भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

MICHIGAN STATE UNIVERSITY

An Innovative Approach to Achieve Compositionality Efficiently using Multi-Version Object Based Transactional 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

20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2018)

Outline

Outline

Transaction

Sequence of instructions guaranteed to execute atomically.

Transaction

Sequence of instructions guaranteed to execute atomically.

History

Concurrent execution of transactions.

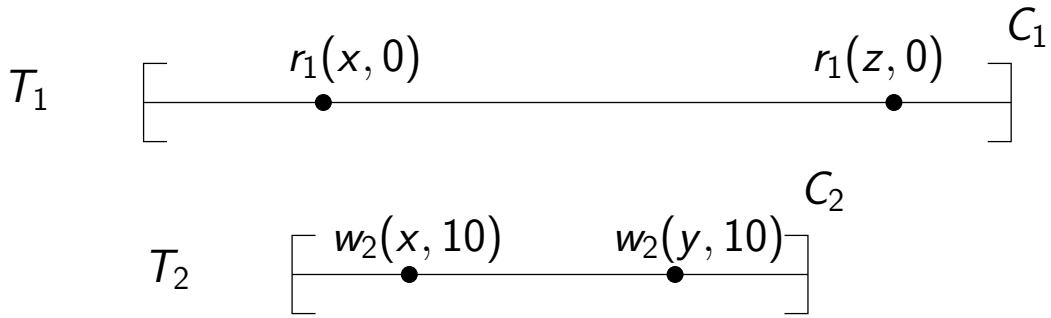


Figure: History of an STM

STMs Cont'd..

Introduction

- **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.

STMs Cont'd..

Introduction

- **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 RWSTM).

A thread safe concurrent linked-list(LL) using STM

Algorithm 1 Insert(LL, e): Invoked by a thread to insert an element e into a linked-list LL. This method is implemented using transactions.

```
1:  retry = 0;
2:  while (true) do
3:      id = t_begin(retry);
4:      ...
5:      ...
6:      v = t_read(id, x);                    ▷ reads the value of x as v
7:      ...
8:      ...
9:      t_write(id, x, v');                   ▷ writes a value v' to x
10:     ...
11:     ...
12:     ret = tryC(id);                       ▷ tryC can return commit or abort
13:     if (ret == commit) then
14:         break;
15:     else
16:         retry++;
17:     end if
18: end while
```

