1.OCC在2pc的commit阶段执行，COCO的2pc和普通2pc区别只在于普通的是准备和提交一个事务，而这个是准备和提交一个epoch的事务

2.initiating a transaction的是coordinator节点，在执行阶段时，所有数据已经到了各个节点了
代码应该执行在coordinator上，有事务的读写集,其中读到的locked是判断有没有被除自己以外的事务锁定

将事务所有记录的primary node给锁定上，这个tid是新生成的，如果这条记录在写集中，那么就遵循Thomas写规则，record的tid用最新的tid覆盖。然后ok的判断是判断这个锁有没有被其他的事务占用，如果占用了，他为了避免死锁，采用了no_wait的思索防止策略，就是一旦有其他的事务占用锁了，那就直接abort就好。后面的TID不同就是可能这条记录被覆盖掉了，所以就abort，写写冲突。

Locking the write set

P T | O C C

```
1  for record in T.WS:
2      ok, tid = call_n(lock, record.key)
3      if record not in T.RS:
4          record.tid = tid
5      if ok == false or tid != record.tid:
6          abort = true
```

再次确认此时的记录被当前事务lock并且没被其他修改

Validating the read set

```
for record in T.RS \ T.WS:
    # begin atomic section
    locked, tid = call_n(read_metadata, record.key)
    # end atomic section
    if locked or tid != record.tid:
        abort()
```

没有被abort就先提交到primary node的数据库，解锁这条记录，然后再异步的把这数据更改到其他participants node

Writing back to the database

```
for record in T.WS:
    call_n(db_write, record.key, record.value, record.tid)
    call_n(unlock, record.key)
    for i in get_replica_nodes(record.key) \ {n}:
        call_i(db_replicate, record.key, record.value, T.tid)
```

wts代表记录何时写入，rts代表在[wts, rts]区间都可以读

大部分相同，wts是记录写的时间，也相当于之前physical time的TID，rts是可以读取的时间，因为在当前时间有读写集，所以rts更新到最新的时间

locking the write set

LTIOCC

```
1  for record in T.WS:
2      ok, {wts, rts} = call_n(lock, record.key)
3      if record not in T.RS:
4          record.wts = wts
5      if ok == false or wts != record.wts:
6          abort()
7      record.rts = rts
8
```

TID>写集中的rts并且TID>=读集中的wts 中的最小的时间戳，TID之后就是读集中的记录已经全部写入，可读的时间，读集中的rts<Tid时需要尝试扩展,>Tid就无需验证，(TID>写集中的rts是否没有存在的意义)，事务会在两种情况abort 1.事务记录被其他并发事务修改也就是wts被更改 2.rts<T.tid此时需要扩展rts，让该记录在TID时刻可读，但是如果被其他事务锁定，此时需要abort。否则就把rts扩展到tid

validating the read set

```
for record in T.RS \ T.WS:
    if record.rts < T.tid:
        # begin atomic section
        locked, {wts, rts} = call_n(read_metadata, record.key)
        if wts != record.wts or (rts < T.tid and locked):
            abort()
        call_n(write_metadata, record.key, locked, {wts, T.tid})
        # end atomic section
```