分布式算法

配置、事件（comp, del）
调度是事件的序列。
安全性 、 活跃性条件
容许执行：满足活跃性条件。
同步模型：**时间由轮来度量**

广播
在同步模型中，在广播算法的每个容许执行里，树中每个距离 pr 为 t 的处理器在第 t 轮里接收消息 M
Th2.2 当**生成树高度为 d** 时，存在一个消息复杂度为 n-1，时间复杂度为 d 的同步广播算法。

异步算法：
引理 2.3 在异步模型的广播算法的每个容许执行里，树中每个距离 pr 为 t 的处理器至多在时刻 t 接收消息 M。
Th2.5 当**生成树高为 d 时**，存在一个异步的敛播方法，其 msg 复杂性为 n-1，时间复杂度为 d。
广播和敛播算法用途：初始化某一信息请求(广播发布)，然后用敛播响应信息至根。

容许的执行：无限的执行
复杂性度量：
时间复杂度
①同步系统：最大轮数，即算法的任何容许执行直到终止的最大轮数。
②异步系统：假设：①节点计算任何有限数目事件的时间为 0；②一条消息发送和接收之间的时间至多为 1 个时间单位，定义为：所有计时容许执行中直到终止的最大时间（与最大时刻的距离）。
Flooding 算法构造生成树（在异步算法中）：
一收到消息 M 就转发给除 pj 以外的所有邻居。【从 pr 开始，发送 M 到其所有邻居。当 pi 第 1 次收到消息 M（不妨设此 msg 来自于邻居 pj）时，pi 发送 M 到除 pj 外的所有邻居。】
msg 复杂性：
2m-(n-1)个 msgs。m 是系统中信道总数，它可能多达 n(n-1)/2。
时间复杂性：O(D)　　D—网直径

Alg2.2 构造生成树
相互确认。发送一个 msg,pi 收到来自 pj，当收到第一个 msg 时，回发 parent，达成子结点配对，其他均发 reject。
证明：
为何从根能到达每一结点？(连通)
反证：假设某结点在 G 中从 pr 不可达，因网络是连通的，若存在两个相邻的结点 pi 和 pj 使得 pj 在 G 中是从 pr 可达的(以下简称 pj 可达)，但 pi 不可达。因为 G 里一结点从 pr 可达当且仅当它曾设置过自己的 parent 变量(因为收到过 M 返回后 parent 后才会加入到图 G 中，容许执行，被赋过值)，所以 pi 的 parent 变量在整个执行中仍为 nil，而 pj 在某点上已设置过自己的 parent 变量，于是 pj 发送 M 到 pi(line9)，因该执行是容许的，此 msg 必定最终被

pi 接收，使 pi 将自己的 parent 变量设置为 j。矛盾！

Th2.7 对于**具有 m 条边和直径 D 的**网络，给定一特殊结点，存在一**个 msg 复杂性为 O(m)，时间复杂性为 O(D)**的**异步算法**找到该网络的一棵生成树。

Alg2.2 在**同步模型**下仍可工作。其分析类似于异步情形。然而，与异步不同的是，它所构造的生成树一定是一棵**广度优先搜索(BFS)树**。

Lemma2.8 在同步模型下，Alg2.2 的每次容许执行均构造一棵根为 pr 的广度优先搜索 BFS 树。

Th2.9 对于具有 m 条边直径为 D 的网络，给定一个特殊结点，存在一个**同步算法**在 msg 复杂性为 O(m)，时间复杂性为 O(D)内找到一棵 BFS 树。

