第二部分 分布式算法

第四次课

中国科学技术大学计算机系

国家高性能计算中心（合肥）

第三章 环上选举算法

本章提纲

- Leader选举问题
- 匿名环
- 异步环
- 同步环

- **问题**
  在一组处理器中选出一个特殊结点作为leader
- **用途**
  ① 简化处理器之间的协作；
  有助于达到容错和节省资源。
  例如，有了一个leader，就易于实现广播算法
  ② 代表了一类破对称问题。
  例如，当死锁是由于处理器相互环形等待形成时，
  可使用选举算法，找到一个leader并使之从环上删去，即可打破死锁。

§ 3.1 leader选举问题

Leader选举问题：

问题从具有同一状态的进程配置开始，最终达到一种配置状态。每个处理器最终确定自己是否是一个leader，但只有一个处理器确定自己是leader，而其他处理器确定自己是non-leader。

算法的作用：

如果要执行一个分布式算法，且没有一个优先的优选人做为算法的初始进程，就要进行进程选举。(例如指定根的DFS树的生成问题)

§ 3.1 leader选举问题

选举算法的定义：

（1）每个处理器具有相同的局部算法；
（2）算法是分布式的，处理器的任意非空子集都能开始一次计算；
（3）每次计算中，算法达到终止配置。在每一可达的终止配置中，只有一个处理器处于领导人状态，其余均处于失败状态。（此项有时可以弱化）

一个算法解决了leader选举问题需满足(根据形式化模型)：

① 终止状态被划分为两类：选中和未选中状态。一旦一个处理器进入选中(或未选中)状态，则该处理器上的转换函数将只会将其变为相同的状态；

② 在每个容许执行里，只有一个处理器进入选中状态，其余处理器进入非选中(non-selected)状态。

本章只讨论系统的拓扑结构是环的情况。

§ 3.1 leader选举问题

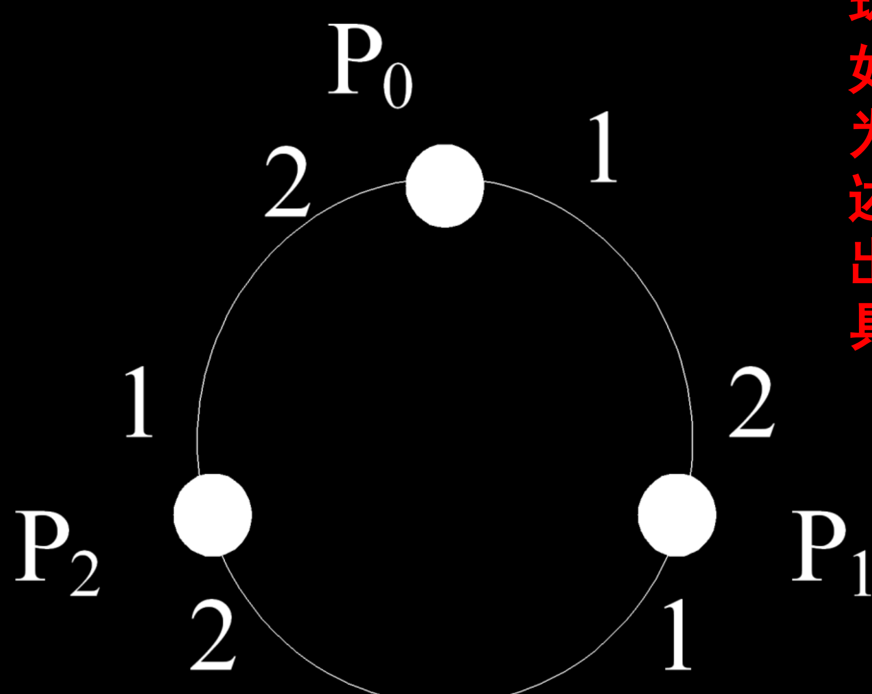
- **环的形式化模型**

  对每个i，0≤i ≤n-1，

  P_i到P_{i+1}的边标号为1，称为左(顺时针)

  P_i到P_{i-1}的边标号为2，称为右(逆时针)

