DAOWOO
达 沃 时 代

Paxos算法学习分享

2016.06.16

涂仲秋 田末

目录

Paxos算法的适用场景、解决的问题

- 分布式系统中节点间通信方式
  - 共享内存(Shared memory)
  - 消息传递(Messages passing)

- 适用场景
  - 基于消息传递模型的分布式系统中，各节点如何就某一个值达成一致

- 存在的异常
  - 进程响应慢
  - 进程可能会被杀掉
  - 进程会重启
  - 消息传递存在延时、丢失、重复（基础Paxos算法不考虑消息被篡改的情况）

- 解决的问题
  - 在一个可能发生上述异常的分布式系统中如何就某个值达成一致，保证无论发生以上任何异常，都不会破坏决议的一致性。

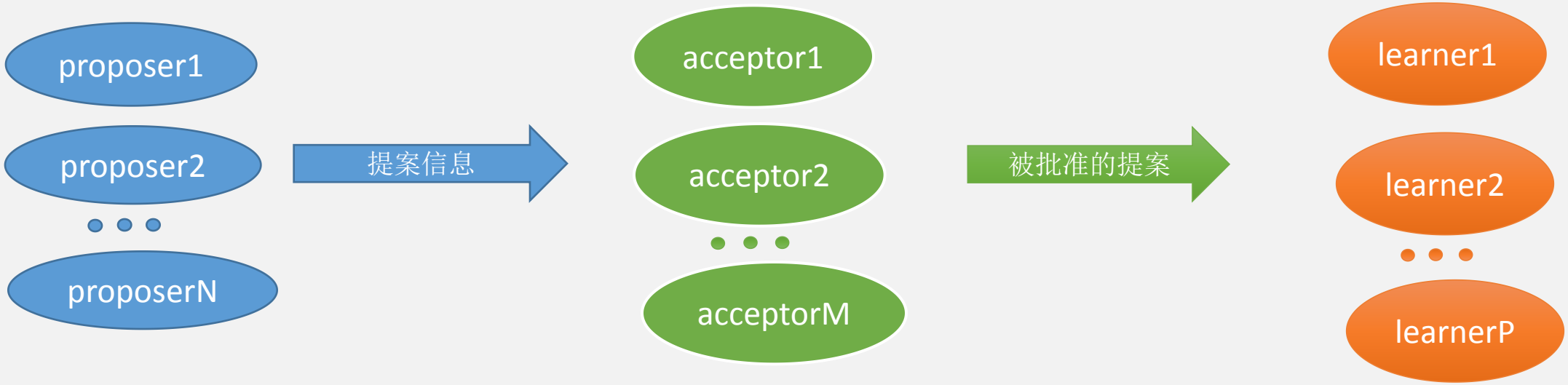
- 典型场景
  - 在一个分布式数据库系统中，如果各节点的初始状态一致，每个节点都执行相同的操作序列，那么它们最后能得到一个一致的状态。为保证每个节点执行相同的命令序列，需要在每一条指令上执行一个“一致性算法”以保证每个节点看到的指令一致。

Paxos算法的核心思想

- 解决思路
  - Paxos算法对各节点上收到的命令进行全局编号，如果能够编号成功，那么所有对节点的操作都按照编号顺序执行，一致性就可以得到保证。
- 如何编号
  - 通过表决，让所有的节点表决来决定哪个节点收到的哪条命令应该排第一、哪条命令排第二...，只要大多数节点都同意某条命令排第几，那就排第几。
- 算法核心
  - 确保每次表决只产生一条命令（一个Value）

Paxos算法推演过程

- 算法的基本角色构成
  - proposer:提案者，提出提案(proposal)，每个提案信息包括提案编号(number)和提案内容（提议的value）
  - acceptor:提案审批者，接收到提案后，可以接受或拒绝提案，如果某个提案获得大多数acceptor的接受，则该提案被通过(be chosen)
  - learner:学习者，只能学习被通过的提案



- 基本约束条件
  - 决议(value)只有在被proposer提出后才能被批准，未经批准的决议称为提案(proposal)；
  - 在一次Paxos算法的执行实例中（一轮选举中），只能批准(chosen)一个value；
  - learners只能获得被批准(chosen)的提案；

- 推演过程
  - P1：一个acceptor必须接受(accept)第一次收到的提案。

  - P2：一旦一个具有value v的提案被批准(chosen)，那么所有被批准的编号更大的提案必须具有value v。

  - P2a：一旦一个具有value v 的提案被批准(chosen)，那么任何acceptor所接受的所有编号更高的提案必须具有value v。

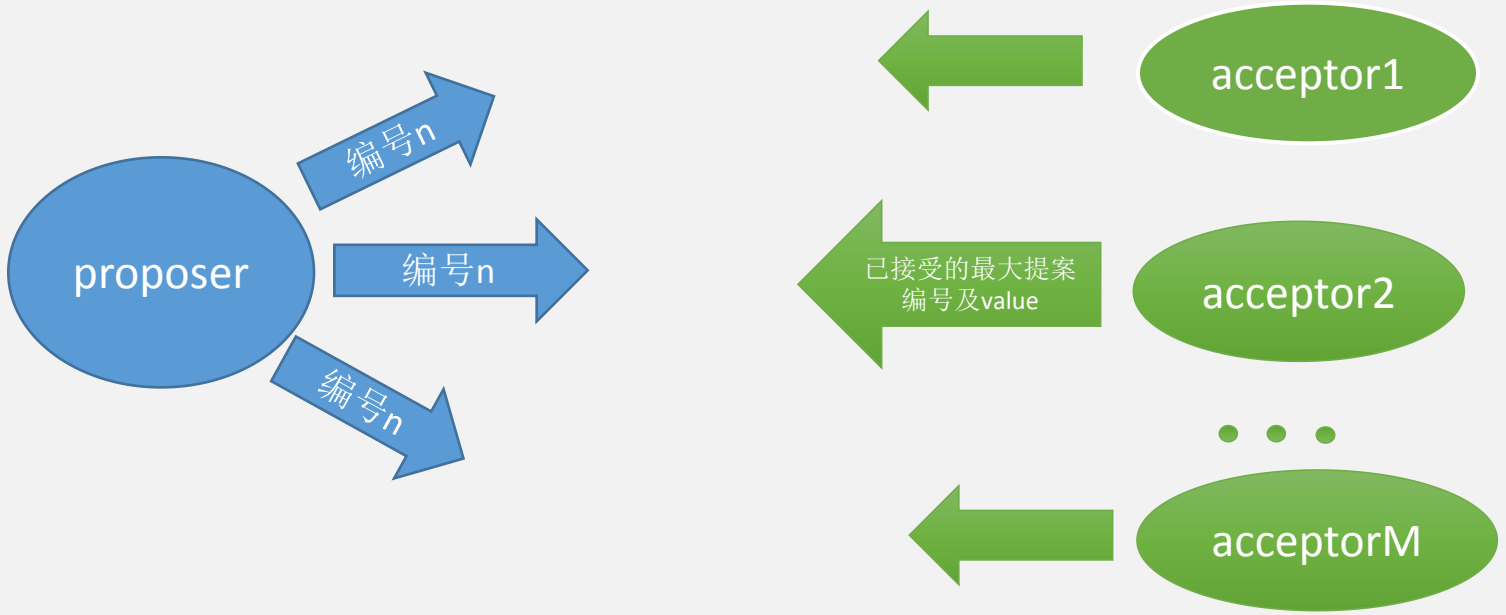
  - P2b：一旦一个具有value v 的提案被批准(chosen)，那么任何proposer提出的任何编号更高的提案必须具有value v。

