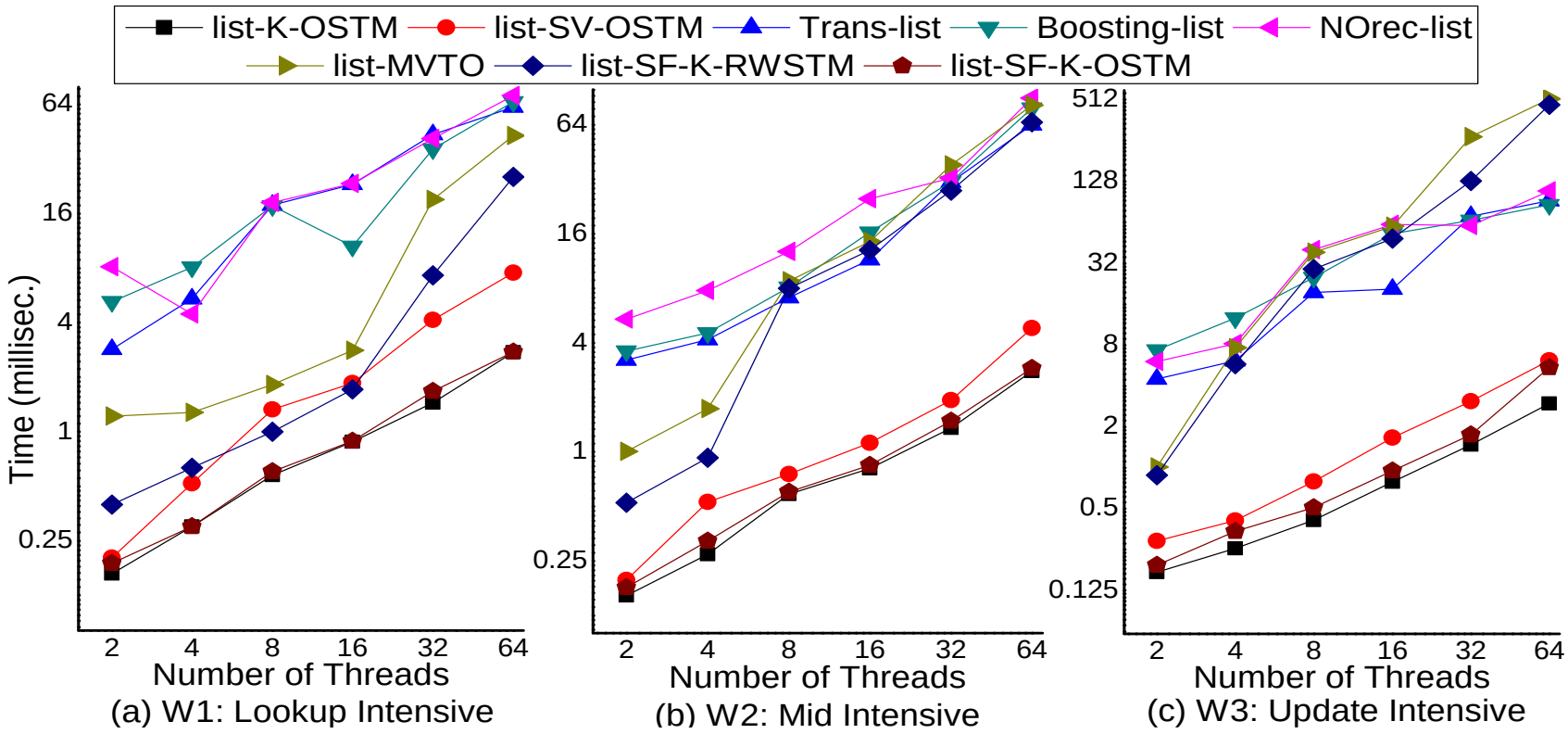


Figure 45: Time comparison of SF-K-OSTM and State-of-the-art STMs on list

Experimental Evaluations Cont'd..