信息的请求和收集

将算法 2.2(求生成树)和汇集算法**组合**即可完成。组合算法的时间复杂性在同步和异步模型中不同,设网是完全图

求生成树算法：同步和异步均为：消息复杂性 O(m)、时间复杂性 O(D)D 是直径，m 是边数

汇集算法：同步和异步均有：msg n-1、 time d //生成树高

组合算法

①同步：组合算法的 msg 复杂性 O(m+n)；BFS 树中，d=1， d≤D，故时间复杂性 O(D+d)=**O(D)**=O(1)。

②异步：组合算法的 msg 复杂性 O(m+n)；生成树高 d=n-1，所以时间复杂性 O(D+d)=**O(d)=**O(n)。

1-time 复杂性的组合算法 T(n)=O(D)。

构造指定根 DFS 生成树：

【依次转发】向未发送 m 的所有邻接任选一个发送 msg,当某节点收到一个 msg 后，反 parent,并发 reject 给后面再发来 m 的邻居。再任选一个邻居发送 m，等待加入的 child。直到给所有的邻居都转发过 m,pi 终止。

构造 DFS 树时每次加一个结点，而不像 Alg2.2 那样，试图在树上同时增加同一层的所有结点。

证明 DFS 生成树：

1. 连通性：假设从 pr 不可达，但是设置过 parent 变量，即已经发送过 M，故矛盾。
2. 无环性：假设存在环，P1 的 parent 是 Pi，但根据代码 P1 会向 Pi 发送一个 reject 而 Pi 未收到 P1 返回的<parent>信息，也不会将 P1 设为 child,矛盾。
3. 子结点先于兄弟结点加入树中： P1，P2 和 P3。P2 和 P3 是 P1 的直接相邻结点。P1 先向 P2 发送 M，P1 当且仅当 P2 向其返回一个<parent>才可能向 P3 发送 M。P2 仅在其向**所有的相邻结点发送过 M 后才会向 P1 返回<parent>**。所以 P2 的子结点是永远先于 P3 加入树中。

Th2.11 对于一个具有 m 条边，n 个结点的网络，以及给定的特殊顶点，存在一个时间复杂性和消息复杂性均为 O(m)的异步算法找到一棵 DFS 树。

证明 Alg2.3 构造 DFS 生成树（指定根） 的时间复杂性为 O(m)。

同步模型：每一轮中，根据算法，有且只有一个消息(M or Parent or Reject)在传输。

消息**只发往一个处理器结点**，除根结点外，所有的处理器都是**收到消息后才被激活**，所以，

不存在多个处理器在同一轮发送消息的情况，所以时间复杂度与消息复杂度一致。
异步模型类似。
算法 2.2 和 2.3 构造连通网络的生成树时，需存在一个特殊的结点作为启动者(Leader)
不存在 leader 时，使用不指定根构造生成树：
不指定根构造生成树：
不指定根构造 DFS 生成树，和后面的领导者选举问题**类似**，都是破对称问题。
算法 2.4 不指定根构造生成树
每个结点均可自发唤醒，试图构造一棵以自己为根的 DFS 生成树。若两棵 DFS 树试图链接同一节点(未必同时)时，该节点将加入根的 id 较大的 DFS 树。初值为自己的 id,遇到比自己大的 leader 赋值为新的 id。
当结点自发唤醒时，将自己的 id 发送给某一邻居。收到来自邻居的 id 时，和自己当前存储的 leader 比较，遇到更大的就更新。
定理：对于一个具有 m 条边和 n 个节点的网络，自发启动的节点共有 p 个，其中 ID 值最大者的启动时间为 t，则算法的消息复杂度为 O(pn^2)，时间复杂度为 O(t+m)。

引理 2.12 **设 Pm 是所有自发唤醒结点中具有最大标识符的结点**。在异步模型的每次容许执行里，算法 2.4 构造根为 Pm 的一棵 DFS 树。
引理 2.13 在**异步模型的每个容许执行里，只有一个处理器终止作为一生成树的根**。

消息复杂性：最坏情况下，每个处理器均试图以自己为根构造一棵 DFS 树。因此，Alg2.4 的 msg 复杂性至多是 Alg2.3 的 n 倍：**O(m*n)**
时间复杂性：类似于 Alg2.3 的 msg **复杂性 O(m)。**

