Detecting Data Races in Multi-Threaded Programs

Eraser
A Dynamic Data-Race Detector
for Multi-Threaded Programs

MultiRace
An Efficient On-the-Fly Data-Race Detection Tool
for Multi-Threaded C++ Programs

John C. Linford

Key Points

1. Data races are easy to cause and hard to debug.

2. We can't detect all data races.

3. Detection of <u>feasible</u> races relies on detection of <u>apparent</u> data races.

4. Data race detection tools are either <u>static</u> or <u>dynamic</u> *(on-the-fly* and *postmortem)*.

Key Points Cont.

5. Data races can be prevented by following a locking discipline.

6. Commonly used detection algorithms are **Lockset** and **DJIT** (Happens-Before).

7. Lockset maintains a set of candidate locks for each shared memory location. If a shared location is accessed when this set is empty, there has been a violation of the locking discipline.

Key Points Cont.

8. Lockset is vulnerable to false alarms.

9. DJIT uses a logging mechanism. Every shared memory access is logged to see that it “happens before” prior accesses to the same location.

10. DJIT is dependent on the scheduler and thread interleaving.

11. Combining happens-before with Lockset can improve detection accuracy.

Data Race Review

Two threads access a shared variable

- At least one access is a write,
- Simultaneous access is not prevented.

- Example (variable X is global and shared)

Execution

Thread 1	Thread 2
X = 2.7	X = 3.1
Z = 2	T = X

Data Race Demonstration

- Data races often lead to unexpected and even nondeterministic behavior
- The outcome may be dependent on specific execution order (threads' interleaving)
- Click image to start

Data Race Demonstration Cont.

```
int[] shared = new int[1];
Thread t1, t2;
public DataRace() {
  // Initialize and start threads (shown below)
}
```

```
t1 = new Thread() {
  public void run() {
    while(t1 != null) {
      ...
      shared[0] = shared[0] + 1;
      ...
      }
    ...
```

```
t2 = new Thread() {
  public void run() {
    while(t2 != null) {
      ...
      shared[0] = shared[0] + 1;
      ...
      }
    ...
```

We Can't Detect All Data Races

- For *t* threads of *n* instructions each, the number of possible orders is about t^{n*t}.
- All possible inputs would have to be tested.
- Adding detection code or debugging information can change the execution schedule.

[Pozniansky & Schuster, 2003]

Feasible Data Races

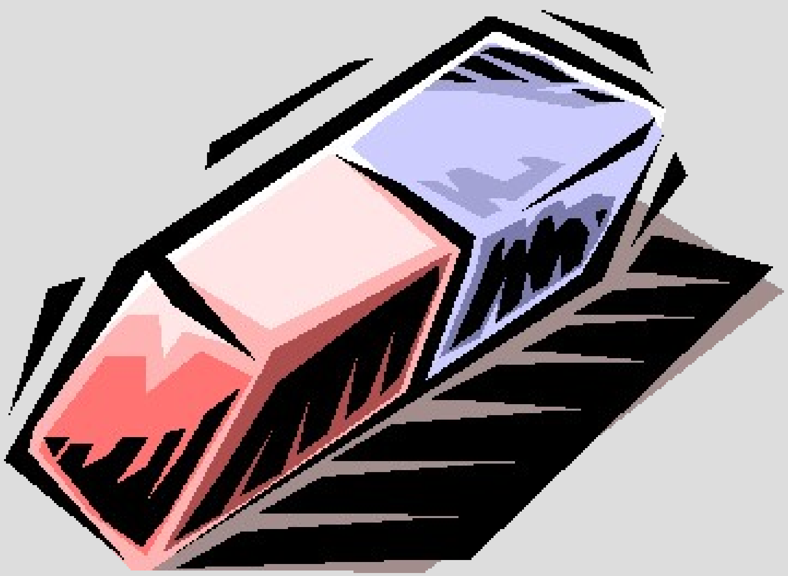
- Races based on possible behavior of the program.
- Actual data races which could manifest in any execution.
- Locating feasible races requires a full analysis of the program's semantics.
- Exactly locating feasible races is NP-hard [Pozniansky & Schuster, 2003].

