

# 高级操作系统

## Advanced Operating System

---

### Distributed Systems

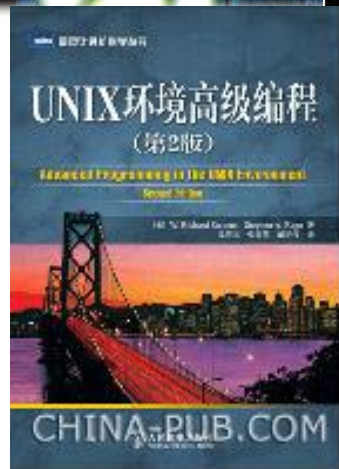
### Principles and Paradigms

朱 青 Qing Zhu,

Department of Computer Science, Renmin University of China

[zqruc2012@aliyun.com](mailto:zqruc2012@aliyun.com)

# Chapter 6 Synchronization 同步化



Teacher: Qing Zhu 朱青

Department of Computer Science,  
Information School,  
Renmin University of China  
zqruc2012@aliyun.com

# Textbook

---



- ⌘ **Distributed Systems – Concepts and Design, George Coulouris, Jean Dollimore, and Tim Kindberg Edition 3,4, © Pearson Education 2001, 2005**
- ⌘ **The lecture is based on this textbook.**

# 第6章 同步化

---

- 6.1 物理时钟同步
- 6.2 逻辑时钟同步
- 6.3 选举算法
- 6.4 互斥控制
- 6.5 分布式系统中的死锁

# 5.1 物理时钟同步

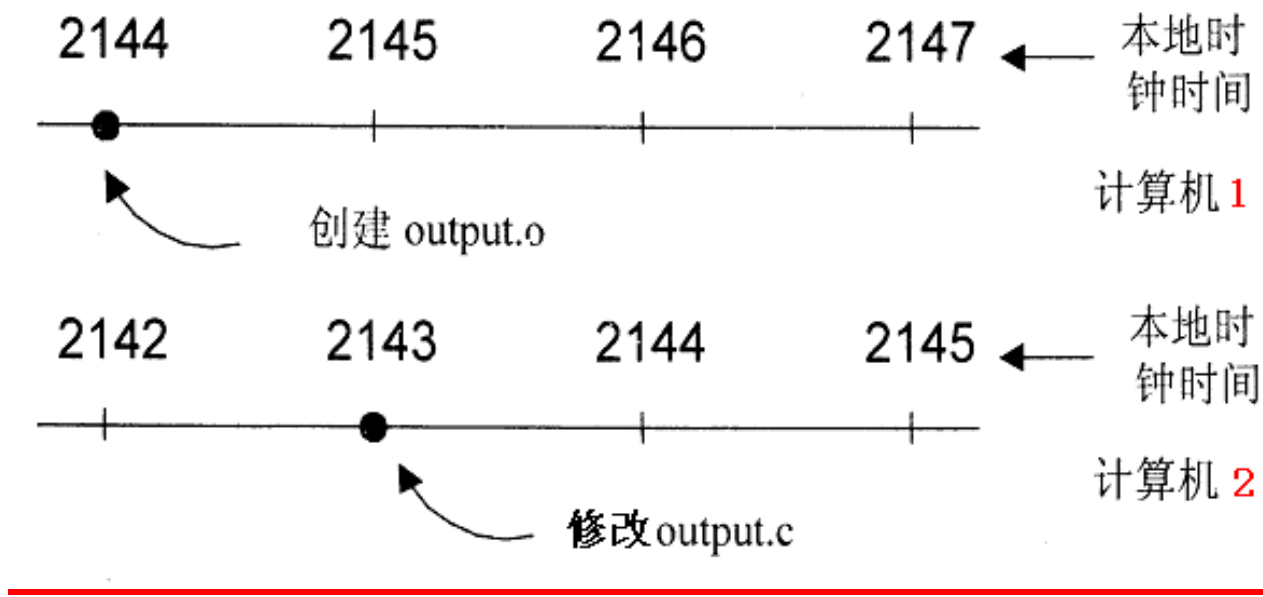
---

- 分布式协同处理
  - 基于时间的同步
- 分布式算法的特点
  - 相关信息散布在多个场地上
  - 每个进程只能基于本地信息做决定
  - 应避免因单点失败造成整个系统的失败
  - 不存在公共时钟或精确的全局时间

# 时钟同步问题

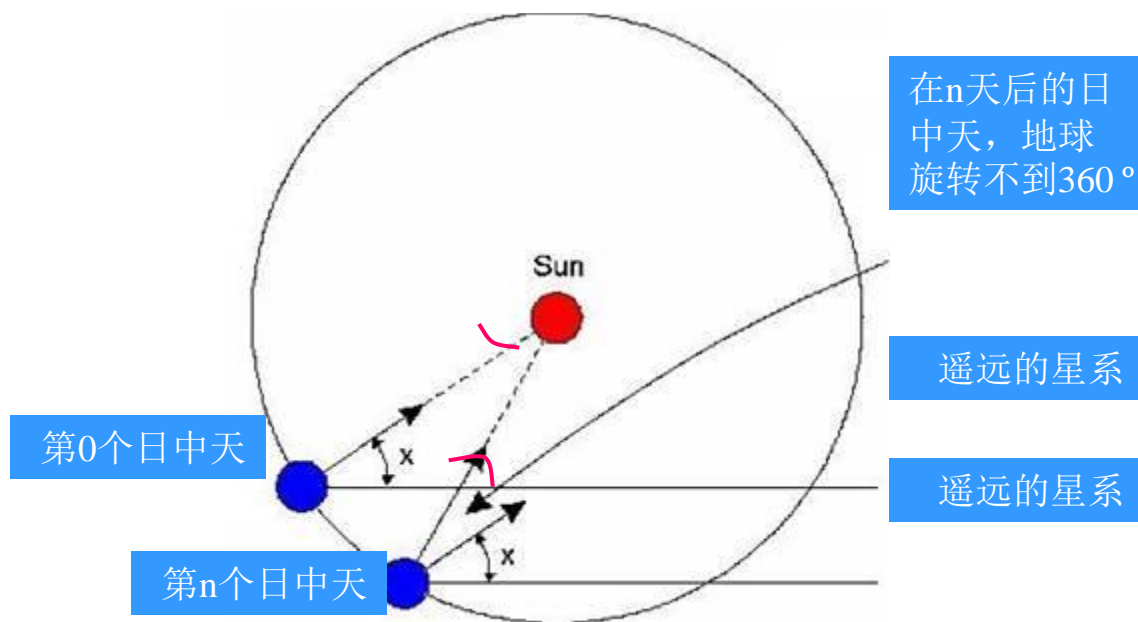
- 例：makefile误差

```
output.o :  
    cc -C output.c
```



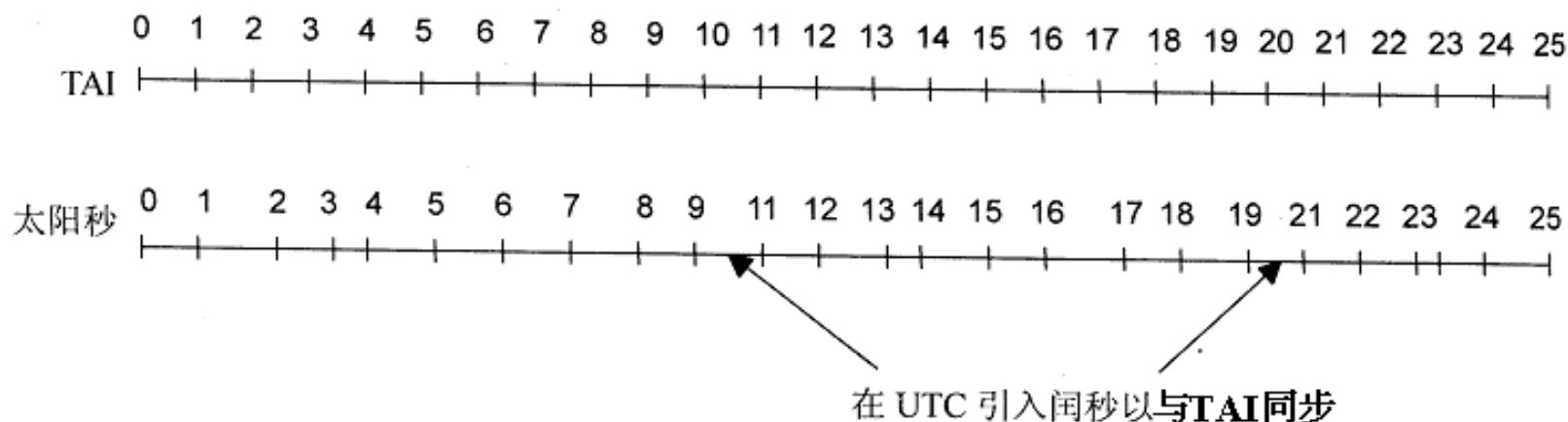
# 物理时钟与现实时钟

- 日中天 (**transit of the sun**) : 太阳升到一天的最高点
- 太阳日: 连续的两次日中天的时间
- 太阳年: 地球围绕太阳旋转 $360^\circ$  (一周) 。
- 太阳秒:  $\text{solar-day}/24 \times 3600 = \text{solar-day}/86400$
- 平均太阳秒:  $n \text{ solar-days} / n \times 86400$ 
  - 如, 格林威治时间



