Eraser: A dynamic data race detector for multithreaded programs

S. Savage, M. Burrows,
G. Nelson, P. Sobalvarro, and
T. Anderson

TOCS Nov. 97

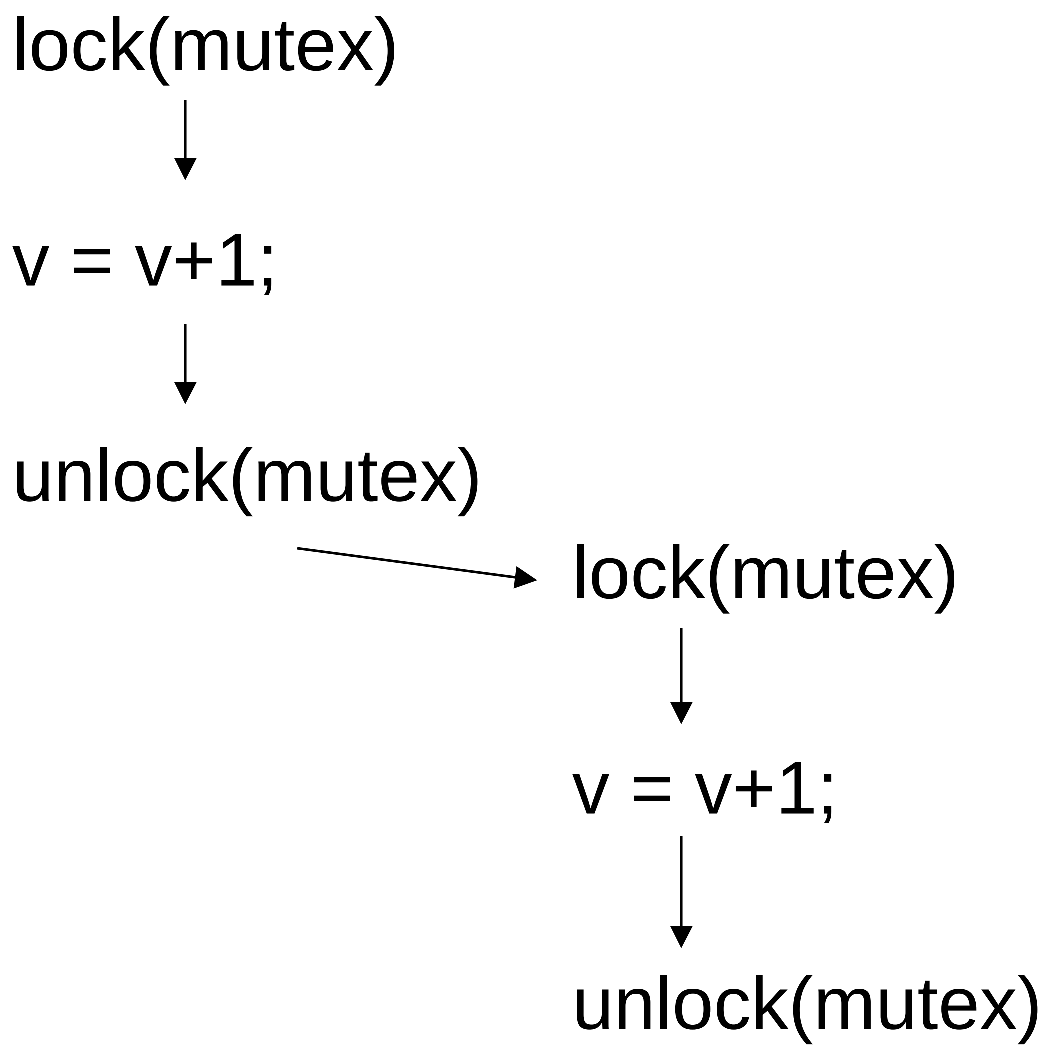
Overview

- Dynamic data race detection tool – testing paradigm instead of static analysis.
- Checks that each shared memory access follows a consistent *locking discipline*
- *Data race* – when 2 concurrent threads access a shared variable and at least one is a write and the threads use no explicit synchronization to prevent simultaneous access.
  - Effect will depend on interleaving

Previous Approaches: Lamport's Happened-Before

Previous work

- If 2 threads access a shared variable and the accesses are not ordered by *happens-before* then potential race.

