Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

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
- $\rightarrow$ 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;
```

CAS Example

```
obj_t *max = NULL;
void find_max(void) {
    for (;;) {
        obj_t *o1 = malloc(sizeof(*o1));
        compute(o1); // fill in o1
retry:
        obj_t *o2 = max;
        if (o2 && o2->val >= o1->val) {
            free(o1);
            continue; }
        if (CAS(&max, o2, o1) != o1)
            goto retry; // max changed
        else
            free(o2); // max replaced }}
```

CAS Example (ABA problem)

```
obj_t *max = NULL;
void find_max(void) {
    for (;;) {
        obj_t *o1 = malloc(sizeof(*o1));
        compute(o1); // fill in o1
retry:
        obj_t *o2 = max;
        if (o2 && o2->val >= o1->val) {
            free(o1);
            continue; }
        if (CAS(&max, o2, o1) != o1)
            goto retry; // max changed
        else
            free(o2); // max replaced }}
```

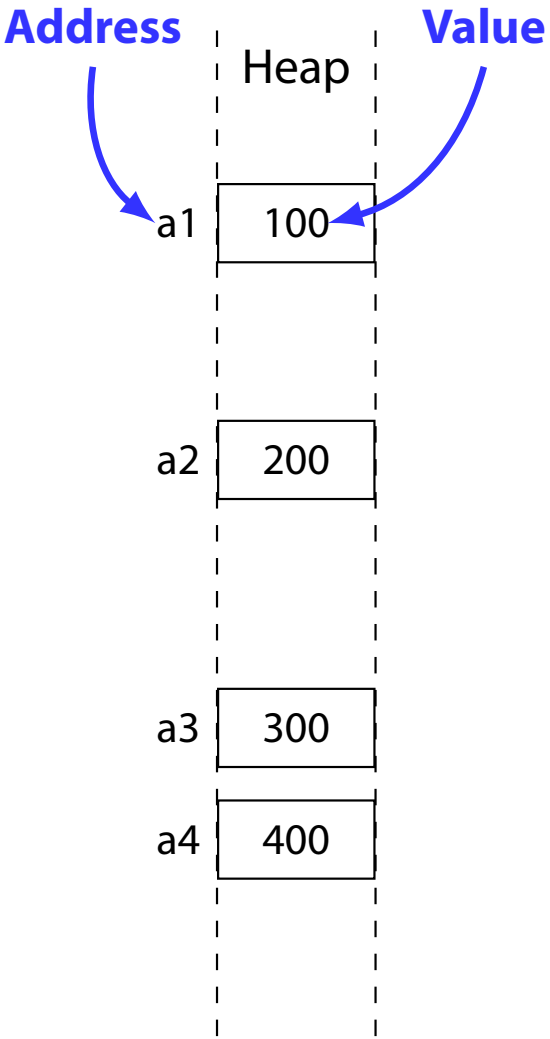
Harris and Fraser 2003

(Language Support for Lightweight Transactions)

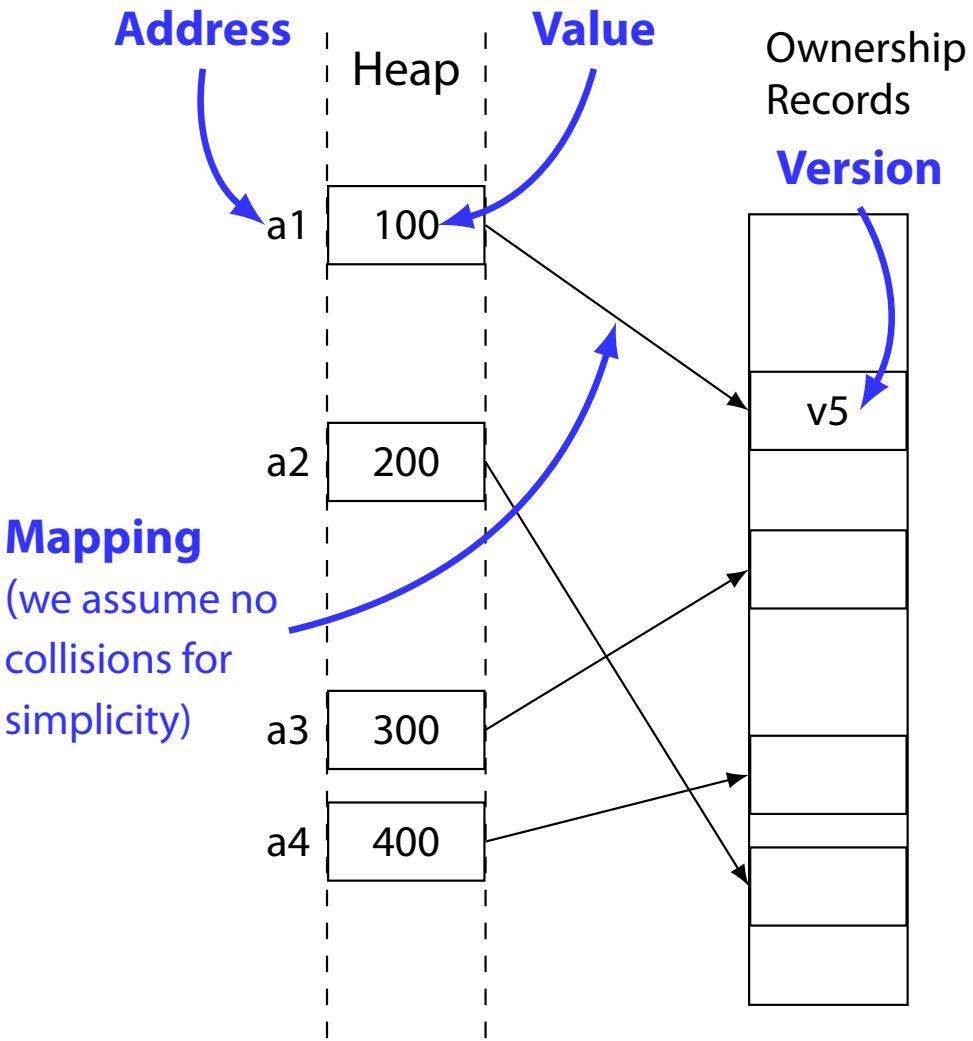
- `void STMStart()`
- `void STMAborT()`
- `bool STMCommit()`
- `bool STMValidate()`
- `void STMWait()`

- `word STMRead(addr a)`
- `void STMWrite(addr a, word w)`

