

### Exercise 3.41.

Ben Bitdiddle worries that it would be better to implement the bank account as follows (where the commented line has been changed):


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

because allowing unserialized access to the bank balance can result in anomalous behavior. Do you agree? Is there any scenario that demonstrates Ben's concern?

### Answer.

Ben's worry is unnecessary. Observe that both `withdraw` and `deposit` access `balance` locally inside the account, rather than through dispatching. By serializing `withdraw` and `deposit`, we've ensured that no interleaving will occur whenever any one of  $P_1$  and  $P_2$  is accessing to the account. Moreover, for each of the two processes, serialization guarantees that just before the moment of change, the balance is still what they thought it was. There won't be any scenario that demonstrates Ben's concern.

---

\*. Creative Commons  2013, Lawrence X. Amlord (颜世敏, aka 颜序).