分散システム基礎とクラウドでの活用

第4回：順序づけ

国立情報学研究所
石川　冬樹
f-ishikawa@nii.ac.jp

今回の内容

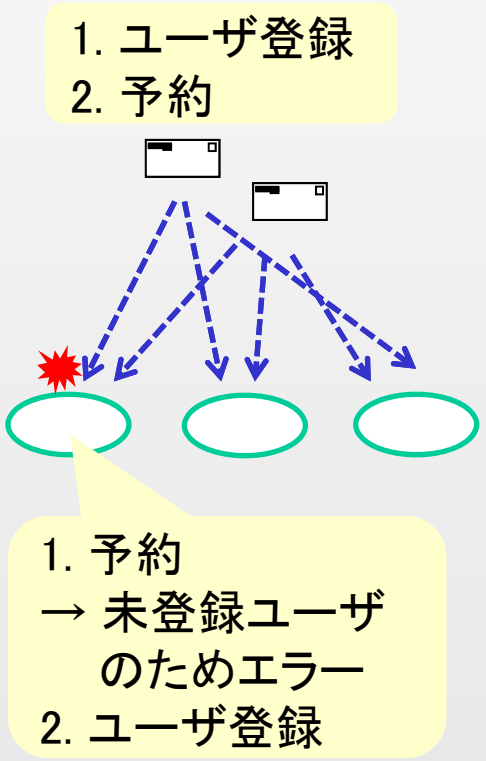
- 順序づけに関する性質とプロトコルを議論する
  - イベントの順序づけのための基本手法である論理クロックを学ぶ
  - 順序つきマルチキャストの性質と実現方法を議論する
  - 複製管理における一貫性の定義を確認する

目次

- 順序と因果関係，一貫性
- 論理クロック
- 順序つきマルチキャスト
- 複製データ管理の一貫性

順序にかかわる不具合

- 複製に対するマルチキャストが様々な順序で到着する可能性があり，状況によってはそれは不具合を引き起こす
  - 右は第1回に挙げた例：
    「意味（因果関係）のある順序で送り出されたが，逆転して到達」

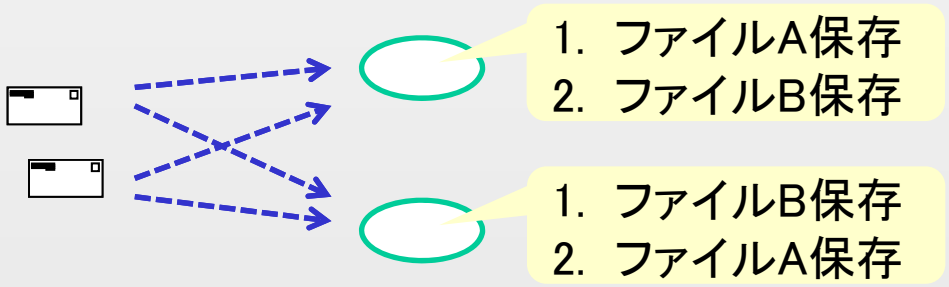


考慮すべき順序は？

どう「順序通り」に届ける？

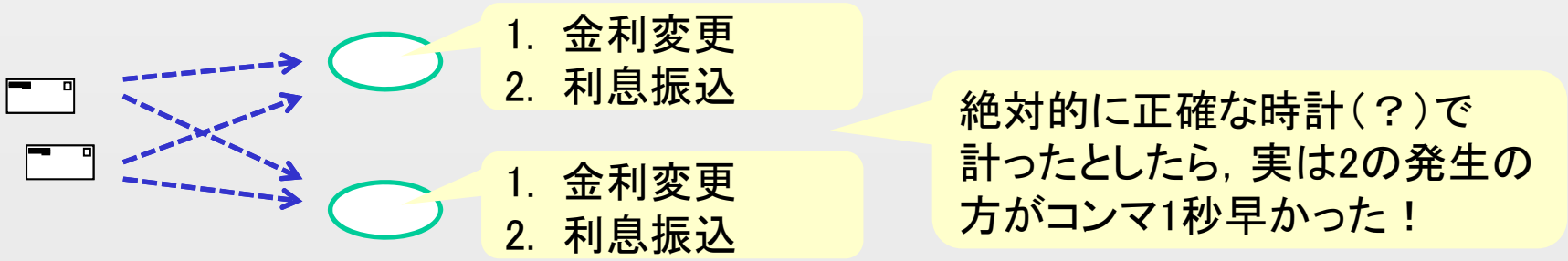
順序と因果関係

- あらゆる順序を考慮したいわけではない
  - 大半のイベントの間には論理的な因果関係や依存性はない
  - それらの前後関係を常にすべて明確にしたり，複数のプロセス間で一意に合意できるようにしたりする必要はない



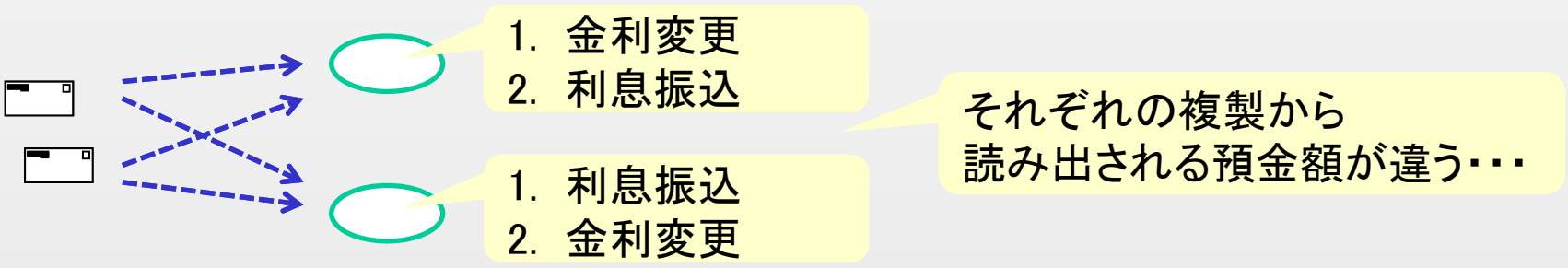
順序と絶対時間

- 絶対的に正しい順序が必要なわけではない
  - 物理的な絶対時間は，ある程度の精度でしか計れず，（神様視点では存在するはずだが）絶対的な前後関係を正確には把握できない
  - アプリケーションユーザにとって意味を持つほどの時間差でなければ，必要がない