Working of STMs methods

Working of STMs methods

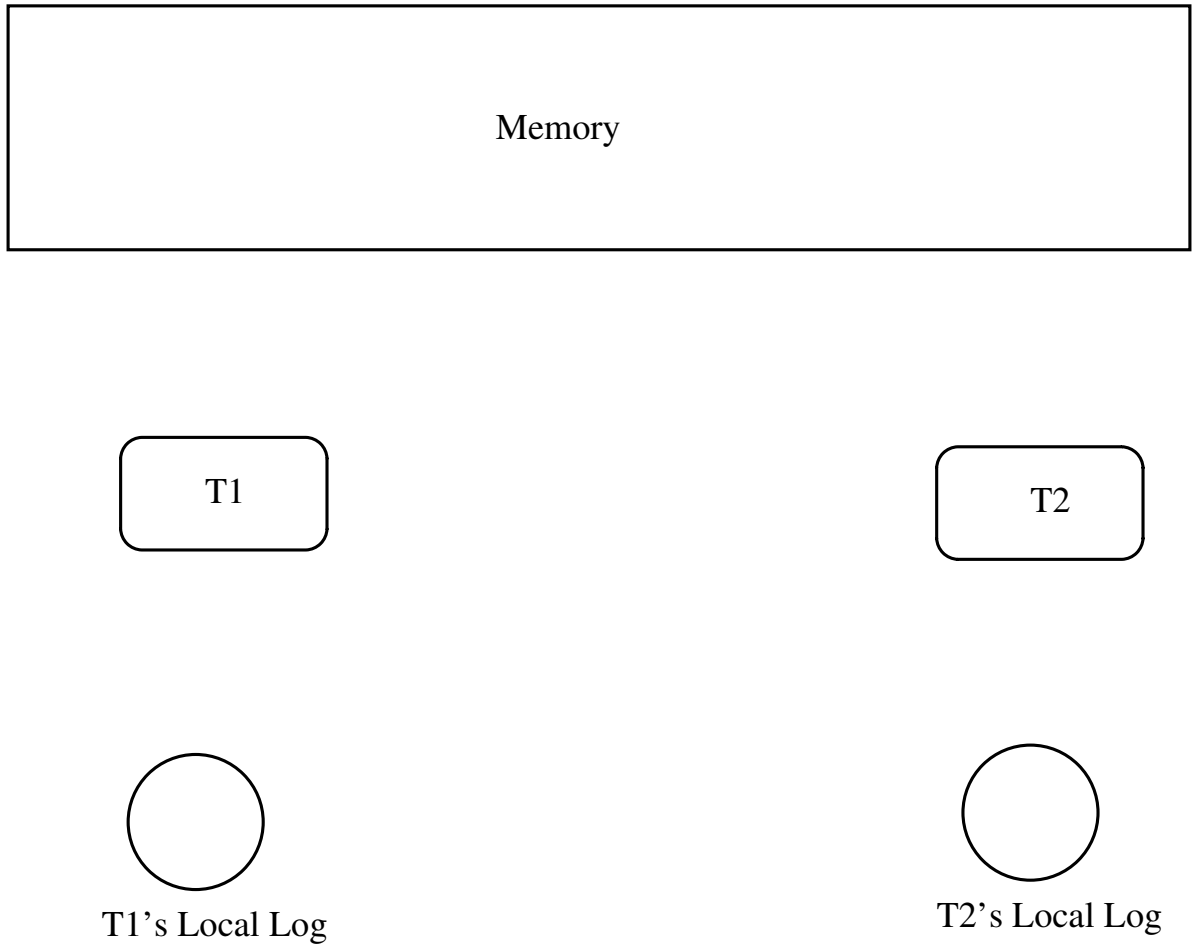


Figure: Working of STM System

Working of STMs methods

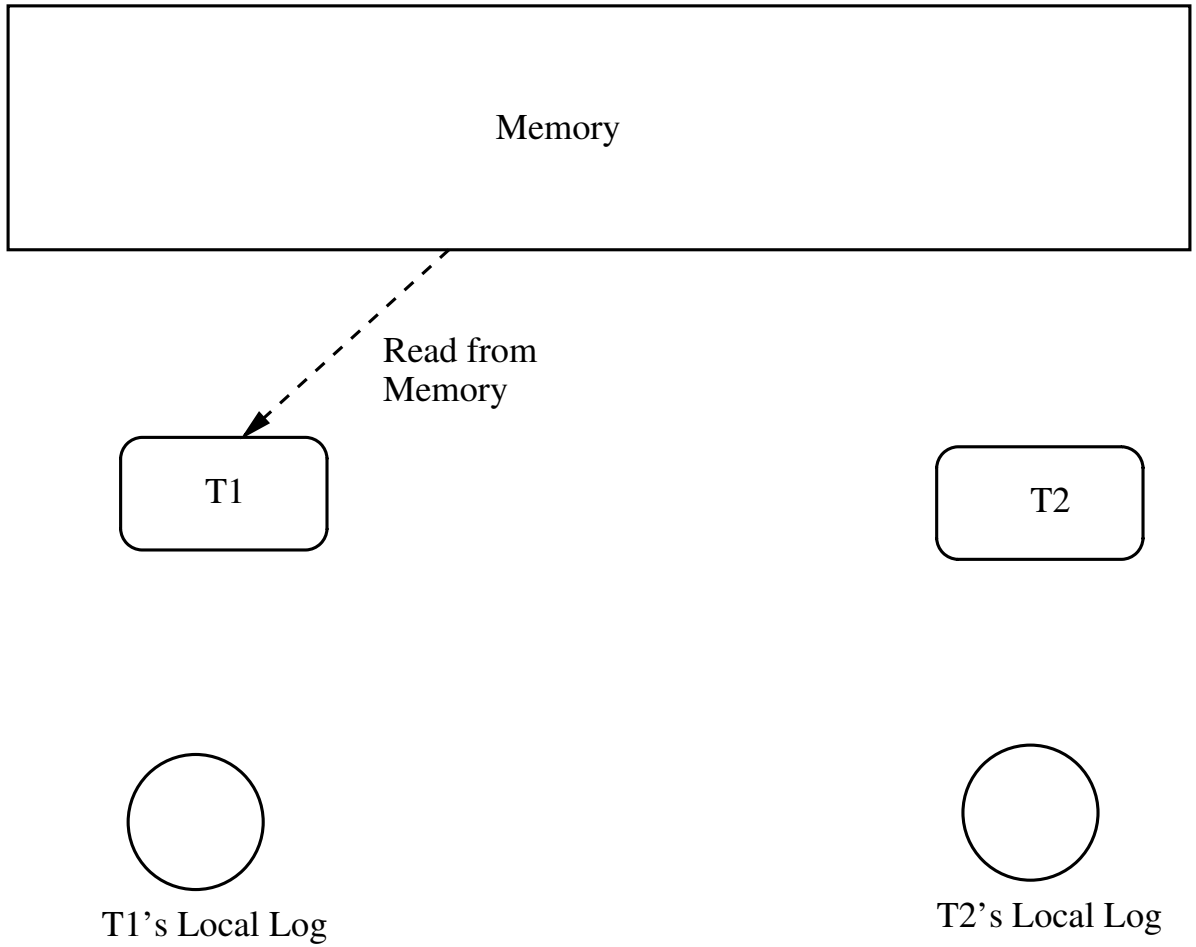


Figure: Working of STM System

Working of STMs methods

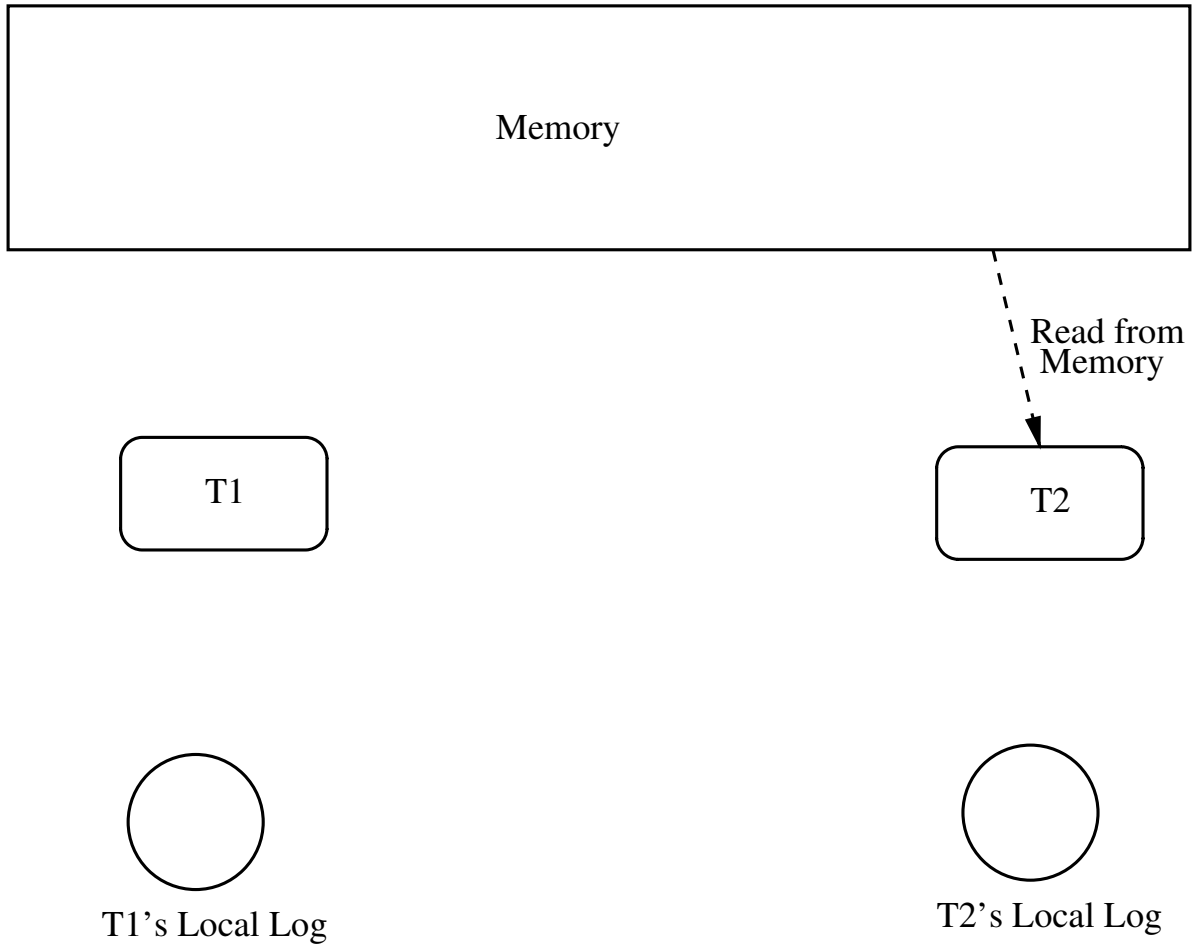


Figure: Working of STM System

Working of STMs methods

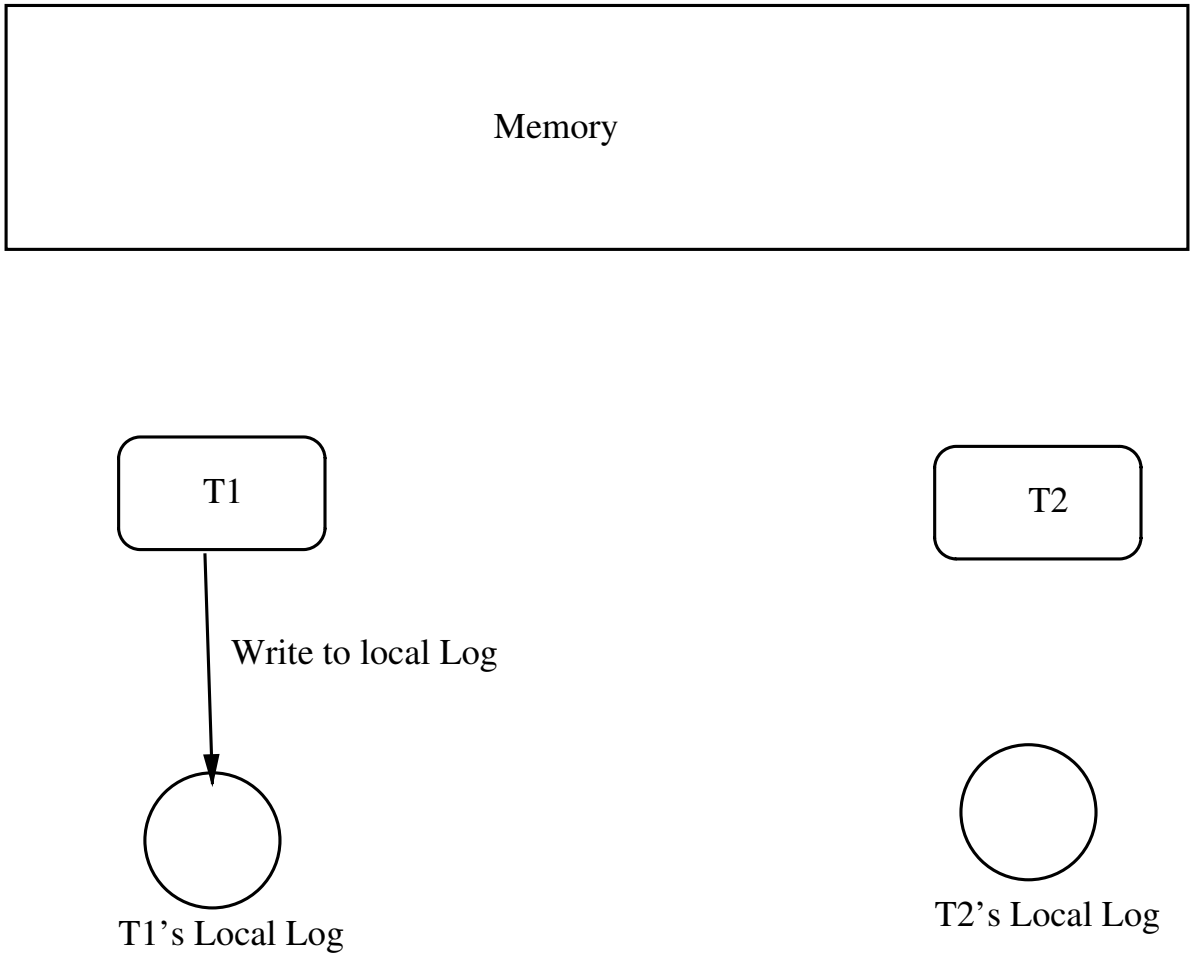


Figure: Working of STM System

Working of STMs methods

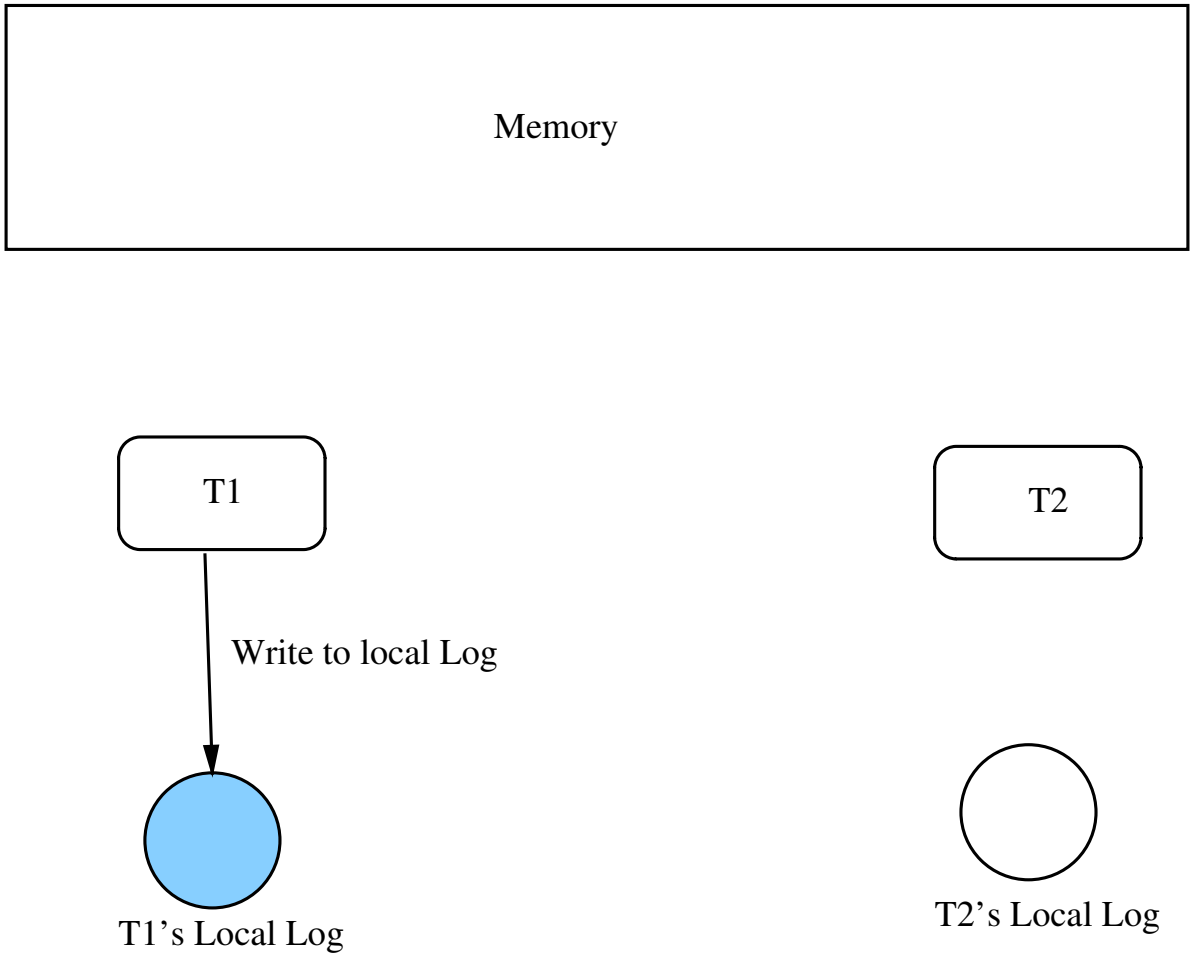


Figure: Working of STM System

Working of STMs methods

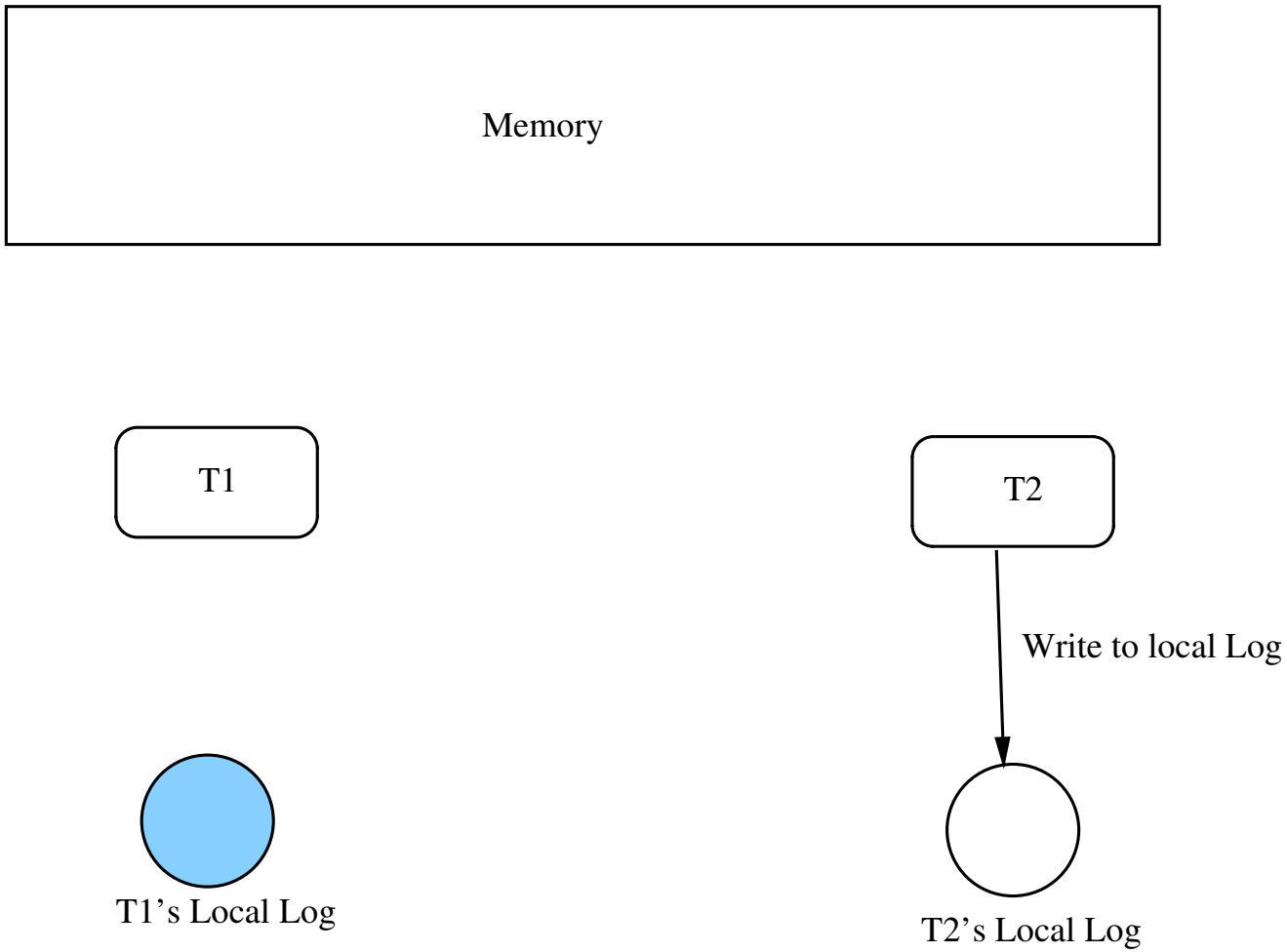


Figure: Working of STM System

Working of STMs methods

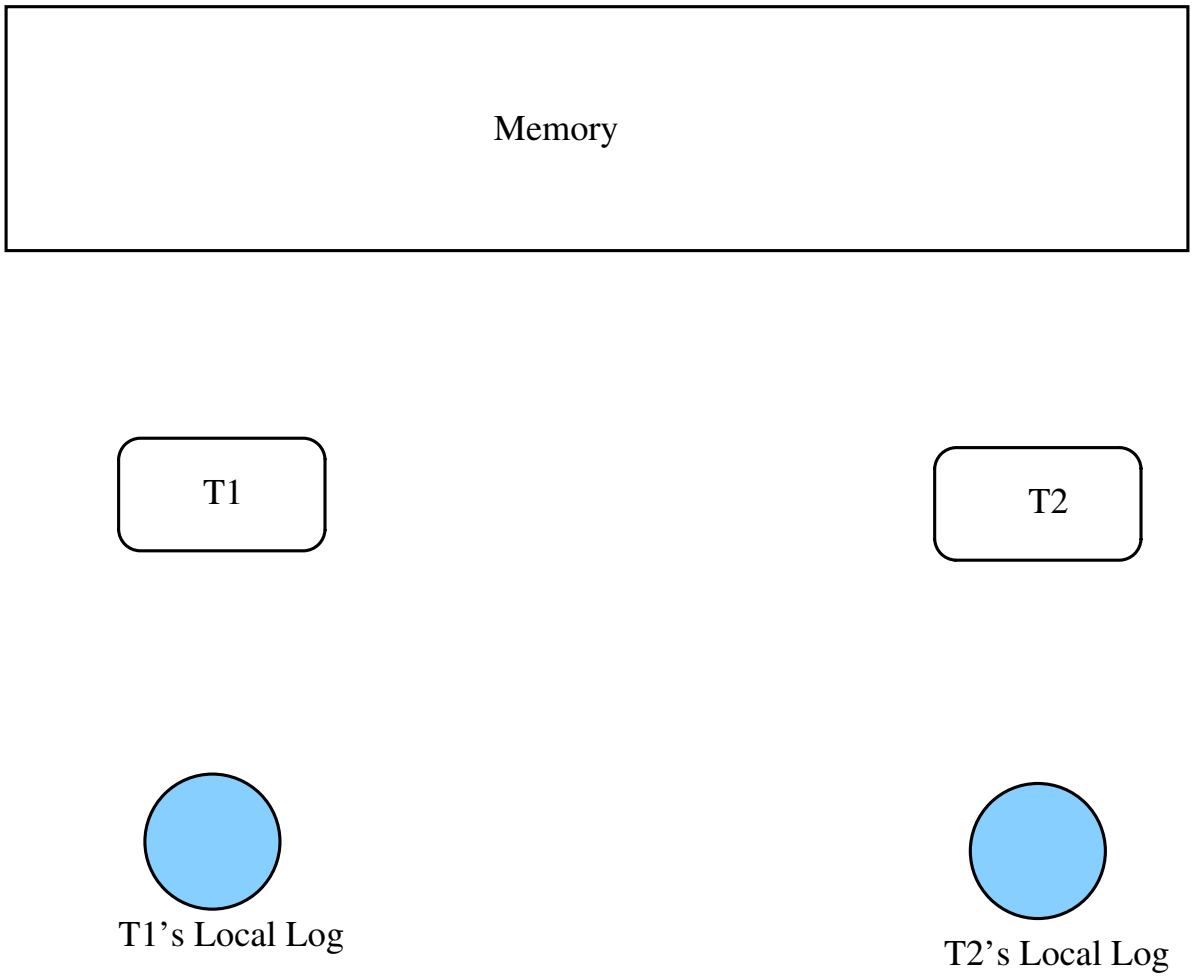


Figure: Working of STM System

Working of STMs methods

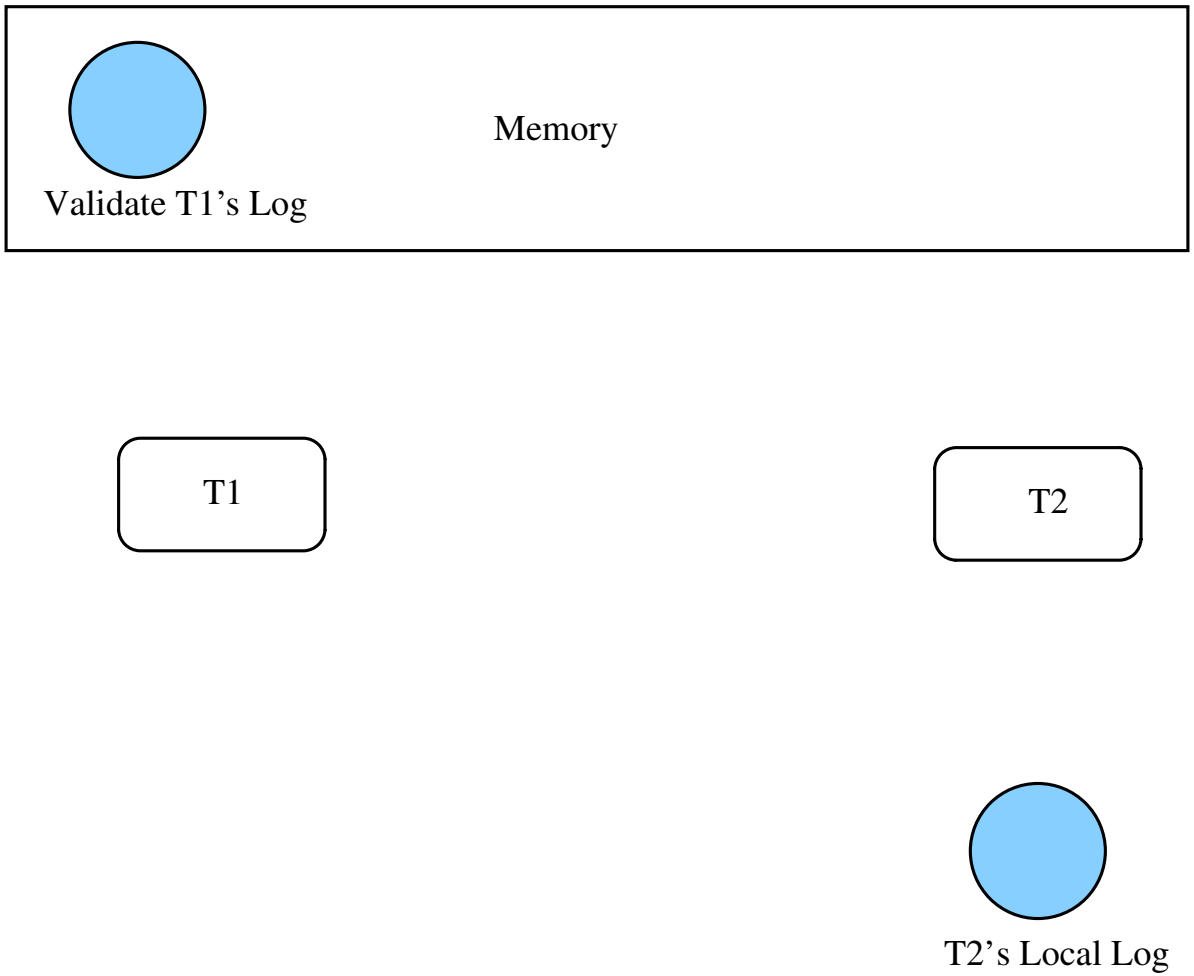


Figure: Working of STM System

Working of STMs methods

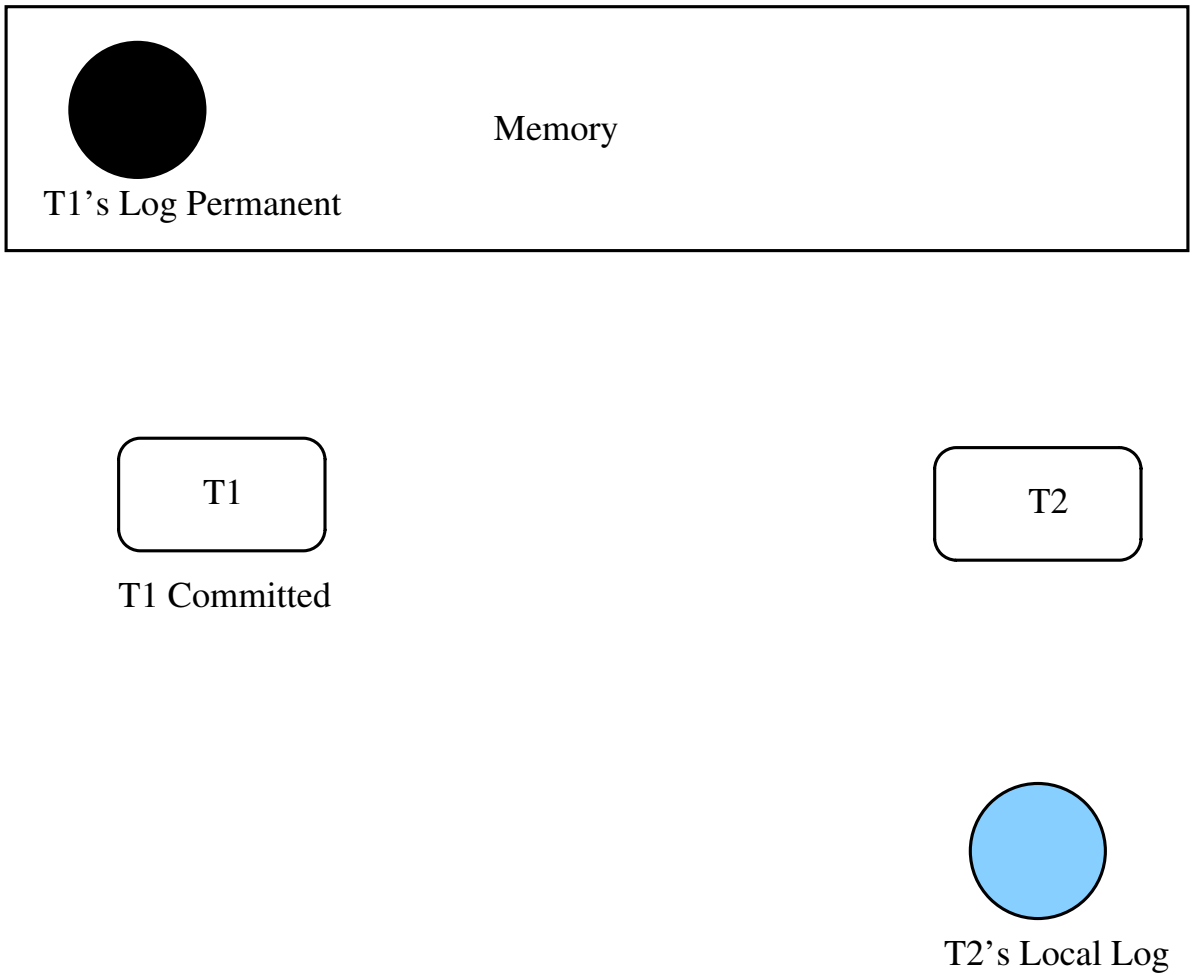


Figure: Working of STM System

Working of STMs methods

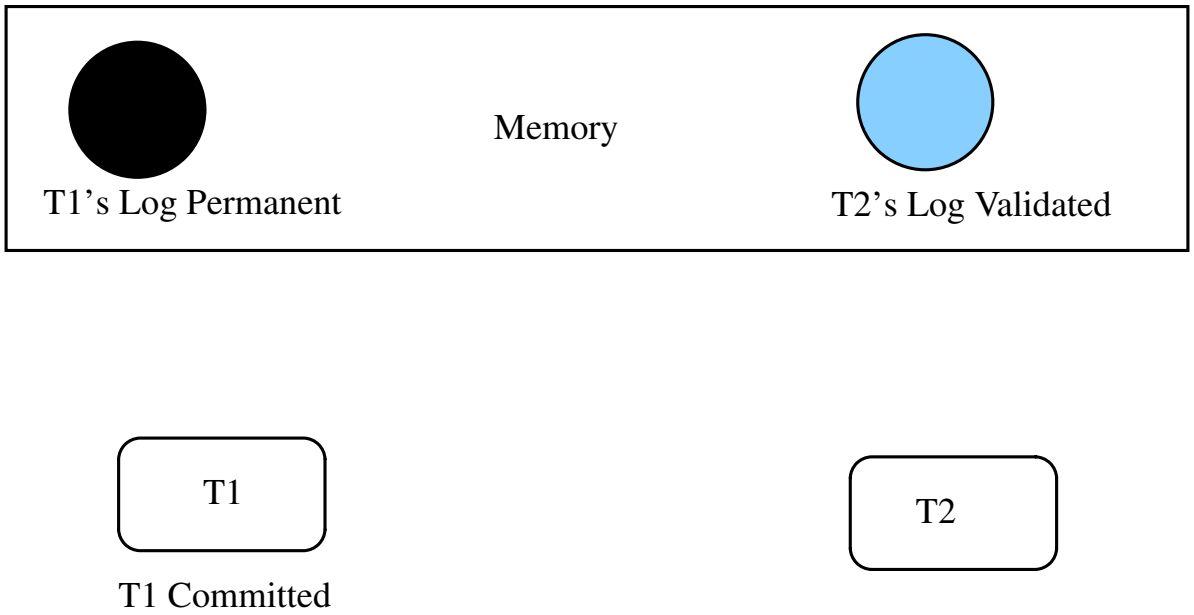


Figure: Working of STM System

Working of STMs methods

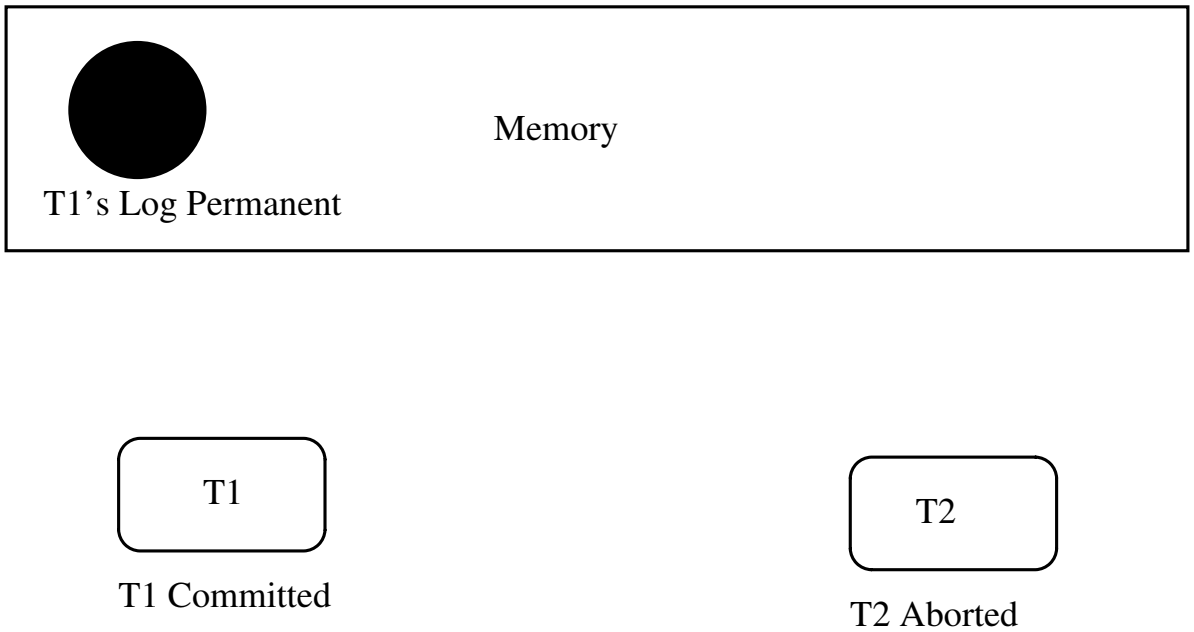