Apparent Data Races

- Approximations of feasible data races based on synchronization behavior in an execution.
- Easier to detect, but less accurate.
- Apparent races exist if and only if at least one feasible race exists.
- Locating <u>all</u> apparent races is NP-hard [Pozniansky & Schuster, 2003].

Eraser [Savage, Burrows, et al., 1997]

- On-the-fly tool.
- Lockset algorithm.
- Code annotations to flag special cases.
- Can be extended to handle other locking mechanisms (IRQs).
- Used in industry.
- Slows applications by a factor of 10 – 30.



The Lockset Algorithm (Simple Form)

- Detects races not manifested in one execution.
- Generates false alarms.

- *Let **locks_held(t)*** be the set of locks held by thread t
- For each shared memory location ***v***, initialize ***C(v)*** to the set of all locks
- On each access to ***v*** by thread ***t***,
  - Set ***C(v)*** := ***C(v)*** ∩ ***locks_held(t)***
  - If ***C(v)*** := {}, then issue a warning

Lockset Refinement

Lockset Refinement Example

Program	***locks_held***	***C(v)***
int v;	{}	{mu1, mu2}
v := 1024;		
lock(mu1);		
	{mu1}	
v := v + 1;		
		{mu1}
unlock(mu1);		
	{}	
lock(mu2);		
	{mu2}	
v := v + 1;		
		{}
unlock(mu2);		
	{}	

Warning!