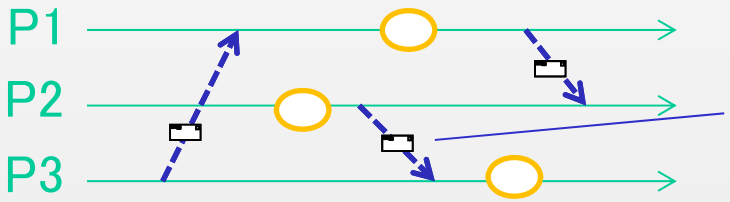
順序と合意

- ただし，一意な順序を合意すべき状況もある
  - 絶対時間での前後関係にかかわらず
  - もしも因果関係がないとしても



補足：順序と因果関係（複製以外での意義）

- 複製管理に限らず，因果関係による順序は重要
  - 例：スナップショット（各プロセスが「いっせいに」実行状態を記録，様々な利用方法あり）



P3のメッセージ受信は記録されているが
P2からの送信は記録されていない

例：バックアップと思うと，再開すると再度配信

例：状態監視やテストログと思うと，ロックを2つのプロセスが持っているような状況もありえるし，そもそも「謎の受信」のデバッグに悪戦苦闘かも

目次

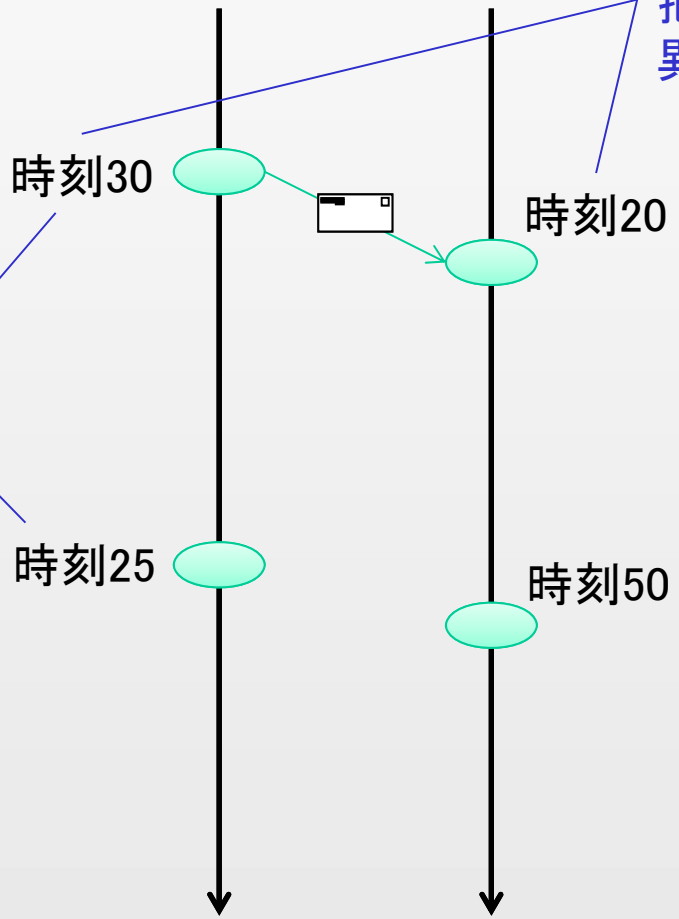
- 順序と因果関係，一貫性
- **論理クロック**
- 順序つきマルチキャスト
- 複製データ管理の一貫性

論理クロック

- 異なるノードが持つ物理クロックの示す時間は，必ずしも厳密，正確に合致しないが，
  - 因果関係のあるイベント間の順序関係を正しく把握できるようにしたい
  - システム全体でイベント間の順序関係を合意できるようにしたい
- イベント間の順序関係を表す値をうまく決める（**論理クロック**）
  - 絶対時間に基づく順序関係と一致する必要はないが，因果関係を反映した値になっていて欲しい

