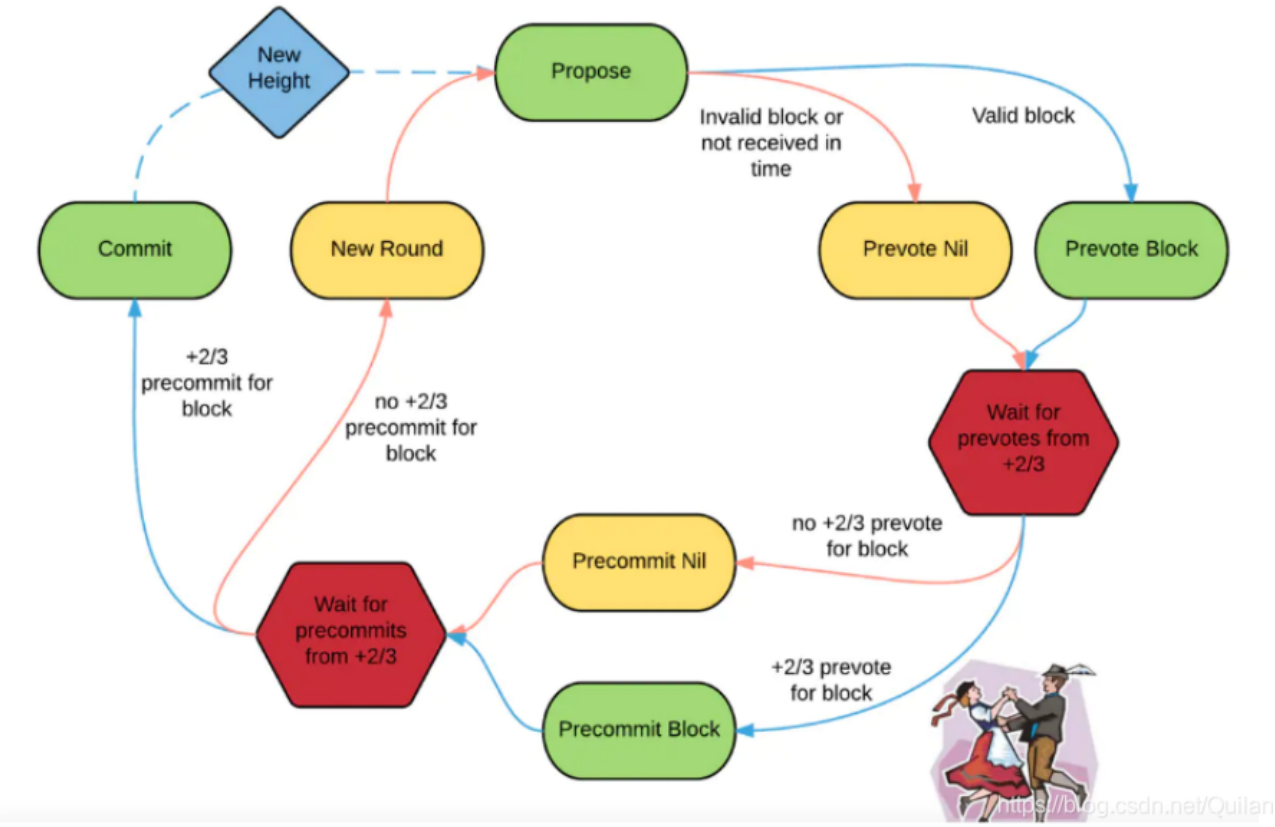
tendermint共识算法



图中的**Round/Height**为直观意思，**step**指图中的每一状态（Proposal、PreVote...） **状态转换周期：**

```
NewHeight -> (Propose -> Prevote -> Precommit)+ -> Commit -> NewHeight ->...
```

每轮的开始对同步有弱的依赖性。每一轮开始期间，存在一个用来计时的本地同步时钟，如果验证者在TimeoutPropose时间内没有收到提议，会立刻在本机生成一个特殊结构的空块，假装这个空块是从Proposer节点那里收到的，这样，无论如何，在时间T内，都会收到一个proposal区块，要么是一个正常块要么是一个空块。然后接着对这个块进行pre-vote投票和pre-commit投票。如果proposer挂了，绝大部分validator看到的都是一个空块，因此空块会获得多数投票，进入commit阶段。commit空块的时候，不会真的往区块链写入一个空块，而是什么都不写，区块高度不自增，保持不变，这样相当于什么也没有干，这一轮(round)是在空转。下一轮开始的时候会换下一个validator当proposer，这样当前那个挂掉的proposer，就不会卡主整个网络。

