


```

time mainA                                mainB
-----
t0  run  main
t1  1 for noRWConflict
t2  2 for OC_bufferUntilCommit
t3                                     run main
t4                                     1 for noRWConflict
t5                                     2 for OC_bufferUntilCommit
t6  7 <Rrite a>
t7                                     7 <Read a>
t8  8 <Write a>
t9  2 <commit>
t10                                    8 <Write a>
t11                                    2 <commit>

time mainA                                mainB
-----
t0  run  main
t1  1 for noRWConflict
t2  2 for OC_bufferUntilCommit
t3                                     run main
t4                                     1 for noRWConflict
t5                                     2 for OC_bufferUntilCommit
t6  6 <WI-lock a>
t7                                     6 <WI-lock a>
t8  8 <Write a>
t9  2 <commit>
t10                                    8 <Write a>
t11                                    2 <commit>

time mainA                                mainB
-----
t0  run  main
t1  1 for noRWConflict
t2  2 for OC_bufferUntilCommit
t3                                     run main
t4                                     1 for noRWConflict
t5                                     2 for OC_bufferUntilCommit
t6  6 <WI-lock a>
t7                                     7 <R-lock a>
t8                                     6 <WI-lock a>
t9  8 <Write a>
t10  2 <commit>
t11                                    8 <Write a>
t12                                    2 <commit>

```

```

time mainA                                mainB
-----
t0  run  main
t1  1 for noRWConflict
t2  1 for OC_noBuffering
t3                                     run main
t4                                     1 for noRWConflict
t5                                     1 for OC_noBuffering
t6  6 <WI-lock a>
t7                                     7 <R-lock a>
t8                                     6 <WI-lock a>
t9  8 <Write a>
t10                                    8 <Write a>
t11  2 <commit>
t12                                    2 <commit>

time mainA                                mainB
-----
t0  run  main
t1  1 for noRWConflict
t2  3 for OC_defaultBuffering
t3                                     run main
t4                                     1 for noRWConflict
t5                                     3 for OC_defaultBuffering
t6  6 <WI-lock a>
t7                                     7 <R-lock a>
t8                                     6 <WI-lock a>
t9  8 <Write a>
t10                                    8 <Write a>
t11  2 <commit>
t12                                    2 <commit>

```

■ Task 3: Understand mainTran1.cxx.

2 Transactions(2): Nested Transaction

```

iwaki% cd ~/transactions
iwaki% make tran2
iwaki% ./mainTran2 ${USER}tutorialDB

```

This example consists of `Person.cxx` and `mainTran2.cxx`. The explanation for the function `bind_person()` in `main.C` is given below.

```

void bind_person(Person *p)
{
    // This will need to change when we eventually phase out TRefs.
    OC_Reference to_x(p, p)
    to_x.getReferent(p);
}

```

There is a way to ensure that an in-memory object modified in a nested transaction reverts to its prior state if the transaction is aborted. In a nested transaction, call `OC_Reference::getReferent()` on a reference to the object. If no such `OC_Refernce::getReferent()` exists, you must build one and call `getReferent()` on it. This must be done before any modifications are made to the object at the nested level. The reasons can be found in the two examples below.

- **Task 4:** What is the internal state of object “b” before/after the nested transaction is aborted in the first example?
- **Task 5:** What is the internal state of the object “b” before/after the nested transaction is aborted in the second example?
- **Task 6:** The two examples abort the nested transaction. Which abort is an expected abort? How do we manage the consistency between nested-transactions?

```

-----
t1  choose '0' for enableRWConflict
t2  choose '8'<Dump b>
t3  choose '9'<Write b>
t4  choose '8'<Dump b>
t5  choose '1'<Start>    start a nested transaction.
t6  choose '0' for RWConflict
t7  choose '11' <Write already-active b>
t8  choose '8'<Dump b>
t9  choose '3'<Abort>    abort the nested transaction.
t10 choose '8'<Dump b>

```

```

-----
t1  choose '0' for enableRWConflict
t2  choose '8'<Dump b>
t3  choose '9'<Write b>
t4  choose '8'<Dump b>
t5  choose '1'<Start>    start a nested transaction.
t6  choose '0' for RWConflict
t7  choose '10'<Rebind already-active b>
t8  choose '11'<Write already-active b>
t9  choose '8'<Dump b>

```

```
t10 choose '3'<Abort>    abort the nested transaction
t11 choose '8'<Dump b>
```

- **Task 7:** Understand mainTran2.cxx.
- **Task 8:** Understand nested transactions.