並列分散コンピューティング
(10)リーダー選出問題

大瀧保広

今日の内容

- 分散アルゴリズムを考える時の前提と評価について
- リーダー選出問題
  ※文脈やアルゴリズムによって様々な表現が使われる。
  - リーダー
  - 代表プロセス
  - 調停者（Coordinator）

  - ブリーアルゴリズム
  - リングアルゴリズム
    - 素朴なアルゴリズム
    - Chang-Robertsアルゴリズム
    - （Franklinアルゴリズム）
    - Patersonアルゴリズム

分散アルゴリズムを考える時の前提について

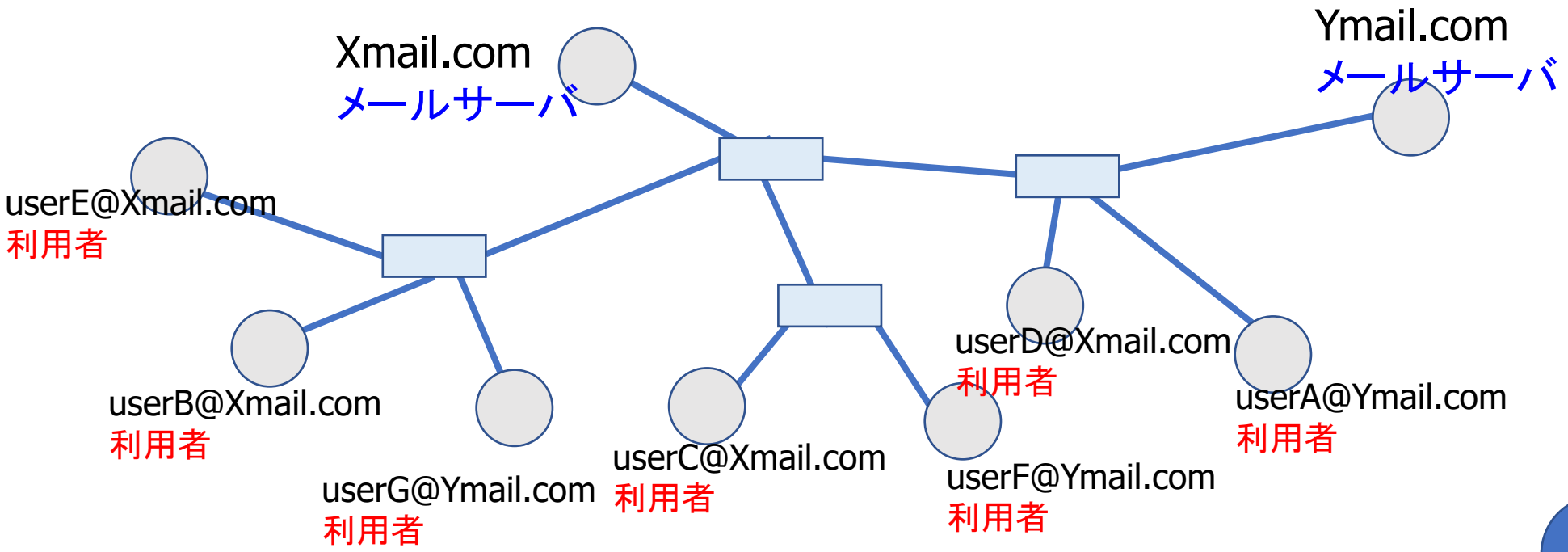
分散アルゴリズムを考えるときには、
問題によって、モデル化や簡素化のために様々な前提が置かれる。

- ネットワークの形態
- プロセスや通信路の状態

- ネットワークの形態
  - アプリケーションレイヤーで認識される「つながり方」は
    物理的な接続と異なることはよくある。
    オーバーレイネットワーク（Over Lay Network）などと呼ぶこともある。
  - 分散システムを構成する各プロセス（コンピュータ）が
    どのように相互に「接続」されているか
    「接続」は、通信路があるかどうか　だけを意味しており、
    各ノード間が物理的にどう接続されているかは気にしていない。

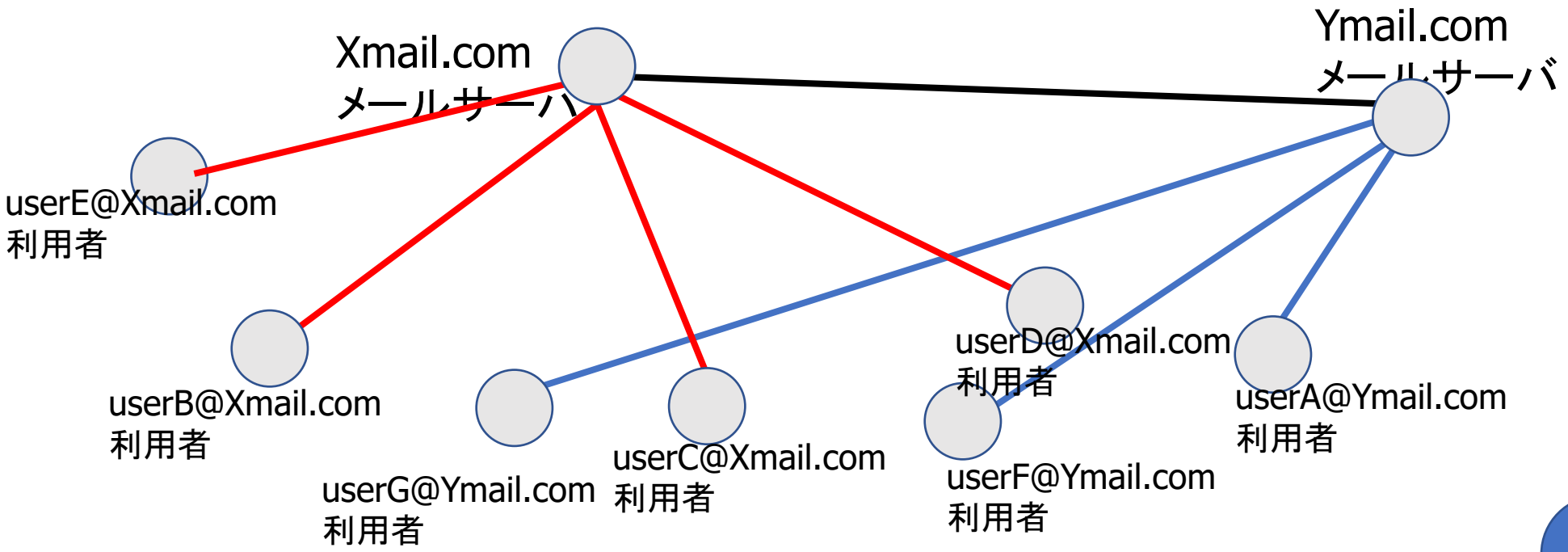
例：メールシステム（物理接続でみると）

メールサーバや利用者のPCがルータなどで相互接続されてる。
利用するメールサーバが異なる利用者が
同じルータの下に混在している。



例：メールシステム（論理接続でみると）

- 各利用者のPCは自分が利用しているメールサーバと通信
- メール配送のためメールサーバ同士が通信



分散アルゴリズムの評価

- 一般にCPUの処理よりも通信の方が時間がかかる
- 分散アルゴリズムの性能は
  通信のオーバーヘッドに大きく影響される。

- 一般に以下の2つの観点で評価される
  - 通信ステップ数：
    アルゴリズムが停止するまでに要する時間
  - 通信複雑さ：
    アルゴリズムが停止するまでに送信される、
    メッセージの総数

分散アルゴリズムの説明で出現する用語

- 文脈によるが、以下の用語は ほぼ同じ意味で使われる
  - ノード
  - プロセス　←OS用語のプロセスとは使い方がズレている
  - コンピュータ

- **始動プロセス**

  アルゴリズムの実行を開始するプロセス

- それ以外のプロセスは

  他のプロセスからの通信を受けて、
  アルゴリズムを開始（起動）する。

分散アルゴリズムを学ぶ時の注意点

- 分散アルゴリズムの説明では、
  分散システム全体を見渡せる「神の視点」での説明をする。

- 分散アルゴリズムは、個々のプロセスで実行される
  個別の処理の集合体として実装される。
- このとき、各プロセスの個別の処理では、
  自分のローカルな情報だけで処理を進めており、
  全体の状況は見えていない。

全体が見渡せる視点と、各プロセスの立場に立った視点の
2つが違うことを、意識しておくこと。

リーダー選出問題

Leader Election Problem

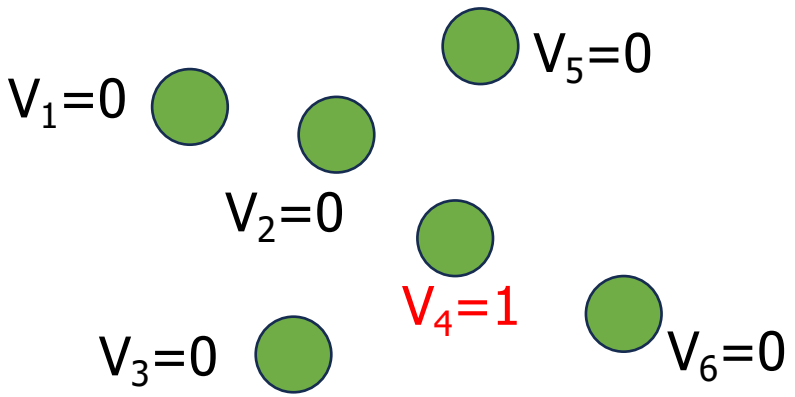
問題の背景

- 分散システムでは、集中管理は望ましくないとされている。
  (管理プロセスのところがSingle Point of Failure になってしまう)
- 耐故障性の観点からは、全プロセスが平等な役割を持つように
  設計した方がよいが、状況によっては
  代表役のプロセスがあるほうが処理が簡単な場合も多い。
  - 通信を開始する権利をもつ通信制御プロセス
  - 資源の割り当てを行うプロセス

- 例：Berkleyアルゴリズムに基づく時刻同期では、
  時刻同期アルゴリズムを定期的に始動するマスターサーバが必要。

リーダー選出問題の定式化

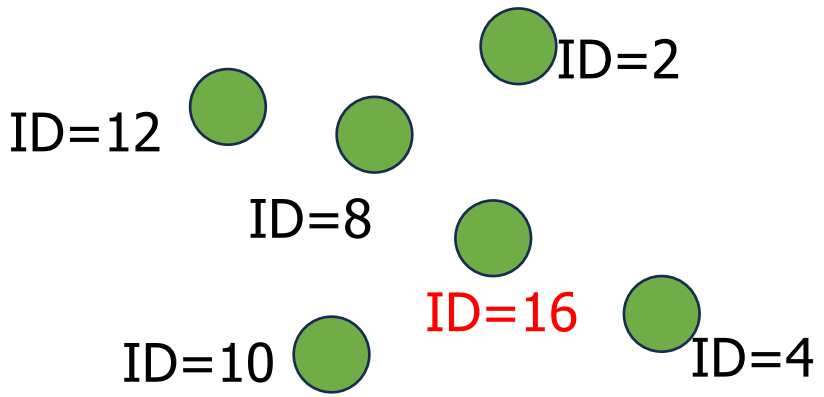
- リーダー選出問題とは、分散システムにおいて
  特別な役割をもつプロセスを1つ決定する問題である。
  - 記号で記述するならば
    各プロセス P_i が変数 V_i を持つとき、
    あるプロセス P_a の変数 V_a のみが 1となり、
    それ以外のプロセスの変数は 0 となるように
    セットする問題
    と定式化される。



