Committing

# ~~Aborting~~ Conflicting Transactions in an STM

PPoPP'09 2/17/2009

**Hany Ramadan, Indrajit Roy, Emmett Witchel**
*University of Texas at Austin*
**Maurice Herlihy**
*Brown University*

## TM and its discontents

- Contention is a challenge for TM
  - Performance suffers due to aborts/stalls
  - Transactions become serialized, *just like locks* !
- Contention manifests as conflicts
  - Write-sharing (e.g. counters)
  - Structural, not semantic conflicts (e.g. lists)
- Solutions often sacrifice TM advantages
  - Complicate programming model, weaken semantics

| Contention | Locks | TM |
|---|---|---|
| High | ✓ | ~~✗~~ ✓ |
| Low | ✗ | ✓ |

## State Of the Art Mitigates Problem

- Conflict management
  - Karma, Polka, Kindergarten, Eruption, Polite, Timestamp, … [Scherer&Scott07]
- API/programming model
  - Galois [Kulkarni07], Boosting [Herlihy08]
  - ANTs [Harris07]
  - Escape actions [Zilles06], Early release [Skare06]
  - Open nesting [Moss05]
- Isolation
  - TSTM [Adyonat08], SI-STM [Riegel06]

**Dependence-Aware Transactional Memory (DATM): transparency AND performance**

## Example of Conflicts

```
A: atomic
{
   //work

   counter  += 10


}
```

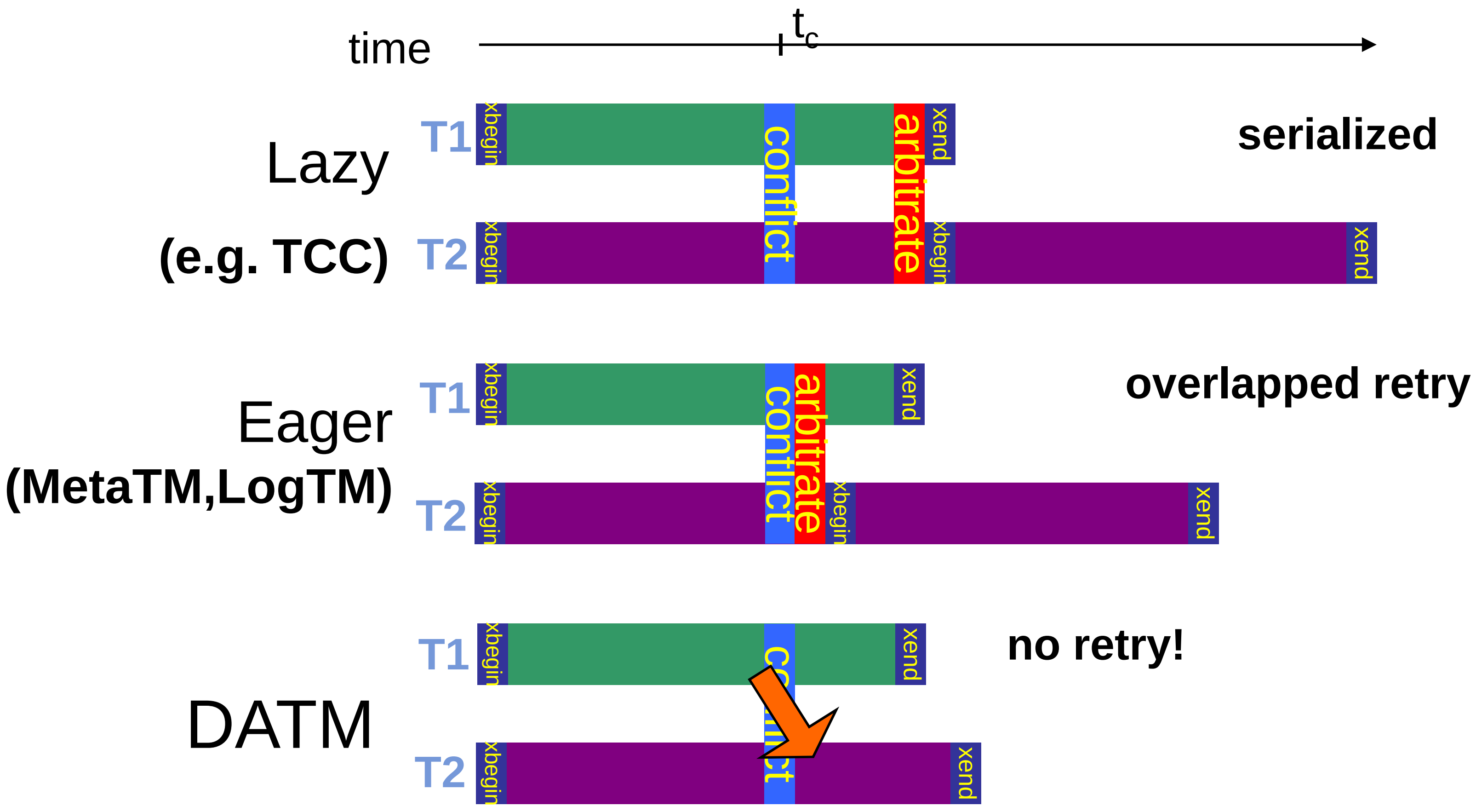
```
B: atomic
{

  // work

  tmp = counter
  counter = tmp + 10
}
```

