Exercise 3.45.

Louis Reasoner thinks our bank-account system is unnecessarily complex and error-prone now that deposits and withdrawals aren't automatically serialized. He suggests that `make-account- and-serializer` should have exported the serializer (for use by such procedures as `serialized-exchange`) in addition to (rather than instead of) using it to serialize accounts and deposits as `make-account` did. He proposes to redefine accounts as follows:

```
(define (make-account-and-serializer balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! blance (+ balance amount))
    balance)
  (let ((balance-serializer (make-serializer)))
    (define (dispatch m)
      (cond ((eq? m 'withdraw) (balance-serializer withdraw))
            ((eq? m 'deposit) (balance-serializer deposit))
            ((eq? m 'balance) balance)
            ((eq? m 'serializer) balance-serializer)
            (else (error "Unknown request -- MAKE-ACCOUNT"
                         m))))
    dispatch))
```

Then deposits are handled as with the original `make-account`:

```
(define (deposit account amount)
  ((account 'deposit) amount))
```

Explain what is wrong with Louis's reasoning. In particular, consider what happens when `serialized-exchange` is called.

Answer.

When `serialized-exchange` is called by evaluating the expression

```
(serialized-exchange a1 a2)
```

to exchange the account balances in $a1$ and $a2$, the entire `exchange` procedure is serialized by both accounts using their own `balance-serializer`s. The `serialized-exchange` procedure in turn proceeds its job by evaluating

```
((a1 'withdraw) difference)
((a2 'deposit) difference)
```

At this point, according to Louis, we will evaluate

```
((balance-serializer withdraw) difference)
```

inside the procedure object `a1` by dispatching. Note that here we exploit the `balance-serializer` for a second time to serialize another procedure `withdraw` other than `exchange`. This indicates that both `exchange` and `withdraw` locate in the same serialized set, even though they are implemented hierarchically. Remember that when the serializer was first introduced, we stipulated that only one execution of a procedure in each serialized set is permitted to happen at a time. Therefore, so long as `exchange` is executed, another process that tries to execute `withdraw` in `a1` will be forced to wait until the first execution has finished. On the other hand, we see that `exchange` cannot accomplish its job without executing `withdraw`. Yet an eternal waiting would be arouse by this contradiction.

Similarly, evaluating

```
((balance-serializer deposit) difference)
```

inside `a2` also leads to an endless waiting.