# 现实时钟

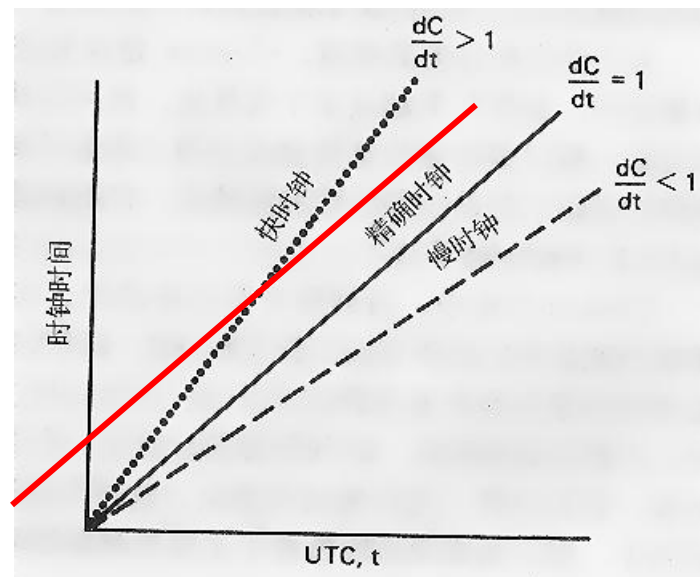
- 铯133原子钟：9, 192, 631, 770次跃迁/1秒
- TAI秒：国际原子时间
- UTC秒：世界时间（在TAI秒中加入闰秒）
- Universal Coordinated Time 统一协调时间
- 时间服务：WWV电台、GEOS卫星





# 时钟同步算法

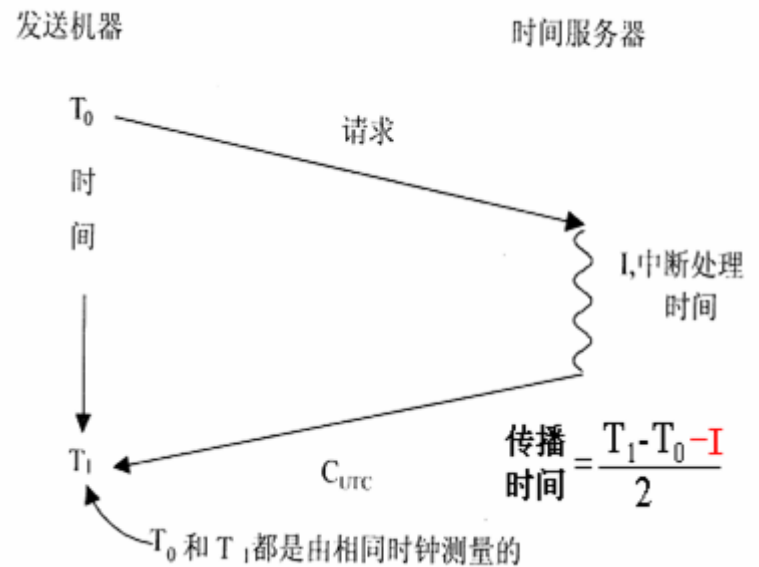
- ✓ 如何与现实时钟同步
- ✓ 如何使不同机器之间相互同步
- 设进程P的机器时钟值 $C_p(t)$ ,  
     $t$  为UTC时间
- 最大偏移率
  - 精确时钟:  $dC/dt = 1$
  - 快时钟:  $dC/dt > 1$
  - 慢时钟:  $dC/dt < 1$



# Christian算法 -- 逐步调整法

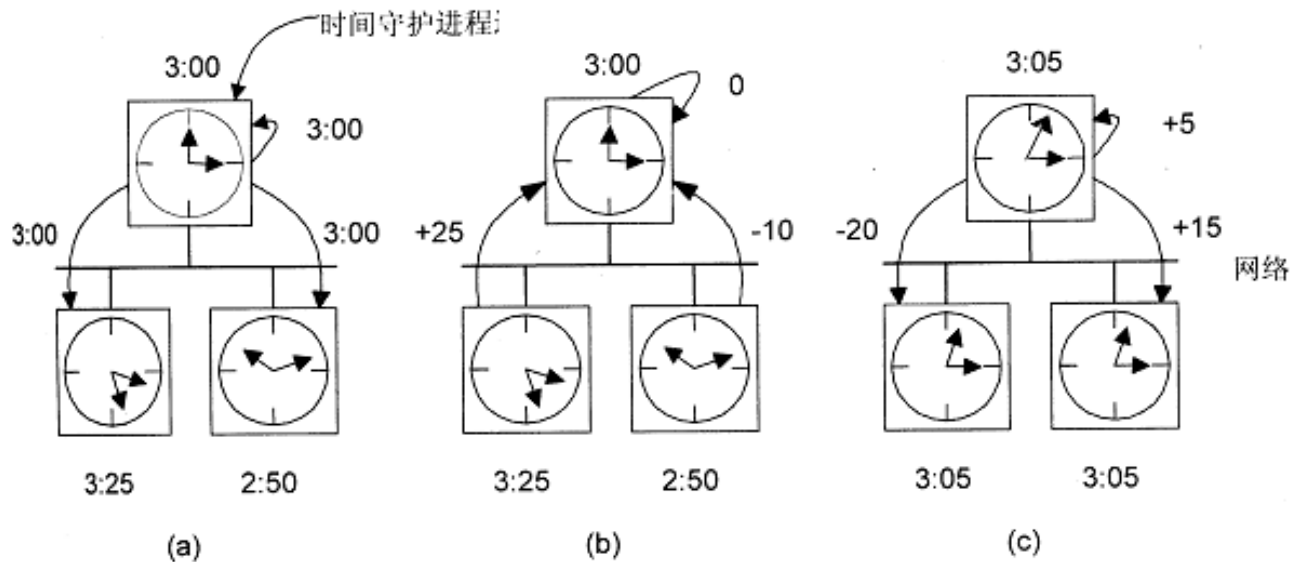
- 时间服务器，可接受WWV的UTC时间
- 每隔  $\delta / (2\rho)$  校准时间（允许误差  $\delta$ ，存在误差  $\rho$ ）
- ✓ 校准原则：单调递增  $\uparrow$

- 假设：每秒产生100次中断，每次中断将时间加10毫秒
- ✓ 若调慢时钟，中断服务程序每次只加9毫秒；
- ✓ 若加快时钟，则加11毫秒。
- ✓ 传播时间  $= (T_1 - T_0 - I) / 2$