- Conflicting working sets : $\varnothing \neq \{W_a\} \cap \{R_b \cup W_b\}$
- Our approach:  <u>No API changes</u>
- Concepts clarified:  <u>Isolated != Conflict-free</u>

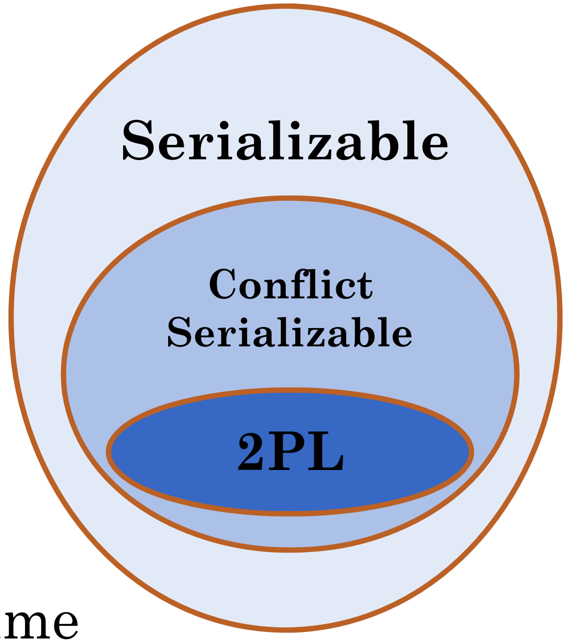
# DATM Improves Performance



*(Lazy/Lazy: Updates buffered, conflict detection at commit time)*

*(Eager/Eager: Updates in-place, conflict detection at time of reference)*

## TM Today is Conservative



- Serializable schedule
  - Equivalent to some serial interleaving
- Conflict-serializability (CS)
  - Order of conflicting operations is the same
- Two phase locking serializability (2PL)
  - Growing phase: growing amount of mutual exclusion
  - Shrinking phase: release mutual exclusion
- 2PL is ubiquitous
  - The DB way: "Deny  all who request <u>my</u> lock"
  - The TM way: "Abort all who request <u>my</u> memory"

## TM Schedules

Not serializable

Serial

2PL serializable

Conflict-serializable

```
A: atomic
{
 //work



 counter += 10



}
```

```
B: atomic
{
 //work

 tmp = counter



 counter = tmp + 10
}
```

Not serializable (Neither equivalent to A,B nor to B,A)

## TM Schedules

Not serializable

Serial

2PL serializable

Conflict-serializable

```
A: atomic
{
 //work


 counter += 10


}
```

```
B: atomic
{
 //work
 tmp = counter
 counter = tmp + 10

}
```

Serial (A,B)

## TM Schedules

Not serializable

Serial

2PL serializable

Conflict-serializable

```
A: atomic
{
 //work


 counter += 10


}
```

```
B: atomic
{
 //work


←can't read counter


← finally allowed
 tmp = counter
 counter = tmp + 10
// more work
}
```

2PL
(equivalent to A,B)

## TM Schedules

Not serializable

Serial

2PL serializable

Conflict-serializable

```
A: atomic
{
 //work


 counter += 10


}
```

```
B: atomic
{
 //work



 tmp = counter
 counter = tmp + 10
}
```

Conflict-serializable
(equivalent to A,B)