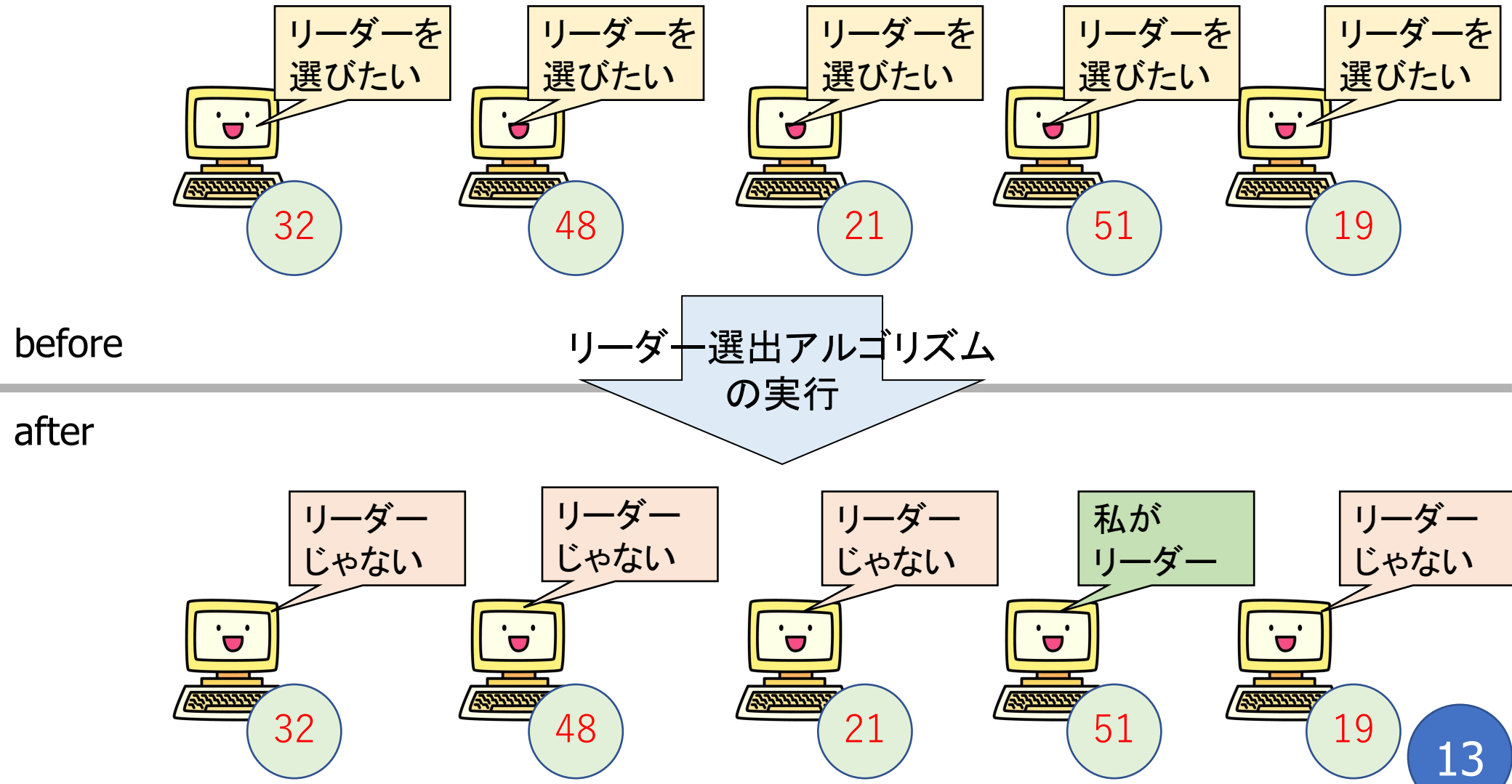
リーダー選出問題の定式化

- リーダー選出問題は「最大値発見問題」に帰着できる。

最大値発見問題：
各プロセス P_i が相異なる固有の値(ID) をもっているとき、
最大のIDをもつプロセスを見つける問題



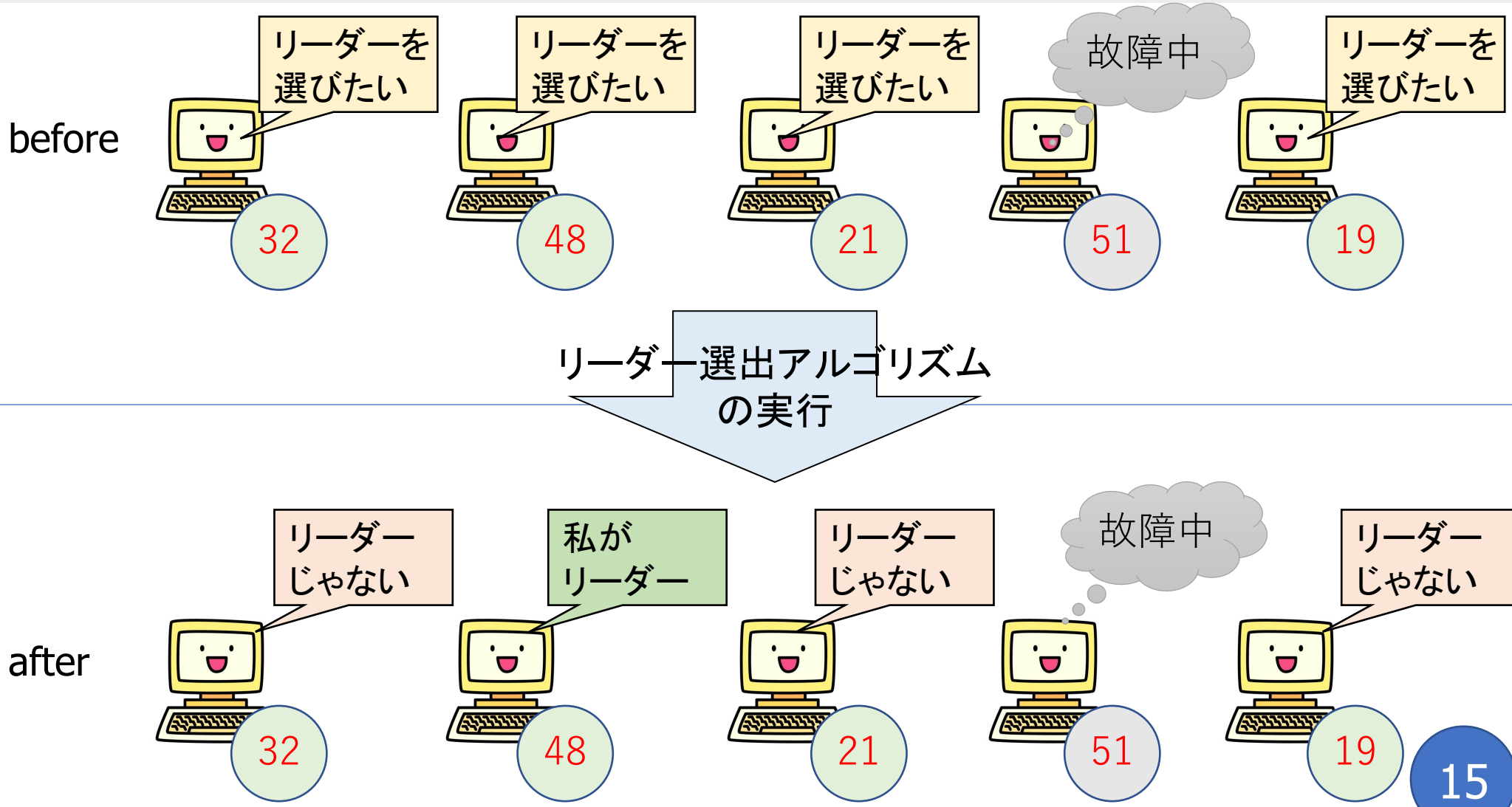
リーダー選出問題 ビフォア アフター



リーダー選出問題のゴール（要求要件）

- 一つのプロセスだけがリーダーとなる。
- 選出されたプロセスは、
  自分がリーダーになったことを知る。
- 選出されなかったプロセスは、
  - 選出されなかったことを知る。
  - 誰がリーダーになったのか を知る。

リーダー選出問題、どこが難しいのか



リーダー選出問題、どこが難しいのか

- リーダー選出問題は、合意問題の一種である。
- 合意問題は、
  - 同期システム　か　非同期システム　か
  - 通信路で起きる通信障害の種類
  - プロセスの障害（故障）はあるか
    どのような故障のタイプか

  などによって「解決不能（Unsolvable）」になることがある。

リーダー選出問題の前提

- プロセスの状態は
  正常稼働 または 完全停止 のいずれか
  とする。
- 通信路は信用できる。
  正常なプロセス間の通信は
  大幅な遅延なく、誤りなく到達する。

送信メッセージに対して返事が来ることが想定されている場合、
そのプロセスが生きていれば、一定時間以内に必ず返事が来る。

ブリーアルゴリズム（Bully Algorithm）

いじめっこアルゴリズム：強いものが弱いものを蹴散らしていく感じ

Bully-free：いじめのない〜

ブリーアルゴリズム

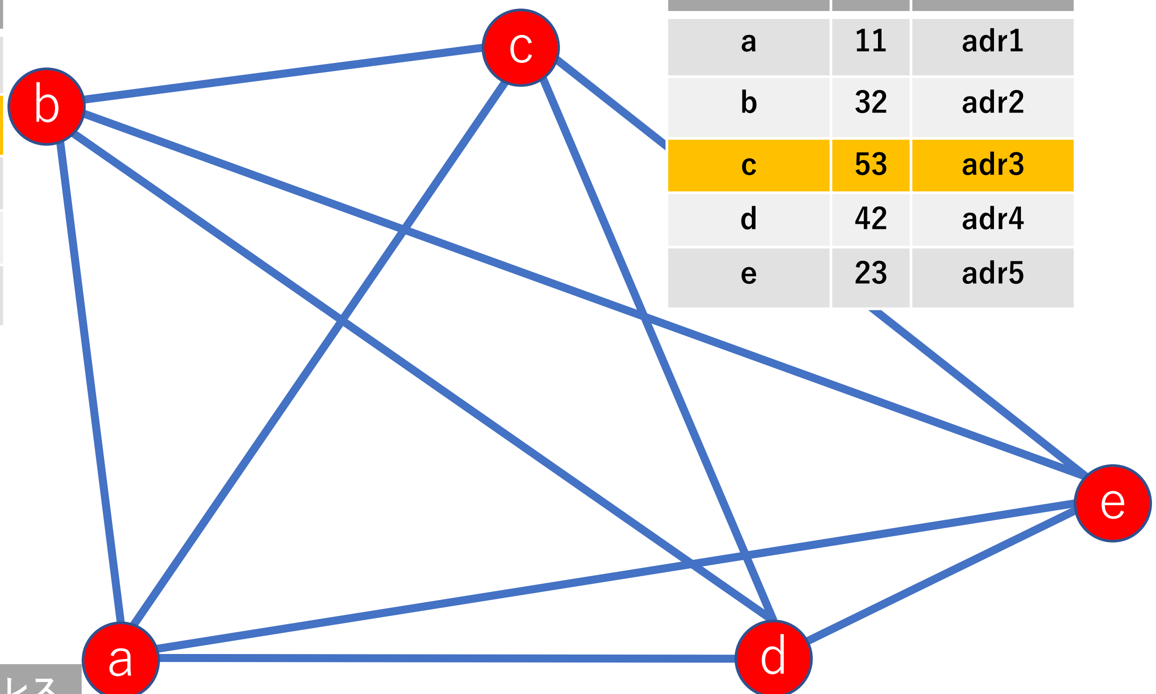
- 前提
  - 各プロセスには一意な数値IDがある。
  - 通信路は完全グラフ（＝どの相手とも直接通信できる）
  - 各プロセスは、他のプロセスのIDとそのアドレスを知っている。
  - どのプロセスが稼働しているか、停止しているかはわからない。
  - 複数のプロセスが同時に始動プロセスとなる可能性がある。

最大のIDをもつプロセスは
最初からわかってるんじゃないの？

ブリーアルゴリズムが想定する世界

プロセス	ID	アドレス
a	11	adr1
b	32	adr2
c	53	adr3
d	42	adr4
e	23	adr5

プロセス	ID	アドレス
a	11	adr1
b	32	adr2
c	53	adr3
d	42	adr4
e	23	adr5



プロセス	ID	アドレス
a	11	adr1
b	32	adr2
c	53	adr3
d	42	adr4
e	23	adr5

プロセス	ID	アドレス
a	11	adr1
b	32	adr2
c	53	adr3
d	42	adr4
e	23	adr5

プロセス	ID	アドレス
a	11	adr1
b	32	adr2
c	53	adr3
d	42	adr4
e	23	adr5

ブリーアルゴリズム

- ブリーアルゴリズムで使うメッセージの種類は3つ。
  - 「Election」
  - 「OK」
  - 「I won」

次の順序で説明していきます。

- 始動プロセスが行う処理の流れ
- それ以外のプロセスが行う処理の流れ
- 選出後の処理の流れ

ブリーアルゴリズム(始動プロセス)

始動プロセスでの処理

■(以前の)リーダープロセスの停止を検知したプロセスPは、
選出アルゴリズムを起動する（始動プロセスとなる）。

選出アルゴリズム

- 自分より大きなIDをもつプロセスに対して、
  Electionメッセージ を送信する。
- **もし誰も応答しなければ、**
  自分がリーダーとして選出されたことになる。
  （この後、選出後の処理へ）
- もし応答（OKメッセージ）があれば、
  自分より大きいIDをもつプロセスがいたことになる。
  選出アルゴリズムはそのプロセスに引き継がれる。
  自分の作業は一旦終了。　（選出終了の連絡待ち）

ブリーアルゴリズム(始動プロセス以外)

始動プロセス以外のプロセスの処理

■各プロセスは常に自分より小さいIDを持つプロセスから
Electionメッセージを受け取る可能性がある。

■Electionメッセージを受け取ったプロセスは、
(自分の方が大きいIDなので)OKメッセージを返す。

■OKを返したら、選出アルゴリズムを引き継ぐ。
選出アルゴリズムを（まだ起動していなければ）起動する。
（＝前のスライドの処理を行う）

ブリーアルゴリズム(選出後)

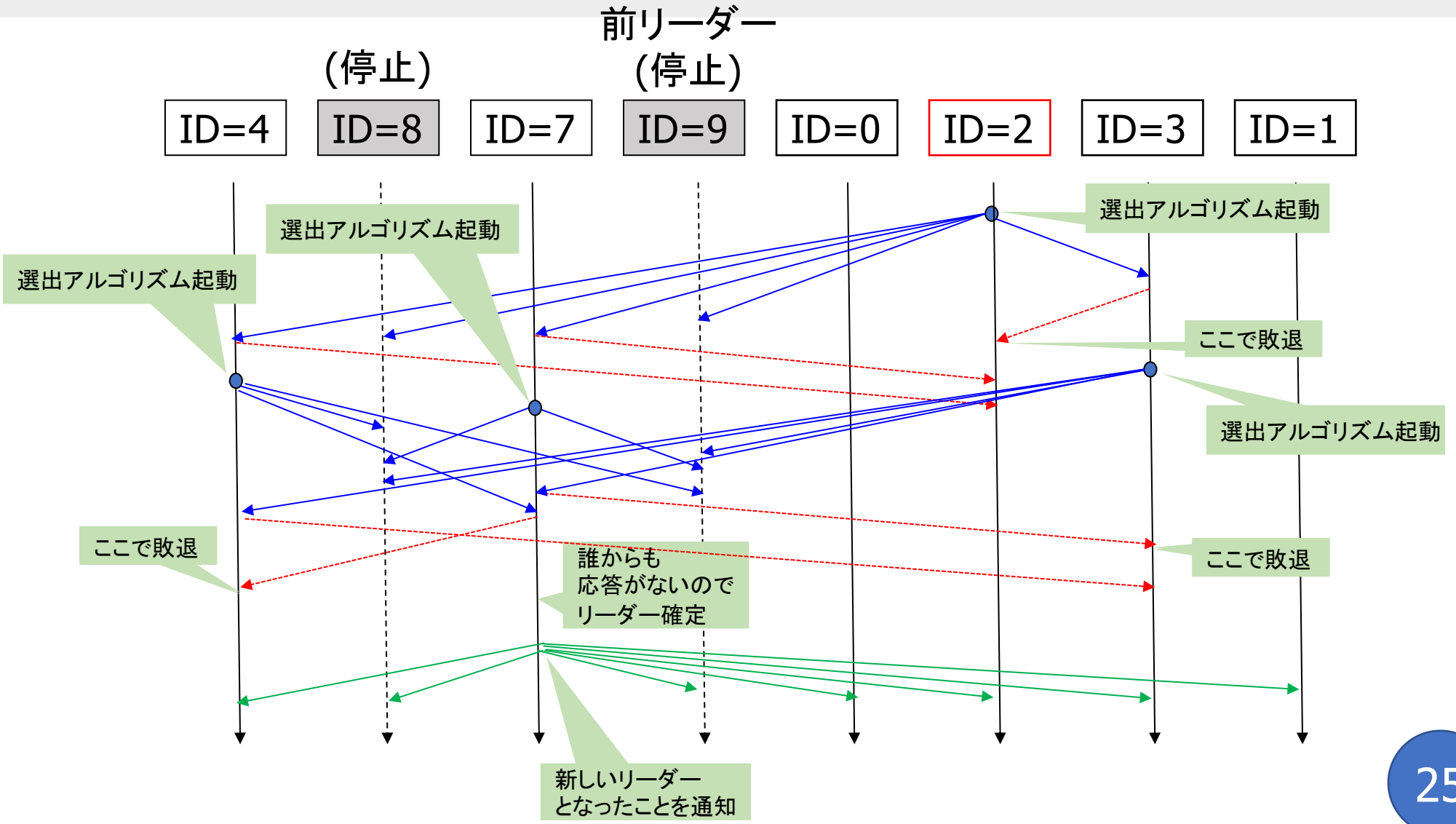
何回かElectionメッセージのやりとりが行われた後、
1つのプロセスを除いて他のプロセスは敗れ去る。

選出後の処理

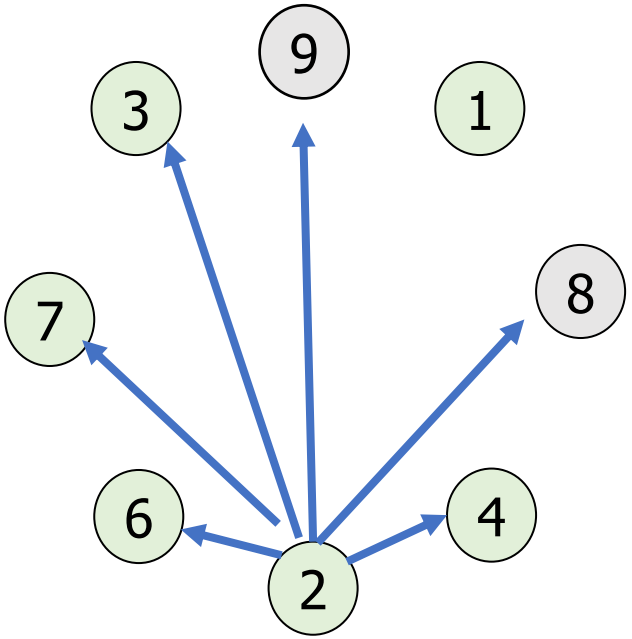
■勝ち残ったプロセスは、他の全てのプロセスに対して
「I won」メッセージを送信することで、
自分が新しいリーダーになったことを知らせる。(選出終了)

■他のプロセスは、「I won」メッセージを受け取ることにより選出が終わったこと　と
新しいリーダーが誰であるか　を知る。(選出終了)