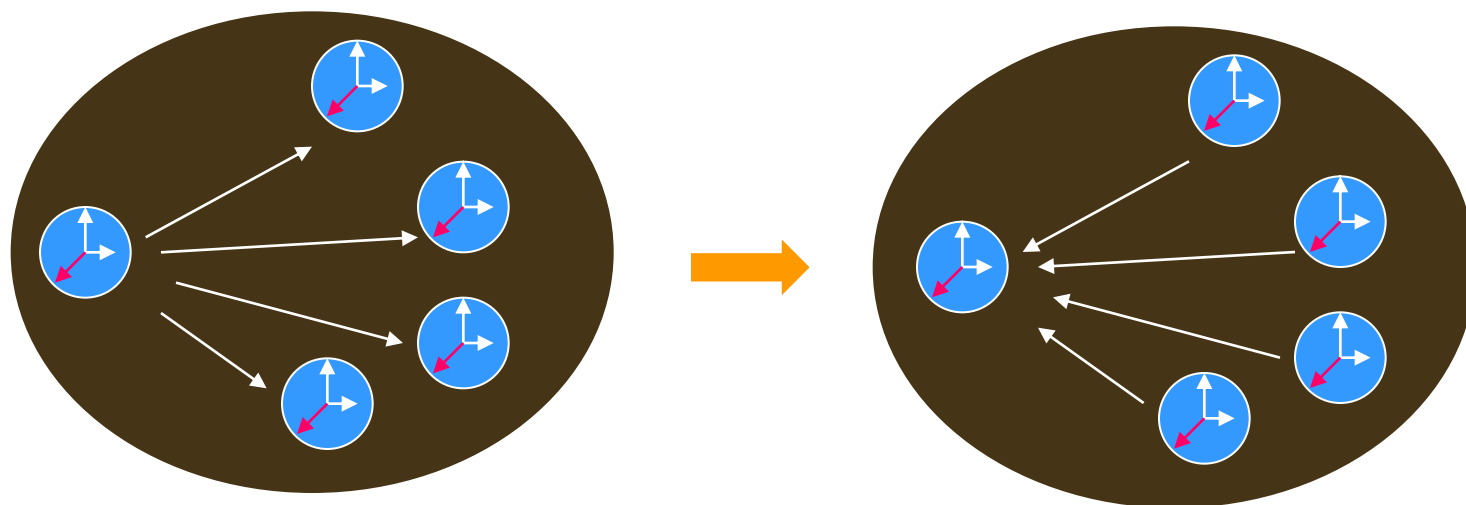
# Berkeley算法--主动式方法

1. 时间监控器定期查询其他机器时间
2. 计算出平均值
3. 通知其他机器调整时间



# 平均值算法--非集中式方法

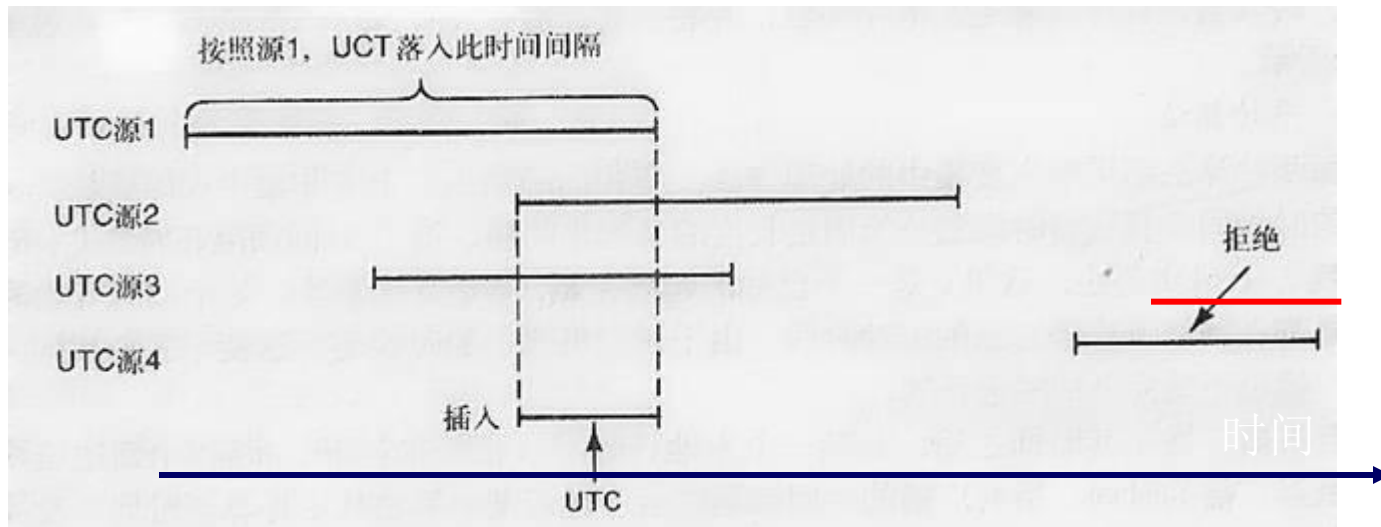
1. 划分固定时间间隔 $R$ ;
2. 在每个间隔, 所有机器广播自己的时钟时间
3. 启动本地计时器收集在 $S$ 时间间隔中到达的其他机器广播的时间
4. 执行平均时间计算算法, 得到新的时间值



# 多重外部时间源法

## □ 例：OSF DCE方法

1. 接受所有时间源的当前UTC区间
2. 去掉与其他区间不相交的区间
3. 将相交部分的中点作为校准时间



# 同步时钟的应用

---

- 最多一次消息提交
  1. 每个消息携带一个ID和一个时间戳 $ts$  (timestamp)
  2. 服务器的表 $T$ 中, 记录每个连接 $C$ 最近的时间印 $t$
  3. 对到达的消息 $m$ , 如果 $ts(m) < t$ , 则拒绝 $m$

## ➤ 服务器设置的全局变量

$$G = \text{CurrentTime} - \text{MaxLifetime} - \text{MaxClockSkew}$$

- 所有 $< G$ 的时间印从表 $T$ 中清除
- 对于具有新的ID的到达消息 $m$ , 如果 $ts(m) < G$ 则拒绝 $m$ , 否则, 接受 $m$
- 按照 $\Delta T$ , 定期地将 $G$ 写入磁盘。
- 当系统重启后,  $G' = G + \Delta T$

# 同步时钟的应用

---

- 基于时钟的缓存一致性
  1. 当客户读取一个副本到缓存时，设置一个租期（lease）
  2. 在租期过期之前，客户可更新副本，重续租期
  3. 如果已经过期，缓存中的副本失效

## 改进的一致性协议

- 当客户修改文件时，只需将所有没有到期的缓存副本设为无效
- 如果某个客户崩溃，则等待直到该客户的租期过期

# 第6章 同步化

---

- 6.1 物理时钟同步
- 6.2 逻辑时钟同步
- 6.3 选举算法
- 6.4 互斥控制
- 6.5 分布式系统中的死锁



## 6.2 逻辑时钟同步

---

- 计时器：石英晶体+计数器
- 时钟偏差 (clock skew)
- 物理时钟：真实时间
- 逻辑时钟：相对时间
- “之前”关系： $\rightarrow$ 
  - 事件a在b之前出现，则 $a \rightarrow b$
  - a为发送消息m，b为接收m，则 $a \rightarrow b$
  - 具有传递性： $a \rightarrow b$ ， $b \rightarrow c$ ，则 $a \rightarrow c$
- 并发事件 (concurrent)
  - 两个事件相互对立。既不 $a \rightarrow b$ ，不  $b \rightarrow a$

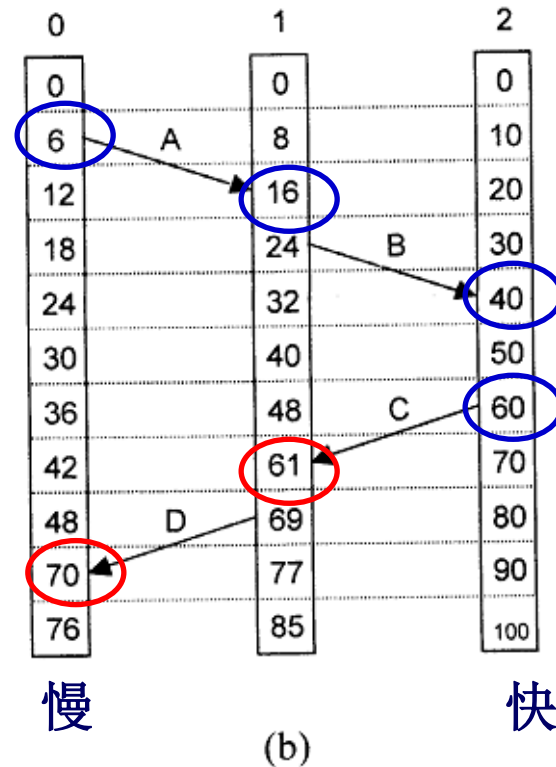
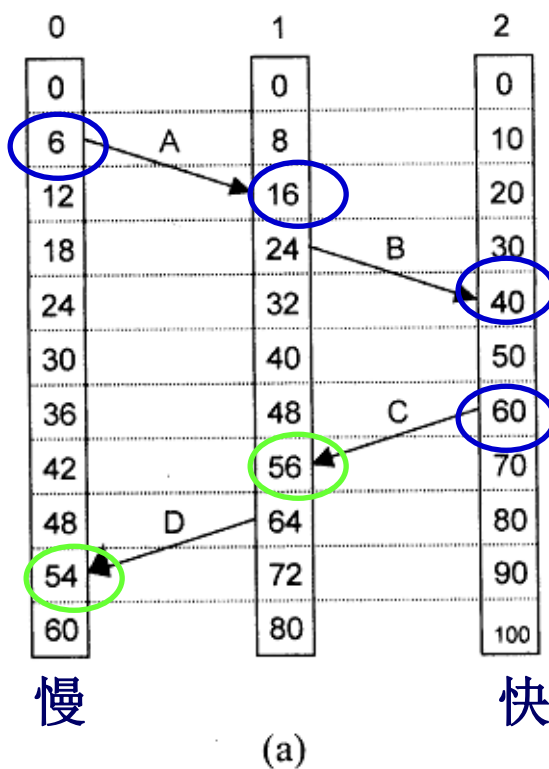
# Lamport算法

---

- $C(a)$  表示事件 $a$ 的时钟值。性质：
  - if  $a \rightarrow b$ ; 则  $C(a) < C(b)$
  - $\forall a, b \ C(a) \neq C(b)$
  - $C$ 是递增的
- 校正算法
  - $a \rightarrow b$ ,
  - if  $C(b) < C(a)$ , 则  $C(b) = C(a) + 1$