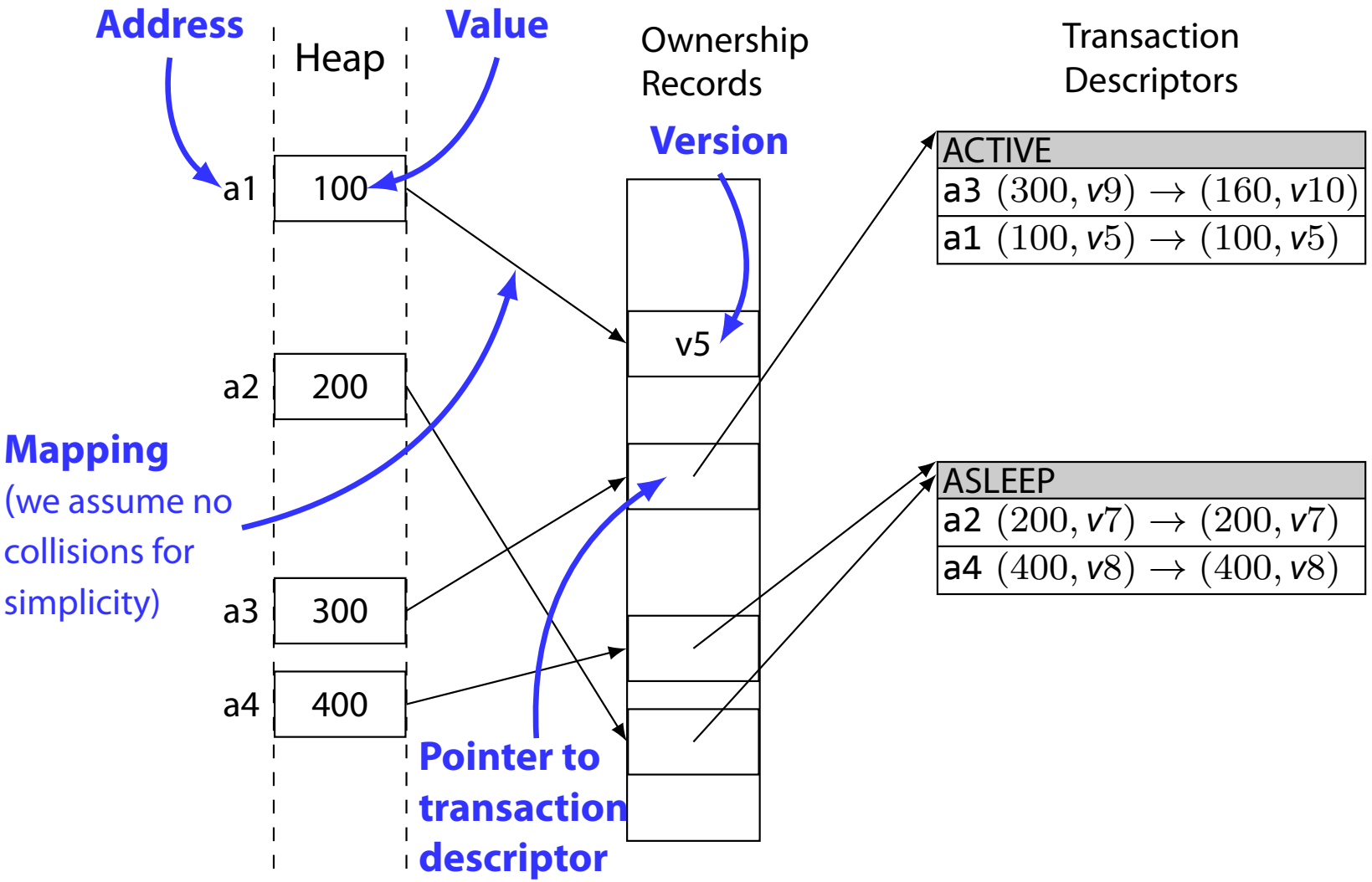
Harris and Fraser 2003: Data structures



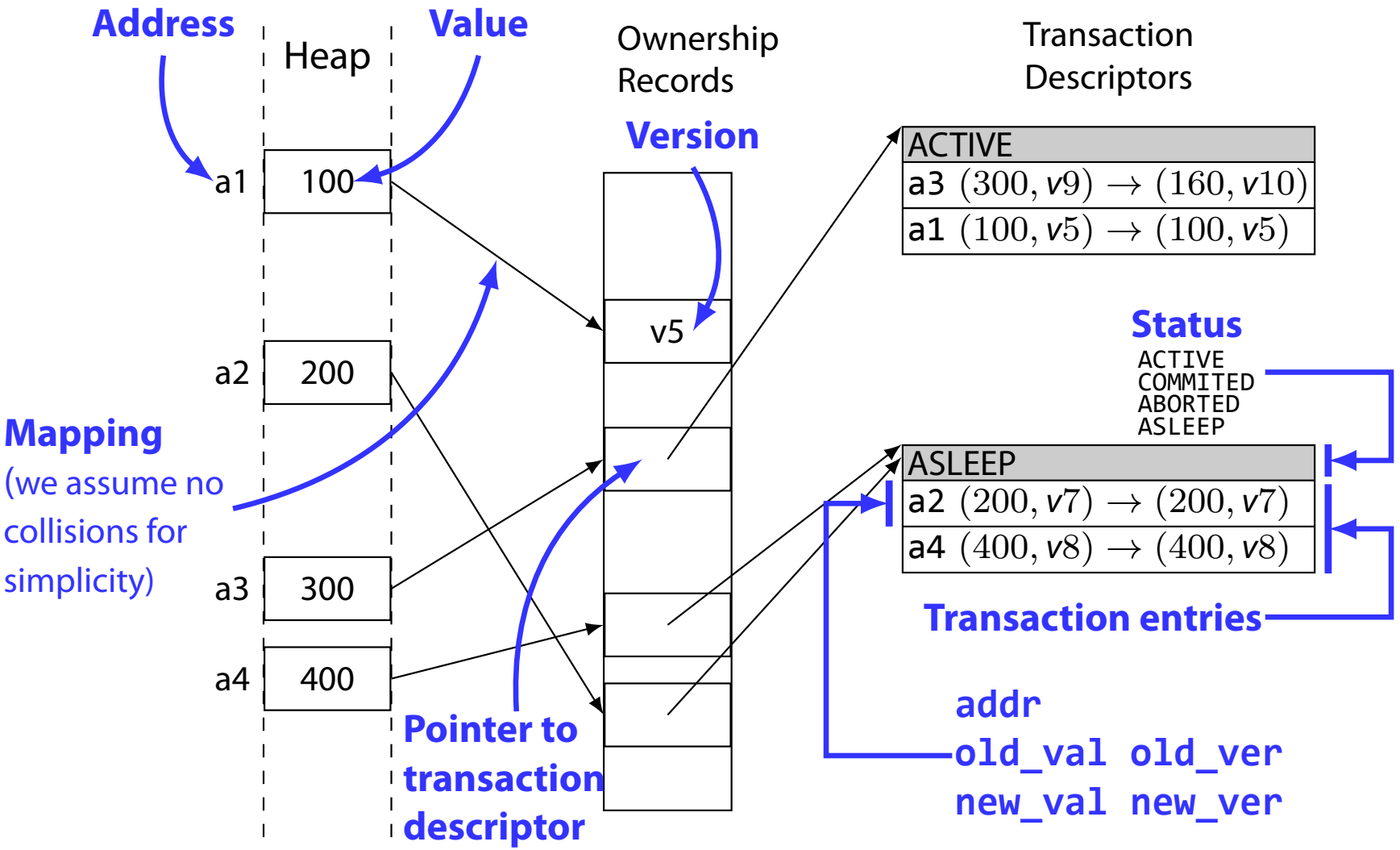
Harris and Fraser 2003: Data structures



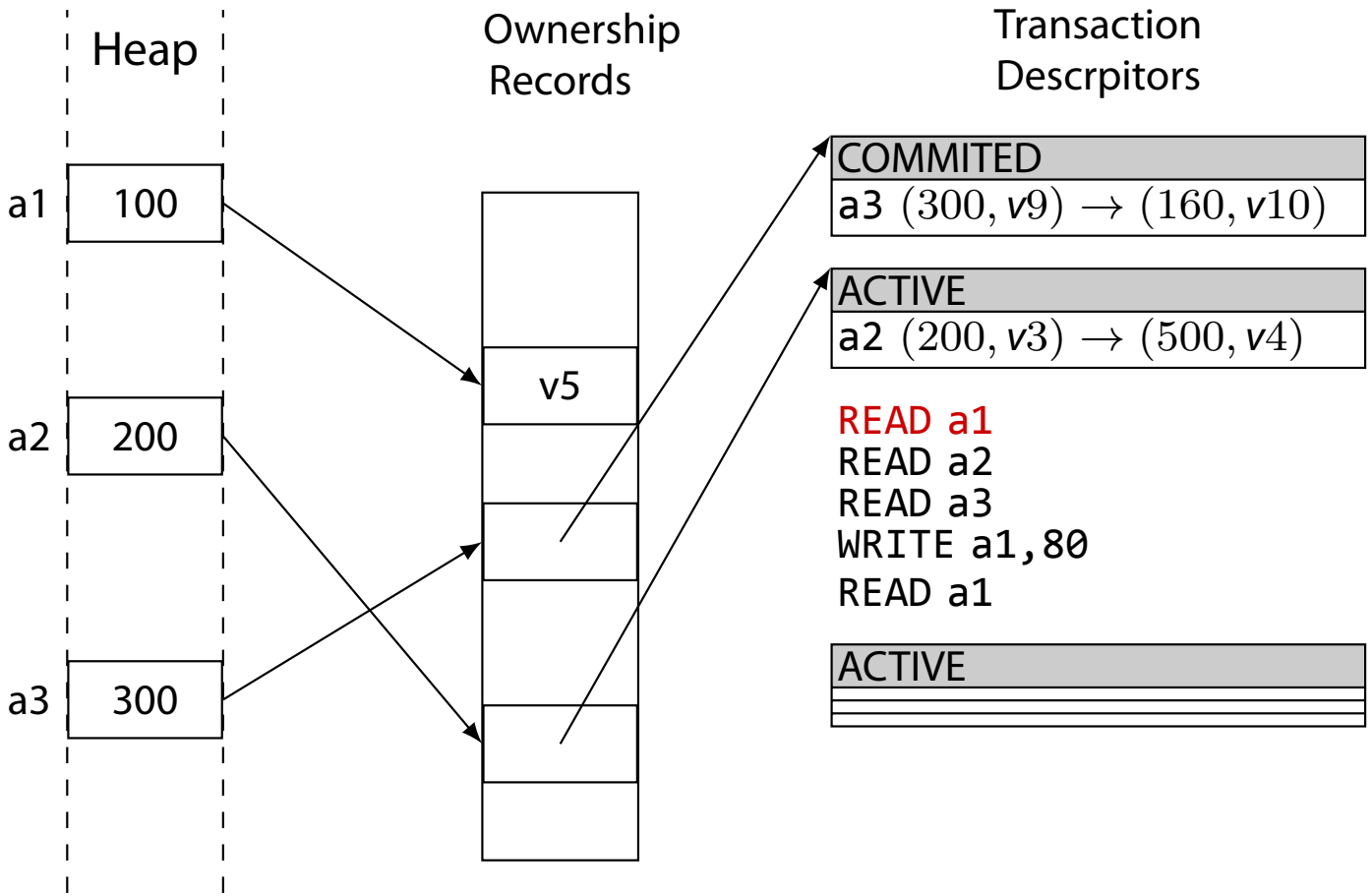
Harris and Fraser 2003: Data structures



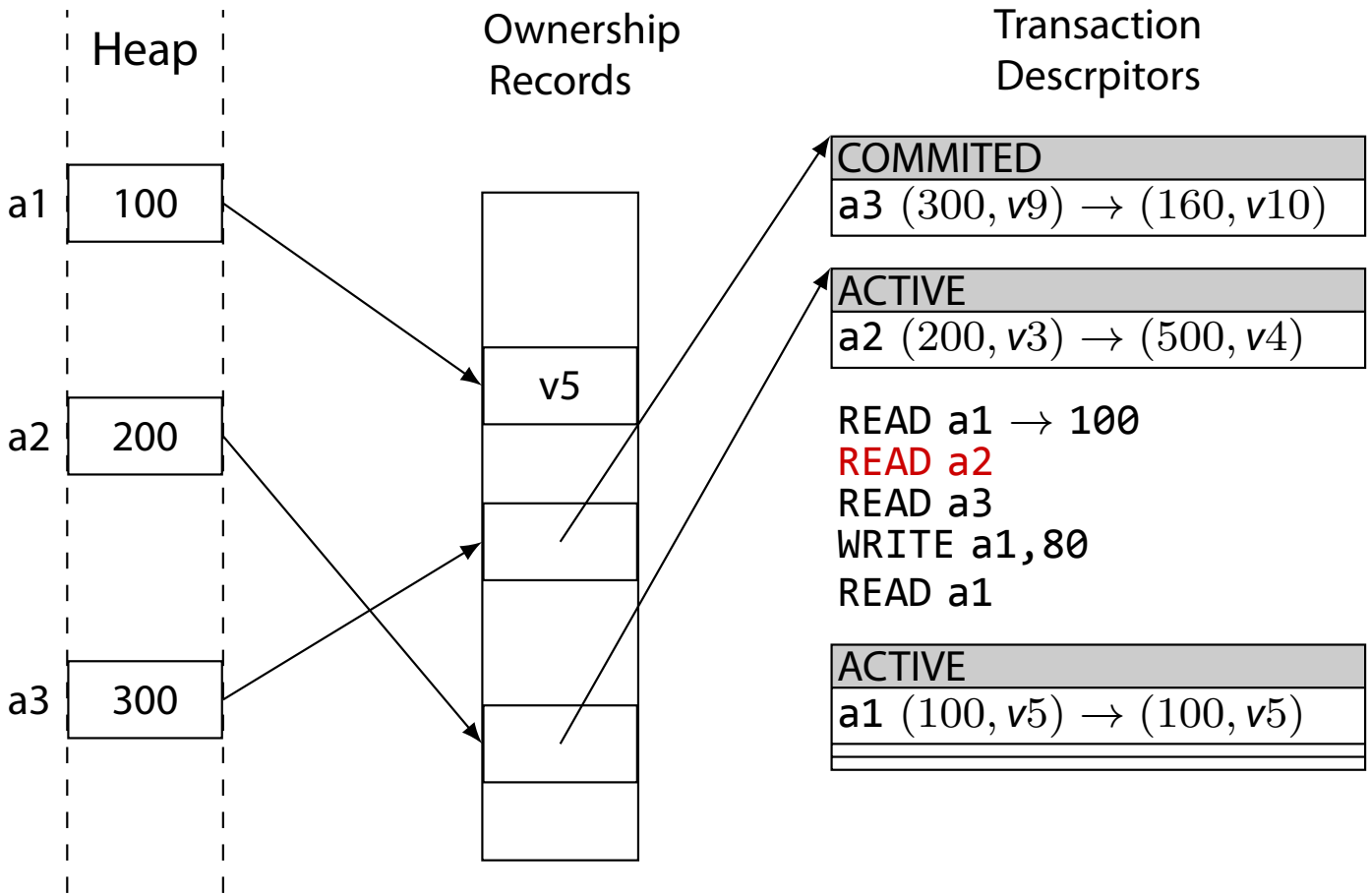
Harris and Fraser 2003: Data structures



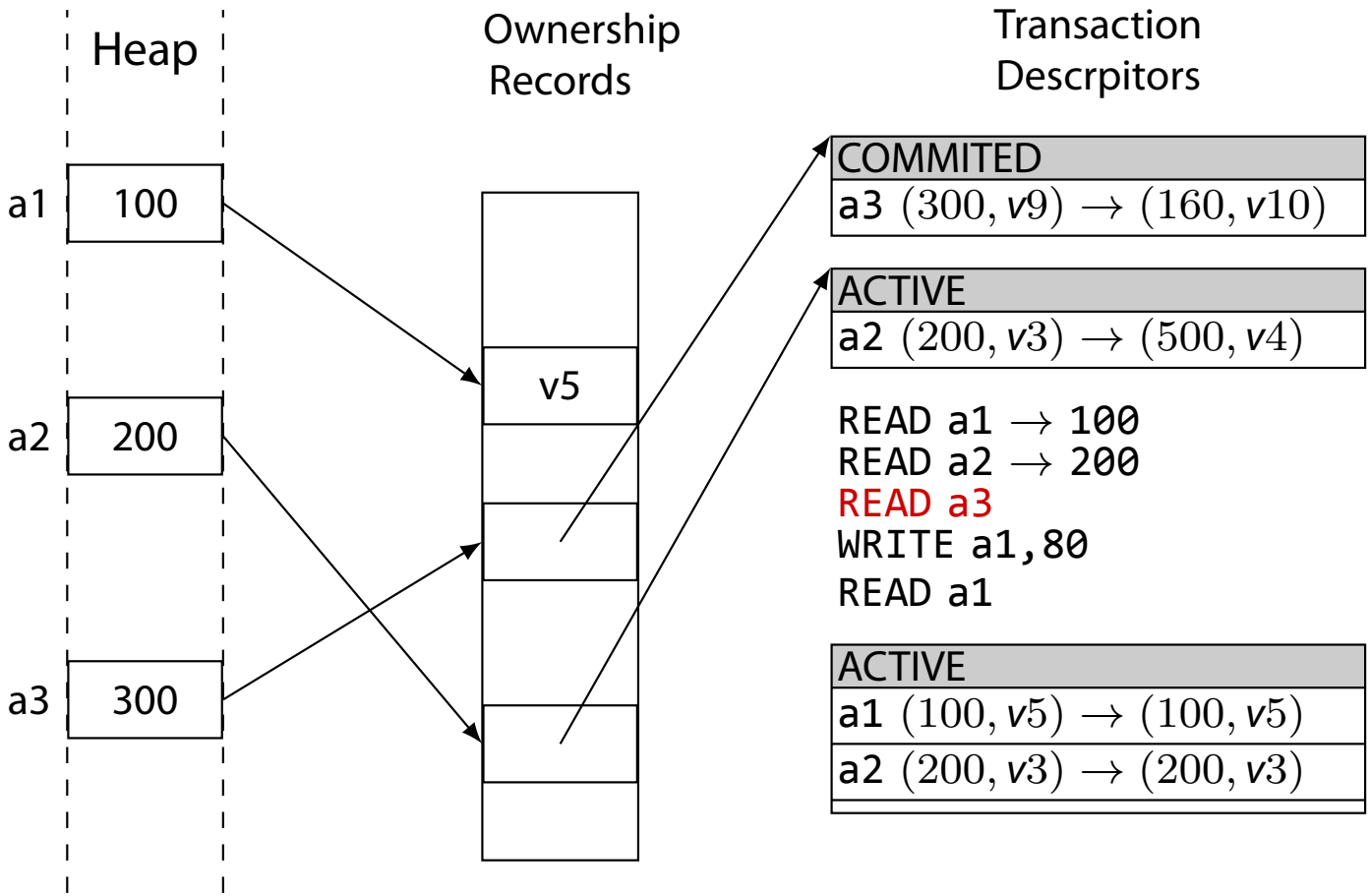
Harris and Fraser 2003: Read and Write



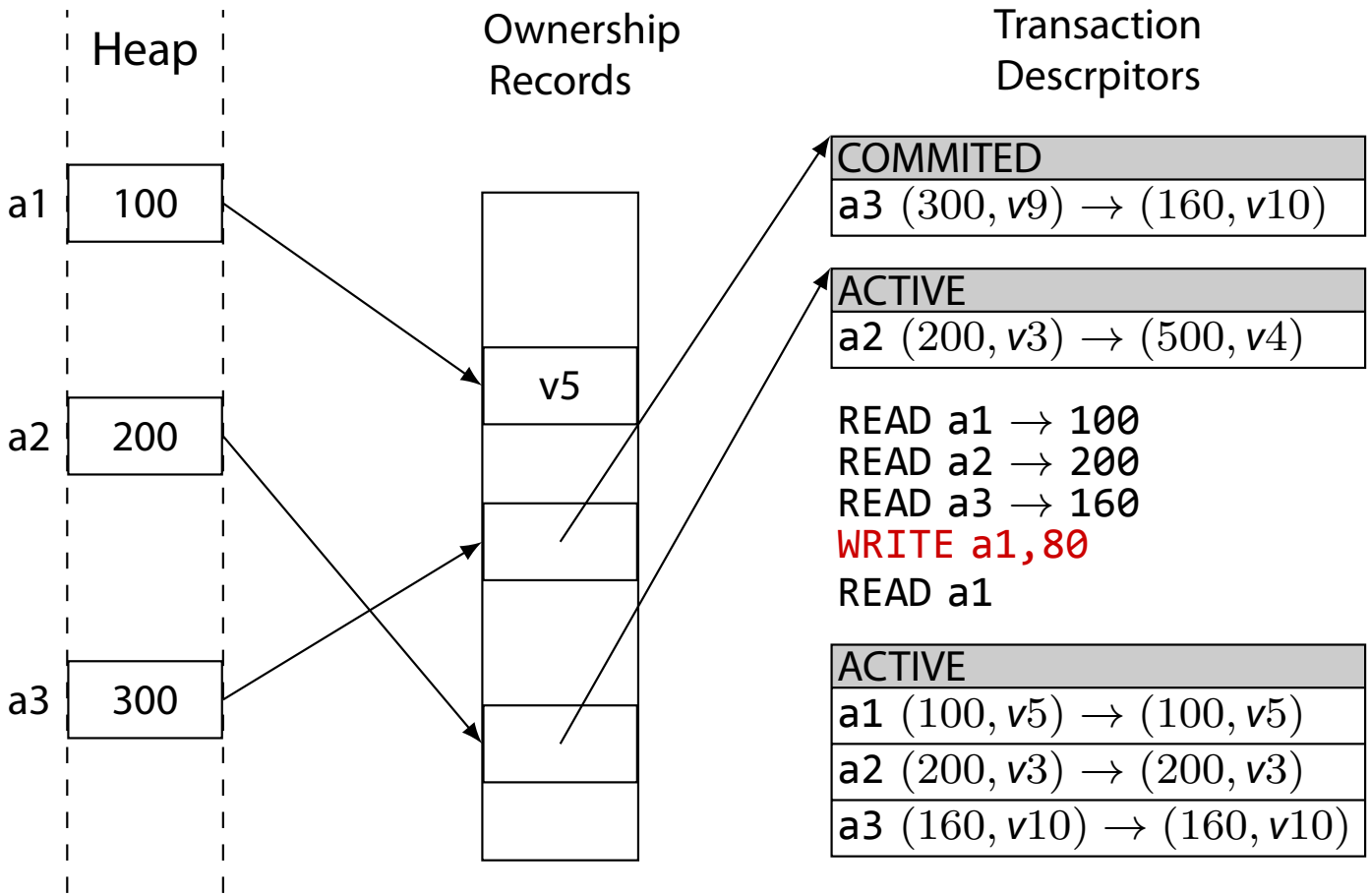
Harris and Fraser 2003: Read and Write



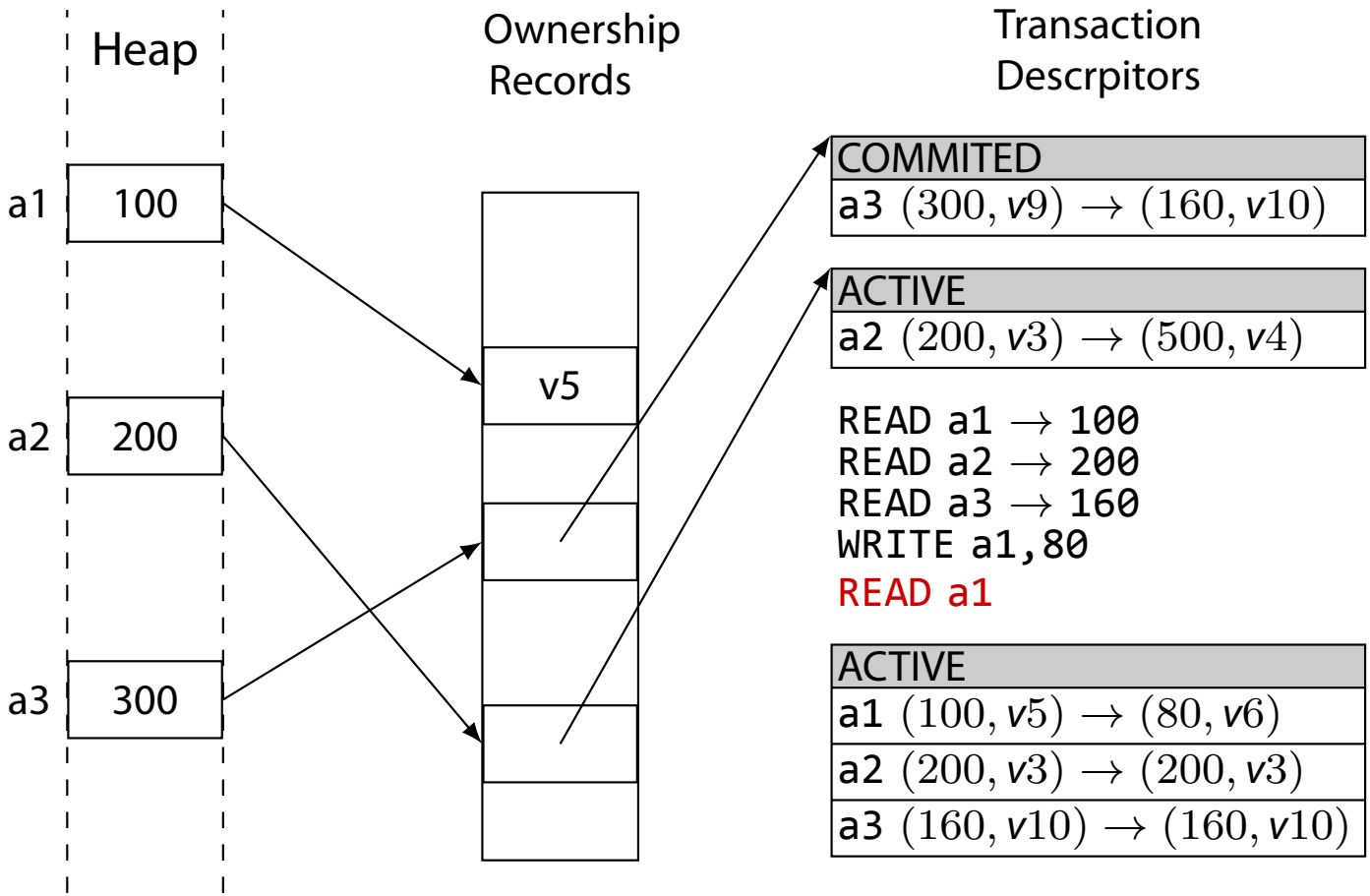
Harris and Fraser 2003: Read and Write



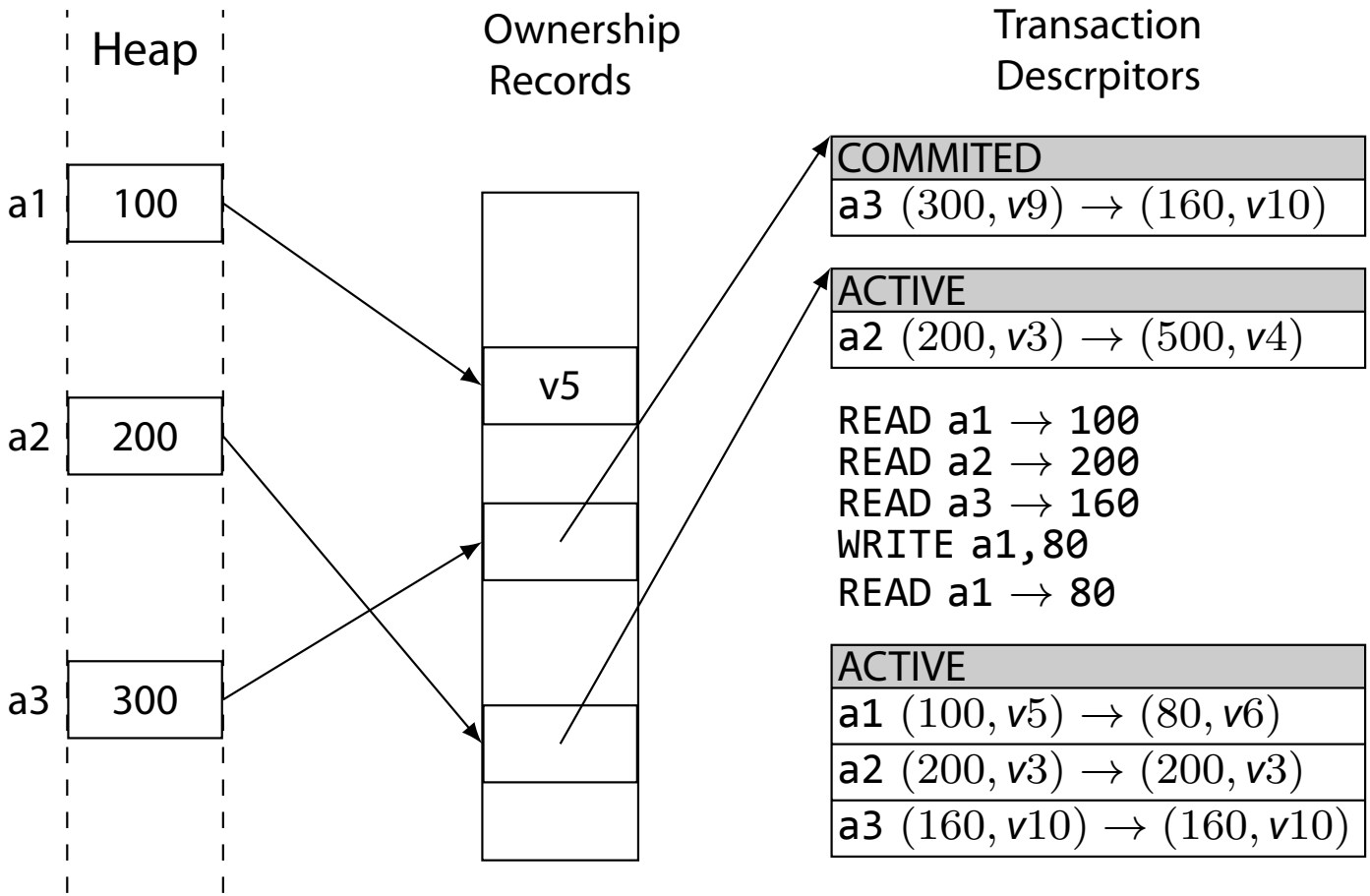
Harris and Fraser 2003: Read and Write



