Transactional Memory: Architectural Support for Lock-Free Data Structures

Maurice Herlihy (DEC), J. Eliot & B. Moss (UMass)

Presenter: Mariano Diaz

Introduction

- What's a transaction?
- Transaction: a finite sequence of machine instructions, executed by a single process, that satisfies the following:
  - **Serializability/Isolation**
  - **Atomicity**
- Concept "borrowed" from databases..

Introduction con't

- So what's transaction memory?
- Allows for concurrent, lock-free synchronization of shared memory using transactions on a multicore platform.
- Why lock-free?
  - Priority inversion
  - Convoying
  - Deadlock
  - Nasty programming!..

Introduction con't

- Approaches
  - **Hardware (faster, size-limitations, platform dependent)**
  - **Software (slower, unlimited size, platform independent)**
  - **Hybrid (common cases resolved in hardware, exceptions in software)**
    - Object-based (course-grained)
    - Word-based (fine-grained)..

Intended Use

- To replace short critical sections.
- Instructions for accessing memory:
  - **LT (Load-transactional)**
  - **LTX (Load-transactional-exclusive)**
  - **ST (Store-transactional)**
- Instructions to manipulate transaction state:
  - **Commit**
  - **Abort**
  - **Validate**

Intended Use con't

- Typical process:
  - **LT/ LTX to read from locations**
  - **Validate**
  - **ST to modify locations**
  - **Commit**
  - **If fail rinse and repeat**

- When contention is high, use adaptive backoff before retrying.

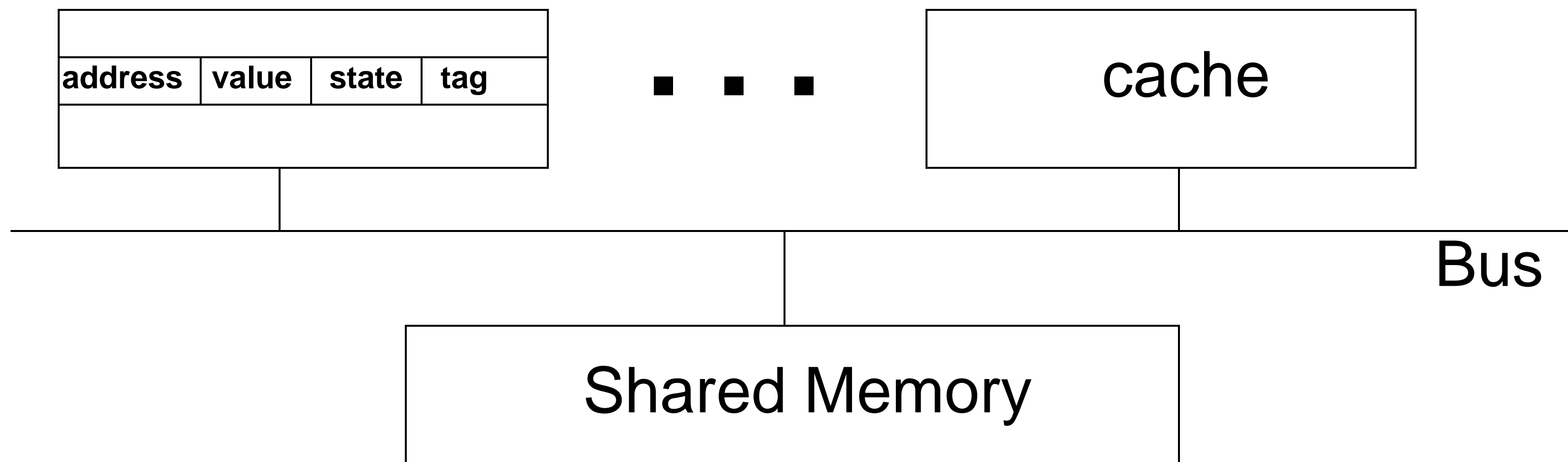
Example

```
shared int counter;

void process(int work)
{
  int success = 0, backoff = BACKOFF_MIN;
  unsigned wait;

  while (success < work) {
    ST(&counter, LTX(&counter) + 1);
    if (COMMIT()) {
      success++;
      backoff = BACKOFF_MIN;
    }
    else {
      wait = random() % (01 << backoff);
      while (wait--);
      if (backoff < BACKOFF_MAX)
        backoff++;
    }
  }
}
```

