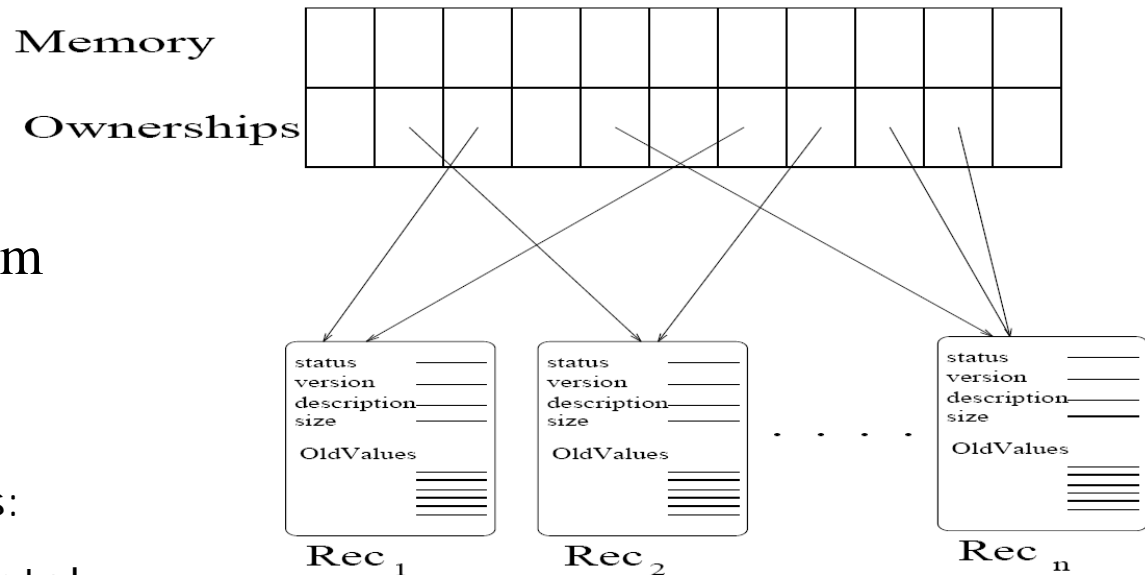


Transactional Memory

Part 2: Software-Based Approaches

Word-based STM (Shavit&Touitou)

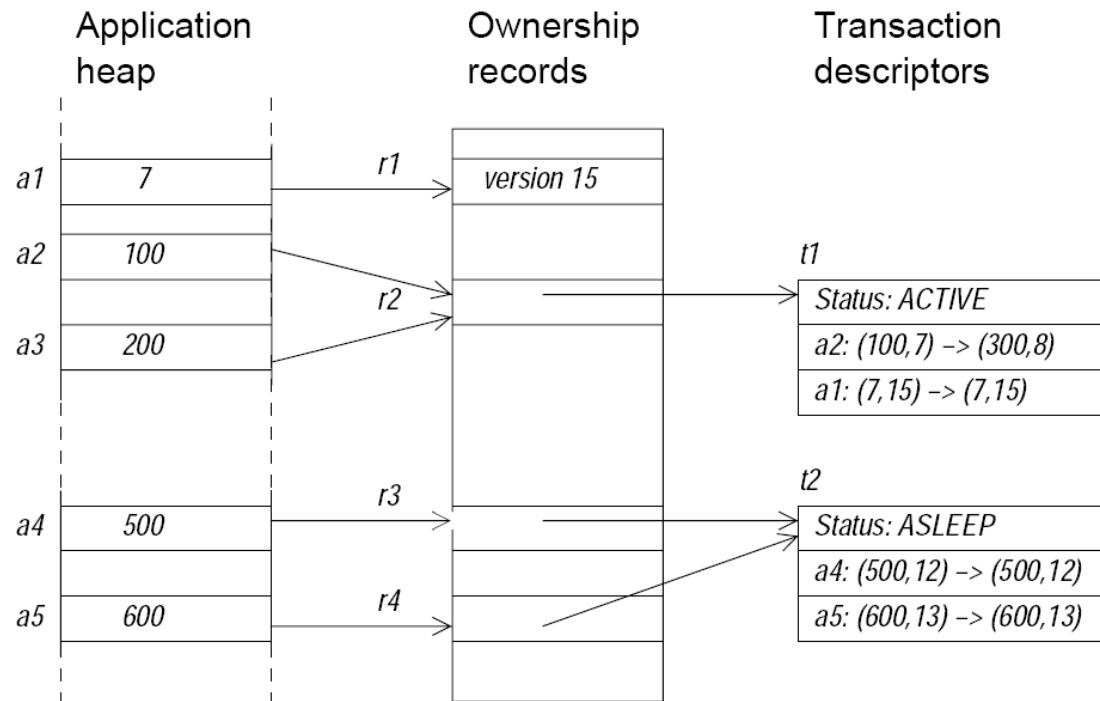


- Guarantees lock-freedom
- Uses a non-recursive “helping” strategy
- Limitations
 - Static transactions: ownership must be acquired in some total order to avoid livelock
 - Memory costs
 - Helping requires transaction to yield same results under multiple (partial) executions

