T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	$B := B + temp$
	write(B)
	commit

Figure 17.5 Schedule 4—a concurrent schedule resulting in an inconsistent state.

actions are programs, it is difficult to determine exactly what operations a transaction performs and how operations of various transactions interact. For this reason, we shall not consider the various types of operations that a transaction can perform on a data item, but instead consider only two operations: read and write. We assume that, between a read(Q) instruction and a write(Q) instruction on a data item Q, a transaction may perform an arbitrary sequence of operations on the copy of Q that is residing in the local buffer of the transaction. In this model, the only significant operations of a transaction, from a scheduling point of view, are its read and write instructions. Commit operations, though relevant, are not considered until Section 17.7. We therefore may show only read and write instructions in schedules, as we do for schedule 3 in Figure 17.6.

In this section, we discuss different forms of schedule equivalence but focus on a particular form called **conflict serializability**.

Let us consider a schedule S in which there are two consecutive instructions, I and J, of transactions T_i and T_j, respectively ($i \neq j$). If I and J refer to different data items, then we can swap I and J without affecting the results of any instruction in the schedule. However, if I and J refer to the same data item Q, then the order of the two steps may matter. Since we are dealing with only read and write instructions, there are four cases that we need to consider:

1. I = read(Q), J = read(Q). The order of I and J does not matter, since the same value of Q is read by T_i and T_j, regardless of the order.