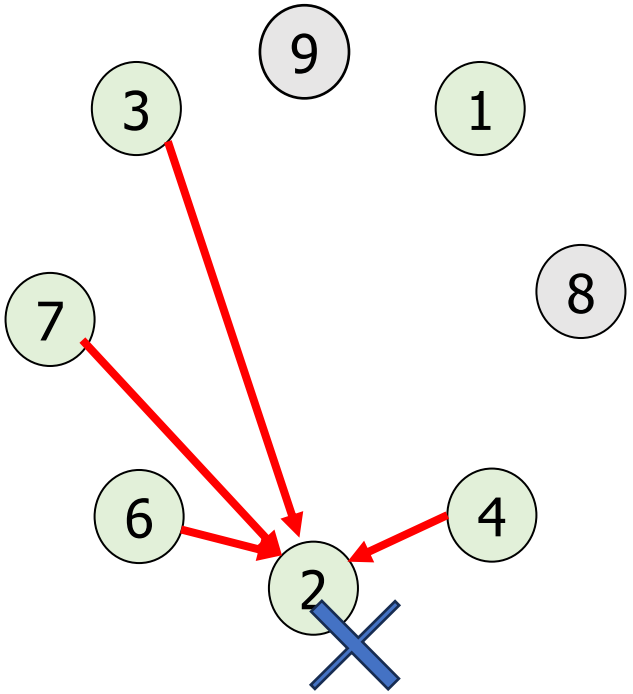
ブリーアルゴリズムの例



step1

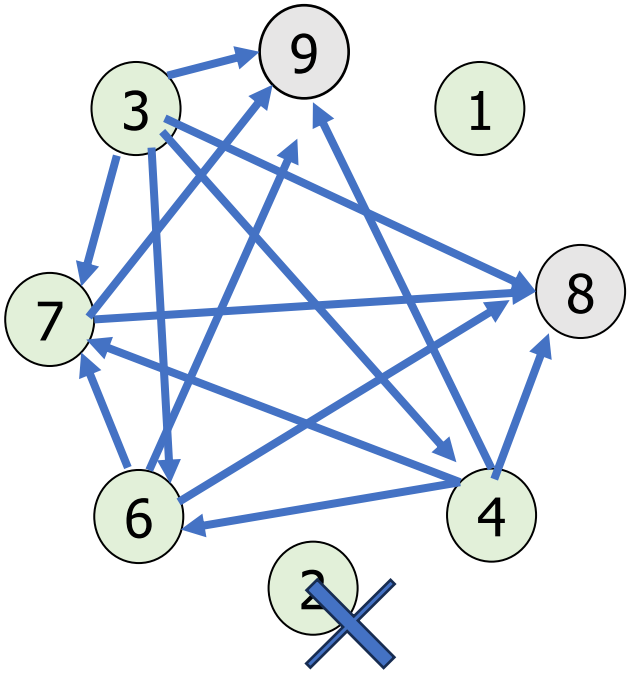


2が始動プロセス
「Election」を送信

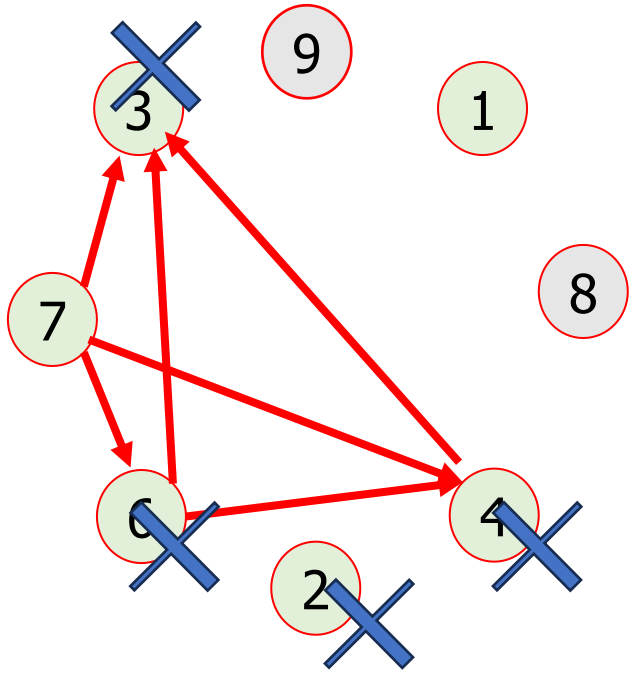


「OK」の送信
返事が来たので、2は敗退

step2

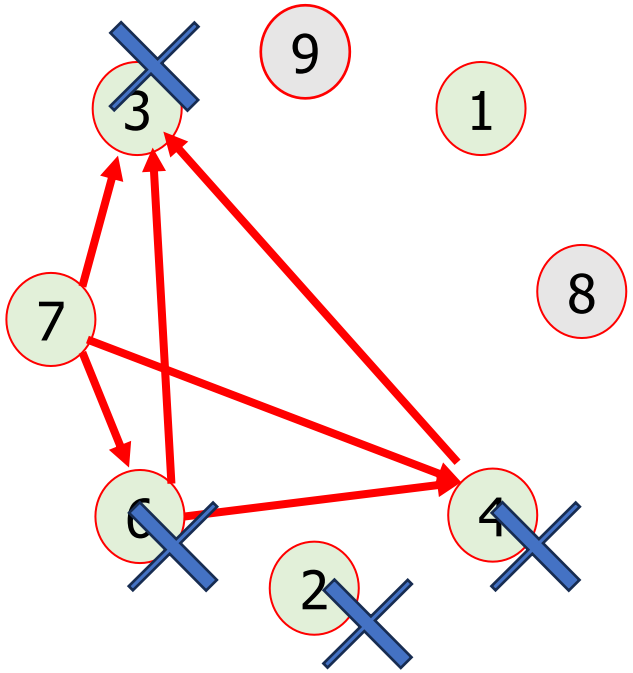


OKを返したプロセスたちが
それぞれ「Election」を送信

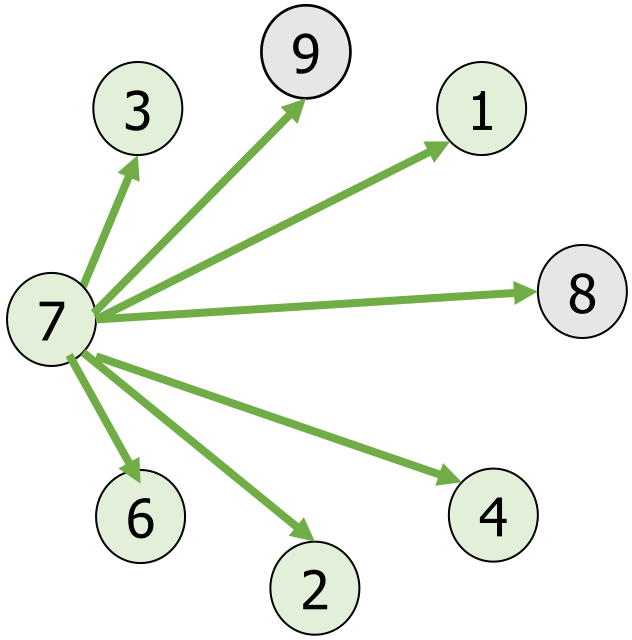


「OK」の送信
返事が来た3, 4, 6 が敗退

step3



7には、一定時間経っても
「OK」メッセージは戻ってこない。
自分がリーダーになったことに気づく。

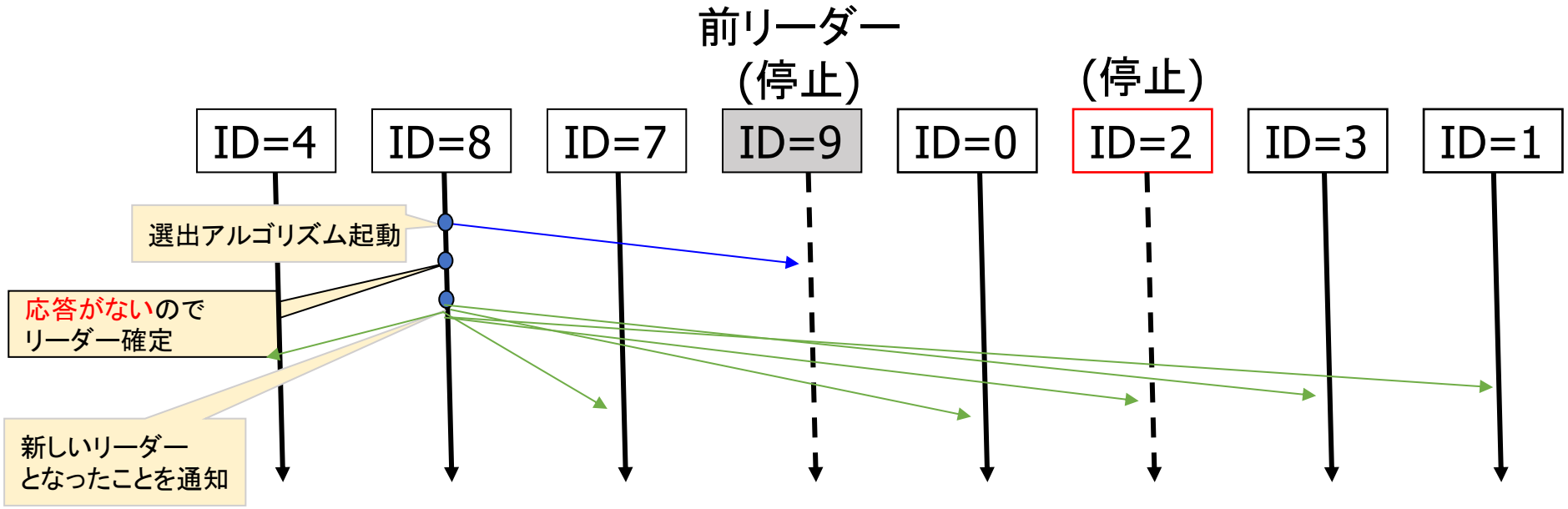


「I won」を送信

ブリーアルゴリズムの解析

最も幸運な場合

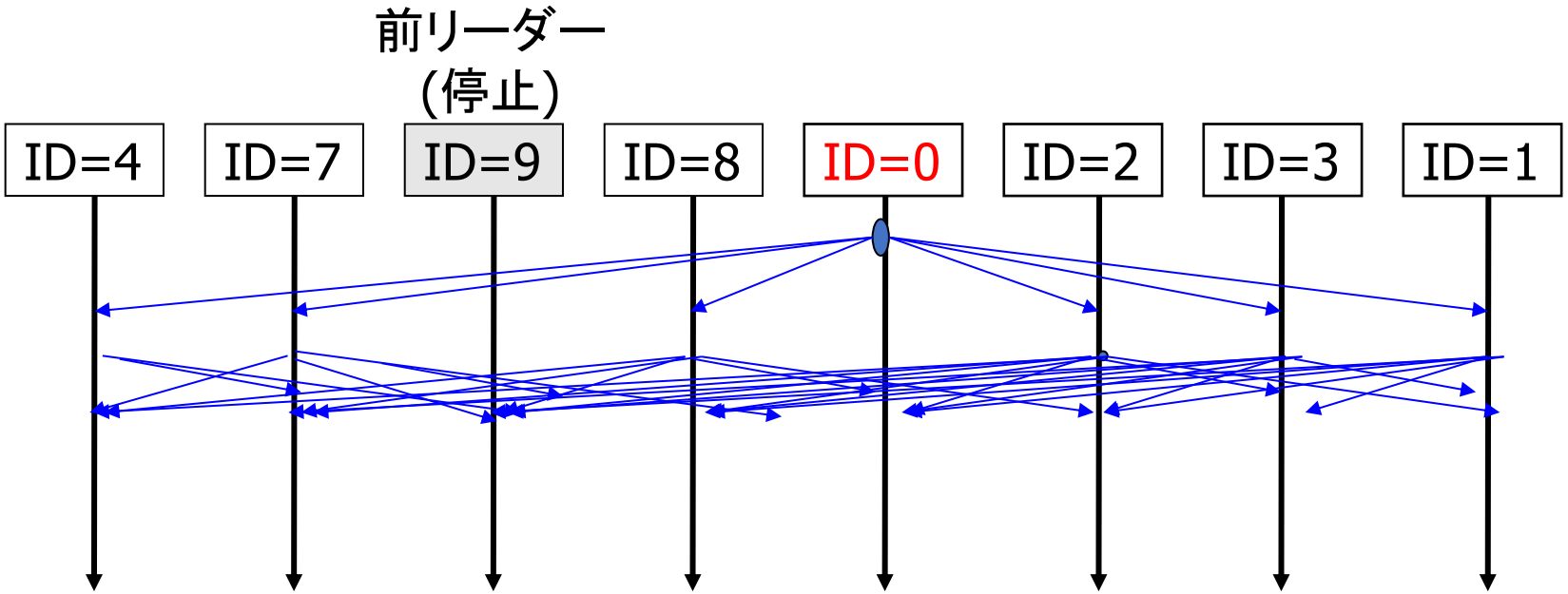
- 故障したリーダーの次に大きいIDをもつプロセスだけが選出アルゴリズムを始動したとき。
- 総プロセス数をnをすると、メッセージ数は 1+ n-2。
  (2:故障したリーダーと自分自身)



ブリーアルゴリズムの解析(つづき)

最悪の場合

- もっとも小さいIDをもつプロセスが選出アルゴリズムを開始したとき。
- メッセージを受信した n-2個のプロセスで、さらに選出アルゴリズムが起動される。
- メッセージの総数は $O(n^2)$



リングアルゴリズム（3種）

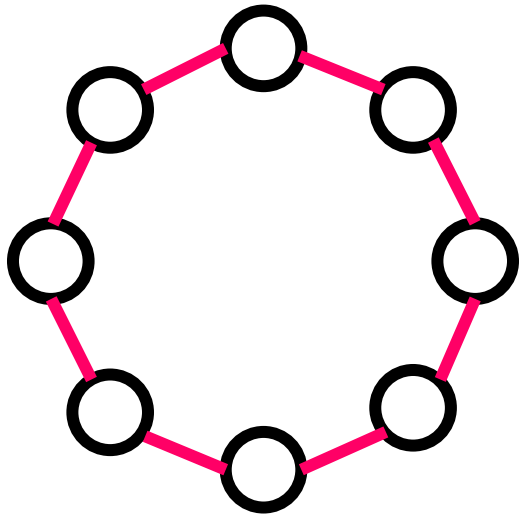
リングネットワークの上でのリーター選出アルゴリズム

- 素朴なアルゴリズム
- Chang-Robertsアルゴリズム
- （Franklinのアルゴリズム）
- Patersonアルゴリズム

リングネットワーク

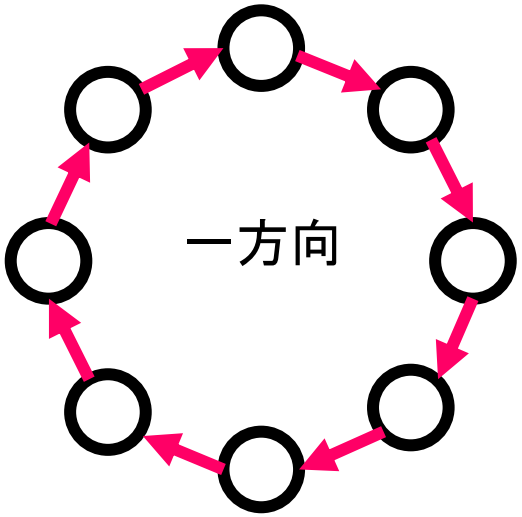
すべてのノードは論理的にリングネットワークを構成する。

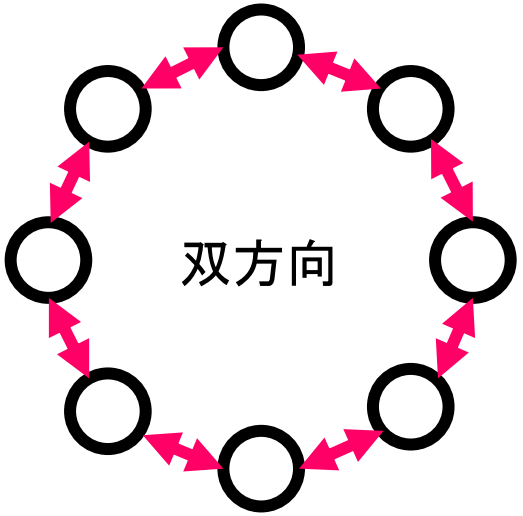
- 各ノードは前後のノードに関する情報のみを知っていれば良い。
  →知っているべき情報が少なくて済む
  →現実に分散システムを構成しやすい。
- リングネットワークに接続されているノードは正常動作
  （停止故障したノードはリングネットワークから外れる）



リングアルゴリズム

- 前提
  - 接続されている順番はIDの順番とは限らない。
  - 全ノードは、メッセージの送出方向を理解している。
    - 一方向に限られているか、双方向に送ることができるか によって、異なる選出アルゴリズムがある。
  - 複数のプロセスがリーダー選出アルゴリズムを始動する 可能性がある。