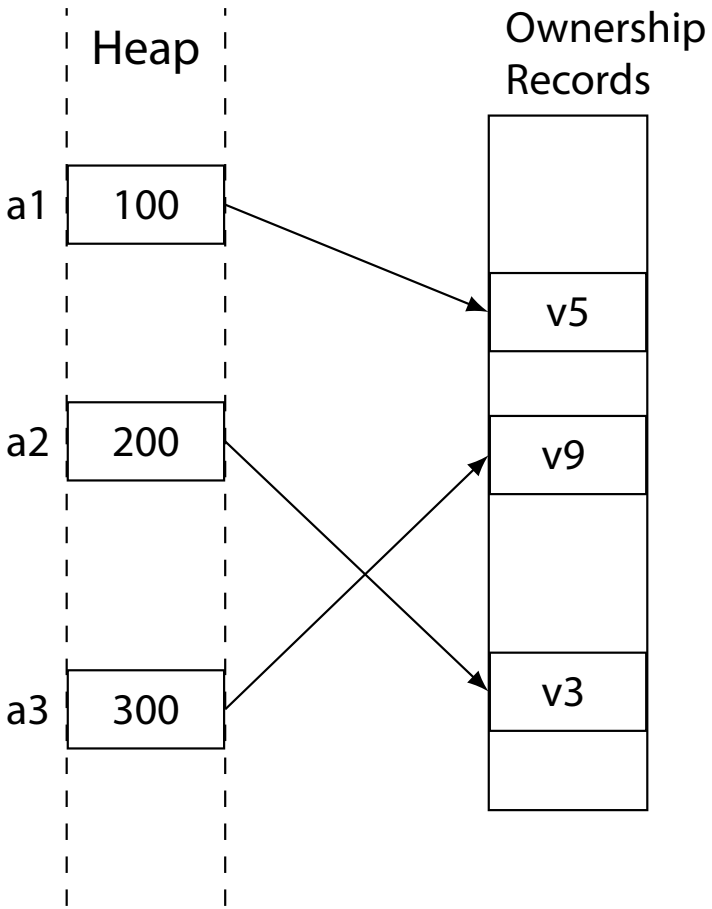
Harris and Fraser 2003: Read and Write



Harris and Fraser 2003: Read and Write



Harris and Fraser 2003: Commit

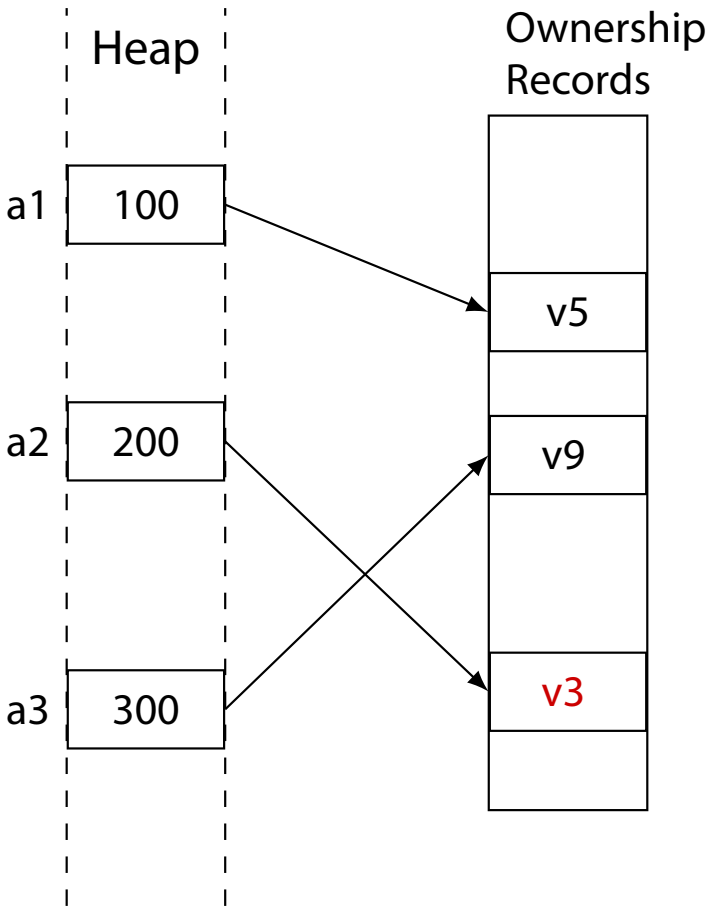


Transaction Descrpitors

ACTIVE
a2 $(200, v3) \rightarrow (60, v4)$
a3 $(300, v9) \rightarrow (160, v10)$

ACTIVE
a2 $(200, v3) \rightarrow (500, v4)$

Harris and Fraser 2003: Commit

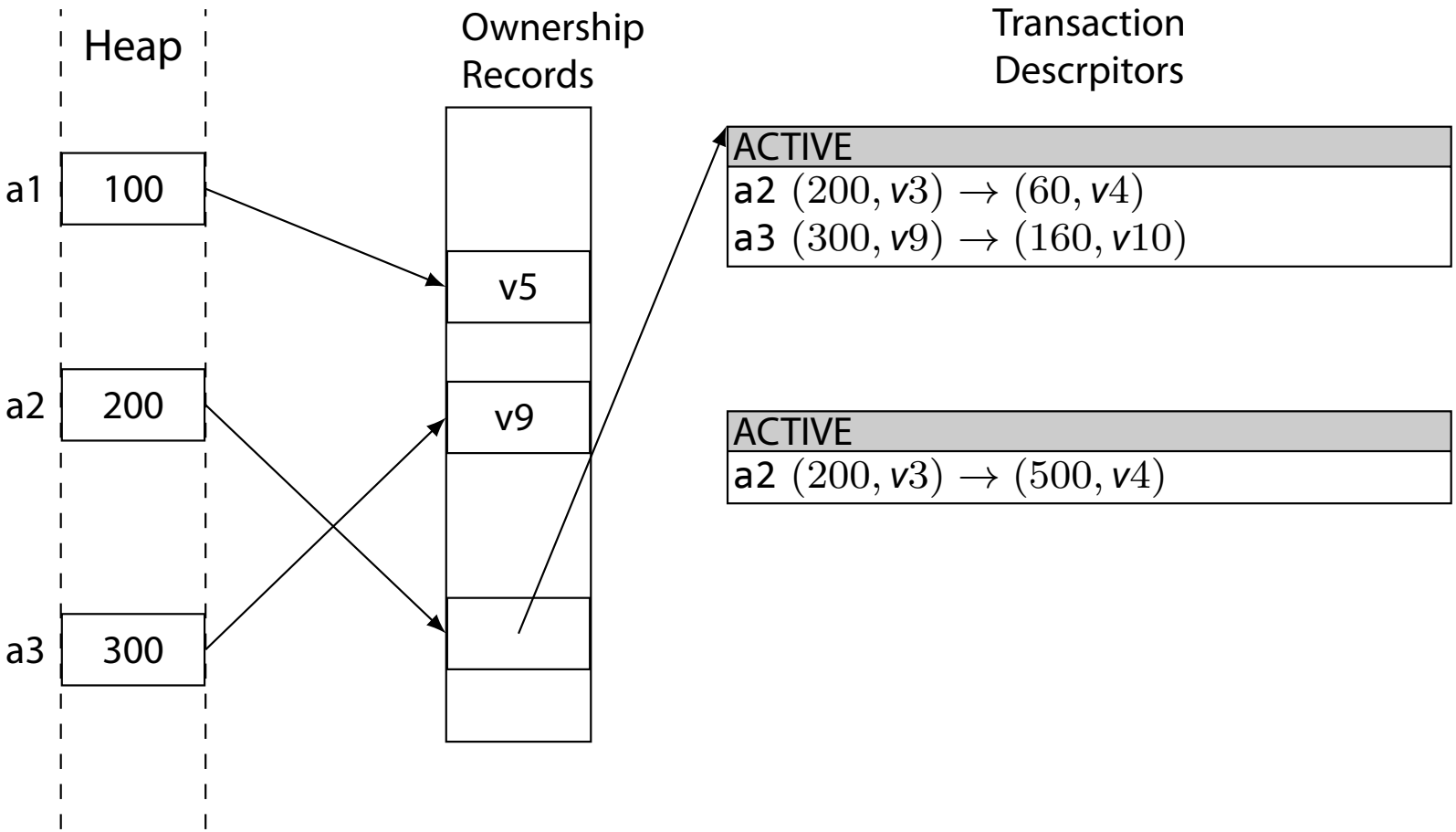


Transaction Descrpitors

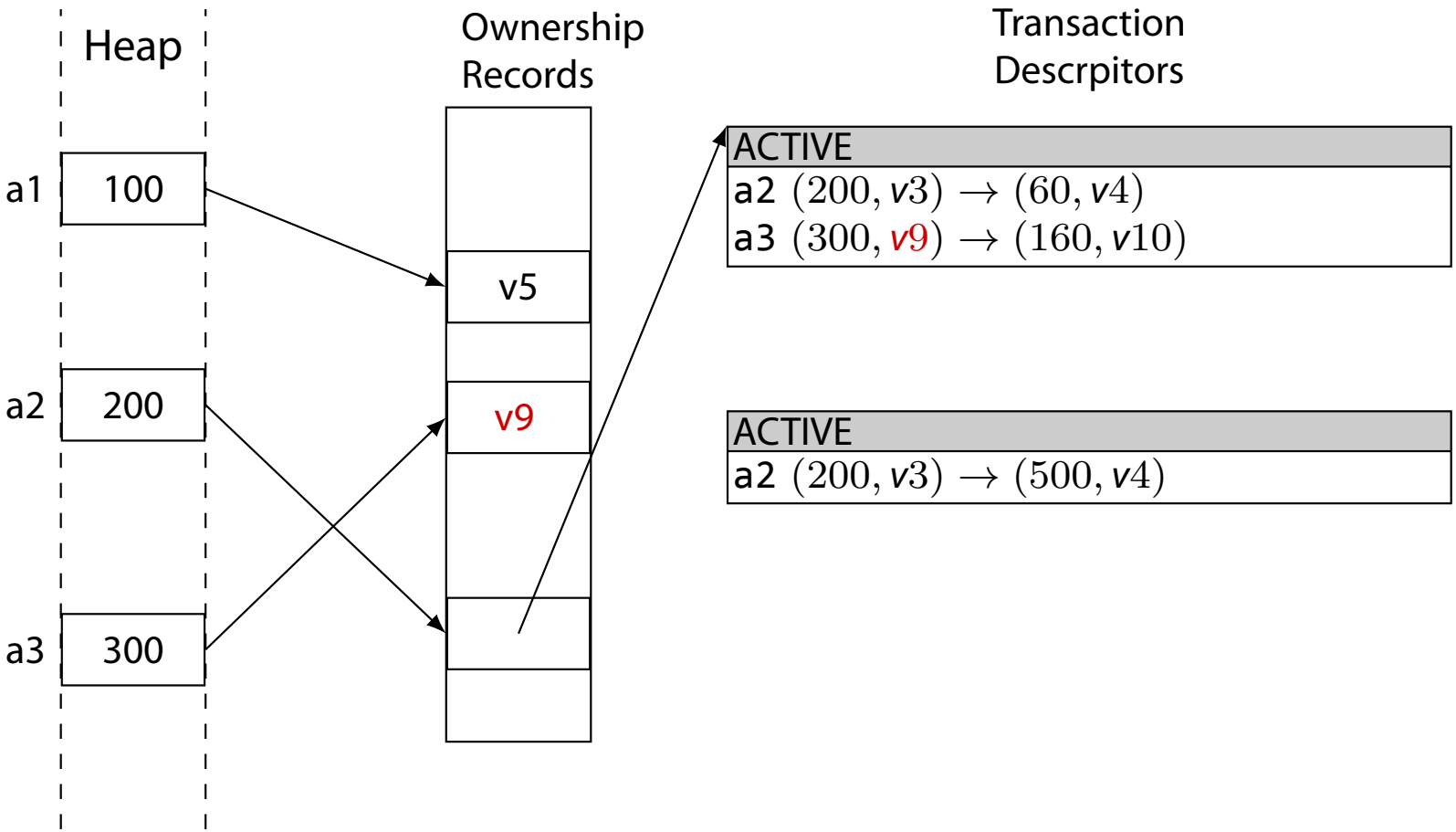
ACTIVE
a2 $(200, v3) \rightarrow (60, v4)$
a3 $(300, v9) \rightarrow (160, v10)$

ACTIVE
a2 $(200, v3) \rightarrow (500, v4)$

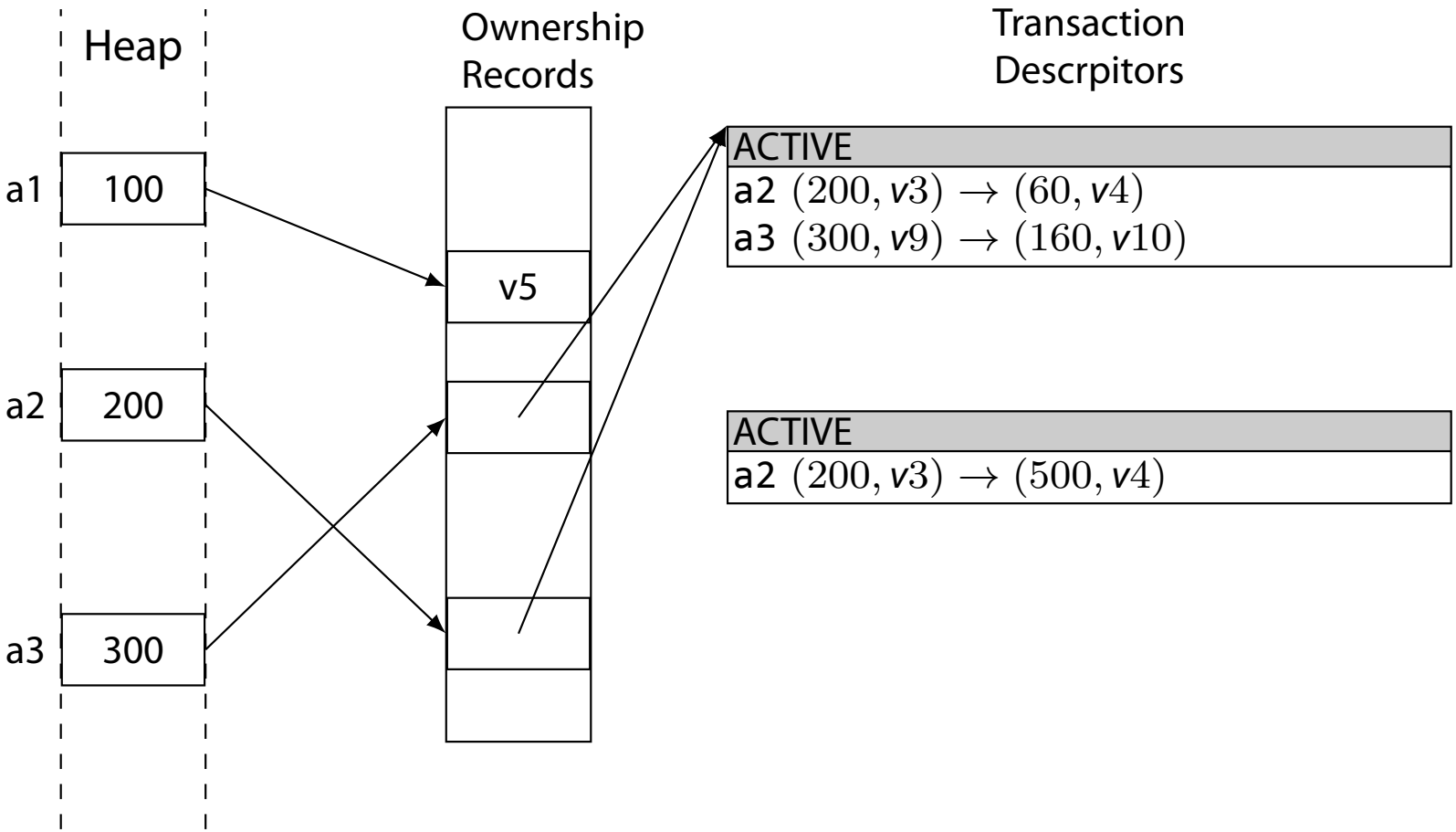
