

共识论文分享：从PBFT到LibraBFT

腾讯 长安链研发工作组 基础组件开发二组 刘克猛

目录

1. 前置知识
2. 论文介绍
3. 算法对比

1 同步、异步概念

- **异步**(asynchrony)：系统中各个节点可能存在较大的时钟误差、消息传递时间是任意长的，各节点对消息的处理时间也可能是任意长的。
- **同步**(synchrony)：系统中各个节点的时钟误差存在上限、消息传递在一定时间内完成，各节点完成处理消息的时间是一定的。

2 FLP不可能原理

- 在异步通信网络中，只要存在一个故障节点，那么就不存在一个可终止的“正确的”一致性协议。
- “共识问题的正确性”：一致性、有效性、终止性。

3 部分同步假设 (PBFT、Tendermint、HotStuff、LibraBFT)

- **部分同步**(partial synchrony)：在一个全局稳定时间 (GST) 之前，系统可能处于异步状态，GST之后，会有一段同步的状态。

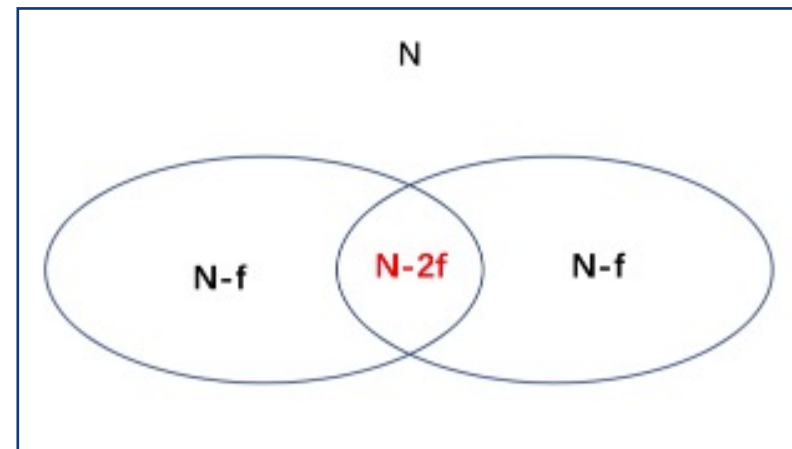
4 CAP定理

- 一个分布式系统不可能同时满足一致性（强一致性）、可用性和分区容错性，只能三选二。
分区容错性是分布式系统基本要求，因此共识算法将在强一致性与可用性，即安全性与活性之间权衡。

相关概念

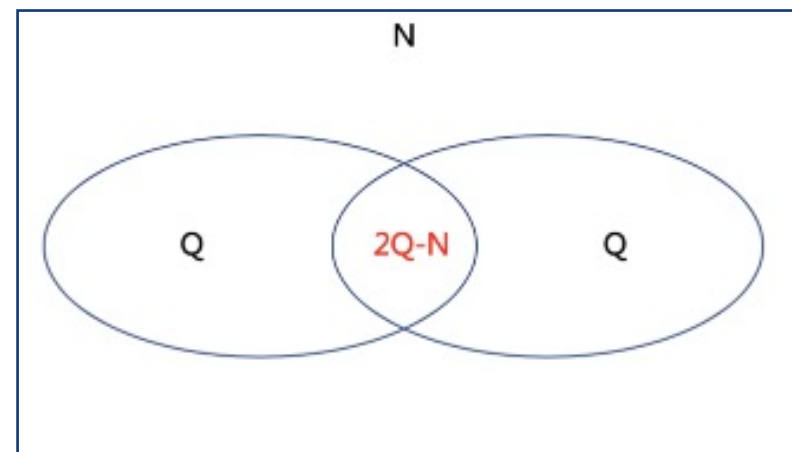
1 $N \geq 3f + 1$

- 因为有 f 个节点可能是拜占庭节点（即不作出响应），因此两次读写请求都需要在 $N - f$ 个节点返回响应后完成共识，这两次操作响应的节点交集至少有 $N - 2f$ 个。
- 为了保证不让 f 个拜占庭节点成功作恶，对外提供一致的应答，需要让交集满足 $N - 2f > f$ ，即 $N > 3f$ ，也即 $N \geq 3f + 1$ 。



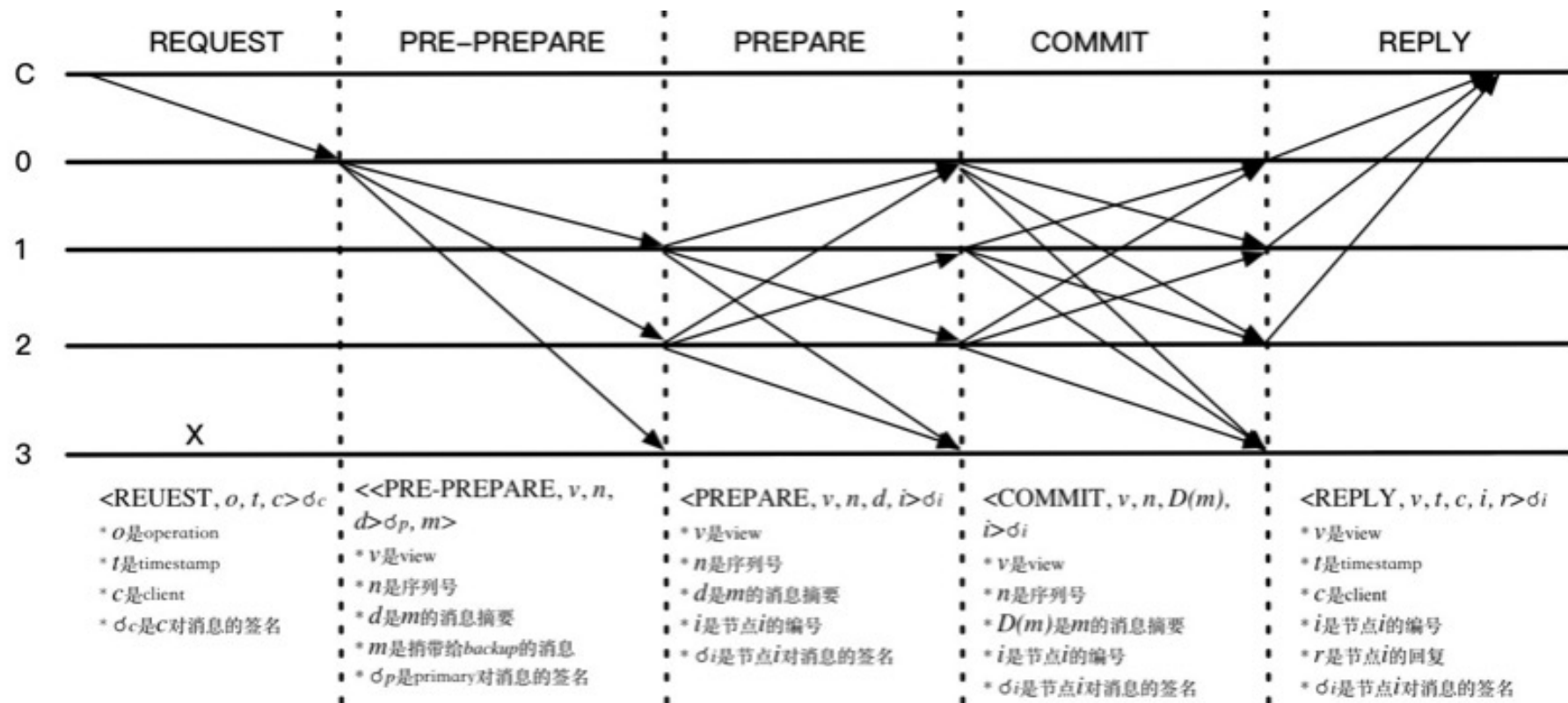
2 $\text{Quorum} \geq 2f + 1$

- 假设quorum的大小为 Q ，那么：任意两个quorum集合的交集大小至少为 $2Q - N$ ，交集中至少要有超过 f 个节点，因此： $2Q - N \geq f + 1$ ，即： $Q \geq (N + f + 1) / 2$ ，也即 $Q \geq 2f + 1$ （不严谨）。