  这里的标号加减均是mod n的



环网络之所以吸引了如此多的研究，是因为它们的行为易于描述；且从环网络推导出的下界，可应用于具有任意拓扑结构的网络算法设计

§3.2 匿名环（anonymous）

- **匿名算法**：若环中处理器没有唯一的标识符，则环选举算法是匿名的。**更形式化的描述：**每个处理器在系统中具有相同的状态机，在这种算法里，msg接收者只能根据信道标号来区别。
- **（一致性的）uniform算法**：若算法不知道处理器数目，则算法称之为uniform，因为该算法对每个n值看上去是相同的。
- **non-uniform算法**：算法已知处理器数目n
- **形式化描述：**在一个匿名、一致性的算法中，所有处理器只有一个状态机；在一个匿名、非一致性的算法中，对每个n值（处理器数目）都有单个状态机，但对不同规模有不同状态机，也就是说n可以在代码中显式表达。

§3.2 匿名环（anonymous）

- 对于环系统，不存在匿名的选举算法。
  为简单起见，我们只证明
  非均匀（非一致性）算法：非均匀算法（n已知）的不可能性=>均匀（n未知）算法的不可能性。
  Ex3.1 证明同步环系统中不存在匿名的、一致性的领导者选举算法。

  同步算法：同步算法的不可能性=>异步算法的不可能性。（同步是异步的一种特例）
  Ex3.2 证明异步环系统中不存在匿名的领导者选举算法。

§ 3.2 匿名环

- 同步算法的不可能性

在同步系统中，一个算法以轮的形式进行。每轮里所有待发送msg被传递，随后每个处理器进行一步计算。

一个处理器的初始状态包括在outbuf里的任何msg。这些消息在第一轮里被传递到某处理器的左和右邻居。

不可能性：

①在一个匿名环中，处理器间始终保持对称，若无某种初始的非对称(如，标识符唯一)，则不可能打破对称。在匿名环算法里，所有处理器开始于相同状态。

②因为他们执行同样的程序(即他们的状态机相同)，在每轮里各处理器均发送同样的msg，所以在每一轮里各处理器均接收同样的msg，改变状态亦相同。

因此，若选中一个处理器，则其他所有处理器亦被选中。因此，不可能有一个算法在环中选中单个处理器为leader。

§3.2 匿名环

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。

- **Lemma3.1** 在环R上算法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。

Pf. 对k用归纳法

K=0(第一轮之前)，因为处理器在开始时都处在相同的初始状态，故结论是显然的。

设引理对k-1轮成立。因为在该轮里各处理器处在相同状态，他们都发送相同的消息m_r到右边，同样的消息m_l到左边，所以在第k轮里，每处理器均接收右边的m_l，左边的m_r。因此，所有处理器在第k轮里接收同样的消息，又因为它们均执行同样的程序，故第k轮它们均处于同样的状态。

§3.2 匿名环

上述引理蕴含着：若在某轮结束时，一个处理器宣布自己是leader(进入选中状态)，则其它处理器亦同样如此，这和A是一个leader选举算法的假定矛盾！因此证明：

- **Th3.2** 对于同步环上的leader选举，不存在非均匀的匿名算法。

＋　　　　　　　　　　＋

同步环→异步环　　　　非一致性→一致性算法

↓↓

对于环系统，不存在匿名的选举算法

§3.3 异步环

本节将讨论异步环上leader选举问题的msg复杂性上下界。

由Th3.2知，对环而言没有匿名的leader选举算法存在。因此以下均假定处理器均有唯一标识符。

当一个状态机(局部程序)和处理器P_i联系在一起时，其状态成分变量id_i被初始化为P_i的标识符的值，故各处理器的状态是有区别的。

- **环系统：**通过指派一个处理器列表按顺时针(从最小标识符起)指定环。注意是通过id排列，不是通过P_i的下标i来排列($0 \le i \le n-1$)，假定id_i是P_i的标识符。（因为下标i通常是不可获得的）

§3.3 异步环

- 在非匿名算法中，均匀（一致性）和非均匀（非一致性）的概念稍有不同
  ① 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。（不管环的规模如何，只要处理器分配了对应其标识符的唯一状态机，算法就是正确的。）
  ② 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。（对每一个n和给定规模n的任意一个环，当算法中每个处理器具有对应其标识符的环规模的状态机时，算法是正确的。）