# Lamport算法

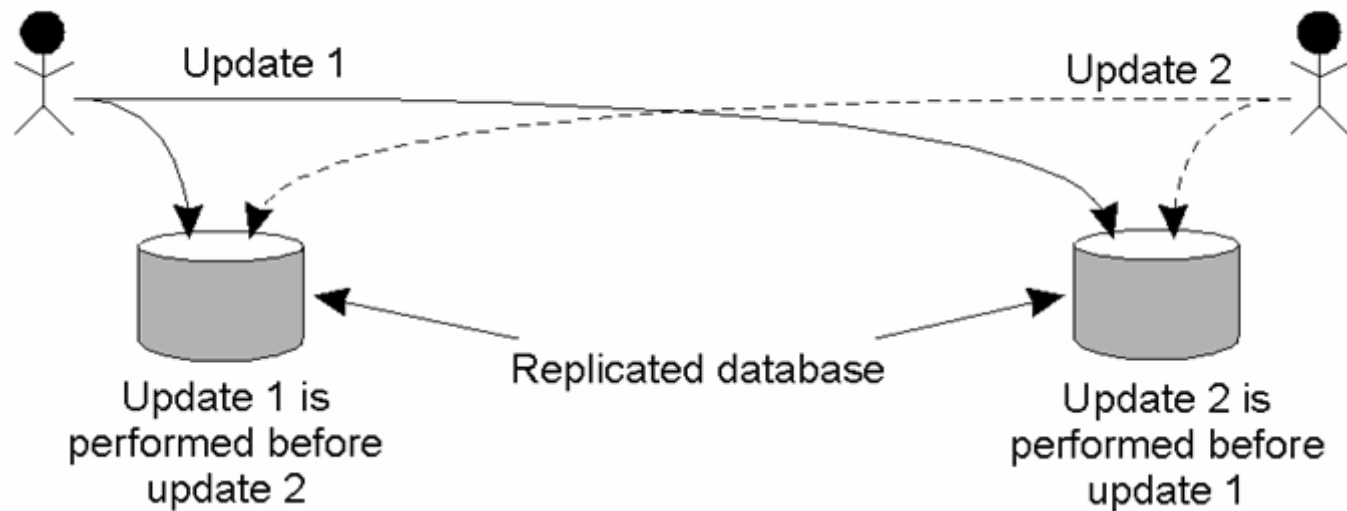
- a) 三个进程，各有自己的局部时钟，它们速率不同
- b) 通过Lamport算法，校正时钟



时间

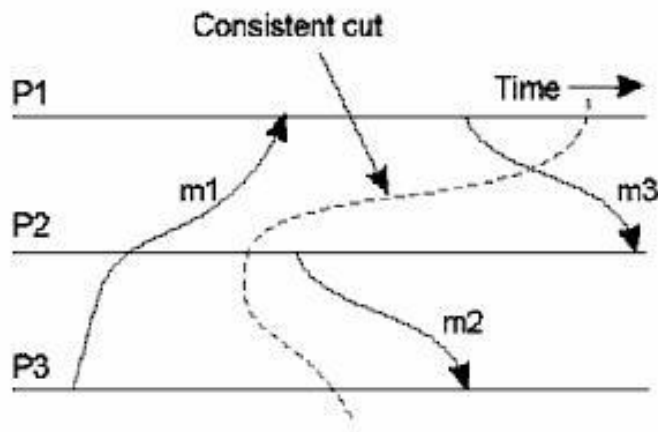
# 全序多播 (1)

- 问题：两个进程分别对同一个复制数据库进行更新时，造成不一致状态
- 原因：全局次序不一致。 $u1 \rightarrow u2$ ;  $u2 \rightarrow u1$

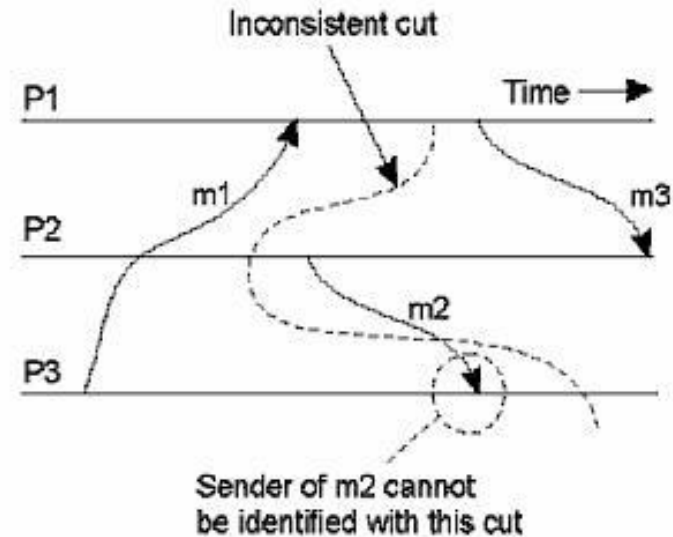


# 全局状态 (2)

- 分布式快照 (snapshot)
  - 反映分布式系统的一致性全局状态
- 割集：全局部状态的图形表示



(a) 一致性割集



(b) 不一致性割集

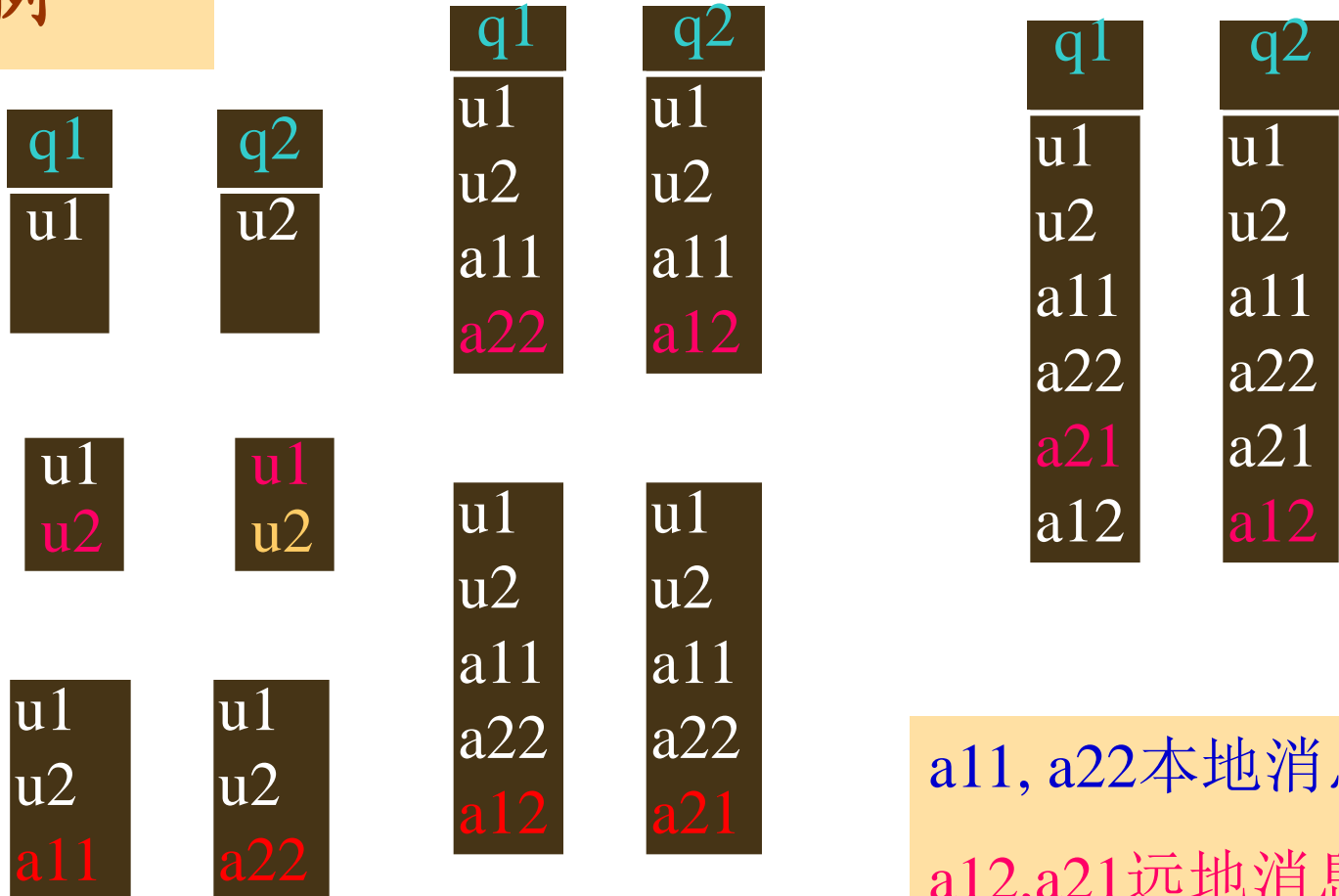
# 全序多播 (2)

---

- 解决方案：全序多播
  - 发送进程多播发送消息 $m$ 时，给 $m$ 带上当前时间戳 $ts$
  - 当接收进程收到消息 $m$ 后，存放其局部队列 $q$ ，按时间戳排序
  - 接收进程向所有进程多播发送应答
  - 当消息 $m$ 被所有进程应答，且排在队列 $q$ 队首后，方可处理
- 定理：各个进程的局部队列的值最终都相同