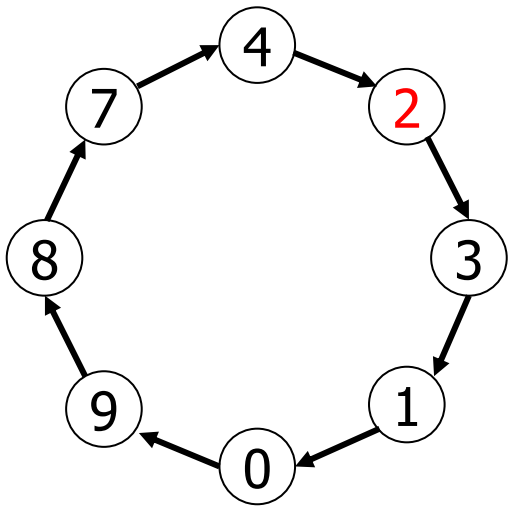




素朴なアルゴリズム（Naive Algorithm）

基本となる考え方

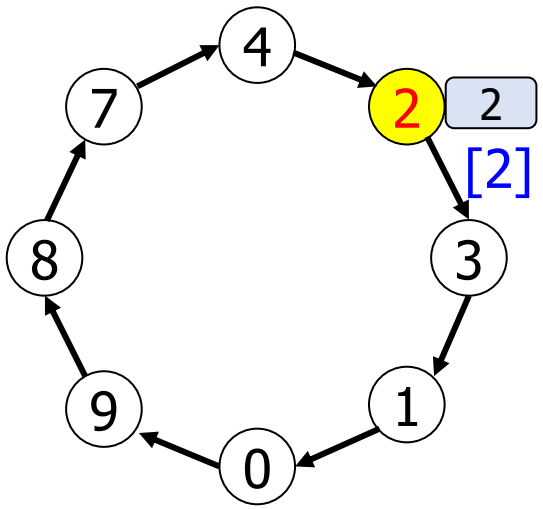
- 各プロセスは、メッセージとして自分のIDを次のプロセスに送信する。
- 各プロセスは、受信したIDを次のプロセスに送信する。
- 自分のIDと同じIDを受信したら、メッセージは一周したことになる。
  それまでに通過したIDの最大値を記録しておけば、
  誰がリーダーかわかる。



この考え方を非活性ノードを含むような
リングネットワークを対象としたものは
LeLann's Algorithmと呼ばれる。

「素朴なアルゴリズム」の選出アルゴリズム

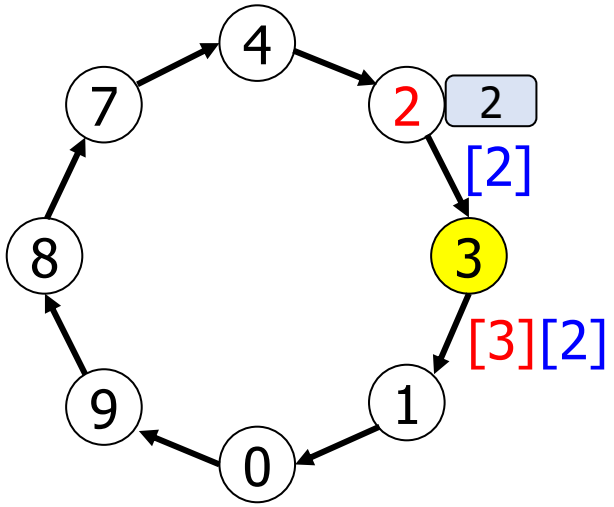
始動プロセスは、
選出アルゴリズム
（election_initiator）
を起動し、
最初に自分のIDを送信する。




```
election_initiator(my_pid)
{
    coordinator = my_pid;
    send(my_pid);
    while(true){
        receive(m);
        if (m==my_pid){
            return coordinator;
        }
        if (m>coordinator){
            coordinator=m;
        }
        send(m);
    }
}
```

「素朴なアルゴリズム」の選出アルゴリズム

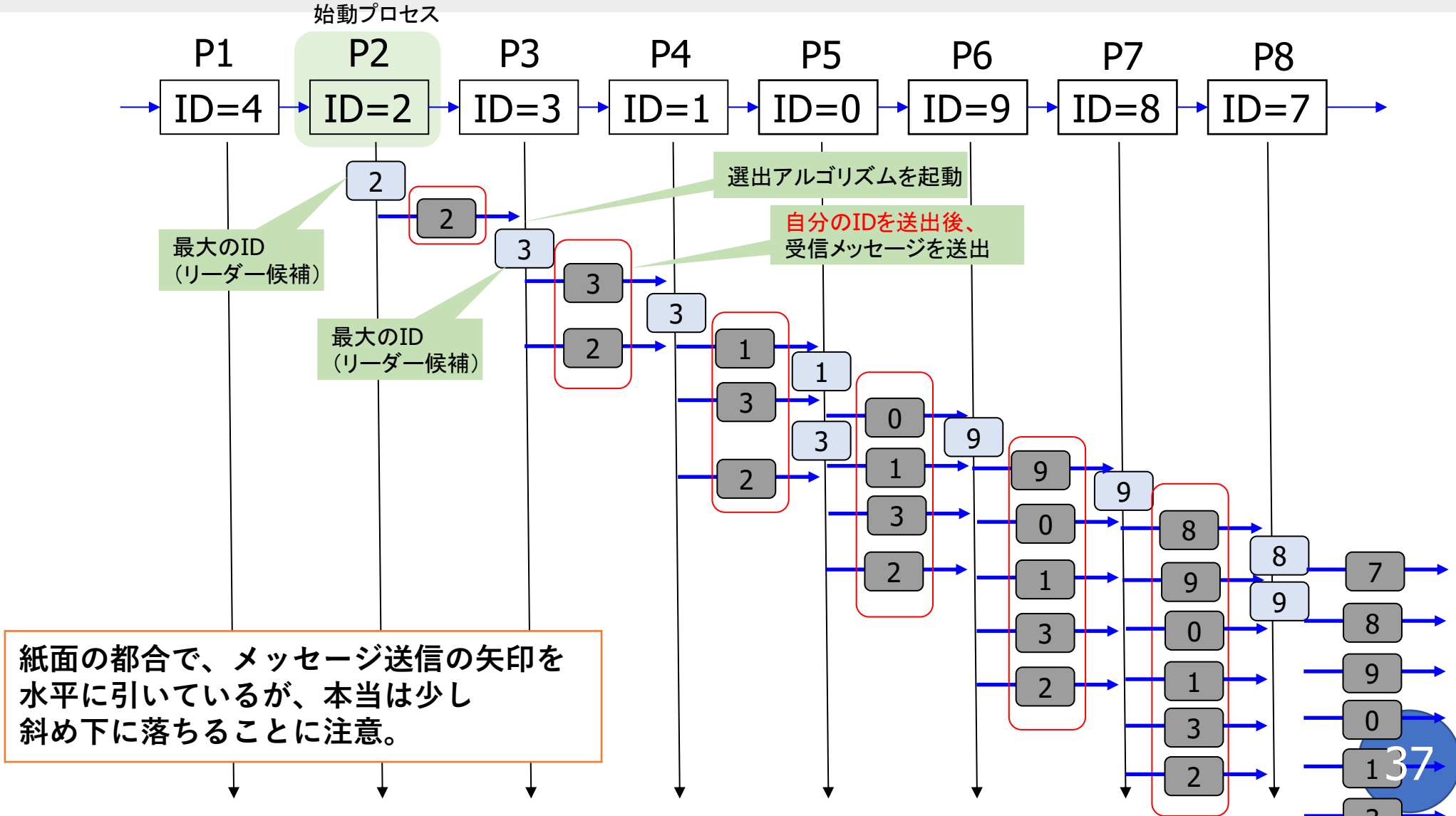
それ以外のプロセスは、
最初のメッセージの
受信を機に
選出アルゴリズム
（election_responder）
を起動する。



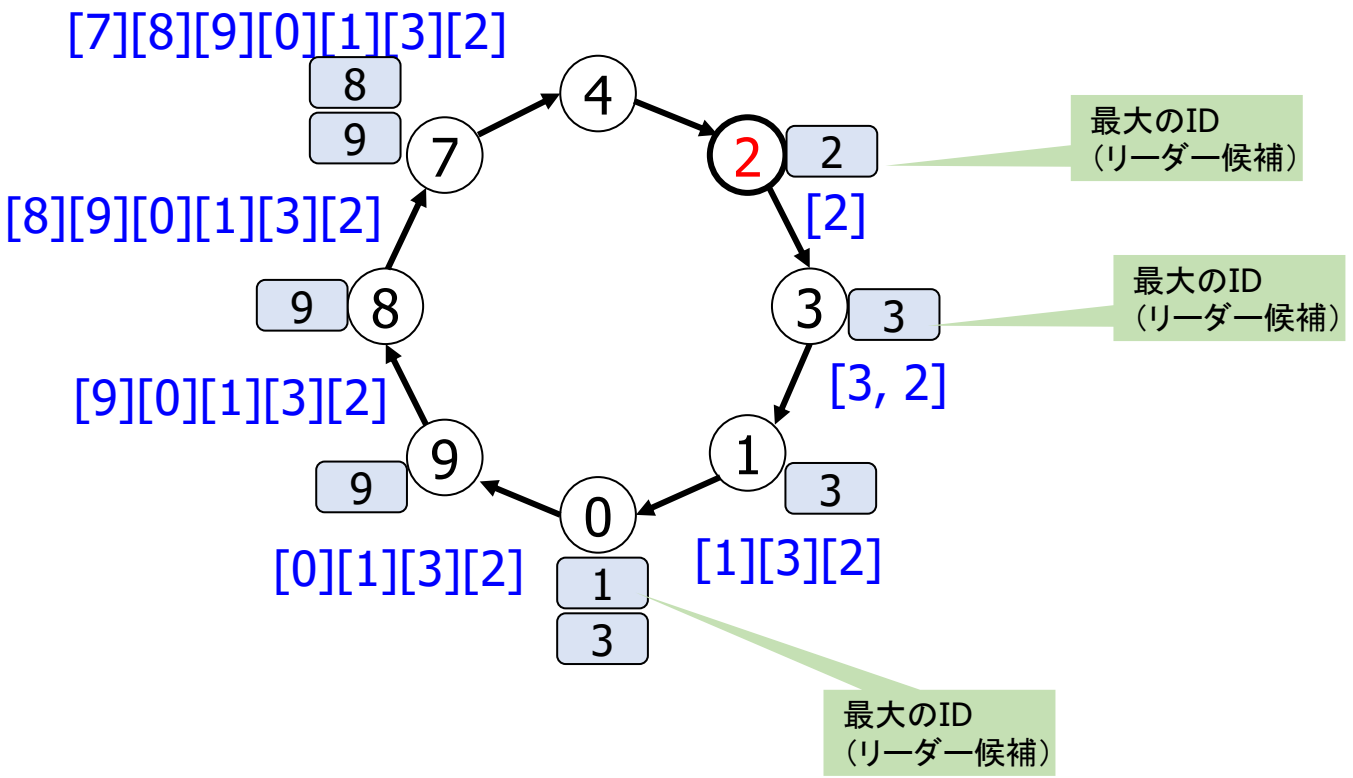

```
election_responder(my_pid)
{
    receive(m);
    coordinator=max(m, my_pid);
    send(my_pid);
    send(m);
    while(true){
        receive(m);
        if (m==my_pid){
            return coordinator;
        }
        if (m>coordinator){
            coordinator=m;
        }
        send(m);
    }
}
```

自分のIDを前に
割り込ませて
送出することに注意

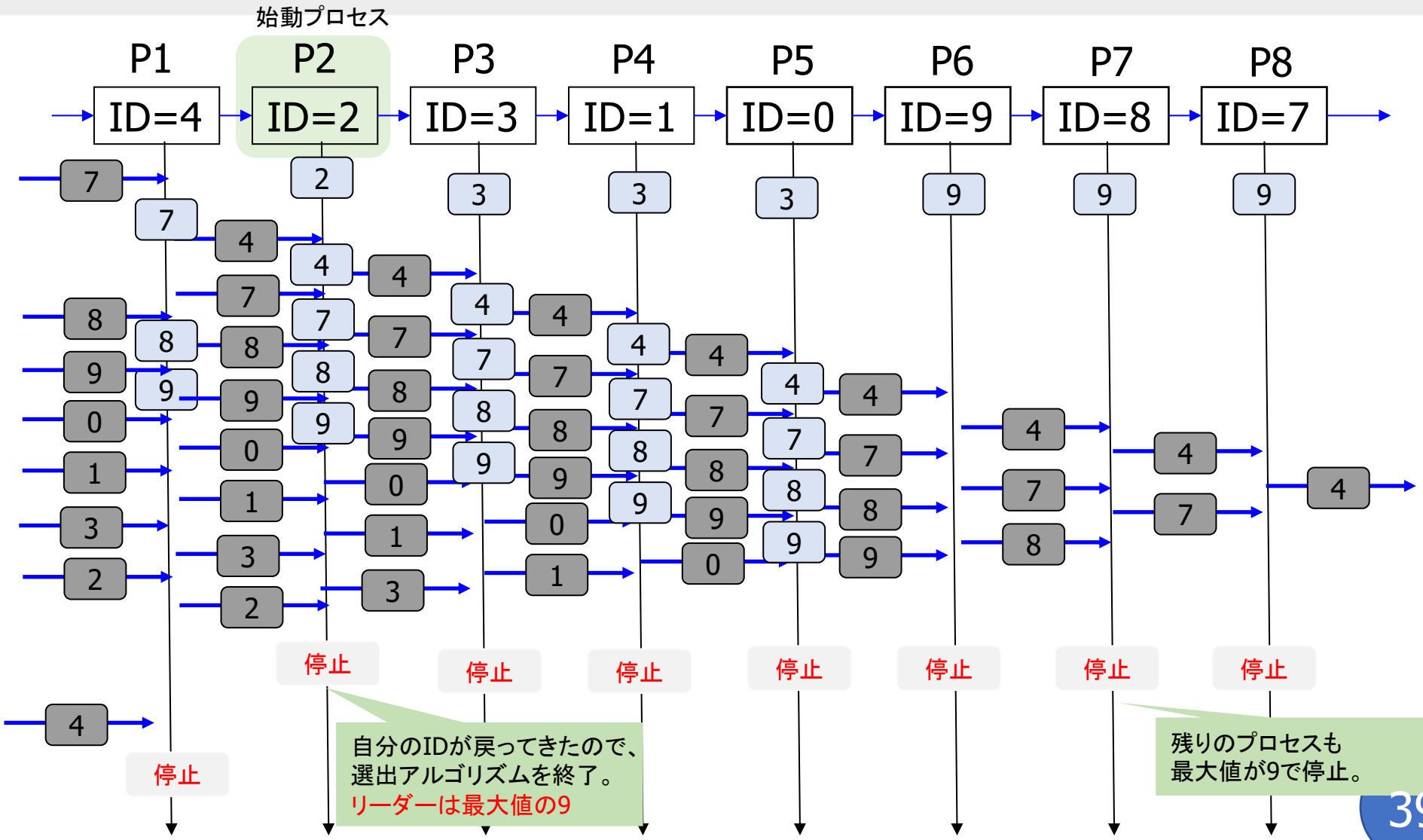
素朴なアルゴリズム（前半）



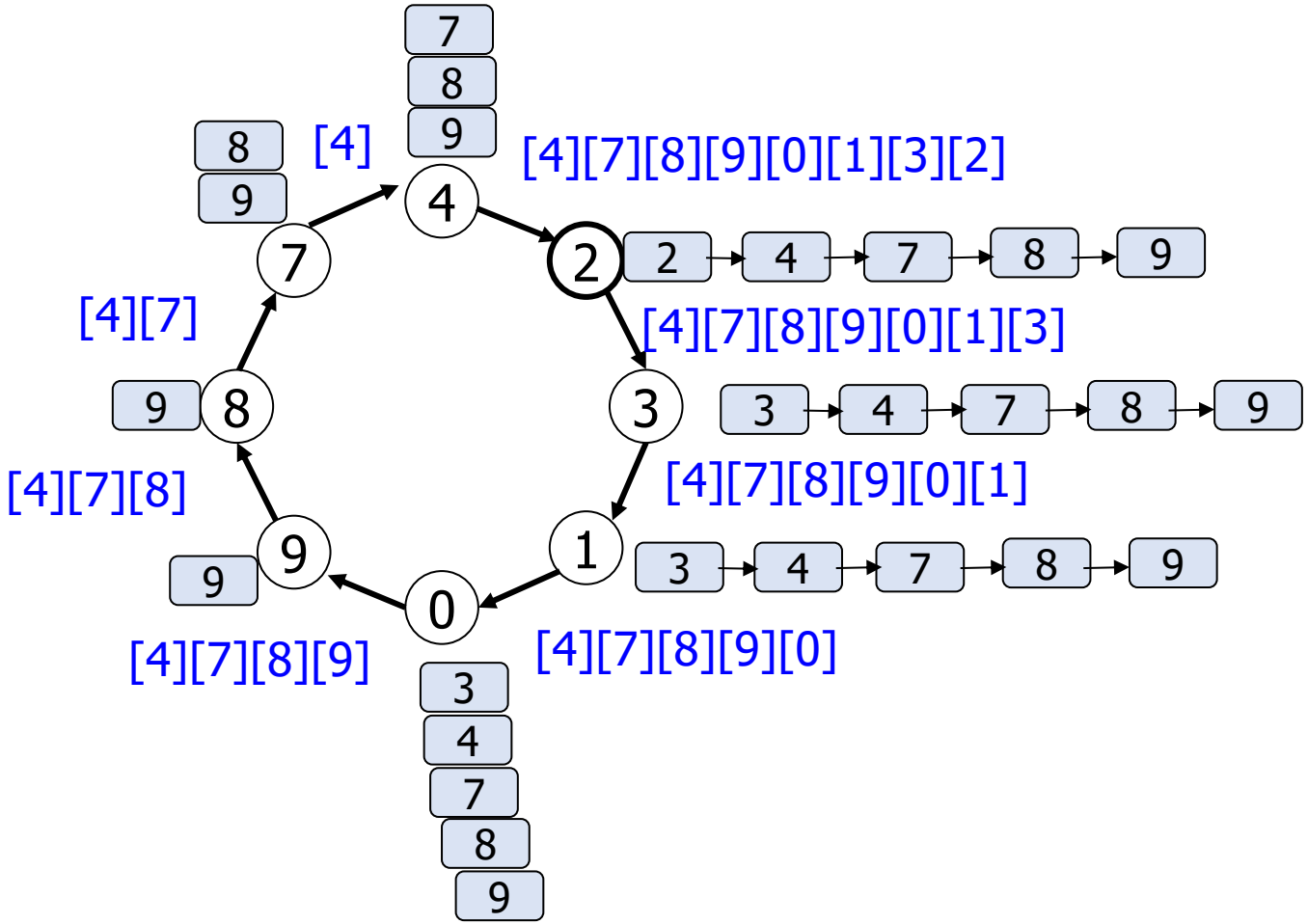
素朴なアルゴリズム（前半）



素朴なアルゴリズム（後半）



素朴なアルゴリズム（後半）



素朴なアルゴリズムの解析

- 「素朴なアルゴリズム」を解析すると
  - 通信ステップ数 O(n)
  - メッセージ総数 $O(n^2)$

    すべての（生きている）ノードが自分のIDを一周させる。
    n メッセージ x 一周nステップ

- 最大値の更新に役立っていないメッセージも転送している。これは無駄ではないだろうか。
  →改良の余地がありそう？

Chang-Robertsアルゴリズム

方針

- 基本的には「素朴なアルゴリズム」と同様にIDを転送する。
- ただし、自分が保持している（現時点での）最大値よりも小さいIDを受信したときには転送しない。
- これにより、小さいIDは途中で止まり、一番大きいIDだけが一周する。