Figure: Working of STM System

Correctness of STM System: Opacity [a]

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

Correctness of STM System: 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 STM System Cont'd..

Example of opacity

- H: $r_1(x,0)\, w_2(x,10)\, w_2(y,10)\, C_2\, r_1(y,A)\, A_1$

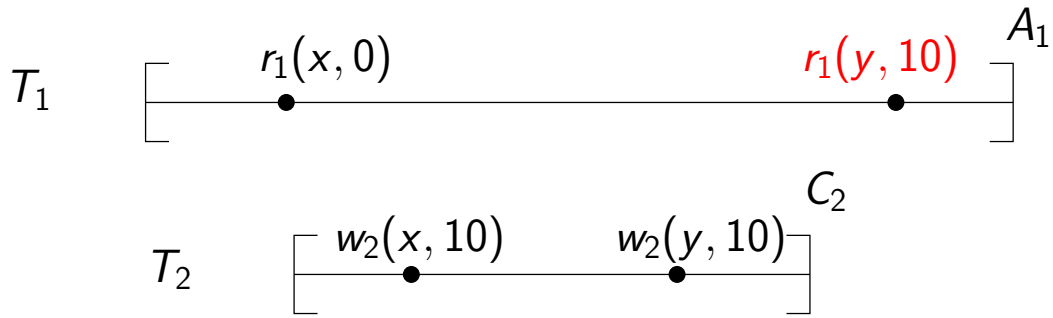


Figure: History H is not Opaque but Serializable

Correctness of STM System Cont'd..

Example of opacity

- H: $r_1(\mathrm{x},0)\, w_2(\mathrm{x},10)\, w_2(\mathrm{y},10)\, C_2\, r_1(\mathrm{y},\mathrm{A})\, A_1$

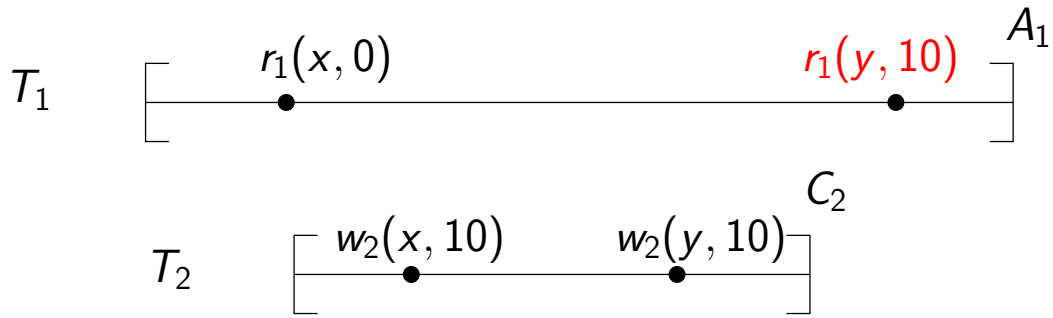


Figure: History H is not Opaque but Serializable

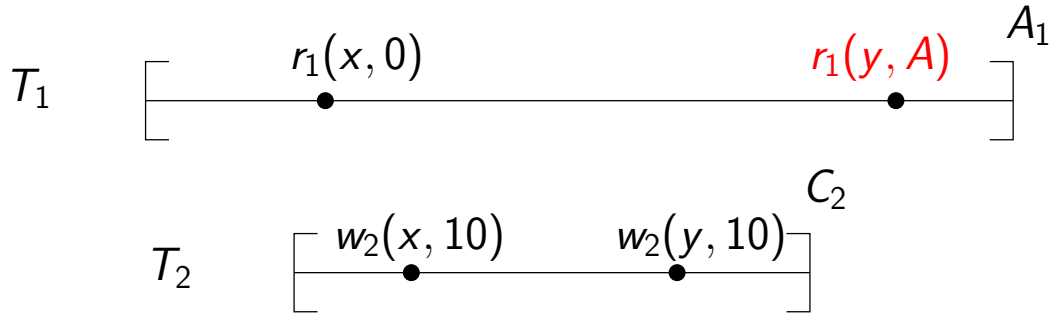


Figure: Opaque History H

Correctness of STM System Cont'd..

Example of opacity

- H: $r_1(x,0)\, w_2(x,10)\, w_2(y,10)\, C_2\, r_1(y,A)\, A_1$

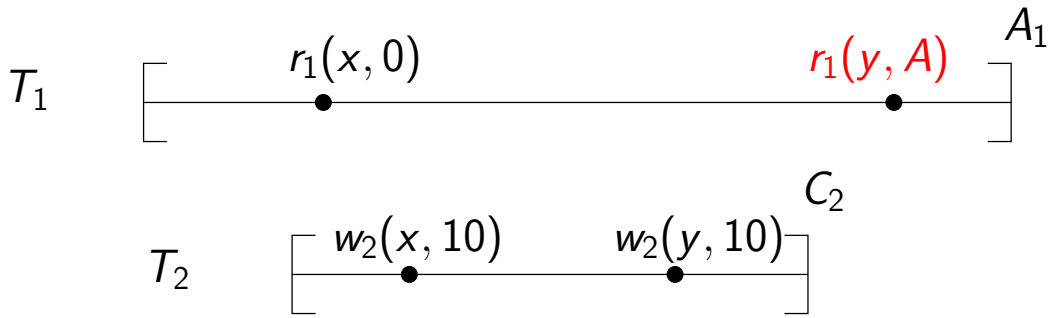


Figure: Opaque History H

Correctness of STM System Cont'd..

Example of opacity

- H: $r_1(\mathrm{x},0)\, w_2(\mathrm{x},10)\, w_2(\mathrm{y},10)\, C_2\, r_1(\mathrm{y},\mathrm{A})\, A_1$