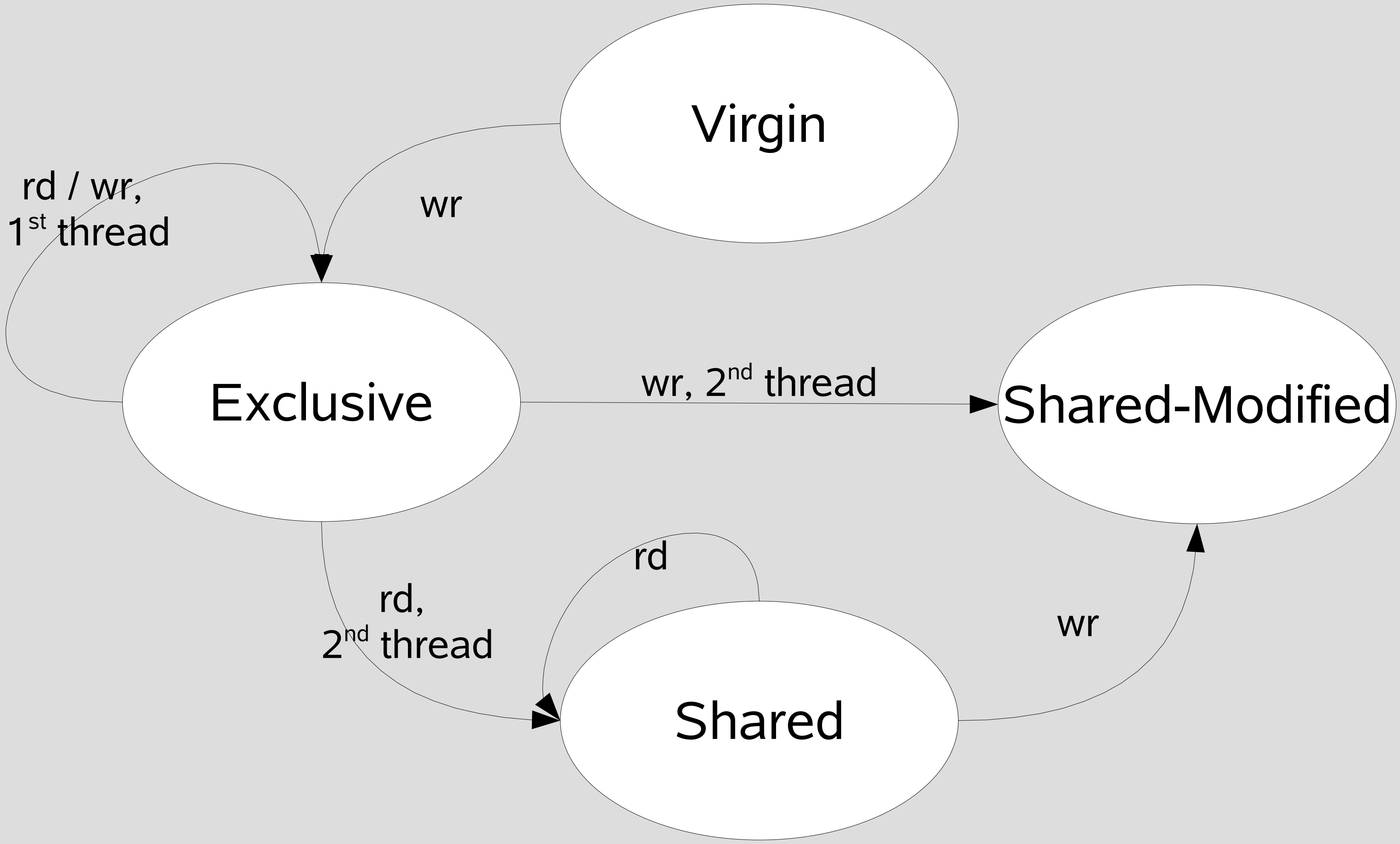
Simple Lockset is too Strict

Lockset will produce false-positives for:

- Variables initialized without locks held.
- Read-shared data read without locks held.
- Read-write locking mechanisms (producer / consumer).

Lockset State Diagram

Warnings are issued only in the Shared-Modified state



Lockset State Example

	Program	***locks_held***	***C(v)***	***State(v)***
	int v;	{}	{mu1, mu2}	Virgin
T1	v := 1024;			
				Exclusive
T2	lock(mu1);			
		{mu1}		
	v := v + 1;			Shared Shared-Modified
	unlock(mu1);		{mu1}	
		{}		
T1	lock(mu2);			
		{mu2}		
	v := v + 1;			
			{}	
	unlock(mu2);			
		{}		

Race detected correctly

The Lockset Algorithm (Extended)

- Let ***locks_held(t)*** be the set of locks held in any mode by thread ***t***
- Let ***write_locks_held(t)*** be the set of locks held in write mode by thread ***t***
- For each shared memory location ***v***, initialize ***C(v)*** to the set of all locks
- On each read of ***v*** by thread ***t***,
  - Set ***C(v)*** := ***C(v)*** ∩ ***locks_held(t)***
  - If ***C(v)*** = {}, then issue a warning
- On each write of *v* by thread *t*,
  - Set ***C(v)*** := ***C(v)*** ∩ ***write_locks_held(t)***
  - If ***C(v)*** = {}, then issue a warning

Unhandled Cases in Eraser

- Memory reuse
- Unrecognized thread API
- Initialization in different thread
- Benign races

```
if(fptr == NULL) {
   lock(fptr_mu);
   if(fptr == NULL) {
         fptr = open(filename);
   }
   unlock(fptr_mu);
}
```

Unhandled Cases in Eraser Cont.

- Race on ★ and ★★ will be missed if ★★ executes first

```
int[] shared = new int[1];
Thread t = new Thread() {
  public void run() {
    shared = shared + 1;
    ...
};
...
shared = 512;
t.start();
shared = shared + 256;
...
```

[Seragiotto, 2005]

Unhandled Cases in Eraser Cont.

Program	***locks_held***	***C(v)***	***State(shared)***
int[] shared = new int[1];	{}	{mu1}	Virgin
shared = 512;			
			Exclusive
t.start();			
★★ shared = shared + 256;			
Thread t = new Thread() {			
public void run() {			
★ shared = shared + 1;			Shared
...			Shared-Modified
};		{}	
...			

