同期(2)
「排他制御、選任（選択）」

佐藤

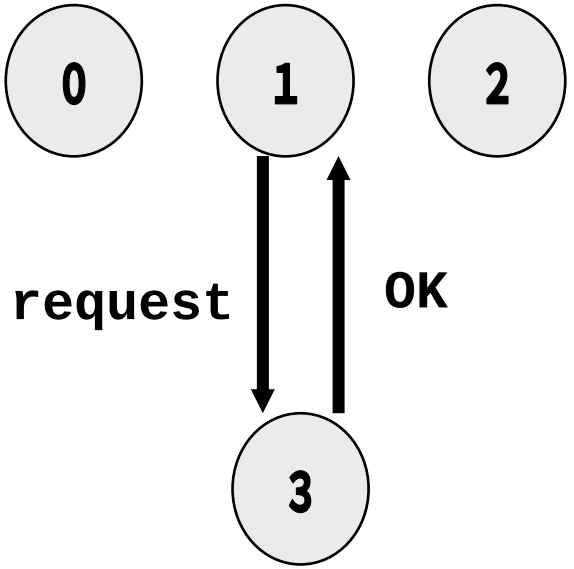
排他制御　(mutual exclusion)

- 排他制御とは、あるリソースに対する同時のアクセスに対して、競合するアクセスを排除すること
  - critical region: 排他制御する領域・期間
- 単一プロセッサシステムでは、
  - 割り込み禁止の操作をつかうことによって、ロックを実装できる。
- 共有メモリプロセッサシステムでは、
  - アトミックな操作（test-and-set命令）によって、ロックを実装できる。
- では、分散システムでは？？

排他制御：集中アルゴリズム

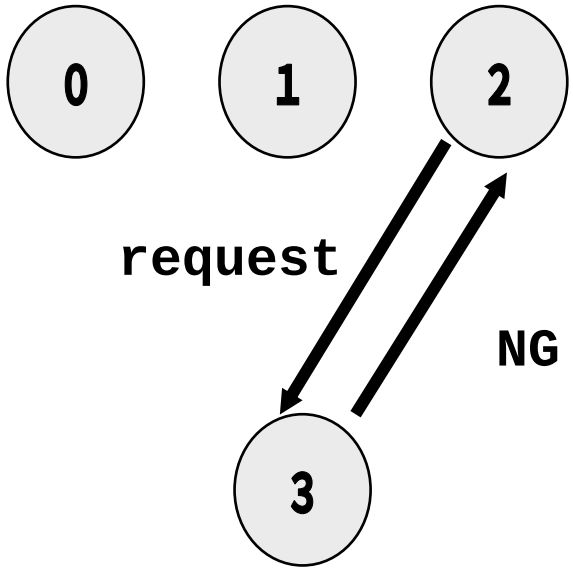
- **集中アルゴリズム(Centralized Algorithm)**
  - 一つの決まったプロセッサで、排他制御を実装

- **アルゴリズム**
  - どれかをcoordinatorにする。
  - critical regionに入りたいプロセスは、リクエストをcoordinatorに送る。
  - 既にcritical regionに入っているプロセスがなければ、許可のメッセージを送る。
  - 許可のメッセージを受け取ったプロセスは、critical regionに入る



排他制御：集中アルゴリズム

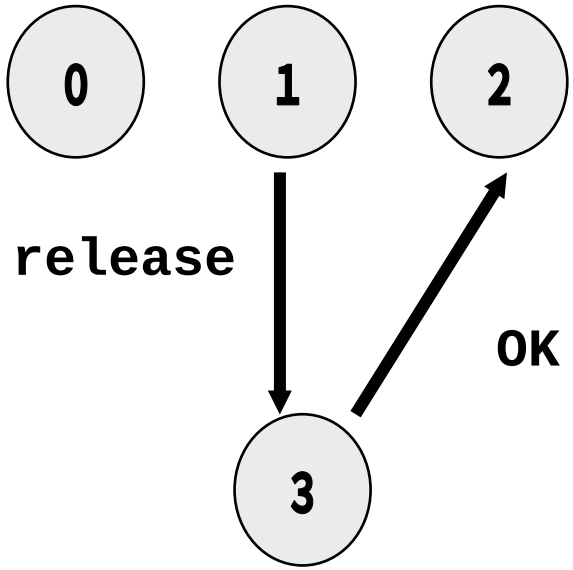
- アルゴリズム（続き）
  - だれかが、critical regionにはいっている場合には、queueにいれておく。
  - メッセージをおくらない、か、NGのメッセージをおくって、待たせておく。
    - NGのほうが親切？