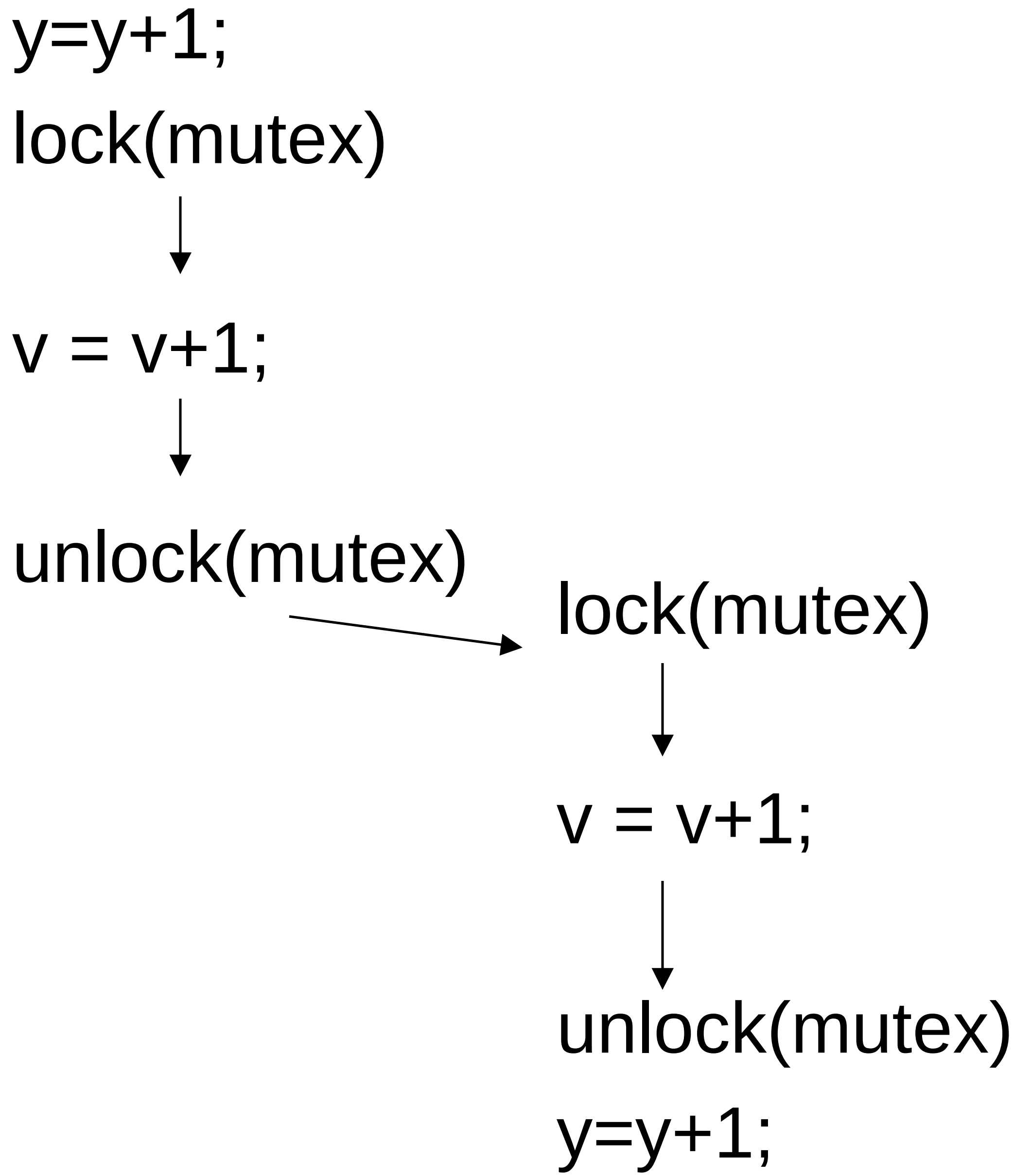


Drawbacks of Happened-Before

- Difficult to implement efficiently – need per-thread information about access ordering to all shared memory locations.
- Highly dependent on scheduler – needs large number of test cases.

