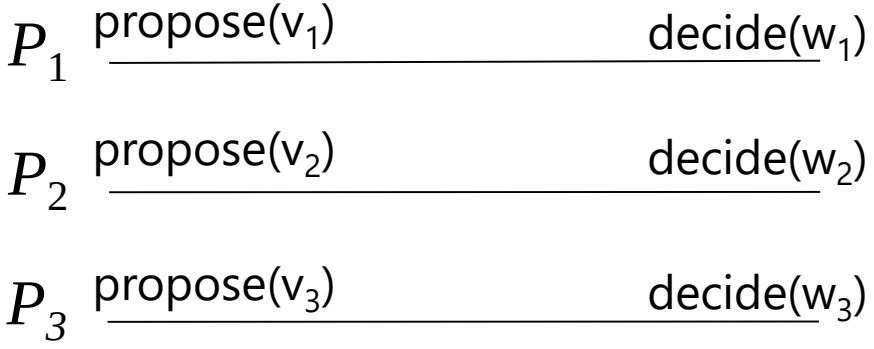
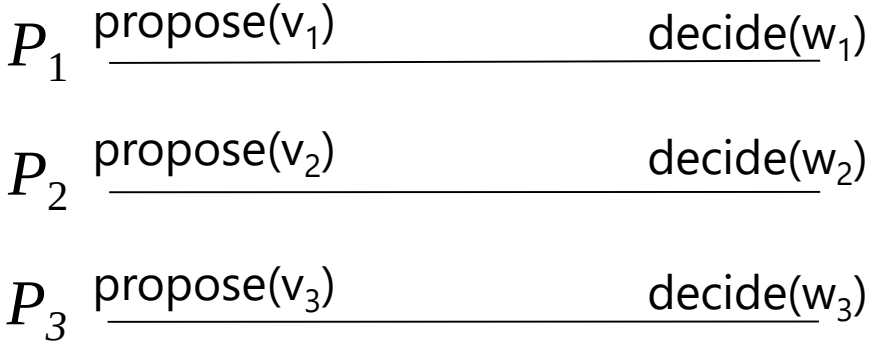
コンセンサス (Consensus)

- プロセスが一致した決定をする問題
- システムモデル
  - 非同期システム
  - クラッシュ故障
    - ◆他のモデルを仮定する場合も多い
- プロセス
  - $P_1, P_2, \ldots, P_n$
- プロセスP_iが行う操作
  - propose(x_i)
  - decide(w_i)



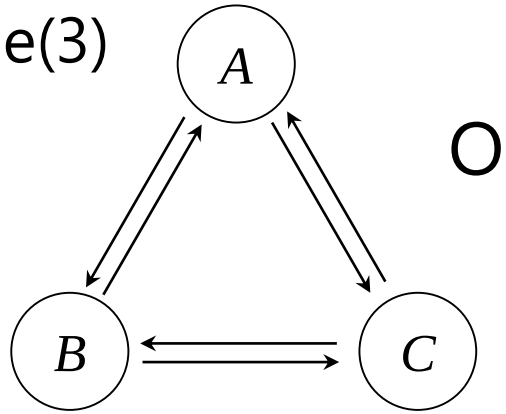
Conditions

- 停止性 termination
  - P_i (i =1,..,n)は, 正常なら, いずれdecideする
- 合意 agreement*
  - P_i, P_j (i, j =1,.., n)がw_i, w_jをdecideしたとき, w_i = w_j
- 妥当性 validity
  - w_iがdecideされたとき，w_iをproposeしたプロセスが存在 (つまり, v_j=w_iであるP_jが存在)



*正確には uniform agreement

propose(1)
decide(3)

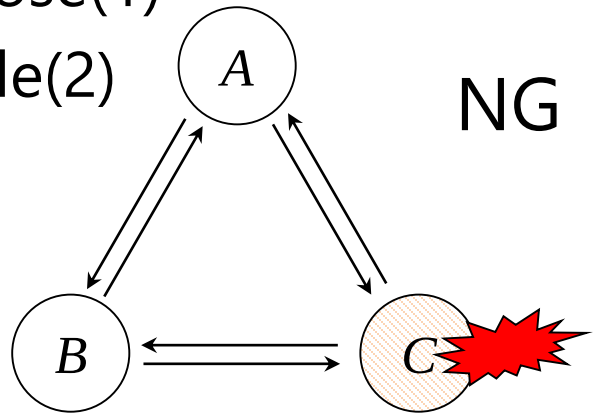


OK

propose(2)
decide(3)

propose(3)
decide(3)

propose(1)
decide(2)

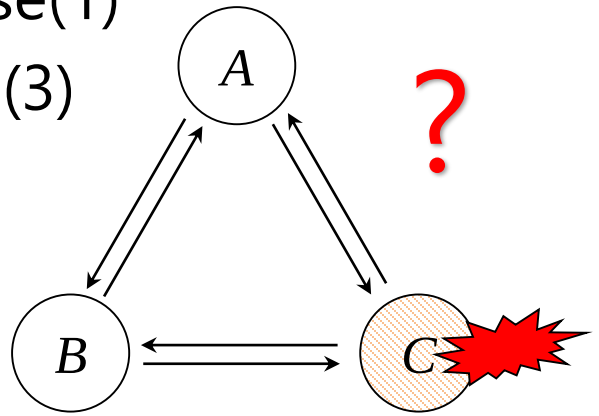


NG

propose(2)
decide(2)

propose(3)
decide(3)

propose(1)
decide(3)



