# Debugging with Transactional Memory

Yossi Lev

*Brown University &*
*Sun Microsystems Laboratories*

Joint work with:

Mark Moir

*Sun Microsystems Laboratories*

# The Problem

- Debuggers *should change*
  - STM: provides "illusion" of multiple locations changing atomically
  - Regular memory accesses breaks the illusion/atomicity
- This talk: *How* to change debuggers
  - Support all common debugging features
  - Leverage STM infrastructure to support advanced debugging features

# Preliminaries

- Programmer Specifies Atomic Blocks
- All Data Access - with transactions
- Concentrate on STM
- Not a specific STM
  - Ownership Tracking Mechanism
    - Read or Write Ownership on a location/object
  - Contention Manager
    - Resolve conflicts to guarantee progress

# Basic Techniques

- Debugger uses Transactions
  - View data, not memory
- "Hijack" Contention Manager
  - For example, don't wait for stalled threads
- Exploit Ownership Tracking Mechanism
  - For example, trigger debugger if ownership is granted/revoked

# Agenda

- Basic Features:
  - Breakpoints & Stepping
  - Viewing and Modifying Data
- Advanced Features:
  - Watchpoints
  - Replay Debugging & Delayed Breakpoints

# Breakpoints and Stepping

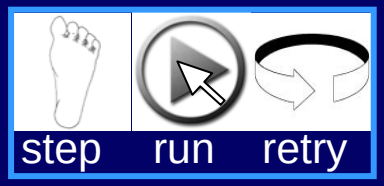
- As with regular code, but:
  - Failures & Retries:
    - Debugged transaction invalidation
      - Notify: No sudden “step backwards”
      - Try to Prevent: “Super Transaction” - Contention Manager makes a transaction win any conflict
    - User-Controlled Abort & Retry (or Skip)

# Agenda

- Basic Features:
  - Breakpoints & Stepping
  - Viewing and Modifying Data
- Advanced Features:
  - Watchpoints
  - Replay Debugging & Delayed Breakpoints

# Viewing and Modifying Shared Data

- Stopped outside of an atomic block: *mini-transaction*
- Stopped inside an atomic block: *Which transaction to use?*




# Example

Code

```
foo (…) {

        ...
        withdraw(checking, 100);
        ...
}


withdraw (Account ac, int amount) {

        atomic {
            if (ac.balance >= amount)
                ac.balance -= amount;
            else
                handleOverdraft();
        }
}
```


# Example

Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```


# Example

## Code

```
foo (…) {
    ...
    withdraw(checking, 100);
    ...
}

withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

# Example


## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

| Others | | |
|---|---|---|
| saving.balance | 1000 | |


# Example

## Code

```
foo (…) {
    ...
    withdraw(checking, 100);
    ...
}

withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

| Others | | |
|---|---|---|
| saving.balance | 1000 | |

# Example


## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

| Others | | |
|---|---|---|
| saving.balance | 1000 | |

# Example


## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

| Others | | |
|---|---|---|
| saving.balance | 1000 | |


# Example

## Code

```
foo (…) {
    ...
    withdraw(checking, 100);
    ...
}

withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |

| Others | | |
|---|---|---|
| saving.balance | 1000 | |


# Example

## Code

```
foo (…) {
    ...
    withdraw(checking, 100);
    ...
}

withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |
| saving.balance | 1000 | 1000 |


# Example

## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 50 |
| saving.balance | 1000 | 1000 |

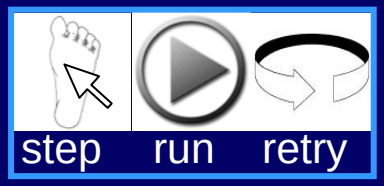

# Example

## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 550 | 50 |
| saving.balance | 500 | 1000 |




# Example

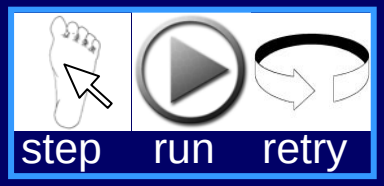
## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 550 | 50 |
| saving.balance | 500 | 1000 |