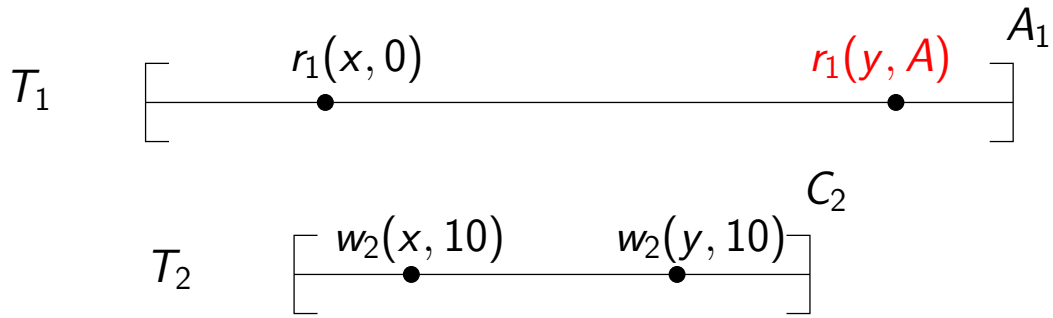


Figure: Opaque History H

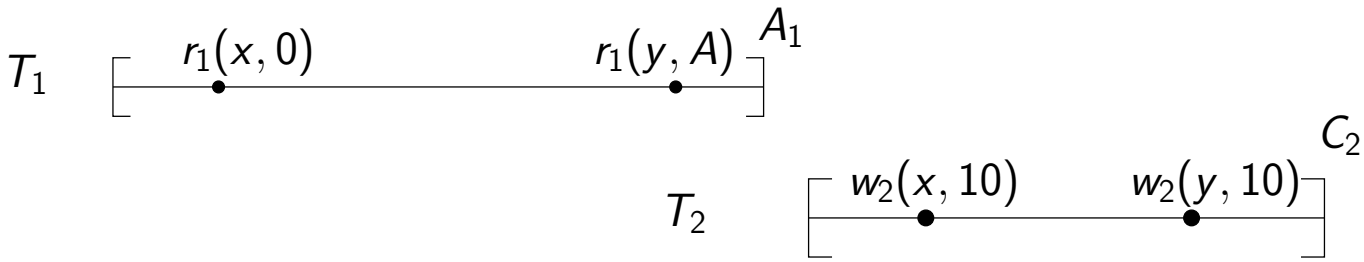


Figure: Equivalent serial history S: T_1, T_2

Outline

Problem with read-write STMs

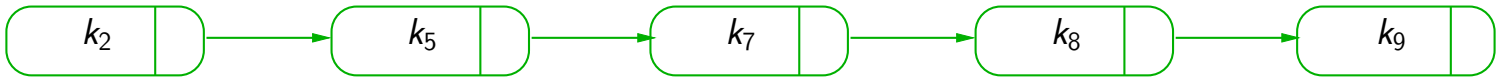


Figure: A sample concurrent list

Problem with read-write STMs

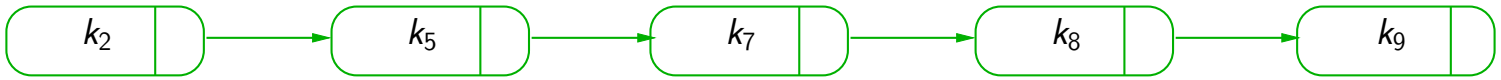


Figure: A sample concurrent list

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

Problem with read-write STMs

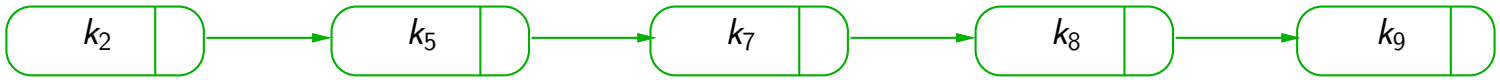


Figure: A sample concurrent list

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

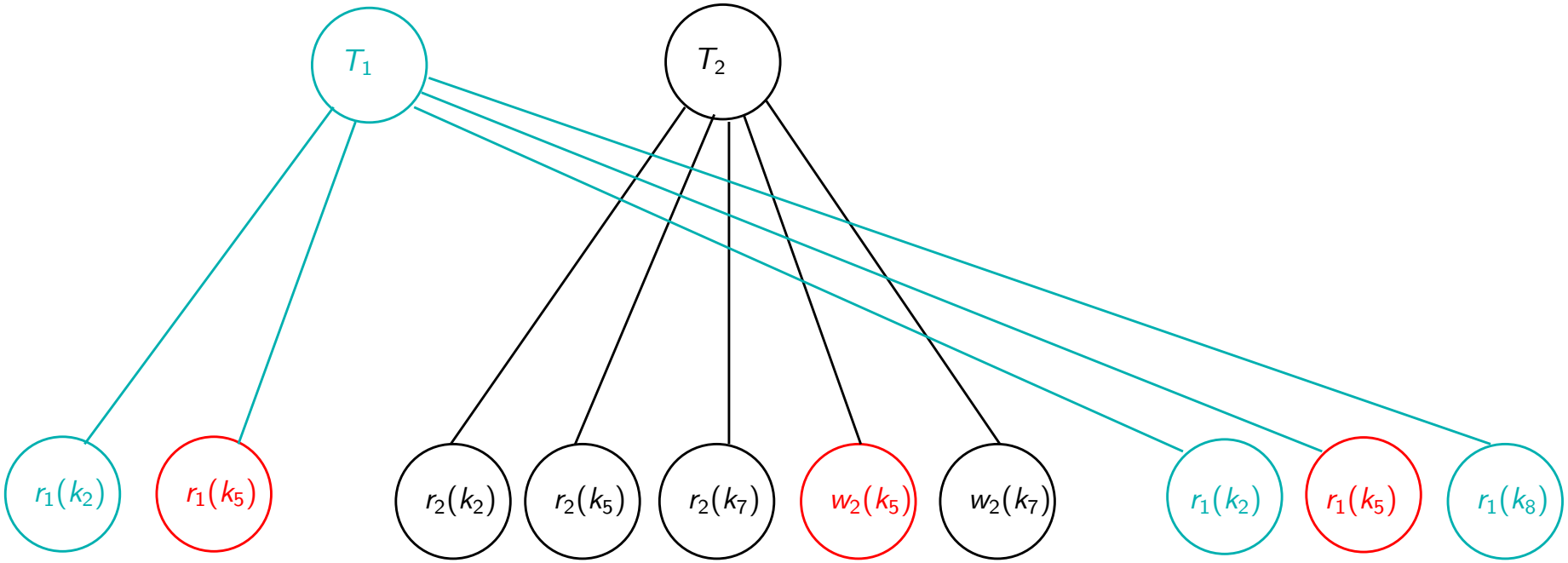


Figure: Tree Structure : conflicts are ($r_1(k_5)$, $w_2(k_5)$) and ($w_2(k_5)$, $r_1(k_5)$)

Problem at read-write level

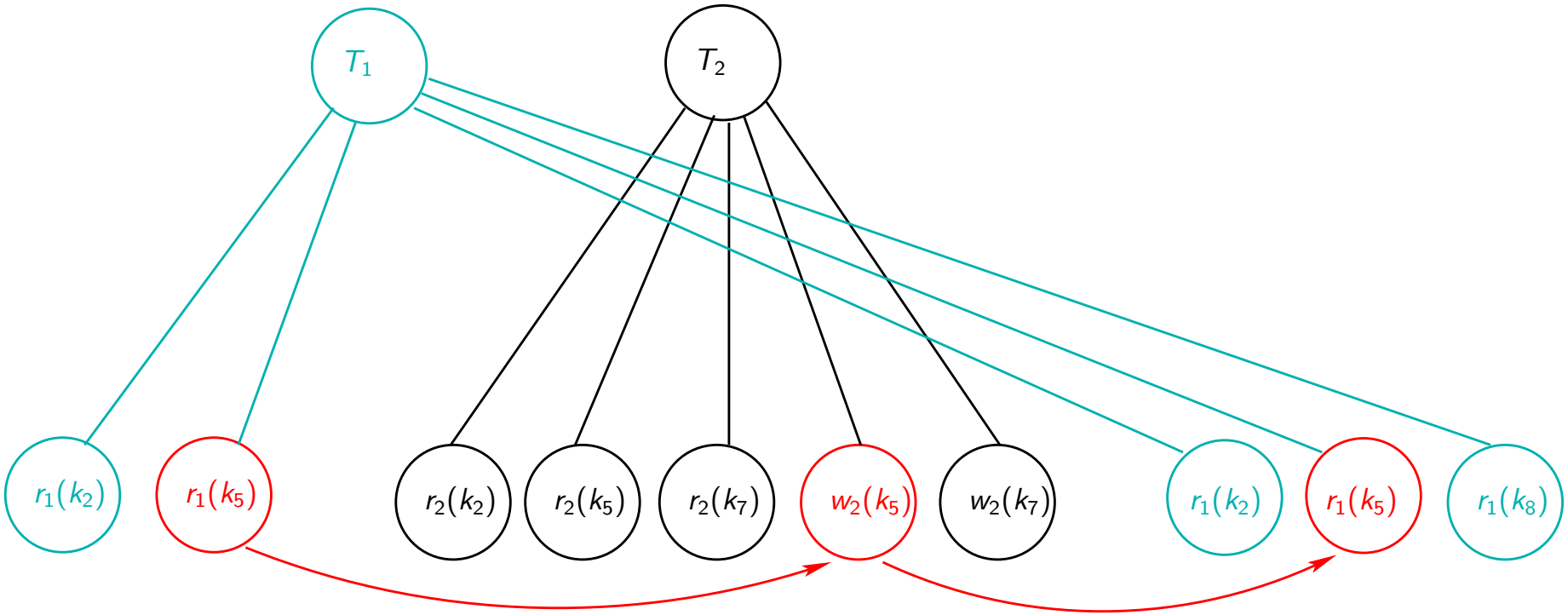


Figure: Tree Structure

Schedule cannot be accepted by a RWSTM.

Problem at read-write level

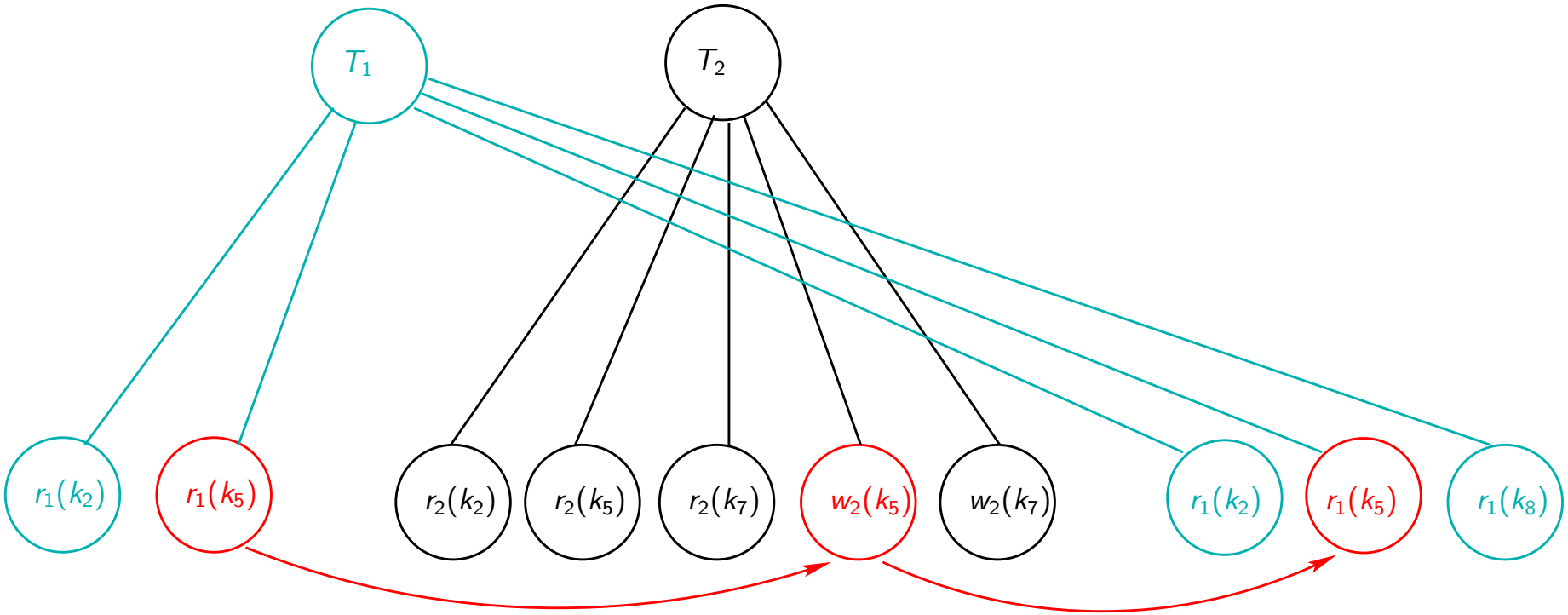


Figure: Tree Structure

Schedule cannot be accepted by a RWSTM.

Figure: Cycle (Not Serial)

Outline

- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read & 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

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

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

OSTM

Execution at layer-1

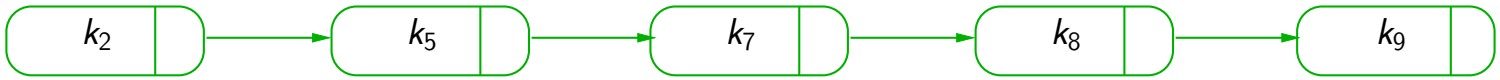


Figure: A sample concurrent list

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

OSTM

Execution at layer-1

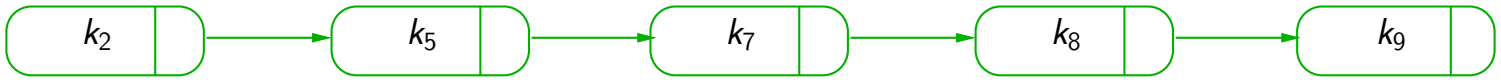


Figure: A sample concurrent list

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

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

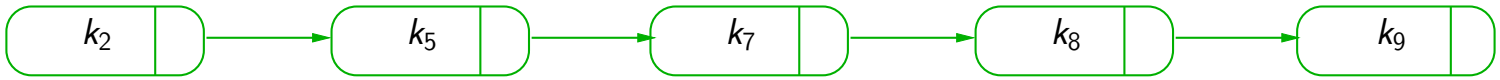


Figure: A sample concurrent list

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

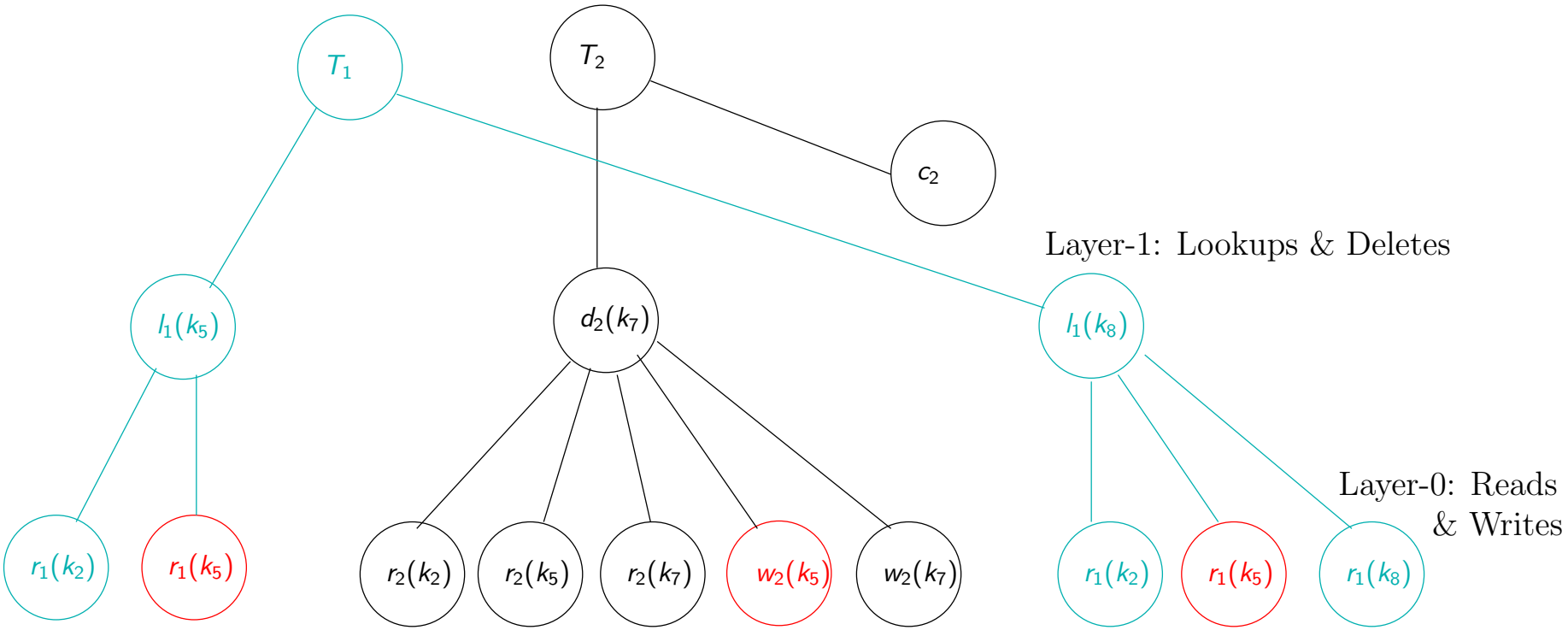


Figure: Tree Structure : no conflict at Layer-1 [d]