これにより、メッセージ数の削減が期待できるはず。

Chang-Robertsアルゴリズム

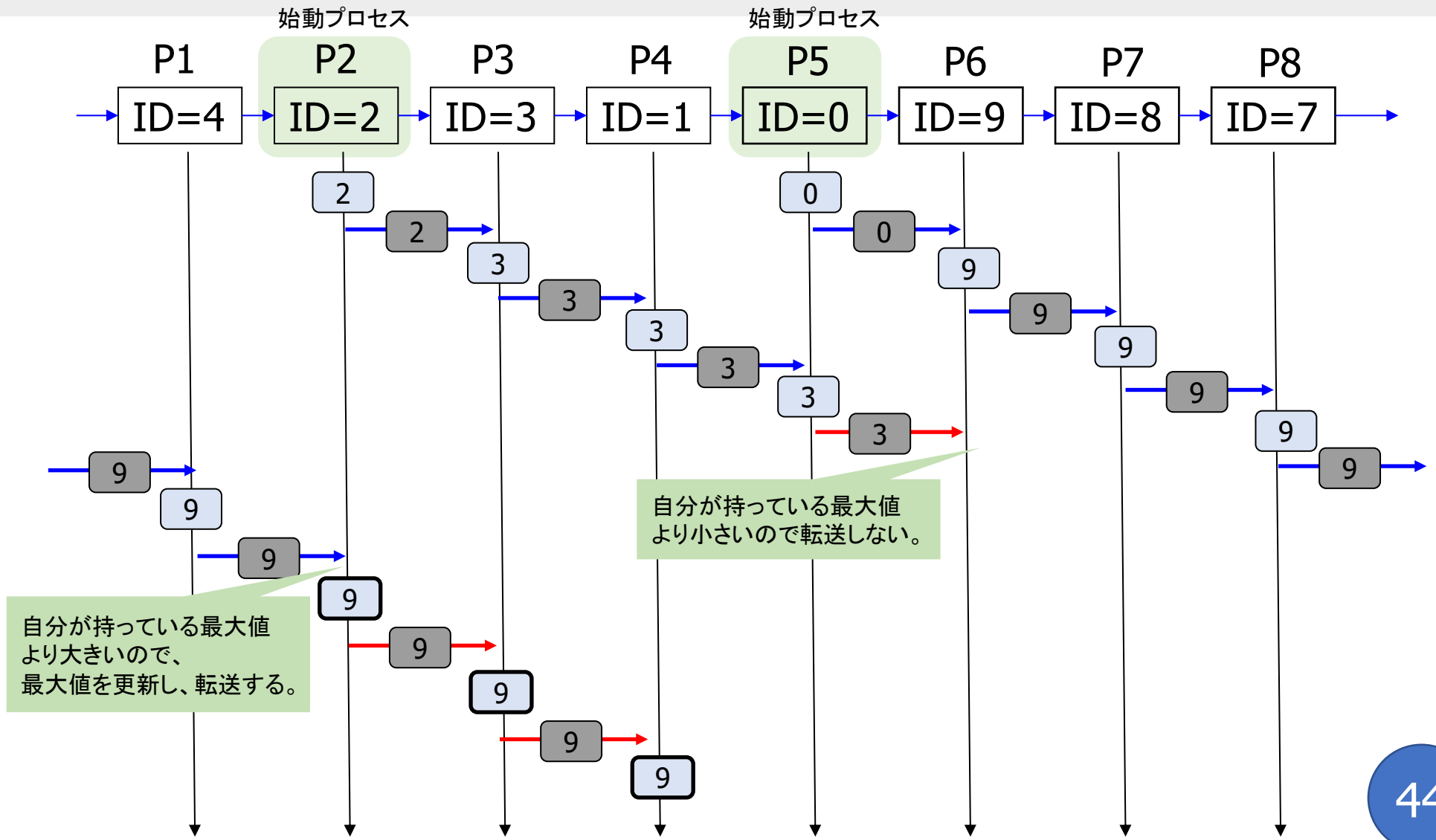
■「自分のIDを受信した」プロセスは
　自分のIDが一周した → IDが一番大きい
ということで、**自分がリーダーになったことがわかる。**

■ところで、小さいIDを持つプロセスのところでは
自分のIDが戻ってこないため、
選出アルゴリズムの終了したのか判断できない。
（誰がリーダーになったのかわからない）

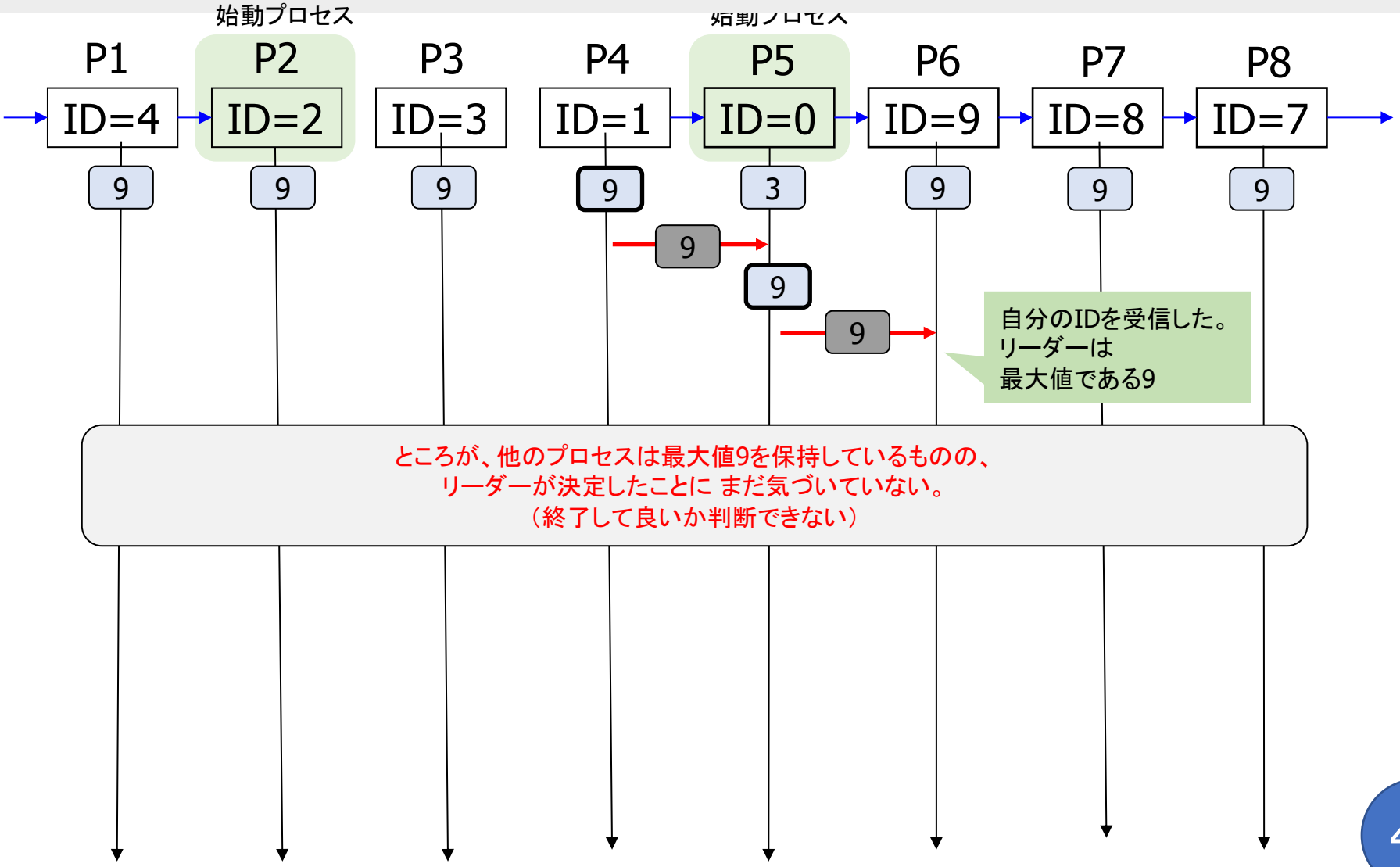
■そこで、新しいリーダーは
「リーダー選出が終わったことを知らせるメッセージ」
を送信する必要がある。

■メッセージとして Election と Elected の2種類を使う。

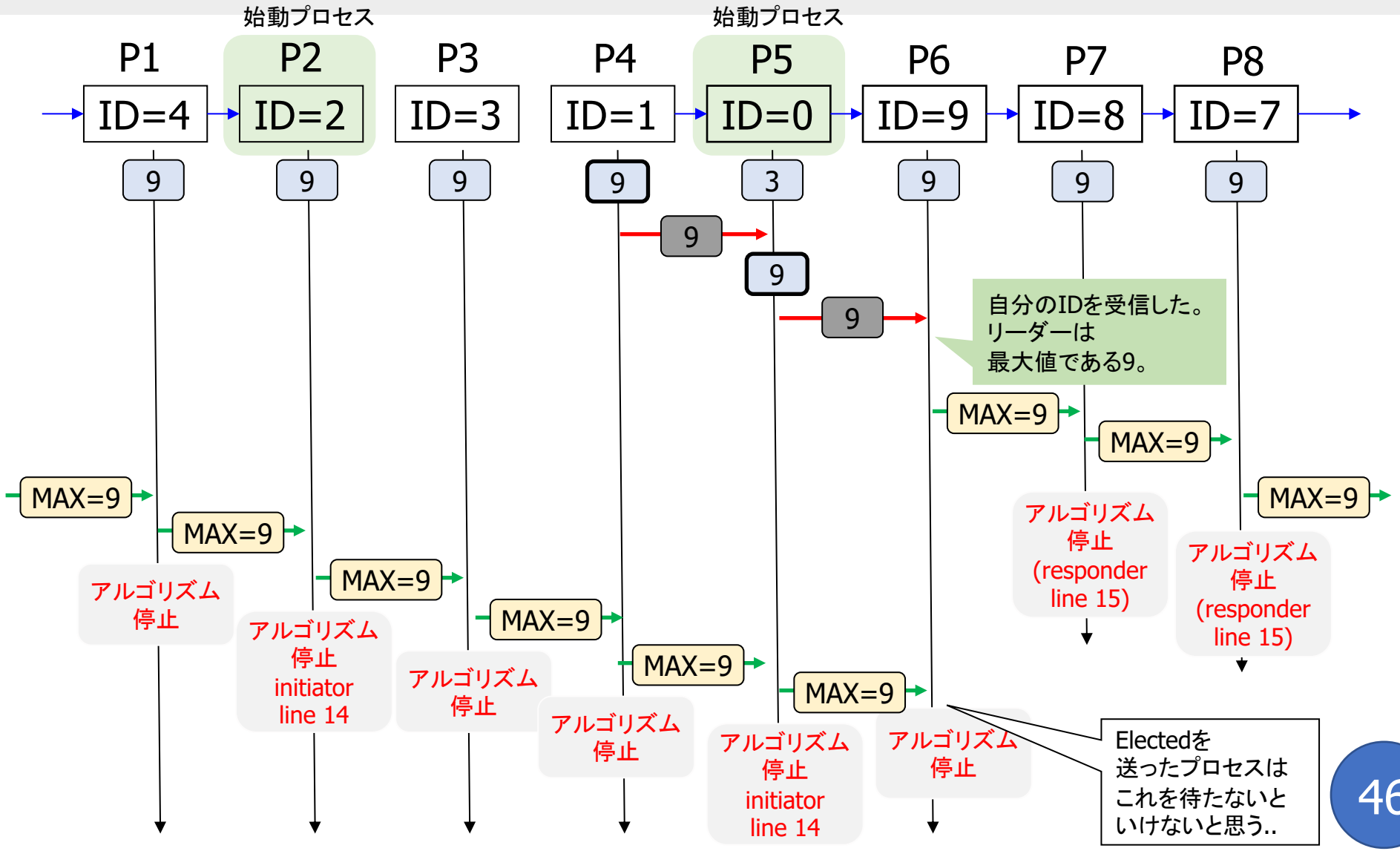
Chang-Robertsアルゴリズム



Chang-Robertsアルゴリズム（つづき）



Chang-Robertsアルゴリズム（つづき）



Chang-Robertsアルゴリズム（日本語表記）

■**始動プロセス**：

1. 自分のIDを最大値として保存する。
2. その最大値を隣のプロセスにElectionメッセージとして送る。

3. 反対側のプロセスからメッセージを待つ。

4. 受け取ったElectionメッセージのIDが自分のIDと一致した場合、
   自分がリーダーである。
   さらにElectedメッセージを送出し、アルゴリズム終了。

5. 受け取ったのがElectedメッセージであれば、
   隣にメッセージを送信し、アルゴリズムを終了。

6. 受け取ったElectionメッセージに含まれるID が保持する最大値より
   大きければ、そのIDを最大値として保存する。
   隣にメッセージを送信し、ステップ3へ。

Chang-Robertsアルゴリズム（日本語表記）

■**始動プロセス以外のプロセス**：

■前のプロセスからのメッセージ受信を機に
選出アルゴリズムを起動。

1. 受信したIDと自分のIDの大きい方を最大値として保存する。
2. その最大値を隣のプロセスにElectionメッセージとして送る。
3. 前のプロセスからのメッセージを待ち、受け取る。
4. 受け取ったElectionメッセージのIDが自分のIDと一致した場合、
自分がリーダーである。
さらにElectedメッセージを送出し、アルゴリズム終了。
5. 受け取ったのがElectedメッセージであれば、
メッセージを次に送信し、アルゴリズムは終了。
6. 受け取ったのがElection メッセージで、
そこに含まれるIDが自分が保存しているメッセージより
大きければ、そのIDを最大値として保存し、
メッセージを次に送信する。ステップ3へ。

