Practical Byzantine Fault Tolerance

主要内容

- PBFT 论文概述
- 系统模型与相关假设
- 算法
- 垃圾回收
- View change
- 三阶段协议总结
- Safety & Liveness

PBFT 概述

- PBFT是Practical Byzantine Fault Tolerance的缩写。该算法是Miguel Castro (卡斯特罗)和Barbara Liskov（利斯科夫）在1999年提出来的，它解决了原始拜占庭容错算法效率不高的问题。早期的拜占庭容错算法有的是基于系统同步的假设，有的则是由于性能太低而不能在实际系统中运作，而PBFT是基于真实网络环境、实用的拜占庭容错算法。
- 该算法可以工作在异步环境中，可以承受小于1/3的节点同时faulty，为系统提供safety 和 liveness

为什么可以容忍小于1/3的恶意节点

f：拜占庭节点所占比例

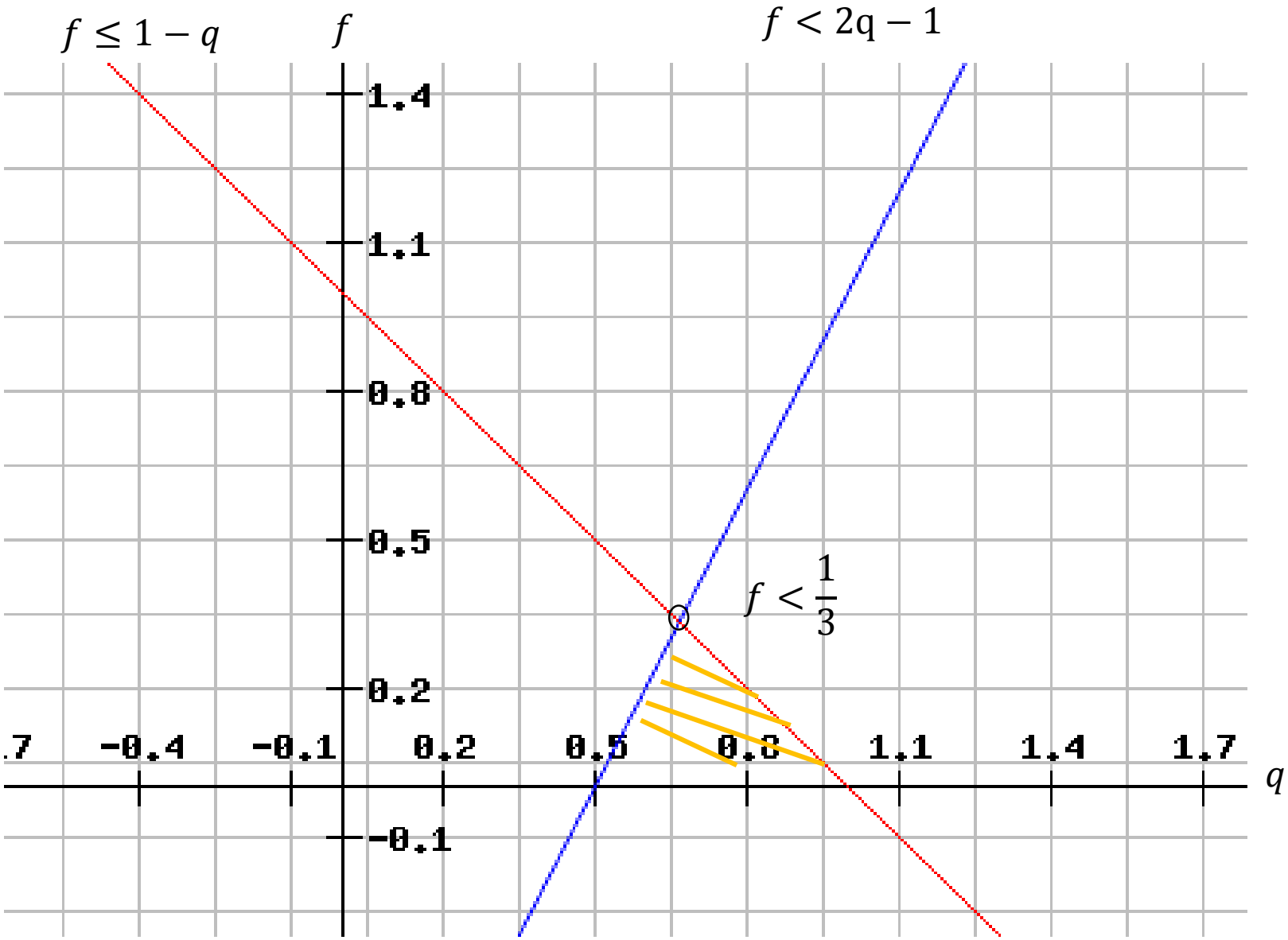
$1-f$：善意节点比例

q：一致投票达到这个比例后，才算达成一致

因此有：

1. $f+\frac{1-f}{2}<q$

2. $1-f \geq q$


$f \leq 1 - q$
f
$f < 2q - 1$
1.4
1.1
0.8
0.5
0.2
-0.1
.7
-0.4
-0.1
0.2
0.5
0.8
1.1
1.4
1.7
q
$f < \frac{1}{3}$

System model & Assumption

System model

- 异步分布式系统，节点由网络连接，真实的网络环境，会出现消息丢失、延迟、重复、乱序
- Byzantine Failure model：faulty的节点may behave arbitrarily .