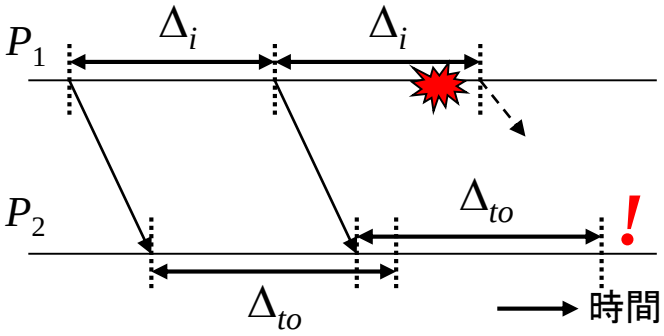
?

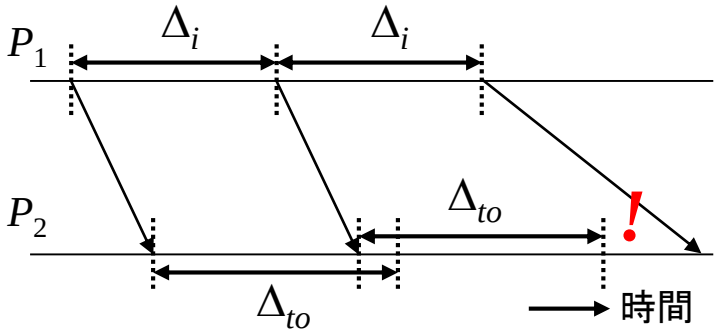
propose(2)
decide(3)

propose(3)

Impossibility result

- 1プロセスがクラッシュする可能性があるだけでも, 非同期システムにおけるアルゴリズムはない
  - FLP impossibility result
- メッセージ遅延と, プロセスの故障を区別できないため





コンセンサスアルゴリズム

- 現実的にコンセンサスを解くためのアプローチ
  - システムに同期性がまったくない場合でも，悪いことは起こらないようにする
  - システムが同期性の点で安定したら，停止性を満たす
- アルゴリズムの例
  - Paxos アルゴリズム (n > 2k, k：故障プロセスの数)
    - ◆Omission故障に対応
  - PBFTアルゴリズム　(n > 3k)
    - ◆Byzantine故障に対応

安全性と活性 (safety and liveness)

- 分散システム・アルゴリズムの性質は2つに大別される
  - 安全性 safety：悪いことが起こらない
    - ◆注意. 語句は同じだが, 人命や身体への危害とは無関係
  - 活性 liveness：良いことがおこる
- コンセンサスの条件は, それぞれどちらの性質か？
  - 停止性 termination →
  - 合意 agreement →
  - 妥当性 validity → 安全性

