# クラウドシステム基礎

# 第3回：定足数によるアプローチ

国立情報学研究所
石川　冬樹
f-ishikawa@nii.ac.jp

# 今回の内容

- 全プロセスで同一状態を維持しなくとも，全体として進み続けるための仕組みを議論する
  - 投票型の障害対応に関する重要な概念として，quorumおよびビザンチン合意を学ぶ
  - 総合的な例として，グループ管理サービスの構築方法を議論する

# 目次

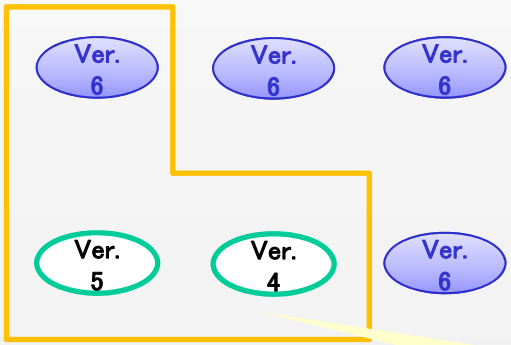
- 固定グループにおける定足数アプローチ
- グループ管理サービスの構築

# Quorum：概要

- （数が固定の）複製管理の方法において，過半数を用いる方法がよく知られている
  - 書き込みの際には，過半数の複製に書き込みが成功することを確認する
  - ➡競合する書き込み2つが両立することはない
  - 読み込みの際には，過半数の複製から読み込むようにする
  - ➡少なくとも1個の複製は最新の値を知っている
- 実際には「過半数」に限らないこともよく知られている：quorum（クォーラム）：定足数

# Quorum：例

4個に最新の値を書き込んだ状態

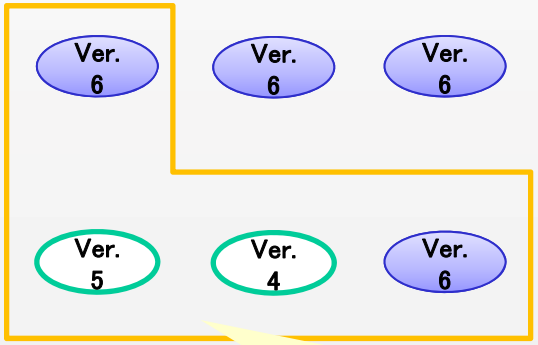



3個読み出すと？

最新の値を見つけられる
（4 + 3 > 6 なので）

「書き込み3・読み込み3」や，
「書き込み4・読み込み2」だと，
そうとは限らない

4個に最新の値を書き込んだ状態




Ver. 5が最新だと思っていて，
4個にVer. 6を書きに行くと？

競合している更新を見つけられる
（4 + 4 > 6 なので）

「書き込み3」だと，
そうとは限らない

# Quorum：要件

- パラメータ
  - 読み込まなければならない最小の複製数QR（Quorum-Read）
  - 書き込みを行わなければならない最小の複製数QW（Quorum-Write）
- 必要な条件：プロセス数をnとしたとき，

  QW+QW ＞ n かつ QR+QW ＞ n

# Quorum：補足

- 書き込みには2PCが必要
- なお，プロセス数nに対し，
  - QW=n，QR=1 とすると，
    「全てに書き込み」「読み出しはどれでも1つから」
  - QW=QR= (nを2で割った商+1) とすると，
    「読み書きともに過半数が必要」

※ 上記は，読み書きのどちらが多く高速に行いたいかに関する両極端の選択肢
（これらの間のどこを採ってもよいということ）

# Quorum：他の活用

- 先の図は複数ノードの読み・書き競合への適用
- 障害時への対応にも適用される
  - ネットワークが分断されノードグループが複数できたとき，それぞれが他が死んだと思って処理を進めてしまうと一貫性が失われる可能性がある
  - Quorumを確保できたときだけ処理を進めるようにする（そのようなグループは高々1つだけ）
  - Quorumの考え方・言葉が出てくる実例：
    http://technet.microsoft.com/ja-jp/library/cc731739.aspx

