

Supporting Parallelism in Operating Systems & Programming Languages

Transactional Memory

Kornilios Kourtis
<kornilios.kourtis@inf.ethz.ch>

Introduction

TM System example: Harris and Fraser 2003

TM Design options

Final Remarks

Locks are problematic

- ▶ complicated when using fine-grained locking
- ▶ locks do not compose
- ▶ pessimistic

- ▶ multicore dominance calls for synchronization interfaces that are programmer-friendly
 - ▶ more productive
 - ▶ easier
 - ▶ less error-prone

- ▶ → Transactional Memory

Transactional Memory

- ▶ inspired by Databases / Distributed Systems
- ▶ Atomicity
- ▶ Consistency
- ▶ Isolation
- ▶ Durability

TM Interface

```
atomic {  
    ...  
}
```

```
do {  
    TxBegin();  
    ...  
} while (!TxCommit())
```

Other issues:

- ▶ wait conditions
- ▶ exceptions
- ▶ nesting (flattened, closed, open)
- ▶ mixed-mode access (weak vs strong)
- ▶ ...

Introduction

TM System example: Harris and Fraser 2003

TM Design options

Final Remarks

Compare-and-Swap (CAS)

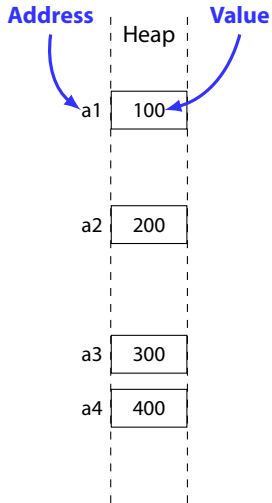
- ▶ powerful atomic operation
- ▶ intel: `cmpxchg`

```
CAS(addr, oldval, newval):  
    ret = *addr;  
    if (ret == oldval)  
        *addr = newval;  
    return ret;
```

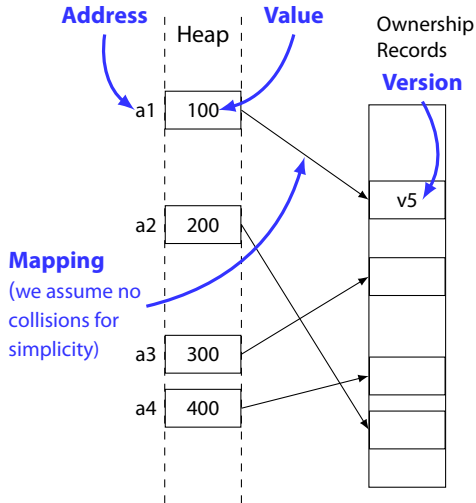
Harris and Fraser 2003: Operations

- ▶ STMStart()
 - ▶ STMAbort()
 - ▶ STMCommit()
 - ▶ STMValidate()
 - ▶ STMWait()
- ▶ STMRead()
 - ▶ STMWrite()