2をqueueにいれておく

排他制御：集中アルゴリズム

- アルゴリズム（続き）
  - critical regionにはいっているプロセスが、critical regionからでる時には、releaseのメッセージを送る。
  - coordinatorがreleaseのメッセージをうけとった時に、queueにまっているプロセスがある場合には、そのプロセスにOKを送る



排他制御：集中アルゴリズム

- 利点
  - 簡単
- 欠点
  - coordinatorがcrashすると、全体が停止
  - プロセスが多い場合には、coordinatorが性能のボトルネックになる。
    - critical regionごとにcoordinatorを変えるなど、…　これが、Decentralized Algorithm へ、…

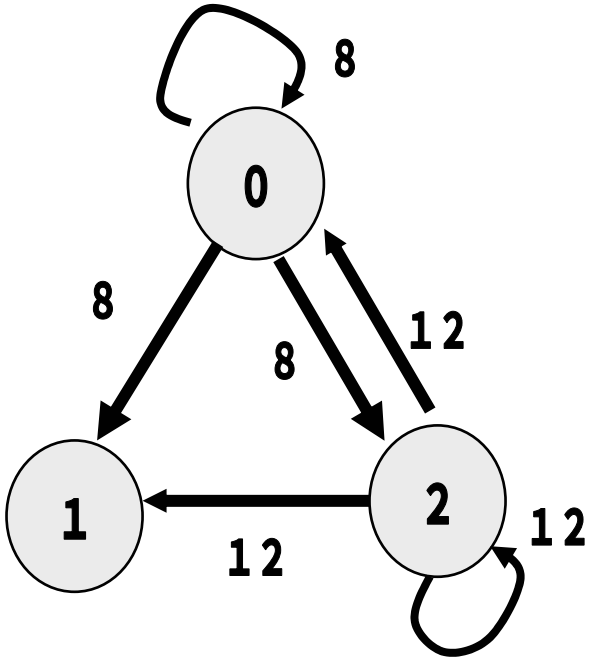
非集中アルゴリズム

- 非集中アルゴリズム(Decentralized Algorithm)
- DHT (Distributed Hash Table)-based System

排他制御：分散アルゴリズム

- **分散アルゴリズム(Distributed Algorithm [Ricart and Agrawala 1981])**
  - **集中のcoordinatorを持たない**
  - **システム全体で、全イベントに全順序がついている**
    - **Lamportの論理クロック**

- **アルゴリズム**
  - **critical regionに入りたいプロセスは、入りたいcritical regionの名前（ID)、プロセス番号と論理クロックのタイムスタンプを含むメッセージをすべてのプロセスに送信**
    - **通信は、reliableのものとする。パケットロスはない。**