Figure 1: Counting Benchmark

Cache Implementation

- What hardware are we working with?
- Two types of cache:
  - **Regular**
  - **Transactional**

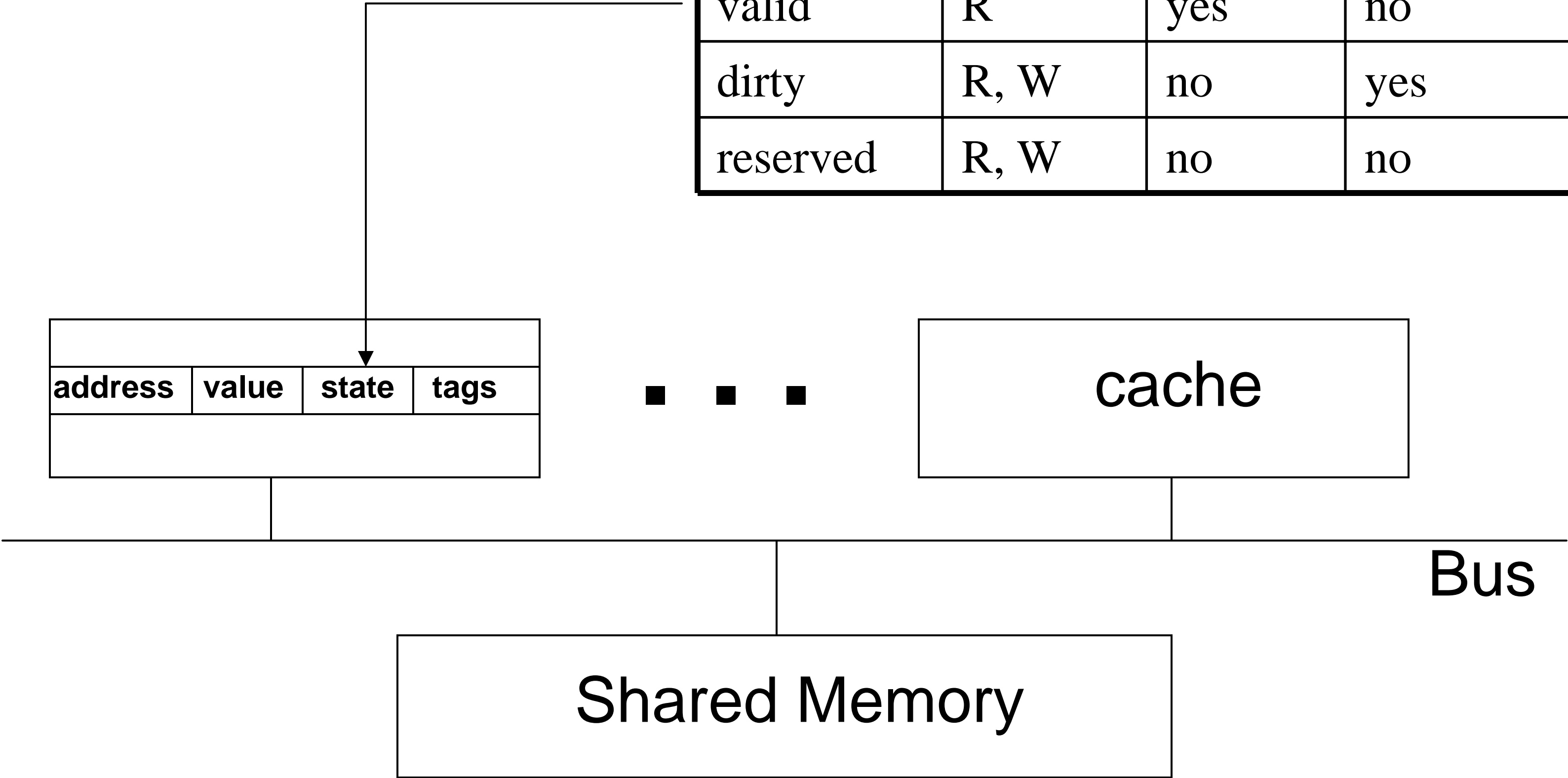


Cache Implementation con't

- Caches "snoop" on shared bus and react via cache coherence protocol. Since already built in, allows for free transaction conflict detection.
- Cache coherence keeps the an address/value pair consistent across a set of caches.
- Cache lines can be in either regular or transactional caches, both not both (same processor)..