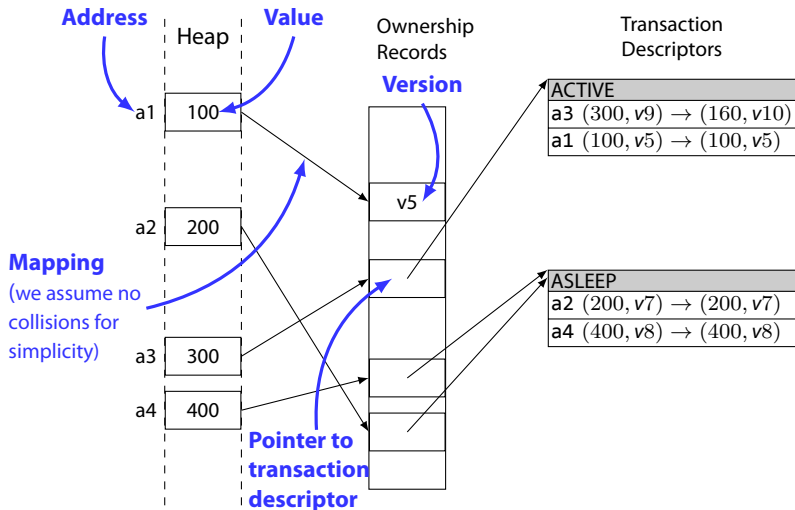
Harris and Fraser 2003: Data structures



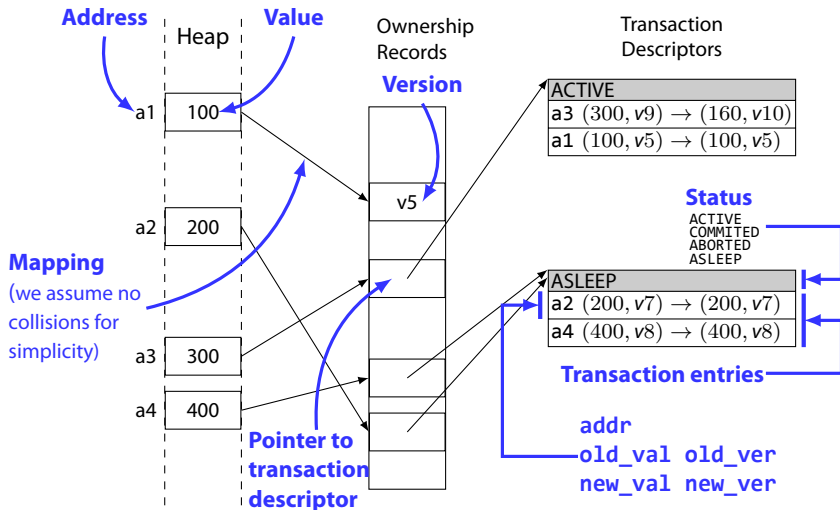
Harris and Fraser 2003: Data structures



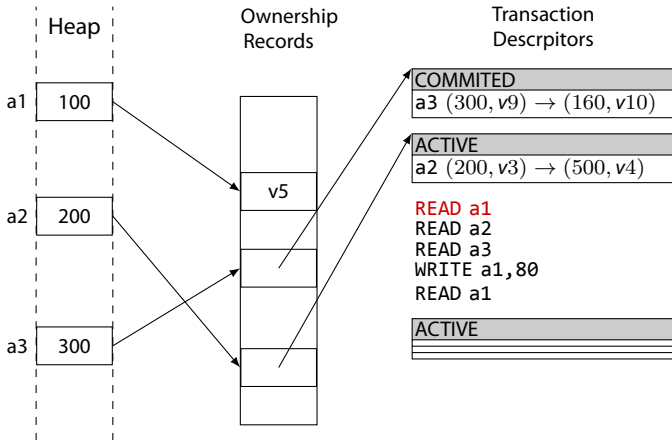
Harris and Fraser 2003: Data structures



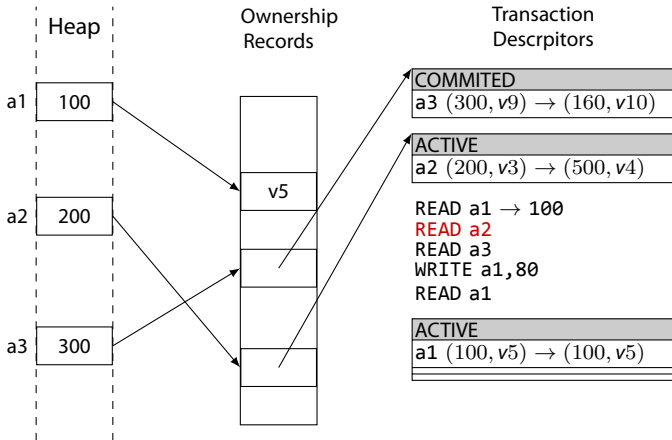
Harris and Fraser 2003: Data structures



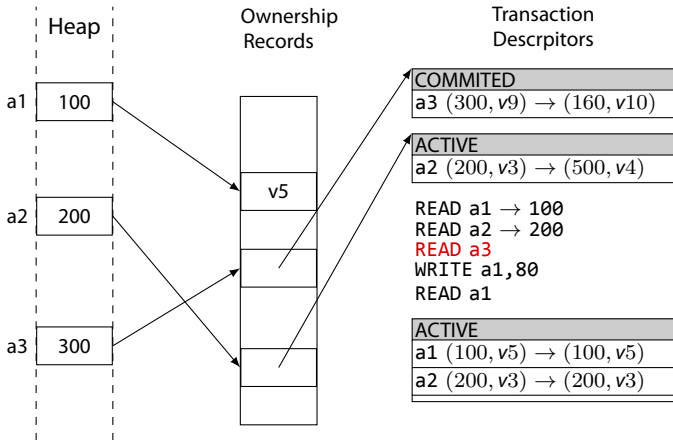
Harris and Fraser 2003: Read and Write



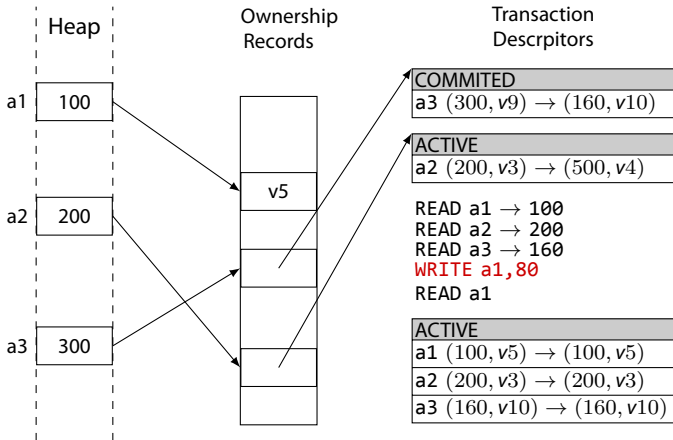
Harris and Fraser 2003: Read and Write