論理クロック：ダメな与え方の例

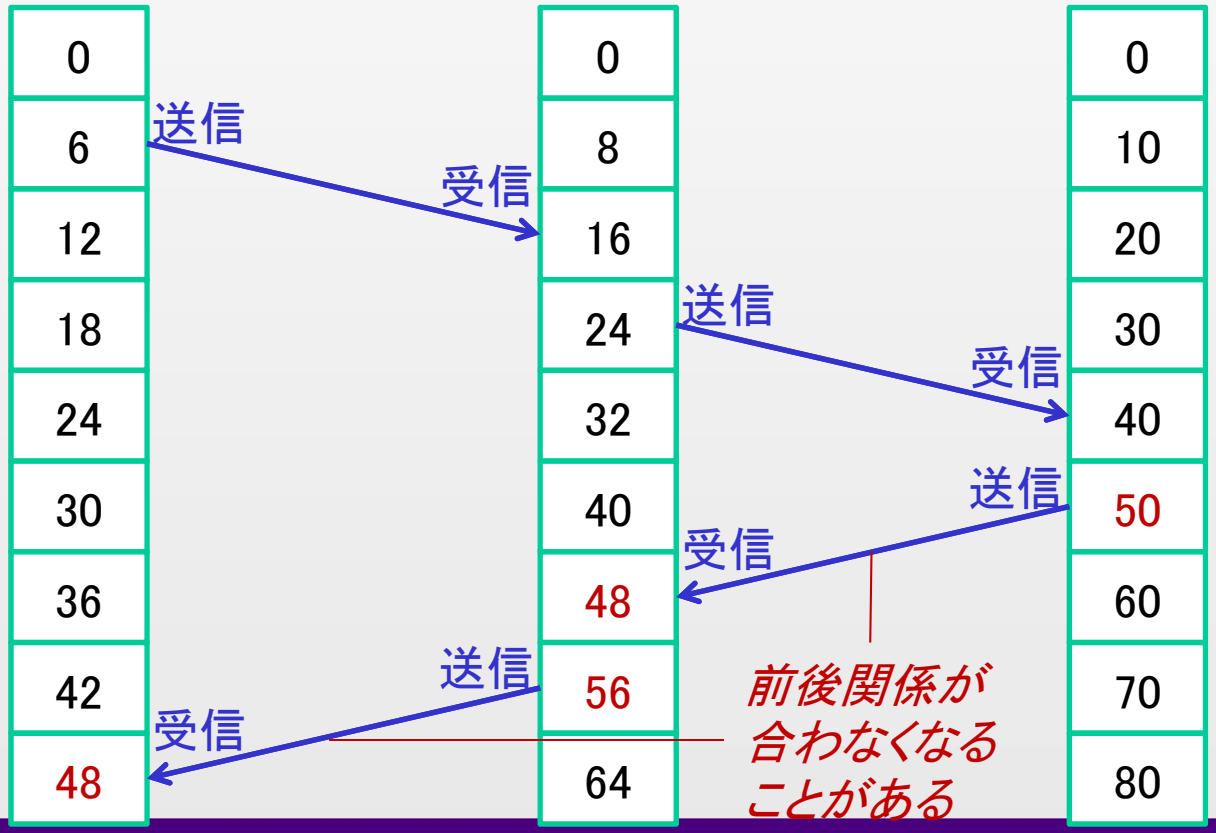
送受信という因果関係から
把握できる前後関係と
異なっている！



同一プロセスの中で
明らかに把握できる
前後関係と
異なっている！

論理クロック：クロック修正

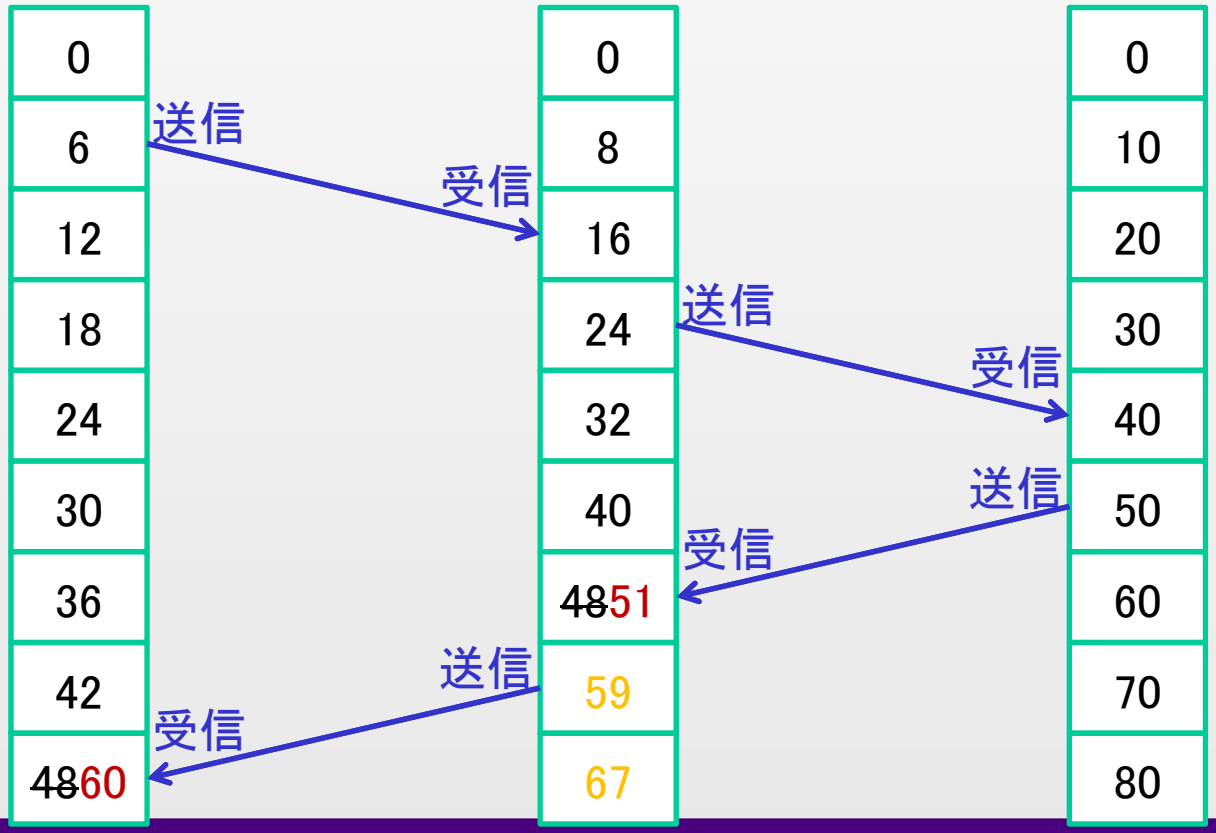
- もし各ノードの物理クロック（互いにずれている）をそのまま使うと・・・



論理クロック：クロック修正

- 論理クロック修正 [Lamport, 1978]
  - 受信時のクロックを必要に応じ調整する

LaTeX開発者！



受信時のクロック値が該当する送信発生時のクロック値以下なら，「送信発生のクロック値＋1」に修正

論理クロック：クロック修正

- 保証できる性質
  - 同じプロセスにおいて，イベントaがbより前に発生するならば，Clock(a) < Clock(b)
  - イベントaおよびbがそれぞれ同じメッセージの送信，受信を表すならば， Clock(a) < Clock(b)
- 追加：各プロセスの識別子を小数点以下に付けると，すべてのイベント間に順序が定まる
  - 例えばプロセス1の40を40.1とする
  - このとき，すべての異なるイベントaおよびbに対して，Clock(a) ≠ Clock(b)

論理クロック：クロック修正

- 保証できない性質
  - Clock(a) < Clock(b) であっても，イベントaがbより早く起きたとは限らない
  - 因果関係による順序を反映しないことがある

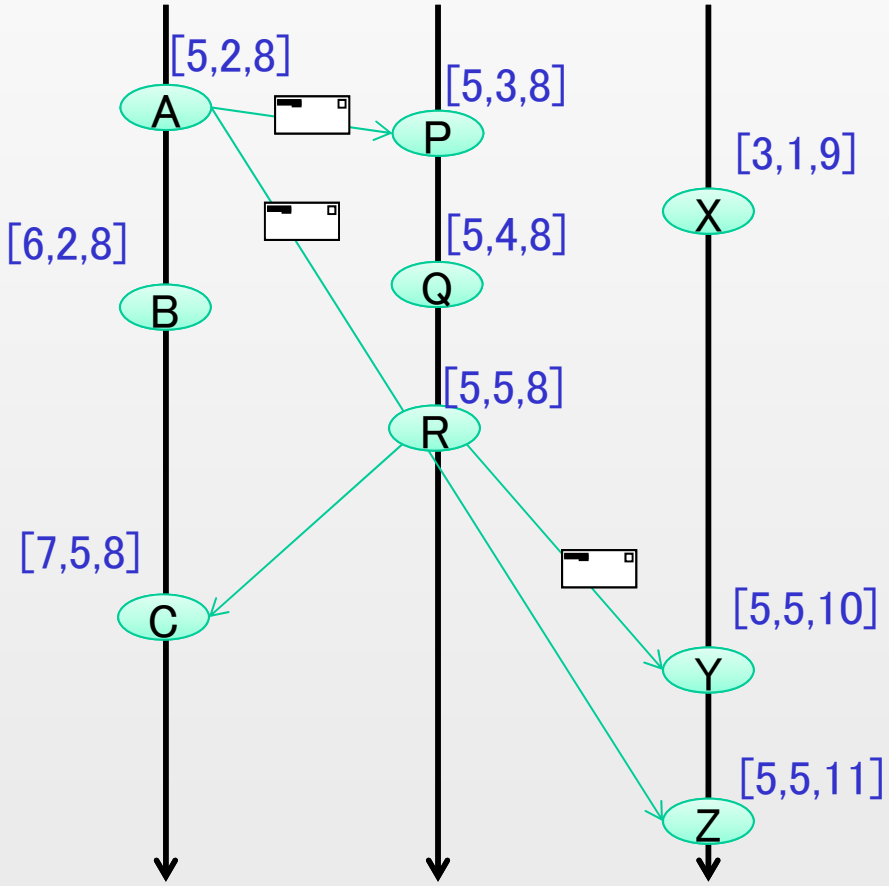
論理クロック：ベクトルタイムスタンプ

- ベクトルタイムスタンプ
  - 「これまでどのプロセスで何個のイベントが発生したのか？」という情報（各プロセスが自分なりに把握しているもの）をタイムスタンプと見なす
  - この情報をメッセージ送信時に添付し，受信者はその情報を更新する

論理クロック：ベクトルタイムスタンプ

- ベクトルタイムスタンプ
  - 各プロセス P_i は，ベクトル V_i を維持し，タイムスタンプとしてメッセージ送信時に添付
    - $V_i[i]$：P_i において今まで発生したイベントの数（イベント発生時にインクリメント）
    - $V_i[j]$：P_j において今まで発生したイベントの数に関するPiの知識
  - 各プロセス P_j は P_i からのメッセージ内のベクトル vs を受信したら，（上記インクリメントに加えて）
    - $V_j[k]$ を $\max\{V_j[k], vs[k]\}$ に更新（自分が知らなかったイベント発生を教えてもらう）

論理クロック：ベクトルタイムスタンプ



v1, v2において,
∀i v1 [i] ≦ v2 [i]
∃i v1 [i] ＜ v2 [i]

⇔ v1はv2より前に起こった

例：
A[5,2,8]はY[5,5,10]より前

C[7,5,8]とY[5,5,10]はどちらが前とも見なさない（並行）

目次

- 順序と因果関係，一貫性
- 論理クロック
- 順序つきマルチキャスト
- 複製データ管理の一貫性

順序つきマルチキャスト

- 今回の最初に挙げたような不具合を避けるために，特定の順序をつけてマルチキャストを実現する方法を考える
  - 一般的にはミドルウェアやフレームワークなどが実装すべき機能
  - 満たすべき順序の定義はいくつかありうる

FIFOマルチキャスト

- 同じプロセスが送ったメッセージは，そのプロセスが送り出した順序で，各プロセスに受信される
  - TCPやそれに類似した仕組みを用いればよい：

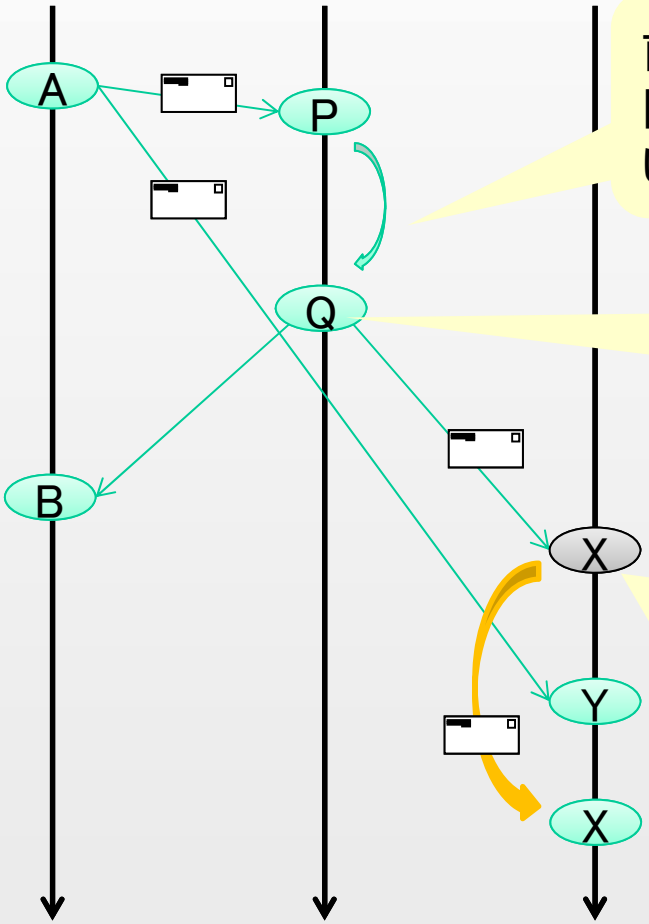
    送信プロセスはシーケンス番号をインクリメントしながら送信し，受信プロセスはそれに基づいてメッセージを並び替えて（届いていない番号が若いメッセージを待って）届ける
  - 送信プロセスがクラッシュしても順番を覚えているようにし，全プロセスに受信されたことを確認するようにする

