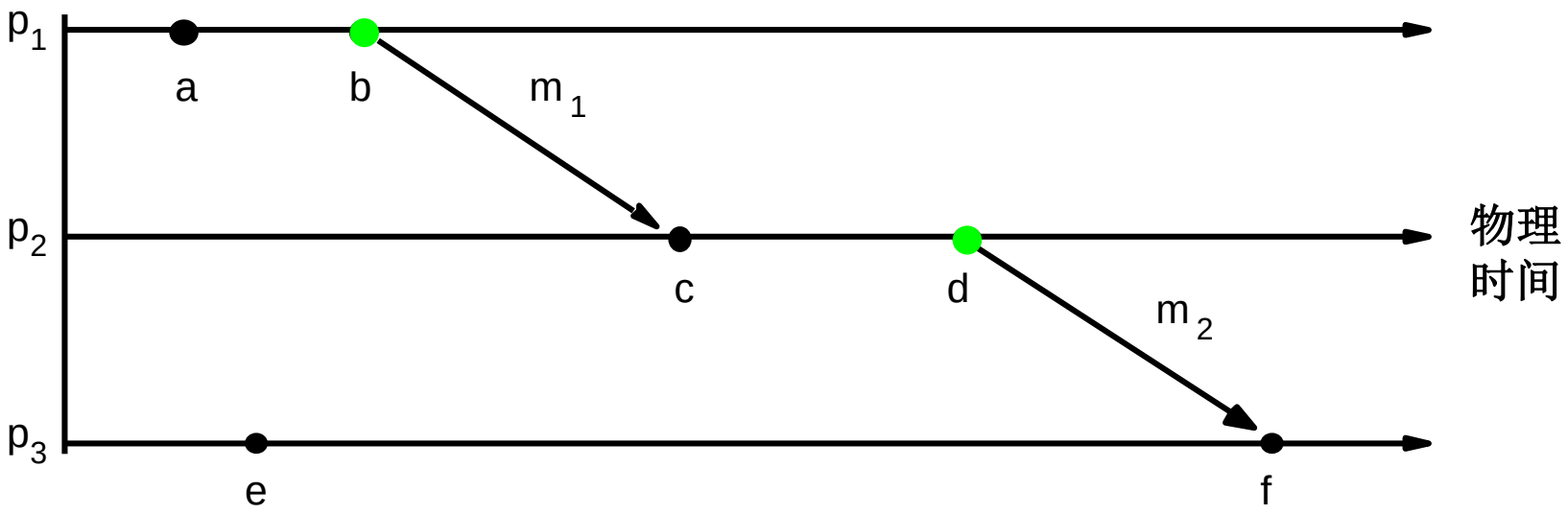
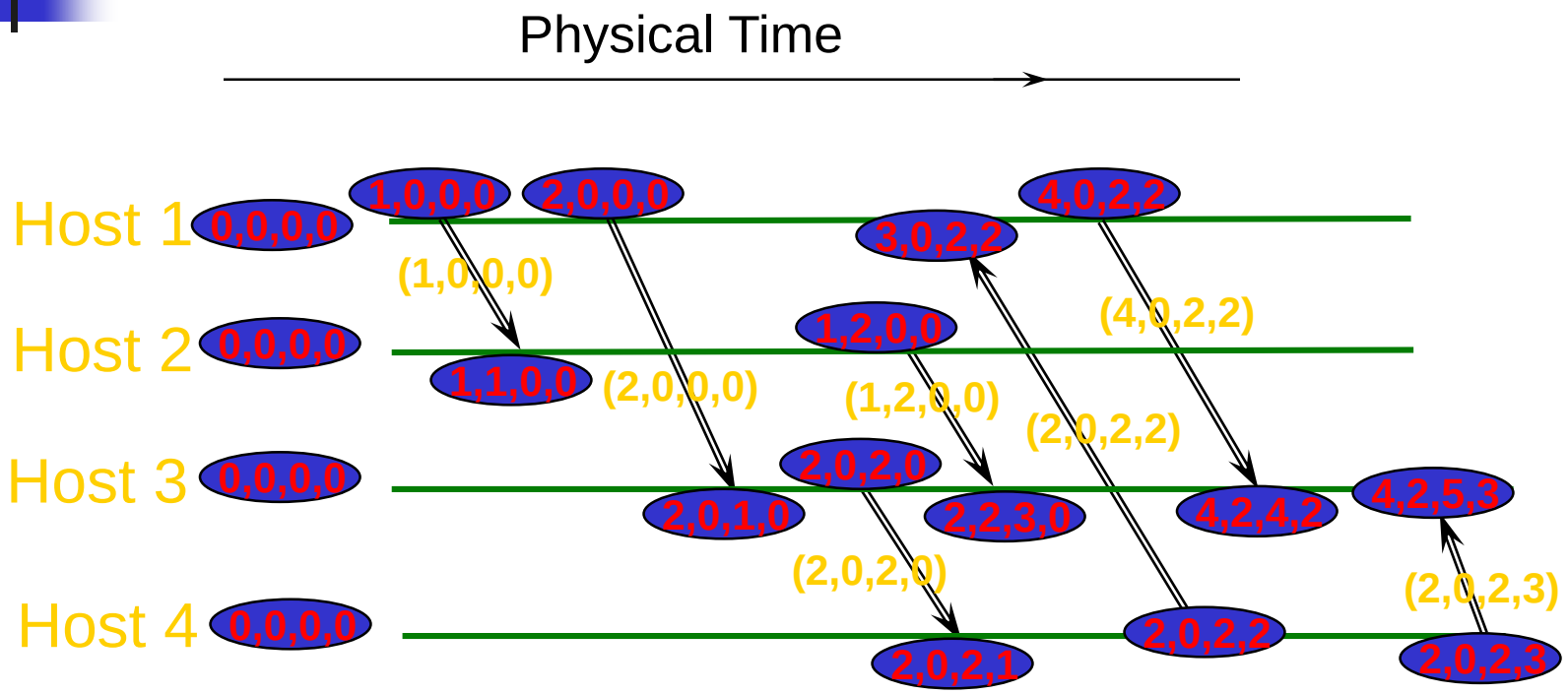
Review:逻辑时间和逻辑时钟

- 事件排序示例



- b→c，c→d和d →f成立
- b→f与e→f均成立
- 事件b和e无法比较，即b||e

Review:向量肘钟



n,m,p,q Vector logical clock

(vector timestamp) Message

Review:全局状态

- 进程状态

  s_i^k：进程p_i在第k个事件发生之前的状态

- 全局状态——单个进程状态的集合

  $S = (s_1, s_2, \ldots s_N)$

- 割集——系统全局历史的子集

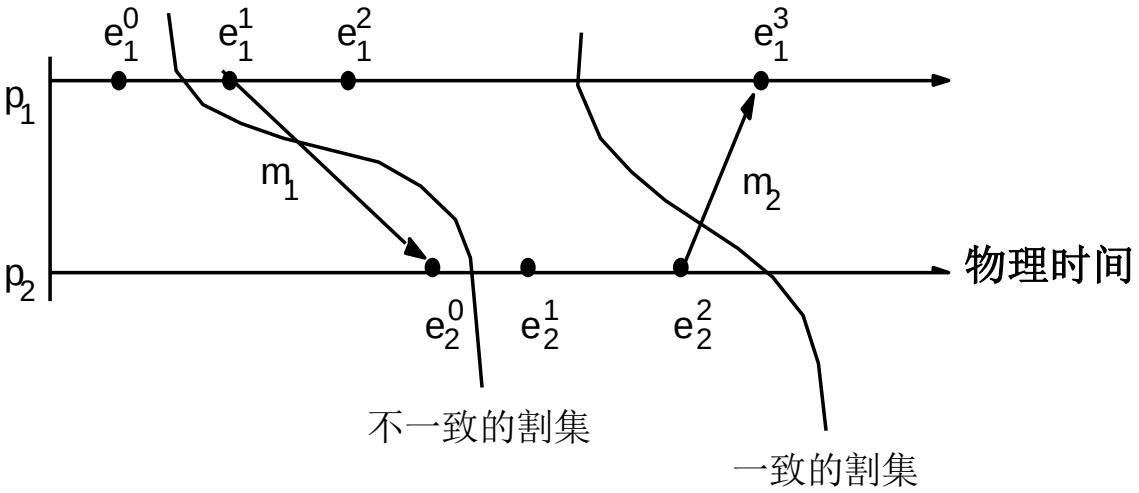
  $C = \langle h_1^{c1}, h_2^{c2} \ldots h_3^{c3} \rangle$