Chang-Robertsアルゴリズム（疑似コード）

```
election_initiator(my_pid)
{
    coordinator = my_pid;
    send(election,coordinator);
    while(true){
        receive(mode,m);
        if (mode==election && m==my_pid){
            send(elected, coordinator);
            return coordinator;
        }
        if (mode==elected)
            coordinator=m;
            send(elected, coordinator);
            return coordinator;
        }
        if (m>coordinator){
            coordinator=m;
            send(election, coordinator);
        }
    }
}
```

```
election_responder (my_pid)
{
    receive(mode,m);
    coordinator=max(m, my_pid);
    send(election, coordinator);
    while(true){
        receive(mode,m);
        if (mode==election && m==my_pid){
            send(elected, coordinator);
            return coordinator;
        }
        if (mode==elected)
            coordinator=m;
            send(elected, coordinator);
            return coordinator;
        }
        if (m>coordinator){
            coordinator=m;
            send(election, coordinator);
        }
    }
}
```

Chang-Robertsアルゴリズムの解析

通信ステップ数：$O(n)$
　nステップでリーダーが決定できる。
　リーダーに選出されたプロセスが自分のIDを
　他プロセスに知らせるためのElectedメッセージの送信のために
　さらにnステップかかる。

メッセージ総数：

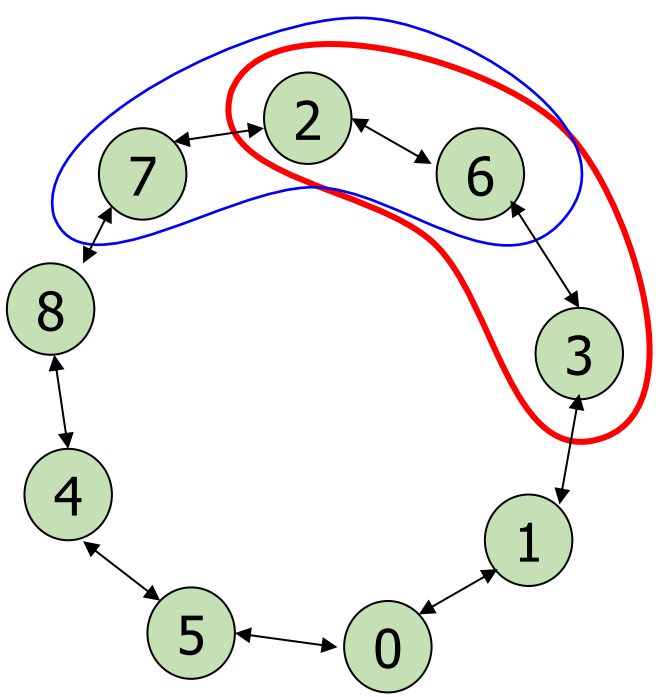
- 平均では　$O(n \log n)$　（導き方むずかしいそうです）
- 最悪の場合：$O(n^2)$
  リングネットワークに並ぶIDの順番による。
  各通信ステップで送られるメッセージ数が$O(n)$で
  これをn回繰り返すことになる。

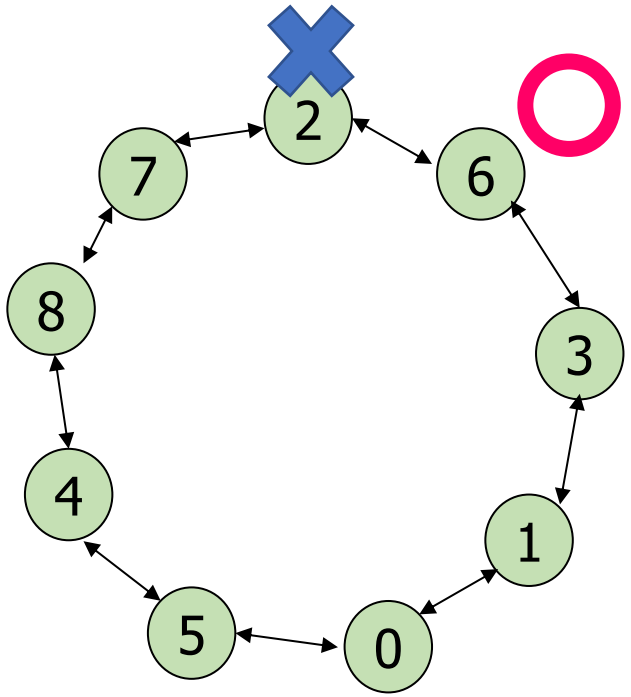
  減ってない！
  メッセージ数が$O(n^2)$ より少ないアルゴリズムはないのか？

Patersonのアルゴリズム
その前に：Franklinのアルゴリズム

基本的な考え方

- 両隣とIDをメッセージ交換し、自分が最大値でないとわかったプロセスが候補者リングから外れていく。

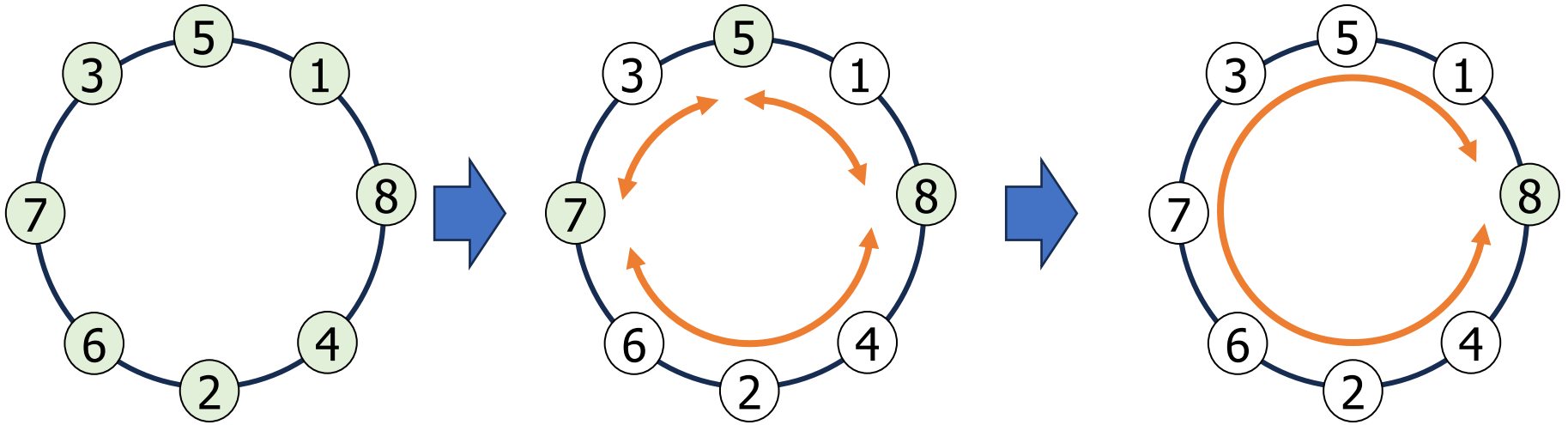




Franklinのアルゴリズム

基本的な考え方

- 両隣とIDをメッセージ交換し、自分が最大値でないとわかったプロセスが候補者リングから外れていく。
- 最後に残ったプロセスがリーダーである。

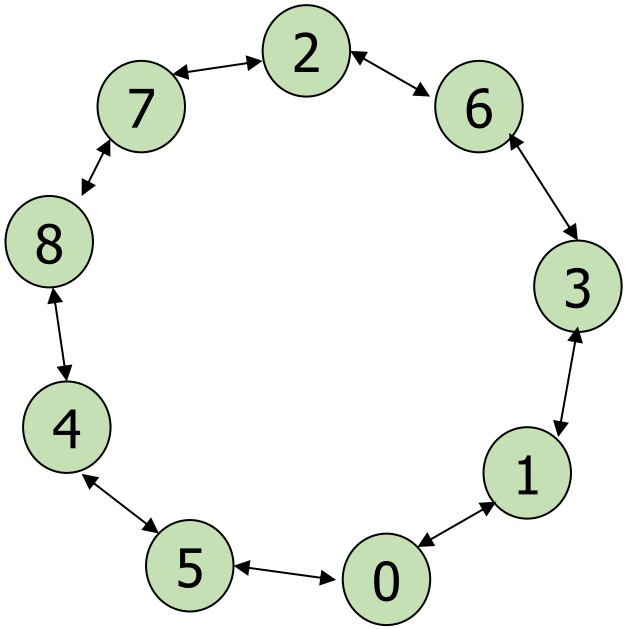


Franklinのアルゴリズム

■動作中のプロセスは Active と Passive のいずれかの状態を保つ。
- ■Active ：自分の値が最大値である可能性がある状態
- ■Passive ：自分が最大値でないことが明らかな状態

■Passiveなノードは、値を転送するだけの動き。

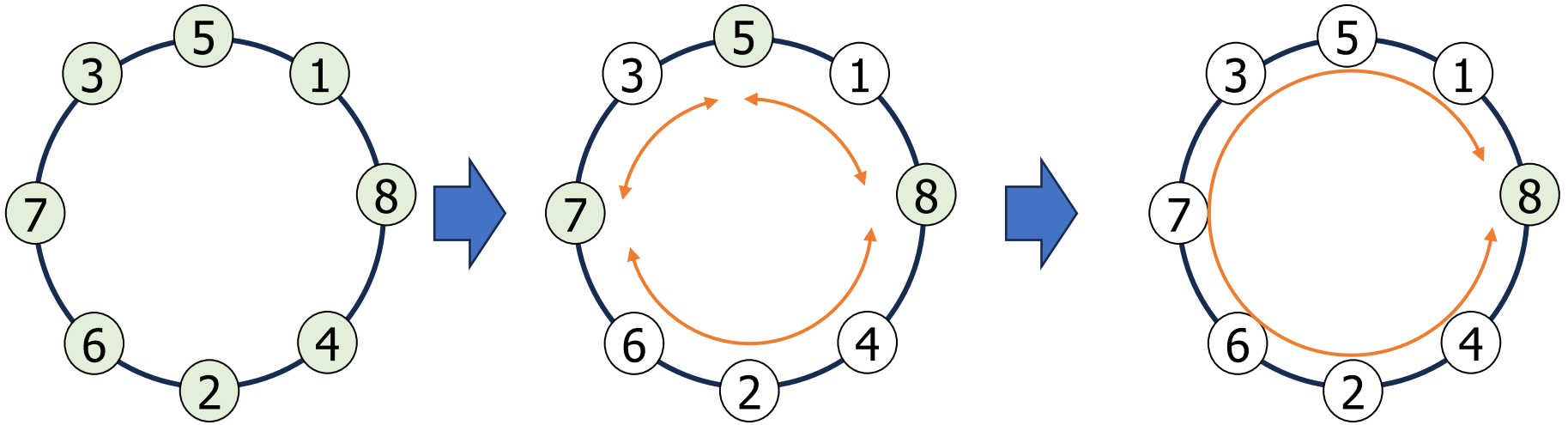
■Activeなノードは自分の両隣に値を送る。
- ■もし「隣」から受け取った値が
  自分の値だったら、
  残っているのは自分だけということになる。
  →リーダーは自分。



■Passiveなノードは
リーダー選出が終了したかどうか判断できない。
最後にリーダー選出が終了したことを
知らせるメッセージを一周させる必要がある。

Franklinのアルゴリズムの解析

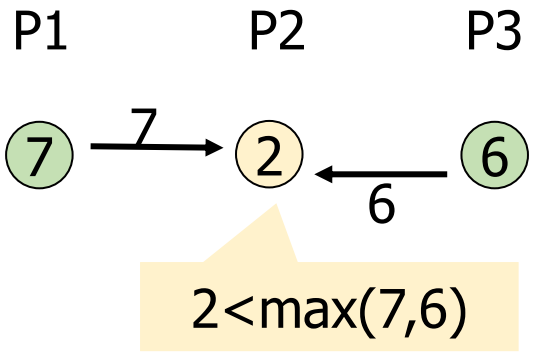
- メッセージ数：
  各ノードが両隣にメッセージを送信=2n
  次以降のステップ：転送を考えると同じく 2nメッセージ
- 時間：O(n)



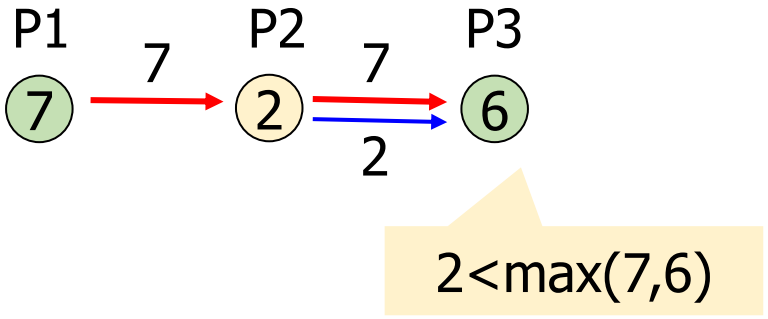
2x8=16メッセージ　2x8=16メッセージ　2x8=16メッセージ

Patersonのアルゴリズム (Paterson's 1st Algorithm)

- Franklinのアルゴリズムを、単方向リングネットワークで実現するようにしたもの。
- 基本的な考え方：
  一つ隣のActiveノードと、隣の隣のActiveノードのIDを受信して、Franklinアルゴリズムを適用する。



Franklinアルゴリズムでの判定



Patersonのアルゴリズムでの判定

ズレるので すごく混乱する！

Patersonのアルゴリズム

- 各ノードは4つの変数を管理する。

- state = { candidate, relay, leader }
- tid: temporary identity
- ntid: first id received （隣）
- nntid: second id received (隣の隣)

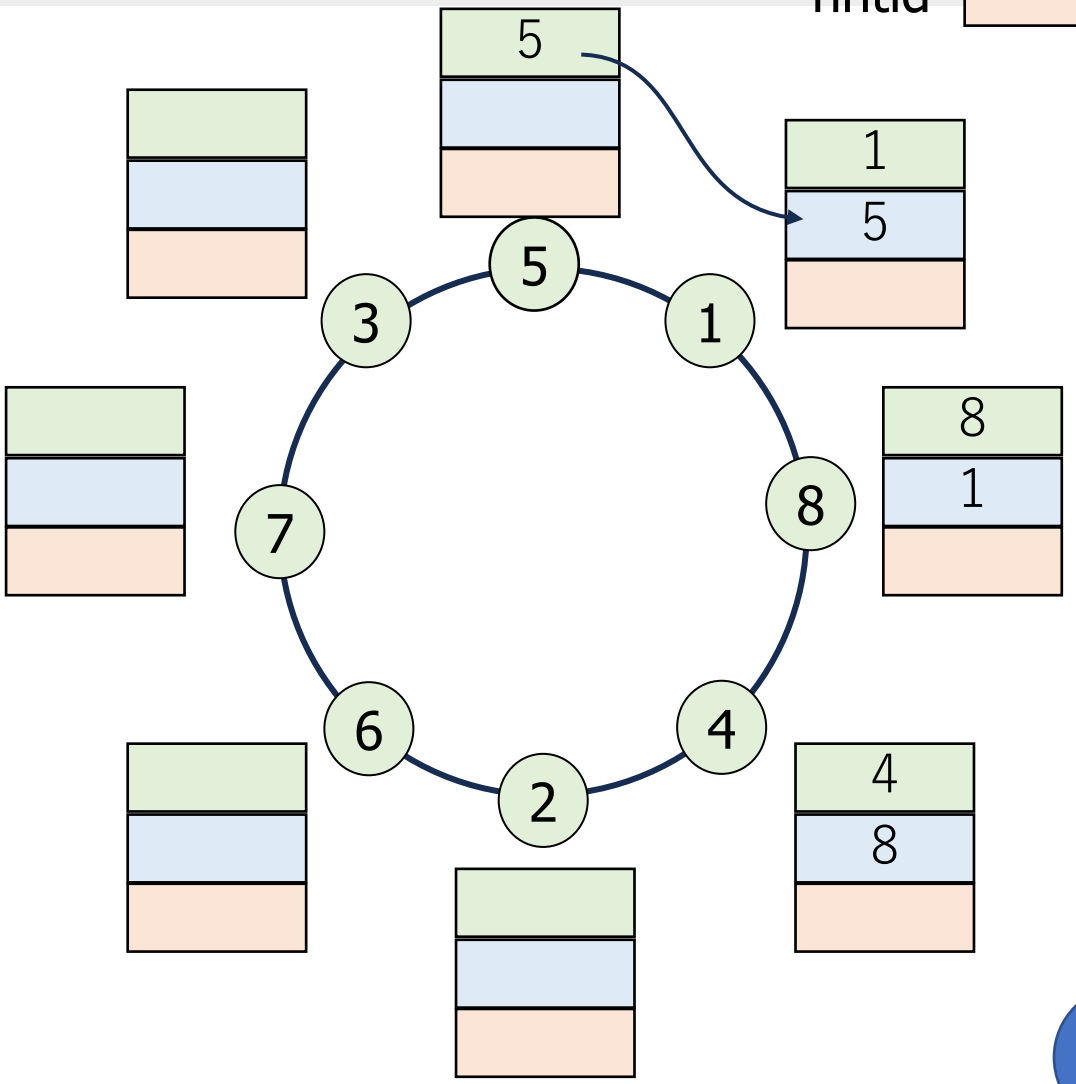
Patersonのアルゴリズム（疑似コード）

```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid== my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
              else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```

