**Transaction**：

1. 主从节点收到Tx后，将Tx封装为Request；

**Request**：

1. 主从节点将构造的Request广播给其他节点；
   1. **存储**：主从节点将不重复的request存到obcBatch.reqStore. outstandingRequests；
   2. **定时器**：主从节点判断此时没有在进行数据同步，没有在执行交易，没有在视图转换，并且outstandingRequests中有未处理完的 request，就软开启newViewTimer；

**RequestBatch：这部分都是主节点在处理**

1. 主节点收到Request，调用obcBatch.leaderProcReq()进行处理：
   1. **存储**：将其存到obcBatch.batchStore和obcBatch.reqStore.pendingRequests；
   2. **定时器：**检查并开启打包阈值定时器obcBatch.batchTimer；
2. 到达打包时间或数量阈值，主节点调用obcBatch.sendBatch()将batchStore中的所有request打包构造并广播requestBatch；
   1. **定时器**：主节点会停止打包阈值定时器obcBatch.batchTimer；
   2. **存储：**主节点会清空obcBatch.batchStore中的request；
3. 此时主节点会进入到pbftCore.recvRequestBatch()方法；
   1. **存储**：主节点将requestBatch存储到pbftCore.reqBatchStore和pbftCore.outstandingReqBatches；
   2. **定时器**：主节点检查并软重置pbftCore.newViewTimer，停止obcBatch.nullRequestTimer；

**Pre-prepare**:

5.1 主节点检查requestBatch还没有存在pbftCore.certStore中，将构造并广播Pre-prepare消息；

a) **存储**：将Pre-prepare消息存到pbftCore.certStore和pbftCore.qset；

5.2. 主节点调用maybeSendCommit()方法判断这个Pre-prepare消息是否到了prepared状态，到了就构造并发送commit消息；

**Prepare:**

6. 从节点收到Pre-prepare消息后，调用pbftCore.recvPrePrepare()进行处理；

a） **验证**：验证主节点身份、验证n是否在水线范围内、是否需要强制视图转换、验证是否跟certStore中的Pre-prepare消息存在冲突，后面三种不合法的话就出发视图转换；

b） **存储**：从节点节点直接将requestBatch存储到pbftCore.reqBatchStore和pbftCore.outstandingReqBatches，将PrePrepare存储到pbftCore.certStore和qset中；

c） **定时器**：从节点软重置pbftCore.newViewTimer，停止obcBatch. nullRequestTimer；

7. 从节点验证当前请求到达Pre-prepared状态并且自己还没有发送过Prepare消息，就构造并广播Prepare消息；

8. 主从节点收到Prepare消息后，调用pbftCore.recvPrepare()进行处理：

a） **验证**：验证主节点身份、验证v、n;

b） **存储**：主从节点将Commit消息存储到pbftCore.certStore.commit中；

8.1 主从节点调用pbftCore.calcPSet(),将到达Prepared状态的requestBatch信息存储到pbftCore.pset；

8.2. 主从节点调用pbftCore.maybeSendCommit(),判断这个requestBatch是否达到Commited状态，到了就发送Commit消息；

**Commit:**

9. 主从节点收到Commit消息后，调用pbftCore.Commit()，进行如下处理：

a）**验证**：验证v、n;

b）**存储**：主从节点将Commit消息存储到pbftCore.certStore.commit;

主从节点验证这个requestBatch是否到达了Committed状态，到达了，进行如下处理：

a）**定时器**：停止pbftCore.newViewTimer;

b）**存储**：主从节点删除pbftCore.outstandingReqBatches中committed的requestBatch；

**Execute：**

9.1（开始） 调用pbftCore.executeOutstanding()进行执行；

这个方法通过for循环，调用pbftCore.executeOne(),执行pbftCore.certStore中一个到达committed状态的requestBatch；

9.1.1 在pbftCore.executeOne()中，验证是否落后，落后就进行数据同步，不落后分两种情况：

第一，如果是null request,直接调用pbftCore.execDoneSync()；

第二，如果是正常的requestBatch，调用pbftCore.consumer.execute()，然后在pbftCore.consumer是obcBatch,所以，本质上是调用obcBatch.executeOne();

9.1.2 在obcBatch.executeOne()中，先调用obcBatch.reqStore.remove(req)，

a) **存储**：删除reqStore中的要执行的request；

然后，将requestBatch变成txs，调用由【执行模块】实现的接口obcBatch.stack.Execute()；其他模块执行完后，会调用obcBatch实现的Executed()接口，最终会返回一个executedEvent事件；

9.1.3在obcBatch.ProcessEvent()中executedEvent分支会调用由【提交模块】实现的obcBatch.obcGeneric.stack.Commit()接口，其他模块提交完这个块后，肯定最终会返回一个committedEvent事件，然后再返回execDoneEvent事件；（其实在executed之后是没有任何处理的，直接调用了Commit进行提交，所以这里可以直接执行并提交就可以，估计写这个代码的人做过传统的分布式，熟悉两阶段提交等）