- 割集的一致性

割集C是一致的：对于所有事件$e \in C$，$f \rightarrow e \quad \Rightarrow f \in C$

Review:全局状态

- 割集示例



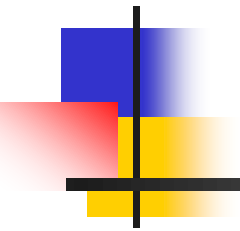
Review:分布式调试

- 方法
  - 监控器进程

    收集进程状态信息
  - 全局状态谓词ϕ的判断

    -可能的ϕ：存在一个一致的全局状态S，H的一个线性化走向经历了这个全局状态S，而且该S使得ϕ(s)为True。

    -明确的ϕ：对于H的所有线性化走向L，存在L经历的一个一致的全局状态S，而且该S使得ϕ(s) 为True。



分布式系统

协调和协定

第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结

简介

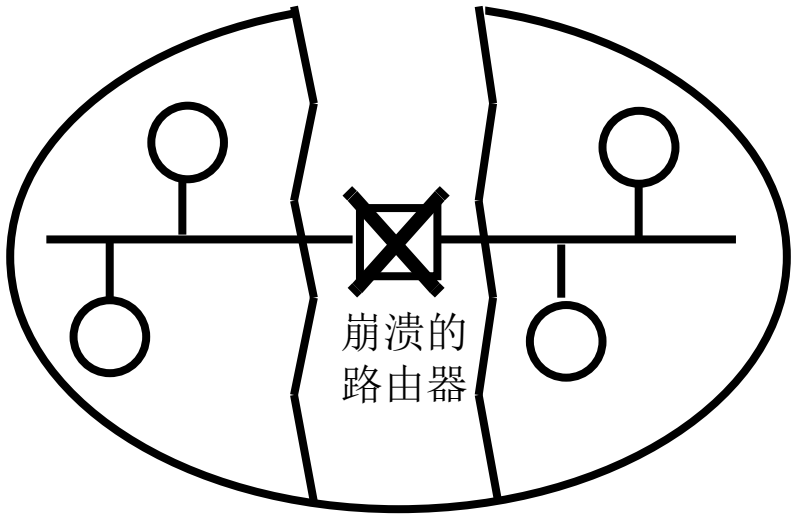
- 构造分布式系统的主要动力：*资源共享和协作*
- 分布式系统中的进程需要*协调动作*和对共享资源*达成协定*
- 分布式中的协作
  - 互斥
  - 选举
  - 组播

    可靠性和排序语义
  - 进程间的协定

    共识和拜占庭协定

简介

- 故障模型
  - 先考虑 无故障模型
  - 再考虑 良性故障
  - 然后考虑 随机故障
- Internet现状
  - 网络分区
  - 非对称路由
  - 连接的非传递性
    - p—q，q—r，但p-\-r



简介

- **通道假设**

  进程通过可靠的通道连接

- **进程假设**

  进程仅在崩溃时出现故障

- **故障检测器**
  - 不可靠的故障检测器

    产生值：Unsuspected和*Suspected*
  - 可靠的故障检测器

    产生值：Unsuspected和*Failed*

简介

- 不可靠故障检测器的实现示例
  - 每个进程以周期T通告自己正常消息
  - 检测器以时间D作为最大消息传输延迟
  - 若检测器在时间T+D内没有收到进程的通告消息，则设定此进程的状态为“Suspected”
  - 时间参数T、D可动态设置

第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结

分布式互斥

- 目的

  仅基于消息传递，实现对资源的互斥访问

- 假设
  - 异步系统
  - 无故障进程
  - 可靠的消息传递
- 执行临界区的应用层协议
  - enter( )　　//进入临界区——若必要，可以阻塞进入
  - resourceAccesses( )　　//在临界区访问共享资源
  - exit( )　　//离开临界区——其它进程现在可以进入