  下面将讨论msg复杂性：$O(n^2) \to O(n\log n) \to \Omega(n\log n)$

下界

§3.3.1 一个$O(n^2)$算法

Le Lann、Chang和Roberts给出，LCR算法

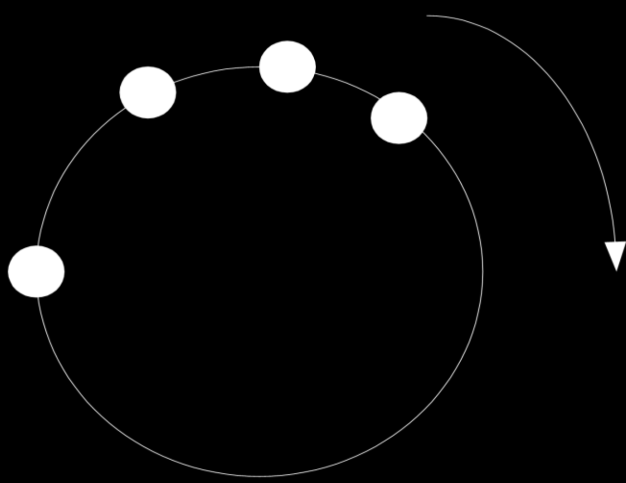
- 基本思想
  ① 每个处理器P_i发送一个msg(自己的标识符)到左邻居，然后等其右邻居的msg
  ② 当它接收一个msg时，检验收到的id_j，若$id_j>id_i$，则P_i转发id_j给左邻，否则没收id_j(不转发)。

§ 3.3.1 一个O(n^2)算法

③ 若某处理器收到一个含有自己标识符的msg，则它宣布自己是leader，并发送一个终止msg给左邻，然后终止。

④ 当一处理器收到一个终止msg时，向左邻转发此消息，然后作为non-leader终止。

因为算法不依赖于n，故它是均匀的。



i—表示id 单向

§ 3.3.1 一个$O(n^2)$算法

Code for P_i

```
init var: asleep←true, id ←I
Begin
 While (receiving no message) do
  (1) if asleep do
      (1.1) asleep←false
      (1.2) send <id> to left-negihbor
    end if
 End while
 While (receiving <i> from right-neighbor) do
  (1) if id<<i> then send <i> to left-neighbor
       end if
  (2) if id=<i> then
      (2.1) send <Leader,i> to left-neighbor
      (2.2) terminates as Leader
    end if
 End while
 While (receiving <Leader,j> from right-neighbor) do
  (1) send <Leader,j> to left-neighbor
  (2) terminates as non-Leader
 End while
end
```

§ 3.3.1 一个O(n^2)算法

- **分析**

① **正确性**

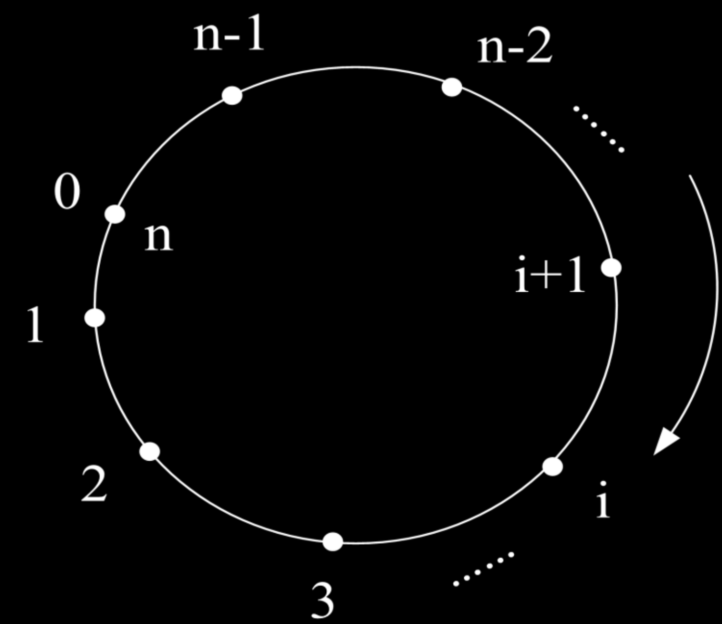
在任何容许执行里，只有最大标识符id_{max}不被没收，故只有具有id_{max}的处理器接受自己的标识符并宣布是leader，其他处理器不会被选中，故算法正确。