复杂度总结：
指定根 DFS 生成树
时间复杂性和消息复杂性均为 O(m)。
不指定根构造 DFS 树：
消息复杂性：自发启动的节点共有 p 个，其中 ID 值最大者的启动时间为 t，则算法的消息复杂度为 **$O(pn^2)$**，Alg2.4 的 msg 复杂性至多是 Alg2.3 的 n 倍：**O(m*n)**
时间复杂性： **O(t+m)**，类似于 Alg2.3 的 msg **复杂性 O(m)。**

环选举
选出一个特殊结点作为 leader.（简化处理器协作，容错、节省资源）易于实现广播算法
一类破对称问题：打破死锁
对于环系统，不存在匿名的选举算法。
证明：非均匀算法的不可能性 同步算法的不可能性 从而推出均匀算法、异步算法的不可能，综合起来即为：不存在匿名的选举算法。
匿名环保持对称 同步系统 所有处理器始于相同状态，每一轮里 执行同样的程序、发送、接收同样的 msg 改变状态相同。 一个选中，所有选中，矛盾！
非均匀：
假设 R 是大小为 n>1 的环（非均匀），A 是其上的一个匿名算法，它选中某处理器为 leader。因为环是同步的且只有一种初始配置，故在 R 上 A 只有唯一的合法执行。
引理 3.1 在环 R 上匿名算法 A 的容许执行里，对于每一轮 k，所有处理器的状态在第 k 轮

结束时是相同的。

Th3.2 对于同步环上的 leader 选举，不存在非均匀的匿名算法。

异步环【选最大的】:

异步环上 **leader 选举问题的 msg 复杂度下界为 Ω(nlgn)** ，其算法的关键是 msg 延迟是任意长的。

假定具有唯一标识符。 从最小标识符开始指定环。

均匀算法：每个标识符 id，**均有一个唯一的状态机，但与环大小 n 无关**。而在匿名算法中，均匀则指所有处理器只有同一个状态。

非均匀算法：**每个 n 和每个 id 均对应一个状态机**，而在匿名非均匀算法中，每个 n 值对应一个状态机。

一个 O(n2)算法：每个处理器 Pi 都向左邻居发送含有自己标识符的 msg，再等待其右邻居的 msg,收到比自己大的则转发给左邻居，否则没收。即一个容许执行发送 O(n2)个 msgs。

$n+\sum_{i=0}^{n-1}(i+1)=\theta(n^2)$ //前一项为终止msg

O(nlgn)算法：按阶段执行，在第l阶段一个处理器试图成为其 2^l-邻接的临时 leader，当选 leader 后才可以进入下一阶段。每次都是向左右 2^l-邻居发送 msg，若两边均返回了 reply 则当选。

在 phase l 里：

一个处理器启动的 msg 数目至多为：$4*2^l$ 发过去 msg 再返回 reply 两边 2^l。

有多少个处理器是启动者？ l=0 时有 n 个， l≥1，临时 leader 为启动者。

引理 3.3 对每个 k≥1，在 phase k 结束时，临时 leader 数至多为 n/(2k+1).

Th3.4. **存在一个异步的 leader 选举算法，其 msg 复杂性为 O(nlgn).**

$$4n+\sum_{l=1}^{\lg(n-1)}(4\cdot 2^l)\frac{n}{2^{l-1}+1}+n\le 8n\lg n$$

phase0 msg 为 4n，终止 msg 为 n。

开调度：

σ 是算法 A 的一个调度，若该环上存在边 e，其任意方向上均无 msg 传递，则 σ 称为是 open。开调度未必是容许的调度（无限执行），即它可能是有限的事件序列。

Th3.5 对于每个 n 及每个标识符集合(大小为 n)，存在一个由这些标识符组成的环，该环有一个 A 的开调度，其中至少接收 M(n)个消息，这里：

$$\begin{cases} M(2)=1 & n=2 \\ M(n)=2M(\frac{n}{2})+\frac{1}{2}(\frac{n}{2}-1) & n>2 \end{cases}$$

以下内容用于证明 Th3.5.