```
//now state=relay
while (state!=leader)
{
  receive(tid);
  if (tid==my_id) {state=leader;}
  send(tid);
}
```

(1a)

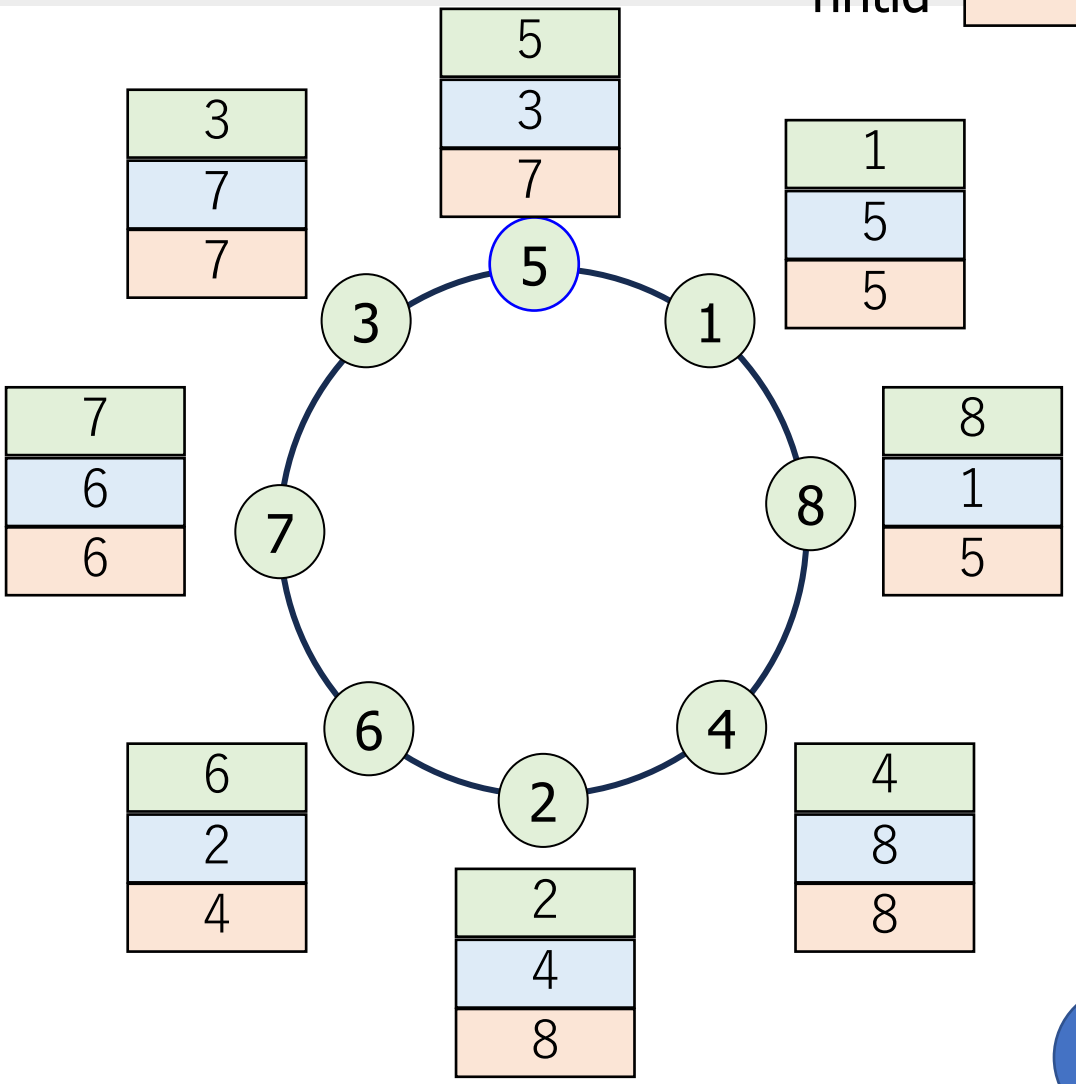
```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
        :
```



(1b)

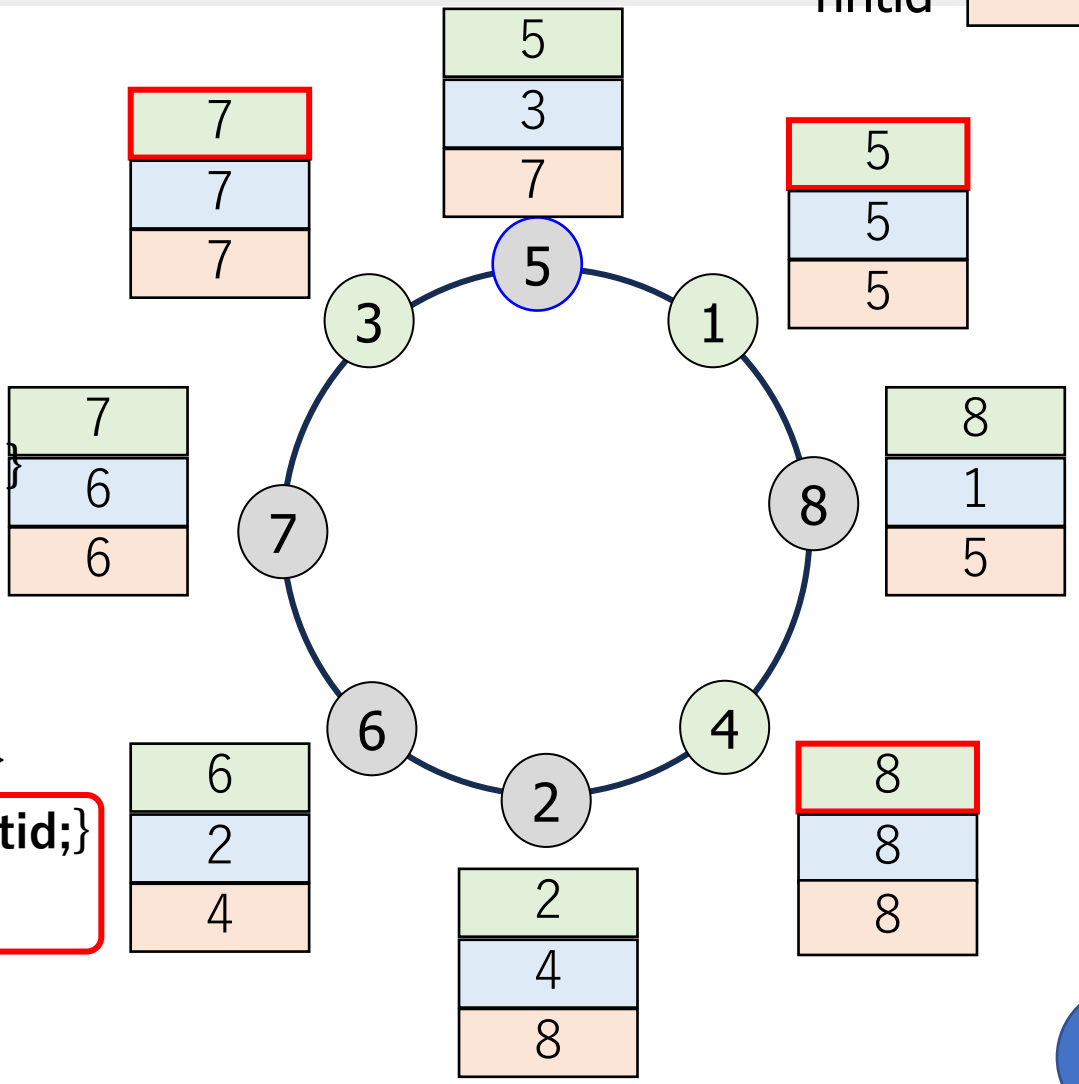
```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid== my_id) { sate=leader; }
```

```
if (tid>ntid) { send(tid);}
        else {send(ntid);}
receive(nntid);
```



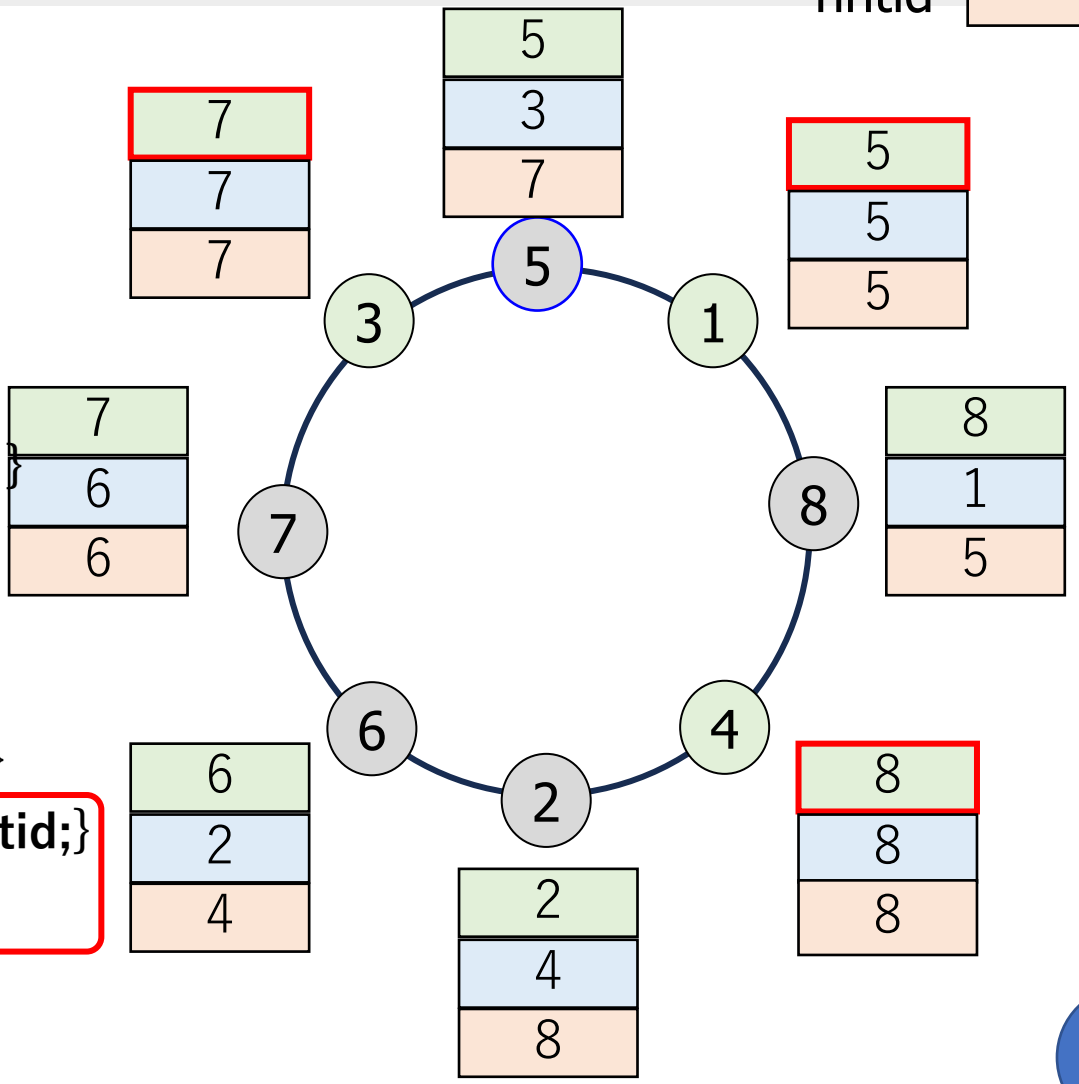
(1c)

```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid==my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
            else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```



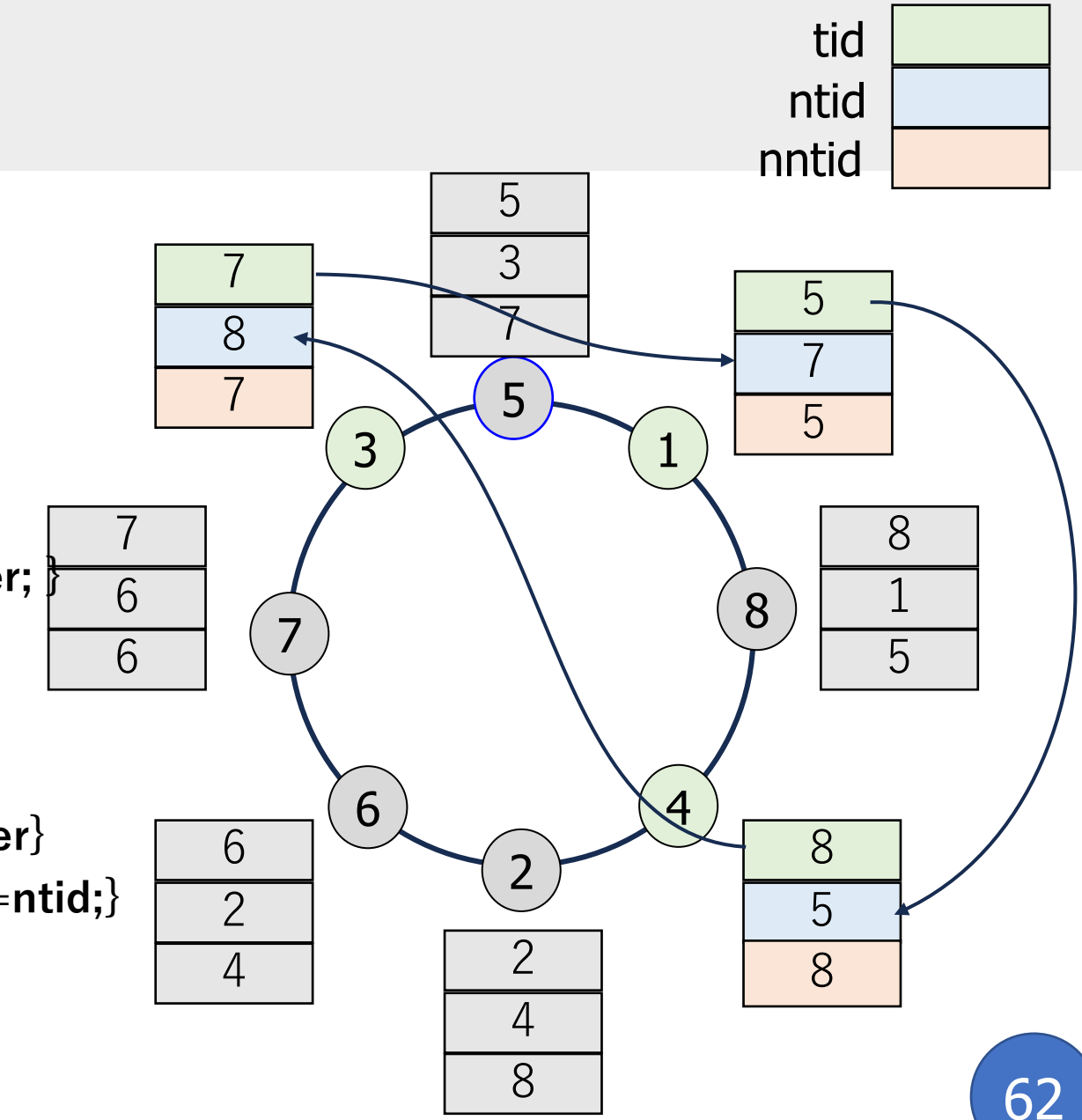
(1c)

```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid==my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
            else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```



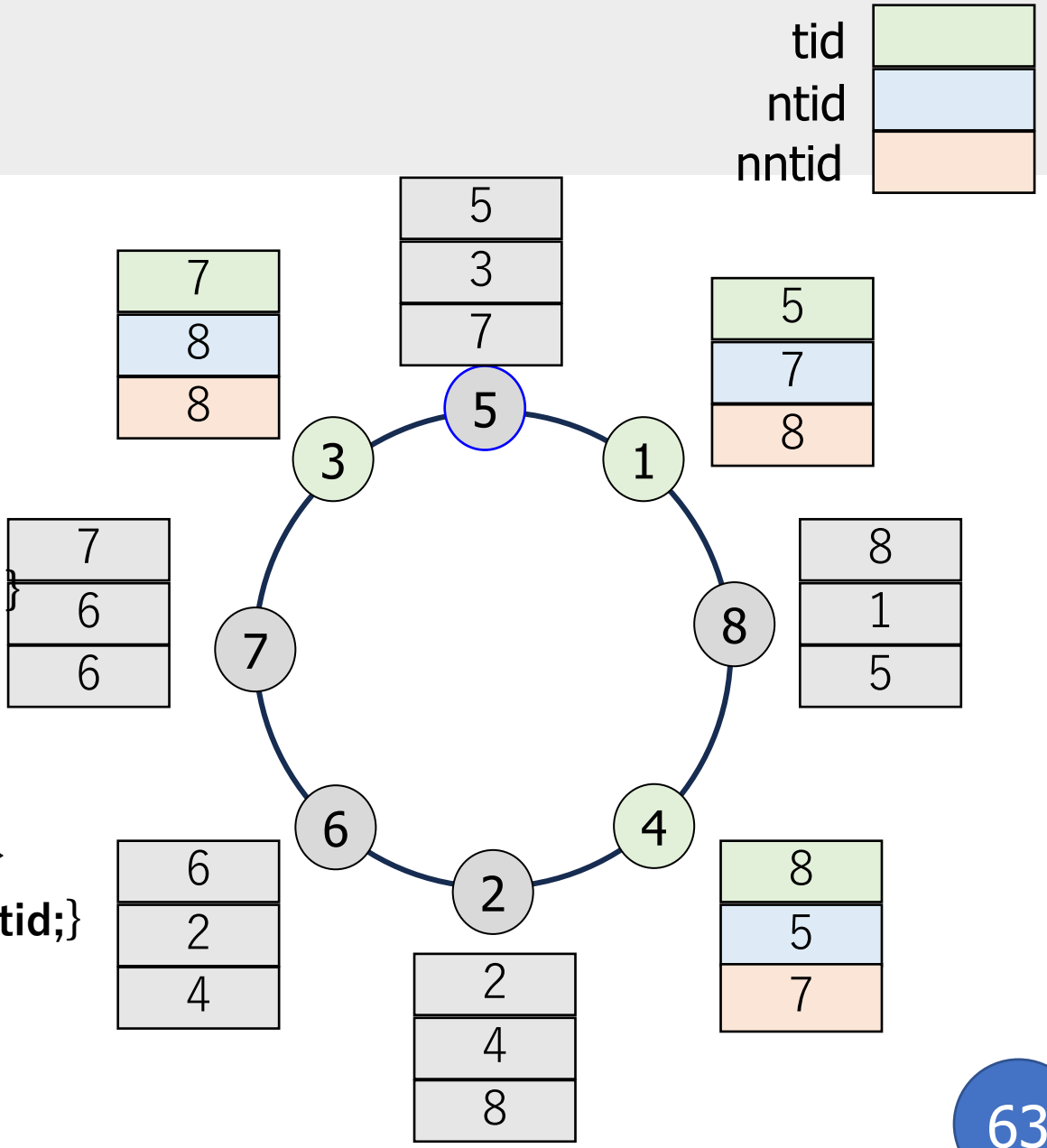
(2a)