分布式互斥

基本要求

- 安全性

  在临界区内一次最多有一个进程可以执行

- 活性

  进入和离开临界区的请求最终成功执行

- →顺序

  如果一个进入临界区的请求发生在先，则进去临界区时仍按此顺序。

分布式互斥

算法的性能评价

- 带宽消耗

  在每个 enter 和 exit 操作中发送的消息数

- 客户延迟

  由enter和exit操作引起的延迟

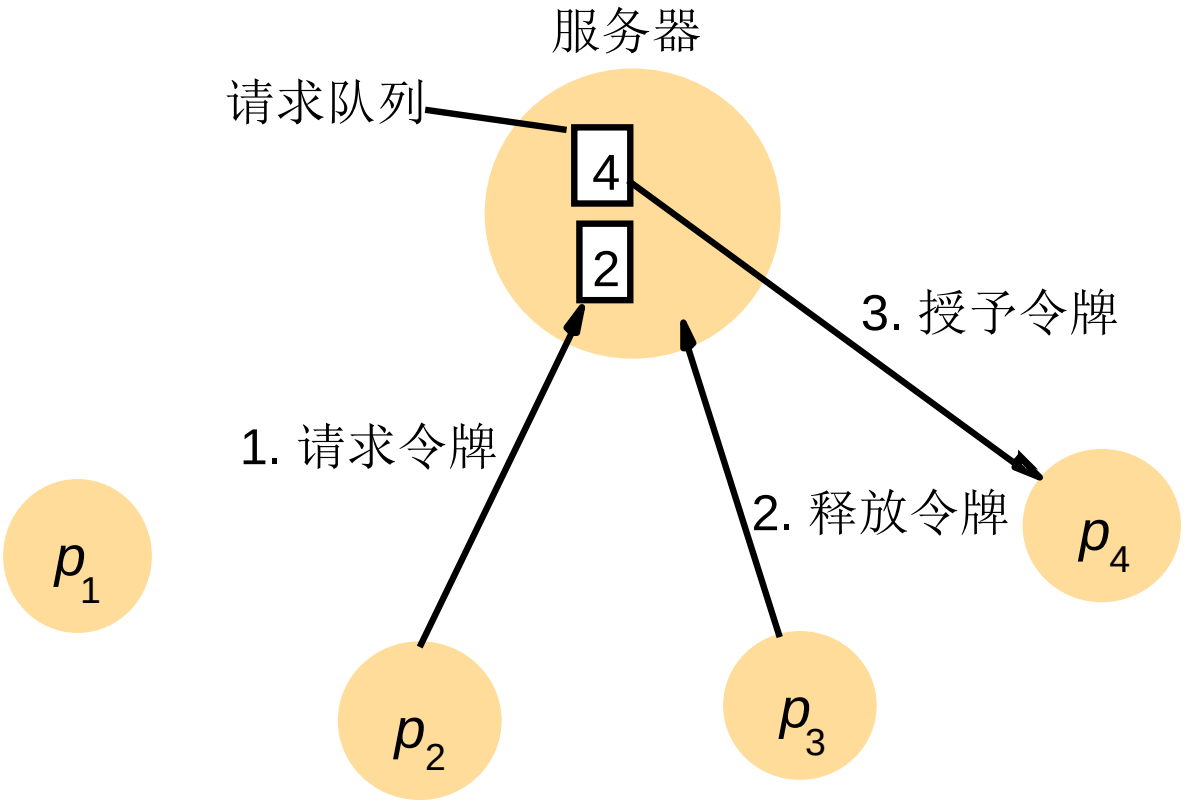
- 吞吐量

  用*同步延迟*来衡量，即一个进程离开临界区和下一个进程进入临界区之间的延迟。

分布式互斥

中央服务器算法

- 构架



分布式互斥

- 满足安全性和活性要求，但不满足顺序要求。
- 性能
  - 带宽消耗

    enter(): 2个消息，即请求消息和授权消息

    exit(): 1个消息，即释放消息
  - 客户延迟

    消息往返时间导致请求进程延迟
  - 同步延迟

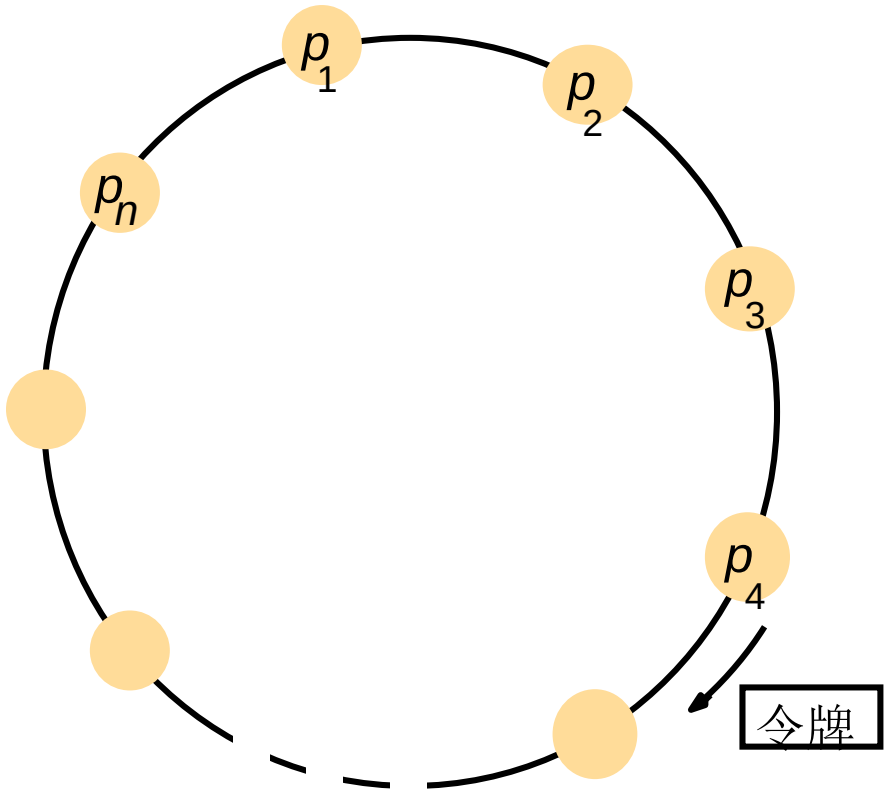
    1个消息的往返时间
  - 性能瓶颈

    服务器

分布式互斥

基于环的算法

- 构架



分布式互斥

- 满足安全性和活性要求，但不满足顺序要求。
- 性能
  - 带宽消耗

    由于令牌的传递，会持续消耗带宽
  - 客户延迟

    Min: 0个消息，正好收到令牌

    Max: N个消息，刚刚传递了令牌
  - 同步延迟

    Min: 1个消息，进程依次进入临界区

    Max: N个消息，一个进程连续进入临界区，期间无其他进程进入临界区

分布式互斥

- 使用组播和逻辑时钟的算法
  - 基本思想
    - 进程进入临界区需要所有其它进程的同意

      组播+应答
    - 并发控制

      采用Lamport时间戳避免死锁

分布式互斥

- 算法伪码

初始化:

state:=RELEASED;

为了进入临界区

state:=WAITED;

组播请求给所有进程;

T:=请求的时间戳;

Wait until (接收到的应答数=(N－1));

state:=HELD;

在p_j(i≠j)接收一个请求<T_i,p_i>

if (*state* = HELD or (*state* = WANTED *and* (T, p_j) < (T_i, p_i)))

then 将请求放入p_i队列,不给出应答;

else 马上给p_i应答;

end if

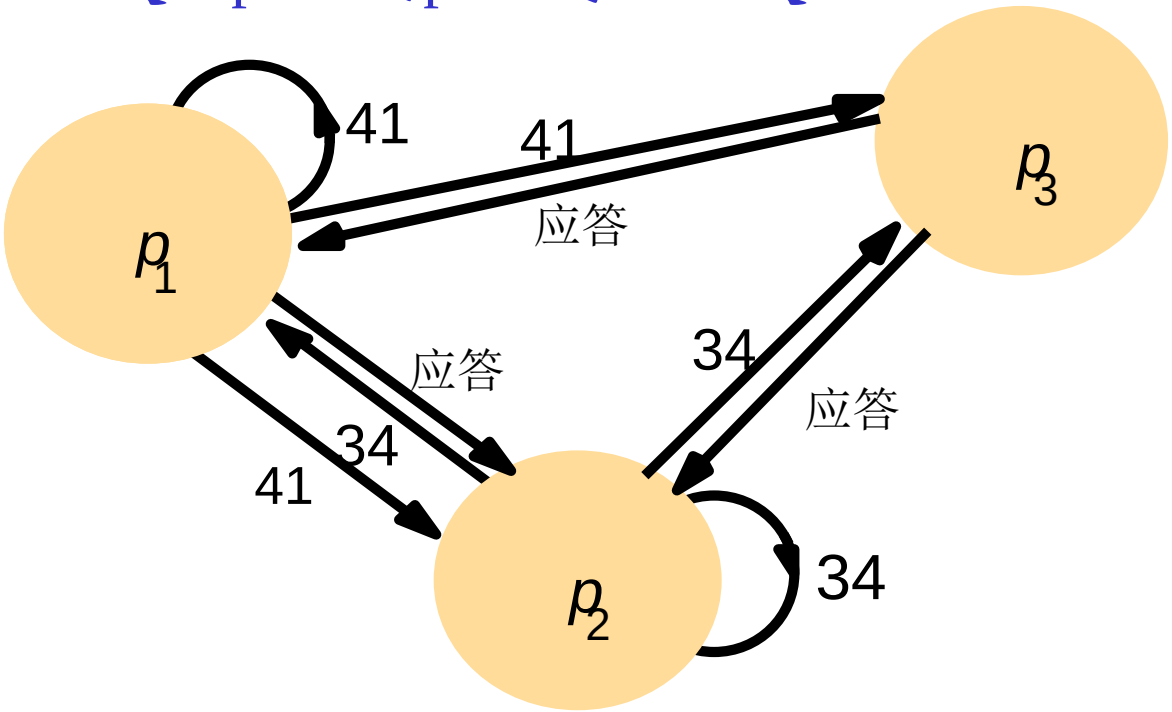
为了退出临界区;

state := RELEASED;

对已进入队列的请求给出应答;

分布式互斥

- 示例　p_1、p_2并发请求进入临界区（时间戳小优先）
- p3不进入，p1,p2竞争。
- p2先发时间戳34, p1后发时间戳41，p2收到p1后先不回复，p1收到p2后马上回复



分布式互斥

- 满足安全性、活性和顺序要求。
- 性能
  - 带宽消耗

    enter(): 2(N – 1)，即(N – 1)个请求、(N – 1)个应答
  - 客户延迟

    1个消息往返时间
  - 同步延迟

    1个消息的传输时间

分布式互斥

- **Maekawa投票算法**
  - 基本思想
    - 进程进入临界区需要部分其它进程的同意
    - 选举集
      - $V_i \subseteq \{p_1, p_2, \ldots, p_n\}$
      - $p_i \in V_i$
      - $V_i \cap V_j \neq \phi$
      - $|V_i| = K$, To be fair, $K \approx \sqrt{n}$
      - 每个进程p_j包括在选举集V_i中的M个集合中，$M = K$

分布式互斥

- 算法伪码

初始化:

state:=RELEASED;

voted:=FALSE;

p_i为了进入临界区

state:=WAITED;

将请求组播给v_i中的所有进程;

Wait until (接收到的应答数=K);

state:=HELD;

在p_j(i≠j)接收来自p_i的请求：

if (*state* = HELD or voted= TRUE)

将来自p_i的请求放入队列,不予应答;

else

将应答发送给p_i;

voted:=TRUE

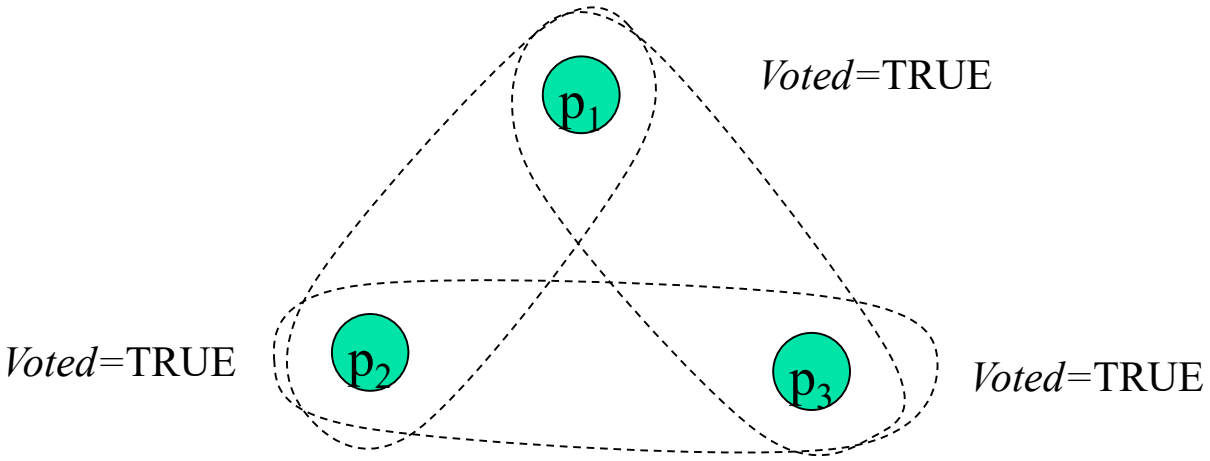
end if

分布式互斥

- Maekawa算法会产生死锁

三个进程p_1、p_2和p_3，且$V_1=\{p_1, p_2\}$，$V_1=\{p_2, p_3\}$，$V_1=\{p_3, p_1\}$。若三个进程并发请求进入临界区，考虑下列情况：