- P2c：对任意的编号n和value v，如果一个编号为n内容为v的提案被大多数acceptor接受，那么存在一个由acceptor中的大多数所构成的一个集合S，满足如下两个条件之一：
  - (a) S中所有acceptor都没有接受(accept)过任何编号小于n的提案；
  - (b) S中所有acceptor已经接受了的所有编号小于n的提案中，v是编号最大的那个提案的内容(value)；

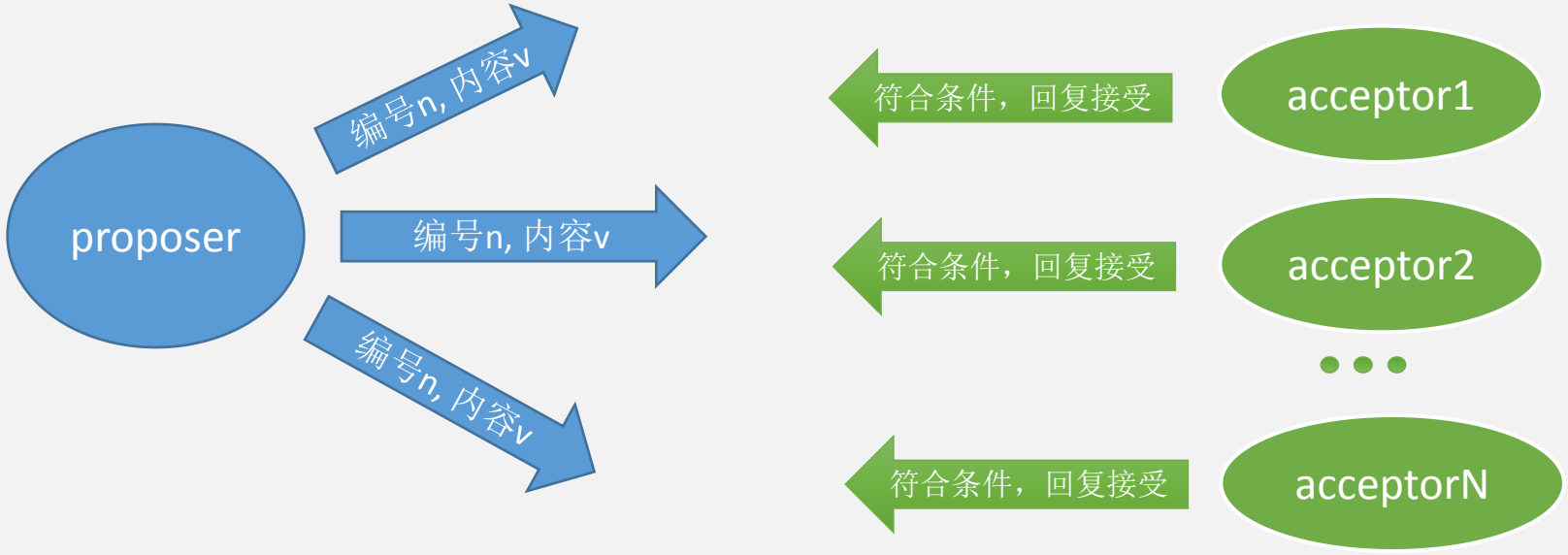
- P2c'：如果存在一个由acceptor中的大多数所构成的一个集合S，满足如下两个条件之一：
  - (a) S中所有acceptor都没有接受(accept)过任何编号小于n的提案；
  - (b) S中所有acceptor已经接受了的所有编号小于n的提案中，v是编号最大的那个提案的内容(value)；
- 那么，编号为n内容为v的提案能够被大多数acceptor接受。

Paxos算法形成

- 算法分为两个阶段：
  - Phase1：prepare
    - (a) proposer 选择一个编号为 n 的提案，将编号 n 发送给acceptor中的一个多数派；
    - (b) 如果acceptor发现 n 是它已经回复过的提案中编号最大的，它会回复它已经接受的最大编号提案的编号以及对应的提案内容(value)，同时会承诺proposer：不会接受编号小于 n 的提案。



- Phase2:accept
  - (a)如果proposer收到了多数派的回应，它发送一个accept消息（编号为 n，value为 v）到acceptor中的多数派；
  - (b)acceptor收到proposer发出的accept消息后检查，如果没有回复过编号比n大的提案，就接受该提案；否则拒绝或不回应。