# ビザンチン将軍問題：問題

- 通信に障害はなくプロセスに障害がありうる場合を表す比喩
  - シナリオ
    - A軍がn個に分かれている
    - 将軍は電話でそれぞれの部隊の人数を知らせ合い，軍の全人数について同意する
  - 問題
    - n人の将軍のうちm人は裏切り者であり，不正確で矛盾する情報を流し，同意に至るのを防ごうとする
    - nとmの関係により同意に至れない

# ビザンチン将軍問題：解法

- よく知られた解法 [Lamport, 1982]
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

# ビザンチン将軍問題：例（１）

- 例：4人のうち将軍3が裏切り者の場合
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える

将軍1が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> a，部隊4 -> C4
将軍2が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> b，部隊4 -> C4
将軍3が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> C3，部隊4 -> C4
将軍4が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> c，部隊4 -> C4

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：例（１）

- 例：４人のうち将軍3が裏切り者の場合
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る

将軍1が将軍2, 3, 4から受け取る情報：(C1, C2, b, C4), (d, e, f, g), (C1, C2, c, C4)

将軍2が将軍1, 3, 4から受け取る情報：(C1, C2, a, C4), (h, i, j, k), (C1, C2, c, C4)

将軍4が将軍1, 2, 3から受け取る情報：(C1, C2, a, C4), (C1, C2, b, C4), (l, m, n, o)

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：例（１）

- 例：４人のうち将軍3が裏切り者の場合
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

将軍1が将軍2, 3, 4から受け取る情報：(C1, C2, b, C4), (d, e, f, g), (C1, C2, c, C4)
-> (C1, C2, ?, C4)

将軍2が将軍1, 3, 4から受け取る情報：(C1, C2, a, C4), (h, i, j, k), (C1, C2, c, C4)
-> (C1, C2, ?, C4)

将軍4が将軍1, 2, 3から受け取る情報：(C1, C2, a, C4), (C1, C2, b, C4), (l, m, n, o)
-> (C1, C2, ?, C4)

忠実な将軍は合意に至ることができる

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：例（２）

- 別の例：3人のうち将軍3が裏切り者の場合
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える

将軍1が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> a
将軍2が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> b
将軍3が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> C3

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：例（２）

- 別の例：3人のうち将軍3が裏切り者の場合
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る

将軍1が将軍2, 3から受け取る情報：(C1, C2, b), (c, d, e)

将軍2が将軍1, 3から受け取る情報：(C1, C2, a), (f, g, h)

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：例（２）

- 別の例：3人のうち将軍3が裏切り者の場合
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

将軍1が将軍2, 3から受け取る情報：(C1, C2, b), (c, d, e)
–> (?, ?, ?)
将軍2が将軍1, 3から受け取る情報：(C1, C2, a), (f, g, h)
–> (?, ?, ?)

忠実な将軍は合意に至ることができない

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

# ビザンチン将軍問題：位置づけ

- この解法が満たす性質
  - m個障害を持つプロセスがあっても，2m+1個正常なプロセスがあれば（全体の3分の2より多くが正常ならば），合意形成が可能
- 特定の障害の分類としては著名
  - ビザンチン故障（実行結果のぶれや壊れ，もしくは応答なしなど，計算に関する任意の障害）
  - ビザンチン故障に耐えられるかどうか：ビザンチン・フォールトトレラント性（BFT）

# ビザンチン将軍問題：位置づけ

- 「プロトコル」としては想定が合わないかも
  - 通信については楽観的
  - プロセスについては，乗っ取りや誤動作などを悲観的に想定する一方，それらが起きるプロセスの数は一定以下と見積もれなければならない
- セキュリティ観点などの応用事例も
  - 例：外部サーバ群にデータを複製して置き，データの完全性（改ざんなし，など）を実現
  - 今回の話とは多少異なるが，ビットコインの仕組み説明に登場