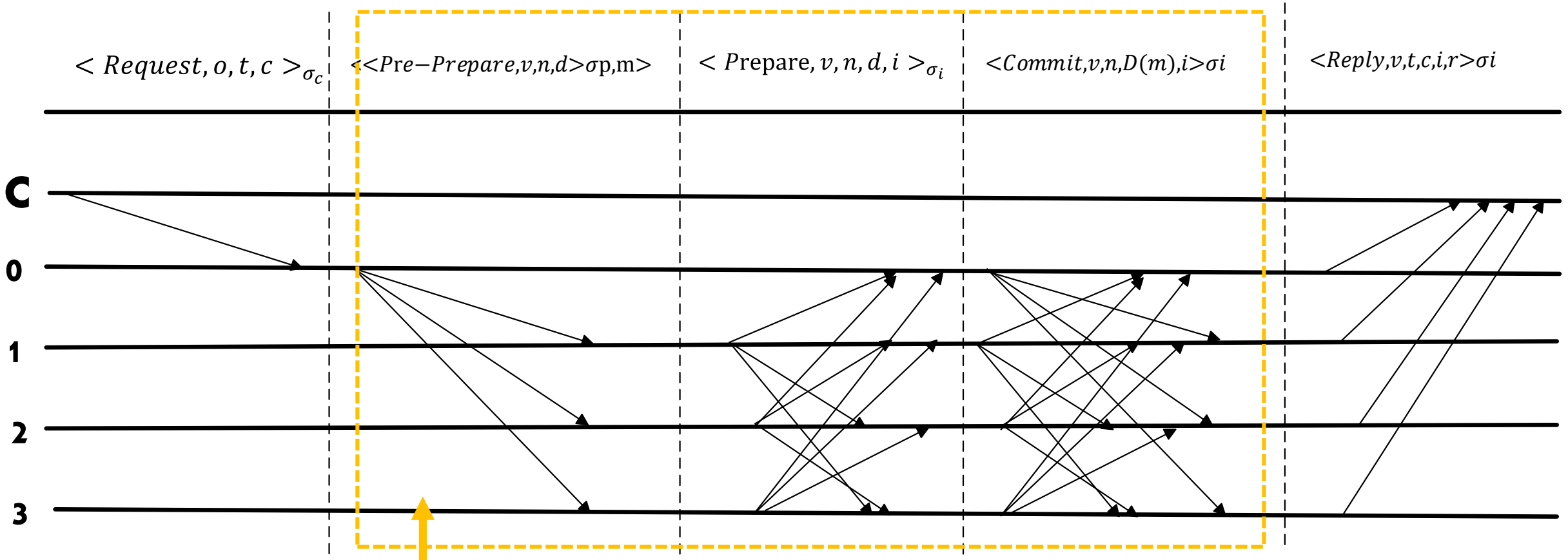
Assumption

- 节点间失效独立，不互相影响
- 无法推翻依赖的密码学保障
- 攻击者无法无限延迟non-faulty的节点
- 发送方一直重发请求，直到被接收

符号说明

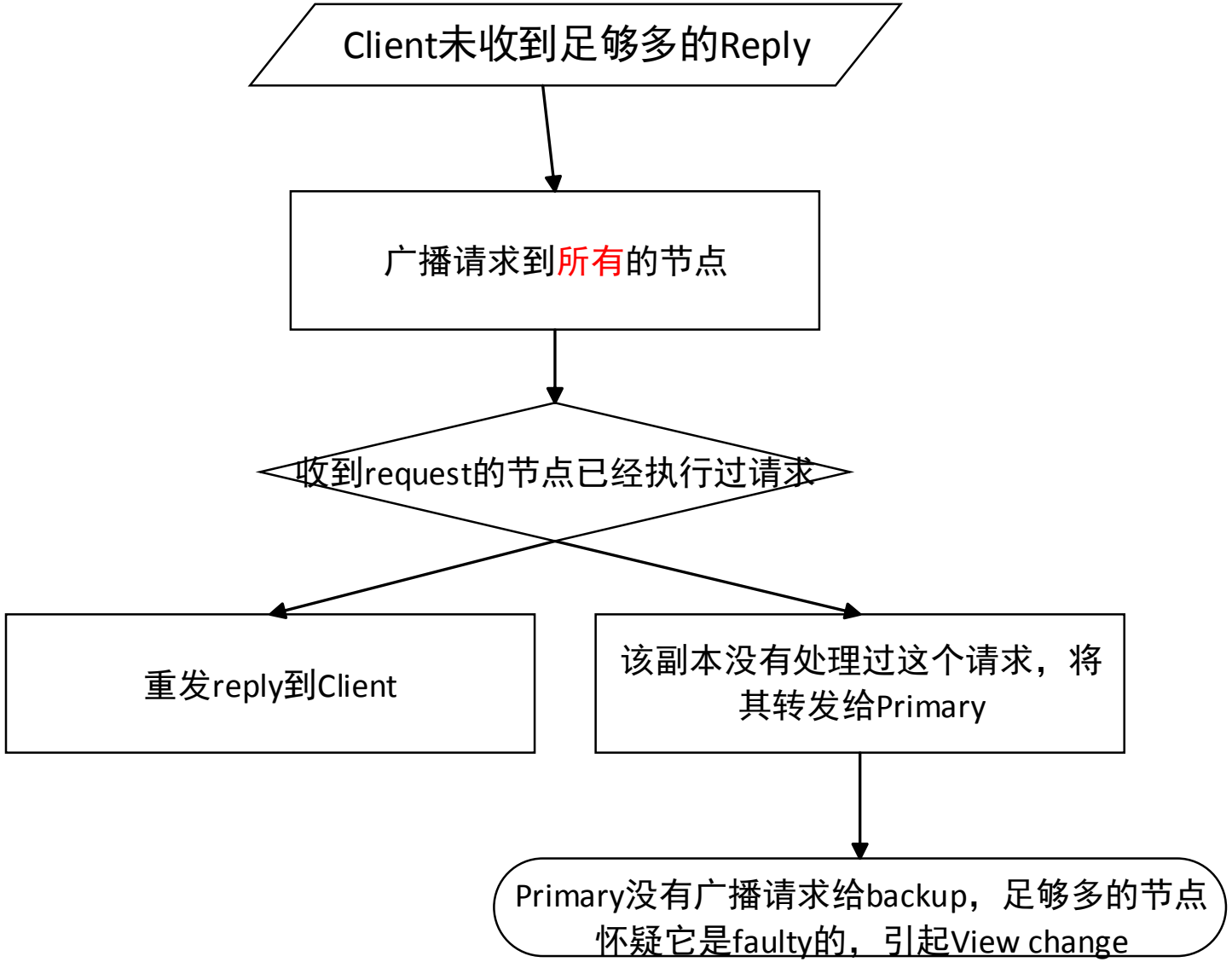
- m: 消息
- $D(m)$: 消息摘要
- $< m > \sigma_i$ ： 节点i对消息m签名
- $R = 3f + 1$ ： 至多f个节点是faulty的，R为总的节点个数，0~$R - 1$编号
- $p = v \ mod \ |R|$ ：当view number是 v时，应为primary节点的序号

Normal Case Operation



① Client 发送请求到当前view下的primary
② Primary广播请求到backups节点，开始三阶段协议（具体见后面）
③ 节点执行请求，并直接发送reply到Client
④ Client等待f +1个来自不同节点、但具有相同结果的reply，即为请求的执行结果

Client未收到足够多的Reply



三阶段协议

Pre-Prepare ： Primary广播 $<< Pre-Prepare, v, n, d >_{\sigma_p,}$ m $>$

Prepare: 若Backup 接受 Pre-Prepare message,进入到Prepare阶段，广播 $< \text{Prepare}, v, n, d, i >_{\sigma_i}$ 到**所有其他**节点,将pre-prepare和prepare消息写入自己的消息日志。任何副本节点接收到prepare消息后，会验证v, σ_i, n

Prepare为真(Prepared状态)：

replica i将prepare消息插入到日志中

接收到 $2f$ 个不同backup节点的prepare消息（与pre-prepare消息相匹配）

Commit ：当**prepare 为真**时，向其他节点广播 $< Commit, v, n, D(m), i >_{\sigma_i}$
开始Commit阶段，其他节点验证并且接受commit消息，则该副本节点将Commit消息写入消息日志中。

Committed（m,v,n）为真：任意$f+1$个non-faulty Backup集合中的所有副本节点都处于prepared状态；

Committed-local(m,v,n,i)为真：Backup i处于prepared状态，并且节点i已经接受了$2f+1$个commits（可能包括自身）与预准备消息一致。