Harris and Fraser 2003: Commit



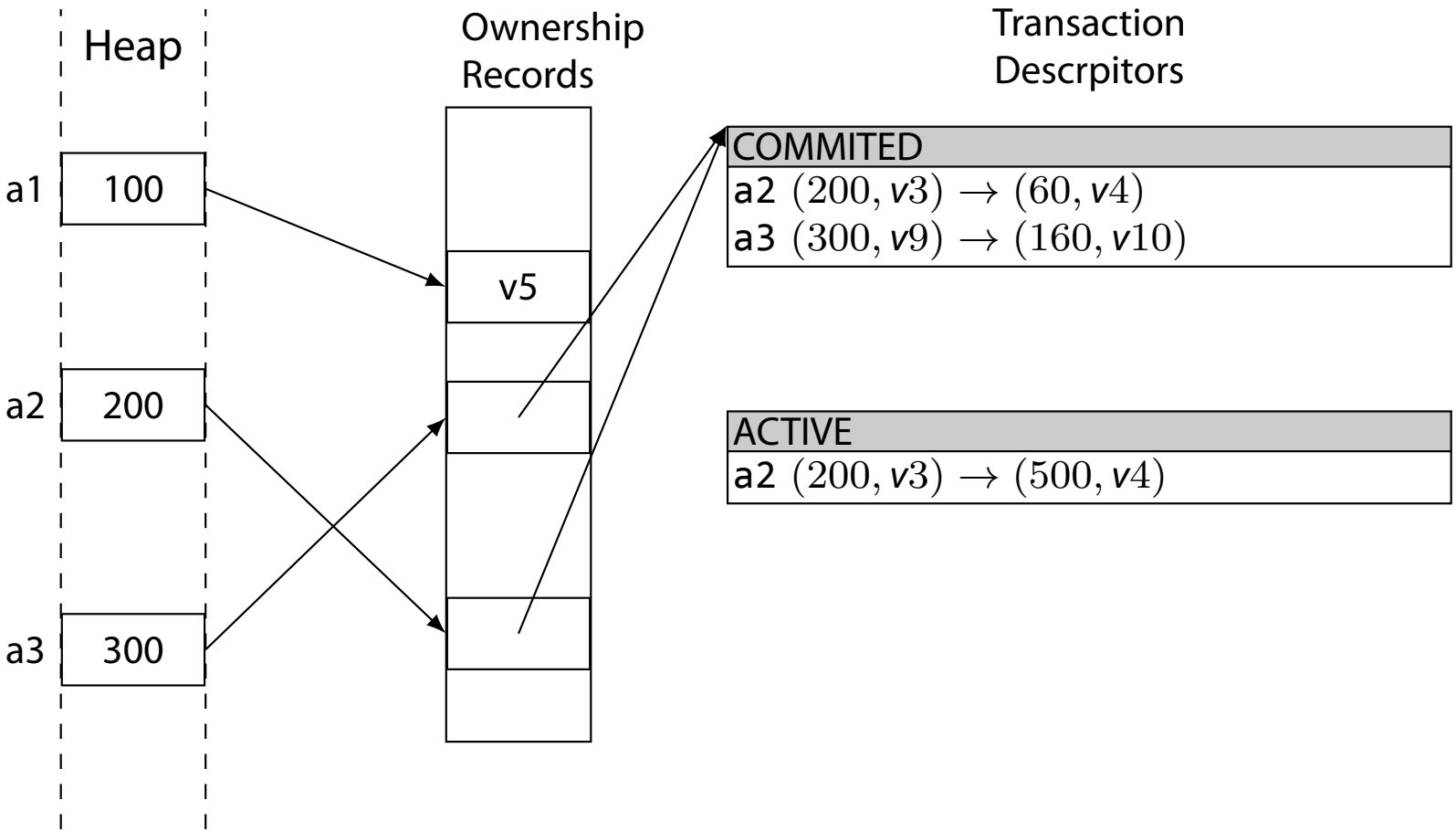
Harris and Fraser 2003: Commit



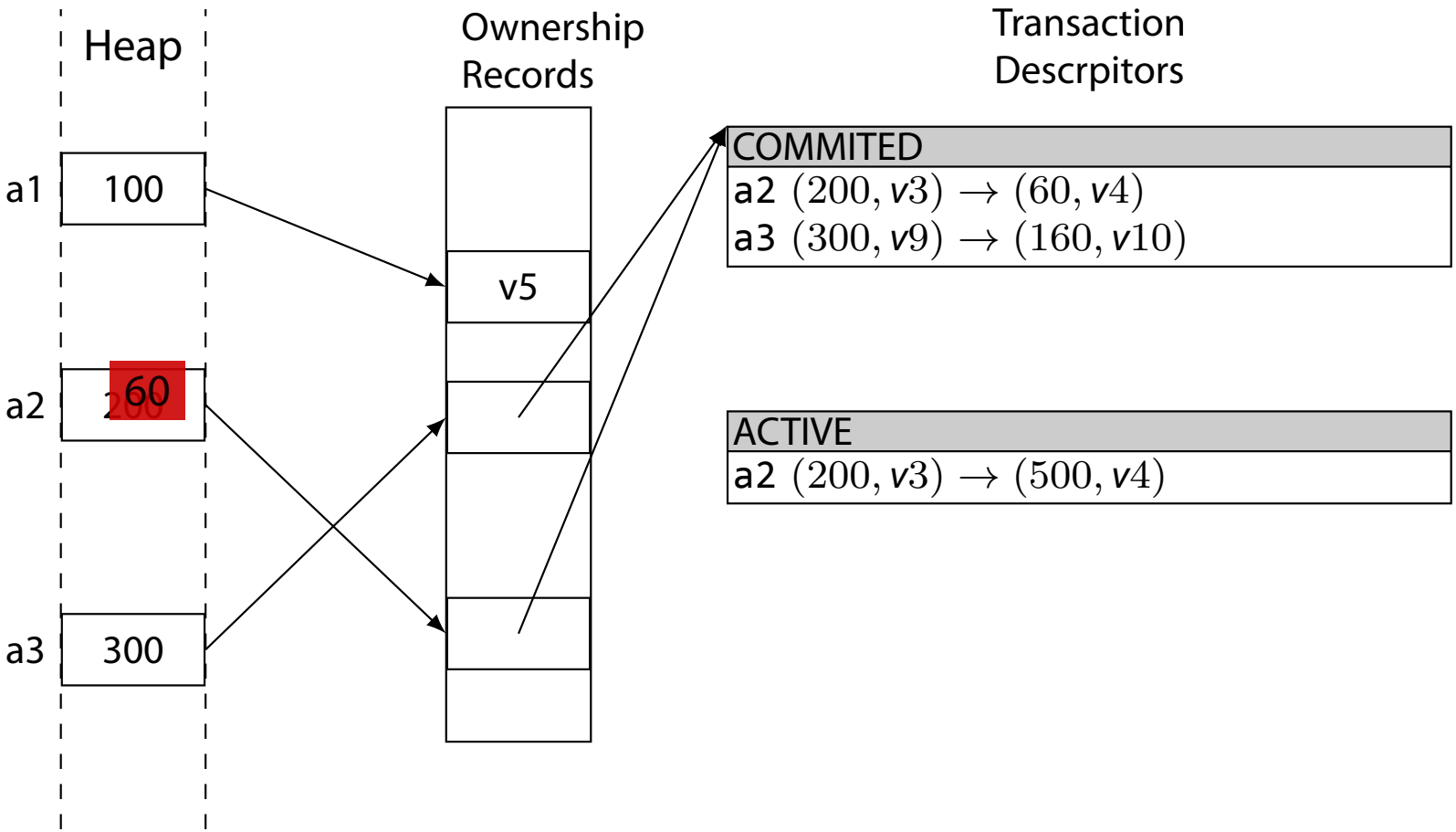
Harris and Fraser 2003: Commit



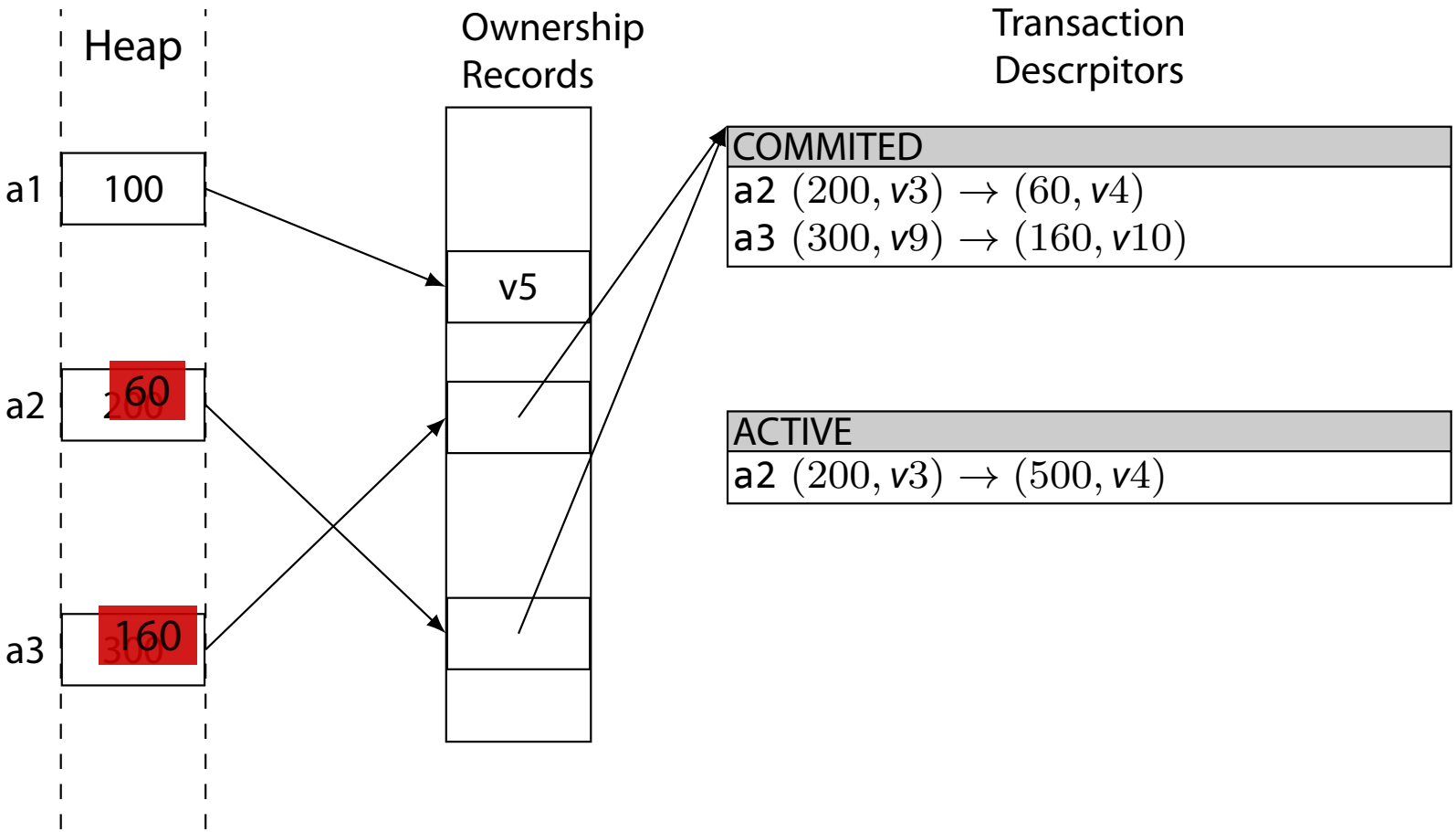
Harris and Fraser 2003: Commit



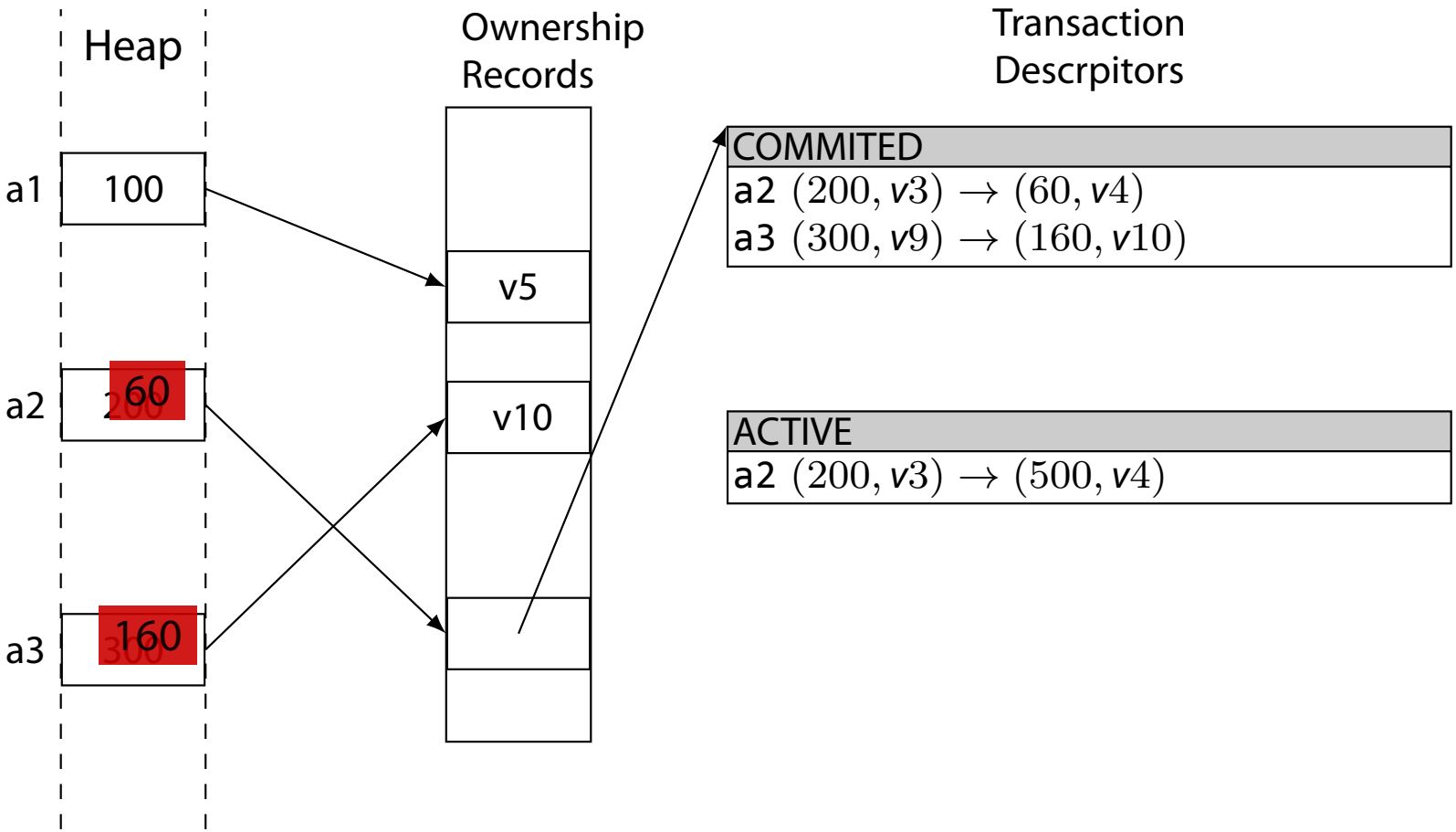
Harris and Fraser 2003: Commit



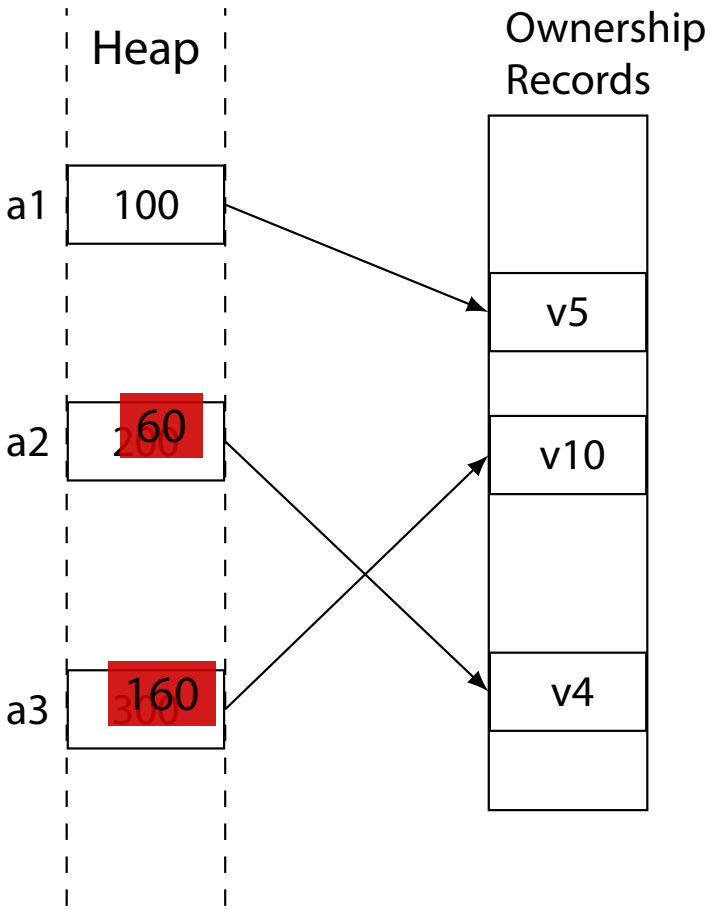
Harris and Fraser 2003: Commit



Harris and Fraser 2003: Commit



Harris and Fraser 2003: Commit

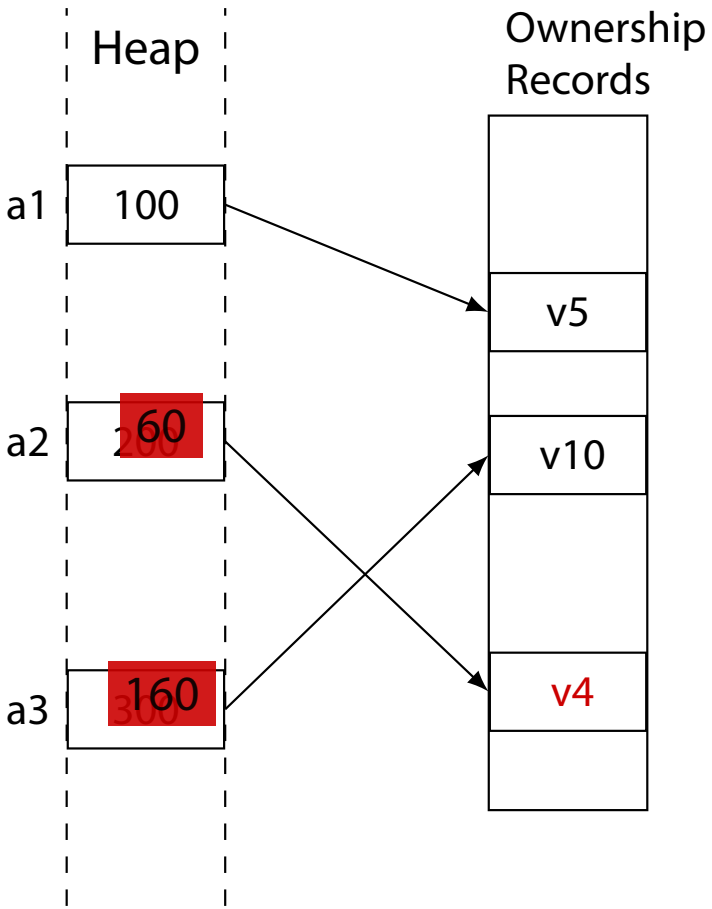


Transaction Descrpitors

COMMITED
a2 $(200, v3) \rightarrow (60, v4)$
a3 $(300, v9) \rightarrow (160, v10)$

ACTIVE