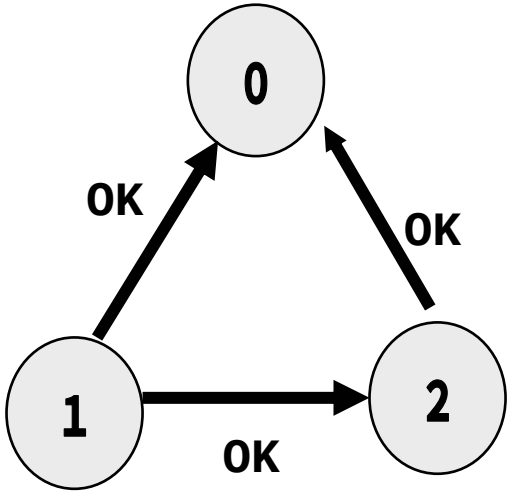


プロセス0とプロセス2が、リクエスト

排他制御：分散アルゴリズム

- リクエストを受信したプロセスは、
  - 受信プロセスが、critical regionにはいっていなくて、リクエストもしていなければ、OKのメッセージを返信
  - critical regionに既にはいっていれば、queueにいれて覚えておく。
  - 自分もリクエスト中ならば、time stampを比較して、受信したメッセージの方が早ければＯＫのメッセージを送る。
  - 全部のプロセスからＯＫをもらったらcritical sectionに入る

プロセス1は、2からのメッセージがあたらしいかったのでqueueしておく



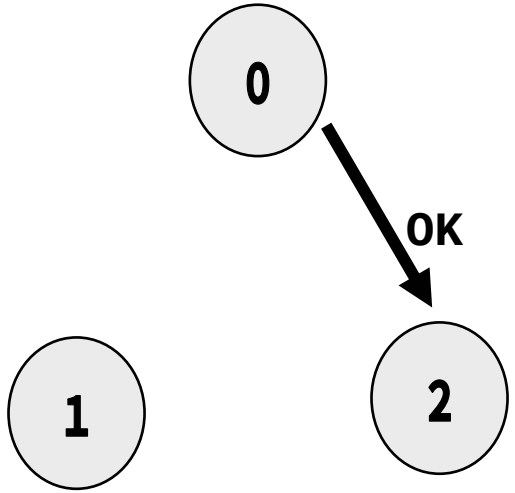
プロセス1は、0と2にOK

プロセス2は、受信メッセージが早かったので、1にＯＫ

排他制御：分散アルゴリズム

- critical regionから、出たプロセスは、queueに入っているプロセスに対して、OKのメッセージを送る。
  - 全部のプロセスからＯＫをもらったらcritical sectionに入る
- dead lockやstarvationは起こらない
  - time stampが順番を保障している限り

プロセス1はcritical sectionからでたら、queueにあるプロセス(2)にOKを送る。



プロセス2は、これで全部のＯＫがそろったので、critical sectionに入る

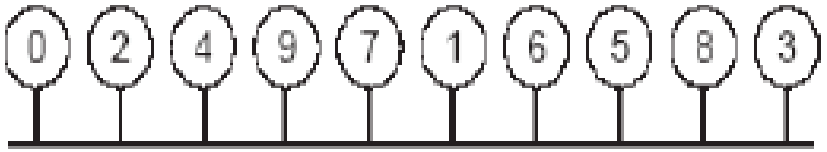
排他制御：分散アルゴリズム

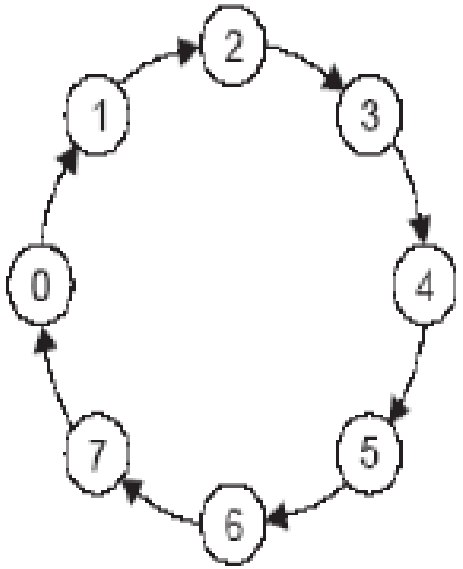
- 必要なメッセージ数は　2*(n-1) , n はプロセス数
- プロセスの一つでも、crashするとだめ
  - リクエストに対して返事がないと、それがcritical regionにはいっていて返さないのか、failしたのかわかわからない
    - 一つの手は、critical regionにはいっていたときはNGのメッセージをだすことも考えられる
  - 集中アルゴリズムと比較して、効率、耐故障も劣る