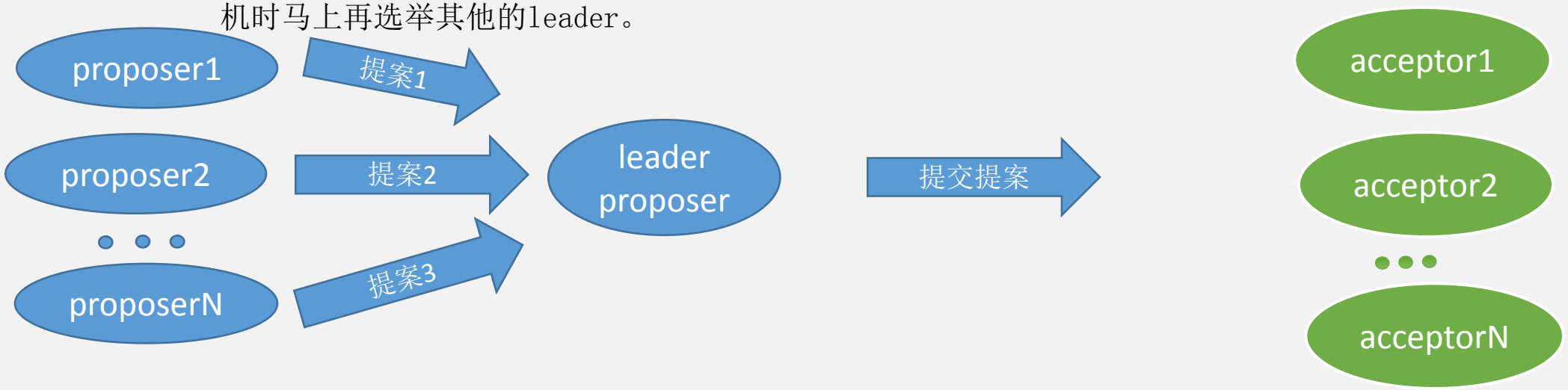


Paxos算法中存在的常见问题及解决方案

DAOWOO 达沃时代

- 唯一编号
  - 如何实现？
    - Google关于chubby的论文中提供了一种保证所有提案编号全序的方法：
      - 假设系统中有n个proposer，每个编号为i_r(0<= i_r <n)，提案编号的任何值 s 都应该大于它已知的最大值，并且满足：s % n = i_r => s = m*n + i_r
      - proposer已知的最大提案编号来自两部分：proposer自己对编号自增后的值和接收到acceptor对第一阶段prepare请求的拒绝后所得到的值。

- 活锁(live lock)
  - 产生背景
    - 当某个proposer在prepare阶段提交的提案编号被拒绝或accept阶段提交的提案被拒绝时，可能是因为acceptor已经接收到了更大编号的提案，因此proposer提高编号继续提交。
  - 定义
    - 当两个或多个proposer发现自己提案的编号过低转而提交更高编号的提案时，会导致算法陷入死循环，称为活锁。
  - 解决方案
    - 选举leader：选举出一个proposer作为leader，所有的提案都通过这个leader来提交，当leader宕机时马上再选举其他的leader。

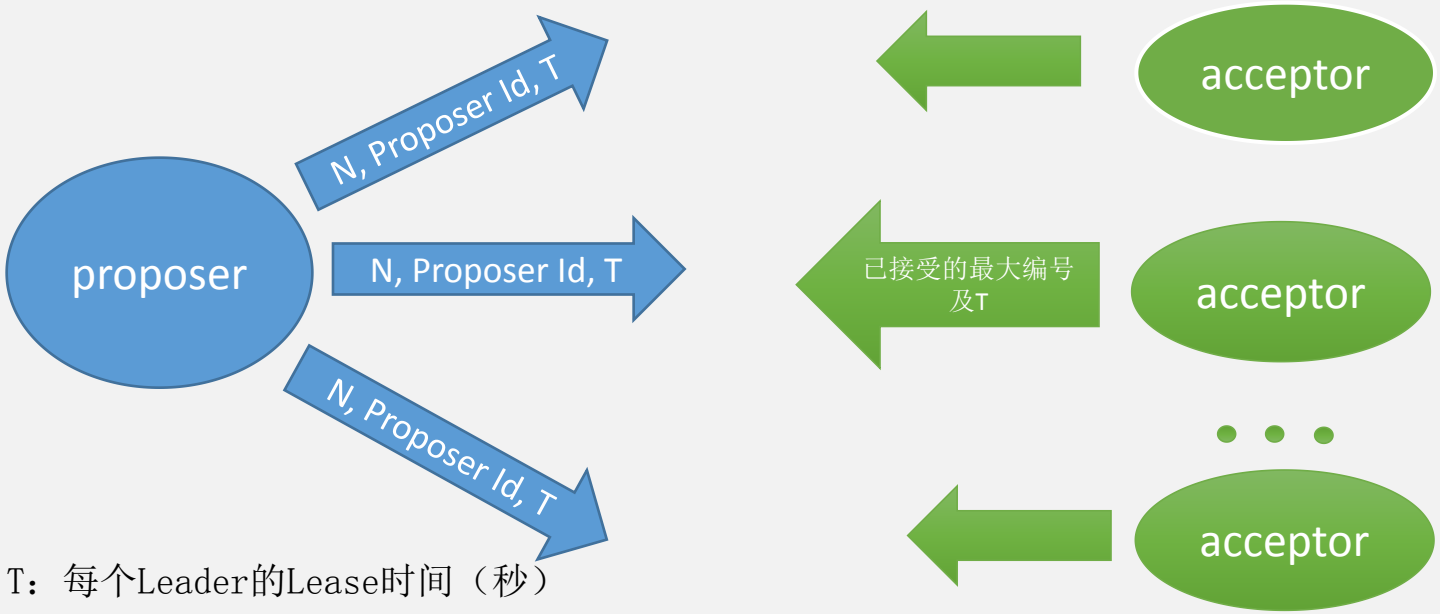


Leader选举(PaxosLease算法)

- Leader选举与Value选举不同之处：
  - 执行频率低（正常情况下，节点失败概率不会太高）
  - 无Paxos的持久化，重启后状态丢失
  - 保证算法纯正性，不依赖于其他算法
- PaxosLease算法
  - 它是Kespace开发的基于Paxos、Lease的Leader选举算法，算法的角色与Paxos一样，也分Proposer、Acceptor、Learn，各角色的行为也与paxos基本一致，但除下面两个重要的区别：
    - 不要求acceptor持久化数据
    - 不存在Leader

PaxosLease算法形成

- 算法分为两个阶段：
  - Phase1: prepare
    - (a) proposer启动定时器Timer1，等待T秒便超时；
    - (b) proposer向acceptor发送(ballot number N，proposer id，T)；
    - (c) acceptor接收到prepare消息后，判断：
      - 如果msg.ballot_number < local.promisedBallotNumber，则发送拒绝消息
      - 否则，发送accept，包含的内容已经promise的最大编号和T