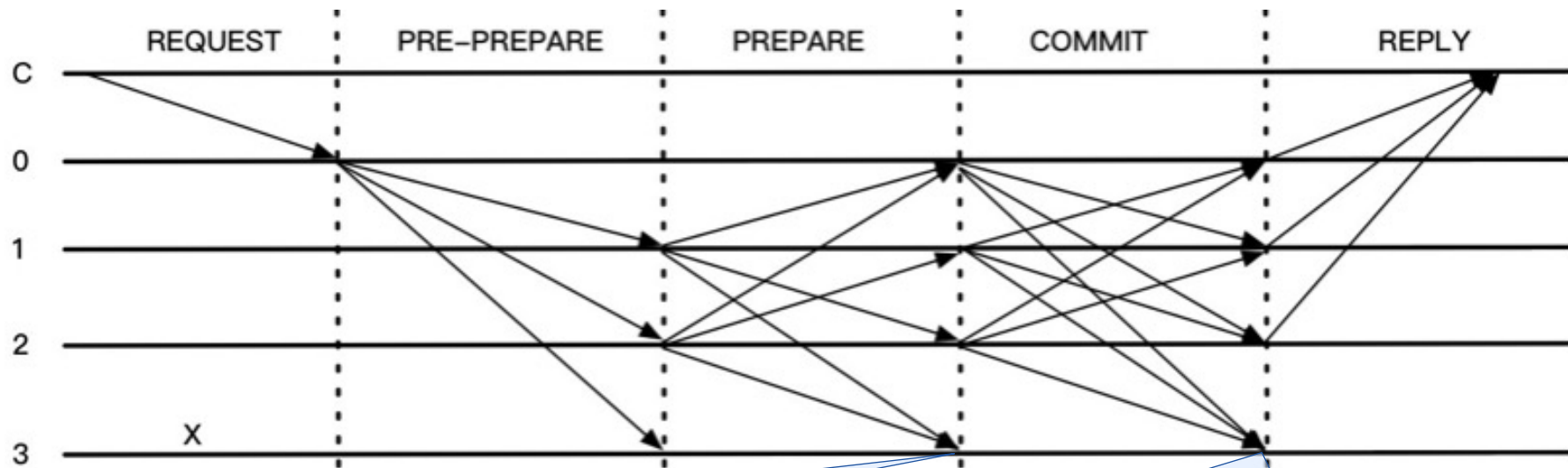
Normal-Case Operation

1. **视图** (View)：将时间分为一段一段连续的view，在一个view中有一个primary，其他节点是backups；
2. **主节点** (Primary)：决定client请求操作的执行顺序，并打包发给backups；
3. **副本节点** (Backups)：验证消息的合法性，在收到quorum个消息时，对共识的消息达成一致；



Normal-Case Operation

1. **视图** (View)：将时间分为一段一段连续的view，在一个view中有一个primary，其他节点是backups；
2. **主节点** (Primary)：决定client请求操作的执行顺序，并打包发给backups；
3. **副本节点** (Backups)：验证消息的合法性，在收到quorum个消息时，对共识的消息达成一致；



收到1个Pre-Prepare消息和2f个Prepare消息，
节点达到 ***Prepared***(m, v, n, i) 状态

收到2f+1个Commit消息，节点达到
Committed-local(m, v, n, i) 状态，
requestBatch达到 ***Committed***(m, v, n) 状态，此
时至少有f+1个诚实的节点到达了 ***Prepared*** 状态

Garbage Collection

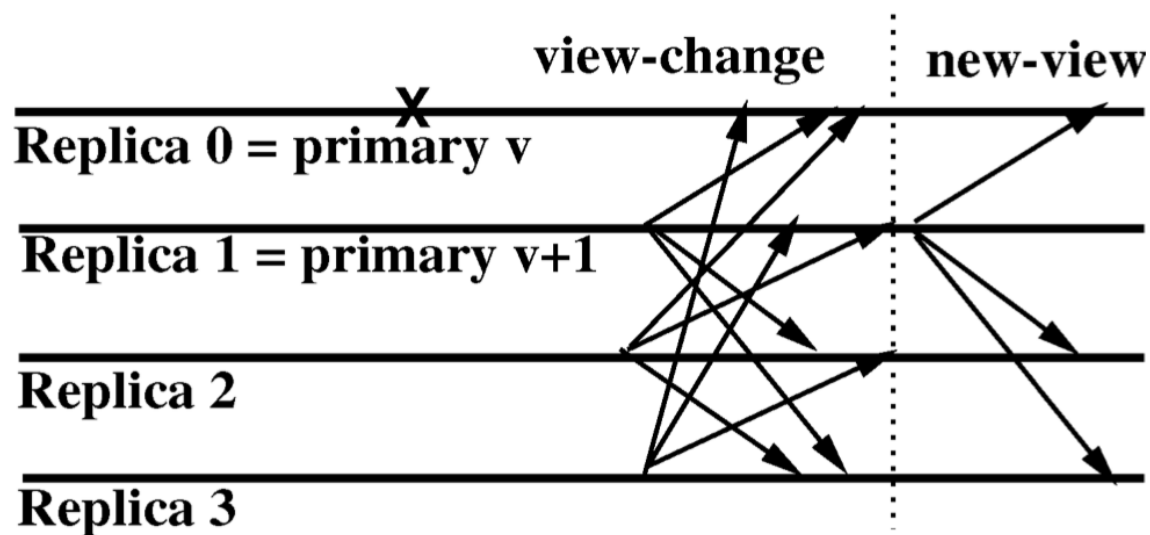
1. PBFT将定时（每执行K个requestBatch）产生checkpoint记录当前链状态，并向其他节点组播
 $\langle \text{CHECKPOINT}, n, d, i \rangle \sigma_i$ ，**n**是“最近”被执行的requestBatch的编号，**d**是这个batch执行后链对应的状态的哈希；
2. 当节点收集到 **2f+1** checkpoint，并且序号n和d是不冲突的，这个n被认为是**稳定检查点**。此时，节点会丢弃所有小于等于n的log entries和更早的checkpoints。
3. 然后，会更新自己low、high water marks：
 - $h = n$ ；
 - $H = h + L$ ($L = K * m$ ，m为常数，通常K=10，m=4)

作用：

1. 清除共识过程中产生的已经过期的log entries，减少缓存；
2. 在checkpoint过程中，可以及时发现自己落后了，执行stateUpdate进行状态同步；
3. 在viewchange过程中，作为下一个视图共识从哪个序号继续进行的依据；

View Changes

1. **View-change阶段**：(1)当主节点超时没有产生PrePrepare消息，(2)或者在Prepare阶段验证主节点产生的PrePrepare消息不合法或者冲突，(3)或者在Prepare、Commit阶段没在规定时间内收到 $2f + 1$ 个一致合法的消息。则发送 $\langle \text{VIEW-CHANGE}, v+1, n, \mathbf{C}, \mathbf{P}, i \rangle_{\sigma_i}$ 消息，其中 n 是节点 i 的最新稳定检查点， \mathbf{C} 是 $2f + 1$ 个有效checkpoint消息， \mathbf{P} 是序号大于 n 的到达 *Prepared* 状态的消息集合， \mathbf{P}_m 由关于 m 的1个PrePrepare消息和 $2f + 1$ 个prepared消息组成。