Basic transaction process:

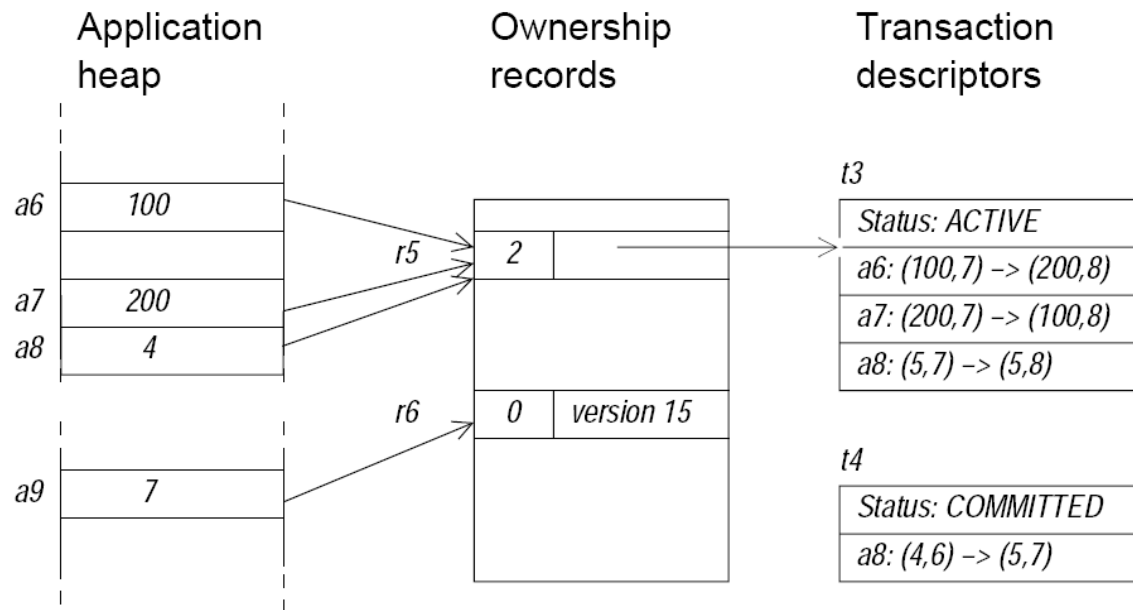
1. Read old values into transaction record
2. Acquire ownership of memory location for each value
 - a. Succeed: Perform transaction; update memory; release ownership.
 - b. Fail: release ownership; help if not already helping (non-recursive); abort.

Word-based STM (WSTM): Harris&Fraser



- Multiple addresses map to the same ownership record.
- Logical state: a (value, version) pair representing the contents of a memory location.
- Ownership record stores either version number or address of transaction descriptor.
- Read/write operations create entries in a transaction descriptor.
- Commit operation attempts to gain ownership of the locations it reads/writes by placing the address of its transaction descriptor in the ownership records.
- Guarantees obstruction-free execution.

Stealing



- Transaction attempting to commit, “steals” transaction entry from conflicting transaction
- Provides non-blocking commit operation (guarantee of obstruction-free execution)
- Requires ownership record to store the number of transaction holding a transaction record for a location mapping to the ownership record

Language Support

Conditional Critical Region (CCR)

Syntax:

```
atomic (condition) {  
    statements;  
}
```

- conditional critical region syntax added to Java
- source-to-bytecode compiler handles translation of atomic blocks and creates separate method of each atomic block
- methods of data access provide STMRead and STMWrite for methods defined for atomic blocks

Translation:

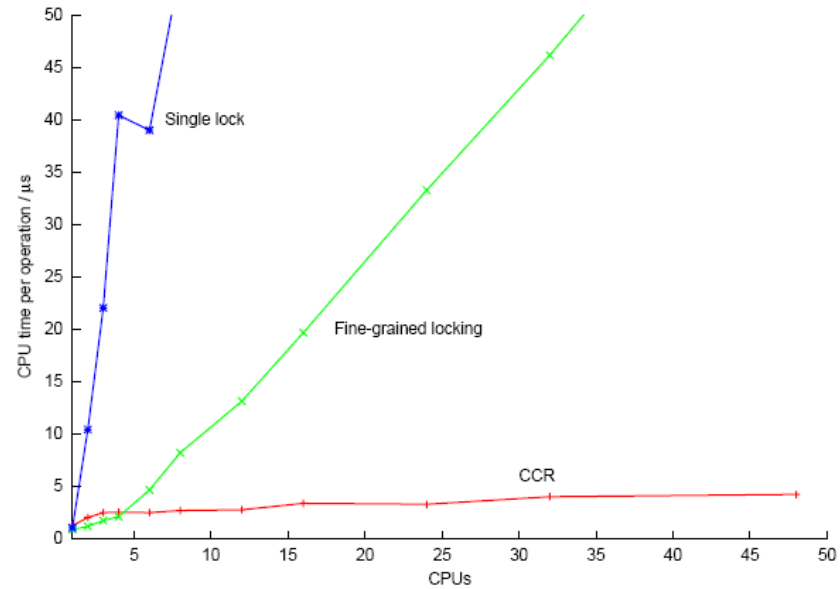
```
boolean done = false;  
while (!done) {  
    STMStart();  
    try {  
        if (condition) {  
            statements;  
            done = STMCommit();  
        } else {  
            STMWait();  
        } catch (Throwable t) {  
            done = STMCommit();  
            if (done) {  
                throw t;  
            }  
        }  
    }  
}
```

