Key Idea: Eraser does **not necessarily** detect races. Eraser detects dangerous locking (often leading to races).

Simple Data Race:

\[
v = 0
\]
\[
\text{def } f():
\]
\[
v = v + 1
\]

when *f* is run by multiple threads, *v* can end up with an unexpected value (such as 1)

when analyzing concurrency, useful to split program into reads and writes

Ex:

\[
\text{def } f():
\]
\[
x = v \text{ (read } v \text{)}
\]
\[
v = x + 1 \text{ (write } v \text{)}
\]

Now the race is obvious

\[
T_1:
\]
\[
\text{read } v
\]
\[
\text{write } v = 1
\]

\[
T_2:
\]
\[
\text{read } v
\]
\[
\text{write } v = 1
\]
Eraser will warn us of this problem because variable v is not protected by any locks.

**Correct Locking:**

```python
def FC():
    Lock(l1)
    v=v+1
    Unlock(l1)
```

**Question:** Eraser slows down a program, which can prevent races from happening. Why is this okay?

**Answer:** As long as two threads access a shared variable (even at completely different times) without a lock, Eraser will detect it.

**Ex.**

T₁:

- v = v+1
- (time)
- v = v+1

T₂:

- Eraser will still notice that v₂ is accessed without a lock, even though no race actually happens
Eraser False Positives:

Read-only variables:

\[
\begin{align*}
T_1: & \quad T_2': \\
X = V & \quad Y = V
\end{align*}
\]

Eraser solves this by waiting until a variable is read-write shared before issuing warnings.

Initialization:

\[
\begin{align*}
T_1: & \quad T_2: \\
X = \text{new object()} & \quad \cdot \\
X, Y = \text{"foo"} & \quad \cdot \\
\text{init}(x_{\text{-lock})} & \quad \cdot \\
// all accesses to \\
// x now happen \\
// with lock held
\end{align*}
\]

Eraser solves this by waiting until another thread accesses the variable before checking its locks.
Private Lock Implementations:

Eraser adds its routines by checking a program for calls to the pthread library (a common C-library).

If a program uses other locking schemes, Eraser won't detect this.

Benign Races:

```c
extern

if (x == null) {
    lock(c);
    if (x == null)
        x = new object(c);
    unlock(c);
}
```

This code is correct. Why would a programmer do this?

*Performance*: Locking is expensive.
Eraser False Negatives:

```plaintext
if (today == Xmas)
    v = v + 1
else {
    lock(x)
    v = v + 1
    unlock(x)
}
```

Eraser will only detect bad locking practice on Xmas!

Code needs to execute for Eraser to detect problems.

2010 Quiz 1 Problem 5:

A. Each account is a shared variable and is protected by its own lock when accessed. Eraser will not issue a warning.

B. Now Eraser will issue a warning b/c TotalBalance is reading an account variable without a lock.
C. True. Transfer maintains the same total amount between accounts when run to completion.

D. False

Ex:

\[ T_1 (\text{running total balance}) : \]
\[ \text{total} = \text{total} + \text{change}(a, 0) \]
\[ \text{total} = \text{total} + \text{change}(b, 0) \]
\[ \text{total will end up with a balance too large} \]

E. False

Ex:

\[ T_1 : \]
\[ \text{change}(a, -100) \]
\[ \text{change}(b, 100) \]
\[ T_2 : \]
\[ \text{Total Balance}() \]
This problem is a good example of another Eraser false negative.

Eraser issued no warnings, but we still had a bug!