因果順序マルチキャスト

- 以下のように因果関係に基づく前後関係があるメッセージは，それを満たす順序（causal order）で各プロセスに受信される（結果FIFOにもなる）
  - あるメッセージの送信イベントは，そのメッセージの受信イベントよりも前に起きた，と考える
  - 同一プロセス内で，逐次的な実行の中で順に先に実行されたイベントは，後に実行されたイベントよりも前に起きた，と考える
  - イベントAがイベントBより前に起きた，イベントBがイベントCより前に起きた，というとき，イベントAはイベントCより前に起きた，と考える

因果順序マルチキャスト

リクエスト
マルチキャスト



前後関係を考慮すべきような，
因果関係があるかもしれない
と考える

リクエストに反応した
返信マルチキャスト
（因果関係があるシナリオ例）

AがPより前，PがQより前
だから，AはQより前なので，
Qで送信されたメッセージは
Aで送信されたものよりも
後に受け取るようにする

（受信側が並び替えをすることによる実現方式のイメージ）

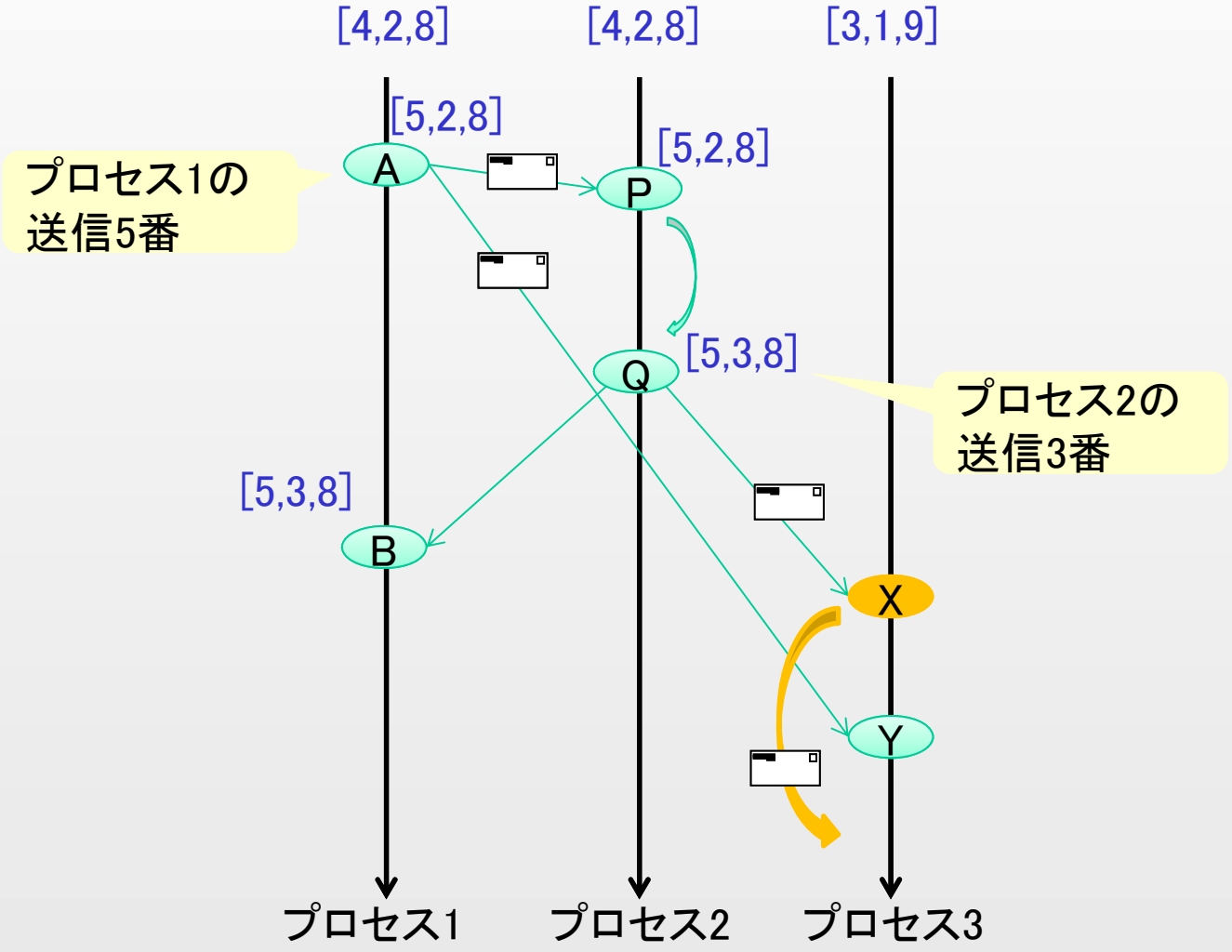
因果順序マルチキャスト

- 論理クロックを用い，「因果的に先に起きたはずのマルチキャスト」を把握すればよい
- ベクトルタイムスタンプの場合，
  - マルチキャスト送信イベントに対してだけ，ベクトル内のカウントインクリメントを行うよう変更
  - あるマルチキャストのメッセージは，それよりも先に起きたはずのマルチキャストをすべて受け取るまで，遅らされる

演習2：課題

- 次スライドに，3つのプロセスの実行状況を示す
  - マルチキャストの発生の様子や，それに付けられたタイムスタンプが記されている
  - 上部には，それ以前より各プロセスが保持していたタイムスタンプが記されている
  - タイムスタンプの管理方法は前スライドの通り
- 因果順序マルチキャストに従うとき，プロセス3にXにて届いたメッセージを実際に配信するために何個のメッセージを待つ必要があるか？