# 目次

- 固定グループにおける定足数アプローチ
- グループ管理サービスの構築

# グループ管理：概要

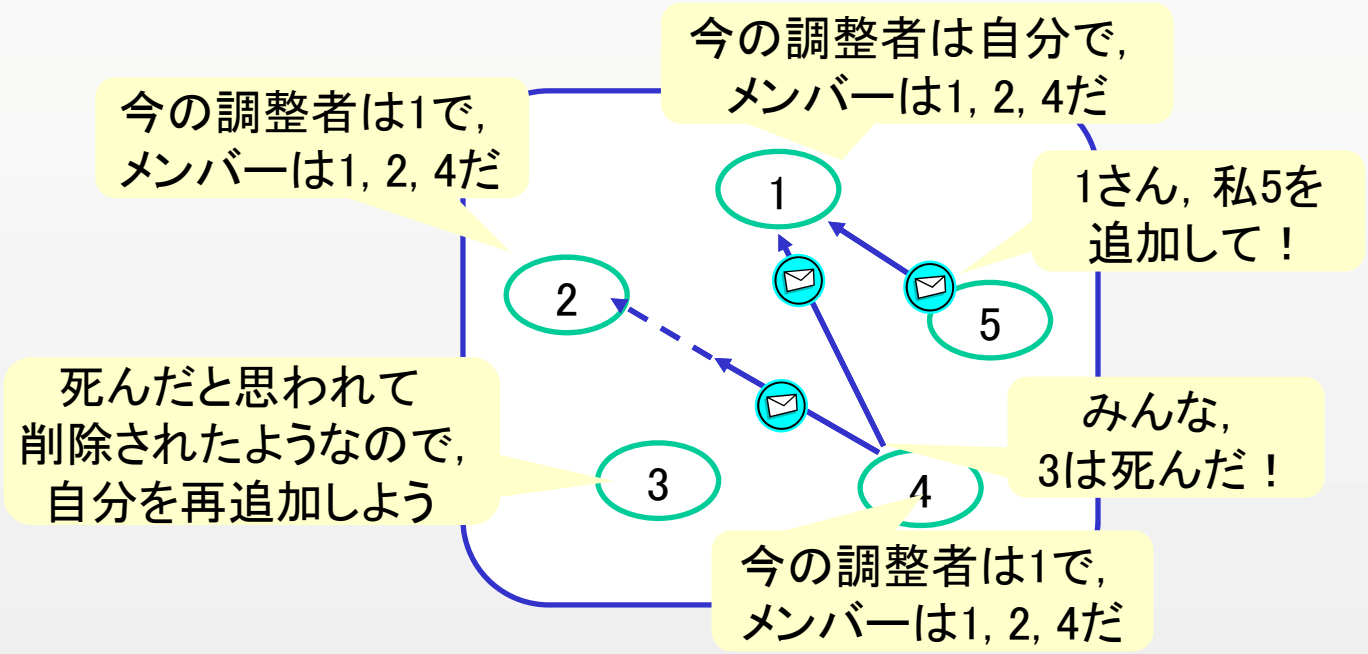
- プロセスのグループ内で，互いの存在を把握，管理し続ける仕組みを検討する
  - これまで固定数のグループを考えたが，プロセスの追加・削除を扱う
    - プロセス障害あるいはネットワーク分断への対応（再起動・再接続による復帰を前提としない）
    - あるいは，負荷に応じた意図的なスケーリング