如果对一些non-faulty节点 commited-local为真，则commted即为真

每个节点在commited-local为真后，即可执行请求，执行完毕后，发送reply到Client

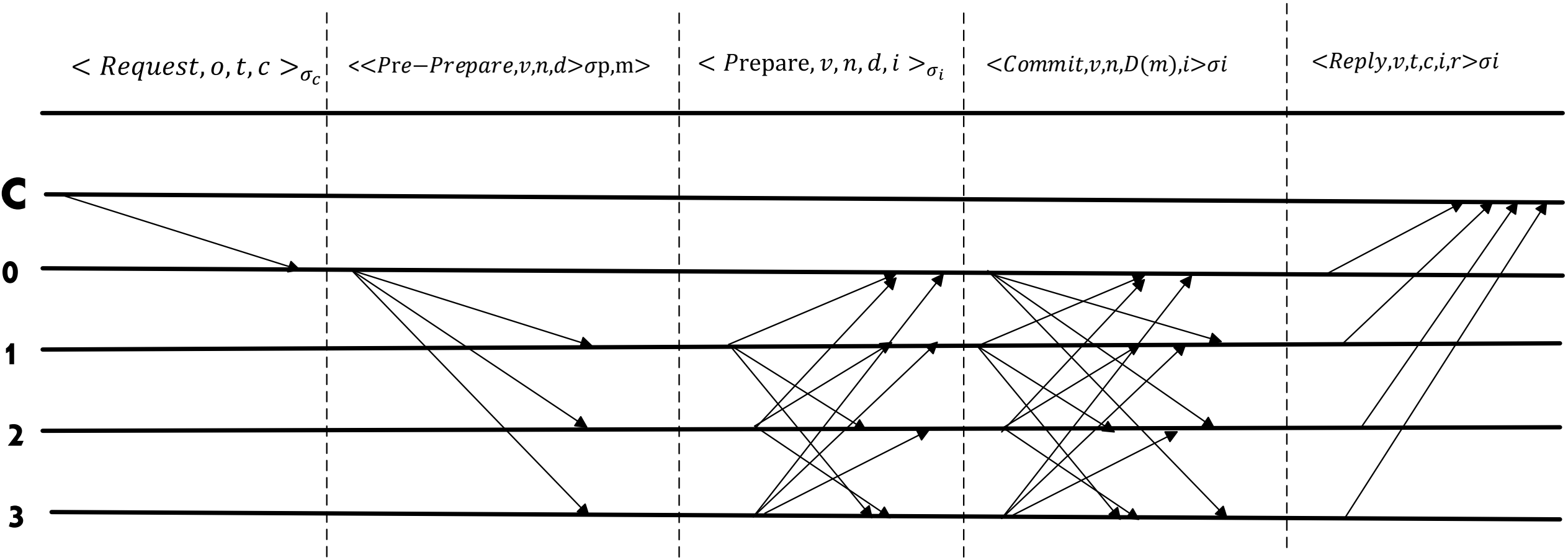
网络中有4个节点为例，有以下三种情况：

1. 都是non-faulty节点

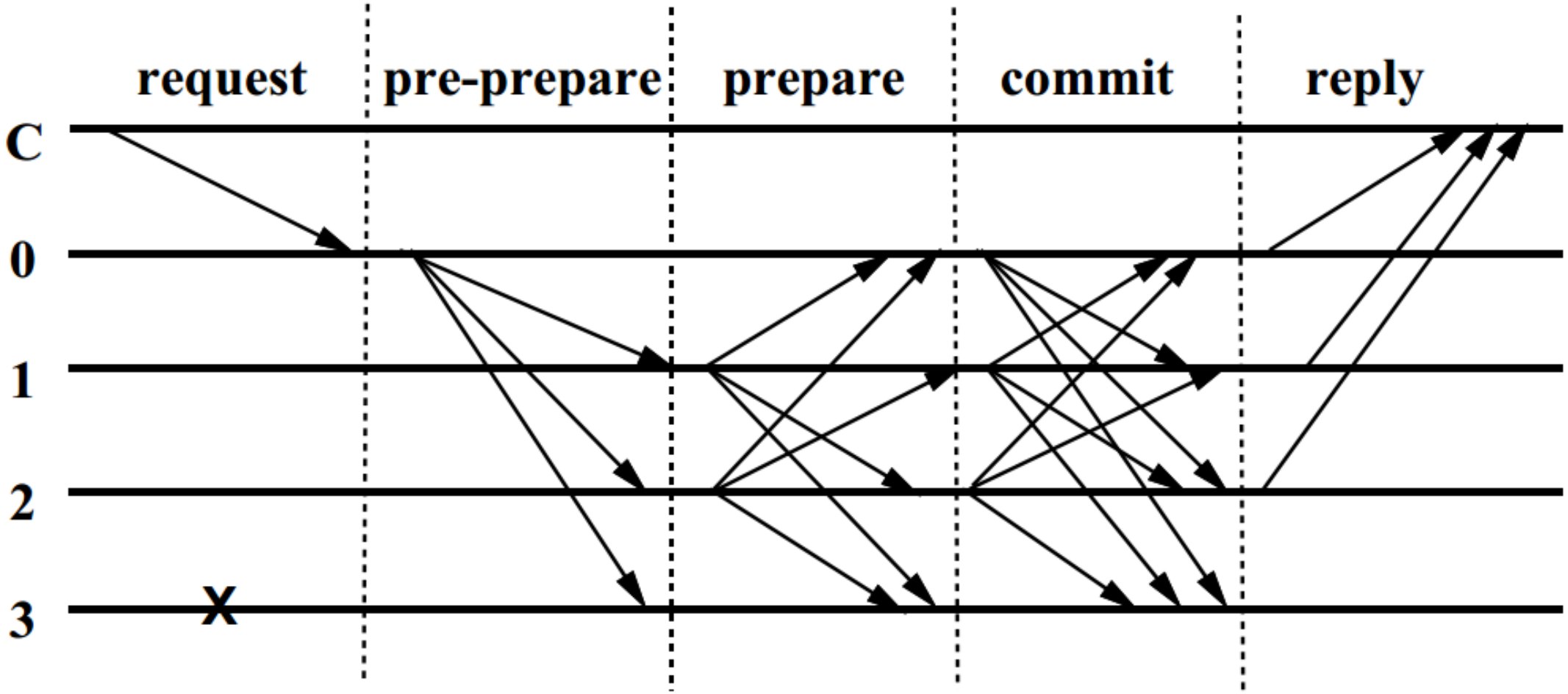
2. 主节点是non-faulty节点，备份节点里有一个faulty的

3. 主节点是faulty节点：eg 发给不同备份节点不同的请求

1 全部为好节点


$< Request, o, t, c >_{\sigma_c}$
<<Pre−Prepare,v,n,d>σp,m>
$< P\text{repare}, v, n, d, i >_{\sigma_i}$
<Commit,v,n,D(m),i>σi
<Reply,v,t,c,i,r>σi
C
0
1
2
3