```
state=candidate;
tid=my_id;
while (state!=relay) {
   send(tid);
   receive(ntid)
   if (ntid==my_id) { sate=leader; }
   if (tid>ntid) { send(tid);}
            else {send(ntid);}
   receive(nntid);
   if (nntid==myid){ state=leader}
   if (ntid>=max(tid, nntid) {tid=ntid;}
   else state=relay;
}
//now state=relay
```



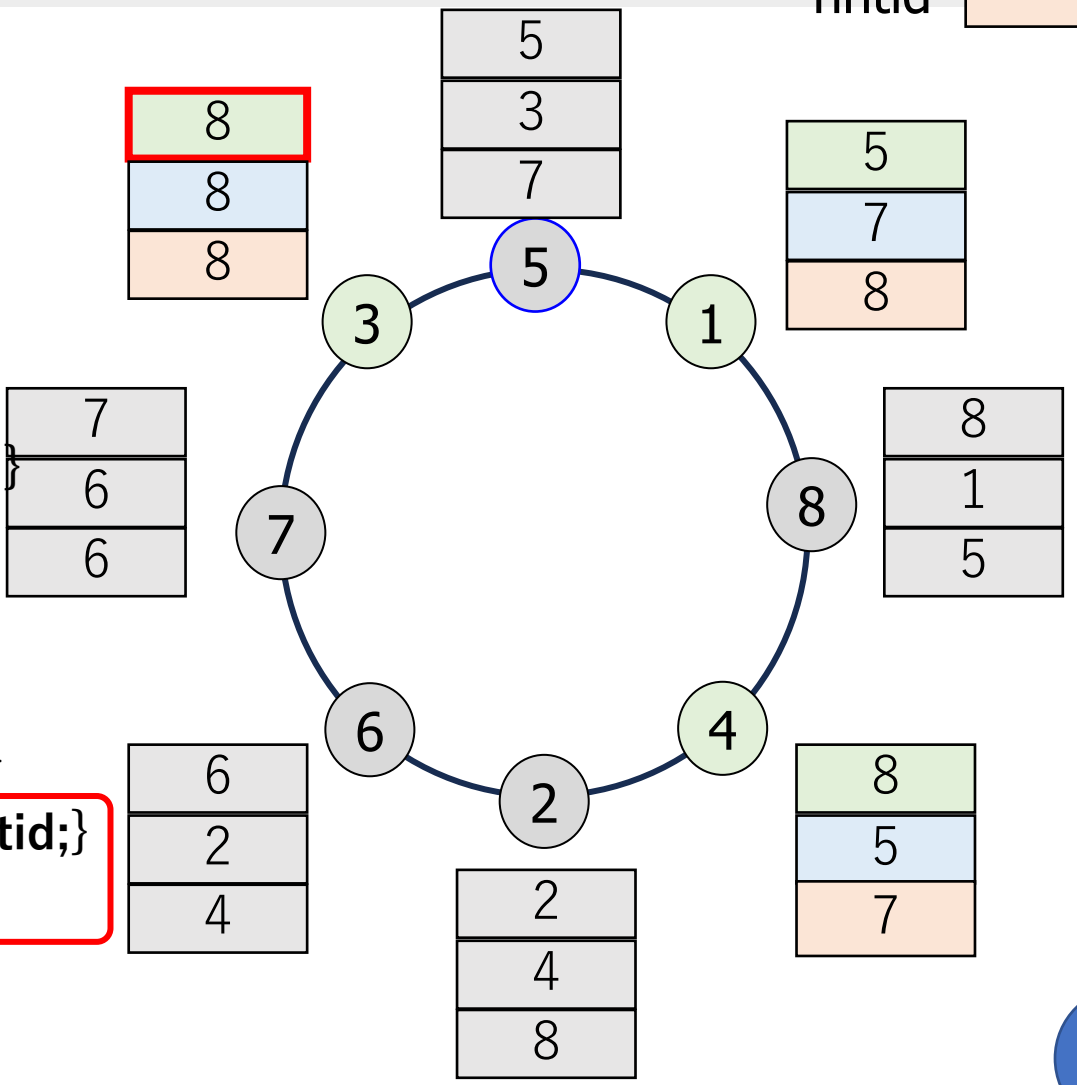
(2b)

```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid==my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
              else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```



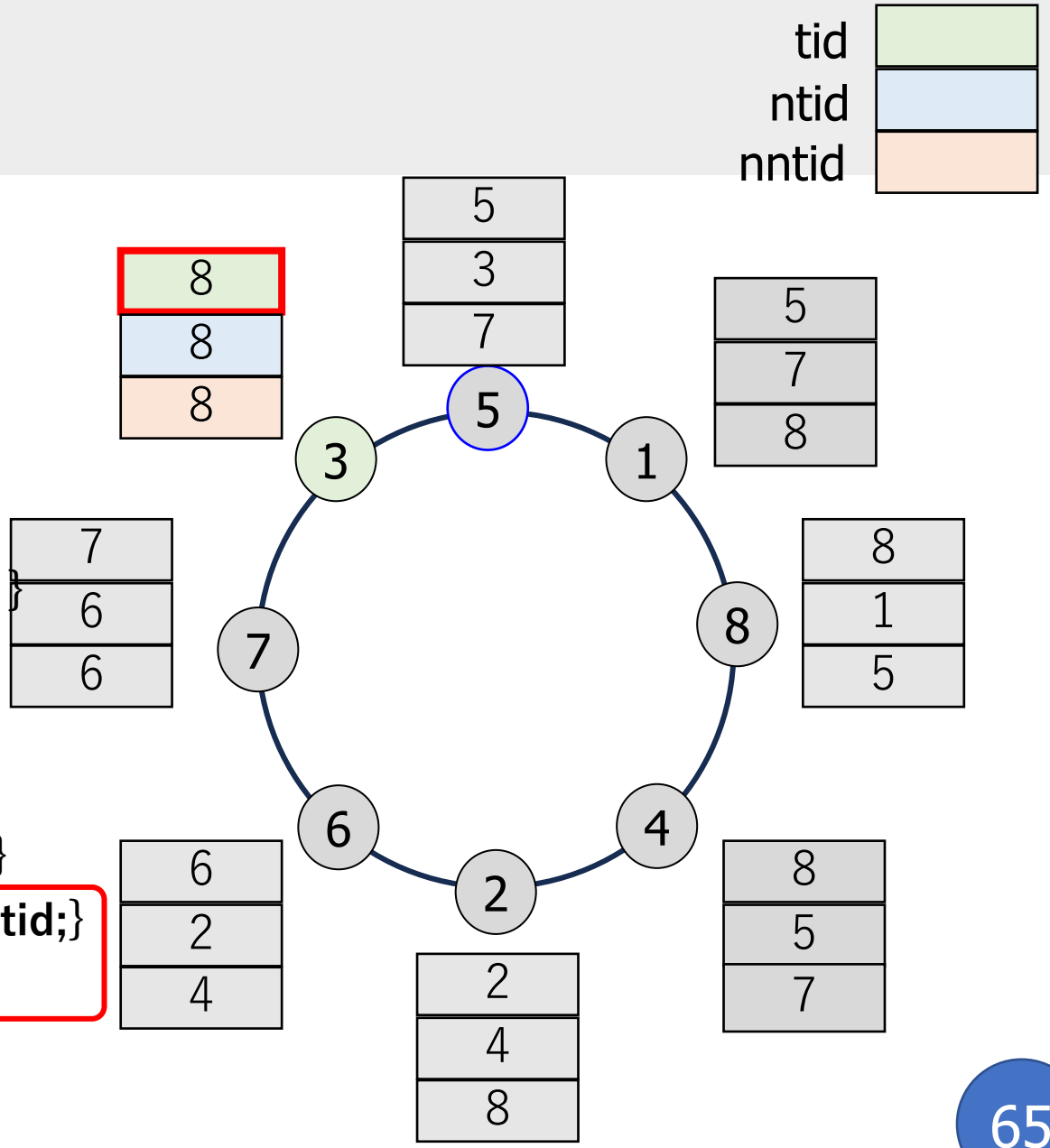
(2c)

```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid==my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
              else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```



(2c)

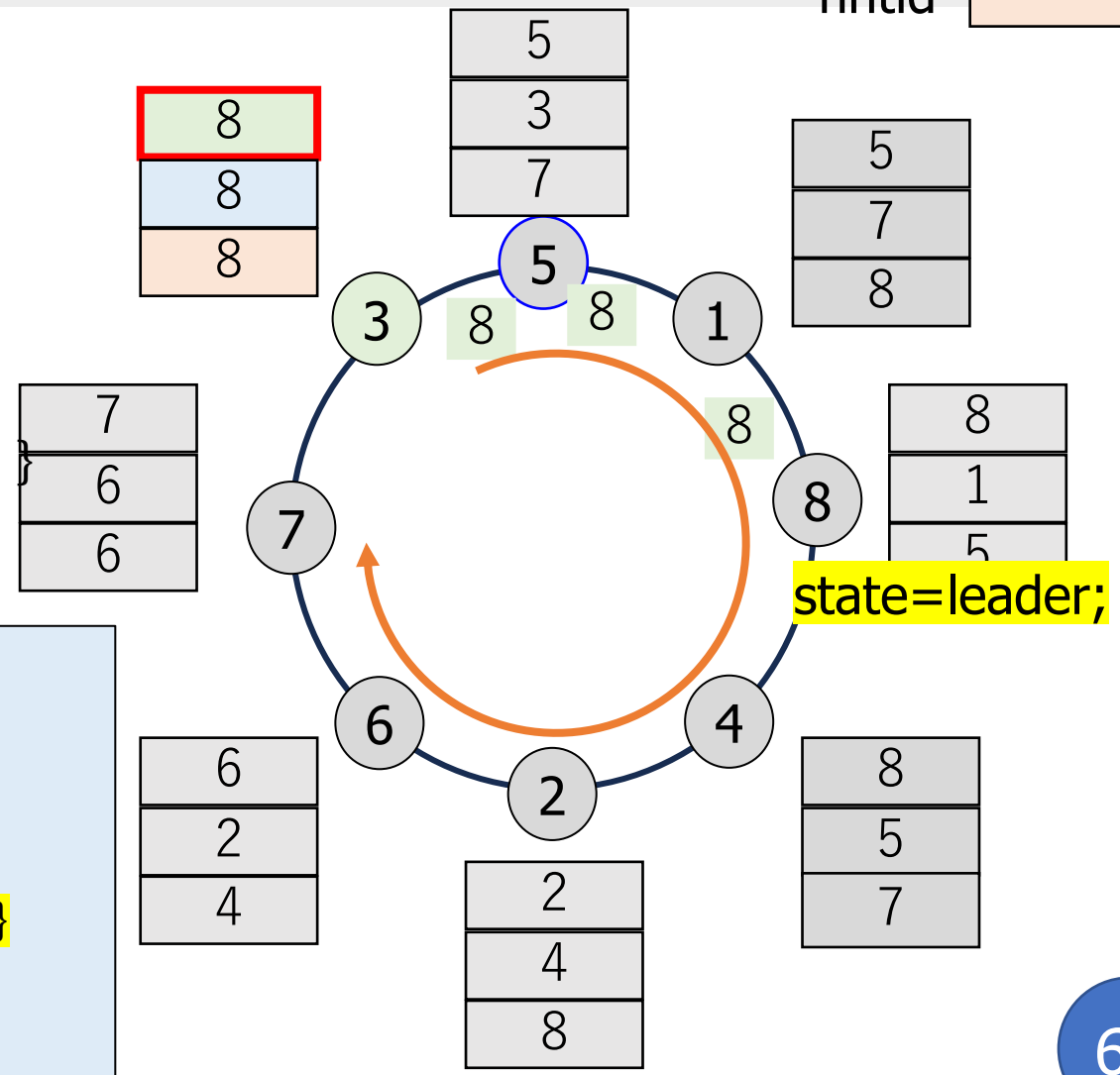
```
state=candidate;
tid=my_id;
while (state!=relay) {
    send(tid);
    receive(ntid)
    if (ntid==my_id) { sate=leader; }
    if (tid>ntid) { send(tid);}
            else {send(ntid);}
    receive(nntid);
    if (nntid==myid){ state=leader}
    if (ntid>=max(tid, nntid) {tid=ntid;}
    else state=relay;
}
//now state=relay
```



(3a)

```
state=candidate;
tid=my_id;
while (state!=relay) {
   send(tid);
   receive(ntid)
   if (ntid==my_id) { sate=leader; }
   if (tid>ntid) { send(tid);}
         else {send(ntid);}
   re
   if
   if
   el
}
//no
```

```
//now state=relay
while (state!=leader)
{
  receive(tid);
  if (tid==my_id) {state=leader;}
  send(tid);
}
```



今日の内容

- リーダー選出問題
  - 最大値問題に帰着される
  - 故障（完全停止）プロセスがいる

- （古典的な）選出アルゴリズム
  - ブリーアルゴリズム
  - リングアルゴリズム
    - 素朴なアルゴリズム（≒LeLannのアルゴリズム）
    - Chang-Robertsのアルゴリズム
    - Patersonのアルゴリズム

考え方が似てる これらをまとめて
LCRアルゴリズムと呼ぶこともある。