1. p_1应答了自己，但延缓p_2；
2. p_2应答了自己，但延缓p_3；
3. p_3应答了自己，但延缓p_1。



分布式互斥

- Maekawa算法改进后可满足安全性、活性和顺序性

  进程按发生在先顺序对待请求队列

- 性能

  - 带宽消耗

    $3\sqrt{N}$：即进入需要$2\sqrt{N}$个消息，退出需要$\sqrt{N}$个消息

  - 客户延迟

    1个消息往返时间

  - 同步延迟

    较差，1个往返时间，非单个消息的往返时间

第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结

选举

基本概念

- 选举算法

  选择一个唯一的进程来扮演特定角色的算法

- 召集选举

  一个进程启动了选举算法的一次运行

- 参加者

  进程参加了选举算法的某次运行

- 非参加者

  进程当前没有参加任何选举算法

- 进程标识符

  唯一且可按全序排列的任何数值

选举

- 基本要求
  - 安全性

    参与的进程p_i有 $elected_i = \perp$ 或 $elected_i = P$

    $\perp$表示该值还没有定义
  - 活性

    所有进程p_i都参加并且最终置$elected_i \neq \perp$或进程p_i崩溃
- 性能评价
  - 带宽消耗
  - 回转时间

    从启动算法到终止算法之间的串行消息传输的次数

选举

基于环的选举算法

- 目的

  在异步系统中选举具有最大标识符的进程作为协调者

- 基本思想

  按逻辑环排列一组进程

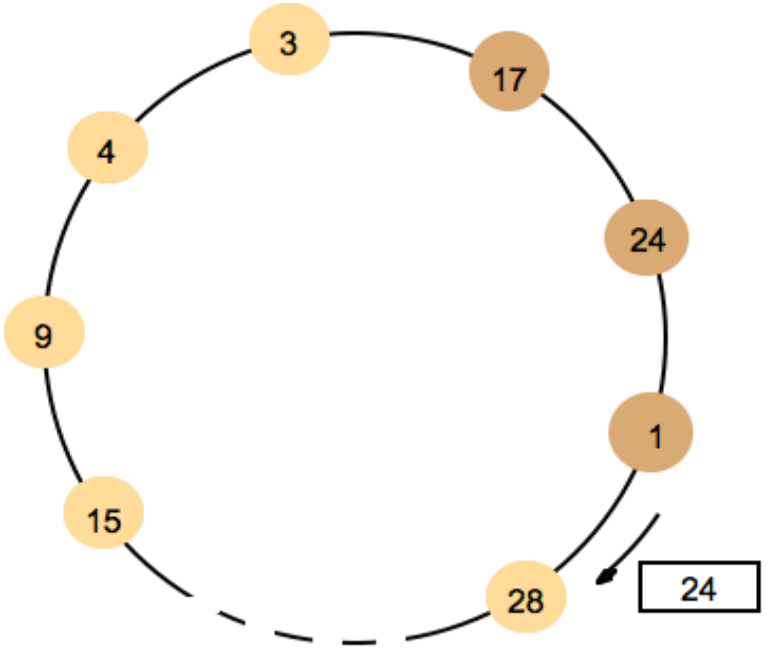
选举

- 算法
  - 最初，每个进程标记非参加者
  - 任一进程可以开始一次选举
    - 将自身标记为参加者
    - $id_{msg} = id_{local}$，发送 {elect, id_{msg}} 至邻居
  - 非参加者转发选举消息
    - 将自身标记为参加者
    - 发送 {elect, MAX(id_{local}, id_{msg})} 至邻居
  - 当 $id_{local}=id_{msg}$ 时，该进程成为协调者
    - 将自身标记为非参加者
    - $id_{coordinator} = id_{local}$，发送{elected, $id_{coordinator}$ } 至邻居
  - 参加者转发选举结果消息
    - 将自身标记为非参加者
    - 记录$id_{coordinator}$

选举

- 算法示例

选举从进程17开始。到目前为止，所遇到的最大的进程标识符是24。参与的进程用深色表示。



分布式互斥

- 性能
  - 最坏情况

    启动选举算法的逆时针邻居具有最大标识符，共计需要3N－1个消息，回转时间为3N－1。
  - 最好情况

    回转时间为2N。
  - 不具备容错功能

选举

霸道算法

- 假设
  - 系统是同步的，使用超时检测进程故障
  - 通道可靠，但允许进程崩溃
  - 每个进程知道哪些进程具有更大的标识符
  - 每个进程可以和所有具有更大标识符的进程通信

选举

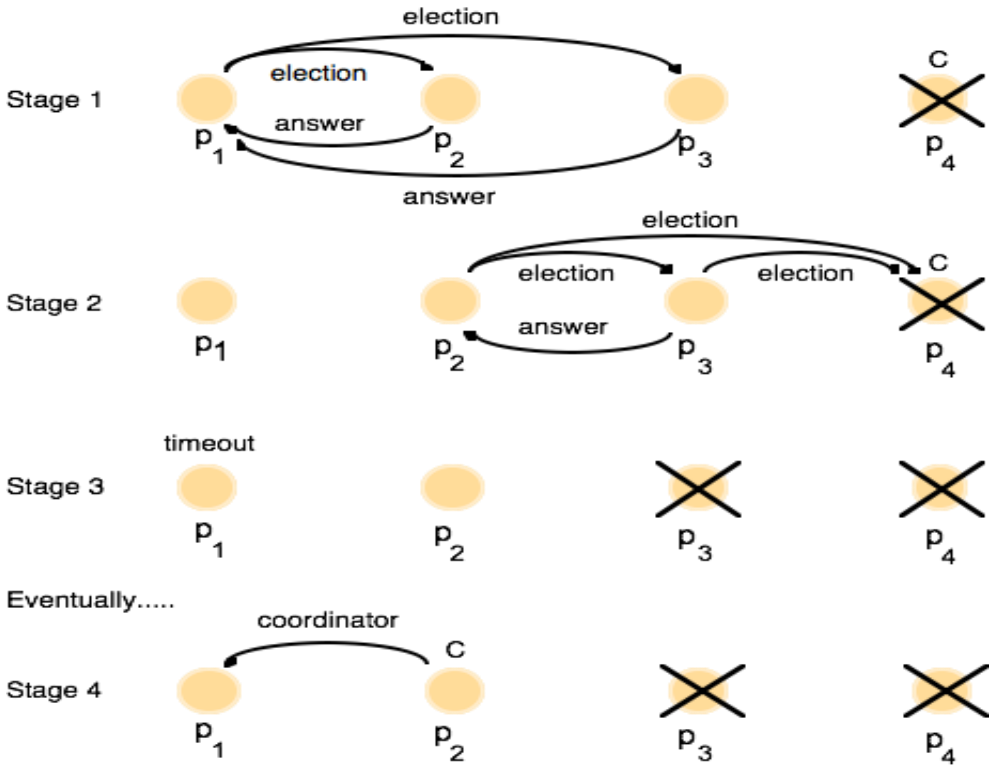
- 算法
  - 选举初始化

    进程P在发现协调者失效后启动一次选举，将选举消息发送给具有更大标识符的进程
  - 接收进程回送一个回答并开始另一次选举
  - 协调者（知道自己有最大标识符）发送协调者消息
    - 若进程P 没有收到回答消息，则给所有具有较小标识符的进程发送协调者消息。
    - 若进程P 收到回答消息，则等待协调者消息；若消息在一段时间没没有到达，则启动一次新的选举算法。
  - 进程收到协调者信息后，设置 electedi = idcoordinator

选举

- 算法示例

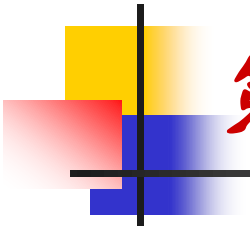
p_4、p_3相继出现故障



分布式互斥

- 性能
  - $\text{bandwidth}_{best} = N - 2$
    - 标识符次大的进程发起选举
    - 发送$N-2$个协调者消息
    - 回转时间为1个消息
  - $\text{bandwidth}_{worst} : O(N^2)$

    标识符最小的进程发起选举



第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结

组播通信

- **组播/广播**

  组播：发送一个消息给进程组中的每个进程

  广播：发送一个消息给系统中的所有进程

- **组播面临的挑战**
  - 效率
    - 带宽使用
    - 总传输时间
  - 传递保证
    - 可靠性
    - 顺序
  - 进程组管理

    进程可任意加入或退出进程组

组播通信

系统模型

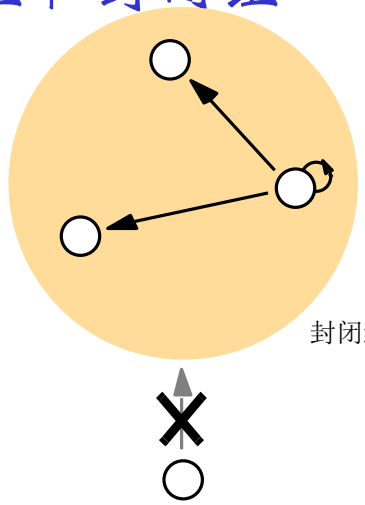
- multicast(g,m)