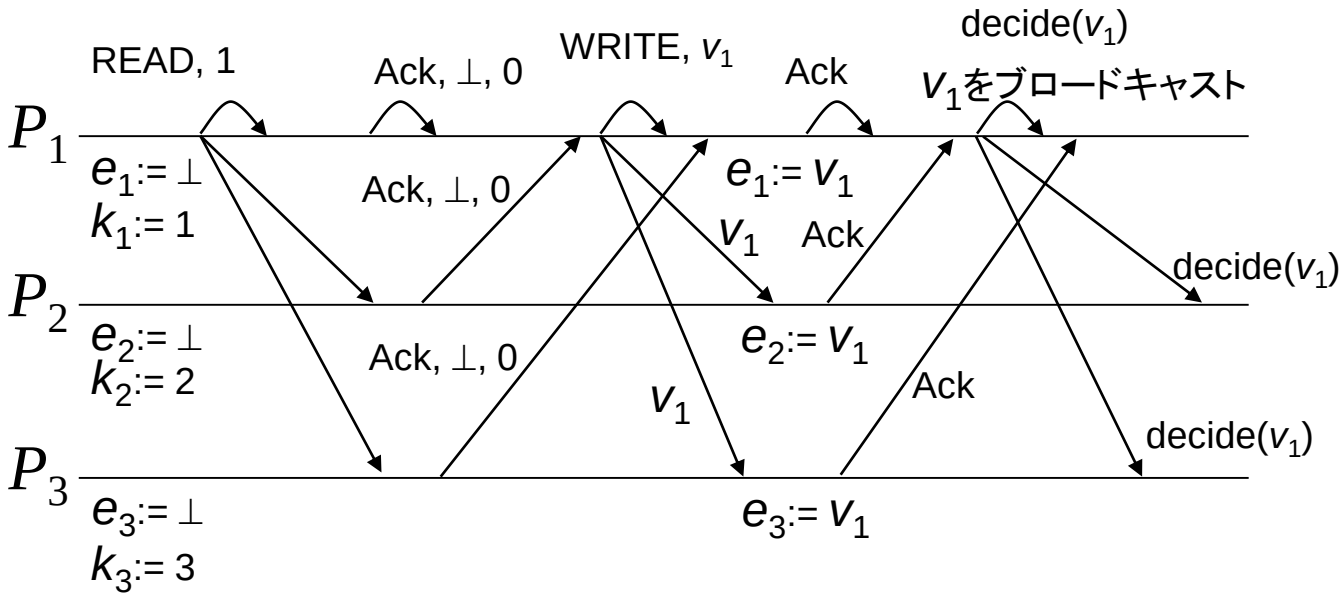
Paxos

- 代表的なアルゴリズム*
  - 半数未満のプロセスの故障に耐えられる
  - それ以上故障しても停止性以外はみたす
  - 停止性は，弱い同期性がなりたてば満たされる
- アルゴリズムの概要
  - Read phase：決定できる値を過半数のプロセスから収集する
  - Write phase: Readで得た値をプロセスにWriteするように要求
    - ◆過半数のプロセスがWriteしたら，その値を全プロセスに決定させる

*L. Lamport, The Part-Time Parliament, ACM Trans. Comp. Sys, 1998.

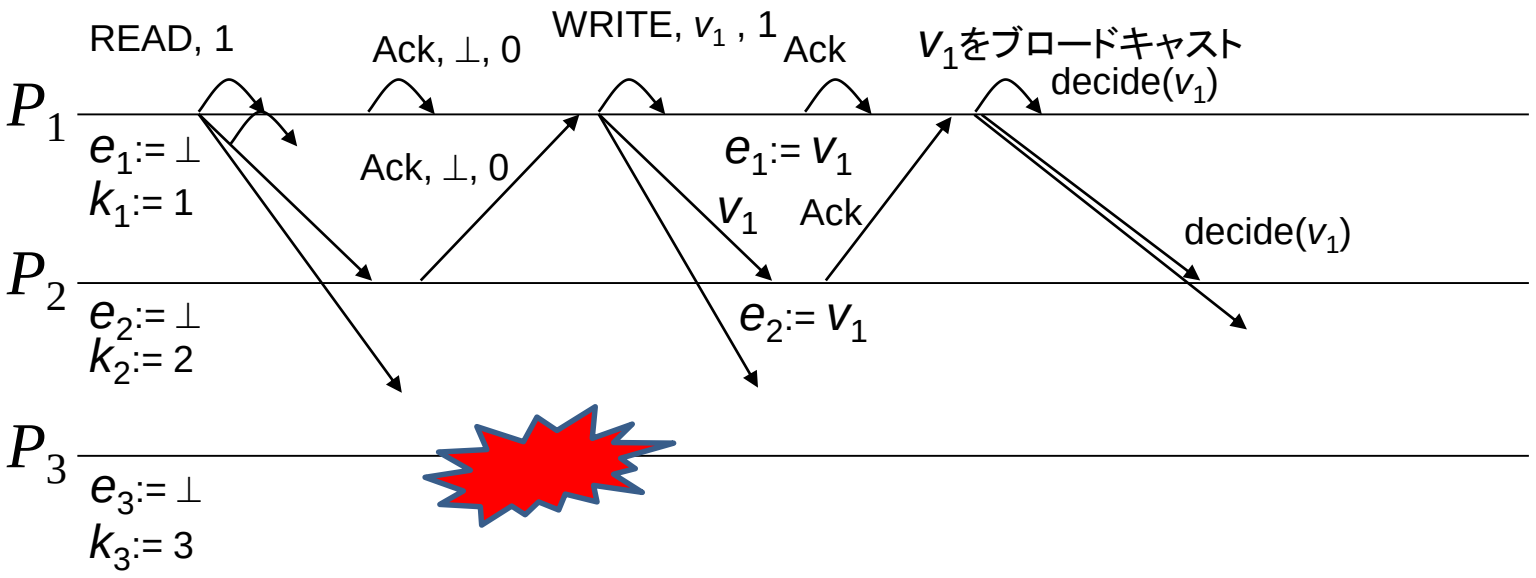
Paxos

- 自分がリーダと思っているプロセスが処理を開始
- ラウンド番号k_iを送り，過半数のプロセスから，Writeされた値e_jがあるか収集
- あればラウンド番号最大の値，なければ，自身の提案値v_iを選択し，Writeを要求
- 過半数からAckを得れば，決定