② **msg复杂性**

在任何容许执行里，算法绝不会发送多于O(n^2)个msgs，更进一步，该算法有一个容许执行发送O(n^2)个msgs：

§ 3.3.1 一个O(n²)算法

考虑处理器标识符为0，1，…，n-1构成的环，其次序如右图：



在这种配置里，id=i的处理器的msg恰好被发送i+1次，即发送到i-1,i-2,…,1,0，直到n-1时没收。因此，msg总数为：

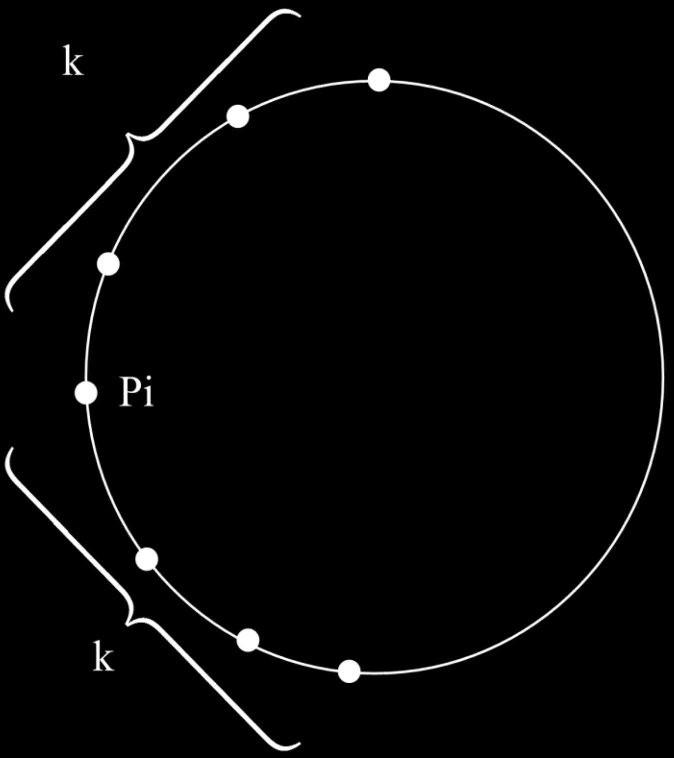
$$n+\sum_{i=0}^{n-1}(i+1)=\theta(n^2)$$ //前一项为终止msg

§ 3.3.2 一个O(*n*lg*n*)算法

仍然是绕环发送id，但使用更聪明的方法。保证最大id在环上周游且返回。

- **k邻居**

一个处理器P_i的k邻居是一个处理器集合：该集合中的任一处理器与P_i在环上的距离至多是k，一个处理器的k-邻居集合中恰好有2k+1个处理器。



k=3，共有7个结点

§ 3.3.2 一个O(*n*lg*n*)算法

■ 基本思想

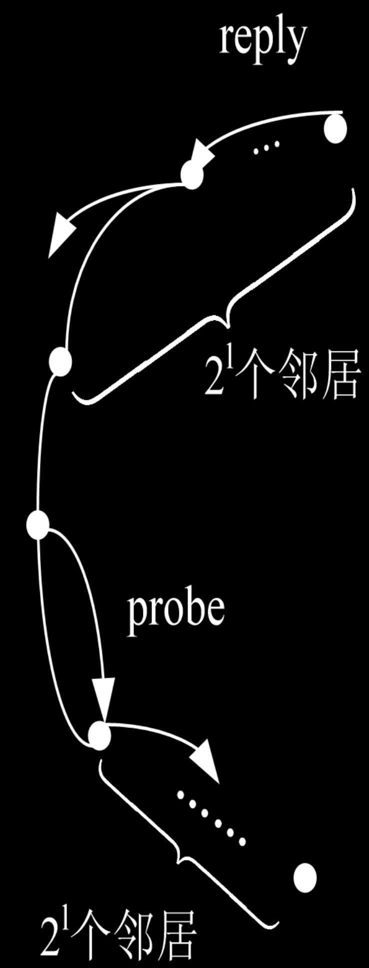
算法按阶段执行，在第*l*阶段一个处理器试图成为其2^l-邻接的临时leader。只有那些在*l*-th阶段成为临时领袖的处理器才能继续进行到(*l*+1)th阶段。因此，*l*越大，剩下的处理器越少。直至最后一个阶段，整个环上只有一个处理器被选为leader。