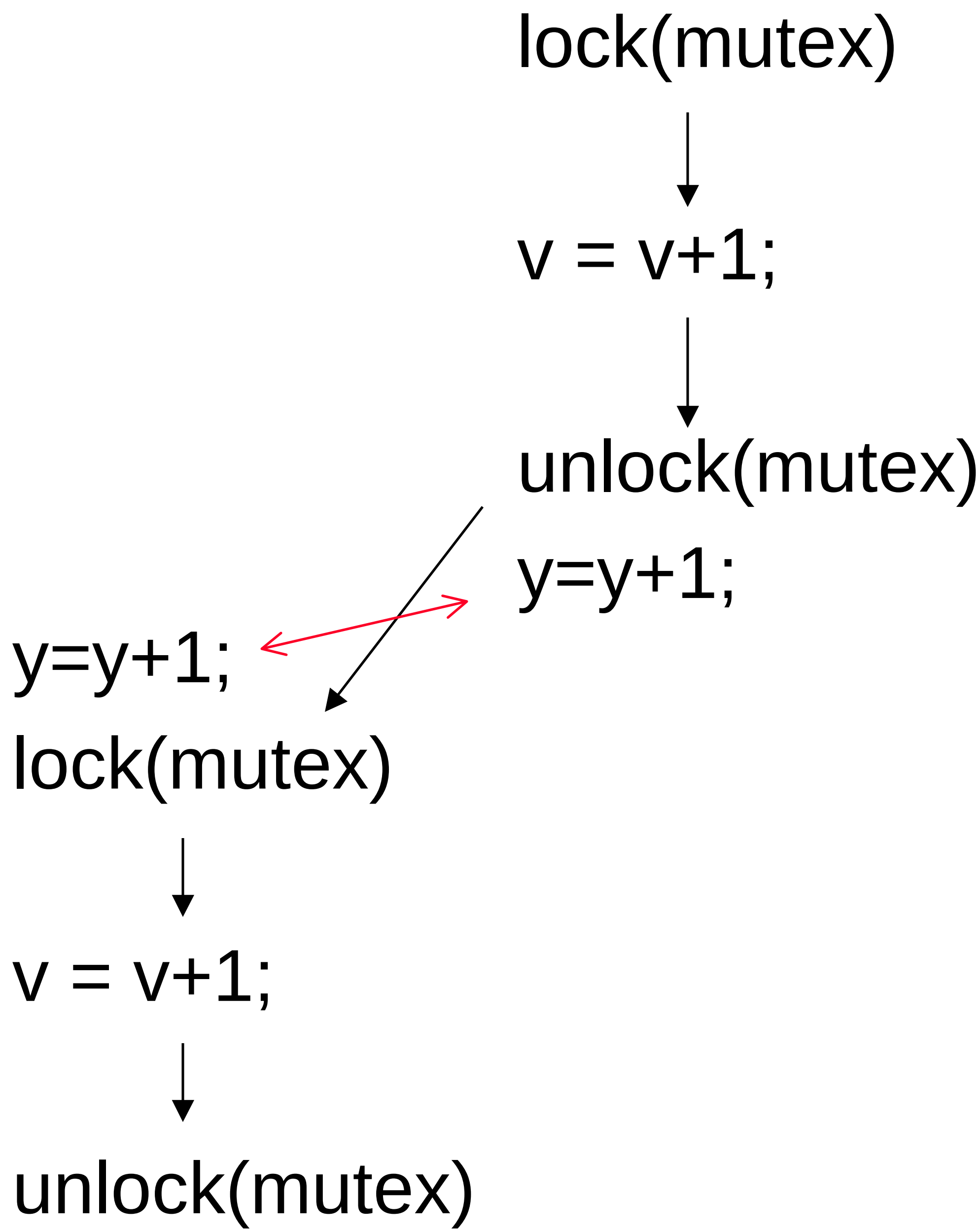
Previous work

- If 2 threads access a shared variable and the accesses are not ordered by *happens-before* then potential race.
- Depends on scheduler



Previous work

- If 2 threads access a shared variable and the accesses are not ordered by *happens-before* then potential race.
- Depends on scheduler



Idea in Eraser

- Checks that locking discipline is observed.
  - That the same lock(s) is held whenever the shared data object is accessed.
  - Infer which locks protect which data items

Lockset Algorithm

- C(v) – candidate locks for v
- locks-held(t) – set of locks held by thread t
- Lock refinement

for each v, init C(v) to set of all locks

On each access to v by thread t:

C(v) = C(v) ∩ locks-held(t)