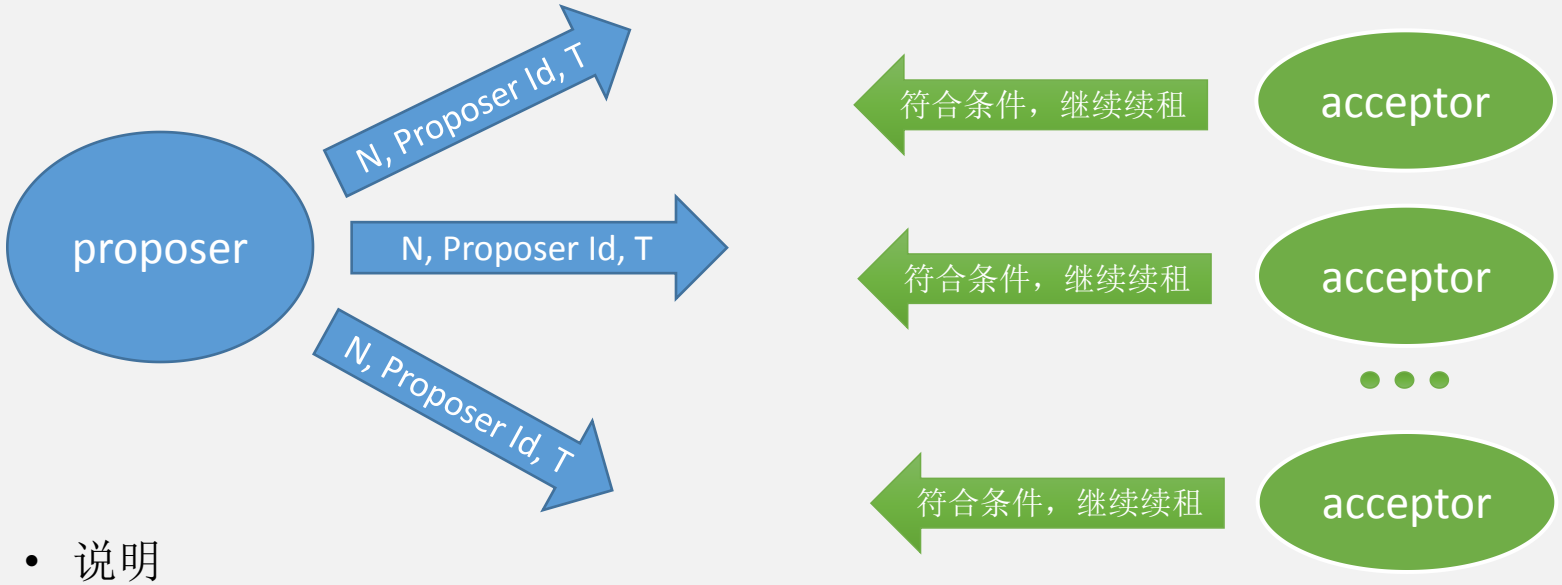
其中，

- T：每个Leader的Lease时间（秒）
- M：全局Lease时间，要确保 M > T
- ballot number：每次发送的proposal编号

PaxosLease算法形成

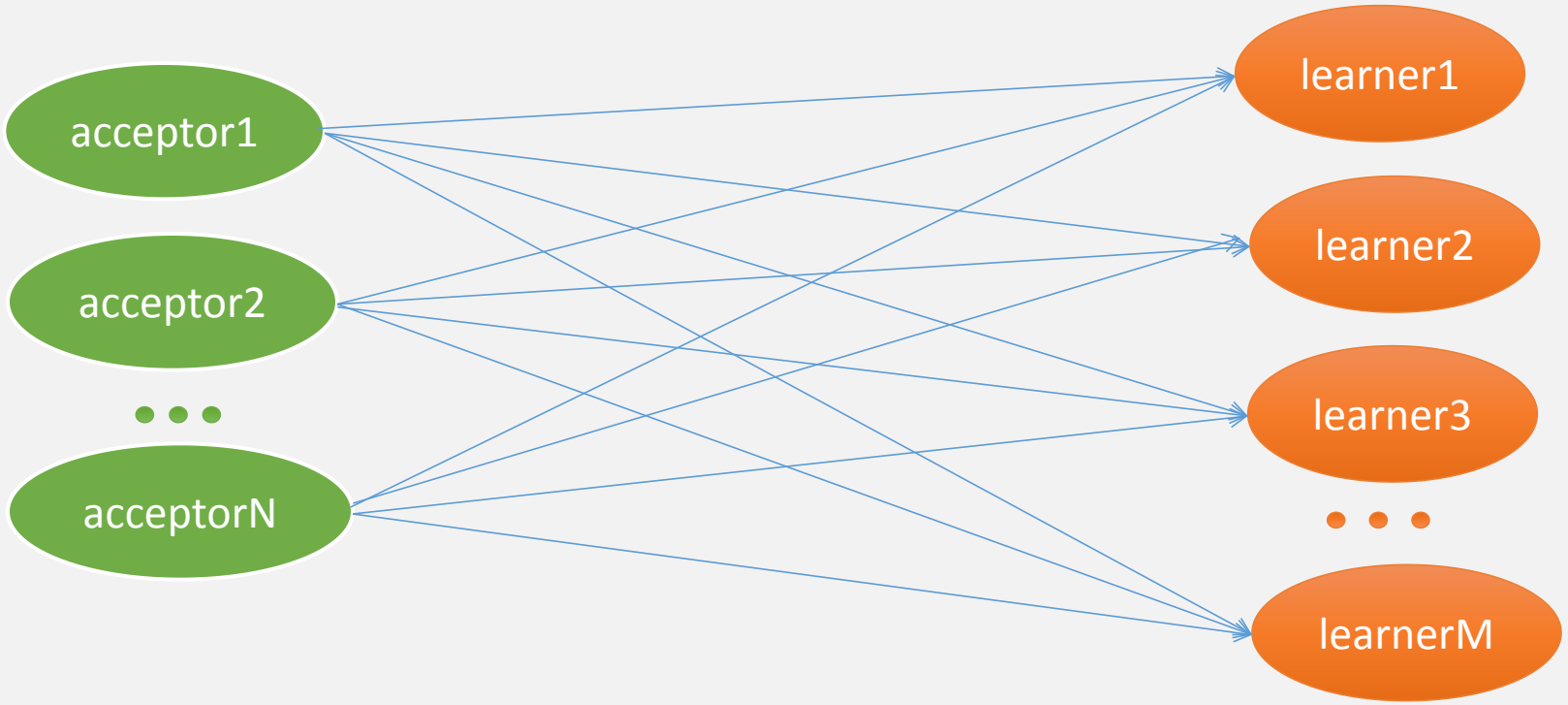
- Phase2:propose
  - (a)proposer接收acceptor的response
    - 如果多数派accept，则进入promise阶段
    - 否则，随机等待一段时间，提高编号重启prepare过程
  - (b)proposer执行promise阶段
    - 如果prepare阶段接收的value不为空，则终止promise
    - 否则，发送(ballot number N, proposer id ,T)
  - (c)acceptor接收到promise请求
    - 如果msg.ballot_number < local.promisedBallotNumber，则回应拒绝消息
    - 否则，启动定时器Timer2，等待T秒超时
  - (d)proposer接收acceptor的response
    - 如果接收到多数派的回应
      - 删除Timer1
      - 启动extend Timer3，等待时间T1 < T
      - 发送Learn消息，转入(e)
    - 否则，重启prepare过程
  - (e)Learn接收到proposer的learn消息
    - 停止Timer1
    - 重新启动Timer1