■ 具体实现

① phase0: 每个结点发送1个probe消息(其中包括自己的id)给两个1-邻居，若接收此msg的邻居的id大于消息中的id，则没收此msg；否则接收者发回一个reply msg。若一个结点从它的两个邻居收到回答msg reply，则该结点成为phase0里它的1-邻居的临时leader，此结点可继续进行phase1。

§ 3.3.2 一个O($n$lg$n$)算法

② **phase l**：在l-1阶段中成为临时leader的处理器P_i发送带有自己id的probe消息至它的2^l邻居。若此msg中的id小于左右两个方向上的$2*2^l$个处理器中任一处理器的id，则此msg被没收。若probe消息到达最后一个邻居而未被没收，则最后一个处理器发送reply消息给P_i，若P_i从两个方向均接收到reply消息，则它称为该阶段中2^l邻居的临时leader，继续进入下一阶段。



③ **终止**：接收到自己的probe消息的结点终止算法而成为leader，并发送一个终止msg到环上。

§ 3.3.2 一个O($n$lg$n$)算法

④ 控制probe msg的转发和应答

probe消息中有三个域：<prob, id, l, hop>

id-标识符

l-阶段数

hop-跳步计数器：初值为0，结点转发probe消息时加1.

若一结点收到的probe消息时，hop值为2^l，则它是2^l邻居中最后一个处理器。若此时msg未被没收也不能向前转发，而应该是向后发回reply消息。

§ 3.3.2 一个O(*n*lg*n*)算法

- 算法：Alg3.1 异步leader选举

```
var asleep init true;
upon receiving no msg:
   if  asleep then{
       asleep:=false;//每个结点唤醒后不再进入此代码
       send<probe, id, 0, 0> to left and right;
   }
upon receiving <probe, j, l, d> from left (resp, right):
   if(j=id) then //收到自己id终止，省略发终止msg
        terminate as the leader;
  if(j>id) and (d<2^l) then   //向前转发probe msg
              send <probe, j, l, d+1> to right (resp, left)
```

§ 3.3.2 一个O(*n*lg*n*)算法

if(j>id) and (d≥2^l)then//到达最后一个邻居仍未没收

send <reply, j, *l* > to left(resp, right) // 回答

//若j<id, 则没收probe消息

upon receiving <reply ,j , *l*> from left (resp, right):

if j≠id then

send<reply, j, l> to right (resp, left); //转发reply

else //j=id时，P_i已收到一个方向的回答msg

if already received <reply, j, *l*> from right (resp, left)

then//也收到另一方向发回的reply

send <probe, id, l+1, 0> to left and right;

//P_i是phase *l*的临时leader，继续下一阶段

§ 3.3.2 一个O($n\lg n$)算法

■ 分析

① **正确性：**因为具有最大id的处理器的probe消息是不会被任何结点没收的，所以该处理器将作为leader终止算法；另一方面，没有其他probe消息能够周游整个环而不被吞没。因此，最大id的处理器是算法选中的唯一的leader。