引理 3.6 对每个由两个标识符构成的集合，存在一个使用这两个标识符的环 R，R 有 A 的一个开调度，其中至少有一个 msg 被接受。（归纳基础）

处理器状态是"静止"的：若从该状态开始的计算事件序列中**不 send 消息**，即处理器接收一个 msg 之前不发送另一 msg（即处理器的**内部事件不引发 send 动作**）

配置是"静止"的(关于 ep 和 eq)：若**除开边 ep 和 eq 外，没有 msgs 处在传递之中**，每个处理器均为静止状态。

断言 3.8 存在一个有限的调度片断 σ4，其中有 (n/2-1)/2 个 msgs 被接收，σ1σ2σ3σ4 是一个开调度，其中 ep 或 eq 是开的。

总结： Th3.5 的证明可分为 3 步：

1) 在 R1 和 R2 上构造 2 个独立的调度，每个接收 2M(n/2)个 msg: σ1 σ2
2) 强迫环进入一个静止配置： σ1 σ2 σ3 （主要由调度片断 σ3 ）
3) 强迫(n/2-1)/2 个附加 msg 被接收，并保持 ep 或 eq 是开的: σ1 σ2 σ3 σ4。

因此我们已构造了一个开调度，其中至少有 2M(n/2)+ (n/2-1)/2 个 msg 被接收。

异步环复杂性：

消息复杂性：O(n2)→O (nlogn) →Ω(nlogn)（下界）

同步环【选 id 最小的】:

选举问题上下界

上界：两个 msg 复杂性为 O(n)的算法（msg 复杂性最优，运行时间与 id 值相关）

下界：

1）只讨论基于标识符之间比较的算法

2）时间受限（即若干轮内终止，轮数只依赖于环大小,不依赖于 id 值）的算法

当算法受限于上述两个条件时，至少需要 Ω(nlgn)个 msgs。

同步环上

1）msg 延迟是确定的 2) 获取 info 不仅来自于接收 msg,在某轮里的内附件也能获取 info

本节提出两个算法，msg 复杂性上界为 O(n),针对单向环，但是用于双向环

1) 非均匀的：要求环中**所有的结点开始于同一轮**，标准的同步模型 **(需知道 n)**

2) 均匀的： **结点可开始于不同轮**，弱同步模型 (无需知道 n)

同步环消息复杂度上界 O（n）

非均匀的（若干个阶段 phase,每一个阶段有 n 轮）:

n 已知，**选择环中最小 id 为 leader**，按 phase 运行，每个阶段由 n 轮组成。在 Phase i ($i \geq 0$)，若存在一个 id 为 i 的结点，则该结点为 leader，并终止算法，因此，最小 id 的结点被选为 leader,phase 数目取决于 n 个节点的标志符的取值。

复杂性：

Msg 复杂性：恰有 n 个 msg 被发送，故复杂性为 O(n)。注意这 n 个 msg 均是在找到 leader 的那个 Phase 里发送的。

时间复杂性：依赖于环大小和环上最小标志符，不妨设环大小为 n，最小标识符为 i，则**算法执行轮数为：n·(i+1)**，不妨设 i≥0 //运行时间与环大小及标识符取值相关

id 为 i 的结点要在 phase i 发？ 结点互相不知道彼此 id 的值，在 id=i 的时候发送自己的 id

每个阶段（phase）需要 n 轮。

均匀算法（需增加唤醒阶段）:

1.无需知道环大小，2.弱同步模型，**结点可开始于不同轮**

处理器可在任意轮里自发唤醒自己，也可以收到另一个处理器的 msg 后被唤醒。

算法思想：

1.源于不同节点的 msg 以**不同速度转发**

源于 **id 为 i 的节点的 msg**，在每一个接收该 msg 的节点沿顺时针转发到下一个处理器之前，**被延迟 2^i-1 轮**。

2.为克服非同时启动，须增加一个基本的唤醒阶段，其中每个自发唤醒的结点绕环发送一个

唤醒 msg，该 msg 转发时无延迟。

3.若一个结点在算法启动前收到一个唤醒 msg，则**该结点不参与算法**，只是扮演一个 relay(转发)角色：即转发或没收 msg。

在基本阶段之后，选举 leader 是在参与结点集中进行的，即只有自发唤醒的结点才有可能当选为 leader。

实现：

1.唤醒 接收到唤醒 msg 之前未启动的结点不参与；

2.延迟

Note： 一个 msg 到达一个醒着的节点之后，它要到达的所有节点均是醒着的。**一个 msg 在被一个醒着的节点接收之前是处在 1st 阶段 (唤醒 msg，非延迟)** ，在到达一个醒着的节点之后，它就处于 2nd 阶段，并以 2^i 速率转发(非唤醒 msg，延迟)

参与结点：没收大于当前已看到的最小 id（包括自己）的 msg

转发结点：接收到的 id 大于当前已看到（except self）则没收

Alg3.2 同步 leader 选举

引理 3.9 在参与的节点中，只有最小 id 的节点才能收回自己的 id。恰有一个结点收回自己的 msg，故它是唯一声明自己是 leader 的结点，即算法正确。

算法容许执行中，msg 分为三个类型：

第一类：第一阶段的 msg(唤醒 msg)

第二类：最终 leader 的 msg 进入自己的第二阶段之前(其它结点发出的)发送的第二阶段 msg

第三类：最终 leader 的 msg 进入自己的第二阶段之后(包括 leader 自己发出的)发送的第二阶段 msg。

引理 3.10 第一类 msg 总数至多为 n：每个节点在第一阶段至多转发一个 msg。

引理 3.11 设 pi 是最早开始执行算法的结点中的某一个，其启动轮数为 r，若 pj 距离 pi 为 k(顺时针)，则 pj 接收的第一阶段的 msg 不迟于第 r+k 轮。**3.12 第二类 msg 总数至多为 n.**

$$\sum_{i=1}^{n-1} \frac{n}{2^i} \le n$$

第三类 msg：leader 进入自己的第二阶段之后，所有非唤醒 msg

引理 3.13 在<idi>返回 pi 之后，没有 msg 被转发。

引理 3.14 第三类 msg 总数至多为 2n。

$$\sum_{j=0}^{n-1} \frac{n}{2^{id_j - id_i}} \le \sum_{k=0}^{n-1} \frac{n}{2^k} \le 2n$$

Th3.15 存在一个同步的 leader 选举算法，其 msg 复杂度至多为 4n，即为 O(n)。

由引理 3.13 知，当 leader 接收到自己的 id 时，计算终止。这发生在第一个启动算法的节点之后的 **$O(n \cdot 2^i)$轮**，其中 i 是 leader 的标识符。当 i=0 时，为 O(n)轮。

以上算法缺陷：

1. 在每个容许的执行中，执行轮数依赖于 id，而 id 相对于 n 而言可能是巨大的。**(主要的)**
2. 它们用一种非同寻常的方式使用 id，即 id 决定 msg 延迟多长。

为何非唤醒 msg 要延迟 2^i -1 轮？

降低消息复杂度（id 最小的节点被选举为 Leader，Leader 节点消息的转发速度最快）。

如何修改算法 3.2 来改善时间复杂性？

方案 1：添加 relay 变量，保证消息在转发节点不延迟，时间复杂度由 O(n*2^i)降为 O(N*2^i+n-N)，n 为总节点数，N 自发唤醒的节点数。
方案 2：原算法延迟函数为 $f(id)=2^{id}$ 时间复杂度为 $O(n*2^i)$ 若改为 f(id)=c*id 等，但是会带来消息复杂度的提高。