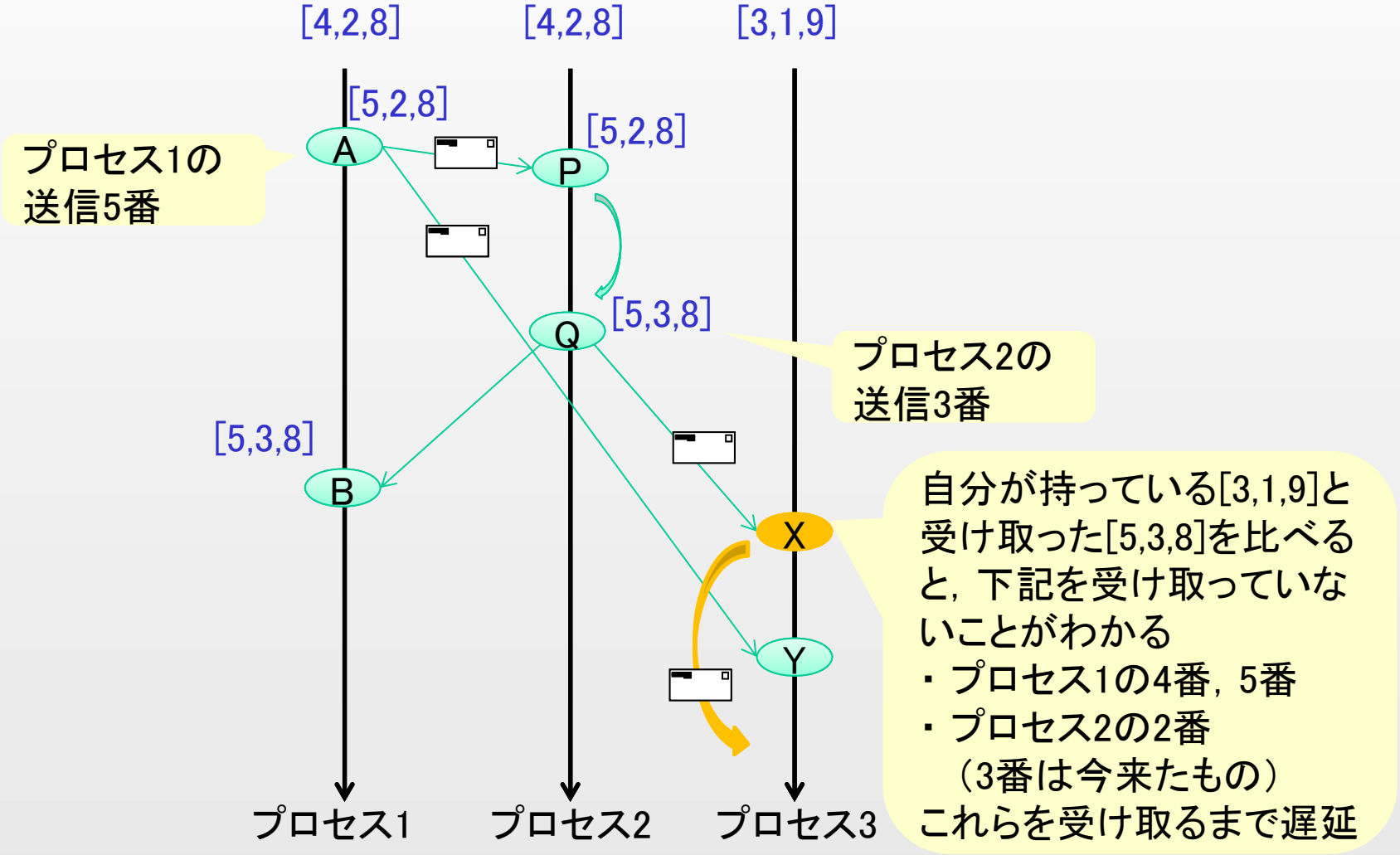
演習2：シナリオ



演習2：課題

- 余裕があれば，さらに，下記の条件を検討せよ
  - プロセス P_j がタイムスタンプ V_j を持つときに，プロセス P_i からタイムスタンプ vs を持つメッセージを受信したとする
  - 因果順序に従うとき，このメッセージを実際に配信してよいのは，V_j と vs の間にどのような条件が成り立つときか？

因果順序マルチキャスト



因果順序マルチキャスト

（正確に書いておくと）

- P_i からのメッセージの P_j による受信は，それに付いているタイムスタンプ ts が下記の条件を満たすまで遅らされる
  - $ts[i] = V_j[i] + 1$
    （該当メッセージは，受信者 P_j が送信者 P_i からすでに受け取ったメッセージの，ちょうど次のもの）
  - $ts[k] \leqq V_j[k]$ （$k \neq i$）
    （送信者 P_i が，該当メッセージの送信以前に受け取っていた他プロセスからのメッセージを，受信者 P_j もすでに受け取っている）

因果順序マルチキャスト

- 限界
  - 同じプロセス内の2つの送信イベント間の順序は常に考慮される
    - アプリケーションロジック上では因果関係がなく，たまたまどちらかが先になっただけかも
    - ミドルウェアやフレームワークでの汎用的な機能
  - 不要な前後関係まで強制するオーバーヘッドが発生し，性能に影響を与える

    （とはいえ，アプリケーションロジックに順序制御を埋め込むか，というトレードオフ）

全順序マルチキャスト

- **全順序**（total-order）：マルチキャストが，各プロセスに同じ順序で配信される
  - 全順序マルチキャスト，全順序FIFOマルチキャスト，全順序因果マルチキャストがありうる
  - 一般的に実現コスト（オーバーヘッドが高い）

※ 因果順序だけであれば，タイムスタンプによるメッセージ増加（ただし圧縮の工夫あり）と，順序調整の待ち時間（ネットワークが安定していれば余り起きない）

全順序マルチキャスト

- 全順序マルチキャストの実現方法
  - 実現例：トークンをプロセス間で順に回し，トークンを持つプロセスが順序を決めて周知
  - 実現例：調整者が，各プロセスでの受信時クロックの値を集め，その最大値を皆で用いる
  - 実現例：各プロセスはメッセージ受信時にackをマルチキャストし，クロック修正されたタイムスタンプの小さい順にキューに入れる．キューの仙頭にあるものに対し，他の全プロセスからackが来たら実際に受信．

アトミックマルチキャスト

- アトミック(原子的): マルチキャストが対象プロセスのうち生存しているものすべてに確かに配送される，またはいずれにも配送されない
  - 前回のGMSに基づくと，メッセージが届かなかった受信者は対象グループから外されるという扱いになるので，基本的に「生存しているものすべてに着く」
  - ただし，送信者がマルチキャストの途中でクラッシュし，一部のプロセスにだけメッセージが着いた場合，他のプロセスに届ける必要がある

アトミックマルチキャスト

- 実現方法の例
  - マルチキャストを受け取ったプロセスは，必要ならそれを自分が転送できるように保管しておく
  - 送信者は，TCPなどの到達確認を用いる場合など，送信が成功したら，各ノードにその旨通知し，保管の必要性がないことを伝える（これは次のマルチキャストに織り込んでもよい）
  - 送信者がグループから脱落した場合（GMSが把握），各受信プロセスは何かの順に従い，送信者の代わりにマルチキャストと上記の成功通知

耐久性のあるアトミックマルチキャスト

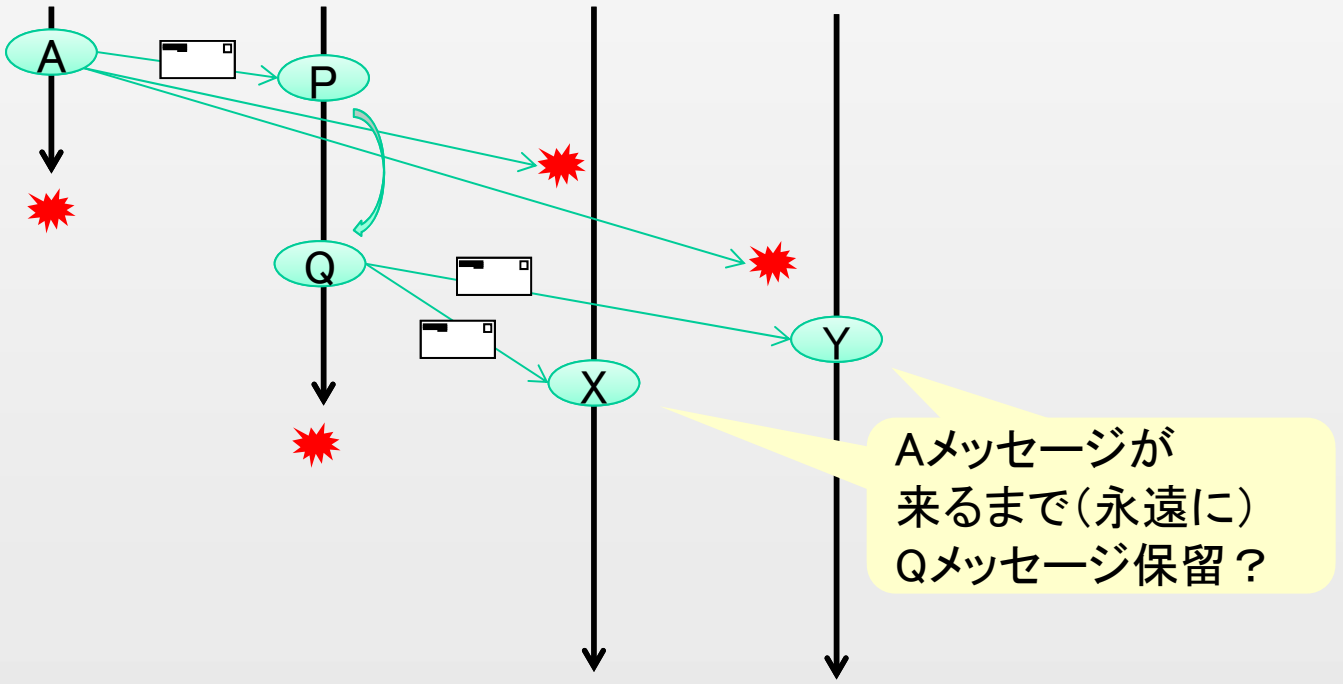
- 「配送される」だけでは，直後にクラッシュして消えてしまうかもしれない
- トランザクション（分散コミット）同様に，耐久性がある（durable）ように，メッセージの効果が生きているプロセスに反映されるようにしたい
- ➡ 2PC同様の，送信（準備），確定という手順を経る（コストが大きい）