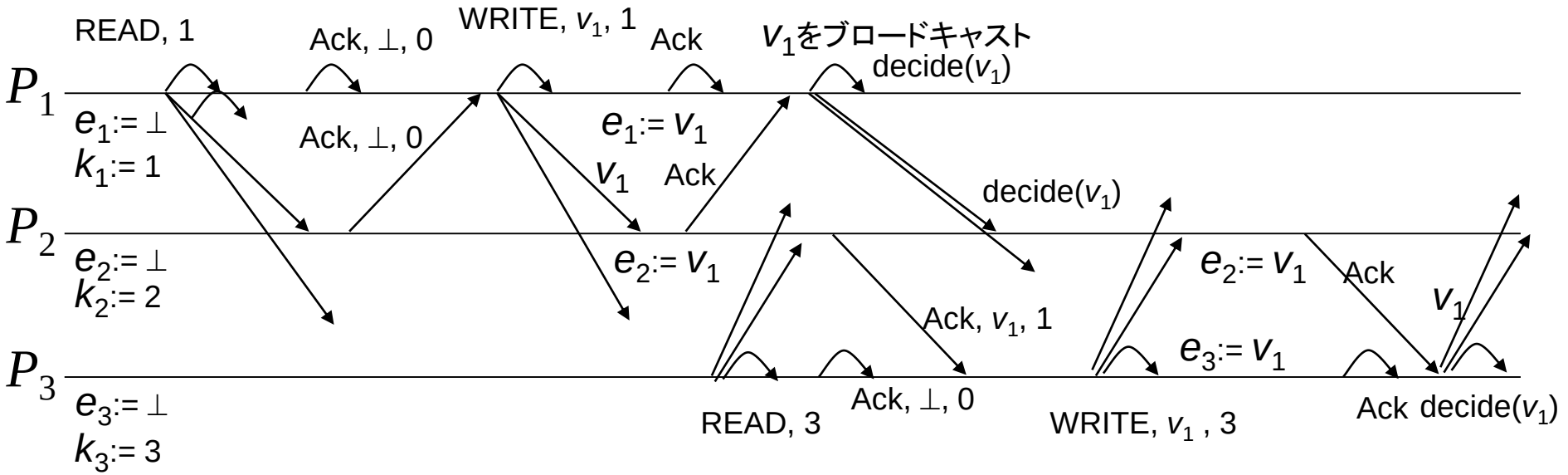
Paxos

- 自分がリーダと思っているプロセスが処理を開始
- ラウンド番号k_iを送り，過半数のプロセスから，Writeされた値e_jがあるか収集
- あればラウンド番号最大の値，なければ，自身の提案値v_iを選択し，Writeを要求
- 過半数からAckを得れば，決定



Paxos

- 自分がリーダと思っているプロセスが処理を開始
- ラウンド番号k_iを送り，過半数のプロセスから，Writeされた値e_jがあるか収集
- あればラウンド番号最大の値，なければ，自身の提案値v_iを選択し，Writeを要求
- 過半数からAckを得れば，決定



合意性が満たされる理由

- 過半数のプロセスからAckを受信したとき, Read, Write処理が成功
  - Readでは, 過半数のプロセスの返信のうち, ラウンド番号(k_i)がもっとも大きい値をWriteする値として選択
  - 値wが決定される
    ⇒過半数のプロセスが, その値wをWriteしている ($e_i = w$)
    ⇒以降のラウンドのReadで, 必ずwが選ばれる
- どのような2つの過半数集合も, かならず共通部分をもつ

Quorum systems

- プロセスの集合の集合で, どの2つの集合(quorum)も共通部分をもつもの
- 例. プロセスP_1, P_2, P_3
  - $\{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}$
  - $\{\{P_1\}\}$
- 例. 過半数集合
  - Paxos

Voting

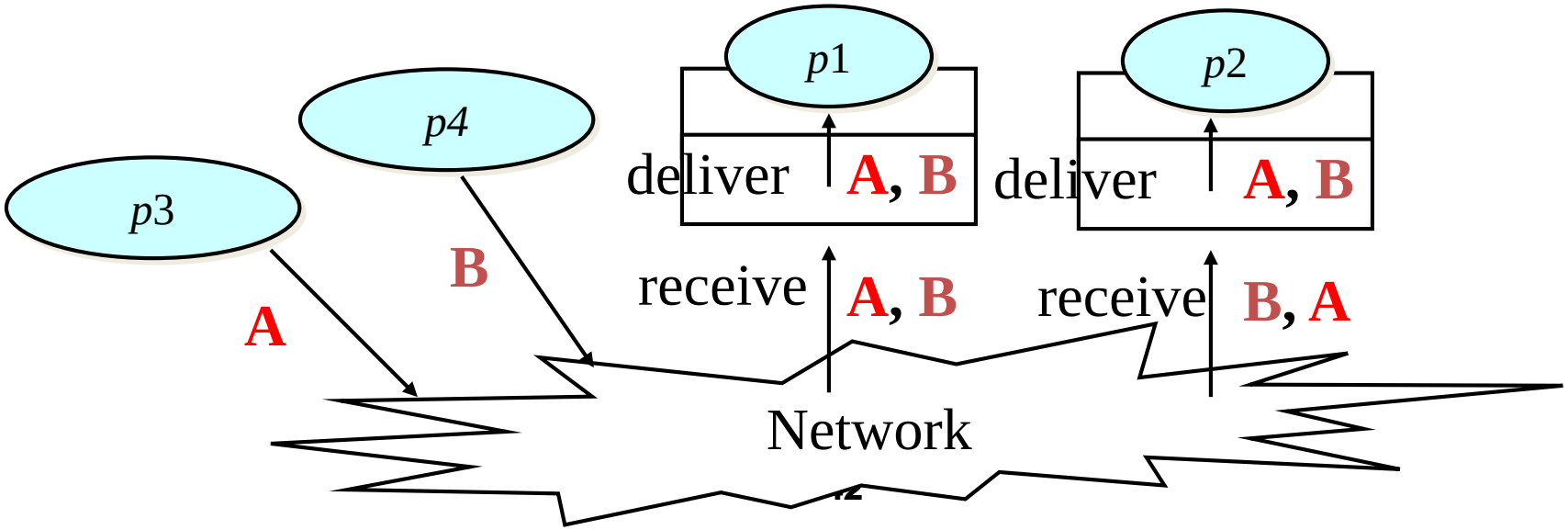
- Quorum systemを形成する1手法
  - 各プロセスP_iに, 票として非負整数v_iをわりあてる
  - Quorum: 過半数の票をもつ極小(minimal)なプロセス集合
- 例. P_1:1, P_2:2, P_3:2
  - 票の総和: 5
  - Quorum
    - $\{P_1, P_2\}$, $\{P_1, P_3\}$, $\{P_2, P_3\}$
- 例. Apache Zookeeper
  - Paxosは使っていないが, 似たアルゴリズムを使用