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

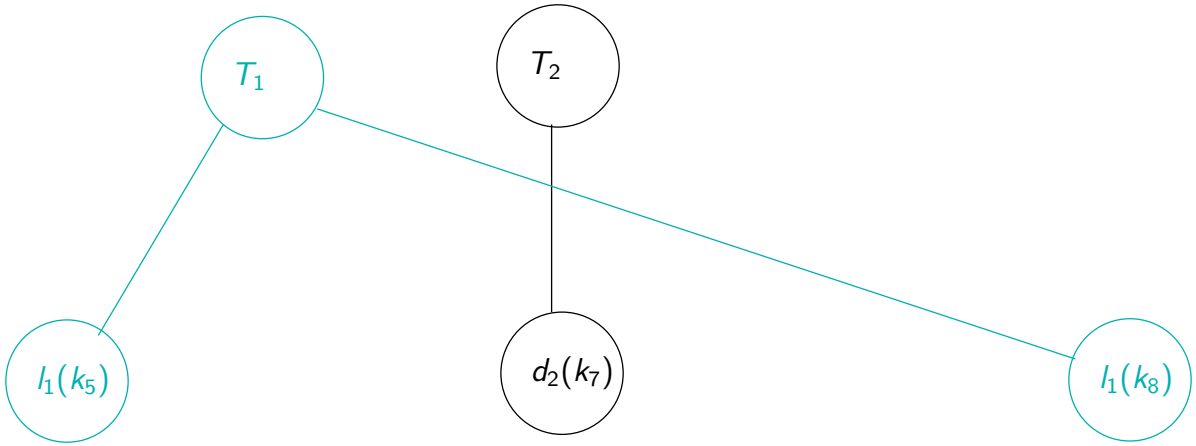


Figure: Pruned Tree

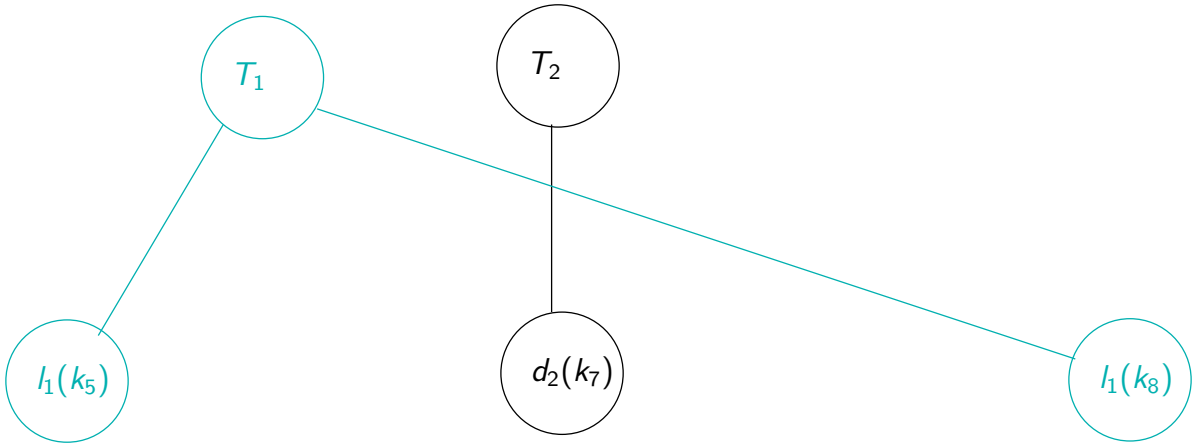


Figure: Pruned Tree

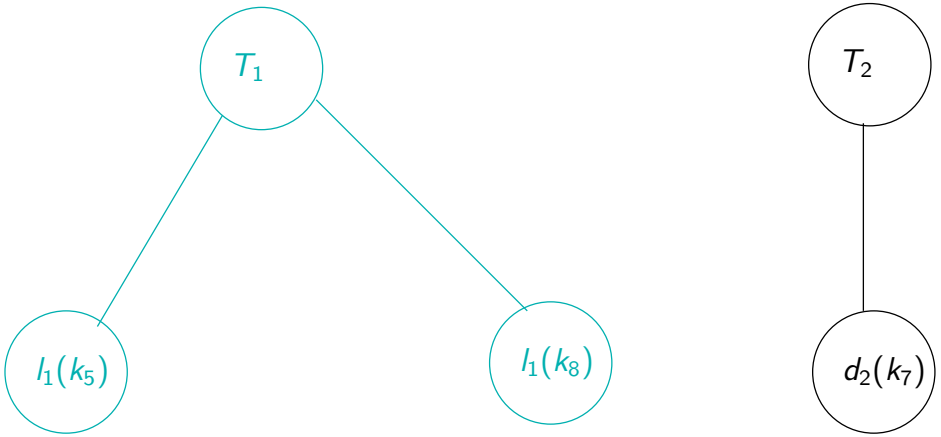


Figure: Sequential Schedule

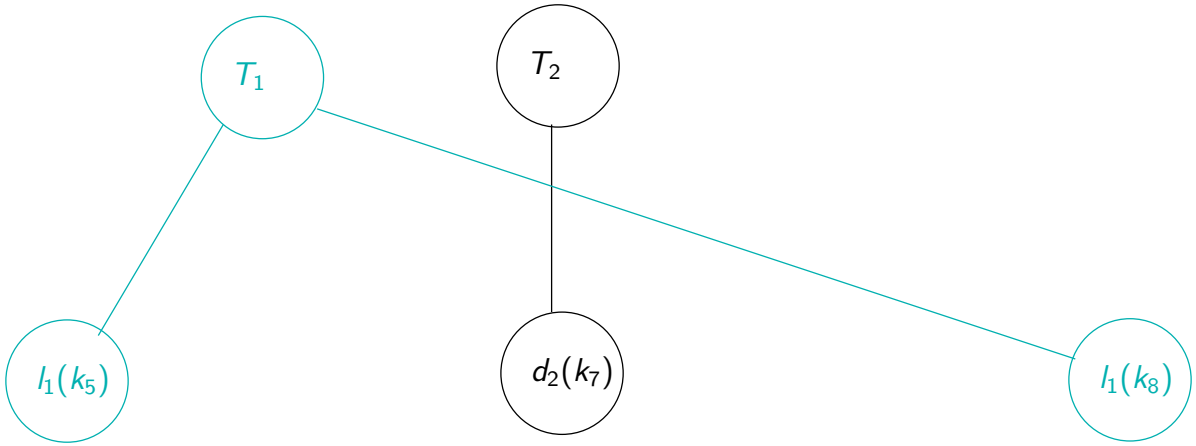
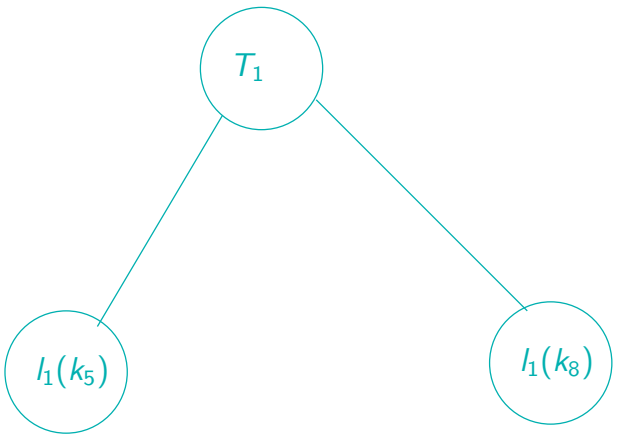


Figure: Pruned Tree



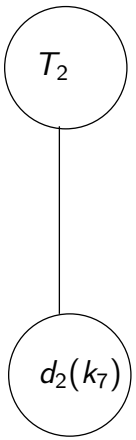


Figure: Sequential Schedule

Figure: Serial History

Inefficiency with OSTM

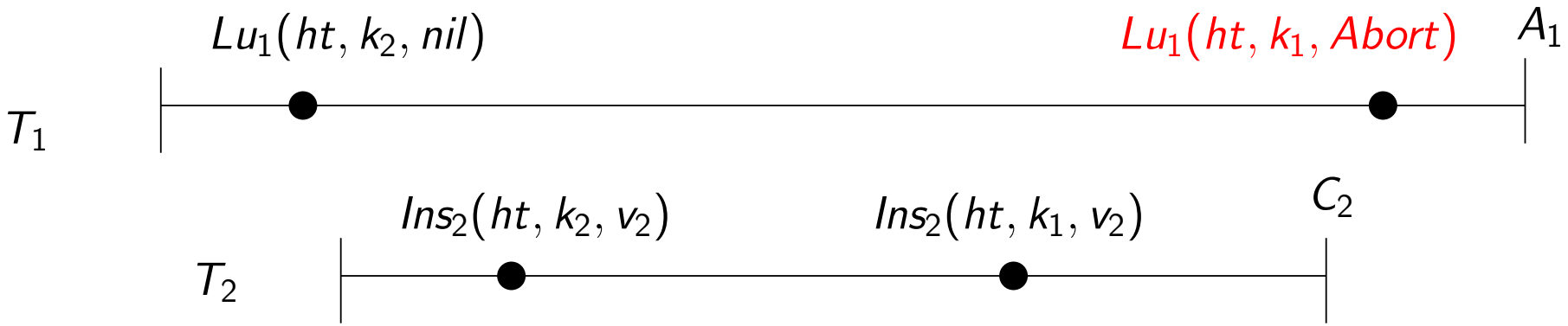


Figure: Single version OSTM

Outline

Motivation towards MV-OSTM

Advantages of multi-version[e] over single version RWSTM

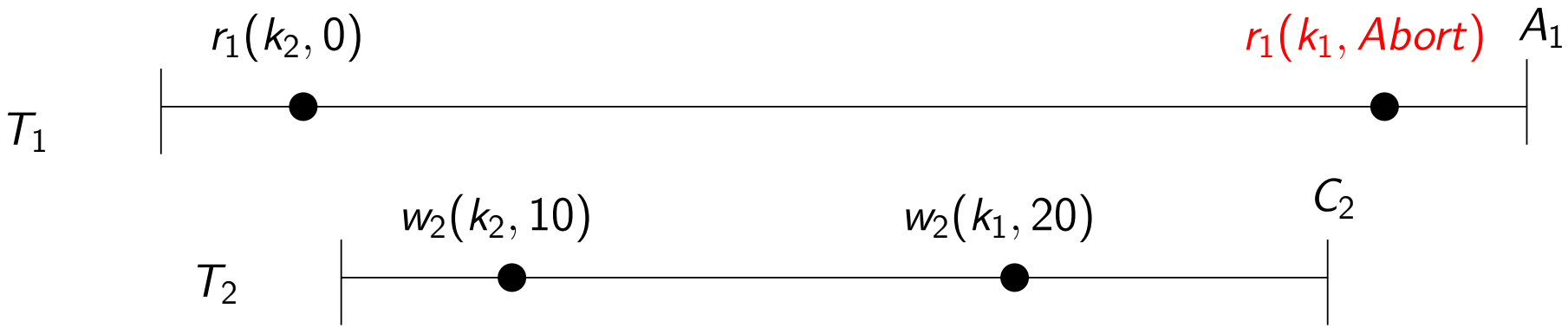


Figure: Single version RWSTM

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

Motivation towards MV-OSTM

Advantages of multi-version[e] over single version RWSTM

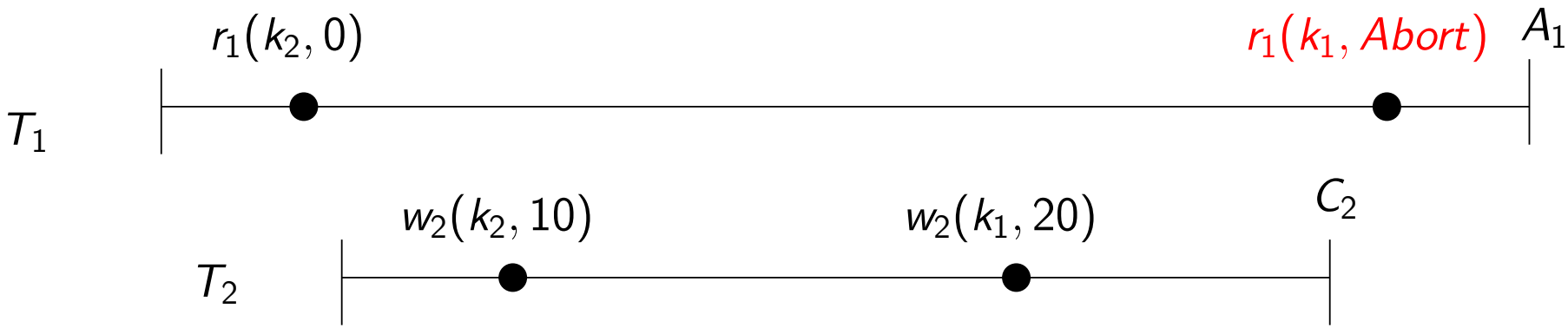


Figure: Single version RWSTM

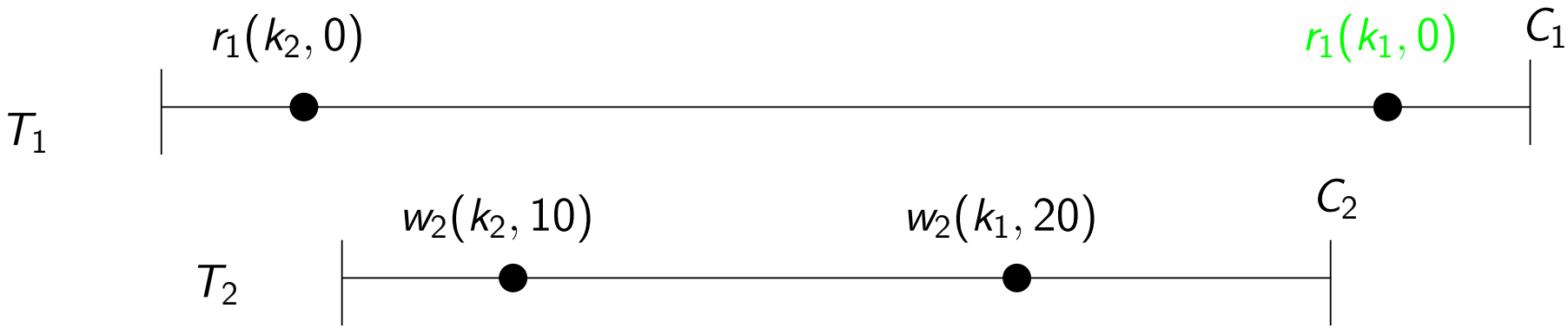


Figure: Multi-version RWSTM (MV-RWSTM) : (T_1, T_2)