障害の考慮：送信者の脱落

- 送信者がマルチキャストの途中でクラッシュする状況：メッセージを受け取った一部の受信者が再送できる
  - メッセージが行き渡る前に，送信者の脱落がGMSにより共有されてしまうかもしれない
  - タスクの分配などの場合，一部のタスクが実行されないという不整合が発生する（送信者以外が落ちているなら，送信者が確認，調整できるが・・・）
  - GMSによる脱落通知を止めて該当プロセスからのメッセージの配信を確定させる，宛先の変化を想定する，といった対応を検討する必要がある

障害の考慮：送信者の脱落

- 因果順序マルチキャストの場合，脱落した送信者によるメッセージを無視すべき状況がありうる



目次

- 順序と因果関係，一貫性
- 論理クロック
- 順序つきマルチキャスト
- **複製データ管理の一貫性**

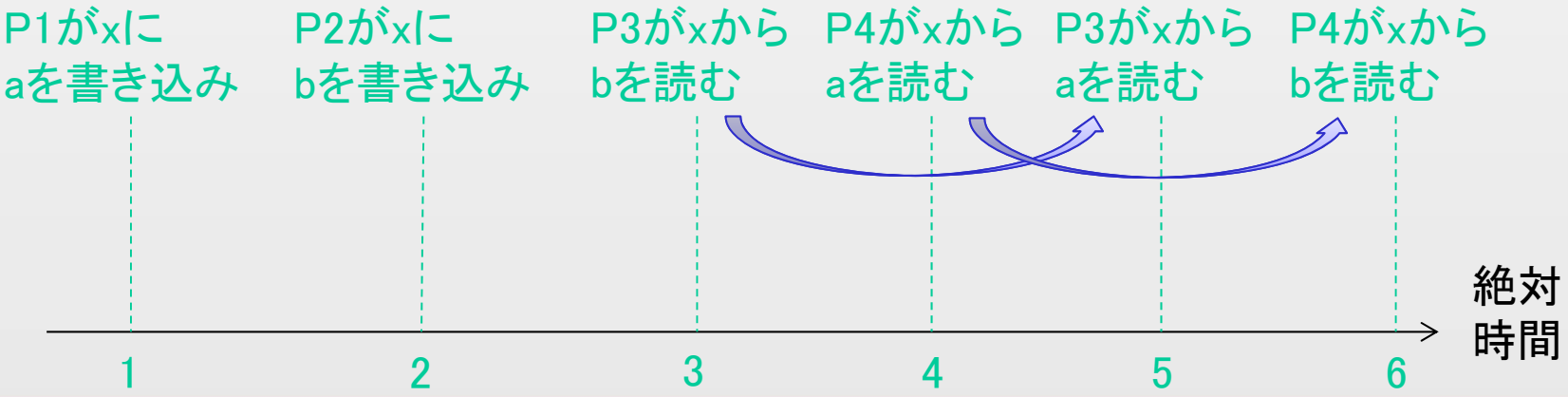
データの複製における一貫性

- 今度はアプリケーション側の視点から，データの複製に対する操作を考えてみる
  - 複製管理機能を提供するミドルウェアやフレームワークなどでは，データの読み取りや書き込みのAPIをアプリケーションに提供
    - 裏では，マルチキャストなどを用い，複製間で書き込み内容が共有されるなどの制御が行われる
  - アプリケーションにとって，読み書きしたときにどう見えるのか？

    （保証される「一貫性」は？）

データの複製における一貫性

- 順序制御など全く行わない場合の例
  - 書き込みは，（順序制御なしの）マルチキャストで複製に反映
  - 読み込みは，複製いずれかに対して行われる
  - 複製によって，書き込み履歴が異なり，プロセスによってどの履歴を見るかが異なるかも



一貫性モデル

- 一貫性モデル
  - データを管理するデータストア(ミドルウェアなど)と，アプリケーションプロセスとの間の約束事
    - プロセスが守るべきルール
    - データストアが保証する性質
  - 例：厳密な一貫性(strict consistency)

    「データに対する読み取りは，必ずそのデータの絶対時間で最も新しい書き込みの結果を返す」
    - 分散システムでは実装できない
      (例：絶対時間1ナノ秒の差で2つの更新が別の複製に行われた場合，新しい値を必ず残せる？)

一貫性モデル

- 一貫性モデルの実現
  - より強い性質を保証するような一貫性は，実現が高コストになる
  - 先のマルチキャスト同様，様々な種類の一貫性モデルが定義されている

一貫性：順序一貫性

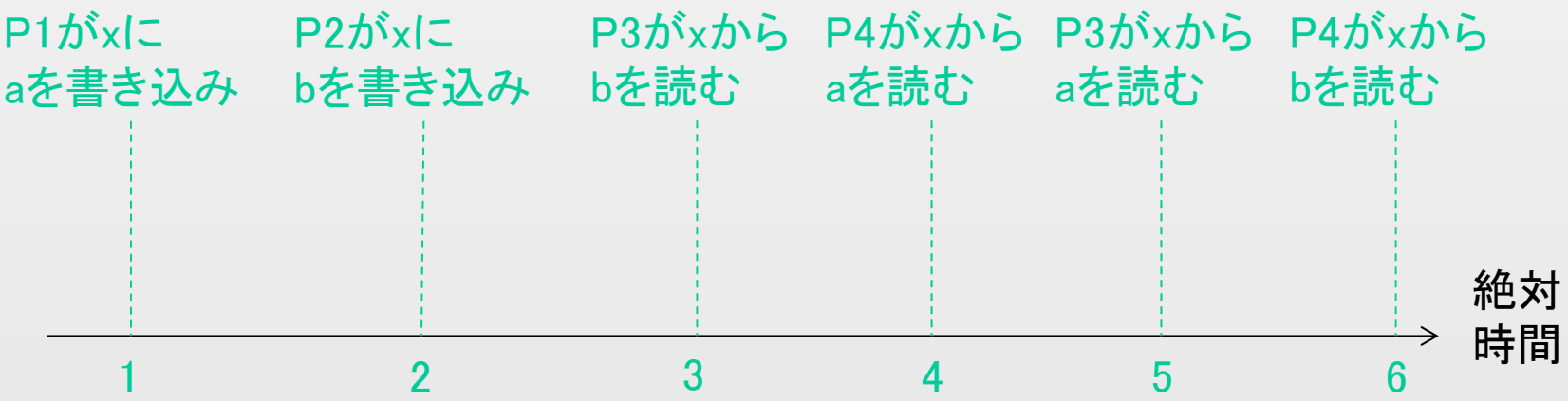
- 順序一貫性（sequential consistency）
  - 「どのような操作の結果も，すべてのプロセスによる読み書き操作があたかもある直線的な順序で行われたとしたときの結果と同一となる」
  - 「各プロセスの操作は，そのプログラムによって指定された順序で現れる」