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

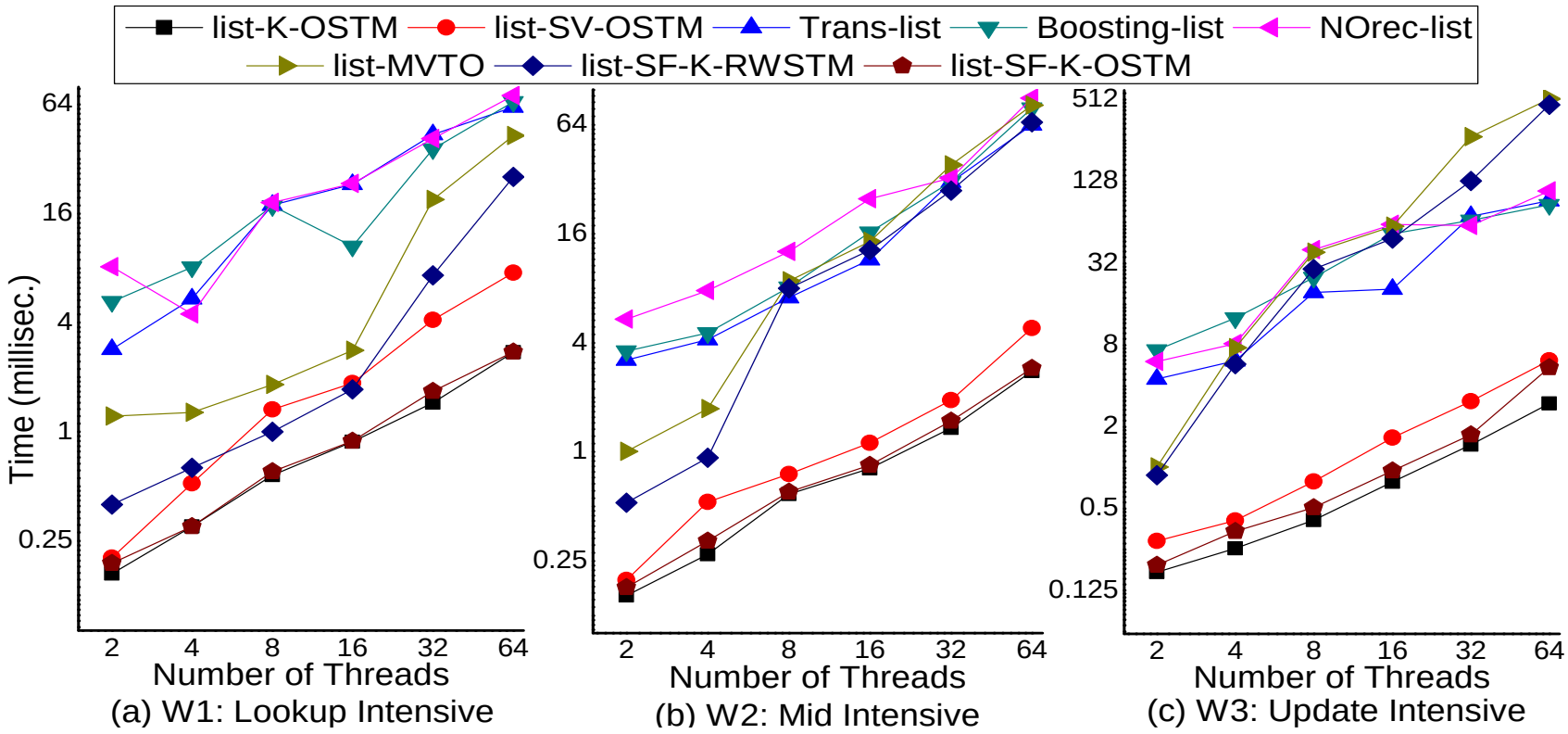


Figure 45: Time comparison of SF-K-OSTM and State-of-the-art STMs on list

- Proposed list based SF-K-OSTM (list-SF-K-OSTM) performs better than all other state-of-art-STMs algorithms but slightly lesser than the non starvation-free list-K-OSTM.

Experimental Evaluations Cont'd..

Low Contention Environment: 1000 Shared data-items & 10 operations/transaction

Algorithm	W1	W2	W3
list-SF-SV-OSTM	1.29	1.26	1.22
list-SF-MV-OSTM	1.25	1.29	1.12
list-SF-MV-OSTM-GC	1.14	1.5	1.13
list-SV-OSTM	2.4	1.5	1.4
Trans-list	24.15	19.06	23.26
Boosting-list	22.43	22.52	27.2
NOrec-list	26.12	27.33	31.05
list-MVTO	10.8	23.1	19.57
list-SF-K-RWSTM	5.7	18.4	74.20
list-K-OSTM	0.96	0.98	0.8

Table 3: Speedup by list-SF-K-OSTM

return