For one things, it shows that a distributed algorithm is at least possible. … Also, by pointing out the shortcomings, we may stimulate future theoreticians to try to produce algorithms … Finally, like eating spinach and learning Latin in high school, some things are said to be good for you in some abstract way. …

Token Ring Algorithm

- プロセス群を論理的なリング状に構成
- ネットワークの物理トポロジとは無関係に構成可能
- 各プロセスは自分の後が誰かを知っておけばよい





Token Ring Algorithm

- アルゴリズム
  - トークン(token)をリングに沿って回し、トークンを持っているプロセスのみがクリティカルリージョンに入る権利を持つ
  - もしトークンを持っているプロセスがクリティカルリージョンに入りたければ、入って処理を行い、終了したらトークンを次のプロセスに回す
  - クリティカルリージョンに入りたくなければ、単に次のプロセスにトークンを回す
- 問題点：トークンが失われたら機能しない
  - 失われたことの検出も困難（次のトークンが回ってくるまでの時間に上限がない　タイムアウトでは検出不可）

アルゴリズムの比較

- 集中アルゴリズム
  - （クリティカルリージョンに1回入って出るのに）必要なメッセージ数＝３（リクエスト、許可、解放）
- 分散アルゴリズム
  - 必要なメッセージ数＝2(n-1) （リクエストと許可がそれぞれ(n-1)プロセス分）
- TokenRingアルゴリズム
  - 必要なメッセージ数＝1から∞（不要なときもトークンを回し続ける必要があるので）

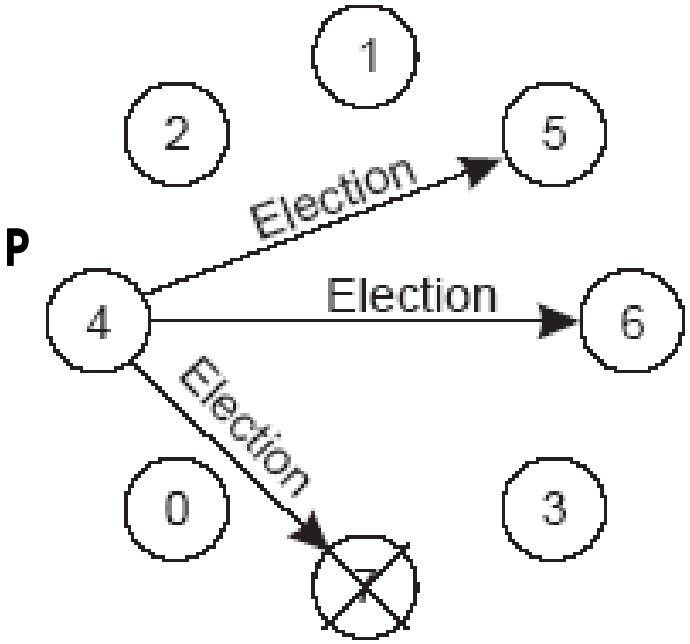
選任(Election)アルゴリズム

- 多くの分散アルゴリズム：ある一つのプロセスを取りまとめ役(リーダー、コーディネーター)として選ぶ必要
  - どのプロセスでも構わないが１つ選ぶ必要がある
- 複数プロセスからリーダー（コーディネーター）を選ぶアルゴリズム
  - 仮定：
    - 各プロセスには一意の番号(ID)が割り振られている
    - 任意のプロセスは他の全てのプロセスの番号を知っている
- ゴール：全てのプロセスが、誰がコーディネーターになったかに関して合意

ブリーアルゴリズム(bully algorithm)