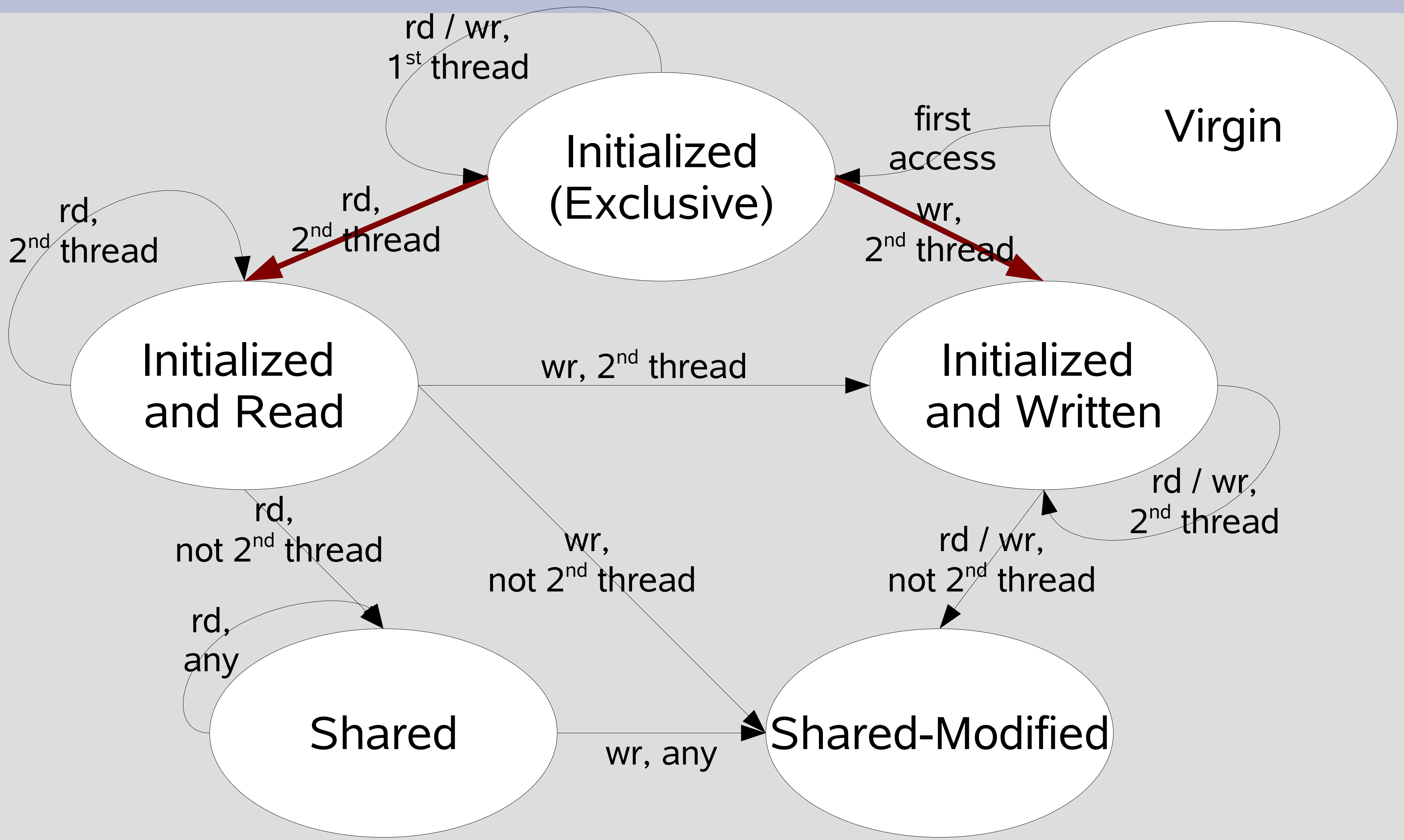
Data race is not detected!