Performance

μs per operation

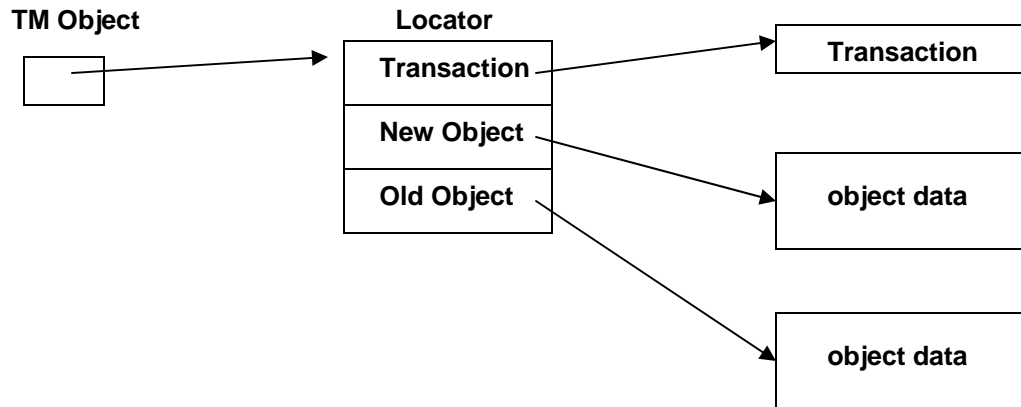
CPU _s	1% updates			16% updates		
	CCR	S-1	FG-1	CCR	S-1	FG-1
1	1.8	1.1	0.9	1.9	1.1	0.9
2	1.8	3.3	0.9	2.0	7.9	1.0
3	2.1	25	1.3	2.4	23	1.1
4	1.8	30	1.1	2.4	30	1.4

CPU _s	size=256			size=4096		
	CCR	S-1	FG-1	CCR	S-1	FG-1
1	4.8	2.1	2.6	5.1	2.3	2.7
2	6.2	17	5.0	6.3	17	4.4
3	7.2	27	6.4	7.2	28	6.3
4	7.4	37	8.3	7.5	40	6.9



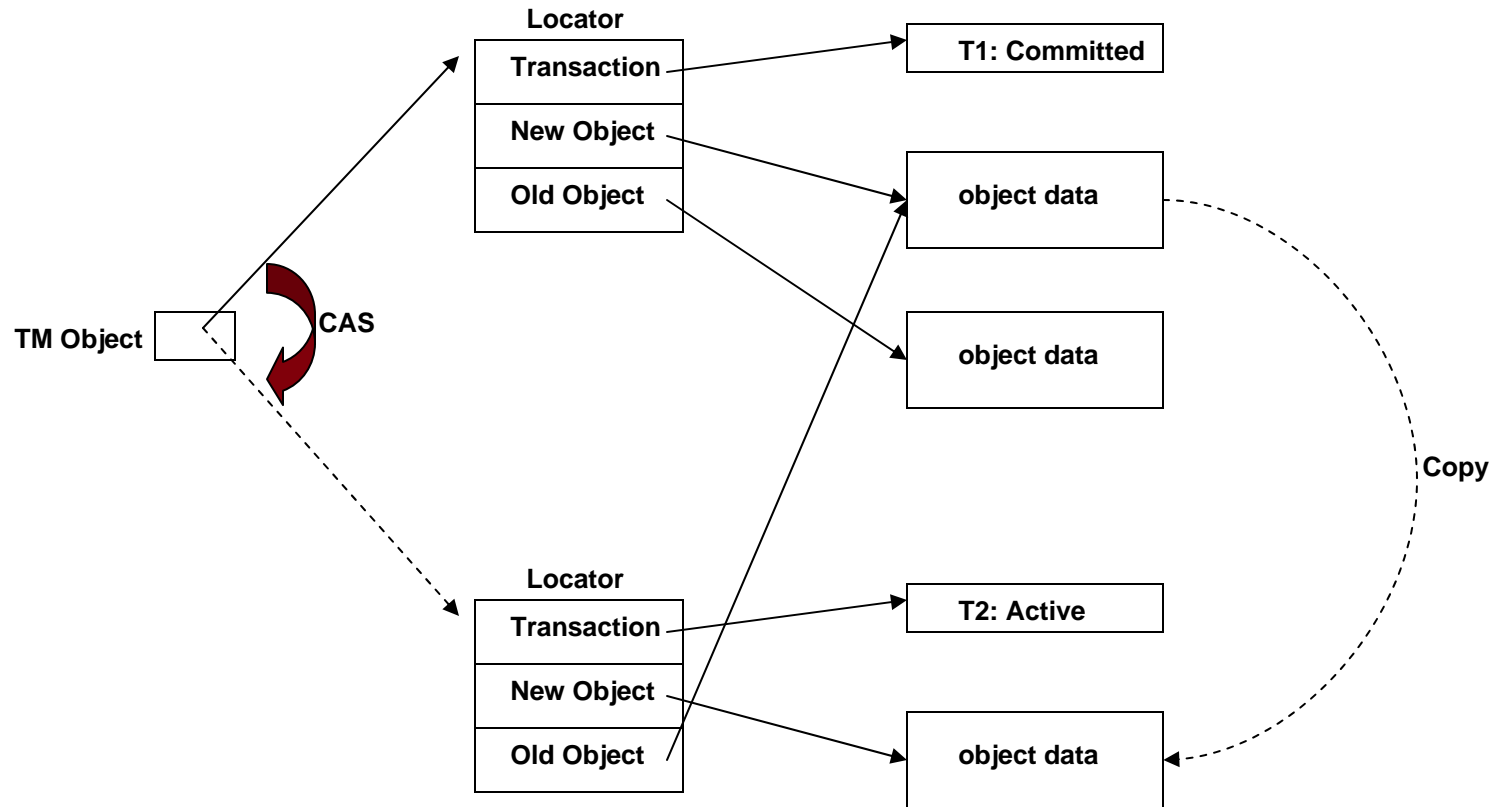
- WSTM is superior to simple synchronization schemes (CCR vs. S-1) on few processors
- WSTM is competitive with sophisticated synchronization schemes (CCR vs. FG-1) on few processors
- WSTM is superior to other synchronization schemes on large number of processors