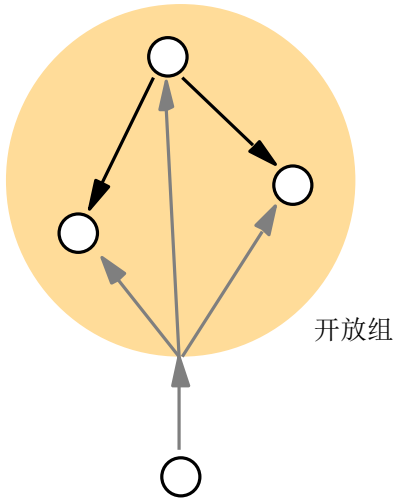
  1个进程发送消息给进程组g的所有成员

- deliver(m)

  传递由组播发送的消息到调用进程

- 开放组和封闭组





组播通信

- 基本组播
  - 一个正确的进程最终会传递消息
  - 原语: B-multicast、B-deliver
  - 可靠组播，与IP组播不同
- 简单实现
  - B-multicast(g,m)：对每个进程p∈g，send(p,m)
  - 进程p receive(m)时：p执行B-deliver(m)
- 多线程
  - 利用线程来并发执行send操作
- 确认爆炸
  - 确认从许多进程几乎同时到达
  - 组播进程丢弃部分确认消息导致重发现象

组播通信

可靠组播

- 性质

  -完整性

  一个正确的进程p传递一个消息m至多一次

  -有效性

  如果一个正确的进程组播消息m，那么它终将传递m。

  - 协定——具有原子性

  如果一个正确的进程传递消息m，那么在group(m)中的其它正确的进程终将传递m。

组播通信

- 用B-multicast实现可靠组播

-算法

初始化：

Received:={};

进程p为了将R-multicast消息发送给组g：

B-multicast(gm);

在进程q On B-deliver(m)时，其中g=group(m)

if (m∉Received)

then

Received:=Received ∪ {m}

if (q≠p) then B-multicast(g,m); end if

R-deliver m;

end if

组播通信

-算法评价

- 满足有效性

  一个进程的最终将B-deliver消息到它自己。

- 满足完整性

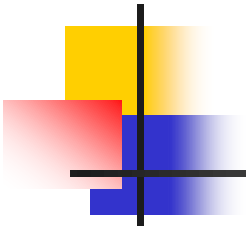
  B-multicast中的通信通道具有完整性

- 遵循协定

  每个正确的进程在B-deliver消息后都B-multicast该消息到其它进程

- 效率低

  每个消息被发送到，每个进程|g|次。



组播通信

- 用IP组播实现可靠组播

  - 特点

    - 基于IP组播

      IP组播通信通常是成功的

    - 捎带确认

      在发送给组中的消息中捎带确认

    - 否认确认

      进程检测到它们漏过一个消息时，发送一个单独的应答消息。

组播通信

-算法

S_g^p: 进程为它属于的组g维护的序号，初始化为0。

R_g^p: 进程记录来自进程q并且发送到组g的最近消息的序号

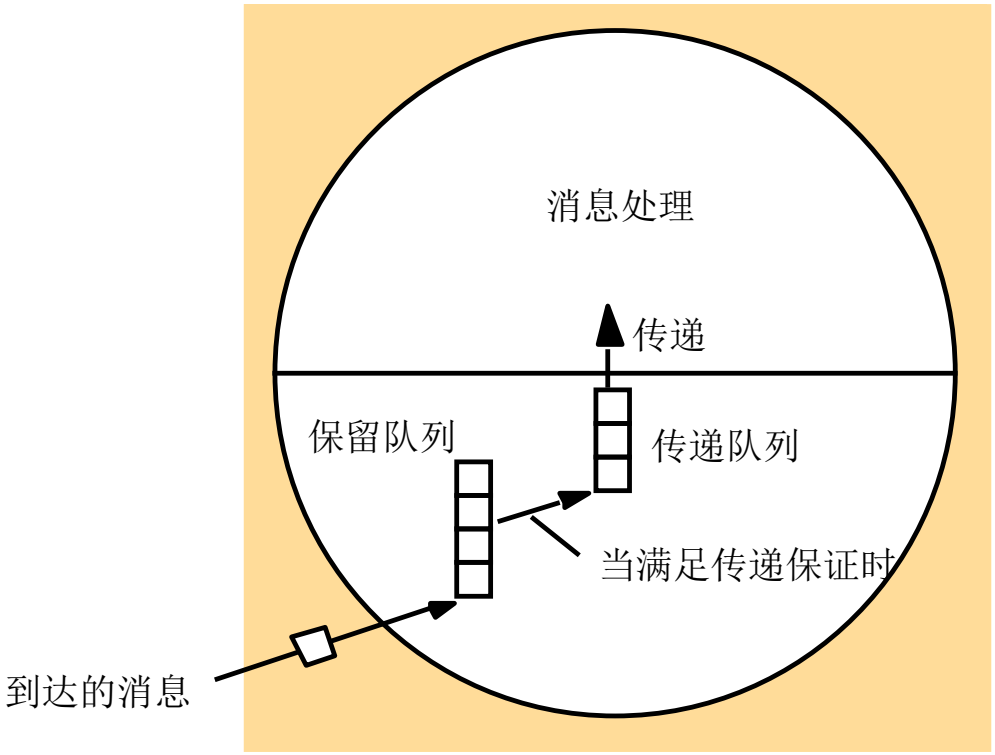
R-multicast 一个消息到组g: 捎带S_g^p和确认，即$<q, R_g^q>$;

$$S_g^p = S_g^p + 1$$

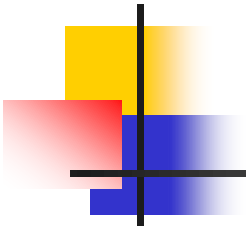
R-deliver 一个消息:

1. 当且仅当$m.S = R_g^p + 1$传递消息； $R_g^p = R_g^p + 1$
2. 若$m.S <= R_g^p$, 则该消息已传递，直接丢弃。
3. 若$m.S > R_g^p + 1$或对任意封闭的确认$<q, R_g^q>$有$m.R > R_g^q$, 则漏掉了一个或多个消息,将消息保留在保留队列中,并发送否认确认。

组播通信



保留队列



组播通信

-算法评价

- 完整性

  通过检测副本和IP组播性质实现

- 有效性

  仅在IP组播具有有效性时成立

- 协定

  进程无限组播消息时成立

- 某些派生协议实现了协定

组播通信

- 统一性质
  - 统一

    无论进程是否正确都成立的性质
  - 统一协定

    如果一个进程传递消息m，不论该进程是否正确还是出故障，在group(m)中的所有正确的进程终将传递m

组播通信

- 符合统一协定的算法示例

初始化：

Received:={};

进程p为了将R-multicast消息发送给组g：

B-multicast(gm);

在进程q On B-deliver(m)时，其中g=group(m)

if (m∉Received)

then

Received:=Received ∪ {m}

if (q≠p) then B-multicast(g,m); end if

R-deliver m;