② **msg复杂性（最坏情况下）**

在phase l 里：

- 一个处理器启动的msg数目至多为：$4*2^l$
- 有多少个处理器是启动者呢？

  - $l=0$，有n个启动着（最多）

  - $l \geq 1$，在l-1阶段结束时成为临时leader的节点均是启动者

§ 3.3.2 一个O($nlgn$)算法

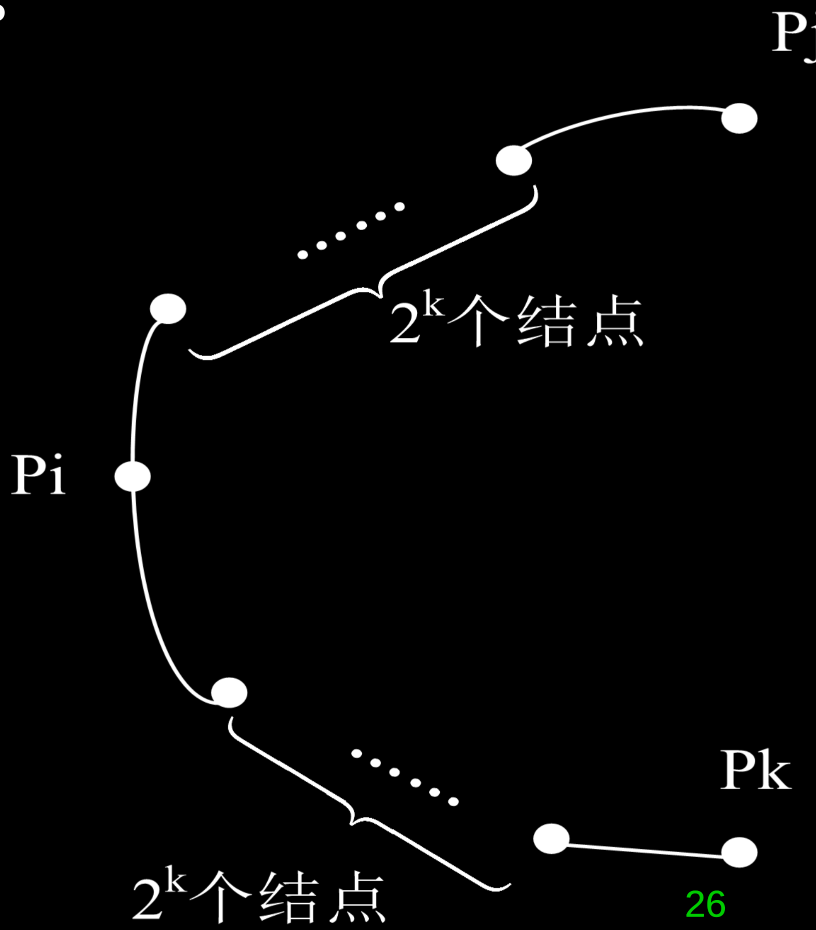
Lemma 3.3 对每个k≥1，在phase k结束时，临时leader数至多为$n/(2^k+1)$.

pf: 若一结点P_i在k阶段结束时是一临时leader，则在P_i的2^k-邻居里每个结点的id均小于P_i的id。

在该阶段里，距离最近的两个临时leader P_i和P_j必满足：

P_i的2^k邻居的左边恰好P_j的2^k-邻居的右边，即P_i和P_j之间有2^k个处理器。

因此，在phase k里临时leader的最大数目必是以上述方式分布的，因为每2^k+1个结点至多有一个临时leader，所以leader数至多是$n/(2^k+1)$.



§ 3.3.2 一个O(*n*lg*n*)算法

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

Pf: 由lemma3.3知， 知道phase lg(n-1)时只剩下一个leader(最后的leader). msg总数：

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

i) phase 0: msg数为4n.

ii)终止msgs： n.

Note: 双向通信. 该msg复杂性的常数因子不是最优的.

§ 3.3.3 下界Ω(*n*lg*n*)

现证明对于uniform算法，异步环里任何leader选举算法至少发送**Ω(*n*lg*n*)**个**msgs**。

我们的下界证明是针对leader选举问题的一个变种：

- 选中的leader必定是环上具有最大id的处理器。
- 所有处理器必须知道被选中leader的id，即每处理器终止前，将选中leader的id写入一个特殊变量。

- **基本思想。**

设A是一个能解上述leader选举变种问题的均匀算法，证明存在A的一个允许执行，其中发送了**Ω(*n*lg*n*)个msgs，证明采用构造法。**

§ 3.3.3 下界$\Omega(n\lg n)$

对于大小为n/2的环构造算法的一个耗费执行(指msg的耗费)，然后将两个大小为n/2的不同环粘贴在一起形成一个大小为n的环，将两个较小环上的耗费执行组合在一起，并迫使θ(n)个附加msg被接收。这种扩展依赖于算法是一致的且对各种规模的环以相同的方式执行