Dynamic STM (DTSM): Herlihy et.al.

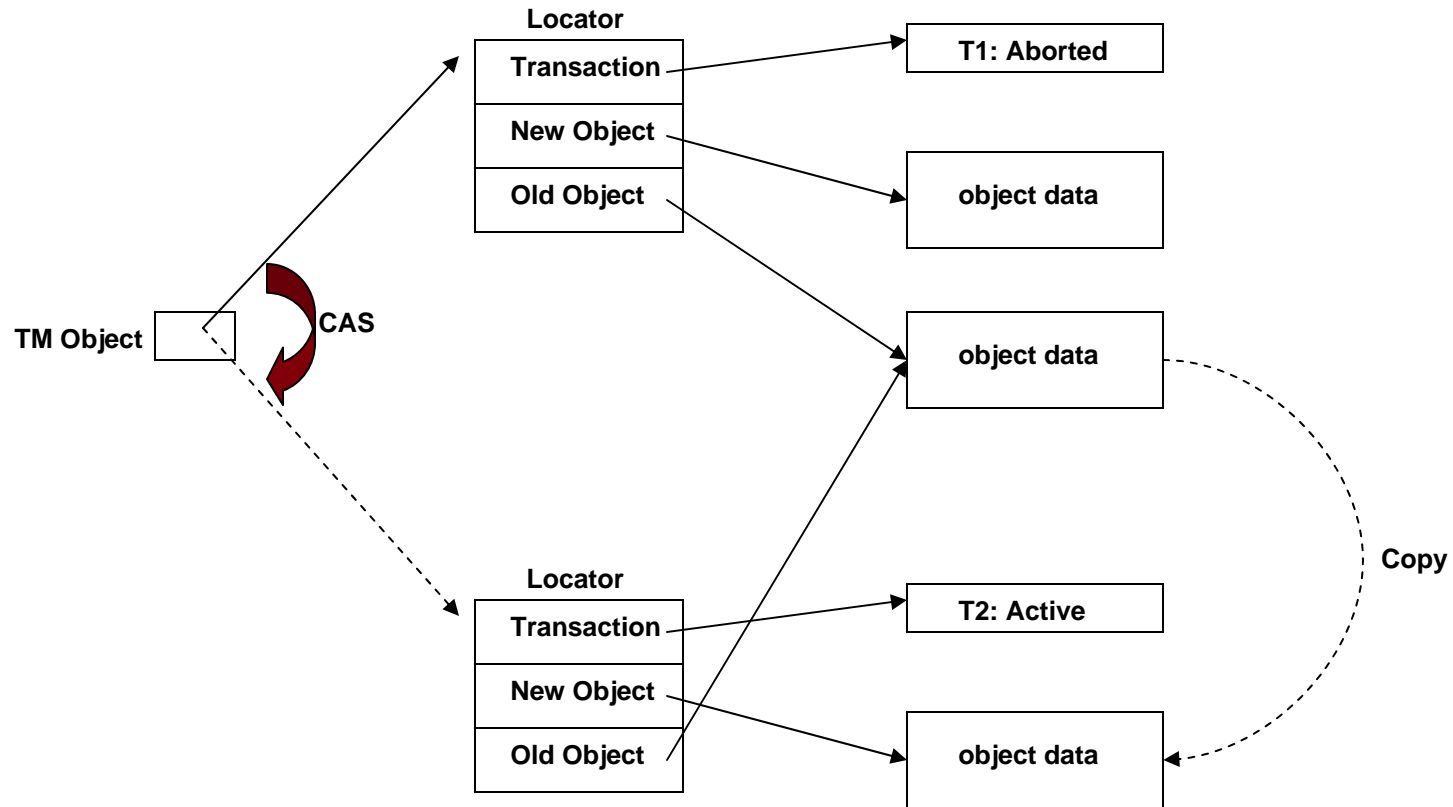


- TMOBJect is a handle for an object.
- An “open” operation on the TMOBJect is required before object can be accessed.
- Transaction state may be: ACTIVE, COMMITTED, ABORTED.
- The “current” form of the object data is maintained (Old Object).
- A shadow copy of to-be-committed updates to the object is also maintained.