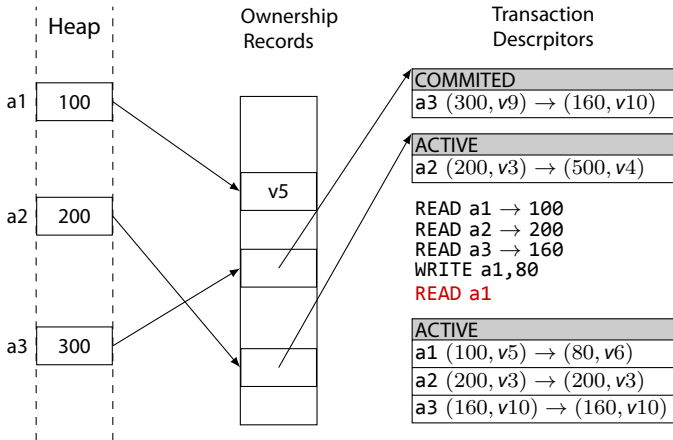
Harris and Fraser 2003: Read and Write



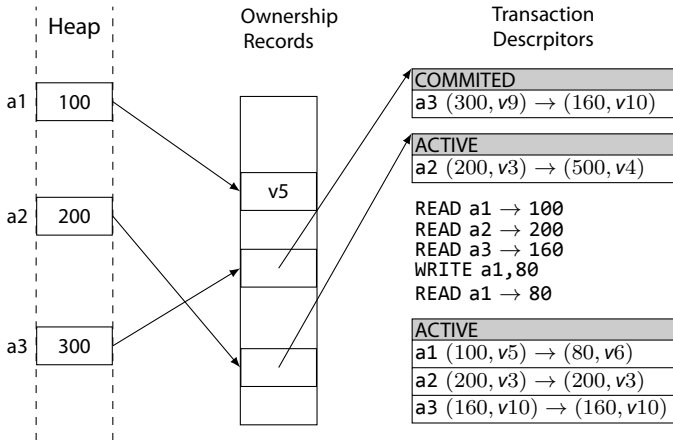
Harris and Fraser 2003: Read and Write



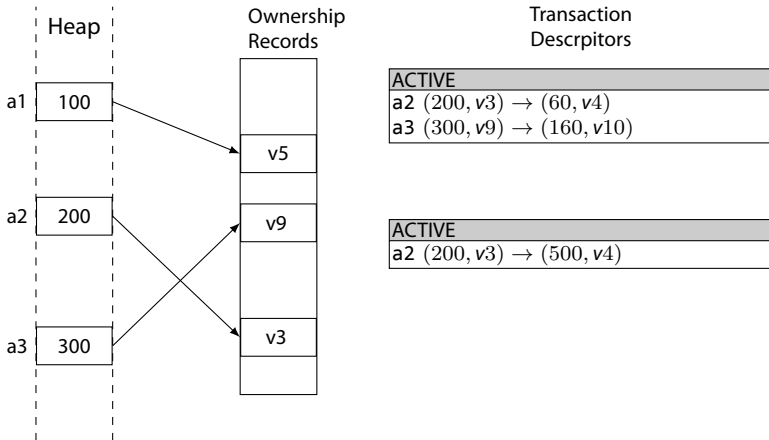
Harris and Fraser 2003: Read and Write



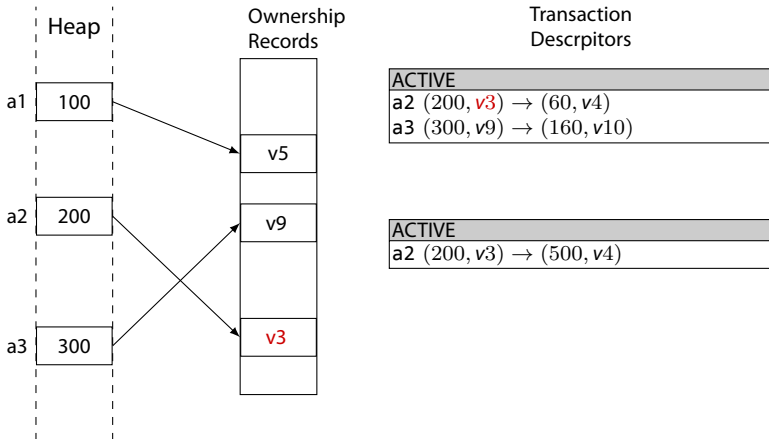
Harris and Fraser 2003: Read and Write



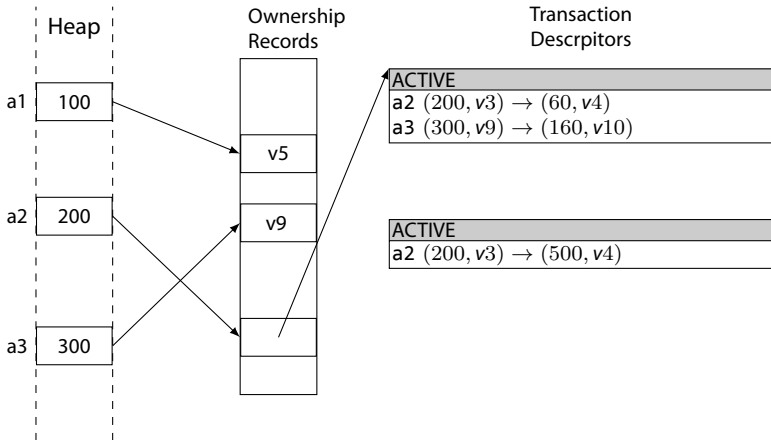
Harris and Fraser 2003: Commit



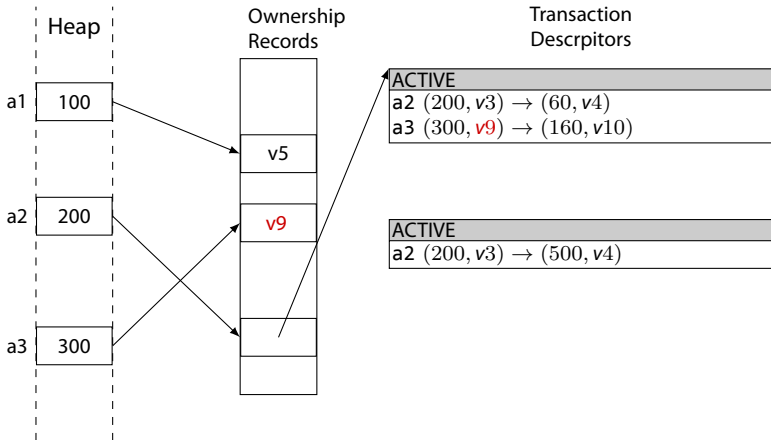
Harris and Fraser 2003: Commit



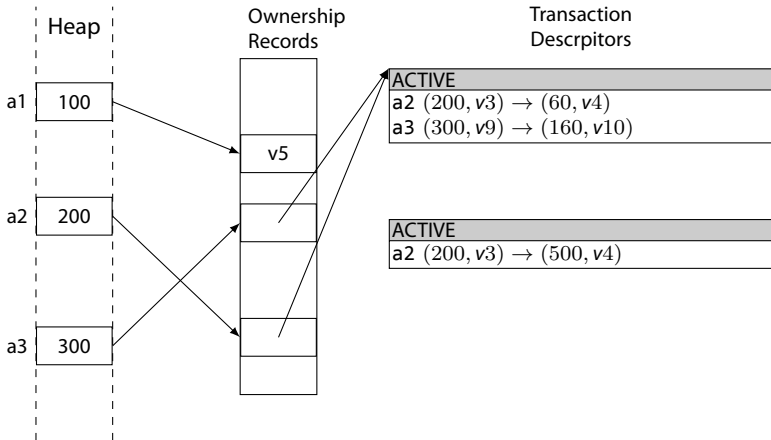
