Transactional Memory: An Overview

Written by Harris et al.

Goal

- Give an overview of transactional memory
- Implementations will be discussed next week

"Writing applications that benefit from ... multicore chip multiprocessors will not be an easy task for mainstream programmers accustomed to sequential algorithms rather than parallel ones ..."

Tim Harris et al.

Immense Opportunity ...

- Multicore processors
  - exploit thread-level parallelism

If Done Right ...

- Multithreaded programming
  - low level synchronization primitives
    - lock, pessimistic approach to synchronization
    - hard to get right -> deadlock
    - lead to parallel-programming wall

Transactions

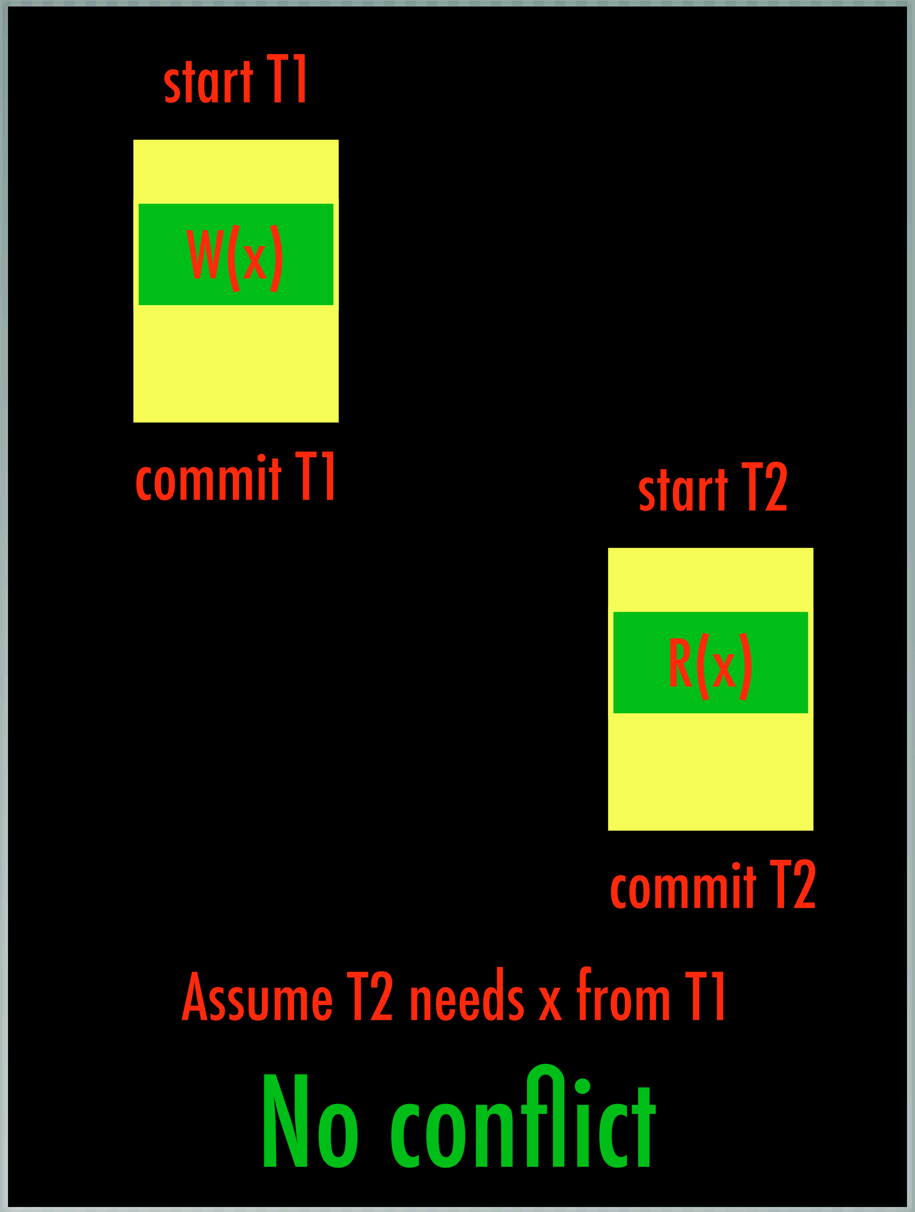
- A sequence of instructions, including reads and write to memory that either executes completely (commit) or has no effects (abort)
  - commit: all writes become visible to other transactions
  - abort: speculative writes are discarded