# 全序多播 (3)

- 举例



$a11, a22$ 本地消息

$a12, a21$ 远地消息

# 时间戳向量(1)

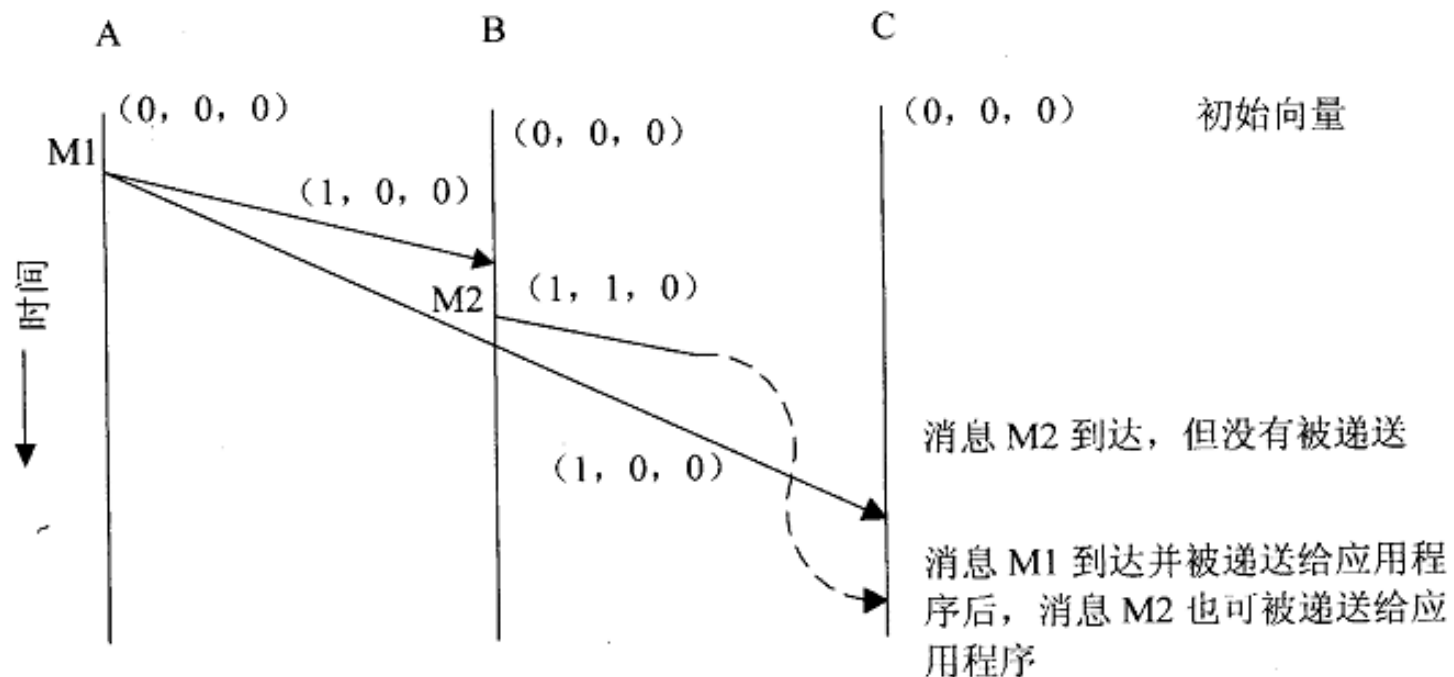
---

- 因果性 (causality):
  - 如果事件 $a$ ,  $b$ 存在因果关系。 $a$ 为因,  $b$ 为果, 则  $C(a) < C(b)$
- Vector timestamp
  - 如果  $VT(a) < VT(b)$ , 则 $a$ 与 $b$ 为因果关系
  - $V_i[i] = n$ , 在 $P_i$ 中发生了 $n$ 个事件
  - $V_i[j] = k$ , 在 $P_j$ 中发生了 $k$ 个事件
- 修改规则
  - 当进程 $P_i$ 发送消息 $m$ 时,  $V_i[i] = V_i[i] + 1$ ,  $vt(m) = V_i$
  - 当进程 $P_j$ 收到 $m$ 后,  $V_j[k] = \max\{V_j[k], vt(m)[k]\}$



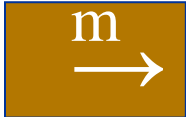
## 时间戳向量 (2)

- $P_i$ 消息 $m$ 在进程 $P_k$ 递交的条件:
  - $vt(m)[i] = V_k[i] + 1$
  - $vt[m][j] \leq V_k[j]$  for all  $i \neq j$



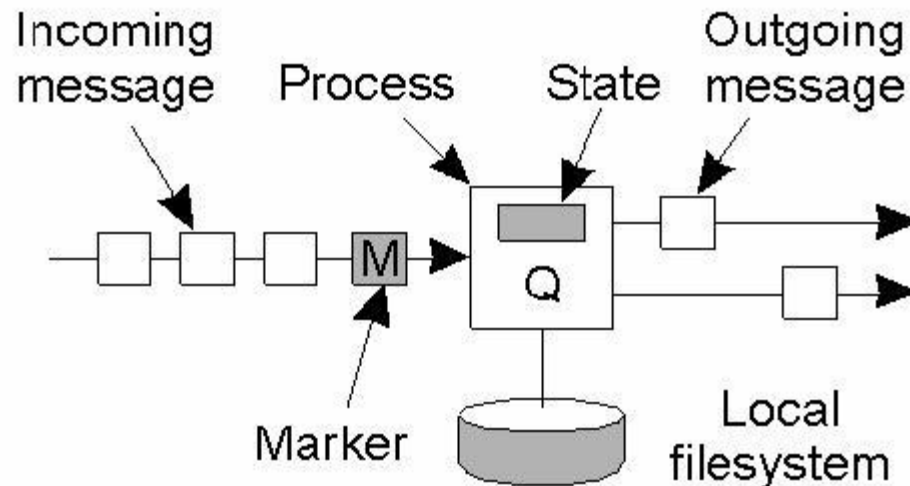
# 全局状态 (1)

---

- 局部状态：
  - 进程正在处理的变量值、对象状态等
  - 例：分布库系统中的数据库记录
- 全局状态：
  - 由每个进程的局部状态和正在传递的消息组成
- 一致性状态：
  - 符合逻辑关系——时钟次序
  - 例：P  Q, 如有Q的接收状态, 必有P的发送状态

# 全局状态 (3)

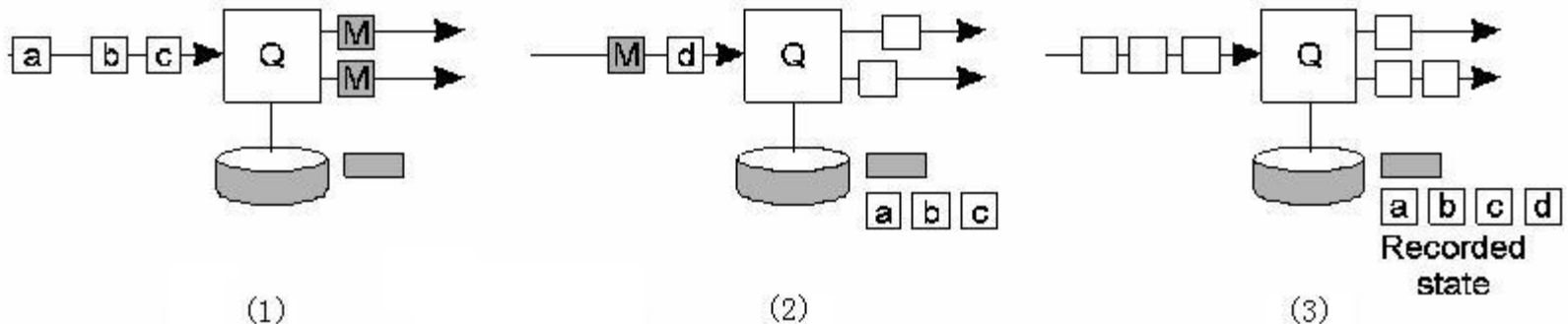
- 分布式割集的实现
  - 通信通道结构：单向的点对点通信
    - 输出通道：接收消息
    - 输入通道：发出消息
  - 标记(Marker)：指示收集状态



# 全局状态（4）

- 算法：

- 发起者进程P向所有输出通道发送marker消息
- 进程 Q第一次收到marker，记录其局部状态（图1）
- 进程 Q记录所有的到来消息（图2）
- 进程Q 再次收到marker后，完成对该输入通道的状态记录（图3）
- 进程Q记录完所有进入通道的状态后，向发起者返回局部状态和通道状态