Harris and Fraser 2003: Commit



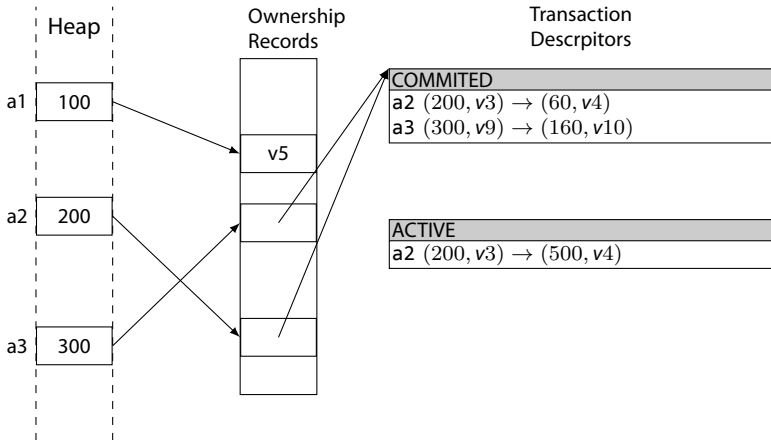
Harris and Fraser 2003: Commit



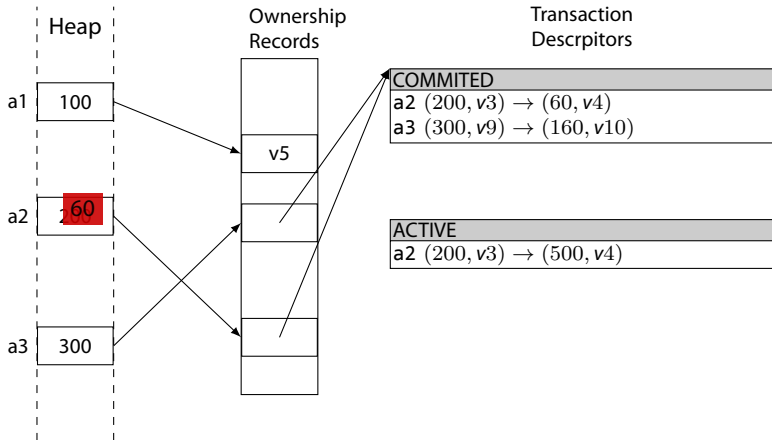
Harris and Fraser 2003: Commit



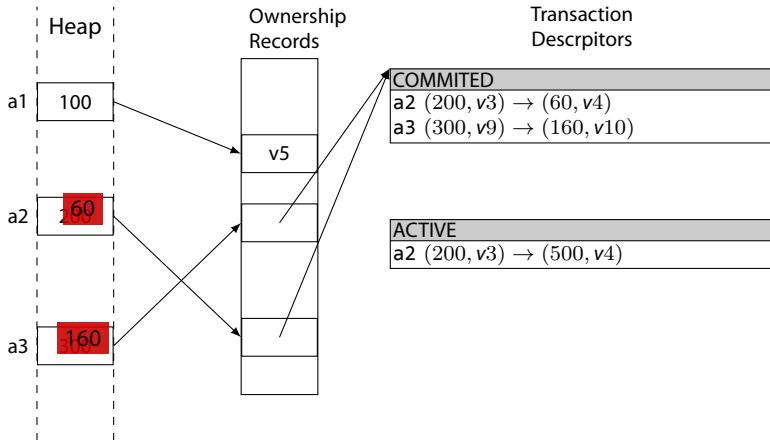
Harris and Fraser 2003: Commit



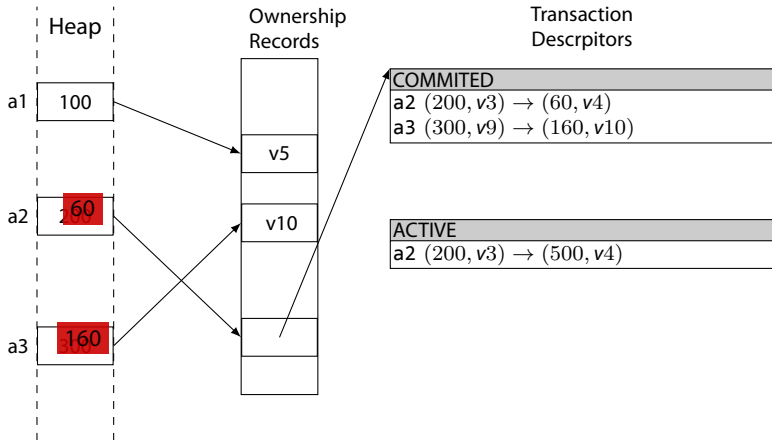
Harris and Fraser 2003: Commit



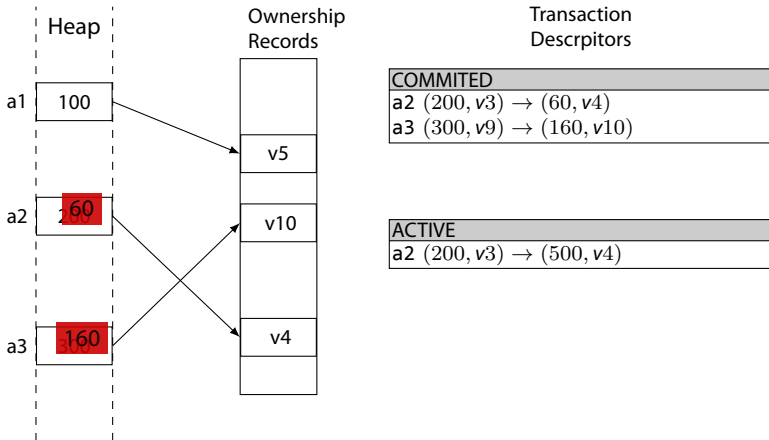
Harris and Fraser 2003: Commit



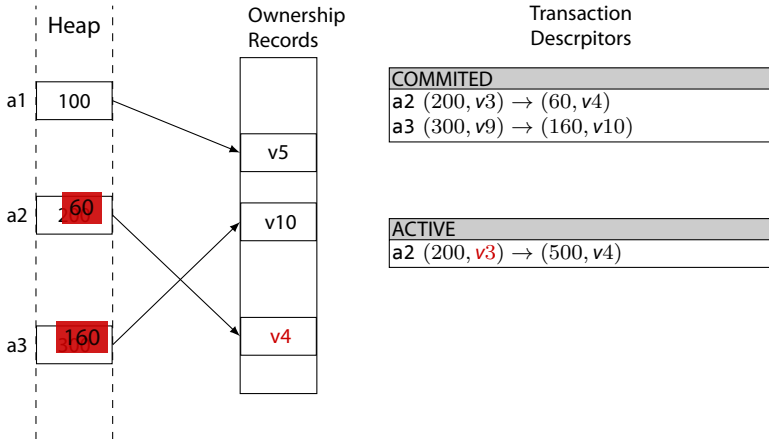
Harris and Fraser 2003: Commit