end if

若颠倒这两行，则算法不满足统一协定

组播通信

- 有序组播
  - FIFO排序

    如果一个正确的进程发出multcast(g,m)，然后发出multicast(g,m')，那么每个传递m'的正确的进程将在m'前传递m。

  - 因果排序

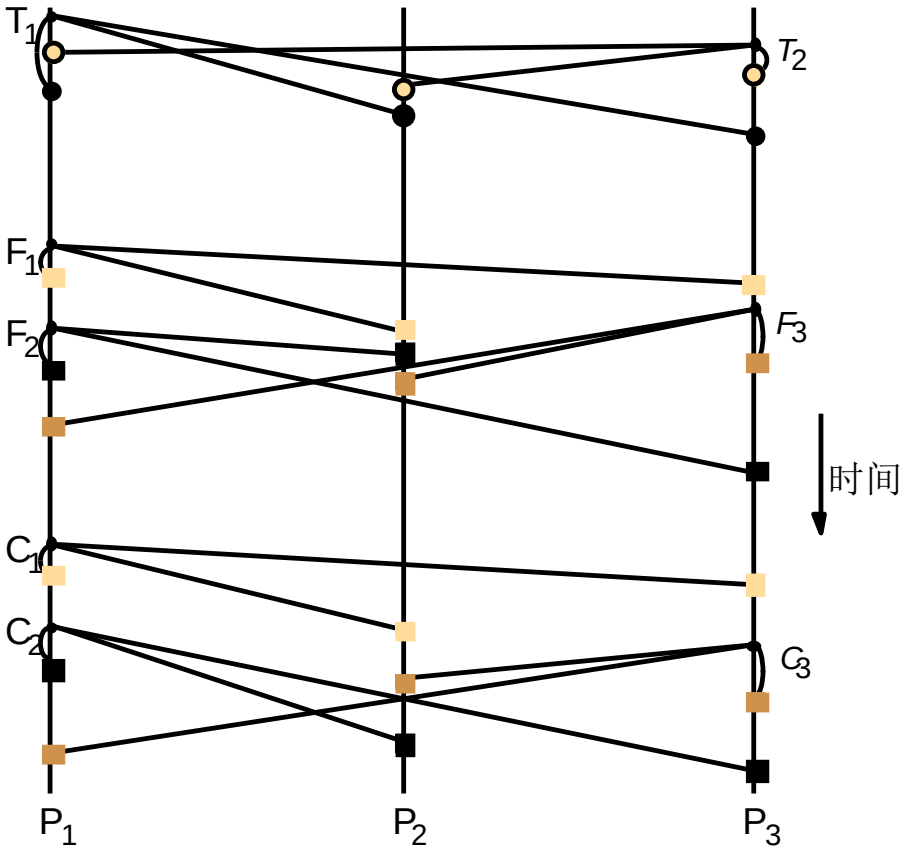
    如果multcast(g,m) → multicast(g,m')，那么任何传递m'的正确进程将在m'前传递m。

  - 全排序

    如果一个正确的进程在传递m'前传递消息m，那么其它传递m'的正确进程将在m'前传递m。

组播通信

- 不同排序示例（全排序，FIFO，因果排序）



组播通信

- 公告牌的例子

公告牌：对操作系统感兴趣的		
编号	张贴人	主题t
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
结束		

组播通信

- 实现FIFO排序
  - 基于序号实现
  - FO-multicast/FO-deliver
  - 算法

    与基于IP组播的可靠组播类似，即采用S_g^p、R_g^q和保留队列

组播通信

- 实现全排序
  - 为组播消息指定全排序标识符
  - TO-multicast/TO-deliver
  - 使用顺序者的全排序算法

    1. 组成员p的算法

    初始化：r_g:=0;

    为了给组g发TO-multicast消息：

    B-multicast(g∪{sequencer(g)},<m,i>);

    在B-deliver(M_{order}=<"order",I,s>)时，其中g=group(M_{order})

    Wait until <m,i>在保留队列中并且S=r_g;

    To-deliver m; //在从保留队列删除它之后

    rg=S+1;

组播通信

- 使用顺序者的全排序算法(续)

  2. 顺序者g的算法

  初始化：s_g:=0;

  在B-deliver(<m,i>)时，其中g=group(M_{order})

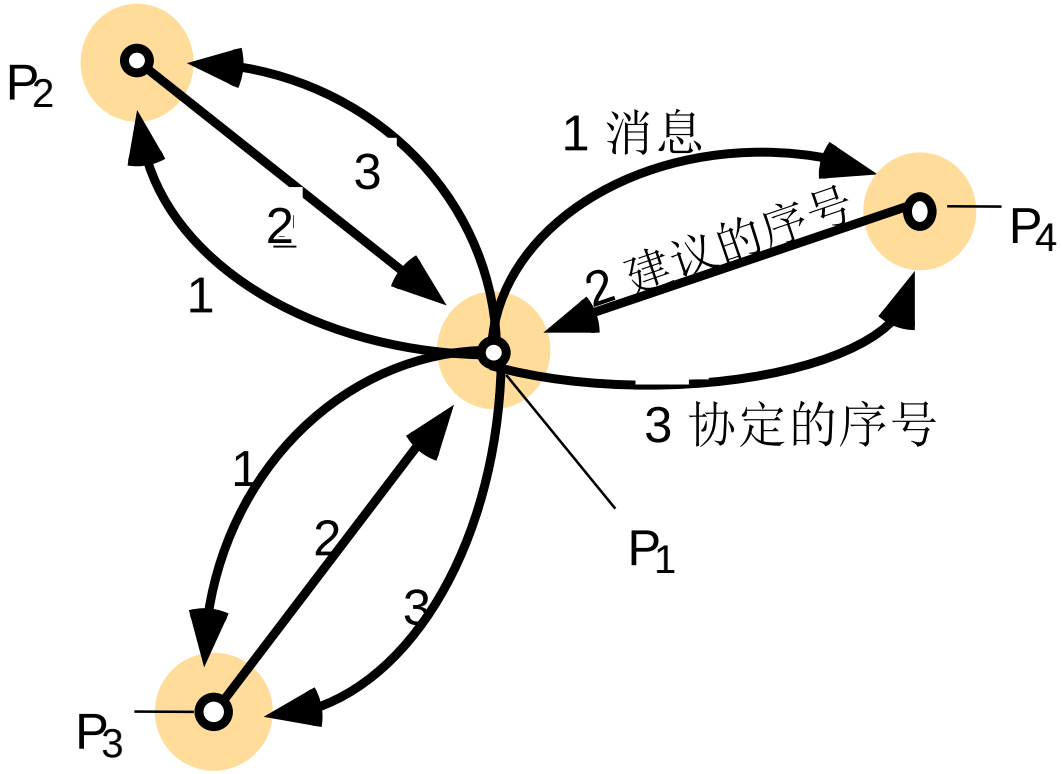
  B-multicast(g,<“orer”,I,s_g>);

  s_g:=s_g+1;

- 基于顺序者的算法缺点：顺序者会成为瓶颈

组播通信

- 全排序的ISIS算法



组播通信

- 全排序的ISIS算法(序)

A_g^q: 进程迄今为止 从组g观察到的最大的协定序号

P_g^q: 进程自己提出的最大序号

进程p组播消息m到组g的算法：

1. p B-multicasts <m, i>到g， 其中i 是m的一个委员的标识符
2. 每个进程： (1) $P_g^q = \max(A_g^q, P_g^q)+1$; (2)把 P_g^q添加到消息m， 并把m放入保留队列; (3)用序号P_g^q 回答p
3. P收集P_g^q,选择最大的数a作为下一个协定序号， 然后 B-multicast<i,a> 到g
4. g中的每个进程q置$A_g^q := \max(A_g^q, a)$, 并把a附加到消息上。

组播通信

- 全排序的ISIS算法(序)

1. 正确的进程最终会对同一组序号达成一致；
2. 序号是单调递增的；
3. 不保证因果或FIFO序；
4. 比基于顺序者的组播有更大的延迟

组播通信

- 实现因果排序
  - 非重叠封闭组算法，只考虑由组播消息建立的发生在先关系
  - 向量时间戳

    每个进程维护自己的向量时间戳
  - CO-multicast

    在向量时间戳的相应分量上加1，附加时间戳到消息
  - CO-deliver

    根据时间戳递交消息

组播通信

- 使用向量时间戳的因果排序算法

对组成员p_i (i=1,2,…,N)的算法

初始化：

$V_i^g[j]:=0$ (j=1,2,…M);

为了给组g发CO-multicast消息m:

$V_i^g[j]=V_i^g[j]+1$;

B-multicast(g,<V_i^g,m>);

在B-deliver(<V_i^g,m>)来自p_j(i≠j)的一个消息时，其中g=group(m);

将<V_i^g,m>放入保留队列,直到$V_i^g[j]=V_i^g[j]+1$和$V_i^g[j]=V_i^g[j]+1$ (k ≠j);

CO-deliver m; //在把它从保留队列删除后

$V_i^g[j]=V_i^g[j]+1$;

组播通信

- 组重叠

  - **全局FIFO排序**

    如果一个正确的进程发出multicast(g,m)，然后发出multicast(g',m')，则两个消息被发送到g∩g'的成员。

  - **全局的因果排序**

    如果multicast(g,m) → multicast(g',m')，则g∩g'中的任何传递m'的正确进程将在m'前传递m。

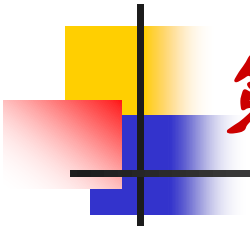
组播通信

- 进程对的全排序

  如果一个正确的进程在传递发送到g'的消息m'前传递了发送到g的消息m，则g∩g'中的任何传递m'的正确进程将在m'前传递m。

- 全局的全排序

  令“<”是传递事件之间的排序关系。我们要求“<”遵守进程对的全排序，并且无环。



第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结

共识和相关问题

简介

- 分布式系统中的协定问题
  - 互斥：哪个进程可以进入临界区
  - 全排序组播：组播消息的顺序
  - 拜占庭将军：进攻还是撤退
- 共识问题
  - 一个或多个进程提议了一个值后，应达成一致意见
  - 共识问题、拜占庭将军和交互一致性问题
- 故障模型
  - 进程崩溃故障、拜占庭进程故障