9.1.4 在pbftCore.ProcessEvent(event）的execDoneEvent分支，调用pbftCore.execDoneSync()，这个时候是共识模块更新相关变量lastExec、currentExec，检查是否需要gc；注意，在pbftCore.execDoneSync()的结尾又调用了9.1中的pbftCore.executeOutstanding（），继续执行committed的requestBatch；

9.1（结束）pbftCore.executeOutstanding()结束的时候，调用pbftCore.startTimerIfOutstandingRequests();如果outstandingReqBatches中还有需要待处理的requestBatch,软重置 newViewTimer；如果没有待处理的requestBatch就重启 nullRequestTimer。

**Checkpoint**:

10. 经过K轮之后，主从节点调用pbftCore.Checkpoint()，构造、存储并广播Checkpoint；

a) **存储**：将Checkpoint到自己的pbftCore.chkpts中，chkpts存的是<n,id>序号，链状态；

11. 主从节点收到checkpoint之后，调用pbftCore.recvCheckpoint()进行处理；

11.1 首先，调用pbftCore.weakCheckpointSetOutOfRange(Checkpoint)，验证这个Checkpoint的编号是不是不在水线范围内；

如果低于我的最低水线说明其他节点落后了，不做处理；（因为落后的节点也会收到checkpoint，会发现自己落后的）；

如果高于我的最高水线H，则将这个Checkpoint存到pbftCore.hChkpts中，如果有f+1个checkpoint的序号都比我的H高，说明我落后了，需要将skipInProgress置为true进行状态同步，并将batch的缓存清空，移动水线；

11.2 到这里说明这个Checkpoint消息的序号在我的水线范围之内，如果收到了f+1个Checkpoint，说明对这次gc到达了弱一致性，这个时候节点调用pbftCore.witnessCheckpointWeakCert(Checkpoint),将highStateTarget字段更新为弱一致性的chkpt序号，这样如果自己落后了，就可以提前进行状态更新；

11.3 如果此时,pbftCore.checkpointStore中有2f+1个同样序号的checkpoint；

如果在pbftCore.chkpts中没有存此序号对应的链状态（id）说明我落后了，返回nil，其他部分会发现我落后了，再进行同步处理；

如果在pbftCore.chkpts中有此序号对应的链状态，说明到达稳定检查点，调用moveWatermarks()，再调用pbftCore.processNewView();

11.4 接下来说一下pbftCore.moveWatermarks()函数做了什么。首先，更新低水线为当前的稳定检查点h= n / instance.K \* instance.K；随后就是清除过期的缓存；

a) **存储**：主从节点删除pbftCore.certStore、pbftCore.reqBatchStore、pbftCore.checkpointStore、

pbftCore.pset、pbftCOre.qset中低于新的最低水线的requesBatch；删除pbftCore.chkpts中序号低于h的全局状态；

11.4.1 在移动水线moveWatermarks方法里面，最后会调用pbftCore.resubmitRequestBatches()方法，如果pbftCore.outstandingReqBatches中存在还没进入到共识阶段的requestBatch，主节点调用pbftCore.recvRequestBatch()继续对这些块进行共识；

**View-Change:**

当发现主节点作恶，newTimer定时器超时，或者需要进行强制视图转换等，构造并发送vc消息。

15. 主从节点调用pbftCore.sendViewChange()构造并发送vc消息；这个函数的主要作用是计算vc中的Pset和Qset，清空相关缓存，构造并广播vc消息；

15.1 首先，停掉newViewTimer定时器，然后view++;

15.2-15.3：其次，调用calcPSet()、calcQSet()更新Pset和Qset,这里的话是在原来Pset和Qset基础上，将根据pbftCore.certStore，将到达prepared和preprepared的requestBach消息分别追加到Pset、QSet中；

15.4-15.6：在构造vc之前，是不是可以清除一下小于这个视图的一些缓存？

a)**存储**：清除pbftCore.certStore、pbftCore.newViewStore、pbftCore.viewChangeStore中小于新视图编号的共识消息，viewChange、newView消息；

15.7 构造viewChange，其中CSet是pbftCore.chkpts中自己构造的checkpoint；然后，对viewChange消息签名，并广播；

15.8 在sendViewChange()的最后，会开启pbftCore.vcResendTimer定时器；

**NewView**:

16. 主从节点收到vc消息后，将会调用pbftCore.recvViewChange(vc)进行处理，他的主要作用是验证vc的合法性，若收到2f+1个合法的vc，返回viewChangeQuorumEvent事件；