Quorum systemのアベイラビリティ
Availability of a quorum system

- Availability = 1つ以上Quorumが通信可能な確率
  - システムが進行可能な条件＝ 1つ以上Quorumが通信可能
- 例. Voting
  - 仮定
    - プロセスのアベイラビリティ：P_1: 0.95, P_2:0.92, P_3:0.9
    - 票：P_1:1, P_2:1, P_3:1
    - ネットワークは故障しない
  - Availability
    - 0.95*0.92*0.9 + 0.95*0.92*(1-0.9)
      + 0.95*(1-0.92)*0.9 + (1-0.95)*0.92*0.9

原子ブロードキャスト
atomic broadcast/total order broadcast

- 原子ブロードキャスト
  - どのプロセスもブロードキャストできる
  - ブロードキャストされたメッセージを，すべてのプロセスが同じ順序でdeliver
    - ◆受信しても一旦bufferingすることが必要

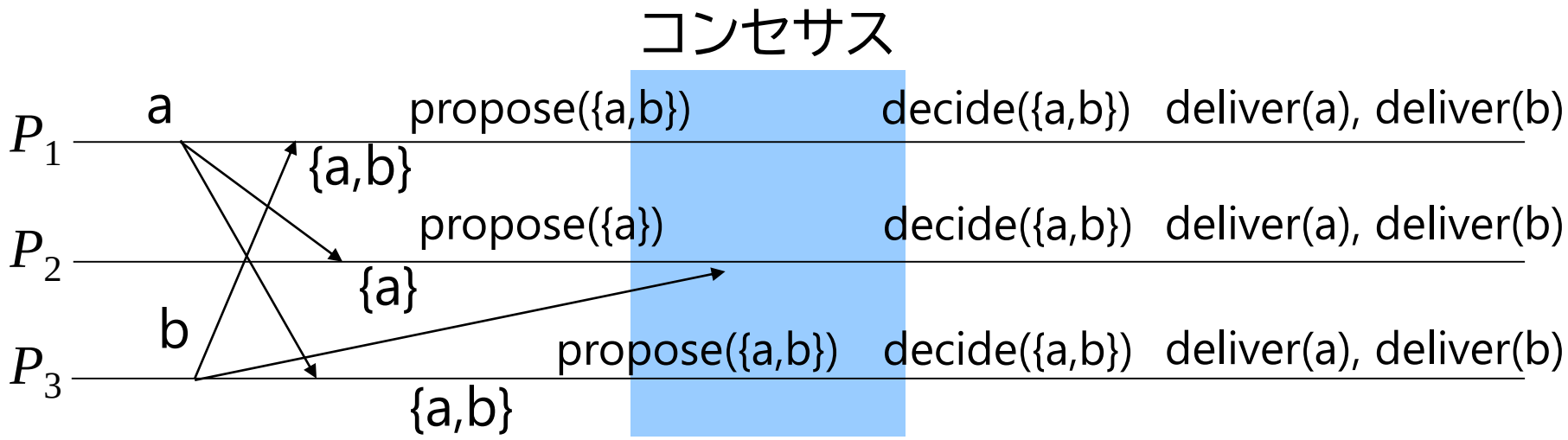


Conditions

- ここではCrash faults を仮定
- 停止性
  - 正常なプロセスPがmをブロードキャスト
    → PはmをdeliverP
- 合意
  - プロセスがmをdeliver
    →すべての正常なプロセスがmをdeliver
- 妥当性
  - プロセスがmをdeliver
    →mはブロードキャストされたもの
- 全順序 (total order)
  - プロセスP, QがmをdeliverP して, Pがmより前にm’をdeliver
    → QもmP より前にm’をdeliver