PaxosLease算法形成

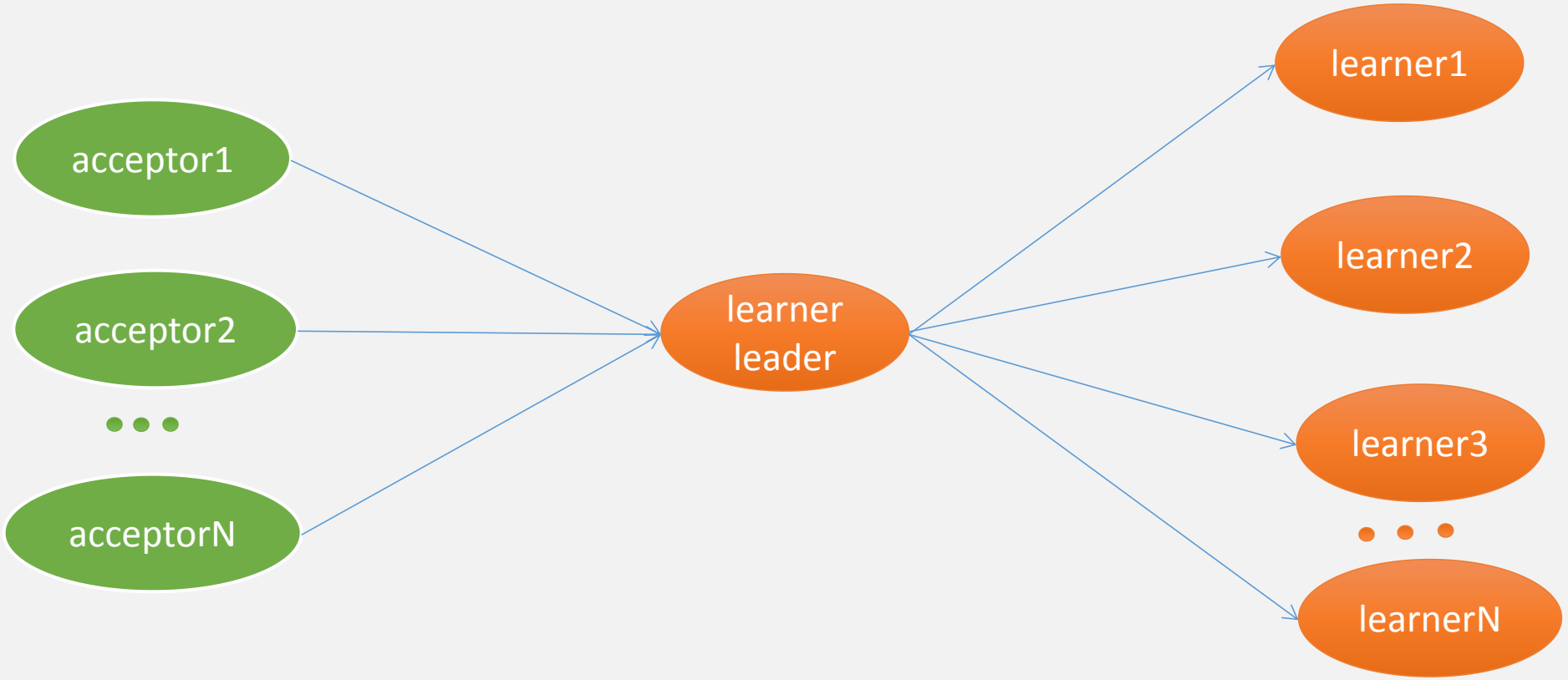


- 说明
  - Timer1(等待时间T，超时便发起prepare请求)运行于所有的Node，任何一个Node的Timer1超时便发起prepare请求
  - Timer2(等待时间T，超时便清空本次paxos instance状态，继续下一次)仅运行与参与选举过程的Node，如果超期则清空本次选举状态
  - Timer3(等待时间T1 < T，超时便执行prepare)仅运行于获得lease的节点，目的是在lease超期之前续租
- 小结
  - 其实PaxosLease的正确性是有Paxos算法保证的，PaxosLease只是在Paxos的基础上限定了一个时间。在时间T之内，任何节点都不能申请lease，因此Master宕机后重新选择Master的最大时间为T，也即服务不可用的最大时间为T。
  - T设的过小会减少服务不可用的时间，但会产生更多的内部消息；设的过大内部消息减少，但会导致更长的宕机时间。

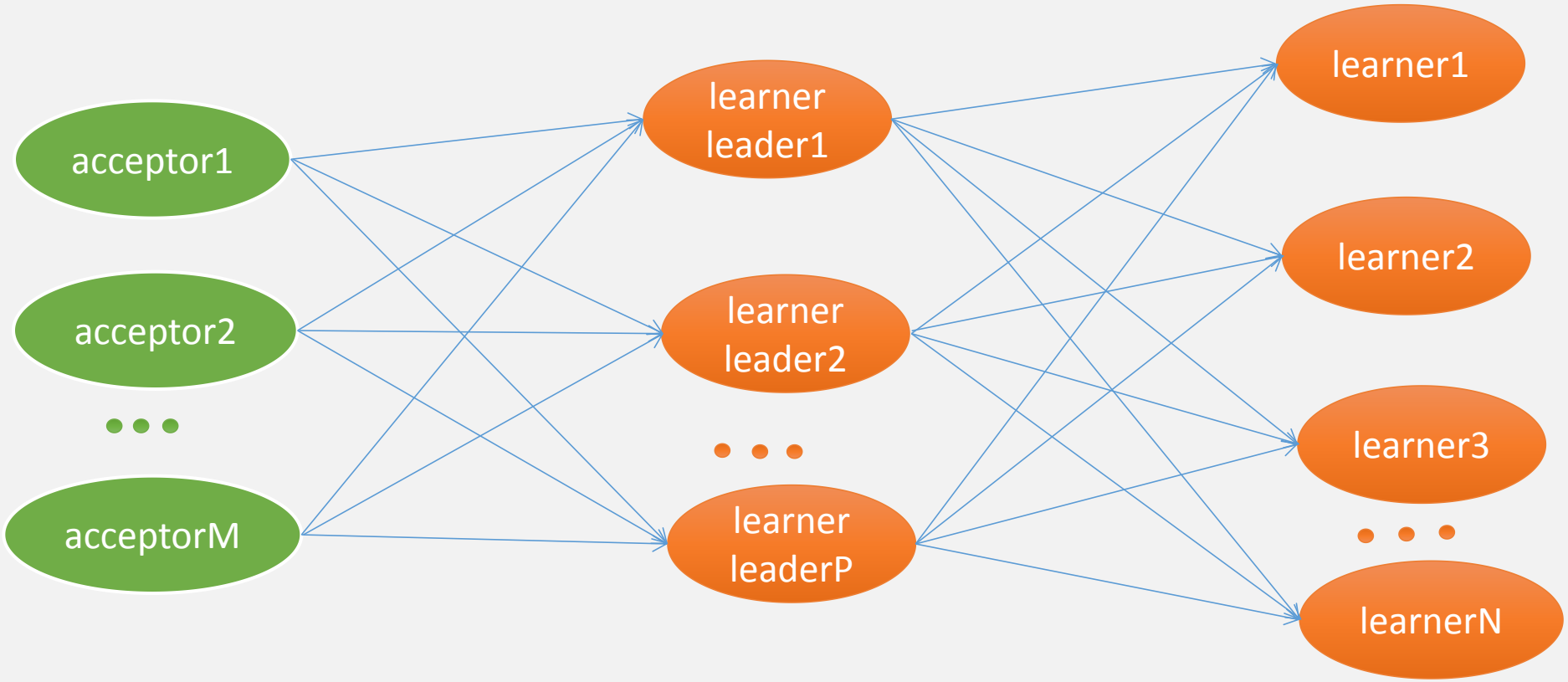
- 学习决议
  - 在一轮选举中(instance)，当一个决议被最终选出后，最重要的事情就是让learner学习该决议。
  - learner如何知道决议已经被选出了，即某个提案已经被acceptor中的大多数接受(accept)了？
    - 三种实现方式：
    - 1）没有 learner leader，acceptor在批准某项提案之后就把该提案的信息发送给learner；