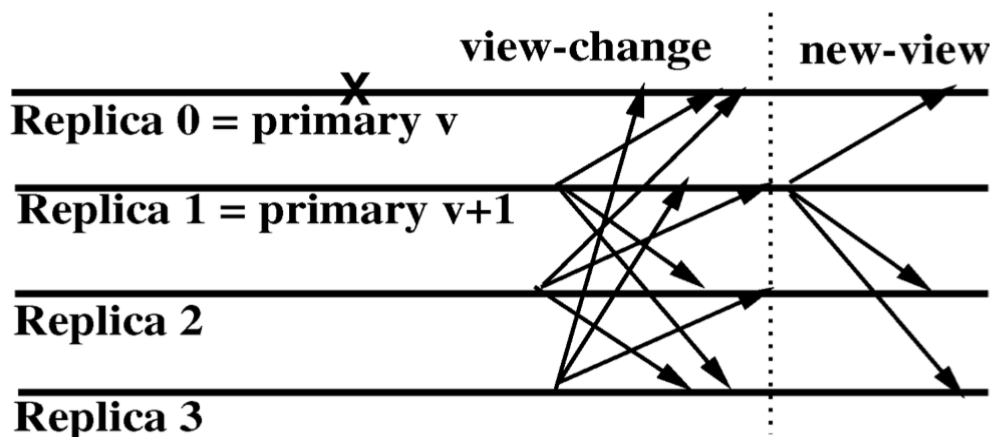
View Changes

2. **New-view阶段**：当view+1所对应的New primary收到了 $2f+1$ 个有效的view-change消息，它就会广播 $\langle \text{NEW-VIEW}, v+1, V, O \rangle$ 消息，其中 V 是它收到的 $2f+1$ 个view-change消息，证明new-view的正确性， O 是新生成的 pre-prepare消息的集合，新的主节点按照如下过程计算 O ：

- 根据view-changes，将最近稳定检查点作为 **min-s**，最高的prepared块的序号作为 **max-s**；
- 对介于 **min-s** 和**max-s** 之间的每个序号n创建新的pre-prepare消息：

如果 P 中存在一个 P_m 序号为n，那么 $\langle \text{PRE-PREPARE}, v+1, n, d \rangle \sigma_i$

如果不存在，那么就放一个空块， $\langle \text{PRE-PREPARE}, v+1, n, d^{\text{null}} \rangle \sigma_i$



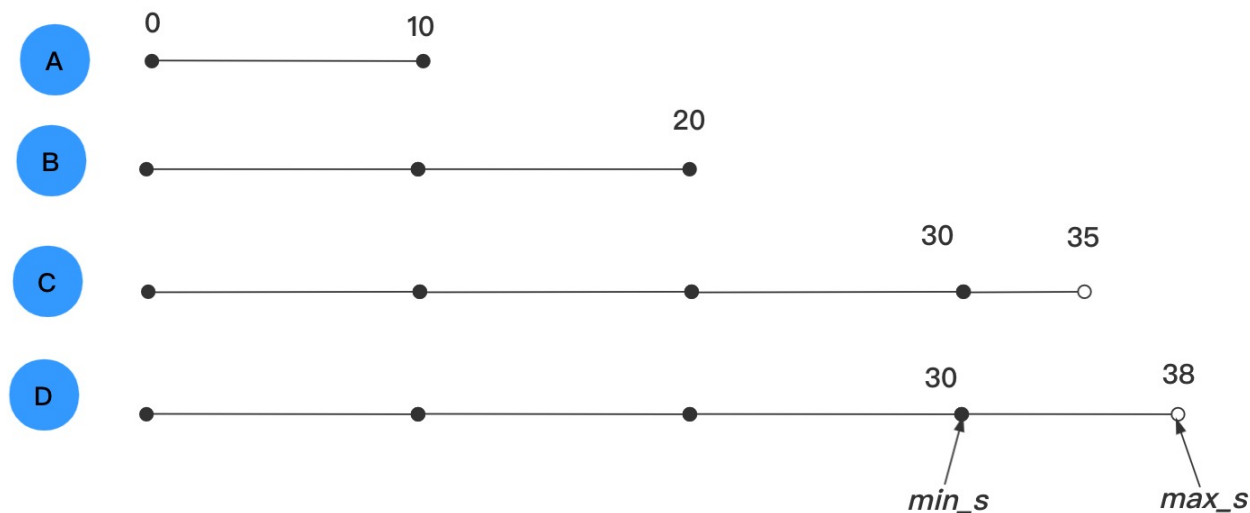
View Changes

2. New-view阶段：当view+1所对应的New primary收到了 $2f$ 个有效的view-change消息，它就会广播 $\langle \text{NEW-VIEW}, v+1, V, O \rangle$ 消息，其中 V 是它收到的 $2f$ 个view-change消息，证明new-view的正确性， O 是新生成的 pre-prepare消息的集合，新的主节点按照如下过程计算 O ：

- 根据view-changes，将最近稳定检查点作为 ***min-s***，最高的prepared块的序号作为 ***max-s***；
- 对介于 ***min-s*** 和***max-s*** 之间的每个序号n创建新的pre-prepare消息：

如果 P 中存在一个 P_m 序号为 n ，那么 $\langle \text{PRE-PREPARE}, v+1, n, d \rangle \sigma_i$

如果不存在，那么就放一个空块， $\langle \text{PRE-PREPARE}, v+1, n, d^{\text{null}} \rangle \sigma_i$



Safety and Liveness

1 Safety: 所有诚实的节点对于到达committed的消息的序号达成一致

- 在一个视图内，不会有两个requestBatch（序号都是 n ）都到达**Prepared**状态；
- 在不同的视图间，通过视图转换 $\langle \text{VIEW-CHANGE}, v+1, n, C, P, i \rangle \sigma_i$ ，把到达**Prepared**状态的requestBatch“显式”地传到下一个视图，这样保障所有到达**committed**状态的requestBatch在诚实的节点间序号是一致的；

2 Liveness: 不能对一个requestBatch达成共识时，需要进入到下一个视图

（还需要保证至少 $2f+1$ 个诚实节点有足够的时间进入到这个视图）

- 视图转换也保障了活性；
- 为了避免视图转换太频繁，保障所有的诚实节点都有足够的时间转换到下一个视图。每一次视图切换设置一个定时器，并且下一次视图转换定时器时间为上次的double；
- 为了避免视图转换太慢了，收到 $f+1$ 个有效的view-change，节点就会发送view-change；

优缺点总结

1 优势:

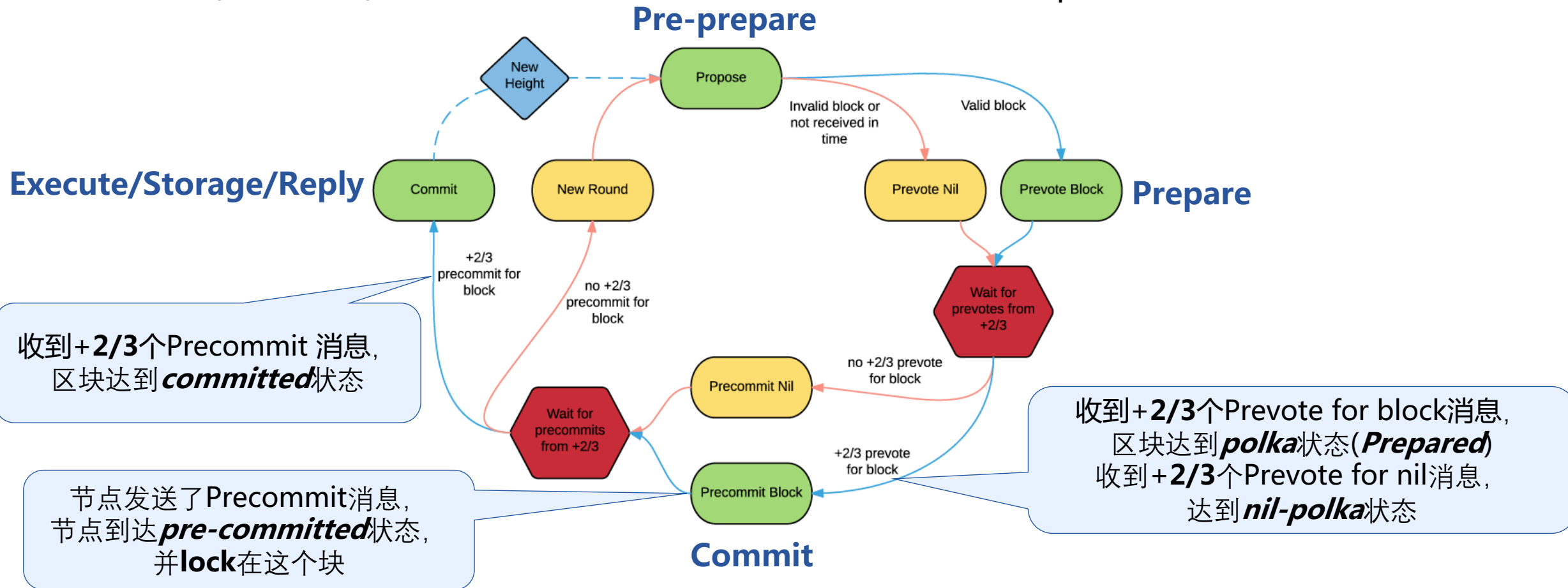
- 首个实用的BFT容错算法，使得复杂度从指数级降到多项式级 $O(n^2)$ ；
- 响应性（Responsiveness）：在网络状况OK，主节点不犯错，拜占庭节点数量不超过 f 个时，在下一个视图一定是可以达成共识的；

2 不足:

- 通信复杂度 $O(n^2)$ ，可扩展性低；
- 视图转换的复杂度 $O(n^3)$ ，要是连续视图转换复杂度 $O(f * n^3)$ ，视图转换复杂，很“重”；
- 视图转换期间只能接收viewchange、newview消息不能对外提供服务；
- Safety和Liveness耦合，都需要通过视图转换来保障；
- 去重机制没有提；

Consensus Process

1. **轮次**(Round)：类似但不同于View，Round是基于某一具体高度的块，新的块Round会置为0；
2. **提案者** (Proposer)：打包出块节点，同PBFT中的Primary；
3. **验证者**(Validators)：所有的共识节点都是验证者，同PBFT中的Backups；



Safety and Liveness

1 Prevote-the-Lock:

- 验证者在之后的round中只会Prevote给它锁定的那个块（ *pre-committed* 的块 ）；
- 如果成为提案者，他会propose它锁定的那个块；

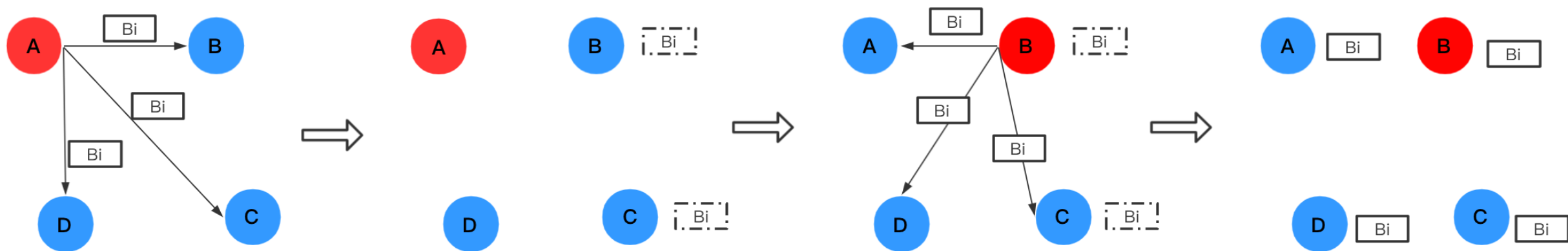
2 Unlock-on-Polka:

- 当验证者收到一个round比自己所处的轮次大，并且到达*polka*状态的块时，接收这个块作为这个高度的块，并释放之前锁定的那个块；

Safety and Liveness

1 Prevote-the-Lock:

- 验证者在之后的round中只会Prevote给它锁定的那个块 (*pre-committed* 的块) ；
- 如果成为提案者，他会propose它锁定的那个块 ；



Safety and Liveness

1 Prevote-the-Lock:

- 验证者在之后的round中只会Prevote给它锁定的那个块（ *pre-committed* 的块）；
- 如果成为提案者，他会propose它锁定的那个块；

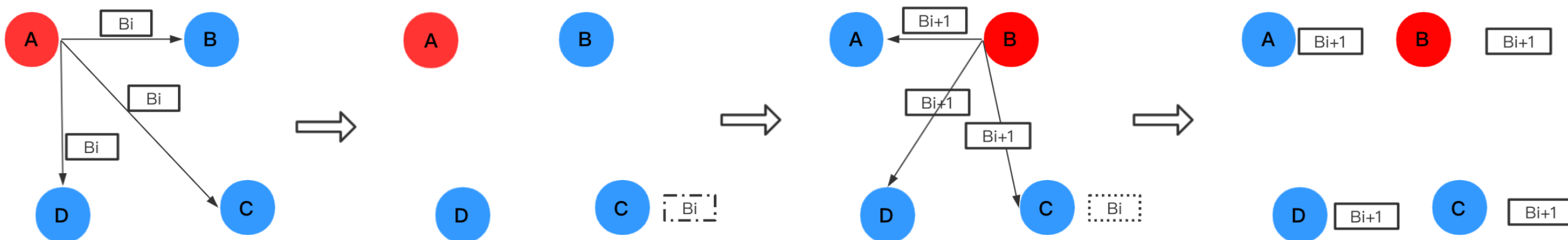
2 Unlock-on-Polka:

- 当验证者收到一个round比自己所处的轮次大，并且到达*polka*状态的块时，接收这个块作为这个高度的块，并释放之前锁定的那个块；

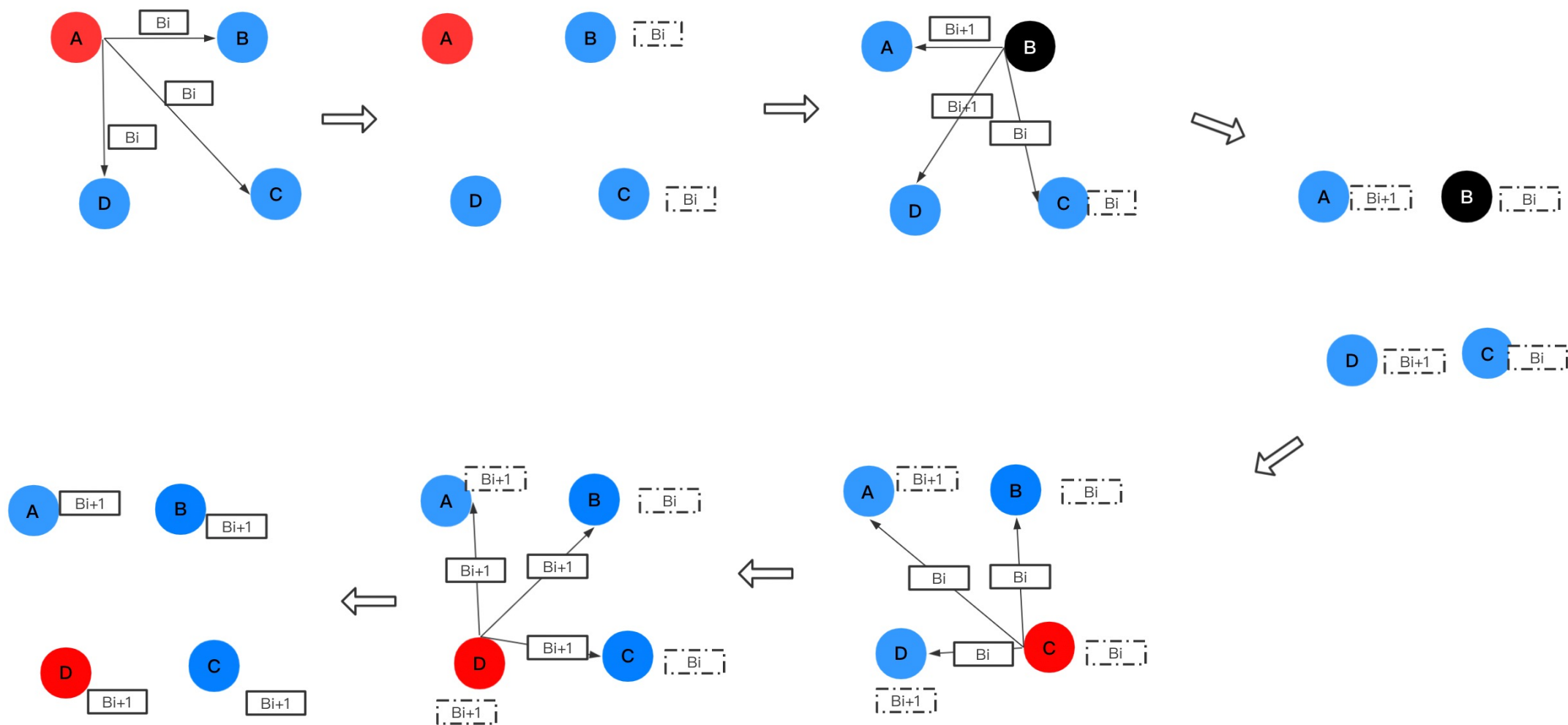
Safety and Liveness

2 Unlock-on-Polka:

- 当验证者收到一个round比自己所处的轮次大，并且到达`polka`状态的块时，接收这个块作为这个高度的块，并释放之前锁定的那个块；



No Responsiveness



优缺点总结

1 优势:

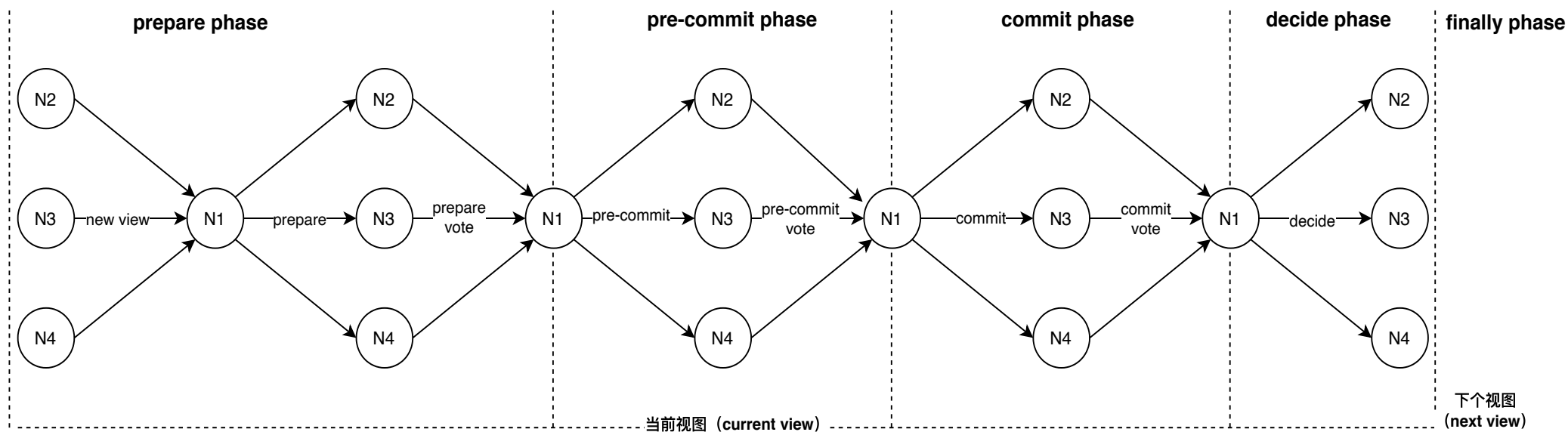
- 用锁的机制代替了PBFT中复杂的viewChange，视图切换不再依赖复杂的网络通信，但是丧失了响应性；
- 视图切换时，不在需要piggybacked new view时的消息；
- 工程化成功，可直接基于其开源的框架搭建一条链；
- 联盟链和公有链均可用；

2 不足:

- 丧失响应性：没有“显式”地将*polka*状态的proposals传到下一个round；
- Safety和Liveness耦合：加锁、解锁机制耦合在一起；

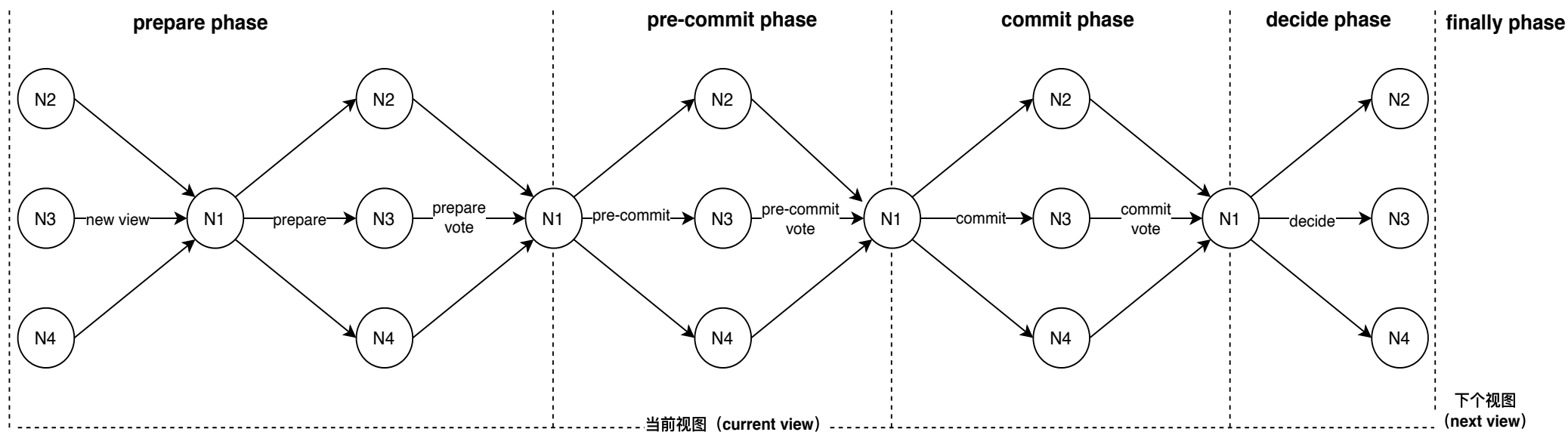
Model

1. **视图**(View)：同PBFT中的View，严格递增；
2. **主节点** (Leader)：打包定序、出块节点，同PBFT中的Primary；
3. **从节点**(Replicas)：接收、验证主节点发送过来的消息，并发送投票消息，同PBFT中的Backups；