同步环有限制算法的下界 Ω(nlgn)(限制轮数上界)

1.这些性质对于基于消息的有效算法而言是固有的；
2.若一个算法中的标识符仅用于比较，则它需要 Ω(nlgn)个 msgs；
3.若一个算法中，限制轮数的上界，但独立于 id，则它的 msg 复杂度亦为 Ω(nlgn)。

同步不能导出异步，但异步可以导出同步（异步的特殊情形）：

同步的下界**不可能从**异步的下界导出，因为**同步模型需要附加假定**。同步的下界对于非均匀和均匀算法均成立，但异步的下界只对均匀算法成立。但是从同步导出的异步结果是正确的，并且提供了一个非均匀算法的异步下界。

异步通信模型中领导者选举问题所需的消息数下界为 Ω(nlgn)且算法不依赖于比较的或者限时的。

为计算下界，需要假定同一轮开始执行，环是由结点表按顺时针方向指定的，结点表开始于最小标识符。同步模型里，算法的容许执行完全由初始配置定义。

匹配：若环 R1 上的结点 pi 和 R2 上的 pj 在各自的环里具有相同的位置，则称 pi 和 pj 是匹配的，即：匹配的结点在**各自环上距其最小 id 的结点的距离相同**。

一个算法在两个**相对次序相同的环**上具有**相同的行为**，则该**算法是基于比较**的。

1.序相同：对每个 i 和 j，xi<xj, 当且仅当 yi<yj 两个环 x0 x1… y0 y1 …序等价。
2.行为相同：在序等价的环 R1 和 R2 上的容许执行里，发送同样的 msg 做同样的决策
msg 的模式是：msg 何时何地发送。
3.节点在第 k 轮里行为相似：在同一轮均发一个 msg 给左邻，且同一轮里作为一个 leader 终止。
4.节点的行为相似：每一轮均相似。
5.算法行为相似：指每对匹配的结点行为相似。

算法是基于比较的：每对序等价的环 R1 和 R2，每对匹配的节点在 exec(R1)和 exec(R2)里的行为均相似。即：一算法只与环上标识符之间的相对次序相关，而与具体 id 值无关，则该算法一定只是基于标识符的比较。

基于比较的下界：

只要两个节点有序等价的邻域，它们在 A 里就有同样的行为。
note: 第 1 到第 n 轮里未接收到 msg，实际上蕴含着信息：环里没有标识符为 0 的结点。
若某一轮在任何**次序等价的环上均无 msg 发送**，则**该轮是无用的**，而有用的轮被称为是主动的(active)。
Def 3.3 在一个环 R 上的一个执行里，某轮 r 称为是 active 的，若该执行的第 r 轮里，某一个结点发送一个 msg。当 R 是从上下文已知时，用 r_k 表示第 k 个 active 轮。
一个基于比较的算法在等价环上的行为相似。
一个结点在第 k 个主动轮之后的状态只依赖于它的 k-邻居。这实际上告诉我们：信息只有在主动轮里才能获得。

引理 3.16 **设 R1 和 R2 是次序等价的两个环**，设 Pi 和 Pj 分别是 R1 和 R2 上具有相同 k-邻居的两个节点，那么在 exec(R1)的第 1 至第 r_k 轮里 Pi 所经历的转换序列和在 exec(R2)的第 1 至第 r_k 轮里 Pj 所经历的转换序列相同。

空隙环：每两个 id 之间均有 n 个未使用的标识符 首先数长度 n，然后找标识符之间的差，若每两个 id 的差值均大于 n 则为有空隙的环。

引理 3.17 **设 R 是有空隙环（定义在一个环里）**，Pi 和 Pj 是 R 上具有序等价 k-邻居的两个结点。则 Pi 和 Pj 在 exec(R)的第 1 到第 rk 轮里有相似的行为。