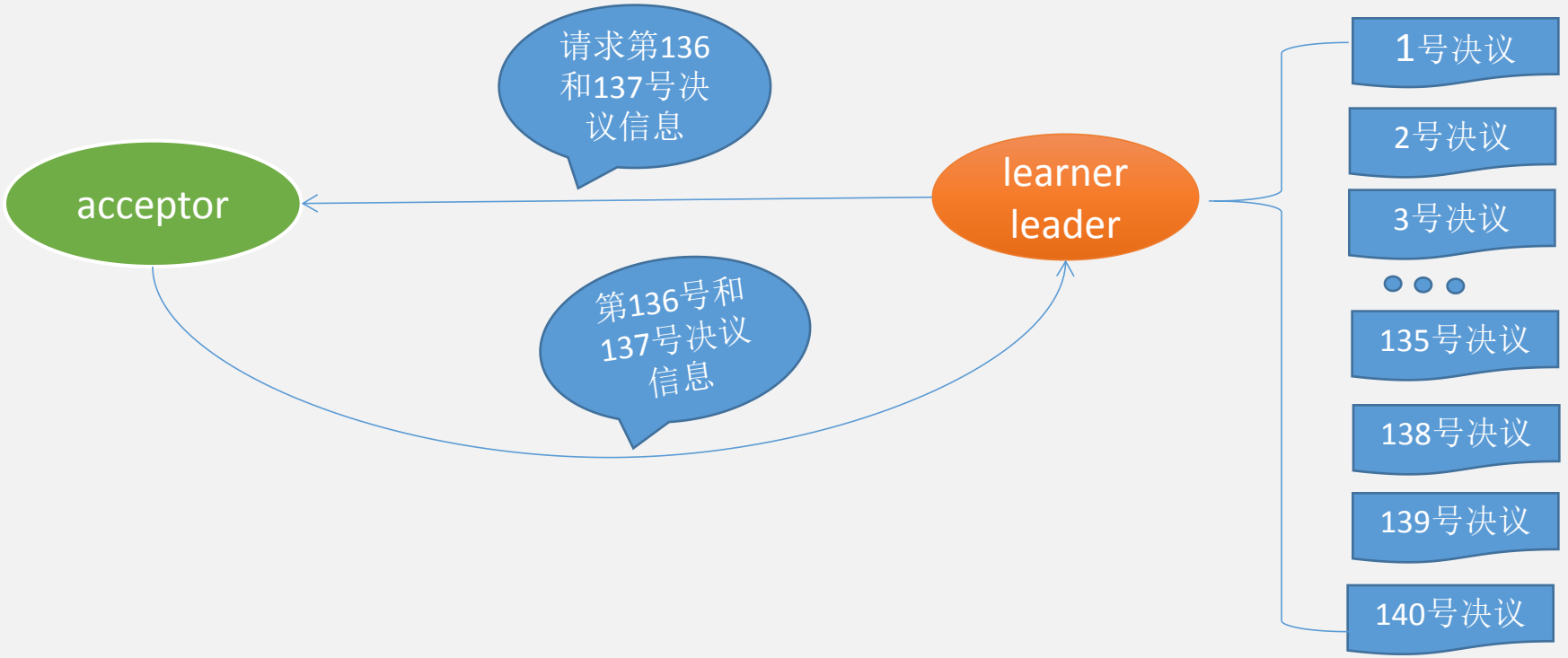
- 2)选举出一个learner leader，当acceptor批准某项提案之后，就把该提案的信息发给这个leader，然后再由这个leader把提案通过的信息发给系统中的其他learner，但是这样会形成单点；



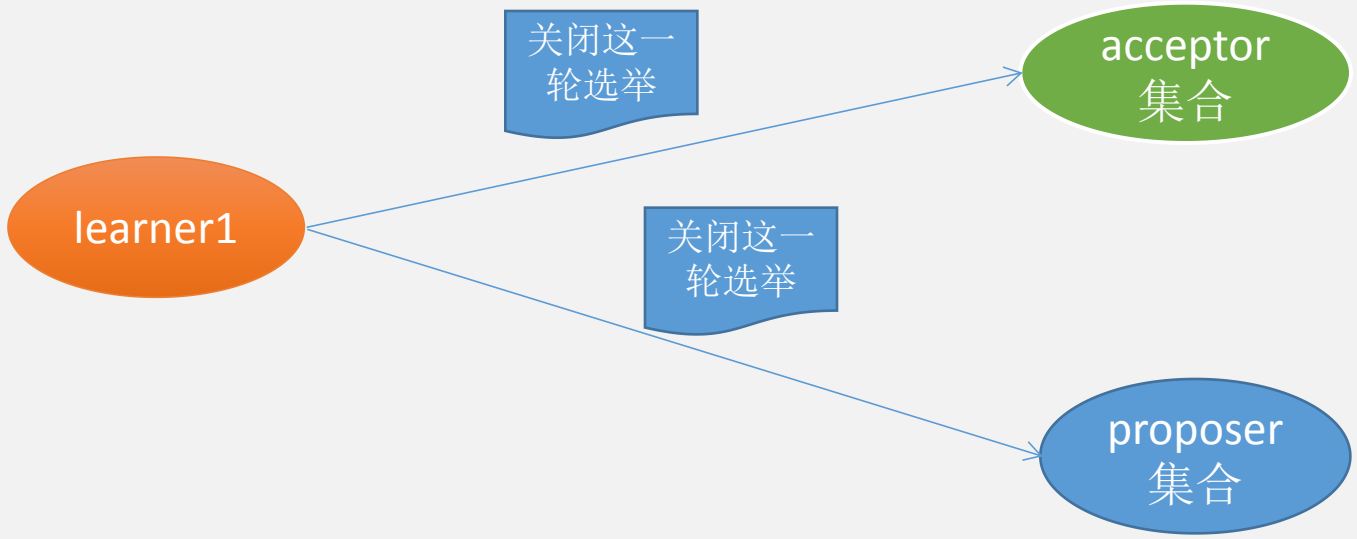
- 3)选举出多个leader learner，acceptor批准某项提案之后，就把该提案信息发给这个leader learner集合，然后由这个leader learner集合把批准的提案信息发送给其他learner；