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

Proposed Algorithm : MV-OSTM

Advantages of multi-version over single version OSTM

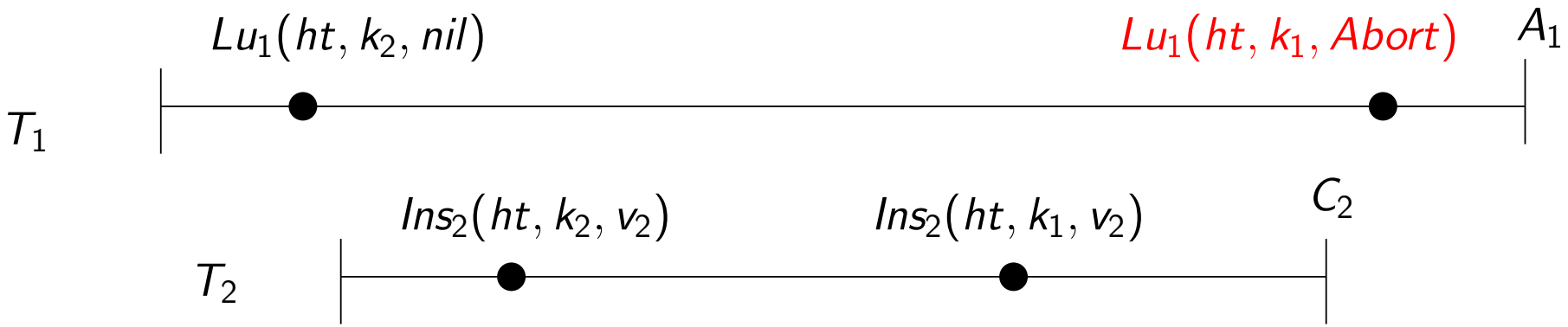


Figure: Single version OSTM

Proposed Algorithm : MV-OSTM

Advantages of multi-version over single version OSTM

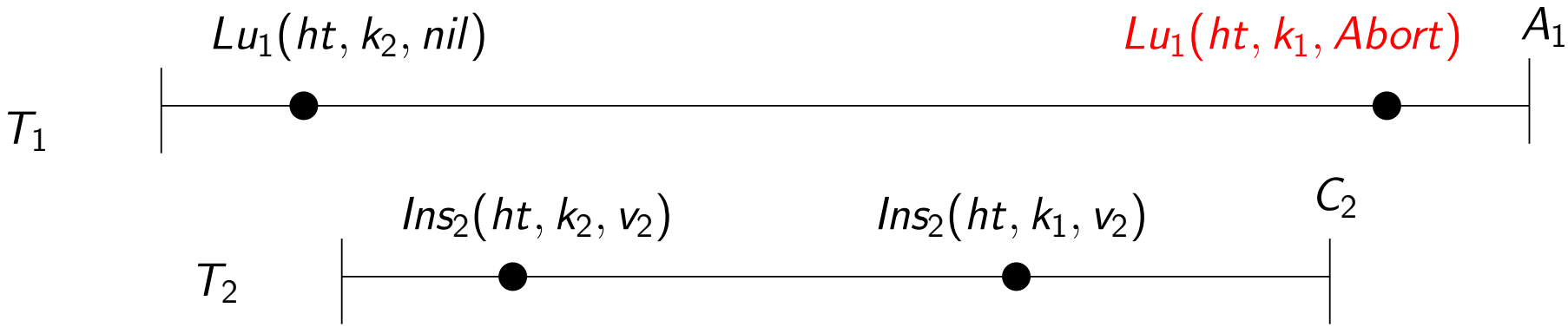


Figure: Single version OSTM

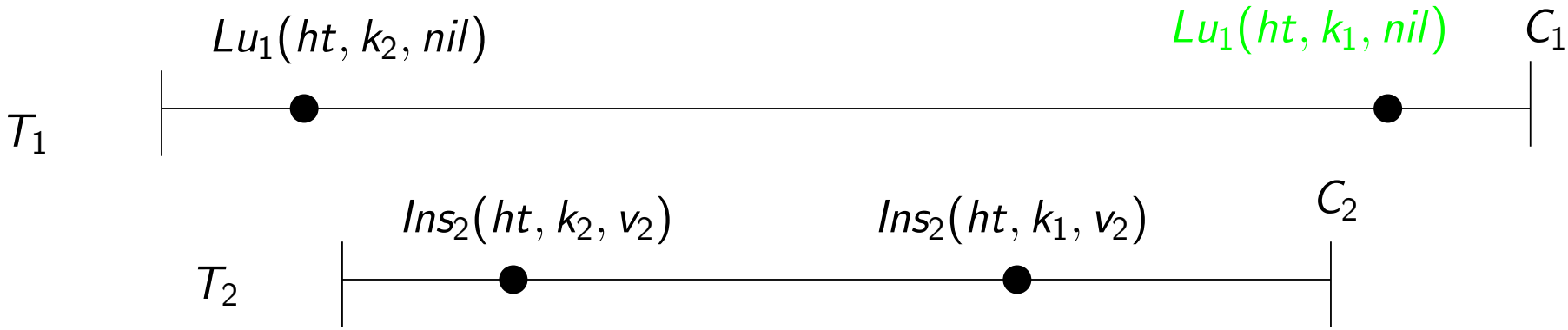


Figure: Multi-version OSTM (MV-OSTM) : (T_1, T_2)

Proposed Algorithm : MV-OSTM

High-level data structure

- A transaction T_i is assigned a unique timestamp i, when invoked.
- Timestamps are monotonically increasing.
- T_i can issue t_lookup(), t_insert(), t_delete() and tryC() methods.

[f] Sathya Peri, Ajay Singh, and Archit Somani, Efficient means of Achieving Composability using Object based Conflicts on Transactional Memory, NETYS'18

Proposed Algorithm : MV-OSTM

High-level data structure

- A transaction T_i is assigned a unique timestamp i, when invoked.
- Timestamps are monotonically increasing.
- T_i can issue t_lookup(), t_insert(), t_delete() and tryC() methods.

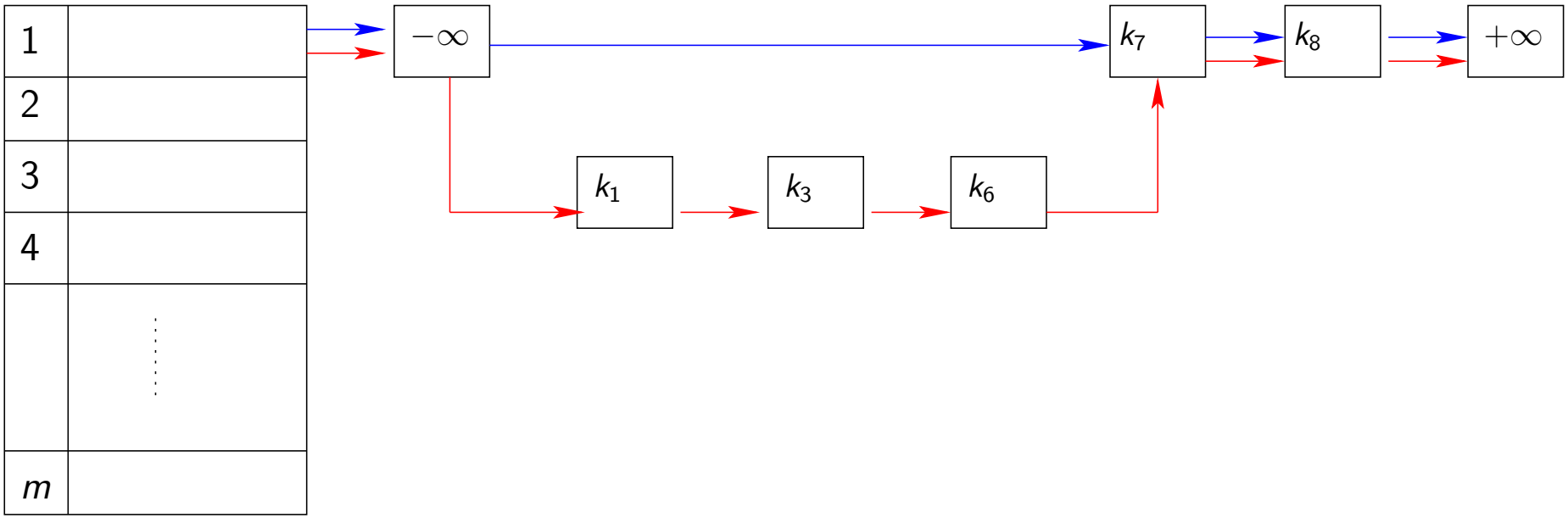


Figure: MV-OSTM design[f]

[f] Sathya Peri, Ajay Singh, and Archit Somani, Efficient means of Achieving Composability using Object based Conflicts on Transactional Memory, NETYS'18

Proposed Algorithm : MV-OSTM

High-level data structure cont'd..

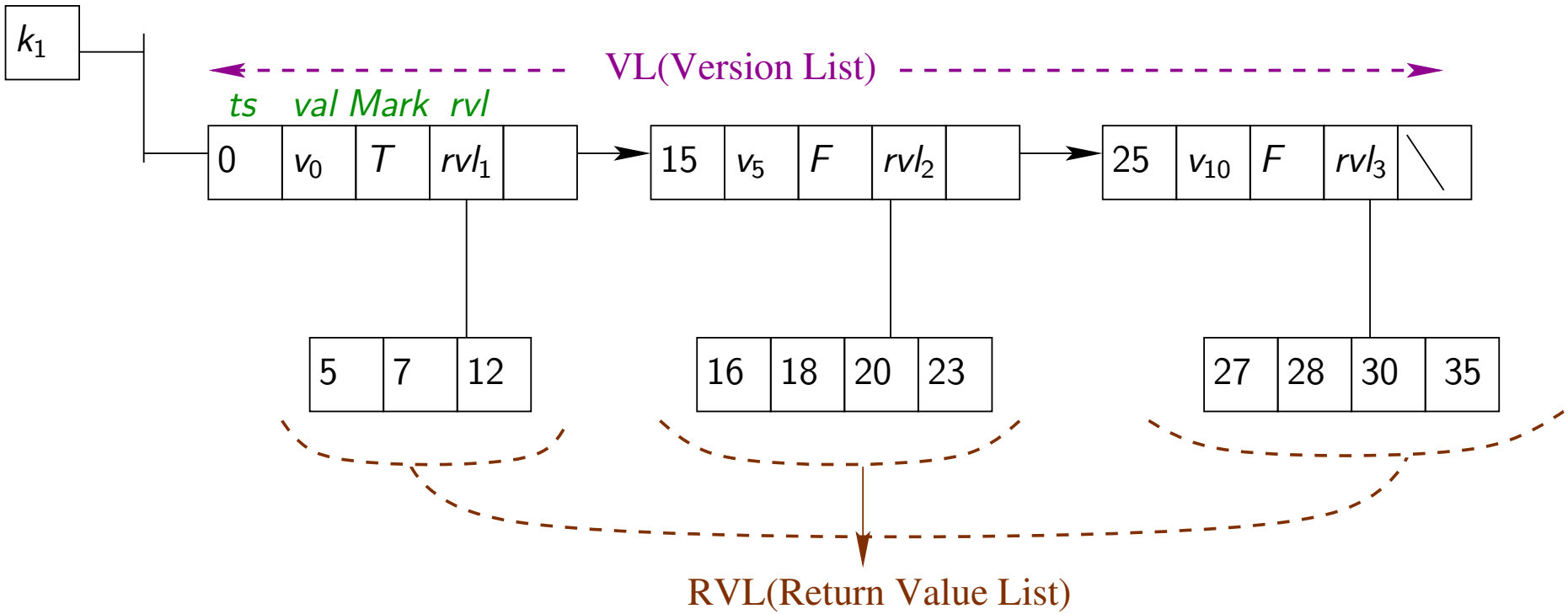


Figure: Data Structure for maintaining versions

Proposed Algorithm : MV-OSTM

t_lookup() method

- **t_lookup() rule:** T_i on invoking $Lu_i(k_1)$ lookups the value v inserted by a transaction T_j that commits before $Lu_i(k_1)$ and j is the largest timestamp $\leq i$.

Proposed Algorithm : MV-OSTM

t_lookup() method

- **t_lookup() rule:** T_i on invoking $Lu_i(k_1)$ lookups the value v inserted by a transaction T_j that commits before $Lu_i(k_1)$ and j is the largest timestamp $\leq i$.

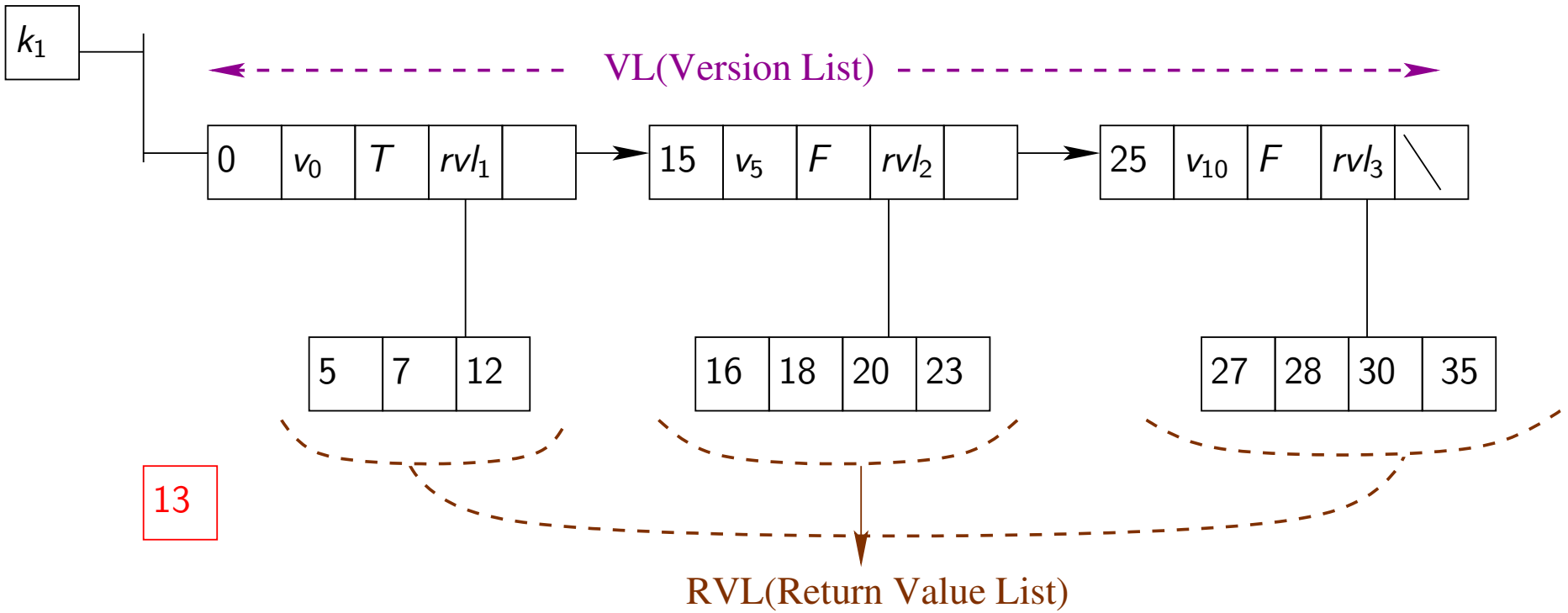


Figure: Lookup on key k_1 by T_{13}

Proposed Algorithm : MV-OSTM

t_lookup() method cont'd..

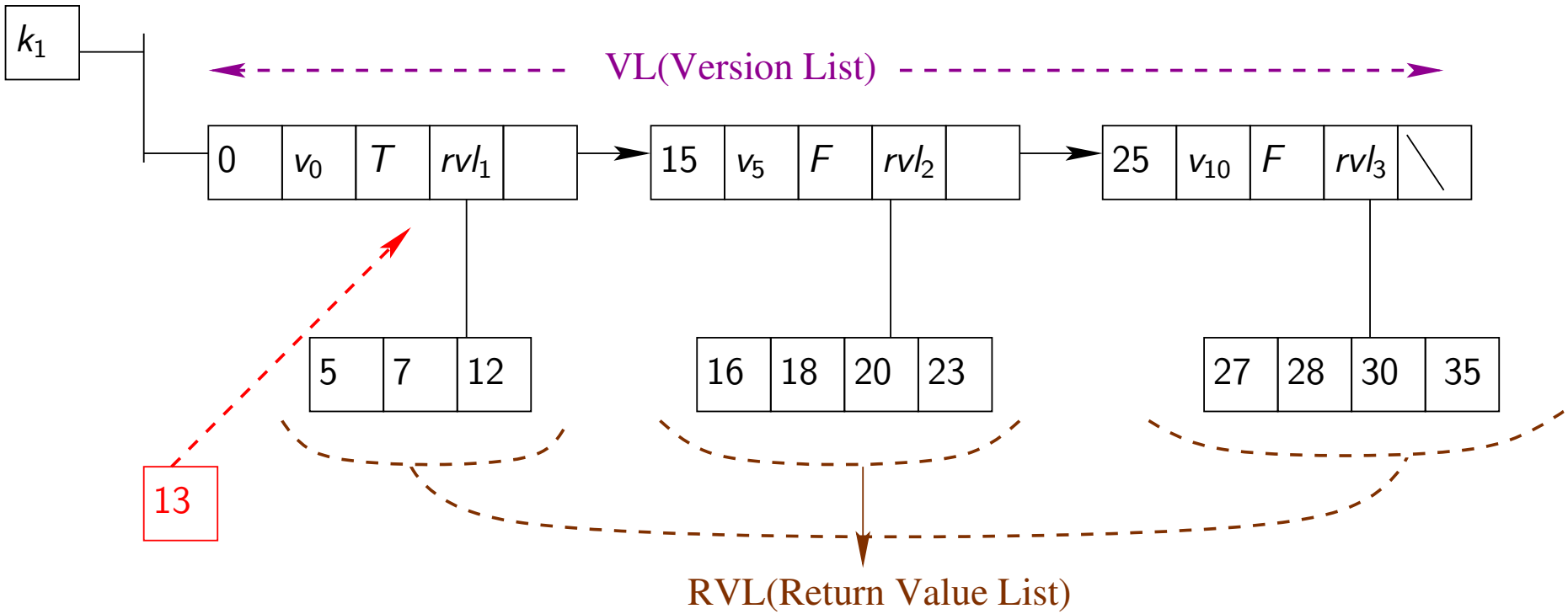


Figure: T_{13} searching appropriate place in version list of k_1

Proposed Algorithm : MV-OSTM

t_lookup() method cont'd..