- 调度：前面定义过调度是执行中的事件序列，下面给出能够被粘贴在一起的调度。
- Def3.1 开调度

设σ是一个特定环上算法A的一个调度，若该环中存在一条边e使得在σ中，边e的任意方向上均无msg传递，则σ称为是open，e是σ的一条开边。

§ 3.3.3 下界$\Omega(n\lg n)$

Note：开调度未必是容许的调度，即它可能是有限的事件序列，环上的处理器不一定是终止的。

直观上，既然处理器不知道环的大小，我们能将两个较小的开调度粘贴为一个较大环的开调度，其依据是：算法是均匀的。

为简单起见，不妨设n为2的整数次幂。

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：

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

§ 3.3.3 下界Ω($n$lg$n$)

显然递归方程的解为M(n)=θ($n$lg$n$)，他蕴含了异步环选举问题消息复杂度下界。下面用归纳法证明之，其中

$$\begin{cases} \text{引理3.6是归纳基础}(n = 2^1) \\ \text{引理3.7是归纳步骤}(n = 2^i, i > 1) \end{cases}$$

Lemma3.6 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）

pf：假定R有两个处理器P_0和P_1，其标识符分别为x和y，不妨设x>y.

§ 3.3.3 下界Ω($nlgn$)

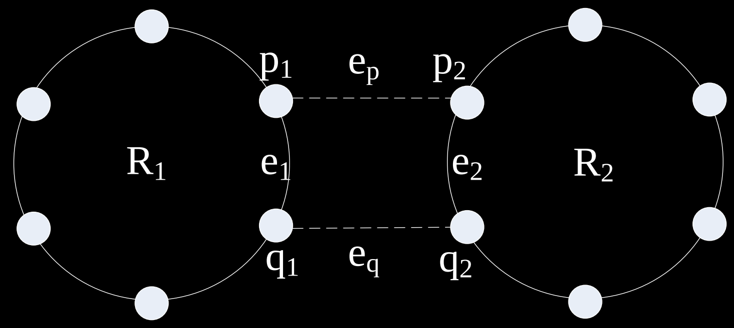
设α是A的一个容许执行，因为A是正确的，在α中，最终P_1定将P_0的标识符写入其中。因此，α中至少须接收一个msg，否则P_1不知道P_0的标识符为x.

设σ是α的调度的最短前缀：它包括第一个接受msg的事件。因为没有接收第一条msg的边是开的，因此σ中只有一个msg被接收且有一条开边，故引理成立。故σ是满足引理的开调度。

Lemma 3.7 选择n>2，假定对每个大小为n/2标识符集合，存在一个使用这些标识符的环，它有A的一个开调度，其中至少接收M(n/2)个msgs(归纳假设)，那么对于n个标识符的每个集合，存在一个使用这些标识符集的环，它有A的一个开调度，其中接收至少2M(n/2)+(n/2-1)/2个msgs(归纳步骤)。

§ 3.3.3 下界Ω(*n*lg*n*)

pf：设S是n个标识符的集合，将S划分为两个集合S1和S2，每个大小为n/2，由假设分别存在一个使用S1和S2中标识符的环R1和R2，它们分别有A的一个开调度σ1和σ2，其中均至少接收M(n/2)个msgs，设e1和e2分别是σ1和σ2的开边，不妨设邻接于e1和e2的处理器分别是p1，q1和p2，q2，将e1，e2删去，用ep链接p1和p2，eq链接q1和q2，即可将两个环R1和R2粘贴在一起形成环R。



R_1 U R_2=>R

现说明如何在R上构造一个A的开调度σ，其中至少有2M(n/2)+(n/2-1)/2个msg被接收。其想法是先让每个较小环分别执行"耗费"的开调度。

§ 3.3.3 下界Ω(*nlgn*)

1) $\sigma_1\sigma_2$构成R上A的一个开调度

考虑从R的初始配置开始发生的事件序列σ_1，因为R_1中的处理器由这些事件并不能区别R_1是一个独立的环还是R的一个子环，它们执行σ_1恰像R_1是独立的那样。考虑环R上后续事件序列σ_2(与上类似)，因为没有msg在e_p和e_q上传递，故R_2中处理器在σ_2中亦不能区别R_2是独立环还是R的子环。