- あるプロセスPが現在のコーディネータが反応しない（障害などで停止した）ことに気づいたときに、選任(election)を開始

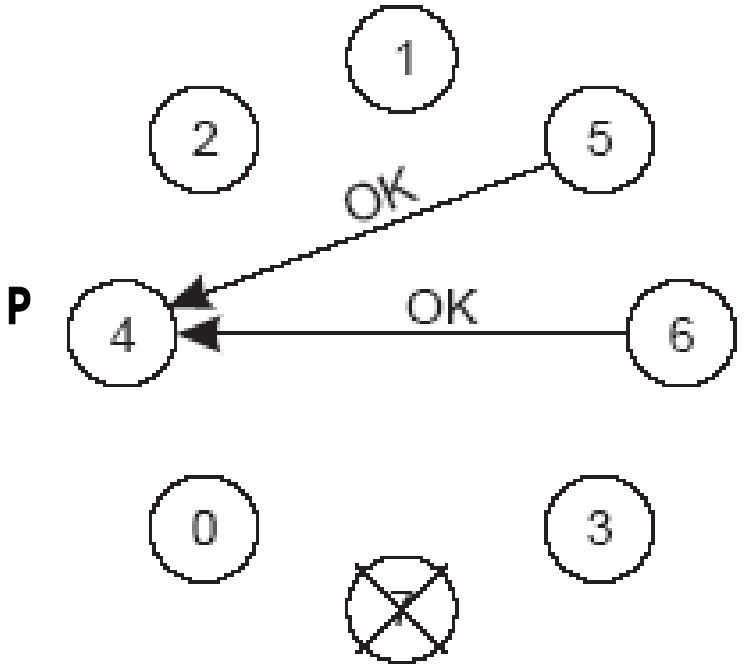
- Pは次のように選任の作業を行う
  - PはELECTIONメッセージを、自分より上の番号の全てのプロセスに対して送信



7が反応しないのを、P=4が気がついた

ブリーアルゴリズム(bully algorithm)

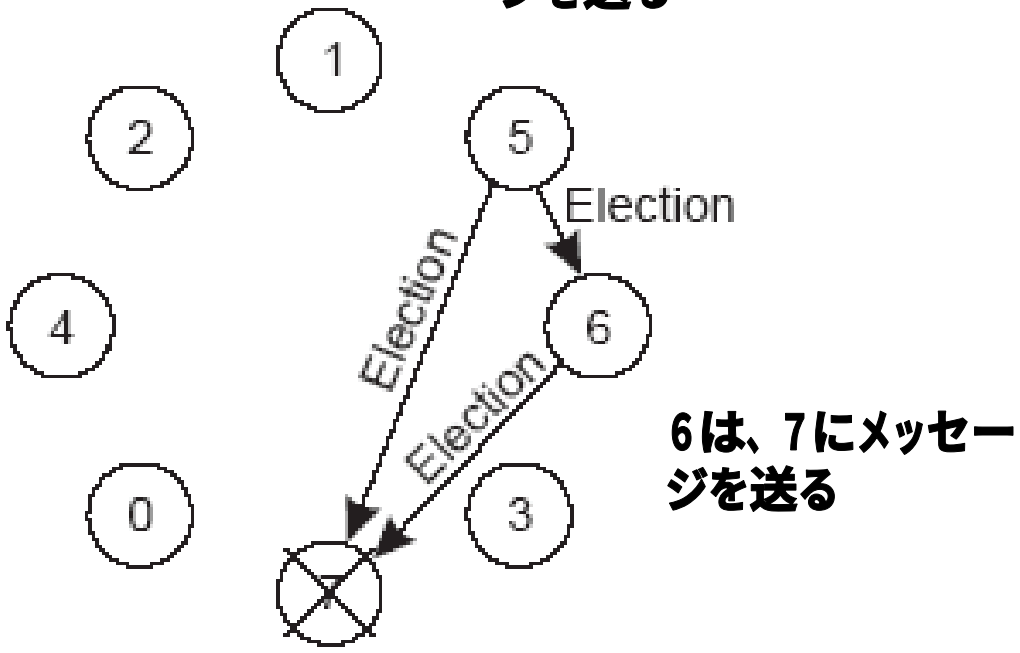
- もし誰も応答しなければ、Pは選挙に勝ったと見なし、自分がコーディネータとなる
- m 誰か上位の番号のプロセスが応答すれば、（選挙に負けたと判断し）Pの仕事は終了