# グループ管理：概要

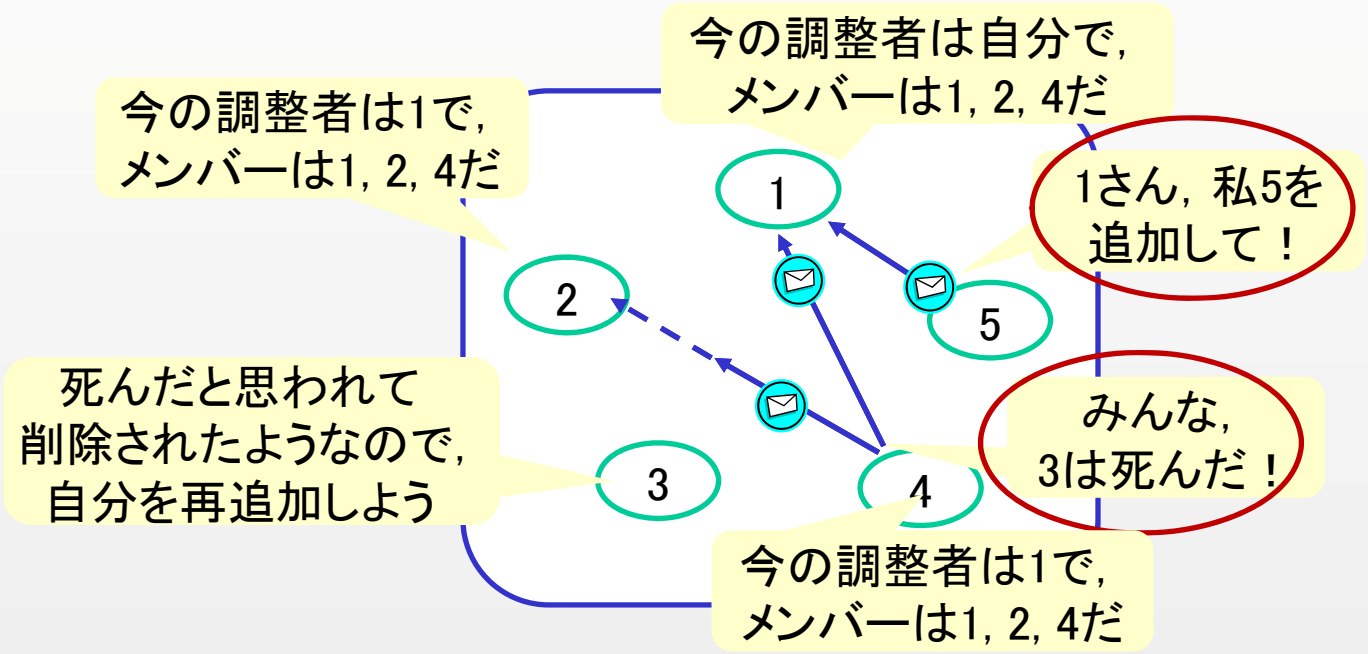
- プロセスのグループ内で，互いの存在を把握，管理し続ける仕組みを検討する（続）
  - ある程度限られた数の複製のみを考えるとして，密に（コストをかけて）管理する方針を採る
  - より膨大なグループの管理に対しては，この仕組みで作ったグループで多重化された管理サービスを実現すればよい