If C(v) = {} issue warning

Example

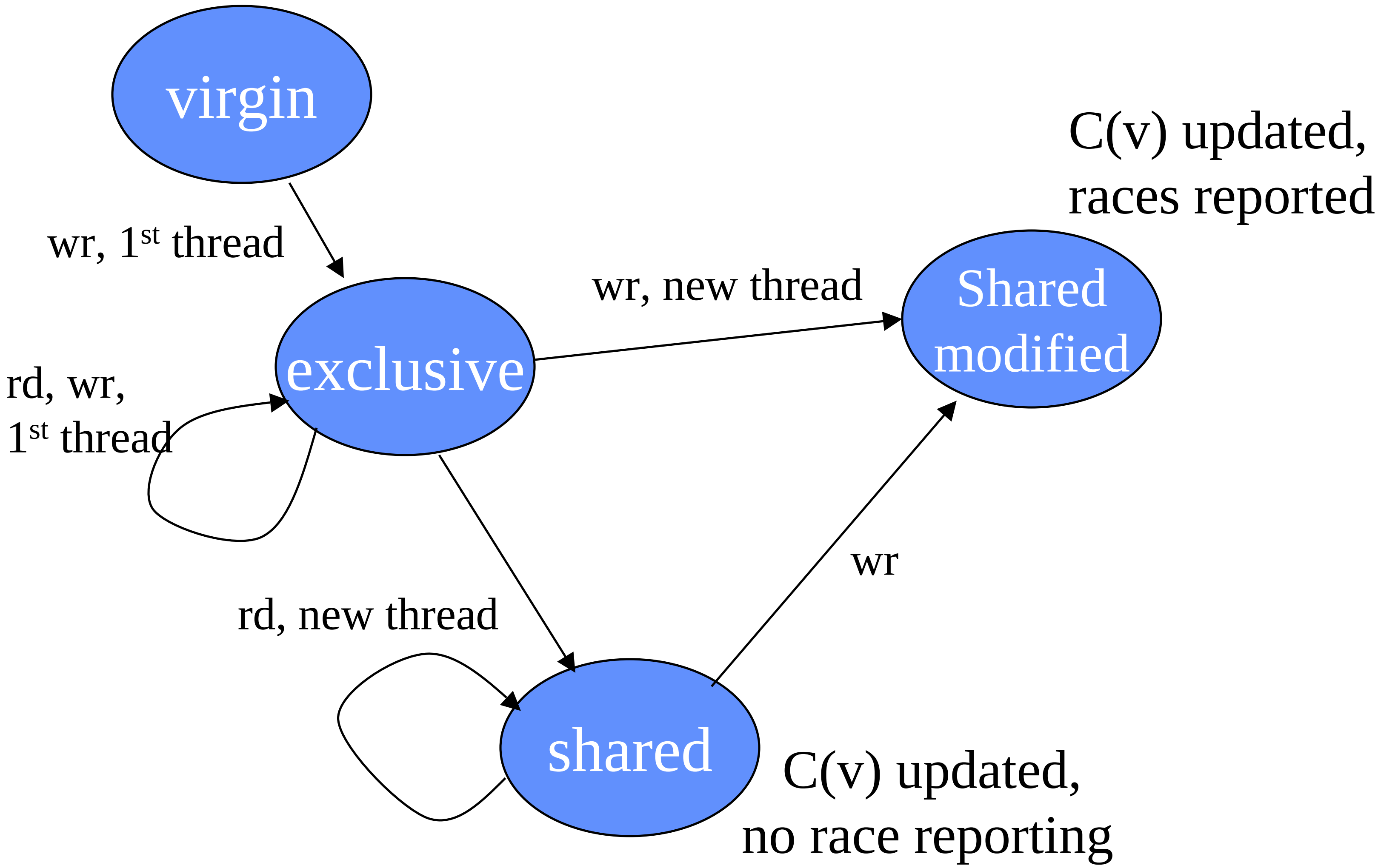
	locks-held	C(v)
	{}	{mu1, mu2}
lock(mu1)	{mu1}	
v=v+1		{mu1}
unlock(mu1)	{}	
lock(mu2)	{mu2}	
v=v+1		{}
unlock(mu2)	{}	

More Sophistication

- Initialization without locks
- Read-shared data (written only during init, read-only afterwards)
- Reader-writer locking (multiple readers)
- False Alarms still possible

- Don't start until see a second thread
- Report only after it becomes write shared
- Change algorithm to reflect lock types
  - On read of v by t:
    $C(v) = C(v) \cap$ locks-held(t)
  - On write of v by t:
    $C(v) = C(v) \cap$ write-locks-held(t)