a2 $(200, v3) \rightarrow (500, v4)$

Harris and Fraser 2003: Commit

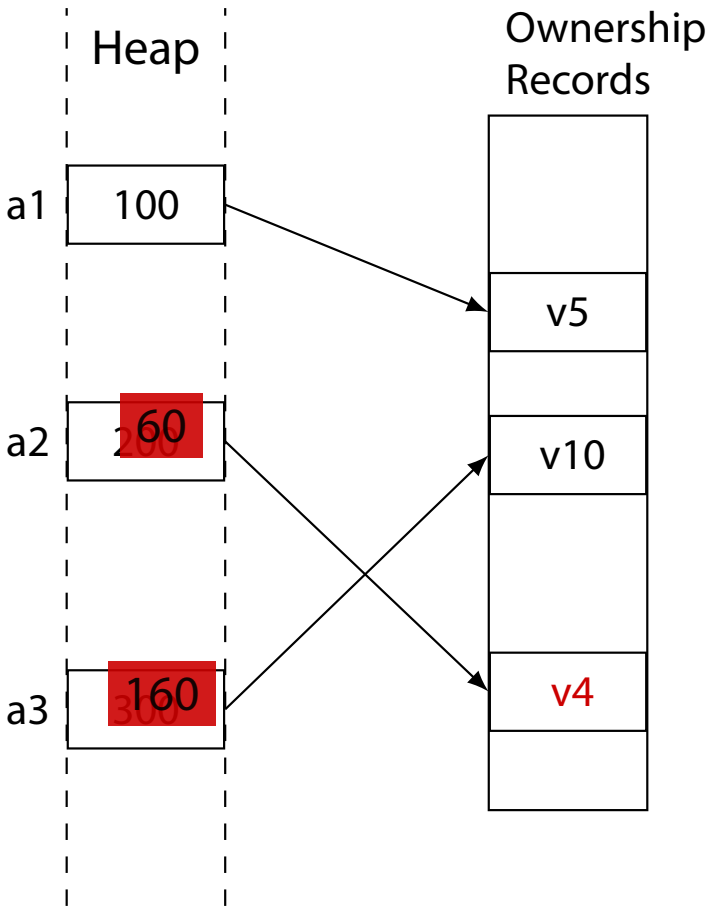


Transaction Descrpitors

COMMITED
a2 $(200, v3) \rightarrow (60, v4)$
a3 $(300, v9) \rightarrow (160, v10)$

ACTIVE
a2 $(200, v3) \rightarrow (500, v4)$

Harris and Fraser 2003: Commit



Transaction Descrpitors

COMMITED
a2 $(200, v3) \rightarrow (60, v4)$
a3 $(300, v9) \rightarrow (160, v10)$

ABORTED
a2 $(200, v3) \rightarrow (500, v4)$

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 $\rightarrow$ apply undo-log
- Lazy
  - keep updates in a write buffer (redo-log)
  - if commit succeeds $\rightarrow$ apply redo-log

Version Management