2 主节点为好节点，备份节点中一个坏节点

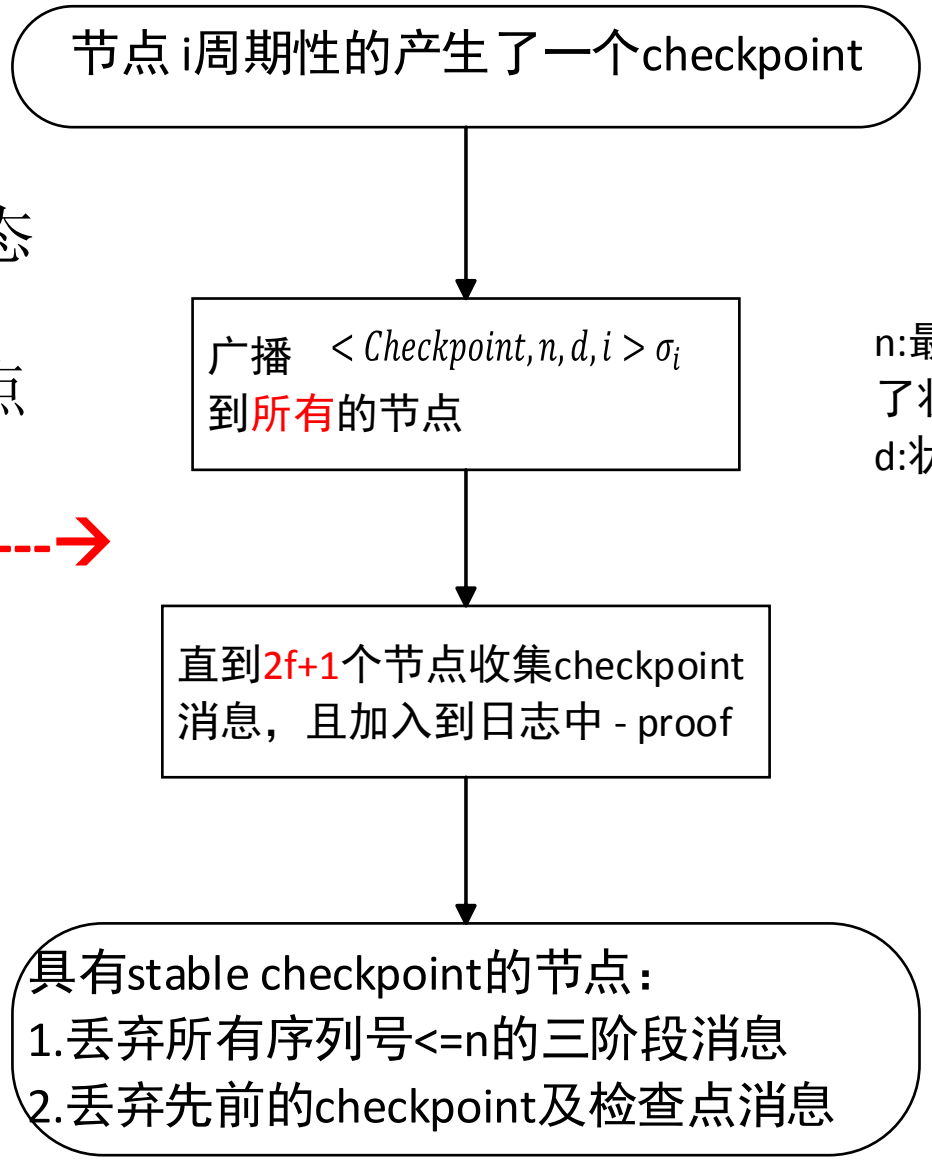


垃圾回收—清理本地日志消息的机制

Checkpoint ： 请求执行后得到的新状态

Stable checkpoint ： 具有proof的检查点

Proof的产生过程 --->



n:最近一次执行结果改变了状态的请求序列号
d:状态的摘要

Watermark

- Watermark

  低水位h：节点latest stable checkpoint里最大的消息序列号

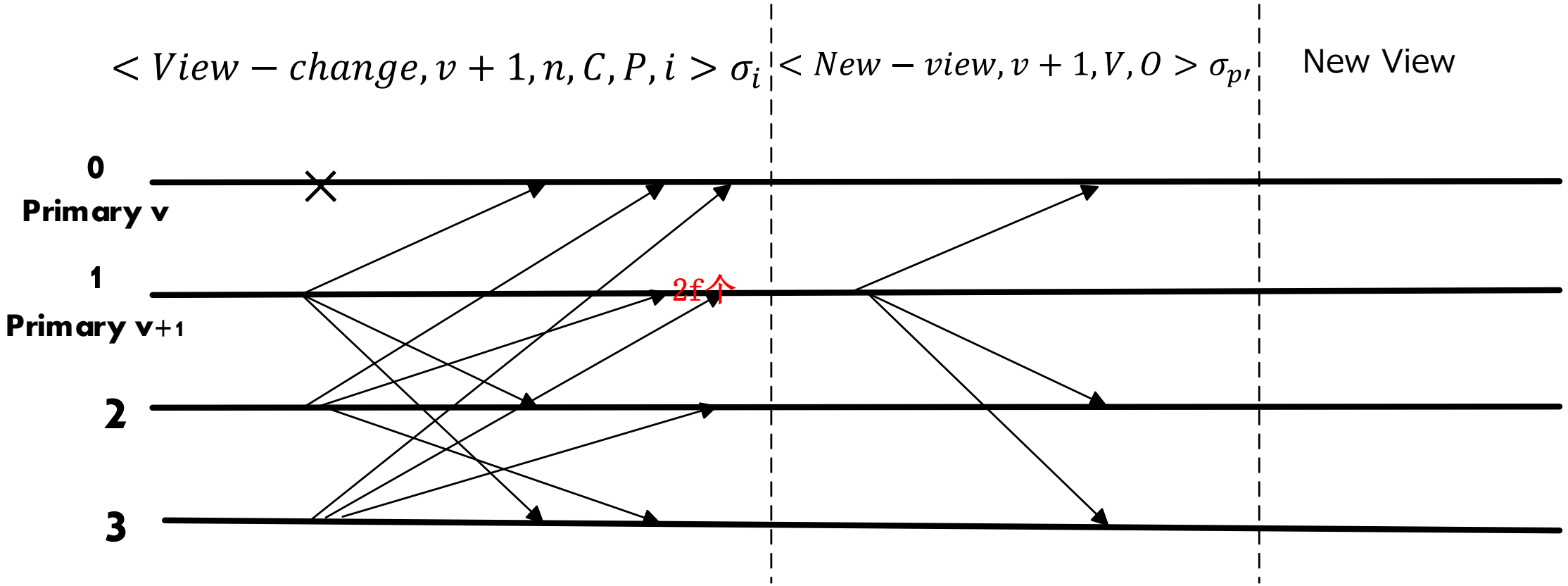
  高水位H=h+L，L是指定数值，一般为checkpoint周期K的常数倍

- Watermark的作用：**水位线高低值限定了可以被接受的消息。**。

- Checkpoint协议可以用来更新watermark的高低值

View Change —替换主节点

- View Change 被**超时**触发



$$< View - change, v + 1, n, C, P, i > \sigma_i$$

n： i节点stable checkpoint s 里最大的序列号

C： $2f + 1$个stable checkpoint s的proof

P： P是集合，集合中的每个元素是集合Pm，Pm对应在backup i节点处已经prepared的序列号大于n的请求。Pm包括一个有效的pre-prepare消息（不带有相应的客户端消息）和$2f$个来自不同backup签名的具有相同view，相同序列号，相同数字摘要的匹配的有效的prepare消息。