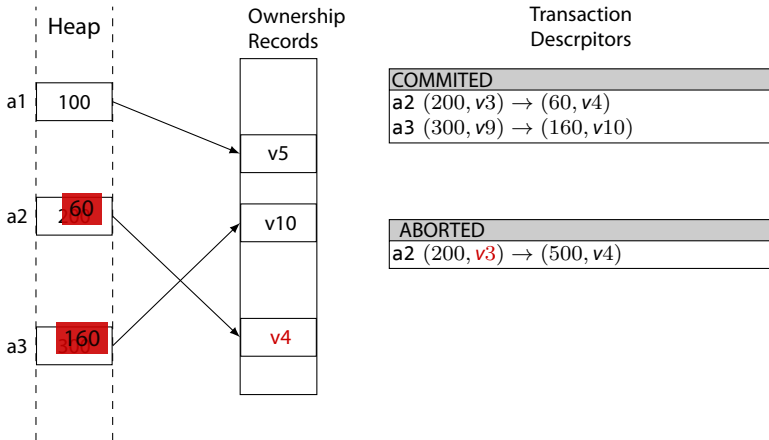
Harris and Fraser 2003: Commit



Harris and Fraser 2003: Commit



Harris and Fraser 2003: Commit



Introduction

TM System example: Harris and Fraser 2003

TM Design options

Final Remarks

TM Design choices

Version Management: When data are written to memory?

- ▶ eager (undo-log)
- ▶ lazy (redo-log / write buffering)

Concurrency Control: When a conflict is detected/resolved?

- ▶ pessimistic
- ▶ optimistic

Conflict detection granularity:

- ▶ object
- ▶ word