Transactional Memory

- Abstraction of complexities due to concurrent accesses
  - multiple threads try to access shared data atomically and simultaneously
    - if no conflicts, all accesses within a thread are successful
    - if conflicts, all accesses within a thread are unsuccessful

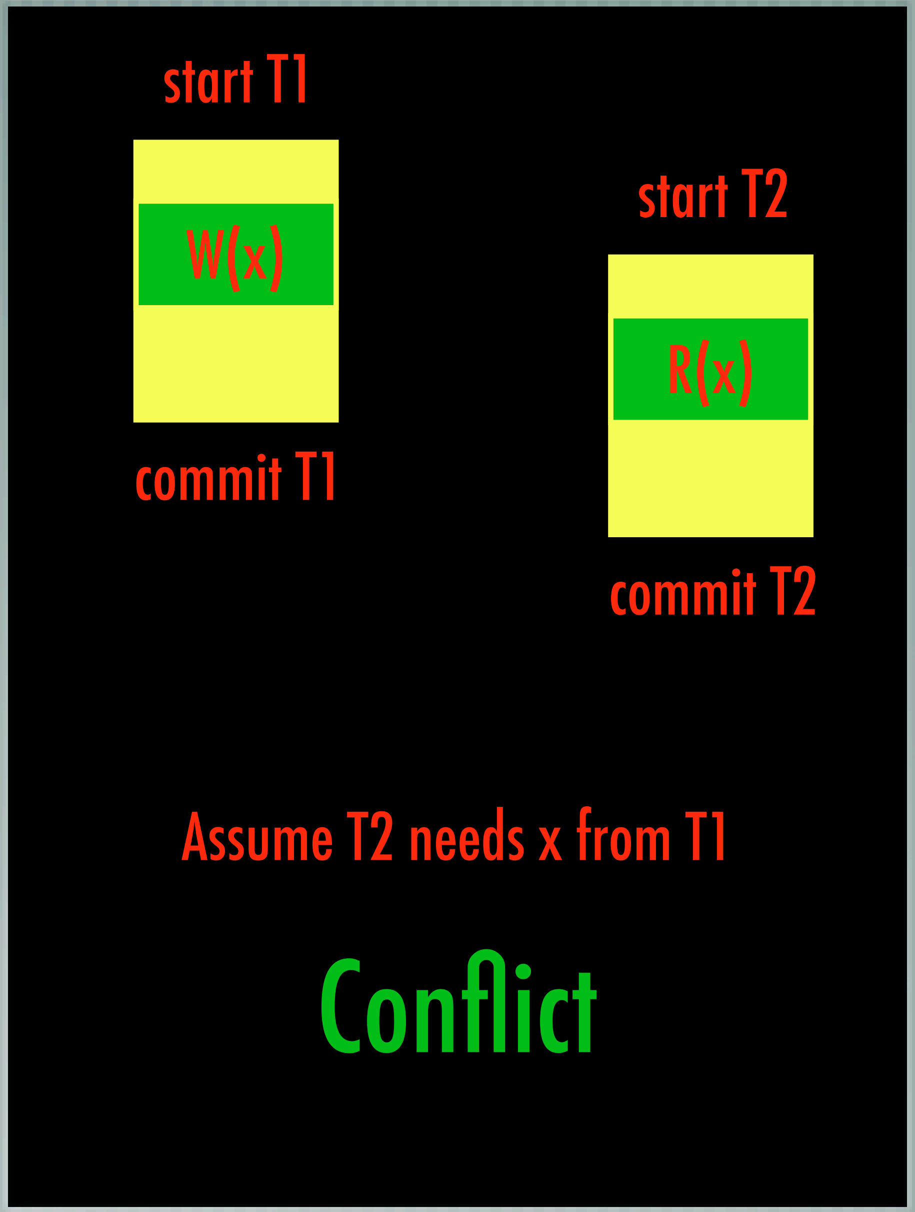
Transactional Memory

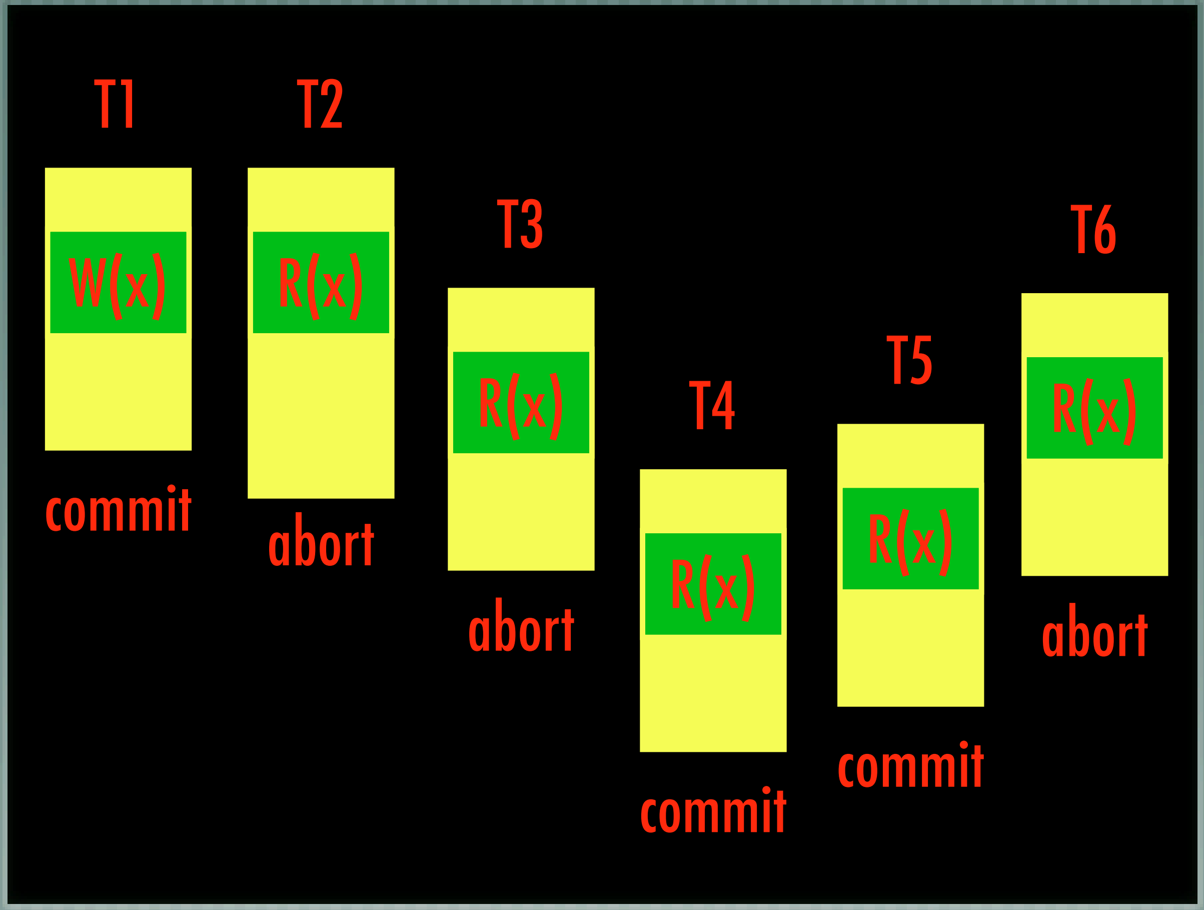
- Defining conflict: violation of a temporal order
  - e.g. read operation from an on-going transactions fails to used the write result from a previous transaction



Transactional Memory

- Defining conflict: violation of a temporal order
  - e.g. read operation from an on-going transactions fails to used the write result from a previous transaction





Transactional Memory

Transactional Memory

- When a conflict occurs
  - abandon the work of conflicting transactions
  - reexecute the abandoned transactions

Transactional Memory

- Two major tasks:
  - conflict detection
  - conflict resolution

Summary: TM versus Lock

- Locking mechanism
  - programmers identify a portion of code that forms a critical section
  - programmers write code that isolates the critical section
- Transactional Memory (TM)
  - programmers identify a portion of code that forms a critical section
  - a runtime system tries to execute the critical section in isolation from other threads

Summary: TM versus Lock

TM

- high-level abstraction
- better scaling/effort
- no deadlock

Lock

- allow fine-grained locking
- better performance
- easily deadlock

Speculative Writes

- Undo log (eager versioning)
  - optimized for rarely occurring conflicts
    - write the the actual memory but record old values for roll-back
- Buffered update (lazy versioning)
  - more straight forward
    - each transaction has its own buffer
    - store write values in the buffer until commit time