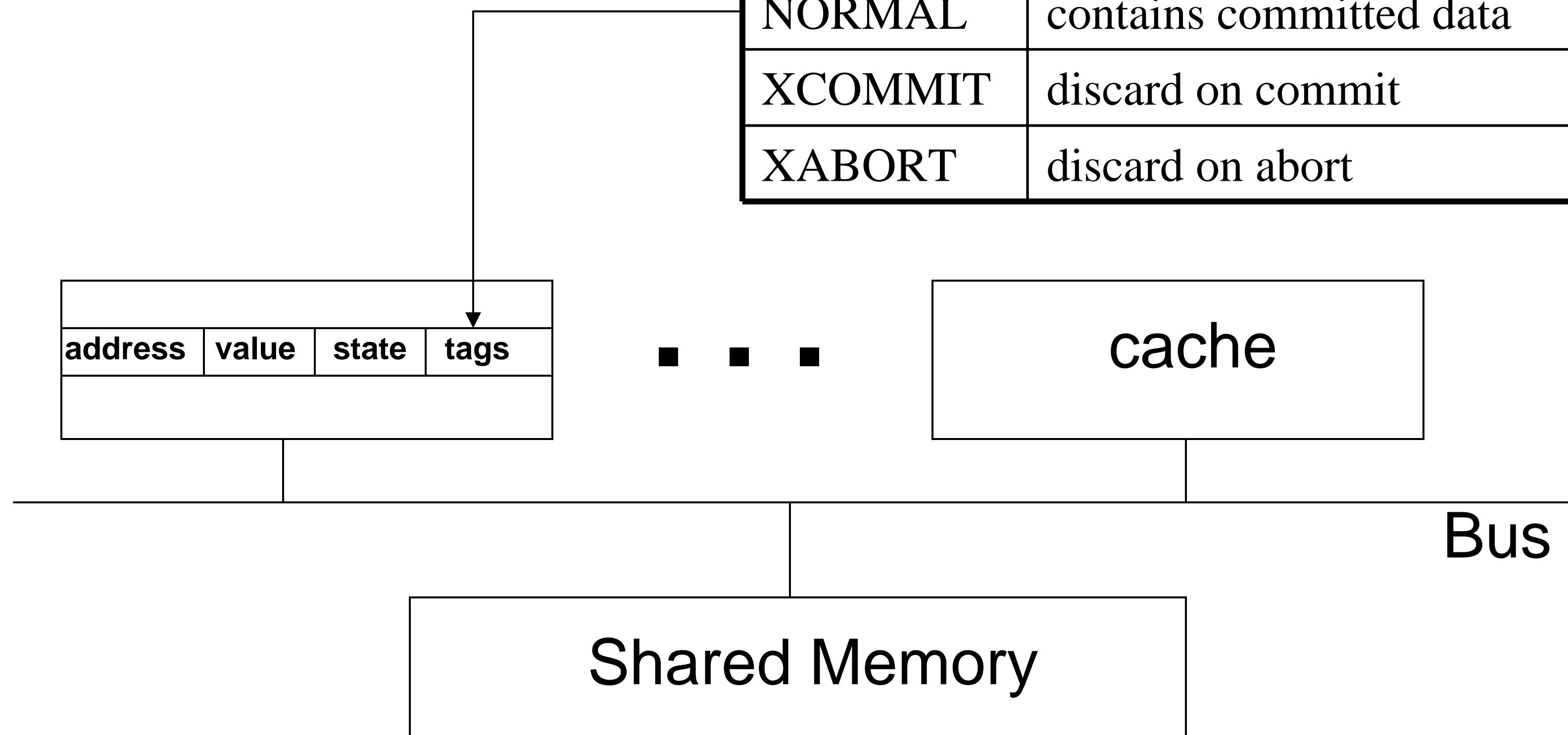
Line States

Name	Access	Shared?	Modified?
invalid	none	---	---
valid	R	yes	no
dirty	R, W	no	yes
reserved	R, W	no	no