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

McRT-STM

Version Management:

- eager (undo-log)

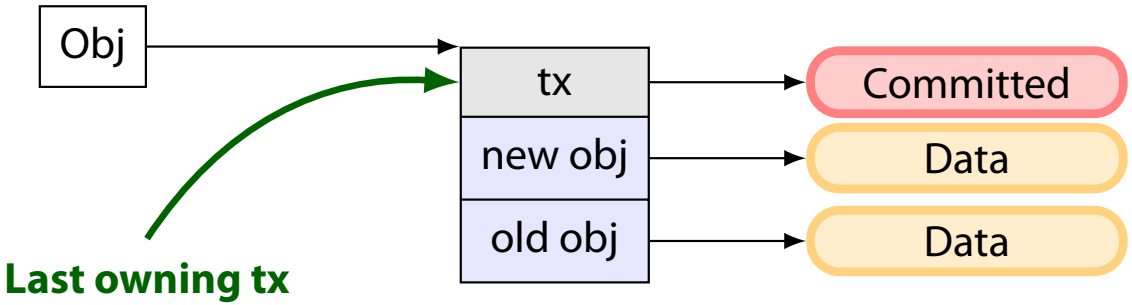
Concurrency control:

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

Conflict detection granularity

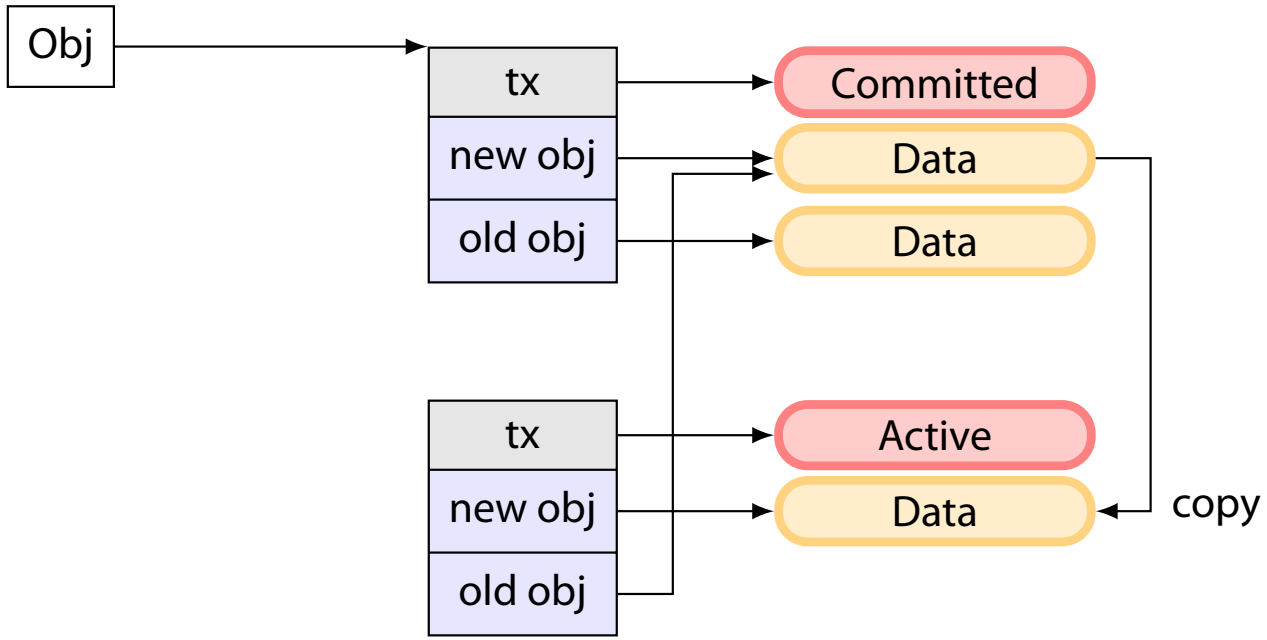
- cache-line
  - mainly used in HTM
- address based (word)
  - mapped addresses to fixed-size metadata table
- object
  - metadata can be added to the object header
  - access to different fields can cause conflicts
  - (more) transaparent to programmer
  - difficult for languages like C
  - TM coupled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

Herlihy et al. 2003