一貫性：順序一貫性

- 順序一貫性（sequential consistency）

順序一貫ではないデータストアの例
- P3からは，2が1の前に起きたように見えている
- P4からは，1が2の前に起きたように見えている

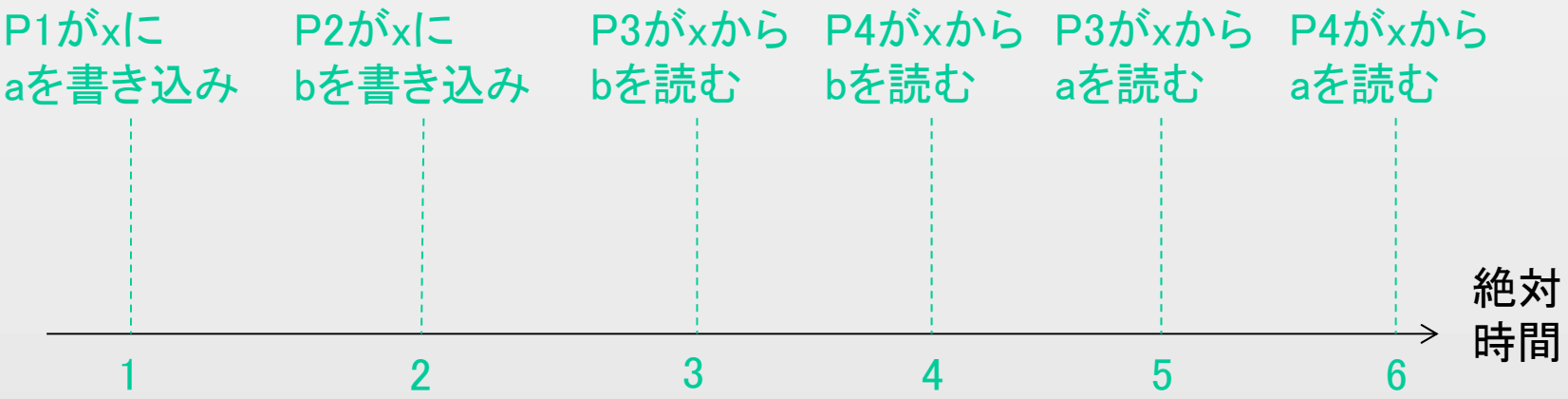
この時点で「ただ1つの直線的な順序がある」ようにはできていない



一貫性：順序一貫性

- 順序一貫性（sequential consistency）

（示されている範囲では）
順序一貫であるデータストアの例
前スライドのような不整合はなく，
（可能性の1つとして）「2, 3, 4, 1, 5, 6」など，
1つの直線的な順序で起きたと見なせる