※ 有名なプロトコルというより，これまでの話を総合的に用いる一つの設計例（教科書2より）

# グループ管理：概要

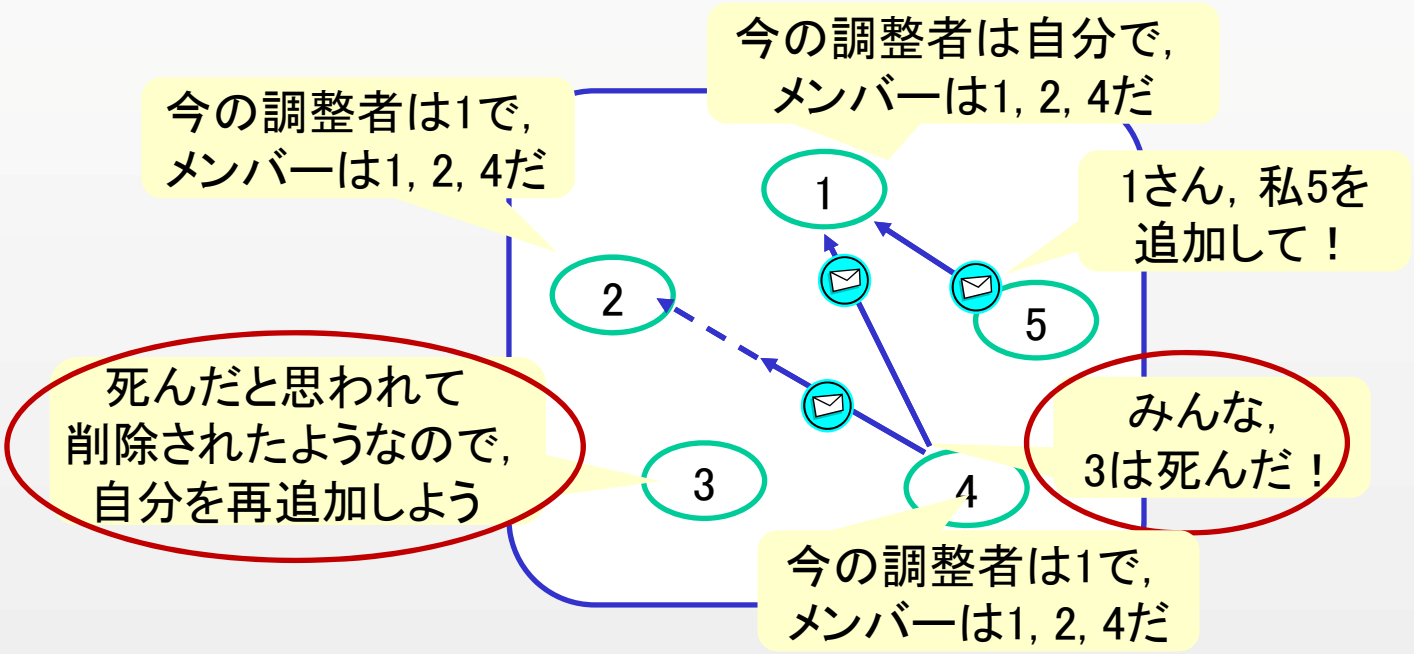


# グループ管理：機能



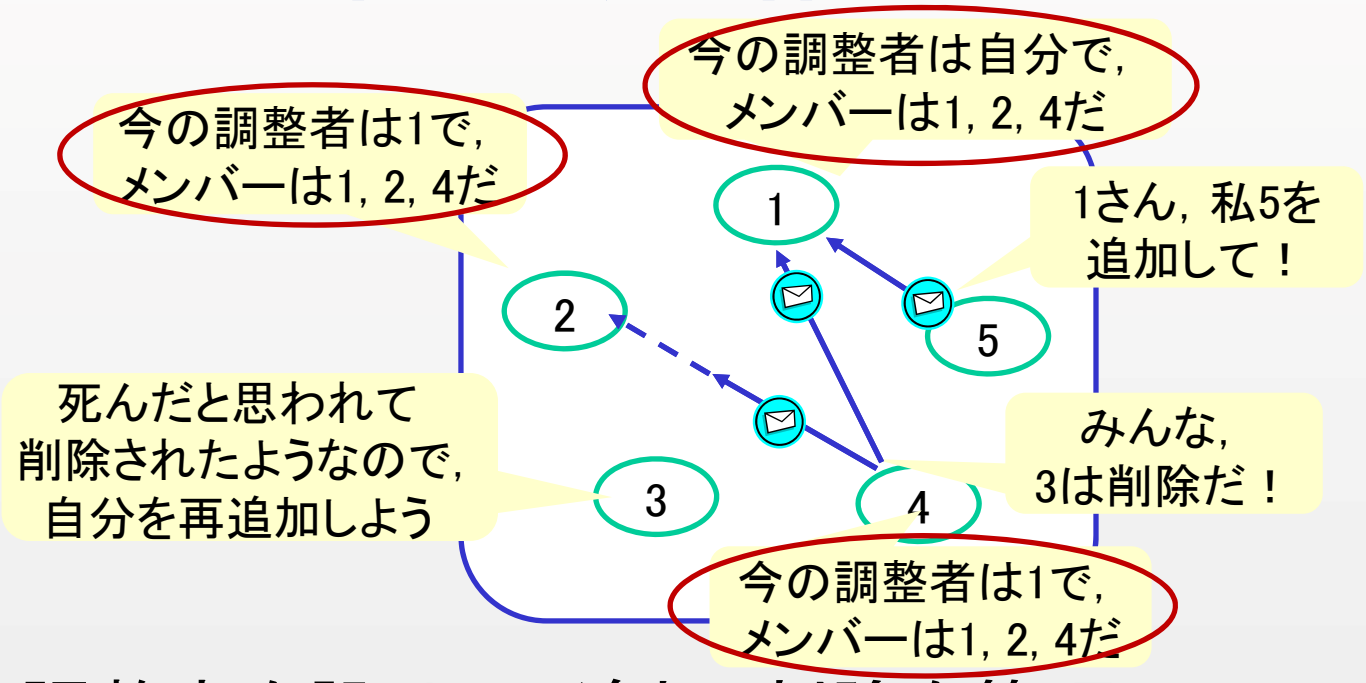

- 操作はすべて冪等とする
  - 追加，削除，監視（イベントリスナー）登録等

# グループ管理：生存確認




- 定期的に互いにpingなどで生存確認
  - 反応がなければ脱落扱いとして周知してしまう
  - 該当プロセスが実際生きていてメッセージを送ってもエラーが返されるので，自分を再度新規追加する

# グループ管理：調整者




- 調整者を設けて，追加・削除を管理
  - 参加順などで順序を一意に決めておき，削除された場合には次のプロセスが引き継ぐ
  - 追加・削除の詳細は次頁で

# グループ管理：調整者

- 調整者は追加または削除を管理する
  - 追加，削除対象は複数プロセスでよいが，削除対象は現在のプロセス数の過半数未満に限る
  - まず，追加または削除の確認メッセージを，現在のプロセスリスト内の全プロセスに送る
  - 過半数のプロセスからackが来たら，更新内容を確定するように周知する

※ ackが来なければ脱落扱い，と生存確認を兼ねてもよい

# グループ管理：留意事項

- 調整者が，過半数のプロセスからackがとれなかった場合
  - 分断されていて，自分が少数派にいるということとなる（別のところで過半数がつながっていて自分たちは死んだものとされているか，皆が過半数未満に分断され全体が維持できなくなっているか）