Th3.18 对于每个 n≥8(n 是 2 的幂)，存在一个大小为 n 的环 Sn，使得对每个**基于比较的同步 leader 选举算法 A**，在 Sn 上 A 的容许执行里发送 **msg 的数目为 Ω(nlgn)**。

(1)R 上的 Pi 和 R'上的 Pj 的前 k 个主动轮行为相似

(2)R 上的 Pj 和 R'上的 Pj 的前 k 个主动轮行为相似

(3)R 上两节点的 k-邻居序等价，则其行为在前 k 个主动轮里相似

引理 3.19 对所有 k<n/8 以及每个 Sn 的 k-邻居集 N，在 Sn 中与 N 序等价的 k-邻居集的个数(包括 N 本身)大于 n/(2*(2k+1))

引理 3.20 在 exec(Sn)里，主动轮的数目至少为 n/8。

引理 3.21 对于∀k∈[1，n/8]，在 exec(Sn)的第 k 个主动轮里，至少有 n/2*(2k+1)个 msg 被发送。由引理 3.20 和 3.21 知，在 exec(Sn) 里发送 msg 的总数至少为：

$$\sum_{k=1}^{n/8}\frac{n}{2(2k+1)} \geq \frac{n}{6}\sum_{k=1}^{n/8}\frac{1}{k} > \frac{n}{6}\ln\frac{n}{8}$$

即 Ω(nlgn)，Th3.18 得证。

时间受限算法的下界：

Def3.4 若对任意的 n，当标识符取自于自然数集合时，在所有大小为 n 的环上同步算法 A 的**最坏时间是有界**的，则称 A 为时间有界(或时间受限 time-bounded)。

算法的时间不依赖于 id，仅受限于环大小 n，即使 id 可能是无界的(因为它们来自于自然数集合)。

证明时间受限的同步算法的 msg 复杂性的下界的基本思想是：

将时间受限算法映射为基于比较的算法。从而获得时间受限算法的 msg 复杂性下界 Ω(nlgn)。

Def3.5 设 R1 和 R2 是任意两个大小为 n 的序等价的环，若每对匹配的结点在 exec(R1)和 exec(R2)的第 1 至 t 轮里有相似的行为，则同步算法 A 对于环大小为 n 的标识符集合 S 是基于 t-比较的。

引理 3.22 设 A 是一个运行时间为 r(n)的时间受限的同步算法，则对于每个 n，存在一个具有 n^2+2n 个 id 的集合 Cn，使得 A 是 Cn 上的一个基于 r(n)-比较的算法，这里 n 是环大小。

Th3.23 对每个**同步的时间有界的 leader 选举算法 A**，以及每个 n>8，n 为 2 的方幂，存在一个大小为 n 的环 R，使得 A 在 R 上的容许执行里发送 Ω(nlgn)个 msgs。

Happens-before

H-DAG 有向无环图

若分布式系统中存在全局时钟，则系统中的事件均可安排为全序。例如，可以更公平地分配系统资源。**全序对事件的影响**和**由 H 关系确定的偏序对事件的影响**是**一致**的。