# 全局状态 (5)

---

- 举例：终止检测
  - 检查一个分布式计算是否结束
- 结束条件
  - 发起者进程P收到所有后继者返回的DONE消息
- DONE消息：返回给前趋者
  - 进程Q的所有后继者都返回DONE消息
  - 进程Q在记录局部状态和收到marker之间不再收到任何消息
- CONTINUE消息：返回给前趋者
  - 发起者进程P继续发起另一个snapshot。

# 第6章 同步化

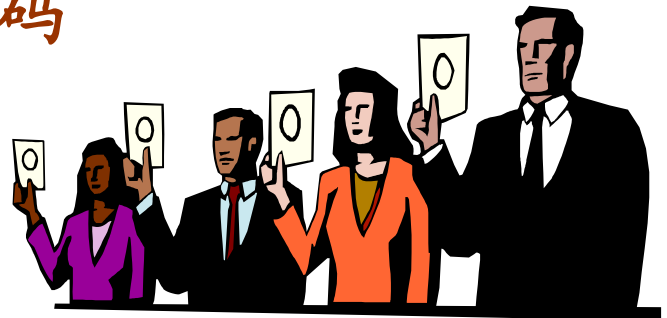
---

- 6.1 物理时钟同步
- 6.2 逻辑时钟同步
- 6.3 选举算法
- 6.4 互斥控制
- 6.5 分布式系统中的死锁

## 6.3 选举算法

---

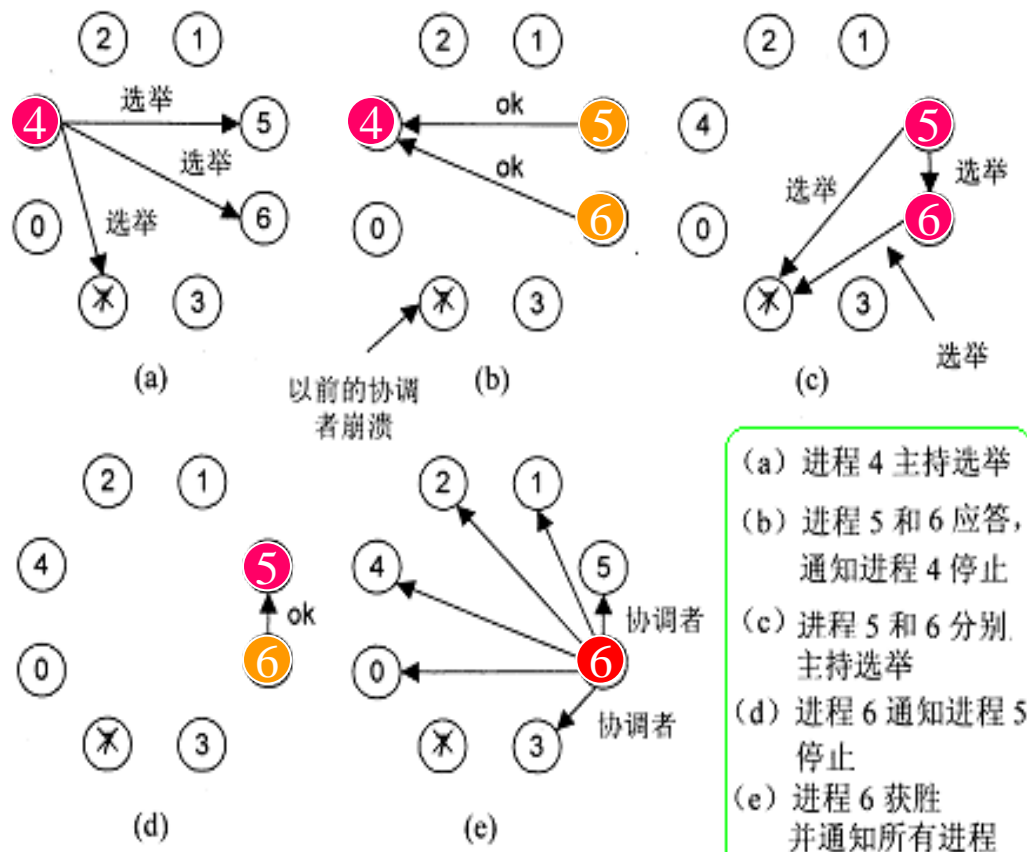
- 作用：
- 在分布式进程之间做出统一的决定
- 例如：确定协调者
- 前提：
- 每个进程具有唯一的号码，如IP地址
- 每个进程知道其它进程的号码
- 选举标准：
- 确定具有最大号码的进程



# 霸道 (Bully) 算法

## ❖ 将进程进行排序

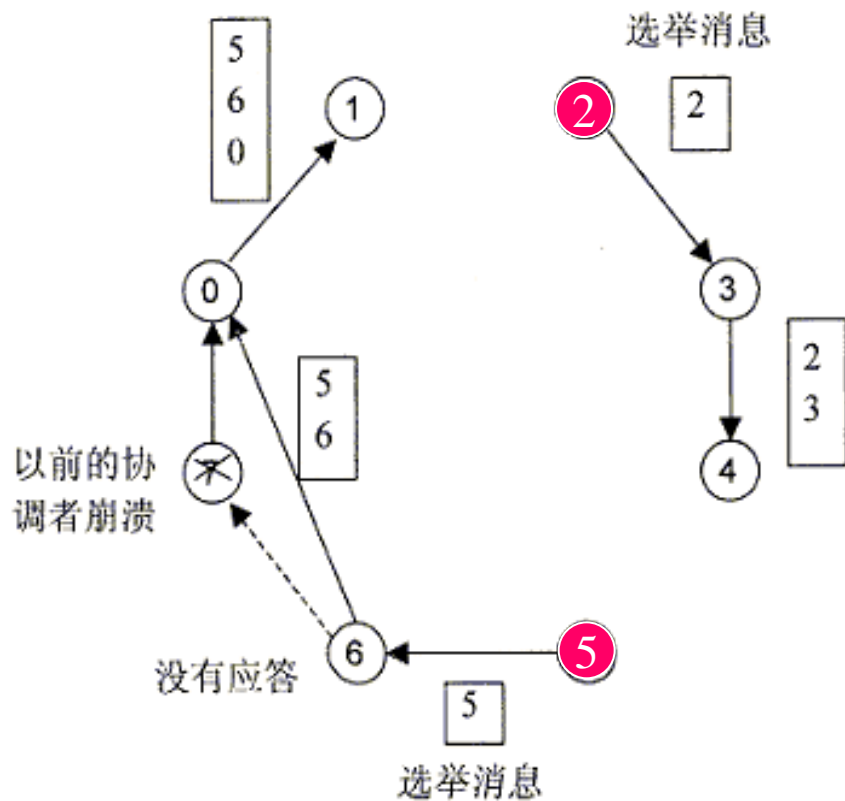
1. P向高的进程发E消息
2. 如果没有响应，P选举获胜
3. 如果有进程Q响应，则P结束，Q接管选举并继续下去。





# 环算法

- 所有进程按逻辑或物理次序排序，形成一个环
1. 当一个进程P发现协调者C失效后，向后续进程发送E消息
  2. 每个进程继续向后传递E消息，直到返回P
  3. P再将新确定的协调者C' 传给所有进程



# 第5章 同步化

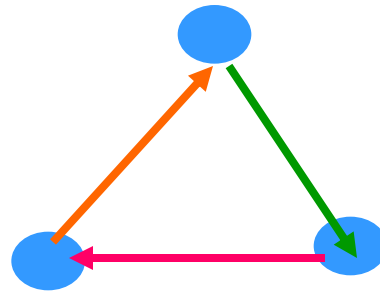
---

- 5.1 物理时钟同步
- 5.2 逻辑时钟同步
- 5.3 选举算法
- 6.4 互斥控制
- 5.5 分布式系统中的死锁

## 6.4 互斥控制

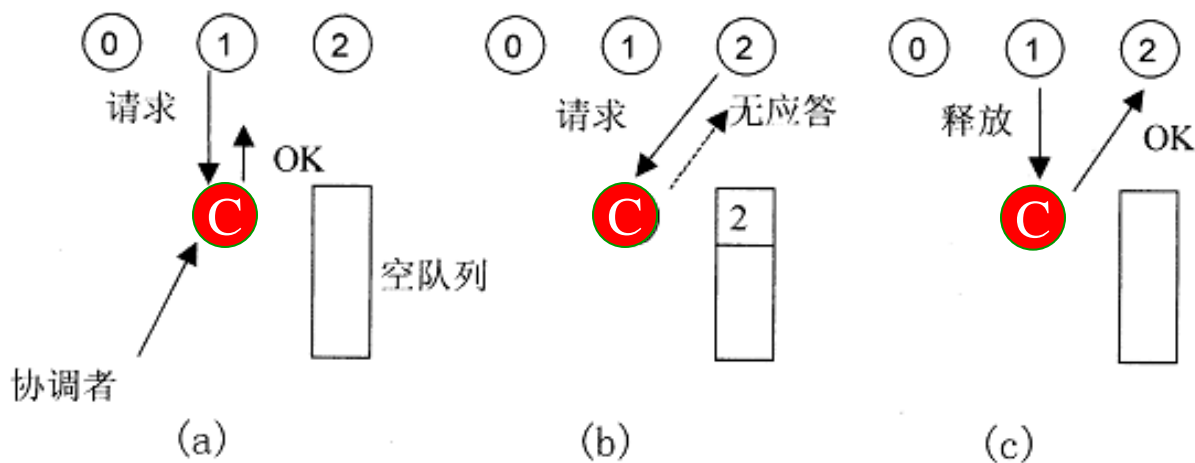
---

- 基本概念
  - 当一个进程使用某个共享资源，其他进程不允许对这个资源操作
- 临界区：
  - 对共享资源进行操作的程序段
- 基本方法：
  - 信号量、管程
- 问题：
  - 死锁
  - 饥饿