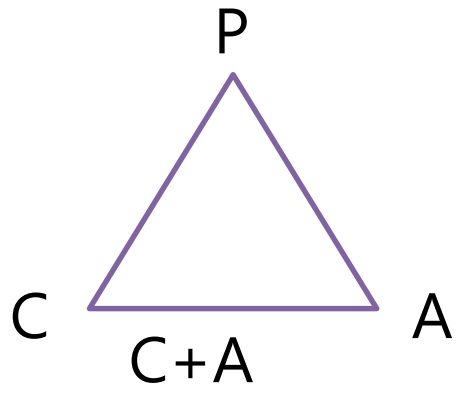
コンセンサスを利用した原子ブロードキャストの実現

- コンセンサスを繰り返し実行
  - 受信したメッセージの集合を提案
  - 決定したメッセージ集合のメッセージを辞書順にdeliver



CAP定理

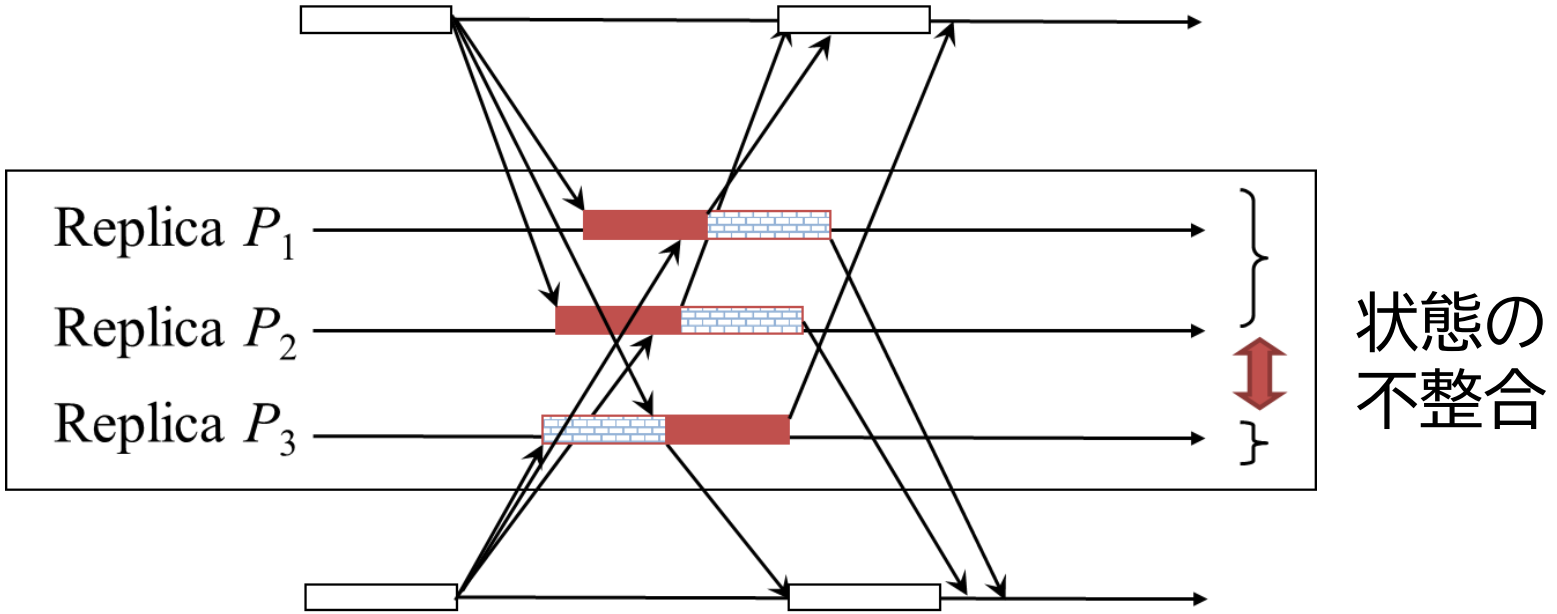
- 非同期分散システムでは, 3条件を同時に満たせない
  - 一貫性（Consistency）
    - ◆全プロセスが同じデータを1つの実体であるようにあつかえる
  - 可用性（Availability）
    - ◆プロセスは, 受信したリクエストに対し, 必ずレスポンスする
  - 分断耐性（Partition tolerance）
    - ◆ネットワークの分断に耐えられる
    - ◆分散システムでは必須
- 実質, C か A かの選択
  - 以降, Cを中心に説明