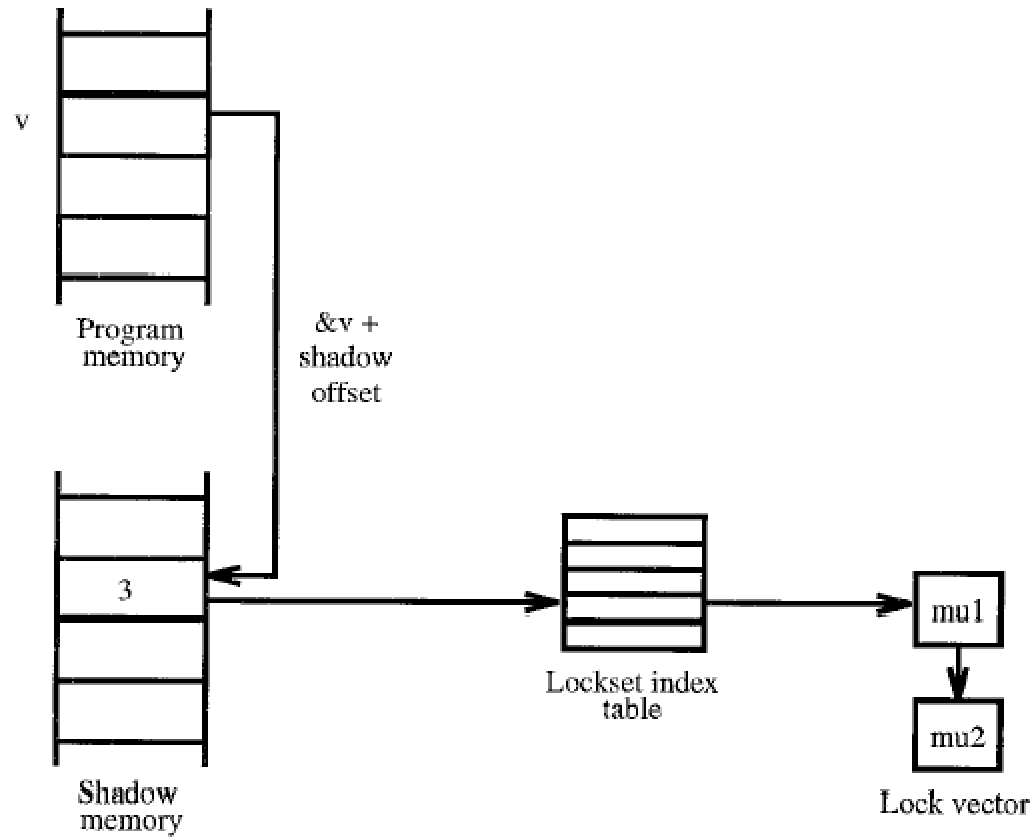
Per-Location State



Implementation

- Binary rewriting used
  - Add instrumentation to call Eraser runtime
  - Each load and store updates C(v)
  - Each Acquire and Release call updates locks-held(t)
  - Calls to storage allocator initializes C(v)
- Storage explosion handled by table lookup and use of indexes to represent sets
  - Shadow word holds index number
- Slowdown by factor of 10 to 30x
  - Will change interleavings

Shadow Memory and Lockset Indexes



Common False Alarms - Annotations

- Memory reuse
  - ➢ EraseReuse – resets shadow word to virgin state
- Private locks
  - ➢ Lock annotations
- Benign races
  - ➢ EraserIgnoreOn() EraserIgnoreOff()

```
if (some_condition) {
    LOCK m DO
      if (some_condition)
      {stuff}
    END
}
```

Races inside OS

- Using interrupt system to provide mutual exclusion – this implicitly locks everything affected (by interrupt level specified)
  - Explicitly associate a lock with interrupt level – disabling interrupt is like acquiring that lock
- Signal and wait kind of synchronization
  - V to signal for P which waits -- semaphore not “held” by thread.

An OK Race in AltaVista

```
if (p→ip_fp == (NI2_XFILE *) 0) {             // has file pointer been set?
    NI2_LOCKS_LOCK (&p→ip_lock);              // no? take lock for update
    if (p→ip_fp == (NI2_XFILE *) 0) {         // was file pointer set
                                              // since we last checked?
       p→ip_fp = ni2_xfopen (                 // no? set file pointer
                     p→ip_name, "rb");
    }
    NI2_LOCKS_UNLOCK (&p→ip_lock);
}
                     . . .                    // no locking overhead if file
                                              // pointer is already set
```

Bad Race in Vesta

```
Combine::XorFPTag::FPVal( ) {
  if (!this→validFP) {                      // is fingerprint marked valid?
                                            // no? calculate fingerprint
    NamesFP(fps, bv, this→fp, imap);        //   (NamesFP changes this→fp)
    this→validFP = true;                    // and mark it as valid
  }
  return this→fp;
}
```

This is a serious data race, since in the absence of memory barriers the Alpha semantics does not guarantee that the contents of the **validFP** field are consistent with the **fp** field.

Core Loop of Lock-Coupling

```
// ptr–>lock.Acquire(); has been done before loop

while (next != null & key > next->data)
      {next–>lock.Acquire();
       ptr–>lock.Release();
       ptr=next;
       next=ptr–>next;
}
```