Basic HotStuff

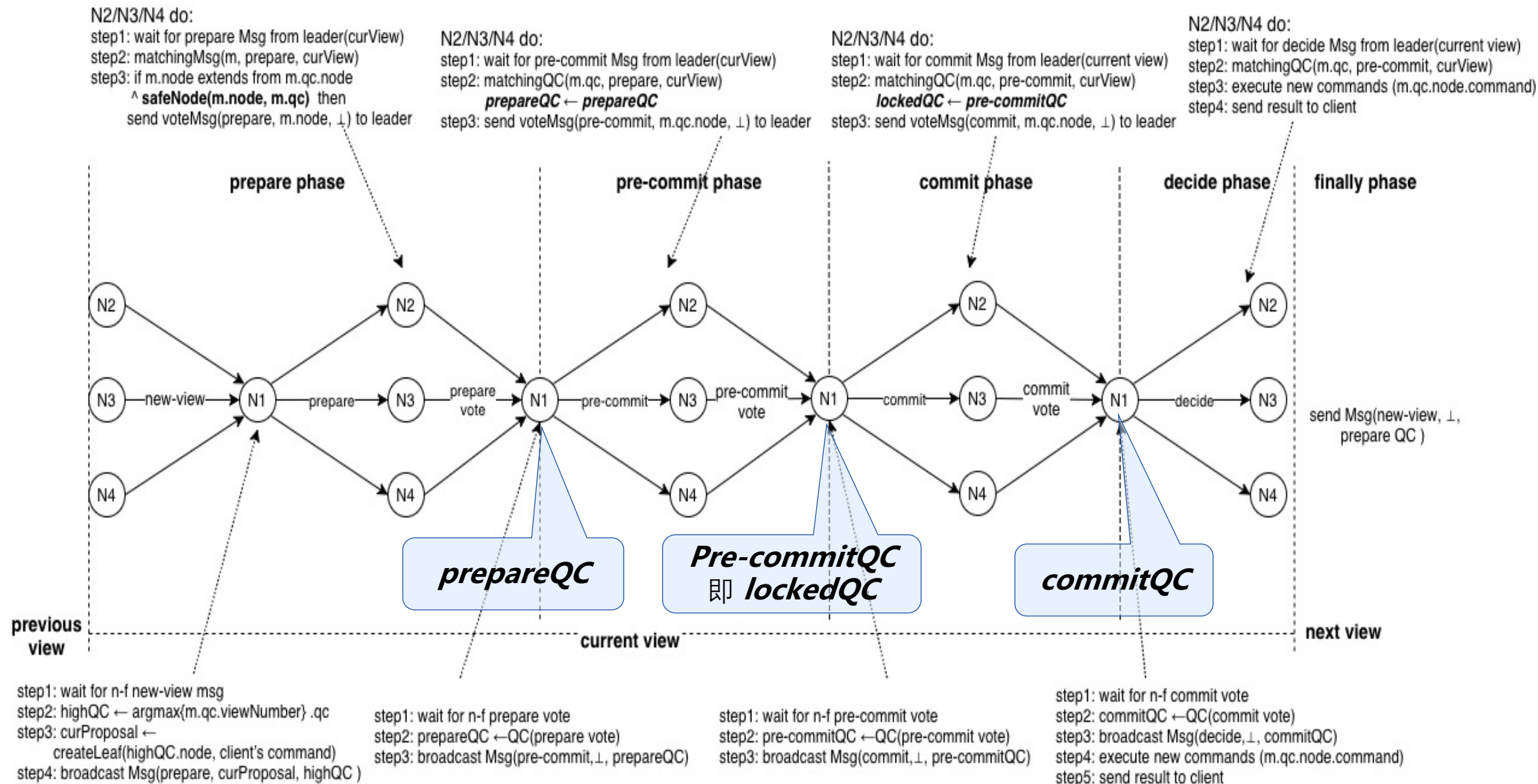
1. Basic HotStuff整个过程包括5个阶段：准备阶段（PREPARE）、预提交阶段（PRE-COMMIT）、提交阶段（COMMIT）、决定阶段（DECIDE）和最终阶段（FINALLY）。
2. 所有共识消息通过**签名**来证实其有效性，使得每一轮交互**只需要主节点**进行广播，然后收集n-f个节点对同一个块提案的投票消息，利用门限签名将其合成一个QC，网络复杂度降低至 **$O(n)$** 线性复杂度。



QC (quorum certificate, 证书)：主节点收到n-f个节点对同一个区块提案的投票消息（带节点签名）后，利用门限签名将其合成一个QC证书。

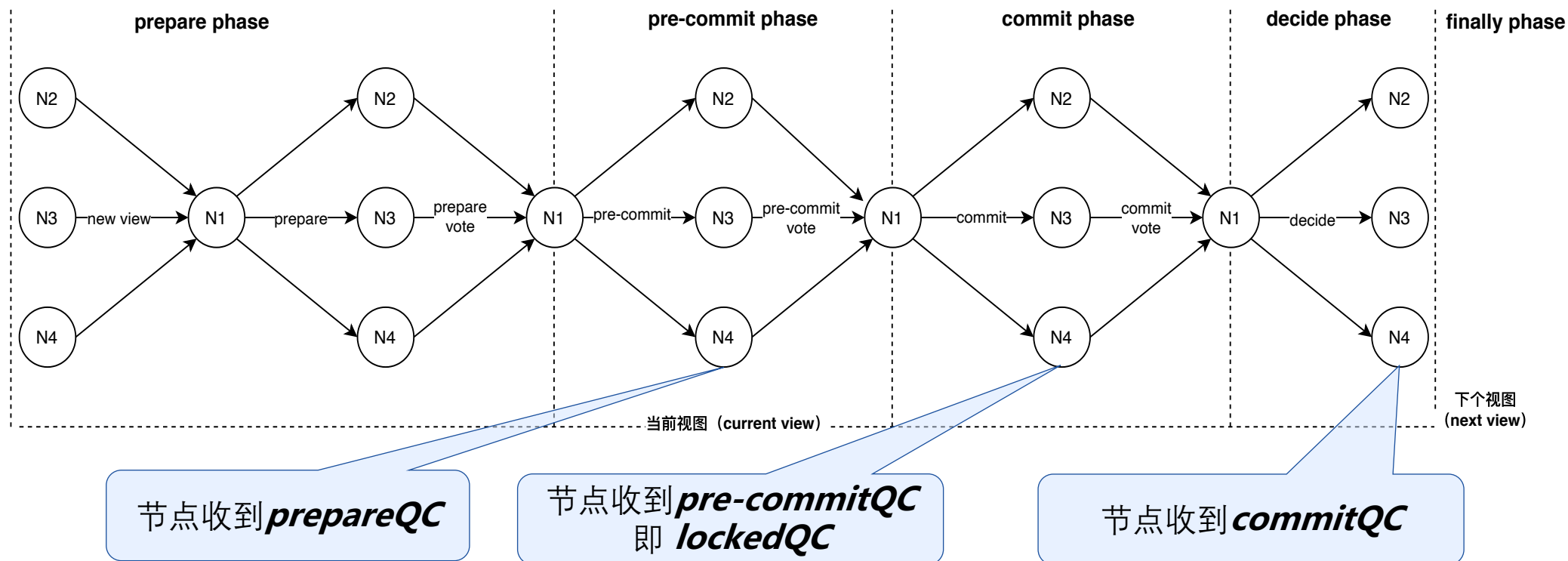
2. 论文介绍

HotStuff算法



SafeNode predicate

1. **Safety Rule** : Proposal消息中的block(**m.block**)是从本节点**lockedQC**对应的区块扩展产生;
2. **Liveness Rule** : 当Proposal的highQC(**m.justify**)高于本节点**lockedQC**中的viewNumber时也会接收该proposal ;



