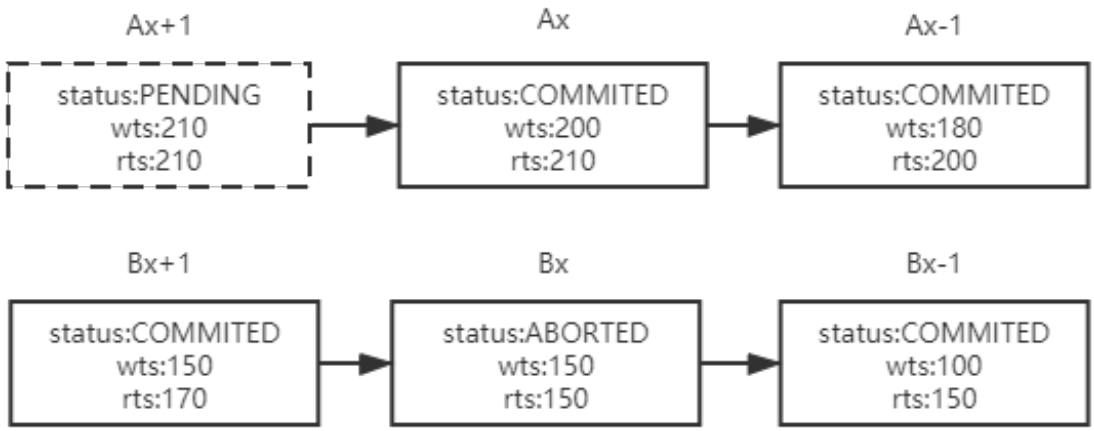
MVCC+OCC参考算法

一、数据结构



如图所示，我们用A/B代表一个数据项，Ax/Ax-1等代表数据项A的多个版本。
数据项的每个版本需要维护如下信息：
- wts: 当前版本的写入时间戳
- rts: 读过该版本的所有事务的时间戳的最大值
- status: 写这个版本的事务的状态，共有PENDING(尚未提交)、COMMITED(已经提交)、ABORTED(已经回滚)三种
- pointer: 从新版本指向下一个旧版本的指针

每个事务txn需要维护:

- Rset/Wset: 读集/写集
- Cset: 验证时写入的临时版本集合
- Tstart: 事务的时间戳，事务开始时获取

二、主要思想

定序：以事务开始时间戳Tstart作为定序标准，

检验：在验证阶段对三种冲突进行检验。

我们不妨假设有事务T1和T2且`T1.Tstart < T2.Tstart`，根据三种冲突梳理检验逻辑：

- 写写冲突：在验证阶段写入PENDING版本。如果T1和T2同时写入一个数据项
  - 回滚后写入PENDING版本的事务
- 写读冲突：若事务T1写入版本v，T2要读版本v
  - 如果事务T1已经提交，T2可以读取到版本v，因为与按Tstart所确定顺序一致
  - 如果事务T1还没提交但已经写入PENDING版本，事务T2等T1提交后再读。
  - 如果事务T1还没写入PENDING版本，事务T2只能读取到v的上一个版本(v-1)，T2在验证时需要调整(v-1)版本的rts，使其满足：`(v-1).rts >= T2.Tstart`，从而形成`(v-1).rts > T1.Tstart`，而让事务T1回滚
- 读写冲突：T1事务读了版本v，T2写新版本(v+1)
  - 如果T1先读，T2后写，读写顺序和时间戳顺序相同
  - 如果T2先写，T1后读，由于MVCC，T2只能读取到版本v，读取不到(v+1)，读写顺序和时间戳顺序相同

三、算法逻辑

Transaction start

```
txn.Tstart = generate next timestamp
```

Exeute phase

在执行阶段，仅根据事务时间戳获取对应版本，并维护读写集

```
if (find the newest Ax statisfy Ax.wts <= Tstart)
    if (Ax.status == PENDING) wait
    if (Ax.status == ABORTED) find next version
    if (Ax.status == COMMITED) {
        Rset or Wset <-- Ax
        return Ax
    }
return A does not exist
```

1. 根据事务txn的开始时间戳Tstart去获取版本，判断当前版本的状态status
   a) 如果是PENDING，那么代表当前版本没提交，先进行等待
   b) 如果是COMMITED，将获取到的版本存入读集(读操作)或者写集(写操作)，并返回该版本。
   c) 如果是ABORTED，找下一个版本

Validate phase

```
// insert pending version
for Ax in Wset
    if ([Ax.rts > Tstart] or [exist Ax+1])
        return abort
    else
        new Ax+1
        Ax+1.status = PENDING
        Ax+1.rts = Ax+1.wts = Tstart
        Cset <-- Ax+1
// Update read timestamp
for Bx in Rset
    if (Bx.rts < Tstart)
        Bx.rts = Tstart
// Check version consistency
for Bx in Rset
    find the newest Bx2 statisfy Bx2.wts <= Tstart
    if (Bx != Bx2) return abort
for Ax in Wset
    if (Ax.rts > Tstart) return abort
return commit
```

1. 针对写集中的版本Ax，尝试写入新版本Ax+1。
   a) 若Ax.rts>Tstart，当前事务回滚
   b) 如果已经存在一个新版本Ax+1，那么当前事务回滚
   c) 若不存在a、b两种情况，写入PENDING版本，设置status为PENDING，wts和rts设置为事务Tstart。
2. 更新读集的rts，调整读集中版本Bx.rts>=Tstart
3. 版本一致性检测
   a) 重新读取一遍读集数据项，如果读取不到之前的结果(比如插入了新版本)，当前事务回

滚

b) 检测写集中数据项版本Ax.rts，如果不满足Ax.rts<=Tstart，当前事务回滚。

Commit phase

```
// commit
for Cx in Cset
    Cx.status = COMMITED
```

将验证时写入的版本状态改成COMMITED

Abort phase

```
// abort
for Cx in Cset
    Cx.status = ABORTED
```

将验证时写入的版本状态改成ABORTED