# 集中式算法

- 协调者：确定那个进程可进入临界区
- 通信量：3个消息：请求-许可-释放



- 缺点：单点失败

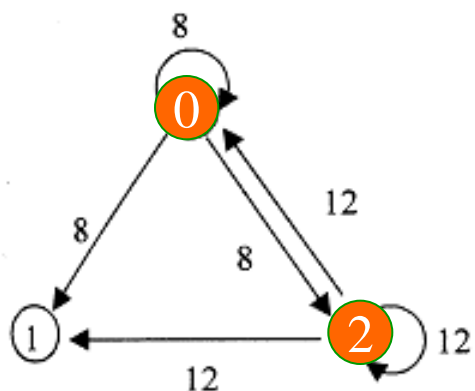
# 分布式算法（Ricart-Agrawala算法）

---

1. 在一个进程P打算进入临界区R之前，向所有其他进程广播消息 **〈临界区R名、进程号、时间戳〉**
2. 当一个进程P' 收到消息后，做如下决定：
  - 若P' 不在临界区R中，也不想进入R，它就向P发送OK；
  - 若P' 已经在临界区R中，则不回答，并将P放入请求队列；
  - 若P' 也同时要进入临界区R，但是还没有进入时，则将发来的消息和它发送给其余进程的时间戳对比。如果P时间戳小，则向P发送OK；否则，不回答，并将P放入请求队列；
3. 当P收到所有的OK消息后，进入R。否则，等待。
4. 当P退出R时，如果存在等待队列，则取出一个请求者，向其发送OK消息。

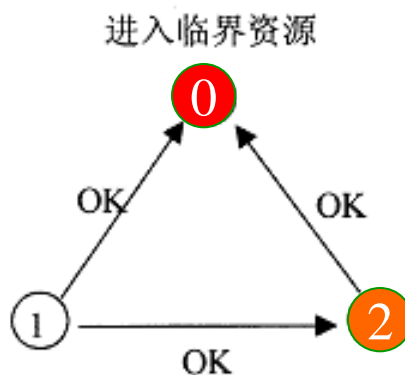
# 分布式算法举例

举例： 共有0, 1, 2三个进程。  
进程0, 2申请进入临界区



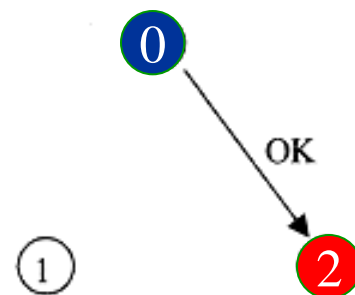
(a)

两个进程在同一时刻  
进入同一个临界区



(b)

进程 0 的时间戳小，获胜



(c)

进入临界资源  
进程 0 完成，它发送消息 OK，  
进程 2 现在可进入临界区

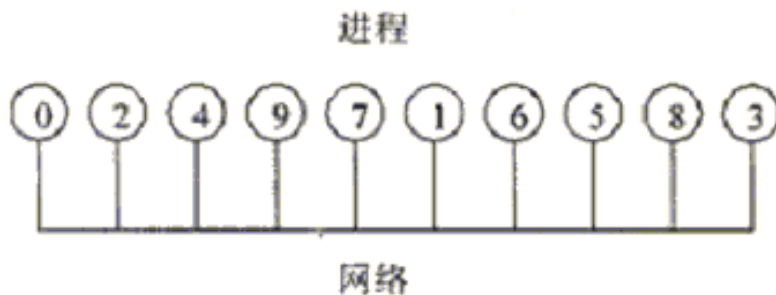
# 分布式算法评价

---

- 缺点：
  - $n$ 点失败
  - $n$ 点瓶颈
  - $2(n-1)$ 个消息
- 改进方案：
  - 超时重发
  - 组通信
  - 简单多数同意 ( $>1/2$ )

# 令牌环算法

- 构造一个逻辑环，得到令牌才可进入临界区
- 问题：令牌丢失检测



(a)

网络中一组未排序的进程



(b) 用软件构造进程的逻辑环



# 三种互斥算法的比较

---

| 算法  | 每次进出需要的消息   | 进入前的延迟<br>(按消息次数) | 存在问题       |
|-----|-------------|-------------------|------------|
| 集中式 | 3           | 2                 | 协调者崩溃      |
| 分布式 | $2(n-1)$    | $2(n-1)$          | 任何一个进程崩溃   |
| 令牌环 | 1到 $\infty$ | 0到 $n-1$          | 丢失令牌, 进程崩溃 |

# 第6章 同步化

---

- 6.1 物理时钟同步
- 6.2 逻辑时钟同步
- 6.3 选举算法
- 6.4 互斥控制
- 6.5 分布式系统中的死锁

## 6.5 分布式系统的死锁处理

---

- 死锁解决策略
  - 鸵鸟法：留给用户考虑
  - 检测法：发现死锁，进行处理
  - 预防法：在设计上使死锁不可能发生
  - 避免法：在运行中，防止出现死锁
    - 银行家算法[Dijkstra, 1965]
    - $P\langle \text{has}, \text{max} \rangle, \text{free}$

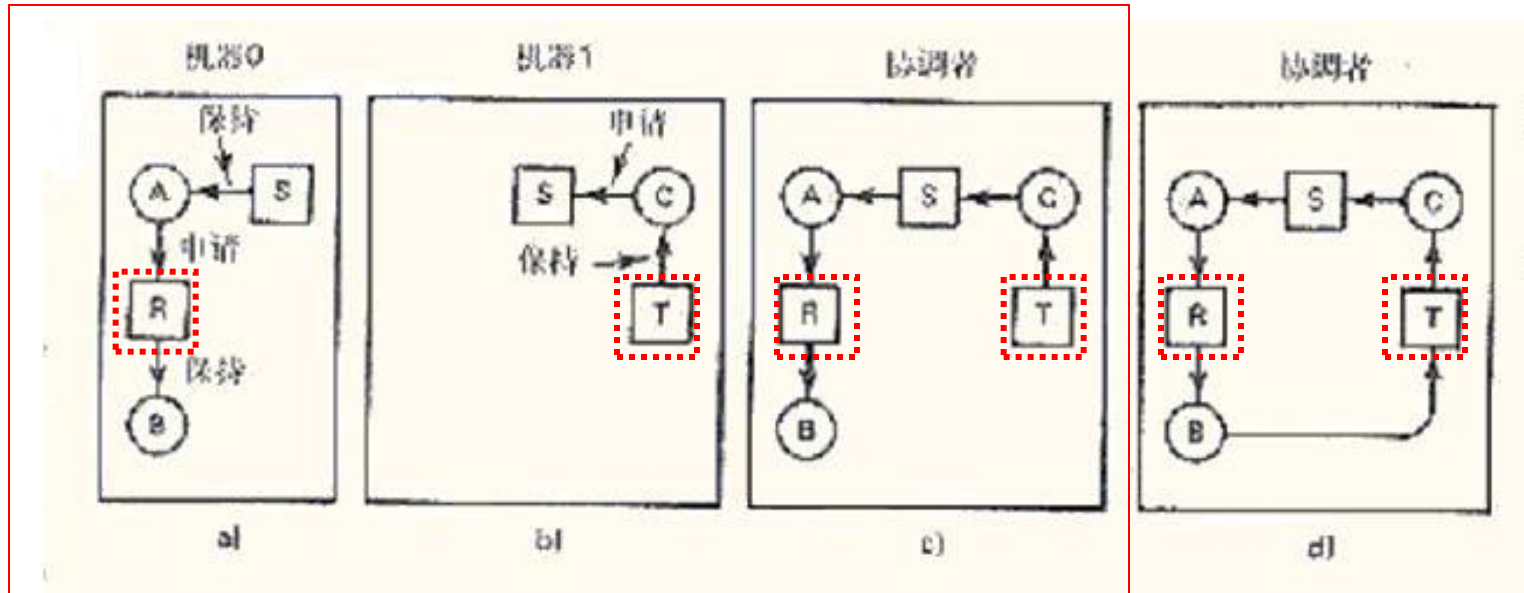
# 集中式检测方法

---

- 进程-资源等待图
  - 节点：进程P、资源R
  - 有向边：(1)  $P \rightarrow R$  请求关系；(2)  $R \rightarrow P$  拥有关系；
- 死锁检测协调者
  - 负责检测死锁
- 资源图的维护策略：
  - 当资源图中，有一条边加入/删除时，通知协调者
  - 每个进程周期性地向协调者发送图的更新消息
  - 协调者在需要时，向参入者请求