5，6が反応したので、4は負けた

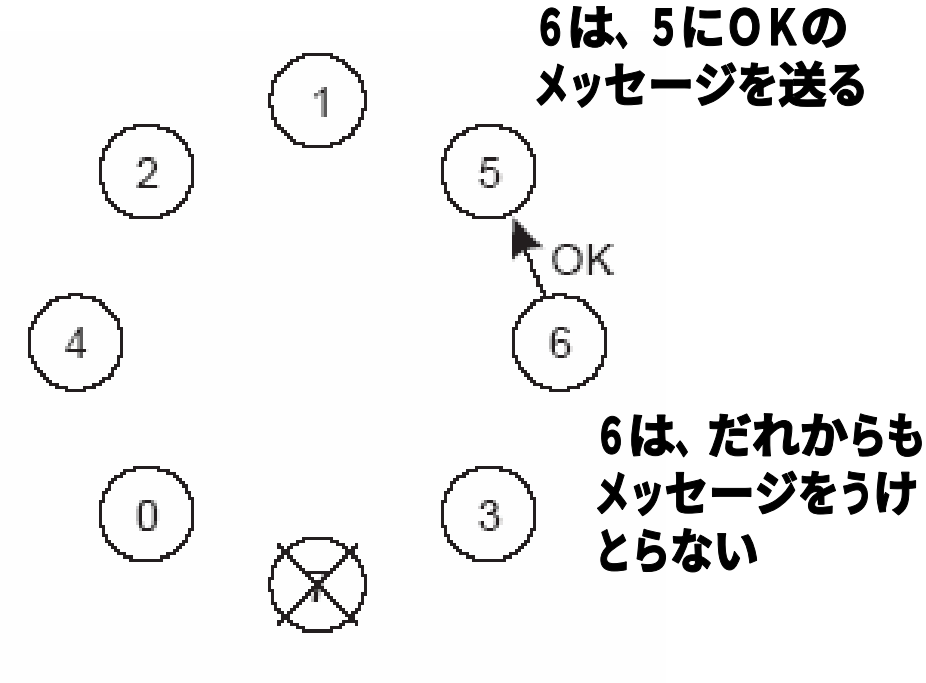
ブリーアルゴリズム(bully algorithm)

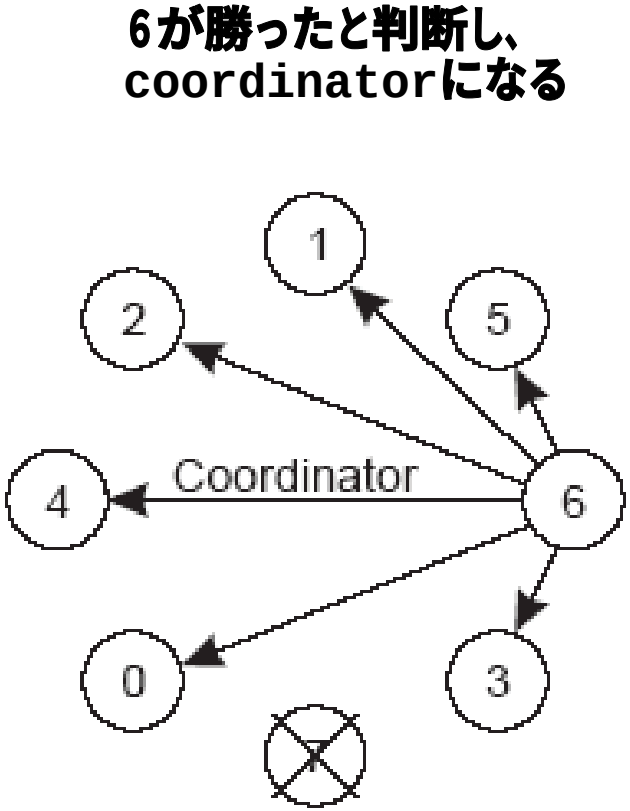
- ELECTIONメッセージに応答したプロセスは、それぞれ自分が選任アルゴリズムを開始する



