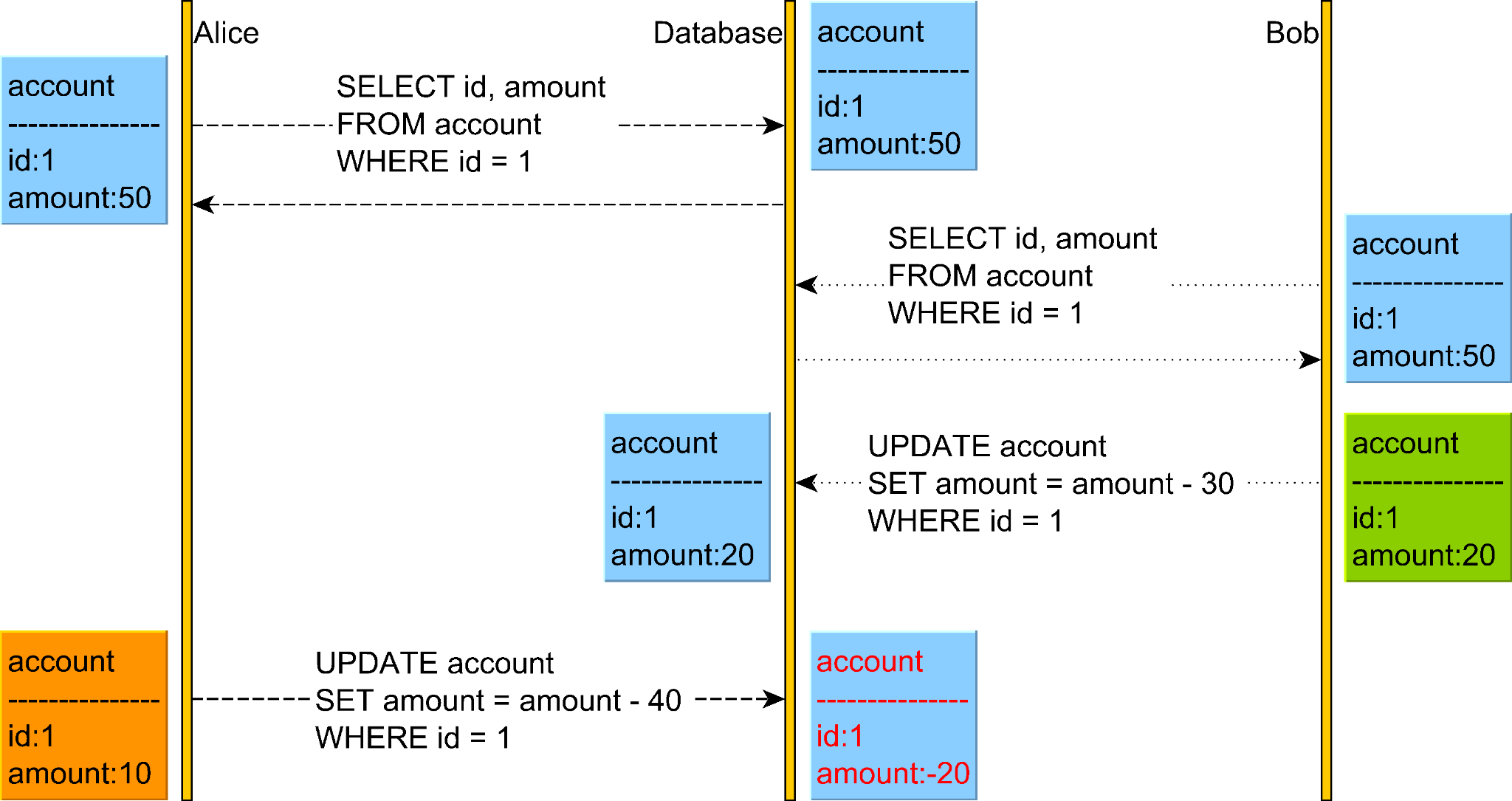
**Optimistic locking**

* Strategy where you **read a record, take note of a version number and check that the version hasn't changed before you write the record back**. When you write the record back you **filter the update on the version to make sure it's atomic and update the version in one hit**. If the record is dirty (i.e. different version to yours) you abort the transaction and the user can re-start it.
* This strategy is **most applicable to high-volume systems and three-tier architectures** where you **do not necessarily maintain a connection to the database for your session**. In this situation, the client cannot actually maintain database locks as the connections are taken from a pool and you may not be using the same connection from one access to the next.