通过 H 关系确定的偏序关系来建立一个"一致"的全序关系？

1）在<H 的 DAG 上拓扑排序

2）On the fly：Lamport 提出了动态即时地建立全序算法

Lamport 算法的思想： 事件时戳 e.TS、节点时戳 my_TS、msg 时戳 m.TS。

事件发生时：

节点执行一个事件时，将自己的时戳赋给该事件；
节点发送 msg 时，将自己的时戳赋给所有发送的 msg。
Lamport 算法的实现：取 msg 时戳和节点时戳的较大者作为新时戳；节点自身+1；
改进：节点地址作为时戳低位。则可得到全序：1.1　1.2… 2.1
特点：任何进程在发送消息前，先将自己的本地计数器累加 1；接收进程总是计算自己的本地计数器和接受到计数器中较大值加上 1 的结果。
然而，**不能通过事件的时戳判定两事件之间是否是因果相关**。
引入向量时戳：VT
my_VT：每个节点有局部的向量时戳
e.VT：每个事件有向量时戳
m.VT：每个 msg 有向量时戳
当且仅当 e.VT 和 e'.VT 是不可比时，称向量时戳是捕获并发的！
向量时钟的次序为：
e1.VT≤V e2.VT iff e1.VT[i]≤ e2.VT[i],i=1,2,…,M
e1.VT<V　e2.VT iff e1.VT≤ VTe2.VT 且 e1.VT≠e2.VT
e1.VT=(5,4,1,3)　e2.VT=(3,6,4,2)　　e3.VT=(0,0,1,3)
e1、e2 是并发的（有大有小），e3 在因果序上先于 e1（e1 所有值都大于 e3）
保证通信不违反因果关系：处理器不能选择 msg 达到的次序，但能**抑制过早达到的 msg 来修正传递**（指提交给应用）次序。
举例：FIFO 通信（如 TCP）若目的处理器收到一个编号为 x 的 msg 但未收到较小编号的 msg 时，则延迟传递直至能够顺序传递为止
因果通信：源：附上时戳，目的地：延迟错序 msg。
抑制从 P 发送的消息 m，直至可断定没有来自于其它处理器上的消息 m'，使 m' <$_V$m.
构建 earliest[k]表示在 P 上，对节点 k 能够传递的 msg 的时戳的下界
blocked[1..M]：阻塞队列数组，每个分量是一个队列:
earliest[k] =1k，k=1,…,M
blocked[k] ={ } ，k=1,…,M//每个阻塞队列置空
not_earliest(A,B,i) 前者不早于后者时为真，其他节点 i 的 earliest[i]不比 k 早，即不比 k 的时间戳小。
使用向量时戳较好，不会假定序。
1）初始化：在本地节点（self）上，能够最早传递的来自于节点 k 的 msg 的时戳存储在本地的 earliest[k]中，line1 初始化正确
2）处理接收的消息：当本地节点接收来自于 p 的消息 m 时，行 5,6：若 blocked[p]中无 msg，则 earliest[p]被置为 m 的时戳，这是因为从 p 上不会有更早的 msg 达到本地，更早的已处理过。
什么样的 msg 是非阻塞的？
当阻塞队列 k（指 blocked[k]）非空时，检测其能否解除阻塞：
设队头 msg 是 m，在 self 上若其他节点 i（除 self 和 k 之外）无更早时戳 earliest[i](即比 m.timestamp=earliest[k]更小)的 msg 可能被传递，则消息 m 是非阻塞的。此时，m 可安全传递，m 出队（行 9）。
当阻塞队列 k 为非空时，检测是否能解除，当队头没有更早时戳 earliest[i]的 msg 需要被传递，则消息 m 非阻塞。
上述算法可能会发生死锁：若一节点长时间不发送你要的 msg，会发生死锁。
组播的应用：若一节点长时间不发送你要的 msg，会发生死锁。多个成员组成的分布式

应用，组内成员往往会发生变动，这需要我们维护关系的一致性。组播消息往往会形成一种相互依赖的因果序关系所以需要因果消息传递算法去控制先后关系。

算法复杂性：
异步环：非均匀 时间　　消息 $O(n^2)$　**选最大标识符**
k 邻居算法 消息复杂性 O(nlgn)

同步环：非均匀 时间 $n\cdot(i+1)$　消息：O(n)　**选最小标识符**
均匀：开始于不同轮，存在唤醒阶段时间：$O(n\cdot 2^i)$ 消息：O(4n)

消息的位（bit）复杂性是通信复杂性　指发送消息的总数
消息链复杂性：是时间复杂度。 最长消息链表征了最长一次发送消息的数量也就是整个系统处理的时间。