例. 分散システムでないDB

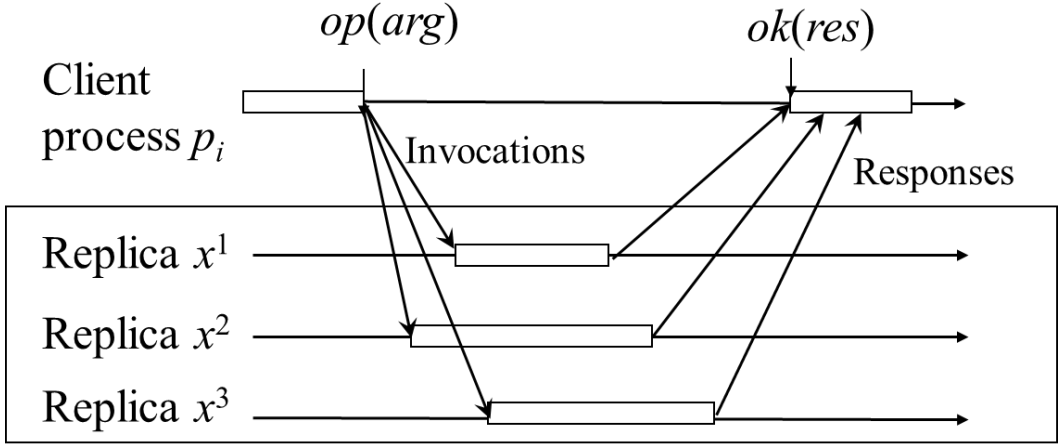
プロセスレプリケーション

- 分散システム上のサービスを実現
- 故障耐性のため，複数のプロセスで同じ命令を実行
- 状態がある場合，簡単には実現できない



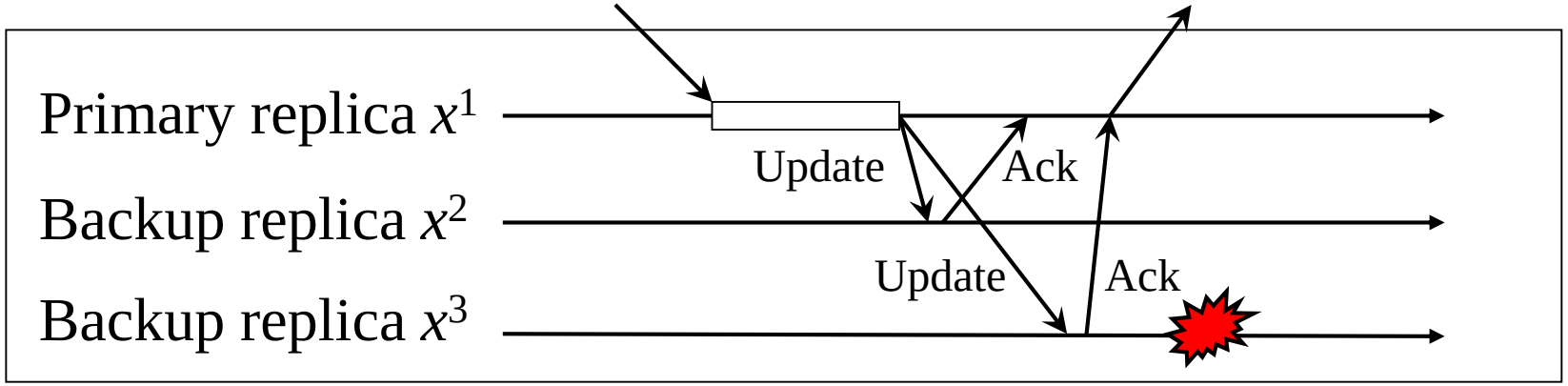
ステートマシーンレプリケーション
Statemachine replication

- プロセスに同じ命令を同じ順番で実行させることで, プロセスの状態を一致させる方法
  → 原子ブロードキャストを利用
  - 実装例：Hyperledger Fabric, Bft-SMaRt
- 命令の実行は決定的 (deterministic)でないといけない
  - 非決定的（nondeterministic）になる原因は？