Detecting Conflicts

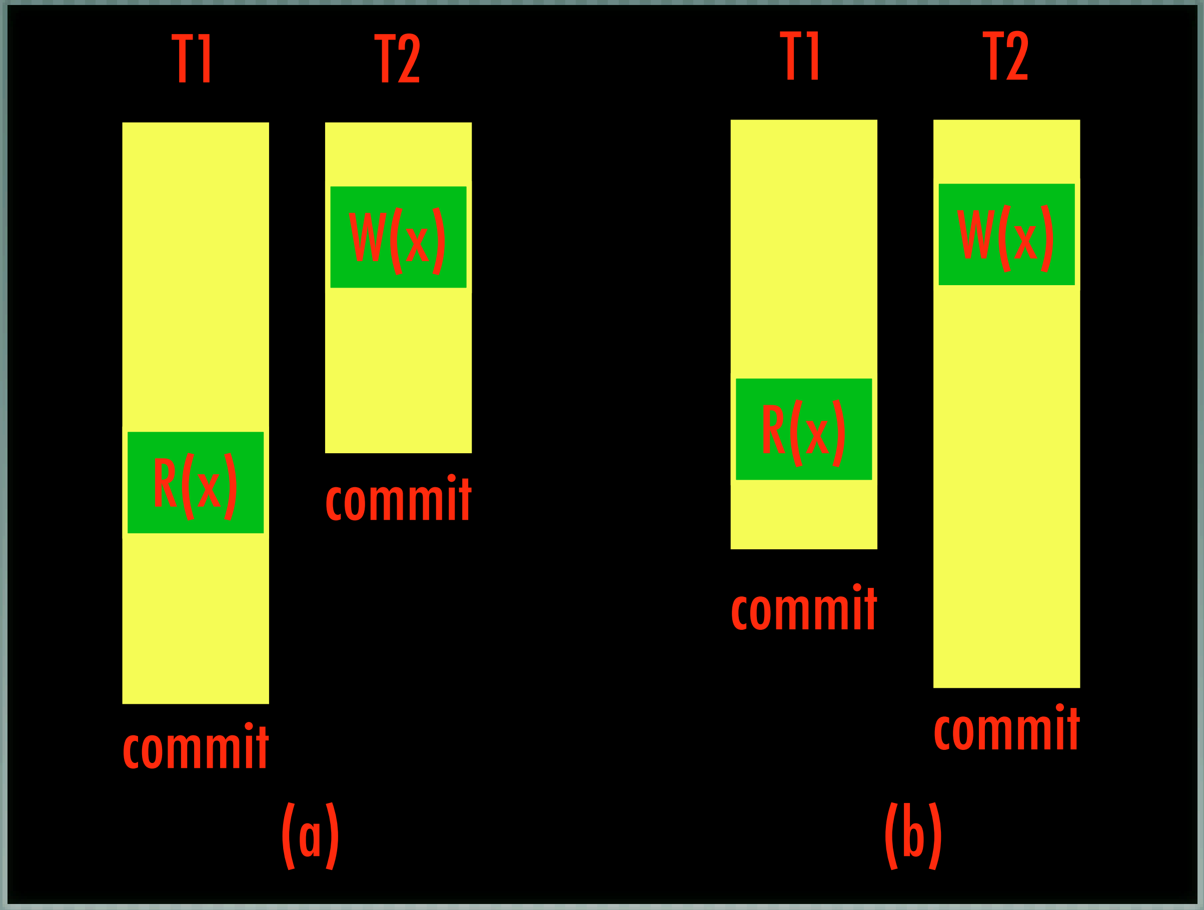
- Conflict occurs when two or more transactions operate concurrently on the same data with at least one transaction writing a new version
- Read-set and write-set
  - inside each transaction, each load is added to the read set and each store is added to the write set

Detecting Conflicts

- Pessimistic detection
  - the read set and write set of every transaction is available to other transaction
    - check every read and write operation to determine conflicts

Detecting Conflicts

- Optimistic detection
  - wait until commit time of a transaction before checking its read and write sets against other transactions' read and write sets
  - does not work with eager versioning



Detecting Conflict

Conflict Resolution

- Stalling in place (applicable to eager versioning)
- Abort mid-transaction (applicable to eager versioning)
- Abort during commit process (applicable to lazy versioning)

Next Weeek

- Implementation of Transactional Memory
  - Software
  - Hardware