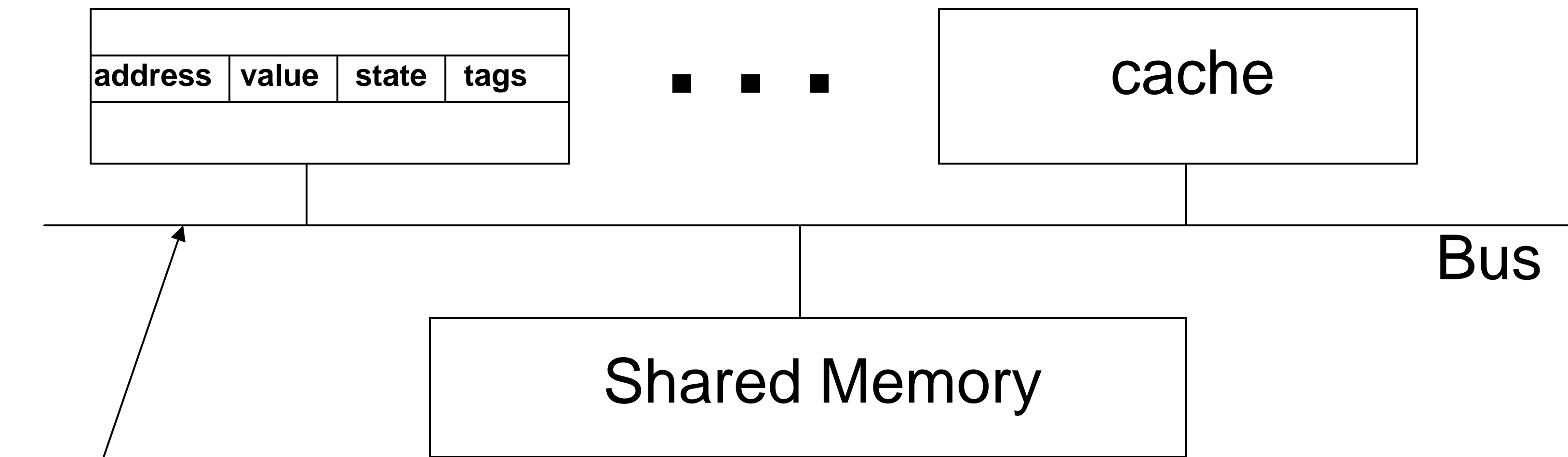


Transactional Tags

Name	Meaning
EMPTY	contains no data
NORMAL	contains committed data
XCOMMIT	discard on commit
XABORT	discard on abort



Bus Cycles



Name	Kind	Meaning	New access
READ	regular	read value	shared
RFO	regular	read value	exclusive
WRITE	both	write back	exclusive
T_READ	transaction	read value	shared
T_RFO	transaction	read value	exclusive
BUSY	transaction	refuse access	unchanged

Processor Flags

- Processor maintains two flags:
  - **Transaction active (TACTIVE)**
  - **Transaction status (TSTATUS)**
- TSTATUS indicates whether the transaction is active (True)
- TACTIVE is set when first transactional operation is executed within transaction.
- Used directly by Commit, Abort, Validate instructions.

Scenarios

- LT instruction
  - **If XABORT entry in transactional cache: return value**
  - **If NORMAL entry**
    - Change NORMAL to XABORT
    - Allocate second entry with XCOMMIT (same data)
    - Return value
  - **Otherwise**
    - Issue T_READ bus cycle
      - **Successful: set up XABORT/XCOMMIT entries**
      - **BUSY: abort transaction**
- LTX instruction
  - **Same as LT instruction except that T_RFO bus cycle is used instead and cache line state is RESERVED**
- ST instruction
  - **Same as LTX except that the XABORT value is updated**

Scenarios & Transaction States

- Validate
  - **Returns TSTATUS flag**
    - *False*: set TACTIVE to *False*
      set TSTATUS to *True*
- Abort
  - **Sets TSTATUS to *True***
  - **Sets TACTIVE to *False***
- Commit
  - **Returns TSTATUS**
  - **Sets TSTATUS to *True***
  - **sets TACTIVE to *False***

Performance (Counting Benchmark)