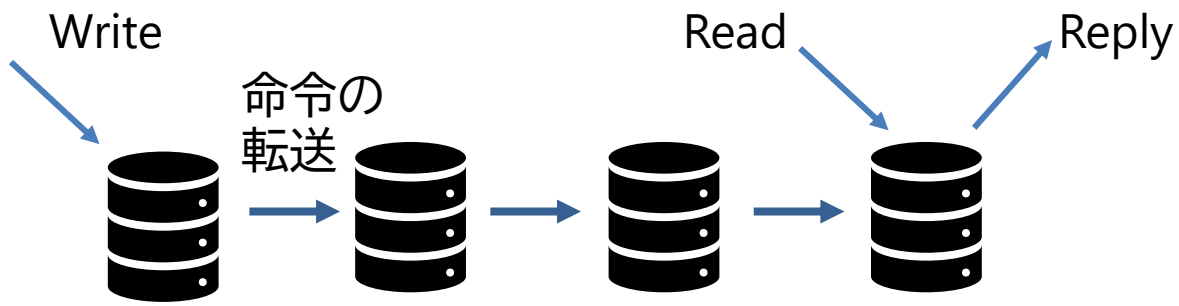
受動的多重化 passive replication

- 1つのプロセスが処理を行い，結果をブロードキャスト
- 実装例：Apache Zookeeper
- 命令の実行は決定的でなくてもよい



Chain replication*

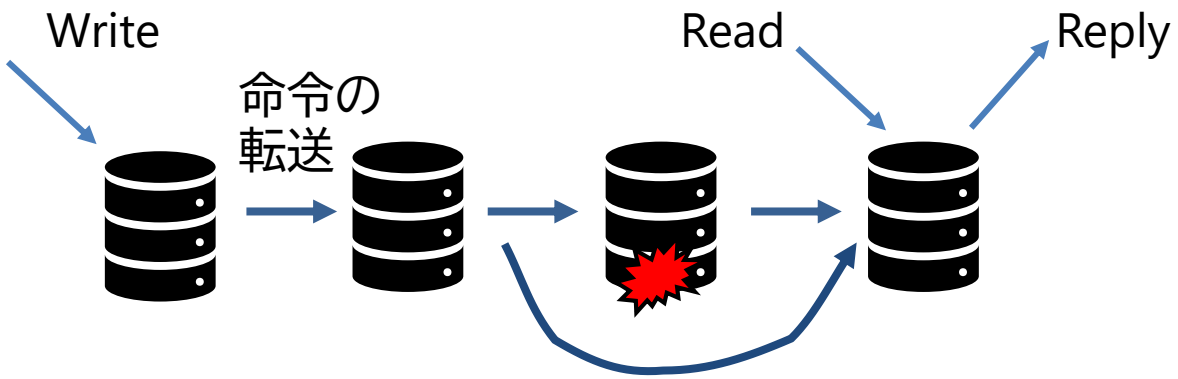
- 実用的なリプリケーション手法
  - 書き込みは, headから実行
    - ◆状態を更新して, リクエストをtail側に転送
  - 読み出しは, tailから実行



*Robbert Van Renesse and Fred B. Schneider, Chain Replication for Supporting High Throughput and Availability, 6th Symposium on Operating Systems Design & Implementation (OSDI 04), 2004

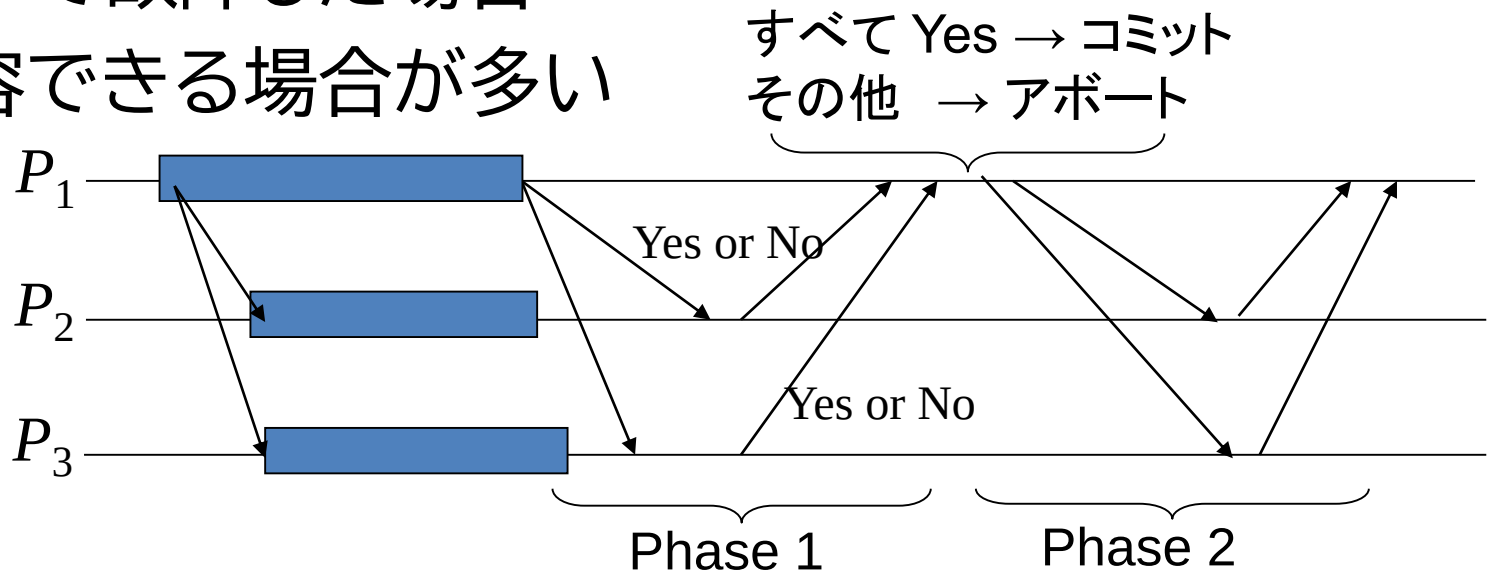
Chain replication

- 故障の場合
  - 故障の検出, 構成の管理は, 別のサービスを利用 (例. Zookeeper)
  - Head, tailの故障：次のプロセスに置き換える
  - 途中のプロセスの故障
    - ◆後ろのプロセスを故障プロセスの前にリンク
    - ◆未実行の更新を実行



分散コミット
distributed commit

- 複数のプロセスで, コミットかアボートを一致して実行
- 2相コミット (2-phase commit)
  - 分散コミットで広く使われている手法
  - ブロックすることがある
    - ◆例. P1が途中で故障した場合
  - 実用的には許容できる場合が多い



結果整合性 eventual consistency

- レプリカ間のデータが, 最後の更新から時間が経てば, いずれ一致
- A(vailability)+P
  - 一貫性を保証しない分, データの可用性が向上
  - ACID vs. BASE (Basically Available, Softstate, Eventually consistent)
- システムの例
  - Cassandra
  - Redis
  - MongoDB