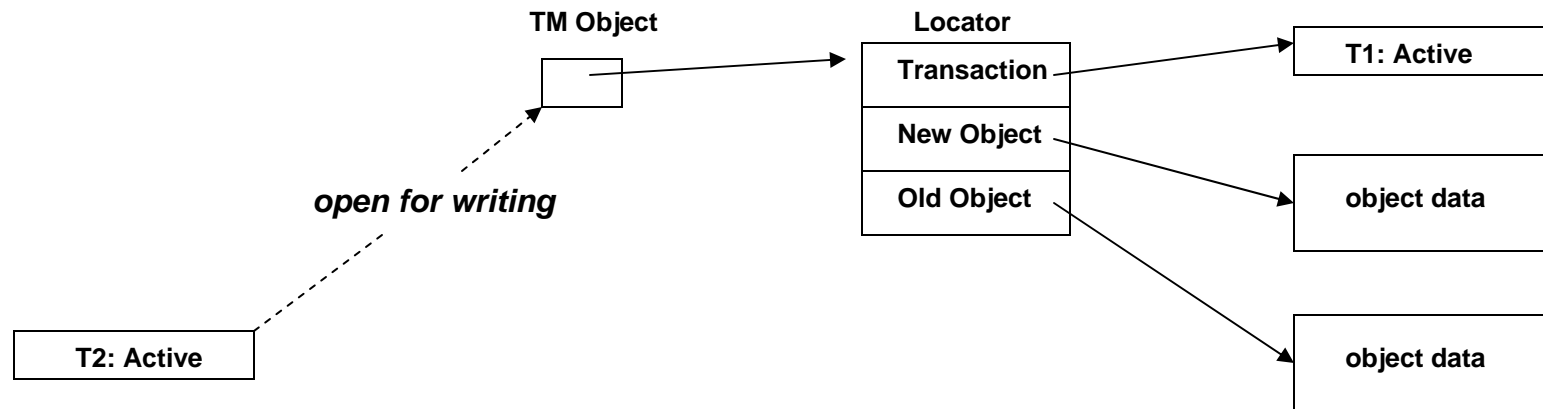
Opening a TMOBJECT for Writing



Opening a TMOBJECT for Writing

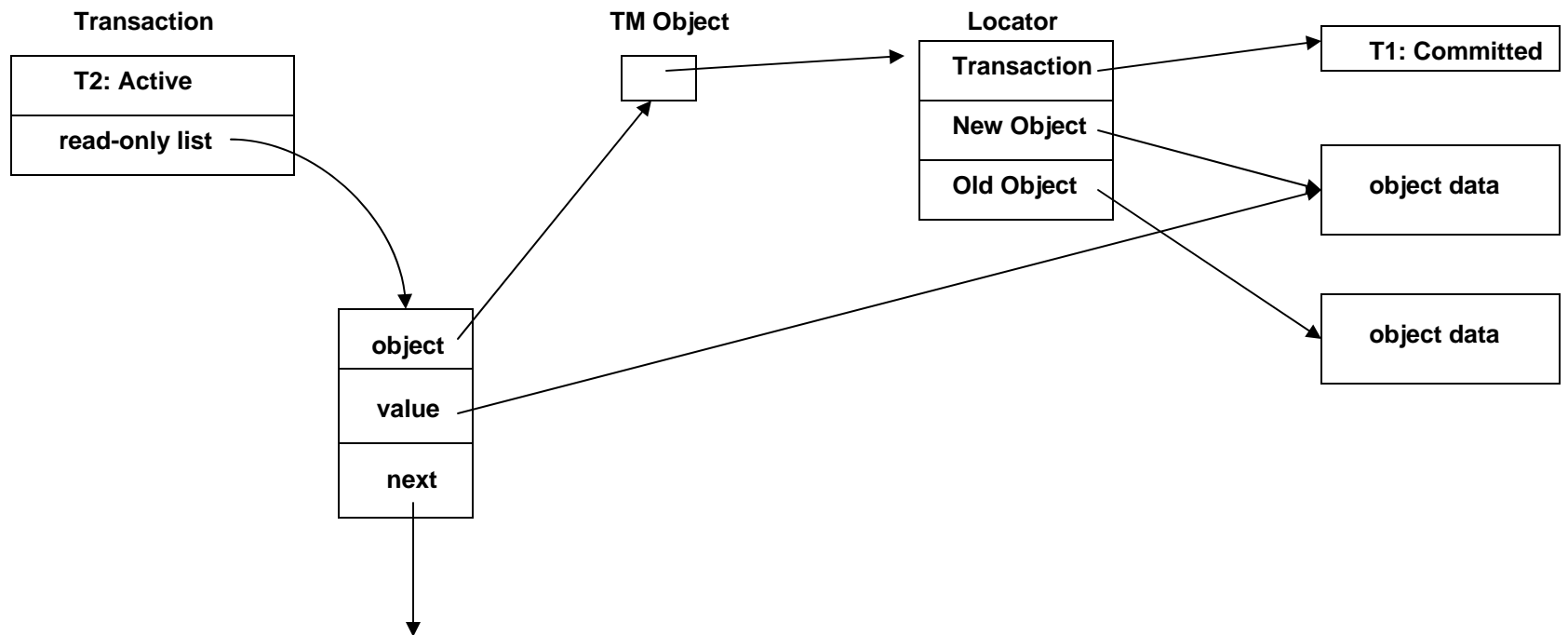


Opening a TMOBJECT for Writing

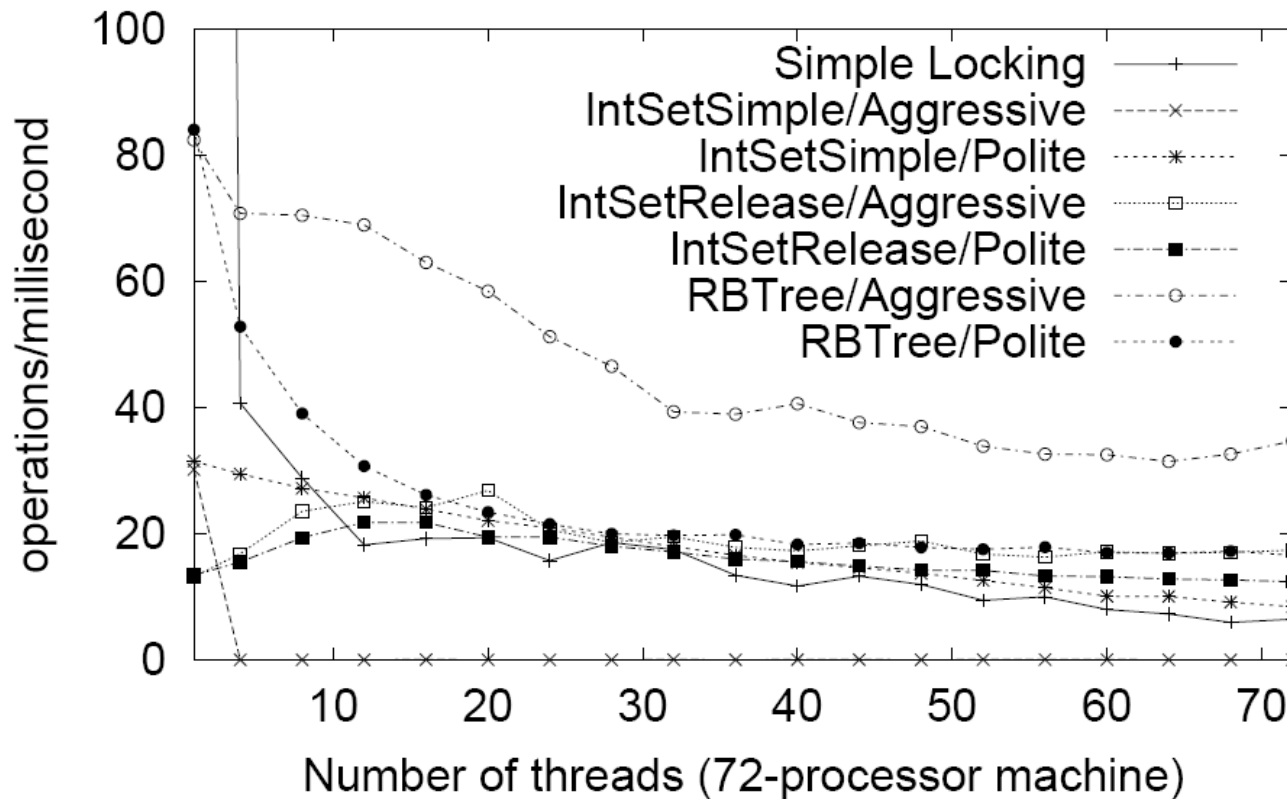


- one of T1 or T2 must abort to resolve conflict without blocking
- each thread has a ContentionManager
 - aggressive – always/immediately aborts conflicting transaction
 - polite – adaptive back-off
- contention reduced by “early release”
 - reference to object dropped before transaction commits
 - releasing transaction must insure that subsequent changes to the released object does not jeopardize consistency