- ▶ cacheline

Version Management

- ▶ Eager
 - ▶ update memory location when update happens
 - ▶ maintain an undo-log
 - ▶ if commit fails → apply undo-log

- ▶ Lazy
 - ▶ keep updates in a write buffer (redo-log)
 - ▶ if commit succeeds → apply redo-log

Version Management

- ▶ Eager
 - ▶ update memory location when update happens
 - ▶ maintain an undo-log
 - ▶ if commit fails → apply undo-log
 - need to protect memory location until transaction ends
- ▶ Lazy
 - ▶ keep updates in a write buffer (redo-log)
 - ▶ if commit succeeds → apply redo-log
 - need to check write buffer for reads (RAW dependency)
 - slow commit

McRT-STM

Version Management:

- ▶ eager (undo-log)

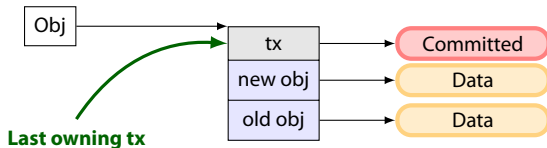
Concurrency control:

- ▶ Readers:
 - ▶ check if memory location is locked by a writer
 - ▶ read value and log version
 - ▶ validate versions on commit time (*optimistic*)
- ▶ Writers:
 - ▶ lock memory location before doing update
 - ▶ at commit time, updates version, releases lock