Safety and Liveness

1 Safety:

- 同一个View下，不会对冲突的区块，产生相同类型的QC；

这样的话至少有一个诚实节点给两个冲突的块都投票了；

- 正常情况，Replica不会对两个冲突的区块到达committed；

同一个视图下，正常的replica不会对冲突的区块产生commitQC，所以不会commit冲突的区块。

在不同的视图下，假设viewNumber在v1和v2时 ($v1 < v2$)，commit了冲突的区块，即存在commitQC_1 = {block1, v1}, commitQC_2={block2, v2}，且block1与block2冲突。在v1和v2之间,肯定存在一个最小的 $v_s(v1 < v_s \leq v2)$ ，使得 v_s 下存在有效的prepareQC_s{block_s, v_s},其中block_s与block1冲突。当含有block_s的prepare被广播后，节点会对该消息做safety验证。

由于block_s与block1冲突，所以显然，不符合safety规则1；

假如block_s.parent.viewNumber > block_1.viewNumber，那么显然block_s.parent与block_1冲突，所以block_s.parent是更早的与block1冲突的，这与 v_s 最小矛盾。

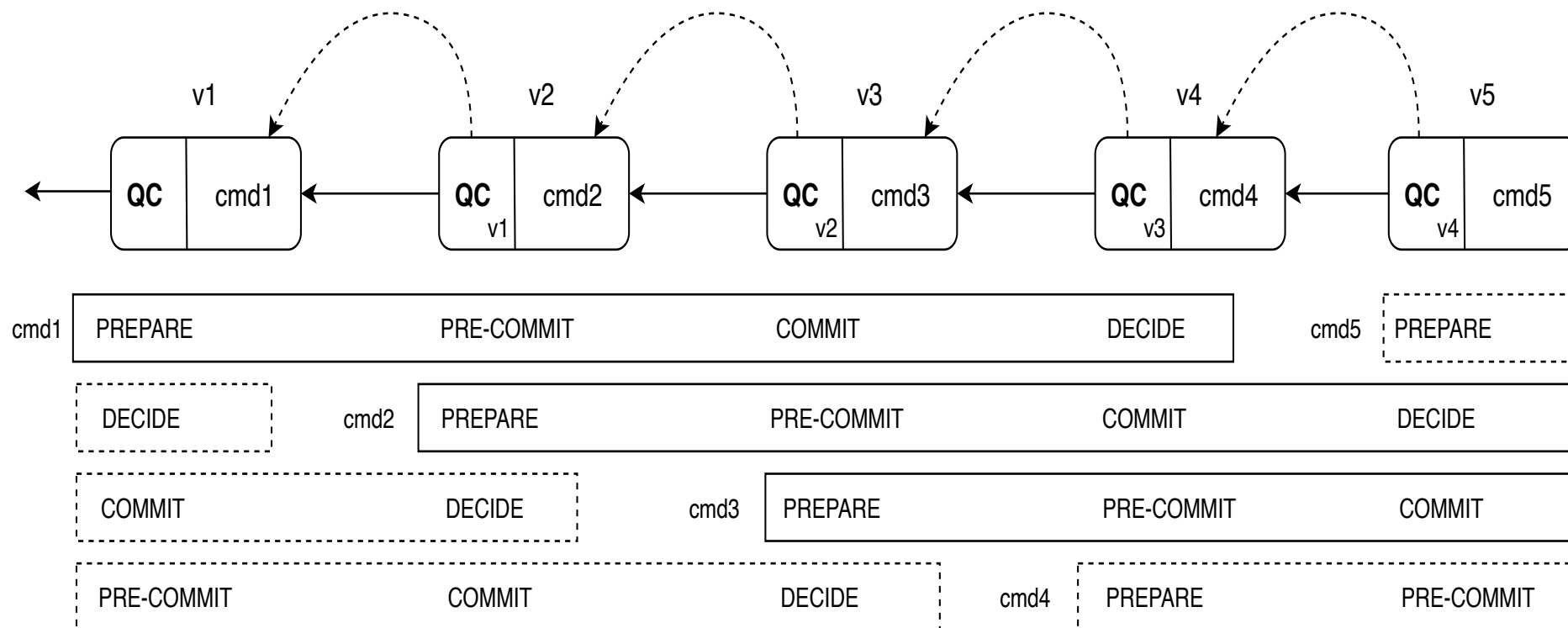
Safety and Liveness

2 Liveness :

- 有一个节点对这个块到了lockedQC，那么就有 $f+1$ 个诚实的节点对这个块到达了prepareQC，这样就会至少有一个诚实的节点把这个prepareQC状态的块传到下一个视图。就可以基于这个prepareQC作为highQC构造新的块。
- 并且如果在GST之后，主节点是诚实节点，拜占庭节点小于 f 个，那么一定会达成共识，满足optimistical responsive。

Chained HotStuff

Basic HotStuff各个阶段的流程高度相似，HotStuff作者便提出Chained HotStuff来简化Basic HotStuff的消息类型，并允许Basic HotStuff的各阶段进行流水线（pipelining）处理。



优缺点总结

1 优势:

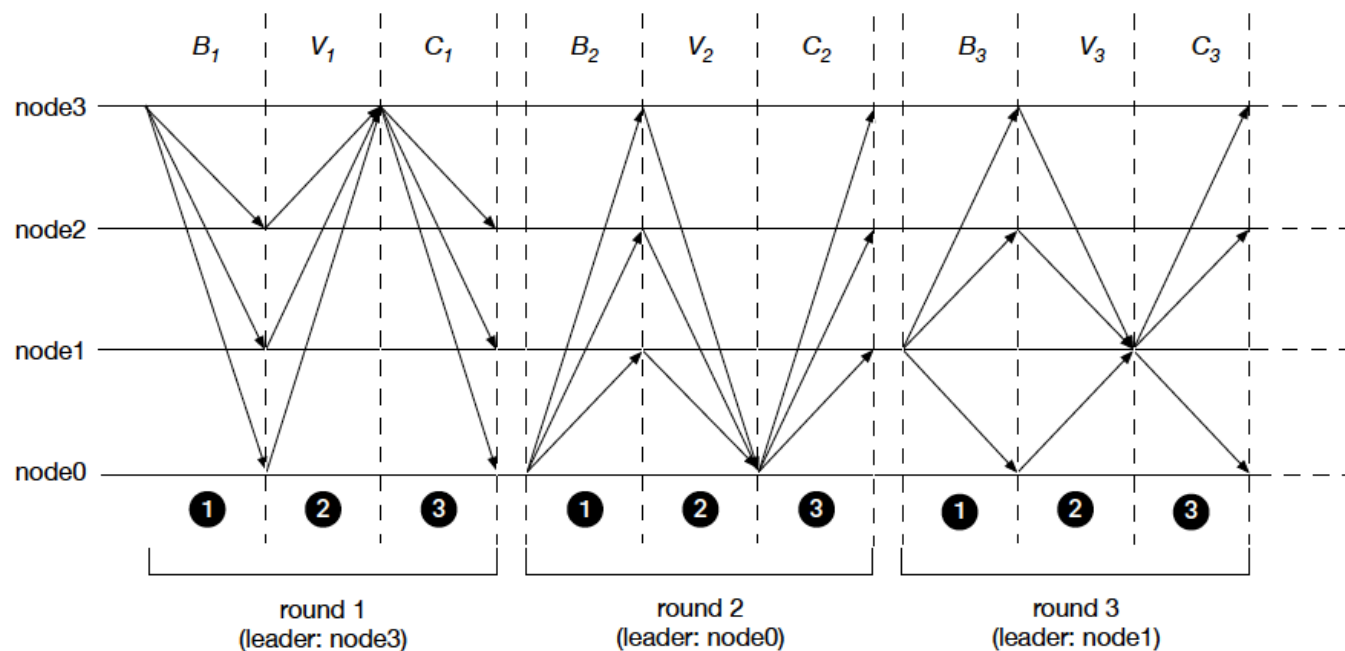
- 采用门限签名将通信复杂度从 $O(n^2)$ 降低为 $O(n)$ ；
- 线性视图转换，把轮换主节点融入了常规共识流程中，切换主节点无需增加其他协议和代价，且系统在此期间还能继续对外提供服务，并且兼具响应性；
- 将安全性和活性解耦，活性组件Pacemaker作为一个工作模块独立于共识协议的安全条件，这极大地降低了HotStuff定制活性机制的难度，无需修改协议本身的内容便可获取更多的活性保障；