Opening a TMOBJECT for Reading

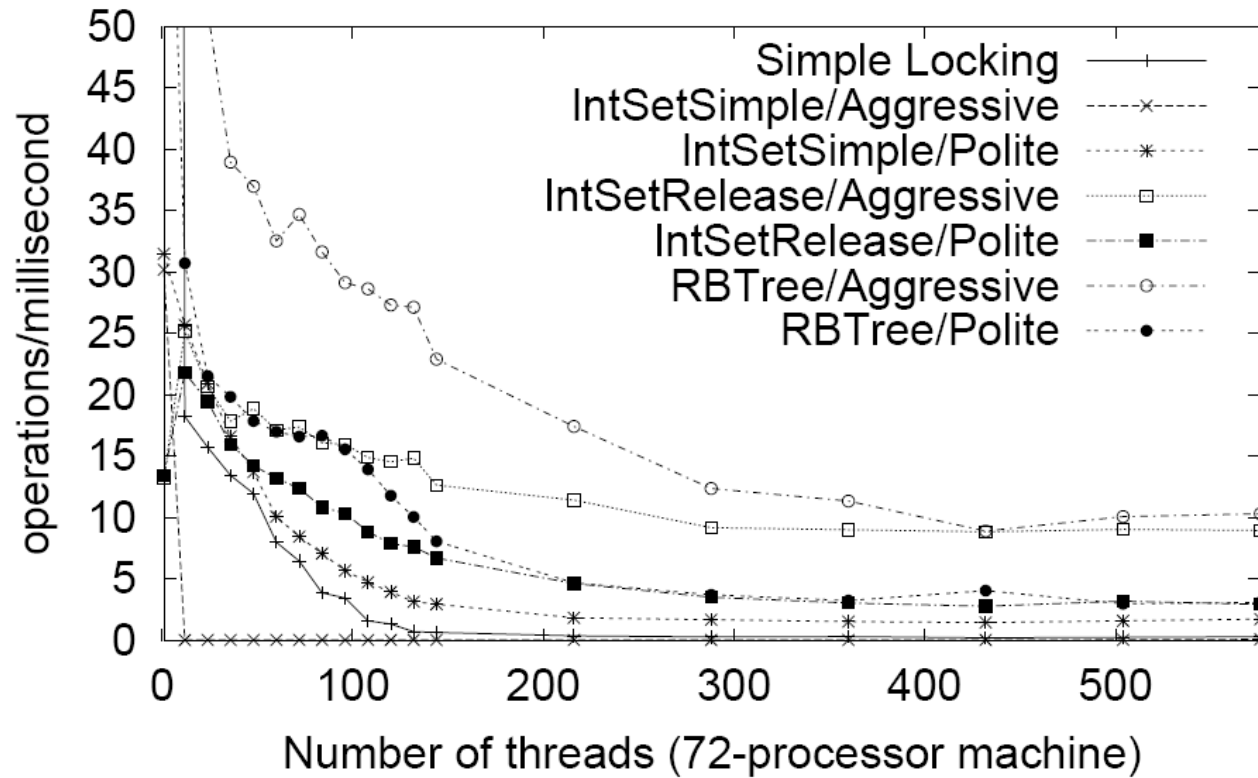


Performance



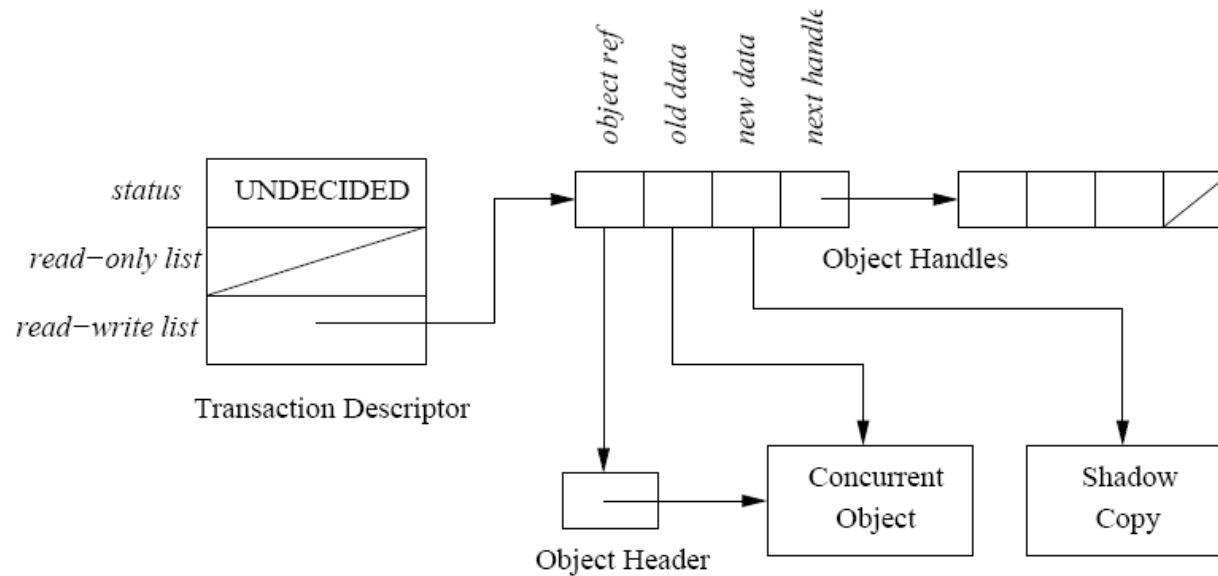
- STM versions competitive with simple locking scheme
- Aggressive contention management can cause performance to collapse under high contention