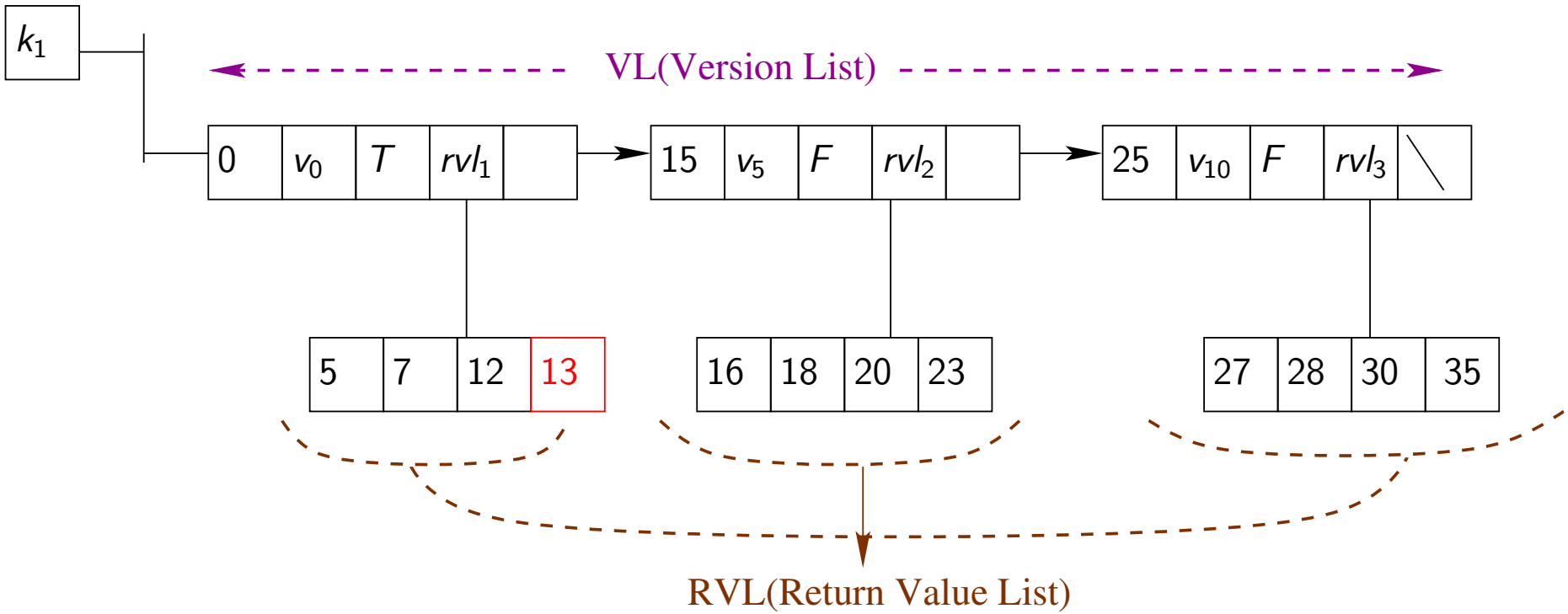


Figure: T_{13} successfully added into rvl_1

Proposed Algorithm : MV-OSTM

t_insert(), t_delete(), and tryC() methods

- **t_insert() and t_delete() rule:** T_i inserts and delete into local memory. The actual effect of both the methods will come in tryC().
- **tryC() rule:** T_i on invoking tryC() operation checks for each key k, in its Insert set and Delete set:
  1. If a transaction T_k has lookups k from T_j and k is committed with $j < i < k$, then $tryC_i()$ returns abort.
  2. otherwise, the transaction T_i is allowed to commit.

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method

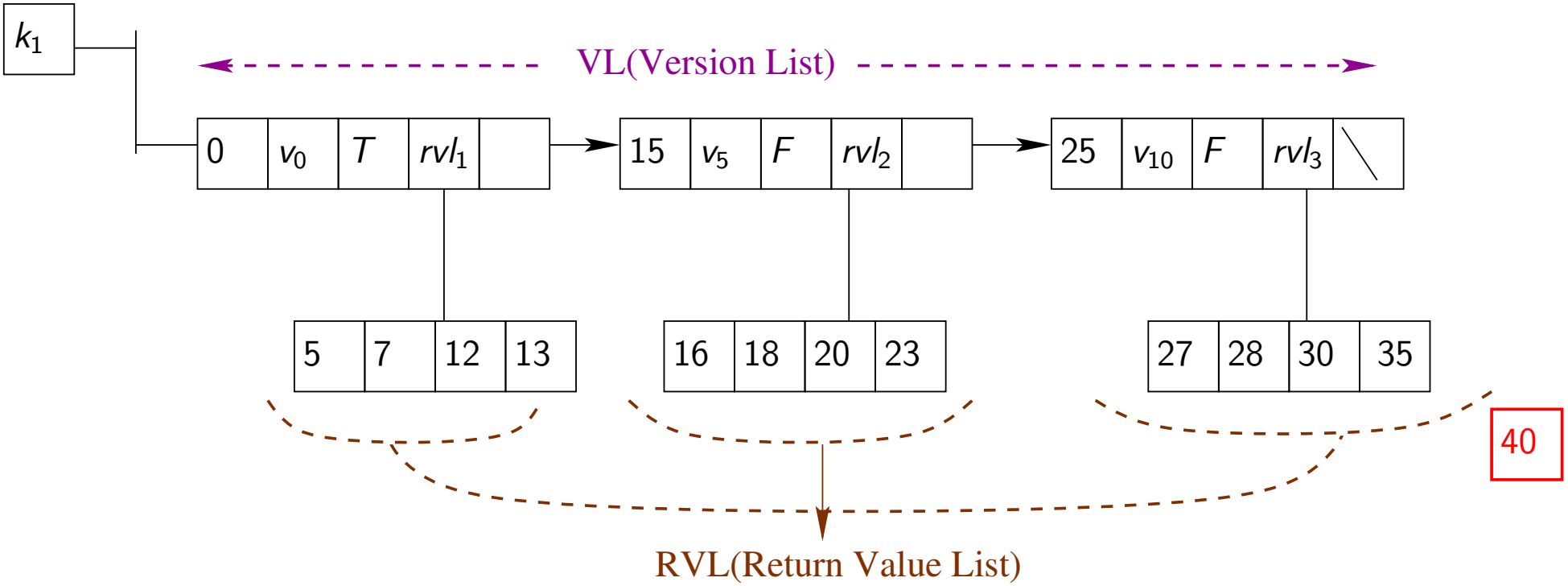


Figure: Insert a version of key k_1 by T_{40}

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method cont'd..

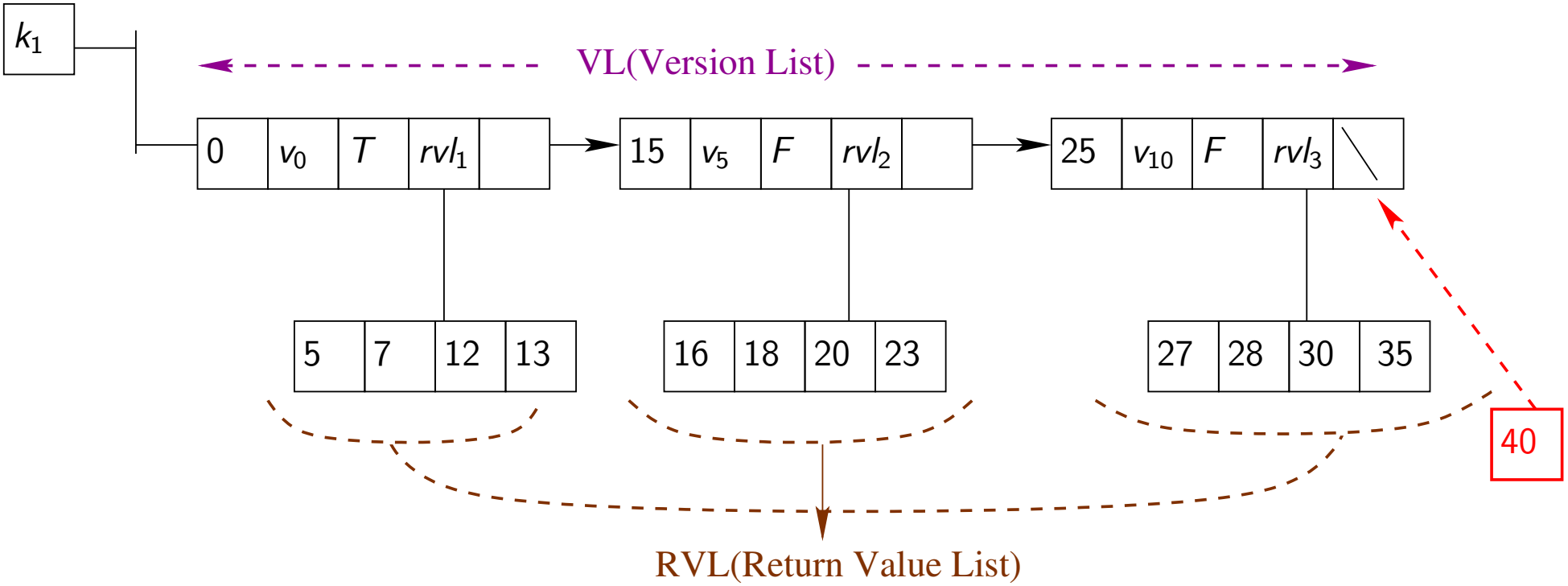


Figure: T_{40} searching appropriate place in version list of k_1

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method cont'd..

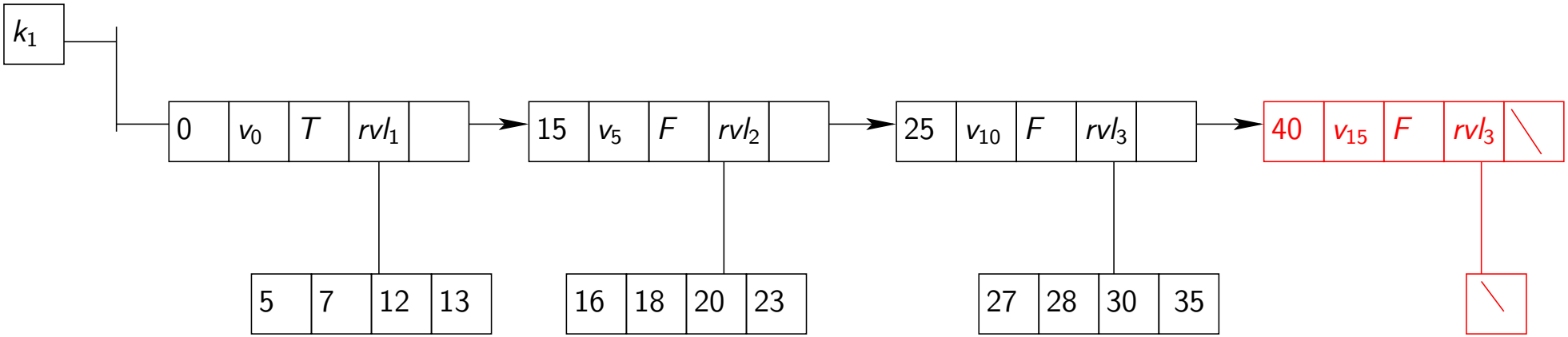


Figure: T_{40} successfully created a new version of k_1

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method cont'd..

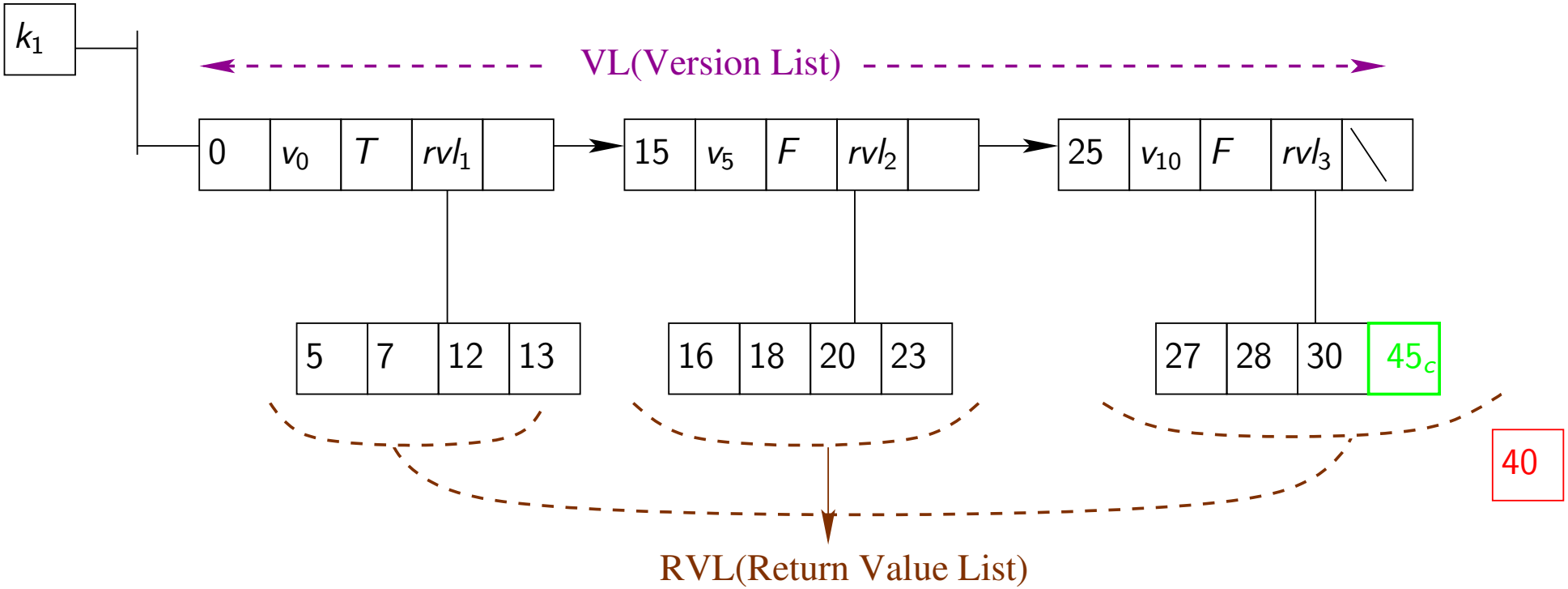


Figure: Insert a version of key k_1 by T_{40}

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method cont'd..

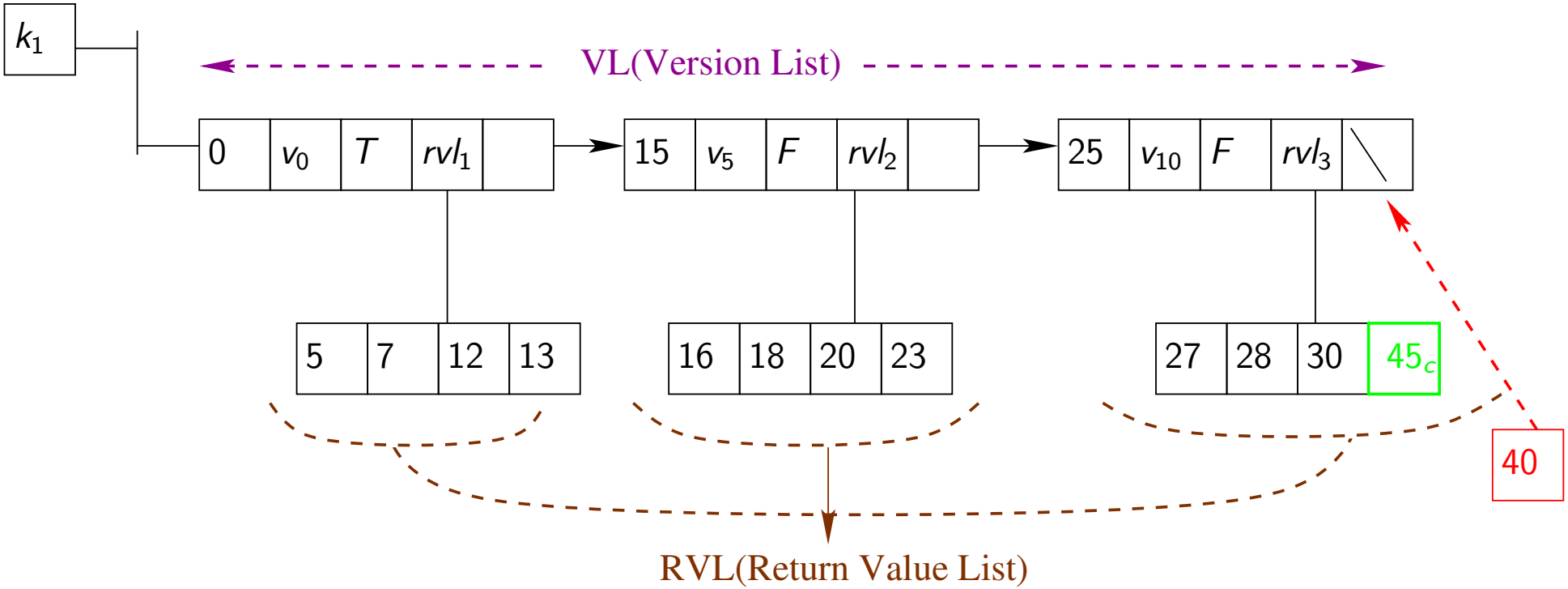


Figure: T_{40} searching appropriate place in version list of k_1

Proposed Algorithm : MV-OSTM

tryC() : t_insert() method cont'd..