ブリーアルゴリズム(bully algorithm)

- もし誰も応答しなければ、Pは選挙に勝ったと見なし、自分がコーディネータとなる





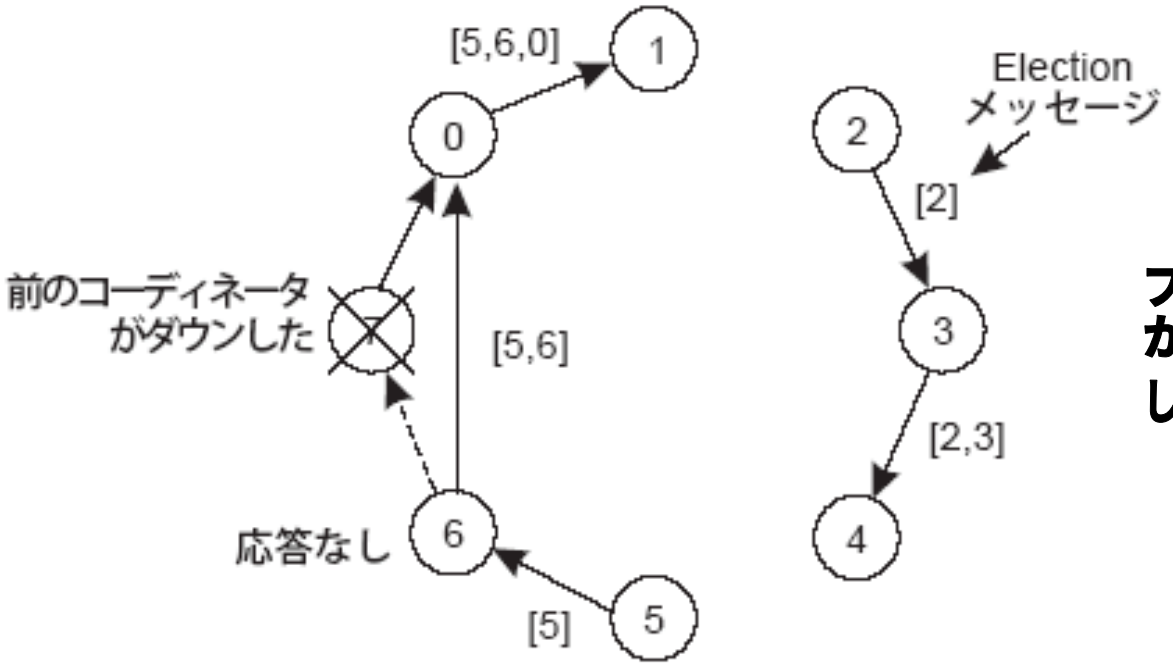
ブリーアルゴリズム(bully algorithm)

- まとめ
  - 任意のプロセスは自分より下位の番号のプロセスからELECTIONメッセージを受信する可能性がある
  - もし受信したならば，自分が生きていることを示すOKメッセージを返信
  - そして、今度は自分が（上記の）選任の仕事を実行
  - いつかは、１つを除いた全てのプロセスが選任の仕事を終え、ある1つのプロセスがコーディネータとなる
  - 選挙に勝ったプロセスは、他の全てのプロセスに対して自分の勝利を通知