2 不足:

- 活性机制灵活性不够：只有一个全局超时时间，没有为共识流程中的每个步骤设置完善的超时机制；
- 没有提到去重机制；

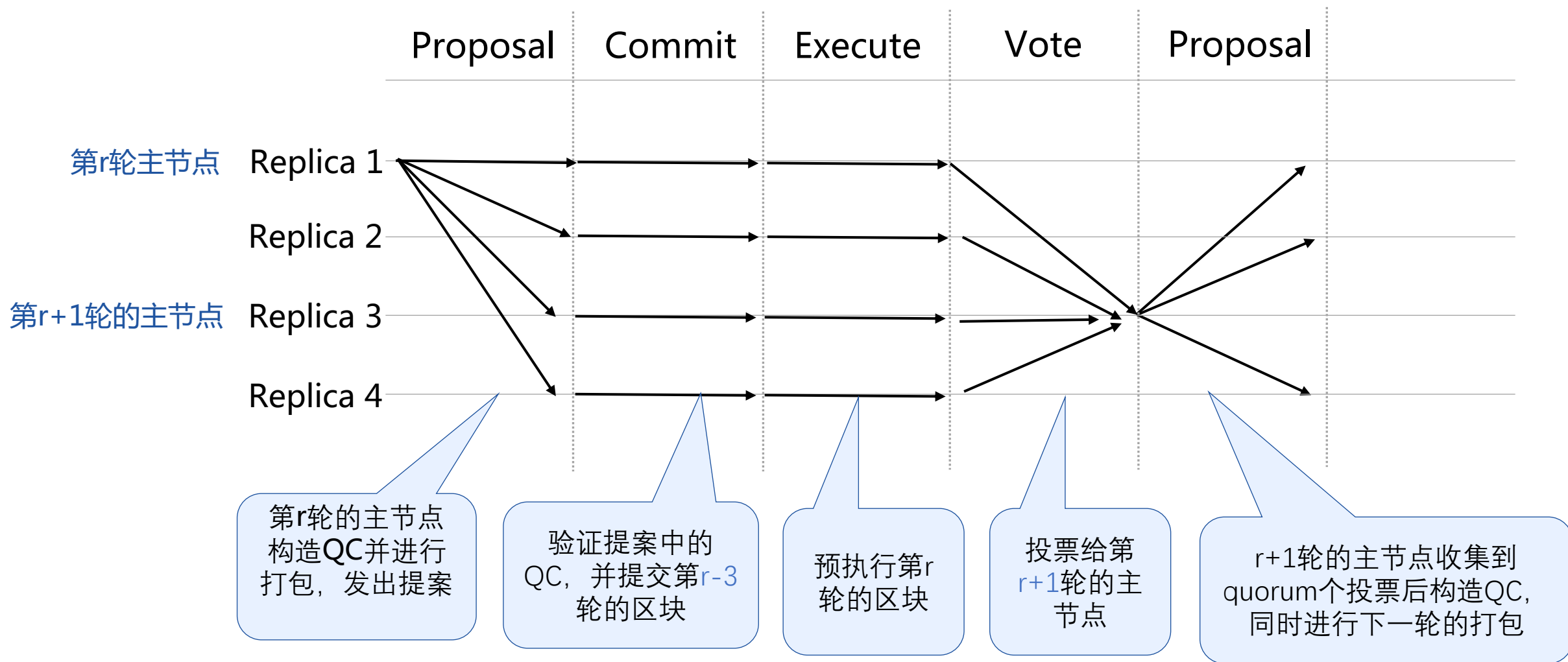
相关概念

1. **轮次**(Round)：同HotStuff中的流水线形式下的轮次投票，严格递增；
2. **主节点** (Leader)：打包定序、出块节点，同HotStuff中的Leader；
3. **验证者**(Validator)：接收、验证主节点发送过来的消息，并发送投票消息，同HotStuff中的Validator；



主流程

共识模块主要负责交易的定序与执行结果的一致性校验，通过3-chain达成提交条件：



优缺点总结

1 优势:

- 共识的是区块的状态，及执行的结果，而不仅仅是交易的顺序，增强了鲁棒性，也可以允许客户端使用QC验证从数据库里读出的数据；
- 扩展了HotStuff的Pacemaker，通过Pacemaker来发出明确的超时信号，验证者通过它发出的消息自动进入到下一个视图；
- Leader的选举变得不可预测，以最新提交的区块信息作为种子生成一个可验证的随机函数来判断自己是否成为下一个节点；
- 采用了交易缓冲池来进行交易去重，

2 不足:

- 去重机制简单，只有新提交交易的序列号比该账户最新提交的交易的序列号大一位，Libra才会将这笔交易加入共识提案中，造成同一账户下的交易之间产生了依赖性；
- 采用了Ed25519签名算法，但是没有实现聚合；

算法对比

Protocol	Authenticator complexity			Responsiveness
	<i>Correct leader</i>	<i>Leader failure (view-change)</i>	<i>f leader failures</i>	
DLS [25]	$O(n^4)$	$O(n^4)$	$O(n^4)$	
PBFT [20]	$O(n^2)$	$O(n^3)$	$O(fn^3)$	✓
SBFT [30]	$O(n)$	$O(n^2)$	$O(fn^2)$	✓
Tendermint [15] / Casper [17]	$O(n^2)$	$O(n^2)$	$O(fn^2)$	
Tendermint* / Casper*	$O(n)$	$O(n)$	$O(fn)$	
HotStuff	$O(n)$	$O(n)$	$O(fn)$	✓

■ 联盟区块链共识算法发展趋势

目前联盟区块链中拜占庭共识算法分为两类：

- **同步共识**：PBFT -> Tendermint -> HotStuff -> LibraBFT ...

目前已经有了一些基于HotStuff的进一步优化论文，比如2-chain的论文，还有采用其他聚合签名算法的论文。

- **异步共识**：HoneybadgerBFT(2018) -> Dumbo(2020)-> Dumbo-MVBA(2020)

国内蚂蚁链和京东链分别用了HoneybadgerBFT、和Dumbo。

- **BFT+TEE**：FastBFT，将 $2f+1$ 降到 $f+1$ ，采用树形分发进行共识消息的传播，降低网络复杂度。

追踪作者：

- Down song团队、新泽西理工唐强、软件所张振峰、浙大刘建。

共识的研究点：

- 共识需要解决的两大问题：提高tps、降低时延。
- 共识瓶颈在哪里？主节点的出口带宽、共识过程消息通信的复杂度。

谢谢大家！