每轮收到提议以后，进入完全异步模式。之后验证者的每一个网络决定需要得到2/3验证者以上的同意。这样降低了对同步时钟的依赖或者网络的延迟。但是这也意味着如果得不到1/3以上验证者的响应，整个网络将瘫痪。Tendermint在Liveness方面有所妥协，换取了更强的Finality。 举个例子，如果在某一轮中proposer节点广播出了一个新块blockX，某个validator A节点没有按时收到新块，那么该A就会在本机构造一个空块，当做是从proposer收到的，发出一个pre-vote nil投票消息然后进入pre-vote循环，并启动一个超时定时器，这时进入了红色内圈循环，A开始监听网络并收集投票信息，

如果在规定时间内，收集到的投票数，无论是投给空块的还是blockX的，加起来，没有超过2/3，则无限等待，直到投票总数超过2/3

收集到了超过2/3的投票总数后，如果投给空块的票数超过2/3，则发出pre-vote nil投给空块，依旧留在红色内圈；如果投给blockX的票数超过2/3，则发出pre-vote投给blockX，切换到蓝色外圈；如果空块和blockX各自的票数，都没有超过2/3，那么发出pre-vote nil 消息投票给空块，进入pre-commit阶段，依旧在红色内圈。

一旦A发出了pre-commit nil的投票消息，A还是留在红色内圈循环，pre-commit流程与上面类似。总而言之，红色内圈的流程，需要假设网络是半同步的。

简言之，每轮开始提议**弱同步**，之后投票**完全异步**。

tendermint和传统PBFT的比较

1. Tendermint没有pBFT那种View Change阶段，Tendermint很巧妙的把超时的情况，跟普通情况融合成了统一的形式，都是 propose->pre-vote->pre-commit 三阶段，只是超时的时候新块是一个特殊的空块。切换proposer是通过提交commit空块来触发的，而pBFT是有一个单独的view change过程来触发primary轮换。
2. Tendermint的所有信息都存储在blockchain里。因为pBFT是1999年提出来的，那时候还没有blockchain这个东西(blockchain是2009年比特币出现之后才有的)，因此 pBFT的所有节点虽有有一致的数据，但数据是分散存放的。Blockchain就是一个分布式数据库，好比在MySQL这类DBMS数据库没出现之前，人们都是把数据写入文件然后存在硬盘上，发明出各种奇怪的文件格式和组织方式。有了MySQL后，管理数据就方便多了。同理，Tendermint 把数据全部存入blockchain, pBFT没有blockchain这样一个分布式数据库，所有全节点需要自己在硬盘上管理数据，比如为了压缩消息日志，丢弃老的消息，节省硬盘空间，引入了checkpoint的概念，光是数据管理这一块就多了很多繁琐的步骤。

股权

验证者在共识协议中可能具有不同的投票"权重"。 因此，Tendermint并不关注验证者数目的1/3或2/3， 而是关注总投票权的比例。此外，这个比例可能不是在各个验证者中均匀分布的。

验证者持有的货币可以绑定到保证金中，并且如果发现它们在共识协议中行为不当，则可以将其销毁。 这为协议的安全性增加了一个经济因素，当拜占庭节点小于三分之一的假设被打破时，人们可以量化产生的代价。

锁

波尔卡（Polka）：对提议区块的预投票数量超过2/3，对应图中两个人跳舞的地方 **什么时候锁**：验证者为某一轮的区块预提交，则锁定在该轮的区块上，锁定的验证者只能为锁定的区块预提交，不能为其他区块预投票和预提交 **什么时候解锁**：当出现一个波尔卡，则解锁