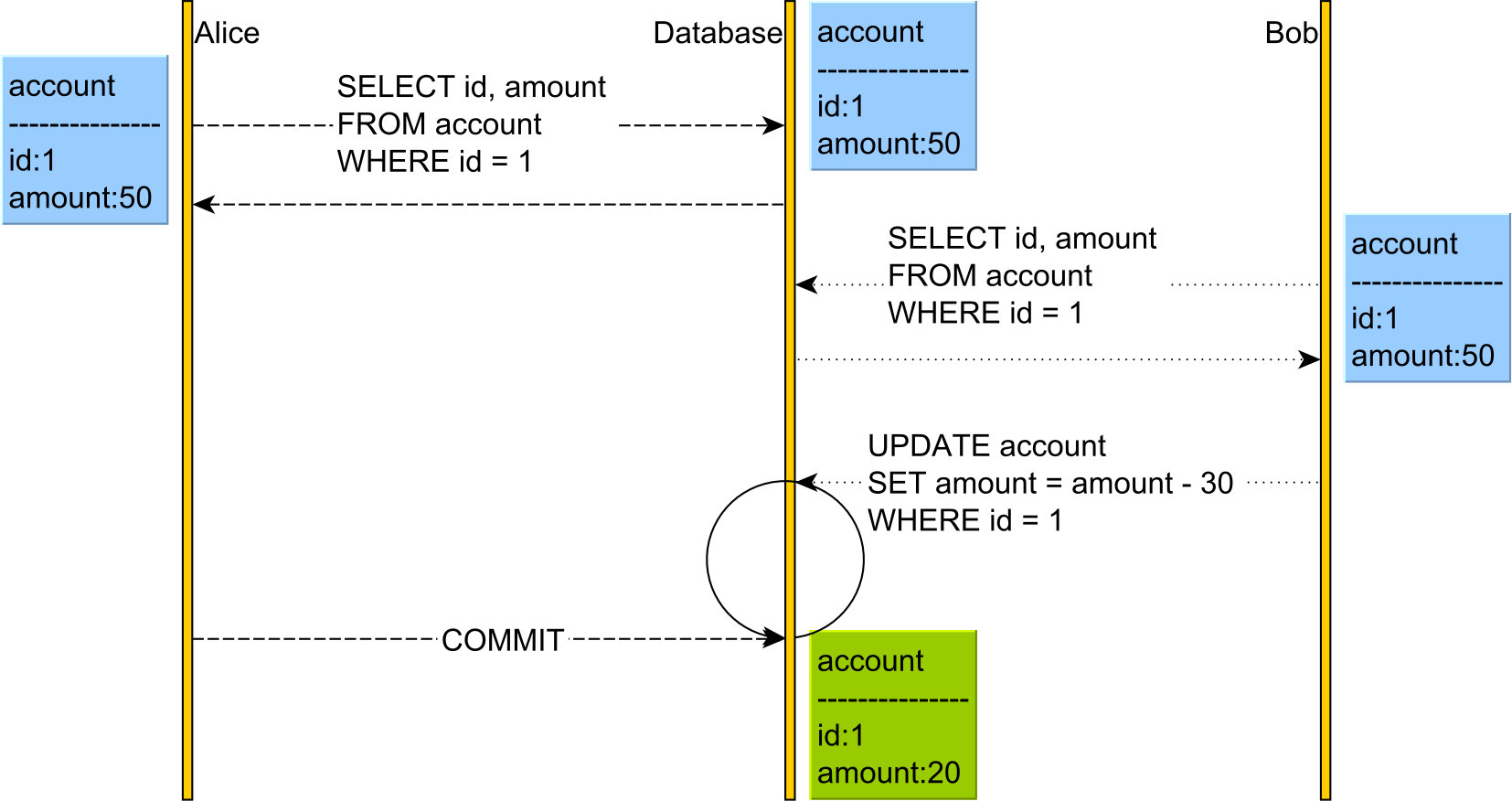
**Pessimistic Locking**

* Startegy where you **lock the record for your exclusive use until you have finished with it**. It has **much better integrity than optimistic locking** but requires you to be **careful with your application design to avoid** [**Deadlocks**](http://en.wikipedia.org/wiki/Deadlock).
* To use pessimistic locking you **need either a direct connection to the database or an externally available transaction ID that can be used independently of the connection**. In the latter case, you **open the transaction with the TxID and then reconnect using that TxID**. The **DBMS maintains the locks and allows you to pick the session back up through the TxID**. This is how distributed transactions using two-phase commit protocols work.