Conflict detection granularity

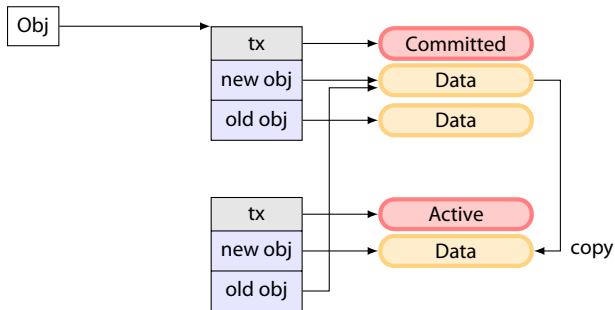
- ▶ cache-line
 - ▶ mainly used in HTM
- ▶ address based (word)
 - ▶ mapped addresses to fixed-size metadata table
- ▶ object
 - ▶ metadata can be added to the object header
 - ▶ access to different fields can cause conflicts
 - ▶ (more) transparent to programmer
 - ▶ difficult for languages like C
 - ▶ TM coupled with language RTS facilities (e.g., GC)
 - ▶ language facilities can be used for optimization (e.g., compiler, annotations, ...)

Herlihy et al. 2003



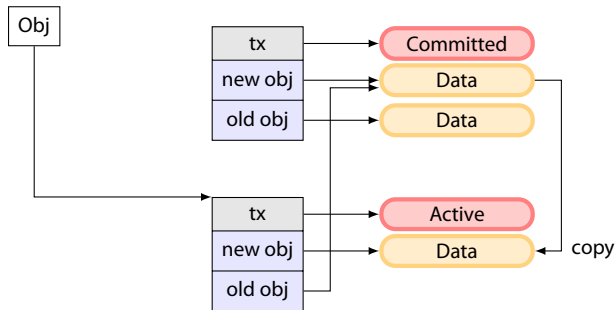
- ▶ `current = (tx.state == COMMITTED) ? new:old`

Herlihy et al. 2003



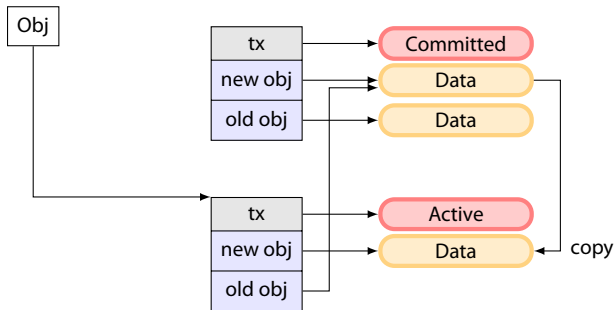
- ▶ `current = (tx.state == COMMITTED) ? new:old`

Herlihy et al. 2003



- ▶ `current = (tx.state == COMMITTED) ? new:old`

Herlihy et al. 2003



- ▶ `current = (tx.state == COMMITTED) ? new:old`
- ▶ a transaction can update all its objects atomically:
`tx.state = COMMITTED`
- ▶ GC

Introduction

TM System example: Harris and Fraser 2003

TM Design options

Final Remarks

Examples of TM in languages

- ▶ Haskell (TVars)
- ▶ Clojure (persistent data-structures)
- ▶ C/C++ (draft standard¹, gcc)
- ▶ ...

¹<http://software.intel.com/file/21569>

Hardware TM

- ▶ SunOracle's Rock
- ▶ Intel Transaction Synchronization eXtensions
 - ▶ Hardware Lock Ellision (HLE)
 - ▶ XACQUIRE
 - ▶ XRELEASE
 - ▶ Restricted Transactional Memory (RTM)
 - ▶ XBEGIN
 - ▶ XEND
 - ▶ XABORT
- ▶ Hybrid TM

Issues with TM

- ▶ I/O
 - ▶ cannot rollback (at least trivially)
- ▶ practicality remains to be seen

Bibliography

- ▶ Tim Harris and Keir Fraser
Language support for lightweight transactions
OOPSLA '03
- ▶ Maurice Herlihy et al.
Software transactional memory for dynamic-sized data structures
PODC '03
- ▶ Bratin Saha et al.
McRT-STM: a high performance software transactional memory system for a multi-core runtime
PPoPP '06
- ▶ Dan Grossman
The Transactional Memory / Garbage Collection Analogy
OOPSLA '07