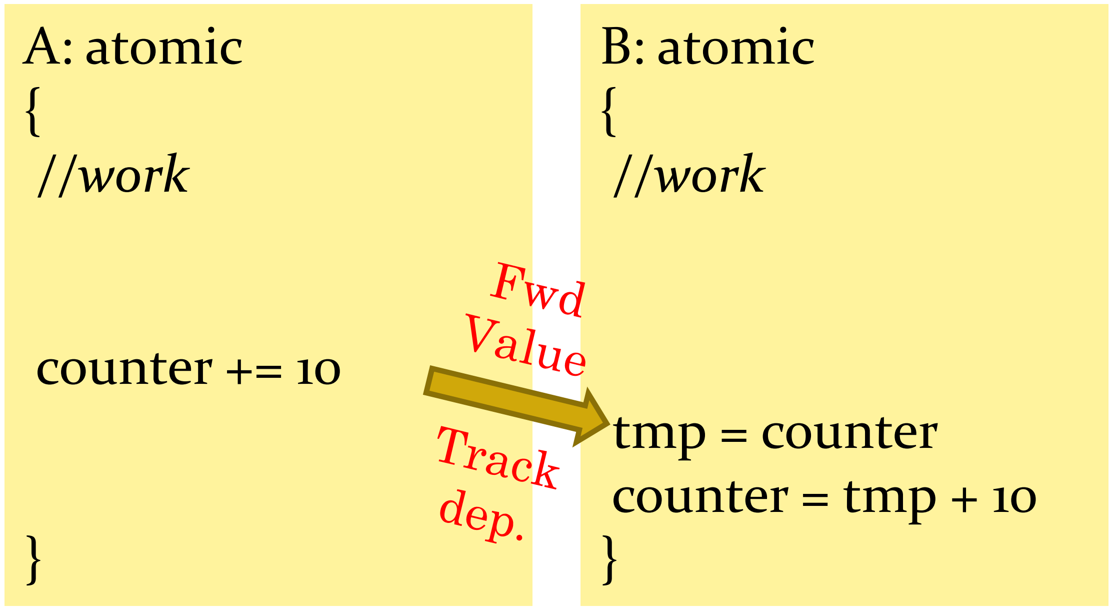
## TM Schedules

Not serializable

Serial

2PL serializable

Conflict-serializable



```
A: atomic
{
 //work


 counter += 10


}
```

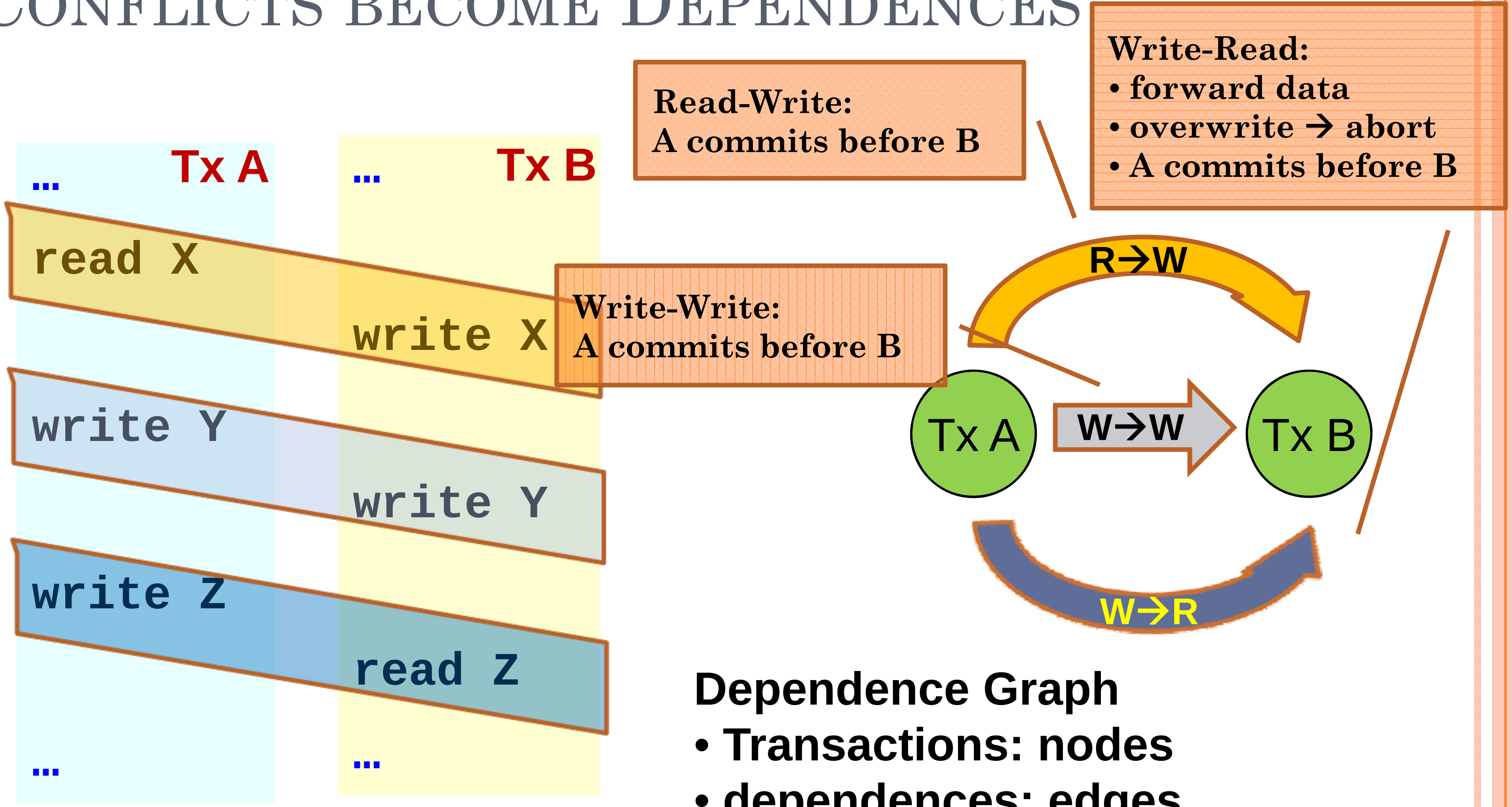
```
B: atomic
{
 //work



 tmp = counter
 counter = tmp + 10
}
```

How DATM is Conflict-serializable (equivalent to A,B)

# Conflicts become Dependences



**Dependence Graph**

- **Transactions: nodes**
- **dependences: edges**
- ***edges dictate commit order***
- ***no cycles → conflict serializable***

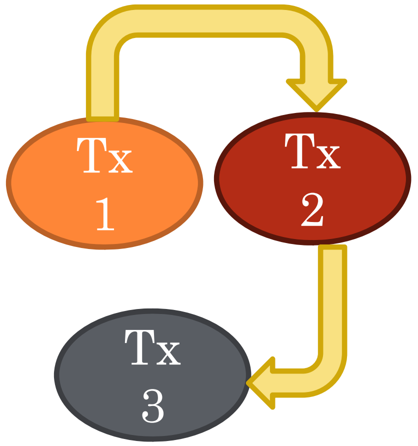
## FORMAL MODEL AND PROOF

- Safety:
  - Accepts *only* conflict-serializable schedules
- Completeness
  - Accepts *all* conflict-serializable schedules
- Key Idea
  - DATM tracks read/write dependences and aborts transactions only if cycles exist in the underlying serialization graph
  - Note: implementations can reject some schedules for variety of reasons (e.g. imprecise deadlock detection)

- *See paper for details…*

## Dependence Requirements

- Enforcing commit order
  - Wait for dependences to resolve
- W→R dependences
  - Forward data
  - Guarantee data is not stale
- Handle cyclic dependences
  - Prevent
  - Detect and abort
  - Timeout
- Tracking multiple versions of data
  - Beyond eager/lazy version management



## DASTM Prototypes

- DASTM-C
  - C, word-based
- DASTM-J
  - Java, object-based
- Global metadata table (a la TL2)
  - Require unique metadata structure per slice
- Metadata contains
  - Multiple versions for each shared item
  - Forward/receive flags