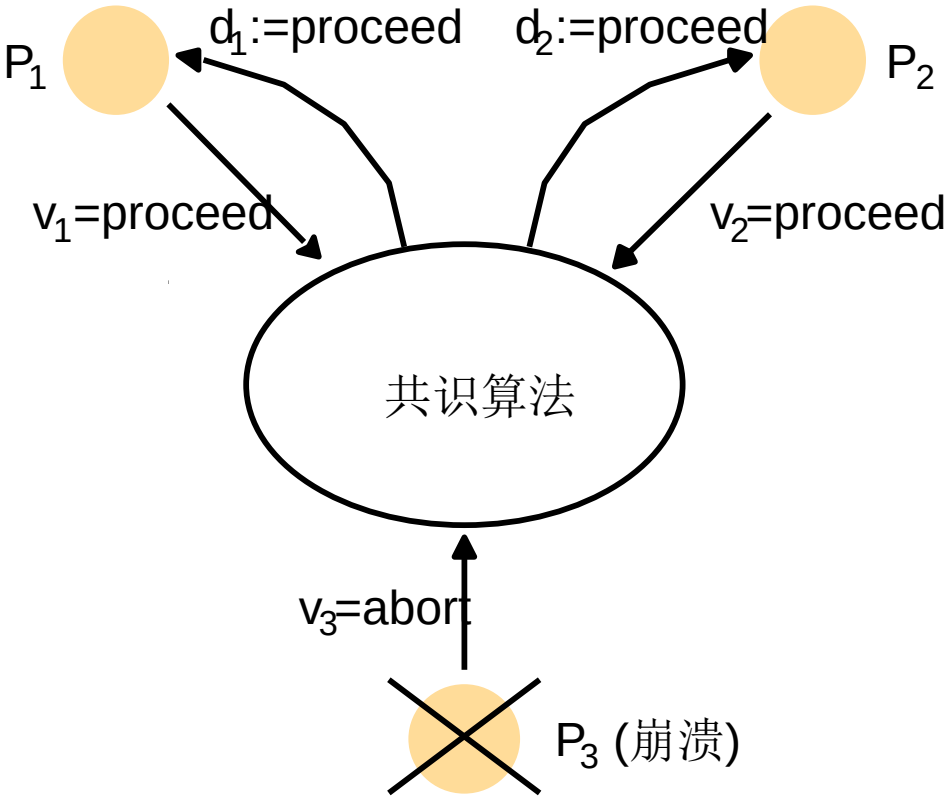
共识和相关问题

共识问题定义

- 符号
  - p_i: 进程i
  - v_i: 进程p_i的提议值
  - d_i: 进程p_i的决定变量

共识和相关问题

- 3个进程的共识问题示例



共识和相关问题

- 共识算法的基本要求

  - 终止性

    每个正确的进程最终设置它的决定变量

  - 协定性

    如果p_i和p_j是正确的且已进入决定状态，那么$d_i=d_j$，其中i,j=12,…N。

  - 完整性

    如果正确的进程都提议了同一个值，那么处于决定状态的任何正确进程已选择了该值。

共识和相关问题

- 算法
  - 每个进程组播它的提议值
  - 每个进程收集其它进程的提议值
  - 每个进程计算 $V = \text{majority}(v_1, v_2, \ldots, v_N)$

    majority()函数为抽象函数，可以是max()、min()等等
- 算法分析
  - 终止性

    由组播操作的可靠性保证
  - 协定性和完整性

    由majority()函数定义和可靠组播的完整性保证

共识和相关问题

拜占庭将军问题

- 问题描述
  - 3个或更多的将军协商是进攻还是撤退
  - 一个或多个将军可能会叛变
  - 所有未叛变的将军执行相同的命令
- 与共识问题的区别

  每个进程(将军)都提议一个值
- 算法要求
  - 终止性
  - 协定性
  - 完整性

共识和相关问题

交互一致性

- 就一个值向量达成一致
  - 决定向量: 向量中的每个分量与一个进程的值对应
- 算法要求
  - 终止性

    每个正确进程最终设置它的决定变量
  - 协定性

    所有正确进程的决定变量都相同
  - 完整性

    如果进程p_i是正确的,那么所有正确的进程都把v_i作为他们决定向量中的第i个分量.

共识和相关问题

- 共识问题与其它问题的关联
  - 目的

    重用已有的解决方案
  - 问题定义
    - 共识问题C

      $C_i(v_1,v_2,\ldots v_N)$: 返回进程p_i的决定值
    - 拜占庭将军BG

      $BG_i(j,v)$: 返回进程p_i的决定值,其中p_j是司令,它建议的值是v
    - 交互一致性问题IC

      $IC_i(v_1,v_2,\ldots,v_N)[j]$: 返回进程$p_i$的决定向量的第j个分量

共识和相关问题

- 从BG构造IC
  - 将BG算法运算N次,每次都以不同的进程pi作为司令
  - $IC_i(v_1,v_2, \ldots, v_N)[j] = BG_i(j,v), (i, j = 1, 2, \ldots, N)$
- 从IC构造C
  - $C_i(v_1,v_2, \ldots, v_N) = majority(IC_i(v_1,v_2, \ldots, v_N)[1],\ldots, IC_i(v_1,v_2, \ldots, v_N)[N])$
- 从C构造BG
  - 司令进程p_j把它提议的值v发送给它自己以及其余进程
  - 所有的进程都用它们收到的那组值$v_1,v_2, \ldots, v_N$作为参数运行C算法
  - $BG_i(j,v) = C_i(v_1,v_2, \ldots, v_N), (i = 1, 2, \ldots, N)$

共识和相关问题

同步系统中的共识问题

- 故障假设

  N个进程中最多有f个进程会出现崩溃故障

- 算法

对$p_i \in g$的进程算法：算法进行到f+1轮

初始化：

$valuses_i^1:=\{v_i\}$; $Values_i^0=\{\}$;

在第r轮($1 \le r \le f+1$)

B-multicast($g, Values_i^r - Values_i^{r-1}$); //仅发送还没有发送的值

$Values_i^{r+1}:=Values_i^r$;

While(在第r轮) {

在B-deliver(V_j)来自p_j的消息时: $Values_i^{r+1}:=Values_i^{r+1} \cup V_j$;}

在(f+1)轮之后

将d赋成min($Values_i^{f+1}$);

共识和相关问题

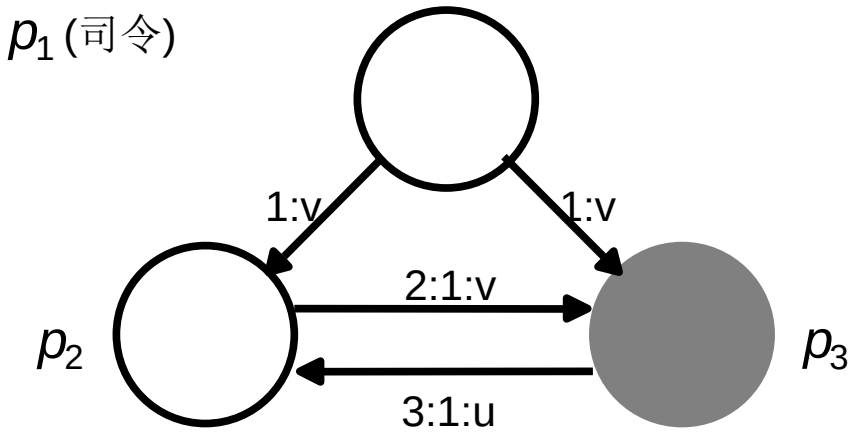
- 算法分析
  - 终止性质
    - 由同步系统保证
  - 协定性和完整性
    - 假设p_i得到的值是v,而p_j不是
    - P_{k1}在把v传送给p_i后,还没来得及传送给p_j就崩溃了
    - P_{k2}在把v传送给p_i后,还没来得及传送给p_j就崩溃了
    - …
    - $P_{k(f+1)}$在把v传送给p_i后,还没来得及传送给p_j就崩溃了
    - 但我们假设至多有f个进程崩溃
    - 因此,p_i和p_j的值相同
    - 因此,$min(Values_i^{f+2})$相同