Performance



- By lowering contention, early release sustains performance of aggressive contention management.
- Contention management useful and has possibly complex relationship to data structure design.

FSTM: Fraser



- Objects are accessed by an *open* operation on the *object header*
- An object may be open in multiple transactions at the same time
- Transaction maintains an *object handle* for each open object
- Object handles are organized into two lists: a *read-only list* and a *read-write list*
- For each writeable object the transaction maintains a *shadow copy* of the object private to the transaction
- Conflicts among transactions are detected and resolved at commit-time
- Guarantees lock-freedom

Commit operation in FTSM

Phase	Description
<i>Acquire</i>	<p>Action: Acquire each object in the read-write list in global total order using atomic CAS for each object</p> <p>Outcomes:</p> <ul style="list-style-type: none"> ■ Abort if conflict with committed transaction detected ■ Help if conflict with uncommitted transaction detected
<i>Read-checking</i>	<p>Action: Verify consistency of each object in the read-only list</p> <p>Outcomes:</p> <ul style="list-style-type: none"> ■ Abort if change is detected in object held by Undecided transaction ■ If conflict detected with Read-checking transaction: <ul style="list-style-type: none"> □ Help if other transaction precedes current transaction □ Abort if current transaction precedes other transaction
<i>Release</i>	Release each acquired object

Comparison Criteria*

- **Strong or Weak Isolation**
 - Weak isolation: conflicting operation outside of a transaction may not follow the STM protocols
 - Strong isolation: all conflicting operations are (converted to) execute in transactions
- **Transaction Granularity**
 - Word: conflicts detected at word level
 - Block: conflicts detected at block level
- **Direct or Deferred Update**
 - Direct: memory is updated by transaction and restored to original value on abort
 - Deferred: updates are stored privately and applied to memory on commit
 - Update in place: private values copied to memory
 - Cloned replacement: private copy replaces original memory
- **Concurrency control**
 - Pessimistic: conflict is immediately detected and resolved
 - Optimistic: conflict detection and/or resolution deferred
- **Synchronization**
 - Blocking
 - Non-blocking (wait-, lock-, obstruction-freedom)

* *From: Transactional Memory, James Larus and Ravi Rajwar*

Comparison Criteria* (cont.)

- **Conflict Detection**
 - Early: conflicts detected on open/acquire or by explicit validation
 - Late: conflicts detected at time of commit operation
- **Inconsistent reads**
 - Validation: check for updates to memory being read
 - Invalidation: abort reading transaction when update is made
 - Toleration: allow inconsistency (expecting subsequent validation/abort)
- **Conflict resolution**
 - System-defined: help or abort conflicting transactions
 - Application-defined: contention manager resolves conflicts
- **Nested Transactions**
 - Flattened: aborting inner transaction aborts outer transaction - inner transaction only commits when outer transaction commits
 - Not-Flattened: aborting inner transaction does not cause outer transaction to abort
 - Closed: effects of inner transaction not visible to other transaction until outer transaction commits (rollback possible)
 - Open: effects of inner transaction visible to other transaction when inner transaction commits (rollback not possible)
- **Exceptions**
 - Terminate: a commit operation is attempted when an exception occurs in the transaction before propagating the exception
 - Abort: the transaction is aborted

* *From: Transactional Memory, James Larus and Ravi Rajwar*

Comparison

Characteristic	System			
	STM-1	WSTM	DSTM	FSTM
Strong/Weak Isolation	N/A	Weak	Weak	Weak
Granularity	Word	Word	Object	Object
Direct/Deferred Update	Direct	Deferred (update in place)	Deferred (clone replacement)	Deferred (clone replacement)
Concurrency Control	Pessimistic	Optimistic	Optimistic	Optimistic
Synchronization	Lock-free	Obstruction-free	Obstruction-free	Lock-free
Conflict Detection	Early	Late	Early	Late
Inconsistent Reads	None	Toleration	Validation	Validation
Conflict Resolution	helping	Helping/aborting	Contention manager	Abort
Nested Transactions		Flattened	Flattened	closed
Exceptions		terminate		