Unhandled Cases in Eraser Cont.

	Program	*locks_held*	*C(v)*	*State(shared)*
	int[] shared = new int[1];	{}	{mu1}	Virgin
	shared = 512;			
				Exclusive
	t.start();			
	Thread t = new Thread() {			
	public void run() {			
★	shared = shared + 1;			Shared
	...			Shared-Modified
	};		{}	
★★	shared = shared + 256;			

Data race is detected!

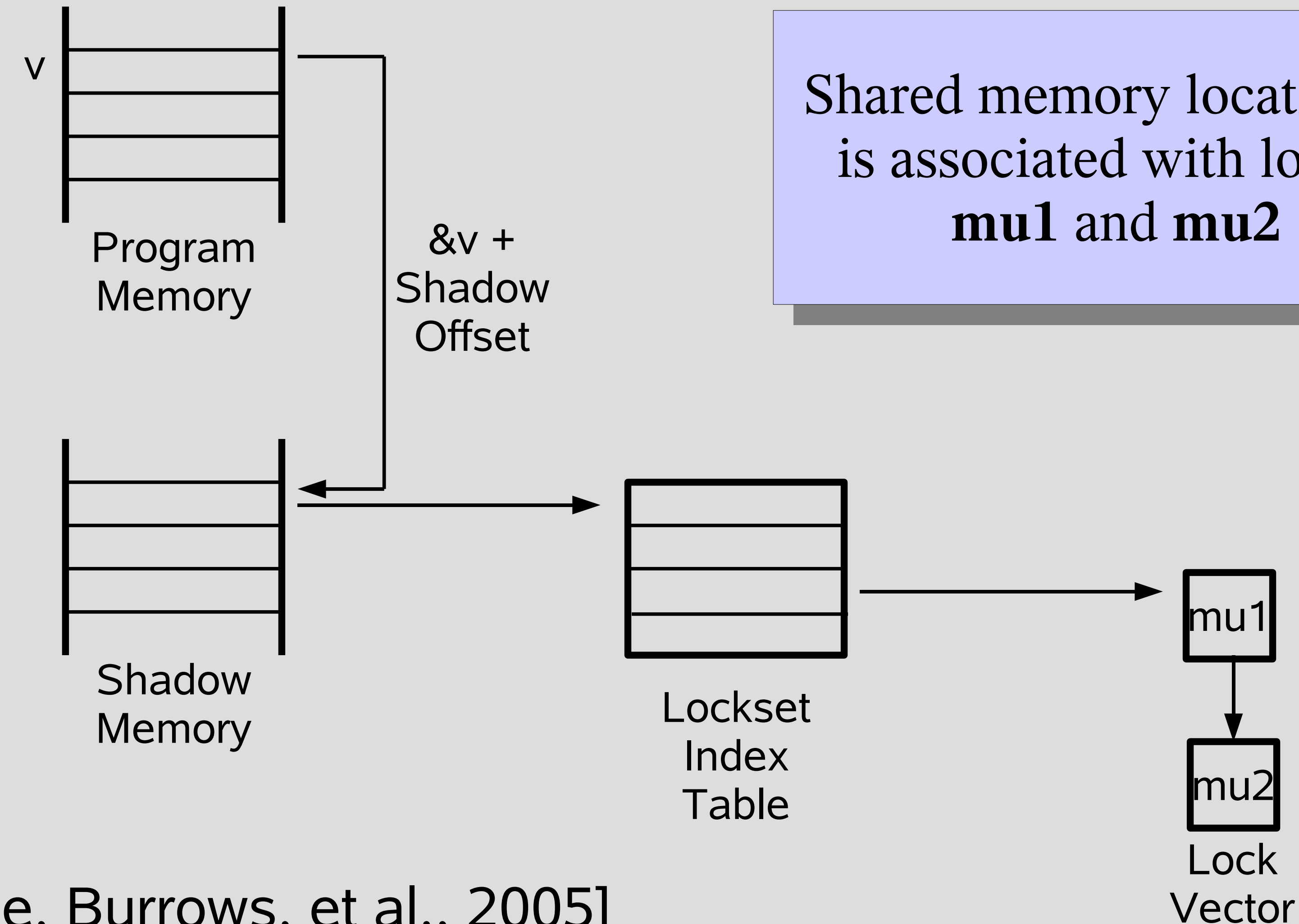
Improved Lockset State Diagram [Seragiotto, 2005]