16.1 首先，调用pbftCore.verify(vc)进行验签；

16.2 其次，调用pbftCore.correctViewChange(vc)验证vc的合法性，验证 Pset、Qset中消息的视图编号是否比vc中的视图编号小，n是否在高低水线之间；验证Cset中的Checkpoint是否在高低水线之间；

16.3 如果这个vc是合法的，那么将调用16.3 pbftCore.viewChangeStore(vc)，将vc存到自己的viewChangeStore中；

16.4 此时，判断一下，若收到f+1个视图编号比节点自己的view大的合法的vc，那么以f+1个vc中最小的view编号构造并发送vc；这个其实是论文的活性证明里面提到的，是为了尽快的进入到下一个视图；

16.5 最后，若收到2f+1个视图编号和节点自己的view一样大的合法的vc，并且没有在视图转换调用pbftCore.vcResendTimer.Stop()，关闭关闭pbftCore.vcResendTimer定时器；调用instance.startTimer()，开启pbftCore.newViewTimer；返回 viewChangeQuorumEvent事件；

17. 在pbftCore.ProcessEvent()中viewChangeQuorumEvent分支，主节点调用pbftCore.sendNewView()，他的主要作用就是构造并广播newView消息；

17.1 首先，根据viewChangeStore计算vset;根据vset计算最近的稳定检查点cp；根据cp和vset计算xset,xset是个map，map里面存的是<n,batch的摘要>对；

17. 2 最后，计算好了相关变量，那就构造、存储并广播NewView消息

a) 存储：将newView存到pbftCore.newViewStore中；

主节点直接调用pbftCore.processNewView()进行处理；

1. 从节点收到newView后，调用pbftCore.recvNewView()验证newView消息的合法性，主要验证view、primary

验证Vset中签名的合法性；如果newView合法，

a) **存储**：将newView存到自己的pbftCore.newViewStore中

18.1 随后调用pbftCore.processNewView(nv)进行处理，首先，验证CSet,XSet的合法性（同主节点计算过程）

分三种情况：

1）如果pbftCore. lastExec < cp.SequenceNumber&&之间的batch都已经到达了committed，返回nil；

2）如果pbftCore. lastExec < cp.SequenceNumber&&不能自己执行到cp，则调用状态更新方法；

3）到这里是pbftCore. lastExec >=cp.SequenceNumber,此时，验证自己是否存有Xset中对应的batch；

a) 存在缺失的batch，调用pbftCore.fetchRequestBatches()；

b) 不缺失batch,调用pbftCore.processNewView2(nv)；

18.2 然后，主从节点调用pbftCore.processNewView2(nv)进一步对newView进行处理，

a)**定时器**：首先关闭newViewTimer定时器，关闭pbftCore.nullRequestTimer；

根据XSet构造Pre-prepare消息并存储到pbftCore.certStore、pbftCore.qset中；

18.2.1 如果是从节点，则构造并广播Prepare消息；

18.2.2 如果是主节点，调用pbftCore.resubmitRequestBatches()，将在pbftCore.outstandingReqBatches 中还没进入到共识阶段的 requestBatch,调用pbftCore.recvRequestBatch()发起共识;

18.2.3 最后的时候，主从节点都会调用pbftCore.startTimerIfOutstandingRequests()，

a)**定时器**：如果outstandingReqBatches中还有requestBatch，软启动newViewTimer，否则就重启nullRequestTimer定时器；

18.2.4 返回viewChangedEvent事件；

19. 在obcBatch.ProcessEvent()的viewChangedEvent分支，

a) **存储**：清空pendingRequests并将obcBatch.pbft.cerStore中的req加入到pendingRequests；

然后，调用obcBatch.resubmitOutstandingReqs(),将在outstandingRequests但不在pendingRequests中的req Inject到Queue中；

**StateUpdate：**

两种情况发现自己需要进行状态同步：

1、在pbftCore.weakCheckpointSetOutOfRange()中，收到了f+1个序号超过自己高水位的checkpoint，说明数据落后了，将skipInProgress置为true；

2、 pbftCore.processNewView()中，发现自己落后newview中的checkpoint的时候会调用pbftCore.stateTransfer()将skipInProgress置为true，进行同步；

20. 调用pbftCore.retryStateTransfer()方法进行数据同步；通过currentExec字段判断是否有区块正在执行，通过stateTransferring字段判断是否正在进行数据同步，都没有的话就进行数据同步  ；

20.1 最终会调用其他模块的Executor.UpdateState()方法进行状态同步；执行完后会返回stateUpdatedEvent事件；

21. obcBatch.ProcessEvent()中的stateUpdatedEvent分支，

a) **存储**：清空obcBatch.reqStore中的request，没用了；

22. pbftCore.ProcessEvent()中的stateUpdatedEvent分支，

a) **存储**：删除pbftCore.chkpt中小于n的消息，移水线；