$$< New - view, v+1, V, O > \sigma_{p'}$$

V： v+1的主节点收到的 **$2f$+1**个view change消息

O: 一系列pre-prepare 消息

O的计算方法

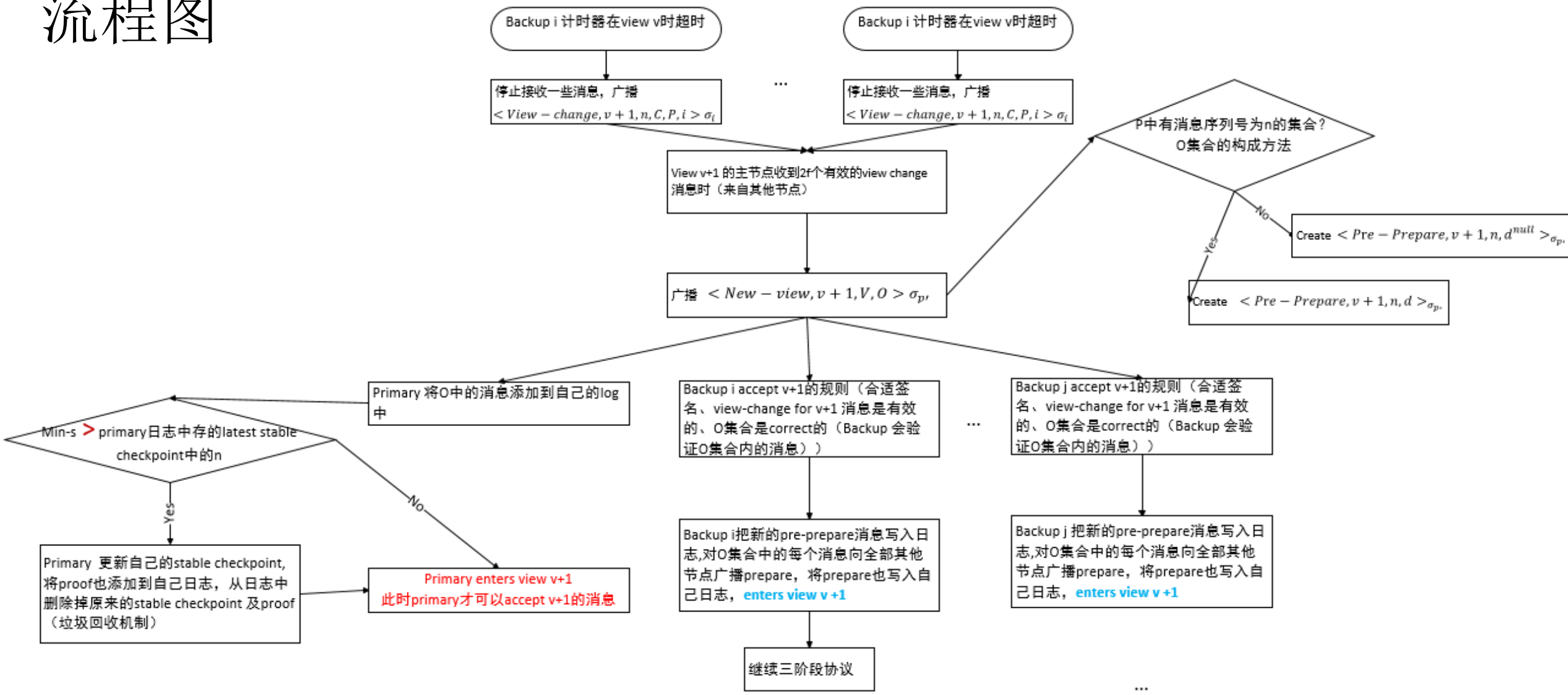
1.latest stable checkpoint 中最大的那个序列号n 是O中最小的序列号min，由所有的prepare消息得到最大的序列号max

2.primary节点在view v+1 中对于位于min和max之间的任何一个n产生一个新的pre-prepare消息。两种情况：

i. P中至少有一个集合Pm，序列号是n， 发出 $< Pre - Prepare, v+1, n, d >_{\sigma_{p'}}$

ii. 不存在上述集合 发出 $< Pre - Prepare, v+1, n, d^{null} >_{\sigma_{p'}}$

View-change 流程图



预准备和准备阶段：

- 保证了善意节点在同一个view中对请求的最终顺序达成一致（即使对请求进行排序的主节点失效了）

- 若prepared（m，v，n，i）为真，则prepared（m’，v，n，j）为假。

  有以下两种情况：

  1. primary节点是善意的
  2. primary节点是恶意的

准备和提交阶段

- 确保在**不同的视图**之间提交的请求是严格排序的。

- 发生view change时，由于view-change 和new-view携带了C和P消息，足够让$v+1$的主节点知道，哪些请求已经处于prepared状态，这些请求在新的view下仍依照之前的序列号重新开始三阶段协议。没有处于prepared状态的序列号被赋予了一个空请求，执行后不会引起状态转换