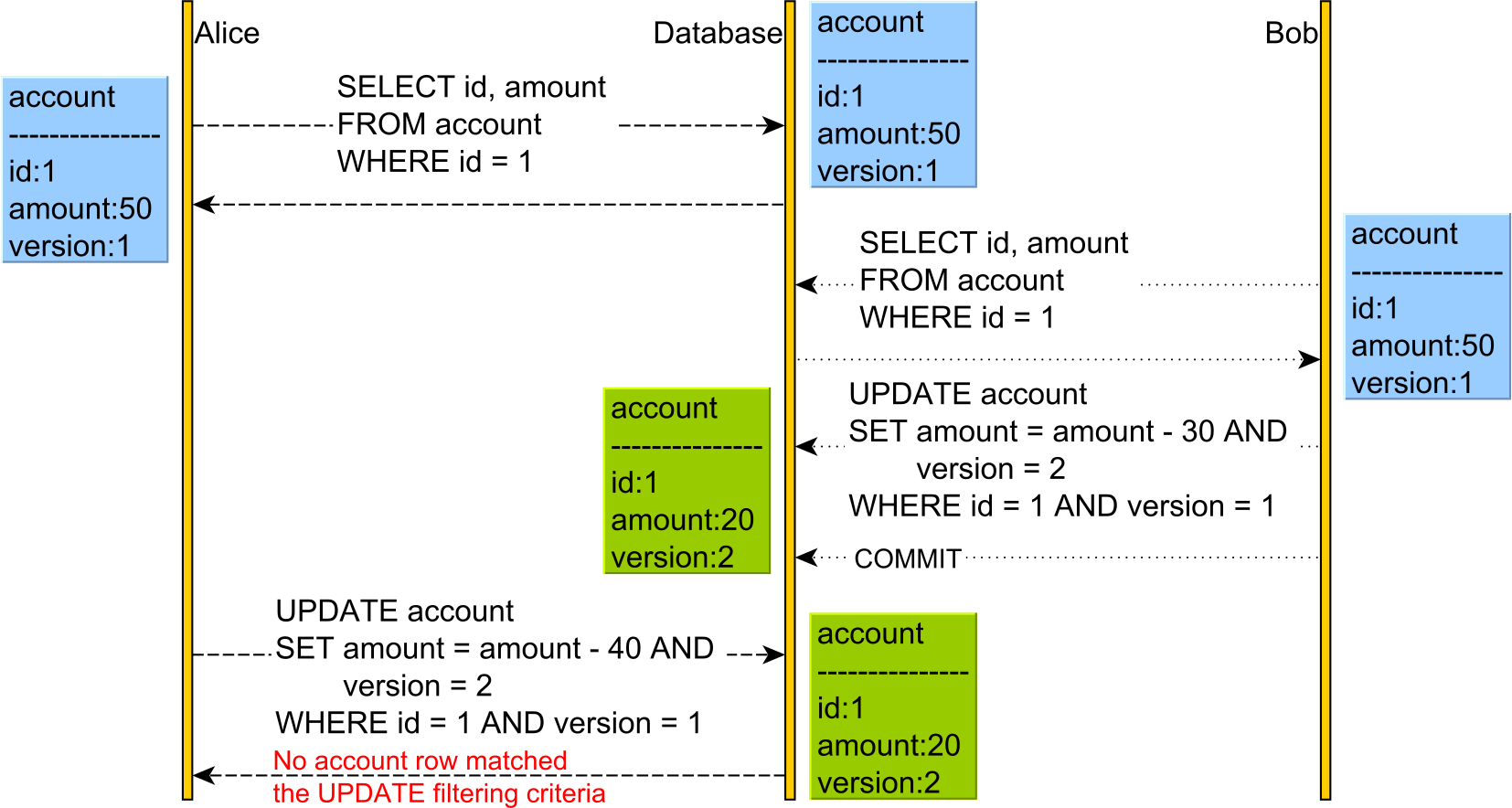
Let's consider the following **Lost Update anomaly**:



**Pessimistic locking prevents this by taking a shared or read lock on the account** so Alice is prevented from changing the account.



**Optimistic Locking allows the conflict to occur** but detects it upon applying Alice's UPDATE as the version has changed.



The **downside of optimistic locking** is that **a rollback will be triggered by the data access framework upon catching an OptimisticLockException**, therefore **losing all the work we've done previously by the currently executing transaction**.