# 集中式检测方法举例

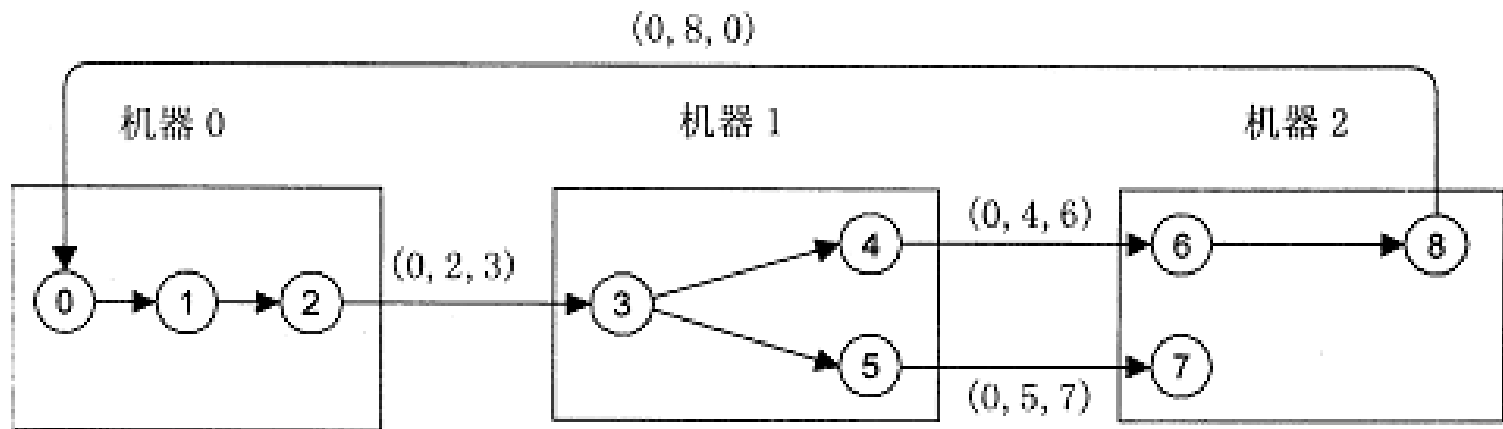
- 假死锁问题：B释放R，请求T。若请求T消息先到达协调者



- 解决方案一：协调者确认（消息的全局时序）

# 分布式检测方法

- Chandy—Misra—Haas分布式死锁检测算法,
- 探测消息:  $\langle \text{阻塞Pid}, \text{请求Pid}, \text{接收Pid} \rangle$
- e.g.  $(0, 2, 3)$ ,  $(0, 4, 6)$ ,  $(0, 5, 7)$ ,  $(0, 8, 0)$  构成死锁



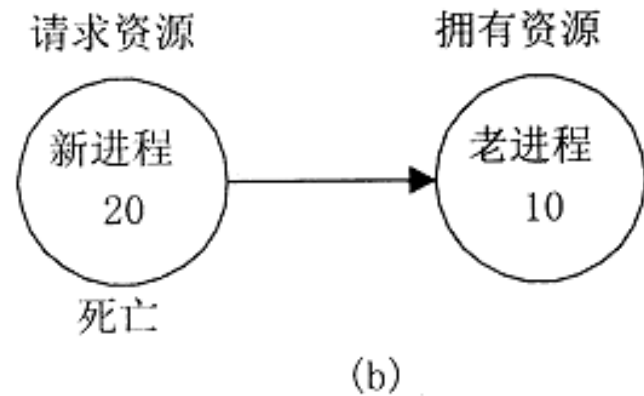
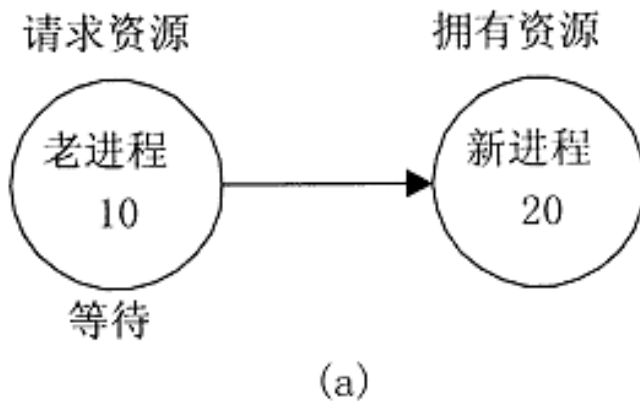
# 分布式深度限制算法 (DWDL)

---

- 90%的死锁发生在两个进程之间
- 算法:
  - // p1为请求者; L(p1)为p1的寿命
  - 1) if ( waitQueue = p2->p1->p0 ) then
    - if ( L(p1)<L(p2) or L(p1))<L(p0) then restart p1;
    - else restart p0;
  - 2) if (waitQueue = p1->p1'->p0 ) then
    - if ( L(p'1)<L(p1) or L(p'1)<L(p0)) then restart p'1;
    - else restart p0;
  - 3) if (waitQueue = p2->p1->p'1->p0 ) then
    - if ( L(p1)<L(p2) or L(p1)<L(p0)) then restart p1;
    - else restart p'1;

# 分布式死锁预防

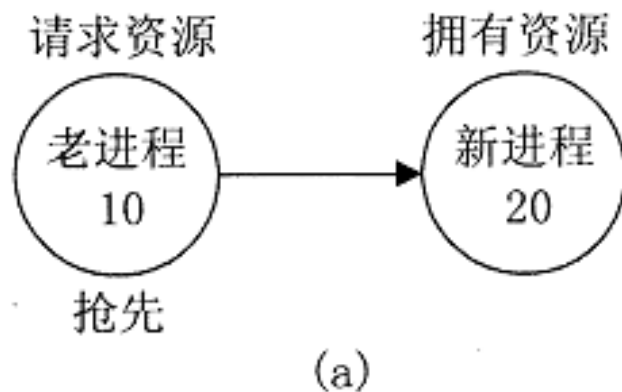
- 等待-死亡算法(wait-die)
  - 设请求进程0的时间印 $t_0$ ，拥有资源的进程1的时间印 $t_1$
  - 如果 $t_0 < t_1$ ，0等待；
  - 否则，撤销0





# 分布式死锁的预防

- 负伤-等待算法 (wound-wait)
  - 设请求进程0的时间印 $t_0$ , 拥有资源的进程1的时间印 $t_1$
  - 如果 $t_0 < t_1$ , 撤销1;
  - 否则, 0等待



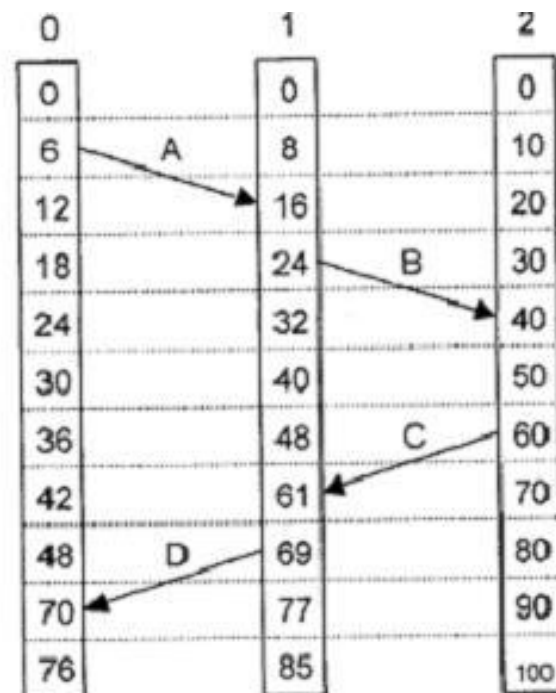
# 小结

---

- 物理时钟的同步算法
- 基于逻辑时钟的同步算法
- 一致性全局状态的检测
- 选举算法
- 互斥算法
- 分布式系统的死锁处理

# 习 题 (1)

1. 在右图中加入一条与A并发的新消息。它既不在A之前，也不在A之后。
2. 假定有两台机器A和B， 它们的时钟都是每毫秒滴答1000次。但实际上B每毫秒滴答900次。如果每分钟根据UTC时间校正一次时钟，那么A和B之间的最大时钟偏差是多少？



## 习 题 (2)

---

3. 假定A和B是相互独立的两个临界区，进程0要进入A，进程1要进入B，R-A分布式互斥算法会导致死锁吗？说明理由。
4. 对所介绍的bully选举算法，进行优化。
5. 事务时间戳为50的进程申请事务时间戳为100的进程占用的资源。按以下两种策略，结果会如何？(1) 等待-死亡； (2) 负伤-等待。