- 番号が上位のプロセスが常に勝利するので、“bully(ガキ大将) algorithm”と呼ばれる

Ring Algorithm

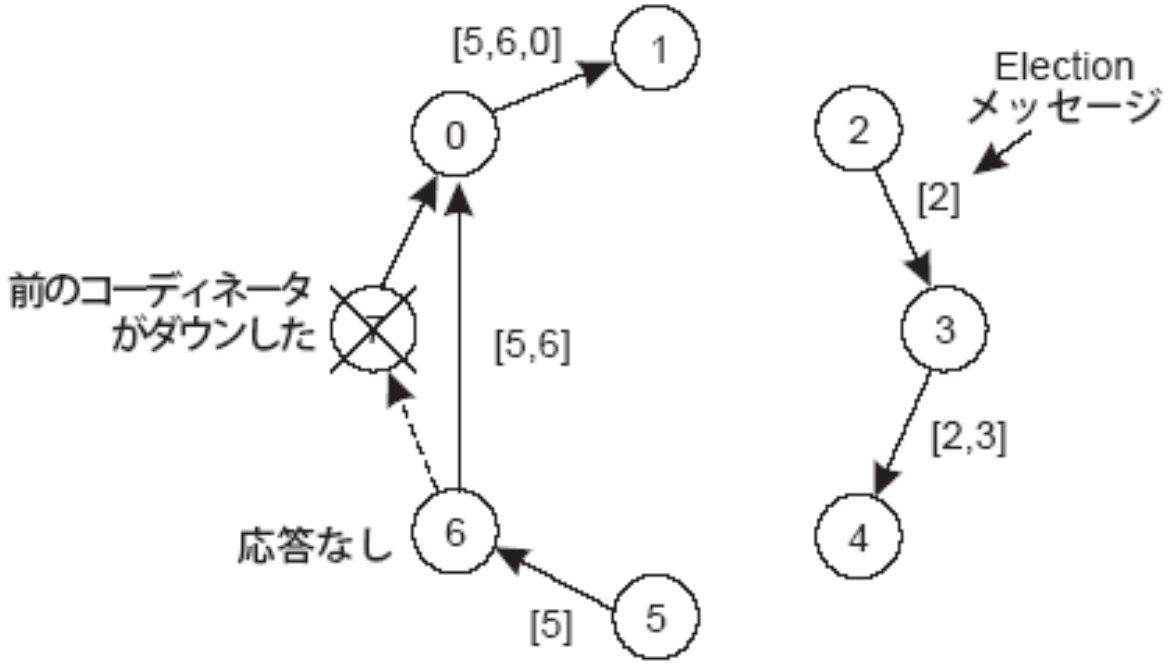
- リングを用いた選任アルゴリズム
- プロセス群は順序付けられており、各プロセスは自分の次の順番のプロセス(後続者、successor)を知っていると仮定
- コーディネータが機能していないことに気づいたプロセスは、ELECTIONメッセージに自分のプロセス番号を付けて、後続者に送信



プロセス2，5
が同時に、7がdown
したことを検知

Ring Algorithm

- もし後続者がダウンしていたら、一つとばした次のプロセスに送信（生きているプロセスが見つかるまで繰り返す）
- ELECTIONメッセージを受信したメッセージは、自分が生きていることを示す返事を返すとともに、受信したELECTIONメッセージに自分のプロセス番号をリストに追加して、自分の後続者に転送



プロセス2，5
が同時に、7がdown
したことを検知

Ring Algorithm

- いつかは自分が送信したELECTIONメッセージが自分に返ってくる（リストに自分の番号があることにより認識）
  - 今度はメッセージタイプをCOORDINATORに変更してもう一度回覧
  - 今度は誰がコーディネータであるか（リストのうち最も大きな番号のプロセス）、および、リングを構成する新しいメンバーが誰であるかを知らせるため
  - COORDINATORメッセージの回覧が終了すると、それぞれが自分の仕事に戻る

- 複数のプロセスが同時にELECTIONメッセージの回覧を始めることがある
  - 最終的に選ばれるコーディネータは一意なので実害は無い
  - メッセージが重複するがトラフィックに与える影響はわずか

まとめ

- 分散システムでの排他制御
  - 集中アルゴリズム
  - 分散アルゴリズム
  - Token Ring アルゴリズム
- 分散システムでの「選任」
  - Bully アルゴリズム
  - Ring アルゴリズム