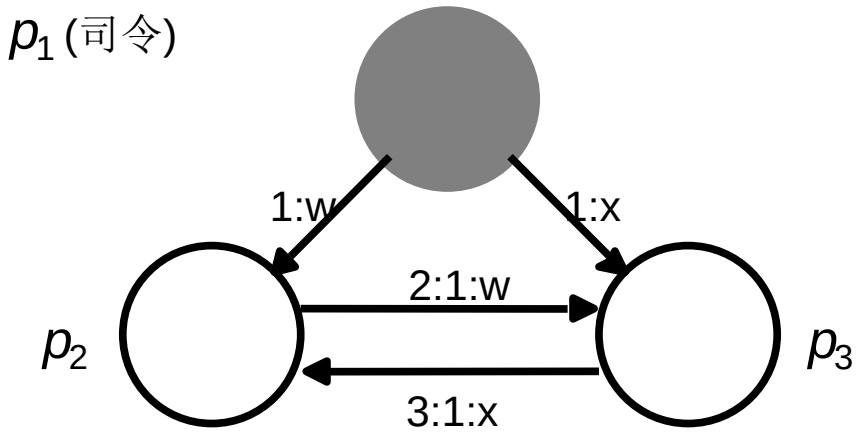
共识和相关问题

- 同步系统中的拜占庭将军问题
  - 随机故障假设
    - N个进程中最多有f个进程会出现随机故障
  - $N \leq 3f$
    - 无解决方法
  - $N \geq 3f+1$
    - Lamport于1982给出了解决算法

共识和相关问题

- 三个拜占庭将军
  - 三个进程的不可能性





有故障的进程用灰色表示

共识和相关问题

- 三个进程的不可能性
  - 如果存在一个解决方法，在左边的场景中，根据完整性条件，p_2选择1:v
  - 由于p_2不能分别这两个场景，因此p_2在右边的场景中选择1:w
  - 根据对称性，p_3在右边的场景中选择1:x
  - 与协定定义矛盾
- 三个进程实现拜占庭协定
  - 对发出的消息使用数字签名

共识和相关问题

- 对于$N \leq 3f$的不可能性

  - n_1, n_2, n_3

    将N个将军分成3组，$n_1+n_2+n_3=N$且$n_1, n_2, n_3 \leq N/3$

  - p_1, p_2, p_3

    让进程p_1, p_2, p_3分别模仿n_1, n_2, n_3个将军

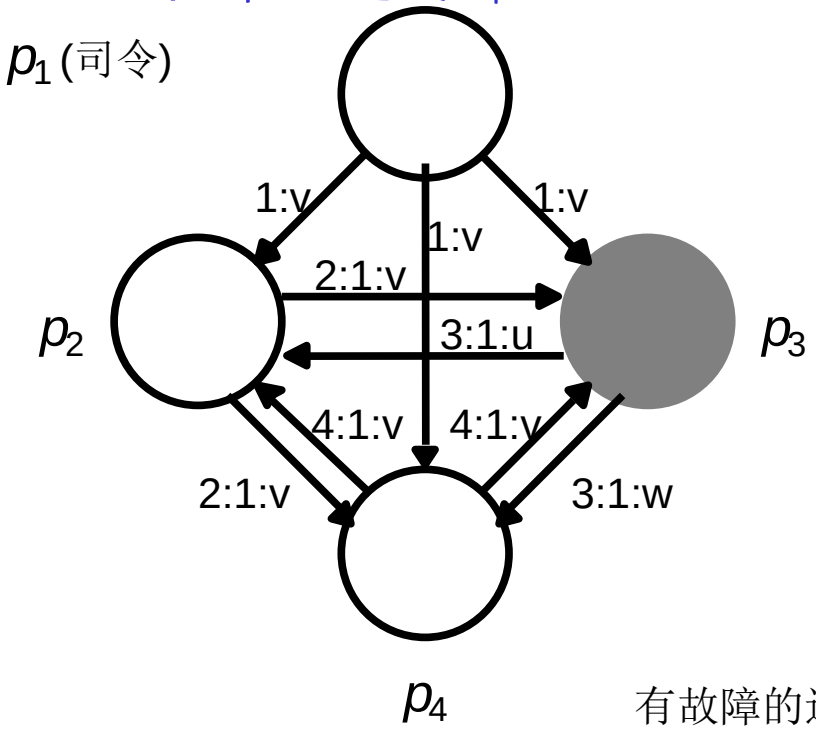
  - 根据对称性，p_3在右边的场景中选择1:x
  - 若存在一个解决方法，即达成一致且满足完整性条件
  - 与三个进程的不可能性结论矛盾

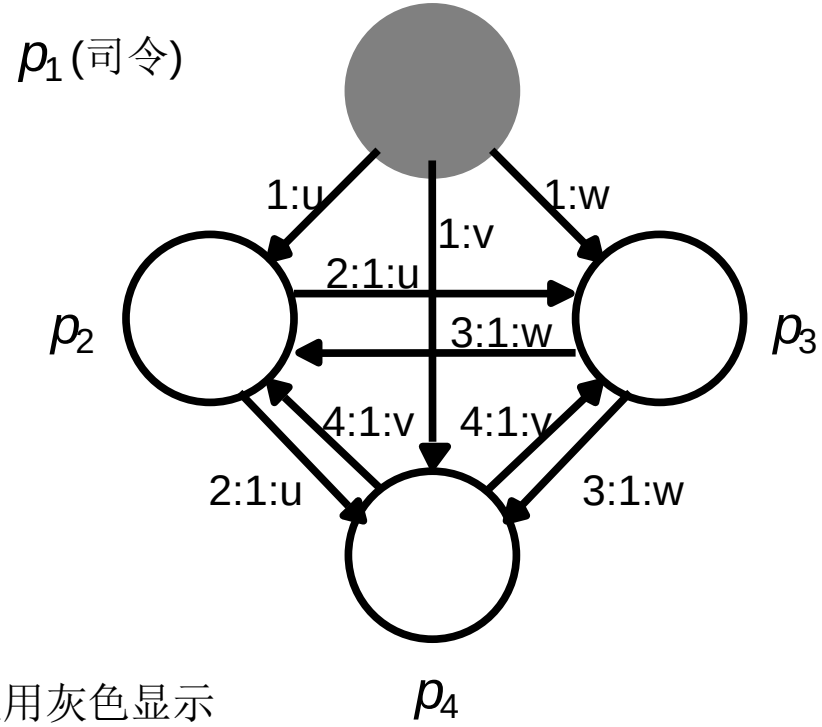
共识和相关问题

- 对一个有错进程的解决方案
  - 假设N=4,f=1
  - 正确的将军通过两轮消息取得一致
    - 第一轮，司令给每个中尉发送一个值
    - 第二轮，每个中尉将收到的值发送给与自己同级的
    - 每个中尉执行majority()函数

共识和相关问题

- 4个拜占庭将军示例





有故障的进程用灰色显示

左边： d2 = majority(v, u, v) = v，

d4 = majority(v, v, w) = v

右边： $d_2 = d_3 = d_4$ = majority(u, v, w) = ⊥

共识和相关问题

- 性能讨论

- 衡量标准
  - 进行了多少轮消息传递？
  - 发送了多少消息，消息的长度是多少？

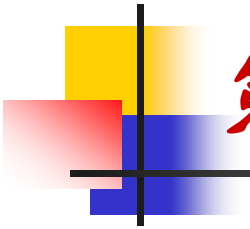
- Lamport算法
  - f+1轮转
  - $O(N^{f+1})$条消息

- Fischer和Lynch于1982年证明（FLP定理）
  - 如果允许出现拜占庭故障，那么任何确定性的解决共识问题的算法至少需要f+1轮消息传递。

共识和相关问题

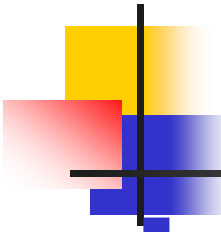
异步系统的不可能性

- 没有算法能够保证达到共识
  - 无法分辨一个进程是速度很慢还是已经崩溃
- 故障屏蔽
  - 屏蔽发生的所有进程故障
- 使用故障检测器达到共识
  - 在仅依靠消息传递的异步系统中，不存在完美的故障检测器，甚至是最弱的故障检测器。
- 使用随机化达到共识
  - 引入一个关于进程行为的可能性元素。



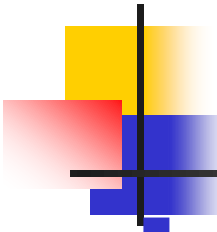
第七章 协调和协定

- 简介
- 分布式互斥
- 选举
- 组播通信
- 共识和相关问题
- 小结



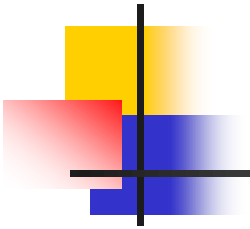
小结

- 分布式互斥
  - 中央服务器算法
  - 基于环的算法
  - 使用组播和逻辑时钟的算法
  - Maekawa投票算法
- 选举
  - 基于环的选举算法
  - 霸道算法



小结

- 组播通信
  - 基本组播
  - 可靠组播
  - 有序组播
- 共识
  - 共识
  - 拜占庭将军
  - 交互一致性



思考题

- 中文书P396
- 15.4 在用于互斥的中央服务器算法中，描述使得两个请求不是按照发生在先顺序处理的情景
- 15.5
- 15.8