Safety & Liveness

- 安全性：所有non-faulty的节点都认可已经commit locally的序列号

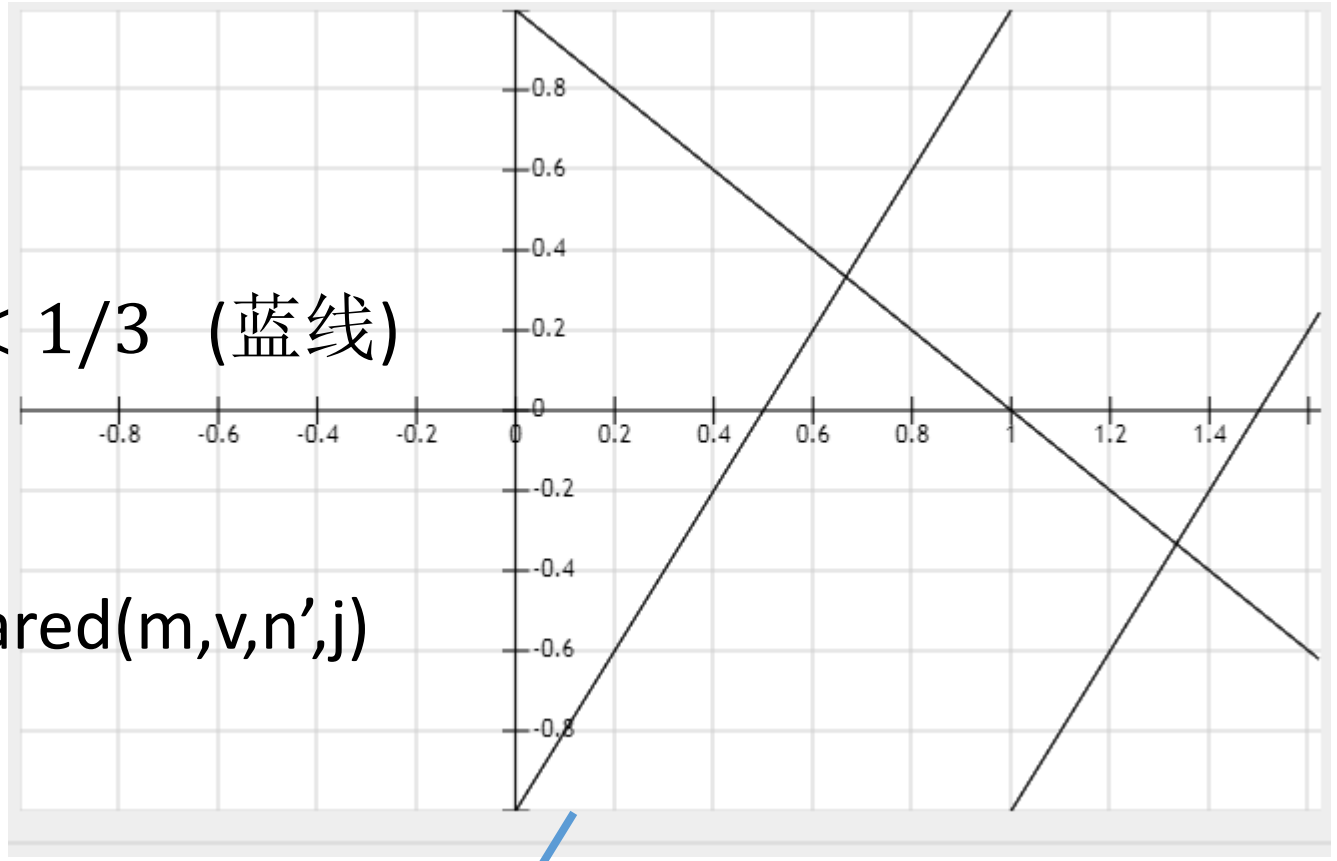
- 可用性：当节点无法请求时，必须进行view change ， 进入新的view

谢谢！

问题

- 是否需要第三个公式去证明$f < 1/3$ (蓝线)
$f + (1 - f)/2 + 1 >= q$

- 会出现Prepared(m,v,n,i) 和Prepared(m,v,n’,j)
都为真的情况？



对共识本质的理解

- 共识保证相同高度时，所有节点看到的消息是一致的，但是不保证消息一定是有效的。共识保证的是一致性，不代表经过共识的东西一定是正确的、有效的。