- 学习决议的顺序
  - learners必须严格按照决议被通过(be chosen)的顺序来学习决议，即先学习第一次选举选出的1号决议，再学习第二次选举选出的2号决议。假设learner leader已经知道了1~135轮选举选出的决议，也即要执行的命令序列中第1~135条命令，以及138~140号决议，这个时候只能执行第1~135条命令，不能执行138~140条命令。



- 关闭一轮选举(instance)
  - 收到acceptor批准某项提案的learner还需要周期性的检测批准某项提案的acceptor的个数，如果达到大多数，则该提案正式通过，称为决议。此时，需要learner来关闭这一轮选举(instance)，并通知proposer和acceptor。



- Learner小结
  - paxos中的learn相对比较抽象，好理解但难以想象能做什么，原因在于对paxos的应用场景不清晰。一般说来有两种使用paxos的场景：
    - paxos作为单独的服务，比如google的chubby，hadoop的zookeeper
    - paxos作为应用的一部分，比如Keyspace、BerkeleyDB
  - 如果把paxos作为单独的服务，那learn的作用就是达成决议后通知客户端；如果是应用的一部分，learn则会直接执行业务逻辑，比如开始数据复制。

  - 持久化
    - learn所依赖的所有信息就是value和instance，这些信息都已在acceptor中进行了持久化，所以learn不需要再对消息进行持久化，当learn新加入或重启时要做的就是能通过acceptor把这些信息取回来。

  - 错误处理
    - learn可能会重启或新加入后会对“之前发生的事情”不清楚，解决办法是：使learn继续监听消息，直至某个instance对应的value达成一致时，learn再向acceptor请求之前所有的instance。

Paxos算法小结

DAOWOO 达沃时代

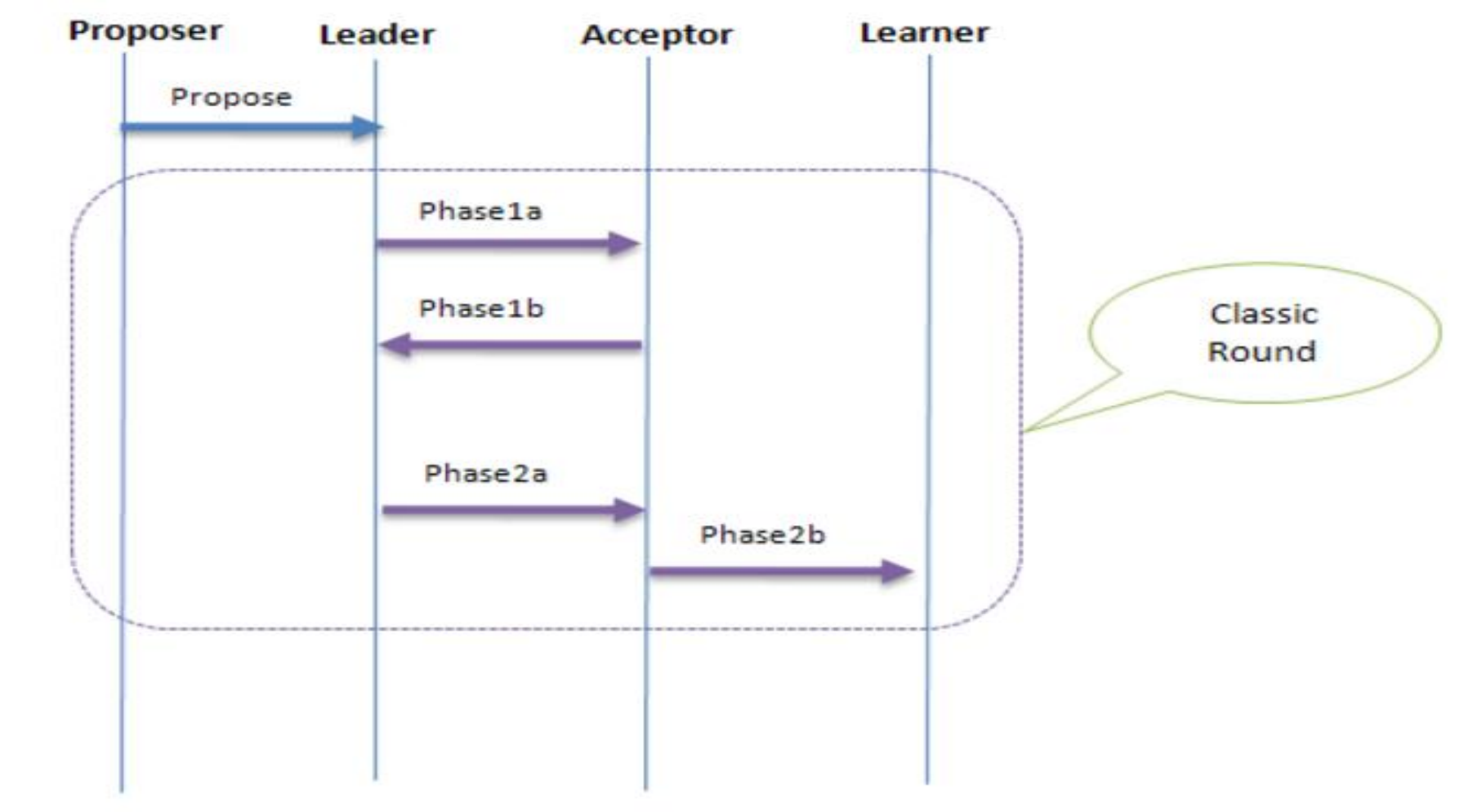
- 持久化存储
  - propose存储已提交的最大proposal编号、决议编号（instance id）
  - acceptor存储已promise的最大编号；已accept的最大编号和value、决议编号
  - learn存储已学习过的决议和编号
- 异常情况
  - 在算法执行的过程中会产生很多的异常情况，比如proposer宕机、acceptor在接收proposal后宕机，proposer接收消息后宕机，acceptor在accept后宕机，learn宕机等，甚至还有存储失败等诸多错误。
  - 无论何种错误必须保证paxos算法的正确性，这就需要proposer、aceptor、learn都做能持久存储，以做到server”醒来“后仍能正确参与paxos处理。
- 其他
  - 在实际工程实现时要考虑磁盘损坏、文件损坏、Leader身份丢失等诸多的错误。
  - 总之，我们通过讨论了paxos个角色的职责，以及它们是如何通过paxos算法达成一致，另外我们还讨论了paxos算法中涉及到系列问题以及其解决方案，和工程实现中paxos算法可能存在的诸多的问题。通过上述的分析和讨论，相信我们对Paxos算法有了进一步的了解和熟悉。