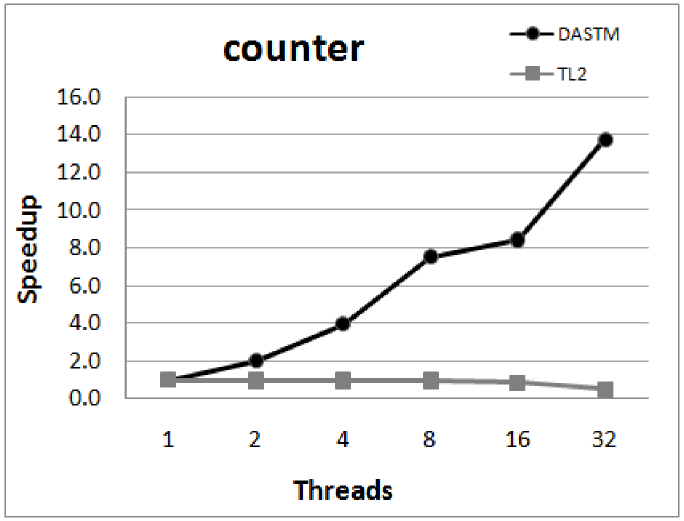
## DASTM PROTOTYPES (2)

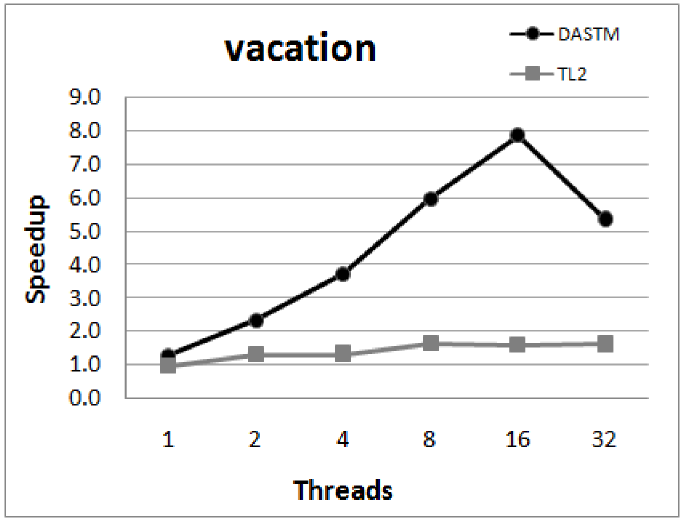
- Optimize read-only data
  - to avoid unnecessary dependences
- Deadlock detection
  - Dreadlocks[Herlihy08] (DASTM-J)
  - Timeout (DASTM-C)
- Vector-clocks to summarize dependences
  - Used only for commit order
- *Zombies are back !*

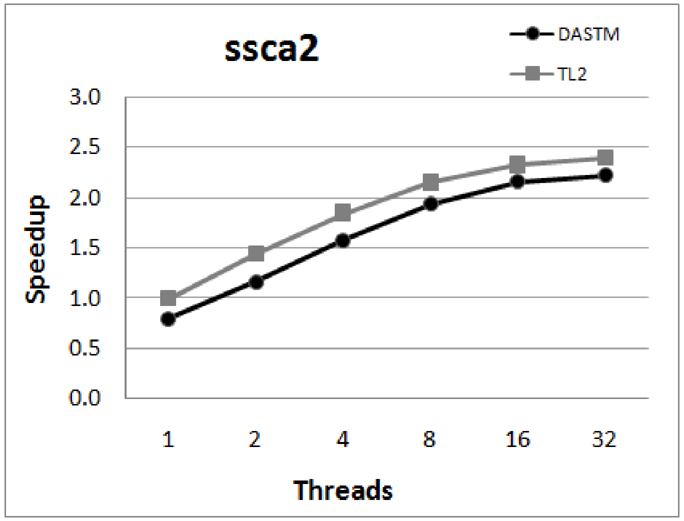
## DASTM: Experimental Setup

- Sun T1 (Niagara) multi-core system
  - 32 processing contexts (8 cores, 4 ctx/core)
  - Memory: private L1 / shared L2
  - Linux 2.6.24
- Compare DASTM-C to TL2
  - Measure speedup, from 1 to 32 threads
- Benchmarks
  - Counter micro-benchmark
    - “think time” to simulate work
  - STAMP benchmarks
    - vacation – large transactions
    - ssca2 – small transactions

# DASTM-C: Results







## Conclusion

- DATM turns conflicts into commits
  - Isolation != Conflict-free
- Transparent to programmer
  - Doesn’t unnecessarily complicate TM API
- DATM is proven safe
- Prototypes demonstrate good performance