因此，$\sigma_1\sigma_2$是一个调度，其中至少有2M(n/2)个msgs被接收。

2) 现说明如何通过连通e_p和e_q(但不是二者)来迫使算法接收(n/2-1)/2个附加的msgs。

考虑每个形式为$\sigma_1\sigma_2\sigma_3$的有限调度，因为$\sigma_1\sigma_2$中e_p和e_q均为开的，若σ_3中存在一边上至少有(n/2-1)/2个msg被接收，则$\sigma_1\sigma_2\sigma_3$是要找的开调度，引理被证。

假设没有这样的调度，那么存在某个调度$\sigma_1\sigma_2\sigma_3$，它导致相应执行中的一个“静止”配置。(配置：由全体结点状态构成)

一个处理器状态是“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）

§3.3.3 下界$\Omega(n\lg n)$

一个配置是“静止”的(关于e_p和e_q)：若除开边e_p和e_q外，没有msgs处在传递之中，每个处理器均为静止状态。

不失一般性，假设R中最大id的处理器是在子环R_1中，因为没有msg从R_1传到R_2中，R_2中的处理器不知道leader的id，因此R_2里没有处理器能够在$\sigma_1\sigma_2\sigma_3$结束时终止。(在$\sigma_1\sigma_2\sigma_3$结束时，R_2里无结点终止)

我们断定在每个扩展$\sigma_1\sigma_2\sigma_3$的容许调度里，子环R_2里的每个处理器在终止前必须接收至少一个附加msg，因为R_2里每一处理器只有接收来自R_1的msg才知道leader的id。

上述讨论清楚地蕴含在环R上必须接收$\Omega(n/2)$个msgs，但因为e_p和e_q是连通的，故调度未必是开的，即两边上均可能传递msg。

但若能说明e_p或e_q只有一个是连通的，迫使通过它接收$\Omega(n/2)$个msgs，即可证明。这就是下一断言。

§ 3.3.3 下界Ω(*n*lg*n*)

Claim3.8 存在一个有限的调度片断σ_4，其中有 (n/2-1)/2个msgs被接收，$\sigma_1\sigma_2\sigma_3\sigma_4$是一个开调度，其中$e_p$或$e_q$是开的。

Pf: 设 σ_4 使得$\sigma_1\sigma_2\sigma_3$ σ_4 是一个容许调度，因此所有的msgs在e_p和e_q上传递，所有结点终止。

因为R_2里，每个节点在终止前必须收到一个msg，故在A终止前在 σ_4 里至少接收n/2个msgs,设 σ_4' 是 σ_4'' 里接收n/2-1个msg的最短前缀。

考虑 R 里在 σ_4 中所有已接收msg的结点，因为我们是从一个静止位置开始的，其中只有在e_p和e_q上有msg在传输，故这些结点形成了两个连续的结点集合P和Q：

P包含由于连通e_p而被唤醒的结点，故P至少包含p_1和p_2

Q包含由于连通e_q而被唤醒的结点，故Q至少包含q_1和q_2

§ 3.3.3 下界Ω(*n*lg*n*)

因为P∪Q中至少包含n/2-1个结点（由 σ_4' 决定），且又因它们中的结点是连续的，所以P∩Q=Φ。Ｐ和Ｑ这两个集合中有一个集合，其中的结点至少接收 (n/2-1)/2个msg,(因为Ｐ，Ｑ中的结点共接收n/2-1个msg),不失一般性，假定这样的集合是Ｐ。

设σ_4是 σ_4'的子序列，σ_4只包含在Ｐ中结点上发生的事件，因为 σ_4'里Ｐ中节点和Q中结点之间没有通信，故$\sigma_1\ \sigma_2\ \sigma_3\ \sigma_4$是一个调度。

因为σ_4里至少有 (n/2-1)/2各msg被接收，且由构造可知，e_q上无msg传递，因此$\sigma_1\ \sigma_2\ \sigma_3\ \sigma_4$是一个满足要求的开调度。

§ 3.3.3 下界$\Omega(n\lg n)$

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: $\sigma_1\ \sigma_2$

2) 强迫环进入一个静止配置： $\sigma_1\ \sigma_2\ \sigma_3$ （主要由调度片断σ_3 ）

3) 强迫(n/2-1)/2个附加msg被接收，并保持ep或eq是开的: $\sigma_1\ \sigma_2\ \sigma_3\ \sigma_4$。

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。

下次继续！