加入**锁**（**Lock**）主要是用于避免在同一高度的不同轮提交不同的区块。 举个例子：考虑4个验证者A,B,C,D，假设有一个第R轮关于blockX的提议。现在假设blockX已经有一个波尔卡，但是A看不见它，预提交（pre-commit）为空，然而其他人对blockX进行了预提交。进一步假设只有D看见了所有的预提交，然而其他人并没有看见D的预提交（他们只看见他们的预提交和A的空预提交）。D现在将要提交（commit）这个区块，然而其他人进入到R+1轮。由于任何验证者都可能是新的提议者，如果ABC提议并投票了一个新的区块blockY，他们可能达成共识并提交这个区块。可是D已经提交了bockX，因此损害了系统的安全性。注意，这里并没有任何拜占庭行为，仅仅是不同时性。 **锁**解决了这个问题通过强迫验证者粘附在他们预提交（pre-commit）的区块上，因为其他的验证者可能居于这个预提交进行了提交（如上例中的D）。本质上，在任何一个节点一旦存在超过2/3预提交（pre-commit），整个网络被锁定在这个区块上，也就是说在下一轮中无法产生一个不同块的波尔卡。这是预投票锁的直接动机。 每一个**pre-commit**（**预提交**）都必须由同一轮的波尔卡（polka）来证明。

问题：

1. 为什么不能用区块编号来区分区块，最终只提交统计超过2/3投票的区块？ 原因：tendermint不存在弱中心来统计投票确认的数量
2. 假设验证者全为非拜占庭节点，超过2/3的节点预投票后，1/3~2/3的节点预提交（预投票的节点网络出现问题掉线），剩下的的节点无法产生波尔卡，如何解锁？？
3. 为什么会产生多轮，怎样会触发新的一轮？

提议者的选举

满足必要要求Rx和可选要求Ox： **R1:Determinism** 在分布式系统中，对于同一高度同一轮，两个诚实节点生成的提议者是一样的（原文的process不知道是啥）

```
proposer_p(h,r) = proposer_q(h,r)

where proposer_p(h,r) is the proposer returned by the Proposer Selection Procedure
at process p, at height h and round r.
```

R2:Fairness 根据权重尽可能地“公平”

```
Given a validator set with total voting power P and a sequence S of elections. In
any sub-sequence of S with length C*P, a validator v must be elected as proposer
P/VP(v) times, i.e. with frequency:

f(v) ~ VP(v) / P

where C is a tolerance factor for validator set changes with following values:

C == 1 if there are no validator set changes
C ~ k when there are validator changes
```

算法

- 所有地验证者根据股权向前移动：通过投票权提升优先级
- 优先级队列的第一个被选为区块提议者
- 向后移动区块提议者：通过总投票权降低优先级

Validator	p1	p2
VP	1	3

节点集 算法流程图

Priority Run	-2	-1	0	1	2	3	4	5	Alg step
			p1,p2						Initialized to 0
run 1				p1		p2			A(i)+=VP(i)
		p2		p1					A(p2)-= P
run 2					p1,p2				A(i)+=VP(i)
	p1				p2				A(p1)-= P
run 3		p1						p2	A(i)+=VP(i)
		p1		p2					A(p2)-= P
run 4			p1				p2		A(i)+=VP(i)
			p1,p2						A(p2)-= P

节点集更改

1.投票权更改

依然按照原有算法进行

2.验证者缺失

为了是优先级总和保持为0，在原有算法上增添一步：每个验证者的优先级减去平均优先级

Validator	p1	p2	p3
VP	1	2	3

Priority Run	-3	-2	-1	0	1	2	4	Comment
last run	p3				p1	p2		remove p2
nextrun								
new step		p3				p1		A(i) -= avg, avg = -1
					p3	p1		A(i)+=VP(i)
			p1		p3			A(p1)-= P

3.验证者增添

首先，添加的验证者V的初始优先级为：

```
A(V) = -1.125*P, P为包括V投票权的总投票权
```

Validator	p1	p2	p3
VP	1	3	8

然后每个验证者再次减去平均优先级(1+3+8)$\approx$-13 A(p3)=-1.125*

Priority Run	-13	-9	-5	-2	-1	0	1	2	5	6	7	Alg step
last run				p2				p1				add p3
	p3			p2				p1				A(p3) = -4
next run		p3						p2		p1		A(i) -= avg, avg = -4
					p3				p2		p1	A(i)+=VP(i)
			p1		p3				p2			A(p1)-=P

-13

提议者选择的参考链接：Proposer selection procedure in Tendermint 想法： 选举验证者采用VRF，选举打包者采用上述算法