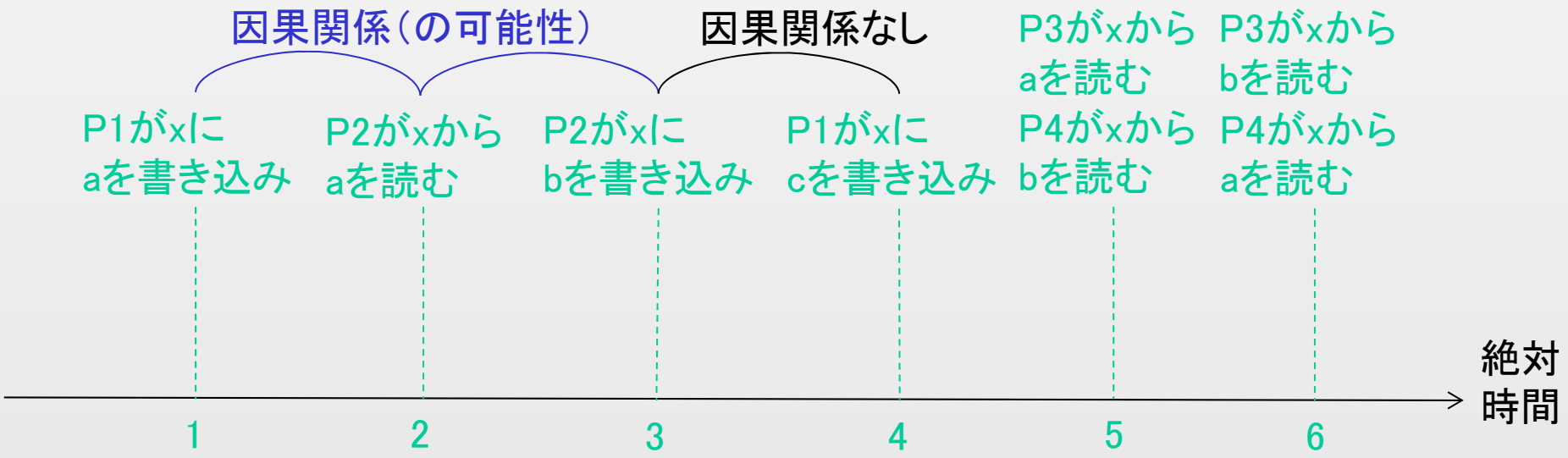
一貫性：因果一貫性

- 因果一貫性（causal consistency）
  - 「因果的に関連している可能性のある書き込みは，すべてのプロセスによって同じ順序で観測される」

一貫性：因果一貫性

- 因果一貫性（causal consistency）

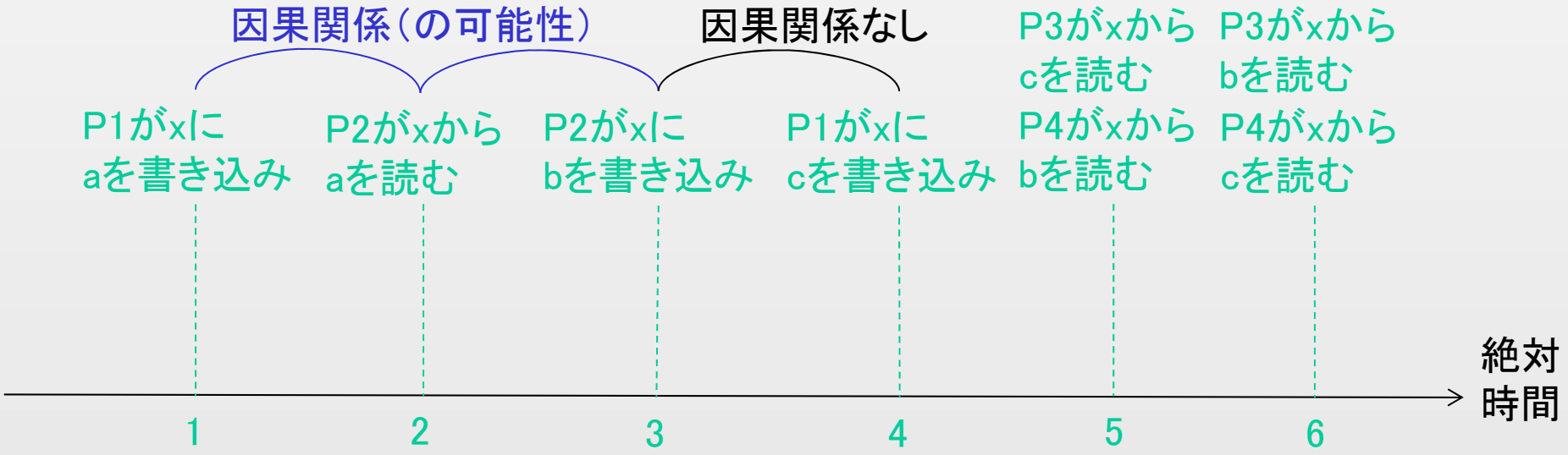
因果一貫でないデータストアの例



一貫性：因果一貫性

- 因果一貫性（causal consistency）

因果一貫である（順序一貫でない）データストアの例
（1と3の順序は共有・合意されるが，3と4の順序はされない）



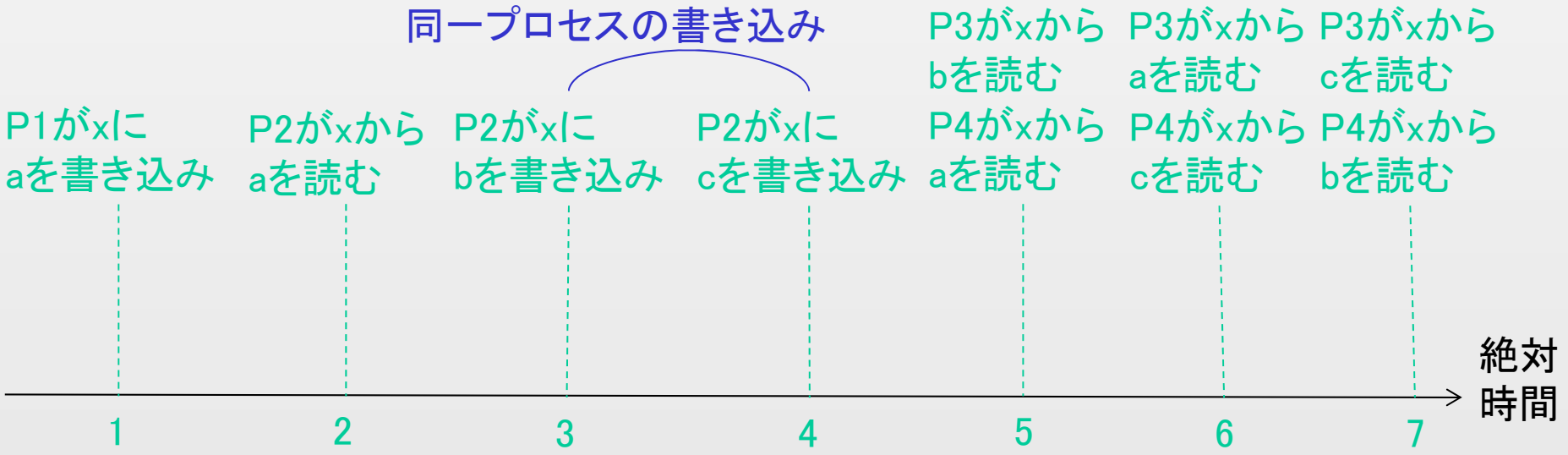
一貫性モデル：FIFO一貫性

- FIFO一貫性（FIFO consistency）
  - 「単一のプロセスによってなされた書き込みは，すべてのプロセスによって同じ順序で観察される」

一貫性モデル：FIFO一貫性

- FIFO一貫性（FIFO consistency）

FIFO一貫でないデータストアの例
（3と4の順序が共有・合意されていない）



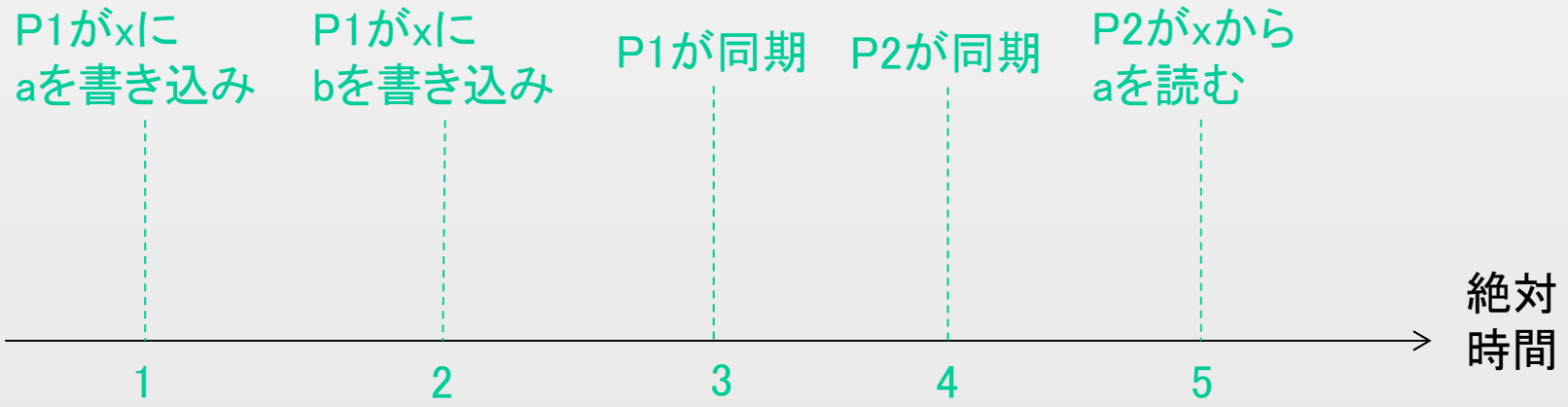
一貫性モデル：弱一貫性

- 弱一貫性（weak consistency）
  - 「同期変数へのアクセスは順序一貫である」
    - タイミング合わせを行う手段を用意する
  - 「すべての先行の書き込みがすべての場所で完了するまで，同期変数への操作は許容されない」
    - 書き込み後に同期操作を行えば，進行中・部分的な書き込み，書き込み結果の伝搬を完了できる
  - 「すべての先行の同期変数へのオペレーションが完了するまで，データへの操作は許容されない」
    - 読み込み前に同期操作を行えば，最新の値を読み取ることができる

一貫性モデル：弱一貫性

- 弱一貫性（weak consistency）

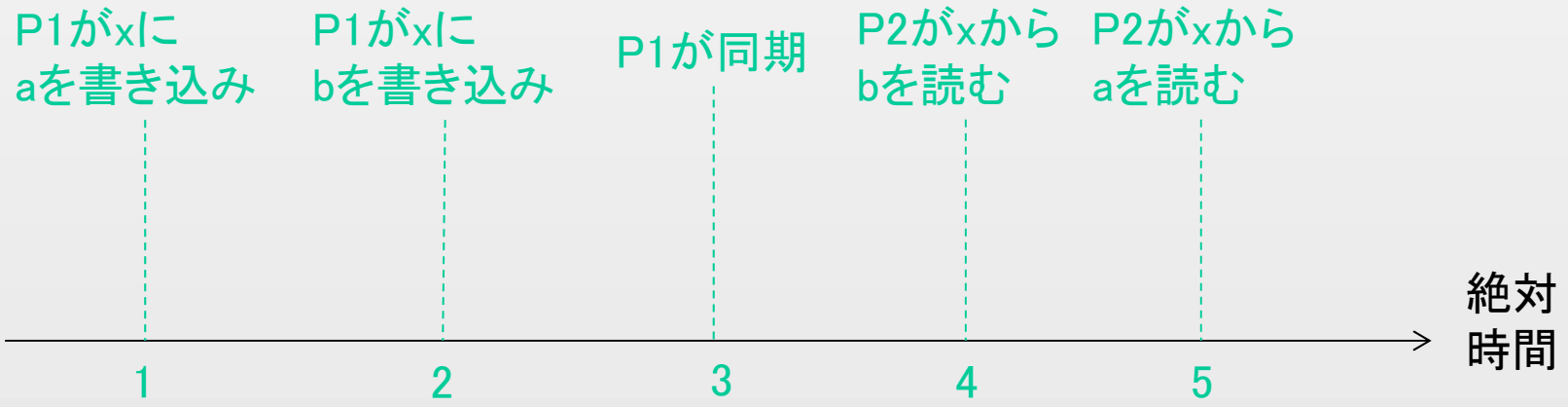
弱一貫性のないデータストアの例
（P2は同期したので，その読み込むことは最新のはずである）



一貫性モデル：弱一貫性

- 弱一貫性（weak consistency）

弱一貫性のあるデータストアの例
（P2はまだ同期していないので，その読み込むことは何も保証されない
　プロセス側にもプログラミング上のルールがある）



一貫性モデル：イベンチュアル一貫性

- イベンチュアル一貫性（eventual consistency）
  - 「更新はすべてのコピーに伝搬する」（更新がなければすべての複製は同一に収束していく）
  - 特別な性質を持つデータストアが対象
    - 書き込み同士の衝突はない
    - 読み込みが必ずしも最新でなくてもかまわない

  （例：Webサイト，DNS）

注：クライアントがアクセスするコピーが変わると，古い情報に変わるといったことがありうる

一貫性モデル：クライアント中心一貫性

- データ中心一貫性： データストアにおけるシステム全体での一貫性
  - これまで述べたもの
- クライアント中心一貫性：
  - 先の移動するユーザにとっての一貫性の例
  - 例： モノトニック読み取り

    「あるプロセスがデータ項目を読み取ったとき，同じプロセスによる同じデータ項目への読み取りは，同じ値かより新しい値を返答する」

今回のまとめ

- イベント発生の順序，特に因果関係によるものの制御や合意は，アプリケーションが整合性ある形で実行されるために非常に重要である
  - その制御や合意のための実現手法は多く提案されており，利用されている
  - 実現する性質とオーバーヘッドの間にトレードオフがあるため，アプリケーションの整合性担保のためにも，実現すべき性質をよく議論すべきである
- 次回：特にクラウドの利用や構築において，これまで学んだ知識・技術の活用を議論する