Implementations: Eraser

- Maintains hash table of sets of locks.
- Represents each set of locks with an index.
- Every shared memory location has shadow memory containing lockset index and state.
- Shadow memory is located by adding offset to shared memory location address.

Implementations: Eraser



Shared memory location **v** is associated with locks **mu1** and **mu2**

[Savage, Burrows, et al., 2005]

Implementations: Ladybug [Seragiotto, 2005]

- GC Eraser:
  - Maintains lock list for threads <u>and</u> variables.
  - Uses weak references (less memory usage).
- Fast Eraser:
  - Maintains lock list for threads <u>and</u> variables.
  - Uses strong references (faster).
- Vanilla Eraser:
  - Same as eraser, but keeps hash table of lock sets already created.

Ladybug Demonstration

- Rewrite class file
  - `java -cp Ladybug.jar br.ime.usp.ladybug.LadybugClassRewriter DataRace.class`
- Run modified class
  - `java -cp Ladybug.jar:. DataRace`
- Races reported as exceptions

```
br.ime.usp.ladybug.RCException: [line 9]
Race condition detected: t2 of DataRace (hash code = 1b67f74) with Thread-0
        at br.ime.usp.ladybug.StaticLadybug.warn(StaticLadybug.java:1014)
        at br.ime.usp.ladybug.eraser.EraserGC.writeField(EraserGC.java:47)
        ...
        at DataRace.access$202(DataRace.java:9)
        at DataRace$1.run(DataRace.java:37)
```

- Can also use GUI

MultiRace [Pozniansky & Schuster, 2003]

- On-the-fly tool.
- Improved Lockset and DJIT+.
- Significantly fewer false alarms than Eraser.
- Minimal impact on program speed.

DJIT

- Based on Lamport's Happens-Before relationship.
- Detects the first apparent data race when it actually occurs.
- Can be extended to detect races after the first (DJIT+).
- Dependent on scheduling order.

Benefits of Combining Lockset and DJIT

- Races are in the intersection of warnings.
- Lockset's insensitivity compensates for DJIT's sensitivity to thread interleaving.
- Lockset reduces DJIT execution overhead.
- Lockset warnings are “ranked” by DJIT.
- Implementation overhead is minimized.

Conclusion

1. Data races are easy to cause and hard to debug.
2. Data race detection tools are either <u>static</u> or <u>dynamic</u> *(on-the-fly* and *postmortem)*.
3. Commonly used detection algorithms are **Lockset** and **DJIT** (Happens-Before).
4. Lockset is vulnerable to false alarms.
5. DJIT is dependent on the scheduler and thread interleaving.
6. Combining happens-before with Lockset can improve detection accuracy.

References

- S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T.E. Anderson. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. In *ACM Transactions on Computer Systems*, 15(4): pp. 391-411, 1997.
- E. Pozniansky and A. Schuster. Dynamic Data-Race Detection in Lock-Based Multi-Threaded Programs. In *Principles and Practice of Parallel Programming*, pp. 170-190, 2003.
- E. Pozniansky and A. Schuster. *MultiRace: Efficient Data Race Detection Tool for Multithreaded C++ Programs*. 2005. http://dsl.cs.technion.ac.il/projects/multirace/MultiRace.htm.
- C. Seragiotto. *Ladybug: Race Condition Detection in Java. 2005*. http://www.par.univie.ac.at/~clovis/ladybug/