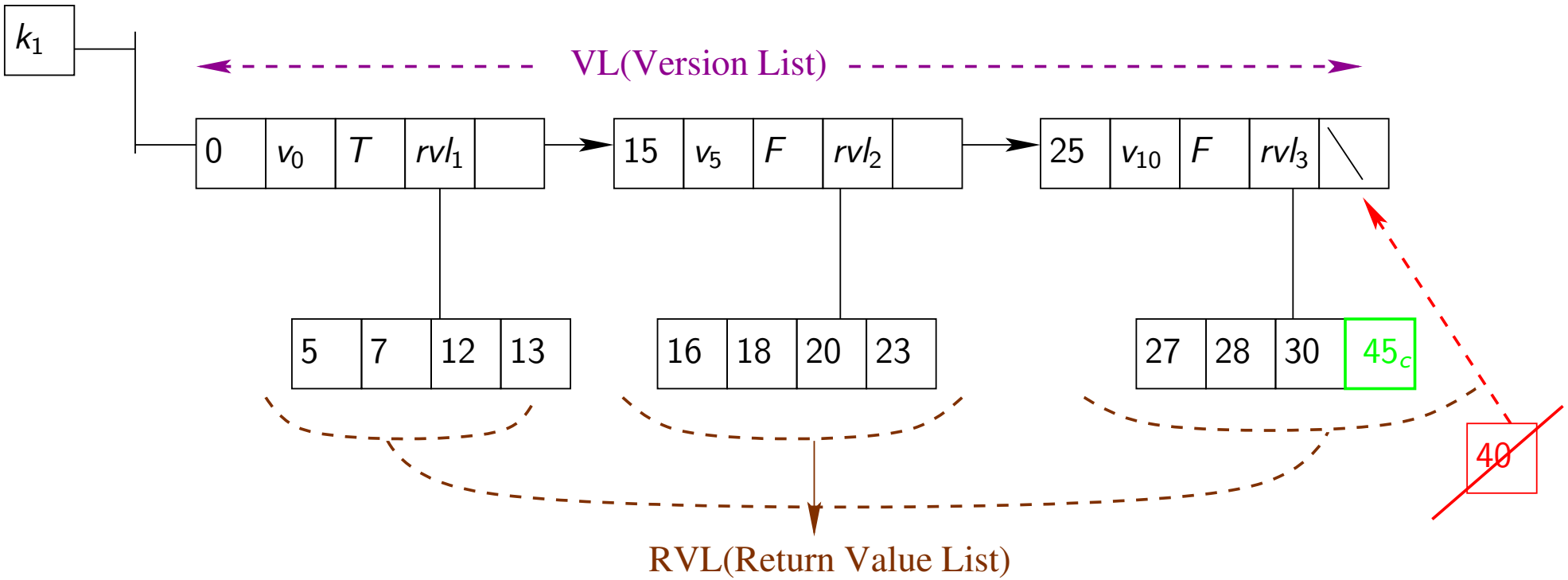


Figure: Abort T_{40} : T_{45} committed before T_{40}

Correctness of MV-OSTM [g]

Theorem (1)

Any history generated by HT-MVOSTM is opaque.

Theorem (2)

HT-MVOSTM with unbounded versions ensures that lookup methods do not return abort.

[g] Arxiv Link: https://arxiv.org/abs/1712.09803

Outline

Experimental Evaluation

Setup

- Intel(R) Xeon(R) CPU, 32 GB RAM, 56 Threads.
- We have compared proposed HT-MVOSTM with HT-OSTM / Elastic STM(ESTM)[a] / Read Write STM (RWSTM) / Multi-Version Time stamp ordering Protocol(HT-MVTO).
- We have compared proposed list-MVOSTM with list-OSTM / Boosted-list / NOrec-list[b] / Trans-list[c] / list-MVTO

---

[a] Felber, P., Gramoli, V., Guerraoui, R.: Elastic Transactions. J. Parallel Distrib. Comput.100(C), 2017

[b] Dalessandro, L., Spear, M.F., Scott, M.L.: NOrec: Streamlining STM by Abolishing Own-ership Records. In Govindarajan, R., Padua, D.A., Hall, M.W., eds.: PPOPP'10

[c] Zhang, D., Dechev, D.: Lock-free Transactions Without Rollbacks for Linked Data Structures. SPAA'16

Experimental Evaluation Cont'd..

- Lookup Intensive Workload: lookup:90%, insert:8% & delete:2%
- Update Intensive Workload: lookup:10%, insert:45% & delete:45%

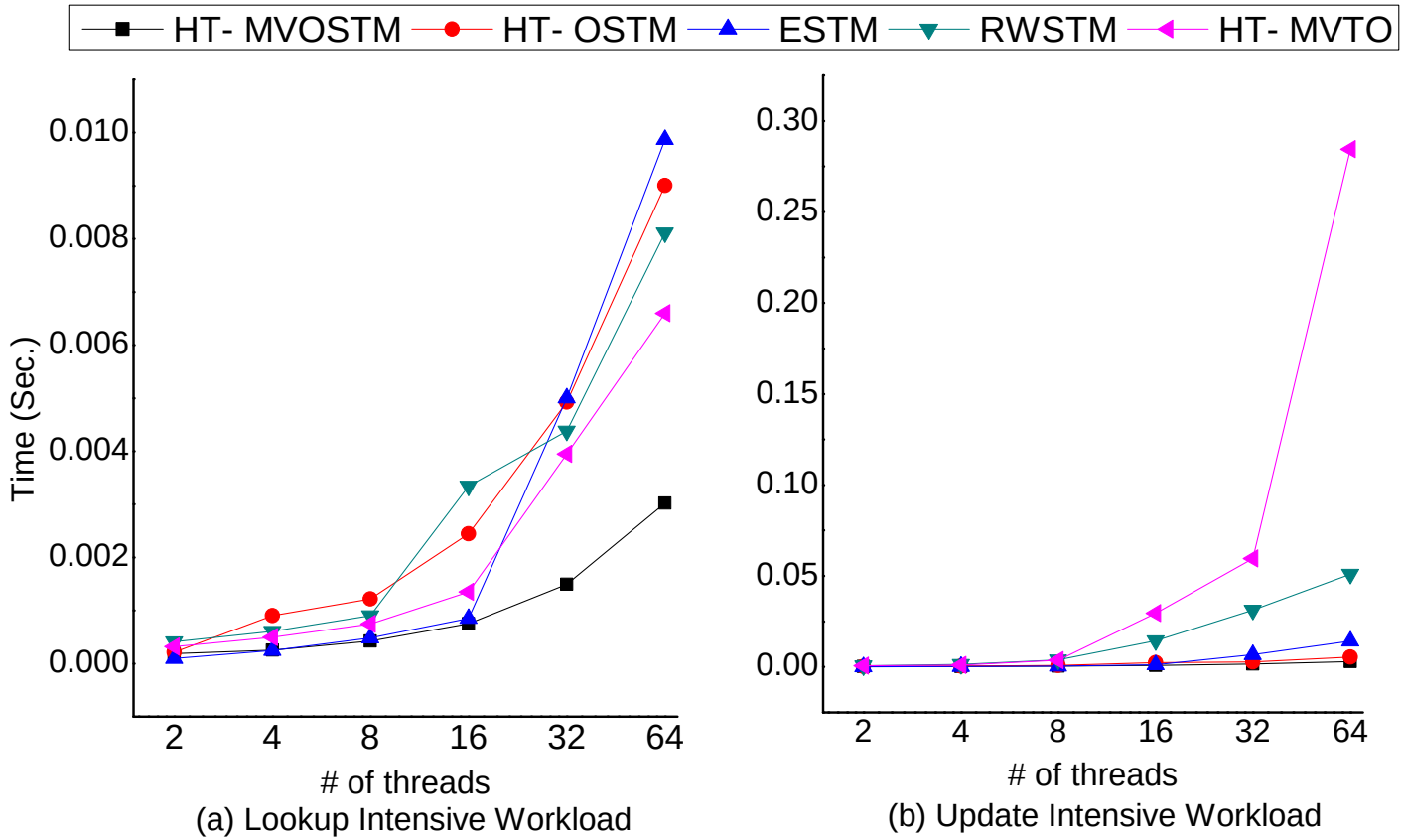


Figure: Performance of HT-MVOSTM

- Proposed HT-MVOSTM gives better performance while improving the concurrency.

Experimental Evaluation Cont'd..

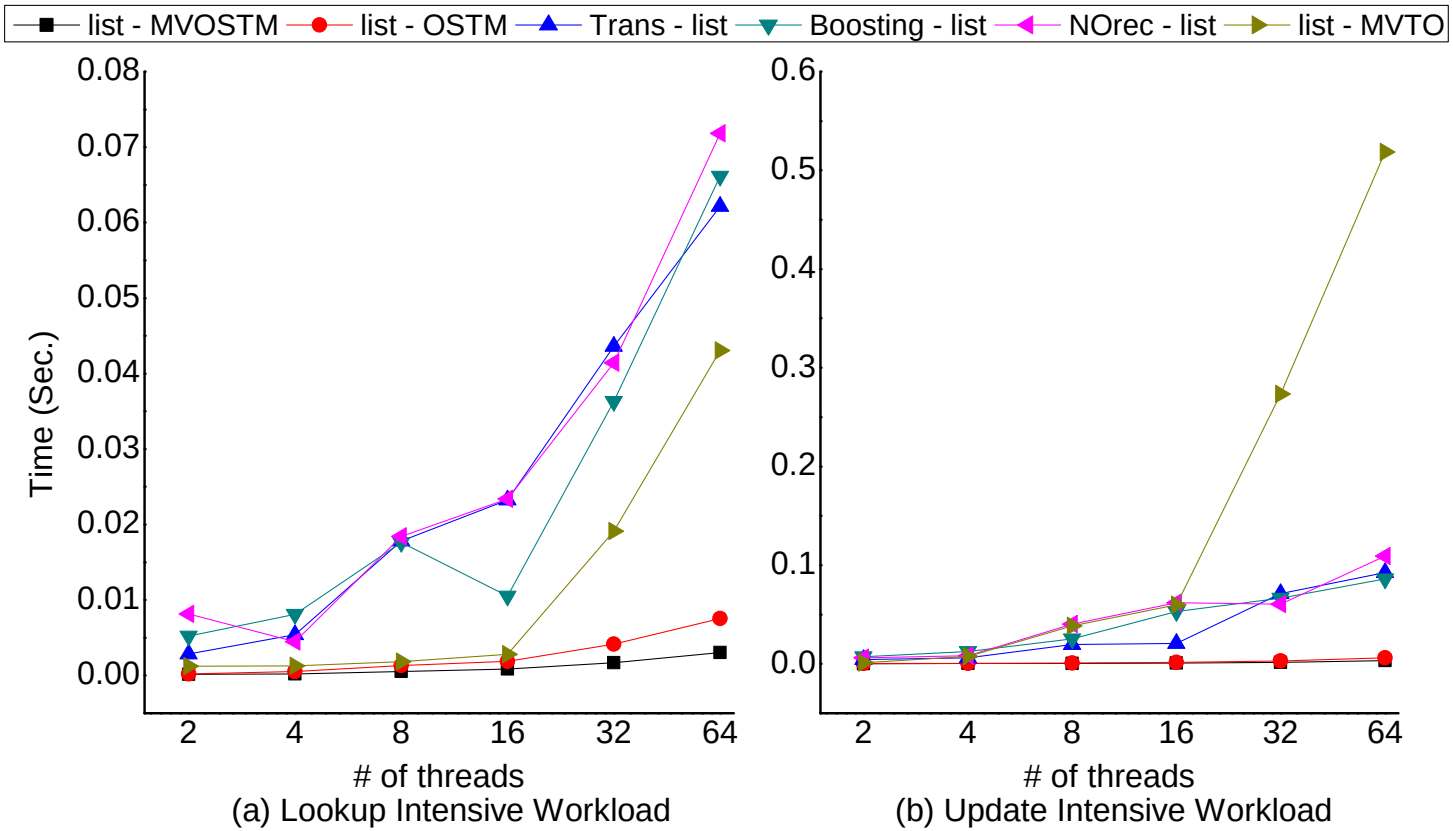


Figure: Performance of list-MVOSTM

- Proposed list-MVOSTM gives better performance while improving the concurrency.

Outline

Conclusion

- We have combined both multi-version and OSTM ideas carefully called as MV-OSTM, for harnessing greater concurrency in STMs.

Conclusion

- We have combined both multi-version and OSTM ideas carefully called as MV-OSTM, for harnessing greater concurrency in STMs.
- HT-MVOSTM shows average speedup of 6.3, 11.2, 4.7, 2.7 times for state of the art HT-MVTO, RWSTM, ESTM & HT-OSTM respectively.

Conclusion

- We have combined both multi-version and OSTM ideas carefully called as MV-OSTM, for harnessing greater concurrency in STMs.
- HT-MVOSTM shows average speedup of 6.3, 11.2, 4.7, 2.7 times for state of the art HT-MVTO, RWSTM, ESTM & HT-OSTM respectively.
- The average speedup achieved by list-MVOSTM from state of the art list-MVTO, NOrec-list, Boosting-list, Trans-list, list-OSTM are 91.5, 29.5, 23, 24, 2.1 respectively.

Outline

- MV-OSTM model can be extended to **Starvation-free multi-version OSTM (SF-MVOSTM)**.

[h] Ni, Yang and Menon, Vijay S. and Adl-Tabatabai, Ali-Reza and Hosking, Antony L. and Hudson, Richard L. and Moss, J. Eliot B. and Saha, Bratin and Shpeisman, Tatiana, Open Nesting in Software Transactional Memory, PPoPP '07

- MV-OSTM model can be extended to **Starvation-free multi-version OSTM (SF-MVOSTM)**.
- MV-OSTM can be enlarged to other data structures like Queue, Stack, Tree etc.

[h] Ni, Yang and Menon, Vijay S. and Adl-Tabatabai, Ali-Reza and Hosking, Antony L. and Hudson, Richard L. and Moss, J. Eliot B. and Saha, Bratin and Shpeisman, Tatiana, Open Nesting in Software Transactional Memory, PPoPP '07

- MV-OSTM model can be extended to **Starvation-free multi-version OSTM (SF-MVOSTM)**.
- MV-OSTM can be enlarged to other data structures like Queue, Stack, Tree etc.
- An interesting aspect is exploring **Nesting**[h] for MV-OSTM in which one object-based transaction invokes other read-write transaction.

[h] Ni, Yang and Menon, Vijay S. and Adl-Tabatabai, Ali-Reza and Hosking, Antony L. and Hudson, Richard L. and Moss, J. Eliot B. and Saha, Bratin and Shpeisman, Tatiana, Open Nesting in Software Transactional Memory, PPoPP '07


Arigato Gozaimasu!
(c) ClipArtIllustration.com