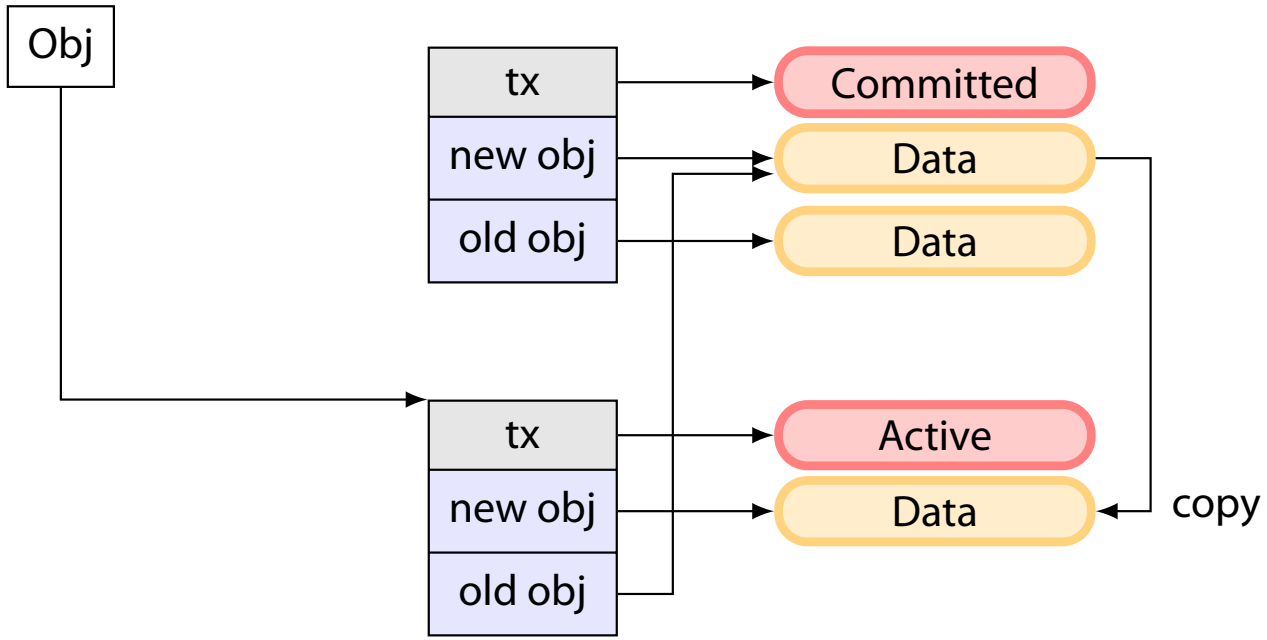
- `current = (tx.state == COMMITTED) ? new:old`

Herlihy et al. 2003



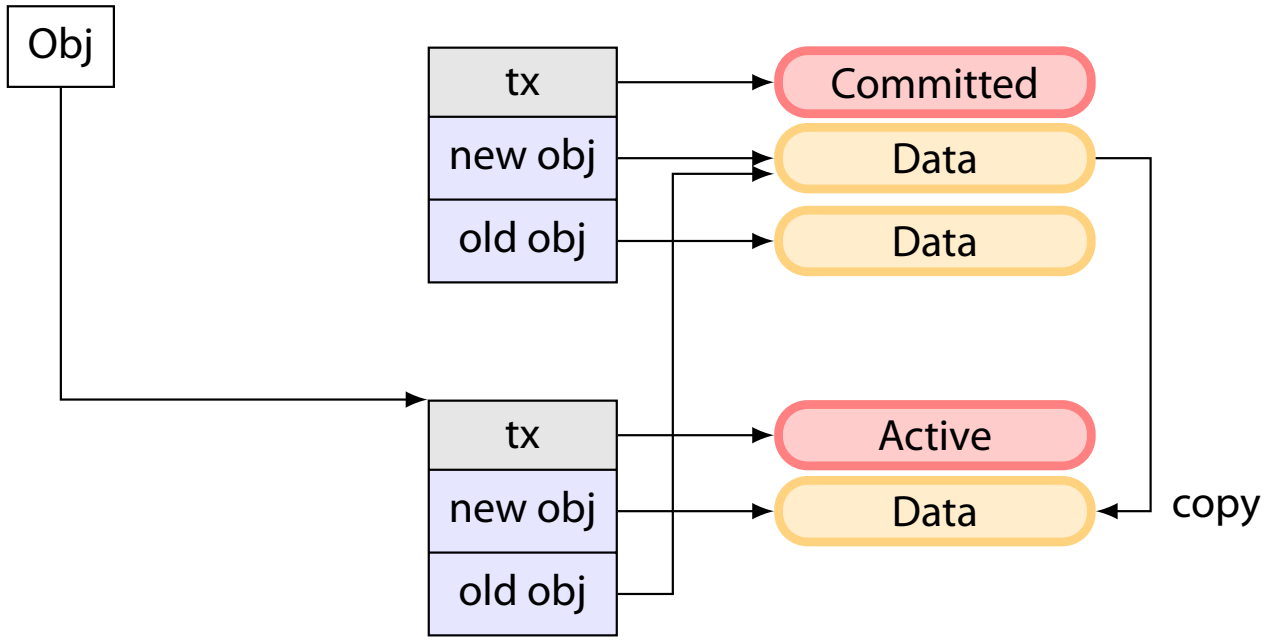
- `current = (tx.state == COMMITTED) ? new:old`

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 standard[1], gcc)
- ...

[1] http://software.intel.com/file/21569

Hardware TM

- ~~Sun~~Oracle's Rock
- Intel Transaction Synchronization eXtensions
  - Hardware Lock Elision (HLE)
    - `XACQUIRE`
    - `XRELEASE`
  - Restricted Transactional Memory (RTM)
    - `XBEGIN`
    - `XEND`
    - `XABORT`
  - HLE support for pthread mutexes in glibc
- 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