Fast Paxos算法介绍

- 背景介绍
  - Lamport 于2005年发表的论文Fast Paxos描述了这一算法；
  - 对Paxos算法的改进，通过减少消息传递的步骤加快算法的执行。
- 算法角色
  - Client/Proposer/Learner: 负责提案并执行提案
  - Coordinator：Proposer协调者，可有多个，Client通过Coordinator进行提案
  - Leader：在众多的Coordinator中指定一个作为Leader
  - Acceptor：负责对Proposal进行投票表决

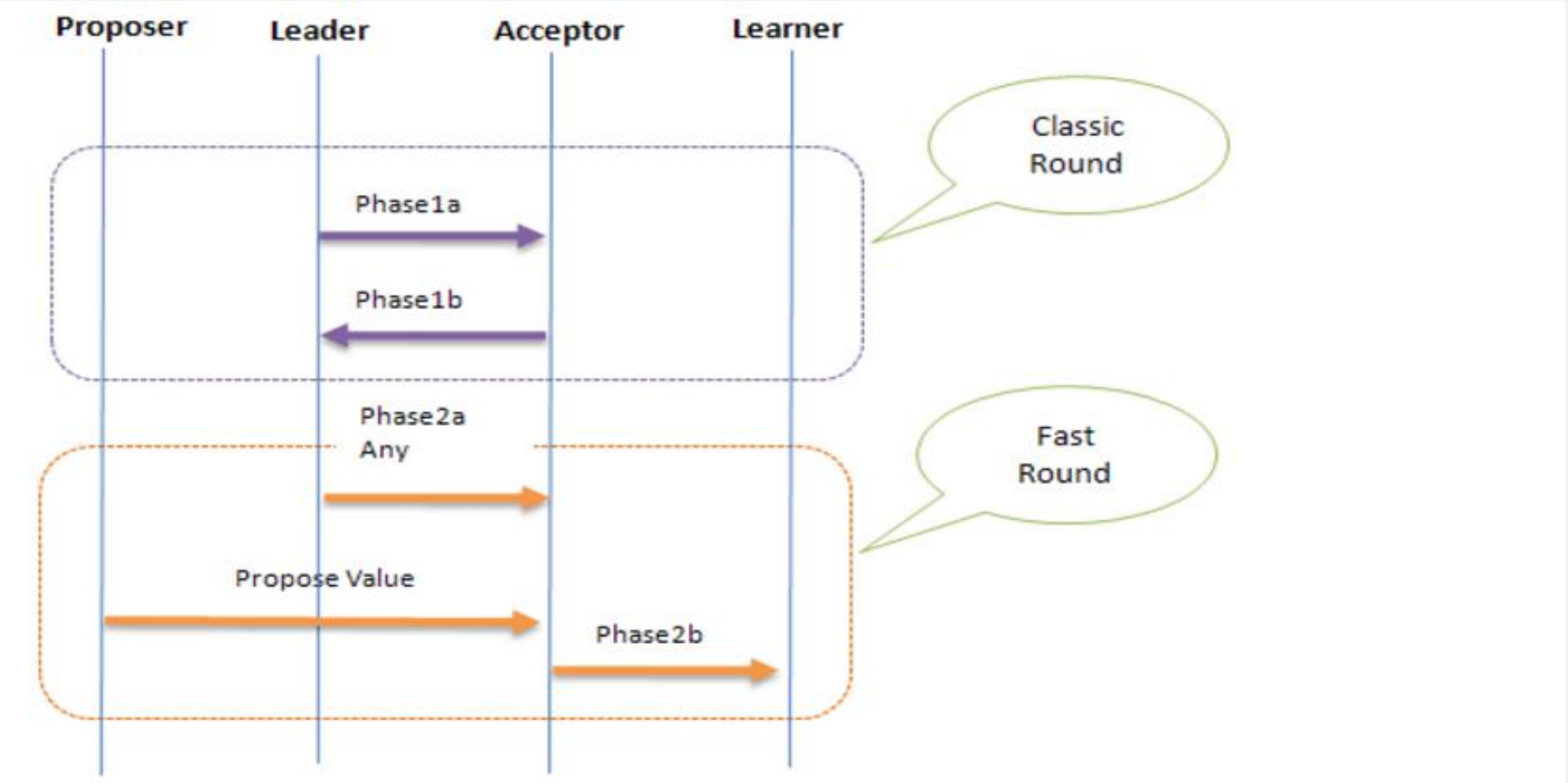
- classic Paxos的几个阶段：
  - phase1a：leader提交提案编号到acceptor中的大多数
  - phase1b：收到请求的acceptor回应已经接收到提案的最大编号和该编号对应的内容（如果存在）
  - phase2a：leader收集acceptor的返回值
  - phase2a.1：如果acceptor返回的信息中没有提案内容，则leader自由决定一个提案内容（提案内容不受限）
  - phase2a.2：如果acceptor返回的信息中有提案内容，则leader选择收到的编号最大的提案内容作为本次提案内容发送给acceptor中的大多数
  - phase2b：acceptor把表决结果发送到learner

- classic Paxos一轮选举的时序图：



- Fast Paxos的几个阶段：
  - phase1a：leader提交提案编号到acceptor中的大多数
  - phase1b：收到请求的acceptor回应已经接收到提案的最大编号和该编号对应的内容（如果存在）
  - phase2a：leader收集acceptor的返回值
  - phase2a.1：如果acceptor返回的信息中没有提案内容，则发送一个any消息给acceptor，然后acceptor便等待proposer提交提案
  - phase2a.2：如果acceptor返回的信息中有提案内容，则根据规则选取一个
  - phase2b：acceptor把表决结果发送到learner以及leader

- Fast Paxos一轮选举的时序图：



Thanks !

联系我们

北京总部
北京市朝阳区安定路一号国家奥林匹克体育中心体育场东南门2层2160室
邮编：100029
电话：+8610 8843 7583
传真：+8610 8437 6490

武汉分公司
武汉市洪山区珞瑜路33号中部创意大厦1907、1908
邮编：430074
电话：+8627 8786 2881
传真：+8627 8786 2567