  - 今回の想定（小さい数の大事なグループ）では，こういった分断や障害について，ハードウェアなどの別の仕組みで検出，判断することも考えられる
  - あきらめずに処理を続けておき，つながったら情報をマージするような拡張も考えられる

# グループ管理：留意事項

- 調整者の削除
  - いったん，新しい調整者は古い調整者の削除について周知した方がよい
  - 各プロセスは古い調整者を「敬遠」するようになる
  - 古い調整者が完了していない追加，削除に関する情報をついでに集めてもよい
  - その後，これまで示した削除の手続きをとる

# グループ管理：性質まとめ

- グループに属するプロセスの初期リストから始め，追加・削除が行われたリストが更新されていく
- 各プロセスは，その更新の履歴列の部分列（そのプロセスが加わってから，抜けるまで）の報告を受けることになる
- i番目のリストにおける過半数が，i+1番目のリストに同意しなければならない
（競合する更新が同時に行われることはない）
- プロセスpがプロセスqが落ちているようだと判断し，全体が機能しているならば，pまたはqの片方または両方がリストから抜ける

# 今回のまとめ

- 必ずしもすべてのノードで同一状態を保持しなくても，一定数のみ確認することで整合性を維持する考え方もある
- 実際には，典型的な考え方を統合して実現方法を検討することになる
  - 調整者の設定，固定順序などの調整者交代方法，幂等な操作定義，定足数に基づいた情報更新，役割交代するための情報の冗長保存，・・・
- 次回：イベントの順序づけについて議論する