# Example




## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 550 | 50 |
| saving.balance | 500 | 1000 |


# Example

## Code

```
foo (…) {

    ...
    withdraw(checking, 100);
    ...
}


withdraw (Account ac, int amount) {

    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 450 | 50 |
| saving.balance | 500 | 1000 |

# Viewing and Modifying Shared Data

- Stopped outside of an atomic block: *mini-transaction*
- Stopped inside an atomic block: *Which transaction to use?*
  - For variables accessed by the debugged transaction:
    - Use the debugged transaction to get/modify the value
  - For all others:
    - Use a separate mini-transaction – independent access
    - Or also use the debugged transaction – extends the transaction operation

# Agenda

- Basic Features:
  - Breakpoints & Stepping
  - Viewing and Modifying Data
- Advanced Features:
  - Watchpoints
  - Replay Debugging & Delayed Breakpoints

# Watchpoints

- Stop when a variable is accessed
  - Today: Limited in hardware, Big overhead in software.

- With STM: Watch for ownership acquisition
  - can even stop on a read access
- Also:
  - Conditional Watchpoint
    *even on multiple variables (more complicated)*
  - Dynamic Watchpoints
    *address of the watched variable might change*
    *for example: first value in a linked list*

# Agenda

- Basic Features:
  - Breakpoints & Stepping
  - Viewing and Modifying Data
- Advanced Features:
  - Watchpoints
  - Replay Debugging & Delayed Breakpoints

# Replay Debugging

- The ability to replay an execution of a program
  *"how did I get here?"*
- Prior work required either hardware support, or induced a big overhead in software

- Use STM infrastructure to replay a transaction:
  - Only needs to know the result of each read executed.
  - Transaction is atomic → Result of a read is either the pre-transactional value or a value written by *that* transaction.

⇒ All we need are the pre-transactional values!
*Most STM can give it to us with no additional cost*

# Replay Debugging

- What is it good for:
  - How control reached a breakpoint in an atomic block
  - The "What If" Game:
    *Change variables accessed by the replayed execution,*
    *See how the execution changes*
  - Commit the modified execution
    *done by implicitly aborting the replayed transaction*
    *and beginning a new one*

# Delayed Breakpoints

- Do not stop an atomic-block's execution:
  - Place a breakpoint inside the atomic block
  - Stop after the block is executed, if the breakpoint was hit

  Motivation: more natural run
  *smaller effect on interleaving time, stop only on committed executions*

- Stop-On-Commit: Invoke debugger when:
  - transaction is guaranteed to commit successfully
  - no one has yet seen its effect.

⇒ User can still abort and skip/retry the atomic-block

- With replay debugging: Use delayed, experience regular

# Example


Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

# Example


Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 1000 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 900 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 1000 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 1000 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 1000 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 1000 |

# Example


## Code

```
atomic {
    ...
    withdraw(...,...);
    ...
}


withdraw (Account ac, int amount) {
    atomic {
        if (ac.balance >= amount)
            ac.balance -= amount;
        else
            handleOverdraft();
    }
}
```

D

## View

| My Transaction | | |
|---|---|---|
| amount | 100 | 100 |
| ac.balance | 50 | 1000 |

# What about HTM?

- Hybrid Transactional Memory (HyTM) [ASPLOS 06]
  *[Damron, Fedorova, Lev, Luchangco, Moir, Nussbaum]*
  - Try with HTM, revert to STM if fails
    *two execution paths*
  - Works today (no HTM),
    Works better tomorrow (with best-effort HTM)
- Debugging with HyTM (in the paper):
  - Do not assume any debugging support in hardware
  - For each feature:
    which atomic blocks should be forced to be executed using STM ?

# Summary

- First work on Debugging with TM
  - Debuggers change to support *all* standard debugging.
  - Leverage TM infrastructure to support enhanced debugging.